RENESAS RA MCU 基础知识

Richard Oed

2023.01

免责声明：

本手册仅供参考之用，不保证正确性和完整性。手册内容并不适合用作设计参考指南，对于因使用本手册而导致的任何后果不承担任何责任。

目录

前言

在当今的联网设备环境中，安全性是一个无所不在的问题。每个联网的应用都是一个攻击点，潜在的黑客可能利用潜在漏洞获得对网络的访问权限，从而获取有保护价值的数据。

在快速发展的物联网领域，嵌入式安全具有额外的意义，因为这一领域需要经济高效、可扩展、节能且易于配置的安全解决方案。因此，即使没有为此留出可用的开发资源或相应的开发时间，当今的嵌入式应用制造商也必须认真对待安全性这一主题。

因此，在时间压力下被迫首次处理安全相关问题的工程师人数在不断增加。这样做的原因有很多，例如新的立法、竞争压力和知识产权保护等。对于其中许多公司而言，首次接触安全相关问题可能会极具挑战性，因为安全性涉及到许多复杂的方面，包含硬件和软件组件，而且必须从整体上考量。这意味着安全性不能在开发过程结束时才增加，而是必须从一开始就作为系统方法的一部分。通常而言，很多公司缺乏必要的能力，另外，实现安全功能会给其产品增加额外的成本。事实上，一些公司正承受着这种实际压力。此外，我们还仔细评估了客户在市场和技术趋势方面可能面临的其他挑战。

瑞萨的开发部门已经将帮助客户作为自己的一项使命。我们的重点是开发经济高效、可扩展且节能的安全解决方案，以便为未来嵌入式系统的安全需求提供支持。

我们满怀热情地为客户提供最优质的服务，瑞萨 RA 产品家族即是我们努力的成果。RA 产品家族基于 Arm Cortex-M 内核以及瑞萨及其合作伙伴提供的完整生态系统，可提供业界领先的物联网安全性和软件灵活性。这样一来，我们的客户便能更快地实现其包括安全性在内的各类应用目标。

我们树立了高度关注细节的态度，致力于简化客户的开发工作。32 位 RA MCU 产品家族就是有力证明。

我们希望本手册可以帮助您使用 RA 产品家族轻松完成下一个项目。

Bernd Westhoff
高级经理 - RA MCU 市场营销部
Renesas Electronics

1 瑞萨 RA 产品家族简介

RA 产品家族单片机 (MCU) 于 2019 年 10 月推出，丰富了瑞萨的 32 位 MCU 系列产品。在此之前，瑞萨拥有基于 Arm® Cortex®-M 内核的 Renesas Synergy™ 平台，该平台将 MCU 与商业级、有品质保证的软件和开发工具融于一体。瑞萨的另一条产品线 Renesas eXtreme (RX) 产品家族则采用自有的 RX 内核，具有业界领先的 32 位 CoreMark®/MHz 性能以及大容量代码闪存和 SRAM。瑞萨 RA 家族单片机，将 Synergy 平台与 RX 产品的优势有机融合，使该产品能够满足客户的不同需求并为助力客户创造价值。全新 RA 产品家族包括：RA2 系列，适用于低功耗应用；RA4 系列，适用于需要低功耗、高性能和高安全性的设备；RA6 系列，具有卓越的连接性能和安全性能；RA8 系列，可以为采用人机界面、连接、安全和模拟功能的应用提供出色性能。

图 0-1：RA 产品家族补充了 Renesas 的 32 位单片机产品线

最初发布的 RA 产品家族器件就已经采用了基于硬件的安全功能，从 AES 加速、真随机数发生器 (TRNG) 到单片机内部隔离的全集成加密子系统。瑞萨备受青睐的安全加密引擎 (SCE) 针对 RA 产品家族进行了增强，继承了获得美国国家标准技术研究院 (NIST) 的加密算法验证计划 (CAVP) 认证的对称和非对称加密解密算法、哈希函数和先进的密钥处理功能，包括密钥生成和 MCU 硬件相关的密钥封装。如果未遵循正确的访问协议，访问管理电路就会关闭加密引擎，SCE 专用 RAM 可确保明文密钥绝不会暴露给任何 CPU 或外围总线。此外，多个系列产品还采用了 Arm 的 v8-M TrustZone®。

整个产品家族均已通过 PSA Certified® 1 级认证，还可提供篡改检测功能，并增强了抵御侧信道攻击的能力。这些功能使开发人员能够提高用于工业和楼宇自动化、计量、医疗保健和家用电器应用的高性能物联网 (IoT) 端点和边缘设备的安全性。

得益于 RA 产品家族的功能和引脚兼容性以及其外设 IP 模块在不同系列中的通用性，可轻松跨越不同系列进行转换。所有关键部件、芯片、工具和软件均经过优化，能够协同工作，既可以在必要时创建新模块，也可以重复使用处于前沿水平的成熟 IP 模块。

对于开发人员来说，这可确保他们能够顺利开始硬件和软件开发工作，而不必担心工作流程的各个不同部分之间会出现不兼容的情况。

RA 产品家族的工具生态系统可为工程师提供支持，这套系统提供了基于 Eclipse 的 e² studio 集成开发环境 (IDE)、编译器、片上调试器、评估板、设计文件、原理图、印刷电路板 (PCB) 布线和物料清单 (BOM)。瑞萨开发了易于使用的灵活配置软件包 (FSP)，可提供一种开放式架构，允许设计人员重用其原有代码，并将这些代码与瑞萨的软件示例相结合。此外，瑞萨还构建了一个全面的 RA 合作伙伴生态系统，以提供开箱即用的其他软件和硬件构建模块。

所有这些举措都极大地减轻开发人员的负担，让他们无需在基本任务上花费多精力，从而帮助他们专注于要完成的主要任务：创建增值应用。

1.1 当今嵌入式系统设计的挑战

从麻省理工学院 (M.I.T.) 于 20 世纪 60 年代初开发的阿波罗导航计算机，到如今用于物联网 (IoT) 的高端汽车应用或全连接设备，在过去的数十年里，嵌入式系统发生了翻天覆地的变化。在本世纪初之前，这些系统只采用一些非常基本的接口，如用于输入的按钮，或者用于输出的字符 LED 甚至辉光管，其软件仅包含一个函数，大多数通过 `main()` 函数内部的一个简单循环实现，通过中断来处理有限的任务。具有数 MIPS（每秒百万条指令）处理能力、几 KB 内存和基本串行通信接口的单片机 (MCU) 已经足以满足此类应用的需求。

然而，如今的嵌入式系统高度互联，需要各种各样的接口，如以太网、无线或图形显示，所有这些接口都需要经过配置和处理，同时它们不仅可以彼此交换数据和消息，而且还可与外部环境交互，构成完整的应用。这可能需要时钟速度至少为 60 MHz 的 MCU、几 MB 的闪存以及可能数十 KB 的片上 SRAM。

实时操作系统 (RTOS) 不仅非常实用，而且有时必不可少，因为需要对不同的线程进行优先级排序和并发执行。随着对连接性、功能丰富的人机界面和安全性的需求不断提升，这类系统不再以硬件为中心，而是更倾向于以软件为中心，因此无法按照传统系统的设计方式来开发此类系统。

此外，开发周期越来越短，而需要新功能的情况越来越多。所有这些难题让工程师不得不更加频繁地应对新挑战，这不仅给他们带来了沉重的负担，而且也需要巨大的投资，但并非所有问题一开始就会显现出来。这一切都意味着软件设计人员在开发应用程序时需要帮助，以便能够有效地处理数据和接口，而不必从头开始编写所有代码。他们希望专注于为增加应用价值，而不是自己编写底层驱动程序或安全程序。

在这种背景下，适用于 RA 产品家族的灵活配置软件包 (FSP) 登上了舞台。它提供了板级支持包 (BSP)高效率的 HAL 驱动程序以及易于使用的中间件。FSP 中内含 Amazon 的 FreeRTOS® 实时操作系统和微软的 Azure® RTOS，但由于 FSP 符合 CMSIS RTOS 标准，因此工程师也可以采用他们选择的任何 RTOS。系统设计时也可以重用客户的原有代码。借助这种开放软件生态系统，设计人员可以轻松创建互联 IoT 终端系统或基于人工智能的边缘应用所需的功能。

1.2 RA 产品家族单片机

瑞萨 RA 产品家族单片机包括四个系列——已经发布的 RA2、RA4 和 RA6 系列，以及计划发布的 RA8 系列——适用于从小型电池供电的传感器应用到高性能、处理密集型的嵌入式终端产品。借助用于模拟、连接、人机界面、安全、电机和逆变器控制等的片内外设，该产品家族非常适合快速扩张的物联网 (IoT) 和边缘计算应用领域，但并不仅限于此。

所有 RA MCU 均基于 32 位的 Arm® Cortex®-M 内核：RA2 系列基于 M23 内核，而 RA4 和 RA6 系列器件基于 M4F 内核或带有 Arm v8-M TrustZone® 的 M33F 内核。所有这些器件均包含 Arm 的标准外设，如嵌套矢量中断控制器 (NVIC)、Arm 存储器保护单元 (MPU) 或串行线调试 (SWD) 和嵌入式跟踪缓冲器 (ETB)，非常便于开发。此外，瑞萨还在 Arm 没有解决方案和需要额外性能或功能的领域加入了自己的知识产权 (IP) 模块。这些额外的 IP 模块基于瑞萨的成熟技术，可满足 RA 产品家族的需求。

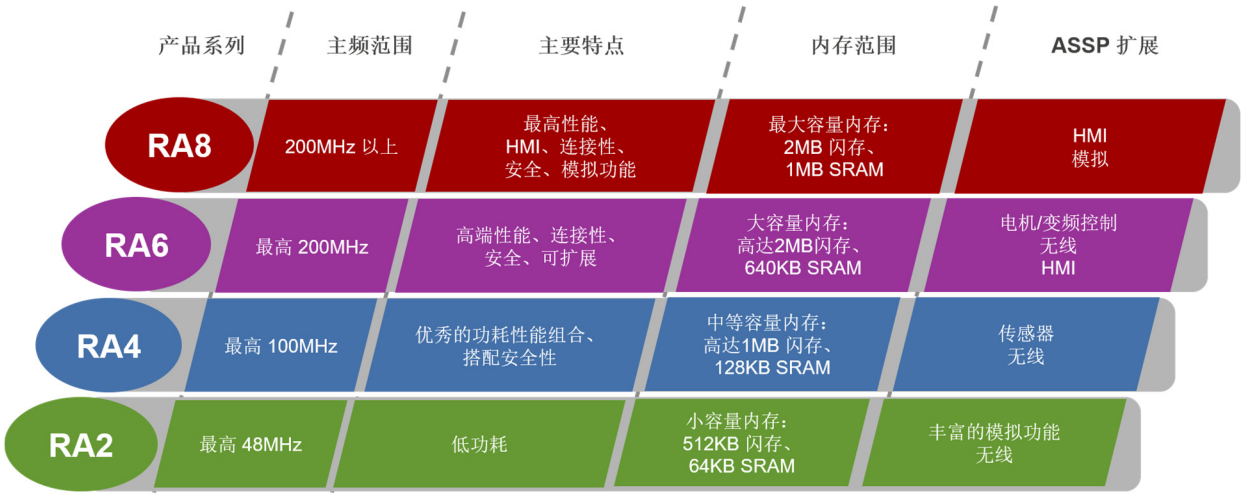


图 1-2：RA 产品家族单片机的初始系列

撰写本文时，已经发布了三个 RA 单片机系列，每个系列分成多个产品群，具体特性如下。

- **RA2 系列 – 低功耗：**基于 Arm Cortex-M23 内核，最高频率 60 MHz，拥有高达 256 KB 的闪存和 32 KB 的 SRAM。电源电压范围为 1.6 V 到 5.5 V。外设包括全速 USB、CAN、24 位 ∑-△ 模数转换器 (ADC)、16 位数模转换器 (DAC)、电容式触摸感应以及安全功能。

- **RA4 系列 – 高性能和出色的功耗：**基于支持 TrustZone 的 Arm Cortex-M33F 内核或 Arm Cortex-M4F 内核构建，最高频率 100 MHz。高达 1 MB 的闪存和 128 KB 的 SRAM。电压范围为 1.6 V 到 5.5 V。外设包括电容式触摸感应、段码式 LCD 控制器、全速 USB、CAN、安全功能以及数据转换器和定时器。RA4W1 系列器件还额外配备了 Bluetooth® 低功耗 (BLE) 5.0。

- **RA6 系列 – 高性能：**基于支持 TrustZone 的 Arm Cortex-M33F 内核或 Arm Cortex-M4F 内核。最高频率 240 MHz。高达 2 MB 的闪存和 640 KB 的 SRAM。电压范围为 2.7 V 到 3.6 V。外设包括数据转换器、定时器、外部存储总线、以太网、全速和高速 USB、CAN、安全功能、电容式触摸感应和用于 TFT 显示的图形 LCD 控制器，以及一个 2D 图形引擎。RA6T1 和 RA6T2 系列器件带有用于电机控制的增强型外设，如高分辨率 PWM 定时器或高级模拟模块。

随着新器件的推出，每个系列的产品都会逐步丰富起来。

每个系列的所有 MCU（在某种程度上，整个产品家族的 MCU）在功能上和大部分引脚上都是兼容的。小型器件上的外设大体上是大型器件上外设的子集。这便于实现可扩展性和不同器件之间的代码重用。对于不同系列的类似封装，其引脚排列几乎相同。这样一来，开发人员便不必在一开始就选好最终使用的器件，因为稍后可以改用其他器件。此外，在 PCB 布线时在器件封装内创建多种尺寸的封装，可为最终产品的制造提供灵活的选项。

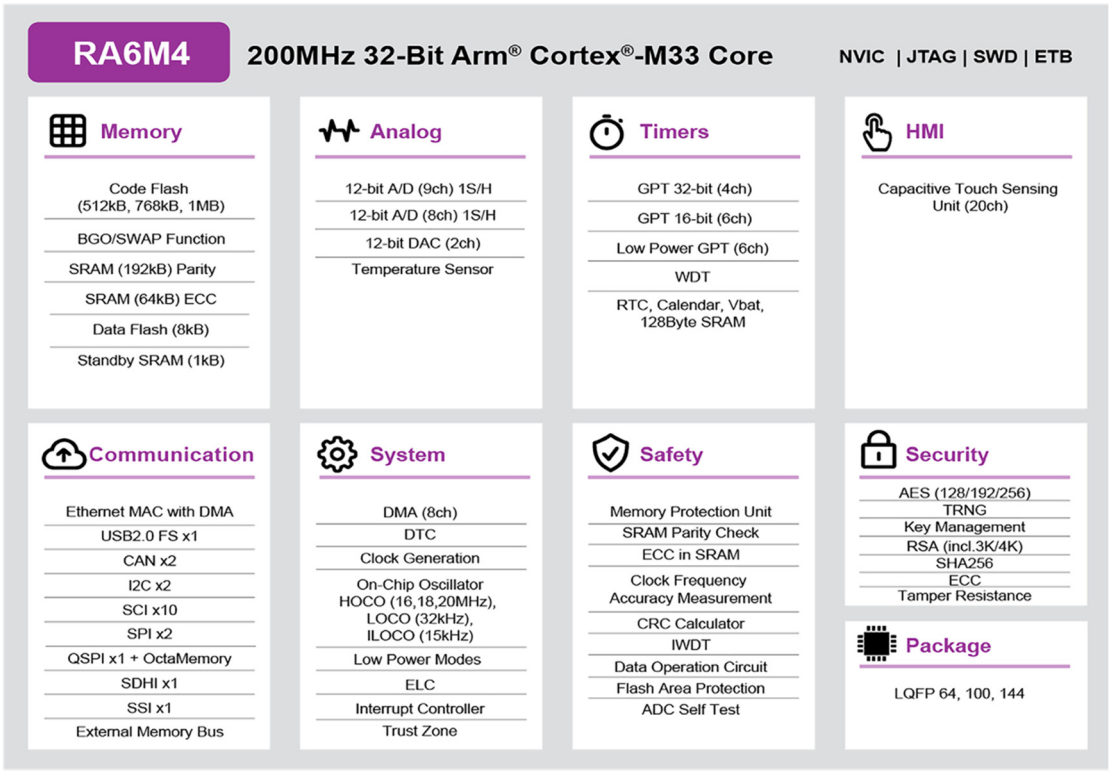


图 0-3：RA6M4 器件框图

图 0-3 显示了 RA6M4 系列器件的主要特性和外设，代表了 RA 产品家族的高性能 RA6 系列。本手册中的大多数示例和项目均基于该单片机的评估板套件。

有时候（特别是对初学者来说），要理解 RA 产品家族产品型号中的不同数字和字母并不容易，因此*图 0-4* 解释了不同字段的含义

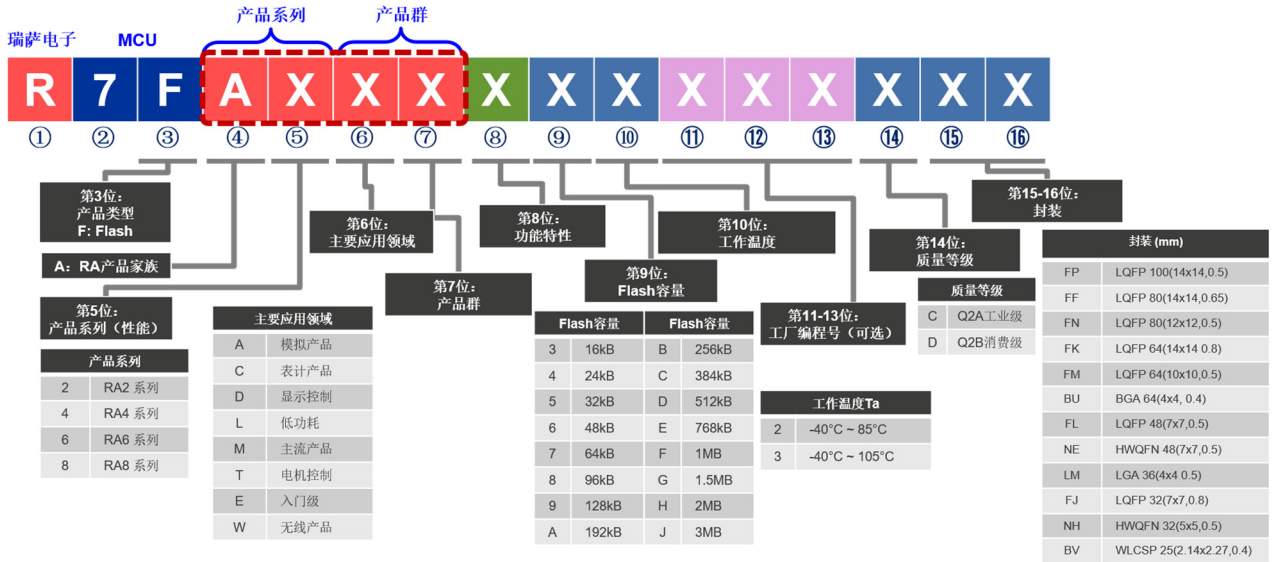


图 0-4：适用于 RA 产品家族的产品型号解释图

1.3 灵活配置软件包

适用于 RA 产品家族单片机的瑞萨灵活配置软件包 (FSP) 提供了一种快速的通用方法，可创建智能物联网安全互联设备所需的软件，并已专门针对 RA 单片机的架构进行过优化。FSP 具有开箱即用的中间件-和协议（例如 TCP/IP 协议或安全功能协议）、板级支持包 (BSP)（可为瑞萨的 MCU 和开发板提供启动和初始化代码），以及用于所有外设的硬件抽象层 (HAL) 驱动程序。这些驱动程序不仅性能高，而且占用的存储空间也极小。

所有驱动程序、协议栈和中间件功能均可通过易于使用的应用程序编程接口 (API) 进行访问，从而轻松实现互换，并可与实时操作系统 (RTOS) 以及裸机实现搭配使用。此外，软件的各个层均内置了对 Arm 的 TrustZone 的感知，并使用 Arm 的统一 API 来保证安全。FSP 是开源软件，提供完整的源代码，但仅限用于瑞萨的硬件。

除了上述软件外，FSP 还包含 Amazon 的 FreeRTOS™ 和微软的 Azure™ RTOS 作为实时操作系统。此系统可通过符合 Cortex 微控制器软件接口标准 (CMSIS) 的 Arm RTOS 接口进行访问。借助此标准接口，软件工程师能够使用其选择的任何 RTOS，而不会失去 FSP 的优势。

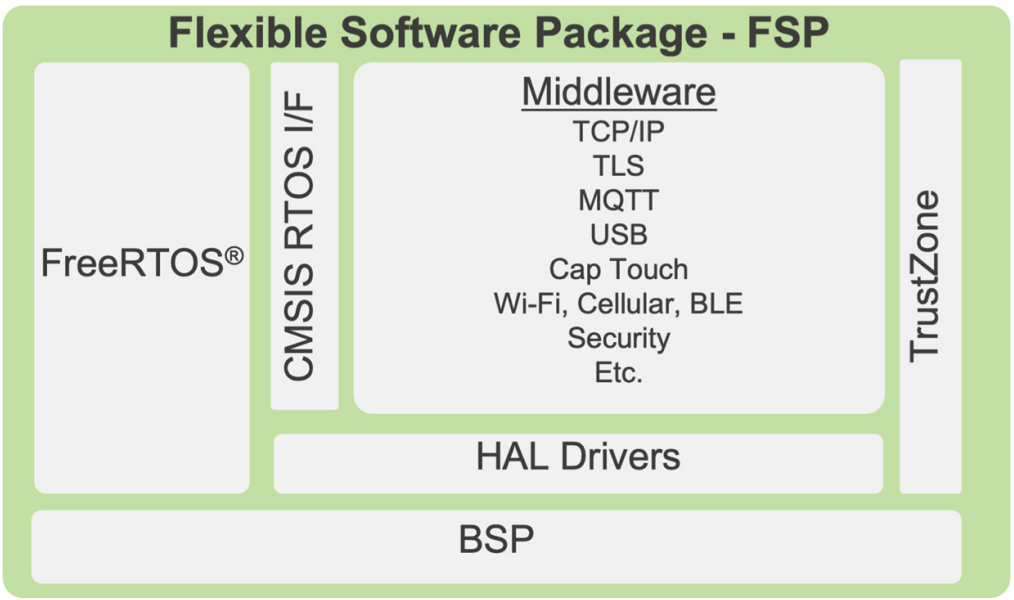


图 0-5：灵活配置软件包的架构

为了使用起来尽可能简单省力，FSP 配备了直观的配置器和代码生成器，不仅可以初始化 MCU 及其外设，还可以初始化 RTOS 和中间件模块。开发人员不会受到 FSP 功能的束缚：他们可以随时集成自己的应用程序代码和中间件模块。反之亦然：FSP 可与整个 Arm 软件生态系统配合使用。

如果您想了解有关 FSP 的详细信息，请不要着急。我们将在第 2 章和第 3 章中加以介绍。

1.4 RA 开发工具和评估板套件：概述

瑞萨为创建实用的软件和硬件开发工具付出了巨大的努力，这些工具可以用来探索 RA 产品家族的技术能力，并在从评估阶段一直到生产阶段的所有环节内为用户提供支持。

除了在第 0 章中介绍的灵活配置软件包外，在撰写本文时，可用的开发工具包括：

- **e² studio**：基于 Eclipse 的集成开发环境 (IDE) 可用于 Microsoft Windows®和 Linux 操作系统，其包含了用于创建、编译和调试 RA 产品家族项目的所有必要工具。其配置器允许轻松地以图形方式访问诸如创建新项目之类的任务，或访问诸如时钟模块或引脚功能选择之类的不同硬件功能的配置，以及访问用于中间件、驱动程序、板级支持包和 RTOS 的软件模块选项。所有这些操作均不需要特别详细地研究用户指南。
  这些配置器都将自动创建所有必要的设置和初始化代码，并包含错误检查功能，以便在设计阶段就检测出有问题的设置。对于代码编译，e² studio 使用 GNU Arm® Embedded 工具链的 GCC 编译器。

- **QE 工具**：QE 是“Quick and Efficient”（快速和高效）的缩写，用于实现电容式触摸的 QE 和用于实现 Bluetooth® 低功耗 5.0 的 QE 已直接集成到 e² studio 中。用于实现电容式触摸的 QE 简化了触摸用户界面的初始设置和灵敏度的调整，而用于实现 BLE 5.0 的 QE 可以测试蓝牙低功耗协议栈的通信功能。

- **RA Smart Configurator**：RA Smart Configurator (RASC) 是一款桌面应用程序，可为用户提供与集成到 e² studio 中的 Smart Configurator 相同的功能。借助 RASC，使用第三方工具链和 IDE 的开发人员可以像 e² studio 的用户一样，以相同的方式访问软件系统（BSP、HAL 驱动程序、中间件、RTOS）的项目设置和图形化配置、引脚分配和时钟设置。

- **评估板套件**：瑞萨针对各个 RA 系列的多款产品提供单独的评估板套件，并为各 MCU 系列中的每个超集器件提供至少一个评估板套件。评估板上集成了调试器，使设计人员能够充分评估器件的数字和模拟引脚，以及其连接和安全功能——甚至可在自己的硬件准备就绪之前完成评估。
  评估板可以通过连接器进一步扩展，以兼容一些备受欢迎的生态系统，如 Digilent Pmods 或兼容的 Arduino™ 开发板。由于提供了包含 BOM、图纸和制造文件的设计包，这些评估板可作为构建定制硬件的快速入门工具，从而进一步加快应用程序的开发速度。

- **第三方工具和瑞萨 RA 合作伙伴生态系统**：一个全面的 RA 合作伙伴生态系统，提供可立即与瑞萨 RA 产品家族 MCU 搭配使用的各种软件和硬件构建模块。开发人员可以从一系列解决方案中任意选择，包括安保、安全和联网等技术。这些第三方工具的示例有 Keil® 的微控制器开发工具包 (MDK) 和 IAR Systems® Embedded Workbench®。

除了软件和硬件工具之外，瑞萨还提供了设计支持：有关各种白皮书、教学视频和示例项目，请访问瑞萨 RA 产品家族的网页 (https://www.renesas.com/ra) 获取。它们可以帮助开发人员和系统设计人员轻松加快设计速度。另外，如果您想深入了解一些主题，可以登录瑞萨学院 (https://academy.renesas.com/?eid=1618) 网站查找相关培训课程。

在开发过程中，有时会遇到问题或难以继续，但往往只需点击几下鼠标问题就可迎刃而解。建议登录 RA 论坛 (https://www.renesas.com/ra/forum) 或知识库 (https://en-support.renesas.com/knowledgeBase) 寻找解决方法。也可以从 e² studio 直接访问论坛和知识库：*只需转到“Help → RA Helpdesk”（帮助 → RA 帮助台）*或转到 *“Help → RenesasRulz Community Forum”（帮助 → RenesasRulz 社区论坛）*。

本章要点：

- 瑞萨 RA 产品家族单片机将致力于实现安全、互联和智能 IoT 方面的承诺。
- RA 产品家族以丰富多样的软件和硬件工具以及 RA 合作伙伴生态系统为后盾。
- 访问瑞萨网站即可轻松获得帮助。

2 灵活配置软件包 (FSP)

您将在本章中学到以下内容：

- 灵活配置软件包 (FSP) 所包含的不同组件，及其层次划分。
- 不同层的详细信息以及它们如何协同工作。
- 有关 FreeRTOS™ 和 Azure RTOS 实时操作系统的详细信息以及使用 RTOS 的优势。

如果在单片机 (MCU) 上运行的软件不满足任务要求，或者软件生态系统过于复杂而不便于开发人员使用，都将阻碍 MCU 发挥其全部潜力。成熟的理念（如 RA 产品家族 MCU），加上配套的软件开发工具和 RA 合作伙伴生态系统，可以帮助工程师开发易于编写和轻松维护的应用程序，而且还能够充分发挥该产品家族微控制器的性能和特性。

开发人员在使用 RA 产品家族时体验到的简洁性得益于所有主要部分（MCU、软件、工具、评估板和解决方案）的完美配合，瑞萨为此投入了大量的资源，以确保实现这个雄心勃勃的目标。

灵活配置软件包 (FSP) 专门针对 RA 产品家族 MCU 的架构进行优化，RA 产品家族 MCU 的开发也充分兼顾该软件的特性。在开发 FSP 的过程中，首要目标是为工程师提供简单高效的功能和驱动程序，以简化嵌入式系统中常见用例（如通信和安全）的实现。它们构成了一个开放的软件生态系统，可以灵活使用旧代码并与第三方工具结合使用。

FSP 集成了中间件协议栈、独立于 RTOS 的硬件抽象层 (HAL) 驱动程序（适用于生产），以及作为所有这些组件基础工具的板级支持包 (BSP)，还有广泛使用的来自 Amazon Web Services® 的 FreeRTOS™ 实时操作系统 (RTOS)。以此为嵌入式系统设计提供了一个经过优化且易于使用的高质量软件包，该软件包可扩展，并且可以通过操作简单而功能强大的应用程序编程接口 (API) 调用来访问所有功能，从而轻松实现互换性。整个 FSP 完全由瑞萨公司提供支持，开发人员可以从集成开发环境或 FSP 的 GitHub® 库 (https://github.com/renesas/fsp/releases) 查看或下载 FSP 的源代码，从而全面了解 FSP。

2.1 FSP 简介

灵活配置软件包 (FSP) 是一款综合性软件，旨在以较低的内存占用量提供快速高效的驱动程序和协议栈，可满足嵌入式系统软件开发阶段的大多数需求。FSP 中包括以下部分：

- **板级支持包 (BSP)**，针对每个硬件评估板和 RA 产品家族的微控制器进行定制。它为所有支持的模块提供起始代码并作为这些模块的基础，以确保 FSP 模块顺利运行。使用自定义硬件的开发人员也可以充分利用 BSP，因为开发人员可以借助 e² studio 中内置的 User Pack Creator 针对其最终产品和电路板来定制 BSP。

- 独立于 RTOS 的**硬件抽象层 (HAL) 驱动程序**，以较少的内存占用量为所有片上外设和系统服务提供高效的驱动程序。它们可以从您的硬件中提取位设置和寄存器地址，因此无需对微控制器中底层硬件的文档进行深入研究。

- **中间件栈和协议**，可以独立使用或与 RTOS 结合使用，使用 Arm® 提供的统一 API。它们简化了连接功能的实现，如 WiFi、Bluetooth® 低功耗或到云服务的 MQTT 连接。还包括其他协议栈，例如支持 USB 传输、图形处理或电容式触摸的协议栈。

- **FreeRTOS™ 实时操作系统，**提供可进行多任务处理的实时内核（采用抢占式调度形式），面向对象的灵活 RAM 分配，以及用于任务通知、队列、信号量和缓冲区的不同实现方法。FreeRTOS+FAT 和 FreeRTOS+TCP 库为需要网络连接的应用提供额外的功能。用户可自行选择是否使用 FreeRTOS；FSP 也可以与裸机系统或任何其他 RTOS 一起使用。

- **Microsoft Azure®** RTOS 实时操作系统，包含了 Azure RTOS ThreadX® 等组件和各种堆栈和中间件元素，如文件系统（FileX®）、图形用户界面（GUIX™), USB 和 TCP/IP 通信（USBX™ 和 NetX Duo）。ThreadX RTOS 提供抢占式调度程序、线程间通信和同步、软件定时器和小内存占用。其内存管理允许灵活分配 RAM 和检测堆栈溢出。所有这些软件包都集成到了 FSP 生态系统中，并简化了与 Azure IoT 的连接 。

- FSP 中还包含其他**第三方软件解决方案**。例如，Arm® Cortex® 微控制器软件接口标准 (CMSIS) 硬件抽象层、Arm Mbed™ Crypto 和 TLS 加密库、Arm Littlefs 故障安全文件系统、emWin 嵌入式图形库和 Segger 的 J-Link® 调试器软件，以及 TES D/AVE 2D 图形渲染库。

在 FSP 开发过程中要实现的一个目标是，创建简单易用的软件以及条理清晰、整齐划一的 API，并进行规范的文档记录。工程师针对每个模块都编制了详细的用户文档（包括示例代码），位于 GitHub 资源库中或通过 e² studio 的智能手册功能，可在需要的位置（即开发环境内部）显示信息。FSP 使用 Doxygen 作为默认的文档工具，因此各模块源代码的 Doxygen 注释中也提供了其他详细信息。

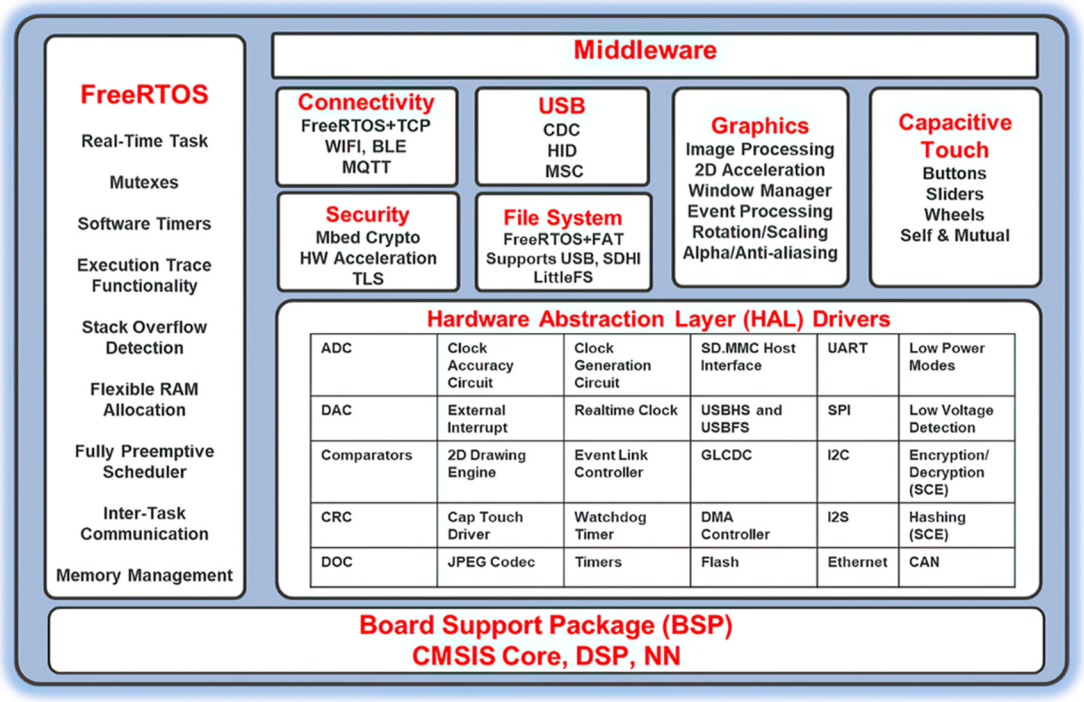


图 2-1：灵活配置软件包的层次划分及其功能

每个模块都支持构建时配置，它们可用于优化应用所需功能集的模块大小。FSP 的扩展性非常高，因为其模块对于所有 RA 产品家族微控制器都是相同的，前提是 MCU 具有使用该模块所需的外设。FSP 的质量通过同行评审、基于需求的自动测试和自动静态分析来保证。

图 **2-1** 显示了不同层的架构。可以通过 API 调用从用户应用程序访问各个层，但是强烈建议应用程序仅调用它们所连接的层的 API 函数。尽管 FSP 根据 ISO/IEC 9899:1999 (C99) C 语言编程标准编写代码，但所有 FSP 模块均允许用户在 e² studio 或其他支持的开发环境中使用 C 和 C++ 语言编写应用程序。软件的各个层均具有 Arm TrustZone® 感知功能，便于设计人员轻松创建安全程序。

FSP 定期进行更新和错误修复，计划的发行版本遵循产品路线图，每个主要或次要版本都会推出新功能。瑞萨始终对产品提供良好的支持，维护时间超过业内平均水平。这意味着您不会在获得“as is”分发的软件后便失去支持。

2.2 板级支持包 (BSP) 简介

板级支持包构成灵活配置软件包 (FSP) 的底层，并且为其他 FSP 模块正常协同工作奠定了基础。为此，其主要任务是确保 MCU 从复位状态切换为用户应用程序状态。在实现此目的的过程中，它将设置时钟、中断、堆栈、堆、堆栈监视器和 C 语言运行环境。它还会配置端口的 I/O 引脚，并执行任何特定的电路板初始化。

因此，此软件包为特定的电路板和 MCU 组合专用。二者都在 e² studio 中使用集成开发环境 (IDE) 的项目向导设置项目时选择。BSP 支持瑞萨提供的每一个评估板。e² studio 内的配置器将从 FSP 中提取必要的文件，并根据用户界面中的输入对它们进行设置。板级支持包本身基于大量的数据，包含配置文件、头文件和应用程序编程接口 (API)。

BSP 的内核符合 CMSIS（Arm® Cortex® 微控制器软件接口标准），遵循该标准的要求和命名规则。板级支持包为每个 MCU 使用一个静态的数据宏列表，其中宏本身对所有处理器而言都是相同的，但对于不同的 MCU，它们会有不同的值。

BSP 提供公共函数，供任何使用该软件包的项目使用，这些函数允许访问 BSP 所支持的 MCU 和电路板之间通用的功能。这些功能包括锁定/解锁硬件和软件、中断处理、注册回调函数和清除标志位、软件延迟和寄存器保护。这些例程的名称以 R_BSP_ 开头，相关的宏以 BSP_ 开头，数据类型定义以 _bsp 开头，以便于与 FSP 的其他部分区分开来。唯一的例外是提供 CMSIS 内核中描述的功能的例程。此外，BSP 还包括一些用于引脚功能的例程，如读或写，以及返回电路板上可用的 LED 信息（数量和 I/O 引脚）。应用程序代码可直接使用这些功能。

新评估板和器件一经推出，便会立即添加到 BSP 中，以确保为现有设计和新设计奠定坚实的长期基础。借助 e² studio 中内置的 User Pack Creator，可轻松生成对定制电路板的板级支持包。

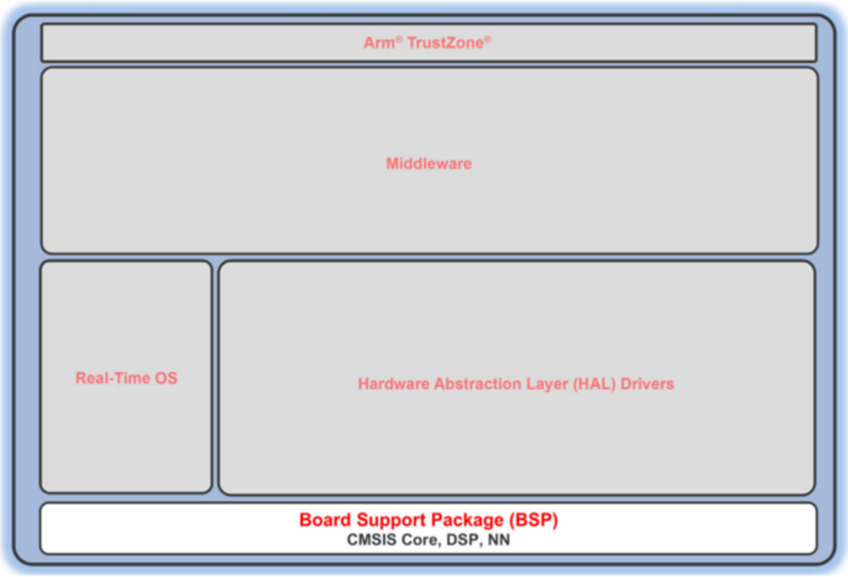


图 2-2：板级支持包奠定了灵活配置软件包的功能基础

2.3 HAL 驱动程序简介

在板级支持包 (BSP) 之上是硬件抽象层 (HAL)，它以较小的内存占用量为外设提供高效的设备驱动程序，并且与 MCU 的寄存器一致，以实现易于使用的接口，使程序员无需考虑具体的硬件。它是模块的集合（参见*图 2-3*），每个模块都是 RA 产品家族微控制器中可用的外设的驱动程序，如串行外设接口 (SPI) 或模数转换器 (ADC)，其名称以 r_ 开头，便于与灵活配置软件包 (FSP) 的其他部分区分开来。所有这些模块本质上均与 RTOS 无关。

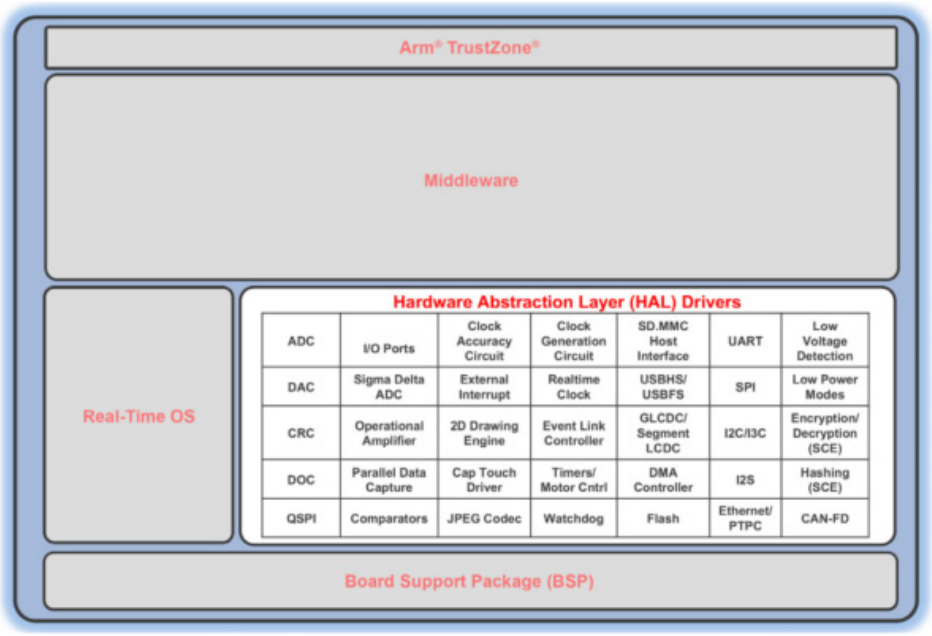


图 2-3：HAL 层为片上外设提供驱动程序

用于将硬件抽象化的接口在 HAL 的所有模块中都是一致的，并且可以扩展。有些外设支持多个接口，有些接口获多个外设支持。优势是操作更加灵活，因为可以在更高级别上修改需求。例如，如果代码最初是为专用的硬件 SPI 外设编写的，但后来需要使用串行通信接口 (SCI) 的 SPI 功能，只需更改配置信息即可。应用代码本身保持不变（参见*图 2-4*）。虽然可以通过 HAL 接口直接访问 API 功能，但大多数功能也可以通过 FSP 中的不同中间件和协议来访问，这实际上是首选方法。

通过使用符合行业标准的静态和动态分析工具执行单元和系统测试，可确保 HAL 驱动程序处于生产就绪状态。默认情况下，所有驱动程序都是非阻塞的，并返回执行状态。同样，驱动本身不分配任何内存；调用例程需要将工作内存传递给驱动程序。

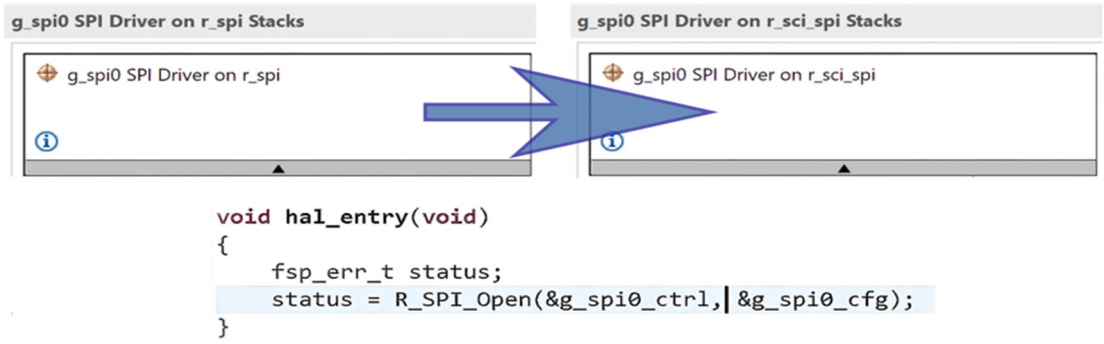


图 2-4：即使底层外设发生了改变（在此示例中，从 SPI 外设更改为 SCI 端口），应用程序代码仍保持不变

通过使用静态和动态分析的行业标准工具执行单元和系统测试，确保 HAL 驱动程序的生产准备就绪。默认情况下，所有驱动程序都是非阻塞的，并返回执行状态。此外，它们本身不分配任何内存；工作内存由调用例程传递给驱动程序。

2.4 中间件简介

中间件层位于 HAL 层之上和用户应用程序之下，为应用程序提供功能栈和协议。其目的是简化诸如以太网和 USB 之类的复杂外设的使用，或简化 RA 产品家族单片机 (MCU) 的不同安全功能。中间件层的模块可对各种系统级和技术特定的服务进行抽象化，从而通过简单的 API 实现丰富的功能。它们与 FreeRTOS™ 集成以管理资源冲突和多个用户线程之间的同步，但它们也可用于裸机实现或任何其他实时操作系统。

灵活配置软件包 (FSP) 的中间件模块提供可轻松实现连接的选项，如通过 FreeRTOS+TCP 实现可扩展且线程安全的 TCP/IP 协议栈，或通过易于使用的 API 访问 MCU 芯片上的以太网外设。其他模块支持包括器件配置在内的 WiFi 和套接字实现。另外，还提供 Bluetooth® 低功耗 (BLE) 5.0 功能，使用户能够在特定的微控制器上控制 BLE 无线电外设，如 RA4W1 系列器件。

可以在主机和设备模式下支持多类 USB 通信。所有提到的中间件模块均可通过不同的配置器或 QE 工具轻松设置。

借助 FreeRTOS+FAT 和 Azure RTOS FileX，可使用与 DOS 兼容的文件系统，该系统可实现基于文件的访问和存储，并且为线程感知型且可扩展。允许应用使用 FAT12、FAT16 或 FAT32 文件系统。还支持 Arm 的 LittleFS，这是一套专为微控制器和嵌入式系统开发的小型故障安全的文件系统。

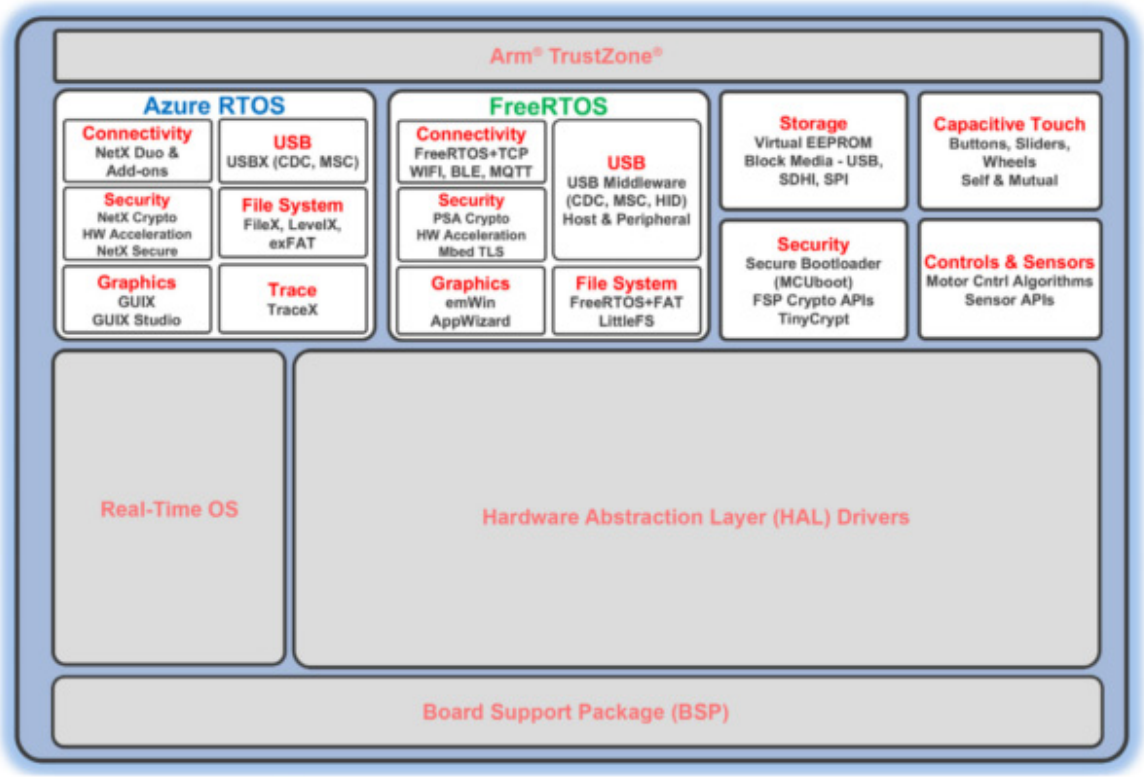


图 2-5：FSP 的不同中间件模块可提供简单的连接和其他功能

其他模块可帮助软件工程师创建具有端到端云连接的应用程序。通过这些模块支持将用户软件与 Amazon Web Services®、Microsoft Azure™ 和 Google Cloud Platform™ 服务直接连接。通过以太网或 WiFi 接口进行通信，并支持 MQTT 协议。通过 Arm 的 Mbed™ Crypto 接口的传输层安全 (TLS) 协议可保证连接的安全性。

FSP 的加密 API 基于 Arm 的 Mbed Crypto，它们支持硬件和软件加密技术。支持 AES 128、AES 256、SHA-256、SHA-224 和 RSA 2048 算法，以及椭圆曲线加密法 (ECC) 和真随机数生成 (TRNG)。还可以为使用 Arm PSA Crypto API 的 Mbed Crypto 实现硬件加速以及安全调试。

对于使用电容式触摸感应的应用，基于瑞萨电容式触摸感应单元 (CTSU) 技术的驱动程序和中间件唾手可得。它们包括可轻松集成按钮和滑块等部件的 API，这些 API 可通过“QE for Capacitive Touch”工具轻松创建。

对于某些 MCU（例如 RA6M3 系列器件），可以使用支持图形的中间件：FSP 包含高质量的 Segger emWin 图形库。它们可实现图像处理、2D 加速、窗口管理器、事件处理、旋转/缩放、alpha 混合/抗锯齿等功能。

使用中间件和协议的开发人员将受益于更高级别的抽象化能力，可以在不同的处理器或者说潜在产品上重用代码，代码一致性更高，并且不必在不同的应用程序之间重新创建常用的任务。同时，灵活配置软件包的其他部分进一步减少了与微控制器中的硬件直接连接的需要。

2.5 实时操作系统的介绍

灵活软件包（FSP）不仅有一个，而且有两个非常流行的实时操作系统（RTOS）：免费 RTOS™ 来自 Amazon Web Services®和 Microsoft Azure®RTOS ThreadX®。两者不仅为软件工程师提供服务从操作系统（如实时线程、抢先调度程序、内存管理器或通过队列、信号量和缓冲区。它们还提供了额外的中间件库。其中包括文件系统、通信堆栈 USB 和 TCP/IP 等。通过这种方式，他们积极支持工程师，同时开发深度嵌入式应用程序独立或连接到各种云服务。

但为什么要使用 RTOS？各个操作系统的具体功能是什么？以下段落将给出这些问题的答案。

2.5.1 为什么使用 RTOS？

不算太久之前，嵌入式系统的软件开发人员的工作还比较简单。大多数系统仅包含一个单一的后台任务，该后台任务在 main() 函数中无限循环运行。系统执行的任务包括从 I/O 引脚读取输入（例如按钮的状态），然后进行某些计算（例如将测量的电压和电流相乘以得到功耗），最后更新各个输出等。

连接到微控制器的为数不多的外设会发出中断，通知 CPU 可能有来自模数转换器的新转换结果可用，应对其进行读取。或者，RS-232 端口完成了最后一个字的传输，正在等待新数据。传统上，这些异步事件都是在中断服务程序内部处理的。如果没有待处理的中断，则软件将在后台循环中处于空闲状态，等待发生中断。

随着嵌入式系统的复杂性日益提升，与外部环境的连接需求增加，或者需要服务的用户界面和需要执行的任务增多，上面提到的裸机设置难以应对这种局面，而操作系统不仅受到青睐，更是成为不可或缺的部分。原因何在？后台循环的响应时间难以预测，而且在裸机实现中无法确定，因为它受代码内部决策树的影响，一旦对软件进行修改，就会发生变化。另一个缺点是，后台循环中的所有任务（或“线程”）都具有相同的优先级，因

为代码是连续执行的，这意味着即使多次按下按钮，上文示例中的按钮读取和功率计算操作也总是一个接一个地执行。换句话说，其中的一些事件可能会遗漏，这在嵌入式计算中是不允许出现的。如今的系统需要更迅速的响应和更高的可预测性；它们需要实时处理能力。

有一种方法是自己编写所需的功能。但是，设计和实现自己的调度程序或任务间通信是非常复杂的工作，而且容易出错。而且，这样的软件很难移植到新的处理器上，更难以维护。因此，实时操作系统使程序员的工作变得更加轻松，特别是在从一开始就将 RTOS 包含在软件中时，尤其如此。

明确了对操作系统的需求后，下一个问题是：为什么不使用诸如 Windows® 或 Linux 这类现成的操作系统？这些系统非常常见，并且性能也非常不错。不使用它们的原因有以下几个。首先，它们提供的功能过多，其中大部分功能在嵌入式系统中是不必要的，而且它们的可配置能力较差。其次，它们需要更多的资源和内存空间，而资源有限的嵌入式应用中无法提供。最后一点是，它们的时序不确定性仍然太大，无法满足实时系统的需求。

所有这些都表明，在我们的嵌入式系统上运行的操作系统需满足一些特殊需求。第一个也是最重要的一个需求是时序行为的可预测性。RTOS 必须具有确定性，这意味着开发人员必须知道并可获取阻塞时间的上限，例如禁用中断的时间。

第二个需求是 RTOS 必须管理时序和调度。RTOS 必须清楚知道时间限制和截止时间，并且必须提供高精度的时间服务。最后，操作系统应该具有快速、小巧和可配置的特性。

RTOS 提供的另一个主要功能是线程管理，通过处理线程的状态、进程的排队，允许使用线程（一个线程可以理解为一个轻量级进程）在 MCU 上执行准并行任务，并允许抢占式线程和中断处理。为应用程序提供的其他服务包括调度、线程同步、线程间通信、资源共享和作为时间参考的实时时钟。

许多软件设计人员对使用 RTOS 具有畏惧情绪，他们担心其复杂性和学习难度。但是，实时操作系统优势非常多，比如对实时事件的响应速度提高；可对线程进行优先级排序且容易插入；入门后即可以减少开发时间；以及可将服务添加到应用程序中。因此，其优点远远超过了最初的困难。

万物都不可能十全十美，使用 RTOS 当然也有一些缺点：它需要额外的 RAM 和闪存，而且每个线程都需要自己的堆栈，这进一步增加了内存需求。成本可能也是一个因素，但这并不是针对 FreeRTOS™ 本身，因为该操作系统是免费的。

前面看了使用 RTOS 带来的好处，现在是时候研究 FreeRTOS 和 Azure RTOS 的各种特性了。

2.5.2 FreeRTOS™ 的主要特性

FreeRTOS™ 是专门为嵌入式系统中的微控制器和小型微处理器而开发，在这些系统中，内存资源十分有限，并且必须具有经过验证的可靠性。如今，它已成为公认的嵌入式操作系统标准。它具有多任务调度程序，用于对象的内存分配的多个选项，以及多种用于任务通知、队列、信号量和不同缓冲区的实现方法。

FreeRTOS 的系统开销非常小，占用的内存也很小。通常，一个 FreeRTOS 内核二进制镜像需要 6 到 12 KB 的闪存，外加用于内核本身的几百字节的 RAM。该操作系统的设计十分简单：其内核仅包含三个通用源文件，以及一个目标微控制器的专用文件。

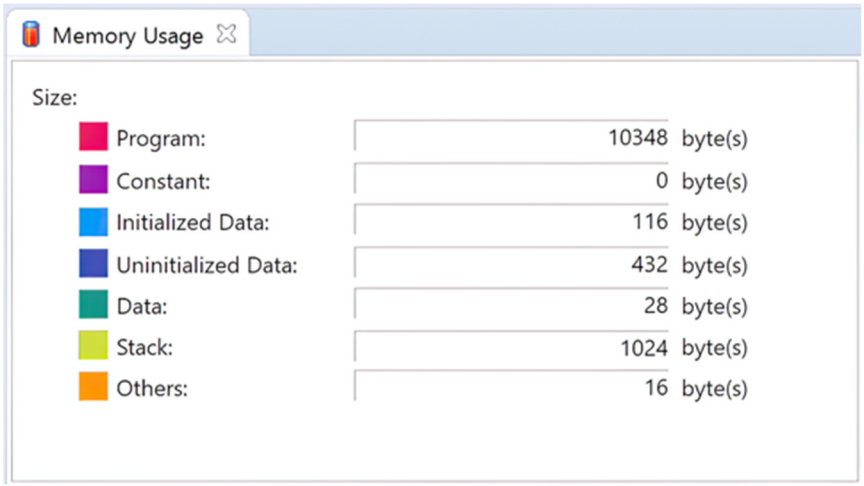


图 2-6： FreeRTOS 占用的内存非常小。在该示例中，一个没有线程的空 FreeRTOS FSP 项目仅占用略微超过 10 KB 的闪存，其中包括通过 BSP 初始化 MCU

这款实时操作系统功能丰富且有全面维护，是与世界领先的 MCU 制造商历时 15 余年共同开发的，现在归 Amazon Web Services® 所有。该操作系统根据 MIT 开源许可证以源代码的形式免费分发，并且适合 RA 产品家族的版本已集成到灵活配置软件包 (FSP) 中。该操作系统具有非常严格的质量管理，核心源文件符合 MISRA® 编码标准指南的要求。

FreeRTOS 允许进行抢占式或协作式操作，支持多线程、队列、互斥、信号量和软件计时器，并有非常灵活的任务优先级管理。任务通知使用简单、快速且通用的机制。已制定了用于跟踪记录和在运行时收集线程的统计数据的规定。

RTOS 的不同资源（如线程、互斥或内存管理）可以通过图形用户界面进行配置，既可以在 e² studio 内配置，也可以使用其他集成开发环境 (IDE) 的 Smart Configurator 配置，因此可轻松实现 RTOS 的功能。

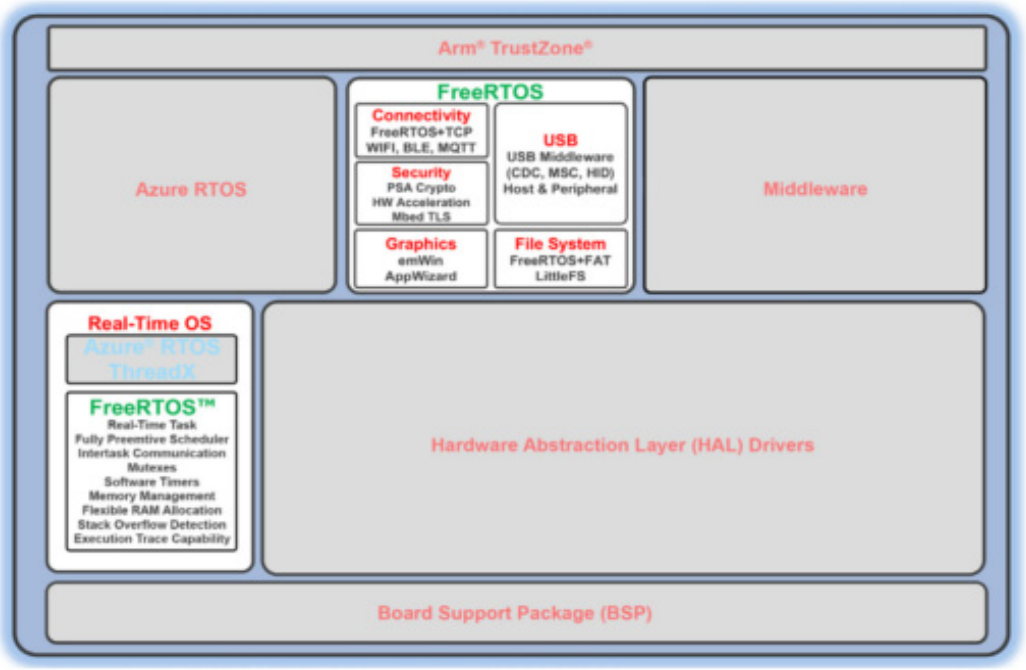


图 2-7：FreeRTOS™ 具有多任务调度程序，用于对象的内存分配的选项，以及多种用于任务通知、队列、信号量和不同缓冲区的方法。

FreeRTOS 的实时调度算法始终确保将执行已就绪的具有最高优先级的线程。如果有一个以上相同优先级的线程处于准备就绪状态，则有两种选择：如果在 RTOS 的配置文件中禁用了时间分片（FSP 的默认设置），则一旦发生 RTOS 时钟节拍（Tick）中断，调度程序不会在线程之间切换。如果启用了时间分片，则调度程序将在每个时钟节拍中断时在优先级相同的线程之间切换。轮循机制与时间分片的设置无关，用于确保所有线程都将在某个时刻进入运行状态。

对线程进行优先级排序时，如果启用了架构优化优先级排序方法，则软件工程师最多可以使用 32 个优先级；如果禁用了该方法并使用通用方法，则优先级数量可任意增减。由于每个优先级都会消耗内核的 RAM，并对执行时间产生负面影响，因此应尽可能地减少优先级的数量。另外，一个应用程序需要超过 10 到 15 个不同优先级的情况十分罕见，因为级别相同的线程将使用轮循机制进行调度，以确保每个线程都能得到执行。

FreeRTOS 中的内核对象（例如线程、队列或信号量）可以在编译时完全以静态方式创建，也可以在运行时以动态方式创建。这两种形式各有优缺点。具体取决于开发人员：他必须根据其应用程序的需求来做决定。动态内存对象分配更加简单，并且有可能减少应用程序的最大 RAM 使用量，因为删除对象后，就可以重新使用内存。动态分配自动执行，因此编写者无需处理此任务。

静态的内存分配为开发人员提供了更多的控制权，因为他可以将对象放置在所需的内存位置。他还可以确定链接时应用程序所需的 RAM 数量。选择静态还是动态分配的决定并不是一成不变的：如果有必要，这两种方法都可以在同一个程序内使用，从而实现最大的灵活性和对有限资源“内存”的优化管理。

RTOS 管理的其他资源为软件计时器。这些应用程序计时器有两种操作模式：单次和自动重载。单次计时器仅在计时器溢出后才调用一次用户函数，而周期性计时器则以固定间隔反复进入该用户函数。FreeRTOS 中软件计时器的实现非常高效，因为除非计时器溢出，否则它不会占用任何处理时间，并且不会在时钟节拍中断中增加任何开销。此外，在禁用中断时，该实现也不会遍历任何链表结构，也不会从中断服务程序 (ISR) 内部进入任何计时器回调函数。

软件计时器需要在使用前创建，它们在计时器服务任务的上下文中执行，该任务在调度程序启动后自动创建。计时器的创建可以通过 API 调用完成，也可以在编译时静态完成。计时器的周期以时钟节拍为单位，并且计时器的精度不能超过一个内核时钟节拍周期。

线程同步以及线程之间或中断与线程之间的通信是嵌入式系统中的另一个重要主题。FreeRTOS 为此提供了多种机制，使这项工作变得非常简单和直接。

线程间通信的主要形式是队列。它们是线程安全的先进先出 (FIFO) 缓冲区，并且通过复制消息将新数据添加到队列的末尾。这意味着消息本身将被完全复制，而不仅仅是引用。

信号量和互斥（互斥量）有助于防止优先级倒置并达到同步目的。对于诸如取消阻塞另一个线程或更新通知值之类的简单事件，可以使用简单的快速替代方法来替代二进制或计数信号量，有时还可以替代队列：直接指向任务的通知。它们允许线程与其他线程或 ISR 进行交互，而无需其他对象，从而节省了内存和时间。

最后要提到的同步功能是消息队列和流缓冲区。借助流缓冲区，开发人员可以在线程之间或从 ISR 到线程传递字节流。该字节流可以具有任意长度，并且一次可以写入和读取任意数量的字节。另一方面，消息缓冲区可以传递不同长度的离散消息。长度没有重大影响，但如果发送方写入到消息缓冲区的消息为 32 字节，则接收方读取的消息也需要为 32 字节。它们不能作为单个字节读取。

最后，FreeRTOS 还包括其他两个非常重要的功能：内置的跟踪功能，用于查看不同事件的顺序，以及检测与堆栈相关的问题（如溢出）的可能性。对应用程序进行故障排除时，两者都非常有用！

2.5.3 Azure® RTOS ThredX™ 的主要特性

ThreadX® 操作系统是微软 Azure RTOS 的一部分。该实时操作系统（RTOS）最初由 Express Logic 开发，目标是高端的、图形丰富的应用程序以及内存有限且在确定性方面有特殊要求的嵌入式系统。它具有内存小，RAM 需求小，上下文切换时间短的特点。

作为一种多线程 RTOS，ThreadX 使用多种高级调度算法，提供实时事件跟踪、消息传递和中断管理等诸多其他服务，并且具有完全确定性。它在消费、医疗电子和工业自动化市场拥有超过 100 亿次的部署，并符合许多重要的安全和质量标准。

其他高级功能包括诸如 picokernel™体系结构、Preemption-Threshold™调度、Event-Chaining™以及一整套丰富的系统服务。另外，它还与 Microsoft Azure RTOS 的其他组件集成，如 FileX®、GUIX™、27 TraceX®、NetX Duo™、 LevelX™、USBX™。它的 RAM 内存占用空间很小，设计者对其进行了优化，以实现快速执行和低开销，将微控制器的大部分资源用在了应用程序上。

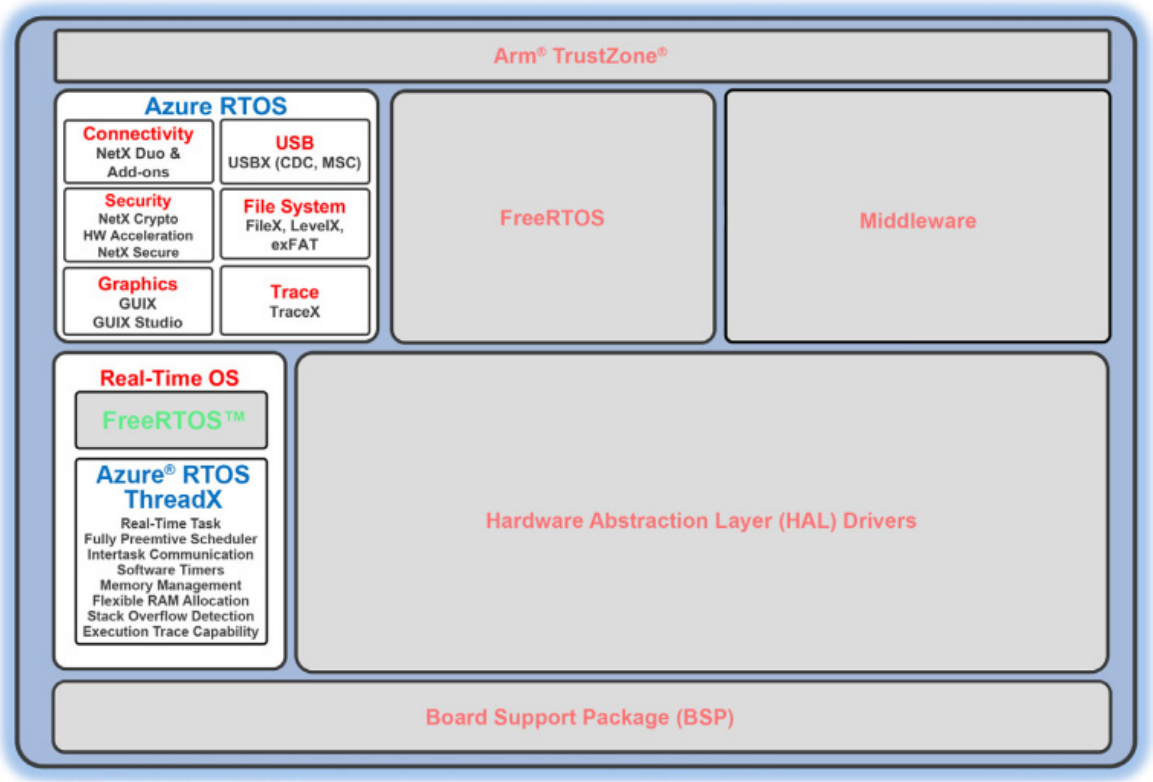


图 2-8：ThreadX®提供抢占阈值多任务调度器、事件链接和通信、内存管理以及其他系统服务

增强型的实时调度算法和高效的多任务例程提供了轮询调度和时间切片，以及抢占和 Preemption-Threshold™调度。为了对线程进行更优的排序，ThreadX®提供了多达 1024 个的线程优先级可供软件工程师使用，默认优先级为 32。

嵌入式系统中的资源主要受执行时间和可用内存的限制。ThreadX 为此提供了强大的选项进行管理。内存分配需快速且具有确定性，因此 ThreadX 允许创建和管理任意数量的固定大小的内存块池。由于池的大小是固定的，因此内存碎片不是问题，但内存块的大小必须大于等于应用程序的最大内存请求，否则内存分配将会失败。但如果多个请求为不同大小的块，则要满足其足够大可能会浪费内存。解决方法是创建包含不同大小内存块的多个内存块池。由于这些分配是在可用列表的开头完成的，因此快速的分配及取消分配弥补了这一不足。由此提供了尽可能快的链接列表处理，并可将实际内存块保存在缓存中。

ThreadX 不仅允许使用固定块大小的内存块池，还允许创建多字节内存池，此种操作与标准 C 堆类似。若采用首次适应的方式，便会以字节为单位分配所需内存。这样做的缺点是，就像 C 中的堆一样，字节池很容易被分割，从而产生某种不确定性行为。

RTOS 管理的其他资源是应用程序计时器，允许创建不限数量的软件计时器。这些应用程序计时器有三种操作模式：单次、周期和相对。单次计时器将在计时器到期后只调用一次用户函数，而周期计时器会在固定间隔后重复调用用户函数。相对计时器是一个连续递增的 32 位滴答计数器。所有计时器的过期时间都是以刻度表示的。例如，1 个刻度等于 10ms，当然刻度的大小是可配置的。

线程的同步和任务之间的通信是嵌入式系统中的另一个重要课题。ThreadX 为此提供了几种机制，使该课题的实现变得轻而易举。信号量和互斥有助于防止优先级反转，并允许创建无限数量的对象。另外还提供了复杂且先进的回调函数和事件链接，以及可选的优先级继承。此外，ThreadX 可以按 FIFO 或优先级顺序挂起，避免了线程缺少处理时间的问题。

事件标志是另一个线程同步功能。事件标志组由 32 位组成，每 1 位都代表一个不同的逻辑事件，线程可以等待这些位的子集。事件标志还支持事件链接。与信号量一样，可以创建无限数量的对象，并且可以获得有关标志运行时性能的信息。

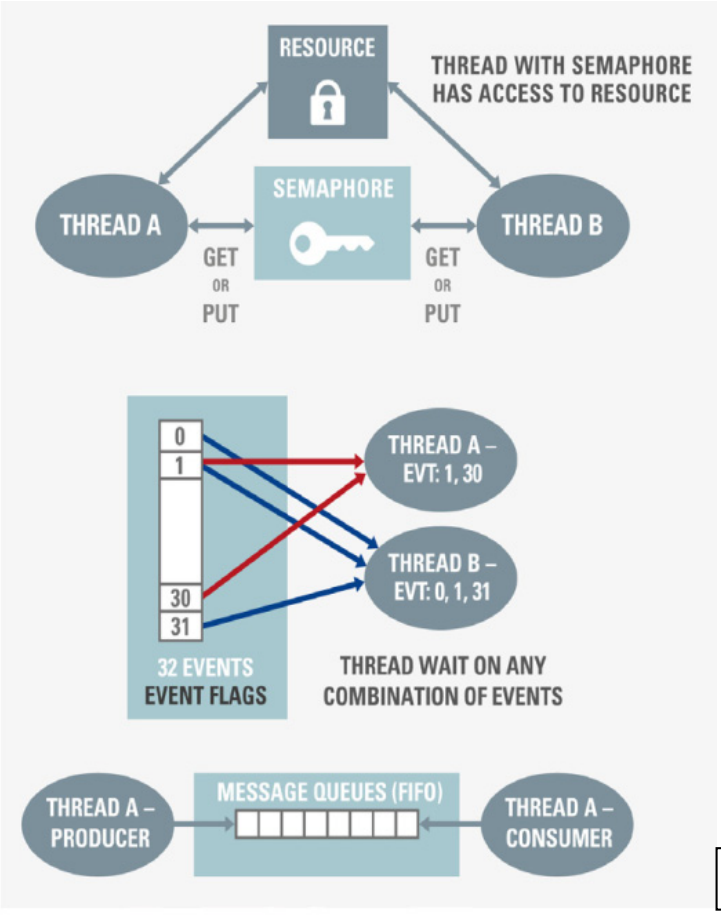


图 2-9：ThreadX® 同步和通信特性

这里要提到的最后一个同步特性是消息队列。通过生产者-消费者模式进行任务间通信，ThreadX 可支持 1~16 个四字节单词的消息大小，而邮箱则是大小为 1 的消息队列的特殊情况。应用程序线程可以注册回调函数以接收更多的通知。与事件标志一样，可以获得有关运行时性能的信息。

为方便调试，ThreadX 还有一个重要功能：TraceX® 内置了允许查看不同事件的顺序事件跟踪功能，以查看不同事件的序列。TraceX 虽是一个独立的 Windows® 应用程序，但编程人员可以在 e² stuido 中启动它。

最后，ThreadX 符合 Misra-C:2004 和 Misra-C2012 的所有“必需”和“强制性”规则，通过了 EAL4+通用标准安全认证，并通过了以下标准的预认证。

- IEC 61508 SIL 4
- IEC 62304 Class C
- UL/IEC 60730-1 H
- CSA E60730-1 H
- ISO 26262 ASIL D
- UL/IEC 60335-1 R
- UL 1998
- EN 50128 SW-SIL 4

本章要点：

- 灵活配置软件包 (FSP) 在各层中构建，所有功能都可以通过简单的标准化 API 调用进行访问。
- 中间件提供了丰富的连接性和安全性，以及图形和电容式触摸功能，而无需从零开始编写代码。
- FreeRTOS™ 和 ThreadX 都是功能强大、易于使用的操作系统，占用的内存很小，可将微控制器的大部分资源分配给应用程序。

3 灵活配置软件包的 API 简介

您将在本章中学到以下内容：

- FSP 代码的语法、风格和命名约定。
- 有关应用程序编程接口及其使用方法的详细信息。

瑞萨的设计人员在为 RA 产品家族单片机开发灵活配置软件包 (FSP) 时，考虑到了易用性。FSP 功能十分强大，但使用却极其简单，这是因为它的应用程序编程接口 (API) 架构非常简单和全面，在封装了 FSP 复杂性的同时，可由程序员完全控制各种功能。即使对诸如 USB 传输之类的复杂任务进行编程，也只需几行代码就可以实现，无需阅读厚厚的手册或研究指定单片机外设的寄存器组的每一个特性，让开发人员可以专注于开发应用程序的函数集，而不是编写对设计毫无价值但编写和测试却相当耗时的低级代码，因此极大地减轻了他们的负担。我们来了解一下 API 的一些详细信息！

3.1 API 概述

无论所需功能位于哪个层，应用程序均可通过直观、简单和统一的 API 调用来访问灵活配置软件包的所有功能。这样，就能以非常简单和直接的方式编写易于理解、维护简单、移植方便的代码，例如，目前设计中经常遇到的必须对特定任务使用不同的单片机外设的情况。

在该示例中，如果使用 FSP，从片上独立 SPI 外设更改为片上 SCI 端口的 SPI 功能仅需进行简单的更改即可：在 FSP 配置器中，“Stacks”选项卡中的 *“SPI (r_spi) driver”*（r_spi 上的 SPI 驱动程序）需要替换为 *“SPI (r_sci_spi) driver on r_sci_spi”*（r_sci_spi 上的 SPI 驱动程序）。由于两个驱动程序使用相同的实例 *g_spi0*，因此应用程序代码不会发生任何更改。

FSP 不同层的架构如*图 3-1* 所示：底层是 RA MCU，上面一层是板级支持包 (BSP)。BSP 负责使 MCU 复位为主应用程序，并为上层软件提供其他服务。BSP 的上一层是硬件抽象层 (HAL)，该层使开发人员不必直接处理单片机的寄存器组，并使 HAL 上一层的软件更容易在整个 RA 产品家族中移植。

HAL 之上是中间件栈和协议层以及实时操作系统。中间件层为用户应用程序提供连接选项、图形处理功能和安全功能。最后，用户程序位于最上层，通过统一的 API 接口调用下面各层。有关 FSP 的各层和各部分的详细信息，请参见第 2 章。

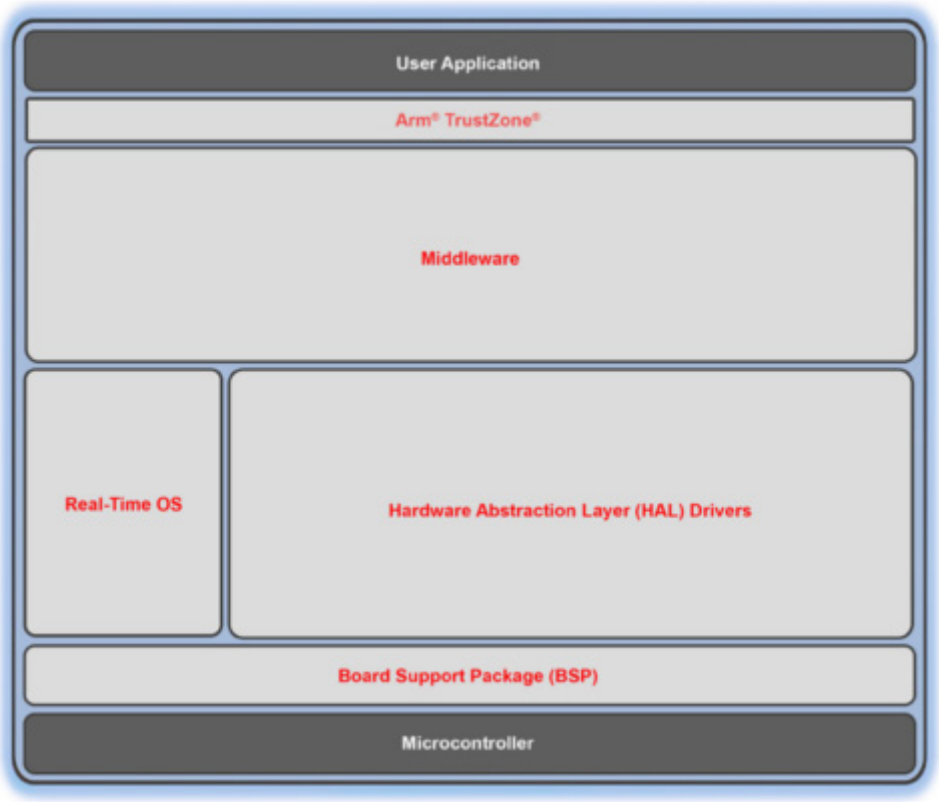


图 3-1：灵活配置软件包的层次划分

各个不同的层都可以通过 API 调用直接访问，或以堆叠方式访问。例如，需要 USB 传输的应用程序将调用 USB HCDC 驱动程序，该驱动程序使用基本的 USB 驱动程序，后者本身使用直接内存访问控制器 (DMAC) 模块的两个实例：一个用于传输，一个用于接收（参见*图* **3-2**）。当然，最终应用程序也可以直接调用 HAL 驱动程序和板级支持包，但使用中间件栈和协议层要容易得多，因为在这种情况下无需详细了解底层部分。

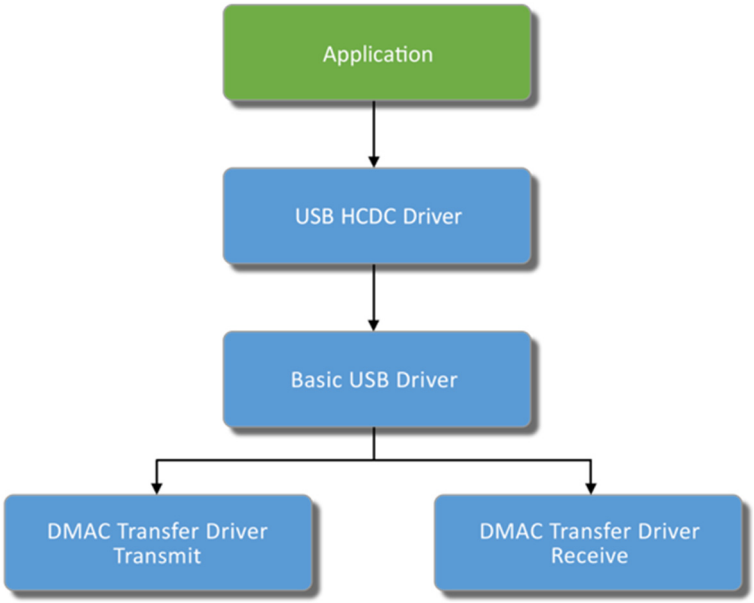


图 3-2：模块可以堆叠以便于使用

3.2 API 语法

在深入了解 API 的详细信息之前，我们应大致了解不同 API 和文件的命名约定。了解后，不仅应用程序编程变得更容易，而且数月之后，也更容易理解其他程序员的代码或自己的代码。后面这一点不容小觑，但我们往往会低估它的作用。掌握这些知识后，灵活配置软件包 (FSP) 所提供的清晰结构会大有裨益。

一般来说，内部函数遵循“NounNounVerb”（名词名词动词）的命名约定，例如 communicationAbort()。所有数据均在函数的输出参数中返回，并且第一个参数始终是指向其控制结构体的指针。由于此结构体存储了模块实例的内存地址，因此用户应用程序可以灵活地决定将其放置在何处。

下面是 FSP 中常用前缀的列表：

- **R_BSP_xxx**：常用的 BSP 函数的前缀，例如 `R_BSP_VersionGet()`。
- **BSP_xxx**：BSP 宏的前缀，例如 `BSP_IO_LEVEL_LOW`。
- **FSP_xxx**：常用的 FSP 的前缀，主要定义错误代码（例如 `FSP_ERR_INVALID_ARGUMENT`）和版本信息（例如 `FSP_VERSION_BUILD`）。
- **g_<interface>_on_<instance>**：实例的常量全局结构体的名称，用于将接口 API 函数与模块提供的函数关联起来。例如一个 HAL 层函数 `g_spi_on_spi`。
- **r_<interface>_api.h**：接口模块头文件的名称，例如 `r_spi_api.h`。
- **R_<MODULE>_<Function>**：FSP 驱动程序 API 的名称，例如 `R_SPI_WriteRead()`。
- **RM_<MODULE>_<Function>**：中间件函数的名称，例如 `RM_BLE_ABS_Open()`。

FreeRTOS™ 操作系统也遵循一种命名约定。例如，文件作用域静态函数具有前缀 *prv*。API 函数遵循 *<return type><Filename><Operation>* 的命名方法，其中 *“return type”*（返回值类型）遵循变量的命名约定，如果函数为 void 类型，则以字母“v”为前缀。 *“File”*（文件）字段包含定义 API 函数的文件名，“Operation”（操作）字段以 UpperCamelCase 语法命名需要执行的操作。例如，`vTaskStartScheduler(void)`，这是在 *tasks.c* 中定义的 void 类型的函数，用于启动 RTOS 调度程序；或者 `uxQueueMessagesWaiting()`，该函数返回非标准类型无符号 `BaseType_t` 数据，在文件 *queue.c* 中定义，返回队列中处于等待状态的消息数量。

类似的规则也适用于宏：定义宏的文件用作宏的前缀，以小写字母表示，后跟全部是大写字母的名称。例如，在文件 *tasks.h* 中定义的宏 `taskWAITING_NOTIFICATION`。

有关 FreeRTOS 命名约定的更多详细信息，请参见 Internet (**https://www.freertos.org/FreeRTOS-Coding-Standard-and-Style-Guide.html**) 上的 FreeRTOS 编码标准、测试和代码风格指南。

Microsoft Azure RTOS 的命名约定类似。它们是以以下方式构建的：<ID>_<名词>_<动词>。ID 是模块，名词是所讨论的对象（计时器、信号量等），动词是要执行的动作（创建、关闭、接收…）。例如 tx_queue_create()，它在 ThreadX®中创建消息队列。

以下是各种 Azure®RTOS 模块的几个 ID 的摘要：
fx: FileX®相关函数，例如 fx_directory_create()
gx: GUIX™ 相关函数，例如 gx_display_create()
lx：X 级™ 相关功能，例如 lx_nand_flash_open()
nxd: NetX DuoTM 相关函数，例如 nxd_ipv6_enable()
tx: ThreadX®相关函数，例如 tx_thread_create()
ux: USBXTM 相关函数，例如 ux_device_stack_initize()

除了掌握 API 的形式语法外，还有必要统一几个定义。我们对一些词汇的理解常常会有些许偏差，从而产生困惑。为此，我们提供了本手册和 FSP 中通用的术语列表：

- **模块（Modules）**：模块可以是外设驱动程序、纯软件或介于这两者之间，并且是 FSP 的构建模块。模块通常是独立的单元，但它们可能依赖于其他模块。可以通过组合多个模块来构建应用程序，为用户提供所需功能。
- **模块实例（Module Instance）**：单个、独立的实例化（配置）模块。例如，USB 端口可能需要使用 *r_dmac* 模块的两个实例与其他端口之间来回传输数据。
- **接口（Interfaces）**：接口包含 API 定义，具有相似功能的模块可以共用这些 API 定义。模块通过这些定义提供常用功能。通过这些 API 定义，使用相同接口的模块可以互换使用。可以将接口视为两个模块之间的合同，两个模块均同意使用合同中达成一致的信息进行协作。接口只是定义，并不会增加代码的大小。
- **实例（Instances）**：接口规定所提供的功能，而实例则真正实现了这些功能。每个实例都与特定的接口关联，并使用接口中的枚举、数据结构和 API 原型。这样，应用程序便可以在需要时交换实例。
- **驱动程序（Drivers）**：驱动程序是一种特定类型的模块，可以直接修改 RA 产品家族 MCU 上的寄存器。
- **堆（Stacks）**：FSP 架构所采用的设计方式是，模块可以协同工作以形成一个堆。堆由顶层模块及其所有依赖项组成。
- **应用程序（Application）**：归用户所有并由用户维护的代码，即使使用瑞萨提供的示例代码也是如此。
- **回调函数（Callback Functions）**：当有事件发生时（例如，USB 接收到一些数据时），将调用这些函数。它们是应用程序的组成部分，如果用于中断，应尽量简短，因为它们将在中断服务程序内运行，会阻止其他中断执行。

3.3 API 常量、变量和其他主题

如前所述，模块是灵活配置软件包的构建模块。它们可以执行不同的任务，但所有模块都具有基本的概念，即向上层提供功能，同时需要使用下层的功能，如*图* **3-3** 中所示。因此，使用 FSP 的最简单的应用程序包含一个模块，其中用户代码位于顶层。

模块可以堆叠，从而形成 FSP 堆，如*图* **3-2** 所示。将一个模块所能提供的功能与另一个模块所需要的功能相匹配，即执行了堆叠过程。例如，Block Media 的实现需要 SD/MMC 驱动程序，而 SD/MMC 驱动程序又需要数据传输接口，该接口可以由数据传输控制器 (DTC) 驱动程序模块提供。其代码并未特意包含在块媒体模块中，从而允许（底层）DTC 模块也可被其他模块（例如 UART）重用。

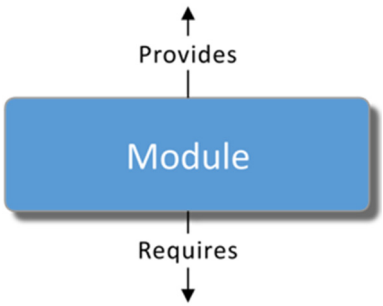


图 3-3：模块的构建方式如下：它们向主调函数提供功能，并需要使用下层的功能

模块堆叠能力有很大好处，因为它能保证应用程序所需要的灵活性。如果它们直接依赖于其他模块，则很难实现不同用户设计之间的可移植性。FSP 的架构设计使其可以通过使用 FSP 接口来交换模块（例如，用 UART 驱动程序来交换 SPI 驱动程序），因为使用相同接口的模块可以互换使用。还记得前面提到的合同吗？所有模块均同意使用合同中约定的信息协作。

在 RA 产品家族 MCU 上，不同的外设之间偶尔会发生功能重叠。例如，IIC 通信可以使用独立 IIC 外设或 SCI 外设的 IIC 模式来实现，并且两个外设提供不同的功能集。这些接口所采用的构建方式能提供大多数用户期望的常用功能，而省略了一些更高级的功能，但在大多数情况下，仍可通过接口扩展来获得这些功能。

每个 FSP 接口中至少包括三个数据结构体：第一个是取决于模块的控制结构体，名为 `<interface>_ctrl_t*`，用作使用该模块的唯一标识符。第二个是名为 `<interface>_cfg_t*` 的配置结构体，在 open() 调用期间作为模块初始配置的输入；第三个是实例结构体，包含三个指针：一个指向控制结构体，一个指向配置结构体，还有一个指向 API 结构体。后一种结构体的名称采用如下标准化形式：`g_<interface>_on_<instance>`，例如 `g_spi_on_spi`，该结构体本身可以通过实例的头文件中的 `extern` 声明来使用，例如 *r_spi.h*。虽然对于简单的驱动程序而言可以将这些结构体组合，但是通过这种方式创建结构体可以扩展接口的功能。

所有接口均包括一个 API 结构体，其中含有所有受支持函数的指针。例如，数模转换器 (DAC) 的结构体 `dac_api_t` 包含了 *open*、*close*、*write*、*start*、*stop* 和 *versionGet* 等函数的指针。API 结构体允许模块与使用相同接口实例的其他模块进行交换。

一旦有事件发生，模块就会通过回调函数向用户应用程序发出提醒。例如，通过 UART 通道接收一个字节就属于此类事件。此外，还需要回调以允许用户应用程序对中断作出反应。回调要尽可能简短，因为需要从中断服务程序内部对其进行调用。否则，它们可能会对系统的实时性能产生不良影响。

接口规定所提供的功能，而实例则真正实现了这些功能。每个实例都与特定的接口关联，并使用接口中的枚举、数据结构和 API 原型。这样，使用接口的应用程序便可以根据需要交换实例，从而在需要更改代码或所用外设时节省大量的时间。在 RA 产品家族 MCU 上，一些外设（例如 IIC）将具有一对一的映射关系（只映射到 IIC 接口），而其他外设（例如 SCI）将具有一对多的映射关系（实现三个接口：IIC、UART、SPI）。请参考 *图 3-4* 了解这种映射的图形表示。

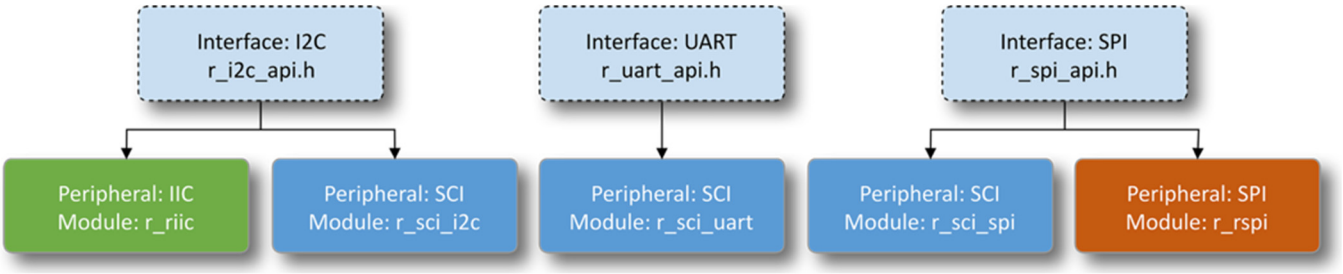


图 3-4：在 RA 产品家族单片机上，一些外设在接口和实例之间具有一对一的映射关系，而其他外设则具有一对多的映射关系

3.4 API 用法

了解了所有理论后，我们最后看看使用灵活配置软件包 (FSP) 有多简单。为此，我们将以一个简单的 SPI 为例，解释如何使用 API 以及在哪里可以找到有关不同项的信息。如果要在 PC 上执行此操作，可以在第 5 章之后返回本节。

使用 FSP 模块的第一步始终是为所需功能选择正确的接口。在我们的示例中，我们希望通过串行通信接口 (SCI) 的 SPI 接口模块进行通信，RA 产品家族的所有系列 MCU 均具有这种接口。因此，在 e² studio 中，我们在 FSP 配置器中选择 *"SPI (r_sci_spi) driver on "*（r_sci_spi 上的 SPI 驱动程序）。要实现上述目的，首先要在配置器的 *"Stacks "*（堆）选项卡的左侧突出显示 *"HAL/Common "*（HAL/通用）线程，然后在 *"HAL/Common Stacks "*（HAL/通用堆）下的右侧选择 *"New Stack → Connectivity → SPI (r_sci_spi) "*（新建堆 → 驱动程序 → 连接 → r_sci_spi 上的 SPI 驱动程序）。添加完驱动程序后，其他必需项会自动添加，直至达到用户需要选择或确定功能的程度。以使用 SPI 为例，r_dtc(数据传输控制器)模块 的 g_transfer0 和 g_transfer1 的实例会添加到工程中。第一个会将数据传输给 SCI 端口，并且默认的把 SCI 中断连接到 TXI，同时第二个和 RXI 中断连接起来并且接收来自串行接口的数据。

无需任何其他操作。在上面的示例中，接收、传输和错误中断均由 FSP 配置器自动启用。如果愿意，可以为默认名称为 `g_spi0`（数字取决于同类型驱动程序的添加顺序）的模块实例指定一个用户名（例如 `my_spi`）。此操作会额外增加一个抽象层，如果软件设计人员正确地选择了名称，可以使代码更容易移植并得到更好的维护，当然也会使其更易读。

可在 *g_spi0 SPI (r_sci_spi)* 模块的"Properties"（属性）视图中配置 SPI 端口的比特率、比特顺序等参数。所有配置均已在图形用户界面中完成，因此无需使用正确的数值手动初始化端口。所有操作均由 FSP 配置器完成。瑞萨付出了巨大的努力来创建有用的软件和硬件开发工具，这些工具可用于探索 RA 系列的技术能力，并引领用户从评估阶段一直到生产阶段。

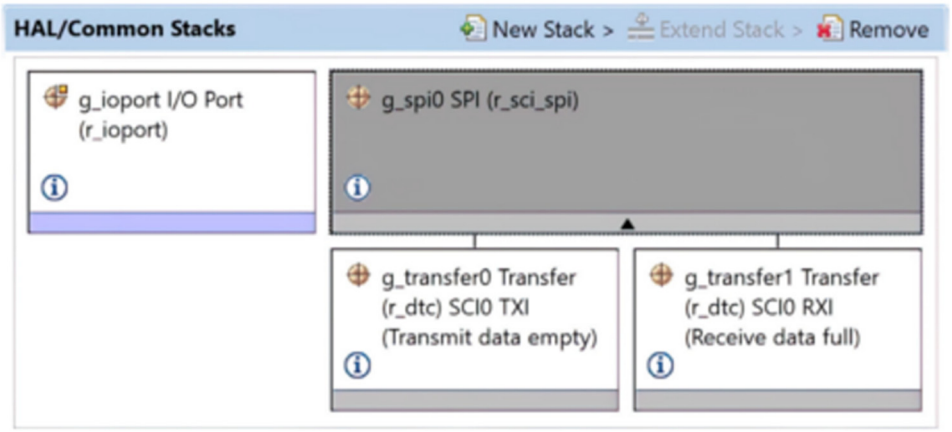


图 3-5：e² studio 中的 SPI 堆

现在，无需关注"*HAL/Common Stacks*"（HAL/通用堆）窗口中的另一项：这是 I/O 端口的板级支持包所需的基本驱动程序。该驱动程序由配置器自动添加并在启动时初始化。

剩下的工作就是检查在 FSP 配置器的 *"Pins "*（引脚）选项卡中，SCI-SPI 的 I/O 引脚是否已正确配置，以便通过此外设实际发送和接收数据。展开视图左侧的"Peripherals"（外设）列表将显示可用的接口。进一步展开 *Connectivity:SCI* 树则可以选择 SCIO 条目，并将显示该端口的 *"Pin Configuration "*（引脚配置）。在

“Operation Mode”（操作模式）下选择 *“Simple SPI”*（简单 SPI）可启用 *TXD_MOSI*、*RXD_MISO*、*SCK* 和 *CTS_RTS_SS* 下拉列表框，在该列表框中，所用引脚可以与实际硬件相匹配。

使用 *<ctrl>+<s>* 保存更改并单击 *“Generate Project Content”*（生成项目内容）图标，将创建必要的源文件和头文件，并将填充控制结构体和配置结构体，名称分别为 `p_ctrl` 和 `p_cfg`，类型分别为 `spi_ctrl_t*` 和 `spi_cfg_t*`，并将它们放置在要包含在用户代码内的应用程序特定头文件中，可以使用 *g_spi0* API 接口提供的指针从头文件中访问这两种结构体。

完成设置并创建 FSP 文件后，便可轻松与接口进行交互。以 SPI 为例，可以调用以下函数来打开和配置 SPI 外设：

```
err = g_spi0.p_api->open(g_spi0.p_ctrl, g_spi0.p_cfg);
```

`g_spi0` 是在“Properties”（属性）视图中配置实例时给定的实例名称，`p_api`、`p_ctrl` 和 `p_cfg` 是指向配置器所生成的结构体的指针。同样，写入 SPI 将需要调用以下函数：

```
err = g_spi0.p_api->write(g_spi0.p_ctrl, tx_buffer, length, bit_width);
```

其中，`tx_buffer` 是指向用户定义数组的指针，该数组存放要通过 SPI 发送的数据，`length` 是要作为 32 位无符号整数写入的单元数量，而 `bit_width` 的类型是 `spi_bit_width_t`，用于存储单元大小。确切语法可以查看 RA 灵活配置软件包文档中的接口参考章节，也可以使用 e² studio 的智能文档功能提取。

也可以不像上述示例那样使用接口函数，而是使用直接实现函数 `R_<MODULE>_<Function>`。在这种情况下，用于打开 SPI 外设并对其进行配置的代码如下所示：

```
err = R_SPI_OPEN(&g_spi_ctrl, &g_spi0_cfg);
```

其中 `g_spi0_ctrl` 和 `g_spi0_cfg` 是由 FSP 配置器创建的数据结构体。同样，写入 SPI 端口将需要调用以下函数：

```
err = R_SPI_WRITE(&g_spi0_ctrl, tx_buffer, length, bit_width);
```

其中，`tx_buffer` 依然是一个指针，所指向的数组存放要通过端口写入的数据，`length` 是单元数量，`bit_width` 发送数据宽度的定义。例如，请使用 FSP 提供的宏 `SPI_BIT_WIDTH_8_BITS` 来代替 `bit_width`。

这两个简短的示例充分证明了灵活配置软件包的功能。在每个示例中，只需编写两行代码即可配置端口并通过该端口发送数据数组。其他内容已由 FSP 的配置器和驱动程序创建并执行。无需通读单片机手册，也无需学习所涉及的不同寄存器，我相信这在应用程序的开发过程中可以节省很多时间。

本章要点：

- FSP 遵循明确的命名约定，简化了编码过程。
- 多个 FSP 模块可以轻松堆叠，为应用程序提供丰富的功能。
- API 语法简单、直观。
- 除了接口函数外，还可以使用实现函数。

4 启动并运行工具链

您将在本章中学到以下内容：

- RA 产品家族工具链的获取位置及安装方法。
- 首次启动 e² studio 及创建新项目的必要步骤。

在本章中，我们将下载并安装瑞萨 RA 产品家族单片机的软件项目所需的两个工具：带 GCC Arm® Embedded 工具链的 e² studio 集成开发环境 (IDE) 和灵活配置软件包 (FSP)。

以下段落概述了所有的必要步骤。您会发现工具的安装过程十分简单，因为选好安装内容和安装位置后，安装程序会自动完成所有步骤。最后，在安装完成后，我们将进行简短的健全性检查，以确保一切正常。

4.1 下载及安装 e² studio 和 FSP

e² studio 包含为 RA 产品家族单片机创建、编译和调试项目所需的所有必要工具。它基于时下流行的 Eclipse™ IDE，但瑞萨在其中加入了一些面向解决方案的组件和插件，使其功能更加强大。配置器尤为如此，它提供了生成新项目的简单方法，并能以图形访问方式轻松访问不同的硬件和软件功能，如引脚配置或添加软件堆，无需深入研究用户手册。这些配置器将自动创建所有必要的设置和初始化代码，其中还加入了错误检查功能，在设计时就能检测出有问题的组合，从而节省大量可能会浪费在编写和/或调试对应用程序并无意义的代码上的时间。如果在我刚开始为嵌入式系统编写软件时就有这样的工具，至少我会特别开心。

接下来的段落将引导您完成在 Windows® 工作站上下载和安装 e² studio 和 FSP 的不同阶段。这些内容适用于 FSP/e² studio 工具链的新手用户。如果您已经安装了最新版本的 e² studio 和 GCC Arm® Embedded 工具，则可以只下载 FSP 包，以缩短这一过程。以上工具可通过访问 FSP 的 GitHub 页面 (https://github.com/renesas/fsp/releases) 的 *“Assets”*（资产）部分获取。提供两个版本：一个名为 *FSP_Packs_<version>.zip* 的压缩文件和一个名为 *FSP_Packs_<version>.exe* 的安装程序。

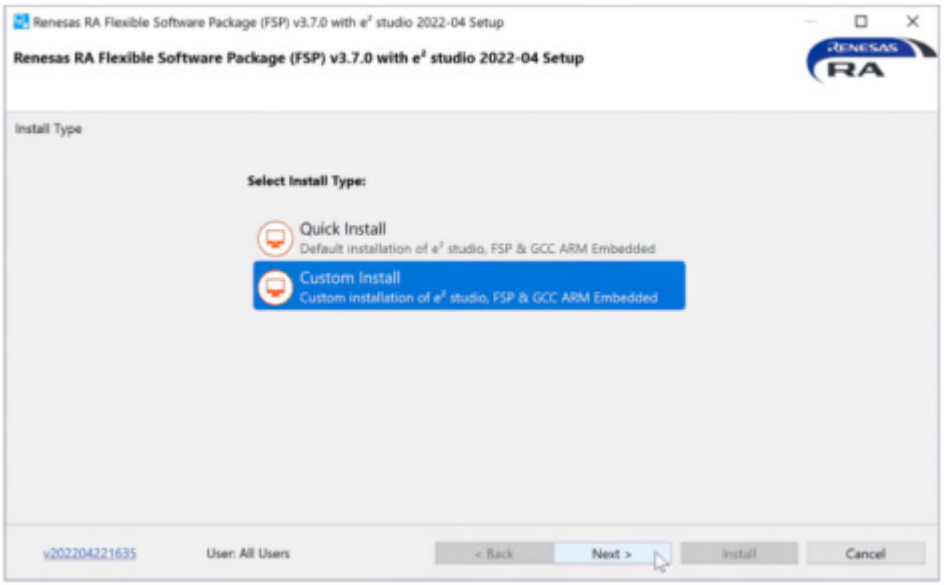
如果您打算使用适用于 Arm 的 IAR Embedded Workbench® 或 Keil® MDK 微控制器开发工具包，也可以利用 FSP 的配置器：RA Smart Configurator (RASC) 与这些工具链集成，并提供相同的功能。该软件也可以在 GitHub 上的 *“Assets”*（资产）部分找到。

4.1.1 下载安装程序

第一步是在 GitHub® (**https://github.com/renesas/fsp/releases**) 上 FSP 页面的 *“Assets”* 资产部分找到*带有 e² studio 安装程序的最新版本 FSP*。在该部分可以找到名为 *setup_fsp_v<fsp_version>_e2s_v<e² studio version>.exe* 的安装程序可执行文件，例如 *setup_fsp_v3_7_0_e2s_v2022-04.exe*。将此文件下载到您的硬盘。该文件相当大，大小超过 1.3 GB，因此在下载过程中，可花些时间查看页面上的其他项目。在该页面上，您也可以下载压缩文件格式的 FSP 最新 HTML 文档，或者阅读 *README.md* 文件中的发行说明。

4.1.2 安装工具链

下载完*捆绑 e² studio 安装程序的 FSP* 后，在硬盘上找到该文件，然后双击以开始安装过程。将安装文件解压需要一些时间。出现的第一个界面会询问您是要为所有用户安装开发工具，还是只为当前用户安装。选择最适合您的安装类型。接下来，会显示 Windows® 的“用户帐户控制”对话框，询问“你要允许此应用对你的设备进行更改吗？”您可以放心地回答 *“是”*。



在下一个界面上，安装程序将询问您安装类型（参见***错误!未找到引用源。***）。有两个选项：“Quick Install”（快速安装），如果选择该选项，则只安装 e² studio、灵活配置软件包、GCC Arm® Embedded 代码生成工具和用户手册及其默认设置，运行时极少需要用户交互。“Custom Install”（自定义安装），如果选择该选择，则不仅允许安装 IDE 和灵活配置软件包，还允许安装一些可选组件，例如 FreeRTOS™ 的调试支持、GIT 版本控制系统的集成或用于实现电容式触摸和 Bluetooth® 低功耗的 QE 工具。如果您想尽快完成安装，则可以选择快速安装。但是，如果您想安装任何其他功能，或者只是想知道还有哪些其他功能可用，可以选择自定义安装。无论您现在如何选择，以后都可以通过重新运行安装程序随时进行更改。

出于指导安装的目的，我们选择 *“Custom Install”*（自定义安装），然后单击 *“Next”*（下一步）。现在，安装程序将检查安装目录是否就绪，以及所需组件是否全部可用。您还可以在此处更改软件的安装文件夹。默认路径为 *C:\Renesas\RA\ e2studio_v<e² studio version>_fsp_v<fsp_version>*，例如 *C:\Renesas\RA\e2studio_v2022-*

04_fsp_V3.7.0。如有必要，可改为您选择的路径。然后单击 *“Next ”*（下一步），退出 *“Results ”*（结果）界面。

出现 *“Extra Features ”*（额外功能）对话框后，完整浏览显示的列表，然后选择要安装的功能。如果您已选好安装项，请单击 *“Next ”*（下一步）继续。

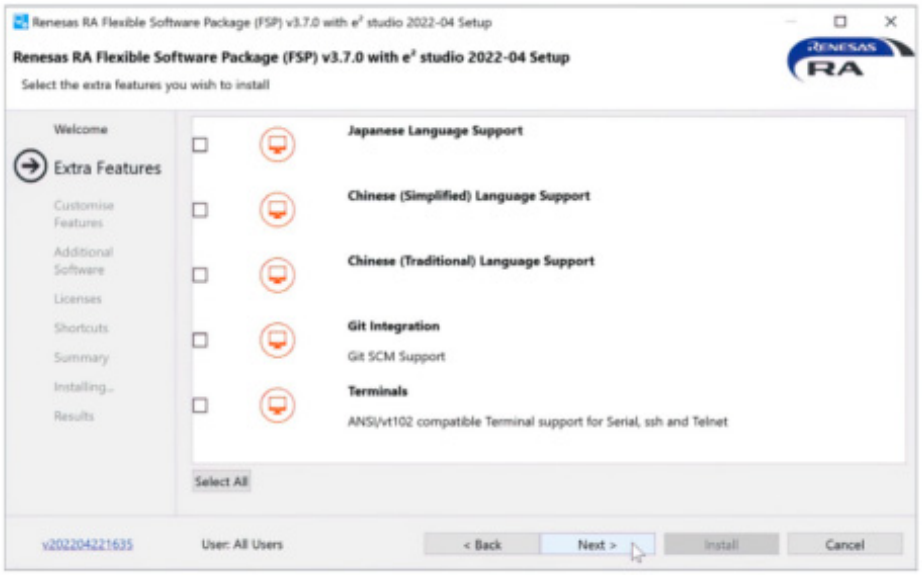


图 4-1：可通过此界面安装其他功能

将显示另一个界面，名为 *“Customise Features ”*（自定义功能）（参见*图 4-2*）。您可以在此界面中添加其他安装组件，例如用于 e² studio 的内核感知插件 WHIS (Wittenstein High Integrity Systems™) StateViewer。该

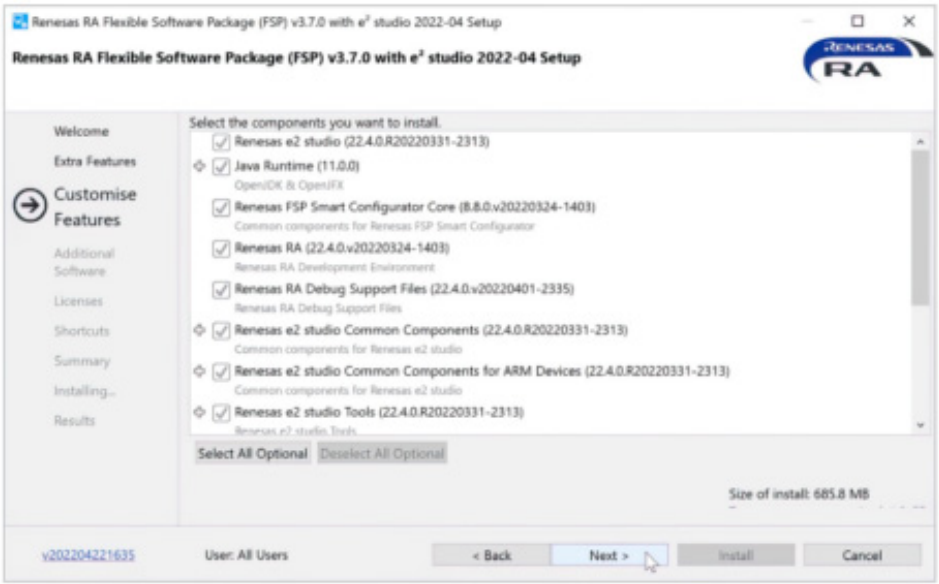


图 4-2：可在该窗口中安装其他组件

插件以表格形式提供 FreeRTOS 和 OpenRTOS 内核数据结构（如任务、队列和信号量）的快照视图，使软件工程师能够更快速、更轻松地开发和调试应用程序。此功能已选中。如果需要则保留，如果不需要，则清除该复选框。完成后，再次单击 *“Next ”*（下一步）。

在随后显示的 *“Additional Software ”*（附加软件）界面上，有两个选项卡（参见*图 4-3*）：一个名为 *Renesas RA*，另一个名为 *Renesas QE*。在 *Renesas RA* 选项卡上，确保在 *“Toolchains ”*（工具链）下选择了 *“GNU*

ARM Embedded”，因为编译和链接代码时需要此工具链。在第二个选项卡中，选择您要安装的 QE 工具。如果您需要有关这些内容的更多信息，请参见第 1 章。准备就绪后，单击 *“Next”*（下一步）。

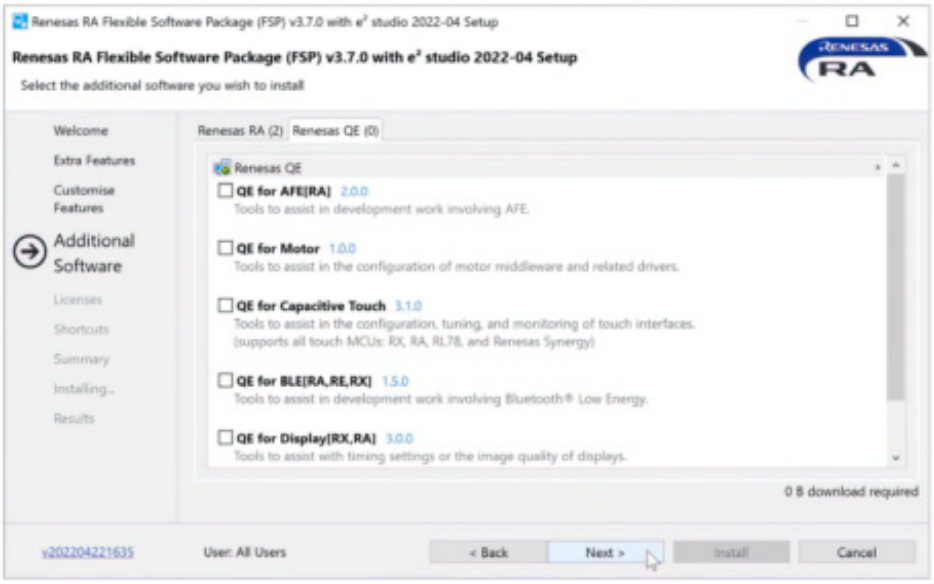


图 4-3：选择您要安装的 QE 工具

在随后出现的窗口中，将显示待安装组件的各种软件协议。仔细阅读每一个协议，然后选中“I accept the terms of the Software Agreements”（我接受软件协议的条款）旁边的复选框，然后再次单击 *“Next”*（下一步）。在出现的 *“Shortcuts”*（快捷方式）界面上，接受建议的“Start”（开始）菜单组，将其改为对您更有意义的名称，或者清除该复选框。单击 *“Next”*（下一步）进入下一个窗口后，安装程序会显示待安装软件的摘要。完整浏览摘要内容，然后单击 *“Install”*（安装）开始安装。

在安装过程中，会出现一个“Windows 安全”对话框，询问“您想安装这个设备软件吗？”（参见*图 4-4*）。选中*始终信任来自“Renesas Electronics Corporation”的软件*旁边的复选框，然后单击*安装*。

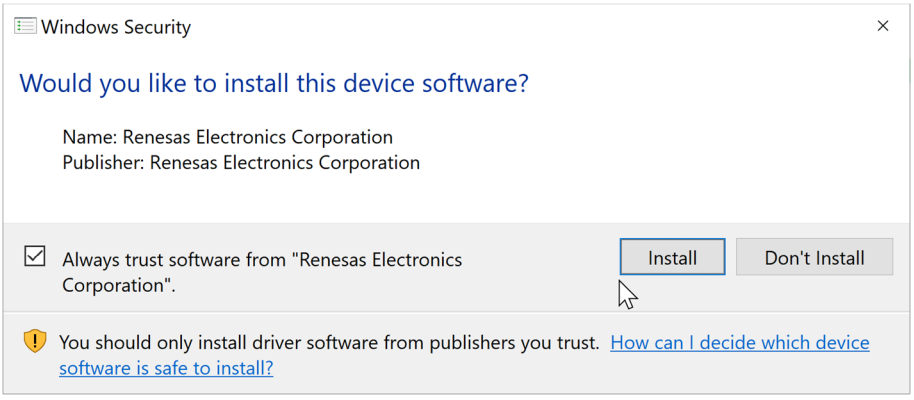


图 4-4：允许安装 Renesas Electronics Corporation 发布的设备驱动程序

安装完成后，将显示 *“Results”*（结果）窗口。清除所有复选框，查看可提供有关 FSP 和 GCC Arm Embedded 的更多信息的链接，以及《FSP 用户手册》的链接。最后，单击 *“OK”*（确定）关闭安装程序。至此便完成安装，现在可以对开发环境进行第一次测试。

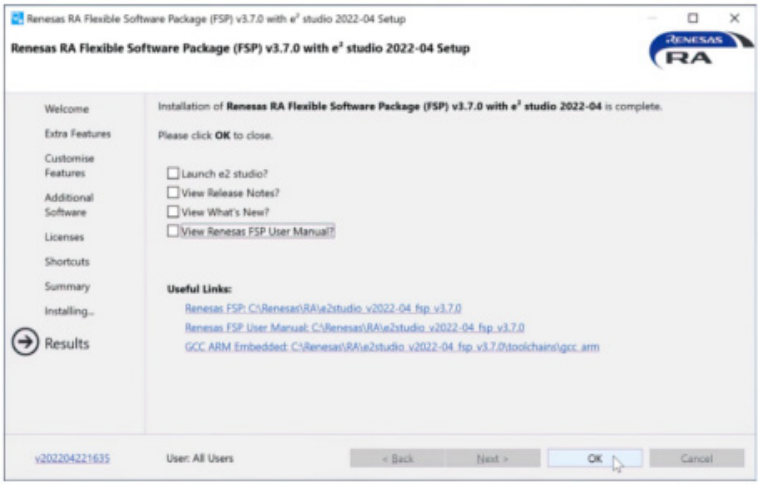


图 4-5：安装完成后，将显示此窗口。清除所有复选框，然后单击“OK”（确定）关闭安装程序。

4.2 首次启动

从 Windows® 工作站的“开始”菜单中打开 e² studio。该软件在 Renesas RA <version number> 菜单组中，如果您在安装过程中更改了快捷方式，则在您输入的菜单组中。如果该软件未在“Start”（开始）菜单中，请查看安装目录 C:\Renesas\RA\e2studio_v<e2 studio version number>_fsp_v<fsp version number>\eclipse。

E² studio 将要求您为工作区建立一个文件夹。您可以接受默认文件夹，也可以自行选择。单击 *“Launch”*（启动）。这是您首次启动集成开发环境 (IDE)，需要将其他支持文件解压并刷新，因此启动时间可能较长。将显示 *“Logging/Reporting”*（记录/报告）对话框。做出决定后，选择 *“Apply”*（应用）。出现 *“Welcome”*（欢迎）界面后，单击界面右上角的 *“Hide”*（隐藏）图标关闭该界面。

下一步是创建第一个简单项目，以确保一切正常。为此，在菜单栏上选择 *“File → New → Renesas C/C++ Project→ Renesas RA”*（文件 → 新建 → C/C++ 项目→瑞萨 RA），然后会显示一个名为 *“Templates for Renesas RA Project”*的新窗口。然后选择 *“Renesas RA C/C++ Project”*（参见*图 4-6*）。单击 *“Next”*（下一步）。之后便会启动项目配置器。输入项目的名称，例如“*MyRaProject*”。

将显示一个新窗口来询问您的“我的瑞萨”登陆凭据。提供给他们并决定是否同意记录和发送使用数据。如果稍后您想改变主意，可以在 e2studio 中更改您的决定。如果您还没有完成“我的瑞萨”的注册，也可以通过这个界面完成注册。

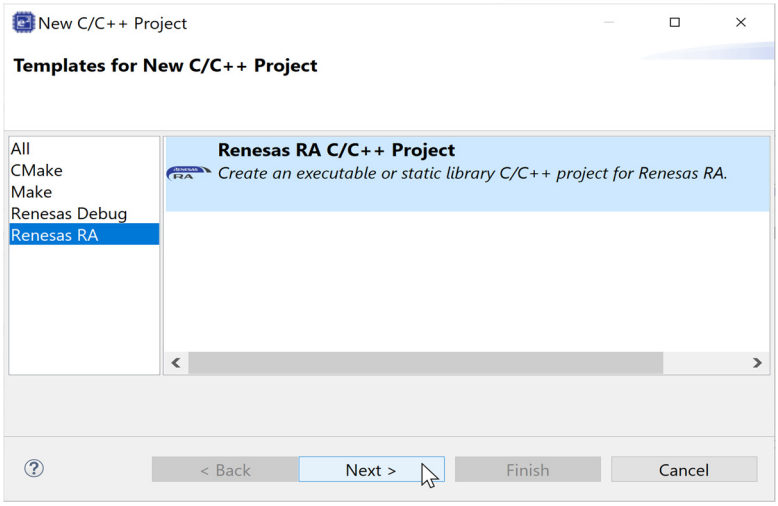


图 4-6：选择“Renesas RA C/C++ Project”（Renesas RA C/C++ 项目）条目，调用 RA 产品家族的项目配置器。

单击 *“Login”*（登陆）进入 *“Device and Tools Selection”*（器件和工具选择）界面。首先，选择一个电路板。我们只是想测试开发环境，因此选择 *EK-RA6M4*，如果未列出，则将相应的器件设置为 *R7FA6M4AF3CFB*。查看工具链：它应显示为 *GNU Arm® Embedded*。单击 *“Next”*（下一步）。

如果已经安装了 GNU 工具，但此处并未显示，则需要手动集成这些工具。为此，转到 e² studio 的“Help”（帮助）菜单，选择 *“Add Renesas Toolchains”*（添加瑞萨工具链）条目。请注意， *“Scan”*（扫描）选项只适用于已经安装在默认位置的工具链。如果尚未在默认位置安装 GNU 工具，请在输入字段中输入自定义文件夹的路径，或使用 *“Add”*（添加）命令和文件浏览器浏览到该位置。

在当前出现的界面中，可以在非 TrustZone® 与安全和非安全 TrustZone 项目之间进行选择。选择 *“Flat (Non-TrustZone) Project”*（扁平化（非 TrustZone）项目）（参见*图 4-7*），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。保持原来的设置不变，即在 *“Build Artifact Selection”*（构建工件选择）下选择 *“Executable”*（可执行），在 *“RTOS Selection”*（RTOS 选择）下选择 *“No RTOS”*（无 RTOS）。不必担心 RTOS 选择：如果需要，稍后可在 RA 配置器中将 FreeRTOS™ 和 Azure RTOS ThreadX® 激活。

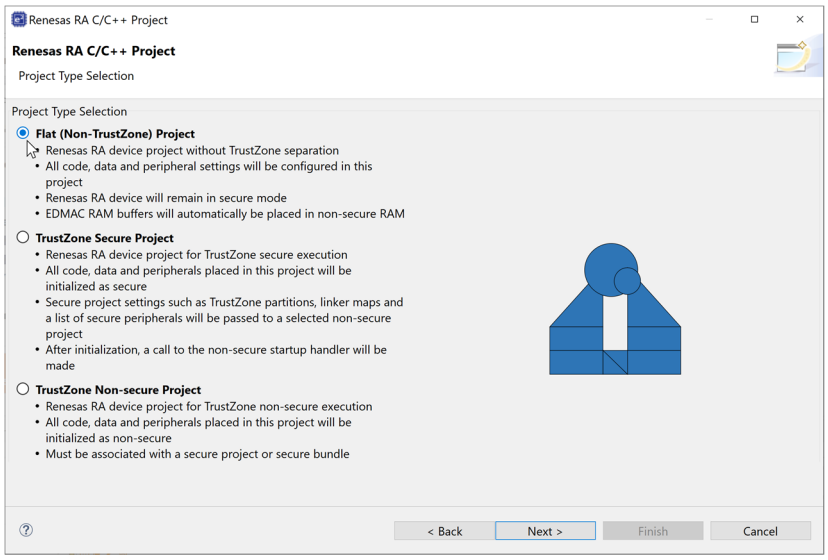


图 4-7：现在，只需让已经选择的扁平化项目保持激活状态即可

单击 *“Next”*（下一步），转到下一个名为 *“Project Template Selection”*（项目模板选择）的界面。对于我们的测试，使用示例软件包即可，因此选择 *“Bare Metal – Blinky”*（裸机 – Blinky）。单击 *“Finish”*（完成）。现在配置器有足够的信息来创建一个项目模板，其中包括正确的器件和基于电路板的板级支持库以及示例项目。

然后，配置器将所有必要文件解压，并询问您是否要打开 FSP 配置透视图。回答 *“Open Perspective”*（打开透视图），随即显示透视图。在配置器中，可以配置 FSP 和处理器的大部分片上硬件。对于我们在本示例中使用的模板，所有内容均已设置完毕，因此请单击窗口右上角的绿色小箭头 *“Generate Project Content”*（生成项目内容），然后配置器将创建所有必要的配置文件并将这些文件添加到项目中。

最后，单击 IDE 顶部工具栏左上角附近的小“锤子”符号，项目将进行编译和链接。完成后，项目应该显示 *“0 errors, 0 warnings”*（0 个错误、0 个警告）的状态。现在已完成所有设置，并且运行正常，您可以开始自己的项目了。

如果在安装过程中出现意外错误，您可以随时获得帮助，甚至可以直接从 e^2 studio 获得帮助：转到 *“Help → Renesas Help→Renesas Helpdesk”*（帮助 →瑞萨帮助 → 瑞萨服务台），然后您将进入瑞萨主页上的 *“Design & Support”*（设计与支持）页面。还可以在“FSP Configuration”（FSP 配置）透视图中的“Summary”（摘要）选项卡上单击“Support”（支持）图标。该操作会在您的浏览器中打开*知识库*页面。

4.3 使安装保持最新状态

安装完工具后，您会希望工具始终都能运行良好，即希望工具保持最新状态。您也可能对基于 Eclipse™ 的开发环境具备的所有可能性兴奋难耐，跃跃欲试，想要安装其他软件。从 **https://github.com/renesas/fsp/releases** 下的 GitHub® 库中下载最新文档和安装文件，即可保持安装的最新状态。不建议使用 e^2 studio 内部的更新功能（ *“Help → Check for updates”*（帮助 → 检查更新），因为内部更新对 RA 产品家族单片机可能毫无益处，而且搜索更新相当费时。

如果您想安装其他软件，可以再次运行安装程序，也可以在 e^2 studio 中转到 *“Help → Install New Software”*（帮助 → 安装新软件）。在随后显示的界面上，从窗口顶部的 *“Work with”*（使用）下面的下拉列表中选择 *“All Available Sites”*（所有可用站点）条目，以搜索所有站点，然后将显示可供下载的软件列表。还可以进入瑞萨网站 (**https://www.renesas.com/ra**) 的 RA 产品家族页面，从中可以找到瑞萨 RA 合作伙伴生态系统提供的其他软件和解决方案。

现在，安装已生效，接下来要查看评估板，并编写您自己的代码。

本章要点：

- 最新版本工具链应从 GitHub® 下载。
- 使用带 e² studio 的 FSP 安装程序时，工具链的安装十分简单。
- 配置器将指导您完成创建项目和配置的过程。

5 使用 e² studio

您将在本章中学到以下内容：

- 透视图、视图和编辑器之间的区别。
- 不同配置器的用途和使用方法。
- 如何导入和导出项目。

我们已详细了解瑞萨灵活配置软件包 (FSP) 及其应用程序编程接口的使用方法，现在，我们可以研究 RA 产品家族单片机的开发环境 e² studio。

e² studio 由瑞萨开发和维护，其依托于 Eclipse™，Eclipse 是一种时下流行且用途广泛的开源集成开发环境，可用于不同的编程语言和目标平台。Eclipse 可以轻松进行定制和扩展，因此成为全球成千上万开发人员的首选 IDE，并且成为了一个事实上的标准。

e² studio 充分利用 Eclipse 的所有优点，并加入了额外的视图和配置器透视图，以支持 RA 产品家族的所有功能。它包含创建、编译和调试任意大小和复杂程度的项目所需的所有工具，并指导开发人员完成软件设计的三个阶段：准备、构建和调试。而且，它会定期更新，从而能够使用最新的 Eclipse SDK 和 CDT 工具。

本章的以下部分将详细介绍 Eclipse 工作台以及如何使用其不同的组成元素。我们将介绍一些基础知识，但不会详细解释所有内容。有关更多信息，请参见《Workbench User Guide》（工作台用户指南），可在 e² studio 中转至*“Help → Help Contents”*（帮助 → 帮助内容）查看该指南。

5.1 Eclipse 体系简介

e² studio 的主窗口称为工作台，由一些基本的用户界面元素构成，如透视图、视图、编辑器、菜单栏和工具栏。

e² studio 启动后，会打开上次使用的透视图。透视图是排列在工作台中的一组视图、编辑器和工具栏。如果重新排列窗口、工具栏或视图，这些排列上的更改将保存在当前的透视图中，下次打开透视图时，排列方式不变。

e² studio 提供多个预定义的透视图，您可以同时看到多个透视图 – 比如 C/C++ 透视图和资源透视图。要从一个透视图切换到另一个透视图，可以单击工具栏最右侧透视图列表旁边的方形图标，或从主菜单栏中选择 *"Window → Perspective → Open Perspective → Other ..."*（窗口 → 透视图 → 打开透视图 → 其他...）。这两种方法都会弹出一个窗口，逐项列出所有可用的透视图。您可以使用相同的菜单条目或右键单击透视图列表中的透视图来关闭透视图。如果右键单击打开的透视图的名称，还会将其分离出来，并在工作台内自由移动。

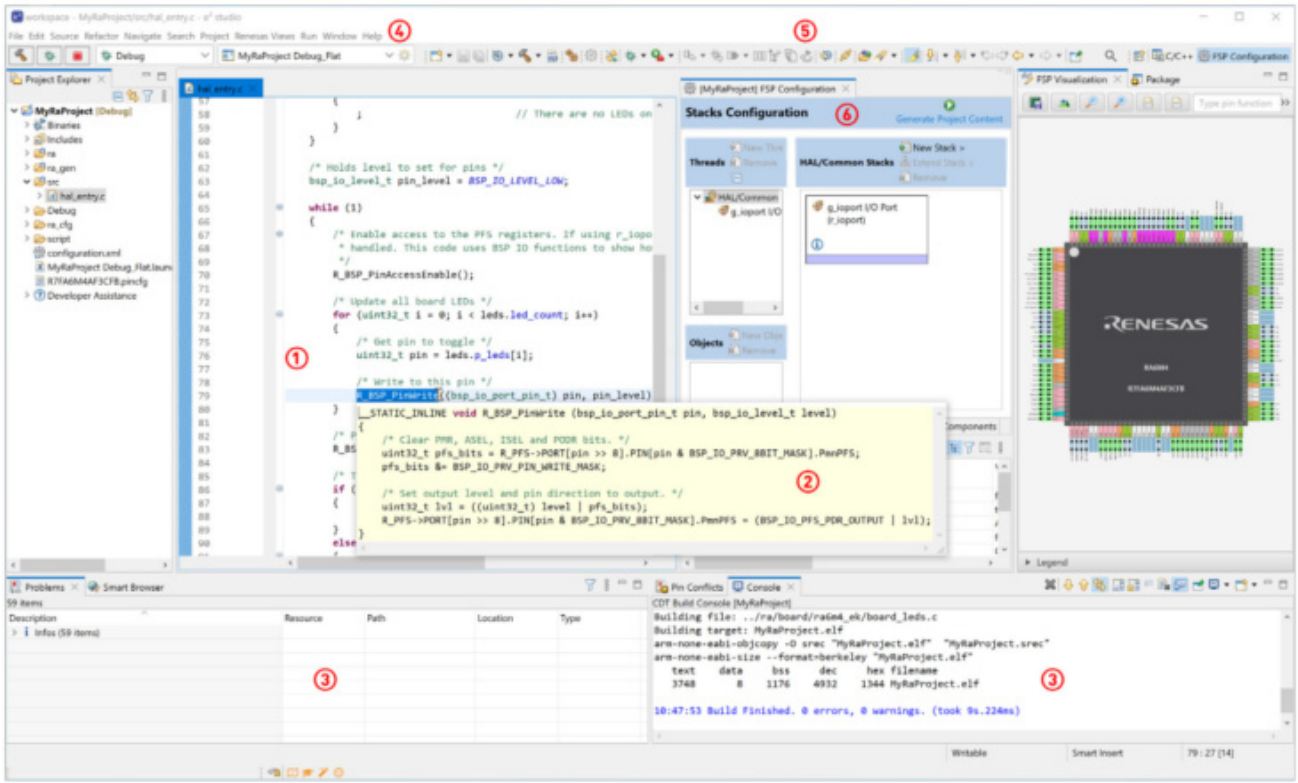


图 5-1： e² studio 的工作台包含编辑器 (1)、智能手册 (2)、视图 (3)、菜单栏 (4)、工具栏 (5) 和透视图 (6)

5.1.1 e2 studio 中的透视图

- **C/C++：**这是 e² studio 的标准透视图，用于开发程序和编辑源代码。默认情况下，它将在中间位置打开编辑器，在左侧打开用于管理项目的“Project Explorer”（项目资源管理器）视图，“Outline”（大纲）视图用于显示编辑器中活动源文件的所有变量和定义，在底部显示选项卡式笔记本，其中有几个叠加的视图，比如“Problems”（问题）（例如任何语法错误或编译错误）、“Tasks”（任务）（列出 Doxygen 任务）或“Properties”（属性）视图。

- **调试：**该透视图用于运行程序以及诊断运行时出现的问题并进行调试。e² studio 将默认打开以下视图：“Debug”（调试）视图（没错，“调试”透视图中包含“调试”视图），调试工具栏、编辑器窗口、大纲视图（与 C/C++ 透视图相同）显示在右侧，一些叠加视图（用于分析和可视化）显示在底部，选项卡式笔记本显示在顶部，用于监视变量、断点、寄存器和其他内容。

- **资源：**默认情况下，此透视图包括 C/C++ 透视图中的大多数视图，但底部的窗口仅显示“Tasks”（任务）视图。该透视图可用来进行常规的资源查看及导航。

- **FSP 配置：**开始时，您需要花费大量时间去了解这个透视图。工作台的最大部分由灵活配置软件包 (FSP) 配置透视图所占据（一个透视图中也可以包含其他透视图），还有几个选项卡式视图，用于各种 FSP 配置器，如时钟配置器或引脚配置器。它还包含“Properties”（属性）视图，用于对不同的外设、线程和驱动程序进行设置；以及“Package”（封装）视图，用来查看为单片机选择的封装的可视轮廓，其中显示当前的引脚分配及状态（说明是否检测到配置问题）。如果需要，也可以从 C/C++ 透视图和调试透视图访问 FSP 配置器。

- **团队同步：**在该透视图中，您可以将您所做的更改与团队其他成员所做的更改合并。

- **跟踪：**该透视图将打开“Statistics”（统计数据）和“Histogram”（柱状图）视图，以及“Debug Call Flow”（调试调用流）视图。

如果您对透视图进行了更改，但不喜欢这些更改，则可以随时在主工具栏中右键单击透视图的名称，选择 *“Reset”*（重置），将其重置为默认设置。也可以用这种方法消除在大量视图和编辑器打开且未停靠的情况下进行大量的编码或调试所导致的混乱情况！屏幕的大小无法让视图排列效果令人满意，所以最终完成工作时，工作台可能会非常杂乱。

5.1.2 视图

视图是一个窗口，可在其中查看信息，例如查看一组寄存器、属性或文件列表。视图还有其他表现形式，例如运行线程的实时图表或连接变量的仪表和控制器。视图可以有自己的菜单栏或工具栏。由视图的菜单栏或工具栏所触发的操作只会影响该视图中的项目，而不会影响其他视图中的项目，即使这些视图位于同一个透视图中也不影响。

多个视图可以在同一个窗口中叠加在一起，也就是所谓的选项卡式笔记本。您也可以将视图从一个笔记本移到另一个笔记本，或者将它们完全取消停靠。这个 e² studio 特性的优点是，您可以根据自己的工作流程来排列所有内容，平台会自动为您保存最后排定的透视图布局。

除了 e² studio 已经提供的诸多视图外，瑞萨还创建了几个额外的视图，如“Memory Usage”（内存使用情况）视图、“Fault Status”（故障状态）视图或“RTOS Resources”（RTOS 资源）视图，从而使集成开发环境的功能更加全面。所有这些视图都可以在主菜单栏的 *“Renesas Views”*（瑞萨视图）下找到。

5.1.3 编辑器

这个视图将占用每个软件开发人员的大部分时间（编辑器被视为一种特殊类型的视图）。除了代码补全或关键字突出显示等常见和预期功能外，e² studio 内置的编辑器还包括一个特殊的功能，称为“智能手册”。借助智能手册，您无需再去研究数千页的文档，因为您可以在编辑器视图中直接获得关于 FSP 和单片机本身的上下文相关帮助。

将鼠标悬停在 e² studio 中的任何关键字、函数、变量或宏上，将出现一个显示相关信息的窗口。对于变量，将显示声明，对于结构体和枚举，将显示所有成员，而对于函数，将显示描述、原型和参数详细信息。如果变量与单片机的寄存器关联，还会显示有关位定义的特定信息。智能手册甚至会提取适用的应用笔记和丰富的媒体说明材料（如果有），并在 C/C++ 透视图底部的智能浏览器视图中显示这些内容。

这一强大功能可以节省大量时间，因为您不必在 FSP 及 RA 单片机手册与 e² studio 之间来回切换。您可以在最需要的地方获得相关信息：将光标悬停在此处即可。

```
/* Get pin to toggle */
uint32_t pin = leds.p_leds[i];

/* Write to this pin */
R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);
```

```
__STATIC_INLINE void R_BSP_PinWrite (bsp_io_port_pin_t pin, bsp_io_level_t level)
{
    /* Clear PMR, ASEL, ISEL and PODR bits. */
    uint32_t pfs_bits = R_PFS->PORT[pin >> 8].PIN[pin & BSP_IO_PRV_8BIT_MASK].PmnPFS;
    pfs_bits &= BSP_IO_PRV_PIN_WRITE_MASK;

    /* Set output level and pin direction to output. */
    uint32_t lvl = ((uint32_t) level | pfs_bits);
    R_PFS->PORT[pin >> 8].PIN[pin & BSP_IO_PRV_8BIT_MASK].PmnPFS = (BSP_IO_PFS_PDR_OUTPUT | lvl);
}
```

图 5-2： e² studio 的智能手册功能会恰好在需要的地方显示信息。

代码补全是 IDE 的另一项功能，可在代码开发过程中节省大量的时间。在变量后面按 *<ctrl>-<space>* 将显示一个包含可用选项（例如 API 结构体的组成元素）的窗口。单击其中一个成员可将其插入代码中。同样，无需翻阅手册，只需光标指向此处并单击即可！

5.2 配置器：简介

e² studio 内的配置器以图形方式指导软件开发人员了解 RA 产品家族单片机 (MCU) 的特定器件选项，并提出使用建议。例如引脚选择或冲突警告。配置器还可生成启动代码或将 FSP 软件组件放置在项目中。

5.2.1 项目配置器

有多个配置器可以帮助到设计人员，大多数用户最先用到的是项目配置器，它将引导设计人员从头开始创建新项目，或基于配置器提供的模板创建新项目。利用该配置器可以选择项目的大部分设置，例如要使用的工具链、器件和电路板，或者是否应生成一个示例项目。

在这个过程结束时，会在 e² studio 的工作台中自动生成项目，并会在项目中添加所有必要的文件。无需手动创建任何内容；无需查找生成文件的参数和设置，无需研究要包括哪些头文件，以及如何建议编译器使用大量 RA 产品家族单片机中的特定系列和器件。您在最后一个配置器界面上单击 *“Finish ”*（完成）后，即完成了所有工作。

代码生成完成后，项目配置器将切换到“FSP Configuration”（FSP 配置）透视图，可在其中对不同的 FSP 组件进行设置。

5.2.2 FSP 配置器

FSP 配置器以透视图形式出现后，它将显示项目的只读摘要，说明所选的电路板和器件以及所使用的工具链。还会列出项目中的各种软件组件及其版本号。在页面底部，通过快捷方式可以快速访问 YouTube™ 上的瑞萨 FSP 播放列表，其中包含多个关于 RA 产品家族的短视频，还可快速访问 Renesas.com 上的 Renesas FSP 支持页面，以及灵活配置软件包的用户手册（参见*图 5-3*）。

配置器底部的下一个选项卡是 BSP（板级支持包）选项卡，可用来查看和编辑各项电路板设置，如器件或电路板选择。在关联的“Properties”（属性）视图中，可以为 BSP 进行其他设置，例如主堆栈的大小（在线程上下文外部使用的堆栈）或 MCU 的某些安全功能。

图 5-3：FSP 配置器的“Summary”（摘要）选项卡

下一个选项卡名为“Clocks”（时钟），用于分配初始时钟配置。提供了片上时钟系统的图形表示（参见*图 5-4*），并可以对时钟树进行修改。将鼠标指针悬停在相应项目上会显示对应的简短描述。如果设置了不兼容的项，相应成员将以红色突出显示，并会提供问题说明。此外，选项卡本身会显示一个小感叹号，表明存在问题。

“Pins”（引脚）选项卡涵盖了 RA MCU 的初始引脚设置。可以根据端口或外设列出引脚。如果设置冲突或不完整，则配置器右侧的“Package”（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。

下一个选项卡是“Interrupts”（中断）选项卡，用来指定用户定义的（非 FSP）驱动程序如何在 RA 项目内使用单片机中断控制器单元 (ICU)，以及将哪些中断服务程序 (ISR) 与 ICU 事件（中断）相关联。它还提供所有分配的 ICU 事件的完整列表。该表包括由 FSP 模块实例生成的事件，这些实例也已在配置器的“Stacks”（堆）视图中创建。

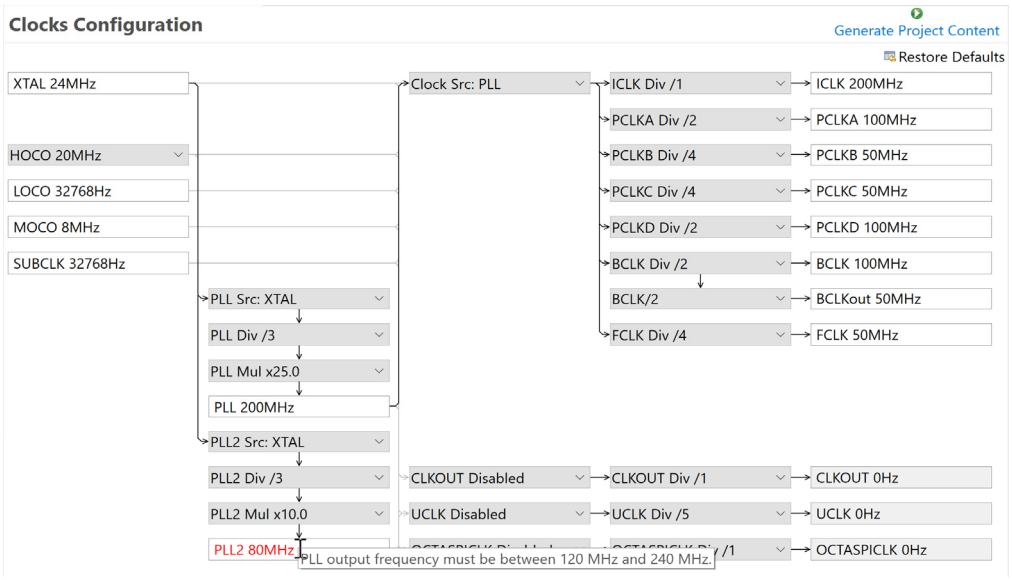


图 5-4：FSP 配置器的时钟视图，其中包含一个有意引入的错误

“Event Links”（事件链接）选项卡的作用与“Interrupts”（中断）选项卡类似。用户可以在此处指定其驱动程序如何在 RA 项目内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。

下一个选项卡是“Stacks”（堆）选项卡，可用来在 RA 项目内添加及配置线程和堆。可以在各个堆中添加不同的模块和对象，并可以在“Properties”（属性）视图中修改其属性。“Stacks”（堆）视图显示不同线程的堆，从而创建模块的图形配置。可以轻松添加新的堆，并自动插入所有必要的模块，直到达到需要开发人员干预的程度。达到这种程度时，如果将鼠标悬停在模块上，模块会被标为红色，并会提供所需设置或问题的说明。如果问题解决，模块将恢复为标准颜色。

“FSP Configuration”（FSP 配置）透视图中的最后一个选项卡名为“Components”（组件），用户可在其中显示及选择不同的 FSP 组件。还列出了可用的 RA CMSIS 软件组件。最好在“Stacks”（堆）选项卡中执行修改（例如在当前项目中添加或删除模块），因为还可以在后面的选项卡中对修改进行配置。

在“FSP Configuration”（FSP 配置）中进行了所有的设置后，就可以从 FSP 生成或提取相关的源代码并将其添加到项目中。单击透视图顶部的 *“Generate Project Content”*（生成项目内容）符号即可完成这一过程。如果忘记单击这个符号，也不必担心！如果检测到配置或生成的文件有变化，会自动触发这个动作。

FSP 配置器是一款非常出色的工具，它会指导您完成准备项目并进行初始设置所需的所有步骤。该配置器与项目配置器一样，不需要细读数千页的文档并进行深入研究，因为它能提供摘要级别的必要信息，并确保所有设置都正确、合理、一致。如果您还记得您的上一个项目，那时您还没有使用这类配置器，单单是正确设置所有引脚布线和中断，以及处理大量相互依赖的硬件寄存器，就让您花费了大量时间，那么现在，您一定会对瑞萨为这些工具的付出而感激不尽。使用这些工具减少了很多麻烦！

5.3 导入、导出和使用项目

有时，您需要导入或导出项目。也许您想使用瑞萨网站上诸多示例项目中的一个，或者需要与某位同行分享最新开发成果。无论想要导入还是导出，都可以在 e² studio 内使用导入向导或 FSP 导出向导方便地完成。

5.3.1 导入项目

有两种方法可以调用导入向导：在“Project Explorer”（项目资源管理器）视图中单击鼠标右键，然后从显示的菜单中选择 *“Import”*（导入），即可打开向导，另一种方法是转至菜单栏，然后选择 *“File → Import”*（文件 → 导入）。无论用哪种方法，都要在导入向导窗口显示出来后，展开 *“General”*（常规）条目，然后选择 *“Existing Projects into Workspace”*（将现有项目导入工作区）或 *“Rename & Import Existing C/C++ Project into Workspace”*（将现有 C/C++ 项目重命名并导入工作区）。如果选择后者，则可以为导入的项目输入新名称。

无论是哪种情况，均可以选择待导入项目所在的源目录，或者选择存档文件。如果文件夹中有一个或多个项目，则会将它们全部标记为要导入。取消选择您不想导入的项目。单击 *“Finish”*（完成）后，项目会导入到工作区中，如果选中 *“Copy projects into workspace”*（将项目复制到工作区）旁边的复选框，则会将项目复制到工作区。导入的项目需重新进行编译才能运行。

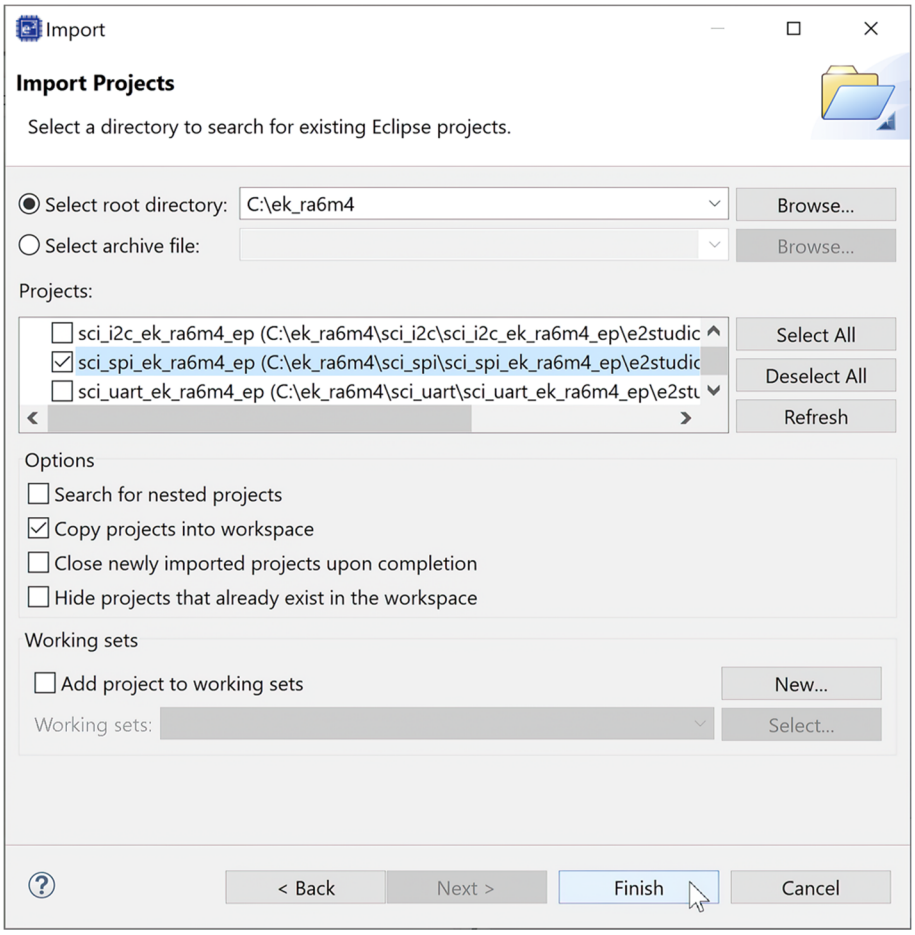


图 5-5：列出了在指定位置找到的项目以进行导入

5.3.2 导出项目

导出项目和导入项目一样简单。同样，有两个方法可以完成导出。第一个方法是转至 e² studio 的菜单栏，选择 *“File → Export”*（文件 → 导出）。第二个方法是右键单击要导出的项目，然后选择 *“Export FSP Project”*（导出 FSP 项目）。如果使用第一个方法，则必须在显示的导出窗口中执行两个附加步骤：第一步是展开 *“General”*（常规）条目，第二步是在显示的列表中选择 *“Renesas FSP Archive File”*（瑞萨 FSP 存档文件）条目。这两个方法都会弹出 *“FSP Export Wizard”*（FSP 导出向导）。

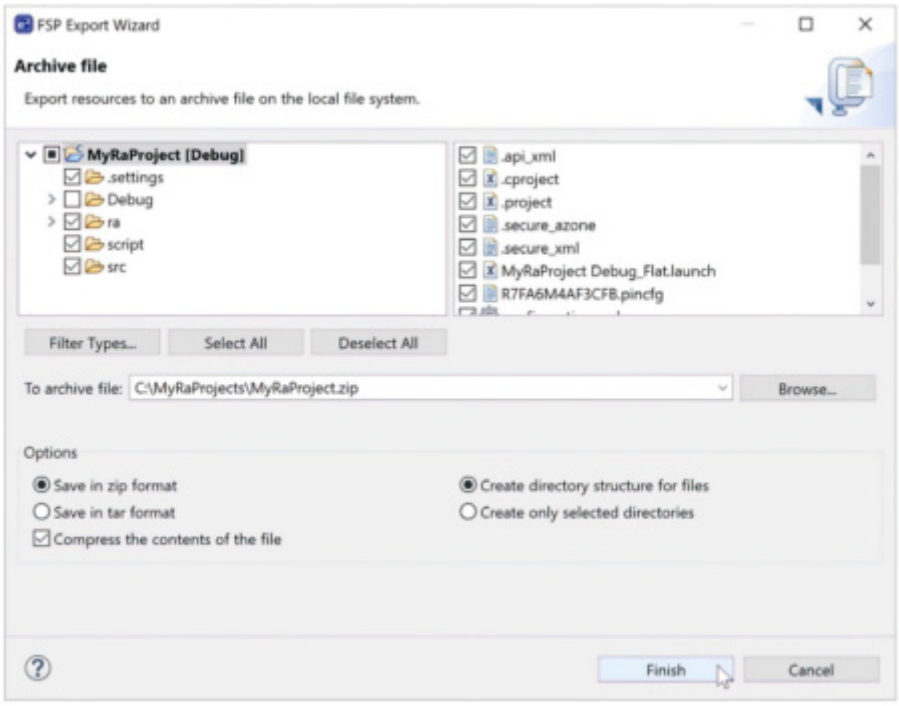


图 5-6： e² studio 中的 FSP 导出向导

FSP 导出向导显示出来后，可以选择要导出的项目和文件，以及压缩格式和所需的目录结构。单击 *“Finish”*（完成），结束导出过程。

注意，默认情况下，不会选中 *“Debug”*（调试）和 *“ra”* 条目。这些条目无需包含在导出中，因为包含这些信息的 configuration.xml 文件是导出的一部分，缺失的内容可以通过打开文件（将启动 FSP 配置器）并再次单击 *“Create Project Content”*（创建项目内容）来重新创建。这会恢复导出时遗漏的文件。以这种方式导出的项目可以在其他工作站上方便地共享和导入。

本章要点：

- e² studio 提供不同的透视图和视图，从而将功能分组。
- 智能手册功能、项目配置器和 FSP 配置器可加快开发速度并减少错误。
- 使用相应的向导可以轻松导入和导出项目。

6 RA 产品家族的硬件评估板套件

您将在本章中学到以下内容：

- RA 产品家族有哪些不同的开发工具包以及它们的主要功能。

在每个开发项目中，都有一个需要通过硬件来运行第一个测试的时间点。而且，所有工程师都知道，在自己的电路板即将投入使用之前，情况几乎总是如此。或者，硬件开发者可能不愿把为数不多的原型之一借给他们亲爱的软件部同事，因为他们可能会毁掉电路板（我从未遇到过这种情况！）。

为此，瑞萨提出了一种解决方案：这种方案提供各种易于使用的评估板套件，可帮助软件设计人员简单省力地开始使用平台，因为他们可以立即测试其程序。而且，硬件设计人员也将受益于这些电路板，因为他们可以通过评估板的不同连接器方便地访问单片机 (MCU) 的大多数引脚。如有必要，他们还可以通过 Digilent Pmod™、SparkFun® Qwiic®、SeeedGrove® 系统、Arduino™ (Uno R3) 连接器或 MikroElektronika™ mikroBUS 连接器等标准化接口使用流行的生态系统插件来丰富电路板的功能。这些第三方模块既可以单独使用，也可以同时使用。

各 MCU 系列中的每个超集器件（存储空间最大且片上外设最多的器件）均提供一个评估板 (EK)，可以方便地入门 RA 产品家族单片机，并且在稍后需要较小的存储器或较少的外设时，也可以平稳切换到 MCU 系列中的其他器件。所有 EK 的电路板上均提供了通用的基础系列功能，并且具有足够高的灵活性，可以在 PCB 上集成特殊功能。所有评估板均带有预编程的快速入门示例项目，您可以使用它们来熟悉硬件。瑞萨还提供了其他示例代码，工程师可将这些代码用作更复杂的自定义嵌入式应用程序的构建模块。

所有评估板的构建方式均可使用户的学习曲线保持平缓。它们为 RA 产品家族单片机的评估、原型设计和开发提供了价格实惠的起点。

从物理上看，电路板包括两个不同的部分：第一部分是系统控制和生态系统访问区域，包括板上调试、生态系统扩展、用户 I/O、串行连接、电源调节、复位和引导配置。这一部分是基于该架构的所有电路板上的标准配置。

第二部分是每个评估板特有的，包括特殊功能访问区域和原生 MCU 引脚访问区域。特殊功能访问区域包括为每个评估板提供独有功能的外设（例如以太网、Octo-SPI、Quad-SPI 和 USB 高速功能）。MCU 原生引脚访问区域包含单片机，并允许通过公头引脚插针直接访问 MCU 端子，也可通过钳位环实现电流测量。

所有评估板中均提供了一些常用功能：

- 电源
- 调试接口
- 串行接口
- 用户 LED
- 有线连接
- 单片机电流测量测试点
- 访问大多数 MCU 数字和模拟引脚。

除评估板外，瑞萨还提供其他电路板和系统板。它们可满足电容式触摸、电机控制或图形等特定应用的特殊需求，并且包含一个或多个专门针对其所服务的应用空间定制的电路板。所有评估板套件的共同之处它们均具有模块化特性和可扩展性，同时支持以简单且防错的方式进行配置。

每个评估板都有自己的网站，URL 遵循语法 reneasas.com/<评估板名称>。因此，可通过在网络浏览器中输入 **renesas.com/ek-ra6m4** 访问实验使用的 EK-RA6M4 页面。在此页面上，不仅可以查看每个电路板的详细信息，还可以下载评估板的用户手册、快速入门指南、白皮书和示例项目等。此外，还可以下载完整的“设计包”，其中包括原理图、物料清单、3D 和机械图纸以及可以用作您自己的开发基础的制造和设计文件。

最后但同样重要的一点是，评估板符合多种国际标准，例如 FCC Notice – 第 15 部分和 CE Class A（适用于 EMC/EMI）、欧盟 RoHS 和中国 SJ/T 113642014（适用于废物、回收和材料选择）或 UL 94V-0（适用于安全）等。

6.1 EK-RA6M4 评估板套件

EK-RA6M4 是用于 RA6M4 MCU 系列单片机 (MCU) 的评估板套件。它是围绕 R7FA6M4AF3CFB MCU 构建的，其中 MCU 基于支持 TrustZone® 的 200 MHz Arm® Cortex®-M33 内核，采用 144 引脚 LQFP 封装。它配有 1 MB 的片上代码闪存和 256 kB 的内部 SRAM。凭借此评估板，用户能够轻松评估 RA6M4 MCU 系列的功能并开发复杂的嵌入式系统应用程序。

多个时钟源提供 24.000 MHz 和 32,768 Hz 的高精度参考时钟。RA 单片机内部可访问额外的低精度时钟。电流测量点也位于 PCB 上，作用是准确测量电流消耗。

可通过 USB 全速主机和设备端口以及以太网连接器与外界建立连接。通过 4 个 40 引脚公头插针以及 5 个最流行第三方生态系统的扩展连接器可以访问单片机的大多数功能。通过板上 J-Link® 调试器可以轻松调试软件和硬件。灵活配置软件包 (FSP) 和 e² studio IDE 完全支持该电路板。

评估板的其他功能包括：

- 存储器：
  - 片上：1 MB 代码闪存及 256 kB SRAM
  - 片外：64 MB 八通道 SPI 闪存及 32 MB 四通道 SPI 闪存
- 有线连接：
  - USB（1 个调试，1 个 USB 全速主机和设备，micro-AB 连接器）
  - 带 RMII 的以太网（RJ45 连接器）
- 调试模式：
  - 板上调试 (SWD)
  - 外部调试接口（ETM、SWD 和 JTAG）
  - 调试功能输出 (SWD)
- 用户交互：
  - 2 个用户按钮
  - 复位按钮
  - 3 个用户 LED
  - 电源 LED
  - 调试 LED
- 扩展：
  - 4 个 40 引脚公头插针，用于访问 MCU 引脚
  - 2 个 SeedGrove® 扩展连接器（I2C/模拟）
  - SparkFun® Qwiic® 连接器
  - 2 个 Digilent Pmod™ 连接器（SPI 和 UART）
  - Arduino™ Uno R3 连接器
  - MikroElektronika™ microbus 连接器

图 6-1：EK-RA6M4 评估板套件

瑞萨网站上提供了有关该评估板套件的一套丰富文档：快速入门指南（引导用户初步了解评估板）、用户手册（提供电路板的所有技术细节）和设计文件包（包含评估板原理图、BOM、Gerber 文件等）。由于这是我们将用于练习的电路板，您可能需要下载用户手册，以便在遇到问题时参考。可以从同一网页访问多个示例项目，完成本手册中的实验后，您可能想要探究这些示例项目。

6.2 带图形扩展板的 EK-RA6M3G 评估板套件

如果涉及功能强大的人机界面 (HMI) 或彩色显示屏，则 RA 产品家族单片机中的 RA6M3 系列是一个不错的选择。为了评估其用于 TFT 的片上图形 LCD 控制器和其 2D 图形引擎，瑞萨提供了 EK-RA6M3G 图形评估板套件。这样，工程师便可无缝评估 RA6M3 MCU 系列的功能，并利用灵活配置软件包 (FSP) 和 e^2 studio IDE 开发基于自有图形的嵌入式系统应用程序。这款评估板是围绕 R7FA6M3AH3CFC MCU 构建的，其中 MCU 基于 120 MHz Arm® Cortex®-M4 内核，采用 176 引脚 LQFP 封装。它配有 2 MB 的片上代码闪存和 640 kB 的内部 SRAM。

EK-RA6M3G 评估板包含两个电路板：带 MCU 的 EK-RA6M3 电路板和带 4.3 英寸 TFT 彩色 LCD（带有电容式触摸屏）的图形扩展板。为了驱动 LCD 上的 LED 背光，图形扩展板还包括一个背光控制器。

可通过全速和高速 USB 端口以及以太网连接器与外部环境建立连接。通过 4 个 40 引脚公头插针以及 4 个最流行第三方生态系统的扩展连接器可以访问单片机的各个功能。通过板上 J-Link® 调试器可以轻松调试软件和硬件。

多个时钟源提供 24.000 MHz 和 32,768 Hz 的高精度参考时钟。RA 单片机内部可访问额外的低精度时钟。电流测量点也位于 PCB 上，作用是详细测量 MCU 和 USB 端口的电流消耗。

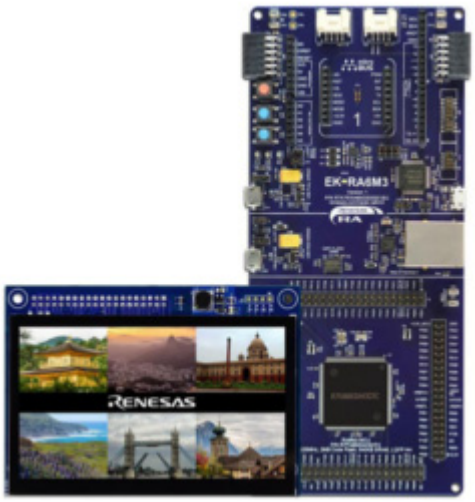


图 6-2：带图形扩展板的 EK-RA6M3G 评估板

评估板的其他功能包括：

- 存储器：
  - 片上：2 MB 代码闪存及 640 kB SRAM
  - 片外：32 MB 四通道 SPI 闪存
- 有线连接：
  - 3 个 USB（1 个调试，1 个 USB 全速，1 个 USB 高速）
  - 带 RMII 的以太网（RJ45 连接器）
- 调试模式：
  - 板上调试 (SWD)
  - 外部调试接口（ETM、SWD 和 JTAG）
  - 调试功能输出 (SWD)
- 用户交互：
  - 2 个用户按钮
  - 复位按钮
  - 3 个用户 LED
  - 电源 LED
  - 调试 LED
- 扩展：
  - 4 个 40 引脚公头插针，用于访问 MCU 引脚
  - 2 个 SeedGrove® 扩展连接器 (I2C)
  - 2 个 Digilent Pmod™ 连接器（SPI 和 UART）
  - Arduino™ Uno R3 连接器
  - MikroElektronika™ microbus 连接器
- 图形扩展板：
  - 4.3 英寸 TFT 彩色 LCD，分辨率为 480 x 272 像素，采用带控制器的电容式触摸屏
  - 背光控制器

瑞萨网站上提供了有关该评估板套件的一套丰富文档：白皮书、应用笔记、快速入门指南（引导用户初步了解评估板）、用户手册（提供评估板的所有技术细节）和设计文件包（包含电路板原理图、BOM、Gerber 文件等）。图形元素的开发由 Segger™ emWin 嵌入式 GUI 库和随附的运行在主机系统上的 AppWizard 提供支持，用于为 emWin 库创建可立即运行的应用程序。借助它们，开发功能强大的 HMI 变得轻而易举。

6.3 RA6T1 电机控制评估系统

适合瑞萨 RA 产品家族单片机 (MCU) 的 RA6T1 系列的电机控制评估系统提供了一种低压永磁同步电机（无刷直流电机）解决方案。它将用于评估电机控制系统的所有必要硬件捆绑在一起，构成一个易于使用的评估套件。这样一来，用户便可轻松进行评估，从而高效开发此类系统。

这一综合解决方案支持永磁同步电机、三电阻电流检测，提供过流保护，并可通过 USB 端口与 Renesas Motor Workbench 通信。Motor Workbench 允许自动调整控制系统的参数，以及对该系统进行实时监测。此评估套件本身由一个 RA6T1 CPU 板卡、一个 48 V 逆变器板、一个永磁同步电机和必要的线缆组成。通过使用另一个瑞萨 CPU 板卡替换随附的 CPU 板卡，可以使用另一个 MCU 评估电机控制系统。

图 6-3：Motor Control Workbench 允许自动调整参数和监测运行系统

此评估套件是围绕 120 MHz R7FA6T1AD3CFP MCU 构建的，其中 MCU 配有 512 kB 的片上闪存和 64 kB 的 SRAM。此 MCU 基于 Arm® Cortex®-M4 内核，具有高分辨率 PWM 定时器、12 位 A/D 转换器、高速模拟比较器和各种接口。

评估板套件的其他功能包括：

- RA6T1 CPU 卡：
  - R7FA6T1AD3CFP MCU
  - 板对板连接器
  - 板上 J-Link™ 的 USB 连接器
  - 用于 Renesas Motor Workbench 通信的 SCI 连接器
  - 用于下列各项的连接器：
    - *CAN 通信*
    - *SPI 通信*
    - *霍尔传感器信号输入*
    - *编码器信号输入*
    - *用于 Arm® 调试器的 10/20 引脚通孔*
  - 用于第二个逆变器的通孔
  - 2 个用户控制 LED
  - MCU 外部复位开关
- 逆变器板：
  - 拨动开关
  - 按钮开关
  - 3 个用户 LED
  - 逆变器控制电路块的电源 LED
  - 板对板连接器
  - USB 连接器
  - 电机连接器
  - 可变电阻

图 6-4：RA6T1 电机控制评估系统

此评估套件随附预编程的软件，可帮助您轻松入门。要使电机旋转，只需组装硬件元件即可。与其他系统一样，该评估套件的网站上提供了一套完整的文档：白皮书、应用笔记、快速入门指南（引导用户初步了解评估套件）、用户手册（提供电路板的所有技术细节）和设计文件包（包含电路板原理图、BOM、Gerber 文件等）。此外，还可以下载示例代码，以帮助开发人员在设计方案时占尽先机。

6.4 EK-RA/6M5 评估套件

图 6-5：EK-RA6M5 开发套件

对于 IoT 应用，其要求增强的安全性，以太网或者 CAN 总线的连接，并且需要容量大和嵌入型的存储器，瑞萨提供了 RA6M5 系列控制器。提供 EK-RA6M5 开发套件来支持客户的设计。它是围绕基于 176 引脚 LQFP 封装中 200 MHz Arm®Cortex®-M33 内核 R7FA6M5BH3CFC MCU 构建的。该处理器拥有 2MB 片上闪存存储代码和 512KB 内部 SRAM。它的功耗很低，当从 Flash 执行 CorkMark®算法的时候可以达到 107uA/MHz。

内核嵌入的 Arm®TrustZone® 安全加密引擎，使 RA6M5 系列 MCU 适用于火灾和防盗检测或面板控制等安全应用。受益于安全功能和丰富连接，其他适用的领域有电表和自动抄表设备，以及机器人、自动售货机或 UPS（不间断电源）等工业应用。

EK-RA6M5 有丰富的接口和外部世界连接：一个全速 USB 和一个高速的可以做主机和从机设备 USB，一个以太网接口和一个符合 CAN FD(灵活数据速率)标准的 CAN（控制器区域网络） 总线。CAN FD 延伸了原生 CAN 总线标准，并且允许更好的数据传输速率和更大的有效载荷。

由于 EK 开发套件集成了板载的 J-Link® 调试器，让调试软件和硬件都变得很容易。可以通过测量点测量精确的电流消耗情况。外部 24.000MHz 和 32768Hz 的参考时钟提供一个精确的时钟基准。RA6M5 同样也提供了额外的内部低精度时钟。

- 存储器：
  - 片上：2 MB 代码闪存及 512 kB SRAM
  - 片外：64 MB 八通道 SPI 闪存及 32 MB 四通道 SPI 闪存
- 有线连接：
  - 3 个 USB（1 个调试，1 个 全速 USB，1 个 高速 USB ）
  - 以太网（RMII，PHY, RJ45 连接器）
- 调试模式：
  - 板上调试 (SWD)
  - 外部调试接口（ETM、SWD 和 JTAG）
  - 调试功能输出 (SWD)
- 扩展：
  - 4 个 40 引脚公头插针，用于访问 MCU 引脚
  - 2 个 SeedGrove® 扩展连接器（I2C/模拟）
  - 2 个 Digilent Pmod™ 连接器（SPI 和 UART）
  - SparkFun® Qwiic® 连接器
  - Arduino™ Uno R3 连接器
  - MikroElektronika™ microbus 连接器
- 用户交互：
  - 3 个用户按钮
  - 复位按钮
  - 3 个用户 LED
  - 电源 LED
  - 调试 LED
- 与其他评估开发套件一样，提供了用户手册、应用说明和其他文档以及示例项目。开发人员还可以从评估开发套件的网站下载包含原理图、设计文件、材料清单和机械图纸的完整设计包。

7 首次使用瑞萨 EK-RA6M4 评估板

您将在本章中学到以下内容：

- 如何将 Renesas 评估板连接到您的工作站。
- 什么是调试配置以及如何创建它。
- 如何在集成开发环境中下载并启动程序。

在本章中，我们将验证 EK-RA6M4 评估板 (EK) 是否正常工作并与您的 Windows® 工作站和 e² studio 的调试器通信。为此，我们将使用第 4.2 节中生成的项目。如果您没有做过此练习，不必担心，可以从本手册的网站下载。但是，在后一种情况下，您应该已经按照第 4 章中的说明安装并测试了开发工具链。

7.1 连接和开箱即用演示

如果您未曾将 EK 从箱子中取出，请现在完成这一步。检查内容并确认物品是否齐全：EK 主板、一根 USB Type A 转 micro-B 线缆、一个 Micro USB OTG 适配器和一根交叉以太网电缆。该评估板的文档（包括 EK-RA6M4 快速入门指南和用户手册）可从其网站（“*Documentation*”（文档）链接后的 **https://www.renesas.com/ra/ek-RA6M4**）下载。请注意，电路板包含一个 SEGGER J-Link® 板上 (OB) 调试器（提供完整的调试和编程功能），因此在使用 EK 和评估板随附的 USB 线缆时不需要仿真器。

首先，将 USB 线缆的 micro-B 端插入电路板的 USB 调试端口 J10（位于系统控制和生态系统访问区域的右下角），将另一端插入工作站的空闲 USB 端口。白色 LED4（构成文字“EK-RA6M4”中的连字符）应点亮，表示电路板已通电。电路板一旦启动并运行，便会执行预编程的演示程序，使蓝色 LED1 以 1 秒的间隔和 10% 的亮度闪烁。绿色 LED2 将以最强亮度常亮，而红色 LED3 则会熄灭。通过多次按下用户按钮 S1，可以将 LED1 的亮度从 10% 调整为 50%，再到 90%，然后返回到 10%。通过多次按下按钮 S2，可以将 LED1 的闪烁频率从 1 Hz 调整为 5 Hz，再到 10 Hz，然后返回到 1 Hz。

如果调试端口旁的橙色调试 LED (LED5) 继续闪烁，则评估板无法检测到 J-Link 驱动程序。如果发生这种情况，请仔细检查是否已将 USB 线缆连接到电路板右侧的调试端口 J10，而不是左侧的 USB 全速端口 J11。此外，请打开 Windows 的“*Device Manager*”（设备管理器），展开“*Universal Serial* Bus controllers”（通用串行总线控制器）树，以验证安装过程中 J-Link 驱动程序是否已复制到主机工作站。J-Link 驱动程序应在此处列出（请参见*图 7-1*）。否则，请使用“*Custom Install*”（自定义安装）功能重新运行评估安装程序并允许安装来自 Renesas Electronics Corporation 的软件。此操作应该可以解决问题。

此外，电路板还随附其他一些演示程序，例如 Web 服务器或 quad-SPI 和 octo-SPI 速度比较。要运行这些演示，您将需要通过工作站上的终端程序与评估板通信。请参见 EK-RA6M4 快速入门指南来设置连接和演示。如果只是熟悉流程，便不需要运行这些演示。

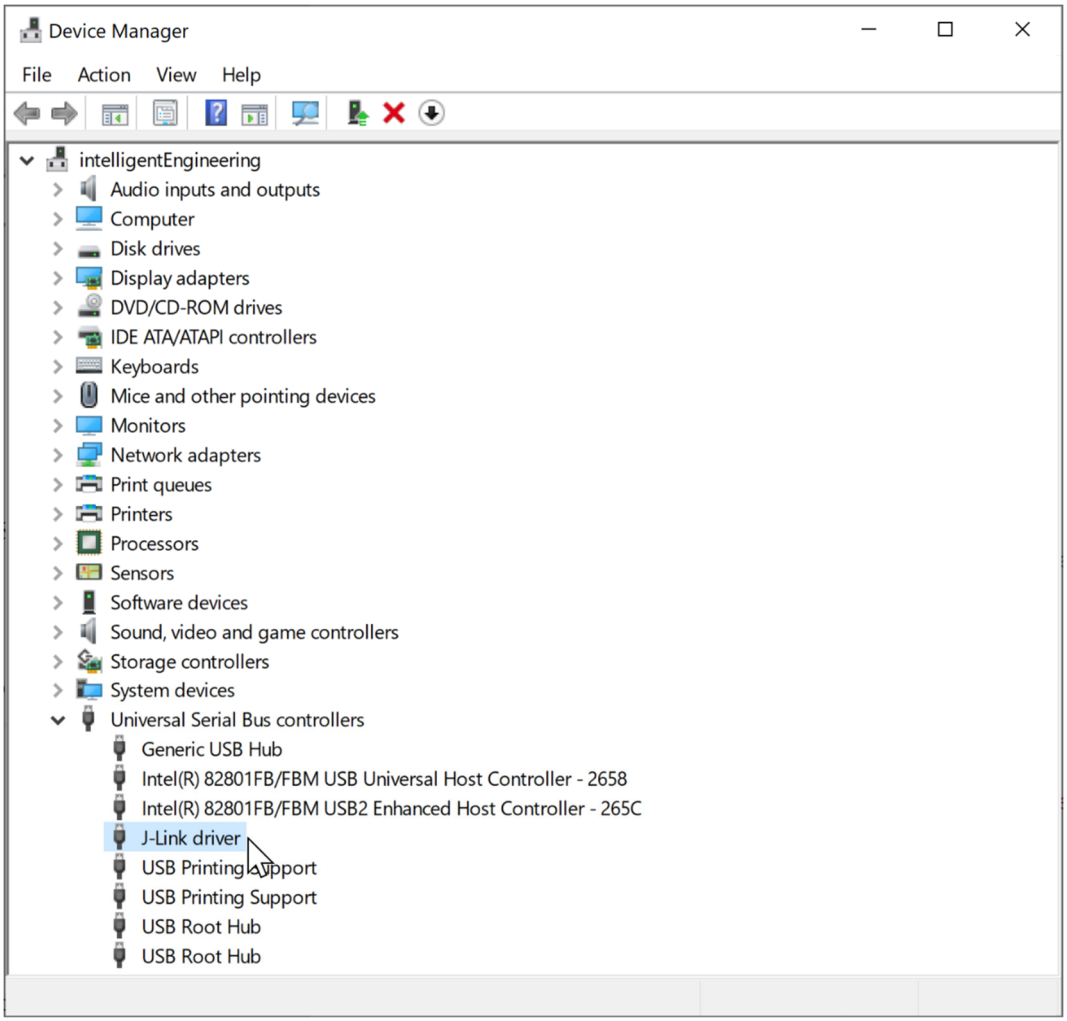


图 7-1：如果黄色调试 LED 没有停止闪烁，请确保正确安装了 J-Link 驱动程序

7.2 下载并测试示例

在 EK-RA6M4 仍连接到 Windows® 工作站的情况下，从操作系统的“Start”（开始）菜单打开 e² studio。如果系统提示您输入工作区的位置，请使用在第 4.2 节的练习中输入的位置（应该已经列出）。如果您没有做过这个练习，不用担心，可以从为本手册创建的网站（**https://www.renesas.com/ra-book**）下载项目。下载后，根据第 5.3.1 节中概述的说明将其导入工作区。在这种情况下，请使用所选的文件夹。

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）。

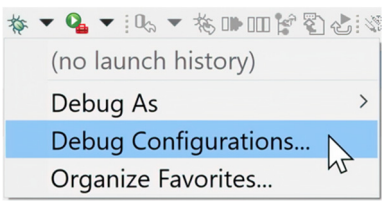


图 7-2：要开始调试，请从下拉列表框中选择“Debug Configurations”（调试配置）

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 *MyRaProject 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）打开一个新界面，其中显示相关的所有选项（请参见*图 7-3*）。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）。

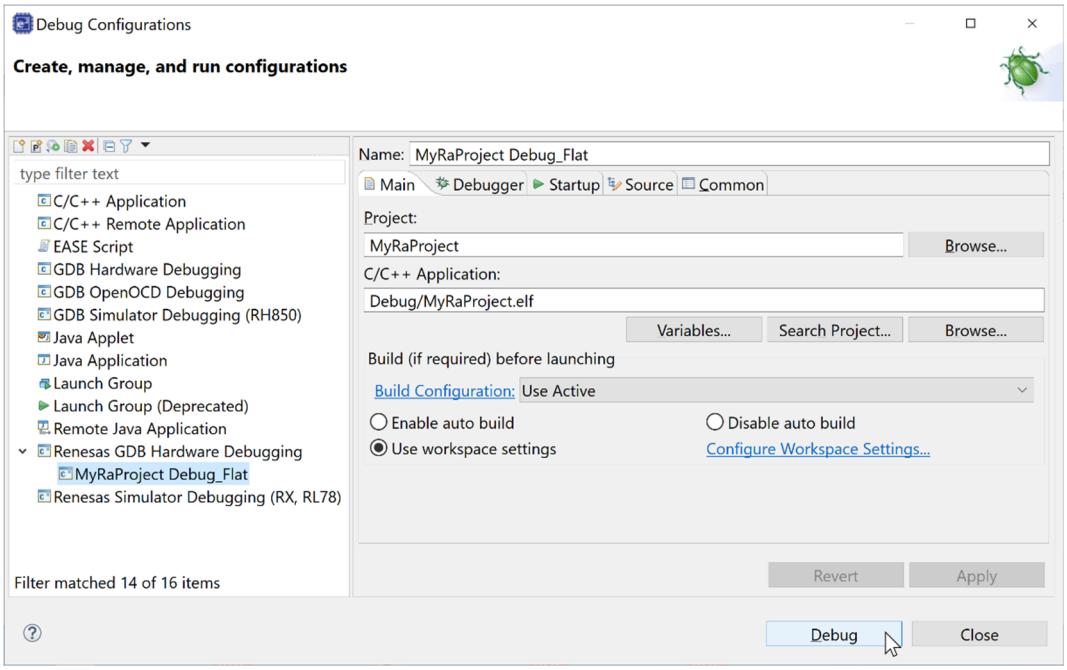


图 7-3：在“Renesas GDB Hardware Debugging”（Renesas GDB 硬件调试）下选择项目后，无需在出现的窗口中进行任何更改。

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 7-4*），调试器会将程序计数器设置为程序的入口点，即复位处理程序。单击“*Resume*”（恢复）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*”（恢复），程序将继续执行，评估板上的蓝色、绿色和红色 LED（LED 1 至 3）以 1 秒的间隔同时闪烁。

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。

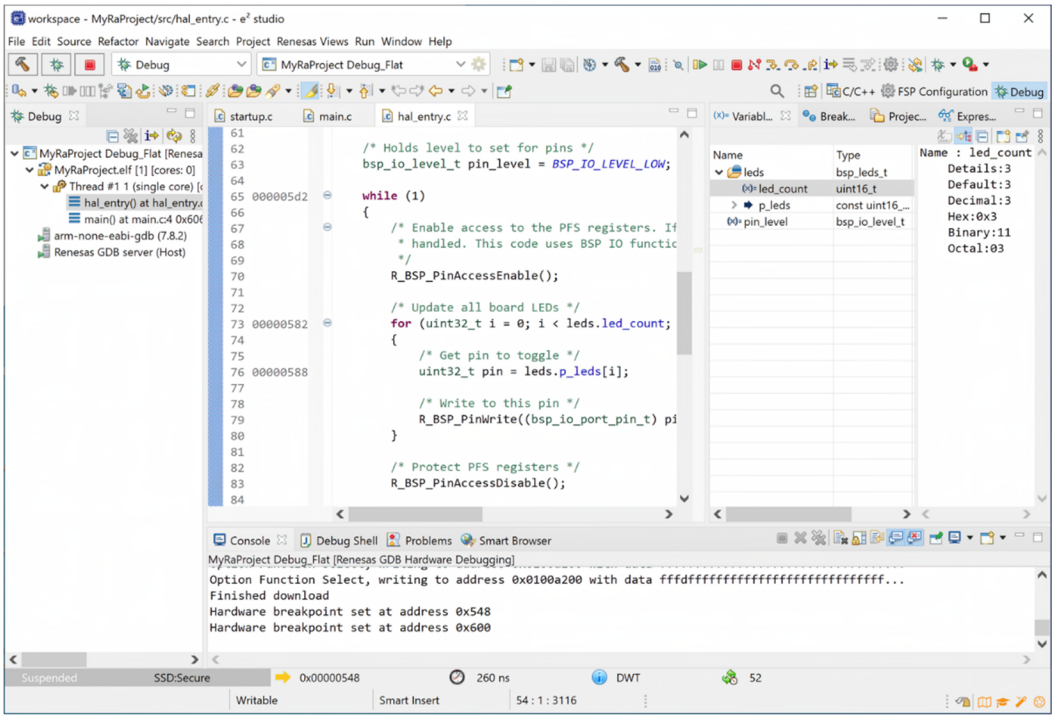


图 7-4：e² studio 的“Debug”（调试）透视图

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。

8 Hello World! – Hi Blinky!

您将在本章中学到以下内容：

- 如何从头开始为 EK-RA6M4 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。
- 如何编写代码以实现 EK 上的用户 LED 闪烁。
- 如何下载和测试程序。

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。对我而言，就是在编辑器中键入“Writeln (‘Hello World’)”，如同我开始学习 Pascal 一样。从那时起，我用其他几种语言编写了类似的代码行，主要是为了对新开发环境的安装进行完整性检查。

20 世纪 80 年代末，当我开始编写嵌入式系统时，没有可以接收字符串的屏幕。那么，如何指示处理器发出正常工作的信号？在这些应用中几乎找不到 LED，因此，必须要切换仅有的 I/O 引脚之一并用示波器观察波形。这些年来，LED 成为了一种商品，我们在电路板上放置了大量的 LED，将它们的闪烁作为新的“Hello World”。

这也是本章的目标：使用您在之前章节中学到的所有知识，切换 RA6M4 系列器件的评估板 (EK) 上的 LED 的亮灭状态：您将（几乎）从头开始编写代码，使用配置器创建一个新项目，调用灵活配置软件包 (FSP) 的 API，最后下载、调试并运行代码。这项练习将各个操作步骤集中到一起。

前提条件包括：e^2 studio 以及 FSP 应安装在 Windows® 工作站上（有关详细信息，请参见第 4 章），您应当已验证设置是否如第 7 章中所述工作。对于之前进行过动手实验的人员：请理解一下作者，因为作者决定再次介绍先前已经探讨过的一些主题，以方便直接从目录转到本章的读者。

在本练习中，我们将再次使用 EK-RA6M4，它允许您立即探究 RA6M4 单片机及其所有外设，因此非常适合此类任务。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 RA6M4 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 8-1* 所示为电路板的框图，其中突出显示了主要元件。

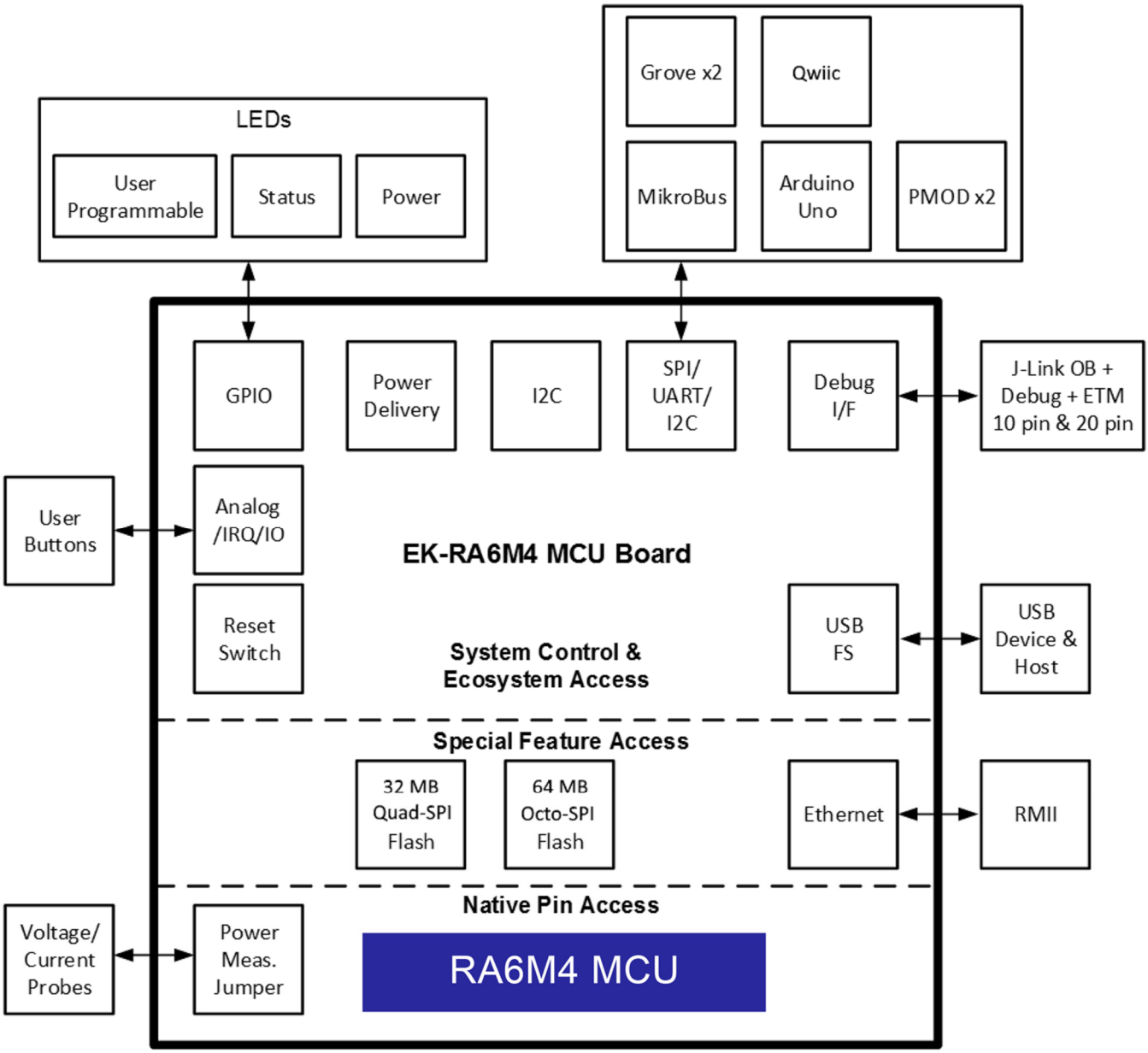


图 8-1：EK-RA6M4 评估板的框图

8.1 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）界面（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→Renesas RA*”（文件 → 新建 → 瑞萨 C/C++ 项目→瑞萨 RA），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 → C/C++ Project*”（新建 → 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）（请参见*图 4-7*）。然后单击“*Next*”（下一步）。

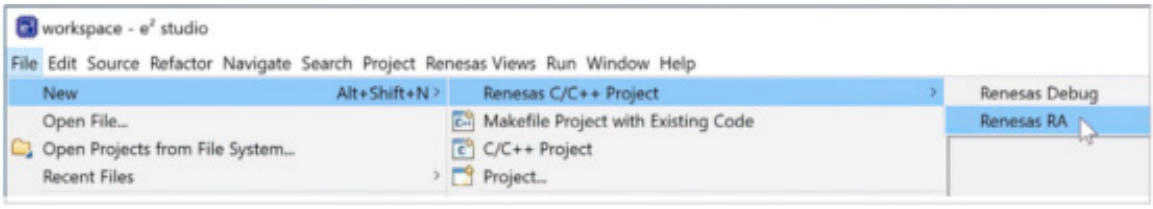


图 8-2：第一步是调用项目配置器

出现“*Project Configurator*”（项目配置器）后，为项目命名（例如 *MyBlinkyProject*），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）界面。

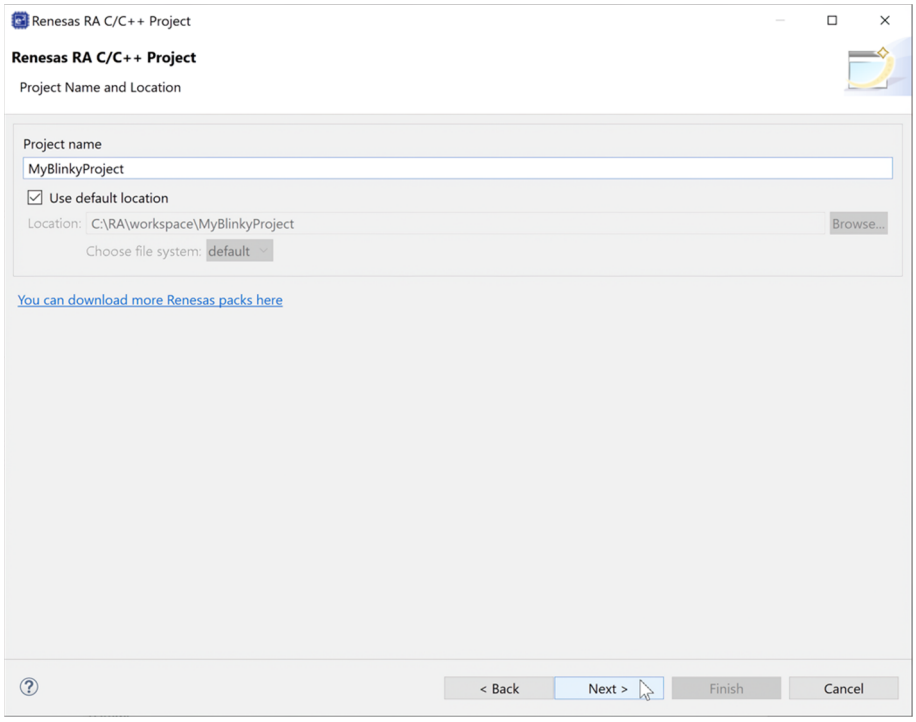


图 8-3：项目配置器的第一个界面主要询问项目的名称和位置

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 *EK-RA6M4*，因为这是我们用于小型“Hello World”程序的硬件。该列表通常将包含 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。验证 *R7FA6M4AF3CFB* 是否在“*Device*”（器件）旁显示，它应该已经自动插入。如果未显示，请浏览下拉列表，直到发现为止。在“*Toolchains*”（工具链）框中，验证是否列出了 *GCC ARM® Embedded, 10.3.1.20210824* 或更高版本，以及“*Debugger*”（调试器）框中是否已选择 *J-Link® Arm*。这些字段应预先填入。如果未预先填入，请相应修改以匹配上面给出的值。

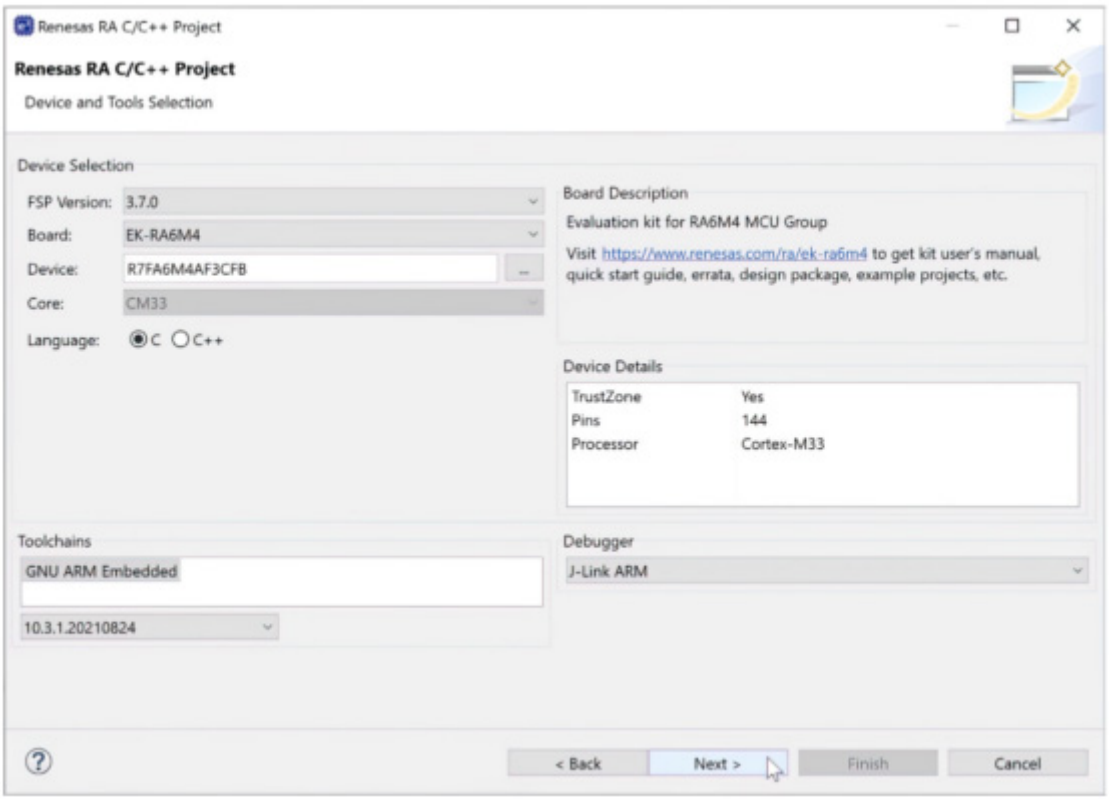


图 8-4：可以在此页面中选择项目的电路板和器件

如果一切正常，请单击“*Next*”（下一步），打开“*Project Type Selection*”（项目类型选择）界面。在此处，可以选择您的项目为“扁平化项目”（即无需 TrustZone® 隔离即可立即执行的项目）、包含安全启动代码和其他安全代码的安全 TrustZone 项目，还是包含与安全项目一起使用的非安全代码的非安全 TrustZone 项目。对于本章中的练习，选择“*Flat (Non-TrustZone) Project*”（扁平（非 TrustZone）项目），然后单击“*Next*”（下一步）继续操作。

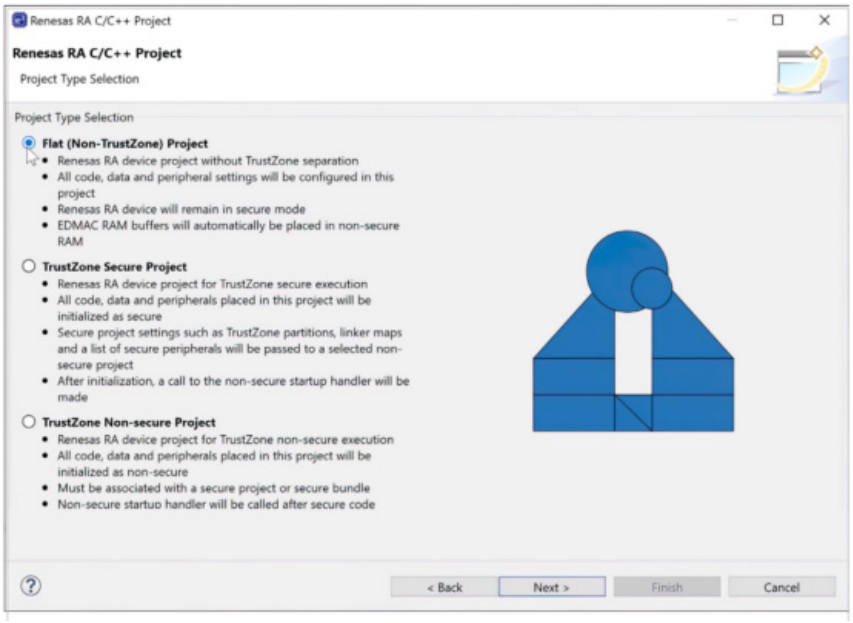


图 8-5：使用“Project Type Selection”（项目类型选择）界面，可以在 TrustZone 和非 TrustZone 项目之间进行选择

下一页是“*Build Artifact and RTOS Selection*”（构建工件和 RTOS 选择）界面，可以在其中设置构建的类型。只有选择了非 TrustZone 器件或者在上一个窗口中为 TrustZone 器件选择了扁平化项目时，才会显示此界面。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)。

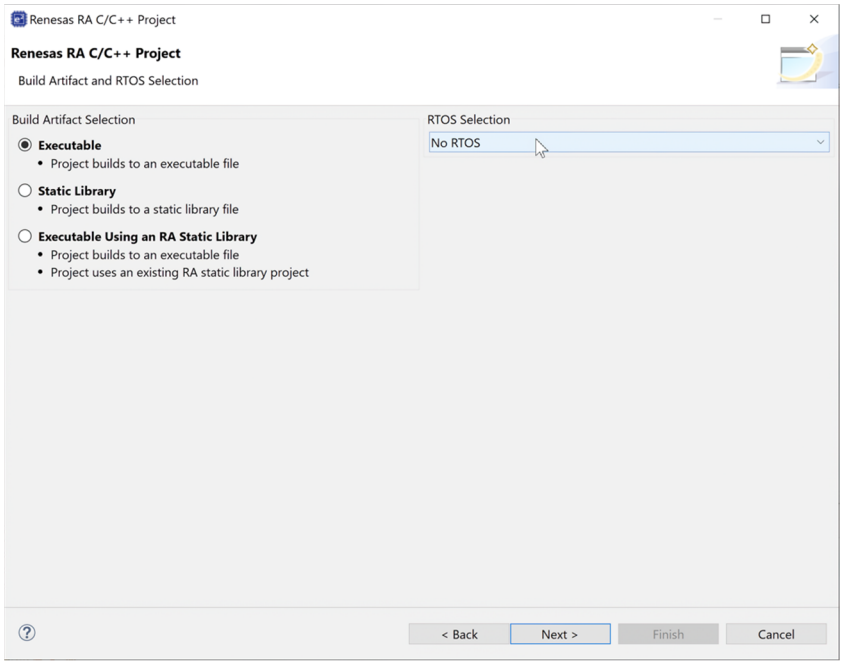


图 8-6：我们要编译一个无 RTOS 的可执行项目

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“*Next*”（下一步）。

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界面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“*Finish*”（完成），完成项目的配置。

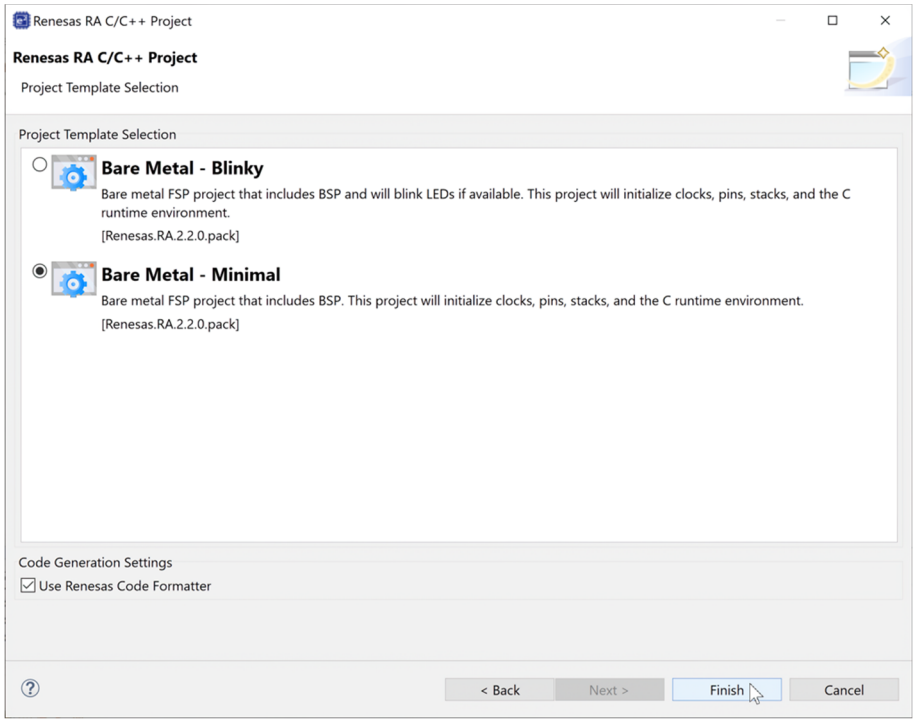


图 8-7：“Project Template Selection Page”（项目模板选择页面）将显示适合项目最初内容的模板

“*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）。

8.2 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 Communities 论坛）以及硬盘上的 FSP 用户手册。

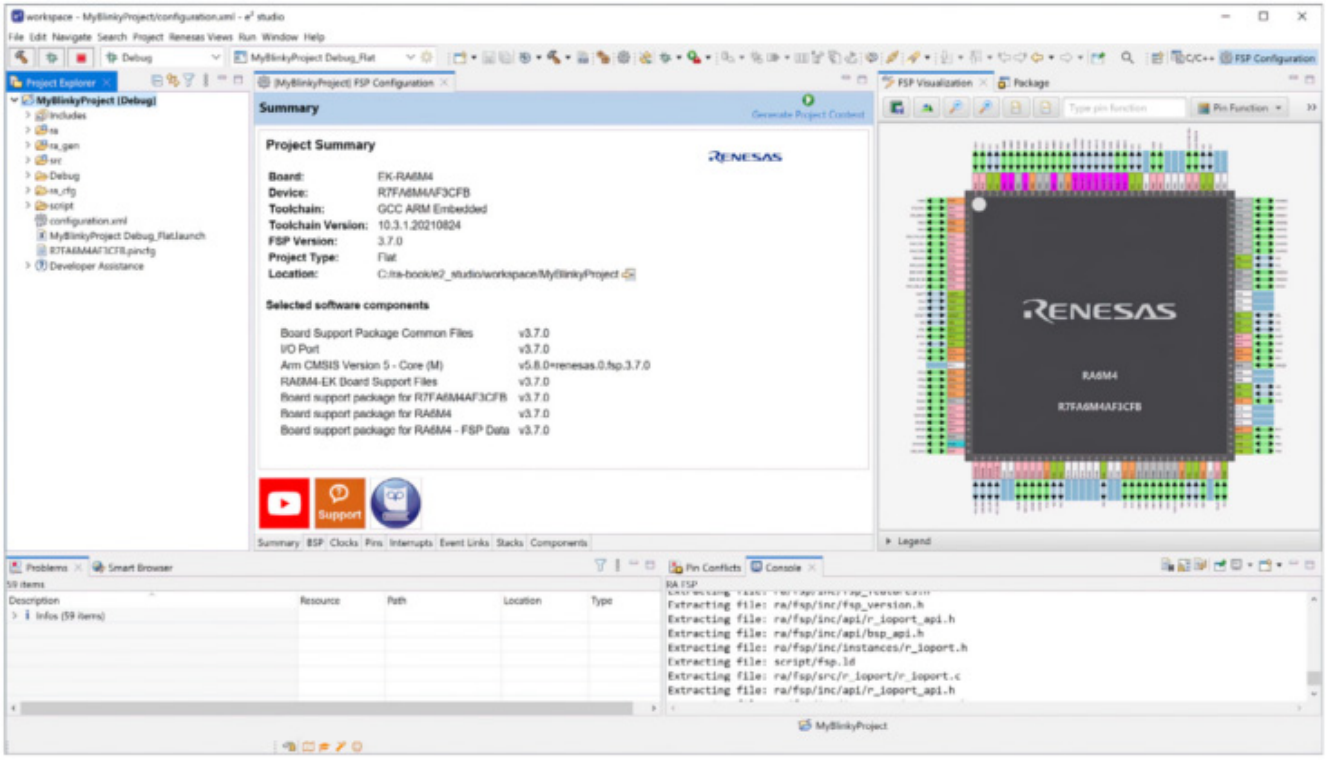


图 8-8：e² studio 内部的“FSP Configuration”（FSP 配置）透视图

在名为 *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。可以在此处手动指定用户定义的（即非 FSP 生成的）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。可以在此处手动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。

8.3 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，则更改会丢失，因为每次单击“*Generate Project Content*”（生成项目内容）时，都将覆盖该文件。

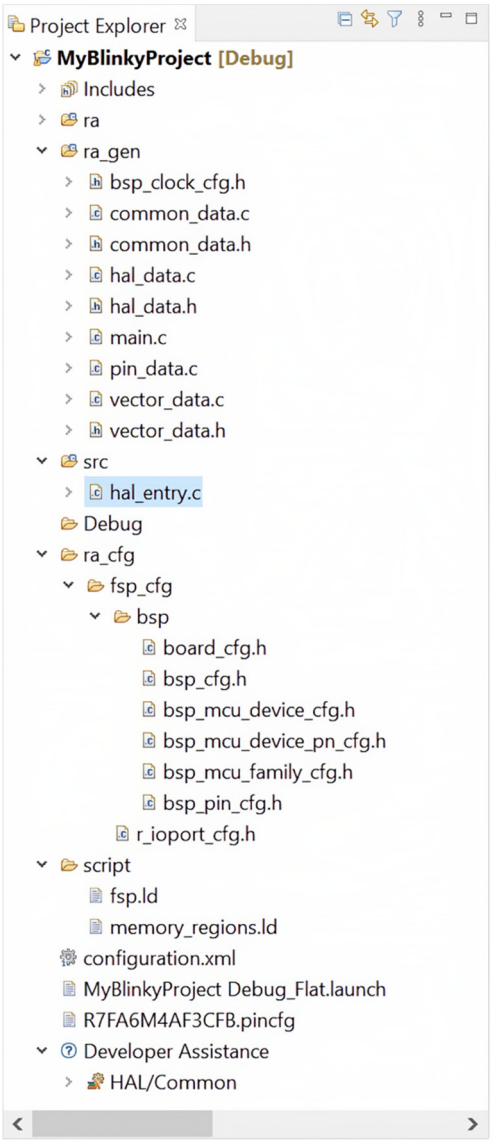


图 8-9：FSP 配置器创建所需文件后的项目树

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的根目录中的文件。

接下来，为 RA 产品家族单片机编写第一个真实源代码。计划是在 EK-RA6M4 评估板上的绿色 LED2 和红色 LED3 之间每秒交替切换，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。如何实现？

实际上有两种选择：一种是通过接口函数来使用 API，另一种是使用 BSP 实现函数。您认为哪一种更好？如果您不确定答案，可以回顾第 2 章。

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 LED2 和 LED3 连接到哪些 I/O 端口，以及有多少个 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\boards\ra6m4_ek* 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绿色 LED2（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_LED2],
                         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 LED3。
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。

完成这些操作后，接下来要做的是复制/粘贴三行代码，并反转第二组中 LED 的引脚电平。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_LED2],
                             BSP_IO_LEVEL_LOW);
    g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                             Leds.p_leds[BSP_LED_LED3],
                             BSP_IO_LEVEL_HIGH);
    R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);

    g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                             Leds.p_leds[BSP_LED_LED2],
                             BSP_IO_LEVEL_HIGH);
    g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                             Leds.p_leds[BSP_LED_LED3],
                             BSP_IO_LEVEL_LOW);
    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。

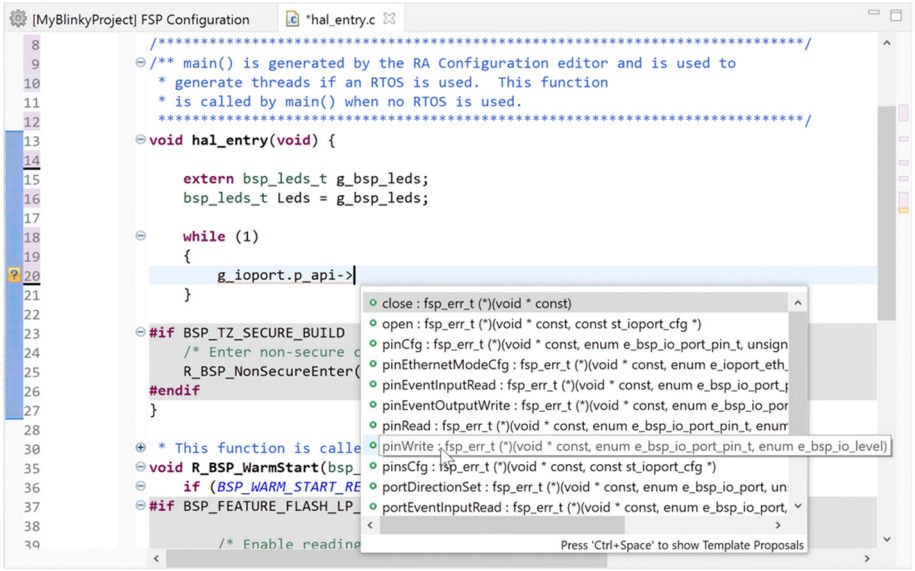


图 8-10：在变量或函数上按下 <Ctrl>-<Space> 将激活 e² studio 的代码补全功能

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。

8.4 编译第一个项目

输入所有内容后，便可随时编译程序。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）。

对于第一个测试，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，编译器的反馈将直接插入编辑器窗口（如果可能）。

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

8.5 下载和调试第一个项目

下一步是在评估板 (EK) 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 端插入系统控制和生态系统访问区域右下角的 USB 调试端口 J10，将另一端插入 PC 上的空闲端口。白色 LED4（构成文字“EK-RA6M4”中的连字符）应点亮，表示电路板已通电。如果该评估板支持开箱即用，则预编程的演示会运行，表明一切都按预期运行。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。

如果 USB 端口旁边的橙色调试 LED5 在短时间内不停地闪烁，则表示工作站上的 J-Link® 驱动程序可能有问题。如果发生这种情况，请参见第 7.1 节获取可能的解决方案。

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 *MyBlinkyProject 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 EK 上的 RA6M4 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。

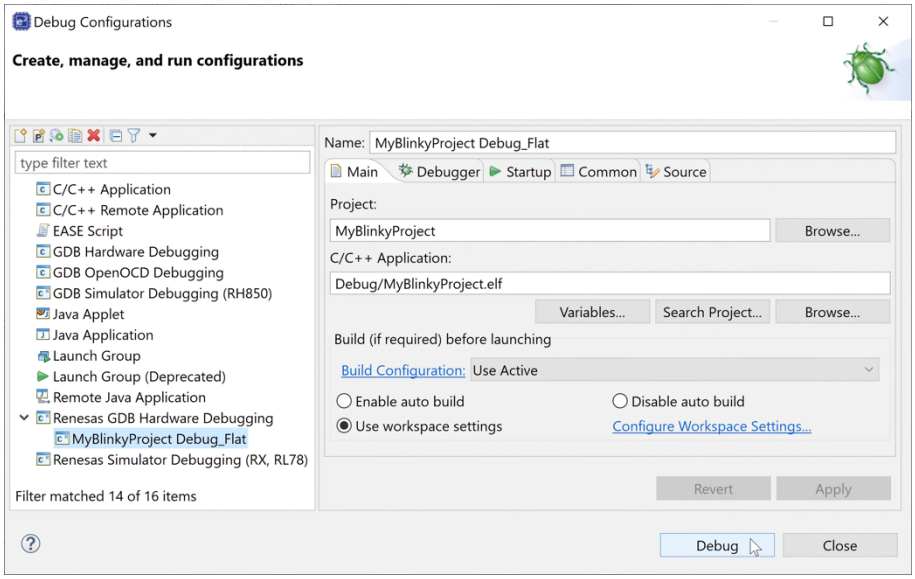


图 8-11：选择 MyBlinkyProject Debug_flat 后，无需在其他选项卡上进行任何更改

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，并按预期的 1 秒时间间隔在评估板上的绿色和红色 LED 之间切换。

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将其终止。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。

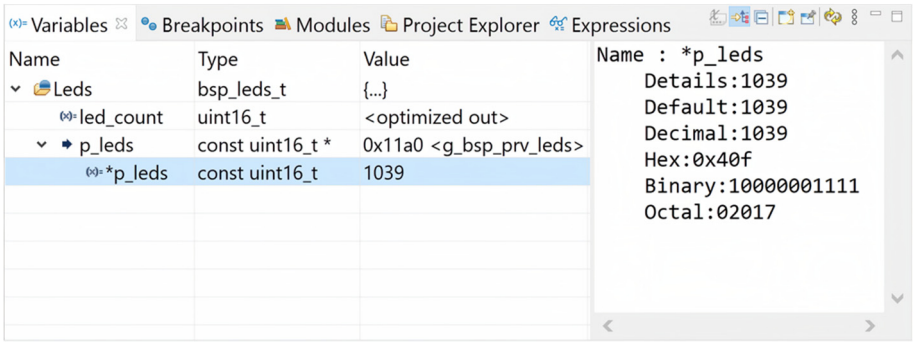


图 8-12：变量及其值可以在“Variables”（变量）视图中进行检查

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。

9 使用实时操作系统

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 LED。

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。尽管这一章节使用的是 FreeRTOS，你依然可以用 Azure®RTOS 做练习，因为它们的步骤是很相似的。

9.1 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。Azure RTOS 提供了相同的功能。

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 和 Azure RTOS 提供计数信号量和二进制信号量。尽管二进制信号量由于只能采用两个值（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()`(FreeRTOS) 或者 `tx_semaphore_get()`(Azure RTOS)（将使信号量递减 1）和 `xSemaphoreGive()` 或者 `tx_semaphore_set()`（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，它不存储引用而是消息本身。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。在 AzureRTOS 中，这两个函数分别被命名为 tx_queue_send() 和 tx_queue_receive()。

9.2 使用 e² studio 将线程添加到 FreeRTOS 中

接下来的练习也是基于 EK-RA6M4 评估板。这次，我们将使用电路板左上方的蓝色按钮 S1 向应用程序传输事件，应用程序将切换绿色 LED2 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。

像往常一样，第一步是使用项目配置器创建一个新项目，这已在第 4 章和第 8 章中做过练习。首先，转到“*File → New → Renesas C/C++ Project → Renesas RA*”（文件 → 新建 → 瑞萨 C/C++ 项目 → 瑞萨 RA）。在所出现窗口的侧边栏中选择 *Renesas RA*，并突出显示“*Renesas RA C/C++ Project*”（瑞萨 RA C/C++ 项目）条目。单击“*Next*”（下一步）并在出现的界面上输入项目名称，例如 *MyRtosProject*。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 *EK-RA6M4* 并将相应的器件设置为 *R7FA6M4AF3CFB*（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。

在当前出现的界面中，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）的界面。在此，选择“*FreeRTOS – Minimal – Static Allocation*”（FreeRTOS – 最小化 – 静态分配）。

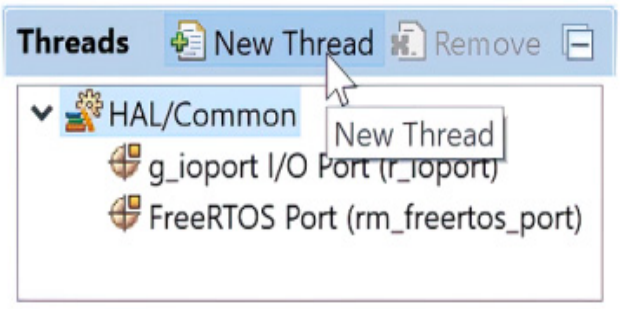


图 9-1：在 FSP 配置器出现之后，将仅显示一个线程。选择“New Thread”（新线程）按钮，添加另一个线程

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）线程的两个条目，其中包含 I/O 端口的驱动程序另一个包含 FreeRTOS。单击窗格顶部的“*New Thread*”（新线程）图标（请参见***错误!未找到引用源。*** 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“*Input -> External IRQ Driver on r_icu*”（输入 → r_icu 上的外部 IRQ 驱动程序）（请参见*图 9-2*）。

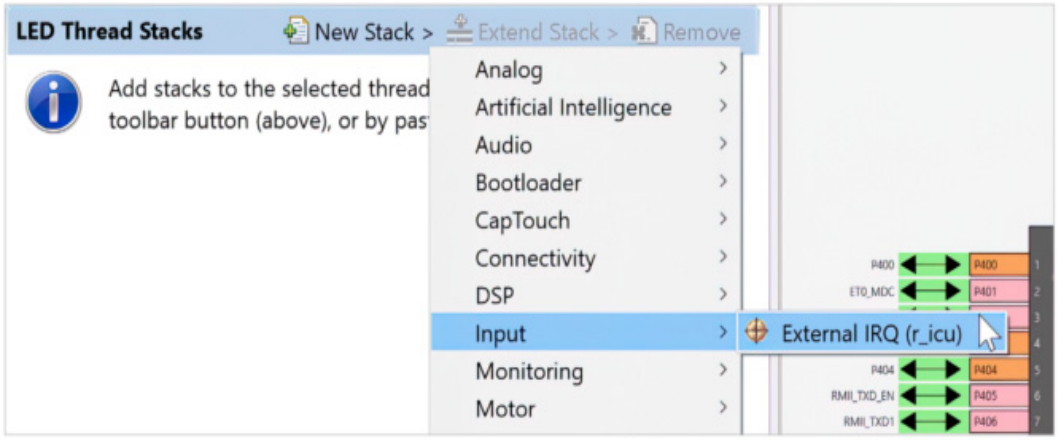


图 9-2：添加新驱动程序只需单击几下鼠标

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）并进行一些更改：首先，将“*Channel*”（通道）从 *0* 更改为 *10*，因为 S1 所连引脚连接到 IRQ10。出于相同的原因，将名称更改为 *g_external_irq10* 或您喜欢的任何名称。

为中断分配优先级 12，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 12，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 15 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。

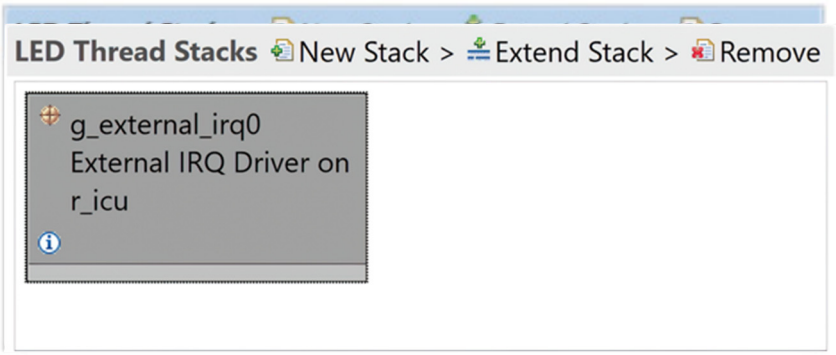


图 9-3：堆元素的灰色条表示此驱动程序是模块实例，只能由另一个 FSP 模块实例引用。

将“*Trigger*”（触发源）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）。可以将“*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_irq10_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 9-4* 给出了必要设置的摘要。

Property	Value
⌄ Common	
Parameter Checking	Default (BSP)
⌄ Module g_external_irq10 External IRQ (r_icu)	
Name	g_external_irq10
Channel	10
Trigger	Falling
Digital Filtering	Enabled
Digital Filtering Sample Clock (Only valid when Digit	PCLK / 64
Callback	external_irq10_callback
Pin Interrupt Priority	Priority 12
⌄ Pins	
IRQ10	P005

图 9-4：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 9-5*。

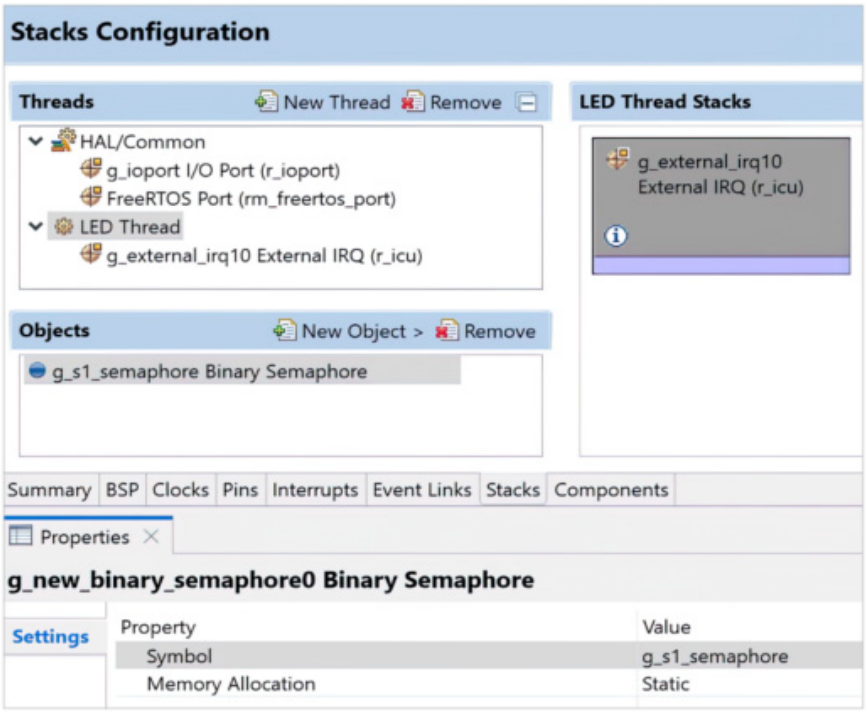


图 9-5：这是添加 LED 线程和信号量后“Stacks”（堆）选项卡应呈现的外观

FSP 配置器中的最后一步是将 S1 连接的 I/O 引脚配置为 IRQ10 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 *P005*。在 RA6M4 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 S1*，并确保其他设置与*图 9-6* 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 135/P005，这样便可获得引脚位置的图形参考。

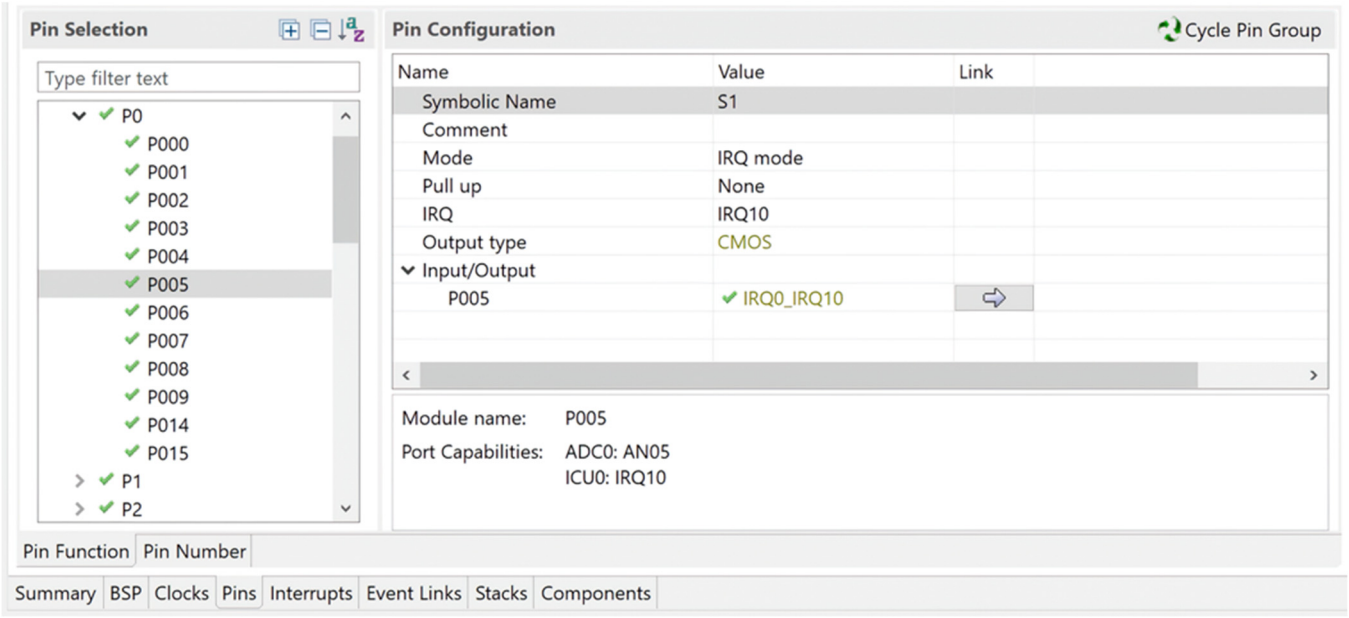


图 9-6：为 IRQ10 正确配置了端口 P005

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，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 8 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 LED2 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 EK-RA6M4 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 IRQ10。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_irq10.p_api->open(g_external_irq10.p_ctrl,
                             g_external_irq10.p_cfg);
g_external_irq10.p_api->enable(g_external_irq10.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 LED2 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 8 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 IRQ 10 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_irq10 External IRQ Driver on r_icu*”（开发人员帮助 → LED 线程 → r_icu 上的 g_external_irq10 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_irq10_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 8 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 EK 上的 S1 时，绿色 LED2 都应切换。

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_irq10.p_api->open(g_external_irq10.p_ctrl,
                                g_external_irq10.p_cfg);
   g_external_irq10.p_api->enable(g_external_irq10.p_ctrl);

   while (1)
   {
       g_ioport.p_api->pinWrite(&g_ioport_ctrl
                                Leds.p_leds[BSP_LED_LED2],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 S1 push button; sets the semaphore */
void external_irq10_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。

10 使用“灵活配置软件包”通过 USB 端口发送数据

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”的中间件来设置 USB 传输。
- 如何在主机工作站上接收 MCU 发送的数据。

在本部分，我们将使用瑞萨 RA 产品家族微控制器的“灵活配置软件包”(FSP) 的 USB 中间件，在每次按下用户按钮 S1 时，将 LED2 的当前状态作为文本字符串通过 USB 端口发送到 Windows® 工作站。与第 9 章不同的是，我们在此实验中不使用实时操作系统和信号量，而使用全局变量来指示按钮开关已激活和绿色 LED2 的状态已更改。

LED 状态（ON 或 OFF）更新、USB 端口的写操作，以及保存按钮按下时的信息的全局变量更新将在 IRQ10 的回调例程中完成。端口的写操作将触发 USB 传输，将 LED 的相关信息发送给主机。返回到 `hal_entry()` 函数内部的无限循环后，将处理 USB 事件，并通过将全局变量设置为“false”和将下一个字符串及下一个 LED 电平分配给各自的变量来准备 LED 状态的下次更新。*图* **10-1** 详细描绘了该程序的流程和中断回调函数的流程。

该 USB 端口的大部分设置将在 FSP 配置器的图形界面中完成，应用程序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 FSP 给用户提供的便利，即便在构建如 USB 之类的复杂通信系统时也非常方便。

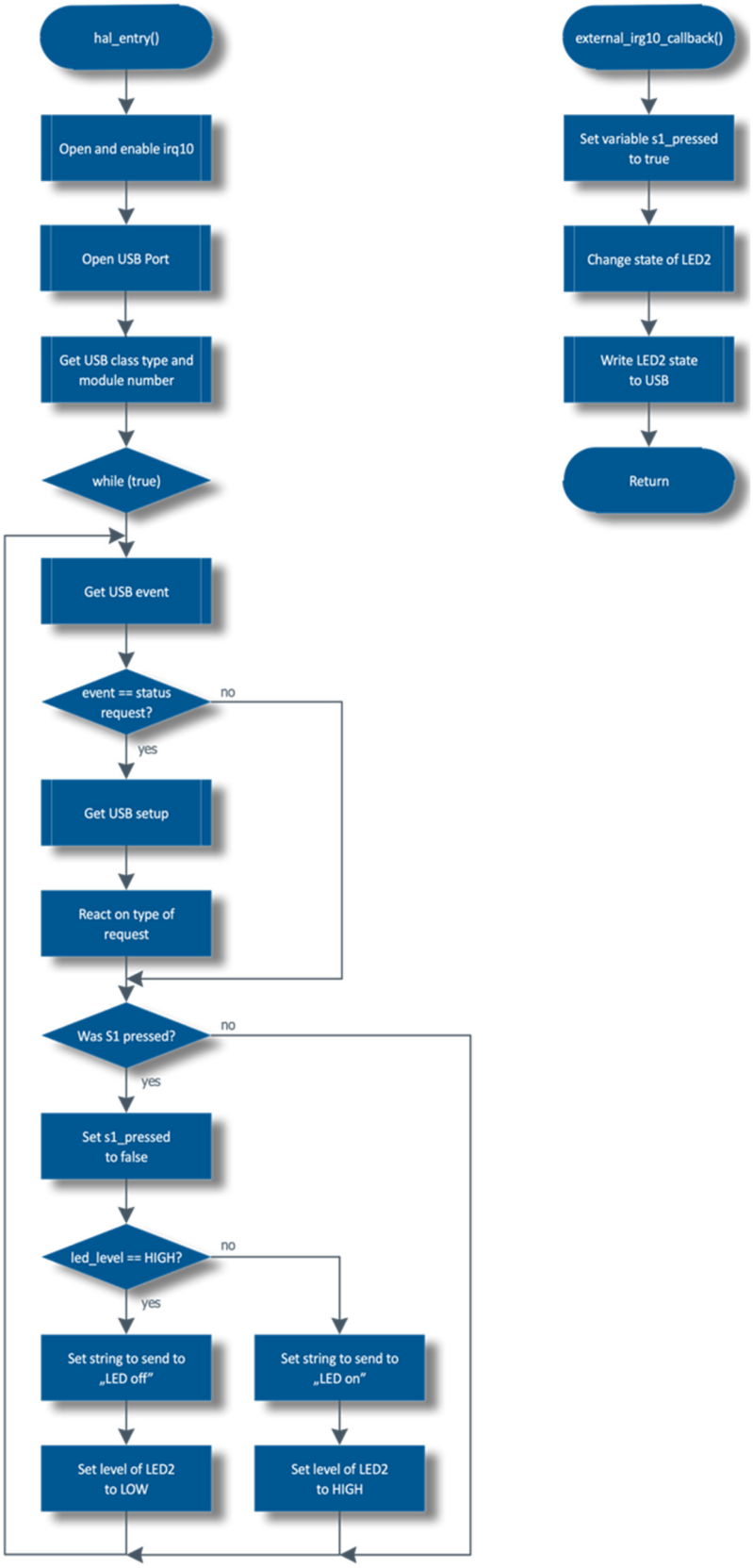


图 10-1：本章练习的流程图

10.1 使用 FSP 配置器设置 USB 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 *MyUSBProject*，在进入 *"Device and Tools Selection"*（器件和工具选择）界面后，选择 *EK-RA6M4* 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，直接转到 *"Stacks"*（堆）选项卡。首先，我们需要添加用于连接到用户按钮 S1 的外部中断的模块。在 *"HAL/Common Stacks"*（HAL/通用堆栈）窗格上，单击 *"New Stack"*（新堆），然后选择 *"Input -> External IRQ Driver on r_icu"*（输入 → r_icu 上的外部 IRQ 驱动程序）。

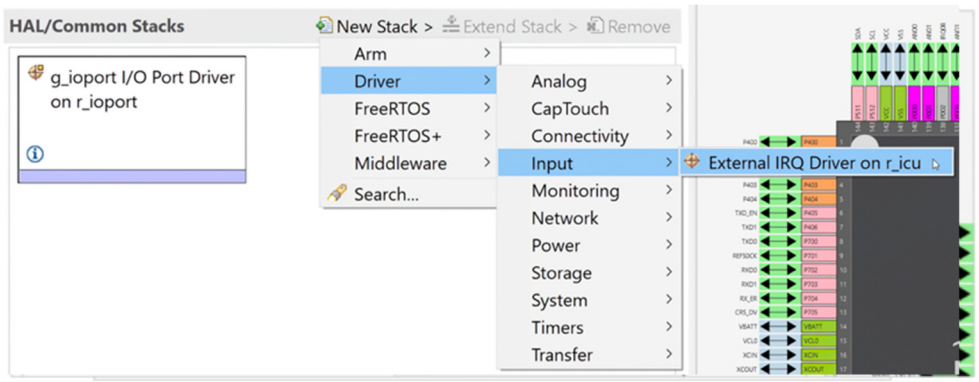


图 10-2： 首先添加 S1 中断的驱动程序

在 "Properties"（属性）视图中，将中断的 *"Name"*（名称）修改为 *g_external_irq10*，将它的 *"Channel"*（通道）修改为 *10*，以作为中断使用的通道。启用 *"Digital Filtering"*（数字滤波）并将 *"Trigger"*（触发器）设置为 *"Falling"*（下降）。这有助于消除开关的抖动。最后，需要提供用于该中断的回调函数的名称：将其命名为 *external_irq10_callback*，并将 *"Priority"*（优先级）改为 *14*，因为我们要确保 USB 中断的优先级高于按钮（参见*图 10-3*）。

Property	Value
⌄ Common	
Parameter Checking	Default (BSP)
⌄ Module g_external_irq10 External IRQ Driver on r_icu	
Name	g_external_irq10
Channel	10
Trigger	Falling
Digital Filtering	Enabled
Digital Filtering Sample Clock (Only valid when Digita	PCLK / 64
Callback	external_irq10_callback
Pin Interrupt Priority	Priority 14
⌄ Pins	
IRQ10	P005

图 10-3： 以上是 IRQ10 的必要设置

接下来，将 USB 外设通信设备类 (PCDC) 的中间件添加到系统中：创建新堆栈，并选择 *“Connectivity -> USB -> USB PCDC driver on r_usb_pcdc ”*（连接 → USB → r_usb_pcdc 上的 USB PCDC 驱动程序）（参见 *图 10-4*）。

此操作将四个模块添加到项目中：用于全速 USB 端口的实际 PCDC 驱动程序（用于实现应用程序级 USB

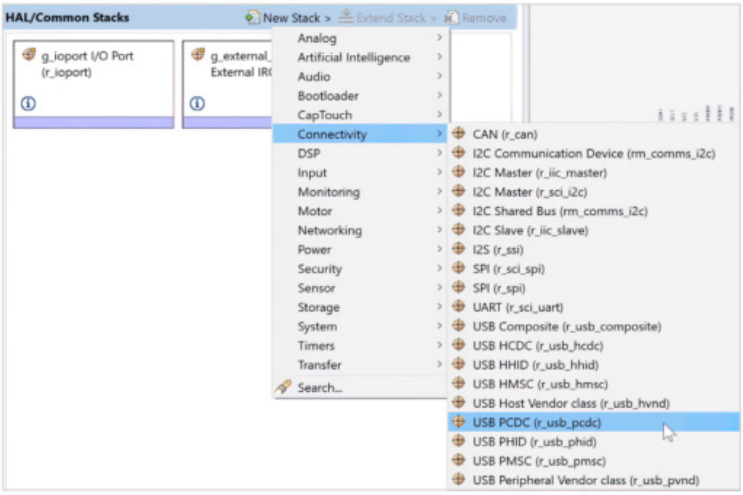


图 10-4：需要将 USB 外设通信设备类驱动程序的中间件添加到系统中

PCDC 接口），以及 r_usb_basic 上的基本 USB 驱动程序。栈中还显示两个具有粉红色横条的模块。这些模块用于添加可选的直接内存访问控制器 (DMAC) 驱动程序，以传输或接收数据。我们将使用 USB 写入 API 函数直接发送状态消息，因此无需添加它们。关于模块的其他色彩色条的含义，只需记住以下规则：灰色标记仅可由一个其他模块实例引用的模块实例，蓝色标记可由多个其他模块实例（甚至跨多个栈）引用的通用模块实例。通过彩色条中的小三角形，可以展开或折叠模块树。

将 USB 端口作为 PCDC 设备来实现，可以将 USB 端口用作虚拟 COM 端口，从而简化主机端的接收器设置，因为在注册到 Windows® 后，便可通过终端应用程序进行数据通信。这就是我们与评估板进行对话的方式。

添加所有堆栈后，*“Stacks ”*（堆栈）窗格的外观如*图 10-5* 所示：

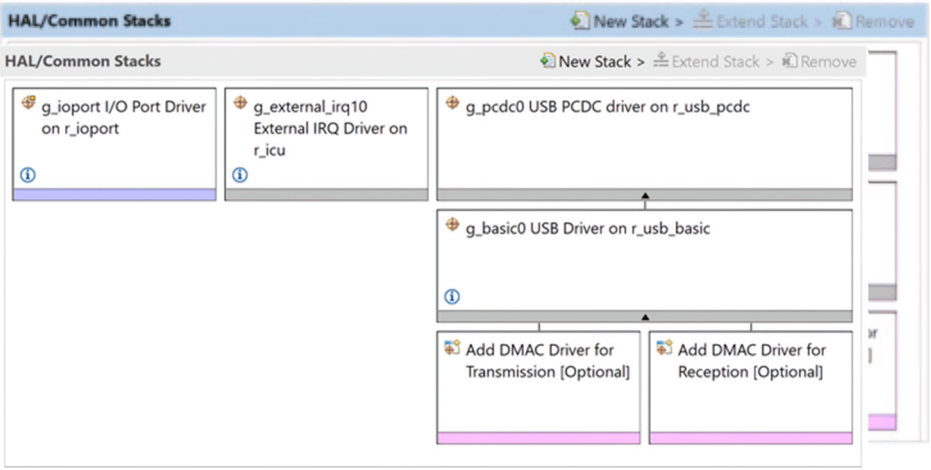


图 10-5：添加 USB 驱动程序后“堆”窗格的外观

不需要对基本 USB 驱动程序的 *“Properties”*（属性）做出更改。只要保证 “DMA Support”为 Disable。记录 *“g_basic0 USB Driver （ r_usb_basic ）”*（r_usb_basic 上的 g_basic0 USB 驱动程序）部分的 *“USB Descriptor”*（USB 描述符）的名称：*g_usb_descriptor*。稍后将创建一个具有该名称的结构体，以描述系统 USB 的功能，因此应记住这个名称。*图 10-6* 显示了修改后模块的属性。

需要注意 *g_basic0 USB(r_usb_basic)* 这个模块以红色显示的。把鼠标悬停在其上面会显示一下解释：USB 需要一个 48MHz 的时钟。我们可以在配置器中的“时钟”选项卡中更正过来，但是在我们改这个之前，我们会首先为 USB 设置正确的操作模式。

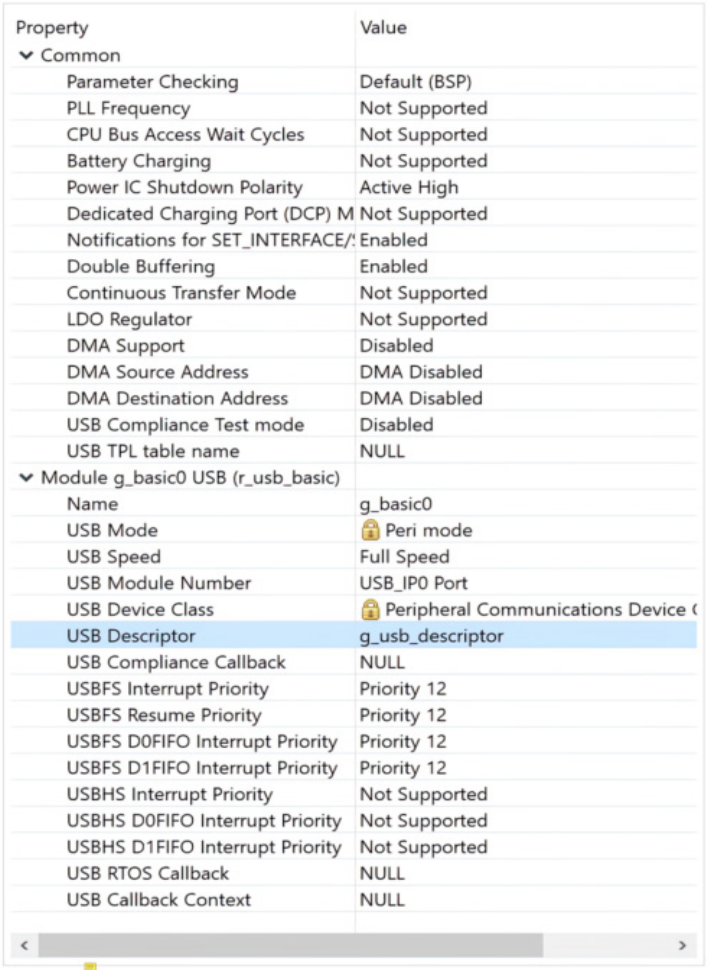


图 10-6：进行必要更改后的连续传输设置。记录“USB Descriptor”（USB 描述符）的名称

在 *“Stacks ”*（堆）选项卡中完成所有设置后，现在需要设置 USB 端口的正确操作模式。为此，请切换到 *“Pins ”*（引脚）选项卡，在“Pin Selection”（引脚选择）窗格中，首先展开 *“Peripherals ”*（外设）下拉列表，然后展开 *“Connectivity: USB ”*（连接：USB）列表。在 *“Pin Configuration ”*（引脚配置）窗格中，将 *“Operation Mode ”*（操作模式）从 *“Custom ”*（自定义）修改为 *“Device ”*（设备），作为要使用的模式。注意，输入/输出引脚分配将相应改变。

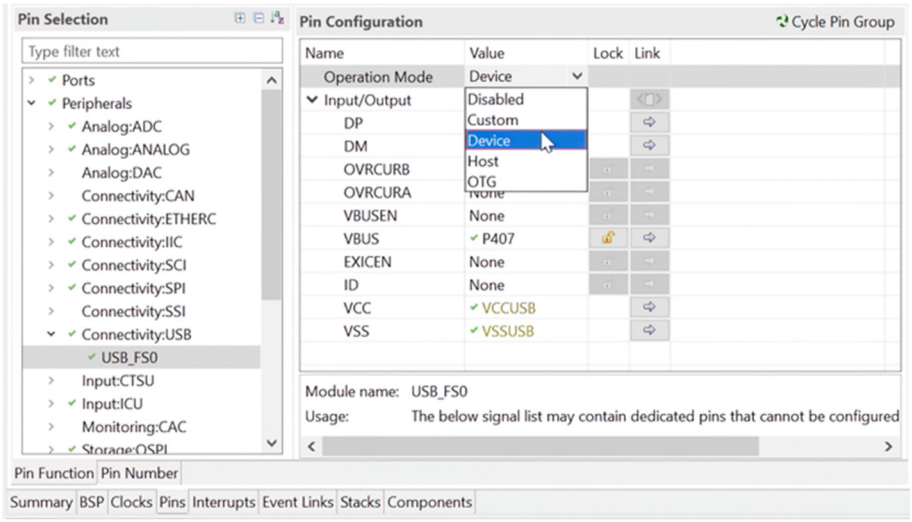


图 10-7：USB 端口将使用器件模式，请进行相应更改

现在还差一步就完成了 USB 端口的设置。最后一步是启用 *“USB clock (UCLK) ”*（USB 时钟 (UCLK)），用作全速 (FS) USB 模块的工作时钟，并将其设置为所需的 48 MHz 频率。为此，激活 *“Clocks ”*（时钟）选项卡，可通过该选项卡配置时钟生成电路。首先启用 USB 时钟，方法为：将靠近窗格底部的 *UCLK* 从 *“Disabled ”*（已禁用）更改为 *“Enabled ”*（已启用），并选择 *PLL2* 作为源。接下来，启用 *PLL2*，它是 USB 模块的专用 PLL，并选择 *高速片上振荡器 (HOCO)* 作为源。请注意，此时给定的 *UCLK 频率*为 40 MHz，并以红色突出显示，因为 USB 要求的频率为 48 MHz。将 *PLL2* 的*乘数值*更改为 *24*，以便将 PLL2 的频率更改为 240 MHz。再使用 UCLK 的除数 5，此时 UCLK 的频率已正确设置为 48 MHz。这里，采用 Arm® Cortex®-M33 内核的 RA MCU 系列凸显了巨大的优势：微控制器上还有第二个 PLL，可以将 USB 的时钟频率设置为 48 MHz，同时 MCU 以最高速度 200MHz 运行。

最后，将标准 PLL 的源从 *XTAL* 更改为 *HOCO*。如果不确定要更改哪些字段，请参见*图 10-8*。

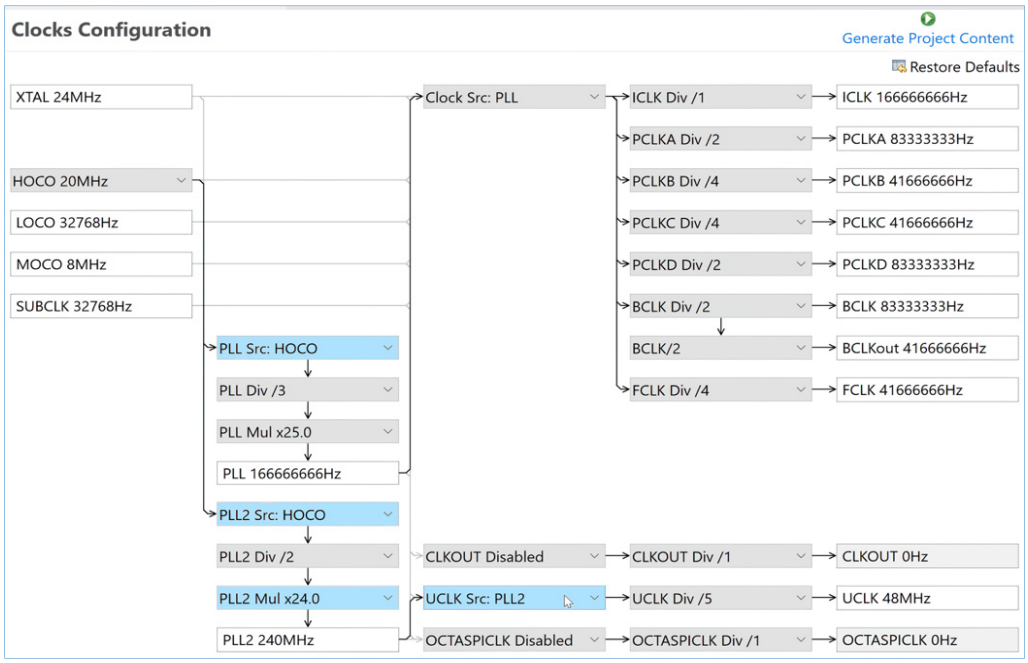


图 10-8：USB FS 需要使用 48 MHz 的时钟，因此需要相应地更改时钟生成电路。必要的更改已突出显示

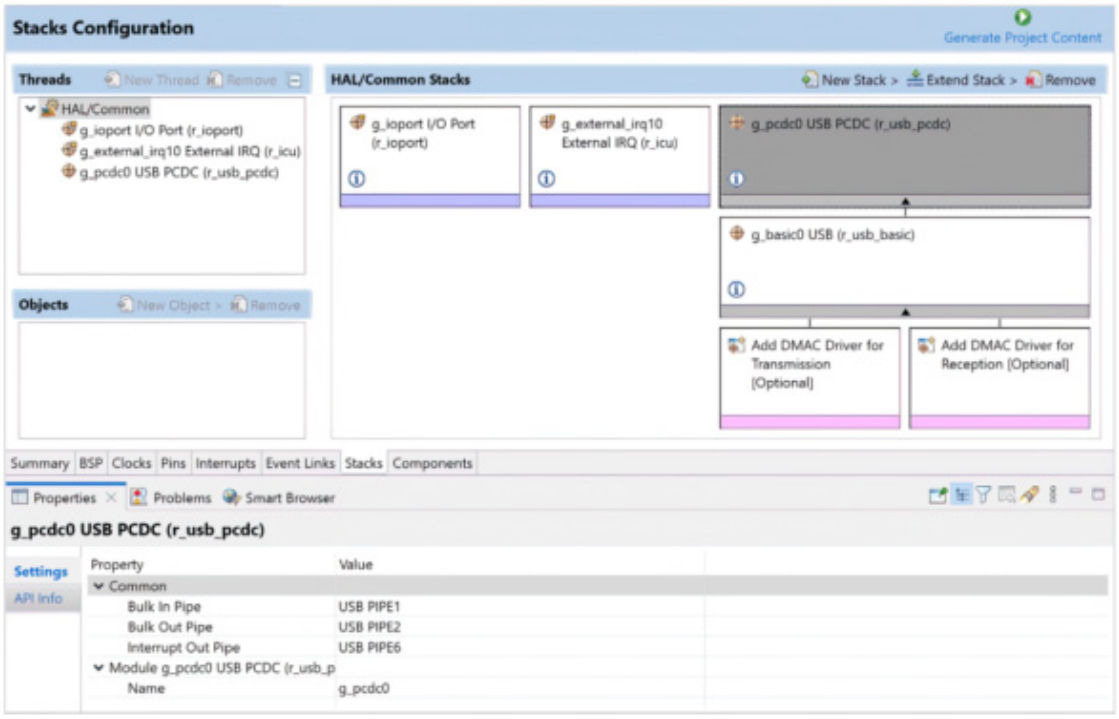


图 10-9：所有的修改和添加完成之后，栈选项卡应该像这样显示

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。

10.2 创建代码

现在，添加初始化 USB 端口和执行端口写入所需的代码。由于本练习需要输入大量的内容，建议您通过瑞萨网站从本手册对应的网页下载该实验的解决方案，这样只需按照说明进行操作，而无需手动输入代码。

如果决定自行编写所有内容，首先在 *“Project Explorer”*（项目资源管理器）中通过双击打开 *hal_entry.c* 文件。为了确保程序正常运行，需要定义多个全局变量。首先，在 *hal_entry()* 函数的头部之前声明 USB 驱动程序状态的枚举。其类型应该是 usb_status_t，可以将其命名为 usb_event。接下来，添加一个 usb_setup_t 类型的结构体（在 *r_usb_basic_api.h* 中进行声明），并将其命名为 usb_setup。我们稍后将在解码某些 USB 事件时使用该变量，该变量将在 USB 事件循环内进行初始化。

接下来，我们需要一个变量来保存 USB 模块的编号。将其设置为 uint8_t 类型，命名为 g_usb_module_number，并为其赋值“0x00”。最后，声明类型为 usb_class_t 的 USB 类类型的结构体，将其命名为 g_usb_class_type，并为其赋值“0x00”。如果要了解我们使用的各种类型的详细信息，请参见《Renesas 灵活配置软件包 (FSP) 用户手册》，该手册可以从 FSP 的 GitHub® 网站下载。

添加这些内容后，此部分代码现在应如下所示：

```
/* Global variables for the USB */
usb_status_t usb_event;
usb_setup_t usb_setup;

uint8_t g_usb_module_number = 0x00;
usb_class_t g_usb_class_type = 0x00;
```

我们自己的代码也依赖于一些静态全局变量。请添加到 USB 全局变量下方：

```
/* Global variables for the program */
static char send_str[12] = { "LED on\n\r" };
static volatile uint8_t s1_pressed = false;
static uint8_t led_level = BSP_IO_LEVEL_HIGH;
```

命名为 send_str 的字符数组用于保存我们要通过 USB 发送的文本。将其初始化为“LED on\n\r”，因为将 LED2 切换到“ON”后，将首次使用该变量。下一个变量为 s1_pressed，其类型为 uint8_t，并需要声明为 volatile，因为其值在用户按钮 S1 的回调例程中将更改为 true。默认情况下，其值为 false，将由 IRQ10 中断的回调例程设置为 true，表示已按下该按钮，因此通知主程序该事件已发生。

如果没有将该变量声明为 volatile，C 编译器的优化程序可能不会在每次使用该变量时重新读取其值，因此 hal_entry() 内部的循环可能无法识别到更改。

第三个变量用于保存 LED2 的电平，在启动时应该初始化为 BSP_IO_LEVEL_HIGH。每次激活 S1 时，切换该变量的值。

至此，我们已经声明了所有的全局变量，可以继续编写 hal_entry() 函数内的代码。首先，我们需要一个静态变量，用于保存虚拟 UART 通信端口的设置，如比特率、停止位和数据位的数量以及奇偶校验类型。该变量的类型应该是 usb_pcdc_linecoding_t，建议将其命名为 g_line_coding。将“在此添加您自己的代码”占位符替换为声明。稍后将在 USB 事件处理程序循环中初始化该变量。

```
Static usb_pcdc_linecoding_t g_line_coding;
```

接下来，编写代码以打开并启用外部 IRQ10，将其连接到评估板的 S1。与第 9 章一样，使用 IRQ FSP 驱动程序的相应函数：

```
g_external_irq10.p_api->open (g_external_irq10.p_ctrl,
                              g_external_irq10.p_cfg);
g_external_irq10.p_api->enable (g_external_irq10.p_ctrl);
```

启用中断后，需要打开 USB 并获取类类型和模块编号。为此，使用 *r_usb_basic 上的 g_basic0 USB 驱动程序*模块的相关函数，并将控制结构体传递给这些函数，将引用传递给配置结构体（适用于 Open() 函数）和相关的变量。温馨提示，e² studio 中的代码补全功能和*开发人员帮助*可帮助您编写这些代码行。

```
R_USB_Open (&g_basic0_ctrl, &g_basic0_cfg);
R_USB_ClassTypeGet (&g_basic0_ctrl, &g_usb_class_type);
R_USB_ModuleNumberGet (&g_basic0_ctrl, &g_usb_module_number);
```

中断和 USB 端口的初始化现已完成。接下来编写的所有代码都应该放置在 while(1) 循环内，因为这部分程序将循环执行。首先，我们编写用于获取和处理端口的 USB 相关事件的代码。USB 驱动程序关联多个事件，但为了简洁起见，仅处理 USB_STATUS_REQUEST 事件。如果要全面了解事件处理程序，请参见《灵活配置软件包 (FSP) 用户手册》中的 USB 外设通信设备类 (r_usb_pcdc) 文档。在此，可以找到此类处理程序的代码示例以及流程图。

现在，您的第一个任务是通过调用 R_USB_EventGet() 函数来初始化 usb_event 变量，然后编写处理程序，只有发生 USB_STATUS_REQUEST 事件时才能执行该处理程序。在 if – then – else 结构中，首先设置 USB 端口，然后确定是否请求线路设置。如果是，通过传递 g_line_coding 变量来配置虚拟 UART 设置。

如果否，则查询主机是否要接收 UART 设置。如果是，请将其发送给主机。最后，如果发生事件，在此不进行处理，直接确认。

下面是我们的处理程序版本的完整代码：

```
/* Obtain USB related events */
 R_USB_EventGet (&g_basic0_ctrl, &usb_event);

/* USB event received by R_USB_EventGet */
if (usb_event == USB_STATUS_REQUEST)
{
     R_USB_SetupGet (&g_basic0_ctrl, &usb_setup);

    if (USB_PCDC_SET_LINE_CODING == (usb_setup.request_type
                                     & USB_BREQUEST))
    {
        /* Configure virtual UART settings */
        R_USB_PeriControlDataGet (&g_basic0_ctrl,
                            (uint8_t*) &g_line_coding,
LINE_CODING_LENGTH);
    }
    else if (USB_PCDC_GET_LINE_CODING == (usb_setup.request_type
                                          & USB_BREQUEST))
    {
        /* Send virtual UART settings back to host */
        R_USB_PeriControlDataSet (&g_basic0_ctrl,
                                  (uint8_t*) &g_line_coding,
                                  LINE_CODING_LENGTH);

    }
    else if (USB_PCDC_SET_CONTROL_LINE_STATE == (usb_setup.request_type
                                                 & USB_BREQUEST))
    {
        /* Acknowledge all other status requests */
        R_USB_PeriControlStatusSet (&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
    }
    else
    {
    }
}
```

可以看到，在处理程序中 LINE_CODING_LENGTH 出现两次。由于我们还没有定义 LINE_CODING_LENGTH 的值，请返回到文件的顶部，并将其定义为无符号值 0x07。

```
#define LINE_CODING_LENGTH          (0x07U)
```

返回到 `while(1)` 循环中，添加在激活 S1 后更改 LED2 电平的代码，以 `s1_pressed` 的 `true` 值表示。类似于第 9 章中写入的内容，但此时需要将要通过 USB 发送的字符串复制到 `send_str` 变量，并将 `s1_pressed` 变量设置为 false：

```
if (s1_pressed == true)
{
    s1_pressed = false;

    if (led_level == BSP_IO_LEVEL_HIGH)
    {
        strcpy (send_str, "LED off\n\r");
        led_level = BSP_IO_LEVEL_LOW;
    }

else
    {
        strcpy (send_str, "LED on\n\r");
        led_level = BSP_IO_LEVEL_HIGH;
    }
}
```

最后要添加的代码是用于外部 IRQ10 的回调函数的代码。将其放置在 `hal_entry ()` 函数的括号后面。通过复习第 9 章，了解回调函数的一些详细信息。首先，需要导入 `g_bsp_leds` 结构体，并用其初始化我们的本地 `Leds` 变量。然后将 `s1_pressed` 设置为 true，表示事件已发生，接下来将新值写入引脚寄存器。最后，利用 *r_usb_basic* 模块的 `R_USB_Write()` API，通过 USB 端口发送该字符串。

```
Void external_irq10_callback(external_irq_callback_args_t *p_args)
{
    FSP_PARAMETER_NOT_USED(p_args);
    extern bsp_leds_t g_bsp_leds;
    bsp_leds_t Leds = g_bsp_leds;

    s1_pressed = true;
    g_ioport.p_api->pinWrite (&g_ioport_ctrl,
                              Leds.p_leds[BSP_LED_LED2],
                              led_level);

    R_USB_Write (&g_basic0_ctrl,
                 (uint8_t*) send_str,
                 ((uint32_t) strlen (send_str)),
                 (uint8_t) g_usb_class_type);
}
```

还记得 USB 描述符 `g_usb_descriptor` 吗？现在该描述符将发挥作用。USB 需要有关器件、其配置和供应商信息的准确描述。该文件十分复杂，具有长达 484 行代码。有关该描述符的说明，请参阅《FSP 用户手册》的 *r_usb_basic* 部分，有关如何构建该描述符的详细说明，请参见通用串行总线规范 2.0 版 (**http://www.usb.org/developers/docs/**)。

但这里有两个捷径：一个是在本手册的网站上下载本手册练习的源文件（**www.renesas.com/ra-book**）。另一个是使用 FSP 配置器放置在项目 *ra* 目录下的模板。其名称为 *r_usb_pcdc_descriptor.c.template*，可以在“项目资源管理”中转到 *ra → fsp → src → r_usb_pcdc* 文件夹进行访问（参见*图 10-7*）。将该文件复制到 *hal_entry.c* 所在的 *src* 文件夹中，并将其重命名为 *r_usb_descriptor.c*。修改*供应商 ID* 和*产品 ID*，以便与您自己的产品 ID 相匹配。如果尚未获得这些数据，暂时使用值 `0x045BU` 和 `0x5310U`。到这一步已经完成了要进行的设置和要编写的代码。

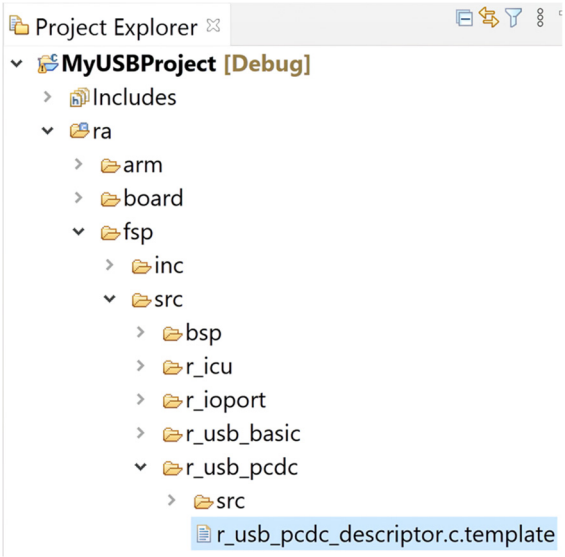


图 10-7：FSP 配置器自动创建 USB 描述符的模板

最后还需要编译项目。第一次执行此操作需要较长时间，因为需要对项目中包含的所有 FSP 模块的代码进行编译。在项目编译完成后，如果没有任何错误和警告，即可连接 EK-RA6M4 评估板并启动调试会话。打开 *“Debug”*（调试）透视图，双击 *“Resume”*（恢复）以启动程序。作为快速测试手段，按一次 S1，以查看 LED2 是否切换。

10.3 在主机端设置接收器

在程序运行的情况下，将第二根 USB type A 转 Micro-B 电缆连接到评估板的*系统控制和生态系统访问*区域左下方标有 J11 的 USB 端口。将另一端插入 Windows® 工作站，稍等片刻，直到 Windows® 识别该电路板，对其进行枚举并安装驱动程序。

启动终端仿真器程序。在本练习的开发过程中，用到了 Tera Term（可从 **https://ttssh2.osdn.jp/** 下载），它是一款非常实用的工具。在 Tera Term 中，可以看到列出的 CDC 串行端口。在*图 10-8* 中显示为 COM3，但在其

他 PC 上可能有所不同。如果不确定，使用 Windows® 的“Device Manager”（设备管理器）来查找电路板所连接的端口。

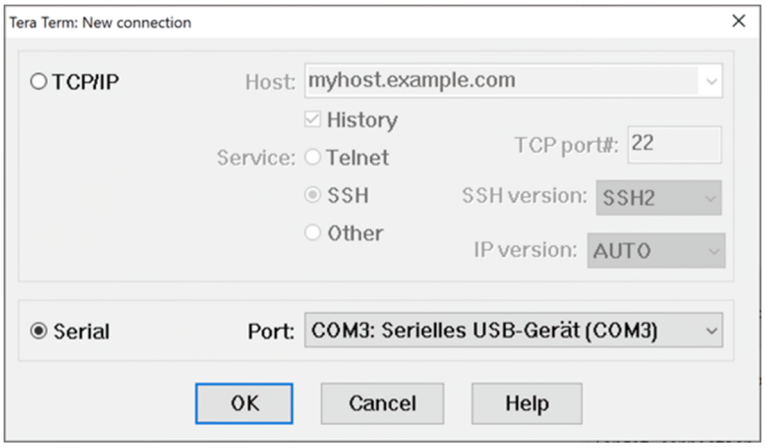


图 10-8：如果 Windows® 正确识别该电路板，它将在 Tera Term 中列为串行连接

如果没有列出该电路板，或者“Device Manager”（设备管理器）指示错误，则驱动程序可能有问题。请参见瑞萨知识库中有关此主题的最新支持条目以解决此问题：
https://en-support.renesas.com/knowledgeBase/18959077。

在已建立连接并运行 Tera Term 的情况下，多次按下 S1，应该可以看到绿色 LED2 切换，其输出到终端的状态如*图 10-9* 所示。

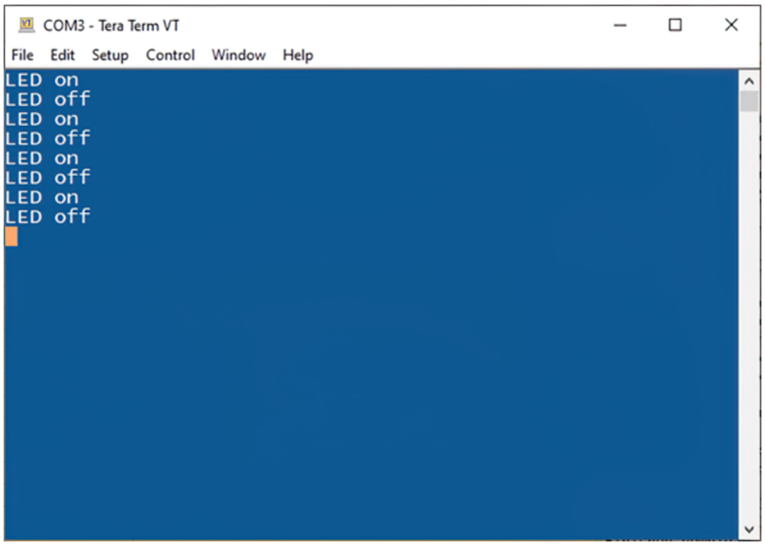


图 10-9：在传输运行的情况下，每次按下 S1 时，终端程序都会显示 LED2 的状态

```
#include "hal_data.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

#define LINE_CODING_LENGTH          (0x07U)

/* Global variables for the USB*/
usb_status_t usb_event;
usb_setup_t usb_setup;
uint8_t g_usb_module_number = 0x00;
usb_class_t g_usb_class_type = 0x00;

/* Global variables for the program */
static char send_str[12] = { "LED on\n\r" };
static volatile uint8_t s1_pressed = false;
static uint8_t led_level = BSP_IO_LEVEL_HIGH;

/*******************************************************************************************/
/* main() is generated by the RA Configuration editor and is used to generate threads if  */
/* an RTOS is used.  This function is called by main() when no RTOS is used.              */
/*******************************************************************************************/
void hal_entry(void) {
    static usb_pcdc_linecoding_t g_line_coding;

    /* open and enable irq10 */
    g_external_irq10.p_api->open (g_external_irq10.p_ctrl, g_external_irq10.p_cfg);
    g_external_irq10.p_api->enable (g_external_irq10.p_ctrl);

    /* Open USB instance */
    R_USB_Open (&g_basic0_ctrl, &g_basic0_cfg);

    /* Get USB class type */
    R_USB_ClassTypeGet (&g_basic0_ctrl, &g_usb_class_type);

    /* Get module number */
    R_USB_ModuleNumberGet (&g_basic0_ctrl, &g_usb_module_number);

    while (true)
    {
        /* Obtain USB related events */
        R_USB_EventGet (&g_basic0_ctrl, &usb_event);

        /* USB event received by R_USB_EventGet */
        if (usb_event == USB_STATUS_REQUEST)
        {
            R_USB_SetupGet (&g_basic0_ctrl, &usb_setup);

            if (USB_PCDC_SET_LINE_CODING == (usb_setup.request_type & USB_BREQUEST))
            {
                /* Configure virtual UART settings */
                R_USB_PeriControlDataGet (&g_basic0_ctrl,
                                          (uint8_t*) &g_line_coding, LINE_CODING_LENGTH);
            }
            else if (USB_PCDC_GET_LINE_CODING == (usb_setup.request_type & USB_BREQUEST))
            {
                /* Send virtual UART settings back to host */
```

```
                R_USB_PeriControlDataSet (&g_basic0_ctrl,
                                          (uint8_t*) &g_line_coding, LINE_CODING_LENGTH);

            }
            else if (USB_PCDC_SET_CONTROL_LINE_STATE == (usb_setup.request_type
                                                         & USB_BREQUEST))
            {
                 /* Acknowldedge all other status requests */
                 R_USB_PeriControlStatusSet (&g_basic0_ctrl, USB_SETUP_STATUS_ACK);
            }
            else {
            }
        }

        if (s1_pressed == true)
        {
            s1_pressed = false;

            if (led_level == BSP_IO_LEVEL_HIGH)
            {
                 strcpy (send_str, "LED off\n\r");
                 led_level = BSP_IO_LEVEL_LOW;
            }

            else
            {
                strcpy (send_str, "LED on\n\r");
                led_level = BSP_IO_LEVEL_HIGH;
            }
        }
    }

#if BSP_TZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*******************************************************************************************/
/* callback function for the S1 push button; writes the new level to LED2 and sends it    */
/* through the USB to the PC                                                               */
/*******************************************************************************************/
void external_irq10_callback(external_irq_callback_args_t *p_args)
{
    /* Not currently using p_args */
    FSP_PARAMETER_NOT_USED(p_args);
    extern bsp_leds_t g_bsp_leds;
    bsp_leds_t Leds = g_bsp_leds;

    s1_pressed = true;
    g_ioport.p_api->pinWrite (&g_ioport_ctrl, Leds.p_leds[BSP_LED_LED2], led_level);

    R_USB_Write (&g_basic0_ctrl,
                 (uint8_t*) send_str,
                 ((uint32_t) strlen (send_str)),
                 (uint8_t) g_usb_class_type);
}
```

恭喜！

您已成功完成本练习！

本章要点：

- 使用 FSP 配置器和 USB 中间件便于增加对 USB 端口的支持。
- 要进行 USB 传输，必须具有 USB 描述符文件。

11 安全性和 TrustZone®

您将在本章中学到以下内容：

- 什么是 TrustZone 以及它如何保护器件。
- Renesas 的 TrustZone 可以带来哪些好处。
- 对应用的安全部分和非安全部分进行分隔的原理。
- 如何实现器件生命周期管理。

安全性至关重要！必须从项目伊始就考虑到安全性，因为无法在后续流程中加入该要素。至少，无法避免费用高昂的重新设计工作，甚至还可能面临要重建整个应用。可将安全性视为建筑物的地基：没有华丽的外表，而且其所需的设计和建造时间可能超过建造其他功能所需的时间总和。而且即便完成搭建，也可能感觉不到它的存在。但是，安全性对于互联环境中的所有应用都至关重要。

什么时候需要确保安全性？产品通过有线或无线接口（如 WiFi、蓝牙、USB 或以太网）进行通信时。存储十分重要的信息（如密钥、证书、密码、个人数据、测量数据或算法）时。产品升级（例如，进行固件更新、功能升级或购买付费服务）时。

因此，安全性涉及各种应用领域。安全性与“智能”世界息息相关，如智能电网、智慧城市、智能楼宇、智能家居、智能照明、智能手表、智能电器或智能服装等。它还会影响到物联网或工业应用，因为传感器、机器人技术、楼宇自动化或农业也需要确保安全性。还有医疗应用，人命关天，必须谨慎对待安全问题。

瑞萨的 RA 产品家族微控制器支持以多种方式来创建安全应用：通过搭载 Arm® Cortex®-M4、-M23 或 -M33 内核的器件上的安全加密引擎，通过搭载 Cortex-M4 或 -M23 内核的器件上的安全内存保护单元，也可以通过搭载 Cortex-M33 内核的 MCU 上瑞萨对 Arm 的 TrustZone®的自有实现。本章将讨论最后一种方式。

最后值得一提的是：大多数 RA 产品家族 MCU 均已通过 PSA Certified™ 1 级认证，其中 RA6M4 MCU 系列已通过 PSA Certified™ 2 级认证。其中包括对采用 Trusted Firmware M (TF-M) 的 RA 产品家族灵活配置软件包进行 PSA 功能 API 认证。多个 RA 产品家族 MCU 系列已通过 SESIP1 认证，无论选择何种软件平台，都能确保实现基本的安全功能。此外，安全加密引擎获多项 NIST CAVP 认证，可确保各种加密功能正常发挥作用。从而增强了用户面向互联环境开发安全产品时的信心。

11.1 什么是 TrustZone®，它有什么作用？

TrustZone® 是 Arm® 开发的一项核心技术，作为 Arm v8-M 架构的一部分，通过强制硬件隔离提供系统级安全措施。TrustZone® 要求在软件中划分安全和非安全 MCU 区域对应的逻辑分区，从而创建受保护的环境。该技术通常在搭载 Arm® Cortex®-M33 内核的器件上实现，个别情况下也在搭载 Arm Cortex-M23 内核的器件上实现。

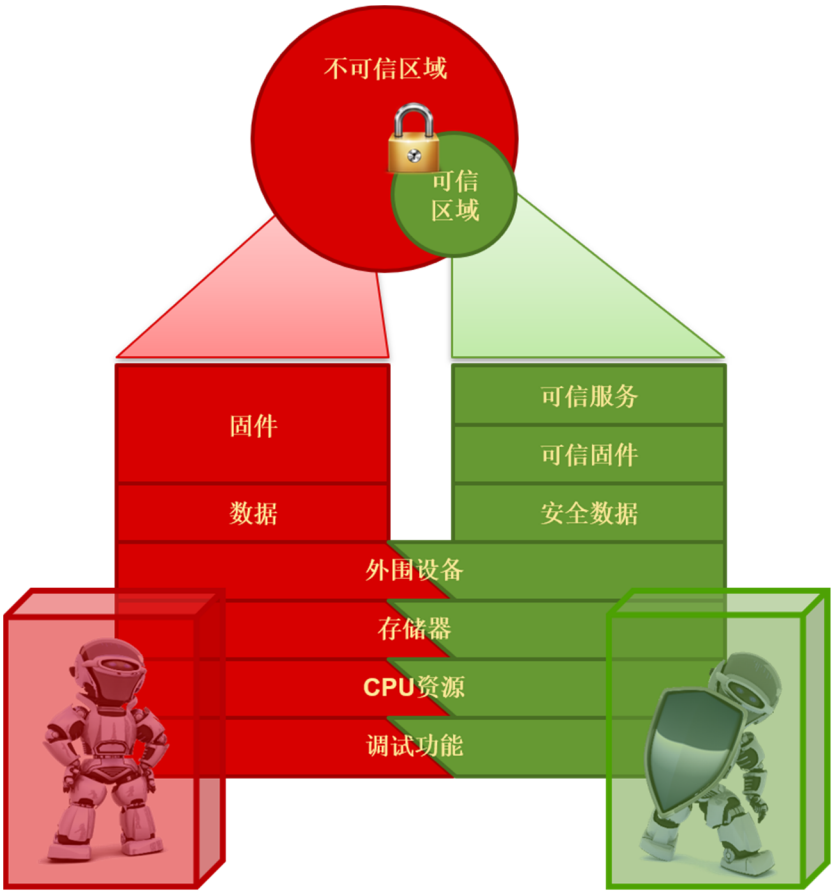


图 11-1：TrustZone® 可以将微控制器的资源划分为非安全区和安全区

内存区域划分为三个不同的分区：一个安全分区，用于密钥存储和数据解密等受信任或受保护的 IP；一个非安全分区，用于常规应用；以及一个非安全可调用分区，用作其他两个分区之间的网关。通过最后一个分区，位于非安全分区中的代码可以调用安全分区中的服务（参见*图 11-2*）。此功能可以通过跳板 (veneer) 实现，支持隔离安全分区和非安全分区。

使用 TrustZone 时，非安全环境（非安全处理环境 (NSPE)）中的代码无法直接访问安全环境（安全处理环境 (SPE)）中的内容。举例来说，这意味着非安全环境不能更改或使用安全环境中的密钥或证书。只有位于安全环境中的代码才能进行此操作。这可以实现 IP 保护和沙盒，并减少关键组件的攻击面，同时还可简化嵌入式器件的安全评估。因此，如果微控制器中存在需要保护的知识产权 (IP)，请将其放置在安全环境中！

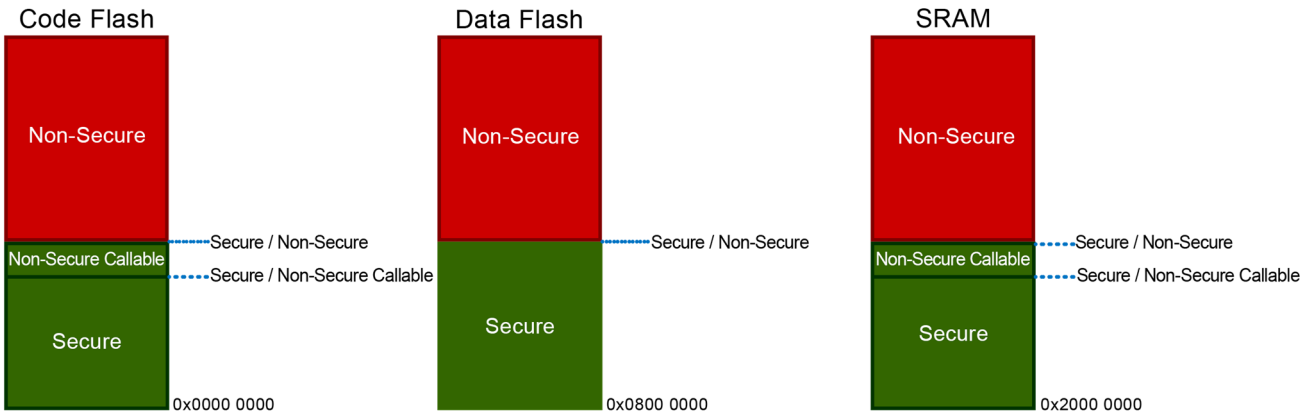


图 11-2：TrustZone 支持对安全区域和非安全区域进行隔离

在安全和非安全状态下都可以访问非安全环境中的函数和数据，而非安全环境中的例程只能通过调用位于非安全可调用环境 (NSC) 的跳板来访问安全环境中的服务。通过此方式提供一个进入安全环境的确定入口点。

虽然 Arm 对 TrustZone 的原始定义适用于闪存、RAM 和内置组件（如 SysTick 定时器或 MPU），并要求将应用程序代码和数据与安全项目分离，但其不包括对 DMA 控制器或外设分离的保护。

瑞萨扩展了这一概念，并将 TrustZone 过滤器应用于其他总线以及引脚和外设。总线过滤器可防止非安全代码通过 DMA、DTC 和类似机制提取安全代码和数据。换句话说：它可以防止外设访问不应该访问的内存。另一方面，应用于引脚和外设的 TrustZone 过滤器允许锁定引脚，使其仅可供安全环境访问，而不支持从任何位置进行访问。

这不仅可保护外部接口，而且还能防止非安全代码窃听引脚。例如，位于非安全环境中的恶意代码不能监听分配给安全环境的引脚状态，也不能重建通过 UART 进行的传输内容。非安全代码也不能覆盖输出。

瑞萨还确保在启动时没有安全漏洞。其他制造商在其实现中使用软件配置的安全属性单元 (SAU)，这导致 TrustZone 在复位后短时间内处于未配置状态，瑞萨决定在其 RA 产品家族中使用实现定义属性单元 (IDAU)，通过非易失性寄存器配置。这意味着，在获取复位向量时，也就是在执行任何应用程序之前，已经配置了 TrustZone 并且保证器件是安全的。安全属性通过 SCI（片内工厂）引导模式在非易失性内存中设置，应用程序无法修改其内容，只能读取其内容。

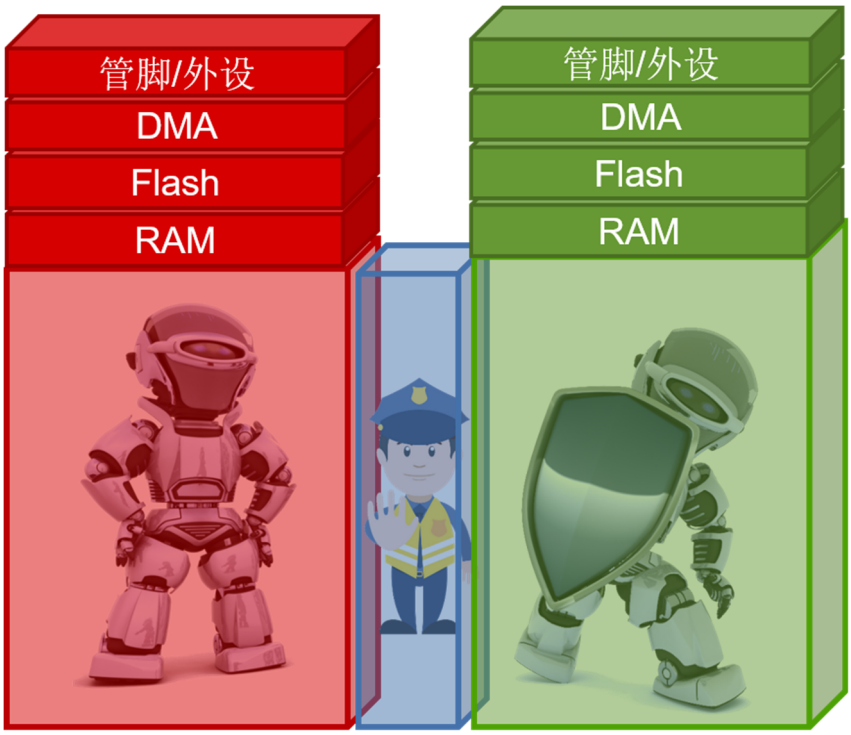


图 11-3：Renesas 的 TrustZone 实现不仅将过滤器应用于内存，还将其应用于其他总线、引脚和外设

还需要注意：仅通过使用 TrustZone 不能实现应用的安全。仅使用 TrustZone 并不意味着该器件已具备安全保护。TrustZone 只是安全工具箱中可选择的一款工具，必须以正确的方式使用才能发挥作用。如果安全应用程序本身存在漏洞（如不合规范的代码），则 TrustZone 自身并不能避免被攻击。安全代码必须验证其正确性和可信度。

但 TrustZone 有助于提供并执行信任根 (RoT)。RoT 是始终可信赖的源，也是所有安全操作、数据和算法的基础。它不仅使代码和数据完整性和隔离、系统验证、薄膜或身份验证之类的服务可用于当前或其他受信任的系统或器件，而且还包含密钥、证书或密码。但这也意味着，如果信任根遭到破坏，则完全丧失保护功能。

TrustZone 通过将敏感数据存储在安全环境中来对数据和固件提供保护。它还可以将关键软件预装为安全固件，并将外设配置为仅允许从安全环境访问，从而提供操作保护。

有关 TrustZone 的更多信息和更详细的解释，请参见 Arm 的文档 (**https://developer.arm.com/ip-products/security-ip/trustzone**)，但请注意，Arm 通常采用术语“受信任”和“非受信任”，其定义直接对应于“安全”和“非安全”。

11.2 安全环境和非安全环境的划分

现在我们已经清楚，程序需要划分为安全和非安全环境，我们该如何相应地对软件进行分区？为此，基于 TrustZone® 的系统始终包含两个不同的项目：一个安全项目，另一个是非安全项目。二者都可以利用 SRAM 以及代码和数据闪存，但只有安全代码可以直接访问两个（安全和非安全）分区。

可借助 e² studio 中的项目配置器设置这些项目。创建新项目后，系统将提示您选择项目应具有的类型：

- 扁平化（非 TrustZone）项目
- TrustZone 安全项目
- TrustZone 非安全项目

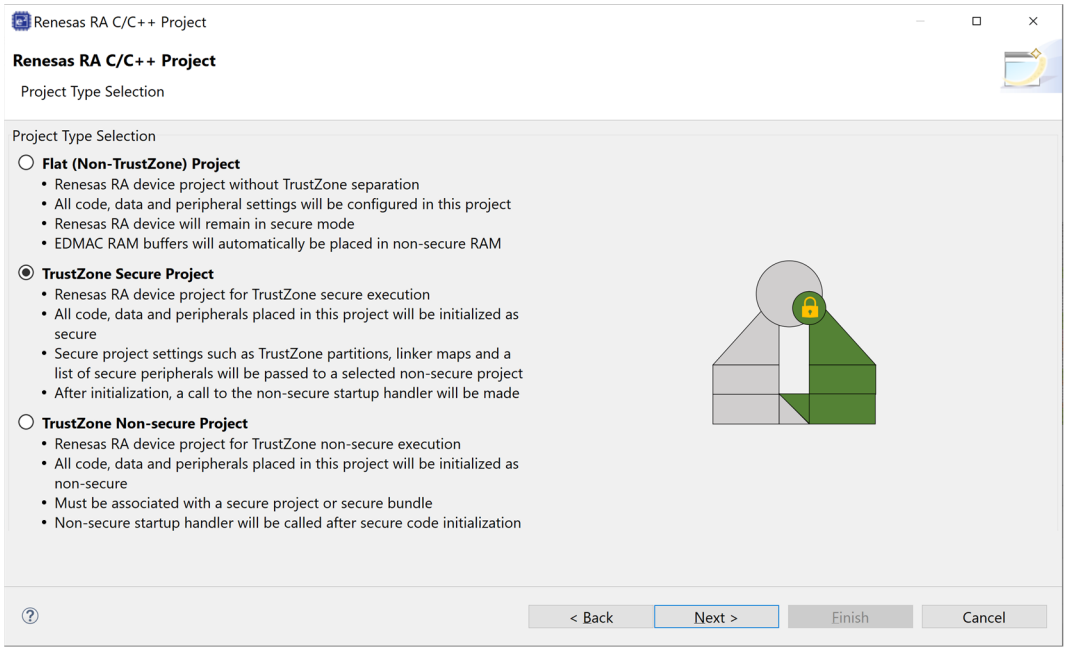


图 11-4：通过项目配置器的“Type Selector”（类型选择器）页面，可以在扁平化、安全和非安全项目之间进行选择

请注意，如果选择扁平化（非 TrustZone）项目，则微控制器将在引导后保持安全模式。另外，在设置 TrustZone 项目时需要格外小心，以确保正确管理安全和非安全分区之间的连接。在项目配置器中创建非安全项目时，将非安全项目与安全项目或捆绑包进行关联以实现此要求。在 *“Project Type Selection”*（项目类型选择）界面上选择 *“TrustZone Non-secure Project”*（TrustZone 非安全项目）并单击 *“Next”*（下一步）后，e² studio 将要求您为非安全项目指定一个对应的安全项目。

创建安全项目后，即可将安全的堆和驱动程序提供给非安全环境。为此，右键单击最上面的模块，从弹出的菜单中选择 *“Non-secure Callable”*（非安全可调用）。选择该条目后，请注意左侧的小箭头：它指示此模块现在为非可安全调用模块（参见*图 11-* **5**）。

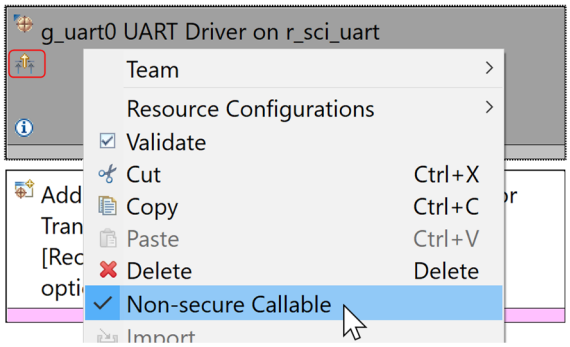


图 11- 5：可以将安全项目中堆栈的最顶层模块设为非安全可调用模块

最后一点是，也可以在 e² studio 内对内存进行分区：转到 *"Run → Renesas Debug Tools -> Renesas Device Partition Manager"*（运行 → 瑞萨调试工具 → 瑞萨器件分区管理器），将运行一个实用程序。器件分区管理器可以在开发期间执行生命周期状态管理，另外还允许设置和查询 IDAU 区域，以及解锁已擦除的闪存模块。

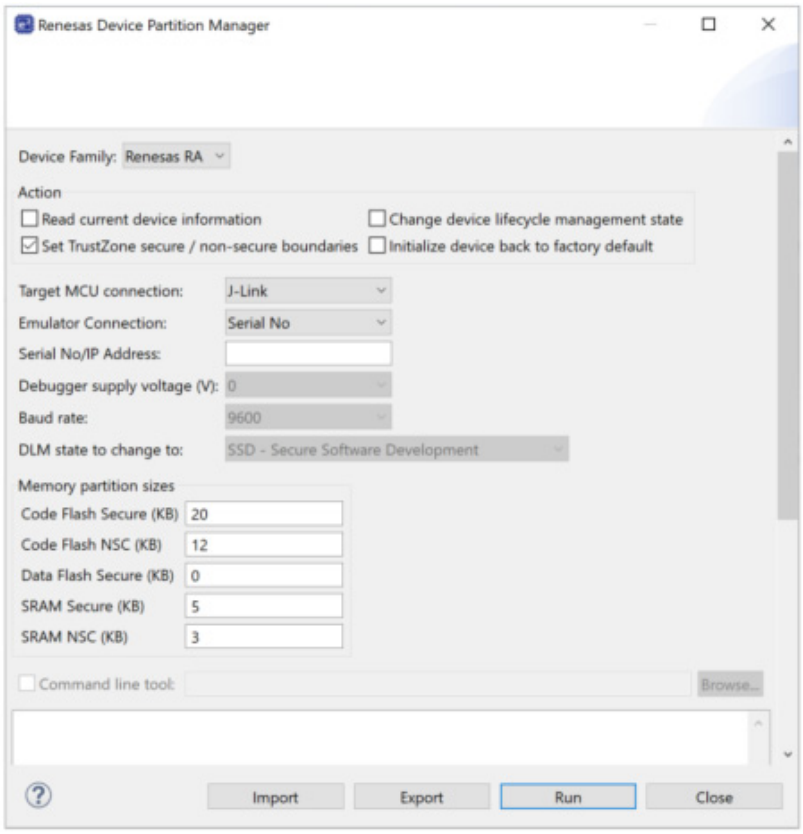


图 11- 6：器件分区管理器可以定义各种内存分区的大小

如果要深入了解用于配置 RA 产品家族微控制器专用的 Arm TrustZone 的工具和相关工作流程：Renesas 的网站 (**https://www.renesas.com/cn/zh/document/apn/ra-arm-trustzone-tooling-primer**) 上提供了 RA Arm TrustZone 工具入门，介绍有关使用工具和设置项目的基础知识。

11.2.1 跨边界的函数调用

现在，如果位于非安全环境的应用程序的一部分要调用位于安全环境的闪存外设，以对非安全数据闪存进行编程，会发生什么情况？为此，Arm® v8M Cortex®-M33 内核的指令集中添加了一条新指令：SG 或安全网关。该指令必须位于内存的安全和非安全部分之间的非安全可调用 (NSC) 区域。这可确保即使在安全环境的其他位置找到 SG 操作码，也不能将其用作入口点。在 SG 指令之后，可以对安全端的代码进行调用（参见*图 11-7*）。

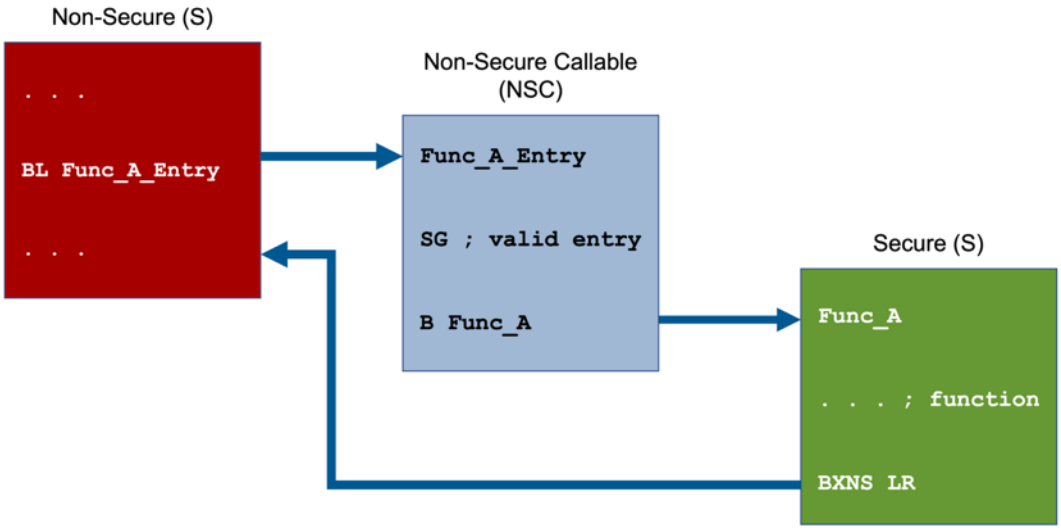


图 11-7：从非安全环境中调用安全函数

将通过 BXNS LR（BXNS = 跳转并交换到非安全状态）操作码从安全端返回到非安全端，此操作将跳转到 BL Func_A_Entry 分支期间放置在链接寄存器 (LR) 中的地址。在函数返回时，函数的返回状态被存储在 LR 中返回地址的 LSB 中。这一位的数值将和返回到调用函数时的状态进行比较，以防止从非安全代码调用的安全 API 返回到一个指向安全地址的假返回地址。

在第一条指令不是 NSC 区域中的 SG 操作码的情况下，执行位于非安全环境中的代码对安全环境中的代码的调用，则在带 CM33 内核的微控制器上会发生安全故障。将在安全状态下处理该故障。

还可以从安全代码调用非安全代码，但不建议这样做，因为这有可能导致数据泄露，导致安全问题。安全代码可以通过参数将某些寄存器值传输到非安全环境，并且编译器将从其余寄存器中清除其他安全数据。该机制还隐藏了安全软件的返回地址，从而确保非安全环境中的代码不会操纵返回地址（参见*图 11-8*）。

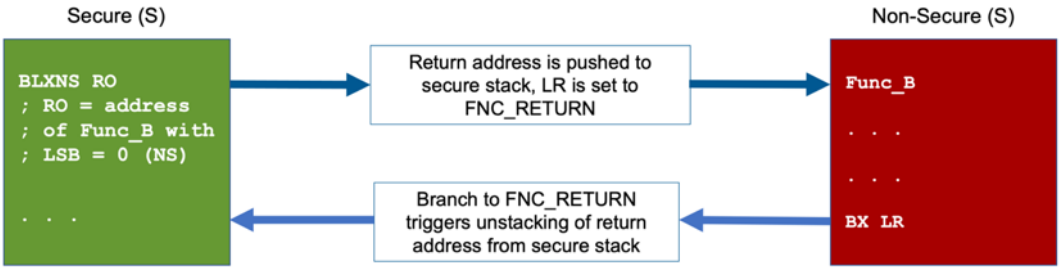


图 11-8：从安全环境中调用非安全函数

通过 BLXNS（通过链接跳转并交换到非安全状态）指令从安全代码调用非安全代码的推荐方法是：在第一次启动时初始化安全环境中的代码，然后将程序控制传递给非安全环境。此后，从非安全环境到安全环境的任何数据传输均应通过 FSP 回调进行管理。

11.2.2 从安全代码到非安全代码的回调

外设位于安全环境，但提供中断服务的回调函数位于非安全环境，我们该如何处理这种情况？通常，在 FSP 回调中，回调结构体由 ISR 在堆栈中分配，然后将由回调函数使用。由于中断服务程序 (ISR) 和回调函数位于不同的环境中，如果回调函数试图访问安全环境中的回调结构体，将发生安全故障。

FSP 通过将回调结构体分配到两种环境都可以使用的内存区域解决了这个问题。通过使用 `callbackSet()` API 对其进行初始化，这是一个允许访问安全环境的保护函数（有关保护函数的介绍，请参见第 0 章）。该调用如下所示：

```
fsp_err_t (* callbackSet)(uart_ctrl_t * const p_ctrl,
                          void (* p_callback) (uart_callback_args_t *),
                          void const * const p_context,
                          uart_callback_args_t * const
p_callback_memory);
```

在模块的配置结构体中已经提供了回调函数指针和上下文指针，但二者必须再次创建，因为安全端的配置结构体与非安全端的结构体是分开构建的。指向易失性回调内存的指针指向某个位置，可在该位置将内存分配给可从两种环境访问的结构体。这样便可消除安全故障。

11.2.3 保护函数

保护函数的应用程序编程接口 (API) 允许从非安全项目访问位于安全环境中的驱动程序。瑞萨实现的这项功能是独一无二的，并且正在申请专利。灵活配置软件包 (FSP) 将自动为 FSP 配置器中标记为非安全可调用的所有堆栈顶部模块和/或驱动程序 API 生成保护函数，并将其添加到 NSC 区域的项目中。此外，FSP 将为相应的 NSC 实例创建非安全模块实例。

这些实例的使用方法没有特别之处，但其 p_ctrl 和 p_cfg 成员设置为 FSP_SECURE_ARGUMENT，相当于 NULL，并且其 p_api 成员指向保护函数而不是实际的成员函数。保护函数本身将 p_ctrl 和 p_cfg 成员硬编码到安全环境内存中。在安全和非安全内存中都存在驱动程序，并在不同端使用不同通道的情况下，这可消除通过操纵 p_ctrl 和/或 p_cfg 结构体以直接从非安全代码访问安全通道的可能性。保护函数还会检查所有输入指针，以确保调用方不会覆盖安全内存。

```
const uart_api_t g_uart0_api = {
       .open               = guard_g_uart0_R_SCI_UART_Open,
       .close              = guard_g_uart0_R_SCI_UART_Close,
       .write              = guard_g_uart0_R_SCI_UART_Write,
       .read               = guard_g_uart0_R_SCI_UART_Read,
       .infoGet            = guard_g_uart0_R_SCI_UART_InfoGet,
       .baudSet            = guard_g_uart0_R_SCI_UART_BaudSet,
       .versionGet         = guard_g_uart0_R_SCI_UART_VersionGet,
       .communicationAbort = guard_g_uart0_R_SCI_UART_Abort,
       .callbackSet        = guard_g_uart0_R_SCI_UART_CallbackSet,
};
/* Create non-secure instance that has a non-secure callable API */

const uart_instance_t g_uart0 =
{
       .p_ctrl = FSP_SECURE_ARGUMENT, // CTRL is static in guard
function
       .p_cfg  = FSP_SECURE_ARGUMENT, // CFG is static in guard
function
       .p_api = &g_uart0_api
};
```

此外，如果设计人员希望仅向非安全程序员公开有限范围的 API，则可以选择添加访问控制的附加级别或删除保护函数。继续以 SCI 为例，安全环境的程序员可以打开一个通道并配置为所需的波特率，但通过删除 g_uart0_write_guard() API 之外的所有 API，使得只有写 API 可供非安全应用程序的开发人员使用。

11.3 器件生命周期管理

器件生命周期定义了器件寿命的不同阶段，并控制调试接口、串行编程接口和瑞萨测试模式的功能。出于安全原因，这样便可将位于安全环境中的代码的编写与在非安全环境中运行的应用程序的编写分开，并且可以由两个独立的团队来开发产品：由安全开发人员团队创建信任根 (RoT) 或孤立的子系统，并由非安全环境的设计人员创建使用该 RoT 或子系统的应用程序。灵活配置软件包 (FSP) 和 e² studio 支持这种设计划分。

安全环境的代码准备就绪后，可将其预先烧录到器件中，并将生命周期设置为 NSECSD，从而锁定安全环境，或者作为捆绑包由非安全项目引用。应用程序设计人员将以此作为起点，在非安全环境中编写应用程序，对其进行调试，然后将其烧录到器件中。如果需要，他们还可以禁用所使用的闪存模块的编程和擦除功能。最后一步，将生命周期的状态设置为已部署、调试锁定或引导锁定。这样一来，整个器件可获得保护，并且不能对编程接口和器件进行调试、读取或编程。*图 11-9* 显示了可能的状态和转换，而*图 11-10* 中的表格提供了对每个生命周期的解释。

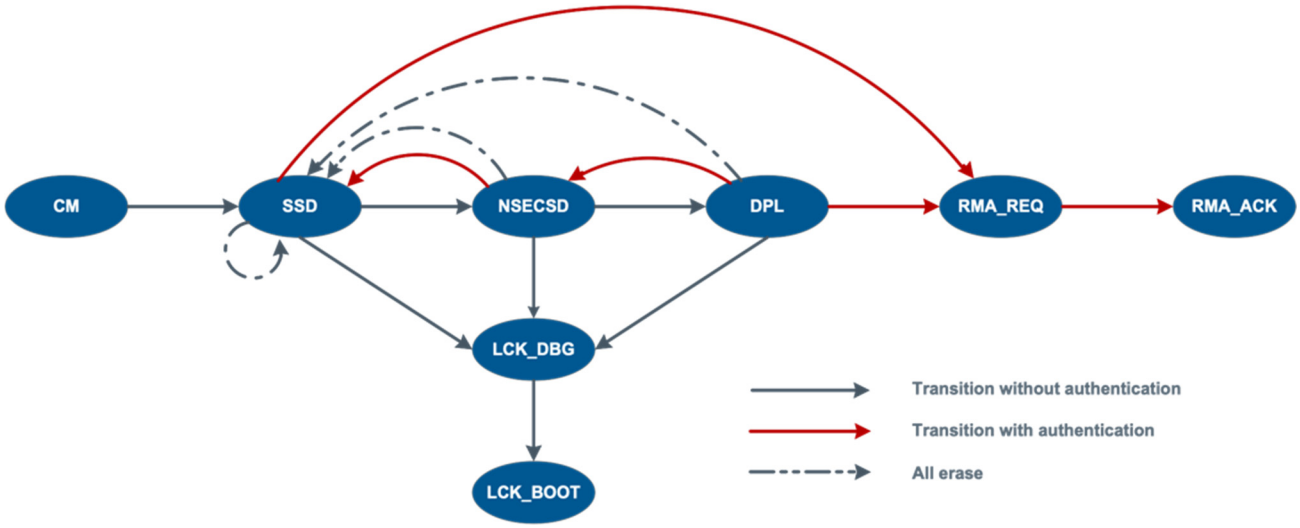


图 11-9：器件生命周期管理的不同状态

有三种不同的调试访问级别，它们视生命周期的状态而变化：

- **DBG2：**允许调试器连接，访问存储器和外设没有任何限制。
- **DBG1：**允许调试器连接，但访问只限于非安全内存区域和外设。
- **DBG0：**不允许进行调试连接。

生命周期	定义	调试级别	串行编程	测试模式
CM	“Chip Manufacturing” “芯片制造” 客户收到器件时的状态。	DBG2	可用，无法访问代码/数据闪存	不可用
SSD	“Secure Software Development” “安全软件开发” 该应用程序的安全部分正在开发中。	DBG2	可用 可以编程/擦除/读取全部代码/数据闪存区域	不可用
NSECSD	“Non-Secure Software Development” “非安全软件开发” 该应用程序的非安全部分正在开发中。	DBG1	可用 可以编程/擦除/读取非安全代码/数据闪存区域	不可用
DPL	“Deployed” “已部署” 器件已经投放市场。	DBG0	可用 无法访问代码/数据闪存区域	不可用
LCK_DBG	“Lock Debug” “锁定调试” 调试接口已永久禁用。	DBG0	可用 无法访问代码/数据闪存区域	不可用
LCK_BOOT	“Lock Boot Interface” “锁定引导接口” 调试接口和串行编程接口已永久禁用。	DBG0	不可用	不可用
RMA_REQ	“Return Material Authorization Request” “申请退货授权 RMA”申请 RMA。 在该状态下，客户必须将器件发送给瑞萨。	DBG0	可用 无法访问代码/数据闪存区域	不可用
RMA_ACQ	“Return Material Authorization Acknowledged” “退货授权 RMA 已确认” 在瑞萨进行故障分析	DBG2	可用 无法访问代码/数据闪存区域	可用

图 11-10：器件周期管理不同阶段的描述

可以使用瑞萨闪存编程器或瑞萨器件分区管理器执行从一种状态到另一种状态的转换，但后者可选择的状态有限。可以通过使用身份验证密钥来保护状态之间的转换。有关不同 DLM 状态和器件特定转换的更多信息，请参见相应微控制器的用户手册。

11.4 TrustZone® 用例

现在我们已经了解了什么是 TrustZone®，它是如何帮助实现数据和知识产权 (IP) 保护的，瑞萨实施该技术可以带来哪些好处，以及安全和非安全环境的划分情况如何，接下来介绍一些具体用例。本章的以下部分将介绍 TrustZone 如何协助保护 IP，支持法律相关代码的隔离，以及保证信任根 (RoT) 的安全。

11.4.1 预烧写算法的 IP 保护

如果应用程序必须访问受到防篡改保护的功能算法，那么对安全算法与其余代码分别进行开发的可能性，将为客户带来巨大的利益。算法的设计人员将首先使用 e² studio 中的项目和 FSP 配置器创建定义好应用程序编程接口 (API) 的安全项目，编写算法，并使用任何可用的调试接口对其进行调试。然后，如果需要，可以将其烧写到微控制器中，如有必要，还可以通过禁用已使用的闪存区块的编程或擦除功能来对其进行保护。安全项目将自动配置 TrustZone。在将预烧写的器件交给应用开发人员之前，安全团队会将生命周期状态设置为非安全软件开发 (NSECSD)，以使调试器或闪存编程器无法读取该算法。

然后，应用程序编写人员将在 e² studio 中创建一个非安全项目，编写其应用程序并使用任何调试接口对其进行调试。他们的应用程序可以顺利地调用任何安全项目的已公开 API。完成后，将最终代码烧录到微控制器中，禁用所用闪存区块的编程或擦除功能，并将器件生命周期状态设置为已部署 (DPL)、调试锁定 (LCK_DBG) 或引导锁定 (LCK_BOOT)。现在，整个装置都受到保护，可以随时发给客户。

TrustZone 在此提供的最大优势是其可以防止算法滥用（无论是否有意），并允许将应用程序设计划分为安全和非安全方。但是，如果安全算法中存在缺陷，需要纠正，该怎么办？在*图 11-9* 中可以看到，如果安全开发人员未禁用擦除功能，则可以通过擦除受保护的算法回到 SSD 状态。这可最大限度减少预烧写器件的报废率。

11.4.2 智能电表中法律相关代码的代码分离

欧洲的智能电表规范定义了法律相关代码，该代码已通过认证。该代码必须与电表的其他部分隔离。目前，大多数客户通过使用两个微控制器来进行物理隔离。这样做费用高昂，但可简化认证。

另一种方法是在使用支持 TrustZone 的单片机上对法律相关代码、数据和外设以及应用代码进行逻辑分隔，如用于显示的代码或 DLMS/Cosem（设备语言消息规范/能源计量的配套规范）。这样一来，TrustZone 将提供可验证的隔离，并防止单个器件上的代码滥用和损坏。

11.4.3 保护信任根

正如第 0 章的说明，信任根 (RoT) 奠定了整个产品的安全基础，因此必须得到保护。所有更高级别的安全都建立在 RoT 之上，RoT 可提供经过身份验证的固件更新和安全通信。此外，RoT 能够从更高级别的安全故障中恢复，但是，如果 RoT 遭到破坏，则以它为基础的任何内容都将不再安全。

这意味着所有出厂密钥、器件身份、任何校验和、闪存映像验证、加密服务、密钥和证书以及敏感数据都应保存在安全的环境中。其他所有内容，例如主应用程序、用户接口、接口协议的不安全元素、服务以及其他内容，都应放置在非安全环境中。为了尽可能减小安全环境的攻击面，应将整个应用程序中尽可能多的内容放置在非安全环境中。

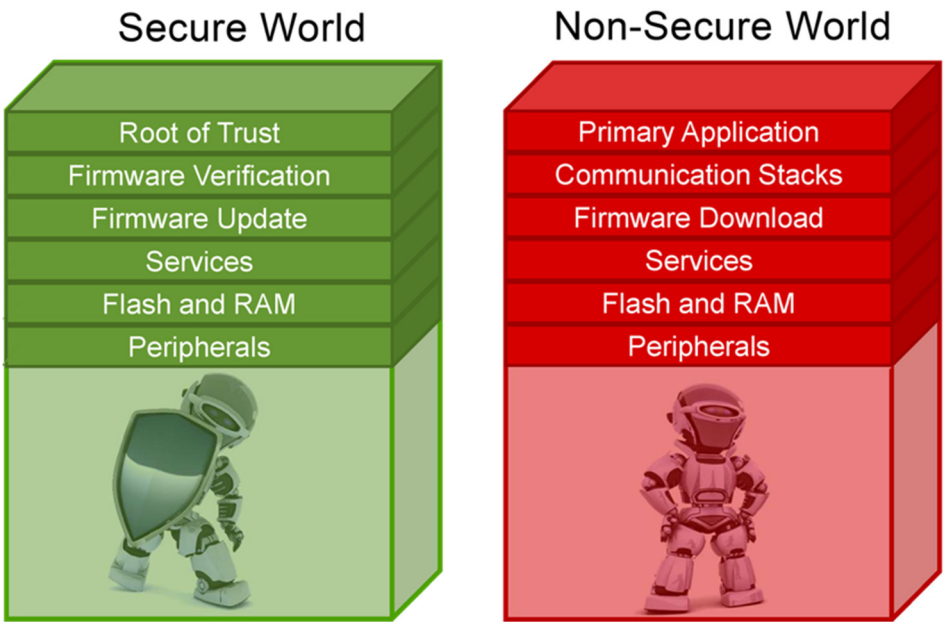


图 11-11：在安全和非安全环境中要保存的内容

本章要点：

- TrustZone 简化了安全和非安全环境的分隔。
- Renesas 的 TrustZone 实现确保在启动时没有安全漏洞。
- 有两个不同的项目必不可少：一个是安全项目，另一个是非安全项目。
- 非安全可调用分区允许通过保护函数调用安全区域的内容。
- 生命周期管理有助于在不同团队之间拆分开发流程。

12 补充资源

您将在本章中学到以下内容：

- RA 合作伙伴生态系统的介绍以及它缩短开发时间的方法。
- 获取示例项目的方法。
- 获取更多在线培训、白皮书和应用笔记的方法。

到目前您已经学习了 RA 产品家族微控制器的基本功能、如何为其编写程序以及可用的软件和硬件工具支持，您可能还希望了解更多内容。为了满足您的需要，本章将为您提供瑞萨或其合作伙伴提供的其他支持的链接和提示。

12.1 瑞萨 RA 合作伙伴生态系统

在过去的几年里，嵌入式系统的设计日益复杂，随之而来的是各种设计问题同时涌现各种新技术。随着开发难度的不断增加同时面临不断压缩的开发时间，边缘或 IoT 设备的开发人员需要付出更多的努力才能按时交付项目。在软件或硬件两方面，对灵活的即用型解决方案和构建模块的需求日益增加。

瑞萨的 RA 合作伙伴生态系统拥有 50 多个合作伙伴，可提供一系列第三方应用程序和构建模块，从而有效满足市场需求。通过提供各种开箱即用的即插即用型选件，该生态系统可帮助工程师加快面向物联网 (IoT) 领域的嵌入式系统的开发速度。这些选项包括安全功能、语音用户界面、图形、机器学习、人机界面和云连接等。每个新合作伙伴的构建模块解决方案都标有“RA READY”徽章，表明该产品旨在解决客户的实际问题。有关详细信息，请访问合作伙伴生态系统的网站：**https://www.renesas.com/ra-partners**。

图 12-1：Renesas RA 合作伙伴生态系统包括 50 多个合作伙伴提供的即用型软件和硬件组件

12.2 示例项目

为了帮助您加快开发速度并按时交付项目，瑞萨提供了多个示例项目，您可以免费下载和使用。《灵活配置软件包用户手册》提供了每个 API 和模块的简短示例，此外还提供了大量示例项目捆绑包，可随时在您选择的评估板上运行。对于这些捆绑包，均配有使用指南，其中介绍了如何安装这些工具以及如何导入和运行这些项目。

这些捆绑包包括适用于大多数片上外设（如 A/D 转换器、闪存外设、USB、SPI 或 UART 连接和 FreeRTOS™，Azure®RTOS 等）的示例项目。捆绑包内的每个项目都带有一个自述文件，列出所提供的功能、硬件要求、跳线设置和硬件连接。

还有其他示例项目，例如 IEC 60730/60335 自检库、电机控制软件或传感器网络解决方案的示例代码。可以从评估板的主页或从 RA 微控制器的主页获取下载文件。

12.3 在线培训、白皮书和应用笔记

在本手册的最后，可能有人会问“我在哪里可以学习到 RA 产品家族微控制器的更多知识？”。如果要进一步深入了解该主题，我们提供了丰富的信息、培训和教程供您自学。本章的以下段落将重点介绍其中的一部分，对于其他资料在此不做介绍，大部分资料的链接可以在 RA 产品家族网站 (**https://www.renesas.com/ra**) 的 *“Support”*（支持）下找到。

12.3.1 在线培训

瑞萨学院是一个在线学习平台，该平台将所有学习资料集中在一起，方便读者使用，您可以根据自己的时间安排随时随地学习所需知识。该平台提供的课程不仅涵盖 RA 产品家族微控制器，还包括瑞萨的其他产品系列，如 Synergy、RX、RZ 或模拟和电源产品。

您可以上网并在课程目录中快速找到所需的培训。每门课程细分为多个小模块，您可以根据自己的时间表合理安排学习计划。只需在瑞萨学院网站 (**https://academy.renesas.com**) 上完成注册，即可随时接受任何培训。最新课程包括 RA 产品家族的各种微控制器的介绍和 RA 合作伙伴生态系统的培训等主题。各类全新和更新的课程、内容和功能将不断推陈出新，敬请随时关注。

12.3.2 白皮书和应用笔记

Renesas 提供了大量的白皮书和应用笔记，涵盖了硬件和软件设计主题、安全性、IoT 连接、电机控制、人机界面等。所有这些资料将帮助您解决设计问题，以免您走弯路，因此将加快您的开发速度。示例项目可以从微控制器页面或所使用的评估板页面进行访问。另外，RA 产品家族的主页是一个最佳切入点。

如果要浏览瑞萨提供的所有白皮书和应用笔记，可以转到以下页面并输入对应的部件编号作为过滤条件：**https://www.renesas.com/eu/en/support/document-search?doc_category_tier_1=&doc_category_tier_2=&doc_category_tier_3=&doc_category_tier_4=&doc_part_numbers=&title=&sort_order=DESC&sort_by=field_document_revision_date#documentation-tools-results**。

本章要点：

- 示例项目涵盖 RA 微控制器的综合应用或某些方面。
- 在线视频提供了熟悉工具和器件的捷径。
- RA 合作伙伴生态系统提供了来自第三方的即用型构建模块。
- 白皮书和应用笔记涵盖了广泛的主题。

关于作者：

Richard Oed 在嵌入式处理领域担任现场应用和系统工程师已超过 23 年，为 DSP、微控制器、微处理器和数据转换器提供支持及编写软件。自 2015 年秋开始，他走上了自由撰稿人、作家和记者之路。Richard 是三项专利的共同发明人，并且是 IEEE、ACM 和 VDE 的成员。他也是《瑞萨 Synergy 平台基础知识》的作者，该手册可通过访问瑞萨网站 (**www.renesas.com/synergy-book**) 获取。

在购买或使用本文所列的任何 Renesas Electronics 产品之前，请先查阅最新的产品手册和/或数据表。

Renesas Electronics Corp.
www.renesas.com

文档编号 R01TU0337CD0200