

ModusToolbox 上 PSoC 6 MCU 入门

作者: Srinivas Nudurupati

相关器件系列: 所有 **PSoC® 6 MCU** 器件

软件版本: **ModusToolbox™ 2.0**

相关应用笔记和代码示例: [点击这里](#)

更多代码示例? 我们倾听到了你的声音。

要查看持续增长的成百上千的使用 ModusToolbox IDE 的 PSoC 代码示例, 请访问 [Cypress GitHub](#) 站点, 您也可以在此[浏览 PSoC 视频库](#)。

AN228571 介绍 PSoC 6 MCU, 这是一款采用 Arm® Cortex®-M4 和 Cortex-M0+处理器的双 CPU 可编程片上系统。本应用笔记可帮助您探索 PSoC 6 MCU 架构和开发工具, 并向您展示如何使用 ModusToolbox 创建第一个项目。本应用笔记还指导您获取更多在线资源, 以加快您对 PSoC 6 MCU 的学习。

目录

1 简介	1	4.2 关于设计	10
1.1 前提条件	2	4.3 第 1 部分:创建新应用:	11
2 开发生态系统	3	4.4 第 2 部分: 查看与修改设计	14
2.1 PSoC 资源	3	4.5 第 3 部分: 写固件	17
2.2 固件/应用开发	3	4.6 第 4 部分: 构建应用程序	22
2.3 支持其他 IDE	7	4.7 第 5 部分: 对设备进行编程	23
2.4 FreeRTOS 对 ModusToolbox 的支持	8	4.8 第 6 部分: 测试您的设计	24
2.5 编程/调试	8	5 总结	25
2.6 PSoC 6 MCU 开发套件	8	6 相关应用笔记和代码示例	26
3 器件特性	9	Appendix A. 术语表	27
4 我的第一个使用 ModusToolbox IDE 的 PSoC 6 MCU 设计	10	修订记录	28
4.1 使用这些说明	10	销售、解决方案以及法律信息	29

1 简介

PSoC 6 MCU 是赛普拉斯的超低功耗 PSoC 器件, 具有针对智能家居, 物联网网关等量身定制的双 CPU 架构。PSoC 6 MCU 器件是一款可编程嵌入式片上系统, 集成了以下功能到单个芯片:

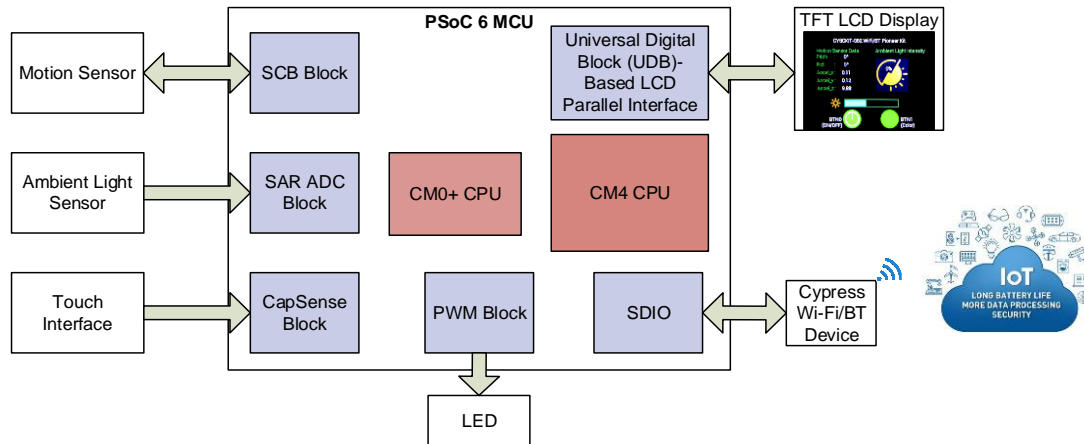
- 单 CPU 微控制器: Arm Cortex-M4 (CM4) 或双 CPU 微控制器: Arm Cortex-M4 (CM4) 和 Cortex-M0+ (CM0+)。
- 可编程模拟和数字外设
- 最多 2 MB 闪存和 1 MB SRAM。
- 第四代 CapSense®技术。
- PSoC 6 MCU 适用于各种功耗敏感的应用, 例如:
 - 智能家居传感器和控制器。
 - 智能家电。
 - 游戏控制器
 - 运动, 智能手机和虚拟现实 (VR) 配件

- 工业传感器节点
- 工业逻辑控制器
- 高级遥控器

可编程模拟和数字子系统使用 **ModusToolbox™ IDE** (用于开发 PSoC 6 MCU 应用程序的基于 Eclipse 的 IDE)，实现设计的灵活性和动态微调。

Figure 1 显示了使用 PSoC 6 MCU 的实际用例的应用级框图

Figure 1. 使用 PSoC 6 MCU 的应用级框图



PSoC 6 MCU 是一种功能强大且灵活的解决方案。例如，Figure 1 中的实际用例利用了以下功能：

- 降压转换器（未在上图中显示），用于超低功耗操作
- 设备内的模拟前端 (AFE)，用于调节和测量传感器输出，如环境光传感器
- 串行通信模块 (SCB)，用于连接多个数字传感器，如运动传感器
- CapSense 技术可实现可靠的触摸和接近感应
- 可编程数字逻辑 (通用数字模块或 UDB) 和外设 (定时器计数器 PWM 或 TCPWM) 分别驱动显示器和 LED
- 与 Cypress Wi-Fi/BT 设备的 SDIO 接口可提供 IoT 云连接
- 由 CM0+ CPU 管理的设备安全功能和 CM4 CPU 执行的应用程序功能

有关更多详细信息，请参见 [器件特性](#) 和 [器件数据表](#)。

本应用笔记向您介绍了 PSoC 6 MCU 的功能，概述了开发生态系统，并让您开始学习使用 PSoC 6 MCU 的简单设计。

我们将向您展示如何从一个空白的初始程序创建应用程序，但是也提供完整的设计作为 [GitHub 上 ModusToolbox 的代码示例](#)。

有关硬件设计的注意事项，请参见 [AN218241 – PSoC 6 MCU Hardware Design Considerations](#) (AN218241 – PSoC 6 MCU 硬件设计注意事项)。

1.1 前提条件

在开始之前，请确保您已安装开发套件并已安装所需的软件。在项目创建期间，您还需要接入互联网以访问 Cypress GitHub 存储库。

1.1.1 硬件

- [PSoC 6 Wi-Fi BT 原型套件 \(CY8CPROTO-062-4343W\)](#)

1.1.2 软件

- [ModusToolbox 2.0](#)

安装软件后，请参阅 ModusToolbox IDE 中的“快速入门指南”和“用户指南”以获取软件概述。

2 开发生态系统

2.1 PSoC 资源

赛普拉斯在 www.cypress.com 上提供了大量数据，可帮助您选择合适的 PSoC 器件，并快速有效地将其集成到您的设计中。有关 PSoC 6 MCU 资源的完整列表，请参见赛普拉斯社区的 [How to Design with PSoC 6 MCU - KBA223067](#)。以下是 PSoC 6 MCU 的简要资源列表。

- **概述:** [PSoC Portfolio](#), [PSoC Roadmap](#)
- **产品选择器:** [PSoC 6 MCU](#)
- **数据手册** 描述并提供每个器件系列的电气规范。
- **应用笔记和代码示例** 涵盖了从基础到高级的广泛主题。您还可以浏览我们的代码示例集合。请参阅 [代码示例](#)。
- **技术参考手册 (TRMs)** 提供了每个器件系列的架构和寄存器的详细说明
- **PSoC 6 MCU 编程规范** 提供了对 PSoC 6 MCU 器件的非易失性存储器进行编程所需的信息。
- **CapSense 设计指南:** 了解如何使用 PSoC 器件设计电容式触摸传感应用。
- **开发工具**
 - **PSoC 6 Wi-Fi-BT Pioneer 套件 (CY8CKIT-062-WiFi-BT)** 套件是一款开发套件，支持带 Wi-Fi 和 BT 连接的 PSoC 62 系列 MCU。
 - **PSoC 6 BLE Pioneer 套件 (CY8CKIT-062-BLE)** 是一款易于使用且价格低廉的开发平台，适用于具有 BLE 连接功能的 PSoC 63 系列 MCU。
 - **PSoC 6 Wi-Fi BT 原型套件 (CY8CPROTO-062-4343W)** 是一款易于使用的套件，支持基于 CYW4343W 模块 Wi-Fi 和 BT 连接的 PSoC 62 系列 MCU。
 - **PSoC 6 BLE 原型套件 (CY8CPROTO-063-BLE)** 是一款易于使用的套件，支持带 BLE 连接的 PSoC 63 MCU。
- **培训视频:** 赛普拉斯提供有关我们产品和工具的 [视频培训](#)，包括 [PSoC 6 MCU](#) 专用系列。

2.2 固件/应用开发

赛普拉斯提供两个开发平台，您可以使用它们进行 PSoC 6 MCU 应用开发：

- **ModusToolbox:** ModusToolbox 软件包括配置工具，底层驱动程序，中间件库和操作系统支持，以及使您能够创建 MCU 和无线应用程序的其他软件包。它还包括可选的 ModusToolbox IDE。
ModusToolbox IDE 是基于 Eclipse 的开发环境，可在 Windows、macOS 和 Linux 平台上运行，并包含各种工具。ModusToolbox 支持完全集成到 IDE 中的独立器件和中间件配置器。使用配置器设置器件中不同模块的配置，并生成可用于固件开发的代码。ModusToolbox 支持所有 PSoC 6 MCU 器件。建议您将 ModusToolbox 用于 PSoC 6 MCU 的所有应用程序开发。有关更多信息，请参见 [ModusToolbox 软件概述](#)。
赛普拉斯在 [Cypress GitHub](#) 网站上提供了库和实施软件。所有开发人员都会使用某些资源。其他的资源将由特定生态系统中的开发人员使用。
GitHub 上可用的赛普拉斯软件资源支持一个或多个目标生态系统：
 - MCU 和蓝牙 SOC 生态系统 - 用于 PSoC 6、Wi-Fi、蓝牙和蓝牙低功耗应用开发的全功能平台
 - Mbed OS 生态系统 - 提供嵌入式操作系统，传输安全性和云服务以创建连接的嵌入式解决方案
 - Amazon FreeRTOS 生态系统 - 通过软件库扩展了 FreeRTOS 内核，可轻松将小型低功耗设备安全地连接到 AWS 云服务ModusToolbox 工具和资源也可以在命令行中使用。有关详细文档，请参阅 [从命令行运行 ModusToolbox](#)。
- **PSoC Creator:** 赛普拉斯专有的 IDE，仅在 Windows 上运行。它支持 PSoC 6 MCU 器件的子集以及 PSoC 3，PSoC 4 和 PSoC 5LP 等其他 PSoC 器件系列。有关更多信息，请参见 [AN221774 Getting Started with PSoC 6 on PSoC Creator](#)。

2.2.1 选择 IDE

最新一代的工具集 ModusToolbox 包含 ModusToolbox IDE。IDE 是基于 Eclipse 的，因此支持 Windows, Linux 和 MacOS 平台。该工具支持所有 PSoC 6 MCU 器件。相关的硬件和中间件配置器也可以在这三个主机操作系统上工作。

ModusToolbox IDE 当前不支持 PSoC 6 MCU 的某些功能，例如 UDB 和 USB 主机。赛普拉斯将来会发布新版本的 ModusToolbox，以支持这些功能并改善用户体验。

如果您具有基于 Eclipse 的工具的使用经验，并且想利用基于 Eclipse 的 IDE 的强大功能和可扩展性，或者希望在 Linux 或 MacOS 上使用开发环境，请选择 ModusToolbox。如果要使用赛普拉斯 IoT 设备构建 IoT 应用程序，或者正在使用 PSoC Creator 不支持的 PSoC 6 MCU 设备，则也应该选择 ModusToolbox。

PSoC Creator 是使用已久的赛普拉斯专有工具，仅在 Windows 上运行。这个成熟的 IDE 包含一个图形编辑器，该图形编辑器在组件的帮助下支持基于原理图的设计输入。PSoC Creator 支持所有 PSoC 3, PSoC 4, PSoC 5LP 器件以及一部分 PSoC 6 MCU 器件。PSoC 6 MCU 设备的子集包括高达 1 MB 的闪存的设备。

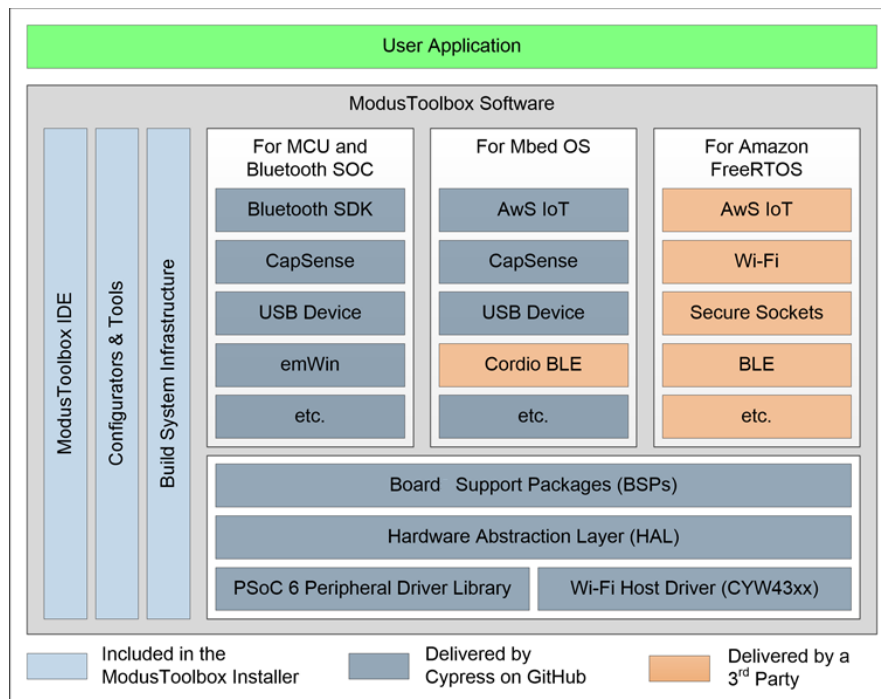
如果您倾向于使用图形编辑器进行设计输入和代码生成，并且 IDE 支持您计划使用的 PSoC MCU，或者如果您打算在 PSoC MCU 上使用 UDB，请选择 PSoC Creator。

2.2.2 ModusToolbox 软件

ModusToolbox 是一套工具和软件，可为创建融合 MCU 和无线系统的客户提供身临其境的开发体验，并使您能够将 Cypress 器件集成到现有的开发方法中。为了实现此目标，ModusToolbox 利用了流行的第三方生态系统，例如 Amazon FreeRTOS, Arm Mbed，并为 Wi-Fi、蓝牙、CapSense 和安全添加了赛普拉斯特定的功能。

其中一种工具是称为 ModusToolbox IDE 的基于 Eclipse 的多平台集成开发环境（IDE），它支持应用程序配置和开发。[Figure 2](#) 显示了 ModusToolbox 软件中包含的工具/资源的高级视图。有关 ModusToolbox 软件的更深入概述，请参见 [ModusToolbox 软件概述](#)。

Figure 2. ModusToolbox 软件



ModusToolbox 安装程序包括 ModusToolbox IDE，设计配置器和工具以及构建系统基础结构。

构建系统基础结构包括新的项目创建向导，该向导可以独立于 ModusToolbox IDE，make 基础结构和其他工具运行。

ModusToolbox 为使用 PSoC 6 MCU 和其他赛普拉斯 Wi-Fi/BT 器件的固件开发提供了三个参考流程。

1. **Amazon FreeRTOS (a:FreeRTOS)** 开发流程- a:FreeRTOS 是微控制器的开源操作系统，使小型，低功耗的边界设备易于编程、部署、保护、连接和管理。Amazon FreeRTOS 扩展了 FreeRTOS 内核，该内核是微控制器的流行开源操作系统，其软件库可轻松将小型低功耗设备安全地连接到 [AWS IoT Core](#) 等 AWS 云服务或运行 [AWS IoT Greengrass](#) 的更强大的边界设备。

要开始使用 Amazon FreeRTOS 在 PSoC 6 上进行固件开发，请访问 [Amazon FreeRTOS 入门](#)。

2. **Arm Mbed OS** 开发流程-Arm Mbed OS 是一款免费的开源嵌入式操作系统，专门为物联网中的“物”设计。它包含开发基于 Arm Cortex-M 微控制器的连接产品所需的所有功能，包括安全性、连接性、RTOS 以及用于传感器和 I/O 设备的驱动程序。

要开始使用 Mbed OS 在 PSoC 6 上进行固件开发，请参阅 [Getting Started with PSoC 6 MCU and CYW43xxx in Mbed OS](#)。

3. **MCU 和蓝牙 SOC 生态系统** - 这是 ModusToolbox IDE 支持 PSoC 6 MCU 的本地开发流程。您可以在裸机上使用它，也可以在您选择的 RTOS 上使用它。

4. 以上所有开发流程均取决于赛普拉斯提供的底层资源。这些包括，

- **板级支持软件包 (BSP)** -BSP 是固件层，其中包含板级特定的驱动程序和其他功能。板级支持软件包是一组库，这些库提供固件 API 来初始化板级并提供对板级外围设备的访问。它包含任何低级资源（例如 PSoC 6 MCU 的 PDL 库），并具有板外设宏的定义。它使用 HAL 来配置板。可以创建自定义 BSP，以支持最终应用板。有关更多信息，请参考 [ModusToolbox BSP User Guide](#)。

- **硬件抽象层 (HAL)** -赛普拉斯的硬件抽象层 (HAL) 提供了一个高级接口，可在 Cypress MCU 上配置和使用硬件模块。它是一个通用接口，可以在多个产品系列中使用。注重易用性和可移植性意味着 HAL 不会公开所有底层的外设功能。HAL 包装了底层的驱动程序（如 PSoC 6 PDL），并提供了与 MCU 的高级接口。该接口被抽象为可在任何 MCU 上工作。这有助于您独立于目标 MCU 编写应用程序固件

HAL 可以在单个应用程序中与特定平台的库（例如 PSoC 6 PDL）结合使用。您可以将 HAL 的更简单、更通用的接口用于大多数应用程序，即使其中一部分需要更细粒度的控制。

- **PSoC 6 外设驱动程序库 (PDL)** -赛普拉斯 PDL 将设备头文件、启动代码和外设驱动程序集成到一个软件包中。PDL 支持 PSoC 6 器件系列。驱动程序将硬件功能抽象为一组易于使用的 API。这些在《PDL API 参考》中有完整记录。

PDL 减少了解寄存器用法和位结构的需求，从而简化了 PSoC 6 系列中大量外设的软件开发。您为应用程序配置驱动程序，然后使用 API 调用来初始化和使用外设。

- **大量的中间件库**，为应用程序提供特定功能。[可用的中间件](#)涵盖连接性（蓝牙，AWS IoT，BLE）和特定于 PSoC-6 的功能（CapSense，USB，设备固件升级（DFU），emWin）。所有中间件均通过赛普拉斯 GitHub 存储库作为 Library 交付。

2.2.3 PSoC 6 软件资源

PSoC 6 软件包括驱动程序和中间件配置器，可让您开始使用 PSoC 6 MCU 开发固件。它包含配置程序、驱动程序、库、中间件以及各种实用程序、makefile 和脚本。它还包括与赛普拉斯 IoT 设备和连接解决方案配合使用的相关驱动程序、中间件和示例。您可以在自己喜欢的任何环境中使用任何或所有工具。

2.2.3.1 配置器

ModusToolbox 软件提供了称为 **Configurators** 的图形应用程序，使配置硬件模块更加容易。例如，无需搜索所有文档以将串行通信模块配置为具有所需配置的 UART，只需打开适当的 **Configurator** 并设置波特率、奇偶校验和停止位。保存硬件配置后，该工具将生成“C”代码，以使用所需的配置初始化硬件。

配置器彼此独立，但是可以一起使用以提供灵活的配置选项。它们可以单独使用，也可以与其他工具结合使用，也可以在完整的 IDE 中使用。一切都捆绑在一起，作为统一 SDK 的一部分，以进行分发。配置器用于：

- 设置选项并生成代码以配置驱动程序
- 设置外设的连接，例如引脚和时钟
- 设置选项并生成代码以配置中间件

对于 PSoC 6 MCU 应用，可用的配置器包括：

- **设备配置器**: 设置系统（平台）功能以及基本外设（例如 UART，定时器，PWM）。
- **CapSense 配置器和调谐器**: 配置 CapSense 并生成所需的代码。

- USB 配置器: 配置 USB 设置并生成所需的代码。
- QSPI 配置器: 配置外部存储器并生成所需的代码。
- 智能 I/O 配置器: 配置智能 I/O。
- BLE 配置器: 配置蓝牙低功耗 (BLE) 设置。

上面的每个配置器都会创建自己的文件 (例如: CapSense 的 *design.cycapsense*)。配置程序文件 (*design.modus* 或 *design.cycapsense*) 通常随 BSP 一起提供。当基于 BSP 创建应用程序时, 文件将被复制到该应用程序中。您还可以为应用程序创建自定义设备配置程序文件, 并覆盖 BSP 提供的文件。有关更多详细信息, 请参见 ModusToolbox 帮助。

2.2.3.2 PSoC 6 MCU 的软件开发

赛普拉斯提供了重要的源代码和工具来支持 PSoC 6 MCU 的软件开发。您可以使用工具指定如何配置硬件, 为此目的生成代码以在固件中使用, 还可以使用各种中间件库来实现其他功能, 例如 BLE 连接或 FreeRTOS。此源代码使为支持的设备开发固件更加容易。它可以帮助您快速自定义和构建固件, 而无需了解寄存器集。

在 ModusToolbox 环境中, 您可以使用配置器来配置设备或中间件库, 例如 BLE 堆栈或 CapSense 功能。

驱动程序代码作为 *psoc6pdl* 库提供。中间件作为每个功能/特性的单独库提供。

无论您使用 ModusToolbox IDE, 第三方 IDE 还是命令行, 希望在寄存器级别工作的固件开发人员都应从 PDL 中引用驱动程序源代码。PDL 包含项目所需的所有特定于设备的头文件和启动代码。它也可以作为每个驱动程序的参考。由于 PDL 是作为源代码提供的, 因此您可以在寄存器级别看到它如何访问硬件。

某些设备不支持特定的外设。PDL 是任何受支持设备的所有驱动程序的超集。此超集设计意味着:

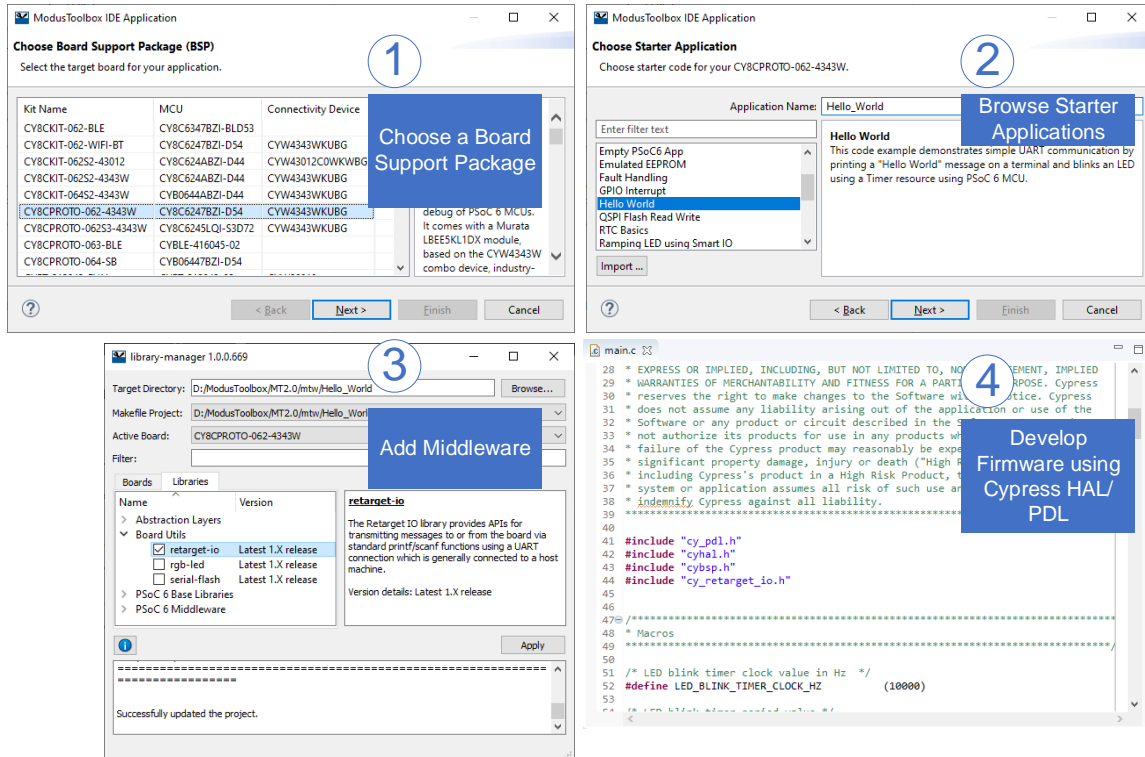
- 初始化、配置和使用外设所需的所有 API 元素均可用。
- 无论可用的外设如何, PDL 在各种 PSoC 6 MCU 器件中都非常有用。
- PDL 包括错误检查, 以确保所选设备上存在目标外设。

只要外设可用, 这就能使代码保持 PSoC 6 器件系列成员之间的兼容性。设备头文件指定了设备可用的外设。如果编写的代码尝试使用不受支持的外设, 则在编译时会出现错误。在编写使用外设的代码之前, 请查阅特定器件的数据表以确认对该外设的支持。

如 Figure 3 所示, 使用 ModusToolbox IDE, 您可以:

1. 选择一个板级支持包 (BSP)。
2. 根据启动应用程序列表创建一个新应用程序, 并按套件进行过滤。
3. 添加中间件。
4. 使用赛普拉斯 HAL 或 PSoC 6 MCU PDL 开发应用程序固件。

Figure 3. ModusToolbox IDE 资源和中间件



2.2.4 ModusToolbox 帮助

ModusToolbox 软件概述提供了 ModusToolbox 软件的高级概述。

访问 [ModusToolbox](#) 主页以下载并安装最新版本的 ModusToolbox。启动 ModusToolbox IDE，并导航至以下各项：

选择 **Help > ModusToolbox IDE Documentation >**:

- **快速入门指南 (Quick Start Guide)**: 本指南为您提供使用 ModusToolbox 的基础知识。
- **用户指南 (User Guide)**: 本指南主要涵盖构建、编程和调试应用程序的 ModusToolbox IDE 方面。它还涵盖了与 IDE 一起安装的工具的各个方面
- **IDE 生存指南**

可在 **Help > ModusToolbox General Documentation** 下的 ModusToolbox 文档登录页面上找到其他文档。

2.3 支持其他 IDE

您可以使用喜欢的 IDE（例如 IAR Embedded Workbench 或 Visual Studio Code）为 PSoC 6 MCU 开发固件。

ModusToolbox 配置器是独立工具，可用于设置和配置 PSoC 6 MCU 资源和其他中间件组件，而无需使用 ModusToolbox IDE。设备配置程序和中间件配置程序在应用程序工作空间内使用 *design.x* 文件。然后，您可以指向生成的源代码，并继续在 IDE 中开发固件。

如果设备配置发生更改，请使用配置器编辑 *design.x* 文件，并重新生成目标 IDE 的代码。赛普拉斯建议您使用 ModusToolbox 软件随附的配置工具来生成资源配置。

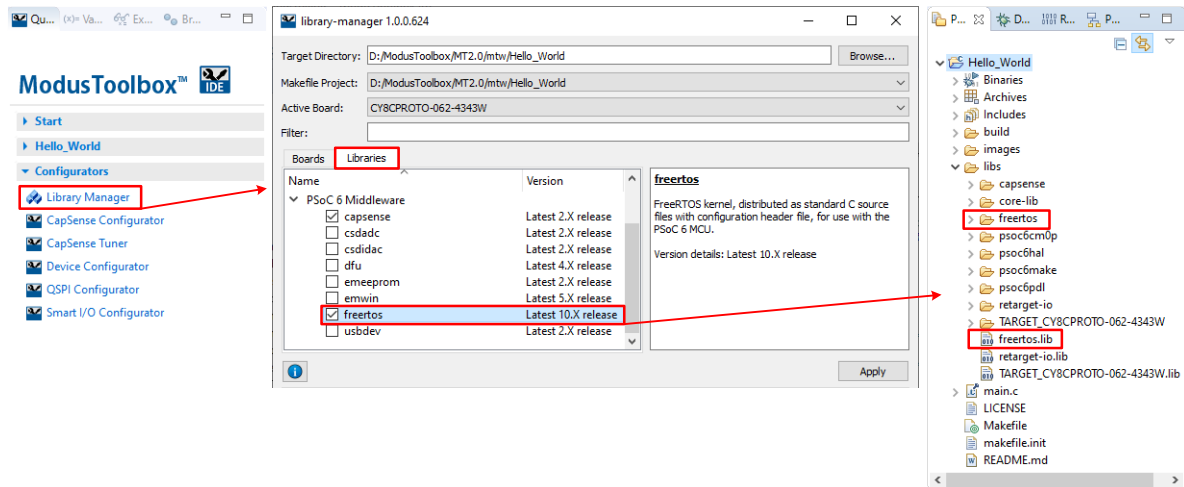
有关详细信息，请参见将 ModusToolbox 软件与第三方 IDE 配合使用 ([AN225588 – Using ModusToolbox Software with a Third-Party IDE](#))。

2.4 FreeRTOS 对 ModusToolbox 的支持

向 ModusToolbox 应用程序项目添加本地 FreeRTOS 支持就像添加任何库/中间件一样。您可以使用库管理器（Library Manager）将 FreeRTOS 中间件导入您的应用程序。选择应用程序项目，然后单击“Quick Panel”中的“Library Manager”链接。从 **Libraries > PSoC 6 Middleware** 对话框中选择 freertos，如 Figure 4 所示。

指向 FreeRTOS 中间件的 .lib 文件已添加到应用程序项目中。中间件内容也被下载并放置在名为 **freertos** 的相应文件夹中。要继续使用 FreeRTOS，请遵循 [FreeRTOS 文档](#) 的“快速入门（Quick Start）”部分中的步骤。

Figure 4. 在 ModusToolbox IDE 应用程序中导入 FreeRTOS 中间件



2.5 编程/调试

PSoC 6 Wi-Fi BT 原型开发套件 (CY8CPROTO-062-4343W) 具有 KitProg3 板载编程器/调试器。它支持 Cortex 微控制器软件接口标准-调试访问端口（CMSIS-DAP）。有关详细信息，请参见 [KitProg3 用户指南](#)。

ModusToolbox IDE 需要 KitProg3，并使用 [OpenOCD](#) 协议来调试 PSoC 6 MCU 应用程序。它还使用 [Segger J-Link](#) 等行业标准探针支持 GDB 调试。

注意： **PSoC 6 Wi-Fi-BT Pioneer Kit (CY8CKIT-062-WiFi-BT)** 和 **PSoC 6 BLE Pioneer Kit (CY8CKIT-062-BLE)** 具有 KitProg2 板载编程器/调试器。要使用 ModusToolbox IDE，请将套件升级到 KitProg3。

ModusToolbox 包含 **fw-loader** 命令行工具，用于更新 CY8CIT-062-WiFi-BT 和 CY8CKIT-062-BLE 套件，并将 KitProg 固件从 KitProg2 切换到 KitProg3。有关更多详细信息，请参考 ModusToolbox IDE 用户指南中的“PSoC 6 MCU KitProg 固件加载程序”部分。

有关使用 ModusToolbox 在 PSoC 器件上调试固件的更多信息，请参考 [ModusToolbox 帮助](#)。

2.6 PSoC 6 MCU 开发套件

PSoC 6 Wi-Fi-BT Pioneer 套件 (CY8CKIT-062-WiFi-BT) 和 **PSoC 6 Wi-Fi BT 原型套件 (CY8CPROTO-062-4343W)** 是开发套件，支持 PSoC 62 系列 MCU 以及 Wi-Fi 和 BT 连接。

PSoC 6 BLE Pioneer 套件 (CY8CKIT-062-BLE) 和 **PSoC 6 BLE 原型套件 (CY8CPROTO-063-BLE)** 支持具有蓝牙低功耗（BLE）连接功能的 PSoC 6 MCU。有关更多信息，请参考 [PSoC 6 产品页面](#)。

3 器件特性

PSoC 6 MCU 器件具有广泛的功能集，如 Figure 5 所示。以下是其主要功能列表。有关更多信息，请参见器件数据手册，技术参考手册 (TRM) 以及相关应用笔记和代码示例。

■ MCU 子系统

- 150-MHz Arm Cortex-M4 和 100-MHz Arm Cortex-M0+
- 高达 2 MB 闪存，另外 32 KB 用于 EEPROM 仿真和 32 KB 监控闪存
- 高达 1 MB 的 SRAM，可选择深度睡眠保持粒度，32 KB 保留边界
- 硬件支持处理器间通信
- DMA 控制器

■ 加密特性

- 密码加速器和真正的随机数生成器功能
- 一次性可编程 eFUSE，用于安全密钥存储
- 通过基于硬件哈希的身份验证进行安全启动

■ I/O 子系统

- 多达 104 个具有可编程驱动模式，驱动强度，压摆率的 GPIO
- 两个带有 Smart I/O 的端口，可以实现布尔运算

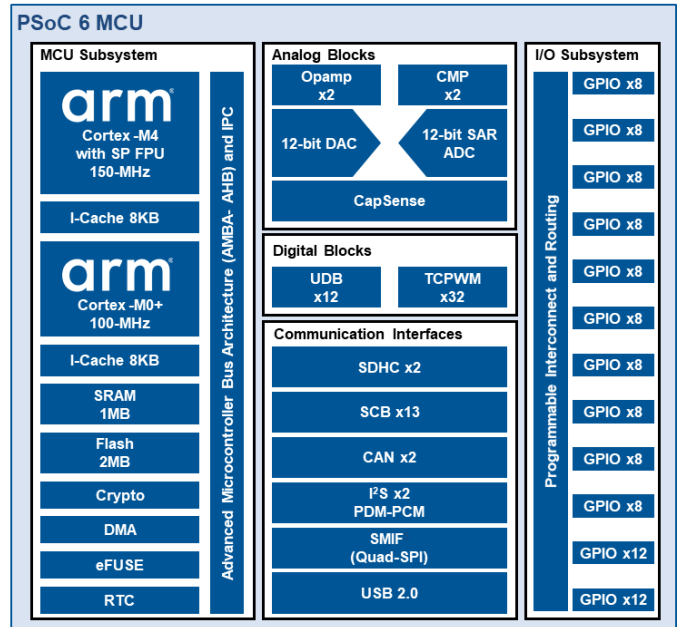
■ 可编程模拟模块

- 两个 6 MHz 增益带宽 (GBW) 的运算放大器和两个低功耗比较器
- 最多一个 12 位，1 Msps SAR ADC 和一个 12 位电压模式 DAC

■ 可编程数字模块，通信接口

- 最多 12 个 UDB 用于定制数字外设
- 最多 32 个 TCPWM 模块可配置为 16 位/32 位定时器，计数器，PWM 或正交解码器
- 最多 13 个 SCB 可配置为 I2C 主设备或从设备，SPI 主设备或从设备或 UART
- 控制器区域网络接口，具有灵活的数据速率
- 最多两个安全数字主机控制器，支持 SD，SDIO 和 eMMC 接口
- 音频子系统，最多具有两个 I2S 接口和两个 PDM 通道
- SMIF 接口，支持从外部四路 SPI 闪存就地执行以及动态加密和解密
- 具有设备和主机功能的全速双角色 USB

Figure 5. PSoC 6 MCU 框图



■ 带有 SmartSense™ 自动调谐功能的 CapSense

- 支持 CapSense Sigma-Delta (CSD) 和 CapSense 发送/接收 (CSX) 控制器
- 提供一流的 SNR，液体容差和接近感应

■ 工作电压范围，电源域和低功耗模式

- 器件工作电压: 1.71 V 至 3.6 V，用户可选择 1.1 V 或 0.9 V 的内核逻辑操作
- 多个片上稳压器: 低压输出 (用于活动，深度睡眠模式的 LDO)，降压转换器
- 六种电源模式可实现精细的电源管理
- 具有内置 RTC，电源管理集成电路 (PMIC) 控制和有限 SRAM 备份的“始终开启”备用电源域

4 我的第一个使用 ModusToolbox IDE 的 PSoC 6 MCU 设计

本节执行以下操作:

- 演示如何构建基于 PSoC 6 MCU 的简单设计, 并将其编程到开发套件中。
- 使您可以轻松学习 PSoC 6 MCU 设计技术以及如何使用 ModusToolbox IDE。

4.1 使用这些说明

这些说明分为几个部分。每个部分都致力于应用程序开发工作流程的一个阶段。主要部分如下:

- [第 1 部分: 创建新应用:](#)
- [第 2 部分: 查看与修改设计](#)
- [第 3 部分: 写固件](#)
- [第 4 部分: 构建应用程序](#)
- [第 5 部分: 对设备进行编程](#)
- [第 6 部分: 测试您的设计](#)

此设计是针对 [PSoC 6 Wi-Fi BT 原型套件 \(CY8CPROTO-062-4343W\)](#) 开发的。通过在创建应用程序时选择适当的工具包, 可以使用其他受支持的工具包来测试此示例。以下各节中描述的代码已经在以下附加工具包上进行了测试。

- [PSoC 6 Wi-Fi-BT Pioneer 套件 \(CY8CKIT-062-WiFi-BT\)](#)
- [PSoC 6 BLE Pioneer 套件 \(CY8CKIT-062-BLE\)](#)
- [PSoC 6 BLE 原型套件 \(CY8CPROTO-063-BLE\)](#)
- [PSoC 62S2 Wi-Fi BT Pioneer 套件 \(CY8CKIT-062S2-43012\)](#)

4.2 关于设计

该设计使用 PSoC 6 MCU 的 CM4 CPU 执行两项任务: UART 通信和 LED 控制。

器件复位后, 赛普拉斯提供的预置 CM0+ 应用程序映像将启用 CM4 CPU 并配置 CM0+ CPU 进入睡眠状态。CM4 CPU 使用 UART 在串行端口流上打印 “Hello World” 消息, 并让套件上的用户 LED 闪烁。当用户在串行控制台上按 Enter 键时, 闪烁将暂停或恢复。

4.3 第 1 部分:创建新应用:

本节将逐步指导您创建新的应用。它使用 **“Empty PSoC6 App”** 启动应用程序，并指导您完成设计开发阶段和编程。

如果您熟悉使用 ModusToolbox 开发项目，则可以直接使用 **“Hello World”** 启动程序。它是一个完整的设计，所有固件都为 CY8CPROTO-062-4343W 套件编写。您可以遍历说明，并观察如何在代码示例中实现这些步骤。

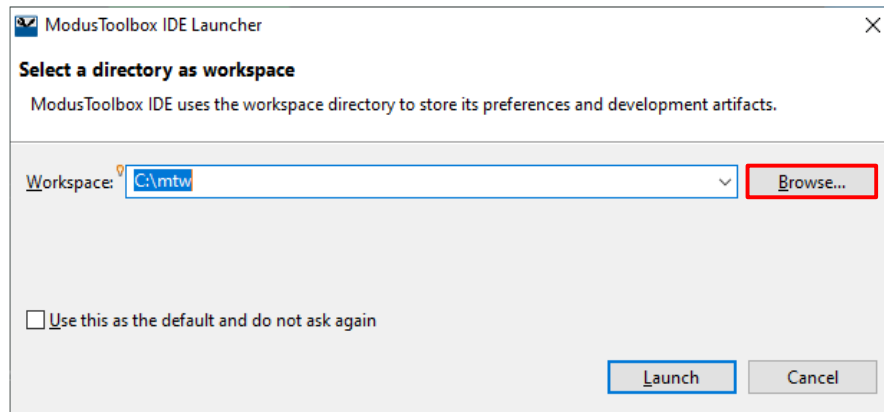
如果您是从头开始并遵循本应用笔记中的所有说明，则在遵循说明的同时，请使用代码示例作为参考。

启动 ModusToolbox 以开始使用。请注意，ModusToolbox 将需要接入互联网才能成功将启动程序应用程序克隆到您的计算机上。

1. 选择一个新的工作区

在启动时，ModusToolbox 会显示一个对话框，以选择一个目录用作工作空间目录。工作空间目录用于存储工作空间首选项和开发工件。您可以通过单击 **Browse** 按钮选择一个现有的空目录，如 [Figure 6](#) 所示。或者，您可以输入目录名称以及完整路径作为工作空间目录，然后 ModusToolbox 将为您创建目录。

Figure 6. 选择一个目录作为 Workspace

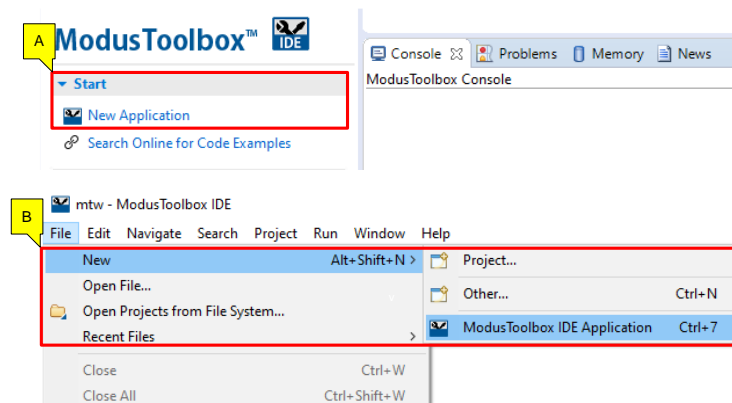


2. 创建一个新的 ModusToolbox 应用程序

- 在“快速面板”（**Quick Panel**）的“开始”（**Start**）组中单击“新建应用程序”（**New Application**）。
- 或者，您可以选择 **File > New > ModusToolbox IDE Application**，如 [Figure 7](#) 所示。

出现 ModusToolbox IDE 应用程序窗口。

Figure 7. 创建一个新的 ModusToolbox IDE 应用

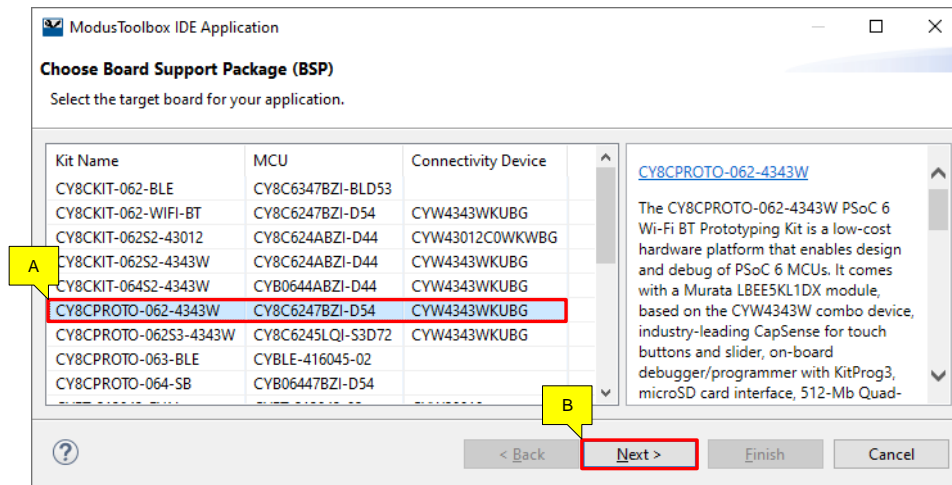


3. 选择一个目标 PSoC 6 开发套件

ModusToolbox 通过提供 BSP 来加速开发过程，这些 BSP 可以在新的应用程序对话框中为指定的开发套件设置各种工作区/项目选项。

- A. 在 **Choose Board Support Package (BSP)** 对话框中，选择您拥有的套件名称。遵循的步骤假定为 **CY8CPROTO-062-4343W**。有关此步骤的帮助，请参见 [Figure 8](#)。
- B. 单击 **Next**。

Figure 8. 选择目标硬件



- C. 在 **Starter Application** 对话框中，选择 **Empty PSoC6 App** 启动应用程序。
- D. 在 **Name** 字段中，输入应用程序的名称，例如 **Hello_World**。如果愿意，可以选择保留默认名称。
- E. 单击 **Next**。出现应用程序摘要对话框。

Figure 9. 选择启动应用程序

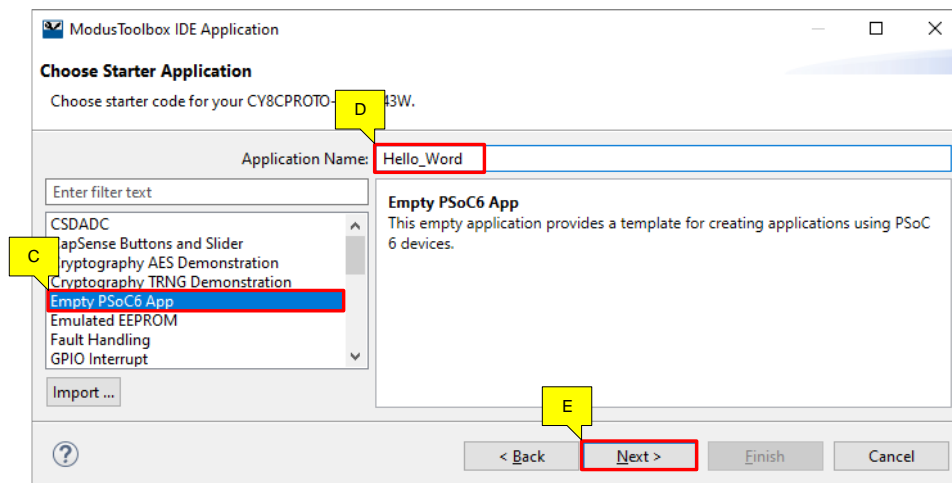
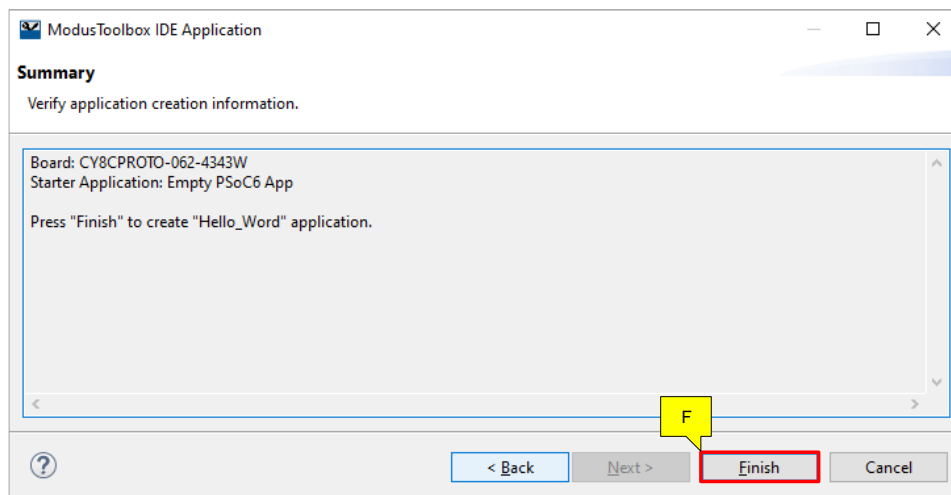


Figure 10. 完成新应用创建



F. 单击 **Finish** 以让 ModusToolbox 为您创建应用程序项目。

您已经为 PSoC 6 MCU 成功创建了一个新的 ModusToolbox 应用程序。

BSP 使用 CY8C624ABZI-D54 作为安装在 [PSoC 6 Wi-Fi-BT 原型套件 \(CY8CPROTO-062-4343W\)](#) 上的默认设备，配置 CYW4343WKUBG Wi-Fi/BT 无线功能。

如果您使用的是基于 PSoC 6 MCU 的自定义硬件，或者使用不同的 PSoC 6 MCU 部件号，请参考 [ModusToolbox BSP 用户指南](#)。该指南也可在 ModusToolbox 安装目录的 `ide_2.0> docs` 文件夹下找到。

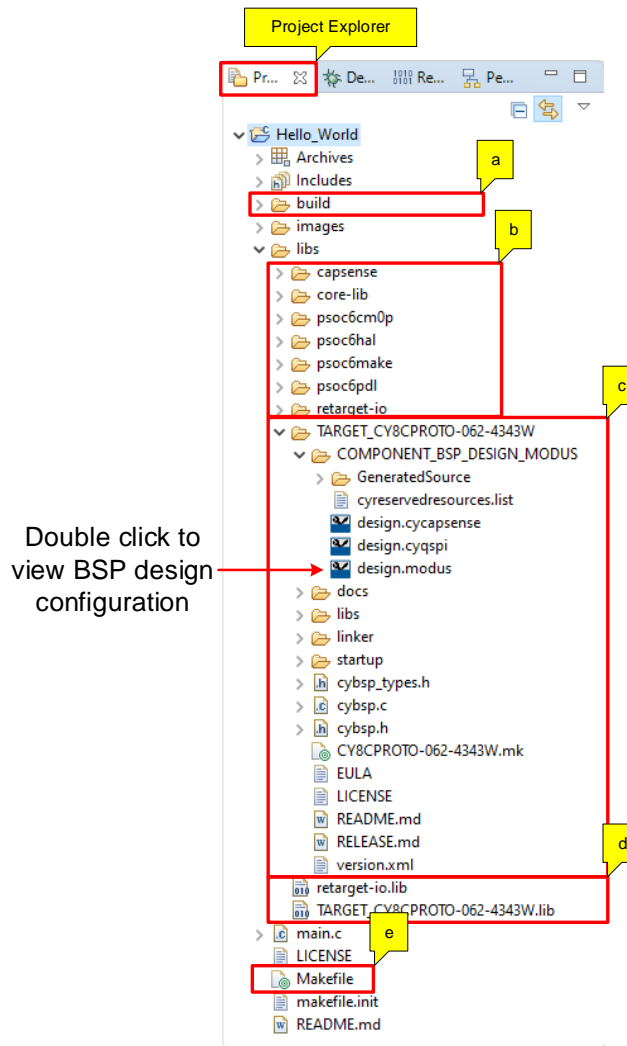
4.4 第 2 部分: 查看与修改设计

Figure 11 显示了 ModusToolbox 项目浏览器界面，显示了应用程序项目的结构。

在 ModusToolbox IDE 中，PSoC 6 MCU 应用程序包含一个项目，用于为 CM4 CPU 开发代码。项目文件夹包含多个子文件夹，每个子文件夹代表项目的特定方面。

- build** 文件夹包含由项目的 **make** 构建产生的所有组件。输出文件由目标 **BSP** 组织。
- libs** 文件夹中的文件夹属于不同的中间件、BSP 和 PSoC 6 PDL。这些是赛普拉斯提供的单个库，它们是根据项目中提供的 **.lib** 文件下载的。

Figure 11. 项目管理器视图



- BSP 提供的文件在 **TARGET_x** 文件夹下列出。设备和外设配置器生成的所有配置文件都包含在 **BSP** 的 **GeneratedSource** 文件夹中，并以 **cycfg_** 作为前缀。这些文件包含 **BSP** 定义的设计配置。您可以通过双击项目中的 **design.x** 文件来查看和修改设计配置。但是，请注意，如果将 **BSP** 库升级到较新的版本，则会丢失对 **design.x** 文件所做的手动编辑。您还可以为应用程序创建自定义设备配置程序文件，并覆盖 **BSP** 提供的文件。有关更多信息，请参见 *ModusToolbox 帮助*。

BSP 文件夹还包含 **.lib** 文件，这些文件指定了项目中使用的中间件和其他库，板上的 **PSoC 6 MCU** 器件的链接程序脚本和启动代码。

- d. `lib` 文件提供了 ModusToolbox 中提取内容的位置。这些文件通常包含整个库的 GitHub 位置。`.lib` 文件可以指向包含另一个 `.lib` 文件的内容。ModusToolbox 递归处理此嵌套的 `.lib` 文件并下载所有库。

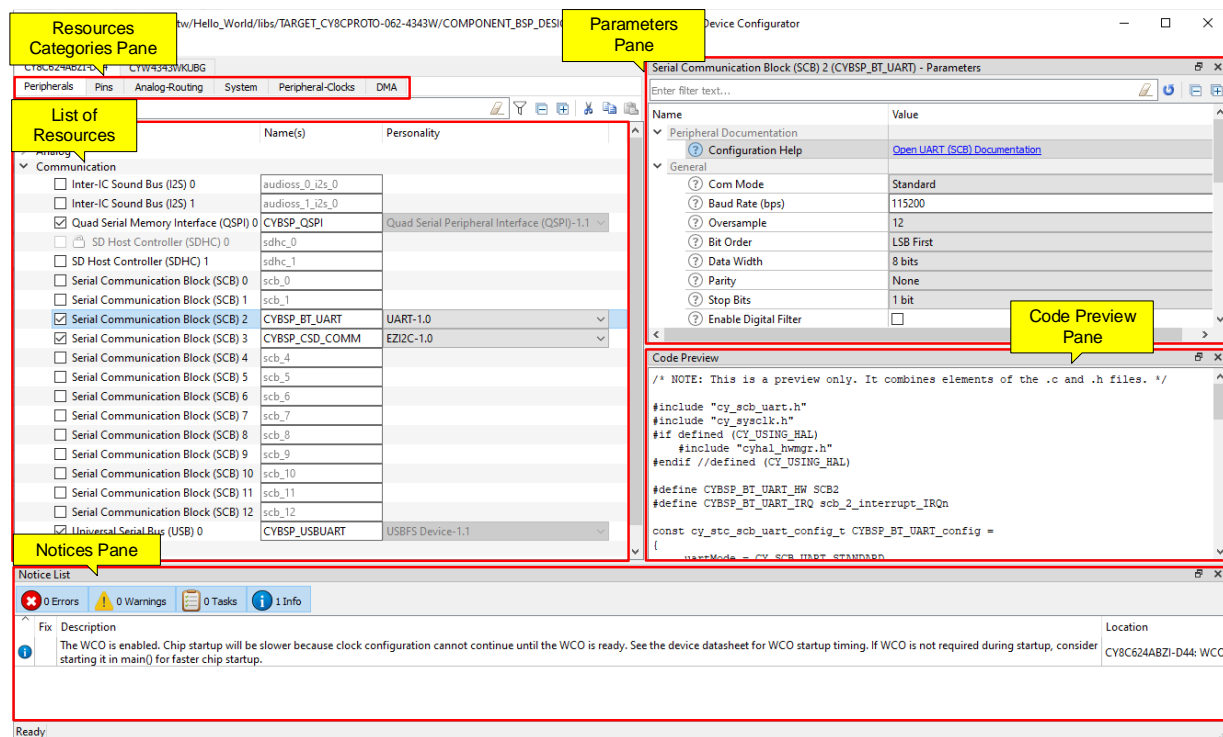
例如，BSP 库文件 `TARGET_CY8CPROTO-062-4343W.lib` 指向

https://github.com/cypresssemiconductorco/TARGET_CY8CPROTO-062-4343W/#latest-v1.X。链接中的 `latest-v1.X` 标记表示 BSP 的特定版本。

- e. 作为第二个示例，`retarget-io.lib` 指向托管在 <https://github.com/cypresssemiconductorco/retarget-io/#latest-v1.X> 上的库。
- f. 它包含有关如何重新创建项目的说明。该文件还包含 `make` 工具用来编译和链接应用程序项目的一组指令。

感兴趣的是 `COMPONENT_BSP_x` 文件夹中的配置文件。在项目资源管理器的配置项目中双击 `design.modus` 文件，或在 **Quick Panel** 中单击 **Device Configurator** 链接。Figure 12 显示了生成的窗口，称为设备配置器窗口。您也可以双击打开其他 `design.x` 文件，以在各自的配置器中打开它们，或单击 **Quick Panel** 中的相应链接。

Figure 12. `design.modus` 概述



Device Configurator 窗口提供了 **Resources Categories** 面板。您可以在此处从资源列表中选择设备中可用的不同资源，例如外设，引脚和时钟。

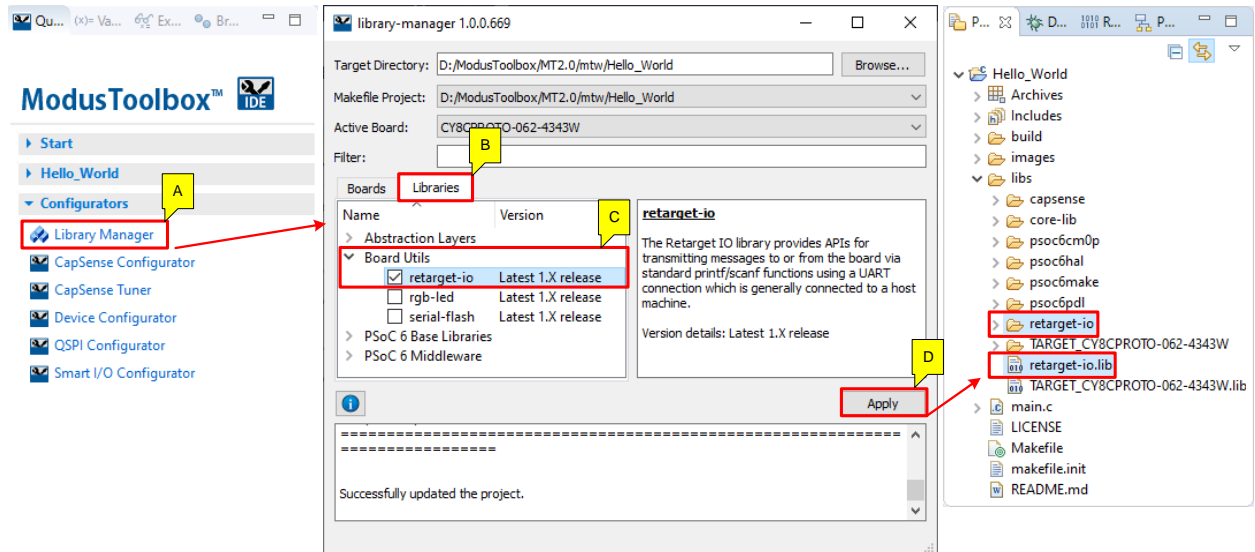
您可以通过选择资源的特性 (**Personality**) 来选择资源的行为。例如，串行通信块 (SCB) 资源可以具有 **EZIC**, **I2C**, **SPI** 或 **UART** 特性。别名 (**Alias**) 是您的资源名称，用于固件开发。可以使用逗号来指定一个或多个别名 (不带空格)。

在 **Parameters** 窗格中，您可以为每个启用的资源和所选特性输入配置参数。**Code Preview** 窗格显示了根据所选配置参数生成的配置代码。此代码填充在 **GeneratedSource** 文件夹的 `cycfg` 文件中。由配置引起的任何错误、警告和信息消息将显示在 **Notices** 窗格中。

该应用程序项目包含相关文件，这些文件可帮助您为 **CM4 CPU** (`main.c`) 创建应用程序，而 **CM0+** 应用程序由 Cypress 作为 C 文件 (针对 **CY8C624ABZI-D44** 器件的 `psoc6_02_cm0p_sleep.c`) 提供。请参见 `psoc6cm0p` 库。作为正常构建过程的一部分，此 C 文件将被编译并与 **CM4** 映像链接。

在开发过程中的这一点上，我们准备将所需的中间件添加到设计中。**Hello World** 应用程序所需的唯一中间件是 `retarget-io` 库。

Figure 13. 添加 retarget-io 中间件



1. 添加 retarget-io 中间件

在此步骤中，您将添加 **retarget-io** 中间件，以将标准输入和输出流重定向到 BSP 配置的 UART。中间件的初始化将在 **main.c** 代码中完成。

- A. 在 **Quick Panel** 中，单击 **Library Manager** 链接。
- B. 在随后的对话框中，选择 **Libraries** 选项卡。
- C. 在 **Board Utils** 下，选择并启用 **retarget-io**。
- D. 单击 **Apply**。

使用 **retarget-io** 中间件所需的文件已添加到 **libs>retarget_io** 文件夹中，并且还添加了 **.lib** 文件，如 **Figure 13** 所示。

2. UART，定时器外设，引脚和系统时钟的配置

调试 UART 外设、定时器外设、引脚和系统时钟的配置可以使用 **BSP** 和 **HAL** 提供的功能 **API** 在代码中直接完成。请参阅第 3 部分：写固件。

4.5 第 3 部分: 写固件

在开发过程的这一点上, 您已经在应用程序模板的帮助下创建了一个应用程序, 并对其进行了修改以添加 **retarget-io** 中间件。在这一部分中, 您将编写实现设计功能的固件。

如果使用 **Empty PSoC 6** 入门应用程序从头开始工作, 则可以从本节提供的代码片段中将相应的源代码复制到应用程序项目的 **main.c** 中。如果您使用的是 **Hello World** 代码示例, 则所有必需文件都已在应用程序中。

固件流程

现在, 我们检查应用程序的 **main.c** 文件中的代码。

CM0+ CPU 退出复位并启用 **CM4** CPU。然后, 由赛普拉斯提供的 **CM0+** 应用程序将 **CM0+** CPU 配置为进入睡眠状态。此示例的资源初始化由 **CM4** CPU 执行。它配置系统时钟、引脚、到外设连接的时钟以及其他平台资源。

启用 **CM4** CPU 后, 时钟和系统资源将通过 **BSP** 初始化功能进行初始化。将 **retarget-io** 中间件配置为使用调试 **UART**, 并初始化用户 **LED**。调试 **UART** 在终端仿真器上打印 “Hello World!” 消息-板载 **KitProg3** 充当 **USB-UART** 桥接器以创建虚拟 **COM** 端口。计时器对象配置为每 1000 毫秒生成一次中断。在每个定时器中断时, **CM4** CPU 都会切换套件上的 **LED** 状态。

固件旨在接受 “Enter” 键作为输入, 并且每次按 “Enter” 键时, 固件都会启动或停止 **LED** 闪烁。

请注意, 应用程序代码使用 **BSP/HAL**/中间件功能来执行预期的功能。

cybsp_init()- 此 **BSP** 函数设置 **HAL** 硬件管理器并初始化设备的所有系统资源, 包括但不限于系统时钟和电源调节器。

cy_retarget_io_init()- 来自 **retarget-io** 中间件的此函数使用为调试 **UART** 引脚设置的别名, 以 115200 的标准波特率配置调试 **UART**, 并将输入/输出流重定向到调试 **UART**。

cyhal_gpio_init()-来自 **gpio HAL** 的此函数初始化物理引脚以驱动 **LED**。所使用的 **LED** 源自 **BSP** 定义。

timer_init()-此函数包装了一组定时器 **HAL** 函数调用, 以实例化和配置定时器。它还为定时器中断设置了回调。

将以下代码片段复制到您的应用程序项目的 **main.c** 中。

```
#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"
#include "cy_retarget_io.h"

/*****
 * Macros
 *****/

/* LED blink timer clock value in Hz */
#define LED_BLINK_TIMER_CLOCK_HZ      (10000)

/* LED blink timer period value */
#define LED_BLINK_TIMER_PERIOD        (9999)

/*****
 * Function Prototypes
 *****/
void timer_init(void);
static void isr_timer(void *callback_arg, cyhal_timer_event_t event);

/*****
 * Global Variables
 *****/
bool timer_interrupt_flag = false;
bool led_blink_active_flag = true;

/* UART HAL object used by Retarget-IO for Debug UART port */
extern cyhal_uart_t cy_retarget_io_uart_obj;

/* Variable for storing character read from terminal */
uint8_t uart_read_value;
```

```

/* Timer object used for blinking the LED */
cyhal_timer_t led_blink_timer;

/*****
 * Function Name: main
 *****/
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /* Initialize retarget-io to use the debug UART port */
    result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX, \
                                CY_RETARGET_IO_BAUDRATE);

    /* retarget-io init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize the User LED */
    result = cyhal_gpio_init((cyhal_gpio_t) CYBSP_USER_LED, \
                             CYHAL_GPIO_DIR_OUTPUT, CYHAL_GPIO_DRIVE_STRONG, \
                             CYBSP_LED_STATE_OFF);

    /* gpio init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[;H");

    printf("***** "
           "PSoC 6 MCU: Hello World! Example "
           "***** \r\n\n");

    printf("Hello World!!!\r\n\n");

    printf("For more PSoC 6 MCU projects, "
           "visit our code examples repositories:\r\n\n");

    printf("1. ModusToolbox Examples:\r\n https://github.com/"
           "cypresssemiconductorco/Code-Examples-for-ModusToolbox"
           "-Software\r\n\n");

    printf("2. Mbed OS Examples:\r\n https://os.mbed.com/teams/Cypress/\r\n\n");

    /* Initialize timer to toggle the LED */
    timer_init();

    printf("Press 'Enter' key to pause or "
           "resume blinking the user LED \r\n\r\n");

    for(;;)
    {

```



```

/* Check if 'Enter' key was pressed */
if(cyhal_uart_getc(&cy_retargt_io_uart_obj, &uart_read_value, 1) \
    == CY_RSLT_SUCCESS)
{
    if (uart_read_value == '\r')
    {
        /* Pause LED blinking by stopping the timer */
        if (led_blink_active_flag)
        {
            cyhal_timer_stop(&led_blink_timer);

            printf("LED blinking paused \r\n");
        }
        else /* Resume LED blinking by starting the timer */
        {
            cyhal_timer_start(&led_blink_timer);

            printf("LED blinking resumed\r\n");
        }

        /* Move cursor to previous line */
        printf("\x1b[1F");

        led_blink_active_flag ^= 1;
    }
}

/* Check if timer elapsed (interrupt fired) and toggle the LED */
if(timer_interrupt_flag)
{
    /* Clear the flag */
    timer_interrupt_flag = false;

    /* Invert the USER LED state */
    cyhal_gpio_toggle((cyhal_gpio_t) CYBSP_USER_LED);
}
}

/*****
 * Function Name: timer_init
 *****/
void timer_init(void)
{
    cy_rslt_t result;

    const cyhal_timer_cfg_t led_blink_timer_cfg = \
    {
        .compare_value = 0,          /* Timer compare value, not used */
        .period = LED_BLINK_TIMER_PERIOD, /* Defines the timer period */
        .direction = CYHAL_TIMER_DIR_UP, /* Timer counts up */
        .is_compare = false,         /* Don't use compare mode */
        .is_continuous = true,       /* Run timer indefinitely */
        .value = 0                   /* Initial value of counter */
    };

    /* Initialize the timer object. Does not use pin output ('pin' is NC) and
     * does not use a pre-configured clock source ('clk' is NULL). */
    result = cyhal_timer_init(&led_blink_timer, (cyhal_gpio_t) NC, NULL);

    /* timer_init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Configure timer period and operation mode such as count direction,
     duration */
    cyhal_timer_configure(&led_blink_timer, &led_blink_timer_cfg);
}

```

```
/* Set the frequency of timer's clock source */
cyhal_timer_set_frequency(&led_blink_timer, LED_BLINK_TIMER_CLOCK_HZ);

/* Assign the ISR to execute on timer interrupt */
cyhal_timer_register_callback(&led_blink_timer, isr_timer, NULL);

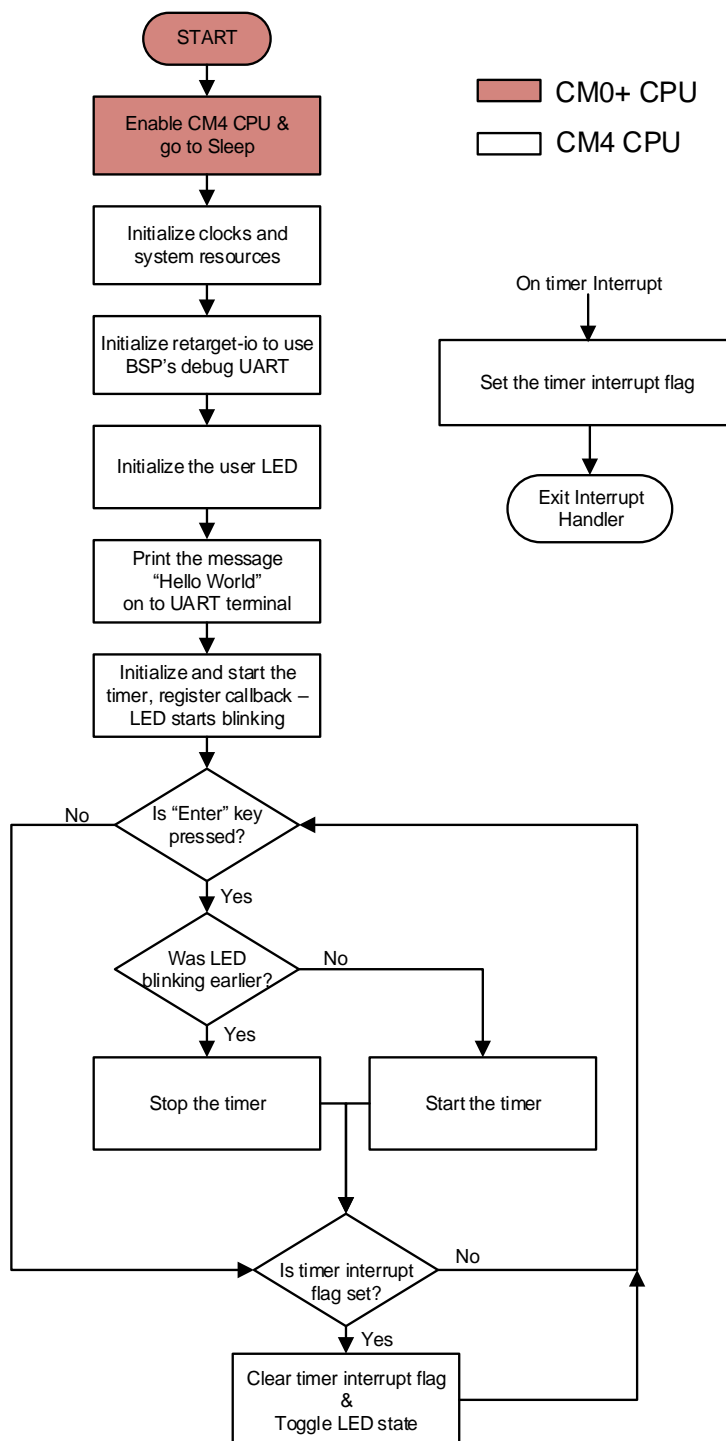
/* Set the event on which timer interrupt occurs and enable it */
cyhal_timer_enable_event(&led_blink_timer, CYHAL_TIMER_IRQ_TERMINAL_COUNT \
                        , 7, true);

/* Start the timer with the configured settings */
cyhal_timer_start(&led_blink_timer);
}

/*****
 * Function Name: isr_timer
 *****/
static void isr_timer(void *callback_arg, cyhal_timer_event_t event)
{
    (void) callback_arg;
    (void) event;

    timer_interrupt_flag = true;
}
```

Figure 14. 固件流程图



这样就完成了代码示例中固件工作方式的总结。随意浏览源文件以加深理解。

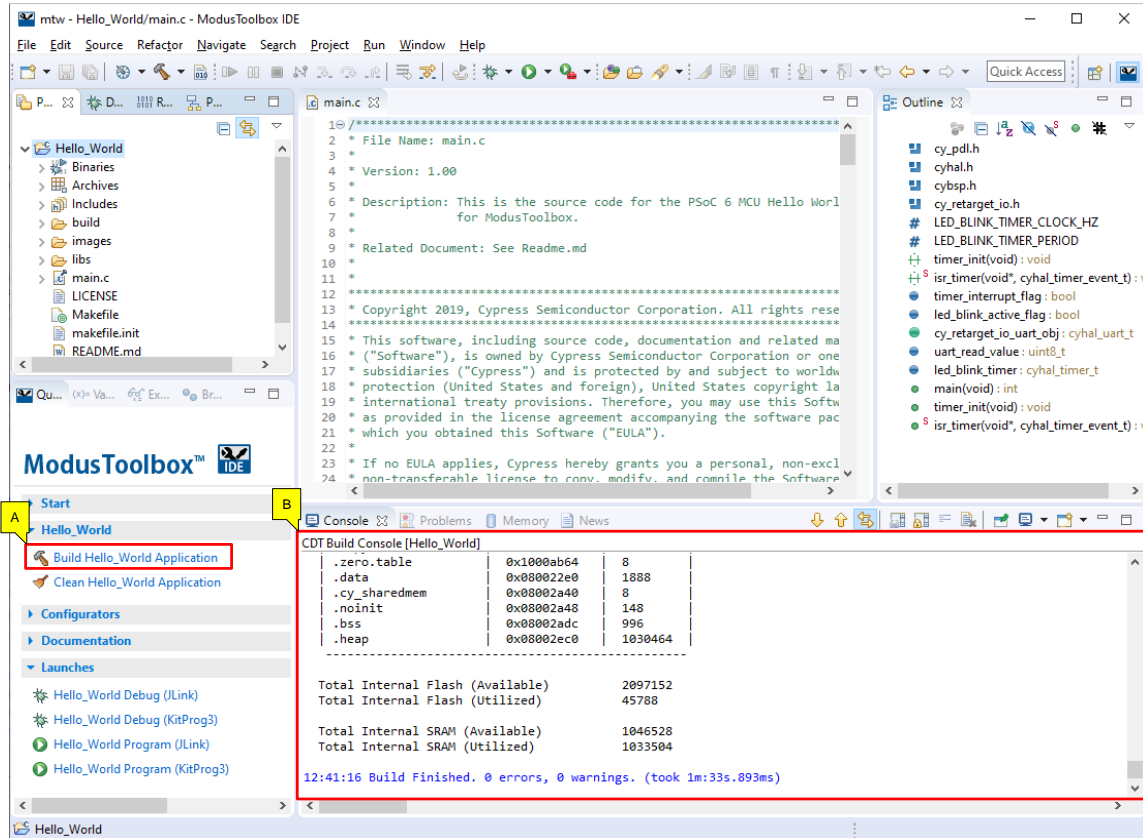
4.6 第 4 部分: 构建应用程序

本节介绍如何构建应用程序。

1. 构建应用程序

- A. 在“项目资源管理器”窗口中选择应用程序项目，然后单击“快速面板”中“<name>”组下的“**Build <name> Application**”快捷方式。它选择“**Debug**”构建配置，并编译/链接构成该应用程序的所有项目。
- B. 控制台视图列出了构建操作的结果，如 [Figure 15](#) 所示

Figure 15. 构建应用程序



如果遇到错误，请重新访问以前的步骤以确保您完成了所有必需的任务。

注意: 您也可以使用命令行界面 (CLI) 来构建应用程序。请参考 [Running ModusToolbox from the Command Line](#) 指南。该文档位于 ModusToolbox 安装路径下的 `/ide_2.0/docs/` 文件夹中。

4.7 第 5 部分: 对设备进行编程

本节介绍如何对 PSoC 6 MCU 器件进行编程。

ModusToolbox 使用 OpenOCD 协议在 PSoC 6 MCU 器件上编程和调试应用程序。要使 ModusToolbox 识别套件上的设备，该套件必须运行 KitProg3。某些 Cypress 套件随机配送 KitProg2 固件而不是 KitProg3。有关详细信息，请参见 [编程/调试](#)。ModusToolbox 包含 fw-loader 命令行工具，用于将 KitProg 固件从 KitProg2 切换到 KitProg3。有关更多详细信息，请参考 ModusToolbox IDE 用户指南中的 **PSoC 6 MCU KitProg Firmware Loader** 部分。

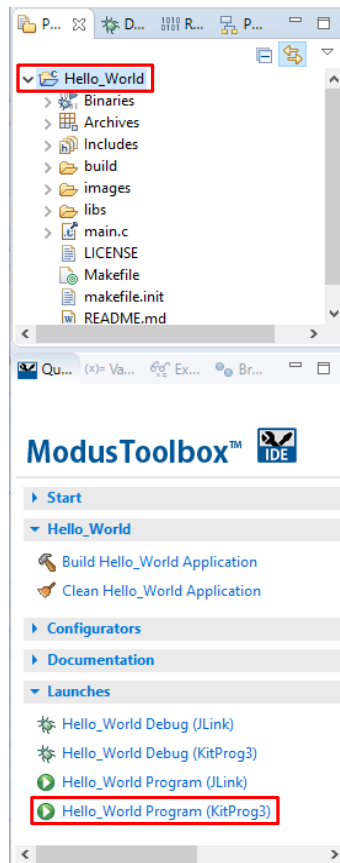
如果您使用带内置编程器的开发套件 (例如 CY8CPROTO-062-4343W)，请使用 USB 电缆将电路板连接到计算机。

如果您使用自己的硬件进行开发，则可能需要硬件编程器/调试器；例如，赛普拉斯 [CY8CKIT-005 MiniProg4](#)。

1. 对应用程序编程

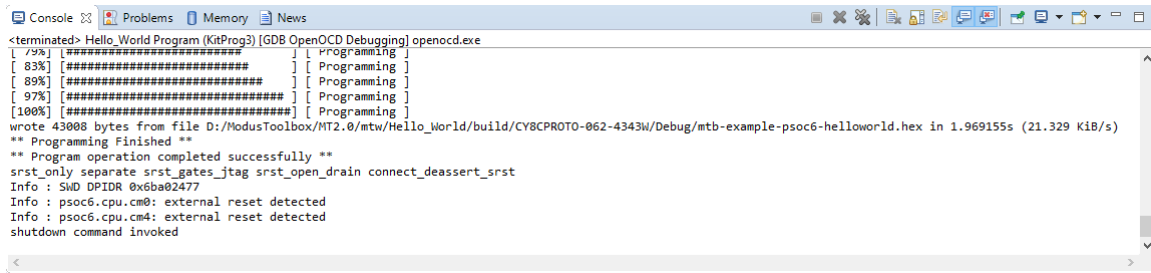
- A. 连接到电路板并执行以下步骤。
- B. 选择应用程序项目，然后单击快速面板中“**Launches**”组下的<application name> Program (KitProg3)快捷方式，如 [Figure 16](#) 所示。IDE 将选择并运行适当的运行配置。请注意，如果自上次构建以来已修改了任何文件，则此步骤还将执行构建。

Figure 16. 将应用程序编程到设备



Console 视图列出了编程操作的结果，如 [Figure 17](#) 所示。

Figure 17. 控制台 - 编程结果



```

<terminated> Hello_World Program (KitProg3) [GDB OpenOCD Debugging] openocd.exe
[ /9%] [*****] [ Programming ]
[ 83%] [*****] [ Programming ]
[ 89%] [*****] [ Programming ]
[ 97%] [*****] [ Programming ]
[100%] [*****] [ Programming ]
wrote 43008 bytes from file D:/ModusToolbox/MT2.0/mtw/Hello_World/build/CY8CPROTO-062-4343W/Debug/mtb-example-psoc6-helloworld.hex in 1.969155s (21.329 KiB/s)
** Programming Finished **
** Program operation completed successfully **
srst_only separate srst_gates_jtag srst_open_drain connect_deassert_srst
Info : SWD DPIDR 0x6ba02477
Info : psoc6.cpu.cm0: external reset detected
Info : psoc6.cpu.cm4: external reset detected
shutdown command invoked
  
```

4.8 第 6 部分: 测试您的设计

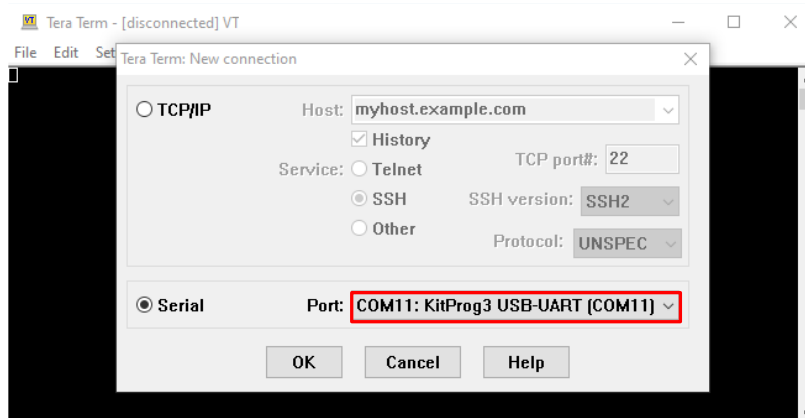
本节介绍如何测试您的设计。

请按照以下步骤观察设计的输出。本应用笔记使用 Tera Term 作为 UART 终端仿真器来查看结果。您可以使用您选择的任何终端查看输出

1. 选择串行端口

启动 Tera Term 并选择 USB-UART COM 端口，如 Figure 18 所示。注意您的 COM 端口号可能与图中不一致。

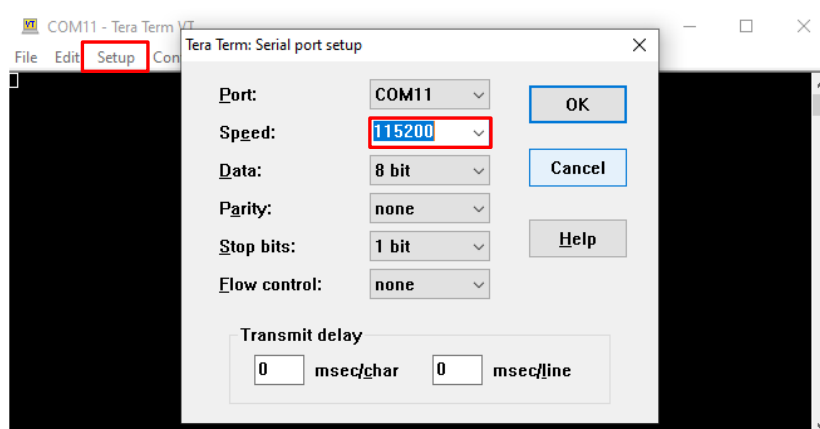
Figure 18. 选择 Tera Term 中的 KitProg3 COM 端口



2. 设置波特率

在 **Setup > Serial port** 路径下，设置波特率为 115200，如 Figure 19 所示。

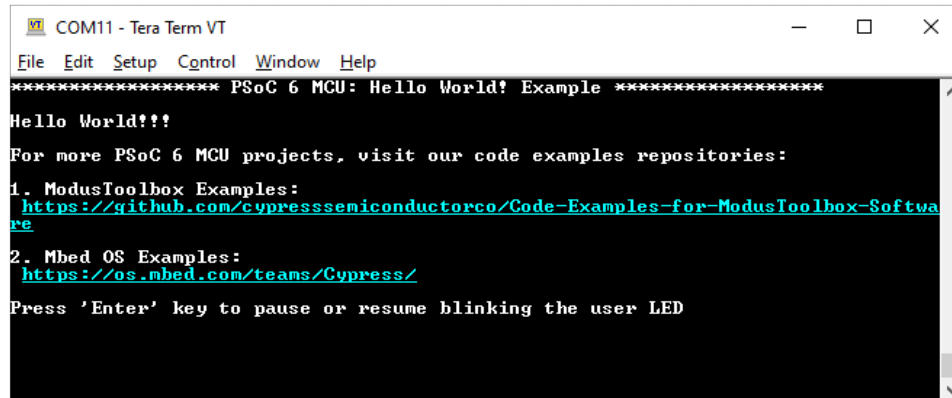
Figure 19. 在 Tera Term 中配置波特率



3. 重置设备

按下套件上的重置开关（SW1）。一条消息出现在终端上，如 Figure 20 所示。套件上的用户 LED 将开始闪烁。

Figure 20. 从 CM4 CPU 打印的 UART 消息



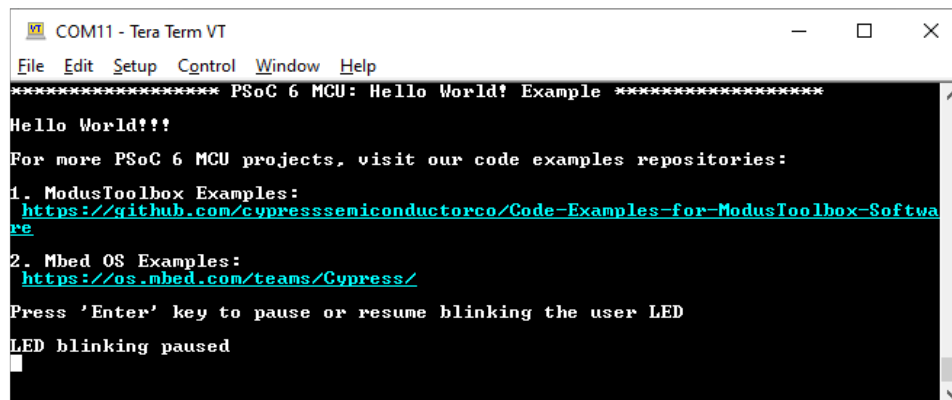
```

COM11 - Tera Term VT
File Edit Setup Control Window Help
***** PSoC 6 MCU: Hello World! Example *****
Hello World!!!
For more PSoC 6 MCU projects, visit our code examples repositories:
1. ModusToolbox Examples:
https://github.com/cypresssemiconductorco/Code-Examples-for-ModusToolbox-Software
2. Mbed OS Examples:
https://os.mbed.com/teams/Cypress/
Press 'Enter' key to pause or resume blinking the user LED
  
```

4. 暂停/恢复 LED 闪烁功能

按 Enter 键暂停/恢复 LED 闪烁。当 LED 闪烁暂停时，相应的消息将显示在终端上，如 Figure 21 所示。

Figure 21. 来自 CM4 CPU 的 UART 信息



```

COM11 - Tera Term VT
File Edit Setup Control Window Help
***** PSoC 6 MCU: Hello World! Example *****
Hello World!!!
For more PSoC 6 MCU projects, visit our code examples repositories:
1. ModusToolbox Examples:
https://github.com/cypresssemiconductorco/Code-Examples-for-ModusToolbox-Software
2. Mbed OS Examples:
https://os.mbed.com/teams/Cypress/
Press 'Enter' key to pause or resume blinking the user LED
LED blinking paused
  
```

5 总结

本应用笔记探讨了 PSoC 6 MCU 器件架构和相关的开发工具。PSoC 6 MCU 是一款真正可编程的嵌入式片上系统，在单芯片上具有可配置的模拟和数字外设功能，存储器和双 CPU 系统。整合的功能和低功耗模式使 PSoC 6 MCU 成为智能家居、物联网网关和其它相关应用的理想选择。

6 相关应用笔记和代码示例

有关 PSoC 6 MCU 代码示例的完整和更新列表，请访问我们的 [Cypress GitHub](#)。有关 PSoC 6 相关文档的更多信息，请访问我们的 [PSoC 6 MCU](#) 产品网页。

Table 1 列出了系统级和一般应用笔记，建议用于学习 PSoC 6 MCU 和 ModusToolbox 的后续步骤。

Table 1. 通用和系统级应用笔记

文档	文档名称
AN221774	PSoC Creator 上 PSoC 6 MCU 入门
AN210781	具有蓝牙低功耗 (BLE) 连接的 PSoC 6 MCU 入门
AN218241	PSoC 6 MCU 硬件设计注意事项
AN225588	第三方 IDE 使用 ModusToolbox 软件
AN219528	PSoC 6 MCU 低功耗模式和降低功耗技术

Table 2 列出了特定外设和应用的应用笔记 (AN)。

Table 2. 与 PSoC 6 MCU 功能相关的文档

文档	文档名称
系统资源, CPU, 和中断	
AN215656	PSoC 6 MCU 双 CPU 系统设计
AN217666	PSoC 6 MCU 中断
CapSense	
AN92239	CapSense 的接近感应
AN85951	PSoC 4 和 PSoC 6 MCU CapSense 设计指南
设备器件升级	
AN213924	PSoC 6 MCU 设备固件升级软件开发套件指南

Appendix A. 术语表

本节列出了使用赛普拉斯 PSoC 系列器件时可能遇到的最常用术语。

板级支持软件包 (BSP) : BSP 是固件层，其中包含板级特定的驱动程序和其他功能。板级支持软件包是一组库，这些库提供固件 API 来初始化电路板并提供对板级外设的访问。

赛普拉斯编程器: 赛普拉斯编程器是一种灵活的跨平台应用程序，用于对赛普拉斯器件进行编程。它可以编程、擦除、验证和读取目标设备的闪存。

硬件抽象层 (HAL) : HAL 包含了较低层的驱动程序（例如 PSoC 6 PDL），并提供了与 MCU 的高层接口。该接口被抽象为可在任何 MCU 上工作。

KitProg: KitProg 是具有 USB-I2C 和 USB-UART 桥接功能的板载编程器/调试器。KitProg 已集成到大多数 PSoC 开发套件中。

MiniProg3/MiniProg4: 用于开发的编程硬件，用于对自定义板上的 PSoC 器件或不支持内置编程器的 PSoC 开发套件进行编程。

特性 (Personality) : 个性表示功能的资源可配置性。例如，SCB 资源可以配置为 UART，SPI 或 I2C 特性。

PSoC: PSoC 是一个包含 CPU (如 32 位 Arm Cortex-M0)、可编程的模拟和数字模块的可编程嵌入式设计平台。通过它，可以使用稳定且易于使用的解决方案 (如触摸感应) 来加快嵌入式系统的设计，并能够创建低功耗设计。

中间件: 中间件是一组固件模块，可为应用程序提供特定功能。某些中间件可以提供网络协议（例如 MQTT），而某些中间件可以提供与设备功能（例如 USB，音频）的高级软件接口。

ModusToolbox: 面向 IoT 设计人员的基于 Eclipse 的嵌入式设计平台，结合了业界部署最广泛的 Wi-Fi 和蓝牙技术以及功耗最低，最灵活的 MCU 和一流的传感技术，可提供单一、连贯且熟悉的设计体验。

外设驱动程序库: 外设驱动程序库 (PDL) 简化了 PSoC 6 MCU 架构的软件开发。PDL 减少了了解寄存器用法和位结构的需求，从而简化了针对大量可用外设的软件开发。

WICED: 赛普拉斯的 WICED (嵌入式设备的无线互联网连接) 是一个功能齐全的平台，拥有成熟的软件开发套件 (SDK) 和来自合作伙伴的交钥匙硬件解决方案，可以在系统设计中轻松实现 Wi-Fi 和蓝牙连接。

修订记录

文档标题: AN228571 – ModusToolbox 上 PSoC 6 MCU 入门

文档编号: 002-29260

版本	ECN	提交日期	变更说明
**	6754232	12/17/2019	翻译自 002-28571 Rev. **

销售、解决方案以及法律信息

全球销售和设计支持

赛普拉斯公司拥有一个由办事处、解决方案中心、原厂代表和经销商组成的全球性网络。如欲查找离您最近的办事处，请访问[赛普拉斯所在地](#)。

产品

Arm® Cortex®微控制器	cypress.com/arm
汽车级产品	cypress.com/automotive
时钟与缓冲器	cypress.com/clocks
接口	cypress.com/interface
物联网	cypress.com/iot
存储器	cypress.com/memory
微控制器	cypress.com/mcu
PSoC	cypress.com/psoc
电源管理 IC	cypress.com/pmic
触摸感应	cypress.com/touch
USB 控制器	cypress.com/usb
无线连接	cypress.com/wireless

PSoC®解决方案

[PSoC 1](#) | [PSoC 3](#) | [PSoC 4](#) | [PSoC 5LP](#) | [PSoC 6 MCU](#)

赛普拉斯开发者社区

[社区](#) | [项目](#) | [视频](#) | [博客](#) | [培训](#) | [组件](#)

技术支持

cypress.com/support



Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© 赛普拉斯半导体公司，2019 年。本文件是赛普拉斯半导体公司及其子公司，包括 Spansion LLC（“赛普拉斯”）的财产。本文件，包括其包含或引用的任何软件或固件（“软件”），根据全球范围内的知识产权法律以及美国与其他国家签署条约由赛普拉斯所有。除非在本款中另有明确规定，赛普拉斯保留在该等法律和条约下的所有权利，且未就其专利、版权、商标或其他知识产权授予任何许可。如果软件并不附随有一份许可协议且贵方未以其他方式与赛普拉斯签署关于使用软件的书面协议，赛普拉斯特此授予贵方属人性质的、非独家且不可转让的如下许可（无再许可权）（1）在赛普拉斯特软件著作权项下的下列许可权（一）对以源代码形式提供的软件，仅出于在赛普拉斯硬件产品上使用之目的且仅在贵方集团内部修改和复制软件，和（二）仅限于在有关赛普拉斯硬件产品上使用之目的将软件以二进制代码形式的向外部最终用户提供（无论直接提供或通过经销商和分销商间接提供），和（2）在被软件（由赛普拉斯公司提供，且未经修改）侵犯的赛普拉斯专利的权利主张项下，仅出于在赛普拉斯硬件产品上使用之目的制造、使用、提供和进口软件的许可。禁止对软件的任何其他使用、复制、修改、翻译或汇编。

在适用法律允许的限度内，赛普拉斯未对本文件或任何软件作出任何明示或暗示的担保，包括但不限于关于适销性和特定用途的默示保证。没有任何电子设备是绝对安全的。因此，尽管赛普拉斯在其硬件和软件产品中采取了必要的安全措施，但是赛普拉斯并不承担任何由于使用赛普拉斯产品而引起的问题及安全漏洞的责任，例如未经授权的访问或使用赛普拉斯产品。此外，本材料中所介绍的赛普拉斯产品有可能存在设计缺陷或设计错误，从而导致产品的性能与公布的规格不一致。（如果发现此类问题，赛普拉斯会提供勘误表）赛普拉斯保留更改本文件的权利，届时将不另行通知。在适用法律允许的限度内，赛普拉斯不对因应用或使用本文件所述任何产品或电路引起的任何后果负责。本文件，包括任何样本设计信息或程序代码信息，仅为供参考之目的提供。文件使用者应负责正确设计、计划和测试信息应用和由此生产的任何产品的功能和安全性。赛普拉斯产品不应被设计为、设定为或授权用作武器操作、武器系统、核设施、生命支持设备或系统、其他医疗设备或系统（包括急救设备和手术植入物）、污染控制或有害物质管理系统中的关键部件，或产品植入之设备或系统故障可能导致人身伤害、死亡或财产损失其他用途（“非预期用途”）。关键部件指，若该部件发生故障，经合理预期会导致设备或系统故障或会影响设备或系统安全性和有效性的部件。针对由赛普拉斯产品非预期用途产生或相关的任何主张、费用、损失和其他责任，赛普拉斯不承担全部或部分责任且贵方不应追究赛普拉斯之责任。贵方应赔偿赛普拉斯因赛普拉斯产品任何非预期用途产生或相关的所有索赔、费用、损失和其他责任，包括因人身伤害或死亡引起的主张，并使之免受损失。

赛普拉斯、赛普拉斯徽标、Spansion、Spansion 徽标，及上述项目的组合，WICED，及 PSoC、CapSense、EZ-USB、F-RAM 和 Traveo 应视为赛普拉斯在美国和其他国家的商标或注册商标。请访问 cypress.com 获取赛普拉斯商标的完整列表。其他名称和品牌可能由其各自所有者主张为该方财产。