# ModusToolbox™上 PSoC™ 6 MCU 入门

## 关于本文档

### 范围和目的

AN228571 介绍 PSoC™ 6 MCU，这是一款采用 Arm® Cortex®-M4 和 Cortex®-M0+处理器的双核可编程片上系统。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™软件创建第一个项目。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。

### 目标读者

本文档面向 PSoC™ 6 MCU 和 ModusToolbox™软件的新用户。

### 相关器件系列

所有 PSoC™ 6 MCU 器件

### 软件版本

ModusToolbox™软件 3.0 或更高版本

### 更多代码示例? 我们倾听到了您的声音。

要查看持续增长的使用 ModusToolbox™ 的 PSoC™代码示例，请访问 GitHub 站点，您也可以在 PSoC™ video library 浏览 PSoC™视频库。

## 内容


关于本文档 . . . 1
内容 . . . 1
1 简介 . . . 3
2 开发生态系统 . . . 5
2.1 PSoC™资源 . . . 5
2.2 固件/应用开发 . . . 5
2.2.1 安装 ModusToolbox™工具包 . . . 6
2.2.2 选择 IDE . . . 6
2.2.3 ModusToolbox™™软件 . . . 6
2.2.4 ModusToolbox™应用 . . . 8
2.2.5 PSoC™ 6 软件资源 . . . 11
2.2.5.1 配置器 . . . 11
2.2.5.2 PSoC™ 6 MCU 库管理 . . . 12
2.2.5.3 PSoC™ 6 MCU 的软件开发 . . . 12
2.2.6 ModusToolbox™帮助 . . . 13
2.3 支持其他 IDE . . . 14
2.4 FreeRTOS 对 ModusToolbox™的支持 . . . 14
2.5 使用 Eclispse IDE 编程/调试 . . . 15
2.6 PSoC™ 6 MCU 开发套件 . . . 16

内容


**3 器件特性 ..... 17**

**4 我的第一个使用 Eclipse IDE for ModusToolbox™的 PSoC™ 6 MCU 设计 ..... 19**

4.1 前提条件 ..... 19

4.1.1 硬件 ..... 19

4.1.2 软件 ..... 19

4.2 使用这些说明 ..... 19

4.3 关于设计 ..... 20

4.4 第 1 部分: 创建新应用: ..... 20

4.5 第 2 部分: 查看与修改设计配置 ..... 23

4.5.1 打开设备配置器 ..... 25

4.5.2 添加 retarget-io 中间件 ..... 26

4.5.3 UART、定时器外设、引脚和系统时钟的配置。 ..... 27

4.6 第 3 部分: 写固件 ..... 27

4.7 第 4 部分: 编译应用程序 ..... 35

4.8 第 5 部分: 对设备进行编程 ..... 35

4.9 第 6 部分: 测试您的设计 ..... 38

**5 总结 ..... 41**

**参考文档 ..... 42**

**术语表 ..... 43**

**文档修订记录 ..... 44**

**免责声明 ..... 45**

# 1 简介

PSoC™ 6 MCU 是超低功耗 PSoC™器件，具有针对智能家居，物联网网关等量身定制的双核架构。PSoC™ 6 MCU 器件是一款可编程嵌入式片上系统，单个芯片集成了以下功能:

- 单核微控制器: Arm® Cortex®-M4 (CM4) 或双核微控制器: 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
  - 可穿戴设备

ModusToolbox™ software environment 支持 PSoC™ 6 MCU 应用开发，提供了一套工具，用于配置器件，设置外设，并通过世界一流的中间件对您的项目进行补充。请参见 Infineon GitHub 库，获取所有套件的板级支持包 (BSP)、CAPSENSE™和 emWin 等流行功能的库，以及一系列全面的示例应用，以便您入门。

图 1 显示了使用 PSoC™ 6 MCU 的实际用例的应用级框图

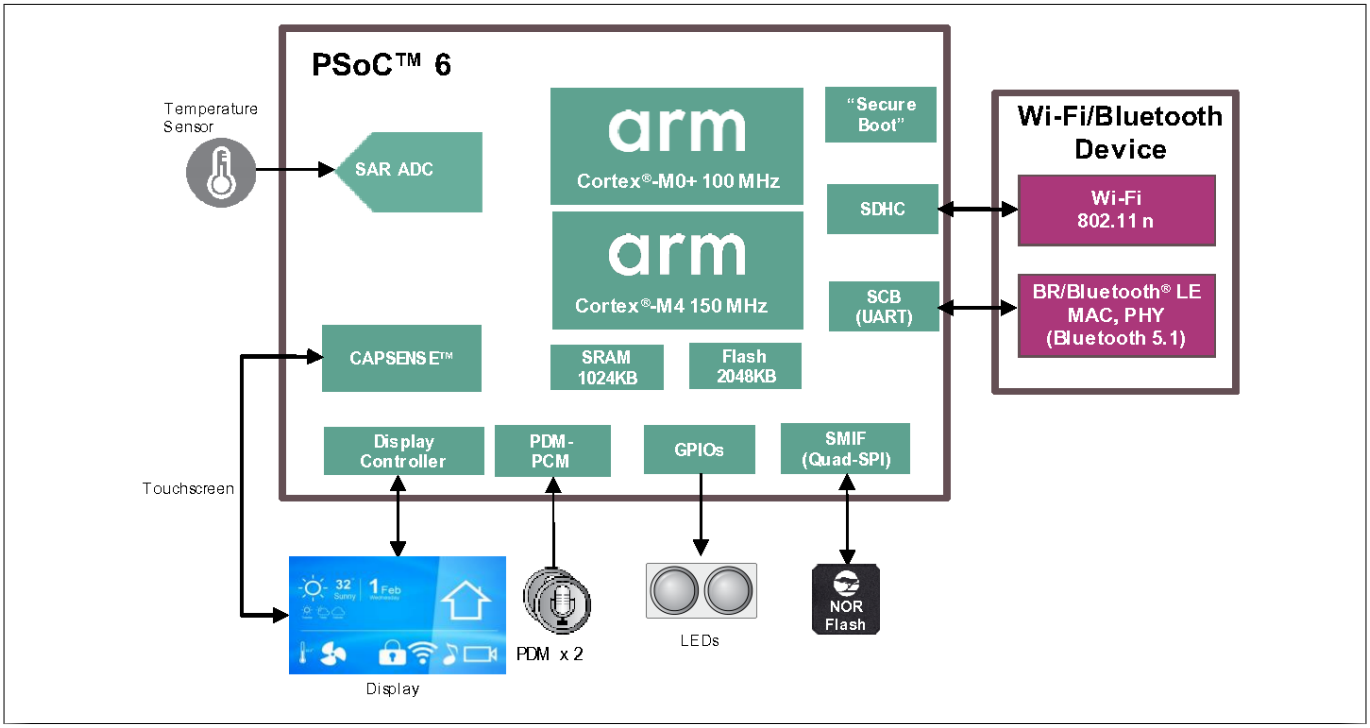



**图 1 使用 PSoC™ 6 MCU 的应用级框图**

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

- 降压转换器，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

## 1 简介

- 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 (Smart I/O) 和外设 (定时器计数器 PWM 或 TCPWM) 分别驱动显示器和 LED
- 与 Wi-Fi/Bluetooth®设备的 SDIO 接口可提供 IoT 云连接
- 由 CM0+核管理的产品安全功能和 CM4 核执行的应用程序功能

PSoC™ 6 共有 4 条产品线，以满足不同的应用需求。表 1 提供了不同产品系列的概述。

**表 1　　PSoC™ 6 MCU 产品线**

| 产品线 | 安全固件 | 器件系列 | 详情 | 应用 |
|---|---|---|---|---|
| 可编程 | 否 | CY8C61x | 单核: 150-MHz Arm® Cortex®-M4 | 物联网网关、智能家居、家电、人机界面、音频处理和工业集中器 |
| 性能 | 否 | CY8C62x | 双核架构: 150-MHz Arm® Cortex®-M4 和 100-MHz Cortex®-M0+ | |
| 连接 | 否 | CY8C63x | 双核架构: 150-MHz Arm® Cortex®-M4 和 100-MHz Cortex®-M0+<br>蓝牙低能耗(Bluetooth® Low Energy (LE))5.0 无线电，数据吞吐量 2-Mbps | 可穿戴设备、便携式医疗、工业物联网和智能家居 |
| 安全 | 安全启动 | CYB064x | 150-MHz Arm® Cortex®-M4，用于用户应用<br>硬件隔离, 100-MHz Arm® Cortex®-M0+，有权限访问内存和外设，以实现安全功能<br>蓝牙低能耗 Bluetooth® LE 5.0 无线电，数据吞吐量 2-Mbps<br>Arm®平台安全架构认证 - PSA L1, FIPS 140-2 | |
| | AWS 标准加密 | CYS064x | 150-MHz Arm® Cortex®-M4，用于用户应用<br>硬件隔离, 100-MHz Arm® Cortex®-M0+，有权限访问内存和外设，以实现安全功能<br>Arm®平台安全架构认证 - PSA L2 | 物联网网关、智能家居、家电、人机界面、音频处理和工业集中器 |

请注意，并非所有产品系列中的所有设备都具备所有功能。更多细节请参见设备数据表。

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)。

# 2 开发生态系统

## 2.1 PSoC™资源

这里提供了大量数据，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 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™ 6 MCU webpage
- Product 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。
- 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: 提供许多低成本的套件和屏蔽板，用于评估、设计和开发使用 PSoC™ 6 MCU 的不同应用。
- 培训视频: 提供有关产品和工具的视频培训，包括有关 PSoC™ 6 MCU 的专门系列。
- 技术支持: PSoC™ 6 社区论坛, 知识库文章

## 2.2 固件/应用开发

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，中间件库和以及使您能够创建 MCU 和无线应用程序的其他软件包。所有工具都可以运行在 Windows、macOS 和 Linux。ModusToolbox™还包括 Eclipse IDE, 提供与所有 ModusToolbox™工具的集成流程。同时也支持其它 IDE，包括 Visual Studio Code，IAR Embedded Workbench 和 Arm® MDK (µVision®)。

  ModusToolbox™支持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™软件用于 PSoC™ 6 MCU 的所有应用程序开发。有关更多信息，请参见 ModusToolbox™ tools package user guide。

  在 GitHub 网站上提供了库和实施软件。

  GitHub 上可用的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 MCU、蓝牙和蓝牙低功耗应用开发的全功能平台
  - 连接生态系统 - 一系列库文件提供 Wi-Fi 核心功能，包括连接性、安全性、固件升级支持和应用层协议的应用。
  - 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英飞凌设备安全地连接到 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 ModusToolbox™ tools package user guide 中的编译系统章节。

- **PSoC™ Creator:** 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 安装 ModusToolbox™工具包

参考 ModusToolbox™ tools package installation guide 了解更多细节。

### 2.2.2 选择 IDE

ModusToolbox™软件是最新一代的工具集，支持跨 Windows、Linux 和 macOS 平台。ModusToolbox™软件支持第三方 IDE，包括 Visual Studio Code、Arm® MDK (µVision)和 IAR Embedded Workbench。该工具包包括一个用于 Eclipse IDE 的实现，以方便你使用。该工具支持所有 PSoC™ 6 MCU 器件。相关的 BSP 和库配置器也可在所有三种主机操作系统上使用。

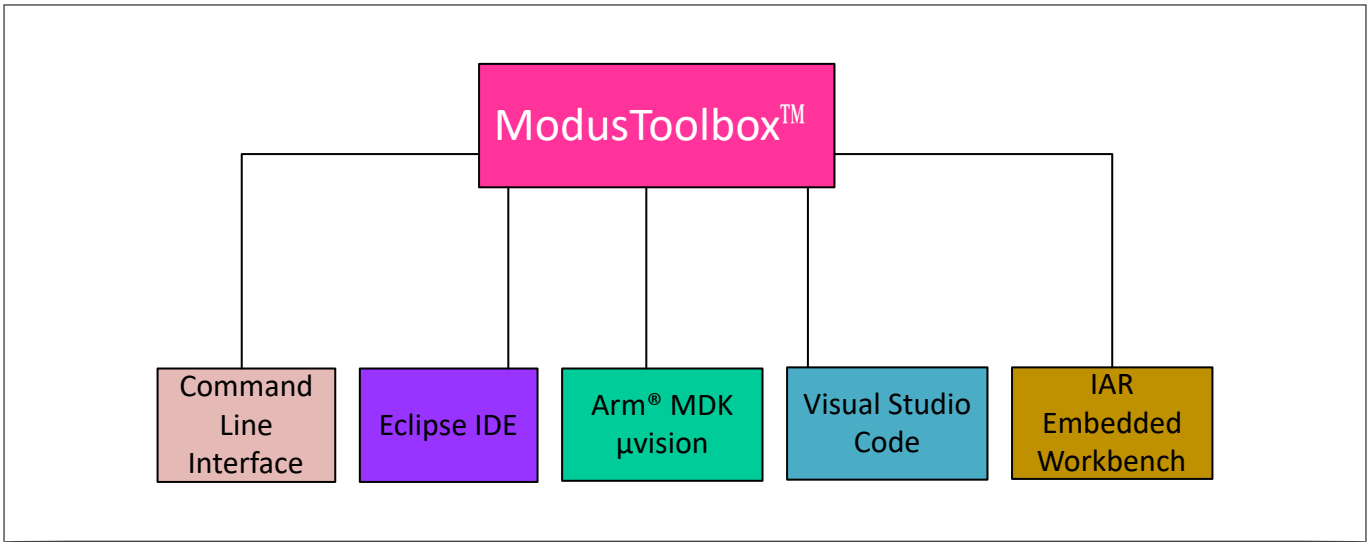



**图 2 ModusToolbox™环境**

ModusToolbox™ 2.x 和更低的版本不支持 PSoC™ 6 MCU 的某些功能，例如 UDB 和 USB。ModusToolbox™的较新版本支持 USB 主机功能，并通过真正的多核调试支持改善用户体验。

如果您想使用物联网器件构建物联网应用，或者您使用的是 PSoC™ Creator 不支持的 PSoC™ 6 MCU，建议使用 ModusToolbox™。

PSoC™ Creator 是使用已久的专有工具，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。

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。

### 2.2.3 ModusToolbox™™软件

ModusToolbox™是一套工具和软件，可为创建融合的 MCU 和无线系统提供身临其境的开发体验，并使您能够将我们的器件集成到现有的开发方法中。这些工具包括配置工具、低级驱动程序、库和操作系统支持，其中大部分与 Linux、macOS 和 Windows 主机环境兼容。

图 3 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™ tools package user guide。

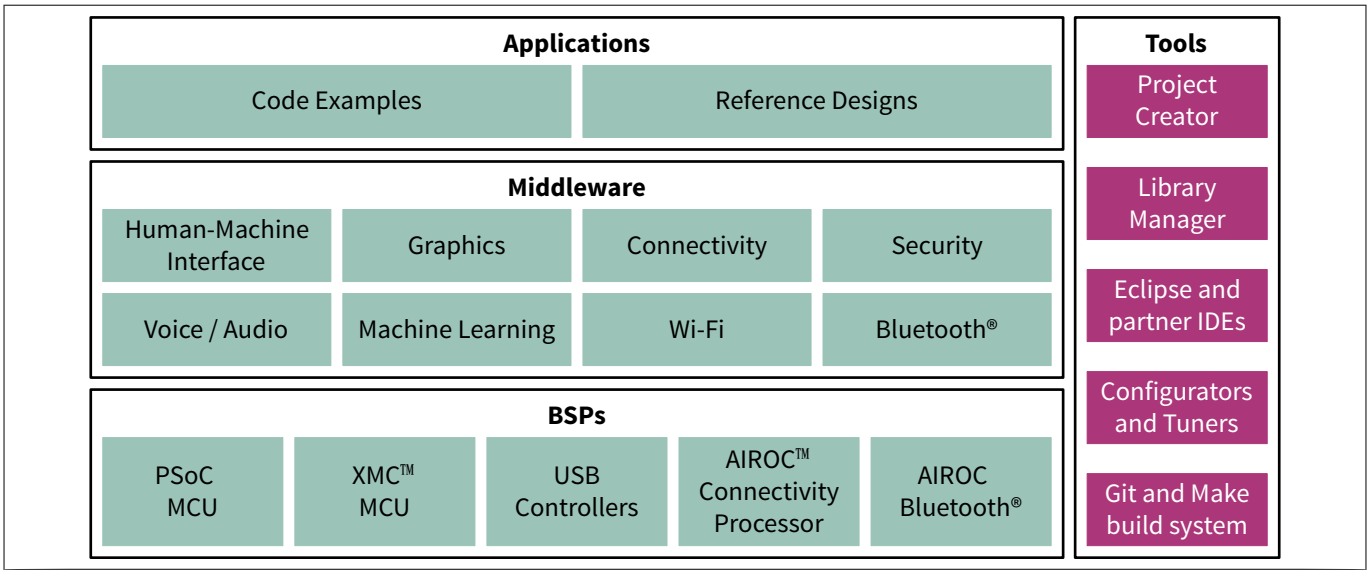



**图 3 ModusToolbox™™软件**

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

构建系统基础设施包括新的项目创建向导，可以独立于 Eclipse IDE、make 基础设施和其他工具运行。这意味着你可以选择你的编译器、IDE、RTOS 和生态系统，而不影响可用性或对我们业界领先的 CAPSENSE™ (人机界面)、AIROC™ Wi-Fi 和蓝牙、安全性和各种其他功能的访问。

ModusToolbox™生态系统的一部分是运行时软件，它可以帮助您使用连接组合器件，如 AIROC™ CYW43012 和 CYW43439 (以及其他)，与 PSoC™ 6 MCU 一起快速开发 Wi-Fi 和蓝牙应用。详见 ModusToolbox™ run-time software reference guide(ModusToolbox™运行时软件参考指南)。

设计配置器是帮助您为 BSP/中间件创建可配置代码的工具。跳转到配置器以了解更多信息。

图 4 是一个运行时软件图，展示了英飞凌器件使用 ModusToolbox™软件的一些应用功能。

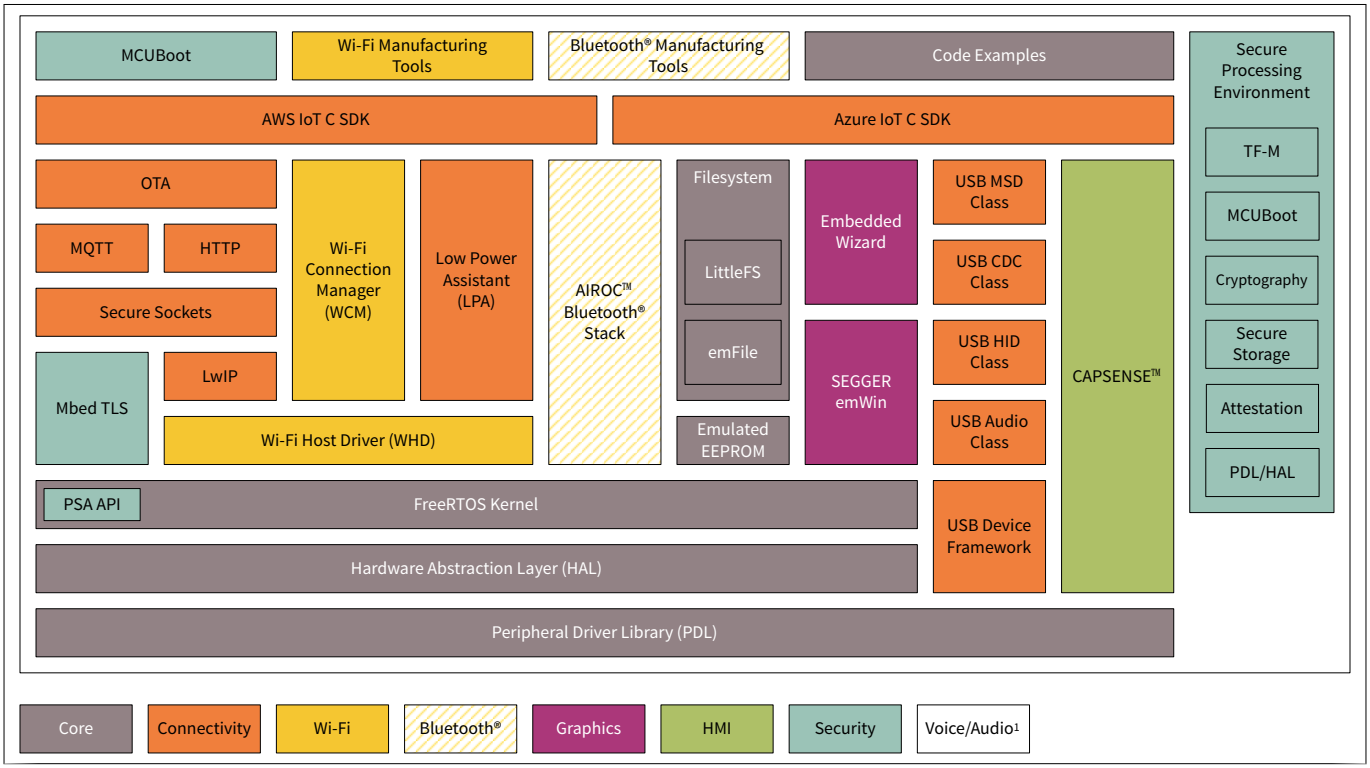



**图 4 ModusToolbox™运行时软件图**

以上所有开发流程均取决于提供的底层资源。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。BSP 是一组库，这些库提供 API 来初始化电路板并提供对电路板外围设备的访问。它包含低级资源 (例如 PSoC™ 6 MCU 的 PDL 库)，并具有板外设宏。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 BSP Assistant 创建你的 BSP。
- 硬件抽象层 (HAL) - 硬件抽象层 (HAL) 提供了一个高级接口，可在 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低级别的控制。
- PSoC™ 6 外设驱动程序库 (PDL) - 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 MCU 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 减少了了解寄存器用法和位结构的需求，从而简化了 PSoC™ 6 MCU 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。
- 中间件 - 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 (OTA，蓝牙，AWS IoT，Bluetooth® Low Energy, 安全套接字) 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 GitHub 存储库作为 Library 交付。

### 2.2.4 ModusToolbox™应用

随着 ModusToolbox™ v3.x 的发布，引入了多核支持，这使得文件夹结构与 ModusToolbox™之前的版本相比有了轻微改变。

ModusToolbox™有两种类型的应用程序。

- 单核应用
- 多核应用

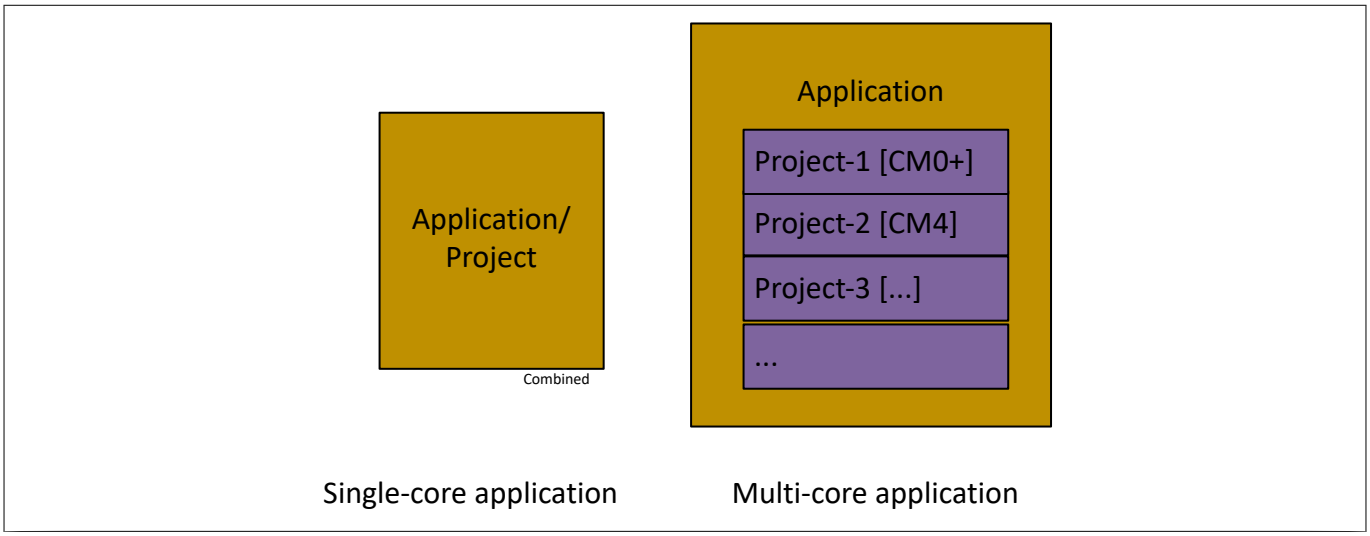



**图 5** 应用类型

## 2 开发生态系统

下面显示了一个单核应用程序样例的新文件夹结构:

```
<root>
  ApplicationName
  ->Makefile (MTB_TYPE=COMBINED)
  ->deps
      lib1.mtb (local)
      lib2.mtb (shared)
  ->libs
      lib1 (Infineon Git repo)
  ->bsps
      TARGET_BSP1 (not an Infineon Git repo; completely app-owned)
  ->templates
      TARGET_BSP1
        design.modus
        design.capsense
  ->main.c
  ->helper.h
  ->helper.c
  mtb_shared
    lib2/... (Infineon Git repo)
```

**图 6** 单核应用的文件夹结构

下面显示了一个多核应用程序样例的新文件夹结构:

```
<root>
  ApplicationName
  ->Makefile (MTB_TYPE=APPLICATION)
  ->common.mk
  ->common_app.mk
  ->bsps
      TARGET_BSP1 (not an Infineon Git repo; completely app-owned)
  ->templates
      TARGET_BSP1
        design.modus
        design.capsense
  ->project1
      Makefile (MTB_TYPE=PROJECT)
      deps
        lib3.mtb (local)
        lib4.mtb (shared)
      libs
        lib3 (Infineon Git repo)
      main.c
      project1_helper.h
      project1_helper.c
  ->project2
      Makefile (MTB_TYPE=PROJECT)
      deps
        lib5.mtb (local)
        lib6.mtb (shared)
      libs
        lib5 (Infineon Git repo)
      main.c
      project2_helper.h
      project2_helper.c
  mtb_shared
    lib4/... (Infineon Git repo)
    lib6/... (Infineon Git repo)
```

**图 7　　多核应用的文件夹结构**

使用 ModusToolbox™3.x 版本的新流程可以支持一个应用程序中的多个项目。对于多核应用，有多个项目，但每个核只有一个项目。应用程序有应用程序拥有的 BSP，这意味着 BSP 将是一个多核应用程序内部所有项目所共有的。

进一步说，本文第 4 节介绍了使用 ModusToolbox™软件创建一个新的单核应用程序。

### 2.2.5 PSoC™ 6 软件资源

PSoC™ 6 MCU 软件包括配置、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 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。

#### 2.2.5.1 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。

有两种类型的配置器: 配置 MCU 硬件特定项目的 BSP 配置器和配置中间件库选项的库配置器。

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。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、引脚和基本外设 (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, 以及调谐 CAPSENSE™应用。
- **LIN 配置器:** 配置 LIN 中间件并生成所需的配置。
- **ML 配置器:** 将预先训练好的模型与一组优化参数配合到目标设备上 (仅作为独立包的一部分)。
- **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。
- **Bluetooth®配置器:** 配置蓝牙设置。
- **EZ-PD™配置器:** 配置参数并选择 PD 堆栈中间件的功能。
- **安全策略配置器:** 为"安全" MCU 设备打开、创建或改变策略配置文件。
- **SegLCD 配置器:** 配置并生成 SegLCD 驱动所需的结构。

上面的每个配置器都会创建自己的文件 (例如: CAPSENSE™的 *design.cycapsense*)。BSP 配置器文件 (如 *design.modus* 或 *design.cycapsense*) 作为 BSP 的一部分，以默认配置提供，而库配置器 (如 design.cybt) 则由应用程序提供。当基于英飞凌 BSP 创建一个应用程序时，该应用程序会使用英飞凌 BSP repo 中的 BSP 配置器文件。您可以使用 ModusToolbox™软件根据您的应用要求定制/创建所有的配置器文件。请参阅 BSP 助手来创建您的定制 BSP。更多细节请参见 ModusToolbox™帮助。

### 2.2.5.2 PSoC™ 6 MCU 库管理

应用程序可以有项目的 shared/local 库。如果需要，不同的项目可以使用同一个库的不同版本。共享库被下载到 `mtb_shared` 目录下。应用程序应该使用 `deps` 文件夹来添加库的依赖文件。`deps` 文件夹包含以`.mtb` 文件为扩展名的文件，ModusToolbox™使用它来下载其 git 仓库。这些库是 ModusToolbox™项目的直接依赖项。

库管理器有助于添加/删除/更新你的项目的库。它还可以使用 GitHub 上的清单库来识别特定的库是否直接依赖任何其他库，并获取该特定库的所有依赖关系。这些依赖库是 ModusToolbox™项目的间接依赖项。这些依赖项可以在 `libs` 文件夹下看到。欲了解更多信息，请参阅位于`<install_dir> /ModusToolbox/tools_<version>/library-manager/docs/library-manager.pdf` 的库管理器用户指南。

### 2.2.5.3 PSoC™ 6 MCU 的软件开发

ModusToolbox™生态系统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 Bluetooth® LE 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 Bluetooth® LE 堆栈或 CAPSENSE™。BSP 配置器文件用于使用外设驱动库代码来配置设备外设、引脚和内存。中间件以独立的库形式交付给每个特性/功能，这样它就可以在多个平台上使用。例如，抽象-rtos、lwip、usb 等。

希望在寄存器级别工作的固件开发人员应参考 PDL 的驱动程序源代码。PDL 包括你的项目所需的所有特定设备的头文件和启动代码。它还可作为每个驱动程序的参考。因为 PDL 是以源代码的形式提供的，你可以看到它是如何在寄存器级别访问硬件的。

有些设备不支持特定的外设。PDL 是任何支持设备的所有驱动程序的超集。这种超集设计意味着:

- 所有初始化、配置和使用外设所需的 API 元素都可用。
- PDL 适用于各种 PSoC™ 6 MCU，而不受可用外设的限制。
- PDL 包括错误检查，以确保目标外设存在于所选器件上。

这使得代码能够在 PSoC™ 6 MCU 系列的各个成员中保持兼容性，只要外设是可用的。器件头文件规定了一个器件可用的外设。如果您编写的代码试图使用不支持的外设，您将在编译时得到一个错误。在编写使用外设的代码之前，请查阅特定设备的数据手册以确认对该外设的支持。

如图 8 所示，使用 ModusToolbox™软件，你可以:

1. 选择一个 BSP (Project Creator)。
2. 根据启动程序的列表创建一个新的应用程序，按每个应用程序支持的 BSP 进行筛选 (Project Creator)。
3. 添加 BSP 或中间件库 (Library Manager)。
4. 使用 PSoC™ 6 MCU 的 HAL 或 PDL 开发您的应用固件 (选择 IDE 或命令行)。

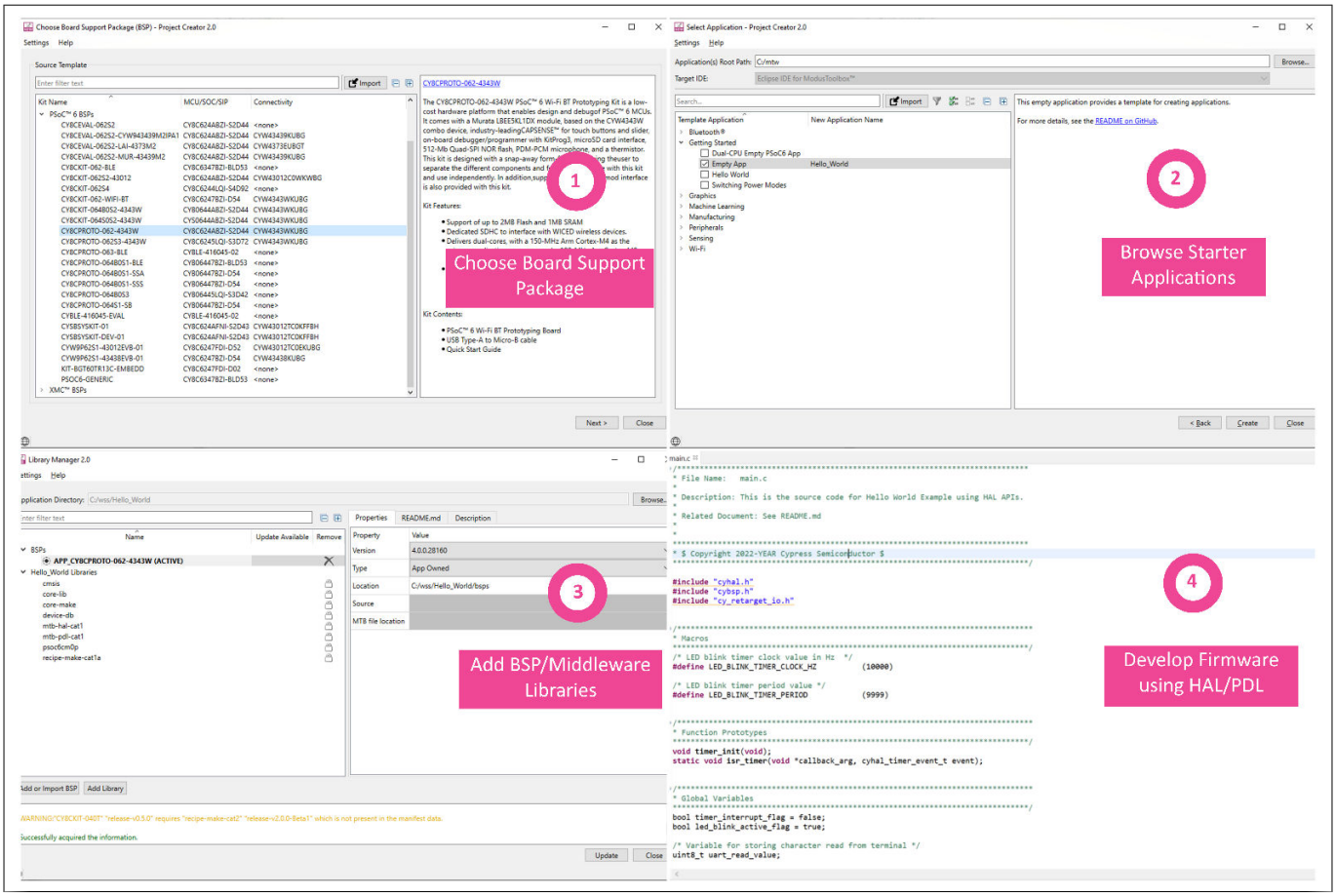



图 8 ModusToolbox™资源和中间件

### 2.2.6 ModusToolbox™帮助

ModusToolbox™生态系统提供文档和培训。访问它的一个方法是启动 Eclipse IDE for ModusToolbox™，并导航到以下**帮助**菜单项。

选择 **Help > ModusToolbox™ General Documentation**:

- **ModusToolbox™文档索引:** 提供简要描述和链接到作为 ModusToolbox™软件一部分的各种类型的文档。
- **ModusToolbox™安装指南:** 提供安装 ModusToolbox™软件的说明。
- **ModusToolbox™用户指南:** 本指南主要包括 ModusToolbox™构建、编程和调试应用程序的方面。它还包括与 IDE 一起安装的工具的各个方面。
- **ModusToolbox™培训课程材料:** 链接到 https://github.com/Infineon/training-modustoolbox 的培训材料。
- **发布说明**

关于 Eclipse IDE for ModusToolbox™的文档，选择 **Help > Eclipse IDE for ModusToolbox™ General Documentation**:

- **快速入门指南:** 为您提供使用 Eclipse IDE for ModusToolbox™的基础知识。
- **用户指南:** 提供关于创建应用程序以及使用 Eclipse IDE 构建、编程和调试应用程序的说明。
- **Eclipse IDE for ModusToolbox™帮助:** 提供关于如何创建新的应用程序、更新应用程序代码、更改中间件设置和编程/调试应用程序的说明。
- **Eclipse IDE 生存指南**

## 2.3 支持其他 IDE

您可以使用喜欢的 IDE (例如 IAR Embedded Workbench, Keil µVision 5 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 Eclipse 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。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 ModusToolbox™ tools package user guide。

## 2.4 FreeRTOS 对 ModusToolbox™的支持

向 ModusToolbox™应用程序项目添加本地 FreeRTOS 支持就像添加任何中间件库一样。您可以使用库管理器 (Library Manager) 将 FreeRTOS 中间件导入您的应用程序。如果使用 Eclipse IDE，选择应用程序项目，然后单击“**Quick Panel**”中的“**Library Manager**”链接。从 **Core** 对话框中点击 **Add Library** 并选择 **freertos**，如图 9 所示。

指向 FreeRTOS 中间件的`.mtb` 文件已添加到应用程序项目的 `deps` 目录中。中间件内容也被下载并放置在名为 **freertos** 的相应文件夹中。默认位置位于名为 mtb_shared 的共享资产存储库中。要继续使用 FreeRTOS，请遵循 FreeRTOS documentation 的“快速入门 (Quick Start)”部分中的步骤。

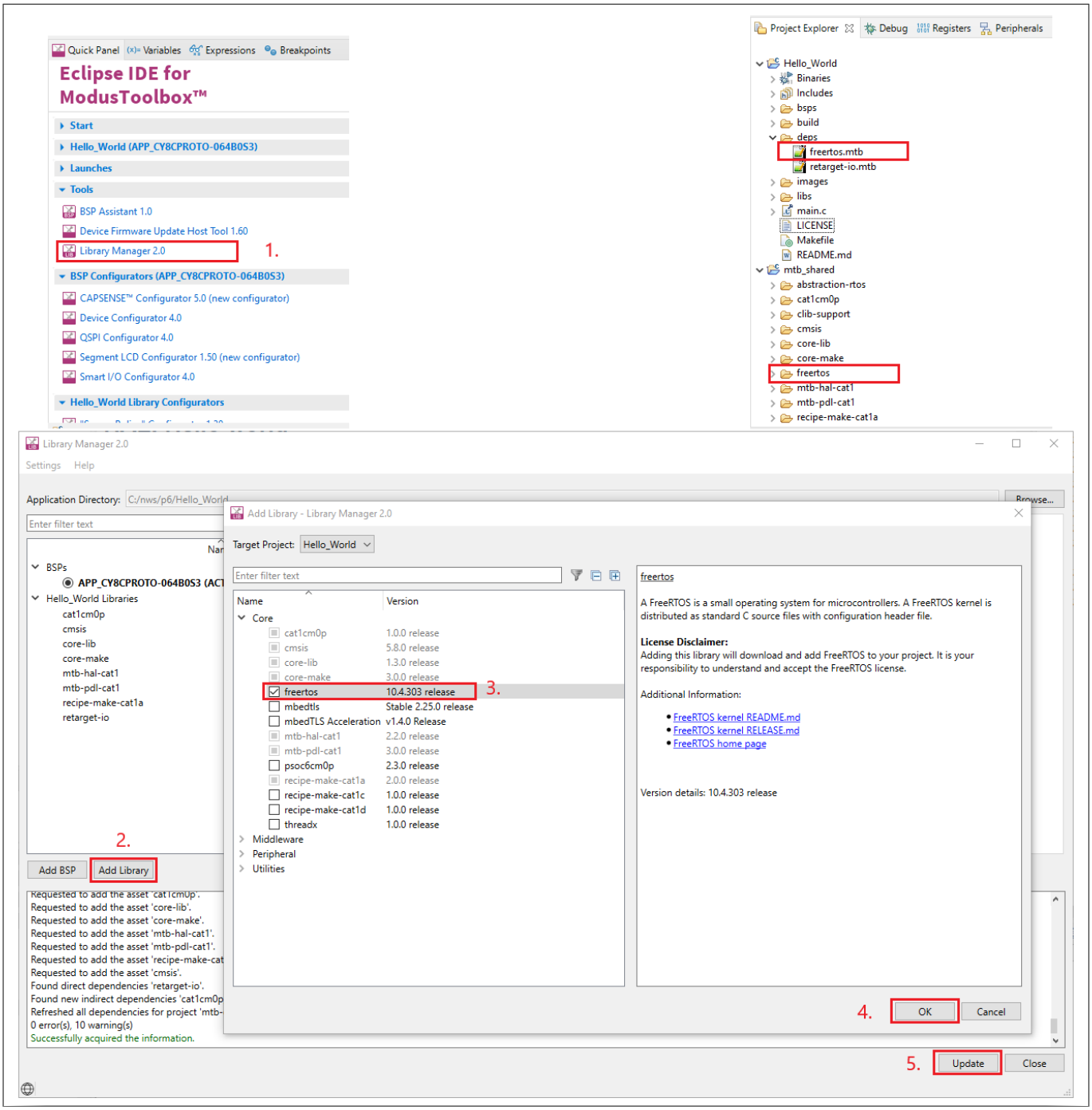



**图 9　　在 ModusToolbox™应用程序中导入 FreeRTOS 中间件**

## 2.5　使用 Eclispse IDE 编程/调试

所有 PSoC™ 6 套件具有 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。

Eclipse IDE 需要 KitProg3，并使用 OpenOCD 协议来调试 PSoC™ 6 MCU 应用。它还支持使用行业标准的探针 (如 Segger J-Link) 进行 GDB 调试。

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

*KitProg3。ModusToolbox 包含 fw-loader 命令行工具。请参考 Eclipse IDE for ModusToolbox™ user guide 中的“PSoC™ 6 Programming/ Debugging - KitProg Firmware Loader”部分。*

有关使用 ModusToolbox™在 PSoC 器件上调试固件的更多信息，请参考 Eclipse IDE for ModusToolbox™ user guide 中的“*编程和调试*”部分。

## 2.6 PSoC™ 6 MCU 开发套件

**表 2 开发套件**

| 产品线 | 开发套件 |
|---|---|
| 性能 | PSoC™ 6 WiFi-Bluetooth® Pioneer 套件 (CY8CKIT-062-WiFi-BT)<br>PSoC™ 6 Wi-Fi Bluetooth®原型套件 (CY8CPROTO-062-4343W)<br>PSoC™ 62S2 Wi-Fi Bluetooth® Pioneer 套件 (CY8CKIT-062S2-43012)<br>PSoC™ 62S3 Wi-Fi Bluetooth®原型套件 (CY8CPROTO-062S3-4343W)<br>PSoC™ 62S1 Wi-Fi Bluetooth® Pioneer 套件 (CYW9P62S1-43438EVB-01)<br>PSoC™ 62S1 Wi-Fi Bluetooth® Pioneer 套件 (CYW9P62S1-43012EVB-01)<br>PSoC™ 62S4 Pioneer 套件 (CY8CKIT-062S4) |
| 连接 | PSoC™ 6 Bluetooth® LE Pioneer 套件 (CY8CKIT-062-BLE)<br>PSoC™ 6 Bluetooth® LE 原型套件 (CY8CPROTO-063-BLE) |
| 安全 | PSoC™ 64 “Secure Boot” Wi-Fi Bluetooth® Pioneer 套件 (CY8CKIT-064B0S2-4343W)<br>PSoC™ 64 Standard “Secure” - AWS Wi-Fi Bluetooth® Pioneer 套件 (CY8CKIT-064S0S2-4343W) |

有关 PSoC™ 6 MCU 和屏蔽模块的完整套件列表，请参见 Microcontroller (MCUs) kits 页面。

# 3 器件特性

PSoC™ 6 MCU 产品线具有广泛的功能，如图 10 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)，和 参考文档。

- **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

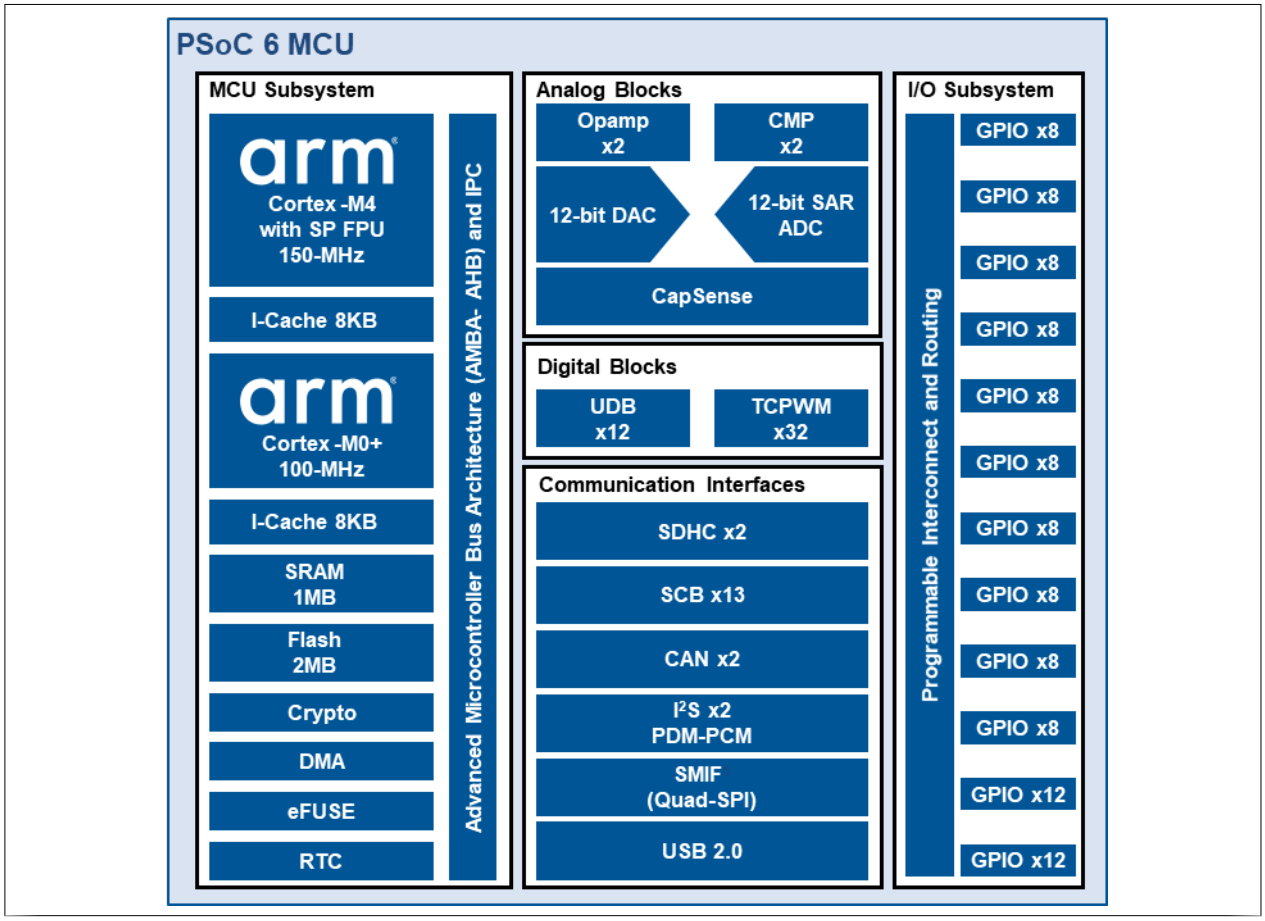



**图 10 PSoC™ 6 MCU 框图**

- **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
- USB 全速器件接口

• **可编程模拟模块**
- 最多两个运算放大器可以在系统深度睡眠模式下工作
- 多达两个 12 位 SAR ADC，最大采样率为 2 Msps，并能在部分 PSoC™ 6 MCU 的系统深度睡眠模式下工作。
- 一个 12 位、500 ksps 的电压模式 DAC。
- 最多两个低功耗比较器，可用于从所有低功耗模式中唤醒设备。
- 1.2V 带隙基准，1%容差，用于 SAR ADC 和 DAC。

• **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 Eclipse IDE for ModusToolbox™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 Eclipse IDE for ModusToolbox™。

## 4.1 前提条件

在开始之前，请确保您拥有适合您的 PSoC™ 6 MCU 产品系列的开发套件，并安装了所需的软件。在项目创建期间，您还需要接入互联网访问 GitHub 仓库。

### 4.1.1 硬件

以下显示的示例设计是为 PSoC™ 6 Wi-Fi Bluetooth® prototyping kit (CY8CPROTO-062-4343W)开发的。但是，您可以为其他开发套件构建项目。但是，您可以为其他开发套件构建项目。请参阅 "使用这些说明"部分。

### 4.1.2 软件

- ModusToolbox™ 3.0 或更高版本

安装软件后，请参考 ModusToolbox™ tools package user guide 来了解软件的概况。

## 4.2 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: 查看与修改设计配置
- 第 3 部分: 写固件
- 第 4 部分: 编译应用程序
- 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 Bluetooth®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 Bluetooth® Pioneer 套件 (CY8CKIT-062-WiFi-BT)
- PSoC™ 6 Bluetooth® LE Pioneer 套件 (CY8CKIT-062-BLE)
- PSoC™ 6 Bluetooth® LE 原型套件 (CY8CPROTO-063-BLE)
- PSoC™ 62S2 Wi-Fi Bluetooth® Pioneer 套件 (CY8CKIT-062S2-43012)
- PSoC™ 62S1 Wi-Fi Bluetooth® Pioneer 套件 (CYW9P62S1-43438EVB-01)
- PSoC™ 62S1 Wi-Fi Bluetooth® Pioneer 套件 (CYW9P62S1-43012EVB-01)
- PSoC™ 62S3 Wi-Fi Bluetooth®原型套件 (CY8CPROTO-062S3-4343W)
- PSoC™ 64 “Secure Boot” Wi-Fi Bluetooth® Pioneer 套件 (CY8CKIT-064B0S2-4343W)
- PSoC™ 62S4 Pioneer 套件 (CY8CKIT-062S4)

## 4.3 关于设计

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

器件复位后，英飞凌提供的预置 CM0+应用程序映像将启用 CM4 核并配置 CM0+核进入睡眠状态。CM4 核使用 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。

## 4.4 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 App** 启动程序并手动添加 **Hello World** 启动程序的功能。说明中使用了 ModusToolbox™的 Eclipse IDE，但如果你愿意，可以使用任何 IDE 或命令行。

如果你熟悉用 ModusToolbox™软件开发项目，你可以直接使用 **Hello World** 的启动程序。它是一个完整的设计，为支持的套件编写了所有的固件。你可以通过说明，观察代码示例中的步骤是如何实现的。

如果你从头开始并遵循本应用说明中的所有指示，你可以使用 **Hello World** 代码示例作为参考，同时遵循说明。

启动 Eclipse IDE for ModusToolbox™集成开发环境来开始创建过程。请注意，ModusToolbox™软件需要接入互联网才能成功地将启动程序克隆到你的机器上。

1. **选择一个新的工作区**

   在启动时，Eclipse IDE for 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，如图 11 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 Eclipse IDE 将为您创建目录。

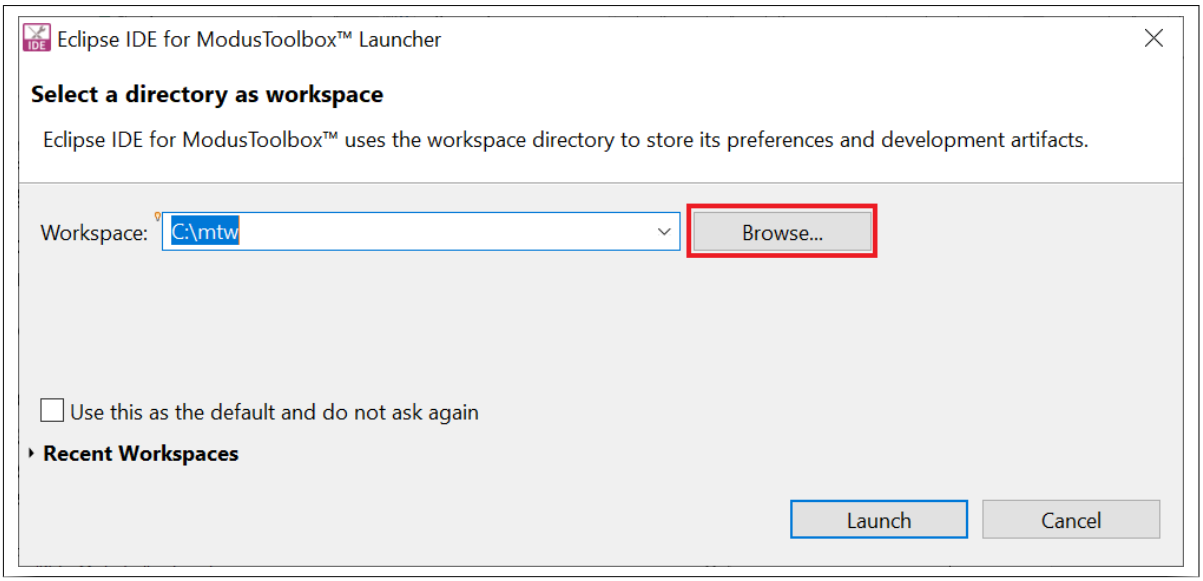



**图 11 选择一个目录作为 Wokspace**

2. **创建一个新的 ModusToolbox™应用程序**

   a. 在“快速面板” (Quick Panel) 的“开始” (Start) 组中单击“新建应用程序” (**New Application**)。

   b. 或者，您可以选择 **File > New > ModusToolbox™ Application**，如图 12 所示。

   出现 Project Creator 应用程序窗口。

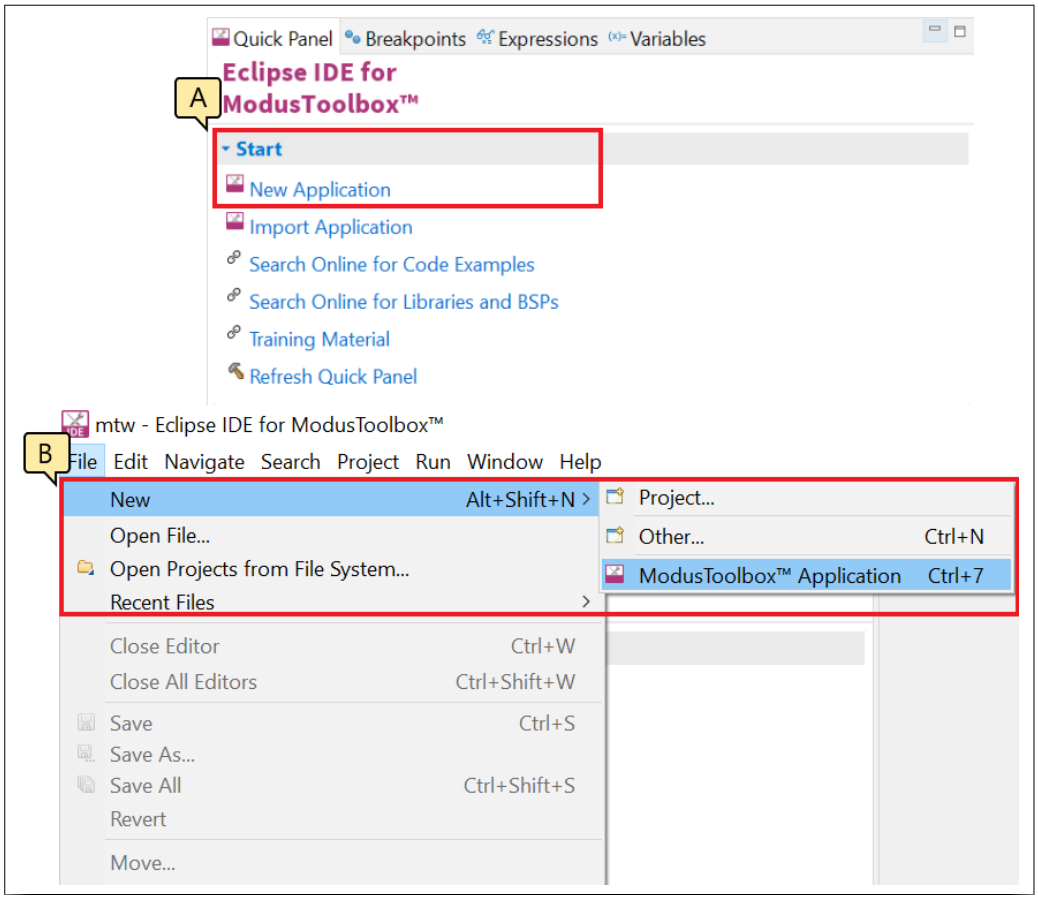



**图 12　　创建新的 ModusToolbox™应用程序**

**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**。接下来的步骤使用 **CY8CPROTO-062-4343W**。有关此步骤的帮助，请参见图 13。

**b.**　单击 **Next**。

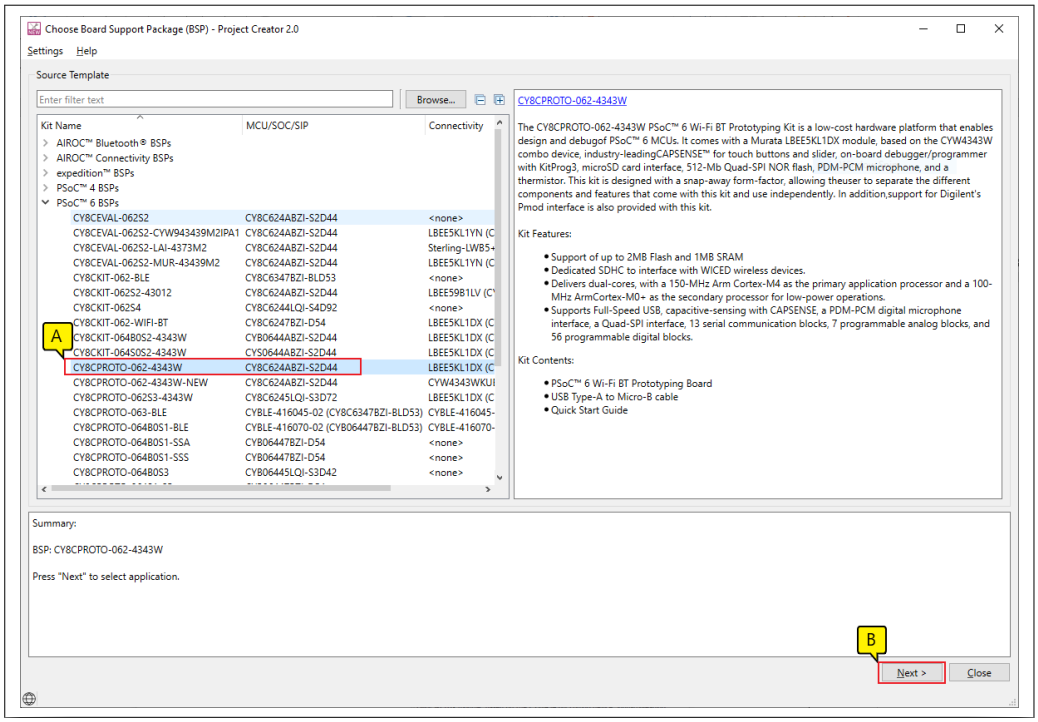



图 13 选择目标硬件

c. 在 **Select Application** 对话框中，选择 **Empty App** 启动应用程序，如图 14 所示。

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 **Create** 创建应用程序，如图 14 所示，等待项目创建成功后，Project Creator 自动关闭。

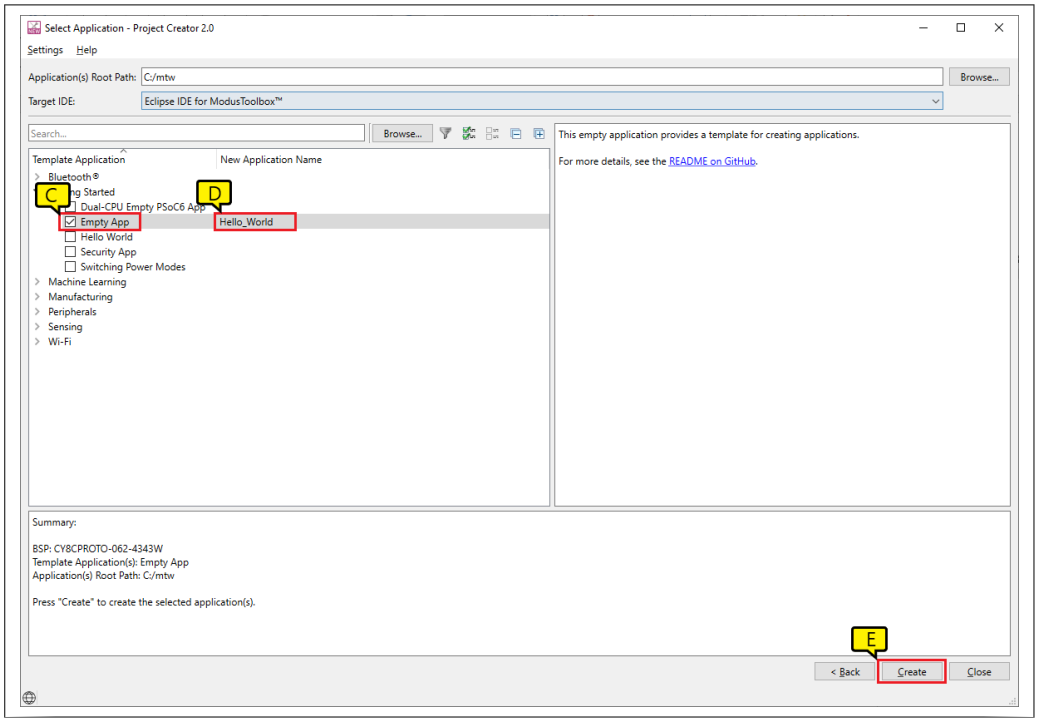



图 14 选择启动应用程序

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-Bluetooth®原型套件 (CY8CPROTO-062-4343W)上的默认设备，配置 CYW4343WKUBG Wi-Fi/Bluetooth®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，请参考自定义 BSP 应用笔记或 BSP 助手用户指南。

## 4.5 第 2 部分: 查看与修改设计配置

图 15 显示了 Eclipse IDE 项目浏览器界面，显示了应用程序项目的结构。

PSoC™ 6 MCU 应用程序包含一个项目，用于为 CM4 核心开发代码。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。

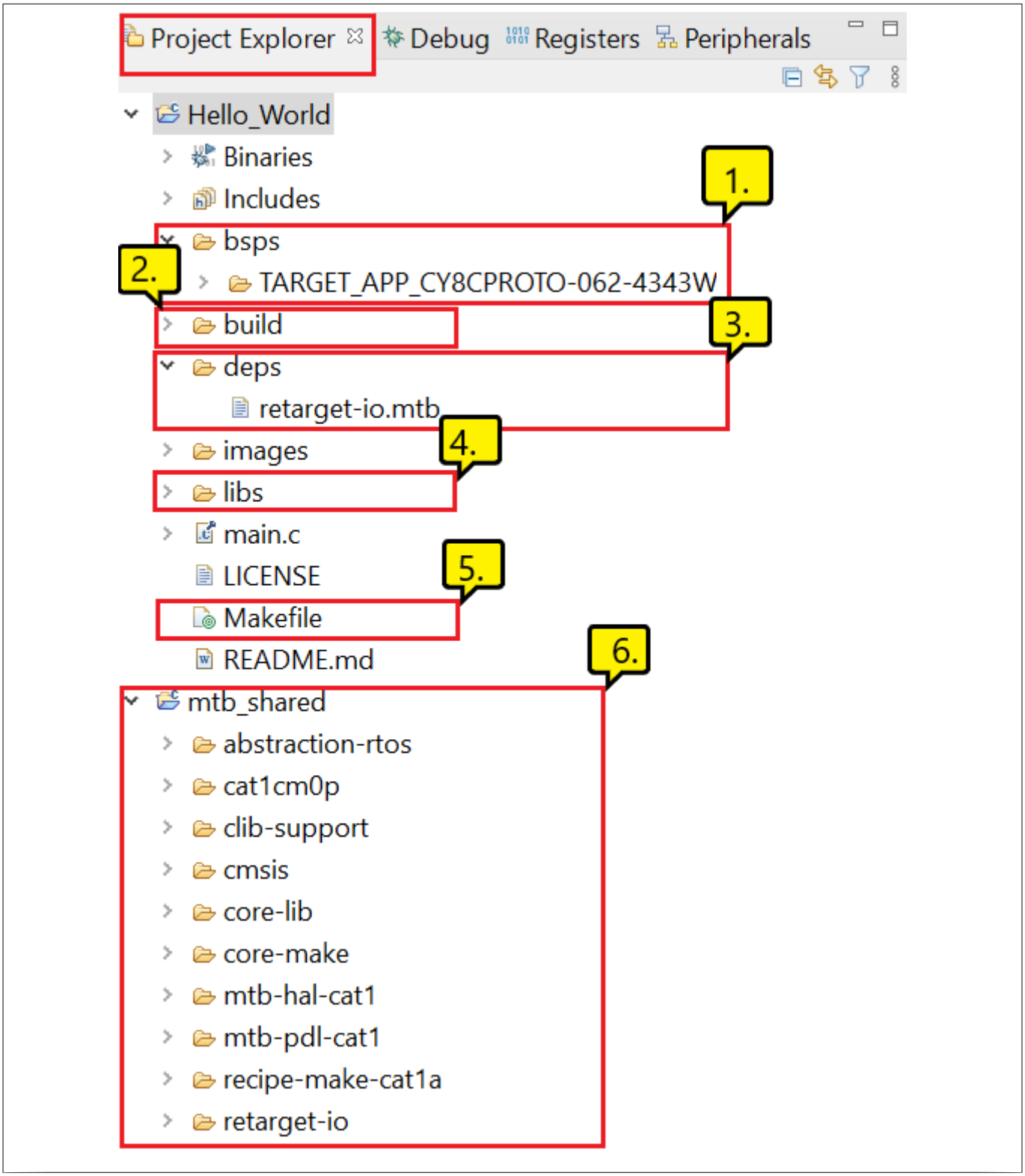



**图 15　　项目资源管理器视图**

**1.** BSP 提供的文件都在 `bsps` 文件夹中，并列在 `TARGET_<bsp name>`子文件夹下。器件和外设配置器的所有输入文件都在 BSP 里面的 config 文件夹中。BSP 中的 `GeneratedSource` 文件夹包含由配置器生成的文件，其前缀为 `cycfg_`。这些文件包含 BSP 所定义的设计配置。从 ModusToolbox™ 3.x 或更高版本开始，你可以直接为你的应用程序定制 BSP 的配置器文件，而不是用自定义设计配置器文件覆盖默认设计配置器文件，因为 BSP 完全由应用程序拥有。

BSP 文件夹还包含链接器脚本和电路板上使用的 PSoC™ 6 MCU 的启动代码。

2. `build` 文件夹包含了项目的构建所产生的所有工件。输出文件按目标 BSP 组织。
3. `deps` 文件夹中包含`.mtb` 文件，它提供了 ModusToolbox 拉取应用程序直接引用的库的位置。这些文件通常包含整个库的 GitHub 位置。`.mtb` 文件还包含一个 git Commit Hash 或 Tag，指示要获取库的哪个版本以及库的存储路径。

   例如，这里 `retarget-io.mtb` 指向 `mtb://retarget-io#latest-v1.X#$$ASSET_REPO$$/retarget-io/latest-v1.X`，变量`$$ASSET_REPO$$`指向共享位置的根，默认为 `mtb_shared`。如果该库必须是应用程序的本地库，而不是共享库，请使用`$$LOCAL$$`代替`$$ASSET_REPO$$`。
4. `libs` 文件夹还包含`.mtb` 文件。这些文件指向作为 BSP 或其他库的依赖关系而间接包含的库。对于每个间接依赖，库管理器会在这个文件夹中放置一个`.mtb` 文件。这些文件已经根据 `deps` 文件夹中可用的目标进行了填充。

   例如，使用 BSP 库文件 `CY8CPROTO-062-4343W` 用以下`.mtb` 文件填充 `libs` 文件夹：*cmsis.mtb*, ***core-lib.mtb***, *core-make.mtb*, *mtb-hal-cat1.mtb*, *mtb-pdl-cat1.mtb*, *cat1cm0p.mtb*, *recipe-make-cat1a.mtb*。

   `libs` 文件夹中包含了 `mtb.mk` 文件，它存储了应用程序所需的所有库的相对路径。构建系统使用这个文件来查找应用程序所需的所有库。

   `libs` 文件夹中的所有内容均由库管理器生成，因此您不应手动编辑该文件夹中的任何内容。
5. 一个应用程序包含一个 Makefile，它位于应用程序的根目录下。这个文件包含 make 工具用来编译和连接应用程序项目的指令集。在一个应用程序中可能有不止一个项目。在这种情况下，在应用程序层面有一个 Makefile，在每个项目内有一个。参见 AN215656 - PSoC™ 6 MCU 双核系统设计，了解与多项目应用相关的细节。
6. 默认情况下，当创建一个新的应用程序或在现有的应用程序中添加一个库到现有应用并指定为共享时，所有的库都会被放置在应用程序目录旁边的 `mtb_shared` 目录中。

   `mtb_shared` 文件夹在一个工作区的不同应用程序之间共享。如果有必要，不同的应用程序可以使用不同版本的共享库。

### 4.5.1 打开设备配置器

BSP 配置器文件在 `bsps/TARGET_<BSP-name>/config` 文件夹中。例如，从**项目资源管理器**中点击 **<Application-name>**，然后点击**快速面板**中的 **Device Configurator** 链接，在**设备配置器**中打开 `design.modus` 文件，如图 16 所示。你也可以在各自的配置器中打开其他配置文件，或者点击**快速面板**中的相应链接。

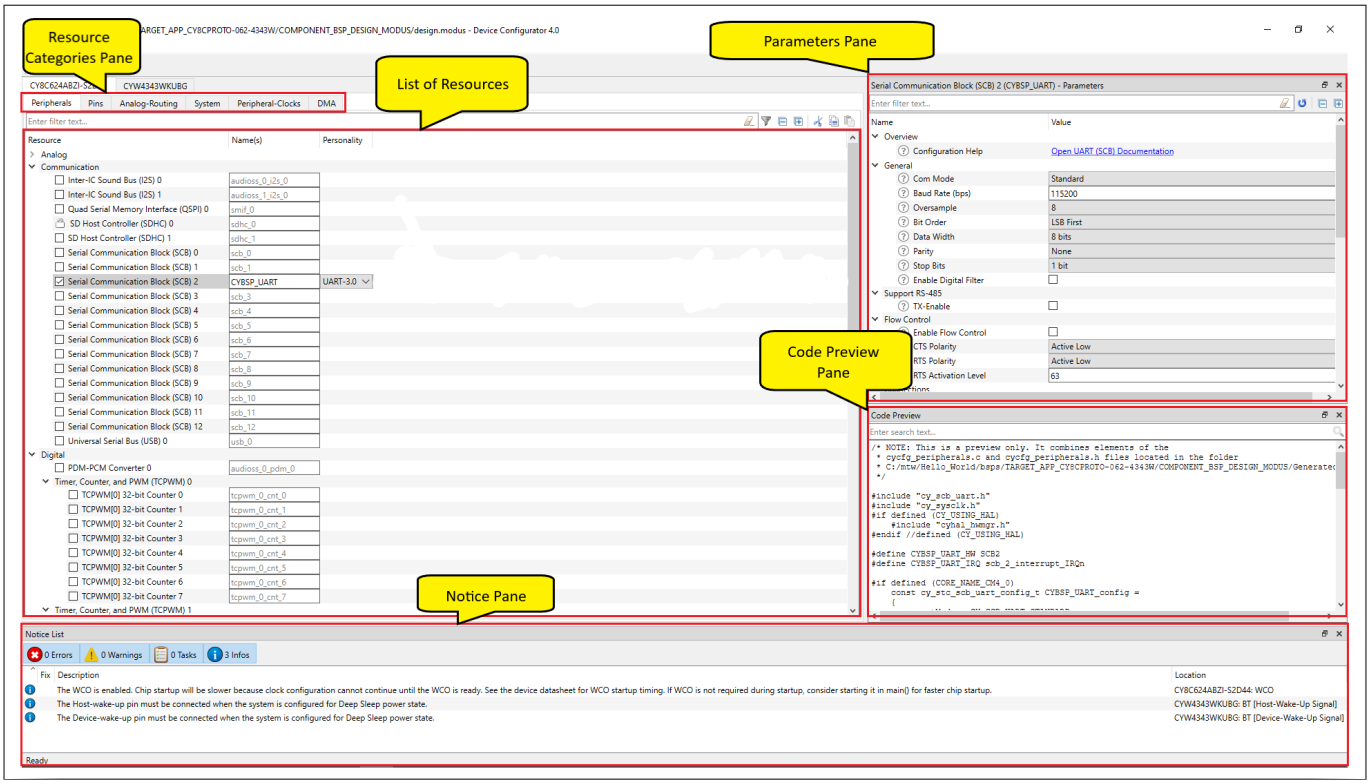



**图 16　　设备配置器**

设备配置器 (**DeviceConfigurator**) 窗口提供了一个资源类别 (**Resources Categories**) 窗格。在这里，您可以从资源列表 (**List of Resources**) 中选择设备中可用的不同资源，如外设、引脚和时钟。

您可以通过为资源选择特性 (**Personality**) 来选择资源的行为方式。例如，**串行通信块 (SCB)** 资源可以有 **EZI2C**、**I2C**、**SPI** 或 **UART** 特性。别名 (**Alias**) 是资源的名称，用于固件开发。可以通过使用逗号来隔开一个或多个别名 (没有空格)。

在参数 (**Parameters**) 窗格中，您可以为每个已启用的资源和选定的特性输入配置参数。代码预览 (**Code Preview**) 窗格显示了根据所选配置参数生成的配置代码。该代码被填充在 `GeneratedSource` 文件夹中的 `cycfg_`文件中。参数窗格和代码预览窗格可能会显示为页签而不是窗口，但内容相同。

任何因配置而产生的错误、警告和信息消息都会显示在通知(Notices)窗格中。

目前，设备配置器支持使用 PDL 源代码进行配置。如果你选择在你的应用程序中使用 HAL 库，那么你不需要在这里做任何设备配置的改变。应用程序项目包含帮助你为 CM4 内核创建应用程序的源文件 (例如，`main.c`)，而 CM0+应用程序是以默认的 C 文件提供的 (`CY8C624ABZI-D44` 器件的 `psoc6_02_cm0p_sleep.c`)。参见 cat1cm0p 库。这个 C 文件被编译并与 CM4 图像连接，作为正常构建过程的一部分。

在开发过程的这一点上，我们已经准备好向设计中添加所需的中间件。Hello World 应用程序需要的中间件只有 retarget-io 库。

### 4.5.2　　添加 retarget-io 中间件

在此步骤中，你将添加 retarget-io 中间件，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。

1. 在 **Quick Panel** 中，单击 **Library Manager** 链接。
2. 在随后的对话框中，选择 **Add Libraries** 选项卡。
3. 在 **Peripherals** 下，选择并启用 retarget-io。
4. 点击 **OK** 和 **Update**。

使用 retarget-io 中间件所需的文件被添加到 `mtb_shared > retarget_io` 文件夹中，`.mtb` 文件也被添加到 `deps` 文件夹中，如图 17 所示。

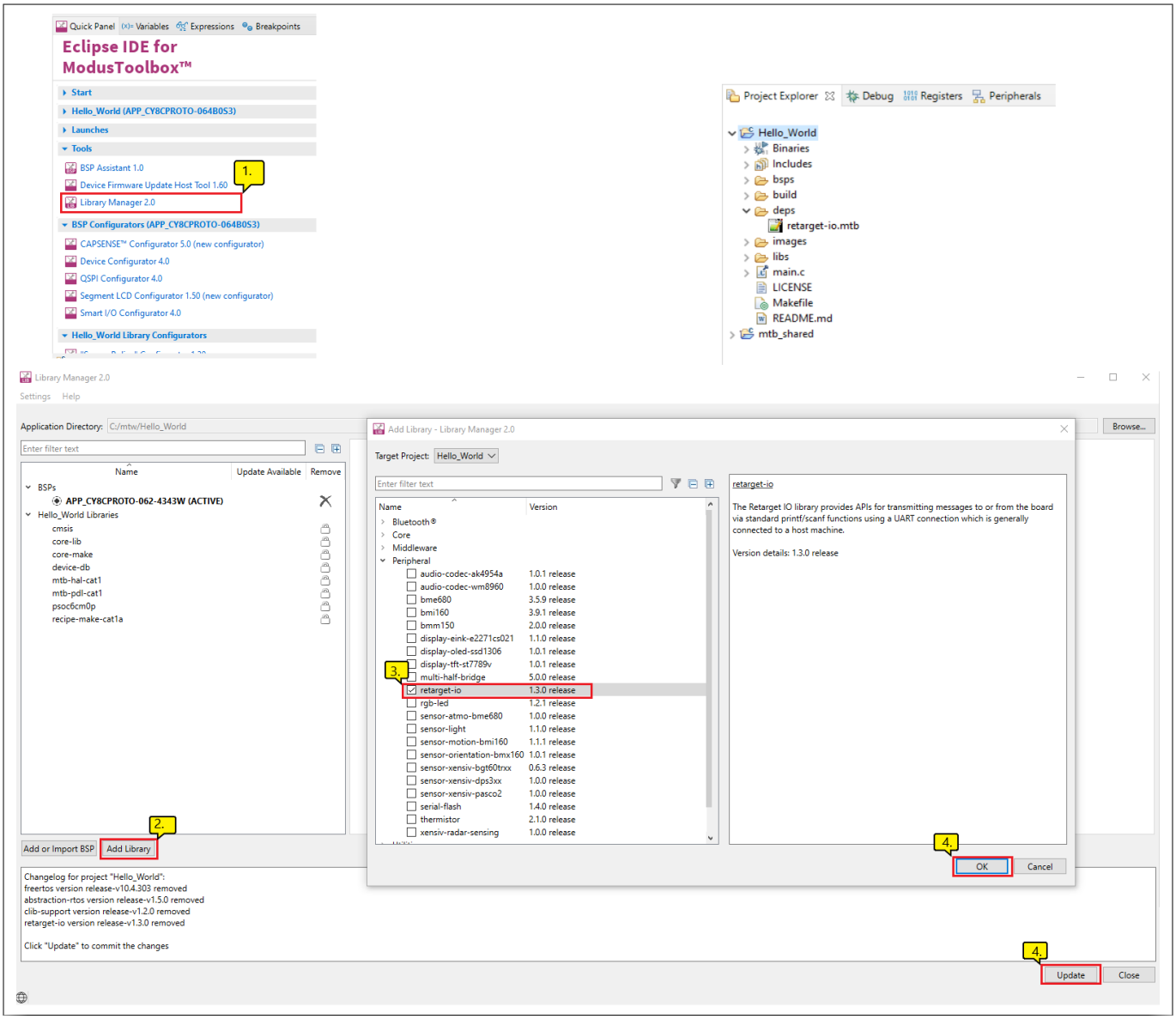



**图 17　　添加 retarget-io 中间件**

### 4.5.3　　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利用 BSP 和 HAL 提供的函数 API 在代码中直接完成。因此，不必通过器件配置器配置这些参数。请参阅第 3 部分: 写固件。

## 4.6　　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。图 18 显示了固件流程图。

CM0+内核退出复位并启用 CM4 内核。然后，由赛普拉斯提供的 CM0+应用程序将 CM0+内核配置为进入睡眠状态。此示例的资源初始化由 CM4 内核执行。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 内核后，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 内核都会切换套件上的 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 引脚在 BSP 中设置的别名，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。

*注: 你可以打开器件配置器查看 BSP 中设置的别名。*

`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 中定义的引脚别名。

`timer_init()`- 此函数包装了一组定时器 HAL 函数调用，以实例化和配置硬件定时器。它还为计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。

代码列表 1: main.c 文件

```
#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;

/* 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
********************************************************************************
* Summary:
* This is the main function for CM4 core. It sets up a timer to trigger a
* periodic interrupt. The main while loop checks for the status of a flag set
* by the interrupt and toggles an LED at 1Hz to create an LED blinky. The
* while loop also checks whether the 'Enter' key was pressed and
* stops/restarts LED blinking.
*
* Parameters:
*  none
*
* Return:
*  int
```

```
*
*******************************************************************************/
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(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("https://github.com/Infineon/"
           "Code-Examples-for-ModusToolbox-Software\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_retarget_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(CYBSP_USER_LED);
        }
    }
}


/*******************************************************************************
* Function Name: timer_init
********************************************************************************
* Summary:
* This function creates and configures a Timer object. The timer ticks
```

```
* continuously and produces a periodic interrupt on every terminal count
* event. The period is defined by the 'period' and 'compare_value' of the
* timer configuration structure 'led_blink_timer_cfg'. Without any changes,
* this application is designed to produce an interrupt every 1 second.
*
* Parameters:
*  none
*
*******************************************************************************/
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 input pin ('pin' is NC) and
     * does not use a pre-configured clock source ('clk' is NULL). */
    result = cyhal_timer_init(&led_blink_timer, 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
********************************************************************************
* Summary:
* This is the interrupt handler function for the timer interrupt.
*
* Parameters:
*    callback_arg    Arguments passed to the interrupt callback
*    event            Timer/counter interrupt triggers
*
*******************************************************************************/
static void isr_timer(void *callback_arg, cyhal_timer_event_t event)
{
    (void) callback_arg;
    (void) event;

    /* Set the interrupt flag and process it from the main while(1) loop */
    timer_interrupt_flag = true;
}
```

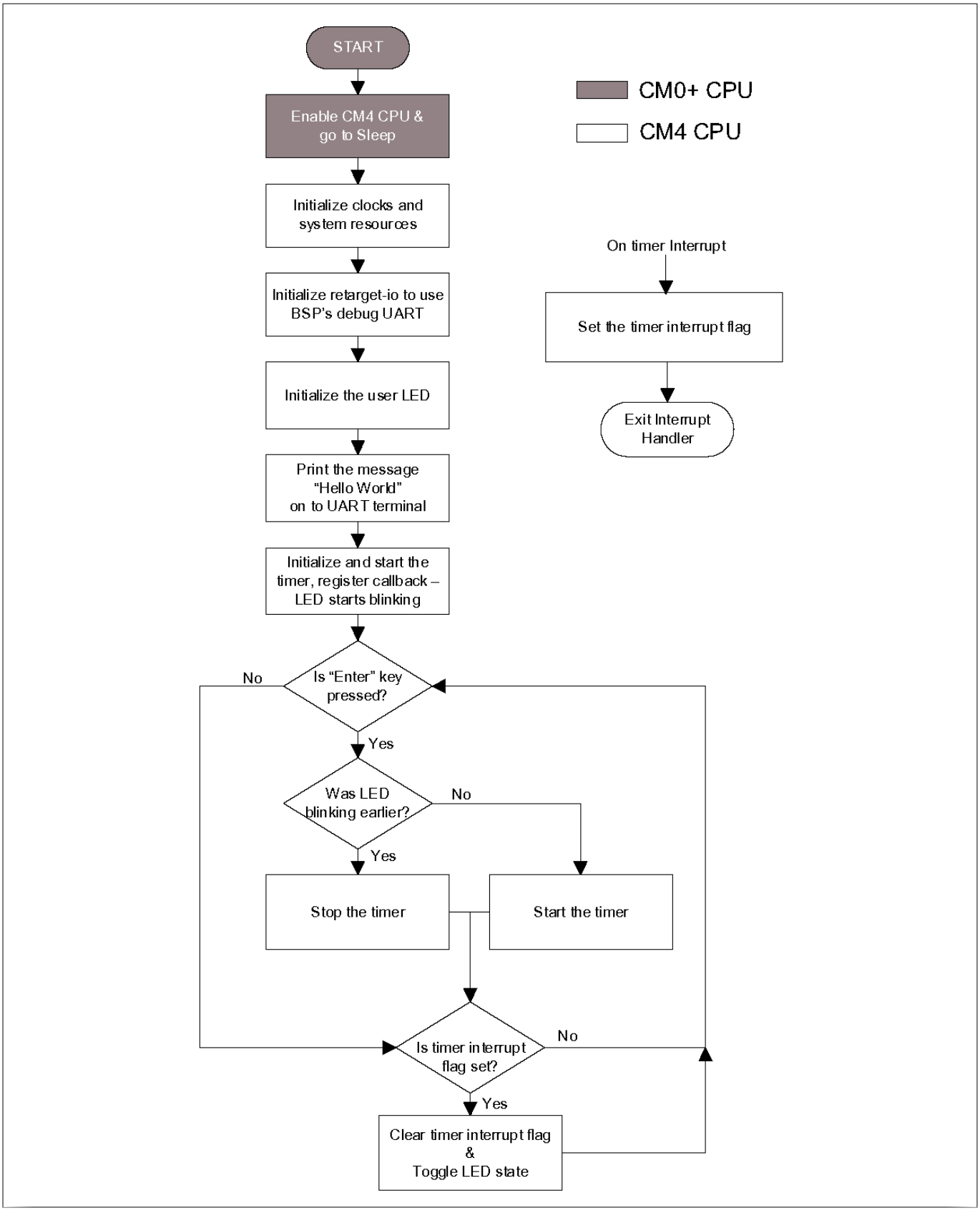



**图 18　　固件流程**

到这里就完成了代码示例中的固件工作方式的总结。请随意浏览使用源文件以获得更深入的理解。

## 4.7 第 4 部分: 编译应用程序

本节介绍如何编译应用程序。

1. 在“**Project Explorer**”窗口中选择应用程序项目
2. 单击“**快速面板**”中“**<name>**”组下的“**Build Application**”快捷方式。
   它从 Makefile 中选择构建配置，并编译/链接构成该应用程序的所有项目。默认情况下，选择的是 Debug 配置。
3. **控制台视图**列出了编译操作的结果，如图 19 所示

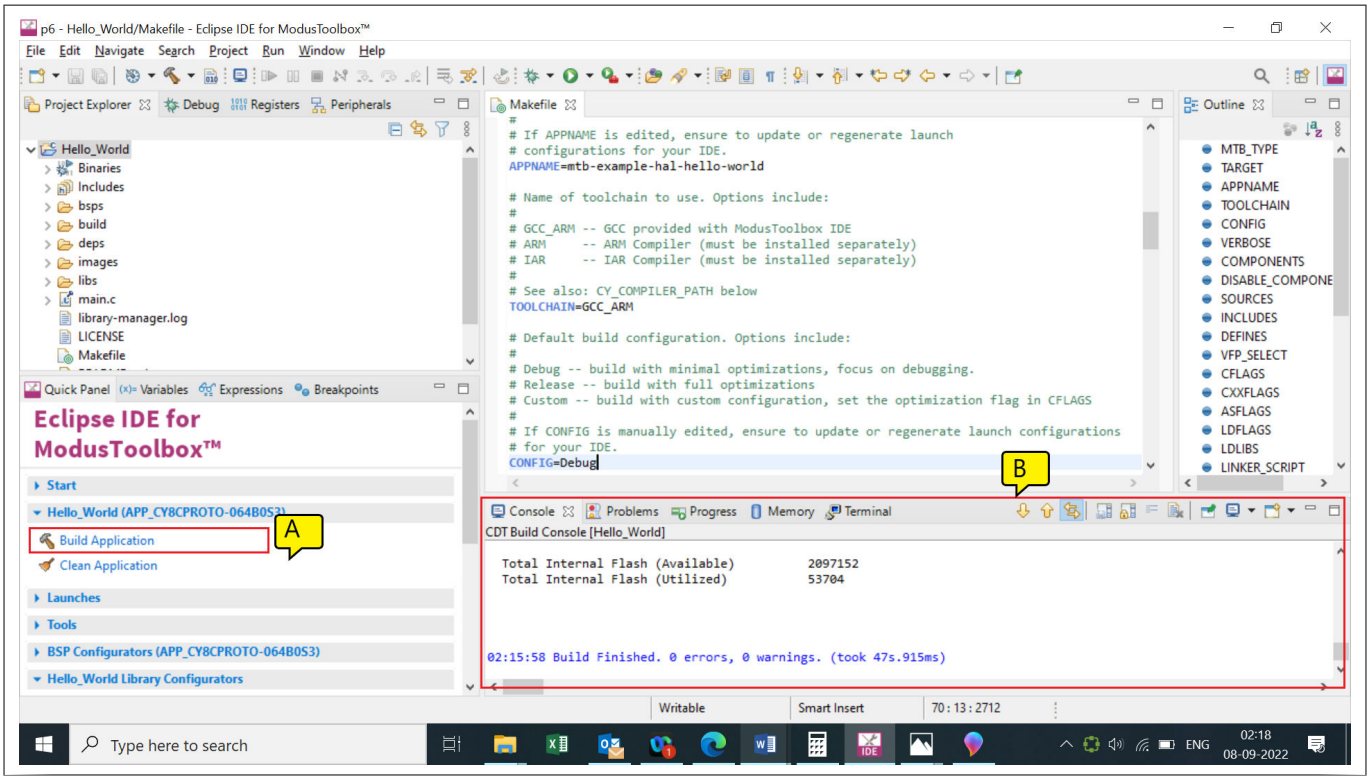



**图 19 编译应用程序**

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: 您也可以使用命令行界面(CLI)来编译应用程序。请参考 ModusToolbox™ tools package user guide 中的 **Build system** 一节。该文档位于 ModusToolbox™安装下的 /docs_<version>/文件夹。*

## 4.8 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™软件使用 OpenOCD 协议在 PSoC™ 6 MCU 器件上编程和调试应用程序。该套件必须运行 KitProg3。某些套件随机配送 KitProg2 固件而不是 KitProg3。有关详细信息，请参见使用 Eclispse IDE 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 Eclipse IDE for ModusToolbox™ user guide 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，CY8CKIT-005 MiniProg4, J-Link, 或 ULINKpro。

选择应用程序项目，点击**快速面板**中"**Launches**"组下的**<application name> Program (KitProg3_MiniProg4)**快捷方式，如图 20 所示。IDE 将选择并运行适当的运行配置。

*注: 如果自上次构建以来有任何文件被修改，该步骤也会执行一次构建。*

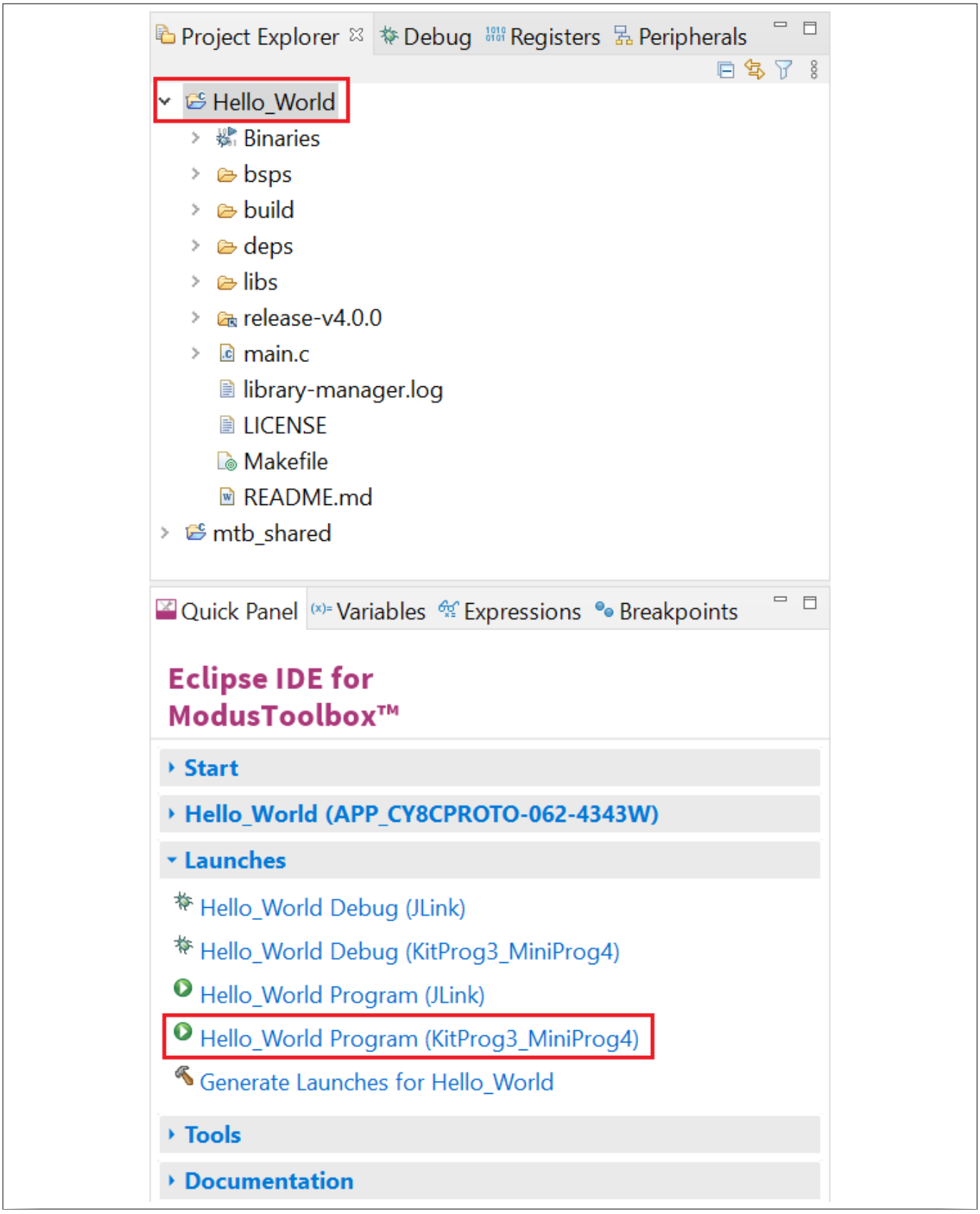



**图 20　　将应用程序编程到设备中**

Console 视图列出了编程操作的结果，如图 21 所示。

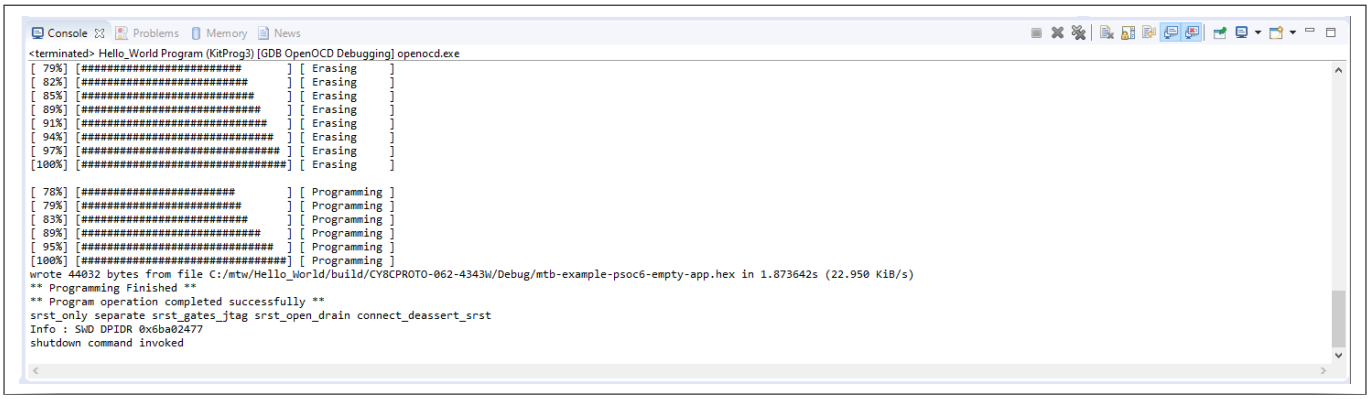



**图 21　　控制台 - 编程结果**

## 4.9　　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，如图 22 所示。注意您的 COM 端口号可能与图中不一致。

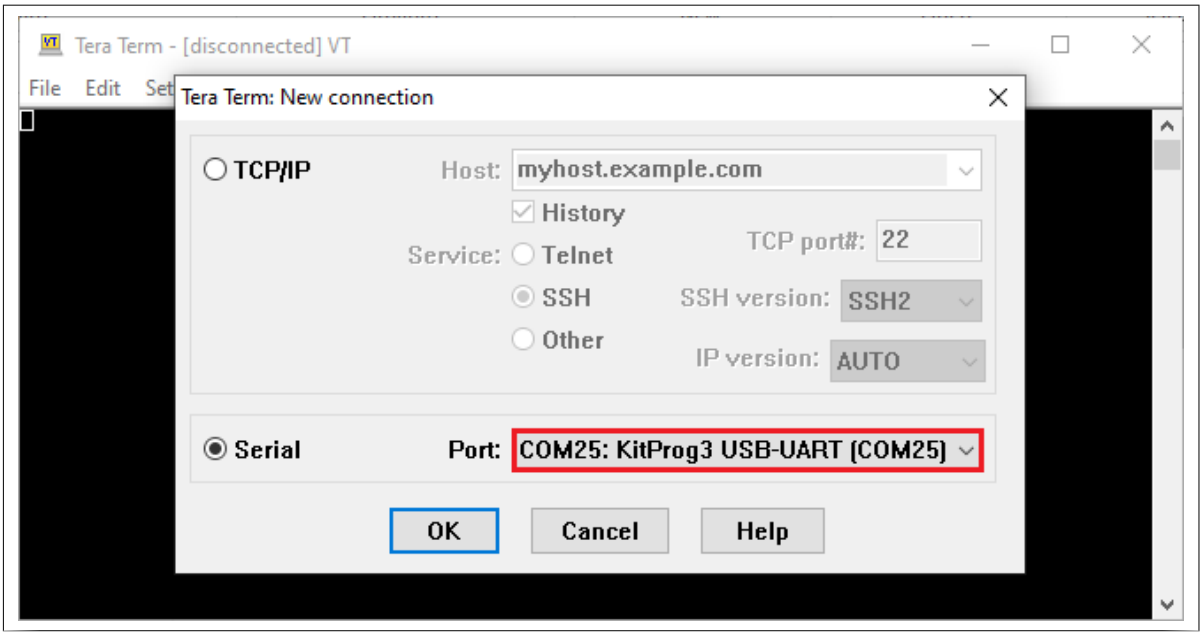



**图 22　　选择 Tera Term 中的 KitProg3 COM 端口**

**2.　设置波特率**

在 **Setup > Serial port** 路径下，设置波特率为 115200，如图 23 所示。

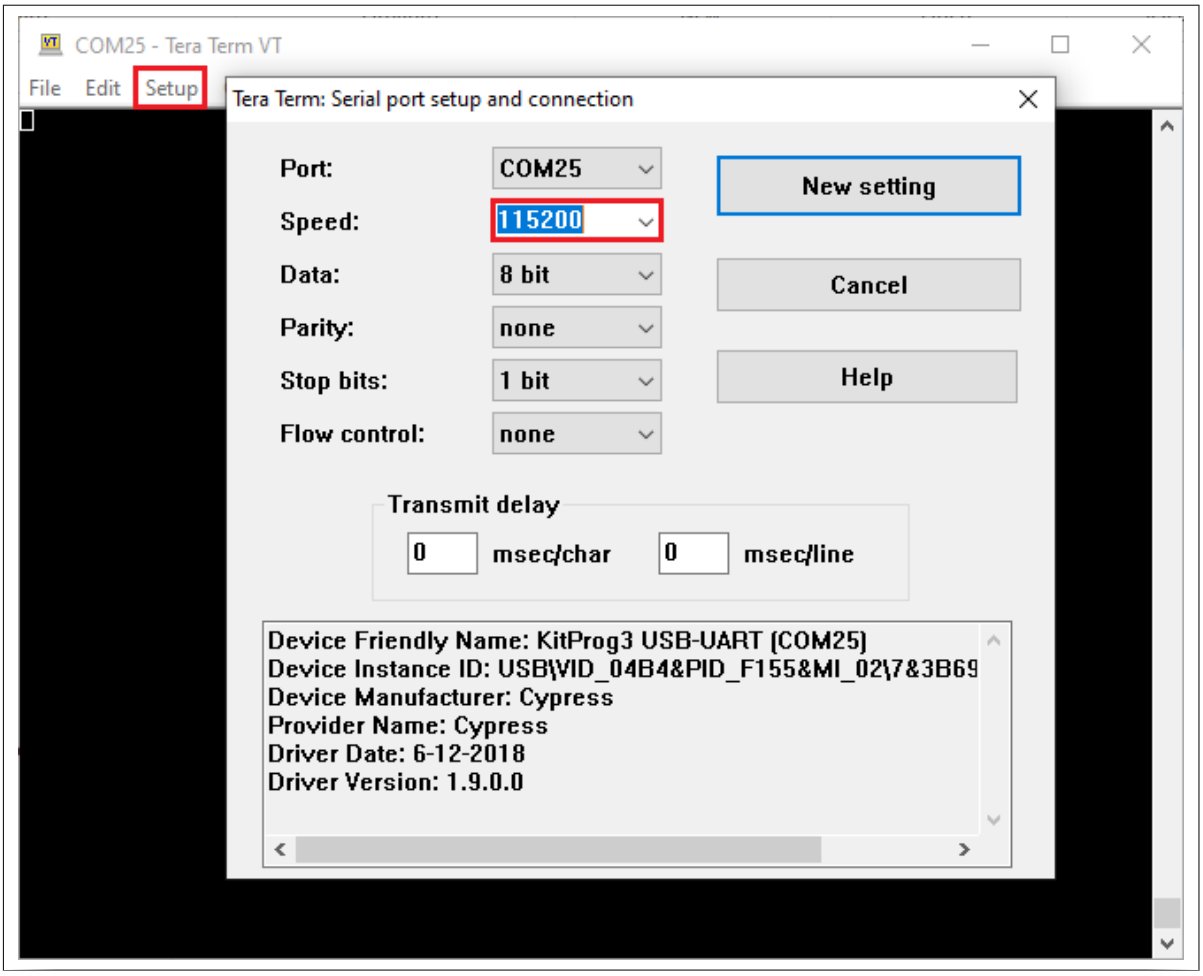



图 23 在 Tera Term 中配置波特率

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，如图 24 所示。套件上的用户 LED 将开始闪烁。

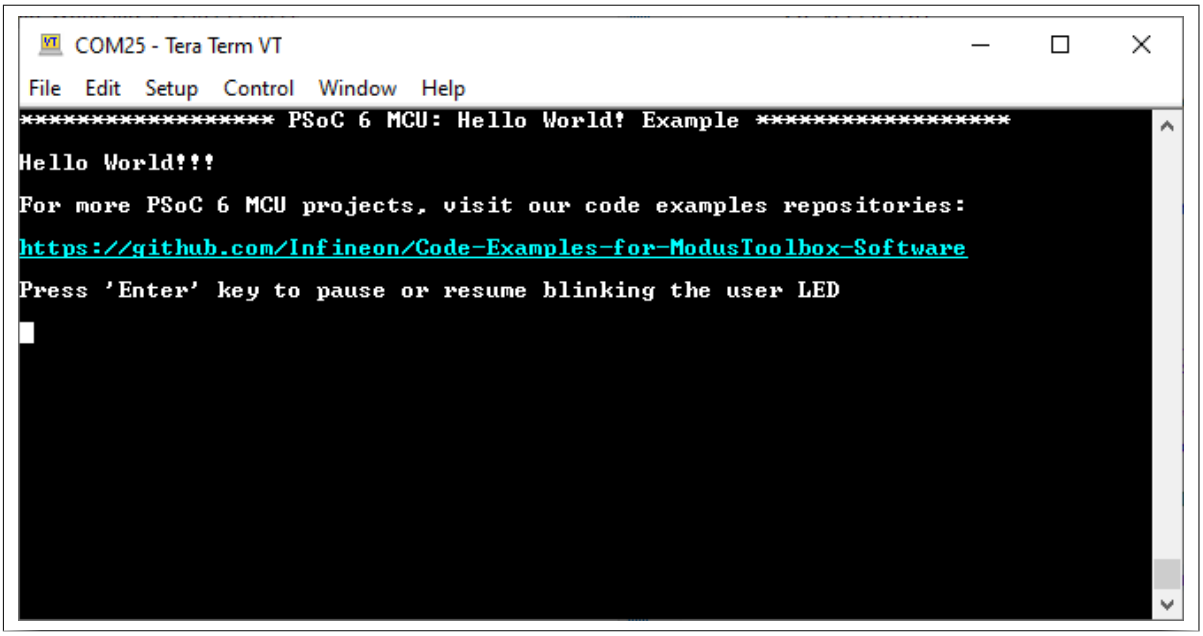



图 24 从 CM4 内核打印的 UART 消息

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，如图 25 所示。

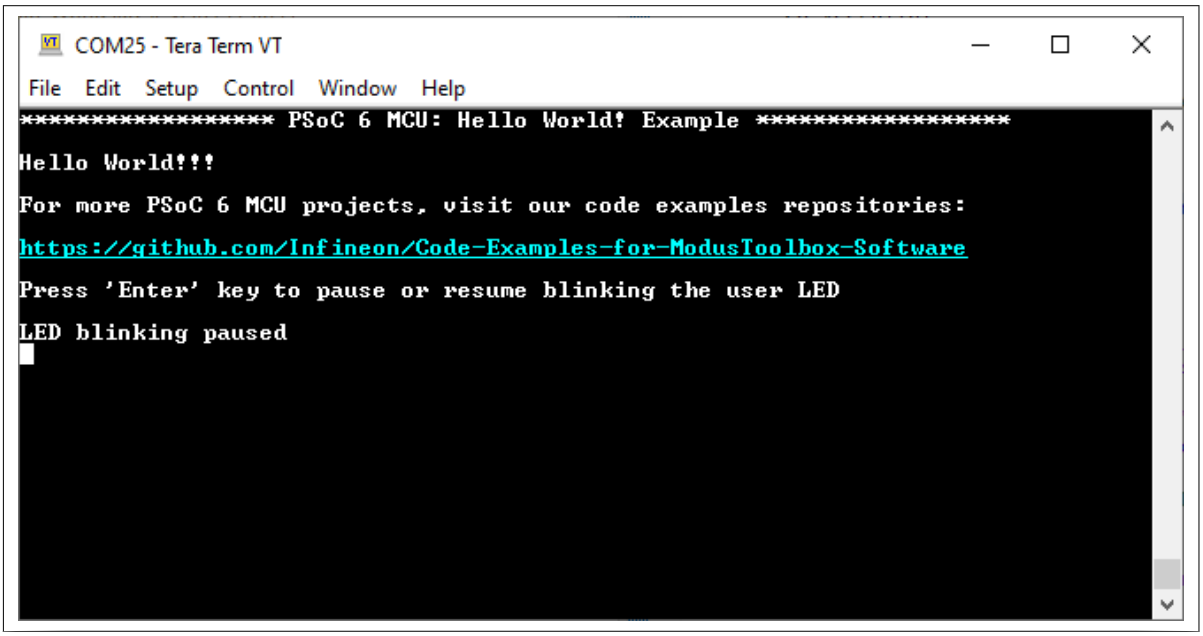



图 25 从 CM4 内核的 UART 消息

# 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，存储器和双内核系统。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。

# 参考文档

有关 PSoC™ 6 MCU 代码示例的完整和更新列表，请访问我们的 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。

表 3 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。

**表 3　　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 | 带有 PSoC™ Creator 上的蓝牙低功耗 (Bluetooth® Low Energy) 连接的 PSoC™ 6 MCU 入门 |
| AN218241 | PSoC™ 6 MCU 硬件设计注意事项 |
| AN219528 | PSoC™ 6 MCU 低功耗模式和降低功耗技术 |

表 4 列出了特定外设和应用的应用笔记 (AN)。

**表 4　　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 |
| AN235279 | 在 PSoC 6 MCU 上进行 ETM 和 ITM 跟踪 |
| **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 |
| **低功耗模拟** | |
| AN230938 | PSoC™ 6 MCU 低功耗模拟 |

# 术语表

本节列出了使用 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 MTB-PDL-CAT1)，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 (PDL): 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: 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。

# 文档修订记录

| 文档版本 | 提交日期 | 变更说明 |
|---|---|---|
| ** | 2019-12-17 | 翻译自 002-28571 Rev. ** |
| 英文版*A | - | 此版本仅以英文提交。对英文版的修订: Updated screenshots with latest release of ModusToolbox™<br>Added new supported PSoC™ 6 MCU devices<br>Added AnyCloud description under ModusToolbox™ software |
| 英文版*B | - | 此版本仅以英文提交。对英文版的修订: Added new supported PSoC™ 6 MCU device – PSoC™ 62S4<br>Added information on PSoC™ 6 product lines and development kits available for each product line |
| *A | 2021-03-18 | 翻译自 002-28571 Rev. *C。对英文版的修订: Updated 图 1<br>Updated Screenshots with MTB v2.2<br>Added mtb_shared folder description, updated application creation process with MTB flow |
| 英文版*D | - | 此版本仅以英文提交。对英文版的修订: Updated to Infineon template |
| 英文版*E | - | 此版本仅以英文提交。对英文版的修订: Updated the GitHub links<br>Added reference to new PSoC™ 6 MCU low-power analog<br>Updated 图 22 to 图 25<br>Firmware updated to the latest version |
| 英文版*F | - | 此版本仅以英文提交。对英文版的修订: Template update |
| *B | 2023-02-27 | 翻译自 002-28571 Rev. *G。对英文版的修订: Updated the development flow as per MTB v3.0 software<br>Updated link references<br>Updated configurator info<br>Added new Figure 2 and Figure 7<br>Updated Figure 8 and Figure 14<br>Added reference to new AN235279<br>Added a new section for ModusToolbox™应用<br>Removed reference to deprecated AN225588<br>Updated Figure 17 to Figure 19 |

**Trademarks**

All referenced product or service names and trademarks are the property of their respective owners.

**Edition 2023-02-27**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**

**© 2023 Infineon Technologies AG**

**All Rights Reserved.**

**Do you have a question about any aspect of this document?**

**Email: erratum@infineon.com**

**Document reference**

**IFX-ofg1649405242329**

**重要提示**

本应用说明中包含的信息仅作为产品实施的提示， 在任何情况下均不应视为对产品特定功能、产品状 况或产品质量的描述或保证。在产品实施之前，本 应用说明的接收方必须在实际应用中验证此处给出 的所有功能和其他技术信息。对于本应用说明中提 供的所有信息，英飞凌科技公司不承担任何形式的 任何保证和责任（包括但不限于不侵犯任何第三方 知识产权的保证）。

本文档所含的数据仅供经过专业技术培训的人员使用。客户自身的技术部门有义务对于产品是否适宜 于其预期的应用和针对该等应用而言本文档中所提 供的信息是否充分自行予以评估。

**警告事项**

由于技术所需产品可能含有危险物质。如需了解该 等物质的类型，请向离您最近的英飞凌科技办公室 接洽。

除非由经英飞凌科技授权代表签署的书面文件中做 出另行明确批准的情况外，英飞凌科技的产品不应 当被用于任何一项一旦产品失效或者产品使用的后 当被用于任何一项一旦产品失效或者产品使用的后 领域。