

## 创建自定义的 BLE 配置文件

作者: Rohit Kumar

相关器件系列: CY8C4XXX-BL、CYBL10XXX

相关应用笔记: 要想获取完整列表, 请点击[此处](#)。

如果需要获取此应用笔记的最新版本或相关项目文件, 请访问 <http://www.cypress.com/AN91162>。

更多代码示例? 我们明白。

如需寻找包含上百 PSoC 代码示例并有不断更新的网上资源, 请浏览我们的[代码示例网页](#)。  
您还可以在[此处](#)观看 PSoC 视频库。

AN91162 说明了通过使用自定义低功耗蓝牙 (Bluetooth® Low Energy — BLE) 配置文件开发 PSoC 4 BLE 或 PSoC BLE 器件上的 BLE 应用的方法。它简单介绍了自定义配置文件和服务, 并说明了通过使用 RGB LED 控制来构建 PSoC 4 BLE 示例应用的流程。本应用笔记也适用于 PSoC BLE 器件。

## 内容

1 简介 .....	1	5.4 RGB LED 控制 .....	19
2 PSoC 资源 .....	2	5.5 配置项目的设计范围内的资源 .....	22
2.1 PSoC Creator .....	3	5.6 构建项目 .....	24
2.2 PSoC Creator 帮助 .....	3	5.7 将源文件/头文件添加到项目中 .....	24
2.3 代码示例 .....	4	5.8 项目文件 .....	25
3 标准服务与自定义服务 .....	5	5.9 配置固件 .....	25
4 定义一个自定义的 BLE 配置文件 .....	5	5.10 编译和编程 .....	31
4.1 定义服务 .....	5	5.11 使用 CySmart 手机应用进行测试 .....	33
4.2 定义特性 .....	5	5.12 使用 CySmart 中央仿真工具进行测试 .....	34
4.3 定义描述符 .....	6	6 总结 .....	38
4.4 生成自定义 UUID .....	6	7 相关信息 .....	38
5 PSoC Creator 项目: RGB LED 的自定义配置文件 .....	7	A 附录 .....	39
5.1 创建 PSoC Creator 项目 .....	9	A.1 发送通知 .....	39
5.2 配置组件 .....	10		
5.3 配置 BLE 外设 .....	16		

## 1 简介

低功耗蓝牙 (BLE) 是由蓝牙技术联盟 (SIG) 介绍的一个超低功耗无线标准, 使用于短距离通信。通过设计和优化 BLE 物理层、协议栈以及配置文件架构, 可以尽量降低功耗。同传统蓝牙相似, BLE 的工作 ISM 频段为 2.4 GHz, 但其带宽较窄 (1 Mbps)。

赛普拉斯的 PSoC 4 BLE 是可编程嵌入式片上系统 (SoC), 在单芯片中集成了 BLE、可编程模拟和数字外设功能、存储器以及 ARM® Cortex®-M0 微控制器。

本应用笔记介绍了如何使用 BLE 组件的图形用户界面 (GUI) 轻松创建一个自定义 BLE 配置文件。您将定义自定义配置文件的结构。工具将自动生成 API 和需要使用的事件代码。可以执行各相似的步骤, 以发送或接收数据类型和数据长度 (根据您的自定义配置文件的要求进行)。然后, 您会使用赛普拉斯的 [CY8CKIT-042-BLE Pioneer 套件](#) 测试自定义配置文件。

本应用笔记假设您已经基本了解了 BLE 架构和相关术语。

- 如果您还不熟悉 BLE 或 PSoC，请参考应用笔记 [AN91267 — PSoC 4 BLE 入门](#)。
- 欲了解 PSoC Creator 环境中的 BLE 组件架构以及如何使用 BLE 服务来开发各种应用的信息，请参考应用笔记 [AN91184 — PSoC 4 BLE 设计各种 BLE 应用](#)。
- 有关 BLE 规范的完整信息，请访问 [BLE 开发者门户网站](#)。

请安装 [套件网站](#) 上最新版本的 BLE Pioneer 套件软件，它提供了 BLE 应用开发与调试的相关工具。CY8CKIT-042-BLE 或 BLE Pioneer 套件是赛普拉斯的 BLE 开发套件，它支持 PSoC 4 BLE 和 PProC BLE 系列器件。该套件还包括连接到一个 Pioneer 基板的可插入 PSoC 4 BLE（和 PProC BLE）模块。该套件用于演示本应用笔记所提供的示例项目。该套件包括一组 BLE 示例项目和可帮助您开始开发自己的 BLE 应用的文档。

## 2 PSoC 资源

赛普拉斯网站 [www.cypress.com](http://www.cypress.com) 上提供了大量资料，有助于您正确选择 PSoC 器件，并能够快速和有效地将器件集成到您的设计中。有关资源的完整列表，请参考 [KBA86521 — 如何使用 PSoC 3、PSoC 4 和 PSoC 5LP 进行设计](#)。下面是 PSoC 4 BLE 的简要列表：

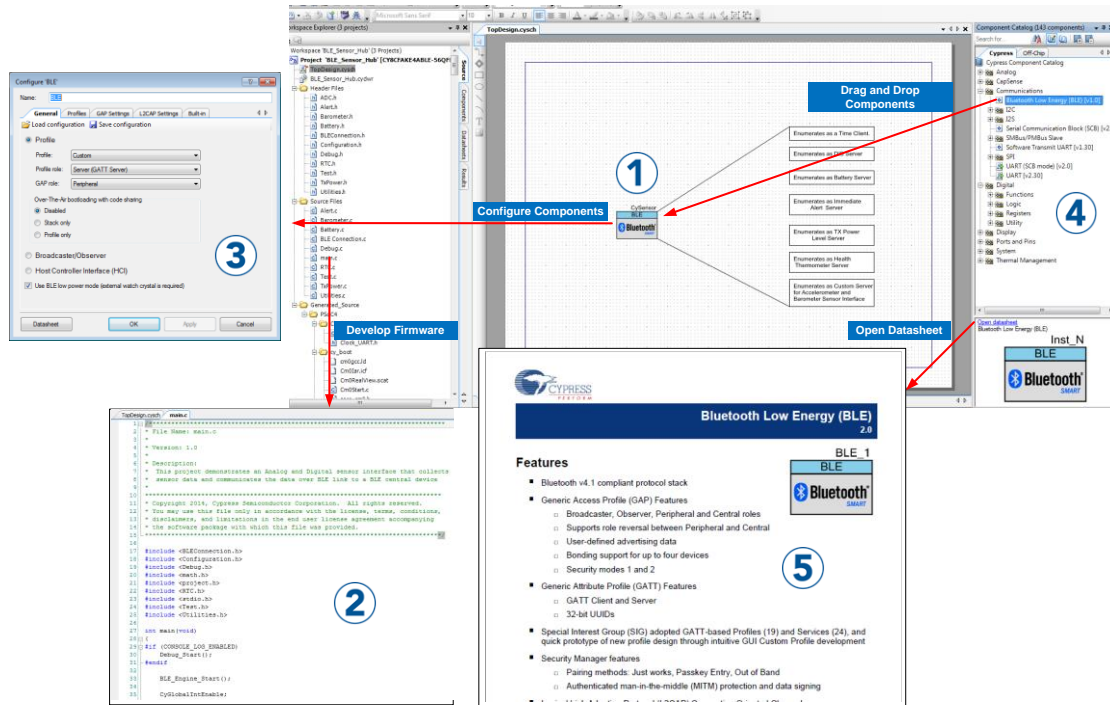
- **概况：** [PSoC 产品系列](#)、[PSoC 产品路线图](#)
- **产品选型：** [PSoC 1](#)、[PSoC 3](#)、[PSoC 4](#) 或 [PSoC 5LP](#)。此外，[PSoC Creator](#) 还包含了一个器件选择工具。
- **数据手册：** 介绍了 [PSoC 41XX-BL](#) 和 [PSoC 42XX-BL](#) 器件系列，并且提供了它们的相关电气规范。
- **应用笔记和代码示例：** 包括从基本到高级的广泛主题。许多应用笔记包括了代码示例。[PSoC Creator](#) 提供了额外的代码示例，请参考 [代码示例](#)。  
此外，您可以在 [PSoC 3/4/5 代码示例](#) 网页上查找 PSoC 器件的代码示例以及各相关套件。对于 BLE 器件，可滚动到 [CY8CKIT-042-BLE Pioneer](#) 套件的表格。
- **技术参考手册（TRM）：** 详细说明了每个 PSoC 4 BLE 器件系列中的架构和寄存器。
- **CapSense 设计指南：** 了解如何在 PSoC 4 BLE 器件系列中设计电容式触摸感应应用。
- **开发工具**
  - [CY8CKIT-042-BLE 低功耗蓝牙（BLE）Pioneer](#) 套件是一款针对 BLE 器件系列的易用且价廉的开发平台。该套件包括用于 [Arduino™](#) 兼容子卡和 [Digilent® Pmod™](#) 子卡的连接器。
  - 适用于 [Windows](#)、[iOS](#) 和 [Android](#) 版本的 [CySmart BLE](#) 主机仿真工具是一种易于使用的 GUI，通过它您可以进行测试和调试 BLE 外设应用。
- **技术支持**
  - [常见问题（FAQ）](#)：加深了解我们的 BLE 生态系统
  - [BLE 论坛](#)：查看您在 [PSoC 4 BLE](#) 和 [PProC BLE](#) 论坛上咨询的内容是否得到了各开发伙伴方的回复。
  - 赛普拉斯支持：仍未解决？请访问我们的 [支持页面](#)，并创建一个 [技术支持案例](#) 或联系 [本地销售代表](#)。如果您在美国，可以通过拨打我们的免费电话，直接与技术支持团队取得联系：+1-800-541-4736。提示音后选择 2。

## 2.1 PSoC Creator

**PSoC Creator** 是一个基于 Windows 的免费集成设计环境 (IDE)。通过它可以在 PSoC 4 BLE 或 PSoC 4 BLE 器件中同时设计硬件和固件系统。如图 1 所示, 通过 PSoC Creator, 您可以进行以下操作:

1. 将组件图标拖放到主设计工作区中, 以进行您的硬件系统设计。
2. 协作设计您的应用固件和 PSoC 硬件。
3. 使用配置工具来配置各组件。
4. 浏览包含 100 多个组件的库。
5. 查看组件数据手册。

图 1. PSoC Creator 的原理图输入项和组件



## 2.2 PSoC Creator 帮助

请访问 [PSoC Creator](#) 主页, 以下载并安装 PSoC Creator 的最新版本。然后, 启动 PSoC Creator, 并导航到下列各项:

- **快速入门指南:** 依次选择 **Help > Documentation > Quick Start Guide**。本指南提供了开发 PSoC Creator 项目的基本知识。
- **简单的组件示例项目:** 依次选择 **File > Open > Example projects**。这些示例项目展示了如何配置及使用 PSoC Creator 组件。
- **入门设计:** 依次选择 **File > New > Project > PSoC 4 Starter Designs**。这些入门设计展示了 PSoC 4 BLE 的特定性能。
- **系统参考指南:** 依次选择 **Help > System Reference > System Reference Guide**。该指南列出并描述了 PSoC Creator 提供的系统功能。
- **组件数据手册:** 右键单击组件, 然后选择“Open Datasheet”项。请访问 [PSoC 4 BLE 组件的数据手册](#) 网页, 以获取 PSoC 4 BLE 组件的所有数据手册列表。
- **文档管理工具:** PSoC Creator 提供了一款文档管理工具, 便于寻找和查看文档资源。要想打开文档管理工具, 请依次选择菜单项: **Help > Document Manager**。

## 2.3 代码示例

PSoC Creator 包含了多个代码示例项目。可以从 PSoC Creator 的起始页上获取这些项目，如图 2 所示。

这些示例项目通过为您提供完整的设计（并非一个空白页），可以加快您的设计过程。示例项目还介绍了如何将 PSoC Creator 组件使用于不同应用中。此外，它还包含了多个代码示例和数据手册，如图 3 所示。

在图 3 所示的 **Find Example Project**（查找示例项目）对话框中，您可以选择以下选项：

- 根据 **architecture**（架构）或 **device family**（器件系列）（例如：PSoC 4、PSoC 4 BLE、PSoC 4 BLE 等）、**category**（类型）或 **keyword**（关键词）等选项筛选示例。
- 从 **Filter Options**（滤波选项）的示例菜单中进行选择。您可以从拥有 20 多个 BLE 示例项目的库开始，如图 3 所示。
- 通过 **Documentation**（文档）选项卡，查看选中的数据手册。
- 查看已选的代码示例。您可以复制该窗口中的代码，然后将其粘贴在您的项目内，从而加快代码开发过程；
- 或根据所选项目创建一个新的项目（需要时可添加新的工作区）。向您提供了一个完整的基本设计，该方式可以加快您的设计进程。然后，您可以根据自己的应用要求来调整该设计。

图 2. PSoC Creator 中的代码示例

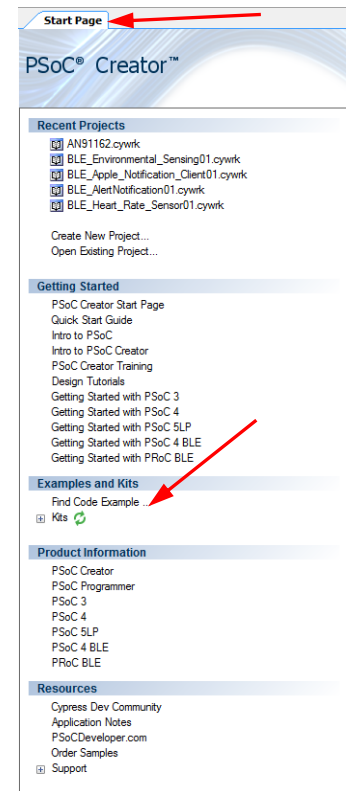
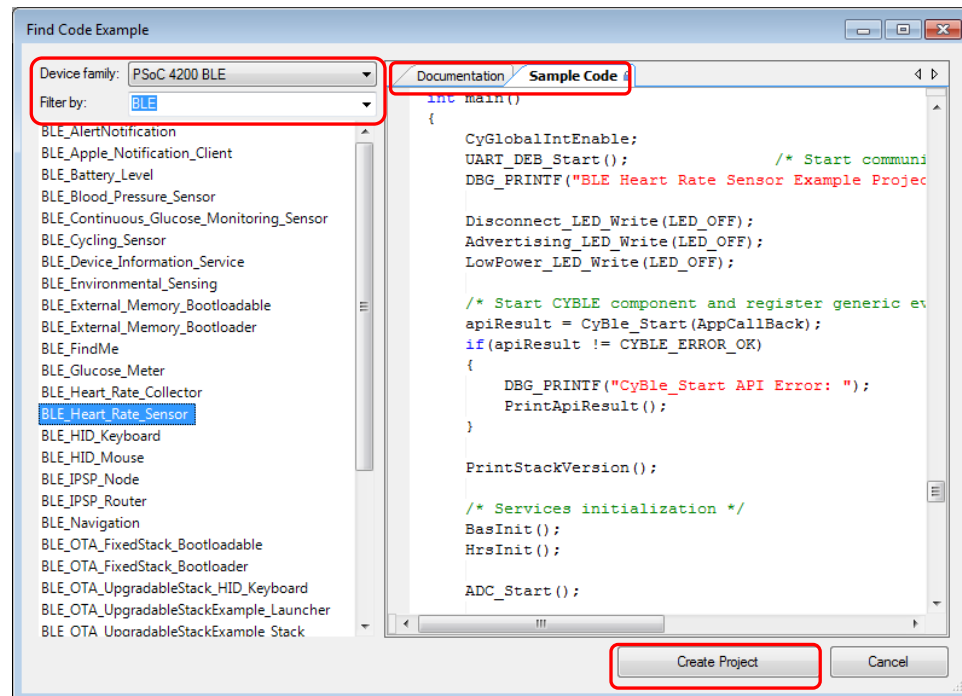


图 3. 带样本代码的代码示例项目



### 3 标准服务与自定义服务

服务是一组特性，用于定义某个具体的函数。它分为以下两种。一种是标准服务，它由蓝牙技术联盟（SIG）定义，并用于 BLE 器件的一些通用应用。Heart Rate（心率）、Health Thermometer（健康体温计）、Blood Pressure（血压）以及 Alert Notifications（警报通知）等是它的一些实例应用。有关标准服务的完整列表，请参考[蓝牙开发者门户网站](#)。请参见应用笔记 [AN91184 — PSoC 4 BLE 设计各种 BLE 应用](#)，了解如何通过使用 PSoC 4 BLE 来设计一个标准应用。

另一种是自定义服务。对于这种类型的服务，顾名思义是指自定义应用程序，但未得到普遍认可。通过这些服务，您可以配置 BLE 器件，这些 BLE 器件具有在 BLE SIG 定义的一组限制服务范围外，但仍然使用了 BLE 框架的自定义应用。在开发 BLE 应用过程中，任何人都可以制定自定义服务。本应用笔记中的示例项目将展示以下内容：通过各个自定义服务，您可以在 BLE Pioneer 套件与具有 BLE 功能的手机或 PC 之间发送自定义 RGB LED 数据。

### 4 定义一个自定义的 BLE 配置文件

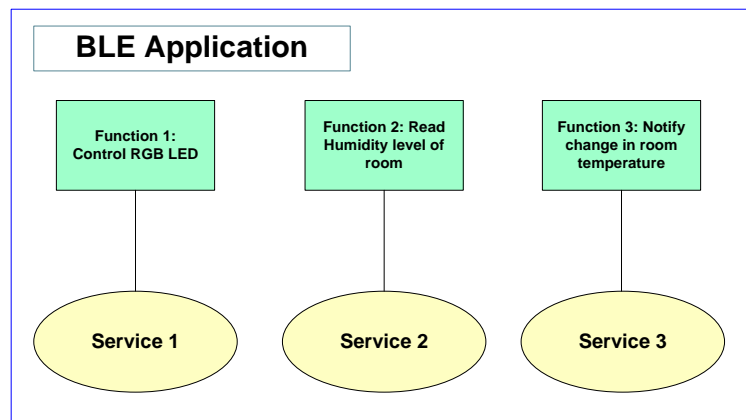
自定义的 BLE 配置文件包含了自定义服务及其特性。另外，它还包含标准服务及其特性。

#### 4.1 定义服务

在创建自定义 BLE 应用中，首先需要分析应用所需的各个函数组。每个函数均由一种自定义服务表示，然后使用它来获取需要的所有数据。

例如，一个函数可以控制某个 RGB LED 的红、绿、蓝等颜色的亮度。该函数可以由一个命名为“RGB LED Control”（RGB LED 控制）的自定义服务表示。其他函数可以读取室内湿度或室内温度。图 4 显示的是一个应用实例，它定义了用于执行三个函数的三种自定义服务。

图 4. 定义自定义服务



仅与所提供的数值类型不同的函数可以被归入到一个服务中。在 RGB LED 控制示例中，不需要创建四个不同的自定义服务来控制四个 RGB LED 颜色值（红色、绿色、蓝色和亮度）。由于该函数用于控制 RGB LED 的值，因此只要使用一个服务。定义好服务后，将唯一一个通用识别码（UUID）分配给唯一能识别它们的服务。在自定义服务中，这些 UUID 应为 128 位的值。

#### 4.2 定义特性

接下来，需要定义每个服务的特性。该定义包含了以下内容：

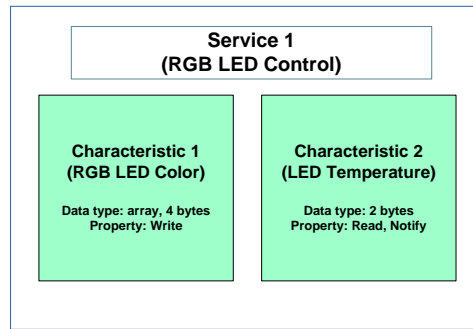
- 数据值：数据值说明了所传输数据的类型和长度。支持的数据类型包括无符号字节、有符号字节、字、字符串以及数组。
- 属性：属性说明了如何访问数据值。可用的选项为：Broadcast、Read、Write、WriteWithoutResponse、Notify、Indicate、SignedWrite 以及 WritableAuxiliaries。
- 权限：权限说明了数据的访问权限。可将“权限”设置为 Encryption、Authentication 和 Authorization 三种类型。
- UUID：UUID 值（128 位）唯一地识别特性。



在 RGB LED 控制示例中，已定义的特性发送了四字节的阵列，其中一个字节对 RGB LED 的每个颜色值进行定义，另外一个用于控制颜色亮度。特性的定义取决于应用程序解释数据的方式。因为 GATT 客户端将新的 RGB LED 值写入到 GATT 服务器中，所以该特性的属性是“Write”（写入）。

类似的，您可以添加其他特性，该特性将提供从监控 LED 过热的板上热传感器的 2 字节的温度信息。图 5 提供了上述特性的概况。

图 5. 定义服务中的特性

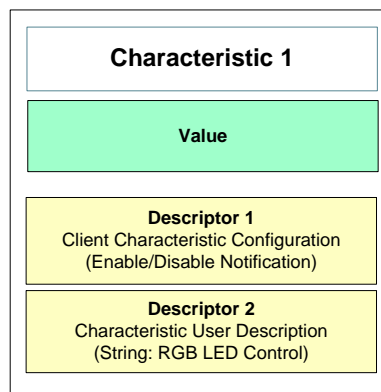


### 4.3 定义描述符

根据特性，您可以添加描述符。这些描述符为用户提供了特性的信息。GATT 客户端器件可以使用这些描述符来使能或禁用通知和指示。

图 6 显示的是自定义特性中的一个描述符示例。在该示例中，GATT 客户端使用了被称为客户端特性配置的描述符，用于使能和禁用通知或指示。特性下面的描述符提供了通知或指示选项。另一个描述符示例是特性的用户描述，它提供了一个人类可读格式来识别特性的字符串。

图 6. 定义各种特性中的描述符



### 4.4 生成自定义 UUID

所有 BLE 自定义服务和特性必须使用 128 位 UUID 来识别，并且要确保基本 UUID 与 BLE 定义的基本 UUID（00000000-0000-1000-8000-00805F9B34FB）不一样。基本 UUID 是一个 128 位的数值，根据该值可定义标准 UUID（16 位和 32 位）。

BLE 规范未定义生成 BLE 服务和特性的自定义 UUID 的方式。用户会决定如何生成自己的 128 位 UUID，该 UUID 与 BLE 定义的基本 UUID 不同。可以采用多种方法生成自定义服务和特性的 UUID。

赛普拉斯使用以下机制来生成自定义服务和特性的 UUID。您也可以使用类似的方法创建自己的 UUID。

自定义 UUID 值: **XXXXYYYY-0000-1000-8000-00805F9B0131**

表 1. 赛普拉斯根据 BLE 定义的基本 UUID 生成 128 位 UUID 的方法

UUID 器件	说明
XXXX	用于识别器件/产品的 16 位数值
YYYY	特定服务或特性的 16 位 UUID
00805F9B0131	所有赛普拉斯的自定义服务和特性的基本 UUID 这是由 BLE SIG 定义的基本 UUID 的最后 6 个字节，其中最后两个字节由赛普拉斯的蓝牙分配公司标识符（0x0131）替换。

PSoC 4 BLE 的 XXXX 将被设置为 0x0003。

例如，在该项目中，我们将采用 RGB LED 自定义服务。其 128 位自定义 UUID 被设为 **0003CBBB-0000-1000-8000-00805F9B0131**。此处，器件标识符被设为 0x0003，基本 UUID 的最后 6 个字节被设为 0x00805F9B0131，RGB LED 服务特定的 16 位 UUID 则被设定为 0xCBBB。

另外，您可以参考 <http://www.itu.int/en/ITU-T/asn1/Pages/UUID/uuids.aspx> 网页，了解如何生成唯一的 128 位 UUID。

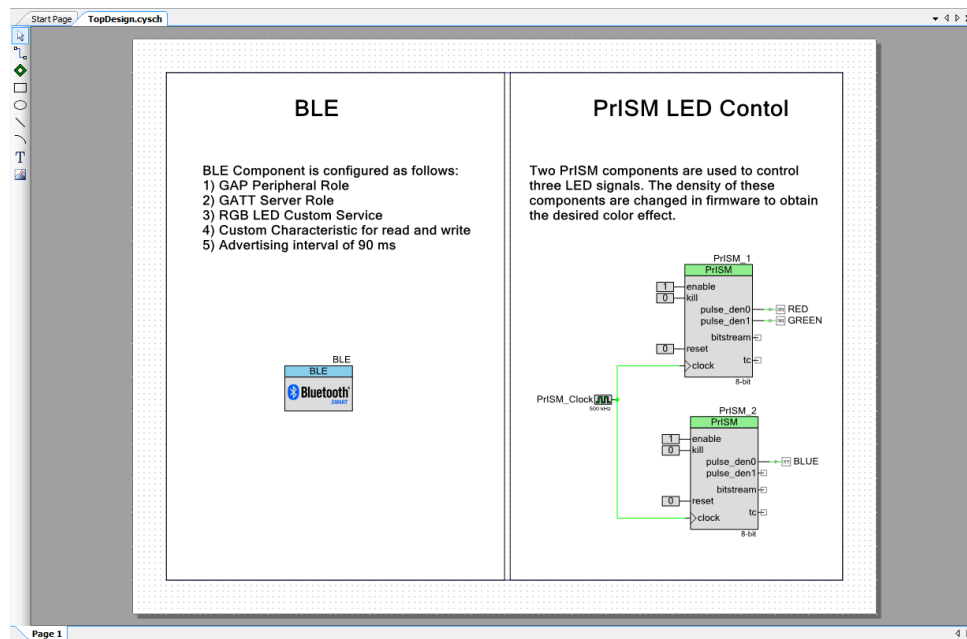
## 5 PSoC Creator 项目：RGB LED 的自定义配置文件

为了创建和验证该项目，请确保您已经具备了以下先决条件：

- PSoC Creator 3.3 SPI（或更高版本）以及 PSoC Programmer 3.24（或更高版本）
- CySmart™ 中央仿真工具
- CySmart iOS 应用或 CySmart Android 应用
- CY8CKIT-042-BLE Pioneer 套件

该项目使用了 BLE、PrISM™、时钟以及数字输出引脚等 PSoC Creator 组件。图 7 显示的是 PSoC Creator 项目原理图。

图 7. PSoC Creator 项目原理图



要想实现该项目，请执行以下步骤：

1. 创建一个 PSoC Creator 项目。
2. 配置 PSoC Creator 中的各个组件。
3. 对固件进行写操作，以处理 BLE 事件及其他组件。
4. 构建项目并编程 BLE Pioneer 套件。
5. 通过使用 CySmart 工具或应用程序，测试该项目。

该示例项目包括 RGB LED 控制的自定义服务，它用于控制 BLE Pioneer 套件板上 RGB LED 的颜色和亮度。

对于 RGB LED 控制，您可以使用 uint8 类型的四字节阵列来定义数据格式，如图 8 所示。该服务支持写入和读取操作。

图 8. RGB LED 的数据格式





## 5.1 创建 PSoC Creator 项目

- 依次点击 **Start > All Programs > Cypress > PSoC Creator 3.3 > PSoC Creator 3.3** 来打开 PSoC Creator。
- 创建一个新项目（依次点击 **File > New > Project**）。选择 PSoC 4100 BLE / PSoC 4200 BLE Design 模板，然后选择 CY8C4247LQI-BL483 器件。将该项目被命名为 AN91162，并将工作区保存在所需位置。

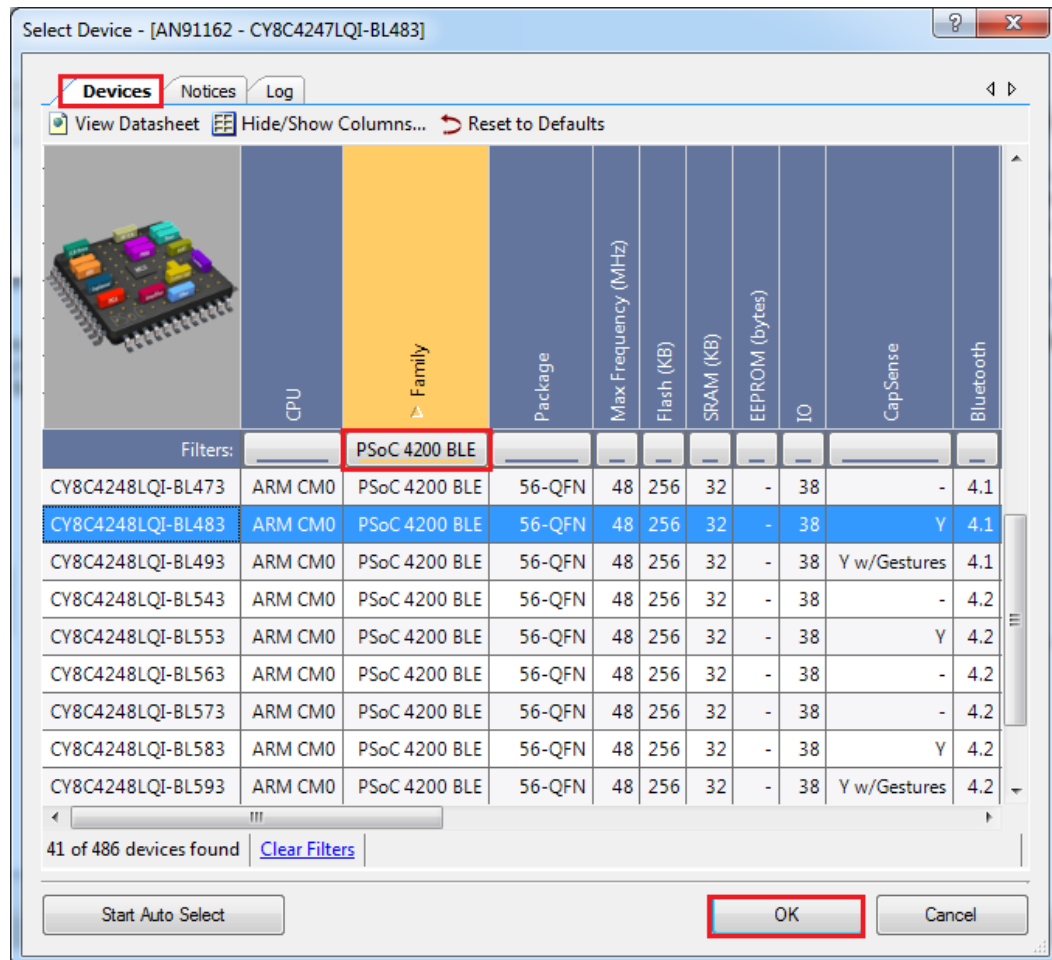
**注意：** CY8C4XX7-BL 器件具有 128 K 的闪存和 16 K 的 SRAM。如果使用具有 256K 闪存和 32K SRAM 的 PSoC 4 BLE 器件，请选择 CY8C4XX8-BL 器件。BLE 组件 2.3 支持这些器件。

CY8C4248LQI-BL583 是一个具有 256K 闪存和 23K SRAM 的器件，支持 BLE 4.2。如果您拥有 BLE 4.2 功能器件，并将 BLE 组件更新为 3.0 或更高版本，那么可以将该项目更改为器件选择器设置中的该器件编号。

若想更改各种 PSoC 4 BLE 器件间的器件编号，需要执行以下步骤：

- 在 **Workspace Explorer** 中，右键点击项目名称。
- 选择 **Device Selector...**。
- 然后将 **Family** 项设定为 **PSoC 4200 BLE**。
- 从列表中选择您的器件编号，点击 **OK**（请参考图 9）。

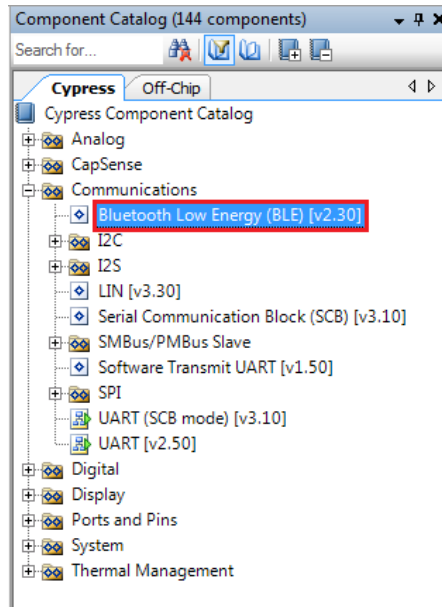
图 9. 器件选择器设置



## 5.2 配置组件

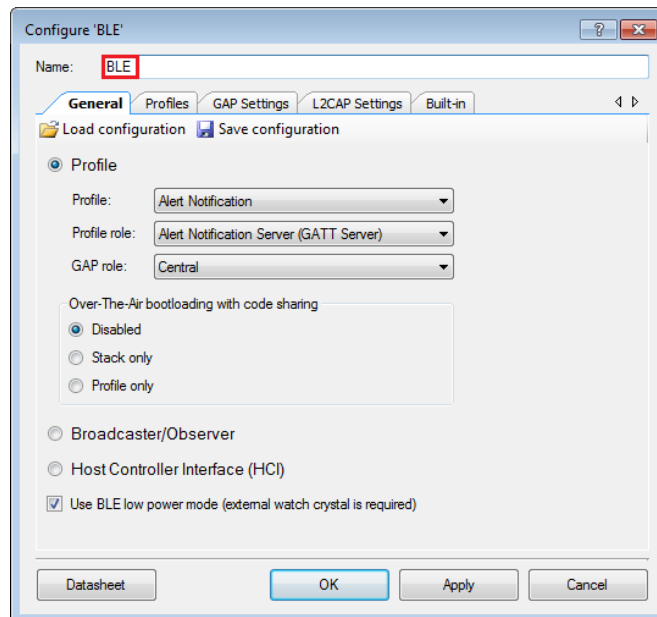
1. 从 PSoC Creator IDE 的右侧的 Component Catalog（组建目录），将 BLE 组件施放到顶层设计中，如图 10 所示。

图 10. 组建目录中的 BLE 组件



2. 双击组件，以打开它的配置窗口。将组件的实例名被改为 **BLE**，如图 11 所示。

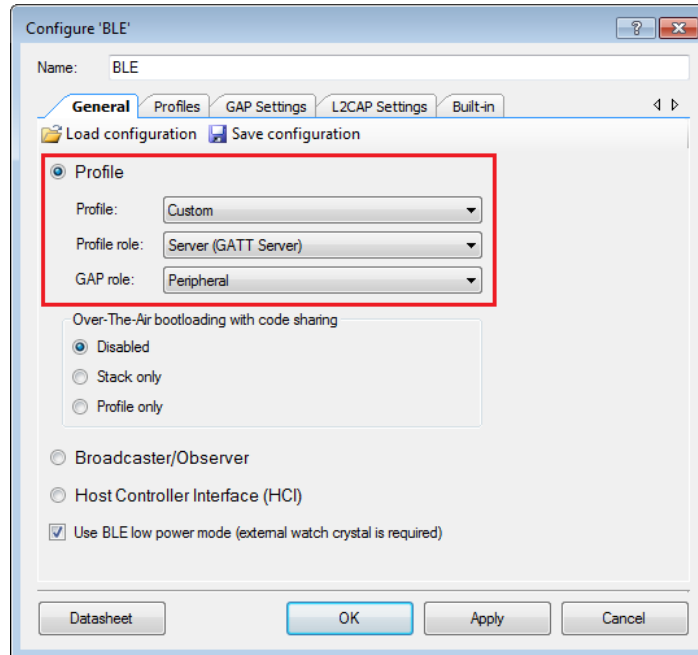
图 11. BLE 组件的实例名



**注意：**与其他 PSoC Creator 组件不同，设置在 BLE 组件中的实例名不会改变 API 命名规范。当关闭 BLE 库时，BLE 组件的 API 总是以 “CyBle\_” 开始，而不是实例名。实例名仅会改变所生成的文件名。

- 在 **General** 选项卡上，选择 **Profile** 选项，并将 **Profile** 项设定为 **Custom**，如图 12 所示。其他两个选项 “Profile role” 和 “GAP role” 将分别自动被设置为 **GATT Server** 和 **Peripheral**。

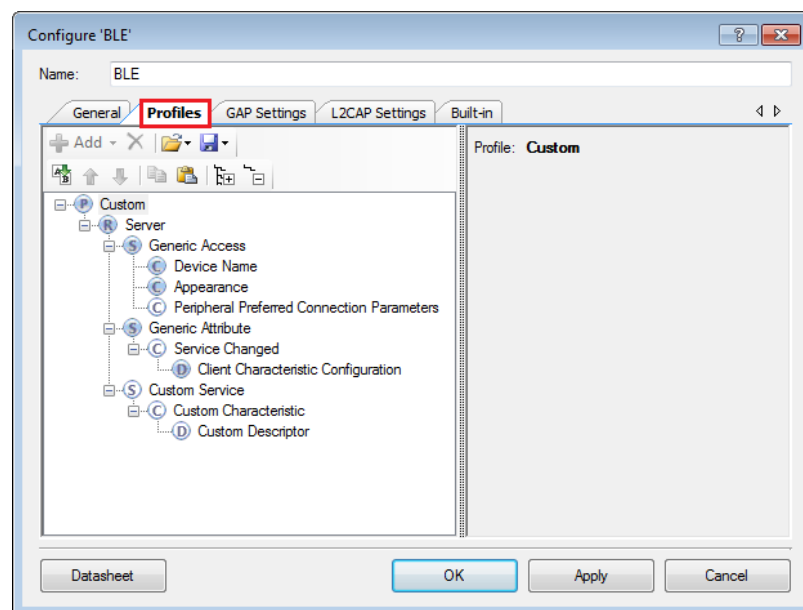
图 12. 将 “Profile Role” 设置为 “Custom”



- 在 **Profiles** 选项卡上，配置特定的 “配置文件” 参数。该组件以树形结构显示了各个服务、特性以及描述符，如图 13 所示。

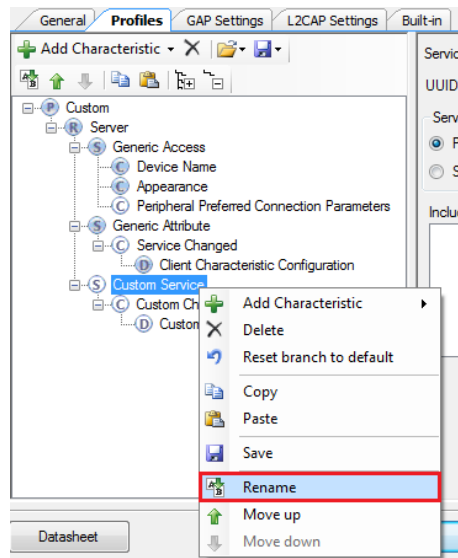
不用对 **Generic Access** 和 **Generic Attribute** 服务进行任何修改。

图 13. BLE 组件中的默认自定义配置文件树



5. 右击 **Custom Service** 并选择 **Rename**。将该服务重新命名为 **RGB LED**，如图 14 所示。

图 14. 重新命名自定义服务

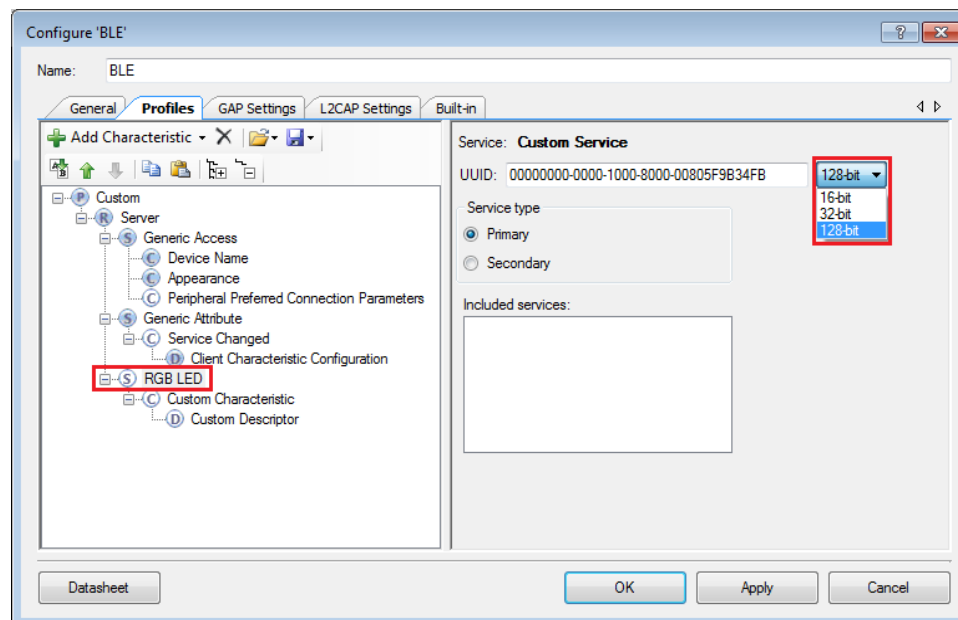


6. 请点击服务树上的 **RGB LED** 服务，然后将您自定义服务中的 UUID 格式设置为 **128 位**，如图 15 所示。GATT 客户端器件使用该 UUID 来识别 GATT 服务器器件的属性。

**注意：**该组件中的 128 位 UUID 默认值（00000000-0000-1000-8000-00805F9B34FB）是由 BLE SIG 定义的基本 UUID，用于将 16 位和 32 位 UUID 计算成完整的 128 位 UUID。

BLE SIG 建议为自定义属性使用 128 位的自定义 UUID（它与基本 UUID 不同），用于确保它不会与标准服务中现有的 UUID 发生冲突。请参考[生成自定义 UUID](#)，了解用于生成自定义 UUID 的方法。

图 15. 选择 128 位 UUID 格式

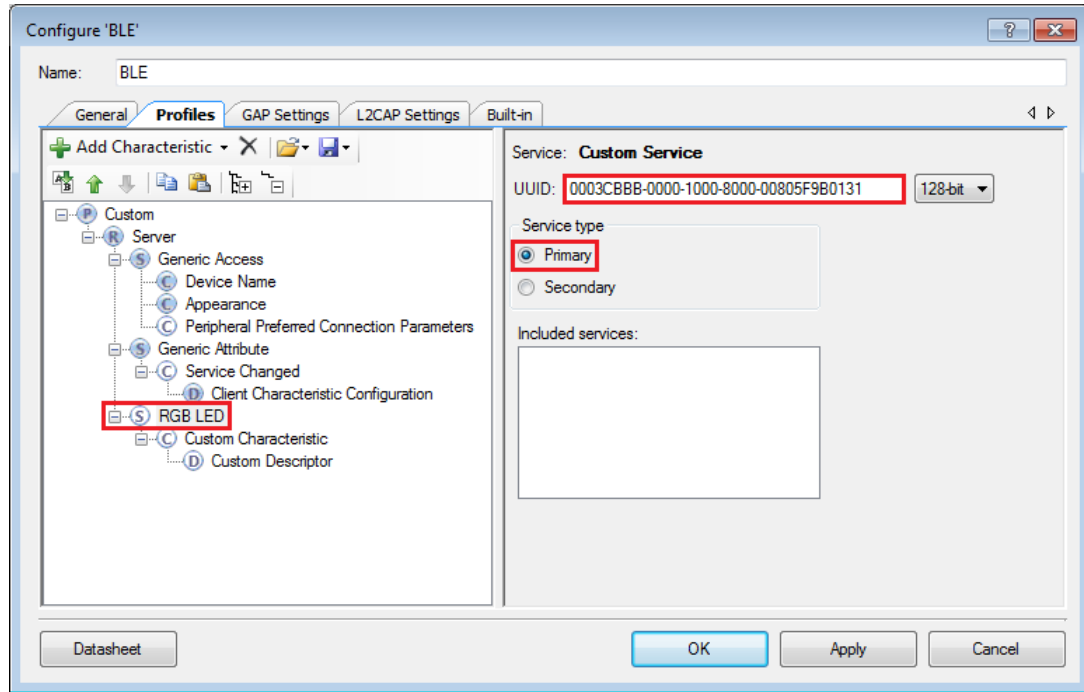


- 将 128 位 UUID 值修改成十六进制值 **0003CBBB-0000-1000-8000-00805F9B0131**。将服务类型选择为 **Primary**，如图 16 所示。

**注意：**您必须将 UUID 值指定为 **0003CBBB-0000-1000-8000-00805F9B0131**。对于 RGB LED 服务，赛普拉斯公司将它定义为 UUID。CySmart 应用则使用该 UUID 来显示 RGB LED 的 GUI 页面。

对于其他用户服务/特性，您需要生成自己的 128 位 UUID 并将其填入框中。

图 16. 设置 RGB LED 服务的 UUID

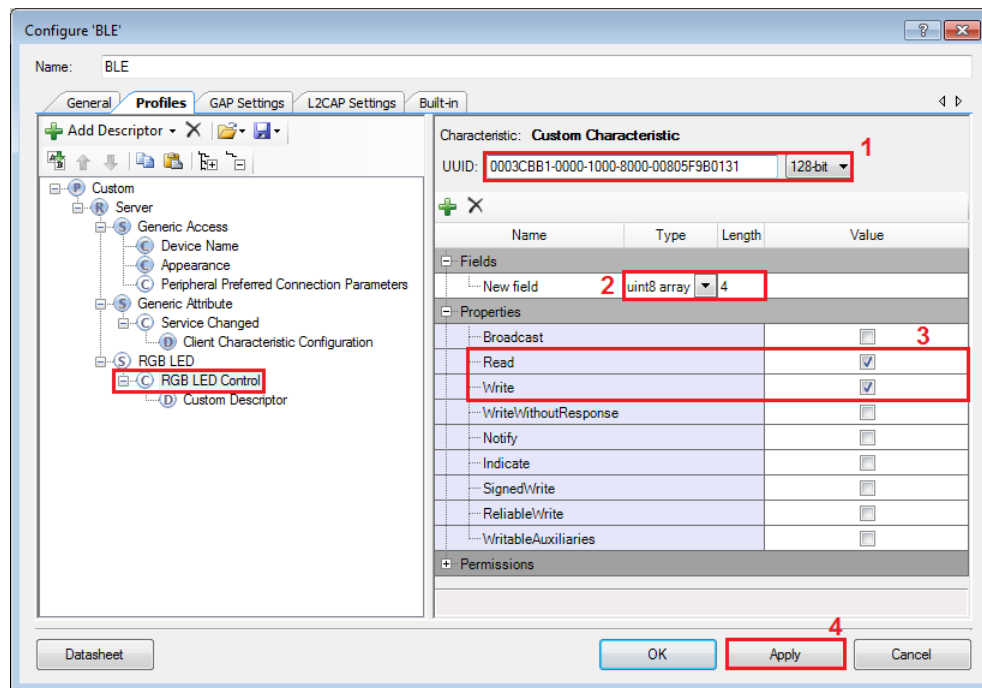


- 右击 **RGB LED** 服务下方的 **Custom Characteristic** 选项，将它命名为 **RGB LED Control**，然后按表 1 来编辑它的参数。这些修改显示在图 17 中。

表 1. RGB LED 特性参数

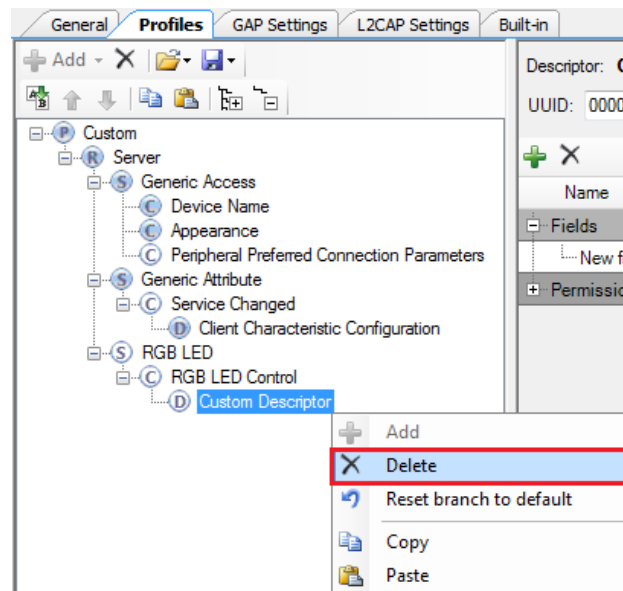
参数	值	说明
UUID	0003CBB1-0000-1000-8000-00805F9B0131	指定 RGB LED 特性的 128 位 UUID。将该值作为 UUID 使用，让移动应用为 RGB LED 提供正确的 GUI 页面。
字段	类型：uint8 阵列 长度：4	指定即将传输的数据的类型。
属性 (复选框)	读取、写入	表示 GATT 客户端器件可以对该特性进行读和写操作。

图 17. RGB LED 特性值



9. 该特性不需要自定义描述符，所以请右击 **Custom Descriptor**，并选择 **Delete**，如图 18 所示。如果您要向某个特性添加自定义信息，可以将自定义描述符加入到该特性中。

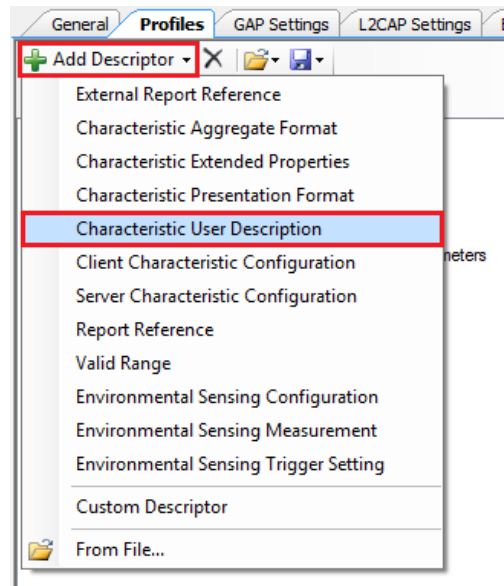
图 18. 删除自定义描述符



10. 选择 **RGB LED Control** 特性，然后单击工具栏上的 **Add Descriptor** 选项。从下拉列表中，选择 **Characteristic User Description** 描述符，如图 19 所示。



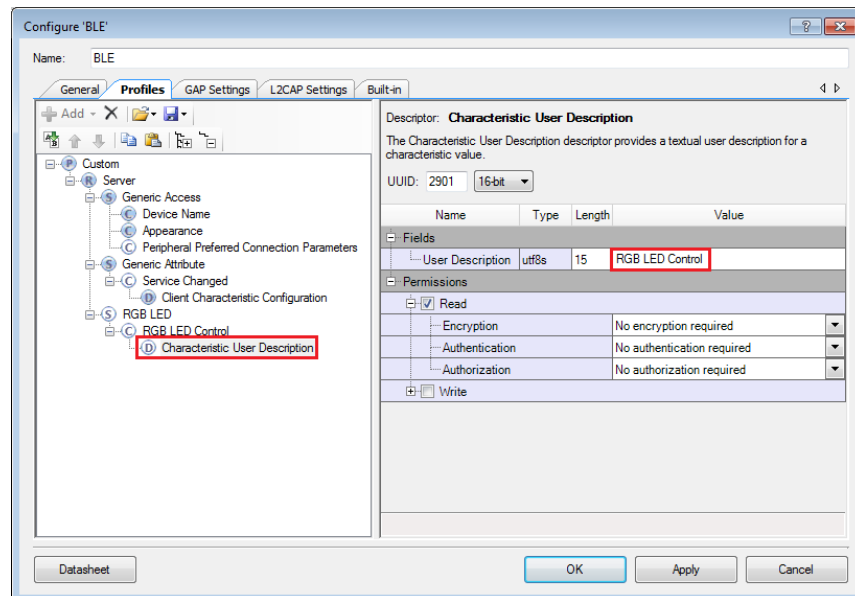
图 19. 向 RGB LED 特性添加特性用户说明



11. 点击 **Characteristic User Description** 描述符。在窗口右侧的 **Fields** 下面，请点击 **Value** 并键入 **RGB LED Control** 作为该特性的用户说明，然后将 **Permissions** 设定为 'Read'，如图 20 所示。这样客户端可以读取 RGB LED 控制特性的名称。

**注意：**特性用户说明描述符是一个由 BLE SIG 定义的标准描述符。根据 BLE 组件的规范，它的 16 位 UUID 值为 0x2901。BLE 组件添加了具有正确的 UUID 值的描述符，所以不需要对它进行任何修改。

图 20. RGB LED 特性的特性用户说明



### 5.3 配置 BLE 外设

- 在 BLE 组件配置窗口中的 **GAP Settings** 选项卡上，根据表 2 配置 **General** 配置下方的各个参数，然后点击 **Apply**。

表 2. 外设器件的通用 GAP 设置

参数	值	说明
公共地址	00A050-XXXXXX	指定蓝牙器件的 6 字节地址。在广播过程中，使用此地址。如果选中“芯片生成”选项，则地址的最后三个字节均由芯片生成。根据 BLE SIG 标准，该值必须是一个非零值。 请注意，00A050 是赛普拉斯半导体公司的公司编号。
芯片所生成的器件地址中的“公司分配”部分	已检查	允许芯片生成公共地址的公司分配部分。通过该设置，每个器件将有一个唯一的公共地址。
器件名称	赛普拉斯的自定义 BLE	指定在扫描时 GATT 客户端需要识别的器件名称。
外观	未知	指定器件的外观。对于该自定义服务，把它设置为‘Unknown’。
MTU 大小（字节）	23（默认值）	指定协议数据单元（PDU）可以在属性级别传输的大小。
Adv/San TX 的功率（dBm）	0（默认值）	指定在广播数据时的无线通信 Tx 功率。
连接 TX 的功率（dBm）	0（默认值）	指定在连接期间广播数据过程中的无线通信 Tx 功率。

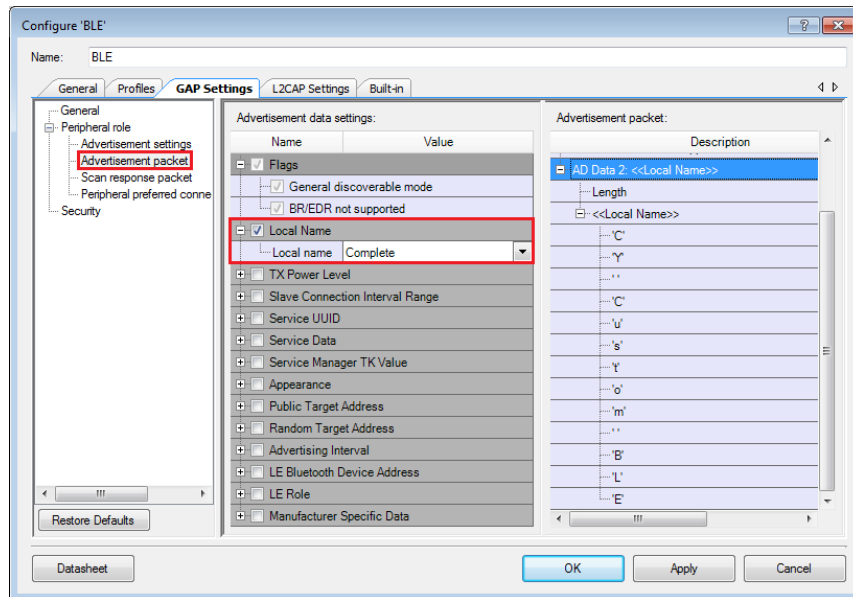
- 点击 **Peripheral role**（外设角色）下方的 **Advertisement settings**（广播设置），并根据表 3 的内容配置各参数，然后点击 **Apply** 按钮。

表 3. 广播设置配置

参数	值	说明
检测模式	普通	将器件配置为在通用模式下进行广播，以便使所有中央器件都可以找到它。
广播类型	可连接的无向广播	配置外设，使它能够进行广播而不会优先任何中央器件，并且接收来自任何扫描它的中央器件的连接请求。
过滤政策	扫描请求：任意 连接请求：任意	配置外设选择从一个特定器件或从任何中央器件接收扫描和连接要求。在该项目中，配置外设以便接收从任意中央器件发出的扫描请求和连接请求。
广播通道映射	所有通道	将外设配置为在所有三个广播通道（第 37、38 和 39 通道）上进行广播。
快速的广播间隔：最短时间（ms）	80 ms	指定广播数据的最小间隔。通过广播的最小和最大间隔，可以计算得出实际广播间隔。
快速广播间隔：最长时间（ms）	100 ms	指定广播数据的最大间隔。通过广播的最小和最大间隔，可以计算得出实际广播间隔。
快速广播间隔：超时（s）	未检查	指定外设器件在该广播间隔超时并停止之前还连续广播的时间。 如果未检查该参数，则会连续进行广播，不存在任何超时。
慢速广播间隔	未检查	禁用慢速广播。 如果该设置有效，则外设器件在快速广播超时后进入慢速广播模式。在慢速广播模式下，广播之间的间隔会更长，但是能够节省功耗。此项目不使用该特性。

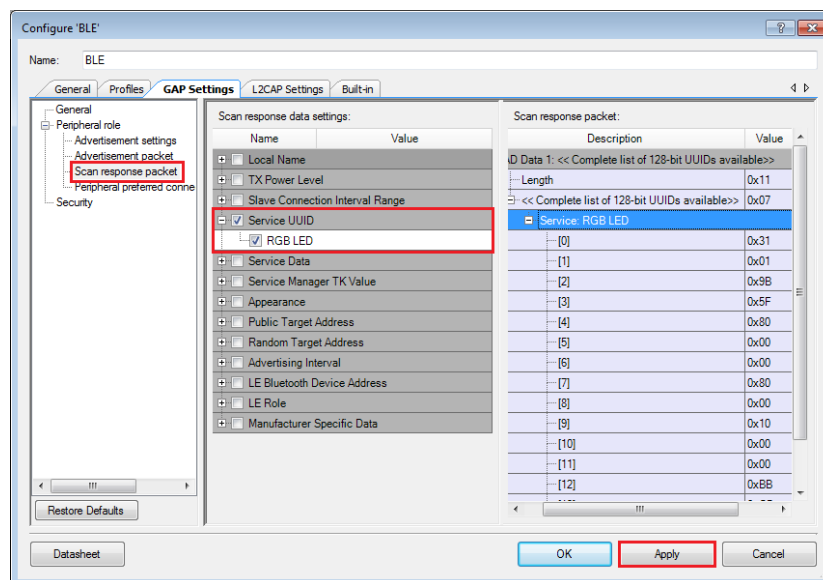
- 点击 **Peripheral role** 设置下方的 **Advertisement packet**，以指定中央器件将接收的广播数据包中的信息。对于该项目，选择需要发送的完整 **Local name** 作为广播数据包的一部分，如图 21 所示。

图 21. 广播数据包设置



4. 点击 **Peripheral role** 下方的 **Scan response packet** 项，以便指定外设扫描过程中所传输的数据，用于响应中央器件的请求。在该项目中，选择 **Service UUID > RGB LED 服务数据**（如图 22 所示），然后点击 **Apply**。

图 22. 外设的扫描响应数据包设置



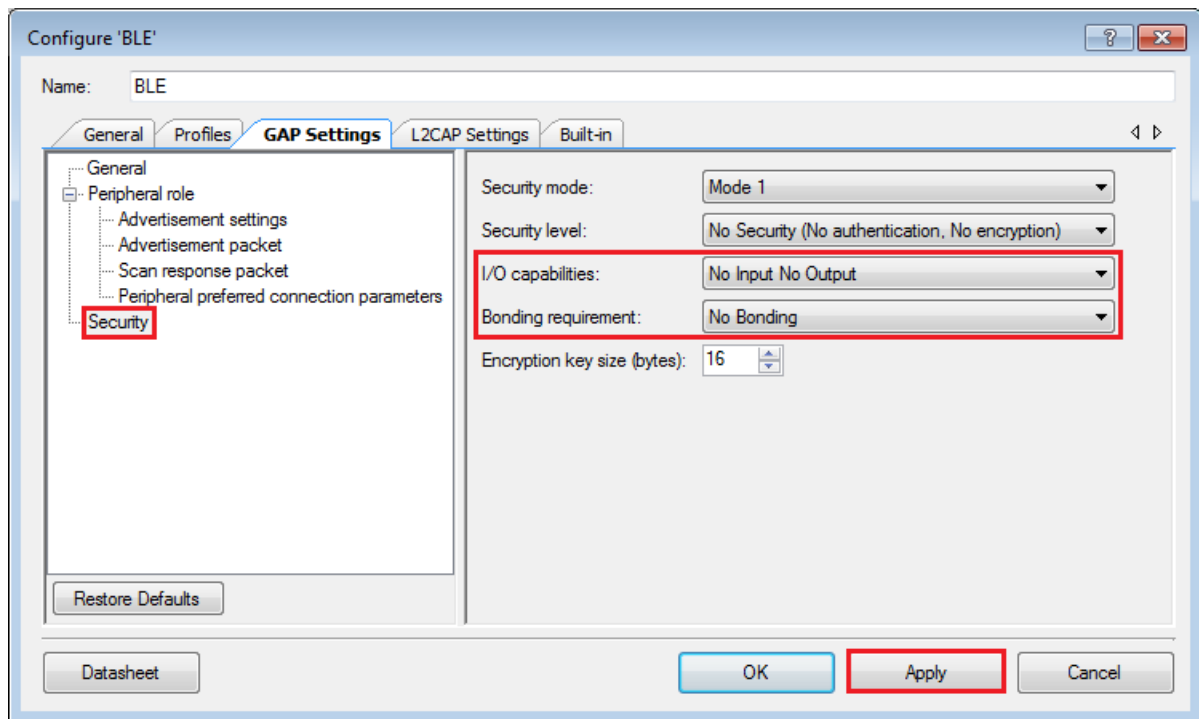
5. 点击外设优先连接的参数，并设置这些参数，如表 4 所示：

表 4. 外设优先连接的参数

参数	值	说明
连接间隔：最短时间 (ms)	75 ms	设置最小间隔，在该间隔中外设和中央器件将进入传输模式，以便在连接外设器件后反复传送数据。最小间隔越低，数据传输速率越快，但功耗越高。
连接间隔：最长时间 (ms)	80 ms	指定外设器件支持的最大连接间隔。为了成功连接，中央器件和外设器件必须满足连接间隔的要求。在连接期间内，实际的连接间隔取决于中央设备。
从设备延迟	0	设置外设器件在中央器件发出数据请求时可以选择最多不响应次数。这有助于器件需要以更快的速率传输数据，但也要在没有数据传输时仍保持低功耗模式。对于该项目，不需要任何从设备延迟。
连接监视超时 (ms)	2000 (2 秒)	设置最后一次成功连接后外设或中央器件确认连接有效的总时间。如果在这段时间内未进行任何连接，那么假定断开连接，因此器件间的连接也被断开。

6. 点击 **Peripheral role** 下方的 **Security** 项，实现 BLE 通信的安全级别设置。该项目不要求安全性的设置，所以将 “I/O capabilities” (I/O 功能) 和 “Bonding requirement” 分别被设置为 **No Input No Output** 和 **No Bonding**。将其余设置保持为其默认值，并点击 **Apply**，如图 23 所示。

图 23. 安全性设置



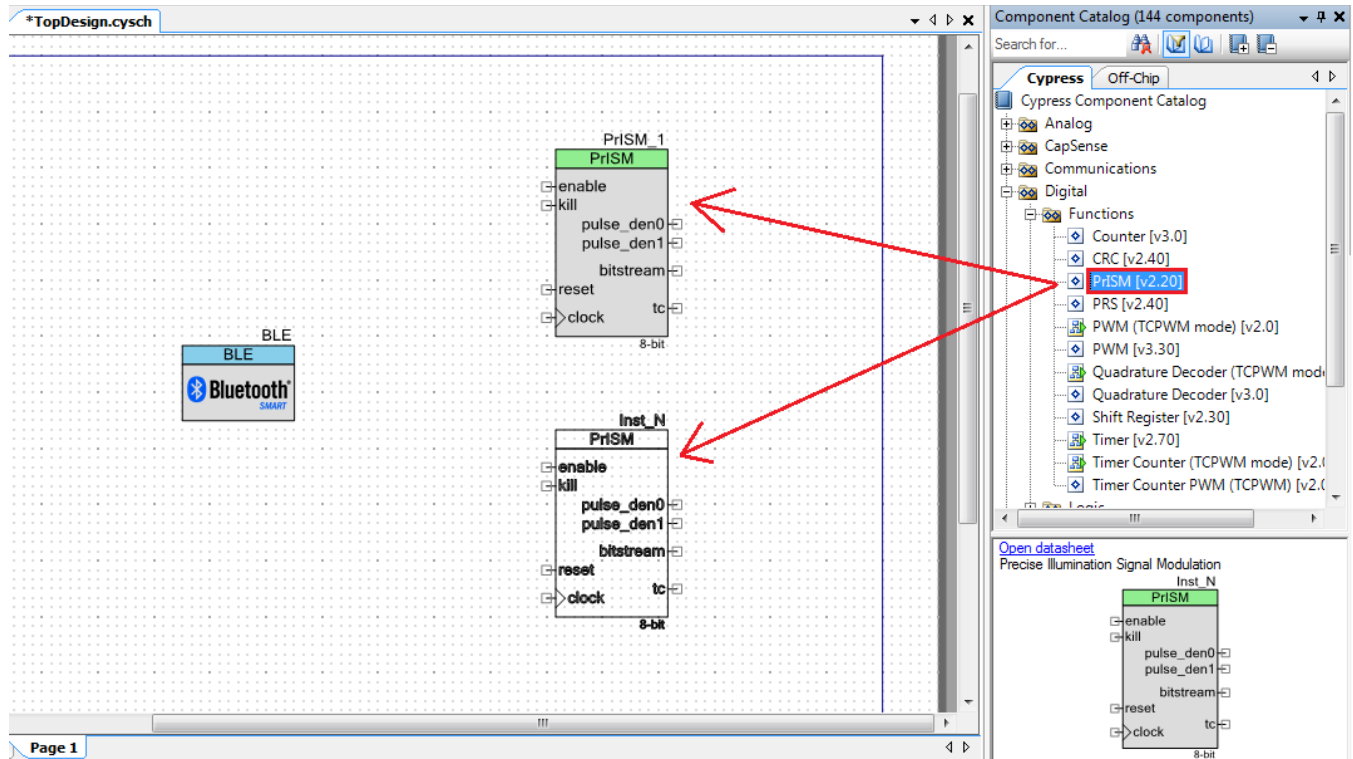
7. 点击 **OK** 来保存这些更改和关闭 BLE 组件配置窗口。

## 5.4 RGB LED 控制

对于 RGB LED 控制，该项目采用了一个基于赛普拉斯专有技术的 **PrISM™** 组件来控制 LED 发光强度。该组件使用了随机信号密度调制来控制单个 LED 的发光强度。通过结合多个 LED，可以控制颜色和强度。更多信息，请参见 [AN47372 — LED 调光的 PrISM™ 技术](#)。

1. 拖拽 Component catalog 下方的两个 **PrISM** 组件（依次选择 Cypress > Digital > PrISM），如图 24 所示。每个组件支持两个输出，所以需要使用两个 PrISM 组件来控制三个 LED。

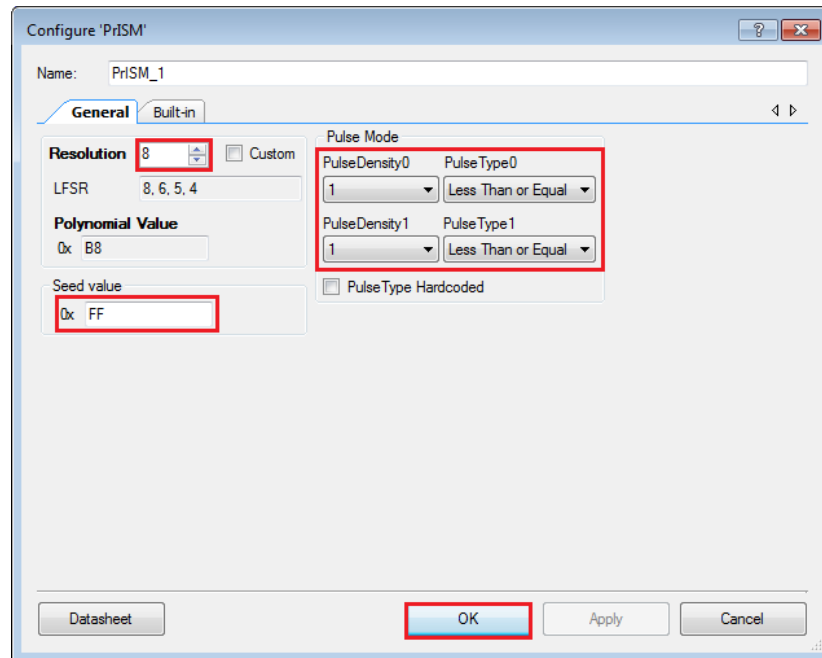
图 24. 将 PrISM 组件放置在顶层设计中。



2. 双击第一个 PrISM 组件。在配置窗口上，对 **General** 选项卡进行以下操作，然后点击 **OK**，如图 25 所示：

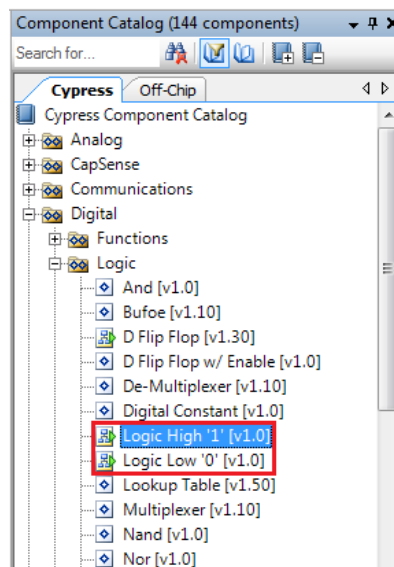
- 分别将 **Resolution** 和 **Seed value** 设置为 8 位和 0xFF。
- 在 **Pulse Mode** 模式下方，请将 **PulseDensity0** 和 **PulseDensity1** 均保持为 '1'（默认值）。比较所生成的随机数和该值。
- 将 **PulseType0** 和 **PulseType1** 都设为 **Less than or Equal**。这样，每当组件生成的随机数小于或等于所设定的脉冲密度值时，在 pulse\_den0 和 pulse\_den1 的组件输出均为高电平；否则，它将为低电平。

图 25. PrISM 组件设置



3. 以相同的设置方式配置 **PrISM\_2** 组件。在该组件中，只使用 **pulse\_den0** 输出控制第三个 LED。其他输出都处于无连接状态。
4. 将以下的组件添加到 **Component Catalog** 中两个 PrISM 组件的输入连接端：
  - 将 **Logic High (1)** 组件连接到使能输入引脚上，以便默认使能组件。
  - 将 **Logic Low (0)** 组件连接到停止 (Kill) 和复位输入引脚上，以便禁用组件的硬件复位和停止选项。该项目不需要这两个选项。

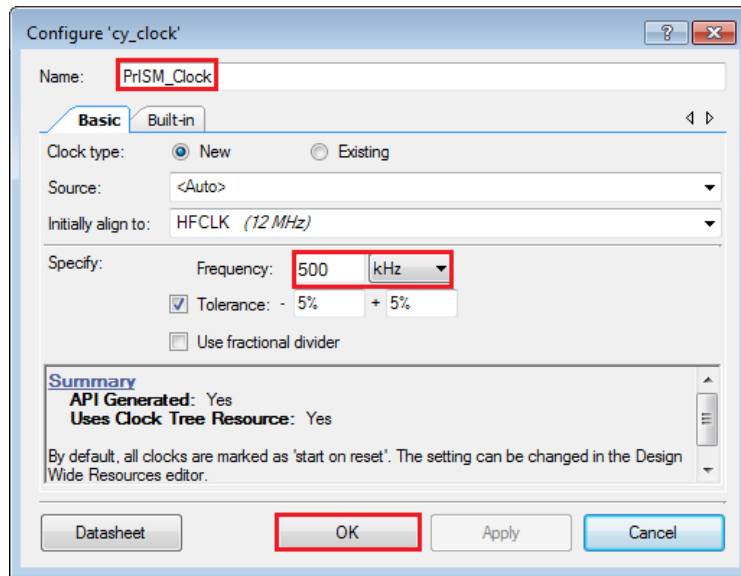
图 26. 逻辑高电平和逻辑低电平组件





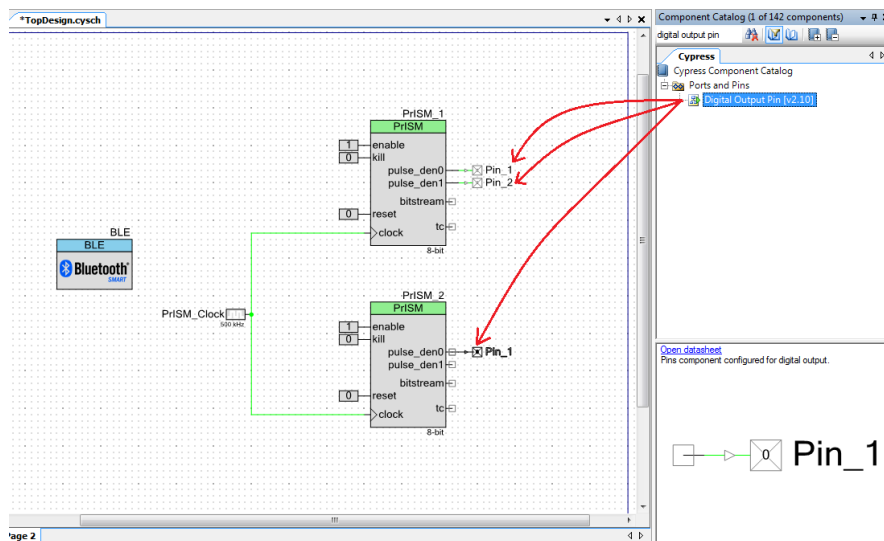
5. 从 ‘Component Catalog’ 下方的 **System** 组中施放 **Clock** 组件，并按图 27 配置它，然后点击 **OK**：
  - a. 将实例重新命名为 **PrISM\_Clock**。
  - b. 将频率设置为 **500 kHz**。

图 27. PrISM 时钟配置



6. 通过使用连线工具，将 **Clock** 组件连接至两个 PrISM 组件的 **clock** 输入端（在顶层设计中的任何位置按 ‘w’，以便使能连线功能，然后点击连接点）。
7. 从 ‘Component Catalog’ 下方的 **Ports and Pins** 组中，拖放三个 **Digital Output Pin** 组件。将这些组件连接到 PrISM\_1 的 **pulse\_den0** 和 **pulse\_den1** 引脚以及 PrISM\_2 的 **pulse\_den0** 引脚上，如图 28 所示。这些引脚由 PrISM 组件驱动，并控制 RGB LED。

图 28. 将数字输出引脚连接到 PrISM

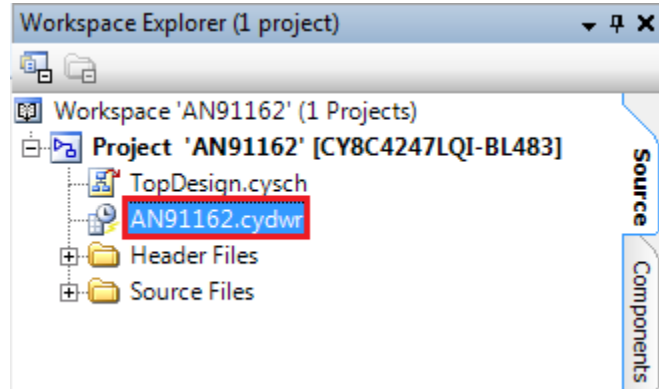


8. 双击 Pin\_1、Pin\_2 和 Pin\_3 组件，并分别将它们命名为 **RED**、**GREEN** 和 **BLUE**。将驱动模式设置为 **High impedance Analog** 模式。这样做是因为 BLE Pioneer 套件上的 RGB LED 是低电平有效的，RGB LED 引脚的初始强驱动模式将导致 RGB LED 在较短的时间内发出白光。

## 5.5 配置项目的设计范围内的资源

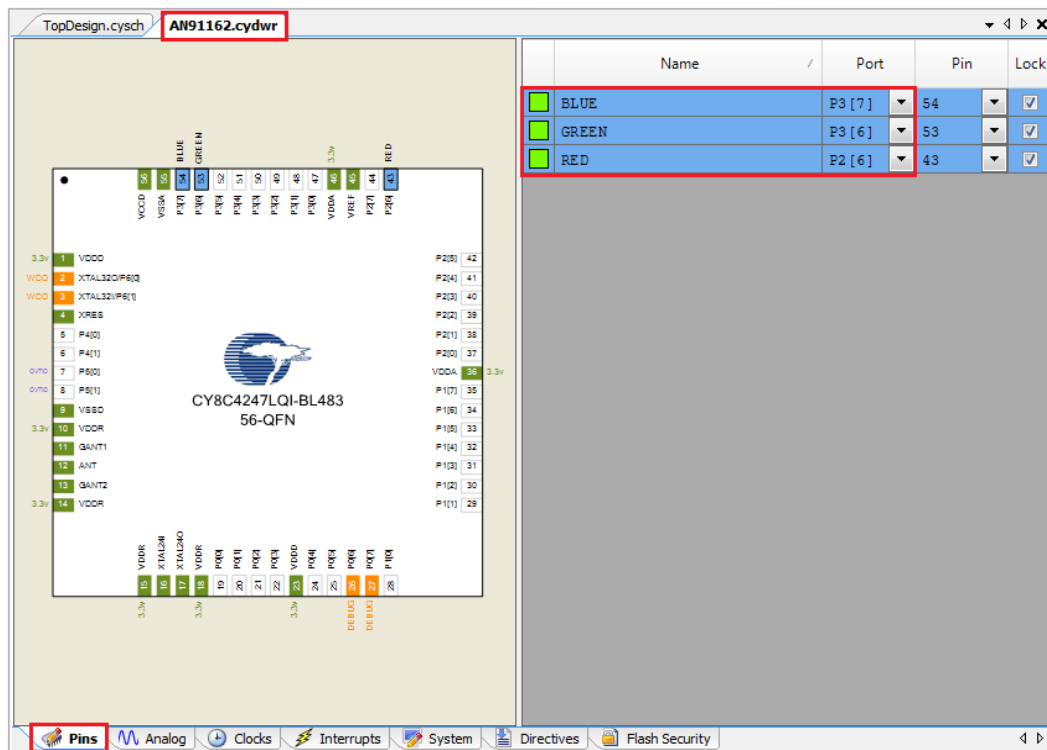
1. 为了将各端口分配给各个组件，请双击工作区浏览器中的项目 CYDWR，如图 29 所示。

图 29. 打开项目 CYDWR



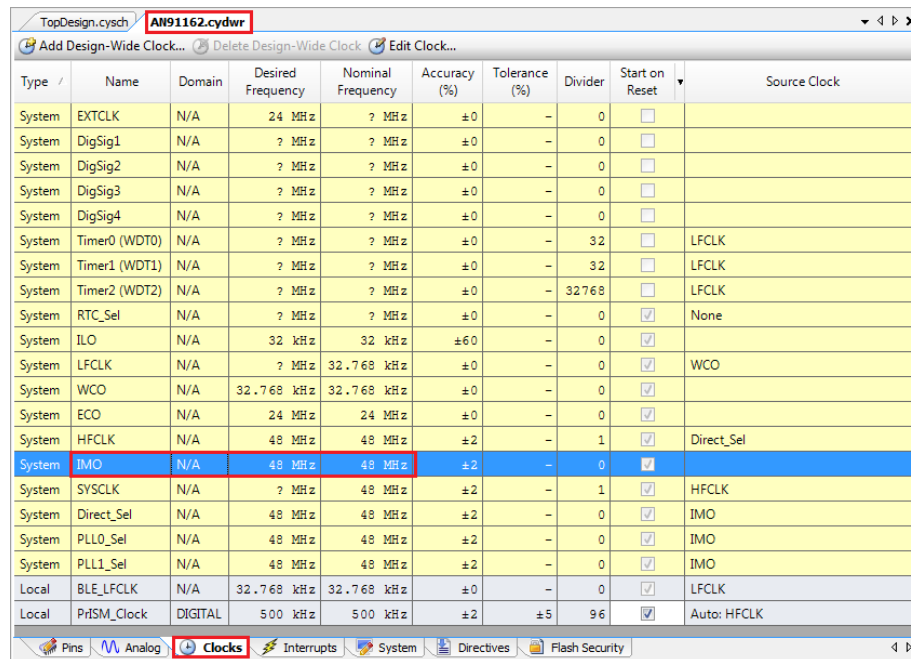
2. 在 **Pins** 选项卡上，配置每个组件的引脚数量，如图 30 所示。您可以使用下拉菜单，键入端口名称（例如：P3[7]），或将引脚名称拖动到图中所需位置，以便分配各个端口。

图 30. CYDWR 中的引脚配置



- 在 **Clocks** 选项卡上，通过双击 **IMO** 时钟，打开系统时钟配置窗口，如图 31 所示。

图 31. CYDWR 时钟设置

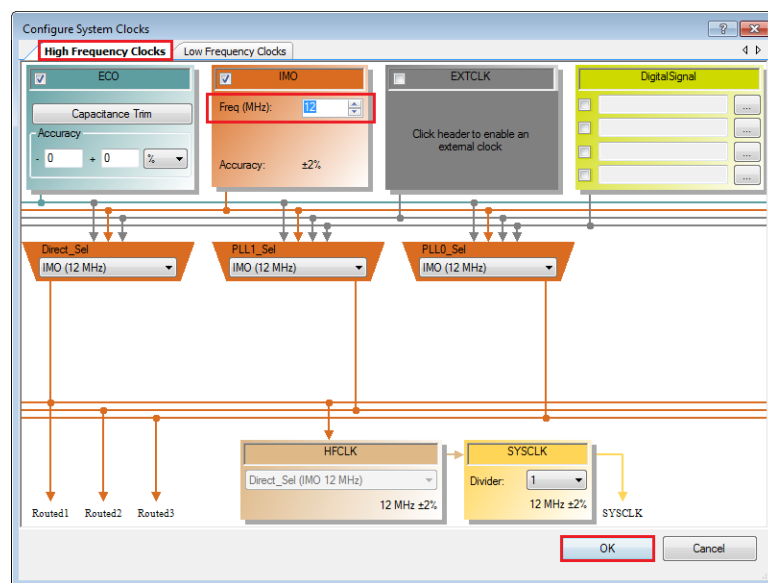


Type	Name	Domain	Desired Frequency	Nominal Frequency	Accuracy (%)	Tolerance (%)	Divider	Start on Reset	Source Clock
System	EXTCLK	N/A	24 MHz	? MHz	±0	—	0	<input type="checkbox"/>	
System	DigSig1	N/A	? MHz	? MHz	±0	—	0	<input type="checkbox"/>	
System	DigSig2	N/A	? MHz	? MHz	±0	—	0	<input type="checkbox"/>	
System	DigSig3	N/A	? MHz	? MHz	±0	—	0	<input type="checkbox"/>	
System	DigSig4	N/A	? MHz	? MHz	±0	—	0	<input type="checkbox"/>	
System	Timer0 (WDT0)	N/A	? MHz	? MHz	±0	—	32	<input type="checkbox"/>	LFCLK
System	Timer1 (WDT1)	N/A	? MHz	? MHz	±0	—	32	<input type="checkbox"/>	LFCLK
System	Timer2 (WDT2)	N/A	? MHz	? MHz	±0	—	32768	<input type="checkbox"/>	LFCLK
System	RTC_Sel	N/A	? MHz	? MHz	±0	—	0	<input checked="" type="checkbox"/>	None
System	ILO	N/A	32 kHz	32 kHz	±60	—	0	<input checked="" type="checkbox"/>	
System	LFCLK	N/A	? MHz	32.768 kHz	±0	—	0	<input checked="" type="checkbox"/>	WCO
System	WCO	N/A	32.768 kHz	32.768 kHz	±0	—	0	<input checked="" type="checkbox"/>	
System	ECO	N/A	24 MHz	24 MHz	±0	—	0	<input checked="" type="checkbox"/>	
System	HFCLK	N/A	48 MHz	48 MHz	±2	—	1	<input checked="" type="checkbox"/>	Direct_Sel
System	IMO	N/A	48 MHz	48 MHz	±2	—	0	<input checked="" type="checkbox"/>	
System	SYSCLK	N/A	? MHz	48 MHz	±2	—	1	<input checked="" type="checkbox"/>	HFCLK
System	Direct_Sel	N/A	48 MHz	48 MHz	±2	—	0	<input checked="" type="checkbox"/>	IMO
System	PLL0_Sel	N/A	48 MHz	48 MHz	±2	—	0	<input checked="" type="checkbox"/>	IMO
System	PLL1_Sel	N/A	48 MHz	48 MHz	±2	—	0	<input checked="" type="checkbox"/>	IMO
Local	BLE_LFCLK	N/A	32.768 kHz	32.768 kHz	±0	—	0	<input checked="" type="checkbox"/>	LFCLK
Local	PrISM_Clock	DIGITAL	500 kHz	500 kHz	±2	±5	96	<input checked="" type="checkbox"/>	Auto: HFCLK

- 对于该项目，IMO 频率（高频率时钟）将从默认的 48 MHz 下降到 12 MHz，以降低功耗，如图 32 所示。保持其他时钟配置为各自的默认值，然后点击 **OK**。

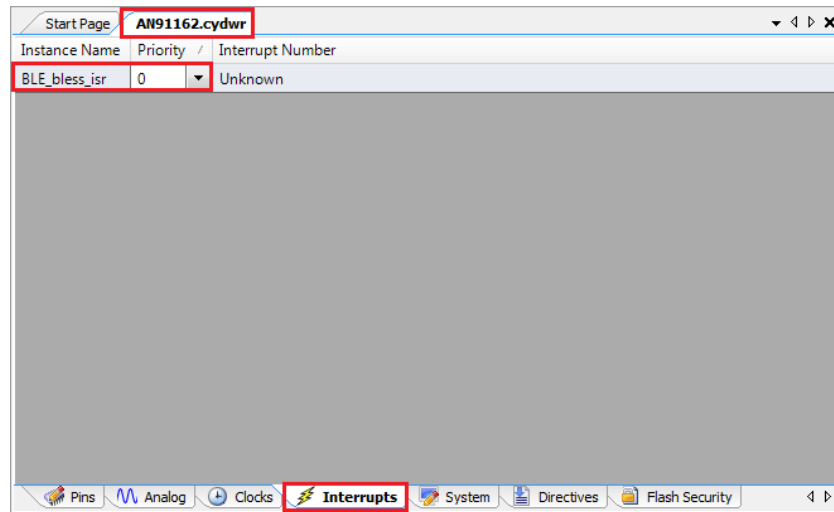
**注意：**此处设置的 CPU 时钟频率将影响器件的总功耗。此外，一些外设需要最低的时钟频率才能正常工作。选择一个 CPU 时钟频率来保持低功耗但不妨碍项目操作。

图 32. IMO 时钟设置



- 在 **Interrupt** 选项卡上，将 BLE 中断的优先级设置为 '0'，如图 33 所示。这样能确保所添加的任何其他低优先级的中断都不会影响 BLE 的操作。

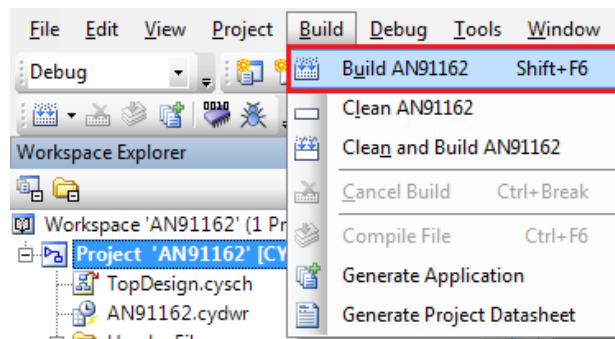
图 33. CYDWR 中断设置



## 5.6 构建项目

依次选择 **Build > Build AN91162 [Shift+F6]** 来编译完整的项目，如图 34 所示。

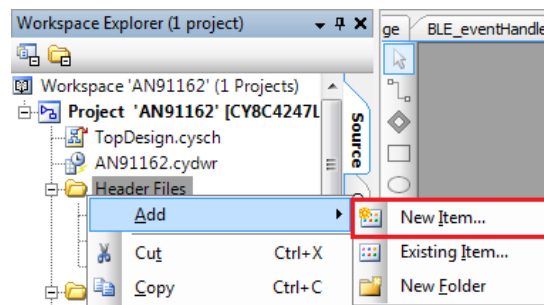
图 34. 编译项目



## 5.7 将源文件/头文件添加到项目中

为了将一个新的头（H）文件或源（C）文件添加到项目中，请右击 **Header Files** 或 **Source Files**，然后依次选择 **Add > New Item**，如图 35 所示。选择需要添加的文件类型，输入所需名称，然后点击 **OK**。

图 35. 添加一个源文件/头文件



## 5.8 项目文件

相关项目包含以下文件：

- **main.c/h**: 这些文件包含了可作为系统输入点的主函数。它可以初始化系统（包括 BLE），并定期调用函数来处理 BLE 事件。
- **BLEProcess.c/h**: 这些文件包含了各个函数的定义，这些函数用于处理 BLE 事件回调和更新 GATT 数据库中的 RGB LED 特性值。
- **led.c/h**: 这些文件对函数进行了定义，该函数处理用于显示 RGB LED 的颜色和亮度的组件。

## 5.9 配置固件

该项目的固件执行以下操作：

- 初始化组件并使能中断。
- 处理由 BLE 堆栈生成的 BLE 事件，如启动 BLE、连接请求和写命令。
- 当接收到 GATT 客户端传输的一个新的颜色数据时，固件将通过 RGB LED 显示颜色。

该项目使用了以下的 BLE API：

API	说明
<b>CyBle_Start(CYBLE_CALLBACK_T)</b>	启动 BLE 组件并将一个函数注册为事件处理程序，以处理 BLE 堆栈所生成的事件。该函数的参数就是事件处理程序的名称。
<b>CyBle_ProcessEvents(void)</b>	处理 BLE 堆栈和应用程序间的 BLE 事件。主函数将继续调用该 API。该函数中没有任意参数。
<b>CyBle_GappStartAdvertisement(uint8)</b>	在 BLE 组件中设置的间隔（如表 3 所述）内启动 BLE 外设广播。参数定义了广播速度是快速、慢速还是自定义的。
<b>CyBle_GattsWriteRsp(CYBLE_CONN_HANDLE_T)</b>	GATT 客户端器件一旦发送写请求，该函数会立即将写响应传输给它。该函数将连接句柄作为它的参数使用。
<b>CyBle_GattsWriteAttributeValue(CYBLE_GATT_HANDLE_VALUE_PAIR_T *, uint16, CYBLE_CONN_HANDLE_T *, uint8)</b>	更新某个属性（如一个特性）的数据值，这样 GATT 客户端器件可以读取该值。该函数使用四个参数来接收更新后的数据、偏移、连接句柄以及与所传送数据相关的标志。

### 5.9.1 宏定义

每个头文件包含了代码中使用的常量宏。每个文件中的宏显示如下：

main.h

```
#define TRUE          0x01
#define FALSE        0x00
```

BLEProcess.h

```
/* RGB LED Characteristic data length */
#define RGB_CHAR_DATA_LEN 4
```

led.h

```
/* LED Color and status related Macros */
#define RGB_LED_MAX_VAL 0xFF
#define RGB_LED_OFF 0xFF
#define RGB_LED_ON 0x00

/* Index values in array where respective color coordinates
 * are saved */
#define RED_INDEX 0x00
#define GREEN_INDEX 0x01
```

```
#define BLUE_INDEX                0x02
#define INTENSITY_INDEX          0x03
```

### 5.9.2 系统初始化

在固件配置过程中，第一步要初始化系统中的各组件。先要调用以下函数，然后进入 *main.c*。双击 PSoC Creator 窗口左侧的 Workspace Explorer 窗口上的 *main.c*，以打开该函数。将以下函数定义添加到 *main.c* 内：

```
void InitializeSystem(void)
{
    /* Enable Global Interrupt Mask */
    CyGlobalIntEnable;

    /* Start BLE stack and register the event callback function. */
    CyBle_Start(GeneralEventHandler);

    /* Start PrISM modules for LED control */
    PrISM_1_Start();
    PrISM_2_Start();

    /* Switch off the RGB LEDs through PrISM modules */
    PrISM_1_WritePulse0(RGB_LED_OFF);
    PrISM_1_WritePulse1(RGB_LED_OFF);
    PrISM_2_WritePulse0(RGB_LED_OFF);

    /* Set Drive modes of the output pins to Strong */
    RED_SetDriveMode(RED_DM_STRONG);
    GREEN_SetDriveMode(GREEN_DM_STRONG);
    BLUE_SetDriveMode(BLUE_DM_STRONG);
}
```

### 5.9.3 事件处理程序的注册

与其他组件的启动不同，BLE 在启动时要求注册**事件回调函数**。调用该函数处理 BLE 事件，包括通用事件（如 BLE 堆栈为‘ON’）和 GAP/GATT 层的事件（如连接、断开和写命令）。通用事件处理程序函数定义在示例项目的 *BLEProcess.c* 中。您可以将其放置在某个单独的文件中或 *main.c* 中。有关包含在切换语句中的事件的说明，请参见表 5。在以下所述的函数定义中，每一 Case 都是空白的。我们将添加代码以处理下一节中的事件。

```
void GeneralEventHandler(uint32 event, void * eventParam)
{
    /* Structure to store data written by Client */
    CYBLE_GATTS_WRITE_REQ_PARAM_T *wrReqParam;

    /* 'RGBledData[]' is an array to store 4 bytes of RGB LED data*/
    uint8 RGBledData[RGB_CHAR_DATA_LEN];

    switch(event)
    {
        case CYBLE_EVT_STACK_ON:
            /* This event is generated when BLE stack is ON */

            break;

        case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
            /* This event is generated at GAP disconnection. */

            break;
    }
```



```

    case CYBLE_EVT_GATTS_WRITE_REQ:
        /* This event is generated when the connected Central */
        /* device sends a Write request. */
        /* The parameter 'eventParam' contains the data written */

        break;

    default:

        break;

}

```

在应用程序中需要处理这些基础事件，这样才能成功连接到 BLE。表 5 说明了这些事件。“CYBLE\_EVENT\_T” enum 中的 *BLE\_Stack.h* 文件说明了由 BLE 组件生成的其他事件。

表 5. BLE 事件

事件名称	事件说明	事件处理
CYBLE_EVT_STACK_ON	调用 CyBle_Start()后，可成功初始化 BLE 堆栈。	BLE 堆栈为 ‘ON’ 时，将启动广播。
CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP	外设广播启动或停止。	进入低功耗模式或重新启动该广播。
CYBLE_EVT_GAP_DEVICE_DISCONNECTED	外设与中央器件间的 BLE 连接被断开。	进入低电平模式或重新启动该广播。
CYBLE_EVT_GATT_CONNECT_IND	建立了外设与中央器件间的连接。	更新连接句柄变量。 不适用于该项目。
CYBLE_EVT_GATT_DISCONNECT_IND	GATT 与中央器件间的连接被断开。	复位 GATT 数据库的值。
CYBLE_EVT_GATTS_WRITE_REQ	GATT 客户端器件已发送一个写请求。	提取 GATT 客户端发送的数据并发送写响应。

#### 5.9.4 启动广播

当项目是一个 GAP 外设时，它要求启动广播，以便能够将 GAP 中央器件连接到该项目。可以通过两种事件启动广播。在发生以下事件时，将相应的代码放置在通用事件的回调函数中：

- 系统上电和 BLE 堆栈为 ‘ON’ (event CYBLE\_EVT\_STACK\_ON)

```

case CYBLE_EVT_STACK_ON:
    /* BLE stack is on. Start BLE advertisement */
    CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
break;

```

- GAP 与中央器件间的连接被断开(event CYBLE\_EVT\_GAP\_DEVICE\_DISCONNECTED)

```

case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
    /* This event is generated at GAP disconnection. */
    /* Restart advertisement */
    CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
break;

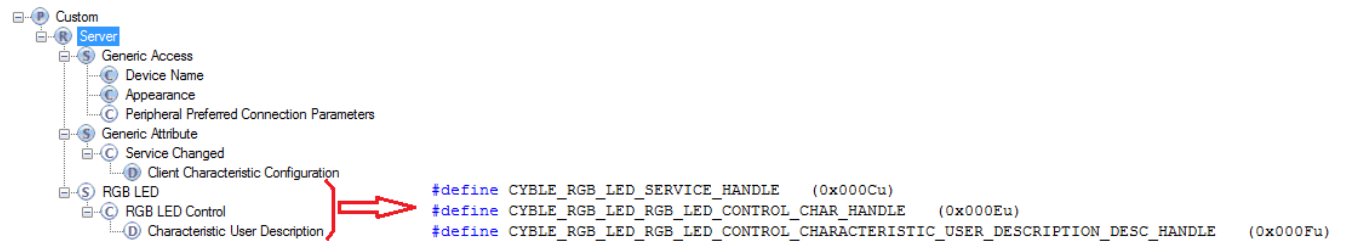
```

#### 5.9.5 自定义服务的属性句柄

在 BLE 进行通信时，GATT 客户端和 GATT 服务器都通过使用一个属性句柄访问服务、特性或描述符等属性中的数据。属性句柄是一个 16 位的数值，它只会识别建立连接后的属性。

对于所添加到 BLE 组件内的自定义服务和特性，这些句柄的值都由组件生成，可以在所生成的文件 *BLE\_custom.h* 中找到这些句柄（其格式为 *#defines*）。对于该项目中添加的 BLE 自定义服务（RGB LED），所生成的句柄如图 36 中所示。

图 36. 自定义服务的属性句柄数据结构



RGB LED 服务支持使用数值为 *0x000E* 的属性句柄对同一个特性进行读/写操作。

### 5.9.6 处理写请求

对于 RGB LED 特性，GATT 客户端发送了四字节数据的写请求。将接收到的数据作为通用事件回调函数中 *CYBLE\_EVT\_GATTS\_WRITE\_REQ* 事件的一部分。将所接收的数据的属性句柄与 RGB LED 控制特性的句柄进行比较。如果它们相互匹配，则采取以下措施：

1. 提取四个字节的的数据并将它们存储在阵列中。
2. 调用 RGB LED 更新函数（*UpdateRGBLED*），以更新板上 LED 的颜色。
3. 调用 RGB 控制特性更新函数（*UpdateRGBcharacteristic*），以更新内部 GATT 数据库的值。
4. 不管属性句柄是否与 RGB LED 控制特性句柄匹配，都要使用 BLE 函数 *CyBle\_GattsWriteRsp* 将写响应回送到客户端器件内，从而使客户端知道已经收到数据。

将以下代码放置在 *CYBLE\_EVT\_GATTS\_WRITE\_REQ* 事件中：

```
case CYBLE_EVT_GATTS_WRITE_REQ:
    /* Extract the Write data sent by Client */
    wrReqParam = (CYBLE_GATTS_WRITE_REQ_PARAM_T *) eventParam;

    /* If the attribute handle of the characteristic written to
     * is equal to that of RGB_LED characteristic, then extract
     * the RGB LED data */
    if (CYBLE_RGB_LED_RGB_LED_CONTROL_CHAR_HANDLE ==
        wrReqParam->handleValPair.attrHandle)
    {
        /* Store RGB LED data in local array */
        RGBledData[RED_INDEX] =
            wrReqParam->handleValPair.value.val[RED_INDEX];
        RGBledData[GREEN_INDEX] =
            wrReqParam->handleValPair.value.val[GREEN_INDEX];
        RGBledData[BLUE_INDEX] =
            wrReqParam->handleValPair.value.val[BLUE_INDEX];
        RGBledData[INTENSITY_INDEX] =
            wrReqParam->handleValPair.value.val[INTENSITY_INDEX];

        /* Update the PrISM component density value to represent color */
        UpdateRGBLED(RGBledData, sizeof(RGBledData));

        /* Update the GATT DB for RGB LED read characteristics*/
        UpdateRGBcharacteristic(RGBledData,
                                sizeof(RGBledData),
                                CYBLE_RGB_LED_RGB_LED_CONTROL_CHAR_HANDLE);
    }
}
```

```

    /* Send the response to the write request received. */
    CyBle_GattsWriteRsp(cyBle_connHandle);
    break;

```

UpdateRGBLED 函数通过使用所接收的四字节数据（红色、绿色、蓝色和亮度）值来计算每个 RGB LED 的亮度，然后对 PrISM 组件的密度值进行更新，以获得所需颜色。将以下函数放置到项目中（定义在相关项目的 *led.c* 文件中）：

```

void UpdateRGBLED(uint8* ledData, uint8 len)
{
    /* Local variables to store calculated color components */
    uint8 calc_red;
    uint8 calc_green;
    uint8 calc_blue;

    /* Check if the array has length equal to expected length for
    * RGB LED data */
    if(len == RGB_CHAR_DATA_LEN)
    {
        /* True color to be displayed is calculated on basis of color
        * and intensity value received */
        calc_red = (uint8)
        (((uint16)ledData[RED_INDEX]*ledData[INTENSITY_INDEX])/RGB_LED_MAX_VAL);
        calc_green = (uint8)
        (((uint16)ledData[GREEN_INDEX]*ledData[INTENSITY_INDEX])/RGB_LED_MAX_VAL);
        calc_blue = (uint8)
        (((uint16)ledData[BLUE_INDEX]*ledData[INTENSITY_INDEX])/RGB_LED_MAX_VAL);

        /* Update the density value of the PrISM module */
        PrISM_1_WritePulse0(RGB_LED_MAX_VAL - calc_red);
        PrISM_1_WritePulse1(RGB_LED_MAX_VAL - calc_green);
        PrISM_2_WritePulse0(RGB_LED_MAX_VAL - calc_blue);
    }
}

```

设置好 LED 的颜色时，就要更新 RGB LED 特性的 GATT 数据库，从而客户端会收到发送读请求时所设置的最新 RGB 颜色。UpdateRGBcharacteristic 函数为 RGB LED 颜色控制更新了属性值。将以下函数添加到项目中（定义在相关项目的 *BLEProcess.c* 文件中）：

```
void UpdateRGBcharacteristic(uint8* ledData, uint8 len, uint16 attrHandle)
{
    /* 'rgbHandle' stores RGB control data parameters */
    CYBLE_GATT_HANDLE_VALUE_PAIR_T      rgbHandle;

    /* Update RGB control handle with new values */
    rgbHandle.attrHandle = attrHandle;
    rgbHandle.value.val = ledData;
    rgbHandle.value.len = len;

    /* Update the RGB LED attribute value. This will allow
    * Client device to read the existing color values over
    * RGB LED characteristic */
    CyBle_GattsWriteAttributeValue(&rgbHandle,
                                   FALSE,
                                   &cyBle_connHandle,
                                   CYBLE_GATT_DB_PEER_INITIATED);
}
```

### 5.9.7 处理 BLE 断开连接

当器件与中央器件断开连接时，在建立下一个连接前，应该复位 RGB LED 和 GATT 的数据库。将以下的代码段以及启动广播 API 调用函数放置在通用事件回调函数中的 CYBLE\_EVT\_GAP\_DEVICE\_DISCONNECTED 事件内：

```
case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
    /* This event is generated at GAP disconnection. */

    /* Reset the color values*/
    RGBledData[RED_INDEX] = FALSE;
    RGBledData[GREEN_INDEX] = FALSE;
    RGBledData[BLUE_INDEX] = FALSE;
    RGBledData[INTENSITY_INDEX] = FALSE;

    /* Switch off LEDs */
    UpdateRGBLED(RGBledData, sizeof(RGBledData));

    /* Register the new color in GATT DB*/
    UpdateRGBcharacteristic(RGBledData,
                            sizeof(RGBledData),

                            CYBLE_RGB_LED_RGB_LED_CONTROL_CHAR_HANDLE);

    /* Restart advertisement */
    CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
break;
```

### 5.9.8 主函数

如果已经完成通用事件回调函数，那么我们目前可以修改主函数，实现初始化项目中的组件和处理 BLE 事件。按照下面提供的代码修改 *main.c* 中已存在的函数：

```
int main()
{
    /* Start the components */
    InitializeSystem();

    for(;;)
```

```

{
    /* Process BLE Events. This generates events in the callback function */
    CyBle_ProcessEvents();
}

```

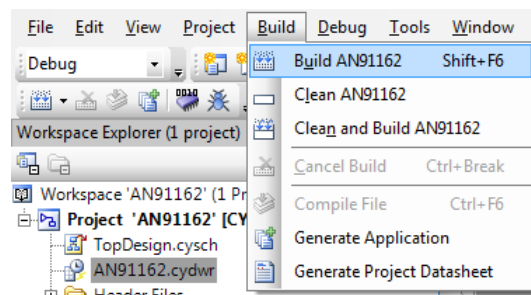
应该定期调用 `CyBle_ProcessEvents()`，至少在每个 BLE 连接间隔内调用一次，以成功处理 BLE 事件。

有关完整的固件，请参考相关项目。

## 5.10 编译和编程

- 依次选择 **Build > Build AN91162**，以便构建并编译固件，如图 37 所示。项目应在无警告或错误的情况下完成编译流程。

图 37. 编译项目



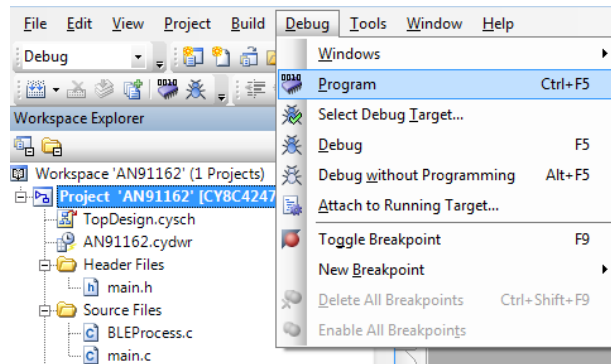
- 将 PSoC 4 BLE 模块（红色模块）插入到 BLE Pioneer 基板上，然后通过使用 USB 标准 A 转 Mini B 型线缆将该套件连接至您的 PC（请参见图 38）。这样可以在 PC 上完成 USB 枚举。

图 38. 使用一根 USB 线缆将套件连接到 PC 上



- 依次选择 **Debug > Program**（请参见图 39）。如果 PC 上只连接了一个套件，将自动开始编程。如果有多个套件连接至 PC，那么 PSoC Creator 将提示您选择要编程的套件。

图 39. 编程 PSoC 4 BLE 器件



完成编程流程后，BLE Pioneer 套件将开始广播。

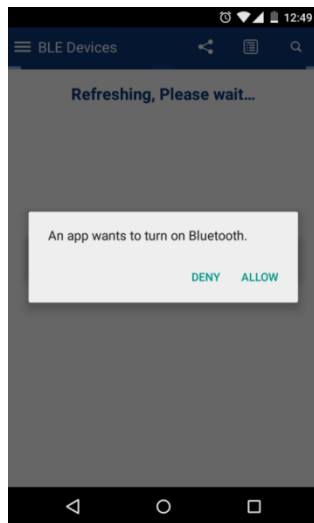


## 5.11 使用 CySmart 手机应用进行测试

1. 从您的使能的 BLE 手机端下载 CySmart 手机应用。对于 iOS 设备（iPhone 4S 或更高版本），可以从 [App Store](#) 中下载应用。对于 Android 设备（Android 4.3 或更新版本），可以从 [Play Store](#) 上下载应用。
2. 启动您手机上的应用。如果尚未打开您手机的蓝牙，应用将提示您打开它，如图 40 所示。

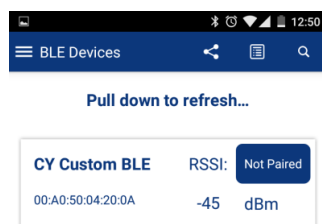
**注意：**这些 CySmart Android 应用的屏幕截图。对于 CySmart iOS 应用的外观和体验可能不太一样。

图 40. 打开手机蓝牙



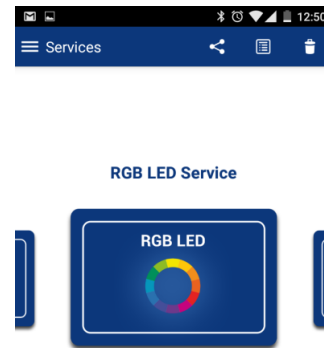
3. 打开蓝牙后，会显示设备屏幕。向下滑动，以列出附近所有 BLE 器件，包括 PSoC 4 BLE 自定义服务项目“CY Custom BLE”（图 41）。

图 41. 所列出的 BLE 器件



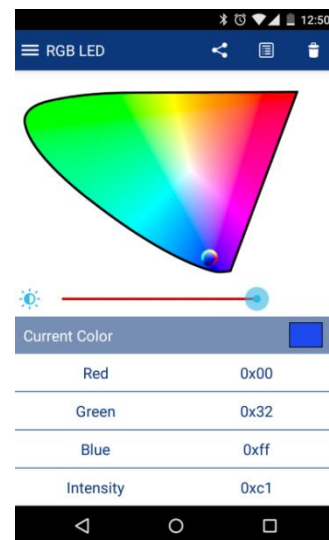
4. 选择 **CY Custom BLE** 器件。应启动连接程序并连接器件。屏幕将显示已连接器件所支持的配置文件/服务页面（请参考图 42）。

图 42. 服务页面



5. 如果屏幕中没有显示 RGB LED 图标，请向左或向右滑动。出现 RGB LED 图标时，请选择它。
6. 在 RGB LED GUI 屏幕上，色域（请参考图 43）控制着红色、绿色和蓝色组件的值，而线性滑条控制着灯光强度。使用滑条增加光线强度，然后滑动色域，以查找在 BLE Pioneer 套件的 RGB LED 上需要设置的相同颜色。

图 43. RGB LED 颜色控制



7. 要想断开器件的连接，请连续点击应用上的 **Back**（返回）按键，直到返回到 **Device Search**（器件搜索）页面为止。

## 5.12 使用 CySmart 中央仿真工具进行测试

CySmart 中央仿真工具和 BLE Dongle 仿真某个 BLE GATT 客户端器件。通过该操作，您可以连接至任何 BLE 器件，检测它们的属性，并通过这些属性将数据发送给某个外设器件。您可以从 [www.cypress.com/cysmart](http://www.cypress.com/cysmart) 网站上下载 CySmart 中央仿真工具的最新版本，并可以从 [www.cypress.com/CY8CKIT-042-BLE](http://www.cypress.com/CY8CKIT-042-BLE) 网站上下载固件 HEX 文件的最新版本。

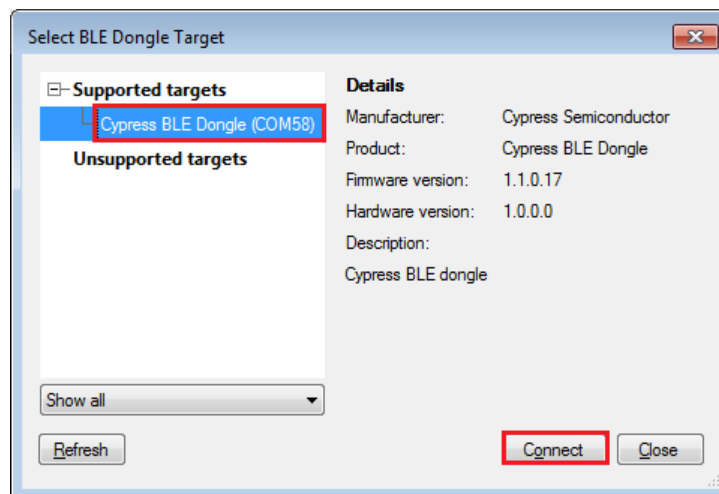
**注意：**目前只有 Windows PC 支持 CySmart 中央仿真工具。

要想使用 CySmart 中央仿真工具测试项目，请按照下面各步骤进行操作：

1. 将 BLE Dongle 连接到您的 PC 上。这样可以完成 USB 枚举操作。
2. 请依次选择 **Start > All Programs > Cypress > CySmart <version> > CySmart <version>** 来启动 CySmart 工具。
3. 在 CySmart 中央仿真工具上，请选择 **Supported targets** 列表下方的 **Cypress BLE Dongle**，并点击 **Connect**，如图 44 所示。

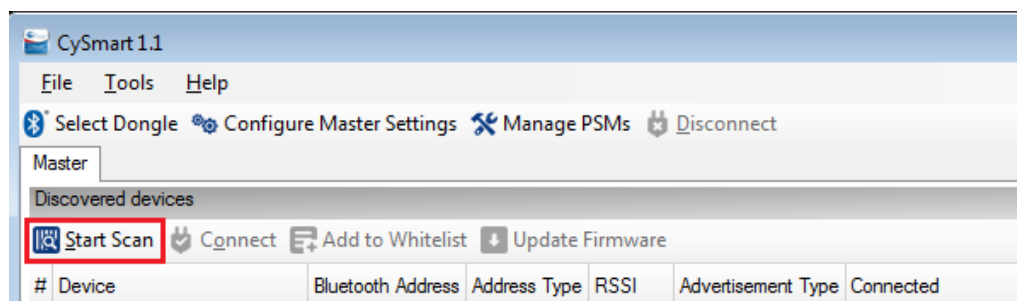
**注意：**如果未列出 BLE Dongle，请按下 BLE Dongle 上的复位按键，然后点击 **Refresh**。

图 44. 选择 BLE Dongle 目标器件



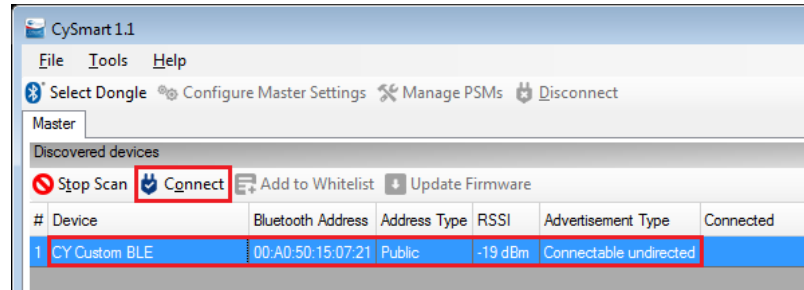
4. 选择 BLE Dongle 后，请点击窗口左上方的 **Start Scan**，开始扫描 BLE 外设器件，如图 45 所示。

图 45. 使用 CySmart 中央仿真工具开始扫描



- 选择名称为 **CY Custom BLE** 的器件，然后点击 **Connect**，如图 46 所示。

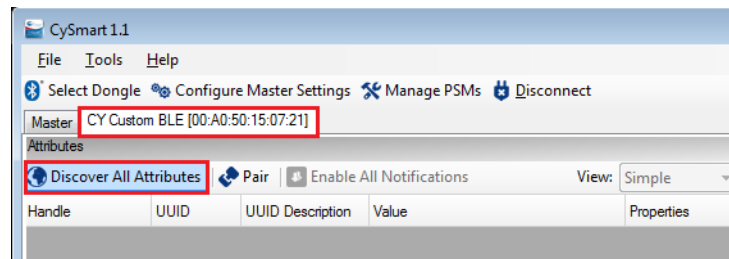
图 46. 连接至 BLE\_Custom 器件



器件连接后，将在 CySmart 中央仿真工具上的 ‘Master’ 选项卡旁边出现一个新选项卡，它的名称就是它所连接的器件的名称，如图 47 所示。

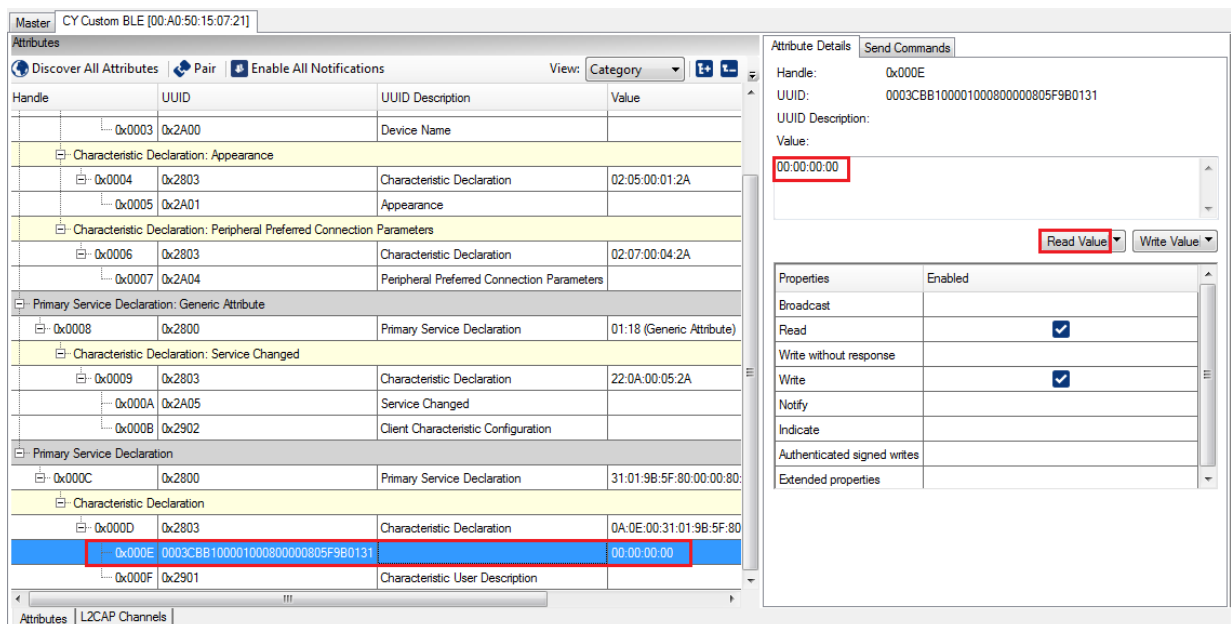
- 选择 **Discover All Attributes** 项，以启动 CySmart 工具用于查询 **CY Custom BLE** 器件所支持的属性，如图 47 所示。

图 47. ‘Discover All Attributes’ 项



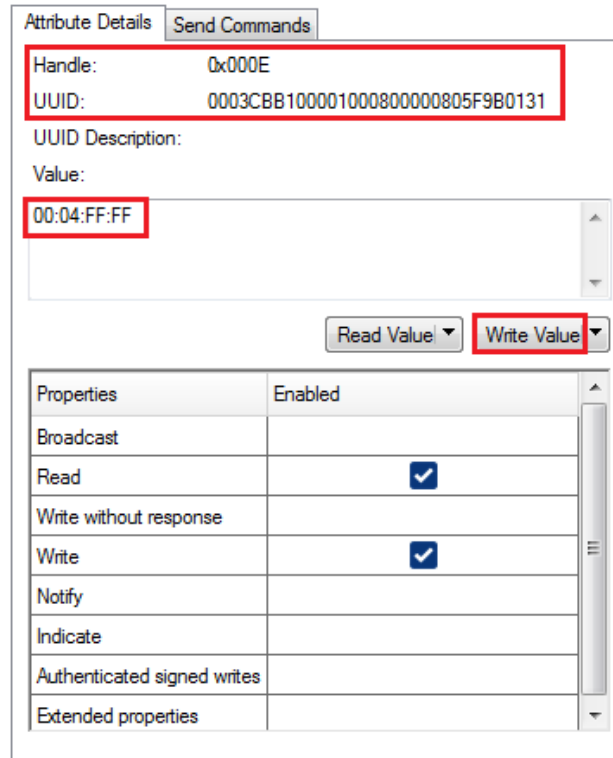
向下滚动属性列表，并点击 RGB LED 自定义特性（UUID **0003CBB1-0000-1000-8000-00805F9B0131**）。该特性支持读和写操作，如 CySmart 窗口的左侧部分上的属性信息所示。点击 **Read Value**，以读取现有的颜色值，如图 48 所示。值字段将显示一个 4 字节的数值。

图 48. RGB LED 自定义特性



- 将 4 个字节大小的非零值写入到 **Value**（值）字段中，并点击 **Write Value**（写入值）来传送新的颜色值，如图 49 所示。4 字节数值的格式是 **Red:Green:Blue:Intensity**（红色：蓝色：绿色：强度），其中 ‘0’ 是最低值，‘FF’ 是最高值。

图 49. 将新颜色值写入到自定义特性内



Attribute Details Send Commands

Handle: 0x000E

UUID: 0003CBB100001000800000805F9B0131

UUID Description:

Value: 00:04:FF:FF

Read Value Write Value

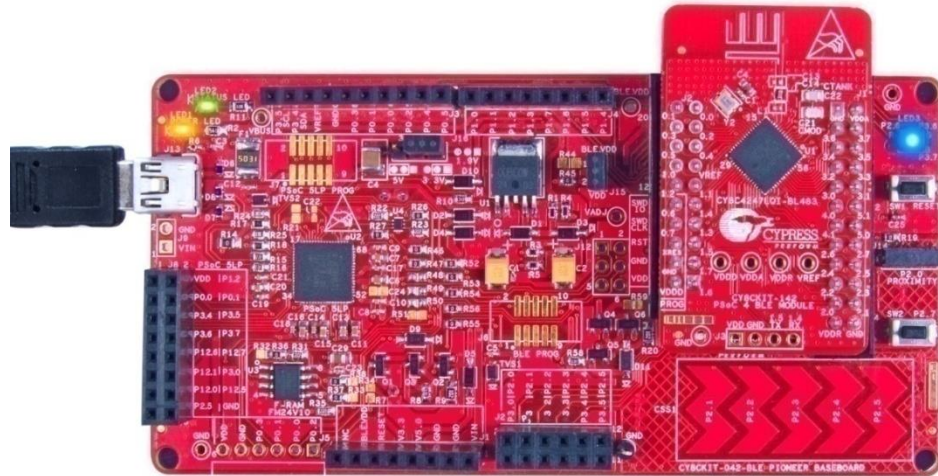
Properties	Enabled
Broadcast	
Read	<input checked="" type="checkbox"/>
Write without response	
Write	<input checked="" type="checkbox"/>
Notify	
Indicate	
Authenticated signed writes	
Extended properties	

发送其他任意 4 字节的数据，并观察相应的颜色。

RGB 数据	观察到的颜色
00:00:00:00	无颜色
FF:00:00:FF	全部为红色
00:FF:00:FF	全部为绿色
00:00:FF:FF	全部为蓝色
FF:00:FF:22	紫色，低亮度
FF:FF:00:55	黄色，中等亮度

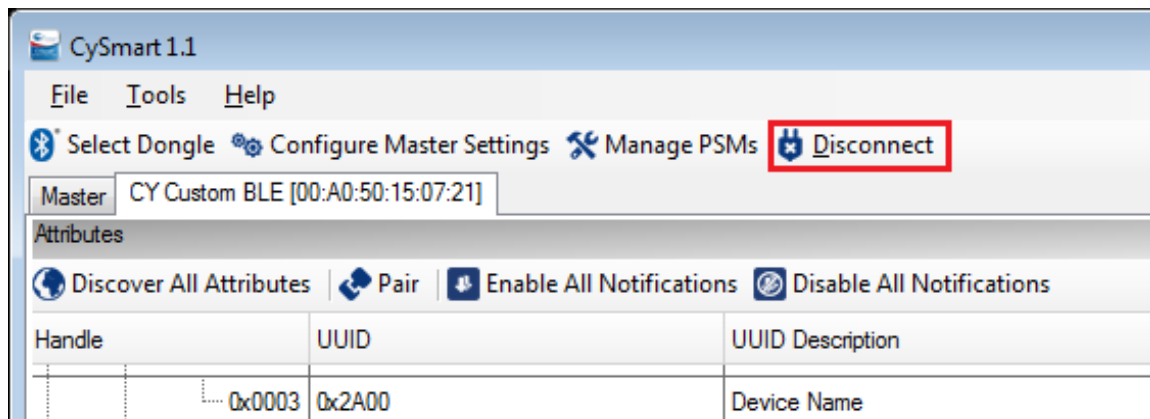
- 观察 BLE Pioneer 套件中 RGB LED 的新颜色，如图 50 所示。

图 50. BLE Pioneer 套件上的 RGB LED 控制



9. 要想断开器件连接，请点击 **Disconnect**，如图 51 所示。

图 51. 断开器件连接



## 6 总结

本应用笔记描述了通过使用 BLE 组件，配置各种服务并对 BLE GAT 客户端器件进行读写操作，从而实现将自定义 BLE 服务添加到 PSoC 4 BLE 项目的流程。可以轻松将此处所描述的方法应用于 PSoC 4 BLE 项目中 BLE 自定义服务的任何类型和数量。

## 7 相关信息

- [AN91267 — PSoC 4 BLE 入门](#)
- [AN91184 — PSoC 4 BLE: 设计 BLE 应用](#)
- [CY8CKIT-042-BLE Pioneer 套件](#)
- [BLE 开发者门户网站](#)
- [CySmart iOS 应用](#)
- [CySmart Android 应用](#)

---

## 关于作者

姓名: Rohit Kumar  
职务: 高级应用工程师

## A 附录

### A.1 发送通知

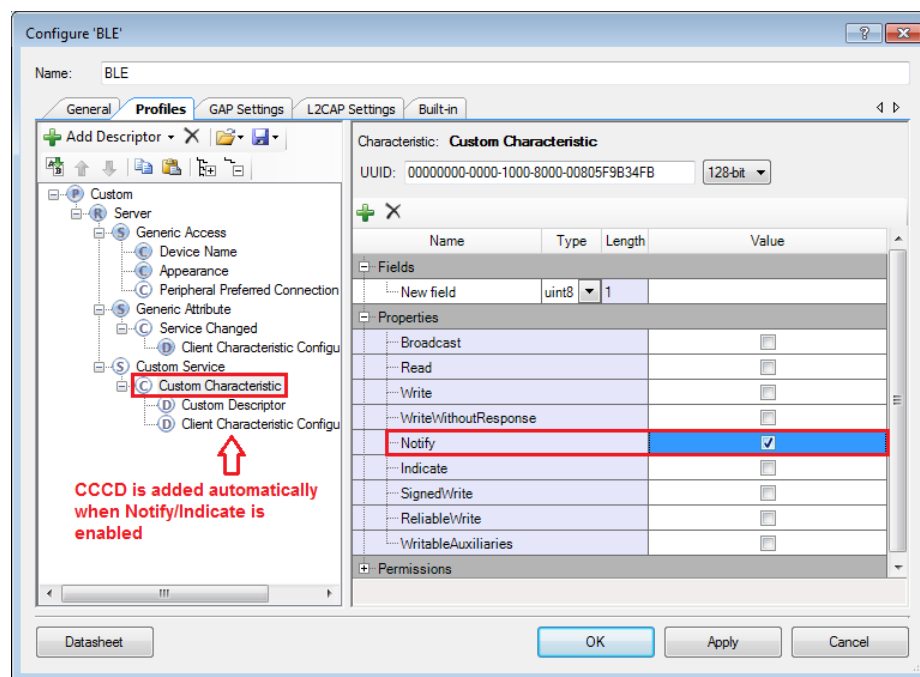
除了对特性进行读写操作外，通常还需要进行通知访问。通过使用这些通知，GATT 服务器可以将新数据发送到 GATT 客户端，而 GATT 客户端则无需连续轮询它。

支持这些通知的所有特性都有一个相关的描述符，即客户端特性配置描述符（CCCD）。GATT 客户端通过写入该 CCCD 可以启用和禁用 GATT 服务器上的通知。GATT 客户端启用 GATT 服务器上的通知后，GATT 服务器才能通过通知来发送数据。

为了能够对自定义特性进行通知访问并将数据发送给 GATT 客户端器件，请在您的项目中执行以下步骤。对于 Indicate（显示）支持，也要进行同样的流程：

1. 在 BLE 组件配置窗口中，选择通知被启用的特性。然后勾选 **Notify** 复选框，如图 52 所示。客户端特性配置描述符被自动添加到特性下方的属性列表中。点击 **OK**。

图 52. 选择组件的通知访问



2. 要想启用 GATT 服务器上的通知，GATT 客户端会将 **0x0001** 值写入 CCCD 内。发生 ‘CYBLE\_EVT\_GATTS\_WRITE\_REQ’ 事件时，要执行以下操作：
  - a. 检查写请求是否针对 CCCD 的属性句柄。
  - b. 如果是，请检查所发送的值是否只有最低两位中的一位，并且另一位没有被置位。只能将这些位作为 CCCD 上写请求的一部分进行发送。
  - c. 如果不是，则将记录 GATT 服务器内的 CCCD 值。
  - d. 根据 CCCD 的写入操作是否成功，将写响应或错误响应回送给客户端。

```

case CYBLE_EVT_GATTS_WRITE_REQ:

wrReqParam = (CYBLE_GATTS_WRITE_REQ_PARAM_T *) eventParam;

/* Check if the returned handle is matching to CCCD attribute */
if(CYBLE_CUSTOM_CLIENT_CHARACTERISTIC_CONFIGURATION_DESC_HANDLE ==
    wrReqParam->handleValPair.attrHandle)
{
    /* Only the first and second lowest significant bit can be
    * set when writing on CCCD. If any other bit is set, then
    * send error code */
    if(FALSE ==
        (wrReqParam->handleValPair.value.val
        [CYBLE_CUSTOM_CLIENT_CHARACTERISTIC_CONFIGURATION_DESC_INDEX] &
        (~CCCD_VALID_BIT_MASK)))
    {
        /* Set flag for application to know status of notifications.
        * Only one byte is read as it contains the set value. */
        startNotification =
            wrReqParam->handleValPair.value.val
            [CYBLE_CUSTOM_CLIENT_CHARACTERISTIC_CONFIGURATION_DESC_INDEX];

        /* Update GATT DB with latest CCCD value */
        CyBle_GattsWriteAttributeValue(&wrReqParam->handleValPair,
            FALSE,
            &cyBle_connHandle,
            CYBLE_GATT_DB_PEER_INITIATED);
    }
    else
    {
        /* Send error response for Invalid PDU against Write
        * request */
        CYBLE_GATTS_ERR_PARAM_T err_param;

        err_param.opcode = CYBLE_GATT_WRITE_REQ;
        err_param.attrHandle = wrReqParam->handleValPair.attrHandle;
        err_param.errorCode = ERR_INVALID_PDU;

        /* Send Error Response */
        (void)CyBle_GattsErrorRsp(cyBle_connHandle, &err_param);

        /* Return to main loop */
        return;
    }
}

/* Send response to the Write request */
CyBle_GattsWriteRsp(connectionHandle);
break;

```

根据“BLE 内核规范中第 3 卷 F 部分的第 3.4.1 节”，错误代码 ‘ERR\_INVALID\_PDU’ 的值为 0x04。

在您应用代码中进行如下定义：

```

/*****GATT Error code*****/
#define ERR_INVALID_PDU                0x04
#define CCCD_VALID_BIT_MASK           0x03
#define NOTIFY_BIT_MASK                0x01

```

- 在主应用中，每当数据可用并且 GATT 客户端启用了通知时，可以通过通知来发送数据。



```
/* 'notificationHandle' is handle to store notification data parameters */
CYBLE_GATTS_HANDLE_VALUE_NTF_T notificationHandle;

/* Check if the notification bit is set or not */
if(startNotification & NOTIFY_BIT_MASK)
{
    /* Update Notification handle with new data*/
    notificationHandle.attrHandle = CYBLE_CUSTOM_CHAR_HANDLE;
    notificationHandle.value.val = &data;
    notificationHandle.value.len = dataLength;

    /* Report data to BLE component for sending data by notifications*/
    CyBle_GattsNotification(connectionHandle, &notificationHandle);
}
```

4. 有关通过自定义配置文件实现 BLE 通知的实力项目的更多信息，请参考 CY8CKIT-042-BLE Pioneer 套件中的 CapSense\_Proximity 或 CapSense\_Slider\_and\_LED 项目。

## 文档修订记录

文档标题: AN91162 — 创建 BLE 的自定义配置文件

文档编号: 002-00011

版本	ECN	提交日期	变更说明
**	4984940	10/26/2015	本文档版本号为 Rev**, 译自英文版 001-91162 Rev*A。
*A	5691702	04/11/2017	更新徽标和版权。
*B	6651892	03/31/2020	本文档版本号为 Rev*B, 译自英文版 001-91162 Rev*D。

## 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、厂商代表和经销商组成的全球性网络。要想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	<a href="http://cypress.com/arm">cypress.com/arm</a>
汽车级产品	<a href="http://cypress.com/automotive">cypress.com/automotive</a>
时钟与缓冲器	<a href="http://cypress.com/clocks">cypress.com/clocks</a>
接口	<a href="http://cypress.com/interface">cypress.com/interface</a>
物联网	<a href="http://cypress.com/iot">cypress.com/iot</a>
存储器	<a href="http://cypress.com/memory">cypress.com/memory</a>
微控制器	<a href="http://cypress.com/mcu">cypress.com/mcu</a>
PSoC	<a href="http://cypress.com/psoc">cypress.com/psoc</a>
电源管理 IC	<a href="http://cypress.com/pmuc">cypress.com/pmuc</a>
触摸感应	<a href="http://cypress.com/touch">cypress.com/touch</a>
USB 控制器	<a href="http://cypress.com/usb">cypress.com/usb</a>
无线连接	<a href="http://cypress.com/wireless">cypress.com/wireless</a>

### 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](#) | [代码示例](#) | [项目](#) | [视频](#) | [博客](#) | [培训](#) | [组件](#)

### 技术支持

[cypress.com/support](http://cypress.com/support)

此处引用的所有其它商标或注册商标都归其各自所有者所有。



赛普拉斯半导体公司  
198 Champion Court  
San Jose, CA 95134-1709

© 赛普拉斯半导体公司，2015-2020 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归赛普拉斯所有。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。如果软件没有附带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适用于个人的、非独占性、不可转让的许可（无转授许可权）（1）在版权保护下的软件（a）以源代码形式提供的软件，只能是在组织内部为了使用赛普拉斯的硬件去修改和复制。（b）以二进制代码形式从外部发到终端用户（直接或间接通过经销商和分销商），仅用于赛普拉斯硬件产品单元。（2）在软件（由赛普拉斯公司提供，且未经修改）侵犯赛普拉斯专利的权利主张下，仅许可在赛普拉斯硬件产品上制造、使用、提供和导入软件。禁止对软件的任何其他使用、复制、修改、翻译或编译。

赛普拉斯不对此材料提供任何类型的明示或暗示保证，包括但不限于针对特定用途的适销性和适用性的暗示保证。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安全问题及安全漏洞的责任，例如未经授权的使用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。在适用法律允许的范围内，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。文件使用人应负责正确设计、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。贵方应赔偿并保护赛普拉斯免受所有索赔的损害，包括因人身伤害或死亡引起的索赔、费用、损失和其它责任。

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](http://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。