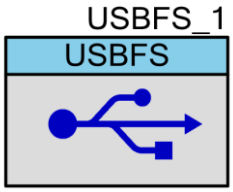
# 全速 USB（USBFS）

2.70




## 特性

- USB 全速器件接口驱动程序
- 支持中断传输、控制传输、批量传输和同步传输类型
- 对描述符集选择的运行时支持
- 可选的 USB 字符串描述符
- 支持可选的 USB HID 类
- 支持可选的 Bootloader
- 支持可选的音频类（请参见 USBFS 音频部分）
- 支持可选的 MIDI 器件（请参见 USBFS MIDI 部分）
- 支持可选的 CDC 类（请参见 USBUART 部分）

## 概述

USBFS 组件提供与第 9 章兼容的 USB 全速器件框架。该组件为 USB 主机发出的请求进行解码和调度的控制端点提供了低层驱动程序。另外，该组件提供 USBFS 自定义程序以轻松实现描述符的构建。您可以选择构建基于 HID 的器件或普通 USB 器件。通过设置配置/接口描述符，选择 HID（和 HID 与泛型之间的切换）。

有关描述符的其他信息，请参考 USB-IF 器件类文档 （http://www.usb.org/developers/devclass/）。

**注意**：赛普拉斯免费提供一组与赛普拉斯芯片一同使用的 USB 开发工具（称为 SuiteUSB）。可以从赛普拉斯网站获取 SuiteUSB：http://www.cypress.com。

### 何时使用 USBFS

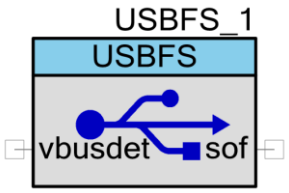
当您要提供具有兼容 USB 2.0 器件接口的应用程序时，使用 USBFS 组件。

### 快速启动

1. 将 USBFS 组件从组件目录拖动到您的设计中。
2. 请留意 Notice List（通知列表）窗口中的时钟错误；双击某个错误可打开系统时钟编辑器。
3. 配置下列时钟：
   a) **ILO**：选择 100 kHz。
   b) **IMO**：选择振荡器 24.000 MHz。
   c) **USB**：使能并选择 IMOx2 – 48.000 MHz。
4. 选择 **Build** 以生成 API。

## 输入/输出连接

本节介绍了 USBFS 组件的输入和输出连接。I/O 列表中的星号（*）表示，在 I/O 说明中列出的情况下，该 I/O 可能不可见。




### sof – 输出 *

使用帧起始（sof）输出，端点可识别帧起始，可以将内部端点时钟与主机进行同步化。在自定义程序的 **Advanced** 选项卡下，如果选中了 **Enable SOF Output** 参数，则此输出可见。

### vbusdet – 输入*

vbusdet 输入提供了连接至 VBUS 的功能，用于监控 VBUS 电源。在自定义程序的 **Advanced** 选项卡下，如果选中了 **Enable VBUS Monitoring** 和 **External VBUS** 参数，则此输入可见。

# 组件参数

将 USBFS 组件拖放到您的设计中，双击它以打开 **Configure USBFS** 对话框。

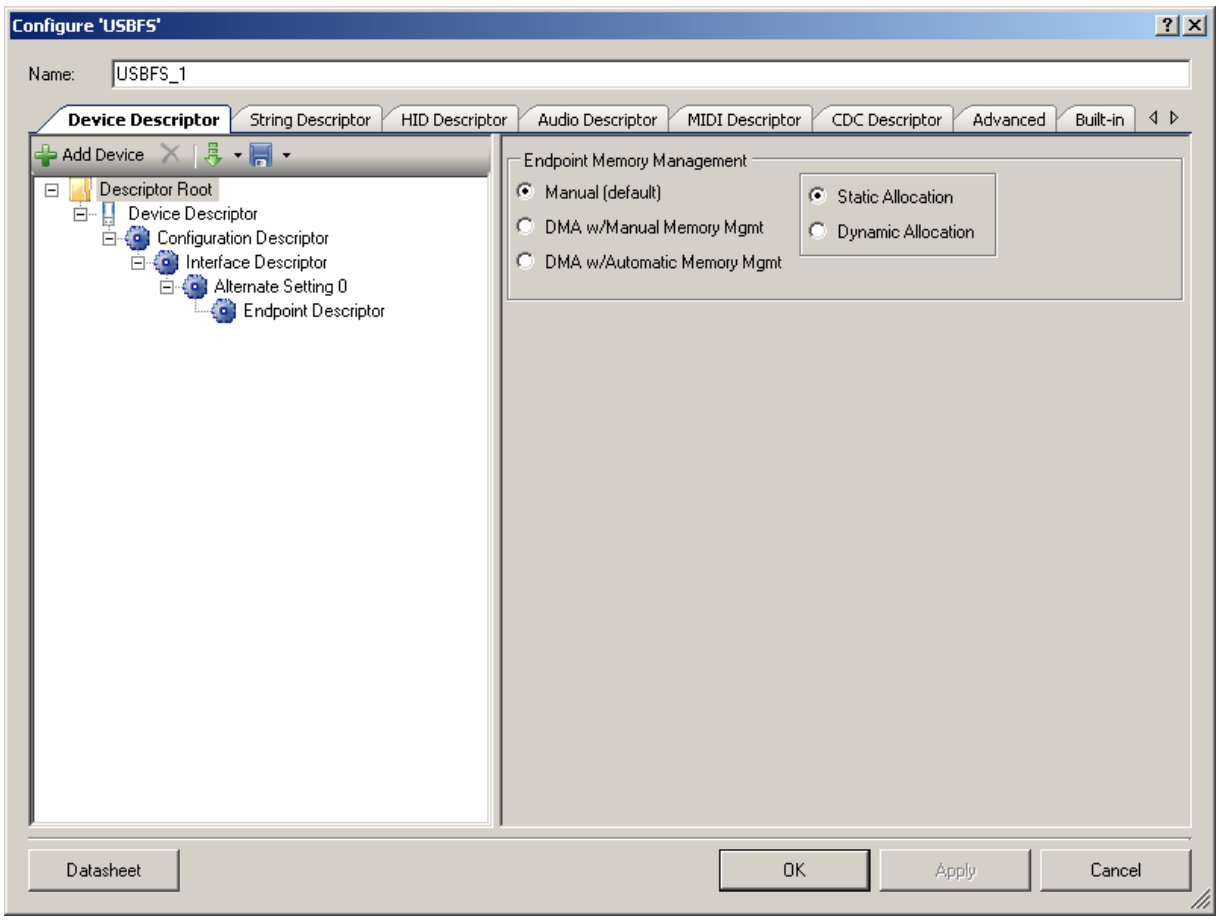
组件由“USBFS Configure”对话框中生成的信息来驱动的。此对话框（Dialog）或自定义程序（Customizer）简化了 USB 描述符的构建，并将生成的信息集成到用于器件枚举的驱动程序固件中。

如果不首先运行向导并选择相应的属性来描述您的器件，USBFS 组件将无法正常工作。代码发生器采用了您的器件信息，并生成所有必需的 USB 描述符。

**Configure USBFS** 对话框包含下列选项卡及设置：

## Device Descriptor（器件描述符）选项卡

### Descriptor Root（根描述符）



### Endpoint Memory Management（端点存储器管理）

USBFS 模块包含目标存储器的 512 个字节，供数据端点使用。然而，该架构支持直通（Cut-through）操作模式（带有自动存储器管理的 DMA）。该模式可以根据系统性能降低对存储器的要求。

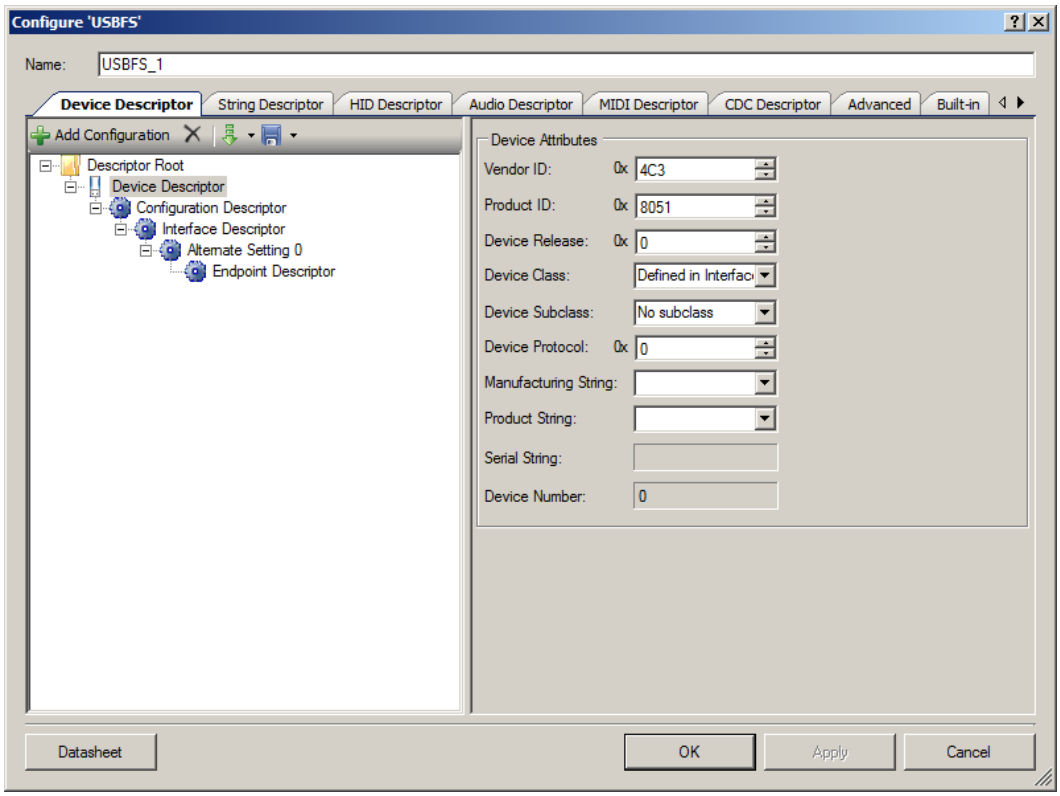
某些应用程序可以利用直接存储器访问（DMA）将数据移进和移出端点存储器缓冲区。

- **Manual**（手动）（默认选择）— 选择此选项可使用 LoadInEP/ReadOutEP 加载和卸载端点缓冲区。
  - **Static Allocation**（静态分配）— SET_CONFIGURATION 请求后立即分配端点的存储器。当多个备用设置使用相同的端点（EP）编号时，需要的时间最长。
  - **Dynamic Allocation**（动态分配）— 每个 SET_CONFIGURATION 和 SET_INTERFACE 请求后动态分配端点的存储器。当多个备用设置用于互斥 EP 设置时，此选项非常有用。
- **DMA w/Manual Memory Management**（带有手动存储器管理的 DMA）— 为手动 DMA 数据操作选择此选项。LoadInEP/ReadOutEP 函数完全支持这种模式，并自动初始化 DMA。
- **DMA w/Automatic Memory Management**（带有自动存储器管理的 DMA）— 为自动 DMA 数据操作选择此选项。这是支持占用量超过 512 个字节的组合数据端点的唯一配置。LoadInEP/ReadOutEP 函数适用于初始的 DMA 配置。

PSoC 不支持 USB 端点与其他外设之间的直接 DMA 数据操作。所有涉及 USB 端点的 DMA 数据操作（输入和输出）必须随主系统存储器终止或启动。

某些应用程序需要 USB 端点与其他外设之间的直接 DMA 数据操作，这时必须使用两个 DMA 数据操作。作为 USB 端点与其他外设之间的中间步骤，这两个数据操作将数据移动到主系统存储器中。

## Device Descriptor（器件描述符）




## Device Attributes（器件属性）

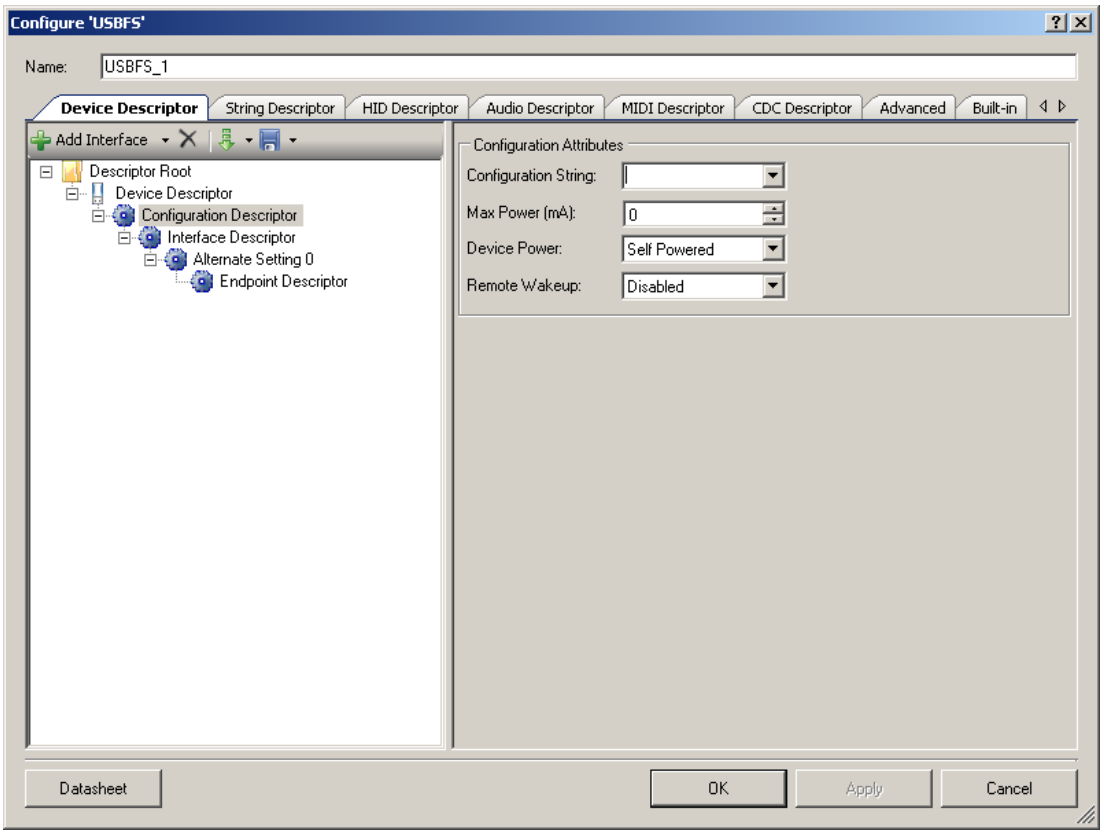
- **Vendor ID**（供货商 ID） — 贵公司 USB 供货商 ID（从 USB-IF 获取）。

  **注意：**供货商 ID 0x4B4 是仅赛普拉斯使用的 VID，并只能用于开发目的。使用此 VID 不能发布产品；您必须获取自己的 VID。

- **Product ID**（产品 ID） — 您的特定产品 ID。
- **Device Release**（器件版本） — 您的特定器件版本（器件 ID）。
- **Device Class**（器件类别） — 器件类别是在**接口描述符**、 **CDC** 或**特定于供货商**中定义的。
- **Device Subclass**（器件子类别） — 取决于 **Device Class**。
- **Device Protocol**（器件协议）
- **Manufacturing String**（制造商字符串） —当连接器件时显示的特定于制造商的说明字符串。
- **Product String**（产品字符串） — 当连接器件时显示的特定于产品的说明字符串。
- **Serial String**（序列字符串）

- **Device Number**（器件编号） — 各器件阵列中的器件索引编号。

## Configuration Descriptor（配置描述符）




### 配置属性

- **Configuration string**（配置字符串）
- **Max Power（mA）**（最大功耗） — 在此特定的配置中，当器件全速运行时，总线上 USB 器件的最大功耗。

  **注意：Device Power**（器件功耗）参数报告了配置是采用总线供电还是自供电。器件状态报告器件当前是否采用自供电。如果器件断开与其外部电源的连接，它将更新器件状态以指示它不再采用自供电。当器件失去其外部电源时，不能将从总线得到的电源增加到超过其（Max power）配置报告的值。
- **Device Power**（器件供电） — **Bus Powered**（总线供电）或者 **Self Powered**（自供电）器件。USBFS 不支持同时设置这两种供电。
- **Remote Wakeup**（远程唤醒） — **Enabled**（使能）或 **Disabled**（禁用）。

### Interface Association Descriptor （接口关联描述符）

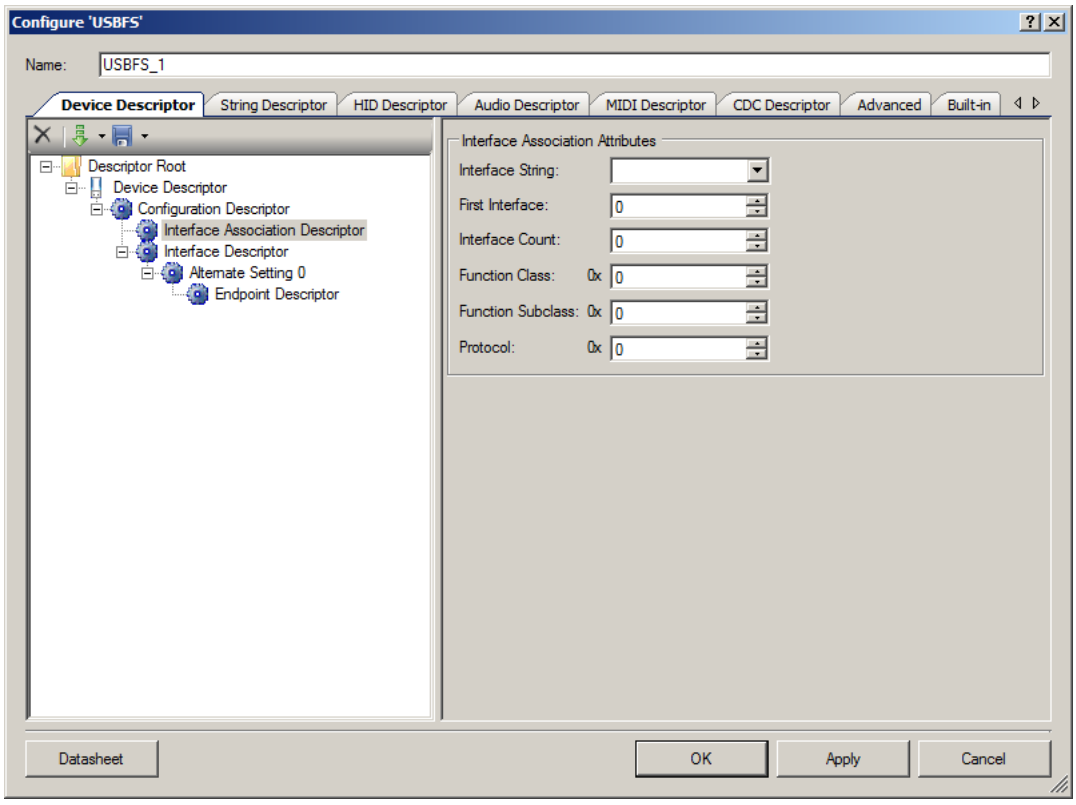
使用接口关联描述符（IAD）将多功能器件中的多个接口组合成一个逻辑器件功能。

使用 IAD 的器件必须使用下表中定义的器件类别、器件子类别以及器件协议等代码。这些类代码被定义为*多接口功能器件类代码*。

| 器件属性 | 数值 | 说明 |
|---|---|---|
| 器件类别 | 0xEF | 各种器件类别 |
| 器件子类别 | 0x02 | 普通类别 |
| 器件协议 | 0x01 | 接口关联描述符 |

**要想添加接口关联描述符，请执行下面各操作：**

1. 选择 **Descriptor Root** 目录下的 **Configuration Descriptor** 项。
2. 单击 **Add Interface** 工具按键，选择 **Association**。



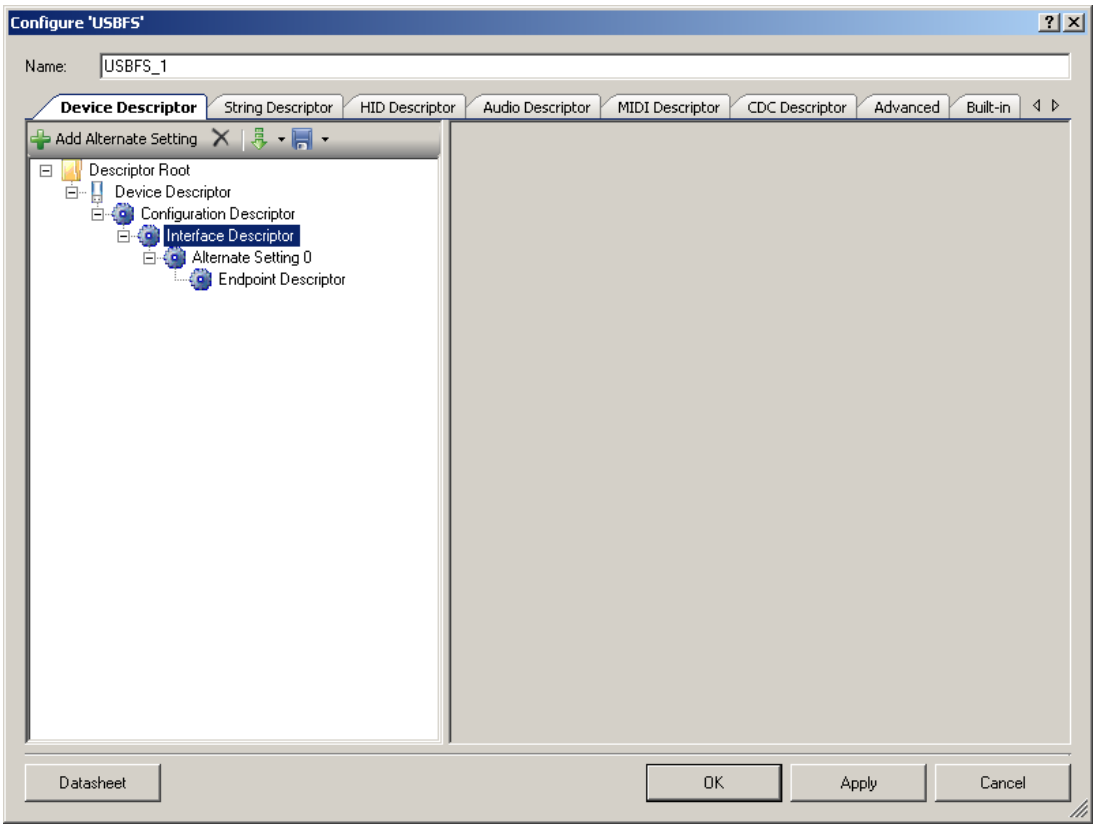

### 接口关联属性

- **Interface String**（接口字符串） — 该功能中所描述的字符串描述符的索引。

- **First Interface**（第一个接口） — 与此功能相关的第一接口的接口编号。
- **Interface Count** （接口数量） — 与此功能相关的连续接口的数量。
- **Function Class**（函数类别） — 类代码。它通常与第一个关联接口的类型值相同。
- **Function Subclass**（接口子类别） — 子类型代码。
- **Protocol** （协议）— 协议代码 。

### 接口描述符

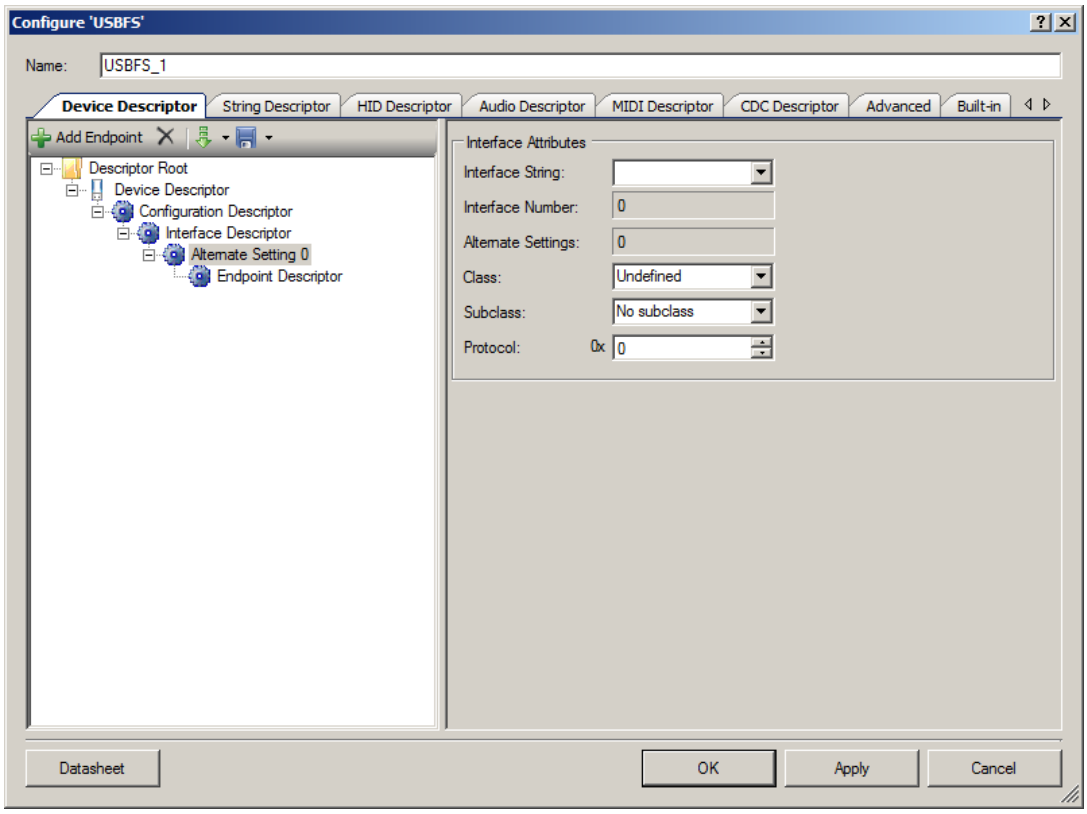
此项用于添加和删除接口备用设置。接口在 Alternate Setting（备用设置）中进行配置。




自动提供 **Alternate Setting 0** 以配置您的器件。如果器件使用同步端点（ISO endpoint），则 USB 2.0 规范要求器件默认接口设置不包含任何具有非零数据有效负载大小的同步端点（ISO endpoint）。这是通过 **Endpoint Descriptor** 中的 **Max Packet Size** 指定的。

对于同步器件，应当使用备用接口设置（而不是默认备用设置 0）来为同步端点指定非零数据的有效负载大小。另外，如果同步端点具有较大的数据有效负载大小，应使用其他备用配置或接口设置来指定数据有效负载大小范围。这可以增加该器件与其他 USB 器件成功组合使用的机会。

## 接口描述符 — 备用设置




### 接口属性

- **Interface String**（接口字符串）
- **Interface Number**（接口编号） — 通过定制器计算。
- **Alternate Settings**（备用设置） — 通过定制器计算。
- **Class**（类别） — **HID**、**Vendor Specific**（特定于供货商）或 **Undefined**（未定义）。
- **Subclass**（子类别）— 依赖于选定的类别。
- **Protocol**（协议）

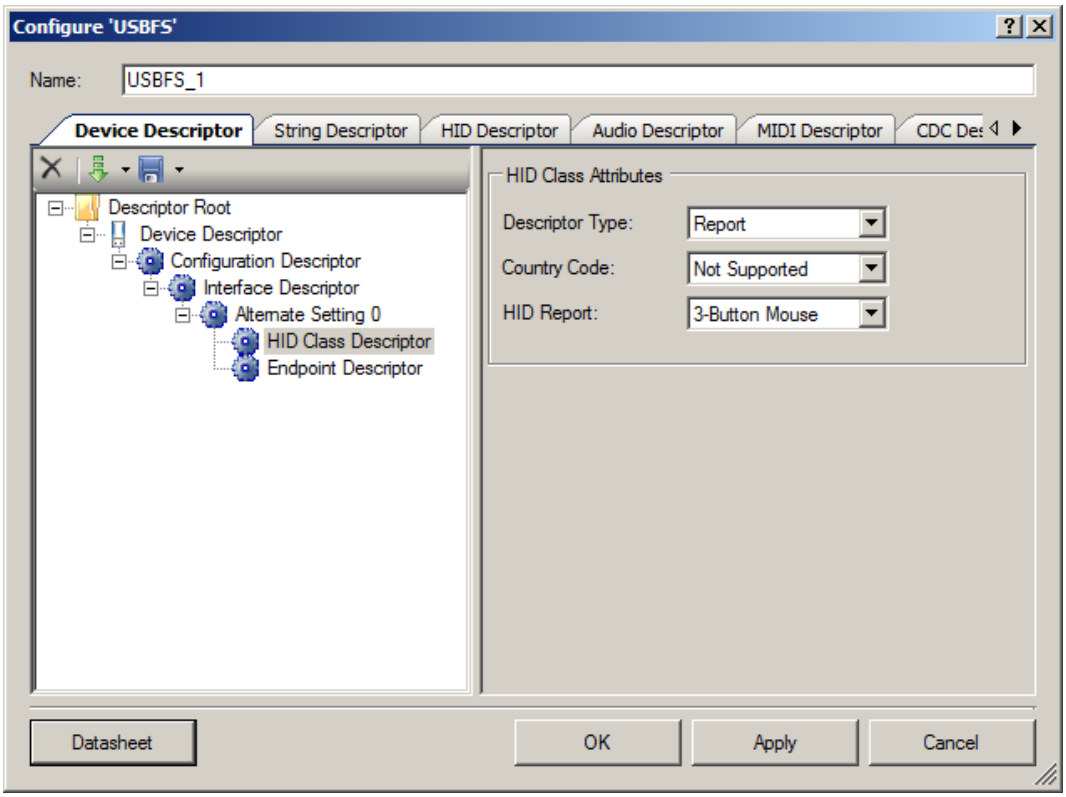
**注意：**字符串描述符是可选的。如果器件不支持字符串描述符，则器件、配置以及接口描述符中对字符串描述符的所有引用必须设置为零。

### HID 类描述符

默认情况下，不显示“HID 类描述符”项。它用于向“备用设置”中添加“HID 报告”。

**要添加“HID 类描述符”：**

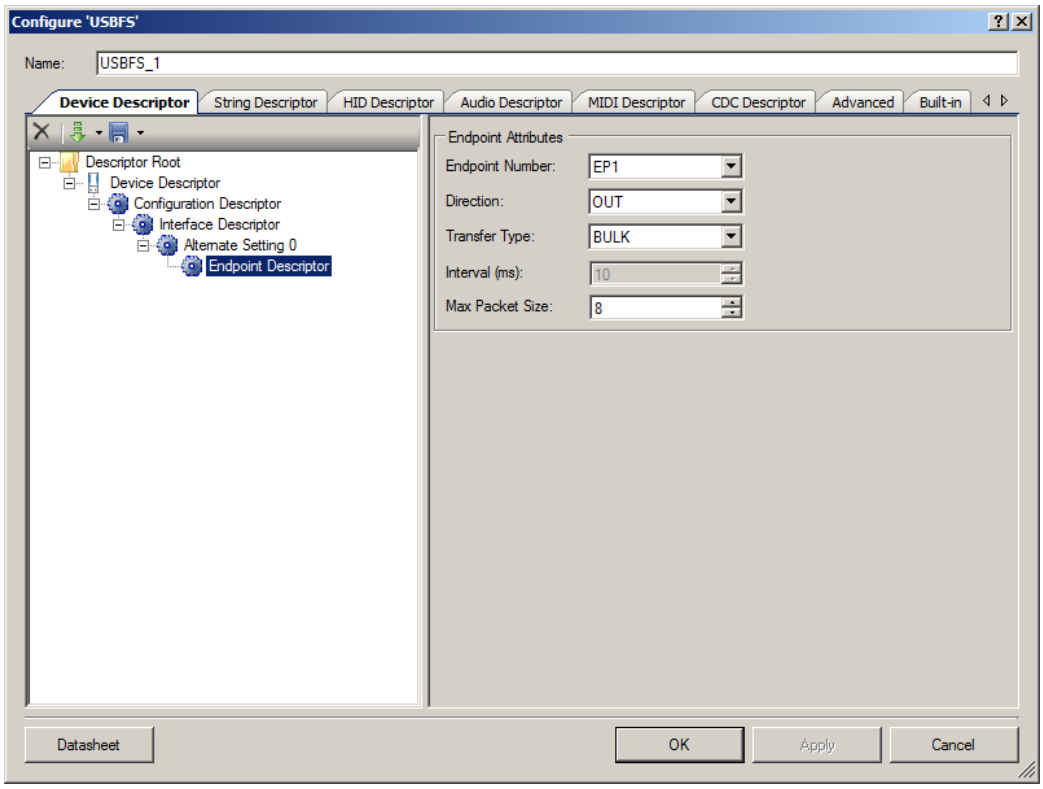
1. 选择 **Descriptor Root** 目录中的 **Alternate Setting** 项。
2. 在右侧的 **Interface Attributes** 下，为 **Class** 字段选择 **HID**。




### HID 类属性

- **Descriptor Type**（描述符类别） — 标识类描述符类别的常量名称。
- **Country Code**（国家代码） — 标识本地化硬件的国家/地区代码的数字表达式。
- **HID Report**（HID 报告） — 可用报告描述符列表。报告描述符取自 **HID Descriptor** 选项卡。此字段为必填字段。

## Endpoint Descriptor （端点描述符）




### 端点属性

- **Endpoint Number**（端点编号）
- **Direction**（方向） — 输入或输出。USB 传输是主机为中心；因此， **IN** 指的是传输到主机；**OUT** 指的是从主机输出。
- **Transfer Type**（传输类别） — 控制传输（**CONT**）、中断传输（**INT**）、批量传输（**BULK**）或同步数据传输（**ISOC**）
- **Interval** (ms)（间隔） — 特定于此端点的轮询间隔。全速端点可以指定从 1 ms 到 255 ms 的轮询周期。
- **Max Packet Size**（最大数据包大小）（字节） — 在全速器件中，对于批量或中断端点，**Max Packet Size** 为 64 个字节；对于同步端点，此数值为 512 个字节（处于自动 DMA 模式，此数值为 1023 个字节）。对于全速器件，批量端点只有 8 个字节、16 个字节、32 个字节并 64 个字节值是允许的。

  在操作的存储器管理手动模式下，同步端点的最大数据包大小受本地存储器大小的限制，但操作的带自动存储器管理的 DMA 模式不受此限制。这是因为本地存储器被视为临时缓冲区。

### Import 和 Save 工具按键

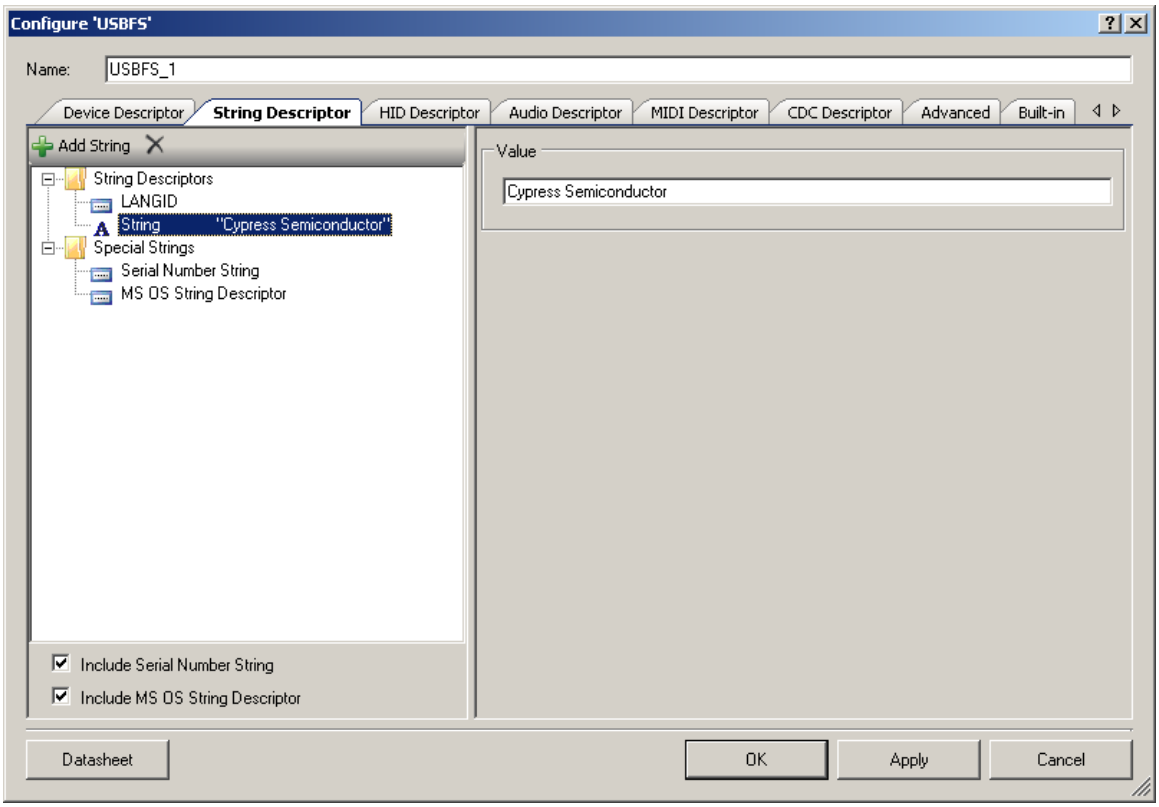
**Save** 按键允许将组件配置的信息保存到 XML 配置文件中。在下拉列表中，您可以选择 **Save Current Descriptor** 或 **Save Root Descriptor**。第一选项保存已选择的描述符的配置。第二选项保存整个器件描述符目录。

**Import** 按键允许您导入描述符配置。在下拉列表中，您可以选择 **Import Current Descriptor** 或 **Import Root Descriptor**。第一选项加载了选定描述符的配置。第二选项加载描述符的目录。在这种情况下，上一个配置描述符不被删除。

**注：**相同 **Import** 和 **Save** 工具按键在其他“描述符”选项卡中显示：**HID Descriptor**、**Audio Descriptor** 以及 **CDC Descriptor**。这些选项卡用于保存和导入在这些选项卡中配置的描述符配置。

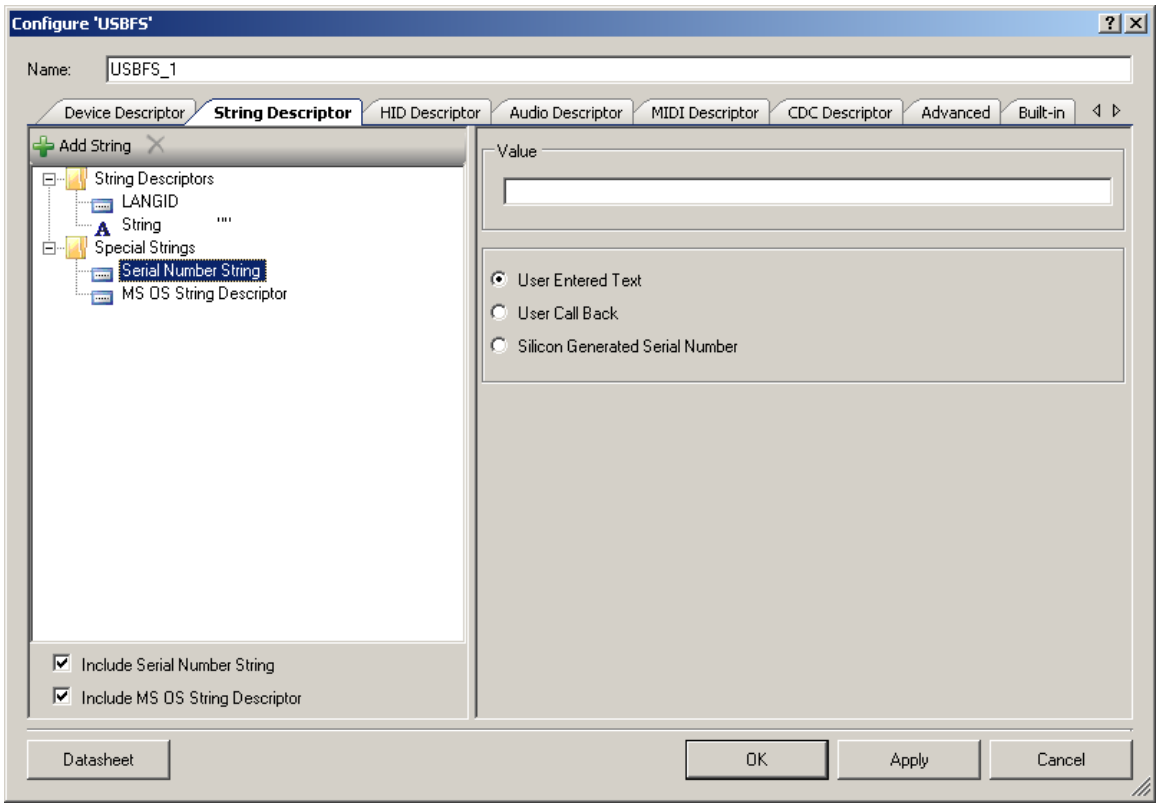
## String Descriptor（字符串描述符）选项卡

### String Descriptors 字符串描述符




- **LANGID** — 语言 ID 选择
- **String**（字符串） — 字符串描述符的值

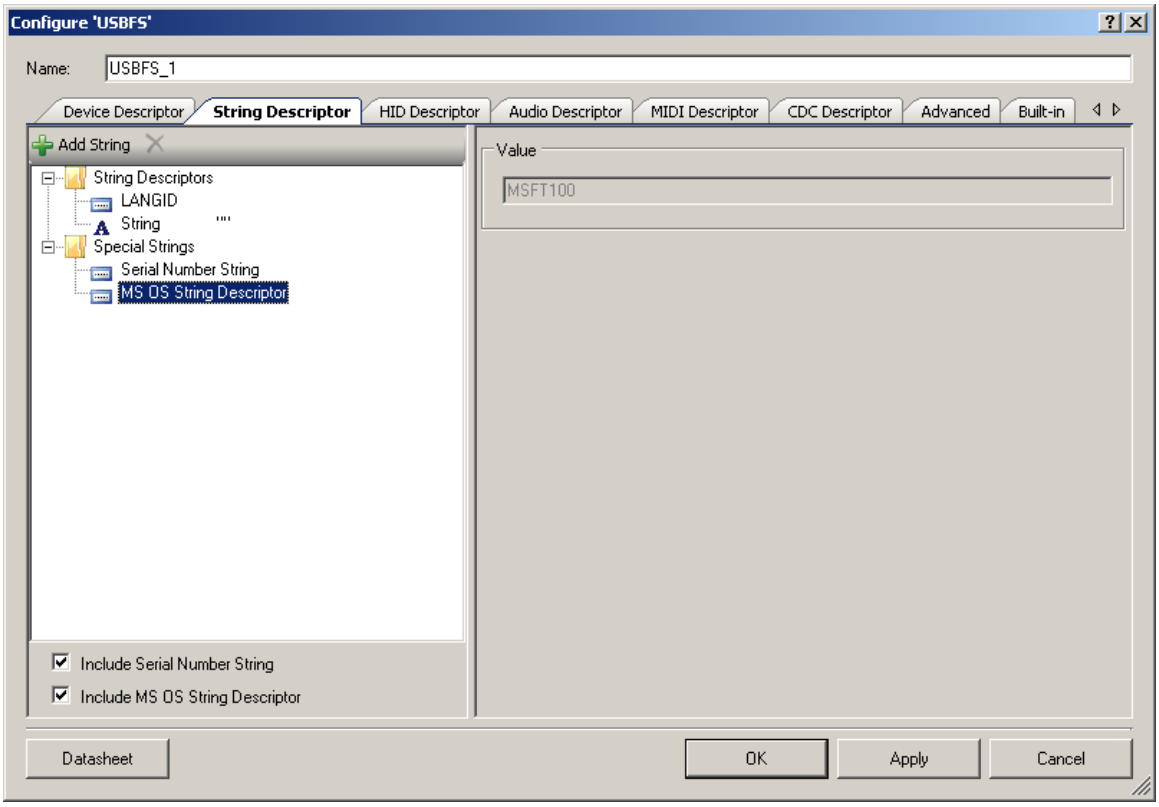
### 序列号字符串




- **Value**（值） — 默认字符串。
- **User Entered Text**（用户所输入的文本） — 使能 **Value** 文本框。
- **User Call Back**（用户回调） — USBFS_SerialNumString()函数设置指针以使用用户生成的序列号字符串描述符。应用固件可以在运行时间内提供 USB 器件描述符的序列号字符串的源。
- **Silicon Generated Serial Number**（芯片生成的序列号） — 该号码适用于在生产时器件内的非易失性存储器。不能保证它是唯一的。

### MS 操作系统字符串描述符

Microsoft 操作系统描述符为 USB 器件提供一种方法来向最新 Microsoft 操作系统提供其他配置信息。

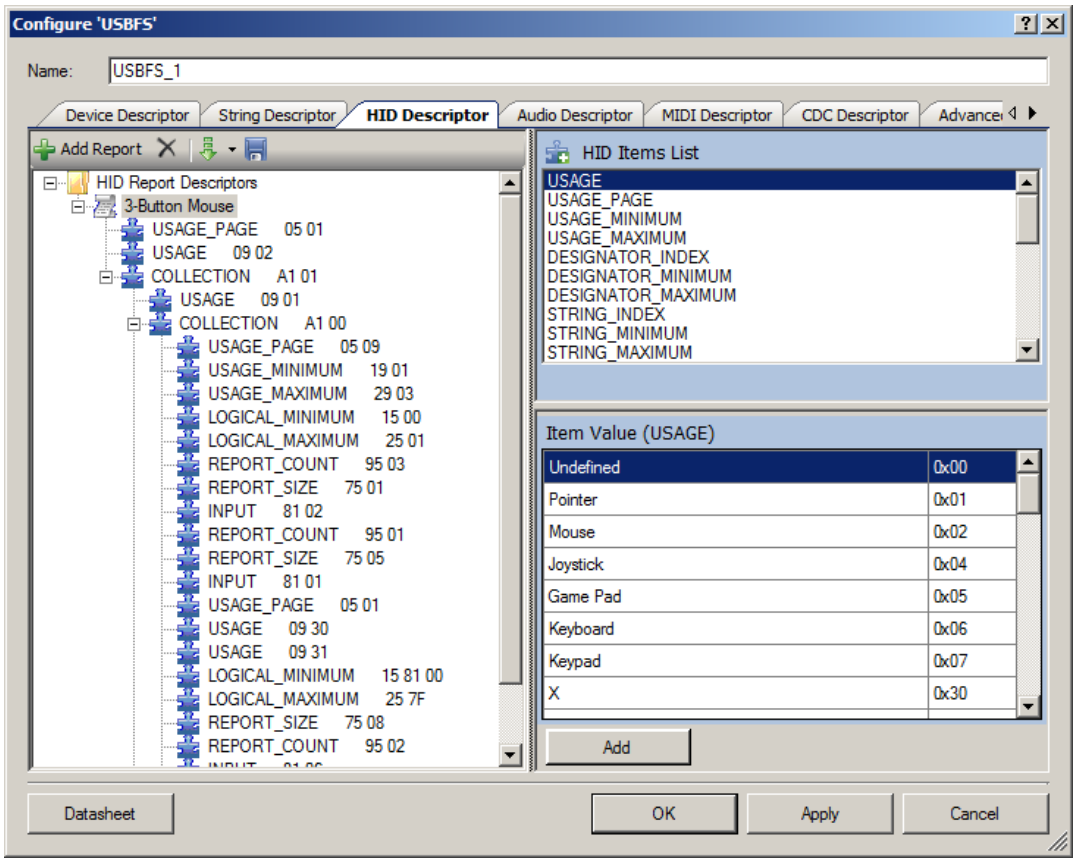



- **Value**（值） **—** 常量字符串 **MSFT100**

## HID Descriptor（HID 描述符）选项卡

**HID Descriptor** 选项卡可为器件快速构建 HID 描述符。使用 **Add Report** 按键以添加和配置 HID 报告描述符。

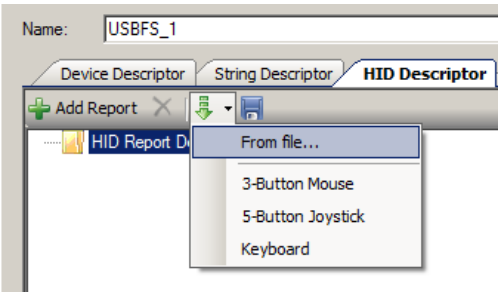
### HID 描述符




- **HID Items List**（HID 项目列表） — 要在 HID 报告中添加的项目。
- **Item Value**（项目值） — 在 **HID Items List** 或目录中选择的项目的值。

### 导入 HID 报告

**Import** 按键允许您导入 HID 报告。




您可以选择下拉列表中各个模板中的一个，也可以通过点击 **From file** 实现所需选择。模板选项立即加载已选定的 HID 报告。**From file** 选项将打开由 USBFS 组件创建的或来自 USB-IF HID 描述符工具的 HID 报告。更多有关 HID 描述符工具的信息，请参考 USB-IF 网站：http://www.usb.org/developers/hidpage#HID Descriptor Tool。

支持工具的版本 2.4。受支持的文件格式为“.hid”、“.h”以及“.dat”。根据源文件格式，您需要在“Open File”对话框中为文件选择合适的扩展名。

## Audio Descriptor（音频描述符）选项卡

**Audio Descriptor** 选项卡用于添加和配置音频接口描述符。欲了解更多信息，请参见 USBFS 音频中的 Audio Descriptor（音频描述符）选项卡部分所介绍的内容。

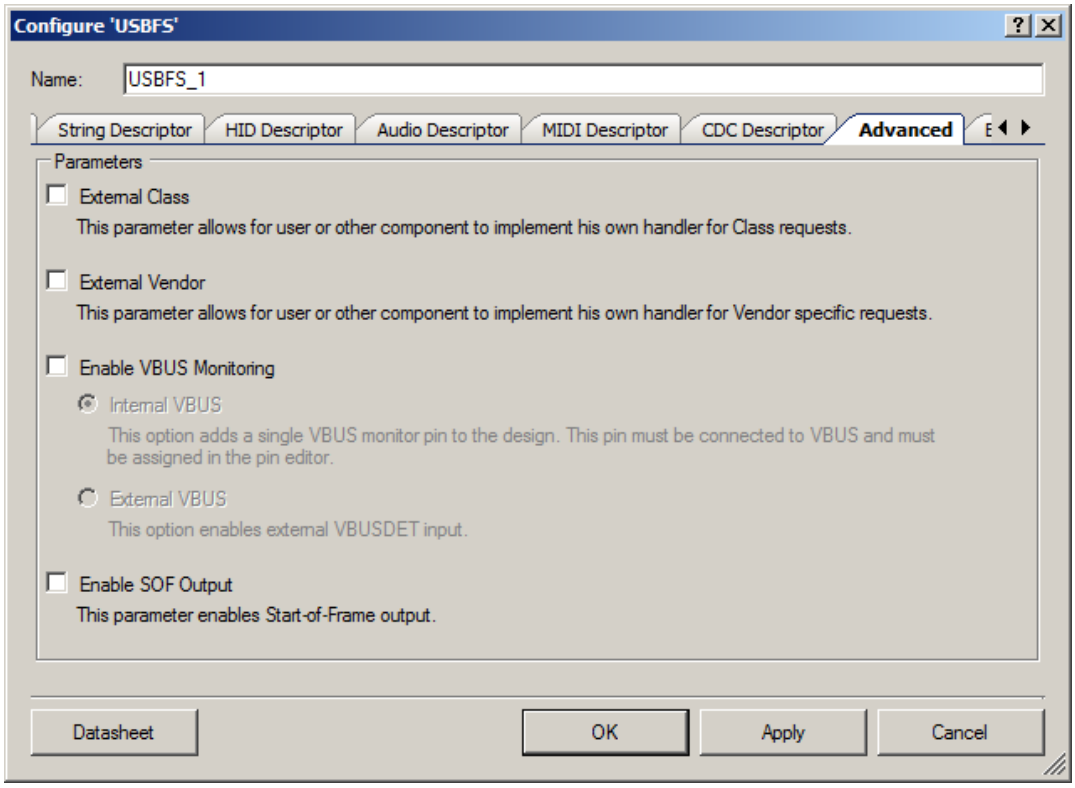
## MIDI Descriptor（MIDI 描述符）选项卡

**MIDI Descriptor** 选项卡用于添加和配置 MIDI 流接口描述符。欲了解更多信息，请参见 USBFS MIDI 的 MIDI Descriptor 选项卡部分所介绍的内容。

## CDC Descriptor（CDC 描述符）选项卡

**CDC Descriptor** 选项卡用于添加和配置通信及数据接口描述符。欲了解更多的信息，请参见 USBUART 中的 CDC Descriptor 选项卡部分所介绍的内容。

## Advanced（高级）选项卡




### 外部类

此参数允许用户固件或其他位于解决方案级别的组件提供对类请求的管理。如果使能此参数，应该执行 USBFS_DispatchClassRqst()函数。

### External Vendor（外部供货商）

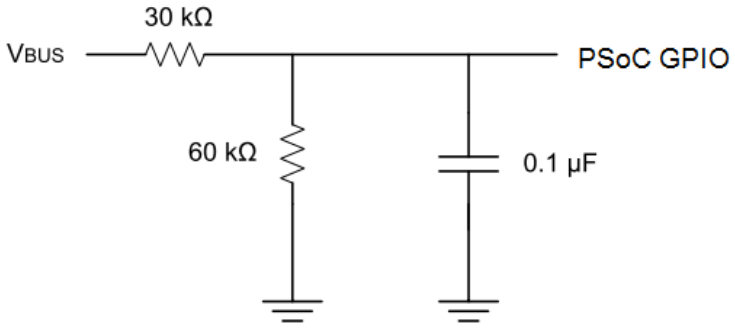
此参数允许用户固件或其他位于解决方案级别的组件提供对提供商特定请求的管理。如果使能此参数，应该执行 USBFS_HandleVendorRqst()函数。

### 使能 VBUS 监控

USB 规范要求在任何时候都不应当有器件在上沿端口处为 VBUS 提供电流。为了满足此要求，器件必须监控 VBUS 是否存在，如果 VBUS 不存在，则移除来自“D+/D-”上拉电阻的供电。

对于总线供电设计，当从主机拔除 USB 电缆时，肯定会移除电源；但是对于自供电设计，器件必须符合此要求，才能进行正确的操作和通过 USB 认证。

如果选中 **Internal VBUS** 选项，此参数将单一 VBUS 监控引脚添加到设计中。默认情况下，可将此引脚的驱动模式配置为 High Impedance Digital（高阻抗数字），也可以通过引脚特定的 API USBFS_VBUS_SetDriveMode()将其设置为其他模式。当选中 **External VBUS** 选项时，数字输入引脚组件应置于原理图内并必须连接到 vbusdet 输入终端。此引脚必须通过电阻式网络连接到 VBUS，并在引脚编辑器中进行分配。下图显示的是一个示例原理图。




如果为 PSoC Creator 引脚编辑器中的 SIO 端口分配了某个监控引脚，便可将该引脚直接连接到 VBUS。该操作取决于 SIO 的热交换功能。

USBFS_VBusPresent()函数返回 VBUS 的状态。有关其他信息，请参见自供电器件的 USB 符合性部分的内容。

### Enable SOF Output（使能 SOF 输出）

此参数使能“帧起始 SOF”输出。

## 时钟设置

USB 硬件模块要求通过 PSoC Creator 设计范围资源时钟编辑器配置系统时钟。当使用 USBFS 组件时，时钟设置具有下列要求：

- 必须使能 USB 时钟。

- ILO 必须设置为 100 KHz。

有一些方法可将系统时钟配置为符合这些要求。图 1 和图 2 显示的是可用的选项集。您的设计可能需要不同的设置。

**图 1. 系统时钟配置，IMO 是源时钟**

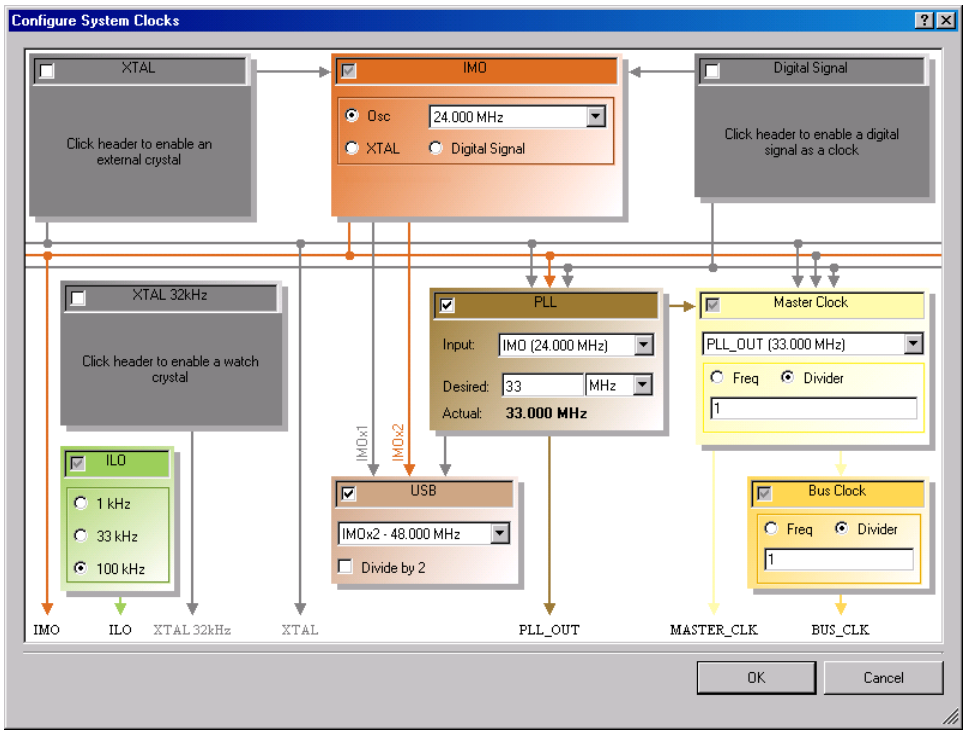
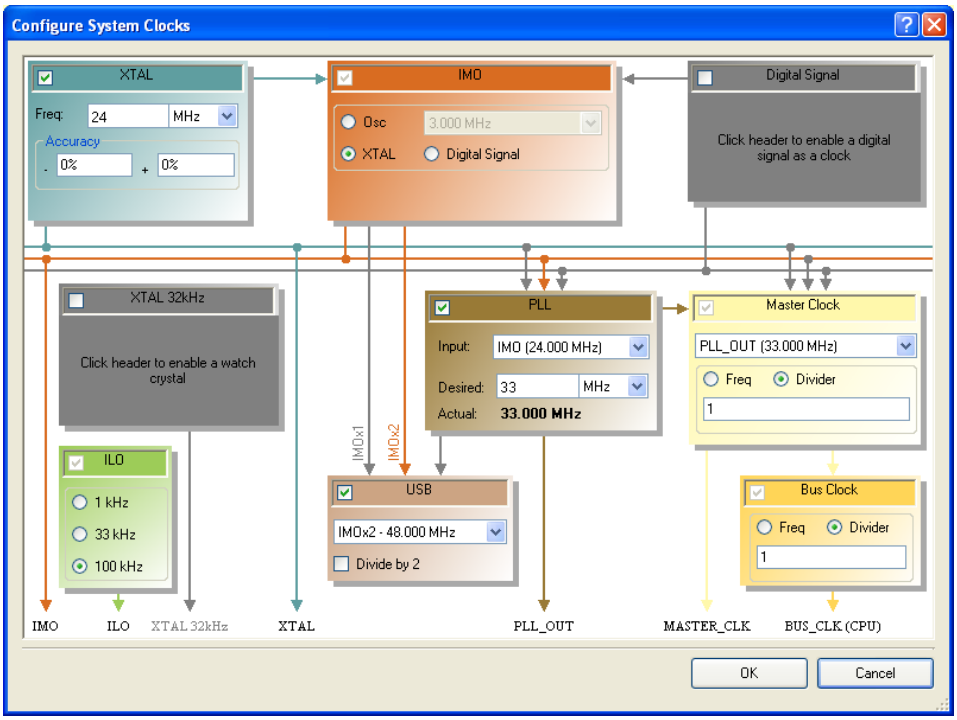


**图 2. 系统时钟配置，XTAL 是时钟源**




# 应用编程接口（API）

通过应用编程接口（API），您可以使用软件对组件进行配置。下表列出并说明每个函数的接口。以下各节将更详细地介绍每个函数。

默认情况下，PSoC Creator 将实例名称“USBFS_1”分配给指定设计中组件的第一个实例。您可以将其重新命名为任何一个符合标识符语法规则的值。实例名称会成为每个全局函数名称、变量和符号常量的前缀。为便于阅读，下表使用的实例名称为“USBFS”。

## 基本 USBFS 器件 API

| 函数 | 说明 |
|---|---|
| USBFS_Start() | 激活要用于器件和特定电压模式的组件。 |
| USBFS_Init() | 初始化组件的硬件。 |
| USBFS_InitComponent() | 初始化组件的全局变量，并通过上拉D+线路初始化与主机通信的数据操作。 |
| USBFS_Stop() | 禁用组件。 |
| USBFS_GetConfiguration() | 返回当前分配的配置。如果未配置器件，则返回0。 |
| USBFS_IsConfigurationChanged() | 返回读取时清除的配置状态。 |

| 函数 | 说明 |
|---|---|
| USBFS_GetInterfaceSetting() | 返回指定接口的当前备用设置。 |
| USBFS_GetEPState() | 返回指定USBFS端点的当前状态。 |
| USBFS_GetEPAckState() | 确定ACK数据操作是否在此端点上发生。 |
| USBFS_GetEPCount() | 从指定USBFS端点返回当前字节计数。 |
| USBFS_InitEP_DMA() | 为EP数据传输初始化DMA。 |
| USBFS_LoadInEP() | 加载并使能用于IN传输的指定USBFS端点。 |
| USBFS_ReadOutEP() | 从端点RAM读取字节的指定数量，并将其放入由pSrc指向的RAM阵列。此函数返回主机发送的字节数量。 |
| USBFS_EnableOutEP() | 使能指定的USB端点以接受OUT传输。 |
| USBFS_DisableOutEP() | 禁用指定的USB端点，以否认OUT传输。 |
| USBFS_SetPowerStatus() | 将器件设置为自供电或总线供电。 |
| USBFS_Force() | 在USB D+/D-引脚上强制J、K或SE0状态通常用于远程唤醒。 |
| USBFS_SerialNumString() | 在运行时提供USB器件序列号字符串描述符的源。 |
| USBFS_TerminateEP() | 终止端点传输。 |
| USBFS_VBusPresent() | 确定是否存在用于自供电器件的VBUS。 |

## 全局变量

| 变量 | 说明 |
|---|---|
| USBFS_initVar | 指示是否已初始化USBFS。该变量初始化为 0，在第一次调用 USBFS_Start() 时设置为 1。这使得组件无需在第一次调用USBFS_Start()子程序后重新初始化便可重新启动。<br>如果需要组件重新初始化，在调用USBFS_Start()子程序之前将此变量设置为‘0’。另外，可以通过调用USBFS_Init()和USBFS_InitComponent()函数来重新初始化USBFS。 |
| USBFS_device | 包含启动器件编号。此变量由USBFS_Start()或USBFS_InitComponent() API设置。 |
| USBFS_transferState | 此变量由通信函数用来处理当前传输状态。<br>在USBFS_InitComponent() API中以及在状态阶段完成传输后，初始化为TRANS_STATE_IDLE。<br>根据请求类型，在设置数据操作中更改为TRANS_STATE_CONTROL_READ或TRANS_STATE_CONTROL_WRITE。 |

| 变量 | 说明 |
|---|---|
| USBFS_configuration | 包含主机使用SET_CONFIGURATION请求设置的当前配置编号。此变量在USBFS_InitComponent() API中初始化为零，然后返回到USBFS_GetConfiguration() API的应用程序级别。 |
| USBFS_configurationChanged | SET_CONFIGURATION和SET_INTERFACE请求后，此变量被设置为'1'。然后通过USBFS_IsConfigurationChanged() API返回到应用程序级别。 |
| USBFS_deviceStatus | 这是两位变量，第一位中指示电源状态（DEVICE_STATUS_BUS_POWERED或DEVICE_STATUS_SELF_POWERED），第二位指示远程唤醒状态(DEVICE_STATUS_REMOTE_WAKEUP)。此变量在 USBFS_InitComponent() API 中初始化为零，由 USBFS_SetPowerStatus() API 配置。 |

## void USBFS_Start(uint8 device, uint8 mode)

**说明：** 此函数执行USBFS组件所要求的所有初始化操作。

**参数：** uint8 device：包含所需器件描述符的器件编号。器件编号位于“Configure”对话框的“Device Descriptor”选项卡中所需器件描述符设置下的**Device Number**字段中。

uint8 mode：工作电压。它确定是否针对5 V操作使能了电压调压器，或者是否针对3.3 V操作使用了通过模式。下表中给出了象征名称及其相关值。

| 电源设置 | 注释 |
|---|---|
| USBFS_3V_OPERATION | 禁用电压调压器和通过$V_{CC}$的上拉 |
| USBFS_5V_OPERATION | 使能电压调压器，并将调节器用于上拉 |
| USBFS_DWR_VDDD_OPERATION | 根据DWR中的$V_{DDD}$电压配置，使能或禁用电压调压器。 |

**返回值：** 无

**其他影响：** 无

## void USBFS_Init(void)

**说明：** 此函数根据配置窗口 Configure（配置）对话框设置来初始化或恢复器件。不需要调用USBFS_Init()，因为 USBFS_Start() 子程式会调用此函数，这是开始组件操作的首选方法。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void USBFS_InitComponent(uint8 device, uint8 mode)

**说明：** 此函数初始化组件的全局变量，并通过上拉D+线路初始化与主机通信的操作。

**参数：** uint8 device：包含所需器件描述符的器件编号。器件编号位于“Configure”对话框的“Device Descriptor”选项卡中所需器件描述符设置下的**Device Number**字段中。

uint8 mode：工作电压。它确定是否针对5 V操作使能了电压调压器，或者是否针对3.3 V操作使用了通过模式。下表中给出了象征名称及其相关值。

| 电源设置 | 注释 |
|---|---|
| USBFS_3V_OPERATION | 禁用电压调压器和通过$V_{CC}$的上拉 |
| USBFS_5V_OPERATION | 使能电压调压器，并将调节器用于上拉 |
| USBFS_DWR_VDDD_OPERATION | 根据DWR中的$V_{DDD}$电压配置，使能或禁用电压调压器。 |

**返回值：** 无

**其他影响：** 无

## void USBFS_Stop(void)

**说明：** 此函数执行USBFS组件所要求的所有初始化操作。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## uint8 USBFS_GetConfiguration(void)

**说明：** 此函数获取USB器件的当前配置。

**参数：** 无

**返回值：** uint8：返回当前分配的配置。如果未配置器件，则返回 0。

**其他影响：** 无

## uint8 USBFS_IsConfigurationChanged(void)

**说明：** 此函数将返回读取时清除的配置状态。当PC为相同的配置编号发送两个SET_CONFIGURATION请求时，该函数将会非常有用。

**参数：** 无

**返回值：** uint8：更改了新的配置后，会返回一个非零值；否则返回的数值是零。

**其他影响：** 无

## uint8 USBFS_GetInterfaceSetting(uint8 interfaceNumber)

**说明：** 此函数获取指定接口的当前备用设置。

**参数：** uint8 interfaceNumber：接口编号

**返回值：** uint8：返回指定接口的当前备用设置。

**其他影响：** 无

## uint8 USBFS_GetEPState(uint8 epNumber)

**说明：** 此函数返回请求端点的状态。

**参数：** uint8 epNumber：数据端点编号

**返回值：** uint8：返回指定USBFS端点的当前状态。下表中给出了象征名称及其相关值。当您编写代码以更改端点状态（例如编写ISR代码以处理发送或接收的数据）时，请使用这些常量。

| 返回值 | 说明 |
|---|---|
| USBFS_NO_EVENT_PENDING | 端点正在等待SIE操作 |
| USBFS_EVENT_PENDING | 端点正在等待CPU操作 |
| USBFS_NO_EVENT_ALLOWED | 端点已被锁定，无法访问 |
| USBFS_IN_BUFFER_FULL | 加载IN端点，并模式设置为ACK IN |
| USBFS_IN_BUFFER_EMPTY | 发生IN数据操作，可以加载更多数据 |
| USBFS_OUT_BUFFER_EMPTY | OUT端点设置为ACK OUT，并正在等待数据 |
| USBFS_OUT_BUFFER_FULL | 发生OUT数据操作，可以读取数据 |

**其他影响：** 无

### uint8 USBFS_GetEPAckState(uint8 epNumber)

**说明：** 通过读取端点控制寄存器中的ACK位，此函数确定此端点上是否发生ACK数据操作。此函数不清除ACK位。

**参数：** uint8 epNumber：包含数据端点编号。

**返回值：** uint8：如果发生已确认的数据传输，此函数会返回非零值。否则，此函数将返回零值。

**其他影响：** 无

### uint16 USBFS_GetEPCount(uint8 epNumber)

**说明：** 此函数返回请求端点的传输计数。计数寄存器提供的值包含两个计数，对应于数据包的两个字节校验和。此函数减去这两个计数。

**参数：** uint8 epNumber：包含数据端点编号。

**返回值：** uint16：返回指定USBFS端点提供的当前字节计数，或者针对无效端点返回‘0’值。

**其他影响：** 无

### void USBFS_InitEP_DMA(uint8 epNumber, const uint8 *pData)

**说明：** 此函数分配并初始化DMA通道，以便USBFS_LoadInEP()或USBFS_ReadOutEP() API使用它传输数据。“Endpoint Memory Management”参数被设置为DMA时，此函数可用。

可从USBFS_LoadInEP()和USBFS_ReadOutEP() API自动调用此函数。

**参数：** uint8 epNumber：包含数据端点编号。

const uint8 *pData：指向与EP传输相关的数据阵列的指针。

**返回值：** 无

**其他影响：** 无

## void USBFS_LoadInEP(uint8 epNumber, const uint8 pData[], uint16 length)

**说明：** 手动模式：针对IN数据传输，此函数加载并使能指定USB数据端点。

**手动DMA：**

- 配置从数据 RAM 到端点 RAM 的 DMA 数据传输。
- 生成传输请求。

**自动DMA：**

- 配置 DMA。由于只需执行一次，因此只有 pData 参数为非 NULL（空）时，才能实现该操作。当 pData 指针为 NULL（空）时，此函数会跳过该任务。
- 设置数据就绪状态：生成第一个 DMA 传输并准备端点 RAM 存储器中的数据。将 pData 参数设置为 NULL，以执行该任务。

**参数：** uint8 epNumber：包含数据端点编号。

const uint8 pData[]：指向数据阵列（可从该阵列加载端点空间的数据）的指针。

uint16 length：要从阵列中传输、然后因IN请求而发送的字节数。有效值介于0至512之间（自动DMA模式为1023）。

**返回值：** 无

**其他影响：** 无

### uint16 USBFS_ReadOutEP(uint8 epNumber, uint8 pData[], uint16 length)

**说明：** 手动模式：此函数将指定数量的字节从端点RAM移动到数据RAM。从端点 RAM 实际传输到数据 RAM 的字节数是主机发送的实际字节数和 wCount 参数请求的字节数二者中的较小者。

**手动DMA：**

- 配置从端点 RAM 到数据 RAM 的 DMA 数据传输。
- 生成传输请求。
- 调用 USB_ReadOutEP() API 后，并在使用预期数据前，它必须等待 DMA 传输完成。例如，通过下面内容检查 EPstate：

```
while (USBFS_GetEPState(OUT_EP) == USB_OUT_BUFFER_FULL);
```

**自动DMA：**

- 配置 DMA。只需执行一次。

**参数：** uint8 epNumber：包含数据端点编号。

uint8 pData[]：指向数据阵列（可从该阵列加载端点空间的数据）的指针。

uint16 length：要从USB OUT端点传输并加载到数据阵列中的字节数。有效值介于0至512之间（自动DMA模式为1023）。如果主机发送的字节少于请求的字节数，则该函数移动的字节少于请求字节数。

**返回值：** uint16：接收的字节数

**其他影响：** 无

### void USBFS_EnableOutEP(uint8 epNumber)

**说明：** 此函数使能用于OUT传输的指定端点。不要为IN端点调用此函数。

**参数：** uint8 epNumber：包含数据端点编号。

**返回值：** 无

**其他影响：** 无

### void USBFS_DisableOutEP(uint8 epNumber)

**说明：** 此函数禁用指定的USBFS OUT端点。不要为IN端点调用此函数。

**参数：** uint8 epNumber：包含数据端点编号。

**返回值：** 无

**其他影响：** 无

## void USBFS_SetPowerStatus(uint8 powerStatus)

**说明：** 此函数设置当前电源状态。器件将根据此值应答USB GET_STATUS请求。这允许器件针对USB第9章合规性正确地报告其状态。器件可以随时将其电源从自供电更改为总线供电，并将其当前电源报告为器件状态的一部分。您应当在器件从自供电更改为总线供电或从总线供电更改为自供电时调用此函数，并相应设置状态。

**参数：** uint8 powerStatus： 包含所需的电源状态，‘1’表示自供电，‘0’表示总线供电。下面提供了象征名称及其相关值：

| 电源状态 | 说明 |
|---|---|
| USBFS_DEVICE_STATUS_BUS_POWERED | 将器件设置为总线供电 |
| USBFS_DEVICE_STATUS_SELF_POWERED | 将器件设置为自供电 |

**返回值：** 无

**其他影响：** 无

## void USBFS_Force(uint8 state)

**说明：** 此函数在D+/D-线路上强制USB J、K或SE0状态。此函数为USB器件应用程序提供必需的机制来执行 USB远程唤醒。欲了解更多有关信息，请参见USB 2.0规范，以获取有关暂停和恢复功能的详情。

**参数：** uint8 state：一个字节，用于指示四个总线状态中需要使能的一个。下面列出了象征名称及其相关值：

| 状态 | 说明 |
|---|---|
| USBFS_FORCE_SE0 | 在D+/D-线路上强制SE0状态 |
| USBFS_FORCE_J | 在D+/D-线路上强制J状态 |
| USBFS_FORCE_K | 在D+/D-线路上强制K状态 |
| USBFS_FORCE_NONE | 将总线返回SIE控制 |

**返回值：** 无

**其他影响：** 无

## void USBFS_SerialNumString(uint8 snString[])

**说明：** 只有选中了**Serial Number String**描述符属性中的 **User Call Back**选项时，此函数才可用。应用程序固件可以在运行时间内提供USB器件序列号字符串描述符的源。如果应用程序固件不使用此函数或设置了错误的字符串描述符，将使用默认字符串。

**参数：** uint8 snString[]：指向用户定义的字符串描述符的指针。字符串描述符应当符合*通用串行总线规范修订版2.0第9.6.7章*的要求。

| 偏移 | 大小 | 数值 | 说明 |
|---|---|---|---|
| 0 | 1 | N | 该描述符的大小（单位为字节） |
| 1 | 1 | 0x03 | STRING描述符类型 |
| 2 | N - 2 | 编号 | UNICODE编码字符串 |

例如：*uint8 snString[16]={0x0E,0x03,'F',0,'W',0,'S',0,'N',0,'0',0,'1',0};*

**返回值：** 无

**其他影响：** 无

## void USBFS_TerminateEP(uint8 epNumber)

**说明：** 此函数终止指定的USBFS端点。此函数应在端点重新配置之前使用。

**参数：** uint8 epNumber：包含数据端点编号。

**返回值：** 无

**其他影响：** 当所选的端点上发生任何数据操作时，器件都会使用NAK信号进行响应。

## uint8 USBFS_VBusPresent(void)

**说明：** 确定是否存在用于自供电器件的VBUS。

在“Advanced”选项卡中使能了VBUS监控选项时，此函数才可用。

**参数：** 无

**返回值：** 返回值如下所示：

| 返回值 | 说明 |
|---|---|
| 1 | VBUS存在 |
| 0 | VBUS不存在 |

**其他影响：** 无

## 人机接口器件（HID）类支持

| 函数 | 说明 |
|---|---|
| USBFS_UpdateHIDTimer() | 更新指定接口的HID报告定时器，如果定时器到期，则返回1，如果未到期，则返回0。如果定时器到期，它将重新加载定时器。 |
| USBFS_GetProtocol() | 返回指定接口的协议 |

### 全局变量

如果 HID 描述符包含报告，定时器将为来自 HID 类器件的数据报告创建一个报告存储区域。它会为 IN、OUT 和 FEATURE 报告创建单独的报告区域。如果没有其他报告 ID 标志出现在报告描述符中，但只存在一个输入、输出和特性报告结构，则此区域足够用。

| 变量 | 说明 |
|---|---|
| USBFS_hidProtocol | 此变量在USBFS_InitComponent() API中初始化为PROTOCOL_REPORT值。主机使用HID_SET_PROTOCOL请求来控制此变量。USBFS_GetProtocol() API 将该值返回给用户代码。 |
| USBFS_hidIdleRate | 此变量控制HID报告速率。主机使用HID_SET_IDLE请求来控制此变量，并USBFS_UpdateHIDTimer() API使用此变量来重新加载定时器。 |
| USBFS_hidIdleTimer | 此变量包含由USBFS_UpdateHIDTimer() API递减和重新加载的定时器计数器。 |
| USBFS_DEVICEx_CONFIGURATIONx_INTERFACEx_ALTERNATEx_HID_FEATURE[1]_BUF_IDx[2] [][3] | 在HID描述符内创建特性（输入或输出）报告描述符时，该可选的报告缓冲区将作为一个变量使用。根据HID报告的定义，此缓冲区的大小由定制器自动计算，并定义为USBFS_DEVICEx_CONFIGURATIONx_INTERFACEx_ALTERNATEx_HID_FEATURE[1]_BUF_SIZE_IDx[2][3]。主机通过使用SET_REPORT和GET_REPORT请求控制该缓冲区。用户代码通过检查状态完成模块，可以确定特定报告ID中的控制传输是否完成。 |

[1] 变量名称中的“FEATURE”字段表示报告的类型，并能分别修改为与 IN 或 OUT 报告相应的“IN”或“OUT”。

[2] 只有在报告描述符中指定了报告 ID 时，变量名称中的“_Idx”字段才会出现，同时，“x”符号会变为相关的报告 ID 编号。

[3] 变量名称中的“x”符号取决于取自定制器中描述符根的相关器件、配置、接口以及备用设置数量。

<table>
<tr><th>变量</th><th>说明</th></tr>
<tr><td>USBFS_DEVICEx_CONFIGURATIONx_INTERFACEx_ALTERNATEx_HID_FEATURE[1]_RPT_SCB_IDx[2][3]</td><td>状态完成模块包含两个数据项，即一个一字节的完成状态代码和一个二字节的数据传输长度。此模块具有以下结构。<br><pre>typedef struct _USBFS_XferStatusBlock
{
    uint8  status;
    uint16 length;
} T_USBFS_XFER_STATUS_BLOCK;</pre>“main”应用程序会监控完成状态，以确定如何继续。可在下表中找到完成状态代码。数据传输长度为所传输数据字节的实际数目。
<table>
<tr><th>返回值</th><th>注释</th></tr>
<tr><td>USBFS_XFER_IDLE</td><td>指示了相关数据缓冲区无有效数据，并应用程序不应使用该缓冲区。实际数据传输发生在完成代码为USBFS_XFER_IDLE时，但是它不会指示某个传输正在运行中。</td></tr>
<tr><td>USBFS_XFER_STATUS_ACK</td><td>指示控制传输状态阶段已成功完成。此时，应用程序使用相关数据缓冲区及其内容。</td></tr>
<tr><td>USBFS_XFER_PREMATURE</td><td>指示了控制传输被后续控制传输的SETUP中断。对于控制写入，相关数据缓冲区的内容包含直到提前完成时的数据。</td></tr>
<tr><td>USBFS_XFER_ERROR</td><td>指示未收到预期的状态阶段令牌。</td></tr>
</table>
</td></tr>
</table>

## uint8 USBFS_UpdateHIDTimer(uint8 interface)

**说明：** 此函数更新HID报告空闲定时器并返回状态；如果定时器到期，此函数将加载它。

**参数：** uint8 interface：包含接口编号。

**返回值：** uint8：返回HID定时器的状态。下面提供了象征名称及其相关值：

| 返回值 | 注释 |
|---|---|
| USBFS_IDLE_TIMER_EXPIRED | 定时器已到期。 |
| USBFS_IDLE_TIMER_RUNNING | 定时器正在运行。 |
| USBFS_IDLE_TIMER_IDEFINITE | 当数据或状态更改时发送报告。 |

**其他影响：** 无

### uint8 USBFS_GetProtocol(uint8 interface)

**说明：** 此函数返回所选接口的HID协议值。

**参数：** uint8 interface：包含接口编号。

**返回值：** uint8：返回协议值。

**其他影响：** 无

## Bootloader 支持

USBFS 组件可以作为 Bootloader 的通信组件使用。使用以下配置为外部系统与 Bootloader 之间的通信协议提供支持：

- 端点编号：EP1，方向：OUT，传输类型：INT，最大数据包大小：64
- 端点编号：EP2，方向：IN，传输类型：INT，最大数据包大小：64

模板文件（*bootloader.root.xml*）中存储了完整的建议配置。在 **Device Descriptor** 树中选择 **Descriptor Root**，单击 **Import** 按键，浏览到以下目录，然后打开 *bootloader.root.xml* 文件。

*<INSTALL>\psoc\content\cycomponentlibrary\CyComponentLibrary.cylib\USBFS_v2.70\Custom\template*

更多有关 Bootloader 的信息，请参考*《系统参考指南》*。

USBFS 组件为 Bootloader 提供了一组 API 函数。

| 函数 | 说明 |
|---|---|
| USBFS_CyBtldrCommStart() | 执行针对USBFS组件的所需初始化，等待枚举，然后使能通信。 |
| USBFS_CyBtldrCommStop() | 调用 USBFS_Stop() 函数。 |
| USBFS_CyBtldrCommReset() | 复位接收和传输通信缓冲区。 |
| USBFS_CyBtldrCommWrite() | 允许调用程序将数据写入Bootloader主机。此函数将以轮询方式处理以便将数据块完整发送到主机器件。 |
| USBFS_CyBtldrCommRead() | 允许调用程序读取Bootloader主机中的数据。该函数将以轮询方式处理以便从主机组件完整接收数据块。 |

### void USBFS_CyBtldrCommStart(void)

**说明：** 此函数执行针对USBFS组件的所需初始化，等待枚举，然后使能通信。

**参数：** 无

**返回值：** 无

**其他影响：** 此函数使用3 V操作启用USBFS。

### void USBFS_CyBtldrCommStop(void)

**说明：** 此函数执行USBFS组件所要求的所有初始化操作。

**参数：** 无

**返回值：** 无

**其他影响：** 调用 USBFS_Stop() 函数。

### void USBFS_CyBtldrCommReset(void)

**说明：** 此函数复位接收和传输通信缓冲区。

**参数：** 无

**返回值：** 无

**其他影响：** 无

### cystatus USBFS_CyBtldrCommWrite(const uint8 pData[], uint16 size, uint16 *count, uint8 timeOut)

**说明：** 此函数允许调用程序将数据写入Bootloader主机中。它将以轮询方式处理以便将数据块完整发送到主机器件。

**参数：** const uint8 pData[]：发送到组件的数据块的指针。

uint16 size：要写入的字节数。

uint16 *count：指向无符号Short型变量的指针，此变量用于写实际写入的字节数。

uint8 timeout：等待的单位数，之后会因超时而返回。

**返回值：** cystatus：如果未遇到任何问题则返回CYRET_SUCCESS，或是返回对该问题描述最准确的值。更多有关信息，请参考*《系统参考指南》*中“返回代码”一节中的内容。

**其他影响：** 无

### cystatus USBFS_CyBtldrCommRead(uint8 pData[], uint16 size, uint16 *count, uint8 timeOut)

**说明：** 此函数允许调用程序读取引导加载程序主机中的数据。此函数将以轮询方式处理以便从主机器件完整接收数据块。

**参数：** uint8 pData[]：指向用于存储从器件接收的数据模块的区域的指针。

uint16 size：要读取的字节数

uint16 *count：指向无符号Short型变量的指针，此变量用于写实际读取的字节数。

uint8 timeOut：等待的单位数，之后会因超时而返回。

**返回值：** cystatus：如果未遇到任何问题则返回CYRET_SUCCESS，或是返回对该问题描述最准确的值。更多有关信息，请参考*《系统参考指南》*中“返回代码”一节中的内容。

**其他影响：** 无

## USB 暂停、恢复以及远程唤醒

USBFS 组件支持 USB 暂停、恢复以及远程唤醒。因为这些特性紧密耦合到用户应用程序，因此 USBFS 组件提供了一组 API 函数。

| 函数 | 说明 |
|---|---|
| USBFS_CheckActivity() | 检查并清除USB总线活动标志。如果自从上次检查后USB处于活动状态，则返回‘1’；否则返回‘0’。 |
| USBFS_Suspend() | 禁用USBFS模块，并准备断电模式。 |
| USBFS_Resume() | 在断电模式后使能USBFS模块。 |
| USBFS_RWUEnabled() | 返回当前远程唤醒状态。 |

## uint8 USBFS_CheckActivity(void)

**说明：** 此函数返回总线的活动状态，并清除状态硬件以便在下次调用此子程序时提供最新活动状态。

此函数提供一种方法来确定是否发生任何USB总线活动。应用程序使用此函数以确定是否满足进入USB暂停的条件。

**参数：** 无

**返回值：** uint8 cystatus：标准API返回值。

| 返回值 | 说明 |
|---|---|
| 1 | 自从上次调用此函数后，检测到总线活动 |
| 0 | 自从上次调用此函数后，未检测到总线活动 |

**其他影响：** 无

## void USBFS_Suspend(void)

**说明：** 此函数禁用USBFS模块，并准备断电模式。应当刚好在进入睡眠之前调用。

USBFS_Suspend()也初始化D+引脚的中断，以唤醒PICU源中的睡眠模式。

满足进入USB暂停的条件后，应用程序会采取适当的步骤来减少电流消耗，以满足暂停电流要求。要将 USB SIE 和收发器置于断电模式，应用程序需要调用 USBFS_Suspend() API 函数和 USBFS_CheckActivity() API 来检测 USB 活动。此函数禁用 USBFS 模块，但是维护当前 USB 地址（在 USBCR 寄存器中）。该器件使用睡眠功能减少功耗。

**参数：** void

**返回值：** void

**其他影响：** 无

## void USBFS_Resume(void)

**说明：** 此函数在断电模式后使能 USBFS 模块。应当刚好在从睡眠唤醒后调用。

在器件暂停时，它定期检查以确定是否符合离开暂停状态的条件。检查恢复条件的一种方法是使用睡眠定时器定期唤醒器件。另一种方法是配置器件以从PICU唤醒。

如果符合恢复条件，则应用程序调用USBFS_Resume() API函数。此函数使能 USBFS SIE 和收发器，使它们脱离断电模式。这样，不会更改USBCR寄存器的USB地址字段；从而可保持以前由主机分配的USB地址。

**参数：** void

**返回值：** void

**其他影响：** 无

### uint8 USBFS_RWUEnabled(void)

**说明：** 此函数返回当前远程唤醒状态。

如果器件支持远程唤醒，则应用程序可以使用此函数来确定主机是否使能远程唤醒。当器件暂停且确定满足用于启动远程唤醒的条件时，应用程序会使用USBFS_Force() API函数来强制在USB上的相应J、K状态，从而标志着远程唤醒。

**参数：** void

**返回值：** True：使能远程唤醒

False：禁用远程唤醒

**其他影响：** 无

睡眠模式 API 使用示例，PICU 源用于唤醒：

```
USBFS_Suspend();
CyPmSaveClocks();
CyPmSleep(PM_SLEEP_TIME_NONE, PM_SLEEP_SRC_PICU);
CyPmRestoreClocks();
USBFS_Resume();
```

## 音频类支持

更多有关信息，请参见 USBFS 音频中音频类支持部分中的内容。

## MIDI Class Support

更多有关信息，请参见 USBFS MIDI 中 MIDI 支持部分内容。

## CDC 类支持

更多有关信息，请参见 USBUART 中 CDC 类支持部分中内容。

# 中断服务子程序

该组件提供了空的 SOF ISR。默认情况下该中断被禁用。如果应用程序需要此中断，可以通过调用以下函数使能它：

CyIntEnable(USBFS_SOF_VECT_NUM);

您可以将自定义代码放入指定的区域中，以执行所需的其他函数。

在**带手动和自动存储器管理的 DMA** 模式下，仲裁器中断（USBFS_arb_int）表示 DMA 请求的服务已经完成。系统必须设置此中断的优先级，使其高于 USBFS_ep_[0..8]和 USBFS_ord_int 中断的优先级。因此，将此中断的优先级设置为 USBFS_Init()函数中的 USBFS_ARB_PRIOR 值。

# MISRA 符合性

本节介绍了 MISRA-C:2004 符合性和本组件的偏差情况。定义了两种类型的偏差：

- 项目偏差 — 适用于所有 PSoC Creator 器件的偏差
- 特定偏差 — 仅适用于该组件的偏差

本节介绍了有关组件特定偏差的信息。*《系统参考指南》*的“MISRA 符合性”章节中介绍了项目偏差以及有关 MISRA 符合性验证环境的信息。

USBFS 组件具有以下特定偏差：

| MISRA-C:2004 规则 | 规则类别（必须(R)/建议(A)） | 规则说明 | 偏差说明 |
|---|---|---|---|
| 11.4 | A | 不同对象指针类型间不能实现转换。 | PutString() API使用一个指向字符串的指针作为参数。此函数将该指针转换成uint8阵列，然后将其发送给LoadInEP() API，以向HOST（主机）传输数据。<br>器件描述符结构使用指针来阻止不同描述符类型被组合起来使用。这些指针被转换为另一个指针类型。<br>注释：由于解析结构的函数已经知道了对象的类型，所以此操作是安全的。 |
| 11.5 | R | 不执行转换操作，以删除由指针寻址的类型中所有的常量或易失性资质。 | 器件描述符结构使用常量资质。将指针发送到通用LoadInEP() API之前，需要删除常量资质。 |
| 16.7 | A | 如果该指针不用于更改地址对象，应将函数原型中的指针参数声明为指向“const”常量的指针。 | 这种偏差只适用于DMA存储器管理模式。ReadOutEP() API的pData参数用于修改手动存储器管理模式中的寻址对象。 |
| 17.4 | R | 数组索引是唯一允许的指针运算形式。 | 组件将数组下标适用于一个指针类型的对象，以便访问带有描述符的结构。 |

该组件具有下面的嵌入式组件：中断、时钟、DMA。MISRA 符合性与特定偏差的相关信息，请参见相应组件数据手册。

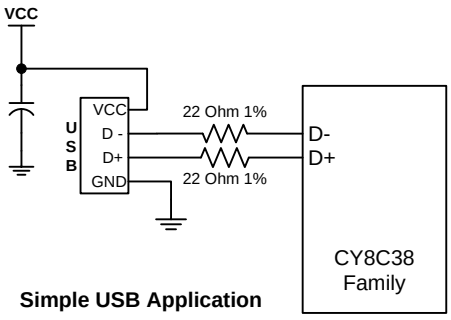
# 示例固件源代码

PSoC Creator 在 Find Example Project（查找示例项目）对话框中提供了很多包括原理图和代码示例的示例项目。要查看特定组件实例，请打开 Component Catalog 中的对话框或原理图中的组件示例。要查看通用示例，请打开 Start Page 或 **File** 菜单中的对话框。根据要求，可以通过使用对话框中的 **Filter Options** 选项来限定可选的项目列表。

更多有关信息，请参考《PSoC Creator 帮助》部分中主题为“查找示例项目”的内容。

# 功能说明

下图显示的是一个简单的总线供电 USB 应用，它连接了 PSoC 器件的‘D+’和‘D-’引脚。




**Simple USB Application**

## USB 符合性

USB 可以向器件提供各种总线条件，包括总线复位和不同时序要求。提供的示例中并不能正确阐述所有这些条件和要求。您应当确保设计的应用程序符合 USB 规范。

## 自供电器件的 USB 符合性

如果您创建的器件采用自供电，则必须通过电阻网络将 GPIO 引脚连接到 VBUS，并写入固件以监控 GPIO 的状态。可以使用 USBFS_Start()和 USBFS_Stop() API 子程序来控制 D+和 D- 引脚上拉。调用 USBFS_Start()之前，上拉电阻不会向数据线供电。USBFS_Stop()断开上拉电阻与数据引脚的连接。

根据使用 USBFS_SetPowerStatus()函数设置的状态，器件对 GET_STATUS 请求做出响应。如果器件配置为自供电，若要设置正确的状态，应当至少调用一次 USBFS_SetPowerStatus()。您还应当在器件更改状态时调用 USBFS_SetPowerStatus()函数。

## USB 标准器件请求

本节介绍的是 USBFS 组件支持的请求。如果是不支持的请求，USBFS 组件响应 STALL，表示错误的请求。

| 标准器件请求 | USB 组件支持说明 | USB 2.0 规范章节 |
|---|---|---|
| CLEAR_FEATURE | 器件 | 9.4.1 |
| | 接口 | |
| | 端点 | |
| GET_CONFIGURATION | 返回当前器件配置值 | 9.4.2 |
| GET_DESCRIPTOR | 如果描述符存在，将返回指定的描述符。 | 9.4.3 |
| GET_INTERFACE | 返回指定接口的所选备用接口设置 | 9.4.4 |
| GET_STATUS | 器件 | 9.4.5 |
| | 接口 | |
| | 端点 | |
| SET_ADDRESS | 设置用于所有将来器件访问的器件地址 | 9.4.6 |
| SET_CONFIGURATION | 设置器件配置 | 9.4.7 |
| SET_DESCRIPTOR | 不支持此可选请求 | 9.4.8 |
| SET_FEATURE | 器件：<br>DEVICE_REMOTE_WAKEUP支持由bRemoteWakeUp组件参数选择。<br>不支持TEST_MODE。 | 9.4.9 |
| | 接口 | |
| | Endpoint：暂停指定的端点。 | |
| SET_INTERFACE | 此请求允许主机为指定的接口选择备用设置。 | 9.4.10 |
| SYNCH_FRAME | 不支持。组件的将来实现会添加对此请求的支持，以使能与重复帧模式的同步传输。 | 9.4.11 |

## HID 类请求

| 类请求 | USBFS 组件支持说明 | HID的器件类定义 — 章节 |
|---|---|---|
| GET_REPORT | 允许主机通过控制管线接收报告。 | 7.2.1 |
| GET_IDLE | 读取特定输入报告的当前空闲速率。 | 7.2.3 |

| | | |
|---|---|---|
| GET_PROTOCOL | 读取当前处于活动状态的协议（引导协议或报告协议）。 | 7.2.5 |
| SET_REPORT | 允许主机向器件发送报告，可能设置输入、输出或功能控制的状态。 | 7.2.2 |
| SET_IDLE | 停止有关中断输入管道的特定报告，直到发生新事件或者经过指定的时间量。 | 7.2.4 |
| SET_PROTOCOL | 在引导协议和报告协议之间切换（或反过来切换）。 | 7.2.6 |

## 音频类请求

更多有关信息，请参见 USBFS 音频中音频类支持部分的内容。音频类请求 USBFS 音频

## CDC 类请求

更多有关信息，请参见 USBUART 中 CDC 类请求部分的内容。

# USBFS 音频

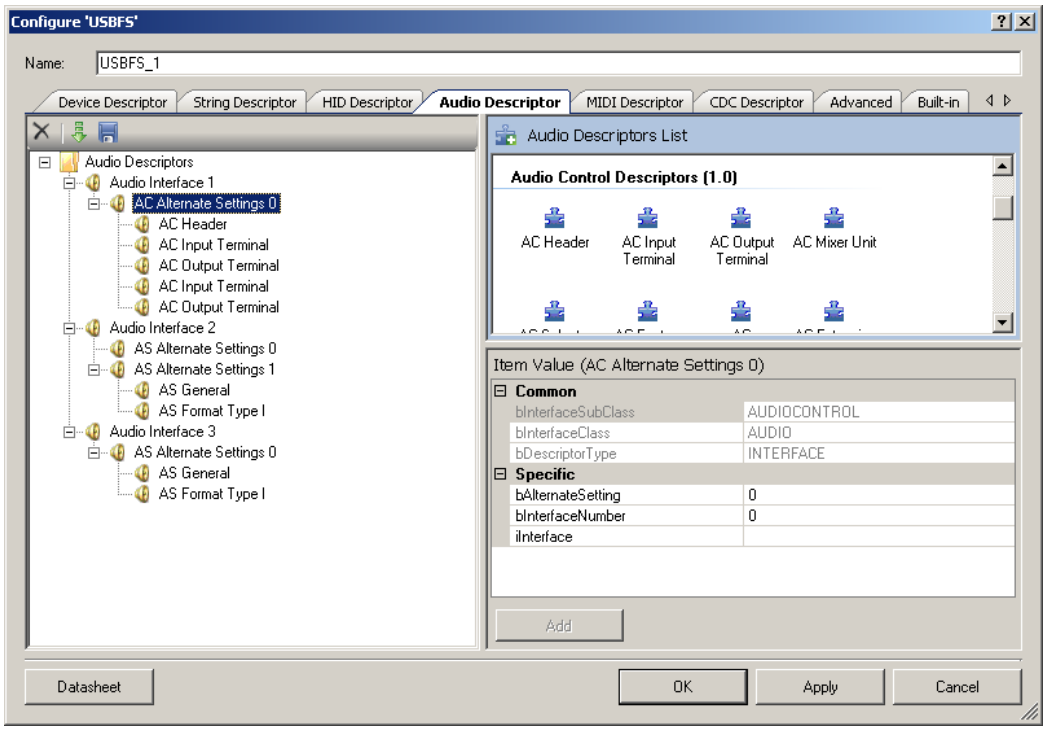
USBFS 组件为音频类描述符提供支持。根据 *音频器件的通用串行总线器件类定义 1.0 和 2.0* 规范，执行 USBFS 音频接口。

## 组件参数

将 USBFS 组件拖放到您的设计中，双击它以打开 **Configure USBFS** 对话框。

### Audio Descriptor（音频描述符）选项卡

**Audio Descriptor** 选项卡用于添加和配置音频接口描述符。




### 添加音频描述符

1. 在左侧目录中，选择 **Audio Descriptors** 根项目。
2. 在右侧的 **Audio Descriptors List** 下，选择 **Audio Control** 或 **Audio Streaming** 接口。
3. 在 **Item Value** 下，适当输入 **bAlternateSetting** 和 **bInterfaceNumber** 值。其他字段都可选的。

   **注意：**这些值是手动设置的。但是对于一般接口描述符，这些值是自动设置的。
4. 点击 **Add** 以将描述符添加到左侧目录中。

   您可以通过选择节点并单击它来重命名 **Audio Interface x** 标题。

### 添加特定于类的音频控制或音频流接口描述符

1. 在左侧的目录中，适当选择 **AC Alternate Settings x** 或 **AS Alternate Settings x** 项目。
2. 在右侧的 **Audio / MIDI Descriptors List**（音频/MIDI 描述符列表）下，适当选择 **Audio Control Descriptors (1.0)**、**Audio Control Descriptors (2.0)**或 **Audio Streaming Descriptors (1.0)**或 **Audio Streaming Descriptors (2.0)**的项目之一。

   有关版本 1.0 和 2.0 的信息，请参考 *音频器件的通用串行总线类定义*规范文件版本。
3. 在 **Item Value** 下，输入 **Specific** 中的适当值。
4. 点击 **Add** 以将描述符添加到左侧目录中。

### 添加音频端点描述符

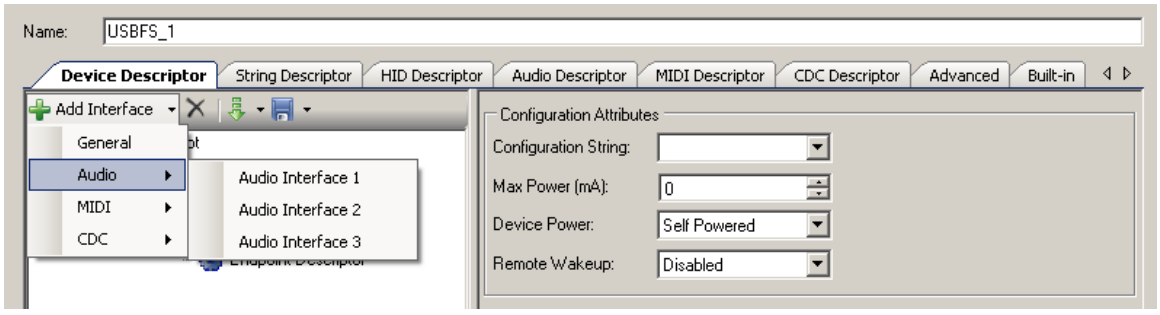
1. 在左侧的目录中，适当选择 **AC Alternate Settings x** 或 **AS Alternate Settings x** 项目。
2. 在右侧的 **Audio Descriptors List** 下，选择 **Endpoint Descriptor** 项。
3. 在 **Item Value** 下，输入 **Specific** 中的适当值。
4. 点击 **Add** 以将描述符添加到左侧目录中。

### 添加标准 AS 同步端点描述符

1. 在左侧的目录中，选择适当的 **Endpoint Descriptor**（端点描述符）。
2. 在右侧的 **Audio Descriptors List** 下，选择 **AS Endpoint Descriptor**（AS 端点描述符）。
3. 在 **Item Value** 下，输入 **Specific** 中的适当值。
4. 点击 **Add** 以将描述符添加到左侧目录中。

### 将配置的音频接口描述符添加到“Device Descriptor”目录中

1. 打开 **Device Descriptor** 选项卡。
2. 选择新接口所在的 **Configuration Descriptor**。
3. 点击 **Add Interface** 工具按键，选择 **Audio**，并选择要添加的相应项目。



Audio interfaces（音频接口）在 **Device Descriptor** 选项卡列表中被禁用，因为它们只能在 **Audio Descriptor** 选项卡中编辑。

**注意：**点击 **Apply** 或 **OK**，以保存各个选项卡的更改。如果点击 **Cancel**，将不保存已添加的所有描述符。

## USBFS 音频应用编程接口

默认情况下，PSoC Creator 将实例名称“USBFS_1”分配给指定设计中组件的第一个实例。您可以将其重新命名为任何一个符合标识符语法规则的值。实例名称会成为每个全局函数名称、变量和符号常量的前缀。为便于阅读，下表使用的实例名称为“USBFS”。

## 音频类支持

### 全局变量

| 变量 | 说明 |
|---|---|
| USBFS_currentSampleFrequency | 包含当前音频采样频率。主机使用发往端点的SET_CUR请求来设置它。 |
| USBFS_frequencyChanged | 此变量作为用户代码的标志使用，一定要注意，已经为主机发送了更改采样频率的请求。将在下一个OUT数据操作时发送采样频率。它包含设置时的端点地址。在主代码中检测新采样频率时建议使用以下代码：<br>`if((USBFS_frequencyChanged != 0) &&`<br>`(USBFS_transferState == USBFS_TRANS_STATE_IDLE))`<br>`{`<br>`/* Add core here.*/`<br>`USBFS_frequencyChanged = 0;`<br>`}`<br>检查USBFS_transferState变量以确保传输完成。 |
| USBFS_currentMute | 包含主机设置的静音配置。 |
| USBFS_currentVolume | 包含主机设置的音量级别。 |

## USBFS 音频功能说明

### 音频类请求

本节介绍的是 USBFS 组件支持的请求。如果是不支持的请求，USBFS 组件响应 STALL，表示错误的请求。

| 类请求 | USBFS组件支持说明 | 音频器件类定义 — 章节 |
|---|---|---|
| SET_CUR | 接口：<br>MUTE_CONTROL<br>VOLUME_CONTROL | 5.2.1.1 |
| | 端点：<br>SAMPLING_FREQ_CONTROL | |
| GET_CUR | 接口：<br>MUTE_CONTROL<br>VOLUME_CONTROL | 5.2.1.2 |
| | 端点：<br>SAMPLING_FREQ_CONTROL | |
| GET_MIN | 接口：<br>VOLUME_CONTROL | 5.2.1.2 |
| GET_MAX | 接口：<br>VOLUME_CONTROL | 5.2.1.2 |
| GET_RES | 接口：<br>VOLUME_CONTROL | 5.2.1.2 |
| GET_STAT | 保存状态信息的内容，以将来使用。现在，应返回在控制传输的数据阶段中的空数据包，并应确认接收的空数据包。 | 5.2.4.2 |

# USBFS MIDI

USBFS MIDI 支持与外部 MIDI 设备进行通信它也支持 MIDI 器件的 USB 器件类定义。通过使用该组件，可以将 MIDI I/O 功能添加到独立器件中，也可以通过电脑主机或移动器件的 USB 端口执行该主机和器件的 MIDI 功能。在这种情况下，电脑主机或移动器件将该组件视为类兼容的 USB MIDI 器件。该器件会使用主机中的本地 MIDI 驱动程序。

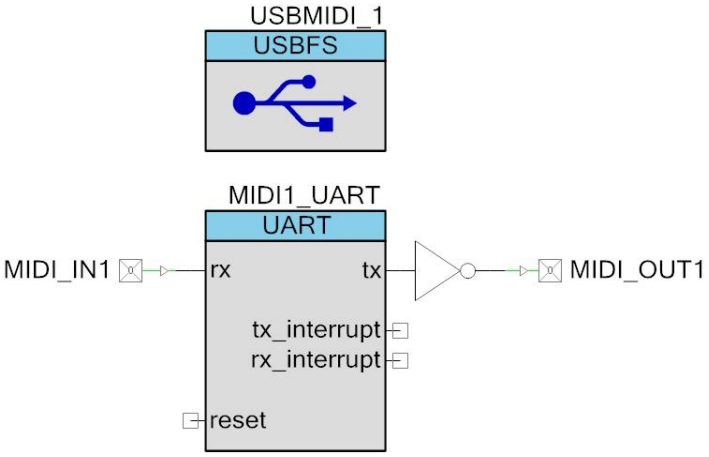
## 特性：

- 提供了 USB MIDI 类兼容的 MIDI 输入和输出。

- 支持硬件通过 UART 与外部 MIDI 设备进行连接。
- 提供了通过中断管理的可调发送和接收缓冲区。
- 处理接收和发送功能的 MIDI 运行状态。
- 通过使用虚拟线缆，支持多达 16 个输入和输出端口。这些端口只使用两个 USB 端点。

PSoC Creator 组件目录中包含 MIDI 接口的原理图宏实现。该宏包含了带硬件 MIDI 接口配置（31.25 kbps，8 数据位）的 UART 组件的实例，以及将描述符配置为支持 MIDI 器件的 USBFS 组件。这样，终端用户可在配置变化最小的情况下使用由 MIDI 使能的 USBFS 组件。

要启动基于 MIDI 的项目，将从组件目录中已标签“USBMIDI”的 USBMIDI 原理图宏拖放您的设计中。已经将该宏配置为带有一个输入和一个输出的外部模式 MIDI 器件。更多有关接口参数（如 VID、PID 以及字符串描述符）更改的信息，请参见本数据手册中组件参数部分的内容。




UART 组件可连接至数字输入和输出引脚的组件。通过“非”（NOT）门连接输出，以准备将反转后的信号提供给外部晶体管。更多有关硬件 MIDI 接口的信息，请参考 *MIDI 1.0 详细规范*中介绍的内容。

**要想更新带有两个输入和两个输出的外部模式 USBMIDI 原理图宏，请执行下面各操作：**

1. 打开 USBMIDI_1 组件中的 **MIDI Descriptor** 选项卡。
2. 点击 **Import MIDI Interface**（导入 MIDI 接口）按键，浏览到下面的目录，并打开 *USBMIDI 2x2.midi.xml* 文件。

   <INSTALL>\psoc\content\cycomponentlibrary\CyComponentLibrary.cylib\USBFS_v2.70\Custom\template\
3. 将组件目录中的 UART 原理图宏拖放到您的设计中。
4. 按下面选项配置 UART：

   名称： MIDI2_UART

   模式： 全双工 UART

| | |
|---|---|
| 每秒位数： | 31250 |
| 数据位： | 8 |
| 奇偶校验： | 无 |
| RX 缓冲区大小（字节） | 255 |
| TX 缓冲区大小（字节） | 255 |

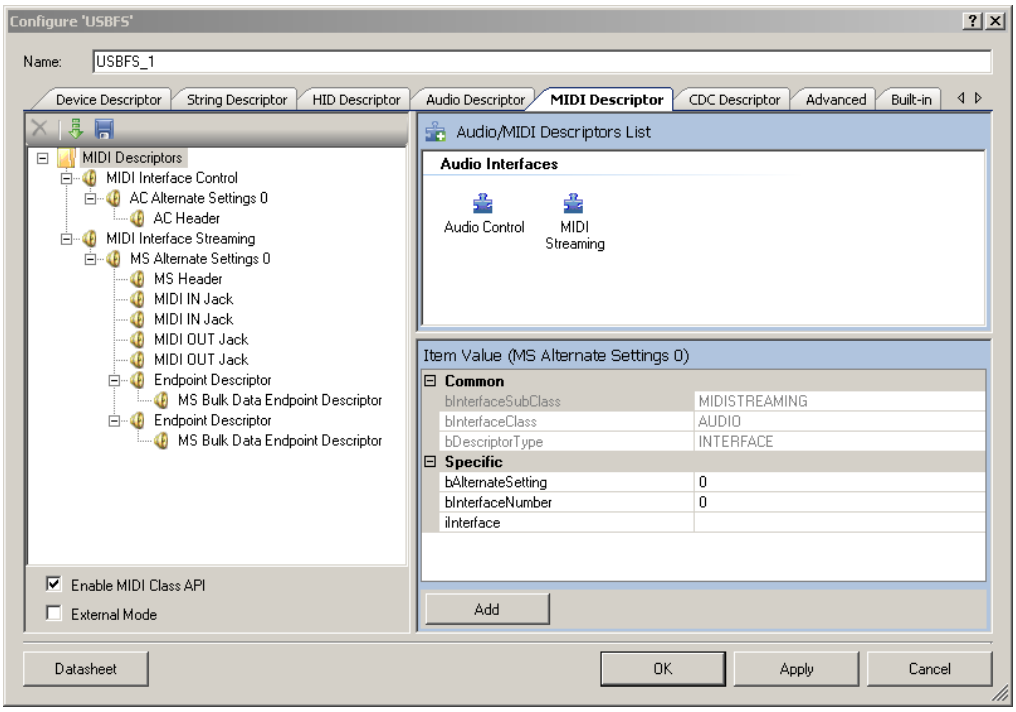
5. 通过 NOT（非）门连接输出引脚。

## USBFS MIDI 参数

将 USBMIDI 宏拖放到您的设计中，并双击它以打开 **Configure USBFS** 对话框。

### MIDI Descriptor 选项卡

**MIDI Descriptor** 选项卡用于添加和配置 MIDI 流接口描述符。




### 添加 MIDI Descriptors（MIDI 描述符）

1. 在左侧的目录中，选择 **MIDI Descriptors** 根项目。
2. 在右侧的 **Audio / MIDI Descriptors List** 下，选择 **Audio Control** 或 **MIDI Streaming** 接口。
3. 在 **Item Value** 下，适当输入 **bAlternateSetting** 和 **bInterfaceNumber** 值。其他字段都可选。

   **注意**这些值是手动设置的。但是对于一般接口描述符，这些值是自动设置的。

4. 点击 **Add** 以将描述符添加到左侧目录中。

   您可以通过选择节点并单击它来重命名“**MIDI 接口 x**”标题。

### 添加特定于类的音频控制或音频流接口描述符

1. 在左侧的目录中，适当选择 **AC Alternate Settings x** 或 **MS Alternate Settings x** 项目。
2. 在右侧的 **Audio / MIDI Descriptors List**（音频/MIDI 描述符列表）下，根据要求选择 **Audio Control Descriptors (1.0)**（音频控制描述符）、**Audio Control Descriptors (2.0)**（音频控制描述符）或 **MIDI Streaming Descriptors**（MIDI 流描述符）下的项目之一。

   有关版本 1.0 和 2.0 的信息，请参考相应的 *音频器件的通用串行总线器件类定义*规范文档。
3. 在 **Item Value** 下，输入 **Specific** 中的适当值。
4. 点击 **Add** 以将描述符添加到左侧目录中。

### 添加 MIDI 端点描述符

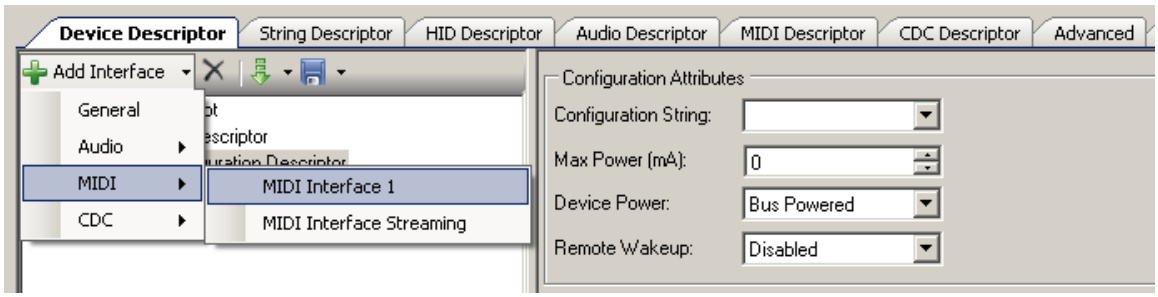
1. 在左侧的目录中，适当选择 **AC Alternate Settings x** 或 **MS Alternate Settings x** 项目。
2. 在右侧的 **Audio / MIDI Descriptors List** 下，选择 **Endpoint Descriptor** 项。
3. 在 **Item Value** 下，输入 **Specific** 中的适当值。
4. 点击 **Add** 以将描述符添加到左侧目录中。

### 要想添加标准的 MS 批量数据的端点描述符

1. 在左侧的目录中，选择适当的 **Endpoint Descriptor**（端点描述符）。
2. 在右侧的 **Audio / MIDI Descriptors List** 下，选择 **MS Endpoint Descriptor**（MS 端点描述符）。
3. 在 **Item Value** 下，输入 **Specific** 中的适当值。
4. 点击 **Add** 以将描述符添加到左侧目录中。

### 将配置的 MIDI 接口描述符添加到“Device Descriptor”目录中

1. 打开 **Device Descriptor** 选项卡。
2. 选择新接口所在的 **Configuration Descriptor**。
3. 点击 **Add Interface** 工具按键，选择 **MIDI**，并选择要添加的相应项目。




MIDI interfaces（MIDI 接口）在 **Device Descriptor** 选项卡列表中被禁用，因为它们只能在 **MIDI Descriptor** 选项卡中编辑。

**注意：**点击 **Apply** 或 **OK**，以保存各个选项卡的更改。如果点击 **Cancel**，将不保存已添加的所有描述符。

## USBFS MIDI 应用编程接口

默认情况下，PSoC Creator 将实例名称 USBFS_1 分配给指定设计中组件的第一个实例。您可以将其重新命名为任何一个符合标识符语法规则的值。实例名称会成为每个全局函数名称、变量和常量符号的前缀。出于可读性考虑，下表中使用的实例名称为 USBMIDI。

### MIDI 支持

当选中 **MIDI Descriptor** 选项卡中的 **Enable MIDI Class API**（使能 MIDI 类 API）项时，下面的高级 API 将可用。

| 函数 | 说明 |
|---|---|
| USBMIDI_MIDI_EP_Init() | 初始化MIDI接口和各个UART，使其准备好接收来自PC和MIDI端口的数据。 |
| USBMIDI_MIDI_IN_Service() | 服务USB MIDI IN端点。 |
| USBMIDI_MIDI_OUT_EP_Service() | 服务USB MIDI IN端点。 |
| USBMIDI_PutUsbMidiIn() | 将一条MIDI信息放置在USB MIDI IN端点缓冲区内。这是发送到主机的MIDI输入信息。 |
| USBMIDI_callbackLocalMidiEvent() | 此函数是来自*USBMIDI_midi.c*的回调函数，用于在*main.c*中进行本地处理。 |

### 全局变量

| 变量 | 说明 |
|---|---|
| USBMIDI_midiInBuffer | 长度等于MIDI IN EP **最大数据包大小**的输入端点缓冲区。此缓冲区用于保存并组合从UART接收的数据和/或由USBMIDI_PutUsbMidiIn()函数内部生成的信息。USBMIDI_MIDI_IN_Service()函数将数据从缓冲区传输到PC。 |
| USBMIDI_midiOutBuffer | 长度等于MIDI OUT EP**最大数据包大小**的输出端点缓冲区。此缓冲区由USBMIDI_MIDI_OUT_EP_Service()函数使用，以保存从PC接收的数据。然后对已接收的数据进行解析。解析后的数据被传输到UART缓冲区内，并且USBMIDI_callbackLocalMidiEvent()函数还将使用该数据进行内部处理。 |
| USBMIDI_midiInPointer | 输入端点缓冲区的指针。此指针是USBMIDI_midiInBuffer写入数据时所使用的索引。它可由USBMIDI_MIDI_EP_Init()函数清除至零。 |
| USBMIDI_midi_in_ep | 包含了midi IN端点编号。发生基于用户描述符的SET_CONFIGURATION请求后，此函数将被初始化。通过在MIDI API中使用此函数，可将数据发送给PC。 |
| USBMIDI_midi_out_ep | 包含midi OUT端点编号发生基于用户描述符的SET_CONFIGURATION请求后，此函数将被初始化。通过在MIDI API中使用此函数，可接收来自PC的数据。 |
| USBMIDI_MIDI1_InqFlags<br>USBMIDI_MIDI2_InqFlags | 当使能**External Mode**（外部模式）时，可以对这些可选变量进行分配。以下标志有助于检测和生成SysEx信息的响应。<br><br>**标志** / **说明**<br>USBMIDI_INQ_SYSEX_FLAG：所接收的非实时SysEx信息。<br>USBMIDI_INQ_IDENTITY_REQ_FLAG：接收到的身份请求。当生成身份回复信息时，应清除此位。 |

## void USBMIDI_MIDI_EP_Init(void)

**说明：** 此函数初始化MIDI接口和各个UART，使其准备好接收来自PC和MIDI端口的数据。

**参数：** 无

**返回值：** 无

**其他影响：** 更改UART的TX和RX中断的优先级。更多有关信息，请参见中断优先级部分的内容。

## void USBMIDI_MIDI_IN_Service(void)

**说明：** 此函数服务来是自MIDI输入端口（RX UART）的流量，或是由USBMIDI_PutUsbMidiIn()函数生成的。此外，还将数据发送到USBMIDI IN端点上。此函数是非封锁的，并且应该从主前台任务调用它。更多有关该API使用情况的信息，请参见USBFS MIDI功能说明部分的内容。

它不受可重入调用的保护。如果必须在UART RX ISR中使用此函数以保证短延时的话，则要阻止对此函数进行重入调用。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void USBMIDI_MIDI_OUT_EP_Service(void)

**说明：** 此函数服务来自USBMIDI OUT端口的流量，并将数据发送到MIDI输出端口（TX UART）。当UART TX缓冲区中没有足够空间时，UART会封锁此函数。

在带自动存储器管理的DMA模式中，此函数由OUT EP ISR自动调用。在手动模式和带手动EP管理的DMA模式中，必须从主前台任务调用此函数。

**参数：** 无

**返回值：** 无

**其他影响：** 无

## uint8 USBMIDI_PutUsbMidiIn(uint8 ic, const uint8 midiMsg[], uint8 cable)

**说明：** 此函数将一个MIDI信息置于USB MIDI IN端点缓冲区。这是发到主机的MIDI输入信息。只在器件有内部MIDI输入的功能时，才能使用此函数。此外，应该调用USBMIDI_MIDI_IN_Service()函数，以将信息从本地缓冲区发送到IN端点。

**参数：** uint8 ic：下表介绍的是MIDI信息或指令的长度。

| 数值 | 说明 |
|---|---|
| 0 | 无信息（始终不该发生） |
| 1-3 | 完成midiMsg中的MIDI信息 |
| 3 - IN EP**最大数据包大小** | 完成midiMsg中的SysEx信息（不带EOSEX字节） |
| USBMIDI_MIDI_SYSEX | SysEx信息的开始或延续。将事件字节放置于midiMsg缓冲区 |
| USBMIDI_MIDI_EOSEX | SysEx信息的末端。将事件字节放置于midiMsg缓冲区 |
| USBMIDI_MIDI_TUNEREQ | 调试请求信息（单字节系统通用信息） |
| 0xF8 到 0xFF | 单字节实时信息 |

const uint8 midiMsg[]：指向MIDI信息的指针

uint8 cable：线缆编号

**返回值：**

| 返回值 | 说明 |
|---|---|
| USBMIDI_TRUE | 主机尚未准备好接收该信息 |
| USBMIDI_FALSE | 成功传输 |

**其他影响：** 无

## void USBMIDI_callbackLocalMidiEvent(uint8 cable, uint8* midiMsg)

**说明：** 这是一个回调函数，用于对从*main.c*中的PC所接收的数据进行本地处理。根据您的要求执行此函数。此函数由USB输出处理子程序调用，使用于由输出端点缓冲区处理（解码）的每个MIDI输出事件。

**参数：** uint8 cable：线缆编号

uint8* midiMsg：指向3字节MIDI信息的指针

**返回值：** 无

**其他影响：** 无

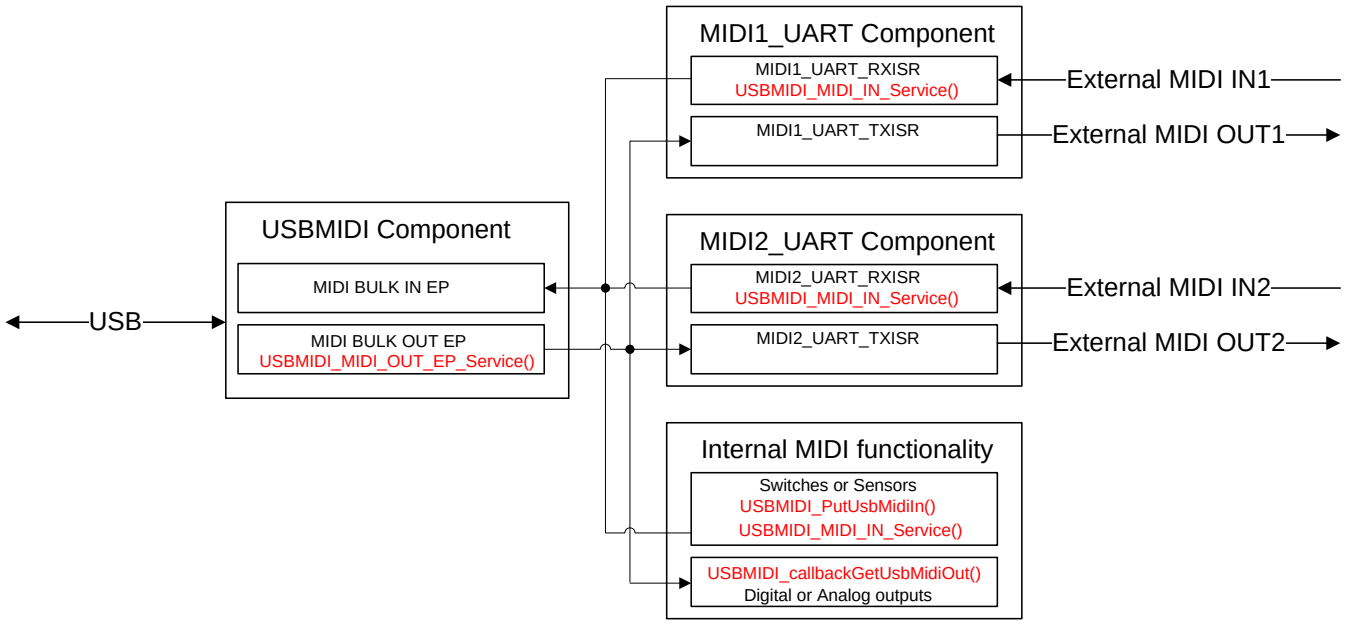
## USBFS MIDI 功能说明

通过“MIDI descriptor”选项卡，可以很容易地创建带有一组或多组物理 MIDI 端口的 MIDI 接口器件（可能需要放置和配置 UART 组件的各个实例）。该器件可处理所有向外部 MIDI 设备发送/接收 MIDI 信息的相关操作。这是一个外部 MIDI 功能，并且是组件中的一个可选设置。

当与外部 MIDI 设备通信时，MIDI 实现将内部处理运行状态。在输出上自动执行运行状态，以减少串行数据的流量。此外，当使用运行状态发送外部 MIDI 设备时，会在输入上管理运行状态，以准确汇编完整的 MIDI 信息。更多有关运行状态性能的信息，请参考 *MIDI 1.0 详细规范*。

图 3 显示的是带有两个输入和两个输出的外部模式 USB-MIDI 接口。

**图 3. 外部模式 USB-MIDI 接口**




要想执行外部功能，需要放置并配置两个名称分别为“MIDI1_UART”和“MIDI2_UART”的 UART 组件。通过这些硬编码名称，USBMIDI 组件可以调用 UART API 并自动将接收数据从主机信息传输到外部 MIDI 端口。在手动模式和带手动 EP 管理的 DMA 模式中，必须从主循环调用 USBMIDI_MIDI_OUT_EP_Service() API。

相反，在手动模式和带手动存储器管理的 DMA 模式下，要想从 UART 组件实现 MIDI 事件数据，必须调用主循环中的 USBMIDI_MIDI_IN_Service() API。在自动 DMA 模式下，从中断服务子程序的用户部分（MIDI[1..2]_UART_RXISR_END）（用于 UART（MIDI[1..2]_UART_RXISR）的 RX 部分）调用此函数。

您可以使用本地开关和传感器，为主机创建 MIDI 信息（通过使用 USBMIDI_PutUsbMidiIn()函数实现）。来自主机的 MIDI 信息可以直接控制本地函数，如数字和模拟输出（执行 USBMIDI_callbackLocalMidiEvent()函数。调用该函数可处理所有的接收信息）。

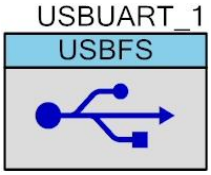
### 中断优先级

在 MIDI BULK OUT EP ISR 内服务从主机上接收到的数据。当您选择较小的 UART **TX 缓冲区大小**时，代码会等到 UART 传输操作完成后再继续填充 TX 缓冲区。UART 中 TX 部分的中断服务子程序优先级要高于 MIDI BULK OUT EP ISR 的。USBMIDI_MIDI_EP_Init()函数自动将所述的默认的中断的优先级更改为 USBMIDI_CUSTOM_UART_TX_PRIOR_NUM 值。赛普拉斯建议您将 UART **TX 缓冲区大小**设置为不小于 MIDI BULK OUT EP **最大数据包的大小**。最佳的 **Max Packet Size**（最大数据包大小）为 32。

UART RX ISR 的优先级需要高于 TX ISR 的优先级，以阻止过载硬件 FIFO 的四个字节。最佳的 UART **RX Buffer Size**（RX 缓冲区大小）为 255。USBMIDI_MIDI_EP_Init()函数自动将默认 UART RX 中断的优先级更改为 USBMIDI_CUSTOM_UART_RX_PRIOR_NUM 值。USBMIDI_MIDI_EP_Init()函数自动将 UART RX 中断的默认优先级更改为 USBMIDI_CUSTOM_UART_RX_PRIOR_NUM 值。

## USBUART

PSoC Creator 组件目录中包含 CDC 接口的原理图宏实现（又称为 USBUART）。这是一个 USBFS 组件，其描述符被配置以执行 CDC 接口。这样，用户可在要求配置最少的情况下使用由 CDC 使能的 USBFS 组件。



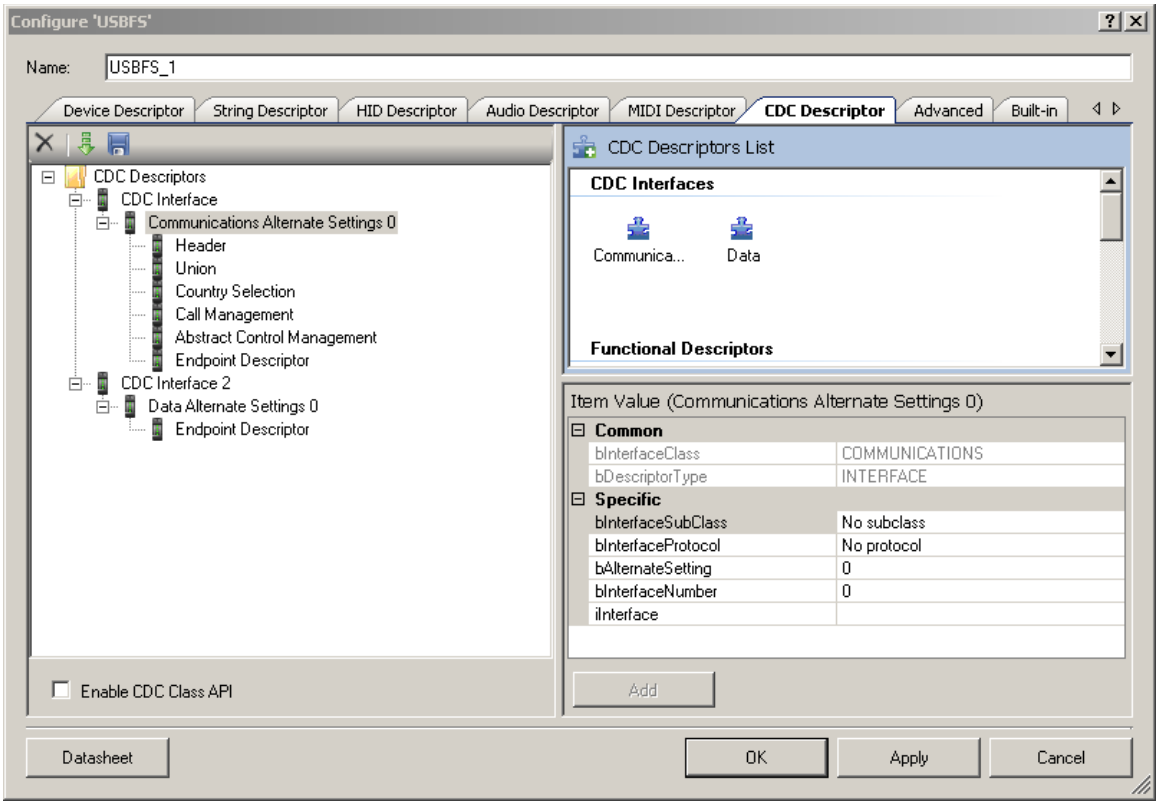

要想启动一个基于 USBUART 的项目，请将组件目录中标签为“USBUART（CDC 接口）”的 USBUART 原理图宏拖放到您的设计中。已经对此宏配置了一个 CDC 器件。更多有关修改该接口的参数信息，请参见改数据手册中组件参数部分的内容，如 VID、PID 以及 String Descriptors。

### USBUART 参数

将 USBUART 宏拖放到您的设计中，并双击它以打开 **Configure USBFS** 对话框。

### CDC Descriptor 选项卡

**CDC Descriptor** 选项卡用于添加和配置通信和数据接口描述符。




#### 添加 CDC Descriptors（CDC 描述符）

1. 在左侧目录中，选择 **CDC Descriptors** 根项目。
2. 在右侧的 **CDC Descriptors List** 下，选择 **Communications** 或 **Data** 接口。
3. 在 **Item Value** 下，适当输入 **bAlternateSetting** 和 **bInterfaceNumber** 值。其他字段都可选。

   **注意：**这些值是手动设置的。但是对于一般接口描述符，这些值是自动设置的。
4. 点击 **Add** 以将描述符添加到左侧目录中。
5. 您可以通过选择节点并单击它来重命名“**CDC 接口 x**”标题。

#### 添加功能描述符

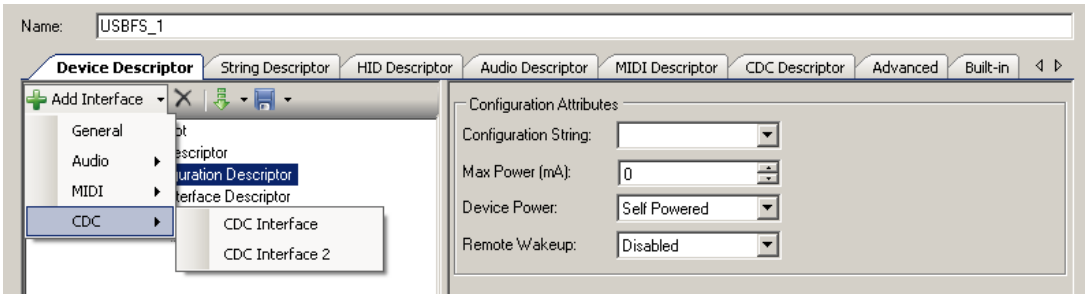
1. 在左侧的目录中，适当选择 **Communications Alternate Settings x**（通信备用设置 x）项。
2. 在右侧的 **CDC Descriptors List** 下，适当选择 **Functional Descriptors** 下的各项之一。
3. 在 **Item Value** 下，输入 **Specific** 中的适当值。
4. 点击 **Add** 以将描述符添加到左侧目录中。

### 添加端点描述符

1. 在左侧的目录中，适当选择 **Communications Alternate Settings x** 或 **Data Alternate Settings x** 项。
2. 在右侧的 **CDC Descriptors List** 下，选择 **Endpoint Descriptor** 项。
3. 在 **Item Value** 下，输入 **Specific** 中的适当值。
4. 点击 **Add** 以将描述符添加到左侧目录中。

### 将配置的 CDC 接口描述符添加到“Device Descriptor”目录中

1. 打开 **Device Descriptor** 选项卡。
2. 选择新接口所在的 **Configuration Descriptor**。
3. 点击 **Add Interface** 工具按键，选择 **CDC**，并选择要添加的相应项目。




CDC interfaces（CDC 接口）在 **Device Descriptor** 选项卡列表中被禁用，因为它们只能在 **CDC Descriptor** 选项卡中编辑。

**注意**点击 **Apply** 或 **OK**，以保存各个选项卡的更改。如果点击 **Cancel**，将不保存已添加的所有描述符。

## USBUART 应用编程接口

默认情况下，PSoC Creator 将实例名称“USBUART_1”分配给指定设计中组件的第一个实例。您可以将其重新命名为任何一个符合标识符语法规则的值。实例名称会成为每个全局函数名称、变量和常量符号的前缀。出于可读性考虑，下表中使用的实例名称为“USBUART”。

### CDC 类支持

选择 **CDC Descriptor** 选项卡中的 **Enable CDC Class API**（使能 CDC Class API）项时，以下高级 API 是可用的。这些 API 不支持带自动存储器管理的 DMA。

| 函数 | 说明 |
|---|---|
| USBUART_CDC_Init() | 初始化CDC接口，为从PC接收数据做出准备。 |
| USBUART_PutData() | 将指定的字节数量从指针指定的位置发送到PC。 |
| USBUART_PutString() | 将空结尾字符串发送到PC |
| USBUART_PutChar() | 将单字符写入PC内 |
| USBUART_PutCRLF() | 将回车符（0x0D）和换行符（0x0A）发送到PC。 |
| USBUART_GetCount() | 返回从PC接收的字节数。 |
| USBUART_CDCIsReady() | 如果组件准备好向PC发送更多的数据，则返回非零值。 |
| USBUART_DataIsReady() | 如果组件接收到了数据或接收到长度为零的数据包，则返回非零值。 |
| USBUART_GetData() | 从输入缓冲区获取指定的字节数，然后将其放置在由移动指针指定的数据阵列中 |
| USBUART_GetAll() | 从输入缓冲区获取所有的接收数据字节，然后将它们放置在指定的数据阵列中 |
| USBUART_GetChar() | 读取从缓冲区接收的数据中的一个字节 |
| USBUART_IsLineChanged() | 返回线路的清除读取状态 |
| USBUART_GetDTERate() | 返回该端口所设置的数据终端速率（单位：每秒位数） |
| USBUART_GetCharFormat() | 返回停止位数 |
| USBUART_GetParityType() | 返回CDC端口的奇偶校验类型 |
| USBUART_GetDataBits() | 返回CDC端口的数据位数 |
| USBUART_GetLineControl() | 返回行控制位图 |

### 全局变量

| 变量 | 说明 |
|---|---|
| USBUART_lineCoding | 包含当前行的编码结构。主机使用SET_LINE_CODING请求设置该变量，并使用USBUART_GetDTERate()、USBUART_GetCharFormat()、USBUART_GetParityType()以及USBUART_GetDataBits() API返回该变量给用户代码。 |
| USBUART_lineControlBitmap | 包含当前控制信号位图。主机使用SET_CONTROL_LINE请求设置此变量，并使用USBUART_GetLineControl() API返回该变量给用户代码。 |

| 变量 | 说明 |
|---|---|
| USBUART_lineChanged | 作为USBUART_IsLineChanged（）API的标志，用于通知该函数已向主机发送了一个请求，以更改行编码或控制位图。 |
| USBUART_cdc_data_in_ep | 包含数据IN端点编号。发生基于用户描述符的SET_CONFIGURATION请求后，此函数将被初始化。通过在CDC API中使用此函数，可将数据发送给PC。 |
| USBUART_cdc_data_out_ep | 包含数据OUT端点编号。发生基于用户描述符的SET_CONFIGURATION请求后，此函数将被初始化。通过在CDC API中使用此函数，可接收来自PC的数据。 |

## void USBUART_CDC_Init(void)

**说明：** 此函数初始化CDC接口，为从PC接收数据做出准备。需要在启动器件后调用此API函数。通过使用USBUART_Start() API配置该函数，以初始化和启动USBFS组件的操作。然后，调用USBUART_GetConfiguration() API，并等到主机枚举和配置完器件为止。例如：

```
USBUART_Start(…);
while(USBUART_GetConfiguration()){};
USBUART_CDC_Init();
```

**参数：** 无

**返回值：** 无

**其他影响：** 无

## void USBUART_PutData(const uint8* pData, uint16 length)

**说明：** 此函数将指定的字节数量从指向指定的位置发送到PC。发送新的数据前，应先调用USBUART_CDCIsReady()函数，以保证已发送完前一个数据。

如果发送的最后数据包大小小于最大数据包大小，该短数据包的USB传输将指定数据段的结束。如果发送的最后数据包大小正好等于最大数据包大小，其后面会是一个长度为零的数据包（即为短数据包），其确保适当指定了数据段的结束。为发送长度为零的数据包，请使用USBUART_PutData() API，其中长度参数被设置为零。

**参数：** const uint8* pData：指向缓冲区（包含将要发送的数据）的指针

uint16 length：指定从pData缓冲区发送的字节数。最大长度为64个字节。如果数据长度超过了最大数据包的大小，则数据将被丢失。

**返回值：** 无

**其他影响：** 无

### void USBUART_PutString(const char8* string[])

**说明：** 此函数将空结尾字符串发送到PC 如果存储器中没有足够空间用于放置整个字符串，则此函数将被阻塞。此函数一直被阻塞，直到整个字符串被写入发送缓冲区为止。通过调用USBUART_PutString()发送数据之前，应先调用USBUART_CDCIsReady()函数，以保证已发送完前一数据。如果由该API发送的最后数据包长度等于 **最大数据包大小**，该函数将自动发送长度为零的数据包。

**参数：** const char8 string[]：指向要发送至PC的字符串的指针。

**返回值：** 无

**其他影响：** 无

### void USBUART_PutChar(char8 txDataByte)

**说明：** 此函数每次只向PC写入单个字符。它并不是发送大量数据的有效方法。

**参数：** char8 txDataByte：将要发送到PC的字符

**返回值：** 无

**其他影响：** 无

### void USBUART_PutCRLF(void)

**说明：** 此函数将回车符（0x0D）和换行符（0x0A）发送到PC。此API用于模仿由其他UART组件所提供的API。

**参数：** 无

**返回值：** 无

**其他影响：** 无

### uint16 USBUART_GetCount(void)

**说明：** 此函数返回来自PC接收的字节数。将所返回的长度值作为一个参数，并将它传输给USBUART_GetData()，以读取所有的接收数据。如果USBUART_GetData（）API不能一次读取所有的接收数据，那么尚未读取的数据将被丢失。

**参数：** 无

**返回值：** uint16：返回已接收的字节数。一次最多可以接收64个数据字节。

**其他影响：** 无

## uint8 USBUART_DataIsReady(void)

**说明：** 如果组件接收到了数据或接收到长度为零的数据包，则此函数返回非零值。即使接收到长度为零的数据包，仍应该调用USBUART_GetAll()或USBUART_GetData() API来读取缓冲区中的数据，并重新初始化OUT端点。当接收到长度为零的数据包时，这些API将返回零值。

**参数：** 无

**返回值：** uint8：如果接收到OUT数据包，此函数会返回非零值。否则，它将返回零值。

**其他影响：** 无

## uint8 USBUART_CDCIsReady(void)

**说明：** 如果组件准备好向PC发送更多的数据，则此函数返回非零值；否则，将返回零值。当使用下列各函数（API（USBUART_PutData()、USBUART_PutString()、USBUART_PutChar或USBUART_PutCRLF()）中的一个时，在发送新的数据前，需要先调用此函数，以确保前一个数据已被发送完毕。

**参数：** 无

**返回值：** uint8：如果缓冲区可以接受新数据，则此函数将返回非零值。否则，它将返回零值。

**其他影响：** 无

## uint16 USBUART_GetData(uint8* pData, uint16 length)

**说明：** 此函数从输入缓冲区获取指定的字节数，然后将其放置在由移动指针指定的数据阵列中。应先调用USBUART_DataIsReady() API，以确保从主机接收数据。如果一次不能读取所有的接收数据，尚未读取的数据将被丢失。通过调用USBUART_GetData() API，获取已接收的字节数。

**参数：** uint8* pData：指向将放置数据区域的数据阵列的指针

uint16 length：字节数在RX缓冲区的数据阵列中读取。最大长度不超过所接收的字节数或64个字节。

**返回值：** uint16：函数从端点RAM移到数据阵列中的字节数。如果主机发送的字节数少于所请求的字节数，或发送的数据包长度为零，则此函数移动的字节数将少于请求的字节数。

**其他影响：** 无

### uint16 USBUART_GetAll(uint8* pData)

**说明：** 此函数获取从输入缓冲区的接收数据的所有字节，并将其置于指定数据数组中应先调用USBUART_DataIsReady() API，以确保数据从主机被接收。

**参数：** uint8* pData：指向将放置数据区域的数据阵列的指针。

**返回值：** uint16：已接收的字节数。一次最多可以接收64个数据字节。

**其他影响：** 无

### uint8 USBUART_GetChar(void)

**说明：** 此函数读取从缓冲区接收数据的一个字节。如果从主机接收的数据大小超过了一个字节，则剩余的数据将被丢失。

**参数：** 无

**返回值：** uint8：已接收到一个字符

**其他影响：** 无

### uint8 USBUART_IsLineChanged(void)

**说明：** 此函数将返回线路的清除状态。当主机向器件发送更新的编码或控制信息时，此函数会返回非零值。应通过调用USBUART_GetDTERate()、USBUART_GetCharFormat()或USBUART_GetParityType()或USBUART_GetDataBits() API来读取数据编码的信息。另外应该调用USBUART_GetLineControl() API读取行控制的信息。

**参数：** 无

**返回值：** uint8：如果接收到SET_LINE_CODING或CDC_SET_CONTROL_LINE_STATE请求，此函数将返回非零值。否则，此函数将返回零值。

| 返回值 | 说明 |
|---|---|
| USBUART_LINE_CODING_CHANGED | 更改后的行编码 |
| USBUART_LINE_CONTROL_CHANGED | 更改后的行控制 |

**其他影响：** 无

### uint32 USBUART_GetDTERate(void)

**说明：** 此函数返回该端口所设置的数据终端速率（单位：每秒位数）。

**参数：** 无

**返回值：** uint32：返回数据速率值（单位：每秒位数）

**其他影响：** 无

## uint8 USBUART_GetCharFormat(void)

**说明：** 此函数返回停止位数。

**参数：** 无

**返回值：** uint8：返回停止位数。

| 返回值 | 说明 |
|---|---|
| USBUART_1_STOPBIT | 一个停止位 |
| USBUART_1_5_STOPBITS | 一个半停止位 |
| USBUART_2_STOPBITS | 两个停止位 |

**其他影响：** 无

## uint8 USBUART_GetParityType(void)

**说明：** 此函数返回CDC端口的奇偶校验类型。

**参数：** 无

**返回值：** uint8：

| 返回值 | 说明 |
|---|---|
| USBUART_PARITY_NONE | 无 |
| USBUART_PARITY_ODD | 奇校验 |
| USBUART_PARITY_EVEN | 偶校验 |
| USBUART_PARITY_MARK | 标记 |
| USBUART_PARITY_SPACE | 空格 |

**其他影响：** 无

## uint8 USBUART_GetDataBits(void)

**说明：** 此函数返回CDC端口的数据位数。

**参数：** 无

**返回值：** uint8：返回数据位数。数据位数可以是5、6、7、8或16。

**其他影响：** 无

### uint16 USBUART_GetLineControl(void)

**说明：** 此函数会返回主机发送到器件的行控制位图。

**参数：** 无。

**返回值：** uint8：

| 返回值 | 注释 |
|---|---|
| USBUART_LINE_CONTROL_DTR | 表示存在DTR信号。此信号与V.24信号108/2和RS232信号DTR相对应。 |
| USBUART_LINE_CONTROL_RTS | 半双工调制解调器的载波控制。此信号与V.24信号105和RS232信号RTS相对应。 |
| RESERVED | 保留剩余的各个位。 |

**注意：** 某些终端仿真程序不能正确处理这些控制信号。只当RTS信号更改了状态时，这些程序才更新DTR和RTS状态的信息。

**其他影响：** 无

## USBUART 功能说明

### CDC 类请求

本节介绍的是 USBUART 组件支持的请求。如果不支持请求，USBUART 组件响应 STALL，表示错误的请求。

| 类请求 | USBUART组件支持说明 | PSTN器件的通信类子类规范 |
|---|---|---|
| SET_LINE_CODING | 允许主机指定典型的异步行字符格式的属性，如：数据终端速率、停止位数、奇偶类型以及数据位数。该请求适用于在主机和器件之间进行数据传输。 | 6.3.10 |
| GET_LINE_CODING | 允许主机查找当前所配置的行编码。 | 6.3.11 |
| SET_CONTROL_LINE_STATE | 生成RS-232/V.24型的控制信号 — RTS和DTR。 | 6.3.12 |

不受同 USBUART 有关的 USBUART 组件请求的支持：

| 类请求 | 说明 | PSTN器件的通信类子类规范 |
|---|---|---|
| SERIAL_STATE | 允许主机读取载波检测（CD）、DSR、中断以及振铃信号（RI）的当前状态。 | 6.5.4 |

# 代码示例（CE60246）USBUART 移植

在 USBFS v2.0 组件中添加 USBUART CDC 支持（在 PSoC Creator 2.0 或更新版本中可用）前，USBUART 组件已作为 PSoC® *3 / PSoC 5* 中 CE60246 - USBUART 内的代码示例组件。由于此 USBUART 代码示例不再受支持，所以建议您移植官方组件。本节详细介绍了移植过程所涉及的步骤。

## 原理图

1. 在 PSoC Creator2.0 或最新版本中，打开现有的设计。
2. 在现有的 USBUART 组件中，记录其组件名称、供货商 ID、产品 ID、器件版本、制造商字符串以及产品字符串。
3. 删除现有的 USBUART 组件。
4. 将 PSoC Creator 组件目录中的“USBUART（CDC 接口）”组件放置到您的设计中。
5. 打开新组件，并使用先前 USBUART 设计中所记录的参数来配置此组件。更多有关如何将 VID、PID 以及各种器件字符串输入新组建的信息，请参考本数据手册中的组件参数部分的内容。

## API

表 1 是从 CE60246 USBUART 移植到 USBUART 的 USBFS v2.0+版本时所需要的 API 更改概述。由于只进行了少量更改，因此可以尽可能减小对现有项目的影响。注意，USBUART 的 USBFS v2.0+版本包含了 CDC 特定 API 的更多选择（请参见本数据手册中的 CDC 类支持 API 列表）。

**表 1. API 移植**

| CE60246 API | USBFS v2.0+ API | 移植过程所需的更改 |
|---|---|---|
| void USBUART_1_Init(void) | void USBUART_1_CDC_Init(void) | ▪ API 名称更改 |
| uint8 USBUART_1_bGetRxCount (void) | uint16 USBUART_1_GetCount(void) | ▪ API 名称更改<br>▪ 将返回值从 uint8 修改为 uint16 |
| void USBUART_1_ReadAll(uint8* pData) | uint16 USBUART_1_GetAll(uint8* pData) | ▪ API 名称更改<br>▪ 将返回值从 void 修改为 uint16 |
| void USBUART_1_Write(uint8 *pData, uint8 bLength) | void USBUART_1_PutData(const uint8* pData, uint16 length) | ▪ API 名称更改<br>▪ 将长度参数类型从 uint8 修改为 uint16 |
| uint8 USBUART_1_bTxIsReady(void) | uint8 USBUART_1_CDCIsReady(void) | ▪ API 名称更改 |

**注意：**此表假定组件名称为 USBUART_1。

# 资源

USB 作为固定功能模块实现。该组件使用 6 个中断和两个引脚。

# API 存储器使用情况

根据编译器、器件、所使用的 API 数量以及组件的配置情况的不同，组件所用的存储器大小也不一样。下表提供了在某一器件配置中的所有 API 使用的存储器大小。

下表中的存储器大小是在将相应编译器设置为 Release（发布）模式并且优化选项为 Size 的情况下测得的。对于特定的设计，分析编译器生成的映射文件后可以确定存储器的使用情况。

| 配置 | PSoC 3（Keil_PK51） | | PSoC 5LP（GCC） | |
|---|---|---|---|---|
| | 闪存 字节 | SRAM 字节 | 闪存 字节 | SRAM 字节 |
| 默认USBFS | 8542 | 222 | 5795 | 242 |
| 最大值，带有HID、CDC、音频、MIDI和Bootloader | 19938 | 440 | 10866 | 468 |

# 直流和交流的电气特性

除非另有说明，否则这些规范的适用条件是：-40 °C ≤ TA ≤ 85 °C 且 TJ ≤ 100 °C。除非另有说明，否则这些规范的适用范围为 1.71 V 到 5.5 V。

## USB 直流规范

| 参数 | 说明 | 条件 | 最小值 | 典型值 | 最大值 | 单位 |
|---|---|---|---|---|---|---|
| $V_{USB_5}$ | 为USB操作提供的器件供电 | 配置了USB并使能了USB调压器 | 4.35 | – | 5.25 | V |
| $V_{USB_3.3}$ | | 配置了USB，但不使用USB调压器 | 3.15 | – | 3.6 | V |
| $V_{USB_3}$ | | 配置了USB，但不使用USB调压器 | 2.85 | – | 3.6 | V |
| $I_{USB_Configured}$ | 在器件活动模式下，器件供电电流，总线时钟和IMO = 24 MHz | $V_{DDD}$ = 5 V | – | 10 | – | mA |
| | | $V_{DDD}$ = 3.3 V | – | 8 | – | mA |

| 参数 | 说明 | 条件 | 最小值 | 典型值 | 最大值 | 单位 |
|---|---|---|---|---|---|---|
| $I_{USB_Suspended}$ | 在器件睡眠模式下，器件供电电流 | $V_{DDD}$ = 5 V，连接到USB主机，PICU配置为在有USB恢复信号时唤醒 | – | 0.5 | – | mA |
| | | $V_{DDD}$ = 5 V，断开与USB主机的连接 | – | 0.3 | – | mA |
| | | $V_{DDD}$ = 3.3 V，连接到USB主机，PICU配置为在有USB恢复信号时唤醒 | – | 0.5 | – | mA |
| | | $V_{DDD}$ = 3.3 V，断开与USB主机的连接 | – | 0.3 | – | mA |

### USB 驱动程序交流规范

| 参数 | 说明 | 条件 | 最小值 | 典型值 | 最大值 | 单位 |
|---|---|---|---|---|---|---|
| Tr | 跃变上升时间 | | – | – | 20 | ns |
| Tf | 跃变下降时间 | | – | – | 20 | ns |
| TR | 上升/下降时间匹配 | | 90% | – | 111% | |
| $V_{CRS}$ | 输出信号交变电压 | | 1.3 | – | 2 | V |

# 组件更改

本节列出了各版本的主要组件更改内容。

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| 2.70 | 纠正了GET_DESCRIPTOR请求的响应。如果器件不支持所请求的描述符，它会响应一个请求错误。 | USB指令验证版本1.4.10.2不通过第9章和HID测试。 |
| | 在带自动存储器管理的DMA模式中，修复了IN端点数据操作的罕见故障。 | 还纠正了其他的硬件资源。在Device Descriptor选项卡上，可通过表达式视图中的epDMAautoOptimization参数优化资源。仅在器件中存在单个IN端点时，才将该参数值设置为真。 |
| | 在具有下面默认配置的条件下初始化USBUART_lineCoding阵列：115200波特、8数据位、无奇偶校验、一个停止位。 | 第一个GET_LINE_CODING请求没有将合适的配置发送到终端软件。 |

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| | 如果由该API发送的最后数据包长度等于最大数据包大小，修改该USBUART_PutString() API，以自动发送长度为零的数据包。 | 如果发送的最后数据包大小正好等于最大数据包大小，其后面会是一个长度为零的数据包。 |
| | 更新了USBFS_LoadInEP() API说明。 | 阐明该过程，以设置数据就绪状态。 |
| 2.60 | 添加了接口关联描述符的支持。 | 执行了接口关联描述符，如*USB ECN：接口关联描述符*文档中所述。 |
| | 添加了HID模板特性的快速导入。 | 可用性的改进。 |
| | 添加了导入HID报告描述符的可能性，该描述符是通过USB-IF HID描述符工具创建的。 | |
| | 更新了“MISRA符合性”章节。 | 根据MISRA-C:2004编码准则符合性验证组件，此外，该组件还有它的特定偏差。 |
| | 添加了可选的vbusdet输入。 | 此输入提供了连接至VBUS的功能，用于监控电源。 |
| | 修复了接口描述符的接口协议字段的不可访问性，同时也修复了某些情况下端点描述符的同步类型和用法类型等字段的不可访问性。 | |
| 2.50 | 通过树形目录中的上下文菜单，即“Rename”指令，可在定制器中编辑HID报告名称及注释。 | 简化HID报告描述符的编辑。 |
| | 修复了USBFS_SetEndpointHalt()和USBFS_ClearEndpointHalt()函数。 | 该修复操作允许在主机清除ENDPOINT_HALT特定时仍继续传输数据。 |
| | 新增MISRA符合性一节。 | 未证明该组件符合MISRA-C:2004编码准则。 |
| 2.40 | 在带自动存储器管理的DMA模式中，修复了IN端点数据操作的罕见故障以及动态端点的重新配置。 | |
| | 在器件选项卡下的器件描述符内，添加了用于显示器件编号的字段。 | 该值作为USBFS_Start（）API的参数使用。 |
| | 为PSoC 5芯片添加了带手动存储器管理的DMA支持。 | |
| | 当批量端点MaxPacketSize（最大数据包大小）的值不是{8、16、32、64}时，会生成DRC错误。 | 此外，还可以在定制器中检查该错误。如果输入了错误的值，则用户不能关闭定制器。 |
| | 为HID描述符添加了多个报告ID的支持。 | |
| 2.30 | 修复了在将SET_INTERFACE请求发送到与受影响端点无关的接口时发生意外的端点上进行的重新配置。 | 带有多个接口和备用设置的器件只需要重新配置SetInterface请求时所要求的端点。 |
| | 已将用户代码部分（`#START`...`#END`）添加到SET_CUR/GET_CUR音频类请求处理程序内。 | 允许用户通过对多通道音频音量控制的支持来更新音量控制请求处理程序。 |

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| 2.20 | 添加PSoC 5LP芯片支持。 | |
| | 更新了特性数据。 | |
| | 对数据手册进行了少量编辑。 | |
| 2.12 | 添加了MIDI器件支持：<br>■ 添加了新的“MIDI Descriptor”选项卡。用户可通过此选项卡配置 MIDI 描述符。<br>■ 可选的高级 API。 | 已根据*MIDI器件v1.0的通用串行总线器件类定义*文档执行了MIDI接口。 |
| | 为只带PSoC 5的器件添加了USBFS_Resume_Condition() API函数，用于检查恢复条件。 | PSoC 5器件没有用来检查唤醒条件的PICU唤醒源和标准D+引脚API。此函数通过USBIO模块读取D+引脚电平，并返回恢复条件。 |
| | 重组数据手册。 | |
| 2.11 | 向.cyre 文件中包括的所有 USBFS API添加了CYREENTRANT关键词。 | 并非所有API都是真正可重入的函数。组件API源文件中的注释指出了适用的函数。<br>对于采用了安全方式并且是不可重入的函数，则需要该项变更，这样可以消除编译器警告：通过标志或关键节防止同时调用。 |
| | 在执行同步端点上的IN数据传输时，始终将数据切换设置为DATA0。 | 根据USB 2.0规范中有关描述同步数据传输的内容，全速器件只应发送数据包中的DATA0 PID。 |
| | 修复了Stop_DMA函数，以释放用于操作模式3的所有DMA TD端点。 | 此函数只能停止一个通道。 |
| | 已将VBUS监控输入引脚的默认驱动器模式更改为高阻抗模式，并取消了此引脚的API生成抑制。 | 通过更改，可以降低低功耗项目的功耗。 |
| 2.10 | 修复了USBFS_DispatchClassRqst()函数中特定类请求的处理程序。 | 停止了Audio（音频）请求。 |

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| 2.0 | 添加了下面的CDC类支持：<br>■ 添加了新的 CDC Descriptor 选项卡。用户可通过此选项卡配置 CDC 描述符。<br>■ SET_LINE_CODING/GET_LINE_CODING CLR_CUR/SET_CONTROL_LINE_STATE CDC 类请求支持。<br>■ 可选高级 API。 | 已根据*通信器件v1.2的USB类定义*中第四节的内容执行CDC接口。 |
| | 添加了音频类2.0类支持。<br>在“Audio”选项卡中，添加了可用描述符的两个新组。<br>其名称分别为Audio Control Descriptors (2.0)（音频控制描述符（2.0））和Audio Streaming Descriptors (2.0)（音频流描述符（2.0））。<br>将Audio Control Descriptors和Audio Streaming Descriptors等现有的组被重新命名为<br>Audio Control Descriptors (1.0)（音频控制描述符（1.0））和Audio Streaming Descriptors (1.0)（音频流描述符（1.0））。 | 新描述符代表*音频器件版本2.0规范的USB器件类定义*。 |
| | 添加了DMA传输实现：<br>■ Mode2：带有手动存储器管理的手动 DMA<br>■ Mode3:带有自动存储器管理的自动 DMA<br>■ 添加了 USBFS_InitEP_DMA() API。<br>■ 修改了 USBFS_LoadInEP()/USBFS_ReadOutEP() API，以支持 DMA 传输。 | 进行DMA数据传输时，不使用CPU。 |
| | 添加了USBFS_IsConfigurationChanged()函数。 | Win 7操作系统为相同的配置编号发送两个SET_CONFIGURATION请求。在这种情况下，用户级代码需要在每个请求后重新使能OUT端点。<br>此函数用于检测是否已经通过PC更改了配置。如果此函数返回的数值非零，则可以使用USBFS_GetConfiguration() API获取配置编号。<br>主循环中使用的模型：<br>`if(USBFS_IsConfigurationChanged() != 0)`<br>`{`<br>`  if(USBFS_GetConfiguration() != 0)`<br>`  {`<br>`  USBFS_EnableOutEP(OUT_EP);`<br>`  }`<br>`}` |

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| | 修复了有关从睡眠模式唤醒的问题。 | 由于USB_BUS_RST_CNT寄存器是非保留的，所以退出睡眠模式后需要重新加载该寄存器，以获取PSoC 3 ES2和PSoC 5芯片的正确USB枚举。 |
| | 已将端点存储器管理组框从器件选项面板转移到根器件选项面板内。 | 整个配置的端点存储器管理设置应该为全局设置。<br>在以前的版本中，每个设备描述符单独使用这些设置。 |
| 1.60 | 添加了USBFS_TerminateEP(uint8 ep)函数，以便NAK（否认）端点。 | 可以在端点重新配置或器件模式切换之前使用此函数。 |
| | 将USBFS_hidProtocol变量初始化为USBFS_InitComponent()和USBFS_reInitComponent() 函数中的HID_PROTOCOL_REPORT值。 | 为了符合 HID“7.2.6 Set_Protocol 请求“--- “初始化时，所有器件默认为报告协议”。 |
| | 添加了发往接口的SET_FEATURE/CLR_FEATURE请求的支持。 | 为了通过WHQL测试。 |
| | 将逻辑添加到SET_IDLE请求处理，以支持正确的时序。 | 为了符合HID “7.2.4 Set_Idle请求” |
| | 添加了音频类请求的支持：SET_CUR/CLR_CUR到用于采样频率、静音以及音量控制的接口和端点。 | 为了符合音频类定义“5.2.1.1设置请求”和“5.2.1.2获取请求” |
| | 重命名了Bootloader API，这样可以先获取实例名称。添加了向后兼容定义。 | 准备将来能够从多个接口引导。 |
| | 向数据手册中添加了特性数据 | |
| | 对数据手册进行了少量编辑和更新 | |
| 1.50.a | 进行了数据手册日志累积更改 | 为了给客户提供方便。 |
| 1.50 | 已添加了USB暂停、恢复以及远程唤醒功能。 | USB器件应当支持暂停和恢复功能。 |
| | 重新命名大多数API以删除匈牙利的标记，支持旧名称以满足向后兼容。 | 为了符合公司代码标准。 |
| | 已添加GET_INTERFACE/SET_INTERFACE请求支持。 | 如果器件具有接口的备用设置，则必须支持GetInterface/SetInterface请求。 |
| | 集成了各个特定API，以支持Bootloader：CyBtldrCommStart、CyBtldrCommStop、CyBtldrCommReset、CyBtldrCommWrite、CyBtldrCommRead。 | USB可以作为具有此功能的Bootloader的通信组件使用。 |
| | 向数据手册添加了通用USB批量包裹传输示例。 | 为用户描述了通用USB用法。 |
| | 在Configure对话框的Advanced选项卡中添加了extern_cls 和 extern_vnd 参数。 | 这些参数在解决方案级别中使能其他组件，以提供它们对供货商和类请求本身的处理。 |

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| | 向“带有手动存储器管理的DMA”部分添加了限制。 | 此限制说明如何正确使用模式2/3传输。 |
| | 修改了Advanced选项卡布局。 | 数据网格替换为带有每个参数信息的复选框，以提高可用性复选框。 |
| | 在Configure对话框中添加了Audio Descriptors选项卡。 | 这样，可以为您的组件添加和配置音频描述符。 |
| | 从Start/Stop API删除了SOF ISR使能/禁用。 | 每1毫秒发生一次SOF中断，但是组件不使用此中断。如果应用程序需要此中断，可以通过调用以下函数使能它：<br>CyIntEnable(USBFS_SOF_VECT_NUM); |
| 1.30.b | 向组件中添加了信息，以说明它与芯片修订版的兼容性。 | 如果组件在不兼容的芯片上使用，该工具将报告错误/警告。如果发生此情况，请更新到支持您的目标器件的修订版。 |
| 1.30.a | 已将本地参数移动到形式参数列表。 | 为了解决PSoC Creator v1.0 Beta 4.1和更早版本中存在的缺陷，更新了组件，以使它可以继续在该工具的较新版本中使用。此组件使用了本地参数（这些参数不向用户公开），以在用户输入时执行后台计算。这些参数已更改为可见的形式参数，但是不可编辑。组件没有功能上的更改，但是现在可在“customizer”对话框的“expression view”（表达式视图）中看到受影响的参数。 |
| 1.30 | 更新了Configure对话框和数据手册。 | 在Configure对话框的Advanced选项卡中添加了Enable SOF Output(使能SOF输出)参数。<br>更新了数据手册中的USBFS_ReadOutEP()函数以反映正确的返回值。 |
| 1.20.b | 向组件中添加了信息，以说明它与芯片修订版的兼容性。 | 如果组件在不兼容的芯片上使用，该工具将报告错误/警告。如果发生此情况，请更新到支持您的目标器件的修订版。 |
| 1.20.a | 已将本地参数移动到形式参数列表。 | 为了解决PSoC Creator v1.0 Beta 4.1和更早版本中存在的缺陷，更新了组件，以使它可以继续在该工具的较新版本中使用。此组件使用了本地参数（这些参数不向用户公开），以在用户输入时执行后台计算。这些参数已更改为可见的形式参数，但是不可编辑。组件没有功能上的更改，但是现在可在Customizer对话框的Expression view（表达式视图）中看到受影响的参数。 |
| 1.10.b | 向组件中添加了信息，以说明它与芯片修订版的兼容性。 | 如果组件在不兼容的芯片上使用，该工具将报告错误/警告。如果发生此情况，请更新到支持您的目标器件的修订版。 |

| 版本 | 更改说明 | 更改原因/影响 |
|---|---|---|
| 1.10.a | 已将本地参数移动到形式参数列表。 | 为了解决PSoC Creator v1.0 Beta 4.1和更早版本中存在的缺陷，更新了组件，以使它可以继续在该工具的较新版本中使用。此组件使用了本地参数（这些参数不向用户公开），以在用户输入时执行后台计算。这些参数已更改为可见的形式参数，但是不可编辑。组件没有功能上的更改，但是现在可在Customizer对话框的Expression view（表达式视图）中看到受影响的参数。 |

© 赛普拉斯半导体公司，2014. 此处，所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品内嵌的电路以外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不根据专利或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于合理预计会发生运行异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯将不批准将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC®是赛普拉斯半导体公司的注册商标，PSoC Creator™和 Programmable System-on-Chip™是赛普拉斯半导体公司的商标。该处引用的所有其他商标或注册商标归其各自所有者所有。

所有源代码（软件和/或固件）均归赛普拉斯半导体公司（赛普拉斯）所有，并受全球专利法规（美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和/或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定的用途外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对该材料提供任何类型的明示或暗示保证，包括（但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不另行通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于合理预计可能发生运转异常和故障，并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而导致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用的赛普拉斯软件许可协议限制。