嵌入式协议栈 μC/USB-Device 应用开发
——基于 STM32 微控制器

μC/USB Device：Universal Serial Bus Device Stack
Application Development for the STM32

张爱华　韩志华　编著
何小庆　审校

北京航空航天大学出版社

目　　录

第 1 章

简 介

USB 接口是计算机系统历史上最成功的通信接口，并已成为连接计算机外设的事实标准。本书是《嵌入式协议栈 μC/USB-Device》的第二部分。本书的内容以 μC/Eval-STM32F107 开发板和 Micriμm 的 μC/USB-Device 协议栈为基础，演示了如何通过一个成熟的硬件和软件平台的组合，构建一个 USB 设备。Micriμm 的 μC/USB-Device 协议栈是专门针对嵌入式系统设计的 USB 设备协议栈。依靠 Micriμm 团队的共同努力，高质量、可伸缩和高可靠性的代码经过了非常严格的验证过程，μC/USB-Device 符合 USB 2.0 规范。μC/Eval-STM32F107 开发板的核是 ARM Cortex-M3 MCU，STM32F107 运行的时钟频率最高可达 72MHz，包含了高性能外设，如 10/100M 以太网 MAC、全速 USB OTG 接口、CAN 控制器、定时器和 UART 等。

本书中的例程涵盖了 USB 设备的最基本的功能，可以帮助读者理解《嵌入式协议栈 μC/USB-Device》一书中涵盖的 USB 概念，同时，它们还提供了一个快速构建 USB 设备的框架，例如：

- USB 转串口适配器(通信设备类)；
- 鼠标或键盘(人机接口设备类)；
- 移动存储设备(大容量存储类)；
- USB 医疗设备(个人健康设备类)；
- 定制设备(供应商类)。

图 1-1 展示了运行本书中 5 个例程所需的基本要素。

F1-1(1)为了运行书中提供的 USB 设备应用例程，读者需要一台运行 Windows 7 及以上系统的电脑，电脑至少有三个 USB A 型接口。一个 USB 接口用来连接仿真器，其它两个 USB 接口用于连接到 USB 协议分析仪，捕获 USB 通信流用于协议分析。

F1-1(2)μC/Eval-STM32F107 开发板中国版可以从北京麦克泰软件技术有限公司购买。开发板包含两个 USB 端口。一个 USB 端口用于开发板供电，另一个 USB 端口用于连接 STM32F107 的 USB 控制器。关于如何设置开发板的更多信息，参见第 2.2.1“μC/Eval-STM32F107 开发板”一节。

F1-1(3)Total Phase公司的BeagleTMUSB 480协议分析仪是一款低成本、非侵入式的高速USB 2.0总线监控仪，支持实时USB类级解码。开发板运行例程并不需要该工具，但为了更好的理解USB协议，推荐用户使用该工具，如果你正在开发一个USB设备，它绝对是一个必需品。关于如何连接协议分析仪的更多信息，参见2.2.2“USB协议分析仪BeagleTM USB 480”一节。

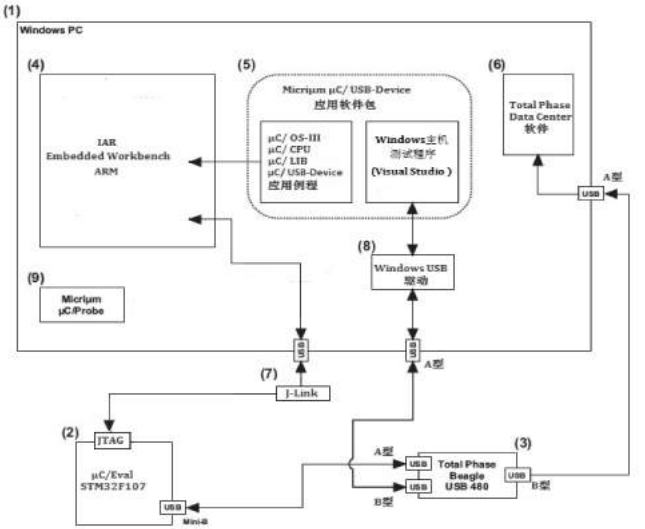


图1-1 USB设备应用例程连接

F1-1(4)为了构建和调试例程代码，用户需从IAR网站下载EWARM集成开发环境。还需要准备JTAG调试工具，例如J-Link。关于安装EWARM集成开发环境的更多信息参见2.1.2“IAR EWARM”一节。

F1-1(5)本书的配套软件包可以从Micriμm网站下载。软件包中不仅提供了应用实例，还包含μC/OS-III源代码和预编译链接库形式的μC/USB-Device代码。此外，一些例程需要定制的Windows应用程序，这些应用程序也包含在配套软件包中，以Visual Studio源代码形式提供。关于安装所需的Micriμm软件的更多信息参见.1.5“μC/USB-Device书的配套软件”一节。

F1-1(6)Total Phase的Data Center软件是一个免费的协议分析软件，Beagle™USB 480协议分析仪需配合该软件使用。关于安装Data Center软件的更多信息参见2.1.4“Total Phase Data Center软件”一节。

F1-1(7)μC/Eval-STM32F107开发板中国版没有内置J-Link调试器，用户需准备JTAG仿真器。

F1-1(8)为了运行例程，Windows PC机上需要安装一些驱动。一些例程如实

现 HID 和 MSC 类的应用，使用了 Windows 本地驱动程序。其他的例程如实现 CDC，PHDC 和供应商定义类的应用，需要安装本书配套软件包中提供的 Windows 驱动程序。

F1-1(9)用户可以从 Micriμm 网站下载 μC/Probe 的试用版，通过 μC/Probe，在例程运行时，不仅可以监控应用程序变量，还可以监控 μC/USB-Device 协议栈和 μC/OS-Ⅲ的内部变量。关于安装 μC/Probe 的更多信息参见 2.1.6“μC/Probe”一节。

本书和 μC/Eval-STM32F107 开发板为创建自己的 USB 设备感兴趣的工程师提供了一个很好的平台。

本书提供的应用实例仅作为参考，以协助工程师使用 Micriμm 的产品。

Micriμmg 公司不允许任何人在没有版权许可的情况下，基于本书中的信息设计或制造任何 USB 设备。

关于这些例程对任何特定用途的适应性，Micriμm 不作任何保证、声明或担保，Micriμm 也不承担由于应用或使用任何实例设计所产生的任何法律责任，并特别声明，任何和所有法律责任包括间接的或偶发的损害。

1.1 本书的章节内容

图 1-2 所示为本书的布局和流程。这对于理解各章节和附录之间的关系非常有帮助。左边第一列是 μC/USB-Device 的结构和应用实例。右边两列是各种附录，进一步解释了第一列章节中的信息。

F1-2(1)第 1 章：介绍，本章。

F1-2(2)第 2 章：准备工作。本章叙述了从准备测试环境到运行 μC/USB-Device 例子的过程。叙述了从何处下载和如何安装所需要的软件，以及如何连接所需的硬件。

F1-2(3)第 3 章：IAR EWARM。本章简要介绍了 IAR EWARM 集成开发环境及如何编译和调试嵌入式应用。

F1-2(4)第 4 章：例程任务模型。本章叙述了书中介绍的所有基于 μC/OS-III 的例程的任务模型。例程中大多数任务是相同的。

F1-2(5)第 5 章：应用实例＃1：通信设备类。本章叙述了如何使 μC/USB-Device 协议栈启动和运行起来。它展示了如何使用 CDC 类和 ACM 子类，通过一个虚拟 COM 端口，从 USB 设备发送和接收串行数据。还介绍了作为开发 USB 应用不可缺少的工具，Total Phase 的 USB 协议分析仪。

F1-2(6)第 6 章：应用实例＃2：人机接口设备。本章叙述了一个使用输入报告符发送信息给主机计算机的模拟鼠标。它主要控制鼠标指针在 Win-

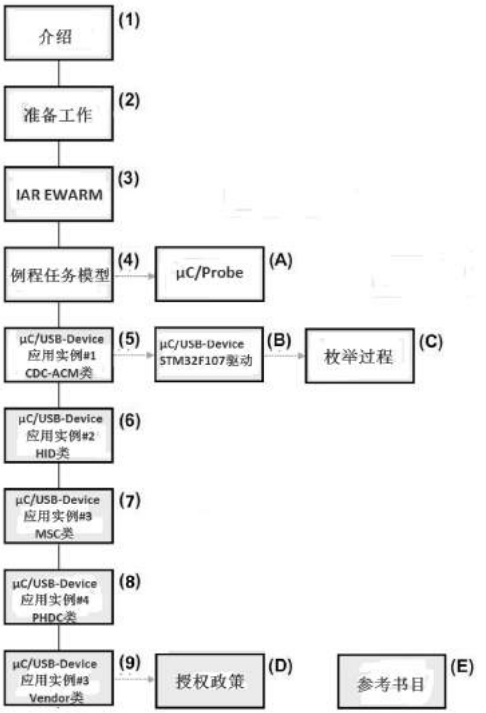


图 1-2 本书内容布局安排

dows 主机中来回移动。通过分析 Beagle USB 480 协议分析仪捕捉到的一些重要的 USB 总线通信事务，帮助用户更好的理解《嵌入式协议栈 μC/USB-Device》书中描述的 USB 原理。

F1-2(7)第 7 章：应用实例＃3：移动存储设备。本章叙述了如何使用 μC/USB-Device 协议栈和 MSC 类创建一个固态 USB 存储设备。叙述了如何运行该实例，并通过分析代码和捕获的 USB 通信流，解释了例程是如何工作的。

F1-2(8)第 8 章：应用实例＃4：个人健康设备类。本章提供了 Micriμm PHDC 类实现的一个示例。用户可以使用该示例作为基础，创建基于 USB 连接的医疗设备；由于例程还提供了与 PHDC 设备接口的 Visual Studio Windows 工程，用户可以参考该例程，创建其相应的 Windows 主机应用。

F1-2(9)第 9 章：应用实例＃5：供应商指定类。如果你的 USB 应用不符合任何前面描述的 USB 类定义，你可以使用供应商指定类来定义自己的应用。本章描述了一个定义 USB 供应商指定类的实例。

F1-2(10)附录 A：μC/Probe 介绍。本附录提供了 Micriμmμ C/Probe 产品的简

单介绍，它让用户可以很方便的修改和显示目标系统变量的实时变化。

F1－2(11)附录 B：STM32F107μC/USB-Device 驱动。本附录提供了 μC/USB-Device STM32F107 USB 控制器驱动的简单描述。

F1－2(12) 附录 C：枚举过程。本附录根据 USB 通信捕获，提供了枚举过程的简单描述。

F1－2(13) 附录 D：μC/OS-III 和 μC/USB-Device 授权政策。

F1－2(14) 附录 E：参考书目。

1.2 致 谢

感谢 Micriμm 的 USB 团队，他们不仅提供了顶级的 USB Device 协议栈，还提供了书中所有原始的资料(针对 Renesas RX63N 平台)。

感谢 ST 公司 Mr. Dominique Jugnon 和 Mr. Olivier Brun，他们的优秀团队设计了 μC/Eval-STM32F107 评估板和撰写了用户指南。感谢 ST 中国公司支持开发和生产 μC/Eval-STM32F107 中国版。感谢 IAR 公司的支持，EWARM 是一个很优秀的开发工具，读者一定会很喜欢在 μC/Eval-STM32F107 评估板上使用 EWARM 运行 μC/USB-Device 例程。还要感谢 Total Phase 提供了 Beagle USB 480 协议分析仪的样品。该协议分析仪和他们的 Data Center 软件是一个优异的工具，我们认为它们是开发基于 USB 应用不可或缺的工具。

本书编写过程得到北京麦克泰软件技术有限公司(www. bmrtech. com)大力支持和帮助，特此致谢！

第2章

准备和设置

本章将学习如何设置 μC/USB-Device 应用项目的运行环境。本章后续将介绍如何设置相应的软件和硬件。请按照后续说明,以正确的顺序配置。

2.1 软　件

2.1.1 Windows PC 系统需求

Windows XP (SP2 或更高版本,32 位和 64 位系统)和 Windows 7 及以上版本的操作系统支持所有需要安装的软件。

用户需要一台基于 Intel 或 AMD 处理器的电脑,运行速度最低为 2.0GHz,1GB 的物理内存,至少有一个高速 USB 端口和两个全速 USB 端口。

2.1.2 IAR EWARM 集成开发环境

本书提供的例程基于 IAR Embedded Workbench for ARM v7.x 开发。你可以在 IAR 网站下载最新版本的 EWARM IDE 30 天评估板的安装程序。下载地址:

http://supp.iar.com/Download/SW/?item=EWARM-EVAL

2.1.3 Total Phase Data Center 软件

Total Phase 的 Data Center™软件是一个免费的总线监控软件,通过 Beagle™硬件分析仪系列,实时捕获并显示 USB,I2C,SPI 和 CAN 总线数据。

Data Center 软件可以从下面的链接下载:

http://www.totalphase.com/products/data_center/

下载 Windows 版本对应的压缩文件,并解压到电脑中的一个文件夹中。下载包中没有安装程序,在桌面上创建指向应用程序文件 Data Center.exe 的快捷方式即可。

2.1.4　μC/USB-Device 书的配套软件包

本书配套的软件包包含：

- 针对 STM32F107 的 μC/OS-Ⅲ源代码；
- 针对 STM32F107 的 μC/CPU 源代码；
- μC/LIB 源代码；
- 针对 STM32F107 的 μC/USB-Device 预编译库；
- 应用工程源代码。

可以通过链接 http://micrium.com/download/micrium_uc-eval-stm32f107_ucos-iii-usbd-lib/下载例程代码。

压缩文件下载后，可以将其解压到电脑的任何位置，解压后的目录结构如图 2-1 所示。

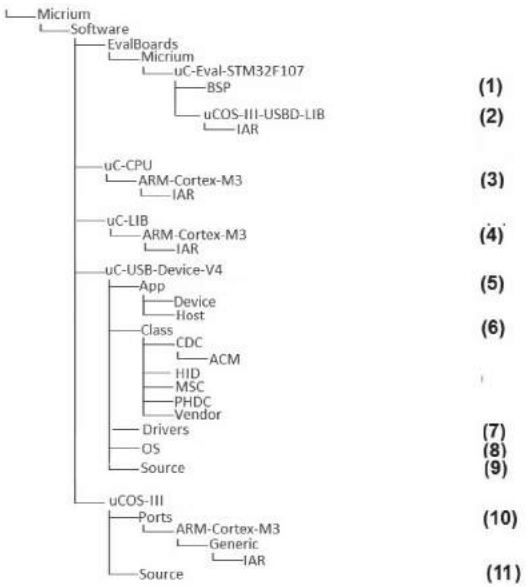


图 2-1　μC/USB-Device 书的配套软件

F2-1(1)BSP 目录包含 μC/Eval-STM32F107 评估板上外设的板级支持包，及 μC/OS-III 相关的函数。

F2-1(2)μCOS-Ⅲ-USBD-LIB 目录包含 μC/USB-Device 协议栈的预编译库、例程应用相关的代码及说明文档。

F2-1(3)μC/CPU 模块针对 STM32F107 Cortex-M3 处理器架构和 IAR 编译器定义的数据类型和临界段宏。μC/USB-Device 例程使用了 μC/CPU 模块中定义的数据类型。

F2-1(4)μC/LIB 模块针对 STM32F107 Cortex-M3 处理器架构和 IAR 编译器

定义的 C 标准库函数，易于用户实现产品认证。μC/USB-Device 例程引用了 μC/LIB 模块定义的函数和宏。

F2-1(5)μC/USB-Device 协议栈的应用级代码位于该目录：

- Device：包含运行在设备上的应用代码。
- Host：包含 PC 端的 windows 应用程序和 INF 文件。

【备注】：此处 Host INF 文件并没有在工程代码中，而是有单独的路径。下载链接如下：

http://micrium.com/download/micrium_usbd-host-app/

F2-1(6)所有 μC/USB-Device 正式用户可获取的 μC/USB-Device 类代码位于该目录中。关于如何购买 μC/USB-Device 类授权的详细信息，参考附录 D“μC/OS-III 和 μC/USB-Device 授权政策”。

F2-1(7)μC/Eval-STM32F107 评估板 USB 控制器的驱动代码位于该目录。关于驱动实现的详细的信息，参考附录 B“μC/USB-Device STM32F107 驱动”。

F2-1(8)μC/USB-Device 的 μC/OS-III 接口文件位于该目录。

F2-1(9)所有 μC/USB-Device 正式用户可获取的 μC/USB-Device 协议栈源代码位于该目录中。关于如何购买 μC/USB-Device 授权的详细信息，参考附录 D“μC/OS-III 和 μC/USB-Device 授权政策”。

F2-1(10)针对 STM32F107 Cortex-M3 的 μC/OS-III 移植位于该目录。它包含任务切换函数和系统节拍函数。

F2-1(11)μC/OS-Ⅲ源代码可以免费提供短期评估。更多信息请参考附录 D“μC/OS-III 和 μC/USB-Device 授权政策”。

【备注】μC/OS for maker 计划。

2.1.5　μC/Probe

μC/Probe 是一个 Windows 应用软件，它支持用户在运行时显示或改变几乎所有内存中或嵌入式目标板上的参数。关于 μC/Probe 的更多信息，参见附录 A“μC/Probe 介绍”。

第一章“介绍”中叙述的实例使用了 μC/Probe 获取运行时信息。μC/Probe 有两个版本：

- 全功能版，允许用户显示和修改任意数量的变量。
- 试用版，无时间限制，但仅允许用户显示和修改 5 个应用变量，但它允许用户监视任意 μC/OS-Ⅲ的变量，因为 μC/Probe 可以识别 μC/OS-Ⅲ。

两个版本都可以通过 Micriμm 网站获取，单击浏览器访问：

http://www.micrium.com/probe

配置 μC/Probe 与 J-Link 一起工作

μC/Eval-STM32F107 评估板中国版没有内置 J-Link 调试器，用户需单独购买。

J-Link 调试器不仅可以下载和调试代码，还可以与 μC/Probe 通信。可以同时运行 IAR 调试 EWARM 和 μC/Probe，如图 2－2 所示。

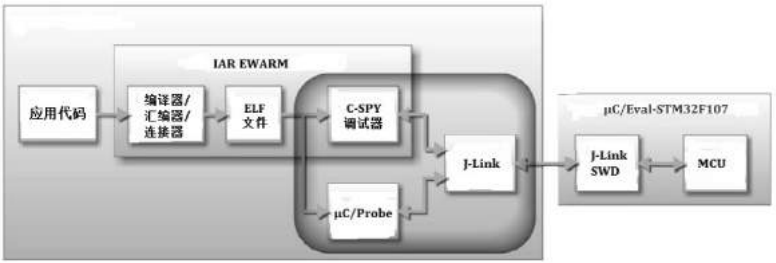


图 2－2　μC/Probe 与目标板的连接

安装 IAR EWARM 后，J-Link 驱动已经安装到电脑中。

打开 μC/Probe 应用程序，单击顶部工具栏的 Setting 按钮，将打开 μC/Probe 配置界面，如图 2－3 所示。

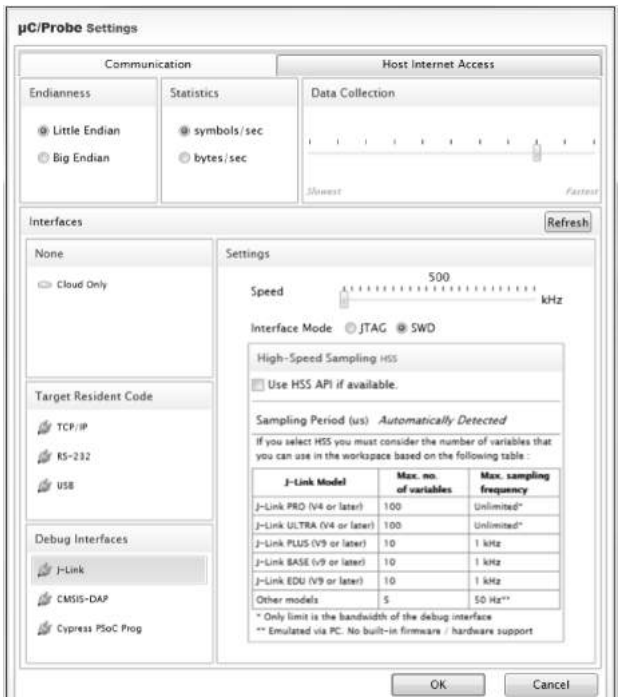


图 2－3　μC/Probe 配置界面

配置界面还允许指定速度，以获取更好的性能。

2.2　硬　件

在连接硬件之前，请确保 2.1“软件”一节叙述的所有需要的软件已安装。

2.2.1　μC/Eval-STM32F107 评估板

μC/Eval-STM32F107 评估板基于 STM32F107VCT 芯片的 100 引脚 TQFP 封装设计，图 2-4 将帮助您在评估板上找到对应的功能模块。

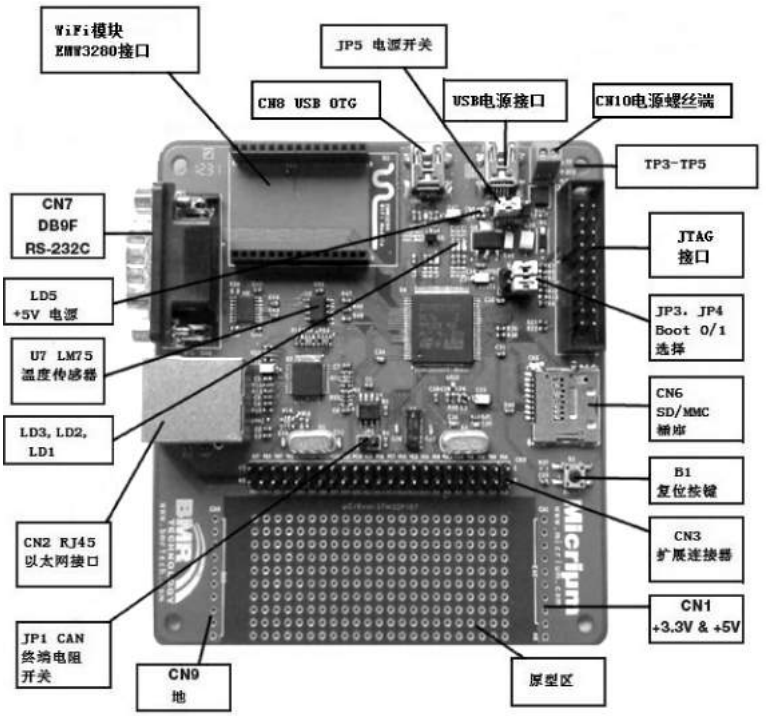


图 2-4　μC/Eval-STM32F107 评估板布局

关于评估板各个功能模块的详细信息，可以参考《嵌入式实时操作系统 μC/OS-III 应用开发》一书或访问北京麦克泰软件技术有限公司网站，下载相应的文档。

2.2.2　USB 协议分析仪 Beagle™USB 480

Total Phase 的 Beagel™ USB 480 协议分析仪是非侵入式的总线监控设备，允许用户实时查看和分析总线上的串行数据。

关于价格和订购信息，请访问：

http://www.totalphase.com/products/beagle_usb480/

Beagle USB 480 检测仪的一侧是一个 USB B 型插孔，这是分析端，如图 2-5 所示。该端口连接到运行 Beagle Data Center 软件的电脑。在分析过程中，确保该端口处于连接状态，防止数据丢失。

图 2-5　Beagle USB 480 协议分析仪：分析端

另一侧是捕捉端，如图 2-6 所示。它包含一个 USB A 型插孔和一个 USB B 型插孔。用于连接主机电脑和评估板（μC/Eval-STM32F107）。主机电脑和分析电脑是同一台电脑。尽管在某些情况下，这不是最好的选择。

图 2-6　Beagle USB 480 协议分析仪：捕捉端

捕捉端作为 USB 通道，为了遵循 USB 2.0 的规范，主机电脑和目标设备之间使用的 USB 电缆不超过 16 英尺。

Beagle USB 480 协议分析仪的顶部有三个 LED 指示灯，如图 2-7 所示。绿色的 LED 作为分析端口连接指示灯。当 Beagle 分析仪正确连接到分析电脑，并通过 USB 供电后，绿色 LED 点亮。黄褐色的 LED 作为目标和主机连接指示灯。当目标主机电脑连接到分析仪后，黄褐色 LED 点亮。最后，红色 LED 是一个活动 LED。它的闪烁速率与监控的总线上传送的数量量成正比。如果总线上没有数据，但分析仪在工作，活动 LED 将保持其状态。

图 2-7　Beagle USB 480 协议分析仪：LED 指示灯

为了监控 μC/Eval-STM32F107 上的 USB 总线通信，使用图 2-8 说明设置连接，连接 Windows PC 主机到 Beagle USB 分析仪捕捉端的 USB B 型插孔，连接目标设备 μC/Eval-STM32F107 到 Beagle USB 分析仪捕捉端的 USB A 型插孔。

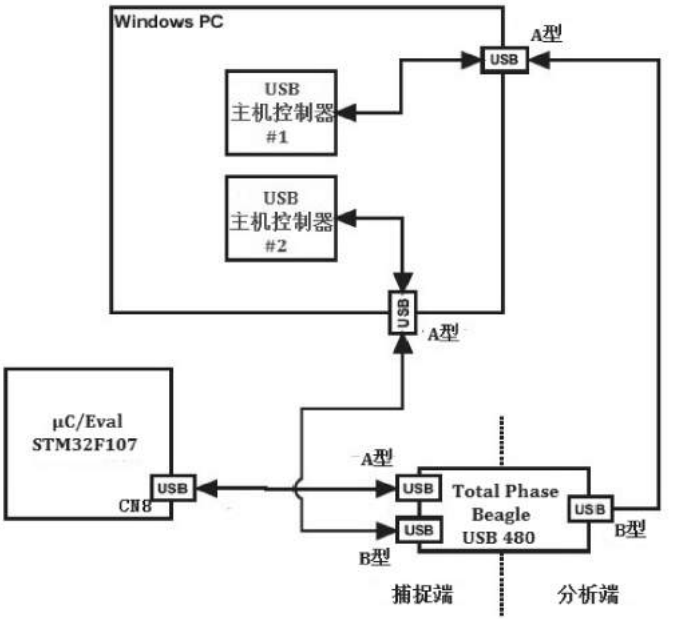


图 2-8　Beagle USB 480 协议分析仪：连接

桌面电脑和笔记本电脑通常有多个 USB 主机控制器。极力推荐分析端和捕捉端分别连接到不同的 USB 主机控制器。以避免 Beagel 分析仪最终捕获一些自己的通信数据。通常，桌面电脑背面的 USB 插孔连接到一个 USB 主机控制器，而前面的 USB 插孔连接到另一个 USB 控制器。同样，在笔记本电脑中，左边和右边的插孔分别连接到不同的主机控制器。如果你的电脑没有多个主机控制器，你可以使用一台单独的电脑运行 Data Center 软件，并连接到 Beagle USB 480 的分析端。

关于 Beagle USB 480 的更多信息，请查阅它们网站的用户手册。

第3章

IAR EWARM

IAR Embedded Workbench(IAR EW)是一套针对嵌入式应用的先进和容易使用的开发工具,它将 IAR C/C++ 编译™、汇编、链接、文本编辑、项目管理和 C-SPY 调试器®集成在一个集成开发环境中(IDE)。

IAR EW 内建了指定芯片(chip-specific)代码优化器,IAR EW 可生成针对 ARM 设备的高效和可靠的 FLASH/ROM 代码,除了这些强大的技术外,IAR System 还提供专业的、全球范围的技术支持。

本章提供了 IAR EWARM 的基本信息和快速入门指南,叙述了如何使用 EWARM 编译和调试代码。如果用户已经熟悉 EWARM IDE 环境,可以不必阅读本章。

3.1 IAR EWARM 基本信息

全功能的 IAR EWARM 具有以下特点:

- 支持的处理器架构:ARM7™、ARM9™、ARM9E™、ARM10™、ARM11、SecurCore、Cortex-M0、Cortex-M1、Cortex-M3/4/7、Cortex-R4/5/7、Cortex-A5/7/8/9/15、Intel® XScale™。
- 几乎支持市场上常见的所有芯片。
- 生成更加快速、高效和紧凑的工业级代码。
- 完全符合 ARM 的 EABI 标准。
- 支持识别插入的 RTOS-aware 调试,包括本书用到的 μC/OS-Ⅲ内核。
- 函数剖析(Function Profiler)。
- 中断图形窗口(Interrupt graph window)。
- 数据记录窗口。
- 支持 MISRA C:2004、2008、2012。
- 支持多种第三方硬件仿真器。。
- 超过 1400 个示范项目。
- 支持多核调试。

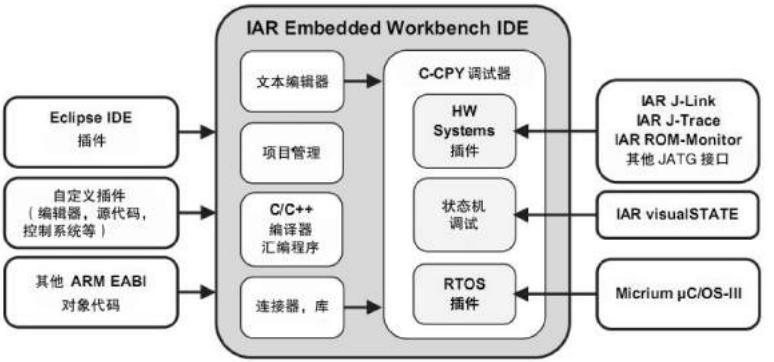


图 3-1　IAR Embedded Workbench

3.2　EWARM 快速入门

3.2.1　在 EWARM 中创建一个新的项目

EWARM 是按项目进行管理的，它提供了应用程序和库程序的项目模板。项目下面可以分级或分类管理源文件。允许为每个项目定义一个或多个编译链接(build)配置。在生成新项目之前，必须建立一个新的工作区(Workspace)。一个工作区中允许存放一个或多个项目。选择主菜单 File > New > Workspace 生成新工作区。在工作区中，通过主菜单 Project > Create New Project，弹出生成新项目窗口，如图 3-2 所示。

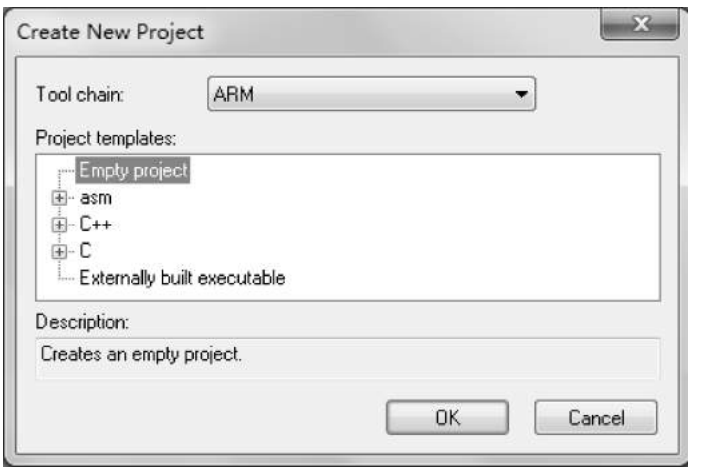


图 3-2　创建新项目

3.2.2　在 EWARM 中打开已有的项目

选择主菜单 File ＞Open＞ Workspace 打开已经创建好的项目。

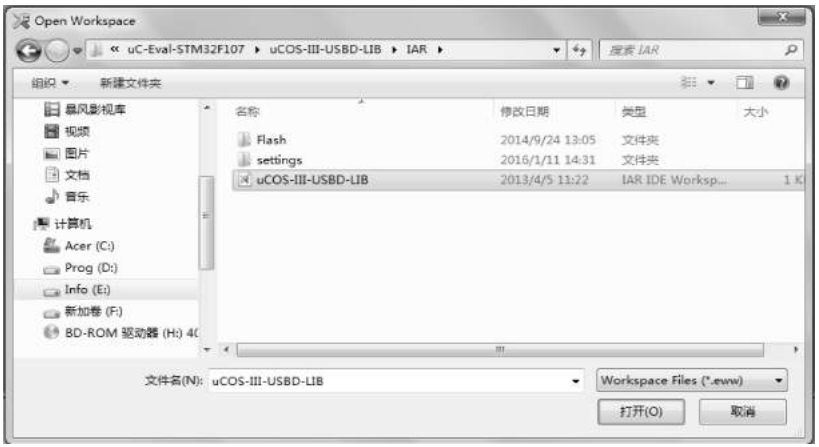


图 3－3　打开已有项目

3.2.3　在 EWARM 中配置项目

选择主菜单 Project->Options. . 或单击工程右键，选择 Options. . 配置工程的编译调试选项。

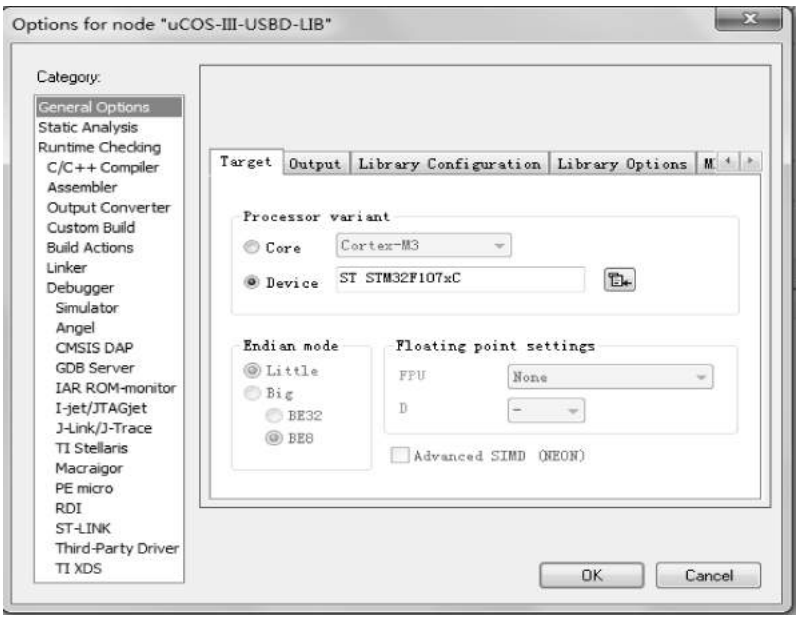


图 3－4　工程 Options 配置

3.2.4　在 EWARM 中构建项目

选择主菜单 Project->Make 或单击工具栏中的快捷方式构建可执行文件。

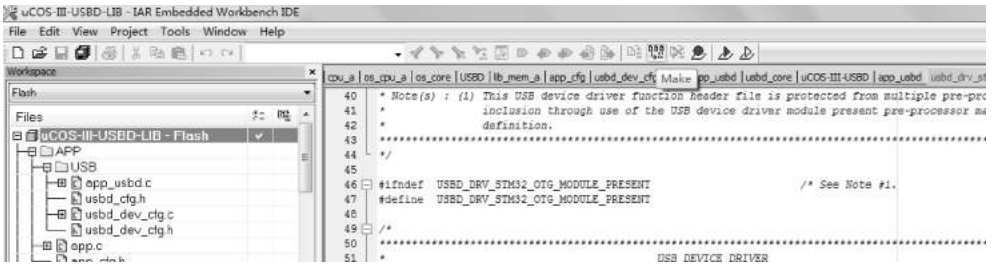


图 3-5　构建应用程序

3.2.5　在 EWARM 中启动调试会话

单击工具栏的 debug and download 按键，下载代码并启动调试。

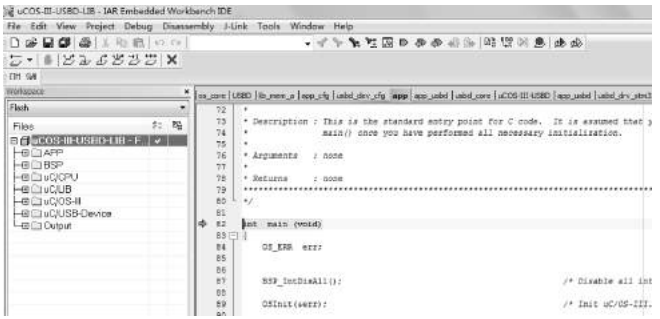


图 3-6　调试界面

第 4 章

例程任务模型

本书描述的所有例程位于 IAR EWARM 的同一个工作区和项目中，用户可以通过配置一系列的预处理宏使能某一个类和实例。

应用程序中实现的基于 μC/OS-III 的任务模型，如图 4－1 所示。

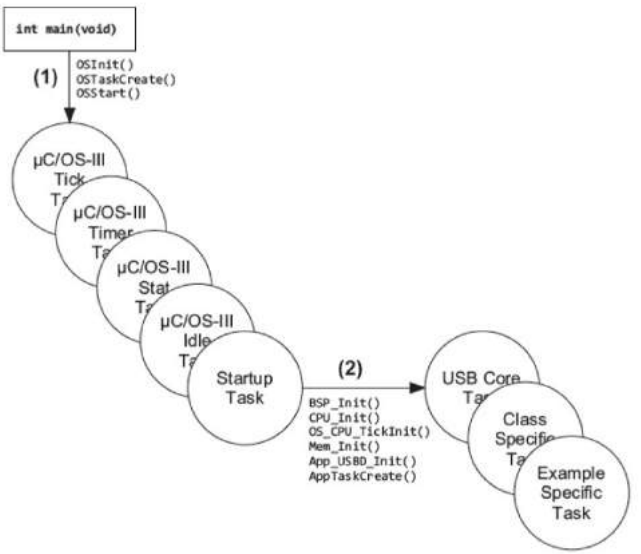


图 4－1　例程任务模型

F4－1(1)main()入口位于在 app.c 文件中。main()首先调用 OSInit()来初始化 μC/OS-III，OSInit()依次创建下列内部任务：

- 时钟节拍任务：μC/OS-III 使用 OS_TickTask()来跟踪正在延时的任务，或在规定时间内等待某个内核对象的任务。OS_TickTask()是一个周期执行的任务，它等待时钟节拍 ISR 发送的信号。
- 定时器任务：μC/OS-III 使用 OS_TimerTask()提供递减计数的定时服务。当计数器减到零时，执行一个动作。
- 统计任务：μC/OS-III 使用 OS_StatTask()来计算运行时的统计数据，如 CPU 使用率。

- 空闲任务：没有任务就绪运行时，μC/OS-III 运行空闲任务 OS_IdleTask()。

之后，main()函数创建 Startup 任务，并通过调用 OSStart()启动多任务处理。

F4-1(2) Startup 任务通过 app.c 中的 AppTaskStart()实现，该任务用来初始化各种硬件和软件模块，包括 μC/USB-Device 协议栈和类实例。通过调用 App_USBD_Init()函数，依次创建下列任务：

- USB 内核任务：μC/USB-Device 协议栈使用 USBD_OS_CoreTask()任务处理所有协议栈的内核事件和《嵌入式协议栈-μC/USB-Device》书中“任务模型”一节描述的相关操作。
- 类相关任务：取决于使能的类，在测试时，你可能需要一个或两个针对 USB 类的任务。请参考《嵌入式协议栈-μC/USB-Device》书中类相关部分的章节，了解关于类任务模型的更多信息。
- 例程相关任务：取决于使能的例程，在测试时，你可能需要一个或两个例程相关的任务。

然后，Startup 任务创建其它应用任务，闪烁 LED，结束启动过程。

4.1　在 μC/Probe 中看到的任务模型

μC/Probe 对 μC/OS-III 可见。图 4-2 展示了 PHDC 实例的内核调试插件截图。

CPU Usage 7.80 %

Task(s)							Performance				Task Stack			
Item	Name	Priority	State	Pending On Object	Pending On	Ticks Remaining	CPU Usage	CtxSwCtr	Interrupt Disable Time (Max)	Scheduler Lock Time (Max)	#Used	#Free	Size	Stack Usage
0	Switches Task	4	Delayed			17	7.54 %	6,040	0.0	0.0	90	166	256	35.16 %
1	Microphone Task	3	Delayed			0	6.05 %	60,813	0.0	0.0	110	146	256	42.97 %
2	LEDs Task	5	Delayed			170	0.00 %	646	0.0	0.0	51	205	256	19.92 %
3	USB Device PHDC tx comm	15	Delayed			100	0.01 %	597	0.0	0.0	52	204	256	20.31 %
4	USB Device PHDC rx comm	16	Delayed			100	0.01 %	598	0.0	0.0	72	184	256	28.13 %
5	USB PHDC Scheduler	8	Pending	Multiple Objects		0	0.00 %	0	0.0	0.0	47	209	256	18.36 %
6	USB Core Task	6	Pending	Task Message Queue	Task Q	0	0.00 %	0	0.0	0.0	56	456	512	10.94 %
7	Startup Task	2	Delayed			51	0.01 %	648	0.0	0.0	86	170	256	33.59 %
8	uC/OS-III Timer Task	6	Pending	Task Semaphore	Task Sem	0	0.42 %	15,438	8.0	8.0	49	463	512	9.57 %
9	uC/OS-III Start Task	14	Delayed			100	0.62 %	1,182	0.0	0.0	73	439	512	14.26 %
10	uC/OS-III Tick Task	1	Pending	Task Semaphore	Task Sem	0	1.61 %	65,078	0.0	0.0	20	364	384	5.21 %
11	uC/OS-III Idle Task	63	Ready			0	84.53 %	59,726	0.0	0.0	53	459	512	10.35 %

图 4-2　μC/Probe 内核插件截图

如果为后续的例程添加更多的功能，你会发现对于优化和调试，μC/Probe 非常有用。关于 μC/Probe 的更多信息，参见附录 A“μC/Probe 介绍”和“配置 μC/Probe 与 J-Link 一起工作”章节。

第5章

CDC ACM 例程:USB 串口转换器

CDC 类及其 ACM(抽象控制模型)子类在《嵌入式协议栈 μC/USB-Device》第一部分“通信设备类”部分介绍。本章结束时介绍了一个 CDC 应用实例。本章叙述了如何使用 IAR EWARM 和 Beagle USB 480 协议分析仪在 μC/Eval-STM32F107 上运行 CDC ACM 类演示实例。

USB CDC 类和其 ACM 子类允许主机电脑访问一个 USB 设备,实现数据传输。连接到主机电脑上的 USB 设备被识别为串行设备(虚拟 COM 端口),从而通过其读写功能实现数据传输。

本书叙述的实例,实现了一个 USB 串口转换器,可以用于类似图 5-1 所示的应用。

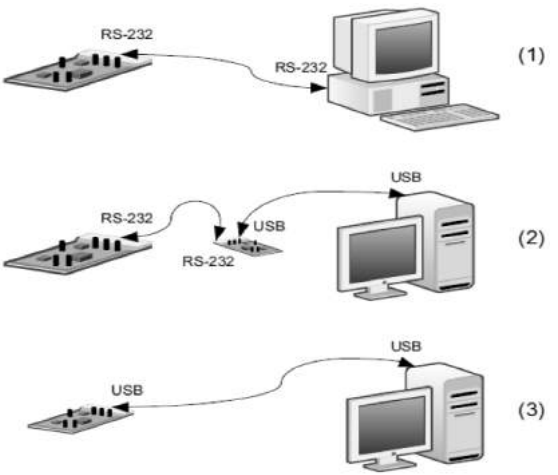


图 5-1　串口应用实例

F5-1(1)尽管听起来有些奇怪,在工业应用中的大多数嵌入式系统和设备设计了串行通信功能,一些实例包括 PLC,RTU,传感器,仪表和控制面板。

F5-1(2) CDC ACM 允许你创建一个 USB 串口转换器,放置在旧的串行设备和没有任何串行端口的新的主机电脑间。由于转换器将 USB 模拟为串行端口(虚拟 COM 口),因此,串行设备和主机软件可以同时保持不变。

F5-1(3) CDC ACM 允许将旧的串行设备设计更新为基于 USB 的新设备。同时,可以替换主机电脑而不需要更改旧的主机软件。

本章叙述的实例实现了一个 USB 串口转换器,可以打开一个串口终端程序,通过应答字符练习 μC/USB-Device 协议栈和其 CDC 及 ACM 子类,如图 5-2 所示。

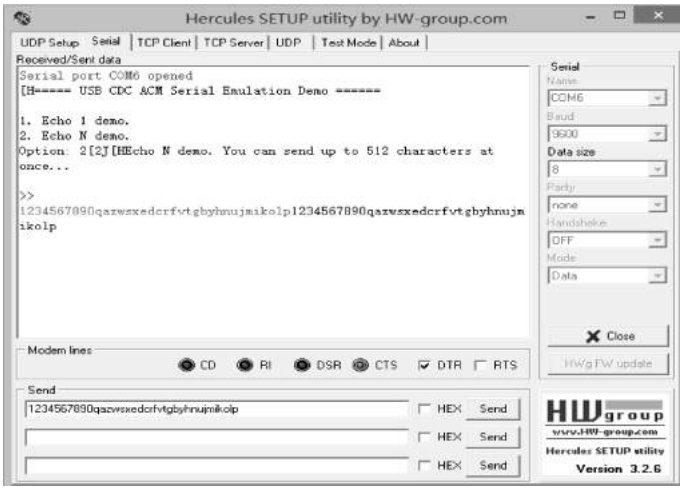


图 5-2　USB 串口转换器实例

5.1　在 IAR 中打开项目

打开 IAR EWARM 软件,导入本书配套软件包中的 C 语言工程,对所有的实例,仅需执行一次该操作,过程在 3.2.2 一节描述。

如果已经实现该过程,打开已选择路径下的工作区。

5.2　配置 USB 串口转换器实例

从项目浏览器视图中打开 app_cfg.h 头文件,如图 5-3 所示。

5.2.1　启用和禁用类

本书描述的所有实例,不仅位于同一个工作区,还在同一个项目中。为了启用 CDC ACM 类和 USB 串口转换器实例,app_cfg.h 中需要编辑的代码行见清单 5-1:

代码清单 5-1 配置 USB 串口转换器实例:app_cfg.h

L5-1(1)设置 APP_CFG_USB_CDC_EN 为 DEF_ENABLED,启用 CDC 类。

L5-1(2)设置 APP_CFG_USB_HID_EN 为 DEF_DISABLED,禁用 HID 类。

L5-1(3)设置 APP_CFG_USB_MSC_EN 为 DEF_DISABLED,禁用 MSC 类。

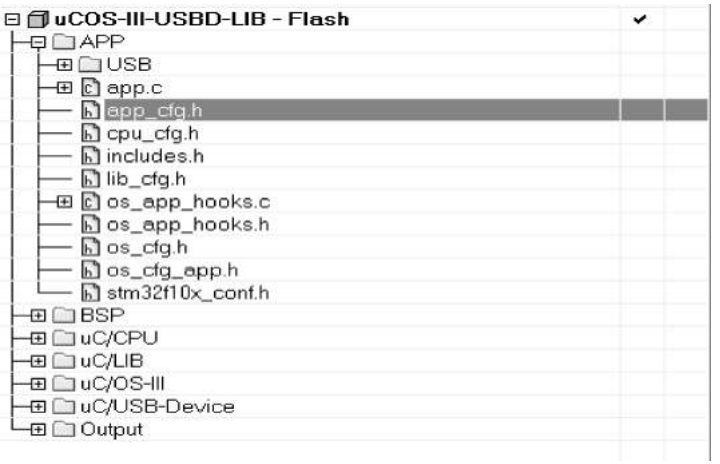


图 5-3　项目浏览器:配置文件

L5-1(4)为了运行 USB 串口转换器实例,设置 APP_CFG_USBD_VENDOR_EN 为 DEF_DISABLED,禁用自定义类。

L5-1(5)设置 APP_CFG_USB_PHDC_EN 为 DEF_DISABLED,禁用 PHDC 类。

L5-1(6)最后,设置 APP_CFG_USB_CDC_SERIAL_TEST_EN 为 DEF_ENABLED,启用 USB 串口转换器实例。

5.3　构建 USB 串口转换器实例项目

为了构建该项目,按下 F7 按键,参见 3.2.4“在 EWARM 中构建项目”部分的描述。

5.4　运行 USB 串口转换器实例

为了运行该实例项目,需要遵循下列步骤,包括准备 INF 文件,Window 使用它来加载本地驱动。

5.4.1　连接开发板

按照图 2-8“Beagle USB 480 协议分析仪:连接”所示,连接开发板,不要连接 USB 电缆到 μC/Eval-STM32F107 开发板上的 CN8。

5.4.2　启动调试会话

按 3.2.5“在 EWARM 中启动调试会话”部分所述,启动调试会话,按 F5 全速运

行程序。

5.4.3　安装 CDC 设备

此时,μC/USB-Device 设备协议栈已运行并等待 USB 事件,如 USB 设备的连接。用 USB 电缆连接 μC/Eval-STM32F107 开发板上 CN8 和 Beagle USB 480 的捕捉端,如图 2-8"Beagle USB 480 协议分析仪:连接"所示。如果没有 USB 协议分析仪,你仍然可以将电缆的另一端直接连接到主机电脑的可用的 USB 插口。

第一次运行该实例,主机电脑将安装 μC/Eval-STM32F107 作用一个虚拟串口,并在桌面电脑右下角的通知区域,使用典型的 Windows 消息显示,通知你设备的安装过程,如图 5-4 所示。

图 5-4　Windows 安装 CDC 设备

如果是第一次安装该设备,Windows 安装过程将会失败,因为这个特定的实例,尽管它使用了一个 Windows 本地驱动,它还需要一个 INF 文件来加载驱动。

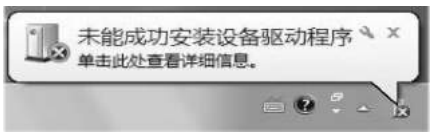


图 5-5　Windows 通知设备安装错误

INF 文件的主题是简要描述《嵌入式协议栈 μC/USB-Device》书中 3.2"关于 INF 文件"部分的内容,本书配套软件包中包含该实例需要的 INF 文件。仅需要确保 INF 文件和 CDC 设备的硬件 ID 匹配。为此,可以使用一个文本编辑器如记事本,打开下面的 INF 文件:

\$\Micrium\Software\μC-USB-Device-V4\App\Host\OS\Windows\CDC\INF\usbser.inf

定位指定供应商 ID=FFFFh 和产品 ID=1234h,如图 5-6 所示,并确认他们与

```
[MANUFACTURER]
%ProviderName% = DeviceList, NTx86, NTamd64

[DeviceList.NTx86]
%PROVIDER_CDC% = DriverInstall, USB\VID_ffff&PID_1234&MI_00
%PROVIDER_CDC% = DriverInstall, USB\VID_ffff&PID_1234

[DeviceList.NTamd64]
%PROVIDER_CDC% = DriverInstall, USB\VID_ffff&PID_1234&MI_00
%PROVIDER_CDC% = DriverInstall, USB\VID_ffff&PID_1234

; ================ Installation sections ==================

[DriverInstall]
include    = mdmcpq.inf
CopyFiles = FakeModemCopyFileSection
```

图 5-6　更新 INF 文件

usbd_dev_cfg.c 中配置的硬件 ID 匹配,如代码清单 5-2 所示。

```
/*
*********************************************************************************************************
*                                        USB DEVICE CONFIGURATION
*********************************************************************************************************
*/

USBD_DEV_CFG  USBD_DevCfg_STM32F_OTG = {
    0xFFFF,                                         /* Vendor  ID.
#if (APP_CFG_USBD_VENDOR_EN == DEF_ENABLED)
    0x1003,                                         /* Product ID (VENDOR).
#elif (APP_CFG_USBD_CDC_EN == DEF_ENABLED)
    0x1234,                                         /* Product ID (CDC).
#elif (APP_CFG_USBD_HID_EN == DEF_ENABLED)
    0x1233,                                         /* Product ID (HID).
#elif (APP_CFG_USBD_MSC_EN == DEF_ENABLED)
    0x1232,                                         /* Product ID (MSC).
#elif (APP_CFG_USBD_PHDC_EN == DEF_ENABLED)
    0x0063,                                         /* Product ID (PHDC).
#endif
    0x0100,                                         /* Device release number.
   "OEM MANUFACTURER",                              /* Manufacturer  string.
   "OEM PRODUCT",                                   /* Product       string.
   "123456789ABCDEF",                               /* Serial number string.
    USBD_LANG_ID_ENGLISH_US                         /* String language ID.
};
```

清单 5-2 USB CDC 设备配置

如果打开 Windows 设备管理器,会找到一个安装失败的 CDC 设备新表项,Windows 在其它设备组中列出了该设备,如图 5-7 所示。

为了使用合适的驱动安装该设备,你需要为 Windows 系统提供最新的 INF 文件。因此,在设备管理器列表中,右击设备名称,选择“更新驱动程序软件”,如图 5-8 所示。

Windows 将启动一系列的对话框,来指定驱动程序的选项,选择浏览计算机以查找驱动程序软件选项,如图 5-8 所示。

浏览保存更新的 INF 文件 usbser.inf 文件的路径,如图 5-9 所示。

Win7 系统会出现警告信息,而 Win8 系统不会出现警告信息,而是错误信息,不能继续安装下去,如下图 5-10 所示。

图 5－7　Windows 项目管理器

图 5－8　更新驱动程序软件

图 5-9　更新驱动程序软件:INF 文件路径

图 5-10　错误:第三方 INF 不包含数字签名信息

解决办法如下:

(1) 将鼠标移动到屏幕右下角,在屏幕右侧弹出的菜单中选择“设置”。

(2) 再单击最下面更改电脑设置选项。

（3）单击“更新和恢复”菜单。

（4）再单击“高级启动”下的“立即重启”。

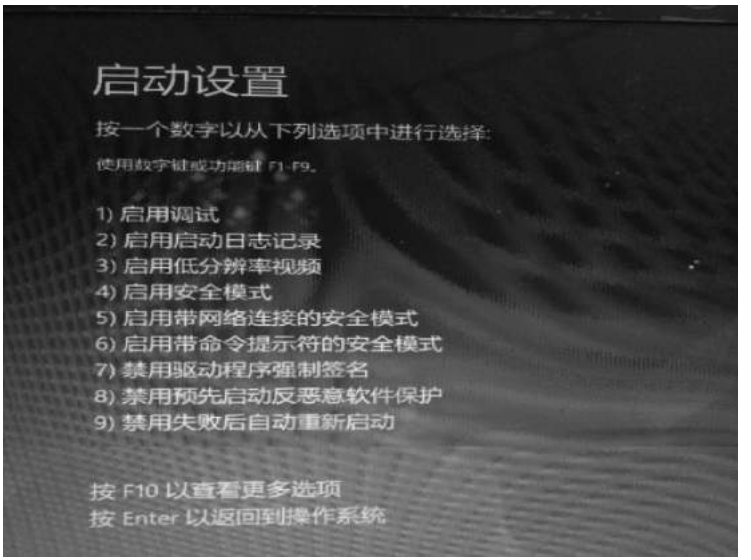
(5)选择“疑难解答”菜单。

(6) 选择“高级选项”菜单。

(7)选择“启动设置”菜单。

(8) 选择“重启”。

(9) 电脑重启后会出现下面菜单,选择禁用数字签名,电脑重启后电脑就禁用了数字签名,这时候重装驱动,在设备管理器中就不会出现黄色感叹号了!



由于该驱动没有数字签名,Windows 可能会给出一个警告信息。忽略该警告是安全的,选择始终安装此驱动程序软件,如图 5-11 所示。

图 5-11　更新驱动程序软件:警告信息

当安装新的 CDC ACM 设备时,Windows 需要几秒钟,并将 INF 文件复制到 $\Windows\inf 目录下。

图 5-12　更新驱动程序软件

安装结束后,Windows 将显示消息框,如图 5-13 所示。

图 5-13　更新驱动程序软件

在设备管理器端口(COM&LPT)组中,你会看到一个新的表项。类似图 5-14 所示。注意串行 COM 端口号。

图 5-14　设备管理器显示新安装的设备

5.4.4　测试新的虚拟串行端口

CDC ACM 设备已经成功安装,作为一个虚拟 COM 端口,可以通过打开串行终端并回送一些字符,验证 μC/USB-Device 协议栈和其 CDC ACM 类。

推荐使用可以控制数据终端就绪(DRT)的串口终端程序,如 HW 集团 s. r. o 的 Hercules Setup。Hercules Setup 组件是一个非常有用的串口终端(RS-485 或 RS-232 终端)、UDP/IP 终端和 TCP/IP 客户服务器终端。HW 组织创建它用于内部使用,但现在是一个免费软件,非常流行。Hercules Setup 可以在 http://www.hw-group.com/products/herclues/index_en.html 页面下载。

如果希望使用其它的串口终端,《嵌入式协议栈 μC/USB-Device》书中表 8-10 "串口终端和 CDC 串口演示程序"提供了可以使用的串口终端列表。

启动 Hercules Setup,打开虚拟 COM 口的一个串口连接,如图 5-15 所示。

图 5－15　Hercules Setup：打开连接

运行在 μC/Eval-STM32F107 上的应用实例将打印一个选项菜单，允许你选择两种测试方式中的一种：

- Echo 1 demo：回应单个字符。
- Echo N demo：回应最多 512 个字符。

在主机发送一个 DTR(数据终端就绪)信号之前，CDC 设备不会显示选项菜单。为了发送该信息，选中 DTR 复选框，如图 5－16 所示。

为了执行 echo 测试，选择测试 2，键入字符 2 并回车。

该测试中，一次最多可以发送 512 字符，如图 5－17 串口终端窗口所示。将字符键入到终端的三个输出缓冲区的一个(文本框)，按下相应的 Send 按键，发送字符。终端将显示回送的字符，如图 5－17 所示。

为了返回主菜单，选择另一个测试，可以发送 Ctrl＋C 的 ASCII 编码值 03H，也可以在一个输出缓冲区(文本框)中，键入 03，选项 hexadecimal 格式复选框，按下相应的 Send 按键，发送 Ctrl＋C，如图 5－18 所示。

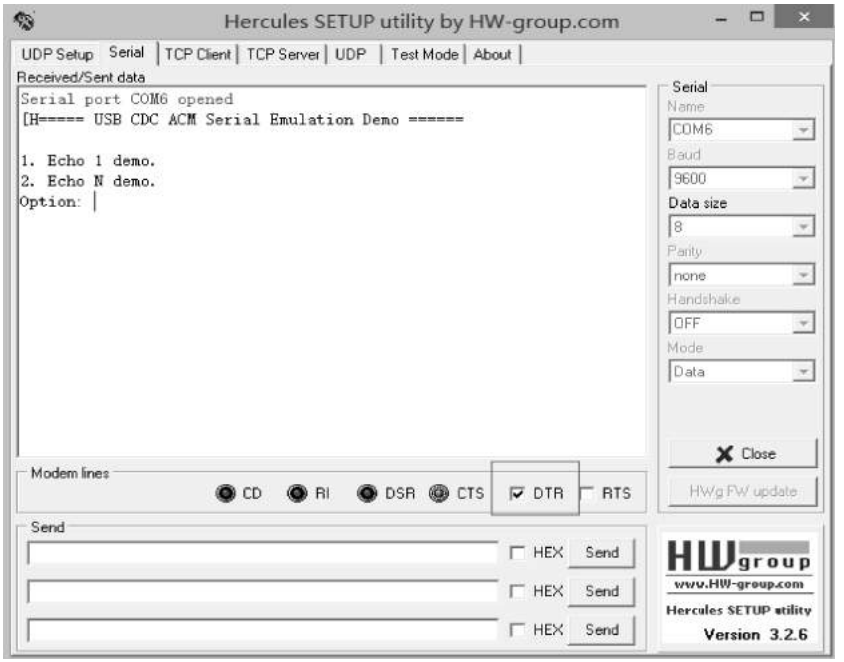


图 5-16　Hercules Setup:发送一个 DRT 信号

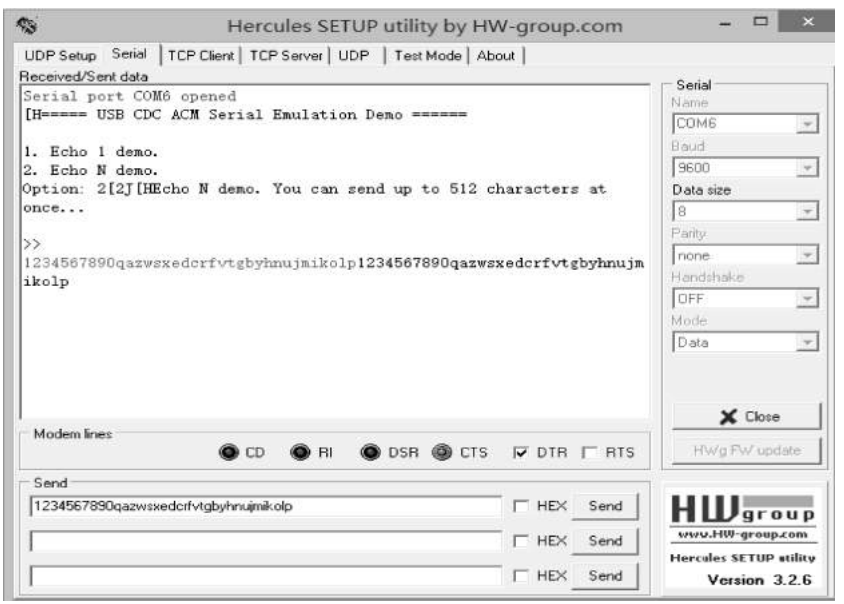


图 5-17　Hercules Setup:Echo N 测试

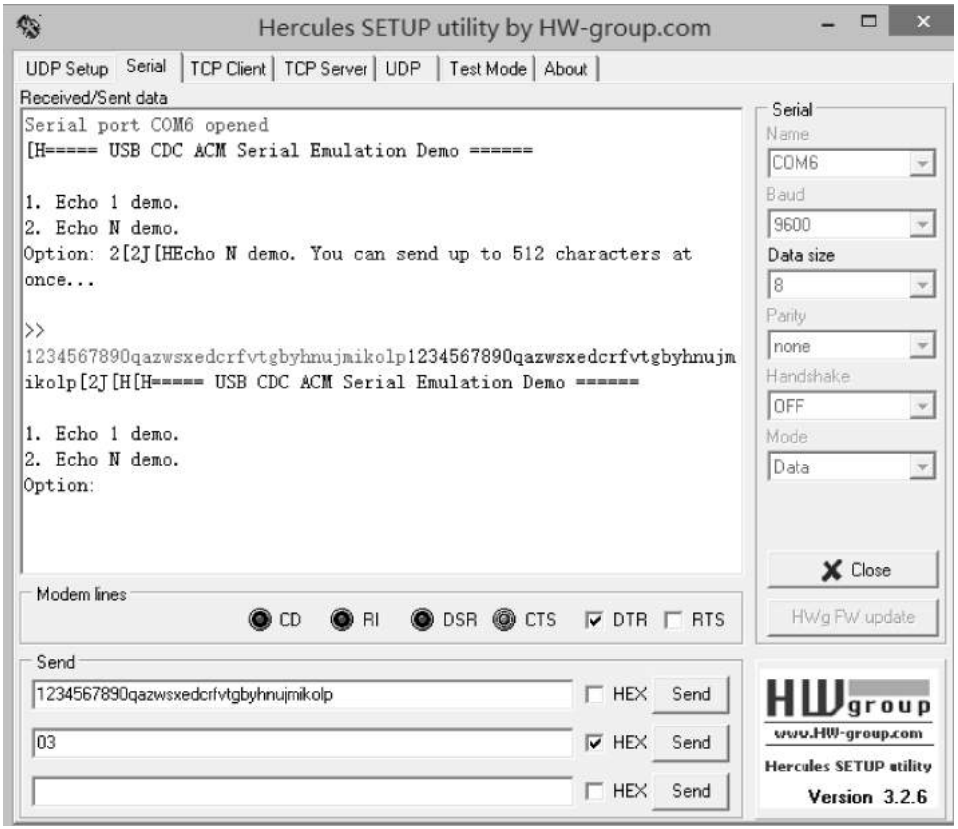


图 5-18　Hercules Setup:返回主菜单

5.5　代码如何工作

5.5.1　初始化

初始化 CDC 类,ACM 子类和 USB 转串口的实例函数在应用文件 app_usbd_cdc.c 中声明。下面的代码清单叙述了初始化过程:

代码清单 5-3 初始化 USB 转串口实例

L5-3(1) USBD_CDC_Init()函数在 usbd_cdc.c 文件中声明,它用于初始化类需要的所有内部结构和变量。关于该函数的更多信息,参考《嵌入式协议栈 μC/USB-Device》书中 C.1.1 "USBD_CDC_Init()"一节。

L5-3(2) USBD_ACM_SerialInit()函数在 usbd_acm_serial.c 中声明。该函数负责 ACM 串口模拟子类的初始化,通过为缓冲区分配内存,初始化其它内部结构和变量。该函数和所有 USBD_ACM 为前缀的函数在嵌

入式协议栈 μC/USB-Device》书附录 C“CDC ACM 子类函数”一节中描述。

```
*              DEF_FAIL, otherwise.
*
* Caller(s)   : App_USBD_Init().
*
* Note(s)     : none.
*********************************************************************************************************
*/

CPU_BOOLEAN  App_USBD_CDC_Init (CPU_INT08U  dev_nbr,
                                CPU_INT08U  cfg_hs,
                                CPU_INT08U  cfg_fs)
{
    USBD_ERR    err;
    USBD_ERR    err_hs;
    USBD_ERR    err_fs;
    CPU_INT08U  subclass_nbr;
#if (APP_CFG_USBD_CDC_SERIAL_TEST_EN == DEF_ENABLED)
    OS_ERR      os_err;
#endif

    APP_TRACE_DBG(("        Initializing CDC class ... \r\n"));

    USBD_CDC_Init(&err);                                (1)
    if (err != USBD_ERR_NONE) {
        APP_TRACE_DBG("        ... could not initialize CDC class w/err = %d\r\n\r\n", err));
        return (DEF_FAIL);
    }

    APP_TRACE_DBG(("            Initializing ACM serial subclass ... \r\n"));

    USBD_ACM_SerialInit(&err);                          (2)
    if (err != USBD_ERR_NONE) {
        APP_TRACE_DBG("            ... could not initialize ACM serial subclass w/err = %d\r\n\r\n", err));
        return (DEF_FAIL);
    }
```


初始化过程的下一步是添加 ACM 子类的新实例,并注册串行线路编码和控制线路回调函数,代码清单 5 - 4 所示。

```
    subclass_nbr = USBD_ACM_SerialAdd(100u, &err);              (1)
    if (err != USBD_ERR_NONE) {
        APP_TRACE_DBG("            ... could not create ACM serial subclass instance w/err = %d\r\n\r\n", err));
        return (DEF_FAIL);
    }

                                                                /* Register line coding and ctrl line change callb
    USBD_ACM_SerialLineCodingReg(       subclass_nbr,
                                        App_USBD_CDC_SerialLineCoding,
                                 (void *)0,
                                        &err);                  (2)

    USBD_ACM_SerialLineCtrlReg(       subclass_nbr,
                                      App_USBD_CDC_SerialLineCtrl,
                               (void *)0,
                                      &err);                    (3)
```

代码清单 5 - 4　ACM 子类实例

L5－4(1)USBD_ACM_SerialAdd()函数也是 usbd_acm_serial.c 的一部分。该函数负责添加 ACM 子类的新的实例,用于模拟串口。该函数的第一个参数是帧或微帧的轮询间隔,用于线状态通知。下一节将展示 Beagel USB 480 捕捉到的该通知的信息。

L5－4(2)函数 USBD_ACM_SerialLineCodingReg()用于注册一个回调函数,响应串口线路编码更改的通知(例如 9600-8-N-1)。

L5－4(3)函数 USBD_ACM_SerialLineCtrlReg()也用于注册一个回调函数,在本例中,用于响应串口控制线路状态变化的通知(例如 DTR 和 RTS 信号)。

两个回调函数都在应用级文件 app_usbd_cdc.c 中声明。

初始化过程的下一步对所有 USB 类是相同的。添加前面创建的实例到全速设备配置中,如代码清单 5－5 所示。

```
USBD_ACM_SerialCfgAdd(subclass_nbr, dev_nbr, cfg_fs, &err_fs);              (1)

if (err_fs != USBD_ERR_NONE) {

  APP_TRACE_DBG(("        ... could not add ACM serial subclass instance #%d to FS configuration
                          w/err = %d\r\n\r\n", subclass_nbr, err_fs));


}
```

代码清单 5－5　添加 ACM 子类实例到全速配置

L5－5(1)继续添加 ACM 子类到全速配置中。如果 ACM 子类实例被添加到 USB 设备中,

没有如内存分配,无效参数等问题,函数 USBD_ACM_SerialCfgAdd()返回 DEF_YES。

关于该函数参数和返回错误代码的更多信息,见《嵌入式协议栈 μC/USB-Device》书 C.2.3“USBD_ACM_SerialCfgAdd()”一节。

App_USBD_CDC_Init()函数最后要做的是创建实现 echo 终端的 μC/OS-III 任务。如果熟悉 μC/OS-III,就会知道函数 OSTaskCreate()用来创建一个受 μC/OS-III 管理的任务。注意 ACM 子类号作为第四个参数如何传递。OSTaskCreate()的第四个参数是一个指向可选数据区的指针,当任务第一次执行时,该数据区可以用来给任务传递参数。在下一节 5.2.2 中,将会了解到实现该任务的函数需要 subclass_nbr 变量来引用 ACM 子类的这个特点实例。

代码清单 5－6　创建实现 Echo 终端的 μC/OS-Ⅲ任务

```
OSTaskCreate(       &App_USBD_CDC_SerialTaskTCB,
                    "USB Device CDC ACM Test",
                     App_USBD_CDC_SerialTask,
             (void *)subclass_nbr,
                     APP_CFG_USBD_CDC_SERIAL_TASK_PRIO,
                    &App_USBD_CDC_SerialTaskStk[0],
                     APP_CFG_USBD_CDC_SERIAL_TASK_STK_SIZE / 10u,
                     APP_CFG_USBD_CDC_SERIAL_TASK_STK_SIZE,
                     0u,
                     0u,
             (void *)0,
                     OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR,
                    &os_err);
if (os_err != OS_ERR_NONE) {
    APP_TRACE_DBG(("        ... could not add CDC ACM serial test task w/err = %d\r\n\r\n", os_err));
    return (DEF_FAIL);
}
```

5.5.2　串口任务

串口任务在 app_usbd_cdc.c 中声明,App_USBD_CDC_SerialTask()函数是一个无限循环。该任务以一个 3 状态机的形式实现 echo 终端,如图 5-19 所示。

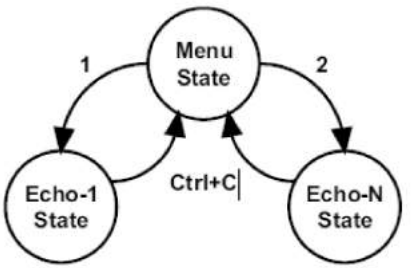


图 5-19　串口任务状态机

菜单状态

在定义为 APP_USBD_CDC_STATE_MENU 的菜单状态期间,任务通过函数 USBD_ACM_SerialTx()发送字符,显示可选菜单。如代码清单 5-7 所示:

代码清单 5-7　打印选项菜单

发送和接收字符的函数和其它以 USBD_ACM_Serial 为前缀的函数是 ACM 子类的一部分,用于模拟串口。其 API 请参考《嵌入式协议栈 μC/USB-Device》书附录 C“CDC ACM 子类函数”。

```
#define  APP_USBD_CDC_MSG                    "===== USB CDC ACM Serial Emulation Demo ======"\
                                             "\r\n"\
                                             "\r\n"\
                                             "1. Echo 1 demo.\r\n"\
                                             "2. Echo N demo.\r\n"\
                                             "Option: "
#define  APP_USBD_CDC_MSG_SIZE                       92u
```

```
(void)USBD_ACM_SerialTx( subclass_nbr,
                         App_USBD_CDC_Msg,
                         APP_USBD_CDC_MSG_SIZE,
                         APP_USBD_CDC_TX_TIMEOUT_mS,
                        &err);
```

打印菜单后,任务不会转换到其它状态,直到用户发送表示他想要执行的演示类型的字符。函数 USBD_ACM_SerialRx()用于读取字符。可以指定超时或通过传递值为 0 的超时值使其成为阻塞函数,如代码清单 5-8 所示。

```
(void)USBD_ACM_SerialRx( subclass_nbr,          /* Wait for character.
                        &ch,
                         1u,
                         APP_USBD_CDC_RX_TIMEOUT_mS,
                        &err);
```

代码清单 5-8　等待菜单选项

变量 ch 包含用户做出的选择,状态机将转到 Echo-1 或 Echo-N 状态。

ECHO-1 状态

在识别为 APP_USBD_CDC_STATE_ECHO_1 的 Echo-1 状态期间。任务将等待用户发送一个字符。通过调用带一个字符缓冲区和超时值为 0 的 USBD_ACM_SerialRx()函数实现,代码类似于代码清单 5-8 菜单状态。

该状态唯一的区别是,如果接收的字符不是 Ctrl+C(ASCII 码为 03h),将通过调用函数 USBD_ACM_SerialTx()回送该字符。否则,状态机将转换回菜单状态。

ECHO-N 状态

在识别为 APP_USBD_CDC_STATE_ECHO_N 状态期间,任务将等待用户发送多个字符(最大 512 个字符)。通过调用带 512 字节缓冲区和超时值为 0 的 USDB_ACM_SerialRx()函数实现。

与 Echo-1 状态类似,如果接收的字符不是 Ctrl+C(ASCII 码为 03h),将通过调用函数 USBD_ACM_SerialTx()回送字符串。不然,状态机将转换回菜单状态。

5.6　分析 USB 通信

为了捕捉 USB 通信,需要连接 Beagle USB 480,如设置一节图 2-8"USB 480 协议分析仪:连接"中阐述。

当监控 USB 通信时,最好在连接 μC/Eval-STM32F107 之前,连接 Beagle USB

480 的分析端,并启动捕捉。这意味着在开始捕捉前,断开 μC/Eval-STM32F107 的 CN8 与 Beagle USB 480 捕捉端的 USB 电缆连接。这使 Beagel USB 480 可以捕捉枚举阶段通信的描述符信息。

使用准备和设置 2.1.4“Total Phase Data Center 软件”一节创建的快捷方式或到软件包解压的目录下,运行 Data Center.exe,启动 Total Phase Data Center 软件。

一旦应用启动,它将检测 Beagle,并自动连接。

可以通过查看顶部的 LED,红色和绿色 LED 应该点亮,确认 Beagle USB 480 正确连接。

图 5-20　Data Center 软件图标

按下 Ctrl+R,或单击工具栏中的 RUN Capture 按键,启动捕捉。

将在表格中看到几个新行,显示类似于 Capture Started 的信息。

此时,你可以通过连接 USB 电缆 μC/Eval-STM32F107 的 CN8,实现 CDC 设备连接。

按照前面 5.4.4“安装 CDC 设备”一节叙述的步骤完成 CDC 设备的安装,接着遵循 5.4.5“测试新的虚拟串口”一节的说明,发送几个字符。

由于总线上可以“看到”通信,它将在 transcation 表格中实时显示。你不仅可以看到枚举过程中捕捉到的一些典型记录,如附录 C“枚举过程”中描述的获取设备描述符,设置地址,获取配置描述符,获取字符串描述符和设置配置;还可以在事务表格中看到一系列的新行,包含串行线路编码和控制线路状态的事务,如图 5-21 所示。

*Untitled - Total Phase Data Center v6.63.000

File Edit Analyzer View Help

Sp	Index	m:s.ms.us	Len	Err	Dev	Ep	Record	Summary
FS	22	0:03.986.603	54.8 ms				<Reset> / <Chirp J> / <Tiny J>	
FS	23	0:04.041.486					<Full-speed>	
FS	24	0:04.041.582	15.0 ms				[16 SOF]	[Frames: 849 - 864]
FS	25	0:04.057.399	0 B		00	00	Set Address	Address=05
FS	35	0:04.057.584	10.0 ms				[11 SOF]	[Frames: 865 - 875]
FS	36	0:04.057.630	18 B		05	00	Get Device Descriptor	Index=0 Length=18
FS	50	0:04.068.586	2.08 us				[1 SOF]	[Frame: 876]
FS	51	0:04.068.979	75 B		05	00	Get Configuration Descriptor	Index=0 Length=255
FS	70	0:04.069.586	2.08 us				[1 SOF]	[Frame: 877]
FS	71	0:04.069.501	34 B		05	00	Get String Descriptor	Index=3 Length=255
FS	85	0:04.069.917	4 B		05	00	Get String Descriptor	Index=0 Length=255
FS	99	0:04.070.146	24 B		05	00	Get String Descriptor	Index=2 Length=255
FS	113	0:04.070.382	0 B		05	00	Control Transfer (STALL)	Index=0 Length=10
FS	122	0:04.070.586	10.0 ms				[11 SOF]	[Frames: 878 - 888]
FS	123	0:04.081.183	18 B		05	00	Get Device Descriptor	Index=0 Length=18
FS	137	0:04.081.587	2.08 us				[1 SOF]	[Frame: 889]
FS	138	0:04.081.404	9 B		05	00	Get Configuration Descriptor	Index=0 Length=9
FS	152	0:04.081.680	75 B		05	00	Get Configuration Descriptor	Index=0 Length=75
FS	171	0:04.082.587	1.00 ms				[2 SOF]	[Frames: 890 - 891]
FS	172	0:04.083.766	0 B		05	00	Set Configuration	Configuration=1
FS	182	0:04.084.587	2.08 us				[1 SOF]	[Frame: 892]
FS	183	0:04.084.493	4 B		05	00	Get String Descriptor	Index=7 Length=4
FS	197	0:04.084.680	22 B		05	00	Get String Descriptor	Index=7 Length=22
FS	211	0:04.085.588	2.08 us				[1 SOF]	[Frame: 893]
FS	212	0:04.085.986	4 B		05	00	Get String Descriptor	Index=7 Length=4
FS	226	0:04.086.174	22 B		05	00	Get String Descriptor	Index=7 Length=22
FS	240	0:04.086.588	41.0 ms				[42 SOF]	[Frames: 894 - 935]
FS	241	0:04.128.020	7 B		05	00	Get Line Coding	9600bps 8N1
FS	255	0:04.128.227	0 B		05	00	Set Control Line State	

图 5-21　捕捉 USB 通信

为了避免并不需要分析的记录填充事务表格，可以按下 Ctrl＋R 或 Stop 按键停止捕捉。

5.6.1　获取配置描述符

图 5－22 GET Configuration Descriptor 获取的本章实例的通信捕获，描述了一个 2 接口的 CDC ACM 设备：

Navigator

Get Descriptor

General		Radix: auto
Timestamp	0:04.955.865.383	
Duration	447.666 us	
Length	75 Bytes	

Configuration Descriptor		Radix: auto
bLength	9	
bDescriptorType	CONFIGURATION (0x02)	
wTotalLength	75	
bNumInterfaces	2	
bConfigurationValue	1	
iConfiguration	Not Requested (4)	
bmAttributes.Reserved	0	
bmAttributes.RemoteWakeup	RemoteWakeup Not Supported (0b0)	
bmAttributes.SelfPowered	Self Powered (0b1)	
bMaxPower	100mA (0x32)	

Interface Association Descriptor		Radix: auto
bLength	8	
bDescriptorType	INTERFACE_ASSOCIATION (0x0b)	
bFirstInterface	0	
bInterfaceCount	2	
bFunctionClass	Communications and CDC Control (0x02)	
bFunctionSubClass	Abstract Control Model (0x02)	
bFunctionProtocol	AT Commands: V.250 etc (0x01)	
iFunction	CDC Device (7)	

Interface Descriptor		Radix: auto
bLength	9	
bDescriptorType	INTERFACE (0x04)	
bInterfaceNumber	0	
bAlternateSetting	0	
bNumEndpoints	1	
bInterfaceClass	Communications and CDC Control (0x02)	
bInterfaceSubClass	Abstract Control Model (0x02)	
bInterfaceProtocol	AT Commands: V.250 etc (0x01)	
iInterface	Not Requested (5)	

Class-Specific Descriptor Header Format		Radix: auto
bFunctionLength	5	
bDescriptorType	CS_INTERFACE (0x24)	
bDescriptorSubtype	Header Functional Descriptor (0x00)	
bcdCDC	1.20 (0x0120)	

Union Interface Functional Descriptor		Radix: auto
bFunctionLength	5	
bDescriptorType	CS_INTERFACE (0x24)	
bDescriptorSubtype	Union Functional Descriptor (0x06)	
bControlInterface	0	
bSubordinateInterface0	1	

图 5－22　捕获的 Get Configuration Descriptor

Abtract Control Management Functional Descriptor	Radix: auto
bFunctionLength	4
bDescriptorType	CS_INTERFACE (0x24)
bDescriptorSubtype	Abstract Control Management (0x02)
bmCapabilities.CommFeature	Supported (0b1)
bmCapabilities.LineStateCoding	Supported (0b1)
bmCapabilities.SendBreak	Supported (0b1)
bmCapabilities.NetworkConnection	Unsupported (0b0)

Call Management Functional Descriptor	Radix: auto
bFunctionLength	5
bDescriptorType	CS_INTERFACE (0x24)
bDescriptorSubtype	Call Management Functional Descriptor (0x01)
bmCapabilities.CallManagement	Handles call management (0b1)
bmCapabilities.DataClass	Call Management over Comm Class interface (0b0)
bDataInterface	0

Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	1 IN (0b10000001)
bmAttributes.TransferType	Interrupt (0b11)
wMaxPacketSize.PacketSize	64
wMaxPacketSize.Transactions	One transaction per microframe if HS (0b00)
bInterval	100

Interface Descriptor	Radix: auto
bLength	9
bDescriptorType	INTERFACE (0x04)
bInterfaceNumber	1
bAlternateSetting	0
bNumEndpoints	2
bInterfaceClass	CDC Data (0x0a)
bInterfaceSubClass	Unknown (0x00)
bInterfaceProtocol	No class specific protocol required (0x00)
iInterface	Not Requested (6)

Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	1 OUT (0b00000001)
bmAttributes.TransferType	Bulk (0b10)
wMaxPacketSize.PacketSize	64
bInterval	0

Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	2 IN (0b10000010)
bmAttributes.TransferType	Bulk (0b10)
wMaxPacketSize.PacketSize	64
bInterval	0

图 5－22　捕获的 Get Configuration Descriptor(续)

- 接口 0:通信类接口(CCI)
  - 端点 1:中断输入
- 接口 1:数据类接口(DCI)
  - 端点 1:块输出
  - 端点 2:块输入

这两个接口在《嵌入式协议栈 μC/USB-Device》书 8.1 节概述中描述。如果在 Data Center 软件中打开 Navigator 窗口,将会发现设备配置综述,包括 2 个接口和相应的端点,如图 5-23 所示。

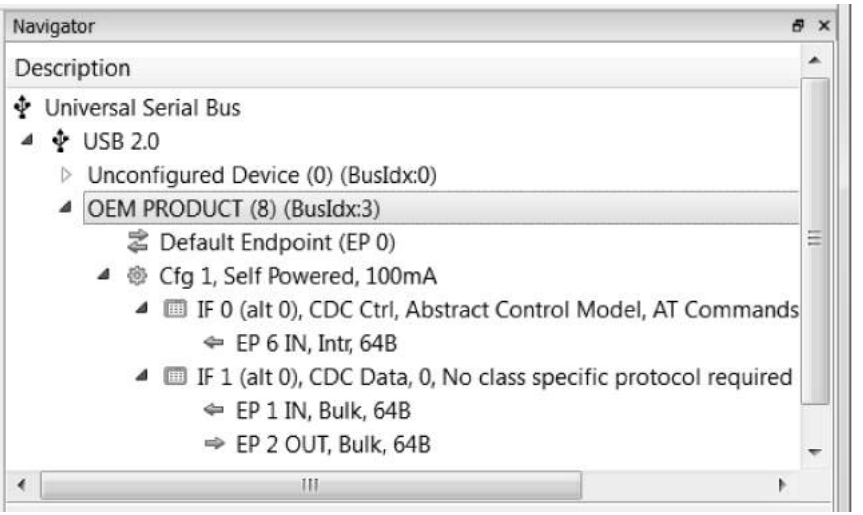


图 5-23　CDC 设备接口和端点

通过端点 1(中断输入)执行的任何事务用于设备管理和可选的呼叫管理。设备管理不仅包含设备配置和控制,还包含事件通知如串口线路状态。而呼叫管理包含建立和中止呼叫的事务。

如果希望专注于通过一个特定端点的事务,可以使用 Data Center 软件的过滤机制实现。

以下各节描述了在捕捉该特定的 CDC ACM 设备的 USB 通信中,一些最重要的事务。

5.6.2　获取和设置线路编码

如果滚动事务表格,将发现一系列标题为 GetLineCoding 和 SetLineCoding 的记录,类似于图 5-24 和图 5-25 所示。这些事务通过默认的控制端点 0 的管道传输,描述了当主机通过虚拟 COM 口,打开一个到 CDC 设备的串口连接的时刻。

获取线路编码

在图 5-24 所示的 GetLineCoding 请求期间,主机发送该请求获取当前 ACM 的配置(波特率,停止位,极性,数据位)。对如本例的模拟串口,串口终端在打开虚拟 COM 口时,自动发送该请求。

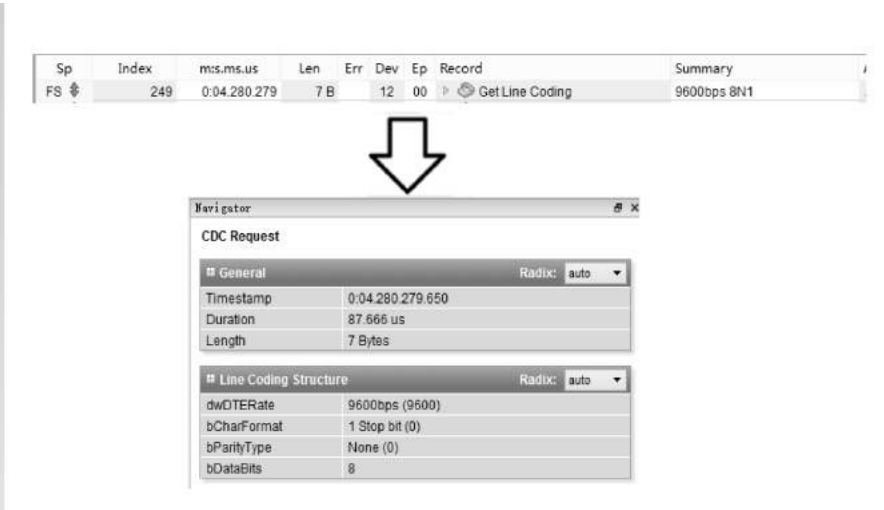


图 5-24　GetLineCoding 请求

设置线路状态

在图 5-25 所示的 SetLineCoding 请求期间,主机发送该请求来配置 ACM 设备的参数,如波特率,停止位数,极性,和数据位数。对模拟串口,在每次为一个打开的虚拟 COM 口配置串口环境时,串口终端自动发送该请求。

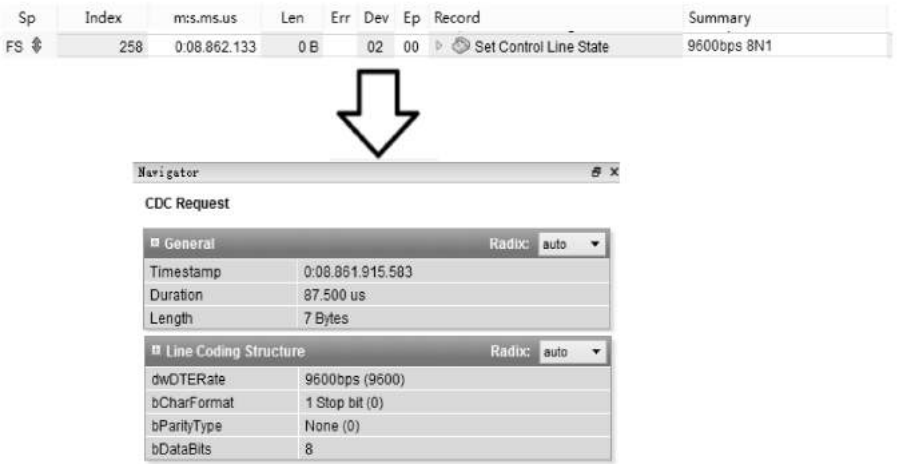


图 5-25　SetLineCoding 请求

5.6.3　设置控制线路状态

定位到标题为 SetControlLineState 的事务,类似于图 5-26 所示。主机发送该 ACM 请求来控制半双工调制解调器的载体,指示数据终端设备(DTE)是否就绪。

在这个特定的模拟串口实例中,DTE 来自 Hercules Setup 串口终端,如图 5-16 “Hercules Setup:发送一个 DTR 信号”描述,当单击 DTR 复选框时,该请求发送。

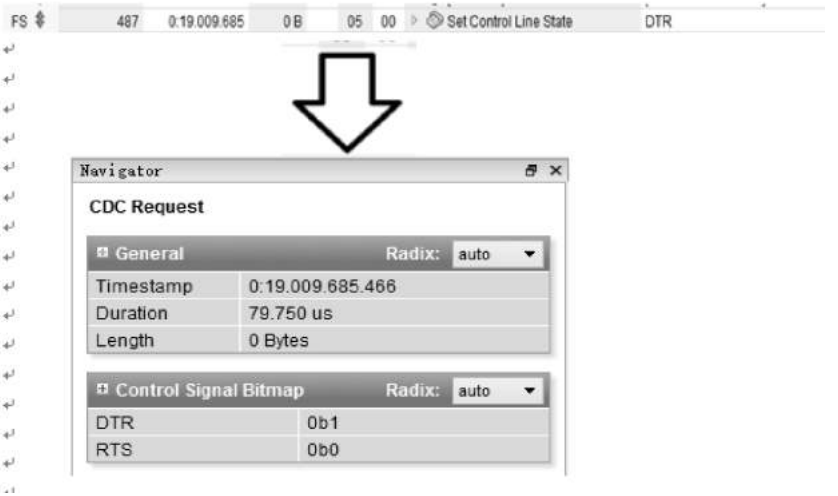


图 5-26　SetControlLineState 请求

5.6.4　CDC 输入/输出数据

如果通过回应字符实验来验证 μC/USB-Device 协议栈和其 CDC ACM 类,将会捕捉到一系列标题为 CDC OUT Data 和 CDC IN Data 的记录,类似于图 5-27 所示:

FS	17	0:09.024.832	36 B	07	01	OUT txn	31 32 33 34 35 36 37 38 39 30 71 61 7A 77 73 78 65 64 63...
FS	18	0:09.024.832	3 B	07	01	OUT packet	E1 87 D8
FS	19	0:09.024.835	39 B	07	01	DATA1 packet	4B 31 32 33 34 35 36 37 38 39 30 71 61 7A 77 73 78 65 64...
FS	20	0:09.024.862	1 B	07	01	ACK packet	D2
FS	21	0:08.000.041	36 B	07	02	IN txn [36891 POLL]	31 32 33 34 35 36 37 38 39 30 71 61 7A 77 73 78 65 64 63...
FS	22	0:08.000.041	1.02 s	07	02	[36891 IN-NAK]	
FS	23	0:09.024.941	3 B	07	02	IN packet	69 07 41
FS	24	0:09.024.945	39 B	07	02	DATA1 packet	4B 31 32 33 34 35 36 37 38 39 30 71 61 7A 77 73 78 65 64...
FS	25	0:09.024.972	1 B	07	02	ACK packet	D2

图 5-27　CDC 输入/输出数据事务

正如你所见,这些事务通过数据类接口(DCI)的端点 1 和 2(分别为 Bulk IN 和 Bulk OUT)管道传输。如《嵌入式协议栈 μC/USB-Device》书中表 8-1“CDC 端点使

用”描述。

图 5-27 所示的事务描述了当用户通过主机发送一个“X”字符(CDC OUT Data),运行在 μC/Eval-STM32F107 的 SerialTask 相应的回答时刻。SerialTask 在本章 5.5.2“串口任务”部分描述。

5.7　总　结

本章描述的实例,是创建自己的 CDC ACM 设备,或更新现有的基于 RS-232/485 的嵌入式系统而不需要创建不兼容的主机软件的一个很好的参考。

本章展示了如何运行实例,并通过描述代码清单和 USB 通信捕捉解释了代码是如何工作的。通过描述 CDC 设备捕捉到的最重要的 USB 通信,不断展示 Total Phase Data Center 软件的更多特性。

分析 USB 通信捕捉是一个非常好的实践练习,通过补充《嵌入式协议栈 μC/USB-Device》书描述的主题内容,最大化学习。

第 6 章

HID 例程:鼠标

HID 类实现了基于 USB 接口的计算机外设连接,如 USB 鼠标、键盘和游戏控制器等。在《嵌入式协议栈 μC/USB-Device》第 9 章"人机接口设备类"中介绍。本章介绍了几个应用范例。描述了如何使用 IAR EWARM 和 Beagle USB 480 协议分析仪在 uC/Eval-STM32F107 上运行 HID 类的演示实例。

电脑鼠标是一个相对位置的指示设备。换言之,鼠标返回的坐标值是它上次报告的移动量。

当主机需要更新屏幕时,电脑计算 X 轴和 Y 轴上的所有报告的鼠标移动量,相对于当前位置,移动指针。

该实例在一个无限循环中发送报告数据来模拟鼠标移动,X 轴和 Y 轴的报告数据分别为 50。任务延时 100ms 后,发送另一个报告数据,X 轴和 Y 轴的值分别为-50。结果,鼠标指针将在主机屏幕上来回移动,如图 6-1 所示。

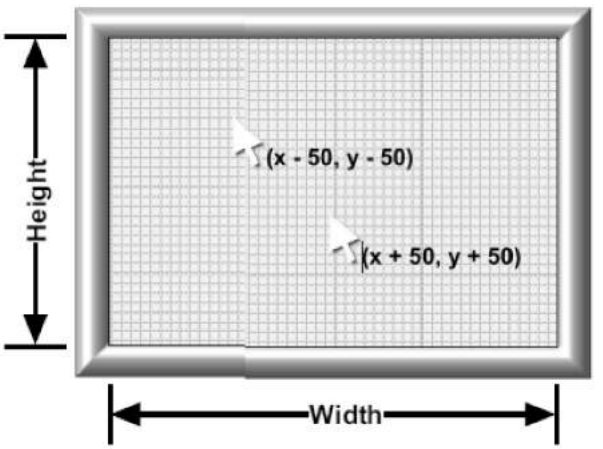


图 6-1　鼠标实例

6.1　在 IAR 里打开项目

打开 IAR EWARM 软件,导入本书配套软件包中的 C 语言工程,对所有的实例,仅需执行一次该操作,过程在 3.2.2 一节描述。

如果已经实现该过程,打开已选择路径下的工作区。

6.2　配置鼠标实例

打开项目浏览器中的 app_cfg.h 文件,如图 6-2 所示。

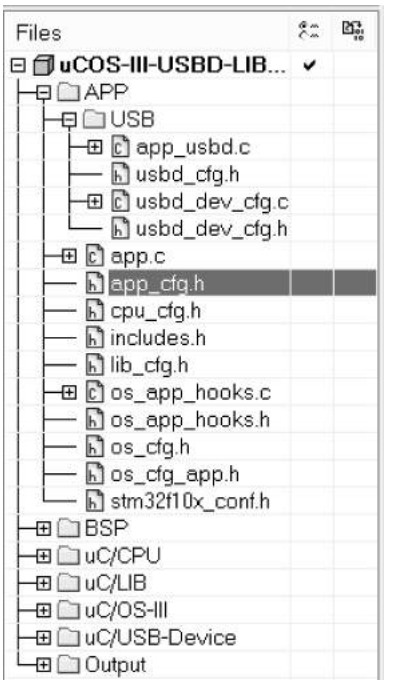


图 6-2　项目浏览器:配置文件

本书描述的所有实例,不仅位于同一个工作区,还在同一个项目中。为了启用 HID 类和鼠标实例,需要修改的 app_cfg.h 中的代码行,如代码清单 6-1 所示。

代码清单 6-1　配置鼠标实例:app_cfg.h

L6-1(1)设置 APP_CFG_USBD_CDC_EN 为 DEF_DISABLED,禁用 CDC 类。

L6-1(2)设置 APP_CFG_USBD_HID_EN 为 DEF_ENABLED,启用 HID 类

L6-1(3)设置 APP_CFG_USBD_MSC_EN 为 DEF_DISABLED,禁用 MSC 类。

L6-1(4)为了运行鼠标实例,设置 APP_CFG_USBD_VENDOR_EN 为 DEF_

DISABLED,禁用厂商自定义类。

L6-1(5)设置 APP_CFG_USBD_PHDC_EN 为 DEF_DISABLED,禁用 PHDC 类。

L6-1(6)最后,设置 APP_CFG_USBD_HID_TEST_MOUSE_EN 为 DEF_ENABLED,启用鼠标实例。

```
/*
*********************************************************************************************************
*                                   uC/USB-DEVICE APPLICATION CONFIGURATION
*********************************************************************************************************
*/

#define  APP_CFG_USBD_CDC_EN                            DEF_DISABLED      (1)
#define  APP_CFG_USBD_HID_EN                            DEF_ENABLED       (2)
#define  APP_CFG_USBD_MSC_EN                            DEF_DISABLED      (3)
#define  APP_CFG_USBD_VENDOR_EN                         DEF_DISABLED      (4)
#define  APP_CFG_USBD_PHDC_EN                           DEF_DISABLED      (5)

#define  APP_CFG_USBD_VENDOR_ECHO_SYNC_EN               DEF_DISABLED
#define  APP_CFG_USBD_VENDOR_ECHO_ASYNC_EN              DEF_DISABLED

#define  APP_CFG_USBD_HID_TEST_MOUSE_EN                 DEF_ENABLED       (6)

#define  APP_CFG_USBD_CDC_SERIAL_TEST_EN                DEF_DISABLED
```

6.3　构建鼠标实例项目

为了构建该项目,按下 F7 按键,如 3.2.4“在 EWARM 中构建项目”一节所述。

6.4　运行鼠标实例项目

由于本例实现了一个标准的 USB 鼠标,Windows 将使用本地驱动程序安装它,不需要在主机上安装任何驱动程序。

6.4.1　连接开发板

按照图 2-8“Beagle USB 480 协议分析仪:连接”所示,连接开发板。不要连接 uC/Eval-STM32F107 CN8 上的 USB 电缆。

6.4.2　启动调试会话

按照 3.2.5“在 EWARM 中启动调试会话”一节所述,启动调试会话,按下 F5 使程序全速运行。

6.4.3　安装 HID 设备

此刻,μC/USB-Device 协议栈已经运行并等待 USB 事件,如连接 USB 设备。通过 USB 电缆连接 μC/Eval-STM32F107 开发板的 CN8 到 Beagle USB 480 的捕捉端,如图 2-8“Beagel USB 480 协议分析仪:连接”所示。如果没有 USB 协议分析仪,

仍然可以将电缆的另一端直接插到主机可用的 USB 插座中。

第一次运行该实例时,主机将安装 μC/Eval-STM32F107 作为一个鼠标。Win8 系统将弹出信息框,通知你设备正在安装,如图 6-4 所示。

图 6-4　Windows 安装 HID 设备

如果 Windows 可以正确的安装设备,将通过图 6-5 所示的类似的消息通知你:

图 6-5　Windows 通知 HID 设备已经可以使用

此时,可以看到鼠标指针在屏幕上来回移动。

由于鼠标变得几乎不可用,可以断开 μC/Eval-STM32F107 CN8 上的 USB 电缆,以停止鼠标指针。

可以尝试从控制面板中打开 Windows 设备管理器,重新连接 USB 电缆,查看如何获取包含在设备列表中的新安装的 HID 设备,如图 6-6 所示。

图 6-6　设备管理器展示新安装的设备

可以通过单击图 3-5“调试界面”所示的 break 按键停止程序执行。

6.5　代码如何工作

初始化 HID 类和鼠标实例的函数如下。

代码清单 6-2　初始化鼠标实例

L6-2(1) USBD_HID_Init()函数在 usbd_hid.c 中声明,用于初始化类需要的所有内部结构和变量,包括 HID OS 层和 HID 报告模块。关于该函数的更多信息,参阅《嵌入式协议栈 μC/USB-Device》书 HID API 手册 D.1.1“USBD_HID_Init()”部分。

L6-2(2) USBD_HID_Add()函数也在 usbd_hid.c 中声明。该函数负责创建一个新的 HID 类实例。在此,可以指定 HID 类使用的协议。本例中,将使用 USB HID 规范定义的 Mouse protocol。该函数还设置了《嵌入式协议栈 μC/USB-Device》书中图 9-1“从主机 HID 解析视图中获取的报告描述符内容”和代码清单 9-2“鼠标报告描述符实例”描述的 Mouse Report　descriptor

同时,App_USBD_HID_Init()函数继续初始化过程,添加新的 HID 类实例到 USB 全速设备配置中,如代码清单 6-3 所示。

```
CPU_BOOLEAN  App_USBD_HID_Init (CPU_INT08U  dev_nbr,
                                CPU_INT08U  cfg_hs,
                                CPU_INT08U  cfg_fs)
{
    USBD_ERR    err;
    USBD_ERR    err_hs;
    USBD_ERR    err_fs;
    CPU_INT08U  class_nbr;
    OS_ERR      os_err;


    err_hs = USBD_ERR_NONE;
    err_fs = USBD_ERR_NONE;

#if (APP_CFG_USBD_HID_TEST_MOUSE_EN == DEF_ENABLED)
    App_USBD_HID_X = 101;
    App_USBD_HID_Y = 101;
#endif

    APP_TRACE_DBG(("        Initializing HID class ... \r\n"));

    USBD_HID_Init(&err);                                        /* Init HID cl
    if (err != USBD_ERR_NONE) {
        APP_TRACE_DBG(("        ... could not initialize HID class w/err = %d\
        return (DEF_FAIL);
    }
                                                                /* Create a HI
    class_nbr = USBD_HID_Add(             USBD_HID_SUBCLASS_BOOT,
#if (APP_CFG_USBD_HID_TEST_MOUSE_EN == DEF_ENABLED)
                                          USBD_HID_PROTOCOL_MOUSE,
#else
                                          USBD_HID_PROTOCOL_NONE,
#endif
                                          USBD_HID_COUNTRY_CODE_NOT_SUPPORTED,
                                          &App_USBD_HID_ReportDesc[0],
                                           sizeof(App_USBD_HID_ReportDesc),
                             (CPU_INT08U *)0,
                                           0u,
                                           2u,
                                           2u,
                                           DEF_YES,
                                          &App_USBD_HID_Callback,
                                          &err);
if (err != USBD_ERR_NONE) {
    APP_TRACE_DBG(("        ... could not instantiate a HID class w/err = %d\r\n\r\n", err));
    return (DEF_FAIL);
}
```

代码清单 6-3　添加 HID 类实例到全速配置

L6-3(1)执行添加新的 HID 实例到该全速配置的操作。

L6-3(2)如果 HID 类实例可以被添加到 USB 设备配置中,没有任何如内存分配,无效参数等的错误,USBD_HID_CfgAdd()函数将返回错误码 USBD_ERR_NONE。

关于该函数参数和返回错误代码的更多信息,请参考《嵌入式协议栈 μC/USB-

```
if (cfg_fs != USBD_CFG_NBR_NONE) {
                                                /* Add HID class to FS dflt cfg.                        */
    USBD_HID_CfgAdd(class_nbr, dev_nbr, cfg_fs, &err_fs);
    if (err_fs != USBD_ERR_NONE) {
        APP_TRACE_DBG(("       ... could not add HID class instance #%d to FS configuration w/err = %d\r\n\r\n", class_nbr, err_fs));
    }
}

                                /* If FS cfg fail, stop class init.                 */
  if (err_fs != USBD_ERR_NONE) {
   return (DEF_FAIL);
  }
```

Device》书 HID API 手册 D.1.3“USB_HID_CfgAdd()”部分。

App_USBD_HID_Init()函数做的最后一件事是创建一个 μC/OS-Ⅲ任务,来模拟鼠标。代码清单 6-4 展示了如何创建该任务。如果不熟悉 μC/OS-Ⅲ,推荐从 Micrium 网站下载 μC/OS-Ⅲ-STM32F107 的实时内核书的 PDF 版本,或者购买中

```
OSTaskCreate(       &App_USBD_HID_MouseTaskTCB,
                     "USB Device HID Mouse",
                     App_USBD_HID_MouseTask,
             (void *)class_nbr,
                     APP_CFG_USBD_HID_MOUSE_TASK_PRIO,
                    &App_USBD_HID_MouseTaskStk[0],
                     APP_CFG_USBD_HID_TASK_STK_SIZE / 10u,
                     APP_CFG_USBD_HID_TASK_STK_SIZE,
                     0u,
                     0u,
             (void *)0,
                     OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR,
                    &os_err);
if (os_err != OS_ERR_NONE) {
    APP_TRACE_DBG(("       ... could not add HID mouse task w/err = %d\r\n\r\n", os_err));
    return (DEF_FAIL);
}
```

文版《嵌入式实时操作系统 μC/OS-Ⅲ》。代码清单 6-4　创建鼠标任务

《嵌入式协议栈 μC/USB-Device》9.1“概述”一节描述了报告是一种数据结构,用于 USB 主机和 HID 设备之间交换数据。本例中,报告由 Microsoft 定义,并在 app_usbd_hid.c 中声明该结构为 App_USBD_HID_ReportDesc[]。该描述符的实例在《嵌入式协议栈 μC/USB-Device》表 9-2“鼠标报告描述符实例”中展示。

主机将通过中断 IN 端点周期性查询 μC/Eval-STM32F107。按照 μC/Eval-STM32F107 硬件手册,该端点是端点 1 到 6 范围中的一个。长度为 4 字节的报告数据,发送会主机,以响应查询令牌。4 字节数据包含每个 X 轴和 Y 轴的位移。

代码清单 6-5 展示了更新带位移的数据报告的函数。由于本实例的目的是移动鼠标指针。位移设置为 50,全局布尔型变量 App_USBD_HID_X 和 App_USBD_HID_Y 作为状态标志,改变前后移动:

```
static void  App_USBD_HID_MouseSetReport (CPU_INT08U  *p_buf)
{
    CPU_SR_ALLOC();


    p_buf[0] = 0;
    p_buf[1] = 0;

    CPU_CRITICAL_ENTER();
    p_buf[2] = (App_USBD_HID_X > 0) ? 50 : -50;
    p_buf[3] = (App_USBD_HID_Y > 0) ? 50 : -50;

    App_USBD_HID_X = !App_USBD_HID_X;
    App_USBD_HID_Y = !App_USBD_HID_Y;
    CPU_CRITICAL_EXIT();
```

代码清单 6-5　准备报告数据

鼠标任务每隔 100ms 调用一次 App_USBD_HID_MouseSetReport()函数,如代码清单 6-6 所示。

```
static  void  App_USBD_HID_MouseTask (void  *p_arg)
{
    CPU_INT08U   class_nbr;
    CPU_BOOLEAN  conn;
    USBD_ERR     err;
    OS_ERR       os_err;


    class_nbr = (CPU_INT08U)(CPU_ADDR)p_arg;

    while (DEF_TRUE) {                                                  (1)
        conn = USBD_HID_IsConn(class_nbr);                              (2)
        while (conn != DEF_YES) {                                       (3)
            OSTimeDlyHMSM(0, 0, 0, 250, OS_OPT_TIME_HMSM_STRICT, &os_err);

            conn = USBD_HID_IsConn(class_nbr);
        }

        App_USBD_HID_MouseSetReport(&App_USBD_HID_ReportBuf[0]);        (4)

        if ((App_USBD_HID_ReportBuf[2] != 0u) ||
            (App_USBD_HID_ReportBuf[3] != 0u)) {

            (void)USBD_HID_Wr( class_nbr,
                              &App_USBD_HID_ReportBuf[0],
                               APP_USBD_HID_REPORT_LEN,
                               0u,
                              &err);                                    (5)

            OSTimeDlyHMSM(0, 0, 0, 100, OS_OPT_TIME_HMSM_STRICT, &os_err);
        } else {
            OSTimeDlyHMSM(0, 0, 0,   2, OS_OPT_TIME_HMSM_STRICT, &os_err);
        }
    }
}
```

代码清单 6-6　鼠标任务

L6-6(1)如果熟悉 μC/OS-Ⅲ,应该知道,一个典型的 μC/OS-Ⅲ任务是通过一个无限循环的方式实现的。

L6-6(2)USBD_HID_IsConn()函数在 usbd_hid.c 中声明,如果 USB 设备和 HID 类实例都处于 configured 状态,它将返回 DEF_TRUE。

L6-6(3)如果 USBD_HID_IsConn()函数返回 DEF_FALSE,while 循环将继续检查设备和 HID 类实例的状态,直到它们转变为 configured 状态。

L6-6(4)函数 App_USBD_HID_MouseSetReport()在代码清单 6-5 中描述,它基于 X 轴和 Y 轴位移更新报告数据。

L6-6(5) USBD_HID_Wr()函数在 usbd_hid.c 中声明,通过中断 IN 端点来发送数据给主机。该端点在 STM32F107 上端点 1 或 EP1。由于该函数的第四个参数(timeout)为 0,函数将会阻塞,直到所有数据发送完。它将每隔 100ms 执行一次该操作,意味上主机上的鼠标指针将每 100ms 来回移动一次。

6.6　分析 USB 通信

为了捕捉 USB 通信,需要按照图 2-8“Beagle USB 480 协议分析仪:连接”说明,连接 Beagle USB 480。

当监控 USB 通信时,建议在连接 μC/Eval-STM32F107 之前,连接 Beagle USB 480 的分析端,并启动捕捉。这意味着,在启动捕捉前,确保 μC/Eval-STM32F107 CN8 上的 USB 电缆没有连接 Beagle USB 480 的捕捉端。这使 Beagle USB 480 可以捕捉枚举通信阶段的描述符信息。

使用 2.1.3“Total Phase Data Center 软件”一节创建的快捷方式启动 Total Phase Data Center 软件,或到软件包解压目录下,运行 Data Center.exe 启动软件。

一旦软件启动,它将自动检测 Beagle 并连接。

可以其查看顶部的 LED,如果红色和绿色 LED 点亮,表明 Beagle USB 480 已正确连接。

图 6-7　Data Center 软件图标

按下 Ctrl+R 或单击工具栏的 Run Capture 按键,启动捕捉。

在网格中,将看到一些新的行,显示类似于 Capture Started 的信息。

此时,可以连接 μC/Eval-STM32F107 CN8 的 USB 电缆,以连接 HID 设备。由于总线上已经有通信,它将在 transactions 网格中实时

显示通信信息。看到的第一个事务是枚举过程,对所有 USB 类是相同的过程,在附录 C“枚举过程”中描述。一旦设备被主机成功枚举,你不仅会看到鼠标指针在屏幕上来回移动,还将在网格中看到描述鼠标报告被发送到主机的所有行,如图 6-8 所示。

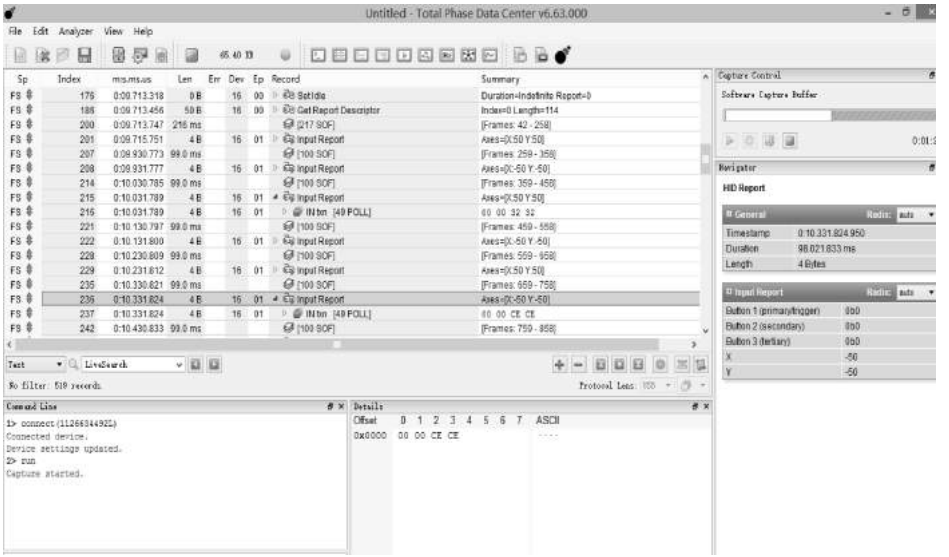


图 6-8　捕获的 USB 通信

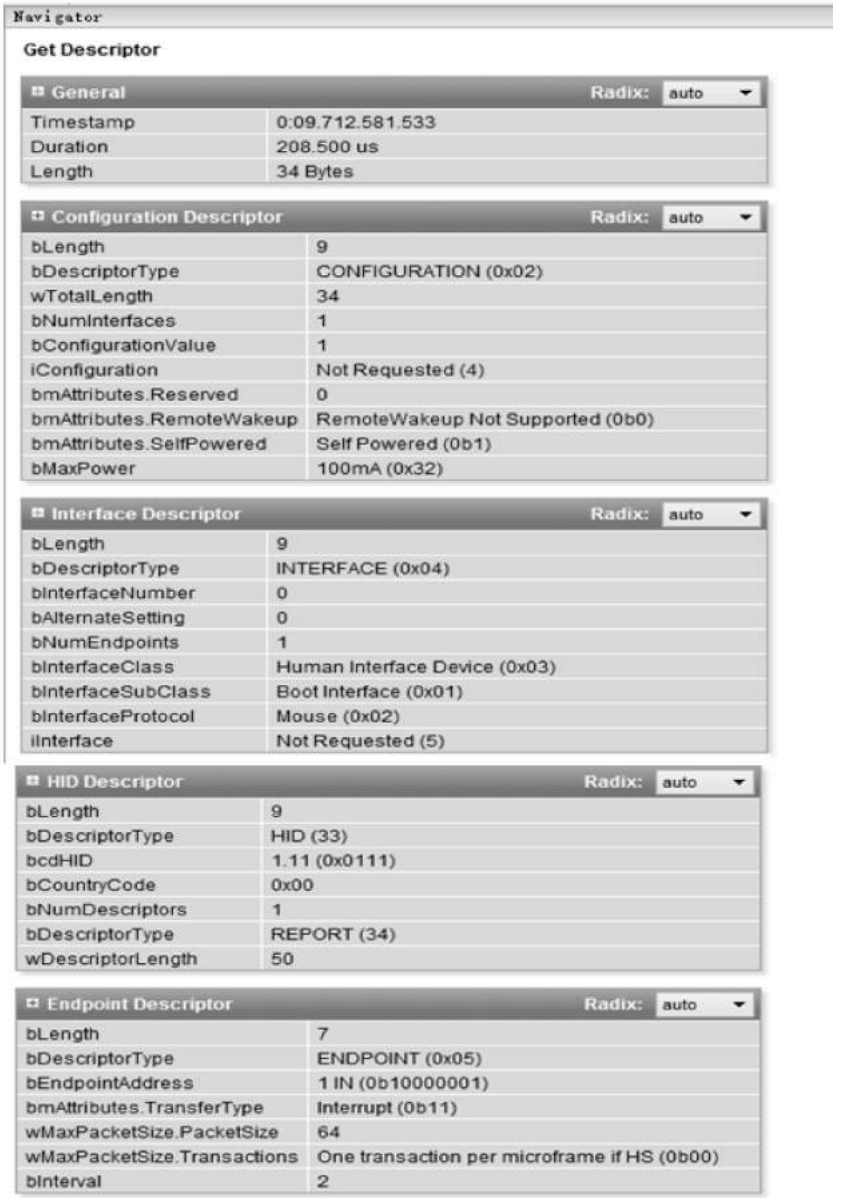
由于不能使用鼠标停止捕捉,可以再次按下 Ctrl+R 停止捕捉或通过断开,μC/Eval-STM32F107CN8 上的 USB 电缆,断开 HID 设备连接,进而分析 USB 捕捉。

6.6.1　获取配置描述符

Get Configuration Descriptor 如图 6-9 所示,指定了可用的配置数。获取的配置描述符中包含接口描述符和端点描述符,它们进一步描述了可用的接口和端点。本例中,仅有一个配置,带一个接口和一个端点。

6.6.2　获取报告描述符

HID 类特定的获取描述符请求之一是事务表格中标题为 Get Report Descriptor 的请求,如图 6-10 所示。该行展现了当 μC/Eval-STM32F107 通过发送《嵌入式协议栈 μC/USB-Device》表 9-2“鼠标报告描述符实例”列出的报告描述符,描述报告给主机的数据格式的时刻。


Navigator

Get Descriptor

General — Radix: auto

Timestamp	0:09.712.581.533
Duration	208.500 us
Length	34 Bytes

Configuration Descriptor — Radix: auto

bLength	9
bDescriptorType	CONFIGURATION (0x02)
wTotalLength	34
bNumInterfaces	1
bConfigurationValue	1
iConfiguration	Not Requested (4)
bmAttributes.Reserved	0
bmAttributes.RemoteWakeup	RemoteWakeup Not Supported (0b0)
bmAttributes.SelfPowered	Self Powered (0b1)
bMaxPower	100mA (0x32)

Interface Descriptor — Radix: auto

bLength	9
bDescriptorType	INTERFACE (0x04)
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	Human Interface Device (0x03)
bInterfaceSubClass	Boot Interface (0x01)
bInterfaceProtocol	Mouse (0x02)
iInterface	Not Requested (5)

HID Descriptor — Radix: auto

bLength	9
bDescriptorType	HID (33)
bcdHID	1.11 (0x0111)
bCountryCode	0x00
bNumDescriptors	1
bDescriptorType	REPORT (34)
wDescriptorLength	50

Endpoint Descriptor — Radix: auto

bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	1 IN (0b10000001)
bmAttributes.TransferType	Interrupt (0b11)
wMaxPacketSize.PacketSize	64
wMaxPacketSize.Transactions	One transaction per microframe if HS (0b00)
bInterval	2

图 6－9　获取配置描述符

Navigator

Get Report Descriptor

General	Radix: auto
Timestamp	0:09.713.456.533
Duration	142.500 us
Length	50 Bytes

Report Descriptor	Radix: auto
Usage Page	Generic Desktop Controls (0x01)
Usage	Mouse (0x02)
Collection	Application (0x01)
Usage	Pointer (0x01)
Collection	Physical (0x00)
Usage Page	Button (0x09)
Usage Minimum	Button 1 (primary/trigger) (0x01)
Usage Maximum	Button 3 (tertiary) (0x03)
Logical Minimum	0
Logical Maximum	1
Report Count	3
Report Size	1
Input	Data (0x02)
Report Count	1
Report Size	13
Input	Constant (0x01)
Usage Page	Generic Desktop Controls (0x01)
Usage	X (0x30)
Usage	Y (0x31)
Logical Minimum	-127
Logical Maximum	127
Report Size	8
Report Count	2
Input	Data (0x06)
End Collection	
End Collection	

图 6-10　捕获的 Get Report Descriptor

6.6.3　鼠标输入报告

如《嵌入式协议栈 μC/USB-Device》1.4“总线状态”一节所述,一旦设备从 addressed 状态转变为 configured 状态,设备可以使用。你将找到类似于图 6-11 所述的一系列的 Input Report 记录,按照代码清单 6-6 所示的鼠标任务,数据报告每隔 100ms 发送一次,因此,它是捕获最多的数据,所以在事务表格中,可以很容易的找到这些记录。

定位到事务表格中 Input Report 记录,注意 Navigator 窗口中的 Info 框如何报告 x 轴和 y 轴的位移,位移通过 app_usbd_hid.c 中声明的 App_USBD_HID_MouseSetReport()设置为 50。

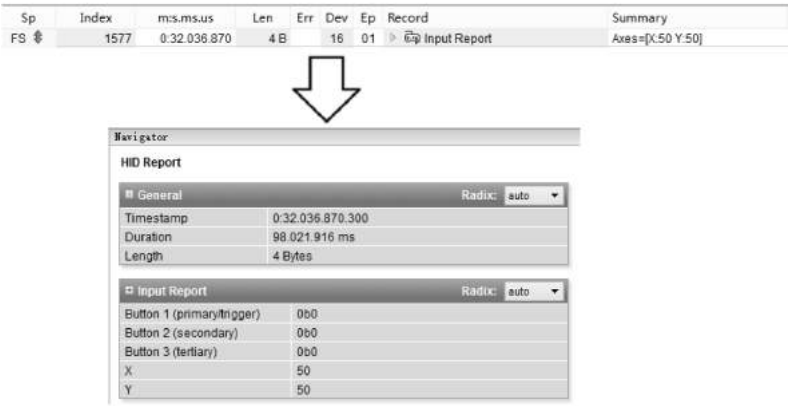


图 6-11　捕获的输入报告

还需注意图 6-11 中的 Ep 列如何显示,端点号 1 是中断输入端点,在 HID 类应用中用于数据通信,如《嵌入式协议栈 μC/USB-Device》第 9 章"用户接口设备类"所述。

图 6-12 显示的 Block 视图,提供了选定记录的另一种表示方式,它合并了事务表格中的层次布局和 Info 框中的详细信息。

选定记录所属的记录树在 Block 视图中显示,其中选定记录以蓝色高亮显示,可以展开行中第一行边上的箭头,以获取隐藏的字段。

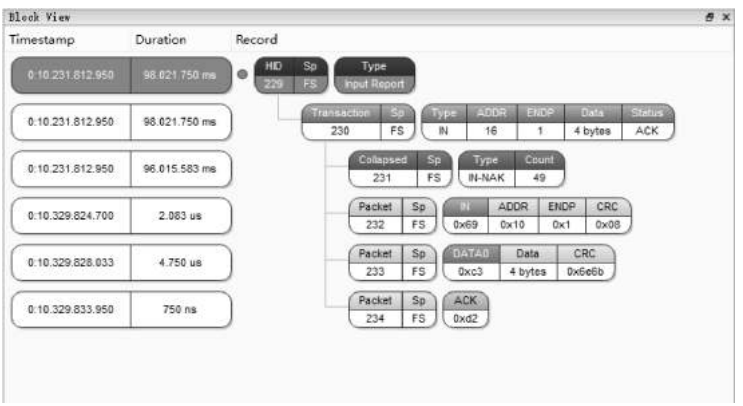


图 6-12　Block 视图中看到的输入报告

6.7　总　结

本章描述的实例是创建自己的 HID 设备的一个很好的参考。并说明了使用 USB 协议分析仪实现调试的重要性,介绍了 μC/USB-Device 协议栈和 HID 类。本章展示了如何运行该实例,并通过描述 USB 通信捕获和代码清单,解释了代码如何工作。

如果想一步使用该实例,意法半导体的 μC/Eval-STM32F107 是一个很好的平台,你可以简单的使用按键和加速器创建自己的鼠标,用于游戏控制。

第 7 章

MSC 例程:移动存储设备

MSC 类在《嵌入式协议栈 μC/USB-Device》第 10 章“大容量存储类”中介绍。本章介绍了一个演示应用。描述了如何使用 IAR EWARM 和 Beagle USB 480 协议分析仪在 μC/Eval-STM32F107 上运行 MSC 类演示实例。

USB 大容量存储类定义了一个协议,允许主机访问 USB 设备,实现文件传输。USB 设备在主机上显示为一个可移除的存储设备,因此可以通过拖放的方式实现文件传输。

本章描述的实例实现了一个 32K 容量的 USB 移动存储设备。可以创建小的文本文件来测试 μC/USB-Device 协议栈和其 MSC 类,如图 7-1 所示。

图 7-1　移动存储设备实例

7.1　在 IAR 中打开项目

打开 IAR EWARM 软件,导入本书配套软件包中的 C 语言工程,对所有的实例,仅需执行一次该操作,过程在 3.2.2 部分描述。

如果已经实现该过程,打开已选择路径下的工作区。

7.2　配置移动存储设备实例

从 Project Explorer(项目浏览器)中打开头文件,app_cfg.h,如图 7-2 所示。

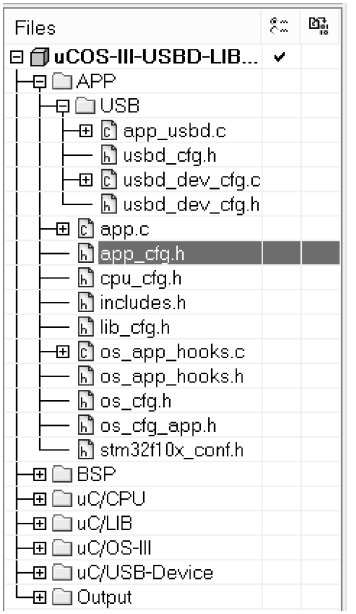


图 7-2　项目浏览器:配置文件

7.2.1　启用和禁用类

本书中介绍的所有实例,不仅位于同一工作空间,还位于同一项目中,为了使能 MSC 类和移动存储设备实例,app_cfg.h 中需要编辑的代码行如代码清单 7-1 所示:

```
/*
*********************************************************************************
*                                uC/USB-DEVICE APPLICATION CONFIGURATION
*********************************************************************************
*/

#define  APP_CFG_USBD_CDC_EN                                DEF_DISABLED        (1)
#define  APP_CFG_USBD_HID_EN                                DEF_DISABLED        (2)
#define  APP_CFG_USBD_MSC_EN                                DEF_ENABLED         (3)
#define  APP_CFG_USBD_VENDOR_EN                             DEF_DISABLED        (4)
#define  APP_CFG_USBD_PHDC_EN                               DEF_DISABLED        (5)
```

代码清单 7-1　配置移动存储设备实例:app_cfg.h

L7-1(1)设置 APP_CFG_USBD_CDC_EN 为 DEF_DISABLED 禁用 CDC 类。

L7-1(2)设置 APP_CFG_USBD_HID_EN 为 DEF_DISABLED 禁用 HID 类。

L7-1(3)设置 APP_CFG_USBD_MSC_EN 为 DEF_ENABLED 启用 MSC 类。

L7-1(4)为了运行移动存储设备实例,通过设置 APP_CFG_USBD_VENDOR_EN 为 DEF_DISABLED 禁用用户自定义 Vendor 类。

L7-1(5)设置 APP_CFG_USBD_PHDC_EN 为 DEF_DISABLED 禁用 PHDC 类。

7.2.2　配置存储容量

本章介绍的实例将使用 STM32F107 的一些 RAM 块。这些内存块作为非易失性内存空间使用。这被称之为 RAMDisk,尽管存储在这种类型介质上的数据在开发板断电时会丢失,它足以演示 MSC 类。当然,也可以选择使用文件系统协议栈,如 Micriμm 的 μC/FS,甚至是基于私有文件系统的存储介质来演示。在使用私有文件系统的存储介质时,需要创建一个存储层接口,实现存储介质和 μC/USB MSC 设备的通信。请参考《嵌入式协议栈 μC/USB-Device》10.6“MSC 的存储层移植”一节,学习如何实现该存储层。

为了在 app_cfg.h 中配置 RAMDisk,必须指定分配的块在 RAM 中的位置,块的数量和块大小。

不需要改变本例中的任何设置,代码清单 7-2 列出了的需要设置的预处理指令,仅供参考。

```
#define  USBD_RAMDISK_CFG_NBR_UNITS          1          (1)
#define  USBD_RAMDISK_CFG_BLK_SIZE           512        (2)
#define  USBD_RAMDISK_CFG_NBR_BLKS           44         (3)
#define  USBD_RAMDISK_CFG_BASE_ADDR          0          (4)
```

代码清单 7-2 RAMDisk 配置

L7-2(1)RAMDisk 区的逻辑单元数。

L7-2(2)RAMDisk 块的字节大小。

L7-2(3)RAMDisk 逻辑单元中块的数量。本例中,44 个 512 相当于 22508 字节的存储容量。请注意,部分容量将保留,用于文件系统。

L7-2(4)内存中 RAMDisk 的起始地址(基址)。该常量用来定义 RAMDisk 的数据区,如果设置为 0,表示 RAMDisk 的数据区作为程序数据区的一个表。

7.3　构建移动存储设备实例项目

为了构建该项目,按下 F7 按键,如 3.2.4“在 EWARM 中构建项目”一节所述。

7.4　运行移动存储设备实例项目

由于本例实现了一个标准的移动存储设备,Windows 将使用本地驱动程序安装它,不需要在主机上安装任何驱动程序。

7.4.1　连接开发板

按照图 2-8“Beagle USB 480 协议分析仪:连接”所示,连接开发板。不要连接 μC/Eval-STM32F107CN8 上的 USB 电缆。

7.4.2　启动调试会话

按照 3.2.5“在 EWARM 中启动调试会话”一节所述,启动调试会话,按下 F5 使程序全速运行。

7.4.3　安装 MSC 设备

此时,μC/USB-Device 协议栈已经运行,并等待 USB 事件,如连接 USB 设备。用 USB 电缆连接 μC/Eval-STM32F107 开发板的 CN8 和 Beagle USB 480 的捕捉端,如图 2-8“Beagle USB 480 协议分析仪:连接”所示。如果没有 USB 协议分析仪,可以将电缆的另一端直接连接到主机任何可用的 USB 插座。

第一次运行该实例,主机将安装 μC/Eval-STM32F107 作为移动存储设备。Win8 系统会在在桌面弹出一个消息框,通知你设备正在安装,如图 7-3 所示。

图 7-3　Windows 安装 MSC 设备

7.4.4　格式化新的移动存储设备

如果是第一次安装该设备,Windows 将弹出一个类似于图 7-4 所示的提示框,询问是否格式化硬盘驱动器。

图 7-4　Windows 格式化新的移动存储设备

格式化设备操作是安全的,因为它仅影响 app_cfg.h 中配置的 RAM 块。

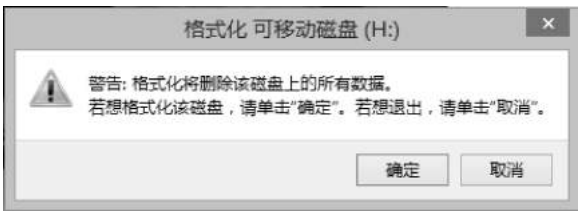


图 7-5　Windows 格式化警告

配置卷标,使用一个将来可以容易识别的名称,例如 STM32F107。可以选择想使用的文件系统,本例测试时使用 FAT 文件系统,如图 7-6 所示。

图 7-6　Windows 格式化配置

一旦格式化完成,Windows 将使用类似于图 7-7 所示的消息框通知你。

图 7-7　Windows 格式化完成

7.4.5　浏览新的移动存储设备

一些 Windows 系统版本将自动打开到新的移动存储设备路径,否则,手动打开 Windows 资源管理器并定位到新的 MSC 设备:

图 7-8　Windows 资源管理器显示新的移动存储设备

右击表示新的移动存储设备的图标,选择属性,查看容量,如图 7-9 所示。

注意,Windows 报告的容量为 2.00KB 字节,而 app_cfg.h 中配置的 RAMDisk 容量为 22KB。这是因为保留了 RAMDisk 的部分扇区,用于 FAT 文件系统。

还可以从控制面板中打开 Windows 设备管理器,在设备列表中获取新安装的 MSC 设备信息,如图 7-10 所示。

图 7-9　Windows 显示新的移动存储设备的容量

7.4.6　测试新的移动存储设备

此时,可以写一些数据到新设备中,以使用 μC/USB-Device 协议栈和其 MSC 类。

打开记事本,写一些文本文件用于测试,并将文件保存在新设备中,如图 7-11 所示。

只要 μC/Eval-STM32F107 不断电,文件将保持在内存中。意味着,如果通过连接到 μC/Eval-STM32F107 CN4 的 J-Link 端口的 USB 电缆供电,可以先断开 CN8 上的 USB 电缆,再连接,通过 Windows 查看移动存储设备上的测试文件是否保留。

可以按下图 3-5"调试界面"所示的调试工具栏中的 break 按键停止程序执行。

图 7－10　设备管理器显示新安装的设备

图 7－11　NotePad 测试

7.5　代码如何工作

初始化 MSC 类和移动存储设备实例的函数在应用文件 app_usbd_msc.c 中声明。下面的代码清单描述了初始化过程:

代码清单 7－3 初始化移动存储设备实例

L7－3(1)USBD_MSC_Init()函数在 usbd_msc.c 中声明,用于初始化类需要的

所有内部结构和变量,包括初始化 MSC 任务处理程序。MSC 任务处理程序负责实现 MSC 协议,允许设备和主机间的通信。关于该函数的更多信息,查阅《嵌入式协议栈 μC/USB-Device》MSC API 参考手册 E.1.1"USBD_MSC_Init()"部分。

L7-3(2)USBD_MAS_Add()函数也在 usbd_msc.c 中声明,该函数负责创建新的 MSC 类实例,成功分配新类后,函数返回类号。类号用来识别该实例,并作为所有 MSC API 函数的一个输入参数。

```
CPU_BOOLEAN   App_USBD_MSC_Init (CPU_INT08U  dev_nbr,
                                 CPU_INT08U  cfg_hs,
                                 CPU_INT08U  cfg_fs)
{
    USBD_ERR      err;
    CPU_INT08U    msc_nbr;
    CPU_BOOLEAN   valid;

    APP_TRACE_DBG(("        Initializing MSC class ... \r\n"));

    USBD_MSC_Init(&err);                                                          (1)
    if (err != USBD_ERR_NONE) {
        APP_TRACE_DBG(("        ... could not initialize MSC class w/err = %d\r\n\r\n", err));
        return (DEF_FAIL);
    }

    msc_nbr = USBD_MSC_Add(&err);                                                 (2)
```

初始化过程的下一步是添加 MSC 类的新实例到 USB 全速设备配置中,如代码清单 7-4 所示:

```
                                /*  Add MSC class to the FS dflt cfg  .*/   (1)

    valid = USBD_MSC_CfgAdd (msc_nbr,
                             dev_nbr,
                             cfg_fs,
                             &err);

    if (valid != DEF_YES) {
        APP_TRACE_DBG(("        ... could not add msc instance #%d to FS configuration w/err = %d\r\n\r\n", msc_nbr, err));
        return (DEF_FAIL);
    }
```

代码清单 7-4　添加 MSC 类实例到全速配置

L7-4(1)继续添加新的 MSC 实例到全速配置。如果 MSC 类实例可以添加到 USB 设备配置中,没有任何如内存分配,无效参数等错误,USBD_MSC_CfgAdd()函数返回 DEF_YES。关于该函数参数和返回错误代码的更多信息,见《嵌入式协议栈 μC/USB-Device》MSC API 参考手册 E.1.3"USBD_MSC_CfgAdd()"部分。

App_USBD_MSC_Init()函数最后要做的是,调用 USBD_MSC_LunAdd(),为 MSC 接口添加逻辑单元号。如代码清单 7-5 所示。在这里可以指定逻辑单元的类型和卷的详细信息,如供应商 ID 和产品 ID。逻辑单元通过字符串名由存储设备驱动名称(如 RAMDisk)和从 0 开始的逻辑单元号组成的设备驱动程序添加,本例中,指向逻辑单元驱动的指针称之为 ram:0。

```
                                                    /* Add Logical Unit to MSC interface. */


USBD_MSC_LunAdd((void *)"ram:0:",
                        msc_nbr,
                (void *)"Micrium",
                (void *)"MSC FS Storage",
                        0x00,
                        DEF_FALSE,
                       &err);
if (err != USBD_ERR_NONE) {
    APP_TRACE_DBG(("        ... could not add LU to MSC class w/err = %d\r\n\r\n", err));
    return (DEF_FAIL);
}

return (DEF_OK);
}
```

代码清单 7-5 添加逻辑单元到 MSC 类实例

7.6　分析 USB 通信

为了捕捉 USB 通信,需要按照图 2-8“Beagle USB 480 协议分析仪:连接”说明,连接 Beagle USB 480。

当监控 USB 通信时,建议在连接 μC/Eval-STM32F107 之前,连接 Beagle USB 480 的分析端,并启动捕捉。这意味着,在启动捕捉前,确保 μC/Eval-STM32F107 开发板 CN8 上的 USB 电缆没有连接 Beagle USB 480 的捕捉端。这使 Beagle USB 480 可以捕捉枚举通信阶段的描述符信息。

使用 2.1.3“Total Phase Data Center 软件”一节创建的快捷方式启动 Total Phase Data Center 软件,或到软件包解压目录下,运行 Data Center.exe 启动软件。

一旦软件启动,它将自动检测 Beagle 并连接。

可以查看其顶部的 LED,如果红色和绿色 LED 点亮,表明 Beagle USB 480 已正确连接。

按下 Ctrl+R 或单击工具栏的 Run Capture 按键,启动捕捉。

图 7-12　Data Center 软件图标

在表格中,将看到一些新的行,显示类似于

Capture Started 的信息。

此时,可以连接 μC/Eval-STM32F107 开发板 CN8 的 USB 电缆,以连接 MSC 设备。

完成格式化移动存储设备的步骤,并按照 7.4“运行移动存储设备实例项目”一节描述的步骤创建测试文件。

由于总线上已经有通信,它将在事务表格中实时显示通信信息。看到的第一个事务是枚举过程,对所有 USB 类是相同的过程,在附录 C“枚举过程”中描述。

一旦设备被主机成功枚举,你不仅会看到格式化新的移动存储设备的 Windows 对话框,还将在事务表格中看到一堆新的行,包含描述 SCSI 命令的信息,SCSI 命令在《嵌入式协议栈 μC/USB-Device》书中表 10-3“SCSI 命令”部分描述。

FS	21	0:23.100.658	54.8 ms			<Reset> / <Target disconnecte...		
FS	22	0:23.155.541				<Full-speed>		
FS	23	0:23.156.046	15.0 ms			[16 SOF]	[Frames: 755 - 770]	
FS	24	0:23.171.060	0 B	00	00	Set Address	Address=02	
FS	34	0:23.172.048	9.00 ms			[10 SOF]	[Frames: 771 - 780]	
FS	35	0:23.181.236	18 B	02	00	Get Device Descriptor	Index=0 Length=18	.....@..2.....
FS	49	0:23.182.049	32.0 ms			[33 SOF]	[Frames: 781 - 813]	
FS	50	0:23.214.624	32 B	02	00	Get Configuration Descriptor	Index=0 Length=255	.. ..2.....P.....@.....§..
FS	64	0:23.215.053	2.08 us			[1 SOF]	[Frame: 814]	
FS	65	0:23.214.938	34 B	02	00	Get String Descriptor	Index=3 Length=255	".1.2.3.4.5.6.7.8.9.0.A.B.C.D...
FS	79	0:23.215.201	4 B	02	00	Get String Descriptor	Index=0 Length=255	....
FS	93	0:23.215.319	24 B	02	00	Get String Descriptor	Index=2 Length=255	..O.E.M. .P.R.O.D.U.C.T.
FS	107	0:23.215.572	0 B	02	00	Control Transfer (STALL)	Index=0 Length=10	
FS	116	0:23.216.053	88.0 ms			[89 SOF]	[Frames: 815 - 903]	
FS	117	0:23.304.661	18 B	02	00	Get Device Descriptor	Index=0 Length=18	.....@..2.....
FS	131	0:23.305.064	2.08 us			[1 SOF]	[Frame: 904]	
FS	132	0:23.304.874	9 B	02	00	Get Configuration Descriptor	Index=0 Length=9	2
FS	146	0:23.305.149	32 B	02	00	Get Configuration Descriptor	Index=0 Length=32	.. ..2.....P.....@.....§..
FS	160	0:23.305.393	2 B	02	00	Get String Descriptor	Index=0 Length=2	..
FS	174	0:23.305.534	4 B	02	00	Get String Descriptor	Index=0 Length=4	....
FS	188	0:23.305.680	2 B	02	00	Get String Descriptor	Index=3 Length=2	".
FS	202	0:23.306.064	2.08 us			[1 SOF]	[Frame: 905]	
FS	203	0:23.305.902	34 B	02	00	Get String Descriptor	Index=3 Length=34	".1.2.3.4.5.6.7.8.9.0.A.B.C.D...
FS	217	0:23.306.511	0 B	02	00	Set Configuration	Configuration=1	
FS	227	0:23.307.064	2.08 us			[1 SOF]	[Frame: 906]	
FS	228	0:23.307.068	1 B	02	00	Get Max LUN	Max LUN = 0	.
FS	242	0:23.307.220	36 B	02	01	Inquiry [0]	Passed	........Micrium.MSC FS Storage...
FS	260	0:23.307.477	36 B	02	01	Inquiry [0]	Passed	........Micrium.MSC FS Storage...

图 7-13　捕捉 USB 通信

可以再次按下 Ctrl+R 停止捕捉,进而分析 USB 捕捉。

7.6.1　获取配置描述符

主机通过一系列的获取描述符请求,例如事务表格中标题为 Get Configuration Descriptor 的记录来获取设备的能力,如图 7-14 所示,这些行反应了 μC/Eval-STM32F107 通过描述可以的 MSC 端点,阐述给主机可用的设备配置的时刻。

图 7-14　获取配置描述符捕获

7.6.2　MSC 协议

MSC 协议在《嵌入式协议栈 μC/USB-Device》10.1.1“大容量存储类协议”一节介绍,该节描述了 MSC 三个通信阶段中,主机和设备间交换的每个命令。图 7-15

图 7-15　MSC 协议的三个阶段

阐述了读取设备容量命令的三个阶段:

7.6.3　SCSI 命令

如《嵌入式协议栈 μC/USB-Device》1.4“总线状态”一节所述,一旦设备从 addressed 状态转换到 configured 状态,设备可以使用。你将发现一系列类型于图 7-16所示的记录。这些记录反映了主机发送的小型计算机系统接口(SCSI)命令。这些命令设置传输数据块和状态,及控制信息如设备的容量和数据交换是否准备就绪的具体请求。反映了当 Windows 格式化新的移动存储设备时刻的特定的 SCSI 命令,如图 7-16 所示。

FS	82008	0:38.857.123	8 B	02	01	Read Capacity [0]	Passed	...+....
FS	82026	0:38.857.947	2.08 us			[1 SOF]	[Frame: 71]	
FS	82027	0:38.857.866	36 B	02	01	Inquiry [0]	Passed	........Micrium.MSC FS Storage..
FS	82058	0:38.858.947	1.00 ms			[2 SOF]	[Frames: 72 - 73]	
FS	82059	0:38.859.230	512 B	02	01	Write [0]	LBA = 0 Length = 1 block (Pa...	t.o.o.l.t.i.p.s._.c.l.a.s.s.3...
FS	82156	0:38.860.947	2.08 us			[1 SOF]	[Frame: 74]	
FS	82157	0:38.860.584	512 B	02	01	Read [0]	LBA = 0 Length = 1 block (Pa...	t.o.o.l.t.i.p.s._.c.l.a.s.s.3...
FS	82210	0:38.861.948	2.08 us			[1 SOF]	[Frame: 75]	
FS	82211	0:38.861.543	512 B	02	01	Write [0]	LBA = 43 Length = 1 block (P...	.....
FS	82308	0:38.862.948	2.08 us			[1 SOF]	[Frame: 76]	
FS	82309	0:38.862.951	512 B	02	01	Read [0]	LBA = 43 Length = 1 block (P...	.....
FS	82362	0:38.863.948	1.00 ms			[2 SOF]	[Frames: 77 - 78]	
FS	82363	0:38.863.810	512 B	02	01	Write [0]	LBA = 6 Length = 1 block (Pa...	.....
FS	82463	0:38.865.948	2.08 us			[1 SOF]	[Frame: 79]	
FS	82464	0:38.865.382	512 B	02	01	Read [0]	LBA = 6 Length = 1 block (Pa...	.....
FS	82516	0:38.866.948	2.08 us			[1 SOF]	[Frame: 80]	
FS	82517	0:38.866.320	512 B	02	01	Write [0]	LBA = 7 Length = 1 block (Pa...	.....
FS	82610	0:38.867.948	2.08 us			[1 SOF]	[Frame: 81]	
FS	82611	0:38.867.658	512 B	02	01	Read [0]	LBA = 7 Length = 1 block (Pa...	.....
FS	82662	0:38.868.948	34.0 ms			[35 SOF]	[Frames: 82 - 116]	
FS	82663	0:38.868.693	16384 B	02	01	Write [0]	LBA = 8 Length = 32 blocks (...	STM32F107 .....?PCH.....
FS	85262	0:38.903.953	20.0 ms			[21 SOF]	[Frames: 117 - 137]	
FS	85263	0:38.903.695	16384 B	02	01	Read [0]	LBA = 8 Length = 32 blocks (...	STM32F107 .....?PCH.....
FS	86540	0:38.924.955	6.00 ms			[7 SOF]	[Frames: 138 - 144]	
FS	86541	0:38.924.580	3072 B	02	01	Write [0]	LBA = 0 Length = 6 blocks (P...	.<.MSDOS5.0.....,.....
FS	87074	0:38.931.956	3.00 ms			[4 SOF]	[Frames: 145 - 148]	
FS	87075	0:38.931.758	3072 B	02	01	Read [0]	LBA = 0 Length = 6 blocks (P...	.<.MSDOS5.0.....,.....
FS	87325	0:38.935.957	2.08 us			[1 SOF]	[Frame: 149]	
FS	87397	0:38.937.957	54.0 ms			[55 SOF]	[Frames: 151 - 205]	
FS	87398	0:38.992.229		02	01	Test Unit Ready [0]	Passed	
FS	87410	0:38.992.963	208 us			[1 SOF]	[Frame: 206]	
FS	87411	0:38.992.400	512 B	02	01	Read [0]	LBA = 0 Length = 1 block (Fa...	.<.MSDOS5.)....,.....
FS	87463	0:38.993.455		02	01	Test Unit Ready [0]	Passed	
FS	87475	0:38.993.655	8 B	02	01	Read Capacity [0]	Passed	...+....
FS	87493	0:38.993.964	208 us			[1 SOF]	[Frame: 207]	
FS	87494	0:38.993.861	0 B	02	01	Read Capacity [0]	Passed	...+....
FS	87517	0:38.994.122	512 B	02	01	Read [0]	LBA = 0 Length = 1 block (Fa...	.<.MSDOS5.)....,.....
FS	87570	0:38.994.964	2.16 us			[1 SOF]	[Frame: 208]	
FS	87571	0:38.994.967	512 B	02	01	Read [0]	LBA = 0 Length = 1 block (Fa...	.<.MSDOS5.)....,.....
FS	87624	0:38.995.964	208 us			[1 SOF]	[Frame: 209]	
FS	87625	0:38.995.773	8 B	02	01	Read Capacity [0]	Passed	...+....
FS	87642	0:38.996.016	8 B	02	01	Read Capacity [0]	Passed	...+....
FS	87660	0:38.996.964	208 us			[1 SOF]	[Frame: 210]	
FS	87661	0:38.996.233	512 B	02	01	Read [0]	LBA = 0 Length = 1 block (Fa...	.<.MSDOS5.)....,.....
FS	87717	0:38.997.195	8 B	02	01	Read Capacity [0]	Passed	...+....
FS	87734	0:38.997.964	5.90 ms			[6 SOF]	[Frames: 211 - 216]	
FS	87735	0:38.997.478	4096 B	02	01	Read [0]	LBA = 0 Length = 8 blocks (P...	.<.MSDOS5.)....,.....
FS	88065	0:39.003.965	4.90 ms			[5 SOF]	[Frames: 217 - 221]	
FS	88066	0:39.003.067	4096 B	02	01	Read [0]	LBA = 8 Length = 8 blocks (P...	STM32F107 .....?PCH.....
FS	88396	0:39.008.965	2.08 us			[1 SOF]	[Frame: 222]	
FS	88397	0:39.008.655	20 B	02	01	Mode Sense [0]	Passed	.....
FS	88428	0:39.009.657		02	01	Test Unit Ready [0]	Passed	
FS	88440	0:39.009.965	2.16 us			[1 SOF]	[Frame: 223]	
FS	88441	0:39.009.873		02	01	Test Unit Ready [0]	Passed	
FS	88453	0:39.010.100	8 B	02	01	Read Capacity [0]	Passed	...+....
FS	88471	0:39.010.966	208 us			[1 SOF]	[Frame: 224]	
FS	88472	0:39.010.351	512 B	02	01	Read [0]	LBA = 2 Length = 1 block (Fa...	.....
FS	88523	0:39.011.300		02	01	Test Unit Ready [0]	Passed	

图 7-16　格式化时的 SCSI 命令

μC/USB-Device 协议栈的 SCSI 接口在文件 usbd_scsi.c 中实现,包含表 10-3“SCSI 命令”列出的 SCSI 命令。

请注意图 7-17 中 Ep 列如何显示,端点号 2 是块输出端点,这意味着该端点用于 MSC 类应用中的数据通信,如第 10 章“大容量存储类”所述。

μC/Eval-STM32F107 端点表作为 μC/USB-Device 驱动的板级支持包的一部分,在文件 usbd_bsp_stm32fxxx.c 中声明(见附录 B“μC/Eval-STM32F107 的端点信息表”)。

写命令

现在将注意力集中到其中一个 SCSI 命令,图 7-17 所示的 Write 命令是你用记事本保存测试文件是捕获的。

注意:写命令如何包含 7-6-2“MSC 协议”一节描述的 MSC 协议的三个阶段。

μC/USB-Device 设备协议栈的 SCSI 命令在 usbd_scsi.c 文件中实现,该文件包含《嵌入式协议栈 μC/USB-Device》书中表 10-3“SCSI 命令”中描述的 SCSI 命令。

还需注意图 7-16 中 Ep 列的内容,该列显示了端点 1 作为批量输出端点,用于 MSC 类通信的数据传输,关于 MSC 类的详细信息,请参阅《嵌入式协议栈 μC/USB-Device》书第 10 章“大容量存储类”。

STM32F107 的端点信息表在文件 μC/USB-Device 设备驱动的板级支持包 usbd_bsp_stm32fxxx.c 文件中声明(参见附录 B“端点信息表”一节)。

写命令

现在我们来关注一下 SCSI 命令,图 7-17 所示的写命令,在保存写字板创建的测试文本文件时捕获。

FS	99286	1:28.428.749	512 B	02	01	Write [0]	LBA = 43 Length = 1 block (P...	Testing ..C/USB-Device and the...
FS	99287	1:28.428.749	31 B	02	01	Command Transport		USBC...V.....*....+.....
FS	99292	1:28.428.876	512 B	02	01	Data Transport	54 65 73 74 69 6E 67 20...	Testing ..C/USB-Device and the...
FS	99293	1:28.428.876	64 B	02	01	OUT txn	54 65 73 74 69 6E 67 20...	Testing ..C/USB-Device and the...
FS	99294	1:28.428.876	3 B	02	01	OUT packet	E1 82 18	...
FS	99295	1:28.428.879	67 B	02	01	DATA0 packet	C3 54 65 73 74 69 6E 67...	.Testing ..C/USB-Device and th...
FS	99296	1:28.428.925	1 B	02	01	ACK packet	D2	.
FS	99297	1:28.428.927	64 B	02	01	OUT txn [2 POLL]	31 30 37 00 00 00 00 00...	107.....
FS	99310	1:28.429.148	64 B	02	01	OUT txn [2 POLL]	00 00 00 00 00 00 00 00...	.....
FS	99323	1:28.429.325	64 B	02	01	OUT txn [2 POLL]	00 00 00 00 00 00 00 00...	.....
FS	99336	1:28.429.502	64 B	02	01	OUT txn [2 POLL]	00 00 00 00 00 00 00 00...	.....
FS	99349	1:28.429.679	64 B	02	01	OUT txn [2 POLL]	00 00 00 00 00 00 00 00...	.....
FS	99362	1:28.429.856	64 B	02	01	OUT txn [1 POLL]	00 00 00 00 00 00 00 00...	.....
FS	99371	1:28.429.980	64 B	02	01	OUT txn [1 POLL]	00 00 00 00 00 00 00 00...	.....
FS	99380	1:28.430.162	13 B	02	01	Status Transport	Passed	USBS...V.....
FS	99386	1:28.431.093	298 ms			[299 SOF]	[Frames: 1676 - 1974]	
FS	99387	1:28.729.747		02	01	Test Unit Ready [0]	Passed	
FS	99398	1:28.730.128	694 ms			[695 SOF]	[Frames: 1975 - 621]	
FS	99399	1:29.424.821		02	01	Test Unit Ready [0]	Passed	
FS	99411	1:29.425.212	2.08 us			[1 SOF]	[Frame: 622]	
FS	99412	1:29.425.064		02	01	Test Unit Ready [0]	Passed	

图 7-17　写命令

注意:写命令的实现包含了 MSC 协议的 3 个阶段,详细信息参考 7.6.2“MSC 协议”一节。

还需注意 NAK 记录的值。180 个 NAK 记录总共使用了 7.90ms。这完全正常,NAK 握手包通常用于流量控制,指示一个 USB 设备暂时不能发送或接收数据。

7.7　总　结

本章描述的实例是创建自己的 MSC 设备的一个很好的参考,它介绍了 μC/USB-Device 设备协议栈和 MSC 类。本章还展示了如何运行该实例,并通过分析代码列表和 USB 通信捕获,解释了代码是如何工作的。

如果想进一步深入该实例,意法半导体的 μC/Eval-STM32F107 开发板是一个很好的平台,仅需要简单的添加一个文件系统协议栈如 μC/USB-FS,使用 μC/Eval-STM32F107 上的 CN6 Micro SD 卡,创建一个非易失性可移动存储设备。

第 8 章

PHDC 例程:通信监测仪

μC/USB-Device 设备协议栈和其 PHDC 类遵循医疗行业标准,在医疗设备间实现 USB 连接。PHDC 类在《嵌入式协议栈 μC/USB-Device》书的“个人健康设备类”中介绍。本章结束时介绍了一个 PHDC 的应用实例。本章叙述了如何使用 IAR EWARM 和 Beagle USB 480 协议分析仪在 μC/Eval-STM32F107 上运行 PHDC 类演示实例。

本书描述的实例不仅包含运行在 μC/Eval-STM32F107 上的嵌入式应用的源代码,还包含 Windows PC 上的一个 Visual Studio 项目软件,如图 8-1 所示。

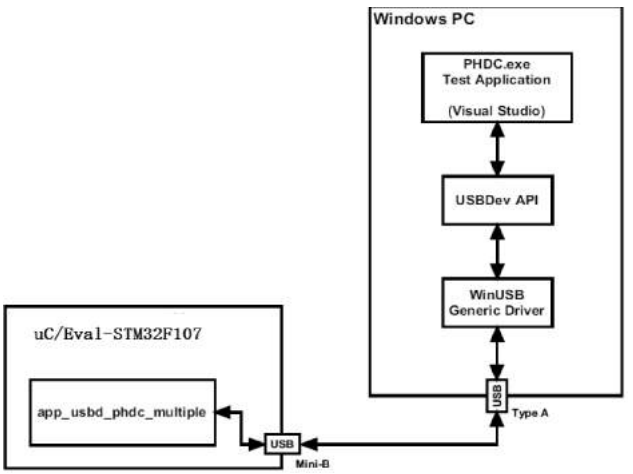


图 8-1　PHDC 实例

应用实例演示了如何在 μC/Eval-STM32F107 和主机之间,采用不同的延迟/可靠性(服务质量)组合来回的传输项目,如表 8-1 所列。

表 8-1　QoS 级别

QoS(延时/可靠性)	延迟	信息传输率	传输方向	一般应用
低/良好	<20ms	50～1.2Mb/s	IN	实时监测(快速采样)
中等/良好	<200ms	50～1.2Mb/s	IN	—

续表 8-1

QoS(延时/可靠性)	延迟	信息传输率	传输方向	一般应用
中等/更好	<200ms	几十字节	IN,OUT	测量参数的回放或者实时发送
中等/最佳	<200ms	几十字节	IN,OUT	事件,提醒,请求,控制,设备状态,监测对象的生理状况
高/最佳	<2s	几十字节	IN,OUT	生理状况和设备状态警告
极高/最佳	<20s	几十字节到 GB 级别	IN,OUT	传输报告、历史或者离线数据

8.1　在 IAR 里打开项目

打开 IAR EWARM 软件,导入本书配套软件包中的 C 语言工程,对所有的实例,仅需执行一次该操作,过程在 3.2.2 部分描述。

如果已经实现该过程,打开已选择路径下的工作区。

8.2　配置 PHDC 实例

从 Project Explorer(项目浏览器)中打开头文件 app_cfg.h,如图 8-2 所示。

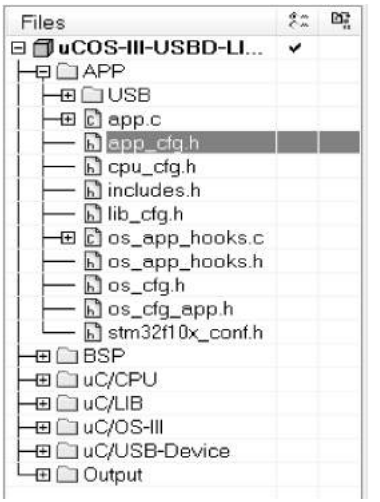


图 8-2　项目浏览器:配置文件

8.2.1　启用和禁用类

本书中介绍的所有实例,不仅位于同一工作空间,还位于同一项目中,为了使能 PHDC 类和实例,app_cfg.h 中需要编辑的代码行如代码清单 8-1 所示。

```
/*
*********************************************************************************************************
*                                    uC/USB-DEVICE APPLICATION CONFIGURATION
*********************************************************************************************************
*/

#define  APP_CFG_USBD_CDC_EN                            DEF_DISABLED          (1)
#define  APP_CFG_USBD_HID_EN                            DEF_DISABLED          (2)
#define  APP_CFG_USBD_MSC_EN                            DEF_DISABLED          (3)
#define  APP_CFG_USBD_VENDOR_EN                         DEF_DISABLED          (4)
#define  APP_CFG_USBD_PHDC_EN                           DEF_ENABLED           (5)

#define  APP_CFG_USBD_VENDOR_ECHO_SYNC_EN               DEF_DISABLED
#define  APP_CFG_USBD_VENDOR_ECHO_ASYNC_EN              DEF_DISABLED

#define  APP_CFG_USBD_HID_TEST_MOUSE_EN                 DEF_DISABLED

#define  APP_CFG_USBD_CDC_SERIAL_TEST_EN                DEF_DISABLED


#define  USBD_RAMDISK_CFG_NBR_UNITS                     1
#define  USBD_RAMDISK_CFG_BLK_SIZE                      512
#define  USBD_RAMDISK_CFG_NBR_BLKS                      44
#define  USBD_RAMDISK_CFG_BASE_ADDR                     0

#define  APP_CFG_USBD_PHDC_ITEM_DATA_LEN_MAX            8u                    (6)
#define  APP_CFG_USBD_PHDC_ITEM_NBR_MAX                 2u                    (7)
```

代码清单 8-1 配置 PHDC 实例:app_cfg.h

L8-1(1)设置 APP_CFG_USBD_CDC_EN 为 DEF_DISABLED 禁用 CDC 类。

L8-1(2)设置 APP_CFG_USBD_HID_EN 为 DEF_DISABLED 禁用 HID 类。

L8-1(3)设置 APP_CFG_USBD_MSC_EN 为 DEF_DISABLED 禁用 MSC 类。

L8-1(4)为了运行 PHDC 实例,通过设置 APP_CFG_USBD_VENDOR_EN 为 DEF_DISABLED 禁用供应商类。

L8-1(5)设置 APP_CFG_USBD_PHDC_EN 为 DEF_ENABLED 启用 PHDC 类。

L8-1(6)传输的数据最大字节数>=5。

L8-1(7)应用支持的最大对象数至少为 1。

8.3　构建 PHDC 实例项目

为了构建该项目,按下 F7 按键,如 3.2.4“在 EWARM 中构建项目”一节所述。

8.4　运行 PHDC 实例项目

为了运该实例项目,需要遵循下列步骤,包括准备 INF 文件,Windows 需要使用它加载其本地驱动。

8.4.1　连接开发板

按照图 2-8 所示连接“Beagle USB 480 协议分析仪:连接”,连接开发板,不要连

接 USB 电缆到 μC /Eval-STM32F107 开发板上的 CN8。

8.4.2　启动调试会话

按照 3.2.5“在 EWARM 中启动调试会话”一节所述,启动调试会话,按下 F5 使程序全速运行。

8.4.3　安装 PHDC 设备

此时,μC/USB-Device 协议栈已经运行,并等待 USB 事件,如连接 USB 设备。用 USB 电缆连接 μC/Eval-STM32F107 开发板的 CN8 和 Beagle USB 480 的捕捉端,如图 2-8“Beagle USB 480 协议分析仪:连接”所示。如果没有 USB 协议分析仪,可以将电缆的另一端直接连接到主机任何可用的 USB 插座。

第一次运行该实例,主机将安装 μC/Eval-STM32F107 作为 PHDC 设备。Win8 系统通过 Widows 消息框,通知你设备正在安装,如图 8-3 所示。

图 8-3　Windows 安装 PHDC 设备

如果是第一次安装该设备,Windows 将安装失败,因为这个特定的实例,虽然它使用了一个 Windows 本地驱动,它还需要一个 INF 文件来加载该驱动程序。

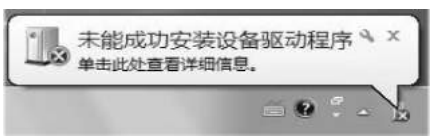


图 8-4　Windows 通知设备安装失败

INF 文件在《嵌入式协议栈 μC/USB-Device》书 3.2 节“关于 INF 文件”中有简要的描述,本书配套软件包中包含该实例需要的 INF 文件。仅需要确保 INF 文件和 PHDC 设备的硬件 ID 匹配。为此,可以使用一个文本编辑器如记事本,打开下面的

INF 文件:

$\Micrium\Software\uC-USB-Device-V4\App\Host\OS\Windows\PHDC\INF\WinUSB_single.inf

```
; ========== Manufacturer/Models sections ===========

[Manufacturer]
%ProviderName% = MyDevice_WinUSB,NTx86,NTamd64,NTia64

[MyDevice_WinUSB.NTx86]
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_0063
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_0063&MI_00

[MyDevice_WinUSB.NTamd64]
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_0063
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_0063&MI_00

[MyDevice_WinUSB.NTia64]
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_0063
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_0063&MI_00

; ================== Installation ==================
```

图 8-5　更新 INF 文件

定位到指定供应商 ID=FFFFh 和产品 ID=0063h 所在的行,如图 8-5 所示。确定他们与 usbd_dev_cfg.c 中配置的硬件 ID 匹配,如代码清单 8-2 所示:

```
/*
*********************************************************************************************************
*                                        USB DEVICE CONFIGURATION
*********************************************************************************************************
*/

USBD_DEV_CFG  USBD_DevCfg_STM32F_OTG = {
    0xFFFF,                                         /* Vendor  ID.
#if (APP_CFG_USBD_VENDOR_EN == DEF_ENABLED)
    0x1003,                                         /* Product ID (VENDOR).
#elif (APP_CFG_USBD_CDC_EN == DEF_ENABLED)
    0x1234,                                         /* Product ID (CDC).
#elif (APP_CFG_USBD_HID_EN == DEF_ENABLED)
    0x1233,                                         /* Product ID (HID).
#elif (APP_CFG_USBD_MSC_EN == DEF_ENABLED)
    0x1232,                                         /* Product ID (MSC).
#elif (APP_CFG_USBD_PHDC_EN == DEF_ENABLED)
    0x0063,                                         /* Product ID (PHDC).
#endif
    0x0100,                                         /* Device release number.
   "OEM MANUFACTURER",                              /* Manufacturer  string.
   "OEM PRODUCT",                                   /* Product       string.
   "1234567890ABCDEF",                              /* Serial number string.
    USBD_LANG_ID_ENGLISH_US                         /* String language ID.
};
```

代码清单 8-2 USB PHDC 设备配置

如果打开 Windows 设备管理器,将会找到一个新的安装失败的 PHDC 设备新表项。Windows 在其他设备组中列出该设备,如图 8-6 所示。

图 8-6　Windows 设备管理器

为了使用合适的驱动安装该设备,你需要给 Windows 提供更新过的 INF 文件,然后在设备管理器中,右击该设备名称,选择“更新驱动程序软件”,如图 8-6 所示。

Windows 将启动一系列的对话框,来指定驱动程序的选项,选择“浏览计算机以查找驱动程序软件”选项,如图 8-7 所示。

浏览到更新后的 INF 文件 WinUSB_single.inf 保存的路径,如图 8-8 所示。

由于该驱动没有数字签名,Windows 可能会给出一个警告信息。忽略该警告是安全的,选择“始终安装此驱动程序文件”,如图 8-9 所示:

Windows 需要几秒钟的时间安装新的 PHDC 设备,并将 INF 文件复制到 $\Windows\inf 目录下。

安装结束时,Windows 将显示消息框,如图 8-11 所示。

现在,你将在设备管理器 USB Sample Class 中看到一个新的表项,类似图 8-12 所示。

图 8-7　更新驱动软件:浏览

图 8-8　更新驱动软件:INF 文件路径

图 8-9　更新驱动软件:警告信息

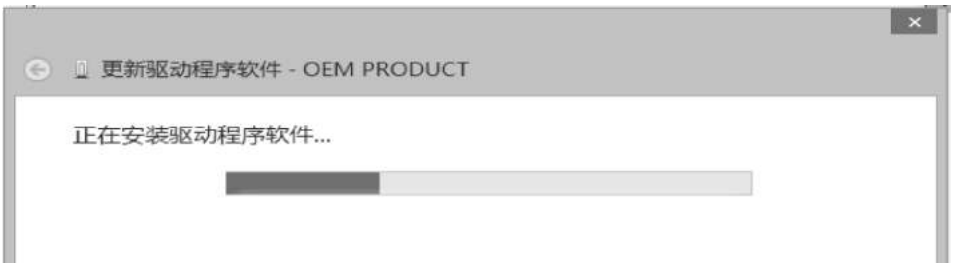


图 8-10　更新驱动软件:安装

图 8-11　更新驱动软件:结束

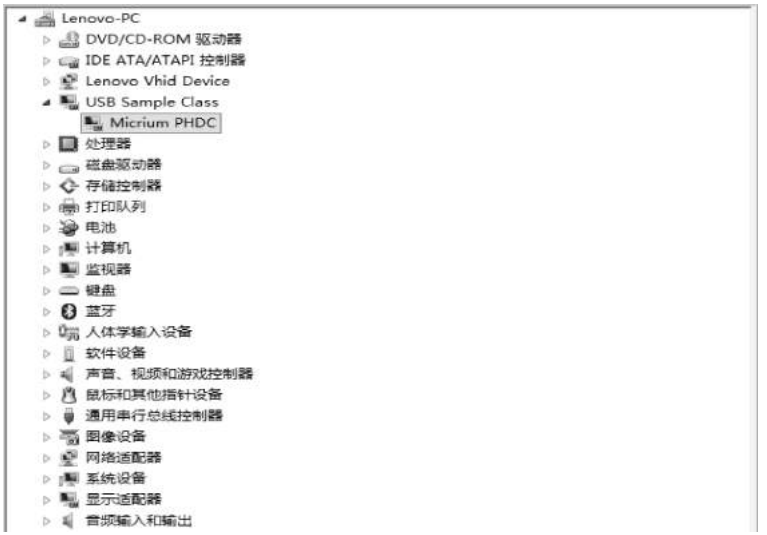


图 8-12　设备管理器显示新安装的设备

8.4.4　测试新的 PHDC 设备

此时,PHDC 设备已成功安装,可以使用书配套软件中的 PHDC 监控仪来验证 μC/USB-Device 协议栈和其 PHDC 类。

PHDC 监控仪是一个以 Visual Studio 项目形式发布的 Windows 应用,你可以尝试执行位于

\$\Micrium\Software\uC-USB-Device-V4\App\Host\OS\Windows\PHDC\Visual Studio 2010\exe\PHDC.exe 下的文件,

或打开位于

\$\Micrium\Software\uC-USB-Device-V4\App\Host\OS\Windows\PHDC\Visual Studio 2010\PHDC.sln 下的 Visual Studio 解决方案,重新构建可执行文件。

如图 8-13 所示,PHDC 监控仪是一个控制台,它能够让你通过以下方式来验证在 μC/Eval-STM32F107 上的 μC/USB-Device 协议栈和其 PHDC 类:

- 发送命令给 μC/Eval-STM32F107,让其周期性的连续不断的发送不同 QoS 级别的数据。
- 为每次数据传输计算和显示流量统计信息,如均值和标准差。

在一个特定的 QoS 和周期内发送的数据请求称之为一个“项目”,表 8-2 列出了每个项目的属性。

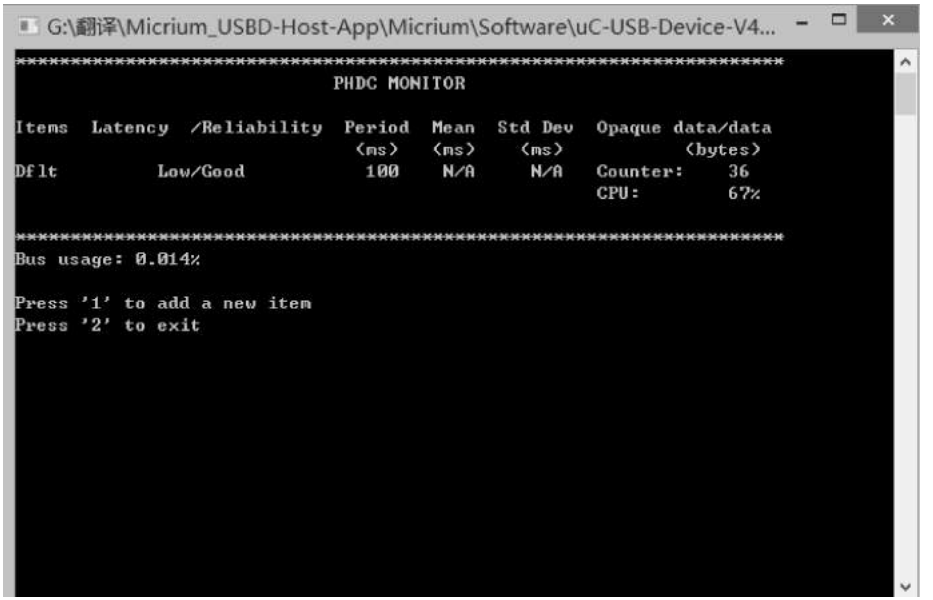


图 8-13　PHDC 监控仪控制台

表 8-2　项目属性

属性名字	说明	举例
周期	传输的周期	100ms
QoS	服务质量(延时/传输可靠性)	Low/Good
Opaque Data	PHDC 类的不透明应用数据	“Opaque Data”
Data	PHDC 类的透明数据	“Data”

第一次运行 PHDC 监控仪时,应用程序发送一个命令给 μC/Eval-STM32F107,创建一个 QoS 级别为 Low/Good,周期为 100ms 的默认项目,通过在控制台中按“1”可以创建更多的项来评估不同的 QoS 级别。

控制台将提示每项的属性,如图 8-14 所示。

尝试添加更多的项目来评估不同的 QoS 和周期,可以采用类似于图 8-15 所示的方式结束一个控制台的应用。

8.5　代码如何工作

PHDC 类实例在 app_usbd_phdc_multiple.c 中以任务方式实现,其中接收任务名字为 App_USBD_PHDC_RxCommTask(),一个或多个发送任务名字 App_USBD_PHDC_TxCommTask(),如图 8-16 所示。

图 8-14　PHDC 监控仪添加一个项目

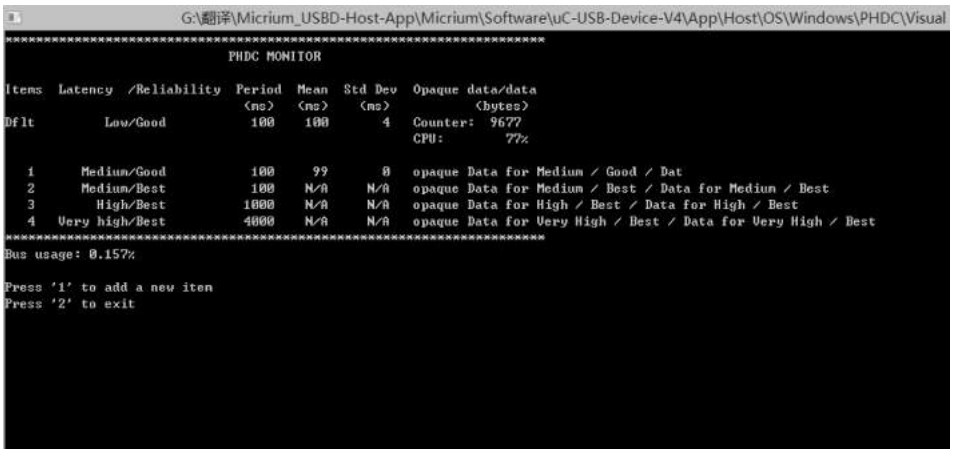


图 8-15　PHDC 监控仪多个项

F8-17(1)本例描述的 PHDC 传输采用了非常高的 QoS,高级和中级延迟,因此,运行模式需要 STM32F107 的一个批量输出端点和一个批量输入端点,如《嵌入式协议栈 μC/USB-Device》书中 11.1.2"操作模型"一节所述。

F8-17(2)负责所有 QoS 数据接收的 μC/OS-III 任务由 app_usbd_phdc_multiple.c 中的函数 App_USBD_PHDC_RXCommTask()实现。

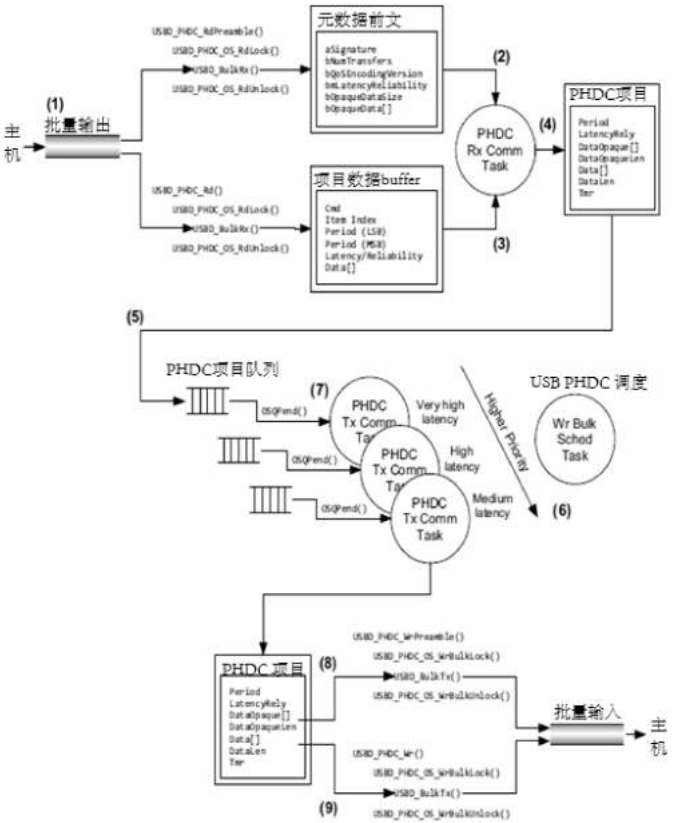


图 8-16　PHDC 类实例

元数据前文默认情况是禁用的,并且只有主机可以使能或禁用它。如果使能,RxCommTask 任务通过调用函数 USBD_PHDC_RdPreamble()读取前导符,以了解下次传输的 QoS,以及本次 QoS 使用的传输次数。关于元数据前文的更多信息见《嵌入式协议栈 μC/USB-Device》表 11-3“元数据前文”。

F8-17(3)主机的下一次传输是实际数据,不仅包含添加一个新的项目到测试的命令,还包含新项目的属性如 QoS 和周期。RxCommTask 任务调用函数 USBD_PHDC_Rd()读取该数据。关于该函数的更多信息见《嵌入式协议栈 μC/USB-Device》PHDC API 参考手册 F.1.9“USBD_PHDC_Rd()”。

F8-17(4) RxCommTask 任务根据最后收到报文的前导符和数据,创建一个新的 PHDC 项目。

F8-17(5) PHDC 项目插入到队列中。部分 PHDC 项目包含一个单次定时器,根据项目的周期,负责超时时发布一个信号。

F8-17(6) MicriμmPHDC 类实现的一个最好的特性是一个传输任务负责只传输特定的 QoS 级别的能力,因此,本例中,你可以有很多 TxCommTask 任务,对应不同的 QoS 级别。PHDC 类包含一个内部任务,负责使用批量输入端点传输数据到主机的调度。PHDC 调度器将为需要更低延迟(时延)的传输提供更高的优先级。PHDC 调度器依赖于操作系统,在 OS 层实现,其实现在 usbd_phdc.os.c 中的函数 USBD_PHDC_OS_WrBulkSchedTask()。

F8-17(7) TxCommTask 等待信号以出列下一个传输的项目。项目的传输周期由 PHDC 项目的定时器定义。由 PHDC 调度器根据其 QoS 决定任务允许使用的批量输入端点。

F8-17(8)与接收任务的处理方式相同,如果主机端使能,TxCommTask 任务将发送元数据前文。前导符包含项目的不透明数据,通过调用函数 USBD_PHDC_WrPreamble()发送。

F8-17(9) TxCommTask 通过递增的计数器和读取到的当前 CPU 使用率,来准备需要发送的数据。这两个读数是报文数据的一部分,通过调用函数 USBD_PHDC_Wr()发送。

8.5.1　初始化

初始化 PHDC 类和实例的函数在应用级文件 app_usbd_phdc_multiple.c 中声明,下面的代码清单描述了初始化过程。

```
CPU_BOOLEAN  App_USBD_PHDC_Init (CPU_INT08U  dev_nbr,
                                 CPU_INT08U  cfg_hs,
                                 CPU_INT08U  cfg_fs)
{
    CPU_BOOLEAN          valid_cfg_hs;
    CPU_BOOLEAN          valid_cfg_fs;
    USBD_ERR             err;
    OS_ERR               os_err;
    OS_TICK              tmr_period;
    LATENCY_RELY_FLAGS   latency_rely_flags;
    APP_USBD_PHDC_ITEM  *p_dflt_item;


    USBD_PHDC_Init(&err);                                   /* -------------------- INIT PHDC ---------------------- */   (1)
    if (err != USBD_ERR_NONE) {
        return (DEF_FAIL);
    }
                                                            /* Add PHDC instance.                                    */   (2)
    App_USBD_PHDC_ClassNbr = USBD_PHDC_Add(DEF_NO,          /* Vendor-defined data format.                           */
                                           DEF_YES,         /* Preamble capable.                                     */
                                           App_USBD_PHDC_SetPreambleEn,
                                           10,
                                           &err);
    if (err != USBD_ERR_NONE) {
        return (DEF_FAIL);
    }
```

代码清单 8-3 初始化 PHDC 类实例

L8-3(1)USBD_PHDC_Init()函数在 usbd_phdc.c 中声明,用来初始化类需要

的所有内部结构和变量。关于该函数的更多信息,请参阅《嵌入式协议栈 μC/USB-Device》书 PHDC API 参考手册 F.1.1"USBD_PHDC_Init()"一节。

L8-3(2)USBD_PHDC_Add()函数也在 usbd_phdc.c 中声明,它负责创建一个新的 PHDC 类实例。如果创建时,没有任何错误,函数返回分配给全局变量 App_USBD_PHDC_ClassNbr 用于标识的值。

初始化过程的下一步是采用正确的 QoS 和不透明数据类型来配置读写管道,如代码清单 8-4 所示。

```
latency_rely_flags = USBD_PHDC_LATENCY_VERYHIGH_RELY_BEST |
                     USBD_PHDC_LATENCY_HIGH_RELY_BEST     |
                     USBD_PHDC_LATENCY_MEDIUM_RELY_BEST;
USBD_PHDC_RdCfg(App_USBD_PHDC_ClassNbr,                  /* Cfg rd xfers with all possible latency/rely flags. */
                latency_rely_flags,
                App_USBD_PHDC_OpaqueDataRx,
                sizeof(App_USBD_PHDC_OpaqueDataRx),
                &err);                                                                          (1)
if (err != USBD_ERR_NONE) {
    return (DEF_FAIL);
}


latency_rely_flags = USBD_PHDC_LATENCY_VERYHIGH_RELY_BEST |
                     USBD_PHDC_LATENCY_HIGH_RELY_BEST     |
                     USBD_PHDC_LATENCY_MEDIUM_RELY_BEST   |
                     USBD_PHDC_LATENCY_MEDIUM_RELY_BETTER |
                     USBD_PHDC_LATENCY_MEDIUM_RELY_GOOD;
USBD_PHDC_WrCfg(App_USBD_PHDC_ClassNbr,                  /* Cfg very high, high and medium xfers with metadata.*/
                latency_rely_flags,
                App_USBD_PHDC_OpaqueDataTx,
                sizeof(App_USBD_PHDC_OpaqueDataTx),
                &err);                                                                          (2)

if (err != USBD_ERR_NONE) {
    return (DEF_FAIL);
}

USBD_PHDC_WrCfg(App_USBD_PHDC_ClassNbr,                  /* Cfg low latency xfers with specific metadadata. */
                USBD_PHDC_LATENCY_LOW_RELY_GOOD,
                App_USBD_PHDC_OpaqueDataTxLowLatency,
                sizeof(App_USBD_PHDC_OpaqueDataTxLowLatency),
                &err);                                                                          (3)

if (err != USBD_ERR_NONE) {
    return (DEF_FAIL);
}
```

代码清单 8-4 配置 QoS

L8-4(1)函数 USBD_PHDC_RdCfg()也位于 usbd_phdc.c 中,该函数负责配置通信管道,将数据从主机发送到设备。函数带有参数如 QoS 和执行不透明数据缓冲区的指针等。这一步配置的时延(非常高,高,中等)需要 STM32F107 的一个批量输出管道。

L8-4(2)函数 USBD_PHDC_WrCfg()与前一个函数类似,它配置将数据从设备发送到主机的管道。这一步配置的时延(非常高,高,中等)需要 STM32F017 的一个批量输入管道。

L8-4(3)从设备到主机的低延时时间传输,例如用于实时监控和高采样速率,都是通过一个中断输入管道完成。该步中调用函数 USBD_PHDC_WrCfg()配置该管道。

初始化过程的下一步是添加新的 PHDC 实例到全速设备配置中,如代码清单 8-5 所示:

```
if (cfg_fs != USBD_CFG_NBR_NONE) {
  valid_cfg_fs = USBD_PHDC_CfgAdd(App_USBD_PHDC_ClassNbr,
                                  dev_nbr,
                                  cfg_fs,
                                  &err);              /* Add PHDC class to FS dflt cfg.                    */
}
if (valid_cfg_hs == DEF_NO) {
    return (DEF_FAIL);
}
```

代码清单 8-5 添加 PHDC 类实例到全速配置

初始化过程的最后几步是真正的应用代码初始化。首先是创建默认项目。回想 8.4.4"测试新的 PHDC 设备"一节,当你第一次启动控制台应用时,它已经有了一个默认配置的的项目,配置为周期 100ms,QoS 级别为低延迟/好的可靠性。下面的代码是函数 App_USBD_PHDC_Init()创建默认项目的部分:

```
/* -------------------- INIT APP ---------------------- */
p_dflt_item               = &App_USBD_PHDC_Items[0];       /* Init dflt item.
p_dflt_item->Period       = APP_USBD_PHDC_ITEM_DFLT_PERIOD;
p_dflt_item->LatencyRely  = USBD_PHDC_LATENCY_LOW_RELY_GOOD;
p_dflt_item->DataLen      = APP_USBD_PHDC_ITEM_DATA_LEN_MIN + 3;
p_dflt_item->DataOpaqueLen = 0;
p_dflt_item->Data[1]      = 0;                          /* See Note 1 of APPLICATION ITEM section.
p_dflt_item->Data[2]      = (CPU_INT08U)APP_USBD_PHDC_ITEM_DFLT_PERIOD;
p_dflt_item->Data[3]      = (CPU_INT08U)(APP_USBD_PHDC_ITEM_DFLT_PERIOD >> 8);
p_dflt_item->Data[5]      = 0;                          /* Ctr value, 0 by dflt.
p_dflt_item->Data[6]      = 0;
p_dflt_item->Data[7]      = OSStatTaskCPUUsage;
```

```
tmr_period = ((((OS_TICK)APP_USBD_PHDC_ITEM_DFLT_PERIOD * OSCfg_TmrTaskRate_Hz)  + 1000u - 1u) / 1000u);
OSTmrCreate(          &p_dflt_item->Tmr,                  /* Create tmr for dflt item.
            (CPU_CHAR *)0,
                       tmr_period,
            (OS_TICK   )0,
                       OS_OPT_TMR_ONE_SHOT,
                       App_USBD_PHDC_TmrCallback,
            (void     *)p_dflt_item,
                       &os_err);

if (os_err != OS_ERR_NONE) {
    APP_TRACE_DBG(("Could not add default item timer w/err = %d\r\n", os_err));
    return (DEF_FAIL);
}
```

代码清单 8-6 创建默认项目

取决于 PHDC 优先级数的最大值,初始化函数创建一个基于 μC/OS-Ⅲ的消息队列,来保存需要传输的下一个项目。μC/OS-Ⅲ函数 OSQCreate()用来创建新的消息队列,如代码清单 8-7 所示:

```
                                        /*create one Q per xfer priortiy*/

OSQCreate(&App_USBD_PHDC_ItemQ,
          "PHDC application item Q",
           APP_USBD_PHDC_ITEM_Q_NBR_MAX,
          &os_err);

if (os_err != OS_ERR_NONE) {
    APP_TRACE_DBG(("Could not create item Q w/err = %d\r\n", os_err));
    return (DEF_FAIL);
}
```

代码清单 8-7 创建消息队列

如果回看图 8-17(2)的注释,RxCommTask 任务负责所有数据接收传输,创建该任务的部分代码如下:

```
OSTaskCreate(       &App_USBD_PHDC_RxCommTaskTCB,          /* Create task that will handle rd procedures.
                     "USB Device PHDC rx comm",
                     App_USBD_PHDC_RxCommTask,
            (void *)App_USBD_PHDC_ClassNbr,
                     APP_CFG_USBD_PHDC_RX_COMM_TASK_PRIO,
                    &App_USBD_PHDC_RxCommTaskStk[0],
                     APP_CFG_USBD_PHDC_TASK_STK_SIZE / 10u,
                     APP_CFG_USBD_PHDC_TASK_STK_SIZE,
                     0u,
                     0u,
            (void *)0,
                     OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR,
                    &os_err);

if (os_err != OS_ERR_NONE) {
    APP_TRACE_DBG(("Could not add PHDC rx comm task w/err = %d\r\n", os_err));
    return (DEF_FAIL);
}
```

代码清单 8-8 创建 RxCommTask

最后,与创建消息队列类似的方式,初始化过程为每个 PHDC 优先级创建一个 TxCommTask 任务,如代码清单 8-9 所示:

```
OSTaskCreate(       &App_USBD_PHDC_TxCommTaskTCB,          /* Create task that will handle wr procedures.
                     "USB Device PHDC tx comm",
                     App_USBD_PHDC_TxCommTask,
             (void *)0,
                     APP_CFG_USBD_PHDC_TX_COMM_TASK_PRIO,
                    &App_USBD_PHDC_TxCommTaskStk[0],
                     APP_CFG_USBD_PHDC_TASK_STK_SIZE / 10u,
                     APP_CFG_USBD_PHDC_TASK_STK_SIZE,
                     0u,
                     0u,
             (void *)0,
                     OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR,
                    &os_err);

  if (os_err != OS_ERR_NONE) {
      APP_TRACE_DBG(("Could not add PHDC tx comm task w/err = %d\r\n", os_err));
      return (DEF_FAIL);
  }
  return (DEF_OK);
}
```

代码清单 8-9 创建 TxCommTask

8.6　分析 USB 通信

为了捕捉 USB 通信,需要按照图 2-8“Beagle USB 480 协议分析仪:连接”说明,连接 Beagle USB 480。

当监控 USB 通信时,建议在连接 μC/Eval-STM32F107 之前,连接 Beagle USB 480 的分析端,并启动捕捉。这意味着,在启动捕捉前,确保 μC/Eval-STM32F107 开发板 CN8 上的 USB 电缆没有连接 Beagle USB 480 的捕捉端。这使 Beagle USB 480 可以捕捉枚举通信阶段的描述符信息。

使用准备工作 2.1.4“Total Phase Data Center 软件”一节创建的快捷方式启动 Total Phase Data Center 软件,或到软件包解压目录下,运行 Data Center.exe 启动软件。

一旦软件启动,它将自动检测 Beagle 并连接。

图 8-17　Data Center 软件图标

可以查看其顶部的 LED,如果红色和绿色 LED 点亮,表明 Beagle USB 480 已正确连接。

按下 Ctrl+R 或单击工具栏的 Run Capture 按键,启动捕捉。

在表格中,将看到一些新的行,显示类似于 Capture Started 的信息。

此时,可以连接 μC/Eval-STM32F107 开发板 CN8 的 USB 电缆,以连接 PHDC 设备。

按照前面 8.4.3“安装 PHDC 设备项目”一节描述的步骤创完成 PHDC 设备的安装。然后,按照 8.4.4“测试新的 PHDC 设备”一节描述的指令发送默认项目

由于总线上已经有通信,它将在事务表格中实时显示通信信息。看到的第一个事务是枚举过程,对所有 USB 类是相同的过程,在附录 C“枚举过程”中描述。

8.6.1　获取配置描述符

获取配置描述符如图 8-18 所示,多数设备通常不超过一个配置,指定一个可用的配置,获取配置描述符包含接口描述符和端点描述符,进一步描述可用的接口和端点。本例中,仅有一个配置,包含一个接口和三个端点。

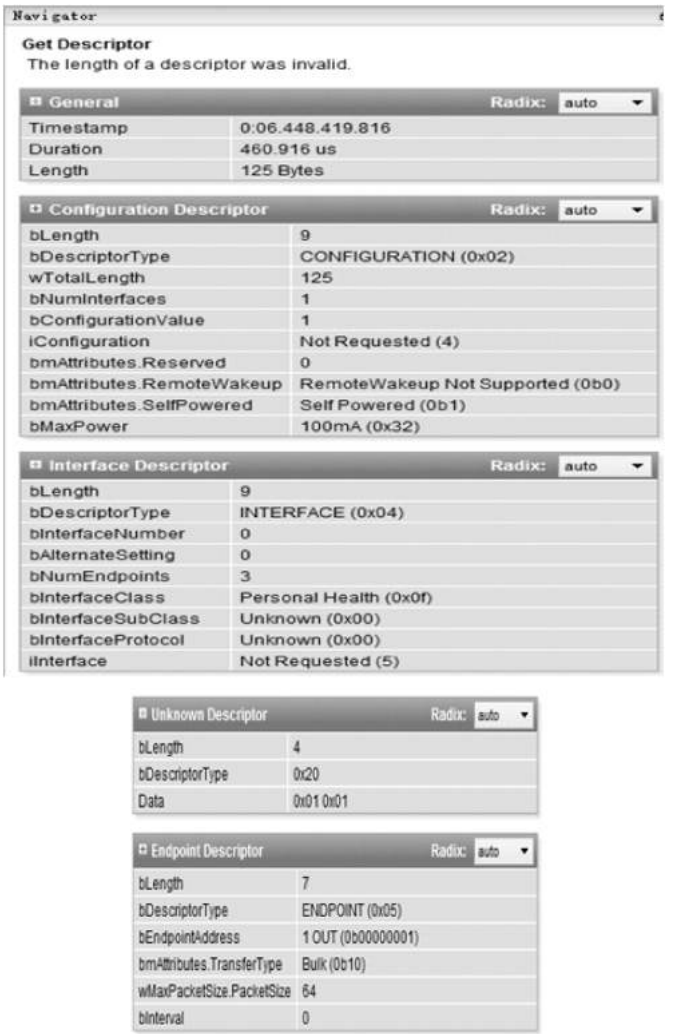


图 8-18　获取配置描述符捕获

Unknown Descriptor	Radix: auto
bLength	4
bDescriptorType	0x21
Data	0x01 0x38

Unknown Descriptor	Radix: auto
bLength	18
bDescriptorType	0x22
Data	0x4D 0x69 0x63 ...

Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	1 IN (0b10000001)
bmAttributes.TransferType	Bulk (0b10)
wMaxPacketSize.PacketSize	64
bInterval	0

Unknown Descriptor	Radix: auto
bLength	4
bDescriptorType	0x21
Data	0x01 0x3E

Unknown Descriptor	Radix: auto
bLength	18
bDescriptorType	0x22
Data	0x4D 0x69 0x63 ...

Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	2 IN (0b10000010)
bmAttributes.TransferType	Interrupt (0b11)
wMaxPacketSize.PacketSize	64
wMaxPacketSize.Transactions	One transaction per microframe if HS (0b00)
bInterval	10

Unknown Descriptor	Radix: auto
bLength	4
bDescriptorType	0x21
Data	0x01 0x01

Unknown Descriptor	Radix: auto
bLength	30
bDescriptorType	0x22
Data	0x4D 0x69 0x63 ...

图 8-18　获取配置描述符捕获(续)

注意:由于软件目前对 PHDC 类支持有限,一些描述符不能正确的解析。

标题为 unknown 的描述符是 PHDC QoS 和元数据描述符。

一旦主机发布一个设置配置请求,如附录 C"选择配置"所述,枚举过程结束,Data Center 软件提供一个枚举过程总结。通过打开导航器窗口,选择 Bus 标签下的枚举标签,打开枚举过程总结,如果 8-19 所示。

注意:代码清单 8-4"配置 QoS"展示的 App_USBD_PHDC_Init()部分配置了 3 个端点。

如果枚举过程成功,向下滚动事务表格,将找到一条命名为 control transfer 的记录,它在《嵌入式协议栈 μC/USB-Device》"控制传输"一节中介绍,如图 8-20 所示。

此时,继续在 PHDC 监控仪控制台应用窗口添加需要评估的 QoS 级别的新项目,注意,如果延迟配置为非常高,高,中等级别而非默认项目的低延迟,事务将使用

图 8-19　PHDC 枚举总结

Sp	Index	m:s.ms.us	Len	Err	Dev	Ep	Record
FS	440	0:45.298.406	0 B		25	00	Control Transfer
FS	441	0:45.298.406	8 B		25	00	SETUP txn
FS	442	0:45.298.406	3 B		25	00	SETUP packet
FS	443	0:45.298.409	11 B		25	00	DATA0 packet
FS	444	0:45.298.418	1 B		25	00	ACK packet
FS	445	0:45.298.427	0 B		25	00	IN txn [1 POLL]
FS	446	0:45.298.427	4.00 us		25	00	[1 IN-NAK]
FS	447	0:45.298.510	3 B		25	00	IN packet
FS	448	0:45.298.513	3 B		25	00	DATA1 packet
FS	449	0:45.298.516	1 B		25	00	ACK packet

图 8-20　PHDC 控制传输

批量端点,而非中断端点来实现。

图 8-21 展示了基于批量端点的事务,通常用于高于低延迟的 QoS 级别的项目。在添加周期为 1000ms,QoS 级别为高/最好(延迟/可靠性)的项目后捕捉。由于

同时有多个 QoS 运行,可以使用 Data Center 软件的过滤机制,仅显示基于批量端点 1 事务.

FS	4708	1:27.867.679	47 B	06	01	OUT txn	50 68 64 63 51 6F 53 53..	PhdcQoSSignature..	.opaque Data !
FS	4712	1:27.867.856	25 B	06	01	OUT txn	02 01 E8 03 10 44 61 74..	.....Data for High / Best	
FS	4777	1:27.776.147	47 B	06	01	IN txn [39244 POLL]	50 68 64 63 51 6F 53 53..	PhdcQoSSignature....	opaque Data !
FS	4782	1:28.866.654	8 B	06	01	IN txn	02 01 E8 03 10 44 61 74	.....Dat	
FS	4847	1:28.866.782	47 B	06	01	IN txn [35986 POLL]	50 68 64 63 51 6F 53 53..	PhdcQoSSignature....	opaque Data !
FS	4852	1:29.866.742	8 B	06	01	IN txn	02 01 E8 03 10 44 61 74	.....Dat	
FS	4917	1:29.866.863	47 B	06	01	IN txn [35984 POLL]	50 68 64 63 51 6F 53 53..	PhdcQoSSignature....	opaque Data !
FS	4923	1:30.866.697	8 B	06	01	IN txn	02 01 E8 03 10 44 61 74	.....Dat	
FS	4988	1:30.866.778	47 B	06	01	IN txn [35989 POLL]	50 68 64 63 51 6F 53 53..	PhdcQoSSignature....	opaque Data !
FS	4994	1:31.866.802	8 B	06	01	IN txn	02 01 E8 03 10 44 61 74	.....Dat	
FS	5059	1:31.866.941	47 B	06	01	IN txn [35984 POLL]	50 68 64 63 51 6F 53 53..	PhdcQoSSignature....	opaque Data !
FS	5064	1:32.866.759	8 B	06	01	IN txn	02 01 E8 03 10 44 61 74	.....Dat	
FS	5129	1:32.866.914	47 B	06	01	IN txn [35987 POLL]	50 68 64 63 51 6F 53 53..	PhdcQoSSignature....	opaque Data !
FS	5134	1:33.866.870	8 B	06	01	IN txn	02 01 E8 03 10 44 61 74	.....Dat	
FS	5199	1:33.867.034	47 B	06	01	IN txn [35983 POLL]	50 68 64 63 51 6F 53 53..	PhdcQoSSignature....	opaque Data !
FS	5204	1:34.866.842	8 B	06	01	IN txn	02 01 E8 03 10 44 61 74	.....Dat	
FS	5269	1:34.866.954	47 B	06	01	IN txn [35985 POLL]	50 68 64 63 51 6F 53 53..	PhdcQoSSignature....	opaque Data !
FS	5274	1:35.866.896	8 B	06	01	IN txn	02 01 E8 03 10 44 61 74	.....Dat	
FS	5339	1:35.867.012	47 B	06	01	IN txn [35986 POLL]	50 68 64 63 51 6F 53 53..	PhdcQoSSignature....	opaque Data !
FS	5344	1:36.866.875	8 B	06	01	IN txn	02 01 E8 03 10 44 61 74	.....Dat	

图 8-21　批量事务

8.7　总　结

本章描述的实例是为个人医疗设备增加 USB 连接的一个很好的参考,通过期望的医学信号从 PHDC 设备以不同的 QoS 传输的数据测试,评估 μC/USB-Device 协议栈和 PHDC 类。

本章还展示了如何运行该实例,并通过分析代码列表和 USB 通信捕获,解释了代码是如何工作的。

第 9 章

供应商类例程:批量同步/异步通信

μC/USB-Device 设备协议栈和其供应商类使设备之间的 USB 连接不必遵前面章节中描述的 USB 类。用户自定义类允许用户指定私有的协议,通过 USB 实施者论坛认证过的 USB 类维护自己的设备。供应商类在《嵌入式协议栈 μC/USB-Device》书第 12 章"供应商类"中介绍。本章介绍了一个应用实例。叙述了如何使用 IAR EWARM 和 Beagle USB 480 协议分析仪在 μC/Eval-STM32F107 上运行用户自定义类演示实例。

本书描述的实例不仅包含运行在 μC/Eval-STM32F107 上的嵌入式应用的源代码,还包含 Windows PC 上的一个 Visual Studio 项目软件,如图 9-1 所示。

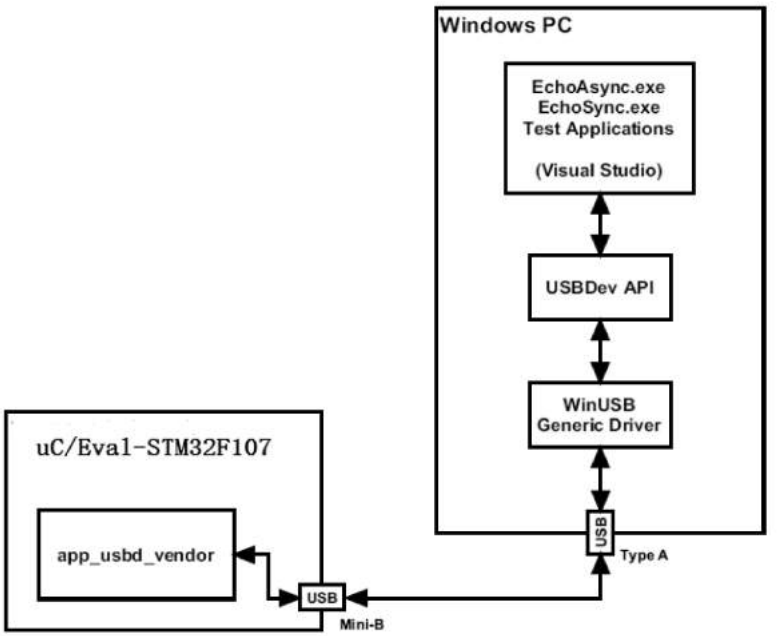


图 9-1　用户自定义类实例

实例应用演示了如何在 μC/Eval-STM32F107 和主机间来回传输数据,采用两种类型的通信:

- 同步通信:传输阻塞应用。
- 异步通信:传输不阻塞应用。

为了简洁,后面的内容仅描述了异步通信实例。然而,你可以尝试相同的实例,采用同步通信方式或同时使用这两个方式,可以通过使能/禁止相应的配置宏(见 9.2.1 一节)实现。如果使能两个接口,那你将创建一个复合设备。

9.1　在 IAR 里打开项目

打开 IAR EWARM 软件,导入本书配套软件包中的 C 语言工程,对所有的实例,仅需执行一次该操作 3.2.2 部分描述。

如果已经实现该过程,打开已选择路径下的工作区。

9.2　配置供应商类实例

从 Project Explorer(项目浏览器)中打开头文件 app_cfg.h,如图 9-2 所示。

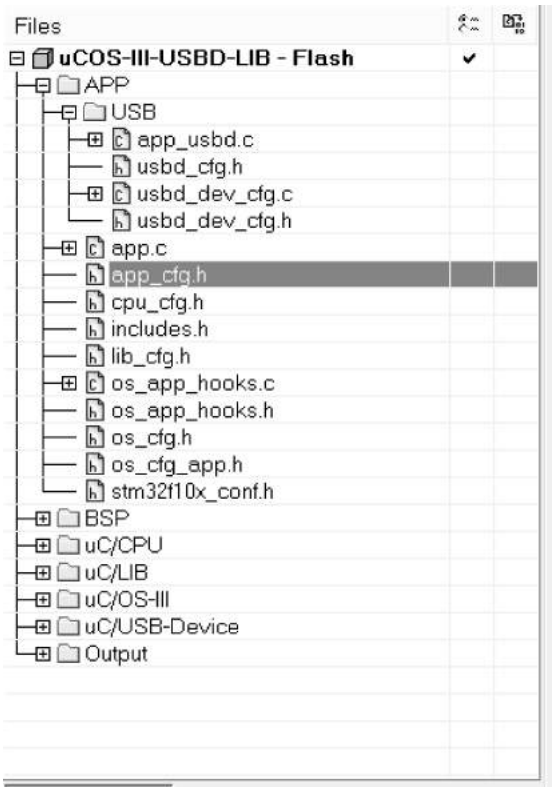


图 9-2　项目浏览器:配置文件

9.2.1　启用和禁用类

本书中介绍的所有实例,不仅位于同一工作空间,还位于同一项目中,为了使能用户自定义类和实例,app_cfg.h 中需要编辑的代码行如代码清单 9-1 所示:

```
/*
*********************************************************************************************************
*                                   uC/USB-DEVICE APPLICATION CONFIGURATION
*********************************************************************************************************
*/

#define  APP_CFG_USBD_CDC_EN                            DEF_DISABLED        (1)
#define  APP_CFG_USBD_HID_EN                            DEF_DISABLED        (2)
#define  APP_CFG_USBD_MSC_EN                            DEF_DISABLED        (3)
#define  APP_CFG_USBD_VENDOR_EN                         DEF_ENABLED         (4)
#define  APP_CFG_USBD_PHDC_EN                           DEF_DISABLED        (5)

#define  APP_CFG_USBD_VENDOR_ECHO_SYNC_EN               DEF_DISABLED        (6)
#define  APP_CFG_USBD_VENDOR_ECHO_ASYNC_EN              DEF_ENABLED
```

代码清单 9-1　配置用户自定义类实例:app_cfg.h

L9-1(1)设置 APP_CFG_USBD_CDC_EN 为 DEF_DISABLED 禁用 CDC 类。

L9-1(2)设置 APP_CFG_USBD_HID_EN 为 DEF_DISABLED 禁用 HID 类。。

L9-1(3)设置 APP_CFG_USBD_MSC_EN 为 DEF_DISABLED 禁用 MSC 类。

L9-1(4)为了运行用户自定义类实例,通过设置 APP_CFG_USBD_VENDOR_EN 为 DEF_ENABLED 使能供应商类。

L9-1(5)设置 APP_CFG_USBD_PHDC_EN 为 DEF_DISABLED 禁用 PHDC 类。

L9-1(6)选择通信类型:同步或异步。

9.3　构建供应商类实例项目

为了构建该项目,按下 F7 按键,如 3.2.4“在 EWARM 中构建项目”一节所述。

9.4　运行供应商类实例

为了运行该实例项目,需要遵循下列步骤,包括准备 INF 文件,Windows 需要使用它加载其本地驱动。

9.4.1　连接开发板

按照图 2-8 所示连接“Beagle USB 480 协议分析仪:连接”,连接开发板,不要连接 USB 电缆到 μC /Eval-STM32F107 开发板上的 CN8。

9.4.2　启动调试会话

按照 3.2.5“在 EWARM 中启动调试会话”一节所述,启动调试会话,按下 F5 使程序全速运行。

9.4.3　安装供应商指定设备

此时,μC/USB-Device 协议栈已经运行,并等待 USB 事件,如连接 USB 设备。用 USB 电缆连接 μC/Eval-STM32F107 开发板的 CN8 和 Beagle USB 480 的捕捉端,如图 2-8“Beagle USB 480 协议分析仪:连接”所示。如果没有 USB 协议分析仪,可以将电缆的另一端直接连接到主机任何可用的 USB 插座。

第一次运行该实例,主机将安装 μC/Eval-STM32F107 作为用户自定义设备。Win8 系统将显示一个 Widows 消息框,通知你设备正在安装,如图 9-3 所示。

图 9-3　Windows 安装用户指定的设备

如果是第一次安装该设备,Windows 将安装失败,因为这个特定的实例,虽然它使用了一个 Windows 本地驱动,它还需要一个 INF 文件来加载该驱动程序。

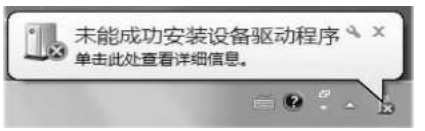


图 9-4　Windows 通知设备安装失败

INF 文件的内容在《嵌入式协议栈 μC/USB-Device》书 3.2 节“关于 INF 文件”中有简要的描述,本书配套软件包中包含该实例需要的 INF 文件。仅需要确保 INF 文件和用户指定的设备的硬件 ID 匹配。为此,可以使用一个文本编辑器如记事本,

打开下面的 INF 文件:

$\Micrium\Software\uC-USB-Device-V4\App\Host\OS\Windows\Vendor\INF\WinUSB_single.inf

```
; ========== Manufacturer/Models sections ===========

[Manufacturer]
%ProviderName% = MyDevice_WinUSB,NTx86,NTamd64,NTia64

[MyDevice_WinUSB.NTx86]
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_1003
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFE&PID_1003&MI_00

[MyDevice_WinUSB.NTamd64]
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_1003
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_1003&MI_00

[MyDevice_WinUSB.NTia64]
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_1003
%USB\MyDevice.DeviceDesc% =USB_Install, USB\VID_FFFF&PID_1003&MI_00

; ================== Installation ==================
```

图 9-5　更新 INF 文件

定位到指定供应商 ID=FFFFh 和产品 ID=1003h 的所在的行,如图 9-6 所示。确定他们与 usbd_dev_cfg.c 中配置的硬件 ID 匹配,如代码清单 9-2 所示:

```
/*
*********************************************************************************************
*                                        USB DEVICE CONFIGURATION
*********************************************************************************************
*/

USBD_DEV_CFG  USBD_DevCfg_STM32F_OTG = {
    0xFFFF,                                                 /* Vendor  ID.
#if (APP_CFG_USBD_VENDOR_EN == DEF_ENABLED)
    0x1003,                                                 /* Product ID (VENDOR).
#elif (APP_CFG_USBD_CDC_EN == DEF_ENABLED)
    0x1234,                                                 /* Product ID (CDC).
#elif (APP_CFG_USBD_HID_EN == DEF_ENABLED)
    0x1233,                                                 /* Product ID (HID).
#elif (APP_CFG_USBD_MSC_EN == DEF_ENABLED)
    0x1232,                                                 /* Product ID (MSC).
#elif (APP_CFG_USBD_PHDC_EN == DEF_ENABLED)
    0x0063,                                                 /* Product ID (PHDC).
#endif
    0x0100,                                                 /* Device release number.
   "OEM MANUFACTURER",                                      /* Manufacturer  string.
   "OEM PRODUCT",                                           /* Product       string.
   "1234567890ABCDEF",                                      /* Serial number string.
    USBD_LANG_ID_ENGLISH_US                                 /* String language ID.
};
```

代码清单 9-2　USB 用户指定设备配置

如果打开 Windows 设备管理器,将会找到一个新的安装失败的用户定义设备新表项。Windows 在其他设备组中列出该设备,如图 9-6 所示。

图 9-6　Windows 设备管理器

为了使用合适的驱动安装该设备,你需要给 Windows 提供更新过的 INF 文件,然后在设备管理器中,右击该设备名称,选择“更新驱动程序软件”,如图 9-6 所示。

Windows 将启动一系列的对话框来指定驱动选项,选择“浏览计算机以查找驱动程序”选项,如图 9-7 所示。

图 9-7　更新驱动软件

浏览到更新后的 INF 文件 WinUSB_single.inf 保存的路径,如图 9-8 所示:

图 9-8　更新驱动软件:INF 文件路径

由于该驱动没有数字签名,Windows 可能会给出一个警告信息。忽略该警告,选择“始终安装此驱动程序文件”,如图 9-9 所示。

图 9-9　更新驱动软件:警告信息

Windows 需要几秒钟的时间安装新的用户指定设备,并将 INF 文件复制到 $\Windows\inf 目录下。

安装结束时,Windows 将显示消息框,如图 9-11 所示。

现在,你将在设备管理器 USB Sample Class 组中看到一个新的表项,类似于图 9-12 所示。

图 9-10　更新驱动软件:安装

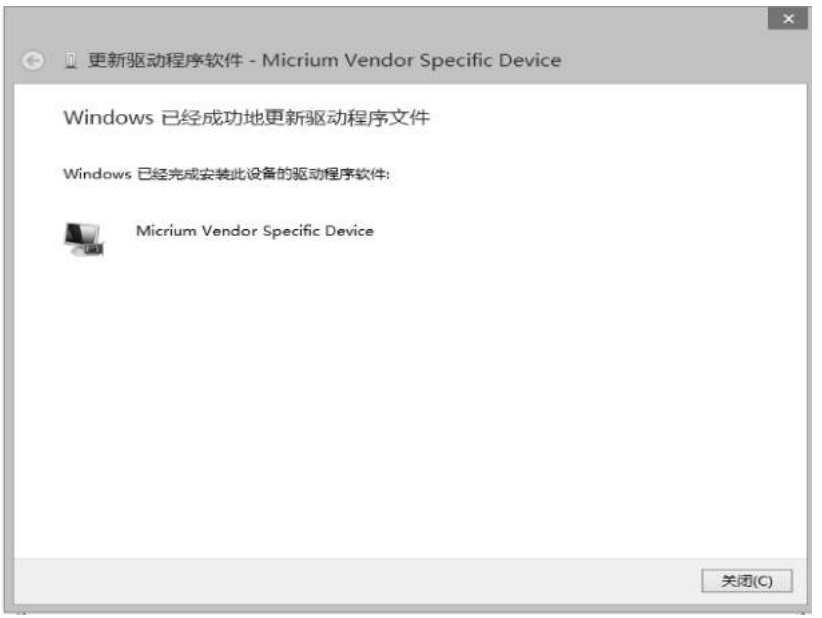


图 9-11　更新驱动软件:结束

9.4.4　测试新的供应商指定设备

现在,用户指定设备已经安装成功,可以使用本书配套软件中的 Echo Sync 或 Echo Async 控制台应用程序来验证 μC/USB-Device 协议栈和其用户自定义类。

为了执行相应的 Windows 应用程序:

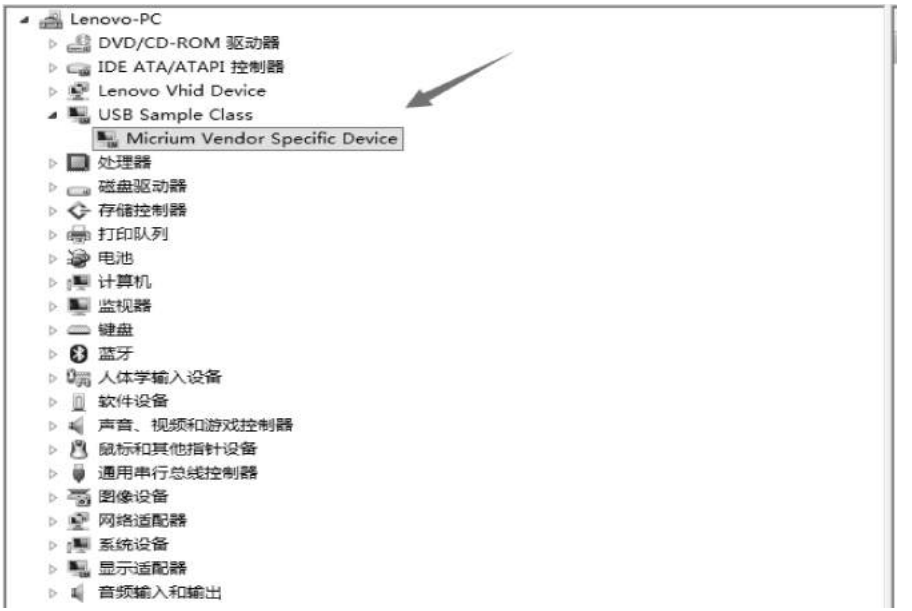


图 9-12　设备管理器显示新安装的设备

- EchoSync.exe:Echo Sync 实例,使用 WinUSB 同步模式运行。设备的同步通信 API 在《嵌入式协议栈 μC/USB-Device》12.2.4“同步通信”一节中描述。
- EchoAsync.exe:EchoAsync 实例,使用 WinUSB 异步通信模式运行。设备的异步通信 API 在《嵌入式协议栈 μC/USB-Device》12.2.5“异步通信”一节中描述。

这些 Windows 应用以一个 Visual Studio 项目的形式发布,你可以选择合适的文件执行,文件位于:

- Windows 64 位:

$\Micrium\Software\uC-USB-Device-V4\App\Host\OS\Windows\Vendor\Visual Studio 2010\exe\x64\

- Windows 32 位:

$\Micrium\Software\uC-USB-Device-V4\App\Host\OS\Windows\Vendor\Visual Studio 2010\exe\x86\

或者,根据特定配置(见《嵌入式协议栈 μC/USB-Device》书表 12-7“Windows 应用常量配置”),打开 Visual Studio 解决方案,重新构建可执行文件,文件位于:

$\Micrium\Software\uC-USB-Device-V4\App\Host\OS\Windows\Vendor\Visual Studio 2010\Vendor.sln

如图 9-13 所示,Echo Sync 或 Echo Async 是一个终端应用,能够让我们在 μC/

Eval-STM32F107 开发板上练习 μC/USB-Device 协议栈和用户自定义类应用。就像网络组件中的 Ping 功能一样,发送一个数据包并等待返回的应答。

图 9-13　用户自定义类终端应用

第一次执行 Echo Sync 或 Async 终端应用时,应用程序会提示需要测试传输的次数。下面的图片展示了 10 次传输的测试结果。

图 9-14　用户自定义类终端应用结果

下面的截图反应了测试中第三次事务,在终端中标签为“Sending/Receiving[3] bytes…OK”的事务。

F9-16(1)主机发送 2 个字节的报文头,其中包含后面报文负载的字节数(本例

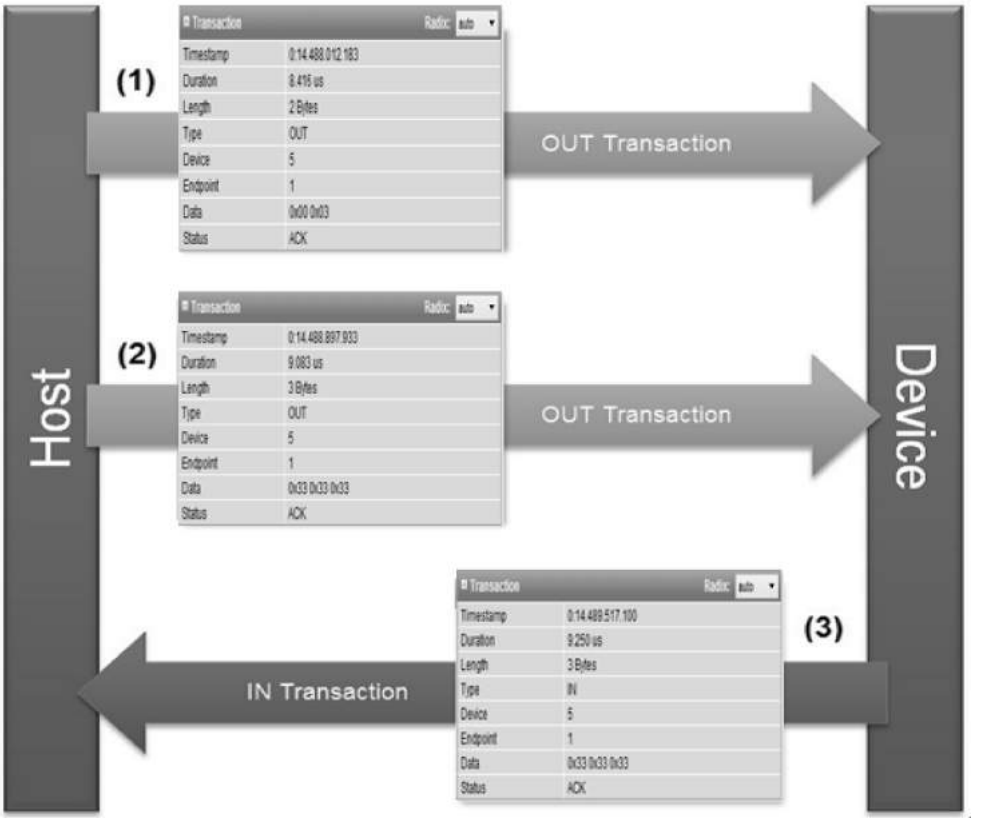


图 9-15　用户自定义类实例

中,由于它是第三组测试,负载长度为 3 个字节),开始第三次测试。

F9-16(2)然后,主机发送实现负载(本例中,是数字 3 的 ASCII 编码,3 次,0x33,0x33 和 0x33)。

F9-16(3)设备接收到负载,并回应相同的负载。

9.5　代码如何工作

9.5.1　同步通信

用户指定类同步通信实例通过 app_usbd_vendor.c 中的一个任务 EchoSyncTask 实现,如图 9-16 所示。

F9-16(1)本例的运行模型需要 μC/Eval-STM32F107 的一个批量输出端点和一个批量输入端点,如《嵌入式协议栈 μC/USB-Device》书中用户自定义类 12.1 一节"概述"所述。

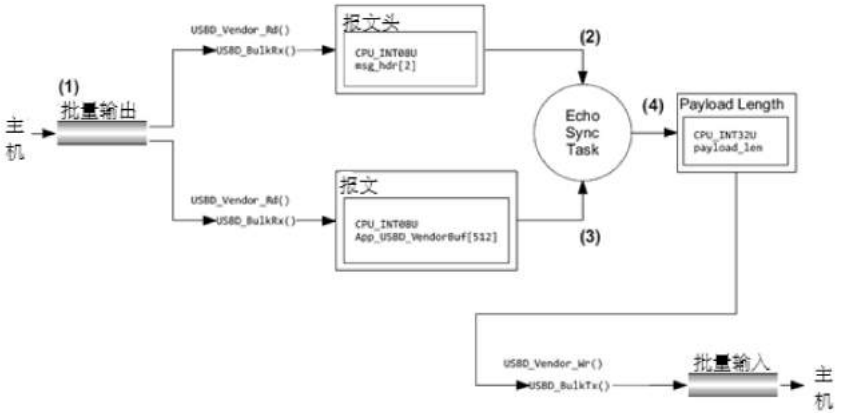


图 9-16　用户自定义类 Echo Sync

主机通过批量输出管道发送一个报文头启动传输。

F9-16(2) μC/OS-III 任务负责所有的同步接收和传输事务，通过 app_usbd_vendor.c 中的函数 App_USBD_Vendor_EchoSyncTask()来实现。

EchoSyncTask 任务通过调用函数 USBD_Vendor_Rd()读取 2 字节的报文头，以了解后面有效载荷的字节长度。

F9-16(3) 主机下次传输的是实际数据负载。主机发送的最大数据块为 512 字节。对每个数据块，主机都会等待带有相同块数据的应答。

同时，设备通过调用阻塞函数 App_USBD_Vendor_Rd()读取数据块。

F9-16(4) EchoSyncTask 通过调用函数 App_USBD_Vendor_Wr()回送相同的负载，然后返回 F9-16(3)所示的位置，继续发送下一个数据块，直到所有数据载荷发送完毕。

9.5.2　异步通信

用户指定类异步通信实例也是通过位于 app_usbd_vendor.c 中的任务 EchoAsyncTask 和其它的内核对象如信号量 EchoAsyncSem 来实现的，如图 9-17 所示。

F9-17(1) 本例的运行模式需要 μC/Eval-STM32F107 的一个批量输出端点和一个批量输入端点，如《嵌入式协议栈 μC/USB-Device》用户自定义类“概述”所述，主机通过批量输出管道发送一个报文头启动传输。

F9-17(2) EchoAsyncTask 任务通过调用非阻塞函数 USBD_Vendor_RdAsync()准备对报文头接收，该函数参数包含指向头缓冲区 App_USBD_Vendor_HeaderBuf[2]的指针，及回调函数 App_USBD_Vendor_RxHeaderCmpl()的名称，它将在传输完成时被设备协议栈调用。

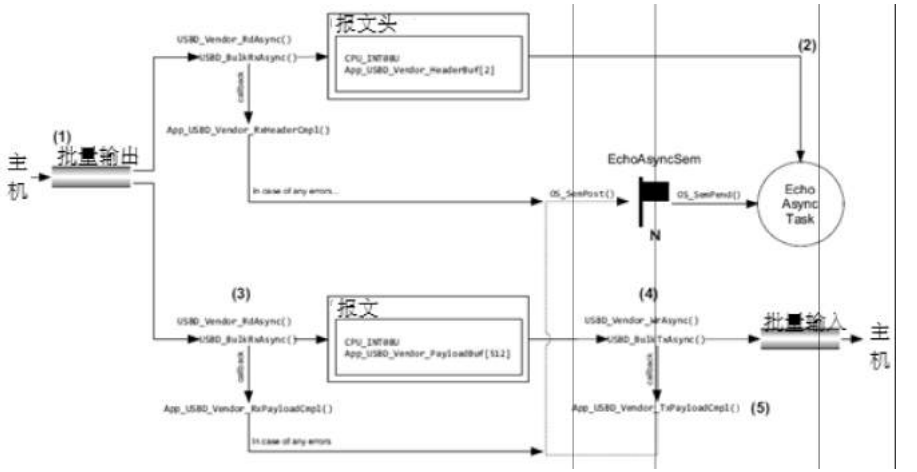


图 9-17　用户自定义类 Echo Async

F9-17(3)一旦报文头传输完成,设备协议栈将调用 App_USBD_Vendor_RxHeaderCmpl()回调函数,它不仅解析报文头以计算数据负载的长度,还调用非阻塞函数 USBD_Vendor_RdAsync()准备接收数据负载,该函数参数包含指向数据负载缓冲区 App_USBD_Vendor_PayloadBuf[512]的指针,以及回调函数 App_USBD_Vendor_RxPayloadCmpl()的名称。

F9-17(4)一旦数据负载传输或至少数据块传输完成,设备协议栈将调用回调函数 App_USBD_Vendor_RxPayloadCmpl(),它通过调用非阻塞函数 USBD_Vendor_WrAsync()准备数据负载(Echo)的传输,该函数参数包含一个指向数据块 buffer p_buf 的指针和回调函数 App_USBD_Vendor_TxPayloadCmpl()的名称。

F9-17(5)一旦数据负载或其中一块数据传输完成,设备协议栈将调用回调函数 App_USBD_Vendor_TxPayloadCmpl(),准备下一个数据块或一个新的报文头的接收。

每个回调函数返回的处理的结果将会被评估。如果有一个错误的信号发布给信号量 App_USBD_Vendor_EchoAsyncSem,它将触发 EchoAsyncTask 任务再次重新启动准备报文头接收的序列。

9.5.3　初始化

初始化用户指定类和 Echo Sync 及 Async 实例的函数在应用级文件 app_usbd_vendor.c 中声明。下面的代码列表描述了初始化过程:

代码清单 9-3 初始化用户特定实例

```
CPU_BOOLEAN  App_USBD_Vendor_Init (CPU_INT08U  dev_nbr,
                                   CPU_INT08U  cfg_hs,
                                   CPU_INT08U  cfg_fs)
{
    USBD_ERR    err;
    USBD_ERR    err_hs;
    USBD_ERR    err_fs;
    CPU_INT08U  class_nbr_0;
#if ((APP_CFG_USBD_VENDOR_ECHO_SYNC_EN  == DEF_ENABLED) || \
     (APP_CFG_USBD_VENDOR_ECHO_ASYNC_EN == DEF_ENABLED))
    OS_ERR      os_err;
#endif
#if ((APP_CFG_USBD_VENDOR_ECHO_SYNC_EN  == DEF_ENABLED) && \
     (APP_CFG_USBD_VENDOR_ECHO_ASYNC_EN == DEF_ENABLED))
    CPU_INT08U  class_nbr_1;
#endif


    err_hs = USBD_ERR_NONE;
    err_fs = USBD_ERR_NONE;

    APP_TRACE_DBG(("        Initializing Vendor class ... \r\n"));

    USBD_Vendor_Init(&err);                                     /* Init Vendor class.      */     (1)
    if (err != USBD_ERR_NONE) {
        APP_TRACE_DBG(("        ... could not initialize Vendor class w/err = %d\r\n\r\n", err));
        return (DEF_FAIL);
    }
                                                               /* Create a Vendor class instance.
    class_nbr_0 = USBD_Vendor_Add(DEF_FALSE, 0u, App_USBD_Vendor_VendorReq, &err);                (2)
    if (err != USBD_ERR_NONE) {
        APP_TRACE_DBG(("        ... could not instantiate a Vendor class w/err = %d\r\n\r\n", err));
        return (DEF_FAIL);
    }
```

L9 - 3(1)USBD_Vendor_Init()函数在 usbd_vendor.c 中声明,用来初始化类需要的所有内部结构和变量。关于该函数的更多信息,参阅《嵌入式协议栈 μC/USB-Device》供应商类 API 参考手册 G.1.1"USBD_Vendor_Init()"一节。

L9 - 3(2)USBD_Vendor_Add()函数也在 usbd_vendor.c 中声明,它负责创建一个新的供应商类实例。如果创建时没有任何错误,参数返回用于标识的类号,这里,还有机会注册一个回调函数,以实现任何供应商指定的处理,例如解析任何私有的请求。

初始化过程的下一步是添加新的供应商类实例到全速设备配置中,如代码清单 9 - 4 所示。

```
if (cfg_fs != USBD_CFG_NBR_NONE) {
                                                           /* Add vendor class to FS dflt cfg.                 */
    USBD_Vendor_CfgAdd(class_nbr_1, dev_nbr, cfg_fs, &err_fs);
    if (err_fs != USBD_ERR_NONE) {
        APP_TRACE_DBG(("        ... could not add Vendor class instance #%d to FS configuration w/err = %d\r\n\r\n", class_nbr_1, err_fs));
    }
}


 if (err_fs != USBD_ERR_NONE) {            /* If HS and FS cfg fail, stop class init.            */
    return (DEF_FAIL);
}
```

代码清单 9-4 添加供应商类实例到全速配置

初始化过程最后一系列步骤是真正的应用初始化,它包含创建所有必需的内核对象,如任务和 9.5.1“同步通信”及 9.5.2“异步通信”中所述的信号量。

9.6　分析 USB 通信

为了捕捉 USB 通信,需要按照图 2-8“Beagle USB 480 协议分析仪:连接”说明,连接 Beagle USB 480。

当监控 USB 通信时,建议在连接 uC/Eval-STM32F107 之前,连接 Beagle USB 480 的分析端,并启动捕捉。这意味着,在启动捕捉前,确保 μC/Eval-STM32F107 开发板 CN8 上的 USB 电缆没有连接 Beagle USB 480 的捕捉端。这使 Beagle USB 480 可以捕捉枚举通信阶段的描述符信息。

使用准备工作 2.1.3“Total Phase Data Center 软件”一节创建的快捷方式启动 Total Phase Data Center 软件,或到软件包解压目录下,运行 Data Center.exe 启动软件。

一旦软件启动,它将自动检测 Beagle 并连接。

可以查看其顶部的 LED,如果红色和绿色 LED 点亮,表明 Beagle USB 480 已正确连接。

按下 Ctrl+R 或单击工具栏的 Run Capture 按键,启动捕捉。

在表格中,将看到一些新的行,显示类似于 Capture Started 的信息。

图 9-18　Data Center 软件图标

此时,可以连接 μC/Eval-STM32F107 开发板 CN8 的 USB 电缆,以连接用户指定设备。

按照 9.4.3“安装供应商指定设备”一节描述的步骤完成用户指定设备的安装。

由于总线上已经有通信,它将在事务表格中实时显示通信信息。看到的第一个事务是枚举过程,对所有 USB 类是相同的过程,在附录 C“枚举过程”中描述。

9.6.1　获取配置描述符

获取配置描述符如图 9-19 所示,指定一个可用的配置,大多数设备通常只有一个配置。获取配置描述符包含接口描述符和端点描述符,用来进一步的描述可用的接口和端点。本例中,仅有一个配置,一个接口和两个端点。

Data Center 软件提供了枚举过程概述。

通过打开向导窗口,选择 Bus 标签下的 Enumeration 标签,查看枚举概述,如图 9-20 所示。

注意,配置中包含两个批量输入和输出端点,其所需要的操作类型在图 9-16

Navigator

Get Descriptor

General	Radix: auto
Timestamp	0:03.851.509.766
Duration	237.833 us
Length	32 Bytes

Configuration Descriptor	Radix: auto
bLength	9
bDescriptorType	CONFIGURATION (0x02)
wTotalLength	32
bNumInterfaces	1
bConfigurationValue	1
iConfiguration	Not Requested (4)
bmAttributes.Reserved	0
bmAttributes.RemoteWakeup	RemoteWakeup Not Supported (0b0)
bmAttributes.SelfPowered	Self Powered (0b1)
bMaxPower	100mA (0x32)

Interface Descriptor	Radix: auto
bLength	9
bDescriptorType	INTERFACE (0x04)
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	2
bInterfaceClass	Vendor Specific (0xff)
bInterfaceSubClass	Unknown (0xff)
bInterfaceProtocol	Unknown (0xff)
iInterface	Not Requested (5)

Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	1 OUT (0b00000001)
bmAttributes.TransferType	Bulk (0b10)
wMaxPacketSize.PacketSize	64
bInterval	0

Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	1 IN (0b10000001)
bmAttributes.TransferType	Bulk (0b10)
wMaxPacketSize.PacketSize	64
bInterval	0

图 9-19　获取配置描述符

“用户自定义类 Echo Sync”和图 9-17“用户自定义类 Echo Async”这两步中描述。

通过 9.4.4“测试新的供应商指定设备”一节描述的其中一个终端应用继续发送

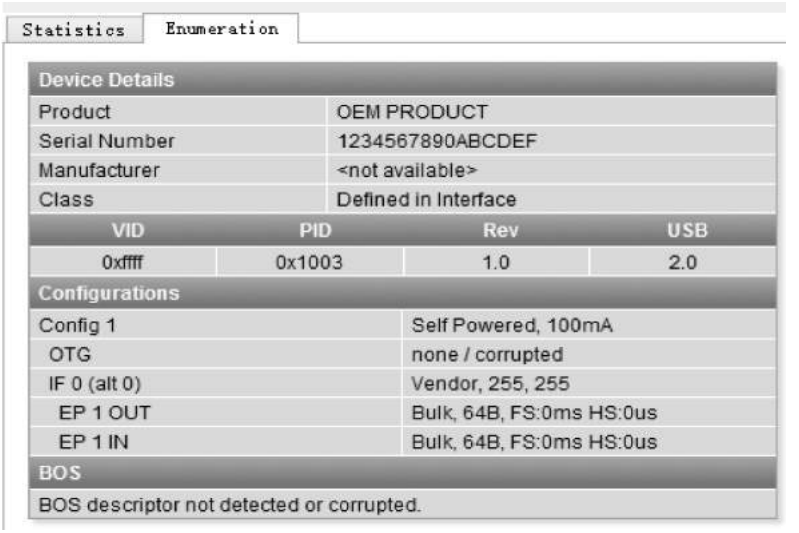


图 9-20　用户自定义枚举概述

一些 echo 请求。

如果配置终端应用发送 10 个 echo 请求,你需要捕捉 10 次事务集。每个事务集包含报文头和主机发出的有效负载及设备发出有效负载的事务,如图 9-21 所示。

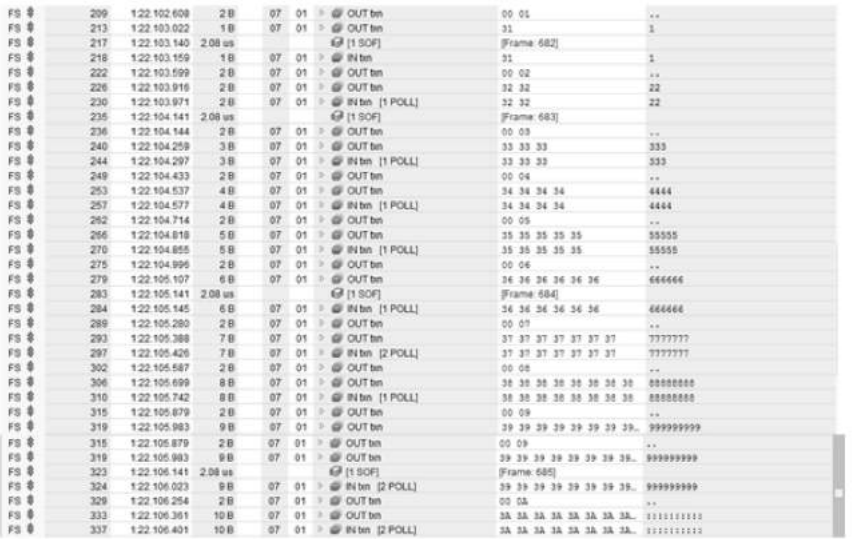


图 9-21　Echo 事务

9.7　总结

本章描述的实例是为任何用户指定设备添加 USB 连接的一个很好的参考。通过测试批量端点的数据输入和输出传输,可以评估 μC/USB-Device 协议栈和其用户自定义类的性能。

本章还展示了如何运行该实例,并通过一系列的流程图,分析代码列表和 USB 通信捕获,解释了代码是如何工作。通过描述用户指定设备的一些重要的 USB 通信捕获,展示了 Total Phase Data Center 软件的更多特性。

附录 A

μC/Probe 介绍

本附录简要介绍 μC/Probe。μC/Probe 是一个 Windows 应用软件，用于在运行时读写嵌入式目标处理器的内存。内存位置映射到控制面板上的一系列的虚拟控制键和指示器。图 A－1 展示了系统概览和数据流向。

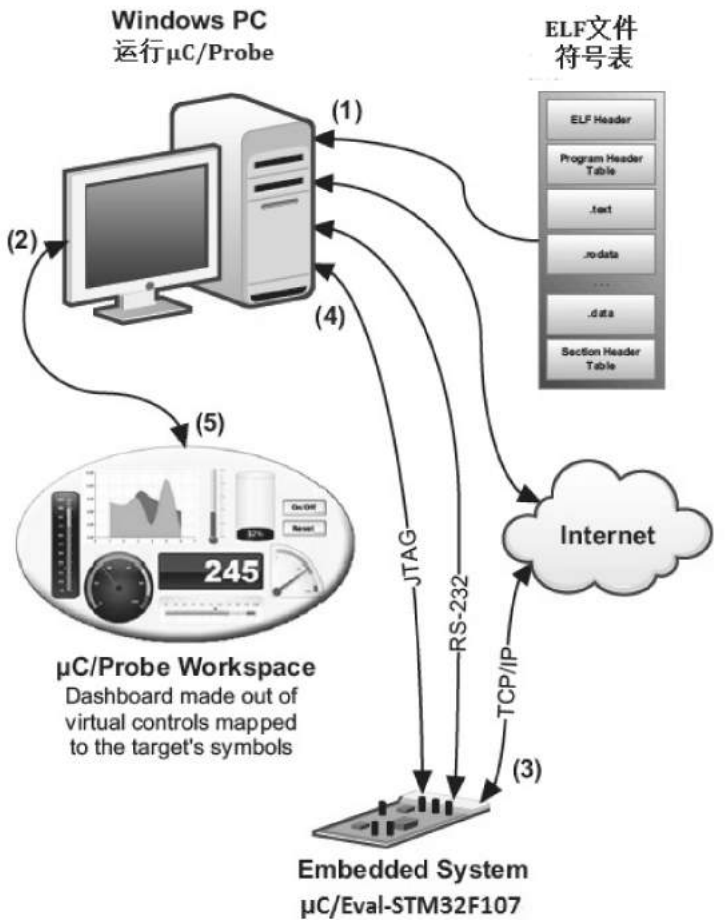


图 A－1　μC/Probe 数据流图

FA-1(1)需要给 μC/Probe 一个带调试信息的 ELF 文件。ELF 文件通过 IAR EWARM 集成开发环境的链接器生成。μC/Probe 解析 ELF 文件,读取每个嵌入式目标代码的符号(例如全局变量)地址,并创建一个称之为 symbol browser 的目录,在设计时,用户使用该目录选择在控制面板上显示的符号。通过 IAR EWARM 构建 ELF 文件的更多信息,参见 3.2.4"在 IAR EWARM 中构建项目"。

FA-1(2)设计时,在 Windows PC 上使用 μC/Probe 创建一个工作区。通过拖放虚拟控制键和指示器到 data screen 区,创建自己的控制面板。每个虚拟控制键和指示器需要映射到一个嵌入式目标代码的符号,可以通过 symbol browser 符号浏览器选择该符号。

FA-1(3)运行之前,需要配置 μC/Probe 使用的通信接口,有多种通信接口:J-Link,RS232,TCP/IP,USB。本书介绍的实例采用了 J-Link 接口,更多接口配置的信息,参见 2.1.5"μC/Probe"一节。需要单击 Run 按键,启动 μC/Probe,μC/Probe 将请求读取每个虚拟控制键和指示器(例如按键和仪表)对应的内存位置的值。同时,μC/Probe 发送命令写虚拟控制键(例如单击事件按键)对应的内存位置。

FA-1(4)对于读请求,嵌入式目标系统回应变量最新的值。对写请求,嵌入式目标系统回应一个确认应答信号。由于本书的例程采用了 J-Link 接口,不需要在 μC/Eval-STM32F107 开发板上驻留 μC/Probe 固件。其它类型的接口如 RS-232,TCP/IP 或 USB,需要在开发板上驻留 Micrium 提供的目标代码。

FA-1(5) μC/Probe 解析嵌入式目标板的应答信息,更新虚拟控制键和指示器的显示。

A.1　进一步学习

Micriμm 网页上的 μC/Probe 页面提供了教学视频和官方文档,请访问:
http://micrium.com/tools/ucprobe/

- 下载 μC/Probe 目标手册,了解驻留在嵌入式系统上的固件的更多信息。
- 下载 μC/Probe 用户手册了解系统 Windows PC 端的更多使用信息。

附录 B

STM32F107 USB 设备驱动程序

μC/USB-Device 协议栈可以运行在各种 USB 设备控制器上，它具有一个硬件抽象层，可以将协议栈移植到任何 USB 设备控制器上。市场上有很多可用的 USB 设备控制器，每个控制器都需要相应的设备驱动程序，配合 μC/USB-Device 协议栈一起工作。Micriμm 提供了很多 USB 设备控制器的驱动程序，并定期添加新的驱动程序。

本章叙述了 STM32F107 MCU 的 μC/USB-Device 设备驱动程序，该驱动程序以预编译链接对象的形式包含在本书配套的软件中。如果需要访问驱动程序的源代码，需要从 Micriμm 获得授权（见附录 D“μC/OS-III 和 μC/USB-Device 许可政策”）。

本章首先介绍了 STM32F107 USB 设备控制器的一些主要特性，然后解释了 μC/Eval-STM32F107 开发板 USB 接口相关的一些细节，最后描述了驱动程序代码。

Micriμm 提供了 USB 设备驱动 API 和数据类型的命名约定，通过遵循这些命名约定，及 Micriμm 的编码规范和软件开发风格，可以简化设备驱动程序的调试和测试，帮助开发人员熟悉其它人写的设备驱动程序。

由于 μC/USB-Device 支持相同类型的多个接口，所以开发一个可重入的驱动程序非常重要。避免使用全局宏和变量（例如，使用设备数据域，在设备的.c 文件中定义宏），驱动开发人员需要确保包含多个设备驱动文件的项目可以编译。

《嵌入式协议栈 μC/USB-Device》书中第 6 章“设备驱动程序指南”提供了一个 USB 设备控制器驱动结构介绍，Micriμm 也提供了一个 μC/USB-Device 驱动模板，该文件位于目录：

$\Micrium\Software\uC-USB-Device-V4\Drivers\Template

STM32F107 的 μC/USB-Device 设备驱动程序通过了 USB 2.0 规范的一致性认证。USB 实施者论坛（USB-IF）在 www.usb.org 上提供了一个软件工具 USB Command Verifier，来评估 USB 设备与 USB 设备框架（《嵌入式协议栈 μC/USB-Device》第 9 章）、HID 类、MSC 类和 PHDC 类等的一致性。

B.1 STM32F107 片上 USB OTG 控制器

STM32F107 片上 USB OTG 控制器可以作为全速主机或设备控制器使用。该

模块内置 USB 收发器，支持《嵌入式协议栈 μC/USB-Device》1.8.3“传输”一节描述的所有传输类型。

该模块作为一个 USB 设备控制器一些重要特性如下：

- 支持 USB 2.0 全速传输(12Mbps)；
- 集成 USB 2.0 收发器(PHY)；
- 支持四种传输类型：控制、批量、中断和同步传输；
- 7 个端点。
  - 一个双向控制端点
  - 3 个输入端点
  - 3 个输出端点
- USB 通信专用的缓存

B.2　μC/Eval-STM32F107 评估板

μC/Eval-STM32F107 评估板包含了两个 USB 接口，一个 USB-OTG 接口(CN8)和一个 USB 供电接口(CN5)。

μC/Eval-STM32F107 评估板上 USB-OTG 接口支持全速(12Mbps)通信，可以配置作为主机，设备或 OTG 使用。

设备模式不需要使用任何外部供电电路实现，USB 电路连接到 μC/Eval-STM32F107 评估板上的连接器 CN8 上。

后续内容解释了 STM32F107 集成的 USB 设备控制器的驱动实现，该驱动基于 $\Micrium\Software\uC-USB-Device-V4\Drivers\Template 目录下的驱动程序模板。

B.3　设备驱动约定

所有的 USB 设备驱动文件命名为 usbd_drv_<controller>.c 和.h 文件，<controller>标识 USB 设备控制器的名称。按照该约定，本书使用的驱动文件为 usbd_drv_stm32f_fs.c 和 usbd_drv_stm32f_fs.h，位于图 B-1 所示的目录下：

由于 ST 公司的 STM32F105、STM32F107 系列微控制器使用了相同的 USB 控制器，μC/USB-Device 的 STM32_FS 驱动设计为一个通用驱动，针对不同的开发板带有相应的 BSP 抽象层，如图 B-1 所示。

FB-1(1)文件 usbd_drv_stm32f_fs.c 和 usbd_drv_ stm32f_fs.h 实现了驱动 API，它们将在 B.4“设备驱动 API“一节详细描述。

FB-1(2)文件 usbd_bsp_stm32fxxx.c 和 usbd_bsp_ stm32fxxx.h 包含 BSP 抽象层代码实现，如时钟，中断控制器和 GPIO 等。针对 STM32F107，

这些文件不仅定义了 μC/Eval-STM32F107 开发板上 USB 设备控制器的初始化、连接和断开函数，还定义了端点信息表。

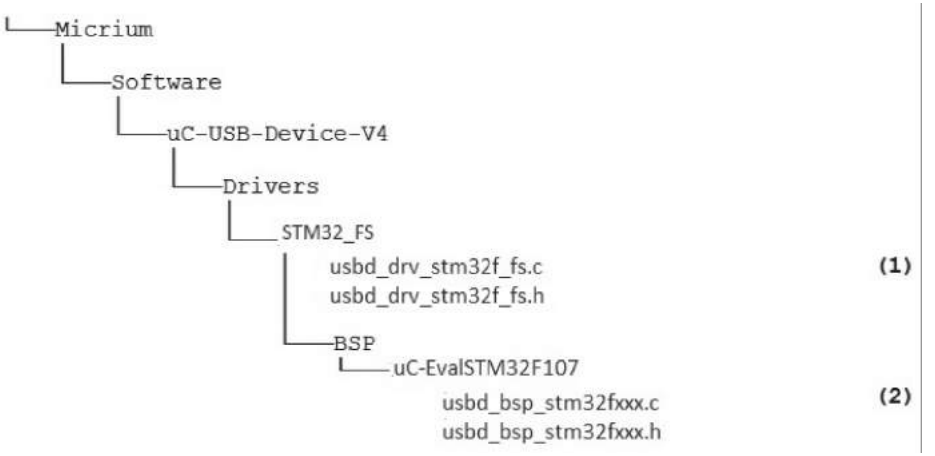


图 B－1　μC/Eval-STM32F107 的 μC/USB-Device 驱动源代码

B.3.1　端点变量名称

端点是 USB 通信的基本单位，在《嵌入式协议栈 μC/USB-Device》1.8.1“端点”一节描述，在源代码中，它们通过下列值标识：

- 端点地址：ep_addr；
- 端点物理号：ep_phy_nbr；
- 端点逻辑号：ep_log_nbr。

这些变量在 μC/USB-Device 设备协议栈源代码中广泛使用，STM32F107 的 USB 驱动采用了相同的命名约定。

图 B－2 说明了每个端点变量名称的含义。

B.3.2　端点信息表

如《嵌入式协议栈 μC/USB-Device》1.8.1“端点”一节所述，每个 USB 设备控制器设计了一组端点，作为数据源端或目的地。STM32F107 的 USB 设备控制器有 7 个端点，包括默认端点 0-EP0，这些端点的能力在《嵌入式协议栈 μC/USB-Device》6.5“设备配置”一节叙述的端点信息表中描述，针对 STM32F107 USB 驱动，端点信息表在 usbd_bsp_stm32fxxx.c 中声明，如代码清单 B－1 所示。

代码清单 B-1 STM32F107 的端点信息表

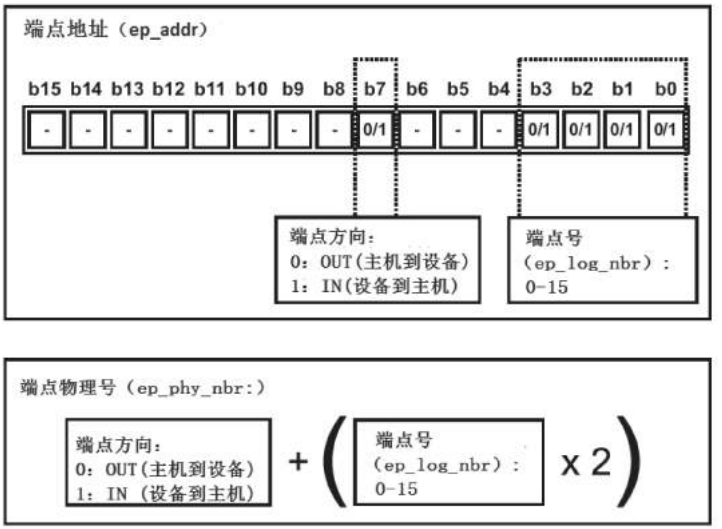


图 B-2　端点变量命名

```
USBD_DRV_EP_INFO  USBD_DrvEP_InfoTbl_STM32F105_7xx[] = {
    {USBD_EP_INFO_TYPE_CTRL | USBD_EP_INFO_DIR_OUT, 0u,   64u},
    {USBD_EP_INFO_TYPE_CTRL | USBD_EP_INFO_DIR_IN,  0u,   64u},
    {USBD_EP_INFO_TYPE_CTRL | USBD_EP_INFO_TYPE_ISOC |
     USBD_EP_INFO_TYPE_BULK | USBD_EP_INFO_TYPE_INTR | USBD_EP_INFO_DIR_OUT, 1u,   64u},
    {USBD_EP_INFO_TYPE_CTRL | USBD_EP_INFO_TYPE_ISOC |
     USBD_EP_INFO_TYPE_BULK | USBD_EP_INFO_TYPE_INTR | USBD_EP_INFO_DIR_IN,  1u,   64u},
    {USBD_EP_INFO_TYPE_CTRL | USBD_EP_INFO_TYPE_ISOC |
     USBD_EP_INFO_TYPE_BULK | USBD_EP_INFO_TYPE_INTR | USBD_EP_INFO_DIR_OUT, 2u,   64u},
    {USBD_EP_INFO_TYPE_CTRL | USBD_EP_INFO_TYPE_ISOC |
     USBD_EP_INFO_TYPE_BULK | USBD_EP_INFO_TYPE_INTR | USBD_EP_INFO_DIR_IN,  2u,   64u},
    {USBD_EP_INFO_TYPE_CTRL | USBD_EP_INFO_TYPE_ISOC |
     USBD_EP_INFO_TYPE_BULK | USBD_EP_INFO_TYPE_INTR | USBD_EP_INFO_DIR_OUT, 3u,   64u},
    {USBD_EP_INFO_TYPE_CTRL | USBD_EP_INFO_TYPE_ISOC |
     USBD_EP_INFO_TYPE_BULK | USBD_EP_INFO_TYPE_INTR | USBD_EP_INFO_DIR_IN,  3u,   64u},
    {DEF_BIT_NONE
                                                              ,   0u,    0u}
};
```

B.4　设备驱动 API

如《嵌入式协议栈 μC/USB-Device》第 6 章“设备驱动程序指南”中所述，所有的设备驱动必须在源代码中将相应的设备驱动 API 结构实例声明为一个全局变量。API 结构是一个有序的函数指针列表，当需要设备硬件服务时，μC/USB-Device 设备协议栈通过相应的指针调用函数。Usbd_drv_stm32f_fs.c 中声明的 API 结构如代码清单 B-2 所示。

```
/*
*********************************************************************************************************
*                                      USB DEVICE CONTROLLER DRIVER API
*********************************************************************************************************
*/

USBD_DRV_API  USBD_DrvAPI_STM32F_FS = { USBD_DrvInit,
                                        USBD_DrvStart,
                                        USBD_DrvStop,
                                        USBD_DrvAddrSet,
                                        DEF_NULL,
                                        DEF_NULL,
                                        DEF_NULL,
                                        USBD_DrvFrameNbrGet,
                                        USBD_DrvEP_Open,
                                        USBD_DrvEP_Close,
                                        USBD_DrvEP_RxStart,
                                        USBD_DrvEP_Rx,
                                        USBD_DrvEP_RxZLP,
                                        USBD_DrvEP_Tx,
                                        USBD_DrvEP_TxStart,
                                        USBD_DrvEP_TxZLP,
                                        USBD_DrvEP_Abort,
                                        USBD_DrvEP_Stall,
                                        USBD_DrvISR_Handler,
                                      };
```

代码清单 B－2 STM32F107 μC/USB-Device 设备驱动 API

每个驱动 API 函数接受一个指向 USBD_DRV 类型结构的指针作为它的一个参数，通过该结构，可以访问下列字段：


```
typedef  const  struct  usbd_drv_cfg {
    CPU_ADDR          BaseAddr;    /* Base address of device controller hardware regs.  */ (1)
    CPU_ADDR          MemAddr;     /* Base address of device controller dedicated mem.  */ (2)
    CPU_ADDR          MemSize;     /* Size         of device controller dedicated mem.  */ (3)
    USBD_DEV_SPD      Spd;         /* Speed        of device controller.                */
    USBD_DRV_EP_INFO *EP_InfoTbl;  /* Device controller EP information table.           */ (4)
} USBD_DRV_CFG;


typedef usb_drv {
    CPU_INT08U        DevNbr;                                                             (5)
    USBD_DRV_API     *API_Ptr;     /* Device controller API.                            */ (6)
    USBD_DRV_CFG     *CfgPtr;      /* Device controller configuration.                  */ (7)
    void             *DataPtr;     /* Device controller local data.                     */ (8)
    USBD_DRV_BSP_API *BSP_API_Ptr; /* Device controller board specific API.             */ (9)
} USBD_DRV;
```

代码清单 B－3 USB 设备驱动数据结构

LB－3(1) STM32F107 USB 设备硬件寄存器的基地址。

LB－3(2)由于 STM32F107 USB 控制器没有专用的内存，该项没有使用。

LB－3(3)由于 STM32F107 USB 控制器没有专用的内存，该项没有使用。

LB－3(4) STM32F107 端点信息表。

LB－3(5)识别设备的唯一索引号。

LB－3(6)指向 USB 设备控制器驱动 API 的指针。

LB－3(7)指向 USB 设备控制器驱动配置的指针。

LB-3(8)指向 USB 设备控制器驱动特定数据的指针。

LB-3(9)指向 μC/Eval-STM32F107 USB 设备控制器 BSP 的指针。

后面会注意到,在做任何工作之前,几乎每个驱动 API 将通过该结构获取指向硬件寄存器和 BSP API 的指针。

每个设备驱动 API 函数的描述参见《嵌入式协议栈 μC/USB-Device》书中的附录 B"设备控制器驱动 API 参考手册"。

如果已经购买了 μC/USB-Device 设备协议栈的授权,那你可以访问驱动程序的源代码,否则,下面章节提供的 STM32F107 的 USB 控制器的驱动代码以 C 注释形式呈现。

B.4.1　USBD_DrvInit()

```
static  void  USBD_DrvInit (USBD_DRV  *p_drv,
                            USBD_ERR  *p_err)
{
                              /* Alloc drv internal data.                              */
                              /* Store drv internal data ptr.                          */
                              /* Get driver BSP API reference.                         */
                              /* Get USB ctrl reg ref.                                 */
                              /* Call board/chip specific device controller ...        */
                              /* ... initialization function.                          */
                              /* Disable the global interrupt                          */
                              /* ---------------------- PHY CFG ---------------------- */

                              /* --------------------- CORE RESET --------------------- */

                              /* ------------------- DEVICE INIT -------------------- */
                              /* Reset the PHY clock                                   */
                              /* Flush all transmit FIFOs                              */
                              /* Wait for the flush completion                         */
                              /* Flush the receive FIFO                                */
                              /* Wait for the flush completion                         */
                              /* Clear all pending Device interrupts                   */
                              /* Dis. interrupts for the Device IN Endpoints           */
                              /* Dis. interrupts for the Device OUT Endpoints          */
                              /* Dis. interrupts foo all Device Endpoints              */

                              /* ----------------- IN ENDPOINT RESET ---------------- */
                              /* Clear any pending Interrupt                           */
                              /* ---------------- OUT ENDPOINT RESET ---------------- */
                              /* Clear any pending Interrupt                           */
                             /* Disable all interrupts                                */
                              /* Set Device Address to zero                            */
}
```

代码清单 B-4 STM32F107 USB 设备驱动 USBD_DrvInit()

B.4.2　USB_DrvStart()

```
static  void  USBD_DrvStart (USBD_DRV  *p_drv,
                             USBD_ERR  *p_err)
{

                             /* Get driver BSP API reference.                    */
                             /* Get USB ctrl reg ref.                            */

                             /* Clear any pending interrupt                      */
                             /* Enable interrupts                                */
                             /* Enable Global Interrupt                          */
                             /* Call board/chip specific connect function.       */
                             /* Generate Device connect event to the USB host    */
}
```

代码清单 B-5 STM32F107 USB 设备驱动 USBD_DrvStart()

B.4.3　USBD_DrvStop()

```
static  void  USBD_DrvStop (USBD_DRV  *p_drv)
{
                             /* Get driver BSP API reference.                    */
                             /* Get USB ctrl reg ref.                            */

                             /* Disable all interrupts                           */
                             /* Clear any pending interrupt                      */
                             /* Disable the global interrupt                     */
                             /* Generate Device Disconnect event to the USB host */
}
```

代码清单 B-6 STM32F107 USB 设备驱动 USBD_DrvStop()

B.4.4　USBD_DrvAddrSet()

```
static  CPU_BOOLEAN  USBD_DrvAddrSet (USBD_DRV    *p_drv,
                                      CPU_INT08U   dev_addr)
{
                            /* Set Device Address                                */

}
```

代码清单 B-7 STM32F107 USB 设备驱动 USBD_DrvAddrSet()

B. 4. 5　USBD_DrvGetFrameNbr()

```
static  CPU_INT16U  USBD_DrvFrameNbrGet (USBD_DRV  *p_drv)
{
                                /* Get Frame number                                     */
}
```

代码清单 B-8 STM32F107 USB 设备驱动 USBD_DrvGetFrameNbr()

B. 4. 6　USBD_DrvEP_Open()

```
static  void  USBD_DrvEP_Open (USBD_DRV    *p_drv,
                               CPU_INT08U   ep_addr,
                               CPU_INT08U   ep_type,
                               CPU_INT16U   max_pkt_size,
                               CPU_INT08U   transaction_frame,
                               USBD_ERR    *p_err)
{
                               /* En. Device EP0 IN interrupt                          */
                               /* En. common IN EP Transfer complete interrupt         */
                               /* En. EP0 OUT interrupt                                */
                               /* En. common OUT EP Setup $ Xfer complete interrupt    */
                               /* cfg EP max packet size                               */
                               /* cfg EP type                                          */
                               /* Tx FIFO number                                       */
                               /* EP start data toggle                                 */
                               /* USB active EP                                        */

                               /* En. Device EPx IN Interrupt                          */
                               /* cfg EP max packet size                               */
                               /* cfg EP type                                          */
                               /* EP start data toggle                                 */
                               /* USB active EP                                        */
                               /* En. Device EPx OUT Interrupt                         */
}
```

代码清单 B-9 STM32F107 USB 设备驱动 USBD_DrvEP_Open()

B. 4. 7　USBD_DrvEP_Close()

```
static  void  USBD_DrvEP_Close (USBD_DRV    *p_drv,
                                CPU_INT08U   ep_addr)
{
                            /* ------------------- IN ENDPOINTS ------------------- */
                            /* Deactive the Endpoint                                */

                            /* ------------------ OUT ENDPOINTS ------------------- */
                            /* Deactive the Endpoint                                */

                            /* Dis. EP interrupt                                    */
}
```

代码清单 B-10 STM32F107 USB 设备驱动 USBD_DrvEP_Close()

B.4.8　USBD_DrvEP_RxStart()

```
static  CPU_INT32U  USBD_DrvEP_RxStart (USBD_DRV    *p_drv,
                                        CPU_INT08U   ep_addr,
                                        CPU_INT08U  *p_buf,
                                        CPU_INT32U   buf_len,
                                        USBD_ERR    *p_err)
{
                            /* Read Control EP reg                              */
                            /* Read Transer EP reg                              */
                            /* Clear EP transfer size and packet count          */

                            /* Set transfer size to max pkt size                */
                            /* Set packet count                                 */

                            /* Set packet count                                 */
                            /* Set transfer size                                */

                            /* Clear EP NAK and Enable EP for receiving         */
}
```

代码清单 B-11 STM32F107 USB 设备驱动 USBD_DrvEP_RxStart()

B.4.9　USBD_DrvEP_Rx()

```
static  CPU_INT32U  USBD_DrvEP_Rx (USBD_DRV    *p_drv,
                                   CPU_INT08U   ep_addr,
                                   CPU_INT08U  *p_buf,
                                   CPU_INT32U   buf_len,
                                   USBD_ERR    *p_err)
{
                        /* Receive data from endpoint                   */
}
```

代码清单 B-12 STM32F107 USB 设备驱动 USBD_DrvEP_Rx()

B.4.10　USBD_DrvEP_RxZLP()

```
static  void  USBD_DrvEP_RxZLP (USBD_DRV    *p_drv,
                                CPU_INT08U   ep_addr,
                                USBD_ERR    *p_err)
{
                    /* Receive zero-length packet from endpoint             */
}
```

代码清单 B-13 STM32F107 USB 设备驱动 USBD_DrvEP_RxZLP()

B.4.11　USBD_DrvEP_Tx()

```
static  CPU_INT32U  USBD_DrvEP_Tx (USBD_DRV    *p_drv,
                                   CPU_INT08U   ep_addr,
                                   CPU_INT08U  *p_buf,
                                   CPU_INT32U   buf_len,
                                   USBD_ERR    *p_err)
{
                                   /* transmit data to  endpoint            */
}
```

代码清单 B-14 STM32F107 USB 设备驱动 USBD_DrvEP_Tx()

B. 4. 12　USBD_DrvEP_TxStart()

```
static  void  USBD_DrvEP_TxStart (USBD_DRV    *p_drv,
                                  CPU_INT08U   ep_addr,
                                  CPU_INT08U  *p_buf,
                                  CPU_INT32U   buf_len,
                                  USBD_ERR    *p_err)
{
                                /* Read EP control reg.                          */
                                /* Read EP transfer reg                          */

                                /* Clear EP transfer size                        */
                                /* Transfer size                                 */
                                /* Packet count                                  */

                                /* Clear EP NAK mode & Enable EP transmitting.   */
                                /* Write to Tx FIFO of associated EP             */
}
```

代码清单 B-15 STM32F107 USB 设备驱动 USBD_DrvEP_TxStart()

B. 4. 13　USBD_DrvEP_TxZLP()

```
static  void  USBD_DrvEP_TxZLP (USBD_DRV    *p_drv,
                                CPU_INT08U   ep_addr,
                                USBD_ERR    *p_err)
{
                            /* transmit zero-length packet from endpoint         */
}
```

代码清单 B-16 STM32F107 USB 设备驱动 USBD_DrvEP_TxZLP()

B. 4. 14　USBD_DrvEP_Abort()

```
static  CPU_BOOLEAN  USBD_DrvEP_Abort (USBD_DRV    *p_drv,
                                       CPU_INT08U   ep_addr)
{
                            /* ------------------- IN ENDPOINTS ------------------- */
                            /* Set Endpoint to NAK mode                          */
                            /* Wait for IN EP NAK effective                      */
                            /* Wait for EP disable                               */
                            /* Clear EP Disable interrupt                        */

                            /* Flush EPx TX FIFO                                 */
                            /* Wait for the flush completion                     */
                            /* ------------------ OUT ENDPOINTS ------------------ */

                            /* Flush EPx RX FIFO                                 */

                            /* Wait for the flush completion                     */
}
```

代码清单 B-17 STM32F107 USB 设备驱动 USBD_DrvEP_Abort()

B. 4. 15　USBD_DrvEP_Stall()

```
static  CPU_BOOLEAN  USBD_DrvEP_Stall (USBD_DRV     *p_drv,
                                       CPU_INT08U    ep_addr,
                                       CPU_BOOLEAN   state)
{
                        /* ------------------- IN ENDPOINTS -------------------- */
                        /* Disable Endpoint if enable for transmit              */
                        /* Set Stall bit                                        */
                        /* Clear Stall Bit                                      */
                        /* Set DATA0 PID                                        */
                        /* ------------------- OUT ENDPOINTS ------------------- */
                        /* Set Stall bit                                        */
                        /* Clear Stall Bit                                      */
                        /* Set DATA0 PID                                        */
}
```

代码清单 B-18 STM32F107 USB 设备驱动 USBD_DrvEP_Stall()

B. 4. 16　USBD_DrvISR_Handler()

ISR 代码首先读取中断状态寄存器,处理每个中断。

```
static  void  USBD_DrvISR_Handler (USBD_DRV  *p_drv)
{
                        /* Read global interrrupt status register               */
                        /* ------------ RX FIFO NON-EMPTY DETECTION ------------ */
                        /* --------------- IN ENDPOINT INTERRUPT --------------- */
                        /* -------------- OUT ENDPOINT INTERRUPT --------------- */
                        /* ---------------- USB RESET DETECTION ---------------- */
                        /* Notify bus reset event.                              */
                        /* Enable Global OUT/IN EP interrupt...                 */
                        /* ...and RX FIFO non-empty interrupt.                  */
                        /* Clear USB bus reset interrupt                        */
                        /* Set Rx FIFO depth                                    */
                        /* Set EP0 to EP3 Tx FIFO depth                         */
                        /* ------------ ENUMERATION DONE DETECTION ------------ */
                        /* Clear Enumeration done interrupt                     */
                        /* ------------------ MODE MISMATCH -------------------- */
                        /* -------------- SESSION REQ DETECTION ---------------- */
                        /* Notify connect event.                                */
                        /* Clear all OTG interrupt sources.                     */
                        /* ------------------ OTG INTERRUPT -------------------- */
                        /* Notify disconnect event.                             */
                        /* Clear all OTG interrupt sources.                     */
                        /* ------------- EARLY SUSPEND DETECTION --------------- */
                        /* Clear the Early Suspend interrupt                    */
                        /* --------------- USB SUSPEND DETECTION --------------- */
                        /* Notify Suspend Event                                 */
                        /* Clear USB Suspend interrupt                          */
                        /* ----------------- WAKE-UP DETECTION ----------------- */
                        /* Notify Resume Event                                  */
                        /* Clear Remote wakeup signaling                        */
                        /* Clear Remote wakeup interrupt                        */

}
```

代码清单 B-19 STM32F107 USB 设备驱动 USBD_DrvISR_Handler()

附录 C

枚举过程

如《嵌入式协议栈 μC/USB-Device》书中 1.12“枚举”一节所述，主机通过枚举过程了解设备的能力，并配置设备。本附录按照 Beagle USB 480 获取的 USB 捕获信息叙述了枚举过程。

C.1 捕捉 USB 通信

为了捕捉 USB 通信，需要按照图 2-8“Beagle USB 480 协议分析仪：连接”中的说明，连接 Beagle USB 480 与开发板。当监控 USB 通信时，最好在连接 μC/Eval-STM32F107 评估板之前，先连接 Beagle USB 480 的分析端，并启动捕捉。这意味着，在启动捕捉前，确保 μC/Eval-STM32F107 开发板 CN8 接口上的 USB 电缆没有连接 Beagle USB 480 的捕捉端。这使 Beagle USB 480 可以捕捉枚举通信阶段的描述符信息。使用 2.1.3“Total Phase Data Center 软件”一节创建的快捷方式启动 Total Phase Data Center 软件，或到软件包解压目录下，运行 Data Center.exe 启动软件。

一旦应用软件启动，它将自动检测 Beagle USB 480 并连接。

图 C-1 Data Center 软件图标

可以查看 Beagle USB 480 顶部的 LED，如果红色和绿色 LED 点亮，表明 Beagle USB 480 已正确连接。

按下 Ctrl+R 或单击工具栏的 Run Capture 按键，启动数据捕捉。

在 Data Center 软件界面中，将看到一些新的行，显示类似于 Capture Started 的信息。

此时，可以连接 μC/Eval-STM32F107 发板 CN9 的 USB 电缆，以连接设备。

参考表 C-1，按照相应的实例章节描述的步骤完成 USB 设备的安装。

表 C-1 设备安装参考手册

USB 类	例程	章节
CDC ACM	USB 转串口	5.4.4“安装 CDC 设备”

续表 C-1

USB 类	例程	章节
HID	鼠标	6.4.4“安装 HID 设备”
MSC	便携式存储设备	7.4.4“安装 MSC 设备”
PHDC	通信监控	8.4.4“安装 PHDC 设备”
Vendor	批量同步/异步通信	9.4.4“安装供应商自定义设备”

由于总线上已经有通信，它将在事务表格(transactions grid)中实时显示通信信息。你将看到的一组标准请求，这些请求对所有 USB 类都是相同的；和另一组请求，不同类之间这些请求类似，取决于枚举的 USB 类，其序号，大小和内容会不同。这些请求在图 C-2 中显示。

Sp	Index	m:s.ms.us	Len	Dev	Ep	Record	Summary
	0	0:00.000.000				Capture started (Aggregate)	[09/06/12 11:42:00]
	1	0:00.000.000				<Host connected>	
	6	0:01.856.094				<Host disconnected>	
	7	0:01.856.297				<Host connected>	
	8	0:01.869.088	733 ns			<Reset> / <Chirp K> / <Tiny K>	
	9	0:01.869.089	302 us			<Reset> / <Target disconnected>	
	10	0:01.869.392	4.13 us			<Reset> / <Chirp J> / <Tiny J>	
FS	11	0:01.869.396				<Full-speed>	
FS	12	0:01.984.326	11.0 ms			<Reset> / <Target disconnected>	
FS	13	0:01.995.327	5.20 us			<Reset> / <Chirp J> / <Tiny J>	
FS	29	0:02.247.039				<Full-speed>	
FS	30	0:02.560.246	11.0 ms			<Reset> / <Chirp J> / <Tiny J>	
FS	31	0:02.571.247				<Full-speed>	
FS	32	0:02.601.242	18 B	00	00	Get Device Descriptor	Index=0 Length=64
FS	63	0:02.602.116	11.0 ms			<Reset> / <Chirp J> / <Tiny J>	
FS	64	0:02.613.118				<Full-speed>	
FS	65	0:02.735.409	0 B	00	00	Set Address	Address=08
FS	74	0:02.762.142	18 B	08	00	Get Device Descriptor	Index=0 Length=18
FS	101	0:02.786.403	125 B	08	00	Get Configuration Descriptor	Index=0 Length=255
FS	133	0:02.787.737	34 B	08	00	Get String Descriptor	Index=3 Length=255
FS	160	0:02.789.131	4 B	08	00	Get String Descriptor	Index=0 Length=255
FS	187	0:02.790.147	24 B	08	00	Get String Descriptor	Index=2 Length=255
FS	214	0:02.792.676	18 B	08	00	Get Device Descriptor	Index=0 Length=18
FS	241	0:02.793.653	9 B	08	00	Get Configuration Descriptor	Index=0 Length=9
FS	268	0:02.794.629	125 B	08	00	Get Configuration Descriptor	Index=0 Length=125
FS	300	0:02.805.085	2 B	08	00	Get Device Status	
FS	327	0:02.805.776	0 B	08	00	Set Configuration	Configuration=1

图 C-2　枚举过程

首先执行端口复位操作，通过<Reset>标签标识，使 USB 设备进入端口使能和连接状态，该状态下，USB 设备可以响应发给默认设备地址 0 的所有请求。后面的章节将描述后续请求。

C.1.1　获取设备描述符

主机发送的第一个请求，通过一条标题为“Get Device Descriptor”的记录描述，如图 C-3 所示。

该请求发到设备 0，该行信息反应了 μC/Eval-STM32F107 描述整个 USB 设备信息给主机的时刻，信息包括支持的 USB 版本，控制端点的最大包尺寸，供应商 ID，产品 ID 和设备可用的配置数(大多数信息在 usbd_dev_cfg. c 中声明)。这是枚举过程的第一步：

Sp	Index	m:s.ms.us	Len	Dev	Ep	Record	Summary
FS	30	0:10.698.918	18 B	00	00	Get Device Descriptor	Index=0 Length=64

Navigator

Get Descriptor

General	Radix: auto
Timestamp	0:10.698.918.083
Duration	191.750 us
Length	18 Bytes

Device Descriptor	Radix: auto
bLength	18
bDescriptorType	DEVICE (0x01)
bcdUSB	2.00 (0x0200)
bDeviceClass	Defined in Interface (0x00)
bDeviceSubClass	Defined in Interface (0x00)
bDeviceProtocol	Defined in Interface (0x00)
bMaxPacketSize0	64
idVendor	0xfffe
idProduct	0x0824
bcdDevice	1.00 (0x0100)
iManufacturer	Not Requested (1)
iProduct	Not Requested (2)
iSerialNumber	Not Requested (3)
bNumConfigurations	1

Bus　LiveFilter　Info

图 C-3　获取设备描述符捕获

如果设备描述符请求成功，所有后续的控制事务将使用图 C-3 显示的设备描述符指定的 64 个字节的最大包长度传输。

执行第二次端口复位为下次枚举状态做准备。

C.1.2　设置地址

枚举过程的下一步是主机控制器为设备分配一个唯一的地址，在 Get Device Descriptor 和第二次端口复位记录后，将会找到一条 Set Address 记录，如图 C-4 所示，它反映了主机分配唯一的地址 8 给 USB 设备的时刻。

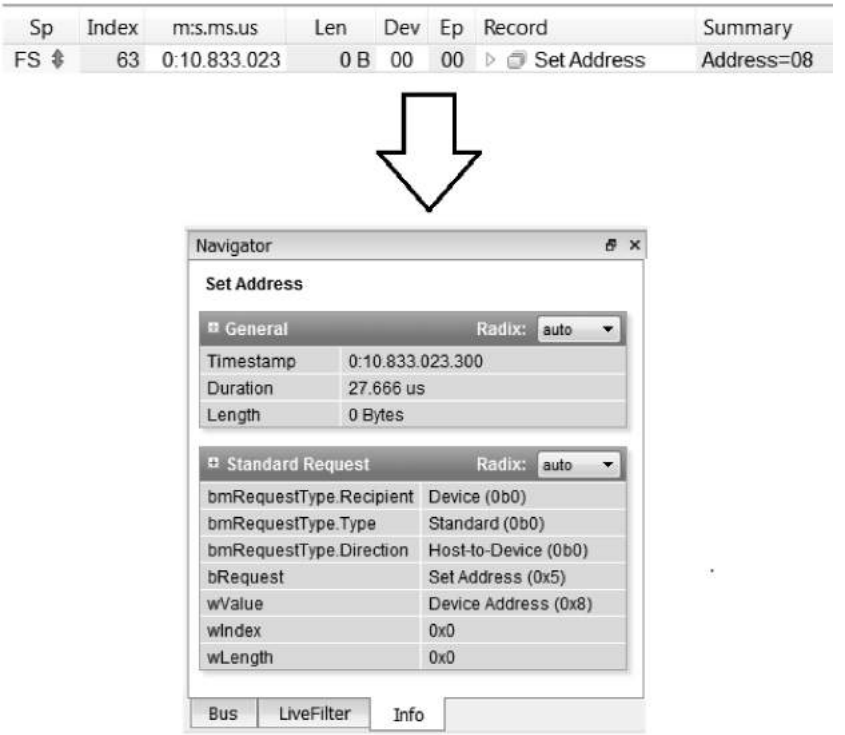


图 C-4　设置地址捕获

注意，该记录之前的所有事务使用默认的设备地址 0（参见列标题 Dev 的内容）传输给设备，此后的所有事务将使用设备地址 8。

为了方便，在图 C-4 底部显示的向导（Navigator）窗口的信息（Info）面板中，提供了数据包的解析视图。

如果设备地址分配成功，在发布一系列获取配置描述符（Get Configuration Descriptor）和获取字符串描述符（Get String Descriptor）请求之前，将再次发送设备描述符请求到新的设备地址，以了解设备的能力。

C.1.3　获取字符串描述符

标题为 Get String Descriptor 的记录反映了设备提供的可选的用户可读信息的时刻，信息包含产品名称，生产商和序列号等，这些信息作为设备配置的一部分，在 usbd_dev_cfg.c 中设置。

Get String Descriptor 在图 C-5 中展示，获取的描述符中还包含一个语言 ID，以允许支持多种语言：

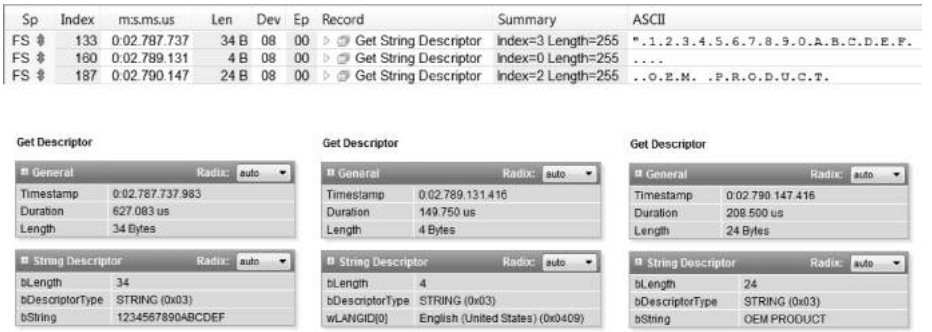


图 C-5　获取字符串描述符

C.1.4　获取配置描述符

一个 USB 设备可以实现多种不同的配置，配置描述符仅仅是接口描述符的一部分。因此，获取配置描述符(Get Configuration Descriptor)的捕获并不相同，它取决于 USB 类和应用配置。关于不同例程的 Get Configuration Descriptor 捕获，参阅表 C-2，查看相应的实例章节。

表 C-2 不同 USB 类获取的配置描述符

USB 类	例程	章节
CDC ACM	USB 转串口	5.6.1“获取配置描述符”
HID	鼠标	6.6.1“获取配置描述符”
MSC	便携式存储设备	7.6.1“获取配置描述符”
PHDC	通信监控	8.6.1“获取配置描述符”
Vendor	批量同步/异步通信	9.6.1“获取配置描述符”

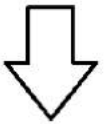
注意，对于 USB 通信，Data Center 软件可以解析一些标准的 USB 类，如 HID，CDC ACM 和 MSC，因此，可以方便的显示相应的事务。然而，对 PHDC 和供应商(Vendor)类的支持有限，仅有 USB 协议相关的原始数据被解析。

C.1.5　选择配置

主机选择了其中一个有效的配置后，枚举过程结束。标题为 Set Configuration 的记录反映了该过程，其记录如图 C-6 所示。

Set Configuration 请求使 USB 设备可以通过所选的配置，从指定的端点开始通信。

Sp	Index	m:s.ms.us	Len	Dev	Ep	Record	Summary
FS	380	0:10.903.366	0 B	08	00	Set Configuration	Configuration=1



Navigator

Set Configuration

General	Radix: auto
Timestamp	0:10.903.366.133
Duration	177.583 us
Length	0 Bytes

Standard Request	Radix: auto
bmRequestType.Recipient	Device (0b0)
bmRequestType.Type	Standard (0b0)
bmRequestType.Direction	Host-to-Device (0b0)
bRequest	Set Configuration (0x9)
wValue	Configuration Value (0x1)
wIndex	0x0
wLength	0x0

Bus　LiveFilter　Info

图 C－6　选择配置(Set Configuration)捕获

附录 D

μC/OS-III 和 μC/USB-Device 许可政策

μC/OS-III 以源码方式提供免费的短期评估，可用于教学目的或者用于和平目的的研究。为了方便用户的使用、帮助用户体验 μC/OS-III，我们提供了全部的源代码。但事实上，提供源代码并不意味着用户可以在不支付许可费用的情况下，将其用于商业目的。源代码所提供的知识也不能被用于开发类似的产品。本书还包含了 μC/USB-Device 的预编译链接库形式的库文件。

用户可以从北京麦克泰软件技术有限公司单独购买 μC/Eval-STM32F107 评估板，并在 μC/Eval-STM32F107 评估板上使用 μC/OS-III 和 μC/USB-Device。针对教育目的应用，在初期用户不需要购买其他东西，不过一旦代码被用于盈利性的商业项目或产品，你需要去购买许可。

当决定在产品设计中使用 μC/OS-III 和 μC/USB-Device 时，即需要购买授权，而不是当产品即将量产是才购买。

如果用户不确定是否需要为自己的程序购买软件使用许可证，请联系 Micriμm 公司并与销售代表讨论计划中的用途。

D.1 μC/USB-Device 维护更新

μC/USB-Device 授权提供了一年的技术支持，维护和源代码更新服务。续签维护协议以获取连续的支持和源代码更新，更多信息请联系 sales@Micrium.com。

D.2 μC/USB-Device 源码更新

如果在维护期内，当源码有更新时，用户将自动获取 email 通知，然后，用户可以通过 Micrium 用户入口下载代码更新，如果超出服务期，或忘记 Micrium 用户名和密码，请联系 sales@Micrium.com。

D.3 μC/USB-Device 支持

授权用户可以获取技术支持，请访问 www.Micrium.com 网页的用户支持界面

(customer support)。如果您还不是正式用户,请注册创建一个自己的账号,系统将提供一个 web 形式的页面,以帮助您提供需要支持的问题。

授权用户还可以采用下列方式联系我们:

联系 Micriμm 公司

Micriμm

1290 Weston Road,Suite 306

Weston,FL 33326

+1 954 217 2036

+1 954 217 2037(传真)

邮件:sales@Micrium.com

网址:www.Micrium.com

附录 E

参考文献

[1] Labrosse Jean J. μC/OS-III, The Real-Time Kernel for the Renesas RX62N. Micriμm Press,2010, ISBN 978-0-9823375-7-8.

[2] Renesas Electronics. RX63N Group User's Manual: Hardware. Rev. 0.9. December 1, 2011. Available at: www. renesas. com/products/mpumcu/rx/rx600/Documentation. jsp.

[3] Total Phase, Inc. Beagle Protocol Analyzers Data Sheet v4. 20. January 31, 2012. Available at:www. totalphase. com.

[4] Total Phase, Inc. Data Center Software Manual v6. 20. July 12, 2012. Available at:www. totalphase. com.

[5] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips. Universal Serial BusSpecification Revision 2. 0. April 27, 2000. Available at: www. usb. org/developers/docs.

[6] USB Implementers Forum, Inc. Universal Serial Bus, Class Definitions for-Communications Devices, Revision 1. 2. November 3, 2010. Available at: www. usb. org/developers/devclass_docs.

[7] USB Implementers Forum, Inc. Universal Serial Bus, Communications, Subclass for PSTN Devices, Revision 1. 2. February 9, 2007. Available at: www. usb. org/developers/devclass_docs.

[8] USB Implementers Forum, Inc. Device Class Definition for Human Interface Devices(HID), Version 1. 11. June 27, 2001. Available at: www. usb. org/developers/devclass_docs.

[9] USB Implementers Forum, Inc. Universal Serial Bus HID Usage Tables, Version 1. 12. October 28, 2004. Available at: www. usb. org/developers/ devclass _docs.

[10] USB Implementers Forum, Inc. Universal Serial Bus Mass Storage Class SpecificationOverview, Revision 1. 3. September 5, 2008. Available at: www. usb. org/developers/devclass_docs.

[11] USB Implementers Forum, Inc. Universal Serial Bus Mass Storage Class

Bulk-OnlyTransport，Revision 1.0. September 31，1999. Available at：www.usb.org/developers/devclass_docs.

[12] USB Implementers Forum，Inc. USB Device Class Definition for Personal HealthcareDevices，Release 1.0，November 8，2007. Available at：www.usb.org/developers/devclass_docs.