

Freescalé MQX™ RTOS USB 主机用户 指南

Document Number: MQXUSBHOSTUG
Rev 7, 08/2014

小节编号	内容 标题	页
第 1 章 开始之前		
1.1	关于本书.....	5
1.2	首字母缩略词和缩写词.....	5
1.3	参考资料.....	6
第 2 章 熟悉内容		
2.1	简介.....	7
2.2	软件套件.....	7
2.3	目录结构.....	8
第 3 章 设计概述		
3.1	设计目标.....	11
3.1.1	模块化.....	11
3.1.2	硬件抽象.....	11
3.1.3	性能.....	11
3.2	目标设计.....	12
3.2.1	完整的 USB 协议栈示意图.....	12
3.3	组件概述.....	13
3.3.1	主机概述.....	13
3.3.1.1	主机应用程序.....	14
3.3.1.2	类驱动程序库.....	14
3.3.1.3	大容量存储类驱动程序设计的示例.....	15
3.3.1.4	架构和数据流.....	15
3.3.1.5	通用类 API.....	17
3.3.1.6	USB 第 9 章 API.....	18
3.3.1.7	主机 API.....	19
3.3.1.8	HCI (主机控制器接口)	19

小节编号	标题	页
第 4 章		
开发应用程序		
4.1	编译 Freescale MQX USB 主机协议栈.....	21
4.1.1	为什么要重新编译 USB 主机协议栈.....	21
4.1.1.1	开始之前.....	21
4.1.1.2	USB 目录结构.....	22
4.1.1.3	Freescale MQX RTOS 中的 USB 主机协议栈编译工程.....	23
4.1.1.3.1	编译后处理.....	23
4.1.1.3.2	编译目标.....	23
4.1.1.4	重新编译 Freescale MQX USB 主机协议栈.....	24
4.2	主机应用程序.....	24
4.2.1	背景信息.....	24
4.2.2	创建工程.....	24
4.2.3	定义驱动程序信息表.....	26
4.2.4	主应用程序函数流程.....	27
4.2.4.1	初始化主机控制器.....	27
4.2.4.2	注册服务.....	28
4.2.4.3	设备的枚举过程.....	28
4.2.4.4	选择设备上的接口.....	29
4.2.4.5	取回和储存管道句柄.....	29
4.2.4.6	向设备发送/接收数据.....	30
4.2.4.7	其他主机函数.....	30
4.3	USB HDK 在 MQX RTOS 3.8.1 中的修改.....	31
4.3.1	USB 主机应用程序的移植步骤.....	31

第 1 章 开始之前

1.1 关于本书

本文档描述了 Freescale MQX™ USB 主机协议栈架构。本文档不对 USB 1.1 和 USB 2.0 信息进行区分，两者存在差异的情况除外。

本文档包含以下主题：

- 第 1 章, “开始之前”
- 第 2 章, “熟悉内容”
- 第 3 章, “设计概述”
- 第 4 章, “开发应用程序”
- 附录 A, “利用软件工作”
- 附录 B, “人机接口设备 (HID) 例子”
- 附录 C, “大容量存储设备 (MSD) 例子”
- 附录 D, “虚拟通信 (COM) 例子”
- 附录 E, “音频主机例子”

1.2 首字母缩略词和缩写词

表 1-1. 首字母缩略词和缩写词

术语	说明
API	应用程序编程接口
CDC	通信设备类
DCI	设备控制器接口

下一页继续介绍此表...

表 1-1. 首字母缩略词和缩写词 (继续)

术语	说明
HCI	主机控制器接口
HID	人机接口设备
MSD	大容量存储设备
MSC	大容量存储类
PHD	个人健康护理设备
PHDC	个人健康护理设备类
QOS	服务质量
SCSI	小型计算机系统接口
USB	通用串行总线

1.3 参考资料

建议参考以下资料：

- 通用串行总线规范版本 1.1
- 通用串行总线规范版本 2.0
- *Freescle MQX RTOS USB 主机 API 参考手册 (MQXUSBHOSTAPIRM)*

第 2 章 熟悉内容

2.1 简介

Freescale MQX USB 主机协议栈可以工作在低速和全速模式下。MQX USB 主机协议栈支持完整的软件协议栈，包括基本内核驱动程序、类驱动程序和示例程序，可用于实现所需目标产品。本文档旨在帮助客户了解 USB 主机协议栈，以及提供开发和调试 USB 应用的有用信息。在您使用本文档时，可以结合软件套件中提供的 API 文档一起使用，这有助于您更好地理解 USB 主机协议栈。本文档的目标读者为熟悉基本 USB 术语及直接参与 USB 主机协议栈上产品开发的固件和软件工程师。

2.2 软件套件

Freescale MQX USB 主机协议栈软件由以下内容组成：

- 类层 API
- 设备框架
- 主机层 API

类层 API 和 USB 主机类 API 函数可在类这层使用。这使您可以实现基于预定义类的新应用程序。本文档描述了五种通用类实现：通信设备类 (CDC)、人机接口设备 (HID)、大容量存储类 (MSD)、集线器类和音频类 API 函数，均在软件套件中提供。为这些类定义的 API 函数可用于创建应用程序。

设备框架函数用于支持对所有 USB 设备均通用的设备请求。

主机层 API 函数可用于主机层，并支持在类层上的实现。

欲了解更多信息，请参见《Freescale MQX™ RTOS USB 主机 API 参考手册》(MQXUSBHOSTAPIRM)。

2.3 目录结构

以下各项显示了 Freescale MQX USB 主机协议栈的完整目录树

- Freescale MQX RTOS \usb\host\examples——用于主机固件开发的示例代码。
- Freescale MQX RTOS \usb\host\source——驱动设备控制器内核的所有源文件。
- Freescale MQX RTOS \lib——USB 软件树编译时所有库和头文件的输出位置。

下表简要阐述了源码树中每个目录的目的。

表 2-1. 目录结构

目录名称	描述符
usb	软件协议栈的根目录
\lib\usbh	编译的输出目录
usb\host\build	*.mcp——用于编译完整工程
usb\host\examples	主机端类驱动程序示例
usb\host\source\classes	类驱动程序源文件
usb\host\source	USB 主机 API 源文件
usb\host\source\host\khci	USB 主机内核驱动程序
usb\host\source\rtos\mqx	RTOS 层源程序

下图显示了默认的目录结构。

注

下图不一定能代表最新的目录树，但是可以提供 USB 主机协议栈结构的概念。

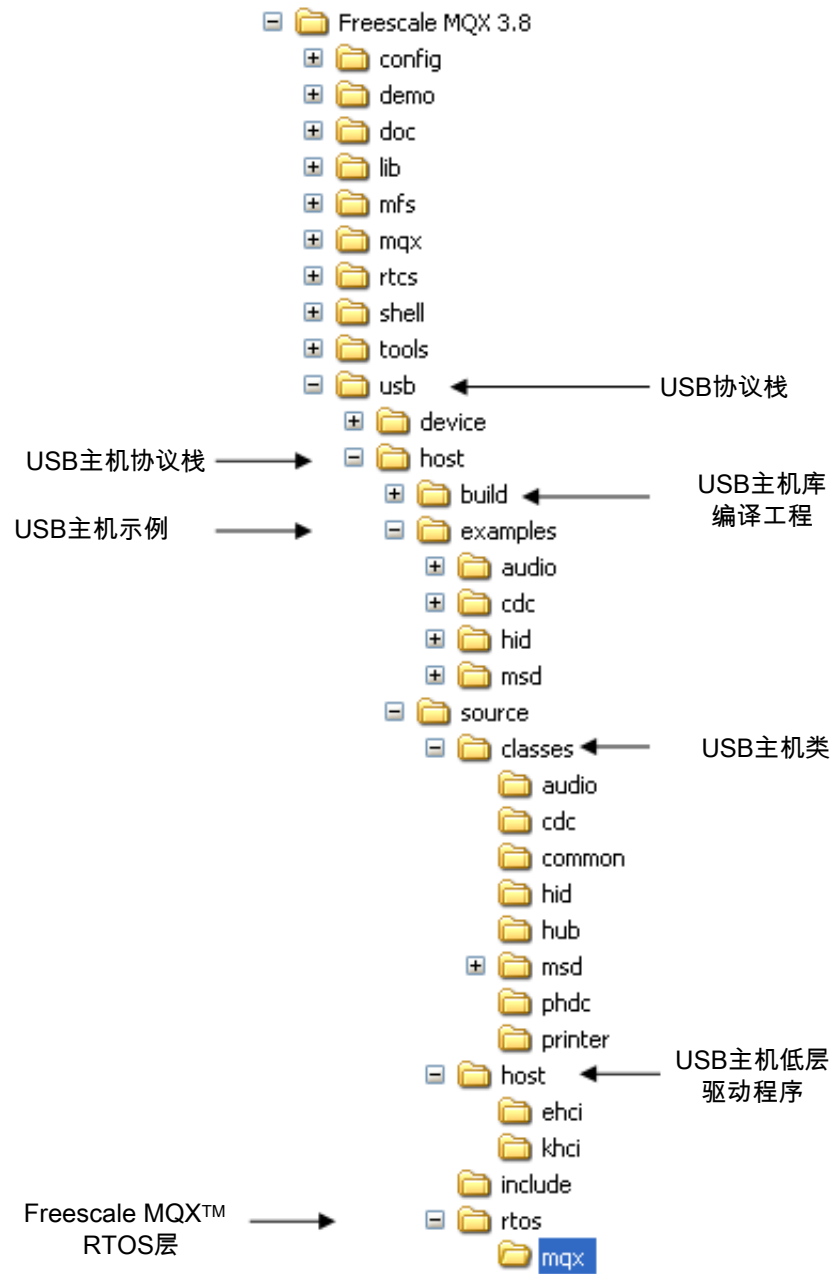


图 2-1. 默认目录结构

第 3 章 设计概述

3.1 设计目标

Freescale MQX USB 主机协议栈针对下列目标而设计。

3.1.1 模块化

嵌入式软件设计的目标之一是以最小的代码尺寸和产品设计所需的最少组件实现目标功能。USB 主机协议栈设计具有灵活的架构，因此可从软件套件中移除不必要的组件。USB 主机协议栈团队不断努力以模块插件形式增加额外功能，这些插件在编译时可完全被删除。

3.1.2 硬件抽象

USB 主机协议栈的目标之一是使应用程序代码具有真正的硬件无关性。USB 主机协议栈同时提供 API 例程，可检测内核速度，并允许应用程序层根据其运行速度进行智能选择。

3.1.3 性能

USB 主机协议栈已经为保持高性能而设计，因为 API 提供了与硬件的紧密互操作。为了获取目标的最佳性能，可从协议栈软件中删去类驱动程序、操作系统或不必要的例程。USB 主机协议栈设计工作于单线程，能保证最小的中断延迟。检查代码时，请注意 USB 主机协议栈实际为双层协议栈，其目的始终是确保以最少的层数和代码行数实现抽象目标。

3.2 目标设计

USB 系统由主机和一个或多个响应主机请求的设备组成。下图显示了 USB 协议栈的目标设计。

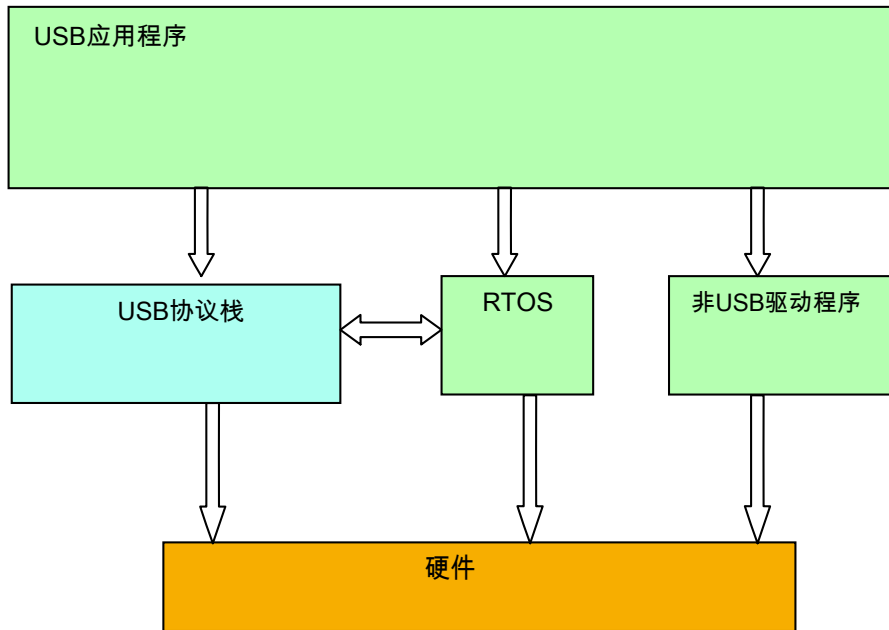


图 3-1. USB 主机协议栈的目标设计

客户应用可以通过 Freescale MQX USB 协议栈与 USB 硬件通信。图 3-2 显示了详细的协议栈设计。

3.2.1 完整的 USB 协议栈示意图

下图显示了 USB 协议栈示意图。

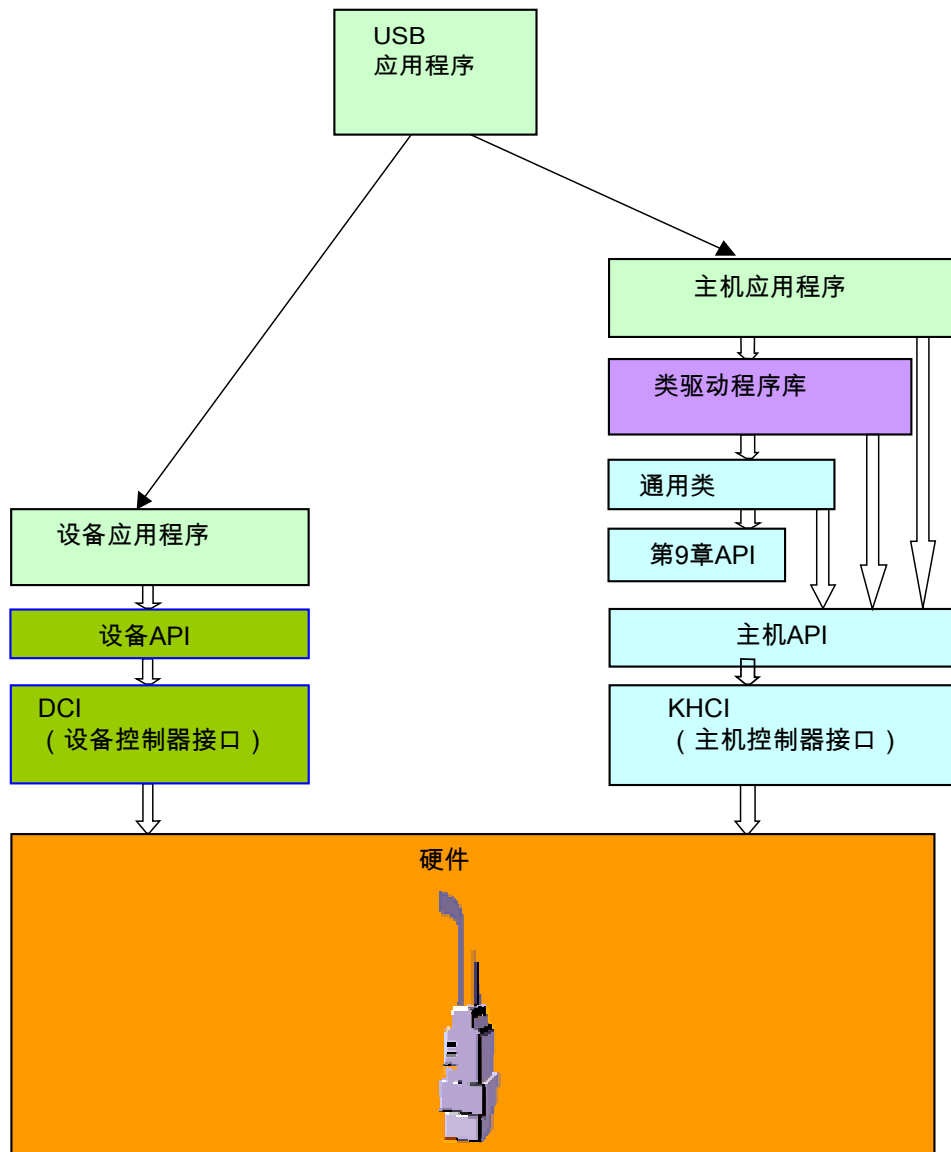


图 3-2. 完整的 USB 协议栈示意图

3.3 组件概述

本部分说明了 MQX USB 主机协议栈中包含的组件。

3.3.1 主机概述

Freescle MQX USB 主机协议栈的目的是提供 USB 硬件控制器内核抽象。使用主机 API 编写的软件应用程序可以运行在全速或低速内核上，与硬件无关。在 USB 中，主机通过逻辑管道与设备交互。在设备枚举后，软件应用程序需能打开和关闭

管道。当一个设备管道打开后，软件应用程序可以在任意方向上列队传输，并通过错误状态通知传输结果。简言之，主机和设备之间的通信通过逻辑管道完成，这些管道在协议栈中表示为管道句柄。下图中说明的每个框表示一个主机 API。

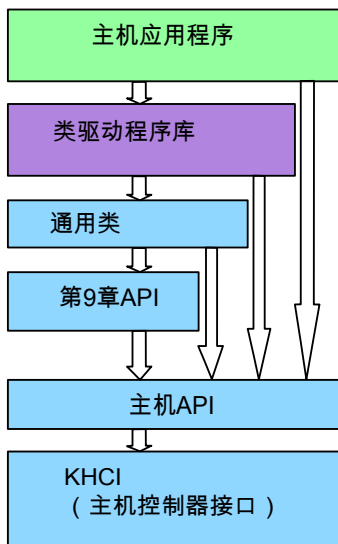


图 3-3. Freescale MQX USB 主机协议栈

3.3.1.1 主机应用程序

主机嵌入式应用程序软件，也称为设备驱动程序，用于目标设备或设备类的实现。Freescale MQX USB 主机协议栈包含一些用于 USB 主机类的示例。应用程序软件可以使用 API 例程来启动和停止与 USB 总线的通信。它可以挂起总线或等待用于总线的恢复信号。关于更多详细信息，请参见第 4 章“开发应用程序”。

3.3.1.2 类驱动程序库

类驱动程序库是一组封装例程，可以链接到应用程序。这些例程实现了在 USB 类规范中定义的设备类的标准功能。需注意的是，虽然类软件库有助于应用程序级的实现，但有些 USB 设备并没有应用类规范，因而很难编写单个应用程序代码实现与所有设备的通信。存储设备大多遵循通用软驱接口 (UFI) 规范，因此编写一个可与大多数存储设备通信的应用程序十分简单。

USB 主机协议栈包含一个用于 UFI 命令的类软件库，可以在与此类设备通信时进行调用。需了解的是，类驱动程序库是一组基于 USB 主机协议栈编写的封装例程，并非必须使用的程序。

USB 主机协议栈中的类驱动程序库的设计遵循一个标准模板。所有的软件库均具有一个初始化例程，当接入一个特定类的设备时由 USB 主机协议栈调用。该例程为类驱动程序分配所需的存储区，并构成类句柄结构。当设备中的接口选定时，应用

程序软件可以接收类句柄。请参见第 4 章“开发应用程序”。一旦接口选定，应用程序可以通过该句柄和其他所需参数调用类驱动程序例程。类指定例程的实现途径取决于类，不同类之间的实现途径完全不同。图 3-4 显示了所有类驱动程序库的通用设计。

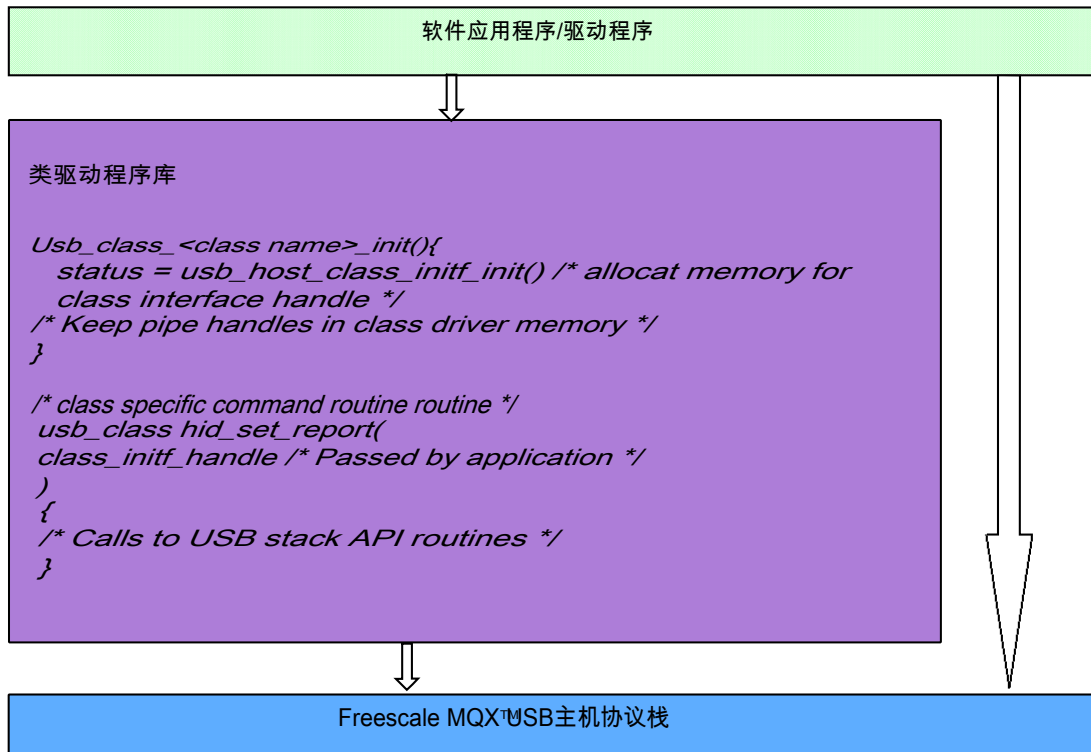


图 3-4. 类驱动程序库的通用设计

3.3.1.3 大容量存储类驱动程序设计的示例

本部分说明了大容量存储类的驱动程序设计。

3.3.1.4 架构和数据流

所有的 USB 大容量存储命令由以下三个字段组成：

- CBW（命令块封装）
- 数据
- CSW（命令状态封装）

关于这些字段的详细信息，请参见关于 USB 大容量存储规范的参考文档。大容量存储类 API 文档中列出了类驱动程序库支持的大容量存储命令。

Freescale MQX USB 主机协议栈大容量存储设备类驱动程序库从应用程序提取命令并在本地队列中排队。然后从 **CBW** 字段开始传输，随后是 **DATA** 和 **CSW** 字段。当状态字段完成后，将会从队列中选择下一个传输并重复相同步骤。软件库可以对 UFI 命令集执行较高层次的调用，如读容量或格式化驱动，并等待直到接收到一个完成中断。

CBW 和 CSW 的字段数据通过两个 USB 标准数据结构进行描述：

- *CBW_STRUCT*
- *CSW_STRUCT*

关于大容量存储命令的所有信息，例如 **CBW** 和 **CSW** 结构指针、数据指针和长度等均包含在命令结构 *COMMAND_OBJECT_STRUCT* 中。使用该结构的应用程序在类驱动程序库中排队等候命令。如果应用程序想向存储设备发送供应商指定调用，它必须填写该结构字段，并通过能将命令传递给 USB 的例程将该结构发送给库。

所有命令在大容量存储 API 中均具有一个单函数入口点。不过，所有命令均映射到一个单函数，使用 *#defined* 关键字使代码更有效率：

```
usb_mass_ufi_generic() (see usb_mass_ufi.c and usb_mass_ufi.h).
```

所有数据缓冲区传输均通过指针和长度参数执行。在函数中不执行缓冲区复制。

大容量存储设备类驱动程序和 USB 主机驱动程序之间的接口为 **usb_host_send_data()** 函数。该函数的参数如下：

- USB 主机外设上的句柄。
- 具有大容量存储设备的通信通道上的句柄。
- 描述传输参数、大小和数据指针的结构。

图 3-5 中的对象表示如下内容：

- 黄色箭头表示在 USB 主机驱动程序中产生的事件。
- 黑色箭头表示在大容量存储命令队列中的进出动作。
- 方块表示执行及调用的函数。

数字 1-10 表示事件或队列活动的顺序。

名称中带有 *_mass_* 的函数来自 USB 大容量存储库。名称中带有 *_host_* 的函数链接到 USB 主机驱动程序库。若要深入了解代码流，请查看类驱动程序库源文件中的函数。

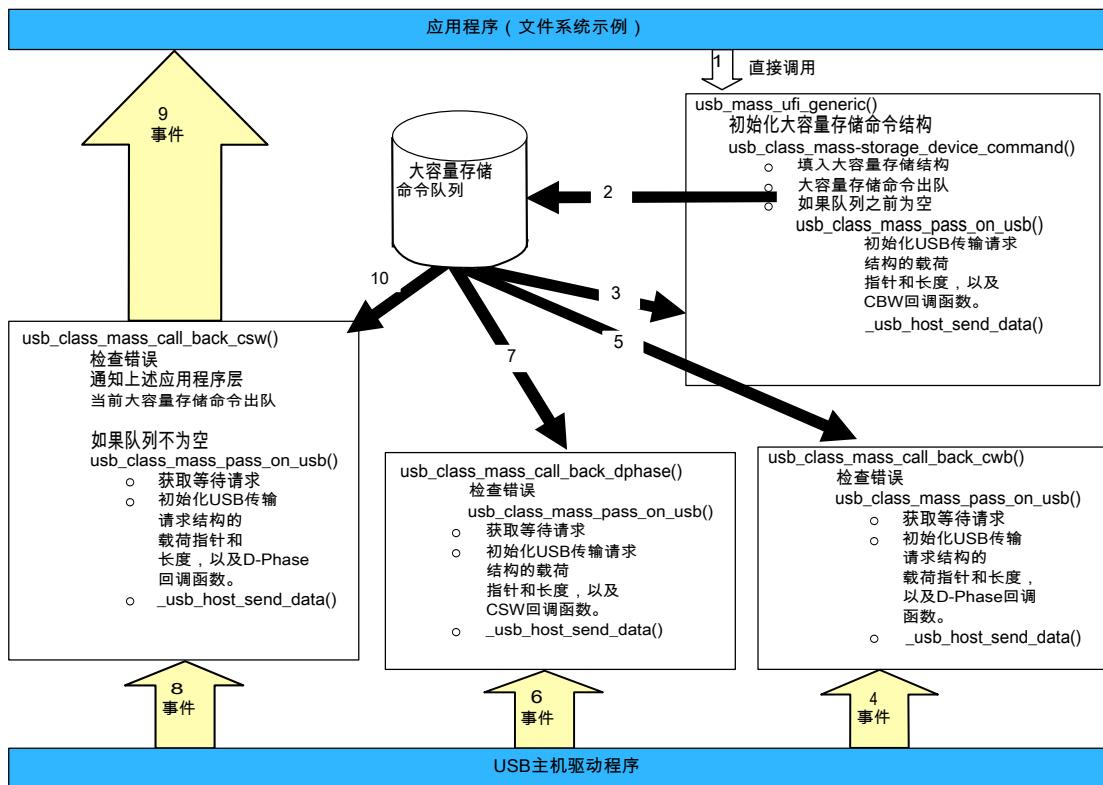


图 3-5. 大容量存储驱动程序代码流

3.3.1.5 通用类 API

通用类 API 为 USB 的通用类规范和 USB 系统的操作系统级抽象提供了接口实现。该层与主机 API 层函数进行互操作。从 API 文档中很难看出哪些接口属于该层。这是一种权衡设计, 旨在重复使用接口, 尽量精简代码。

通用类层中的接口要求在 USB 中所有设备均采用相同的命令序列进行枚举。当一个 USB 设备接入主机硬件时, 会产生中断, 较低层的驱动程序则会调用通用类层来启动设备。通用类层中的接口会分配存储区、指定 USB 地址和枚举设备。在识别设备描述符后, 通用类层会搜索为该设备或接入设备而注册的应用程序。如果找到匹配的应用程序, 则回调此应用程序, 以启动与设备的通信。关于应用程序如何处理接入设备的更多信息, 请参见第 4 章, “开发应用程序”。下图显示了设备接插是如何工作的。

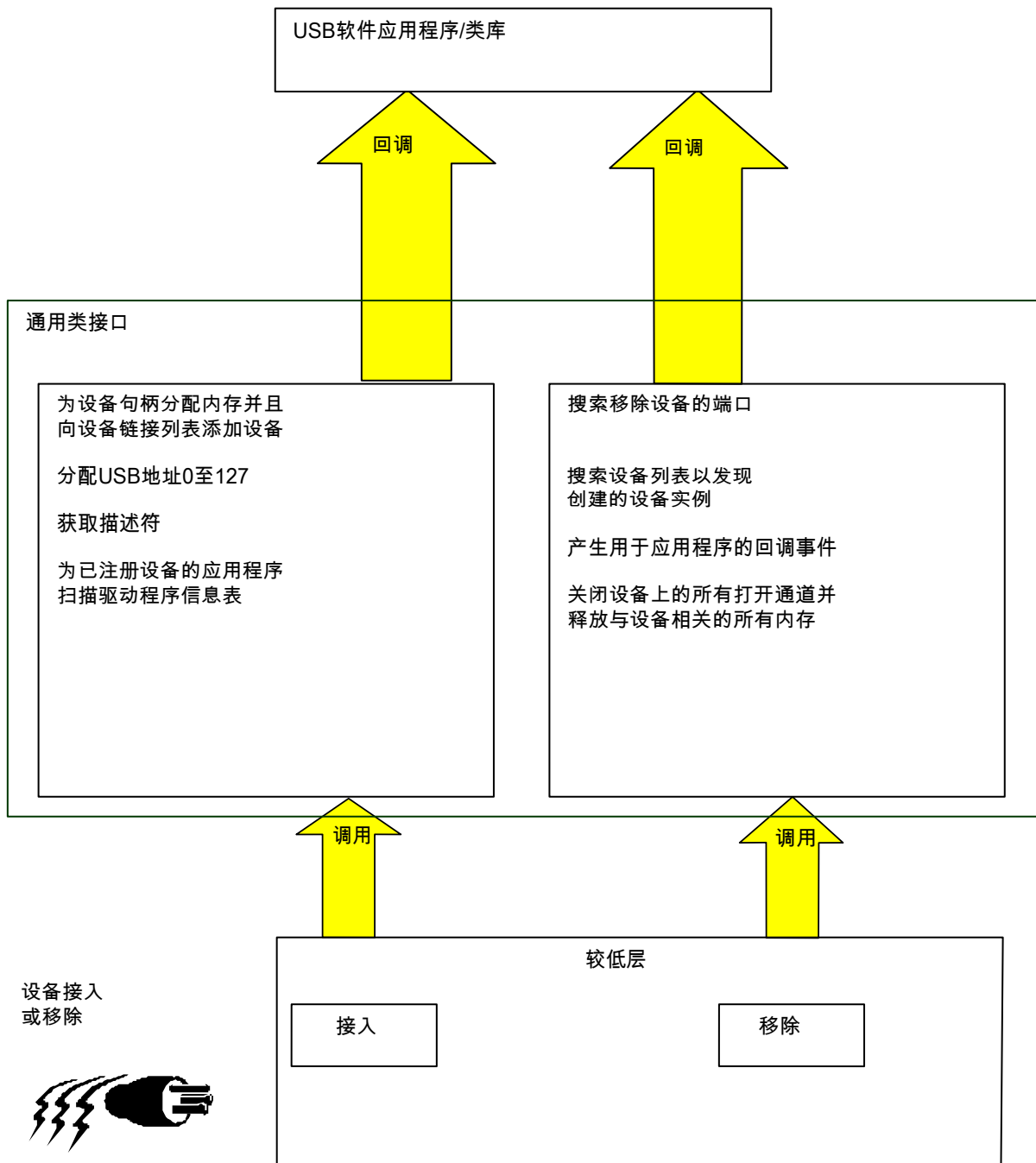


图 3-6. 如何接入和移除设备

3.3.1.6 USB 第 9 章 API

USB 规范文件中的第 9 章旨在标准化所有 USB 设备应用的命令协议。所有 USB 设备需要响应来自主机的一组特定请求。该 API 是低层 API，实现第 9 章中所述的所有 USB 命令。所有的客户应用程序可以只使用该 API 进行编写，而无需通用类 API 或类软件库。

第 9 章命令不在本文档的讨论范围内，且要求详细了解 USB 协议和 USB 设备的较高级抽象。最后，以下列出了该 API 实现的一些示例接口。关于实现的详细信息，请见源代码。

- `usb_host_ch9_dev_req()`——用于发送控制管道设置数据包。
- `_usb_host_ch9_clear_feature()`——用于清除特定 USB 命令。
- `_usb_host_ch9_get_descriptor()`——用于从设备接收描述符。

3.3.1.7 主机 API

主机 API 是 Freescale MQX USB 主机协议栈的硬件抽象层。该层实现的接口与下层的 USB 控制器无关。例如，`usb_host_init()`通过调用适当的硬件相关接口来初始化主机控制器。同样，`usb_host_shutdown()`可用来关断适当的硬件主机控制器。以下是该层实现的架构化函数。

- 当调用 `usb_host_open_pipe()`时，该层从预分配管道内存结构池中分配管道。
- 该层负责维护每个管道上所有待传输的列表，当管道关闭时释放所有内存空间。
- 该层还负责维护在协议栈上注册的所有服务（回调）的链接列表。当发生特定硬件事件（如接入或移除）时，将产生回调并通知上一层。
- 该层提供取消 USB 传输的接口，通过调用硬件相关的函数实现。
- 该层还提供其他硬件控制接口，如关断控制器和挂起总线等功能。

若要更深入地了解 API 层内部的源程序，请阅读 API 接口并跟踪其到硬件驱动程序。

3.3.1.8 HCI（主机控制器接口）

HCI 完全是一个硬件相关接口集，用于列队和处理 USB 传输和搜索硬件事件。若要理解该层的源代码，您必须了解硬件。

MQX RTOS 实现了两种 HCI 的驱动程序：

- KHCI——用于 ColdFire V2 和某些 Kinetis 器件。
- EHCI——用于高端 ColdFire V4 和某些 Kinetis 及 PowerPC 器件。

第 4 章 开发应用程序

4.1 编译 Freescale MQX USB 主机协议栈

本部分说明了编译 MQX USB 主机协议栈的操作。

4.1.1 为什么要重新编译 USB 主机协议栈

如果进行了以下操作，则需要重新编译 USB 主机协议栈：

- 改变了编译器选项（优化等级）
- 改变了 USB 主机协议栈编译时间配置选项
- 合并了对 USB 主机源代码所做的修改

警告

不建议修改 USB 主机协议栈数据结构。如果修改了 USB 主机协议栈数据结构，则主机软件开发工具的 Precise Solution™ 主机工具系列中的一些组件可能无法正常执行。只有当您具有丰富 USB 主机协议栈经验时，才可以修改 USB 主机协议栈数据结构。

4.1.1.1 开始之前

在重新编译 USB 主机协议栈之前，先执行以下内容：

- 阅读《MQX RTOS 用户指南》，以获取 MQX RTOS 重编译指令。这对于 USB 主机协议栈也适用。
- 阅读 Freescale MQX RTOS 随附的《MQX RTOS 版本说明》，以获取特定于目标环境和硬件的信息。

- 具备目标环境所需的工具：
 - 编译器
 - 汇编器
 - 链接器
- 熟悉 USB 主机协议栈的目录结构和重编译指令，这在版本记录和后续章节中进行了说明。

4.1.1.2 USB 目录结构

下图显示了 MQX USB 主机协议栈的目录结构。

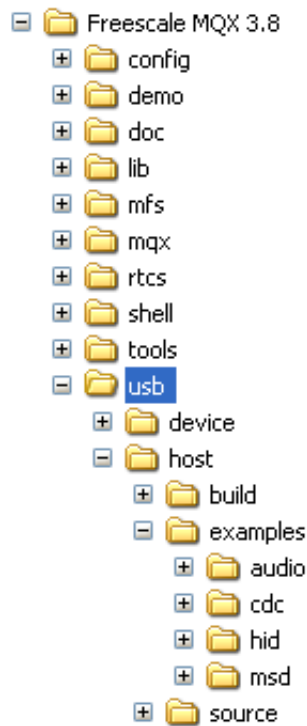


图 4-1. MQX USB 主机协议栈目录结构

下表说明了 USB 主机协议栈的目录结构。

config	主配置目录
<board>	特定板目录，包含主配置文件（user_config.h）
usb\host	Freescale MQX 中的 USB 主机协议栈的根目录
\build	
\<compiler>	特定开发工具的编译文件（工程文件）
	下一页继续介绍此表...

<code>\examples</code>	
<code>\example</code>	示例及示例编译工程的源文件 (.c)。
<code>\source</code>	所有的 USB 主机协议栈源代码文件
<code>\lib</code>	
<code>\<board>.<comp>\usb</code>	用于编译硬件和环境的 USB 主机协议栈库文件

4.1.1.3 Freescale MQX RTOS 中的 USB 主机协议栈编译工程

USB 主机协议栈编译工程的结构与 Freescale MQX RTOS 中包含的其他内核库工程相同。对于给定的开发环境（例如 CodeWarrior），编译工程位于 `usb\host\build\<compiler>` 目录中。USB 主机协议栈代码并不针对特定的开发板或处理器衍生品。但是，对于每一个支持的板子，都有一个单独的 USB 主机协议栈编译工程。结果库文件编译输出到特定板的输出目录 `lib\<board>.<compiler>`。

独立于板子的代码之所以要编译到特定板的输出目录是因为这样便于分别配置每块开发板。编译时间用户配置文件取自特定板目录 `config\<board>`。对于两块不同的开发板，用户可能希望编译不同的结果库代码。更多信息，请参见《MQX RTOS 用户指南》。

4.1.1.3.1 编译后处理

所有 USB 主机协议栈编译工程被配置为在顶层 `lib\<board>.<compiler>\usb` 目录中生成二进制库文件。例如，用于 M52259EVB 板的 CodeWarrior 库编译到 `lib\m52259evb.cw\usb` 目录中。

USB 主机协议栈编译工程还设定了执行编译后的批处理文件，将所有公共头文件复制到目标目录中。这使得输出 `\lib` 目录成为应用程序代码可访问的唯一位置。MQX 应用程序工程需要使用 USB 主机协议栈服务，无需引用 USB 主机协议栈源码树。

4.1.1.3.2 编译目标

开发工具提供多种编译配置，称为编译目标。Freescale MQX USB 主机协议栈中的所有工程至少包含两个编译目标：

- 调试目标——编译器优化设为低，以简化调试。使用该目标编译的库以“_d”后缀命名。例如，`lib\m52259evb.cw\usb\usb_hdk_d.a`。
- 发布目标——编译器优化设为最高，以实现最小的代码大小和最快的执行速度。这样得到的代码很难进行调试。生成的库名字没有任何后缀。例如，`lib\m52259evb.cw\usb\usb_hdk.a`。

4.1.1.4 重新编译 Freescale MQX USB 主机协议栈

重新编译 MQX USB 主机协议栈库非常简单，只需在开发环境中打开合适的编译工程，并对其进行编译。切记选择合适的编译目标进行编译或编译所有目标。

关于重新编译 MQX USB 主机协议栈的具体信息和示例应用程序，请参见《Freescale MQX RTOS 版本说明》。

4.2 主机应用程序

下述步骤用于实现主机功能。

4.2.1 背景信息

在 USB 系统中，主机软件按照协议定义的规则控制总线，并与目标设备对话。设备由配置表示，配置是一个或多个接口的集合。每个接口由一个或多个端点组成。从应用程序软件来看，每一个端点由一个逻辑管道表示。

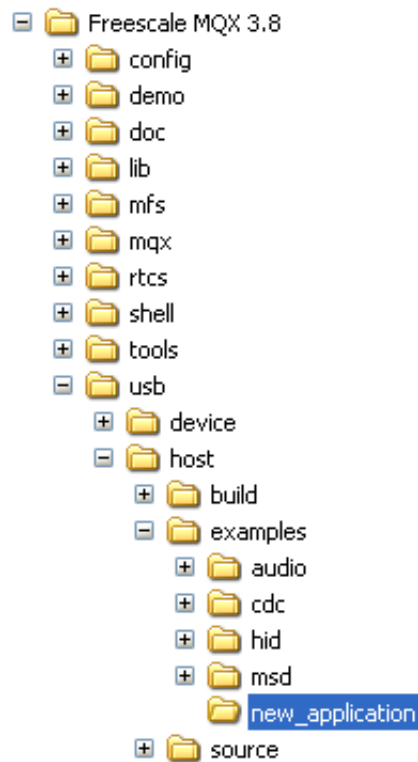
主机应用程序软件向 USB 主机协议栈注册服务例程，并描述了驱动程序信息表中的回调例程。当连接 USB 设备时，USB 主机协议栈驱动程序会自动枚举设备，为应用程序产生中断。在连接的设备上，为每个接口产生一个中断。如果应用程序希望与一个接口对话，它可以选择该接口并接收所有端点的管道句柄。参见主机 API 文档中的源代码示例，以了解寻找通道句柄时调用哪些例程。应用程序软件接收到管道句柄后，开始与管道进行通信。如果软件希望与另一个接口或配置对话，它可以再次调用合适的例程以选择另一个接口。

在与 USB 设备开始通信之前，USB 主机协议栈还包含一些代码。USB 协议栈示例可以仅使用主机 API 编写。不过，大部分协议栈示例使用类驱动程序编写。类驱动程序作为主机 API 的功能补充。它使目标功能实现起来更为容易（详情请见示例源代码），无需处理麻烦的标准例程应用。对于任意具体设备，在主机应用驱动程序中使用以下代码步骤。

4.2.2 创建工程

执行以下步骤来开发一个新的应用程序：

1. 在/host/examples 目录下创建一个新的应用程序目录。新的应用程序目录用于创建新的应用程序。


图 4-2. 创建应用程序目录

2. 从现有的类似应用程序中复制 `usb_classes.h` 文件。

`usb_classes.h` 用于定义在应用程序中使用的类。

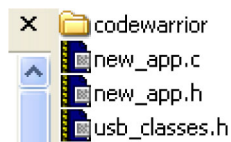
例如，如果您希望创建使用 HID 和 HUB 的应用程序，可以对其进行如下定义：

```
#define USBCLASS_INC_HID
#define USBCLASS_INC_HUB
```

3. 创建一个目录，该目录用于存放新应用程序的工程文件。


图 4-3. 创建 CodeWarrior 工程和 `usb_classes.h` 文件

4. 创建用于主应用程序函数和回调函数的新文件。


图 4-4. 创建头文件和代码文件

- `new_app.h` 文件包含应用程序类型和定义：

示例：

```
/* Used to initialize USB controller */
#define MAX_FRAME_SIZE 1024
```

```

#define HOST_CONTROLLER_NUMBER      0
#define HID_BUFFER_SIZE             4
#define USB_DEVICE_IDLE             (0)
#define USB_DEVICE_ATTACHED        (1)
#define USB_DEVICE_CONFIGURED      (2)
#define USB_DEVICE_SET_INTERFACE_STARTED (3)
#define USB_DEVICE_INTERFACED      (4)
#define USB_DEVICE_SETTING_PROTOCOL (5)
#define USB_DEVICE_INUSE           (6)
#define USB_DEVICE_DETACHED        (7)
#define USB_DEVICE_OTHER           (8)
/*
** Following structs contain all states and pointers
** used by the application to control/operate devices.
*/
typedef struct device_struct {
    uint32_t                DEV_STATE; /* Attach/detach state */
    _usb_device_instance_handle DEV_HANDLE;
    _usb_interface_descriptor_handle INTF_HANDLE;
    CLASS_CALL_STRUCT      CLASS_INTF; /* Class-specific info */
} DEVICE_STRUCT, * DEVICE_STRUCT_PTR;
/* Alphabetical list of Function Prototypes */
#ifdef _cplusplus
extern "C" {
#endif
void usb_host_hid_rcv_callback(_usb_pipe_handle, void *, unsigned char *, uint32_t,
    uint32_t);
void usb_host_hid_ctrl_callback(_usb_pipe_handle, void *, unsigned char *, uint32_t,
    uint32_t);
void usb_host_hid_mouse_event(_usb_device_instance_handle,
    _usb_interface_descriptor_handle, uint32_t);
#ifdef _cplusplus
}
#endif
#endif

```

- new_app.c 文件包含驱动程序信息、回调函数和事件函数，以及主函数。

4.2.3 定义驱动程序信息表

驱动程序信息表定义了目标应用程序所支持和处理的设备。该表定义了 USB 设备的 PID、VID、类和子类。当一个设备与表中记录匹配时，主机协议栈会调用驱动程序信息表中的回调函数。然后，应用程序可以与设备进行通信。以下结构定义了该表中的一个成员。如果供应商-产品对与设备不匹配，则会检查类-子类和协议，以进行匹配。在子类和协议结构成员中使用 0xFF 来匹配任意子类/协议。

```

/* Information for one class or device driver */
typedef struct driver_info
{
    uint8_t        idVendor[2]; /* Vendor ID per USB-IF */
    uint8_t        idProduct[2]; /* Product ID per manufacturer */
    uint8_t        bDeviceClass; /* Class code, 0xFF if any */
    uint8_t        bDeviceSubClass; /* Sub-Class code, 0xFF if any */
    uint8_t        bDeviceProtocol; /* Protocol, 0xFF if any */
    uint8_t        reserved; /* Alignment padding */
    event_callback attach_call; /* event callback function*/
} USB_HOST_DRIVER_INFO, * USB_HOST_DRIVER_INFO_PTR;

```

以下为一个驱动程序信息表示例。请查看示例源代码。请注意，下表定义了属于 boot class 的 HID MOUSE 设备。表中通常以 NULL 项结束，用于查找终点。

由于 HID MOUSE 应用程序中使用了两个类 (HID 和 HUB), 因此 DriverInfoTable 变量具有 3 个元素。针对两个类, 有两个事件回调函数:

usb_host_hid_keyboard_event 用于 HID 类, **usb_host_hub_device_event** 用于 HUB 类。

```

/* Table of driver capabilities this application wants to use */
static USB_HOST_DRIVER_INFO DriverInfoTable[] = {
{
    {0x00, 0x00},          /* Vendor ID per USB-IF          */
    {0x00, 0x00},          /* Product ID per manufacturer   */
    USB_CLASS_HID,         /* Class code                     */
    USB_SUBCLASS_HID_BOOT, /* Sub-Class code                 */
    USB_PROTOCOL_HID_KEYBOARD, /* Protocol                       */
    0,                     /* Reserved                       */
    <add the name of your event callback function here>
    usb_host_hid_keyboard_event /* Application call back function */
},
    /* USB 1.1 hub */
{
    {0x00, 0x00},          /* Vendor ID per USB-IF          */
    {0x00, 0x00},          /* Product ID per manufacturer   */
    USB_CLASS_HUB,         /* Class code                     */
    USB_SUBCLASS_HUB_NONE, /* Sub-Class code                 */
    USB_PROTOCOL_HUB_LS,  /* Protocol                       */
    0,                     /* Reserved                       */
    <add the name of your event callback function here>
    usb_host_hub_device_event /* Application call back function */
},
{
    {0x00, 0x00},          /* All-zero entry terminates     */
    {0x00, 0x00},          /* driver info list.             */
    0,
    0,
    0,
    0,
    0,
    NULL},
}

```

4.2.4 主应用程序函数流程

在主应用程序函数中, 需要遵循以下步骤:

1. 初始化硬件
2. 初始化主机控制器
3. 注册服务

在死循环中调用任务。

4.2.4.1 初始化主机控制器

第一步, 需要作为主机工作, 在主机模式下初始化协议栈。这允许协议栈安装主机中断句柄并对协议栈运行所需的内存初始化。以下示例对此进行了说明:

```
error = _usb_host_init(0, 1024, &host_handle);
if (error != USB_OK)
{
    printf("\nUSB Host Initialization failed. Error: %x", error);
    fflush(stdout);
}
```

上述代码中的第二个参数（上述例子中为 1024）为周期帧列表的大小。全速设备可忽略该参数。

4.2.4.2 注册服务

主机初始化完成后，USB 主机协议栈就可以提供服务。应用程序可以参照主机 API 文档注册服务例程。主机 API 文档允许应用程序注册接入服务，不过使用驱动程序信息表的应用程序无需注册此服务，因为驱动程序信息表已注册了回调例程。以下示例显示了如何在主机协议栈上注册服务：

```
error = _usb_host_register_service (host_handle,
    USB_SERVICE_HOST_RESUME,
    App_process_host_resume);
```

当 USB 主机控制器在挂起后恢复工作时，该代码注册了一个名称为 `app_process_host_resume()` 的例程。关于如何在 USB 主机下挂起并恢复工作的详细信息，请参见 USB 规范。

注

有些示例中没有注册服务，因为驱动程序信息表中已经注册了用于接入服务的必要例程。

4.2.4.3 设备的枚举过程

当软件注册了驱动程序信息表和其他服务后，就可以处理设备了。在 USB 主机协议栈中，用户无需编写任何枚举代码。只要设备与主机控制器连接，USB 主机协议栈就会枚举设备并找到支持的接口数。此外，它会扫描注册的驱动程序信息表并为设备的每个接口找到已注册的应用程序。如果设备标准与信息表相匹配，USB 主机协议栈会提供一个回调函数。应用程序软件必须选择接口。下列示例代码显示了该操作：

```
void usb_host_hid_mouse_event
(
    /* [IN] pointer to device instance */
    _usb_device_instance_handle dev_handle,
    /* [IN] pointer to interface descriptor */
    _usb_interface_descriptor_handle intf_handle,

    /* [IN] code number for event causing callback */
    uint32_t event_code
)
{ /* Body */
    INTERFACE_DESCRIPTOR_PTR intf_ptr =
```

```

    (INTERFACE_DESCRIPTOR_PTR) intf_handle;
fflush(stdout);
switch (event_code) {
    case USB_CONFIG_EVENT:
        /* Drop through into attach, same processing */
    case USB_ATTACH_EVENT:
        fflush(stdout);
        printf("State = %d", hid_device.DEV_STATE);
        printf("  Class = %d", intf_ptr->bInterfaceClass);
        printf("  SubClass = %d", intf_ptr->bInterfaceSubClass);
        printf("  Protocol = %d\n", intf_ptr->bInterfaceProtocol);
        fflush(stdout);
        if (hid_device.DEV_STATE == USB_DEVICE_IDLE) {
            hid_device.DEV_HANDLE = dev_handle;
            hid_device.INTF_HANDLE = intf_handle;
            hid_device.DEV_STATE = USB_DEVICE_ATTACHED;
        } else {
            printf("HID device already attached\n");
            fflush(stdout);
        } /* Endif */
        break;
}

```

请注意，在上述代码中，应用程序将与一个调用 `USB_ATTACH_EVENT()` 相匹配，并将接口句柄存放在本地变量 `hid_device.INTF_HANDLE` 中。其还将程序状态改变为

`USB_DEVICE_ATTACHED`。

4.2.4.4 选择设备上的接口

如果已获取了接口句柄，应用程序软件可以选择接口并取回管道句柄。以下代码演示了这个流程：

```

case USB_DEVICE_ATTACHED:
    printf("Mouse device attached\n");
    hid_device.DEV_STATE = USB_DEVICE_SET_INTERFACE_STARTED;
    status = _usb_hostdev_select_interface(hid_device.DEV_HANDLE,
        hid_device.INTF_HANDLE, (void *)&hid_device.CLASS_INTF);
    if (status != USB_OK) {
        printf("\nError in _usb_hostdev_select_interface: %x", status);
        fflush(stdout);
        exit(1);
    } /* Endif */
    break;

```

作为内部信息，`_usb_hostdev_select_interface` 使协议栈分配内存并为与设备开始通信做必要的准备。该例程打开了逻辑管道，并在周期性管道上分配带宽。在复杂算法下，该带宽分配可能非常耗时。

4.2.4.5 取回和储存管道句柄

如果已选定接口，则可以通过调用取回管道句柄，如以下示例所示：

```

pipe = _usb_hostdev_find_pipe_handle(hid_device.DEV_HANDLE,
    hid_device.INTF_HANDLE, USB_INTERRUPT_PIPE, USB_RECV);

```

在该代码中，管道是存放句柄的内存指针（详情参见代码示例）。请注意，该例程指定了取回管道的类型。该代码显示了如何与具有中断管道的鼠标通信以获取该中断管道的管道句柄。

4.2.4.6 向设备发送/接收数据

USB 数据包通过传输请求（TR）在 USB 软件函数上传输。这与 Windows 和 Linux 中的 URB 概念类似。对于 Windows，驱动程序不断发送 URB 到协议栈，等待 URB 完成的事件或回调。每个 URB 完成例程均有一个回调或事件。USB 协议栈的概念也是一样，但 TR 中的字段可能不同。TR 是一种内存结构，其作为整体来描述一次传输。USB 协议栈提供一个称为 `usb_hostdev_tr_init()` 的帮助例程，可用于初始化 TR。每个到协议栈的 TR 具有一个由 `tr_init()` 例程分配的唯一编号。以下代码示例显示了如何调用该例程：

```
usb_hostdev_tr_init(&tr, usb_host_hid_rcv_callback, <parameter>);
```

该例程的输入为 `tr` 指针（指向需初始化的结构）和回调例程（在 TR 完成时调用）的名称。还可提供附加参数，在 TR 完成时回调。与基于 PC 的系统不同，嵌入式系统存储空间有限，因此建议重复利用提供的 TR。应用程序可以使一些 TR 挂起，并在旧的 TR 完成任务后重复使用。详情请参见代码示例。

在 TR 初始化且管道句柄可用后，就可以方便地向设备发送和接收数据了。使用周期性数据的 USB 设备需要通过周期性调用来发送和接收数据。建议使用操作系统定时器，以确保数据收发调用及时完成，从而在设备收发过程中不会丢失数据包。这些是 USB 驱动程序设计的详细内容，超出了本文档的讨论范围。以下代码为接收数据完成的示例。

```
status = _usb_host_rcv_data(host_handle, pipe, &tr);
```

4.2.4.7 其他主机函数

USB 协议栈提供丰富的例程，可用于实现各种可利用的功能。提供的例程可用于打开管道、关闭管道、获取帧数、微帧数，甚至关断协议栈。这些例程易于从名称分辨，且许多例程在代码中多处使用。例如，`_usb_host_bus_control()` 例程可用于在软件控制下随时挂起 USB 总线。类似的，调用 `usbhost_shutdown()` 可关断主机协议栈并释放所有内存。该例程确保所有管道均关闭，且协议栈释放内存。这些函数可根据需要使用。建议查阅使用这些例程的示例，并根据需要复制代码。

4.3 USB HDK 在 MQX RTOS 3.8.1 中的修改

为了在同一个应用程序中支持设备和主机，Freescale MQX 版本 3.8.1 中对 USB 主机和设备协议栈进行了重大修改。这些变更使得内部和外部 API 均作了修改。

USB 协议栈的通用部分移至 *usb/common* 文件夹中。在该文件夹中，您可以找到通用的头文件和能够同时演示主机和设备功能的示例应用程序。公共通用头文件在 USB-HDK 或 USB-DDK 编译后直接复制到 *lib/<board>.<comp>/usb* 文件夹中。

如果从早期 MQX 版本移植，应用程序开发人员应注意下列 API 修改：

BSP:

- 增加了新文件 (*init_usb.c*)，包含用于板上 USB 控制器的初始化结构

USB-HDK:

- USB_STATUS _usb_host_driver_install(usb_host_if_struct *) API 修改
- USB_STATUS _usb_host_init(usb_host_if_struct *) API 修改
- void (_CODE_PTR_tr_callback)(void *, void *, unsigned char *, uint32_t, USB_STATUS) API 修改
- 修改了 TR_INIT_PARAM_STRUCT，并在顶部包含了泛型上层结构。

在文件 *source/bsp/<bsp_name>/<board>.h* 中，您可以找到对于给定板子的 USB 主机控制器的定义，例如：

```
#define USBCFG_DEFAULT_HOST_CONTROLLER (&_bsp_usb_host_ehci0_if)
```

您可以在 *_usb_host_driver_install* 和 *_usb_host_init* 函数中使用这些宏。

4.3.1 USB 主机应用程序的移植步骤

1. 应用程序工程——添加编译器搜索路径 (*lib/<board>.<comp>/usb*) 使编译器可查找到通用的头文件 (*usb.h...*)。该路径的优先级应高于其他 USB 路径。
2. 在您的应用程序中包含以下内容：

```
#include <mqx.h>
#include <bsp.h>
#include <usb.h>
#include <hostapi.h> /* if you want to use USB-HDK */
/* Additionally, include USB host class driver header files... */
```

3. 将低层驱动程序的安装方法改为：

```
res = _usb_host_driver_install(USBCFG_DEFAULT_HOST_CONTROLLER);
```

...传统的 `_usb_host_driver_install` 需要使用额外参数。

4. 将低层驱动程序初始化改为:

```
res = _usb_host_init(USBCFG_DEFAULT_HOST_CONTROLLER, &host_handle);
```

5. 由于 `tr_callback` API 已改变, 应将回调函数的最后一个参数从 `uint32_t` 修改为 `USB_STATUS`。虽然将 `USB_STATUS` 定义为 `uint32_t` 现在不存在什么问题, 但可能会导致将来出现兼容性问题。

6. 如果 USB 主机应用程序显式地准备发送, 应使用 `TR_INIT_PARAM_STRUCT` 类型的变量, 并在调用 `usb_hostdev_tr_init` 函数后修改访问其成员的方式。如上所述, 该结构已经改变。在 `TR_INIT_PARAM_STRUCT` 的成员中添加“G”上层结构。如下所示:

```
tr.G.RX_BUFFER, tr.G.RX_LENGTH, tr.G.TX_BUFFER, tr.G.TX_LENGTH
```

7. 若您将 SRAM 内存作为缓冲区使用, 则在 MK70F KHCI 控制器中会发生已知问题。因此, 强烈建议您使用 `_usb_hostdev_get_buffer(_usb_device_instance_handle, uint32_t, void **)` 函数, 以获得正确分配的缓冲区, 并在设备移除时自动取消分配。为接入主机的设备分配缓冲区:

```
res = _usb_hostdev_get_buffer(handle, size, &buffer);
```


附录 A

利用软件工作

A.1 简介

本章节向您介绍如何在 MQX USB 主机协议栈软件中使用应用程序。本章节介绍以下部分：

- 准备设置
- 编译应用程序
- 运行应用程序

了解 CodeWarrior IDE 将有助于理解本部分。在阅读本章节的同时，可以练习所提及的步骤。

A.1.1 硬件设置

建立如[图 A-1](#) 中所示连接。

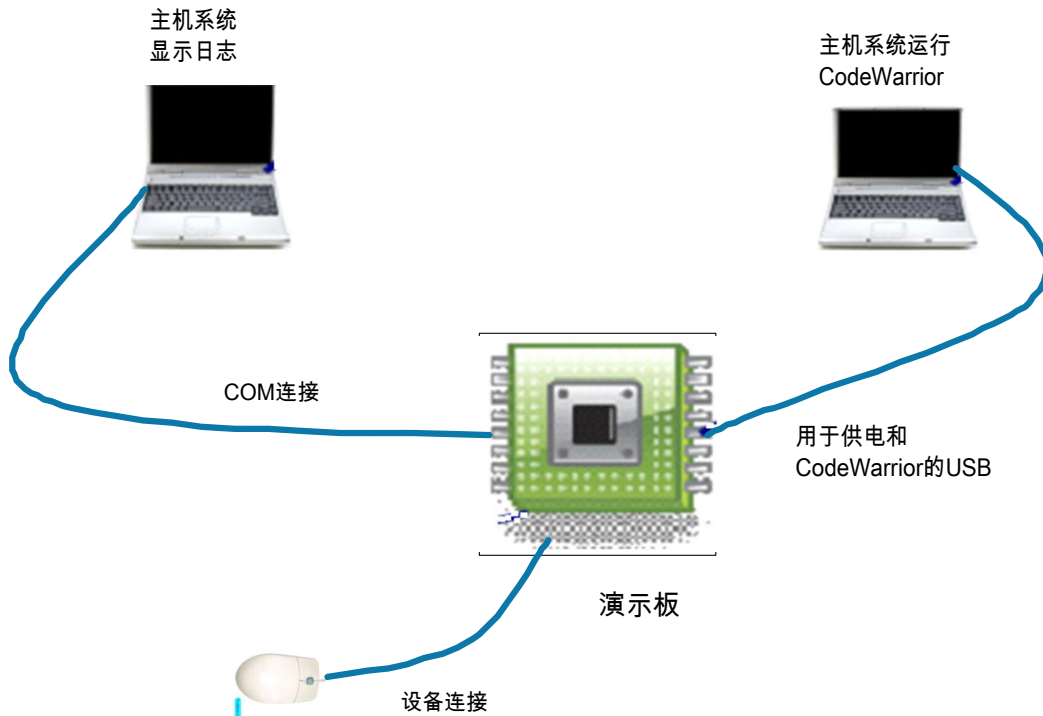


图 A-1. ColdFire V1 USB 设置

- 在个人计算机（已安装软件）和演示板（在其上运行应用程序）之间建立第一个 USB 连接。要求该连接能为演示板供电，并能将镜像下载到闪存中。
- 在演示板和个人计算机之间建立第二个连接，用于显示例子日志。
- 在设备和演示板之间建立第三个连接。

A.1.2 编译应用程序

该软件通过 CodeWarrior 6.3 编译而成，因此含有可用于工程编译的应用程序工程文件。

在编译工程的流程开始之前，请确保您的计算机上已安装了 CodeWarrior 6.3。

为了编译 ColdFire V1 工程：

1. 在 CodeWarrior IDE 中，浏览至工程文件并打开 xxx_m51jmevb.mcp 工程文件。

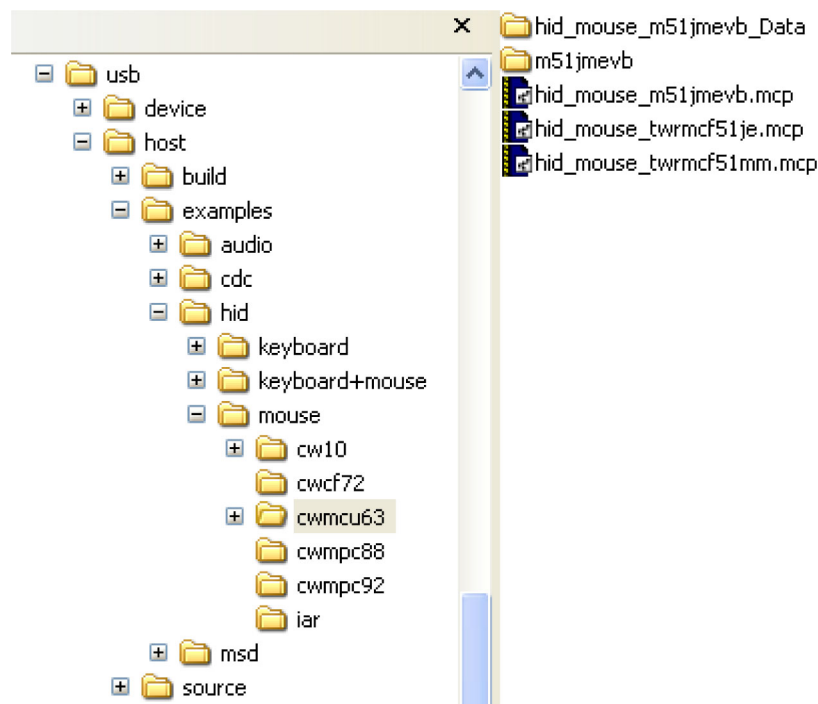


图 A-2. 打开 xxx_m51jmevb.mcp 工程文件

2. 打开工程后，将出现以下窗口。若要编译工程，单击图 A-3 中所示按钮。

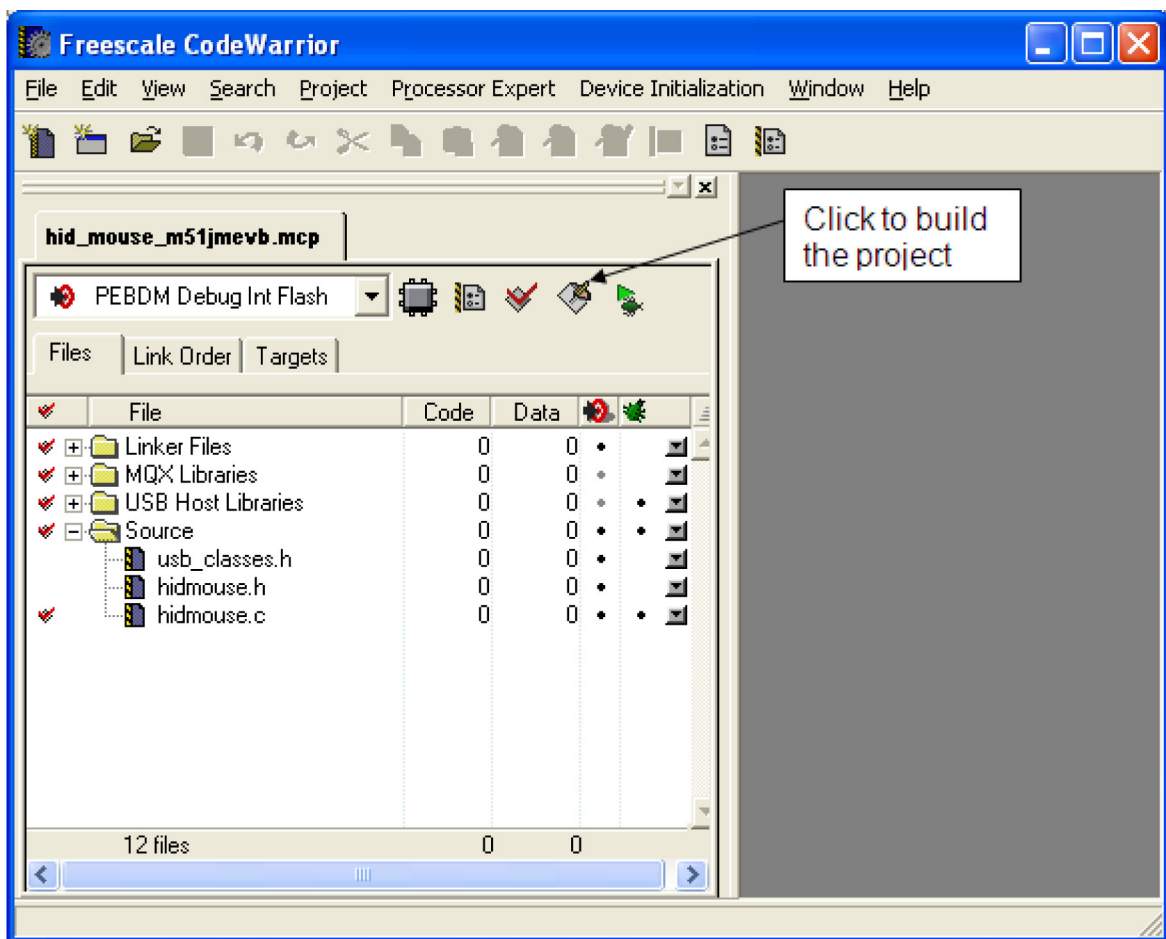


图 A-3. 编译工程

3. 工程编译后，文件中的代码和数据栏位必须显示为已填状态。

A.1.3 运行应用程序

1. 若要运行应用程序，请单击图 A-4 中所示按钮。

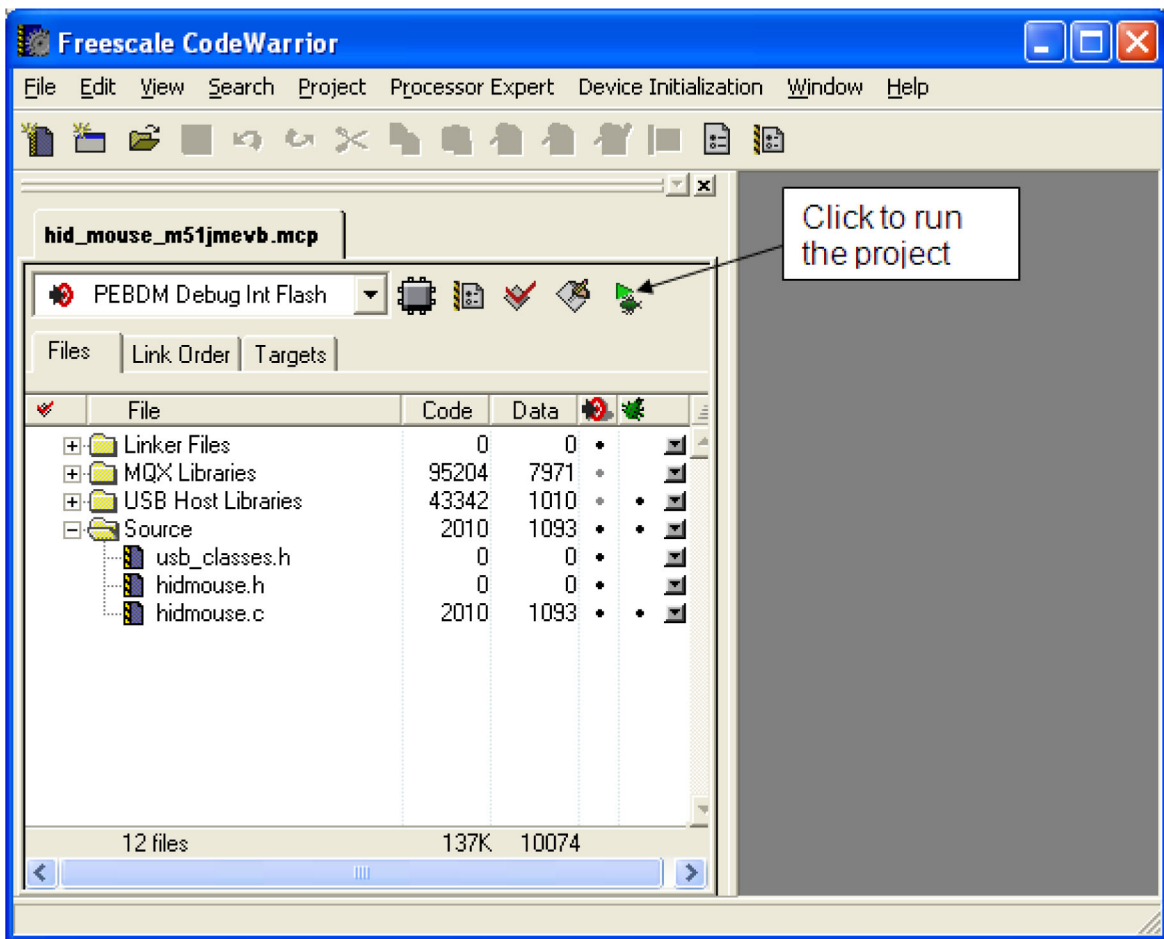


图 A-4. 运行应用程序

2. 单击 **Connect (Reset)** (连接 (复位)) 按钮来连接硬件，如图 A-5 所示。

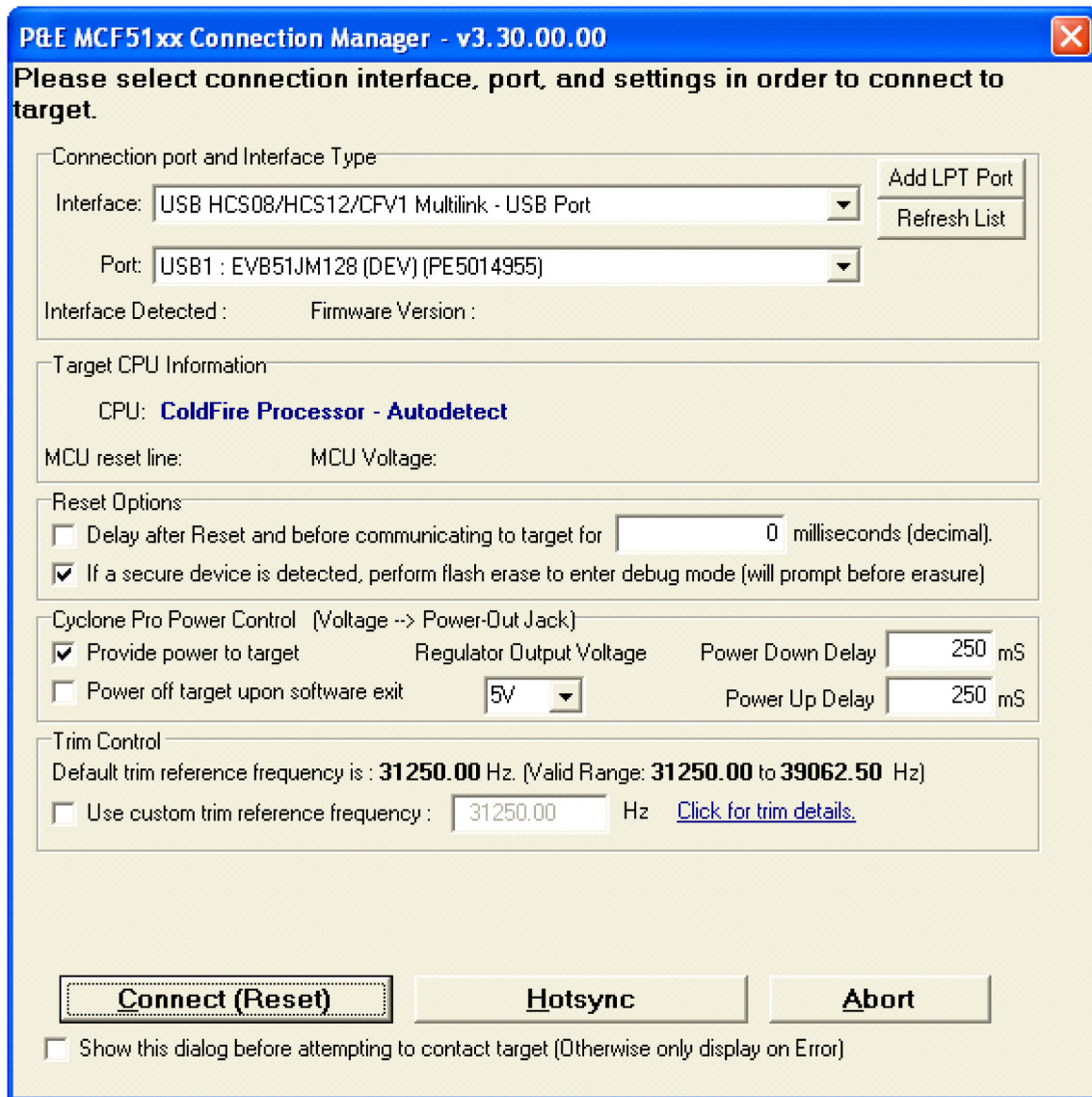


图 A-5. 连接管理器

- 图 A-6 中的弹出窗口打开表示擦除 JM128 闪存，并向其编程编译镜像。

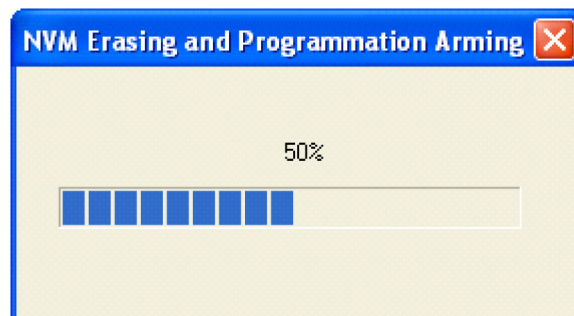


图 A-6. 擦除和编程窗口

- 图 A-7 中的弹出窗口打开表示向 JM128 闪存加载编译镜像。

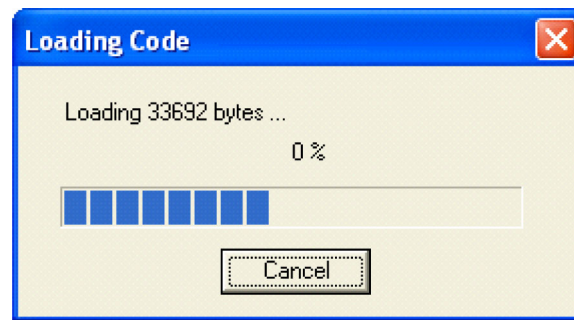


图 A-7. 加载窗口

5. 在编译镜像加载到闪存后，调试器窗口将打开，如图 A-8 所示。单击图 A-8 中所示的绿色箭头以运行已编程镜像。

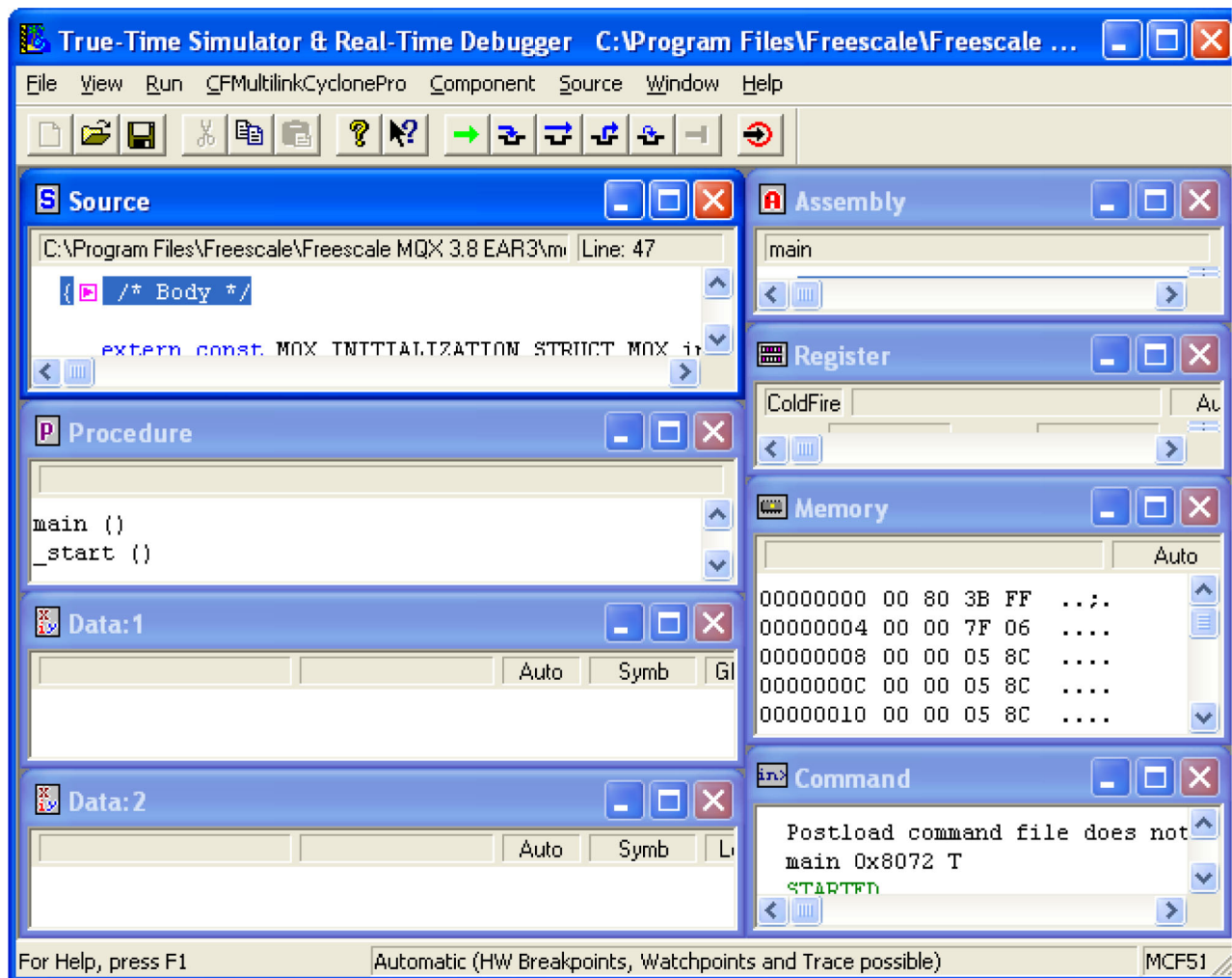


图 A-8. 仿真和实时调试器

A.1.4 设置 HyperTerminal 以获取日志

为了确保应用程序正确运行，利用 HyperTerminal 从与 CFV1 连接的设备中获取事件。以下步骤用于配置 HyperTerminal：

1. 如图 A-9 所示，打开 HyperTerminal 应用程序。

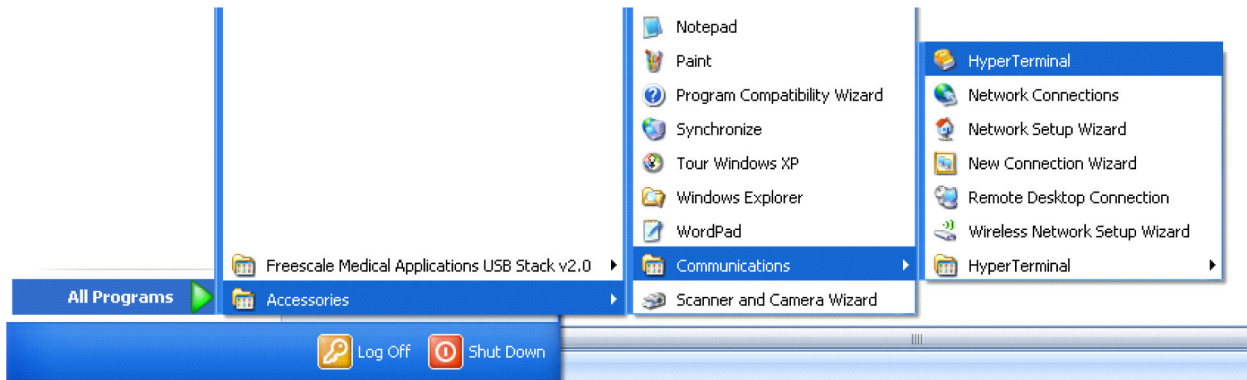


图 A-9. 启动 HyperTerminal 应用程序

2. HyperTerminal 打开时如图 A-10 所示。输入连接名称，并单击 **OK** 按钮。

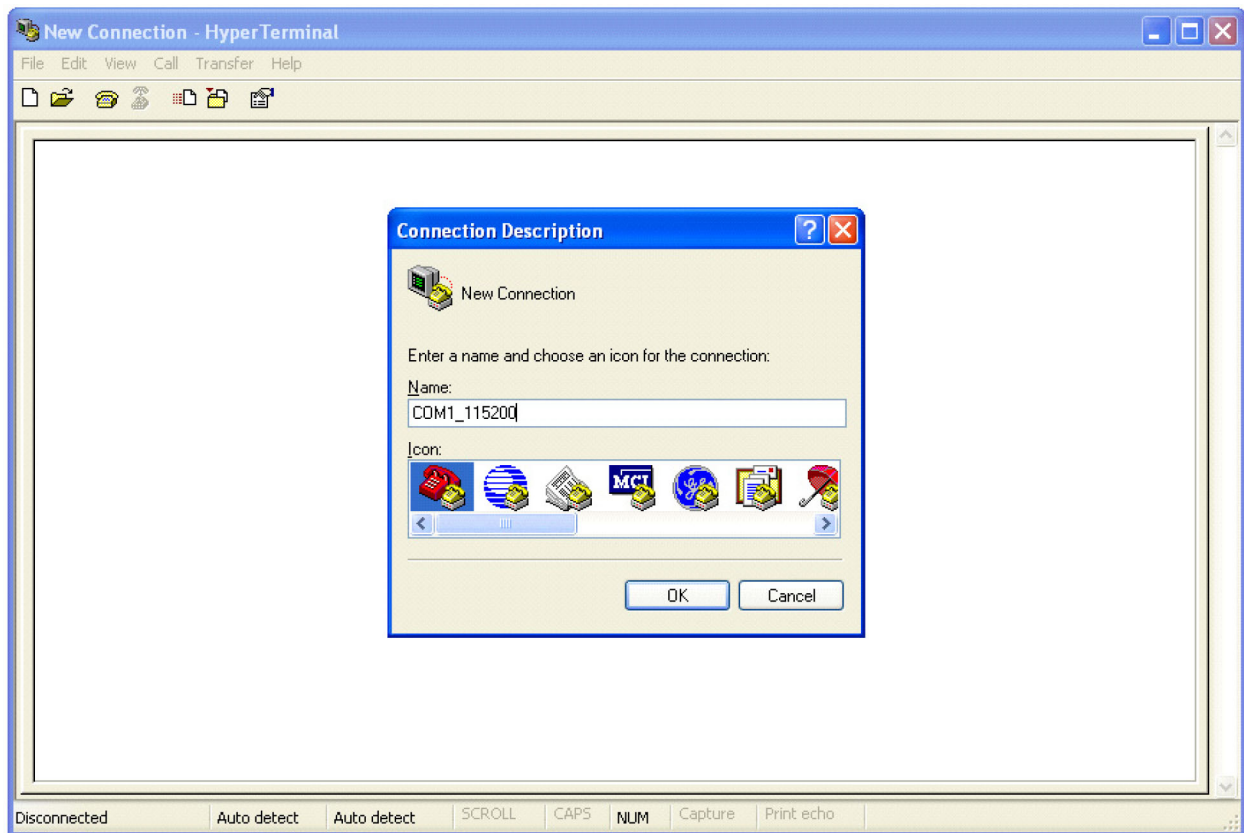


图 A-10. HyperTerminal GUI

3. 将显示如图 A-11 所示的窗口。选择与设备管理器中所示一致的 COM 端口。

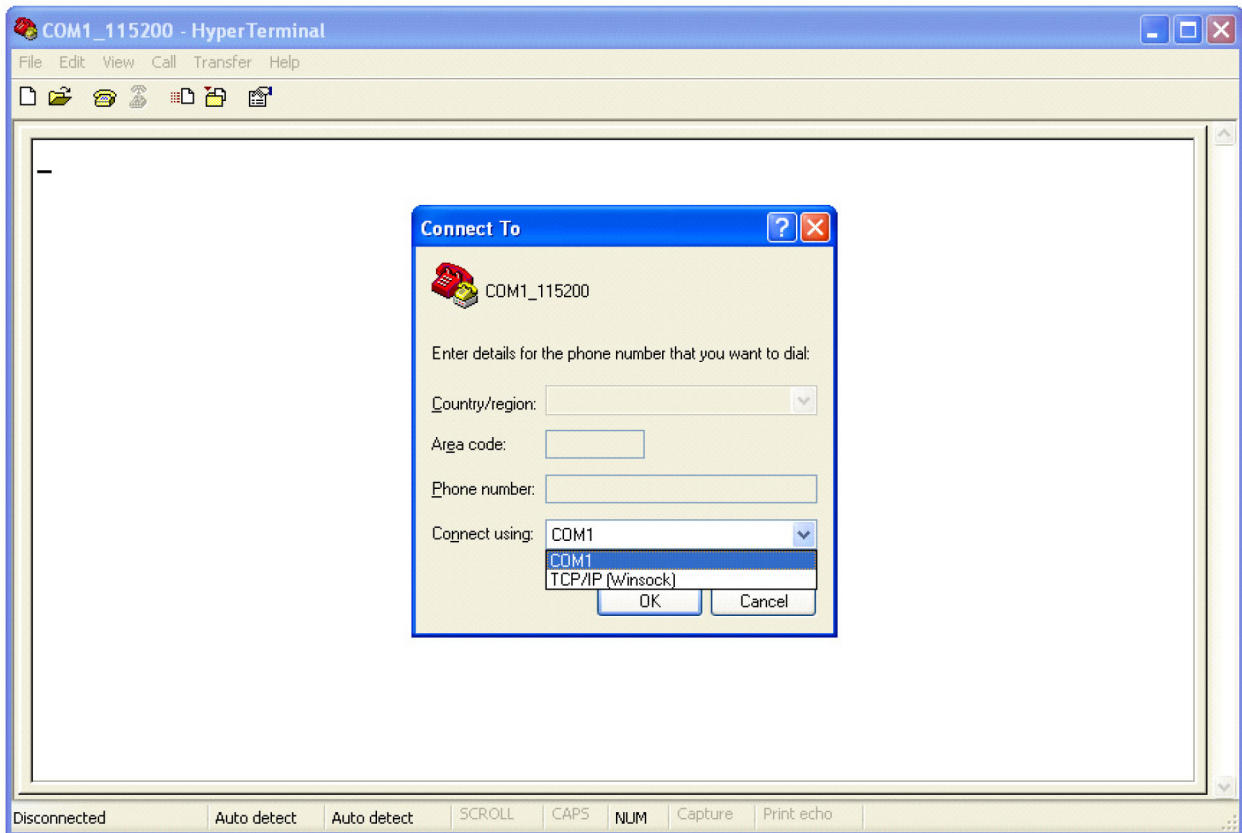


图 A-11. 使用 COM1 连接

4. 如图 A-12 所示，配置虚拟 COM 端口的波特率及其他属性。

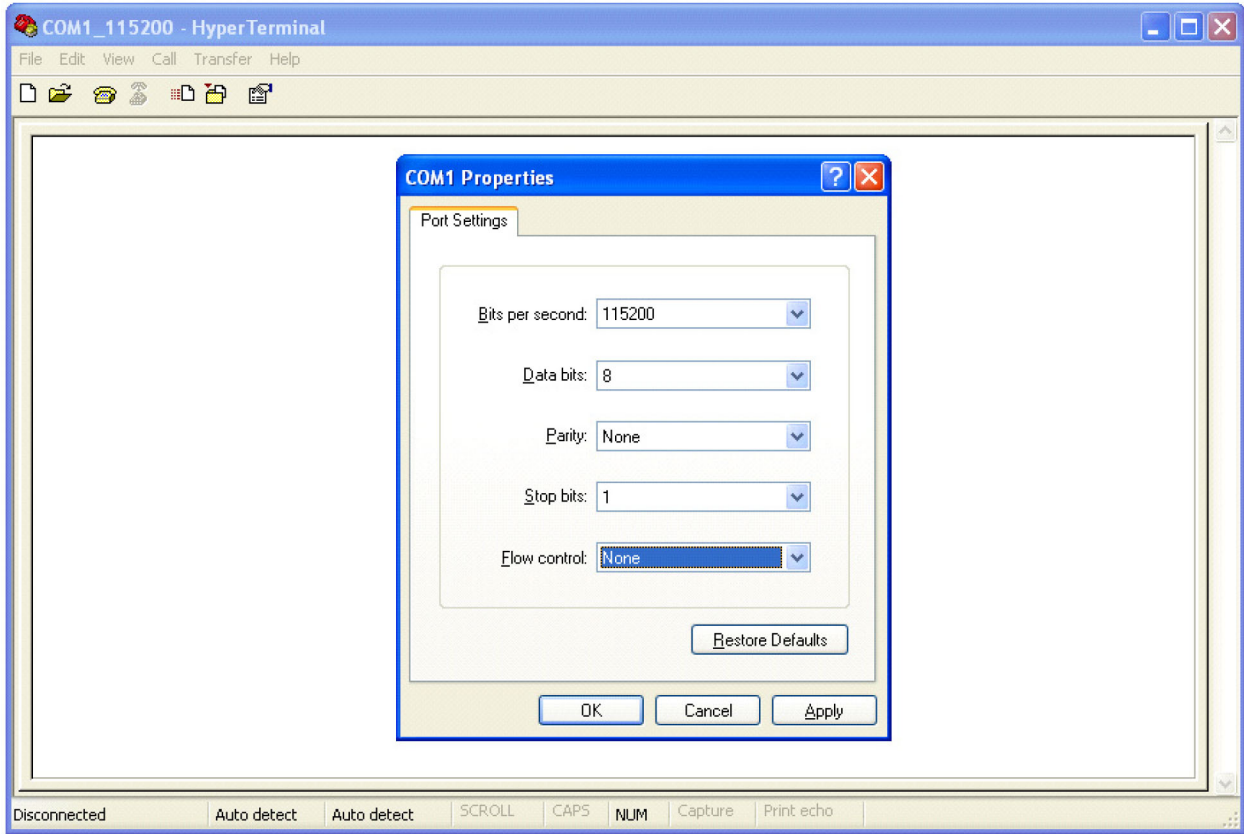


图 A-12. COM1 属性

5. 如图 A-13 所示，配置 HyperTerminal。单击 **OK** 按钮，提交修改。

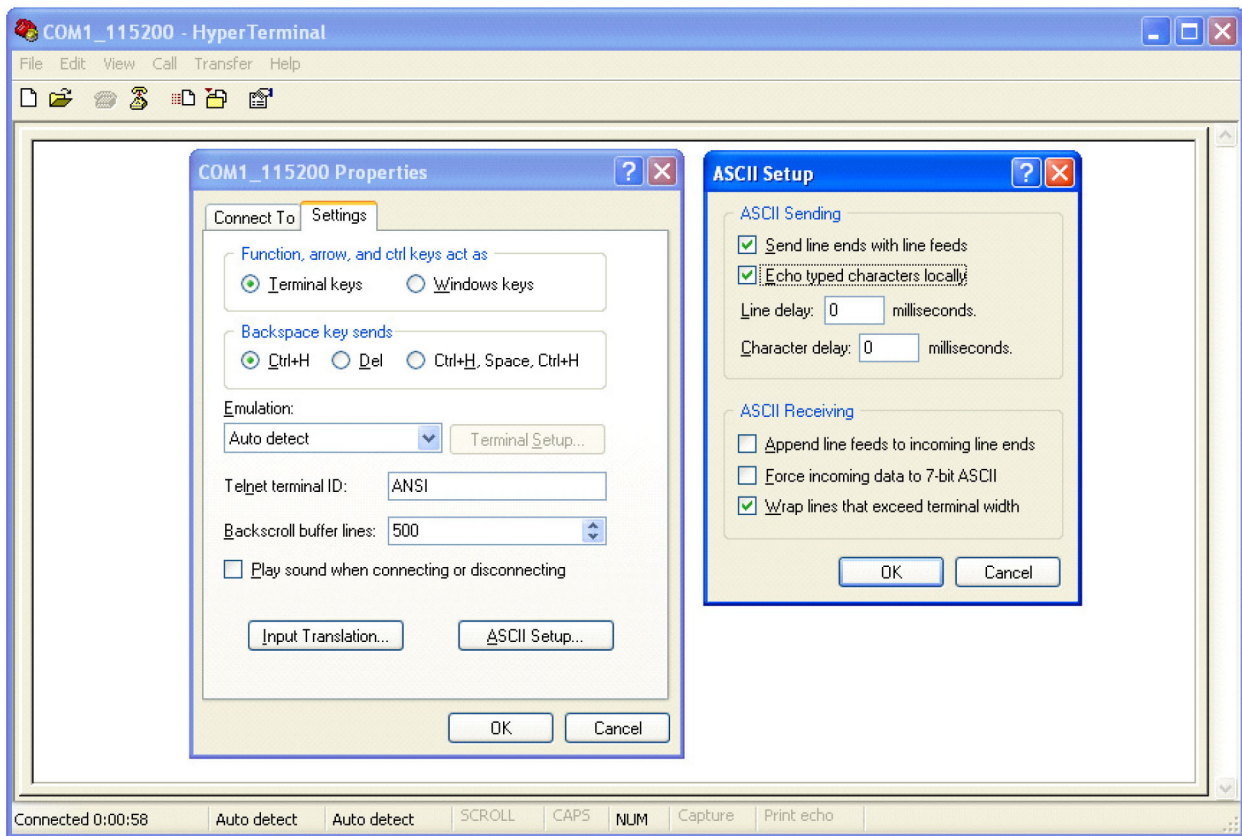


图 A-13. 配置 COM1_115200 - HyperTerminal

6. HyperTerminal 配置完成。

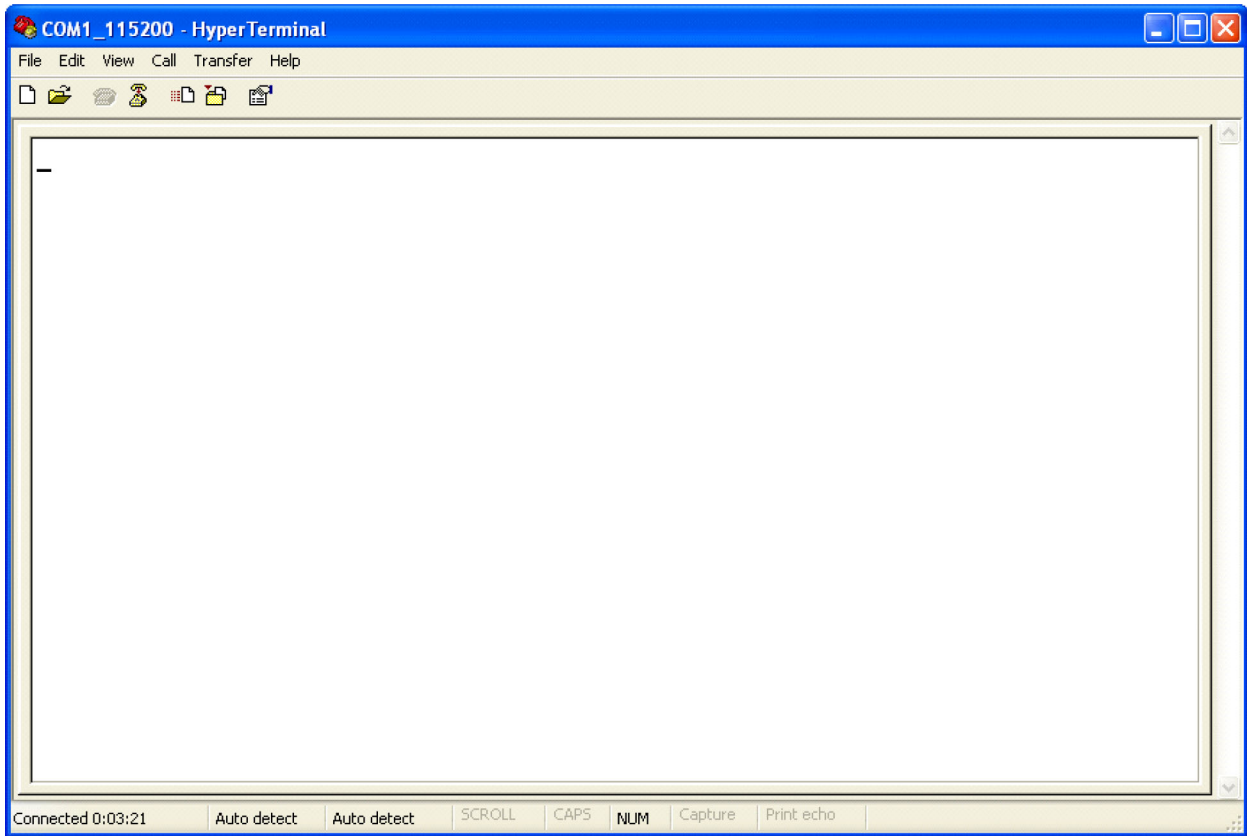


图 A-14. COM1_115200 已配置

附录 B 人机接口设备（HID）演示

B.1 设置例子

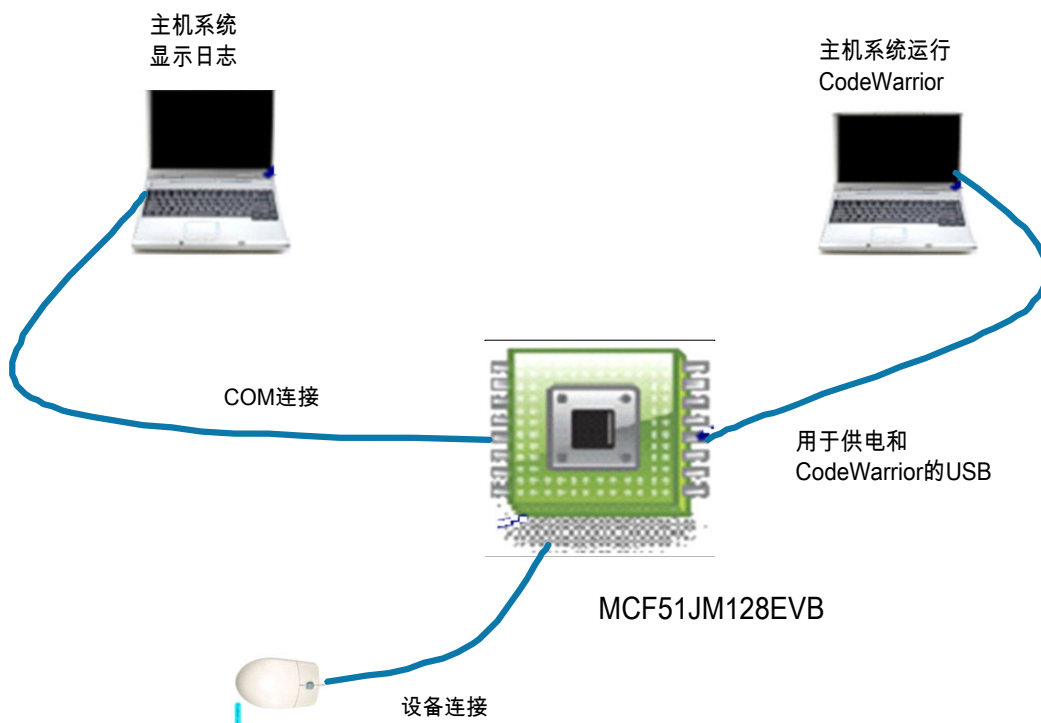


图 B-1. HID 例子设置

图 B-1 说明了 HID 例子的设置。MCF51JM128EVB 用作 USB 主机。MCF51JM128EVB 板与第一台个人计算机通过 USB 线缆连接。该计算机用于向演示板供电，以及将镜像编程到闪存中。MCF51JM128EVB 板与第二台个人计算机通过 COM 端口连接。该计算机用于记录在 USB 主机上发生的事件。设备（鼠标或键盘）连接到 MCF51JM128EVB 板。虽然图 B-1 中显示了两台计算机，也可以仅使用一台计算机实现连接。

B.2 运行例子

HID 工程位于：

`<MQX installation folder>\usb\host\examples\hid\mouse\`

HID 设备类有 3 种应用程序。

- 鼠标
- 键盘
- 键盘与鼠标

B.2.1 鼠标例子

根据以下步骤运行鼠标例子：

1. 打开鼠标应用程序镜像并加载到演示板。
2. 在镜像加载成功后，HyperTerminal 中的内容将如图 B-2 所示：

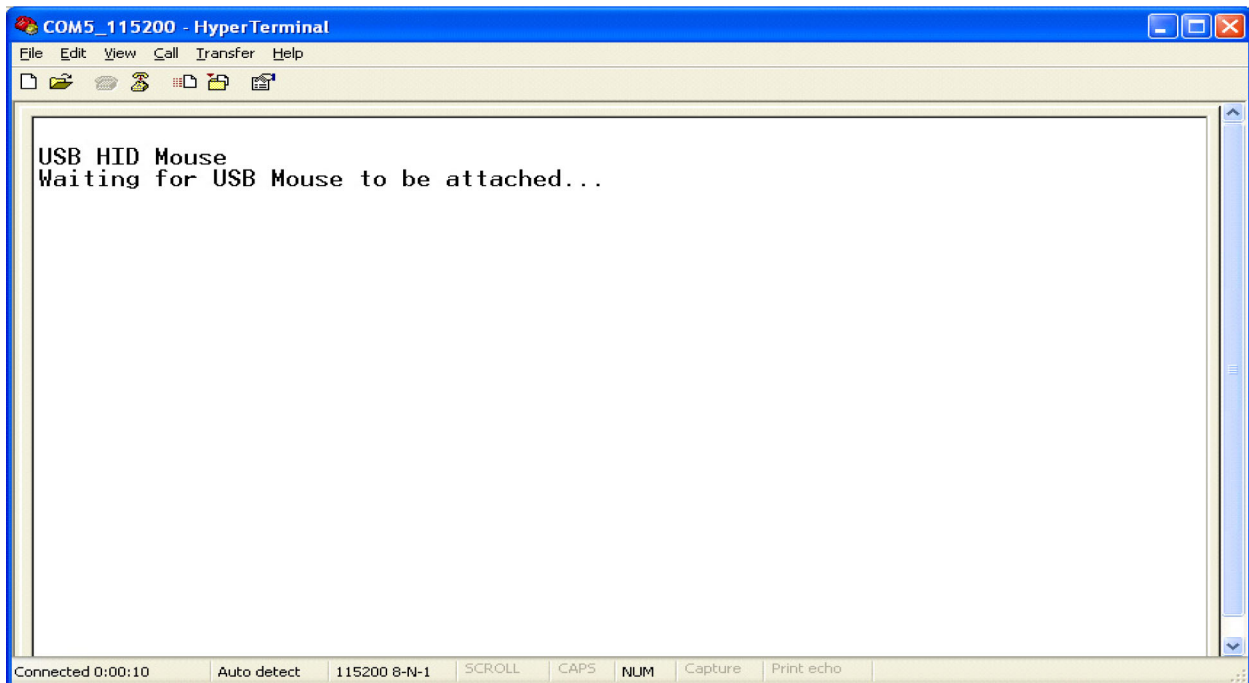
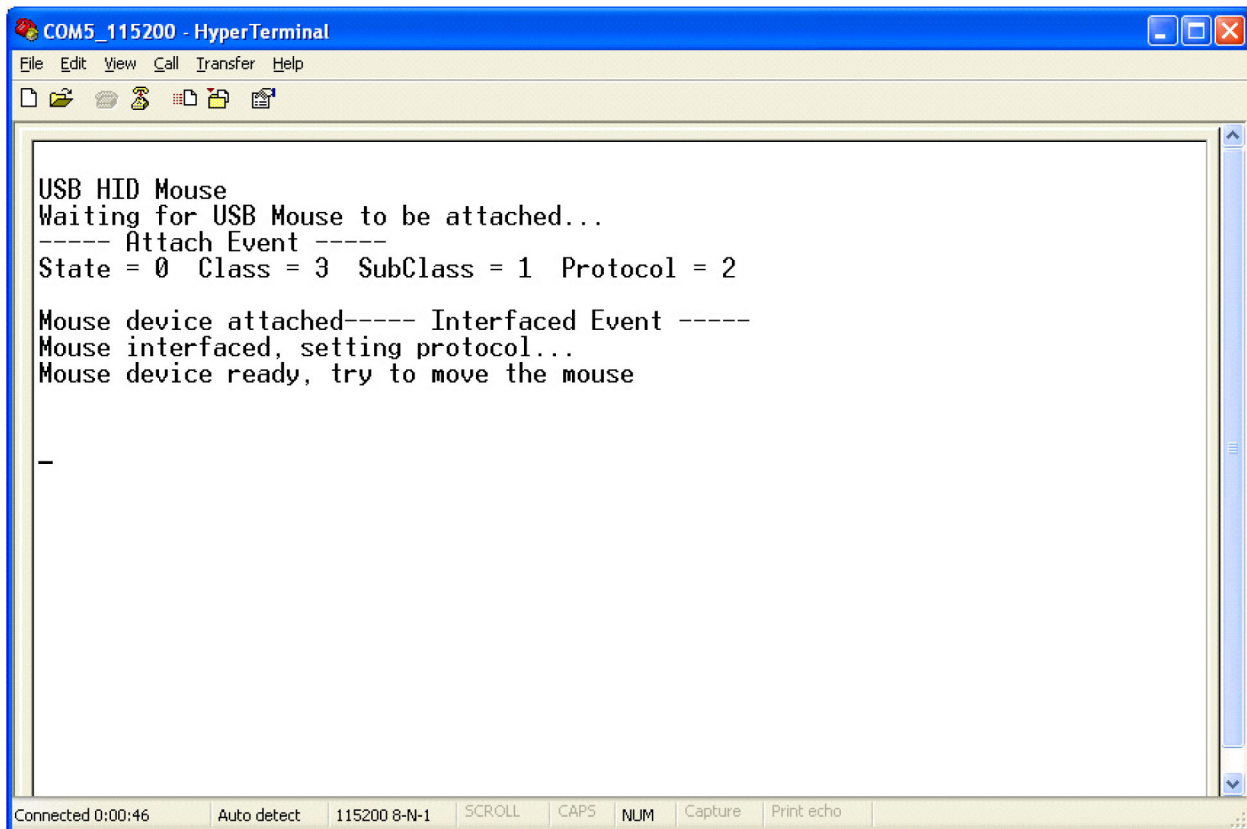


图 B-2. USB 主机正在等待鼠标接入事件

3. 将鼠标接入演示板。Hyperterminal 中的内容如图 B-3 所示：



```
COM5_115200 - HyperTerminal
File Edit View Call Transfer Help
[Icons]
USB HID Mouse
Waiting for USB Mouse to be attached...
----- Attach Event -----
State = 0 Class = 3 SubClass = 1 Protocol = 2

Mouse device attached----- Interfaced Event -----
Mouse interfaced, setting protocol...
Mouse device ready, try to move the mouse

-

Connected 0:00:46 Auto detect 115200 8-N-1 SCROLL CAPS NUM Capture Print echo
```

图 B-3. 鼠标已接入

4. 当事件 (单击鼠标右键和单击鼠标左键等) 实现时, 这些事件会如图 B-4 所示:

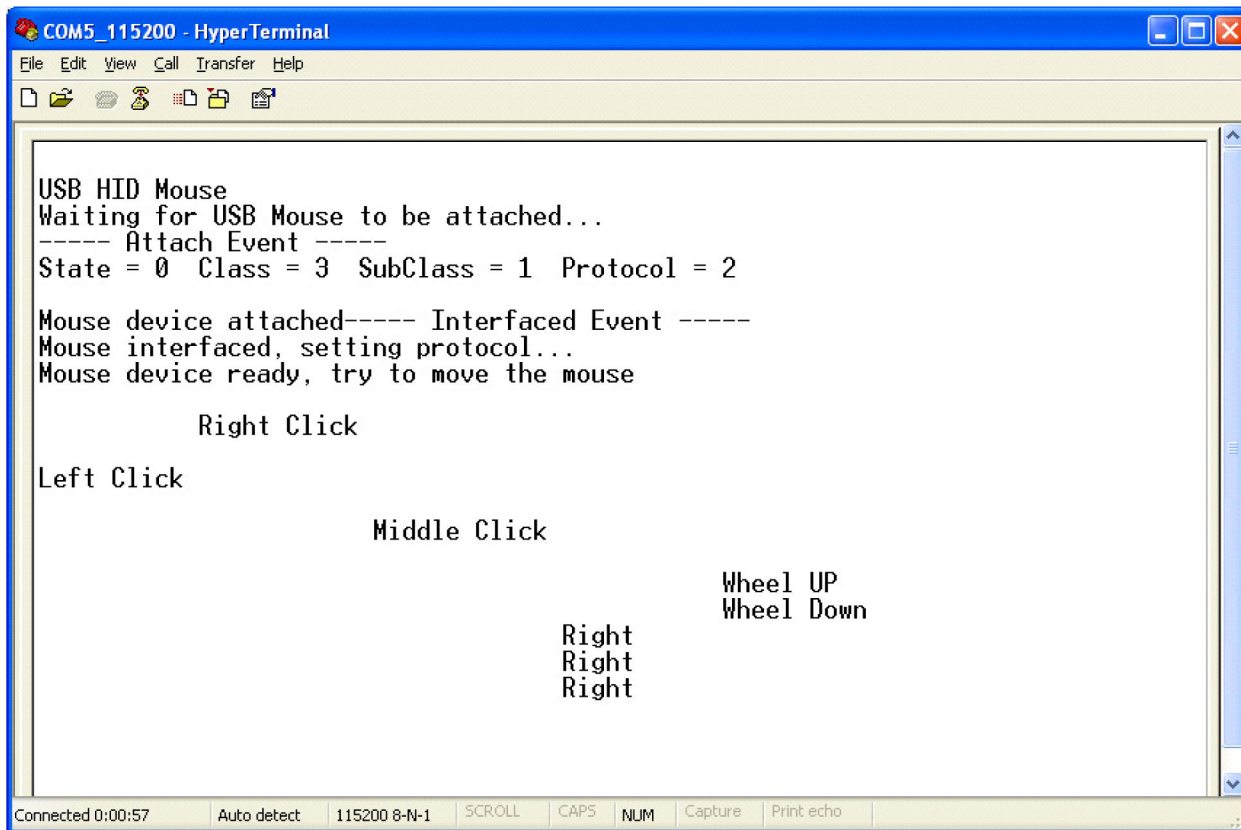
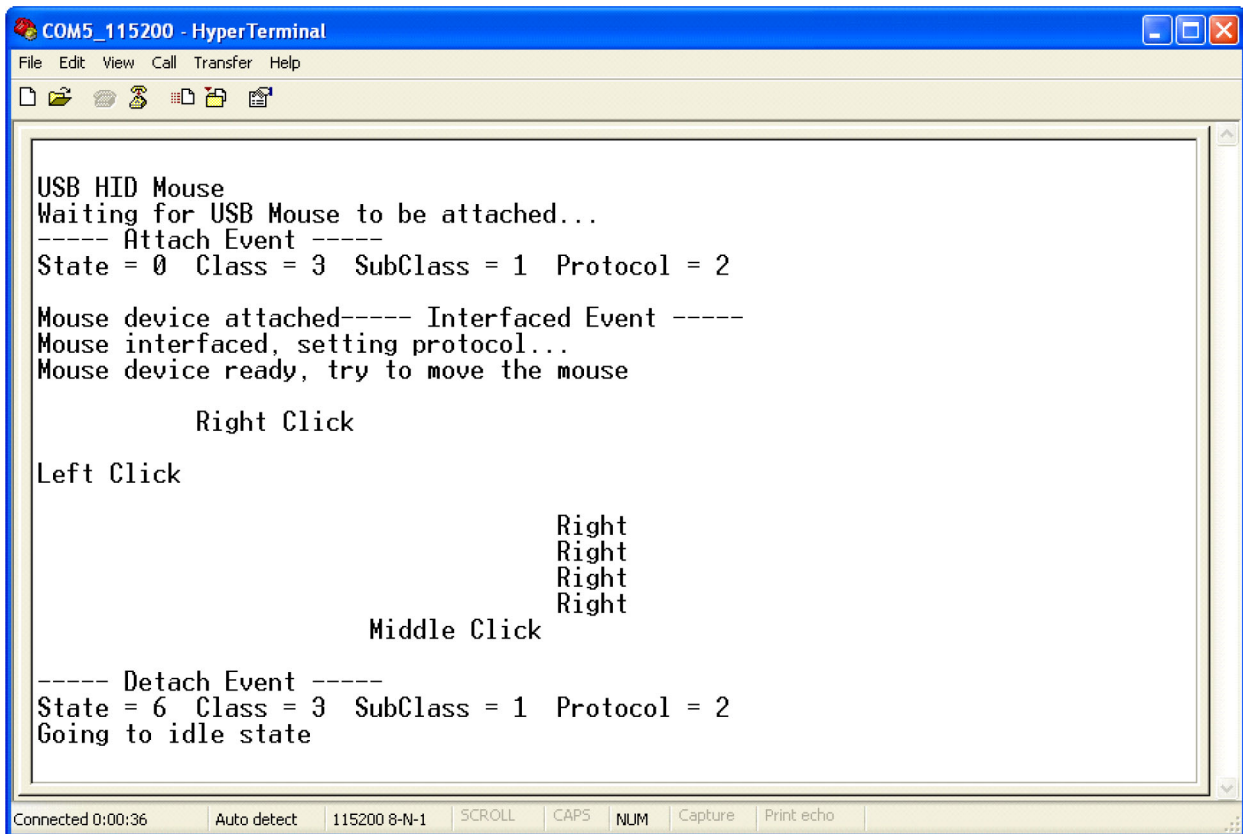


图 B-4. 来自鼠标的事件

5. 从演示板移除鼠标。Hyperterminal 会显示一条消息，如图 B-5 所示：



```
COM5_115200 - HyperTerminal
File Edit View Call Transfer Help
[Icons]
USB HID Mouse
Waiting for USB Mouse to be attached...
----- Attach Event -----
State = 0 Class = 3 SubClass = 1 Protocol = 2

Mouse device attached----- Interfaced Event -----
Mouse interfaced, setting protocol...
Mouse device ready, try to move the mouse

        Right Click

Left Click

                                Right
                                Right
                                Right
                                Right

                    Middle Click

----- Detach Event -----
State = 6 Class = 3 SubClass = 1 Protocol = 2
Going to idle state

Connected 0:00:36 | Auto detect | 115200 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo
```

图 B-5. 鼠标已移除

B.2.2 键盘例子

根据以下步骤运行键盘例子：

1. 打开并加载镜像到演示板。
2. 运行例子。首先，USB 主机等待设备接入事件。HyperTerminal 会显示一条消息，如图 B-6 所示：

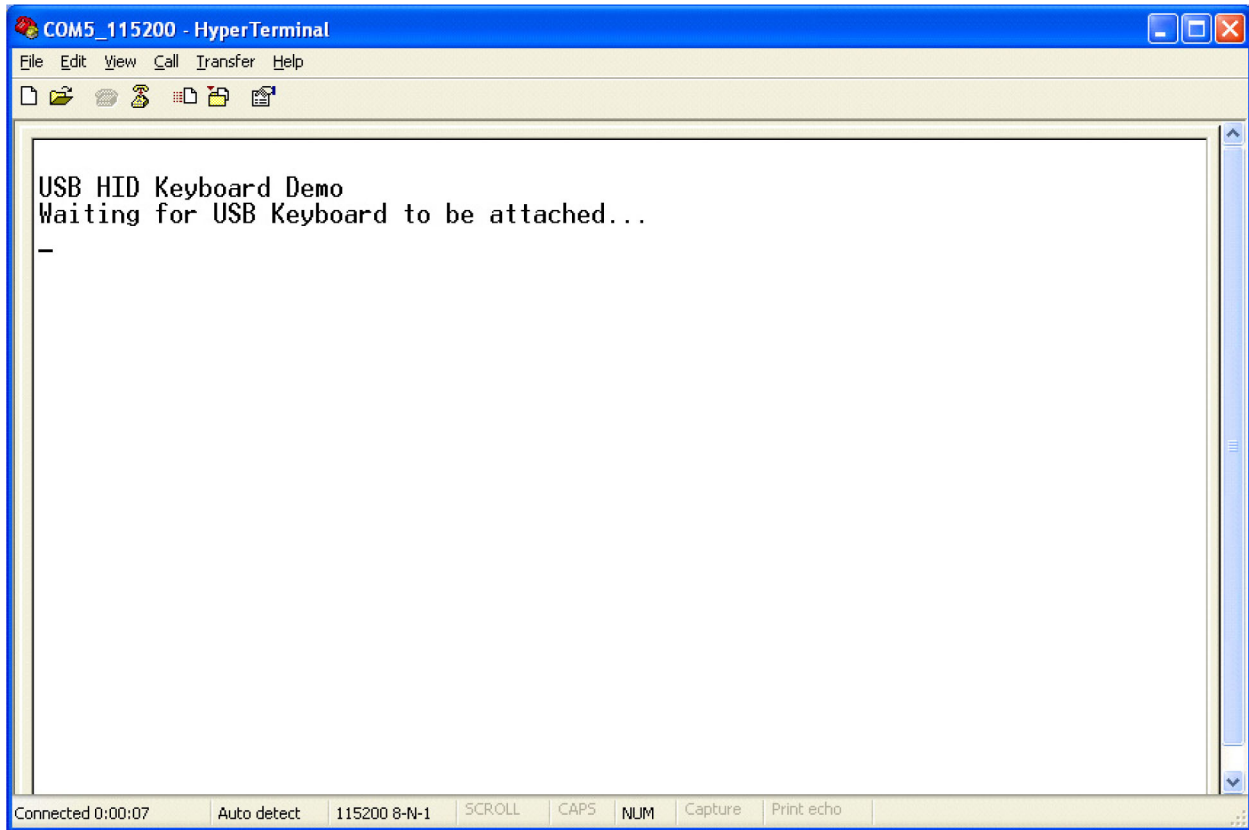
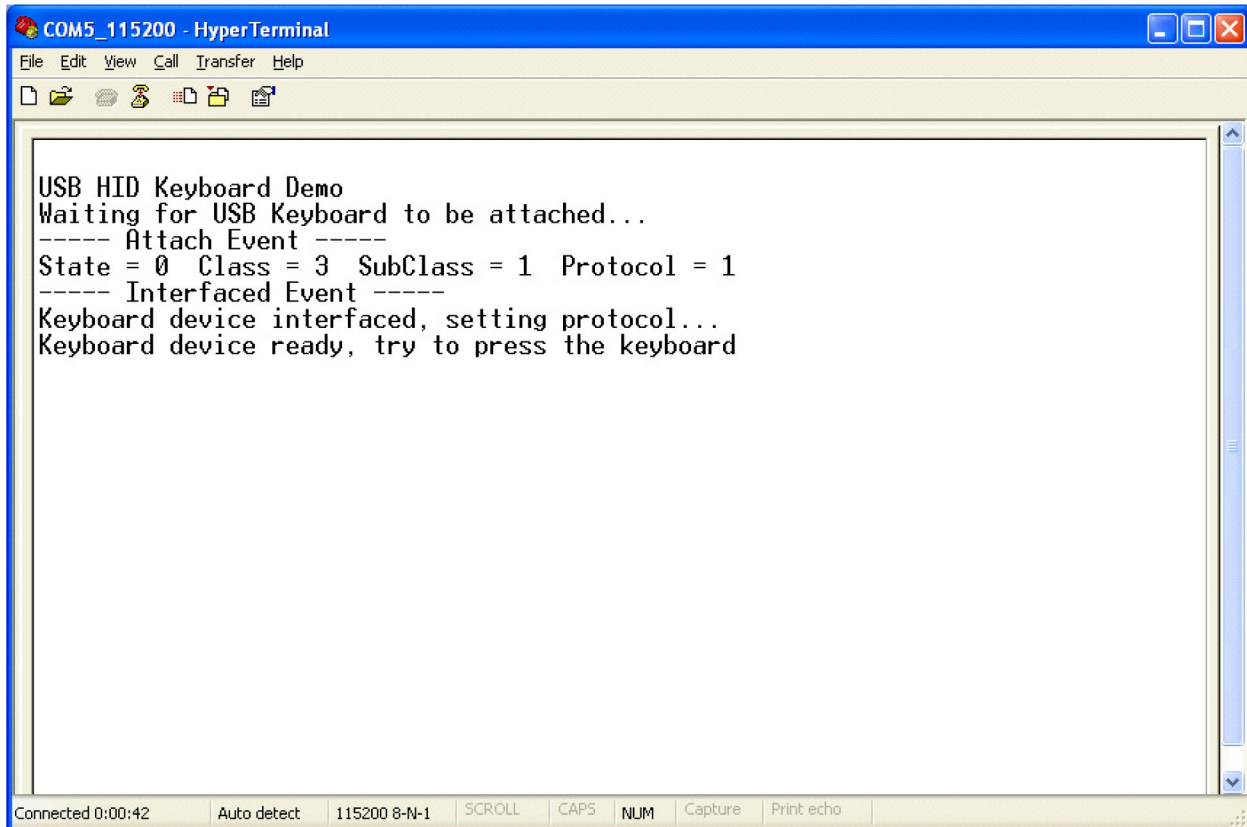


图 B-6. USB 主机正在等待键盘接入事件

3. 接入键盘。HyperTerminal 会显示一条消息，如图 B-7 所示：



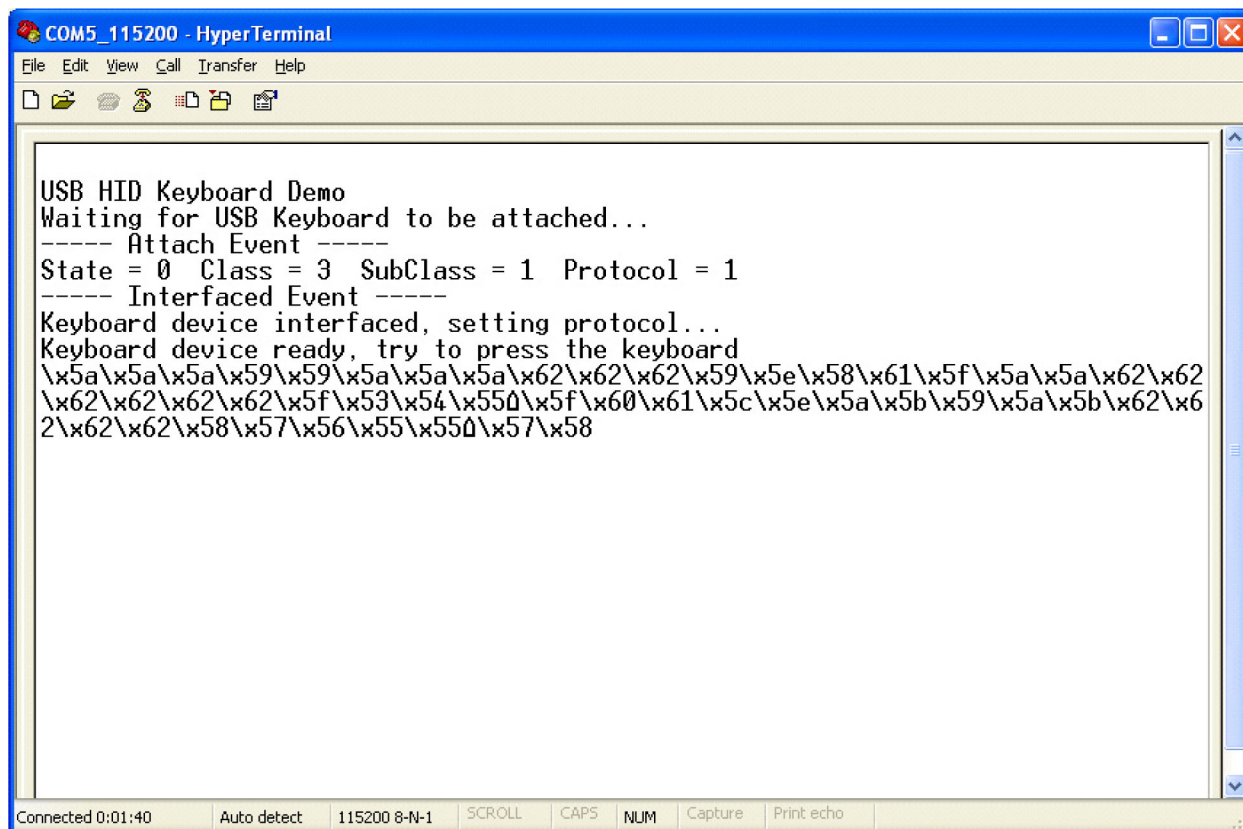
The screenshot shows a HyperTerminal window titled "COM5_115200 - HyperTerminal". The window contains the following text output:

```
USB HID Keyboard Demo
Waiting for USB Keyboard to be attached...
----- Attach Event -----
State = 0 Class = 3 SubClass = 1 Protocol = 1
----- Interfaced Event -----
Keyboard device interfaced, setting protocol...
Keyboard device ready, try to press the keyboard
```

The status bar at the bottom of the window shows "Connected 0:00:42", "Auto detect", "115200 8-N-1", "SCROLL", "CAPS", "NUM", "Capture", and "Print echo".

图 B-7. 键盘已接入

4. 输入一些字符。HyperTerminal 中将会显示这些字符的十六进制形式，如图 B-8 所示：



The screenshot shows a HyperTerminal window titled "COM5_115200 - HyperTerminal". The window contains the following text:

```
USB HID Keyboard Demo
Waiting for USB Keyboard to be attached...
----- Attach Event -----
State = 0 Class = 3 SubClass = 1 Protocol = 1
----- Interfaced Event -----
Keyboard device interfaced, setting protocol...
Keyboard device ready, try to press the keyboard
\x5a\x5a\x5a\x59\x59\x5a\x5a\x5a\x62\x62\x62\x59\x5e\x58\x61\x5f\x5a\x5a\x62\x62
\x62\x62\x62\x62\x5f\x53\x54\x55\x5f\x60\x61\x5c\x5e\x5a\x5b\x59\x5a\x5b\x62\x6
2\x62\x62\x58\x57\x56\x55\x55\x57\x58
```

The status bar at the bottom of the window shows: "Connected 0:01:40", "Auto detect", "115200 8-N-1", "SCROLL", "CAPS", "NUM", "Capture", and "Print echo".

图 B-8. 从键盘输入字符

注

HyperTerminal 仅显示每个字符的十六进制形式 ASCII 代码。

5. 移除键盘。HyperTerminal 会显示一条消息，如图 B-9 所示：

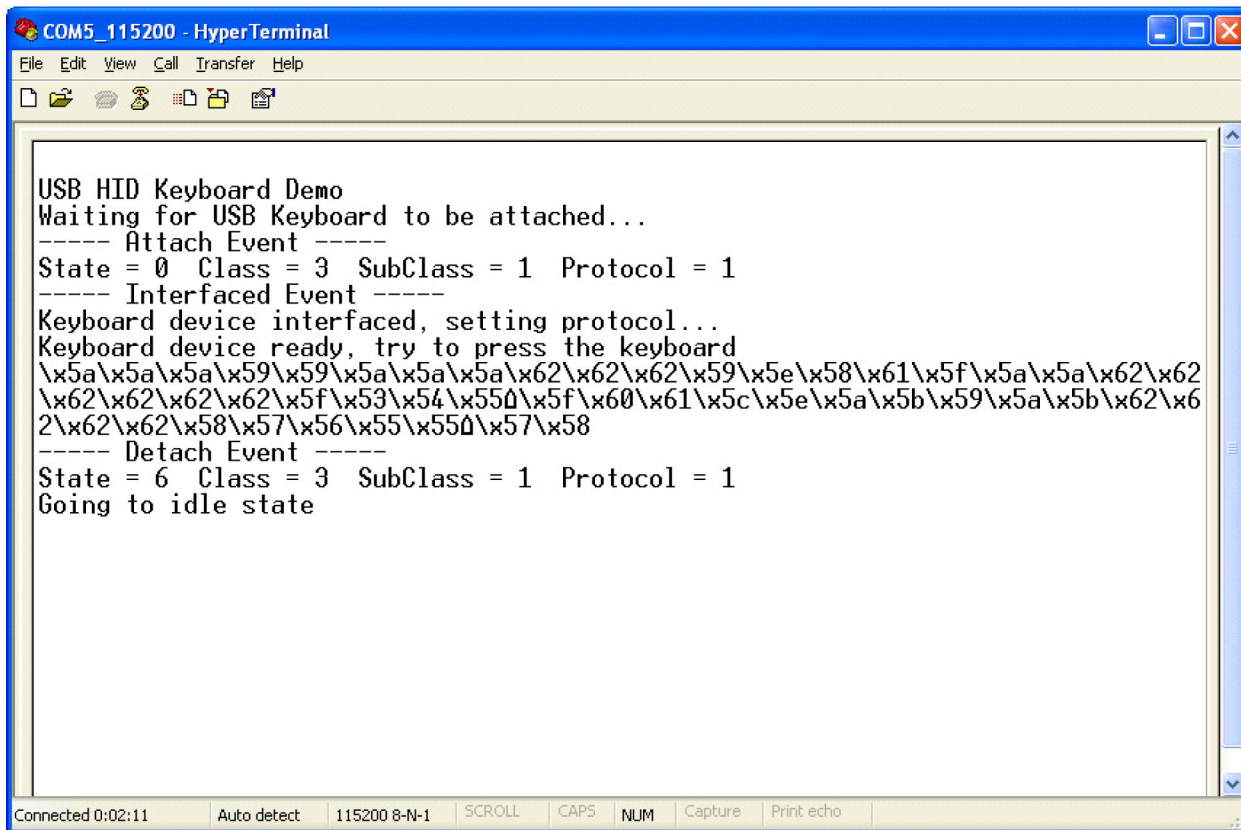


图 B-9. 键盘已移除

B.2.3 鼠标和键盘演示

该应用程序是上述两个程序的结合。它为用户切换所要使用的 HID 设备（鼠标或键盘）提供了一个便捷的选择。

附录 C

大容量存储设备（MSD）例子

C.1 设置例子

按照附录 B“人机接口设备（HID）例子”中的描述设置系统。

C.2 运行例子

根据以下步骤运行例子：

1. 打开 MSD 例子工程，并将镜像加载到演示板上。

```
<install_dir>\usb\host\examples\msd\msd_commands
```

2. 运行例子。HyperTerminal 会显示一条消息，如图 C-1 所示。

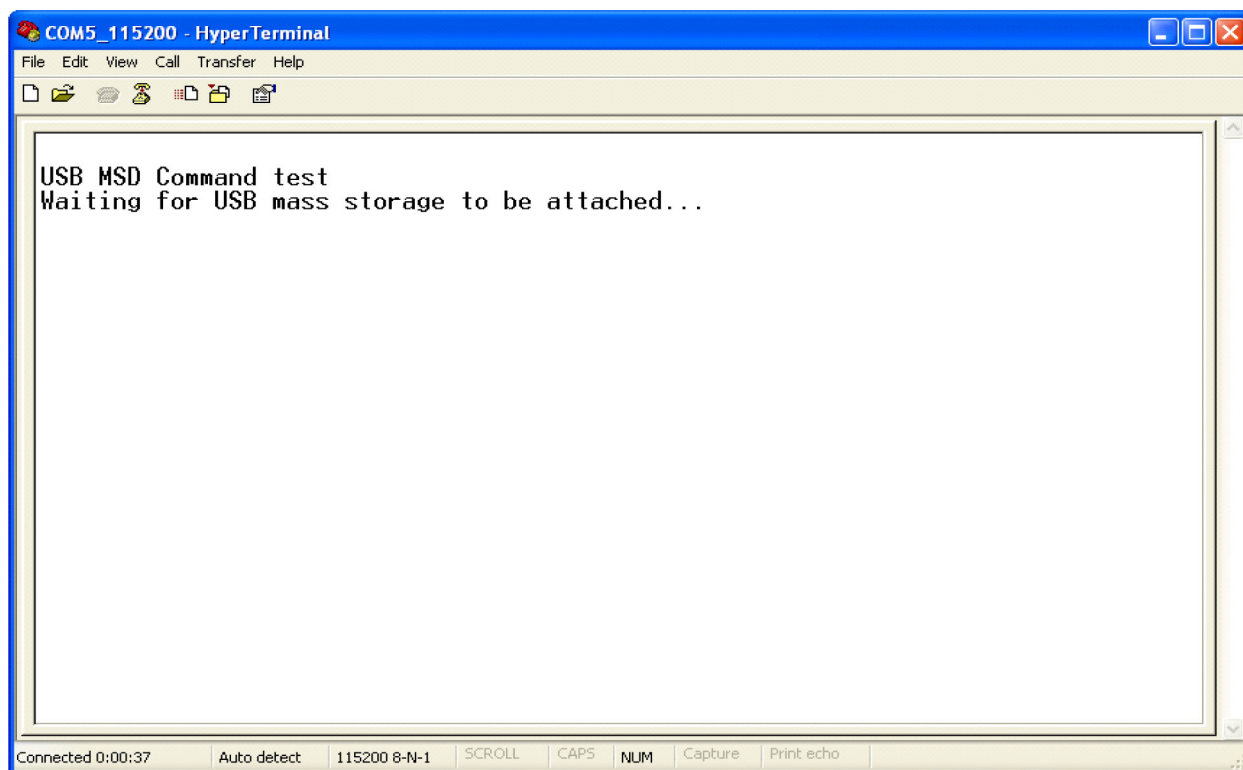
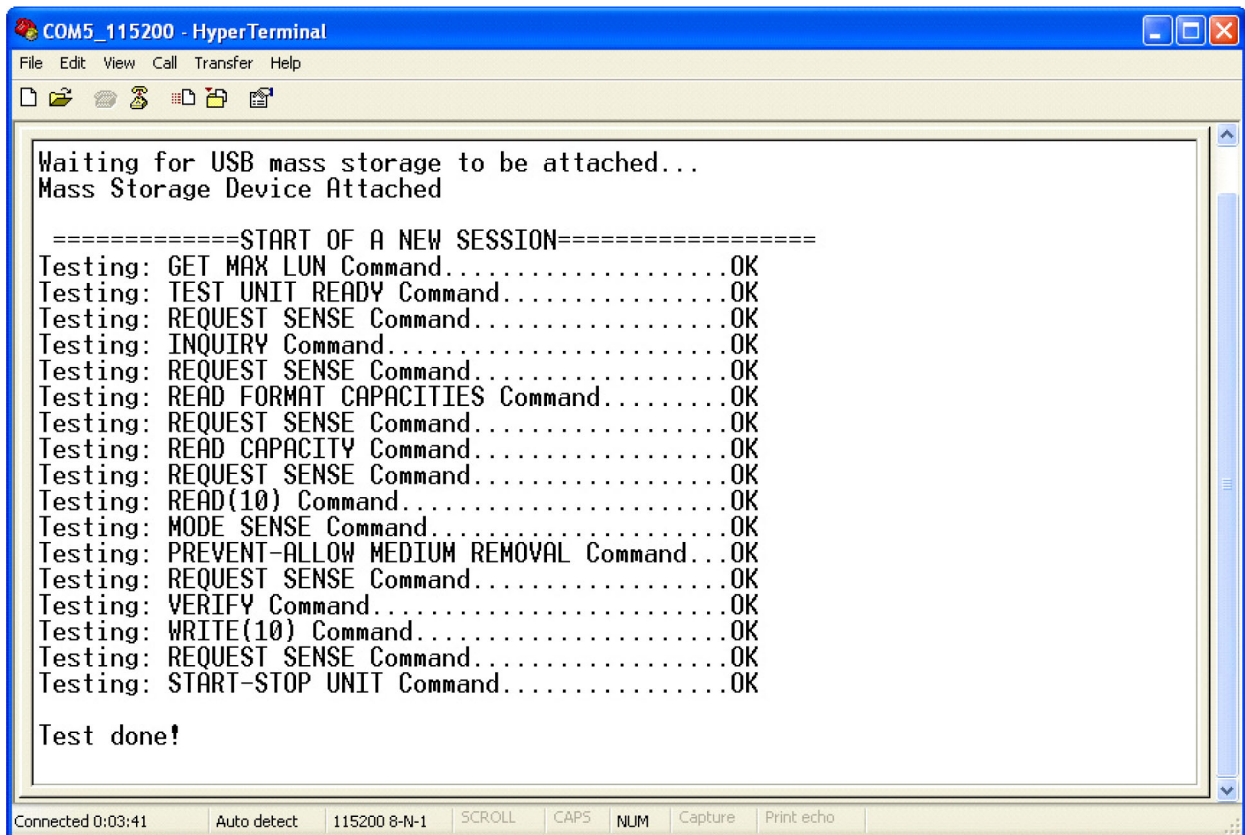


图 C-1. USB 主机等待 USB 大容量存储设备接入

3. 将 USB 大容量存储设备接入该演示板。HyperTerminal 会显示测试结果，如图 C-2 所示。



```
COM5_115200 - HyperTerminal
File Edit View Call Transfer Help
Waiting for USB mass storage to be attached...
Mass Storage Device Attached

=====START OF A NEW SESSION=====
Testing: GET MAX LUN Command.....OK
Testing: TEST UNIT READY Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: INQUIRY Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: READ FORMAT CAPACITIES Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: READ CAPACITY Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: READ(10) Command.....OK
Testing: MODE SENSE Command.....OK
Testing: PREVENT-ALLOW MEDIUM REMOVAL Command...OK
Testing: REQUEST SENSE Command.....OK
Testing: VERIFY Command.....OK
Testing: WRITE(10) Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: START-STOP UNIT Command.....OK

Test done!

Connected 0:03:41 Auto detect 115200 8-N-1 SCROLL CAPS NUM Capture Print echo
```

图 C-2. 测试结果

消息显示：所有测试项目均通过。

4. 移除设备。HyperTerminal 会显示设备已移除的消息，如图 C-3 所示。

```

COM5_115200 - HyperTerminal
File Edit View Call Transfer Help
=====START OF A NEW SESSION=====
Testing: GET MAX LUN Command.....OK
Testing: TEST UNIT READY Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: INQUIRY Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: READ FORMAT CAPACITIES Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: READ CAPACITY Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: READ(10) Command.....OK
Testing: MODE SENSE Command.....OK
Testing: PREVENT-ALLOW MEDIUM REMOVAL Command..OK
Testing: REQUEST SENSE Command.....OK
Testing: VERIFY Command.....OK
Testing: WRITE(10) Command.....OK
Testing: REQUEST SENSE Command.....OK
Testing: START-STOP UNIT Command.....OK

Test done!

Mass Storage Device Detached

Connected 0:07:27 Auto detect 115200 8-N-1 SCROLL CAPS NUM Capture Print echo

```

图 C-3. USB 大容量存储设备已移除

附录 D 虚拟通信 (COM) 例子

D.1 简介

USB 转串行端口的例子实现了 USB CDC 设备类中的抽象控制模型 (ACM) 子类, 使主机 PC 上的串行端口应用程序可通过 USB 端口收发串行数据。

D.2 设置例子

按照[附录 B“人机接口设备 \(HID\) 例子”](#)中的描述设置系统。

注

若要运行该例子, 需要一个 CDC 设备。

在该例子中, 从键盘输入的数据回显到 HyperTerminal 中。

CDC 例子中的数据流如[图 D-1](#)所示。

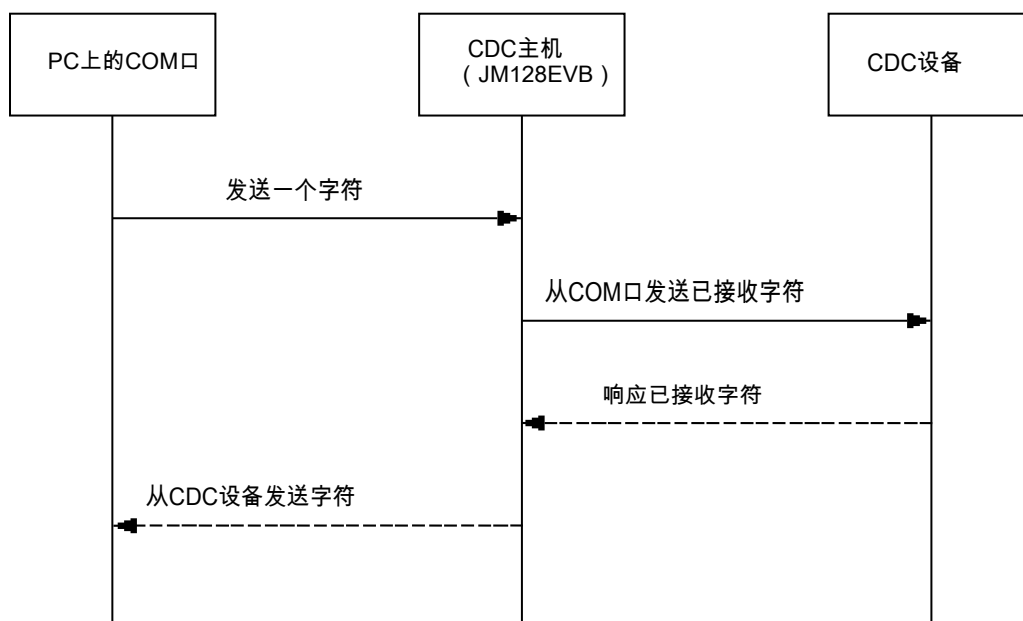


图 D-1. 数据流

D.3 运行例子

根据以下步骤运行 CDC 例子：

1. 打开 CDC 例子工程，并将镜像加载到演示板上。

CDC 应用程序工程位于：

```
<install_dir>\usb\host\examples\cdc\cdc_serial\
```

2. 利用附录 A.1.4“设置 HyperTerminal 以获取日志”中所示的步骤，将演示板的 COM1 与 PC 相连接。
3. 运行例子。HyperTerminal 会显示一条消息，如图 D-2 所示。

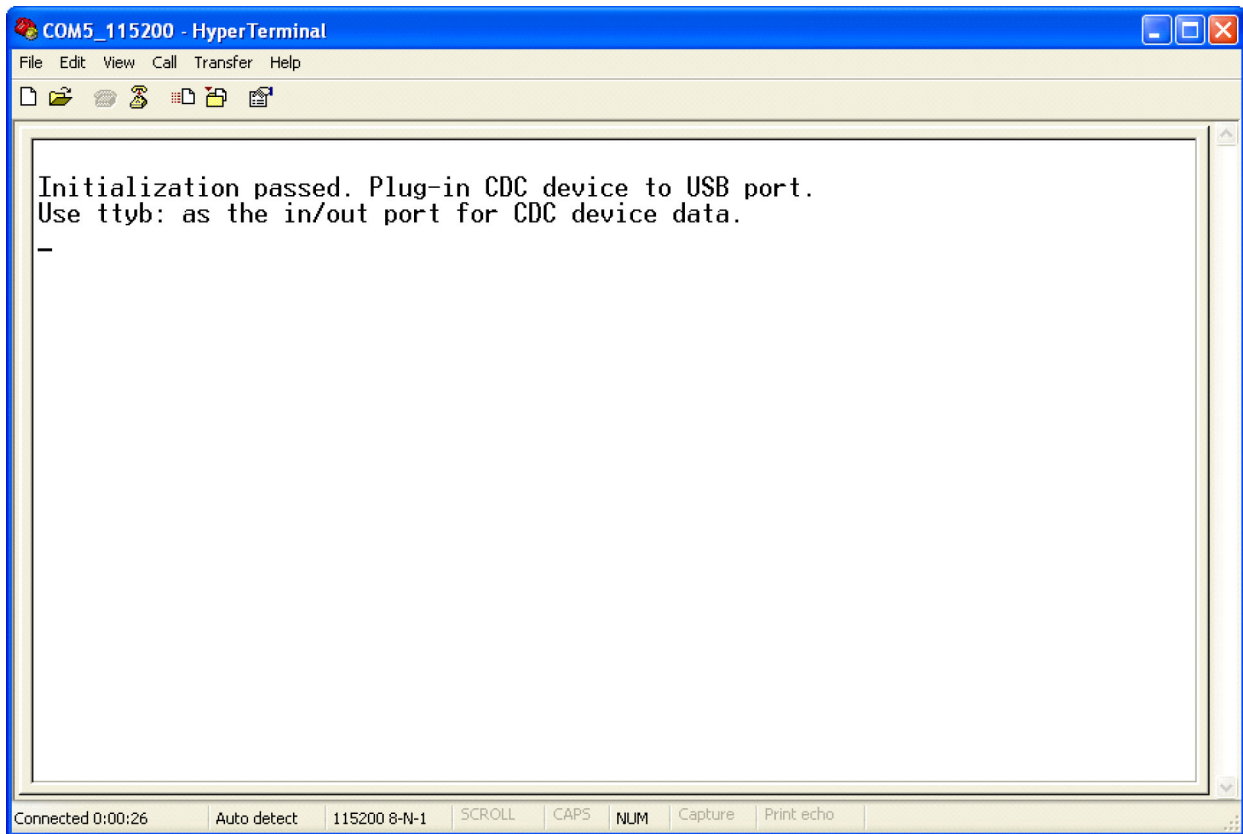
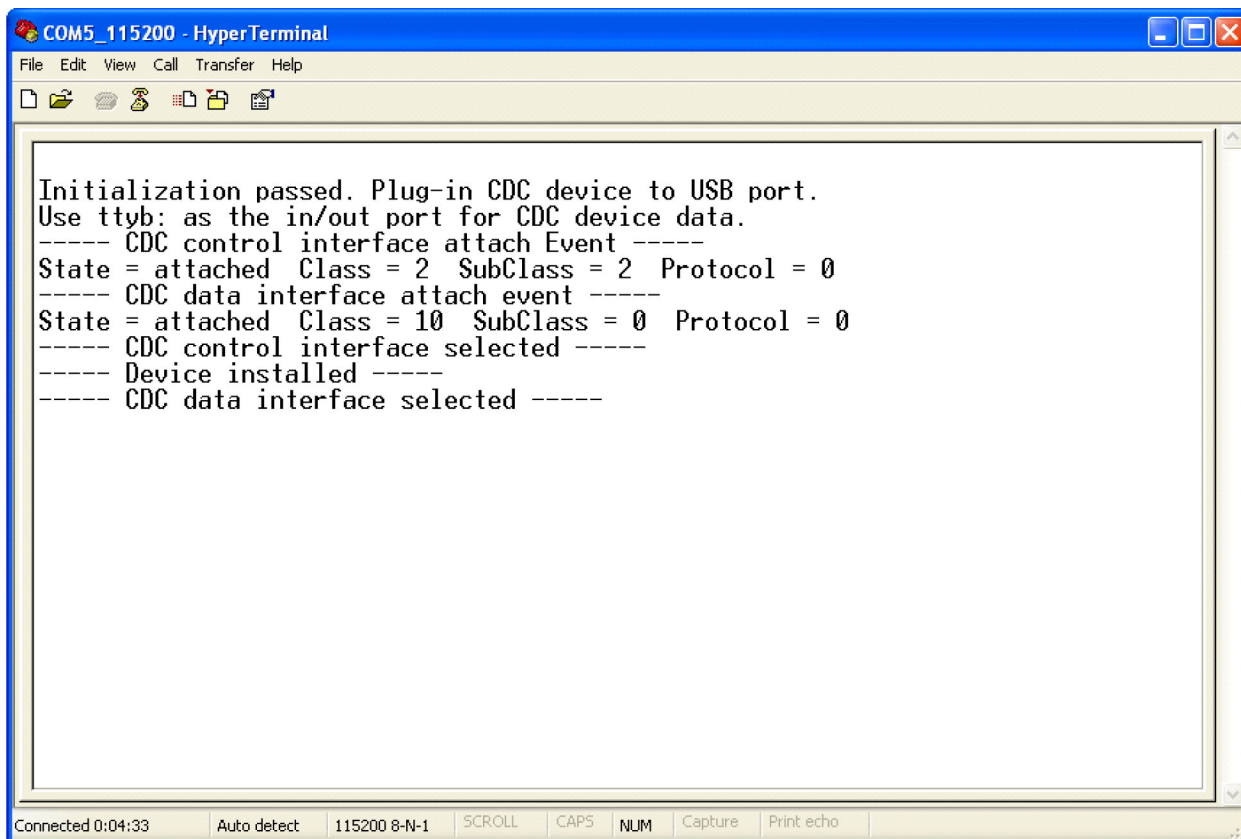


图 D-2. USB 主机等待 CDC 设备接入

- 接入 CDC 设备到 CDC 主机 (板)。HyperTerminal 会显示一条消息, 如图 D-3 所示。



The screenshot shows a HyperTerminal window titled "COM5_115200 - HyperTerminal". The window contains the following text:

```
Initialization passed. Plug-in CDC device to USB port.  
Use ttyb: as the in/out port for CDC device data.  
----- CDC control interface attach Event -----  
State = attached Class = 2 SubClass = 2 Protocol = 0  
----- CDC data interface attach event -----  
State = attached Class = 10 SubClass = 0 Protocol = 0  
----- CDC control interface selected -----  
----- Device installed -----  
----- CDC data interface selected -----
```

The status bar at the bottom of the window shows: "Connected 0:04:33", "Auto detect", "115200 8-N-1", "SCROLL", "CAPS", "NUM", "Capture", and "Print echo".

图 D-3. 设备信息

5. 从 PC 断开 COM1 的连接。连接 COM2 到 PC，并尝试键入内容。结果回显到 HyperTerminal 中。

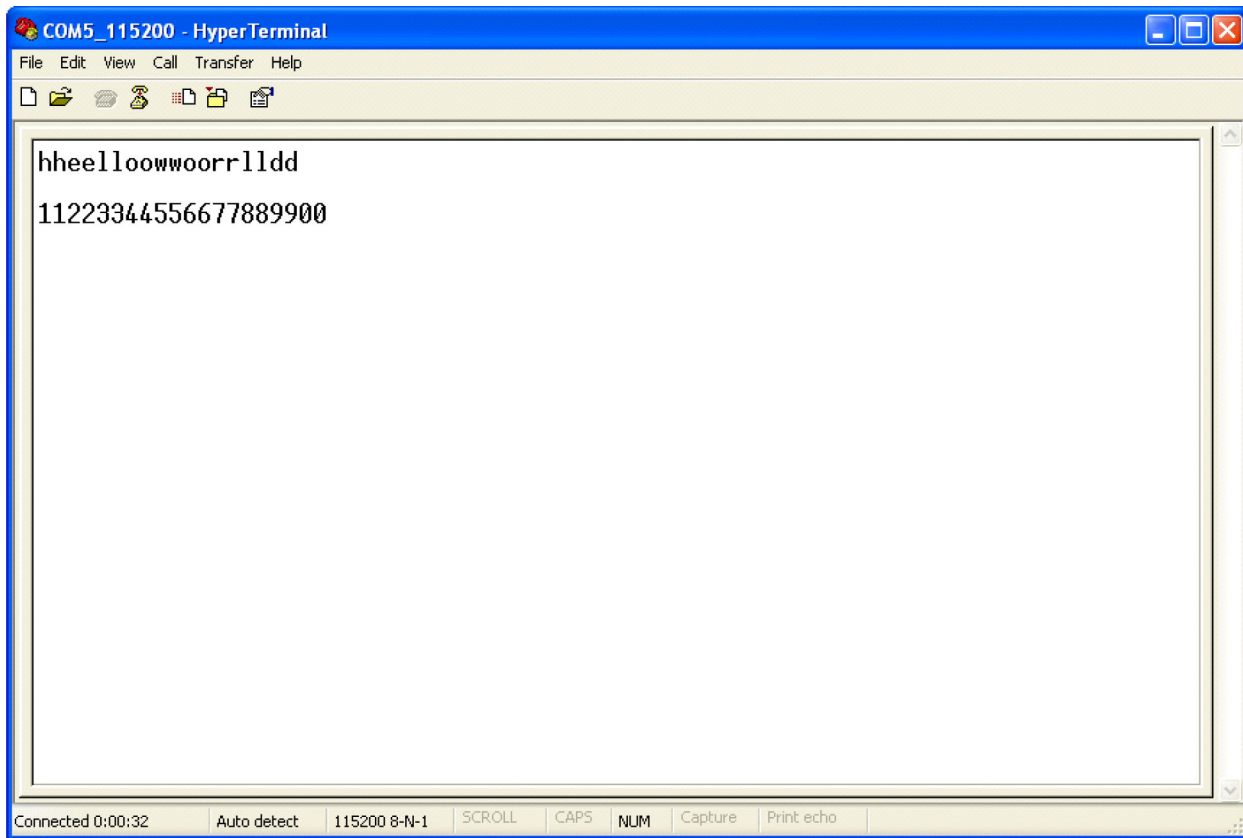


图 D-4. 字符回显于 HyperTerminal 中

6. 移除 CDC 设备。HyperTerminal 中的内容如图 D-5 所示。

```

COM5_115200 - HyperTerminal
File Edit View Call Transfer Help
-----
Initialization passed. Plug-in CDC device to USB port.
Use ttyb: as the in/out port for CDC device data.
----- CDC control interface attach Event -----
State = attached Class = 2 SubClass = 2 Protocol = 0
----- CDC data interface attach event -----
State = attached Class = 10 SubClass = 0 Protocol = 0
----- CDC control interface selected -----
----- Device installed -----
----- CDC data interface selected -----
----- CDC control interface detach event -----
State = detached Class = 2 SubClass = 2 Protocol = 0
-----
Connected 0:01:00 Auto detect 115200 8-N-1 SCROLL CAPS NUM Capture Print echo

```

图 D-5. CDC 设备已移除

附录 E 音频主机例子

E.1 简介

本章节是关于使用 USB 音频主机例子软件包的快速指南。例子应用程序用于控制音频设备并与其进行通信。例子的操作取决于音频设备类型：

- 扬声器类型（支持流输出的音频设备）：发送音频数据流到设备。
- 麦克风类型（支持流输入的音频设备）：接收来自设备的音频流数据并播放。

在这两种情况下，应用程序均支持向音频设备发送指定请求，如静音控制。本指南中通过使用 MCF52259 演示板对例子进行说明。

注

音频主机例子支持等时通道上的音频数据发送接口或音频数据接收接口。如果音频设备支持多数据接口，则仅支持最后的音频数据接口。

E.2 设置例子

设置如[图 E-1](#) 中所示的连接。

1. 在 PC（已安装软件）和演示板（已安装芯片）之间建立第一个 USB 连接。要求该连接能为演示板供电，并能将镜像下载到闪存中。
2. 在演示板和 PC 之间建立第二个连接，用于显示例子日志。
3. 在音频设备和演示板之间建立第三个连接。
4. 在扬声器和音频设备之间建立第四个连接。
5. 如果音频设备为麦克风，则将扬声器连接到音频主机，而不是音频设备。

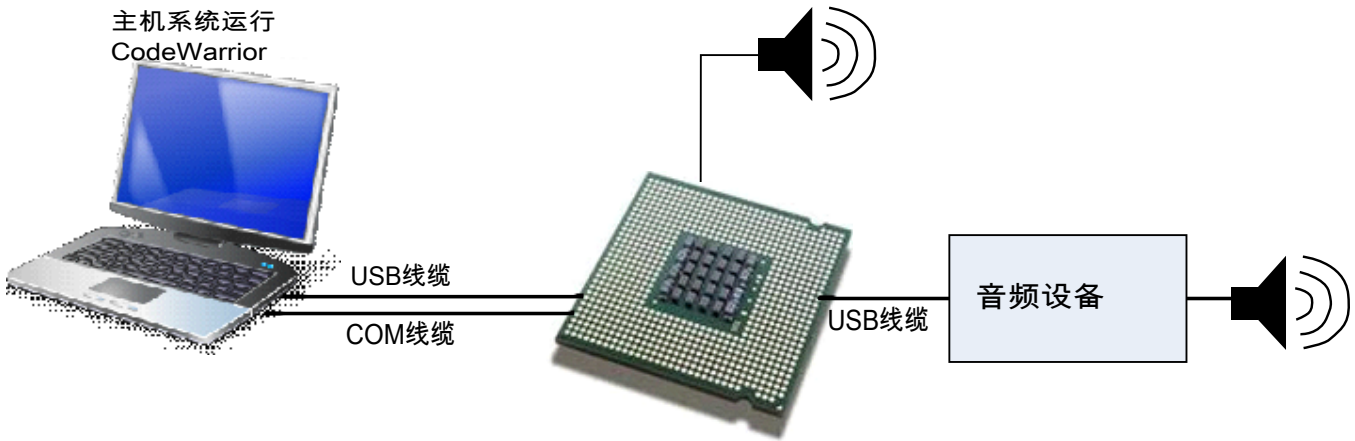


图 E-1. 音频例子设置

E.3 运行例子

执行以下步骤来运行音频主机例子：

1. 打开音频例子应用程序的镜像并加载到演示板上。
2. 在镜像加载成功后，HyperTerminal 中的内容将如图 E-2 所示。

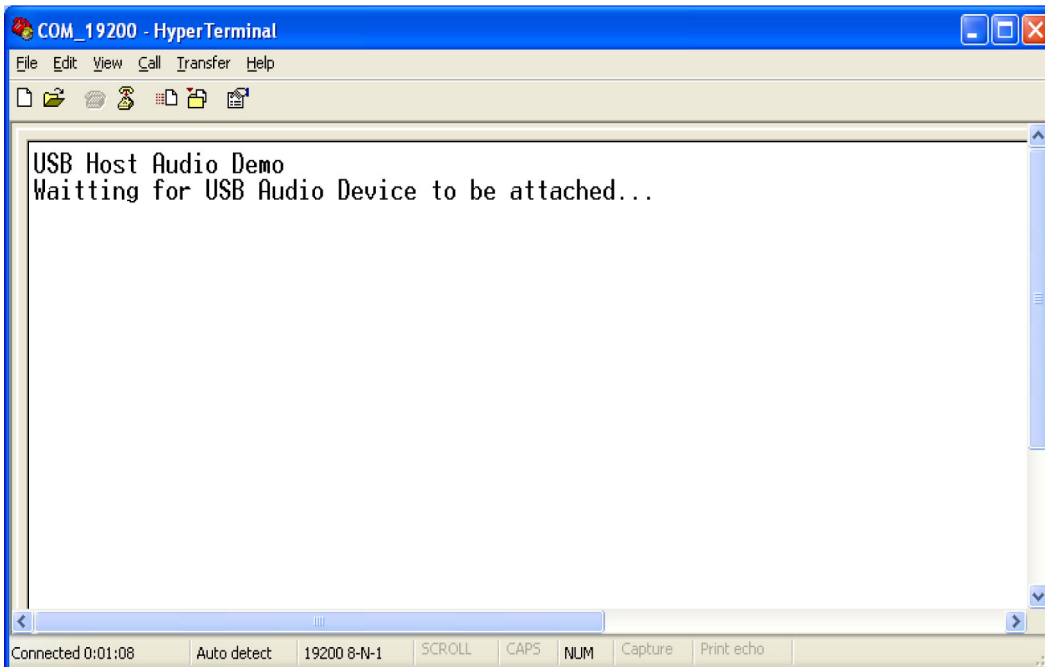


图 E-2. USB 主机等待音频设备接入事件

3. 将音频设备插接到演示板。演示板将接入音频设备。如果音频设备为扬声器类型，则设备信息显示为图 E-3 所示。如果音频设备为麦克风类型，见图 E-4。

```

COM_19200 - HyperTerminal
File Edit View Call Transfer Help
USB Host Audio Demo
Waiting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached Class = 1 SubClass = 1 Protocol = 0
----- Audio stream interface: attach event -----
State = attached Class = 1 SubClass = 2 Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: interface event-----

Audio device information:
- Device type      : Speaker
- Frequency       : 8 KHz
- Bit resolution  : 8 bit
-
Connected 0:00:59  Auto detect  19200 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
    
```

图 E-3. 接入设备为扬声器类型

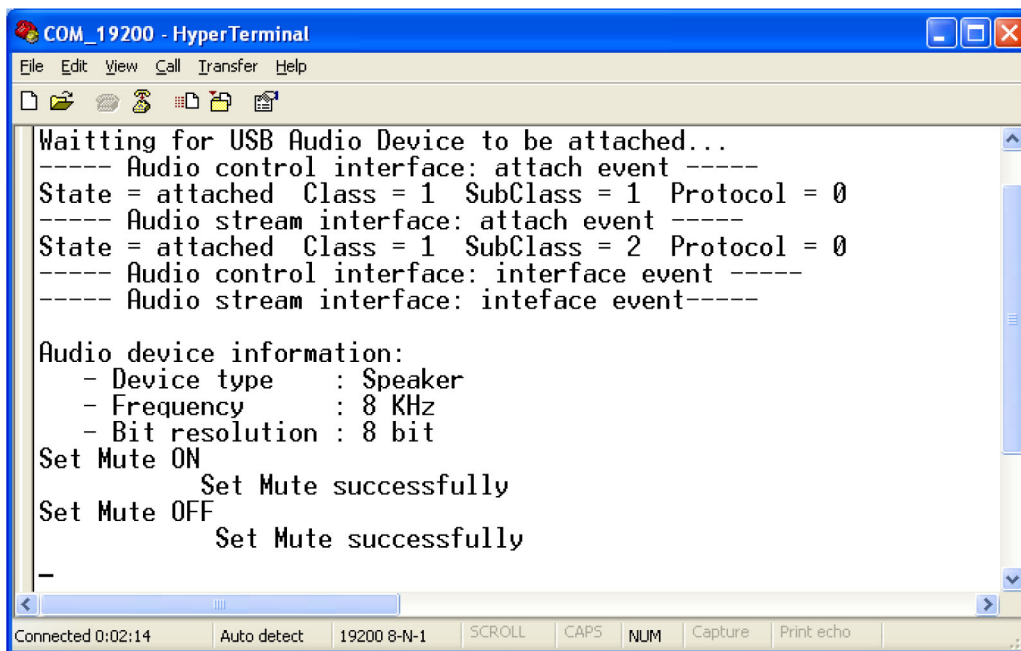
```

COM_19200 - HyperTerminal
File Edit View Call Transfer Help
USB Host Audio Demo
Waiting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached Class = 1 SubClass = 1 Protocol = 0
----- Audio stream interface: attach event -----
State = attached Class = 1 SubClass = 2 Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: interface event-----

Audio device information:
- Device type      : Microphone
- Frequency       : 8 KHz
- Bit resolution  : 8 bit
-
Connected 0:00:51  Auto detect  19200 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
    
```

图 E-4. 接入设备为麦克风类型

4. 按下开关 1 以设置静音开/关。HyperTerminal 窗口中的内容如图 E-5 所示。



```
COM_19200 - HyperTerminal
File Edit View Call Transfer Help
Waiting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached Class = 1 SubClass = 1 Protocol = 0
----- Audio stream interface: attach event -----
State = attached Class = 1 SubClass = 2 Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: interface event-----

Audio device information:
- Device type      : Speaker
- Frequency       : 8 KHz
- Bit resolution  : 8 bit
Set Mute ON
      Set Mute successfully
Set Mute OFF
      Set Mute successfully
-
Connected 0:02:14  Auto detect  19200 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
```

图 E-5. 设置静音开/关。

5. 按下开关 2 以开始/停止音频主机和音频设备之间的音频数据流传输。
 - 如果接入设备为扬声器类型, 您将可以从音频设备所连接的扬声器中听到声音播放。
 - 如果接入设备为麦克风类型, 您将可以从音频主机所连接的扬声器中听到声音播放。

HyperTerminal 窗口中的内容如图 E-6 所示。

```

COM_19200 - HyperTerminal
File Edit View Call Transfer Help
USB Host Audio Demo
Waitting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached Class = 1 SubClass = 1 Protocol = 0
----- Audio stream interface: attach event -----
State = attached Class = 1 SubClass = 2 Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: inteface event-----

Audio device information:
- Device type      : Speaker
- Frequency        : 8 KHz
- Bit resolution   : 8 bit
Set Mute ON
      Set Mute successfully
Set Mute OFF
      Set Mute successfully
Playing ...

Paused.

```

图 E-6. 开始/停止传输音频数据

6. 移除音频设备。HyperTerminal 会显示一条消息，如图 E-7 所示。

```

COM_19200 - HyperTerminal
File Edit View Call Transfer Help
USB Host Audio Demo
Waitting for USB Audio Device to be attached...
----- Audio control interface: attach event -----
State = attached Class = 1 SubClass = 1 Protocol = 0
----- Audio stream interface: attach event -----
State = attached Class = 1 SubClass = 2 Protocol = 0
----- Audio control interface: interface event -----
----- Audio stream interface: inteface event-----

Audio device information:
- Device type      : Speaker
- Frequency        : 8 KHz
- Bit resolution   : 8 bit
Set Mute ON
      Set Mute successfully
Set Mute OFF
      Set Mute successfully
Playing ...

Paused.
----- Audio control interface: detach event -----
State = detached Class = 1 SubClass = 1 Protocol = 0

```

图 E-7. 音频设备已移除

附录 F

修订历史记录

F.1 修订历史记录

公司网站上提供的文档修订版为最新版本，供您获取最新信息。您的印刷副本可能是较早的版本。若要验证您是否拥有最新信息，请访问 freescale.com 并导航至 Design Resources>Software and Tools>AllSoftware and Tools>Freescale MQX Software Solutions。

以下修订历史记录表总结了本文档中所包含的更改。

修订编号	修订日期	变更说明
Rev. 1	01/2009	最初公开版本。
Rev. 2	04/2011	在文件中添加了关于 USB OTG 软件/示例在当前 MQX 版本中不可用的说明。
Rev. 3	12/2011	重新编写文档，增加了 5 个附录。
Rev. 4	06/2012	更新了 MQX 版本 3.8.1——见 4.3“USB HDK 在 MQX RTOS 3.8.1 中的修改”
Rev. 5	04/2013	微小修改。
Rev. 6	10/2013	更新了内容，以反映从 MQX 类型向 C99 类型的转变。
Rev. 7	08/2014	对版本 C 和版本 D 进行了微小修改。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不限于后果性的或附带性的损害在内的所有责任。

Freescale 的数据表和/或规格中所提供的“典型”参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。所有运行参数，包括“经典值”在内，必须经由客户的技术专家对每个客户的应用程序进行验证。

Freescale 未转让与其专利权及其他权利相关的许可。Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：freescale.com/SalesTermsandConditions。

Freescale, the Freescale logo, and Kinetis, are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2014 Freescale Semiconductor, Inc.

© 2014 飞思卡尔半导体有限公司