小川工作室编写，本书为 LM3S 的 USB 芯片编写，上传的均为草稿，还有没修改，可能还有很多地方不足，希望各位网友原谅！

QQ:2609828265

TEL:15882446438

E-mail:paulhyde@126.com

# 第七章 Bulk 设备

## 7.1 bulk 设备介绍

USB 通道的数据传输格式有两种，而且这两种格式还是互斥的。有消息和流两种对于流状态，不具有 usb 数据的格式，遵循的规则就是先进先出。对于消息通道，它的通信模式符合 usb 的数据格式，一般有三个阶段组成首先是建立阶段，数据阶段，确认阶段。所有的通信的开始都是由主机方面发起的。

USB 协议制定时，为了方便不同设备的开发商基于 USB 进行设计，定义了不同的设备类来支持不同类型的设备。Bulk 也是其中一种，Bulk 设备使用端点 Bulk 传输模式，可以快速传输大量数据。

批量传输(BULK)采用的是流状态传输,批量传送的一个特点就是支持不确定时间内进行大量数据传输，能够保证数据一定可以传输，但是不能保证传输的带宽和传输的延迟。而且批量传输是一种单向的传输，要进行双向传输必须要使用两个通道。本章将介绍双向 BULK 的批量传输。

## 7.2 bulk 数据类型

usbdbulk.h、usblib.h 中已经定义好 Bulk 设备类中使用的所有数据类型和函数，下面介绍 Bulk 设备类使用的数据类型。

```
typedef struct
{
    //定义本 Buffer 是用于传输还是接收，True 为接收，False 为发送。
    tBoolean bTransmitBuffer;
    //Callback 函数，用于 Buffer 数据处理完成后
    tUSBCallback pfnCallback;
    //Callback 第一个输入参数
    void *pvCBData;
    //数据传输或者接收时调用的函数，用于完成发送或者接收的函数。
    tUSBPacketTransfer pfnTransfer;
    //数据传输或者接收时调用的函数。
    //发送时，用于检查是否有足够空间；接收时，用于检查可以接收的数量。
    tUSBPacketAvailable pfnAvailable;
    //在设备模式下，设备类指针
    void *pvHandle;
    //用于存放发送或者接收的数据.
    unsigned char *pcBuffer;
    //发送或者接收数据大小
    unsigned long ulBufferSize;
    //RAM Buffer
    void *pvWorkspace;
}
tUSBBuffer;
```

tUSBBuffer，数据缓存控制结构体，定义在 usblib.h 中，用于在 Bulk 传输过程中，发送数据或者接收数据。是 Bulk 设备传输的主要载体，结构体内部包含数据发送、接收、处理函数等。

```
typedef enum
{
    //Bulk 状态没定义
    BULK_STATE_UNCONFIGURED,
    //空闲状态
    BULK_STATE_IDLE,
    //等待数据发送或者结束
    BULK_STATE_WAIT_DATA,
    //等待数据处理.
    BULK_STATE_WAIT_CLIENT
} tBulkState;
```

tBulkState，定义 Bulk 端点状态。定义在 usbdbulk.h。用于端点状态标记与控制，可以保证数据传输不相互冲突。

```
typedef struct
{
     //USB 基地址
    unsigned long ulUSBBase;
     //设备信息
    tDeviceInfo *psDevInfo;
     //配置信息
    tConfigDescriptor *psConfDescriptor;
     //Bulk 接收端点状态
    volatile tBulkState eBulkRxState;
     //Bulk 发送端点状态
    volatile tBulkState eBulkTxState;
     //标志位
    volatile unsigned short usDeferredOpFlags;
    //最后一次发送数据大小
    unsigned short usLastTxSize;
     //连接是否成功
    volatile tBoolean bConnected;
```

```
     //IN 端点号
    unsigned char ucINEndpoint;
     //OUT 端点号
    unsigned char ucOUTEndpoint;
     //接口号
    unsigned char ucInterface;
}
tBulkInstance;
```

tBulkInstance，Bulk 设备类实例。定义了 Bulk 设备类的 USB 基地址、设备信息、IN 端点、OUT 端点等信息。

```
typedef struct
{
    //VID
    unsigned short usVID;
    //PID
    unsigned short usPID;
    //最大耗电量
    unsigned short usMaxPowermA;
    //电源属性
    unsigned char ucPwrAttributes;
    //接收回调函数，主要用于接收数据处理
    tUSBCallback pfnRxCallback;
    //接收回调函数的第一个参数。
    void *pvRxCBData;
    //发送回调函数，主要用于发送数据处理
    tUSBCallback pfnTxCallback;
    //发送回调函数的第一个参数。
    void *pvTxCBData;
    //字符串描述符集合
    const unsigned char * const *ppStringDescriptors;
    //字符串描述符个数
    unsigned long ulNumStringDescriptors;
    //Bulk 设备实例
    tBulkInstance *psPrivateBulkData;
}
tUSBDBulkDevice;
```

tUSBDBulkDevice，Bulk 设备类，定义了 VID、PID、电源属性、字符串描述符等，还包括了一个 Bulk 设备类实例。其它设备描述符、配置信息通过 API 函数储入 tBulkInstance 定义的 Bulk 设备实例中。

## 7.3 API 函数

在 Bulk 设备类 API 库中定义了 11 个函数，完成 USB Bulk 设备初始化、配置及数据处理。以及 11 个 Buffer 操作函数，下面为 usbdbulk.h 中定义的 API 函数：

```
void *USBDBulkInit(unsigned long ulIndex,
                         const tUSBDBulkDevice *psDevice);
```

```
void *USBDBulkCompositeInit(unsigned long ulIndex,
                                   const tUSBDBulkDevice *psDevice);
unsigned long USBDBulkPacketWrite(void *pvInstance,
                                         unsigned char *pcData,
                                         unsigned long ulLength,
                                         tBoolean bLast);
unsigned long USBDBulkPacketRead(void *pvInstance,
                                        unsigned char *pcData,
                                        unsigned long ulLength,
                                        tBoolean bLast);
unsigned long USBDBulkTxPacketAvailable(void *pvInstance);
unsigned long USBDBulkRxPacketAvailable(void *pvInstance);
void USBDBulkTerm(void *pvInstance);
void *USBDBulkSetRxCBData(void *pvInstance, void *pvCBData);
void *USBDBulkSetTxCBData(void *pvInstance, void *pvCBData);
void USBDBulkPowerStatusSet(void *pvInstance, unsigned char ucPower);
tBoolean USBDBulkRemoteWakeupRequest(void *pvInstance);
```

```
void *USBDBulkInit(unsigned long ulIndex,
                          const tUSBDBulkDevice *psDevice);
```

作用：初始化 Bulk 设备硬件、协议，把其它配置参数填入 psDevice 实例中。

参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psDevice，Bulk 设备类。

返回：指向配置后的 tUSBDBulkDevice。

```
void *USBDBulkCompositeInit(unsigned long ulIndex,
                                    const tUSBDBulkDevice *psDevice);
```

作用：初始化 Bulk 设备协议，本函数在 USBDBulkInit 中已经调用。

参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psDevice，Bulk 设备类。

返回：指向配置后的 tUSBDBulkDevice。

```
unsigned long USBDBulkPacketWrite(void *pvInstance,
                                          unsigned char *pcData,
                                          unsigned long ulLength,
                                          tBoolean bLast);
```

作用：通过 Bulk 传输发送一个包数据，底层驱动，在 Buffer 中使用。

参数：pvInstance，tUSBDBulkDevice 设备指针。pcData，待写入的数据指针。ulLength，待写入数据的长度。bLast，是否传输结束包。

返回：成功发送长度，可能与 ulLength 长度不一样。

```
unsigned long USBDBulkPacketRead(void *pvInstance,
                                         unsigned char *pcData,
                                         unsigned long ulLength,
                                         tBoolean bLast);
```

作用：通过 Bulk 传输接收一个包数据，底层驱动，在 Buffer 中使用。

参数：pvInstance，tUSBDBulkDevice 设备指针。pcData，读出数据指针。ulLength，读出数据的长度。bLast，是否是束包。

返回：成功接收长度，可能与 ulLength 长度不一样。

unsigned long USBDBulkTxPacketAvailable(void *pvInstance);

作用：获取可用发送数据长度。

参数：pvInstance，tUSBDBulkDevice 设备指针。

返回：发送包大小，用发送数据长度。

unsigned long USBDBulkRxPacketAvailable(void *pvInstance);

作用：获取接收数据长度。

参数：pvInstance，tUSBDBulkDevice 设备指针。

返回：可用接收数据个数，可读取的有效数据。

void USBDBulkTerm(void *pvInstance);

作用：结束 Bulk 设备。

参数：pvInstance，指向 tUSBDBulkDevice。

返回：无。

void *USBDBulkSetRxCBData(void *pvInstance, void *pvCBData);

作用：改变接收回调函数的第一个参数。

参数：pvInstance，指向 tUSBDBulkDevice。pvCBData，用于替换的参数

返回：旧参数指针。

void *USBDBulkSetTxCBData(void *pvInstance, void *pvCBData);

作用：改变发送回调函数的第一个参数。

参数：pvInstance，指向 tUSBDBulkDevice。pvCBData，用于替换的参数

返回：旧参数指针。

void USBDBulkPowerStatusSet(void *pvInstance, unsigned char ucPower);

作用：修改电源属性、状态。

参数：pvInstance，指向 tUSBDBulkDevice。ucPower，电源属性。

返回：无。

tBoolean USBDBulkRemoteWakeupRequest(void *pvInstance);

作用：唤醒请求。

参数：pvInstance，指向 tUSBDBulkDevice。

返回：无。

在这些函数中 USBDBulkInit 和 USBDBulkPacketWrite、USBDBulkPacketRead、USBDBulkTxPacketAvailable、USBDBulkRxPacketAvailable 函数最重要并且使用最多，USBDBulkInit 第一次使用 Bulk 设备时，用于初始化 Bulk 设备的配置与控制。USBDBulkPacketRead 、 USBDBulkPacketWrite 、 USBDBulkTxPacketAvailable 、USBDBulkRxPacketAvailable 为 Bulk 传输数据的底层驱动函数用于驱动 Buffer。

usblib.h 中定义为 11 个 Buffer 操作函数，用于数据发送、接收、以及回调其它函数，下面介绍 11 个 Buffer 操作函数：

```
const tUSBBuffer *USBBufferInit(const tUSBBuffer *psBuffer);
void USBBufferInfoGet(const tUSBBuffer *psBuffer,
                      tUSBRingBufObject *psRingBuf);
unsigned long USBBufferWrite(const tUSBBuffer *psBuffer,
                             const unsigned char *pucData,
                             unsigned long ulLength);
void USBBufferDataWritten(const tUSBBuffer *psBuffer,
                          unsigned long ulLength);
```

```
void USBBufferDataRemoved(const tUSBBuffer *psBuffer,
                                 unsigned long ulLength);
void USBBufferFlush(const tUSBBuffer *psBuffer);

unsigned long USBBufferRead(const tUSBBuffer *psBuffer,
                                 unsigned char *pucData,
                                 unsigned long ulLength);
unsigned long USBBufferDataAvailable(const tUSBBuffer *psBuffer);
unsigned long USBBufferSpaceAvailable(const tUSBBuffer *psBuffer);
void *USBBufferCallbackDataSet(tUSBBuffer *psBuffer, void *pvCBData);

unsigned long USBBufferEventCallback(void *pvCBData,
                                            unsigned long ulEvent,
                                            unsigned long ulMsgValue,
                                            void *pvMsgData);
```

```
const tUSBBuffer *USBBufferInit(const tUSBBuffer *psBuffer);
```

作用：初始化 Buffer，把它加入到当前设备中。首次使用 Buffer 必须使用此函数。

参数：psBuffer，待初始化的 Buffer。

返回：指向配置后的 Buffer。

```
void USBBufferInfoGet(const tUSBBuffer *psBuffer,
                          tUSBRingBufObject *psRingBuf);
```

作用：获取 Buffer 信息。psRingBuf 与 psBuffer 建立关系。

参数：psBuffer，操作的目标 Buffer。psRingBuf,申明一个 tUSBRingBufObject 变量。

返回：无。

```
unsigned long USBBufferWrite(const tUSBBuffer *psBuffer,
                                const unsigned char *pucData,
                                unsigned long ulLength);
```

作用：写入一组数据。直接写入。

参数：psBuffer，目标 Buffer。pucData,待写入数据指针。ulLength，写入长度。

返回：写入的数据长度。

```
void USBBufferDataWritten(const tUSBBuffer *psBuffer,
                                unsigned long ulLength);
```

作用：写入一组数据。使用前要调用 USBBufferInfoGet。

参数：psBuffer，目标 Buffer。ulLength，写入长度。

返回：写入的数据长度。

```
void USBBufferDataRemoved(const tUSBBuffer *psBuffer,
                                unsigned long ulLength);
```

作用：从 Buffer 中移出数据。

参数：psBuffer，目标 Buffer。ulLength，移出数据个数。

返回：无。

```
void USBBufferFlush(const tUSBBuffer *psBuffer);
```

作用：清除 Buffer 中数据。

参数：psBuffer，目标 Buffer。

返回：无。

```
unsigned long USBBufferRead(const tUSBBuffer *psBuffer,
                                   unsigned char *pucData,
                                   unsigned long ulLength);
```

作用：读取数据。

参数：psBuffer，目标 Buffer。pucData，数据存放指针。ulLength，读取个数。

返回：读取数据个数。

```
unsigned long USBBufferDataAvailable(const tUSBBuffer *psBuffer);
```

作用：可读取数据个数。

参数：psBuffer，目标 Buffer。

返回：可读取数据个数。

```
unsigned long USBBufferSpaceAvailable(const tUSBBuffer *psBuffer);
```

作用：可用数据空间大小。

参数：psBuffer，目标 Buffer。

返回：数据空间大小。

```
void *USBBufferCallbackDataSet(tUSBBuffer *psBuffer, void *pvCBData);
```

作用：修改设备 Buffer。

参数：psBuffer，用于替换的新 Buffer 指针。

返回：旧 Buffer 指针。

```
unsigned long USBBufferEventCallback(void *pvCBData,
                                          unsigned long ulEvent,
                                          unsigned long ulMsgValue,
                                          void *pvMsgData);
```

作用：Buffer 事件调用函数。

参数：pvCBData，设备指针。ulEvent，Buffer 事务。ulMsgValue，数据长度。pvMsgData 数据指针。

返回：函数是否成功执行。

以上是 11 个 Buffer 处理函数，用于 Buffer 数据接收、发送及处理。在 Bulk 传输中大量使用。

### 7.4 Bulk 设备开发

Bulk 设备开发只需要 4 步就能完成。如图 2 所示，Bulk 设备配置（主要是字符串描述符）、callback 函数编写、USB 处理器初始化、数据处理。

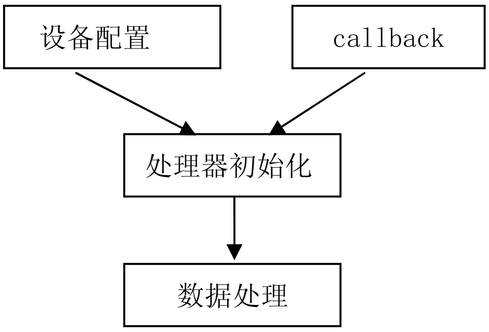


图 2

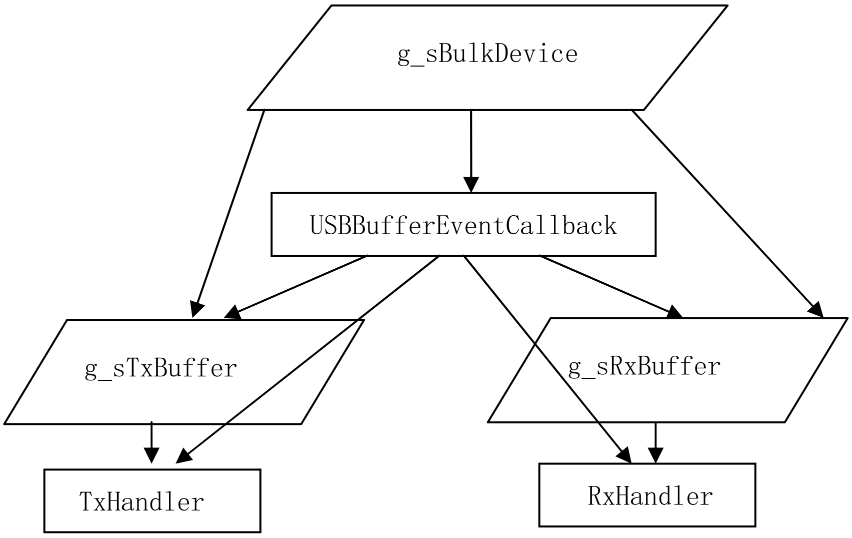
第一步：Bulk 设备配置（主要是字符串描述符），按字符串描述符标准完成串描述符配置，进而完成 Bulk 设备配置。

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_udma.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/usb.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdbulk.h"
#include "uartstdio.h"
#include "ustdlib.h"
//每次传输数据大小
#define BULK_BUFFER_SIZE 256
unsigned long RxHandler(void *pvCBData, unsigned long ulEvent,
                               unsigned long ulMsgValue, void *pvMsgData);
unsigned long TxHandler(void *pvlCBData, unsigned long ulEvent,
                               unsigned long ulMsgValue, void *pvMsgData);
unsigned long EchoNewDataToHost(tUSBDBulkDevice *psDevice, unsigned char *pcData,
                  unsigned long ulNumBytes);
#define COMMAND_PACKET_RECEIVED 0x00000001
#define COMMAND_STATUS_UPDATE   0x00000002
volatile unsigned long g_ulFlags = 0;
char *g_pcStatus;
static volatile tBoolean g_bUSBConfigured = false;
volatile unsigned long g_ulTxCount = 0;
volatile unsigned long g_ulRxCount = 0;
const tUSBBuffer g_sRxBuffer;
const tUSBBuffer g_sTxBuffer;
//*****************************************************************************
// 设备语言描述符.
//*****************************************************************************
const unsigned char g_pLangDescriptor[] =
{
    4,
    USB_DTYPE_STRING,
    USBShort(USB_LANG_EN_US)
};
//*****************************************************************************
// 制造商 字符串 描述符
```

```
//*****************************************************************************
const unsigned char g_pManufacturerString[] =
{
    (17 + 1) * 2,
    USB_DTYPE_STRING,
    ’T’, 0, ’e’, 0, ’x’, 0, ’a’, 0, ’s’, 0, ’ ’, 0, ’I’, 0, ’n’, 0, ’s’, 0,
    ’t’, 0, ’r’, 0, ’u’, 0, ’m’, 0, ’e’, 0, ’n’, 0, ’t’, 0, ’s’, 0,
};
//*****************************************************************************
//产品 字符串 描述符
//*****************************************************************************
const unsigned char g_pProductString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    ’G’, 0, ’e’, 0, ’n’, 0, ’e’, 0, ’r’, 0, ’i’, 0, ’c’, 0, ’ ’, 0, ’B’, 0,
    ’u’, 0, ’l’, 0, ’k’, 0, ’ ’, 0, ’D’, 0, ’e’, 0, ’v’, 0, ’i’, 0, ’c’, 0,
    ’e’, 0
};
//*****************************************************************************
//   产品 序列号 描述符
//*****************************************************************************
const unsigned char g_pSerialNumberString[] =
{
    (8 + 1) * 2,
    USB_DTYPE_STRING,
    ’1’, 0, ’2’, 0, ’3’, 0, ’4’, 0, ’5’, 0, ’6’, 0, ’7’, 0, ’8’, 0
};
//*****************************************************************************
// 设备接口字符串描述符
//*****************************************************************************
const unsigned char g_pDataInterfaceString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    ’B’, 0, ’u’, 0, ’l’, 0, ’k’, 0, ’ ’, 0, ’D’, 0, ’a’, 0, ’t’, 0,
    ’a’, 0, ’ ’, 0, ’I’, 0, ’n’, 0, ’t’, 0, ’e’, 0, ’r’, 0, ’f’, 0,
    ’a’, 0, ’c’, 0, ’e’, 0
};
//*****************************************************************************
//   设备配置字符串描述符
//*****************************************************************************
const unsigned char g_pConfigString[] =
{
    (23 + 1) * 2,
    USB_DTYPE_STRING,
    ’B’, 0, ’u’, 0, ’l’, 0, ’k’, 0, ’ ’, 0, ’D’, 0, ’a’, 0, ’t’, 0,
    ’a’, 0, ’ ’, 0, ’C’, 0, ’o’, 0, ’n’, 0, ’f’, 0, ’i’, 0, ’g’, 0,
    ’u’, 0, ’r’, 0, ’a’, 0, ’t’, 0, ’i’, 0, ’o’, 0, ’n’, 0
};
//******************************************************************************
// 字符串描述符集合
//******************************************************************************
const unsigned char * const g_pStringDescriptors[] =
{
    g_pLangDescriptor,
    g_pManufacturerString,
    g_pProductString,
    g_pSerialNumberString,
    g_pDataInterfaceString,
    g_pConfigString
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /                \
                                sizeof(unsigned char *))
//******************************************************************************
// 定义 Bulk 设备实例
//******************************************************************************
tBulkInstance g_sBulkInstance;
//******************************************************************************
// 定义 Bulk 设备
//******************************************************************************
const tUSBDBulkDevice g_sBulkDevice =
{
    0x1234,
    USB_PID_BULK,
    500,
    USB_CONF_ATTR_SELF_PWR,
    USBBufferEventCallback,
    (void *)&g_sRxBuffer,
    USBBufferEventCallback,
    (void *)&g_sTxBuffer,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    &g_sBulkInstance
};
//******************************************************************************
// 定义 Buffer
//******************************************************************************
```

```
unsigned char g_pucUSBRxBuffer[BULK_BUFFER_SIZE];
unsigned char g_pucUSBTxBuffer[BULK_BUFFER_SIZE];
unsigned char g_pucTxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
unsigned char g_pucRxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
const tUSBBuffer g_sRxBuffer =
{
    false,                          // This is a receive buffer.
    RxHandler,                      // pfnCallback
    (void *)&g_sBulkDevice,         // Callback data is our device pointer.
    USBDBulkPacketRead,             // pfnTransfer
    USBDBulkRxPacketAvailable,      // pfnAvailable
    (void *)&g_sBulkDevice,         // pvHandle
    g_pucUSBRxBuffer,               // pcBuffer
    BULK_BUFFER_SIZE,               // ulBufferSize
    g_pucRxBufferWorkspace          // pvWorkspace
};
const tUSBBuffer g_sTxBuffer =
{
    true,                           // This is a transmit buffer.
    TxHandler,                      // pfnCallback
    (void *)&g_sBulkDevice,         // Callback data is our device pointer.
    USBDBulkPacketWrite,            // pfnTransfer
    USBDBulkTxPacketAvailable,      // pfnAvailable
    (void *)&g_sBulkDevice,         // pvHandle
    g_pucUSBTxBuffer,               // pcBuffer
    BULK_BUFFER_SIZE,               // ulBufferSize
    g_pucTxBufferWorkspace          // pvWorkspace
};
```

tUSBDBulkDevice g_sBulkDevice、tUSBBuffer g_sTxBuffer、tUSBBuffer g_sRxBuffer 是管理 Bulk 设备的主要结构体，它们三者关系如下图：



tUSBDBulkDevice g_sBulkDevice 主要进行上层协议与通信管理控制，通过 USBBufferEventCallback 函数处理 Buffer 数据：数据发送、接收、控制事件，通过 g_sTxBuffer 中的 USBDBulkPacketWrite 和 USBDBulkTxPacketAvailable 实现底层数据发送，并通过 TxHandler 返回处理结果；通过 g_sRxBuffer 中的 USBDBulkPacketRead 和 USBDBulkRxPacketAvailable 实现底层数据接收，并通过 RxHandler 返回处理结果。在 g_sBulkDevice 层可以直接使用 Buffer 函数对 Buffer 层操作，并通过 TxHandler 和 RxHandler 返回处理结果。所有处理过程中的数据都保存在 Buffer 层的 g_pucUSBTxBuffer 或者 g_pucUSBRxBuffer，隶属于 g_sBulkDevice 的一部分。注意：USBDBulkPacketWrite、USBDBulkTxPacketAvailable、USBDBulkPacketRead、USBDBulkRxPacketAvailable 由 Bulk 设备类 API 定义，可以直接使用。USBBufferEventCallback 为 Buffer 层定义的标准 API，用于处理、调用 USBDBulkPacketWrite、USBDBulkTxPacketAvailable、USBDBulkPacketRead、USBDBulkRxPacketAvailable、TxHandler 和 RxHandler 完成 g_sBulkDevice 层发送的数据接收与发送命令。

第二步：完成 Callback 函数。Callback 函数用于处理输出端点、输入端点数据事务。Bulk 设备接收回调函数包含以下事务：USB_EVENT_CONNECTED、USB_EVENT_DISCONNECTED、USB_EVENT_RX_AVAILABLE、USB_EVENT_SUSPEND、USB_EVENT_RESUME、USB_EVENT_ERROR。Bulk 设备发送回调函数包含了以下事务：USB_EVENT_TX_COMPLETE。如下表：

| 名称 | 属性 | 说明 |
|---|---|---|
| USB_EVENT_CONNECTED | 接收 | USB 设备已经连接到主机 |
| USB_EVENT_DISCONNECTED | 接收 | USB 设备已经与主机断开 |
| USB_EVENT_RX_AVAILABLE | 接收 | 有接受数据 |
| USB_EVENT_SUSPEND | 接收 | 挂起 |
| USB_EVENT_RESUME | 接收 | 唤醒 |
| USB_EVENT_ERROR | 接收 | 错误 |
| USB_EVENT_TX_COMPLETE | 发送 | 发送完成 |

表 2. Bulk 事务

根据以上事务编写 Callback 函数：

```
//****************************************************************************
//USB Bulk 设备类返回事件处理函数（callback）.
```

```
//*****************************************************************************
unsigned long  TxHandler(void *pvCBData, unsigned long ulEvent, unsigned long ulMsgValue,
          void *pvMsgData)
{
    //发送完成事件
    if(ulEvent == USB_EVENT_TX_COMPLETE)
    {
        g_ulTxCount += ulMsgValue;
    }
    return(0);
}
unsigned long RxHandler(void *pvCBData, unsigned long ulEvent,
               unsigned long ulMsgValue, void *pvMsgData)
{
    // 接收事件
    switch(ulEvent)
    {
        //连接成功
        case USB_EVENT_CONNECTED:
        {
              GPIOPinWrite(GPIO_PORTF_BASE,0x40,0x40);
            g_bUSBConfigured = true;
            g_pcStatus = "Host connected.";
            g_ulFlags |= COMMAND_STATUS_UPDATE;
            // Flush our buffers.
            USBBufferFlush(&g_sTxBuffer);
            USBBufferFlush(&g_sRxBuffer);

            break;
        }
        // 断开连接.
        case USB_EVENT_DISCONNECTED:
        {
              GPIOPinWrite(GPIO_PORTF_BASE,0x40,0x00);
            g_bUSBConfigured = false;
            g_pcStatus = "Host disconnected.";
            g_ulFlags |= COMMAND_STATUS_UPDATE;
            break;
        }
        // 有可能数据接收.
        case USB_EVENT_RX_AVAILABLE:
        {
            tUSBDBulkDevice *psDevice;
            psDevice = (tUSBDBulkDevice *)pvCBData;
            // 把接收到的数据发送回去。
            return(EchoNewDataToHost(psDevice, pvMsgData, ulMsgValue));
        }
        //挂起，唤醒
        case USB_EVENT_SUSPEND:
        case USB_EVENT_RESUME:break;
        default:break;
    }
    return(0);
}
//********************************************************************************
//EchoNewDataToHost 函数
//********************************************************************************
unsigned long EchoNewDataToHost(tUSBDBulkDevice *psDevice, unsigned char *pcData,
                  unsigned long ulNumBytes)
{
    unsigned long ulLoop, ulSpace, ulCount;
    unsigned long ulReadIndex;
    unsigned long ulWriteIndex;
    tUSBRingBufObject sTxRing;
    // 获取 Buffer 信息.
    USBBufferInfoGet(&g_sTxBuffer, &sTxRing);
    // 有多少可能空间
    ulSpace = USBBufferSpaceAvailable(&g_sTxBuffer);
    // 改变数据
    ulLoop = (ulSpace < ulNumBytes) ? ulSpace : ulNumBytes;
    ulCount = ulLoop;
    // 更新接收到的数据个数
    g_ulRxCount += ulNumBytes;
    ulReadIndex = (unsigned long)(pcData - g_pucUSBRxBuffer);
    ulWriteIndex = sTxRing.ulWriteIndex;
    while(ulLoop)
    {
        //更新接收的数据
        if((g_pucUSBRxBuffer[ulReadIndex] >= ’a’) &&
           (g_pucUSBRxBuffer[ulReadIndex] <= ’z’))
        {
            //转换
            g_pucUSBTxBuffer[ulWriteIndex] =
                (g_pucUSBRxBuffer[ulReadIndex] - ’a’) + ’A’;
        }
        else
        {
             //转换
            if((g_pucUSBRxBuffer[ulReadIndex] >= 'A') &&
               (g_pucUSBRxBuffer[ulReadIndex] <= 'Z'))
            {
                 //转换
                g_pucUSBTxBuffer[ulWriteIndex] =
                    (g_pucUSBRxBuffer[ulReadIndex] - 'Z') + 'z';
            }
            else
            {
                 //转换
                g_pucUSBTxBuffer[ulWriteIndex] = g_pucUSBRxBuffer[ulReadIndex];
            }
        }
        // 更新指针
        ulWriteIndex++;
        ulWriteIndex = (ulWriteIndex == BULK_BUFFER_SIZE) ? 0 : ulWriteIndex;

        ulReadIndex++;
        ulReadIndex = (ulReadIndex == BULK_BUFFER_SIZE) ? 0 : ulReadIndex;
        ulLoop--;
    }
    // 发送数据
    USBBufferDataWritten(&g_sTxBuffer, ulCount);
    return(ulCount);
}
```

第三步：系统初始化，配置内核电压、系统主频、使能端口、LED 控制等，本例中使用 4 个 LED 进行指示数据传输。在这个例子中，Bulk 传输接收的数据发送给主机。原理图如图 3 所示：

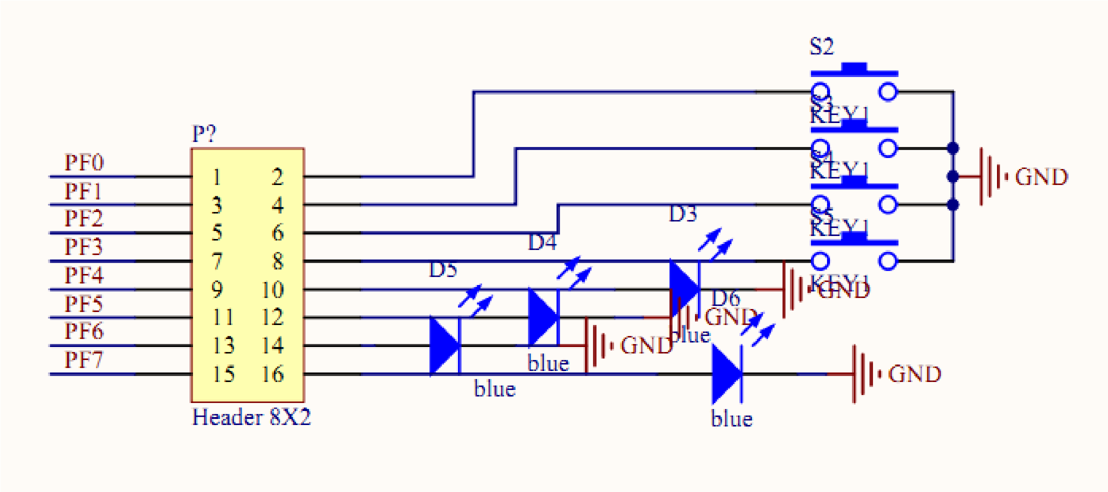


图 3

系统初始化：

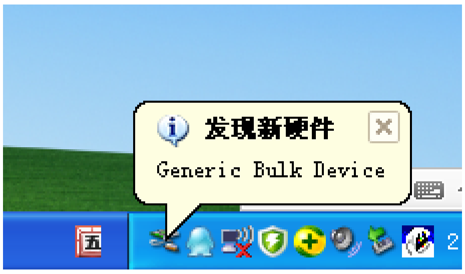
```
 unsigned long ulTxCount = 0;
unsigned long ulRxCount = 0;
```

```
//    char pcBuffer[16];
//设置内核电压、主频 50Mhz
 SysCtlLDOSet(SYSCTL_LDO_2_75V);
 SysCtlClockSet(SYSCTL_XTAL_8MHZ | SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL  | SYSCTL_OSC_MAIN );

 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
 GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,0xf0);
 GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,0x0f);
 HWREG(GPIO_PORTF_BASE+GPIO_O_PUR) |= 0x0f;

 // 初始化发送与接收 Buffer.
USBBufferInit((tUSBBuffer *)&g_sTxBuffer);
USBBufferInit((tUSBBuffer *)&g_sRxBuffer);
 // 初始化 Bulk 设备
USBDBulkInit(0, (tUSBDBulkDevice *)&g_sBulkDevice);
```

第四步：数据处理。主要使用 11 个 Buffer 处理函数，用于 Buffer 数据接收、发送及处理。在 Bulk 传输中大量使用。

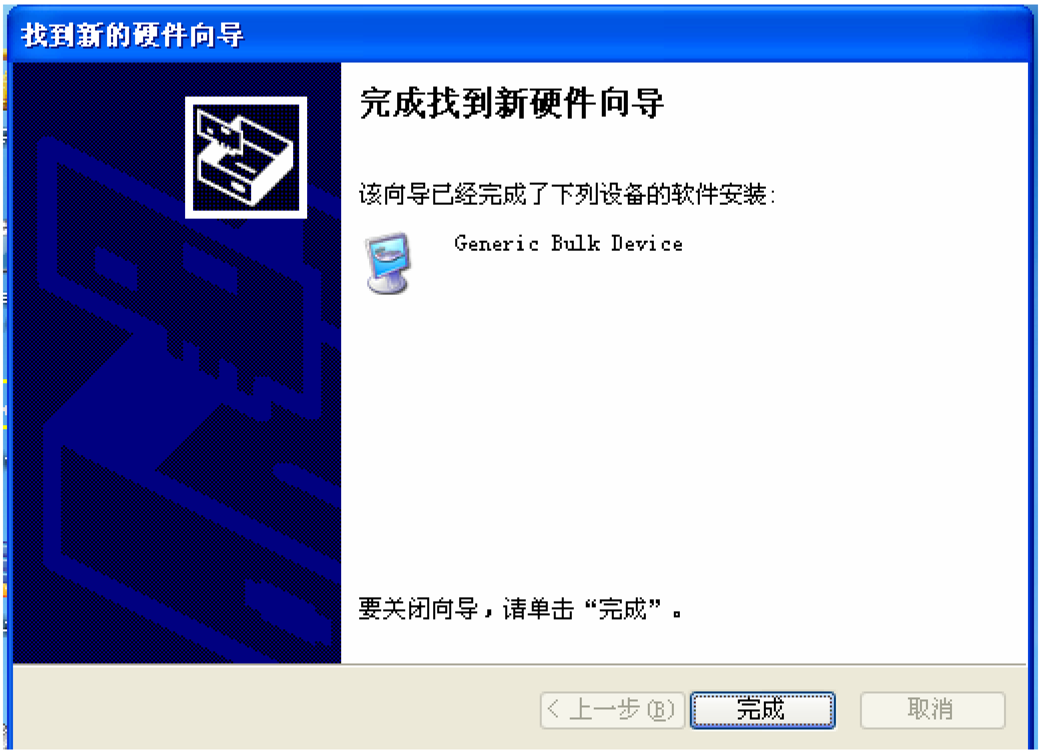
```
while(1)
{
    //等待连接结束.
    while((g_ulFlags & FLAG_CONNECTED) == 0)
    {
    }
     //初始化 Buffer
    g_sBuffer.pucFill = g_sBuffer.pucBuffer;
    g_sBuffer.pucPlay = g_sBuffer.pucBuffer;
    g_sBuffer.ulFlags = 0;
     //从 Bulk 设备类中获取数据
    if(USBBulkBufferOut(g_pvBulkDevice,
                         (unsigned char *)g_sBuffer.pucFill,
                        BULK_PACKET_SIZE, USBBufferCallback) == 0)
    {
        //标记数据放入 buffer 中.
        g_sBuffer.ulFlags |= SBUFFER_FLAGS_FILLING;
    }
    //设备连接到主机.
    while(g_ulFlags & FLAG_CONNECTED)
    {
        // 检查音量是否有改变.
        if(g_ulFlags & FLAG_VOLUME_UPDATE)
        {
            // 清除更新音量标志.
            g_ulFlags &= ~FLAG_VOLUME_UPDATE;
            // 修改音量，自行添加代码.在此以 LED 灯做指示。
            //UpdateVolume();
               GPIOPinWrite(GPIO_PORTF_BASE,0x40,~GPIOPinRead(GPIO_PORTF_BASE,0x40));
        }
        //是否静音
        if(g_ulFlags & FLAG_MUTE_UPDATE)
        {
            //修改静音状态，自行添加函数.在此以 LED 灯做指示。
            //UpdateMute();
               if(g_ulFlags & FLAG_MUTED)
                    GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
               else
                    GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x00);
            // 清除静音标志
            g_ulFlags &= ~FLAG_MUTE_UPDATE;
        }
    }
}
//******************************************************************************
//USBBulkBufferOut 的 Callback 入口参数
//******************************************************************************
void USBBufferCallback(void *pvBuffer, unsigned long ulParam, unsigned long ulEvent)
{
 //数据处理，自行加入代码。
 // Your Codes .........
    //再一次获取数据.
    USBBulkBufferOut(g_pvBulkDevice, (unsigned char *)g_sBuffer.pucFill,
                       BULK_PACKET_SIZE, USBBufferCallback);
}
```

使用上面四步就完成 Bulk 设备开发。Bulk 设备开发时要加入两个 lib 库函数：usblib.lib 和 DriverLib.lib，在启动代码中加入 USB0DeviceIntHandler 中断服务函数。以上 Bulk 设备开发完成，在 Win xp 下运行效果如下图所示：



发现新硬件

Generic Bulk Device


找到新的硬件向导

完成找到新硬件向导

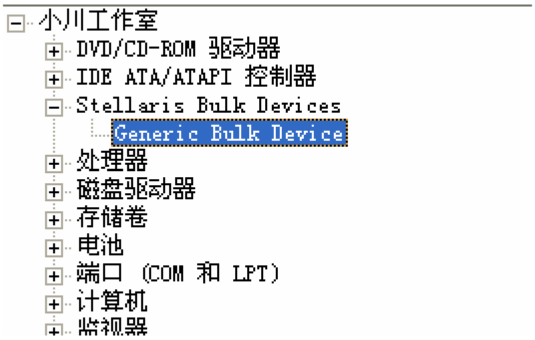
该向导已经完成了下列设备的软件安装:

Generic Bulk Device

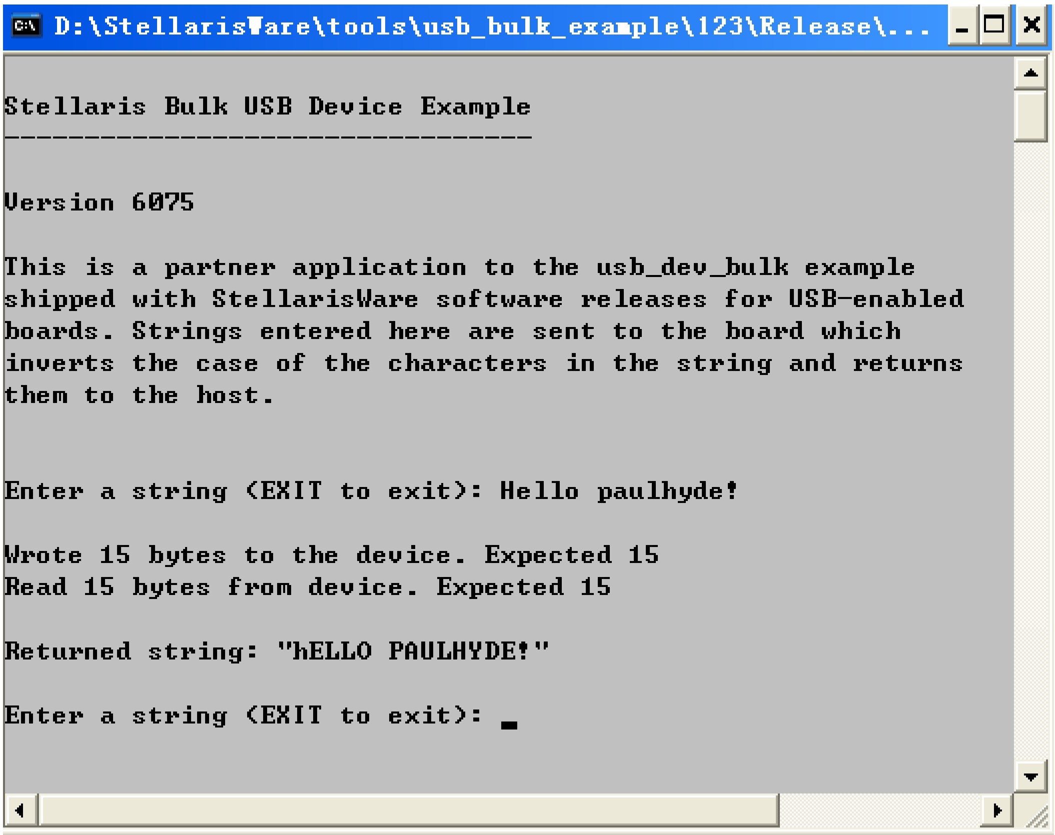
要关闭向导，请单击“完成”。

驱动安装完成

在枚举过程中可以看出，在电脑右下脚可以看到“Generic Bulk Device”字样，标示正在进行枚举，并手手动安装驱动。枚举成功后，在“设备管理器”的“Stellaris Bulk Device”中看到“Generic Bulk Device”设备，如下图。现在 Bulk 设备可以正式使用。


小川工作室
DVD/CD-ROM 驱动器
IDE ATA/ATAPI 控制器
Stellaris Bulk Devices
Generic Bulk Device
处理器
磁盘驱动器
存储卷
电池
端口 (COM 和 LPT)
计算机
监视器

Bulk 设备要配合上位机使用，上位机发送字符串通过 USB Bulk 设备转换后发送给主机。运行图如下：



```
Stellaris Bulk USB Device Example
---------------------------------

Version 6075

This is a partner application to the usb_dev_bulk example
shipped with StellarisWare software releases for USB-enabled
boards. Strings entered here are sent to the board which
inverts the case of the characters in the string and returns
them to the host.


Enter a string (EXIT to exit): Hello paulhyde!

Wrote 15 bytes to the device. Expected 15
Read 15 bytes from device. Expected 15

Returned string: "hELLO PAULHYDE!"

Enter a string (EXIT to exit): _
```

上位机源码如下：

```
#include <windows.h>
#include <strsafe.h>
#include <initguid.h>
#include "lmusbdll.h"
#include "luminary_guids.h"
//*****************************************************************************
// Buffer size definitions.
//*****************************************************************************
#define MAX_STRING_LEN 256
#define MAX_ENTRY_LEN 256
#define USB_BUFFER_LEN 1216
//*****************************************************************************
// The build version number
//*****************************************************************************
#define BLDVER "6075"
//*****************************************************************************
// The number of bytes we read and write per transaction if in echo mode.
//*****************************************************************************
#define ECHO_PACKET_SIZE 1216
//*****************************************************************************
// Buffer into which error messages are written.
//*****************************************************************************
TCHAR g_pcErrorString[MAX_STRING_LEN];
```

```
//****************************************************************************
// The number of bytes transfered in the last measurement interval.
//****************************************************************************
ULONG g_ulByteCount = 0;
//****************************************************************************
// The total number of packets transfered.
//****************************************************************************
ULONG g_ulPacketCount = 0;
//****************************************************************************
LPTSTR GetSystemErrorString(DWORD dwError)
{
    DWORD dwRetcode;
    // Ask Windows for the error message description.
    dwRetcode = FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, ″%0″, dwError, 0,
                              g_pcErrorString, MAX_STRING_LEN, NULL);
    if(dwRetcode == 0)
    {
        return((LPTSTR)L″Unknown″);
    }
    else
    {
        // Remove the trailing ″\n\r″ if present.
        if(dwRetcode >= 2)
        {
            if(g_pcErrorString[dwRetcode - 2] == ’\r’)
            {
                g_pcErrorString[dwRetcode - 2] = ’\0’;
            }
        }
        return(g_pcErrorString);
    }
}
//****************************************************************************
// Print the throughput in terms of Kbps once per second.
//****************************************************************************
void UpdateThroughput(void)
{
    static ULONG ulStartTime = 0;
    static ULONG ulLast = 0;
    ULONG ulNow;
    ULONG ulElapsed;
    SYSTEMTIME sSysTime;
    // Get the current system time.
    GetSystemTime(&sSysTime);
```

```
        ulNow = (((((sSysTime.wHour * 60) +
                   sSysTime.wMinute) * 60) +
                  sSysTime.wSecond) * 1000) + sSysTime.wMilliseconds;
        // If this is the first call, set the start time.
        if(ulStartTime == 0)
        {
            ulStartTime = ulNow;
            ulLast = ulNow;
            return;
        }
        // How much time has elapsed since the last measurement?
        ulElapsed = (ulNow > ulStartTime) ? (ulNow - ulStartTime) : (ulStartTime - ulNow);
        // We dump a new measurement every second.
        if(ulElapsed > 1000)
        {
            printf(″\r%6dKbps Packets: %10d ″, ((g_ulByteCount * 8) / ulElapsed), g_ulPacketCount);
            g_ulByteCount = 0;
            ulStartTime = ulNow;
        }
    }
    //*****************************************************************************
    // The main application entry function.
    //*****************************************************************************
    int main(int argc, char *argv[])
    {
        BOOL bResult;
        BOOL bDriverInstalled;
        BOOL bEcho;
        char szBuffer[USB_BUFFER_LEN];
        ULONG ulWritten;
        ULONG ulRead;
        ULONG ulLength;
        DWORD dwError;
        LMUSB_HANDLE hUSB;
        // Are we operating in echo mode or not? The ″-e″ parameter tells the
        // app to echo everything it receives back to the device unchanged.
        bEcho = ((argc > 1) && (argv[1][1] == ’e’)) ? TRUE : FALSE;
        // Print a cheerful welcome.
        printf(″\nStellaris Bulk USB Device Example\n″);
        printf( ″---------------------------------\n\n″);
        printf(″Version %s\n\n″, BLDVER);
        if(!bEcho)
        {
```

```
        printf("This is a partner application to the usb_dev_bulk example\n");
        printf("shipped with StellarisWare software releases for USB-enabled\n");
        printf("boards. Strings entered here are sent to the board which\n");
        printf("inverts the case of the characters in the string and returns\n");
        printf("them to the host.\n\n");
    }
    else
    {
        printf("If run with the \"-e\" command line switch, this application\n");
        printf("echoes all data received on the bulk IN endpoint to the bulk\n");
        printf("OUT endpoint.  This feature may be helpful during development\n");
        printf("and debug of your own USB devices.  Note that this will not\n");
        printf("do anything exciting if run with the usb_dev_bulk example\n");
        printf("device attached since it expects the host to initiate transfers.\n\n");
    }
    // Find our USB device and prepare it for communication.
    hUSB = InitializeDevice(BULK_VID, BULK_PID,
                            (LPGUID)&(GUID_DEVINTERFACE_STELLARIS_BULK),
                            &bDriverInstalled);
    if(hUSB)
    {
        // Are we operating in echo mode or not? The "-e" parameter tells the
        // app to echo everything it receives back to the device unchanged.
        if(bEcho)
        {
            printf("Running in echo mode. Press Ctrl+C to exit.\n\n"
                "Throughput:      0Kbps Packets:           0");
            while(1)
            {
                // Read a block of data from the device.
                dwError = ReadUSBPacket(hUSB, szBuffer, USB_BUFFER_LEN, &ulRead,
                                        INFINITE, NULL);
                if(dwError != ERROR_SUCCESS)
                {
                    // We failed to read from the device.
                    printf("\n\nError %d (%S) reading from bulk IN pipe.\n", dwError,
                           GetSystemErrorString(dwError));
                    break;
                }
                else
                {
                    // Update our byte and packet counters.
                    g_ulByteCount += ulRead;
                    g_ulPacketCount++;
            // Write the data back out to the device.
            bResult = WriteUSBPacket(hUSB, szBuffer, ulRead, &ulWritten);
            if(!bResult)
            {
                // We failed to write the data for some reason.
                dwError = GetLastError();
                printf(″\n\nError %d (%S) writing to bulk OUT pipe.\n″, dwError,
                       GetSystemErrorString(dwError));
                break;
            }
            // Display the throughput.
           UpdateThroughput();
        }
    }
}
else
{
    // We are running in normal mode.  Keep sending and receiving
    // strings until the user indicates that it is time to exit.
    while(1)
    {
        // The device was found and successfully configured. Now get a string from
        // the user...
        do
        {
            printf(″\nEnter a string (EXIT to exit): ″);
            fgets(szBuffer, MAX_ENTRY_LEN, stdin);
            printf(″\n″);
            // How many characters were entered (including the trailing ’\n’)?
            ulLength = (ULONG)strlen(szBuffer);
            if(ulLength <= 1)
            {
                printf(″\nPlease enter some text.\n″);
                ulLength = 0;
            }
            else
            {
                // Get rid of the trailing ’\n’ if there is one there.
                 if(szBuffer[ulLength - 1] == ’\n’)
                {
                    szBuffer[ulLength - 1] = ’\0’;
                    ulLength--;
                }
            }
            }
            while(ulLength == 0);
            if(!(strcmp(″EXIT″, szBuffer)))
            {
                printf(″Exiting on user request.\n″);
                break;
            }
            // Write the user’s string to the device.
            bResult = WriteUSBPacket(hUSB, szBuffer, ulLength, &ulWritten);
            if(!bResult)
            {
                dwError = GetLastError();
                printf(″Error %d (%S) writing to bulk OUT pipe.\n″, dwError,
                       GetSystemErrorString(dwError));
            }
            else
            {
                // We wrote data successfully so now read it back.
                printf(″Wrote %d bytes to the device. Expected %d\n″,
                       ulWritten, ulLength);
                // We expect the same number of bytes as we just sent.
                dwError = ReadUSBPacket(hUSB, szBuffer, ulWritten, &ulRead,
                                        INFINITE, NULL);
                if(dwError != ERROR_SUCCESS)
                {
                    // We failed to read from the device.
                    printf(″Error %d (%S) reading from bulk IN pipe.\n″, dwError,
                           GetSystemErrorString(dwError));
                }
                else
                {
                    szBuffer[ulRead] = ’\0’;
                    printf(″Read %d bytes from device. Expected %d\n″,
                           ulRead, ulWritten);
                    printf(″\nReturned string: \″%s\″\n″, szBuffer);
                }
            }
        }
    }
}
else
{
    // An error was reported while trying to connect to the device.
    dwError = GetLastError();
```

```
        printf("\nUnable to initialize the Stellaris Bulk USB Device.\n");
        printf("Error code is %d (%S)\n\n", dwError, GetSystemErrorString(dwError));
        printf("Please make sure you have a Stellaris USB-enabled evaluation\n");
        printf("or development kit running the usb_dev_bulk example\n");
        printf("application connected to this system via the \"USB OTG\" or\n");
        printf("\"USB DEVICE\" connectors. Once the device is connected, run\n");
        printf("this application again.\n\n");
        printf("\nPress \"Enter\" to exit: ");
        fgets(szBuffer, MAX_STRING_LEN, stdin);
        printf("\n");
        return(2);
    }
    TerminateDevice(hUSB);
    return(0);
}
```

Bulk 设备开发源码如下：

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_udma.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/usb.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdbulk.h"
#include "uartstdio.h"
#include "ustdlib.h"
//每次传输数据大小
#define BULK_BUFFER_SIZE 256
unsigned long RxHandler(void *pvCBData, unsigned long ulEvent,
                               unsigned long ulMsgValue, void *pvMsgData);
unsigned long TxHandler(void *pvlCBData, unsigned long ulEvent,
                               unsigned long ulMsgValue, void *pvMsgData);
unsigned long EchoNewDataToHost(tUSBDBulkDevice *psDevice, unsigned char *pcData,
                  unsigned long ulNumBytes);
#define COMMAND_PACKET_RECEIVED 0x00000001
#define COMMAND_STATUS_UPDATE   0x00000002
volatile unsigned long g_ulFlags = 0;
char *g_pcStatus;
```

```
static volatile tBoolean g_bUSBConfigured = false;
volatile unsigned long g_ulTxCount = 0;
volatile unsigned long g_ulRxCount = 0;
const tUSBBuffer g_sRxBuffer;
const tUSBBuffer g_sTxBuffer;
//*****************************************************************************
// 设备语言描述符.
//*****************************************************************************
const unsigned char g_pLangDescriptor[] =
{
    4,
    USB_DTYPE_STRING,
    USBShort(USB_LANG_EN_US)
};
//*****************************************************************************
// 制造商 字符串 描述符
//*****************************************************************************
const unsigned char g_pManufacturerString[] =
{
    (17 + 1) * 2,
    USB_DTYPE_STRING,
    'T’, 0, 'e’, 0, 'x’, 0, 'a’, 0, 's’, 0, ' ’, 0, 'I’, 0, 'n’, 0, 's’, 0,
    't’, 0, 'r’, 0, 'u’, 0, 'm’, 0, 'e’, 0, 'n’, 0, 't’, 0, 's’, 0,
};
//*****************************************************************************
//产品 字符串 描述符
//*****************************************************************************
const unsigned char g_pProductString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    'G’, 0, 'e’, 0, 'n’, 0, 'e’, 0, 'r’, 0, 'i’, 0, 'c’, 0, ' ’, 0, 'B’, 0,
    'u’, 0, 'l’, 0, 'k’, 0, ' ’, 0, 'D’, 0, 'e’, 0, 'v’, 0, 'i’, 0, 'c’, 0,
    'e’, 0
};
//*****************************************************************************
//   产品 序列号 描述符
//*****************************************************************************
const unsigned char g_pSerialNumberString[] =
{
    (8 + 1) * 2,
    USB_DTYPE_STRING,
    '1’, 0, '2’, 0, '3’, 0, '4’, 0, '5’, 0, '6’, 0, '7’, 0, '8’, 0
};
```

```
//*****************************************************************************
// 设备接口字符串描述符
//*****************************************************************************
const unsigned char g_pDataInterfaceString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    ’B’, 0, ’u’, 0, ’l’, 0, ’k’, 0, ’ ’, 0, ’D’, 0, ’a’, 0, ’t’, 0,
    ’a’, 0, ’ ’, 0, ’I’, 0, ’n’, 0, ’t’, 0, ’e’, 0, ’r’, 0, ’f’, 0,
    ’a’, 0, ’c’, 0, ’e’, 0
};
//*****************************************************************************
//  设备配置字符串描述符
//*****************************************************************************
const unsigned char g_pConfigString[] =
{
    (23 + 1) * 2,
    USB_DTYPE_STRING,
    ’B’, 0, ’u’, 0, ’l’, 0, ’k’, 0, ’ ’, 0, ’D’, 0, ’a’, 0, ’t’, 0,
    ’a’, 0, ’ ’, 0, ’C’, 0, ’o’, 0, ’n’, 0, ’f’, 0, ’i’, 0, ’g’, 0,
    ’u’, 0, ’r’, 0, ’a’, 0, ’t’, 0, ’i’, 0, ’o’, 0, ’n’, 0
};
//*****************************************************************************
// 字符串描述符集合
//*****************************************************************************
const unsigned char * const g_pStringDescriptors[] =
{
    g_pLangDescriptor,
    g_pManufacturerString,
    g_pProductString,
    g_pSerialNumberString,
    g_pDataInterfaceString,
    g_pConfigString
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /                \
                                sizeof(unsigned char *))
//*****************************************************************************
// 定义 Bulk 设备实例
//*****************************************************************************
tBulkInstance g_sBulkInstance;
//*****************************************************************************
// 定义 Bulk 设备
//*****************************************************************************
const tUSBDBulkDevice g_sBulkDevice =
{
    0x1234,
    USB_PID_BULK,
    500,
    USB_CONF_ATTR_SELF_PWR,
    USBBufferEventCallback,
    (void *)&g_sRxBuffer,
    USBBufferEventCallback,
    (void *)&g_sTxBuffer,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    &g_sBulkInstance
};
//******************************************************************************
// 定义 Buffer
//******************************************************************************
unsigned char g_pucUSBRxBuffer[BULK_BUFFER_SIZE];
unsigned char g_pucUSBTxBuffer[BULK_BUFFER_SIZE];
unsigned char g_pucTxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
unsigned char g_pucRxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
const tUSBBuffer g_sRxBuffer =
{
    false,                            // This is a receive buffer.
    RxHandler,                        // pfnCallback
    (void *)&g_sBulkDevice,           // Callback data is our device pointer.
    USBDBulkPacketRead,               // pfnTransfer
    USBDBulkRxPacketAvailable,        // pfnAvailable
    (void *)&g_sBulkDevice,           // pvHandle
    g_pucUSBRxBuffer,                 // pcBuffer
    BULK_BUFFER_SIZE,                 // ulBufferSize
    g_pucRxBufferWorkspace            // pvWorkspace
};
const tUSBBuffer g_sTxBuffer =
{
    true,                             // This is a transmit buffer.
    TxHandler,                        // pfnCallback
    (void *)&g_sBulkDevice,           // Callback data is our device pointer.
    USBDBulkPacketWrite,              // pfnTransfer
    USBDBulkTxPacketAvailable,        // pfnAvailable
    (void *)&g_sBulkDevice,           // pvHandle
    g_pucUSBTxBuffer,                 // pcBuffer
    BULK_BUFFER_SIZE,                 // ulBufferSize
    g_pucTxBufferWorkspace            // pvWorkspace
};
```

```
//******************************************************************************
//USB Bulk 设备类返回事件处理函数（callback）.
//******************************************************************************
unsigned long  TxHandler(void *pvCBData, unsigned long ulEvent, unsigned long ulMsgValue,
          void *pvMsgData)
{
    //发送完成事件
    if(ulEvent == USB_EVENT_TX_COMPLETE)
    {
        g_ulTxCount += ulMsgValue;
    }
    return(0);
}
unsigned long RxHandler(void *pvCBData, unsigned long ulEvent,
               unsigned long ulMsgValue, void *pvMsgData)
{
    // 接收事件
    switch(ulEvent)
    {
        //连接成功
        case USB_EVENT_CONNECTED:
        {
              GPIOPinWrite(GPIO_PORTF_BASE,0x40,0x40);
            g_bUSBConfigured = true;
            g_pcStatus = "Host connected.";
            g_ulFlags |= COMMAND_STATUS_UPDATE;
            // Flush our buffers.
            USBBufferFlush(&g_sTxBuffer);
            USBBufferFlush(&g_sRxBuffer);

            break;
        }
        // 断开连接.
        case USB_EVENT_DISCONNECTED:
        {
              GPIOPinWrite(GPIO_PORTF_BASE,0x40,0x00);
            g_bUSBConfigured = false;
            g_pcStatus = "Host disconnected.";
            g_ulFlags |= COMMAND_STATUS_UPDATE;
            break;
        }
        // 有可能数据接收.
        case USB_EVENT_RX_AVAILABLE:
        {
            tUSBDBulkDevice *psDevice;
            psDevice = (tUSBDBulkDevice *)pvCBData;
            // 把接收到的数据发送回去。
            return(EchoNewDataToHost(psDevice, pvMsgData, ulMsgValue));
        }
        //挂起，唤醒
        case USB_EVENT_SUSPEND:
        case USB_EVENT_RESUME:break;
        default:break;
    }
    return(0);
}
//****************************************************************************
//EchoNewDataToHost 函数
//****************************************************************************
unsigned long EchoNewDataToHost(tUSBDBulkDevice *psDevice, unsigned char *pcData,
                  unsigned long ulNumBytes)
{
    unsigned long ulLoop, ulSpace, ulCount;
    unsigned long ulReadIndex;
    unsigned long ulWriteIndex;
    tUSBRingBufObject sTxRing;
    // 获取 Buffer 信息.
    USBBufferInfoGet(&g_sTxBuffer, &sTxRing);
    // 有多少可能空间
    ulSpace = USBBufferSpaceAvailable(&g_sTxBuffer);
    // 改变数据
    ulLoop = (ulSpace < ulNumBytes) ? ulSpace : ulNumBytes;
    ulCount = ulLoop;
    // 更新接收到的数据个数
    g_ulRxCount += ulNumBytes;
    ulReadIndex = (unsigned long)(pcData - g_pucUSBRxBuffer);
    ulWriteIndex = sTxRing.ulWriteIndex;
    while(ulLoop)
    {
        //更新接收的数据
        if((g_pucUSBRxBuffer[ulReadIndex] >= 'a') &&
           (g_pucUSBRxBuffer[ulReadIndex] <= 'z'))
        {
            //转换
            g_pucUSBTxBuffer[ulWriteIndex] =
                (g_pucUSBRxBuffer[ulReadIndex] - 'a') + 'A';
        }
        else
```

```
            {
                 //转换
                if((g_pucUSBRxBuffer[ulReadIndex] >= 'A’) &&
                   (g_pucUSBRxBuffer[ulReadIndex] <= 'Z’))
                {
                     //转换
                    g_pucUSBTxBuffer[ulWriteIndex] =
                        (g_pucUSBRxBuffer[ulReadIndex] - 'Z’) + 'z’;
                }
                else
                {
                     //转换
                    g_pucUSBTxBuffer[ulWriteIndex] = g_pucUSBRxBuffer[ulReadIndex];
                }
            }
            // 更新指针
            ulWriteIndex++;
            ulWriteIndex = (ulWriteIndex == BULK_BUFFER_SIZE) ? 0 : ulWriteIndex;

            ulReadIndex++;
            ulReadIndex = (ulReadIndex == BULK_BUFFER_SIZE) ? 0 : ulReadIndex;
            ulLoop--;
        }
        // 发送数据
        USBBufferDataWritten(&g_sTxBuffer, ulCount);
        return(ulCount);
    }
    //****************************************************************************
    // 应用主函数.
    //****************************************************************************
    int   main(void)
    {
         unsigned long ulTxCount = 0;
        unsigned long ulRxCount = 0;
         //   char pcBuffer[16];
        //设置内核电压、主频 50Mhz
         SysCtlLDOSet(SYSCTL_LDO_2_75V);
         SysCtlClockSet(SYSCTL_XTAL_8MHZ    |    SYSCTL_SYSDIV_4    |    SYSCTL_USE_PLL       |
SYSCTL_OSC_MAIN );
         SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
         GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,0xf0);
         GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,0x0f);
         HWREG(GPIO_PORTF_BASE+GPIO_O_PUR) |= 0x0f;
```

```
     // 初始化发送与接收 Buffer.
    USBBufferInit((tUSBBuffer *)&g_sTxBuffer);
    USBBufferInit((tUSBBuffer *)&g_sRxBuffer);
     // 初始化 Bulk 设备
    USBDBulkInit(0, (tUSBDBulkDevice *)&g_sBulkDevice);
    while(1)
    {
         if(g_ulFlags & COMMAND_STATUS_UPDATE)
         {
           //清除更新标志
           g_ulFlags &= ~COMMAND_STATUS_UPDATE;
             GPIOPinWrite(GPIO_PORTF_BASE,0x30,0x30);
         }
         // 发送完成
        if(ulTxCount != g_ulTxCount)
        {
           ulTxCount = g_ulTxCount;
             GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x10);
             //usnprintf(pcBuffer, 16, " %d ", ulTxCount);
        }
        // 接收完成
        if(ulRxCount != g_ulRxCount)
        {
           ulRxCount = g_ulRxCount;
             GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
        }
    }
}
```