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

# 第八章 CDC 设备

## 8.1 CDC 设备介绍

USB 的 CDC 类是 USB 通信设备类（Communication Device Class）的简称。CDC 类是 USB 组织定义的一类专门给各种通信设备(电信通信设备和中速网络通信设备)使用的 USB 子类。根据 CDC 类所针对通信设备的不同，CDC 类又被分成以下不同的模型：USB 传统纯电话业务（POTS）模型，USB ISDN 模型和 USB 网络模型。通常一个 CDC 类又由两个接口子类组成通信接口类（Communication Interface Class）和数据接口类(Data Interface Class)。通信接口类对设备进行管理和控制，而数据接口类传送数据。这两个接口子类占有不同数量和类型的终端点（Endpoints），不同 CDC 类模型，其所对应的接口的终端点需求也是不同的。

## 8.2 CDC 数据类型

usbdcdc.h 中已经定义好 CDC 设备类中使用的所有数据类型和函数，同时也会使用 Buffer 数据类型及 API 函数，第 7 章有介绍 Buffer 数据类型及 API 函数，下面只介绍 CDC 设备类使用的数据类型。

```
typedef enum
{
    //CDC 状态没定义
    CDC_STATE_UNCONFIGURED,
    //空闲状态
    CDC_STATE_IDLE,
    //等待数据发送或者结束
    CDC_STATE_WAIT_DATA,
    //等待数据处理.
    CDC_STATE_WAIT_CLIENT
} tCDCState;
```

tCDCState，定义 CDC 端点状态。定义在 usbdcdc.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;
     //CDC 接收端点状态
    volatile tCDCState eCDCRxState;
     //CDC 发送端点状态
    volatile tCDCState eCDCTxState;
     //CDC 请求状态
    volatile tCDCState eCDCRequestState;
     //CDC 中断状态
    volatile tCDCState eCDCInterruptState;
     //请求更新标志
     volatile unsigned char ucPendingRequest;
     //暂时结束
     unsigned short usBreakDuration;
     //控制
     unsigned short usControlLineState;
     //UART 状态
     unsigned short usSerialState;
     //标志位
    volatile unsigned short usDeferredOpFlags;
    //最后一次发送数据大小
    unsigned short usLastTxSize;
    //UART 控制参数
     tLineCoding sLineCoding;
     //接收数据
     volatile tBoolean bRxBlocked;
     //控制数据
     volatile tBoolean bControlBlocked;
     //连接是否成功
    volatile tBoolean bConnected;
     //控制端点
    unsigned char ucControlEndpoint;
     //Bulk IN 端点
    unsigned char ucBulkINEndpoint;
     // Bulk Out 端点
    unsigned char ucBulkOUTEndpoint;
     //接口控制
    unsigned char ucInterfaceControl;
     //接口数据
    unsigned char ucInterfaceData;
}
```

```
tCDCInstance;
```

tCDCInstance，CDC 设备类实例。定义了 CDC 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;
    //控制回调函数
    tUSBCallback pfnControlCallback;
    //控制回调函数的第一个参数
    void *pvControlCBData;
    //接收回调函数
    tUSBCallback pfnRxCallback;
    //接收回调函数的第一个参数
    void *pvRxCBData;
    //发送回调函数
    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;
    //CDC 类实例
    tCDCSerInstance *psPrivateCDCSerData;
}
tUSBDCDCDevice;
```

tUSBDCDCDevice，CDC 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 CDC 设备类实例。其它设备描述符、配置信息通过 API 函数储入 tCDCSerInstance 定义的 CDC 设备实例中。

### 8.3 API 函数

在 CDC 设备类 API 库中定义了 13 个函数，完成 USB CDC 设备初始化、配置及数据处理。以及 11 个 Buffer 操作函数，Buffer 第 7 章有介绍。下面为 usbdcdc.h 中定义的 API 函数：

```
void *USBDCDCInit(unsigned long ulIndex,
                  const tUSBDCDCDevice *psCDCDevice);
void * USBDCDCCompositeInit(unsigned long ulIndex,
                            const tUSBDCDCDevice *psCDCDevice);
unsigned long USBDCDCTxPacketAvailable(void *pvInstance);
```

```
unsigned long USBDCDCPacketWrite(void *pvInstance,
                                 unsigned char *pcData,
                                 unsigned long ulLength,
                                 tBoolean bLast);
unsigned long USBDCDCRxPacketAvailable(void *pvInstance);
unsigned long USBDCDCPacketRead(void *pvInstance,
                                unsigned char *pcData,
                                unsigned long ulLength,
                                tBoolean bLast);
void USBDCDCSerialStateChange(void *pvInstance,
                              unsigned short usState);
void USBDCDCTerm(void *pvInstance);
void *USBDCDCSetControlCBData(void *pvInstance, void *pvCBData);
void *USBDCDCSetRxCBData(void *pvInstance, void *pvCBData);
void *USBDCDCSetTxCBData(void *pvInstance, void *pvCBData);
void USBDCDCPowerStatusSet(void *pvInstance, unsigned char ucPower);
tBoolean USBDCDCRemoteWakeupRequest(void *pvInstance);
```

```
void *USBDCDCInit(unsigned long ulIndex,
                  const tUSBDCDCDevice *psCDCDevice);
```

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

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

返回：指向配置后的 tUSBDCDCDevice。

```
void * USBDCDCCompositeInit(unsigned long ulIndex,
                            const tUSBDCDCDevice *psCDCDevice);
```

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

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

返回：指向配置后的 tUSBDCDCDevice。

```
unsigned long USBDCDCTxPacketAvailable(void *pvInstance);
```

作用：获取可用发送数据长度。

参数：pvInstance，tUSBDCDCDevice 设备指针。

返回：发送包大小，用发送数据长度。

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

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

参数：pvInstance，tUSBDCDCDevice 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 USBDCDCRxPacketAvailable(void *pvInstance);
```

作用：获取接收数据长度。

参数：pvInstance，tUSBDCDCDevice 设备指针。

返回：可用接收数据个数，可读取的有效数据。

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

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

参数：pvInstance，tUSBDCDCDevice 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。

void USBDCDCSerialStateChange(void *pvInstance,
unsigned short usState);

作用：UART 收到数据后，调用此函数进行数据处理。

参数：pvInstance，tUSBDCDCDevice 设备指针。usState，UART 状态。

返回：无。

void USBDCDCTerm(void *pvInstance);

作用：结束 CDC 设备。

参数：pvInstance，指向 tUSBDCDCDevice。

返回：无。

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

作用：改变控制回调函数的第一个参数。

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

返回：旧参数指针。

void *USBDCDCSetRxCBData(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 tUSBDCDCDevice。pvCBData，用于替换的参数

返回：旧参数指针。

void *USBDCDCSetTxCBData(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 tUSBDCDCDevice。pvCBData，用于替换的参数

返回：旧参数指针。

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

作用：修改电源属性、状态。

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

返回：无。

tBoolean USBDCDCRemoteWakeupRequest(void *pvInstance);

作用：唤醒请求。

参数：pvInstance，指向 tUSBDCDCDevice。

返回：无。

在这些函数中 USBDCDCInit 和 USBDCDCPacketWrite、USBDCDCPacketRead、USBDCDCTxPacketAvailable、USBDCDCRxPacketAvailable 函数最重要并且使用最多，USBDCDCInit 第一次使用 CDC 设备时，用于初始化 CDC 设备的配置与控制。USBDCDCPacketRead、USBDCDCPacketWrite、USBDCDCTxPacketAvailable、USBDCDCRxPacketAvailable 为 CDC 传输数据的底层驱动函数用于驱动 Buffer。

在 CDC 类中也会使用到 11 个 Buffer 处理函数，用于 Buffer 数据接收、发送及处理。在 CDC 传输中大量使用。使用方法在第 7 章有讲解。

## 8.4 CDC 设备开发

CDC 设备开发只需要 4 步就能完成。如图 2 所示，CDC 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。

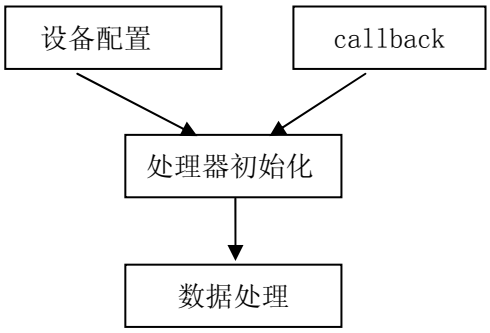



图 2

下面以“USB 转 UART”实例说明使用 USB 库开发 USB CDC 类过程：

第一步：CDC 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 CDC 设备配置。

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/timer.h"
#include "driverlib/uart.h"
#include "driverlib/usb.h"
#include "usblib/usblib.h"
#include "usblib/usbcdc.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdcdc.h"

#define UART_BUFFER_SIZE 256
volatile unsigned long g_ulUARTTxCount = 0;
volatile unsigned long g_ulUARTRxCount = 0;

// UART 设置.
#define USB_UART_BASE           UART0_BASE
#define USB_UART_PERIPH         SYSCTL_PERIPH_UART0
#define USB_UART_INT            INT_UART0
```

```
#define TX_GPIO_BASE            GPIO_PORTA_BASE
#define TX_GPIO_PERIPH          SYSCTL_PERIPH_GPIOA
#define TX_GPIO_PIN             GPIO_PIN_1
#define RX_GPIO_BASE            GPIO_PORTA_BASE
#define RX_GPIO_PERIPH          SYSCTL_PERIPH_GPIOA
#define RX_GPIO_PIN             GPIO_PIN_0
#define DEFAULT_BIT_RATE        115200
#define DEFAULT_UART_CONFIG     (UART_CONFIG_WLEN_8 | UART_CONFIG_PAR_NONE | \
                                 UART_CONFIG_STOP_ONE)
// 发送中断标志.
static tBoolean g_bSendingBreak = false;
//系统时钟
volatile unsigned long g_ulSysTickCount = 0;
// g_ulFlags 使用的标志位.
#define COMMAND_PACKET_RECEIVED 0x00000001
#define COMMAND_STATUS_UPDATE   0x00000002
// 全局标志
volatile unsigned long g_ulFlags = 0;
//状态
char *g_pcStatus;
// 全局 USB 配置标志.
static volatile tBoolean g_bUSBConfigured = false;
unsigned long RxHandler(void *pvCBData, unsigned long ulEvent,
                        unsigned long ulMsgValue, void *pvMsgData);
unsigned long TxHandler(void *pvCBData, unsigned long ulEvent,
                        unsigned long ulMsgValue, void *pvMsgData);
unsigned long ControlHandler(void *pvCBData, unsigned long ulEvent,
                             unsigned long ulMsgValue, void *pvMsgData);

void USBUARTPrimeTransmit(unsigned long ulBase);
void CheckForSerialStateChange(const tUSBDCDCDevice *psDevice, long lErrors);
void SetControlLineState(unsigned short usState);
tBoolean SetLineCoding(tLineCoding *psLineCoding);
void GetLineCoding(tLineCoding *psLineCoding);
void SendBreak(tBoolean bSend);

const tUSBBuffer g_sTxBuffer;
const tUSBBuffer g_sRxBuffer;
const tUSBDCDCDevice g_sCDCDevice;
unsigned char g_pucUSBTxBuffer[];
unsigned char g_pucUSBRxBuffer[];

//*****************************************************************************
// 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[] =
{
    2 + (16 * 2),
    USB_DTYPE_STRING,
    'V', 0, 'i', 0, 'r', 0, 't', 0, 'u', 0, 'a', 0, 'l', 0, ' ', 0,
    'C', 0, 'O', 0, 'M', 0, ' ', 0, 'P', 0, 'o', 0, 'r', 0, 't', 0
};
//*****************************************************************************
//   产品 序列号 描述符
//*****************************************************************************
const unsigned char g_pSerialNumberString[] =
{
    2 + (8 * 2),
    USB_DTYPE_STRING,
    '1', 0, '2', 0, '3', 0, '4', 0, '5', 0, '6', 0, '7', 0, '9', 0
};
//*****************************************************************************
// 设备接口字符串描述符
//*****************************************************************************
const unsigned char g_pControlInterfaceString[] =
{
    2 + (21 * 2),
    USB_DTYPE_STRING,
    'A', 0, 'C', 0, 'M', 0, ' ', 0, 'C', 0, 'o', 0, 'n', 0, 't', 0,
```

```
    'r', 0, 'o', 0, 'l', 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[] =
{
    2 + (26 * 2),
    USB_DTYPE_STRING,
    'S', 0, 'e', 0, 'l', 0, 'f', 0, ' ', 0, 'P', 0, 'o', 0, 'w', 0,
    'e', 0, 'r', 0, 'e', 0, 'd', 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_pControlInterfaceString,
    g_pConfigString
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /                \
                                sizeof(unsigned char *))
//*****************************************************************************
// 定义 CDC 设备实例
//*****************************************************************************
tCDCSerInstance g_sCDCInstance;
//*****************************************************************************
// 定义 CDC 设备
//*****************************************************************************
const tUSBDCDCDevice g_sCDCDevice =
{
    0x1234,
    USB_PID_SERIAL,
    0,
    USB_CONF_ATTR_SELF_PWR,
    ControlHandler,
    (void *)&g_sCDCDevice,
```

```
        USBBufferEventCallback,
        (void *)&g_sRxBuffer,
        USBBufferEventCallback,
        (void *)&g_sTxBuffer,
        g_pStringDescriptors,
        NUM_STRING_DESCRIPTORS,
        &g_sCDCInstance
    };
    //*****************************************************************************
    // 定义 Buffer
    //*****************************************************************************
    unsigned char g_pcUSBRxBuffer[UART_BUFFER_SIZE];
    unsigned char g_pucRxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
    unsigned char g_pcUSBTxBuffer[UART_BUFFER_SIZE];
    unsigned char g_pucTxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
    const tUSBBuffer g_sRxBuffer =
    {
        false,                          // This is a receive buffer.
        RxHandler,                      // pfnCallback
        (void *)&g_sCDCDevice,          // Callback data is our device pointer.
        USBDCDCPacketRead,              // pfnTransfer
        USBDCDCRxPacketAvailable,       // pfnAvailable
        (void *)&g_sCDCDevice,          // pvHandle
        g_pcUSBRxBuffer,                // pcBuffer
        UART_BUFFER_SIZE,               // ulBufferSize
        g_pucRxBufferWorkspace          // pvWorkspace
    };
    const tUSBBuffer g_sTxBuffer =
    {
        true,                           // This is a transmit buffer.
        TxHandler,                      // pfnCallback
        (void *)&g_sCDCDevice,          // Callback data is our device pointer.
        USBDCDCPacketWrite,             // pfnTransfer
        USBDCDCTxPacketAvailable,       // pfnAvailable
        (void *)&g_sCDCDevice,          // pvHandle
        g_pcUSBTxBuffer,                // pcBuffer
        UART_BUFFER_SIZE,               // ulBufferSize
        g_pucTxBufferWorkspace          // pvWorkspace
    };
```

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。CDC 设备接收回调函数包含以下事务：USB_EVENT_RX_AVAILABLE、USB_EVENT_DATA_REMAINING、USB_EVENT_REQUEST_BUFFER；CDC 设备发送回调函数包含了以下事务：USB_EVENT_TX_COMPLETE；CDC 设备控制回调函数包含了以下事务：USB_EVENT_CONNECTED、USB_EVENT_DISCONNECTED、USBD_CDC_EVENT_GET_LINE_CODING、

USBD_CDC_EVENT_SET_LINE_CODING 、 USBD_CDC_EVENT_SET_CONTROL_LINE_STATE 、 USBD_CDC_EVENT_SEND_BREAK 、 USBD_CDC_EVENT_CLEAR_BREAK 、 USB_EVENT_SUSPEND 、 USB_EVENT_RESUME。如下表：

| 名称 | 属性 | 说明 |
|---|---|---|
| USB_EVENT_RX_AVAILABLE | 接收 | 有数据可接收 |
| USB_EVENT_DATA_REMAINING | 接收 | 剩余数据 |
| USB_EVENT_REQUEST_BUFFER | 接收 | 请求 Buffer |
| USB_EVENT_TX_COMPLETE | 发送 | 发送完成 |
| USB_EVENT_RESUME | 控制 | 唤醒 |
| USB_EVENT_SUSPEND | 控制 | 挂起 |
| USBD_CDC_EVENT_CLEAR_BREAK | 控制 | 清除 Break 信号 |
| USBD_CDC_EVENT_SEND_BREAK | 控制 | 发送 Break 信号 |
| USBD_CDC_EVENT_SET_CONTROL_LINE_STATE | 控制 | 控制信号 |
| USBD_CDC_EVENT_SET_LINE_CODING | 控制 | 配置 UART 通信参数 |
| USBD_CDC_EVENT_GET_LINE_CODING | 控制 | 获取 UART 通信参数 |
| USB_EVENT_DISCONNECTED | 控制 | 断开 |
| USB_EVENT_CONNECTED | 控制 | 连接 |

表 2. CDC 事务

根据以上事务编写 Callback 函数：

```
//*****************************************************************************
//CDC 设备类控制回调函数
//*****************************************************************************
unsigned long  ControlHandler(void *pvCBData, unsigned long ulEvent,
               unsigned long ulMsgValue, void *pvMsgData)
{
    unsigned long ulIntsOff;
    // 判断处理事务
    switch(ulEvent)
    {
        //连接成功
        case USB_EVENT_CONNECTED:
            g_bUSBConfigured = true;
            //清空 Buffer。
            USBBufferFlush(&g_sTxBuffer);
            USBBufferFlush(&g_sRxBuffer);
            // 更新状态.
            ulIntsOff = IntMasterDisable();
            g_pcStatus = "Host connected.";
            g_ulFlags |= COMMAND_STATUS_UPDATE;
            if(!ulIntsOff)
            {
                IntMasterEnable();
            }
```

```
            break;
        //断开连接.
        case USB_EVENT_DISCONNECTED:
            g_bUSBConfigured = false;
            ulIntsOff = IntMasterDisable();
            g_pcStatus = "Host disconnected.";
            g_ulFlags |= COMMAND_STATUS_UPDATE;
            if(!ulIntsOff)
            {
                IntMasterEnable();
            }
            break;
        // 获取 UART 通信参数.
        case USBD_CDC_EVENT_GET_LINE_CODING:
            GetLineCoding(pvMsgData);
            break;
        //设置 UART 通信参数。
        case USBD_CDC_EVENT_SET_LINE_CODING:
            SetLineCoding(pvMsgData);
            break;
        // 设置 RS232 RTS 和 DTR.
        case USBD_CDC_EVENT_SET_CONTROL_LINE_STATE:
            SetControlLineState((unsigned short)ulMsgValue);
            break;
        // 发送 Break 信号
        case USBD_CDC_EVENT_SEND_BREAK:
            SendBreak(true);
            break;
        // 清除 Break 信号
        case USBD_CDC_EVENT_CLEAR_BREAK:
            SendBreak(false);
            break;
        // 挂起与唤醒事务
        case USB_EVENT_SUSPEND:
        case USB_EVENT_RESUME:
            break;
        default:
            break;
    }
    return(0);
}

//*****************************************************************************
//CDC 设备类 发送回调函数
```

```
//*****************************************************************************
unsigned long TxHandler(void *pvCBData, unsigned long ulEvent, unsigned long ulMsgValue,
          void *pvMsgData)
{
    switch(ulEvent)
    {
         //发送结束，在此不用处理数据
        case USB_EVENT_TX_COMPLETE:
            break;
        default:
            break;
    }
    return(0);
}
//*****************************************************************************
//CDC 设备类 发送回调函数
//*****************************************************************************
unsigned long TxHandler(void *pvCBData, unsigned long ulEvent, unsigned long ulMsgValue,
          void *pvMsgData)
{
    switch(ulEvent)
    {
         //发送结束，在此不用处理数据
        case USB_EVENT_TX_COMPLETE:
            break;
        default:
            break;
    }
    return(0);
}
//*****************************************************************************
//CDC 设备类 接收回调函数
//*****************************************************************************
unsigned long RxHandler(void *pvCBData, unsigned long ulEvent, unsigned long ulMsgValue,
          void *pvMsgData)
{
    unsigned long ulCount;
    //判断事务类型
    switch(ulEvent)
    {
        //接收数据
        case USB_EVENT_RX_AVAILABLE:
        {
            //UART 接收数据并能过 USB 发给主机。
```

```
            USBUARTPrimeTransmit(USB_UART_BASE);
            UARTIntEnable(USB_UART_BASE, UART_INT_TX);
            break;
        }
        // 检查剩余数据
        case USB_EVENT_DATA_REMAINING:
        {
            ulCount = UARTBusy(USB_UART_BASE) ? 1 : 0;
            return(ulCount);
        }
        //请求 Buffer
        case USB_EVENT_REQUEST_BUFFER:
        {
            return(0);
        }
        default:
            break;
    }

    return(0);
}
//*****************************************************************************
// 设置 RS232 RTS 和 DTR.
//*****************************************************************************
void SetControlLineState(unsigned short usState)
{
    // 根据 MCU 引脚自行添加。
}
//*****************************************************************************
// 设置 UART 通信参数
//*****************************************************************************
tBoolean SetLineCoding(tLineCoding *psLineCoding)
{
    unsigned long ulConfig;
    tBoolean bRetcode;
    bRetcode = true;
    // 数据长度
    switch(psLineCoding->ucDatabits)
    {
        case 5:
        {
            ulConfig = UART_CONFIG_WLEN_5;
            break;
        }
```

```
        case 6:
        {
            ulConfig = UART_CONFIG_WLEN_6;
            break;
        }
        case 7:
        {
            ulConfig = UART_CONFIG_WLEN_7;
            break;
        }
        case 8:
        {
            ulConfig = UART_CONFIG_WLEN_8;
            break;
        }
        default:
        {
            ulConfig = UART_CONFIG_WLEN_8;
            bRetcode = false;
            break;
        }
    }
    // 效验位
    switch(psLineCoding->ucParity)
    {
        case USB_CDC_PARITY_NONE:
        {
            ulConfig |= UART_CONFIG_PAR_NONE;
            break;
        }
        case USB_CDC_PARITY_ODD:
        {
            ulConfig |= UART_CONFIG_PAR_ODD;
            break;
        }
        case USB_CDC_PARITY_EVEN:
        {
            ulConfig |= UART_CONFIG_PAR_EVEN;
            break;
        }
        case USB_CDC_PARITY_MARK:
        {
            ulConfig |= UART_CONFIG_PAR_ONE;
            break;
```

```
        }
        case USB_CDC_PARITY_SPACE:
        {
            ulConfig |= UART_CONFIG_PAR_ZERO;
            break;
        }
        default:
        {
            ulConfig |= UART_CONFIG_PAR_NONE;
            bRetcode = false;
            break;
        }
    }
    //停止位
    switch(psLineCoding->ucStop)
    {
        case USB_CDC_STOP_BITS_1:
        {
            ulConfig |= UART_CONFIG_STOP_ONE;
            break;
        }
        case USB_CDC_STOP_BITS_2:
        {
            ulConfig |= UART_CONFIG_STOP_TWO;
            break;
        }
        default:
        {
            ulConfig = UART_CONFIG_STOP_ONE;
            bRetcode |= false;
            break;
        }
    }
    UARTConfigSetExpClk(USB_UART_BASE, SysCtlClockGet(), psLineCoding->ulRate,
                        ulConfig);
    return(bRetcode);
}

//*****************************************************************************
// 获取 UART 通信参数.
//*****************************************************************************
void GetLineCoding(tLineCoding *psLineCoding)
{
    unsigned long ulConfig;
```

```
unsigned long ulRate;
UARTConfigGetExpClk(USB_UART_BASE, SysCtlClockGet(), &ulRate,
                    &ulConfig);
psLineCoding->ulRate = ulRate;
//发送数据长度
switch(ulConfig & UART_CONFIG_WLEN_MASK)
{
    case UART_CONFIG_WLEN_8:
    {
        psLineCoding->ucDatabits = 8;
        break;
    }
    case UART_CONFIG_WLEN_7:
    {
        psLineCoding->ucDatabits = 7;
        break;
    }
    case UART_CONFIG_WLEN_6:
    {
        psLineCoding->ucDatabits = 6;
        break;
    }
    case UART_CONFIG_WLEN_5:
    {
        psLineCoding->ucDatabits = 5;
        break;
    }
}
// 校验位
switch(ulConfig & UART_CONFIG_PAR_MASK)
{
    case UART_CONFIG_PAR_NONE:
    {
        psLineCoding->ucParity = USB_CDC_PARITY_NONE;
        break;
    }
    case UART_CONFIG_PAR_ODD:
    {
        psLineCoding->ucParity = USB_CDC_PARITY_ODD;
        break;
    }
    case UART_CONFIG_PAR_EVEN:
    {
        psLineCoding->ucParity = USB_CDC_PARITY_EVEN;
```

```
            break;
        }
        case UART_CONFIG_PAR_ONE:
        {
            psLineCoding->ucParity = USB_CDC_PARITY_MARK;
            break;
        }
        case UART_CONFIG_PAR_ZERO:
        {
            psLineCoding->ucParity = USB_CDC_PARITY_SPACE;
            break;
        }
    }
    //停止位
    switch(ulConfig & UART_CONFIG_STOP_MASK)
    {
        case UART_CONFIG_STOP_ONE:
        {
            psLineCoding->ucStop = USB_CDC_STOP_BITS_1;
            break;
        }
        case UART_CONFIG_STOP_TWO:
        {
            psLineCoding->ucStop = USB_CDC_STOP_BITS_2;
            break;
        }
    }
}
//*****************************************************************************
// UART 发送 Break 信号
//*****************************************************************************
void SendBreak(tBoolean bSend)
{
    if(!bSend)
    {
        UARTBreakCtl(USB_UART_BASE, false);
        g_bSendingBreak = false;
    }
    else
    {
        UARTBreakCtl(USB_UART_BASE, true);
        g_bSendingBreak = true;
    }
}
```

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，CDC 传输接收的数据发送给主机。原理图如图 3 所示：

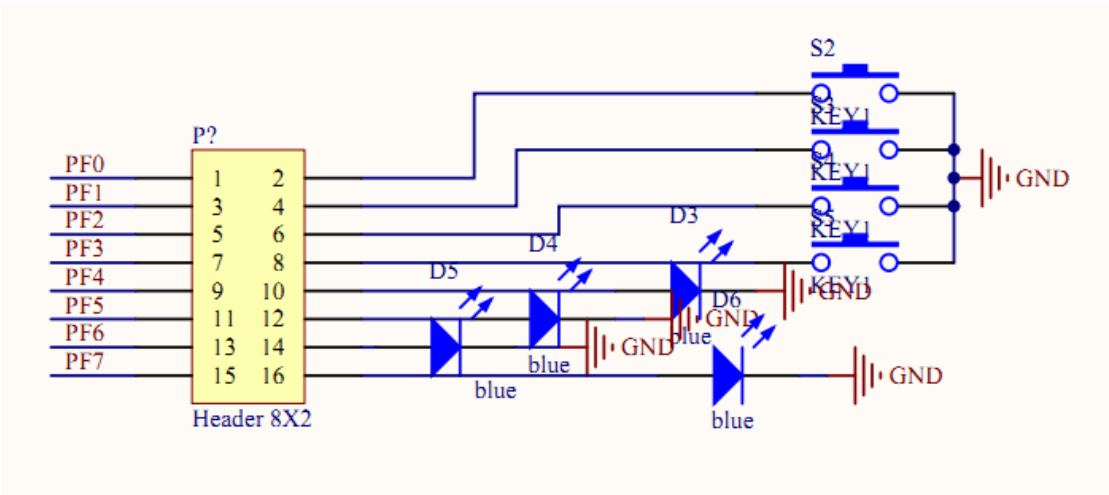



图 3

系统初始化：

```
    unsigned long ulTxCount;
    unsigned long ulRxCount;
    //    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;

   g_bUSBConfigured = false;
   //UART 配置
   SysCtlPeripheralEnable(USB_UART_PERIPH);
   SysCtlPeripheralEnable(TX_GPIO_PERIPH);
   SysCtlPeripheralEnable(RX_GPIO_PERIPH);
   GPIOPinTypeUART(TX_GPIO_BASE, TX_GPIO_PIN);
   GPIOPinTypeUART(RX_GPIO_BASE, RX_GPIO_PIN);
   UARTConfigSetExpClk(USB_UART_BASE, SysCtlClockGet(), DEFAULT_BIT_RATE,
                       DEFAULT_UART_CONFIG);
   UARTFIFOLevelSet(USB_UART_BASE, UART_FIFO_TX4_8, UART_FIFO_RX4_8);
   // 配置和使能 UART 中断.
   UARTIntClear(USB_UART_BASE, UARTIntStatus(USB_UART_BASE, false));
   UARTIntEnable(USB_UART_BASE, (UART_INT_OE | UART_INT_BE | UART_INT_PE |
                 UART_INT_FE | UART_INT_RT | UART_INT_TX | UART_INT_RX));
```

```
    // 初始化发送与接收 Buffer.
    USBBufferInit((tUSBBuffer *)&g_sTxBuffer);
    USBBufferInit((tUSBBuffer *)&g_sRxBuffer);
    // 初始化 CDC 设备
    USBDCDCInit(0, (tUSBDCDCDevice *)&g_sCDCDevice);

     ulRxCount = 0;
    ulTxCount = 0;
    IntEnable(USB_UART_INT);
```

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 CDC 传输中大量使用。

```
//*****************************************************************************
//CheckForSerialStateChange 在接收到串行数据后，处理标志。
//*****************************************************************************
void CheckForSerialStateChange(const tUSBDCDCDevice *psDevice, long lErrors)
{
    unsigned short usSerialState;
    // 设置 TXCARRIER (DSR)和 RXCARRIER (DCD)位.
    usSerialState = USB_CDC_SERIAL_STATE_TXCARRIER |
                    USB_CDC_SERIAL_STATE_RXCARRIER;
    // 判断是什么标志
    if(lErrors)
    {
        if(lErrors & UART_DR_OE)
        {
            usSerialState |= USB_CDC_SERIAL_STATE_OVERRUN;
        }
        if(lErrors & UART_DR_PE)
        {
            usSerialState |= USB_CDC_SERIAL_STATE_PARITY;
        }
        if(lErrors & UART_DR_FE)
        {
            usSerialState |= USB_CDC_SERIAL_STATE_FRAMING;
        }
        if(lErrors & UART_DR_BE)
        {
            usSerialState |= USB_CDC_SERIAL_STATE_BREAK;
        }
        // 改变状态。
        USBDCDCSerialStateChange((void *)psDevice, usSerialState);
    }
```

```
}
//*****************************************************************************
//从 UART 中读取数据，并放在 CDC 设备 Buffer 中发送给 USB 主机
//*****************************************************************************
long ReadUARTData(void)
{
    long lChar, lErrors;
    unsigned char ucChar;
    unsigned long ulSpace;
    lErrors = 0;
    //检查有多少可用空间
    ulSpace = USBBufferSpaceAvailable((tUSBBuffer *)&g_sTxBuffer);
    //从 UART 中读取数据并写入到 CDC 设备类的 Buffer 中发送
    while(ulSpace && UARTCharsAvail(USB_UART_BASE))
    {
        //读一个字节
        lChar = UARTCharGetNonBlocking(USB_UART_BASE);
        //是不是控制或错误标志 。
        if(!(lChar & ~0xFF))
        {
            ucChar = (unsigned char)(lChar & 0xFF);
            USBBufferWrite((tUSBBuffer *)&g_sTxBuffer,
                           (unsigned char *)&ucChar, 1);
            ulSpace--;
        }
        else
        {
            lErrors |= lChar;
        }
        g_ulUARTRxCount++;
    }
    return(lErrors);
}
//*****************************************************************************
// 从 Buffer 中读取数据，通过 UART 发送
//*****************************************************************************
void USBUARTPrimeTransmit(unsigned long ulBase)
{
    unsigned long ulRead;
    unsigned char ucChar;
    if(g_bSendingBreak)
    {
        return;
    }
```

```
    //检查 UART 中可用空间
    while(UARTSpaceAvail(ulBase))
    {
        //从 Buffer 中读取一个字节.
        ulRead = USBBufferRead((tUSBBuffer *)&g_sRxBuffer, &ucChar, 1);
        if(ulRead)
        {
            // 放在 UART TXFIFO 中发送.
            UARTCharPutNonBlocking(ulBase, ucChar);
              g_ulUARTTxCount++;
        }
        else
        {
            return;
        }
    }
}

//*****************************************************************************
// UART 中断处理函数
//*****************************************************************************
void USBUARTIntHandler(void)
{
    unsigned long ulInts;
    long lErrors;
    //获取中断标志并清除
    ulInts = UARTIntStatus(USB_UART_BASE, true);
    UARTIntClear(USB_UART_BASE, ulInts);
    // 发送中断
    if(ulInts & UART_INT_TX)
    {
        // 从 USB 中获取数据并能过 UART 发送.
        USBUARTPrimeTransmit(USB_UART_BASE);
        // If the output buffer is empty, turn off the transmit interrupt.
        if(!USBBufferDataAvailable(&g_sRxBuffer))
        {
            UARTIntDisable(USB_UART_BASE, UART_INT_TX);
        }
    }
    // 接收中断.
    if(ulInts & (UART_INT_RX | UART_INT_RT))
    {
        //从 UART 中读取数据并通过 Buffer 发送给 USB 主机。
        lErrors = ReadUARTData();
```

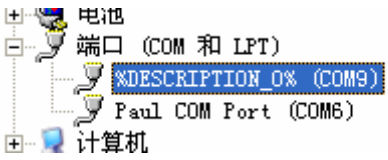
```
        //检查是否有控制信号或者错误信号。
        CheckForSerialStateChange(&g_sCDCDevice, lErrors);
    }
}
 while(1)
    {
         if(g_ulFlags & COMMAND_STATUS_UPDATE)
         {
           //清除更新标志，有数据更新。
           IntMasterDisable();
           g_ulFlags &= ~COMMAND_STATUS_UPDATE;
           IntMasterEnable();

             GPIOPinWrite(GPIO_PORTF_BASE,0x30,0x30);
         }
         // 发送完成
        if(ulTxCount != g_ulUARTTxCount)
        {
            ulTxCount = g_ulUARTTxCount;
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x10);
              //usnprintf(pcBuffer, 16, " %d ", ulTxCount);
        }
        // 接收完成
        if(ulRxCount != g_ulUARTTxCount)
        {
            ulRxCount = g_ulUARTRxCount;
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
        }
    }
```

使用上面四步就完成CDC设备开发。CDC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 中断服务函数和USBUARTIntHandler 中断服务程序。以上 UART→RS232 开发完成，在 Win xp 下运行效果如下图所示：


驱动安装

在枚举过程中可以看出，在电脑右下脚可以看到“Virtual Com Port”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“端口”中看到“DESCRIPTION_0”设备，如下图。现在 CDC 设备可以正式使用。




CDC 设备要配合上位机使用，上位机发送字符串通过 USB—>UART 设备转换后通过 UART 发送给其它 UART 设备。运行图如下：


CDC 设备 USB-->UART 开发源码较多，下面只列出一部分如下：

```
//*****************************************************************************
// 字符串描述符集合
//*****************************************************************************
const unsigned char * const g_pStringDescriptors[] =
{
    g_pLangDescriptor,
    g_pManufacturerString,
```

```
    g_pProductString,
    g_pSerialNumberString,
    g_pControlInterfaceString,
    g_pConfigString
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /               \
                                sizeof(unsigned char *))
//*****************************************************************************
// 定义 CDC 设备实例
//*****************************************************************************
tCDCSerInstance g_sCDCInstance;
//*****************************************************************************
// 定义 CDC 设备
//*****************************************************************************
const tUSBDCDCDevice g_sCDCDevice =
{
    0x1234,
    USB_PID_SERIAL,
    0,
    USB_CONF_ATTR_SELF_PWR,
    ControlHandler,
    (void *)&g_sCDCDevice,
    USBBufferEventCallback,
    (void *)&g_sRxBuffer,
    USBBufferEventCallback,
    (void *)&g_sTxBuffer,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    &g_sCDCInstance
};
//*****************************************************************************
// 定义 Buffer
//*****************************************************************************
unsigned char g_pcUSBRxBuffer[UART_BUFFER_SIZE];
unsigned char g_pucRxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
unsigned char g_pcUSBTxBuffer[UART_BUFFER_SIZE];
unsigned char g_pucTxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
const tUSBBuffer g_sRxBuffer =
{
    false,                          // This is a receive buffer.
    RxHandler,                      // pfnCallback
    (void *)&g_sCDCDevice,          // Callback data is our device pointer.
    USBDCDCPacketRead,              // pfnTransfer
    USBDCDCRxPacketAvailable,       // pfnAvailable
```

```
    (void *)&g_sCDCDevice,          // pvHandle
    g_pcUSBRxBuffer,                // pcBuffer
    UART_BUFFER_SIZE,               // ulBufferSize
    g_pucRxBufferWorkspace          // pvWorkspace
};
const tUSBBuffer g_sTxBuffer =
{
    true,                           // This is a transmit buffer.
    TxHandler,                      // pfnCallback
    (void *)&g_sCDCDevice,          // Callback data is our device pointer.
    USBDCDCPacketWrite,             // pfnTransfer
    USBDCDCTxPacketAvailable,       // pfnAvailable
    (void *)&g_sCDCDevice,          // pvHandle
    g_pcUSBTxBuffer,                // pcBuffer
    UART_BUFFER_SIZE,               // ulBufferSize
    g_pucTxBufferWorkspace          // pvWorkspace
};

//*****************************************************************************
//CheckForSerialStateChange 在接收到串行数据后，处理标志。
//*****************************************************************************
void CheckForSerialStateChange(const tUSBDCDCDevice *psDevice, long lErrors)
{
    unsigned short usSerialState;
    // 设置 TXCARRIER (DSR)和 RXCARRIER (DCD)位.
    usSerialState = USB_CDC_SERIAL_STATE_TXCARRIER |
                    USB_CDC_SERIAL_STATE_RXCARRIER;
    // 判断是什么标志
    if(lErrors)
    {
        if(lErrors & UART_DR_OE)
        {
            usSerialState |= USB_CDC_SERIAL_STATE_OVERRUN;
        }
        if(lErrors & UART_DR_PE)
        {
            usSerialState |= USB_CDC_SERIAL_STATE_PARITY;
        }
        if(lErrors & UART_DR_FE)
        {
            usSerialState |= USB_CDC_SERIAL_STATE_FRAMING;
        }
        if(lErrors & UART_DR_BE)
        {
```

```
            usSerialState |= USB_CDC_SERIAL_STATE_BREAK;
        }
        // 改变状态。
        USBDCDCSerialStateChange((void *)psDevice, usSerialState);
    }
}
//*****************************************************************************
//从 UART 中读取数据，并放在 CDC 设备 Buffer 中发送给 USB 主机
//*****************************************************************************
long ReadUARTData(void)
{
    long lChar, lErrors;
    unsigned char ucChar;
    unsigned long ulSpace;
    lErrors = 0;
    //检查有多少可用空间
    ulSpace = USBBufferSpaceAvailable((tUSBBuffer *)&g_sTxBuffer);
    //从 UART 中读取数据并写入到 CDC 设备类的 Buffer 中发送
    while(ulSpace && UARTCharsAvail(USB_UART_BASE))
    {
        //读一个字节
        lChar = UARTCharGetNonBlocking(USB_UART_BASE);
        //是不是控制或错误标志 。
        if(!(lChar & ~0xFF))
        {
            ucChar = (unsigned char)(lChar & 0xFF);
            USBBufferWrite((tUSBBuffer *)&g_sTxBuffer,
                           (unsigned char *)&ucChar, 1);
            ulSpace--;
        }
        else
        {
            lErrors |= lChar;
        }
        g_ulUARTRxCount++;
    }
    return(lErrors);
}
//*****************************************************************************
// 从 Buffer 中读取数据，通过 UART 发送
//*****************************************************************************
void USBUARTPrimeTransmit(unsigned long ulBase)
{
    unsigned long ulRead;
```

```
    unsigned char ucChar;
    if(g_bSendingBreak)
    {
        return;
    }
    //检查 UART 中可用空间
    while(UARTSpaceAvail(ulBase))
    {
        //从 Buffer 中读取一个字节.
        ulRead = USBBufferRead((tUSBBuffer *)&g_sRxBuffer, &ucChar, 1);
        if(ulRead)
        {
            // 放在 UART TXFIFO 中发送.
            UARTCharPutNonBlocking(ulBase, ucChar);
              g_ulUARTTxCount++;
        }
        else
        {
            return;
        }
    }
}
//*****************************************************************************
// 设置 RS232 RTS 和 DTR.
//*****************************************************************************
void SetControlLineState(unsigned short usState)
{
    // 根据 MCU 引脚自行添加。
}
//*****************************************************************************
// 设置 UART 通信参数
//*****************************************************************************
tBoolean SetLineCoding(tLineCoding *psLineCoding)
{
    unsigned long ulConfig;
    tBoolean bRetcode;
    bRetcode = true;
    // 数据长度
    switch(psLineCoding->ucDatabits)
    {
        case 5:
        {
            ulConfig = UART_CONFIG_WLEN_5;
            break;
```

```
    }
    case 6:
    {
        ulConfig = UART_CONFIG_WLEN_6;
        break;
    }
    case 7:
    {
        ulConfig = UART_CONFIG_WLEN_7;
        break;
    }
    case 8:
    {
        ulConfig = UART_CONFIG_WLEN_8;
        break;
    }
    default:
    {
        ulConfig = UART_CONFIG_WLEN_8;
        bRetcode = false;
        break;
    }
}
// 效验位
switch(psLineCoding->ucParity)
{
    case USB_CDC_PARITY_NONE:
    {
        ulConfig |= UART_CONFIG_PAR_NONE;
        break;
    }
    case USB_CDC_PARITY_ODD:
    {
        ulConfig |= UART_CONFIG_PAR_ODD;
        break;
    }
    case USB_CDC_PARITY_EVEN:
    {
        ulConfig |= UART_CONFIG_PAR_EVEN;
        break;
    }
    case USB_CDC_PARITY_MARK:
    {
        ulConfig |= UART_CONFIG_PAR_ONE;
```

```
            break;
        }
        case USB_CDC_PARITY_SPACE:
        {
            ulConfig |= UART_CONFIG_PAR_ZERO;
            break;
        }
        default:
        {
            ulConfig |= UART_CONFIG_PAR_NONE;
            bRetcode = false;
            break;
        }
    }
    //停止位
    switch(psLineCoding->ucStop)
    {
        case USB_CDC_STOP_BITS_1:
        {
            ulConfig |= UART_CONFIG_STOP_ONE;
            break;
        }
        case USB_CDC_STOP_BITS_2:
        {
            ulConfig |= UART_CONFIG_STOP_TWO;
            break;
        }
        default:
        {
            ulConfig = UART_CONFIG_STOP_ONE;
            bRetcode |= false;
            break;
        }
    }
    UARTConfigSetExpClk(USB_UART_BASE, SysCtlClockGet(), psLineCoding->ulRate,
                        ulConfig);
    return(bRetcode);
}

//*****************************************************************************
// 获取 UART 通信参数.
//*****************************************************************************
void GetLineCoding(tLineCoding *psLineCoding)
{
```

```
unsigned long ulConfig;
unsigned long ulRate;
UARTConfigGetExpClk(USB_UART_BASE, SysCtlClockGet(), &ulRate,
                    &ulConfig);
psLineCoding->ulRate = ulRate;
//发送数据长度
switch(ulConfig & UART_CONFIG_WLEN_MASK)
{
    case UART_CONFIG_WLEN_8:
    {
        psLineCoding->ucDatabits = 8;
        break;
    }
    case UART_CONFIG_WLEN_7:
    {
        psLineCoding->ucDatabits = 7;
        break;
    }
    case UART_CONFIG_WLEN_6:
    {
        psLineCoding->ucDatabits = 6;
        break;
    }
    case UART_CONFIG_WLEN_5:
    {
        psLineCoding->ucDatabits = 5;
        break;
    }
}
// 校验位
switch(ulConfig & UART_CONFIG_PAR_MASK)
{
    case UART_CONFIG_PAR_NONE:
    {
        psLineCoding->ucParity = USB_CDC_PARITY_NONE;
        break;
    }
    case UART_CONFIG_PAR_ODD:
    {
        psLineCoding->ucParity = USB_CDC_PARITY_ODD;
        break;
    }
    case UART_CONFIG_PAR_EVEN:
    {
```

```
            psLineCoding->ucParity = USB_CDC_PARITY_EVEN;
            break;
        }
        case UART_CONFIG_PAR_ONE:
        {
            psLineCoding->ucParity = USB_CDC_PARITY_MARK;
            break;
        }
        case UART_CONFIG_PAR_ZERO:
        {
            psLineCoding->ucParity = USB_CDC_PARITY_SPACE;
            break;
        }
    }
    //停止位
    switch(ulConfig & UART_CONFIG_STOP_MASK)
    {
        case UART_CONFIG_STOP_ONE:
        {
            psLineCoding->ucStop = USB_CDC_STOP_BITS_1;
            break;
        }
        case UART_CONFIG_STOP_TWO:
        {
            psLineCoding->ucStop = USB_CDC_STOP_BITS_2;
            break;
        }
    }
}
//*****************************************************************************
// UART 发送 Break 信号
//*****************************************************************************
void SendBreak(tBoolean bSend)
{
    if(!bSend)
    {
        UARTBreakCtl(USB_UART_BASE, false);
        g_bSendingBreak = false;
    }
    else
    {
        UARTBreakCtl(USB_UART_BASE, true);
        g_bSendingBreak = true;
    }
```

```
}
//*****************************************************************************
//CDC 设备类控制回调函数
//*****************************************************************************
unsigned long  ControlHandler(void *pvCBData, unsigned long ulEvent,
               unsigned long ulMsgValue, void *pvMsgData)
{
    unsigned long ulIntsOff;
    // 判断处理事务
    switch(ulEvent)
    {
        //连接成功
        case USB_EVENT_CONNECTED:
            g_bUSBConfigured = true;
            //清空 Buffer。
            USBBufferFlush(&g_sTxBuffer);
            USBBufferFlush(&g_sRxBuffer);
            // 更新状态.
            ulIntsOff = IntMasterDisable();
            g_pcStatus = "Host connected.";
            g_ulFlags |= COMMAND_STATUS_UPDATE;
            if(!ulIntsOff)
            {
                IntMasterEnable();
            }
            break;
        //断开连接.
        case USB_EVENT_DISCONNECTED:
            g_bUSBConfigured = false;
            ulIntsOff = IntMasterDisable();
            g_pcStatus = "Host disconnected.";
            g_ulFlags |= COMMAND_STATUS_UPDATE;
            if(!ulIntsOff)
            {
                IntMasterEnable();
            }
            break;
        // 获取 UART 通信参数.
        case USBD_CDC_EVENT_GET_LINE_CODING:
            GetLineCoding(pvMsgData);
            break;
        //设置 UART 通信参数。
        case USBD_CDC_EVENT_SET_LINE_CODING:
            SetLineCoding(pvMsgData);
```

```
            break;
        // 设置 RS232 RTS 和 DTR.
        case USBD_CDC_EVENT_SET_CONTROL_LINE_STATE:
            SetControlLineState((unsigned short)ulMsgValue);
            break;
        // 发送 Break 信号
        case USBD_CDC_EVENT_SEND_BREAK:
            SendBreak(true);
            break;
        // 清除 Break 信号
        case USBD_CDC_EVENT_CLEAR_BREAK:
            SendBreak(false);
            break;
        // 挂起与唤醒事务
        case USB_EVENT_SUSPEND:
        case USB_EVENT_RESUME:
            break;
        default:
            break;
    }
    return(0);
}
//*****************************************************************************
//CDC 设备类 发送回调函数
//*****************************************************************************
unsigned long TxHandler(void *pvCBData, unsigned long ulEvent, unsigned long ulMsgValue,
         void *pvMsgData)
{
    switch(ulEvent)
    {
         //发送结束，在此不用处理数据
        case USB_EVENT_TX_COMPLETE:
            break;
        default:
            break;
    }
    return(0);
}
//*****************************************************************************
//CDC 设备类 接收回调函数
//*****************************************************************************
unsigned long RxHandler(void *pvCBData, unsigned long ulEvent, unsigned long ulMsgValue,
         void *pvMsgData)
{
```

```
    unsigned long ulCount;
    //判断事务类型
    switch(ulEvent)
    {
        //接收数据
        case USB_EVENT_RX_AVAILABLE:
        {
            //UART 接收数据并能过 USB 发给主机。
            USBUARTPrimeTransmit(USB_UART_BASE);
            UARTIntEnable(USB_UART_BASE, UART_INT_TX);
            break;
        }
        // 检查剩余数据
        case USB_EVENT_DATA_REMAINING:
        {
            ulCount = UARTBusy(USB_UART_BASE) ? 1 : 0;
            return(ulCount);
        }
        //请求 Buffer
        case USB_EVENT_REQUEST_BUFFER:
        {
            return(0);
        }
        default:
            break;
    }

    return(0);
}

//*****************************************************************************
// UART 中断处理函数
//*****************************************************************************
void USBUARTIntHandler(void)
{
    unsigned long ulInts;
    long lErrors;
    //获取中断标志并清除
    ulInts = UARTIntStatus(USB_UART_BASE, true);
    UARTIntClear(USB_UART_BASE, ulInts);
    // 发送中断
    if(ulInts & UART_INT_TX)
    {
        // 从 USB 中获取数据并能过 UART 发送.
```

```
            USBUARTPrimeTransmit(USB_UART_BASE);
            // If the output buffer is empty, turn off the transmit interrupt.
            if(!USBBufferDataAvailable(&g_sRxBuffer))
            {
                UARTIntDisable(USB_UART_BASE, UART_INT_TX);
            }
        }
        // 接收中断.
        if(ulInts & (UART_INT_RX | UART_INT_RT))
        {
            //从 UART 中读取数据并通过 Buffer 发送给 USB 主机。
            lErrors = ReadUARTData();
            //检查是否有控制信号或者错误信号。
            CheckForSerialStateChange(&g_sCDCDevice, lErrors);
        }
    }
    //*****************************************************************************
    // 应用主函数.
    //*****************************************************************************
    int   main(void)
    {
        unsigned long ulTxCount;
        unsigned long ulRxCount;
    //   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;

         g_bUSBConfigured = false;
         //UART 配置
        SysCtlPeripheralEnable(USB_UART_PERIPH);
        SysCtlPeripheralEnable(TX_GPIO_PERIPH);
        SysCtlPeripheralEnable(RX_GPIO_PERIPH);
        GPIOPinTypeUART(TX_GPIO_BASE, TX_GPIO_PIN);
        GPIOPinTypeUART(RX_GPIO_BASE, RX_GPIO_PIN);
        UARTConfigSetExpClk(USB_UART_BASE, SysCtlClockGet(), DEFAULT_BIT_RATE,
                            DEFAULT_UART_CONFIG);
        UARTFIFOLevelSet(USB_UART_BASE, UART_FIFO_TX4_8, UART_FIFO_RX4_8);
        // 配置和使能 UART 中断.
```

```
    UARTIntClear(USB_UART_BASE, UARTIntStatus(USB_UART_BASE, false));
    UARTIntEnable(USB_UART_BASE, (UART_INT_OE | UART_INT_BE | UART_INT_PE |
                  UART_INT_FE | UART_INT_RT | UART_INT_TX | UART_INT_RX));


     // 初始化发送与接收 Buffer.
    USBBufferInit((tUSBBuffer *)&g_sTxBuffer);
    USBBufferInit((tUSBBuffer *)&g_sRxBuffer);
     // 初始化 CDC 设备
    USBDCDCInit(0, (tUSBDCDCDevice *)&g_sCDCDevice);

     ulRxCount = 0;
    ulTxCount = 0;
    IntEnable(USB_UART_INT);
    while(1)
    {
         if(g_ulFlags & COMMAND_STATUS_UPDATE)
         {
           //清除更新标志，有数据更新。
           IntMasterDisable();
           g_ulFlags &= ~COMMAND_STATUS_UPDATE;
           IntMasterEnable();

             GPIOPinWrite(GPIO_PORTF_BASE,0x30,0x30);
         }
         // 发送完成
        if(ulTxCount != g_ulUARTTxCount)
        {
            ulTxCount = g_ulUARTTxCount;
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x10);
              //usnprintf(pcBuffer, 16, " %d ", ulTxCount);
        }
        // 接收完成
        if(ulRxCount != g_ulUARTTxCount)
        {
            ulRxCount = g_ulUARTRxCount;
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
        }
    }
}
```