```c
#include "common.h"
#include "init_cpu.h"

#include "usblib/usblib.h"
#include "usbcdc.h"
#include "driverlib/usb.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdcdc.h"
#include "usb_descriptors.h"
#include "driverlib/sysctl.h"




// \*****************************************************************************
//
// Handles CDC driver notifications related to control and setup of the device.
//
// \param pvCBData is the client-supplied callback pointer for this channel.
// \param ulEvent identifies the event we are being notified about.
// \param ulMsgValue is an event-specific value.
// \param pvMsgData is an event-specific pointer.
//
// This function is called by the CDC driver to perform control-related
// operations on behalf of the USB host.  These functions include setting
// and querying the serial communication parameters, setting handshake line
// states and sending break conditions.
//
// \return The return value is event-specific.
//
// \*****************************************************************************

//-------------------------------------
//  CDC DRIVER CONTROL NOTIFICATIONS  |
//--------------------------------------------------------------------------------------------------------------------
//  This function is called by the CDC driver to perform control-related operations on behalf of the USB host.
//  Return value is ???

u32 ControlHandler(void *pvCBData, u32 Event, u32 ui32MsgValue, void *pvMsgData) {

    switch(Event) {

        case USB_EVENT_CONNECTED:

            USBBufferFlush(&UsbTxBuffStruct);                       // Flush the USB Buffers.
            USBBufferFlush(&UsbRxBuffStruct);
            break;

        case USB_EVENT_DISCONNECTED:

            // Do whatever we do here.
            break;

        case USBD_CDC_EVENT_GET_LINE_CODING:
        case USBD_CDC_EVENT_SET_LINE_CODING:
        case USBD_CDC_EVENT_SET_CONTROL_LINE_STATE:
        case USBD_CDC_EVENT_SEND_BREAK:
        case USBD_CDC_EVENT_CLEAR_BREAK:
        case USB_EVENT_SUSPEND:
        case USB_EVENT_RESUME:

            break;

        default:

            break;
    }
    return(0);
}
//--------------------------------------------------------------------------------------------------------------------
```

```c
// \*****************************************************************************
//
// Handles CDC driver notifications related to the receive channel (data from
// the USB host).
//
// \param pvCBData is the client-supplied callback data value for this channel.
// \param ui32Event identifies the event we are being notified about.
// \param ui32MsgValue is an event-specific value.
// \param pvMsgData is an event-specific pointer.
//
// This function is called by the CDC driver to notify us of any events
// related to operation of the receive data channel (the OUT channel carrying
// data from the USB host).
//
// \return The return value is event-specific.
//
// \*****************************************************************************

//-----------------------------------------------
//  DATA RECEIVED FROM LINUX COMPUTER VIA USB  |
//-------------------------------------------------------------------------------------------------------------------
//  Return value is Data Remaining.

u32 RxHandler(void *pvCBData, u32 Event, u32 MsgValue, void *pvMsgData) {

    u32 Count;

    switch(Event) {

        case USB_EVENT_RX_AVAILABLE:

            // Deal with received characters.
            break;

        //
        // We are being asked how much unprocessed data we have still to
        // process. We return 0 if the UART is currently idle or 1 if it is
        // in the process of transmitting something. The actual number of
        // bytes in the UART FIFO is not important here, merely whether or
        // not everything previously sent to us has been transmitted.
        //
        case USB_EVENT_DATA_REMAINING:

            //
            // Get the number of bytes in the buffer and add 1 if some data
            // still has to clear the transmitter.
            //
//          Count = UARTBusy(UART0_BASE) ? 1 : 0;
            return(Count);

        //
        // We are being asked to provide a buffer into which the next packet
        // can be read. We do not support this mode of receiving data so let
        // the driver know by returning 0. The CDC driver should not be sending
        // this message but this is included just for illustration and
        // completeness.
        //
        case USB_EVENT_REQUEST_BUFFER:

            break;

        default:

            break;
    }
    return(0);
}
//-------------------------------------------------------------------------------------------------------------------
```

```c
// \***************************************************************************
//
// Handles CDC driver notifications related to the transmit channel (data to
// the USB host).
//
// \param pvCBData is the client-supplied callback pointer for this channel.
// \param ui32Event identifies the event we are being notified about.
// \param ui32MsgValue is an event-specific value.
// \param pvMsgData is an event-specific pointer.
//
// This function is called by the CDC driver to notify us of any events
// related to operation of the transmit data channel (the IN channel carrying
// data to the USB host).
//
// \return The return value is event-specific.
//
// \***************************************************************************

//------------------------------------------------
//  DATA TRANSMITTED TO LINUX COMPUTER VIA USB  |
//-------------------------------------------------------------------------------------------------------------------

u32 TxHandler(void *pvCBData, u32 Event, u32 MsgValue, void *pvMsgData) {

    switch(Event) {

        case USB_EVENT_TX_COMPLETE:

            break;              // Since we're using the USBBuffer, don't need to do anything.

        default:

            break;
    }
    return(0);
}
//-------------------------------------------------------------------------------------------------------------------




//---------------
//  INITIALIZE  |
//-------------------------------------------------------------------------------------------------------------------
void Cold_Init_USB(void) {

    u32 PLLRate   = SysCtlVCOGet(SYSCTL_XTAL_25MHZ, &PLLRate);
    u32 ClockRate = GetCpuClk();

    USBStackModeSet(0, eUSBModeDevice, 0);                      // Device mode, VBUS monitoring.
    USBDCDFeatureSet(0, USBLIB_FEATURE_CPUCLK, &ClockRate);     // Set CPU Clock Rate.
    USBDCDFeatureSet(0, USBLIB_FEATURE_USBPLL, &PLLRate);       // Set PLL Rate.
    USBBufferInit(&UsbRxBuffStruct);                            // Initialize circular buffers.
    USBBufferInit(&UsbTxBuffStruct);                            //
    USBDCDCInit(0, (tUSBDCDCDevice *)&CdcDeviceStruct);         // Let Her Rip.
}
//-------------------------------------------------------------------------------------------------------------------
```