```c
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"        /* XPAR parameters */
#include "xusbps.h"             /* USB controller driver */
#include "xscugic.h"
#include "xusbps_cdc_ch9.h"     /* Generic Chapter 9 handling code */
#include "xusbps_cdc_ch9_cdc.h" /* Storage class handling code */
#include "xil_exception.h"
#include "xpseudo_asm.h"
#include "xreg_cortexa9.h"
#include "xil_cache.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "xusbps_cdc.h"
#include "xusbps_cdc_irq.h"     /* USB interrupt processing code */
#include "xusbps_cdc_buffer.h"


static int setup_interrupts(XScuGic *intc) {
    int status;
    XScuGic_Config *intc_config;

    intc_config = XScuGic_LookupConfig(XPAR_SCUGIC_SINGLE_DEVICE_ID);
    if (NULL == intc_config) {
        return XST_FAILURE;
    }

    status = XScuGic_CfgInitialize(intc, intc_config, intc_config->CpuBaseAddress);
    if (status != XST_SUCCESS) {
        return status;
    }

    Xil_ExceptionInit();

    /* Connect the GIC interrupt handler to the exception vector in the processor */
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_IRQ_INT,
                                 (Xil_ExceptionHandler)XScuGic_InterruptHandler,
                                 intc);

    return XST_SUCCESS;
}

static void reset_usb(void) {
    // Ensure that the PHY is out of reset
    volatile u32 *gpio_base;
    volatile u32 *gpio_oen;
    volatile u32 *gpio_dir;

    /* Ensure that the USB PHY is not in reset */
    gpio_base = (u32 *)0xE000A000;
    gpio_oen = (u32 *)0xE000A208;
    gpio_dir = (u32 *)0xE000A204;

    *(gpio_oen) |= 0x00000080;
    *(gpio_dir) |= 0x00000080;
    *gpio_base = 0xff7f0080;
}

int main()
{
    int status;
    XUsbPs usb;
    XScuGic intc;
    u8 text_buffer[256];
    u32 bytes;

    const char hello[13] = "Received ";
    const char hai[4]="\n\r";

    init_platform();
```

```
72
73      xil_printf("\n\n--- USB CDC ACM Test Application ---\n\n");
74
75      reset_usb();
76
77      // Set up and configure the interrupt system
78      status = setup_interrupts(&intc);
79      if (status != XST_SUCCESS) {
80          xil_printf("ERROR: Unable to initialize interrupt system: %d\n", status);
81          exit(1);
82      }
83
84      status = xusbps_cdc_register_interrupt(&intc, &usb, XPAR_PS7_USB_0_INTR);
85      if (status != XST_SUCCESS) {
86          xil_printf("ERROR: Unable to register USB interrupts: %d\n", status);
87          exit(1);
88      }
89
90      // Enable interrupts in the processor
91      Xil_ExceptionEnableMask(XIL_EXCEPTION_IRQ);
92
93      /* Initialize the USB controller */
94      status = xusb_cdc_init(&usb, XPAR_PS7_USB_0_DEVICE_ID, XPAR_PS7_USB_0_INTR, 64 *
        1024);
95      if (status != XST_SUCCESS) {
96          xil_printf("ERROR: Unable to set up USB controller: %d\n", status);
97          exit(1);
98      }
99
100         while(1) {
101
102                 bytes = xusb_cdc_rx_bytes_available() > 5 ? 5 :
                    xusb_cdc_rx_bytes_available();
103                 if (bytes != 0) {
104                     bytes = xusb_cdc_receive_data(text_buffer, bytes);
105                     text_buffer[bytes] = 0;
106                         xusb_cdc_send_data(&usb, (u8 *)hello, 9);
107                         xusb_cdc_send_data(&usb, (u8 *)text_buffer, 1);
108                         xusb_cdc_send_data(&usb, (u8 *)hai, 2);
109
110                         }
111         }
112
113     cleanup_platform();
114
115     return 0;
116 }
117
```