Application Note: AN00184

USB CDC Class as Virtual Serial Port - Extended on xCORE-200 Explorer

This application note shows how to create a USB device compliant to the standard USB Communications Device Class (CDC) on an XMOS multicore microcontroller.

The code associated with this application note provides an example of using the XMOS USB Device Library (XUD) and associated USB class descriptors to provide a framework for the creation of a USB CDC device that implements Abstract Control Model (ACM).

This example USB CDC ACM implementation provides a Virtual Serial port running over high speed USB. The Virtual Serial port supports the standard requests associated with ACM model of the class.

A serial terminal program from host PC connects to virtual serial port and interacts with the application. The application provides a menu to toggle on-board LEDs, read an I2C device, monitor buttons and loopback characters. This application demo code demonstrates a simple way in which USB CDC class devices can easily be deployed using an xCORE-200 device.

Note: This application note provides a standard USB CDC class device and as a result does not require external drivers to run on Windows, Mac or Linux.

This application note extends AN00124 to provide a virtual serial port application that interfaces to hardware demostrating how to build a system which allows a USB host to connect to custom hardware using an XMOS device.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0.0
- XMOS USB library - Version 2.0.0
- XMOS I2C library - Version 2.0.0

Required hardware

This application note is designed to run on an XMOS xCORE-200 series device.

The example code provided with the application has been implemented and tested on the xCORE-200 explorerKIT but there is no dependancy on this board and it can be modified to run on any development board which uses an xCORE-USB series device.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, the Universal Serial Bus 2.0 Specification and related specifications, the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this application note are linked to in the references appendix.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].
- For the full API listing of the XMOS USB Device (XUD) Library please see the document XMOS USB Device (XUD) Library[2].
- For information on designing USB devices using the XUD library please see the XMOS USB Device

[1] http://www.xmos.com/published/glossary
[2] http://www.xmos.com/published/xuddg

Design Guide for reference[3].

- For information on the USB CDC class using the XMOS USB library see AN00124
- For information on using the xCORE-200 explorerKIT accelerometer see AN00181

[3] http://www.xmos.com/published/xmos-usb-device-design-guide

1 Overview

1.1 Introduction

USB Communication Class is a composite USB device class that enables telecommunication devices like digital telephones, ISDN terminal adapters, etc and networking devices like ADSL modems, Ethernet adapters/hubs, etc to connect to a USB host machine. It specifies multiple models to support different types of communication devices. Abstract Control Model (ACM) is defined to support legacy modem devices and an advantage of ACM is the Serial emulation feature. Serial emulation of a USB device eases the development of host PC application, provides software compatibility with RS-232 based legacy devices, enables USB to RS-232 conversions and gives good abstraction over the USB for application developers.

In this application note, the USB CDC implementation on xCORE-USB device is explained in detail which will help you in two ways. First, it acts as reference for you to build your own USB CDC class device, second, it gives you an idea of how to use this virtual serial port code in your application.

The standard USB CDC class specification can be found in the USB-IF website.

(http://www.usb.org/developers/docs/devclass_docs/CDC1.2_WMC1.1_012011.zip)

This application note extends AN00124 to interface the USB virtual serial port to a selection of hardware interfaces and provides a simple application for accessing them via the virtual serial port on the the USB host machine.

1.2 Block diagram

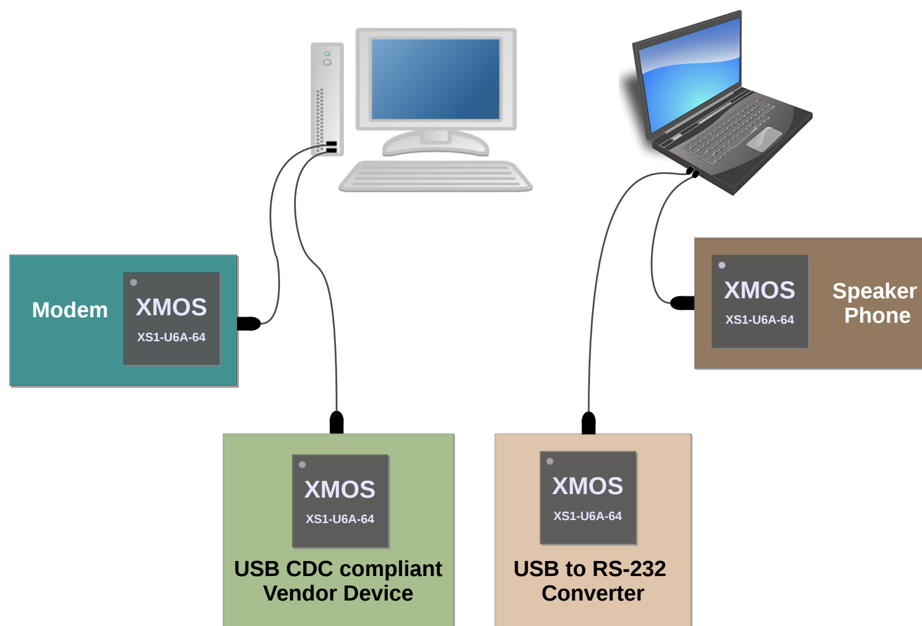

Figure 1: Block diagram of USB CDC applications

2 USB CDC Class - Application note AN00124

This application note takes the majority of its source code from AN00124 to implement the CDC class application used for the example. The details of this and the explanation of the associated source code is provided in that application note. This example provides a simple description of how that application note has been extended to support interfacing hardware to the host machine via a CDC endpoint.

The original CDC virtual com port application code has been replaced with an extended function which interfaces to a selection of hardware devices on the xCORE-200 explorerKIT. A simple menu system is provided on the virtual serial port to communicate with the hardware.

Copyright 2015 XMOS Ltd.

3 USB CDC Class - Extended on xCORE-200 Explorer application note

The example in this application note uses the XMOS USB device library and shows a simple program that enumerates a USB CDC Class device as virtual serial port in a host machine and provides a menu driven application to do I/O control from a serial terminal software like putty, teraterm etc.

For this USB CDC device application example, the system comprises four tasks running on separate logical cores of an xCORE-USB multicore microcontroller.

The tasks perform the following operations.

- A task containing the USB library functionality to communicate over USB.
- A task implementing Endpoint0 responding to both standard and CDC class-specific USB requests.
- A task implementing the data endpoints and notification endpoint of the CDC ACM class. It handles tx and rx buffers and provides interface for applications.
- A task implementing the application logic to interact with user over the virtual serial port.
- A task implementing an I2C interface

These tasks communicate via the use of xCONNECT channels which allow data to be passed between application code running on separate logical cores. In this example, XC interfaces are used, which abstracts out the channel communication details with function level interface.

The following diagram shows the task and communication structure for this USB CDC class application example.

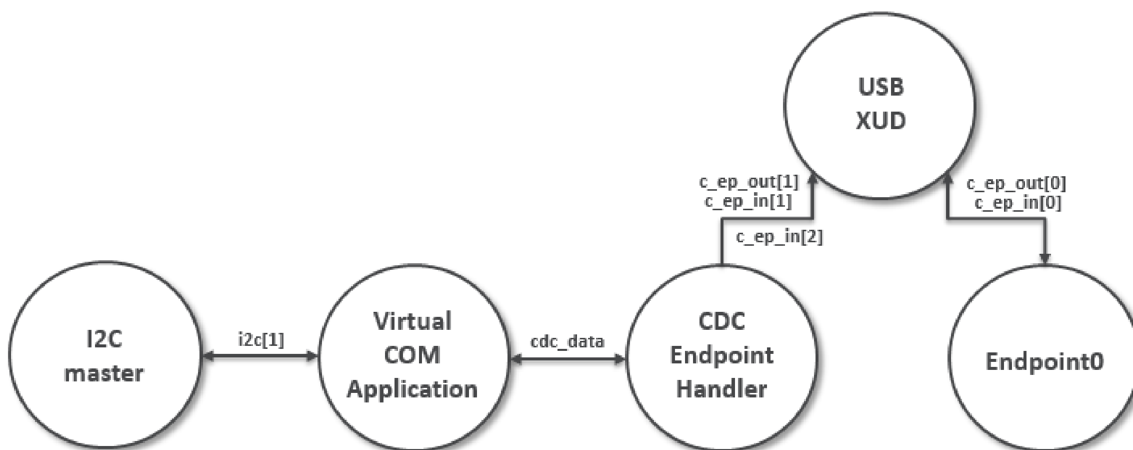

Figure 2: Task diagram of the USB CDC Virtual Serial Port example

3.1 Makefile additions for this example

There are additions to the Makefile provided with AN00124, these relate to using the I2C library.

To start using the I2C library, you need to add **lib_i2c** to your makefile:

```
USED_MODULES = ... lib_i2c ...
```

You can then access the I2C functions in your source code via the *i2c.h* header file:

```
#include <i2c.h>
```

Additionally the USB interface is moved onto xCORE tile 1 from xCORE tile 0 by using the following define:

```
-DUSB_TILE=tile[1]
```

3.2 Accelerometer I2C port declaration

In order to use the temperature sensor, port resources need to be declared to be used by the I2C library. These are defined in the code below,

```
// I2C interface ports
on tile[0]: port p_scl = XS1_PORT_1E;
on tile[0]: port p_sda = XS1_PORT_1F;
```

Futher information on the xCORE-200 explorerKIT accelerometer and interfacing with it can be found on application note AN00181.

3.3 USB CDC class extended application

In order to replace the COM port application code in AN00124 using the xCORE-200 explorerKIT a replacement function has been written. This is contained within the file app_virtual_com_extended.xc and is prototyped as follows,

```
void app_virtual_com_extended(client interface usb_cdc_interface cdc, client interface i2c_master_if i2c)
```

This function takes the CDC and I2C interfaces as parameters to allow the CDC endpoint code to read and write from the virtual com port and access the I2C device. It implements a siple menu driven application which allows the user to interact with the hardware present on the xCORE-200 explorerKIT.

The following code is taken from *app_virtual_com_extended()* function

```
/* Check if user has input any character */
if(cdc.available_bytes())
{
    value = cdc.get_char();

    /* Do the chosen operation */
    if(value == '1') {
        length = strlen(echo_mode_str[0]);
        cdc.write(echo_mode_str[0], length);
        length = strlen(echo_mode_str[1]);
        cdc.write(echo_mode_str[1], length);

        while(value != 0x1A) { /* 0x1A = Ctrl + Z */
            value = cdc.get_char();
            cdc.put_char(value);
        }
        length = strlen(echo_mode_str[2]);
        cdc.write(echo_mode_str[2], length);
    }
    else if((value >= '2') && (value <= '5')) {
```

Copyright 2015 XMOS Ltd.

```
        /* Find out which LED to toggle */
        led_id = (value - 0x30) - 2;    // 0x30 used to convert the ascii to number
        toggle_led(led_id);
    }
    else if(value == '6') {
        char status_data = 0;
        i2c_regop_res_t result;

        // Wait for valid accelerometer data
        do {
          status_data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_DR_STATUS, result);
        } while (!status_data & 0x08);

        // Read x and y axis values
        x = read_acceleration(i2c, FXOS8700EQ_OUT_X_MSB);
        y = read_acceleration(i2c, FXOS8700EQ_OUT_Y_MSB);
        z = read_acceleration(i2c, FXOS8700EQ_OUT_Z_MSB);

        length = sprintf(tmp_string, "Accelerometer: x[%d] y[%d] z[%d]\r\n", x, y, z);
        cdc.write(tmp_string, length);
    }
    else if(value == '7') {
        /* Read 32-bit timer value */
        tmr :> timer_val;
        length = sprintf(tmp_string, "Timer ticks: %u\r\n", timer_val);
        cdc.write(tmp_string, length);
    }
    else {
        show_menu(cdc);
    }
}
```

In the above code you can observe that the interface's functions are accessed via a variable 'cdc'. This variable is the client side of the *usb_cdc_interface*.

3.4 Changes to the main() function of AN00124

The code below shows the changes required to main() in application note AN00124 in order to enable the new extended CDC endpoint. This is a small change which will set up the I2C interface and call the new function provided with this application note.

```
int main() {
    /* Channels to communicate with USB endpoints */
    chan c_ep_out[XUD_EP_COUNT_OUT], c_ep_in[XUD_EP_COUNT_IN];
    /* Interface to communicate with USB CDC (Virtual Serial) */
    interface usb_cdc_interface cdc_data;
    /* I2C interface */
    i2c_master_if i2c[1];

    par
    {
        on USB_TILE: xud(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                         null, XUD_SPEED_HS, XUD_PWR_SELF);

        on USB_TILE: Endpoint0(c_ep_out[0], c_ep_in[0]);

        on USB_TILE: CdcEndpointsHandler(c_ep_in[1], c_ep_out[1], c_ep_in[2], cdc_data);

        on tile[0]: app_virtual_com_extended(cdc_data, i2c[0]);

        on tile[0]: i2c_master(i2c, 1, p_scl, p_sda, 10);
    }
    return 0;
}
```

3.5 Setting up the I2C interface in the application main()

The I2C device is accessed from main() using an interface, this is declared as follows,

```
i2c_master_if i2c[1];
```

The interface is initialised in main via the following call,

```
on tile[0]: i2c_master(i2c, 1, p_scl, p_sda, 10);
```

The interface `i2c` is passed to the new extended CDC function in main() as follows,

```
on tile[0]: app_virtual_com_extended(cdc_data, i2c[0]);
```

3.6 Adding the extended CDC endpoint to main()

In order to access the function `app_virtual_com_extended()` from main() the following header file needs to be added to main.xc of the application.

```
#include "app_virtual_com_extended.h"
```

APPENDIX A - Demo Hardware Setup

To run the demo, connect the xCORE-200 explorerKIT power to a USB socket, plug the XTAG into the board and connect the xTAG USB cable to your development machine

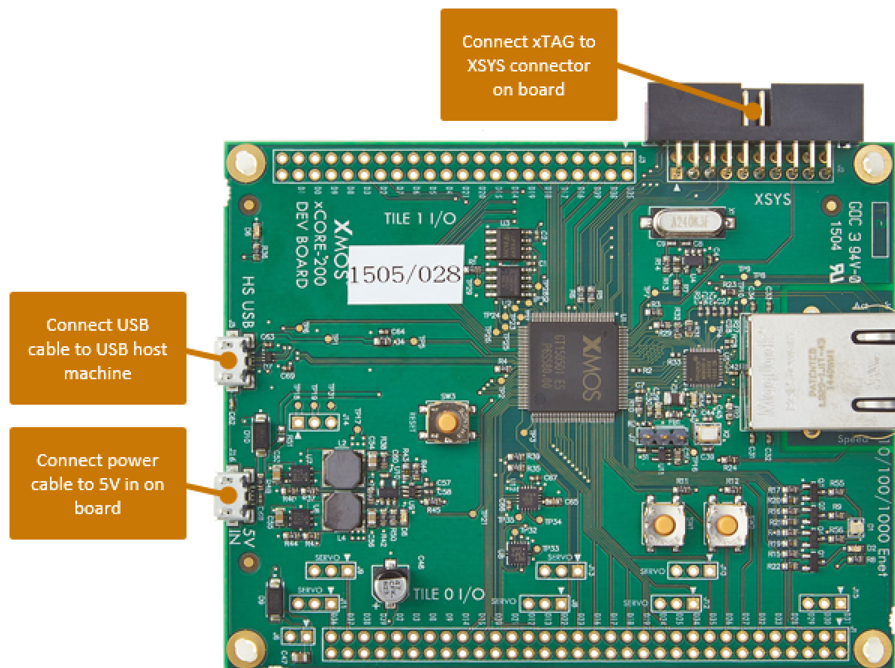

Figure 3: XMOS xCORE-200 explorerKIT

The hardware should be configured as displayed above for this demo:

- The XTAG debug adapter should be connected to the XSYS connector and the XTAG USB cable should be connected to the host machine
- The xCORE-200 explorerKIT should have the power cable connected
- The board should be connected to a USB host via the HS USB connector

Copyright 2015 XMOS Ltd.

APPENDIX B - Launching the demo application

Once the demo example has been built either from the command line using xmake or via the build mechanism of xTIMEcomposer studio we can execute the application on the xCORE-200 explorerKIT.

Once built there will be a *bin* directory within the project which contains the binary for the xCORE device. The xCORE binary has a XMOS standard .xe extension.

B.1 Launching from the command line

From the command line we use the xrun tool to download code to both the xCORE devices. If we change into the bin directory of the project we can execute the code on the xCORE microcontroller as follows:

```
> xrun app_usb_cdc_demo.xe          <-- Download and execute the xCORE code
```

Once this command has executed the CDC device will have enumerated as virtual serial port on your host machine.

B.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio we use the run mechanism to download code to xCORE device. From *Project Explorer* select the xCORE binary from the bin directory, right click and then run as xCORE application will execute the code on the xCORE device.

Once this command has executed, the CDC device will have enumerated as virtual serial port on your host machine.

B.3 Running the Virtual COM demo

To run the demo, you need to have a serial terminal software in your host machine. Once a terminal software connects to the enumerated virtual serial port, a menu will be provided for user to interact with the LEDs, buttons etc.

Following sections describe in detail on how to run the demo on different OS platforms.

B.3.1 Running on Windows

- In Microsoft Windows, when the USB CDC device enumerates for the frist time it will ask for a host driver. Use 'Install driver from specific location' option to point to the 'cdc_demo.inf' supplied along with this application note. This will load the 'usbser.sys' host driver for the virtual COM device.
- Once the driver is installed, the device will be assigned with a COM port number and it will look like the following figure in "Device Manager" (Start->Control Panel->System->Hardware->Device Manager).

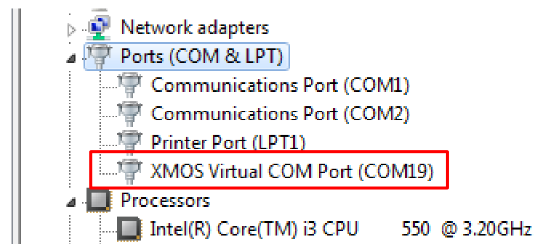

Figure 4: Enumerated Virtual COM Port Device in Windows

- Use any terminal software to open the COM port with default settings. In this demo, we have used *Hercules* as the terminal software.
- The menu and available options to select should now be available on the terminal console.

B.3.2 Running on Linux

- Under Linux, when the USB CDC device enumerates the built-in *ACM* driver will be loaded automatically and the device will be mounted as **/dev/ttyACMx** where 'x' is a number.
- You can execute *dmesg* command in a command prompt to determine the name on which the device is mounted.
- Use any serial terminal software to open the virtual serial port with default settings. In this demo, we have used *Putty* software and the serial port is opened as shown below.

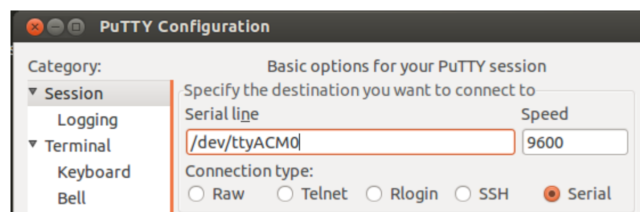

Figure 5: Opening Virtual Serial Port in Putty

- The menu and available options to select should now be available on the terminal console.

B.3.3 Running on Mac OS

- In Mac OS X, the USB CDC device is supported by a default driver available in the OS and the device will appear as /dev/tty.usbmodem*. You can use *ls /dev/tty.usbmodem** command to determine the exact name of the virtual serial device.
- Use any serial terminal software to open the virtual serial port with default settings. In this demo, we have used *CoolTerm* software and the serial port is opened as shown below.

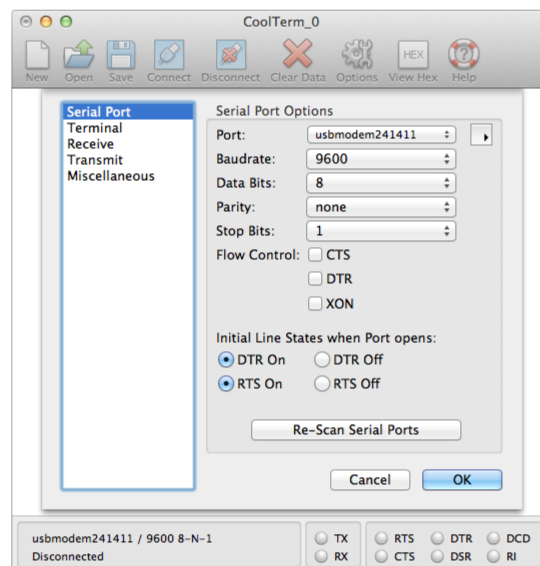

Figure 6: Opening Virtual Serial Device in CoolTerm

- The menu and available options to select should now be available on the terminal console.

APPENDIX C - References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

XMOS xCORE-USB Device Library

http://www.xmos.com/published/xuddg

XMOS USB Device Design Guide

http://www.xmos.com/published/xmos-usb-device-design-guide

USB CDC Class Specification, USB.org:

http://www.usb.org/developers/docs/devclass_docs/

USB 2.0 Specification

http://www.usb.org/developers/docs/usb20_docs/usb_20_081114.zip

APPENDIX D - Full source code listing

D.1 Source code for main.xc

```
// Copyright (c) 2015, XMOS Ltd, All rights reserved

#include <platform.h>
#include <xs1.h>
#include "usb.h"
#include "i2c.h"
#include "xud_cdc.h"
#include "app_virtual_com_extended.h"

// I2C interface ports
on tile[0]: port p_scl = XS1_PORT_1E;
on tile[0]: port p_sda = XS1_PORT_1F;

/* USB Endpoint Defines */
#define XUD_EP_COUNT_OUT   2    //Includes EP0 (1 OUT EP0 + 1 BULK OUT EP)
#define XUD_EP_COUNT_IN    3    //Includes EP0 (1 IN EP0 + 1 INTERRUPT IN EP + 1 BULK IN EP)

int main() {
    /* Channels to communicate with USB endpoints */
    chan c_ep_out[XUD_EP_COUNT_OUT], c_ep_in[XUD_EP_COUNT_IN];
    /* Interface to communicate with USB CDC (Virtual Serial) */
    interface usb_cdc_interface cdc_data;
    /* I2C interface */
    i2c_master_if i2c[1];

    par
    {
        on USB_TILE: xud(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                         null, XUD_SPEED_HS, XUD_PWR_SELF);

        on USB_TILE: Endpoint0(c_ep_out[0], c_ep_in[0]);

        on USB_TILE: CdcEndpointsHandler(c_ep_in[1], c_ep_out[1], c_ep_in[2], cdc_data);

        on tile[0]: app_virtual_com_extended(cdc_data, i2c[0]);

        on tile[0]: i2c_master(i2c, 1, p_scl, p_sda, 10);
    }
    return 0;
}
```

D.2 Source code for app_virtual_com_extended.xc

```
// Copyright (c) 2015, XMOS Ltd, All rights reserved

#include <platform.h>
#include <xs1.h>
#include <stdio.h>
#include <string.h>
#include <timer.h>
#include "i2c.h"
#include "xud_cdc.h"

/* App specific defines */
#define MENU_MAX_CHARS  30
#define MENU_LIST       11
#define DEBOUNCE_TIME   (XS1_TIMER_HZ/50)
#define BUTTON_PRESSED  0x00

// FXOS8700EQ register address defines - From AN00181
#define FXOS8700EQ_I2C_ADDR 0x1E
#define FXOS8700EQ_XYZ_DATA_CFG_REG 0x0E
#define FXOS8700EQ_CTRL_REG_1 0x2A
#define FXOS8700EQ_DR_STATUS 0x0
#define FXOS8700EQ_OUT_X_MSB 0x1
#define FXOS8700EQ_OUT_X_LSB 0x2
#define FXOS8700EQ_OUT_Y_MSB 0x3
```

```
#define FXOS8700EQ_OUT_Y_LSB 0x4
#define FXOS8700EQ_OUT_Z_MSB 0x5
#define FXOS8700EQ_OUT_Z_LSB 0x6

/* PORT_4A connected to the 4 LEDs */
on tile[0]: port p_led = XS1_PORT_4F;

/* PORT_4C connected to the 2 Buttons */
on tile[0]: port p_button = XS1_PORT_4E;

char app_menu[MENU_LIST][MENU_MAX_CHARS] = {
        {"\n\r-------------------------\r\n"},
        {"XMOS USB Virtual COM Demo\r\n"},
        {"-------------------------\r\n"},
        {"1. Switch to Echo mode\r\n"},
        {"2. Toggle LED 1\r\n"},
        {"3. Toggle LED 2\r\n"},
        {"4. Toggle LED 3\r\n"},
        {"5. Toggle LED 4\r\n"},
        {"6. Read Accelerometer\r\n"},
        {"7. Print timer ticks\r\n"},
        {"-------------------------\r\n"},
};

char echo_mode_str[3][30] = {
        {"Entered echo mode\r\n"},
        {"Press Ctrl+Z to exit it\r\n"},
        {"\r\nExit echo mode\r\n"},
};

#define ARRAY_SIZE(x) (sizeof(x)/sizeof(x[0]))

/* Sends out the App menu over CDC virtual port*/
void show_menu(client interface usb_cdc_interface cdc)
{
    unsigned length;
    for(int i = 0; i < MENU_LIST; i++) {
        length = strlen(app_menu[i]);
        cdc.write(app_menu[i], length);
    }
}

/* Function to set LED state - ON/OFF */
void set_led_state(int led_id, int val)
{
  int value;
  /* Read port value into a variable */
  p_led :> value;
  if (!val) {
      p_led <: (value | (1 << led_id));
  } else {
      p_led <: (value & ~(1 << led_id));
  }
}

/* Function to toggle LED state */
void toggle_led(int led_id)
{
    int value;
    p_led :> value;
    p_led <: (value ^ (1 << led_id));
}

/* Function to get button state (0 or 1)*/
int get_button_state(int button_id)
{
    int button_val;
    p_button :> button_val;
    button_val = (button_val >> button_id) & (0x01);
    return button_val;
}

/* Checks if a button is pressed */
int is_button_pressed(int button_id)
{
```

```
    if(get_button_state(button_id) == BUTTON_PRESSED) {
        /* Wait for debounce and check again */
        delay_ticks(DEBOUNCE_TIME);
        if(get_button_state(button_id) == BUTTON_PRESSED) {
            return 1; /* Yes button is pressed */
        }
    }
    /* No button press */
    return 0;
}

int read_acceleration(client interface i2c_master_if i2c, int reg) {
    i2c_regop_res_t result;
    int accel_val = 0;
    unsigned char data = 0;

    // Read MSB data
    data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, reg, result);
    if (result != I2C_REGOP_SUCCESS) {
      return 0;
    }

    accel_val = data << 2;

    // Read LSB data
    data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, reg+1, result);
    if (result != I2C_REGOP_SUCCESS) {
      return 0;
    }

    accel_val |= (data >> 6);

    if (accel_val & 0x200) {
      accel_val -= 1023;
    }

    return accel_val;
}

/* Initializes the Application */
void app_init(client interface i2c_master_if i2c)
{
    i2c_regop_res_t result;

    /* Set all LEDs to OFF (Active low)*/
    p_led <: 0x0F;
    /* Accelerometer setup */

    // Configure FXOS8700EQ
    result = i2c.write_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_XYZ_DATA_CFG_REG, 0x01);
    if (result != I2C_REGOP_SUCCESS) {
      return;
    }

    // Enable FXOS8700EQ
    result = i2c.write_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_CTRL_REG_1, 0x01);
    if (result != I2C_REGOP_SUCCESS) {
      return;
    }
}

/* Application task */
void app_virtual_com_extended(client interface usb_cdc_interface cdc, client interface i2c_master_if i2c)
{
    unsigned int length, led_id;
    int x = 0;
    int y = 0;
    int z = 0;
    char value, tmp_string[50];
    unsigned int button_1_valid, button_2_valid;
    timer tmr;
    unsigned int timer_val;

    app_init(i2c);
    show_menu(cdc);
```

```
    button_1_valid = button_2_valid = 1;

    while(1)
    {
        /* Check for a change in button 1 - Detects 1->0 transition */
        if(is_button_pressed(0)) {
            if(button_1_valid) {
                button_1_valid = 0;
                length = sprintf(tmp_string, "\r\nButton 1 Pressed!\r\n");
                cdc.write(tmp_string, length);
            }
        } else {
            button_1_valid = 1;
        }


        /* Check for a change in button 2 - Detects 1->0 transition */
        if(is_button_pressed(1)) {
            if(button_2_valid) {
                button_2_valid = 0;
                length = sprintf(tmp_string, "\r\nButton 2 Pressed!\r\n");
                cdc.write(tmp_string, length);
            }
        } else {
            button_2_valid = 1;
        }

        /* Check if user has input any character */
        if(cdc.available_bytes())
        {
            value = cdc.get_char();

            /* Do the chosen operation */
            if(value == '1') {
                length = strlen(echo_mode_str[0]);
                cdc.write(echo_mode_str[0], length);
                length = strlen(echo_mode_str[1]);
                cdc.write(echo_mode_str[1], length);

                while(value != 0x1A) { /* 0x1A = Ctrl + Z */
                    value = cdc.get_char();
                    cdc.put_char(value);
                }
                length = strlen(echo_mode_str[2]);
                cdc.write(echo_mode_str[2], length);
            }
            else if((value >= '2') && (value <= '5')) {
                /* Find out which LED to toggle */
                led_id = (value - 0x30) - 2;    // 0x30 used to convert the ascii to number
                toggle_led(led_id);
            }
            else if(value == '6') {
                char status_data = 0;
                i2c_regop_res_t result;

                // Wait for valid accelerometer data
                do {
                  status_data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_DR_STATUS, result);
                } while (!status_data & 0x08);

                // Read x and y axis values
                x = read_acceleration(i2c, FXOS8700EQ_OUT_X_MSB);
                y = read_acceleration(i2c, FXOS8700EQ_OUT_Y_MSB);
                z = read_acceleration(i2c, FXOS8700EQ_OUT_Z_MSB);

                length = sprintf(tmp_string, "Accelerometer: x[%d] y[%d] z[%d]\r\n", x, y, z);
                cdc.write(tmp_string, length);
            }
            else if(value == '7') {
                /* Read 32-bit timer value */
                tmr :> timer_val;
                length = sprintf(tmp_string, "Timer ticks: %u\r\n", timer_val);
                cdc.write(tmp_string, length);
            }
```

```
            else {
                show_menu(cdc);
            }
        }
    } /* end of while(1) */
}
```

D.3 Source code for app_virtual_com_extended.h

```
// Copyright (c) 2015, XMOS Ltd, All rights reserved

#ifndef APP_VIRTUAL_COM_EXTENDED_H_
#define APP_VIRTUAL_COM_EXTENDED_H_

void app_virtual_com_extended(client interface usb_cdc_interface cdc, client interface i2c_master_if i2c);

#endif /* APP_VIRTUAL_COM_EXTENDED_H_ */
```

XMOS

Copyright © 2015, All Rights Reserved.

Xmos Ltd. is the owner or licensee of this design, code, or Information (collectively, the “Information”) and is providing it to you “AS IS” with no warranty of any kind, express or implied and shall have no liability in relation to its use. Xmos Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.