Application Note: AN00126

USB Printer Device Class

This application note shows how to create a USB device compliant to the standard USB printer device class 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 printer device.

The printer framework uses XMOS libraries to provide a unidirectional printer device example over high speed USB. The code used in the application note creates a device which supports the receiving of data from the USB host in order to demonstrate how to build a USB printer interface on an XMOS device.

Text files can be printed from the USB host and the text will be sent back to the host via debug output from the xCORE device demonstrating the operation of the USB printer device in this application.

Note: This application note provides a standard USB Printer Class Device and as a result does not require drivers to run on Windows, Mac or Linux.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0.0
- XMOS USB library - Version 2.0.0
- XMOS debug printing library - Version 2.0.0

Required hardware

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

The example code provided with the application has been implemented and tested on the xCORE-USB sliceKIT (XK-SK-U16-ST) but there is no dependency 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 the document XMOS USB Device (XUD) Library[2].
- For information on designing USB devices using the XUD library please see the XMOS USB Device Design Guide for reference[3].

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

1 Overview

1.1 Introduction

The Universal Serial Bus (USB) is a communications architecture that gives a PC the ability to interconnect a variety of devices via a simple four-wire cable. One such device is the printer. Traditionally, printers have been interfaced using the following technologies:

- Unidirectional parallel port
- Bi-directional parallel port
- Serial port
- SCSI port
- Ethernet/LAN

There are other, more sophisticated printer interfaces, but the ones previously listed are the most popular. USB offers a much greater throughput capability than the serial port and is comparable in speed to the parallel port. This makes both parallel and serial printers good candidates for interfacing with USB.

The USB specification provides as standard device class for the implementation of USB printers.

(http://www.usb.org/developers/devclass_docs/usbprint11.pdf)

1.2 Block diagram

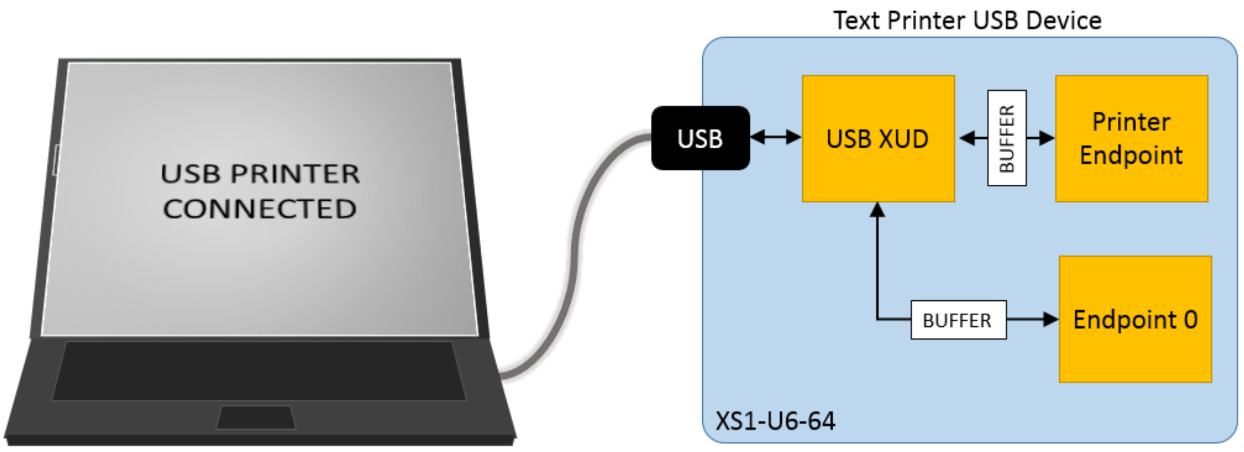


Figure 1: Block diagram of USB printer application example

2 USB Printer Device Class application note

The demo in this note uses the XMOS USB device library and shows a simple program that receives data in PCL format from the host and outputs this data to the console in the form of a log.

For the USB printer device class application example, the system comprises three tasks running on separate logical cores of a 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 both standard and printer class USB requests
- A task implementing the application code for receiving printer data into the device

These tasks communicate via the use of xCONNECT channels which allow data to be passed between application code running on separate logical cores.

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

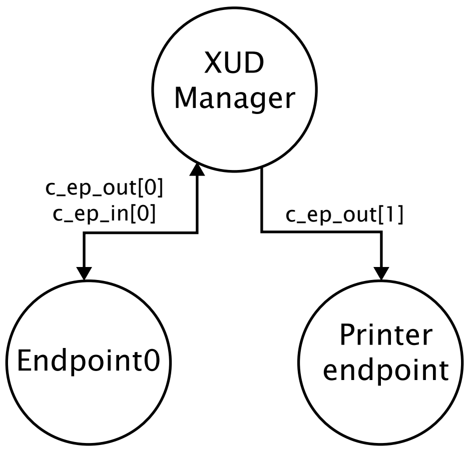


Figure 2: Task diagram of USB printer application example

2.1 Makefile additions for this application

To start using the USB library, you need to add `lib_usb` to your makefile:

```
USED_MODULES = ... lib_usb lib_logging
```

You can then access the USB functions in your source code via the usb.h header file:

```
#include <usb.h>
```

2.2 Declaring resource and setting up the USB components

`main.xc` contains the application implementation for a device based on the USB printer device class. There are some defines in it that are used to configure the XMOS USB device library. These are displayed below.

```
/* USB Endpoint Defines */
#define XUD_EP_COUNT_OUT   2    //Includes EP0 (1 out EP0 + Printer data output EP)
#define XUD_EP_COUNT_IN    1    //Includes EP0 (1 in EP0)
```

These defines describe the endpoint configuration for this device. This example has bi-directional communication with the host machine via the standard endpoint0 and an endpoint for receiving the printer data from the host into our device.

These defines are passed to the setup function for the USB library which is called from main().

2.3 The application main() function

Below is the source code for the main function of this application, which is taken from the source file main.xc

```
int main()
{
    chan c_ep_out[XUD_EP_COUNT_OUT], c_ep_in[XUD_EP_COUNT_IN];

    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: printer_main(c_ep_out[1]);

    }
    return 0;
}
```

Looking at this in a more detail you can see the following:

- The par statement starts three separate tasks in parallel
- There is a task to configure and execute the USB library: xud()
- There is a task to startup and run the Endpoint0 code: Endpoint0()
- There is a task to deal with USB printer data requests arriving from the host printer_main()
- The define USB_TILE describes the tile on which the individual tasks will run
- In this example all tasks run on the same tile as the USB PHY this isa requirement of xud()
- The xCONNECT communication channels used by the application are set up at the beginning of main()
- The USB defines discussed earlier are passed into the function xud()

2.4 Configuring the USB Device ID

The USB ID values used for vendor id, product id and device version number are defined in the file endpoint0.xc. These are used by the host machine to determine the vendor of the device (in this case XMOS) and the product plus the firmware version.

```
/* USB Device ID Defines */
#define BCD_DEVICE   0x1000
#define VENDOR_ID    0x20B1
#define PRODUCT_ID   0x00ed
```

2.5 USB Printer Class specific defines

The USB Printer Class is configured in the file endpoint0.xc. Below there are a set of standard defines which are used to configure the USB device descriptors to setup a USB printer device running on an xCORE-USB microcontroller.

```
/* USB Printer Subclass Definition */
#define USB_PRINTER_SUBCLASS        0x01

/* USB Printer interface types */
#define USB_PRINTER_UNIDIRECTIONAL  0x01
#define USB_PRINTER_BIDIRECTIONAL   0x02
#define USB_PRINTER_1284_4_COMPAT   0x03

/* USB Printer Request types */
#define PRINTER_GET_DEVICE_ID       0x00
#define PRINTER_GET_PORT_STATUS     0x01
#define PRINTER_SOFT_RESET          0x02
```

2.6 USB Device Descriptor

endpoint0.xc is where the standard USB device descriptor is declared for the printer device. Below is the structure which contains this descriptor. This will be requested by the host when the device is enumerated on the USB bus.

```
static unsigned char devDesc[] =
{
    0x12,                  /* 0  bLength */
    USB_DESCTYPE_DEVICE,   /* 1  bdescriptorType */
    0x00,                  /* 2  bcdUSB version */
    0x02,                  /* 3  bcdUSB version */
    0x00,                  /* 4  bDeviceClass - Specified by interface */
    0x00,                  /* 5  bDeviceSubClass  - Specified by interface */
    0x00,                  /* 6  bDeviceProtocol  - Specified by interface */
    0x40,                  /* 7  bMaxPacketSize for EP0 - max = 64*/
    (VENDOR_ID & 0xFF),    /* 8  idVendor */
    (VENDOR_ID >> 8),      /* 9  idVendor */
    (PRODUCT_ID & 0xFF),   /* 10 idProduct */
    (PRODUCT_ID >> 8),     /* 11 idProduct */
    (BCD_DEVICE & 0xFF),   /* 12 bcdDevice */
    (BCD_DEVICE >> 8),     /* 13 bcdDevice */
    0x01,                  /* 14 iManufacturer - index of string*/
    0x02,                  /* 15 iProduct  - index of string*/
    0x00,                  /* 16 iSerialNumber  - index of string*/
    0x01                   /* 17 bNumConfigurations */
};
```

From this descriptor you can see that product, vendor and device firmware revision are all coded into this structure. This will allow the host machine to recognise the printer device when it is connected to the USB bus.

2.7 USB Configuration Descriptor

The USB configuration descriptor is used to configure the device in terms of the device class and the endpoint setup. For the USB printer device the configuration descriptor which is read by the host is as follows.

```
static unsigned char cfgDesc[] = {
    0x09,                 /* 0  bLength */
    USB_DESCTYPE_CONFIGURATION, /* 1  bDescriptortype = configuration*/
    0x19, 0x00,           /* 2  wTotalLength of all descriptors */
    0x01,                 /* 4  bNumInterfaces */
    0x01,                 /* 5  bConfigurationValue */
    0x03,                 /* 6  iConfiguration - index of string*/
    0x40,                 /* 7  bmAttributes - Self powered*/
    0xFA,                 /* 8  bMaxPower - 500mA */

    0x09,                 /* 0  bLength */
    USB_DESCTYPE_INTERFACE,/* 1  bDescriptorType */
    0x00,                 /* 2  bInterfacecNumber */
    0x00,                 /* 3  bAlternateSetting */
    0x01,                 /* 4: bNumEndpoints */
    USB_CLASS_PRINTER,    /* 5: bInterfaceClass */
    USB_PRINTER_SUBCLASS, /* 6: bInterfaceSubClass */
    USB_PRINTER_UNIDIRECTIONAL, /* 7: bInterfaceProtocol*/
    0x00,                 /* 8  iInterface */

    0x07,                 /* 0  bLength */
    USB_DESCTYPE_ENDPOINT,/* 1  bDescriptorType */
    0x01,                 /* 2  bEndpointAddress - EP1, OUT*/
    XUD_EPTYPE_BUL,       /* 3  bmAttributes */
    0x00,                 /* 4  wMaxPacketSize - Low */
    0x02,                 /* 5  wMaxPacketSize - High */
    0x01,                 /* 6  bInterval */
};
```

From this you can see that the USB printer class defines described earlier are encoded into the configuration descriptor along with the bulk USB endpoint description for receiving printer data into the application code. This endpoint allows us to process the printer data request from the host inside the main printer application task.

2.8 USB string descriptors

There are two further descriptors within this file relating to the configuration of the USB Printer Class. These sections should also be modified to match the capabilities of the printer.

```
/* String table - unsafe as accessed via shared memory */
static char * unsafe stringDescriptors[]=
{
  "\x09\x04",             // Language ID string (US English)
  "XMOS",                 // iManufacturer
  "Printomatic 2000",     // iProduct
  "Test config",          // iConfiguration string
};
```

```
/* Class specific string IEEE1288 string descriptor */
static unsigned char deviceIDstring[] = "  MFG:Generic;MDL:Generic_/_Text_Only;CMD:1284.4;CLS:PRINTER;DES:
   ↪ Generic text only printer;";
```

2.9 USB Printer Class requests

Inside endpoint0.xc there is some code for handling the USB printer device class specific requests. These are shown in the following code:

```
/* Printer Class Requests */
XUD_Result_t PrinterInterfaceClassRequests(XUD_ep c_ep0_out, XUD_ep c_ep0_in, USB_SetupPacket_t sp)
{
    unsigned char PRT_STATUS[] = {0b00011000}; /* Paper not empty, selected, no error */

    deviceIDstring[0] = 0;
    deviceIDstring[1] = sizeof(deviceIDstring-1);

    switch(sp.bRequest)
    {
        case PRINTER_GET_DEVICE_ID:
            debug_printf("Class request - get device id\n");
            debug_printf(&deviceIDstring[2]); //Skip first two characters
            debug_printf("\n");

            return XUD_DoGetRequest(c_ep0_out, c_ep0_in, (deviceIDstring, unsigned char []),
                    sizeof(deviceIDstring-1), sp.wLength);
            break;

        case PRINTER_GET_PORT_STATUS:
            debug_printf("Class request - get port status id\n");
            return XUD_DoGetRequest(c_ep0_out, c_ep0_in, PRT_STATUS, 1, sp.wLength);
            break;

        case PRINTER_SOFT_RESET:
            debug_printf("Class request - soft reset id\n");
            /* Do nothing - i.e. STALL */
            /* TODO flush buffers and reset Bulk Out endpoint to default state*/
            break;
    }

    return XUD_RES_ERR;
}
```

These printer specific request are implemented by the application as they do not form part of the standard requests which have to be accepted by all device classes via endpoint0.

2.10 USB Printer Class Endpoint0

The function Endpoint0() contains the code for dealing with device requests made from the host to the standard endpoint0 which is present in all USB devices. In addition to requests required for all devices, the code handles the requests specific to the printer device class.

```
if(result == XUD_RES_OKAY)
{
    /* Set result to ERR, we expect it to get set to OKAY if a request is handled */
    result = XUD_RES_ERR;

    /* Stick bmRequest type back together for an easier parse... */
    bmRequestType = (sp.bmRequestType.Direction<<7) |
                    (sp.bmRequestType.Type<<5) |
                    (sp.bmRequestType.Recipient);

    if(USE_XSCOPE)
    {
        if ((bmRequestType == USB_BMREQ_H2D_STANDARD_DEV) &&
            (sp.bRequest == USB_SET_ADDRESS))
        {
            debug_printf("Address allocated %d\n", sp.wValue);
        }
    }

    /* Handle specific requests first */
    switch(bmRequestType)
    {

        /* Direction: Device-to-host and Host-to-device
         * Type: Class
         * Recipient: Interface
         */
        case USB_BMREQ_H2D_CLASS_INT:
        case USB_BMREQ_D2H_CLASS_INT:

            /* Inspect for printer interface num */
            if(sp.wIndex == 0)
            {
                /* Returns  XUD_RES_OKAY if handled,
                 *          XUD_RES_ERR if not handled,
                 *          XUD_RES_RST for bus reset */
                result = PrinterInterfaceClassRequests(ep0_out, ep0_in, sp);
            }
            break;
    }
}
```

2.11 Receiving printer data from the host

The application endpoint for receiving printer data from the host machine is implemented in the file `main.xc`. This is contained within the function `printer_main()` which is shown below:

```
void printer_main(chanend c_ep_prt_out)
{
    unsigned size;
    unsigned char print_packet[1024]; // Buffer for storing printer packets sent from host

    debug_printf("USB printer class demo started\n");

    /* Initialise the XUD endpoints */
    XUD_ep ep_out = XUD_InitEp(c_ep_prt_out, XUD_EPTYPE_BUL);

    while (1)
    {
        // Perform a blocking read waiting for data to be received at the endpoint
        XUD_GetBuffer(ep_out, print_packet, size);
        debug_printf("**** Received %d byte print buffer ****\n", size);
        print_string(print_packet, size);
    }
}
```

From this you can see the following.

- A buffer is declared to receive the print data which is streamed into the application from the host
- This task operates inside a `while (1)` loop which waits for data to arrive and then processes it
- A blocking call is made to the XMOS USB device library to receive data into the application
- The data received is output to the debug console; in a real printer this data would be processed here or passed onto another task for processing

APPENDIX A - Demo Hardware Setup

To run the demo, connect the xCORE-USB sliceKIT USB-B and xTAG-2 USB-A connectors to separate USB connectors on your development PC.

On the xCORE-USB sliceKIT ensure that the xCONNECT LINK switch is set to ON, as per the image, to allow xSCOPE to function. The use of xSCOPE is required in this application so that the print messages that are generated on the device as part of the demo do not interfere with the real-time behavior of the USB device.

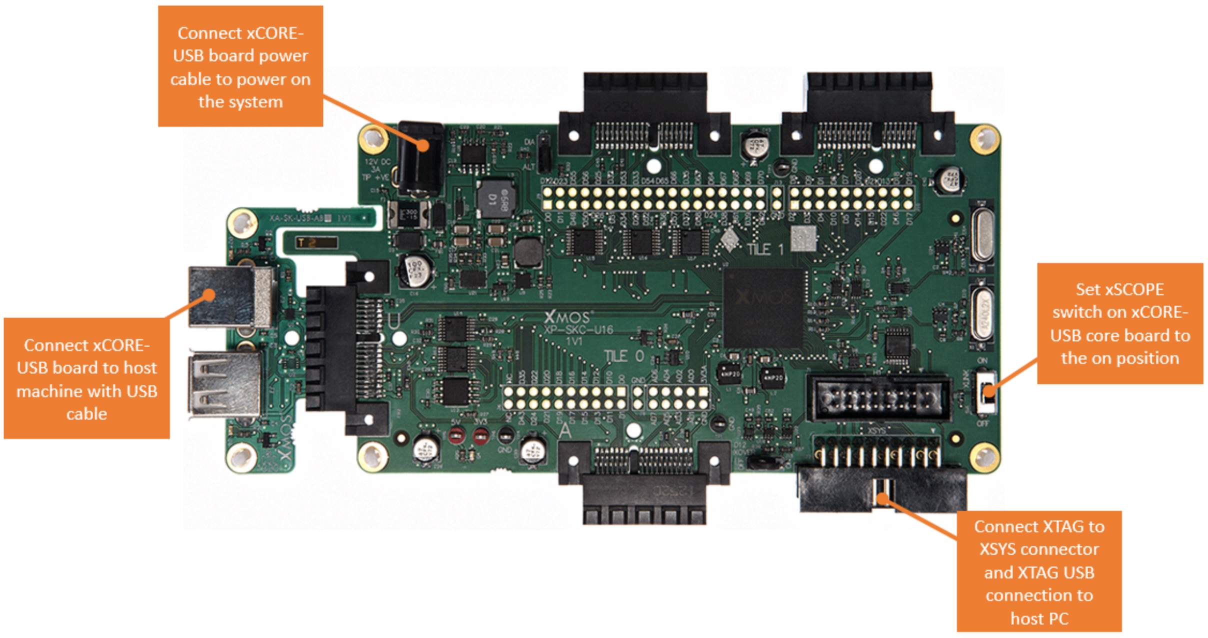

Connect xCORE-USB board power cable to power on the system
Connect xCORE-USB board to host machine with USB cable
Set xSCOPE switch on xCORE-USB core board to the on position
Connect XTAG to XSYS connector and XTAG USB connection to host PC

Figure 3: XMOS xCORE-USB sliceKIT

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-USB core board should have a USB cable connecting the device to the host machine
- The xSCOPE switch on the board should be set to the on position
- The xCORE-USB core board should have the power cable connected

APPENDIX B - Launching the demo device

Once the demo example has been built either from the command line using xmake or via the build mechanism of xTIMEcomposer studio the application can be executed on the xCORE-USB sliceKIT.

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 the xrun tool is used to download code to both the xCORE devices. Changing into the bin directory of the project we can execute the code on the xCORE microcontroller as follows:

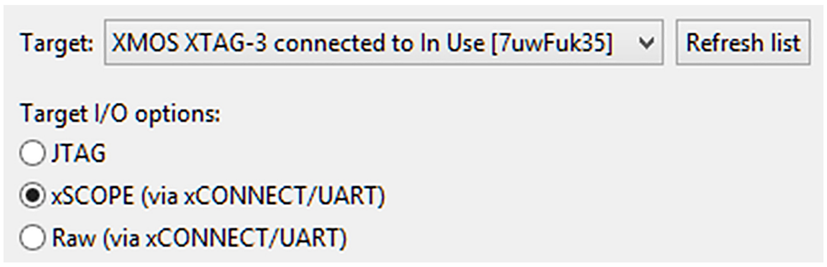
```
> xrun --xscope app_printer_demo.xe          <-- Download and execute the xCORE code
```

Once this command has executed the printer USB device should have enumerated on your machine

B.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio the run mechanism is used to download code to xCORE device. Select the xCORE binary from the bin directory, right click and then follow the instructions below.

- Select **Run As**.
- Select **Run Configurations**.
- Double click on xCORE application**.
- Enable xSCOPE in Target I/O options:


Target: XMOS XTAG-3 connected to In Use [7uwFuk35]

Target I/O options:
JTAG [✓ xSCOPE (via xCONNECT/UART)] Raw (via xCONNECT/UART)

Figure 4: xTIMEcomposer xSCOPE configuration

- Click **Apply** and then **Run**.

Once this command has executed the printer USB device should have enumerated on your machine

APPENDIX C - Running the printer demo

C.1 Windows Host

- To add the printer to Windows Control Panel :
- Open the Control Panel, select **Devices and Printers** and then **Add Printer.**
- Click *The printer that I want isn't listed*.
- Select *Add a local printer or network printer with manual settings*, then click **Next.**
- Select *Use an existing port* and *USB001 (Virtual printer port for USB)*, then click **Next.**
- Select *Generic* and *Generic / Text Only*, then click **Next.**
- Select *Use the driver that is currently installed (recommended)*, then click **Next.**
- Change the name to *XMOS Generic / Text Only*, then click **Next.**
- Select *Do not share this printer*, then click **Next.**
- Click **Finish.**

The printer will now appear in your Control Panel:

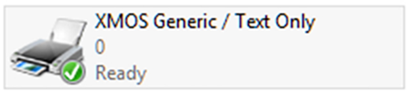

XMOS Generic / Text Only
0
Ready

Figure 5: XMOS printer in control panel

C.2 OSX Host

When connecting the printer to an Apple Mac computer it will be automatically detected as a Printomatic 2000 and you will be able to print directly to the device.

C.3 Testing the printer

To test the printer, create a text file containing the following:

```
Hello World !
```

Print the file to the printer and you will see the following output in the xTIMEcomposer console window (note some of the print decode has been deleted to aid clarity):

```
**** Received 512 byte print buffer ****
Hello World !
```

Note that on OSX the default printer output will be in PCL format, this mean that there are extra characters printed to the console other than just the hello world text.

APPENDIX D - 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 Printer Class Specification, USB.org:

http://www.usb.org/developers/devclass_docs/usbprint11.pdf

USB 2.0 Specification

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

APPENDIX E - Full source code listing

E.1 Source code for endpoint0.xc

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

/*
 * @brief Implements endpoint zero for an example Printer class device.
 */

#include <xs1.h>
#include "usb.h"
#include "debug_print.h"

/* USB Device ID Defines */
#define BCD_DEVICE   0x1000
#define VENDOR_ID    0x20B1
#define PRODUCT_ID   0x00ed

/* USB Printer Subclass Definition */
#define USB_PRINTER_SUBCLASS        0x01

/* USB Printer interface types */
#define USB_PRINTER_UNIDIRECTIONAL  0x01
#define USB_PRINTER_BIDIRECTIONAL   0x02
#define USB_PRINTER_1284_4_COMPAT   0x03

/* USB Printer Request types */
#define PRINTER_GET_DEVICE_ID       0x00
#define PRINTER_GET_PORT_STATUS     0x01
#define PRINTER_SOFT_RESET          0x02

/* USB Device Descriptor */
static unsigned char devDesc[] =
{
    0x12,                  /* 0  bLength */
    USB_DESCTYPE_DEVICE,   /* 1  bdescriptorType */
    0x00,                  /* 2  bcdUSB version */
    0x02,                  /* 3  bcdUSB version */
    0x00,                  /* 4  bDeviceClass - Specified by interface */
    0x00,                  /* 5  bDeviceSubClass  - Specified by interface */
    0x00,                  /* 6  bDeviceProtocol  - Specified by interface */
    0x40,                  /* 7  bMaxPacketSize for EP0 - max = 64*/
    (VENDOR_ID & 0xFF),    /* 8  idVendor */
    (VENDOR_ID >> 8),      /* 9  idVendor */
    (PRODUCT_ID & 0xFF),   /* 10 idProduct */
    (PRODUCT_ID >> 8),     /* 11 idProduct */
    (BCD_DEVICE & 0xFF),   /* 12 bcdDevice */
    (BCD_DEVICE >> 8),     /* 13 bcdDevice */
    0x01,                  /* 14 iManufacturer - index of string*/
    0x02,                  /* 15 iProduct  - index of string*/
    0x00,                  /* 16 iSerialNumber  - index of string*/
    0x01                   /* 17 bNumConfigurations */
};

/* USB Configuration Descriptor */
static unsigned char cfgDesc[] = {
    0x09,                 /* 0  bLength */
    USB_DESCTYPE_CONFIGURATION, /* 1  bDescriptortype = configuration*/
    0x19, 0x00,           /* 2  wTotalLength of all descriptors */
    0x01,                 /* 4  bNumInterfaces */
    0x01,                 /* 5  bConfigurationValue */
    0x03,                 /* 6  iConfiguration - index of string*/
    0x40,                 /* 7  bmAttributes - Self powered*/
    0xFA,                 /* 8  bMaxPower - 500mA */

    0x09,                 /* 0  bLength */
    USB_DESCTYPE_INTERFACE,/* 1  bDescriptorType */
    0x00,                 /* 2  bInterfacecNumber */
    0x00,                 /* 3  bAlternateSetting */
    0x01,                 /* 4: bNumEndpoints */
    USB_CLASS_PRINTER,    /* 5: bInterfaceClass */
    USB_PRINTER_SUBCLASS, /* 6: bInterfaceSubClass */
    USB_PRINTER_UNIDIRECTIONAL, /* 7: bInterfaceProtocol*/
    0x00,                 /* 8  iInterface */

    0x07,                 /* 0  bLength */
    USB_DESCTYPE_ENDPOINT,/* 1  bDescriptorType */
    0x01,                 /* 2  bEndpointAddress - EP1, OUT*/
    XUD_EPTYPE_BUL,       /* 3  bmAttributes */
    0x00,                 /* 4  wMaxPacketSize - Low */
    0x02,                 /* 5  wMaxPacketSize - High */
    0x01,                 /* 6  bInterval */
};

unsafe{
  /* String table - unsafe as accessed via shared memory */
  static char * unsafe stringDescriptors[]=
  {
    "\x09\x04",             // Language ID string (US English)
    "XMOS",                 // iManufacturer
    "Printomatic 2000",     // iProduct
    "Test config",          // iConfiguration string
  };
}

/* Class specific string IEEE1288 string descriptor */
static unsigned char deviceIDstring[] = "  MFG:Generic;MDL:Generic_/_Text_Only;CMD:1284.4;CLS:PRINTER;DES:
   ↪ Generic text only printer;";

/*
       "MFG:Generic;"                   -   manufacturer (case sensitive)
       "MDL:Generic_/_Text_Only;"       -   model (case sensitive)
       "CMD:1284.4;"                    -   PDL command set
       "CLS:PRINTER;"                   -   class
       "DES:Generic text only printer;" -   description
*/


/* Printer Class Requests */
XUD_Result_t PrinterInterfaceClassRequests(XUD_ep c_ep0_out, XUD_ep c_ep0_in, USB_SetupPacket_t sp)
{
    unsigned char PRT_STATUS[] = {0b00011000}; /* Paper not empty, selected, no error */

    deviceIDstring[0] = 0;
    deviceIDstring[1] = sizeof(deviceIDstring-1);

    switch(sp.bRequest)
    {
        case PRINTER_GET_DEVICE_ID:
            debug_printf("Class request - get device id\n");
            debug_printf(&deviceIDstring[2]); //Skip first two characters
            debug_printf("\n");

            return XUD_DoGetRequest(c_ep0_out, c_ep0_in, (deviceIDstring, unsigned char []),
                    sizeof(deviceIDstring-1), sp.wLength);
            break;

        case PRINTER_GET_PORT_STATUS:
            debug_printf("Class request - get port status id\n");
            return XUD_DoGetRequest(c_ep0_out, c_ep0_in, PRT_STATUS, 1, sp.wLength);
            break;

        case PRINTER_SOFT_RESET:
            debug_printf("Class request - soft reset id\n");
            /* Do nothing - i.e. STALL */
            /* TODO flush buffers and reset Bulk Out endpoint to default state*/
            break;
    }

    return XUD_RES_ERR;
}

/* Endpoint 0 Task */
void Endpoint0(chanend chan_ep0_out, chanend chan_ep0_in)
{
    USB_SetupPacket_t sp;
    unsigned bmRequestType;
    XUD_BusSpeed_t usbBusSpeed;

    XUD_ep ep0_out = XUD_InitEp(chan_ep0_out, XUD_EPTYPE_CTL | XUD_STATUS_ENABLE);
    XUD_ep ep0_in  = XUD_InitEp(chan_ep0_in, XUD_EPTYPE_CTL | XUD_STATUS_ENABLE);

    while(1)
    {
        /* Returns XUD_RES_OKAY on success */
        XUD_Result_t result = USB_GetSetupPacket(ep0_out, ep0_in, sp);

        if(result == XUD_RES_OKAY)
        {
            /* Set result to ERR, we expect it to get set to OKAY if a request is handled */
            result = XUD_RES_ERR;

            /* Stick bmRequest type back together for an easier parse... */
            bmRequestType = (sp.bmRequestType.Direction<<7) |
                            (sp.bmRequestType.Type<<5) |
                            (sp.bmRequestType.Recipient);

            if(USE_XSCOPE)
            {
                if ((bmRequestType == USB_BMREQ_H2D_STANDARD_DEV) &&
                    (sp.bRequest == USB_SET_ADDRESS))
                {
                    debug_printf("Address allocated %d\n", sp.wValue);
                }
            }

            /* Handle specific requests first */
            switch(bmRequestType)
            {

                /* Direction: Device-to-host and Host-to-device
                 * Type: Class
                 * Recipient: Interface
                 */
                case USB_BMREQ_H2D_CLASS_INT:
                case USB_BMREQ_D2H_CLASS_INT:

                    /* Inspect for printer interface num */
                    if(sp.wIndex == 0)
                    {
                        /* Returns  XUD_RES_OKAY if handled,
                         *          XUD_RES_ERR if not handled,
                         *          XUD_RES_RST for bus reset */
                        result = PrinterInterfaceClassRequests(ep0_out, ep0_in, sp);
                    }
                    break;
            }
        }

        /* If we haven't handled the request above then do standard enumeration requests */
        if(result == XUD_RES_ERR )
        {
            /* Returns  XUD_RES_OKAY if handled okay,
             *          XUD_RES_ERR if request was not handled (STALLed),
             *          XUD_RES_RST for USB Reset */
             unsafe{
             result = USB_StandardRequests(ep0_out, ep0_in, devDesc,
                        sizeof(devDesc), cfgDesc, sizeof(cfgDesc),
                        null, 0, null, 0, stringDescriptors, sizeof(stringDescriptors)/sizeof(
                          ↪ stringDescriptors[0]),
                        sp, usbBusSpeed);
             }
        }

        /* USB bus reset detected, reset EP and get new bus speed */
        if(result == XUD_RES_RST)
        {
            usbBusSpeed = XUD_ResetEndpoint(ep0_out, ep0_in);
        }
    }
}
```

E.2 Source code for main.xc

```
// Copyright (c) 2015, XMOS Ltd, All rights reserved
#include "usb.h"
#include "debug_print.h"
#include <xscope.h>

/* USB Endpoint Defines */
#define XUD_EP_COUNT_OUT   2    //Includes EP0 (1 out EP0 + Printer data output EP)
#define XUD_EP_COUNT_IN    1    //Includes EP0 (1 in EP0)

/* xSCOPE Setup Function */
#if (USE_XSCOPE == 1)
void xscope_user_init(void) {
    xscope_register(0, 0, "", 0, "");
    xscope_config_io(XSCOPE_IO_BASIC); /* Enable fast printing over links */
}
#endif

/* Prototype for Endpoint0 function in endpoint0.xc */
void Endpoint0(chanend c_ep0_out, chanend c_ep0_in);

/* Global report buffer, global since used by Endpoint0 core */
unsigned char g_reportBuffer[] = {0, 0, 0, 0};

/* Version of print string that doesn't terminate on null */
void print_string(unsigned char *string, unsigned size)
{
    for (int i=0; i<size; i++)
    {
        switch(*string){
            /* ignore nulls */
            case 0x00:
            break;

#ifdef IGNORE_WHITESPACE
            case 0x20:  //space
            case 0x0a:  //tab
            break;
#endif

            default:
            printchar(*string);
            break;
        }
        string++;
    }
    printchar('\n');
}

/* This function receives the printer endpoint transfers from the host */
void printer_main(chanend c_ep_prt_out)
{
    unsigned size;
    unsigned char print_packet[1024]; // Buffer for storing printer packets sent from host

    debug_printf("USB printer class demo started\n");

    /* Initialise the XUD endpoints */
    XUD_ep ep_out = XUD_InitEp(c_ep_prt_out, XUD_EPTYPE_BUL);

    while (1)
    {
        // Perform a blocking read waiting for data to be received at the endpoint
        XUD_GetBuffer(ep_out, print_packet, size);
        debug_printf("**** Received %d byte print buffer ****\n", size);
        print_string(print_packet, size);
    }
}

/* The main function runs three cores: the XUD manager, Endpoint 0, and a Printer endpoint. An array of
   channels is used for both IN and OUT endpoints */
int main()
{
    chan c_ep_out[XUD_EP_COUNT_OUT], c_ep_in[XUD_EP_COUNT_IN];
    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: printer_main(c_ep_out[1]);

    }
    return 0;
}
```