Application Note: AN00129

USB HID Class

This application note shows how to create a USB device compliant to the standard USB Human Interface Device (HID) class on an XMOS multicore microcontroller.

The code associated with this application note provides an example of using the XMOS USB Device Library and associated USB class descriptors to provide a framework for the creation of a USB HID.

The HID uses XMOS libraries to provide a simple mouse example running over high speed USB. The code used in the application note creates a device which supports the standard requests associated with this class of USB devices.

The application operates as a simple mouse which when running moves the mouse pointer on the host machine. This demonstrates the simple way in which PC peripheral devices can easily be deployed using an xCORE device.

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

Required tools and libraries

- xTIMEcomposer Tools - Version 13.0 or later
- XMOS USB library - Version 1.3.2rc0 or later

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 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 thedocument 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 HID class consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of HID class include:

- Keyboards and pointing devices, for example, standard mouse devices, trackballs, and joysticks.
- Front-panel controls, for example: knobs, switches, buttons, and sliders.
- Controls that might be found on devices such as telephones, VCR remote controls, games or simulation devices, for example: data gloves, throttles, steering wheels, and rudder pedals.
- Devices that may not require human interaction but provide data in a similar format to HID class devices, for example, bar-code readers, thermometers, or voltmeters.

Many typical HID class devices include indicators, specialized displays, audio feedback, and force or tactile feedback. Therefore, the HID class definition includes support for various types of output directed to the end user.

The USB specification provides a standard device class for the implementation of HID class devices.

(http://www.usb.org/developers/devclass_docs/HID1_11.pdf)

1.2 Block diagram

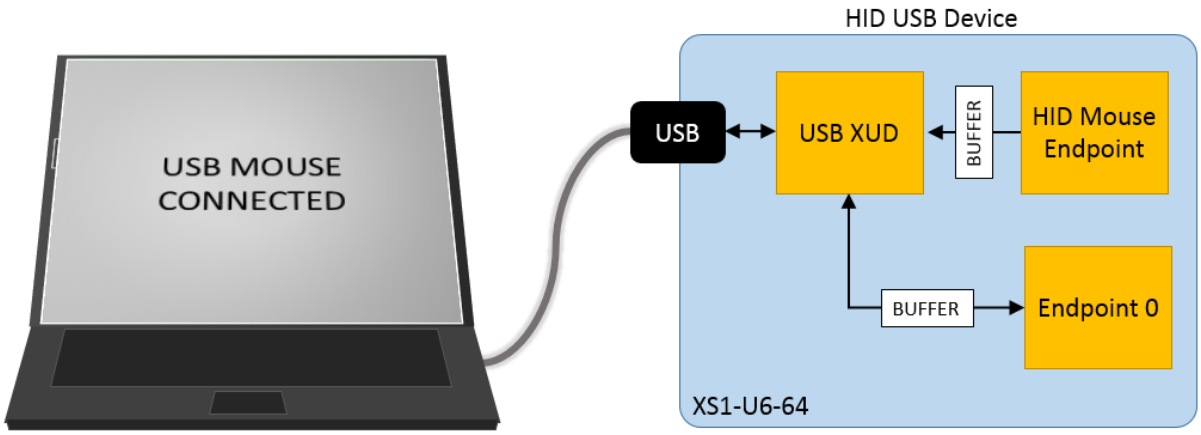

Figure 1: Block diagram of USB HID application example

2 USB HID Class application note

The example in this application note uses the XMOS USB device library and shows a simple program that creates a basic mouse device which controls the mouse pointer on the host PC.

For the USB HID 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 HID class USB requests
- A task implementing the application code for our custom HID interface

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.

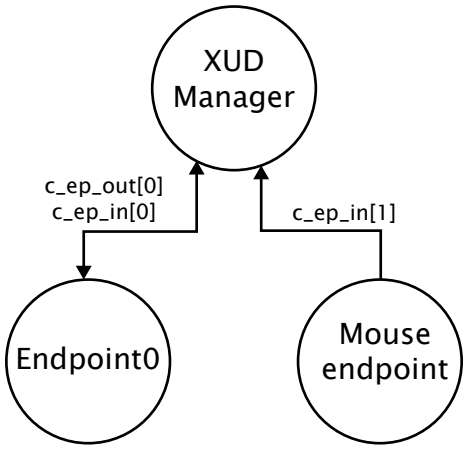

Figure 2: Task diagram of USB HID application example

2.1 Makefile additions for this example

To start using the USB library, you need to add module_xud, module_usb_device, module_usb_shared and module_usb_tile_support to your makefile:

```
USED_MODULES = ... module_xud module_usb_device module_usb_shared
                   module_usb_tile_support ...
```

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

```
#include <xud.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.

```
#define p_usb_rst null
#define clk_usb_rst null
#define PWR_MODE XUD_PWR_SELF
#define XUD_EP_COUNT_OUT   1
#define XUD_EP_COUNT_IN    2
```

The first set of defines set up the reset port and clock for the device to be null, as is required when using an XMOS xCORE-USB device. The define for power mode allows a device to set if it is powered from an external power supply or via the USB power.

In this case the example uses an external power supply.

The second set of 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 implementing the custom part of our HID class 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_Manager(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                                 null, epTypeTableOut, epTypeTableIn,
                                 p_usb_rst, clk_usb_rst, -1, XUD_SPEED_HS, PWR_MODE);

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

        on USB_TILE: hid_mouse(c_ep_in[1]);

    }

    return 0;
}
```

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

- The par functionality describes running three separate tasks in parallel
- There is a function call to configure and execute the USB library: XUD_Manager()
- There is a function call to startup and run the Endpoint0 code: Endpoint0()
- There is a function to deal with ID request generation and sending to the host hid_mouse()
- 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 although this is only a requirement of XUD_Manager()
- 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_Manager()

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 HID Device Product Defines */
#define BCD_DEVICE    0x1000
#define VENDOR_ID     0x20B1
#define PRODUCT_ID    0x1010
```

2.5 USB HID Class specific defines

The USB HID 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 HID class device running on an xCORE-USB microcontroller.

```
/* Standard HID Request Defines */

/* 7. Requests */

/* 7.1 Standard Requests  - Class Descriptor Types - High byte of wValue
 * The following defines valid types of Class descriptors */

#define HID_HID                   0x2100
#define HID_REPORT                0x2200
#define HID_PHYSICAL_DESCRIPTOR   0x2300
/* 0x24 - 0x2F: Reserved */

/* 7.2 Class-Specific Requests - bRequest values */
#define HID_GET_REPORT            0x01          /* Mandatory */
#define HID_GET_IDLE              0x02
#define HID_GET_PROTOCOL          0x03          /* Required only for boot devices */
/* 0x04 - 0x08 reserved */
#define HID_SET_REPORT            0x09
#define HID_SET_IDLE              0x0A
#define HID_SET_PROTOCOL          0x0B          /* Required only for boot devices */
```

2.6 USB Device Descriptor

endpoint0.xc is where the standard USB device descriptor is declared for the HID class 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 */
    0x02,                   /* 3  bcdUSB */
    0x00,                   /* 4  bDeviceClass */
    0x00,                   /* 5  bDeviceSubClass */
    0x00,                   /* 6  bDeviceProtocol */
    0x40,                   /* 7  bMaxPacketSize */
    (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 */
    0x02,                   /* 15 iProduct */
    0x00,                   /* 16 iSerialNumber */
    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 HID mouse 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 HID class device the configuration descriptor which is read by the host is as follows.

```
static unsigned char cfgDesc[] = {
    0x09,                   /* 0  bLength */
    0x02,                   /* 1  bDescriptortype */
    0x22, 0x00,             /* 2  wTotalLength */
    0x01,                   /* 4  bNumInterfaces */
    0x01,                   /* 5  bConfigurationValue */
    0x03,                   /* 6  iConfiguration */
    0x80,                   /* 7  bmAttributes */
    0xC8,                   /* 8  bMaxPower */

    0x09,                   /* 0  bLength */
    0x04,                   /* 1  bDescriptorType */
    0x00,                   /* 2  bInterfacecNumber */
    0x00,                   /* 3  bAlternateSetting */
    0x01,                   /* 4: bNumEndpoints */
    0x03,                   /* 5: bInterfaceClass */
    0x00,                   /* 6: bInterfaceSubClass */
    0x02,                   /* 7: bInterfaceProtocol*/
    0x00,                   /* 8  iInterface */

    0x09,                   /* 0  bLength. Note this is currently
                               replicated in hidDescriptor[] below */
    0x21,                   /* 1  bDescriptorType (HID) */
    0x10,                   /* 2  bcdHID */
    0x11,                   /* 3  bcdHID */
    0x00,                   /* 4  bCountryCode */
    0x01,                   /* 5  bNumDescriptors */
    0x22,                   /* 6  bDescriptorType[0] (Report) */
    0x48,                   /* 7  wDescriptorLength */
    0x00,                   /* 8  wDescriptorLength */

    0x07,                   /* 0  bLength */
    0x05,                   /* 1  bDescriptorType */
    0x81,                   /* 2  bEndpointAddress */
    0x03,                   /* 3  bmAttributes */
    0x40,                   /* 4  wMaxPacketSize */
    0x00,                   /* 5  wMaxPacketSize */
    0x01                    /* 6  bInterval */
};
```

From this you can see that the USB HID class defines described earlier are encoded into the configuration descriptor along with the bulk USB endpoint description for allowing the HID mouse device to report information to the host. This endpoint allows us to simulation a mouse device inside our application and report the mouse movement information.

2.8 USB HID Class Descriptor

For USB HID class devices there is a descriptor that is device in the HID device class specification which needs to be provided to the host in addition to the default decriptor types described above. The host will request this descriptor from the device when it enumerates as a HID class device. The HID descriptor for our mouse demo application is as follows.

```
                                replicated in hidDescriptor[] below */
    0x21,                   /* 1  bDescriptorType (HID) */
    0x10,                   /* 2  bcdHID */
    0x11,                   /* 3  bcdHID */
    0x00,                   /* 4  bCountryCode */
    0x01,                   /* 5  bNumDescriptors */
    0x22,                   /* 6  bDescriptorType[0] (Report) */
    0x48,                   /* 7  wDescriptorLength */
    0x00,                   /* 8  wDescriptorLength */

    0x07,                   /* 0  bLength */
    0x05,                   /* 1  bDescriptorType */
    0x81,                   /* 2  bEndpointAddress */
    0x03,                   /* 3  bmAttributes */
    0x40,                   /* 4  wMaxPacketSize */
    0x00,                   /* 5  wMaxPacketSize */
    0x01                    /* 6  bInterval */
};
```

2.9 USB HID Report Descriptor

Along with the HID class descriptor there is a HID report descriptor which describes to the host the usage of the device and the data it will be reporting when it communicates. As HID devices are supported by standard drivers on a host machine this allow a level of configuration between the host and the device. The HID report descriptor for our example application is below.

```
static unsigned char hidReportDescriptor[] =
{
    0x05, 0x01,          // Usage page (desktop)
    0x09, 0x02,          // Usage (mouse)
    0xA1, 0x01,          // Collection (app)
    0x05, 0x09,          // Usage page (buttons)
    0x19, 0x01,
    0x29, 0x03,
    0x15, 0x00,          // Logical min (0)
    0x25, 0x01,          // Logical max (1)
    0x95, 0x03,          // Report count (3)
    0x75, 0x01,          // Report size (1)
    0x81, 0x02,          // Input (Data, Absolute)
    0x95, 0x01,          // Report count (1)
    0x75, 0x05,          // Report size (5)
    0x81, 0x03,          // Input (Absolute, Constant)
    0x05, 0x01,          // Usage page (desktop)
    0x09, 0x01,          // Usage (pointer)
    0xA1, 0x00,          // Collection (phys)
    0x09, 0x30,          // Usage (x)
    0x09, 0x31,          // Usage (y)
    0x15, 0x81,          // Logical min (-127)
    0x25, 0x7F,          // Logical max (127)
    0x75, 0x08,          // Report size (8)
    0x95, 0x02,          // Report count (2)
    0x81, 0x06,          // Input (Data, Rel=0x6, Abs=0x2)
    0xC0,                // End collection
    0x09, 0x38,          // Usage (Wheel)
    0x95, 0x01,          // Report count (1)
    0x81, 0x02,          // Input (Data, Relative)
    0x09, 0x3C,          // Usage (Motion Wakeup)
    0x15, 0x00,          // Logical min (0)
    0x25, 0x01,          // Logical max (1)
    0x75, 0x01,          // Report size (1)
    0x95, 0x01,          // Report count (1)
    0xB1, 0x22,          // Feature (No preferred, Variable)
    0x95, 0x07,          // Report count (7)
    0xB1, 0x01,          // Feature (Constant)
    0xC0                 // End collection
};
```

2.10 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 */
static char * unsafe stringDescriptors[]=
{
    "\x09\x04",             // Language ID string (US English)
    "XMOS",                 // iManufacturer
    "Example HID Mouse",    // iProduct
    "Config",               // iConfiguration
};
```

2.11 USB HID Class requests

Inside endpoint0.xc there is a function for handling the USB HID device class specific requests. The code for handling these requests is shown as follows:

```
/* HID Class Requests */
XUD_Result_t HidInterfaceClassRequests(XUD_ep c_ep0_out, XUD_ep c_ep0_in, USB_SetupPacket_t sp)
{
    unsigned buffer[64];
    unsigned tmp;

    switch(sp.bRequest)
    {
        case HID_GET_REPORT:

            /* Mandatory. Allows sending of report over control pipe */
            /* Send a hid report - note the use of asm due to shared mem */
            asm("ldaw %0, dp[g_reportBuffer]": "=r"(tmp));
            asm("ldw %0, %1[0]": "=r"(tmp) : "r"(tmp));
            buffer[0] = tmp;

            return XUD_DoGetRequest(c_ep0_out, c_ep0_in, (buffer, unsigned char []), 4, sp.wLength);
            break;

        case HID_GET_IDLE:
            /* Return the current Idle rate - optional for a HID mouse */

            /* Do nothing - i.e. STALL */
            break;

        case HID_GET_PROTOCOL:
            /* Required only devices supporting boot protocol devices,
             * which this example does not */

            /* Do nothing - i.e. STALL */
            break;

         case HID_SET_REPORT:
            /* The host sends an Output or Feature report to a HID
             * using a cntrol transfer - optional */

            /* Do nothing - i.e. STALL */
            break;

        case HID_SET_IDLE:
            /* Set the current Idle rate - this is optional for a HID mouse
             * (Bandwidth can be saved by limiting the frequency that an
             * interrupt IN EP when the data hasn't changed since the last
             * report */

            /* Do nothing - i.e. STALL */
            break;

        case HID_SET_PROTOCOL:
            /* Required only devices supporting boot protocol devices,
             * which this example does not */

            /* Do nothing - i.e. STALL */
            break;
    }

    return XUD_RES_ERR;
}
```

These HID specific requests 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.

2.12 USB HID 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 HID class.

```
switch(bmRequestType)
{
    /* Direction: Device-to-host
     * Type: Standard
     * Recipient: Interface
     */
    case USB_BMREQ_D2H_STANDARD_INT:

        if(sp.bRequest == USB_GET_DESCRIPTOR)
        {
            /* HID Interface is Interface 0 */
            if(sp.wIndex == 0)
            {
                /* Look at Descriptor Type (high-byte of wValue) */
                unsigned short descriptorType = sp.wValue & 0xff00;

                switch(descriptorType)
                {
                    case HID_HID:
                        result = XUD_DoGetRequest(ep0_out, ep0_in, hidDescriptor, sizeof(hidDescriptor), sp.
                          ↪ wLength);
                        break;

                    case HID_REPORT:
                        result = XUD_DoGetRequest(ep0_out, ep0_in, hidReportDescriptor, sizeof(
                          ↪ hidReportDescriptor), sp.wLength);
                        break;
                }
            }
        }
        break;

    /* 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 HID 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 = HidInterfaceClassRequests(ep0_out, ep0_in, sp);
        }
        break;
}
```

2.13 Reporting HID mouse data to the host

The application endpoint for reporting mouse movement data to the host machine is implemented in the file `main.xc`. This is contained within the function `hid_mouse()` which is shown below:

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

/*
 * This function responds to the HID requests
 * - It draws a square using the mouse moving 40 pixels in each direction
 * - The sequence repeats every 500 requests.
 */
void hid_mouse(chanend chan_ep_hid)
{
    int counter = 0;
    int state = 0;

    XUD_ep ep_hid = XUD_InitEp(chan_ep_hid);

    while (1)
    {
        int x;
        g_reportBuffer[1] = 0;
        g_reportBuffer[2] = 0;

        /* Move the pointer around in a square (relative) */
        counter++;
        if (counter >= 500)
        {
            counter = 0;
            if (state == 0)
            {
                g_reportBuffer[1] = 40;
                g_reportBuffer[2] = 0;
                state+=1;
            }
            else if (state == 1)
            {
                g_reportBuffer[1] = 0;
                g_reportBuffer[2] = 40;
                state+=1;
            }
            else if (state == 2)
            {
                g_reportBuffer[1] = -40;
                g_reportBuffer[2] = 0;
                state+=1;
            }
            else if (state == 3)
            {
                g_reportBuffer[1] = 0;
                g_reportBuffer[2] = -40;
                state = 0;
            }
        }

        /* Send the buffer off to the host.  Note this will return when complete */
        XUD_SetBuffer(ep_hid, g_reportBuffer, 4);
    }
}
```

From this you can see the following.

- A buffer is declared to communicate the HID report data to the host, this is accessed via shared memory from endpoint0 and also used from this function.
- This task operates inside a `while (1)` loop which streams mouse movement data to the host machine. It moves the mouse pointer in a square shape on the host machine desktop.
- A blocking call is made to the XMOS USB device library to send data to the host machine at every loop iteration

- The function emulates a mouse device in the code but this could easily be replaced by connecting an external piece of hardware

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.

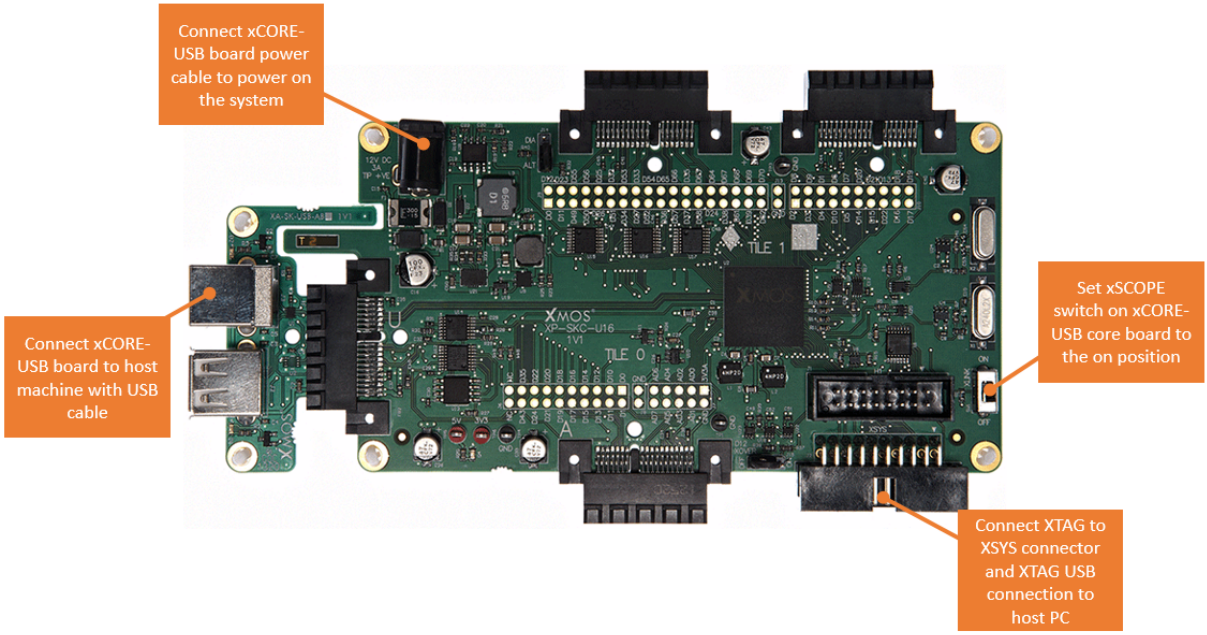

Figure 3: XMOS xCORE-USB sliceKIT

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 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-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 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_hid_mouse_demo.ex          <-- Download and execute the xCORE code
```

Once this command has executed the HID mouse device will have enumerated on your host machine.

B.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio we use the run mechanism to download code to xCORE device. 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 HID mouse device will have enumerated on your host machine.

B.3 Running the HID mouse demo

The USB mouse device once enumerated will start acting as if you have plugged a new USB mouse into your host machine.

This will be shown to be working by the mouse pointer which will now be moving around the screen controlled by the HID endpoint code running on the xCORE microcontroller as described in this application note.

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 HID Class Specification, USB.org:

http://www.usb.org/developers/devclass_docs/HID1_11.pdf

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 endpoint0.xc

```
/*
 * @brief Implements endpoint zero for an example HID mouse device.
 */
#include <xs1.h>
#include "usb_device.h"
#include "usb_std_requests.h"
#include "usb_std_descriptors.h"
#include "hid.h"

/* USB HID Device Product Defines */
#define BCD_DEVICE   0x1000
#define VENDOR_ID    0x20B1
#define PRODUCT_ID   0x1010

/* Standard HID Request Defines */

/* 7. Requests */

/* 7.1 Standard Requests  - Class Descriptor Types - High byte of wValue
 * The following defines valid types of Class descriptors */

#define HID_HID                  0x2100
#define HID_REPORT               0x2200
#define HID_PHYSICAL_DESCRIPTOR  0x2300
/* 0x24 - 0x2F: Reserved */

/* 7.2 Class-Specific Requests - bRequest values */
#define HID_GET_REPORT           0x01          /* Mandatory */
#define HID_GET_IDLE             0x02
#define HID_GET_PROTOCOL         0x03          /* Required only for boot devices */
/* 0x04 - 0x08 reserved */
#define HID_SET_REPORT           0x09
#define HID_SET_IDLE             0x0A
#define HID_SET_PROTOCOL         0x0B          /* Required only for boot devices */

/* Device Descriptor */
static unsigned char devDesc[] =
{
    0x12,                   /* 0  bLength */
    USB_DESCTYPE_DEVICE,    /* 1  bdescriptorType */
    0x00,                   /* 2  bcdUSB */
    0x02,                   /* 3  bcdUSB */
    0x00,                   /* 4  bDeviceClass */
    0x00,                   /* 5  bDeviceSubClass */
    0x00,                   /* 6  bDeviceProtocol */
    0x40,                   /* 7  bMaxPacketSize */
    (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 */
    0x02,                   /* 15 iProduct */
    0x00,                   /* 16 iSerialNumber */
    0x01                    /* 17 bNumConfigurations */
};


/* Configuration Descriptor */
static unsigned char cfgDesc[] = {
    0x09,                  /* 0  bLength */
    0x02,                  /* 1  bDescriptortype */
    0x22, 0x00,            /* 2  wTotalLength */
    0x01,                  /* 4  bNumInterfaces */
    0x01,                  /* 5  bConfigurationValue */
    0x03,                  /* 6  iConfiguration */
    0x80,                  /* 7  bmAttributes */
    0xC8,                  /* 8  bMaxPower */
```

```
    0x09,                   /* 0  bLength */
    0x04,                   /* 1  bDescriptorType */
    0x00,                   /* 2  bInterfacecNumber */
    0x00,                   /* 3  bAlternateSetting */
    0x01,                   /* 4: bNumEndpoints */
    0x03,                   /* 5: bInterfaceClass */
    0x00,                   /* 6: bInterfaceSubClass */
    0x02,                   /* 7: bInterfaceProtocol*/
    0x00,                   /* 8  iInterface */

    0x09,                   /* 0  bLength. Note this is currently
                               replicated in hidDescriptor[] below */
    0x21,                   /* 1  bDescriptorType (HID) */
    0x10,                   /* 2  bcdHID */
    0x11,                   /* 3  bcdHID */
    0x00,                   /* 4  bCountryCode */
    0x01,                   /* 5  bNumDescriptors */
    0x22,                   /* 6  bDescriptorType[0] (Report) */
    0x48,                   /* 7  wDescriptorLength */
    0x00,                   /* 8  wDescriptorLength */

    0x07,                   /* 0  bLength */
    0x05,                   /* 1  bDescriptorType */
    0x81,                   /* 2  bEndpointAddress */
    0x03,                   /* 3  bmAttributes */
    0x40,                   /* 4  wMaxPacketSize */
    0x00,                   /* 5  wMaxPacketSize */
    0x01                    /* 6  bInterval */
};

static unsigned char hidDescriptor[] =
{
    0x09,                /* 0  bLength */
    0x21,                /* 1  bDescriptorType (HID) */
    0x10,                /* 2  bcdHID */
    0x11,                /* 3  bcdHID */
    0x00,                /* 4  bCountryCode */
    0x01,                /* 5  bNumDescriptors */
    0x22,                /* 6  bDescriptorType[0] (Report) */
    0x48,                /* 7  wDescriptorLength */
    0x00,                /* 8  wDescriptorLength */
};

/* HID Report Descriptor */
static unsigned char hidReportDescriptor[] =
{
    0x05, 0x01,          // Usage page (desktop)
    0x09, 0x02,          // Usage (mouse)
    0xA1, 0x01,          // Collection (app)
    0x05, 0x09,          // Usage page (buttons)
    0x19, 0x01,
    0x29, 0x03,
    0x15, 0x00,          // Logical min (0)
    0x25, 0x01,          // Logical max (1)
    0x95, 0x03,          // Report count (3)
    0x75, 0x01,          // Report size (1)
    0x81, 0x02,          // Input (Data, Absolute)
    0x95, 0x01,          // Report count (1)
    0x75, 0x05,          // Report size (5)
    0x81, 0x03,          // Input (Absolute, Constant)
    0x05, 0x01,          // Usage page (desktop)
    0x09, 0x01,          // Usage (pointer)
    0xA1, 0x00,          // Collection (phys)
    0x09, 0x30,          // Usage (x)
    0x09, 0x31,          // Usage (y)
    0x15, 0x81,          // Logical min (-127)
    0x25, 0x7F,          // Logical max (127)
    0x75, 0x08,          // Report size (8)
    0x95, 0x02,          // Report count (2)
    0x81, 0x06,          // Input (Data, Rel=0x6, Abs=0x2)
    0xC0,                // End collection
    0x09, 0x38,          // Usage (Wheel)
    0x95, 0x01,          // Report count (1)
    0x81, 0x02,          // Input (Data, Relative)
```

```
    0x09, 0x3C,          // Usage (Motion Wakeup)
    0x15, 0x00,          // Logical min (0)
    0x25, 0x01,          // Logical max (1)
    0x75, 0x01,          // Report size (1)
    0x95, 0x01,          // Report count (1)
    0xB1, 0x22,          // Feature (No preferred, Variable)
    0x95, 0x07,          // Report count (7)
    0xB1, 0x01,          // Feature (Constant)
    0xC0                 // End collection
};

unsafe{
/* String table */
static char * unsafe stringDescriptors[]=
{
    "\x09\x04",             // Language ID string (US English)
    "XMOS",                 // iManufacturer
    "Example HID Mouse",    // iProduct
    "Config",               // iConfiguration
};
}

/* HID Class Requests */
XUD_Result_t HidInterfaceClassRequests(XUD_ep c_ep0_out, XUD_ep c_ep0_in, USB_SetupPacket_t sp)
{
    unsigned buffer[64];
    unsigned tmp;

    switch(sp.bRequest)
    {
        case HID_GET_REPORT:

            /* Mandatory. Allows sending of report over control pipe */
            /* Send a hid report - note the use of asm due to shared mem */
            asm("ldaw %0, dp[g_reportBuffer]": "=r"(tmp));
            asm("ldw %0, %1[0]": "=r"(tmp) : "r"(tmp));
            buffer[0] = tmp;

            return XUD_DoGetRequest(c_ep0_out, c_ep0_in, (buffer, unsigned char []), 4, sp.wLength);
            break;

        case HID_GET_IDLE:
            /* Return the current Idle rate - optional for a HID mouse */

            /* Do nothing - i.e. STALL */
            break;

        case HID_GET_PROTOCOL:
            /* Required only devices supporting boot protocol devices,
             * which this example does not */

            /* Do nothing - i.e. STALL */
            break;

         case HID_SET_REPORT:
            /* The host sends an Output or Feature report to a HID
             * using a cntrol transfer - optional */

            /* Do nothing - i.e. STALL */
            break;

        case HID_SET_IDLE:
            /* Set the current Idle rate - this is optional for a HID mouse
             * (Bandwidth can be saved by limiting the frequency that an
             * interrupt IN EP when the data hasn't changed since the last
             * report */

            /* Do nothing - i.e. STALL */
            break;

        case HID_SET_PROTOCOL:
            /* Required only devices supporting boot protocol devices,
             * which this example does not */

            /* Do nothing - i.e. STALL */
```

```
            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_ep ep0_in  = XUD_InitEp(chan_ep0_in);

    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 ((bmRequestType == USB_BMREQ_H2D_STANDARD_DEV) &&
                (sp.bRequest == USB_SET_ADDRESS))
            {
              // Host has set device address, value contained in sp.wValue
            }

            switch(bmRequestType)
            {
                /* Direction: Device-to-host
                 * Type: Standard
                 * Recipient: Interface
                 */
                case USB_BMREQ_D2H_STANDARD_INT:

                    if(sp.bRequest == USB_GET_DESCRIPTOR)
                    {
                        /* HID Interface is Interface 0 */
                        if(sp.wIndex == 0)
                        {
                            /* Look at Descriptor Type (high-byte of wValue) */
                            unsigned short descriptorType = sp.wValue & 0xff00;

                            switch(descriptorType)
                            {
                                case HID_HID:
                                    result = XUD_DoGetRequest(ep0_out, ep0_in, hidDescriptor, sizeof(
                                      ↪ hidDescriptor), sp.wLength);
                                    break;

                                case HID_REPORT:
                                    result = XUD_DoGetRequest(ep0_out, ep0_in, hidReportDescriptor, sizeof(
                                      ↪ hidReportDescriptor), sp.wLength);
                                    break;
                            }
                        }
                    }
                    break;

                /* 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 HID 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 = HidInterfaceClassRequests(ep0_out, ep0_in, sp);
                    }
                    break;
            }
        }

        /* If we haven't handled the request about 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);
        }
    }
}
//:
```

D.2 Source code for main.xc

```
/**
 * The copyrights, all other intellectual and industrial
 * property rights are retained by XMOS and/or its licensors.
 * Terms and conditions covering the use of this code can
 * be found in the Xmos End User License Agreement.
 *
 * Copyright XMOS Ltd 2013
 *
 * In the case where this code is a modification of existing code
 * under a separate license, the separate license terms are shown
 * below. The modifications to the code are still covered by the
 * copyright notice above.
 *
 **/

#include "xud.h"

#define p_usb_rst null
#define clk_usb_rst null
#define PWR_MODE XUD_PWR_SELF
#define XUD_EP_COUNT_OUT   1
#define XUD_EP_COUNT_IN    2

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

/* Endpoint type tables - informs XUD what the transfer types for each Endpoint in use and also
 * if the endpoint wishes to be informed of USB bus resets
 */
XUD_EpType epTypeTableOut[XUD_EP_COUNT_OUT] = {XUD_EPTYPE_CTL | XUD_STATUS_ENABLE};
XUD_EpType epTypeTableIn[XUD_EP_COUNT_IN] =   {XUD_EPTYPE_CTL | XUD_STATUS_ENABLE, XUD_EPTYPE_BUL};
```

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

/*
 * This function responds to the HID requests
 * - It draws a square using the mouse moving 40 pixels in each direction
 * - The sequence repeats every 500 requests.
 */
void hid_mouse(chanend chan_ep_hid)
{
    int counter = 0;
    int state = 0;

    XUD_ep ep_hid = XUD_InitEp(chan_ep_hid);

    while (1)
    {
        int x;
        g_reportBuffer[1] = 0;
        g_reportBuffer[2] = 0;

        /* Move the pointer around in a square (relative) */
        counter++;
        if (counter >= 500)
        {
            counter = 0;
            if (state == 0)
            {
                g_reportBuffer[1] = 40;
                g_reportBuffer[2] = 0;
                state+=1;
            }
            else if (state == 1)
            {
                g_reportBuffer[1] = 0;
                g_reportBuffer[2] = 40;
                state+=1;
            }
            else if (state == 2)
            {
                g_reportBuffer[1] = -40;
                g_reportBuffer[2] = 0;
                state+=1;
            }
            else if (state == 3)
            {
                g_reportBuffer[1] = 0;
                g_reportBuffer[2] = -40;
                state = 0;
            }
        }

        /* Send the buffer off to the host.  Note this will return when complete */
        XUD_SetBuffer(ep_hid, g_reportBuffer, 4);
    }
}

/* The main function runs three cores: the XUD manager, Endpoint 0, and a HID endpoint. An array of
 * channels is used for both IN and OUT endpoints, endpoint zero requires both, HID requires just an
 * IN endpoint to send HID reports to the host.
 */
int main()
{
    chan c_ep_out[XUD_EP_COUNT_OUT], c_ep_in[XUD_EP_COUNT_IN];

    par
    {
        on USB_TILE: XUD_Manager(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                                 null, epTypeTableOut, epTypeTableIn,
                                 p_usb_rst, clk_usb_rst, -1, XUD_SPEED_HS, PWR_MODE);

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

        on USB_TILE: hid_mouse(c_ep_in[1]);
```

```
    }

    return 0;
}
```