Application Note: AN00130

USB HID Class - Extended on sliceKIT

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 enhancement to AN00129 for extending the USB HID device to interface with hardware which can provide input for a USB mouse.

This example uses the ADC on the XMOS xCORE-USB device to interface to a mixed signal sliceCARD and provide a joystick interface which allows the USB HID to be controlled.

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.

This application note describes extending XMOS application note AN00129 for the xCORE-USB sliceKIT platform.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0.0
- XMOS USB library - Version 3.1.0
- XMOS U series support 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 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 in the appendix.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].
- Understanding of USB HID class implementation from application note AN00129
- 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

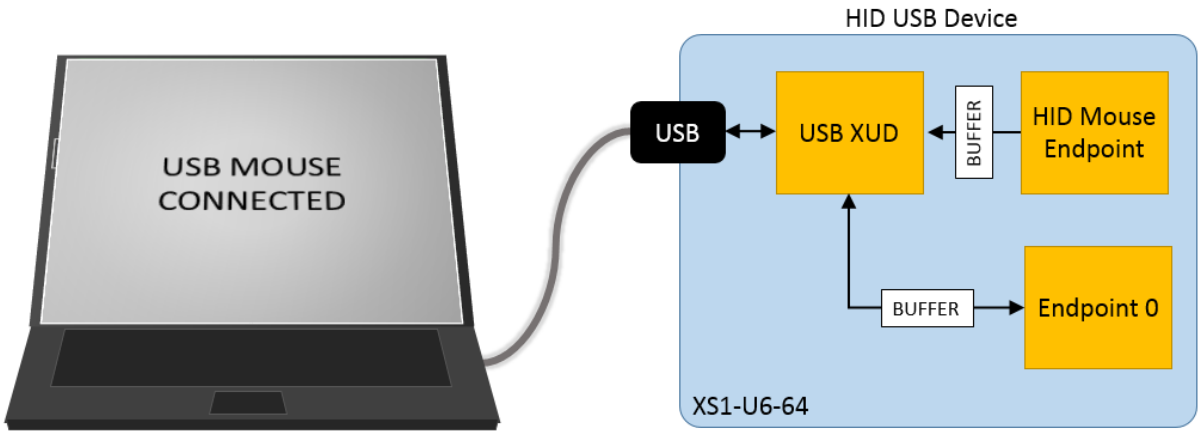

Figure 1: Block diagram of extended USB HID application example

2 USB HID Class - Application note AN00129

This application note takes the majority of its source code from AN00129 to implement the HID 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 HID endpoint.

The original HID mouse application code has been replaced with a extended function which now uses the ADC on the xCORE-USB device to read from a joystick and generate the HID control data.

3 USB HID Class Extended on sliceKIT - 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 the 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 HID device class application example.

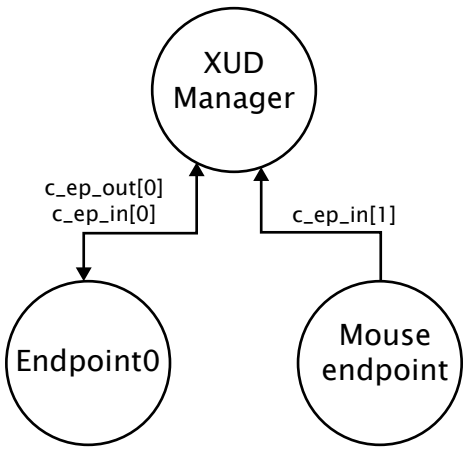

Figure 2: Task diagram of USB HID application example

In this example we take the simple USB HID mouse endpoint and replace it with a version which interfaces to a joystick on the XMOS mixed signal slice using the ADC on the xCORE-USB device.

This allows the example to be extended to produce a USB mouse where movement is controlled by user input. This demonstrates how a USB HID can be customized to interface to the custom control required by the application.

3.1 Makefile additions for this example

There are additions to the Makefile provided with AN00129, these relate to using the support library for the xCORE-USB series to allow access to the ADC.

To start using the U series support library, you need to add `lib_u_series_support` to your `Makefile`:

```
USED_MODULES = ... lib_u_series_support ...
```

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

```
#include <u_series_adc.h>
```

3.2 xCORE-USB ADC port declaration

In order to use the ADC on the xCORE-USB device a port resource needs to be declared to be used as the trigger for the ADC. This is defined in the code below,

```
/* Port for ADC triggering */
on USB_TILE: out port p_adc_trig = PORT_ADC_TRIGGER;
```

3.3 HID mouse endpoint with joystick control

In order to replace the HID mouse endpoint in AN00129 using the ADC of the xCORE-USB device a replacement function has been written. This is contained within the file `hid_mouse_extended.xc` and is prototyped as follows,

```
void hid_mouse_extended(chanend c_ep_hid, chanend c_adc)
```

In this function the ADC is configured, the following code uses the standard library support for configuring the ADC on the xCORE-USB device. This sets up the inputs which are enabled, the required number of samples per packet and the number of bits per sample. This configuration is passed to the `adc_enable` function to start the ADC.

```
/* Configure and enable the ADC in the U device */
adc_config_t adc_config = { { 0, 0, 0, 0, 0, 0, 0, 0 }, 0, 0, 0 };

adc_config.input_enable[2] = 1;
adc_config.input_enable[3] = 1;
adc_config.samples_per_packet = 2;
adc_config.bits_per_sample = ADC_32_BPS;
adc_config.calibration_mode = 0;

adc_enable(usb_tile, c_adc, p_adc_trig, adc_config);

/* Unsafe region so we can use shared memory. */
```

As described in AN00129 there is a global buffer `g_reportBuffer` which is used to signal HID report data to endpoint0. The code below initialises this buffer via a pointer for this USB mouse endpoint.

```
/* Initialise the HID report buffer */
p_reportBuffer[1] = 0;
p_reportBuffer[2] = 0;
```

Next it is possible to get samples from the ADC this is done by triggering the ADC using `adc_trigger_packet()` and reading data back using `adc_read_packet()`. The data returned from the ADC relates to the x and y coordinates of the joystick on the mixed signal slice. The code for performing this operation is as follows,

```
/* Get ADC input */
adc_trigger_packet(p_adc_trig, adc_config);
adc_read_packet(c_adc, adc_config, data);
x = data[0];
y = data[1];
```

The data for the x coordinate is processed by the following code to transform the value so that it is suitable for reporting back to the host as the HID device report.

This involves using the defines for the sensitivity and joystick dead zone that is defined at the top of the file `hid_mouse_extended.xc`.

```
/* Move horizontal axis of pointer based on ADC val (absolute) */
x = ((x >> SHIFT) & MASK) - OFFSET - initialX;
if (x > DEAD_ZONE)
    p_reportBuffer[1] = (x - DEAD_ZONE)/(10 - SENSITIVITY);
else if (x < -DEAD_ZONE)
    p_reportBuffer[1] = (x + DEAD_ZONE)/(10 - SENSITIVITY);
```

The data for the y coordinate is processed by the following code to transform the value so that it is suitable for reporting back to the host as the HID device report.

This involves using the defines for the sensitivity and joystick dead zone that is defined at the top of the file `hid_mouse_extended.xc`.

```
/* Move vertical axis of pointer based on ADC val (absolute) */
y = ((y >> SHIFT) & MASK) - OFFSET - initialY;
if (y > DEAD_ZONE)
    p_reportBuffer[2] = (y - DEAD_ZONE)/(10 - SENSITIVITY);
else if (y < -DEAD_ZONE)
    p_reportBuffer[2] = (y + DEAD_ZONE)/(10 - SENSITIVITY);
```

Once the data processing is complete the HID report is sent back to the USB host using the XMOS USB library function `XUD_SetBuffer()`.

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

The function `hid_mouse_extended` continues to operate in an infinite loop reporting data values back to the USB host as a HID report by reading from the joystick.

3.4 Changes to the main() function of AN00129

The code below shows the changes required to main() in application note AN00129 in order to enable the new extended HID mouse endpoint. This is a small change which will enable the ADC and call the new function provided with this application note.

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

    par
    {
      on tile[0]: xud(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                      null, XUD_SPEED_HS, XUD_PWR_SELF);

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

      on tile[0]: hid_mouse_extended(c_ep_in[1], c_adc);

      xs1_su_adc_service(c_adc);
    }

    return 0;
}
```

3.5 Setting up the ADC in the application main()

The ADC on the xCORE-USB tile is configured in main with the following code which creates an xCONNECT channel which will be connected to the ADC.

```
chan c_adc;
```

This channel can then be connected to the ADC on the xCORE-USB device by using the following call in main(),

```
xs1_su_adc_service(c_adc);
```

The other end of the xCONNECT channel c_adc is passed to the new extended hid mouse function in main() as follows,

```
on tile[0]: hid_mouse_extended(c_ep_in[1], c_adc);
```

3.6 Adding the extended HID mouse endpoint to main()

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

```
#include "hid_mouse_extended.h"
```

4 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.

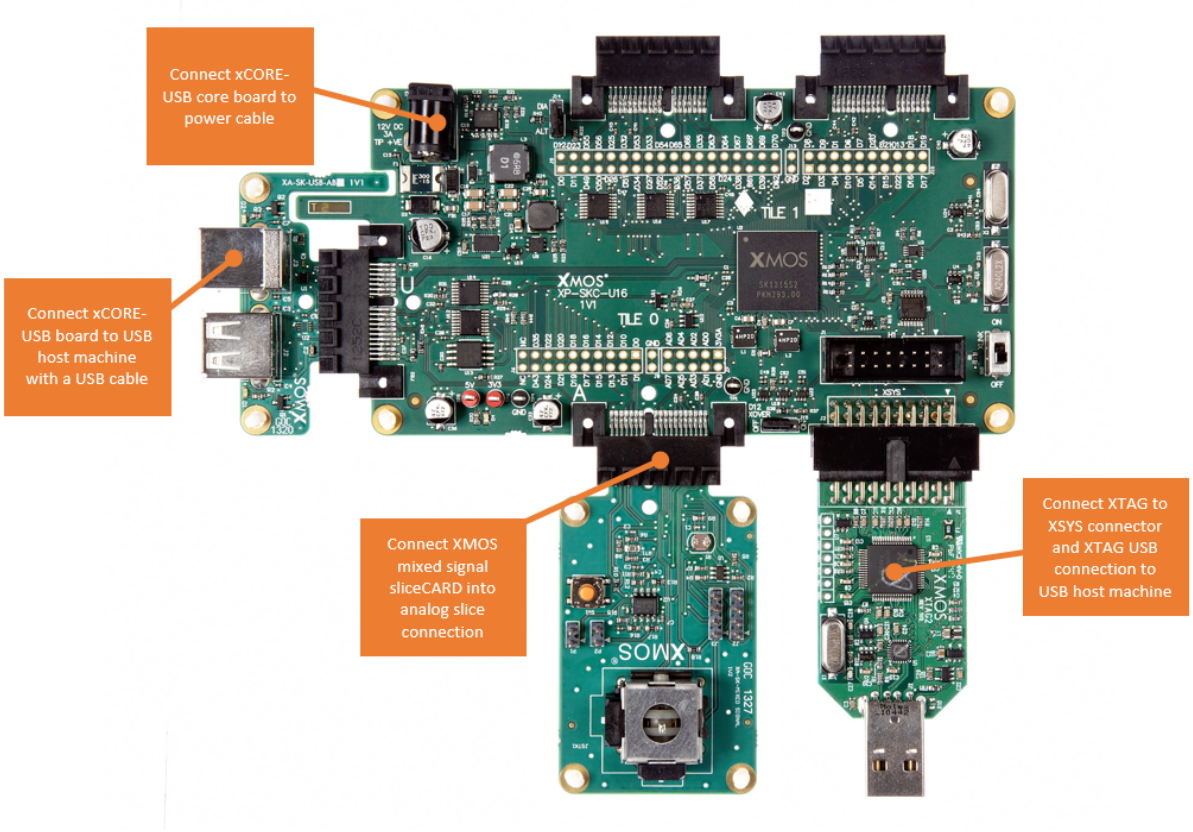

Figure 3: XMOS xCORE-USB sliceKIT

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 mixed signal sliceCARD is connected to the analog connector of the xCORE-USB sliceKIT
- The xCORE-USB core board should have the power cable connected

5 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.

5.1 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.xe          <-- Download and execute the xCORE code
```

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

5.2 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.

5.3 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.

By moving the joystick on the mixed signal slice you will be able to control the mouse pointer on your USB host machine.

6 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

7 Full source code listing

7.1 Source code for main.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include "usb.h"
#include "u_series_adc.h"
#include "hid_mouse_extended.h"

/* 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[4] = {0, 0, 0, 0};

#define XUD_EP_COUNT_OUT   1
#define XUD_EP_COUNT_IN    2

/* 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];
    chan c_adc;

    par
    {
      on tile[0]: xud(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                      null, XUD_SPEED_HS, XUD_PWR_SELF);

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

      on tile[0]: hid_mouse_extended(c_ep_in[1], c_adc);

      xs1_su_adc_service(c_adc);
    }

    return 0;
}
```

7.2 Source code for hid_mouse_extended.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include "usb.h"
#include "u_series_adc.h"
#include "hid_mouse_extended.h"

/* Global report buffer, global since used by Endpoint0 core */
extern unsigned char g_reportBuffer[4];

/* Port for ADC triggering */
on USB_TILE: out port p_adc_trig = PORT_ADC_TRIGGER;

#define BITS 5          // Overall precision
#define DEAD_ZONE 2     // Ensure that the mouse is stable when the joystick is not used
#define SENSITIVITY 0   // Sensitivity range 0 - 9
#define SHIFT  (32 - BITS)
#define MASK   ((1 << BITS) - 1)
#define OFFSET (1 << (BITS - 1))

/*
 * This function responds to the HID requests - it moves the pointers x axis based on ADC input
 */
void hid_mouse_extended(chanend c_ep_hid, chanend c_adc)
{
    int initialDone = 0;
    int initialX = 0;
    int initialY = 0;
```

```
    /* Initialise the XUD endpoint */
    XUD_ep ep_hid = XUD_InitEp(c_ep_hid, XUD_EPTYPE_BUL);

    /* Configure and enable the ADC in the U device */
    adc_config_t adc_config = { { 0, 0, 0, 0, 0, 0, 0, 0 }, 0, 0, 0 };

    adc_config.input_enable[2] = 1;
    adc_config.input_enable[3] = 1;
    adc_config.samples_per_packet = 2;
    adc_config.bits_per_sample = ADC_32_BPS;
    adc_config.calibration_mode = 0;

    adc_enable(usb_tile, c_adc, p_adc_trig, adc_config);

    /* Unsafe region so we can use shared memory. */
    while (1)
    {
      unsafe {
        char * unsafe p_reportBuffer = g_reportBuffer;
        unsigned data[2];
        int x;
        int y;

        /* Initialise the HID report buffer */
        p_reportBuffer[1] = 0;
        p_reportBuffer[2] = 0;

        /* Get ADC input */
        adc_trigger_packet(p_adc_trig, adc_config);
        adc_read_packet(c_adc, adc_config, data);
        x = data[0];
        y = data[1];

        /* Move horizontal axis of pointer based on ADC val (absolute) */
        x = ((x >> SHIFT) & MASK) - OFFSET - initialX;
        if (x > DEAD_ZONE)
            p_reportBuffer[1] = (x - DEAD_ZONE)/(10 - SENSITIVITY);
        else if (x < -DEAD_ZONE)
            p_reportBuffer[1] = (x + DEAD_ZONE)/(10 - SENSITIVITY);

        /* Move vertical axis of pointer based on ADC val (absolute) */
        y = ((y >> SHIFT) & MASK) - OFFSET - initialY;
        if (y > DEAD_ZONE)
            p_reportBuffer[2] = (y - DEAD_ZONE)/(10 - SENSITIVITY);
        else if (y < -DEAD_ZONE)
            p_reportBuffer[2] = (y + DEAD_ZONE)/(10 - SENSITIVITY);

        if (!initialDone)
        {
            initialX = x;
            initialY = y;
            initialDone = 1;
        }

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

7.3 Source code for hid_mouse_extended.h

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#ifndef HID_MOUSE_EXTENDED_H
#define HID_MOUSE_EXTENDED_H
void hid_mouse_extended(chanend c_ep_hid, chanend c_adc);
#endif
```