Application Note: AN00182

USB HID Class - Extended on xCORE-200 Explorer

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-200 explorerKIT platform. The demo application is extended using the functionality described in AN00181 for accesing the accelerometer on the xCORE-200 explorerKIT.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0.0
- XMOS USB library - Version 3.1.0
- XMOS I2C library - Version 2.0.0

Required hardware

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

The example code provided with the application has been implemented and tested on the xCORE-200 explorerKIT but there is no dependancy on this board and it can be modified to run on any development board which uses an xCORE-200 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].
- Application notes AN00129 (USB HID) and AN00181 (Accelerometer)
- 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

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

Design Guide for reference[3].

[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

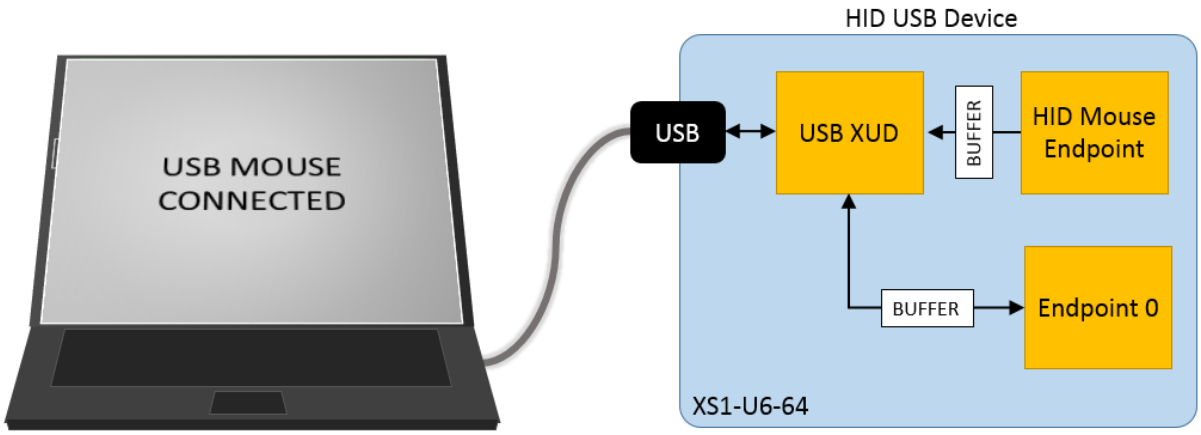

Figure 1: Block diagram of extended USB HID application example

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 an extended function which now uses the accelerometer on the xCORE-200 explorerKIT to generate the HID control data. The accelerometer is interfaced using the XMOS I2C library.

The implementation of the accelerometer interface is not detailed specifically in this application note, for reference, a detailed description of using the I2C interface to access the xCORE-200 explorerKIT accelerometer can be found in application note AN00181.

3 USB HID Class Extended on xCORE-200 Explorer - Application note

The example in this application note uses the XMOS USB device library and shows a simple program that 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
- A task implementing the mouse input interface for reading x and y axis values plue the button state
- A task implementing the I2C interface to the accelerometer

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.

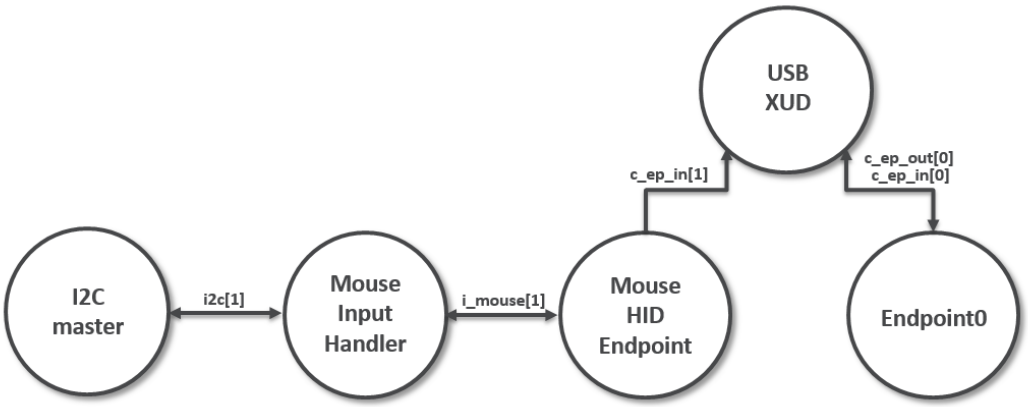

Figure 2: Task diagram of USB HID application example

In this example we take the simple USB HID mouse endpoint and replace it with a version which interfaces to the accelerometer on the xCORE-200 explorerKit.

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 I2C library.

To start using the i2c library, you need to add `lib_i2c` to your `Makefile`:

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

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

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

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

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

3.2 Accelerometer I2C port declaration

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

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

3.3 Button port declaration

The buttons on the xCORE-200 explorerKIT are used in this example as the mouse buttons, these are declared in the code as follows.

```
// Button access
// PORT_4E connected to the 2 Buttons
on tile[0]: port p_button = XS1_PORT_4E;
#define DEBOUNCE_TIME   (XS1_TIMER_HZ/50)
#define BUTTON_PRESSED  0x00
```

3.4 Mouse data input interface

The HID endpoint needs to get data from both the accelerometer and the buttons to report mouse control events back to the USB host via the HID report. In order to provide the HID endpoint with a mechanism to read this data the following interface is defined.

```
interface mouse_input_if {
    struct mouse_input_data get_input_data();
};
```

This interface when called returns a structure containing the mouse input data from the user, the returned structure is defined as follows.

```
struct mouse_input_data {
    int x;
    int y;
    int button_left;
    int button_right;
};
```

The mouse input interface contains 1 interface function which populates the data structure with the values read from the accelerometer and the buttons. This simply assignes the values back into the data structure defined above to be returned to the client. The implementation of this interface function is as follows.

```
void mouse_input(server interface mouse_input_if mi_if, client interface i2c_master_if i2c) {
    init_accelerometer(i2c);
    while (1) {
        select {
            case mi_if.get_input_data() -> struct mouse_input_data data:
            get_accelerometer_data(i2c, data.x, data.y);
            data.button_left = is_button_pressed(1);
            data.button_right = is_button_pressed(0);
            break;
        }
    }
}
```

This function takes the I2C interface as a parameter to allow the mouse input code to read and write to the I2C device. The I2C accelerometer on the xCORE-200 explorerKIT is configured in this function. For more detail on the accelerometer and communicating via I2C refer to AN00181.

3.5 HID mouse endpoint with accelerometer control

In order to replace the HID mouse endpoint in AN00129 using the accelerometer 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 chan_ep_hid, client interface mouse_input_if mi_if)
```

This function is passed a mouse input interface when called, this interface is used to read the mouse input data from the hardware and send the information back to the host via the HID report information.

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
p_reportBuffer[0] = 0;  // Buttons
p_reportBuffer[1] = 0;  // X axis
p_reportBuffer[2] = 0;  // Y axis
```

The main loop of the HID endpoint operates within a `while (1)` loop, within this loop on every iteration a call is made to the to mouse input interface to collect the data to be returned in the HID report. This call to the interface is as follows.

```
// Read from mouse input interface
mi_data = mi_if.get_input_data();
```

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.

```
// Set up HID x axis value report
if (mi_data.x > 10 || mi_data.x < -10) {
  p_reportBuffer[1] = mi_data.x >> 2;
}
```

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.

```
// Set up HID y axis value report
if (mi_data.y > 10 || mi_data.y < -10) {
  p_reportBuffer[2] = (mi_data.y * -1) >> 2;
}
```

For the buttons the left and right button data is processed into the appropriate field of the HID report.

```
// Set up HID button value report
p_reportBuffer[0] |= mi_data.button_left;
p_reportBuffer[0] |= mi_data.button_right << 1;
```

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 accelerometer.

3.6 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 set up the I2C interface and call the new function provided with this application note.

```
int main()
{
    i2c_master_if i2c[1];
    interface mouse_input_if i_mouse_input[1];
    chan c_ep_out[XUD_EP_COUNT_OUT], c_ep_in[XUD_EP_COUNT_IN];

    par
    {
      on tile[1]: xud(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                      null, XUD_SPEED_HS, XUD_PWR_SELF);

      on tile[1]: Endpoint0(c_ep_out[0], c_ep_in[0]);

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

      on tile[0]: mouse_input(i_mouse_input[0], i2c[0]);

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

    return 0;
}
```

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

The I2C device is setup in main() using an interface, this is declared as follows,

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

The interface is initialised in main via the following call,

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

The interface i2c is passed to the mouse input function in main() as follows,

```
on tile[0]: mouse_input(i_mouse_input[0], i2c[0]);
```

3.8 Setting up the mouse input interface in the application main()

The mouse input interface is setup in main() using an interface which is declared as follows,

```
interface mouse_input_if i_mouse_input[1];
```

This interface is passed to both the extended HID endpoint and the mouse input task.

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

on tile[0]: mouse_input(i_mouse_input[0], i2c[0]);
```

This allows them to communicate and the HID endpoint to request the mouse input data it requires.

3.9 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"
```

APPENDIX A - Demo Hardware Setup

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

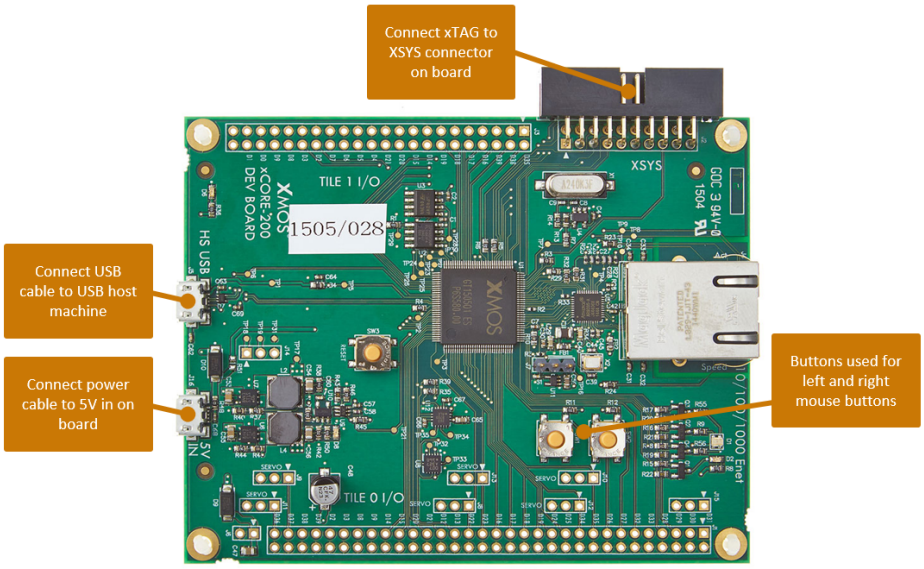

Figure 3: XMOS xCORE-200 explorerKIT

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

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

APPENDIX B - Launching the demo application

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

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

B.1 Launching from the command line

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

```
> xrun app_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.

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.

By tilting the board in the x and y axis you will be able to control the mouse pointer on your USB host machine.

The left and right mouse buttons are provided by the 2 buttons on the xCORE-200 explorer board.

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 main.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include "usb.h"
#include "i2c.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

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

/*
 * The main function runs three cores on tile[1] : 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. On tile[0] the interface to the I2C device and the
 * mouse button handler uses another core.
 */
int main()
{
    i2c_master_if i2c[1];
    interface mouse_input_if i_mouse_input[1];
    chan c_ep_out[XUD_EP_COUNT_OUT], c_ep_in[XUD_EP_COUNT_IN];

    par
    {
      on tile[1]: xud(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                      null, XUD_SPEED_HS, XUD_PWR_SELF);

      on tile[1]: Endpoint0(c_ep_out[0], c_ep_in[0]);

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

      on tile[0]: mouse_input(i_mouse_input[0], i2c[0]);

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

    return 0;
}
```

D.2 Source code for hid_mouse_extended.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <xs1.h>
#include "i2c.h"
#include "usb.h"
#include "hid_mouse_extended.h"

// Accelerometer code taken from AN00181

// FXOS8700EQ register address defines
#define FXOS8700EQ_I2C_ADDR 0x1E
#define FXOS8700EQ_XYZ_DATA_CFG_REG 0x0E
#define FXOS8700EQ_CTRL_REG_1 0x2A
#define FXOS8700EQ_DR_STATUS 0x0
#define FXOS8700EQ_OUT_X_MSB 0x1
#define FXOS8700EQ_OUT_X_LSB 0x2
#define FXOS8700EQ_OUT_Y_MSB 0x3
#define FXOS8700EQ_OUT_Y_LSB 0x4
```

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

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

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

    accel_val = data << 2;

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

    accel_val |= (data >> 6);

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

    return accel_val;
}

void get_accelerometer_data(client interface i2c_master_if i2c, int &x, int &y) {
   i2c_regop_res_t result;
   char status_data = 0;
   // Wait for valid accelerometer data
   do {
      status_data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_DR_STATUS, result);
   } while (!status_data & 0x08);

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

void init_accelerometer(client interface i2c_master_if i2c) {
    i2c_regop_res_t result;

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

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

// Button access
// PORT_4E connected to the 2 Buttons
on tile[0]: port p_button = XS1_PORT_4E;
#define DEBOUNCE_TIME   (XS1_TIMER_HZ/50)
#define BUTTON_PRESSED  0x00

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

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

// Mouse input interface, reads from accelerometer and buttons
void mouse_input(server interface mouse_input_if mi_if, client interface i2c_master_if i2c) {
    init_accelerometer(i2c);
    while (1) {
        select {
            case mi_if.get_input_data() -> struct mouse_input_data data:
            get_accelerometer_data(i2c, data.x, data.y);
            data.button_left = is_button_pressed(1);
            data.button_right = is_button_pressed(0);
            break;
        }
    }
}

// HID report buffer used by endpoint 0
extern unsigned char g_reportBuffer[4];

/*
 * This function responds to the HID requests
 */
void hid_mouse_extended(chanend chan_ep_hid, client interface mouse_input_if mi_if)
{
    XUD_ep ep_hid = XUD_InitEp(chan_ep_hid, XUD_EPTYPE_INT);

    while (1)
    {
      /* Unsafe region so we can use shared memory. */
      unsafe {
        char * unsafe p_reportBuffer = g_reportBuffer;
        int x, y;
        struct mouse_input_data mi_data;

        // Initialise the HID
        p_reportBuffer[0] = 0;  // Buttons
        p_reportBuffer[1] = 0;  // X axis
        p_reportBuffer[2] = 0;  // Y axis

        // Read from mouse input interface
        mi_data = mi_if.get_input_data();

        // Set up HID x axis value report
        if (mi_data.x > 10 || mi_data.x < -10) {
          p_reportBuffer[1] = mi_data.x >> 2;
        }

        // Set up HID y axis value report
        if (mi_data.y > 10 || mi_data.y < -10) {
          p_reportBuffer[2] = (mi_data.y * -1) >> 2;
        }

        // Set up HID button value report
        p_reportBuffer[0] |= mi_data.button_left;
        p_reportBuffer[0] |= mi_data.button_right << 1;

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

D.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_

#include <i2c.h>

struct mouse_input_data {
    int x;
    int y;
    int button_left;
    int button_right;
};

interface mouse_input_if {
    struct mouse_input_data get_input_data();
};

void hid_mouse_extended(chanend chan_ep_hid, client interface mouse_input_if mi_if);
void mouse_input(server interface mouse_input_if mi_if, client interface i2c_master_if i2c);

#endif /* HID_MOUSE_EXTENDED_H_ */
```