Application Note

AN_346

FT51A Mouse Sample

Version 1.2

Issue Date: 2015-12-21

This document provides a guide for using the FT51A development environment to emulate a HID Mouse device in firmware.

Future Technology Devices International Limited (FTDI)
Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777   Fax: + 44 (0) 141 429 2758
Web Site: http://ftdichip.com

Table of Contents

1 Introduction

This application note documents an example firmware project for the FT51A. The source code is available in the "examples\AN_346_Mouse_Sample" folder of the FT51A Software Development Kit.

1.1 Overview

The Mouse firmware project demonstrates a method for emulating a mouse using an FT51A device. The FT51A need only be connected to a host PC via the USB interface. The demonstration code sends mouse movements to the host PC. The host PC treats these as real mouse movements and will move the cursor accordingly.

An example application would be to embed an FT51A in a device that sends HID (Human Interface Device) data from a sensor to a host PC. The PC would receive this data as mouse movements. For example, a force sensor could be attached allowing cursor up or page up key presses to be sent when a sensor is pushed.

The example code also includes the DFU functionality from AN_344 FT51A DFU Sample.

1.2 Features

The mouse example has the following features:

- Open source firmware layered on FT51A USB Library.
- Configurable HID report descriptor.
- HID report structure can be modified as required.
- Implements an interrupt IN endpoint using timers.

1.3 Limitations

The firmware emulates a HID "boot protocol" device which uses a predefined report format. The report descriptor has little room for adding additional functionality without switching to "report protocol".

1.4 Scope

The guide is intended for developers who are creating applications, extending FTDI provided applications or implementing example applications for the FT51A.

In the reference of the FT51A, an "application" refers to firmware that runs on the FT51A; "libraries" are source code provided by FTDI to help user, access specific hardware features of the chip.

The FT51A Tools are currently only available for Microsoft Windows platforms and are tested on Windows 7 and Windows 8.1.

2 HID Mouse Overview

The HID protocol and requirements are documented in the HID 1.1 specification:

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

The FT51A implementation is for a USB device which emulates a "Boot Protocol" mouse. This is a special class of HID device which has a restricted set of features in order to reduce the complexity of the host. This allows a HID device to be used with simpler hosts without requiring the host to decode report descriptors. The format of the report descriptor is therefore fixed.

Configuration descriptors for HID devices will have an interface descriptor containing a HID descriptor along with at least one endpoint descriptor. This firmware has a single interrupt IN endpoint which returns report packets to the host.

2.1 Firmware Overview

The firmware will enumerate as a HID mouse to the host. After a short delay it will send mouse button presses and movements to simulate the actions of a real mouse. The simulated actions can be modified if needed.

2.1.1 FT51A Libraries

The mouse firmware uses the FT51A USB library, DFU library, general config library and the IOMUX library. The IOMUX library is not used in the example code but is included to allow further functionality to be added.

The firmware is designed for the FT51A EVM module and uses the LCD for displaying some information. The I2C Master library is used for communication with the LCD.

DFU functionality is implemented as described in AN_344 DFU Sample.

3 Mouse Firmware

The firmware included in the example code demonstrates a boot protocol mouse.

A report descriptor is included for a standard boot protocol mouse. HID devices, especially boot protocol devices, do not normally require additional drivers on modern operating systems. The USB class, subclass and protocol must therefore be set to identify the device as the correct HID compatible mouse providing boot protocol reports.

The firmware is designed for the FT51A EVM module. It utilises the LCD for user feedback with communications on the I^2C Master bus; the force sensor; and two buttons to control the mouse pointer.

Pressing SW1 and SW2 together will switch between moving the cursor horizontally or vertically. Pressing them individually will move left or right (in horizontal mode) or up and down (in vertical mode). Pressing the force sensor above a threshold will simulate a press of mouse button 1 (MB1).

The I^2C Master SCL is on pin DIO_13 and SDA is on pin DIO_14. There is a reset line to the LCD on pin DIO_8. This is mapped to bit 0 of the P0 I/O Port.

SW1 is on pin DIO_0 and is mapped to bit 1 of P0, SW2 is on pin DIO_7 and mapped to bit 2.

3.1 USB Descriptors

The mouse firmware stores two sets of device descriptors and configuration descriptors. It stores a single table of string descriptors as the strings for run time and DFU modes can be selected by the descriptors as needed from the same table.

The control endpoint max packet size is defined as 8 bytes. The interrupt IN endpoint for reporting is also set to a maximum of 8 bytes. Both settings are in line with other HID mouse devices.

```
// USB Endpoint Zero packet size (both must match)
#define USB_CONTROL_EP_MAX_PACKET_SIZE 8
#define USB_CONTROL_EP_SIZE USB_EP_SIZE_8
```

The Product IDs (PIDs) for run time (0x0FEA) and DFU mode (0x0FEE – the same as the AN_344 DFU Sample interface) are also defined in the source code. Note AN_345 FT51A Keyboard Sample uses the same PID number, so driver uninstall is required if both examples are used.

These are example PID values and **must not** be used in a final product. VID and PID combinations must be unique to an application.

The USB class, subclass and protocols along with other general USB definitions are found in the file "FT51A_usb.h" library include file.

3.1.1 Mouse Descriptors

The first set of descriptors is the run time set for the mouse function. The device descriptor contains the VID and PID for the mouse.

```
__code USB_device_descriptor device_descriptor_mouse =
{
    .bLength = 0x12,
    .bDescriptorType = USB_DESCRIPTOR_TYPE_DEVICE,
    .bcdUSB = USB_BCD_VERSION_2_0,          // V2.0
    .bDeviceClass = USB_CLASS_DEVICE,       // Defined in interface
    .bDeviceSubClass = USB_SUBCLASS_DEVICE, // Defined in interface
    .bDeviceProtocol = USB_PROTOCOL_DEVICE, // Defined in interface
    .bMaxPacketSize0 = USB_CONTROL_EP_MAX_PACKET_SIZE,
    .idVendor = USB_VID_FTDI,    // idVendor: 0x0403 (FTDI)
```

```
    .idProduct = USB_PID_MOUSE, // idProduct: 0x0fea
    .bcdDevice = 0x0101,        // 1.1
    .iManufacturer = 0x01,  // String 1
    .iProduct = 0x02,       // String 2
    .iSerialNumber = 0x03,  // String 3
    .bNumConfigurations = 0x01
};
```

The configuration descriptor contains an interface descriptor, a HID descriptor and an endpoint descriptor for the run time function.

It also has an interface descriptor for a DFU interface. This includes a DFU functional descriptor.

```
// Structure containing layout of configuration descriptor
__code struct config_descriptor_mouse
{
        USB_configuration_descriptor configuration;
        USB_interface_descriptor interface;
        USB_hid_descriptor hid;
        USB_endpoint_descriptor endpoint;
        USB_interface_descriptor dfu_interface;
        USB_dfu_functional_descriptor dfu_functional;
};

struct config_descriptor_mouse config_descriptor_mouse =
{
        .configuration.bLength = 0x09,
        .configuration.bDescriptorType = USB_DESCRIPTOR_TYPE_CONFIGURATION,
        .configuration.wTotalLength = sizeof(struct config_descriptor_mouse),
        .configuration.bNumInterfaces = 0x02,
        .configuration.bConfigurationValue = 0x01,
        .configuration.iConfiguration = 0x00,
        .configuration.bmAttributes =  USB_CONFIG_BMATTRIBUTES_VALUE,
        .configuration.bMaxPower = 0xFA, // 500mA

        // ---- INTERFACE DESCRIPTOR for Mouse ----
        .interface.bLength = 0x09,
        .interface.bDescriptorType = USB_DESCRIPTOR_TYPE_INTERFACE,
        .interface.bInterfaceNumber = 0,
        .interface.bAlternateSetting = 0x00,
        .interface.bNumEndpoints = 0x01,
        .interface.bInterfaceClass = USB_CLASS_HID,             // HID Class
        .interface.bInterfaceSubClass = USB_SUBCLASS_HID_BOOT_INTERFACE, // Boot Protocol
        .interface.bInterfaceProtocol = USB_PROTOCOL_HID_MOUSE,   // Mouse
        .interface.iInterface = 0x02, // String 2
        // ---- HID DESCRIPTOR for Mouse ----
        .hid.bLength = 0x09,
        .hid.bDescriptorType = USB_DESCRIPTOR_TYPE_HID,
        .hid.bcdHID = USB_BCD_VERSION_HID_1_1,
        .hid.bCountryCode = USB_HID_LANG_NOT_SUPPORTED,
        .hid.bNumDescriptors = 0x01,
        .hid.bDescriptorType_0 = USB_DESCRIPTOR_TYPE_REPORT,
        .hid.wDescriptorLength_0 = 0x0034,
        // ---- ENDPOINT DESCRIPTOR for Mouse ----
        .endpoint.bLength = 0x07,
        .endpoint.bDescriptorType = USB_DESCRIPTOR_TYPE_ENDPOINT,
        .endpoint.bEndpointAddress = 0x81,
        .endpoint.bmAttributes = USB_ENDPOINT_DESCRIPTOR_ATTR_INTERRUPT,
        .endpoint.wMaxPacketSize = 0x0008,
        .endpoint.bInterval = 0x0a,

        // ---- INTERFACE DESCRIPTOR for DFU Interface ----
```

```
        .dfu_interface.bLength = 0x09,
        .dfu_interface.bDescriptorType = USB_DESCRIPTOR_TYPE_INTERFACE,
        .dfu_interface.bInterfaceNumber = DFU_USB_INTERFACE_RUNTIME,
        .dfu_interface.bAlternateSetting = 0x00,
        .dfu_interface.bNumEndpoints = 0x00,
        .dfu_interface.bInterfaceClass = USB_CLASS_APPLICATION, // Application Specific
        .dfu_interface.bInterfaceSubClass = USB_SUBCLASS_DFU,   // Device Firmware Update
        .dfu_interface.bInterfaceProtocol = USB_PROTOCOL_DFU_RUNTIME, // Runtime Protocol
        .dfu_interface.iInterface = 0x05, // String 5
        // ---- FUNCTIONAL DESCRIPTOR for DFU Interface ----
        .dfu_functional.bLength = 0x09,
        .dfu_functional.bDescriptorType = USB_DESCRIPTOR_TYPE_DFU_FUNCTIONAL,
        .dfu_functional.bmAttributes = USB_DFU_BMATTRIBUTES_CANDNLOAD |
                USB_DFU_BMATTRIBUTES_WILLDETACH, // bitCanDnload and bitWillDetach
        .dfu_functional.wDetatchTimeOut = DFU_TIMEOUT, // suggest 8192ms
        .dfu_functional.wTransferSize = DFU_BLOCK_SIZE,// typically 64 bytes
        .dfu_functional.bcdDfuVersion = USB_BCD_VERSION_DFU_1_1, // DFU Version 1.1
};
```

The HID descriptor tells the host that there is a report descriptor of length 0x34 bytes to read which will describe the report format of the mouse.

This descriptor will be read with a GET_DESCRIPTOR request for a report descriptor.

```
/**
See Device Class Definition for Human Interface Devices (HID) Version 1.11
 from USB Implementers' Forum USB.org
    0x05, 0x01,        Usage Page: Generic Desktop Controls
    0x09, 0x02,        Usage: Mouse (2)
    0xA1, 0x01,        Collection: Application
    0x09, 0x01,          Usage: Pointer (1)
    0xA1, 0x00,          Collection: Physical
    0x05, 0x09,            Usage Page: Button (9)
    0x19, 0x01,            Usage Minimum: Button 1
    0x29, 0x03,            Usage Maximum: Button 3
    0x15, 0x00,            Logical Minimum: 0
    0x25, 0x01,            Logical Maximum: 1
    0x95, 0x03,            Report Count: 3
    0x75, 0x01,            Report Size: 1
    0x81, 0x02,            Input: Data (2)
    0x95, 0x01,            Report Count: 1
    0x75, 0x05,            Report Size: 5
    0x81, 0x01,            Input: Constant (1)
    0x05, 0x01,            Usage Page: Generic Desktop Controls
    0x09, 0x30,            Usage: X
    0x09, 0x31,            Usage: Y
    0x09, 0x38,            Usage: Wheel
    0x15, 0x81,            Logical Minimum: -127
    0x25, 0x7f,            Logical Maximum: 127
    0x75, 0x08,            Report Size: 8
    0x95, 0x03,            Report Count: 3
    0x81, 0x06,            Input: Data (6)
    0xC0,                End collection
    0xC0               End collection
**/
//@{
__code uint8_t hid_report_descriptor_mouse[52] =
{
    0x05, 0x01, 0x09, 0x02, 0xA1, 0x01, 0x09, 0x01, 0xA1, 0x00,
    0x05, 0x09, 0x19, 0x01, 0x29, 0x03, 0x15, 0x00, 0x25, 0x01,
    0x95, 0x03, 0x75, 0x01, 0x81, 0x02, 0x95, 0x01, 0x75, 0x05,
    0x81, 0x01, 0x05, 0x01, 0x09, 0x30, 0x09, 0x31, 0x09, 0x38,
```

```
    0x15, 0x81, 0x25, 0x7f, 0x75, 0x08, 0x95, 0x03, 0x81, 0x06,
    0xC0, 0xC0
};
```

The format of the report is matched to a C structure containing bitmaps and byte fields described in the report descriptor.

```
typedef struct hid_mouse_report_structure_t
{
    unsigned char mouseB1   : 1;
    unsigned char mouseB2   : 1;
    unsigned char mouseB3   : 1;
    unsigned char mouseResv : 5;
    unsigned char mouseX;
    unsigned char mouseY;
    unsigned char mouseWheel;
} hid_mouse_report_structure_t;
```

Filling in the members of this structure will produce a report for the host with the desired data.

3.1.2 DFU Descriptors

The device descriptor contains the VID and PID for the DFU function. This may or may not be the same as the run time VID and PID.

```
USB_device_descriptor device_descriptor_dfumode =
{
        .bLength = 0x12,
        .bDescriptorType = USB_DESCRIPTOR_TYPE_DEVICE,
        .bcdUSB = USB_BCD_VERSION_2_0,
        .bDeviceClass = USB_CLASS_DEVICE,
        .bDeviceSubClass = USB_SUBCLASS_DEVICE,
        .bDeviceProtocol = USB_PROTOCOL_DEVICE,
        .bMaxPacketSize0 = USB_CONTROL_EP_MAX_PACKET_SIZE,
        .idVendor = USB_VID_FTDI,    // 0x0403 (FTDI)
        .idProduct = DFU_USB_PID_DFUMODE, // 0x0fee
        .bcdDevice = 0x0101,
        .iManufacturer = 0x01,
        .iProduct = 0x04,
        .iSerialNumber = 0x03,
        .bNumConfigurations = 0x01
};
```

The configuration descriptor for DFU will contain only an interface descriptor and a functional descriptor for the DFU interface.

The USB class, subclass and protocol indicate that this device is now in DFU mode.

```
// Structure containing layout of configuration descriptor
struct config_descriptor_dfumode
{
    USB_configuration_descriptor configuration;
    USB_interface_descriptor interface;
    USB_dfu_functional_descriptor functional;
};

struct config_descriptor_dfumode config_descriptor_dfumode =
{
        .configuration.bLength = 0x09,
        .configuration.bDescriptorType = USB_DESCRIPTOR_TYPE_CONFIGURATION,
        .configuration.wTotalLength = sizeof(struct config_descriptor_dfumode),
```

```
        .configuration.bNumInterfaces = 0x01,
        .configuration.bConfigurationValue = 0x01,
        .configuration.iConfiguration = 0x00,
        .configuration.bmAttributes = USB_CONFIG_BMATTRIBUTES_VALUE,
        .configuration.bMaxPower = 0xFA, // 500 mA

              // ---- INTERFACE DESCRIPTOR for DFU Interface ----
        .dfu_interface.bLength = 0x09,
        .dfu_interface.bDescriptorType = USB_DESCRIPTOR_TYPE_INTERFACE,
        .dfu_interface.bInterfaceNumber = DFU_USB_INTERFACE_DFUMODE,
        .dfu_interface.bAlternateSetting = 0x00,
        .dfu_interface.bNumEndpoints = 0x00,
        .dfu_interface.bInterfaceClass = USB_CLASS_APPLICATION, // Application Specific
        .dfu_interface.bInterfaceSubClass = USB_SUBCLASS_DFU,   // Device Firmware Update
        .dfu_interface.bInterfaceProtocol = USB_PROTOCOL_DFU_DFUMODE, // Runtime Protocol
        .dfu_interface.iInterface = 0x05, // String 5

        // ---- FUNCTIONAL DESCRIPTOR for DFU Interface ----
        .dfu_functional.bLength = 0x09,
        .dfu_functional.bDescriptorType = USB_DESCRIPTOR_TYPE_DFU_FUNCTIONAL,
        .dfu_functional.bmAttributes = USB_DFU_BMATTRIBUTES_CANDNLOAD |
                USB_DFU_BMATTRIBUTES_WILLDETACH, // bitCanDnload and bitWillDetach
        .dfu_functional.wDetatchTimeOut = DFU_TIMEOUT, // suggest 8192ms
        .dfu_functional.wTransferSize = DFU_BLOCK_SIZE,// typically 64 bytes
        .dfu_functional.bcdDfuVersion = USB_BCD_VERSION_DFU_1_1,
};
```

The same bmAttributes mask must appear for the DFU functional descriptor in both run time and DFU modes.

3.1.3 Descriptor Selection

The standard request handler for GET_DESCRIPTOR requests needs to select the run time or DFU mode descriptors for device and configuration descriptors. Other descriptors, including the report descriptors and string descriptors, are not affected.

Determining if the firmware is in run time or DFU mode is achieved by calling the dfu_is_runtime() function from the DFU library.

A non-zero response will select the run time mode descriptors and a zero response, the DFU mode descriptors.

```
FT51A_STATUS standard_req_get_descriptor(USB_device_request *req)
{
    uint8_t   *src = NULL;
    uint16_t   length = req->wLength;
    uint8_t    hValue = req->wValue >> 8;
    uint8_t    lValue = req->wValue & 0x00ff;
    uint8_t    i, slen;

    switch (hValue)
    {
    case USB_DESCRIPTOR_TYPE_DEVICE:

        if (dfu_is_runtime())
        {
            src = (char *)&device_descriptor_runtime;
        }
        else
        {
            src = (char *)&device_descriptor_dfumode;
```

```
        }
        if (length > sizeof(USB_device_descriptor)) // too many bytes requested
            length = sizeof(USB_device_descriptor); // Entire structure.
        break;

    case USB_DESCRIPTOR_TYPE_CONFIGURATION:

        if (dfu_is_runtime())
        {
            src = (char *)&config_descriptor_runtime;
            if (length > sizeof(config_descriptor_runtime)) // too many bytes requested
                length = sizeof(config_descriptor_runtime); // Entire structure.
        }
        else
        {
            src = (char *)&config_descriptor_dfumode;
            if (length > sizeof(config_descriptor_dfumode)) // too many bytes requested
                length = sizeof(config_descriptor_dfumode); // Entire structure.
        }
        break;

    case USB_DESCRIPTOR_TYPE_REPORT:

        src = (char *) &hid_report_descriptor_mouse;
        if (length > sizeof(hid_report_descriptor_mouse))
            length = sizeof(hid_report_descriptor_mouse); // Entire structure.
        break;
```

The FT51A USB library will return the structure pointed to by the standard_req_get_descriptor() function.

Note that string descriptor selection is not shown in this code sample.

3.2 USB Class Requests

The firmware is responsible for handling USB class requests. It must determine if the firmware is in run time or DFU mode and whether a request has been directed to the DFU interface. This must not interfere with other class requests that may be decoded in the firmware.

The first check is that the class request is aimed at an interface:

```
FT51A_STATUS class_req_cb(USB_device_request *req)
{
    FT51A_STATUS         status = FT51A_FAILED;
    uint8_t             interface = LSB(req->wIndex) & 0x0F;

    // For DFU requests ensure the recipient is an interface...
    if ((req->bmRequestType & USB_BMREQUESTTYPE_RECIPIENT_MASK) ==
            USB_BMREQUESTTYPE_RECIPIENT_INTERFACE)
    {
```

If this is correct then the firmware must check if it is in run time or DFU mode before checking the interface number. The interface number for the DFU mode may differ from that of the run time mode.

Requests to the DFU interface are passed to the DFU library but others are handled as HID requests.

```
        if (dfu_is_runtime())
```

```
        {
            if ((interface == DFU_USB_INTERFACE_RUNTIME))
            {
                // Handle DFU requests DFU_DETATCH, DFU_GETSTATE and DFU_GETSTATUS
                // when in run time mode.
                switch (req->bRequest)
                {
                case USB_CLASS_REQUEST_DETACH:
                    dfu_class_req_detach(req->wValue);
                    status = FT51A_OK;
                    break;
                case USB_CLASS_REQUEST_GETSTATUS:
                    dfu_class_req_getstatus();
                    status = FT51A_OK;
                    break;
                case USB_CLASS_REQUEST_GETSTATE:
                    dfu_class_req_getstate();
                    status = FT51A_OK;
                    break;
                }
            }
            else
            {
                // Handle HID class requests in run time mode.
                switch (req->bRequest)
                {
                case USB_HID_REQUEST_CODE_GET_REPORT:
                    USB_transfer(1, USB_DIR_IN, (char *) &report_buffer,
                        sizeof(report_buffer));
                    status = FT51A_OK;
                    break;
                case USB_HID_REQUEST_CODE_SET_PROTOCOL:
                    USB_transfer(USB_EP_0, USB_DIR_IN, NULL, 0);
                    report_protocol = req->wValue & 0xff;
                    status = FT51A_OK;
                    break;
                case USB_HID_REQUEST_CODE_GET_PROTOCOL:
                    USB_transfer(USB_EP_0, USB_DIR_IN, &report_protocol, 1);
                    status = FT51A_OK;
                    break;
                }
            }
        }
```

When the device is in DFU mode the DFU request handlers are called as described in AN 344 DFU Sample.

A boot protocol mouse must support GET_REPORT, GET_PROTOCOL and SET_PROTOCOL requests.

The GET_REPORT request will allow a report to be sent from the firmware to the host without using the interrupt IN endpoint.

The SET_PROTOCOL request is used to switch between boot and report modes of operation. It is required by the specification but is not typically used by standard device drivers. A GET_PROTOCOL will return 0 for boot protocol and 1 for report protocol. It is implemented to allow the firmware to be extended to support report protocol.

3.3 USB Reset Handler

The reset function handler is used to make the transition from run time mode to DFU mode.

3.4 Timer

A timer is used to simulate mouse movement and to await a polling interval before transmitting on the interrupt IN endpoint. A read of the 2 buttons (SW1 and SW2) on the FT51A EVM board is also made. This is used in the main loop to calculate mouse movement.

The DFU also needs a millisecond timer to accurately return to the appIDLE state from the appDETACH state. The dfu_timer() function in the DFU library should be called every millisecond to enable this.

```
void detach_interrupt(const uint8_t flags)
{
    (void)flags; // Flags not currently used

    // decrement millisecond timers ...
    ms_timer--;
    mouse_timer--;
    // make note of current button states
    // 0x00 = no keys, 0x01 = SW1 only (left), 0x02 = SW2 only (right), 0x03 = both keys
    mouse_button = ((~P0) >> 1) & 0x03;
    // The DFU detach timer must be called once per millisecond
    dfu_timer();
    // Reload the timer
    TH0 = MSB(MICROSECONDS_TO_TIMER_TICKS(1000));
    TL0 = LSB(MICROSECONDS_TO_TIMER_TICKS(1000));
}

void detach_timer_initialise(void)
{
    // Register our own handler for interrupts from Timer 0
    interrupts_register(detach_interrupt, interrupts_timer0);

    // Timer0 is controlled by TMOD bits 0 to 3, and TCON bits 4 to 5.
    TMOD &= 0xF0; // Clear Timer0 bits
    TMOD |= 0x01; // Put Timer0 in mode 1 (16 bit)
    // Set the count-up value so that it rolls over to 0 after 1 millisecond.
    TH0 = MSB(MICROSECONDS_TO_TIMER_TICKS(1000));
    TL0 = LSB(MICROSECONDS_TO_TIMER_TICKS(1000));
    TCON &= 0xCF; // Clear Timer0's Overflow and Run flags
    TCON |= 0x10; // Start Timer0 (set its Run flag)
}
```

3.5 Mouse Report Generator

A timer is used to make a delay between sending reports and hence give the effect of a mouse moving. The two buttons on the FT51A EVM are read in the timer routine and the main code parses this information to control the mouse pointer.

If both buttons are pressed simultaneously then the action of the buttons pressed individually changes between moving horizontally and vertically. When pressed individually a report is generated for moving the cursor. The longer the button is pressed the faster the mouse will move.

A conversion of the analogue input of the force sensor to a digital reading is initiated and the result used to decide if mouse button 1 should be pressed in the report.

Information about the mouse speed, the reading from the force sensor and the direction of movement are shown on the LCD.

Once the report is ready to send then it is transmitted with an `USB_transfer()` call to the interrupt IN endpoint. The host will read from the device periodically in line with the setting of `bInterval` in the endpoint descriptor for the interrupt IN endpoint. It is recommended to continue polling for SETUP tokens while waiting for a transmission to complete. The `USB_transfer()` function will not block if the transmit buffer is empty.

4 Possible Improvements

The current implementation is "boot protocol". This has limited scope for improvements as the functionality of this type of mouse is fixed.

If the sample were to change to "report protocol" then more flexibility would be gained and more devices could be emulated.

Another use of the force sensor would be to make an analogue joystick. Since there is one sensor on the FT51A EVM it would only control one direction but there are up-to 8 analogue inputs on the FT51A devices which could be configured to make two force sensing 2-axis joysticks without the need for potentiometers.

5 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com

Branch Office – Tigard, Oregon, USA

Future Technology Devices International Limited (USA)
7130 SW Fir Loop
Tigard, OR 97223-8160
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.support@ftdichip.com
E-Mail (General Enquiries) us.admin@ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 1103, No. 666 West Huaihai Road,
Shanghai, 200052
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales) cn.sales@ftdichip.com
E-mail (Support) cn.support@ftdichip.com
E-mail (General Enquiries) cn.admin@ftdichip.com

Web Site

http://ftdichip.com

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A – References

Document References

FTDI MCU web page: http://www.ftdichip.com/MCU.html

USB HID 1.1 specifications: http://www.usb.org/developers/hidpage/HID1_11.pdf

USB Device Firmware Update Class specification:
http://www.usb.org/developers/docs/devclass_docs/DFU_1.1.pdf

Acronyms and Abbreviations

Terms	Description
HID	Human Interface Device
MTP	Multiple Time Program – non-volatile memory used to store program code on the FT51A.
USB	Universal Serial Bus
USB-IF	USB Implementers Forum

Appendix B – Revision History

Document Title: AN_346 FT51A Mouse Sample

Document Reference No.: FT_001124

Clearance No.: FTDI# 433

Product Page: http://www.ftdichip.com/FTProducts.htm

Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	2014-12-12
1.1	Update FT51 references to FT51A	2015-11-26
1.2	Update to DFU	2015-12-21