How to use it

To test this mouse tutorial, a minimal configuration is required:

- Use the Jumpers:
  - PWR_J - depend of voltage source
  - VREFF - present
  - DBG - present

- run program
- use the USB cable to connect the PC to USB of the Board.

Mouse motion events are emulated with the following:

Butt1 - Left movement of the mouse marker
Butt2 - Right movement of the mouse marker
Trimer Clock-wise - Up movement of the mouse marker
Trimer Counter-clock-wise - Down movement of the mouse marker

The first time the device is connected to the computer, Windows will load the driver for identified device. The USB Human Interface Device driver will be automatically loaded.

Software implementation

1 *Device enumeration and configuration* **-** When a USB device is attached, the host issues a reset signal. When the reset signal is released, the device enters the enumerated state.

1.1 USB Reset - When a USB reset signal is detected on the bus, the DEV_STAT bit in the DEVINTS register is set and a USB interrupt is generated.

1.2 Enumeration **-** The host performs a bus enumeration to identify the attached device and to assign a unique address to it. The device responds to the requests sent by the host during the enumeration process on its default pipe (endpoint 0).

Enumeration steps:

a. Get Device descriptor - The host sends a get device descriptor request. The device replies with its device descriptor to report its attributes (Device Class, maximum packet size for endpoint zero).

b. Set address -A USB device uses the default address after reset until the host assigns a unique address using the set address request. The firmware writes the device address assigned by the host by SET_ADDRESS command (0xD0) user must be set bit7 to enable embedded function of USB engine.

c. Get configuration - The host sends a get configuration. The device replies with its configuration descriptor, interface descriptor and endpoint descriptor. The configuration descriptor describes the number of interfaces provided by the configuration, the power source (Bus or Self powered) and the maximum power consumption of the USB device from the bus. The Interface descriptor describes the number of endpoints used by this interface. The Endpoint descriptor describes the transfer type supported and the bandwidth requirements.

d. Set Configuration - The host assigns a configuration value to the device based on the configuration information. The device is then in configured state and can draw the amount of power described in the configuration descriptor. The device is now configured and ready to be used.

For more information, see also the USB specification, chapter 9, “USB Device Framework”.

2. USB Mouse descriptor - USB protocol can configure devices at start-up or when they are plugged-in at run time. These devices are divided into various device classes. Each device class defines the common behavior and protocols for devices that serve similar functions.

2.1 Descriptor Structure - The HID class consists primarily of devices that are used by humans to control the operation of computer systems. Mice, like all pointing devices, are typical examples of HID class devices. These segments are called Descriptors and are divided into several types: Device Descriptor, Configuration Descriptor, Interface Descriptor, HID Descriptor, Endpoint Descriptor, String Descriptor and Report Descriptor. All Descriptors are mandatory in LPC_HID.C.

2.2 Device Descriptors **-** At the top level, a descriptor includes two tables of information referred to as the Device Descriptor and the String Descriptor. A standard USB Device Descriptor specifies the Product ID and other information’s about the device. For example, Device Descriptor fields primarily include: Class, Subclass, Vendor, Product, Version. The following code corresponds to USB Mouse Descriptors applied to a two-buttons, two- axis opto-mechanical mouse.

```
const char devDescriptor[] =
{
  /* Device descriptor */
  0x12,               // bLength
  0x01,               // bDescriptorType
  0x10,               // bcdUSBL
  0x01,               //
  0x00,               // bDeviceClass:
  0x00,               // bDeviceSubclass:
  0x00,               // bDeviceProtocol:
  MAX_CTRL_EP_PK_SIZE,  // bMaxPacketSize0
  0xFF,               // idVendorL
  0xFF,               //
  0x01,               // idProductL
  0x00,               //
  0x00,               // bcdDeviceL
  0x00,               //
  0x01,               // iManufacturer   // Index of string descriptor describing manufacturer
  0x02,               // iProduct        // Index of string descriptor describing produt
  0x00,               // SerialNumber
  0x01                // bNumConfigs
};
```

2.3 Configuration Descriptor - This Descriptor divided into several segments includes Interface Descriptor, HID descriptor and Endpoint Descriptor:

```
const char cfgDescriptor[] =
{
  /* ============= CONFIGURATION 1 ========== */
  /* Configuration 1 descriptor */
  0x09,   // CbLength
  0x02,   // CbDescriptorType
  0x22,   // CwTotalLength 2 EP + Control
  0x00,
  0x01,   // CbNumInterfaces
```

```
 0x01,   // CbConfigurationValue
 0x00,   // CiConfiguration
 0xA0,   // CbmAttributes Bus powered + Remote Wakeup
 0x32,   // CMaxPower: 100mA

 /* Mouse Interface Descriptor Requirement */
 0x09,   // bLength
 0x04,   // bDescriptorType
 0x00,   // bInterfaceNumber
 0x00,   // bAlternateSetting
 0x01,   // bNumEndpoints
 0x03,   // bInterfaceClass: HID code
 0x01,   // bInterfaceSubclass
 0x02,   // bInterfaceProtocol: Mouse
 0x00,   // iInterface

 /* HID Descriptor */
 0x09,   // bLength
 0x21,   // bDescriptor type: HID Descriptor Type
 0x00,   // bcdHID
 0x01,
 0x00,   // bCountry Code
 0x01,   // bNumDescriptors
 0x22,   // bDescriptorType
 sizeof(mouseDescriptor), // wItemLength
 0x00,

 /* Endpoint 1 descriptor */
 0x07,        // bLength
 0x05,        // bDescriptorType
 ((EP_REP&1)<<7) + (EP_REP>>1),// bEndpointAddress and direction, Endpoint Logic address!!
 0x03,        // bmAttributes      INT
 0x04,        // wMaxPacketSize: 3 bytes (button, x, y)
 0x00,
 0x0A,         // polling bInterval
};
```

2.4 Report Descriptor **-** The Report Descriptor is different from the other descriptors in that it is not simply a table of values. It is made up of items that provide information about the data provided by each control in a device. Input items are used to tell the host what type of data will be returned as input to the host for interpretation, whether the data is absolute or relative and other pertinent information. By looking at a Report Descriptor alone, an application knows how to handle incoming data, as well as what the data could be used for. The following descriptor describes a two-buttons, two-axis USB Mouse.

2.5 String Descriptor - The previous descriptors can contain references to string Descriptors that provide displayable
information describing a descriptor in human-readable form. The inclusion of string Descriptors is optional. String Descriptors use UNICODE encoding. In the following String Descriptor example, all fields can be modified to enter your own manufacturer index, product index and serial number index.

```
const char LanguagesStr[] =
{
 /* String descriptor 0*/
 0x04, // bLength
```

```
  0x03, // bDescriptorType
  0x09,0x04 // Language English
};
const char ManufacturerStr[] =
{
  /* String descriptor 1*/
  60,   // bLength
  0x03, // bDescriptorType
  'P',0,'h',0,'i',0,'l',0,'i',0,'p',0,' ',0,'S',0,'e',0,'m',0,'i',0,'c',0,'o',0,'n',0,'d',0, 'u',0,'c',0,'t',0,'o',0,'r',0,'s',0,'
',0,'L',0,'P',0,'C',0,'2',0,'1',0,'4',0,'8',0,
};
const char ProductStr[] =
{
  /* String descriptor 2*/
  98,   // bLength
  0x03, // bDescriptorType
  'I',0,'A',0,'R',0,' ',0,'E',0,'m',0,'b',0,'e',0,'d',0,'d',0,'e',0,'d',0,' ',0,'W',0,'o',0,'r',0,'k',0,'b',0,'e',0,'n',0,'c',0,'h',0,'
',0,'A',0,'R',0,'M',0,' ',0,'-',0,' ',0,'H',0,'I',0,'D',0,' ',0,'D',0,'e',0,'v',0,'i',0,'c',0,'e',0,'
',0,'e',0,'x',0,'a',0,'m',0,'p',0,'l',0,'e',0,'!',0
};
```

3 Data transfer - The USB Mouse should be able to receive data on endpoint 0 and send data through endpoint 0 and endpoint 1. All the decoding / encoding operations on the USB frames are handled by firmware. The firmware must determine the endpoint number which has sent or received data by reading the ENDPINTS register.. The transfer types supported by this application are:

3.1 Control transfer with endpoint 0 (SETUP and IN (Physic Endpoint 1) and OUT (Physic Endpoint 0) tokens) - All control transfers are supported by endpoint 0. There can be control transfers with data phase and control transfers without data phase. As a consequence, a control transfer may have three transaction stages: a Setup stage, a Data stage (not for no-data control transfer) and a Status stage.

3.2. Interrupt transfer with endpoint 1 (Physic Endpoint 3) IN token - After the enumeration phase, the host continuously issues IN tokens through the endpoint 1 interrupt pipe. As the mouse has some data to return to the host, it returns three bytes of data (for a 2-axis mouse). These bytes are in format used for the boot report format for USB Mouse so that the data can be correctly interpreted by the BIOS.

When the mouse has some data to return to the host through the interrupt pipe, it must write this data in the Transmute buffer and enable endpoint 1 in transmission by setting the Buffer Valid command (0xFA). The host polls endpoint 1 with a polling interval given in the endpoint descriptor by sending an IN token. The hardware interface replies with STALL, NAK or data.

4. Mouse handling routines - According to the LPC2146 microcontroller, all USB events are managed by interrupt. When an USB event occurs, a flag of the Device Interrupt State Register (DEVINTS) is set by hardware. Then, the firmware determines the interrupt origin by reading the DEVINTS register, and clears the interrupt flag. The HID_CallBack() routine reads the software register (USB_IntrStaus) to determine the USB interrupt source and jumps to the corresponding interrupt routine.

The Mouse handling routine is divided into two parts executed by the microcontroller.

A first part is executing in forward and implements control endpoint management.

The second part is executing in background and implements the buttons and data transitions to host by the endpoint 1.