Hands-On Workshop: **Implementing USB** Features and Functions Based on Kinetis KL26 MCU

APF-IND-T0622

Neo Xiong | Field Applications Engineer

**J U L Y . 2 0 1 4**
Agenda

• USB basic and Kinetis L USB Module introduce
• FSL USB Stack Overview and Architecture
• USB stack HID Class files and Data flows introduce
• USB stack HID Driver API usage and Resource usage
• HID Class joystick demo introduce and hands-on
• Q&A
Kinetis L USB Module Introduce and USB Basic
Kinetis USB Module Block Diagram

- Kinetis USB system is an integration of three different modules:
  - On chip voltage 3.3V regulator (input supply varying from 2.7 V to 5.5 V);
  - Full-speed USB 2.0 dual-role controller(Host / device);
  - USB FS/LS transceiver (includes internal 15 kΩ pull downs);
Kinetis L series USB Features

• USB features
  – Full-speed USB 2.0 compliant
  – Dual-role operation (Host/Device)
  – 16 double buffered bidirectional endpoints
  – On chip USB full-speed PHY
  – Low power operation
  – Support On-The-Go protocol

• Highlights
  – Output pin (Vreg33) from regulator can be used to power external board components and source up to 120mA, can also power MCU main power supply. Eliminate a external LDO in space/cost sensitive application.

Note: Not all L series chip support USB. (available on KL2x, 4x)
On chip voltage regulator

- On chip voltage regulator features
  - 3.3 voltage output
  - 120mA output current
  - Low drop-out voltage: 300 mV
  - Run, Standby and shutdown operation modes
  - 120uA quiescent current in run mode
  - 1uA quiescent current in standby mode

- Module dependencies
  - USB voltage regulator works completely independent of the MCU, and is enabled even if the MCU is not powered.
  - Must be enabled in a USB application, it also powers internal USB transceiver
USB Transfer Types

- USB defines four transfer types
  - Control Transfers
    - Bi-directional
    - configure and send commands
  - Bulk Transfers
    - Uni-directional
    - burst
  - Interrupt Transfers
    - Uni-directional
    - periodic
  - Isochronous Transfers
    - Uni-directional
    - Periodic and time critical

**NOTE:** KL26 USB Controller supports all four transfers
USB Speed

• Low Speed
  - 1.5Mbps
  - Support only control and interrupt transfers

• Full Speed
  - 12Mbps
  - Support all four USB transfers: control, bulk, interrupt and isochronous transfers

• High Speed
  - 480Mbps (Feature not available on KL, Kinetis support USB HS in K20F, K60F, K61F family)

**NOTE:** KL26 USB Controller supports FS device and FS/LS host
Modes of operation overview

• **Device Mode**
  - USB will be configure to attend external host requests. In this mode the MCU has no control of the USB bus and all the transfers are started by the Host controller.

• **Host Mode**
  - In this mode the module works as the USB master having the entire control of the USB bus. Basically the Serial interface engine takes care of the timing and the frames but the software stack will take care of the transfer management of the bus.
Performance criteria

- **Clocking**
  - USB specification requires a maximum of +/- 0.25% (2,500 ppm) frequency variation. Be sure that the clock source plus the small PLL looses are inside the specification. It's always recommended to use a crystal or a resonator with 0.1% frequency variation for the USB clocking.

- **Layout**
  - Follow the layout recommendations to avoid signal integrity problems, even if this is just a 12Mhz module the performance will be affected if the module detect some errors during the transfers.

- **Software**
  - USB module is just the communication of the entire application so, make sure that your application is getting enough CPU attention using the USB interrupts as triggers for the stack instead of polling the USB flags.

- **Voltage protection**
  - It's always a good idea protect the VBUS and GND lines using ferrites or use a power distribution IC.
USB Stack Overview and Architecture
FSL USB Stack Overview

- **Universal Serial Bus (USB)** is a low-cost, fast, bi-directional, isochronous, and dynamically attachable serial interface that is consistent with the requirements of the PC platform of today and tomorrow. It is widely used in the microphone, keyboards, storage devices, cameras, printers, and so on;

- Freescale USB stack is an open source free software available to all the customers, supporting all Freescale MCUs with USB, includes ColdFire, HCS08, and Kinetis microcontroller families;
- Supports many popular development tools such as CodeWarrior, IAR and Keil uVision4 Integrated Development Environment.
- There are complete API Reference Manual which details all driver APIs and the User Guide which provides general guidelines for the use of the stack to develop new class drivers and applications.
- There are many demos facilitate the customers using, and enables rapid time to market for products.

- These stacks can be downloaded from [http://www.freescale.com/usb](http://www.freescale.com/usb)
FSL USB Stack Demos

<table>
<thead>
<tr>
<th>Number</th>
<th>Demo list</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Human Interface Device (HID) Kinetis Demo</td>
</tr>
<tr>
<td>2</td>
<td>Personal Healthcare – Multi Specialization Device Demo</td>
</tr>
<tr>
<td>3</td>
<td>Human Interface Device (HID) JM Demo</td>
</tr>
<tr>
<td>4</td>
<td>Personal Healthcare – Weigh Scale Device Demo</td>
</tr>
<tr>
<td>5</td>
<td>SD Card Demo</td>
</tr>
<tr>
<td>6</td>
<td>USB Audio Demo</td>
</tr>
<tr>
<td>7</td>
<td>DFU Class Demo</td>
</tr>
<tr>
<td>8</td>
<td>Battery Charging Device Demo</td>
</tr>
<tr>
<td>9</td>
<td>Video Device Class Demo</td>
</tr>
<tr>
<td>10</td>
<td>MSD and CDC Composite Demo</td>
</tr>
<tr>
<td>11</td>
<td>HID Audio Video Composite Demo</td>
</tr>
<tr>
<td>12</td>
<td>Printer Device Demo</td>
</tr>
</tbody>
</table>
USB Stack Architecture

**Freescale USB Stack**

- **USB Driver**
  - **HID Class Driver**
  - **CDC Class Driver**

**USB applications**
- **Callbacks**

**Class Driver API**
- **USB Driver API**

**USB Controller**

**USB Bus**
# USB Stack Layers Description

<table>
<thead>
<tr>
<th>USB layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB Driver layer</td>
<td>This layer is <strong>hardware dependent</strong>, including sits on the USB Controller, initialize or de-initialize a USB device controller, initializes endpoints, reads SETUP data for an endpoint, sends or receives data on an endpoint, sets the address of a USB device controller, stalls or unstalls an endpoint, cancels a pending transfer, and shuts down the USB device controller.</td>
</tr>
</tbody>
</table>
| Class driver layer         | It includes framework module, common class module, and class specific module.  
   ❖ The **framework module** handles all requests to the control endpoint and implements all requests defined;  
   ❖ The **common class module** contains implementation independent to application specific classes. Such as suspend, reset, stall, and SOF that needs to be present for all classes;  
   ❖ The **class specific module** implements class specific functionality. It mainly implements all interactions with non control endpoints, such as data sent and received, and also class specific requests on the control end point. |
| USB applications layer     | This layer use **the services provided by the class drivers** to implement application specific functions based on the USB communication link. Users can complete the applications easily based on the example applications, including the HID mouse, keyboard demo, SD card, USB audio, battery charging, personal health care, video device, MSD, and CDC composite demo. |
USB Stack HID Class Files and Data Flows Introduce
HID Class Driver Tasks

- **The Common Tasks**: Manages USB device controller, sets up USB buffer descriptor table (BDT) in USB RAM space for different transfers, monitors USB packets, encodes and decodes USB packets based on USB transactions and transfers, as well as handles the standard USB device requests.

- **The special Tasks**: Handles the standard requests for the HID interface and HID class-specific requests, it requires callbacks to be defined by its user to return standard USB descriptors.

- **Standard requests for the HID interface**:  
  - GET_DESCRIPTOR (HID Descriptor, Report Descriptor, and Physical Descriptor)

<table>
<thead>
<tr>
<th>Value</th>
<th>Class Descriptor Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x21</td>
<td>HID</td>
</tr>
<tr>
<td>0x22</td>
<td>Report</td>
</tr>
<tr>
<td>0x23</td>
<td>Physical</td>
</tr>
</tbody>
</table>
HID Class Driver

- HID class-specific requests:

<table>
<thead>
<tr>
<th>Request #</th>
<th>Request</th>
<th>Data Source</th>
<th>Data Length</th>
<th>Data Contents</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Get_Report</td>
<td>Device</td>
<td>Report length</td>
<td>Report</td>
<td>Yes</td>
</tr>
<tr>
<td>0x02</td>
<td>Get_Idle</td>
<td>Device</td>
<td>1</td>
<td>Idle duration</td>
<td>No</td>
</tr>
<tr>
<td>0x03</td>
<td>Get_Protocol</td>
<td>Device</td>
<td>1</td>
<td>Protocol</td>
<td>Required for boot devices</td>
</tr>
<tr>
<td>0x09</td>
<td>Set_Report</td>
<td>Host</td>
<td>Report length</td>
<td>Report</td>
<td>No</td>
</tr>
<tr>
<td>0x0A</td>
<td>Set_Idle</td>
<td>Host</td>
<td>0</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>0x0B</td>
<td>Set_Protocol</td>
<td>Host</td>
<td>0</td>
<td>None</td>
<td>Required for boot devices</td>
</tr>
</tbody>
</table>
# Freescale USB stack HID files

<table>
<thead>
<tr>
<th>Files</th>
<th>Description and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB Driver files</td>
<td></td>
</tr>
<tr>
<td>Usb_dci_kinetis.c</td>
<td>USB device controller interface (DCI) code for specific device (Kinetis in this example)</td>
</tr>
<tr>
<td>Usb_dci_kinetis.h</td>
<td>Contains Macro's and functions needed by the DCI layer, device specific (Kinetis in this example)</td>
</tr>
<tr>
<td>Usb_bdt_kineits.h</td>
<td>Buffer Descriptor Table definition for specific device</td>
</tr>
<tr>
<td>Usb_dciapi.h</td>
<td>DCI APIs function definitions</td>
</tr>
<tr>
<td>Usb_driver.c</td>
<td>USB driver API and the related code (HW independent)</td>
</tr>
<tr>
<td>Usb_devapi.h</td>
<td>Header file for device layer APIs (HW independent)</td>
</tr>
<tr>
<td>HID class driver files</td>
<td></td>
</tr>
<tr>
<td>Usb_framework.c</td>
<td>Code to handle USB device framework as described in USB specification Chapter 9</td>
</tr>
<tr>
<td>Usb_framework.h</td>
<td>USB device framework header file</td>
</tr>
<tr>
<td>Usb_hid.c</td>
<td>HID class specific driver</td>
</tr>
<tr>
<td>Usb_hid.h</td>
<td>Header file for HID class specific driver</td>
</tr>
<tr>
<td>Usb_class.c</td>
<td>Class layer class independent driver code</td>
</tr>
<tr>
<td>Usb_class.h</td>
<td>Macros and function prototypes for class independent driver code</td>
</tr>
</tbody>
</table>
**Freescale USB stack HID files— continued**

<table>
<thead>
<tr>
<th>Application code file</th>
<th>Usb_descriptor.c</th>
<th>Template code contains USB Framework Module interface. It contains various descriptors defined by USB Standards like device descriptor, configuration descriptor, string descriptor, and other class specific descriptors, user can modify these variables and function implementations to suit the requirement.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Usb_descriptor.h</td>
<td>Mandatory header files for the application to implement. Framework and class drivers include this file for function prototype definitions and data structures described in usb_descriptor.c. User modifying usb_descriptor.c should also modify MACROs in this file as well</td>
</tr>
<tr>
<td></td>
<td>user_config.c</td>
<td>Required header file to define various compile time macros. These parameters are essential for successful compilation of source code. Mandatory macros that need to be defined are: USB_PACKET_SIZE, LONG_SEND_TRANSACTION, MULTIPLE_DEVICES, DOUBLE_BUFFERING_USED</td>
</tr>
<tr>
<td></td>
<td>user_config.h</td>
<td>Required header file if MULTIPLE_DEVICES macro is not defined in user_config.h. It defines all related service callbacks that are essential for successful compilation of source code.</td>
</tr>
</tbody>
</table>
USB stack HID Driver API usage and Resource usage
HID Driver API usage

- The HID class driver provides many APIs to the higher layer software. In today’s joystick demo, it only applies the following four API.

<table>
<thead>
<tr>
<th>API Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_Class_HID_Init()</td>
<td>Initialize the HID class with controller ID and some callbacks</td>
</tr>
<tr>
<td>USB_Class_HID_DeInit()</td>
<td>De-initialize the HID class</td>
</tr>
<tr>
<td>USB_Class_HID_Periodic_Task()</td>
<td>Periodic call to the class driver to complete pending tasks</td>
</tr>
<tr>
<td>USB_Class_HID_Send_Data()</td>
<td>Sends the HID report to the host</td>
</tr>
</tbody>
</table>
HID Driver API usage

- The class driver also requires the following API functions (USB descriptor related for framework module) to be implemented by the higher layer software:

<table>
<thead>
<tr>
<th>API Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_Desc_Get_Descriptor()</td>
<td>Gets various descriptors as specified from the application</td>
</tr>
<tr>
<td>USB_Desc_Get_Endpoints()</td>
<td>Gets the endpoints used and their properties</td>
</tr>
<tr>
<td>USB_Desc_Get_Interface()</td>
<td>Gets the currently configured interface</td>
</tr>
<tr>
<td>USB_Desc_Set_Interface()</td>
<td>Sets new interface</td>
</tr>
<tr>
<td>USB_Desc_Remote_Wakeup()</td>
<td>Checks whether the application supports remote wakeup or not</td>
</tr>
<tr>
<td>USB_Desc_Valid_Configuration()</td>
<td>Checks whether the configuration being set is valid or not</td>
</tr>
<tr>
<td>USB_Desc_Get_Interface()</td>
<td>Checks whether the interface being set is valid or not</td>
</tr>
</tbody>
</table>

- The reference code implementing all the preceding required APIs is defined in usb_descriptor.c of a demo code.
To use HID class layer API functions, the higher layer software must follow steps below:

- Call `USB_Class_HID_Init()` to initialize the class driver, all the layers below it, and the device controller.
- Wait the `USB_APP_ENUM_COMPLETE` event, then move higher layer into ready state.
- Call `USB_Class_HID_Send_Data()` to send data/report to the host through the underlying layer drivers.
HID driver callback functions

HID driver callback routines are required to handle USB generic class events, USB vendor specific requests, and USB class specific request. The prototype of generic class callback routine is as follows:

typedef void(_CODE_PTR_ USB_CLASS_CALLBACK)(
    uint_8 controller_ID, // USB controller ID
    uint_8 type, // type of event
    void* data ) // event data based on the event type

The prototype of vendor specific request callback is as follows:

typedef uint_8 (_CODE_PTR_ USB_REQ_FUNC)(
    uint_8 controller_ID, // USB controller ID
    USB_SETUP_STRUCT *setup_packet, // SETUP packet received from host
    uint_8_ptr *buff, // pointer to the buffer to be returned with data
    USB_PACKET_SIZE *size); // size of data to be received or sent

The prototype of class specific request callback is as follows:

typedef uint_8 (_CODE_PTR_ USB_CLASS_SPECIFIC_HANDLER_FUNC)(
    uint_8 request, // request code from SETUP packet
    uint_16 value, // value code from SETUP packet
    uint_8_ptr *buff, // pointer to the buffer to be returned with data
    USB_PACKET_SIZE *size); // size of data to be received or sent
USB stack Service events

- The following service events defined by USB driver:

<table>
<thead>
<tr>
<th>Service Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_SERVICE_BUS_RESET</td>
<td>Signals a bus reset is just occurred</td>
</tr>
<tr>
<td>USB_SERVICE_EP&lt;n&gt;</td>
<td>Signals an endpoint transaction is just occurred, where &lt;n&gt; is the endpoint number the device supports.</td>
</tr>
<tr>
<td>USB_SERVICE_ERROR</td>
<td>Signals an error is just occurred</td>
</tr>
<tr>
<td>USB_SERVICE_RESUME</td>
<td>Signals a device resumes from suspend state</td>
</tr>
<tr>
<td>USB_SERVICE_SLEEP</td>
<td>Device sleep event</td>
</tr>
<tr>
<td>USB_SERVICE_SOF</td>
<td>Start of Frame event</td>
</tr>
<tr>
<td>USB_SERVICESTALL</td>
<td>Endpoint stall event</td>
</tr>
<tr>
<td>USB_SERVICE_SUSPEND</td>
<td>Device suspend event</td>
</tr>
</tbody>
</table>
USB stack Service events

The preceding events are intended for class drivers, not for applications. For HID class driver, the service callbacks for the related events are defined in usb_hid.c.

<table>
<thead>
<tr>
<th>Service event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_APP_BUS_RESET</td>
<td>Bus reset event</td>
</tr>
<tr>
<td>USB_APP_CONFIG_CHANGED</td>
<td>Configuration is changed</td>
</tr>
<tr>
<td>USB_APP_ENUM_COMPLETE</td>
<td>Device enumeration is completed and ready for use</td>
</tr>
<tr>
<td>USB_APP_SEND_COMPLETE</td>
<td>Data transfer is completed</td>
</tr>
<tr>
<td>USB_APP_DATA_RECEIVED</td>
<td>Data is received. Not used for HID class</td>
</tr>
<tr>
<td>USB_APP_ERROR</td>
<td>Error occurs</td>
</tr>
<tr>
<td>USB_APP_GET_DATA_BUFF</td>
<td>Get data buffer. Not used for HID class</td>
</tr>
<tr>
<td>USB_APP_EP_STALLED</td>
<td>Endpoint is stalled</td>
</tr>
<tr>
<td>USB_APP_EP_UNSTALLED</td>
<td>Endpoint is unstalled</td>
</tr>
<tr>
<td>USB_APP_GET_TRANSFER_SIZE</td>
<td>Get the transfer size. Not used for HID class</td>
</tr>
</tbody>
</table>
NOTE:

- These data are calculated directly from the linker map file generated by CodeWarrior. The size of HID demo counts in the size of all of its components including USB driver, HID driver, the hid-keyboard, hid-mouse and hid-generic device.
- Flash memory contains code, initialized data and constant variables like USB descriptors. RAM contains initialized and uninitialized data, heap, and stack as well. But here does not count in the heap and stack size for drivers.
HID Class Joystick Demo
Introduce and Hands-On
LAB 1: HID joystick demo tasks

- HID joystick demonstrates joystick control with **two axis** (X and Y), **one throttle**, **one hat switch**, and **four buttons**.

<table>
<thead>
<tr>
<th>On board device</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>FXOS8700</td>
<td>Emulate the mouse movement in X and Y direction and the throttle control (Z direction).</td>
</tr>
<tr>
<td>Touch Slider</td>
<td>Simulate the four buttons.</td>
</tr>
<tr>
<td>POTENTIOMETER</td>
<td>Simulate the movement of the hat switch control, is connected to the ADC channel 9</td>
</tr>
</tbody>
</table>
FXOS8700, Emulate the mouse movement in X and Y direction and throttle control.

Touch slide, Simulate the four buttons

add a external potentiometer to ADC channel 4 to simulate hat switch control

The on board resource used in the demo

KL26 64 LQFP

Reset

KL26Z USB

OpenSDA
The usage of HID Joystick Demo:

- Tilt the board left or right to move X left or right.
- Tilt the board forward or backwards to move Y up or down.
- Pull up or down the board to move throttle up or down.
- Touch the slider pad E1 and slide on it to select different buttons.
- Rotate the external potentiometer to wheel the hat switch, it is connected to the ADC channel 4.
HID Joystick report structure:

<table>
<thead>
<tr>
<th>Byte</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Throttle</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Buttons &amp; Hat Switch</td>
</tr>
</tbody>
</table>

Upper 4 bits are Buttons  Lower 4 bits are
4,3,2,1 from msb to lsb  Hat Switch

static signed char
hid_joy_report_in[4] =
{
  0
};
HID Joystick Demo flow chart

**Reset**

`accel_init()` and `TSI_Init()` to initialize accelerometer and Tss channel

Initialize X, Y, Hat, Throttle fields in the `hid_joy_report_in[4]`

**USB_Class_HID_Init()** to initialize the HID class driver and Controller layer

**ADC_Init()** to Initialize ADC and start conversion to get original X, Y, Throttle, hat

**joy_start_ms_timer()** to start a timer to do X/Y/Throttle/Hat simulation
HID Joystick Demo flow chart

USB_App_Callback() to check the enum is complete

Enum is complete

Check the Joystick X/Y/Throttle/Hat switch states

states changed?

USB_Class_HID_Send_Data() transfer the changed report data to endpoint

joy_scan_matrix() to poll buttons state

States changed?

USB_Class_HID_Send_Data() transfer the changed report data to endpoint
Hands-on -- HID Class joystick demo

第一步： 安装Freescale_USB_Stack_v4.1.1_1.exe；

第二步： 打开xxx\Freescale USB Stack v4.1.1\Source\Device\app目录，新建joystick_demo文件夹用于存放新的工程；
第三步：拷贝xxx\Freescale USB Stack v4.1.1\Source\Device\app\hid目录中如下图红色方框1和2的文件到新建的joystick_demo文件夹，并删除cw10文件夹下除Kinetis_l2k之外的文件夹；
第四步：打开工程工程文件xxx\joystick_demo\cw10\kinetis_l2k，在hid mouse例程的基础上进行修改，主要修改的内容包括：

1. HID class 描述符文件的修改，主要修改文件是usb_descriptor.c；
2. HID class callback函数的修改，主要修改文件是mouse_button.c；
3. Lab 中需要用到板载的FXOS8700, TSI touch slider以及AD采样驱动文件的添加，相应的驱动文件已打包成peripheral_driver.zip；
4. main函数的修改和中断函数的声明，主要修改的文件是vector.c和main_kinetis.c文件

下面将Step by step的介绍如何在一个已有的HID class mouse例程上完成HID class joystick的程序修改。
修改 usb_descriptor.c

Step 1. 在usb_descriptor.c文件中添加新的joystick report报告描述符,为节省时间在本Lab中直接将这些描述符文件定义在g_joy_report_descriptor.c文件中,该文件在附件中,可以直接拷贝到line 49 g_device_descriptor[]定义的后面；

配置该HID class属于Generic Desktop Page (0x01), 用法为 Joystick (0x04)

关于描述符文件的格式和意义请参照usb.org 上HID协议文档和usage表文档。可以通过 USB HID 描述符工具Tool DT.exe生成
修改 `usb_descriptor.c`

**Step 2.**  修改 `usb_descriptor.c` 文件中配置描述符；

改变用于 joystick 的 interface protocol 为 0x00

修改 joystick descriptor 的 Length
修改 usb_descriptor.c

Step 3. 自定义字符串描述符显示，配置显示为DWF HID Joystick DEMO，如果用户需要定义其他的字符串需要在此处添加USB_STR_2长度信息。
Step 4. 修改标准描述符长度信息，增加step 1中report描述符的长度

增加新的需要显示的字符串描述符长度
USB在进行初始化时会传递用3个callback函数：

- Generic class callback，用于判断Enum或者Bus Reset的状态；
- vendor specific request callback本应用没有涉及，为NULL
- Class specific request callback，用于处理HID Class request；

```
void TestApp_Init(void)
{
    uint8 error;

    rpt_buf[3] = 0x00; /* always zero */

    /*((uint32_ptr)rpt_buf) = 0; */
    /*((uint32_ptr)null_buf) = 0; */
    DisableInterrupts;

    #if (defined _MCF51MM256_H) || (defined _MCF51JE256_H)
        usb_int_dis();
    #endif

    /* Initialize the USB interface */
    error = USB_Class_HID_Init(CONTROLLER_ID, USB_App_Callback, NULL,
                               USB_App_Param_Callback);
    UNUSED(error);

    EnableInterrupts;

    #if (defined _MCF51MM256_H) || (defined _MCF51JE256_H)
        usb_int_en();
    #endif
}
```
HID Class Callback函数的修改

Step 5.
mouse_button.c文件中generic class callback USB_APP_Callback的修改，该文件用于在枚举结束后置位枚举结束标志；

```c
void USB_App_Callback(
    uint8 controller_ID, /* [IN] Controller ID */
    uint8 event_type,    /* [IN] value of the event*/
    void* val             /* [IN] gives the configuration value*/
)
{
    UNUSED (controller_ID)
    UNUSED (val)
    if((event_type == USB_APP_BUS_RESET) || (event_type == USB_APP_CONFIG_CHANGED))
    {
        joy_init=FALSE; /*YN*/
    }
    else if(event_type == USB_APP_ENUM_COMPLETE)
    { /* if enumeration is complete set mouse_init so that application can start */
        joy_init=TRUE;
    }
    return;
}
```
HID Class Callback函数的修改

Step 6. mouse_button.c文件中class specific request callback函数USB_APP_Callback的修改，

在HID get report请求中修改report in数据指针指向hid_joy_report_in[];

默认的mouse requests中report pointer的设置
void TestApp_Task(void)
{
    /* call the periodic task function */
    USB_Class_HID_Periodic_Task();
    if(joy_init) /*check whether enumeration is //YN*/
        Emulate_Mouse_WithButton();
}
HID Class Callback函数的修改

Step 7. 在周期行调度函数`Emulate_mouse_withButton`中添加对事件的处理，包括加速度传感器值的改变，TSI触发事件；

该变量会在TMR 200ms中断中根据加速度传感器的状态被改变

TSI touch slider 按键扫描

调用HID Class数据发送函数把hid_joy_report_in的数据发送到Host

```c
if(xy_state_changed) {
    xy_state_changed = FALSE;
    xy_state_changed = FALSE;
    TSI触发事件;
}
if(hat_state_changed) {
    hat_state_changed = FALSE;
    /* Write report data. */
    (void)USB_Class_HID_Send_Data(CONTROLLER_ID,HID_ENDPOINT,hid_joy_report_in,
                                 HID_JOY_REPORT_IN_SIZE);
}
/* Look for buttons state. */
if (cur_state != last_state) {
    last_state = cur_state;
    /!
    (void)USB_Class_HID_Send_Data(CONTROLLER_ID,HID_ENDPOINT,hid_joy_report_in,
                                 HID_JOY_REPORT_IN_SIZE);
}
return;
```
外设驱动文件的添加

Step 8. 解压peripheral_driver.zip，拷贝peripheral_driver文件夹放到xxx\Freescale USB Stack v4.1.1\Source\Device\app\joystick_demo\cw10\kinetis_l2k目录中，并添加该目录的搜索路径

该文件夹中是AD, TSI以及FXOS8700的驱动文件，按键的定义，另外还包括一些数据类型的定义及变量声明，需要在main中把这些头文件包含进来
peripheral_driver文件夹内文件内容

- ADC Driver
- 芯片器件头文件包含
- FRDM 板接口定义
- FXOS8700 Driver
- 串口输出
- 变量定义和函数声明
- TSI Driver
- 数据类型定义
Main函数的修改

Step 9: 在main_kinetis.c文件中添加外设的驱动头文件
```c
#include "common.h" //YNN
#include "freedom_gpio.h"
#include "hal_dev_mma8451.h"
#include "TSIDrv.h"
#include "joy.h"
```

USB初始化、枚举以及数据传输调度

Touch slider，加速度传感器FXOS8700以及ADC通道的初始化

启动Timer定时器200ms中断

Step 10: 在main_kinetis.c的main函数中分别添加外设的初始化代码
```c
if MAX_TIMER_OBJECTS
(void)TimerQInitialize(0);
#endif

TSI_Init();
accel_init();
ADC_Init();
joy_start_mS_timer(200);

/* Initialize the USB Test Application */
TestApp_Init();

while(TRUE)
{
  Watchdog_Reset();
  /* Call the application task */
  TestApp_Task();
}
#endif
```

Main函数的修改
中断向量表的修改

**Step 11:** 修改Vector.c，重定向中断的名称；

声明外部中断函数

添加Timer和TSI中断函数指针到对应的中断号
Step 12: 添加必要的中断函数，为方便应用在本Lab中直接将这些函数定义到mouse_button.c文件中，这段函数定义在附件button_add_code.c中，可以直接拷贝过来到mouse_button.c的void TestApp_Init(void)函数声明的前面；
Run the demo

**Step 13:** 完成以上修改，点击①处编译程序，然后点击②处三角形选择Debug Configuration，③处选择OpenSDA下载工具，最后点击④处启动Debug；
连接FRDM-KL26的USB接口(Not OpenSDA接口)，然后点击启动按钮程序运行后，会显示正在安装device driver，如下图1所示，显示的名称为”DWF HID joys”，正是Step 3中定义的string描述符；
随后PC Host端会自动安装HID设备驱动程序，完成后会在Device Management中看到已经驱动完成的设备，如图3的红色方框所示；
Run the demo

Step 14: 如下图打开Win 7 命令窗口，输入Game Controller，然后回车，会跳出右图小窗口，会看到MK HID Joystick DEMO 已经显示在game controller列表中；
Run the demo

Step 15: 点击上图中的Properties，会弹出右图窗口；

- 执行以下步骤查看窗口变化:
  - 把板子向左或者向右倾斜会移动“+”号在X Axis的坐标位置；
  - 把板子向前或者向后倾斜会移动“+”号在Y Axis的坐标位置；
  - 向上或者向下运动板子会改变 throttle up 或者 down；
  - 触摸模拟滑条E1会看到不同的Buttons会亮灭；
  - 改变ADC通道4的电压会看到View Hat的红色点会沿着圆环改变；
Q&A