NUC501 USB HID Sample Code Reference Guide

V1.01.001

Publication Release Date: May. 2009

The information in this document is subject to change without notice.

The Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Table of Contents

1. Introduction 4

1.1. Scope 4
1.2. Feature 4
1.3. Limitation 4

2. USB Driver Framework 5

3. Calling Sequence 6

3.1. Run a USB Device 6
3.2. Customize a USB Device 6

4. Code Section 8

4.1. Introduction 8
4.2. Function Description 8

5. Execution Environment Setup and Result 9

5.1. Test Smpl_HID 9
5.2. Result (Keyboard) 9
5.3. Result (Mouse) 9

6. Revision History 10

1. Introduction

This document explains a sample code, Smpl_HID which demos how to create a USB HID (Human Interface Device) device in NUC501.

1.1. Scope

This article is provided for programmers who are using USB IP to complete their USB applications. It is assumed that the reader is familiar with the Universal Serial Bus Specification, Revision 1.1 and HID.

1.2. Feature

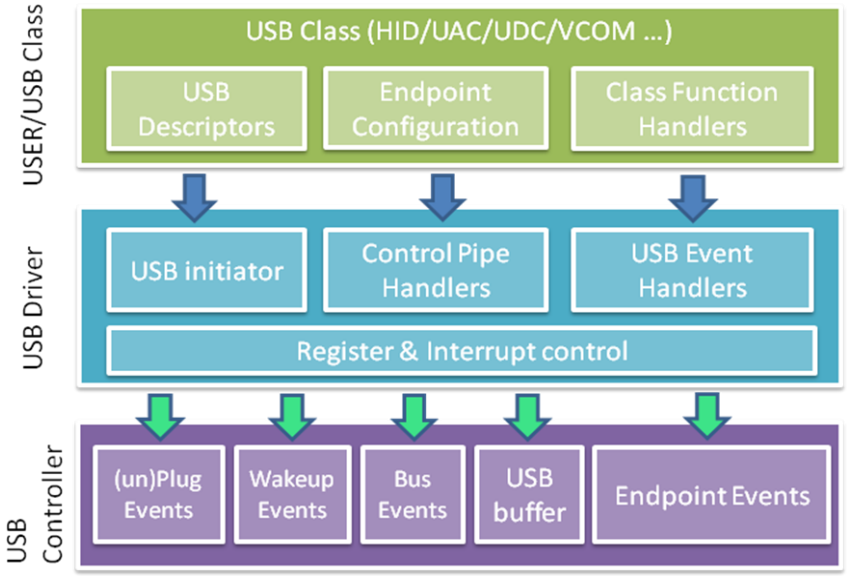
- USB HID Keyboard
- USB HID Mouse

1.3. Limitation

- Support no Interrupt Out.
- Limit total configuration descriptor size to Control packet size.

2. USB Driver Framework

The USB driver is based on an event trigger mechanism to control 501 USB device. According to all necessary initial components, the device driver will call the relative callback handlers to deal the events. The framework chart is as follows:



USB Controller

The USB controller can issue SETUP or IN/OUT event when getting relative tokens. There are 256 bytes buffer in the USB controller to send or receive data when getting these tokens.

USB Driver

The USB driver is used to initialize the USB control according to the USB endpoint configuration. It also processes the standard requests of control pipe of USB and calls the relative USB event handlers.

USER/USB Class

In this layer, user should define the all descriptors, such as device descriptor, configuration descriptors, interface descriptors, endpoint descriptors and class descriptors and etc. An endpoint configuration to define the USB buffer usage of each endpoint is also required. The relative device/class function handler also needs to be implemented here.

3. Calling Sequence

3.1. Run a USB Device

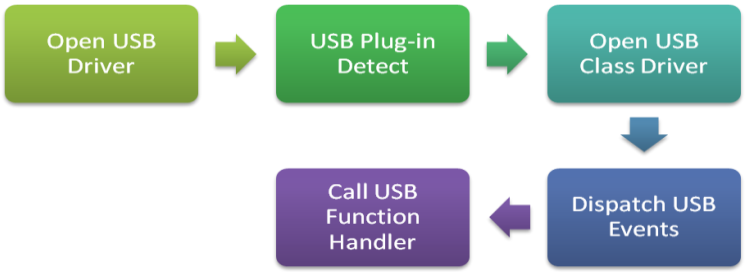


Figure 3-1: Control Flow of Run a USB Device

3.2. Customize a USB Device

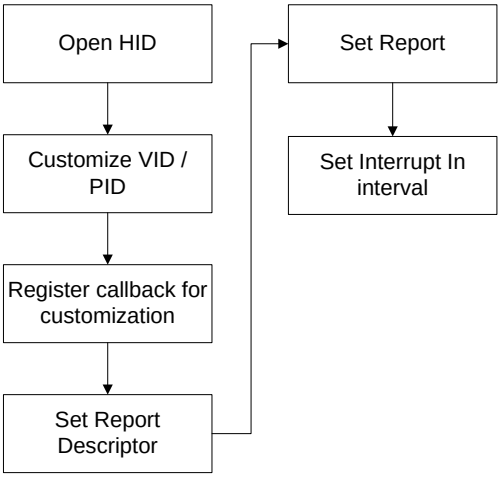


Figure 3-2: Control Flow of Customizing a USB Device

1. Open HID library to create a HID device.
2. Customize VID / PID if needed.

3. Register USB-related callback functions for customization if needed.
4. Set customized Report Descriptor.
5. Set customized Report.
6. Set customized Interrupt In interval if needed.

4. Code Section

4.1. Introduction

The sample code Smpl_HID could support HID keyboard or HID mouse by using conditional compiling. The sample also supports to work with or without interrupt, i.e. handle the USB event by polling.

To configure the Smpl_HID to be HID keyboard, the macro HID_FUNCTION should be defined as HID_KEYBOARD otherwise, it should be defined as HID_MOUSE.

The main routine of Smpl_HID is HID_MainProcess. HID_MainProcess will call the USB driver and HID driver to initial USB and handle the USB events. Because it is implemented as event driving task, it uses a while loop to wait and handle the events. User may add any other code to the while loop to do something.

4.2. Function Description

In the main function, the PLL clock is set to 192MHz and CPU is set to 96MHz because the PLL clock needs to be divided to 48MHz for USB clock.

After setting the proper clock, the main routine HID_MainProcess() could be called to start USB HID device. After connecting to host (PC), the PC will recognized a USB HID device and start to get report from the device. When host getting the report, the callback function HID_GetInReport() will be called. User could change the report contents by modify the HID_GetInReport().

If the HID is configured as a HID keyboard, the index 2 of input buffer could be used to tell the host what is the current pressed key.

If the HID is configured as a HID mouse, the input buffer format is defined as follows:

buf [0]							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
Reserved					Middle Button	Right Button	Left Button

buf [1]							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
Offset of X axis. Must be -127~+127.							

buf [2]							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
Offset of Y axis. Must be -127~+127.							

5. Execution Environment Setup and Result

5.1. Test Smpl_HID

The HID sample code, Smpl_HID could be built to execute in SRAM or boot from SPI Flash. Build the Smpl_HID to SRAM target and download to NUC501 DEV Board through ICE or Build the Smpl_HID to ROM target and download to NUC501 SPI Flash through USB by using NUC501 WriterTool.

Then it is able to execute the code by ICE or reboot form SPI Flash which contains the Smpl_HID code.

5.2. Result (Keyboard)

After execution of this sample code, the test board runs as a keyboard and after connected to Windows, the keyboard on Windows device manager can be showed. And you can open the notepad to test it. The key definitions on NUC501 DEV Board are as follows:

Key of NUC501 DEV Board	Code defined
A	‘a’ or ‘A’
B	‘b’ or ‘B’
C	‘c’ or ‘C’
D	“abc” or “ABC”

5.3. Result (Mouse)

After execution of this sample code, the test board runs as a mouse and after connected to Windows. User can control mouse behavior through UART port of 501. The available keys are ‘a’, ‘b’, ‘c’, ‘d’, ‘.’ And ‘,’.

6. Revision History

Version	Date	Description
V1.00.001	May.14, 2009	● Created.
V1.01.001	Nov.10, 2009	● Update for new USB driver and HID sample code

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.