Implementation of Vendor Command

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	Demonstrate the implementation of USB vendor command
BSP Version	NUC123 Series BSP CMSIS V3.01.001
Hardware	NuTiny-EVB-NUC123-LQFP64 v1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Function Description

1.1 Introduction

This document demonstrates how to implement a vendor command on a USB HID device. It includes 2 vendor command, read and write. USB tool, Bus Hound is used to show the transfer procedure and vendor command sending. User can observe the transfer result through UART terminal tool also.

1.2 Principle

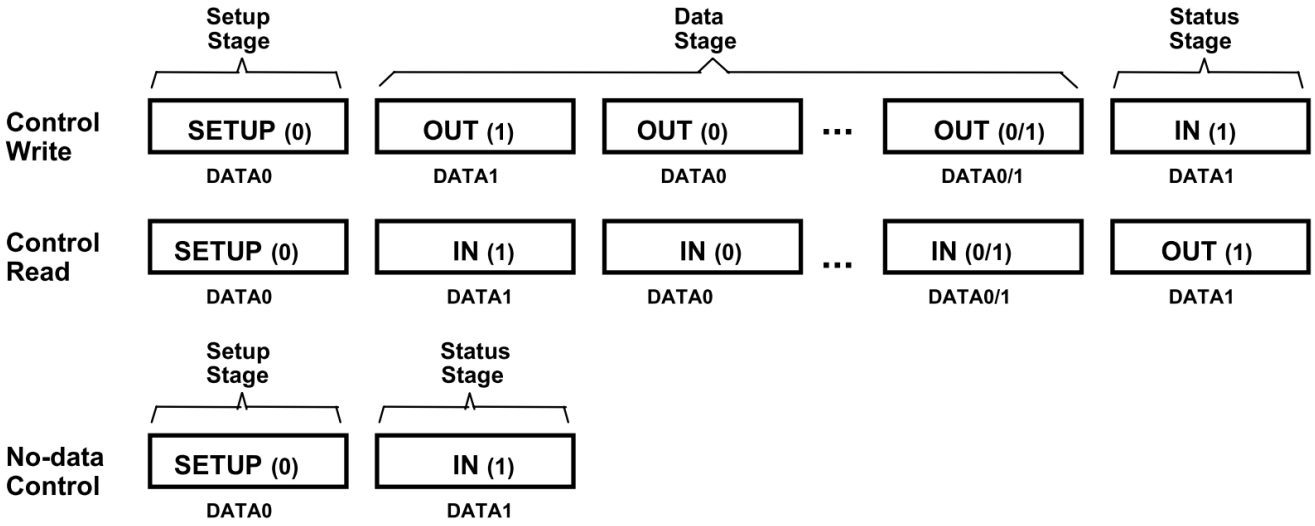


Figure 1: Control Read and Write Sequences

USB vendor command is based on control transfer. It has 3 stages, setup stage, data stage and status stage, shown in Figure 1. Setup stage and status stage are necessary. Data stage is optional depending on application.

Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bitmap	Characteristics of request: D7: Data transfer direction 0 = Host-to-device 1 = Device-to-host D6...5: Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4...0: Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4...31 = Reserved
1	*bRequest*	1	Value	Specific request (refer to Table 9-3)
2	*wValue*	2	Value	Word-sized field that varies according to request
4	*wIndex*	2	Index or Offset	Word-sized field that varies according to request; typically used to pass an index or offset
6	*wLength*	2	Count	Number of bytes to transfer if there is a Data stage

Figure 2: Format of Setup Data

This demonstration implements vendor command doing 64 bytes data reading and writing. Packet in setup stage must compliant with the USB control transfer setup packet format shown in Figure 2. Thus we came up with the command format in Table 1. In this table, field, bmRequestType, defines the command type and transfer direction. Field, bRequest, is the command we define for the reading and writing, 0x1 is the writing command and 0x2 is the reading command. Field, wLength, is the transfer length of payload. Other fields not mentioned are not used in this demonstration so set them as 0.

Vendor-Defined Command	bmRequestType	bRequest	wValue	wIndex	wLength
VENDOR_WRITE_TEST	0x40	0x01	0	0	0x40
VENDOR_READ_TEST	0xC0	0x02	0	0	0x40

Table 1: Setup packet of Vendor Command

1.3 Demo Result

USB tool, Bus Hound, is used for control transfer demonstration. Execute the tool after the device connected desktop, the device tree will show the device on device tree. In Figure 3, we can see the device with hexadecimal VID 0416 and PID B001 which is the device we demonstrate. Follow the steps shown in Figure 3 to enter the window, Bus Commander.

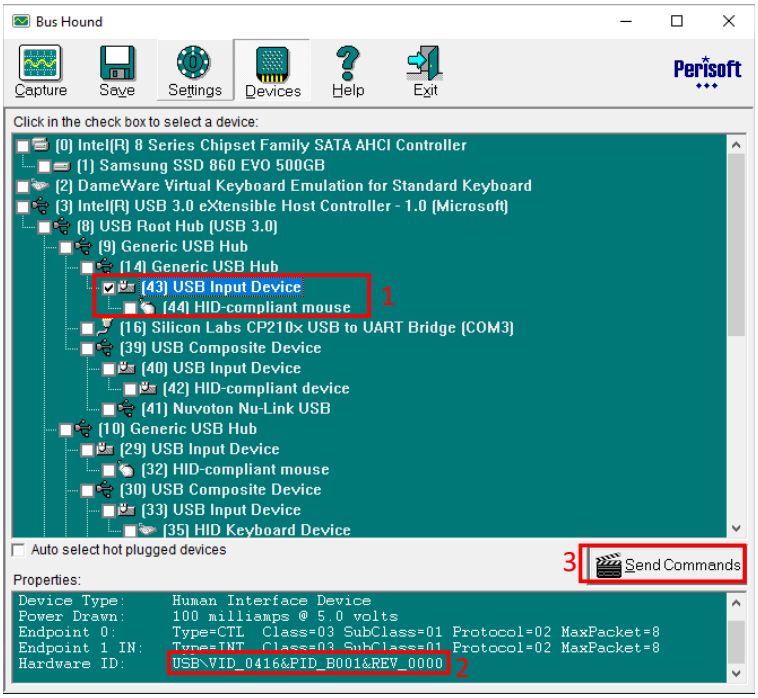


Figure 3: Device Tree on Bus Hound

1.3.1 Reading Command

In bus commander window, execute the following steps to observe the behavior of transfer shown in Figure 4:

1. Check the initial value which is 0x00 ~ 0x3f in data buffer by pressing any key through terminal tool after the program is running.
2. Input the reading command defined in Table 1.
3. Press “Run” button to execute reading command.
4. We can see that the received payload in Bus Commander window are the same with the data shown in step 1. It indicates the transfer is complete.

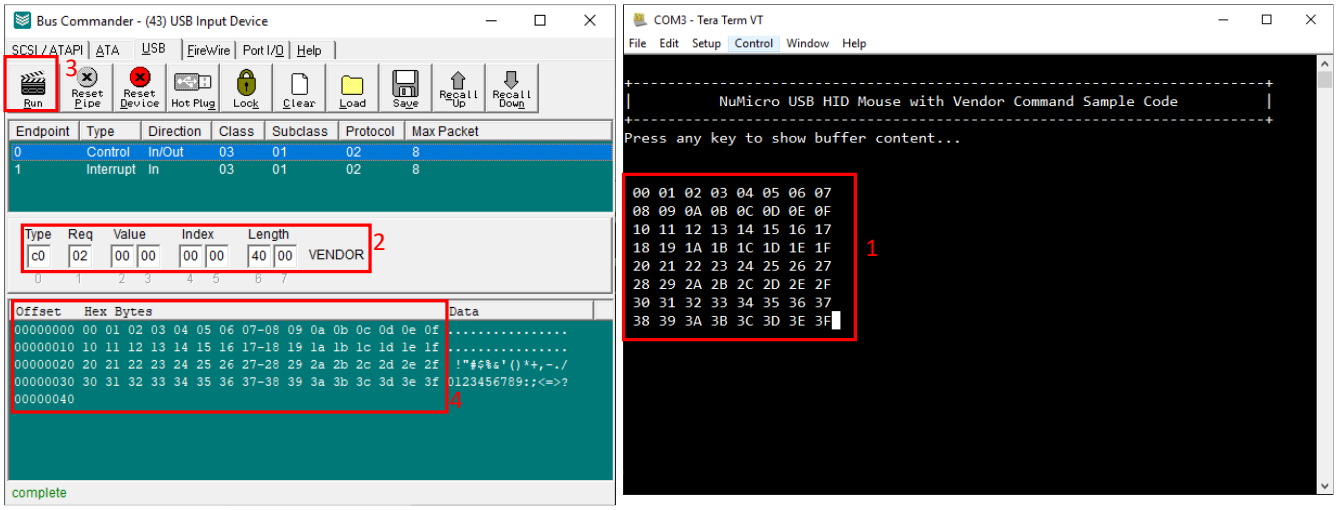


Figure 4: Operation of Reading Command

1.3.2 Writing Command

In bus commander window, execute the following steps to observe the behavior of transfer shown in Figure 5.

1. Check the initial value which is 0x00 ~ 0x3f in data buffer by pressing any key through terminal tool after the program is running.
2. Input the writing command defined in Table 1.
3. Input data to the data field in Bus Commander window.
4. Press “Run” button to execute writing command.
5. Press any key in terminal window, we can see that the received payload in terminal window are the same with the data shown in step 3. It indicates that the input data in Bus Hound are written to the buffer in device.

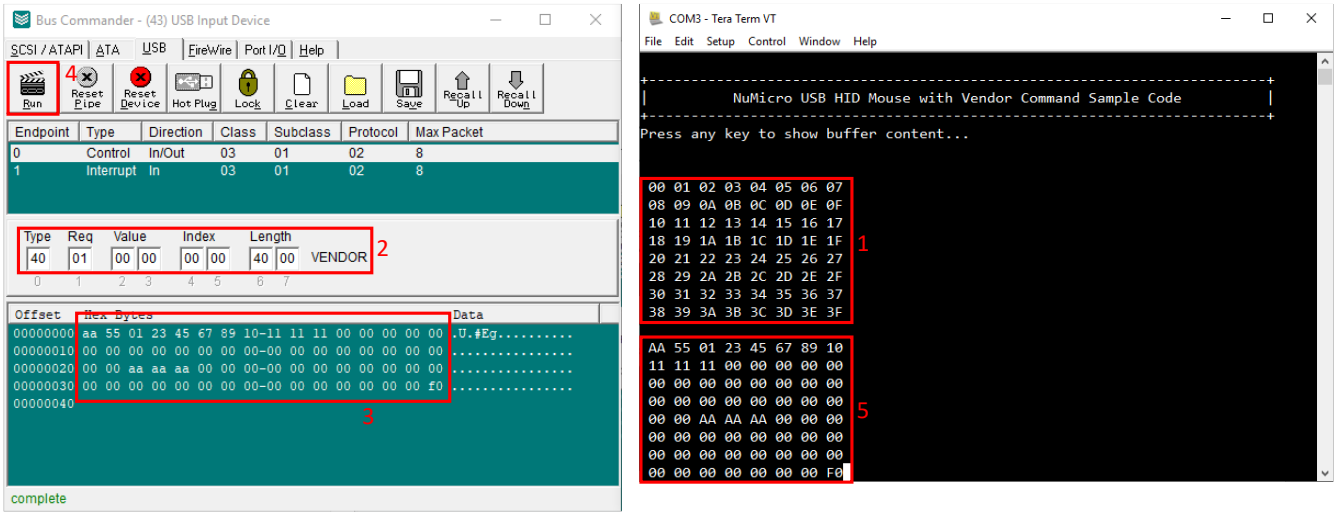


Figure 5: Operation of Writing Command

2 Code Description

2.1 main.c

Call function, USBD_SetVendorRequest(), to assign vendor command handler, VENDOR_Request(), for subsequent vendor command handling.

```
    /* Set Callback function for Vendor Command */
    USBD_SetVendorRequest(VENDOR_Request);
```

2.2 hid_vendor_command.h, hid_vendor_command.c

File, hid_vendor_command.h defines the writing command as 0x1, reading command as 0x2.

```
/*!<USB VENDOR CMD */
#define VENDOR_WRITE_TEST 0x1
#define VENDOR_READ_TEST  0x2
```

Function, VENDOR_Request is for vendor command handling. At the beginning, keep the setup packet in a local buffer for subsequent processing.

```
void VENDOR_Request(void)
{
    uint8_t buf[8];
    uint32_t wLength;

    USBD_GetSetupPacket(buf);

    wLength = buf[7];
    wLength = buf[6] | (wLength << 8);
```

buf[0] is the field which is bmRequestType refer to Table 1. It indicates data transfer direction and command type. The handler has two parts handling different direction of transmission. The setup stage is handled by USB driver. Function, VENDOR_Request, only handles data stage and status stage.
In data stage, it needs checking if the field, bRequest(buf[1]), is the command to be processed. And call function, USBD_PrepareCtrlIn(), setting the buffer address and payload length.

```
    if(buf[0] & 0x80)    /* request data transfer direction */
    {
        /* Device to host */
        switch(buf[1])
        {
            case VENDOR_READ_TEST:
            {
                wLength = Minimum(wLength, sizeof(g_au8temp));

                /* Data stage */
                USBD_PrepareCtrlIn((uint8_t *)g_au8temp, wLength);
```

In status stage, set data toggle flag as 1. And call USBD_PrepareCtrlOut() for endpoint 0 and

payload setting. There is no data for transmitting in status stage so all parameters are 0.

```
                /* Status stage */
                USBD_SET_DATA1(EP1);
                USBD_PrepareCtrlOut(0, 0);
                break;
            }

            default:
            {
                /* Setup error, stall the device */
                USBD_SetStall(EP0);
                USBD_SetStall(EP1);
                break;
            }
        }
    }
```

When writing command, VENDOR_WRITE_TEST, received, call USB driver USBD_PrepareCtrlOut() setting buffer address and payload length for output direction data receiving in data stage. No data transferring in status stage, so set zero size in endpoint 0 payload. Received data will be in buffer, g_au8temp, after transfer is done.

```
  else
    {
        /* Host to device */
        switch(buf[1])
        {
            case VENDOR_WRITE_TEST:
            {
                /* Data stage*/
                USBD_PrepareCtrlOut(g_au8temp, wLength);

                /* Status stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 0);
                break;
            }

            default:
            {
                /* Setup error, stall the device */
                USBD_SetStall(EP0);
                USBD_SetStall(EP1);
                break;
            }
        }
    }
}
```

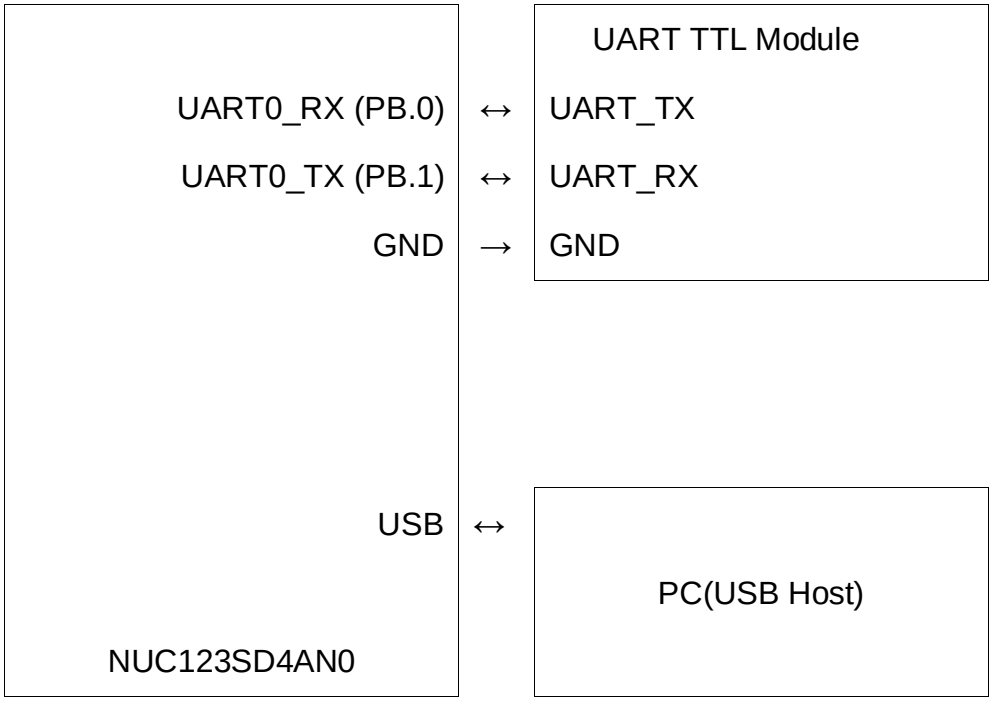
3 Software and Hardware Environment

Software Environment

- BSP version
  - NUC123 Series BSP CMSIS V3.01.001
- IDE version
  - Keil uVersion 5.26
- Tool for testing
  - Terminal tool, ex: putty(https://www.putty.org/)
  - Bus Hound(http://www.perisoft.net/bushound/)

Hardware Environment

- Circuit components
  - NuTiny-EVB-NUC123-LQFP64 v1.0 C
  - USB mini USB cable
  - USB-UART TTL
- Diagram



4 Directory Information

- EC_NUC123_USBD_HID_Vendor_Command_V1.00
  - Library — Sample code header and source files
    - CMSIS — Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
    - Device — CMSIS compliant device header file
    - StdDriver — All peripheral driver header and source files
  - SampleCode
    - ExampleCode — Source file of example code
      - KEIL — KEIL project file

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information(section 4) and double click NUC123_USBD_HID_Vendor_Command.uvproj.
2. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
3. Enter debug mode
   a. Run

6 Revision History

Date	Revision	Description
Aug. 1, 2019	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.