Using TinyUSB to Implement Mouse and Keyboard

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the M463 series microcontroller (MCU) to implement mouse and keyboard functions based on the TinyUSB library.
BSP Version	M460_Series_BSP_CMSIS_V3.00.004
Hardware	NuMaker-M463KG 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 and microprocessor 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. Overview

This example code demonstrates how to implement USB HID (Human Interface Device) mouse and keyboard functions using the Nuvoton M463 series microcontroller (MCU) based on the TinyUSB library. Through this example, users can learn how to initialize the MCU, handle HID class mouse and keyboard events. This example is suitable for developers who have a basic understanding of the USB HID protocol and wish to quickly build a prototype application. The application utilizes the M463's USB High-Speed (USB HS) interface and GPIOs to simulate mouse and keyboard inputs, and uses TinyUSB's API to handle HID data transmission.

1.1 Principle

The M463 series features a USB High-Speed Device controller and supports USB 2.0 device functionality. In this example, the TinyUSB library is used to initialize the USB device and configure descriptors, allowing the evaluation board to be recognized as a HID-class mouse and keyboard when connected to a computer.

TinyUSB is an open-source, cross-platform USB Host/Device stack designed for embedded system development. It aims to provide a lightweight, portable, and easy-to-use USB protocol implementation. The architecture of TinyUSB is modular and consists of the following main components:

- **Device Stack**: Supports USB device functionality and includes various USB classes such as CDC (Virtual COM Port), MSC (Mass Storage), HID (Human Interface Device), MIDI, DFU, and more.
- **Host Stack**: Provides USB host functionality and supports interaction with external devices such as USB flash drives, mice, etc.
- **Class Drivers**: Handles each USB class by parsing data and managing control flows.
- **Board Support Layer**: Interfaces TinyUSB with the USB controller of the target MCU and implements the low-level data transfer logic.

In this example, the HID class driver from the Device Stack is used to enable the M463 series to function as a USB mouse and keyboard when connected to a PC. TinyUSB provides a unified and intuitive API, such as tud_hid_keyboard_report() and tud_hid_mouse_report(), which allows developers to easily transmit HID data without the need to manage the underlying USB protocol details.

This example uses tud_hid_keyboard_report() to simulate keyboard data transmission:

bool tud_hid_keyboard_report(uint8_t instance, uint8_t report_id, uint8_t modifier, uint8_t const keycode[6]);

Parameter description:

- **instance**: HID interface instance number, typically set to 0.

- **report_id**: Set to the corresponding value if Report ID is enabled in the HID descriptor; otherwise, set to 0.
- **modifier**: Modifier keys such as LEFT SHIFT, RIGHT CTRL, etc. Set to 0 if no modifier is used.
- **keycode[6]**: An array of up to 6 keycodes to be sent in one report. Unused slots must be filled with 0.

Additionally, this example also uses tud_hid_mouse_report() to simulate mouse data transmission:

bool tud_hid_mouse_report(uint8_t instance, uint8_t report_id, uint8_t buttons, int8_t x, int8_t y, int8_t vertical, int8_t horizontal);

Parameter description:

- **instance**: HID interface instance number, typically set to 0.
- **report_id**: Set to the corresponding value if Report ID is enabled in the HID descriptor; otherwise, set to 0.
- **buttons**: Mouse button status. Supports up to 8 bits (bit 0~2 represent left, middle, and right buttons, respectively).
- **x**: Relative movement along the X-axis (range: -127 to 127).
- **y**: Relative movement along the Y-axis (range: -127 to 127).
- **vertical**: Vertical scroll wheel movement (positive or negative value).
- **horizontal**: Horizontal scroll wheel movement (positive or negative value).

1.2 Demo Result

After compiling and programming the example code, relevant messages can be viewed via the UART serial port.

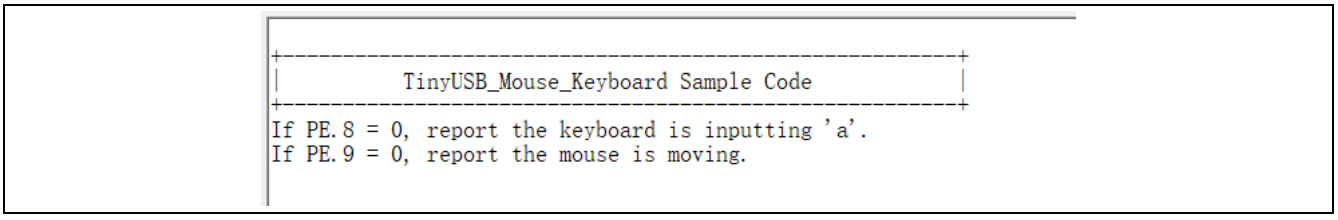


Figure 1-1 UART Message

Once the USB HS port is connected to the PC, you can press BTN0 (PE.8) to simulate a keyboard input of the ‘a’ key, based on the printed messages.

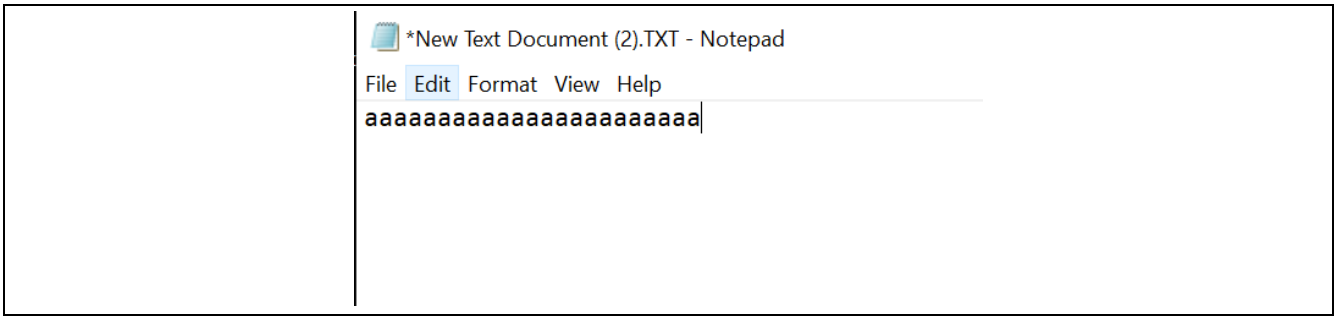


Figure 1-2 Keyboard Input

Pressing BTN1 (PE.9) simulates mouse movement toward the lower-right corner.

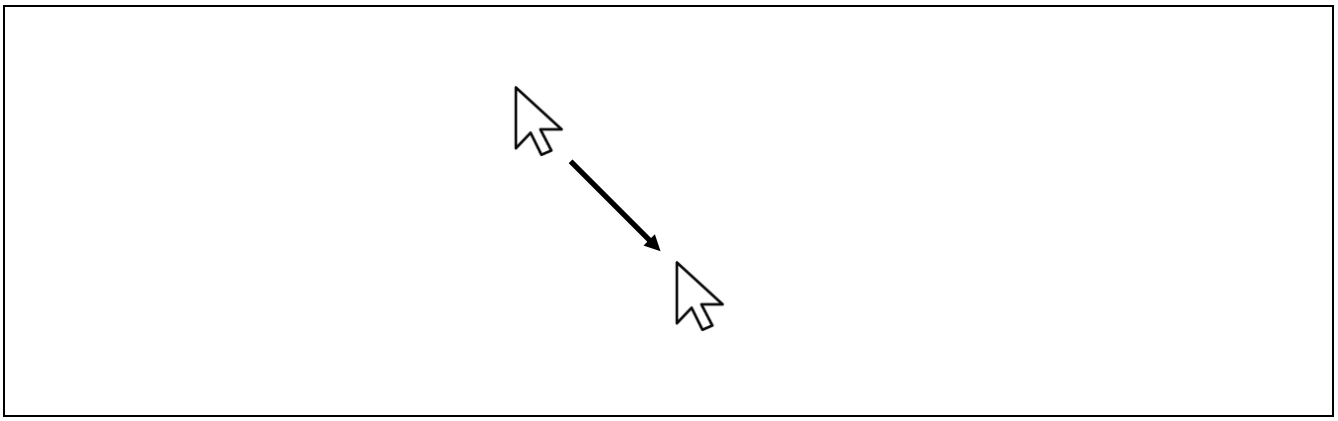


Figure 1-3 Mouse Movement

2. Code Description

Initialize the MCU and call the functions of the TinyUSB library. The code is located in *main.c*.

```
void SYS_Init(void)
{
    uint32_t volatile i;

    /* Unlock protected registers */
    SYS_UnlockReg();

    /*---------------------------------------------------*/
    /* Init System Clock                                 */
    /*---------------------------------------------------*/

    /* Enable HIRC and HXT clock */
    CLK_EnableXtalRC(CLK_PWRCTL_HIRCEN_Msk | CLK_PWRCTL_HXTEN_Msk);

    /* Wait for HIRC and HXT clock ready */
    CLK_WaitClockReady(CLK_STATUS_HIRCSTB_Msk | CLK_STATUS_HXTSTB_Msk);

    /* Set PCLK0 and PCLK1 to HCLK/2 */
    CLK->PCLKDIV = (CLK_PCLKDIV_APB0DIV_DIV2 | CLK_PCLKDIV_APB1DIV_DIV2);

    /* Set core clock to 200MHz */
    CLK_SetCoreClock(FREQ_200MHZ);

    /* Enable all GPIO clock */
    CLK->AHBCLK0 |= CLK_AHBCLK0_GPACKEN_Msk | CLK_AHBCLK0_GPBCKEN_Msk |
CLK_AHBCLK0_GPCCKEN_Msk | CLK_AHBCLK0_GPDCKEN_Msk |
                    CLK_AHBCLK0_GPECKEN_Msk | CLK_AHBCLK0_GPFCKEN_Msk |
CLK_AHBCLK0_GPGCKEN_Msk | CLK_AHBCLK0_GPHCKEN_Msk;
    CLK->AHBCLK1 |= CLK_AHBCLK1_GPICKEN_Msk | CLK_AHBCLK1_GPJCKEN_Msk;

    /* Enable UART0 module clock */
    CLK_EnableModuleClock(UART0_MODULE);

    /* Select UART0 module clock source as HIRC and UART0 module clock divider as 1 */
    CLK_SetModuleClock(UART0_MODULE, CLK_CLKSEL1_UART0SEL_HIRC, CLK_CLKDIV0_UART0(1));

    /* Select HSUSBD */
    SYS->USBPHY &= ~SYS_USBPHY_HSUSBROLE_Msk;

    /* Enable USB PHY */
    SYS->USBPHY = (SYS->USBPHY & ~(SYS_USBPHY_HSUSBROLE_Msk | SYS_USBPHY_HSUSBACT_Msk)) |
SYS_USBPHY_HSUSBEN_Msk;
    for(i = 0; i < 0x1000; i++);   // delay > 10 us
    SYS->USBPHY |= SYS_USBPHY_HSUSBACT_Msk;

    /* Enable HSUSBD module clock */
    CLK_EnableModuleClock(HSUSBD_MODULE);

    /*---------------------------------------------------*/
    /* Init I/O Multi-function                           */
    /*---------------------------------------------------*/

    /* Set multi-function pins for UART0 RXD and TXD */
    SET_UART0_RXD_PB12();
    SET_UART0_TXD_PB13();

    /* Lock protected registers */
    SYS_LockReg();
}

/*---------------------------------------------------*/
```

```
/*  Main Function                                          */
/*-------------------------------------------------------*/
int32_t main(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();

    /* Init UART0 to 115200-8n1 for print message */
    UART_Open(UART0, 115200);

    printf("\n");
    printf("+------------------------------------------------------------+\n");
    printf("|          TinyUSB_Mouse_Keyboard Sample Code                |\n");
    printf("+------------------------------------------------------------+\n");
    printf("If PE.8 = 0, report the keyboard is inputting 'a'.\n");
    printf("If PE.9 = 0, report the mouse is moving.\n");

    /* Set PE.8 & PE.9 as Quasi-bidirectional mode */
    GPIO_SetMode(PE, BIT8| BIT9, GPIO_MODE_QUASI);
    PE8 = 1;
    PE9 = 1;

    // 1ms tick timer
    SysTick_Config(200000000 / 1000);

    // init device stack on configured roothub port
    tud_init(BOARD_TUD_RHPORT);

    while(1) {
        tud_task(); // tinyusb device task

        hid_task();
    }
}
```

Use the BTN0 (PE.8) and BTN1 (PE.9) buttons to trigger mouse and keyboard events. When BTN0 (PE.8) is pressed, the function tud_hid_n_keyboard_report() is called to send keyboard event data. When BTN1 (PE.9) is pressed, the function tud_hid_n_mouse_report() is called to send mouse event data. The code is located in *main.c*.

```
    ……
void hid_task(void)
{
    // Poll every 10ms
    const uint32_t interval_ms = 10;
    static uint32_t start_ms = 0;

    if(board_millis() - start_ms < interval_ms) return;   // not enough time
    start_ms += interval_ms;

    /*------------- Keyboard -------------*/
    if(tud_hid_n_ready(ITF_KEYBOARD)) {
        // use to avoid send multiple consecutive zero report for keyboard
        static bool has_key = false;

        if(PE8==0) {
            uint8_t keycode[6] = { 0 };
            keycode[0] = HID_KEY_A;
```

```
            tud_hid_n_keyboard_report(ITF_KEYBOARD, 0, 0, keycode);

            has_key = true;
        } else {
            // send empty key report if previously has key pressed
            if(has_key) tud_hid_n_keyboard_report(ITF_KEYBOARD, 0, 0, NULL);
            has_key = false;
        }
    }

    /*------------- Mouse -------------*/
    if(tud_hid_n_ready(ITF_MOUSE)) {
        if(PE9==0) {
            int8_t const delta = 5;

            // no button, right + down, no scroll pan
            tud_hid_n_mouse_report(ITF_MOUSE, 0, 0x00, delta, delta, 0, 0);
        }
    }
}
```

The Board Support Layer code handles the interrupt service routines for sending and receiving USB packets through the USB controller. It will pass the packet data to the TinyUSB library. The code is located in *dcd_M463.c*.

```
void dcd_int_handler(uint8_t rhport)
{
    ……
    if(IrqStL & (HSUSBD_GINTSTS_EPAIF_Msk | HSUSBD_GINTSTS_EPBIF_Msk |
HSUSBD_GINTSTS_EPCIF_Msk | HSUSBD_GINTSTS_EPDIF_Msk | HSUSBD_GINTSTS_EPEIF_Msk |
HSUSBD_GINTSTS_EPFIF_Msk | HSUSBD_GINTSTS_EPGIF_Msk | HSUSBD_GINTSTS_EPHIF_Msk |
HSUSBD_GINTSTS_EPIIF_Msk | HSUSBD_GINTSTS_EPJIF_Msk | HSUSBD_GINTSTS_EPKIF_Msk |
HSUSBD_GINTSTS_EPLIF_Msk)) {
        /* service PERIPH_EPA through PERIPH_EPL */
        for(ep_index = PERIPH_EPA, mask = HSUSBD_GINTSTS_EPAIF_Msk, xfer =
&xfer_table[PERIPH_EPA], ep = &HSUSBD->EP[PERIPH_EPA]; ep_index < PERIPH_MAX_EP;
ep_index++, mask <<= 1, xfer++, ep++) {
            if(IrqStL & mask) {
                uint8_t const ep_addr = xfer->ep_addr;
                bool const out_ep = !(ep_addr & TUSB_DIR_IN_MASK);
                uint32_t ep_state = ep->EPINTSTS & ep->EPINTEN;

                if(out_ep) {
                    uint16_t const available_bytes = ep->EPDATCNT &
HSUSBD_EPDATCNT_DATCNT_Msk;
                    /* copy the data from the PC to the previously provided buffer */
                    for(int count = 0; (count < available_bytes) && (xfer-
>out_bytes_so_far < xfer->total_bytes); count++, xfer->out_bytes_so_far++) {
                        *xfer->data_ptr++ = ep->EPDAT_BYTE;
                    }

                    /* when the transfer is finished, alert TinyUSB; otherwise, continue
accepting more data */
                    if((xfer->total_bytes == xfer->out_bytes_so_far) || (available_bytes <
xfer->max_packet_size)) {
```

```
                        dcd_event_xfer_complete(0, ep_addr, xfer->out_bytes_so_far,
XFER_RESULT_SUCCESS, true);
                    }

                } else if(ep_state & HSUSBD_EPINTSTS_BUFEMPTYIF_Msk) {
                    /* send any remaining data */
                    dcd_userEP_in_xfer(xfer, ep);
                } else if(ep_state & HSUSBD_EPINTSTS_TXPKIF_Msk) {
                    /* alert TinyUSB that we've finished */
                    dcd_event_xfer_complete(0, ep_addr, xfer->total_bytes,
XFER_RESULT_SUCCESS, true);
                    ep->EPINTEN = 0;
                }

                ep->EPINTSTS = ep_state;
            }
        }
    }
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M460_Series_BSP_CMSIS_V3.00.004
- IDE version
  - Keil uVision 5.38

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M463KG V1.0
- Pin Connect
  - Connect the UART0 TX (PB.13) pin to the PC UART RX to show the message.
  - Connect the USB cable from the PC to the USB HS port on the evaluation board to use it as a TinyUSB mouse and keyboard.

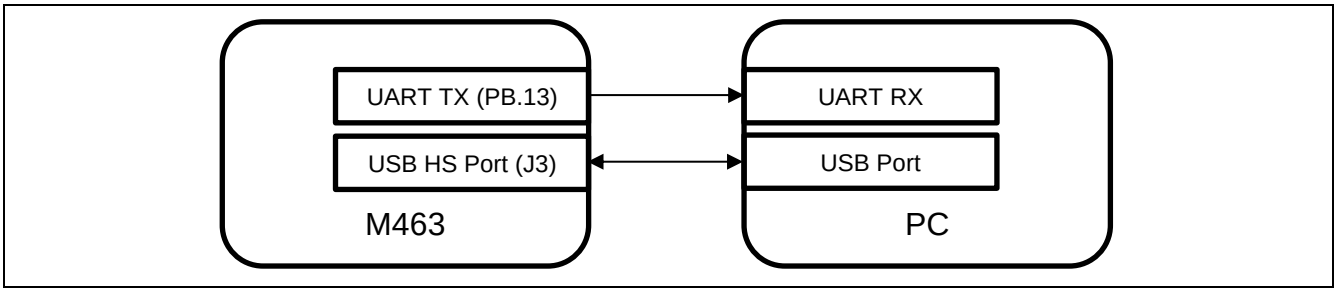


Figure 3-4 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M463_TinyUSB_Mouse_Keyboard_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	
Project	Source file of example code
ThirdParty	
tinyusb-master	An open-source USB Host/Device stack for embedded system. Its official website is: https://www.tinyusb.org/

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *TinyUSB_Mouse_Keyboard.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.04.04	1.00	Initial version.

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.