M032 USB HID to GPIO Bridge

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the M032 series microcontroller (MCU) to implement the USB HID to GPIO bridge function.
BSP Version	M031_Series_BSP_CMSIS_V3.06.000
Hardware	NuMaker-M032TC_V1.2

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 use the NuMicro M032 series microcontroller to implement a USB HID device, which communicates with the PC through a customized HID report protocol and provides a virtual COM port for users to transmit and receive data.

Through USB HID, the PC sends write or read commands to control the pin mode (Input/Output) and logic state (High/Low) of PB.0, PB.1, PB.2, and PB.3 on the M032 series MCU.

1.1 Principle

This example exchanges data through the Interrupt OUT endpoint (receiving commands from the PC) and the Interrupt IN endpoint (returning data to the PC) of USB HID. Since the HID class does not require an additional driver under Windows, a user control interface tool, *M032_HID_Controller.exe*, is also provided. The tool sends reports through the HID channel to control the device and displays the GPIO status in real time, which facilitates development and verification.

For GPIO Configuration, in this example, four pins, PB.0 through PB.3: PB.0 (Pin0), PB.1 (Pin1), PB.2 (Pin2), and PB.3 (Pin3), are mapped for HID communication. Upon system startup, these pins are configured in output mode.

When the USB interface receives a HID_CMD_WRITE or HID_CMD_READ command, the HID_GetOutReport function is triggered to parse the data packet, which subsequently calls the USB_to_Write_GPIO or USB_to_Read_GPIO function. The HID data description is shown in Figure 1-1 .

HID Command Write

	Report ID	CMD_WRITE	CMD_SIZE	StartPage	Pages	Signature	Checksum	Padding	
(MSB)	0x00	0xC3	0x0E	0x20	0x01	0x43444948	0x020A	0x00	(LSB)

HID Command Read

	Report ID	CMD_WRITE	CMD_SIZE	StartPage	Pages	Signature	Checksum	Padding	
(MSB)	0x00	0xD2	0x0E	0x20	0x01	0x43444948	0x0219	0x00	(LSB)

HID Data

	Pin0_Conf	Pin1_Conf	Pin2_Conf	Pin3_Conf	Pin0_WR	Pin1_WR	Pin2_WR	Pin3_WR	
(MSB)	0x00 : GPIO	0x00 : GPIO	0x00 : GPIO	0x00 : GPIO	0x00 : Read 0x01 : Write	0x00 : Read 0x01 : Write	0x00 : Read 0x01 : Write	0x00 : Read 0x01 : Write	
	PIN0_DATA	**PIN1_DATA**	**PIN2_DATA**	**PIN3_DATA**	**PIN0_Write**	**PIN1_Write**	**PIN2_Write**	**PIN3_Write**	
	0x00 : Low 0x01 : High	0x00 : Low 0x01 : High	0x00 : Low 0x01 : High	0x00 : Low 0x01 : High	0x00 : Low 0x01 : High	0x00 : Low 0x01 : High	0x00 : Low 0x01 : High	0x00 : Low 0x01 : High	(LSB)

Figure 1-1 HID Data Description

1.2 Demo Result

After compiling and programming, the example code is used in conjunction with the *M032_HID_Controller.exe*, as shown in Figure 1-2. By clicking the **Search** button, available HID devices are scanned. Once the example device (VID = 0x0416, PID = 0xDC00) is identified, it should be selected and the **Open** button is clicked to establish the connection.

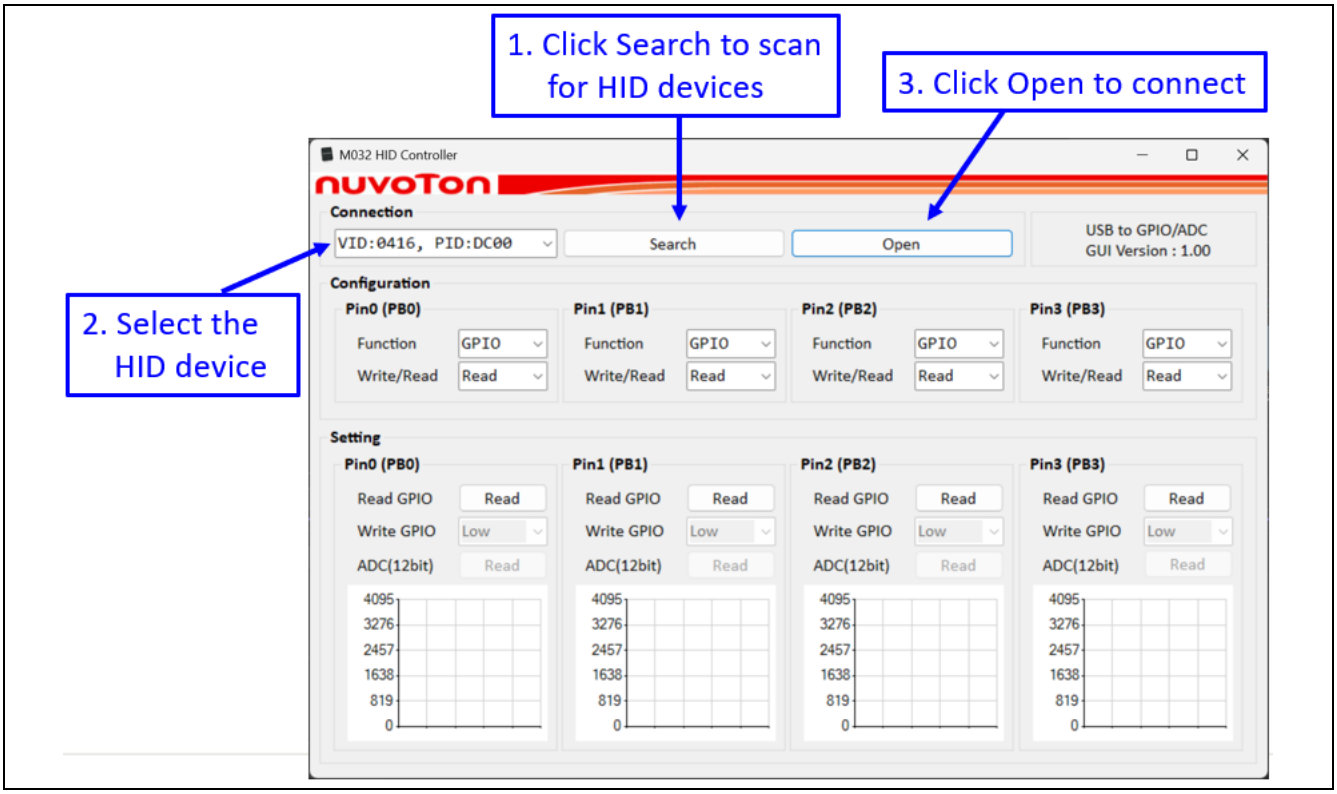

Figure 1-2 M032 HID Controller Connection Description

As shown in Figure 1-3, the GPIO Write operation is performed by selecting the Write function in the user interface and choosing the desired voltage level from the High/Low dropdown menu. This sets the selected pin to either a high or low voltage level.

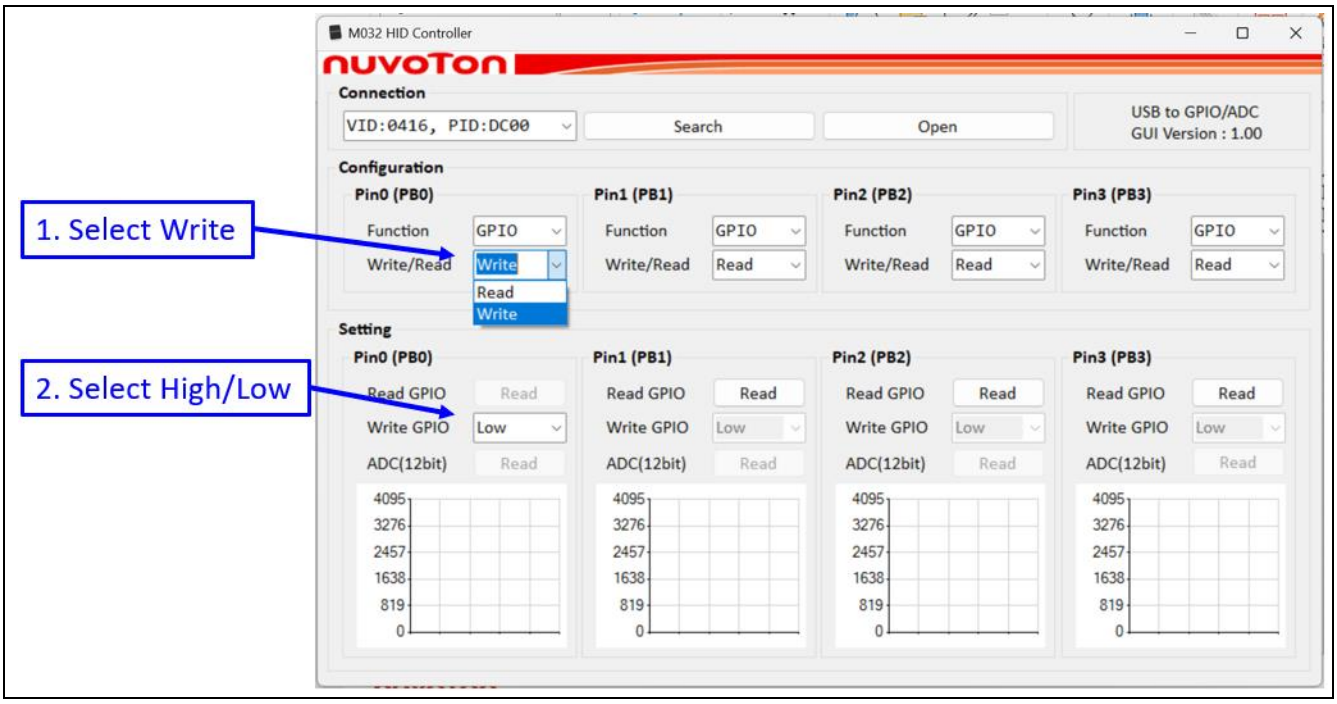

Figure 1-3 GPIO Write Operation

As shown in Figure 1-4, the GPIO Read operation is performed by selecting the Read function and clicking the **Read** button. The current voltage level of the selected pin is then read and displayed immediately on the interface.

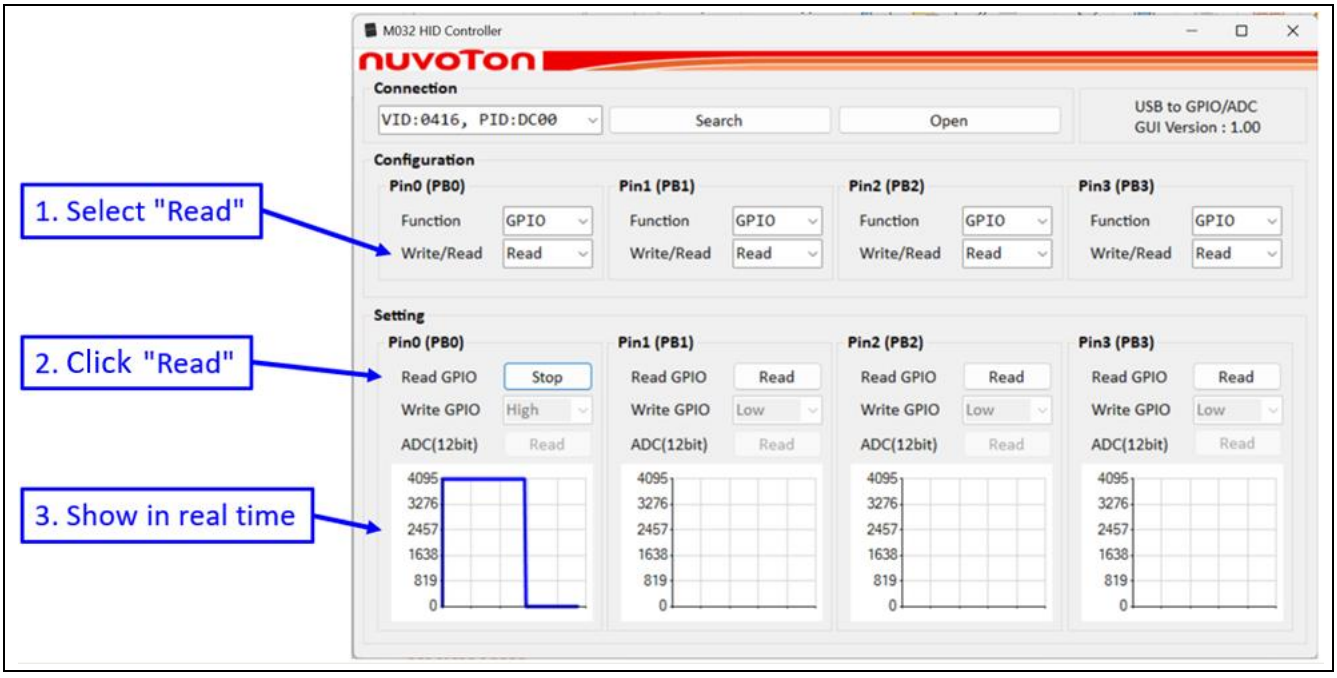

Figure 1-4 GPIO Read Operation

2. Code Description

System initialization and the activation of GPIO and USB peripherals. The code is located in *main.c*.

```
/*---------------------------------------------------------------------------------------------------------*/
/*  Main Function                                                                                          */
/*---------------------------------------------------------------------------------------------------------*/
int32_t main(void)
{
#if CRYSTAL_LESS
    uint32_t u32TrimInit;
#endif

    /* 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("|     NuMicro USB HID and GPIO Transfer Sample Code          |\n");
    printf("+------------------------------------------------------------+\n");
    printf("PID is 0xDC00\n");

    /* Configure PB.0 ~ PB.3 as Output mode */
    GPIO_SetMode(PB, BIT0, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PB, BIT1, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PB, BIT2, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PB, BIT3, GPIO_MODE_OUTPUT);

    /* Open USB controller */
    USBD_Open(&gsInfo, HID_ClassRequest, NULL);

    /* Endpoint configuration */
    HID_Init();

    /* Start USB device */
    USBD_Start();

    NVIC_EnableIRQ(USBD_IRQn);

    NVIC_EnableIRQ(UART02_IRQn);

#if CRYSTAL_LESS
    /* Backup default trim */
    u32TrimInit = M32(TRIM_INIT);
#endif

    /* Clear SOF */
    USBD->INTSTS = USBD_INTSTS_SOFIF_Msk;
```

```
    while (1)
    {
#if CRYSTAL_LESS

        /* Start USB trim if it is not enabled. */
        if ((SYS->HIRCTRIMCTL & SYS_HIRCTRIMCTL_FREQSEL_Msk) != 1)
        {
            /* Start USB trim only when SOF */
            if (USBD->INTSTS & USBD_INTSTS_SOFIF_Msk)
            {
                /* Clear SOF */
                USBD->INTSTS = USBD_INTSTS_SOFIF_Msk;

                /* Re-enable crystal-less */
                SYS->HIRCTRIMCTL = 0x01;
                SYS->HIRCTRIMCTL |= SYS_HIRCTRIMCTL_REFCKSEL_Msk;
            }
        }

        /* Disable USB Trim when error */
        if (SYS->HIRCTRIMSTS & (SYS_HIRCTRIMSTS_CLKERIF_Msk |
             SYS_HIRCTRIMSTS_TFAILIF_Msk))
        {
            /* Init TRIM */
            M32(TRIM_INIT) = u32TrimInit;

            /* Disable crystal-less */
            SYS->HIRCTRIMCTL = 0;

            /* Clear error flags */
            SYS->HIRCTRIMSTS = SYS_HIRCTRIMSTS_CLKERIF_Msk | SYS_HIRCTRIMSTS_TFAILIF_Msk;

            /* Clear SOF */
            USBD->INTSTS = USBD_INTSTS_SOFIF_Msk;
        }

#endif

        /* Enter power down when USB suspend */
        if (g_u8Suspend)
            PowerDown();

        /* USB HID data transfer */
        USB_HID_TransferData();
    }
}
```

This function fills the current GPIO pin states into a buffer, which are subsequently transmitted back to the PC via USB. The code is located in *GPIO_and_HID_Transfer.c*.

```
void USB_to_Read_GPIO(uint8_t *data)
{
    /* GPIO Input */
    data[8]  = PIN0;
    data[9]  = PIN1;
    data[10] = PIN2;
```

```
    data[11] = PIN3;
}
```

This function is responsible for parsing data packets received from the USB interface and configuring the GPIO modes and output states according to the packet content. The code is located in *GPIO_and_HID_Transfer.c*.

```
void USB_to_Write_GPIO(uint8_t *data)
{
    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Pin Config */
    CONF_S.Pin0_Conf = data[0];
    CONF_S.Pin1_Conf = data[1];
    CONF_S.Pin2_Conf = data[2];
    CONF_S.Pin3_Conf = data[3];

    /* Write & Read */
    WR_S.Pin0_WR = data[4];
    WR_S.Pin1_WR = data[5];
    WR_S.Pin2_WR = data[6];
    WR_S.Pin3_WR = data[7];

    /* High & Low */
    HL_S.Pin0_HL = data[12];
    HL_S.Pin1_HL = data[13];
    HL_S.Pin2_HL = data[14];
    HL_S.Pin3_HL = data[15];

    /* Pin 0 config (PB0) */
    switch (CONF_S.Pin0_Conf)
    {
        case 0:
            switch (WR_S.Pin0_WR)
            {
                case 0: // GPIO & Read
                    GPIO_SetMode(PB, BIT0, GPIO_MODE_INPUT);
                    break;

                case 1: // GPIO & Write
                    GPIO_SetMode(PB, BIT0, GPIO_MODE_OUTPUT);

                    switch (HL_S.Pin0_HL)
                    {
                        case 0: // Low
                            PIN0 = 0;
                            break;

                        case 1: // High
                            PIN0 = 1;
                            break;
                    }
                    break;
            }
            break;
```

```
    case 1:
        break;
}

/* Pin 1 config (PB1) */
switch (CONF_S.Pin1_Conf)
{
    case 0:
        switch (WR_S.Pin1_WR)
        {
            case 0: // GPIO & Read
                GPIO_SetMode(PB, BIT1, GPIO_MODE_INPUT);
                break;

            case 1: // GPIO & Write
                GPIO_SetMode(PB, BIT1, GPIO_MODE_OUTPUT);

                switch (HL_S.Pin1_HL)
                {
                    case 0: // Low
                        PIN1 = 0;
                        break;

                    case 1: // High
                        PIN1 = 1;
                        break;
                }
                break;
        }
        break;

    case 1:
        break;
}

/* Pin 2 config (PB2) */
switch (CONF_S.Pin2_Conf)
{
    case 0:
        switch (WR_S.Pin2_WR)
        {
            case 0: // GPIO & Read
                GPIO_SetMode(PB, BIT2, GPIO_MODE_INPUT);
                break;

            case 1: // GPIO & Write
                GPIO_SetMode(PB, BIT2, GPIO_MODE_OUTPUT);

                switch (HL_S.Pin2_HL)
                {
                    case 0: // Low
                        PIN2 = 0;
                        break;

                    case 1: // High
```

```
                            PIN2 = 1;
                            break;
                    }
                    break;
            }
            break;

        case 1:
            break;
    }

    /* Pin 3 config (PB3) */
    switch (CONF_S.Pin3_Conf)
    {
        case 0:
            switch (WR_S.Pin3_WR)
            {
                case 0: // GPIO & Read
                    GPIO_SetMode(PB, BIT3, GPIO_MODE_INPUT);
                    break;

                case 1: // GPIO & Write
                    GPIO_SetMode(PB, BIT3, GPIO_MODE_OUTPUT);

                    switch (HL_S.Pin3_HL)
                    {
                        case 0: // Low
                            PIN3 = 0;
                            break;

                        case 1: // High
                            PIN3 = 1;
                            break;
                    }
                    break;
            }
            break;

        case 1:
            break;
    }

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

This function is responsible for receiving USB HID data from the PC, identifying the command type, and executing the corresponding actions. The code is located in *GPIO_and_HID_Transfer.c*.

```
void HID_GetOutReport(uint8_t *pu8EpBuf, uint32_t u32Size)
{
    uint32_t i;
    uint8_t  u8Cmd;
    uint32_t u32StartPage;
    uint32_t u32Pages;
```

```
    uint32_t u32PageCnt;

    /* Get command information */
    u8Cmd        = gCmd.u8Cmd;
    u32StartPage = gCmd.u32Arg1;
    u32Pages     = gCmd.u32Arg2;
    u32PageCnt   = gCmd.u32Signature; /* The signature word is used to count pages */

    /* Check if it is in the data phase of write command */
    if ((u8Cmd == HID_CMD_WRITE) && (u32PageCnt < u32Pages))
    {
        /* Process the data phase of write command */

        /* Get data from HID OUT */
        USBD_MemCopy(&g_u8PageBuff[g_u32BytesInPageBuf], pu8EpBuf, EP6_MAX_PKT_SIZE);
        g_u32BytesInPageBuf += EP6_MAX_PKT_SIZE;

        /* The HOST must make sure the data is PAGE_SIZE alignment */
        if (g_u32BytesInPageBuf >= PAGE_SIZE)
        {
            printf("Writing page %d\n", u32StartPage + u32PageCnt);
            /* TODO: We should program received data to storage here */
            memcpy(g_u8TestPages + u32PageCnt * PAGE_SIZE, g_u8PageBuff,
                   sizeof(g_u8PageBuff));
            u32PageCnt++;

            /* Write command complete! */
            if (u32PageCnt >= u32Pages)
            {
                u8Cmd = HID_CMD_NONE;
                printf("Write command complete.\n");

                /* Process the data phase of write command */
                USB_to_Write_GPIO(&g_u8TestPages[0]);
            }
            g_u32BytesInPageBuf = 0;
        }

        /* Update command status */
        gCmd.u8Cmd        = u8Cmd;
        gCmd.u32Signature = u32PageCnt;
    }
    else
    {
        /* Check and process the command packet */
        if (ProcessCommand(pu8EpBuf, sizeof(gCmd)))
        {
            printf("Unknown HID command!\n");
        }
    }
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M031_Series_BSP_CMSIS_V3.06.000
- IDE version
  - Keil uVision 5.41

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M032TC_V1.2
- Pin connect
  - The USB interface of the NuMaker-M032TC evaluation board is connected to the USB port of a PC to show the execution results of the example code.

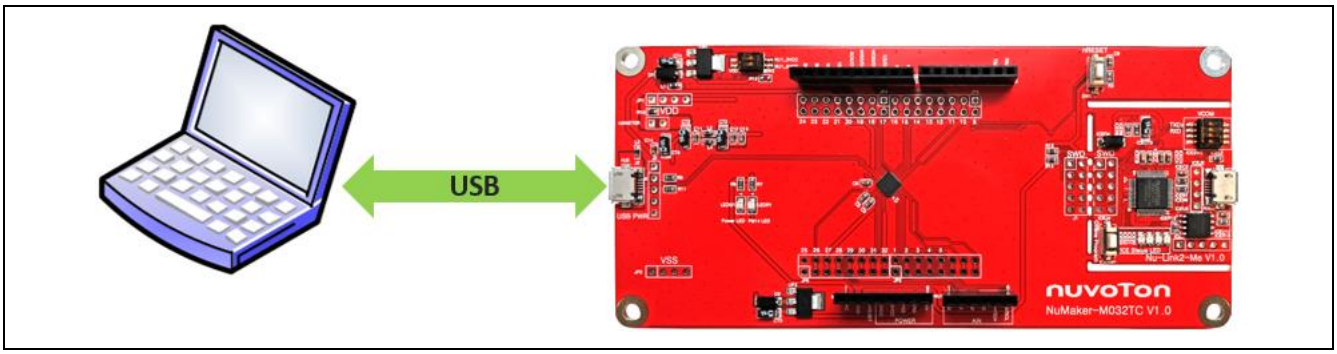

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M032_USB_HID_to_GPIO_Bridge_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
WindowsTool	User Control GUI Tool

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 *USB_HID_to_GPIO_Bridge.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2026.01.27	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.