M253 透過 USB 傳送與接收 CAN 協議

NuMicro® 32 位元系列微控制器範例代碼介紹

文件資訊

應用簡述	本範例代碼使用 M253 系列使用 USB 與 MCU 溝通，並傳送與接收 CAN 協議。
BSP 版本	M253_Series_BSP_CMSIS_V3.00.010
開發平台	NuMaker-M253LE3AE-V1.1

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. 概述

本範例代碼展示如何透過 M253 系列的 USB 介面與主機 (Host) 溝通，藉此傳送與接收 CAN 協議資料，並提供操作方式於 Linux 環境。在 M253 接收到 Host 的指令後，於 CAN 腳位產生訊號，並利用 Numaker-M253LE 上帶的 CAN 收發器(ADM3050E)將 CAN 的邏輯訊號轉換成 CANH/CANL 的差動訊號與其他 CAN 裝置進行通訊。接收訊號同理，在收到其他裝置的 CAN 訊號後，經收發器轉換給 MCU 後，會回報 Host 收到的指令內容。

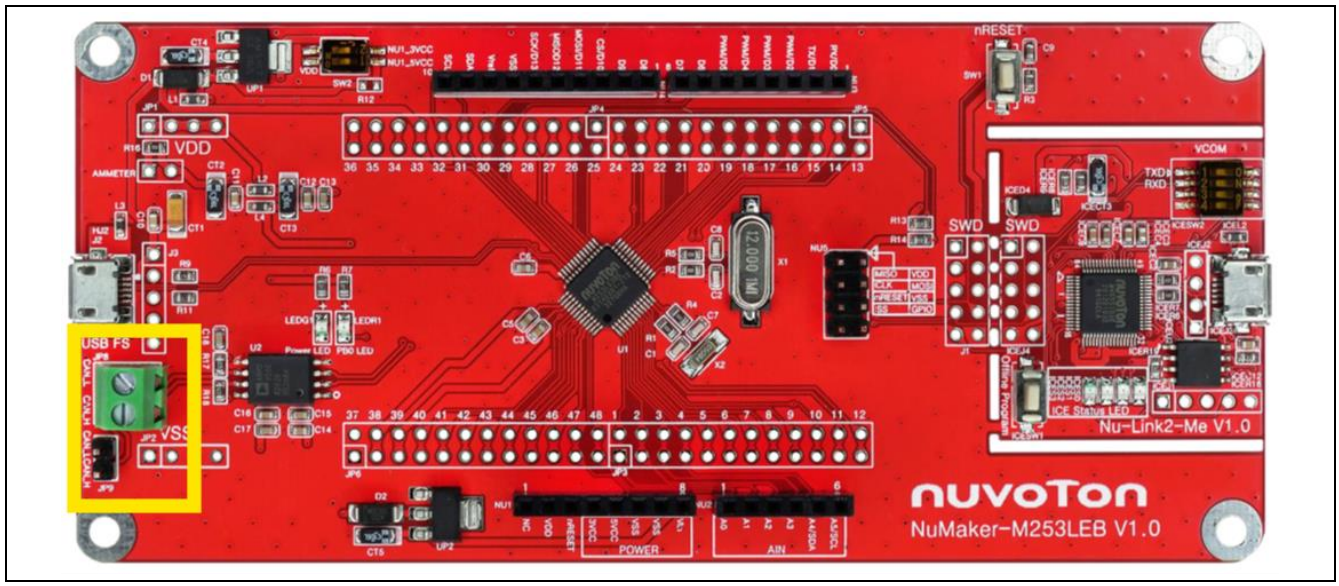


圖 1-1 NuMaker-M253LE 開發板 CAN 腳位位置

1.1 原理

在 Linux 底下，原生支援 SocketCAN。它由 Linux 核心維護，會將 CAN 匯流排視為一個網路設備，使應用程式可以用標準的 BSD(Berkeley Software Distribution) Socket API 進行通訊，不需要額外撰寫底層驅動。

Linux 中有現成的 CAN 工具，本範例使用 can-utils 中的指令來對 CAN 進行操作，使用前需要先安裝 can-utils(安裝指令: sudo apt-get install can-utils)。

會使用到的指令有:

candump: 監測並顯示當前在 CAN 匯流排上的所有訊息。

cansend: 傳送一筆 CAN frame，在此應用中，鎖定上限為 8 bytes。

除了通訊指令以外，在使用前需要先和設備進行連接並設定通訊速度，會使用到的指令如下:

ip link set <device> down: 更改當前裝置狀態為 down。此指令為非必要指令，在設定 can 裝置的 bitrate(下行指令)時裝置必須要為 down 狀態，因此用這條指令把裝置狀態設為 down。

ip link set <device> type can bitrate 500000: 更改當前 CAN 裝置的 bitrate。該指令無法在裝置狀態為 up 時使用。

ip link set <device> up: 更改當前裝置狀態為 up。在設定完 bitrate 後啟動。

上述指令會有權限不足問題，建議加上 sudo。而<device> 會根據裝置的名稱而有所不同(can0, can1…)，可以透過以下指令確認:

ip link list type can: 列出當前有連接在 Host 的 CAN 裝置。

圖 1-2 使用兩個 M253 開發板，一個為裝置 can0，另一個為裝置 can1。

由 can1 傳送指令給 can0，ID:0x611，Packet:0x1122334455667788。

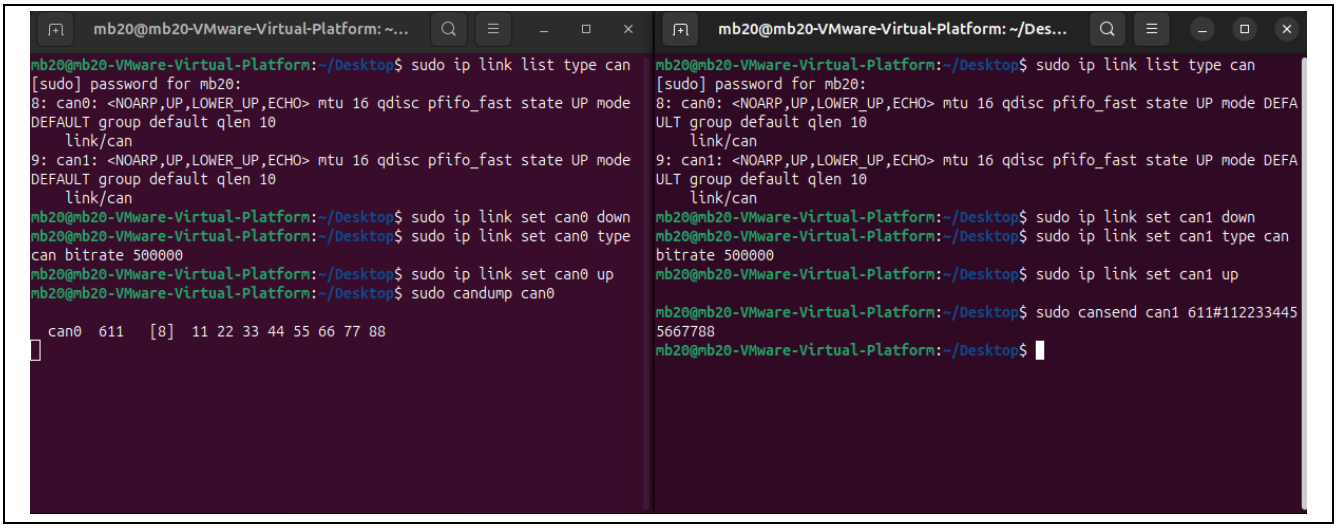


圖 1-2 Linux 環境下用兩個 M253 裝置以 CAN 做通訊

1.2 執行結果

本範例代碼執行編譯和燒錄的結果如圖 1-3 和圖 1-4 所示。

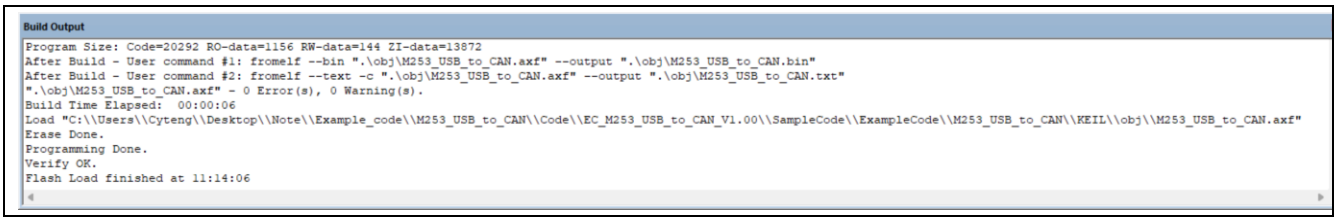


圖 1-3 執行編譯和燒錄結果(Keil)



圖 1-4 執行編譯和燒錄結果(VScode)

2. 代碼介紹

系統初始化相關程式碼位於 *main.c*。在完成系統與 USB 初始化後，程式將進一步設定 CAN 模組以及 MCU 內部的 Timer 計數機制。

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

    /* Init system and multi-function I/O */
    SYS_Init();

    /* Init UART MFP */
    UartDebugInit();
```

開啟 USB 功能，並且設定回傳給 Host 的資料，最後開啟 USB 中斷功能。

```
    /* Open USB controller */
    USBD_Open(&gsInfo, VCOM_ClassRequest, NULL);
    USBD_SetVendorRequest(VCOM_VendorRequest);
    /* Endpoint configuration */
    VCOM_Init();
    /* Start USB device */
    USBD_Start();
    NVIC_EnableIRQ(USBD_IRQn);
```

接續前述初始化流程，程式會進行 CAN 相關指令與 MCU 內部 Timer 計數機制的初始化。

```
    pcan_timestamp_init();
```

函式 pcan_timestamp_init 內容，代碼位於 *pcanpro_timestamp.c*。此函式透過啟用 Timer0 進行系統時間計數，並以 1 毫秒為週期觸發 Timer0 中斷，更新系統時間相關變數，作為後續通訊時間計算。

```
void pcan_timestamp_init(void)
{
    TIMER0->CTL =  TIMER_PERIODIC_MODE | 47;
    TIMER0->CMP = 1000;
    TIMER_EnableInt(TIMER0);
    NVIC_EnableIRQ(TMR0_IRQn);
    TIMER_Start(TIMER0);
}
```

中斷程式 TMR0_IRQHandler，代碼位於 *pcanpro_timestamp.c*。每次進入中斷程式後會遞增 g_au32TMRINTCount，該變數作為時間基準，提供通訊 timeout 判斷。

```
void TMR0_IRQHandler(void)
{
    if (TIMER_GetIntFlag(TIMER0))
```

```
    {
        /* Clear Timer0 time-out interrupt flag */
        TIMER_ClearIntFlag(TIMER0);

        g_au32TMRINTCount++;
    }
}
```

主程式內容，此處為讓系統的 Clock 可以利用 USB 校正，藉此讓誤差縮小。在 USB 的相關 Sample code 中都會有此段程式碼。詳細說明可以閱讀 M253 技術參考手冊 M253 Series - Nuvoton 第 6.2.8 IRC Auto Trim 章節。

```
        /* Start USB trim function if it is not enabled. */
        if ((SYS->HIRCTRIMCTL & SYS_HIRCTRIMCTL_FREQSEL_Msk) != 0x1)
        {
            /* Start USB trim only when USB signal arrived */
            if (USBD->INTSTS & USBD_INTSTS_SOFIF_Msk)
            {
                /* Clear SOF */
                USBD_CLR_INT_FLAG(USBD_INTSTS_SOFIF_Msk);

                /*
                    USB clock trim function:
                    HIRC Trimming with boundary function enhances robustility
                    and keeps HIRC in right frequency while receiving unstable USB signal
                */
                SYS->HIRCTRIMCTL = (0x1 << SYS_HIRCTRIMCTL_REFCKSEL_Pos)
                                 | (0x1 << SYS_HIRCTRIMCTL_FREQSEL_Pos)
                                 | (0x0 << SYS_HIRCTRIMCTL_LOOPSEL_Pos)
                                 | (0x1 << SYS_HIRCTRIMCTL_BOUNDEN_Pos)
                                 | (10  << SYS_HIRCTRIMCTL_BOUNDARY_Pos);
            }
        }

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

            /* Disable USB clock trim function */
            SYS->HIRCTRIMCTL = 0;

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

            /* Clear SOF */
            USBD_CLR_INT_FLAG(USBD_INTSTS_SOFIF_Msk);
        }
```

接續上面，g_u8Suspend 變數會在 USB 中斷內被更動，在收到 Suspend 時會進入 Powerdown，其餘時間會進入監測 Host 的 CAN 訊息。

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

函式 pcan_protocol_poll 內容，代碼位於 *pcanpro_fd_protocol.c*。為主要輪詢函式，負責 USB 與 CAN 資料的接收、封裝與回傳處理。

主要處理流程包含：

擷取當前系統時間-毫秒與微秒

兩個跟時間有關的函數 pcan_timestamp_millis 和 pcan_timestamp_us，皆是回傳 Timer0 中斷程式內更新的 g_au32TMRINTCount 變數以紀錄 MCU 當前的計數值，兩者僅回傳的單位不相同。

處理 CAN 接收與傳送資料

函式透過變數 g_u8RxFifO0CompleteFlag 判斷是否已觸發 CAN 接收中斷。CAN 中斷會發生於收到 CAN 的資料以及 Time out 的時候。收到資料的時候會在 write_manual_rx_frame 函式內將 CAN 收到的資料存放到結構體中等待呼叫。pcan_can_poll 則判斷是否收到要傳送 CAN 資料的指令。若有收到送指令，用 USB 向 Host 發出回傳資料。

將接收到的 CAN 資料透過 USB Bulk IN 回傳至 Host

flush data 部分為處理 CAN 收到資料後向 Host 透過 USB Bulk in 封包報告給 Host，透過 pcan_flush_data 函式作處理(僅有 Endpoint 2 的 Bulk in 會被處理，其他 Endpoint 會回傳錯誤)

週期性更新 CAN bus load 與 time sync 資訊

每 250ms 更新內部的參數，計算傳輸資料的時間占總匯流排的時間，並配置兩塊空間分別存放計算結果(指標 pbs)以及狀態參數(指標 pts, 參數舉例:上次 sync 時間)。在進入更新前，會先確認裝置是否已收到 Vendor Request 指令後繼續往下執行。

```
void pcan_protocol_poll(void)
{
    uint32_t ts_ms = pcan_timestamp_millis();
    uint32_t ts_us = pcan_timestamp_us();
    write_manual_rx_frame();
    pcan_can_poll();

    /* flush data */
    if (data_pos > 0)
    {
        /* endmark */
```

```
        *(uint32_t *)&data_buffer[data_pos] = 0x00000000;
        uint16_t flush_size = data_pos + 4;
        /* align to 64 */
        flush_size += (64 - 1);
        flush_size &= ~(64 - 1);
        int res = pcan_flush_data(&resp_fsm[1], data_buffer, flush_size);

        if (res)
        {
            data_pos = 0;
            pcan_device.last_time_flush = ts_us;
        }
    }
    /* timesync part */
    if (!pcan_device.can_drv_loaded)
        return;

    /* update bus load each 250ms */
    if ((ts_ms - pcan_device.last_bus_load_update) >= 250u)
    {
        pcan_device.last_bus_load_update = ts_ms;

        for (int i = 0; i < CAN_CHANNEL_MAX; i++)
        {
            uint32_t total_ns = pcan_device.can[i].tx_time_ns +
pcan_device.can[i].rx_time_ns;
            /* get bus in percents 0 - 100% */
            pcan_device.can[i].bus_load = total_ns / (250000000u / 100u);

            pcan_device.can[i].tx_time_ns = 0;
            pcan_device.can[i].rx_time_ns = 0;
        }
    }

    if ((ts_ms - pcan_device.last_time_sync) < 1000u)
        return;

    struct ucan_usb_ts_msg *pts = pcan_data_alloc_buffer(UCAN_USB_MSG_CALIBRATION,
sizeof(struct ucan_usb_ts_msg));

    if (!pts)
        return;

    pts->usb_frame_index = pcan_usb_frame_number();
    pts->unused = 0;
    pcan_device.last_time_sync = ts_ms;

    for (int i = 0; i < CAN_CHANNEL_MAX; i++)
    {
        if (!(pcan_device.can[i].opt_mask & UCAN_OPTION_BUSLOAD))
            continue;

        if (!pcan_device.can[i].bus_active)
            continue;

        struct ucan_bus_load_msg *pbs = pcan_data_alloc_buffer(UCAN_MSG_BUSLOAD,
sizeof(struct ucan_bus_load_msg));
```

```
        if (!pbs)
            return;

        pbs->channel = i;
        /* 0 ... 100% => 0 ... 4095 */
        pbs->bus_load = (pcan_device.can[i].bus_load * 4095u) / 100u;
    }

}
```

函式 write_manual_rx_frame 內容，先檢查是否有經過中斷程式，有經過中斷程式代表有 CAN 資料接收，g_u8RxFifO0CompleteFlag 才會被設為 1。該函數用來更新結構體 manual_msg 中的各個參數，結構體 g_sRxMsgFrame 在 CAN 中斷裡更新，並在此函數內將 g_sRxMsgFrame 放到 manual_msg 中。最後更新完 manual_msg 後呼叫 pcan_protocol_rx_frame 進行接收。

```
void write_manual_rx_frame(void)
{
    if (g_u8RxFifO0CompleteFlag != 1)
        return ;

    g_u8RxFifO0CompleteFlag = 0;

    struct t_can_msg manual_msg = { 0 };

    manual_msg.id = g_sRxMsgFrame.u32Id;

    manual_msg.size = g_sRxMsgFrame.u32DLC;

    for (int u8Cnt = 0; u8Cnt <  g_sRxMsgFrame.u32DLC; u8Cnt++)
    {
        manual_msg.data[u8Cnt] = g_sRxMsgFrame.au8Data[u8Cnt];
    }

    if (g_sRxMsgFrame.eIdType == eCANFD_SID)
        manual_msg.flags = MSG_FLAG_STD;
    else
        manual_msg.flags = MSG_FLAG_EXT;

    manual_msg.timestamp = pcan_timestamp_us();

    uint8_t target_channel = CAN_BUS_1;
}

    pcan_protocol_rx_frame(CAN_BUS_1, &manual_msg);
```

函式 pcan_protocol_rx_frame 內容，先將 CAN 協議的 DLC 碼根據接收結果做判斷，將 CAN 收到的資料存到 Buffer 中，並計算 nominal, data bit time，最後把資料搬到配置的空間(指標 pcan_msg)。

```
int pcan_protocol_rx_frame(uint8_t channel, struct t_can_msg *pmsg)
{
```

```
    static const uint8_t pcan_fd_len2dlc[] =
    {
        0, 1, 2, 3, 4, 5, 6, 7, 8,  /* 0 - 8 */
        9, 9, 9, 9,     /* 9 - 12 */
        10, 10, 10, 10,     /* 13 - 16 */
        11, 11, 11, 11,     /* 17 - 20 */
        12, 12, 12, 12,     /* 21 - 24 */
        13, 13, 13, 13, 13, 13, 13, 13, /* 25 - 32 */
        14, 14, 14, 14, 14, 14, 14, 14, /* 33 - 40 */
        14, 14, 14, 14, 14, 14, 14, 14, /* 41 - 48 */
        15, 15, 15, 15, 15, 15, 15, 15, /* 49 - 56 */
        15, 15, 15, 15, 15, 15, 15, 15  /* 57 - 64 */
    };
    struct ucan_rx_msg *pcan_msg = pcan_data_alloc_buffer(UCAN_MSG_CAN_RX, sizeof(struct
ucan_rx_msg) + pmsg->size);

    if (!pcan_msg)
        return -1;

    if (!(pmsg->flags & MSG_FLAG_ECHO))
    {
        uint32_t nqt = pcan_device.can[channel].nominal_qt;
        uint32_t dqt = pcan_device.can[channel].data_qt;
        pcan_device.can[channel].rx_time_ns += pcan_can_msg_time(pmsg, nqt, dqt);

        if (!pcan_device.can[channel].led_is_busy)
        {
            // pcan_led_set_mode( channel ? LED_CH1_RX:LED_CH0_RX, LED_MODE_BLINK_FAST,
237 );
            pcan_led_set_mode(LED_CH0_RX, LED_MODE_BLINK_FAST, 237);
        }
    }

    if (pmsg->size > CAN_PAYLOAD_MAX_SIZE)
        pmsg->size = CAN_PAYLOAD_MAX_SIZE;

    pcan_msg->channel_dlc = UCAN_MSG_CHANNEL_DLC(channel, pcan_fd_len2dlc[pmsg->size]);
    pcan_msg->client = pmsg->dummy;
    pcan_msg->flags = 0;
    pcan_msg->tag_low = 0;
    pcan_msg->tag_high = 0;

    /* we support only regular frames */
    if (pmsg->flags & MSG_FLAG_RTR)
        pcan_msg->flags |= UCAN_MSG_RTR;

    if (pmsg->flags & MSG_FLAG_EXT)
        pcan_msg->flags |= UCAN_MSG_EXT_ID;

    if (pmsg->flags & MSG_FLAG_FD)
    {
        pcan_msg->flags |= UCAN_MSG_EXT_DATA_LEN;

        if (pmsg->flags & MSG_FLAG_BRS)
            pcan_msg->flags |= UCAN_MSG_BITRATE_SWITCH;

        if (pmsg->flags & MSG_FLAG_ESI)
            pcan_msg->flags |= UCAN_MSG_ERROR_STATE_IND;
    }

    if (pmsg->flags & MSG_FLAG_ECHO)
    {
        pcan_msg->flags |= UCAN_MSG_API_SRR | UCAN_MSG_HW_SRR;
        pcan_msg->ts_low = pcan_timestamp_us();
    }
```

```
    else
    {
        pcan_msg->ts_low = pmsg->timestamp;
    }

    pcan_msg->can_id = pmsg->id;
    memcpy(pcan_msg->d, pmsg->data, pmsg->size);
    return 0;
}
```

函式 pcan_can_poll 內容。除了計數以外，呼叫 pcan_can_flush_tx 函式。

```
void pcan_can_poll(void)
{
    static uint32_t err_last_check = 0;
    uint32_t ts_ms;

    ts_ms = pcan_timestamp_millis();


    pcan_can_flush_tx(CAN_BUS_1);
}
```

函式 pcan_can_flush_tx 內容，將 TXFIFO 透過_can_send 發送。結尾更新 TXFIFO 的位置 (tx_tail)。

```
static void pcan_can_flush_tx(int bus)
{
    struct t_can_dev *p_dev = &can_dev_array[bus];
    struct t_can_msg *p_msg;

    /* empty fifo */
    if (p_dev->tx_head == p_dev->tx_tail)
        return;

    p_msg = &p_dev->tx_fifo[p_dev->tx_tail];

    _can_send(p_msg);

    /* update fifo index */
    p_dev->tx_tail = (p_dev->tx_tail + 1) & (CAN_TX_FIFO_SIZE - 1);
}
```

函式 _can_send 內容，設定傳輸的細節，最後交給 Library 中 canfd.c 的函數 CANFD_TransmitTxMsg 來將資料送出。在 M253 BSP 中有包含 CAN 的 Sample code 在 CAN 傳輸前都要包含這個設定。

```
static void _can_send(struct t_can_msg *p_msg)
{
    CANFD_FD_MSG_T      sTxMsgFrame;

    if (p_msg->flags & MSG_FLAG_EXT)
    {
        sTxMsgFrame.u32Id  = p_msg->id & 0x1FFFFFFF;
        sTxMsgFrame.eIdType = eCANFD_XID;
    }
    else
    {
        sTxMsgFrame.u32Id  = p_msg->id & 0x7FF;
```

```
        sTxMsgFrame.eIdType = eCANFD_SID;
    }

    /*Set the frame type*/

    /*Set the bitrate switch */
    sTxMsgFrame.bBitRateSwitch = 0;
    sTxMsgFrame.u32DLC  = p_msg->size;

    if (p_msg->flags & MSG_FLAG_RTR)
        sTxMsgFrame.eFrmType = eCANFD_REMOTE_FRM;
    else
        sTxMsgFrame.eFrmType = eCANFD_DATA_FRM;

    for (int u8Cnt = 0; u8Cnt < sTxMsgFrame.u32DLC; u8Cnt++)
        sTxMsgFrame.au8Data[u8Cnt] = p_msg->data[u8Cnt];

    if (CANFD_TransmitTxMsg(CANFD0, 0, &sTxMsgFrame) != eCANFD_TRANSMIT_SUCCESS)
    {
        printf("Failed to transmit message\n");
    }
}
```

USB 中斷的呼叫程式，在裝置接上時，會宣告此裝置的 Endpoint 的特性，此程式會宣告兩個接收 Host 裝置的 Endpoint 和發送給 Host 裝置的 Endpoint。在裝置收到 Endpoint 上由 Host 發送的 Bulk 指令時，會進入 USB 的中斷程式。指令內容包含初始化設定裝置以及對裝置發送 CAN 向外傳送資料內容，因此分為兩個 Endpoint 中斷。

函式 EP3_Handler 內容，由中斷程式呼叫，在對 Endpoint1 傳送 Bulk out 訊號時會導向此函式。此函式目的在接收來自 Host 的設定指令。

```
void EP3_Handler(void)
{
    int size;
    size = USBD_GET_PAYLOAD_LEN(EP3);
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
    g_pu8RxBuf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));

    for (int i = 0; i < size; i++)
        pcanpro_data.cmd_ep_buffer[i] = g_pu8RxBuf[i];

    pcan_protocol_process_data(PCAN_USB_EP_CMDOUT, pcanpro_data.cmd_ep_buffer, size);
}
```

函式 EP5_Handler 內容，由中斷程式呼叫，在對 Endpoint2 傳送 Bulk out 訊號時會導向此函式，此函式目的在接收來自 Host 的發送 CAN 訊號，由於對於傳訊號是以 512 bytes 去傳送的，所以會進到 pcan_protocol_process_data。

```
void EP5_Handler(void)
{
    unsigned int data_len = 0;
    unsigned int i = 0;
```

```
    data_len = USBD_GET_PAYLOAD_LEN(EP5);
    USBD_SET_PAYLOAD_LEN(EP5, EP5_MAX_PKT_SIZE);

    if (data_len > EP5_MAX_PKT_SIZE)
    {
        DBG_PRINTF("Error! EP3 data overrun!\n");
        data_len = EP5_MAX_PKT_SIZE;
    }


    /* Read bulk-out packet from USBD SRAM to g_Bulk_Buff[] */

    for (i = 0; i < data_len; i++)
    {
        pcanpro_data.data1_ep_buffer[i + data_out_count] = inpb(USBD_BUF_BASE +
EP5_BUF_BASE + i);
    }

    data_out_count = data_out_count + data_len;

    if (data_out_count >= 512)
    {
        pcan_protocol_process_data(PCAN_USB_EP_MSGOUT_CH1, pcanpro_data.data1_ep_buffer,
data_out_count);
        data_out_count = 0;
    }
}
```

函式 pcan_protocol_process_data 內容，根據是由 EP3_Handler 還是 EP5_Handler 呼叫此函數而不同。若 ep 為 1，表示收到的資料為控制指令，進入 pcan_protocol_process_cmd。ep 為 2 時，進入 pcan_protocol_tx_frame，更新 pcanpro_data 資料。

```
void pcan_protocol_process_data(uint8_t ep, uint8_t *ptr, uint16_t size)
{
    if (ep == 1)
    {
        pcan_protocol_process_cmd(ptr, size);
        return;
    }

    /* message data  */
    struct ucan_msg *pmsg = 0;

    while (size)
    {
        if (size < 4)
            break;

        pmsg = (void *)ptr;

        if (!pmsg->size || !pmsg->type)
            break;

        if (size < pmsg->size)
            break;

        size -= pmsg->size;
        ptr += pmsg->size;
```

```
        switch (pmsg->type)
        {
            //to host only
            case UCAN_MSG_CAN_TX:
                pcan_protocol_tx_frame((struct ucan_tx_msg *)pmsg);
                break;

            case UCAN_MSG_CAN_TX_PAUSE:
                break;

            case UCAN_CMD_END_OF_COLLECTION:
            case 0xffff:
                return;

            default:
                MY_ASSERT(0);
                break;
        }
    }
}
```

函式 pcan_protocol_process_cmd，包含各種連接時會用到的指令。

函式 pcan_protocol_tx_frame 內容，解析 Bulk out 中的資料內容，並用 pcan_can_write 放到 can_dev_array[bus].tx_fifo 中。

```
int pcan_protocol_tx_frame(struct ucan_tx_msg *pmsg)
{
    static const uint8_t pcan_fd_dlc2len[] =
    {
        0, 1, 2, 3, 4, 5, 6, 7,
        8, 12, 16, 20, 24, 32, 48, 64
    };

    struct t_can_msg msg = { 0 };
    uint8_t channel;

    channel = UCAN_MSG_CHANNEL(pmsg);

    if (channel >= CAN_CHANNEL_MAX)
        return -1;

    msg.id = pmsg->can_id;

    /* CAN-FD frame */
    if (pmsg->flags & UCAN_MSG_EXT_DATA_LEN)
    {
        msg.flags |= MSG_FLAG_FD;

        if (pmsg->flags & UCAN_MSG_BITRATE_SWITCH)
            msg.flags |= MSG_FLAG_BRS;

        if (pmsg->flags & UCAN_MSG_ERROR_STATE_IND)
            msg.flags |= MSG_FLAG_ESI;

        msg.size = pcan_fd_dlc2len[UCAN_MSG_DLC(pmsg)];
    }
```

```
    else
    {
        msg.size = UCAN_MSG_DLC(pmsg);
    }

    if (msg.size > sizeof(msg.data))
        return -1;

    if (pmsg->flags & UCAN_MSG_RTR)
        msg.flags |= MSG_FLAG_RTR;

    if (pmsg->flags & UCAN_MSG_EXT_ID)
        msg.flags |= MSG_FLAG_EXT;

    if (pmsg->flags & (UCAN_MSG_HW_SRR))
    {
        msg.flags |= MSG_FLAG_ECHO;
        msg.dummy = pmsg->client;
    }

    memcpy(msg.data, pmsg->d, msg.size);

    msg.timestamp = pcan_timestamp_us();

    return 0;
}
```

函式 pcan_can_write 內容。

```
int pcan_can_write(int bus, struct t_can_msg *p_msg)
{
    struct t_can_dev *p_dev = &can_dev_array[bus];

    if (!p_dev)
        return 0;

    if (!p_msg)
        return 0;

    uint32_t  tx_head_next = (p_dev->tx_head + 1) & (CAN_TX_FIFO_SIZE - 1);

    /* overflow ? just skip it */
    if (tx_head_next == p_dev->tx_tail)
    {
        ++p_dev->tx_ovfs;
        return -1;
    }

    p_dev->tx_fifo[p_dev->tx_head] = *p_msg;
    p_dev->tx_head = tx_head_next;

    return 0;
}
```

3. 軟體與硬體需求

3.1 軟體需求

BSP 版本

- M253_Series_BSP_CMSIS_V3.00.010

IDE 版本

- Keil uVision 5.42
- Visual Studio Code 1.105.0

3.2 硬體需求

電路元件

- NuMaker-M253LE3AE-V1.1

腳位連接

- 將 USB 腳位接上 PC 上，並將開發版上的 CAN 腳位接到其他 CAN 裝置上。

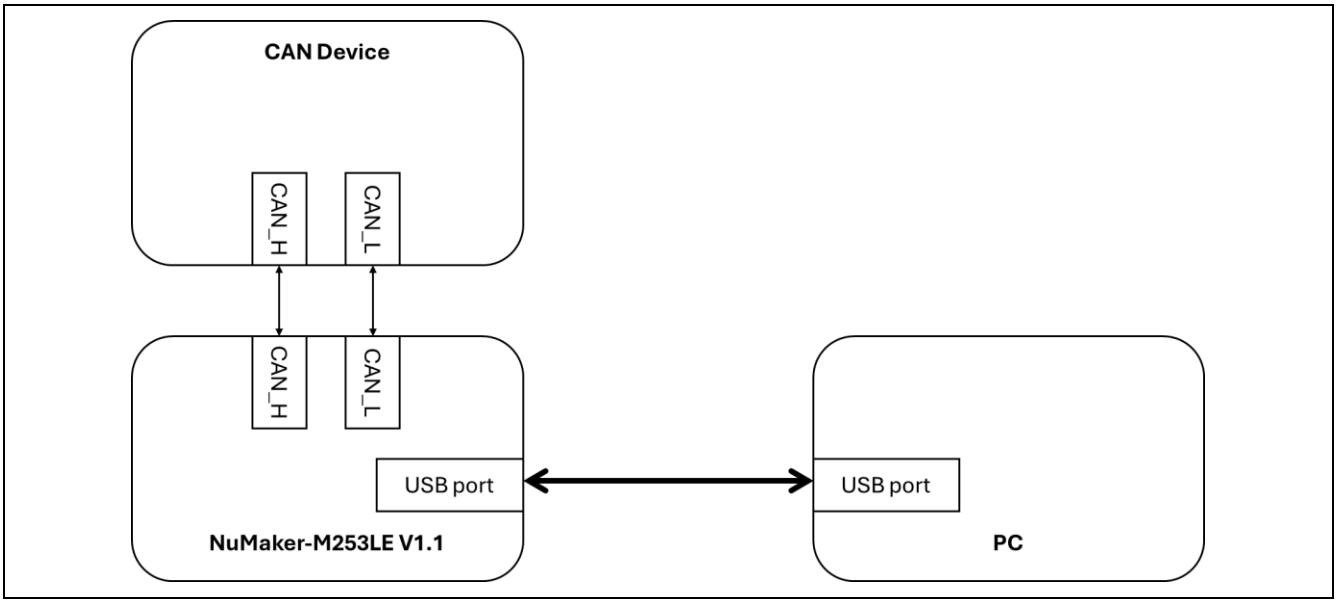


圖 3-1 腳位連接

4. 目錄資訊

目錄結構如下圖所示。

EC_M253_USB_to_CAN_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

圖 4-1 目錄資訊

5. 範例程式執行

5.1 Keil MDK

1. 在 Windows 檔案總管中進入 **Keil** 專案資料夾，再開啟 *M253_USB_to_CAN.uvprojx* 專案檔。
2. 編譯程式碼
3. 下載程式碼
4. 進入偵錯模式
   - 執行程式碼

5.2 VS Code

1. 在 **VS Code** IDE 中開啟 **File** 選單
   - 選取 **Open Folder**
   - 選取專案目錄下的 **VSCode** 資料夾
2. 選擇 **CMSIS**
   - 點選 **Build solution** 編譯專案
   - 點選 **Load & Run application** 下載映像檔至目標晶片
   - 點選 **Load & Debug application** 進入偵錯模式

5.3 執行

1. 使用 Linux 連接開發版的 USB port，並按照以下順序操作:

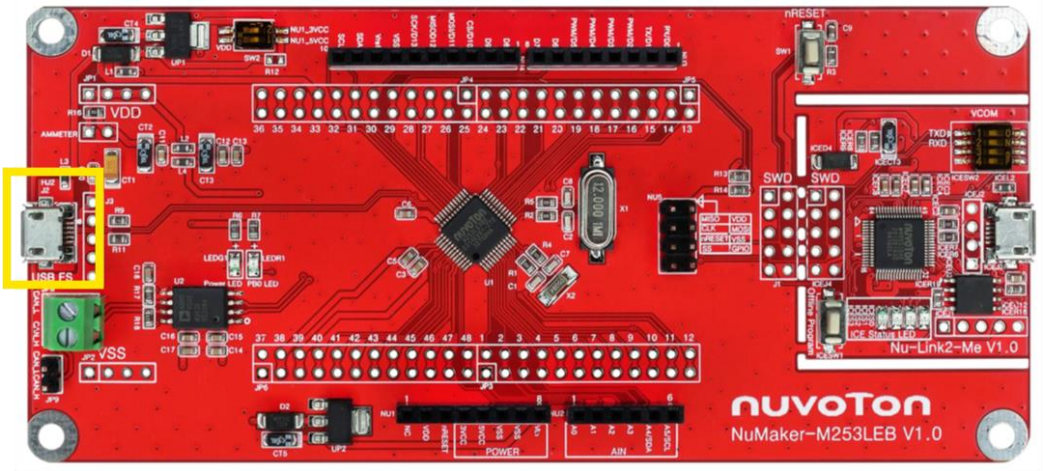


圖 5-1 NuMaker-M253LE 開發板 USB port 位置

2. 列出當前連接的 CAN 裝置(ip link list type can)

3. 停止裝置(ip link set <device> down)

4. 設定裝置的傳輸速度(ip link set <device> type can bitrate 500000)

5. 開啟裝置(ip link set <device> up)

6. 當裝置開啟時，可以讓裝置發送 CAN 訊號(cansend <device> <can_id>#{data})，也可以讓裝置監看訊號線上的傳輸內容(candump <device>)。

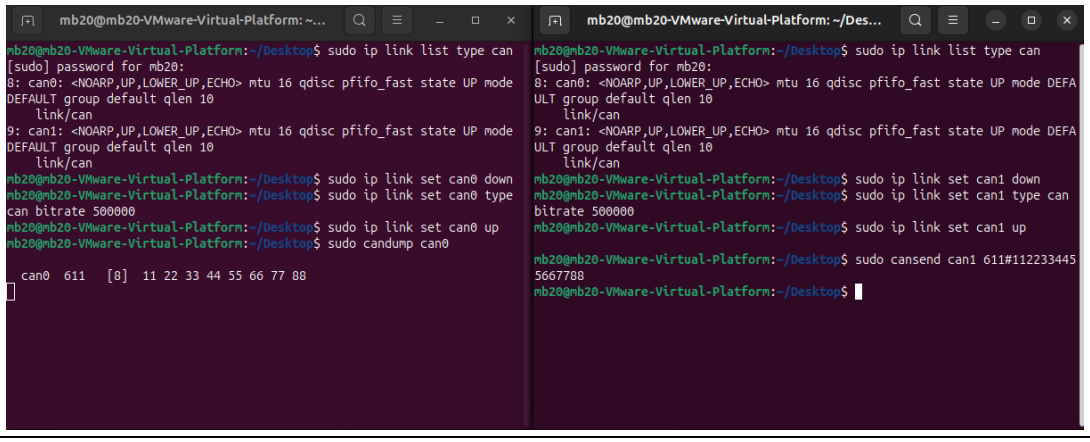


圖 5-2 Linux 環境下用兩個 M253 裝置以 CAN 做通訊

6. 修訂紀錄

Date	Revision	Description
2026.06.18	1.00	初始發布。

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.