使用SPI Flash 作為 USB 儲存裝置

NuMicro® 32 位系列微控制器範例代碼介紹

文件資訊

代碼簡述	微控制器扮演 USB 大容量儲存裝置，並使用 SPI Flash 作為儲存媒介。
BSP 版本	NUC123Series_BSP_CMSIS_v3.01.001
開發平台	NuTiny-EVB-123-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 功能介紹

1.1 簡介

微控制器 NUC123 透過 USB 通訊埠連接電腦，扮演大容量儲存裝置類別，並連接外部的 SPI Flash Memory 作為儲存裝置的媒介，使用者即可將微控制器作為一般的外接 USB 隨身碟使用，可在個人電腦上進行檔案的寫入與讀取。

1.2 原理

本次實驗採用 USB Mass Storage Device Class (MSC) Bulk-Only Transfer 協議，根據該協議規範，Host 與 Device 交換資料的過程可分為三個狀態階段，分別為Command Transport (CBW)，Data-Out / Data-In，以及 Status Transport (CSW)，其狀態切換流程如圖1-1所示。

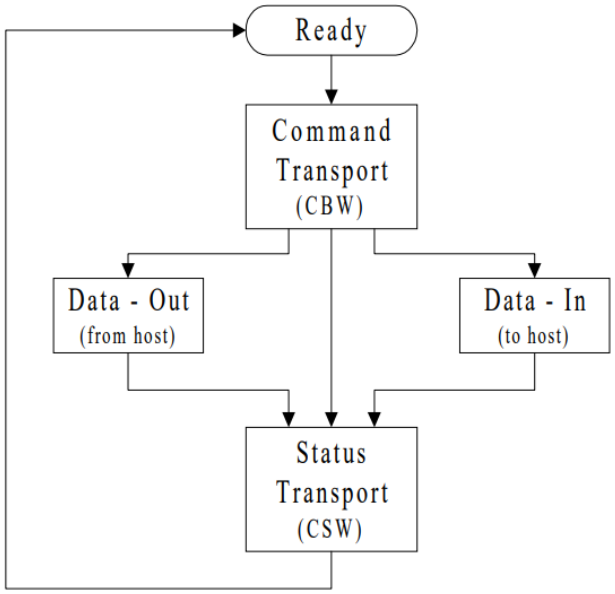


圖1-1 Mass Storage Class Command/Data/Status Protocol

MSC 同時也定義了兩個專屬的類別請求，分別為 Bulk-Only Mass Storage Reset，以及 Get Max LUN 命令，Bulk-Only Mass Storage Reset 命令用以重置 Device 回到 Ready 狀態，通知裝置準備接收 CBW 命令；Get Max LUN 則用以取得 Device 所支援的邏輯單元 (儲存槽) 數量，其命令的詳細內容與格式可參閱 USB Mass Storage Class Bulk-Only Transport Specification 文件。

1.2.1 Command Transport (CBW)

Command Transport 內含有一個 Transaction，資料方向為 OUT，由 Host 透過 Bulk-Out Endpoint 對 Device 送出指定命令的資料封包，並以 Command Block Wrapper (CBW) 之格式封裝，封包格式如圖1-2所示。在 CBW 封包當中，欄位 CBWCB 存放 Host 對於 Device 的儲存媒體進行存取或控制的指令，該指令類型會決定下一個狀態為 Data-In 或是 Data-Out，以及雙方該傳遞的內容，詳細的指令描述可參閱 UFI Command Specification 文件規範。

Table 5.1 - Command Block Wrapper

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11 (08h-0Bh)	*dCBWDataTransferLength*							
12 (0Ch)	*bmCBWFlags*							
13 (0Dh)	Reserved (0)				*bCBWLUN*			
14 (0Eh)	Reserved (0)			*bCBWCBLength*				
15-30 (0Fh-1Eh)	*CBWCB*							

圖1-2 Command Block Wrapper (CBW) 封包格式

1.2.2 Data-In / Data-Out

Data-In / Data-Out 狀態下的行為受 CBW 內的命令所影響，舉例來說，若 CBWCB 內的 UFI 指令為 WRITE，則代表由 Host 希望向 Device 的儲存媒介寫入檔案資料，同時雙方將狀態切換為 Data-Out；反之若 UFI指令為 READ，則表示 Host 向 Device 讀取儲存裝置內的檔案，由 Device 向 Host 送出資料封包，此時狀態切換至 Data-In。Data-In / Data-Out 階段允許包含多個 Transaction，由 CBWCB 內的 dCBWDataTransferLength 欄位所影響。順帶一提，本次實驗的儲存媒介是採用外接的 SPI Flash Memory，微控制器接收到讀取或寫入的指令後，必須驅動周邊 SPI 對外部 Flash Memory 的指定區塊進行資料的讀寫。

1.2.3 Status Transport (CSW)

Status Transport 的用途是 Device 向 Host 回覆本次 CBW 命令執行的狀態，含有一個 Transaction，資料方向為 IN，此階段的資料封包以 Command Status Wrapper (CSW) 之格式封裝，封包格式如圖1-3所示。Host 可透過 CSW 內的 bCSWStatus 欄位判斷 Device 在執行 CBW 命令的過程是否出現錯誤，bCSWStatus 數值意義如圖1-4所示。

Table 5.2 - Command Status Wrapper

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11 (8-Bh)	*dCSWDataResidue*							
12 (Ch)	*bCSWStatus*							

圖1-3 Command Status Wrapper (CSW) 封包格式

Table 5.3 - Command Block Status Values

Value	Description
00h	Command Passed ("good status")
01h	Command Failed
02h	Phase Error
03h and 04h	Reserved (Obsolete)
05h to FFh	Reserved

圖1-4 bCSWStatus 欄位內容

1.2.4 USB 端點配置

本次實驗需要配置四個端點給予 USB 介面使用，如下所示：

- Endpoint 0: Control-In
- Endpoint 1: Control-Out
- Endpoint 2: Bulk-In
- Endpoint 3: Bulk-Out

1.2.5 Serial Flash Memory

本次實驗使用 Winbond 25X16A SPI Flash 作為 Mass Storage Device 的儲存媒介，儲存容量為 2 MB，最小的寫入單位為一個 Page (256 B)，每 16 Page 可組成一個 Sector (4 KB)，每 256 Page 則組成一個 Block (64K Byte)，其中 Sector 為最小的抹除單位。SPI Flash 的控制方式非常簡便，微控制器首先透過以下的命令集合對 SPI Flash 下達指令，包含 Read、Program 或是 Erase 等行為，接著雙方會依據指令的類別，決定後續傳遞的資料的內容與長度。

INSTRUCTION NAME	BYTE 1 CODE	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6	N-BYTES
Write Enable	06h						
Write Disable	04h						
Read Status Register	05h	(S7–S0)(1)					(2)
Write Status Register	01h	S7–S0					
Read Data	03h	A23–A16	A15–A8	A7–A0	(D7–D0)	(Next byte)	continuous
Fast Read	0Bh	A23–A16	A15–A8	A7–A0	dummy	(D7–D0)	(Next Byte) continuous
Fast Read Dual Output	3Bh	A23–A16	A15–A8	A7–A0	dummy	I/O = (D6,D4,D2,D0) O = (D7,D5,D3,D1)	(one byte per 4 clocks, continuous)
Page Program	02h	A23–A16	A15–A8	A7–A0	(D7–D0)	(Next byte)	Up to 256 bytes
Block Erase (64KB)	D8h	A23–A16	A15–A8	A7–A0			
Sector Erase (4KB)	20h	A23–A16	A15–A8	A7–A0			
Chip Erase	C7h						
Power-down	B9h						
Release Power-down / Device ID	ABh	dummy	dummy	dummy	(ID7-ID0)(4)		
Manufacturer/ Device ID (3)	90h	dummy	dummy	00h	(M7-M0)	(ID7-ID0)	
JEDEC ID	9Fh	(M7-M0) Manufacturer	(ID15-ID8) Memory Type	(ID7-ID0) Capacity			

圖1-5 Winbond SPI Flash 命令表

圖1-5為 Winbond 25X16A 指令表，以 Page Program (02h) 與 Read Status Register (05h) 指令為例，可參閱下圖1-6與圖1-7之時序，Master 首先送出 1 Byte 的指令 02h，接續 24-Bit 的資料寫入位址與 256 Byte 的資料後，便完成 Page Program 的指令傳輸。之後，微控制器可再使用 Read Status Register (05h) 指令來詢問 SPI Flash 的狀態，來檢查資料是否已經寫入完

畢。使用者可參閱 Winbond W25X16A Datasheet 文件，內有各個指令詳細的使用方法與說明。

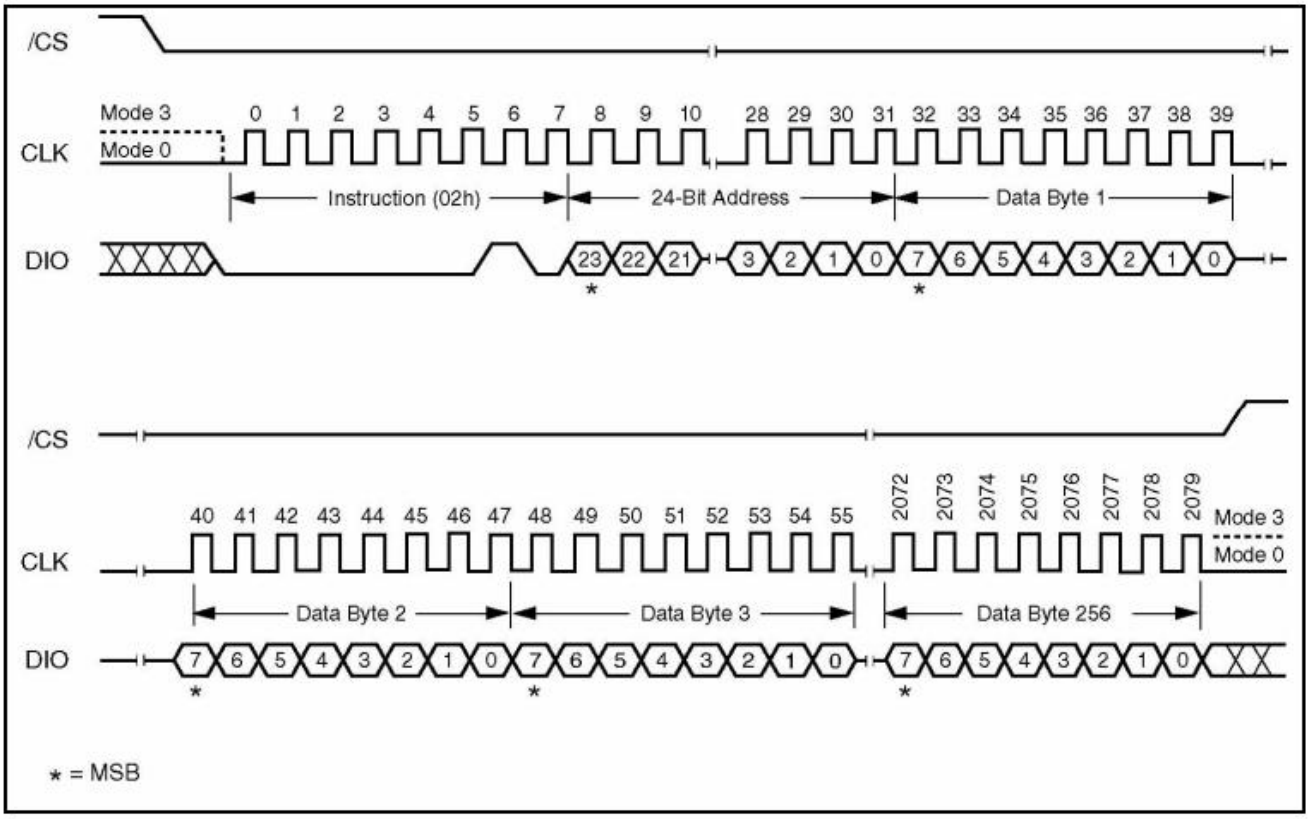


圖1-6 Page Program 命令時序圖

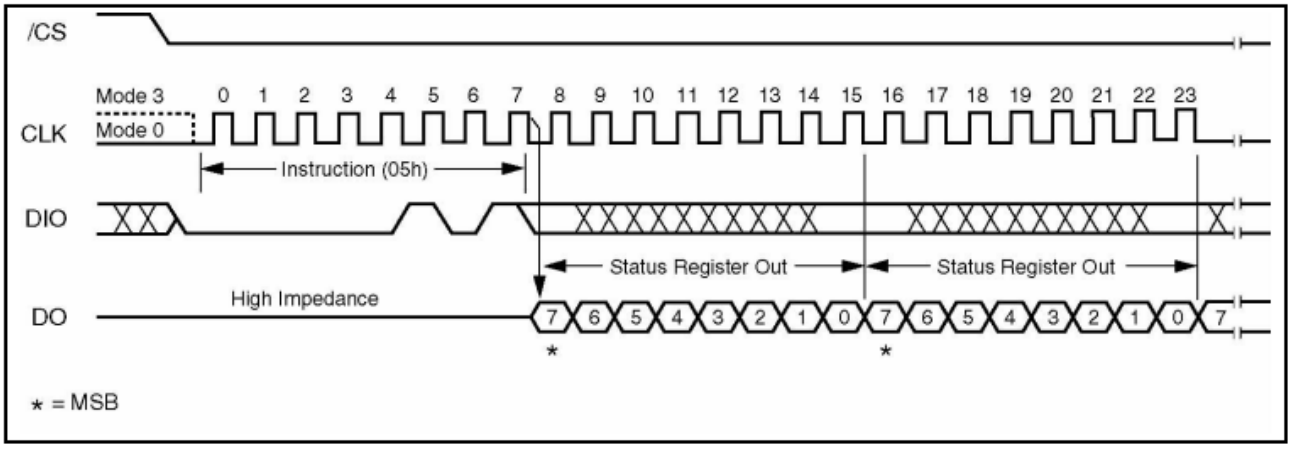


圖1-7 Read Status Register 命令時序圖

1.3 執行結果

將 NUC123 USB 接入電腦，開啟【裝置管理員】，可看到 Disk drives 區域出現 Nuvoton USB Mass Storage USB Device 裝置，如圖1-8所示。此時至【我的電腦】開啟外接裝置的儲存槽，格式化後即可進行檔案的創建、修改、與儲存等動作。舉例來說，本次實驗創建一個 Helloworld.txt檔案，並在檔案輸入 “USB Mass Storage SPI Flash Example.” 字串後儲存，關閉檔案並移除USB 後再重新插入，檢查 Helloworld.txt 的內容可正常讀取，表示資料已經正確寫入至外部 SPI Flash 當中，執行結果如圖1-9所示。

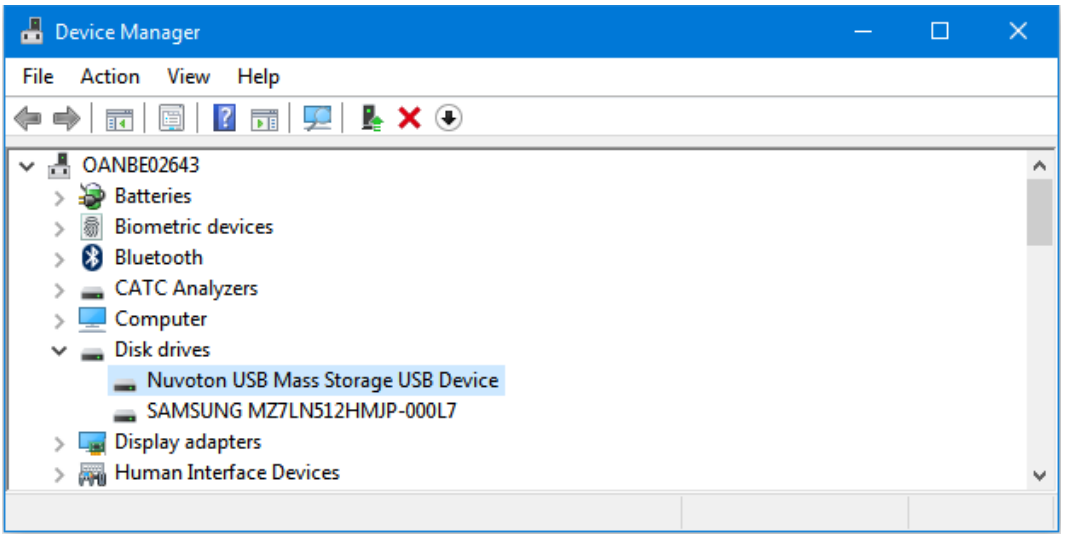


圖 1-8 Device Manager screenshot

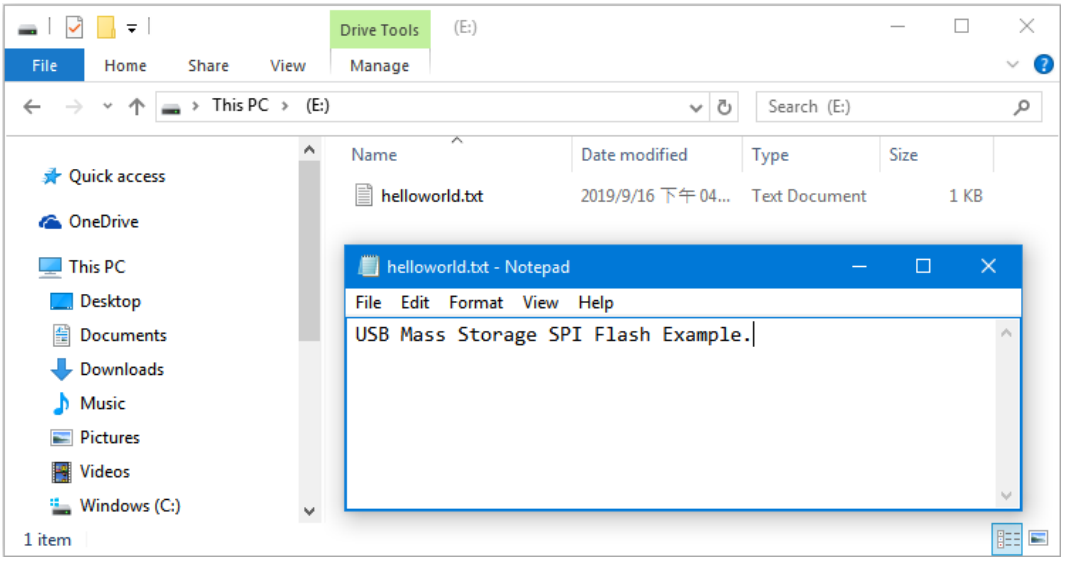


圖 1-9 文字檔案讀寫畫面

2 代碼介紹

2.1 類別請求 - Get Max LUN

當 Device 接收到 Get Max LUN 類別請求，必須回覆該裝置所支援的邏輯單元數量，本次範例僅支援單一邏輯槽 (編號0)，因此將 Endpoint 0 (Control-In) 的緩衝區填入0，資料長度為 1 Byte，並在下一次的Transaction 當中回覆給 Host。

```
void MSC_ClassRequest(void)
{
    uint8_t buf[8];

    USBD_GetSetupPacket(buf);

    if(buf[0] & EP_INPUT)    /* request data transfer direction */
    {
        // Device to host
        switch(buf[1])
        {
            case GET_MAX_LUN:
            {
                if((buf[4] == gsInfo.gu8ConfigDesc[LEN_CONFIG + 2]) && (buf[2] + buf[3] +
                    buf[6] + buf[7] == 1))
                {
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = 0;
                    /* Data stage */
                    USBD_SET_DATA1(EP0);
                    USBD_SET_PAYLOAD_LEN(EP0, 1);
                    /* Status stage */
                    USBD_PrepareCtrlOut(0, 0);
                }
                else
                    USBD_SET_EP_STALL(EP1); // Stall when wrong parameter
                break;
            }
            default:
            {
```

```
                /* Setup error, stall the device */
                USBD_SetStall(0);
                DBG_PRINTF("Unknow MSC req(0x%x). stall ctrl pipe\n", buf[1]);
                break;
            }
        }
    }
```

2.2 類別請求 - Bulk-Only Mass Storage Reset

當 Device 接收到 Bulk-Only Mass Storage Reset 指令，必須將硬體設定至準備接收 CBW 的狀態，可參閱圖 1-1 Mass Storage Class Command/Data/Status Protocol 流程。CBW 封包長度為固定 31 Byte，因此將 Endpoint 3 (Bulk-Out) payload 長度設為 31。

```
    else
    {
        // Host to device
        switch(buf[1])
        {
            case BULK_ONLY_MASS_STORAGE_RESET:
            {
                /* Check interface number with cfg descriptor and check wValue = 0,
                   wLength = 0 */
                if((buf[4] == gsInfo.gu8ConfigDesc[LEN_CONFIG + 2]) && (buf[2] + buf[3] +
                    buf[6] + buf[7] == 0))
                {

                    g_u32Length = 0; // Reset all read/write data transfer
                    USBD_LockEpStall(0);

                    /* Clear ready */
                    USBD->EP[EP2].CFGP |= USBD_CFGP_CLRRDY_Msk;
                    USBD->EP[EP3].CFGP |= USBD_CFGP_CLRRDY_Msk;

                    /* Prepare to receive the CBW */

                    g_u8EP3Ready = 0;
                    g_u8BulkState = BULK_CBW;
```

```
                    USBD_SET_DATA1(EP3);
                    USBD_SET_EP_BUF_ADDR(EP3, g_u32BulkBuf0);
                    USBD_SET_PAYLOAD_LEN(EP3, 31);

                }
                else
                {
                    /* Stall when wrong parameter */
                    USBD_SET_EP_STALL(EP1);
                }
                /* Status stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 0);

                break;
            }
            default:
            {
                // Stall
                /* Setup error, stall the device */
                USBD_SetStall(0);
                DBG_PRINTF("Unknow MSC req (0x%x). stall ctrl pipe\n", buf[1]);
                break;
            }
        }
    }
}
```

2.3 Mass Storage Class 命令處理

CBW 內夾帶了 Host 對於 Device 進行訪問、讀取、寫入等命令， MassStorage.c 檔案內的 MSC_ProcessCmd() 函式當中實作了 Device 如何解析 CBW 封包以及應對各個 UFI 指令的行為，以下以幾條較為基本與常用的指令做為範例。

```
void MSC_ProcessCmd(void)
{
    uint8_t u8Len;
    int32_t i;

    if(g_u8EP3Ready)
    {
        g_u8EP3Ready = 0;
        if(g_u8BulkState == BULK_CBW)
        {
            u8Len = USBD_GET_PAYLOAD_LEN(EP3);
            if(u8Len > 31) u8Len = 31;

            /* Check Signature & length of CBW */
            /* Bulk Out buffer */
            if((*(uint32_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf0) != CBW_SIGNATURE)
            /* || (u8Len != 31)*/)
            {
                USBD_SET_EP_STALL(EP2);
                USBD_SET_EP_STALL(EP3);
                USBD_LockEpStall(1 << EP3);
                g_u8BulkState = BULK_CBW;
                DBG_PRINTF("CBW signature fail. stall bulk out pipe\n");
                return;
            }

            /* Get the CBW */
            for(i = 0; i < u8Len; i++)
                *((uint8_t *)(&g_sCBW.dCBWSignature) + i) = \
                      *(uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf0 + i);

            /* Prepare to echo the tag from CBW to CSW */
            g_sCSW.dCSWTag = g_sCBW.dCBWTag;
```

2.3.1 INQUIRY

主機詢問裝置的基本資訊，例如製造商、產品名稱或版本等資訊。裝置接收到 INQUIRY 指令後，狀態切換至 Data-In，將產品訊息寫入 Endpoint 2 (Bulk-In) 緩衝區。

```
case UFI_INQUIRY:
{
    uint32_t u32Len;
    uint8_t u8PageCode;

    u32Len = g_sCBW.dCBWDataTransferLength;
    /* Limit length */
    if(u32Len > 36) u32Len = 36;
        u8PageCode = g_sCBW.au8Data[0];

    g_u8BulkState = BULK_IN;
    if(u32Len)
    {
        /* u8PageCode should be zero */
        if(u8PageCode)
        {
         /* Expecting a STALL after data phase completes with a zero-length or short
            packet */
            //USBD_SET_EP_STALL(EP0);
            USBD_SET_EP_STALL(EP3);
            USBD_LockEpStall(1 << EP3);
            DBG_PRINTF("INQUIRY page code = %d", u8PageCode);
        }

        /* Bulk IN buffer */
        USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1), \
                     (uint8_t *)g_au8InquiryID, u32Len);
        USBD_SET_PAYLOAD_LEN(EP2, u32Len);
        DBG_PRINTF("Inquiry, len %d\n", u32Len);
    }
    Else
    {
        /* Next is status phase if zero data length in data phase */
        MSC_AckCmd();
     }
       return;
}
```

2.3.2 READ_FORMAT_CAPACITIES

READ_FORAMT_CAPACITIES 為 Host 讀取 Device 的儲存容量與可格式化容量的指令，Device 接收到此指令後，將狀態切換至 Data-In，並呼叫 MSC_ReadFormatCapacity ()。此時根據 SPI Flash 的容量大小換算邏輯磁區的數量，將資訊寫入至 Endpoint 2 (Bulk-In) 緩衝區，並在下一個 Transaction 送出。

```
case UFI_READ_FORMAT_CAPACITY:
{
    g_u32Length = g_sCBW.dCBWDataTransferLength;

    /* format capacity descriptor length is fixed to be 12 bytes */
    if(g_u32Length > 20) g_u32Length = 20;
       g_u32Address = MassCMD_BUF;

   MSC_ReadFormatCapacity();
    g_u8BulkState = BULK_IN;
    if(g_u32Length > 0)
    {
        if(g_u32Length > EP2_MAX_PKT_SIZE)
            g_u8Size = EP2_MAX_PKT_SIZE;
        else
            g_u8Size = g_u32Length;

        /* Bulk IN buffer */
        USBD_MemCopy((uint8_t *)(USBD_BUF_BASE + g_u32BulkBuf1),
                     (uint8_t *)g_u32Address, g_u8Size);
        g_u32BytesInStorageBuf = g_u8Size;

        g_u32Address += g_u8Size;
        USBD_SET_EP_BUF_ADDR(EP2, g_u32BulkBuf0);
        MSC_Read();
    }
//  else
//      MSC_AckCmd();
    return;
}
```

2.3.3 READ_10 / READ_12

READ_10 / READ_12 為 Host 用來讀取 Device 儲存裝置內檔案的指令，Device 接收到該指令後將狀態切換為 Data-In，此處呼叫 MSC_ReadMedia() 對 SPI Flash 進行讀取操作。Device 根據 CBW 內的 Logic Block Address 以及 dCBWDataTransferLength 欄位來讀取指定的磁區位址以及資料長度，若資料長度超過 Endpoint 2 (Bulk-In) 的最大單一封包大小，則可以分割為多個 Bulk-In Transaction 送出。

```
case UFI_READ_10:
case UFI_READ_12:
{
    /* Check if it is a new transfer */
    if(g_u32Length == 0)
    {
        /* Prepare the data for Bulk IN transfer */

        /* Get LBA address */
        g_u32Address = get_be32(&g_sCBW.au8Data[0]);
        g_u32LbaAddress = g_u32Address * UDC_SECTOR_SIZE;
        g_u32Length = g_sCBW.dCBWDataTransferLength;
        g_u32BytesInStorageBuf = g_u32Length;
        DBG_PRINTF("Read addr=0x%x, len=0x%x\n", g_u32LbaAddress, g_u32Length);
        /* Error check  */
        if((g_u32LbaAddress > DRVSPIFLASH_TOTAL_SIZE) || (g_u32LbaAddress + g_u32Length
            > DRVSPIFLASH_TOTAL_SIZE))
        {
            USBD_SET_EP_STALL(EP2);
            USBD_SET_EP_STALL(EP3);
            USBD_LockEpStall((1 << EP2) | (1 << EP3));
            DBG_PRINTF("Stall ep2, ep3. addr=0x%x, len=0x%x\n", g_u32LbaAddress,
                        g_u32Length);

            return;
        }

        i = g_u32Length;
        if(i > STORAGE_BUFFER_SIZE)
            i = STORAGE_BUFFER_SIZE;

        MSC_ReadMedia(g_u32Address * UDC_SECTOR_SIZE, i, (uint8_t *)STORAGE_DATA_BUF);
```

```
        g_u32BytesInStorageBuf = i;
        g_u32LbaAddress += i;
    }
    g_u32Address = STORAGE_DATA_BUF;

    /* Indicate the next packet should be Bulk IN Data packet */
    g_u8BulkState = BULK_IN;

    if(g_u32BytesInStorageBuf > 0)
    {
        /* Set the packet size */
        if(g_u32BytesInStorageBuf > EP2_MAX_PKT_SIZE)
            g_u8Size = EP2_MAX_PKT_SIZE;
        else
            g_u8Size = g_u32BytesInStorageBuf;

        /* Prepare the first data packet (DATA1) */
        /* Bulk IN buffer */
        USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE + g_u32BulkBuf1),
                     (uint8_t *)g_u32Address, g_u8Size);
        g_u32Address += g_u8Size;

        /* kick - start */
        USBD_SET_EP_BUF_ADDR(EP2, g_u32BulkBuf1);
        /* Trigger to send out the data packet */
        USBD_SET_PAYLOAD_LEN(EP2, g_u8Size);
        g_u32Length -= g_u8Size;
        g_u32BytesInStorageBuf -= g_u8Size;
    }
//   else
//         MSC_AckCmd();
      return;
    }
```

2.3.4 WRITE_10 / WRITE_12

WRITE_10 / WRITE_12 為 Host 將檔案寫入 Device 儲存裝置的指令，Device 接收到該指令後將狀態切換為 Data-Out，準備 Endpoint 3 (Bulk-Out) 接收來自 Host 的資料，實際的檔案內容將會在下一個 Transaction 當中到來。同樣地，若寫入的檔案超過 Endpoint 3 的最大單一封包大小，則資料會被分割至數個 Transaction 當中。

```
case UFI_WRITE_10:
case UFI_WRITE_12:
{
    if(g_u32Length == 0)
    {
        g_u32Length = g_sCBW.dCBWDataTransferLength;
        g_u32Address = STORAGE_DATA_BUF;
        g_u32DataFlashStartAddr = get_be32(&g_sCBW.au8Data[0]) * UDC_SECTOR_SIZE;
    }
    DBG_PRINTF("Write 0x%x  0x%x\n", g_u32Address, g_u32Length);

    if((g_u32Length > 0))
    {
        USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
        g_u8BulkState = BULK_OUT;
    }
    return;
}
```

在 Data-Out 狀態下接收到檔案封包，呼叫 MSC_Write() 將 Endpoint 3 緩衝區資料讀出，並驅動周邊 SPI 將檔案寫入 SPI Flash 的指定磁區。

```
else if(g_u8BulkState == BULK_OUT)
{
    switch(g_sCBW.u8OPCode)
    {
        case UFI_WRITE_10:
        case UFI_WRITE_12:
        case UFI_MODE_SELECT_10:
        {
            MSC_Write();
            return;
        }
```

MSC_Write() 會進一步呼叫底層 SpiWrite() 執行寫入操作，此部分須注意若寫入的地址並非 Sector Alignment 則需要採取特別處理，因 SPI Flash 的最小抹除單位為一個 Sector (4K Bytes)，地址沒有 Alignment 的情況下，微控制器會先將整個 Sector 內原有的資料讀出，修改指定位址的內容，擦出整個 Sector 後再將資料重新寫入。

```
void SpiWrite(uint32_t u32Addr, uint32_t u32Size, uint32_t u32BufAddr)
{
    /* This is low level write function of USB Mass Storage */
    int32_t i32Len, i, i32Offset;
    uint32_t *pu32;
    uint32_t u32AlignAddr;

    i32Len = (int32_t)u32Size;

    if (i32Len == DRVSPIFLASH_SECTOR_SIZE && ((u32Addr & (DRVSPIFLASH_SECTOR_SIZE-1)) ==
        0))
    {
        /* one-sector length & the start address is sector-alignment */
        WINBOND25X16A_EraseSector(u32Addr);

        while (i32Len >= DRVSPIFLASH_PAGE_SIZE)
        {
            WINBOND25X16A_ProgramPage( u32Addr, (uint8_t *)u32BufAddr);
            i32Len     -= DRVSPIFLASH_PAGE_SIZE;
            u32BufAddr += DRVSPIFLASH_PAGE_SIZE;
            u32Addr    += DRVSPIFLASH_PAGE_SIZE;
        }
    }
    else
    {
        do
        {
            /* u32AlignAddr: sector address */
            u32AlignAddr = u32Addr & 0x1FFFF000;

             /* Get the sector i32Offset*/
            i32Offset = (u32Addr & (DRVSPIFLASH_SECTOR_SIZE-1));

           if (i32Offset || (u32Size < DRVSPIFLASH_SECTOR_SIZE))
           {
```

```
                /* if the start address is not sector-alignment or the u32Size is less
                   than one sector, read back the data of the destination sector to
                   g_au32SectorBuf[]. */
                SpiRead(u32AlignAddr, DRVSPIFLASH_SECTOR_SIZE,
                        (uint32_t)&g_au32SectorBuf[0]);
            }

            /* Update the data */
            pu32 = (uint32_t *)u32BufAddr;

            /* i32Len: the byte count between the start address and the end of a sector.*/
            i32Len = DRVSPIFLASH_SECTOR_SIZE - i32Offset;

            /* check if the range of data arrive at the end of a sector. */
            if (u32Size < i32Len)
            {
                /* Not arrive at the end of a sector. "i32Len" indicate the actual byte
                    count of data. */
                i32Len = u32Size;
            }
            for (i=0; i<i32Len/4; i++)
            {
                g_au32SectorBuf[i32Offset/4 + i] = pu32[i];
            }

            WINBOND25X16A_EraseSector(u32AlignAddr);

            for (i=0;i<16;i++) /* one sector (16 pages) */
            {
                WINBOND25X16A_ProgramPage(u32AlignAddr + (i << 8),
                                          (uint8_t *)g_au32SectorBuf + (i << 8));
            }

            u32Size -= i32Len;
            u32Addr += i32Len;
            u32BufAddr += i32Len;

        }while (u32Size > 0);
    }
}
```

2.3.5 TEST_UNIT_READY

Host 在沒有其他命令要執行的情況下，會定期向 Device 送出 TEST UNIT READY 指令，檢查 Device 是否 Ready，正常狀態下 Device 僅需回覆 Good Status 的 CSW 封包即可

```
case UFI_TEST_UNIT_READY:
{
    DBG_PRINTF("Test Unit\n");
    g_u8BulkState = BULK_IN;
    MSC_AckCmd();
    return;
}
```

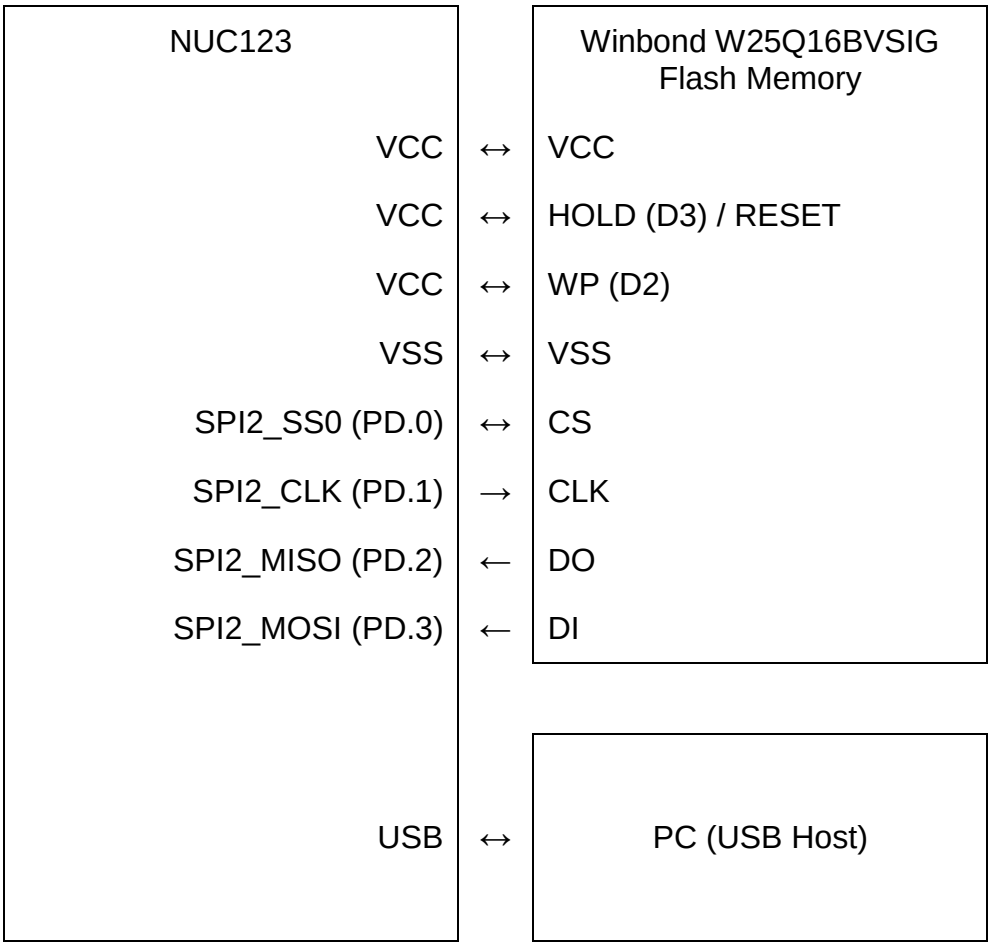
MSC_AckCmd () 內，將 bCSWStatus 欄位設為 0，表示Good Status。

```
case UFI_TEST_UNIT_READY:
{
                g_sCSW.dCSWDataResidue = 0;
                g_sCSW.bCSWStatus = 0;
                break;
}
```

3 軟體與硬體環境

- **軟體環境**
  - BSP 版本
    - NUC123Series_BSP_CMSIS_v3.01.001
  - IDE 版本
    - Keil uVersion 5.26
- **硬體環境**
  - 電路元件
    - NuTiny-EVB-123-LQFP64 V1.0 Developer Board
    - Winbond W25X16A Serial Flash Memory
    - USB Mini USB Cable

■ 示意圖


NUC123
Winbond W25Q16BVSIG
Flash Memory
VCC ↔ VCC
VCC ↔ HOLD (D3) / RESET
VCC ↔ WP (D2)
VSS ↔ VSS
SPI2_SS0 (PD.0) ↔ CS
SPI2_CLK (PD.1) → CLK
SPI2_MISO (PD.2) ← DO
SPI2_MOSI (PD.3) ← DI
USB ↔ PC (USB Host)

4 目錄資訊

- EC_NUC123_USBD_MSC_SPI_Flash_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

5 如何執行範例程式

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 NUC123_USBD_MSC_SPI_Flash.uvprojx。
2. 進入編譯模式介面
   a. 編譯
   b. 下載代碼至記憶體
   c. 進入 / 離開除錯模式
3. 進入除錯模式介面
   a. 執行代碼

6 修訂紀錄

Date	Revision	Description
Sep. 30, 2019	1.00	1. 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.