透過 MSC裝置發送自定義命令

NuMicro® 32位元系列微控制器範例代碼介紹

文件資訊

代碼簡述	本範例代碼使用 M032 透過 MSC 裝置發送自定義命令
BSP 版本	M032_Series_BSP_CMSIS_V3.05.000
開發平台	NuMaker-M032KG V1.0

1. 概述

一般來說必須透過 Host 端的工具才能透過 USB 傳送自定義的命令，但不是所有人都有能力開發這樣的工具。本範例提供不需要 Host 端工具就可以傳送自定義的命令的方案，將 M032 設定為虛擬的隨身碟，並透過對此隨身碟讀寫檔案來達到傳送自定義的命令的目的。

1.1 原理

將 M032 設定為以 FAT12 為檔案系統的隨身碟，再寫入特定內容檔案的方式傳送自定義命令，後續的章節會介紹 FAT 檔案系統與自定義命令控制流程。

1.1.1 FAT 檔案系統

FAT系統包含啟動磁區 (Boot Sector)、檔案配置表(File Allocation Table)、目錄表(Root Directory) 與檔案內容。圖 1-1為FAT系統檔案示意圖。

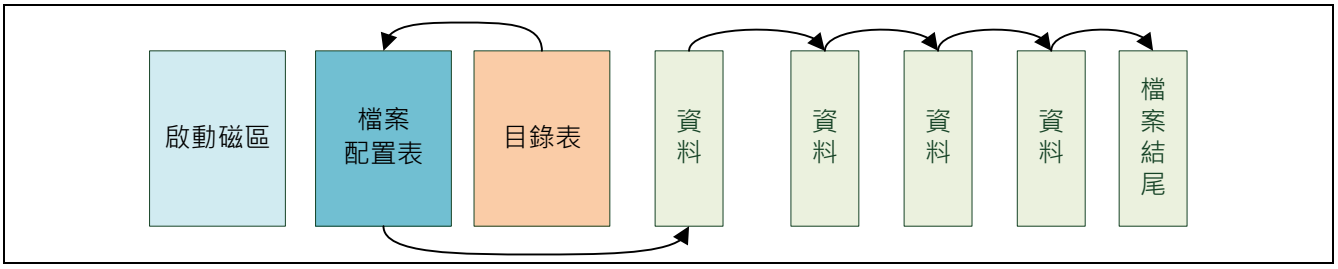

圖 1-1 FAT 系統檔案示意圖

當寫入檔案到FAT系統磁碟時，傳輸的檔案包含檔案配置表、目錄表與檔案內容。我們可以透過磁區的位置來判斷寫入資料代表的意義。

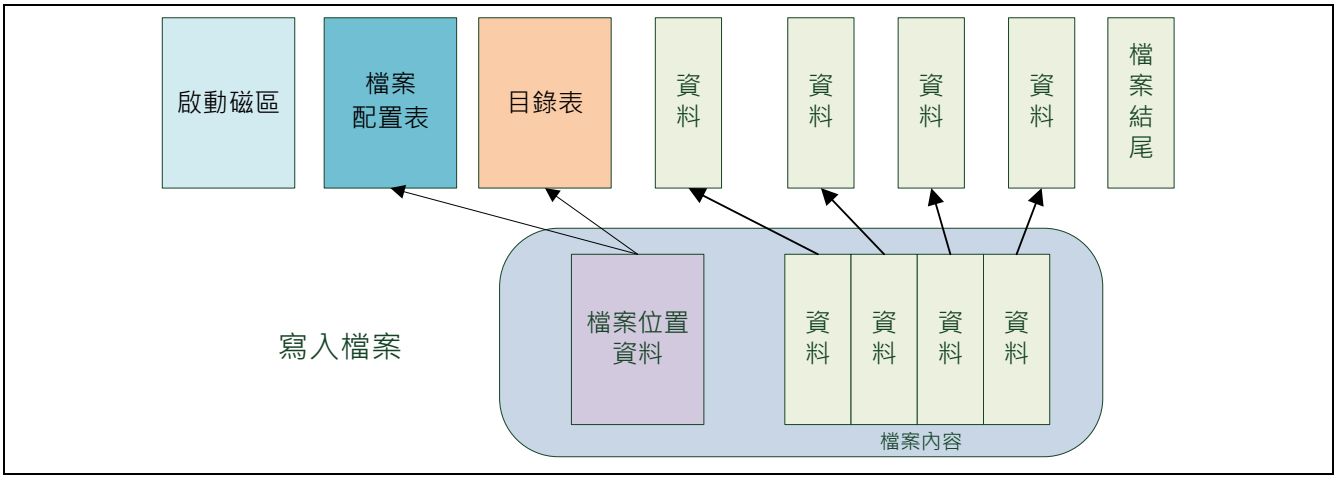

圖 1-2 寫入檔案至 FAT 系統檔案

1.1.1.1 啟動磁區介紹

啟動磁區是 FAT 檔案系統的第一個磁區，包含的系統類型、大小、位置與資料格式等資訊。表 1-1 為本範例使用的啟動磁區設定。

位元組(10進制)	值 (16進制)	功能
00 ~ 02	EB 3C 90	跳轉指令
03 ~ 10	4D 53 44 4F 53 35 2E 30	OEM名稱
11 ~ 12	00 02	每個磁區的位元組數
13	01	每叢集磁區數
14 ~ 15	06 00	保留磁區數
16	02	檔案配置表數目
17 ~ 18	00 02	最大根目錄條目個數
19 ~ 20	80 00	總磁區數
21	F8	媒介描述
22 ~ 23	01 00	每個檔案配置表的磁區
24 ~ 25	01 00	每磁軌的磁區
26 ~ 27	01 00	磁頭數
28 ~ 31	00 00 00 00	隱藏磁區
32 ~ 35	00 00 00 00	總磁區數
36	80	物理驅動器個數
37	00	未使用
38	29	簽章
39 ~ 42	2F 44 83 7A	版本號 (7A83-442F)
43 ~ 53	4E 4F 20 4E 41 4D 45 20 20 20 20	卷標 ("NO NAME ")
54 ~ 61	46 41 54 31 32 20 20 20	FAT檔案系統類型 ("FAT12 ")
62 ~ 509	[snip]	未使用

510 ~ 511	55 AA	磁區結束符 (0xaa55)

表 1-1 啟動磁區範例

1.1.1.2 磁區配置

表 1-2 為啟動磁區 (表 1-1) 設定的磁區配置

- 啟動磁區佔用 1 個磁區 (磁區 0，512 位元組)
- 檔案配置表起始磁區為磁區 6 (位元組 14 & 15)
- 兩組檔案配置表 (位元組 16)，每組 1 個磁區 (512 位元組, 位元組 22 & 23)
- 目錄表可包含 512 個條目(位元組 17 & 18)，每個 32 位元組 (0x4000 位元組，32 磁區)
- 資料磁區起始磁區為磁區 47 (6+1+1+32) ，#0x5000 (0xC00+0x200+0x200+0x4000)

磁區	位址	功能
0	0x00000 ~ 0x001FF	啟動磁區
6	0x00C00 ~ 0x00DFF	檔案配置表(主要)
7	0x00E00 ~ 0x00FFF	檔案配置表(備份)
8 ~ 39	0x01000 ~ 0x04FFF	目錄表
40 ~ 128	0x05000 ~ 0x0FFFF	資料磁區

表 1-2 磁區配置 (表 1-1)

1.1.2 自定義命令

本範例提供簡單的命令格式 (表 1-3) 與自定義命令集 (表 1-4)。

位元組	描述	註解
0	命令	
1	長度	0xE (不包含校驗和 & 保留欄位)
2 ~ 5	參數1	
6 ~ 9	參數2	
10 ~ 13	簽章	0x4343534D - "MSCC"
14 ~ 17	校驗和	
18 ~ 63	保留	

表 1-3 命令格式

描述	命令	參數1	參數2	資料	註解
GET_VERSION	0xD3	無	無	無	取得版本號碼
GET_STATUS	0xD4	無	無	無	取得狀態
GET_PARAM	0xD6	參數1	參數2	無	取得參數
READ_DATA	0xD7	資料長度	無	無	讀取資料
SET_PARAM	0xC5	參數1	參數2	有	設定參數
WRITE_DATA	0xC4	資料長度	無	有	寫入資料

表 1-4 自定義命令集

1.1.3 自定義命令傳輸

M032使用SRAM模擬啟動磁區(磁區0)、檔案配置表(磁區6~8)與目錄表(磁區8~39) 與檔案內容。SRAM與磁區配置如圖 1-3 (此虛擬磁碟會在程式重啟後恢復為預設的狀態)。 檔案內容用於自定義命令控制，所以並不會儲存於Flash或是SRAM。在Host讀取檔案時，M032會依照收到的自定義讀取命令回覆對應的資料；如果沒有收到自定義讀取命令，M032回覆全”0”的資料。

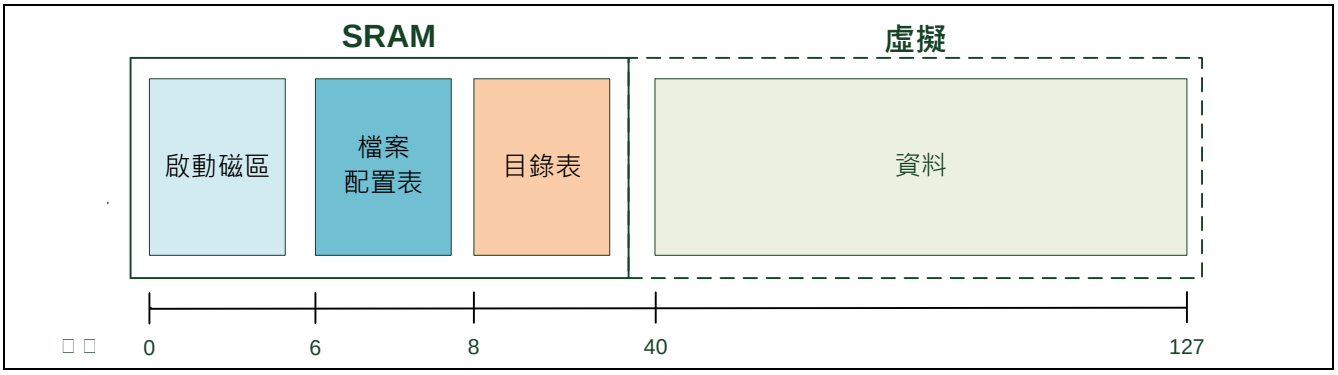

圖 1-3 記憶體與磁區配置範例

M032 連接上 Host 後顯示為容量為 44KB 無內容的隨身碟，像一般隨身碟一樣可以執行寫入、讀取與格式化。

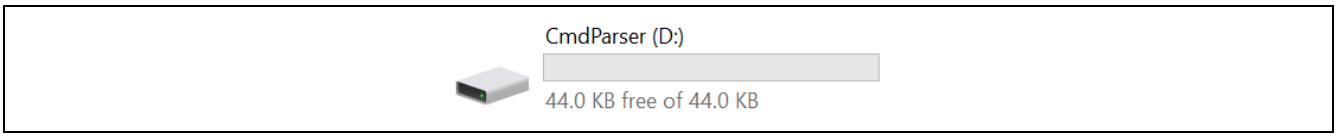

圖 1-4 M032 虛擬磁碟

1.1.3.1 寫入命令控制流程

以WRITE_DAT為例，檔案內容如圖 1-5所示。位元組0~17是命令區域，位元組18~4113是寫入的資料，檔案的大小為18 + 4K位元組。

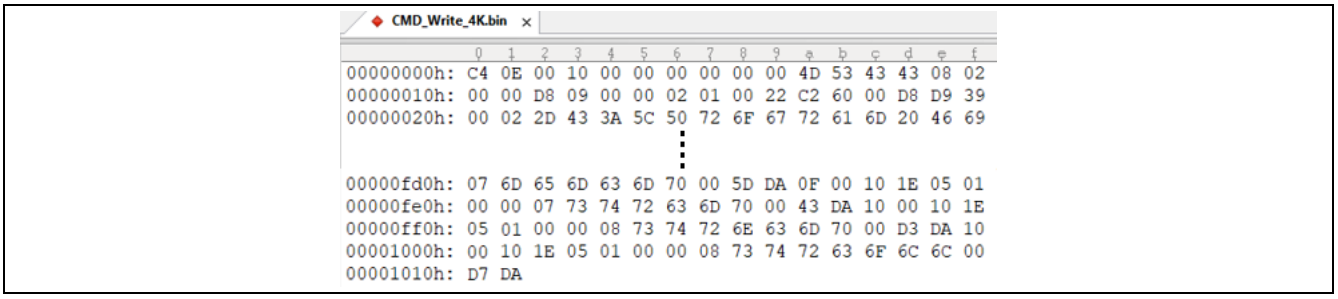

圖 1-5 命令檔案範例 – WRITE_DATA

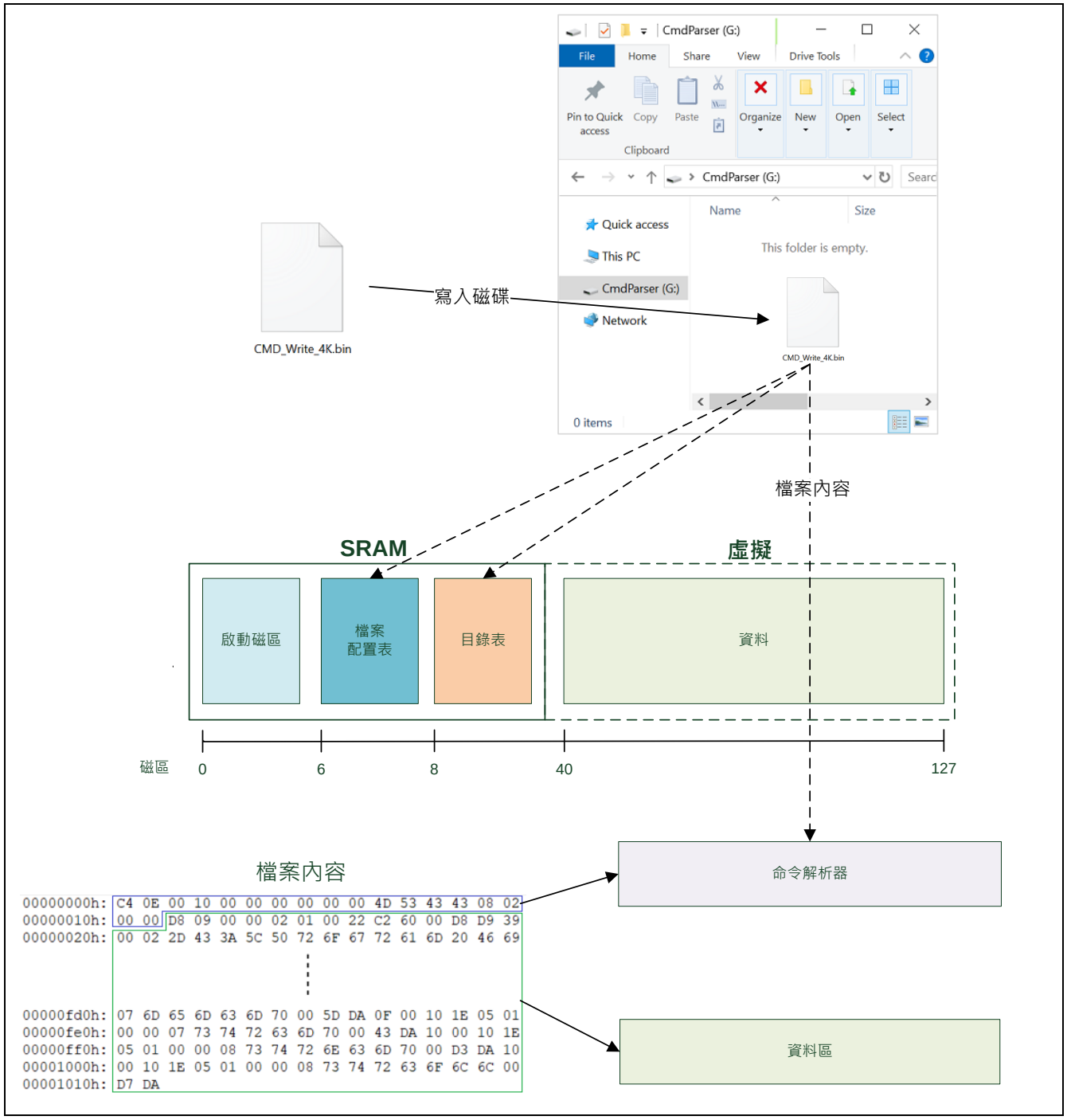

圖 1-6 WRITE_DATA 命令流程

當WRITE_DATA命令檔寫入虛擬磁碟，Host送出包含檔案配置表資料(磁區6與7)、目錄表(磁區8)與檔案內容(磁區位址大於40)的MSC寫入命令，如圖 1-7所示。M032會將磁區6、磁區7、磁區8寫入對應的SRAM位置，並將檔案內容(位元組0~17) 作為自定義命令解析。

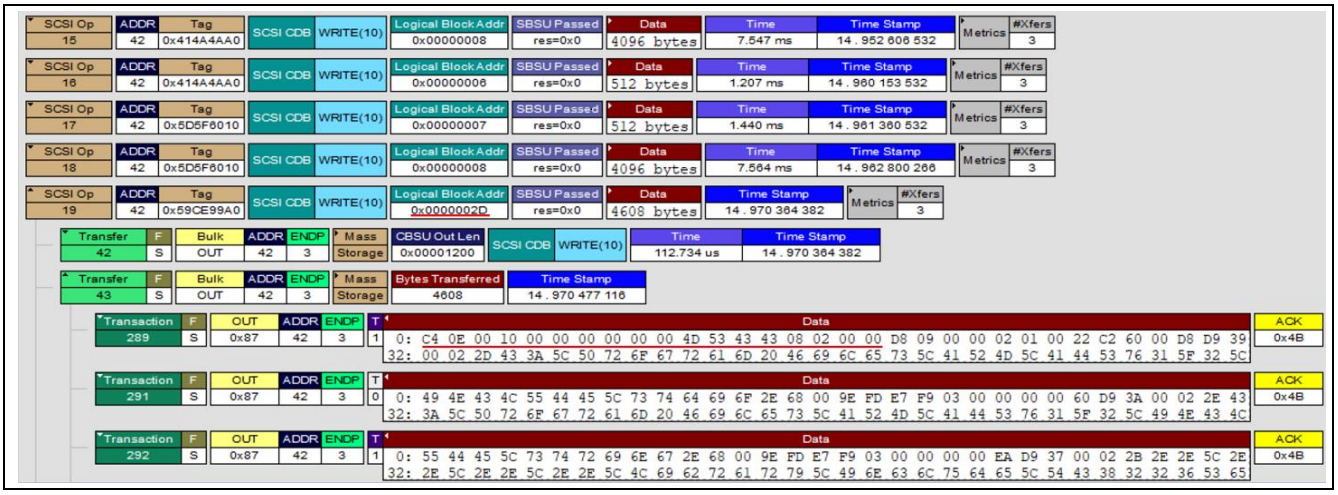

圖 1-7 USB 封包 – WRITE_DATA

1.1.3.2 讀取命令控制流程

讀取資料命令流程包含兩個步驟：寫入讀取命令檔 (圖 1-9)與從虛擬磁碟讀取檔案 (圖 1-9 圖 1-10)。M032 會記錄讀取命令檔所指的資料類型。以 READ_DATA 命令為例，檔案內容如圖 1-8 所示。

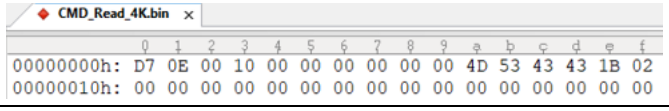

圖 1-8 命令檔案範例 – READ_DATA

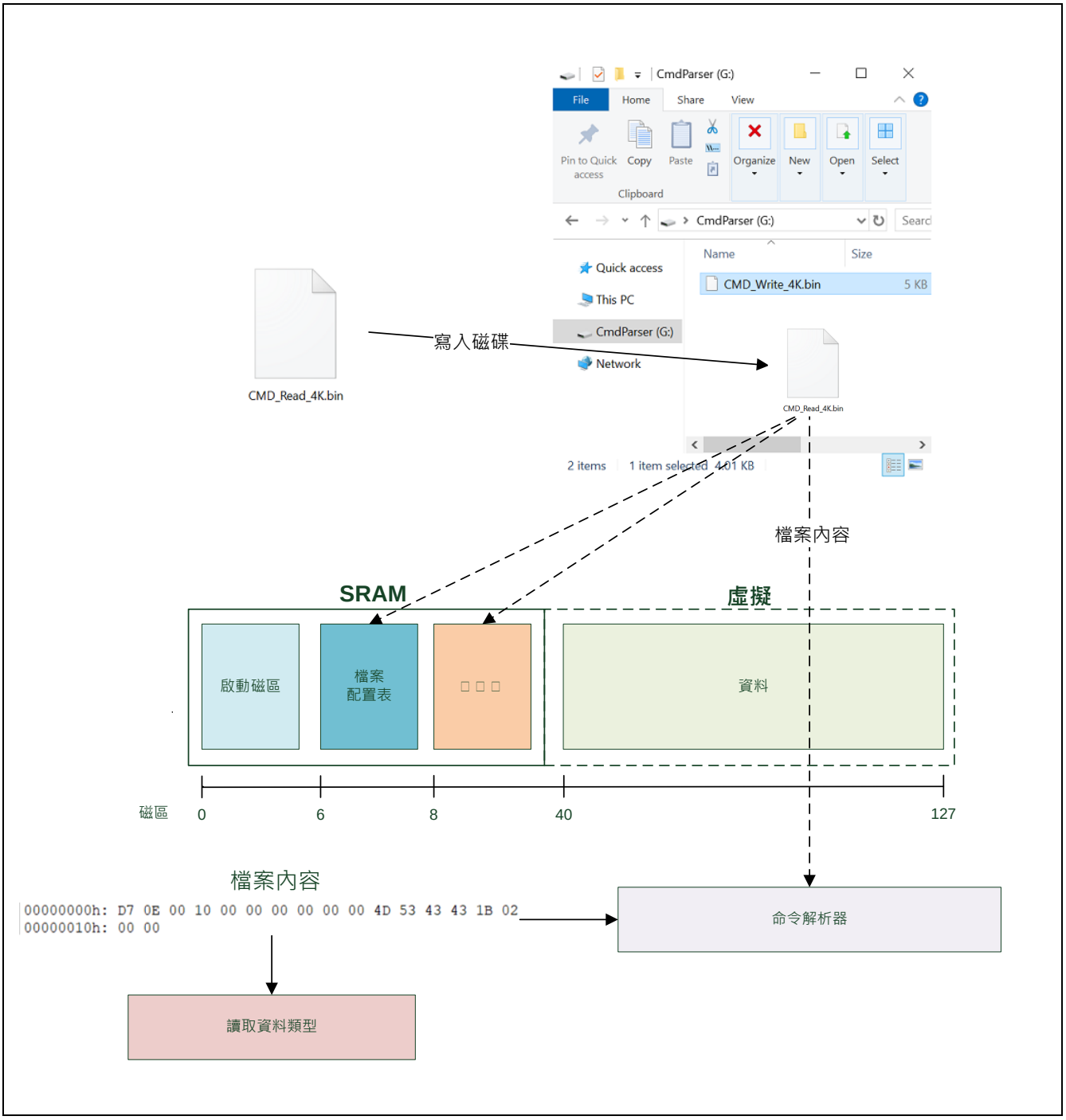

圖 1-9 寫入讀取命令檔

當 M032 收到 MSC SCSI 讀取命令時，會依照收到的自定義的讀取命令回覆對應的資料。

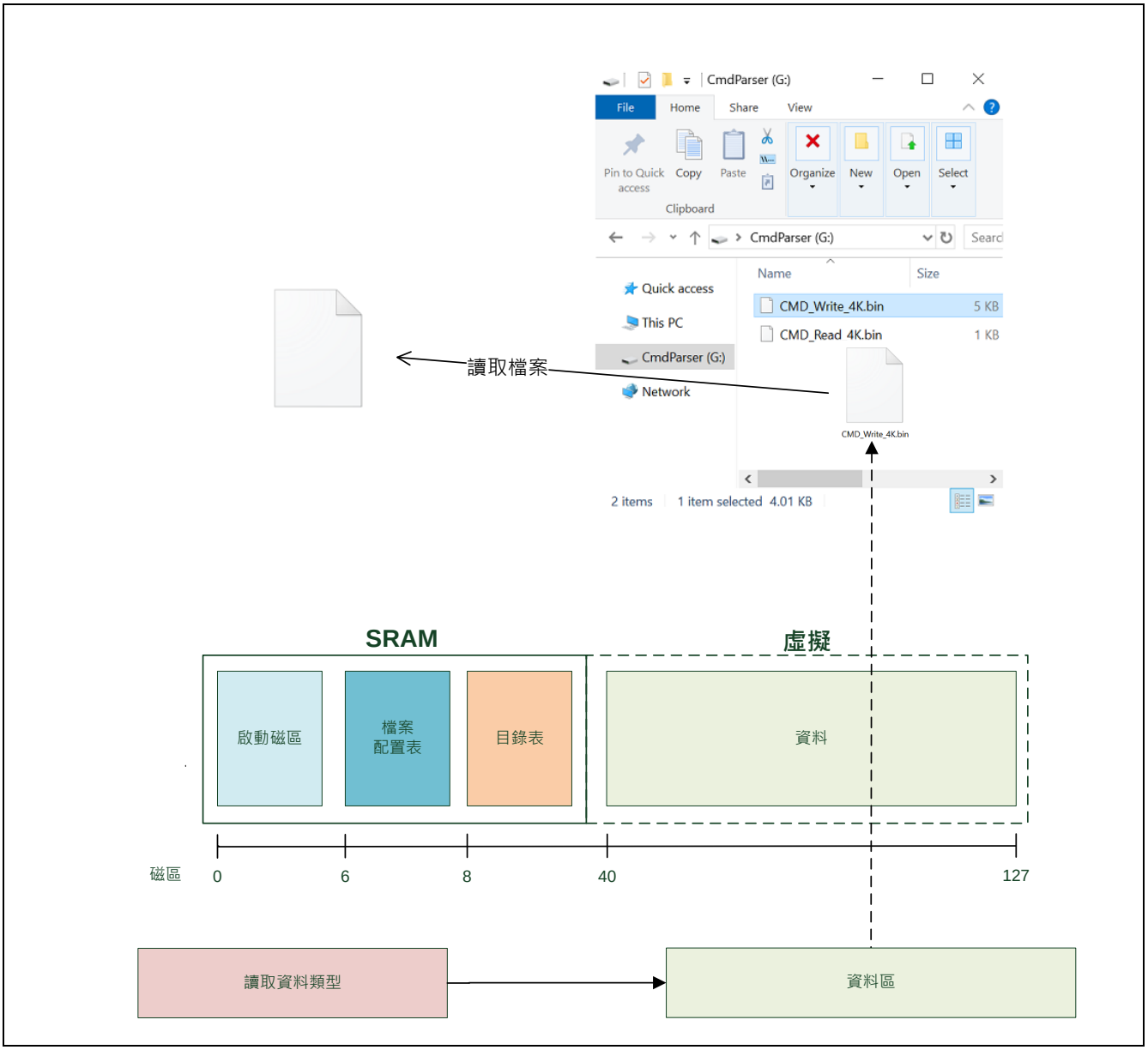

圖 1-10 讀取資料

必須在虛擬磁碟中讀取任一個檔案大小大於欲讀取資料量的檔案。Host 會依照檔案的大小得取資料 (圖 1-11 與圖 1-12)。作業系統會使用緩衝區的資料不實際從虛擬磁碟讀取資料，所以必須使用無緩衝的 I/O 複製方式複製檔案。

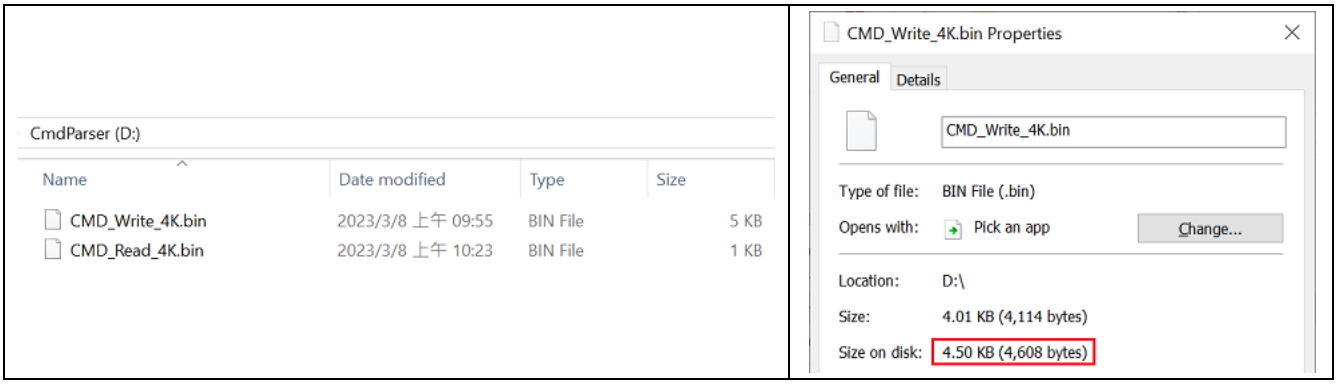

圖 1-11 讀取檔案大小

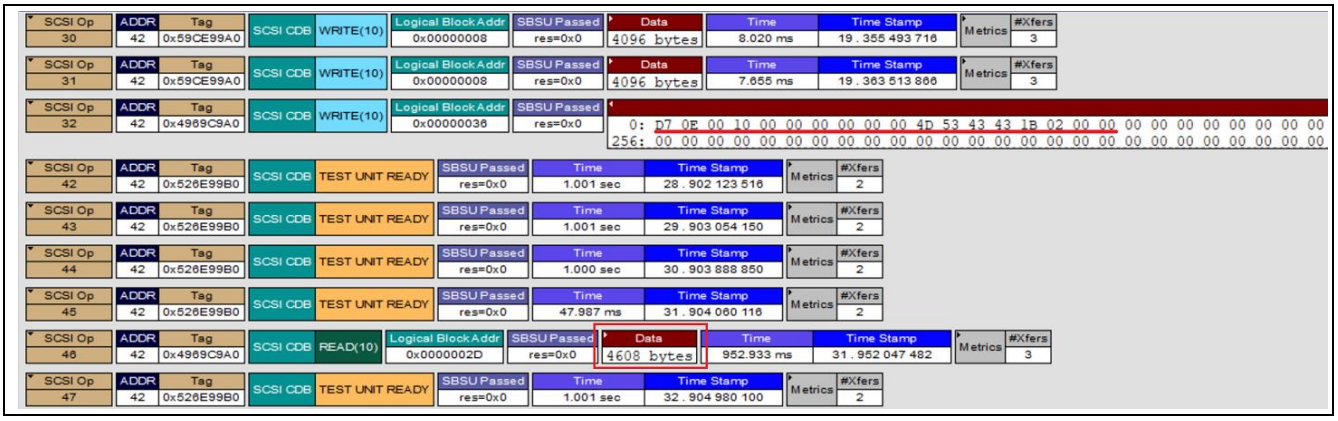

圖 1-12 USB 封包 - READ_DATA

1.2 執行結果

1.2.1 虛擬磁碟

M032連接上Host後顯示為容量為44KB無內容的隨身碟。

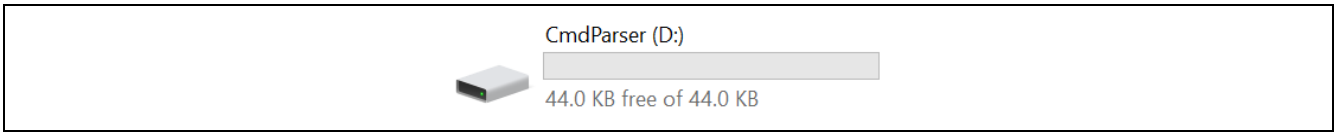

圖 1-13 M032 虛擬磁碟

1.2.2 SET_PARAM

將 SET_PARAM 命令檔寫入虛擬磁碟。

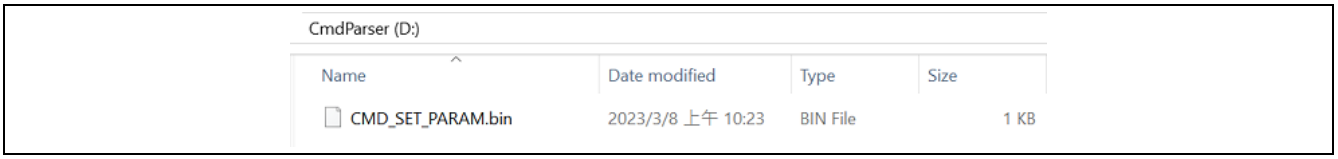

圖 1-14 寫入 SET_PARM 命令檔至虛擬磁碟

M032 收到 SET_PARM 命令後，顯示收到 SET_PARM 命令。

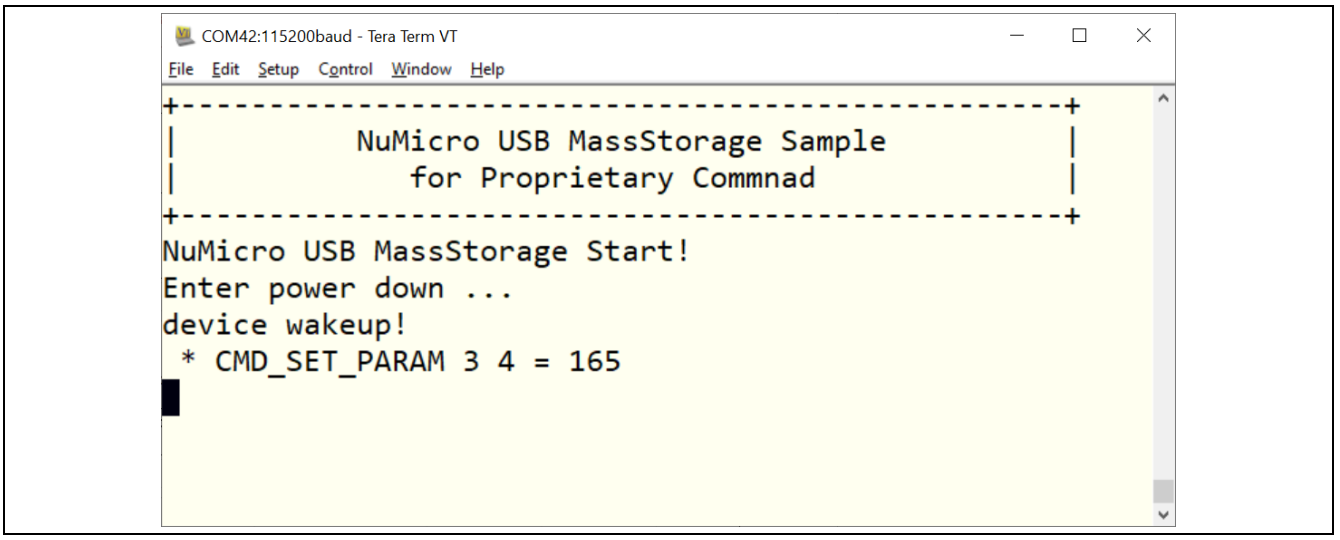

圖 1-15 M032 收到 SET_PARM 命令

1.2.3 WRITE_DATA

將 WRITE_DATA 命令檔寫入虛擬磁碟。

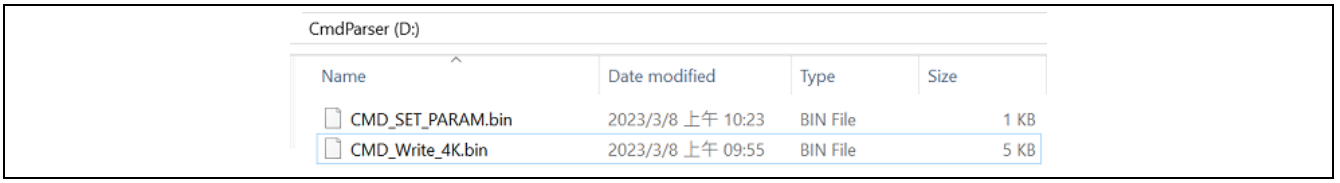

圖 1-16 寫入 WRITE_DATA 命令檔至虛擬磁碟

M032 收到 WRITE_DATA 命令後，顯示收到 WRITE_DATA 命令，顯示寫入進度直到 WRITE_DATA 命令完成。

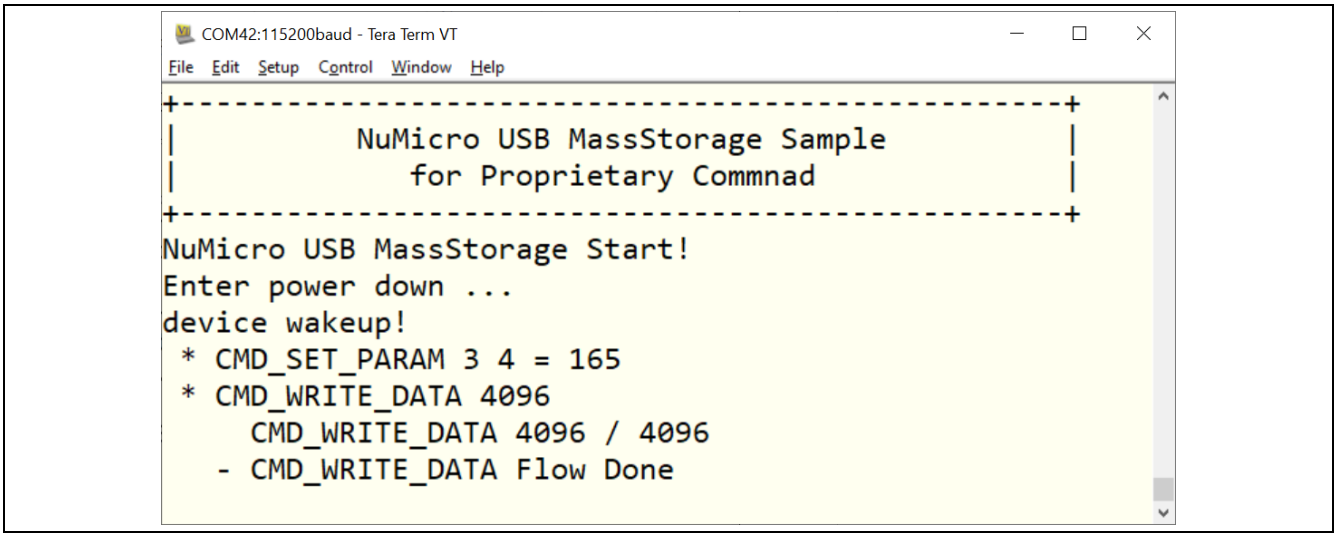

圖 1-17 M032 收到 WRITE_DATA 命令

1.2.4 READ_DATA

將 READ_DATA 命令檔寫入虛擬磁碟。

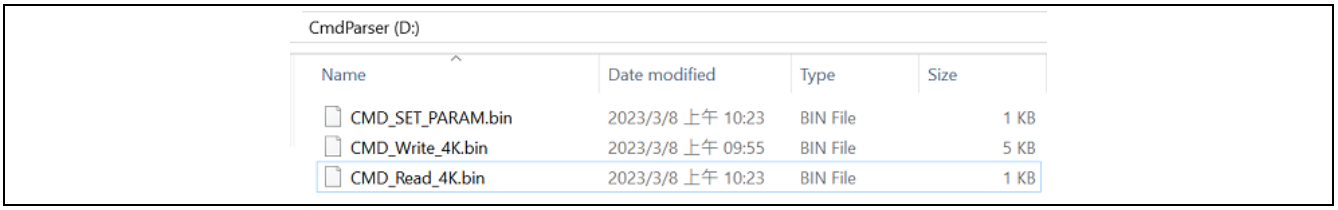

圖 1-18 寫入 READ_DATA 命令檔至虛擬磁碟

M032 收到 READ_DATA 命令後，顯示收到 READ_DATA 命令 (讀取 4KB 資料)，此時還沒開始回傳資料。

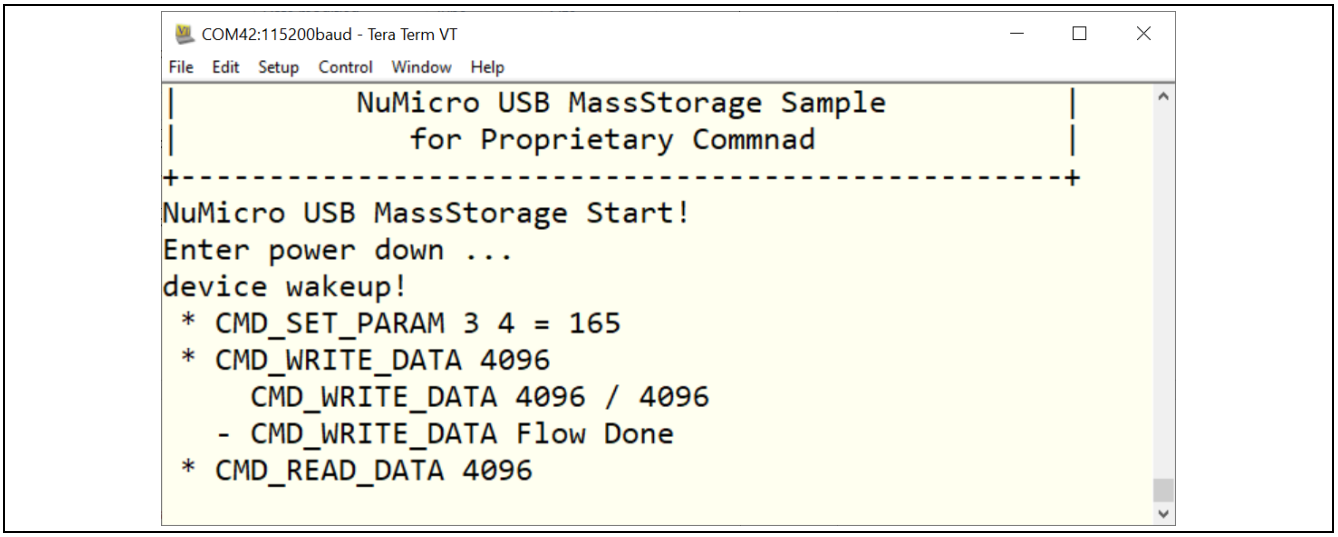

圖 1-19 M032 收到 READ_DATA 命令

從 M032 虛擬磁碟以無緩衝的方式讀取 CMD_Write_4K.bin (選取讀取檔案的條件是必須大於要讀取的資料量 - 4KB)，M032 顯示讀取進度直到 READ_DATA 命令完成。

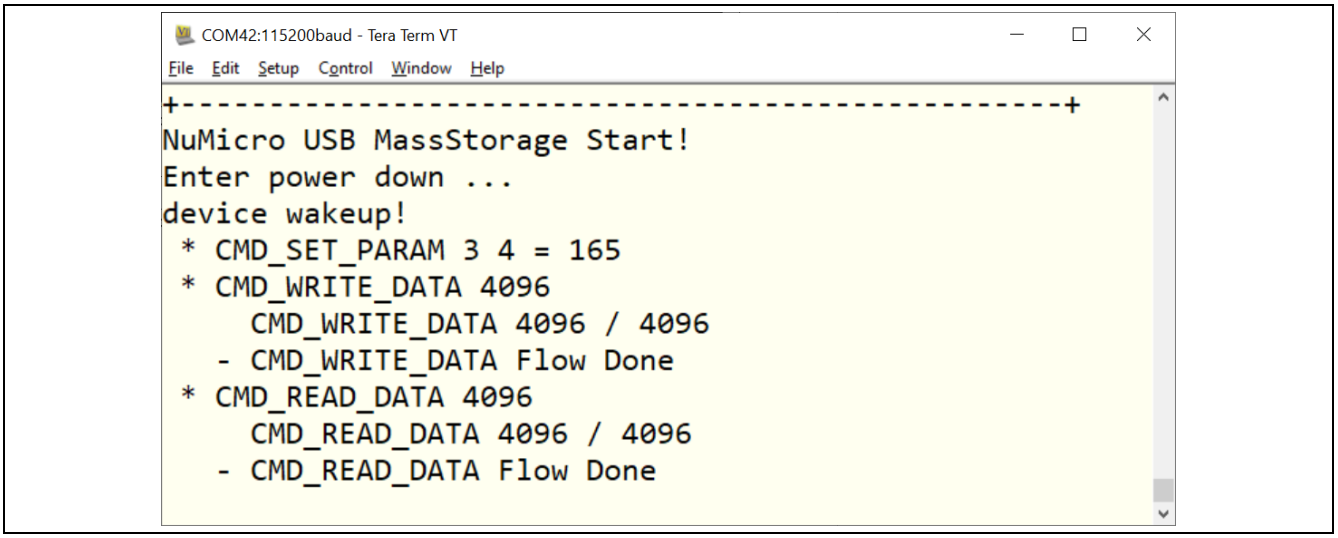

圖 1-20 READ_DATA 流程結束

與寫入資料比對與先前寫入的資料比對結果一致。

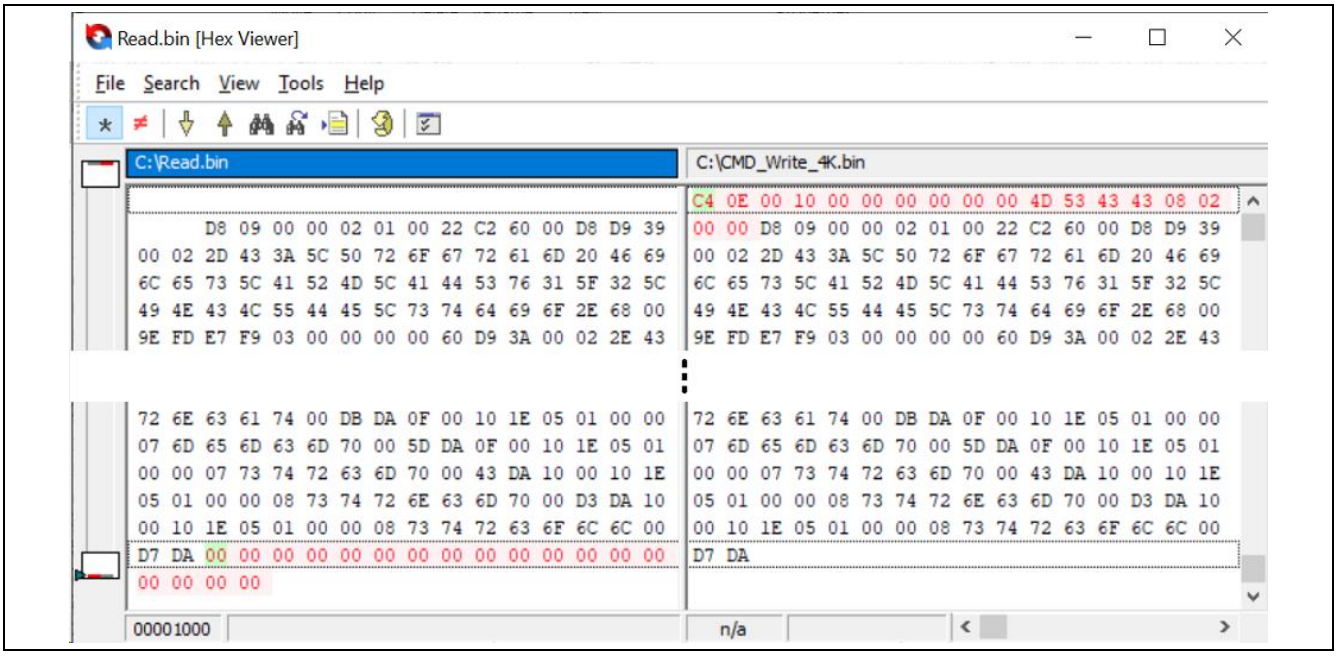

圖 1-21 比較結果

1.2.5 無緩衝的 I/O 複製檔案

1.2.5.1 Windows

Windows command - Xcopy

Xcopy	
參數	**敘述**
/j	使用無緩衝的 I/O 複製，建議使用於非常大的檔案。此功能首次支援於 Windows 7。

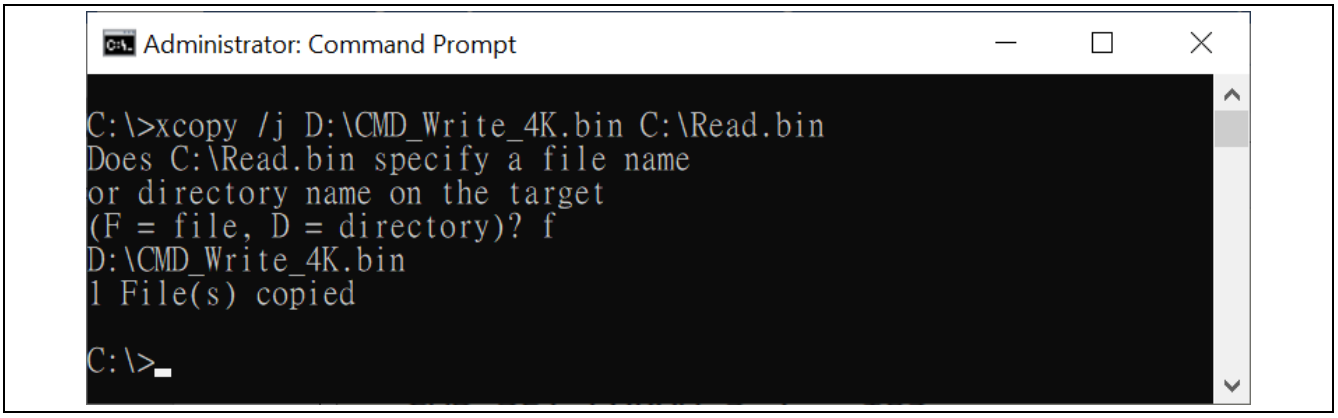

圖 1-22 使用 xcopy 以使用無緩衝的 I/O 複製檔案

執行檔CopyFile使用CreateFileA API搭配參數FILE_FLAG_NO_BUFFERING讀取檔案。此範例提供簡單的程式使用無緩衝的 I/O 複製檔案。

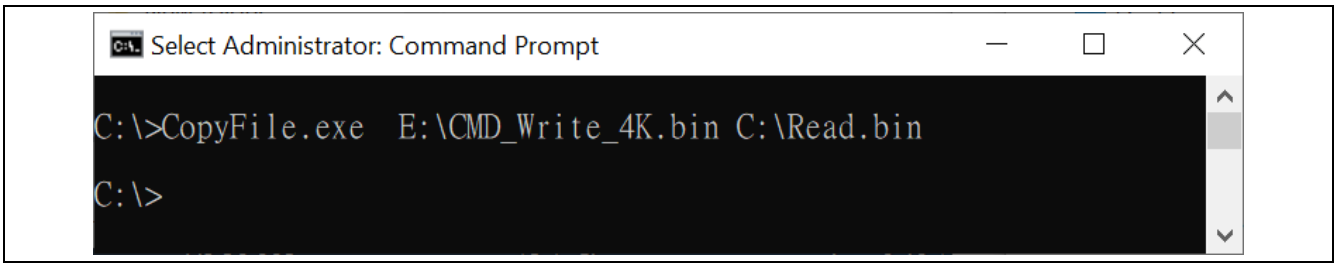

圖 1-23 使用 CopyFile 以使用無緩衝的 I/O 複製檔案

1.2.5.2 Linux

Linux command - dd

dd Command	
參數	**敘述**
iflg=direct	使用direct I/O的方式

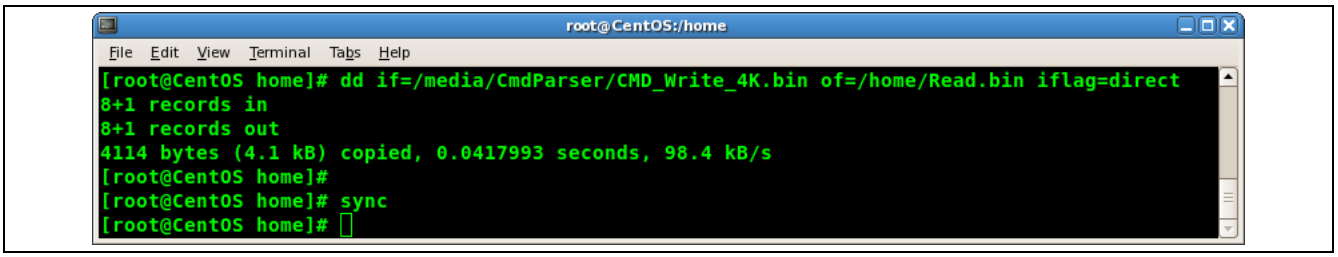

圖 1-24 使用 dd 以使用無緩衝的 I/O 複製檔案

2. 代碼介紹

2.1 虛擬磁碟配置

配置虛擬磁碟的內容，代碼位於 CmdParser.c，以下是虛擬磁碟的啟動磁區內容。

```
uint8_t u8BootSectorData[512] =
{
    /* Instructions to jump to boot code */
    0xEB, 0x3C, 0x90,
    /* Name string (MSDOS5.0) */
    0x4D, 0x53, 0x44, 0x4F, 0x53, 0x35, 0x2E, 0x30,
    /* Bytes/sector (0x0200 = 512) */
    0x00, 0x02,
    /* Sectors/cluster */
    0x01,
    /* Size of reserved area */
    (RSVD_SEC_CNT & 0xFF), (RSVD_SEC_CNT >> 8),
    /* Number of FATs */
    NUM_FAT,
    /* Byte 17 & 18 (Max. number of root directory entries) */
    (ROOT_ENT_CNT & 0xFF) , (ROOT_ENT_CNT >> 8),
    /* Byte 19 & 20 (Total number of sectors) */
    (SECTOR_CNT & 0xFF), (SECTOR_CNT >> 8),
    /* Media type (removable) */
    0xF8,
    /* Byte 22 & 23 (FAT size) */
    (FAT_SZ & 0xFF), (FAT_SZ >> 8),
    /* Sectors/track */
    0x01, 0x00,
    /* Number of heads */
    0x01, 0x00,
    /* Number of sector before partition */
    0x00, 0x00, 0x00, 0x00,
    /* Total number of sectors */
    0x00, 0x00, 0x00, 0x00,
    /* Drive number */
    0x80,
    /* Unused */
    0x00,
    /* Extended boot signature */
    0x29,
    /* Volume serial number */
    0x2F, 0x44, 0x83, 0x7A,
    /* Volume label - "NO NAME " */
    0x4E, 0x4F, 0x20, 0x4E, 0x41, 0x4D, 0x45, 0x20, 0x20, 0x20, 0x20,
    /* File system type label ("FAT12 ") */
    0x46, 0x41, 0x54, 0x31, 0x32, 0x20, 0x20, 0x20,
    /* Not used */
    ...
    /* Signature value (0xaa55) */
    0x55, 0xAA
```

```
};
```

配置虛擬磁碟內容的定義，代碼位於 CmdParser.h，修改”DISK_SIZE”就可以修改虛擬磁碟的大小。預設的磁區配置可能會與作業系統格式化的配置不同導致執行程序不正常運作，可以在系統格式化後，使用定義 “__FAT_INFO__”來確認磁區配置是否需要修正，可能需要修正的是 RSVD_SEC_CNT 與 FAT_SZ 的值。

```
//#define __FAT_INFO__

#define DISK_SIZE               (64 * 1024)

#define RSVD_SEC_CNT            6
#define FAT_SZ                  1

#define BYTE_PER_SEC            512
#define ROOT_ENT_CNT            512
#define ROOT_ENT_SEC_CNT        ((32 * ROOT_ENT_CNT) / BYTE_PER_SEC)
#define NUM_FAT                 2
#define FAT_SEC                 RSVD_SEC_CNT
#define FAT_SEC_ADDR            (RSVD_SEC_CNT * BYTE_PER_SEC)
#define ROOT_SEC_ADDR           (FAT_SEC_ADDR + FAT_SZ * NUM_FAT * BYTE_PER_SEC)
#define DATA_SEC_ADDR           (ROOT_SEC_ADDR + ROOT_ENT_SEC_CNT * BYTE_PER_SEC)
#define SECTOR_CNT              (DISK_SIZE / BYTE_PER_SEC)
```

顯示預設磁區配置與目前磁區配置的內容，代碼位於 CmdParser.c。

```
void DataWrite(uint32_t addr, uint32_t size, uint32_t buffer)
{
    …
    if (addr == 0x00000000)
    {
#ifdef __FAT_INFO__
        if(g_ShowFat && GET_FAT_SZ != 0)
        {
            printf("\n");
            printf("Default FAT_SEC        %08X ", FAT_SEC);
            printf("Current FAT_SEC       %08X\n", GET_FAT_SEC);
            printf("Default FAT_SZ         %08X \n", FAT_SZ);
            printf("Current FAT_SZ        %08X\n", GET_FAT_SZ);
            printf("Default ROOT_SEC_ADDR  %08X ", ROOT_SEC_ADDR);
            printf("Current ROOT_SEC_ADDR %08X\n", GET_ROOT_SEC_ADDR);
            printf("Default DATA_SEC_ADDR  %08X ",DATA_SEC_ADDR);
            printf("Current DATA_SEC_ADDR %08X\n", DATA_SEC_ADDR);
            g_ShowFat = 0;
        }
#endif
        USBD_MemCopy((uint8_t *)pu32Buf, u8BootSectorData, sizeof(u8BootSectorData));
    }
    …
}
```

2.2 模擬 FAT 資訊

模擬虛擬磁碟讀寫啟動磁區、檔案配置表與目錄表，代碼位於 CmdParser.c。

```
void DataRead(uint32_t addr, uint32_t size, uint32_t buffer)
{
    uint32_t i;

    uint32_t * pu32Buf = (uint32_t *)buffer;

    memset((uint8_t *)pu32Buf, 0, STORAGE_BUFFER_SIZE);

    if (addr == 0x00000000)  /* Boot Sector */
    {
        …
        USBD_MemCopy((uint8_t *)buffer, u8BootSectorData, sizeof(u8BootSectorData));
    }
    else
    {
        if (addr >= ROOT_SEC_ADDR && addr < DATA_SEC_ADDR) /* Root Directory */
            USBD_MemCopy( (uint8_t *) buffer, u8DirData + (addr - ROOT_SEC_ADDR), 512);
        else if (addr >= FAT_SEC_ADDR && addr < ROOT_SEC_ADDR)  /* File Allocation Table */
            USBD_MemCopy((uint8_t *) buffer, u8FAT + (addr - FAT_SEC_ADDR), 512);
        else if(addr >= DATA_SEC_ADDR)  /* Data */
        {
            …
        }
    }
}

void DataWrite(uint32_t addr, uint32_t size, uint32_t buffer)
{
    /* This is low level write function of USB Mass Storage */
    if(addr >= DATA_SEC_ADDR)  /* Data */
    {
        …
    }
    else if (addr == 0x00000000)
    {
        USBD_MemCopy(u8BootSectorData,(uint8_t *) buffer, 512);
        …
    }
    else if (addr >= ROOT_SEC_ADDR && addr < DATA_SEC_ADDR) /* Root Directory */
        USBD_MemCopy(u8DirData + (addr - ROOT_SEC_ADDR), (uint8_t *)buffer, 512);
    else if (addr >= FAT_SEC_ADDR && addr < ROOT_SEC_ADDR) /* File Allocation Table */
        USBD_MemCopy(u8FAT + (addr - FAT_SEC_ADDR), (uint8_t *)buffer, 512);
}
```

檔案位置的訊息儲存於SRAM，所以此虛擬磁碟會在程式重啟後恢復為預設的狀態。

2.3 自訂命令定義

自定義的命令的資料結構與指令集，代碼位於 CmdParser.h。

```
/* Definitions for Vendor Commands */
#define CMD_SIGNATURE       0x4343534D
#define CMD_NONE            0x00
#define CMD_GET_VER         0xD3
#define CMD_GET_STS         0xD4
#define CMD_WRITE_DATA      0xC4
#define CMD_SET_PARAM       0xC5
#define CMD_GET_PARAM       0xD6
#define CMD_READ_DATA       0xD7

#ifdef __GNUC__
typedef struct __attribute__((__packed__))
{
    uint8_t  u8Cmd;
    uint8_t  u8Size;
    uint32_t u32Arg1;
    uint32_t u32Arg2;
    uint32_t u32Signature;
    uint32_t u32Checksum;
    uint8_t  u8Data[50];
}
CMD_T;
#else
typedef __packed struct
{
    uint8_t u8Cmd;
    uint8_t u8Size;
    uint32_t u32Arg1;
    uint32_t u32Arg2;
    uint32_t u32Signature;
    uint32_t u32Checksum;
    uint8_t  u8Data[50];
} CMD_T;
#endif
```

2.4 自定義命令流程控制

MSC SCIC 寫入命令流程控制代碼位於 CmdParser.c，需要修改命令集只需要修改 ProcessCommand，另外在 WRITE_DATA 結束後加上需要的程序。

```
void DataWrite(uint32_t addr, uint32_t size, uint32_t buffer)
{
    /* This is low level write function of USB Mass Storage */
    if(addr >= DATA_SEC_ADDR)  /* Data */
    {
        /* Check if it is in the data phase of WRITE_DATA command */
        if((g_u8Cmd == CMD_WRITE_DATA) &&  (g_u32DataCnt <= g_u32TotalCnt))
        {
            if(g_u32BytesInBuf + size >= BUFFER_SIZE)
                size = size - (g_u32BytesInBuf + size - BUFFER_SIZE);

            memcpy((void *)&g_u8WorkingBuff[g_u32BytesInBuf], (void *)buffer, size);

            g_u32BytesInBuf += size;

            g_u32DataCnt = g_u32DataCnt + size;

            printf("     CMD_WRITE_DATA %d / %d\r", g_u32DataCnt, g_u32TotalCnt);

            /* Write command complete! */
            if(g_u32DataCnt >= g_u32TotalCnt)
            {
                g_u8Cmd = CMD_NONE;
                printf("\n    - CMD_WRITE_DATA Flow Done\n");
            }
            else
            {
                /* Update command status */
                g_u8Cmd = CMD_WRITE_DATA;
            }
        }
        Else
        {
            /* Check and process the command packet */
            if(ProcessCommand(buffer, size))
            {
                /* Unknown command - There is no Command Signature */
            }
        }
    }
    …
}
```

MSC SCSI 讀取命令流程控制代碼位於 CmdParser.c，M032 會依照收到的自定義讀取命令會設定變數”g_u8ReadType”，並在後續收到 MSC 讀取命令時回覆對應的資料。

```
void DataRead(uint32_t addr, uint32_t size, uint32_t buffer)
{
    …

    if (addr == 0x00000000) /* Boot Sector */
    {
        …
    }
    Else
    {
        …
        else if(addr >= DATA_SEC_ADDR) /* Data */
        {
            /* Check if it is in data phase of READ_DATA command */
            if((g_u8ReadType == CMD_READ_DATA) &&  (g_u32DataCnt <= g_u32TotalCnt))
            {
                printf("     READ %d / %d\r", g_u32DataCnt, g_u32TotalCnt);

                memcpy((void *)buffer, (void *) (g_u8WorkingBuff + g_u32DataCnt), size);

                g_u32DataCnt += size;

                /* Process the data phase of read command */
                if(g_u32DataCnt >= g_u32TotalCnt)
                {
                    /* The data transfer is complete. */
                    g_u8ReadType = CMD_NONE;
                    printf("     CMD_READ_DATA %d / %d\r", g_u32DataCnt, g_u32TotalCnt);
                    printf("\n   - CMD_READ_DATA Flow Done\n");
                }
            }
            else
            {
                if(g_u8ReadType == CMD_GET_STS)
                {
                    printf("   - READ - CMD_GET_STS\n");
                    pu32Buf[0] = 0xA5A5A5A5;
                    g_u8ReadType = CMD_NONE;
                }
                else if(g_u8ReadType == CMD_GET_VER)
                {
                    printf("   - READ - CMD_GET_VER\n");
                    pu32Buf[0] = 0x20230207;
                    g_u8ReadType = CMD_NONE;
                }
                else if(g_u8ReadType == CMD_GET_PARAM)
                {
                    printf("   - READ - CMD_GET_PARAM\n");
                    pu32Buf[0] = 0x12345678;
                    g_u8ReadType = CMD_NONE
                }
```

```
                else
                {
                    for(i=0;i<STORAGE_BUFFER_SIZE/4;i++)
                        pu32Buf[i] = 0x00000000;
                }
            }
        }
    }
}
```

自定義的命令的主要處理程序，代碼位於 CmdParser.c。

```
int32_t ProcessCommand(uint32_t buffer, uint32_t size)
{
    uint32_t u32sum;

    pCmd = (CMD_T *) buffer;

    /* Check size */
    if((pCmd->u8Size > sizeof(gCmd)))
        return -1;

    /* Check signature */
    if(pCmd->u32Signature != CMD_SIGNATURE)
        return -1;

    /* Calculate checksum & check it*/
    u32sum = CalCheckSum((uint8_t *)pCmd, pCmd->u8Size);

    if(u32sum != pCmd->u32Checksum)
    {
        printf("Checksum is wrong - 0x%08X 0x%08X\n", u32sum,pCmd->u32Checksum);
        return -1;
    }
    switch(pCmd->u8Cmd)
    {
        case CMD_GET_STS:
        {
            printf(" * CMD_GET_STS\n");
            g_u8ReadType = CMD_GET_STS;
            g_u8Cmd = CMD_NONE;
            break;
        }

        case CMD_GET_VER:
        {
            printf(" * CMD_GET_VER\n");
            g_u8ReadType = CMD_GET_VER;
            g_u8Cmd = CMD_NONE;
            break;
        }

        case CMD_SET_PARAM:
        {
            printf(" * CMD_SET_PARAM %d %d = %d\n",pCmd->u32Arg1, pCmd->u32Arg2,
```

```
                    pCmd->u8Data[0]);
            g_u8Cmd = CMD_NONE;
            break;
        }

        case CMD_GET_PARAM:
        {
            printf(" * CMD_GET_PARAM\n");
            g_u8ReadType = CMD_GET_PARAM;
            g_u8Cmd = CMD_NONE;
            break;
        }

        case CMD_WRITE_DATA:
        {
            g_u8Cmd = pCmd->u8Cmd;
            g_u32TotalCnt = pCmd->u32Arg1;
            printf(" * CMD_WRITE_DATA %d\n",g_u32TotalCnt);
            g_u32BytesInBuf = size - 18;
            g_u32DataCnt = g_u32BytesInBuf;
            g_u8WorkingBuff = g_u8Buff;
            memcpy((void *)g_u8WorkingBuff, (void *)(buffer+18), g_u32BytesInBuf);
            break
        }

        case CMD_READ_DATA:
        {
            g_u8Cmd = pCmd->u8Cmd;
            g_u32TotalCnt = pCmd->u32Arg1;
            g_u32DataCnt = 0;
            printf(" * CMD_READ_DATA %d\n",g_u32TotalCnt);
            g_u8ReadType = CMD_READ_DATA;
            g_u8WorkingBuff = g_u8Buff;
            break;
        }

        default:
            return -1;
    }
    return 0;
}
```

3. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - M032_Series_BSP_CMSIS_V3.05.000
- IDE 版本
  - Keil uVersion 5.38

3.2 硬體需求

- 電路元件
  - NuMaker-M032KG V1.0

4. 目錄資訊

EC_M032_ MSC_Vendor_Cmd_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
Commands	Command files of example code
MSC_Vendor_Cmd	Keil project source code
WindowsCopyFile	Windows copy file tool

圖 4- 1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，按兩下 *EC_M032_MSC_Vendor_Cmd.uvprojx*。
2. 進入編譯模式介面
   - 編譯
   - 下載代碼至記憶體
   - 進入 / 離開模擬模式
3. 進入模擬模式介面
   - 執行代碼

6. 修訂紀錄

Date	Revision	Description
2023.03.10	1.00	初始發佈。

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.