透过 MSC装置发送自定义命令

NuMicro® 32位系列微控制器范例代码介绍

文件信息

代码简述	本范例代码使用 M032 透过 MSC 装置发送自定义命令
BSP 版本	M032_Series_BSP_CMSIS_V3.05.000
开发平台	NuMaker-M032KG V1.0

1. 概述

一般来说必须透过 Host 端的工具才能透过 USB 传送自定义的命令，但不是所有人都有能力开发这样的工具。本范例提供不需要 Host 端工具就可以传送自定义的命令的方案，将 M032 设定为虚拟的磁盘，并透过对此磁盘读写档案来达到传送自定义的命令的目的。

1.1 原理

将 M032 设定为以 FAT12 为文件系统的磁盘，再写入特定内容档案的方式传送自定义命令，后续的章节会介绍 FAT 文件系统与自定义命令控制流程。

1.1.1 FAT 文件系统

FAT 系统包含启动扇区(Boot Sector)、文件分配表(File Allocation Table)、目录表(Root Directory)与档案内容。图 1-1 为 FAT 系统档案示意图。

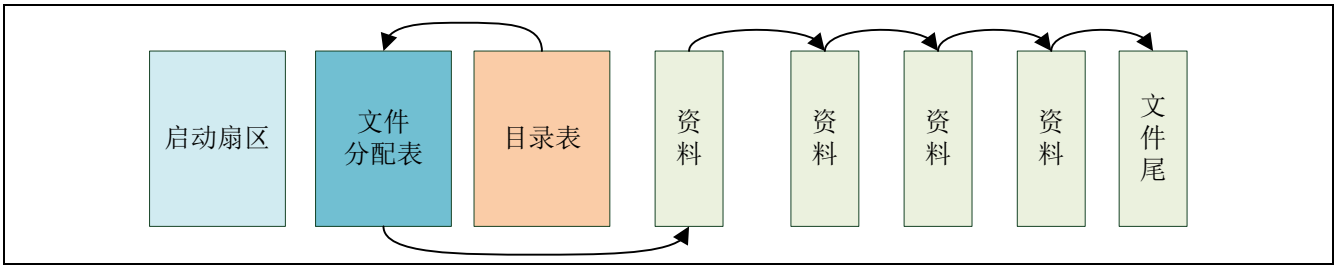

图 1-1 FAT 系统档案示意图

当写入档案到 FAT 系统磁盘时，传输的档案包含文件分配表、目录表与档案内容。我们可以透过扇区的位置来判断写入数据代表的意义。

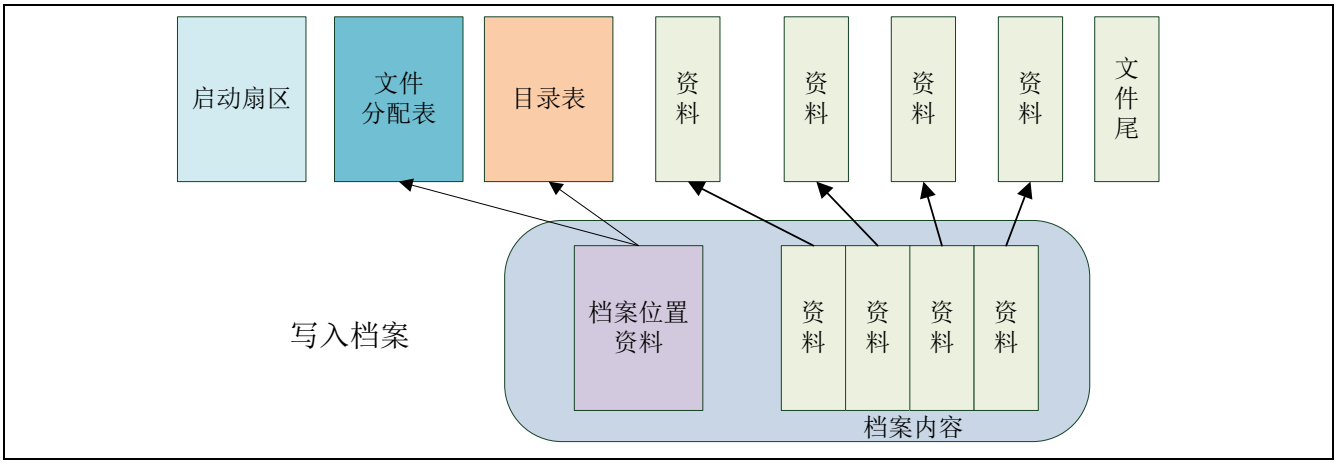

图 1-2 写入档案至 FAT 系统档案启动扇区介绍

1.1.1.1 啟動磁區介紹

启动扇区是 FAT 文件系统的第一个扇区，包含的系统类型、大小、位置与数据格式等信息。表 1-1 为本范例使用的启动扇区设定。

字节(10进制)	值 (16进制)	功能
00 ~ 02	EB 3C 90	转跳指令
03 ~ 10	4D 53 44 4F 53 35 2E 30	OEM名称
11 ~ 12	00 02	扇区字节数
13	01	每簇扇区数
14 ~ 15	06 00	保留扇区数
16	02	FAT数
17 ~ 18	00 02	根目录项数
19 ~ 20	80 00	小扇区数
21	F8	媒体描述符
22 ~ 23	01 00	每FAT扇区数
24 ~ 25	01 00	每道扇区数
26 ~ 27	01 00	磁头数
28 ~ 31	00 00 00 00	隐藏扇区数
32 ~ 35	00 00 00 00	大扇区数
36	80	物理驱动器号
37	00	未使用
38	29	扩展引导标签
39 ~ 42	2F 44 83 7A	卷序号 (7A83-442F)
43 ~ 53	4E 4F 20 4E 41 4D 45 20 20 20 20	卷标 ("NO NAME ")
54 ~ 61	46 41 54 31 32 20 20 20	文件系统类型 ("FAT12 ")
62 ~ 509	[snip]	未使用

510 ~ 511	55 AA	扇区结束标示符 (0xaa55)

表 1-1 启动扇区范例

1.1.1.2 扇区配置

表 1-2 为启动扇区 (表 1-1) 设定的扇区配置

- 启动扇区占用 1 个扇区 (扇区 0 ， 512 字节)
- 文件分配表起始扇区为扇区 6 (字节 14 & 字节 15)
- 两组文件分配表 (字节 16)，每组 1 个扇区 (字节 22 & 23)
- 目录表可包含 512 个项目 (字节 17 & 18)，每个 32 字节 (0x4000 字节，即 32 扇区)
- 数据扇区起始扇区为扇区 47 (6+1+1+32)， 0x5000 (0xC00+0x200+0x200+0x4000)

扇区	地址	功能
0	0x00000 ~ 0x001FF	启动扇区
6	0x00C00 ~ 0x00DFF	文件分配表(主要)
7	0x00E00 ~ 0x00FFF	文件分配表(备份)
8 ~ 39	0x01000 ~ 0x04FFF	目录表
40 ~ 128	0x05000 ~ 0x0FFFF	数据扇区

表 1-2 扇区配置 (表 1-1)

1.1.2 自定义命令

本范例提供简单的命令格式 (表 1-3) 与自定义命令集 (表 1-4)。

字节	描述	批注
0	命令	
1	长度	0xE (不包含校验和与保留字段)
2 ~ 5	参数1	
6 ~ 9	参数2	
10 ~ 13	签名	0x4343534D - "MSCC"
14 ~ 17	校验和	
18 ~ 63	保留	

表 1-3 命令格式

描述	命令	参数1	参数2	资料	批注
GET_VERSION	0xD3	无	无	无	取得版本号码
GET_STATUS	0xD4	无	无	无	取得状态
GET_PARAM	0xD6	参数1	参数2	无	取得参数
READ_DATA	0xD7	数据长度	无	无	读取数据
SET_PARAM	0xC5	参数1	参数2	有	设定参数
WRITE_DATA	0xC4	数据长度	无	有	写入资料

表 1-4 自定义命令集

1.1.3 自定义命令传输

M032 使用 SRAM 仿真启动扇区(扇区 0)、文件分配表(扇区 6~8)与目录表(扇区 8~39) 与档案内容。SRAM 与扇区配置如图 1-3 (此虚拟磁盘会在程序重启后恢复为预设的状态)。档案内容用于自定义命令控制，所以并不会储存于 Flash 或是 SRAM。在 Host 读取档案时，M032 会依照收到的自定义读取命令回复对应的数据；如果没有收到自定义读取命令，M032 回复全”0”的数据。

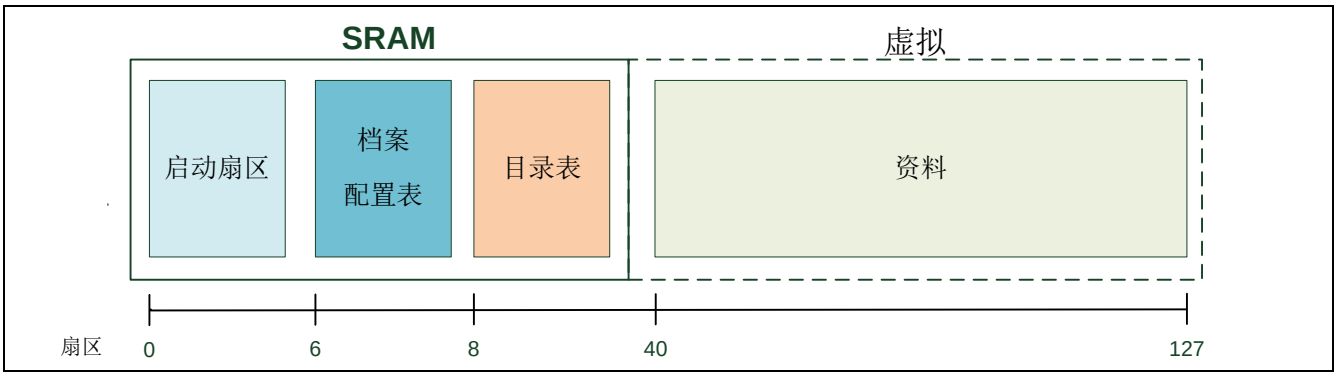

图 1-3 内存与扇区配置范例

M032 连接上 Host 后显示为容量为 44KB 的磁盘，像一般磁盘一样可以执行写入、读取与格式化。

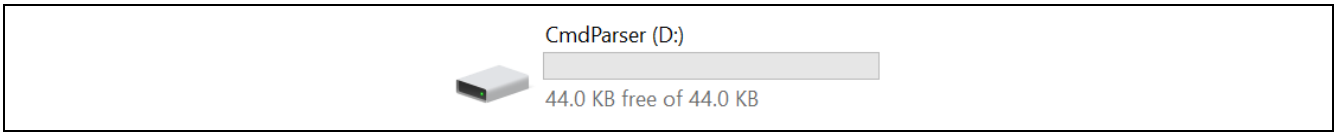

图 1-4 M032 虚拟磁盘

1.1.3.1 写入命令控制流程

以 WRITE_DATA 为例，档案内容如图 1-5 所示。字节 0~17 是命令区域，字节 18~4113 是写入的数据，档案的大小为 18+4K 字节。

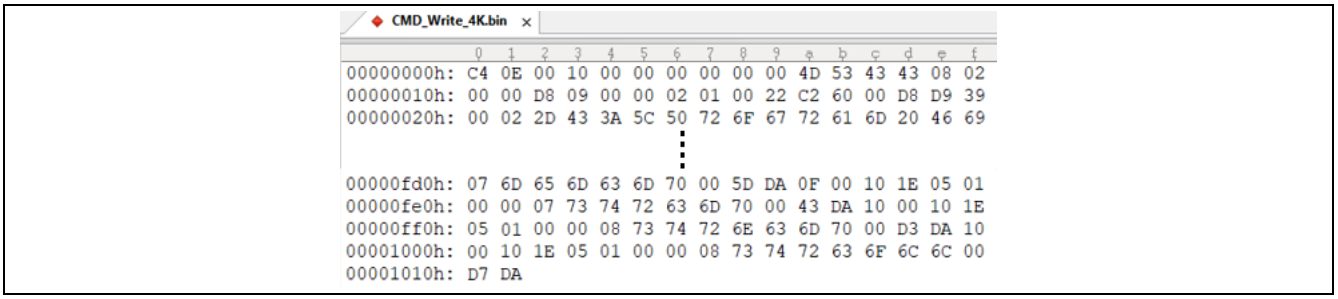

图 1-5 命令档案范例 – WRITE_DATA

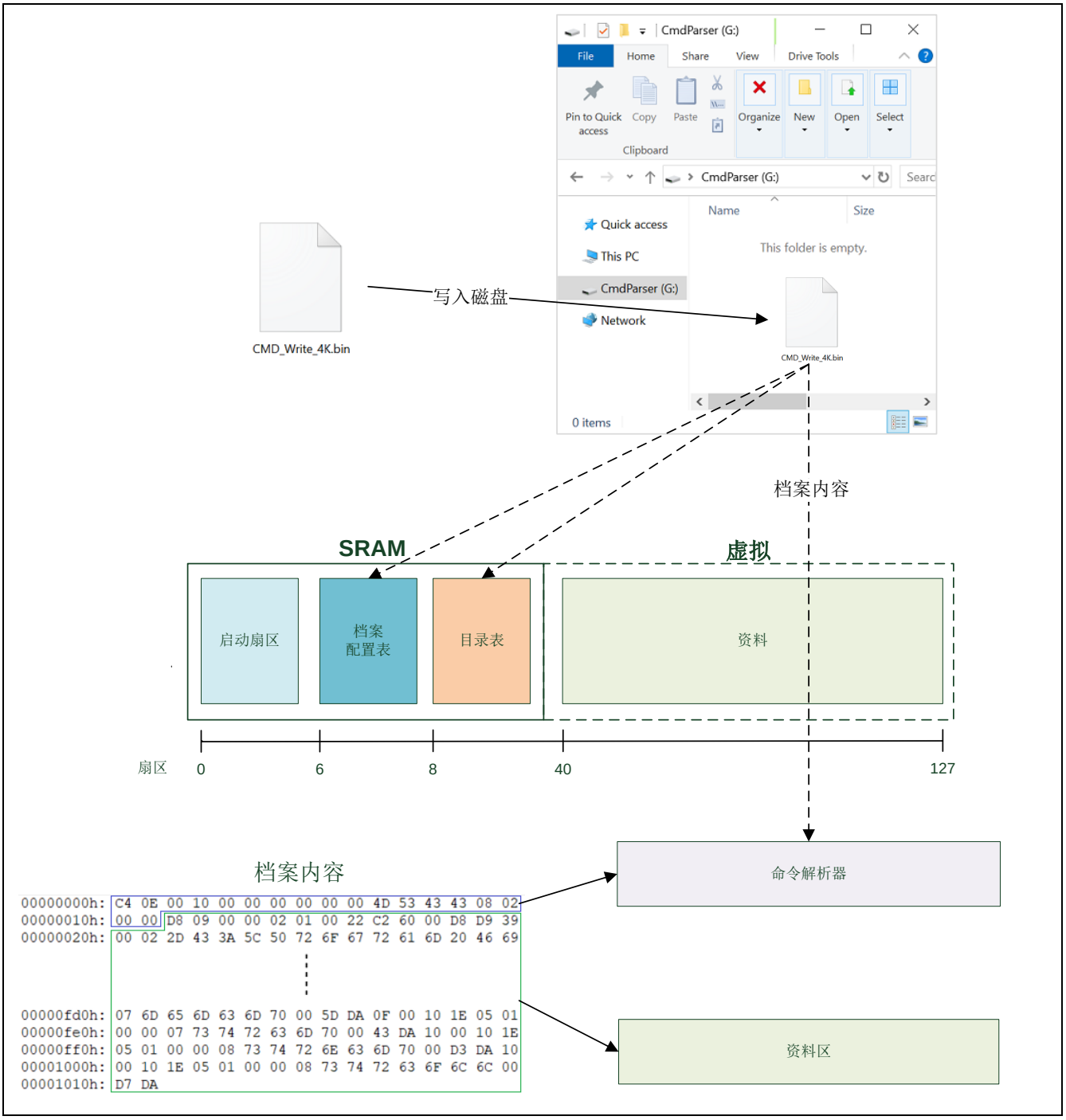

图 1-6 命令档案范例 – WRITE_DATA

当 WRITE_DATA 命令文件写入虚拟磁盘，Host 送出包含文件分配表数据(扇区 6 与 7)、目录表(扇区 8)与档案内容(扇区地址大于 40)的 MSC 写入命令，如图 1-7 所示。M032 会将扇区 6、扇区 7、扇区 8 写入对应的 SRAM 位置，并将档案内容(字节 0~17) 作为自定义命令解析。

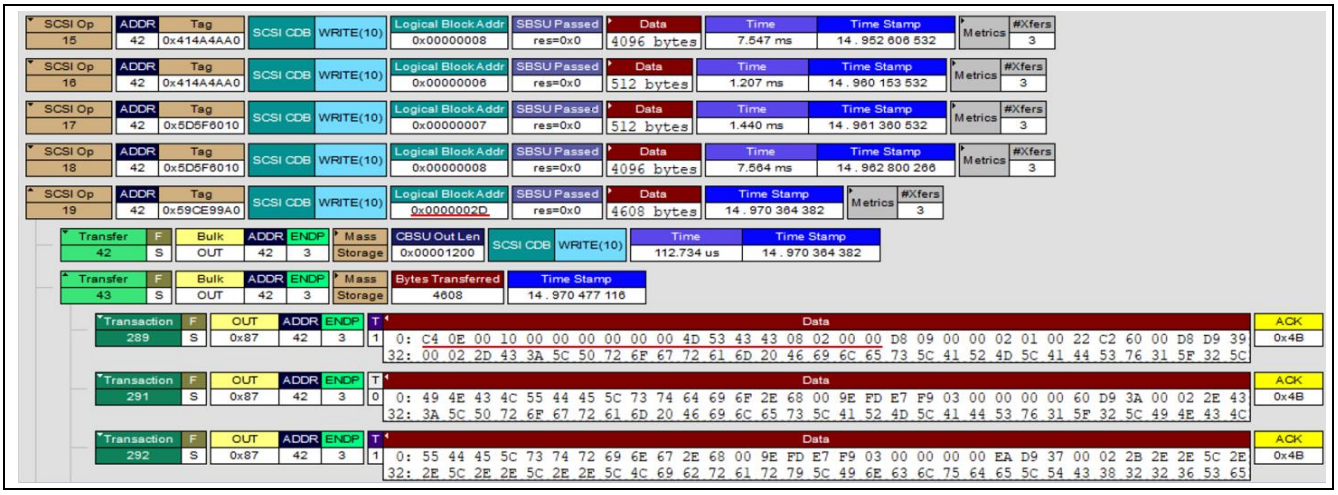

图 1-7 USB 封包 – WRITE_DATA

1.1.3.2 读取命令控制流程

读取数据命令流程包含两个步骤：写入读取命令档 (图 1-9)与从虚拟磁盘读取档案 (图 1-10 图 4-1)。M032 会记录读取命令文件所指的数据类型。以 READ_DATA 命令为例，档案内容如图 1-8 所示。

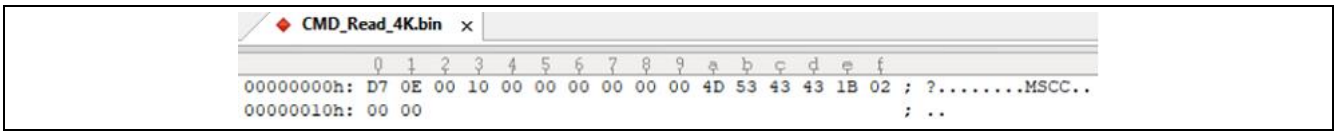

图 1-8 命令档案范例 – READ_DATA

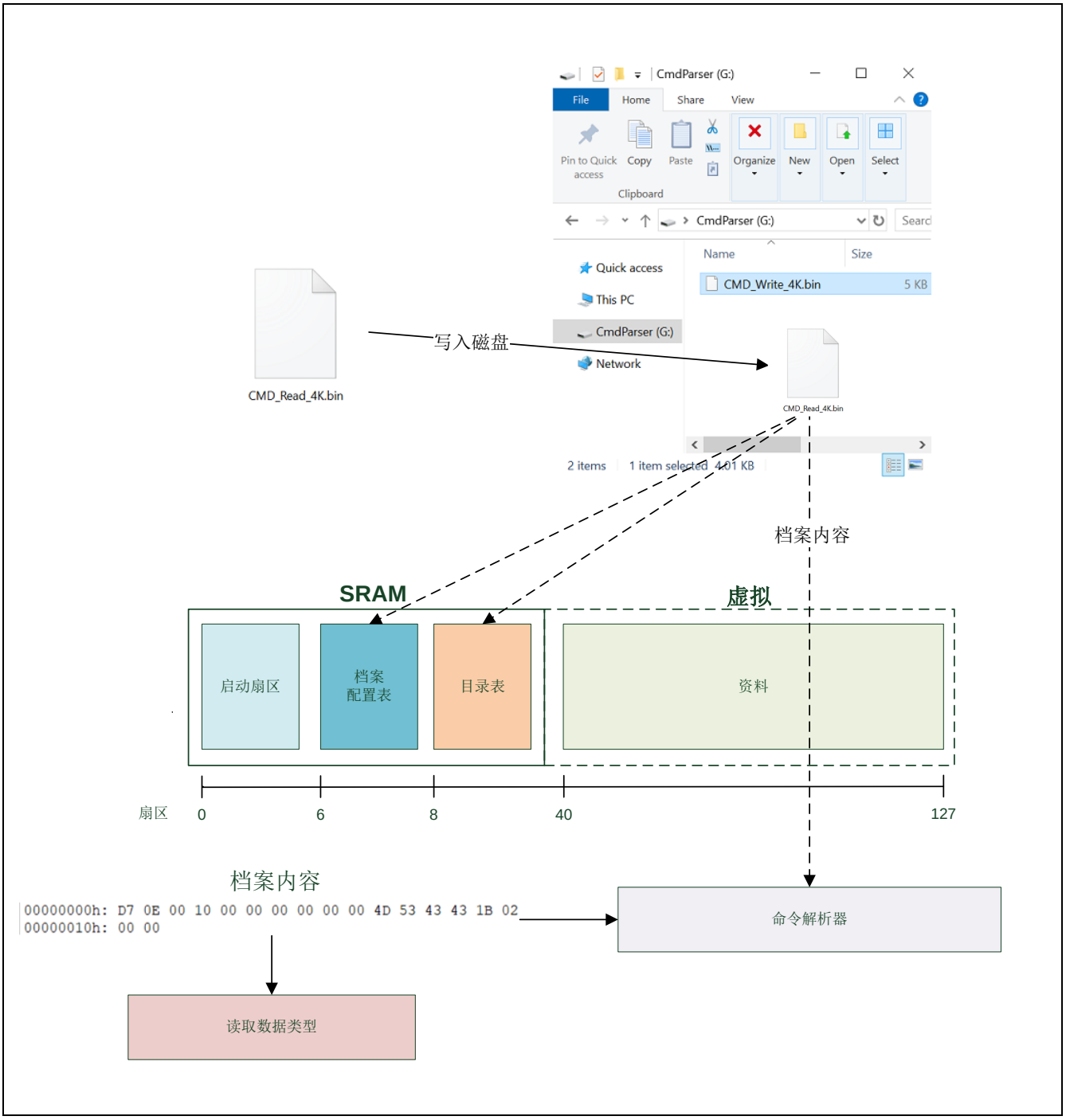

图 1-9 写入读取命令档

当 M032 收到 MSC 读取命令时，会依照收到的自定义的读取命令回复对应的数据。

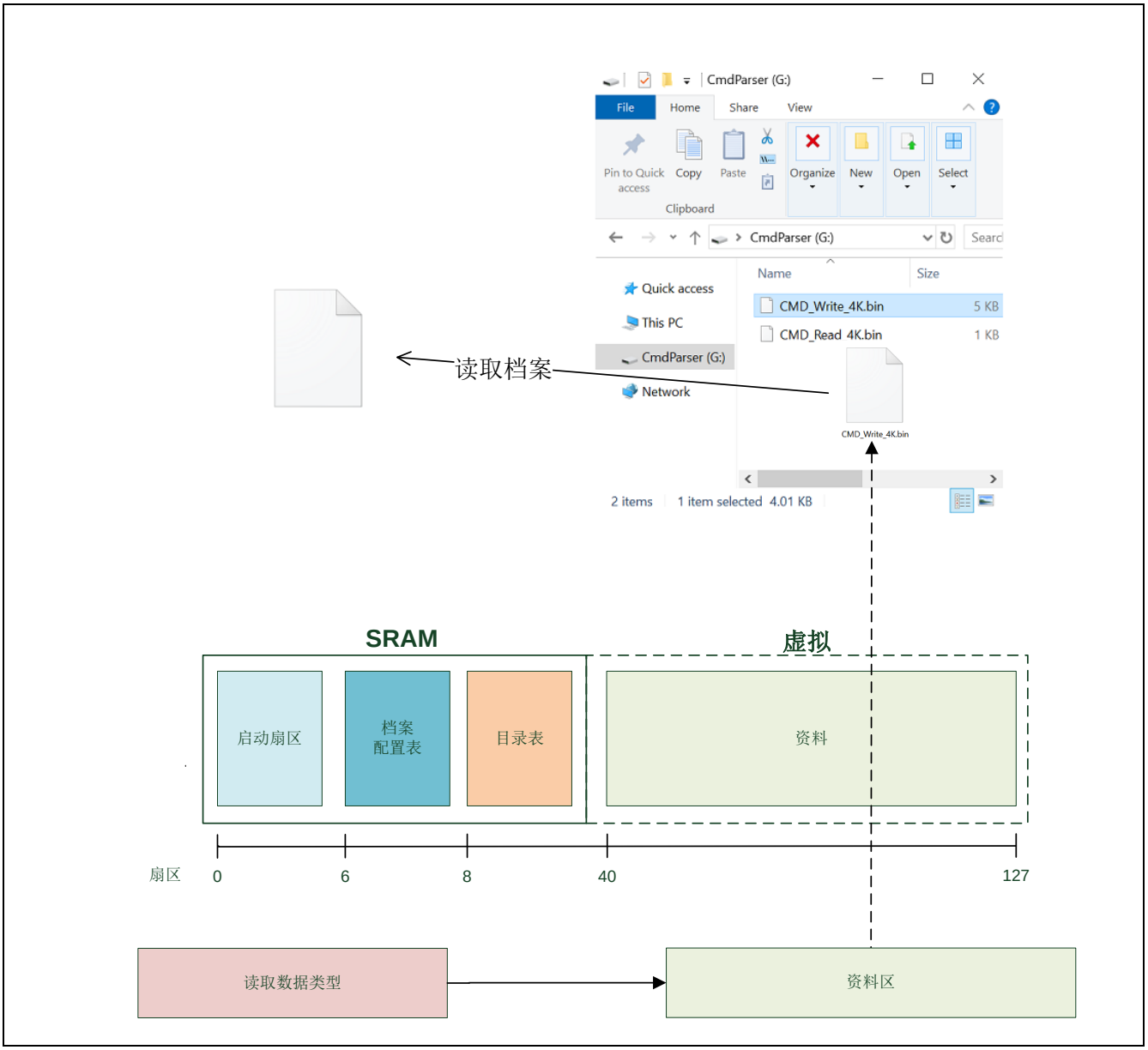

图 1-10 读取数据

将命令档案写入虚拟磁盘后，必须在虚拟磁盘中读取任一个档案大小大于欲读取数据量的档案。Host 会依照档案的大小读取数据(图 1-11 与图 1-12)。操作系统会使用缓冲区的数据不实际从虚拟磁盘读取数据。所以必须使用无缓冲的 I/O 复制方式复制档案。

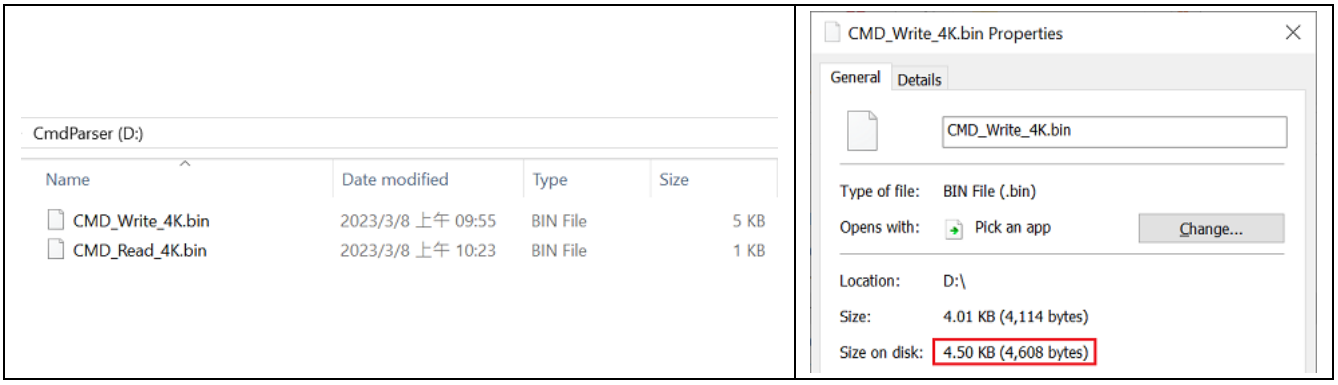

图 1-11 读取档案大小

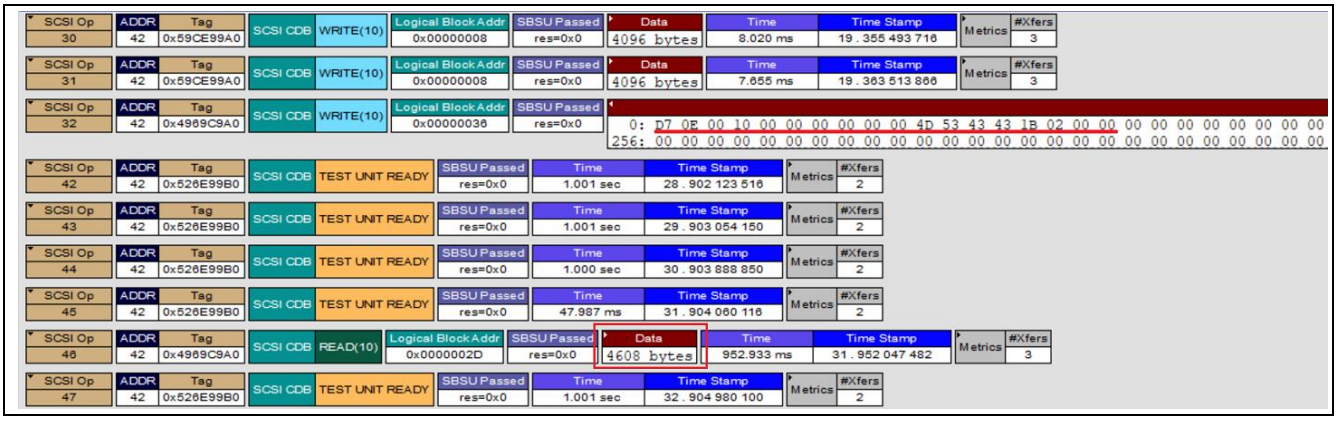

图 1-12 USB 封包 - READ_DATA

1.2 执行结果

1.2.1 虚拟磁盘

M032连接上Host后显示为容量为44KB无内容的磁盘。

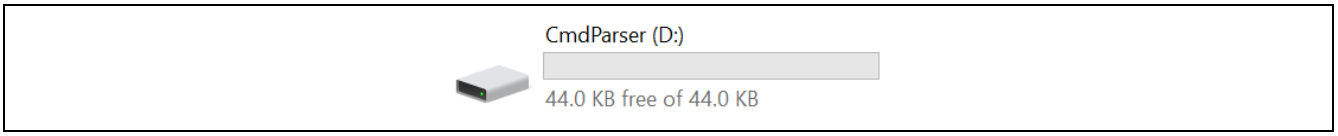

图 1-13 M032 虚拟磁盘

1.2.2 SET_PARAM

将 SET_PARAM 命令文件写入虚拟磁盘。

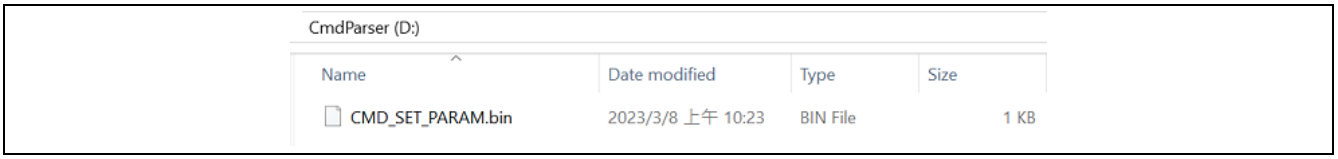

图 1-14 写入 SET_PARM 命令文件至虚拟磁盘

M032 收到 SET_PARM 命令后，显示收到 SET_PARM 命令。

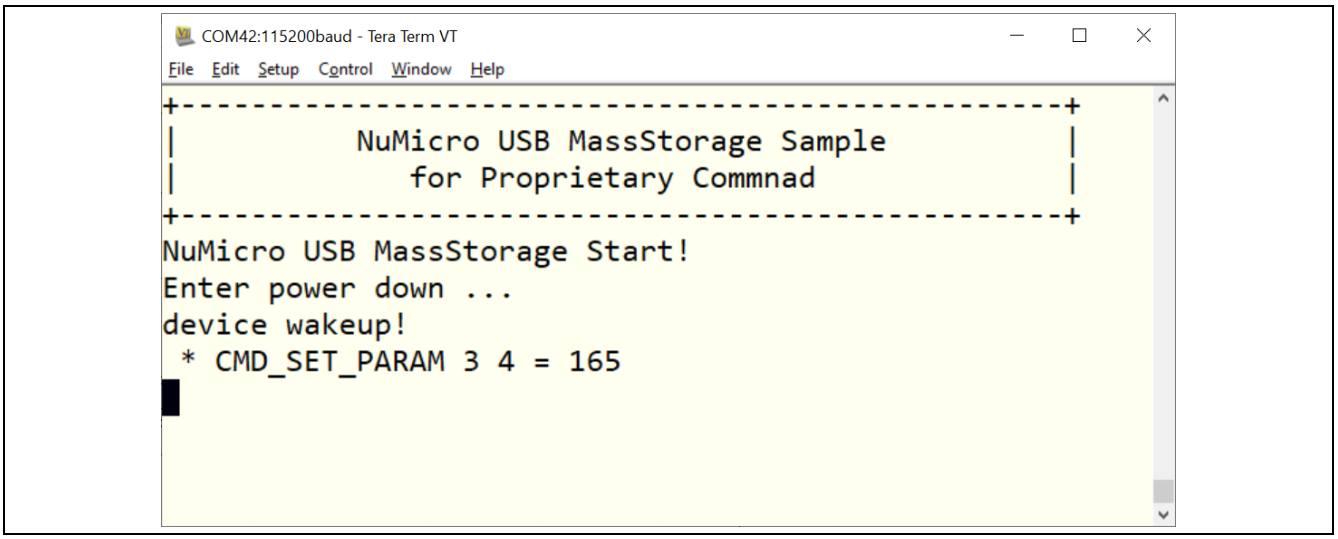

图 1-15 M032 收到 SET_PARM 命令

1.2.3 WRITE_DATA

将 WRITE_DATA 命令文件写入虚拟磁盘。

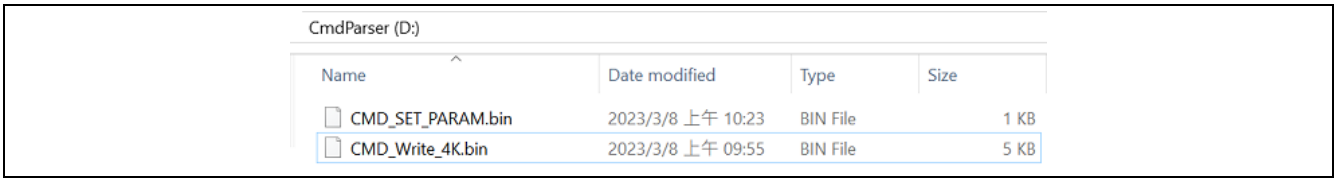

图 1-16 写入 WRITE_DATA 命令文件至虚拟磁盘

M032 收到 WRITE_DATA 命令后，显示收到 WRITE_DATA 命令，显示写入进度直到 WRITE_DATA 命令完成。

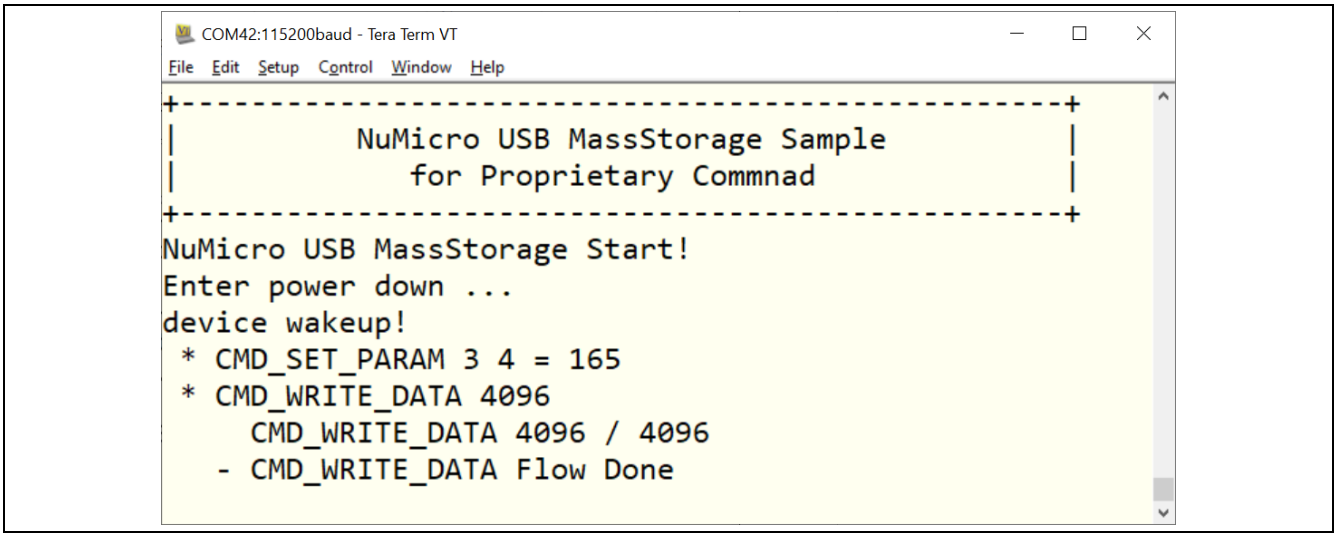

图 1-17 M032 收到 WRITE_DATA 命令

1.2.4 READ_DATA

将 READ_DATA 命令文件写入虚拟磁盘。

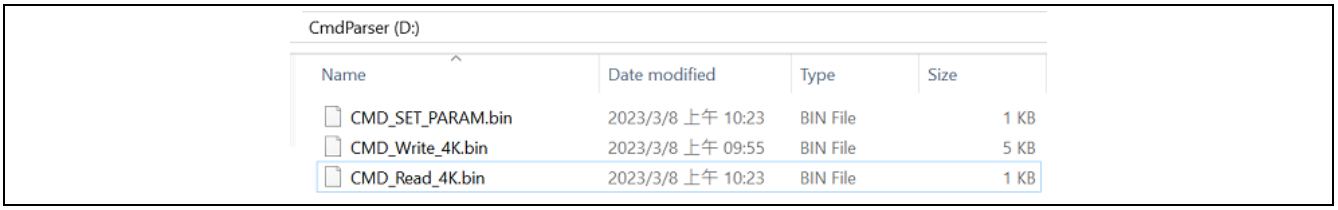

图 1-18 写入 READ_DATA 命令文件至虚拟磁盘

M032 收到 READ_DATA 命令后，显示收到 READ_DATA 命令 (读取 4KB)，此时还没开始回传数据。

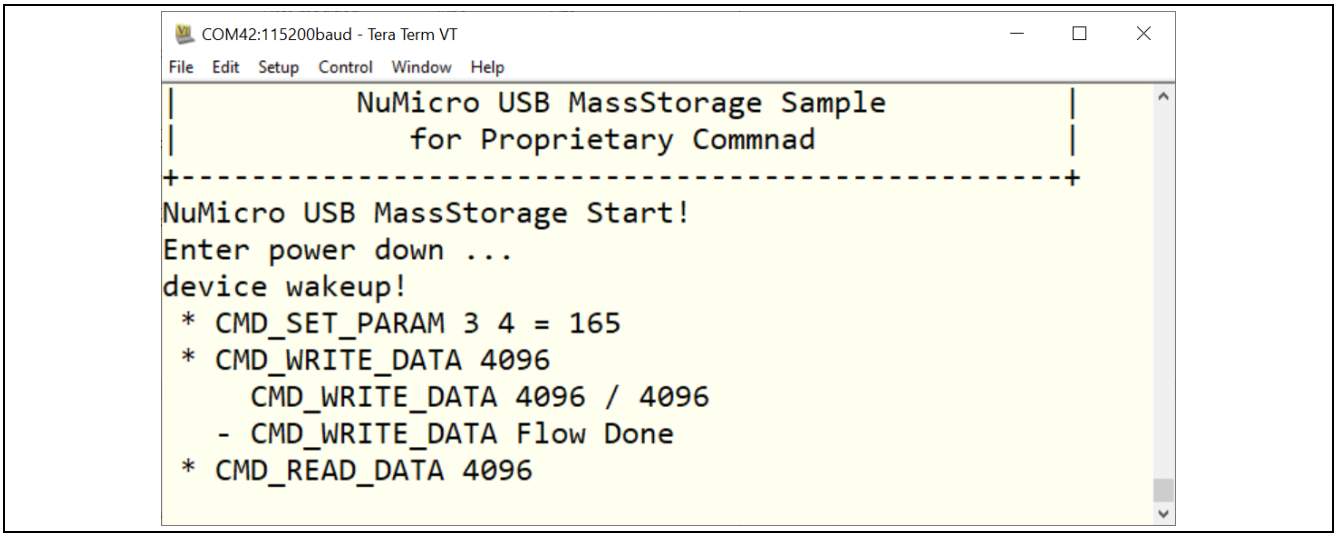

图 1-19 M032 收到 READ_DATA 命令

从 M032 虚拟磁盘已无缓冲的方式读取 CMD_Write_4K.bin (选取读取档案的条件是必须大于要读取的数据量 - 4KB)，M032 显示读取进度直到 READ_DATA 命令完成。

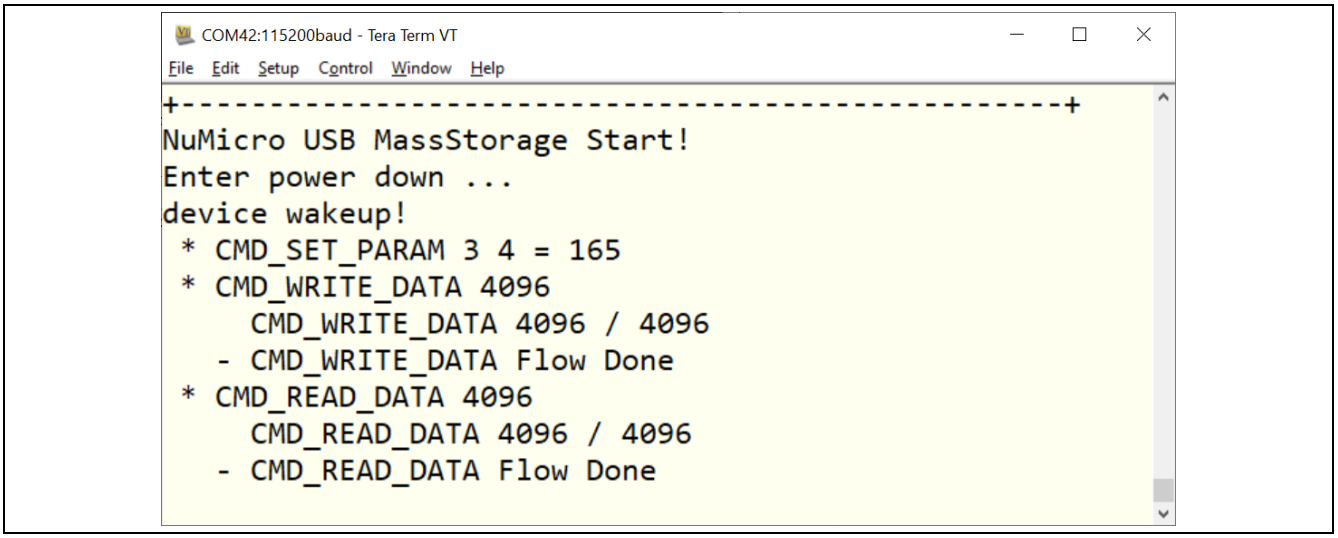

图 1-20 READ_DATA 流程结束

与写入数据比对与先前写入的数据比对结果一致。

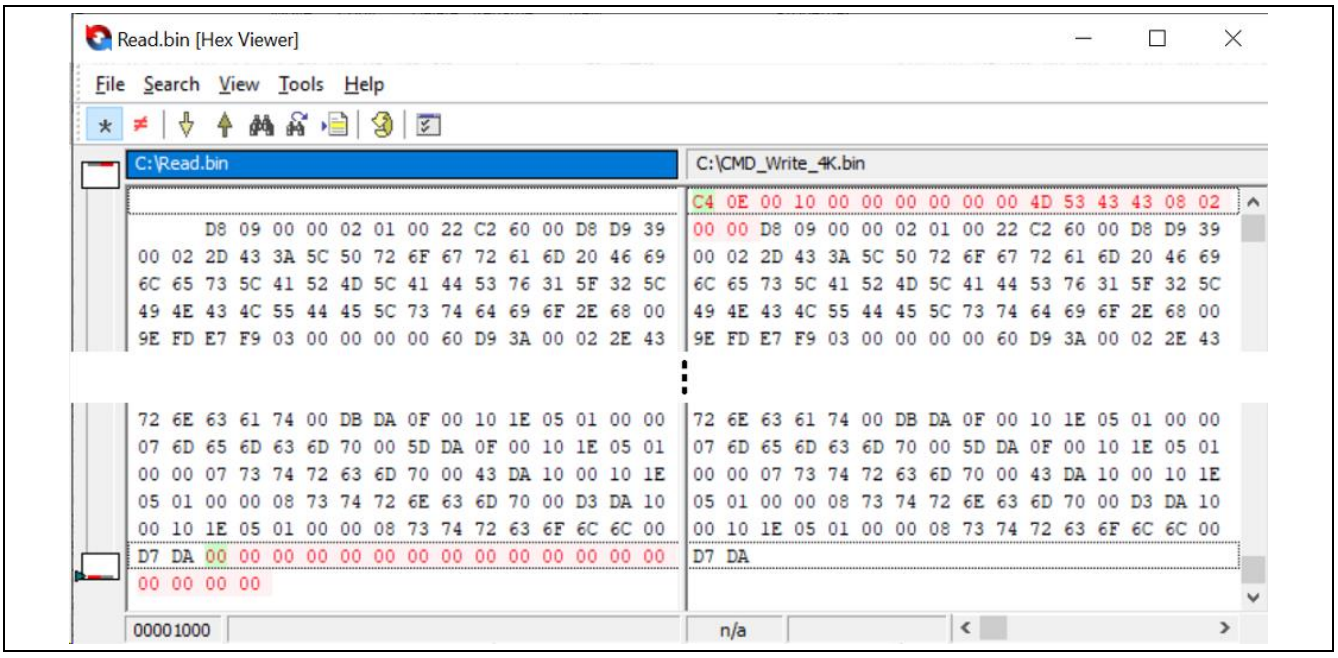

图 1-21 比较结果

1.2.5 无缓冲的 I/O 复制档案

1.2.5.1 Windows

Windows command - Xcopy

Xcopy	
参数	**叙述**
/j	使用无缓冲的 I/O 复制，建议使用于非常大的档案。此功能首次支持于Windows 7。

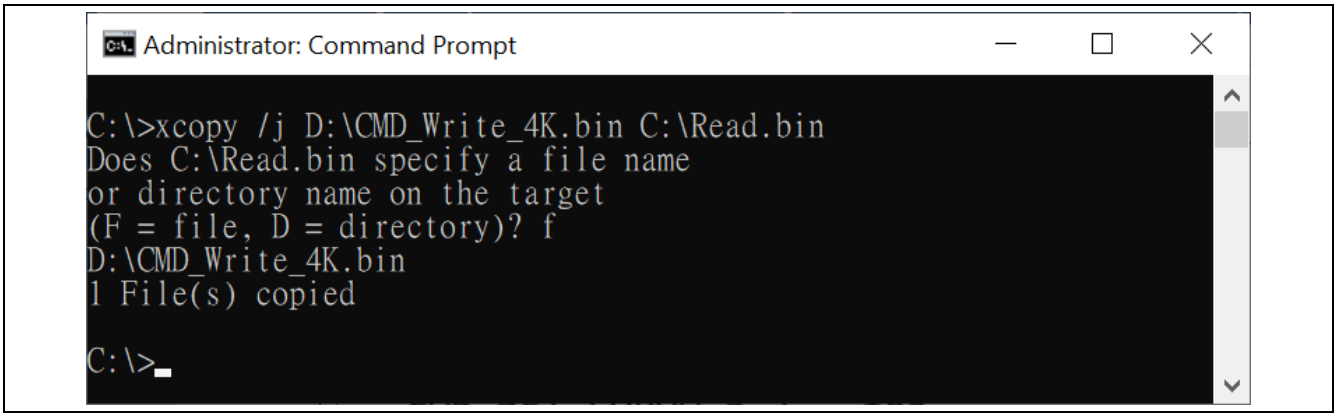

图 1-22 使用 xcopy 以使用无缓冲的 I/O 复制档案

执行档 CopyFile 使用 CreateFileA API 搭配参数 FILE_FLAG_NO_BUFFERING 读取档案。此范例提供简单的程序使用无缓冲的 I/O 复制档案。

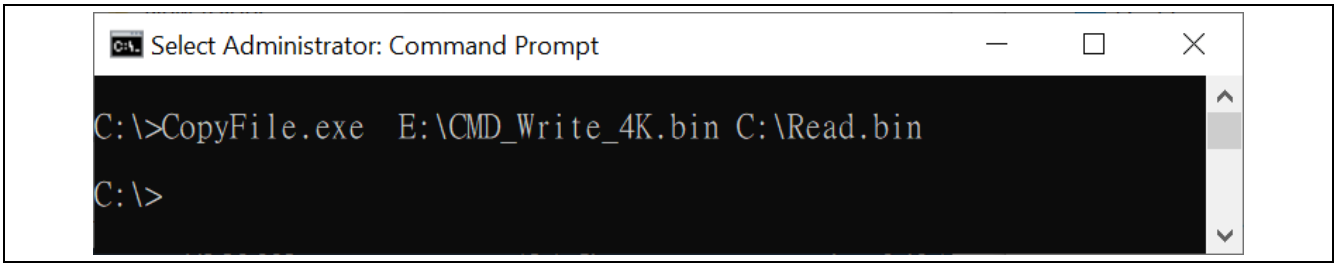

图 1-23 使用 CopyFile 以使用无缓冲的 I/O 复制档案

1.2.5.2 Linux

Linux command - dd

dd Command	
参数	**叙述**
iflg=direct	使用direct I/O的方式

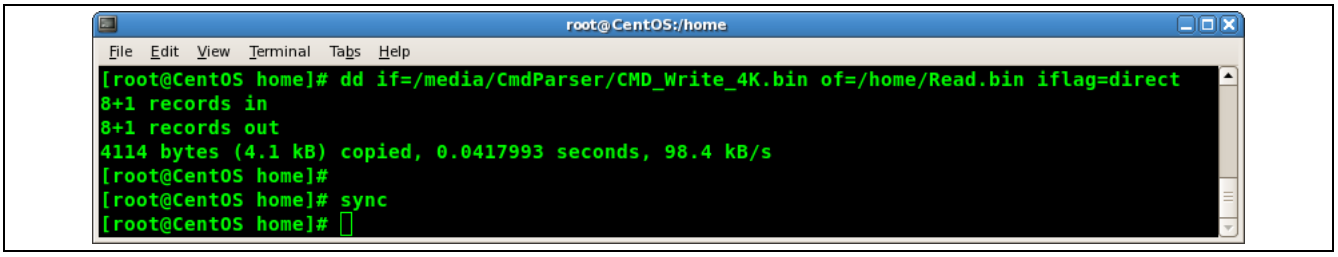

图 1-24 使用 dd 以使用无缓冲的 I/O 复制档案

2. 代码介绍

2.1 虚拟磁盘配置

配置虚拟磁盘的内容，代码位于 CmdParser.c，以下是虚拟磁盘的启动扇区内容。

```
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
};
```

配置虚拟磁盘内容的定义，代码位于 CmdParser.h，修改”DISK_SIZE”就可以修改虚拟磁盘的大小。默认的扇区配置可能会与操作系统格式化的配置不同导致执行程序不正常运作，可以在系统格式化后，使用定义“__FAT_INFO__”来确认扇区配置是否需要修正，可能需要修正的是 RSVD_SEC_CNT 与 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)
```

显示默认扇区配置与目前扇区配置的内容，代码位于 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 仿真 FAT 信息

仿真虚拟磁盘读写启动扇区、文件分配表与目录表，代码位于 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);
}
```

档案位置的信息储存于 SRAM，所以此虚拟磁盘会在程序重启后恢复为预设的状态。

2.3 自定义命令定义

自定义的命令的数据结构与指令集，代码位于 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 自定义命令流程控制

MSC SCSI 写入命令流程控制代码位于 CmdParser.c，需要修改命令集只需要修改 ProcessCommand，另外在 WRITE_DATA 结束后加上需要的程序。

```
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 读取命令流程控制代码代码位于 CmdParser.c，M032 会依照收到的自定义读取命令会设定变量”g_u8ReadType”，并在后续收到 MSC 读取命令时回复对应的数据。

```
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;
                }
            }
        }
    }
}
```

自定义的命令的主要处理程序，代码位于 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. 软件与硬件需求

3.1 软件需求

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

3.2 硬件需求

- 电路组件
  - NuMaker-M032KG V1.0

4. 目录信息

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

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *EC_M032_MSC_Vendor_Cmd.uvproj*。
2. 进入编译模式接口
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式接口
   - 执行代码

6. 修订纪录

Date	Revision	Description
2023.03.10	1.00	初始发布。

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.