Vendor Command Transmitting through Mass Storage Device

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the M032 series microcontroller (MCU) for vendor command transmitting through USB Mass Storage Device.
BSP Version	M032_Series_BSP_CMSIS_V3.05.000
Hardware	NuMaker-M032KG 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 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

In general, a host side tool is needed to deal with the vendor command transmitting. However, it is not easy to maintain such tool for everyone. This example code provides a solution using the M032 series microcontroller (MCU) for vendor command transmitting through USB Mass Storage without any host side tool. The M032 uses Mass Storage Device Class protocol to deal with the vendor command control flow by writing or reading a file to / from the disk.

1.1 Principle

The M032 acts as a virtual disk formatted with FAT12 for user to send vendor command by writing a file with specified data. The following sections will describe the FAT file system and vendor command control flow.

1.1.1 FAT File System

A FAT file system is composed of “file location data” (Boot Sector, File Allocation Table, and Root Directory) + “file data”. The File in FAT architecture is shown in Figure 1-1.

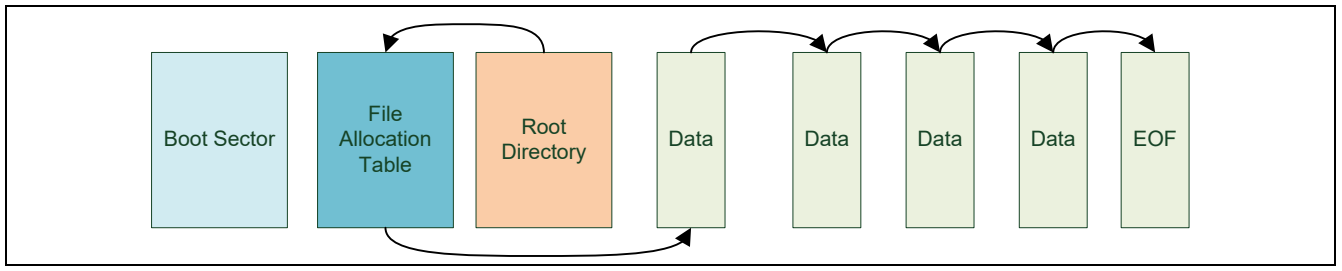

Figure 1-1 A File in FAT File System

When a file is copied to a FAT disk, the transfer data on USB bus includes file location data and file content. User can distinguish between file location data and file content by the sector address.

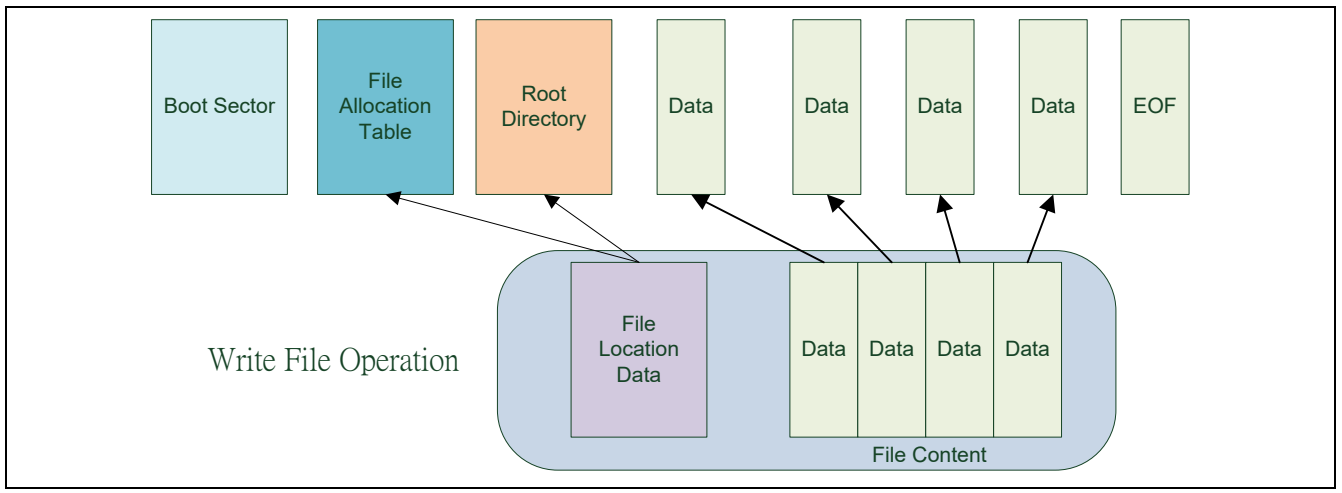

Figure 1-2 Write File to FAT File System

1.1.1.1 Boot Sector Interpretation

Boot Sector is the first sector of a FAT system and it contains most of the information with which to determine the file system type, size and location of data structures. A Boot Sector example is shown in Table 1-1.

Byte(s)	Value (hex)	Description
00 ~ 02	EB 3C 90	Instructions to jump to boot code
03 ~ 10	4D 53 44 4F 53 35 2E 30	Name string (MSDOS5.0)
11 ~ 12	00 02	Bytes/sector
13	01	Sectors/cluster
14 ~ 15	06 00	Size of reserved area
16	02	Number of FATs
17 ~ 18	00 02	Max. number of root directory entries
19 ~ 20	80 00	Total number of sectors
21	F8	Media type (removable)
22 ~ 23	01 00	FAT size
24 ~ 25	01 00	Sectors/track
26 ~ 27	01 00	Number of heads
28 ~ 31	00 00 00 00	Number of sector before partition
32 ~ 35	00 00 00 00	Total number of sectors
36	80	Drive number
37	00	Unused
38	29	Extended boot signature
39 ~ 42	2F 44 83 7A	Volume serial number (7A83-442F)
43 ~ 53	4E 4F 20 4E 41 4D 45 20 20 20 20	Volume label ("NO NAME ")
54 ~ 61	46 41 54 31 32 20 20 20	File system type label ("FAT12 ")
62 ~ 509	[snip]	Not used
510 ~ 511	55 AA	Signature value (0xAA55)

Table 1-1 Boot Sector Example

1.1.1.2 Sector Assignments

The sector assignments of the Boot Sector is shown in Table 1-2.

- Boot Sector is 1 sector (sector #0, 0x200 bytes)
- FAT starts at sector #6 (Byte 14 & 15)
- There are two FATs (Byte 16), 1 sectors for each FAT (Byte 22 & 23)

- The Root Directory can contain 512 entries (Byte 17 & 18), each 32 bytes (0x4000 bytes = 32 sectors)
- File storage starts at sector #47 (6+1+1+32), Address is 0x5000 (0xC00 + 0x200 + 0x200 + 0x4000)

Sector(s)	Address	Function
0	0x00000 ~ 0x001FF	Boot Sector
6	0x00C00 ~ 0x00DFF	File Allocation Table (primary)
7	0x00E00 ~ 0x00FFF	File Allocation Table (secondary)
8 ~ 39	0x01000 ~ 0x04FFF	Root Directory
40 ~ 128	0x05000 ~ 0x0FFFF	File storage space

Table 1-2 Sector Assignments of Table 1-1

1.1.2 Vendor Command

The example code defines a set of vendor command for the data transfer flow. Table 1-3 shows the command structure and Table 1-4 shows the command set.

Byte	Description	Comment
0	Command	
1	Length	0xE (Not include Checksum& Reserved)
2 ~ 5	Arg 1	
6 ~ 9	Arg 2	
10 ~ 13	Signature	0x4343534D - "MSCC"
14 ~ 17	Checksum	
18 ~ 63	Reserved	

Table 1-3 Command structure

Description	CMD	Arg1	Arg2	Data	Comment
GET_VERSION	0xD3	N/A	N/A	N/A	Get Current Version
GET_STATUS	0xD4	N/A	N/A	N/A	Get status
GET_PARAM	0xD6	Arg1	Arg2	N/A	Get parameter
READ_DATA	0xD7	Data Count	N/A	N/A	Read data from buffer
SET_PARAM	0xC5	Arg1	Arg2	Yes	Set parameter
WRITE_DATA	0xC4	Data Count	N/A	Yes	Write data to buffer

Table 1-4 Vendor Command Set

1.1.3 Vendor Command Transmitting

In the example code, the M032 uses SRAM to simulate the Boot Sector (sector #0), File Allocation Table (sector #6~8), and Root Directory areas (sector #8~39). The memory layout and sector assignments are shown in Figure 1-3 (the disk will be reset to empty after example code restarts). The file content is used to be the vendor command, so it is not saved to Flash or SRAM. The M032 will report all “0” when getting MSC Read command if it did not get vendor read command.

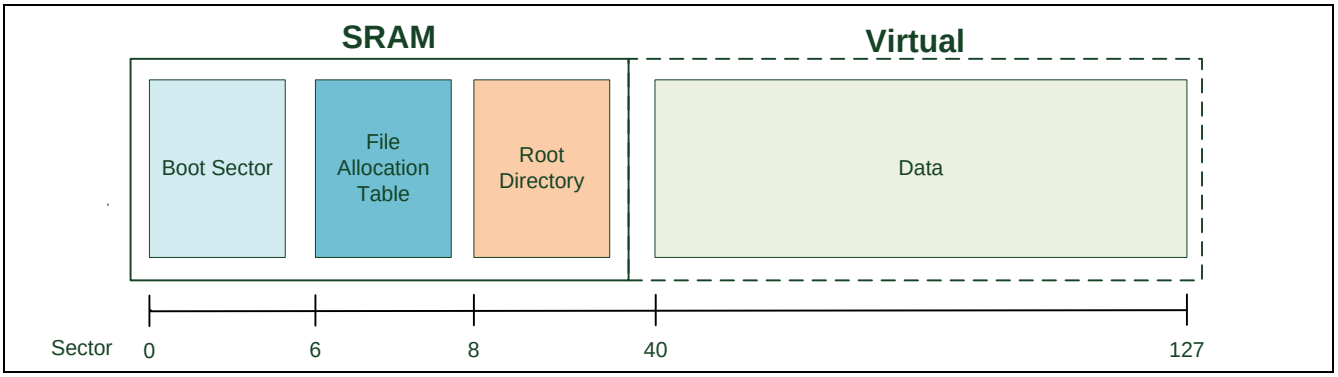

Figure 1-3 Memory Layout and Sector Assignments Example

The M032 exports an empty disk and user can do read / write / format operation to the disk.

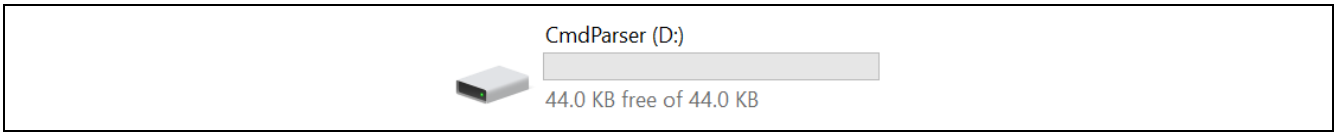

Figure 1-4 M032 Virtual Disk

1.1.3.1 Set Command / Write Data Flow

Taking WRITE_DATA command for example, the file content (18 + 4K bytes) is shown in Figure 1-5. Byte 0 ~ 17 are command, and Byte 18 ~ 4113 are the data for WRITE_DATA command with 4K data.

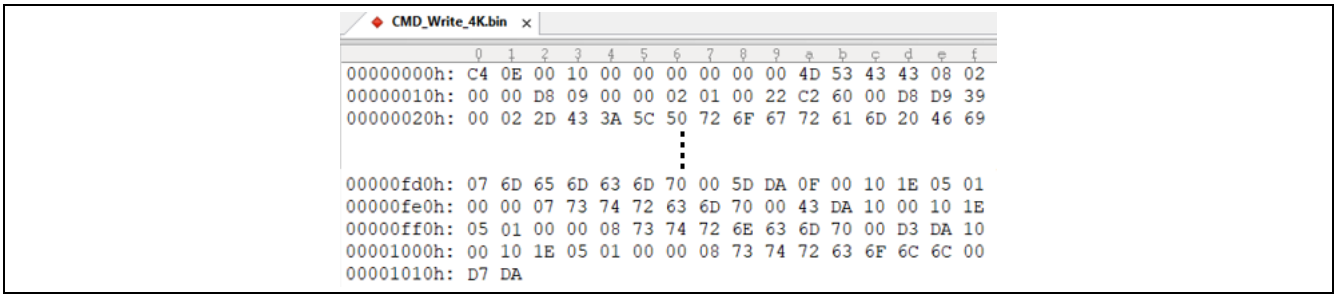

Figure 1-5 Example File – WRITE_DATA Command

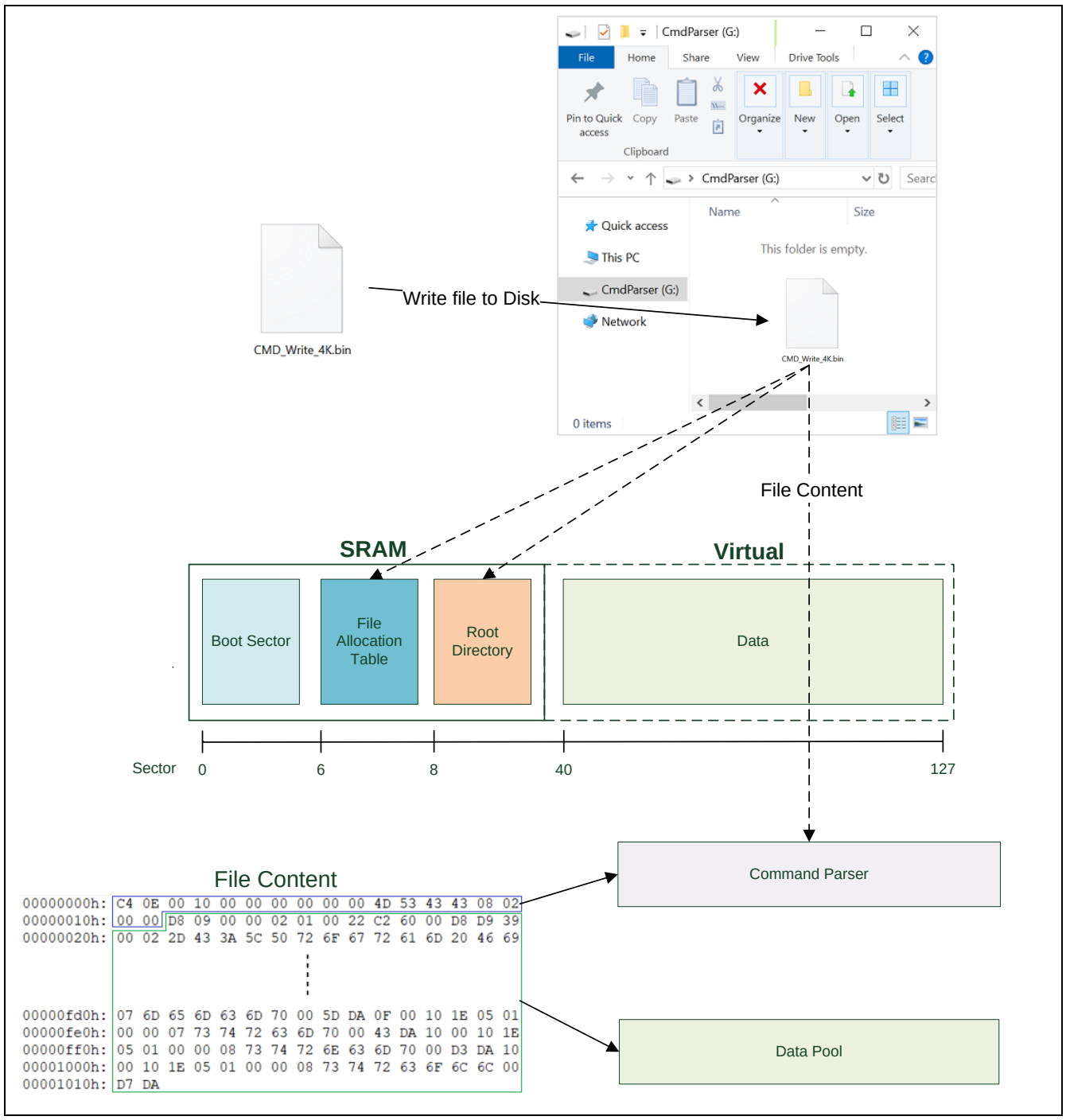

Figure 1-6 Write Command Flow

When the WRITE_DATA command file is copied to the virtual disk, host sends Write command (see Figure 1-7) with data for File Allocation Table (sector #6 & #7), Root Directory (sector #8) and file content (the sector address larger than 40). The M032 writes Sector #6, #7 #8 to the SRAM and parses byte 0 ~ 17 of sector for vendor command control.

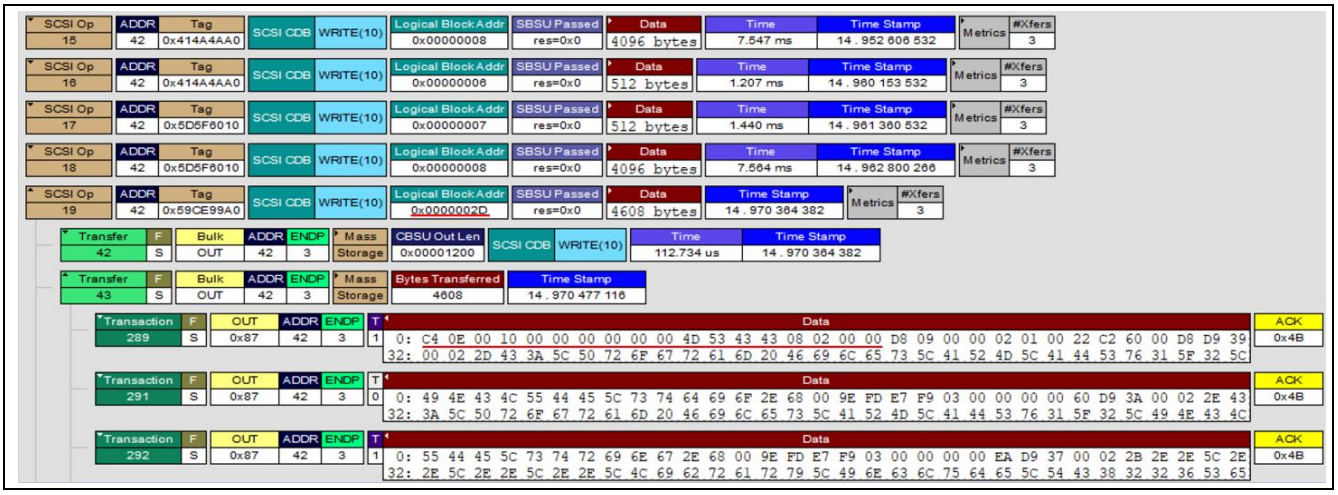

Figure 1-7 USB Packets for WRITE_DATA Command

1.1.3.2 Read Command Flow

The read command flow includes two steps: write read command to disk (see Figure 1-9) and read file from disk (see Figure 1-10). The M032 keeps read command type getting from writing read command file. Taking READ_DATA command for example, the file content is shown in Figure 1-8.

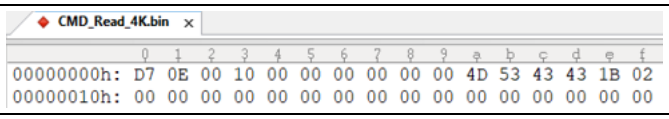

Figure 1-8 Example File – READ_DATA Command

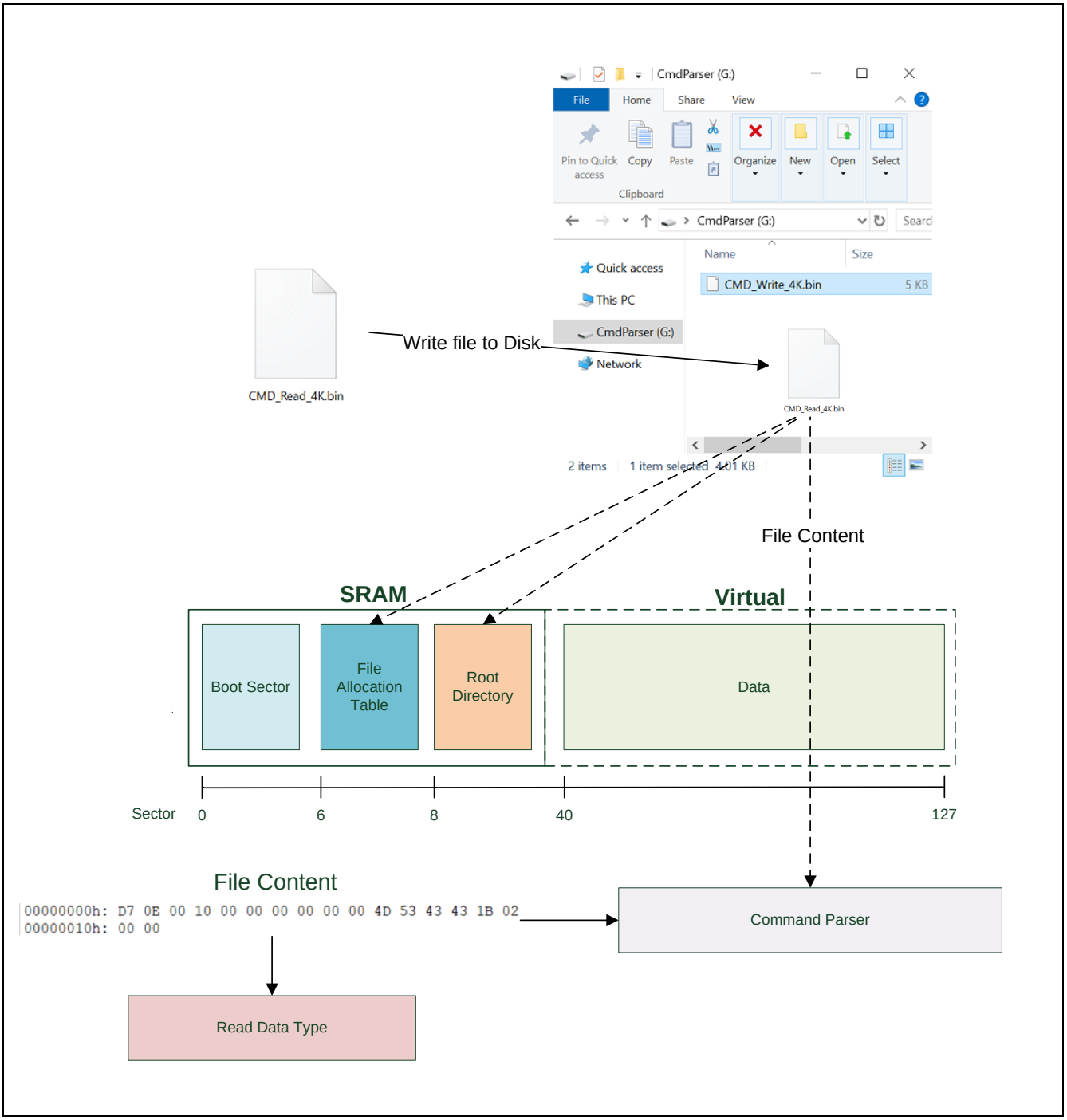

Figure 1-9 Write Read Command

The M032 sends corresponding data to host when getting MSC SCSI Command for read data.

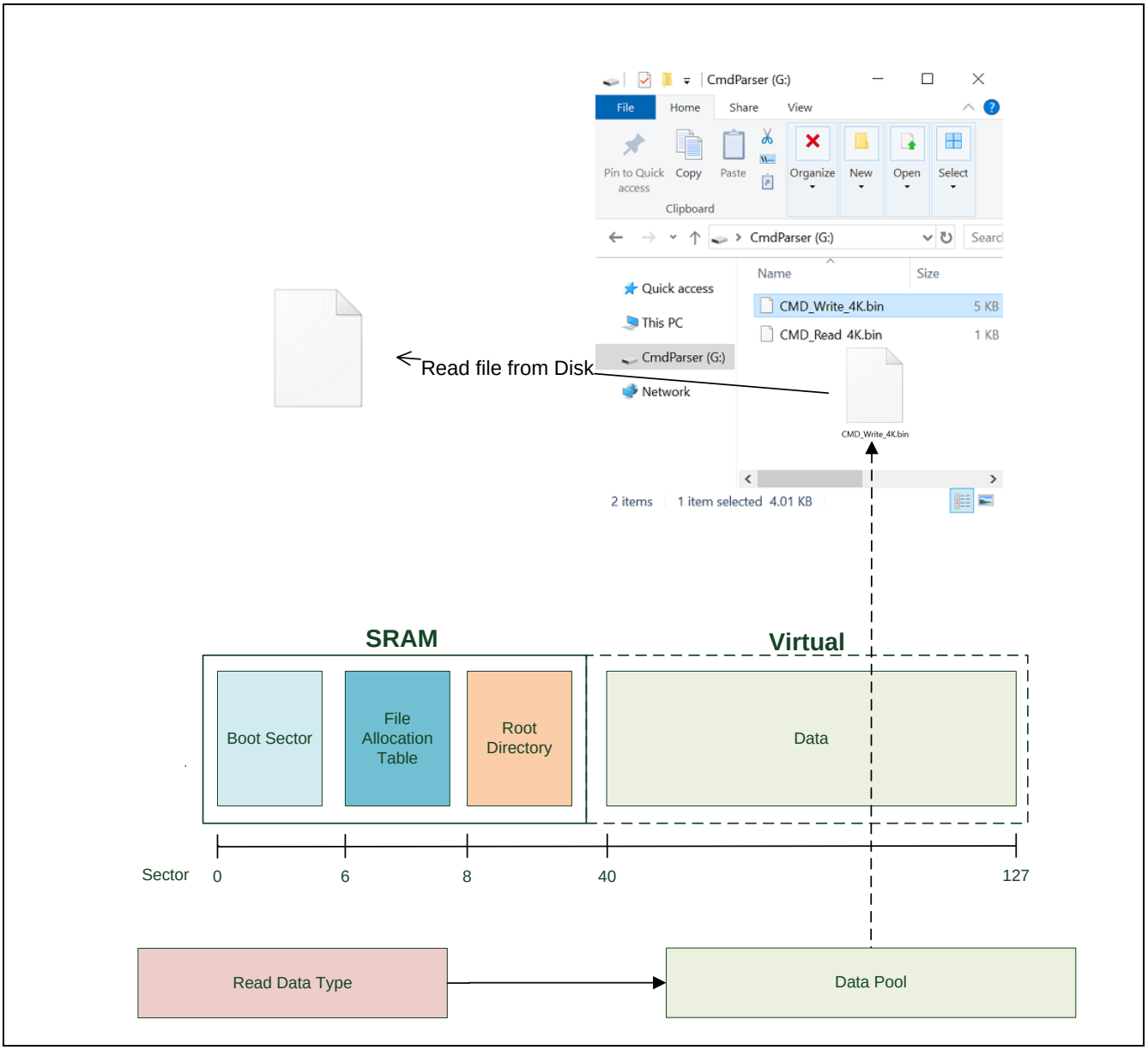

Figure 1-10 Read Data

Please read any file whose size is larger than required size in the virtual disk. The size of read back file is equal to the size of the file (Figure 1-11 & Figure 1-12). Operation system may use buffering data without reading file actually from disk. So please make sure that the file is copied without buffering.

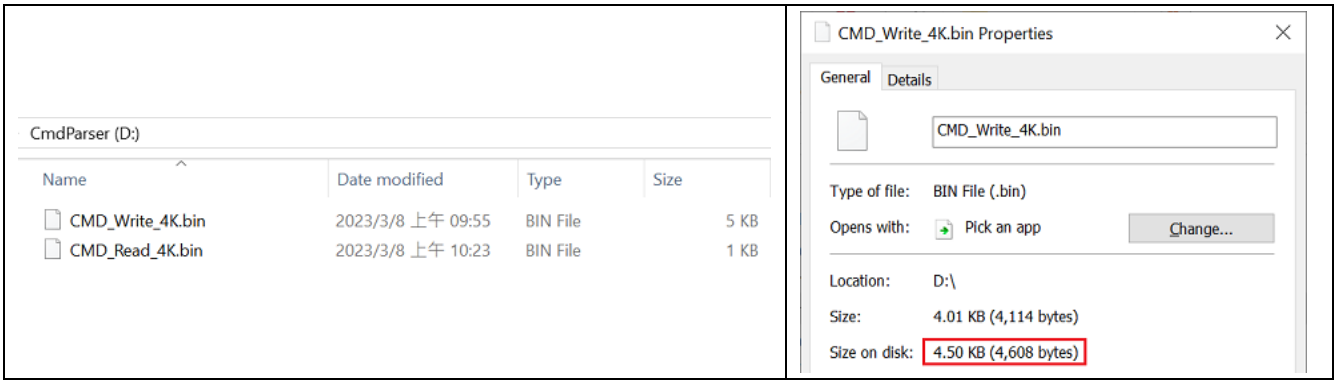

Figure 1-11 File Size

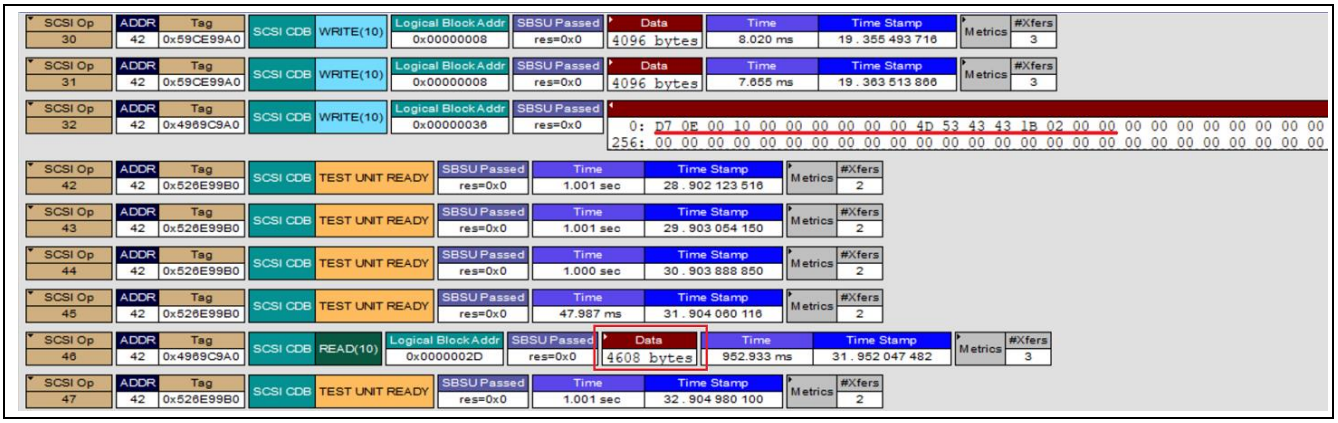

Figure 1-12 USB Packets for READ_DATA Command

1.2 Demo Result

1.2.1 Virtual Disk

The M032 exports an empty disk with a capacity of 44KB.

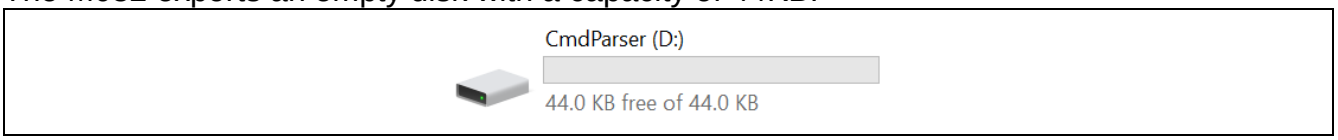

Figure 1-13 M032 Virtual Disk

1.2.2 SET_PARAM

Copy SET_PARAM command file to the disk.

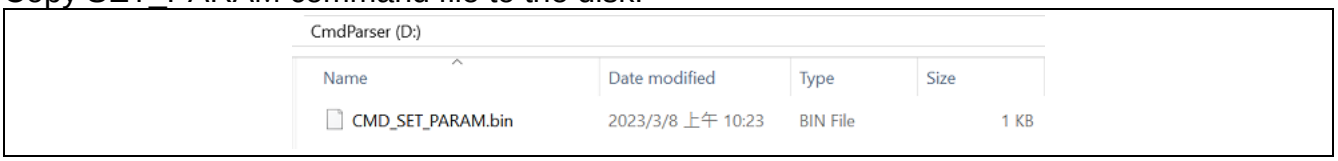

Figure 1-14 Write SET_PARAM File to the Disk

The M032 gets SET_PARAM command and prints messages.

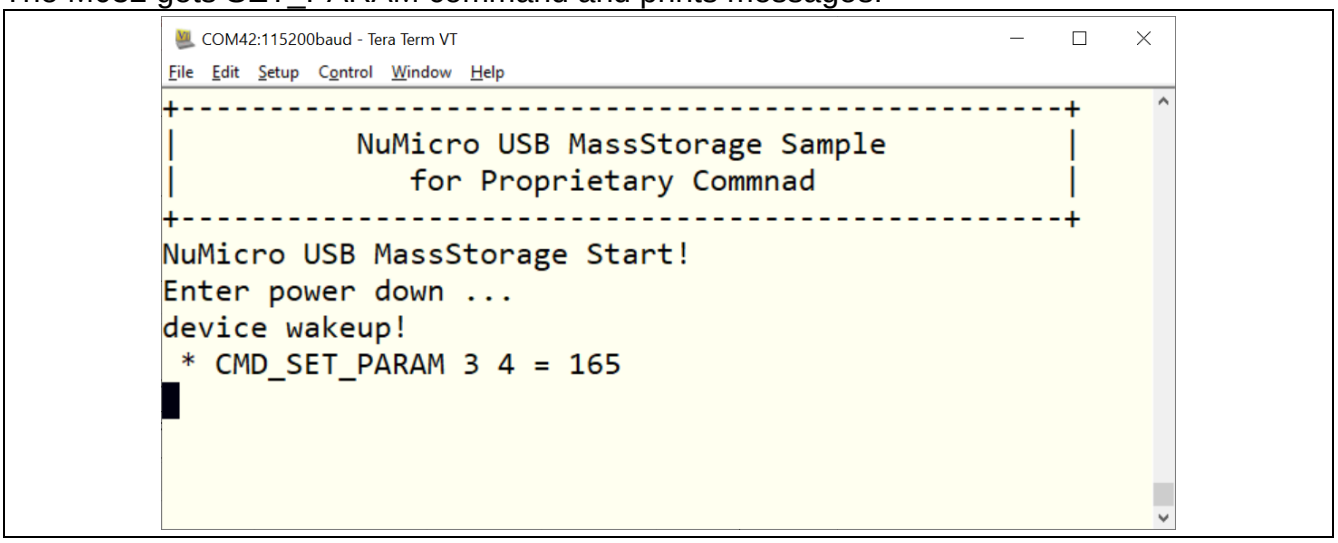

Figure 1-15 Get SET_PARAM Command

1.2.3 WRITE_DATA

Copy WRITE_DATA command file to the disk.

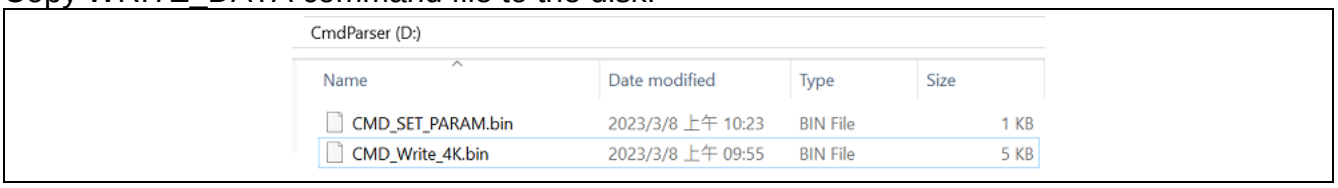

Figure 1-16 Write WRITE_DATA File to the Disk

The M032 gets WRITE_DATA command and prints messages.

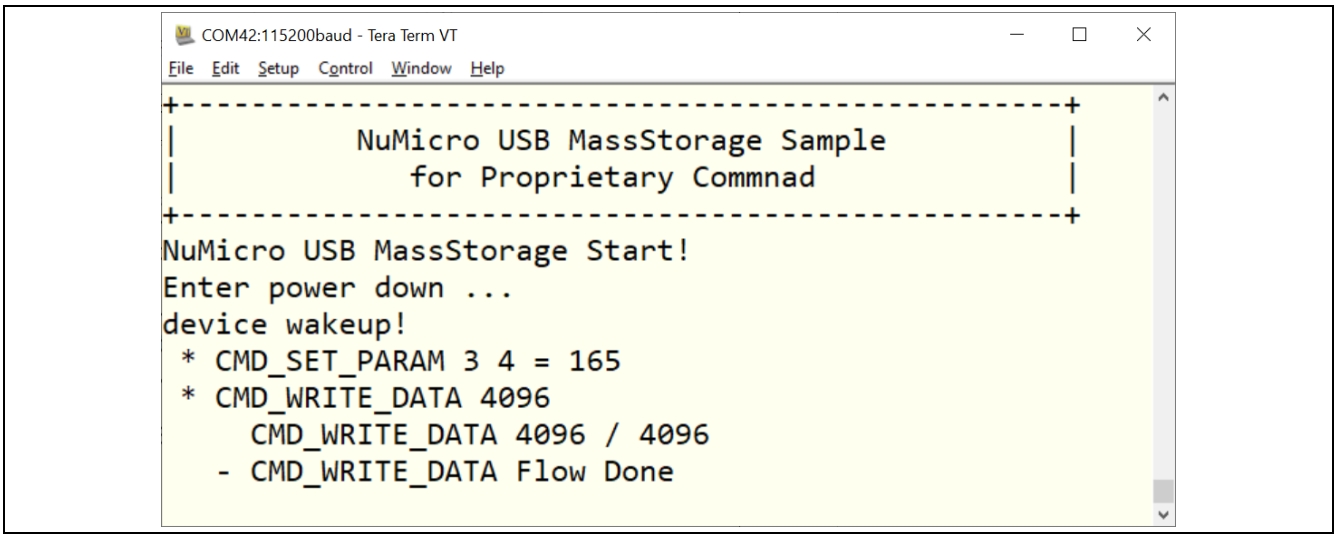

Figure 1-17 Get WRITE_DATA Command

1.2.4 READ_DATA

Copy READ_DATA command file to the disk.

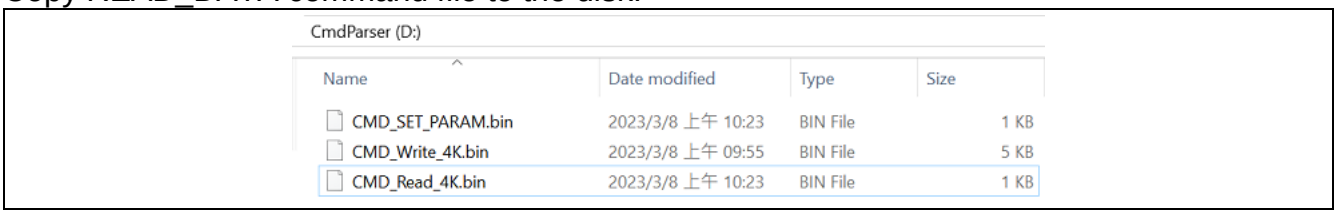

Figure 1-18 Write READ_DATA file to the Disk

The M032 gets READ_DATA command and prints messages (read 4KB).

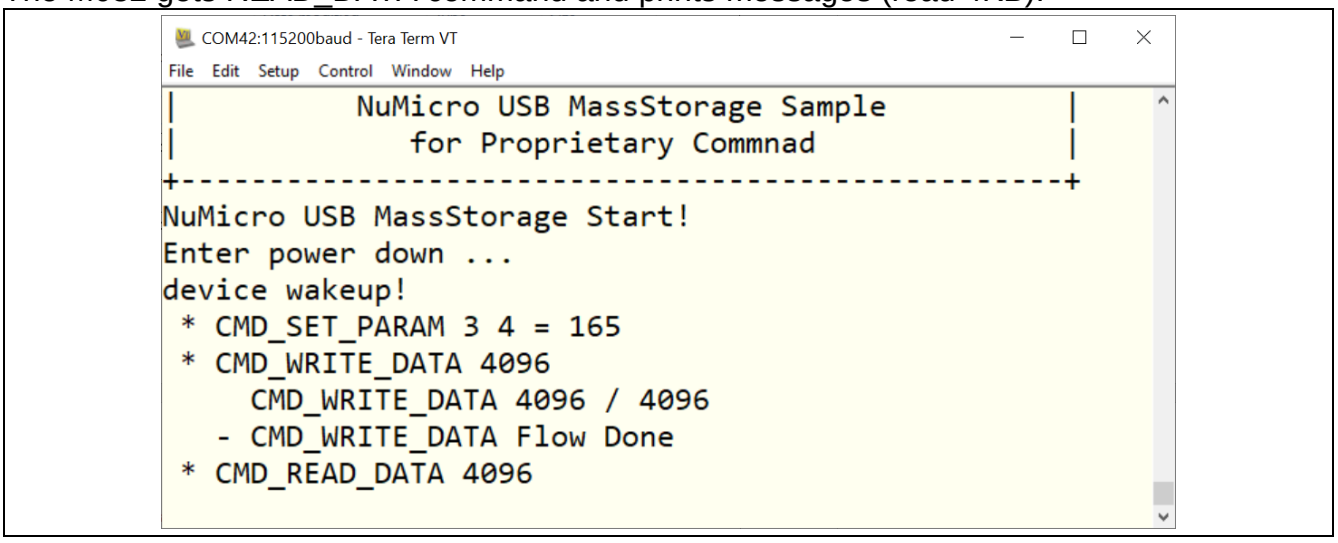

Figure 1-19 Get READ_DATA Command

Copy CMD_Write_4K.bin (the file size must be larger than the read data size) from the disk without buffering and the M032 will return data and print the status.

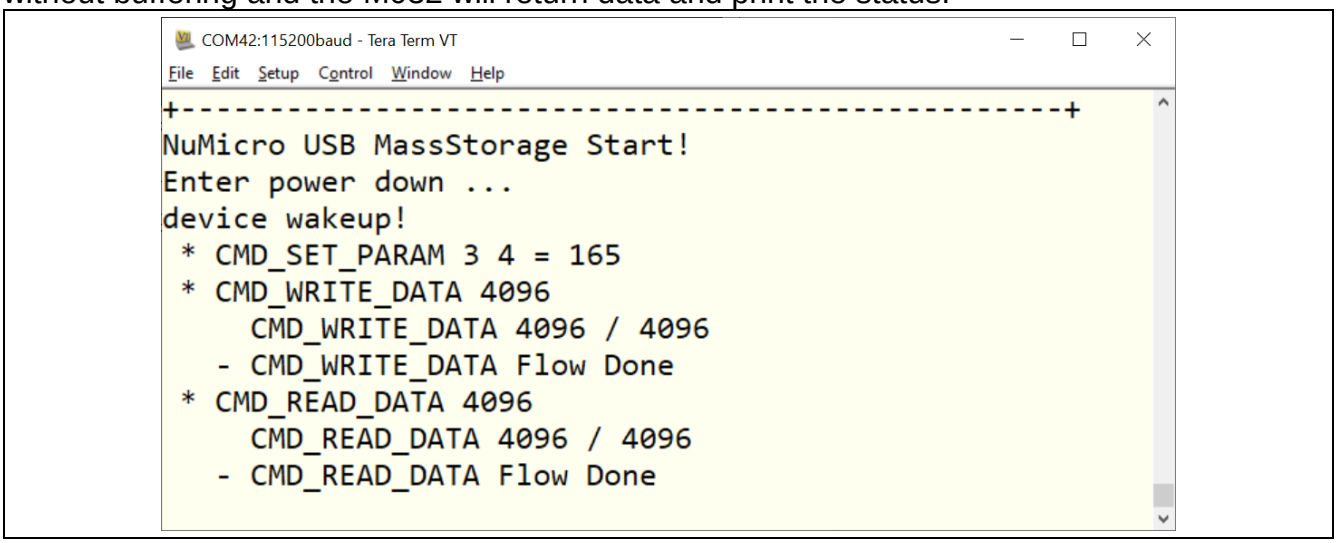

Figure 1-20 M032 Reruns Data

It's the same as the data previously written to M032.

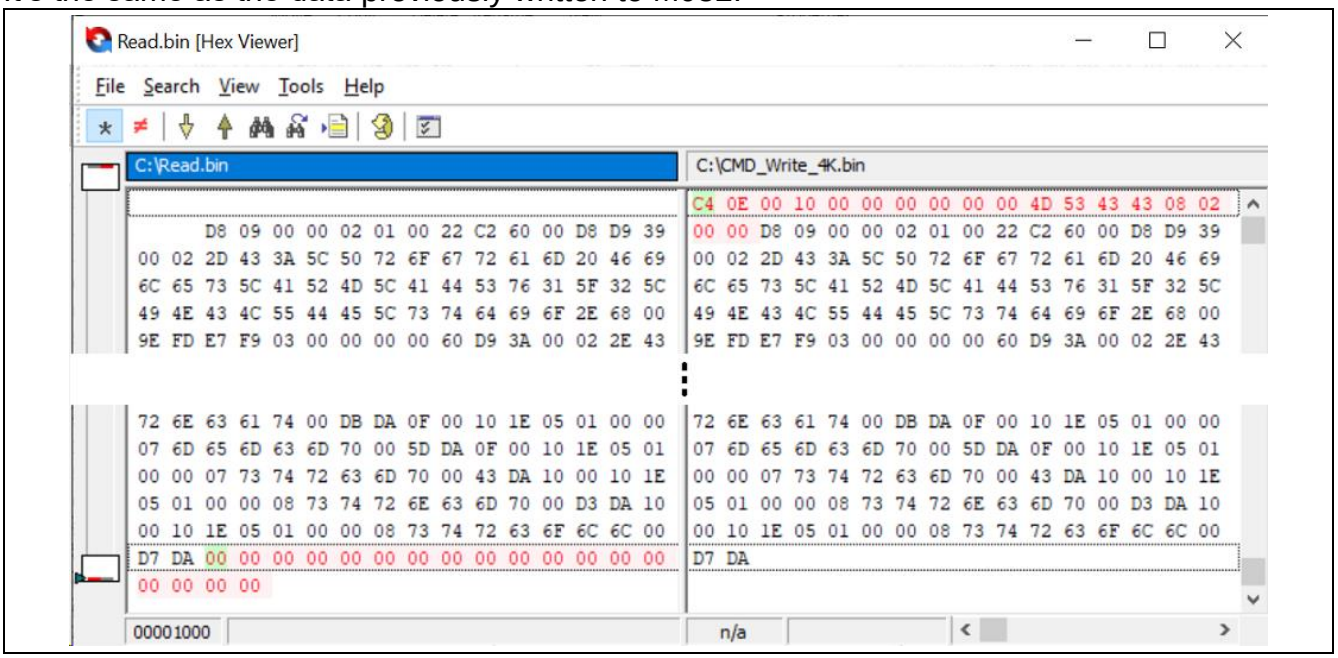

Figure 1-21 Comparison Results

1.2.5 Copy File without Buffering

1.2.5.1 Windows

Windows command - Xcopy

Xcopy Command Options	
Item	**Description**
/j	This option copies files without buffering, a feature useful for very big files. This option was first available in Windows 7.

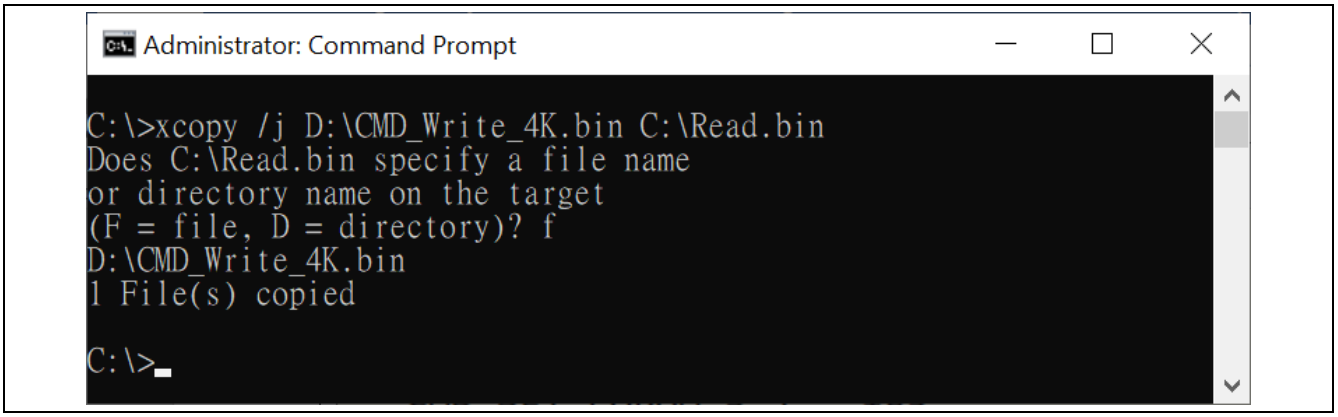

Figure 1-22 Copy File without Buffering by xcopy

CopyFile.exe uses CreateFileA function with FILE_FLAG_NO_BUFFERING. This example code provides a simple tool to copy file without buffering.

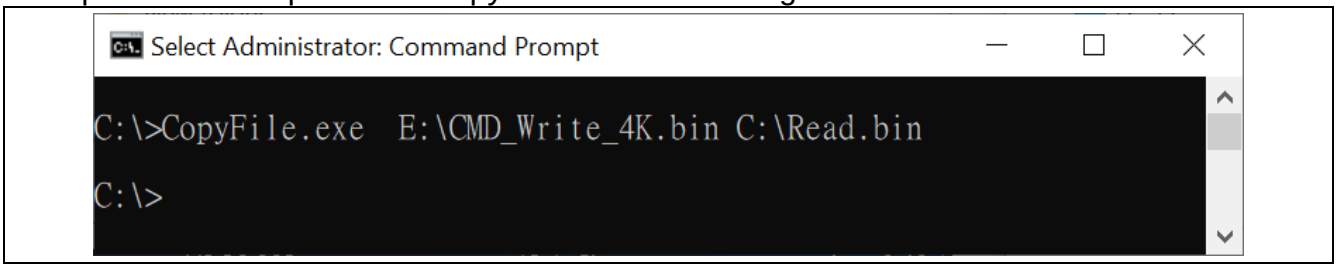

Figure 1-23 Copy File without Buffering by CopyFile Tool

1.2.5.2 Linux

Linux command - dd

dd Command	
Item	**Description**
iflg=direct	use direct I/O for data.

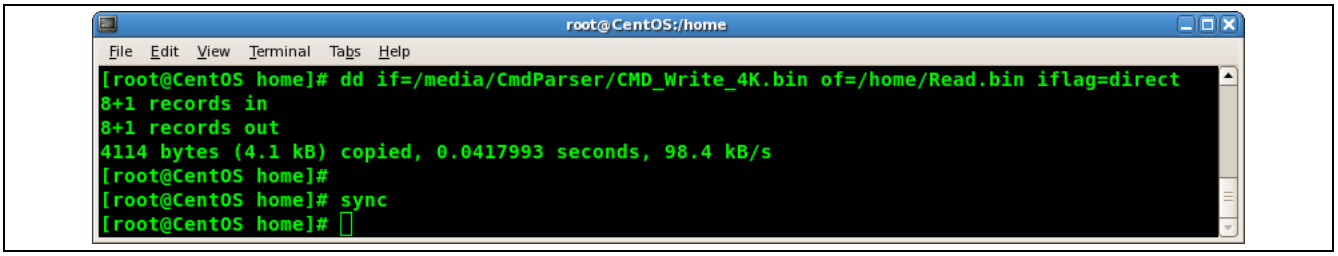

Figure 1-24 Copy File without Buffering by dd Command

2. Code Description

2.1 Virtual Disk Properties

The code for the virtual disk properties is in CmdParser.c as follows, which is the boot sector data for the virtual disk.

```
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
};
```

The definition of the virtual disk properties is in CmdParser.h as follows. Modifying the definition of “DISK_SIZE” can change the virtual disk size.

```
//#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)
```

The sector layout may be different between default setting and host format. User can un-comment the definition of “__FAT_INFO__” to get the properties of the sector layout (Boot sector) after host formatted (RSVD_SEC_CNT & FAT_SZ may be changed).

The code for getting disk properties is in CmdParser.c as follows. If it’s different between default setting and host formatted, please modify the definition of RSVD_SEC_CNT & FAT_SZ according to the message to make sure that the sector layout check mechanism is correct after host formatted.

```
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 Simulate FAT Information

The code for simulating reading / writing Boot Sector, File Allocation Table, and Root Directory is in CmdParser.c as follows.

```
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);
}
```

The FAT information is saved to SRAM. The data will be lost after reset or power-off. So, in other words, the virtual disk will also be reset to be empty after the example code is restarted.

2.3 Vendor Command Definition

The definition of Vendor Command structure is in CmdParser.h as follows.

```
/* 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 Vendor Command Flow Control

The MSC SCSI write command control flow is in CmdParser.c as follows. To change command set, the only thing that needs to do is modifying API – ProcessCommand and adding code for WRITE_DATA command done if needed.

```
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 */
            }
        }
    }
    …
}
```

The MSC SCSI read command control flow is in CmdParser.c as follows. According to previous read command (set the valuable - g_u8ReadType), the M032 returns the corresponding data when getting MSC read command.

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

The code for Vendor Command control is in CmdParser.c as follows.

```
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_u8WorkingBuff = g_u8Buff;
            g_u8ReadType = CMD_READ_DATA;
            break;
        }

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

3. Software and Hardware Requirements

3.1 Software Requirements

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

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M032KG V1.0

4. Directory Information

The directory structure is shown below.

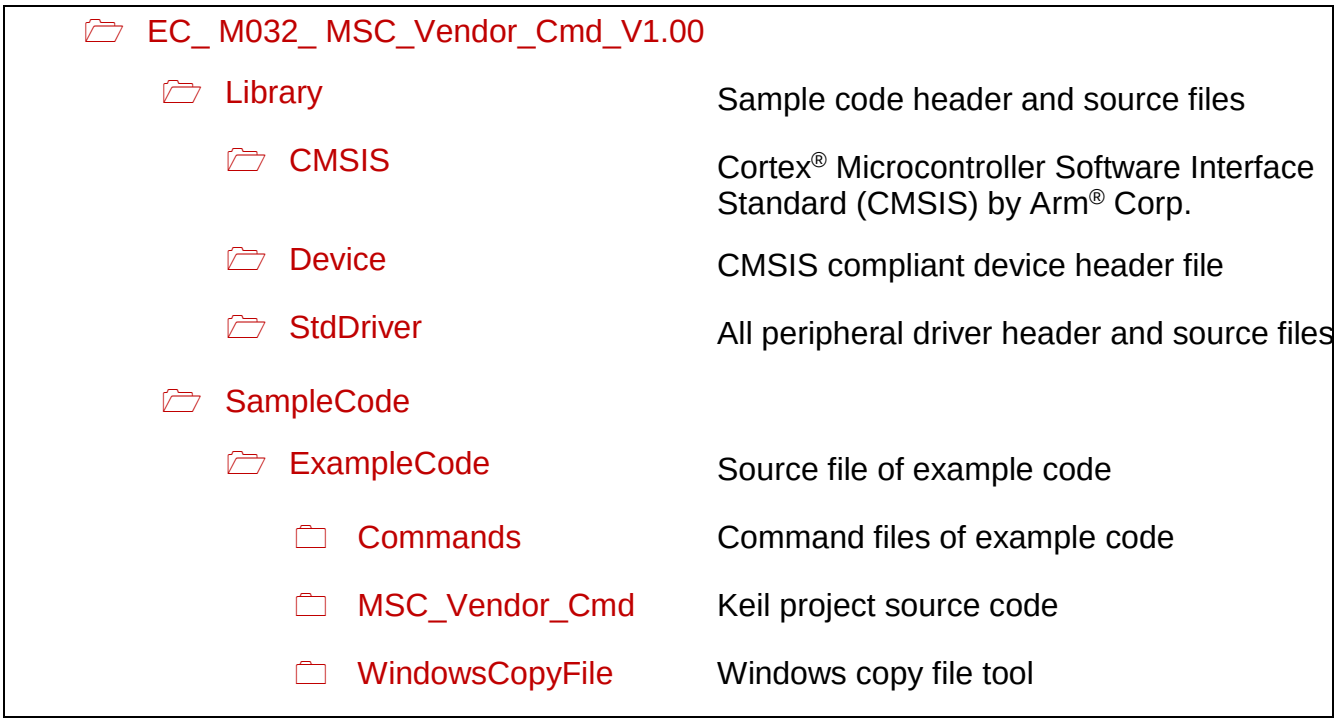

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *EC_M032_MSC_Vendor_Cmd.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2023.03.10	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.