使用SPI Flash 作为 USB 储存装置

NuMicro® 32位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码实现通过大容量储存接口将档案写入 SPI Flash
BSP 版本	M032_Series_BSP_CMSIS_V3.05.000
开发平台	NuMaker-M032KI V1.0

1. 概述

此范例程序使用微控制器 M032 作为 SPI Flash 烧录器，通过 USB 接口更新 SPI Flash。M032 透过 USB 通讯接口连接电脑，扮演大容量储存装置类别，并连接外部的 SPI Flash Memory 作为储存装置的媒介。通过使用大容量储存接口，不需要为 PC 提供任何工具或驱动程序即可实现将档案储存于 SPI Flash。将 USB 数据线插入设备， PC 将识别它为磁盘，使用者只需将档案 ”Update.bin” 复制到磁盘即可达到 ”将档案写入 SPI Flash” 。档案位置的信息储存于 SRAM，所以此磁盘会在程序重启后恢复为预设的状态。

1.1 原理

本次实验采用 USB Mass Storage Device Class (MSC) Bulk-Only Transfer 协议，根据该协议规范，Host 与 Device 交换资料的过程可分为三个状态阶段，分别为 Command Transport (CBW)，Data-Out / Data-In，以及 Status Transport (CSW)，其状态切换流程如图 1-1 所示。

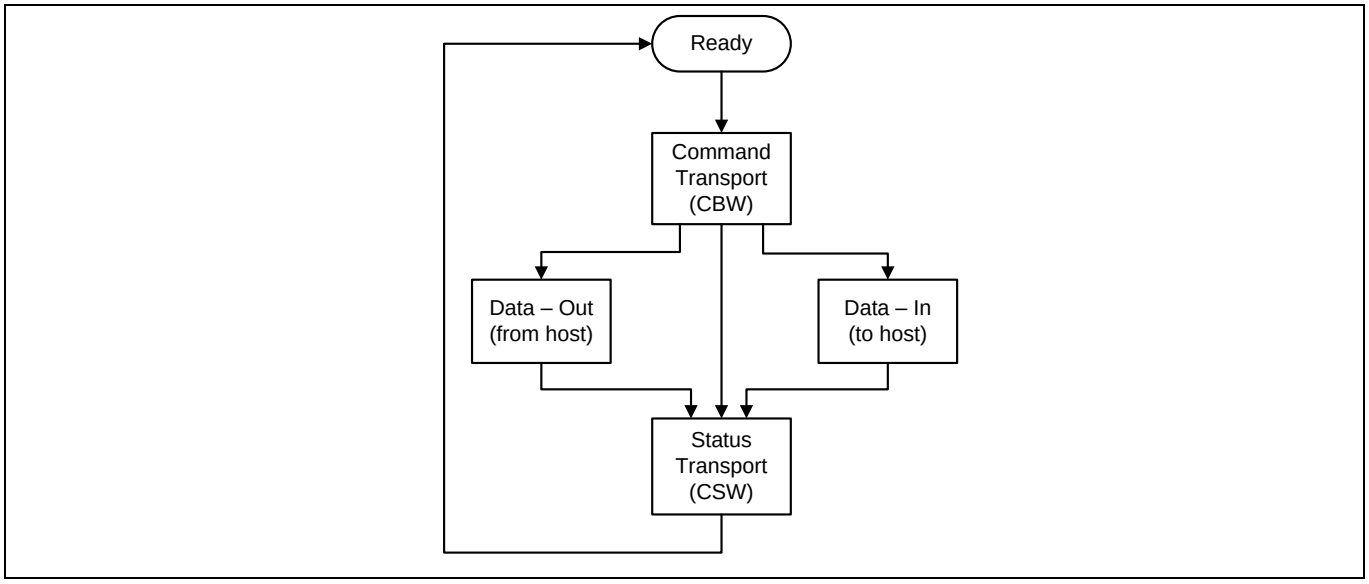

图 1-1 大容量存储类别 Command/Data/Status 协议

MSC 同时也定义了两个专属的类别请求，分别为 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 状态，通知装置准备接收 CBW 命令；Get Max LUN 则用以取得 Device 所支援的逻辑单元 (储存槽) 数量，其命令的详细内容与格式可参阅 USB Mass Storage Class Bulk-Only Transport Specification 文件 (https://www.usb.org/sites/default/files/usbmassbulk_10.pdf)。

1.1.1 Command Transport (CBW)

Command Transport 内含有一个 Transaction，资料方向为 OUT，由 Host 透过 Bulk-Out Endpoint 对 Device 送出指定命令的资料封包，并以 Command Block Wrapper (CBW) 之格式封装，封包格式如表 1-1 所示。在 CBW 封包当中，栏位 CBWCB 存放 Host 对于 Device 的储存媒体进行存取或控制的指令，该指令类型会决定下一个状态为 Data-In 或是 Data-Out，以及双方该传递的内容，详细的指令描述可参阅 UFI Command Specification 文件规范 (https://www.usb.org/sites/default/files/usbmass-ufi10.pdf)。

Byte \ bit	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11	*dCBWDataTransferLength*							
12	*bmCBWFlags*							
13	*Reserved(0)*				*bCBWLUN*			
14	*Reserved(0)*				*bCBWCBLengh*			
15-30	*CBWCB*							

表 1-1 Command Block Wrapper (CBW) 封包格式

1.1.2 Data-In / Data-Out

Data-In / Data-Out 状态下的行为受 CBW 内的命令所影响，举例来说，若 CBWCB 内的 UFI 指令为 WRITE，则代表由 Host 希望向 Device 的储存媒介写入档案资料，同时双方将状态切换为 Data-Out；反之若 UFI 指令为 READ，则表示 Host 向 Device 读取储存装置内的档案，由 Device 向 Host 送出资料封包，此时状态切换至 Data-In。 Data-In / Data-Out 阶段允许包含多个 Transaction，由 CBWCB 内的 dCBWDataTransferLength 栏位所影响。顺带一提，本次实验的储存媒介是采用外接的 SPI Flash Memory，微控制器接收到读取或写入的指令后，必须驱动外设 SPI 对外部 Flash Memory 的指定区块进行资料的读写。

1.1.3 Status Transport (CSW)

Status Transport 的用途是 Device 向 Host 回复本次 CBW 命令执行的状态，含有一个 Transaction，资料方向为 IN，此阶段的资料封包以 Command Status Wrapper (CSW) 之格式封装，封包格式如表 1-2 所示。 Host 可透过 CSW 内的 bCSWStatus 栏位判断 Device 在执行 CBW 命令的过程是否出现错误，bCSWStatus 数值意义如表 1-3 所示。

Byte \ bit	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11	*dCSWDataResidue*							
12	*bCSWStatus*							

表 1-2 Command Status Wrapper (CSW) 封包格式

Value	Descrition
00h	Command Passed (“good status”)
01h	Command Failed
02h	Phase Error
03 and 04h	Reserved (Obsolete)
05h to FF	Reserved

表 1-3 bCSWStatus 栏位内容

1.1.4 USB 端点配置

本范例需要配置四个端点给予 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

1.1.5 Serial Flash Memory

本范例代码使用 Winbond W25Q32JVSIG SPI Flash 作为 Mass Storage Device 的储存媒介，储存容量为 4 MB，最小的写入单位为一个 Page (256 B)，每 16 Page 可组成一个 Sector (4 KB)，每 256 Page 则组成一个 Block (64 KB)，其中 Sector 为最小的擦除单位。 SPI Flash 的控制方式非常简便，微控制器首先透过以下的命令集合对 SPI Flash 下达指令，包含 Read、Program 或是 Erase 等行为，接着双方会依据指令的类别，决定后续传递的资料的内容与长度。

Data Input Output	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Number of Clock(1-1-1)	8	8	8	8	8	8	8
Write Enable	06h						
Volatile SR Write Enable	50h						
Write Disable	04h						
Release Power-down / ID	ABh	Dummy	Dummy	Dummy	(ID7-ID0)(2)		
Manufacturer/Device ID	90h	Dummy	Dummy	00h	(MF7-MF0)	(ID7-ID0)	
JEDEC ID	9Fh	(MF7-MF0)	(ID15-ID8)	(ID7-ID0)			
Read Unique ID	4Bh	Dummy	Dummy	Dummy	Dummy	(UID63-0)	
Read Data	03h	A23-A16	A15-A8	A7-A0	(D7-D0)		
Fast Read	0Bh	A23-A16	A15-A8	A7-A0	Dummy	(D7-D0)	
Page Program	02h	A23-A16	A15-A8	A7-A0	D7-D0	D7-D0(3)	
Sector Erase (4KB)	20h	A23-A16	A15-A8	A7-A0			
Block Erase (32KB)	52h	A23-A16	A15-A8	A7-A0			
Block Erase (64KB)	D8h	A23-A16	A15-A8	A7-A0			
Chip Erase	C7h/60h						
Read Status Register-1	05h	(S7-S0)(2)					
Write Status Register-1(4)	01h	(S7-S0)(4)					
Read Status Register-2	35h	(S15-S8)(2)					
Write Status Register-2	31h	(S15-S8)					
Read Status Register-3	15h	(S23-S16)(2)					
Write Status Register-3	11h	(S23-S16)					
Read SFDP Register	5Ah	A23-A16	A15-A8	A7–A0	dummy	(D7-0)	
Erase Security Register(5)	44h	A23-A16	A15-A8	A7-A0			
Program Security Register(5)	42h	A23-A16	A15-A8	A7-A0	D7-D0	D7-D0(3)	
Read Security Register(5)	48h	A23-A16	A15-A8	A7-A0	Dummy	(D7-D0)	
Global Block Lock	7Eh						
Global Block Unlock	98h						
Read Block Lock	3Dh	A23-A16	A15-A8	A7-A0	(L7-L0)		
Individual Block Lock	36h	A23-A16	A15-A8	A7-A0			
Individual Block Unlock	39h	A23-A16	A15-A8	A7-A0			
Erase / Program Suspend	75h						
Erase / Program Resume	7Ah						
Power-down	B9h						
Enable Reset	66h						
Reset Device	99h						

表 1-4 Winbond SPI Flash 命令表

表 1-4为 Winbond W25Q32JVSIG 指令表，以 Page Program (02h) 与 Read Status Register (05h) 指令为例，可参阅图 1-2与图 1-3之时序，Master 首先送出 1 Byte 的指令 02h，接续 24-Bit 的资料写入位址与 256 Byte 的资料后，便完成 Page Program 的指令传输。之后，微控制器可再使用 Read Status Register (05h) 指令来询问 SPI Flash 的状态，来检查资料是否已经写入完毕。使用者可参阅 Winbond W25Q32JVSIG Datasheet 文件，内有各个指令详细的使用方法与说明。

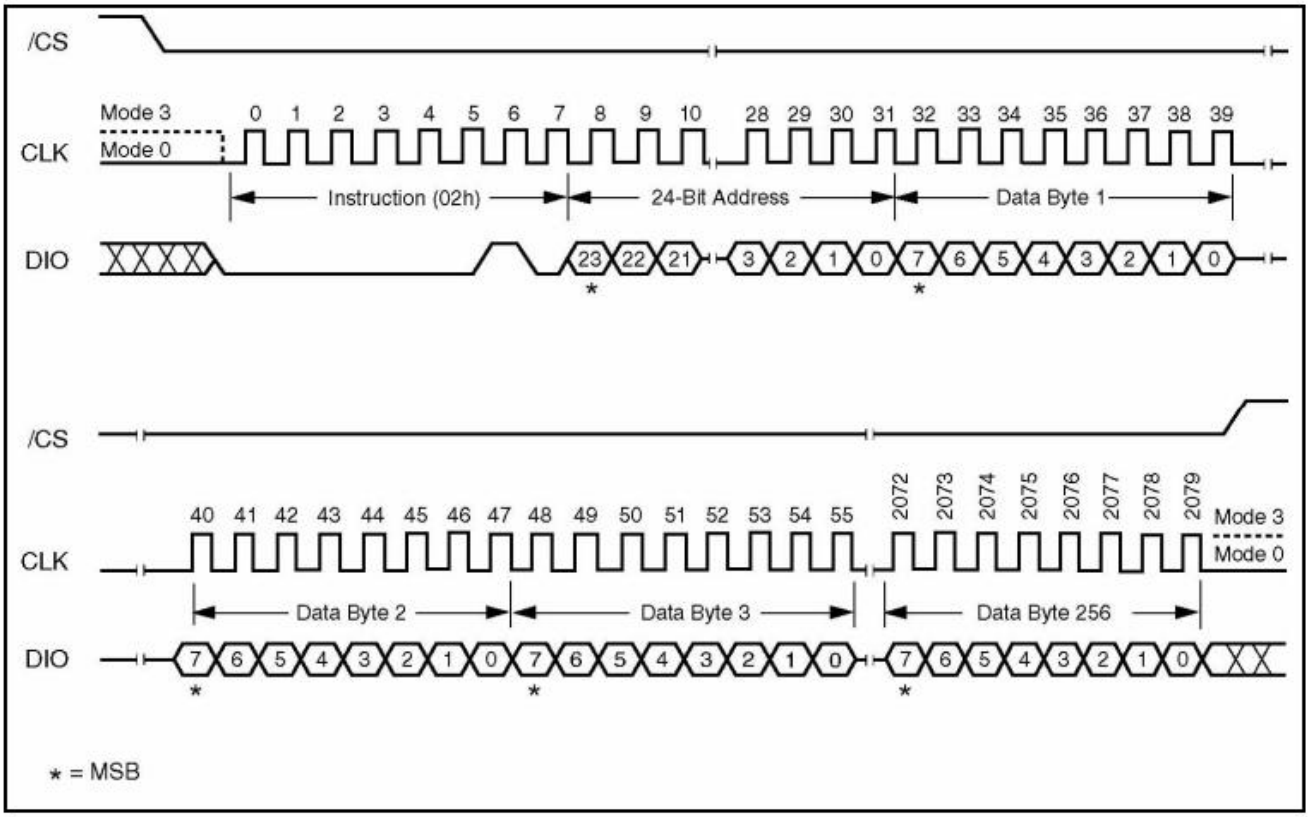

图 1-2 Page Program 命令时序图

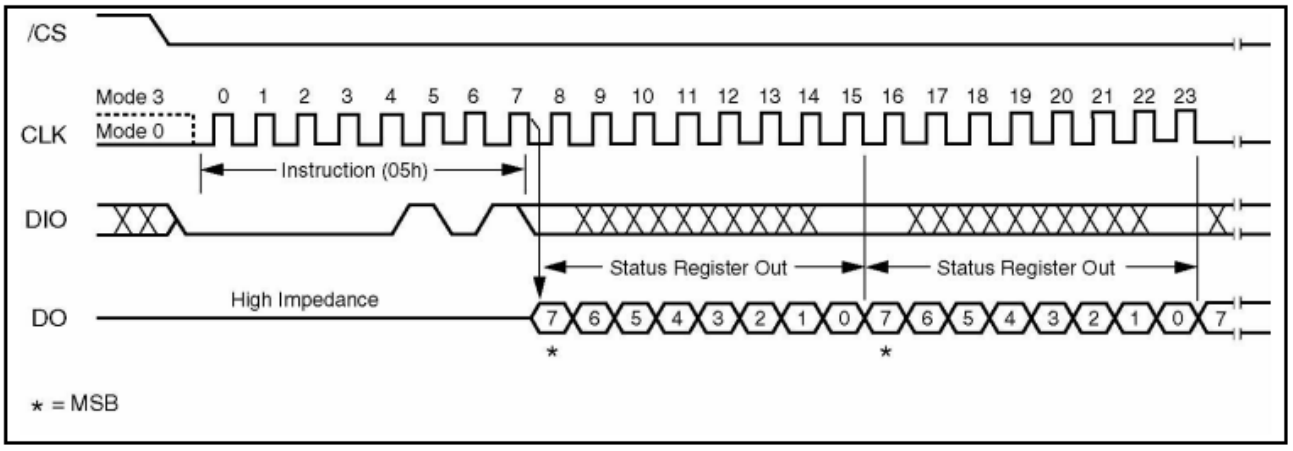

图 1-3 Read Status Register 命令时序图

1.2 执行结果

1.2.1 虚拟磁盘

当 M032 插入 PC USB 通讯接口时，PC会辨识出储存装置，DataFlash 显示容量为3.98 MB的

磁盘。使用者需要将档案命名为 Update.bin并将该档案拖曳或复制并贴上到此磁盘。

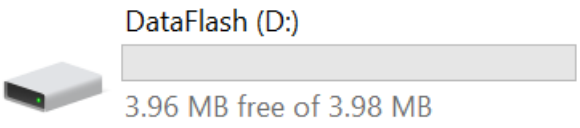

图 1-4 M032 虚拟磁盘

1.2.2 读取资料

从M032虚拟磁盘以无缓冲的方式读取Update.bin, 并将其与先前写入的资料比对结果一致。

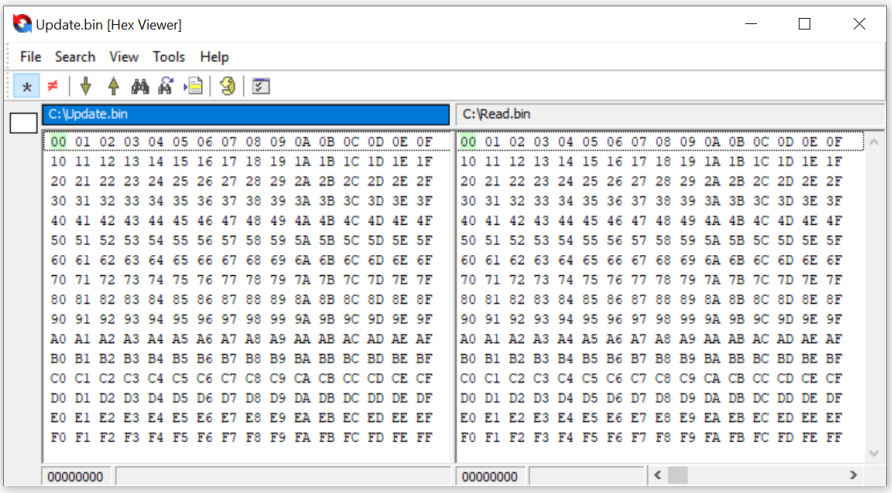

图 1-5 比较结果

1.2.3 无缓冲的 I/O 复制档案

1.2.3.1 Windows

Windows command - Xcopy

Xcopy Command	
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。此功能首次支援于 Window 7。

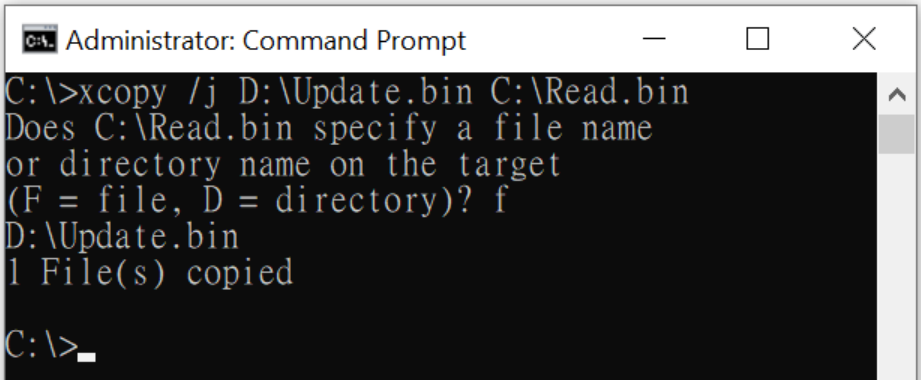

图 1-6 使用 xcopy 以无缓冲的 I/O 复制档案

1.2.3.2 Linux

Linux command - dd

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

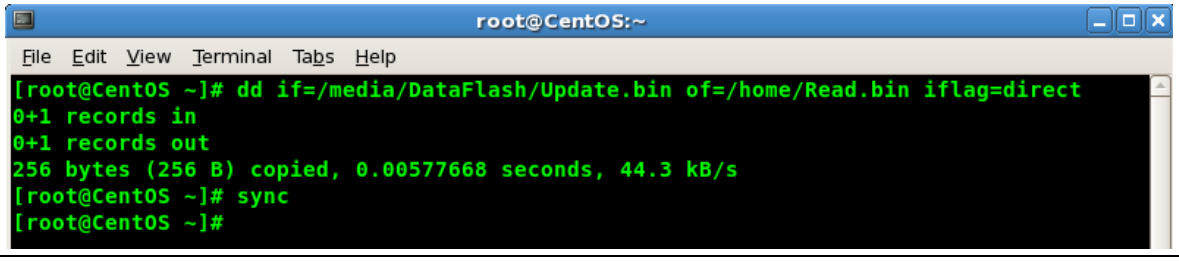

图 1-7 使用 dd 命令以无缓冲的 I/O 复制档案

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 DataFlashProg.c，以下是虚拟磁盘的启动磁区内容。

```
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 */
    0x08,
    /* 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,
    ...
    /* Signature value (0xaa55) */
    0x55, 0xAA
};
```

配置虚拟磁盘内容的定义，代码位于 DataFlashProg.h，修改“MB_SIZE”就可以修改虚拟磁盘的大小。

```
#define MB_SIZE                 4
#define DISK_SIZE               (MB_SIZE*1024 * 1024)
...
#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)
```

2.2 档案配置表(FAT)与读写 Flash

DataFlashRead() 负责回复 PC 端储存装置的档案配置表(FAT)并且根据 CBW 内的 Logic Block Address 以及 dCBWDataTransferLength 栏位来读取指定的磁区位址以及资料长度。

```
void DataFlashRead(uint32_t addr, uint32_t size, uint32_t buffer)
{
    /* This is low level read function of USB Mass Storage */
    uint32_t * pu32Buf = (uint32_t *)buffer;
    memset((uint8_t *)pu32Buf, 0, STORAGE_BUFFER_SIZE);

    if (addr == 0x00000000)
    {
#ifdef __FAT_INFO__
        if(g_ShowFat && GET_FAT_SZ != 0)
        {
            printf("\n");
            printf("Default FAT_SEC        0x%08X ", FAT_SEC);
            printf("Current FAT_SEC       0x%08X\n", GET_FAT_SEC);
            printf("Default FAT_SZ         0x%08X ",FAT_SZ);
            printf("Current FAT_SZ        0x%08X\n", GET_FAT_SZ);
            printf("Default ROOT_SEC_ADDR  0x%08X ",ROOT_SEC_ADDR);
            printf("Current ROOT_SEC_ADDR 0x%08X\n", GET_ROOT_SEC_ADDR);
            printf("Default DATA_SEC_ADDR  0x%08X ",DATA_SEC_ADDR);
            printf("Current DATA_SEC_ADDR 0x%08X\n", DATA_SEC_ADDR);
            g_ShowFat = 0;
        }
#endif
        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)/* % (FAT_SZ *
                        BYTE_PER_SEC)*/), 512);
```

```
        else if(addr > DATA_SEC_ADDR) /* Data */
        {
            SpiRead(u32SPI_RAddress, size, (uint32_t)pu32Buf);
            if(u32SPI_RAddress % 0x40000 == 0)
                DbgPrintf("Read from SPI 0x%08X - 0x%08X\n",u32SPI_RAddress, *(uint32_t
                          *)pu32Buf);
            u32SPI_RAddress+=size;
        }
    }
}
```

DataFlashWrite()调用底层函数 SpiWrite()执行写入操作，将档案储存到 SPI Flash 的指定区域。写入操作完成后，还会调用 SpiRead() 来验证储存在 SPI Flash 中的数据是否正确。

```
void DataFlashWrite(uint32_t addr, uint32_t size, uint32_t buffer)
{
    int k,l;
    /* This is low level write function of USB Mass Storage */
    if(addr >= DATA_SEC_ADDR)  /* Data */
    {
        if(addr == DATA_SEC_ADDR)
        {
            if(g_UpdateEnable == 0 && g_u8Windows == 0)
            {
                DbgPrintf("OS : Windows\n");
                g_u8Windows = 1;
            }
        }
        if(g_UpdateEnable || (g_u8Windows == 0 && g_u8MACOS == 0))
        {
            if(u32SPI_WAddress == 0)
            {
                if(g_u8Windows == 0 && g_u8MACOS == 0)
                    DbgPrintf("OS : Linux\n");
            }

            addr -= DATA_SEC_ADDR;

            SpiWrite(u32SPI_WAddress, STORAGE_BUFFER_SIZE, (uint32_t)buffer);

            SpiRead(u32SPI_WAddress, STORAGE_BUFFER_SIZE, (uint32_t)Storage_Block_Verify);

            if(memcmp((void *)buffer, (void *)Storage_Block_Verify,
               STORAGE_BUFFER_SIZE) !=0)
                DbgPrintf("Verify fail 0x%08X\n",u32SPI_WAddress);

            u32SPI_WAddress += STORAGE_BUFFER_SIZE;

            if(u32SPI_WAddress % 0x40000 == 0)
            {
                DbgPrintf("\rData Transferred %d KB",u32SPI_WAddress >> 10);
                if(g_file_size != 0)
                    DbgPrintf(" / %d KB",*g_file_size >>10);
            }
            if(g_file_size != 0 && u32SPI_WAddress >= *g_file_size)
            {
```

```
                DbgPrintf("\rData Transferred %d KB",u32SPI_WAddress >> 10);
                if(g_file_size != 0)
                    DbgPrintf(" / %d KB",*g_file_size >>10);

                g_TransferDone = 1;
                u32SPI_RAddress = 0;
                g_UpdateEnable = 0;
                DbgPrintf("\nTransferred Done\n");
            }
        }
        else
        {
            uint8_t *pu8Data = (uint8_t *)buffer;

            if(pu8Data[8] == 'M' && pu8Data[9] == 'a' && pu8Data[10] == 'c')
            {
                if(g_u8MACOS == 0)
                {
#if (ENABLE_DEBUG_MSG)
                    char *puchar = (char *)(&pu8Data[8]);
#endif
                    DbgPrintf("OS : %s\n",puchar);
                    g_u8MACOS = 1;
                }
                /* Finder progresses a copy of a file - HFS type code 'brok' & HFS creator
                    code 'MACS' */
                if(pu8Data[50] == 'b' && pu8Data[51] == 'r' && pu8Data[52] == 'o' &&
                    pu8Data[53] == 'k' && pu8Data[54] == 'M' && pu8Data[55] == 'A' &&
                    pu8Data[56] == 'C' && pu8Data[57] == 'S')
                {
                    g_u8MACOS_Update = 1;
                }
            }
        }
    }
    else if (addr == 0x00000000)
    {
        USBD_MemCopy(u8BootSectorData,(uint8_t *) buffer, 512);
#ifdef __FAT_INFO__
        g_ShowFat = 1;
#endif
    }
    else if (addr >= ROOT_SEC_ADDR && addr < DATA_SEC_ADDR) /* Root Directory */
    {
        USBD_MemCopy(u8DirData + (addr - ROOT_SEC_ADDR), (uint8_t *)buffer, 512);

        if((g_UpdateEnable == 0 && g_TransferDone == 0) || ((g_u8MACOS == 1 || g_u8Windows
           == 0)&& g_u8ShowFile != 1))           /* Check File name (When Update Function
           is not Enabled or MAC OS) */
        {
            for(k =0; k<32; k++)                 /* Check Root Directory */
            {
                for(l=0; l<FILE_NAME_LENGTH; l++)
                {
                    if(u8FileName[l] != 0)       /* Need to Check File Name - Character */
```

```
                        if(u8DirData[k*16+i8FileIndex[l]] != u8FileName[l])      /* Check
                           File Name */
                            break;
                }

                if(l == FILE_NAME_LENGTH)                                       /* Match */
                {
                    g_file_size = (uint32_t *)&u8DirData[k *16 + 60];  /* Get File Size */

                    if(*g_file_size == 0)
                    {
                        g_file_size = 0;
                        continue;
                    }
                    g_UpdateEnable = 1;

                    DbgPrintf("\nFile Name:");
                    for(l=0; l<FILE_NAME_LENGTH; l++)   /* Display the Update File Name */
                    {
                        if((u8DirData[k *16 + i8FileIndex[l]] != 0) && (u8DirData[k *16 +
                           i8FileIndex[l]+1] == 0) )
                            DbgPrintf("%c",u8DirData[k *16 + i8FileIndex[l]]);
                        else
                            break;
                    }

                    if(g_u8MACOS == 1 || g_u8Windows == 0)
                    {
                        g_u8ShowFile = 1;
                    }
                    DbgPrintf("\nFile Size: %dB\n",*g_file_size);
                    if(u32SPI_WAddress >= *g_file_size)
                    {
                        u32SPI_RAddress = 0;
                        g_TransferDone = 1;
                        DbgPrintf("\nTransferred Done\n");
                    }
                    break;
                }
            }
        }
    }
    else if (addr >= FAT_SEC_ADDR && addr < ROOT_SEC_ADDR)  /* File Allocation Table */
        USBD_MemCopy(u8FAT + ((addr - FAT_SEC_ADDR) /*% (FAT_SZ * BYTE_PER_SEC)*/),
                     (uint8_t *)buffer, 512);
}
```

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 5.28

3.2 硬件需求

- 电路组件
  - NuMaker-M032KI V1.0
  - Level1-Training
  - Micro USB cable
- 示意图

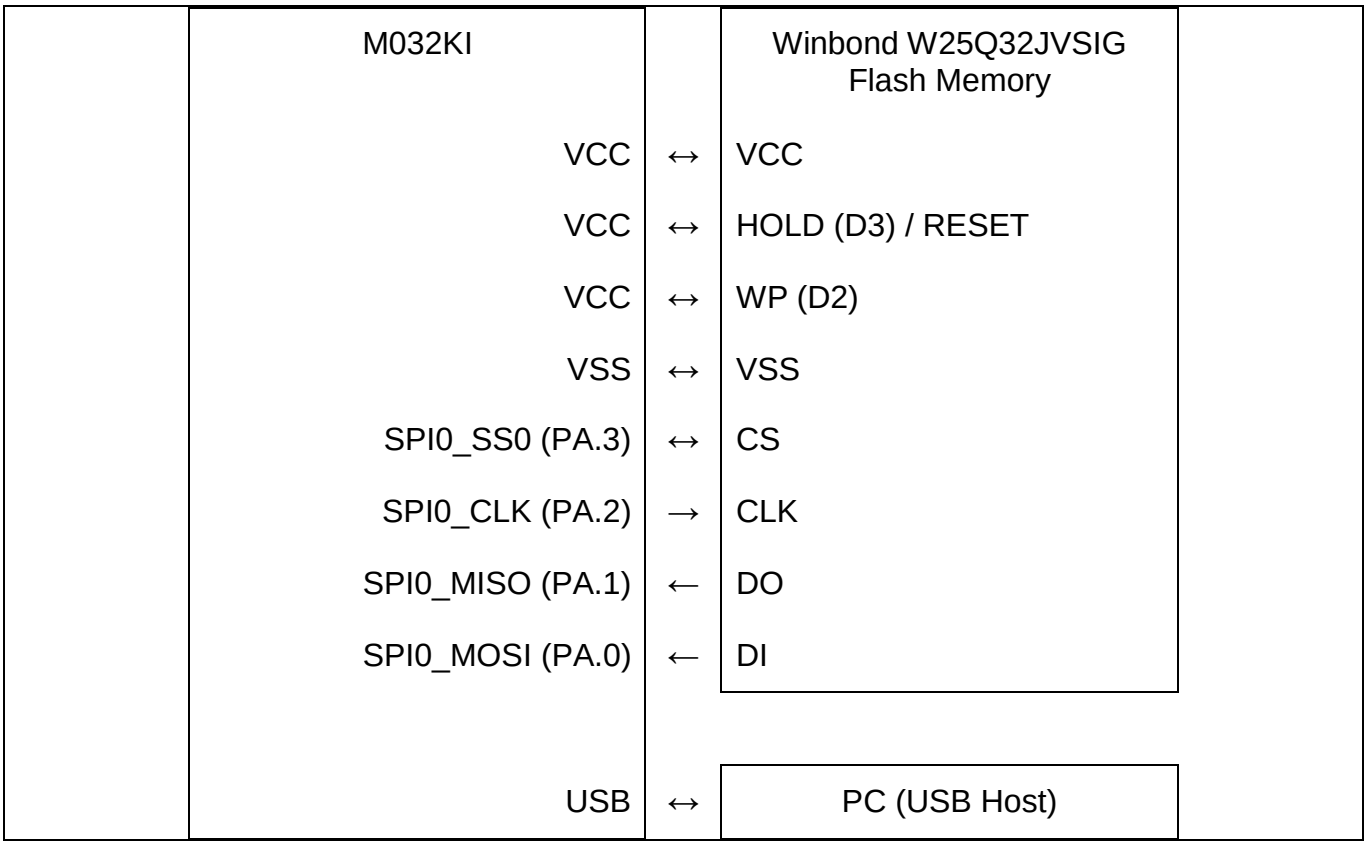

图 3-1 系统连接图

4. 目录信息

EC_M032_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

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 *M032_USBD_MSC_SPI_Flash.uvproj*。
2. 进入编译模式界面
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式界面
   - 执行代码

6. 修订纪录

Date	Revision	Description
2023.06.07	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.