小川工作室编写，本书为 LM3S 的 USB 芯片编写，上传的均为草稿，还有没修改，可能还有很多地方不足，希望各位网友原谅！

QQ:2609828265

TEL:15882446438

E-mail:paulhyde@126.com

第九章 Mass Storage 设备

9.1 Mass Storage 设备介绍

USB 的 Mass Storage 类是 USB 大容量储存设备类（Mass Storage Device Class）。专门用于大容量存储设备，比如 U 盘、移动硬盘、USB CD-ROM、读卡器等。在日常生活中经常用到。USB Mass Storage 设备开发相对简单。

9.2 MSC 数据类型

usbdmsc.h 中已经定义好 MSC 设备类中使用的所有数据类型和函数。下面介绍 MSC 设备类使用的数据类型。

```
typedef enum
{
    //设备加入
    USBDMSC_MEDIA_PRESENT,
    //设备移出
    USBDMSC_MEDIA_NOTPRESENT,
    //设备未知
    USBDMSC_MEDIA_UNKNOWN
}
tUSBDMSCMediaStatus;
```

tUSBDMSCMediaStatus，定义存储设备状态。定义在 usbdmsc.h。USBDMSCMediaChange()函数用手修改此状态。

```
typedef struct
{
    //open 函数指针，用户自己完成该函数编写
    void *(* Open)(unsigned long ulDrive);
    //Close 函数指针，用户自己完成该函数编
    void (* Close)(void * pvDrive);
    // BlockRead 函数指针，用于读取存储设备，用户自己完成该函数编
    unsigned long (* BlockRead)(void * pvDrive, unsigned char *pucData,
                                unsigned long ulSector,
                                unsigned long ulNumBlocks);
    // BlockWrite 函数指针，用于写入存储设备，用户自己完成该函数编
```

```
    unsigned long (* BlockWrite)(void * pvDrive, unsigned char *pucData,
                                 unsigned long ulSector,
                                 unsigned long ulNumBlocks);
    // 读取当前扇区数
    unsigned long (* NumBlocks)(void * pvDrive);
}
tMSCDMedia;
```

tMSCDMedia，存储设备底层操作驱动。用于 MSC 设备对存储设备操作。

```
typedef struct
{
    unsigned long ulUSBBase;
    tDeviceInfo *psDevInfo;
    tConfigDescriptor *psConfDescriptor;
    unsigned char ucErrorCode;
    unsigned char ucSenseKey;
    unsigned short usAddSenseCode;
    void *pvMedia;
    volatile tBoolean bConnected;
    unsigned long ulFlags;
    tUSBDMSCMediaStatus eMediaStatus;
    unsigned long pulBuffer[DEVICE_BLOCK_SIZE>>2];
    unsigned long ulBytesToTransfer;
    unsigned long ulCurrentLBA;
    unsigned char ucINEndpoint;
    unsigned char ucINDMA;
    unsigned char ucOUTEndpoint;
    unsigned char ucOUTDMA;
    unsigned char ucInterface;
    unsigned char ucSCSIState;
}
tMSCInstance;
```

tMSCInstance，MSC 设备类实例。定义了 MSC 设备类的 USB 基地址、设备信息、IN 端点、OUT 端点等信息。

```
typedef struct
{
    unsigned short usVID;
    unsigned short usPID;
    unsigned char pucVendor[8];
    unsigned char pucProduct[16];
    unsigned char pucVersion[4];
    unsigned short usMaxPowermA;
    unsigned char ucPwrAttributes;
    const unsigned char * const *ppStringDescriptors;
    unsigned long ulNumStringDescriptors;
```

```
    tMSCDMedia sMediaFunctions;
    tUSBCallback pfnEventCallback;
    tMSCInstance *psPrivateData;
}
tUSBDMSCDevice;
```

tUSBDMSCDevice，MSC 设备类，定义了 VID、PID、电源属性、字符串描述符等，还包括了一个 MSC 设备类实例。其它设备描述符、配置信息通过 API 函数储入 tMSCInstance 定义的 MSC 设备实例中。

9.3 API 函数

在 MSC 设备类 API 库中定义了 4 个函数，完成 USB MSC 设备初始化、配置及数据处理。下面为 usbdMSC.h 中定义的 API 函数：

```
void *USBDMSCInit(unsigned long ulIndex,
                  const tUSBDMSCDevice *psMSCDevice);
void *USBDMSCCompositeInit(unsigned long ulIndex,
                           const tUSBDMSCDevice *psMSCDevice);
void USBDMSCTerm(void *pvInstance);
void USBDMSCMediaChange(void *pvInstance,
                        tUSBDMSCMediaStatus eMediaStatus);
```

```
void *USBDMSCInit(unsigned long ulIndex,
                  const tUSBDMSCDevice *psMSCDevice);
```

作用：初始化 MSC 设备硬件、协议，把其它配置参数填入 psMSCDevice 实例中。

参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psMSCDevice，MSC 设备类。

返回：指向配置后的 tUSBDMSCDevice。

```
void *USBDMSCCompositeInit(unsigned long ulIndex,
                           const tUSBDMSCDevice *psMSCDevice);
```

作用：初始化 MSC 设备协议，本函数在 USBDMSCInit 中已经调用。

参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psMSCDevice，MSC 设备类。

返回：指向配置后的 tUSBDMSCDevice。

```
void USBDMSCTerm(void *pvInstance);
```

作用：结束 MSC 设备。

参数：pvInstance，指向 tUSBDMSCDevice。

返回：无。

```
void USBDMSCMediaChange(void *pvInstance,
                        tUSBDMSCMediaStatus eMediaStatus);
```

作用：存储设备状态改变。

参数：pvInstance，指向 tUSBDMSCDevice。

返回：无。

在这些函数中 USBDMSCInit 函数最重要并且使用最多，USBDMSCInit 第一次使用 MSC 设备时，用于初始化 MSC 设备的配置与控制。其它数据访问、控制处理由中断直接调用 tMSCDMedia 定义的 5 个底层驱动函数完成。

9.4 MSC 设备开发

MSC 设备开发只需要 5 步就能完成。如图 2 所示，MSC 设备配置（主要是字符串描述符）、callback 函数编写、存储设备底层驱动编写、USB 处理器初始化、数据处理。

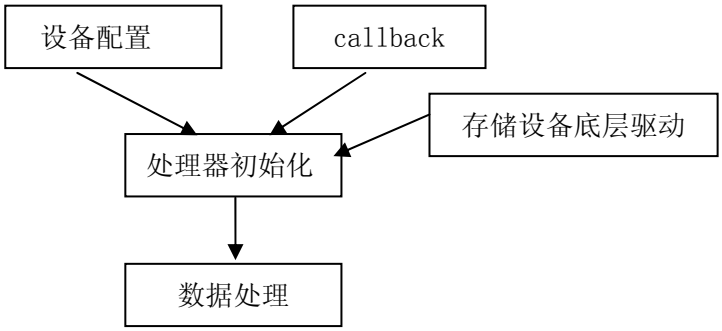


图 2

下面以“USB 转 UART”实例说明使用 USB 库开发 USB MSC 类过程：

第一步：MSC 设备配置（主要是字符串描述符），按字符串描述符标准完成串描述符配置，进而完成 MSC 设备配置。

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/rom.h"
#include "driverlib/systick.h"
#include "driverlib/usb.h"
#include "driverlib/udma.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdmsc.h"
#include "diskio.h"
#include "usbdsdcard.h"

//声明函数原型
unsigned long USBDMSCEventCallback(void *pvCBData, unsigned long ulEvent,
                                   unsigned long ulMsgParam,
                                   void *pvMsgData);

const tUSBDMSCDevice g_sMSCDevice;
//msc 状态
volatile enum
{
    MSC_DEV_DISCONNECTED,
    MSC_DEV_CONNECTED,
```

```
    MSC_DEV_IDLE,
    MSC_DEV_READ,
    MSC_DEV_WRITE,
}
g_eMSCState;
//全局标志
#define FLAG_UPDATE_STATUS      1
static unsigned long g_ulFlags;
//DMA
tDMAControlTable sDMAControlTable[64] __attribute__ ((aligned(1024)));

//*****************************************************************************
// 语言描述符
//*****************************************************************************
const unsigned char g_pLangDescriptor[] =
{
    4,
    USB_DTYPE_STRING,
    USBShort(USB_LANG_EN_US)
};
//*****************************************************************************
// 制造商 字符串 描述符
//*****************************************************************************
const unsigned char g_pManufacturerString[] =
{
    (17 + 1) * 2,
    USB_DTYPE_STRING,
    'T', 0, 'e', 0, 'x', 0, 'a', 0, 's', 0, ' ', 0, 'I', 0, 'n', 0, 's', 0,
    't', 0, 'r', 0, 'u', 0, 'm', 0, 'e', 0, 'n', 0, 't', 0, 's', 0,
};
//*****************************************************************************
//产品 字符串 描述符
//*****************************************************************************
const unsigned char g_pProductString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    'M', 0, 'a', 0, 's', 0, 's', 0, ' ', 0, 'S', 0, 't', 0, 'o', 0, 'r', 0,
    'a', 0, 'g', 0, 'e', 0, ' ', 0, 'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0,
    'e', 0
};
//*****************************************************************************
//   产品 序列号 描述符
//*****************************************************************************
```

```
const unsigned char g_pSerialNumberString[] =
{
    (8 + 1) * 2,
    USB_DTYPE_STRING,
    '1', 0, '2', 0, '3', 0, '4', 0, '5', 0, '6', 0, '7', 0, '8', 0
};

//*****************************************************************************
// 设备接口字符串描述符
//*****************************************************************************
const unsigned char g_pDataInterfaceString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    'B', 0, 'u', 0, 'l', 0, 'k', 0, ' ', 0, 'D', 0, 'a', 0, 't', 0,
    'a', 0, ' ', 0, 'I', 0, 'n', 0, 't', 0, 'e', 0, 'r', 0, 'f', 0,
    'a', 0, 'c', 0, 'e', 0
};
//*****************************************************************************
//   设备配置字符串描述符
//*****************************************************************************
const unsigned char g_pConfigString[] =
{
    (23 + 1) * 2,
    USB_DTYPE_STRING,
    'B', 0, 'u', 0, 'l', 0, 'k', 0, ' ', 0, 'D', 0, 'a', 0, 't', 0,
    'a', 0, ' ', 0, 'C', 0, 'o', 0, 'n', 0, 'f', 0, 'i', 0, 'g', 0,
    'u', 0, 'r', 0, 'a', 0, 't', 0, 'i', 0, 'o', 0, 'n', 0
};
//*****************************************************************************
// 字符串描述符集合
//*****************************************************************************
const unsigned char * const g_pStringDescriptors[] =
{
    g_pLangDescriptor,
    g_pManufacturerString,
    g_pProductString,
    g_pSerialNumberString,
    g_pDataInterfaceString,
    g_pConfigString
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /               \
                                sizeof(unsigned char *))
//*****************************************************************************
```

```
//MSC 实例，配置并为设备信息提供空间
//*****************************************************************************
tMSCInstance g_sMSCInstance;
//*****************************************************************************
//设备配置
//*****************************************************************************
const tUSBDMSCDevice g_sMSCDevice =
{
    USB_VID_STELLARIS,
    USB_PID_MSC,
    "TI      ",
    "Mass Storage    ",
    "1.00",
    500,
    USB_CONF_ATTR_SELF_PWR,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    {
        USBDMSCStorageOpen,
        USBDMSCStorageClose,
        USBDMSCStorageRead,
        USBDMSCStorageWrite,
        USBDMSCStorageNumBlocks
    },
    USBDMSCEventCallback,
    &g_sMSCInstance
};
#define MSC_BUFFER_SIZE 512
```

第二步：完成 Callback 函数。Callback 函数用于返回数据处理状态：

```
//*****************************************************************************
//callback 函数
//*****************************************************************************
unsigned long USBDMSCEventCallback(void *pvCBData, unsigned long ulEvent,
                    unsigned long ulMsgParam, void *pvMsgData)
{
    switch(ulEvent)
    {
        // 正在写数据到存储设备.
        case USBD_MSC_EVENT_WRITING:
        {
            if(g_eMSCState != MSC_DEV_WRITE)
            {
                g_eMSCState = MSC_DEV_WRITE;
                g_ulFlags |= FLAG_UPDATE_STATUS;
```

```
            }
            break;
        }
        //读取数据.
        case USBD_MSC_EVENT_READING:
        {
            if(g_eMSCState != MSC_DEV_READ)
            {
                g_eMSCState = MSC_DEV_READ;
                g_ulFlags |= FLAG_UPDATE_STATUS;
            }

            break;
        }
     //空闲
        case USBD_MSC_EVENT_IDLE:
        default:
        {
            break;
        }
    }
    return(0);
}
```

第三步：完成接口函数编写：

```
#define SDCARD_PRESENT          0x00000001
#define SDCARD_IN_USE           0x00000002

struct
{
    unsigned long ulFlags;
}
g_sDriveInformation;

//*****************************************************************************
//   打开存储设备
//*****************************************************************************
void *    USBDMSCStorageOpen(unsigned long ulDrive)
{
    unsigned char ucPower;
    unsigned long ulTemp;
    // 检查是否在使用.
    if(g_sDriveInformation.ulFlags & SDCARD_IN_USE)
    {
```

```
        return(0);
    }
    // 初始化存储设备.
    ulTemp = disk_initialize(0);
    if(ulTemp == RES_OK)
    {
        //打开电源
        ucPower = 1;
        disk_ioctl(0, CTRL_POWER, &ucPower);
        //设置标志.
        g_sDriveInformation.ulFlags = SDCARD_PRESENT | SDCARD_IN_USE;
    }
    else if(ulTemp == STA_NODISK)
    {
        // 没有存储设备.
        g_sDriveInformation.ulFlags = SDCARD_IN_USE;
    }
    else
    {
        return(0);
    }
    return((void *)&g_sDriveInformation);
}
//*****************************************************************************
// 关闭存储设备
//*****************************************************************************
void USBDMSCStorageClose(void * pvDrive)
{
    unsigned char ucPower;
    g_sDriveInformation.ulFlags = 0;
    ucPower = 0;
    disk_ioctl(0, CTRL_POWER, &ucPower);
}
//*****************************************************************************
//   读取扇区数据
//*****************************************************************************
unsigned long USBDMSCStorageRead(void * pvDrive,
                                 unsigned char *pucData,
                                 unsigned long ulSector,
                                 unsigned long ulNumBlocks)
{
    if(disk_read (0, pucData, ulSector, ulNumBlocks) == RES_OK)
    {
        return(ulNumBlocks * 512);
```

```
    }
    return(0);
}
//*****************************************************************************
// 写数据到扇区
//*****************************************************************************
unsigned long USBDMSCStorageWrite(void * pvDrive,
                                  unsigned char *pucData,
                                  unsigned long ulSector,
                                  unsigned long ulNumBlocks)
{
    if(disk_write(0, pucData, ulSector, ulNumBlocks) == RES_OK)
    {
        return(ulNumBlocks * 512);
    }
    return(0);
}
//*****************************************************************************
// 获取当前扇区
//*****************************************************************************
unsigned long USBDMSCStorageNumBlocks(void * pvDrive)
{
    unsigned long ulSectorCount;
    disk_ioctl(0, GET_SECTOR_COUNT, &ulSectorCount);
    return(ulSectorCount);
}
#define USBDMSC_IDLE            0x00000000
#define USBDMSC_NOT_PRESENT     0x00000001
//*****************************************************************************
// 存储设备当前状态
//*****************************************************************************
unsigned long USBDMSCStorageStatus(void * pvDrive);
```

第四步：系统初始化，配置内核电压、系统主频、使能端口、等。系统初始化：

```
//系统初始化。
SysCtlLDOSet(SYSCTL_LDO_2_75V);
 SysCtlClockSet(SYSCTL_XTAL_8MHZ | SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL  | SYSCTL_OSC_MAIN );

 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
 GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,0xf0);
 GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,0x0f);
 HWREG(GPIO_PORTF_BASE+GPIO_O_PUR) |= 0x0f;
 // ucDMA 配置
 SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
 SysCtlDelay(10);
```

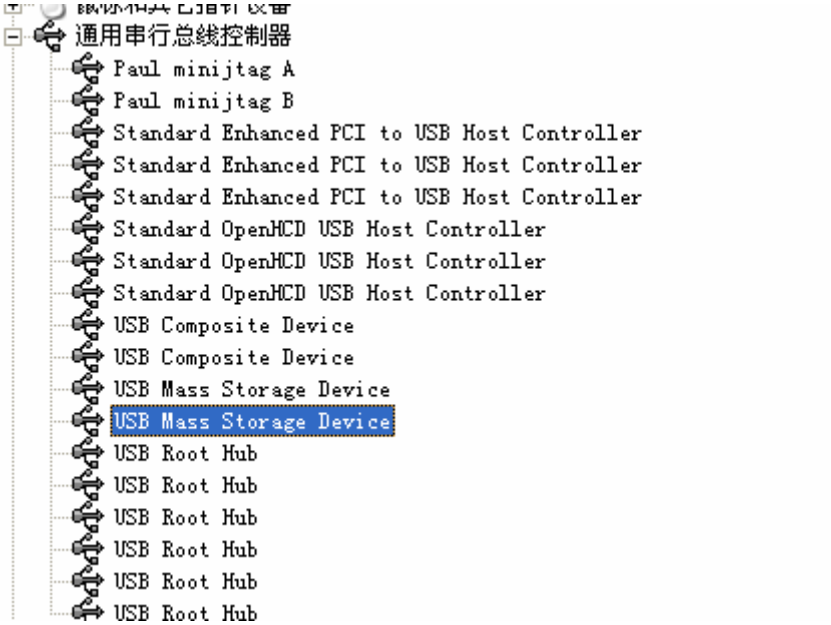
```
uDMAControlBaseSet(&sDMAControlTable[0]);
uDMAEnable();
g_ulFlags = 0;
g_eMSCState = MSC_DEV_IDLE;
//msc 设备初始化
USBDMSCInit(0, (tUSBDMSCDevice *)&g_sMSCDevice);
//初始化存储设备
disk_initialize(0);
```

第五步：状态处理，其它控制。

```
while(1)
{
        switch(g_eMSCState)
        {
            case MSC_DEV_READ:
            {
                if(g_ulFlags & FLAG_UPDATE_STATUS)
                {
                    g_ulFlags &= ~FLAG_UPDATE_STATUS;
                }
                break;
            }
            case MSC_DEV_WRITE:
            {
                if(g_ulFlags & FLAG_UPDATE_STATUS)
                {
                    g_ulFlags &= ~FLAG_UPDATE_STATUS;
                }
                break;
            }
            case MSC_DEV_IDLE:
            default:
            {
                break;
            }
        }
    }
```

使用上面五步就完成MSC设备开发。MSC设备开发时要加入两个lib库函数：usblib.lib和DriverLib.lib，在启动代码中加入USB0DeviceIntHandler中断服务函数。以上MSC设备开发完成，在Win xp下运行效果如下图所示：

在枚举过程中可以看出，在电脑右下脚可以看到“Mass Storage Device”字样，标示正在进行枚举，并手手动安装驱动。枚举成功后，在“设备管理器”的“通用串行总线控制器”中看到“USB Mass Storage Device”设备，如下图。现在 MSC 设备可以正式使用。



MSC 设备可以在“我的电脑”中“有可移动存储的设备”找到：

MSC 设备开发源码较多，下面只列出一部分如下：

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/rom.h"
#include "driverlib/systick.h"
```

```
#include "driverlib/usb.h"
#include "driverlib/udma.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdmsc.h"
#include "diskio.h"
#include "usbdsdcard.h"

//声明函数原型
unsigned long USBDMSCEventCallback(void *pvCBData, unsigned long ulEvent,
                                   unsigned long ulMsgParam,
                                   void *pvMsgData);

const tUSBDMSCDevice g_sMSCDevice;
//msc 状态
volatile enum
{
    MSC_DEV_DISCONNECTED,
    MSC_DEV_CONNECTED,
    MSC_DEV_IDLE,
    MSC_DEV_READ,
    MSC_DEV_WRITE,
}
g_eMSCState;
//全局标志
#define FLAG_UPDATE_STATUS      1
static unsigned long g_ulFlags;
//DMA
tDMAControlTable sDMAControlTable[64] __attribute__ ((aligned(1024)));

//*****************************************************************************
// 语言描述符
//*****************************************************************************
const unsigned char g_pLangDescriptor[] =
{
    4,
    USB_DTYPE_STRING,
    USBShort(USB_LANG_EN_US)
};
//*****************************************************************************
// 制造商 字符串 描述符
//*****************************************************************************
const unsigned char g_pManufacturerString[] =
```

```
{
    (17 + 1) * 2,
    USB_DTYPE_STRING,
    'T', 0, 'e', 0, 'x', 0, 'a', 0, 's', 0, ' ', 0, 'I', 0, 'n', 0, 's', 0,
    't', 0, 'r', 0, 'u', 0, 'm', 0, 'e', 0, 'n', 0, 't', 0, 's', 0,
};
//*****************************************************************************
//产品 字符串 描述符
//*****************************************************************************
const unsigned char g_pProductString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    'M', 0, 'a', 0, 's', 0, 's', 0, ' ', 0, 'S', 0, 't', 0, 'o', 0, 'r', 0,
    'a', 0, 'g', 0, 'e', 0, ' ', 0, 'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0,
    'e', 0
};
//*****************************************************************************
//  产品 序列号 描述符
//*****************************************************************************
const unsigned char g_pSerialNumberString[] =
{
    (8 + 1) * 2,
    USB_DTYPE_STRING,
    '1', 0, '2', 0, '3', 0, '4', 0, '5', 0, '6', 0, '7', 0, '8', 0
};

//*****************************************************************************
// 设备接口字符串描述符
//*****************************************************************************
const unsigned char g_pDataInterfaceString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    'B', 0, 'u', 0, 'l', 0, 'k', 0, ' ', 0, 'D', 0, 'a', 0, 't', 0,
    'a', 0, ' ', 0, 'I', 0, 'n', 0, 't', 0, 'e', 0, 'r', 0, 'f', 0,
    'a', 0, 'c', 0, 'e', 0
};
//*****************************************************************************
//  设备配置字符串描述符
//*****************************************************************************
const unsigned char g_pConfigString[] =
{
    (23 + 1) * 2,
```

```
    USB_DTYPE_STRING,
    'B', 0, 'u', 0, 'l', 0, 'k', 0, ' ', 0, 'D', 0, 'a', 0, 't', 0,
    'a', 0, ' ', 0, 'C', 0, 'o', 0, 'n', 0, 'f', 0, 'i', 0, 'g', 0,
    'u', 0, 'r', 0, 'a', 0, 't', 0, 'i', 0, 'o', 0, 'n', 0
};
//*****************************************************************************
// 字符串描述符集合
//*****************************************************************************
const unsigned char * const g_pStringDescriptors[] =
{
    g_pLangDescriptor,
    g_pManufacturerString,
    g_pProductString,
    g_pSerialNumberString,
    g_pDataInterfaceString,
    g_pConfigString
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /                \
                                sizeof(unsigned char *))
//*****************************************************************************
//MSC 实例，配置并为设备信息提供空间
//*****************************************************************************
tMSCInstance g_sMSCInstance;
//*****************************************************************************
//设备配置
//*****************************************************************************
const tUSBDMSCDevice g_sMSCDevice =
{
    USB_VID_STELLARIS,
    USB_PID_MSC,
    "TI      ",
    "Mass Storage    ",
    "1.00",
    500,
    USB_CONF_ATTR_SELF_PWR,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    {
        USBDMSCStorageOpen,
        USBDMSCStorageClose,
        USBDMSCStorageRead,
        USBDMSCStorageWrite,
        USBDMSCStorageNumBlocks
    },
```

```
    USBDMSCEventCallback,
    &g_sMSCInstance
};
#define MSC_BUFFER_SIZE 512
//*****************************************************************************
//callback 函数
//*****************************************************************************
unsigned long USBDMSCEventCallback(void *pvCBData, unsigned long ulEvent,
                    unsigned long ulMsgParam, void *pvMsgData)
{
    switch(ulEvent)
    {
        // 正在写数据到存储设备.
        case USBD_MSC_EVENT_WRITING:
        {
            if(g_eMSCState != MSC_DEV_WRITE)
            {
                g_eMSCState = MSC_DEV_WRITE;
                g_ulFlags |= FLAG_UPDATE_STATUS;
            }
            break;
        }
        //读取数据.
        case USBD_MSC_EVENT_READING:
        {
            if(g_eMSCState != MSC_DEV_READ)
            {
                g_eMSCState = MSC_DEV_READ;
                g_ulFlags |= FLAG_UPDATE_STATUS;
            }

            break;
        }
         //空闲
        case USBD_MSC_EVENT_IDLE:
        default:
        {
            break;
        }
    }
    return(0);
}

//*****************************************************************************
```

```
//主函数
//*****************************************************************************
int main(void)
{
    //系统初始化。
    SysCtlLDOSet(SYSCTL_LDO_2_75V);
    SysCtlClockSet(SYSCTL_XTAL_8MHZ | SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL |
SYSCTL_OSC_MAIN );
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,0xf0);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,0x0f);
    HWREG(GPIO_PORTF_BASE+GPIO_O_PUR) |= 0x0f;
    // ucDMA 配置
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
    SysCtlDelay(10);
    uDMAControlBaseSet(&sDMAControlTable[0]);
    uDMAEnable();
    g_ulFlags = 0;
    g_eMSCState = MSC_DEV_IDLE;
    //msc 设备初始化
    USBDMSCInit(0, (tUSBDMSCDevice *)&g_sMSCDevice);
    //初始化存储设备
    disk_initialize(0);
    while(1)
    {
       switch(g_eMSCState)
       {
           case MSC_DEV_READ:
           {
               if(g_ulFlags & FLAG_UPDATE_STATUS)
               {
                   g_ulFlags &= ~FLAG_UPDATE_STATUS;
               }
               break;
           }
           case MSC_DEV_WRITE:
           {
               if(g_ulFlags & FLAG_UPDATE_STATUS)
               {
                   g_ulFlags &= ~FLAG_UPDATE_STATUS;
               }
               break;
           }
           case MSC_DEV_IDLE:
```

```
            default:
            {
                break;
            }
        }
    }
}
```