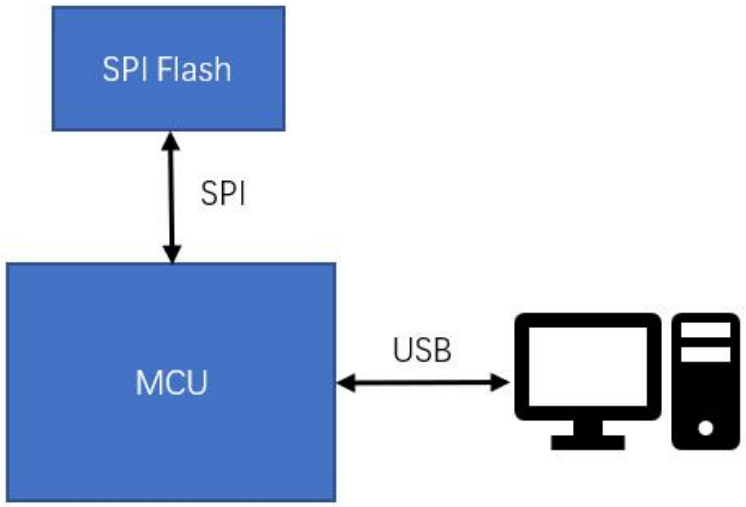
航芯干货分享 | 基于 SPI Flash 的 U 盘程序，从 STM32F103 到 ACM32F403

前言

本项目是以 SPI Flash（如 W25Q128 等）存储元件作为存储单元，MCU 主控完成 USB 接口通信并根据 SCSI 协议实现 U 盘功能。其结构如下图所示：



SPI Flash 部分移植

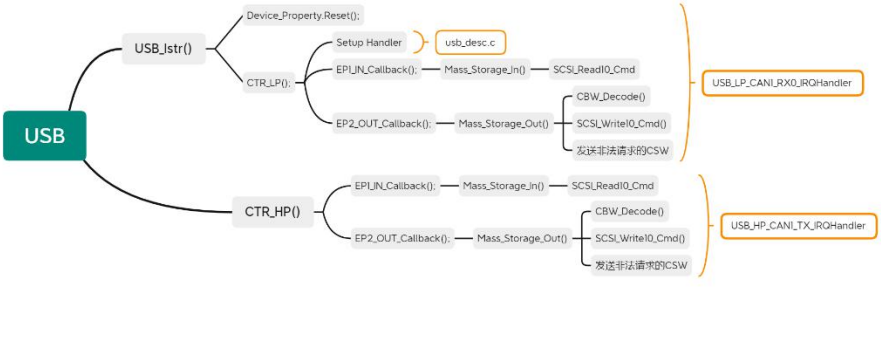
SPI 功能部分相对简单，ACM32F403 的接口引脚和 STM32F103 的相同，可直接对接，按照 ACM32F403 的说明对 SPI 接口进行初始化，并对底层读写函数进行更改即可。

USB 部分移植

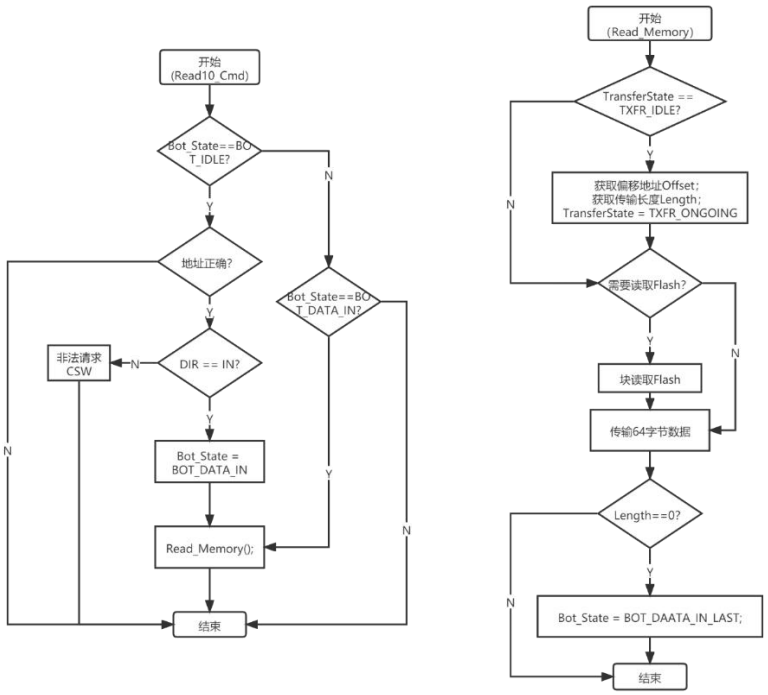
1. STM32F103 代码结构

在 ST 的芯片上，USB 的数据是由两个中断，USB_LP_CAN1_RX0_IRQHandler 和 USB_HP_CAN1_TX_IRQHandler 来进行，其中高优先级中断

（USB_HP_CAN1_TX_IRQHandler）用于处理同步(Isochronous)模式传输或双缓冲块(Bulk)传输模式下的正确传输事件，而低优先级中断（USB_LP_CAN1_RX0_IRQHandler）用于处理其他传输时间。ST 的 USB 数据处理如下图所示：



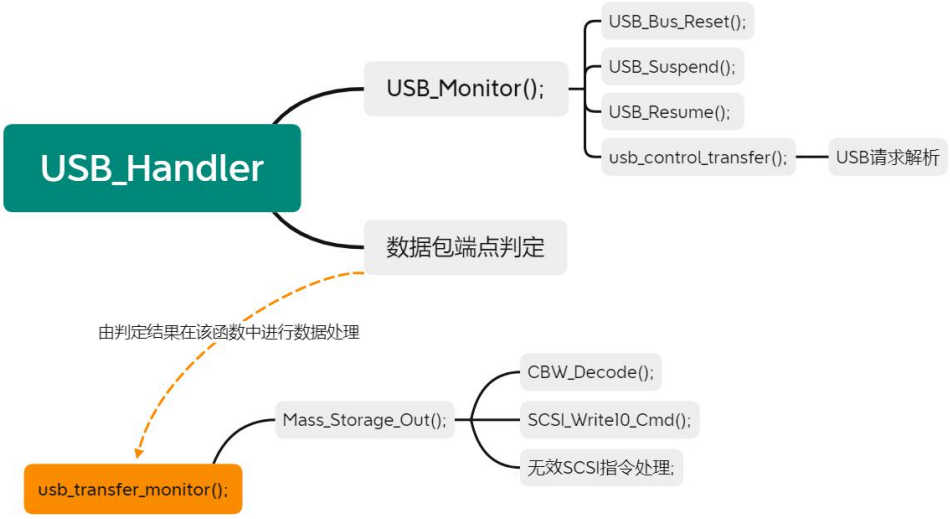
由于 USBFS 协议的限制，一包数据中最多可携带 64 字节数据，因此，当存在大量数据需要进行传输（IN 或 OUT 包）时，需要分批次进行传输。在 ST 的代码中，通过变量“Bot_State”来进行控制，以 Read10 指令为例，其读数据流程可如下图所示：



需要注意的是，Read10 指令解析完成之后（即上图左侧流程图）则进入数据传输阶段，此阶段是通过多次进入 USB 高优先级中断中，调用 Read_Memory()；来实现的。Read_Memory()；函数内每次传输 64 字节数据。

2. ACM32F403 代码移植要点

本文基于上海航芯官方 USB 例程进行移植，移植后的程序结构如下图所示：



ACM32F403 的 USB 是采用一个中断来进行数据处理。在官方例程中，USB 的中断函数内判定接收数据类型，包括 suspend，resume，reset，EP0_pack 以及其他端点的接收数据。判定结束后，会调用 USB_Monitor();函数来处理 suspend，resume，reset 以及 EP0_pack 数据。而其他端点数据会在 usb_transfer_monitor();函数中进行解析，该函数由客户调用，一般在主函数的死循环中进行处理。在本文的移植中，主要需对 USB 的端点数据进行处理。

A. EP0_Pack

EP0接收的setup数据会被存放在SETIP_0_3_DATA和SETIP_4_7_DATA寄存器中 ，数据结构如下所示：

```
dev_req.bmRequestType=USBCTRL->SETIP_0_3_DATA &0xff;
dev_req.bRequest=(USBCTRL->SETIP_0_3_DATA>>8)&0xff;
dev_req.wValue=(USBCTRL->SETIP_0_3_DATA>>16)&0xffff;
dev_req.wIndex    = USBCTRL->SETIP_4_7_DATA&0xffff;
dev_req.wLength=(USBCTRL->SETIP_4_7_DATA>>16)&0xffff;
```

该部分解析，可由用户在函数 void usb_control_transfer(void)中添加需要的处理函数。该函数由航芯官方例程里提供。在做 U Disk 程序移植时，需添加 GetMaxLun 和 Storage_Reset 处理函数，如下图所示：

```
else if(type == USB_CLASS_REQUEST)       //类请求
{
    printfS("class request-->");

    if(dev_req.bRequest == 0xFE) //GetMaxLun
    {
        Get_MAX_LUN();
    }
    else if(dev_req.bRequest == 0xFF) //reset
    {
        Mass_Storage_Reset();
    }

}
```

B. EP1_Pack

在本文所述的代码中，ACM32F403 采用 EP1 完成数据的收发工作。主要是完成对 SCSI 协议的解析工作。移植过程中，需要文件 mass_mal.c、memory.c、scsi_data.c、usb_scsi.c、usb_bot.c 及其头文件。本段主要就上述文件中代码需要改动的地方进行说

明，部分参数需要重新定义，读者可自行解决。下表列出了 ST 和 Aisino 的 USB 收发功能函数，该部分移植时需要修改的主要部分：

	ST	Aisino
USB 底层发送函数	uint32_t USB_SIL_Write(uint8_t bEpAddr, uint8_t* pBufferPointer, uint32_t wBufferSize)	uint8_t HAL_FSUSB_Send_Data(uint8_t *buffer,uint32_t length,uint8_t ep_index)
USB 底层接收函数	uint32_t USB_SIL_Read(uint8_t bEpAddr, uint8_t* pBufferPointer)	uint32_t HAL_FSUSB_Receive_Data(uint8_t *buffer,uint32_t length,uint8_t ep_index, uint8_t single_packet)

a. void Mass_Storage_In (void)

在 ST 的工程代码中该部分主要用于处理 SCSI 的读指令。由于全速 USB 一包数据最大支持 64 字节 ,因此 ,当需要传输的数据个数大于该数值时 ,则需要分包传输。在使用 ACM32F403 时，可直接传送需要的数据长度，内部会进行分包处理，因此，该函数可省略。

b. void Mass_Storage_Out (void)

该函数用于处理 SCSI 指令解析以及发送指令，需在 usb_transfer_monitor()中调用，并将函数内部的接收数据部分更改为：

Data_Len = HAL_FSUSB_Receive_Data(Bulk_Data_Buff, 64, out_ep_index, 1);”

c.void Transfer_Data_Request(uint8_t* Data_Pointer, uint16_t Data_Len)

将 USB 发送函数更改为 ACM32F403 对应的发送函数。在 ST 的工程中，该函数用于传输完数据后，进入 BOT_DATA_IN_LAST 状态，并在下一次的 Mass_Storage_In()函数调用时，回复 CSW 指令。而本文的移植代码中，省略了 Mass_Storage_In()函数，因此，可在该函数的尾部增加 CSW 发送指令：

Set_CSW (CSW_CMD_PASSED, SEND_CSW_ENABLE);

d.void Set_CSW (uint8_t CSW_Status, uint8_t Send_Permission)

将 USB 发送函数更改为 ACM32F403 对应的发送函数。

e.void Bot_Abort(uint8_t Direction)

该函数主要对收发端点的 STALL 状态进行处理，在 ACM32F403 的收发库函数中，对端点的 STALL 已做出相应控制，因此，该函数可省略。

f.void Read_Memory(uint8_t lun, uint32_t Memory_Offset, uint32_t Transfer_Length)

Read_Memory 函数用于收到 PC 端的 IN 包请求后将存储器中的数据读取并发送至 PC 端。而 ACM32F403 的 USB 发送库函数中，自行进行分包操作(一包最大数据为 64 字节)，因此在数据缓冲区容量允许条件下，可直接发送完毕，该函数修改如下：

```
{
    uint32_t Offset, Length;
```

```
    Offset = Memory_Offset * Mass_Block_Size[lun];
    Length = Transfer_Length * Mass_Block_Size[lun];
    CSW.dDataResidue = CBW.dDataLength;
    while(Transfer_Length --)
    {
            MAL_Read(lun ,
                                Offset ,
                                Data_Buffer,
                                Mass_Block_Size[lun]);
            Length = min(Mass_Block_Size[lun], CSW.dDataResidue);
            Offset += Mass_Block_Size[lun];
            HAL_FSUSB_Send_Data((uint8_t *)(Data_Buffer), Length, in_ep_index);
            CSW.dDataResidue -= Length;
    }
Set_CSW (CSW_CMD_PASSED, SEND_CSW_ENABLE);
}
g.void Write\_Memory (uint8\_t lun, uint32\_t Memory\_Offset, uint32\_t
Transfer\_Length)
```

写数据指令完成后，将 Bot_State 值更改为 BOT_IDLE。ST 的工程代码中，变量“Bot_State”收发状态机的状态值，其值如下表所示：

BOT_IDLE	空闲态，该状态下接收到数据即进入 SCSI 指令解析
BOT_DATA_OUT	OUT 包接收态，该状态下循环接收数据，直到 PC 端请求的数据接收完毕
BOT_DATA_IN	IN 包发送态，该状态下循环发送数据，直到 PC 端请求的数据发送完毕
BOT_DATA_IN_LAST	IN 包结束态，该状态下需发送 CSW 应答指令
BOT_CSW_Send	CSW 应答指令发送
BOT_ERROR	错误态

而基于 ACM32F403 的 U Disk 工程，IN 包可由函数 HAL_FSUSB_Send_Data（ ）在其内部进行分包处理，不需要额外逻辑，因此，移植后 Bot_State 仅需要在 BOT_IDLE、BOT_DATA_OUT、BOT_ERROR 之间转换，其他对 Bot_State 的控制可省略。