AN2554

使用MPLAB® Harmony USB设备协议栈创建多LUN USB海量存储类设备

简介

通用串行总线（Universal Serial Bus，USB）协议广泛用于将存储设备连接到USB主机计算机。此类设备使用一组称为USB海量存储类（Mass Storage Class，MSC）的标准。任何允许使用海量存储类协议访问其内部存储空间的设备都可以作为海量存储设备（Mass Storage Device，MSD）通过USB接口连接到主机计算机。

本应用笔记介绍如何使用MPLAB® Harmony USB设备协议栈框架创建一个支持多个逻辑单元（多LUN）的应用程序，其中每个逻辑单元作为单独的驱动器显示在USB主机计算机上。多插槽USB读卡器应用中可以找到多LUN应用程序的典型用例。

本应用笔记首先提供有关MSD特定USB描述符、请求和传输协议的信息，随后概述MPLAB Harmony USB设备架构，然后详细介绍如何使用MPLAB Harmony配置器（MPLAB Harmony Configurator，MHC）逐步创建多LUN应用程序，并介绍重要的数据结构、函数和状态机。

控制传输

USB 2.0协议具有四种数据传输类型：控制、批量、中断和同步。这些数据传输类型各具特性，因此适用于不同类型的应用。MSD通常使用控制和批量传输。有关批量传输类型的详细信息，请参见 **“批量传输”** 部分。

在USB协议中，USB主机通常使用控制传输向设备发送标准请求和类特定请求。控制传输通常在默认端点（端点0）上发送。此设备端点是双向的，始终处于使能状态。对于全速设备，控制端点的大小（单个事务中可传输的最大字节数）可以是8字节、16字节、32字节或64字节。对于高速设备，大小固定为64字节。与其他USB传输不同，控制传输由多个阶段组成：建立阶段、可选数据阶段和状态阶段。

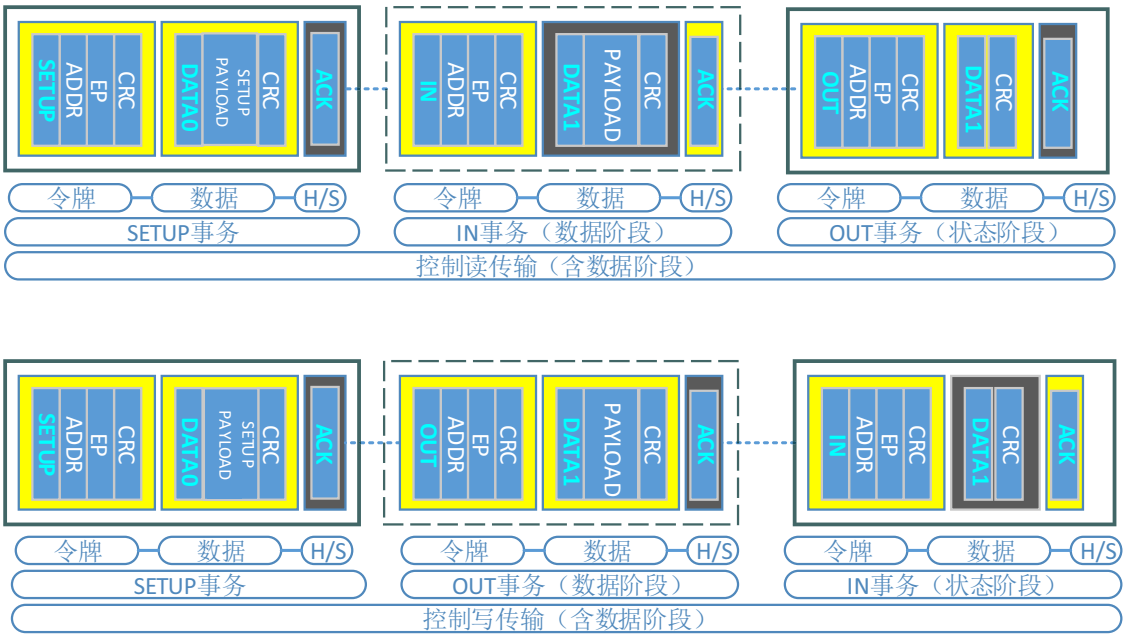
建立阶段包括一个令牌包，然后是一个数据包和一个握手包。令牌包包含一个SETUP令牌。数据包包含一个DATA0令牌和一个8字节的命令，其格式由USB规范定义。此命令包含以下内容的详细信息：请求类型（标准、类特定或供应商）、请求的接收方（设备、接口或端点）、请求代码、接口或端点索引、控制传输的数据阶段的方向以及要传输的字节数（该值可以是0），前提是有数据阶段。与建立命令相关的数据在控制传输的数据阶段中传输。在读控制传输中，数据从设备传输到主机。在写控制传输中，数据从主机传输到设备。

控制传输的可选数据阶段包括三个包，即一个令牌包，然后是一个数据包和一个握手包。用于读传输的令牌包包含一个IN令牌。用于写传输的令牌包包含一个OUT令牌。数据包以DATA1令牌开始，然后在数据阶段中为每个事务交替使用DATA0和DATA1令牌。设备可以向主机发送ACK令牌（表明它已经成功处理事务），或者发送NAK令牌（表明端点繁忙），还可以发送STALL令牌（表明已发生错误）。

控制传输的状态阶段始终与数据传输的方向相反。该阶段包括三个包，即一个令牌包，然后是一个数据包和一个握手包。对于控制读传输，主机通过发送OUT令牌包（零长度数据包（ZLP））来提供状态，而设备将通过握手包中的ACK令牌进行回复。对于控制写传输，主机通过令牌包中的IN令牌请求获取设备的状态。设备通过在数据包中发送DATA1令牌ZLP来做出响应，而主机通过握手包中的ACK令牌进行回复。

图1给出了控制读写传输的不同阶段，还给出了传输中涉及的每个数据包的字段。有关这些字段的信息，请参见www.usb.org上的USB 2.0规范。

图1：　　控制读写传输阶段



建立有效负载——固定8字节格式

批量传输

批量传输通常用于传输大量数据。这种传输类型保证了数据的完整性，但不能保证延时。USB 协议 *不*为批量传输分配带宽。USB 主机在有可用带宽时安排批量传输，通常是在帧中的所有其他传输（控制、中断和同步）均完成之后。这使得批量传输的完成时间不可预测。

批量事务在配置为构建传输的端点上发生。批量端点是单向的。USB 设备可以从主机的批量输出端点上接收数据，并将数据传输到主机的批量输入端点上。对于全速设备，批量端点的大小可以是8字节、16字节、32字节或64字节。对于高速设备，大小固定为512字节。

图2给出了批量输入（读）和批量输出（写）事务中涉及的数据包。

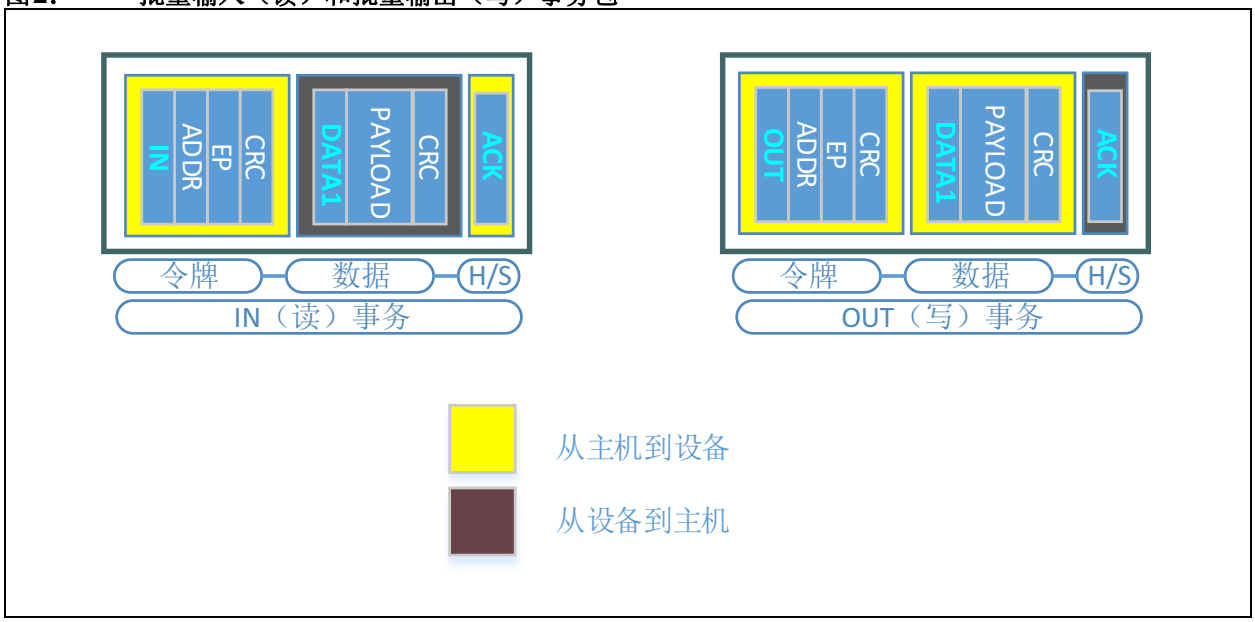
批量读传输（主机从设备读取）可以包含多个IN事务，每个事务包含以下交互：

- 主机向设备发送一个IN请求（令牌包）
- 如果设备准备好将数据发送到主机，则会使用批量数据响应主机（数据包）。如果设备没有要发送的数据或尚未准备好，则将发送 NAK。如果端点发生错误，设备将发送STALL。
- 主机向设备发送ACK（握手包）。

批量写传输（主机向设备写入）可以包含多个OUT事务，每个事务包含以下交互：

- 主机向设备发送OUT请求（令牌包）
- 主机随后向设备发送批量数据。（数据包）
- 设备向主机发送ACK，表示操作成功（握手包）。如果端点缓冲区不为空，则设备将发送NAK。如果端点发生错误，则将发送STALL。

图2：　　批量输入（读）和批量输出（写）事务包



USB海量存储设备类描述符

USB设备使用标准USB描述符将其功能特性通告给USB主机。本节讨论与USB MSD相关的USB描述符。

图3给出了典型USB海量存储类设备的逻辑块。

要保持高水平，USB MSD设备固件必须执行以下任务：

1. 检测并响应端点0上的控制请求。
2. 检测并响应控制端点0上的海量存储类特定请求
3. 解码并响应批量端点上的（块设备/SCSI）命令，并且必须访问介质并响应批量端点上的读/写（SCSI）请求。

图4给出了USB海量存储设备的描述符树。USB海量存储类设备需要设备描述符、配置描述符、接口描述符和两个端点（用于仅批量传输协议）描述符。下表提供了其中每个描述符的详细信息。有关标准USB描述符和USB设备描述符拓扑的更多信息，请参见“USB 2.0规范”的第9章。

图3： USB海量存储类设备框图

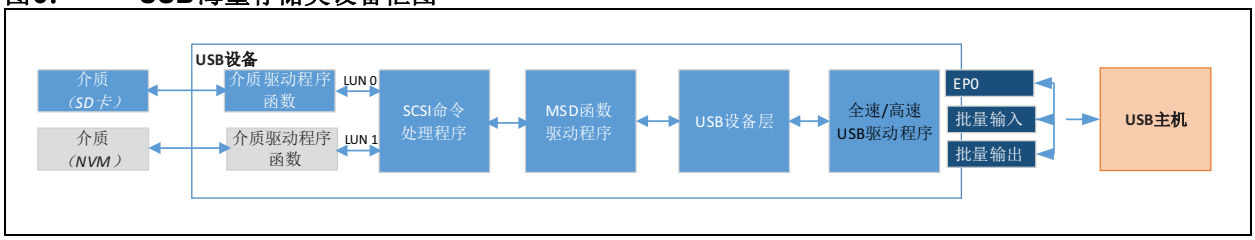


图4： USB海量存储类设备描述符树

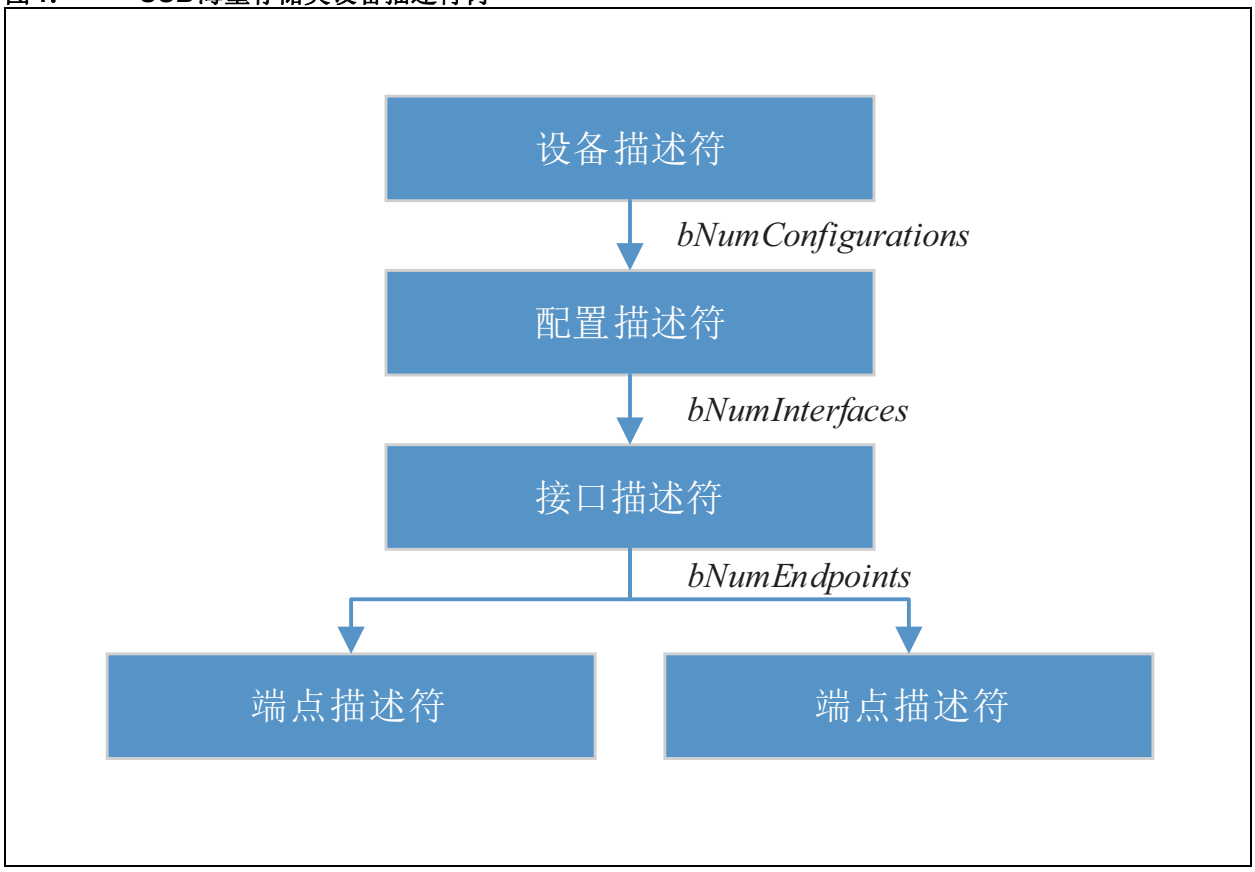


表1提供了本应用中实现的MSD的设备描述符的详细信息。请注意，*bDeviceClass*、*bDeviceSubClass*和*bDeviceProtocol*字段均设置为0x00，这些将由接口描述符定义。

表1： 设备描述符

偏移量	字段	大小（字节）	值（十六进制）	说明
0	*bLength*	1	0x12	设备描述符的大小（字节）。
1	*bDescriptorType*	1	0x01	设备描述符。
2	*bcdUSB*	2	0x0200	支持USB 2.0规范。
4	*bDeviceClass*	1	0x00	类在接口描述符中指定
5	*bDeviceSubClass*	1	0x00	子类在接口描述符中指定
6	*bDeviceProtocol*	1	0x00	协议在接口描述符中指定
7	*bMaxPacketSize0*	1	0x40	端口0的最大数据包大小为64字节。
8	*idVendor*	2	0x04D8	供应商ID（例如，Microchip供应商ID = 04D8h）。
10	*idPacket*	2	0x009	产品ID。
12	*bcdDevice*	2	0x0100	设备版本号（例如01.00）。
14	*iManufacturer*	1	0x01	字符串描述符中制造商字符串的索引。
15	*iProduct*	1	0x02	字符串描述符中产品字符串的索引。
16	*iSerialNumber*	1	0x03	字符串描述符中设备序列号的索引。
17	*bNumConfigurations*	1	0x01	配置数设置为1。

表2提供了本应用中实现的MSD的配置描述符的详细信息。此描述符的*bNumInterfaces*字段设置为0x01。这表明配置只有一个接口。

当设备收到“获取配置描述符”请求时，它会将配置描述符、接口描述符和端点描述符返回给USB主机。枚举期间，USB主机将向设备发出“设置配置”请求，从而要求设备将此配置设置为活动状态。

表2： 配置描述符

偏移量	字段	大小（字节）	值（十六进制）	说明
0	*bLength*	1	0x09	配置描述符的大小（字节）。
1	*bDescriptorType*	1	0x02	配置描述符。
2	*wTotalLength*	2	0x0020	为此配置返回的数据的总长度为32字节。包括配置描述符（9）+接口描述符（9）+端点描述符（7+7）。
4	*bNumInterfaces*	1	0x01	此配置中的接口数（支持仅批量数据接口）。
5	*bConfigurationValue*	1	0x01	用于选择此配置的`SetConfiguration`函数请求的参数值。
6	*iConfiguration*	1	0x00	描述此配置的字符串描述符的索引。
7	*bmAttributes*	1	0xC0	自供电。
8	*MaxPower*	1	0x32	此设备完全正常工作并且选择此配置时，将从总线消耗最多100 mA（2x *MaxPower*）电流。

表3提供了本应用中实现的MSD的接口描述符的详细信息。*bNumEndpoints*的值为0x02。这表明设备包含两个端点。

bInterfaceClass 字段设置为 0x08，对应于海量存储类。这将指示主机将海量存储客户端驱动程序与此设备相关联。

*bInterfaceSubClass*字段标识可以由USB海量存储类传输的行业标准命令集。

表3： 接口描述符

偏移量	字段	大小（字节）	值（十六进制）	说明
0	*bLength*	1	0x09	配置描述符的大小（字节）。
1	*bDescriptorType*	1	0x04	接口描述符。
2	*bInterfaceNumber*	1	0x00	接口编号。从0开始的值，标识此配置支持的并发接口阵列中的索引。
3	*bAlternateSetting*	1	0x00	此接口无备用设置。
4	*bNumEndpoints*	1	0x02	此接口使用的端点数量（批量输入和批量输出）。
5	*bInterfaceClass*	1	0x08	USB海量存储类代码。
6	*bInterfaceSubClass*	1	0x06	使用的数据传输协议是SCSI透明命令集。
7	*bInterfaceProtocol*	1	0x50	支持仅批量传输协议。
8	*iInterface*	1	0x00	此接口无字符串描述符。

表4列出了*bInterfaceSubClass*字段的一些可能值。

表4： *bInterfaceSubClass*的可能值

子类代码	命令块规范	使用方：
01	精简块命令（Reduced Block Command，RBC）	闪存设备
02	MMC-5（ATAPI）	CD和DVD设备
03	QIC-157（过时）	磁带设备
04	UFI	软盘驱动器
05	SFF-8070i（过时）	软盘驱动器
06	SCSI透明命令集	任何设备

大多数设备都推荐使用小型计算机系统接口（Small Computer Systems Interface，SCSI）透明命令集。SCSI透明命令集包括所有SCSI相关规范，其中包括SCSI主命令（SCSI Primary Command，SPC）和SCSI块命令（SCSI Block Command，SBC）。SCSI标准是在USB规范之外定义的。在枚举过程中，USB主机将识别设备支持的命令协议。如果设备支持SCSI透明命令集，则USB主机将在USB传输中发出SCSI命令。SCSI命令允许USB主机读写存储介质中的数据块、请求获取状态信息以及控制设备操作。MPLAB Harmony USB设备协议栈海量存储类函数驱动程序仅支持SCSI透明命令集。

接口描述符中的*bInterfaceProtocol*字段定义了海量存储接口使用的传输协议。它指定在主机和设备之间传输海量存储命令、数据和状态信息时，总线上要使用的USB传输类型。海量存储类定义了两个传输协议：

- 控制/批量/中断（Control/Bulk/Interrupt，CBI）
- 仅批量传输（Bulk-Only Transport，BOT）

USB MSC规范批准CBI传输协议仅用于全速软盘驱动器。此外，CBI完全由仅批量协议取代，USB MSC规范不鼓励在新设计中使用CBI。MPLAB Harmony USB设备协议栈海量存储函数驱动程序仅支持BOT协议。此协议使用批量传输来传输海量存储类操作的命令、数据和状态。本应用笔记仅涉及BOT协议。

在枚举期间，*bInterfaceProtocol*字段将向主机指示设备支持BOT协议，因此主机将使用批量端点来传输BOT的命令、数据和状态阶段。

批量输入端点描述符

批量输入端点用于将数据和状态从设备传输到主机。

表5提供了本应用中实现的MSD的批量输入端点描述符的详细信息。*wMaxPacketSize*字段设置为512字节。

表5：　批量输入端点描述符

偏移量	字段	大小（字节）	值（十六进制）	说明
0	*bLength*	1	0x07	配置描述符的大小（字节）。
1	*bDescriptorType*	1	0x05	端点描述符。
2	*bEndpointAddress*	1	0x81	端点1，输入方向。
3	*bmAttributes*	1	0x02	此端点支持批量传输。
4	*wMaxPacketSize*	2	0x0200	对于高速设备，此端点的最大大小为512字节；对于全速设备，此端点的最大大小为64字节。
6	*bInterval*	1	0x00	不适用于批量端点。将其设置为0。

批量输出端点描述符

批量输出端点用于接收从主机传输到设备的命令和数据。

表6提供了本应用中实现的MSD的批量输出端点描述符的详细信息。*wMaxPacketSize*字段设置为512字节。

表6： 批量输出端点描述符

偏移量	字段	大小（字节）	值（十六进制）	说明
0	*bLength*	1	0x07	配置描述符的大小（字节）。
1	*bDescriptorType*	1	0x05	端点描述符。
2	*bEndpointAddress*	1	0x01	端点1，输出方向。
3	*bmAttributes*	1	0x02	此端点支持批量传输。
4	*wMaxPacketSize*	2	0x0200	对于高速设备，此端点的最大大小为512字节；对于全速设备，此端点的最大大小为64字节。
6	*bInterval*	1	0x00	不适用于批量端点，将其设置为0。

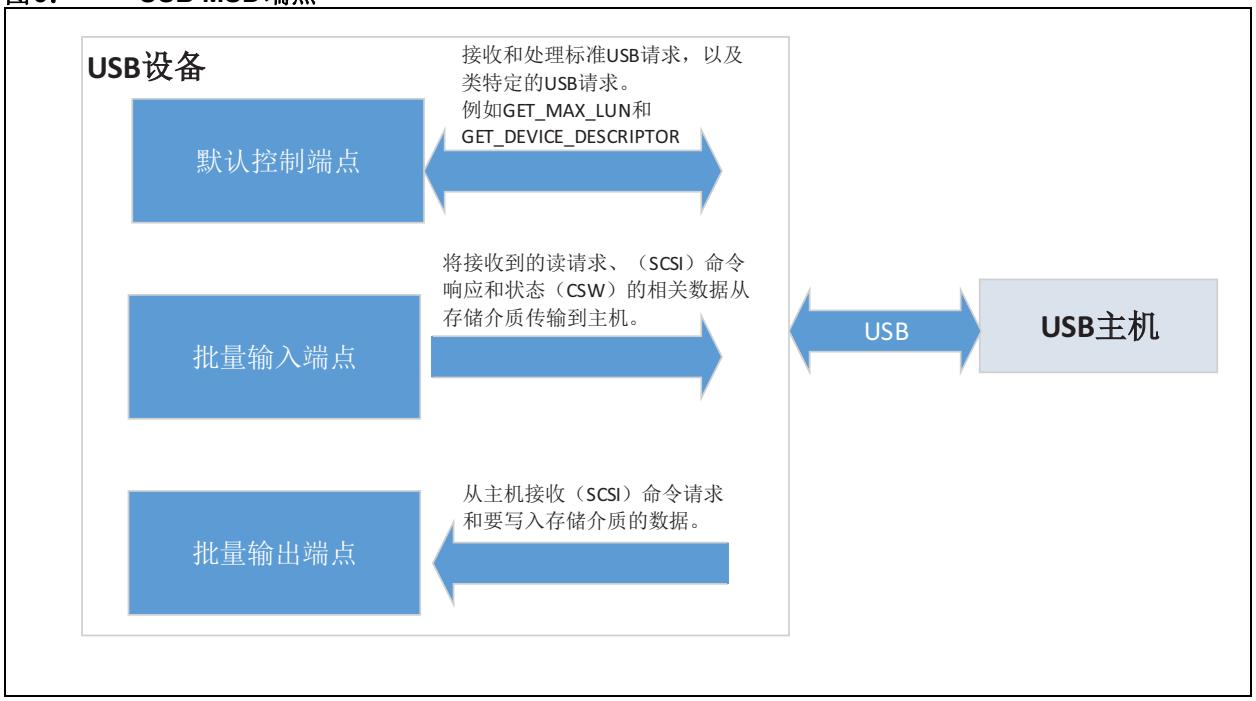
总的来说，USB海量存储类设备（MSD）由1个控制端点和2个批量端点组成。

- 控制端点（端点0）
- 批量输入端点
- 批量输出端点

如图5所示，双向控制端点用于枚举设备并响应标准和类特定（本例中为海量存储类）的USB请求。此端点通常是端点0，始终处于使能状态，并且不需要描述符。

批量输出端点接收从主机写入设备的（SCSI）命令和介质数据。设备通过批量输入端点向主机发送（SCSI）命令回复、（SCSI）命令状态和请求的介质数据。

图5： USB MSD端点



USB海量存储类特定的请求

本部分介绍海量存储类特定的各种控制请求。USB海量存储类仅具有两个类特定的控制请求：仅批量海量存储复位（Bulk-only Mass Storage Reset，BOMSR）和获取最大LUN。

仅批量海量存储复位（BOMSR）

此请求用于复位海量存储设备及其关联的接口。此命令完成后，设备便可接收新的命令。尽管有复位命令，但是不应改变批量数据切换位和端点上的STALL条件。BOMSR这一类特定控制请求由主机在控制端点0上发出。

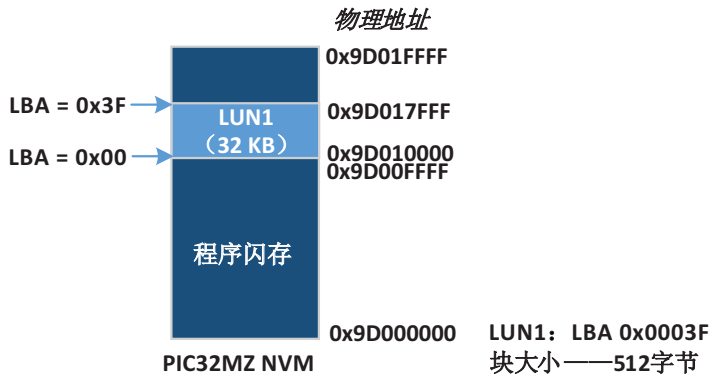
获取最大LUN

逻辑单元是由USB设备提供给USB主机的连续且可单独寻址的单元。逻辑单元编号（Logical Unit Number，LUN）用于标识逻辑单元，也就是可通过SCSI协议寻址的设备。一台设备可以实现多个逻辑单元，其编号最小为0，最大为15。USB主机发送“获取最大LUN”请求以获悉设备中包含多少个逻辑单元。设备应报告一个包含设备支持的最大LUN的数据字节。如果设备支持4个LUN，则应返回值3，LUN编号应为0至3。没有逻辑单元的设备应报告值0。不支持多个LUN的设备可能会停止此请求。“获取最大LUN”这一类特定请求由主机在控制端点0上发出。

逻辑块地址（Logical Block Address，LBA）为0至逻辑单元的大小。每个逻辑单元的起始LBA是0，结束地址取决于每个逻辑单元的大小。对于每个LUN，USB主机将查询LUN的容量以获悉逻辑单元的结束地址。通过LUN和LBA的组合，主机和设备可以识别所寻址的存储单元。

图6所示为此应用中配置为第二个逻辑单元（SD卡配置为第一个逻辑单元）的32 KB内部非易失性存储器（Non-Volatile Memory，NVM）。典型USB MSD主机以扇区为单位访问存储器。一个扇区的大小通常为512字节。因此，尽管PIC32MZ NVM的内部块大小（行大小）为2048字节，但是会以512字节（64个块 × 512字节/块 = 32 KB）的逻辑块（扇区）的形式提供给主机。主机通过向设备传送LUN和LBA以及要读写的连续扇区数（扇区 = 512字节）来寻址存储器。设备必须将LUN和LBA转换为实际的物理地址。

图6： NVM配置



USB海量存储仅批量传输（BOT）

MSC BOT协议传输设备所支持的命令集。在这种情况下，BOT协议会在命令通过USB传输到设备时将其打包。对于外部硬盘或U盘，这些命令可实现SCSI协议。

BOT协议将数据传输分为三个阶段：

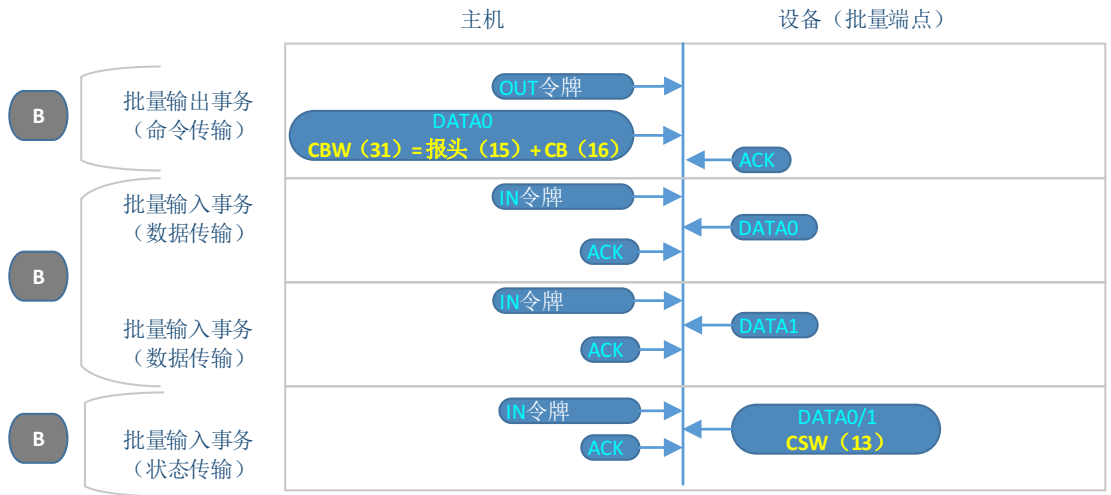
- 命令传输（OUT事务）
- 数据传输（IN事务或OUT事务）
- 状态传输（IN事务）

图7给出了从主机批量读取时涉及的事务。要从设备中读取数据，USB主机将在OUT事务中发出一个读命令，之后是一个或多个IN事务。数据传输以USB主机在IN事务中请求获取设备的状态结束。

用于通过USB传输（SCSI）命令的包装器是命令块包（Command Block Wrapper，CBW）。（SCSI）命令的状态在命令状态包（Command Status Wrapper，CSW）中传输。

每次传输均以主机向设备的批量输出端点发送CBW中所包含的一个（SCSI）命令开始。是否有数据阶段取决于命令。如果有数据阶段，设备将从USB主机的批量输出端点上接收数据，并将数据传输到USB主机的批量输入端点上。数据阶段完成后（或者如果没有数据阶段），USB主机将请求获取设备的状态。设备将向主机的批量输入端点返回CSW中包含的状态，这标志着传输的结束。

图7：　　批量读取事务



从主机批量读取的操作中涉及的事务

bInterfaceClass = 0x05——命令、数据和状态采用批量传输（BBB）。

bInterfaceSubClass = 0x06——（SCSI）命令以CBW和CSW中的状态形式传输。

命令块包（CBW）

表7提供了命令块包（CBW）的结构。CBW包含一个将其标识为CBW包的32位签名、一个将CSW与其相应的CBW相关联的标签、在数据阶段传输的字节数、数据阶段的方向、要接收命令块的LUN、命令块的长度和命令块（即SCSI命令有效负载）本身。

表7： CBW结构

字节	Bit							
	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11（0x08-0x0B）	*dCBWDataTransferLength*							
12（0x0C）	*bmCBWFlags*							
13（0x0D）	保留（0）				*bCBWLUN*			
14（0x0E）	保留（0）				*bCBWCBLength*			
15-30（0x0F-0x1E）	*CBWCB*							

dCBWSignature：

包含值43425355。这将该数据包标识为CBW。

dCBWTag：

由主机发送的32位命令块变量，用于将CSW与其对应的CBW相关联。设备将回显CSW的*dCSWTag*字段中的变量。

dCBWDataTransferLength：

表示主机希望发送或接收的数据字节数（基于*bmCBWFlags*中的方向位）。如果设置为0，设备将忽略*bmCBWFlags*中的方向位，CBW和相关的CSW之间将没有数据传输。

bmCBWFlags：

Bit 7：

1 = 从设备到主机的数据输入（主机要进行读操作）
0 = 从主机到设备的数据输出（主机要进行写操作）

注：	如果*dCBWDataTranferLength*设置为0，设备将忽略此位。

Bit 6：过时——主机将其设置为0。

Bit 5-0：保留——主机将其设置为0。

bCBWLUN：

要接收此命令块的设备的逻辑单元编号（LUN）。USB主机将通过发出类特定请求“获取最大LUN”来获取设备支持的LUN数。对于支持多个LUN的设备，主机应在该字段中放置此命令块所寻址的LUN。如果设备只有一个LUN，则主机会将该字段设置为0。

bCBWCBLength：

该字段定义了命令块（CBWCB）的有效长度。有效值可以是1至16。

CBWCB：

该字段包含了要由设备执行的命令块。设备会将该字段中的第一个*bCBWCBLength*字节解析为接口描述符的*bInterfaceSubClass*字段所标识的命令集定义的命令块。对于海量存储设备类，命令块将包含其中一个SCSI命令。

命令状态包（CSW）

MSD通过命令状态包（CSW）将命令的状态发送到主机。表8列出了CSW的结构。

表8： CSW结构

字节	Bit							
	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11（0x08-0x0B）	*dCBWDataResidue*							
12（0x0C）	*bmCBWStatus*							

dCSWSignature：

包含用于将数据包标识为CSW的值53425355h。

dCSWTag：

设备会将该字段设置为相关CBW的dCBWTag字段中接收的值。

dCSWDataResidue：

对于数据输出，设备会将此值设置为CBW中接收的*dCBWDataTransferLength*与设备处理的实际数据量之差。对于数据输入，设备会将此值设置为CBW中接收的*dCBWDataTransferLength*与设备发送的实际数据之差。

bCSWStatus：

指示（SCSI）命令的状态。下表列出了*bCSWStatus*的可能值。

表9： SCSI命令值

值（十六进制）	说明
0x00	命令通过。
0x01	命令失败。
0x02	阶段错误。主机将执行复位恢复。
0x03-0x04	过时（保留）。
0x05-0xFF	保留。

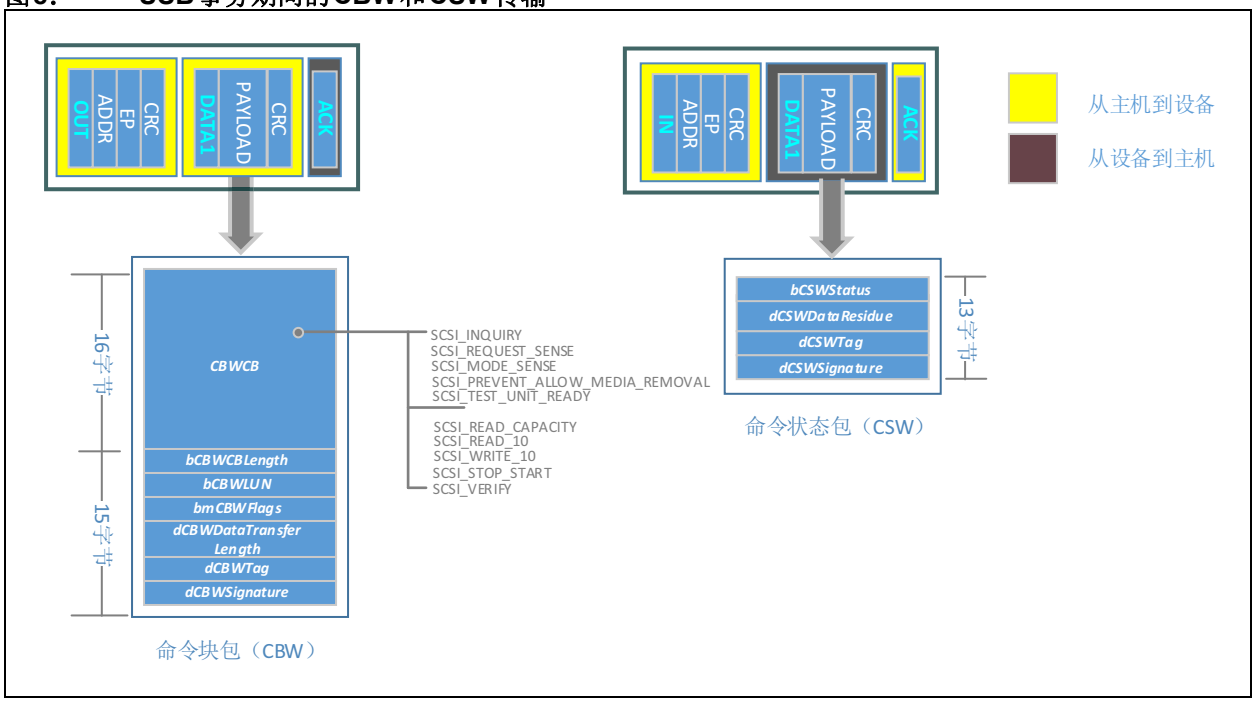
阶段错误表示设备遇到内部错误，除了复位之外，没有可靠的恢复方法。

接收到阶段错误后，USB主机应通过在控制端点上按顺序发出以下命令来执行设备复位恢复。

1. 仅批量海量存储复位：
   这是一个类特定的接口请求。此请求完成后，设备便可接收新的CBW。复位不应改变数据切换位和端点停止条件。
2. 至批量输入端点的清除特性HALT：
   这是一个标准端点请求。设备会将端点的数据切换复位为DATA0。如果可能，端点应恢复正常的通信。
3. 至批量输出端点的清除特性HALT：
   这是一个标准端点请求。设备会将端点的数据切换复位为DATA0。如果可能，端点应恢复正常的通信。

图8所示为USB事务期间的CBW和CSW传输。

图8：　　USB事务期间的CBW和CSW传输



小型计算机系统接口（SCSI）协议

小型计算机系统接口（SCSI）是支持系统与存储设备通信的一系列协议。它为应用程序提供了适合不同类别设备的标准架构和标准化命令集，从而可抽象化底层介质并提供一种标准机制来访问不同类型的介质。SCSI提供了读取、写入和检查介质状态以及执行其他操作的方法。

典型MSD使用SCSI共享命令：SCSI主命令（SPC）和设备特定的SCSI块命令（SBC）。

SCSI主命令是所有设备类型的通用命令。SCSI块命令特定于块设备。这些命令适用于在标准SCSI INQUIRY响应的“外设类型”字段中将自身声明为直接访问块设备的逻辑单元。

表10列出了MPLAB Harmony MSD SCSI实现支持的SCSI命令。

表10： MPLAB HARMONY支持的SCSI命令

SCSI主命令（SPC）	SCSI块命令（SBC）
SCSI INQUIRY（必选）	SCSI READ CAPACITY（10）（必选）
SCSI REQUEST SENSE（必选）	SCSI READ（10）（必选）
SCSI MODE SENSE（可选）	SCSI WRITE（10）（对于可写设备是必选的）
SCSI TEST UNIT READY（必选）	—

SCSI主命令

SCSI INQUIRY（操作码0x12）

USB主机发出SCSI INQUIRY命令以获取有关设备的信息，例如外设类型、所支持SPC的版本、供应商和产品标识。

本应用笔记中实现的MSD发送36个字节以响应SCSI INQUIRY命令，如下表所示。

外设类型设置为0，表示有一个直接访问块设备连接到其逻辑单元。SBC标准进一步规定了直接访问块设备的规范。

RMB位设置为1，表示介质是可移除的。

“版本”设置为0x04，指示设备符合SPC-2版本的规范。

“供应商标识”指示由T10技术委员会指定的供应商标识代码。

有关更多信息，请参见SPC-2规范的**第7.3.2节“标准查询数据”**。

表11： SCSI INQUIRY命令

字节	Bit							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	外设限定符（0）			外设类型（0）				
1	RMB（1）	保留						
2	版本（0x04）							
3	AERC（0）	过时（0）	NormACA（0）	HiSup（0）	响应数据格式（0x02）			
4	附加长度（n-4）（0x1F）							
5	SCCS（0）	保留						
6	BQUE（0）	ENCSERV（0）	VS（0）	MULTIP（0）	MCHNGR（0）	过时（0）	过时（0）	ADDR16（0）
7	RELADR（0）	过时（0）	WBUS16（0）	SYNC（0）	LINKED（0）	过时（0）	CMDQUE（0）	VS（0）
8 15	供应商标识（“Microchp”）							
16 31	产品标识（“Mass Storage”）							
32 35	产品版本（“0001”）							

SCSI MODE SENSE（操作码0x1A）

USB主机发出MODE SENSE命令来读取设备特定的参数。设备按下表所示发送对MODE SENSE命令的响应。

表12： MODE SENSE命令

字节	Bit							
	7	6	5	4	3	2	1	0
0	模式数据长度（0x03）							
1	介质类型（0x00）							
2	设备特定的参数（见**注1**）							
3	块描述符长度（0x00）							

注 1： 这些字段的内容在SBC规范中进行了说明。

“介质类型”由SBC规范定义，设置为0x00。

“设备特定的参数”由SBC规范定义，如下表所示。

表13： 设备特定的参数

Bit	7	6	5	4	3	2	1	0
名称	WP			DPOFUA	保留			

WP位指示介质是否受写保护。在将此位置1前，设备会检查介质的写保护状态。

DPOFUA位设置为0，表示设备不支持缓存。

MPLAB Harmony USB设备协议栈通过将CSW中的*bCSWStatus*位设置为0x01使此命令失败，并在介质不存在时更新SENSE数据。所有其他依靠介质进行响应的SCSI命令均如此。

注： 有关更多信息，请参见SPC-2规范的**第8.3.3节**和SBC-3规范的**第6.3.1节**。

SCSI REQUEST SENSE（操作码0x03）

在收到前一个命令的命令失败状态后，主机可以发出一个REQUEST SENSE命令，以获悉关于该命令失败的更多信息。CSW中的*bCSWStatus*字段指示命令失败还是通过。此时，设备还将记录错误。

设备以检测数据结构响应SCSI REQUEST SENSE命令。尽管检测数据结构有多个字段，但只有以下三个字段比较重要：第一个字段是检测密钥，它表示最后一个命令的结果，如介质未就绪、命令请求非法和介质受保护等。其他两个字段，即附加检测代码（Additional Sense Code，ASC）和附加检测代码限定符（Additional Sense Code Qualifier，ASCQ），提供了对问题的准确描述，例如，介质不存在、命令操作码非法和受写保护。

SCSI TEST UNIT READY（操作码0x00H）

USB主机发出TEST UNIT READY命令，检查介质设备是否准备就绪。当检测数据指示介质未就绪时，主机也可以发出此命令。此命令没有数据阶段。如果设备未就绪，CSW中的*bCSWStatus*字段设置为0x01（命令失败）。主机可以重新尝试发送TEST UNIT READY命令，直到设备在CSW中的*bCSWStatus*字段中报告00h，这表明介质已准备就绪。

对于可插拔介质（如SD卡），如果介质驱动程序检测到介质存在并且介质已成功初始化，则认为介质处于就绪状态，可随时使用。

对于NVM（内部闪存）等不可移除的介质，则认为介质始终处于就绪状态。

SCSI块命令

SCSI READ CAPACITY（10）（操作码0x25）

主机发出READ CAPACITY命令，以获悉设备介质可以存储多少个字节。设备通过一个包含介质上最后一个块的LBA和数据阶段中块大小的结构进行响应，如下表所示。

表14： 块大小和块地址

字节	Bit							
	7	6	5	4	3	2	1	0
0	逻辑块地址							
3								
4	块大小（字节）							
7								

块大小为512字节时的理论限值为2 TB（0xFFFFFFFF x 512）。

大小为256 MB、块大小为512字节的逻辑单元将报告0x7FFFF的LBA和512字节的块大小。

大小为32768字节、内部块大小为2048字节的逻辑单元（本应用情况下为NVM存储器）将报告0x3F [(32768 / 512) - 1]的LBA和512字节的（逻辑）块大小。

SCSI READ（10）（操作码0x28）

USB主机发出SCSI READ命令，从设备介质读取数据。主机将在“逻辑块地址”字段中指定起始地址，在READ请求的“传输长度”字段中指定要读取的连续逻辑块的数量。LUN ID在CBW报头的*bCBWLUN*字段中指定。设备将从介质中读取请求的逻辑块，并在IN数据阶段以512字节的块形式将其发送到主机。

图9给出了USB读操作涉及的事务。

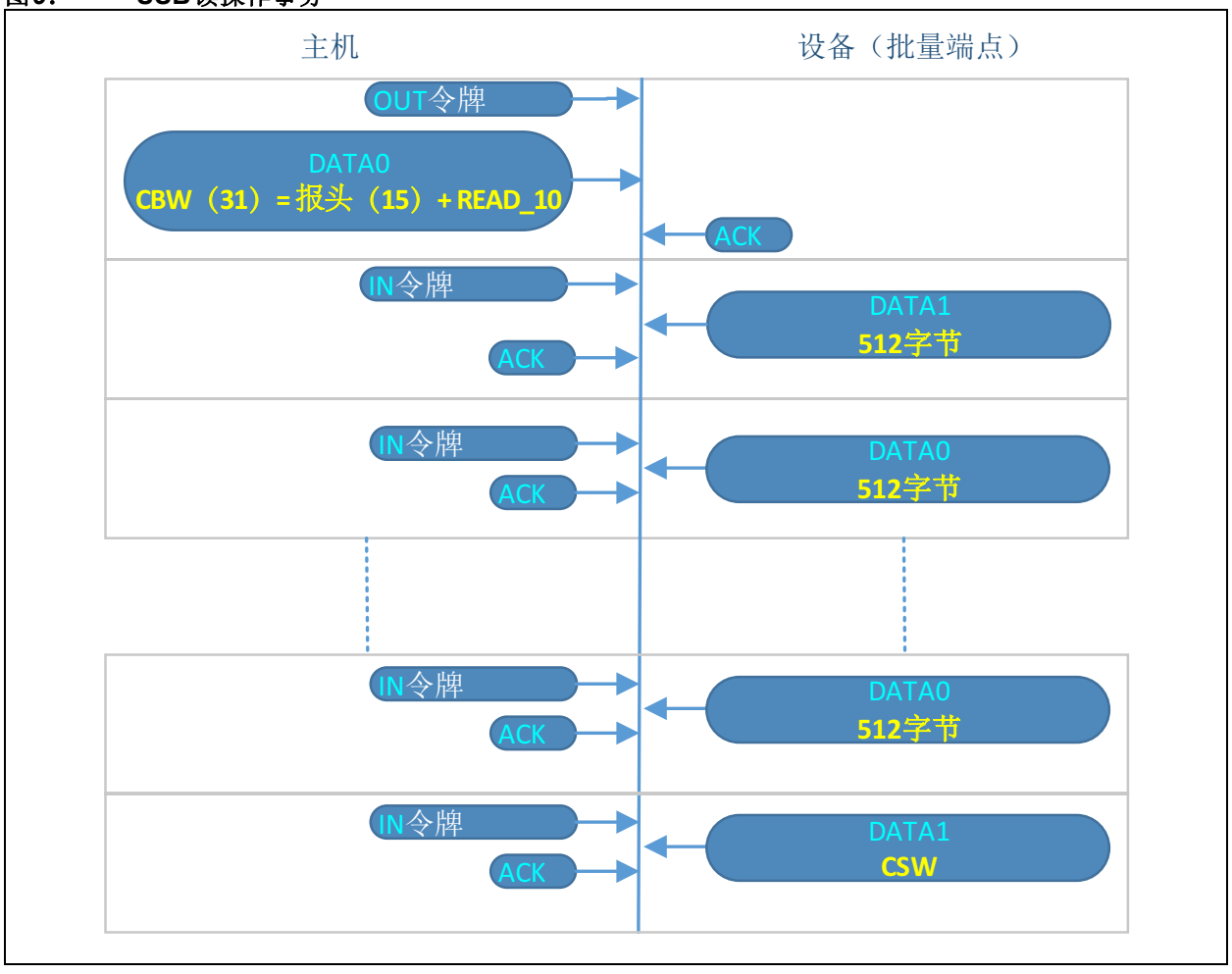
MPLAB Harmony USB设备协议栈允许缓冲从介质读取的数据。例如，如果用户已经配置了8个扇区的缓冲区大小（一个扇区的大小 = 512字节），则读取12个逻辑数据块（12个扇区或6 KB）的请求将仅对介质产生两个读取请求。在第一个读请求期间，将读取8个扇区（4 KB）的数据并保存在缓冲区中。剩余的4个扇区（2 KB）将在介质的下一个读请求中进行读取。

如果介质数据不可用，则设备将发送NAK来响应来自主机的IN数据请求，同时设备还将读取介质。主机随后将重新尝试IN数据事务。在使能缓冲的情况下，介质数据位于RAM缓冲区，因此主机的IN数据阶段可以在没有任何介质读取延时的情况下得到处理。这会导致主机重试次数减少，从而提高读取操作的整体吞吐量，但是会增加对RAM的使用。

主机将在12个IN数据事务期间接收所有12个逻辑数据块，每个IN事务向主机传送512个字节。

直到主机收到任何未完成CBW的CSW，USB仅批量传输规范才允许USB主机将CBW传输到设备。这允许USB海量存储设备协议栈支持的所有逻辑单元共用通用缓冲区。

图9：　　USB读操作事务



SCSI WRITE（10）（操作码0x2A）

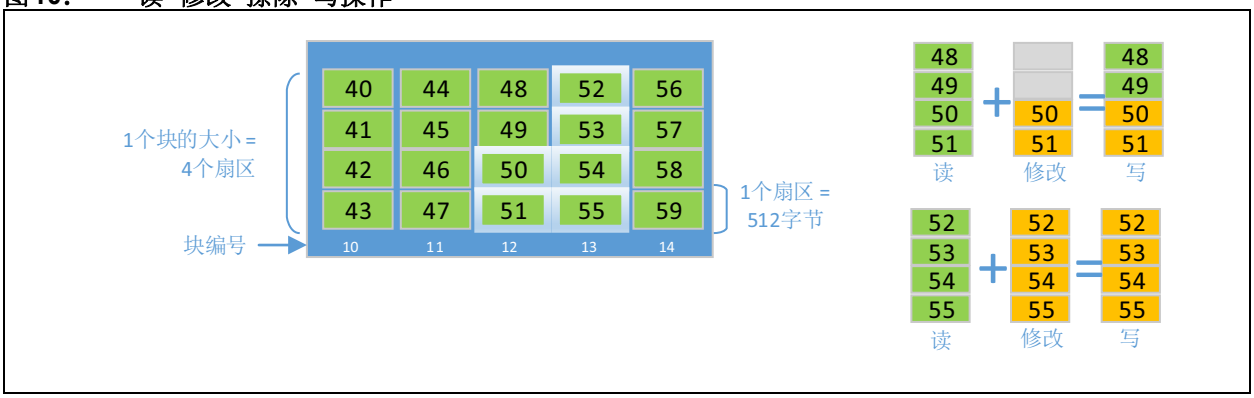
USB主机发出SCSI WRITE命令，向设备介质写入数据。USB主机将在“逻辑块地址”中指定起始地址，在WRITE请求的“传输长度”字段中指定要写入的连续逻辑块的数量。LUN ID在CBW报头的*bCBWLUN*字段中指定。

MPLAB Harmony设备协议栈对块大小大于1个扇区（即512个字节）的介质执行读-修改-擦除-写操作。对于本应用中实现的MSD示例，NVM介质的写入块（行）大小为4个扇区（即2048个字节）。图10所示的WRITE请求中，“逻辑块地址”设置为50，“传输长度”设置为6，设备将首先读取内部块编号12（扇区48-51），并用从主机接收的数据修改扇区50和51。修改后的整个块（扇区48-51）将写回介质。此后，将读取块13（扇区52-55）的内容，并用从主机接收的数据修改。修改后的整个块将写回介质。

主机将在6个OUT事务期间完成发送6个逻辑数据块，每个OUT事务向设备传送512个字节。只有在数据实际写入介质后，MPLAB Harmony设备协议栈才会向主机报告GOOD状态。

对于所有不支持的SCSI命令，USB设备会将CSW的*bCSWStatus*字段设置为01h（命令失败），并通过将“检测密钥”设置为0x05（请求非法）、将“附加检测代码”设置为0x20（命令操作码无效）来更新检测数据。

图10：　读-修改-擦除-写操作

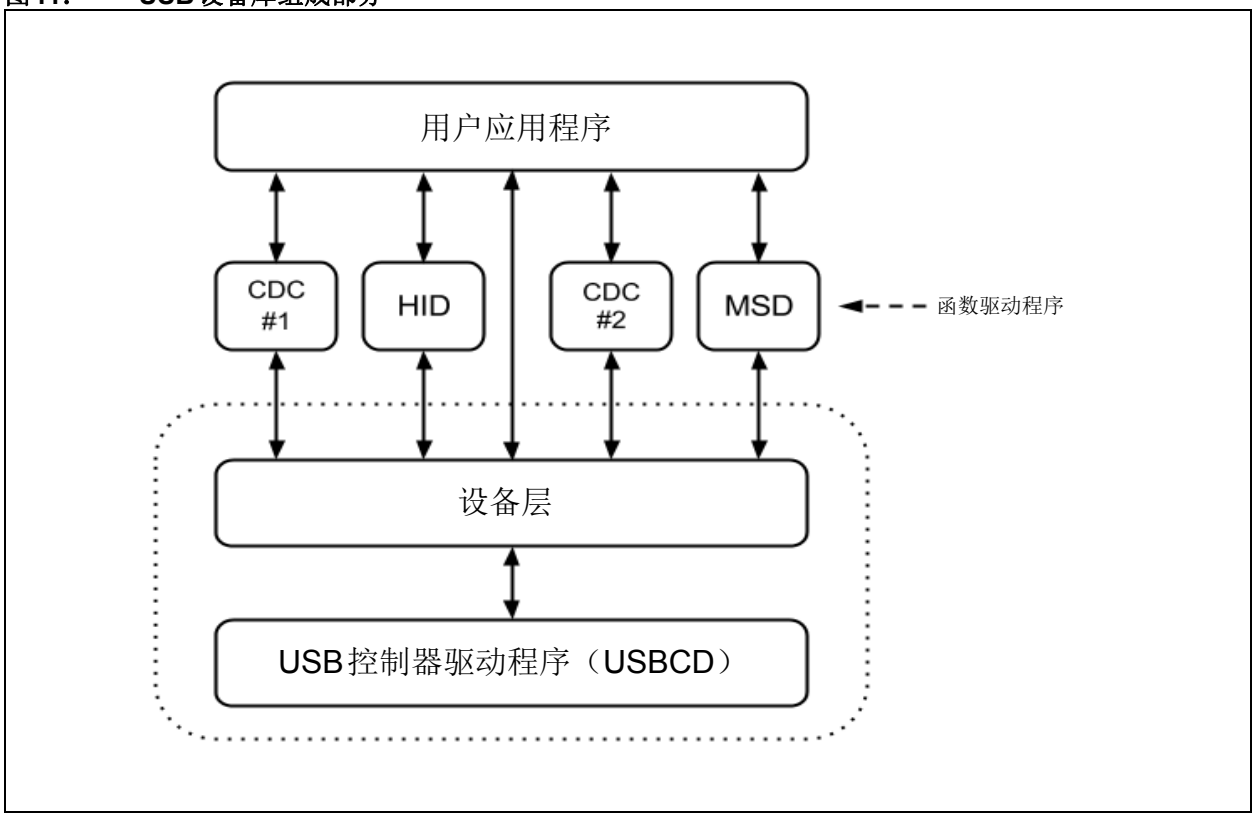


MPLAB HARMONY USB设备库架构概述

MPLAB Harmony USB设备库具有模块化分层框架，使开发人员能够设计和开发各种USB设备。USB设备库通过实现标准USB设备类规范的函数驱动程序来简化标准USB设备的开发。USB设备库包含以下三个主要组成部分，如图11所示。

- USB控制器驱动程序（USB Controller Driver，USBCD）
- 设备层
- 函数驱动程序

图11： USB设备库组成部分



USBCD

USB 控制器驱动程序（USBCD）管理USB 外设的状态，并向设备层提供针对USB 的结构化数据访问方法。它还为设备层提供USB 事件。在USB 外设的每个实例中，它仅支持一个客户端，即设备层。USBCD 仅由设备层访问。建议将USBCD 配置为中断模式。这将减少其他应用程序组件对USB 设备协议栈操作的影响。

USBCD 提供相应函数来实现以下目的：

- 使能、禁止和停止端点
- 安排USB 传输
- 连接或断开设备
- 控制恢复信号传输

设备层

设备层响应USB 主机发出的枚举请求。

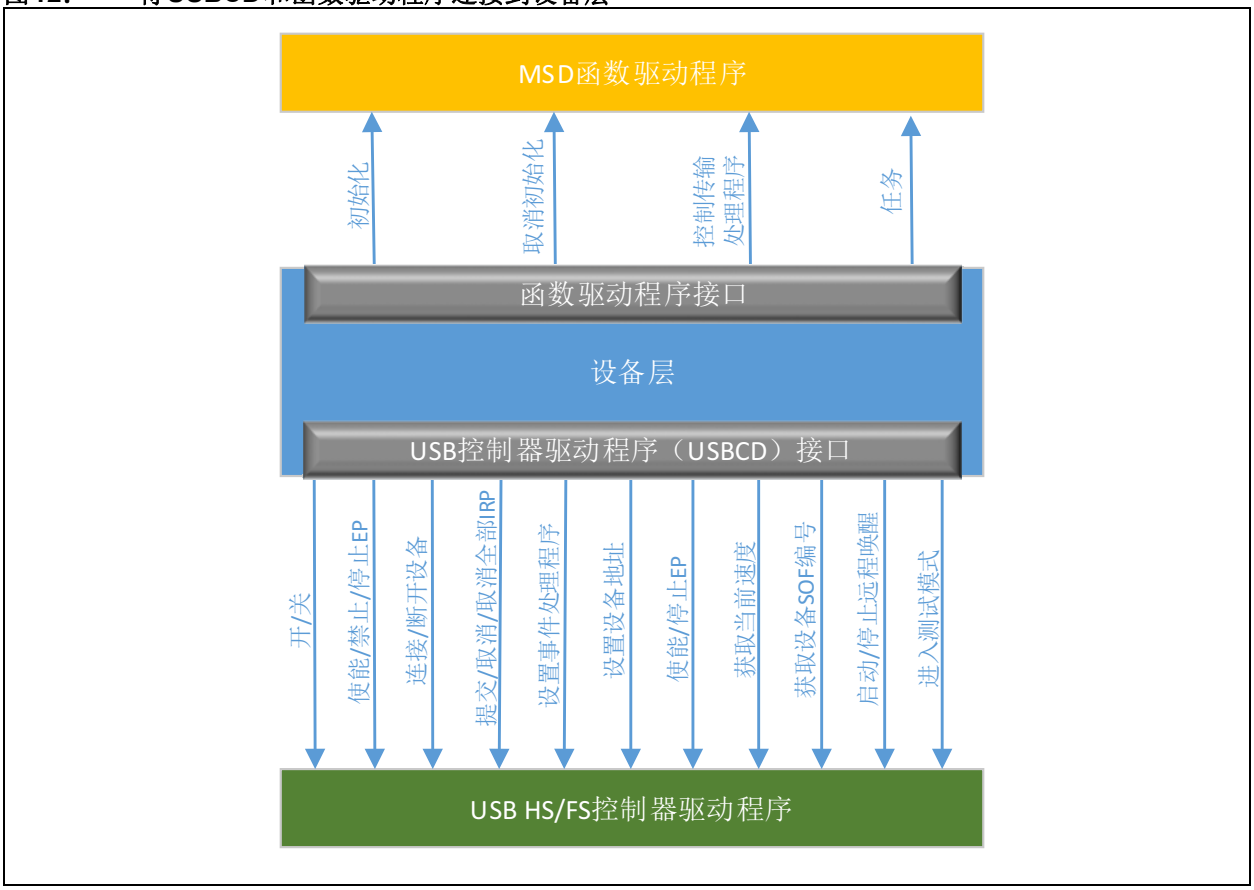
它可以独占访问USBCD 实例和控制端点（端点0）。当主机发出类特定控制传输请求时，设备层将分析控制传输的建立数据包，并将控制传输路由到相应的函数驱动程序。图12所示为设备层如何与USBCD 和函数驱动程序（本例中为MSD）接口。

设备层在从USB 主机收到“设置配置”（对于支持的配置）请求时，将初始化注册的所有函数驱动程序。当USB 复位事件发生时，它将取消初始化函数驱动程序。它将打开USBCD 并注册一个事件处理程序来接收USB 事件。设备层也可以由应用程序打开（应用程序成为设备层的客户端）。然后，应用程序可以接收总线和设备事件并响应控制传输请求。设备层为应用程序提供事件，如“设备已配置”或“设备复位”。其中一些事件是仅通知事件，而其他事件则要求应用程序采取操作。

函数驱动程序

根据类别规范，函数驱动程序可实现各种USB 设备类别。

图12： 将USBCD 和函数驱动程序连接到设备层



USB海量存储设备函数驱动程序

图13所示为应用程序、MSD函数驱动程序、介质驱动程序和USB设备层之间的功能交互。

如图13所示，应用程序不必与MSD函数驱动程序交互。此外，MSD函数驱动程序不具有应用程序可调用的函数。介质驱动程序控制存储介质。应用程序与介质驱动程序交互以更新或访问存储介质上的信息。MSD函数驱动程序与介质驱动程序进行交互，以处理从USB主机接收到的数据读取和写入请求。该数据始终以块的形式访问。

当主机设置包含海量存储接口的配置（“设置配置”控制请求）时，设备层将初始化MSD函数驱动程序。MSD函数驱动程序需要为函数驱动程序的每个实例定义初始化数据结构。

此初始化数据结构的类型应为`USB_DEVICE_MSD_INIT_DATA`。

此初始化数据结构包含以下内容：

- 此MSD函数驱动程序实例中的逻辑单元编号（LUN）的数量
- 指向`USB_MSD_CBW`类型数据结构的指针
- 指向`USB_MSD_CSW`类型数据结构的指针
- 指向介质驱动程序初始化数据结构的数组的指针

图14所示为MSD函数驱动程序初始化数据结构。

图13：　功能交互

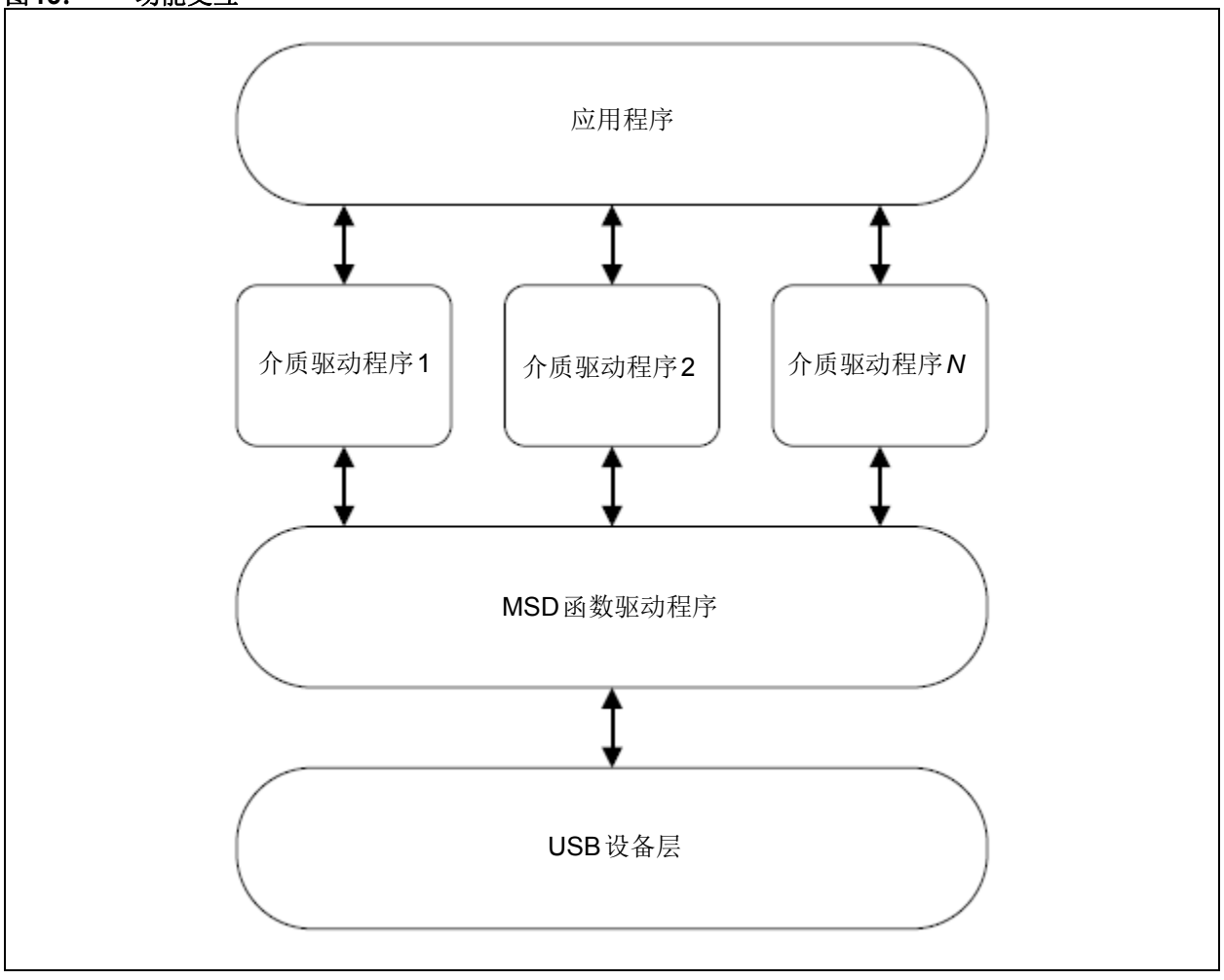
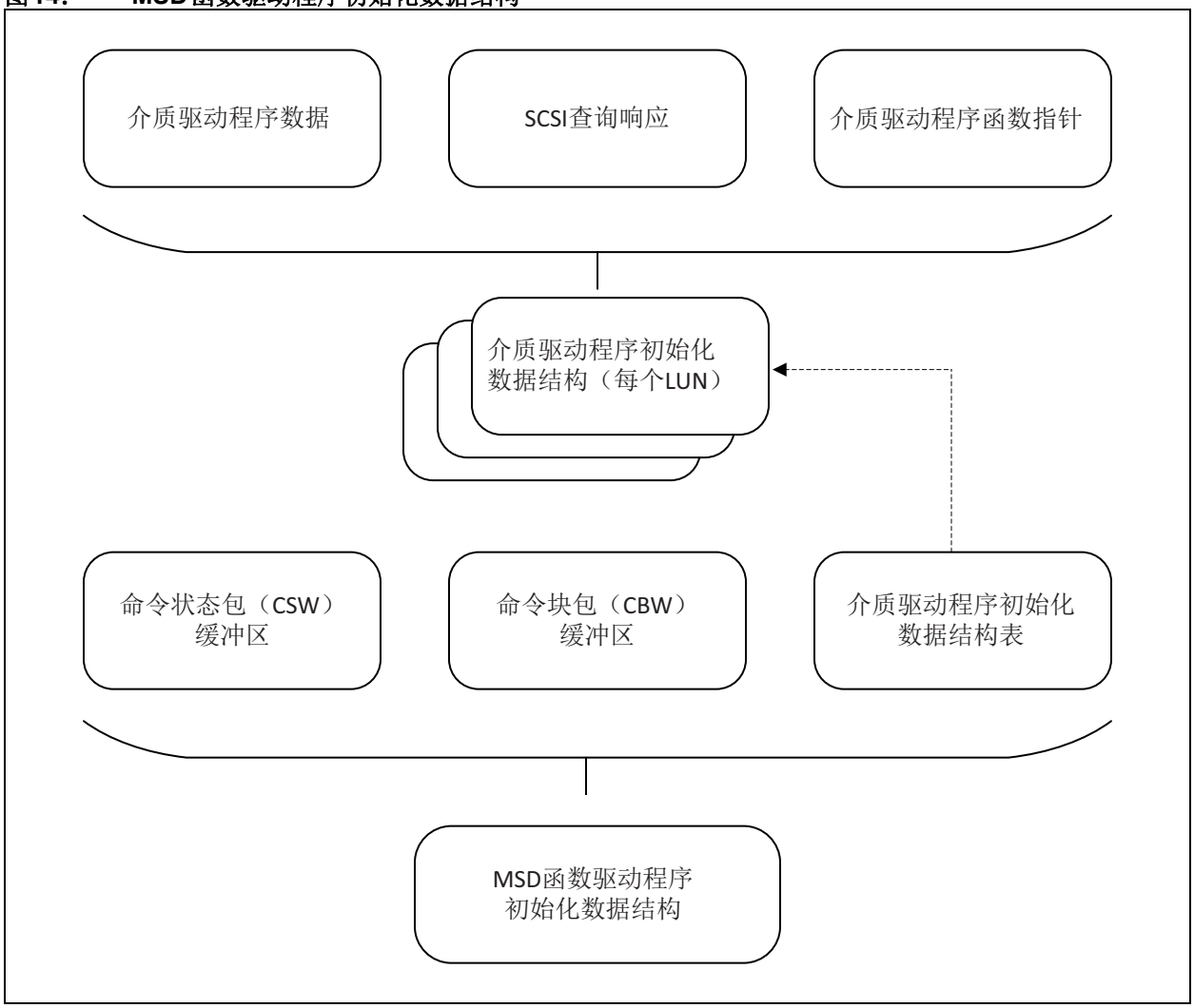


图14： MSD函数驱动程序初始化数据结构



在初始化过程中，MSD函数驱动程序将用USB_DEVICE_MSD_INIT_DATA数据结构中的初始化数据进行初始化。MSD函数驱动程序还将使能批量端点，并将拥有批量端点的独占访问权。

通过在MSD函数驱动程序初始化数据结构中提供介质驱动程序入口点，可将介质驱动程序插入到MSD函数驱动程序中。在多LUN存储的情况下，可以将多个介质驱动程序插入MSD函数驱动程序，每个介质驱动程序都可以访问不同的存储介质类型。

MSD函数驱动程序任务函数在设备层任务函数的上下文中调用。MSD函数驱动程序接口应在USB设备层函数驱动程序注册表中注册。

USB控制请求首先由设备层处理，然后转发到MSD函数驱动程序，以允许处理标准接口请求“获取/设置接口”和类特定接口请求“获取最大LUN”和“USB MSD复位”（对于MSD）。批量端点的标准控制请求由设备层本身处理。MSD函数驱动程序通过设备层访问USB总线和端点0，因为它们由设备层管理。

介质接口

USB_DEVICE_MSD_MEDIA_INIT_DATA数据结构允许将介质驱动程序插入MSD函数驱动程序。任何需要插入到MSD函数驱动程序的介质驱动程序都需要实现由USB_DEVICE_MSD_MEDIA_FUNCTIONS类型指定的接口（函数指针签名）。

对于每个LUN，都需要提供SCSI INQUIRY响应数据结构。

开发介质驱动程序时需要遵循以下准则：

- 读函数应该是非阻塞的。
- 写函数应该是非阻塞的。
- 介质驱动程序应提供一个事件来指示块传输何时完成。它应允许注册事件处理程序。
- 如果需要，写函数应在一次操作中擦除并写入存储区域。MSD函数驱动程序不显式调用擦除操作。
- 介质驱动程序应在需要时提供介质几何对象。凭借此介质几何对象，MSD函数驱动程序可获悉介质特性。此对象的类型为SYS_FS_MEDIA_GEOMETRY。

数据传输

设备配置完成后，设备层将运行MSD函数驱动程序任务。MSD函数驱动程序任务的状态机等待来自USB主机的CBW数据包。接收到CBW数据包后，即对其进行解析，并会处理SCSI命令请求。对于需要介质访问的SCSI命令请求，根据收到的LUN编号，调用相应介质的读写函数来读取和写入介质扇区。

介质事件通过MSD函数驱动程序注册的事件处理程序通知给MSD函数驱动程序。当MSD函数驱动程序与介质驱动程序一起等待介质完成操作时，函数驱动程序将发送NAK来响应MSD数据传输请求的数据阶段。介质驱动程序完成读/写操作后，MSD函数驱动程序将通过设备层向USB驱动程序提交I/O请求数据包（I/O Request Packet，IRP）以响应MSD数据传输请求。USB驱动程序为每个端点维护一个IRP队列，并允许多个请求排入相应的端点队列中。

错误处理

有效CBW —— 如果满足以下条件，则设备认为CBW有效：

- 在设备发送CSW或复位后收到CBW
- CBW的长度为31（0x1F）字节
- *dCBWSignature*等于0x43425355

有意义的CBW —— 如果满足以下条件，则设备认为CBW有意义：

- 没有任何保留的位置1，
- *bCBWLUN*包含设备支持的有效LUN，并且
- *bCBWCBLength*以及CBWCB的内容均符合*bInterfaceSubClass*

无效CBW —— 如果CBW数据包的大小大于31字节，或者CBW中的*dCBWSignature*字段不等于有效签名（0x43425355），则设备将在IN和OUT端点发送STALL条件，而不发送CSW。设备将保持此状态，直到主机通过向设备的两个端点先发送BOMSR命令、再发送“清除特性”（端点暂停）命令执行复位恢复。除此之外，将执行以下检查：

- 无意义的CBW——对于SCSI READ 10或SCSI WRITE 10请求，如果CBW中的*dCBWDataTransferLength*字段不等于CBWCB的“传输长度”字段，则命令失败（*bCSWStatus* = 0x01）
- 无意义的CBW——对于SCSI READ 10或SCSI WRITE 10请求，如果数据传输的方向（由*bmCBWFlags*指示）不正确，则命令失败。
- 对于SCSI READ 10或SCSI WRITE 10请求，如果介质不存在，则CSW中的*bCSWStatus*字段设置为0x01（命令失败），命令失败。请注意：如果介质不存在，则依赖介质进行响应的SCSI请求（例如SCSI READ CAPACITY、SCSI MODE SENSE和SCSI TEST UNIT READY）也将失败。如果介质不存在，则“检测密钥”设置为SCSI_SENSE_NOT_READY（0x02），“附加检测代码（ASC）”更新为SCSI_ASC_MEDIUM_NOT_PRESENT（0x3A）。
- 对于SCSI WRITE 10请求，如果介质受写保护，则命令失败（*bCSWStatus* = 0x01）。此时，SENSE数据将进行更新，“检测密钥”设置为SCSI_SENSE_DATA_PROTECT（0x07），“附加检测代码（ASC）”设置为SCSI_ASC_WRITE_PROTECTED（0x27）。SENSE数据将发送到主机以响应SCSI REQUEST SENSE命令。
- 介质驱动程序错误 —— 对于有效且有意义的CBW，如果介质驱动程序报告任何错误，MSD函数驱动程序将通过CSW中的*bCSWStatus* = 0x01（命令失败）状态响应。

对于一个有效且有意义的CBW，如果由MSD函数驱动程序发送（在IN数据传输期间）或接收（在OUT数据传输期间）的字节数量少于主机在 CBW 数据包的*dCBWDataTransferLength*字段中指示的数量，则MSD函数驱动程序会将CSW的*bCSWStatus*字段设置为0x00（命令通过），并将CSW中的*dCSWDataResidue*字段设置为*dCBWDataTransferLength*与设备接收或发送的实际字节数之差。设备随后将停止批量输入端点。

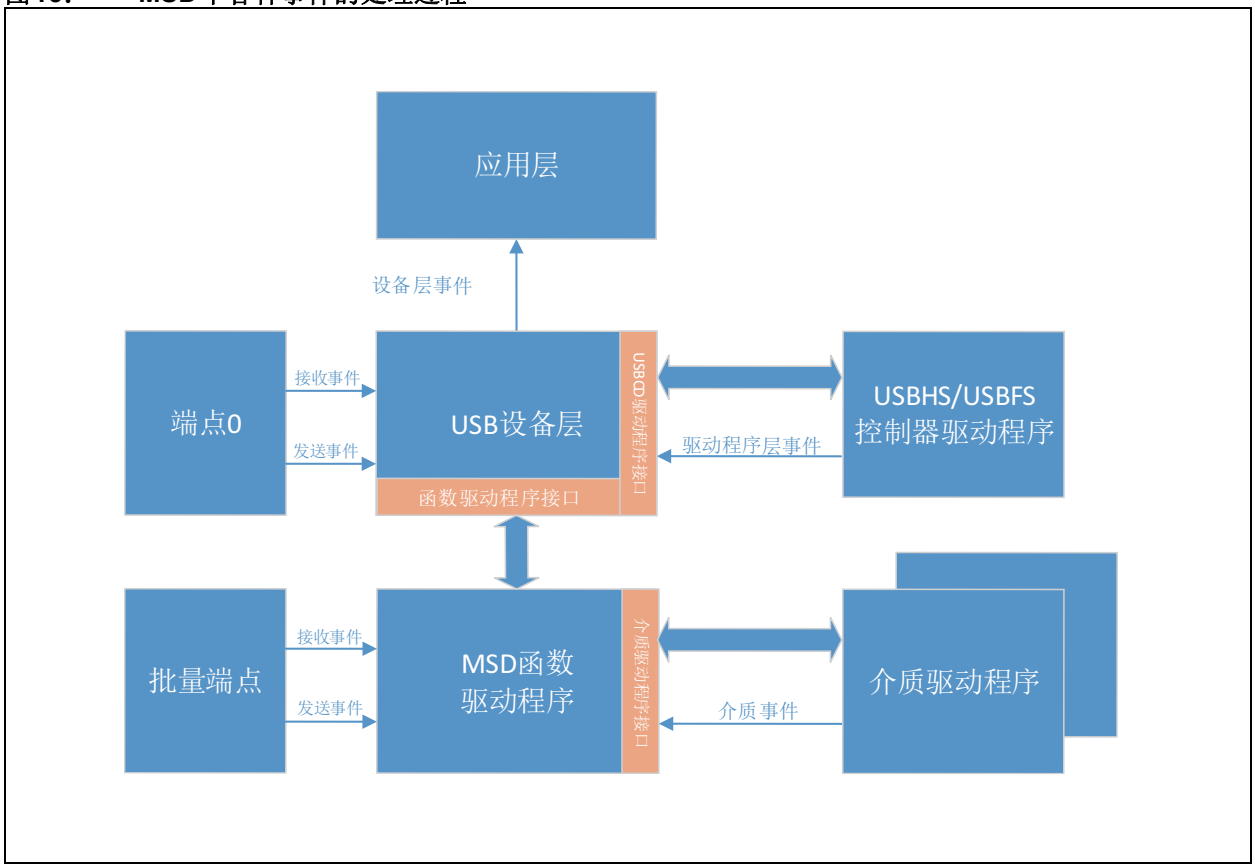
尝试读取CSW时，主机将从设备的输入端点上收到一个STALL条件。收到STALL条件后，主机将发出一个“清除特性”（端点停止）控制请求（带有批量输入端点的地址），以清除批量输入端点上的STALL条件。STALL条件清除后，设备将向主机发送CSW。

对于一个有效且有意义的CBW，在命令执行失败（*bCSWStatus* = 0x01，命令失败）且主机希望从设备接收数据或向设备发送数据（*dCBWDataTransferLength* > 0)）的前提下，如果主机正在从设备读取（*bmCBWFlags* = 1）数据，MSD函数驱动程序将停止输入端点；如果主机正在向设备写入（*bmCBWFlags* = 0）数据，MSD函数驱动程序将停止输出端点。当停止输入端点时，设备将在发送CSW之前通过在控制端点上发出CLEAR FEATURE请求（用于批量输入端点）来等待主机清除批量输入端点上的STALL条件。

事件处理

图15所示为MSD中各种事件的处理过程。

图15： MSD中各种事件的处理过程



应用程序必须为USB设备层注册一个事件处理程序以接收USB_DEVICE_EVENT_CONFIGURED、USB_DEVICE_EVENT_POWER_DETECTED和USB_DEVICE_EVENT_POWER_REMOVED 等设备层事件。当USB_DEVICE_EVENT_POWER_DETECTED事件发生时，可调用USB_DEVICE_Attach函数将设备连接到USB。

设备层通过USB控制器驱动程序注册一个事件处理程序以从控制器驱动程序接收DRV_USB_EVENT_RESET_DETECT、DRV_USB_EVENT_RESUME_DETECT和DRV_USB_EVENT_IDLE_DETECT 等事件。处理事件后，设备层将调用应用程序事件处理程序以允许应用程序处理这些事件。

MSD函数驱动程序通过与每个逻辑单元对应的介质驱动程序注册一个事件处理程序。MSD函数驱动程序将在完成介质操作后接收介质事件：SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_COMPLETE和SYS_FS_MEDIA_EVENT_BLOCK_COMMAND_ERROR。

MSD函数驱动程序不会向应用程序提供任何事件。应用程序不必干预MSD函数驱动程序的功能。

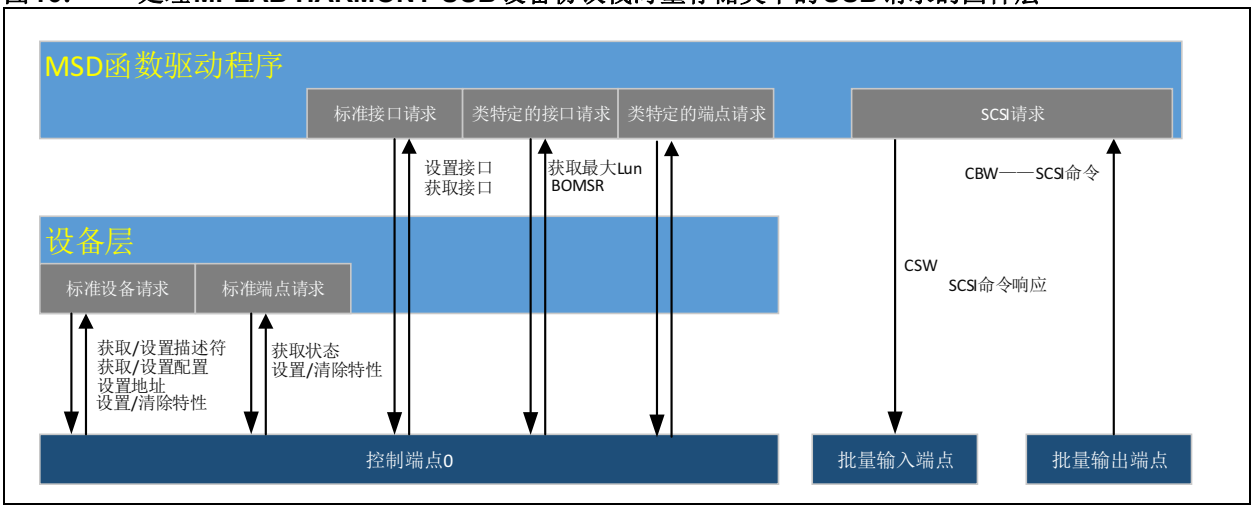
设备层还将注册事件处理程序以接收端点0发送和接收事件。接收或发送控制数据包后，将调用这些回调函数。

MSD函数驱动程序还可以注册事件处理程序以接收由MSD函数驱动程序管理的批量端点的事件。

对于已由MSD函数驱动程序打开的介质驱动程序，有时应用程序仍可将其打开。如果应用程序和MSD函数驱动程序尝试写入同一介质驱动程序，结果可能会不可预测。建议在USB设备插入主机时，令应用程序限制对介质驱动程序的写入访问。

图16所示为处理MPLAB Harmony设备协议栈海量存储类中不同USB请求的固件层。设备层处理设备和端点的标准请求。接口的标准请求以及接口和端点的类特定请求由设备层转发到海量存储函数驱动程序。

图16： 处理MPLAB HARMONY USB设备协议栈海量存储类中的USB请求的固件层

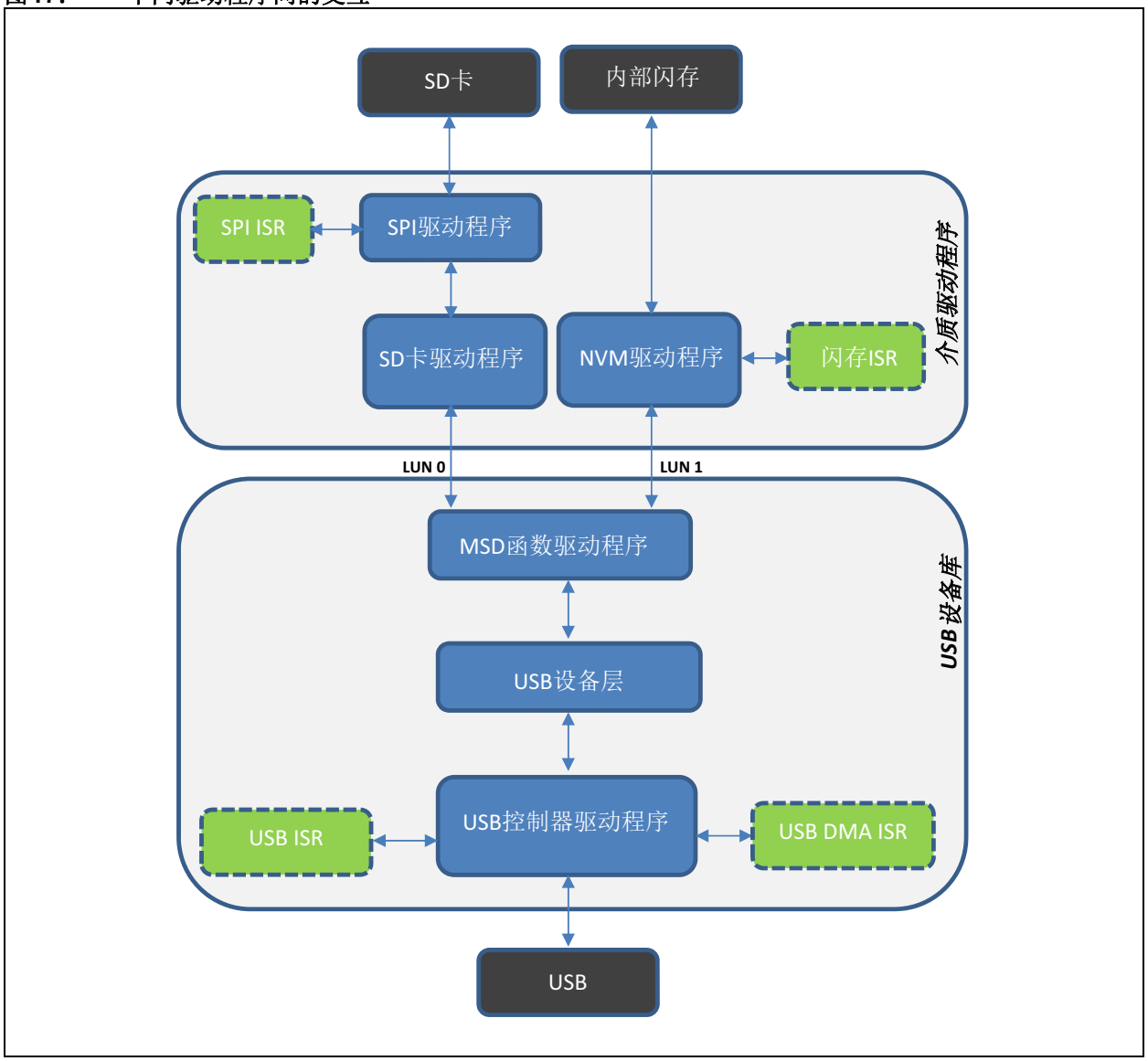


使用MPLAB HARMONY配置USB海量存储设备协议栈以支持两个逻辑单元

本部分将介绍使用MPLAB Harmony创建多LUN MSD所需的配置步骤。SD卡将作为一个逻辑单元，PIC32MZ内部闪存将作为另一个逻辑单元。二者连接到USB主机PC时，将作为两个独立驱动器显示。

图17所示为不同驱动程序之间的交互。对于LUN 0读/写请求，MSD函数驱动程序使用SD卡驱动程序从/向SD卡读取/写入数据。同样，LUN 1读/写请求由MSD函数驱动程序处理，MSD函数驱动程序使用NVM驱动程序访问NVM介质（内部闪存）。SD卡驱动程序使用SPI驱动程序与SD卡进行交互。MHC生成必要的代码以将介质驱动程序与MSD函数驱动程序的介质接口绑定。

图17：　不同驱动程序间的交互



硬件要求

- PIC32MZ EF Curiosity开发板
- microSD Click板和SD卡
- MPLAB REAL ICE调试器（可选）
- USB Type-A转Micro-B电缆

软件要求

- MPLAB X IDE v3.40或更高版本
- MPLAB Harmony v2.02或更高版本

使用MPLAB Harmony配置器（MHC）

使用MHC需遵循以下步骤：

1. 新建一个MPLAB Harmony项目，然后配置BSP和时钟。
2. 为两个逻辑单元（SD卡和NVM）配置海量存储功能的USB设备协议栈。
3. 配置SD卡、SPI驱动程序和SPI I/O引脚。
4. 配置NVM驱动程序。
5. 生成MHC代码。
6. 添加并修改应用程序文件。
7. 编译并运行代码。

步骤1：创建MPLAB Harmony项目，然后配置BSP和时钟。

1. 在MPLAB X IDE中，选择*File > New Project*（文件 > 新建项目），然后新建一个32位MPLAB Harmony项目。指定项目位置、项目名称，然后将PIC32MZ2048EFM100选作目标器件。
2. 单击**Finish**（完成），创建项目（见图18）。
3. 在MPLAB X IDE中，选择*Tools > Embedded*（工具 > 已安装工具），然后打开**MPLAB Harmony Configurator**（MPLAB Harmony配置器）。
4. 在MHC树形视图中，展开“BSP Configuration”（BSP配置），然后选中**USE BSP?**（使用BSP？）。
5. 选择PIC32MZ EF Curiosity Development Board（PIC32MZ EF Curiosity开发板）。选择BSP将自动配置时钟和板特定的硬件（GPIO、LED和开关），如图19所示。

图18： 新建项目

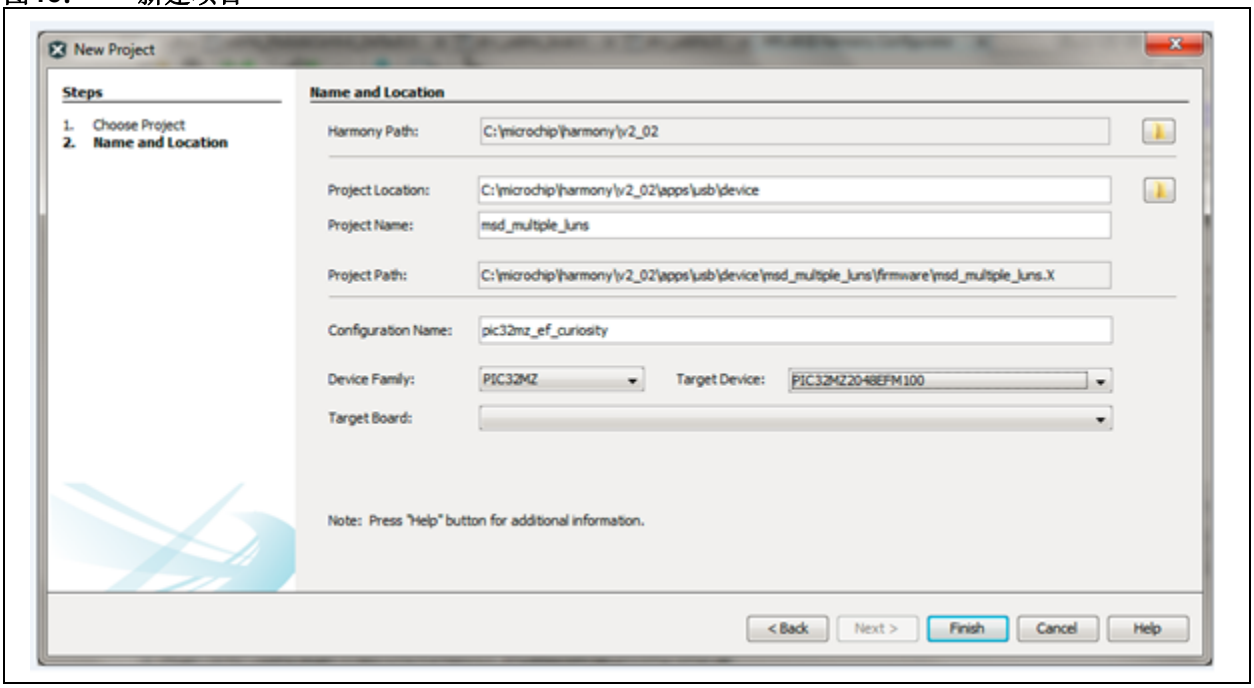
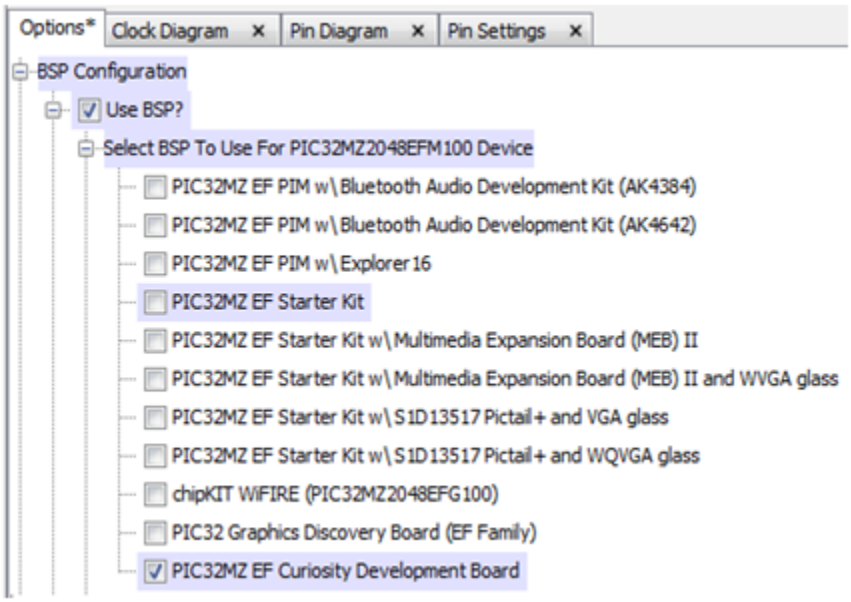
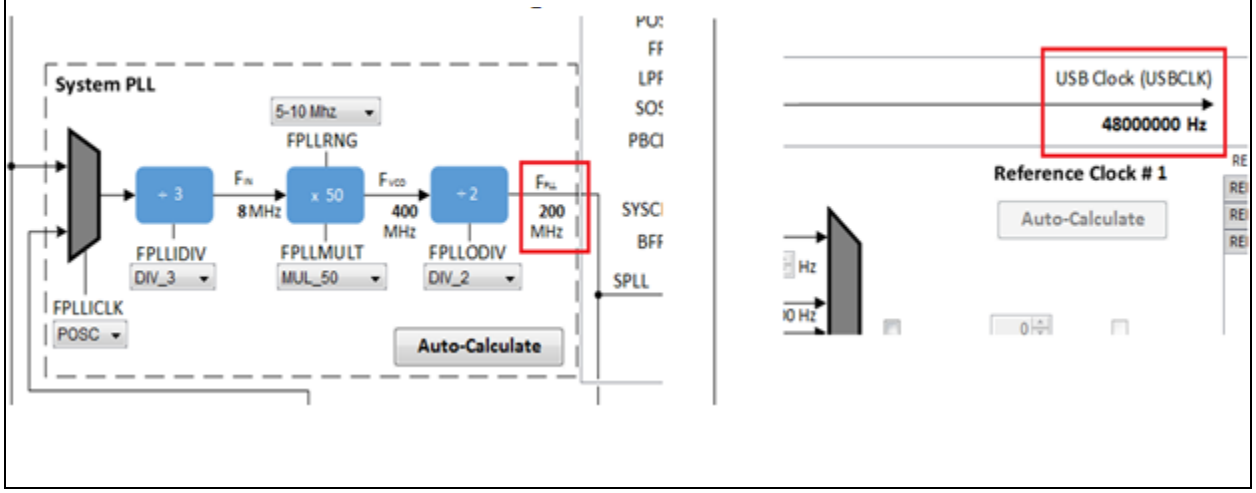


图19： MHC中的BSP选择



6. 单击Clock Diagram（时钟图）选项卡，并验证系统PLL输出是否设置为200 MHz，USB时钟是否设置为48 MHz，如图20所示。

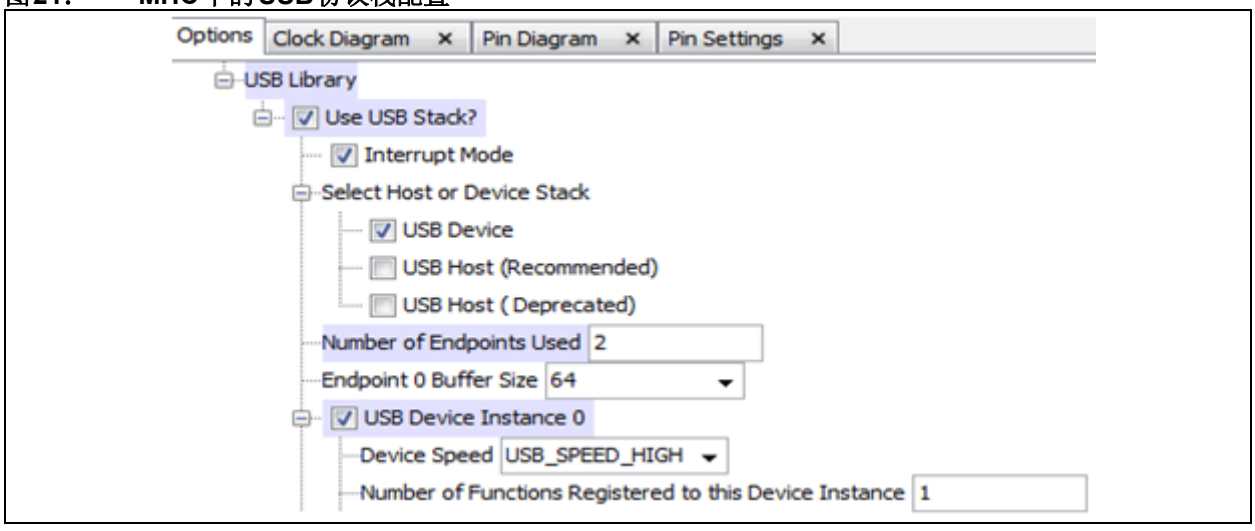
图20： MHC中的时钟配置



步骤2：为两个逻辑单元（SD卡和NVM）配置海量存储功能的USB设备协议栈

1. 选择*Harmony Framework Configuration > USB Library*（Harmony框架配置 > USB库）。
2. 选中**Use USB Stack?**（使用USB协议栈？）。默认选择的是Interrupt Mode（中断模式）。这表示USB驱动程序状态机将从中断上下文运行。
3. 将Select Host or Device Stack（选择主机或设备协议栈）设置为USB Device（USB设备）。
4. 将端点数更改为2。USB MSD使用仅批量传输（BOT）协议。一个端点是用于控制数据包传输的控制端点（EP0）。另一个端点是用于在主机和设备之间进行数据传输的批量端点（批量输入和批量输出）。
5. 保持Endpoint 0 Buffer Size（端点0缓冲区大小）为64。对于高速设备，EP0的大小固定为64。对于全速设备，EP0的大小可以是8字节、16字节、32字节或64字节。
6. 默认选择的是USB Device Instance 0（USB设备实例0）。每个USB设备实例代表单片机上的一个USB外设。由于PIC32MZ器件上只有一个USB外设，因此只有一个USB设备实例。用户需对其进行扩展。
7. 保持设备速度为默认的 USB_SPEED_HIGH。PIC32MZ器件支持全速和高速操作。选择高速将允许器件以全速和高速工作。
8. 将Number of Functions Registered to this Device Instance（此设备实例所注册函数的数量）设置为1。这表示此设备实例注册的USB设备类驱动程序的数量。本例中将为MSD 函数驱动程序（见图21）。
9. 选择 Function1（函数 1），将其展开。为 USB MSD操作配置函数驱动程序。
10. 将Device Class（设备类别）选为MSD。
11. 选择此函数驱动程序将连接的 USB 配置值。USB 设备会将活动配置设置为通过主机的SET CONFIGURATION 控制命令接收的配置值。USB设备任务将遍历所有注册的函数驱动程序并尝试将活动配置的值与该函数驱动程序的USB配置值相匹配。当检测到匹配时，相应的函数驱动程序的任务就会运行。本例中将保留默认值1。
12. 将Start Interface Number（起始接口编号）保留默认值0。这表示接口编号0为此（MSD）函数驱动程序所有。这将导致针对接口的标准和类特定控制请求以及类特定端点请求转发到管理接口编号0的函数驱动程序（本例中为MSD）。
13. 该条目的 Speed（速度）项用于指定应为该函数驱动程序初始化的设备速度。该项可设置为USB_SPEED_FULL、USB_SPEED_HIGH 或者两者的逻辑或组合。如果设备的连接速度与条目的Speed项中提到的速度相匹配，则设备层将初始化该函数。要进行高速和全速操作，请将其设置为USB_SPEED_HIGH|USB_SPEED_FULL。

图21： MHC中的USB协议栈配置



14. 将Endpoint Number（端点编号）设置为1。这表示端点编号1将用于批量输入和批量输出传输。为Endpoint Number选择的值将反映在端点描述符的端点地址字段中。根据端点描述符中的端点地址，MSD函数驱动程序将初始化要进行批量输入和批量输出传输的相应端点。
15. 将Max number of sectors to buffer（缓冲区的最大扇区数）设置为1。这将留出大小为512 x 1字节的缓冲区。要缓冲按SCSI_READ_10命令所述从介质读取的数据，也可以更改该值。缓冲介质数据将最大程度减少为响应输入数据请求而发送到USB主机的NAK数，从而可提高总吞吐量，但是会增加对RAM的使用。
16. 将Number of Logical Units（逻辑单元数）的值设置为2，即SD卡和内部闪存（NVM）。
17. 展开LUN 0并将Media Type（介质类型）选择为SDCARD。选择SDCARD将导致MHC自动使能SD卡和SPI驱动程序。
18. 展开LUN 1并将Media Type选择为NVM。选择NVM将导致MHC自动使能NVM驱动程序，请参见图22。
19. 保留供应商ID、产品ID、制造商字符串和产品字符串的默认值。
20. 保留USB中断和USB DMA中断的默认优先级和次优先级值，请参见图23。
21. 保持Power State（电源状态）为SYS_MODULE_POWER_RUN_FULL，并取消选中其他值。

图22： MHC中的USB协议栈配置（续）

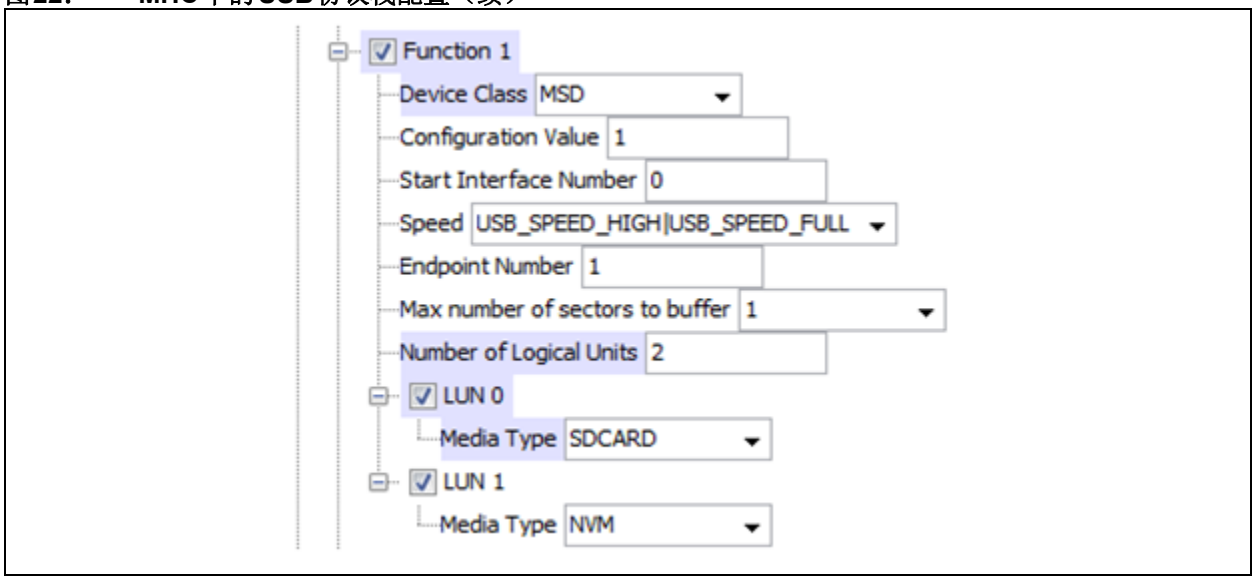
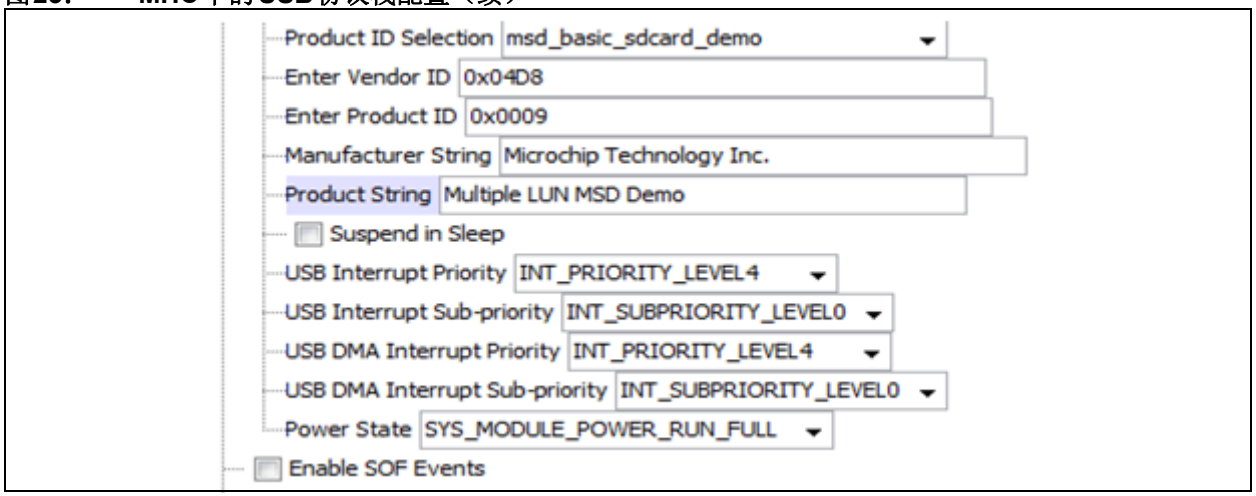


图23： MHC中的USB协议栈配置（续）



步骤3：配置SD卡、SPI驱动程序和SPI I/O引脚

1. 选择*Harmony Framework Configuration > Drivers > SD Card*（Harmony框架配置 > 驱动程序 > SD卡）。
2. 由于LUN0介质类型选为SDCARD，因此已选中“**Use SD Card Driver?**”（使用SD卡驱动程序？）选项。
3. 对于“Clock To Use”（要使用的时钟），请选择CLK_BUS_PERIPHERAL_2，然后清除“Enable Write Protect Check?”（使能写保护检查？）。
4. 对于“Chip Select Port”（片选端口），请选择PORT_CHANNEL_D；对于“Chip Select Port Bit”（片选端口位），请选择PORTS_BIT_POS_5。
5. 清除“Register with File System?”（通过文件系统注册？）。SD存储器将由USB海量存储驱动程序访问，请参见图24。
6. 选择*Harmony Framework Configuration > Drivers > SPI*（Harmony框架配置 > 驱动程序 > SPI）。
7. 由于SD卡配置为使用SPI驱动程序实例0，因此已选中“Use SPI Driver?”（使用SPI驱动程序？）选项。SPI驱动程序配置为8位、增强型缓冲区模式和主模式，请参见图25。
8. 展开SPI驱动程序实例0。请注意，驱动程序配置为中断模式操作。由于SD Click板连接到PIC32 MZ Curiosity开发板上的SPI外设2，因此将SPI Module ID（SPI模块ID）更改为SPI_ID_2，请参见图26。
9. 将Clock Mode（时钟模式）更改为DRV_SPI_CLOCK_MODE_IDLE_LOW_EDGE_FALL，将Input Phase（输入阶段）更改为SPI_INPUT_SAMPLING_PHASE_AT_END，请参见图27。
10. 要配置SPI I/O线，请选择*MPLAB Harmony Configurator > Pin Table*（MPLAB Harmony配置器 > 引脚表）。
11. 将SCK2映射到引脚10，将SDI2映射到引脚88，将SDO2映射到引脚11，如图28所示。

图24： SD卡驱动程序MHC配置

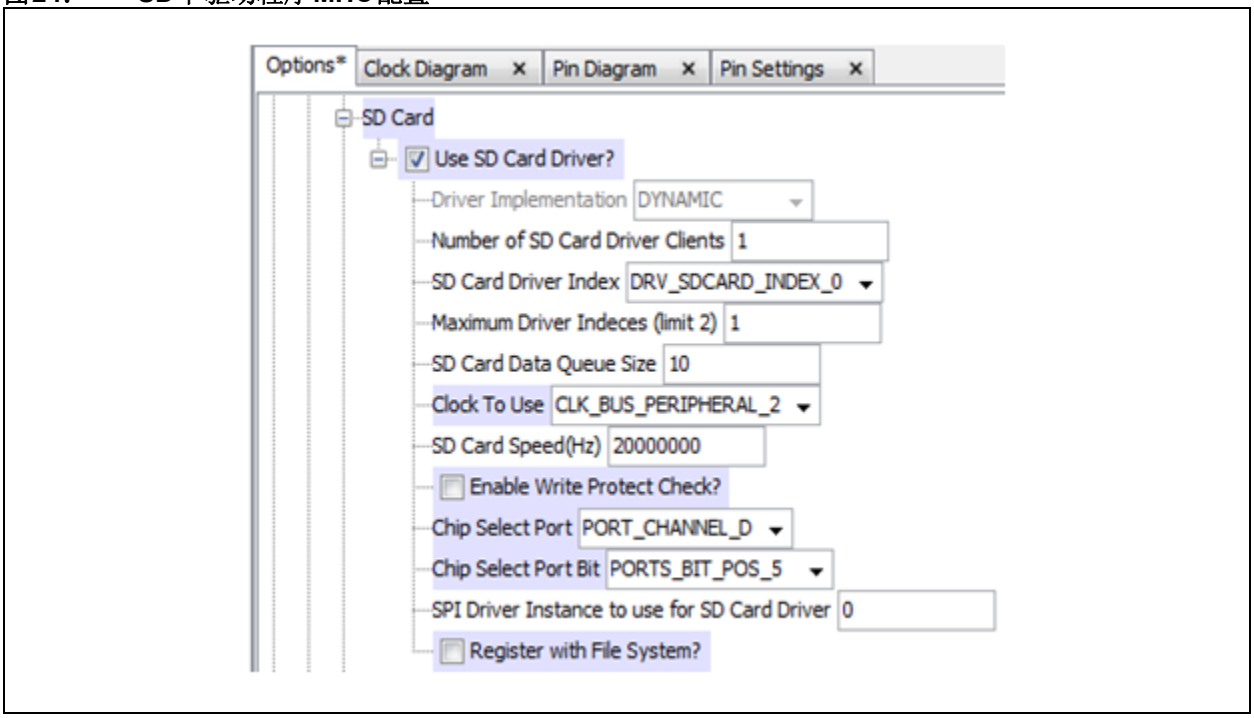


图25： SPI驱动程序MHC配置

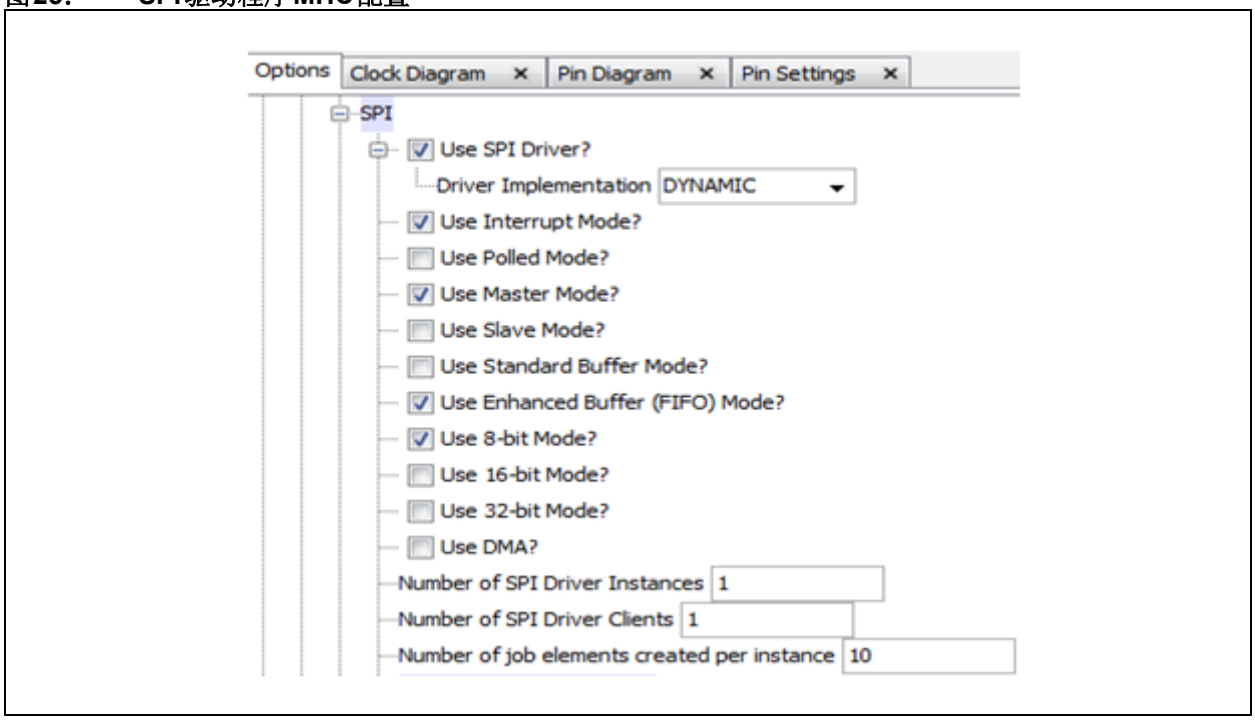


图26： SPI驱动程序MHC配置（续）

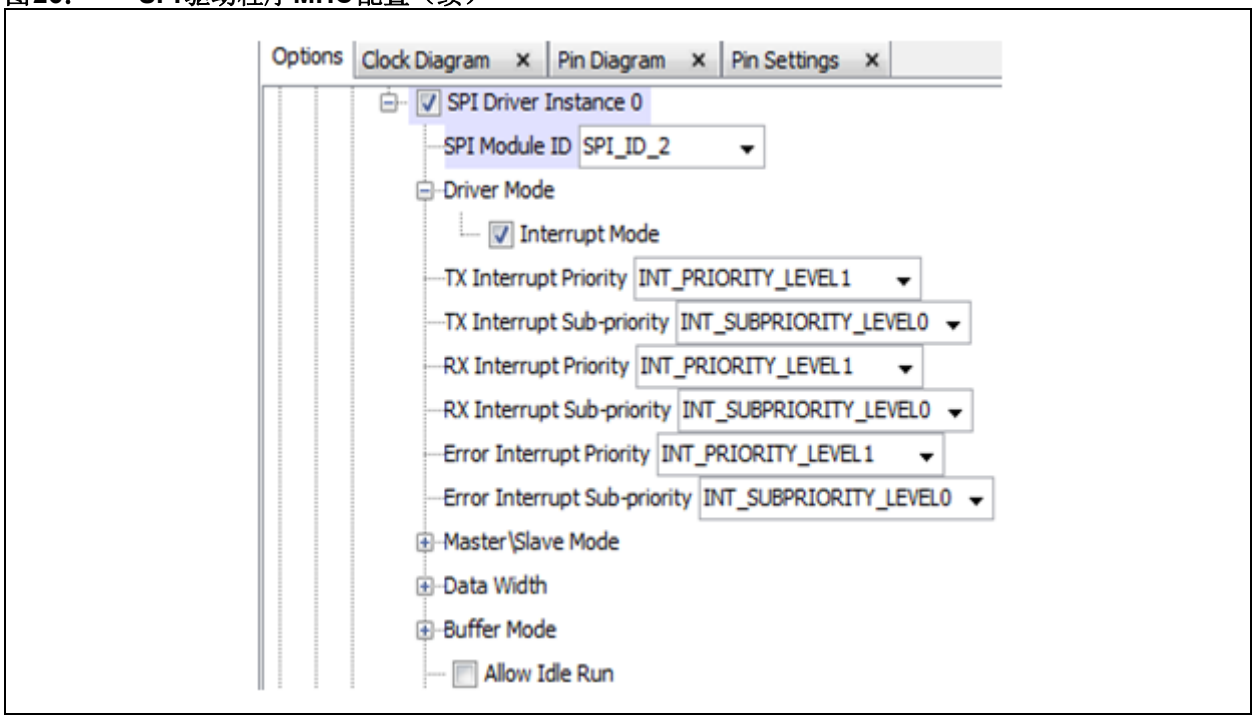


图27： SPI驱动程序MHC配置（续）

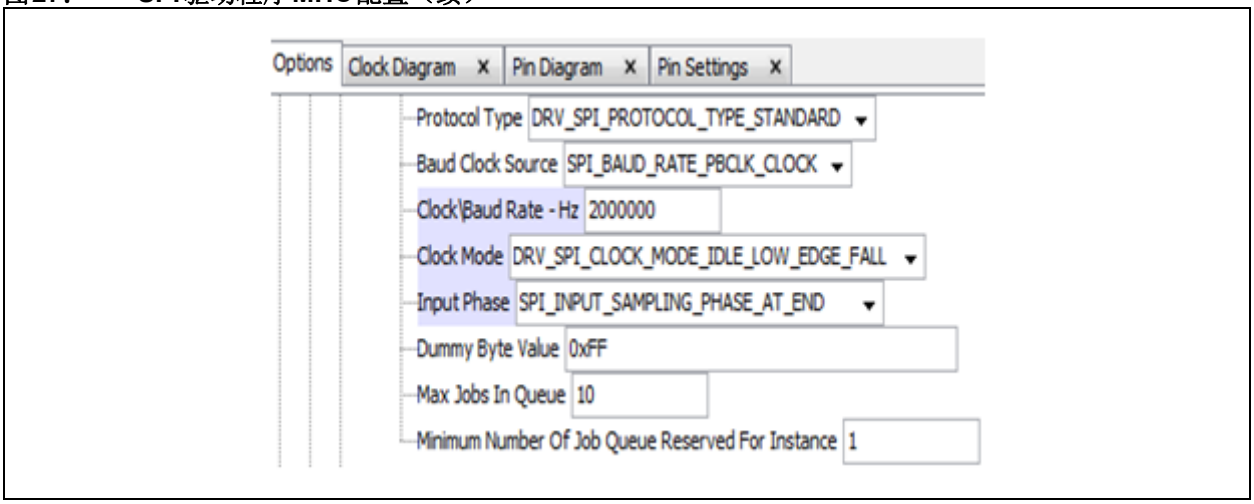
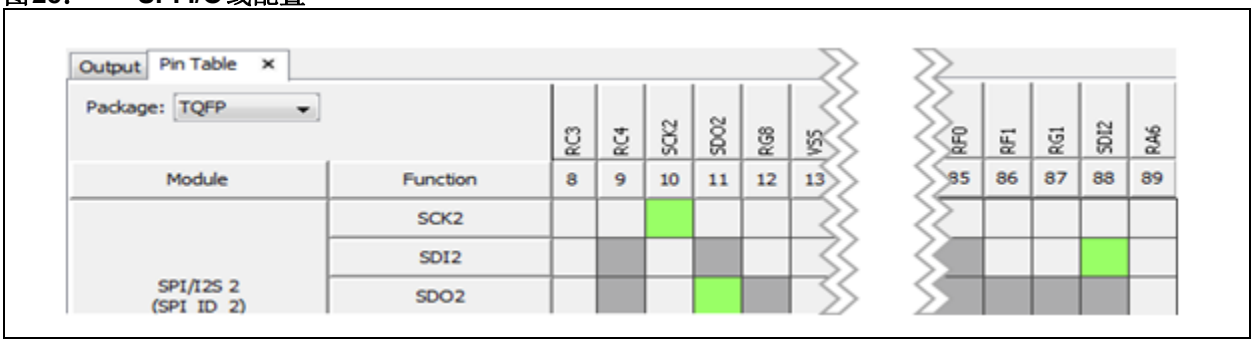


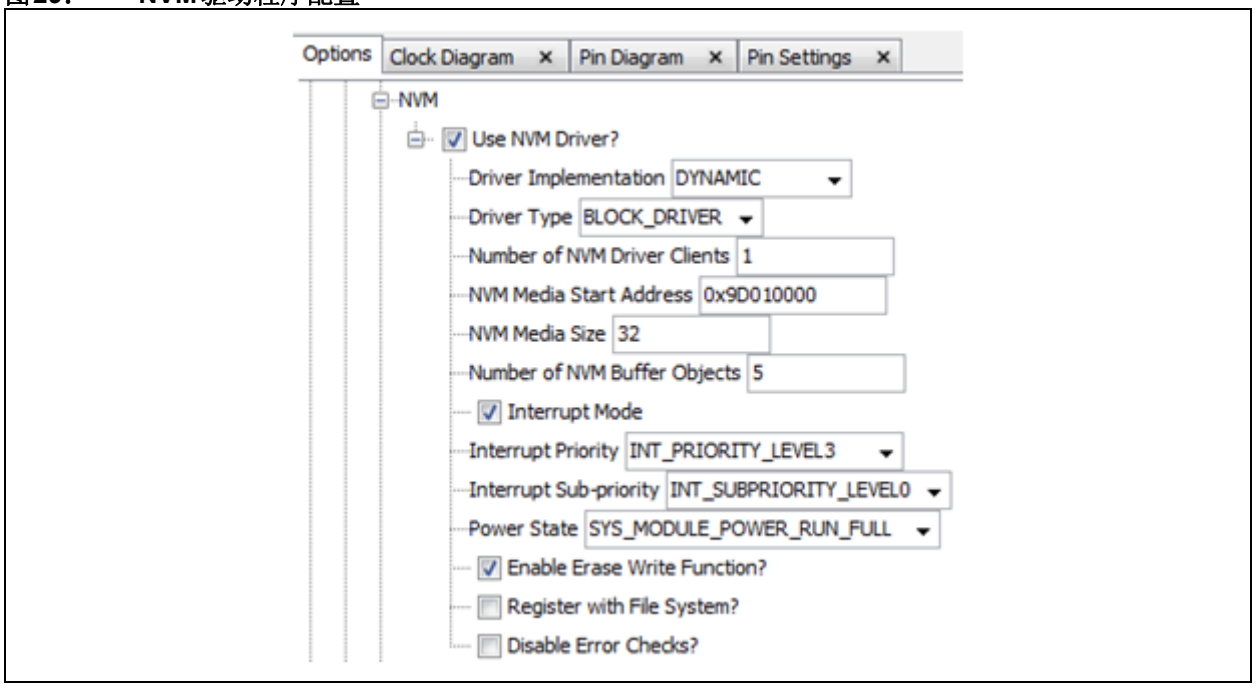
图28： SPI I/O线配置



步骤4：配置NVM驱动程序

1. 选择*Harmony Framework Configuration > Drivers > NVM*（Harmony框架配置 > 驱动程序 > NVM）。
2. 由于LUN 1的介质配置为NVM，因此已选中“Use NVM Driver?”（使用NVM驱动程序？）。NVM驱动程序配置为动态和中断操作模式。
3. 对于“NVM Media Start Address”（NVM介质起始地址），请输入0x9D010000；对于“NVM Media Size”（NVM介质大小），请输入32（KB）。NVM介质上的FAT12文件系统将安装到NVM介质起始地址指向的地址。
4. 选中**Enable Erase Write Function?**（使能擦写功能？），使能行擦写功能，请参见图29。

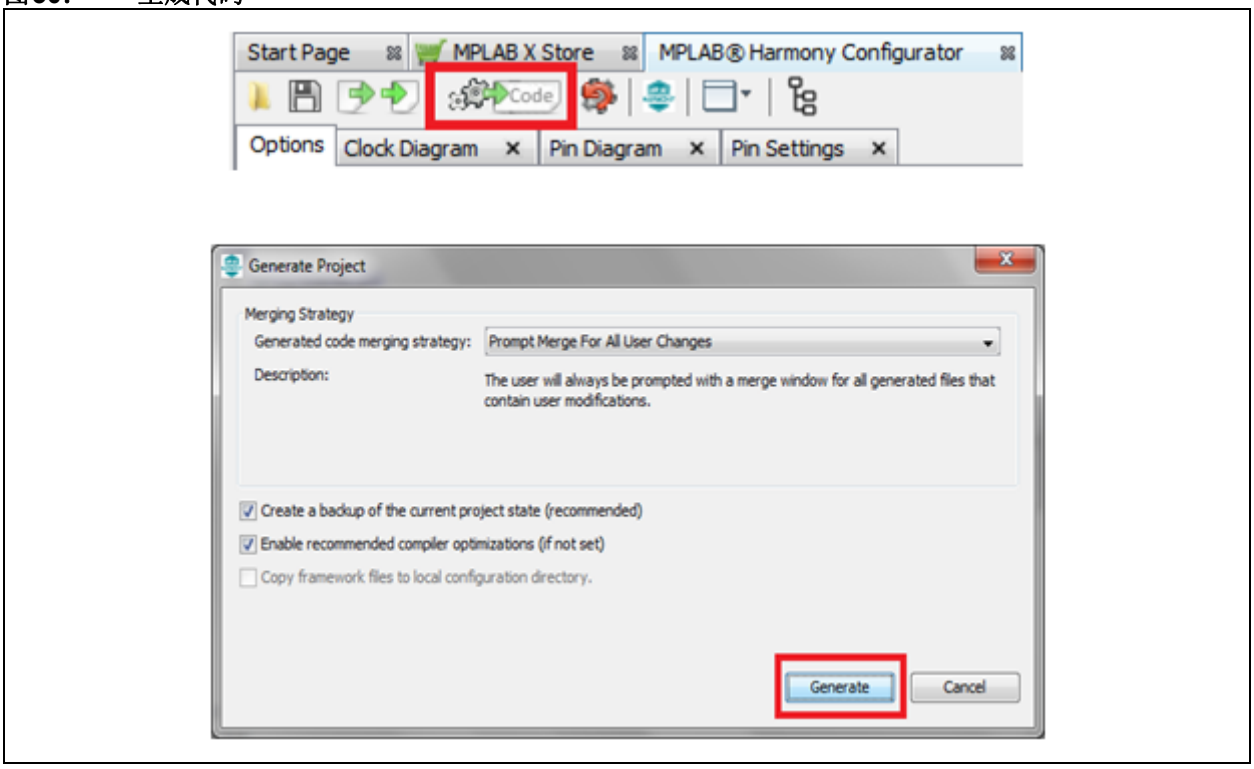
图29： NVM驱动程序配置



步骤5：生成MHC代码

保存MHC 配置，然后单击**Generate**（生成）生成代码，请参见图30。

图30：　生成代码



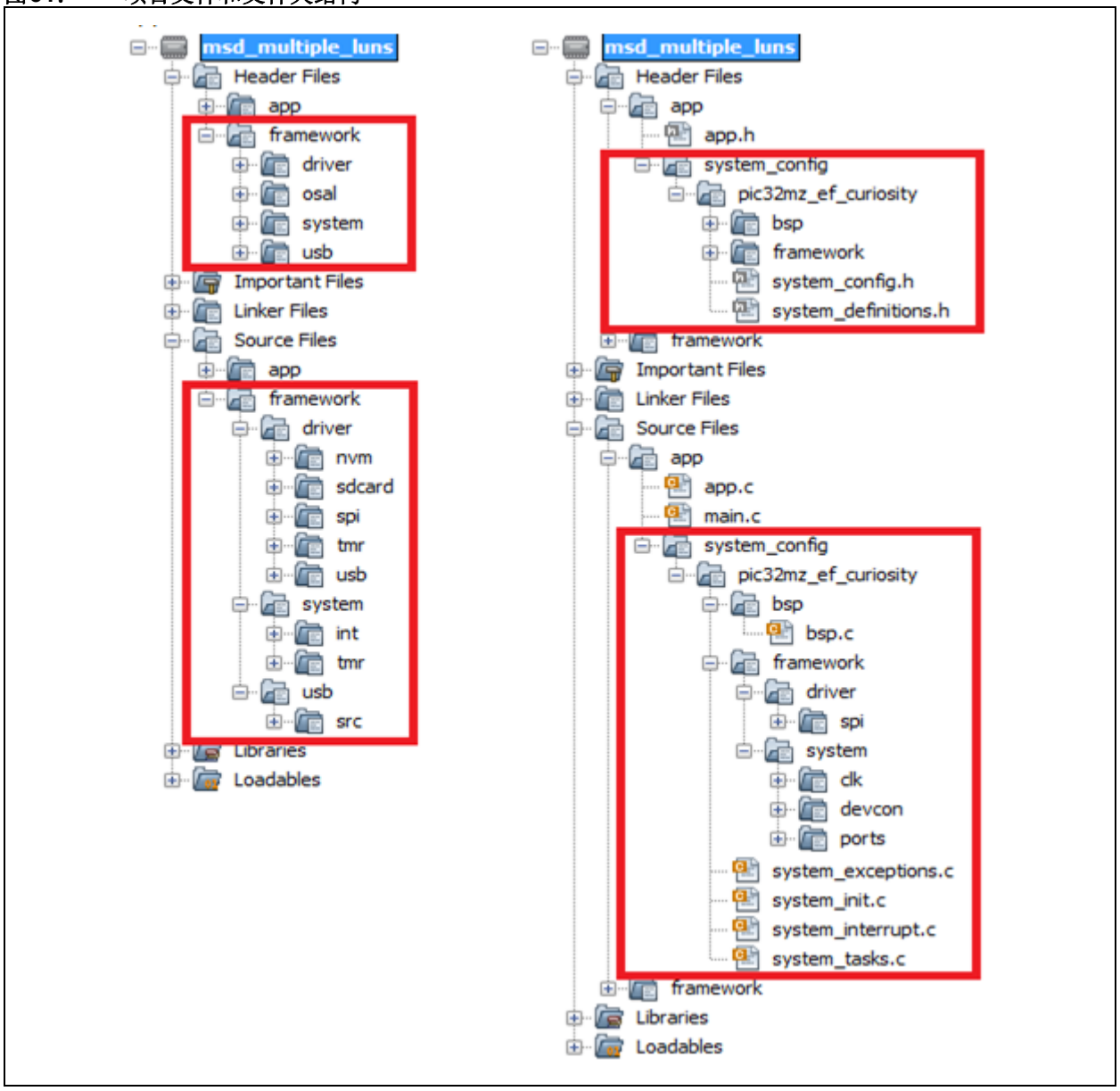
MHC将根据MHC选择生成代码。

确保Merging Strategy（合并策略）选项设置为Prompt Merge For All User Changes（提示合并所有用户更改）。由于没有用户更改，因此不应有任何合并冲突。

Source　Files（源文件）文件夹下的framework（框架）文件夹中包含由MHC（基于MHC选择）添加到项目中的标准MPLAB Harmony框架文件，请参见图31。nvmsdcardspitmr 和 usb 驱动程序文件夹添加到driver文件夹中。系统文件夹包含系统服务使用的驱动程序。usb文件夹包含相关的USB设备协议栈源文件。

app文件夹下的framework文件夹包含自定义的MPLAB Harmony框架文件。这些文件由MHC生成，以响应特定的MHC选择。system_init.c包含各种驱动程序的初始化数据和系统初始化函数SYS_Initialize。SYS_Initialize函数通过初始化时钟系统、BSP、I/O、系统服务、驱动程序和中断来执行系统初始化。随后将调用APP_Initialize函数以进行应用程序初始化。

图31：　项目文件和文件夹结构



system_interrupt.c包含所配置驱动程序的中断处理程序。system_tasks.c包含SYS_Tasks函数，该函数为各种驱动程序和USB 协议栈运行状态机。system_config.h文件包含基于MHC选择的驱动程序配置定义。

步骤6：添加并修改应用程序文件

注册应用程序事件处理程序以接收USB 设备层事件

app.c文件由MHC生成，该文件在APP_Initialize下提供用于初始化应用程序的模板，在APP_Tasks下提供用于运行应用程序任务的模板。应用程序必须为USB设备层注册事件处理程序以接收设备层事件。调用USB_DEVICE_Open会为USB设备层的特定实例返回处理程序。随后，将使用此处理程序通过调用USB_DEVICE_EventHandlerSet函数来为USB设备层注册应用程序事件处理程序，如例1所示。

注册的事件处理程序将由USB设备层调用，以通知设备层事件，例如“USB设备复位”、“USB设备已配置”和“检测到USB设备电源”等。在“检测到 USB 设备电源”的事件中，应用程序必须调用USB_DEVICE_Attach函数，将设备连接到USB，如例2所示。

例1： 打开USB设备层并注册事件处理程序

```
void APP_Tasks ( void )
{

    /* 检查应用程序的当前状态。*/
    switch ( appData.state )
    {
        /* 应用程序的初始状态。*/
        case APP_STATE_INIT:
        {
            appData.usbDevHandle = USB_DEVICE_Open(USB_DEVICE_INDEX_0, DRV_IO_INTENT_READWRITE);

            if(appData.usbDevHandle != USB_DEVICE_HANDLE_INVALID)
            {
                /* 设置事件处理程序。我们将在设置处理程序后
                 * 开始接收事件 */
                USB_DEVICE_EventHandlerSet(appData.usbDevHandle, APP_USBDeviceEventHandler,
                                           (uintptr_t)&appData);

                /* 将应用程序移到下一个状态 */
                appData.state = APP_STATE_SERVICE_TASKS;
            }
```

例2：　　处理USB设备层事件

```
void APP_USBDeviceEventHandler( USB_DEVICE_EVENT event, void * pEventData, uintptr_t context )
{
    /* 这是有关如何使用
       事件处理程序中的上下文参数的示例。*/

    APP_DATA* appData = (APP_DATA*)context;

    switch( event )
    {
        case USB_DEVICE_EVENT_RESET:
        case USB_DEVICE_EVENT_DECONFIGURED:

            /* 设备已复位或取消配置。更新LED状态 */
            BSP_LEDOn ( BSP_LED_1 );
            BSP_LEDOn ( BSP_LED_2 );
            BSP_LEDOn ( BSP_LED_3 );
            break;

-------------------------------------------------------------------------------

    case USB_DEVICE_EVENT_POWER_DETECTED:

            /* 检测到VBUS。连接设备。*/
            USB_DEVICE_Attach(appData->usbDevHandle);
            break;

        case USB_DEVICE_EVENT_POWER_REMOVED:

            /* 未检测到VBUS。断开设备 */
            USB_DEVICE_Detach(appData->usbDevHandle);
            break;

        /* 这些事件不在本演示中使用 */
        case USB_DEVICE_EVENT_RESUMED:
        case USB_DEVICE_EVENT_ERROR:
        case USB_DEVICE_EVENT_SOF:
        default:
            break;
    }
}
```

根据FAT12文件系统格式化NVM存储区域

NVM存储区域必须使用文件系统进行格式化，以使USB主机能够了解介质上文件的布局。FAT12是一个轻量级的文件系统，开销极低。此外，FAT12文件系统可以处理最大12 MB的容量，对于32 KB大小的NVM介质分区而言绰绰有余。`diskImage.c`文件包含NVM存储区域的FAT12映像。FAT12映像将介质格式化为包含`FILE.TXT`的文件，该文件包含字符串“Data”。`diskImage.c`文件不是由MHC生成的，而是可以从`harmony-install-dir/apps/usb/device/msd_multiple_luns`文件夹中的msd_multiple_luns演示复制。

右键单击*Source Files > app > Add Existing Item…*（源文件 > 应用程序 > 添加现有项目...），然后将`diskImage.c`文件添加到项目。

完整的项目源代码随MPLAB Harmony V2.02（及更高版本）安装程序一起提供。项目可以在MPLAB Harmony安装程序中找到，方法是导航到`harmony-install-dir/apps/usb/device/msd_multiple_luns`文件夹。

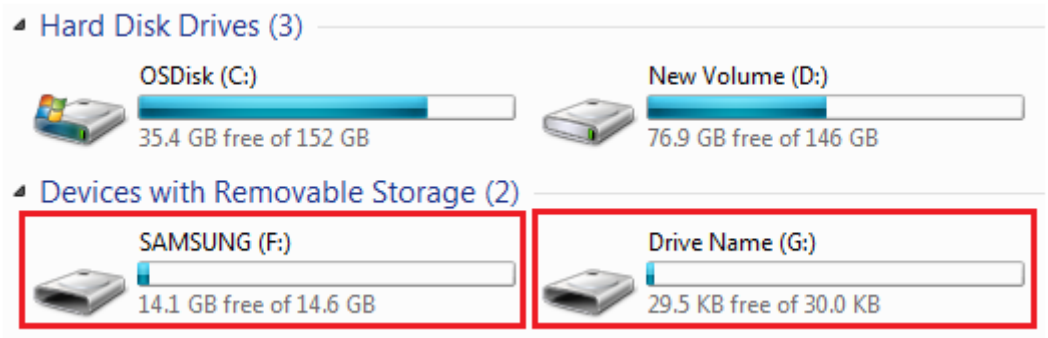
步骤8：编译并运行源代码

在添加必要的源代码之后，编译并烧写PIC32MZ Curiosity开发板。

将microSD click卡（插有SD卡）插入mikro总线插座2（J10）。使用USB调试连接器J3为电路板上电。使用USB micro电缆将USB设备上的J12连接到USB主机PC。

等待PC将USB设备枚举为大容量存储设备。枚举完成后，SD卡和NVM应该在主机PC上显示为两个独立的逻辑驱动器，如图32所示。

图32：　主机PC上的逻辑驱动器



USB协议栈初始化数据

system_init.c文件包含USB协议栈的初始化数据。

USB设备层初始化数据

例3所示为USB设备层初始化数据。

USB_DEVICE_INIT结构包含USB设备层初始化数据。moduleInit允许将驱动器初始化为请求的电源模式。registeredFuncCount指示注册到此USB设备层实例的函数驱动程序的数量。此应用程序将只有一个函数驱动程序，即MSD函数驱动程序。registeredFunctions指向此设备层实例的函数驱动程序表。函数驱动程序表包含MSD函数驱动程序的初始化数据。usbMasterDescriptor指向USB描述符结构，该结构反过来指向设备描述符、配置描述符和字符串描述符。usbDriverInterface指向USB驱动程序函数。凭借这些函数，USB设备层可对USB总线进行结构化访问。

例3： USB设备层初始化数据

```
const USB_DEVICE_INIT usbDevInitData =
{
    /* 系统模块初始化 */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

    /* 注册到此USB设备层实例的
       函数驱动程序的数量 */
    .registeredFuncCount = 1,

    /* 注册到此USB设备层实例的函数驱动程序表 */
    .registeredFunctions = (USB_DEVICE_FUNCTION_REGISTRATION_TABLE*)funcRegistrationTable,

    /* 指向USB描述符结构的指针 */
    .usbMasterDescriptor = (USB_DEVICE_MASTER_DESCRIPTOR*)&usbMasterDescriptor,

    /* USB设备速度 */
    .deviceSpeed = USB_SPEED_HIGH,

    /* 此设备层实例使用的USB驱动程序的索引 */
    .driverIndex = DRV_USBHS_INDEX_0,

    /* 指向USB驱动程序函数的指针。*/
    .usbDriverInterface = DRV_USBHS_DEVICE_INTERFACE,

};
```

USB 设备函数注册表

例4所示为USB设备函数注册表的实例。

funcRegistrationTable包含函数驱动程序列表，允许使用USB设备层注册函数驱动程序。USB设备层初始化数据包含一个指向funcRegistrationTable的指针。

configurationValue、interfaceNumber、速度和 numberOfInterfaces 均基于 MHC 选择。funcDriverIndex用作MSD函数驱动程序实例数据结构（gUSBDeviceMSDInstance）的索引。由于USB MSD函数驱动程序只有一个实例，因此funcDriverIndex设置为0。

注： 尽管有两个介质（SD卡和NVM），但只需要MSD函数驱动程序的一个实例。这两个介质可以基于USB主机在CBW数据包中传送的LUN编号进行访问。

driver指向MSD函数驱动程序暴露给USB设备层的接口。USB设备层将在发生适当的事件时调用这些接口函数。例4所示为USB设备层预期的函数驱动程序接口。

最后，funcDriverInit指向函数驱动程序初始化数据，当USB设备层从USB主机接收到“设置配置”控制命令时，MSD函数驱动程序可以使用该初始化数据自行初始化。

例4： USB设备函数注册表

```
const USB_DEVICE_FUNCTION_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    /* 函数1 */
    {
        .configurationValue = 1,     /* 配置值 */
        .interfaceNumber = 0,        /* 此函数的第一个接口编号 */
        .speed = USB_SPEED_HIGH|USB_SPEED_FULL,     /* 速度函数 */
        .numberOfInterfaces = 1,     /* 接口数量 */
        .funcDriverIndex = 0,  /* MSD函数驱动程序的索引 */
        .driver = (void*)USB_DEVICE_MSD_FUNCTION_DRIVER,     /* 暴露给设备层的USB MSD
                                                                函数数据 */
        .funcDriverInit = (void*)&msdInit0,    /* 函数驱动程序初始化数据 */
    },
};
```

USB 设备函数驱动程序接口

例5所示为USB函数驱动程序接口结构。当从USB主机收到“设置配置”控制请求时，USB 设备层将调用intilizeByDescriptor回调函数。MSD函数驱动程序必须自行初始化并使能批量端点。pDescriptor参数将指向对应于“设置配置”请求中收到的配置值的配置、接口和端点描述符。函数驱动程序将基于pDescriptor指向的接口和端点描述符来初始化批量端点。

当设备层检测到设备断开、配置更改或USB总线复位时，将调用 deInitialize 回调函数。通过调用controlTransferNotification，函数驱动程序可处理针对MSD接口的标准和类特定的控制请求。针对批量端点的标准端点请求由设备层处理。

如果当前的设备速度和当前配置与函数驱动程序注册表中提到的速度和配置值匹配，并且函数驱动程序处于已配置状态，则USB设备层将调用tasks指向的MSD函数驱动程序任务函数。例6所示为MSD函数驱动程序的一个实例。

例5： USB函数驱动程序接口结构

```
typedef struct
{
    /* 当设备层接收到“设置配置”事件时，
       将调用Initialize。设备层将针对每个描述符初始化
       函数驱动程序。函数驱动程序必须基于描述符类型
       自行初始化。*/

    void (*initializeByDescriptor)
    (
        SYS_MODULE_INDEX funcDriverIndex,
        USB_DEVICE_HANDLE usbDeviceHandle,
        void* funcDriverInit,
        uint8_t interfaceNumber,
        uint8_t alternateSetting,
        uint8_t descriptorType,
        uint8_t * pDescriptor
    );

    /* 当设备层检测到设备断开、
       配置更改或USB总线复位时，将调用Deinitialize。*/

    void (*deInitialize)(SYS_MODULE_INDEX funcDriverIndex);

    /* 当存在接口特定的建立数据包请求时，
       设备层将调用此函数 */

    void (*controlTransferNotification)
    (
        SYS_MODULE_INDEX index,
        USB_DEVICE_EVENT controlEvent,
        USB_SETUP_PACKET * controlEventData
    );

    /* 函数驱动程序任务 */
    void (*tasks)(SYS_MODULE_INDEX funcDriverIndex);

    /* 全局初始化函数驱动程序 */
    void (*globalInitialize)(void);

} USB_DEVICE_FUNCTION_DRIVER;
```

例6：　　MSD函数驱动程序实例

```
USB_DEVICE_FUNCTION_DRIVER msdFunctionDriver =
{
    /* MSD初始化函数 */
    .initializeByDescriptor = _USB_DEVICE_MSD_InitializeByDescriptorType ,

    /* MSD取消初始化函数 */
    .deInitialize = _USB_DEVICE_MSD_Deinitialization ,

    /* MSD建立数据包处理程序 */
    .controlTransferNotification = _USB_DEVICE_MSD_ControlTransferHandler ,

    /* MSD任务函数 */
    .tasks = _USB_DEVICE_MSD_Tasks
};
```

USB 设备函数驱动程序初始化数据

例7所示为MSD函数驱动程序的初始化数据。

USB_DEVICE_MSD_INIT包含MSD函数驱动程序的初始化数据：

- numberOfLogicalUnits——这表示在此MSD函数驱动程序实例中的逻辑单元数量，该值设置为2。
- msdCBW——指向CBW数据结构。MSD函数驱动程序使用该指针从主机接收CBW。对于具有数据高速缓存的PIC32MZ器件（如本应用中使用的PIC32MZ），此数组应放入相干存储器中，并且应与16字节边界对齐。
- msdCSW——指向CSW数据结构。MSD函数驱动程序使用该指针将CSW发送到主机。对于PIC32MZ器件，此数组应放入相干存储器并与16字节边界对齐。
- mediaInit——指向MSD介质驱动程序初始化数据结构。每个LUN应该有一个结构。

例7： MSD函数驱动程序初始化数据

```
const USB_DEVICE_MSD_INIT msdInit0 =
{
    .numberOfLogicalUnits = 2,
    .msdCBW = &msdCBW0,
    .msdCSW = &msdCSW0,
    .mediaInit = &msdMediaInit0[0]
};
```

介质驱动程序初始化数据

例8所示为MSD介质驱动程序初始化数据的结构。

例8： MSD介质驱动程序初始化数据结构

```
typedef struct
{
    /* 为此LUN打开的介质驱动程序的实例索引 */
    SYS_MODULE_INDEX instanceIndex;

    /* 此LUN的扇区大小。如果为0，则表示可从介质几何对象
       获取扇区大小。*/
    uint32_t sectorSize;

    /* 指向一个字节缓冲区的指针，该缓冲区的大小就是此介质上扇区的大小。
     *对于PIC32MZ器件，此缓冲区应是相干的并且
     * 与16字节边界对齐 */
    uint8_t * sectorBuffer;

    /* 在此介质的扇区大小小于
     * 写块大小的情况下，应将写块大小的字节缓冲区提供给
     * 函数驱动程序。例如，PIC32MZ NVM闪存驱动程序有一个
     * 大小为4096字节的闪存程序存储器行，超过
     * 标准的512字节扇区。在这种情况下，应用程序应将此
     * 指针设置为指向4096字节缓冲区 */
    uint8_t * blockBuffer;

    /* 此介质上块0的起始地址。如果不为0，则此地址将
       传送到blockStartAddressSet函数。应将其设置为
       介质上的起始存储地址。*/
    void * block0StartAddress;

    /* 指向此LUN的SCSI查询响应的指针 */
    SCSI_INQUIRY_RESPONSE inquiryResponse;

    /* 指向介质驱动程序函数的函数指针 */
    USB_DEVICE_MSD_MEDIA_FUNCTIONS mediaFunctions;

} USB_DEVICE_MSD_MEDIA_INIT_DATA;
```

介质初始化数据包括以下内容：

- instanceIndex——指向为此LUN打开的介质驱动程序的索引。对于LUN0，将其设置为DRV_SDCARD_INDEX_0，即SD卡驱动程序的索引；对于LUN1，将其设置为DRV_NVM_INDEX_0，即NVM驱动程序的索引
- sectorSize——这是此介质上的一个扇区的大小，两个介质均设置为512。这是因为从主机角度来看，介质是以512字节的逻辑块来组织的。
- sectorBuffer——指向一个缓冲区，其大小为512 x 为在介质读操作期间缓冲数据而要缓冲的Max number of sectors。在MHC中，要缓冲的Max number of sectors在USB函数驱动程序配置中设置为1。
- blockBuffer——MSD介质驱动程序使用此缓冲区对块大小大于扇区大小（512字节）的介质执行读-修改-写操作。对于LUN0，将其设置为0（NULL）。这是因为SD卡的块大小为512字节。对于LUN1，NVM写块（行）的大小为2048字节，因此将缓冲区大小设置为2048字节。对于NVM，将在此缓冲区中读取并修改2048字节的完整块，然后再将修改后的块写回到介质。
- inquiryResponse——指向LUN的SCSI查询响应。

mediaFunctions指向此LUN的介质驱动程序函数。MSD函数驱动程序希望介质驱动程序符合USB_DEVICE_MSD_MEDIA_FUNCTIONS接口。USB_DEVICE_MSD_MEDIA_FUNCTIONS 结构包含指向介质驱动程序函数（例如isAttached、open、close、geometryGet、blockRead、blockWrite、isWriteProtected 和blockEventHandlerSet）的函数指针。对于LUN0，mediaFunctions指向SD卡驱动程序函数；对于LUN1，则指向NVM驱动程序函数：

- isAttached——MSD函数驱动程序需要了解介质是否连接以及是否可使用时，将调用此函数。
- open——MSD函数驱动程序调用此函数来获取句柄及访问介质驱动程序的指定实例的功能。
- close——当函数驱动程序由于设备断开或配置更改而被取消初始化时，MSD函数驱动程序将调用此函数。
- geometryGet——MSD函数驱动程序需要了解介质的存储容量时，将调用此函数。
- blockRead——MSD函数驱动程序调用此函数从介质读取数据块。
- blockWrite —— MSD函数驱动程序调用此函数向介质写入数据块。
- isWriteProtected——MSD函数驱动程序调用此函数来确定介质是否受写保护。
- blockEventHandlerSet——MSD函数驱动程序调用此函数来通过介质驱动程序注册块事件回调函数。此事件回调函数将在块相关操作完成时调用。

例9所示为两个逻辑单元的介质初始化数据。

例9：　　介质初始化数据

```
USB_DEVICE_MSD_MEDIA_INIT_DATA msdMediaInit0[2] =
{
        {
        DRV_SDCARD_INDEX_0,
        512,
        sectorBuffer,
        NULL,
        0,
        {
            0x00,// 外设已连接，直接访问块设备
            0x80,   // 可移除
            0x04,// 版本 = 00=> 不符合任何标准，4=> SPC-2
            0x02,// 响应采用SPC-2指定的格式
            0x1F,// 附加长度
            0x00,// sccs等
            0x00,// bque=1且cmdque=0，表示简单队列00已过时，
                 // 但对于其他设备，我们使用00
            0x00,// 00已过时，基本任务队列为0x80
            {
                'M','i','c','r','o','c','h','p'
            },
            {
                'M','a','s','s',' ','S','t','o','r','a','g','e',' ',' ',' ',' '
            },
            {
                '0','0','0','1'
            }
        },
        {
            DRV_SDCARD_IsAttached,
            DRV_SDCARD_Open,
            DRV_SDCARD_Close,
            DRV_SDCARD_GeometryGet,
            DRV_SDCARD_Read,
            DRV_SDCARD_Write,
            DRV_SDCARD_IsWriteProtected,
            DRV_SDCARD_EventHandlerSet,
            NULL
        }
    },
```

例9：　介质初始化数据（续）

```
    {
        DRV_NVM_INDEX_0,
        512,
        sectorBuffer,
        flashRowBackupBuffer,
        (void *)diskImage,
        {
            0x00,// 外设已连接，直接访问块设备
            0x80,   // 可移除
            0x04,// 版本 = 00=> 不符合任何标准，4=> SPC-2
            0x02,// 响应采用SPC-2指定的格式
            0x1F,// 附加长度
            0x00,// sccs等
            0x00,// bque=1且cmdque=0，表示简单队列00已过时，
                    // 但对于其他设备，我们使用00
            0x00,// 00已过时，基本任务队列为0x80
            {
                'M','i','c','r','o','c','h','p'
            },
            {
                'M','a','s','s',' ','S','t','o','r','a','g','e',' ',' ',' ',' '
            },
            {
                '0','0','0','1'
            }
        },
        {
            DRV_NVM_IsAttached,
            DRV_NVM_Open,
            DRV_NVM_Close,
            DRV_NVM_GeometryGet,
            DRV_NVM_Read,
            DRV_NVM_EraseWrite,
            DRV_NVM_IsWriteProtected,
            DRV_NVM_EventHandlerSet,
            NULL
        }
    },
};
```

USB 驱动程序初始化数据

例 10 所示为 USB 驱动程序的初始化数据结构。

operationMode 设置为设备操作模式。
interruptSource 包含 USB 中断源编号。
interruptSourceUSBDma 包含对应于 USB DMA 的中断编号。驱动程序将这些编号作为参数传送给外设库函数以控制（允许/禁止）中断源。

PIC32MZ 器件集成有 8 通道 DMA。DMA 通道（如果提供）用于将从端点 FIFO 接收到的数据移动到端点的数据缓冲区。在传输过程中，DMA 会将数据从端点的数据缓冲区移动到端点 FIFO。

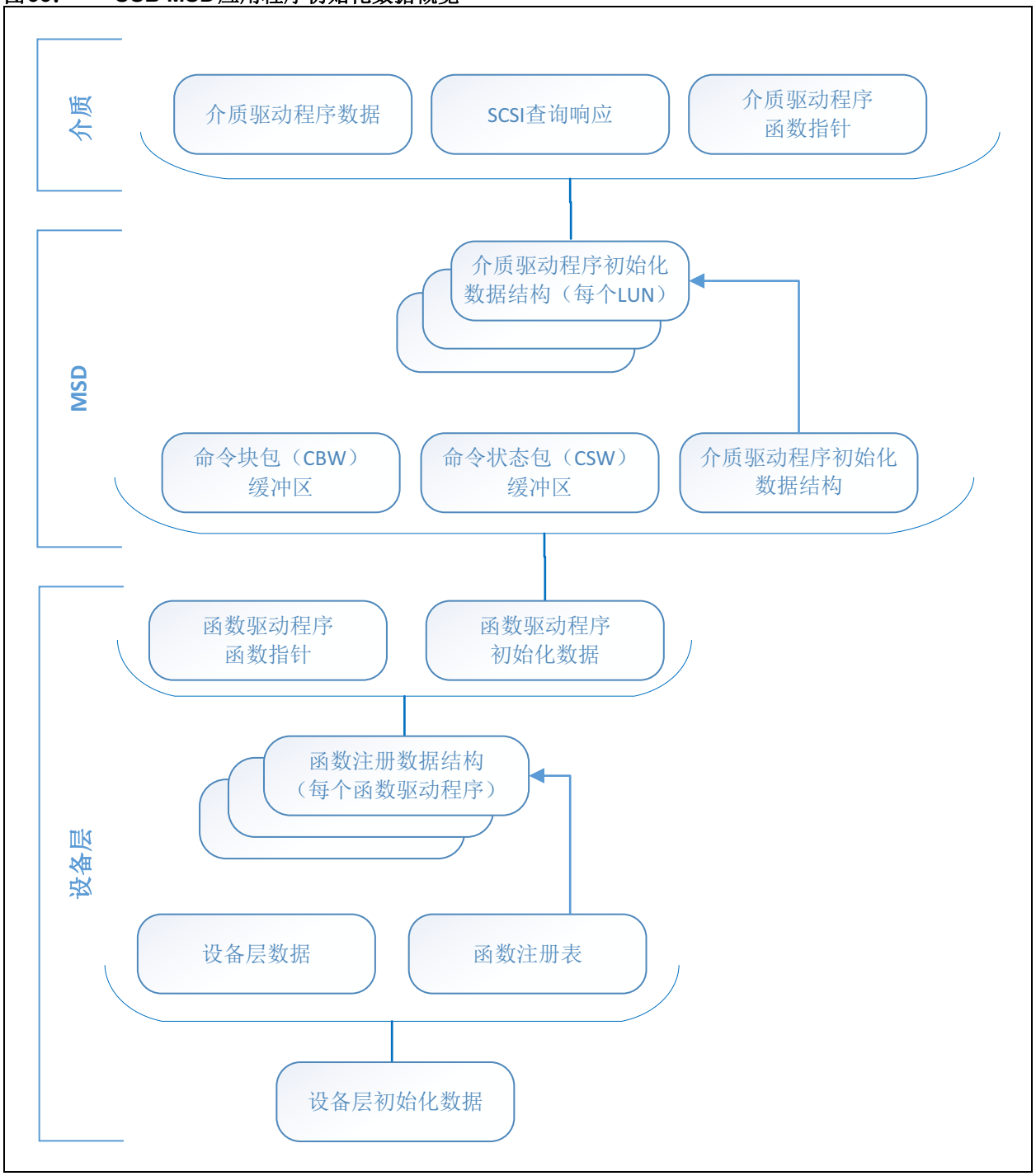
图 33 说明了介质驱动程序如何插入 MSD 函数驱动程序，以及 MSD 函数驱动程序如何插入设备层。

system_init.c 文件还包含高速 USB 和全速 USB 的 USB 描述符。制造商字符串、产品字符串和序列号字符串的字符串描述符也基于 MHC 选择进行填充。

例 10： USB 驱动程序初始化数据结构

```
const DRV_USBHS_INIT drvUSBInit =
{
    /* USB模块的中断源 */
    .interruptSource = INT_SOURCE_USB_1,

    /* USB模块的中断源 */
    .interruptSourceUSBDma = INT_SOURCE_USB_1_DMA,

    /* 系统模块初始化 */
    .moduleInit = {SYS_MODULE_POWER_RUN_FULL},

    .operationMode = DRV_USBHS_OPMODE_DEVICE,

    .operationSpeed = USB_SPEED_HIGH,

    /* 在空闲模式下停止 */
    .stopInIdle = false,

    /* 在休眠模式下暂停 */
    .suspendInSleep = false,

    /* 标识外设（PLIB级）ID */
    .usbID = USBHS_ID_0,
};
```

图33： USB MSD应用程序初始化数据概览



USB海量存储设备任务函数

图34所示为运行设备驱动程序任务、系统服务和应用程序任务的系统任务函数。

图35所示为USB MSD任务函数。USB MSD任务函数（_USB_DEVICE_MSD_Tasks）在USB设备任务（USB_DEVICE_Tasks）的上下文中运行。USB设备任务反过来从系统任务函数（SYS_Tasks）运行。

USB MSD任务可实现处理CBW命令、数据阶段和CSW的MSD状态机。

图34： 系统任务函数

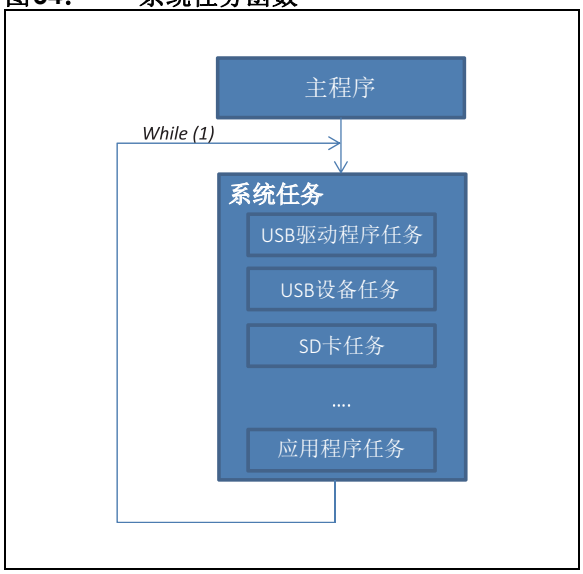


图35： USB MSD任务函数

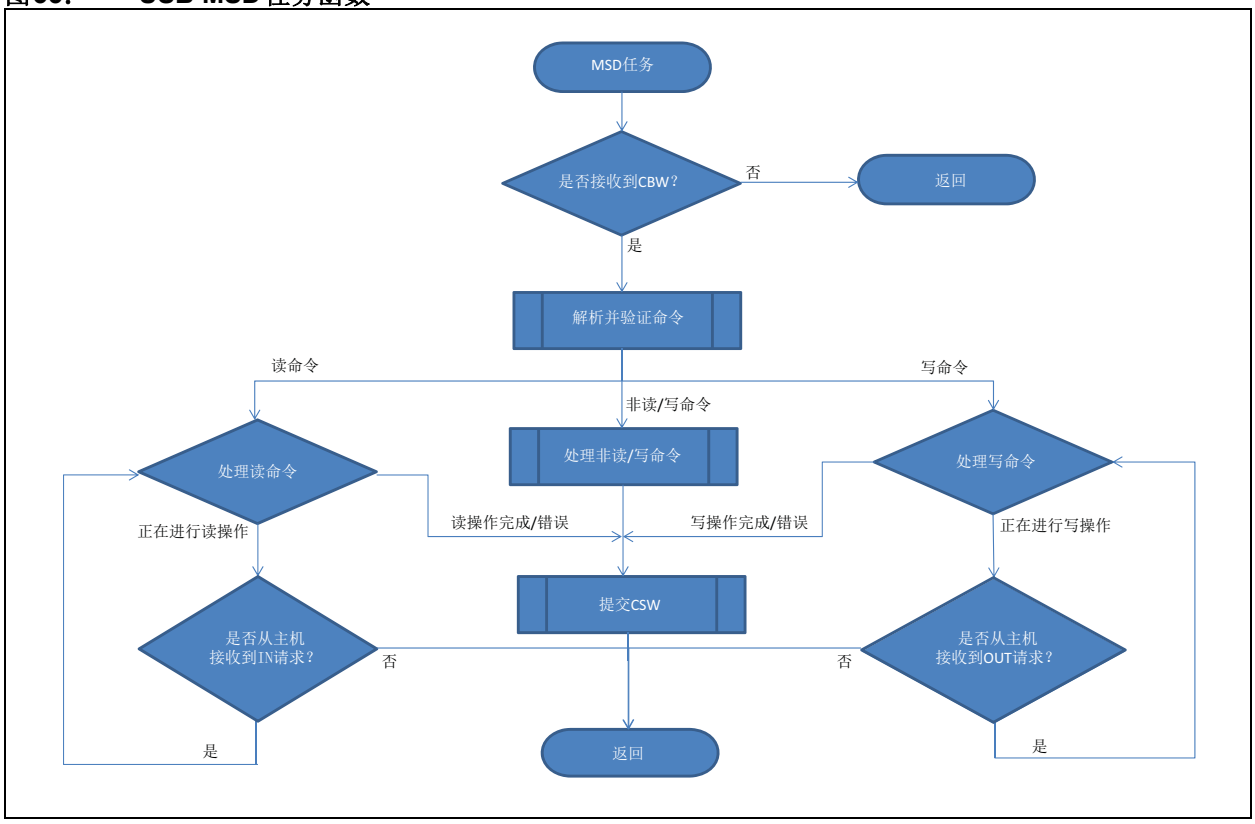


表15列出了MPLAB Harmony安装程序中供应用程序项目使用的重要源文件和头文件的详细信息。当选择库或驱动程序时，所有这些文件都由MHC自动添加到MPLAB X IDE项目中。

表15： 应用程序项目引用的重要源文件和头文件

源文件名称	说明
framework/usb/src/dynamic/usb_device.c	此文件实现了USB设备层接口。
framework/usb/src/dynamic/usb_device_msd.c	此文件实现了MSD函数驱动程序接口。
framework/driver/usb/usbhs/src/dynamic/drv_usbhs.c	此文件实现了系统模块访问的函数，这些函数允许系统模块初始化/取消初始化和维护驱动程序。此外，它还包含允许打开、关闭和实现其他一般驱动程序操作的客户端函数。
framework/driver/usb/usbhs/src/dynamic/drv_usbhs_device.c	此文件实现了允许USB设备协议栈执行USB设备模式特定驱动程序操作的函数。它实现了USB设备层预期的USB驱动程序接口。

结论

开发USB海量存储设备应用程序需要了解各种标准、协议和中间件。用户可以凭借MPLAB Harmony USB设备协议栈框架、附带的MSD固件和介质驱动程序来设计解决方案，而无需管理基础标准或协议。MPLAB Harmony为用户提供了一款用于PIC32单片机的灵活、紧凑的全集成固件开发平台。

本应用笔记介绍了USB海量存储类的基本概念，并概述了MPLAB Harmony USB设备协议栈固件架构。其中涵盖了创建USB海量存储设备应用程序所涉及的术语、协议和标准，还介绍了用户如何使用 MPLAB Harmony配置器（MHC）实用程序来创建和配置支持多个逻辑单元的MSD应用程序。MPLAB Harmony集成软件框架包含各种MSD解决方案的更多示例和演示，可从Microchip网站下载（见 **“参考资料”**）。

参考资料

- AN1142《嵌入式主机上的USB海量存储设备类》（http://www.microchip.com）
- USB海量存储——设计和编程设备与嵌入式主机——Jan Axelson（ISBN-10:1931448043；ISBN-13:978-1931448048）
- AT91 USB海量存储设备驱动程序实现（http://www.microchip.com）
- T10技术委员会网站（http://www.t10.org/drafts.htm）
- 通用串行总线网站（http://www.usb.org）
- help_harmony.pdf，随MPLAB Harmony软件框架安装程序一起提供（http://www.microchip.com/mplab/mplab-harmony）
- 带多个驱动器的USB MSD演示——独立演示示例（https://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=DM320104）
- MikroElektronika的microSD Click板（https://shop.mikroe.com/click/storage/microsd）

注：

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了 《数字器件千年版权法案 （Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2009 认证。 Microchip 的 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品严格遵守公司的质量体系流程。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

ISBN：978-1-5224-2509-0

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://www.microchip.com/support
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 1-678-957-9614
Fax: 1-678-957-1455

奥斯汀 Austin, TX
Tel: 1-512-257-3370

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Novi, MI
Tel: 1-248-848-4000

休斯敦 Houston, TX
Tel: 1-281-894-5983

印第安纳波利斯 Indianapolis
Noblesville, IN
Tel: 1-317-773-8323
Fax: 1-317-773-5453
Tel: 1-317-536-2380

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608
Tel: 1-951-273-7800

罗利 Raleigh, NC
Tel: 1-919-844-7510

纽约 New York, NY
Tel: 1-631-435-6000

圣何塞 San Jose, CA
Tel: 1-408-735-9110
Tel: 1-408-436-4270

加拿大多伦多 Toronto
Tel: 1-905-695-1980
Fax: 1-905-695-2078

亚太地区

中国 - 北京
Tel: 86-10-8569-7000

中国 - 成都
Tel: 86-28-8665-5511

中国 - 重庆
Tel: 86-23-8980-9588

中国 - 东莞
Tel: 86-769-8702-9880

中国 - 广州
Tel: 86-20-8755-8029

中国 - 杭州
Tel: 86-571-8792-8115

中国 - 南京
Tel: 86-25-8473-2460

中国 - 青岛
Tel: 86-532-8502-7355

中国 - 上海
Tel: 86-21-3326-8000

中国 - 沈阳
Tel: 86-24-2334-2829

中国 - 深圳
Tel: 86-755-8864-2200

中国 - 苏州
Tel: 86-186-6233-1526

中国 - 武汉
Tel: 86-27-5980-5300

中国 - 西安
Tel: 86-29-8833-7252

中国 - 厦门
Tel: 86-592-238-8138

中国 - 香港特别行政区
Tel: 852-2943-5100

中国 - 珠海
Tel: 86-756-321-0040

台湾地区 - 高雄
Tel: 886-7-213-7830

台湾地区 - 台北
Tel: 886-2-2508-8600

台湾地区 - 新竹
Tel: 886-3-577-8366

亚太地区

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733

印度 India - Bangalore
Tel: 91-80-3090-4444

印度 India - New Delhi
Tel: 91-11-4160-8631

印度 India - Pune
Tel: 91-20-4121-0141

日本 Japan - Osaka
Tel: 81-6-6152-7160

日本 Japan - Tokyo
Tel: 81-3-6880-3770

韩国 Korea - Daegu
Tel: 82-53-744-4301

韩国 Korea - Seoul
Tel: 82-2-554-7200

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870

菲律宾 Philippines - Manila
Tel: 63-2-634-9065

新加坡 Singapore
Tel: 65-6334-8870

泰国 Thailand - Bangkok
Tel: 66-2-694-1351

越南 Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

丹麦 Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

芬兰 Finland - Espoo
Tel: 358-9-4520-820

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Garching
Tel: 49-8931-9700

德国 Germany - Haan
Tel: 49-2129-3766400

德国 Germany - Heilbronn
Tel: 49-7131-67-3636

德国 Germany - Karlsruhe
Tel: 49-721-625370

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

德国 Germany - Rosenheim
Tel: 49-8031-354-560

以色列 Israel - Ra'anana
Tel: 972-9-744-7705

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

意大利 Italy - Padova
Tel: 39-049-7625286

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

挪威 Norway - Trondheim
Tel: 47-7289-7561

波兰 Poland - Warsaw
Tel: 48-22-3325737

罗马尼亚 Romania - Bucharest
Tel: 40-21-407-87-50

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

瑞典 Sweden - Gothenberg
Tel: 46-31-704-60-40

瑞典 Sweden - Stockholm
Tel: 46-8-5090-4654

英国 UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820