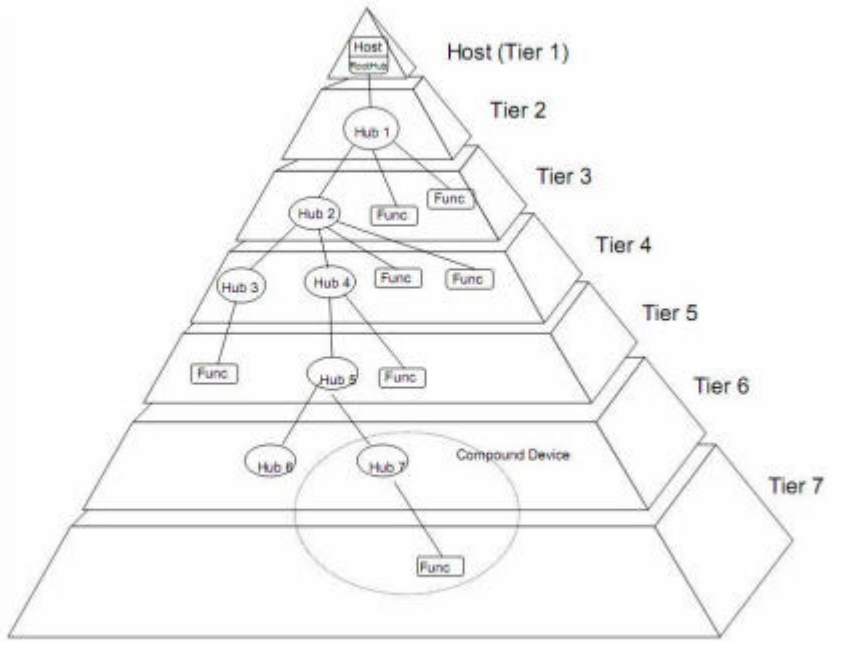
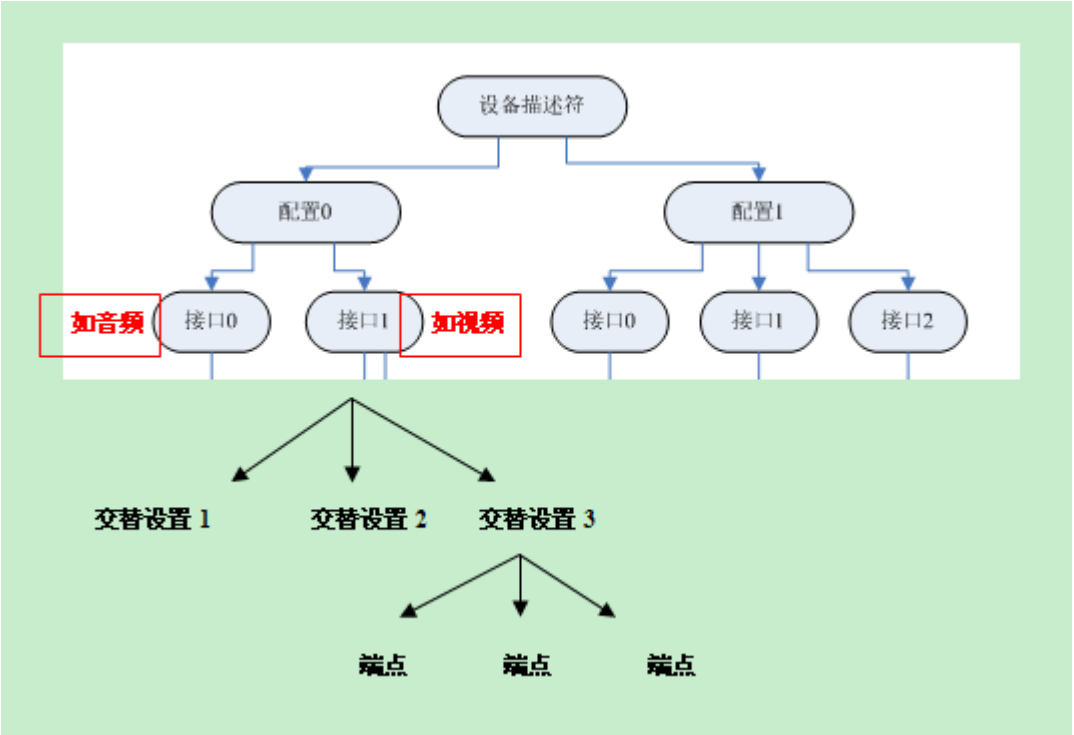
一、USB 协议



以 HOST-ROOT HUB 为起点，最多支持 7 层（Tier），也就是说任何一个 USB 系统中最多可以允许 5 个 USB HUB 级联，ROOT HUB 是一个特殊的 USB HUB，它集成 址。

USB 采用轮询的广播机制传输数据，所有的传输都由主机发起，任何时刻整个 USB 体系内仅允许一个数据包的传输，即不同物理传输线上看到的数据包都是同一被广播的数据包。

1. USB--设备、配置、接口、端点



端点（Endpoint）：是 USB 设备中的可以进行数据收发的最小单元，支持单向或者双向的数据传输。端点 0 支持双向的。

设备支持端点的数量是有限制的，除默认端点外低速设备最多支持 2 组端点（2 个输入，2 个输出），高速和全速设备最多支持 15 组端点。

所谓单向传输，并不是说该传输只支持一个方向的传输，而是指在某个端点上该传输
仅支持一个方向，或输出，或输入。如果需要在两个方向上进行某种单向传输，需要占用
两个端点，分别配置成不同的方向，可以拥有相同的端点编号。

设备通常有一个或多个配置；但同一时刻只有一个有效，配置通常有一个或多个接口，如音频接口和视频接口，多个可以同时有效，接口通常有一个或多个交替设置；每个交替设置都是端点的集合

2. USB 的描述符

包括：

1、 usb 标准设备描述符
2、 usb 集线器描述符
3、 HID（人机接口）设备描述符

一、USB 标准设备描述符

1、设备描述符、
2、配置描述符、
3、接口描述符、
4、端点描述符、
5、字符串描述符、
6、设备限定描述符和其他速率配置描述符，
其中设备限定描述符和其他速率配置描述符用于高速 USB 设备

设备描述符定义

表 5.2　　设备描述符格式

字　段　名	长度/字节	地址偏移量	说　　明
bLength	1	0	描述符的长度（12H 字节）
bDescriptorType	1	1	描述符的类型：设备描述符=01H
bcdUSB	2	2	USB 规范版本号（采用 BCD 码）
bDeviceClass	1	4	类代码
bDeviceSubClass	1	5	子类代码
bDeviceProtocol	1	6	协议代码
bMaxPackerSize0	1	7	端点 0 支持最大数据包的长度
idVendor	2	8	供应商 ID
idProduct	2	10	产品 ID
bcdDevice	2	12	设备版本号（采用 BCD 码）
iManufacturer	1	14	供应商字符串描述符的索引值
iProduct	1	15	产品字符串描述符的索引值

续表

字　段　名	长度/字节	地址偏移量	说　　明
iSerialNumber	1	16	设备序列号字符串描述符的索引值
bNumConfigurations	1	17	所支持的配置数

USB 设备描述符由 14 个字段组成，总长度固定为 18 个字节，其格式如表 5.2 所示。

配置描述符定义

表 5.3　　配置描述符格式

字　段　名	长度/字节	地址偏移量	说　　明
bLength	1	0	描述符的长度（09H 字节）
bDescriptorType	1	1	描述符的类型：配置描述符=02H
wTotalLength	2	2	配置信息的总长度
bNumInterfaces	1	4	该配置所支持的接口数
bConfigurationValue	1	5	配置值
iConfiguration	1	6	字符串描述符的索引值
bmAttributes	1	7	配置特性
bMaxPower	1	8	所需要的最大总线电流（2mA 为单位）

字符串描述符

表 5.4　　字符串描述符格式

字　段　名	长度/字节	地址偏移量	说　　明
bLength	1	0	描述符的长度（N+2 字节）
bDescriptorType	1	1	描述符的类型：字符串描述符=03H
bString	N	2	字符串

接口描述符

表 5.5　　接口描述符格式

字　段　名	长度/字节	地址偏移量	说　　明
bLength	1	0	描述符的长度（09H 字节）
bDescriptorType	1	1	描述符的类型：接口描述符=04H
bInterfaceNumber	1	2	接口号
bAlterateSetting	1	3	可替换设置值
bNumEndpoints	1	4	端点 0 以外的端点数
bInterfaceClass	1	5	类代码
bInterfaceSubClass	1	6	子类代码
bInterfaceProtocol	1	7	协议代码
nterface	1	8	字符串描述符的索引值

端点描述符

表 5.6　　端点描述符格式

字　段　名	长度/字节	地址偏移量	说　　明
Length	1	0	描述符的长度（07H 字节）
DescriptorType	1	1	描述符的类型：端点描述符=05H
EndpointAddress	1	2	端点号、传输方向
mAttributes	1	3	端点特性
vMaxPacketSize	2	4	最大数据包长度
Interval	1	6	访问间隔

设备限定描述符

表 5.7　　设备限定描述符格式

字　段　名	长度/字节	地址偏移量	说　　明
bLength	1	0	描述符的长度（0AH 字节）
bDescriptorType	1	1	描述符的类型：设备限定描述符=06H
bcdUSB	2	2	USB 规范版本号（采用 BCD 码）
bDeviceClass	1	4	类代码
bDeviceSubClass	1	5	子类代码
bDeviceProtocol	1	6	协议代码

续表

字　段　名	长度/字节	地址偏移量	说　　明
bMaxPackerSize0	1	7	端点 0 所支持最大数据包的长度
bNumConfigurations	1	8	所支持的配置数
bReserved	1	9	保留

其他速率配置描述符

表 5.8　　其他速率配置描述符格式

字　段　名	长度/字节	地址偏移量	说　　明
bLength	1	0	描述符的长度（09H 字节）
bDescriptorType	1	1	描述符的类型：其他速率配置描述为 07H
wTotalLength	2	2	配置信息的总长度
bNumInterfaces	1	4	所支持的接口数
bConfigurationValue	1	5	配置值
iConfiguration	1	6	字符串描述符的索引值
bmAttributes	1	7	配置特性
bMaxPower	1	8	所需要的最大 USB 总线电流（2mA 为单位）

二、USB 数据传输

USB 事务处理：是 USB 主机和设备数据传输的基本单位。

一个完整的 USB 事务处理包含 3 个阶段

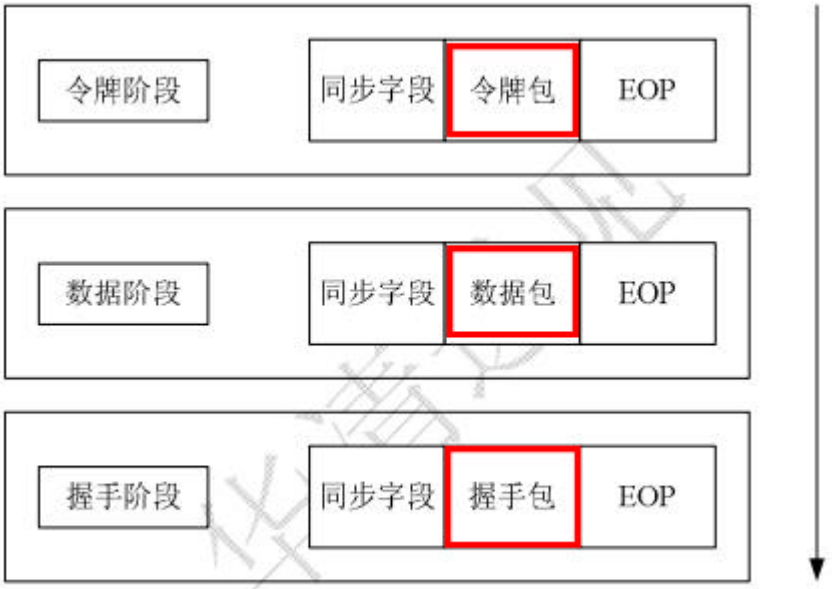


图 3.1　典型的事务处理过程

· 令牌阶段：其中定义了本次传输的类型，用于表征事务处理的开始。令牌阶段由同步字段、令牌包和 EOP 构成，这是所有的 USB 事务处理必须包含的阶段。

· 数据阶段：其中包含了本次传输的数据。其数据大小取决端点和传输类型，最大的数据量为 1024 字节。数据阶段由同步字段、数据包和 EOP 构成。

· 握手阶段： 数据的接收方向发送方报告此次数据传输是否成功， 握手阶段由同步字段、握手包和 EOP 构成

帧的起始由一个特定的包（SOF 包）表示，帧尾为 EOF。EOF 不是一个包，而是一种电平状态，EOF 期间不允许有数据传。

可以根据令牌包的类型将 USB 事务处理分为 7 种：

- IN 事务处理；
- OUT 事务处理；
- SETUP 事务处理；
- PING 事务处理；
- SOF 事务处理；
- SPLIT 事务处理；
- PRE 事务处理。

1. PING 事务处理

PIING 事务处理主要应用于高速数据传输中。PING 事务处理只包含令牌包和握手包阶段，而不包含数据阶段。

PIING 事务处理的操作步骤如下：

（1）USB 主机向 USB 设备发送 PING 令牌包，表示一个 PING 事务的开始。

（2）USB 正确接收到该令牌包，然后 USB 设备向 USB 主机返回各种握手包进行响应。

对于低速和全速 USB 事务处理，USB 主机采用循环重复尝试的方式，不断对 USB 执行相应的事务处理，直至 USB 设备可以处理该事务。如果数据包比较大，则这种方式很浪费 USB 总线资源。PING 事务处理便可以解决这个问题，防止高速 USB 数据传输中重复发送 OUT 事务处理数据包，从而可以提高 USB 总线的效率。

2. SOF 事务处理

SOF 事务处理比较简单，USB 主机发送一个 SOF 令牌包即可。SOF 事务处理表示了一个 USB 帧或者 USB 小帧的开始。整个事务处理过程没有数据阶段，

也不需要 USB 设备进行握手响应。 对于不同的 USB 传输速度，SOF 事务处理的时间要求不一样。

· 对于低速和全速 USB 传输，每隔 1ms 产生一个 SOF 令牌包。
· 对于高速 USB 传输，每隔 125 μs 产生一个小帧，每隔 7 个小帧，产生一个 SOF 令牌包。

3. SPLIT 事务处理

SPLIT 事务处理的目的是在 USB 高速数据传输的过程中，可以插入低速和全速 USB 数据传输，这样可以提高 USB 总线利用率。SPLIT 令牌包分为两种，因此，SPLIT 事务处理可以分为两类。

1． SSPLIT 事务处理

SSPLIT 事务处理是开始 SPLIT。一个完整的 SSPLIT 事务处理流程如图 3.22 所示。整个事务处理过程包括 3 个阶段，其操作步骤如下：
（1）USB 主机发送令牌包，这里包括 SSPLIT 令牌包和低速/全速令牌包两个令牌包。
（2）USB 设备正确接收到该令牌包，此时 USB 主机发送 DATAx 数据包。
（3）USB 设备正确接收到 DATAx 数据包，在握手阶段返回握手信息。

2． CSPLIT 事务处理

CSPLIT 事务处理是结束 SPLIT。整个事务处理过程包括两个阶段，其中 DATAx 数据包和握手包可选，其操作步骤如下：
（1）USB 主机发送令牌包，这里包括 CSPLIT 令牌包和低速/全速令牌包两个令牌包。
（2）USB 设备正确接收到该令牌包，此时 USB 主机发送 DATAx 数据包，或者 USB 设备返回握手信息

4. PRE 事务处理

PRE 事务处理比较简单，USB 主机直接发送 PRE 令牌包即可，也就是说，PRE 事务处理不需要数据包和握手包，SPLIT 事务可以在高速 USB 总线上同时运行低速和全速 USB 事务。但在低速和全速 USB 总线情况下，USB 协议中禁止低速端口，从而阻止全速 USB 事务在低速 USB 设备上的运行。

为此，USB 在开始低速数据传输前，首先需要发送 PRE 事务处理，即先导

包，PRE 事务处理将开启一个 USB 低速数据传输。 这里需要注意的是，PRE 事务处理只在 USB 主机和 USB 集线器之间进行，即只有 USB 集线器才可以响应 PRE 事务处理。在 USB 协议中，PRE 事务处理只在低速数据传输中使用，不适用于 USB 高速数据传输

三、数据包与字段

USB 采用“令牌包”、“数据包”、“握手包”的传输机制，在令牌包中指定数据包去向或者来源的设备地址和端点（Endpoint），从而保证了只有一个设备对被广播的数据包/令牌包作出响应。握手包表示了传输的成功与否。

数据包：USB 总线上数据传输的最小单位，包括 SYNC、数据及 EOP 三个部分。其中数据的格式针对不同的包有不同的格式。但都以 8 位的 PID 开始。

PID 指定了数据包的类型（共 16 种）。具体参考 usb 数据手册 P225

令牌包即指 PID 为 IN/OUT/SETUP 的包。数据包的 PID 为 DATA0，DATA1 等

1. USB 字段格式

在 USB 协议中，USB 数据传输由一系列的字段构成，这些不同功能的字段，按照特定的格式组合便可以构成不同的信息包。USB 总线通信以信息包为基本传输单元进行 USB 事务处理。

这些字段主要包括如下几种：

- 同步字段（SYNC）：用于数据通信的同步。
- 包标识字段（PID）：指明信息包类型，可以用于差错控制。
- 地址字段（ADDR）：指明 USB 总线上的一个 USB 设备。
- 端点字段（ENDP）：指明 USB 的端点。
- 帧号字段：指明当前帧的帧号。
- 数据字段：包含传输的数据。

2. 数据包

- 令牌包：此时 PID[1:0]=01B。
- 握手包：此时 PID[1:0]=10B。
- 数据包：此时 PID[1:0]=11B。
- 特殊包：此时 PID[1:0]=00B。

令牌包

在 USB 协议中，使用了 7 种令牌包，按照其定义的格式，分为如下几种：

- IN、OUT、SETUP 和 PING 令牌包，这些令牌包格式大致相同。
- SOF 令牌包。
- SPLIT 令牌包。
- PRE 令牌包。

一般来说，这些令牌包都是由 USB 主机发送的，下面分别介绍各个令牌包的定义格式。

SETUP 令牌包

包含 8 位的包标识字段 PID、
7 位的地址字段 ADDR、
4 位的端点字段 ENDP 和
5 位的循环校验字段 CRC。

在 SETUP 令牌包中，各个字段的用途如下：

- PID 字段：定义了数据传输方向为 USB 主机到 USB 设备。
- ADDR 字段：指明了 USB 设备地址。
- ENDP 字段：指明了接收数据的端点号。
- CRC 字段：用于对 ADDR 字段和 ENDP 字段进行循环冗余校验。

PING 令牌包

PING 令牌包的定义格式如图 3.11所示。图中包含8位的包标识字段 PID、7 位的地址字段 ADDR、4 位的端点字段 ENDP 和 5 位的循环校验字段 CRC。

SOF 令牌包

SOF 令牌包的定义格式如图 3.12 所示。图中包含 8 位的包标识字段 PID、11 位的帧号字段和 5 位的循环校验字段 CRC。Start-of-Frame (SOF) packets are issued by the host at a nominal rate of once every 1.00 ms +-0.0005 ms for a full-speed bus and 125 us+-0.0625 for a high-speed bu，

The SOF packet delivers two pieces of timing information. A function is informed that an SOF hasoccurred when it detects the SOF PID. Frame timing sensitive functions, that do not need to keep track of frame number (e.g., a full-speed operating hub), need only decode the SOF PID; they can ignore the frame number and its CRC. If a function needs to track frame number, it must comprehend both the PID and the time stamp. Full-speed devices that have no particular need for bus timing information may ignore the SOF packet.

数据包

· PID 字段： 用于指明不同的数据包类型。 支持的 4 种数据包为 DATA0、 DATA1、 DATA2 和 MDATA。在后面章节介绍的数据触发机制中，使用 DATA0 和 DATA1，而前面的 SPLIT
令牌事务处理则使用 DATA0、DATA1 和 MDATA。高速 USB 同步数据传输一般需要使用全
部的数据包。

· 数据字段：其中包含了传输的数据。其数据的大小根据数据传输类型和用户需要而定。
根据 USB 协议中的规定，低速 USB 数据传输，数据字段最大长度为 8 个字节；

全速 USB 数据
传输，数据字段最大长度为 1023 字节；高速 USB 数据传输，数据字段最大长度为 1024 字节。

· CRC 字段：这里使用 16 位的循环冗余校验来对数据字段进行保护

握手包

握手包的定义格式如图 3.17 所示。握手包仅由 8 位的包标识字段 PID 构成。握手包主要用于在数据传输的末尾报告本次数据传输的状态。握手包之后便是整个事务处理的结束信号 EOP。

图 3.17 握手包的定义格式

根据事务处理的状态，握手包进行不同的响应，下面具体说明主要包括的几种响应。

1．ACK 握手包

当 USB 数据传输的接收方正确接收到数据包的时候，接收方将返回 ACK 握手包。ACK 握手包表征了一次正确的数据传输，之后，才可以进行下一次事务处理。

2．NAK 握手包

NAK 握手包一般由 USB 功能设备发出。对于 IN 数据传输，表示 USB 设备没有计划向 USB 主机发送数据；对于 OUT 数据传输，表示 USB 设备无法接收 USB 主机发送的数据。

3．STALL 握手包

STALL 握手包一般由 USB 功能设备发送，表示该 USB 功能设备不支持这个请求，或者 无法发送和接收数据，STALL 握手包分为两种情况：

· 协议 STALL 握手包：在控制传输中使用， 表明该 USB 功能设备不支持这个请求协议。

· 功能 STALL 握手包：表明该 USB 功能设备的相应端点已经停止，无法完成发送数据或者接收数据的操作。

4．NYET 握手包

在 SPLIT 令牌包事务处理中，如果 USB 集线器无法正常处理 SPLIT 请求，

则 USB 集线器向 USB 主机返回 NYET 握手包。NYET 握手包一般只发生在高速数据传输过程中。

5．ERR 握手包

ERR 握手包用于表示总线数据传输发生错误,其一般发生在高速数据传输过程中。

四、USB 数据传输

1. 控制传输：

控制传输是一种可靠的双向 双向 双向 双向传输，一次控制传输可分为三个阶段。

第一阶段为从 HOST 到 Device 的 SETUP 事务传输，这个阶段指定了此次控制传输的请求类型，Device 只能返回 ACK 包，或者不返回任何包；（setup 事物）

第二阶段为数据阶段，也有些请求没有数据阶段，包含多个 IN/OUT 事物；，同样也会采用 PID 翻转的机制

第三阶段为状态阶段，通过一次 IN/OUT 传输，返回 NAK、ACK、STALL 握手包，表明请求是否成功完成。

控制传输通过控制管道在应用软件和 Device 的控制端点之间进行，控制传输过程中传输的数据是有格式定义的，USB 设备或主机可根据格式定义解析获得的数据含义。其他三种传输类型都没有格式定义。

控制传输对于最大包长度有固定的要求。对于高速设备该值为 64Byte；对于低速设备该值为 8；全速设备可以是 8 或 16 或 32 或 64。
一个端点收到/发送了一个长度小于最大包长度的包，即意味着数据传输结束。

建立阶段过后,可能会有数据阶段,这个阶段将会通过一次或多次控制传输事务，完成数据的传输。同样也会采用 PID 翻转的机制。建立阶段，Device 只能返回 ACK 包，或者不返回任何包。最后是状态阶段，通过一次方向与前一次相反的

控制事务传输来表明传输的成功与否。如果成功会返回一个长度为 0 的数据包，否则返回 NAK 或 STALL

2. 中断传输：

中断传输是一种轮询的传输方式，是一种单向的传输，HOST 通过固定的间隔对中断端点进行查询，若有数据传输或可以接收数据则返回数据或发送数据，否则返回 NAK，表示尚未准备好。

中断传输的延迟有保证，但并非实时传输，它是一种延迟有限的可靠传输，支持错误重传。

对于高速/全速/低速端点，最大包长度分别可以达到 1024/64/8 Bytes。高速中断传输不得占用超过 80%的微帧时间，全速和低速不得超过 90%。

中断端点的轮询间隔由在端点描述符中定义，全速端点的轮询间隔可以是 1~255mS，低速端点为 10~255mS，高速端点为(2interval-1)*125uS，其中 interval 取 1 到 16 之间的值。

3. 批量传输：

批量传输是一种可靠的单向传输，但延迟没有保证，它尽量利用可以利用的带宽来完成传输，适合数据量比较大的传输。

低速 USB 设备不支持批量传输，高速批量端点的最大包长度为 512，全速批量端点的最大包长度可以为 8、16、32、64。 批量传输在访问 USB 总线时，相对其他传输类型具有最低的优先级，USB HOST 总是优先安排其他类型的传输，当总线带宽有富余时才安排批量传输。 高速的批量端点必须支持 PING 操作，向主机报告端点的状态，NYET 表示否定应答，没有准备好接收下一个数据包，ACK 表示肯定应答，已经准备好接收下。

若数据量比较大，将采用多次批量事务传输来完成全部数据的传输，传输过程中数据包的 PID 按照 DATA0-DATA1-DATA0-…的方式翻转，以保证发送端和接收端的同步一个数据包。

翻转同步：发送端按照 DATA0-DATA1-DATA0-…的顺序发送数据包，只有成功的事务传输才会导致 PID 翻转，也就是说发送段只有在接收到 ACK 后才会翻转 PID，发送下一个数据包，否则会重试本次事务传输。同样，若在接收端发现接收到到的数据包不是按照此顺序翻转的，比如连续收到两个 DATA0，那么接收端认为第二个 DATA0 是前一个 DATA0 的重传。

4. 同步传输：

同步传输是一种实时的、不可靠的传输，不支持错误重发机制。它没有握手包，也不支持 PID 翻转。

主机在排定事务传输时，同步传输有最高的优先级,只有高速和全速端点支持同步传输，高速同步端点的最大包长度为 1024，低速的为 1023。 除高速高带宽同步端点外，一个微帧内仅允许一次同步事务传输，高速高带宽端点最多可以在一个微帧内进行三次同步事务传输，传输高达 3072 字节的数据。
全速同步传输不得占用超过 80%的帧时间，高速同步传输不得占用超过 90%的微帧时间。

同步端点的访问也和中断端点一样，有固定的时间间隔限制。

五、USB 设备请求概述

在 USB 协议中,主机对 USB 设备的各种配置操作是通过设备请求来实现的。USB 中定义了 11 种标准的 USB 设备请求，见表 6.1。除了这些标准设备请求，USB 协议中还允许使
用自定义的请求 CYPRESS请求码 0xa0代表固件加载请求

USB 请求	USB 请求号	功 能 描 述
GetStatus	00H	读取 USB 设备、接口或端点的状态
ClearFeature	01H	清除或禁止 USB 设备、接口或端点的某些特性
SetFeature	03H	设置或使能 USB 设备、接口或端点的某些特性
SetAddress	05H	分配 USB 设备地址
GetDescription	06H	读取设备描述符
SetDescriptor	07H	更新已有的描述符或添加新的描述符
GetConfiguration	08H	读取 USB 设备当前的配置值
SetConfiguration	09H	为 USB 设备选择一个合适的配置
GetInterface	0AH	读取 USB 指定接口的当前可替换设置值
SetInterface	0BH	为 USB 指定接口选择一个合适的可替换设置
SynchFrame	0CH	读取 USB 同步端点所指定的帧序号

USB 请求和固件反应总表：

Table 2-2. How the Firmware Handles USB Device Requests (RENUM =1)

bRequest	Name	EZ-USB Action	Firmware Response
0x00	Get Status	SUDAV Interrupt	Supply RemWU, SelfPwr or Stall Bits
0x01	Clear Feature	SUDAV Interrupt	Clear RemWU, SelfPwr or Stall Bits
0x02	(reserved)	none	Stall EP0
0x03	Set Feature	SUDAV Interrupt	Set RemWU, SelfPwr or Stall Bits
0x04	(reserved)	none	Stall EP0
0x05	Set Address	Update FNADDR Register	none
0x06	Get Descriptor	SUDAV Interrupt	Supply table data over EP0-IN
0x07	Set Descriptor	SUDAV Interrupt	Application dependent
0x08	Get Configuration	SUDAV Interrupt	Send current configuration number
0x09	Set Configuration	SUDAV Interrupt	Change current configuration
0x0A	Get Interface	SUDAV Interrupt	Supply alternate setting No. from RAM
0x0B	Set Interface	SUDAV Interrupt	Change alternate setting No.
0x0C	Sync Frame	SUDAV Interrupt	Supply a frame number over EP0-IN
Vendor Requests			
0xA0 (Firmware Load)		Upload / Download on-chip RAM	---
0xA1 to 0xAF		SUDAV Interrupt	Reserved by Cypress Semiconductor
All except 0xA0		SUDAV Interrupt	Application dependent

1. 读取状态请求 GetStatus

1．1 读取设备

GetStatus 查询两个位　远程唤醒位和自动上电位，USB 设备是否为自供电：0 表示总线供电，1 表示自供电。D1 位表示是否支持远程唤醒功能：0 表示该功能被禁止，1 表示该功能已启用

Byte	Field	Value	Meaning	Firmware Response
0	bmRequestType	**0x80**	IN, Device	固件反应 Load two bytes into EP0BUF: Byte 0: bit 0 = Self-Powered Byte 0: bit 1 = Remote Wakeup Byte 1: zero
1	bRequest	**0x00**	'Get Status'	
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**0x02**	Two bytes requested	
7	wLengthH	**0x00**		

1.2　读取接口

GetStatus 请求读取接口的格式及返回值如图 6.2 所示。其数据段返回的两个字节数据全部保留，必须置 0。

1.3　读取端点

GetStatus 请求读取端点的格式及返回值，返回端点的 STALL 状态，第四字节指定端点号如图 6.3 所示。其中，D0 位表示端点的停止特性：0 表示未被停止，1 表示该端点已被停止，

Each endpoint has a STALL bit in its EPxCS register. If this bit is set, any request to the endpoint returns a STALL handshake rather than ACK or NAK. The Get Status-Endpoint request returns the STALL state for the endpoint indicated in byte 4 of the request. Note that bit 7 of the endpoint number EP (byte 4) specifies direction (0 = OUT, 1 = IN). Endpoint zero is a CONTROL endpoint, which by USB definition is

bidirectional. Therefore, it has only one stall bit

Table 2-4. Get Status-Endpoint (Stall Bits)

Byte	Field	Value	Meaning	Firmware Response
0	bmRequestType	**0x82**	IN, Endpoint	Load two bytes into EP0BUF: Byte 0: bit 0 = Stall Bit for EP(n) Byte 1: zero
1	bRequest	**0x00**	'Get Status'	
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	**EP**	0x00-0x08: OUT0-OUT8 0x80-0x88: IN0-IN8	
5	wIndexH	0x00		
6	wLengthL	**0x02**	Two bytes requested	
7	wLengthH	**0x00**		

2. 设置特性请求 Set Feature

设置特性用来使能远程唤醒，STALL 一个端点，或者让设备进入特殊测试模式，没有数据阶段

2.1 对设备请求

特殊测试模式

This Set Feature/Device request sets the TEST_MODE fea-ture. This request puts the device into a specific test mode,and power to the device must be cycled in order to exit testmode. The EZ-USB SIE handles this request automatically,but the firmware is responsible for acknowledging the hand-shake phase

Table 2-7. Set Feature-Device (Set TEST_MODE Feature)

Byte	Field	Value	Meaning	Firmware Response
0	bmRequestType	**0x00**	OUT, Device	ACK handshake phase
1	bRequest	**0x03**	'Set Feature'	
2	wValueL	**0x02**	Feature Selector: TEST_MODE	
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0xnn	nn = specific test mode	
6	wLengthL	0x00		
7	wLengthH	0x00		

2.2 对端点请求

将相应的端点 EPXCS 的 STALL 置位

Table 2-8. Set Feature-Endpoint (Stall)

Byte	Field	Value	Meaning	Firmware Response
0	bmRequestType	**0x02**	OUT, Endpoint	Set the STALL bit for the indicated endpoint.
1	bRequest	**0x03**	'Set Feature'	
2	wValueL	**0x00**	Feature Selector: STALL	
3	wValueH	0x00		
4	wIndexL	**EP**	0x00-0x08: OUT0-OUT8 0x80-0x88: IN0-IN8	
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

清楚特性请求 ClearFeature　与上面相反

3. 获取描述符　GET Descriptor

获取设备描述符如下

Table 2-11. Get Descriptor-Device

Byte	Field	Value	Meaning	Firmware Response
0	bmRequestType	**0x80**	IN, Device	Set SUDPTR H:L to start of Device Descriptor table in RAM.
1	bRequest	**0x06**	'Get Descriptor'	
2	wValueL	0x00		
3	wValueH	**0x01**	Descriptor Type: Device	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**LenL**		
7	wLengthH	**LenH**		

剩下描述符请求的略　详见　　EZ-USB_referrence manual.pdf

4. 设置描述符请求 SetDiscriptor

2.3.5 Set Descriptor

Table 2-16. Set Descriptor-Device

Byte	Field	Value	Meaning	Firmware Response
0	bmRequestType	**0x00**	OUT, Device	Read device descriptor data over EP0BUF.
1	bRequest	**0x07**	'Set Descriptor'	
2	wValueL	0x00		
3	wValueH	**0x01**	Descriptor Type: Device	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**LenL**		
7	wLengthH	**LenH**		

5. 设置配置请求 Set Configuration

设置配置请求和设置描述符请求的区别： 固件反应

固件保存配置号后，在固件中改变配置，清除 HSNAK，结束该控制传输，然后主机发出 Set Interface 来设定配置所包含的接口

Table 2-19. Set Configuration

Byte	Field	Value	Meaning	Firmware Response
0	bmRequestType	**0x00**	OUT, Device	Read and store CFG, change configurations in firmware.
1	bRequest	**0x09**	'Set Configuration'	
2	wValueL	**CFG**	Configuration Number	
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

6. 设置接口请求　Set Interface

设置端点 所以没设置端点请求

Table 2-21. Set Interface (Actually, Set Alternate Setting #AS for Interface #IF)

Byte	Field	Value	Meaning	Firmware Response
0	bmRequestType	**0x00**	OUT, Device	Read and store byte 2 (AS) for Interface #IF, change setting for Interface #IF in firmware.
1	bRequest	**0x0B**	'Set Interface'	
2	wValueL	**AS**	Alternate Setting Number	
3	wValueH	0x00		
4	wIndexL	**IF**	Interface Number	
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

The firmware should respond to a Set Interface request by performing the following steps:

1. Perform the internal operation requested (such as adjusting a sampling rate).
2. Reset the data toggles for every endpoint in the inter-face.
3. Restore the endpoints to their default conditions, ready to send or accept data. For EP1 IN, for example, firm-ware should clear the BUSY bit in the EP1CS register; for EP1OUT, firmware should load any value into the EP1 byte-count register.
4. Clear the HSNAK bit (by writing '1' to it) to terminate the Set Interface CONTROL transfer

7. 获取接口请求　Get Interface

8. 设置地址请求 SetAddress

FX2 将主机得到的地址复制到 FNADDR，接着只对该地址的请求反应 该请求 FIFO 处理，固件不处理

9. 帧同步请求

Table 2-23. Sync Frame

Byte	Field	Value	Meaning	Firmware Response
0	bmRequestType	**0x82**	IN, Endpoint	Send a frame number over EP0 to synchronize endpoint #EP
1	bRequest	**0x0C**	'Sync Frame'	
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	**EP**	Endpoint number	
5	wIndexH	0x00		
6	wLengthL	**2**	LenL	
7	wLengthH	**0**	LenH	

CYPRESS 提供的请求处理函数

```
void SetupCommand(void)
{
    void    *dscr_ptr;
    switch(SETUPDAT[1])
    {
        case SC_GET_DESCRIPTOR:              //获得描述符
            if(DR_GetDescriptor())                  //空函数
                switch(SETUPDAT[3])
                {
                    case GD_DEVICE:                 //设备
                       SUDPTRH = MSB(pDeviceDscr);
                       SUDPTRL = LSB(pDeviceDscr);
                        break;
                    case GD_DEVICE_QUALIFIER:                 //设备限定
                           if (HighSpeedCapable())
                           {
                             SUDPTRH = MSB(pDeviceQualDscr);
                              SUDPTRL = LSB(pDeviceQualDscr);
                           }
                           else
                           {
                                EZUSB_STALL_EP0(); //EP0CS |= bmEPSTALL
                                                   //setup 令牌到达该为自动置位
                           }
                           break;
                    case GD_CONFIGURATION:           //  配置
                       SUDPTRH = MSB(pConfigDscr);
                       SUDPTRL = LSB(pConfigDscr);
                        break;
                    case GD_OTHER_SPEED_CONFIGURATION:   //  其它速率配置
                       SUDPTRH = MSB(pOtherConfigDscr);
                       SUDPTRL = LSB(pOtherConfigDscr);
                        break;
                    case GD_STRING:                  //  字符串
                       if(dscr_ptr = (void *)EZUSB_GetStringDscr(SETUPDAT[2]))
                            //SETUPDAT[2]字符串号 This function returns a pointer to
                           //instance StrIdx of a string descriptor in the descriptor table
                        {
                           SUDPTRH = MSB(dscr_ptr);
                           SUDPTRL = LSB(dscr_ptr);
                        }
                        else
                           EZUSB_STALL_EP0();    //  中止端点 0
                        break;
                     default:                //  无效请求
                       EZUSB_STALL_EP0();        //  中止端点 0
                }
            break;
        case SC_GET_INTERFACE:                       //  获得接口
            DR_GetInterface();     //返回接口描述符中的可替换设置值
            break;
        case SC_SET_INTERFACE:                       //  设置接口
            DR_SetInterface();     //设置接口描述符中的可替换设置值
            break;
        case SC_SET_CONFIGURATION:                    //设置配置
            DR_SetConfiguration();    //配置配置数
```

```
    break;
case SC_GET_CONFIGURATION:                  //获得配置
    DR_GetConfiguration();  //获取配置数
    break;
case SC_GET_STATUS:                         //  获得状态
    if(DR_GetStatus())                          //空函数
        switch(SETUPDAT[0])
        {
            case GS_DEVICE:                 //  设备
                EP0BUF[0] = ((BYTE)Rwuen << 1) | (BYTE)Selfpwr;
                EP0BUF[1] = 0;
                EP0BCH = 0;
                EP0BCL = 2;
                break;
            case GS_INTERFACE:              //  接口
                EP0BUF[0] = 0;
                EP0BUF[1] = 0;
                EP0BCH = 0;
                EP0BCL = 2;
                break;
            case GS_ENDPOINT:               //  端点
              EP0BUF[0] = *(BYTE xdata *) epcs(SETUPDAT[4]) & bmEPSTALL;
                  //(EPCS_Offset_Lookup_Table[(EP & 0x7E) | (EP > 128)] + 0xE6A1)
                EP0BUF[1] = 0;
                EP0BCH = 0;
                EP0BCL = 2;
                break;
            default:                    //无效命令
                EZUSB_STALL_EP0();
        }
    break;
case SC_CLEAR_FEATURE:                      //  清除远程唤醒
   if(DR_ClearFeature()) //空函数
        switch(SETUPDAT[0])
        {
            case FT_DEVICE:                 //  设备
                if(SETUPDAT[2] == 1)
                    Rwuen = FALSE;          //  禁止远程唤醒
                else
                    EZUSB_STALL_EP0();
                break;
            case FT_ENDPOINT:
                if(SETUPDAT[2] == 0)
                {
                  *(BYTE xdata *) epcs(SETUPDAT[4]) &= ~bmEPSTALL;
                  EZUSB_RESET_DATA_TOGGLE( SETUPDAT[4] );
                }
                else
                    EZUSB_STALL_EP0();
                break;
        }
    break;
case SC_SET_FEATURE:                        //  设置特性
    if(DR_SetFeature())
        switch(SETUPDAT[0])
        {
            case FT_DEVICE:
                if(SETUPDAT[2] == 1)
                    Rwuen = TRUE;
```

```
                    else if(SETUPDAT[2] == 2)
                        break;
                    else
                        EZUSB_STALL_EP0();
                    break;
                case FT_ENDPOINT:
                  *(BYTE xdata *) epcs(SETUPDAT[4]) |= bmEPSTALL;
                    break;
                default:
                    EZUSB_STALL_EP0();
            }
        break;
      case SC_LED1:                                   //显示数字 0
                DR_LED1();
                break;
      default:                                      // 无效命令
        if(DR_VendorCmnd())
            EZUSB_STALL_EP0();                      //  停止断点 0
   }

   //握手
   EP0CS |= bmHSNAK;
}
```

二、关于 CYPRESS

1. EZ-USB FX2LP 的启动模式

USB 设备接上主机的响应过程

1. 连接了设备的 HUB 在 HOST 查询其状态改变端点 状态改变端点状态改变端点 状态改变端点时返回对应的 bitmap，告知 HOST 某个 PORT 状态发生了改变。

2. 主机向 HUB 查询该 PORT 的状态，得知有设备连接，并知道了该设备的基本特性。

3. 主机等待（至少 100mS）设备上电稳定，然后向 HUB 发送请求，复位并使能该 PORT。

4. HUB 执行 PORT 复位操作，复位完成后该 PORT 就使能了。现在设备进入到 defalut 状态，可以从 Vbus 获取不超过 100mA 的电流。主机可以通过 0 地址与其通讯。

5. 主机通过 0 地址向该设备发送 get_device_descriptor 标准请求，获取设备的描述符。设备收到该请求后，在数 据过程将设备描述符返回给主机。主

机在成功获取到一个数据包的设备描述符后并且确认没有什么错误后（注意：有些 USB 设备的端点 0 大小不足 18 字节（但至 少具有 8 字节），而标准的设备描述有 18 字节，在这种情况下，USB 设备只能暂时按最大包将部分设备描述符返回，而主机在成功获取到前面一部分描述符后， 就不会再请求剩下的设备描述符部分，而是进入设置地址阶段），就会返回一个 0 长度的状态数据包给设备。

6. 主机再次向 HUB 发送请求，复位该 PORT。

7. 主机通过标准请求 set_address 给设备分配地址。USB 设备在收到地址后，返回 0 长度的状态包，主机收到 0 长度的状态包之后，会返回一个 ACK 给设备。设备在收到这个 ACK 之后，就可以启用新的地址了。这样设备就分配到了一个唯一的设备地址，以后主机就通过它来进行访问该设备

8. 主机通过新地址向设备发送 get_device_descriptor 标准请求，获取设备的描述符。这次跟第一次可能有点不一样，这次需要获取完全部的 18 个字节的设备描述符（如果一次读不完会对此请求读取？）。当然，如果你的端点 0 缓冲大于 18 字节的话，那 就跟第一次的情形一样了

9. 主机通过新地址向设备发送其他 get_configuration 请求，获取设备的配置描述符。主机在获取到配置描述符后，根据里面的配置集合总长度，再获取配置集合。配置集合包括配置描述 符，接口描述符，端点描符等等, 如果有字符串描述符的话，还要获取字符串描述符。另外 HID 设备还有 HID 描述符等。使用 BUS HOUND 以及通过串口返回信息，很容易看到具体的过程。总之是主机请求什么，你的程序就响应什么

10. 根据配置信息，主机选择合适配置，通过 set_configuration 请求对设备而进行配置。这时设备方可正常使用。

1.1 无 EEPROM 启动

没有 EEPROM 连接到芯片或 EEPROOM 首字节不是 OXC0 或 0XC2,USB 的设备描述符由芯片内部提供，个描述符如下：

Table 3-3. Default ID Values for EZ-USB FX2LP, No EEPROM / Invalid EEPROM

Vendor ID	0x04B4 (Cypress Semiconductor)
Product ID	0x8613 (EZ-USB FX2LP)
Device Release	0xAnnn (depends on chip revision, nnn = chip revision, where first silicon = 001)

芯片内部的各种描述符进行列举后,将主机的固件程序下载到芯片 RAM 中，芯片退出复位状态，运行固件，在固件中对设备进行重列举，因此不能在同一电脑上使用两个同类芯片实现不同功能。

1.2 首字节为 0xC0 的 EEPROM 启动

芯片从 EEPROM 中读取 VID、PID、DID，进行默认列举，然后主机让 usb 设备处于复位状态（ＣＰＵＣＳ．１置１，详见复位介绍），而其他各种描述符都由芯片内部提供，下载固件，主机让 usb 设备脱离复位ＣＰＵＣＳ．１清零，再重列举，

数据在 EEEPROM 中的存储格式如下：

Table 3-5. ‘C0 Load’ Format

EEPROM Address	Contents
0	**0xC0**
1	Vendor ID (VID) L
2	Vendor ID (VID) H
3	Product ID (PID) L
4	Product ID (PID) H
5	Device ID (DID) L
6	Device ID (DID) H
7	Configuration byte 配置字节

配置字节格式：

Configuration							
b7	b6	b5	b4	b3	b2	b1	b0
0	DISCON	0	0	0	0	0	400KHZ

400KHZ： IIC 总线工作频率　默认 100KHZ

DISCON： USB 链接控制，断开极限，，为 1 时 USBCS.3 位被重写，USB 断开，默认为链接到 USB。allows the EZ-USB to come up 'disconnected.' The EZ-USB core sees that this DISCON bit is set, and sets the USBCS.3 bit before the CPU is taken out of reset.

The DISCON bit in the EEPROM configuration byte cannot be used to instruct the EZ-USB to connect to the USB bus. Once the CPU is running, firmware can modify this bit

USB 的断开与链接

主机或 Hub 的下行端口的 D+与 D 都用 15K 电阻接地，使得在没有设备插入的时候 D+与 D 上的电平始终为低，全速设备的上行端口的 D+通过 1.5K 电阻接到 3.3V 而低速设备的上行端口的 D 通过 1.5K 电阻也接到 3.3V ，使得在设备插入主机或 Hub 的时候 D+或 D 上会产生一个上升沿，芯片可以通过向片内 1.5K 电阻加上和撤销 3.3v 电源来实现 USB 的非物理断开，由 USBCS.3(DISCON)控制。

在芯片寄存器中 USBCS 寄存器

USBCS			USB Control and Status				E680
b7	b6	b5	b4	b3	b2	b1	b0
HSM	0	0	0	DISCON	NOSYNSOF	RENUM	SIGRSUME
R/W	R	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	1	0	0

DISCON：控制 USB 的链接状态，0 为链接状态，1 为断开状态

RENUM：为 0，为芯片自己出来设备请求，为 1，由固件处理请求。上电复位时它将被置 0，固件下载后，被置 1。

1.3　首字节为 0XC2 的 EEPROM 启动

EEPROM 提供 VID、PID、DID、和各种描述符。ＥＥＰＲＯＭ中的存储格式详见　EZ-USB_referrence manual.pdf

２．复位与唤醒

复位

ＲＳＴ引脚复位
ＣＰＵ复位，由内部ＣＵＰＣＳ．１控制
ＵＳＢ协议定义了ＵＳＢ总线复位

２．１ＣＰＵ复位

CPUCS			CPU Control and Status				E600
b7	b6	b5	b4	b3	b2	b1	b0
0	0	PORTCSTB	CLKSPD1	CLKSPD0	CLKINV	CLKOE	8051RES
R	R	R/W	R/W	R/W	R/W	R/W	R
0	0	0	0	0	0	1	0

８０５１ＲＥＳ　：为１　　复位，为０　运行ｃｐｕ，只能被主机操作

时钟选择位：

CPU Clock Speed

CLKSPD1	CLKSPD0	CPU Clock
0	0	12 MHz (Default)
0	1	24 MHz
1	0	48 MHz
1	1	Reserved

２．２ＵＳＢ总线复位

主机产生ＳＥ０状态（Ｄ＋，Ｄ－都为低），而且至少１０ｍｓ时间送出一个总线复位信号，芯片感知此变化，请求ＵＳＢ中断，发生复位中断，但芯片重枚举的特性不变

３．电源管理

１．　挂起ｓｕｓｐｅｎｄ

ＵＳＢ总线３ｍｓ的Ｊ状态（总线空闲），主机发往设备请求，设备响应进入低功耗模式，进入闲置状态。向ＳＵＳＰＥＮＤ寄存器写入任何值就会被挂起。挂起将引发挂起中断，

EZ-USB firmware responds to the Suspend interrupt by tak-ing the following actions:

1. Perform any necessary housekeeping such as shutting off external power-consuming devices.

2. Set bit zero of the PCON register，ｂｉｔ０位。

ＣＹＰＲＥＳＳ提供的中断函数如下：

```
//This function is called before the frameworks enter suspend mode.
//This function contains code that places the device in a low power
// state and returns TRUE. However, the user code can prevent the
//frameworks from entering suspend mode by returning FALSE.
BOOL  TD_Suspend(void)
{
    return(TRUE);
}
```

２．　唤醒＼恢复ｒｅｓｕｍｅ

是从设备向主机发出的信号，请求主机脱离低功耗挂起模式，如果主机使能了远程唤醒功能，就支持这种功能。

以下三种方式会唤醒ＵＳＢ：

■USB activity on the EZ-USB's DPLUS pin

■Assertion of the WAKEUP pin

■Assertion of the WU2（Wakeup 2）pin

芯片唤醒后重启振荡器，ＰＬＬ稳定后产生中断请求，唤醒中断，具有最高优先级，ＦＸ２不同于５１内核，它必须有唤醒中断来退出闲置状态

３．中断

支持的ＵＳＢ中断部分如下：

Table 4-10. Individual USB Interrupt Sources

Priority	INT2VEC Value	Source	Notes
1	00	SUDAV	SETUP Data Available
2	04	SOF	Start of Frame (or microframe)
3	08	SUTOK	Setup Token Received
4	0C	SUSPEND	USB Suspend request
5	10	USB RESET	Bus reset
6	14	HISPEED	Entered high-speed operation*
7	18	EP0ACK	EZ-USB ACK'd the CONTROL Hand-shake
8	1C	reserved	
9	20	EP0-IN	EP0-IN ready to be loaded with data
10	24	EP0-OUT	EP0-OUT has USB data

ＥＡ能使能除ＲＥＳＵＭＥ外的所有中断，因为 RESUME 总被使能

EXIF.4　USB 总中断标志

USBIRQ　具体的 USB 中断标志

中断标志必须手动清零

EIE.0　使能 USB 中断