USB 2.0 规范

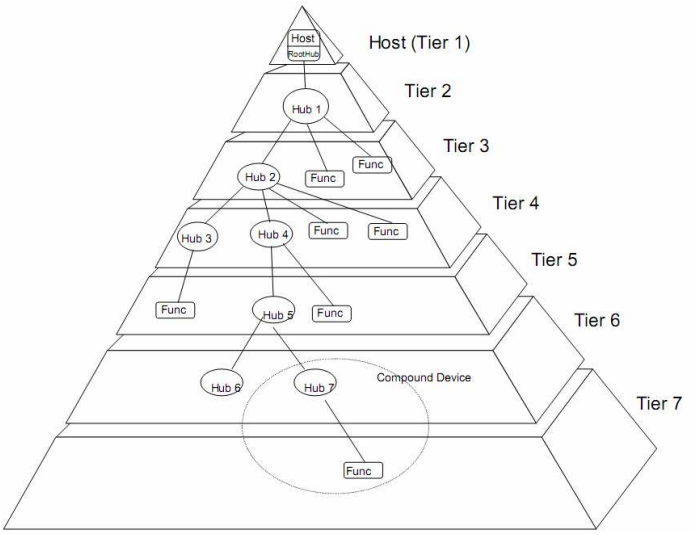
USB 体系简介

USB 是一种支持热插拔的高速串行传输总线，它使用差分信号来传输数据，最高速度可达 480Mb/S。USB 支持“总线供电”和“自供电”两种供电模式。在总线供电模式下，设备最多可以获得 500mA 的电流。USB2.0 被设计成为向下兼容的模式，当有全速（USB 1.1）或者低速（USB 1.0）设备连接到高速（USB 2.0）主机时，主机可以通过分离传输来支持它们。一条 USB 总线上，可达到的最高传输速度等级由该总线上最慢的“设备”决定，该设备包括主机、HUB 以及 USB 功能设备。

USB 体系包括“主机”、“设备”以及“物理连接”三个部分。其中主机是一个提供 USB 接口及接口管理能力的硬件、软件及固件的复合体，可以是 PC，也可以是 OTG 设备。一个 USB 系统中仅有一个 USB 主机；设备包括 USB 功能设备和 USB HUB，最多支持 127 个设备；物理连接即指的是 USB 的传输线。在 USB 2.0 系统中，要求使用屏蔽的双绞线。

> 一个 USB HOST 最多可以同时支持 128 个地址，地址 0 作为默认地址，只在设备枚举期间临时使用，而不能被分配给任何一个设备，因此一个 USB HOST 最多可以同时支持 127 个地址，如果一个设备只占用一个地址，那么可最多支持 127 个 USB 设备。在实际的 USB 体系中，如果要连接 127 个 USB 设备，必须要使用 USB HUB，而 USB HUB 也是需要占用地址的，所以实际可支持的 USB 功能设备的数量将小于 127。

USB 体系采用分层的星型拓扑来连接所有 USB 设备，如下图所示：



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。一个复合设备（Compound Device）将同时占据两层或更多的层。

> ROOT HUB 是一个特殊的 USB HUB，它集成在主机控制器里，不占用地址。ROOT HUB 不但实现了普通 USB HUB 的功能，还包括其他一些功能，具体在增强型主机控制器的规范中有详细的介绍。

> “复合设备（Compound Device）”可以占用多个地址。所谓复合设备其实就是把多个功能设备通过内置的 USB HUB 组合而成的设备，比如带录音话筒的 USB 摄像头等。

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。

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）。令牌包即指 PID 为 IN/OUT/SETUP 的包。
>
> 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。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。

管道（Pipe）是主机和设备端点之间数据传输的模型，共有两种类型的管道：无格式的流管道（Stream Pipe）和有格式的信息管道（Message Pipe）。任何 USB 设备一旦上电就存在一个信息管道，即**默认的控制管道**，USB 主机通过该管道来获取设备的描述、配置、状态，并对设备进行配置。

USB 设备连接到 HOST 时，HOST 必须通过默认的控制管道对其进行**枚举**，完成获得其设备描述、进行地址分配、获得其配置描述、进行配置等操作方可正常使用。USB 设备的即插即用特性即依赖于此。

> 枚举：是 USB 体系中一个很重要的活动，由一系列标准请求组成（若设备属于某个子类，还包含该子类定义的特殊请求）。通过枚举 HOST 可以获得设备的基本描述信息，如支持的 USB 版本、PID、VID、设备分类（Class）、供电方式、最大消耗电流、配置数量、各种类型端点的数量及传输能力（最大包长度）。HOST 根据 PID 和 VID 加载设备驱动程序，并对设备进行合适的配置。只有经过枚举的设备才能正常使用。对于总线供电设备，在枚举完成前最多可从总线获取 100mA 的电流。

USB 体系定义了四种类型的传输，它们是：

控制传输：主要用于在设备连接时对设备进行枚举以及其他因设备而已的特定操作。

中断传输：用于对延迟要求严格、小量数据的可靠传输，如键盘、游戏手柄等。

批量传输：用于对延迟要求宽松，大量数据的可靠传输，如 U 盘等。

同步传输：用于对可靠性要求不高的实时数据传输，如摄像头、USB 音响等。

注意：中断传输并不意味这传输过程中，设备会先中断 HOST，继而通知 HOST 启动传输。中断传输也是 HOST 发起的传输，采用轮询的方式询问设备是否有数据发送，若有则传输数据，否则 NAK 主机。

不同的传输类型在物理上并没有太大的区别，只是在传输机制、主机安排传输任务、可占用 USB 带宽的限制以及最大包长度有一定的差异。

USB 设备通过管道和 HOST 通信，在默认控制管道上接受并处理以下三种类型的请求：

1. **标准请求：**一共有 11 个标准请求，如得到设备描述、设置地址、得到配置描述等。所有 USB 设备均应支持这些请求。HOST 通过标准请求来识别和配置设备。

2. **类（class）请求：**USB 还定义了若干个子类，如 HUB 类、大容量存储器类等。不同的类又定义了若干类请求，该类设备应该支持这些类请求。设备所属类在设备描述符中可以得到。

3. **厂商请求：**这部分请求并不是 USB 规范定义的，而是设备生产商为了实现一定的功能而自己定义的请求。

USB HUB 提供了一种低成本、低复杂度的 USB 接口扩展方法。HUB 的上行 PORT 面向 HOST，下行 PORT 面向设备（HUB 或功能设备）。在下行 PORT 上，HUB 提供了设备连接检测和设备移除检测的能力，并给各下行 PORT 供电。HUB 可以单独使能各下行 PORT，不同 PORT 可以工作在不同的速度等级（高速/全速/低速）。

HUB 由 HUB 重发器（HUB Repeater）、转发器（Transaction Translator）以及 HUB 控制器（HUB Controller）三部分组成。HUB Repeater 是上行 PORT 和下行 PORT 之间的一个协议控制的开关，它负责高速数据包的**重生与分发**。HUB 控制器负责和 HOST 的通信，HOST 通过 HUB 类请求和 HUB 控制器通讯，获得关于 HUB 本身和下行 PORT 的 HUB 描述符，进行 HUB 和下行 PORT 的监控和管理。转发器提供了从高速和全速/低速通讯的转换能力，通过 HUB 可以在高速 HOST 和全速/低速设备之间进行匹配。HUB 在硬件上支持 Reset、Resume、Suspend。

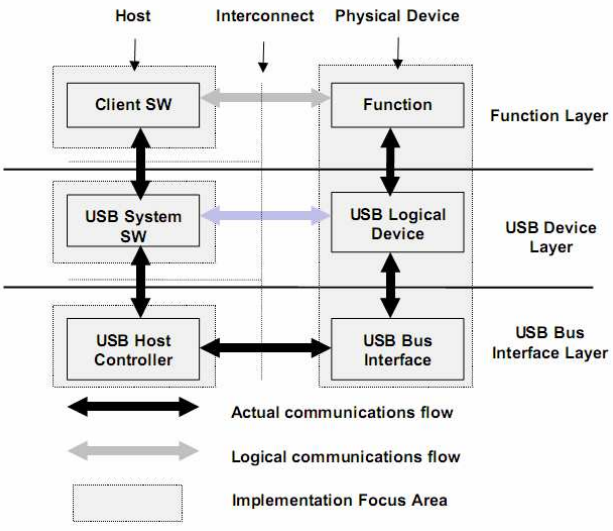
> 重生与分发：指的是 HUB Repeater 需要识别从上行（下行）PORT 上接收到的数据，并分发到下行(上行)PORT。所谓分发主要是指从上行 PORT 接收到的数据包需要向所有使能的高速下行 PORT 发送，即广播。

USB HOST 在 USB 体系中负责设备连接/移除的检测、HOST 和设备之间控制流和数据流的管理、传输状态的收集、总线电源的供给。

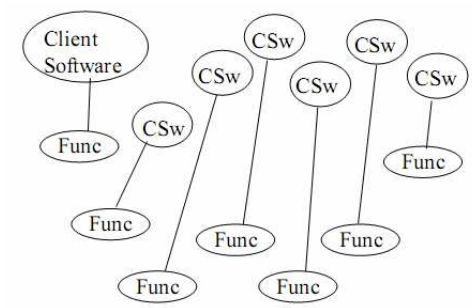
USB 数据流模型

USB 体系在实现时采用分层的结构，如下图所示：

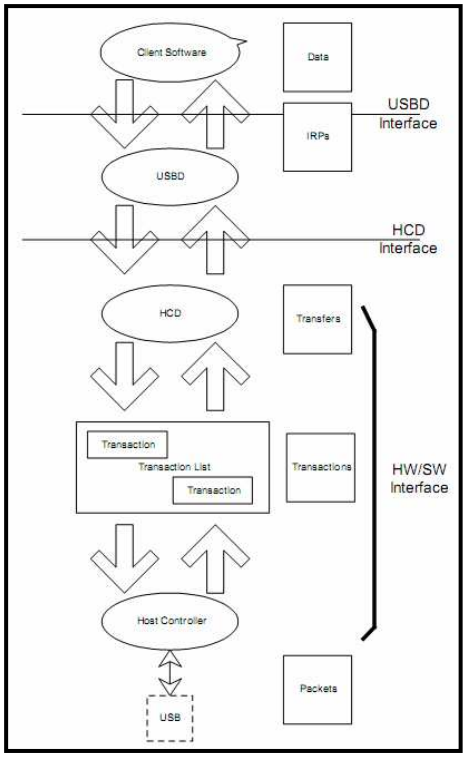
在 HSOT 端，应用软件（Client SW）不能直接访问 USB 总线，而必须通过 USB 系统软件和 USB 主机控制器来访问 USB 总线，在 USB 总线上和 USB 设备进行通讯。从逻辑上可以分为功能层、设备层和总线接口层三个层次。其中功能层完成功能级的描述、定义和行为；设备级则完成从功能级到传输级的转换，把一次功能级的行为转换为一次一次的基本传输；USB 总线接口层则处理总线上的 Bit 流，完成数据传输的物理层实现和总线管理。途中黑色箭头代表真实的数据流，灰色箭头代表逻辑上的通讯。



物理上，USB 设备通过分层的星型总线连接到 HOST，但在逻辑上 HUB 是透明的，各 USB 设备和 HOST 直接连接，和 HOST 上的应用软件形成一对一的关系。如下图所示：



各应用软件-功能设备对之间的通讯相互独立，应用软件通过 USB 设备驱动程序(USBD)发起 IRQ 请求，请求数据传输。主机控制器驱动程序（HCD）接收 IRQ 请求，并解析成为 USB 传输和传输事务（Transaction），并对 USB 系统中的所有传输事务进行任务排定（因为可能同时有多个应用软件发起 IRQ 请求）。主机控制器（Host Controller）执行排定的传输任务，在同一条共享的 USB 总线上进行数据包的传输。如下图所示。



USB 系统中数据的传输，宏观的看来是在 HOST 和 USB 功能设备之间进行；微观的看是在应用软件的 Buffer 和 USB 功能设备的端点之间进行。一般来说端点都有 Buffer，可以认为 USB 通讯就是应用软件 Buffer 和设备端点 Buffer 之间的数据交换，交换的通道称为管道。应用软件通过和设备之间的数据交换来完成设备的控制和数据传输。通常需要多个管道来完成数据交换，因为同一管道只支持一种类型的数据传输。用在一起来对设备进行控制的若干管道称为设备的接口，这就是**端点、管道和接口的关**系。

一个 USB 设备可以包括若干个端点，不同的端点以**端点编号**和**方向**区分。不同端点可以支持不同的传输类型、访问间隔以及最大数据包大小。除端点 0 外，所有的端点只支持一个方向的数据传输。端点 0 是一个特殊的端点，它支持双向的控制传输。管道和端点关联，和关联的端点有相同的属性，如支持的传输类型、最大包长度、传输方向等。

四种传输类型

1. 控制传输：

控制传输是一种可靠的**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；第二阶段为数据阶段，也有些请求没有数据阶段；第三阶段为状态阶段，通过一次 IN/OUT 传输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。

> **最大包长度**表征了一个端点单次接收/发送数据的能力，实际上反应的是该端点对应的 Buffer 的大小。Buffer 越大，单次可接收/发送的数据包越大，反之亦反。当通过一个端点进行数据传输时，若数据的大小超过该端点的最大包长度时，需要将数据分成若干个数据包传输，并且要求除最后一个包外，所有的包长度均等于该最大包长度。这也就是说如果一个端点收到/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。

控制传输在访问总线时也受到一些限制，如：

- 高速端点的控制传输不能占用超过 20%的微帧，全速和低速的则不能超过 10%。
- 在一帧内如果有多余的未用时间，并且没有同步和中断传输，可以用来进行控制传输。

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$(2^{interval-1})$*125uS，其中 interval 取 1 到 16 之间的值。

除高速高带宽中断端点外，一个微帧内仅允许一次中断事务传输，高速高带宽端点最多可以在一个微帧内进行三次中断事务传输，传输高达 3072 字节的数据。

> 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。

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

一个数据包。

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。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。

在主机控制器和 USB HUB 之间还有另外一种传输——分离传输（Split Transaction），它仅在主机控制器和 HUB 之间执行，通过分离传输，可以允许全速/低速设备连接到高速主机。分离传输对于 USB 设备来说是透明的、不可见的。

> 分离传输：顾名思义就是把一次完整的事务传输分成两个事务传输来完成。其出发点是高速传输和全速/低速传输的速度不相等，如果使用一次完整的事务来传输，势必会造成比较长的等待时间，从而降低了高速 USB 总线的利用率。通过将一次传输分成两此，将令牌（和数据）的传输与响应数据（和握手）的传输分开，这样就可以在中间插入其他高速传输，从而提高总线的利用率。

USB 物理规范和电气规范

此节略去，请参考 USB2.0 规范英文版

USB 协议层规范

USB 采用 little edian 字节顺序，在总线上先传输一个字节的最低有效位，最后传输最高有效位，采用 NRZI 编码，若遇到连续的 6 个 1 要求进行为填充，即插入一个 0。

所有的 USB 包都由 SYNC 开始，高速包的 SYNC 宽度为 32bit，全速/低速包的 SYNC 段度为 8bit。实际接收到的 SYNC 产度由于 USB HUB 的关系，可能会小于该值。

USB 数据包的格式

Field	PID	ADDR	ENDP	DATA	CRC
		FrameNumber			
Bits	4+4	7	4	N*8(n=0,1,...,1024)	5/16
		11			

PID 表征了数据包的类型，分为令牌（Token）、数据（Data）、握手（Handshacke）以及特殊包 4 大类，共 16 种类型的 PID。具体定义见英文协议第 196 页。

对于令牌包来说，PID 之后是 7 位的地址和 4 位的端点号。令牌包没有数据域，以 5 位的 CRC 校验和结束。SOF 是一类特殊的令牌包，PID 后跟的是 11 位的帧编号。

对于数据包来说，PID 之后直接跟数据域，数据域的长度为 N 字节，数据域后以 16 位

的 CRC 校验和结束。

握手包仅有 PID 域，没有数据也没有校验和。

分离传输会用到一类特殊的包，Start-Split 和 Complete-Split 包，格式如下：

(lsb) (msb)

Field	SPLIT PID	Hub Addr	SC	Port	S	E	ET	CRC5
Bits	8	7	1	7	1	1	2	5

(lsb) (msb)

Field	SPLIT PID	Hub Addr	SC	Port	S	U	ET	CRC5
Bits	8	7	1	7	1	1	2	5

在 Start-Split 和 Complete-Split 包中主要指定了此次分离传输所在的 HUB 的地址和下行端口编号以及端点类型（控制、中断、批量、同步）。以及此次传输中数据包在整个数据中的位置（第一个包、中间的包、末尾的包）。

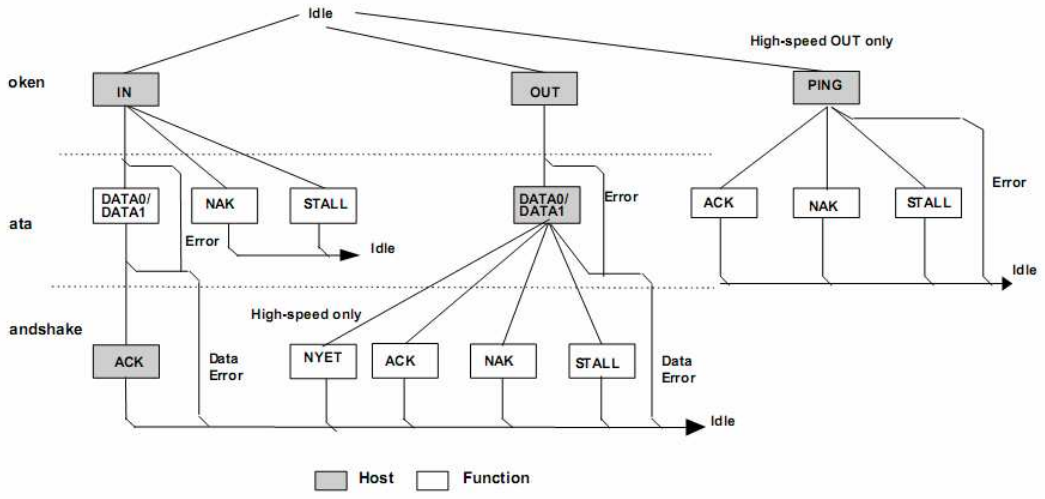
握手包包括 ACK，NAK，STALL 以及 NYET 四种，其中 ACK 表示肯定的应答，成功的数据传输；NAK 表示否定的应答，失败的数据传输，要求重新传输；STALL 表示功能错误或端点被设置了 STALL 属性；NYET 表示尚未准备好，要求等待。

数据在 USB 总线上的传输以包为单位，包只能在帧内传输。高速 USB 总线的帧周期为 125uS，全速以及低速 USB 总线的帧周期为 1mS。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输。

注意：虽然高速 USB 总线和全速/低速 USB 总线的帧周期不一样，当时 SOF 包中帧编号的增加速度是一样的，因为在高速 USB 系统中，SOF 包中帧编号实际上取得是计数器的高 11 位，最低三位作为微帧编号没有使用，因此其帧编号的增加周期也为 1mS。

事务传输（Transaction）的流程

1. 批量事务传输



图中一个方框表示一个 Packet，灰色的包表示主机发出的包，白色的包表示 Device 发出的包。批量传输是可靠的传输，需要握手包来表明传输的结果。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。

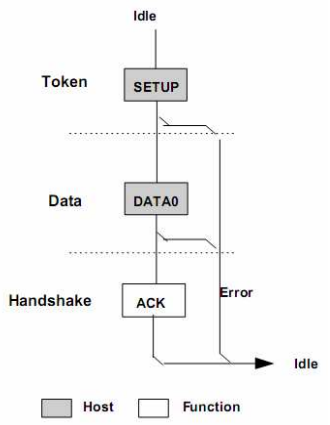
USB 允许连续 3 次以下的传输错误，会重试该传输，若成功则将错误次数计数器清零，否则累加该计数器。超过三次后，HOST 认为该端点功能错误（STALL），放弃该端点的传输任务。

一次批量传输（Transfer）由 1 次到多次批量事务传输（Transaction）组成。

> 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。

2. 控制传输（Transaction）

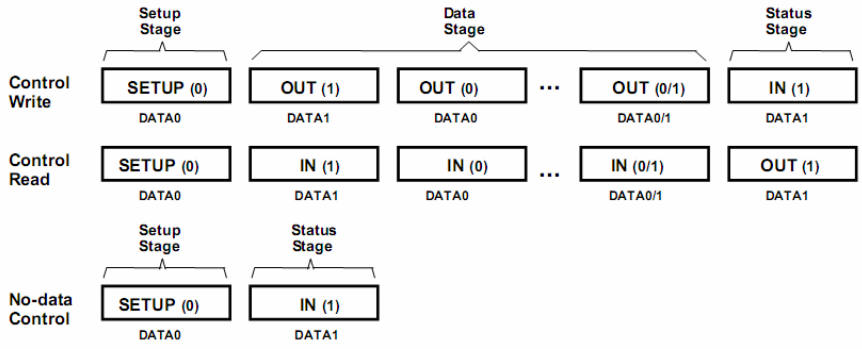
一次控制传输分为三（或两个）个阶段：建立（Setup）、数据（DATA）（可能没有）以及状态（Status）。每个阶段都由一次或多次（数据阶段）事务传输组成（Transaction）。



左图为建立阶段的事务传输流程图。可以看出：与批量传输相比，在流程上并没有多大区别，区别只在于该事务传输发生的端点不一样、支持的最大包长度不一样、优先级不一样等这样一些对用户来说透明的东西。

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。下图为整个控制传输的示意图：



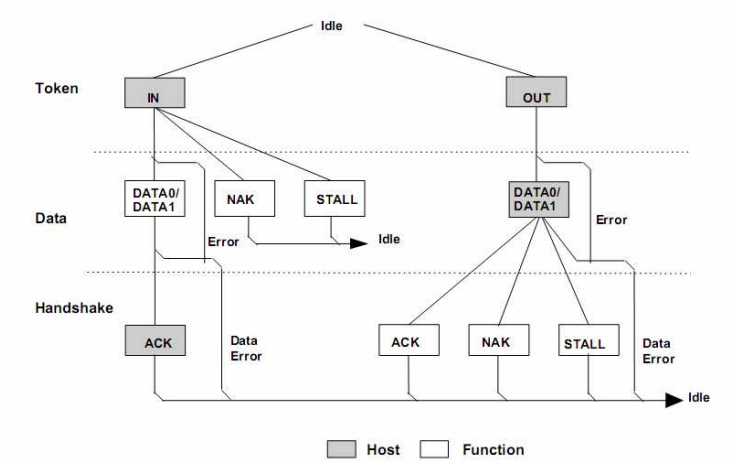
3. 中断传输

中断传输在流程上除不支持 PING 之外，其他的跟批量传输是一样的。他们之间的区别也仅在于事务传输发生的端点不一样、支持的最大包长度不一样、优先级不一样等这样一些对用户来说透明的东西。

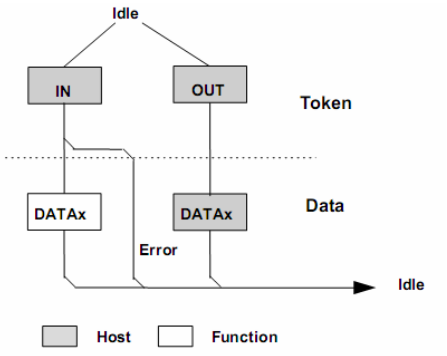
主机在排定中断传输任务时，会根据对应中断端点描述符中指定的查询间隔发

起中断传输。中断传输有较高的优先级，仅次于同步传输。

同样中断传输也采用 PID 翻转的机制来保证收发端数据同步。下图为中断传输的流程图。

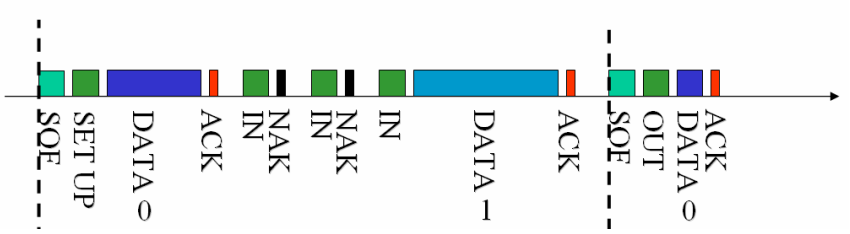


4. 同步传输



同步传输是不可靠的传输，所以它没有握手包，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。

USB 总线上的情形是怎样的？



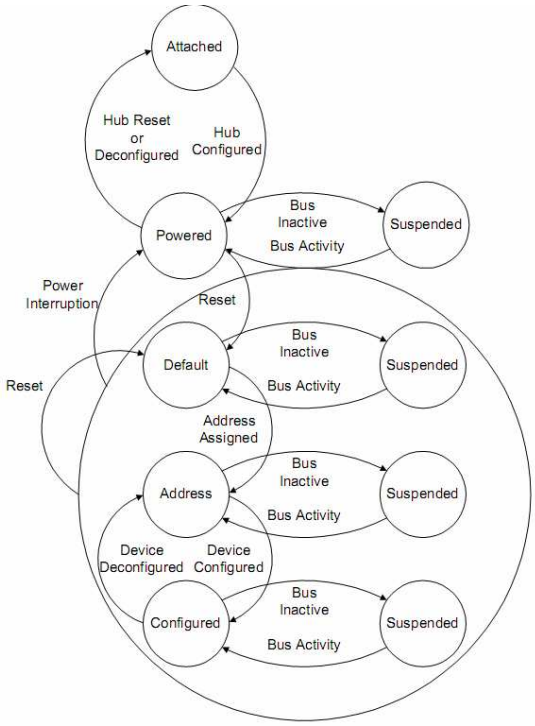
包是 USB 总线是数据传输的最小单位，不能被打断或干扰，否则会引发错误。若干个数据包组成一次事务传输，一次事务传输也不能打断，属于一次事务传输的几个包必须连续，不能跨帧完成。一次传输由一次到多次事务传输构成，可以跨帧完成。

USB 框架

在 USB 框架中，规范主要定义了 USB 设备的各种状态、常用操作、USB 设备请求、描述符、设备类等。

下图为 USB 设备的状态转移图：

Attached	Powered	Default	Address	Configured	Suspended	State
No	--	--	--	--	--	Device is not attached to the USB. Other attributes are not significant.
Yes	No	--	--	--	--	Device is attached to the USB, but is not powered. Other attributes are not significant.
Yes	Yes	No	--	--	--	Device is attached to the USB and powered, but has not been reset.
Yes	Yes	Yes	No	--	--	Device is attached to the USB and powered and has been reset, but has not been assigned a unique address. Device responds at the default address.
Yes	Yes	Yes	Yes	No	--	Device is attached to the USB, powered, has been reset, and a unique device address has been assigned. Device is not configured.
Yes	Yes	Yes	Yes	Yes	No	Device is attached to the USB, powered, has been reset, has a unique address, is configured, and is not suspended. The host may now use the function provided by the device.
Yes	Yes	--	--	--	Yes	Device is, at minimum, attached to the USB and is powered and has not seen bus activity for 3 ms. It may also have a unique address and be configured for use. However, because the device is suspended, the host may not use the device's function.



这里重点介绍下枚举的过程。当设备连接到主机时，按照以下顺序进行枚举：

1. 连接了设备的 HUB 在 HOST 查询其**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。
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。
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。
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。

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。

USB 设备的常用操作包括：设备连接、设备移除、设备配置、地址分配、数据传输、设备挂起、设备唤醒等。

USB 的请求包括标准请求、类请求以及厂商请求三类。所有的请求都通过默认管道发送，按照控制传输的三个阶段进行。首先 HOST 通过一次控制事务传输向 Device 发送一个 8 字节的 Setup 包，这个包说明了请求的具体信息，如请求类型、数据传输方向、接收目标（Device/Interface/Endpoint 等）。具体定义参考 USB2.0 Spec p248。

USB 标准请求共包括 11 个请求，如清除特性（Clear_Feature）、得到配置（Get_Configuration）、得到描述（Get_Descriptor）、设置地址（Set_Address）等。具体参考 USB 2.0 Spec p250

USB 主机：硬件和软件

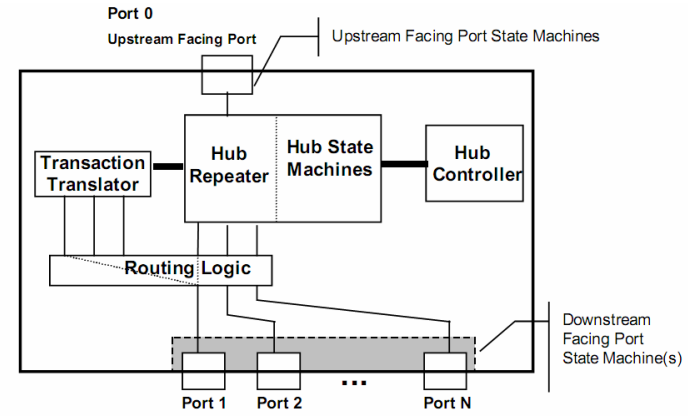
该章节主要定义了 USB 主机的硬件和软件组成及其在 USB 传输中的行为，这部分的硬件设计主要指 EHCI 的设计，软件的设计主要是设备驱动程序和 USB 系统软件的设计。其中 USB 系统软件一般由操作系统提供。由于跟项目关联不大，故略去，如有需要请参考 USB 2.0 Spec Chapter10。

USB HUB 规范

这一章描述了 USB HUB 的架构，主要从 HUB Repeater、HUB Controller 以及 Transaction Translator 三个方面展开。另外还包括 USB HUB 类的请求及描述符。

从功能上来说，HUB 必须支持连接行为、电源管理、设备连接/移除检测、总线错误检测和恢复、高/全/低速设备支持。

下图为 USB HUB 的架构图：



USB HUB 自身的工作速度由上行 PORT 的连接速度决定。从结构上分，USB HUB 由 HUB Repeater、HUB Controller 及 Transaction Translator 三部分组成。其中 HUB Repeater 主要负责连接的建立和撤销，即完成上行 PORT 和下行 PORT 工作在相同速度的连接管理。同时还支持错误的检测与恢复以及设备连接/移除的检测。HUB Controller 负责与 HOST 通讯，完成与 HOST 的交互（请求的响应）、HUB 的控制及管理。Transaction Translator 主要负责高速的分离传输，并把它们分发到连接了全/低速设备的下行 PORT。Routing Logic 负责将下行 PORT 连接到 HUB Repeater 或者 Transaction Translator（以后简

称 TT）。

当 USB HUB 的上行 PORT 连接在全速/低速时，TT 不工作，HUB Repeater 工作在全速/低速模式。当 USB HUB 的上行 PORT 连接在高速时，TT 工作，HUB Repeater 工作在高速模式。

数据的转发

下行方向上：USB HUB 采用广播的方式向所有使能的、工作与上行 PORT 相同速度的下行 PORT 转发来自上行 PORT 的数据包。

上行方向上：USB HUB 将在下行 PORT 接收到的数据包，递交到上行 PORT，而不向其他 PORT 转发。

唤醒信号的转发

唤醒信号（Resume）的转发采用完全广播的方式，在任何 PORT 收到的唤醒信号 USB HUB 都会向其他 PORT 转发。

USB HUB 的帧同步

USB HUB 必须与 HOST 保持同步，拥有和 HOST 相同的帧周期，这是通过帧同步的操作来实现的。在 USB HUB 内有两个计数器一个寄存器，计数器都是用 USB HUB 的本地时钟驱动的。其中一个计数器向上计数，用来测量来自上行 PORT 连续两个 SOF 包的时间间隔；寄存器用来存储此时间间隔，每次 SOF 包到来都会被更新；另一个计数器从测得的时间间隔开始向下计数，计到 0 时即认为新的一帧开始了。然后加载寄存器中存储的值，重新计数。

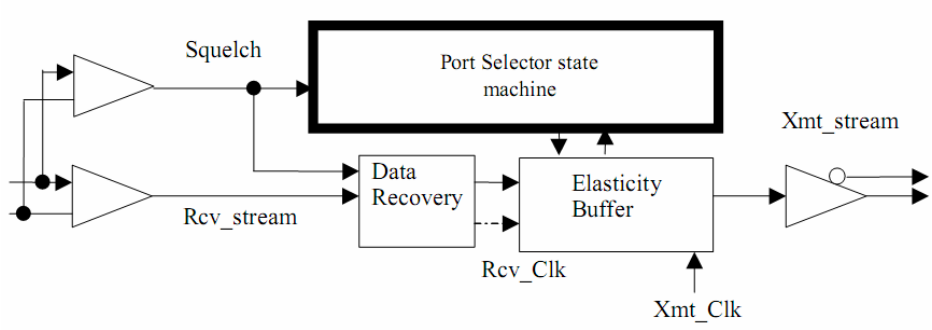
> 由于 USB HUB 检测 SOF 包也是需要时间的，所以实际发现 SOF 包的时间要比该 SOF 包在总线上出现的时间稍晚，如果不加以处理，USB HUB 内部的帧时间会整体比总线上的帧时间推迟一些。该延迟由 USB HUB 本身决定，所以对于 USB HUB 来说是已经的，为了排除此延迟，USB HUB 需要采取一定的方法来处理，如 EOF 提前。即在向下计数器尚未计到 0 时，提前认为当前帧结束。该提前量应恰好等于 USB HUB 检测 SOF 包的延迟。从而保证了在整个 USB 系统中，所有部件的帧时间（帧头和帧尾到来的时刻）除了在 USB 传输线传播带来的传播延迟之外都是一样的。

数字系统的时钟抖动、计数器量化误差都会带来帧周期的抖动，为防止该抖动引起传输错误，USB 规范规定系统中所有部件引起的抖动必须在一定限制范围内，并采取了一定的措施来预防该问题，如 EOF1 和 EOF2 时刻的引入。

> EOF1 和 EOF2 是两个安全时间，比实际的 EOF 时间稍早。USB HUB 在转发数据包时需要参考这两个时间，晚于这两个时间发送的包有可能因为抖动而产生 EOF 期间的干扰。

HUB Repeater

顾名思义，HUB Repeater 将接收到数据 Repeat 出去，这个 repeat 是双向的，既包括从上行 PORT 到下行 PORT 的 repeat，又包括型下行 PORT 到上行 PORT 的 repeat。需要用 USB HUB 的本地时钟从 PORT 上接收数据，然后再用本地时钟将数据放到 PORT 上去。下图为 HUB Repeater 的框图。

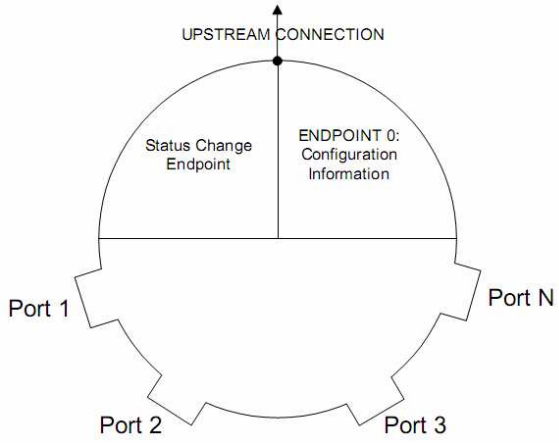


在我们的项目中，USB PHY 芯片完成了 HUB Repeater 大部分的电气层功能，如数据恢复、包检测等。在高速连接中，USB 规范要求从上行 PORT 到下行 PORT 的连接在 SOP（Start of Packet）到来时建立，且延迟不超过 36bit time。实际实验过程中发现即使 USB HUB 的 Repeater 延迟超过 40bit time 也未导致功能失败。

HUB Repeater 的行为受到 EOF1 和 EOF2 时间的影响，具体参考 USB 2.0 Spec P330。

HUB Controller

下图为 HUB Controller 的逻辑结构。



图中右上方是端点 0，即默认的控制端点，HOST 通过向该端点发送标准请求和 HUB 类请求完成 USB HUB 的枚举和控制。如复位某个下行 PORT、使能某个下行 PORT、给某个下行 PORT 上电等。

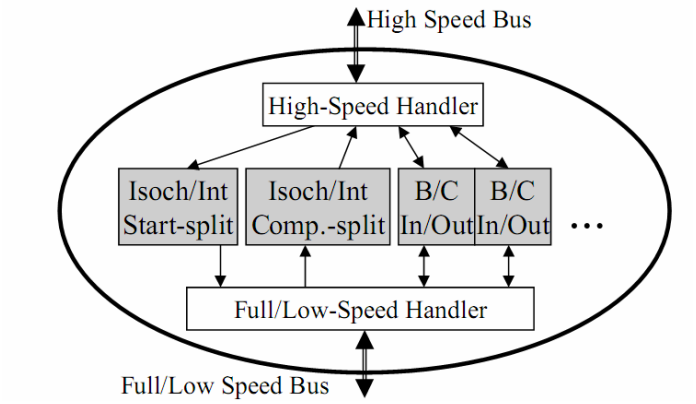
图中左上方是一个中断型的状态改变端点，HOST 通过以固定的时间间隔向该端点发起 IN 的中断传输查询该 HUB 自身和各下行 PORT 的状态是否改变。当没有状态改变时，该端点返回 NAK 的握手包；当有状态改变时返回一个反应了 HUB 自身和各下行 PORT 状态改变信息的 Bitmap。每个 PORT 映射到该 Bitmap 中的一位，当该 PORT 的状态有改变时，对应的位置 1，这是由 USB HUB 硬件完成的。

> 所谓状态改变包括：设备连接、复位完成、设备移除、电流过流等，USB HUB 的硬件负责检测这些事件的发生，并痛过置位反应到 Bitmap 中来。

Transaction Translator

TT 是 USB HUB 中最复杂的一部分，这里我并不准备详细描述它，而只是从结构和基本操作逻辑上展开介绍，更多的细节请在有必要时参考 USB 2.0 Spec。

前面已经说过：TT 在 USB HUB 中主要负责从高速到全/低速传输事务的转发，完成速度的匹配。从传输的角度来说，主要就是完成分离事务传输（Split Transaction）。下图为 TT 的基本结构。



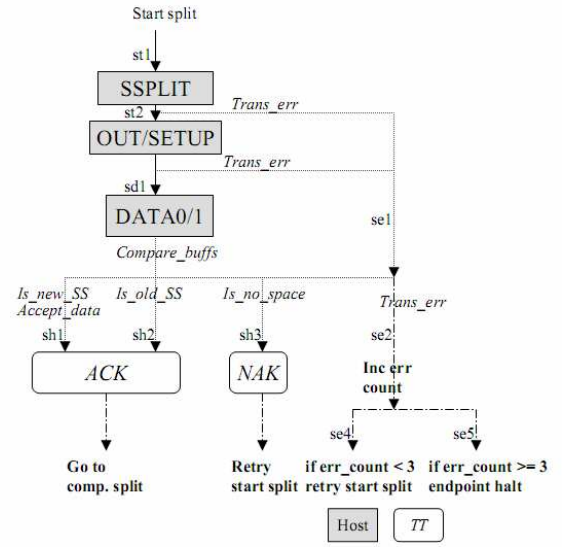
和大部分速度匹配逻辑一样，TT 也采用 Buffer 的方式来完成速度匹配。Buffer 的一端是一个高速 Handler，处理高速数据收发；另一端是全速/低速 Handler 处理全速/低速数据的收发。针对传输类型的不同，TT 采用多个 Buffer 来缓存数据。其中同步传输和中断传输的 Start-Split（S-Split）阶段和 Complete-Split（C-Split）阶段分别用一个 Buffer，批量传输和中断传输的 S-Split 和 C-Split 不分开存储，但是在一个 TT 中至少有 2 个用于批量传输和中断传输的 Buffer。

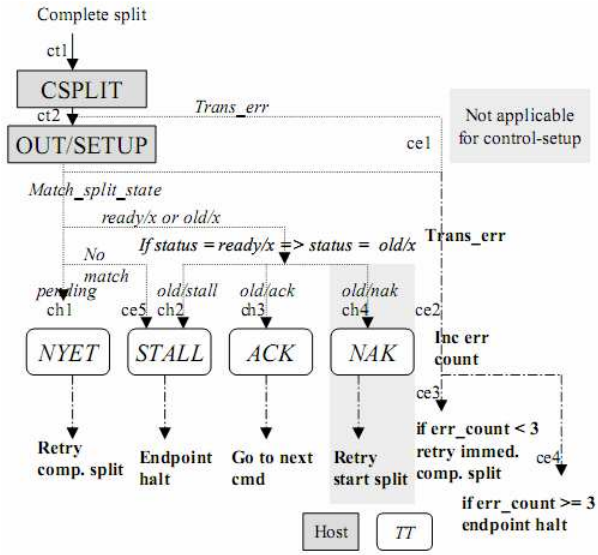
分离传输仅在 HOST 和 HUB 之间发生，对于设备全速/低速设备来说是透明的。一个 USB HUB 可以有多个 TT，这在 USB 协议中是允许的。

按照处理方式的区别，TT 中的分离传输可以分为两类——周期分离传输和非周期分离传输。周期分离传输指的是同步分离传输和中断分离传输，非周期分离传输指的是批量分离传输和控制分离传输。

> 同步传输和中断传输被归为周期传输是因为它们的传输是有周期的，HOST 以固定的间隔向同步端点和中断端点发起传输。相对应的，批量传输和控制传输没有固定的周期。

先来看 OUT 型批量/中断分离传输的流程，如下图所示：



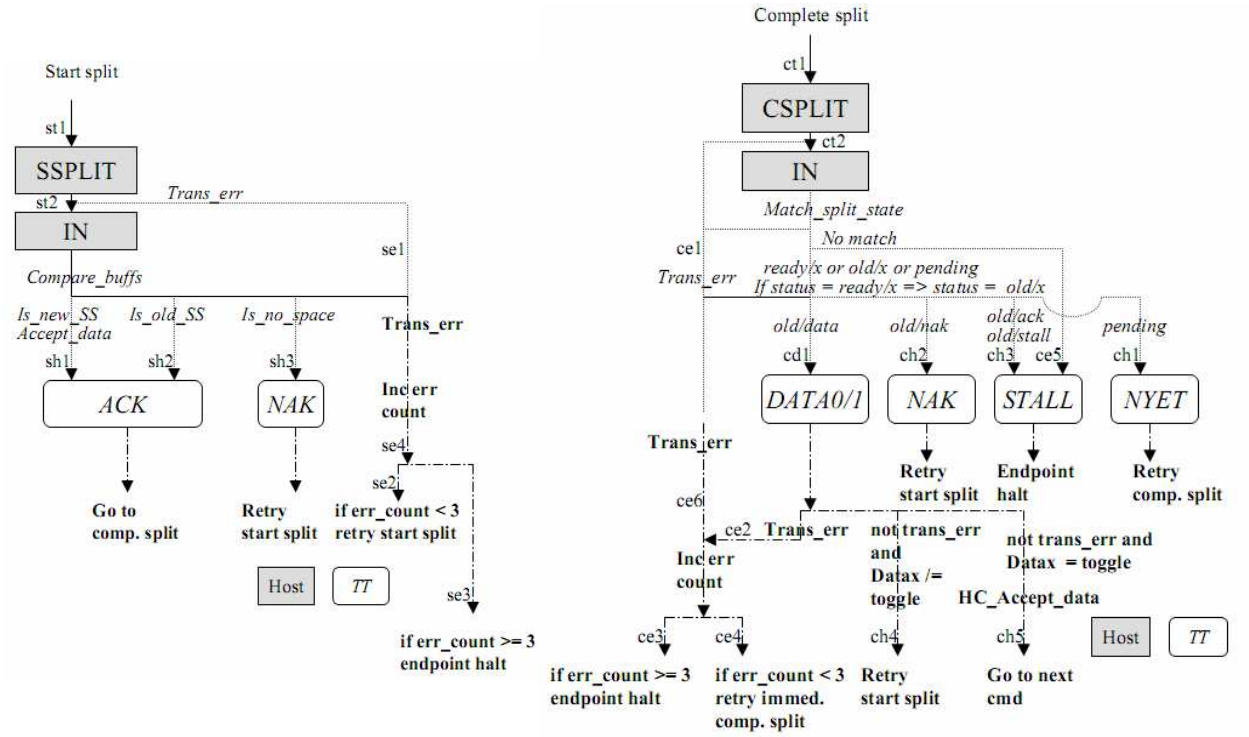


HOST 知道某个 HUB 的某个下行 PORT 上连接的是全速/低速设备（通过查询 HUB 的端口状态得知）。当 HOST 要向该设备传送数据时，它必须启动一次 OUT 型的分离传输。首先 HOST 发送一个 SSPLIT 包，表明：下面我要进行分离传输了。SSPLIT 包中指明了该分离传输所对应的 HUB 地址和 PORT 编号，同时也指明了端点的类型（具体定义参考 Start-Split 包的定义）。然后按照普通批量/中断传输的顺序，先发送一个 OUT/SETUP 的令牌包，然后发送一个数据包，并等待来自 HUB 的握手包。如果没问题，HUB 的 TT 会在 Buffer 开辟一片缓存保存以上接收到的三个包后，并 ACK 主机，表明已经接收到 Start-Split 阶段

的数据包。然后 TT 根据 SSPLIT 包中的信息，通过全速/低速 Handler 向指定的下行 PORT 转发令牌包和数据包，并接收来自设备的握手响应，同样保存在 Buffer 中，并更新此次分离传输的状态信息。

主机在完成 Start-Split 阶段一段时间后，向 HUB 发起 Complete-Split 传输，查询此次传输的状态。具体过程为，先发送一个 CSPLIT 包，然后是 OUT/SETUP 包，TT 根据这两个包可以查询到已经存储在 Buffer 中的分离传输，如果查询不到，则认为传输错误，返回 STALL 握手包。如果查询到了，但设备还没有响应，则返回 NYET 握手包，让主机等待一段时间后重新查询。如果传输已经完成，则 TT 向 HOST 转发来自设备的握手包（STALL/ACK/NAK）。这时一次完整的分离传输完成。

IN 型的批量/控制分离传输有相似的过程，只不过数据包的传输在 C-Split 阶段，下图为其流程图。



周期型的分离传输基本过程和非周期的分离传输大致相同，但在细节上有很多不一样的地方。如周期性的分离传输使用两个 Buffer 分别存放 S-Split 和 C-Split 阶段的数据，主机在安排 S-Split 和 C-Split 也和非周期分离传输有些不同，有较多的限制。

TT 采用流水的方式处理周期型的分离传输，一次周期型的分离传输在 TT 中共有四种状态：New、Pending、Ready 以及 Old。一次分离传输依次经历这四种状态。其中 New 态为这次分离传输在 TT 中建立的状态，是一个暂态、最长不能超过 1 个微帧的时间；Pending 为等待传输完成的状态，最长不能超过 4 个微帧；Ready 为传输已经完成，等待 Host 取回结果的状态，最长不能超过 2 个微帧；Old 表示传输已经全部完成，TT 中该传输所占用的 Buffer 可以重新利用。

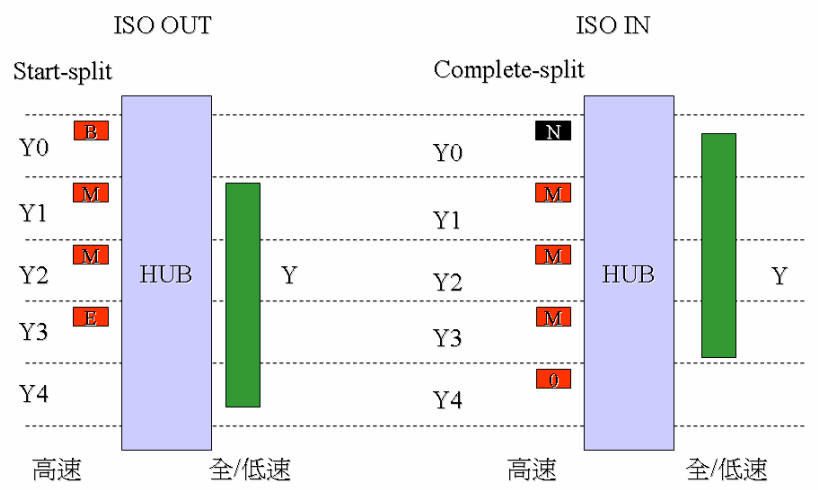
由于速度不匹配，在一个微帧里，最多可以在全速总线上传输 188 字节的数据，在传输的数据量较大的情况下，例如同步传输的最大包长度为 1023 字节，如果等到整个包传完再响应 HOST 的 C-Split，不但要求 TT 有更多的 Buffer，并且会使 HOST 浪费较多的时间在等

待数据传输完成。因此 TT 采用了如下的处理方式：

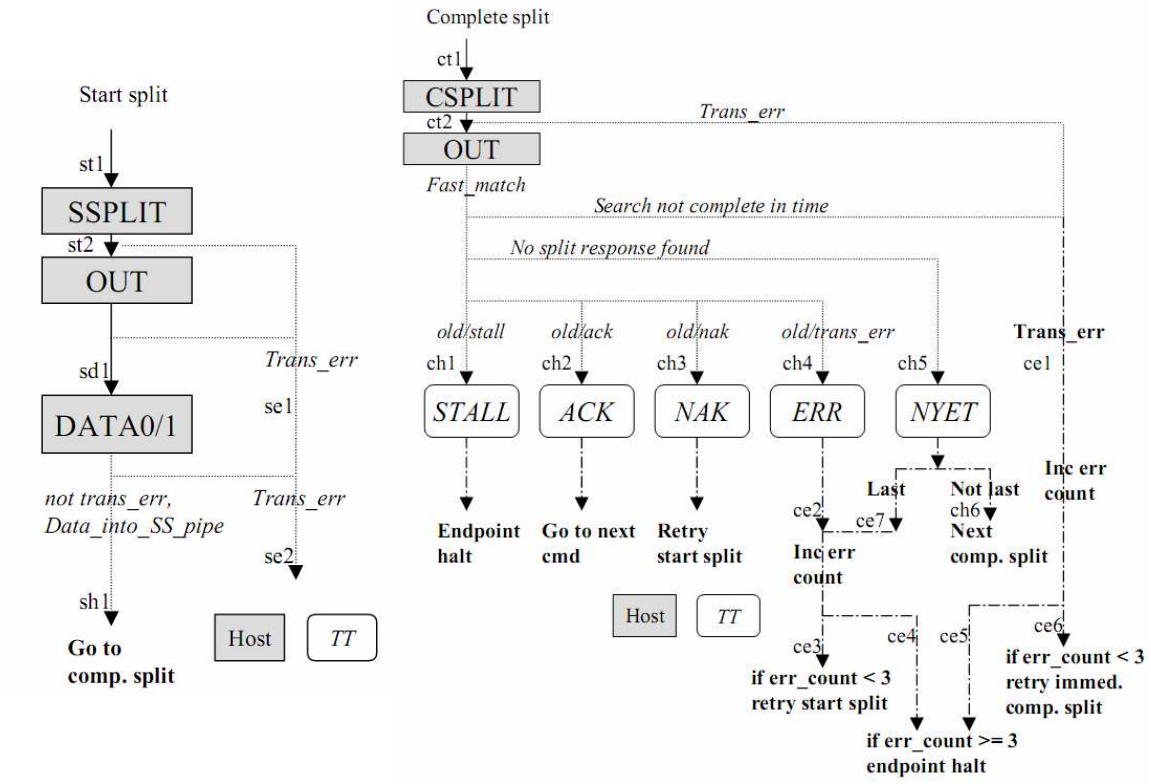
上行方向上，即 IN 型的传输：无论何时，只要 TT 收到超过 2 个字节的数据，就响应 HOST 的 C-Split，向 HOST 发回数据，并在 PID 中以 MDATA，DATA1/0 表明还有没有未传完的数据。如果还有数据，则以 MDATA 发送，否则以 DATA1/0 发送。HOST 在收到 MDATA 后，应继续在下一个微帧里发起 C-Split 传输，向 HUB 请求数据，知道收到一个 DATA0/1 的数据包。

下行方向上，即 OUT 型的传输：HOST 把一个大的数据包拆成最大 188 字节的若干个小包，在连续的多个微帧里用 S-Split 向 TT 发送数据。

如下图所示：

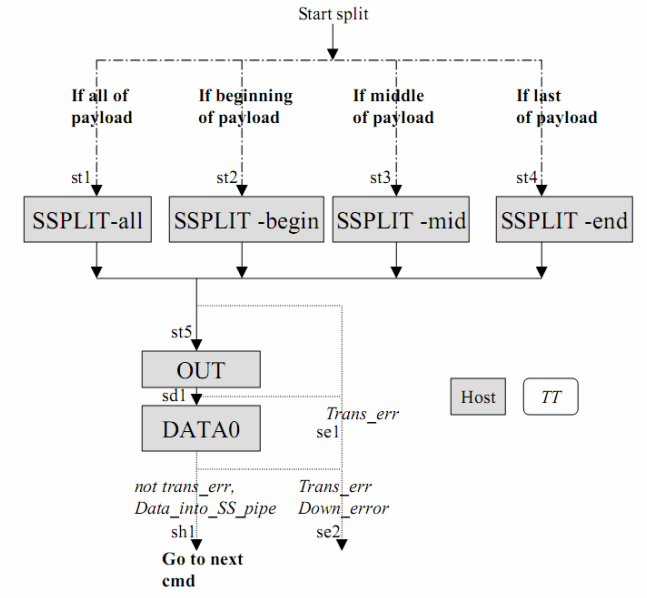


OUT 型中断分离传输



OUT 型的同步分离传输

OUT 型的同步分离传输只有 S-Split 阶段，而没有 C-Split 阶段，因为同步传输是不可靠的传输，HOST 不需要确认传输是否成功完成。



IN 型的同步分离传输