15

Hubs: the Link between Devices and the Host

Every USB peripheral must connect to a hub. As Chapter 1 explained, a hub is an intelligent device that provides attachment points for devices and manages each device's connection to the bus. Devices that plug directly into a PC connect to the root hub. Other devices connect to external hubs downstream from the root hub.

A hub's main jobs are managing its devices' connections and power and passing traffic to and from the host. Managing the connections includes helping to get newly attached devices up and communicating and blocking communications from misbehaving devices so they don't interfere with other communications on the bus. Managing power includes providing the requested bus current to attached devices. The hub's role in passing traffic to and from the host depends on the speed of the host, the device, and the

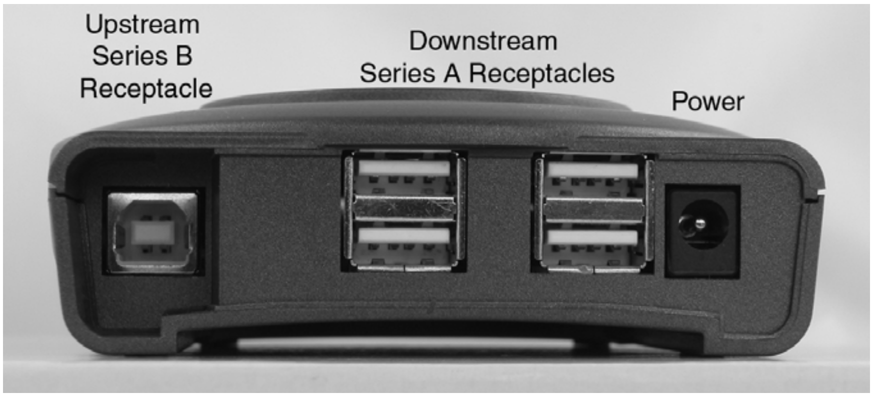

Figure 15-1: A hub has one upstream port and one or more downstream ports.

hubs between them. A hub may just repeat what it receives or it may convert the traffic to a different speed and manage transactions with the device.

This chapter presents essentials about hub communications. You don't need to know every detail about hubs in order to design a USB peripheral. Host applications and device drivers and device firmware don't have to know or care how many hubs are between the host and a device. But some understanding of what the hub does can help in understanding how devices are detected and communicate on the bus.

Hub Basics

Each external hub has one port, or attachment point, that connects in the upstream direction (toward the host) (Figure 15-1). This upstream port may connect directly to the host's root hub, or the port may connect to a downstream port on another external hub. Each hub also has one or more ports downstream from the host. Most downstream ports have a connector for attaching a cable. An exception is a hub that is part of a compound device whose ports connect to functions embedded in the device. Hubs with one, two, four, and seven downstream ports are common. A hub may be self-powered or bus-powered. As Chapter 16 explains, bus-powered hubs are limited because you can't attach high-power devices to them.

Every external hub has a hub repeater and a hub controller. (Figure 15-2). The hub repeater is responsible for passing USB traffic between the host's root hub or another upstream hub and whatever downstream devices are attached and enabled. The hub controller manages communications between the host and the hub repeater. State machines control the hub's response to events at the hub repeater and upstream and downstream ports. (The timing requirements are too strict to be handled by firmware.) A 2.0 hub also has one or more transaction translators and routing logic that enable low- and full-speed devices to communicate on a high-speed bus.

The host's root hub is a special case. The host controller performs many of the functions that the hub repeater and hub controller perform in an external hub, so a root hub may contain little more than routing logic and downstream ports.

The Hub Repeater

The hub repeater re-transmits, or repeats, the packets it receives, sending them on their way either upstream or downstream with minimal changes. The hub repeater also detects when a device is attached and removed, establishes the connection of a device to the bus, detects bus faults such as over-current conditions, and manages power to the device.

The hub repeater in a 2.0 hub has two modes of operation depending on the upstream bus speed. When the hub connects upstream to a full-speed bus segment, the repeater functions as a low- and full-speed repeater. When the hub connects upstream to a high-speed bus segment, the repeater functions as a high-speed repeater. The repeaters in 1.x hubs always function as low- and full-speed repeaters.

The Low- and Full-speed Repeater

The hub repeater in a 1.x hub handles low- and full-speed traffic. A 2.0 hub also uses this type of repeater when its upstream port connects to a full-speed bus. In this case, the 2.0 hub doesn't send or receive high-speed traffic but instead functions identically to a 1.x hub.

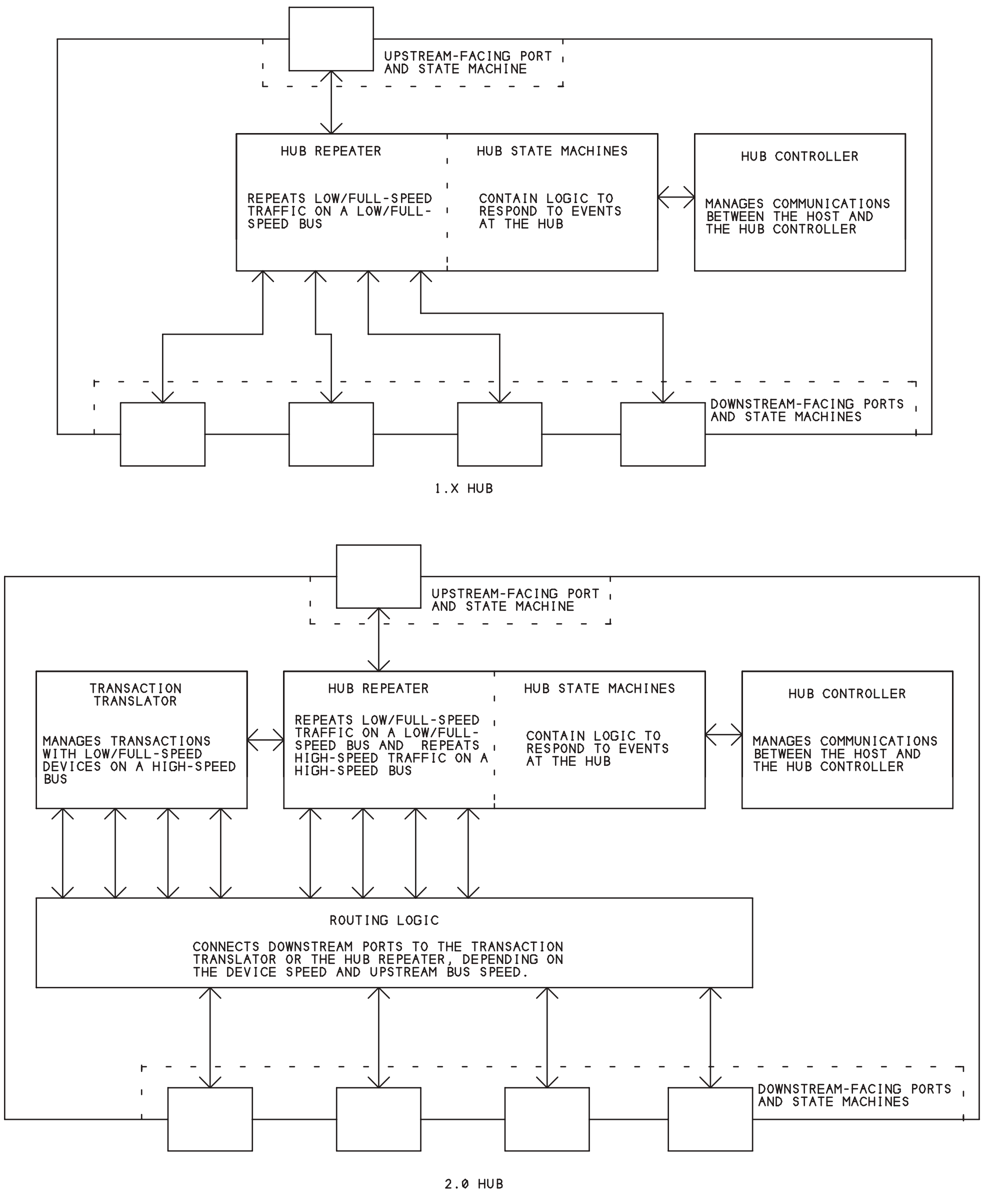
UPSTREAM-FACING PORT AND STATE MACHINE
HUB REPEATER
REPEATS LOW/FULL-SPEED TRAFFIC ON A LOW/FULL-SPEED BUS
HUB STATE MACHINES
CONTAIN LOGIC TO RESPOND TO EVENTS AT THE HUB
HUB CONTROLLER
MANAGES COMMUNICATIONS BETWEEN THE HOST AND THE HUB CONTROLLER
DOWNSTREAM-FACING PORTS AND STATE MACHINES
1.X HUB
UPSTREAM-FACING PORT AND STATE MACHINE
TRANSACTION TRANSLATOR
MANAGES TRANSACTIONS WITH LOW/FULL-SPEED DEVICES ON A HIGH-SPEED BUS
HUB REPEATER
REPEATS LOW/FULL-SPEED TRAFFIC ON A LOW/FULL-SPEED BUS AND REPEATS HIGH-SPEED TRAFFIC ON A HIGH-SPEED BUS
HUB STATE MACHINES
CONTAIN LOGIC TO RESPOND TO EVENTS AT THE HUB
HUB CONTROLLER
MANAGES COMMUNICATIONS BETWEEN THE HOST AND THE HUB CONTROLLER
ROUTING LOGIC
CONNECTS DOWNSTREAM PORTS TO THE TRANSACTION TRANSLATOR OR THE HUB REPEATER, DEPENDING ON THE DEVICE SPEED AND UPSTREAM BUS SPEED.
DOWNSTREAM-FACING PORTS AND STATE MACHINES
2.0 HUB

Figure 15-2: A 2.0 hub contains one or more transaction translators and routing logic that enable a hub on a high-speed bus to communicate with low- and full-speed devices. In a 1.x hub, the hub repeater is routed directly to the downstream ports.

A 1.x hub repeats all low- and full-speed packets received from the host (including data that has passed through one or more additional hubs) to all enabled, full-speed, downstream ports. Enabled ports include all ports with attached devices that are ready to receive communications from the hub. Devices with ports that aren't enabled include devices that the host controller has stopped communicating with due to errors or other problems, devices in the Suspend state, and devices that aren't yet ready to communicate because they have just been attached or are in the process of exiting the Suspend state.

The hub repeater doesn't translate, examine the contents of, or process the traffic to or from full-speed ports in any way. The hub repeater just regenerates the edges of the signal transitions and passes them on.

Low-speed devices never see full-speed traffic. A 1.x hub repeats only low-speed packets to low-speed devices. The hub identifies a low-speed packet by the PRE packet identifier that precedes the packet. The hub repeats the low-speed packets, and only these packets, to any enabled low-speed ports. The hub also repeats low-speed packets to its full-speed downstream ports, because a full-speed port may connect to a hub that in turn connects to a low-speed device. To give the hubs time to make their low-speed ports ready to receive data, the host adds a delay of at least four full-speed bit widths between the PRE packet and the low-speed packet.

Compared to full speed, traffic in a low-speed cable segment varies not only in speed, but also in edge rate and polarity. The hub nearest to a low-speed device uses low speed's edge rate and polarity when communicating with the device. When communicating upstream, the hub uses full-speed's faster edge rate and an inverted polarity compared to low speed. The hub repeater converts between the edge rates and polarities as needed. Chapter 18 has more on the signal polarities, and Chapter 19 has more about edge rates.

The High-speed Repeater

A 2.0 hub uses a high-speed repeater when the hub's upstream port connects to a high-speed bus segment. When this is the case, the hub sends and receives all upstream traffic at high speed, even if the traffic is to or from a low- or full-speed device. The path that traffic takes through a hub with a high-speed repeater depends on the speeds of the attached devices. Routing logic in the hub determines whether traffic to or from a downstream port passes through a transaction translator.

Unlike a low- and full-speed repeater, a high-speed repeater re-clocks received data to minimize accumulated jitter. In other words, instead of just repeating received transitions, a high-speed repeater extracts the data and uses its own local clock to time the transitions when retransmitting. The edge rate and polarity are unchanged. An elasticity buffer allows for small differences between the two clock frequencies. When the buffer is half full, the received data begins to be clocked out.

High-speed devices don't use the transaction translator. Traffic is routed from the receiving port on the hub, through the high-speed repeater, to the hub's transmitting port.

For traffic to and from low- and full-speed devices, the high-speed repeater communicates with the transaction translator that manages the transactions with the devices. Traffic received from upstream is routed to the high-speed repeater, then passes through the transaction translator, which communicates at the appropriate speed with the downstream ports. In the other direction, traffic from low- and full-speed devices is routed to the transaction translator, which processes the received data and takes appropriate action as described in the next section.

The Transaction Translator

Every 2.0 hub must have a transaction translator to manage communications with low- and full-speed devices. The transaction translator communicates upstream at high speed but enables 1.x devices to communicate at low and full speeds in exactly the same way as they do with 1.x hosts. The transaction translator stores received data and then forwards the data on toward its destination at a different speed.

The transaction translator frees bus time by enabling other bus communications to occur while a device is completing a low- or full-speed transaction.

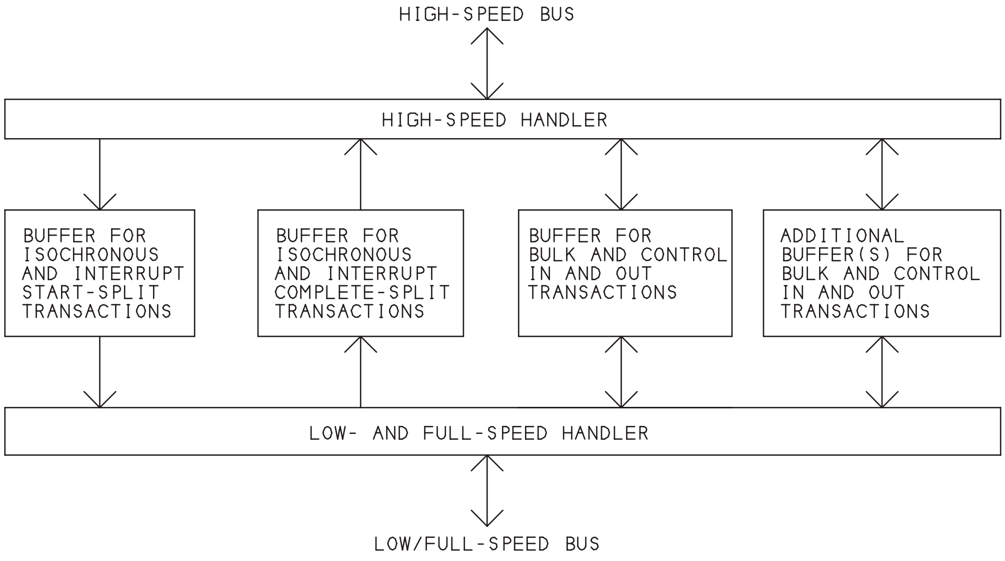
HIGH-SPEED BUS
HIGH-SPEED HANDLER
BUFFER FOR ISOCHRONOUS AND INTERRUPT START-SPLIT TRANSACTIONS
BUFFER FOR ISOCHRONOUS AND INTERRUPT COMPLETE-SPLIT TRANSACTIONS
BUFFER FOR BULK AND CONTROL IN AND OUT TRANSACTIONS
ADDITIONAL BUFFER(S) FOR BULK AND CONTROL IN AND OUT TRANSACTIONS
LOW- AND FULL-SPEED HANDLER
LOW/FULL-SPEED BUS

Figure 15-3: A transaction translator contains a high-speed handler for upstream traffic, buffers for storing information in split transactions, and a low- and full-speed handler for downstream traffic to low- and full-speed devices.

Transaction translators can also enable low- and full-speed devices to use more bandwidth than they would have on a shared 1.x bus.

Sections

The transaction translator contains three sections (Figure 15-3). The high-speed handler communicates with the host at high speed. The low/full-speed handler communicates with devices at low and full speeds. Buffers store data used in transactions with low- and full-speed devices. Each transaction translator has to have at least four buffers: one for interrupt and isochronous start-split transactions, one for interrupt and isochronous complete-split transactions, and two or more for control and bulk transfers.

Managing Split Transactions

When a 2.0 host on a high-speed bus wants to communicate with a low- or full-speed device, the host initiates a start-split transaction with the 2.0 hub that is nearest the device and communicating upstream at high speed. One or more start-split transactions contain the information the hub needs to complete the transaction with the device. The transaction translator stores the information received from the host and completes the start-split transaction with the host.

On completing a start-split transaction, the hub performs the function of a host controller in carrying out the transaction with the device. The transaction translator initiates the transaction in the token phase, sends data or stores returned data or status information as needed in the data phase, and sends or receives a status code as needed in the handshake phase. The hub uses low or full speed, as appropriate, in its communications with the device.

After the hub has had time to exchange data with the device, in all transactions except isochronous OUTs, the host initiates one or more complete-split transactions to retrieve the information returned by the device and stored in the transaction translator's buffer. The hub performs these transactions at high speed.

Figure 15-4 shows the transactions that make up a split transaction. Table 15-1 compares the structure and contents of transactions with low- and full-speed devices at different bus speeds.

In explaining how split transactions work, I'll start with bulk and control transfers, which don't have the timing constraints of interrupt and isochronous transfers. In the start-split transaction, the 2.0 host sends the start-split token packet (SSPLIT), followed by the usual low- or full-speed token packet, and any data packet destined for the device. The 2.0 hub that is nearest the device and communicating upstream at high speed returns ACK or NAK. The host is then free to use the bus for other transactions. The device knows nothing about the transaction yet.

On returning ACK in a start-split transaction, the hub has two responsibilities. The hub must complete the transaction with the device. And the hub must continue to handle any other bus traffic received from the host or other attached devices.

To complete the transaction, the hub converts the packet or packets received from the host to the appropriate speed, sends them to the device and stores the data or handshake returned by the device. Depending on the transac-

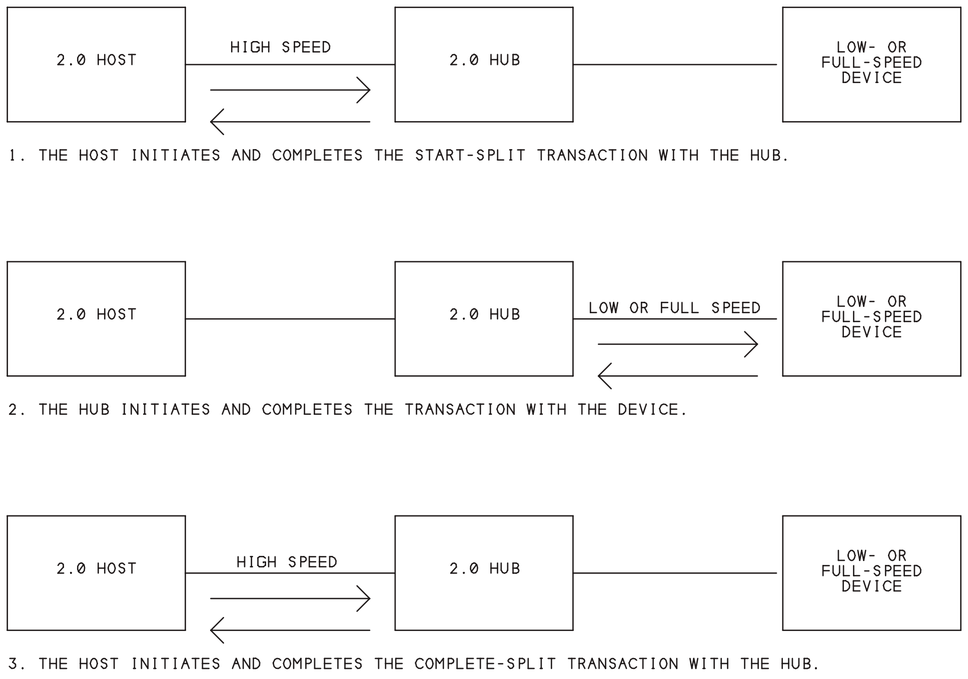
2.0 HOST
HIGH SPEED
2.0 HUB
LOW- OR FULL-SPEED DEVICE
1. THE HOST INITIATES AND COMPLETES THE START-SPLIT TRANSACTION WITH THE HUB.
2.0 HOST
2.0 HUB
LOW OR FULL SPEED
LOW- OR FULL-SPEED DEVICE
2. THE HUB INITIATES AND COMPLETES THE TRANSACTION WITH THE DEVICE.
2.0 HOST
HIGH SPEED
2.0 HUB
LOW- OR FULL-SPEED DEVICE
3. THE HOST INITIATES AND COMPLETES THE COMPLETE-SPLIT TRANSACTION WITH THE HUB.

Figure 15-4: In a transfer that uses split transactions, the host communicates at high speed with a 2.0 hub, and the hub communicates at low or full speed with the device. Isochronous transactions may use multiple start-split or complete-split transactions.

tion, the device may return data, a handshake, or nothing. For IN transactions, the hub returns a handshake to the device. To the device, the transaction has proceeded at the expected low or full speed and is now complete. The device has no knowledge that it's a split transaction. The host hasn't yet received the device's response.

While the hub is completing the transaction with the device, the host may initiate other bus traffic that the device's hub must handle as well. The two functions are handled by separate hardware modules within the hub. When the host thinks the hub has had enough time to complete the transaction with the device, the host begins a complete-split transaction with the hub.

In a complete-split transaction, the host sends a complete-split token packet (CSPLIT), followed by a low- or full-speed token packet to request the data

Table 15-1: When a low- or full-speed device has a transaction on a high-speed bus, the host uses start-split (SSPLIT) and complete-split (CSPLIT) transactions with the 2.0 hub nearest the device. The hub is responsible for completing the transaction at low or full speed and reporting back to the host.

Bus Speed	**Transaction Type**	**Transaction Phase**		
		Token	**Data**	**Handshake**
Low/Full-speed communications with the device	Setup, OUT	PRE if low speed, LS/FS token	PRE if low speed, data	status (except for isochronous)
	IN	PRE if low speed, LS/FS token	data or status	PRE if low speed, status (except for isochronous)
High-speed communications between the 2.0 hub and host in transactions with a low- or full-speed device	Setup, OUT (isochronous OUT has no CSPLIT transaction)	SSPLIT, LS/FS token	data	status (bulk and control only)
		CSPLIT, LS/FS token	–	status
	IN	SSPLIT, LS/FS token	–	status (bulk and control only)
		CSPLIT, LS/FS token)	data or status	–

or status information the hub has received from the device. The hub returns the information. The transfer is now complete at the host. The host doesn't return an ACK to the hub. If the hub doesn't have the packet ready to send, the hub returns a NYET status code, and the host retries later. The device has no knowledge of the complete-split transaction.

In split transactions in interrupt and isochronous transfers, the process is similar, but with more strictly defined timing. The goal is to transfer data to the host as soon as possible after the device has data available to send, and to transfer data to the device just before the device is ready to receive new data. To achieve this timing, isochronous transactions with large packets use multiple start splits or complete splits, transferring a portion of the data in each.

Unlike with bulk and control transfers, the start-split transactions in interrupt and isochronous transfers have no handshake phase, just the start-split token followed by an IN, OUT, or Setup token and data for OUT or Setup transactions.

In an interrupt transaction, the hub schedules the start split in the microframe just before the earliest time that the hub is expected to begin the transaction with the device. For example, assume that the microframes in a frame are numbered in sequence, Y0 through Y7. If the start split is in Y0, the transaction with the device may occur as early as Y1. The device may have data or a handshake response to return to the host as early as Y2. The results of previous transactions and bit stuffing can affect when the transaction with the device actually occurs, so the host schedules time for three complete-split transactions, in Y2, Y3, and Y4. If the hub doesn't yet have the information to return in a complete split, the hub returns a NYET status code and the host retries.

Full-speed isochronous transactions can transfer up to 1023 bytes. To ensure that the data transfers just in time, or as soon as the device has data to send or is ready to receive data, transactions with large packets use multiple start splits or complete splits, with up to 188 bytes of data in each. This is the maximum amount of full-speed data that fits in a microframe. A single transaction's data can require up to eight start-split or complete-split transactions.

In an isochronous IN transaction, the host schedules complete-split transactions in every microframe where the host expects that the device will have at least a portion of the data to return. Requesting the data in smaller chunks ensures that the host receives the data as quickly as possible. The host doesn't have to wait for all of the data to transfer from the device at full speed before beginning to retrieve the data.

In an isochronous OUT transaction, the host sends the data in one or more start-split transactions. The host schedules the transactions so the hub's buffer will never be empty but will contain as few bytes as possible. Each SPLIT packet contains bits to indicate the data's position in the low- or full-speed data packet (beginning, middle, end, or all). There is no complete-split transaction.

Bandwidth Use of Low- and Full-speed Devices

Because a 2.0 hub acts as a host controller in managing transactions, low- and full-speed devices share 1.x bandwidth only with devices that use the same transaction translator. So if two full-speed devices connect to their own 2.0 hubs on a high-speed bus, each device can use all of the full-speed bandwidth it wants. When the hub converts to high speed, the 1.x communications will use little of the high-speed bandwidth.

However, for bulk transactions, the extra transaction with the host in each split transaction can slow the rate of data transfer with a full-speed device on a busy bus that is also carrying high-speed bulk traffic.

Many hubs provide one transaction translator for all ports, but a single hub can also have a transaction translator for each port that connects to a low- or full-speed device.

The Hub Controller

The hub controller manages communications between the host and the hub. The communications include enumeration along with other communications and actions due to events at downstream ports.

As it does for all devices, the host enumerates a newly detected hub to find out its abilities. The hub descriptor retrieved during enumeration tells the host how many ports the hub has. After enumerating the hub, the host requests the hub to report whether there are any devices attached. If so, the host enumerates these as well.

The host finds out if a device is attached to a port by sending the hub-class request Get_Port_Status. This is similar to a Get_Status request, but sent to a hub with a port number in the Index field. The hub returns two 16-bit values that indicate whether a device is attached as well as other information, such as whether the device is low power or in the Suspend state.

The hub controller is also responsible for disabling any port that was responsible for loss of bus activity or babble. Loss of bus activity occurs when a packet doesn't end with the expected End-of-Packet signal. Babble occurs when a device continues to transmit beyond the End-of-Packet signal.

In addition to Endpoint 0, which all devices must have for control transfers, each hub must have a Status Change endpoint configured for interrupt IN transfers. The host polls this endpoint to find out if there have been any changes at the hub. On each poll, the hub controller returns either a NAK if there have been no changes, or data that indicates a specific port or the hub itself as the source of the change. If there is a change, the host sends requests to find out more about the change and to take whatever action is needed. For example, if the hub reports the attachment of a new device, the host attempts to enumerate it.

Speed

An external 2.0 hub's downstream ports must support all three speeds. In the upstream direction, if a 2.0 hub's upstream segment is high speed, the hub communicates at high speed. Otherwise, the hub communicates upstream at low and full speeds.

A 1.x hub's upstream port must support low- and full-speed communications. All downstream ports with connectors must support both low- and full-speed communications. 1.x hubs never support high speed.

Filtering Traffic according to Speed

Low-speed devices aren't capable of receiving full-speed data, so hubs don't repeat full-speed traffic to low-speed devices. This behavior is necessary because a low-speed device would try to interpret the full-speed traffic as low-speed data and might even mistakenly see what looks like valid data. Full- or high-speed data on a low-speed cable could also cause problems due to radiated electromagnetic interference (EMI). In the other direction, hubs receive and repeat any low-speed data upstream.

Low- and full-speed devices aren't capable of receiving high-speed data, so 2.0 hubs don't repeat high-speed traffic to these devices, including 1.x hubs.

Detecting Device Speed

On attachment, every device must support either low or full speed. A hub detects whether an attached device is low or full speed by detecting which

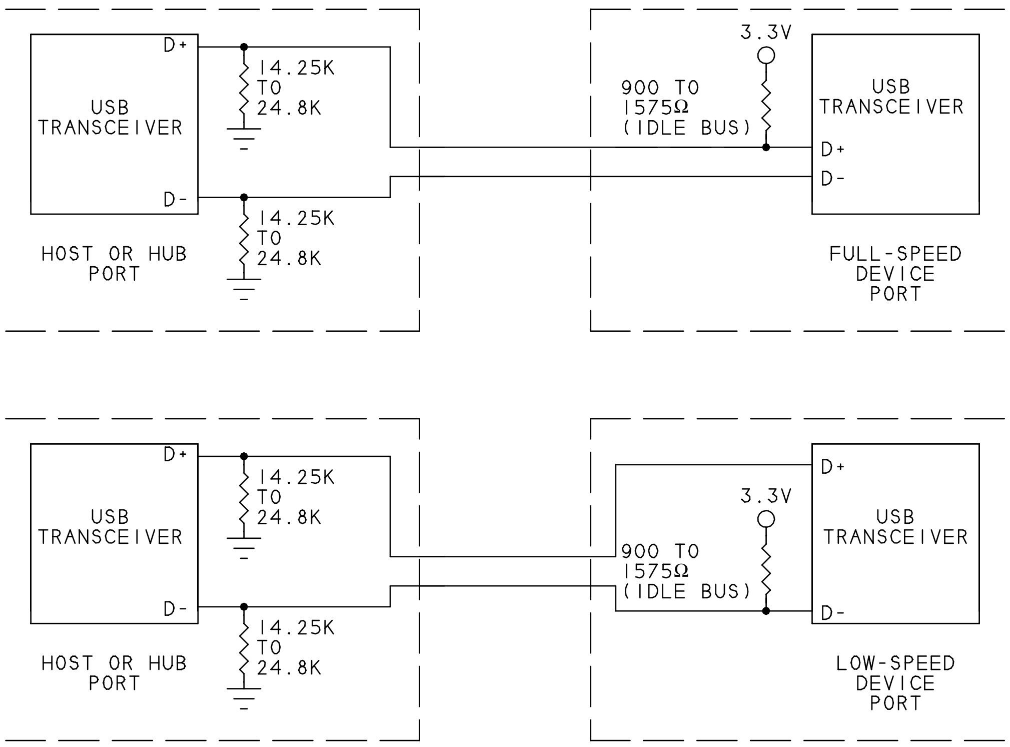

Figure 15-5: The device's port has a stronger pull-up than the hub's. The location of the pull-up tells the hub whether the device is low or full speed. High-speed devices are full speed at attachment.

signal line is more positive on an idle line. Figure 15-5 illustrates. As Chapter 4 explained, the hub has a pull-down resistor of 14.25 to 24.8 kilohms on each of the port's two signal lines, D+ and D-. A newly attached device has a pull-up resistor of 900 to 1575 ohms on either D+ for a full-speed device or D- for a low-speed device. When a device is attached to a port, the line with the pull-up is more positive than the hub's logic-high input threshold. The hub detects the voltage, assumes a device is attached, and detects the speed by which line is pulled up.

After detecting a full-speed device, a 2.0 hub determines whether the device supports high speed by using the high-speed detection handshake. The handshake occurs during the Reset state that the hub initiates during enumeration. If the handshake succeeds, the device removes its pull-up and communications are at high speed. A 1.x hub ignores the attempt to handshake, and the failure of the handshake informs the device that it must use full speed. Chapter 18 has more details about the handshake.

Maintaing an Idle Bus

Start-of-Frame packets keep full- and high-speed devices from entering the Suspend state on an otherwise idle bus. When there is no data on a full-speed bus, the host continues to send a Start-of-Frame packet once per frame, and all hubs pass these packets on to their full-speed devices. When there is no data on a high-speed bus, the host continues to send a Start-of-Frame packet once per microframe, and all hubs pass these packets on to their high-speed devices. A full-speed device that connects to a 2.0 hub that communicates upstream at high speed will also receive a Start-of-Frame once per frame.

Low-speed devices don't see the Start-of-Frame packets. Instead, at least once per frame, hubs must send their low-speed devices a low-speed End-of-Packet (EOP) signal (defined in Chapter 18). This signal functions as a keep-alive signal that keeps a device from entering the Suspend state on a bus with no low-speed activity. A host can also request a hub to suspend the bus at a single port. Chapter 16 has more on how hubs manage the Suspend state.

How Many Hubs in Series?

USB was designed for connecting to peripherals over short to moderate distances. But that hasn't stopped users from wondering just how far a USB peripheral can be from its host.

The USB 2.0 specification doesn't give a maximum length for cable segments, but the maximum allowed propagation delay limits the length to about 5 meters for full and high speed and 3 meters for low speed. You can increase the distance between a device and its host by using a series of hubs, each with a 5-meter cable.

The number of hubs you can connect in series is limited by the electrical properties of the hubs and cables and the resulting delays in propagating signals along the cable and through a hub. The limit is five hubs in series, with the hubs and the final device each using a 5-meter cable. The result is a device that is 30 meters from its host. If the device is low speed, the limit is 28 meters because the device's cable can be no more than 3 meters. Chapter 19 has more about extending the distance between a USB device and its host beyond these limits.

The Hub Class

Hubs are members of the Hub class, which is the only class defined in the main USB specification.

Hub Descriptors

A 1.x hub has a series of five descriptors: device, hub class, configuration, interface, and endpoint. A 2.0 hub has more descriptors because it must support all speeds and because the hub may offer a choice of using one or multiple transaction translators.

A 2.0 hub's descriptors include the device_qualifier descriptor and the other_speed_configuration_descriptor required for all high-speed-capable devices. The device_qualifier descriptor contains an alternate value for bDeviceProtocol in the device descriptor. The hub uses the alternate value when it switches between high and full speeds.

The other_speed_configuration_descriptor specifies the number of interfaces supported by the configuration not currently in use and is followed by the subordinate descriptors for that configuration. A configuration that supports multiple transaction translators has two interface descriptors: one for use with a single transaction translator and an alternate setting for use with multiple transaction translators. The bInterfaceProtocal field specifies whether the interface setting supports one or multiple transaction translators.

Hub Values for the Standard Descriptors

The USB specification assigns class-specific values for some parameters in a hub's device, and interface descriptors. The specification also defines the endpoint descriptor for the hub's status-change endpoint:

The device descriptor has these values:

> bDeviceClass: HUB_CLASSCODE (09h).
>
> bDeviceSubClass: 0.
>
> bDeviceProtocol: 0 for low/full speed, 1 for high speed when the hub supports a single transaction translator, 2 for high speed when the hub supports multiple transaction translators.

These fields also apply to the Device_Qualifier_Descriptor in 2.0 hubs.

The interface descriptor has these values:

> bNumEndpoints: 1.
>
> bInterfaceClass: HUB_CLASSCODE (09h).
>
> bInterfaceSubClass: 0.
>
> bInterfaceProtocol: 0 for a low/full speed hub or a high-speed hub with a single transaction translator. For a hub that supports single and multiple transaction translators, 1 indicates a single transaction translator, and 2 indicates multiple transaction translators.

The endpoint descriptor for the status change endpoint has these values:

> bEndpointAddress: implementation-dependent, with bit 7 (direction) = IN (01h).
>
> wMaxPacketSize: implementation-dependent.
>
> bmAttributes: Transfer Type = Interrupt.
>
> bInterval: FFh for full speed, 0Ch for high speed.

The Hub Descriptor

Each hub must have a hub-class descriptor that contains the following fields:

Identifying the Descriptor

bDescLength. The number of bytes in the descriptor.

bDescriptorType. Hub Descriptor, 29h.

Hub Description

bNbrPorts. The number of downstream ports the hub supports.

wHubCharacteristics:

Bits 1 and 0 specify the power-switching mode. 00=Ganged; all ports are powered together. 01=Ports are powered individually. 1X: used only on 1.0 hubs with no power switching.

Bit 2 indicates whether the hub is part of a compound device (1) or not (0).

Bits 4 and 3 are the Overcurrent Protection mode. 00 = Global protection and reporting. 01=Protection and reporting for each port. 1X = No protection and reporting (for bus-powered hubs only).

Bits 6 and 5 are the Transaction Translator Think Time. These bits indicate the maximum number of full-speed bit times required between transactions on a low- or full-speed downstream bus. 00 = 8; 01 = 16; 10 = 24; 11 = 32. Applies to 2.0 hubs only.

Bit 7 indicates whether the hub supports Port Indicators (1) or not (0). Applies to 2.0 hubs only.

Bits 8 through 15 are reserved.

bPwrOn2PwrGood. The maximum delay between beginning the power-on sequence on a port and when power is available on the port. The value is in units of 2 milliseconds. (Set to 100 for a 200-millisecond delay.)

bHubContrCurrent. The maximum current required by the hub controller's electronics only, in milliamperes.

DeviceRemovable. Indicates whether the device(s) attached to the hub's ports are removable (0) or not (1). The number of bits in this value equals the number of ports on the hub + 1. Bit 0 is reserved. Bit 1 is for Port 1, bit 2 is for Port 2, and so on up.

PortPowerCtrlMask. All bits should be 1. This field is only for compatibility with 1.0 software. Each port has one bit, and the field should be padded with additional 1s so that the field's size in bits is a multiple of 8.

Table 15-2: The 2.0 hub class has 12 class-specific requests, while the 1.x hub class has 9. Many are hub-specific variants of USB's standard requests.

Request	**USB Versions**	**bRequest**	**Data source**	**wValue**	**wIndex**	**Data Length (bytes) (Data stage)**	**Data (in the Data stage)**
Clear Hub Feature	all	Clear_ Feature	no Data stage	feature	0	–	–
Clear Port Feature	all	Clear_ Feature	no Data stage	feature	port	–	–
Clear TT Buffer	2.0 only	Clear_TT _Buffer	no Data stage	device address, endpoint #	TT_port	–	–
Get Bus State	1.x only	Get_State	Hub	0	port	1	per-port bus state
Get Hub Descriptor	all	Get_ Descriptor	Hub	descriptor type & index	0 or language ID	descriptor length	descriptor
Get Hub Sta-tus	all	Get_ Status	Hub	0	0	4	hub status and change indicators
Get Port Status	all	Get_ Status	Hub	0	port	4	port status and change indicators
Get TT State	2.0 only	Get_TT State	hub	TT flags	port	TT state, length	TT state
Reset TT	2.0 only	Reset_TT	no Data stage	0	port	–	–
Set Hub Descriptor (optional)	all	Set_ Descriptor	host	descriptor type and index	0 or language ID	descriptor length	descriptor length
Set Hub Feature	all	Set_ Feature	no Data stage	feature	0	–	–
Set Port Feature	all	Set_ Feature	no Data stage	feature	port	–	–
Stop TT	2.0 only	Stop_TT	no Data stage	0	port	–	–

Hub-class Requests

All hubs accept or return data for seven of the USB's eleven standard requests. Some 2.0 hubs support an additional request. Of the other standard requests, one is optional and the other two are undefined for hubs. Like all devices, hubs must return STALL for unsupported requests.

Hubs respond in the standard way to Clear_Feature, Get_Configuration, Get_Descriptor, Get_Status, Set_Address, Set_Configuration, and Set_Feature requests. Set_Descriptor is optional and should return STALL if not supported. Only 2.0 hubs that support multiple transaction translators support Get_Interface and Set_Interface. A hub can't have an isochronous endpoint, so Synch_Frame is undefined for hubs.

The hub class defines eight hub-specific requests that build on the standard requests with hub-specific values. For example, a Get_Status request directed to a hub with Index = 0 causes the hub to return a value in a Data packet indicating whether the hub is using an external power supply and whether an over-current condition exists.

Table 15-2 shows the hub-specific requests. One request from the 1.x specification, Get_Bus_State, isn't included in the 2.0 spec. This request enables the host to read the states of D+ and D- at a specified port on the hub.

The host uses many of the hub-specific requests to monitor and control the status of the hub and its ports. Get_Hub_Status reads status bits in a hub. Set_Hub_Feature and Clear_Hub_Feature set and clear status bits in a hub. Table 15-3 shows the bits and their meanings. In a similar way, Get_Port_Status, Set_Port_Feature, and Clear_Port_Feature enable the host to read and control status bits for a selected port in a hub. Table 15-4 shows the bits and their meanings.

In 2.0 hubs, Set_Port_Feature can place a port in one of five Test Modes. Chapter 18 has more about these modes.

The four new requests in the 2.0 spec all relate to monitoring and controlling the transaction translator (TT). The requests enable the host to clear a buffer in the TT, stop the TT, retrieve the state of a stopped TT using a vendor-specific format, and restart the TT by resetting it.

Table 15-3: The host can monitor and control Status bits in a hub using Get_Hub_Status, Set_Hub_Feature, and Clear_Hub_Feature.

Field	Bit	Status Indicator	Meaning (0 state/1 state)
Hub Status	0	HUB_LOCAL_POWER	Local power supply is good/not active.
	1	HUB_OVER_CURRENT	An over-current condition exists/does not exist.
	2-15	reserved	Returns 0 when read.
Hub Change	0	C_HUB_LOCAL_POWER	Local power status has not changed/changed.
	1	C_HUB_OVER_CURRENT	Over-current status has not changed/changed.
	2-15	reserved	Returns 0 when read.

Port Indicators

The USB 2.0 specification defines optional indicators to indicate port status to the user. Many hubs have status LEDs. The specification assigns standard meanings to the LEDs' colors and blinking properties. Bit 7 in the wHub-Characteristics field in the hub descriptor indicates whether a hub has port indicators.

Each downstream port on a hub can have an indicator, which can be either a single bi-color green/amber LED or a separate LED for each color. The indicator tells the state of the hub's port, not the attached device. These are the meanings of the indicators to the user:

Green: fully operational
Amber: error condition
Blinking off/green: software attention required
Blinking off/amber: hardware attention required
Off: not operational

Table 15-4: The host can monitor and control Status bits at a port using Get_Port_Status, Set_Port_Feature, and Clear_Port_Feature.

Field	Bit	Status Indicator	Meaning (0 state/1 state)
Port Status	0	PORT_CONNECTION	A device is not present/present.
	1	PORT_ENABLE	The port is disabled/enabled.
	2	PORT_SUSPEND	The port is not/is in the Suspend state.
	3	PORT_OVERCURRENT	An over-current condition exists/does not exist.
	4	PORT_RESET	The hub is not/is asserting Reset at the port.
	5-7	reserved	Returns 0 when read.
	8	PORT_POWER	The port is/is not in the Powered Off state.
	9	PORT_LOW_SPEED	The attached device is full or high speed/low speed.
	10	PORT_HIGH_SPEED	The attached device is full speed/high speed. (2.0 hubs only)
	11	PORT_TEST	The port is not/is in the Port Test mode. (2.0 hubs only)
	12	PORT_INDICATOR	Port indicator displays default/software controlled colors. (2.0 hubs only)
	13-15	reserved	Returns 0 when read.
Port Status Change	0	C_PORT_CONNECTION	Connect status has not changed/changed.
	1	C_PORT_ENABLE	A Port Error condition does not/does exist.
	2	C_PORT_SUSPEND	Resume signaling is not/is complete.
	3	C_PORT_OVERCURRENT	The over-current condition has not/has changed.
	4	C_PORT_RESET	Reset processing is not/is complete.
	5-15	reserved	Returns 0 when read.