Table 5-3 lists information about different-sized high-speed control transfers and the maximum number of transfers possible in a microframe. This table was generated assuming that there is one Data stage transaction and that the Data stage has a zero-length status stage. The table illustrates the possible power of two data payloads less than or equal to the allowable maximum data payload size. The table does not include the overhead associated with bit stuffing.

Table 5-3. High-speed Control Transfer Limits

	Protocol Overhead (173 bytes)		(Based on 480Mb/s and 8 bit interpacket gap, 88 bit min bus turnaround, 32 bit sync, 8 bit EOP: (9x4 SYNC bytes, 9 PID bytes, 6 EP/ADDR+CRC,6 CRC16, 8 Setup data, 9x(1+11) byte interpacket delay (EOP, etc.))			
	Data Payload	**Max Bandwidth (bytes/second)**	**Microframe Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/ Microframe Useful Data**
	1	344000	2%	43	18	43
	2	672000	2%	42	150	84
	4	1344000	2%	42	66	168
	8	2624000	2%	41	79	328
	16	4992000	3%	39	129	624
	32	9216000	3%	36	120	1152
	64	15872000	3%	31	153	1984
Max		60000000				7500

5.5.5 Control Transfer Data Sequences

Control transfers require that a Setup bus transaction be sent from the host to a device to describe the type of control access that the device should perform. The Setup transaction is followed by zero or more control Data transactions that carry the specific information for the requested access. Finally, a Status transaction completes the control transfer and allows the endpoint to return the status of the control transfer to the client software. After the Status transaction for a control transfer is completed, the host can advance to the next control transfer for the endpoint. As described in Section 5.5.4, each control transaction and the next control transfer will be moved over the bus at some Host Controller implementation-defined time.

The endpoint can be busy for a device-specific time during the Data and Status transactions of the control transfer. During these times when the endpoint indicates it is busy (refer to Chapter 8 and Chapter 9 for details), the host will retry the transaction at a later time.

If a Setup transaction is received by an endpoint before a previously initiated control transfer is completed, the device must abort the current transfer/operation and handle the new control Setup transaction. A Setup transaction should not normally be sent before the completion of a previous control transfer. However, if a transfer is aborted, for example, due to errors on the bus, the host can send the next Setup transaction prematurely from the endpoint's perspective.

After a halt condition is encountered or an error is detected by the host, a control endpoint is allowed to recover by accepting the next Setup PID; i.e., recovery actions via some other pipe are not required for control endpoints. For the Default Control Pipe, a device reset will ultimately be required to clear the halt or error condition if the next Setup PID is not accepted.

The USB provides robust error detection and recovery/retransmission for errors that occur during control transfers. Transmitters and receivers can remain synchronized with regard to where they are in a control transfer and recover with minimum effort. Retransmission of Data and Status packets can be detected by a receiver via data retry indicators in the packet. A transmitter can reliably determine that its corresponding receiver has successfully accepted a transmitted packet by information returned in a handshake to the packet. The protocol allows for distinguishing a retransmitted packet from its original packet except for a control Setup packet. Setup packets may be retransmitted due to a transmission error; however, Setup packets cannot indicate that a packet is an original or a retried transmission.

5.6 Isochronous Transfers

In non-USB environments, isochronous transfers have the general implication of constant-rate, error-tolerant transfers. In the USB environment, requesting an isochronous transfer type provides the requester with the following:

- Guaranteed access to USB bandwidth with bounded latency
- Guaranteed constant data rate through the pipe as long as data is provided to the pipe
- In the case of a delivery failure due to error, no retrying of the attempt to deliver the data

While the USB isochronous transfer type is designed to support isochronous sources and destinations, it is not required that software using this transfer type actually be isochronous in order to use the transfer type. Section 5.12 presents more detail on special considerations for handling isochronous data on the USB.

5.6.1 Isochronous Transfer Data Format

The USB imposes no data content structure on communication flows for isochronous pipes.

5.6.2 Isochronous Transfer Direction

An isochronous pipe is a stream pipe and is, therefore, always uni-directional. An endpoint description identifies whether a given isochronous pipe's communication flow is into or out of the host. If a device requires bi-directional isochronous communication flow, two isochronous pipes must be used, one in each direction.

5.6.3 Isochronous Transfer Packet Size Constraints

An endpoint in a given configuration for an isochronous pipe specifies the maximum size data payload that it can transmit or receive. The USB System Software uses this information during configuration to ensure that there is sufficient bus time to accommodate this maximum data payload in each (micro)frame. If there is sufficient bus time for the maximum data payload, the configuration is established; if not, the configuration is not established.

The USB limits the maximum data payload size to 1,023 bytes for each full-speed isochronous endpoint. High-speed endpoints are allowed up to 1024-byte data payloads. A high speed, high bandwidth endpoint specifies whether it requires two or three transactions per microframe. Table 5-4 lists information about different-sized full-speed isochronous transactions and the maximum number of transactions possible in a frame. The table is shaded to indicate that a full-speed isochronous endpoint (with a non-zero *wMaxpacket* size) must not be part of a default interface setting. The table does not include the overhead associated with bit stuffing.

Table 5-4. Full-speed Isochronous Transaction Limits

	Protocol Overhead (9 bytes)	(2 SYNC bytes, 2 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 1-byte interpacket delay)				
	Data Payload	Max Bandwidth(bytes/ second)	Frame Bandwidth per Transfer	Max Transfers	Bytes Remaining	Bytes/Frame Useful Data
	1	150000	1%	150	0	150
	2	272000	1%	136	4	272
	4	460000	1%	115	5	460
	8	704000	1%	88	4	704
	16	960000	2%	60	0	960
	32	1152000	3%	36	24	1152
	64	1280000	5%	20	40	1280
	128	1280000	9%	10	130	1280
	256	1280000	18%	5	175	1280
	512	1024000	35%	2	458	1024
	1023	1023000	69%	1	468	1023
Max		1500000				1500

Table 5-5 lists information about different-sized high-speed isochronous transactions and the maximum number of transactions possible in a microframe. The table is shaded to indicate that a high-speed isochronous endpoint must not be part of a default interface setting. The table does not include the overhead associated with bit stuffing.

Any given transaction for an isochronous pipe need not be exactly the maximum size specified for the endpoint. The size of a data payload is determined by the transmitter (client software or function) and can vary as required from transaction to transaction. The USB ensures that whatever size is presented to the Host Controller is delivered on the bus. The actual size of a data payload is determined by the data transmitter and may be less than the prenegotiated maximum size. Bus errors can change the actual packet size seen by the receiver. However, these errors can be detected by either CRC on the data or by knowledge the receiver has about the expected size for any transaction.

Table 5-5. High-speed Isochronous Transaction Limits

	Protocol Overhead		(Based on 480Mb/s and 8 bit interpacket gap, 88 bit min bus turnaround, 32 bit sync, 8 bit EOP: (2x4 SYNC bytes, 2 PID bytes, 2 EP/ADDR+addr+CRC5, 2 CRC16, and a 2x(1+11)) byte interpacket delay (EOP, etc.))			
	Data Payload	**Max Bandwidth (bytes/second)**	**Microframe Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/ MicroFrame Useful Data**
	1	1536000	1%	192	12	192
	2	2992000	1%	187	20	374
	4	5696000	1%	178	24	712
	8	10432000	1%	163	2	1304
	16	17664000	1%	138	48	2208
	32	27392000	1%	107	10	3424
	64	37376000	1%	73	54	4672
	128	46080000	2%	45	30	5760
	256	51200000	4%	25	150	6400
	512	53248000	7%	13	350	6656
	1024	57344000	14%	7	66	7168
	2048	49152000	28%	3	1242	6144
	3072	49152000	41%	2	1280	6144
Max		60000000				7500

All device default interface settings must not include any isochronous endpoints with non-zero data payload sizes (specified via *wMaxPacketSize* in the endpoint descriptor). Alternate interface settings may specify non-zero data payload sizes for isochronous endpoints. If the isochronous endpoints have a large data payload size, it is recommended that additional alternate configurations or interface settings be used to specify a range of data payload sizes. This increases the chance that the device can be used successfully in combination with other USB devices.

5.6.4 Isochronous Transfer Bus Access Constraints

Isochronous transfers can only be used by full-speed and high-speed devices.

The USB requires that no more than 90% of any frame be allocated for periodic (isochronous and interrupt) transfers for full-speed endpoints. High-speed endpoints can allocate at most 80% of a microframe for periodic transfers.

An isochronous endpoint must specify its required bus access period. Full-/high-speed endpoints must specify a desired period as ($2^{\text{bInterval}-1}$) x F, where bInterval is in the range one to (and including) 16 and F is 125 µs for high-speed and 1ms for full-speed. This allows full-/high-speed isochronous transfers to have rates slower than one transaction per (micro)frame. However, an isochronous endpoint must be prepared to handle poll rates faster than the one specified. A host must not issue more than 1 transaction in a (micro)frame for an isochronous endpoint unless the endpoint is high-speed, high-bandwidth (see below). An isochronous IN endpoint must return a zero-length packet whenever data is requested at a faster interval than the specified interval and data is not available.

A high-speed endpoint can move up to 3072 bytes per microframe (or 192 Mb/s). A high-speed isochronous endpoint that requires more than 1024 bytes per period is called a high-bandwidth endpoint. A high-bandwidth endpoint uses multiple transactions per microframe. A high-bandwidth endpoint must specify a period of 1x125 µs (i.e., a *bInterval* value of 1). See Section 5.9 for more information about the details of multiple transactions per microframe for high-bandwidth high-speed endpoints.

Errors on the bus or delays in operating system scheduling of client software can result in no packet being transferred for a (micro)frame. An error indication should be returned as status to the client software in such a case. A device can also detect this situation by tracking SOF tokens and noticing a disturbance in the specified bus access period pattern.

The bus frequency and (micro)frame timing limit the maximum number of successful isochronous transactions within a (micro)frame for any USB system to less than 151 full-speed one-byte data payloads and less than 193 high-speed one-byte data payloads. A Host Controller, for various implementation reasons, may not be able to provide the theoretical maximum number of isochronous transactions per (micro)frame.

5.6.5 Isochronous Transfer Data Sequences

Isochronous transfers do not support data retransmission in response to errors on the bus. A receiver can determine that a transmission error occurred. The low-level USB protocol does not allow handshakes to be returned to the transmitter of an isochronous pipe. Normally, handshakes would be returned to tell the transmitter whether a packet was successfully received or not. For isochronous transfers, timeliness is more important than correctness/retransmission, and, given the low error rates expected on the bus, the protocol is optimized by assuming transfers normally succeed. Isochronous receivers can determine whether they missed data during a (micro)frame. Also, a receiver can determine how much data was lost. Section 5.12 describes these USB mechanisms in more detail.

An endpoint for isochronous transfers never halts because there is no handshake to report a halt condition. Errors are reported as status associated with the IRP for an isochronous transfer, but the isochronous pipe is not halted in an error case. If an error is detected, the host continues to process the data associated with the next (micro)frame of the transfer. Only limited error detection is possible because the protocol for isochronous transactions does not allow per-transaction handshakes.

5.7 Interrupt Transfers

The interrupt transfer type is designed to support those devices that need to send or receive data infrequently but with bounded service periods. Requesting a pipe with an interrupt transfer type provides the requester with the following:

- Guaranteed maximum service period for the pipe
- Retry of transfer attempts at the next period, in the case of occasional delivery failure due to error on the bus

5.7.1 Interrupt Transfer Data Format

The USB imposes no data content structure on communication flows for interrupt pipes.

5.7.2 Interrupt Transfer Direction

An interrupt pipe is a stream pipe and is therefore always uni-directional. An endpoint description identifies whether a given interrupt pipe's communication flow is into or out of the host.

5.7.3 Interrupt Transfer Packet Size Constraints

An endpoint for an interrupt pipe specifies the maximum size data payload that it will transmit or receive. The maximum allowable interrupt data payload size is 64 bytes or less for full-speed. High-speed endpoints are allowed maximum data payload sizes up to 1024 bytes. A high speed, high bandwidth endpoint specifies whether it requires two or three transactions per microframe. Low-speed devices are limited to eight bytes or less maximum data payload size. This maximum applies to the data payloads of the data packets; i.e., the size specified is for the data field of the packet as defined in Chapter 8, not including other protocol-required information. The USB does not require that data packets be exactly the maximum size; i.e., if a data packet is less than the maximum, it does not need to be padded to the maximum size.

All Host Controllers are required to support maximum data payload sizes from 0 to 64 bytes for full-speed interrupt endpoints, from 0 to 8 bytes for low-speed interrupt endpoints, and from 0 to 1024 bytes for high-speed interrupt endpoints. See Section 5.9 for more information about the details of multiple transactions per microframe for high bandwidth high-speed endpoints. No Host Controller is required to support larger maximum data payload sizes.

The USB System Software determines the maximum data payload size that will be used for an interrupt pipe during device configuration. This size remains constant for the lifetime of a device's configuration. The USB System Software uses the maximum data payload size determined during configuration to ensure that there is sufficient bus time to accommodate this maximum data payload in its assigned period. If there is sufficient bus time, the pipe is established; if not, the pipe is not established. However, the actual size of a data payload is still determined by the data transmitter and may be less than the maximum size.

An endpoint must always transmit data payloads with a data field less than or equal to the endpoint's *wMaxPacketSize* value. A device can move data via an interrupt pipe that is larger than *wMaxPacketSize*. A software client can accept this data via an IRP for the interrupt transfer that requires multiple bus transactions without requiring an IRP-complete notification per transaction. This can be achieved by specifying a buffer that can hold the desired data size. The size of the buffer is a multiple of *wMaxPacketSize* with some remainder. The endpoint must transfer each transaction except the last as *wMaxPacketSize* and the last transaction is the remainder. The multiple data transactions are moved over the bus at the period established for the pipe.

When an interrupt transfer involves more data than can fit in one data payload of the currently established maximum size, all data payloads are required to be maximum-sized except for the last data payload, which will contain the remaining data. An interrupt transfer is complete when the endpoint does one of the following:

- Has transferred exactly the amount of data expected
- Transfers a packet with a payload size less than *wMaxPacketSize* or transfers a zero-length packet

When an interrupt transfer is complete, the Host Controller retires the current IRP and advances to the next IRP. If a data payload is received that is larger than expected, the interrupt IRP will be aborted/retired and the pipe will stall future IRPs until the condition is corrected and acknowledged.

All high-speed device default interface settings must not include any interrupt endpoints with a data payload size (specified via *wMaxPacketSize* in the endpoint descriptor) greater than 64 bytes. Alternate interface settings may specify larger data payload sizes for interrupt endpoints. If the interrupt endpoints have a large data payload size, it is recommended that additional configurations or alternate interface settings be used to specify a range of data payload sizes. This increases the chances that the device can be used successfully in combination with other USB devices.

5.7.4 Interrupt Transfer Bus Access Constraints

Interrupt transfers can be used by low-speed, full-speed, and high-speed devices. High-speed endpoints can be allocated at most 80% of a microframe for periodic transfers. The USB requires that no more than 90% of any frame be allocated for periodic (isochronous and interrupt) full-/low-speed transfers.

The bus frequency and (micro)frame timing limit the maximum number of successful interrupt transactions within a (micro)frame for any USB system to less than 108 full-speed one-byte data payloads, or less than 10 low-speed one-byte data payloads, or to less than 134 high-speed one-byte data payloads. A Host Controller, for various implementation reasons, may not be able to provide the above maximum number of interrupt transactions per (micro)frame.

Table 5-6 lists information about different low-speed interrupt transactions and the maximum number of transactions possible in a frame. Table 5-7 lists similar information for full-speed interrupt transactions. Table 5-8 lists similar information for high-speed interrupt transactions. The shaded portion of Table 5-8 indicates the data payload sizes of a high-speed interrupt endpoint that must not be part of a default interface setting. The tables do not include the overhead associated with bit stuffing.

Table 5-6. Low-speed Interrupt Transaction Limits

	Protocol Overhead (19 bytes)		(5 SYNC bytes, 5 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 5-byte interpacket delay)			
	Data Payload	**Max Bandwidth (bytes/second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame Useful Data**
	1	9000	11%	9	7	9
	2	16000	11%	8	19	16
	4	32000	12%	8	3	32
	8	48000	14%	6	25	48
Max		187500				187

Table 5-7. Full-speed Interrupt Transaction Limits

	Protocol Overhead (13 bytes)	(3 SYNC bytes, 3 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 3-byte interpacket delay)				
	Data Payload	**Max Bandwidth (bytes/second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame Useful Data**
	1	107000	1%	107	2	107
	2	200000	1%	100	0	200
	4	352000	1%	88	4	352
	8	568000	1%	71	9	568
	16	816000	2%	51	21	816
	32	1056000	3%	33	15	1056
	64	1216000	5%	19	37	1216
Max		1500000				1500

Table 5-8. High-speed Interrupt Transaction Limits

	Protocol Overhead	(Based on 480Mb/s and 8 bit interpacket gap, 88 bit min bus turnaround, 32 bit sync, 8 bit EOP: (3x4 SYNC bytes, 3 PID bytes, 2 EP/ADDR+CRC bytes, 2 CRC16 and a 3x(1+11) byte interpacket delay(EOP, etc.))				
	Data Payload	**Max Bandwidth (bytes/second)**	**Microframe Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/ Microframe Useful Data**
	1	1064000	1%	133	52	133
	2	2096000	1%	131	33	262
	4	4064000	1%	127	7	508
	8	7616000	1%	119	3	952
	16	13440000	1%	105	45	1680
	32	22016000	1%	86	18	2752
	64	32256000	2%	63	3	4032
	128	40960000	2%	40	180	5120
	256	49152000	4%	24	36	6144
	512	53248000	8%	13	129	6656
	1024	49152000	14%	6	1026	6144
	2048	49152000	28%	3	1191	6144
	3072	49152000	42%	2	1246	6144
Max		60000000				7500

An endpoint for an interrupt pipe specifies its desired bus access period. A full-speed endpoint can specify a desired period from 1 ms to 255 ms. Low-speed endpoints are limited to specifying only 10 ms to 255 ms. High-speed endpoints can specify a desired period $(2^{bInterval-1})$x125 µs, where *bInterval* is in the range 1 to (including) 16. The USB System Software will use this information during configuration to determine a period that can be sustained. The period provided by the system may be shorter than that desired by the device up to the shortest period defined by the USB (125 µs microframe or 1 ms frame). The client software and device can depend only on the fact that the host will ensure that the time duration between two transaction attempts with the endpoint will be no longer than the desired period. Note that errors on the bus can prevent an interrupt transaction from being successfully delivered over the bus and consequently exceed the desired period. Also, the endpoint is only polled when the software client has an IRP for an interrupt transfer pending. If the bus time for performing an interrupt transfer arrives and there is no IRP pending, the endpoint will not be given an opportunity to transfer data at that time. Once an IRP is available, its data will be transferred at the next allocated period.

A high-speed endpoint can move up to 3072 bytes per microframe (or 192 Mb/s). A high-speed interrupt endpoint that requires more than 1024 bytes per period is called a high-bandwidth endpoint. A high-bandwidth endpoint uses multiple transactions per microframe. A high-bandwidth endpoint must specify a period of 1x125 µs (i.e., a *bInterval* value of 1). See Section 5.9 for more information about the details of multiple transactions per microframe for high-bandwidth high-speed endpoints.

Interrupt transfers are moved over the USB by accessing an interrupt endpoint every specified period. For input interrupt endpoints, the host has no way to determine whether an endpoint will source an interrupt without accessing the endpoint and requesting an interrupt transfer. If the endpoint has no interrupt data to transmit when accessed by the host, it responds with NAK. An endpoint should only provide interrupt data when it has an interrupt pending to avoid having a software client erroneously notified of IRP complete. A zero-length data payload is a valid transfer and may be useful for some implementations.

5.7.5 Interrupt Transfer Data Sequences

Interrupt transactions may use either alternating data toggle bits, such that the bits are toggled only upon successful transfer completion or a continuously toggling of data toggle bits. The host in any case must assume that the device is obeying full handshake/retry rules as defined in Chapter 8. A device may choose to always toggle DATA0/DATA1 PIDs so that it can ignore handshakes from the host. However, in this case, the client software can miss some data packets when an error occurs, because the Host Controller interprets the next packet as a retry of a missed packet.

If a halt condition is detected on an interrupt pipe due to transmission errors or a STALL handshake being returned from the endpoint, all pending IRPs are retired. Removal of the halt condition is achieved via software intervention through a separate control pipe. This recovery will reset the data toggle bit to DATA0 for the endpoint on both the host and the device. Interrupt transactions are retried due to errors detected on the bus that affect a given transfer.

5.8 Bulk Transfers

The bulk transfer type is designed to support devices that need to communicate relatively large amounts of data at highly variable times where the transfer can use any available bandwidth. Requesting a pipe with a bulk transfer type provides the requester with the following:

- Access to the USB on a bandwidth-available basis
- Retry of transfers, in the case of occasional delivery failure due to errors on the bus
- Guaranteed delivery of data but no guarantee of bandwidth or latency

Bulk transfers occur only on a bandwidth-available basis. For a USB with large amounts of free bandwidth, bulk transfers may happen relatively quickly; for a USB with little bandwidth available, bulk transfers may trickle out over a relatively long period of time.

5.8.1 Bulk Transfer Data Format

The USB imposes no data content structure on communication flows for bulk pipes.

5.8.2 Bulk Transfer Direction

A bulk pipe is a stream pipe and, therefore, always has communication flowing either into or out of the host for a given pipe. If a device requires bi-directional bulk communication flow, two bulk pipes must be used, one in each direction.

5.8.3 Bulk Transfer Packet Size Constraints

An endpoint for bulk transfers specifies the maximum data payload size that the endpoint can accept from or transmit to the bus. The USB defines the allowable maximum bulk data payload sizes to be only 8, 16, 32, or 64 bytes for full-speed endpoints and 512 bytes for high-speed endpoints. A low-speed device must not have bulk endpoints. This maximum applies to the data payloads of the data packets; i.e., the size specified is for the data field of the packet as defined in Chapter 8, not including other protocol-required information.

A bulk endpoint is designed to support a maximum data payload size. A bulk endpoint reports in its configuration information the value for its maximum data payload size. The USB does not require that data payloads transmitted be exactly the maximum size; i.e., if a data payload is less than the maximum, it does not need to be padded to the maximum size.

All Host Controllers are required to have support for 8-, 16-, 32-, and 64-byte maximum packet sizes for full-speed bulk endpoints and 512 bytes for high-speed bulk endpoints. No Host Controller is required to support larger or smaller maximum packet sizes.

During configuration, the USB System Software reads the endpoint's maximum data payload size and ensures that no data payload will be sent to the endpoint that is larger than the supported size.

An endpoint must always transmit data payloads with a data field less than or equal to the endpoint's reported *wMaxPacketSize* value. When a bulk IRP involves more data than can fit in one maximum-sized data payload, all data payloads are required to be maximum size except for the last data payload, which will contain the remaining data. A bulk transfer is complete when the endpoint does one of the following:

- Has transferred exactly the amount of data expected
- Transfers a packet with a payload size less than *wMaxPacketSize* or transfers a zero-length packet

When a bulk transfer is complete, the Host Controller retires the current IRP and advances to the next IRP. If a data payload is received that is larger than expected, all pending bulk IRPs for that endpoint will be aborted/retired.

5.8.4 Bulk Transfer Bus Access Constraints

Only full-speed and high-speed devices can use bulk transfers.

An endpoint has no way to indicate a desired bus access frequency for a bulk pipe. The USB balances the bus access requirements of all bulk pipes and the specific IRPs that are pending to provide "good effort" delivery of data between client software and functions. Moving control transfers over the bus has priority over moving bulk transfers.

There is no time guaranteed to be available for bulk transfers as there is for control transfers. Bulk transfers are moved over the bus only on a bandwidth-available basis. If there is bus time that is not being used for other purposes, bulk transfers will be moved over the bus. If there are bulk transfers pending for multiple endpoints, bulk transfers for the different endpoints are selected according to a fair access policy that is Host Controller implementation-dependent.

All bulk transfers pending in a system contend for the same available bus time. Because of this, the USB System Software at its discretion can vary the bus time made available for bulk transfers to a particular endpoint. An endpoint and its client software cannot assume a specific rate of service for bulk transfers. Bus time made available to a software client and its endpoint can be changed as other devices are inserted into and removed from the system or also as bulk transfers are requested for other device endpoints. Client software cannot assume ordering between bulk and control transfers; i.e., in some situations, bulk transfers can be delivered ahead of control transfers.

High-speed bulk OUT endpoints must support the PING flow control protocol. The details of this protocol are described in Section 8.5.1.

The bus frequency and (micro)frame timing limit the maximum number of successful bulk transactions within a (micro)frame for any USB system to less than 72 full-speed eight-byte data payloads or less than 14 high-speed 512-byte data payloads. Table 5-9 lists information about different-sized full-speed bulk transactions and the maximum number of transactions possible in a frame. The table does not include the overhead associated with bit stuffing. Table 5-10 lists similar information for high-speed bulk transactions.

Table 5-9. Full-speed Bulk Transaction Limits

	Protocol Overhead (13 bytes)		(3 SYNC bytes, 3 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 3-byte interpacket delay)			
	Data Payload	**Max Bandwidth (bytes/second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame Useful Data**
	1	107000	1%	107	2	107
	2	200000	1%	100	0	200
	4	352000	1%	88	4	352
	8	568000	1%	71	9	568
	16	816000	2%	51	21	816
	32	1056000	3%	33	15	1056
	64	1216000	5%	19	37	1216
Max		1500000				1500

Table 5-10. High-speed Bulk Transaction Limits

	Protocol Overhead (55 bytes)	(3x4 SYNC bytes, 3 PID bytes, 2 EP/ADDR+CRC bytes, 2 CRC16, and a 3x(1+11) byte interpacket delay (EOP, etc.))				
	Data Payload	Max Bandwidth (bytes/second)	Microframe Bandwidth per Transfer	Max Transfers	Bytes Remaining	Bytes/ Microframe Useful Data
	1	1064000	1%	133	52	133
	2	2096000	1%	131	33	262
	4	4064000	1%	127	7	508
	8	7616000	1%	119	3	952
	16	13440000	1%	105	45	1680
	32	22016000	1%	86	18	2752
	64	32256000	2%	63	3	4032
	128	40960000	2%	40	180	5120
	256	49152000	4%	24	36	6144
	512	53248000	8%	13	129	6656
Max		60000000				7500

Host Controllers are free to determine how the individual bus transactions for specific bulk transfers are moved over the bus within and across (micro)frames. An endpoint could see all bus transactions for a bulk transfer within the same (micro)frame or spread across several (micro)frames. A Host Controller, for various implementation reasons, may not be able to provide the above maximum number of transactions per (micro)frame.

5.8.5 Bulk Transfer Data Sequences

Bulk transactions use data toggle bits that are toggled only upon successful transaction completion to preserve synchronization between transmitter and receiver when transactions are retried due to errors. Bulk transactions are initialized to DATA0 when the endpoint is configured by an appropriate control transfer. The host will also start the first bulk transaction with DATA0. If a halt condition is detected on a bulk pipe due to transmission errors or a STALL handshake being returned from the endpoint, all pending IRPs are retired. Removal of the halt condition is achieved via software intervention through a separate control pipe. This recovery will reset the data toggle bit to DATA0 for the endpoint on both the host and the device.

Bulk transactions are retried due to errors detected on the bus that affect a given transaction.

5.9 High-Speed, High Bandwidth Endpoints

USB supports individual high-speed interrupt or isochronous endpoints that require data rates up to 192 Mb/s (i.e., 3072 data bytes per microframe). One, two, or three high-speed transactions are allowed in a single microframe to support high-bandwidth endpoints.

A high-speed interrupt or isochronous endpoint indicates that it requires more than 1024 bytes per microframe when bits 12..11 of the *wMaxPacketSize* field of the endpoint descriptor (see Table 5-11) are non-zero. The lower 11 bits of *wMaxPacketSize* indicate the size of the data payload for each individual transaction while bits 12..11 indicate the maximum number of required transactions possible. See Section 9.6.6 for restrictions on the allowed combinations of values for bits 12..11 and bits 10..0.

Table 5-11. *wMaxPacketSize* Field of Endpoint Descriptor

Bits	15..13	12..11	10..0
Field	Reserved, must be set to zero	Number of transactions per microframe	Maximum size of data payload in bytes

Note: This representation means that endpoints requesting two transactions per microframe will specify a total data payload size in the microframe that is a multiple of two bytes. Also endpoints requesting three transactions per microframe will specify a total data payload size that is a multiple of three bytes. In any case, any number of bytes can actually be transferred in a microframe.

The host controller must issue an appropriate number of high-speed transactions per microframe. Errors in the host or on the bus can result in the host controller issuing fewer transactions than requested for the endpoint. The first transaction(s) must have a data payload(s) as specified by the lower 11 bits of *wMaxPacketSize* if enough data is available, while the last transaction has any remaining data less than or equal to the maximum size specified. The host controller may issue transactions for the same endpoint one immediately after the other (as required for the actual data provided) or may issue transactions for other endpoints in between the transactions for a high bandwidth endpoint.

5.9.1 High Bandwidth Interrupt Endpoints

For interrupt transactions, if the endpoint NAKs a transaction during a microframe, the host controller must not issue further transactions for that endpoint until the next period.

If the endpoint times-out a transaction, the host controller must retry the transaction. The endpoint specifies the maximum number of desired transactions per microframe. If the maximum number of transactions per microframe has not been reached, the host controller may immediately retry the transaction during the current microframe. Host controllers are recommended to do an immediate retry since this minimizes impact on devices that are bandwidth sensitive. If the maximum number of transactions per microframe has been reached, the host controller must retry the transaction at the next period for the endpoint.

A host controller is allowed to issue less than the maximum number of transactions to an endpoint per microframe only if more than a single memory buffer is required for the transactions within the microframe.

Normal DATA0/DATA1 data toggle sequencing is used for each interrupt transaction during a microframe.

5.9.2 High Bandwidth Isochronous Endpoints

For isochronous transactions, if an IN endpoint provides less than a maximum data payload as specified by its endpoint descriptor, the host must not issue further transactions for that endpoint for that microframe.

For an isochronous OUT endpoint, the host controller must issue the number of transactions as required for the actual data provided, not exceeding the maximum number specified by the endpoint descriptor. The transactions issued must adhere to the maximum payload sizes as specified in the endpoint descriptor.

No retries are ever done for isochronous endpoints.

High bandwidth isochronous endpoints (IN and OUT) must support data PID sequencing. Data PID sequencing provides the required support for the data receiver to detect one or more lost/damaged packets per microframe.

Data PID sequencing for a high-speed, high bandwidth isochronous IN endpoint uses a repeating sequence of DATA2, DATA1, DATA0 PIDs for the data packet of each transaction in a microframe. If there is only a single transaction in the microframe, only a DATA0 data packet PID is used. If there are two transactions per microframe, DATA1 is used for the first transaction data packet and DATA0 is used for the second transaction data packet. If there are three transactions per microframe, DATA2 is used for the first transaction data packet, DATA1 is used for the second, and DATA0 is used for the third. In all cases, the data PID sequence starts over again the next microframe. Figure 5-11 shows the order of data packet PIDs that are used in subsequent transactions within a microframe for high-bandwidth isochronous IN endpoints.

Figure 5-11. Data Phase PID Sequence for Isochronous IN High Bandwidth Endpoints

An endpoint must respond to an IN token for the first transaction with a DATA2 when it requires three transactions of data to be moved. It must respond with a DATA1 for the first transaction when it requires two transactions and with a DATA0 when it requires only a single transaction. After the first transaction, the endpoint follows the data PID sequence as described above.

The host knows the maximum number of allowed transactions per microframe for the IN endpoint. The host expects the response to the first transaction to encode (via the data packet PID) how many transactions are required by the endpoint for this microframe. If the host doesn't receive an error-free, appropriate response to any transaction, the host must not issue any further transactions to the endpoint for that microframe. When the host receives a DATA0 data packet from the endpoint, it must not issue any further transactions to the endpoint for that microframe.

Data PID sequencing for a high-speed, high bandwidth isochronous OUT endpoint uses a different sequence than that used for an IN endpoint. The host must issue a DATA0 data packet when there is a single transaction. The host must issue an MDATA for the first transaction and a DATA1 for the second transaction when there are two transactions per microframe. The host must issue two MDATA transactions and a DATA2 for the third transaction when there are three transactions per microframe. These sequences allow the endpoint to detect if there was a lost/damaged transaction during a microframe. Figure 5-12 shows the order of data packet PIDs that are used in subsequent transactions within a microframe for high-bandwidth isochronous OUT.

Figure 5-12. Data Phase PID Sequence for Isochronous OUT High Bandwidth Endpoints

If the wrong OUT transactions are detected by the endpoint, all of the data transferred during the microframe must be treated as if it had encountered an error. Note that for the three transactions per microframe case with a missing MDATA transaction, USB provides no way for the endpoint to determine which of the two MDATA transactions was lost. There may be application specific methods to more precisely determine which data was lost, but USB provides no method to do so at the bus level.

5.10 Split Transactions

Host controllers and hubs support one additional transaction type called split transactions. This transaction type allows full- and low-speed devices to be attached to hubs operating at high-speed. These transactions involve only host controllers and hubs and are not visible to devices. High-speed split transactions for interrupt and isochronous transfers must be allocated by the host from the 80% periodic portion of a microframe. More information on split transactions can be found in Chapter 8 and Chapter 11.

5.11 Bus Access for Transfers

Accomplishing any data transfer between the host and a USB device requires some use of the USB bandwidth. Supporting a wide variety of isochronous and asynchronous devices requires that each device's transfer requirements are accommodated. The process of assigning bus bandwidth to devices is called transfer management. There are several entities on the host that coordinate the information flowing over the USB: client software, the USB Driver (USBD), and the Host Controller Driver (HCD). Implementers of these entities need to know the key concepts related to bus access:

- Transfer Management: The entities and the objects that support communication flow over the USB
- Transaction Tracking: The USB mechanisms that are used to track transactions as they move through the USB system
- Bus Time: The time it takes to move a packet of information over the bus
- Device/Software Buffer Size: The space required to support a bus transaction
- Bus Bandwidth Reclamation: Conditions where bandwidth that was allocated to other transfers but was not used and can now be possibly reused by control and bulk transfers

The previous sections focused on how client software relates to a function and what the logical flows are over a pipe between the two entities. This section focuses on the different parts of the host and how they must interact to support moving data over the USB. This information may also be of interest to device implementers so they understand aspects of what the host is doing when a client requests a transfer and how that transfer is presented to the device.

5.11.1 Transfer Management

Transfer management involves several entities that operate on different objects in order to move transactions over the bus:

- Client Software: Consumes/generates function-specific data to/from a function endpoint via calls and callbacks requesting IRPs with the USBD interface.
- USB Driver (USBD): Converts data in client IRPs to/from device endpoint via calls/callbacks with the appropriate HCD. A single client IRP may involve one or more transfers.
- Host Controller Driver (HCD): Converts IRPs to/from transactions (as required by a Host Controller implementation) and organizes them for manipulation by the Host Controller. Interactions between the HCD and its hardware is implementation-dependent and is outside the scope of the USB Specification.
- Host Controller: Takes transactions and generates bus activity via packets to move function-specific data across the bus for each transaction.

Figure 5-13 shows how the entities are organized as information flows between client software and the USB. The objects of primary interest to each entity are shown at the interfaces between entities.

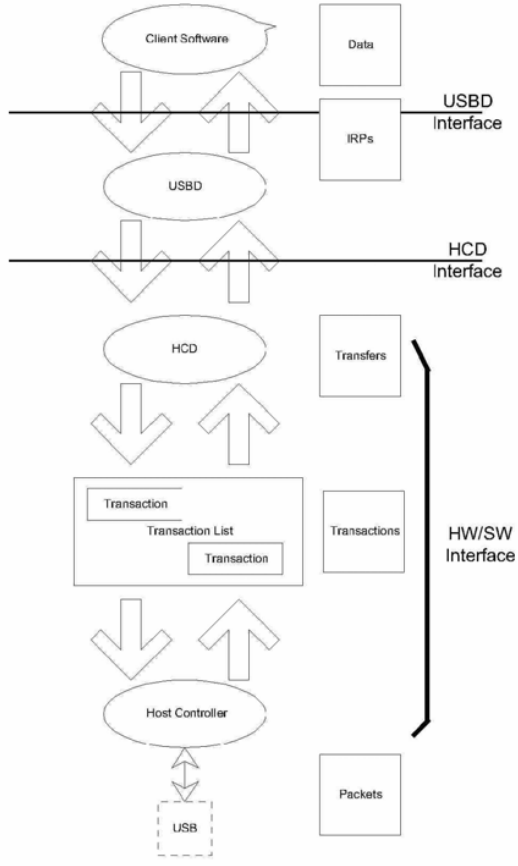

Figure 5-13. USB Information Conversion From Client Software to Bus

5.11.1.1 Client Software

Client software determines what transfers need to be made with a function. It uses appropriate operating system-specific interfaces to request IRPs. Client software is aware only of the set of pipes (i.e., the interface) it needs to manipulate its function. The client is aware of and adheres to all bus access and bandwidth constraints as described previously for each transfer type. The requests made by the client software are presented via the USBD interface.

Some clients may manipulate USB functions via other device class interfaces defined by the operating system and may themselves not make direct USBD calls. However, there is always some lowest level client that makes USBD calls to pass IRPs to the USBD. All IRPs presented are required to adhere to the prenegotiated bandwidth constraints set when the pipe was established. If a function is moved from a non-USB environment to the USB, the driver that would have directly manipulated the function hardware via memory or I/O accesses is the lowest client software in the USB environment that now interacts with the USBD to manipulate the driver's USB function.

After client software has requested a transfer of its function and the request has been serviced, the client software receives notification of the completion status of the IRP. If the transfer involved function-to-host data transfer, the client software can access the data in the data buffer associated with the completed IRP.

The USBD interface is defined in Chapter 10.

5.11.1.2 USB Driver

The Universal Serial Bus Driver (USBD) is involved in mediating bus access at two general times:

- While a device is attached to the bus during configuration
- During normal transfers

When a device is attached and configured, the USBD is involved to ensure that the desired device configuration can be accommodated on the bus. The USBD receives configuration requests from the configuring software that describe the desired device configuration: endpoint(s), transfer type(s), transfer period(s), data size(s), etc. The USBD either accepts or rejects a configuration request based on bandwidth availability and the ability to accommodate that request type on the bus. If it accepts the request, the USBD creates a pipe for the requester of the desired type and with appropriate constraints as defined for the transfer type. Bandwidth allocation for periodic endpoints does not have to be made when the device is configured and, once made, a bandwidth allocation can be released without changing the device configuration.

The configuration aspects of the USBD are typically operating system-specific and heavily leverage the configuration features of the operating system to avoid defining additional (redundant) interfaces.

Once a device is configured, the software client can request IRPs to move data between it and its function endpoints.

5.11.1.3 Host Controller Driver

The Host Controller Driver (HCD) is responsible for tracking the IRPs in progress and ensuring that USB bandwidth and (micro)frame time maximums are never exceeded. When IRPs are made for a pipe, the HCD adds them to the transaction list. When an IRP is complete, the HCD notifies the requesting software client of the completion status for the IRP. If the IRP involved data transfer from the function to the software client, the data was placed in the client-indicated data buffer.

IRPs are defined in an operating system-dependent manner.

5.11.1.4 Transaction List

The transaction list is a Host Controller implementation-dependent description of the current outstanding set of bus transactions that need to be run on the bus. Only the HCD and its Host Controller have access to the specific representation. Each description contains transaction descriptions in which parameters, such as data size in bytes, the device address and endpoint number, and the memory area to which data is to be sent or received, are identified.

A transaction list and the interface between the HCD and its Host Controller is typically represented in an implementation-dependent fashion and is not defined explicitly as part of the USB Specification.

5.11.1.5 Host Controller

The Host Controller has access to the transaction list and translates it into bus activity. In addition, the Host Controller provides a reporting mechanism whereby the status of a transaction (done, pending, halted, etc.) can be obtained. The Host Controller converts transactions into appropriate implementation-dependent activities that result in USB packets moving over the bus topology rooted in the root hub.

The Host Controller ensures that the bus access rules defined by the protocol are obeyed, such as inter-packet timings, timeouts, babble, etc. The HCD interface provides a way for the Host Controller to participate in deciding whether a new pipe is allowed access to the bus. This is done because Host Controller implementations can have restrictions/constraints on the minimum inter-transaction times they may support for combinations of bus transactions.

The interface between the transaction list and the Host Controller is hidden within an HCD and Host Controller implementation.

5.11.2 Transaction Tracking

A USB function sees data flowing across the bus in packets as described in Chapter 8. The Host Controller uses some implementation-dependent representation to track what packets to transfer to/from what endpoints at what time or in what order. Most client software does not want to deal with packetized communication flows because this involves a degree of complexity and interconnect dependency that limits the implementation. The USB System Software (USBD and HCD) provides support for matching data movement requirements of a client to packets on the bus. The Host Controller hardware and software uses IRPs to track information about one or more transactions that combine to deliver a transfer of information between the client software and the function. Figure 5-14 summarizes how transactions are organized into IRPs for the four transfer types. Detailed protocol information for each transfer type can be found in Chapter 8. More information about client software views of IRPs can be found in Chapter 10 and in the operating system specific-information for a particular operating system.

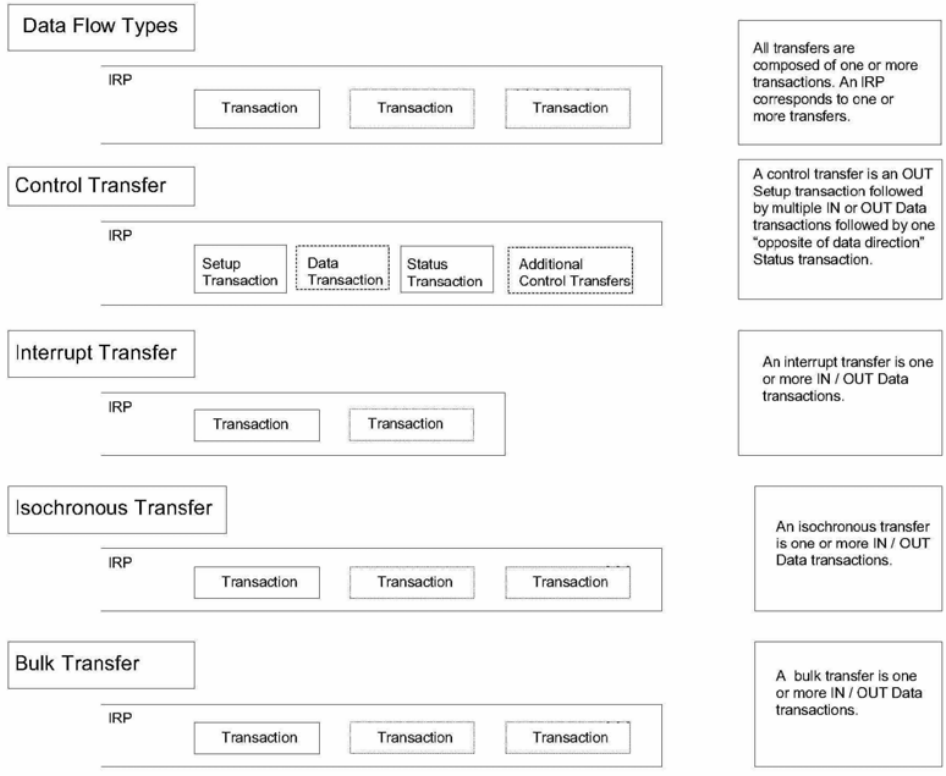

Figure 5-14. Transfers for Communication Flows

Even though IRPs track the bus transactions that need to occur to move a specific data flow over the USB, Host Controllers are free to choose how the particular bus transactions are moved over the bus subject to the USB-defined constraints (e.g., exactly one transaction per (micro)frame for isochronous transfers). In any case, an endpoint will see transactions in the order they appear within an IRP unless errors occur. For example, Figure 5-15 shows two IRPs, one each for two pipes where each IRP contains three transactions. For any transfer type, a Host Controller is free to move the first transaction of the first IRP followed by the first transaction of the second IRP somewhere in (micro)Frame 1, while moving the second transaction of each IRP in opposite order somewhere in (micro)Frame 2. If these are isochronous transfer types, that is the only degree of freedom a Host Controller has. If these are control or bulk transfers, a Host Controller could further move more or less transactions from either IRP within either (micro)frame. Functions cannot depend on seeing transactions within an IRP back-to-back within a (micro)frame nor should they depend on not seeing transactions back-to-back within a (micro)frame.

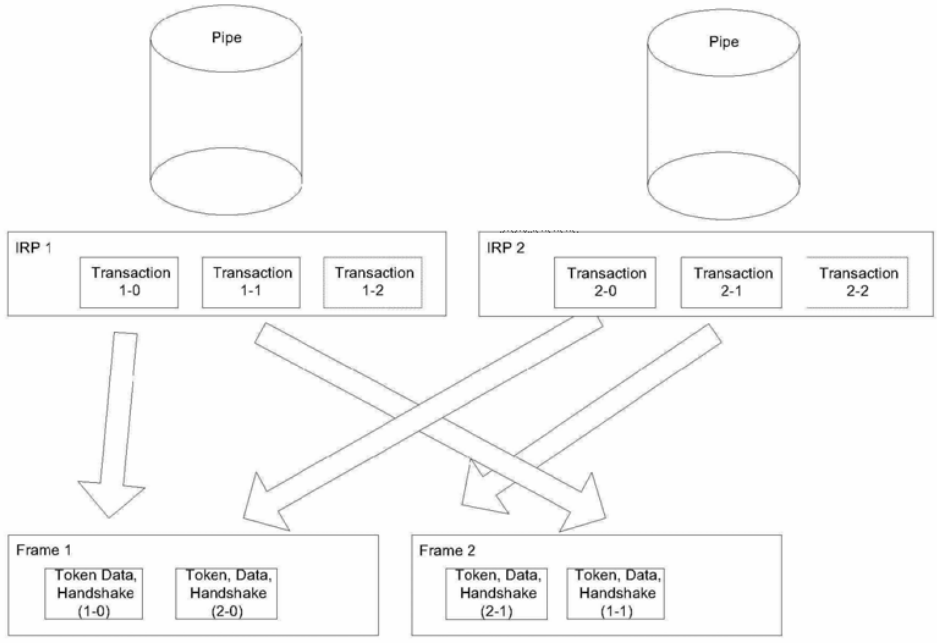

Figure 5-15. Arrangement of IRPs to Transactions/(Micro)frames

5.11.3 Calculating Bus Transaction Times

When the USB System Software allows a new pipe to be created for the bus, it must calculate how much bus time is required for a given transaction. That bus time is based on the maximum packet size information reported for an endpoint, the protocol overhead for the specific transaction type request, the overhead due to signaling imposed bit stuffing, inter-packet timings required by the protocol, inter-transaction timings, etc. These calculations are required to ensure that the time available in a (micro)frame is not exceeded. The equations used to determine transaction bus time are:

KEY:

Data_bc	The byte count of data payload
Host_Delay	The time required for the host or transaction translator to prepare for or recover from the transmission; Host Controller implementation-specific
Floor()	The integer portion of argument
Hub_LS_Setup	The time provided by the Host Controller for hubs to enable low-speed ports; measured as the delay from the end of the PRE PID to the start of the low-speed SYNC; minimum of four full-speed bit times
BitStuffTime	Function that calculates theoretical additional time required due to bit stuffing in signaling; worst case is (1.1667*8*Data_bc)

High-speed (Input)

```
Non-Isochronous Transfer (Handshake Included)
= (55 * 8 * 2.083) + (2.083 * Floor(3.167  + BitStuffTime(Data_bc))) +
Host_Delay

Isochronous Transfer (No Handshake)
= (38 * 8 * 2.083)  + (2.083 * Floor(3.167  + BitStuffTime(Data_bc))) +
Host_Delay
```

High-speed (Output)

```
Non-Isochronous Transfer (Handshake Included)
= (55 * 8 * 2.083)  + (2.083 * Floor(3.167  + BitStuffTime(Data_bc))) +
Host_Delay


Isochronous Transfer (No Handshake)
= (38 * 8 * 2.083)  + (2.083 * Floor(3.167  + BitStuffTime(Data_bc))) +
Host_Delay
```

Full-speed (Input)

```
Non-Isochronous Transfer (Handshake Included)
= 9107 + (83.54 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay

Isochronous Transfer (No Handshake)
= 7268 + (83.54 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay
```

Full-speed (Output)

```
Non-Isochronous Transfer (Handshake Included)
       = 9107 + (83.54 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay


Isochronous Transfer (No Handshake)
       = 6265 + (83.54 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay
```

Low-speed (Input)

```
         = 64060 + (2 * Hub_LS_Setup) +
                (676.67 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay
```

Low-speed (Output)

```
         = 64107 + (2 * Hub_LS_Setup) +
                (667.0 * Floor(3.167 + BitStuffTime(Data_bc))) + Host_Delay
```

The bus times in the above equations are in nanoseconds and take into account propagation delays due to the distance the device is from the host. These are typical equations that can be used to calculate bus time; however, different implementations may choose to use coarser approximations of these times.

The actual bus time taken for a given transaction will almost always be less than that calculated because bit stuffing overhead is data-dependent. Worst case bit stuffing is calculated as 1.1667 (7/6) times the raw time (i.e., the BitStuffTime function multiplies the Data_bc by 8*1.1667 in the equations). This means that there will almost always be time unused on the bus (subject to data pattern specifics) after all regularly scheduled transactions have completed. The bus time made available due to less bit stuffing can be reused as discussed in Section 5.11.5.

The Host_Delay term in the equations is Host Controller-, Transaction Translator(TT)-, and system-dependent and allows for additional time a Host Controller (or TT) may require due to delays in gaining access to memory or other implementation dependencies. This term is incorporated into an implementation of these equations by using the transfer management functions provided by the HCD interface. These equations are typically implemented by a combination of USBD and HCD software working in cooperation.

The results of these calculations are used to determine whether a transfer or pipe creation can be supported in a given USB configuration.

5.11.4 Calculating Buffer Sizes in Functions and Software

Client software and functions both need to provide buffer space for pending data transactions awaiting their turn on the bus. For non-isochronous pipes, this buffer space needs to be just large enough to hold the next data packet. If more than one transaction request is pending for a given endpoint, the buffering for each transaction must be supplied. Methods to calculate the precise absolute minimum buffering a function may require because of specific interactions defined between its client software and the function are outside the scope of this specification.

The Host Controller is expected to be able to support an unlimited number of transactions pending for the bus subject to available system memory for buffer and descriptor space, etc. Host Controllers are allowed to limit how many (micro)frames into the future they allow a transaction to be requested.

For isochronous pipes, Section 5.12.4 describes details affecting host side and device side buffering requirements. In general, buffers need to be provided to hold approximately twice the amount of data that can be transferred in 1ms for full-speed endpoints or 125 µs for high-speed endpoints.

5.11.5 Bus Bandwidth Reclamation

The USB bandwidth and bus access are granted based on a calculation of worst-case bus transmission time and required latencies. However, due to the constraints placed on different transfer types and the fact that the bit stuffing bus time contribution is calculated as a constant but is data-dependent, there will frequently be bus time remaining in each (micro)frame time versus what the (micro)frame transmission time was calculated to be. In order to support the most efficient use of the bus bandwidth, control and bulk transfers are candidates to be moved over the bus as bus time becomes available. Exactly how a Host Controller supports this is implementation-dependent. A Host Controller can take into account the transfer types of pending IRPs and implementation-specific knowledge of remaining (micro)frame time to reuse reclaimed bandwidth.

5.12 Special Considerations for Isochronous Transfers

Support for isochronous data movement between the host and a device is one of the system capabilities supported by the USB. Delivering isochronous data reliably over the USB requires careful attention to detail. The responsibility for reliable delivery is shared by several USB entities:

- The device/function
- The bus
- The Host Controller
- One or more software agents

Because time is a key part of an isochronous transfer, it is important for USB designers to understand how time is dealt with within the USB by these different entities.

Note: The examples in this section describe USB for an example involving full-speed endpoints. The general example details are also appropriate for high-speed endpoints when corresponding changes are made; for example, frame replaced with microframe, 1 ms replaced with 125 µs, rate adjustments made between full-speed and high-speed, etc.

All isochronous devices must report their capabilities in the form of device-specific descriptors. The capabilities should also be provided in a form that the potential customer can use to decide whether the device offers a solution to his problem(s). The specific capabilities of a device can justify price differences.

In any communication system, the transmitter and receiver must be synchronized enough to deliver data robustly. In an asynchronous communication system, data can be delivered robustly by allowing the transmitter to detect that the receiver has not received a data item correctly and simply retrying transmission of the data.

In an isochronous communication system, the transmitter and receiver must remain time- and data-synchronized to deliver data robustly. The USB does not support transmission retry of isochronous data so that minimal bandwidth can be allocated to isochronous transfers and time synchronization is not lost due to a retry delay. However, it is critical that a USB isochronous transmitter/receiver pair still remain synchronized both in normal data transmission cases and in cases where errors occur on the bus.

In many systems that deal with isochronous data, a single global clock is used to which all entities in the system synchronize. An example of such a system is the PSTN (Public Switched Telephone Network). Given that a broad variety of devices with different natural frequencies may be attached to the USB, no single clock can provide all the features required to satisfy the synchronization requirements of all devices and software while still supporting the cost targets of mass-market PC products. The USB defines a clock model that allows a broad range of devices to coexist on the bus and have reasonable cost implementations.

This section presents options or features that can be used by isochronous endpoints to minimize behavior differences between a non-USB implemented function and a USB version of the function. An example is included to illustrate the similarities and differences between the non-USB and USB versions of a function.

The remainder of the section presents the following key concepts:

- USB Clock Model: What clocks are present in a USB system that have impact on isochronous data transfers
- USB (micro)frame Clock-to-function Clock Synchronization Options: How the USB (micro)frame clock can relate to a function clock
- SOF Tracking: Responsibilities and opportunities of isochronous endpoints with respect to the SOF token and USB (micro)frames
- Data Prebuffering: Requirements for accumulating data before generation, transmission, and consumption
- Error Handling: Isochronous-specific details for error handling
- Buffering for Rate Matching: Equations that can be used to calculate buffer space required for isochronous endpoints

5.12.1 Example Non-USB Isochronous Application

The example used is a reasonably generalized example. Other simpler or more complex cases are possible and the relevant USB features identified can be used or not as appropriate.

The example consists of an 8 kHz mono microphone connected through a mixer driver that sends the input data stream to 44 kHz stereo speakers. The mixer expects the data to be received and transmitted at some sample rate and encoding. A rate matcher driver on input and output converts the sample rate and encoding from the natural rate and encoding of the device to the rate and encoding expected by the mixer. Figure 5-16 illustrates this example.

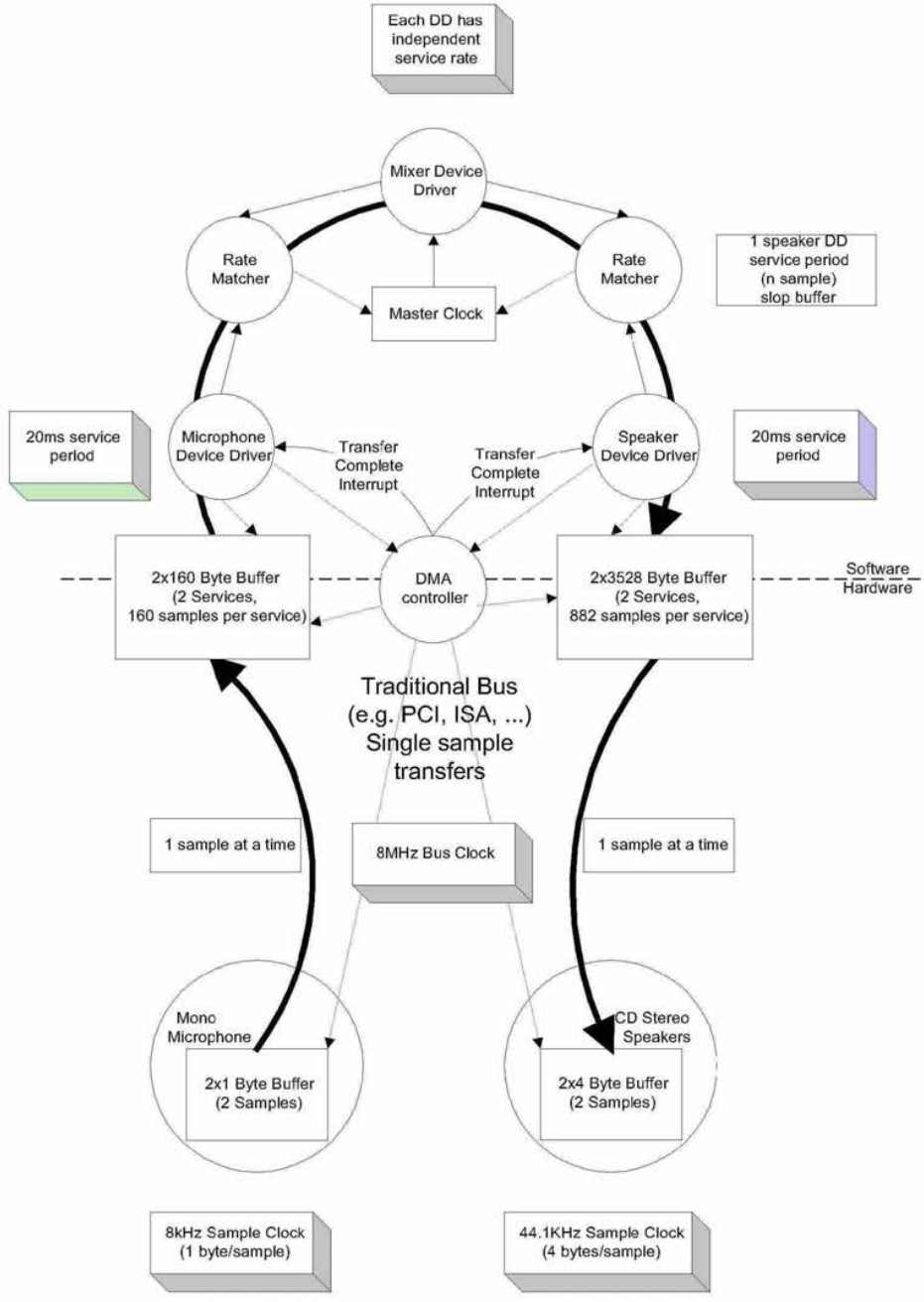

Figure 5-16. Non-USB Isochronous Example

A master clock (which can be provided by software driven from the real time clock) in the PC is used to awaken the mixer to ask the input source for input data and to provide output data to the output sink. In this example, assume it awakens every 20 ms. The microphone and speakers each have their own sample clocks that are unsynchronized with respect to each other or the master mixer clock. The microphone produces data at its natural rate (one-byte samples, 8,000 times a second) and the speakers consume data at their natural rate (four-byte samples, 44,100 times a second). The three clocks in the system can drift and jitter with respect to each other. Each rate matcher may also be running at a different natural rate than either the mixer driver, the input source/driver, or output sink/driver.

The rate matchers also monitor the long-term data rate of their device compared to the master mixer clock and interpolate an additional sample or merge two samples to adjust the data rate of their device to the data rate of the mixer. This adjustment may be required every couple of seconds, but typically occurs infrequently. The rate matchers provide some additional buffering to carry through a rate match.

Note: Some other application might not be able to tolerate sample adjustment and would need some other means of accommodating master clock-to-device clock drift or else would require some means of synchronizing the clocks to ensure that no drift could occur.

The mixer always expects to receive exactly a service period of data (20 ms service period) from its input device and produce exactly a service period of data for its output device. The mixer can be delayed up to less than a service period if data or space is not available from its input/output device. The mixer assumes that such delays do not accumulate.

The input and output devices and their drivers expect to be able to put/get data in response to a hardware interrupt from the DMA controller when their transducer has processed one service period of data. They expect to get/put exactly one service period of data. The input device produces 160 bytes (ten samples) every service period of 20 ms. The output device consumes 3,528 bytes (882 samples) every 20 ms service period. The DMA controller can move a single sample between the device and the host buffer at a rate much faster than the sample rate of either device.

The input and output device drivers provide two service periods of system buffering. One buffer is always being processed by the DMA controller. The other buffer is guaranteed to be ready before the current buffer is exhausted. When the current buffer is emptied, the hardware interrupt awakens the device driver and it calls the rate matcher to give it the buffer. The device driver requests a new IRP with the buffer before the current buffer is exhausted.

The devices can provide two samples of data buffering to ensure that they always have a sample to process for the next sample period while the system is reacting to the previous/next sample.

The service periods of the drivers are chosen to survive interrupt latency variabilities that may be present in the operating system environment. Different operating system environments will require different service periods for reliable operation. The service periods are also selected to place a minimum interrupt load on the system, because there may be other software in the system that requires processing time.

5.12.2 USB Clock Model

Time is present in the USB system via clocks. In fact, there are multiple clocks in a USB system that must be understood:

- Sample Clock: This clock determines the natural data rate of samples moving between client software on the host and the function. This clock does not need to be different between non-USB and USB implementations.
- Bus Clock: This clock runs at a 1.000 ms period (1 kHz frequency) on full-speed segments and 125.000 µs (8 kHz frequency) on high-speed segments of the bus and is indicated by the rate of SOF packets on the bus. This clock is somewhat equivalent to the 8 MHz clock in the non-USB example. In the USB case, the bus clock is often a lower-frequency clock than the sample clock, whereas the bus clock is almost always a higher-frequency clock than the sample clock in a non-USB case.
- Service Clock: This clock is determined by the rate at which client software runs to service IRPs that may have accumulated between executions. This clock also can be the same in the USB and non-USB cases.

In most existing operating systems, it is not possible to support a broad range of isochronous communication flows if each device driver must be interrupted for each sample for fast sample rates. Therefore, multiple samples, if not multiple packets, will be processed by client software and then given to the Host Controller to sequence over the bus according to the prenegotiated bus access requirements. Figure 5-17 presents an example for a reasonable USB clock environment equivalent to the non-USB example in Figure 5-16.

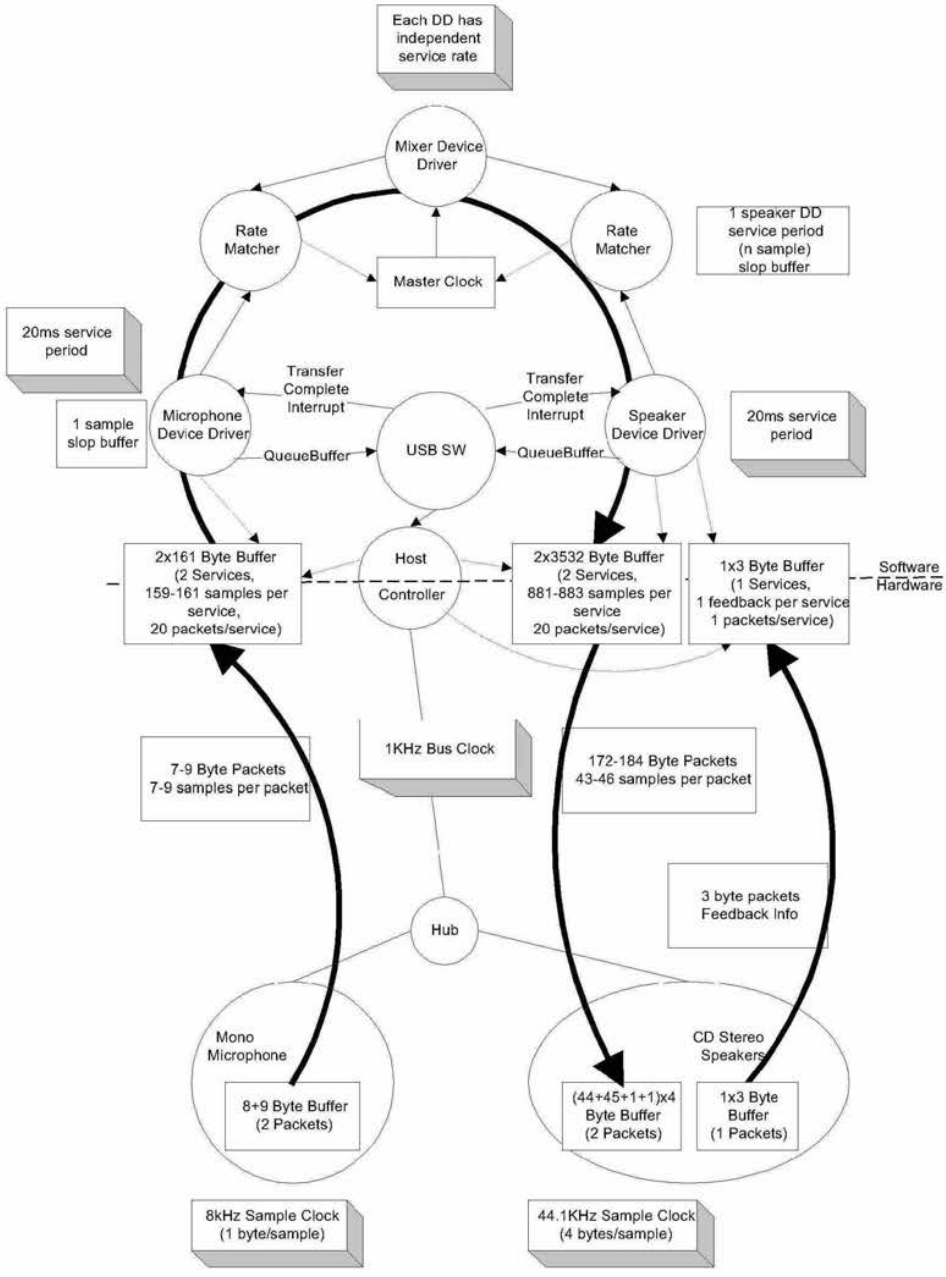

Figure 5-17. USB Full-speed Isochronous Application

Figure 5-17 shows a typical round trip path of information from a microphone as an input device to a speaker as an output device. The clocks, packets, and buffering involved are also shown. Figure 5-17 will be explored in more detail in the following sections.

The focus of this example is to identify the differences introduced by the USB compared to the previous non-USB example. The differences are in the areas of buffering, synchronization given the existence of a USB bus clock, and delay. The client software above the device drivers can be unaffected in most cases.

5.12.3 Clock Synchronization

In order for isochronous data to be manipulated reliably, the three clocks identified above must be synchronized in some fashion. If the clocks are not synchronized, several clock-to-clock attributes can be present that can be undesirable:

- Clock Drift: Two clocks that are nominally running at the same rate can, in fact, have implementation differences that result in one clock running faster or slower than the other over long periods of time. If uncorrected, this variation of one clock compared to the other can lead to having too much or too little data when data is expected to always be present at the time required.
- Clock Jitter: A clock may vary its frequency over time due to changes in temperature, etc. This may also alter when data is actually delivered compared to when it is expected to be delivered.
- Clock-to-clock Phase Differences: If two clocks are not phase locked, different amounts of data may be available at different points in time as the beat frequency of the clocks cycle out over time. This can lead to quantization/sampling related artifacts.

The bus clock provides a central clock with which USB hardware devices and software can synchronize to one degree or another. However, the software will, in general, not be able to phase- or frequency-lock precisely to the bus clock given the current support for "real time-like" operating system scheduling support in most PC operating systems. Software running in the host can, however, know that data moved over the USB is packetized. For isochronous transfer types, a unit of data is moved exactly once per (micro)frame and the (micro)frame clock is reasonably precise. Providing the software with this information allows it to adjust the amount of data it processes to the actual (micro)frame time that has passed.

Note: For high-speed high-bandwidth endpoints, the data exchanged in the two or three transactions per microframe is still considered to belong to the same "single packet." The large amount of data per packet is split into two or three transactions only for bus efficiency reasons.

5.12.4 Isochronous Devices

The USB includes a framework for isochronous devices that defines synchronization types, how isochronous endpoints provide data rate feedback, and how they can be connected together. Isochronous devices include sampled analog devices (for example, audio and telephony devices) and synchronous data devices. Synchronization type classifies an endpoint according to its capability to synchronize its data rate to the data rate of the endpoint to which it is connected. Feedback is provided by indicating accurately what the required data rate is, relative to the SOF frequency. The ability to make connections depends on the quality of connection that is required, the endpoint synchronization type, and the capabilities of the host application that is making the connection. Additional device class-specific information may be required, depending on the application.

Note: The term "data" is used very generally, and may refer to data that represents sampled analog information (like audio), or it may be more abstract information. "Data rate" refers to the rate at which analog information is sampled, or the rate at which data is clocked.

The following information is required in order to determine how to connect isochronous endpoints:

- Synchronization type:
  - Asynchronous: Unsynchronized, although sinks provide data rate feedback
  - Synchronous: Synchronized to the USB's SOF
  - Adaptive: Synchronized using feedback or feedforward data rate information
- Available data rates
- Available data formats

Synchronization type and data rate information are needed to determine if an exact data rate match exists between source and sink, or if an acceptable conversion process exists that would allow the source to be connected to the sink. It is the responsibility of the application to determine whether the connection can be supported within available processing resources and other constraints (like delay). Specific USB device classes define how to describe synchronization type and data rate information.

Data format matching and conversion is also required for a connection, but it is not a unique requirement for isochronous connections. Details about format conversion can be found in other documents related to specific formats.

5.12.4.1 Synchronization Type

Three distinct synchronization types are defined. Table 5-12 presents an overview of endpoint synchronization characteristics for both source and sink endpoints. The types are presented in order of increasing capability.

Table 5-12. Synchronization Characteristics

	Source	Sink
Asynchronous	Free running Fs Provides implicit feedforward (data stream)	Free running Fs Provides explicit feedback (isochronous pipe)
Synchronous	Fs locked to SOF Uses implicit feedback (SOF)	Fs locked to SOF Uses implicit feedback (SOF)
Adaptive	Fs locked to sink Uses explicit feedback (isochronous pipe)	Fs locked to data flow Uses implicit feedforward (data stream)

5.12.4.1.1 Asynchronous

Asynchronous endpoints cannot synchronize to SOF or any other clock in the USB domain. They source or sink an isochronous data stream at either a fixed data rate (single-frequency endpoints), a limited number of data rates (32 kHz, 44.1 kHz, 48 kHz, …), or a continuously programmable data rate. If the data rate is programmable, it is set during initialization of the isochronous endpoint. Asynchronous devices must report their programming capabilities in the class-specific endpoint descriptor as described in their device class specification. The data rate is locked to a clock external to the USB or to a free-running internal clock. These devices place the burden of data rate matching elsewhere in the USB environment. Asynchronous source endpoints carry their data rate information implicitly in the number of samples they produce per

(micro)frame. Asynchronous sink endpoints must provide explicit feedback information to an adaptive driver (refer to Section 5.12.4.2).

An example of an asynchronous source is a CD-audio player that provides its data based on an internal clock or resonator. Another example is a Digital Audio Broadcast (DAB) receiver or a Digital Satellite Receiver (DSR). Here, too, the sample rate is fixed at the broadcasting side and is beyond USB control.

Asynchronous sink endpoints could be low-cost speakers running off of their internal sample clock.

5.12.4.1.2 Synchronous

Synchronous endpoints can have their clock system (their notion of time) controlled externally through SOF synchronization. These endpoints must slave their sample clock to the 1 ms SOF tick (by means of a programmable PLL). For high-speed endpoints, the presence of the microframe SOF can be used for tighter frame clock tracking.

Synchronous endpoints may source or sink isochronous data streams at either a fixed data rate (single-frequency endpoints), a limited number of data rates (32 kHz, 44.1 kHz, 48 kHz, …), or a continuously programmable data rate. If programmable, the operating data rate is set during initialization of the isochronous endpoint. The number of samples or data units generated in a series of USB (micro)frames is deterministic and periodic. Synchronous devices must report their programming capabilities in the class-specific endpoint descriptor as described in their device class specification.

An example of a synchronous source is a digital microphone that synthesizes its sample clock from SOF and produces a fixed number of audio samples every USB (micro)frame. Likewise, a synchronous sink derives its sample clock from SOF and consumes a fixed number of samples every USB (micro)frame.

5.12.4.1.3 Adaptive

Adaptive endpoints are the most capable endpoints possible. They are able to source or sink data at any rate within their operating range. Adaptive source endpoints produce data at a rate that is controlled by the data sink. The sink provides feedback (refer to Section 5.12.4.2) to the source, which allows the source to know the desired data rate of the sink. For adaptive sink endpoints, the data rate information is embedded in the data stream. The average number of samples received during a certain averaging time determines the instantaneous data rate. If this number changes during operation, the data rate is adjusted accordingly.

The data rate operating range may center around one rate (e.g., 8 kHz), select between several programmable or auto-detecting data rates (32 kHz, 44.1 kHz, 48 kHz, …), or may be within one or more ranges (e.g., 5 kHz to 12 kHz or 44 kHz to 49 kHz). Adaptive devices must report their programming capabilities in the class-specific endpoint descriptor as described in their device class specification.

An example of an adaptive source is a CD player that contains a fully adaptive sample rate converter (SRC) so that the output sample frequency no longer needs to be 44.1 kHz but can be anything within the operating range of the SRC. Adaptive sinks include such endpoints as high-end digital speakers, headsets, etc.

5.12.4.2 Feedback

An asynchronous sink must provide explicit feedback to the host by indicating accurately what its desired data rate (F_f) is, relative to the USB (micro)frame frequency. This allows the host to continuously adjust the number of samples sent to the sink so that neither underflow or overflow of the data buffer occurs. Likewise, an adaptive source must receive explicit feedback from the host so that it can accurately generate the number of samples required by the host. Feedback endpoints can be specified as described in Section 9.6.6 for the *bmAttributes* field of the endpoint descriptor.

To generate the desired data rate F_f, the device must measure its actual sampling rate F_s, referenced to the USB notion of time, i.e., the USB (micro)frame frequency. This specification requires the data rate F_f to be

resolved to better than one sample per second (1Hz) in order to allow a high-quality source rate to be created and to tolerate delays and errors in the feedback loop. To achieve this accuracy, the measurement time T_{meas} must be at least 1 second. Therefore:

$$T_{meas} = 2^K$$

where T_{meas} is now expressed in USB (micro)frames and K=10 for full-speed devices (1 ms frames) and K=13 for high-speed devices (125 µs microframes). However, in most devices, the actual sampling rate F_s is derived from a master clock F_m through a binary divider. Therefore:

$$F_m = F_s * 2^P$$

where P is a positive integer (including 0 if no higher-frequency master clock is available). The measurement time T_{meas} can now be decreased by measuring F_m instead of F_s and:

$$T_{meas} = \frac{2^K}{2^P} = 2^{(K-P)}$$

In this way, a new estimate for F_f becomes available every $2^{(K-P)}$ (micro)frames. P is practically bound to be in the range [0,K] because there is no point in using a clock slower than F_s (P=0), and no point in trying to update F_f more than once per (micro)frame (P=K). A sink can determine F_f by counting cycles of the master clock F_m for a period of $2^{(K-P)}$ (micro)frames. The counter is read into F_f and reset every $2^{(K-P)}$ (micro)frames. As long as no clock cycles are skipped, the count will be accurate over the long term.

Each (micro)frame, an adaptive source adds F_f to any remaining fractional sample count from the previous (micro)frame, sources the number of samples in the integer part of the sum, and retains the fractional sample count for the next (micro)frame. The source can look at the behavior of F_f over many (micro)frames to determine an even more accurate rate, if it needs to.

F_f is expressed in number of samples per (micro)frame. The F_f value consists of an integer part that represents the (integer) number of samples per (micro)frame and a fractional part that represents the "fraction" of a sample that would be needed to match the sampling frequency F_s to a resolution of 1 Hz or better. The fractional part requires at least K bits to represent the "fraction" of a sample to a resolution of 1 Hz or better. The integer part must have enough bits to represent the maximum number of samples that can ever occur in a single (micro)frame. Assuming that the minimum sample size is one byte, then this number is limited to 1,023 for full-speed endpoints. Ten bits are therefore sufficient to encode this value. For high-speed endpoints, this number is limited to 3*1,024=3,072 and twelve bits are needed.

In summary, for full-speed endpoints, the F_f value shall be encoded in an unsigned 10.10 (K=10) format which fits into three bytes. Because the maximum integer value is fixed to 1,023, the 10.10 number will be left-justified in the 24 bits, so that it has a 10.14 format. Only the first ten bits behind the binary point are required. The lower four bits may be optionally used to extend the precision of F_f, otherwise, they shall be reported as zero. For high-speed endpoints, the F_f value shall be encoded in an unsigned 12.13 (K=13) format which fits into four bytes. The value shall be aligned into these four bytes so that the binary point is located between the second and the third byte so that it has a 16.16 format. The most significant four bits shall be reported zero. Only the first 13 bits behind the binary point are required. The lower three bits may be optionally used to extend the precision of F_f, otherwise, they shall be reported as zero.

An endpoint needs to implement only the number of bits that it effectively requires for its maximum F_f

The choice of P is endpoint-specific. Use the following guidelines when choosing P:

- P must be in the range [0,K].
- Larger values of P are preferred, because they reduce the size of the frame counter and increase the rate at which F_f is updated. More frequent updates result in a tighter control of the source data rate, which reduces the buffer space required to handle F_f changes.
- P should be less than K so that F_f is averaged across at least two frames in order to reduce SOF jitter effects.
- P should not be zero in order to keep the deviation in the number of samples sourced to less than 1 in the event of a lost F_f value.

Isochronous transfers are used to read F_f from the feedback register. The desired reporting rate for the feedback should be $2^{(K-P)}$ frames. F_f will be reported at most once per update period. There is nothing to be gained by reporting the same F_f value more than once per update period. The endpoint may choose to report F_f only if the updated value has changed from the previous F_f value. If the value has not changed, the endpoint may report the current F_f value or a zero length data payload. It is strongly recommended that an endpoint always report the current F_f value any time it is polled.

It is possible that the source will deliver one too many or one too few samples over a long period due to errors or accumulated inaccuracies in measuring F_f. The sink must have sufficient buffer capability to accommodate this. When the sink recognizes this condition, it should adjust the reported F_f value to correct it. This may also be necessary to compensate for relative clock drifts. The implementation of this correction process is endpoint-specific and is not specified.

5.12.4.3 Implicit Feedback

In some cases, implementing a separate explicit feedback endpoint can be avoided. If a device implements a group of isochronous data endpoints that are closely related and if:

- All the endpoints in the group are synchronized (i.e. use sample clocks that are derived from a common master clock)
- The group contains one or more isochronous data endpoints in one direction that normally would need explicit feedback
- The group contains at least one isochronous data endpoint in the opposite direction

Under these circumstances, the device may elect not to implement a separate isochronous explicit feedback endpoint. Instead, feedback information can be derived from the data endpoint in the opposite direction by observing its data rate.

Two cases can arise:

- One or more asynchronous sink endpoints are accompanied by an asynchronous source endpoint. The data rate on the source endpoint can be used as implicit feedback information to adjust the data rate on the sink endpoint(s).
- One or more adaptive source endpoints are accompanied by an adaptive sink endpoint. The source endpoint can adjust its data rate based on the data rate received by the sink endpoint.

This specification provides the necessary framework to implement synchronization as described above (see Chapter 9). However, exactly how the desired data rate F_f is derived from the data rate of the implied feedback endpoint is implementation-dependent.

5.12.4.4 Connectivity

In order to fully describe the source-to-sink connectivity process, an interconnect model is presented. The model indicates the different components involved and how they interact to establish the connection.

The model provides for multi-source/multi-sink situations. Figure 5-18 illustrates a typical situation (highly condensed and incomplete). A physical device is connected to the host application software through different hardware and software layers as described in this specification. At the client interface level, a virtual device is presented to the application. From the application standpoint, only virtual devices exist. It is up to the device driver and client software to decide what the exact relation is between physical and virtual device.

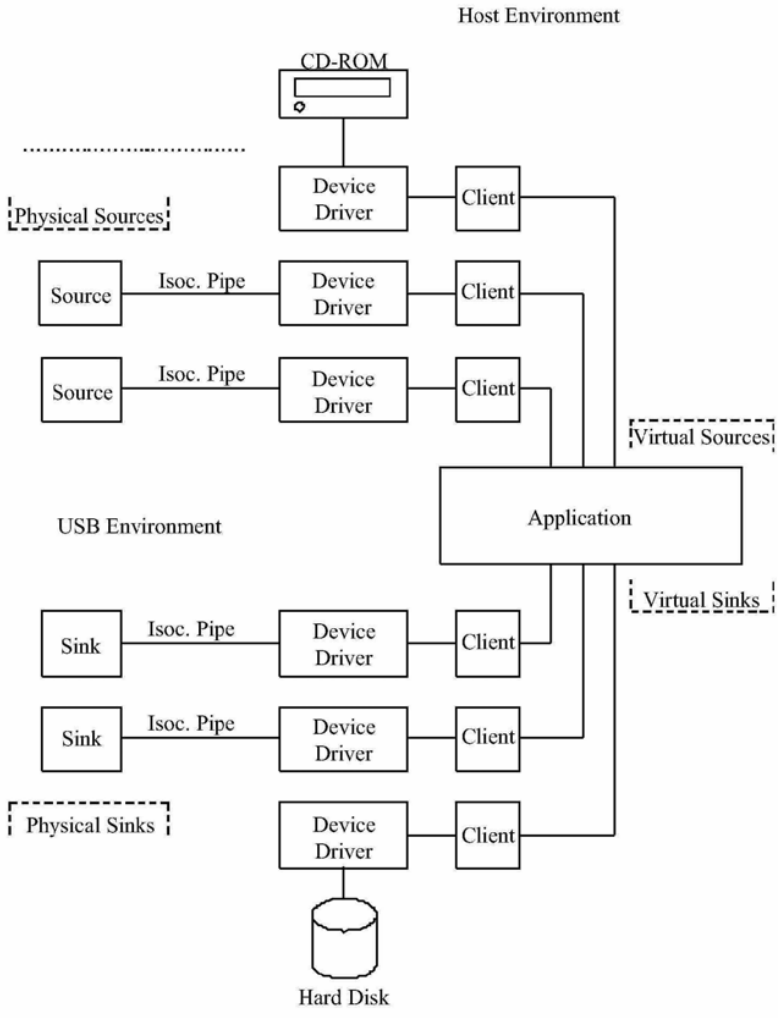

Figure 5-18. Example Source/Sink Connectivity

Device manufacturers (or operating system vendors) must provide the necessary device driver software and client interface software to convert their device from the physical implementation to a USB-compliant software implementation (the virtual device). As stated before, depending on the capabilities built into this software, the virtual device can exhibit different synchronization behavior from the physical device. However, the synchronization classification applies equally to both physical and virtual devices. All physical devices belong to one of the three possible synchronization types. Therefore, the capabilities that have to be built into the device driver and/or client software are the same as the capabilities of a physical device. The word "application" must be replaced by "device driver/client software." In the case of a physical source to virtual source connection, "virtual source device" must be replaced by "physical source device" and "virtual sink device" must be replaced by "virtual source device." In the case of a virtual sink to physical sink connection, "virtual source device" must be replaced by "virtual sink device" and "virtual sink device" must be replaced by "physical sink device."

Placing the rate adaptation (RA) functionality into the device driver/client software layer has the distinct advantage of isolating all applications, relieving the device from the specifics and problems associated with rate adaptation. Applications that would otherwise be multi-rate degenerate to simpler mono-rate systems.

Note: The model is not limited to only USB devices. For example, a CD-ROM drive containing 44.1 kHz audio can appear as either an asynchronous, synchronous, or adaptive source. Asynchronous operation means that the CD-ROM fills its buffer at the rate that it reads data from the disk, and the driver empties the buffer according to its USB service interval. Synchronous operation means that the driver uses the USB service interval (e.g., 10 ms) and nominal sample rate of the data (44.1 kHz) to determine to put out 441 samples every USB service interval. Adaptive operation would build in a sample rate converter to match the CD-ROM output rate to different sink sampling rates.

Using this reference model, it is possible to define what operations are necessary to establish connections between various sources and sinks. Furthermore, the model indicates at what level these operations must or can take place. First, there is the stage where physical devices are mapped onto virtual devices and vice versa. This is accomplished by the driver and/or client software. Depending on the capabilities included in this software, a physical device can be transformed into a virtual device of an entirely different synchronization type. The second stage is the application that uses the virtual devices. Placing rate matching capabilities at the driver/client level of the software stack relieves applications communicating with virtual devices from the burden of performing rate matching for every device that is attached to them. Once the virtual device characteristics are decided, the actual device characteristics are not any more interesting than the actual physical device characteristics of another driver.

As an example, consider a mixer application that connects at the source side to different sources, each running at their own frequencies and clocks. Before mixing can take place, all streams must be converted to a common frequency and locked to a common clock reference. This action can be performed in the physical-to-virtual mapping layer or it can be handled by the application itself for each source device independently. Similar actions must be performed at the sink side. If the application sends the mixed data stream out to different sink devices, it can either do the rate matching for each device itself or it can rely on the driver/client software to do that, if possible.

Table 5-13 indicates at the intersections what actions the application must perform to connect a source endpoint to a sink endpoint.

Table 5-13. Connection Requirements

	Source Endpoint		
Sink Endpoint	**Asynchronous**	**Synchronous**	**Adaptive**
Asynchronous	Async Source/Sink RA See Note 1.	Async SOF/Sink RA See Note 2.	Data + Feedback Feedthrough See Note 3.
Synchronous	Async Source/SOF RA See Note 4.	Sync RA See Note 5.	Data Feedthrough + Application Feedback See Note 6.
Adaptive	Data Feedthrough See Note 7.	Data Feedthrough See Note 8.	Data Feedthrough See Note 9.

Notes:

1. Asynchronous RA in the application. Fs_i is determined by the source, using the feedforward information embedded in the data stream. Fs_O is determined by the sink, based on feedback information from the sink. If nominally $Fs_i = Fs_O$, the process degenerates to a feedthrough connection if slips/stuffs due to lack of synchronization are tolerable. Such slips/stuffs will cause audible degradation in audio applications.
2. Asynchronous RA in the application. Fs_i is determined by the source but locked to SOF. Fs_O is determined by the sink, based on feedback information from the sink. If nominally $Fs_i = Fs_O$, the process degenerates to a feedthrough connection if slips/stuffs due to lack of synchronization are tolerable. Such slips/stuffs will cause audible degradation in audio applications.
3. If Fs_O falls within the locking range of the adaptive source, a feedthrough connection can be established. $Fs_i = Fs_O$ and both are determined by the asynchronous sink, based on feedback information from the sink. If Fs_O falls outside the locking range of the adaptive source, the adaptive source is switched to synchronous mode and Note 2 applies.
4. Asynchronous RA in the application. Fs_i is determined by the source. Fs_O is determined by the sink and locked to SOF. If nominally $Fs_i = Fs_O$, the process degenerates to a feedthrough connection if slips/stuffs due to lack of synchronization are tolerable. Such slips/stuffs will cause audible degradation in audio applications.
5. Synchronous RA in the application. Fs_i is determined by the source and locked to SOF. Fs_O is determined by the sink and locked to SOF. If $Fs_i = Fs_O$, the process degenerates to a loss-free feedthrough connection.
6. The application will provide feedback to synchronize the source to SOF. The adaptive source appears to be a synchronous endpoint and Note 5 applies.
7. If Fs_i falls within the locking range of the adaptive sink, a feedthrough connection can be established. $Fs_i = Fs_O$ and both are determined by and locked to the source.
   If Fs_i falls outside the locking range of the adaptive sink, synchronous RA is done in the host to provide an Fs_O that is within the locking range of the adaptive sink.
8. If Fs_i falls within the locking range of the adaptive sink, a feedthrough connection can be established. $Fs_O = Fs_i$ and both are determined by the source and locked to SOF.
   If Fs_i falls outside the locking range of the adaptive sink, synchronous RA is done in the host to provide an Fs_O that is within the locking range of the adaptive sink.
9. The application will use feedback control to set Fs_O of the adaptive source when the connection is set up. The adaptive source operates as an asynchronous source in the absence of ongoing feedback information and Note 7 applies.

In cases where RA is needed but not available, the rate adaptation process could be mimicked by sample dropping/stuffing. The connection could then still be made, possibly with a warning about poor quality, otherwise, the connection cannot be made.

5.12.4.4.1 Audio Connectivity

When the above is applied to audio data streams, the RA process is replaced by sample rate conversion, which is a specialized form of rate adaptation. Instead of error control, some form of sample interpolation is used to match incoming and outgoing sample rates. Depending on the interpolation techniques used, the audio quality (distortion, signal to noise ratio, etc.) of the conversion can vary significantly. In general, higher quality requires more processing power.

5.12.4.4.2 Synchronous Data Connectivity

For the synchronous data case, RA is used. Occasional slips/stuffs may be acceptable to many applications that implement some form of error control. Error control includes error detection and discard, error detection and retransmit, or forward error correction. The rate of slips/stuffs will depend on the clock mismatch between the source and sink and may be the dominant error source of the channel. If the error control is sufficient, then the connection can still be made.

5.12.5 Data Prebuffering

The USB requires that devices prebuffer data before processing/transmission to allow the host more flexibility in managing when each pipe's transaction is moved over the bus from (micro)frame to (micro)frame.

For transfers from function to host, the endpoint must accumulate samples during (micro)frame X until it receives the SOF token for (micro)frame X+1. It "latches" the data from (micro)frame X into its packet buffer and is now ready to send the packet containing those samples during (micro)frame X+1. When it will send that data during the (micro)frame is determined solely by the Host Controller and can vary from (micro)frame to (micro)frame.

For transfers from host to function, the endpoint will accept a packet from the host sometime during (micro)frame Y. When it receives the SOF for (micro)frame Y+1, it can then start processing the data received in (micro)frame Y.

This approach allows an endpoint to use the SOF token as a stable clock with very little jitter and/or drift when the Host Controller moves the packet over the bus. This approach also allows the Host Controller to vary within a (micro)frame precisely when the packet is actually moved over the bus. This prebuffering introduces some additional delay between when a sample is available at an endpoint and when it moves over the bus compared to an environment where the bus access is at exactly the same time offset from SOF from (micro)frame to (micro)frame.

Figure 5-19 shows the time sequence for a function-to-host transfer (IN process). Data D_0 is accumulated during (micro)frame F*i* at time T*i* and transmitted to the host during (micro)frame F*i*+1. Similarly, for a host-to-function transfer (OUT process), data D_0 is received by the endpoint during (micro)frame F*i*+1 and processed during (micro)frame F*i*+2.

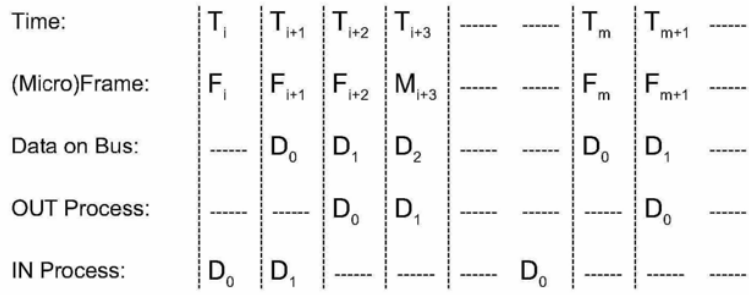

Figure 5-19. Data Prebuffering

5.12.6 SOF Tracking

Functions supporting isochronous pipes must receive and comprehend the SOF token to support prebuffering as previously described. Given that SOFs can be corrupted, a device must be prepared to recover from a corrupted SOF. These requirements limit isochronous transfers to full-speed and high-speed devices only, because low-speed devices do not see SOFs on the bus. Also, because SOF packets can be damaged in transmission, devices that support isochronous transfers need to be able to synthesize the existence of an SOF that they may not see due to a bus error.

Isochronous transfers require the appropriate data to be transmitted in the corresponding (micro)frame. The USB requires that when an isochronous transfer is presented to the Host Controller, it identifies the (micro)frame number for the first (micro)frame. The Host Controller must not transmit the first transaction before the indicated (micro)frame number. Each subsequent transaction in the IRP must be transmitted in succeeding (micro)frames (except for high-speed high-bandwidth transfers where up to three transactions may occur in the same microframe). If there are no transactions pending for the current (micro)frame, then the Host Controller must not transmit anything for an isochronous pipe. If the indicated (micro)frame number has passed, the Host Controller must skip (i.e., not transmit) all transactions until the one corresponding to the current (micro)frame is reached.

5.12.7 Error Handling

Isochronous transfers provide no data packet retries (i.e., no handshakes are returned to a transmitter by a receiver) so that timeliness of data delivery is not perturbed. However, it is still important for the agents responsible for data transport to know when an error occurs and how the error affects the communication flow. In particular, for a sequence of data packets (A, B, C, D), the USB allows sufficient information such that a missing packet (A, _, C, D) can be detected and will not unknowingly be turned into an incorrect data or time sequence (A, C, D or A, _, B, C, D). The protocol provides four mechanisms that support this: a strictly defined periodicity for the transmission of packets and data PID sequencing mechanisms for high-speed high-bandwidth endpoints, SOF, CRC, and bus transaction timeout.

- Isochronous transfers require periodic occurrence of data transactions for normal operation. The period must be an exact power of two (micro)frames. The USB does not dictate what data is transmitted in each frame. The data transmitter/source determines specifically what data to provide. This regular periodic data delivery provides a framework that is fundamental to detecting missing data errors. For high-speed high-bandwidth endpoints, data PID sequencing allows the detection of missing or damaged

transactions during a microframe. Any phase of a transaction can be damaged during transmission on the bus. Chapter 8 describes how each error case affects the protocol.

- Because every (micro)frame is preceded by an SOF and a receiver can see SOFs on the bus, a receiver can determine that its expected transaction for that (micro)frame did not occur between two SOFs. Additionally, because even an SOF can be damaged, a device must be able to reconstruct the existence of a missed SOF as described in Section 5.12.6.
- A data packet may be corrupted on the bus; therefore, CRC protection allows a receiver to determine that the data packet it received was corrupted.
- The protocol defines the details that allow a receiver to determine via bus transaction timeout that it is not going to receive its data packet after it has successfully seen its token packet.

Once a receiver has determined that a data packet was not received, it may need to know the size of the data that was missed in order to recover from the error with regard to its functional behavior. If the communication flow is always the same data size per (micro)frame, then the size is always a known constant. However, in some cases, the data size can vary from (micro)frame to (micro)frame. In this case, the receiver and transmitter have an implementation-dependent mechanism to determine the size of the lost packet.

In summary, whether a transaction is actually moved successfully over the bus or not, the transmitter and receiver always advance their data/buffer streams as indicated by the bus access period to keep data-per-time synchronization. The detailed mechanisms described above allow detection, tracking, and reporting of damaged transactions so that a function or its client software can react to the damage in a function-appropriate fashion. The details of that function- or application-specific reaction are outside the scope of the USB Specification.

5.12.8 Buffering for Rate Matching

Given that there are multiple clocks that affect isochronous communication flows in the USB, buffering is required to rate match the communication flow across the USB. There must be buffer space available both in the device per endpoint and on the host side on behalf of the client software. These buffers provide space for data to accumulate until it is time for a transfer to move over the USB. Given the natural data rates of the device, the maximum size of the data packets that move over the bus can also be calculated.

Figure 5-20 shows the equations used to determine buffer size on the device and host and maximum packet size that must be requested to support a desired data rate. These equations are a function of the service clock rate (F_X), bus clock rate (F_{SOF}), sample clock rate (F_S), bus access period (I), and sample size (S). These equations should provide design information for selecting the appropriate packet size that an endpoint will report in its characteristic information and the appropriate buffer requirements for the device/endpoint and its client software. Figure 5-17 shows actual buffer, packet, and clock values for a typical full-speed isochronous example.

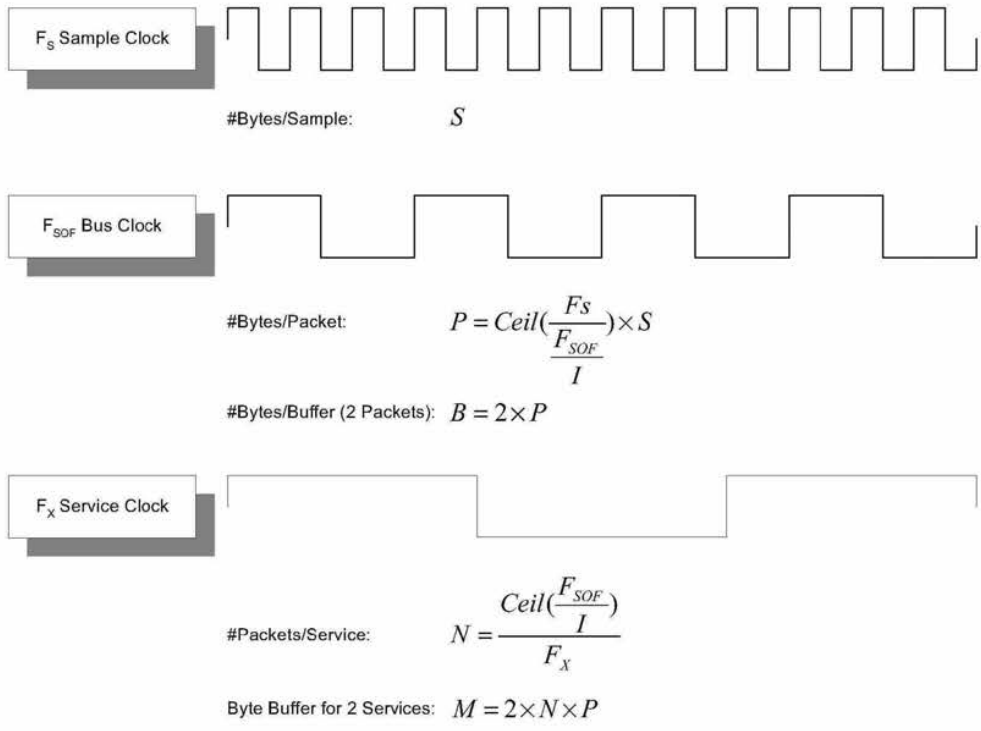

Figure 5-20. Packet and Buffer Size Formulas for Rate-matched Isochronous Transfers

The USB data model assumes that devices have some natural sample size and rate. The USB supports the transmission of packets that are multiples of sample size to make error recovery handling easier when isochronous transactions are damaged on the bus. If a device has no natural sample size or if its samples are larger than a packet, it should describe its sample size as being one byte. If a sample is split across a data packet, the error recovery can be harder when an arbitrary transaction is lost. In some cases, data synchronization can be lost unless the receiver knows in what (micro)frame number each partial sample is transmitted. Furthermore, if the number of samples can vary due to clock correction (e.g., for a non-derived device clock), it may be difficult or inefficient to know when a partial sample is transmitted. Therefore, the USB does not split samples across packets.

Chapter 6
Mechanical

This chapter provides the mechanical and electrical specifications for the cables, connectors, and cable assemblies used to interconnect USB devices. The specification includes the dimensions, materials, electrical, and reliability requirements. This chapter documents minimum requirements for the external USB interconnect. Substitute material may be used as long as it meets these minimums.

6.1 Architectural Overview

The USB physical topology consists of connecting the downstream hub port to the upstream port of another hub or to a device. The USB can operate at three speeds. High-speed (480 Mb/s) and full-speed (12 Mb/s) require the use of a shielded cable with two power conductors and twisted pair signal conductors. Low-speed (1.5 Mb/s) recommends, but does not require the use of a cable with twisted pair signal conductors.

The connectors are designed to be hot plugged. The USB Icon on the plugs provides tactile feedback making it easy to obtain proper orientation.

6.2 Keyed Connector Protocol

To minimize end user termination problems, USB uses a "keyed connector" protocol. The physical difference in the Series "A" and "B" connectors insures proper end user connectivity. The "A" connector is the principle means of connecting USB devices directly to a host or to the downstream port of a hub. All USB devices must have the standard Series "A" connector specified in this chapter. The "B" connector allows device vendors to provide a standard detachable cable. This facilitates end user cable replacement. Figure 6-1 illustrates the keyed connector protocol.

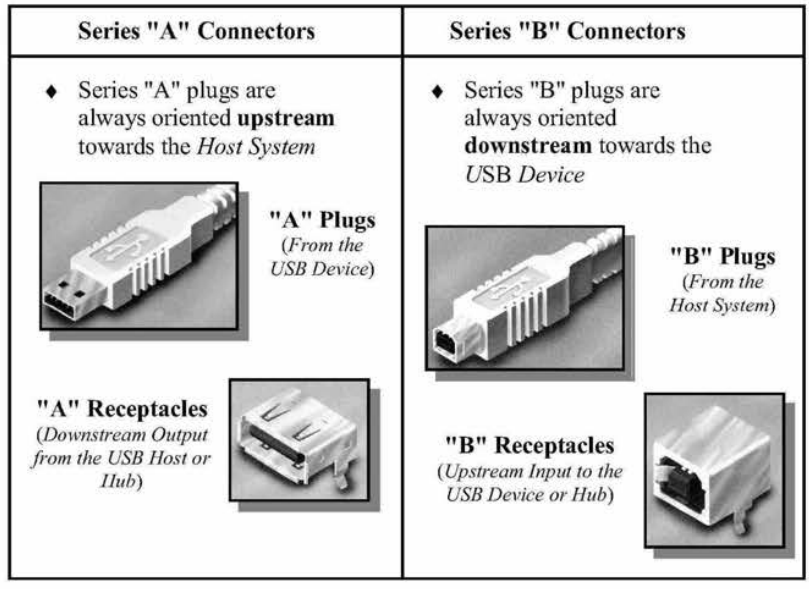

Figure 6-1. Keyed Connector Protocol

The following list explains how the plugs and receptacles can be mated:

- Series "A" receptacle mates with a Series "A" plug. Electrically, Series "A" receptacles function as outputs from host systems and/or hubs.
- Series "A" plug mates with a Series "A" receptacle. The Series "A" plug always is oriented towards the host system.
- Series "B" receptacle mates with a Series "B" plug (male). Electrically, Series "B" receptacles function as inputs to hubs or devices.
- Series "B" plug mates with a Series "B" receptacle. The Series "B" plug is always oriented towards the USB hub or device.

6.3 Cable

USB cable consists of four conductors, two power conductors, and two signal conductors.

High-/full-speed cable consists of a signaling twisted pair, VBUS, GND, and an overall shield. High-/full-speed cable must be marked to indicate suitability for USB usage (see Section 6.6.2). High-/full-speed cable may be used with either low-speed, full-speed, or high-speed devices. When high-/full-speed cable is used with low-speed devices, the cable must meet all low-speed requirements.

Low-speed recommends, but does not require the use of a cable with twisted signaling conductors.

6.4 Cable Assembly

This specification describes three USB cable assemblies: standard detachable cable, high-/full-speed captive cable, and low-speed captive cable.

A standard detachable cable is a high-/full-speed cable that is terminated on one end with a Series "A" plug and terminated on the opposite end with a series "B" plug. A high-/full-speed captive cable is terminated on one end with a Series "A" plug and has a vendor-specific connect means (hardwired or custom detachable) on the opposite end for the high-/full-speed peripheral. The low-speed captive cable is terminated on one end with a Series "A" plug and has a vendor-specific connect means (hardwired or custom detachable) on the opposite end for the low-speed peripheral. Any other cable assemblies are prohibited.

The color used for the cable assembly is vendor specific; recommended colors are white, grey, or black.

6.4.1 Standard Detachable Cable Assemblies

High-speed and full-speed devices can utilize the "B" connector. This allows the device to have a standard detachable USB cable. This eliminates the need to build the device with a hardwired cable and minimizes end user problems if cable replacement is necessary.

Devices utilizing the "B" connector must be designed to work with worst case maximum length detachable cable. Standard detachable cable assemblies may be used only on high-speed and full-speed devices. Using a high-/full-speed standard detachable cable on a low-speed device may exceed the maximum low-speed cable length.

Figure 6-2 illustrates a standard detachable cable assembly.

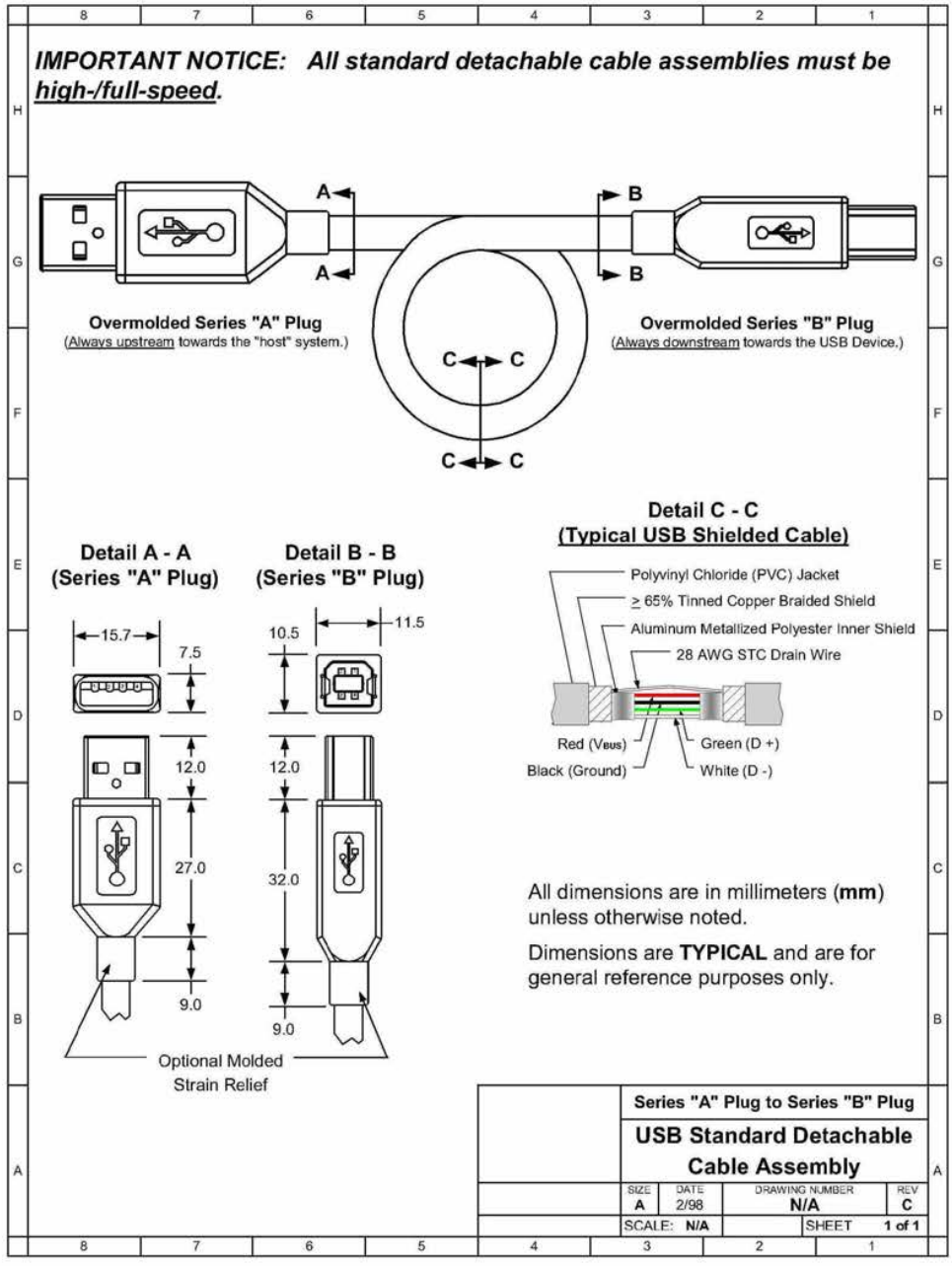

Figure 6-2. USB Standard Detachable Cable Assembly

Standard detachable cable assemblies must meet the following electrical requirements:

- The cable must be terminated on one end with an overmolded Series "A" plug and the opposite end is terminated with an overmolded Series "B" plug.
- The cable must be rated for high-speed and full-speed.
- The cable impedance must match the impedance of the high-speed and full-speed drivers. The drivers are characterized to drive specific cable impedance. Refer to Section 7.1.1 for details.
- The maximum allowable cable length is determined by signal pair attenuation and propagation delay. Refer to Sections 7.1.14 and 7.1.17 for details.
- Differences in propagation delay between the two signal conductors must be minimized. Refer to Section 7.1.3 for details.
- The GND lead provides a common ground reference between the upstream and downstream ports. The maximum cable length is limited by the voltage drop across the GND lead. Refer to Section 7.2.2 for details. The minimum acceptable wire gauge is calculated assuming the attached device is high power.
- The VBUS lead provides power to the connected device. For standard detachable cables, the VBUS requirement is the same as the GND lead.

6.4.2 High-/full-speed Captive Cable Assemblies

Assemblies are considered captive if they are provided with a vendor-specific connect means (hardwired or custom detachable) to the peripheral. High-/full-speed hardwired cable assemblies may be used with either high-speed, full-speed, or low-speed devices. When using a high-/full-speed hardwired cable on a low-speed device, the cable must meet all low-speed requirements.

Figure 6-3 illustrates a high-/full-speed hardwired cable assembly.

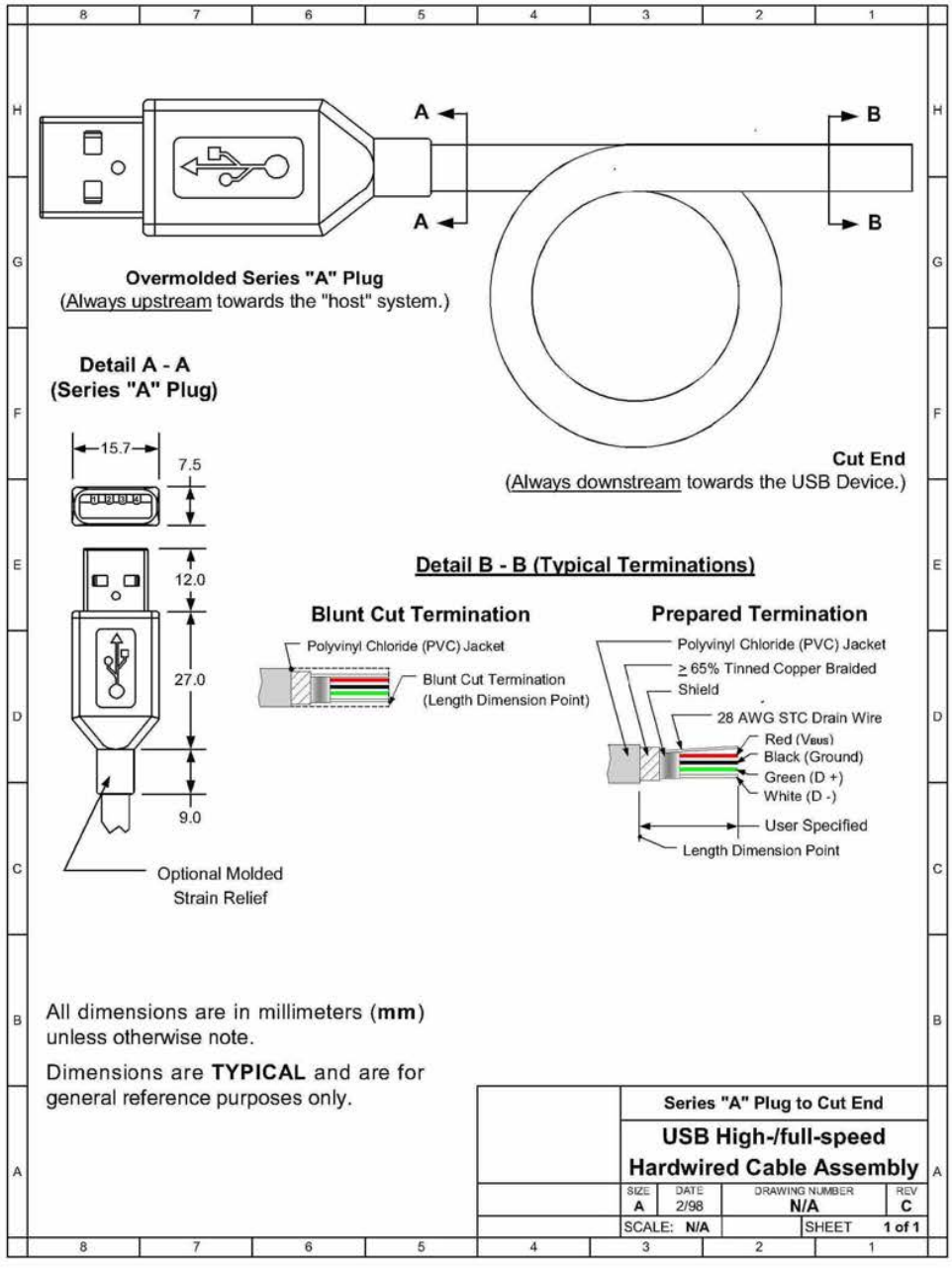

Figure 6-3. USB High-/full-speed Hardwired Cable Assembly

High-/full-speed captive cable assemblies must meet the following electrical requirements:

- The cable must be terminated on one end with an overmolded Series "A" plug and the opposite end is vendor specific. If the vendor specific interconnect is to be hot plugged, it must meet the same performance requirements as the USB "B" connector.
- The cable must be rated for high-speed and full-speed.
- The cable impedance must match the impedance of the high-speed and full-speed drivers. The drivers are characterized to drive specific cable impedance. Refer to Section 7.1.1 for details.
- The maximum allowable cable length is determined by signal pair attenuation and propagation delay. Refer to Sections 7.1.14 and 7.1.17 for details.
- Differences in propagation delay between the two signal conductors must be minimized. Refer to Section 7.1.3 for details.
- The GND lead provides a common reference between the upstream and downstream ports. The maximum cable length is determined by the voltage drop across the GND lead. Refer to Section 7.2.2 for details. The minimum wire gauge is calculated using the worst case current consumption.
- The VBUS lead provides power to the connected device. The minimum wire gauge is vendor specific.

6.4.3 Low-speed Captive Cable Assemblies

Assemblies are considered captive if they are provided with a vendor-specific connect means (hardwired or custom detachable) to the peripheral. Low-speed cables may only be used on low-speed devices.

Figure 6-4 illustrates a low-speed hardwired cable assembly.

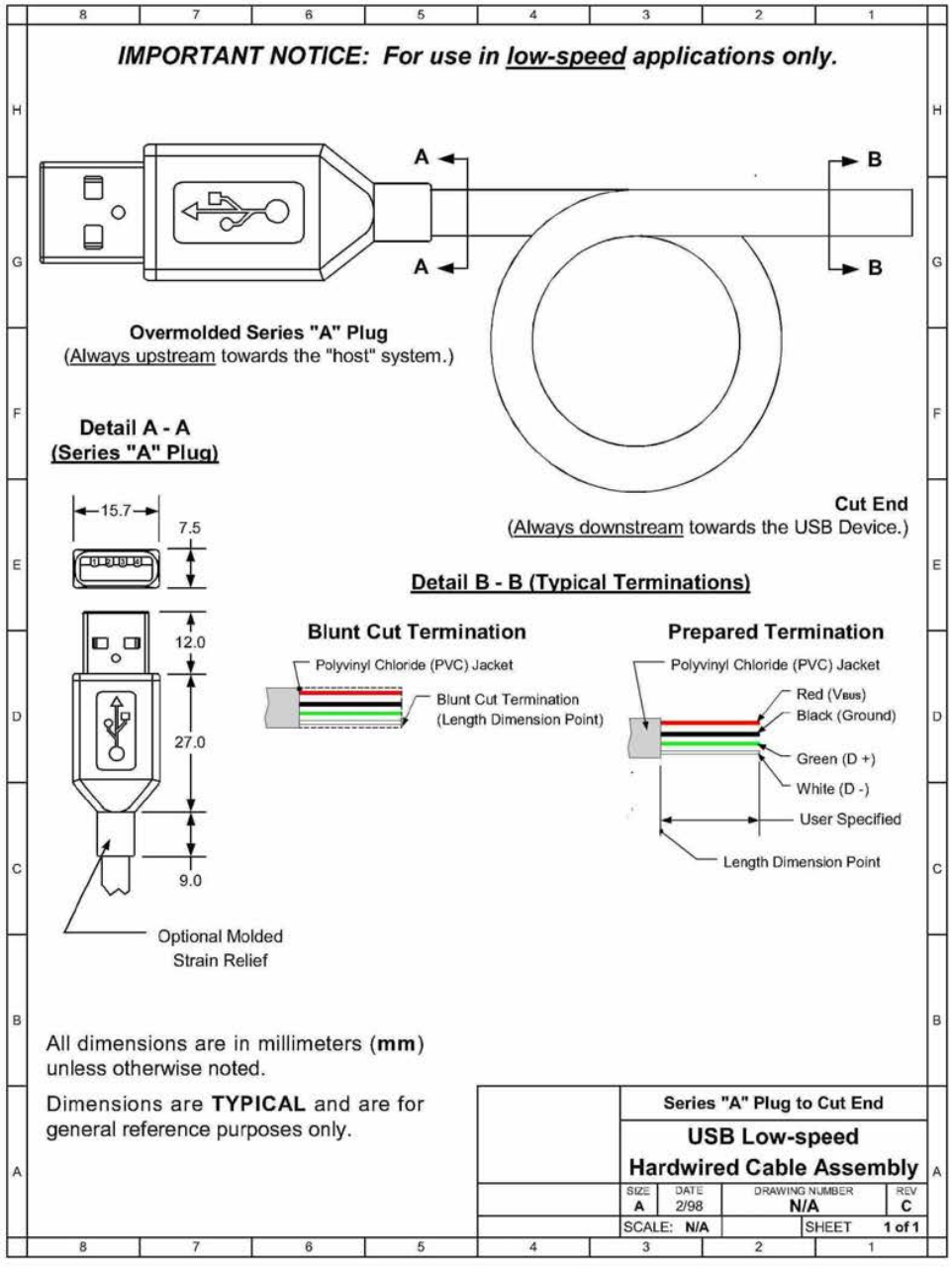

Figure 6-4. USB Low-speed Hardwired Cable Assembly

Low-speed captive cable assemblies must meet the following electrical requirements:

- The cable must be terminated on one end with an overmolded Series "A" plug and the opposite end is vendor specific. If the vendor specific interconnect is to be hot plugged, it must meet the same performance requirements as the USB "B" connector.
- Low-speed drivers are characterized for operation over a range of capacitive loads. This value includes all sources of capacitance on the D+ and D-lines, not just the cable. Cable selection must insure that total load capacitance falls between specified minimum and maximum values. If the desired implementation does not meet the minimum requirement, additional capacitance needs to be added to the device. Refer to Section 7.1.1.2 for details.
- The maximum low-speed cable length is determined by the rise and fall times of low-speed signaling. This forces low-speed cable to be significantly shorter than high-/full-speed. Refer to Section 7.1.1.2 for details.
- Differences in propagation delay between the two signal conductors must be minimized. Refer to Section 7.1.3 for details.
- The GND lead provides a common reference between the upstream and downstream ports. The maximum cable length is determined by the voltage drop across the GND lead. Refer to Section 7.2.2 for details. The minimum wire gauge is calculated using the worst case current consumption.
- The VBUS lead provides power to the connected device. The minimum wire gauge is vendor specific.

6.4.4 Prohibited Cable Assemblies

USB is optimized for ease of use. The expectation is that if the device can be plugged in, it will work. By specification, the only conditions that prevent a USB device from being successfully utilized are lack of power, lack of bandwidth, and excessive topology depth. These conditions are well understood by the system software.

Prohibited cable assemblies may work in some situations, but they cannot be guaranteed to work in all instances.

- **Extension cable assembly**
  A cable assembly that provides a Series "A" plug with a series "A" receptacle or a Series "B" plug with a Series "B" receptacle. This allows multiple cable segments to be connected together, possibly exceeding the maximum permissible cable length.
- **Cable assembly that violates USB topology rules**
  A cable assembly with both ends terminated in either Series "A" plugs or Series "B" receptacles. This allows two downstream ports to be directly connected.

  Note: This prohibition does not prevent using a USB device to provide a bridge between two USB buses.
- **Standard detachable cables for low-speed devices**
  Low-speed devices are prohibited from using standard detachable cables. A standard detachable cable assembly must be high-/full-speed. Since a standard detachable cable assembly is high-/full-speed rated, using a long high-/full-speed cable exceeds the capacitive load of low-speed.

6.5 Connector Mechanical Configuration and Material Requirements

The USB Icon is used to identify USB plugs and the receptacles. Figure 6-5 illustrates the USB Icon.

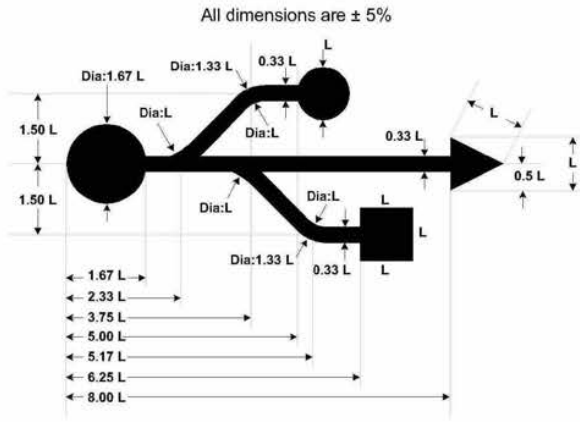

Figure 6-5. USB Icon

6.5.1 USB Icon Location

The USB Icon is embossed, in a recessed area, on the topside of the USB plug. This provides easy user recognition and facilitates alignment during the mating process. The USB Icon and Manufacturer's logo should not project beyond the overmold surface. The USB Icon is required, while the Manufacturer's logo is recommended, for both Series "A" and "B" plug assemblies. The USB Icon is also located adjacent to each receptacle. Receptacles should be oriented to allow the Icon on the plug to be visible during the mating process. Figure 6-6 illustrates the typical plug orientation.

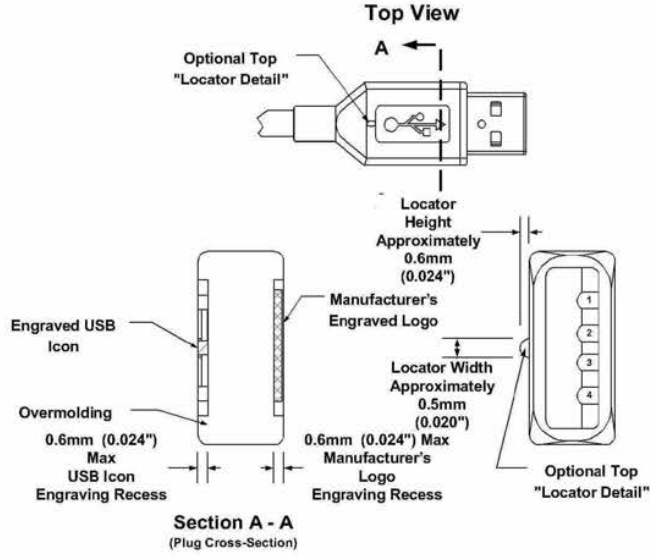

Figure 6-6. Typical USB Plug Orientation

6.5.2 USB Connector Termination Data

Table 6-1 provides the standardized contact terminating assignments by number and electrical value for Series "A" and Series "B" connectors.

Table 6-1. USB Connector Termination Assignment

Contact Number	Signal Name	Typical Wiring Assignment
1	VBUS	Red
2	D-	White
3	D+	Green
4	GND	Black
Shell	Shield	Drain Wire

6.5.3 Series "A" and Series "B" Receptacles

Electrical and mechanical interface configuration data for Series "A" and Series "B" receptacles are shown in Figure 6-7 and Figure 6-8. Also, refer to Figure 6-12, Figure 6-13, and Figure 6-14 at the end of this chapter for typical PCB receptacle layouts.

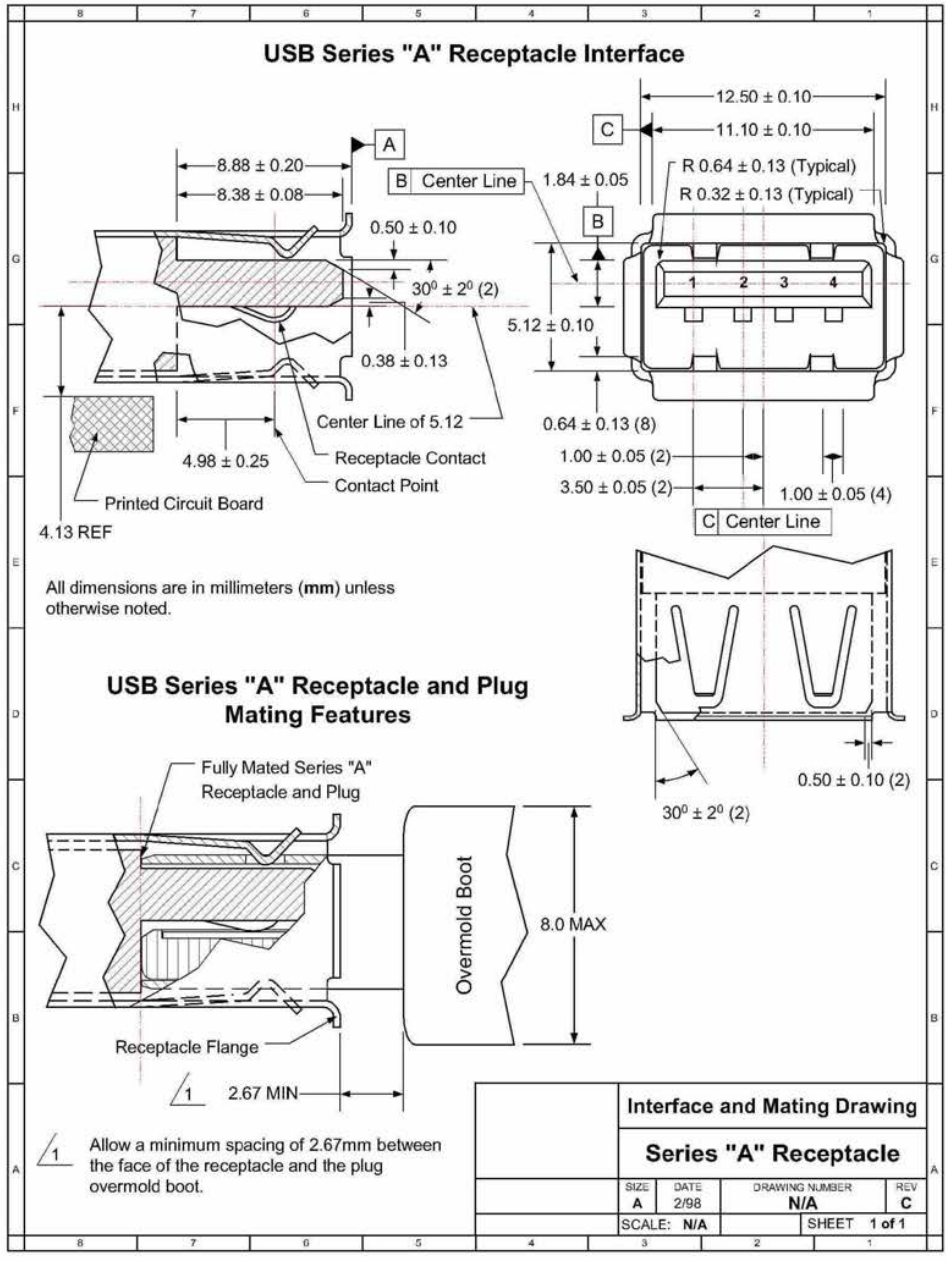

Figure 6-7. USB Series "A" Receptacle Interface and Mating Drawing

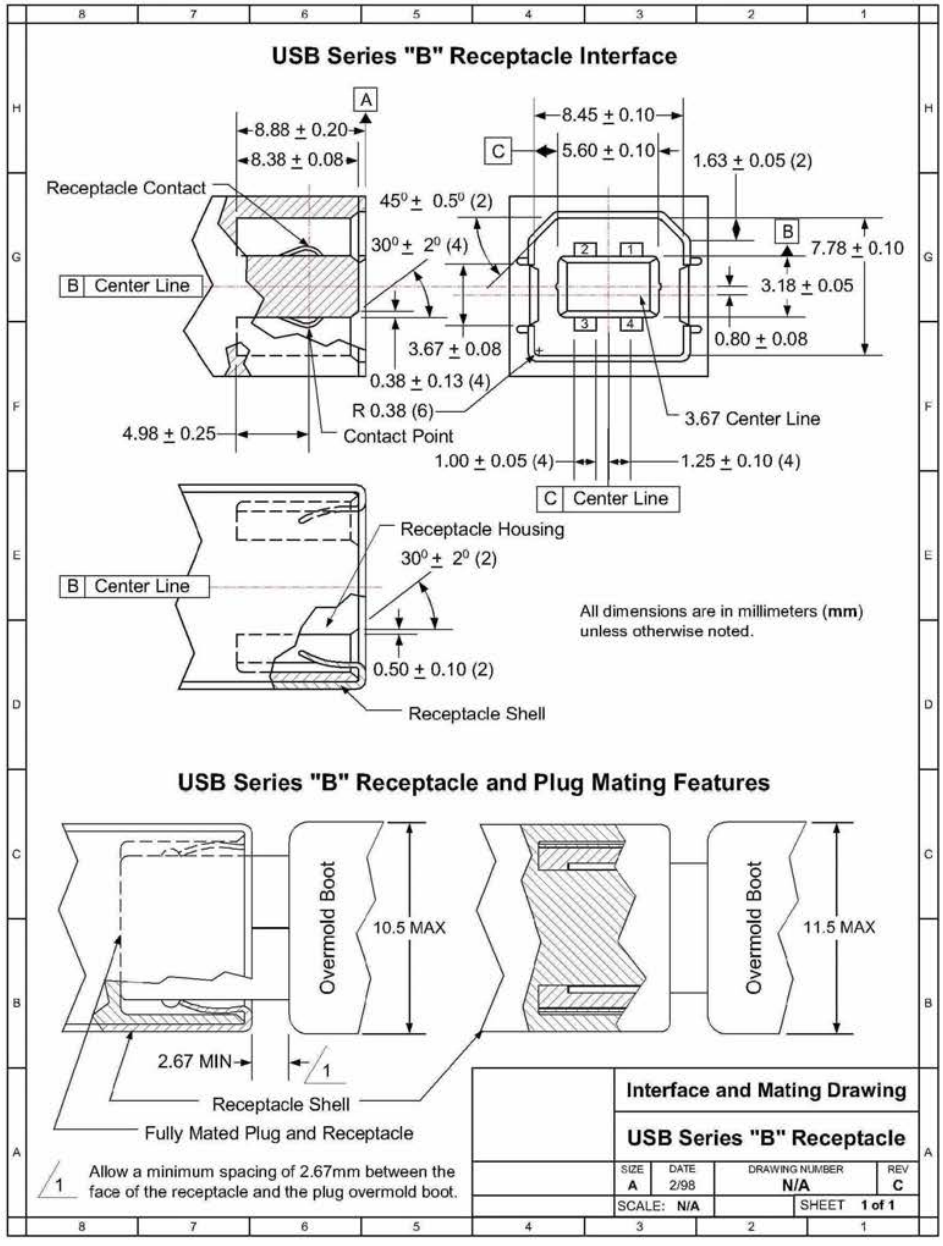

Figure 6-8. USB Series "B" Receptacle Interface and Mating Drawing

6.5.3.1 Receptacle Injection Molded Thermoplastic Insulator Material

Minimum UL 94-V0 rated, thirty percent (30%) glass-filled polybutylene terephthalate (PBT) or polyethylene terephthalate (PET) or better.

Typical Colors: Black, gray, and natural.

Flammability Characteristics: UL 94-V0 rated.

Flame Retardant Package must meet or exceed the requirements for UL, CSA, VDE, etc.

Oxygen Index (LOI): Greater than 21%. ASTM D 2863.

6.5.3.2 Receptacle Shell Materials

Substrate Material: 0.30 + 0.05 mm phosphor bronze, nickel silver, or other copper based high strength materials.

Plating:

1. Underplate: Optional. Minimum 1.00 micrometers (40 microinches) nickel. In addition, manufacturer may use a copper underplate beneath the nickel.
2. Outside: Minimum 2.5 micrometers (100 microinches) bright tin or bright tin-lead.

6.5.3.3 Receptacle Contact Materials

Substrate Material: 0.30 ± 0.05 mm minimum half-hard phosphor bronze or other high strength copper based material.

Plating: Contacts are to be selectively plated.

A. Option I
   1. Underplate: Minimum 1.25 micrometers (50 microinches) nickel. Copper over base material is optional.
   2. Mating Area: Minimum 0.05 micrometers (2 microinches) gold over a minimum of 0.70 micrometers (28 microinches) palladium.
   3. Solder Tails: Minimum 3.8 micrometers (150 microinches) bright tin-lead over the underplate.

B. Option II
   1. Underplate: Minimum 1.25 micrometers (50 microinches) nickel. Copper over base material is optional.
   2. Mating Area: Minimum 0.05 micrometers (2 microinches) gold over a minimum of 0.75 micrometers (30 microinches) palladium-nickel.
   3. Solder Tails: Minimum 3.8 micrometers (150 microinches) bright tin-lead over the underplate.

C. Option III
   1. Underplate: Minimum 1.25 micrometers (50 microinches) nickel. Copper over base material is optional.
   2. Mating Area: Minimum 0.75 micrometers (30 microinches) gold.
   3. Solder Tails: Minimum 3.8 micrometers (150 microinches) bright tin-lead over the underplate.

6.5.4 Series “A” and Series “B” Plugs

Electrical and mechanical interface configuration data for Series "A" and Series "B" plugs are shown in Figure 6-9 and Figure 6-10.

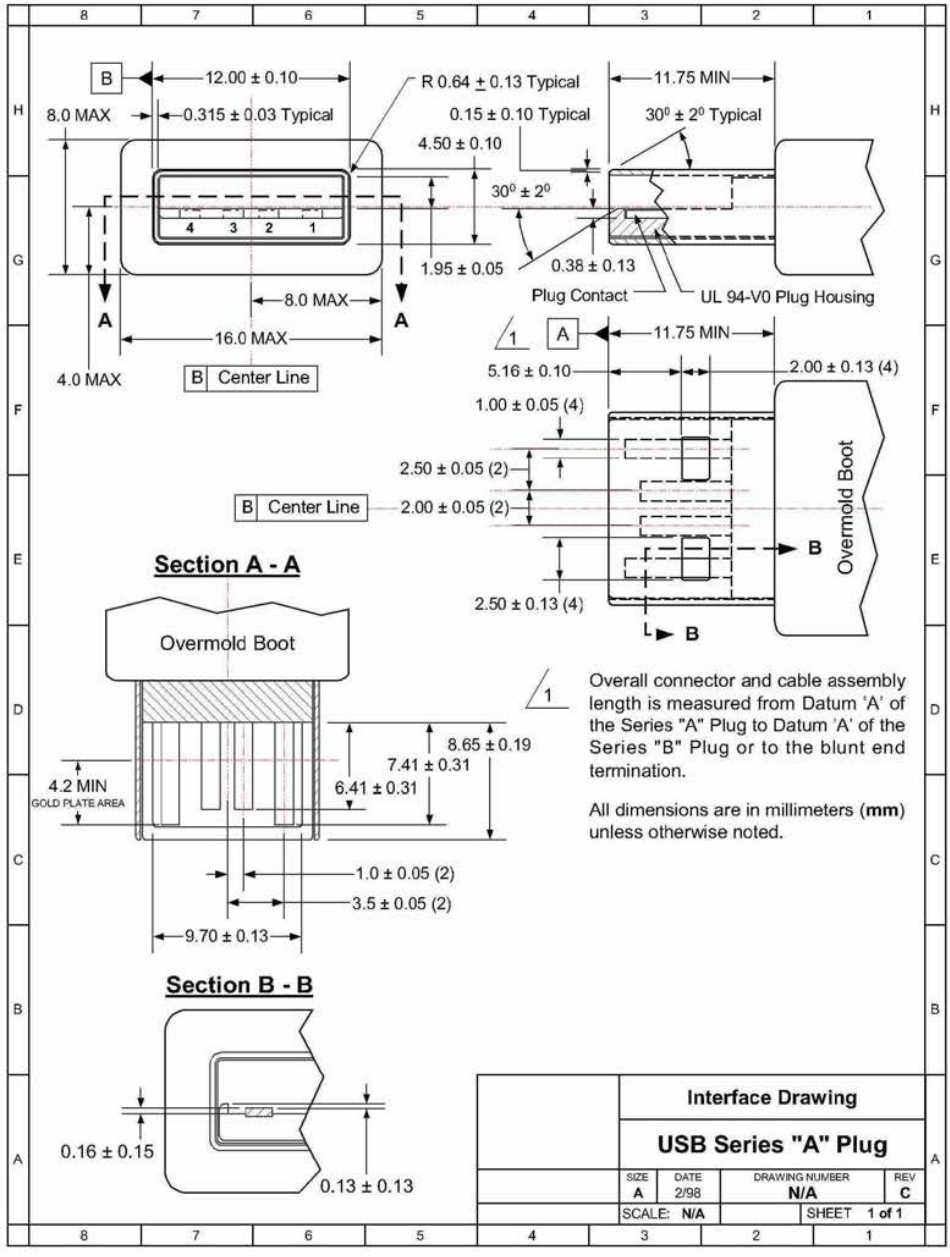

Figure 6-9. USB Series "A" Plug Interface Drawing

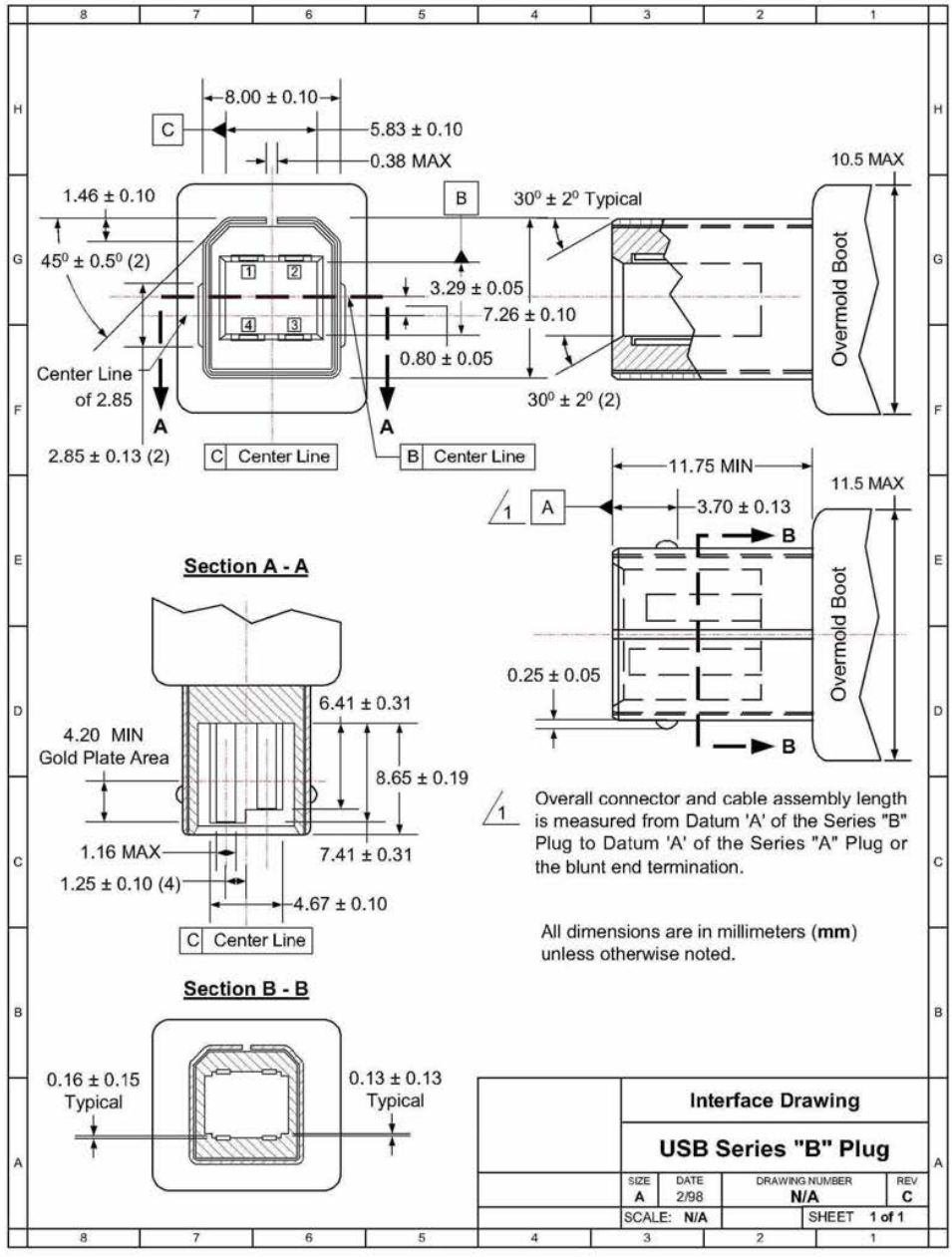

Figure 6-10. USB Series "B" Plug Interface Drawing

6.5.4.1 Plug Injection Molded Thermoplastic Insulator Material

Minimum UL 94-V0 rated, thirty percent (30%) glass-filled polybutylene terephthalate (PBT) or polyethylene terephthalate (PET) or better.

Typical Colors: Black, gray, and natural.

Flammability Characteristics: UL 94-V0 rated.

Flame Retardant Package must meet or exceed the requirements for UL, CSA, and VDE.

Oxygen Index (LOI): 21%. ASTM D 2863.

6.5.4.2 Plug Shell Materials

Substrate Material: 0.30 ± 0.05 mm phosphor bronze, nickel silver, or other suitable material.

Plating:

A. Underplate: Optional. Minimum 1.00 micrometers (40 microinches) nickel. In addition, manufacturer may use a copper underplate beneath the nickel.

B. Outside: Minimum 2.5 micrometers (100 microinches) bright tin or bright tin-lead.

6.5.4.3 Plug (Male) Contact Materials

Substrate Material: 0.30 ± 0.05 mm half-hard phosphor bronze.

Plating: Contacts are to be selectively plated.

A. Option I

1. Underplate: Minimum 1.25 micrometers (50 microinches) nickel. Copper over base material is optional.
2. Mating Area: Minimum 0.05 micrometers (2 microinches) gold over a minimum of 0.70 micrometers (28 microinches) palladium.
3. Solder Tails: Minimum 3.8 micrometers (150 microinches) bright tin-lead over the underplate.

B. Option II

1. Underplate: Minimum 1.25 micrometers (50 microinches) nickel. Copper over base material is optional.
2. Mating Area: Minimum 0.05 micrometers (2 microinches) gold over a minimum of 0.75 micrometers (30 microinches) palladium-nickel.
3. Wire Crimp/Solder Tails: Minimum 3.8 micrometers (150 microinches) bright tin-lead over the underplate.

C. Option III

1. Underplate: Minimum 1.25 micrometers (50 microinches) nickel. Copper over base material is optional.
2. Mating Area: Minimum 0.75 micrometers (30 microinches) gold.
3. Solder Tails: Minimum 3.8 micrometers (150 microinches) bright tin-lead over the underplate.

6.6 Cable Mechanical Configuration and Material Requirements

High-/full-speed and low-speed cables differ in data conductor arrangement and shielding. Low-speed recommends, but does not require, use of a cable with twisted data conductors. Low speed recommends, but does not require, use of a cable with a braided outer shield. Figure 6-11 shows the typical high-/full-speed cable construction.

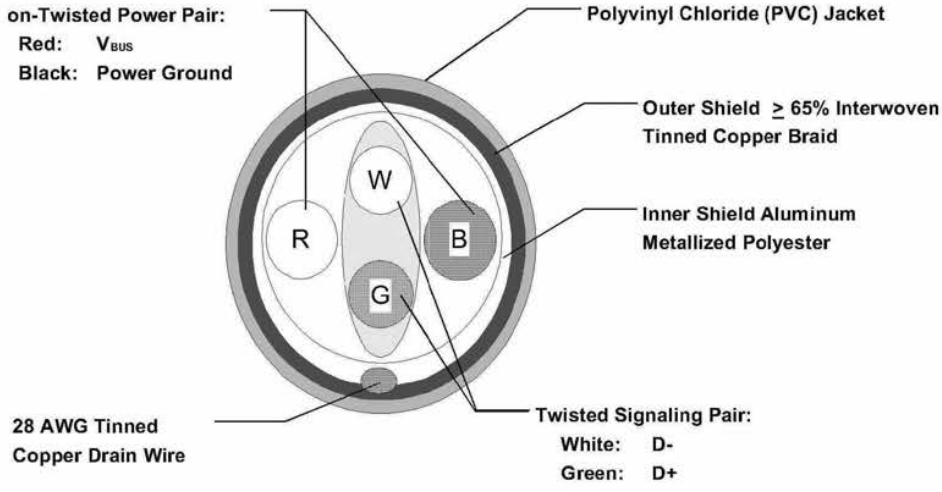

Figure 6-11. Typical High-/full-speed Cable Construction

6.6.1 Description

High-/full-speed cable consists of one 28 to 20 AWG non-twisted power pair and one 28 AWG twisted data pair with an aluminum metallized polyester inner shield, 28 AWG stranded tinned copper drain wire, ≥ 65% tinned copper wire interwoven (braided) outer shield, and PVC outer jacket.

Low-speed cable consists of one 28 to 20 AWG non-twisted power pair and one 28 AWG data pair (a twist is recommended) with an aluminum metallized polyester inner shield, 28 AWG stranded tinned copper drain wire and PVC outer jacket. A ≥ 65% tinned copper wire interwoven (braided) outer shield is recommended.

6.6.2 Construction

Raw materials used in the fabrication of this cable must be of such quality that the fabricated cable is capable of meeting or exceeding the mechanical and electrical performance criteria of the most current USB Specification revision and all applicable domestic and international safety/testing agency requirements; e.g., UL, CSA, BSA, NEC, etc., for electronic signaling and power distribution cables in its category.

Table 6-2. Power Pair

American Wire Gauge (AWG)	Nominal Conductor Outer Diameter	Stranded Tinned Conductors
28	0.381 mm (0.015")	7 x 36
	0.406 mm (0.016")	19 x 40
26	0.483 mm (0.019")	7 x 34
	0.508 mm (0.020")	19 x 38
24	0.610 mm (0.024")	7 x 32
	0.610 mm (0.024")	19 x 36
22	0.762 mm (0.030")	7 x 30
	0.787 mm (0.031")	19 x 34
20	0.890 mm (0.035")	7 x 28
	0.931 mm (0.037")	19 x 32

Note: Minimum conductor construction must be stranded tinned copper.

Non-Twisted Power Pair:

A. Wire Gauge: Minimum 28 AWG or as specified by the user contingent upon the specified cable length. Refer to Table 6-2.

B. Wire Insulation: Semirigid polyvinyl chloride (PVC).

   1. Nominal Insulation Wall Thickness: 0.25 mm (0.010")
   2. Typical Power (V_{BUS}) Conductor: Red Insulation
   3. Typical Ground Conductor: Black Insulation

Signal Pair:

A. Wire Gauge: 28 AWG minimum. Refer to Table 6-3.

Table 6-3. Signal Pair

American Wire Gauge (AWG)	Nominal Conductor Outer Diameter	Stranded Tinned Conductors
28	0.381 mm (0.015")	7 x 36
	0.406 mm (0.016")	19 x 40

Note: Minimum conductor construction must be stranded tinned copper.

B. Wire Insulation: High-density polyethylene (HDPE), alternately foamed polyethylene or foamed polypropylene

1. Nominal Insulation Wall Thickness: 0.31 mm (0.012")
2. Typical Data Plus (+) Conductor: Green Insulation
3. Typical Data Minus (-) Conductor: White Insulation

C. Nominal Twist Ratio (not required for low-speed): One full twist every 60 mm (2.36") to 80 mm (3.15")

Aluminum Metallized Polyester Inner Shield (required for low-speed):

A. Substrate Material: Polyethylene terephthalate (PET) or equivalent material

B. Metallizing: Vacuum deposited aluminum

C. Assembly:

1. The aluminum metallized side of the inner shield must be positioned facing out to ensure direct contact with the drain wire.
2. The aluminum metallized inner shield must overlap by approximately one-quarter turn.

Drain Wire (required for low-speed):

A. Wire Gauge: Minimum 28 AWG stranded tinned copper (STC) non-insulated. Refer to Table 6-4.

Table 6-4. Drain Wire Signal Pair

American Wire Gauge (AWG)	Nominal Conductor Outer Diameter	Stranded Tinned Conductors
28	0.381 mm (0.015")	7 x 36
	0.406 mm (0.016")	19 x 40

Interwoven (Braided) Tinned Copper Wire (ITCW) Outer Shield (recommended but not required for low-speed):

A. Coverage Area: Minimum 65%.

B. Assembly: The interwoven (braided) tinned copper wire outer shield must encase the aluminum metallized PET shielded power and signal pairs and must be in direct contact with the drain wire.

Outer Polyvinyl Chloride (PVC) Jacket:

A. Assembly: The outer PVC jacket must encase the fully shielded power and signal pairs and must be in direct contact with the tinned copper outer shield.

B. Nominal Wall Thickness: 0.64 mm (0.025").

Marking: The cable must be legibly marked using contrasting color permanent ink.

A. Minimum marking information for high-/full-speed cable must include:

USB SHIELDED <Gauge/2C + Gauge/2C> UL CM 75 °C — UL Vendor ID.

B. Minimum marking information for low-speed cable shall include:

USB specific marking is not required for low-speed cable.

Nominal Fabricated Cable Outer Diameter:

This is a nominal value and may vary slightly from manufacturer to manufacturer as a function of the conductor insulating materials and conductor specified. Refer to Table 6-5.

Table 6-5. Nominal Cable Diameter

Shielded USB Cable Configuration	Nominal Outer Cable Diameter
28/28	4.06 mm (0.160")
28/26	4.32 mm (0.170")
28/24	4.57 mm (0.180")
28/22	4.83 mm (0.190")
28/20	5.21 mm (0.205")

6.6.3 Electrical Characteristics

All electrical characteristics must be measured at or referenced to +20 °C (68 °F).

Voltage Rating: 30 V rms maximum.

Conductor Resistance: Conductor resistance must be measured in accordance with ASTM-D-4566 Section 13. Refer to Table 6-6.

Conductor Resistance Unbalance (Pairs): Conductor resistance unbalance between two (2) conductors of any pair must not exceed five percent (5%) when measured in accordance with ASTM-D-4566 Section 15.

The DC resistance from plug shell to plug shell (or end of integrated cable) must be less than 0.6 ohms.

Table 6-6. Conductor Resistance

American Wire Gauge (AWG)	Ohms (Ω) / 100 Meters Maximum
28	23.20
26	14.60
24	9.09
22	5.74
20	3.58

6.6.4 Cable Environmental Characteristics

Temperature Range:

A. Operating Temperature Range: 0 °C to +50 °C

B. Storage Temperature Range: -20 °C to +60 °C

C. Nominal Temperature Rating: +20 °C

Flammability: All plastic materials used in the fabrication of this product shall meet or exceed the requirements of NEC Article 800 for communications cables Type CM (Commercial).

6.6.5 Listing

The product shall be UL listed per UL Subject 444, Class 2, Type CM for Communications Cable Requirements.

6.7 Electrical, Mechanical, and Environmental Compliance Standards

Table 6-7 lists the minimum test criteria for all USB cable, cable assemblies, and connectors.

Table 6-7. USB Electrical, Mechanical, and Environmental Compliance Standards

Test Description	Test Procedure	Performance Requirement
Visual and Dimensional Inspection	EIA 364-18 Visual, dimensional, and functional inspection in accordance with the USB quality inspection plans.	Must meet or exceed the requirements specified by the most current version of Chapter 6 of the USB Specification.
Insulation Resistance	EIA 364-21 The object of this test procedure is to detail a standard method to assess the insulation resistance of USB connectors. This test procedure is used to determine the resistance offered by the insulation materials and the various seals of a connector to a DC potential tending to produce a leakage of current through or on the surface of these members.	1,000 MΩ minimum.
Dielectric Withstanding Voltage	EIA 364-20 The object of this test procedure is to detail a test method to prove that a USB connector can operate safely at its rated voltage and withstand momentary over-potentials due to switching, surges, and/or other similar phenomena.	The dielectric must withstand 500 V AC for one minute at sea level.

Table 6-7. USB Electrical, Mechanical, and Environmental Compliance Standards (Continued)

Test Description	Test Procedure	Performance Requirement
Low Level Contact Resistance	EIA 364-23 The object of this test is to detail a standard method to measure the electrical resistance across a pair of mated contacts such that the insulating films, if present, will not be broken or asperity melting will not occur.	30 mΩ maximum when measured at 20 mV maximum open circuit at 100 mA. Mated test contacts must be in a connector housing.
Contact Current Rating	EIA 364-70 — Method B The object of this test procedure is to detail a standard method to assess the current carrying capacity of mated USB connector contacts.	1.5 A at 250 V AC minimum when measured at an ambient temperature of 25 °C. With power applied to the contacts, the Δ T must not exceed +30 °C at any point in the USB connector under test.
Contact Capacitance	EIA 364-30 The object of this test is to detail a standard method to determine the capacitance between conductive elements of a USB connector.	2 pF maximum unmated per contact.
Insertion Force	EIA 364-13 The object of this test is to detail a standard method for determining the mechanical forces required for inserting a USB connector.	35 Newtons maximum at a maximum rate of 12.5 mm (0.492") per minute.
Extraction Force	EIA 364-13 The object of this test is to detail a standard method for determining the mechanical forces required for extracting a USB connector.	10 Newtons minimum at a maximum rate of 12.5 mm (0.492") per minute.

Table 6-7. USB Electrical, Mechanical, and Environmental Compliance Standards (Continued)

Test Description	Test Procedure	Performance Requirement
Durability	EIA 364-09 The object of this test procedure is to detail a uniform test method for determining the effects caused by subjecting a USB connector to the conditioning action of insertion and extraction, simulating the expected life of the connectors. Durability cycling with a gauge is intended only to produce mechanical stress. Durability performed with mating components is intended to produce both mechanical and wear stress.	1,500 insertion/extraction cycles at a maximum rate of 200 cycles per hour.
Cable Pull-Out	EIA 364-38 Test Condition A The object of this test procedure is to detail a standard method for determining the holding effect of a USB plug cable clamp without causing any detrimental effects upon the cable or connector components when the cable is subjected to inadvertent axial tensile loads.	After the application of a steady state axial load of 40 Newtons for one minute.
Physical Shock	EIA 364-27 Test Condition H The object of this test procedure is to detail a standard method to assess the ability of a USB connector to withstand specified severity of mechanical shock.	No discontinuities of 1 µs or longer duration when mated USB connectors are subjected to 11 ms duration 30 Gs half-sine shock pulses. Three shocks in each direction applied along three mutually perpendicular planes for a total of 18 shocks.

Table 6-7. USB Electrical, Mechanical, and Environmental Compliance Standards (Continued)

Test Description	Test Procedure	Performance Requirement
Random Vibration	EIA 364-28 Test Condition V Test Letter A This test procedure is applicable to USB connectors that may, in service, be subjected to conditions involving vibration. Whether a USB connector has to function during vibration or merely to survive conditions of vibration should be clearly stated by the detailed product specification. In either case, the relevant specification should always prescribe the acceptable performance tolerances.	No discontinuities of 1 µs or longer duration when mated USB connectors are subjected to 5.35 Gs RMS. 15 minutes in each of three mutually perpendicular planes.
Thermal Shock	EIA 364-32 Test Condition I The object of this test is to determine the resistance of a USB connector to exposure at extremes of high and low temperatures and to the shock of alternate exposures to these extremes, simulating the worst case conditions for storage, transportation, and application.	10 cycles –55 °C and +85 °C. The USB connectors under test must be mated.
Humidity Life	EIA 364-31 Test Condition A Method III The object of this test procedure is to detail a standard test method for the evaluation of the properties of materials used in USB connectors as they are influenced by the effects of high humidity and heat.	168 hours minimum (seven complete cycles). The USB connectors under test must be tested in accordance with EIA 364-31.
Solderability	EIA 364-52 The object of this test procedure is to detail a uniform test method for determining USB connector solderability. The test procedure contained herein utilizes the solder dip technique. It is not intended to test or evaluate solder cup, solder eyelet, other hand-soldered type, or SMT type terminations.	USB contact solder tails must pass 95% coverage after one hour steam aging as specified in Category 2.

Table 6-7. USB Electrical, Mechanical, and Environmental Compliance Standards (Continued)

Test Description	Test Procedure	Performance Requirement
Flammability	UL 94 V-0 This procedure is to ensure thermoplastic resin compliance to UL flammability standards.	The manufacturer will require its thermoplastic resin vendor to supply a detailed C of C with each resin shipment. The C of C shall clearly show the resin's UL listing number, lot number, date code, etc.
Flammability	UL 94 V-0 This procedure is to ensure thermoplastic resin compliance to UL flammability standards.	The manufacturer will require its thermoplastic resin vendor to supply a detailed C of C with each resin shipment. The C of C shall clearly show the resin's UL listing number, lot number, date code, etc.
Cable Impedance (Only required for high-/full-speed)	The object of this test is to insure the signal conductors have the proper impedance. 1. Connect the Time Domain Reflectometer (TDR) outputs to the impedance/delay/skew test fixture (Note 1). Use separate 50 Ω cables for the plus (or true) and minus (or complement) outputs. Set the TDR head to differential TDR mode. 2. Connect the Series "A" plug of the cable to be tested to the text fixture, leaving the other end open-circuited. 3. Define a waveform composed of the difference between the true and complement waveforms, to allow measurement of differential impedance. 4. Measure the minimum and maximum impedances found between the connector and the open circuited far end of the cable.	Impedance must be in the range specified in Table 7-9 (ZO).

Table 6-7. USB Electrical, Mechanical, and Environmental Compliance Standards (Continued)

Test Description	Test Procedure	Performance Requirement
Signal Pair Attenuation (Only required for high-/full-speed)	The object of this test is to insure that adequate signal strength is presented to the receiver to maintain a low error rate. 1. Connect the Network Analyzer output port (port 1) to the input connector on the attenuation test fixture (Note 2). 2. Connect the Series "A" plug of the cable to be tested to the test fixture, leaving the other end open-circuited. 3. Calibrate the network analyzer and fixture using the appropriate calibration standards over the desired frequency range. 4. Follow the method listed in Hewlett Packard Application Note 380-2 to measure the open-ended response of the cable. 5. Short circuit the Series "B" end (or bare leads end, if a captive cable) and measure the short-circuit response. 6. Using the software in H-P App. Note 380-2 or equivalent, calculate the cable attenuation accounting for resonance effects in the cable as needed.	Refer to Section 7.1.17 for frequency range and allowable attenuation.

Table 6-7. USB Electrical, Mechanical, and Environmental Compliance Standards (Continued)

Test Description	Test Procedure	Performance Requirement
Propagation Delay	The purpose of the test is to verify the end to end propagation of the cable. 1. Connect one output of the TDR sampling head to the D+ and D- inputs of the impedance/delay/skew test fixture (Note 1). Use one 50 Ω cable for each signal and set the TDR head to differential TDR mode. 2. Connect the cable to be tested to the test fixture. If detachable, plug both connectors in to the matching fixture connectors. If captive, plug the series "A" plug into the matching fixture connector and solder the stripped leads on the other end to the test fixture. 3. Measure the propagation delay of the test fixture by connecting a short piece of wire across the fixture from input to output and recording the delay. 4. Remove the short piece of wire and remeasure the propagation delay. Subtract from it the delay of the test fixture measured in the previous step.	High-/full-speed. See Section 7.1.1.1, Section 7.1.4, Section 7.1.16, and Table 7-9 (TFSCBL). Low-speed. See Section 7.1.1.2, Section 7.1.16, and Table 7-9 (TLSCBL).

Table 6-7. USB Electrical, Mechanical, and Environmental Compliance Standards (Continued)

Test Description	Test Procedure	Performance Requirement
Propagation Delay Skew	This test insures that the signal on both the D+ and D- lines arrive at the receiver at the same time. 1. Connect the TDR to the fixture with test sample cable, as in the previous section. 2. Measure the difference in delay for the two conductors in the test cable. Use the TDR cursors to find the open-circuited end of each conductor (where the impedance goes infinite) and subtract the time difference between the two values.	Propagation skew must meet the requirements as listed in Section 7.1.3.
Capacitive Load Only required for low-speed	The purpose of this test is to insure the distributed inter-wire capacitance is less than the lumped capacitance specified by the low-speed transmit driver. 1. Connect the one lead of the Impedance Analyzer to the D+ pin on the impedance/delay/skew fixture (Note 1) and the other lead to the D- pin. 2. Connect the series "A" plug to the fixture, with the series "B" end leads open-circuited. 3. Set the Impedance Analyzer to a frequency of 100 kHz, to measure the capacitance.	See Section 7.1.1.2 and Table 7-7 (CLINUA).

Note1: Impedance, propagation delay, and skew test fixture
This fixture will be used with the TDR for measuring the time domain performance of the cable under test. The fixture impedance should be matched to the equipment, typically 50 Ω. Coaxial connectors should be provided on the fixture for connection from the TDR.

Note 2: Attenuation text fixture
This fixture provides a means of connection from the network analyzer to the Series "A" plug. Since USB signals are differential in nature and operate over balanced cable, a transformer or balun (North Hills NH13734 or equivalent) is ideally used. The transformer converts the unbalanced (also known as single-ended) signal from the signal generator which is typically a 50 Ω output to the balanced (also known as differential) and likely different impedance loaded presented by the cable. A second transformer or balun should be used on the other end of the cable under test to convert the signal back to unbalanced form of the correct impedance to match the network analyzer.

6.7.1 Applicable Documents

American National Standard/Electronic Industries Association

ANSI/EIA-364-C (12/94)	Electrical Connector/Socket Test Procedures Including Environmental Classifications

American Standard Test Materials

ASTM-D-4565	Physical and Environmental Performance Properties of Insulation and Jacket for Telecommunication Wire and Cable, Test Standard Method
ASTM-D-4566	Electrical Performance Properties of Insulation and Jacket for Telecommunication Wire and Cable, Test Standard Method

Underwriters' Laboratory, Inc.

UL STD-94	Test for Flammability of Plastic materials for Parts in Devices and Appliances
UL Subject-444	Communication Cables

6.8 USB Grounding

The shield must be terminated to the connector plug for completed assemblies. The shield and chassis are bonded together. The user selected grounding scheme for USB devices, and cables must be consistent with accepted industry practices and regulatory agency standards for safety and EMI/ESD/RFI.

6.9 PCB Reference Drawings

The drawings in Figure 6-12, Figure 6-13, and Figure 6-14 describe typical receptacle PCB interfaces. These drawings are included for informational purposes only.

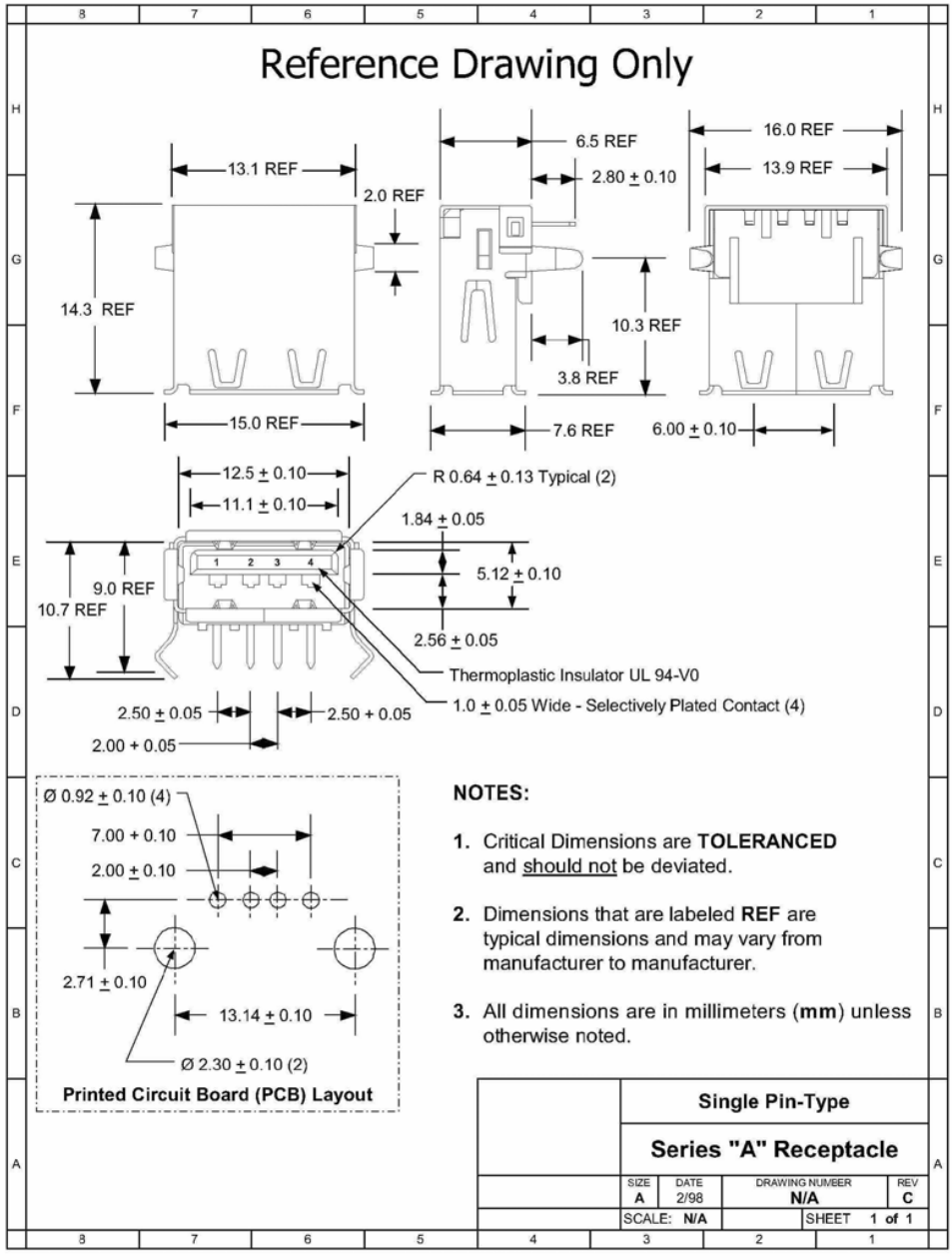

Figure 6-12. Single Pin-type Series "A" Receptacle

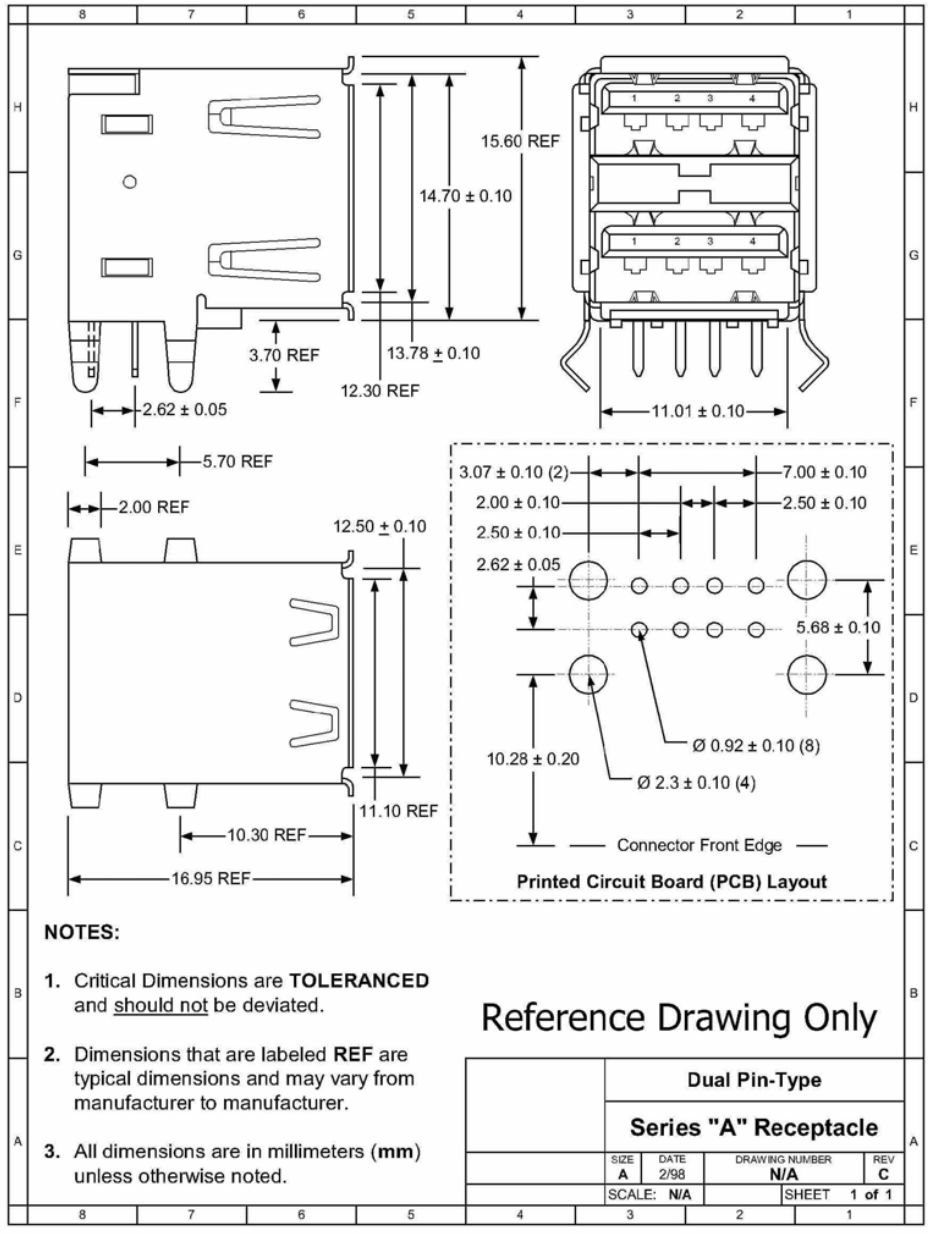

Figure 6-13. Dual Pin-type Series "A" Receptacle

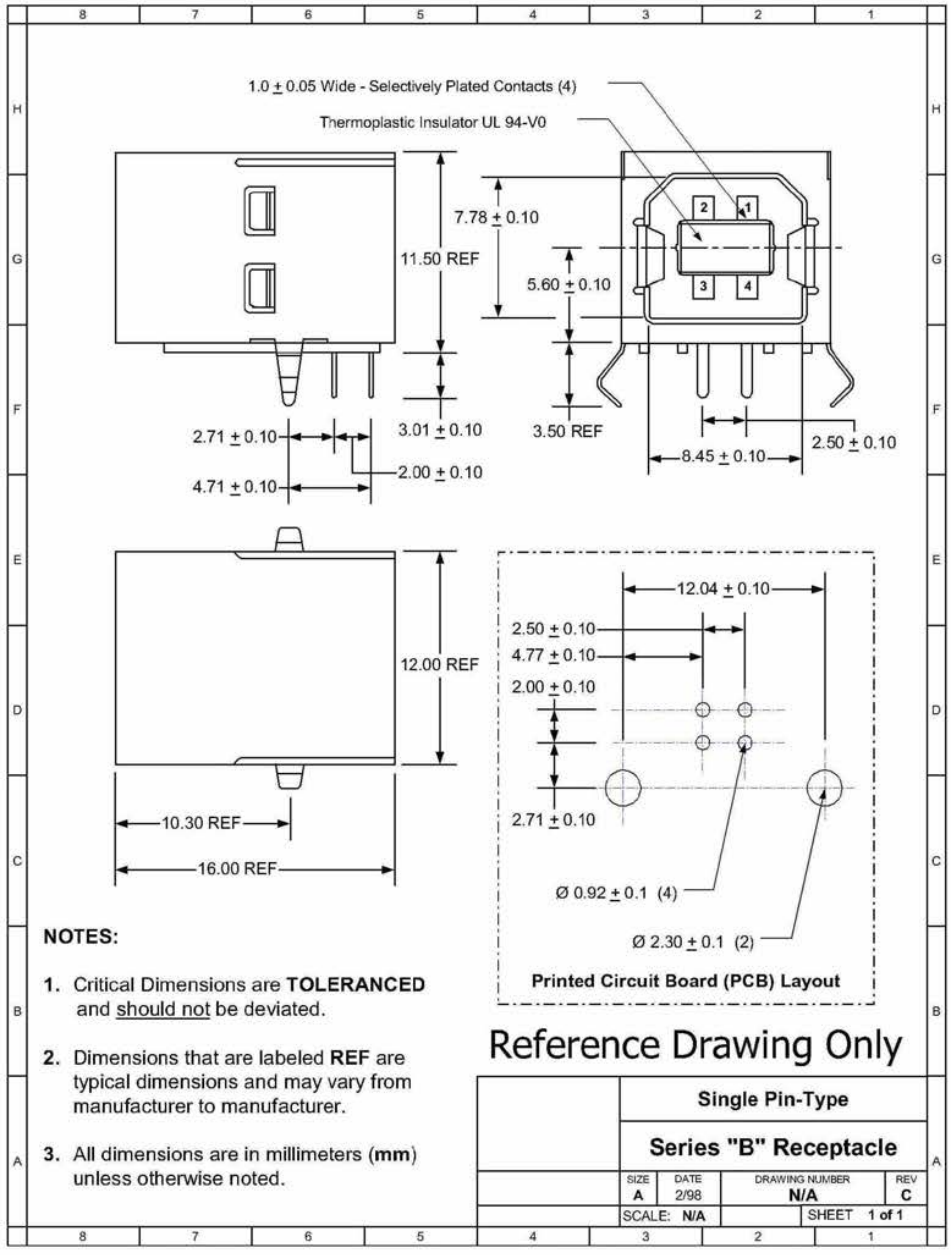

Figure 6-14. Single Pin-type Series "B" Receptacle

Chapter 7
Electrical

This chapter describes the electrical specification for the USB. It contains signaling, power distribution, and physical layer specifications. This specification does not address regulatory compliance. It is the responsibility of product designers to make sure that their designs comply with all applicable regulatory requirements.

The USB 2.0 specification requires hubs to support high-speed mode. USB 2.0 devices are not required to support high-speed mode. A high-speed capable upstream facing transceiver must not support low-speed signaling mode. A USB 2.0 downstream facing transceiver must support high-speed, full-speed, and low-speed modes.

To assure reliable operation at high-speed data rates, this specification requires the use of cables that conform to all current cable specifications.

In this chapter, there are numerous references to strings of J's and K's, or to strings of 1's and 0's. In each of these instances, the leftmost symbol is transmitted/received first, and the rightmost is transmitted/received last.

7.1 Signaling

The signaling specification for the USB is described in the following subsections.

Overview of High-speed Signaling

A high-speed USB connection is made through a shielded, twisted pair cable that conforms to all current USB cable specifications.

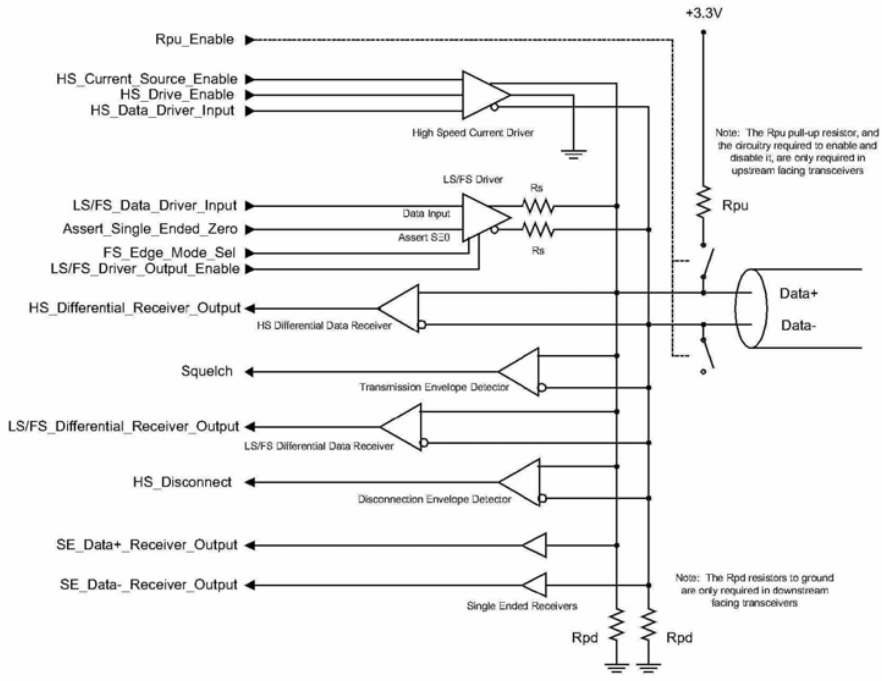

Figure 7-1. Example High-speed Capable Transceiver Circuit

Figure 7-1 depicts an example implementation which largely utilizes USB 1.1 transceiver elements and adds the new elements required for high-speed operation.

High-speed operation supports signaling at 480 Mb/s. To achieve reliable signaling at this rate, the cable is terminated at each end with a resistance from each wire to ground. The value of this resistance (on each wire) is nominally set to 1/2 the specified differential impedance of the cable, or 45 Ω. This presents a differential termination of 90 Ω.

For a link operating in high-speed mode, the high-speed idle state occurs when the transceivers at both ends of the cable present high-speed terminations to ground, and when neither transceiver drives signaling current into the D+ or D- lines. This state is achieved by using the low-/full-speed driver to assert a single ended zero, and to closely control the combined total of the intrinsic driver output impedance and the Rs resistance (to 45 Ω, nominal). The recommended practice is to make the intrinsic driver impedance as low as possible, and to let Rs contribute as much of the 45 Ω as possible. This will generally lead to the best termination accuracy with the least parasitic loading.

In order to transmit in high-speed mode, a transceiver activates an internal current source which is derived from its positive supply voltage and directs this current into one of the two data lines via a high speed current steering switch. In this way, the transceiver generates the high-speed J or K state on the cable.

The dynamic switching of this current into the D+ or D- line follows the same NRZI data encoding scheme used in low-speed or full-speed operation and also in the bit stuffing behavior. To signal a J, the current is directed into the D+ line, and to signal a K, the current is directed into the D- line. The SYNC field and the EOP delimiters have been modified for high-speed mode.

The magnitude of the current source and the value of the termination resistors are controlled to specified tolerances, and together they determine the actual voltage drive levels. The DC resistance from D+ or D- to the device ground is required to be 45 Ω ±10% when measured without a load, and the differential output voltage measured across the lines (in either the J or K state) must be ±400 mV ±10% when D+ and D- are terminated with precision 45 Ω resistors to ground.

The differential voltage developed across the lines is used for three purposes:

- A differential receiver at the receiving end of the cable receives the differential data signal.
- A differential envelope detector at the receiving end of the cable determines when the link is in the Squelch state. A receiver uses squelch detection as indication that the signal at its connector is not valid.
- In the case of a downstream facing hub transceiver, a differential envelope detector monitors whether the signal at its connector is in the high-speed state. A downstream facing transceiver operating in high-speed mode is required to test for this state at a particular point in time when it is transmitting a SOF packet, as described in Section 7.1.7.3. This is used to detect device disconnection. In the absence of the far end terminations, the differential voltage will nominally double (as compared to when a high-speed device is present) when a high-speed J or K are continuously driven for a period exceeding the round-trip delay for the cable and board-traces between the two transceivers.

USB 2.0 requires that a downstream facing transceiver must be able to operate in low-speed, full-speed, and high-speed signaling modes. An upstream facing high-speed capable transceiver must not operate in low-speed signaling mode, but must be able to operate in full-speed signaling mode. Therefore, a 1.5 kΩ pull-up on the D- line is not allowed for a high-speed capable device, since a high-speed capable transceiver must never signal low-speed operation to the hub port to which it is attached.

Table 7-1 describes the required functional elements of a high-speed capable transceiver, using the diagram shown in Figure 7-1 as an example.

Table 7-1. Description of Functional Elements in the Example Shown in Figure 7-1

Element	Description
Low-/full-speed Driver	The low-/full-speed driver is used for low-speed and full-speed transmission. It is required to meet all specifications called out in USB 1.1 for low-speed and full-speed operation, with one exception. The exception is that in high-speed capable transceivers, the impedance of each output, including the contribution of R_S, must be 45 Ω ±10%. The line terminations for high-speed operation are created by having this driver drive D+ and D- to ground. (This is equivalent to driving SE0 in the full-speed or low-speed mode.) Because of the output impedance requirement described above, this provides a well-controlled high-speed termination on each data line to ground. This is equivalent to a 90 Ω differential termination.
Low-/full-speed Differential Receiver	The low-/full-speed differential receiver is used for receiving low-speed and full-speed data.
Single Ended Receivers	The single ended receivers are used for low-speed and full-speed signaling.
High-speed Current Driver	The high-speed current driver is used for high-speed data transmission. A current source derived from a positive supply is switched into either the D+ or D- lines to signal a J or a K, respectively. The nominal value of the current source is 17.78 mA. When this current is applied to a data line with a 45 Ω termination to ground at each end, the nominal high level voltage (V_{HSOH}) is +400 mV. The nominal differential high-speed voltage (D+ - D-) is thus 400 mV for a J and -400 mV for a K. The current source must comply with the Transmit Eye Pattern Templates specified in Section 7.1.2.2, starting with the first symbol of a packet. One means of achieving this is to leave the current source on continuously when a transceiver is operating in high-speed mode. If this approach is used, the current can be directed to the port ground when the transceiver is not transmitting (the example design in Figure 7-1 shows a control line called HS_Current_Source_Enable to turn the current on, and another called HS_Drive_Enable to direct the current into the data lines.) The penalty of this approach is the 17.78 mA of standing current for every such enabled transceiver in the system. The preferred design is to fully turn the current source off when the transceiver is not transmitting.
High-speed Differential Data Receiver	The high-speed differential data receiver is used to receive high-speed data. It is left to transceiver designers to choose between incorporating separate high-speed and low-/full-speed receivers, as shown in Figure 7-1, or combining both functions into a single receiver.

Table 7-1. Description of Functional Elements in the Example Shown in Figure 7-1 (Continued)

Transmission Envelope Detector	This envelope detector is used to indicate that data is invalid when the amplitude of the differential signal at a receiver's inputs falls below the squelch threshold (V_{HSSQ}). It must indicate Squelch when the signal drops below 100 mV differential amplitude, and it must indicate that the line is not in the Squelch state when the signal exceeds 150 mV differential amplitude. The response time of the detector must be fast enough to allow a receiver to detect data transmission, to achieve DLL lock, and to detect the end of the SYNC field within 12 bit times, the minimum number of SYNC bits that a receiver is guaranteed to see. This envelope detector must incorporate a filtering mechanism that prevents indication of squelch during the longest differential data transitions allowed by the receiver eye pattern specifications.
Disconnection Envelope Detector	This envelope detector is required in downstream facing ports to detect the high-speed Disconnect state on the line (V_{HSDSC}). Disconnection must be indicated when the amplitude of the differential signal at the downstream facing driver's connector ≥625 mV, and it must not be indicated when the signal amplitude is ≤525 mV. The output of this detector is sampled at a specific time during the transmission of the high-speed SOF EOP, as described in Section 7.1.7.3.
Pull-up Resistor (R_{PU})	This resistor is required only in upstream facing transceivers and is used to indicate signaling speed capability. A high-speed capable device is required to initially attach as a full-speed device and must transition to high-speed as described in this specification. Once operating in high-speed, the 1.5 kΩ resistor must be electrically removed from the circuit. In Figure 7-1, a control line called R_{PU}_Enable is indicated for this purpose. The preferred embodiment is to attach matched switching devices to both the D+ and D- lines so as to keep the lines' parasitic loading balanced, even though a pull-up resistor must never be used on the D- line of an upstream facing high-speed capable transceiver. When connected, this pull-up must meet all the specifications called out for full-speed operation.
Pull-down Resistors (R_{PD})	These resistors are required only in downstream facing transceivers and must conform to the same specifications called out for low-speed and full-speed operation.

7.1.1 USB Driver Characteristics

The USB uses a differential output driver to drive the USB data signal onto the USB cable.

For low-speed and full-speed operation, the static output swing of the driver in its low state must be below V_{OL} (max) of 0.3 V with a 1.5 kΩ load to 3.6 V, and in its high state must be above the V_{OH} (min) of 2.8 V with a 15 kΩ load to ground as listed in Table 7-7. Full-speed drivers have more stringent requirements, as described in Section 7.1.1.1. The output swings between the differential high and low state must be well-balanced to minimize signal skew. Slew rate control on the driver is required to minimize the radiated noise and cross talk. The driver's outputs must support three-state operation to achieve bi-directional half-duplex operation.

Low-speed and full-speed USB drivers must never "intentionally" generate an SE1 on the bus. SE1 is a state in which both the D+ and D- lines are at a voltage above V_{OSE1} (min), which is 0.8 V.

High-speed drivers use substantially different signaling levels, as described in Section 7.1.1.3.

USB ports must be capable of withstanding continuous exposure to the waveforms shown in Figure 7-2 while in any drive state. These waveforms are applied directly into each USB data pin from a voltage source with an

output impedance of 39 Ω. The open-circuit voltage of the source shown in Figure 7-2 is based on the expected worst-case overshoot and undershoot.

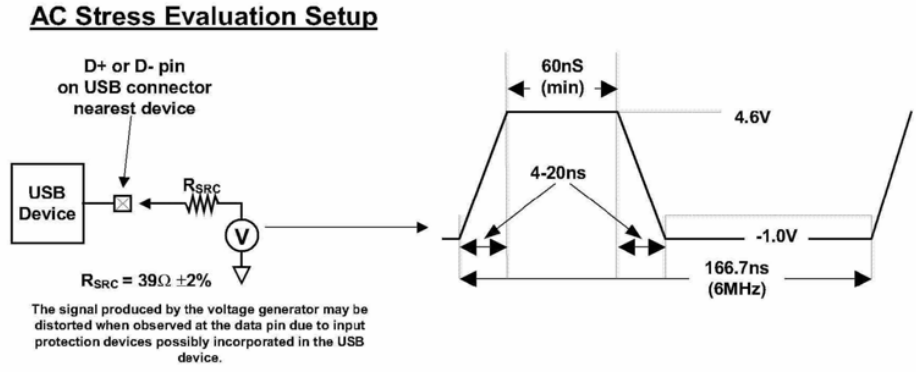

Figure 7-2. Maximum Input Waveforms for USB Signaling

Short Circuit Withstand

A USB transceiver is required to withstand a continuous short circuit of D+ and/or D- to VBUS, GND, other data line, or the cable shield at the connector, for a minimum of 24 hours without degradation. It is recommended that transceivers be designed so as to withstand such short circuits indefinitely. The device must not be damaged under this short circuit condition when transmitting 50% of the time and receiving 50% of the time (in all supported speeds). The transmit phase consists of a symmetrical signal that toggles between drive high and drive low. This requirement must be met for max value of VBUS (5.25 V).

It is recommended that these AC and short circuit stresses be used as qualification criteria against which the long-term reliability of each device is evaluated.

7.1.1.1 Full-speed (12 Mb/s) Driver Characteristics

A full-speed USB connection is made through a shielded, twisted pair cable with a differential characteristic impedance (Z_0) of 90 Ω ±15%, a common mode impedance (Z_{CM}) of 30 Ω ±30%, and a maximum one-way delay (T_{FSCBL}) of 26 ns. When the full-speed driver is not part of a high-speed capable transceiver, the impedance of each of the drivers (Z_{DRV}) must be between 28 Ω and 44 Ω, i.e., within the gray area in Figure 7-4. When the full-speed driver is part of a high-speed capable transceiver, the impedance of each of the drivers (Z_{HSDRV}) must be between 40.5 Ω and 49.5 Ω, i.e., within the gray area in Figure 7-5.

For a CMOS implementation, the driver impedance will typically be realized by a CMOS driver with an impedance significantly less than this resistance with a discrete series resistor making up the balance as shown in Figure 7-3. The series resistor RS is included in the buffer impedance requirement shown in Figure 7-4 and Figure 7-5. In the rest of the chapter, references to the buffer assume a buffer with the series impedance unless stated otherwise.

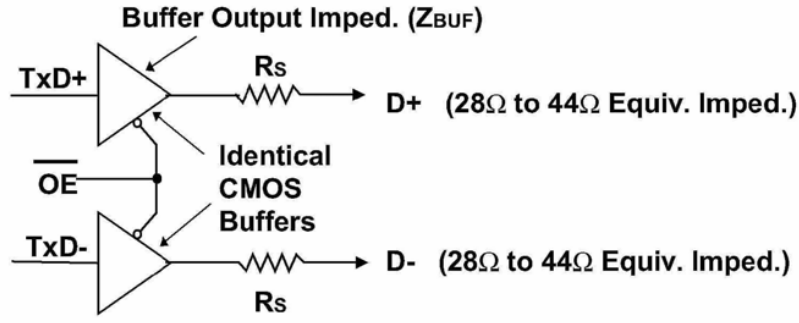

Figure 7-3. Example Full-speed CMOS Driver Circuit (non High-speed capable)

Full-speed Buffers in Transceivers Which are Not High-speed Capable

The buffer impedance must be measured for driving high as well as driving low. Figure 7-4 shows the composite V/I characteristics for the full-speed drivers with included series damping resistor (RS). The characteristics are normalized to the steady-state, unloaded output swing of the driver. The normalized driver characteristics are found by dividing the measured voltages and currents by the actual swing of the driver under test. The normalized V/I curve for the driver must fall entirely inside the shaded region. The V/I region is bounded by the minimum driver impedance above and the maximum driver impedance below. The minimum drive region is intersected by a constant current region of |6.1VOH| mA when driving low and -|6.1VOH| mA when driving high. In the special case of a full-speed driver which is driving low, and which is part of a high-speed capable transceiver, the low drive region is intersected by a constant current region of 22.0 mA. This is the minimum current drive level necessary to ensure that the waveform at the receiver crosses the opposite single-ended switching level on the first reflection.

When testing, the current into or out of the device need not exceed ±10.71*VOH mA and the voltage applied to D+/D- need not exceed 0.3*VOH for the drive low case and need not drop below 0.7*VOH for the drive high case.

Full-speed Buffers in High-speed Capable Transceivers

Figure 7-5 shows the V/I characteristics for a Full-speed buffer which is part of a high-speed capable transceiver. The output impedance, ZHSDRV (including the contribution of RS), is required to be between 40.5 Ω and 49.5 Ω. Additionally, the output voltage must be within 10mV of ground when no current is flowing in or out of the pin (VHSTERM).

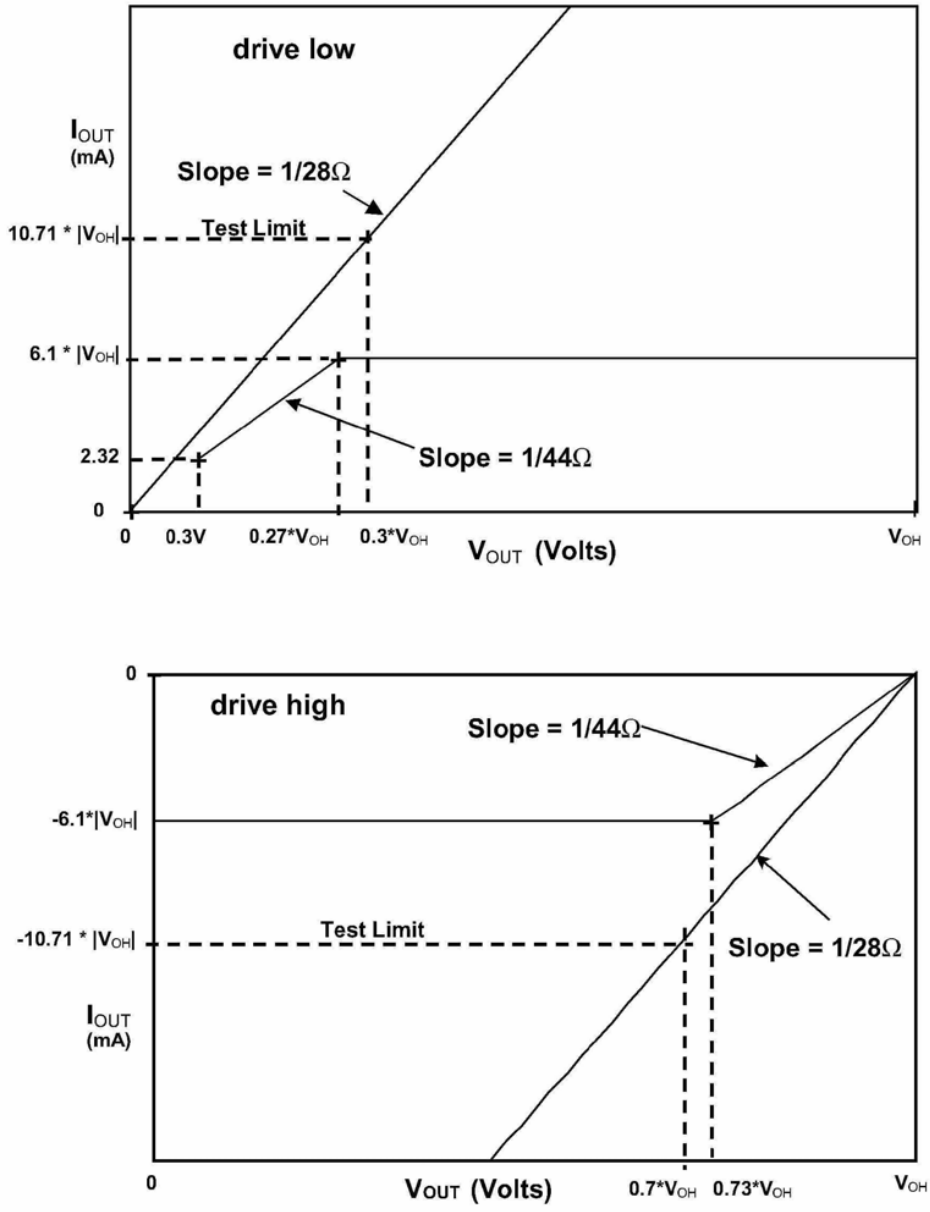

Figure 7-4. Full-speed Buffer V/I Characteristics

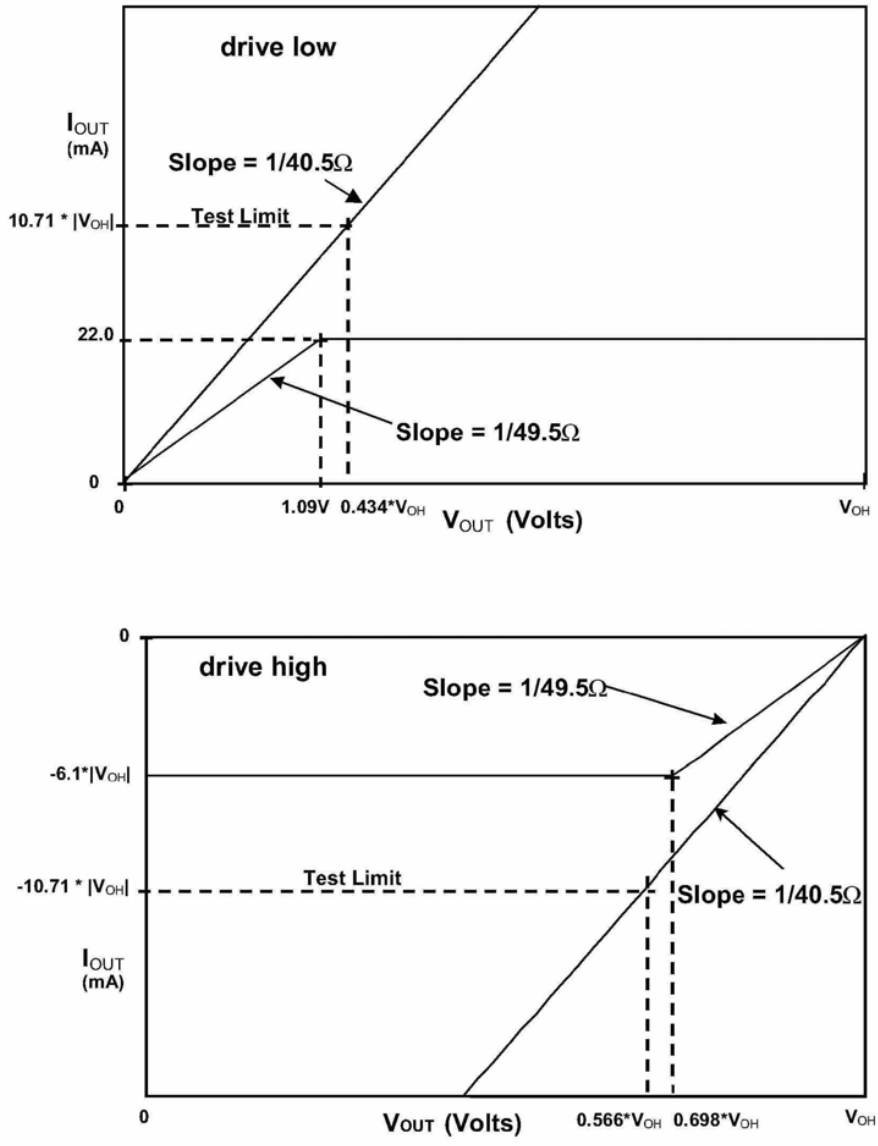

Figure 7-5. Full-speed Buffer V/I Characteristics for High-speed Capable Transceiver