

# **EXHIBIT A**

# **Universal Serial Bus Specification**

**Compaq**

**Intel**

**Microsoft**

**NEC**

**Revision 1.1  
September 23, 1998**

## Universal Serial Bus Specification Revision 1.1

### Scope of this Revision

The 1.1 revision of the specification is intended for product design. Every attempt has been made to ensure a consistent and implementable specification. Implementations should ensure compliance with this revision.

### Revision History

Revision	Issue Date	Comments
0.7	November 11, 1994	Supersedes 0.6e.
0.8	December 30, 1994	Revisions to Chapters 3-8, 10, and 11. Added appendixes.
0.9	April 13, 1995	Revisions to all the chapters.
0.99	August 25, 1995	Revisions to all the chapters.
1.0 FDR	November 13, 1995	Revisions to Chapters 1, 2, 5-11.
1.0	January 15, 1996	Edits to Chapters 5, 6, 7, 8, 9, 10, and 11 for consistency.
1.1	September 23, 1998	Updates to all chapters to fix problems identified.

Universal Serial Bus Specification  
Copyright © 1998, Compaq Computer Corporation,  
Intel Corporation, Microsoft Corporation, NEC Corporation.  
All rights reserved.

### INTELLECTUAL PROPERTY DISCLAIMER

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE. A LICENSE IS HEREBY GRANTED TO REPRODUCE AND DISTRIBUTE THIS SPECIFICATION FOR INTERNAL USE ONLY. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY OTHER INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY.

AUTHORS OF THIS SPECIFICATION DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. AUTHORS OF THIS SPECIFICATION ALSO DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.

GeoPort and Apple Desktop Bus are trademarks of Apple Computer, Inc.  
Windows and Windows NT are trademarks and Microsoft and Win32 are registered trademarks of Microsoft Corporation.  
IBM, PS/2, and Micro Channel are registered trademarks of International Business Machines Corporation.  
AT&T is a registered trademark of American Telephone and Telegraph Company.  
Compaq is a registered trademark of Compaq Computer Corporation.  
UNIX is a registered trademark of UNIX System Laboratories.  
I<sup>2</sup>C is a trademark of Phillips Semiconductors.  
DEC is a trademark of Digital Equipment Corporation.  
All other product names are trademarks, registered trademarks, or servicemarks of their respective owners.

*Please send comments via electronic mail to [techsup@usb.org](mailto:techsup@usb.org)  
For industry information, refer to the USB Implementers Forum web page at <http://www.usb.org>*

# Contents

<b>CHAPTER 1 INTRODUCTION .....</b>	<b>1</b>
1.1 Motivation .....	1
1.2 Objective of the Specification .....	1
1.3 Scope of the Document .....	2
1.4 Document Organization .....	2
<b>CHAPTER 2 TERMS AND ABBREVIATIONS .....</b>	<b>3</b>
<b>CHAPTER 3 BACKGROUND.....</b>	<b>11</b>
3.1 Goals for the Universal Serial Bus .....	11
3.2 Taxonomy of Application Space.....	12
3.3 Feature List.....	12
<b>CHAPTER 4 ARCHITECTURAL OVERVIEW.....</b>	<b>15</b>
4.1 USB System Description .....	15
4.1.1 Bus Topology .....	16
4.2 Physical Interface .....	17
4.2.1 Electrical .....	17
4.2.2 Mechanical .....	17
4.3 Power .....	18
4.3.1 Power Distribution .....	18
4.3.2 Power Management.....	18
4.4 Bus Protocol.....	18
4.5 Robustness.....	19
4.5.1 Error Detection.....	19
4.5.2 Error Handling.....	19
4.6 System Configuration.....	19
4.6.1 Attachment of USB Devices .....	19
4.6.2 Removal of USB Devices.....	20
4.6.3 Bus Enumeration .....	20
4.7 Data Flow Types .....	20
4.7.1 Control Transfers.....	20

## Universal Serial Bus Specification Revision 1.1

4.7.2	Bulk Transfers .....	20
4.7.3	Interrupt Transfers .....	21
4.7.4	Isochronous Transfers .....	21
4.7.5	Allocating USB Bandwidth.....	21
<b>4.8</b>	<b>USB Devices .....</b>	<b>21</b>
4.8.1	Device Characterizations.....	21
4.8.2	Device Descriptions .....	22
<b>4.9</b>	<b>USB Host: Hardware and Software.....</b>	<b>24</b>
<b>4.10</b>	<b>Architectural Extensions .....</b>	<b>24</b>
<b>CHAPTER 5 USB DATA FLOW MODEL.....</b>		<b>25</b>
<b>5.1</b>	<b>Implementer Viewpoints.....</b>	<b>25</b>
<b>5.2</b>	<b>Bus Topology .....</b>	<b>27</b>
5.2.1	USB Host .....	27
5.2.2	USB Devices .....	28
5.2.3	Physical Bus Topology.....	29
5.2.4	Logical Bus Topology.....	30
5.2.5	Client Software-to-function Relationship.....	30
<b>5.3</b>	<b>USB Communication Flow .....</b>	<b>31</b>
5.3.1	Device Endpoints .....	32
5.3.2	Pipes .....	33
<b>5.4</b>	<b>Transfer Types.....</b>	<b>35</b>
<b>5.5</b>	<b>Control Transfers.....</b>	<b>36</b>
5.5.1	Control Transfer Data Format .....	36
5.5.2	Control Transfer Direction.....	37
5.5.3	Control Transfer Packet Size Constraints.....	37
5.5.4	Control Transfer Bus Access Constraints.....	38
5.5.5	Control Transfer Data Sequences.....	40
<b>5.6</b>	<b>Isochronous Transfers .....</b>	<b>41</b>
5.6.1	Isochronous Transfer Data Format.....	41
5.6.2	Isochronous Transfer Direction.....	41
5.6.3	Isochronous Transfer Packet Size Constraints .....	41
5.6.4	Isochronous Transfer Bus Access Constraints .....	42
5.6.5	Isochronous Transfer Data Sequences.....	43
<b>5.7</b>	<b>Interrupt Transfers .....</b>	<b>43</b>
5.7.1	Interrupt Transfer Data Format .....	43
5.7.2	Interrupt Transfer Direction .....	43
5.7.3	Interrupt Transfer Packet Size Constraints.....	43
5.7.4	Interrupt Transfer Bus Access Constraints.....	44
5.7.5	Interrupt Transfer Data Sequences.....	46
<b>5.8</b>	<b>Bulk Transfers .....</b>	<b>46</b>
5.8.1	Bulk Transfer Data Format .....	47
5.8.2	Bulk Transfer Direction .....	47
5.8.3	Bulk Transfer Packet Size Constraints.....	47

## Universal Serial Bus Specification Revision 1.1

5.8.4	Bulk Transfer Bus Access Constraints .....	47
5.8.5	Bulk Transfer Data Sequences .....	48
<b>5.9</b>	<b>Bus Access for Transfers.....</b>	<b>49</b>
5.9.1	Transfer Management.....	49
5.9.2	Transaction Tracking.....	52
5.9.3	Calculating Bus Transaction Times.....	54
5.9.4	Calculating Buffer Sizes in Functions and Software .....	55
5.9.5	Bus Bandwidth Reclamation .....	55
<b>5.10</b>	<b>Special Considerations for Isochronous Transfers.....</b>	<b>55</b>
5.10.1	Example Non-USB Isochronous Application.....	56
5.10.2	USB Clock Model .....	59
5.10.3	Clock Synchronization .....	61
5.10.4	Isochronous Devices.....	61
5.10.5	Data Prebuffering .....	69
5.10.6	SOF Tracking .....	70
5.10.7	Error Handling.....	70
5.10.8	Buffering for Rate Matching .....	71
<b>CHAPTER 6</b>	<b>MECHANICAL.....</b>	<b>73</b>
<b>6.1</b>	<b>Architectural Overview.....</b>	<b>73</b>
<b>6.2</b>	<b>Keyed Connector Protocol.....</b>	<b>73</b>
<b>6.3</b>	<b>Cable.....</b>	<b>74</b>
<b>6.4</b>	<b>Cable Assembly.....</b>	<b>74</b>
6.4.1	Detachable Cable Assemblies .....	74
6.4.2	Full-speed Captive Cable Assemblies .....	76
6.4.3	Low-speed Captive Cable Assemblies .....	78
6.4.4	Prohibited Cable Assemblies.....	80
<b>6.5</b>	<b>Connector Mechanical Configuration and Material Requirements .....</b>	<b>80</b>
6.5.1	USB Icon Location .....	81
6.5.2	USB Connector Termination Data .....	82
6.5.3	Series “A” and Series “B” Receptacles .....	82
6.5.4	Series “A” and Series “B” Plugs .....	86
<b>6.6</b>	<b>Cable Mechanical Configuration and Material Requirements .....</b>	<b>90</b>
6.6.1	Description .....	90
6.6.2	Construction .....	91
6.6.3	Electrical Characteristics .....	93
6.6.4	Cable Environmental Characteristics .....	93
6.6.5	Listing .....	94
<b>6.7</b>	<b>Electrical, Mechanical and Environmental Compliance Standards .....</b>	<b>94</b>
6.7.1	Applicable Documents .....	102
<b>6.8</b>	<b>USB Grounding .....</b>	<b>102</b>
<b>6.9</b>	<b>PCB Reference Drawings.....</b>	<b>102</b>

<b>CHAPTER 7 ELECTRICAL</b> .....	<b>107</b>
<b>7.1 Signaling</b> .....	<b>107</b>
7.1.1 USB Driver Characteristics.....	107
7.1.2 Data Signal Rise and Fall.....	110
7.1.3 Cable Skew.....	112
7.1.4 Receiver Characteristics.....	112
7.1.5 Device Speed Identification.....	113
7.1.6 Input Characteristics.....	114
7.1.7 Signaling Levels.....	115
7.1.8 Data Encoding/Decoding.....	123
7.1.9 Bit Stuffing.....	124
7.1.10 Sync Pattern.....	126
7.1.11 Data Signaling Rate.....	126
7.1.12 Frame Interval and Frame Interval Adjustment.....	126
7.1.13 Data Source Signaling.....	127
7.1.14 Hub Signaling Timings.....	128
7.1.15 Receiver Data Jitter.....	130
7.1.16 Cable Delay.....	132
7.1.17 Cable Attenuation.....	133
7.1.18 Bus Turn-around Time and Inter-packet Delay.....	133
7.1.19 Maximum End-to-end Signal Delay.....	133
<b>7.2 Power Distribution</b> .....	<b>134</b>
7.2.1 Classes of Devices.....	134
7.2.2 Voltage Drop Budget.....	138
7.2.3 Power Control During Suspend/Resume.....	139
7.2.4 Dynamic Attach and Detach.....	140
<b>7.3 Physical Layer</b> .....	<b>141</b>
7.3.1 Regulatory Requirements.....	142
7.3.2 Bus Timing/Electrical Characteristics.....	142
7.3.3 Timing Waveforms.....	151
<b>CHAPTER 8 PROTOCOL LAYER</b> .....	<b>155</b>
<b>8.1 Bit Ordering</b> .....	<b>155</b>
<b>8.2 SYNC Field</b> .....	<b>155</b>
<b>8.3 Packet Field Formats</b> .....	<b>155</b>
8.3.1 Packet Identifier Field.....	155
8.3.2 Address Fields.....	156
8.3.3 Frame Number Field.....	157
8.3.4 Data Field.....	157
8.3.5 Cyclic Redundancy Checks.....	158
<b>8.4 Packet Formats</b> .....	<b>158</b>
8.4.1 Token Packets.....	159
8.4.2 Start-of-Frame Packets.....	159
8.4.3 Data Packets.....	160
8.4.4 Handshake Packets.....	160
8.4.5 Handshake Responses.....	161

<b>8.5</b>	<b>Transaction Formats .....</b>	<b>162</b>
8.5.1	Bulk Transactions.....	163
8.5.2	Control Transfers.....	164
8.5.3	Interrupt Transactions.....	167
8.5.4	Isochronous Transactions .....	168
<b>8.6</b>	<b>Data Toggle Synchronization and Retry .....</b>	<b>168</b>
8.6.1	Initialization via SETUP Token .....	169
8.6.2	Successful Data Transactions .....	169
8.6.3	Data Corrupted or Not Accepted.....	170
8.6.4	Corrupted ACK Handshake.....	170
8.6.5	Low-speed Transactions .....	171
<b>8.7</b>	<b>Error Detection and Recovery.....</b>	<b>172</b>
8.7.1	Packet Error Categories.....	172
8.7.2	Bus Turn-around Timing.....	172
8.7.3	False EOPs .....	173
8.7.4	Babble and Loss of Activity Recovery.....	174
<b>CHAPTER 9</b>	<b>USB DEVICE FRAMEWORK .....</b>	<b>175</b>
<b>9.1</b>	<b>USB Device States.....</b>	<b>175</b>
9.1.1	Visible Device States.....	175
9.1.2	Bus Enumeration .....	179
<b>9.2</b>	<b>Generic USB Device Operations .....</b>	<b>180</b>
9.2.1	Dynamic Attachment and Removal.....	180
9.2.2	Address Assignment.....	180
9.2.3	Configuration .....	180
9.2.4	Data Transfer.....	181
9.2.5	Power Management.....	181
9.2.6	Request Processing.....	181
9.2.7	Request Error.....	182
<b>9.3</b>	<b>USB Device Requests.....</b>	<b>183</b>
9.3.1	bmRequestType.....	183
9.3.2	bRequest.....	184
9.3.3	wValue .....	184
9.3.4	wIndex.....	184
9.3.5	wLength.....	184
<b>9.4</b>	<b>Standard Device Requests.....</b>	<b>185</b>
9.4.1	Clear Feature .....	188
9.4.2	Get Configuration.....	189
9.4.3	Get Descriptor .....	189
9.4.4	Get Interface .....	190
9.4.5	Get Status .....	190
9.4.6	Set Address.....	192
9.4.7	Set Configuration .....	193
9.4.8	Set Descriptor.....	193
9.4.9	Set Feature.....	194
9.4.10	Set Interface.....	195
9.4.11	Synch Frame.....	195
<b>9.5</b>	<b>Descriptors .....</b>	<b>196</b>



## Universal Serial Bus Specification Revision 1.1

<b>9.6 Standard USB Descriptor Definitions.....</b>	<b>196</b>
9.6.1 Device .....	196
9.6.2 Configuration .....	199
9.6.3 Interface .....	201
9.6.4 Endpoint.....	203
9.6.5 String.....	204
<b>9.7 Device Class Definitions .....</b>	<b>205</b>
9.7.1 Descriptors .....	205
9.7.2 Interface(s) and Endpoint Usage .....	205
9.7.3 Requests .....	206
<b>CHAPTER 10 USB HOST: HARDWARE AND SOFTWARE .....</b>	<b>207</b>
<b>10.1 Overview of the USB Host .....</b>	<b>207</b>
10.1.1 Overview.....	207
10.1.2 Control Mechanisms .....	210
10.1.3 Data Flow.....	210
10.1.4 Collecting Status and Activity Statistics .....	211
10.1.5 Electrical Interface Considerations .....	211
<b>10.2 Host Controller Requirements .....</b>	<b>211</b>
10.2.1 State Handling.....	212
10.2.2 Serializer/Deserializer .....	212
10.2.3 Frame Generation.....	212
10.2.4 Data Processing.....	213
10.2.5 Protocol Engine.....	213
10.2.6 Transmission Error Handling .....	213
10.2.7 Remote Wakeup .....	214
10.2.8 Root Hub.....	214
10.2.9 Host System Interface .....	214
<b>10.3 Overview of Software Mechanisms.....</b>	<b>214</b>
10.3.1 Device Configuration .....	215
10.3.2 Resource Management .....	217
10.3.3 Data Transfers .....	217
10.3.4 Common Data Definitions .....	218
<b>10.4 Host Controller Driver.....</b>	<b>218</b>
<b>10.5 Universal Serial Bus Driver .....</b>	<b>219</b>
10.5.1 USBD Overview .....	219
10.5.2 USBD Command Mechanism Requirements.....	221
10.5.3 USBD Pipe Mechanisms.....	223
10.5.4 Managing the USB via the USBD Mechanisms.....	225
10.5.5 Passing USB Preboot Control to the Operating System.....	227
<b>10.6 Operating System Environment Guides.....</b>	<b>227</b>
<b>CHAPTER 11 HUB SPECIFICATION .....</b>	<b>229</b>
<b>11.1 Overview .....</b>	<b>229</b>
11.1.1 Hub Architecture.....	230
11.1.2 Hub Connectivity .....	230

## Universal Serial Bus Specification Revision 1.1

<b>11.2 Hub Frame Timer</b> .....	<b>232</b>
11.2.1 Frame Timer Synchronization.....	233
11.2.2 EOF1 and EOF2 Timing Points .....	234
<b>11.3 Host Behavior at End-of-Frame</b> .....	<b>235</b>
11.3.1 Latest Host Packet.....	235
11.3.2 Packet Nullification.....	235
11.3.3 Transaction Completion Prediction.....	236
<b>11.4 Internal Port</b> .....	<b>237</b>
11.4.1 Inactive .....	237
11.4.2 Suspend Delay.....	237
11.4.3 Full Suspend (Fsus).....	237
11.4.4 Generate Resume (GResume) .....	238
<b>11.5 Downstream Ports</b> .....	<b>239</b>
11.5.1 Downstream Port State Descriptions.....	240
11.5.2 Disconnect Detect Timer.....	243
<b>11.6 Upstream Port</b> .....	<b>244</b>
11.6.1 Receiver.....	244
11.6.2 Transmitter .....	246
<b>11.7 Hub Repeater</b> .....	<b>249</b>
11.7.1 Wait for Start of Packet from Upstream Port (WFSOPFU) .....	250
11.7.2 Wait for End of Packet from Upstream Port (WFEOPFU) .....	250
11.7.3 Wait for Start of Packet (WFSOP) .....	251
11.7.4 Wait for End of Packet (WFEOP).....	251
<b>11.8 Bus State Evaluation</b> .....	<b>251</b>
11.8.1 Port Error.....	251
11.8.2 Speed Detection.....	251
11.8.3 Collision .....	252
11.8.4 Full- versus Low-speed Behavior.....	252
<b>11.9 Suspend and Resume</b> .....	<b>253</b>
<b>11.10 Hub Reset Behavior</b> .....	<b>254</b>
11.10.1 Hub Receiving Reset on Upstream Port.....	254
<b>11.11 Hub Port Power Control</b> .....	<b>255</b>
11.11.1 Multiple Gangs.....	255
<b>11.12 Hub I/O Buffer Requirements</b> .....	<b>256</b>
11.12.1 Pull-up and Pull-down Resistors .....	256
11.12.2 Edge Rate Control .....	256
<b>11.13 Hub Controller</b> .....	<b>256</b>
11.13.1 Endpoint Organization .....	257
11.13.2 Hub Information Architecture and Operation.....	257
11.13.3 Port Change Information Processing.....	259
11.13.4 Hub and Port Status Change Bitmap.....	259
11.13.5 Over-current Reporting and Recovery .....	260
<b>11.14 Hub Configuration</b> .....	<b>261</b>

**Universal Serial Bus Specification Revision 1.1**

<b>11.15 Descriptors .....</b>	<b>263</b>
11.15.1 Standard Descriptors .....	263
11.15.2 Class-specific Descriptors .....	264
<b>11.16 Requests.....</b>	<b>266</b>
11.16.1 Standard Requests .....	266
11.16.2 Class-specific Requests .....	267
<b>INDEX.....</b>	<b>281</b>

# Figures

Figure 3-1. Application Space Taxonomy .....	12
Figure 4-1. Bus Topology .....	16
Figure 4-2. USB Cable.....	17
Figure 4-3. A Typical Hub.....	22
Figure 4-4. Hubs in a Desktop Computer Environment .....	23
Figure 5-1. Simple USB Host/Device View .....	25
Figure 5-2. USB Implementation Areas .....	26
Figure 5-3. Host Composition .....	27
Figure 5-4. Physical Device Composition .....	28
Figure 5-5. USB Physical Bus Topology.....	29
Figure 5-6. USB Logical Bus Topology .....	30
Figure 5-7. Client Software-to-function Relationships.....	30
Figure 5-8. USB Host/Device Detailed View .....	31
Figure 5-9. USB Communication Flow .....	32
Figure 5-10. USB Information Conversion From Client Software to Bus.....	50
Figure 5-11. Transfers for Communication Flows.....	52
Figure 5-12. Arrangement of IRPs to Transactions/Frames .....	53
Figure 5-13. Non-USB Isochronous Example .....	57
Figure 5-14. USB Isochronous Application.....	60
Figure 5-15. Example Source/Sink Connectivity .....	66
Figure 5-16. Data Prebuffering .....	70
Figure 5-17. Packet and Buffer Size Formulas for Rate-Matched Isochronous Transfers.....	72
Figure 6-1. Keyed Connector Protocol.....	73
Figure 6-2. USB Detachable Cable Assembly.....	75
Figure 6-3. USB Full-speed Hardwired Cable Assembly .....	77
Figure 6-4. USB Low-speed Hardwired Cable Assembly .....	79
Figure 6-5. USB Icon.....	81
Figure 6-6. Typical USB Plug Orientation .....	81
Figure 6-7. USB Series "A" Receptacle Interface and Mating Drawing .....	83
Figure 6-8. USB Series "B" Receptacle Interface and Mating Drawing .....	84
Figure 6-9. USB Series "A" Plug Interface Drawing.....	87
Figure 6-10. USB Series "B" Plug Interface Drawing.....	88
Figure 6-11. Typical Full-speed Cable Construction.....	90
Figure 6-12. Single Pin-Type Series "A" Receptacle .....	103

## Universal Serial Bus Specification Revision 1.1

Figure 6-13. Dual Pin-Type Series "A" Receptacle.....	104
Figure 6-14. Single Pin-Type Series "B" Receptacle .....	105
Figure 7-1. Maximum Input Waveforms for USB Signaling .....	107
Figure 7-2. Example Full-speed CMOS Driver Circuit.....	108
Figure 7-3. Full-speed Buffer V/I Characteristics .....	109
Figure 7-4. Full-speed Signal Waveforms.....	110
Figure 7-5. Low-speed Driver Signal Waveforms.....	110
Figure 7-6. Data Signal Rise and Fall Time .....	111
Figure 7-7. Full-speed Load .....	111
Figure 7-8. Low-speed Port Loads .....	112
Figure 7-9. Differential Input Sensitivity Range .....	113
Figure 7-10. Full-speed Device Cable and Resistor Connections.....	113
Figure 7-11. Low-speed Device Cable and Resistor Connections.....	114
Figure 7-12. Placement of Optional Edge Rate Control Capacitors .....	115
Figure 7-13. Upstream Full-speed Port Transceiver.....	117
Figure 7-14. Downstream Port Transceiver.....	117
Figure 7-15. Disconnect Detection .....	118
Figure 7-16. Full-speed Device Connect Detection.....	118
Figure 7-17. Low-speed Device Connect Detection.....	119
Figure 7-18. Bus State Evaluation after reset (optional).....	119
Figure 7-19. Power-on and Connection Events Timing .....	120
Figure 7-20. Packet Voltage Levels.....	121
Figure 7-21. NRZI Data Encoding .....	124
Figure 7-22. Bit Stuffing .....	124
Figure 7-23. Illustration of Extra Bit Preceding EOP.....	125
Figure 7-24. Flow Diagram for Bit Stuffing.....	125
Figure 7-25. Sync Pattern .....	126
Figure 7-26. Data Jitter Taxonomy.....	127
Figure 7-27. SE0 for EOP Width Timing .....	128
Figure 7-28. Hub Propagation Delay of Full-speed Differential Signals.....	129
Figure 7-29. Full-speed Cable Delay.....	132
Figure 7-30. Low-speed Cable Delay .....	132
Figure 7-31. Worst-case End to End Signal Delay Model.....	134
Figure 7-32. Compound Bus-powered Hub.....	135
Figure 7-33. Compound Self-powered Hub .....	136
Figure 7-34. Low-power Bus-powered Function.....	137
Figure 7-35. High-power Bus-powered Function.....	137
Figure 7-36. Self-powered Function.....	138

## Universal Serial Bus Specification Revision 1.1

Figure 7-37. Worst-case Voltage Drop Topology (Steady State) .....	138
Figure 7-38. Typical Suspend Current Averaging Profile .....	139
Figure 7-39. Differential Data Jitter.....	151
Figure 7-40. Differential-to-EOP Transition Skew and EOP Width .....	151
Figure 7-41. Receiver Jitter Tolerance.....	151
Figure 7-42. Hub Differential Delay, Differential Jitter, and SOP Distortion .....	152
Figure 7-43. Hub EOP Delay and EOP Skew.....	153
Figure 8-1. PID Format.....	155
Figure 8-2. ADDR Field .....	157
Figure 8-3. Endpoint Field.....	157
Figure 8-4. Data Field Format .....	157
Figure 8-5. Token Format.....	159
Figure 8-6. SOF Packet.....	159
Figure 8-7. Data Packet Format.....	160
Figure 8-8. Handshake Packet .....	160
Figure 8-9. Bulk Transaction Format.....	163
Figure 8-10. Bulk Reads and Writes.....	164
Figure 8-11. Control SETUP Transaction .....	164
Figure 8-12. Control Read and Write Sequences.....	165
Figure 8-13. Interrupt Transaction Format .....	167
Figure 8-14. Isochronous Transaction Format.....	168
Figure 8-15. SETUP Initialization .....	169
Figure 8-16. Consecutive Transactions.....	169
Figure 8-17. NAKed Transaction with Retry.....	170
Figure 8-18. Corrupted ACK Handshake with Retry .....	170
Figure 8-19. Low-speed Transaction .....	171
Figure 8-20. Bus Turn-around Timer Usage.....	173
Figure 9-1. Device State Diagram .....	176
Figure 9-2. wIndex Format when Specifying an Endpoint .....	184
Figure 9-3. wIndex Format when Specifying an Interface .....	184
Figure 9-4. Information Returned by a GetStatus() Request to a Device .....	191
Figure 9-5. Information Returned by a GetStatus() Request to a Interface .....	191
Figure 9-6. Information Returned by a GetStatus() Request to an Endpoint.....	192
Figure 10-1. Interlayer Communications Model.....	207
Figure 10-2. Host Communications.....	208
Figure 10-3. Frame Creation.....	212
Figure 10-4. Configuration Interactions .....	215
Figure 10-5. Universal Serial Bus Driver Structure.....	219

## Universal Serial Bus Specification Revision 1.1

Figure 11-1. Hub Architecture.....	230
Figure 11-2. Hub Signaling Connectivity.....	231
Figure 11-3. Resume Connectivity.....	232
Figure 11-4. EOF Timing Points.....	234
Figure 11-5. Internal Port State Machine.....	237
Figure 11-6. Downstream Hub Port State Machine.....	239
Figure 11-7. Upstream Port Receiver State Machine.....	244
Figure 11-8. Upstream Hub Port Transmitter State Machine.....	246
Figure 11-9. Hub Repeater State Machine.....	249
Figure 11-10. Example Remote-Wakeup Resume Signaling.....	254
Figure 11-11. Example Hub Controller Organization.....	257
Figure 11-12. Relationship of Status, Status Change, and Control Information to Device States.....	258
Figure 11-13. Port Status Handling Method.....	259
Figure 11-14. Hub and Port Status Change Bitmap.....	260
Figure 11-15. Example Hub and Port Change Bit Sampling.....	260

## Tables

Table 5-1. Full-speed Control Transfer Limits .....	39
Table 5-2. Low-speed Control Transfer Limits .....	40
Table 5-3. Isochronous Transaction Limits .....	42
Table 5-4. Full-speed Interrupt Transaction Limits .....	45
Table 5-5. Low-speed Interrupt Transaction Limits .....	46
Table 5-6. Bulk Transaction Limits .....	48
Table 5-7. Synchronization Characteristics .....	62
Table 5-8. Connection Requirements .....	68
Table 6-1. USB Connector Termination Assignment .....	82
Table 6-2. Power Pair .....	91
Table 6-3. Signal Pair .....	91
Table 6-4. Drain Wire Signal Pair .....	92
Table 6-5. Nominal Cable Diameter .....	93
Table 6-6. Conductor Resistance .....	93
Table 6-7. USB Electrical, Mechanical and Environmental Compliance Standards .....	94
Table 7-1. Signaling Levels .....	116
Table 7-2. Full-speed Jitter Budget .....	131
Table 7-3. Low-speed Jitter Budget .....	131
Table 7-4. Signal Attenuation .....	133
Table 7-5. DC Electrical Characteristics .....	142
Table 7-6. Full-speed Source Electrical Characteristics .....	144
Table 7-7. Low-speed Source Electrical Characteristics .....	145
Table 7-8. Hub/Repeater Electrical Characteristics .....	146
Table 7-9. Cable Characteristics (Note 14) .....	147
Table 7-10. Hub Event Timings .....	148
Table 7-11. Device Event Timings .....	149
Table 8-1. PID Types .....	156
Table 8-2. Function Responses to IN Transactions .....	161
Table 8-3. Host Responses to IN Transactions .....	162
Table 8-4. Function Responses to OUT Transactions in Order of Precedence .....	162
Table 8-5. Status Stage Responses .....	166
Table 8-6. Packet Error Types .....	172
Table 9-1. Visible Device States .....	177
Table 9-2. Format of Setup Data .....	183



## Universal Serial Bus Specification Revision 1.1

Table 9-3. Standard Device Requests .....	186
Table 9-4. Standard Request Codes .....	187
Table 9-5. Descriptor Types .....	187
Table 9-6. Standard Feature Selectors .....	188
Table 9-7. Standard Device Descriptor.....	197
Table 9-8. Standard Configuration Descriptor .....	199
Table 9-9. Standard Interface Descriptor.....	202
Table 9-10. Standard Endpoint Descriptor .....	203
Table 9-11. Codes Representing Languages Supported by the Device .....	205
Table 9-12. UNICODE String Descriptor .....	205
Table 11-1. Hub and Host EOF Timing Points.....	235
Table 11-2. Internal Port Signal/Event Definitions .....	237
Table 11-3. Downstream Hub Port Signal/Event Definitions.....	240
Table 11-4. Upstream Hub Port Receiver Signal/Event Definitions .....	245
Table 11-5. Upstream Hub Port Transmit Signal/Event Definitions .....	247
Table 11-6. Hub Repeater Signal/Event Definitions .....	250
Table 11-7. Hub Power Operating Mode Summary .....	261
Table 11-8. Hub Descriptor .....	264
Table 11-9. Hub Responses to Standard Device Requests .....	266
Table 11-10. Hub Class Requests .....	267
Table 11-11. Hub Class Request Codes.....	268
Table 11-12. Hub Class Feature Selectors.....	268
Table 11-13. Hub Status Field, <i>wHubStatus</i> .....	272
Table 11-14. Hub Change Field, <i>wHubChange</i> .....	272
Table 11-15. Port Status Field, <i>wPortStatus</i> .....	274
Table 11-16. Port Change Field, <i>wPortChange</i> .....	277

# Chapter 1

## Introduction

### 1.1 Motivation

The motivation for the Universal Serial Bus (USB) comes from three interrelated considerations:

- **Connection of the PC to the telephone**  
It is well understood that the merge of computing and communication will be the basis for the next generation of productivity applications. The movement of machine-oriented and human-oriented data types from one location or environment to another depends on ubiquitous and cheap connectivity. Unfortunately, the computing and communication industries have evolved independently. The USB provides a ubiquitous link that can be used across a wide range of PC-to-telephone interconnects.
- **Ease-of-use**  
The lack of flexibility in reconfiguring the PC has been acknowledged as the Achilles' heel to its further deployment. The combination of user-friendly graphical interfaces and the hardware and software mechanisms associated with new-generation bus architectures have made computers less confrontational and easier to reconfigure. However, from the end user's point of view, the PC's I/O interfaces, such as serial/parallel ports, keyboard/mouse/joystick interfaces, etc., do not have the attributes of plug-and-play.
- **Port expansion**  
The addition of external peripherals continues to be constrained by port availability. The lack of a bi-directional, low-cost, low-to-mid speed peripheral bus has held back the creative proliferation of peripherals such as telephone/fax/modem adapters, answering machines, scanners, PDA's, keyboards, mice, etc. Existing interconnects are optimized for one or two point products. As each new function or capability is added to the PC, a new interface has been defined to address this need.

The USB is the answer to connectivity for the PC architecture. It is a fast, bi-directional, isochronous, low-cost, dynamically attachable serial interface that is consistent with the requirements of the PC platform of today and tomorrow.

### 1.2 Objective of the Specification

This document defines an industry-standard USB. The specification describes the bus attributes, the protocol definition, types of transactions, bus management, and the programming interface required to design and build systems and peripherals that are compliant with this standard.

The goal is to enable such devices from different vendors to interoperate in an open architecture. The specification is intended as an enhancement to the PC architecture, spanning portable, business desktop, and home environments. It is intended that the specification allow system OEMs and peripheral developers adequate room for product versatility and market differentiation without the burden of carrying obsolete interfaces or losing compatibility.

### 1.3 Scope of the Document

- **Target audience**  
The specification is primarily targeted to peripheral developers and system OEMs, but provides valuable information for platform operating system/ BIOS/ device driver, adapter IHVs/ISVs, and platform/adaptor controller vendors.
- **Benefit**  
This version of the USB Specification can be used for planning new products, engineering an early prototype, and preliminary software development. All final products are required to be compliant with the USB Specification 1.1.

### 1.4 Document Organization

Chapters 1 through 5 provide an overview for all readers, while Chapters 6 through 11 contain detailed technical information defining the USB.

- Peripheral implementers should particularly read Chapters 5 through 11.
- USB Host Controller implementers should particularly read Chapters 5 through 8, 10, and 11.
- USB device driver implementers should particularly read Chapters 5, 9, and 10.

This document is complemented and referenced by the *Universal Serial Bus Device Class Specifications*. Device class specifications exist for a wide variety of devices. Please contact the USB Implementers Forum for further details.

Readers are also requested to contact operating system vendors for operating system bindings specific to the USB.

## Chapter 2

# Terms and Abbreviations

This chapter lists and defines terms and abbreviations used throughout this specification.

<b>ACK</b>	Handshake packet indicating a positive acknowledgment.
<b>Active Device</b>	A device that is powered and is not in the Suspend state.
<b>Asynchronous Data</b>	Data transferred at irregular intervals with relaxed latency requirements.
<b>Asynchronous RA</b>	The incoming data rate, $F_{s_i}$ , and the outgoing data rate, $F_{s_o}$ , of the RA process are independent (i.e., there is no shared master clock). See also Rate Adaptation.
<b>Asynchronous SRC</b>	The incoming sample rate, $F_{s_i}$ , and outgoing sample rate, $F_{s_o}$ , of the SRC process are independent (i.e., there is no shared master clock). See also Sample Rate Conversion.
<b>Audio Device</b>	A device that sources or sinks sampled analog data.
<b>AWG#</b>	The measurement of a wire's cross section, as defined by the American Wire Gauge standard.
<b>Babble</b>	Unexpected bus activity that persists beyond a specified point in a frame.
<b>Bandwidth</b>	The amount of data transmitted per unit of time, typically bits per second (b/s) or bytes per second (B/s).
<b>Big Endian</b>	A method of storing data that places the most significant byte of multiple-byte values at a lower storage addresses. For example, a 16-bit integer stored in big endian format places the least significant byte at the higher address and the most significant byte at the lower address. See also Little Endian.
<b>Bit</b>	A unit of information used by digital computers. Represents the smallest piece of addressable memory within a computer. A bit expresses the choice between two possibilities and is typically represented by a logical one (1) or zero (0).
<b>Bit Stuffing</b>	Insertion of a "0" bit into a data stream to cause an electrical transition on the data wires, allowing a PLL to remain locked.
<b>b/s</b>	Transmission rate expressed in bits per second.
<b>B/s</b>	Transmission rate expressed in bytes per second.
<b>Buffer</b>	Storage used to compensate for a difference in data rates or time of occurrence of events, when transmitting data from one device to another.
<b>Bulk Transfer</b>	One of the four USB transfer types. Bulk transfers are non-periodic, large bursty communication typically used for a transfer that can use any available bandwidth and can also be delayed until bandwidth is available. See also Transfer Type.
<b>Bus Enumeration</b>	Detecting and identifying USB devices.

## Universal Serial Bus Specification Revision 1.1

<b>Byte</b>	A data element that is eight bits in size.
<b>Capabilities</b>	Those attributes of a USB device that are administrated by the host.
<b>Characteristics</b>	Those qualities of a USB device that are unchangeable; for example, the device class is a device characteristic.
<b>Client</b>	Software resident on the host that interacts with the USB System Software to arrange data transfer between a function and the host. The client is often the data provider and consumer for transferred data.
<b>Configuring Software</b>	Software resident on the host software that is responsible for configuring a USB device. This may be a system configurator or software specific to the device.
<b>Control Endpoint</b>	A pair of device endpoints with the same endpoint number that are used by a control pipe. Control endpoints transfer data in both directions and therefore use both endpoint directions of a device address and endpoint number combination. Thus, each control endpoint consumes two endpoint addresses.
<b>Control Pipe</b>	Same as a message pipe.
<b>Control Transfer</b>	One of the four USB transfer types. Control transfers support configuration/command/status type communications between client and function. See also Transfer Type.
<b>CRC</b>	See Cyclic Redundancy Check.
<b>CTI</b>	Computer Telephony Integration.
<b>Cyclic Redundancy Check (CRC)</b>	A check performed on data to see if an error has occurred in transmitting, reading, or writing the data. The result of a CRC is typically stored or transmitted with the checked data. The stored or transmitted result is compared to a CRC calculated for the data to determine if an error has occurred.
<b>Default Address</b>	An address defined by the USB Specification and used by a USB device when it is first powered or reset. The default address is 00H.
<b>Default Pipe</b>	The message pipe created by the USB System Software to pass control and status information between the host and a USB device's endpoint zero.
<b>Device</b>	<p>A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, device may refer to a single hardware component, as in a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a USB interface device. At an even higher level, device may refer to the function performed by an entity attached to the USB; for example, a data/FAX modem device. Devices may be physical, electrical, addressable, and logical.</p> <p>When used as a non-specific reference, a USB device is either a hub or a function.</p>
<b>Device Address</b>	A seven-bit value representing the address of a device on the USB. The device address is the default address (00H) when the USB device is first powered or the device is reset. Devices are assigned a unique device address by the USB System Software.

## Universal Serial Bus Specification Revision 1.1

<b>Device Endpoint</b>	A uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device. See also Endpoint Address.
<b>Device Resources</b>	Resources provided by UB devices, such as buffer space and endpoints. See also Host Resources and Universal Serial Bus Resources.
<b>Device Software</b>	Software that is responsible for using a USB device. This software may or may not also be responsible for configuring the device for use.
<b>Downstream</b>	The direction of data flow from the host or away from the host. A downstream port is the port on a hub electrically farthest from the host that generates downstream data traffic from the hub. Downstream ports receive upstream data traffic.
<b>Driver</b>	When referring to hardware, an I/O pad that drives an external load. When referring to software, a program responsible for interfacing to a hardware device; that is, a device driver.
<b>DWORD</b>	Double word. A data element that is two words (i.e., four bytes or 32 bits) in size.
<b>Dynamic Insertion and Removal</b>	The ability to attach and remove devices while the host is in operation.
<b>E<sup>2</sup>PROM</b>	See Electrically Erasable Programmable Read Only Memory.
<b>EEPROM</b>	See Electrically Erasable Programmable Read Only Memory.
<b>Electrically Erasable Programmable Read Only Memory (EEPROM)</b>	Non-volatile rewritable memory storage technology.
<b>End User</b>	The user of a host.
<b>Endpoint</b>	See Device Endpoint.
<b>Endpoint Address</b>	The combination of an endpoint number and an endpoint direction on a USB device. Each endpoint address supports data transfer in one direction.
<b>Endpoint Direction</b>	The direction of data transfer on the USB. The direction can be either IN or OUT. IN refers to transfers to the host; OUT refers to transfers from the host.
<b>Endpoint Number</b>	A four-bit value between 0H and FH, inclusive, associated with an endpoint on a USB device.
<b>EOF</b>	End-of-Frame.
<b>EOP</b>	End-of-Packet.
<b>External Port</b>	See Port.
<b>False EOP</b>	A spurious, usually noise-induced event that is interpreted by a packet receiver as an EOP.
<b>Frame</b>	The time from the start of one SOF token to the start of the subsequent SOF token; consists of a series of transactions.

## Universal Serial Bus Specification Revision 1.1

<b>Frame Pattern</b>	A sequence of frames that exhibit a repeating pattern in the number of samples transmitted per frame. For a 44.1kHz audio transfer, the frame pattern could be nine frames containing 44 samples followed by one frame containing 45 samples.
<b>F<sub>s</sub></b>	See Sample Rate.
<b>Full-duplex</b>	Computer data transmission occurring in both directions simultaneously.
<b>Function</b>	A USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers.
<b>Handshake Packet</b>	A packet that acknowledges or rejects a specific condition. For examples, see ACK and NAK.
<b>Host</b>	The host computer system where the USB Host Controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.
<b>Host Controller</b>	The host's USB interface.
<b>Host Controller Driver (HCD)</b>	The USB software layer that abstracts the Host Controller hardware. The Host Controller Driver provides an SPI for interaction with a Host Controller. The Host Controller Driver hides the specifics of the Host Controller hardware implementation.
<b>Host Resources</b>	Resources provided by the host, such as buffer space and interrupts. See also Device Resources and Universal Serial Bus Resources.
<b>Hub</b>	A USB device that provides additional connections to the USB.
<b>Hub Tier</b>	The level of connect within a USB network topology, given as the number of hubs through which the data has to flow.
<b>Interrupt Request (IRQ)</b>	A hardware signal that allows a device to request attention from a host. The host typically invokes an interrupt service routine to handle the condition that caused the request.
<b>Interrupt Transfer</b>	One of the four USB transfer types. Interrupt transfer characteristics are small data, non-periodic, low-frequency, and bounded-latency. Interrupt transfers are typically used to handle service needs. See also Transfer Type.
<b>I/O Request Packet</b>	An identifiable request by a software client to move data between itself (on the host) and an endpoint of a device in an appropriate direction.
<b>IRP</b>	See I/O Request Packet.
<b>IRQ</b>	See Interrupt Request.
<b>Isochronous Data</b>	A stream of data whose timing is implied by its delivery rate.
<b>Isochronous Device</b>	An entity with isochronous endpoints, as defined in the USB Specification, that sources or sinks sampled analog streams or synchronous data streams.
<b>Isochronous Sink Endpoint</b>	An endpoint that is capable of consuming an isochronous data stream that is sent by the host.
<b>Isochronous Source Endpoint</b>	An endpoint that is capable of producing an isochronous data stream and sending it to the host.
<b>Isochronous Transfer</b>	One of the four USB transfer types. Isochronous transfers are used when working with isochronous data. Isochronous transfers provide periodic, continuous communication between host and device. See also Transfer Type.

## Universal Serial Bus Specification Revision 1.1

<b>Jitter</b>	A tendency toward lack of synchronization caused by mechanical or electrical changes. More specifically, the phase shift of digital pulses over a transmission medium.
<b>kb/s</b>	Transmission rate expressed in kilobits per second.
<b>kB/s</b>	Transmission rate expressed in kilobytes per second.
<b>Little Endian</b>	Method of storing data that places the least significant byte of multiple-byte values at lower storage addresses. For example, a 16-bit integer stored in little endian format places the least significant byte at the lower address and the most significant byte at the next address. See also Big Endian.
<b>LOA</b>	Loss of bus activity characterized by an SOP without a corresponding EOP.
<b>LSb</b>	Least significant bit.
<b>LSB</b>	Least significant byte.
<b>Mb/s</b>	Transmission rate expressed in megabits per second.
<b>MB/s</b>	Transmission rate expressed in megabytes per second.
<b>Message Pipe</b>	A bi-directional pipe that transfers data using a request/data/status paradigm. The data has an imposed structure that allows requests to be reliably identified and communicated.
<b>MSb</b>	Most significant bit.
<b>MSB</b>	Most significant byte.
<b>NAK</b>	Handshake packet indicating a negative acknowledgment.
<b>Non Return to Zero Invert (NRZI)</b>	A method of encoding serial data in which ones and zeroes are represented by opposite and alternating high and low voltages where there is no return to zero (reference) voltage between encoded bits. Eliminates the need for clock pulses.
<b>NRZI</b>	See Non Return to Zero Invert.
<b>Object</b>	Host software or data structure representing a USB entity.
<b>Packet</b>	A bundle of data organized in a group for transmission. Packets typically contain three elements: control information (e.g., source, destination, and length), the data to be transferred, and error detection and correction bits.
<b>Packet Buffer</b>	The logical buffer used by a USB device for sending or receiving a single packet. This determines the maximum packet size the device can send or receive.
<b>Packet ID (PID)</b>	A field in a USB packet that indicates the type of packet, and by inference, the format of the packet and the type of error detection applied to the packet.
<b>Phase</b>	A token, data, or handshake packet; a transaction has three phases.
<b>Phase Locked Loop (PLL)</b>	A circuit that acts as a phase detector to keep an oscillator in phase with an incoming frequency.
<b>Physical Device</b>	A device that has a physical implementation; e.g., speakers, microphones, and CD players.
<b>PID</b>	See Packet ID.



## Universal Serial Bus Specification Revision 1.1

<b>Pipe</b>	A logical abstraction representing the association between an endpoint on a device and software on the host. A pipe has several attributes; for example, a pipe may transfer data as streams (stream pipe) or messages (message pipe). See also Stream Pipe and Message Pipe.
<b>PLL</b>	See Phase Locked Loop.
<b>Polling</b>	Asking multiple devices, one at a time, if they have any data to transmit.
<b>POR</b>	See Power On Reset.
<b>Port</b>	Point of access to or from a system or circuit. For the USB, the point where a USB device is attached.
<b>Power On Reset (POR)</b>	Restoring a storage device, register, or memory to a predetermined state when power is applied.
<b>Programmable Data Rate</b>	Either a fixed data rate (single-frequency endpoints), a limited number of data rates (32kHz, 44.1kHz, 48kHz, ...), or a continuously programmable data rate. The exact programming capabilities of an endpoint must be reported in the appropriate class-specific endpoint descriptors.
<b>Protocol</b>	A specific set of rules, procedures, or conventions relating to format and timing of data transmission between two devices.
<b>RA</b>	See Rate Adaptation.
<b>Rate Adaptation</b>	The process by which an incoming data stream, sampled at $F_{s_i}$ , is converted to an outgoing data stream, sampled at $F_{s_o}$ , with a certain loss of quality, determined by the rate adaptation algorithm. Error control mechanisms are required for the process. $F_{s_i}$ and $F_{s_o}$ can be different and asynchronous. $F_{s_i}$ is the input data rate of the RA; $F_{s_o}$ is the output data rate of the RA.
<b>Request</b>	A request made to a USB device contained within the data portion of a SETUP packet.
<b>Retire</b>	The action of completing service for a transfer and notifying the appropriate software client of the completion.
<b>Root Hub</b>	A USB hub directly attached to the Host Controller. This hub is attached to the host (tier 0).
<b>Root Port</b>	The downstream port on a Root Hub.
<b>Sample</b>	The smallest unit of data on which an endpoint operates; a property of an endpoint.
<b>Sample Rate (<math>F_s</math>)</b>	The number of samples per second, expressed in Hertz (Hz).
<b>Sample Rate Conversion (SRC)</b>	A dedicated implementation of the RA process for use on sampled analog data streams. The error control mechanism is replaced by interpolating techniques.
<b>Service</b>	A procedure provided by a System Programming Interface (SPI).
<b>Service Interval</b>	The period between consecutive requests to a USB endpoint to send or receive data.
<b>Service Jitter</b>	The deviation of service delivery from its scheduled delivery time.
<b>Service Rate</b>	The number of services to a given endpoint per unit time.
<b>SOF</b>	See Start-of-Frame.
<b>SOP</b>	Start-of-Packet.

## Universal Serial Bus Specification Revision 1.1

<b>SPI</b>	See System Programming Interface.
<b>SRC</b>	See Sample Rate Conversion.
<b>Stage</b>	One part of the sequence composing a control transfer; stages include the Setup stage, the Data stage, and the Status stage.
<b>Start-of-Frame (SOF)</b>	The first transaction in each frame. An SOF allows endpoints to identify the start of the frame and synchronize internal endpoint clocks to the host.
<b>Stream Pipe</b>	A pipe that transfers data as a stream of samples with no defined USB structure.
<b>Synchronization Type</b>	A classification that characterizes an isochronous endpoint's capability to connect to other isochronous endpoints.
<b>Synchronous RA</b>	The incoming data rate, $F_{s_i}$ , and the outgoing data rate, $F_{s_o}$ , of the RA process are derived from the same master clock. There is a fixed relation between $F_{s_i}$ and $F_{s_o}$ .
<b>Synchronous SRC</b>	The incoming sample rate, $F_{s_i}$ , and outgoing sample rate, $F_{s_o}$ , of the SRC process are derived from the same master clock. There is a fixed relation between $F_{s_i}$ and $F_{s_o}$ .
<b>System Programming Interface (SPI)</b>	A defined interface to services provided by system software.
<b>TDM</b>	See Time Division Multiplexing.
<b>Termination</b>	Passive components attached at the end of cables to prevent signals from being reflected or echoed.
<b>Time Division Multiplexing (TDM)</b>	A method of transmitting multiple signals (data, voice, and/or video) simultaneously over one communications medium by interleaving a piece of each signal one after another.
<b>Timeout</b>	The detection of a lack of bus activity for some predetermined interval.
<b>Token Packet</b>	A type of packet that identifies what transaction is to be performed on the bus.
<b>Transaction</b>	The delivery of service to an endpoint; consists of a token packet, optional data packet, and optional handshake packet. Specific packets are allowed/required based on the transaction type.
<b>Transfer</b>	One or more bus transactions to move information between a software client and its function.
<b>Transfer Type</b>	Determines the characteristics of the data flow between a software client and its function. Four transfer types are defined: control, interrupt, bulk, and isochronous.
<b>Turn-around Time</b>	The time a device needs to wait to begin transmitting a packet after a packet has been received to prevent collisions on the USB. This time is based on the length and propagation delay characteristics of the cable and the location of the transmitting device in relation to other devices on the USB.
<b>USB D</b>	See Universal Serial Bus Driver.
<b>Universal Serial Bus Driver (USB D)</b>	The host resident software entity responsible for providing common services to clients that are manipulating one or more functions on one or more Host Controllers.

## Universal Serial Bus Specification Revision 1.1

<b>Universal Serial Bus Resources</b>	Resources provided by the USB, such as bandwidth and power. See also Device Resources and Host Resources
<b>Upstream</b>	The direction of data flow towards the host. An upstream port is the port on a device electrically closest to the host that generates upstream data traffic from the hub. Upstream ports receive downstream data traffic.
<b>Virtual Device</b>	A device that is represented by a software interface layer. An example of a virtual device is a hard disk with its associated device driver and client software that makes it able to reproduce an audio .WAV file.
<b>Word</b>	A data element that is two bytes (16 bits) in size.

# Chapter 3

## Background

This chapter presents a brief description of the background of the Universal Serial Bus (USB), including design goals, features of the bus, and existing technologies.

### 3.1 Goals for the Universal Serial Bus

The USB is specified to be an industry-standard extension to the PC architecture with a focus on Computer Telephony Integration (CTI), consumer, and productivity applications. The following criteria were applied in defining the architecture for the USB:

- Ease-of-use for PC peripheral expansion
- Low-cost solution that supports transfer rates up to 12Mb/s
- Full support for real-time data for voice, audio, and compressed video
- Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging
- Integration in commodity device technology
- Comprehension of various PC configurations and form factors
- Provision of a standard interface capable of quick diffusion into product
- Enablement of new classes of devices that augment the PC's capability.

### 3.2 Taxonomy of Application Space

Figure 3-1 describes a taxonomy for the range of data traffic workloads that can be serviced over a USB. As can be seen, a 12Mb/s bus comprehends the mid-speed and low-speed data ranges. Typically, mid-speed data types are isochronous, while low-speed data comes from interactive devices. The USB being proposed is primarily a desktop bus but can be readily applied to the mobile environment. The software architecture allows for future extension of the USB by providing support for multiple USB Host Controllers.

<u>PERFORMANCE</u>	<u>APPLICATIONS</u>	<u>ATTRIBUTES</u>
<b>LOW-SPEED</b> <ul style="list-style-type: none"> <li>• Interactive Devices</li> <li>• 10 – 100kb/s</li> </ul>	Keyboard, Mouse Stylus Game Peripherals Virtual Reality Peripherals Monitor Configuration	Lower Cost Hot Plug-unplug Ease-of-use Multiple Peripherals
<b>MEDIUM-SPEED</b> <ul style="list-style-type: none"> <li>• Phone, Audio, Compressed Video</li> <li>500Kb/S - 10Mb/s</li> </ul>	ISDN PBX POTS Audio	Low Cost Ease-of-use Guaranteed Latency Guaranteed Bandwidth Dynamic Attach-Detach Multiple devices
<b>HIGH-SPEED</b> <ul style="list-style-type: none"> <li>• Video, Disk</li> <li>• 25 - 500Mb/s</li> </ul>	Video Disk	High Bandwidth Guaranteed Latency Ease-of-use

Figure 3-1. Application Space Taxonomy

### 3.3 Feature List

The USB Specification provides a selection of attributes that can achieve multiple price/performance integration points and can enable functions that allow differentiation at the system and component level. Features are categorized by the following benefits:

#### Easy to use for end user

- Single model for cabling and connectors
- Electrical details isolated from end user (e.g., bus terminations)
- Self-identifying peripherals, automatic mapping of function to driver, and configuration
- Dynamically attachable and reconfigurable peripherals

## Universal Serial Bus Specification Revision 1.1

### Wide range of workloads and applications

- Suitable for device bandwidths ranging from a few kb/s to several Mb/s
- Supports isochronous as well as asynchronous transfer types over the same set of wires
- Supports concurrent operation of many devices (multiple connections)
- Supports up to 127 physical devices
- Supports transfer of multiple data and message streams between the host and devices
- Allows compound devices (i.e., peripherals composed of many functions)
- Lower protocol overhead, resulting in high bus utilization

### Isochronous bandwidth

- Guaranteed bandwidth and low latencies appropriate for telephony, audio, etc.
- Isochronous workload may use entire bus bandwidth

### Flexibility

- Supports a wide range of packet sizes, which allows a range of device buffering options
- Allows a wide range of device data rates by accommodating packet buffer size and latencies
- Flow control for buffer handling is built into the protocol

### Robustness

- Error handling/fault recovery mechanism is built into the protocol
- Dynamic insertion and removal of devices is identified in user-perceived real-time
- Supports identification of faulty devices

### Synergy with PC industry

- Protocol is simple to implement and integrate
- Consistent with the PC plug-and-play architecture
- Leverages existing operating system interfaces

### Low-cost implementation

- Low-cost subchannel at 1.5Mb/s
- Optimized for integration in peripheral and host hardware
- Suitable for development of low-cost peripherals
- Low-cost cables and connectors
- Uses commodity technologies

### Upgrade path

- Architecture upgradeable to support multiple USB Host Controllers in a system

**Universal Serial Bus Specification Revision 1.1**

# Chapter 4

## Architectural Overview

This chapter presents an overview of the Universal Serial Bus (USB) architecture and key concepts. The USB is a cable bus that supports data exchange between a host computer and a wide range of simultaneously accessible peripherals. The attached peripherals share USB bandwidth through a host-scheduled, token-based protocol. The bus allows peripherals to be attached, configured, used, and detached while the host and other peripherals are in operation.

Later chapters describe the various components of the USB in greater detail.

### 4.1 USB System Description

A USB system is described by three definitional areas:

- USB interconnect
- USB devices
- USB host.

The USB interconnect is the manner in which USB devices are connected to and communicate with the host. This includes the following:

- **Bus Topology:** Connection model between USB devices and the host.
- **Inter-layer Relationships:** In terms of a capability stack, the USB tasks that are performed at each layer in the system.
- **Data Flow Models:** The manner in which data moves in the system over the USB between producers and consumers.
- **USB Schedule:** The USB provides a shared interconnect. Access to the interconnect is scheduled in order to support isochronous data transfers and to eliminate arbitration overhead.

USB devices and the USB host are described in detail in subsequent sections.



### 4.1.1 Bus Topology

The USB connects USB devices with the USB host. The USB physical interconnect is a tiered star topology. A hub is at the center of each star. Each wire segment is a point-to-point connection between the host and a hub or function, or a hub connected to another hub or function. Figure 4-1 illustrates the topology of the USB.

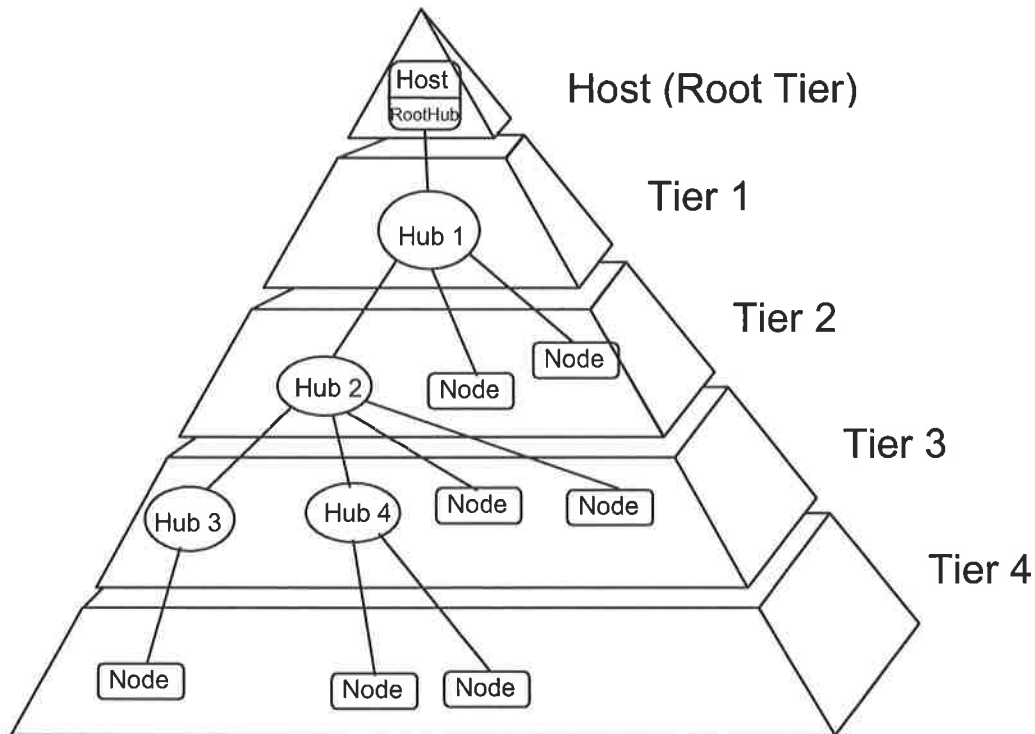


Figure 4-1. Bus Topology

#### 4.1.1.1 USB Host

There is only one host in any USB system. The USB interface to the host computer system is referred to as the Host Controller. The Host Controller may be implemented in a combination of hardware, firmware, or software. A root hub is integrated within the host system to provide one or more attachment points.

Additional information concerning the host may be found in Section 4.9 and in Chapter 10.

#### 4.1.1.2 USB Devices

USB devices are one of the following:

- Hubs, which provide additional attachment points to the USB
- Functions, which provide capabilities to the system, such as an ISDN connection, a digital joystick, or speakers.

## Universal Serial Bus Specification Revision 1.1

USB devices present a standard USB interface in terms of the following:

- Their comprehension of the USB protocol
- Their response to standard USB operations, such as configuration and reset
- Their standard capability descriptive information.

Additional information concerning USB devices may be found in Section 4.8 and in Chapter 9.

### 4.2 Physical Interface

The physical interface of the USB is described in the electrical (Chapter 7) and mechanical (Chapter 6) specifications for the bus.

#### 4.2.1 Electrical

The USB transfers signal and power over a four-wire cable, shown in Figure 4-2. The signaling occurs over two wires on each point-to-point segment.

There are two data rates:

- The USB full-speed signaling bit rate is 12Mb/s.
- A limited capability low-speed signaling mode is also defined at 1.5Mb/s.

The low-speed mode requires less EMI protection. Both modes can be supported in the same USB bus by automatic dynamic mode switching between transfers. The low-speed mode is defined to support a limited number of low-bandwidth devices, such as mice, because more general use would degrade bus utilization.

The clock is transmitted, encoded along with the differential data. The clock encoding scheme is NRZI with bit stuffing to ensure adequate transitions. A SYNC field precedes each packet to allow the receiver(s) to synchronize their bit recovery clocks.



Figure 4-2. USB Cable

The cable also carries VBUS and GND wires on each segment to deliver power to devices. VBUS is nominally +5V at the source. The USB allows cable segments of variable lengths, up to several meters, by choosing the appropriate conductor gauge to match the specified IR drop and other attributes such as device power budget and cable flexibility. In order to provide guaranteed input voltage levels and proper termination impedance, biased terminations are used at each end of the cable. The terminations also permit the detection of attach and detach at each port and differentiate between full-speed and low-speed devices.

#### 4.2.2 Mechanical

The mechanical specifications for cables and connectors are provided in Chapter 6. All devices have an upstream connection. Upstream and downstream connectors are not mechanically interchangeable, thus eliminating illegal loopback connections at hubs. The cable has four conductors: a twisted signal pair of standard gauge and a power pair in a range of permitted gauges. The connector is four-position, with shielded housing, specified robustness, and ease of attach-detach characteristics.

## 4.3 Power

The specification covers two aspects of power:

- Power distribution over the USB deals with the issues of how USB devices consume power provided by the host over the USB.
- Power management deals with how the USB System Software and devices fit into the host-based power management system.

### 4.3.1 Power Distribution

Each USB segment provides a limited amount of power over the cable. The host supplies power for use by USB devices that are directly connected. In addition, any USB device may have its own power supply. USB devices that rely totally on power from the cable are called bus-powered devices. In contrast, those that have an alternate source of power are called self-powered devices. A hub also supplies power for its connected USB devices. The architecture permits bus-powered hubs within certain constraints of topology that are discussed later in Chapter 11. In Figure 4-4 (see Section 4.8.2.1), the keyboard, pen, and mouse can all be bus-powered devices.

### 4.3.2 Power Management

A USB host may have a power management system that is independent of the USB. The USB System Software interacts with the host's power management system to handle system power events such as suspend or resume. Additionally, USB devices typically implement additional power management features that allow them to be power managed by system software.

The power distribution and power management features of the USB allow it to be designed into power-sensitive systems such as battery-based notebook computers.

## 4.4 Bus Protocol

The USB is a polled bus. The Host Controller initiates all data transfers.

All bus transactions involve the transmission of up to three packets. Each transaction begins when the Host Controller, on a scheduled basis, sends a USB packet describing the type and direction of transaction, the USB device address, and endpoint number. This packet is referred to as the "token packet." The USB device that is addressed selects itself by decoding the appropriate address fields. In a given transaction, data is transferred either from the host to a device or from a device to the host. The direction of data transfer is specified in the token packet. The source of the transaction then sends a data packet or indicates it has no data to transfer. The destination, in general, responds with a handshake packet indicating whether the transfer was successful.

The USB data transfer model between a source or destination on the host and an endpoint on a device is referred to as a pipe. There are two types of pipes: stream and message. Stream data has no USB-defined structure, while message data does. Additionally, pipes have associations of data bandwidth, transfer service type, and endpoint characteristics like directionality and buffer sizes. Most pipes come into existence when a USB device is configured. One message pipe, the Default Control Pipe, always exists once a device is powered, in order to provide access to the device's configuration, status, and control information.

The transaction schedule allows flow control for some stream pipes. At the hardware level, this prevents buffers from underrun or overrun situations by using a NAK handshake to throttle the data rate. When NAKed, a transaction is retried when bus time is available. The flow control mechanism permits the construction of flexible schedules that accommodate concurrent servicing of a heterogeneous mix of stream pipes. Thus, multiple stream pipes can be serviced at different intervals and with packets of different sizes.

## 4.5 Robustness

There are several attributes of the USB that contribute to its robustness:

- Signal integrity using differential drivers, receivers, and shielding
- CRC protection over control and data fields
- Detection of attach and detach and system-level configuration of resources
- Self-recovery in protocol, using timeouts for lost or corrupted packets
- Flow control for streaming data to ensure isochrony and hardware buffer management
- Data and control pipe constructs for ensuring independence from adverse interactions between functions.

### 4.5.1 Error Detection

The core bit error rate of the USB medium is expected to be close to that of a backplane and any glitches will very likely be transient in nature. To provide protection against such transients, each packet includes error protection fields. When data integrity is required, such as with lossless data devices, an error recovery procedure may be invoked in hardware or software.

The protocol includes separate CRCs for control and data fields of each packet. A failed CRC is considered to indicate a corrupted packet. The CRC gives 100% coverage on single- and double-bit errors.

### 4.5.2 Error Handling

The protocol allows for error handling in hardware or software. Hardware error handling includes reporting and retry of failed transfers. A USB Host Controller will try a transmission that encounters errors up to three times before informing the client software of the failure. The client software can recover in an implementation-specific way.

## 4.6 System Configuration

The USB supports USB devices attaching to and detaching from the USB at any time. Consequently, system software must accommodate dynamic changes in the physical bus topology.

### 4.6.1 Attachment of USB Devices

All USB devices attach to the USB through ports on specialized USB devices known as hubs. Hubs have status indicators that indicate the attachment or removal of a USB device on one of its ports. The host queries the hub to retrieve these indicators. In the case of an attachment, the host enables the port and addresses the USB device through the device's control pipe at the default address.

The host assigns a unique USB address to the device and then determines if the newly attached USB device is a hub or a function. The host establishes its end of the control pipe for the USB device using the assigned USB address and endpoint number zero.

If the attached USB device is a hub and USB devices are attached to its ports, then the above procedure is followed for each of the attached USB devices.

If the attached USB device is a function, then attachment notifications will be handled by host software that is appropriate for the function.

#### 4.6.2 Removal of USB Devices

When a USB device has been removed from one of a hub's ports, the hub disables the port and provides an indication of device removal to the host. The removal indication is then handled by appropriate USB System Software. If the removed USB device is a hub, the USB System Software must handle the removal of both the hub and of all of the USB devices that were previously attached to the system through the hub.

#### 4.6.3 Bus Enumeration

Bus enumeration is the activity that identifies and assigns unique addresses to devices attached to a bus. Because the USB allows USB devices to attach to or detach from the USB at any time, bus enumeration is an on-going activity for the USB System Software. Additionally, bus enumeration for the USB also includes the detection and processing of removals.

### 4.7 Data Flow Types

The USB supports functional data and control exchange between the USB host and a USB device as a set of either uni-directional or bi-directional pipes. USB data transfers take place between host software and a particular endpoint on a USB device. Such associations between the host software and a USB device endpoint are called pipes. In general, data movement through one pipe is independent from the data flow in any other pipe. A given USB device may have many pipes. As an example, a given USB device could have an endpoint that supports a pipe for transporting data to the USB device and another endpoint that supports a pipe for transporting data from the USB device.

The USB architecture comprehends four basic types of data transfers:

- Control Transfers: Used to configure a device at attach time and can be used for other device-specific purposes, including control of other pipes on the device.
- Bulk Data Transfers: Generated or consumed in relatively large and bursty quantities and have wide dynamic latitude in transmission constraints.
- Interrupt Data Transfers: Used for characters or coordinates with human-perceptible echo or feedback response characteristics.
- Isochronous Data Transfers: Occupy a prenegotiated amount of USB bandwidth with a prenegotiated delivery latency. (Also called streaming real time transfers).

A pipe supports only one of the types of transfers described above for any given device configuration. The USB data flow model is described in more detail in Chapter 5.

#### 4.7.1 Control Transfers

Control data is used by the USB System Software to configure devices when they are first attached. Other driver software can choose to use control transfers in implementation-specific ways. Data delivery is lossless.

#### 4.7.2 Bulk Transfers

Bulk data typically consists of larger amounts of data, such as that used for printers or scanners. Bulk data is sequential. Reliable exchange of data is ensured at the hardware level by using error detection in hardware and invoking a limited number of retries in hardware. Also, the bandwidth taken up by bulk data can vary, depending on other bus activities.

### 4.7.3 Interrupt Transfers

A small, limited-latency transfer to or from a device is referred to as interrupt data. Such data may be presented for transfer by a device at any time and is delivered by the USB at a rate no slower than is specified by the device.

Interrupt data typically consists of event notification, characters, or coordinates that are organized as one or more bytes. An example of interrupt data is the coordinates from a pointing device. Although an explicit timing rate is not required, interactive data may have response time bounds that the USB must support.

### 4.7.4 Isochronous Transfers

Isochronous data is continuous and real-time in creation, delivery, and consumption. Timing-related information is implied by the steady rate at which isochronous data is received and transferred. Isochronous data must be delivered at the rate received to maintain its timing. In addition to delivery rate, isochronous data may also be sensitive to delivery delays. For isochronous pipes, the bandwidth required is typically based upon the sampling characteristics of the associated function. The latency required is related to the buffering available at each endpoint.

A typical example of isochronous data is voice. If the delivery rate of these data streams is not maintained, drop-outs in the data stream will occur due to buffer or frame underruns or overruns. Even if data is delivered at the appropriate rate by USB hardware, delivery delays introduced by software may degrade applications requiring real-time turn-around, such as telephony-based audio conferencing.

The timely delivery of isochronous data is ensured at the expense of potential transient losses in the data stream. In other words, any error in electrical transmission is not corrected by hardware mechanisms such as retries. In practice, the core bit error rate of the USB is expected to be small enough not to be an issue. USB isochronous data streams are allocated a dedicated portion of USB bandwidth to ensure that data can be delivered at the desired rate. The USB is also designed for minimal delay of isochronous data transfers.

### 4.7.5 Allocating USB Bandwidth

USB bandwidth is allocated among pipes. The USB allocates bandwidth for some pipes when a pipe is established. USB devices are required to provide some buffering of data. It is assumed that USB devices requiring more bandwidth are capable of providing larger buffers. The goal for the USB architecture is to ensure that buffering-induced hardware delay is bounded to within a few milliseconds.

The USB's bandwidth capacity can be allocated among many different data streams. This allows a wide range of devices to be attached to the USB. For example, telephony devices ranging from 1B+D all the way up to T1 capacity can be accommodated. Further, different device bit rates, with a wide dynamic range, can be concurrently supported.

The USB Specification defines the rules for how each transfer type is allowed access to the bus.

## 4.8 USB Devices

USB devices are divided into device classes such as hub, locator, or text device. The hub device class indicates a specially designated USB device that provides additional USB attachment points (refer to Chapter 11). USB devices are required to carry information for self-identification and generic configuration. They are also required at all times to display behavior consistent with defined USB device states.

### 4.8.1 Device Characterizations

All USB devices are accessed by a USB address that is assigned when the device is attached and enumerated. Each USB device additionally supports one or more pipes through which the host may communicate with the device. All USB devices must support a specially designated pipe at endpoint zero

## Universal Serial USB Specification Revision 1.1

to which the USB device's USB control pipe will be attached. All USB devices support a common accesses mechanism for accessing information through this control pipe.

Associated with the control pipe at endpoint zero is the information required to completely describe the USB device. This information falls into the following categories:

- **Standard:** This is information whose definition is common to all USB devices and includes items such as vendor identification, device class, and power management. Device, configuration, interface, and endpoint descriptions carry configuration-related information about the device. Detailed information about these descriptors can be found in Chapter 9.
- **Class:** The definition of this information varies, depending on the device class of the USB device.
- **USB Vendor:** The vendor of the USB device is free to put any information desired here. The format, however, is not determined by this specification.

Additionally, each USB device carries USB control and status information.

### 4.8.2 Device Descriptions

Two major divisions of device classes exist: hubs and functions. Only hubs have the ability to provide additional USB attachment points. Functions provide additional capabilities to the host.

#### 4.8.2.1 Hubs

Hubs are a key element in the plug-and-play architecture of the USB. Figure 4-3 shows a typical hub. Hubs serve to simplify USB connectivity from the user's perspective and provide robustness at low cost and complexity.

Hubs are wiring concentrators and enable the multiple attachment characteristics of the USB. Attachment points are referred to as ports. Each hub converts a single attachment point into multiple attachment points. The architecture supports concatenation of multiple hubs.

The upstream port of a hub connects the hub towards the host. Each of the downstream ports of a hub allows connection to another hub or function. Hubs can detect attach and detach at each downstream port and enable the distribution of power to downstream devices. Each downstream port can be individually enabled and attached to either full- or low-speed devices. The hub isolates low-speed ports from full-speed signaling.

A hub consists of two portions: the Hub Controller and the Hub Repeater. The Hub Repeater is a protocol-controlled switch between the upstream port and downstream ports. It also has hardware support for reset and suspend/resume signaling. The Host Controller provides the interface registers to allow communication to/from the host. Hub-specific status and control commands permit the host to configure a hub and to monitor and control its ports.

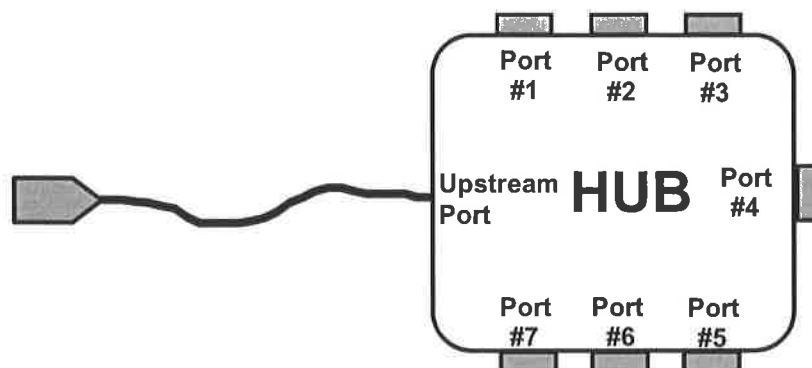


Figure 4-3. A Typical Hub

Figure 4-4 illustrates how hubs provide connectivity in a typical computer environment.

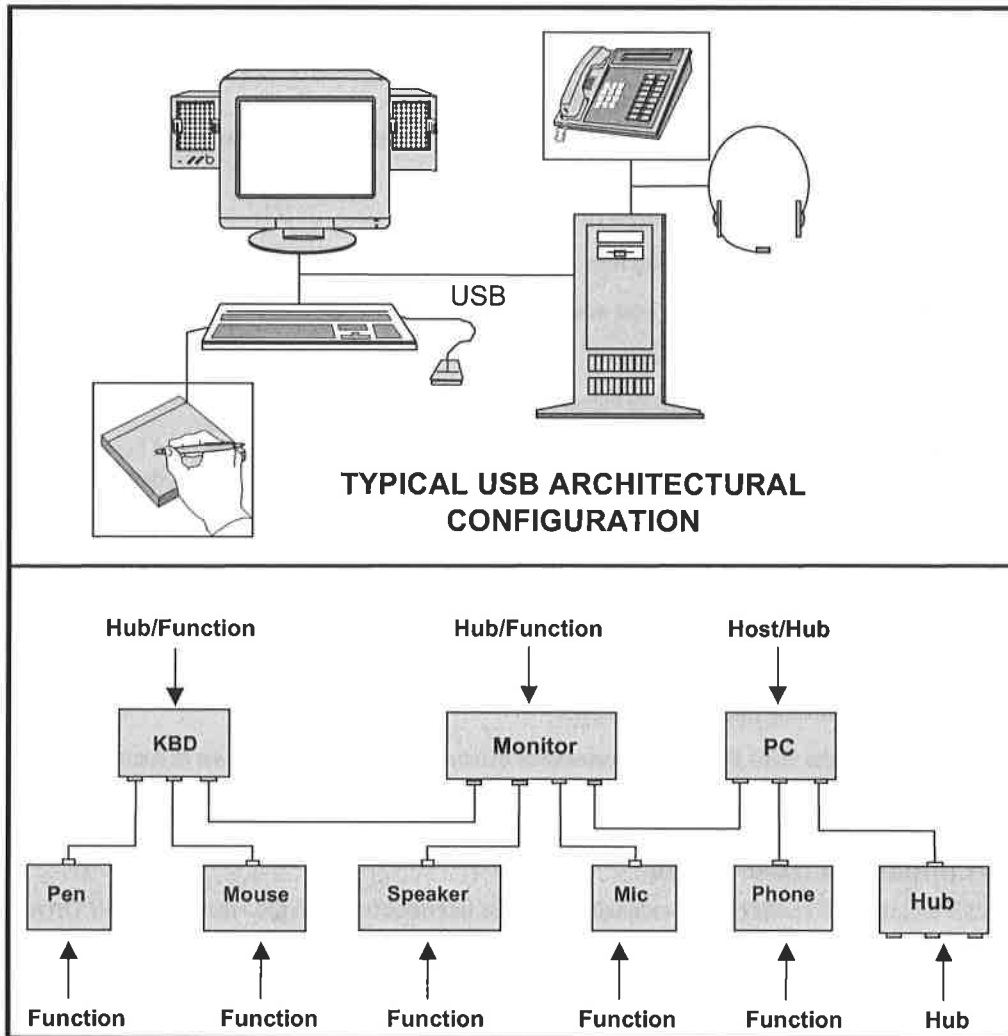


Figure 4-4. Hubs in a Desktop Computer Environment

#### 4.8.2.2 Functions

A function is a USB device that is able to transmit or receive data or control information over the bus. A function is typically implemented as a separate peripheral device with a cable that plugs into a port on a hub. However, a physical package may implement multiple functions and an embedded hub with a single USB cable. This is known as a compound device. A compound device appears to the host as a hub with one or more non-removable USB devices.

Each function contains configuration information that describes its capabilities and resource requirements. Before a function can be used, it must be configured by the host. This configuration includes allocating USB bandwidth and selecting function-specific configuration options.

Examples of functions include the following:

- A locator device such as a mouse, tablet, or light pen
- An input device such as a keyboard



- An output device such as a printer
- A telephony adapter such as ISDN.

#### 4.9 USB Host: Hardware and Software

The USB host interacts with USB devices through the Host Controller. The host is responsible for the following:

- Detecting the attachment and removal of USB devices
- Managing control flow between the host and USB devices
- Managing data flow between the host and USB devices
- Collecting status and activity statistics
- Providing power to attached USB devices.

The USB System Software on the host manages interactions between USB devices and host-based device software. There are five areas of interactions between the USB System Software and device software:

- Device enumeration and configuration
- Isochronous data transfers
- Asynchronous data transfers
- Power management
- Device and bus management information.

Whenever possible, the USB System Software uses existing host system interfaces to manage the above interactions.

#### 4.10 Architectural Extensions

The USB architecture comprehends extensibility at the interface between the Host Controller Driver and USB Driver. Implementations with multiple Host Controllers, and associated Host Controller Drivers, are possible.

## Chapter 5

# USB Data Flow Model

This chapter presents information about how data is moved across the USB. The information in this chapter affects all implementers. The information presented is at a level above the signaling and protocol definitions of the system. Consult Chapter 7 and Chapter 8 for more details about their respective parts of the USB system. This chapter provides framework information that is further expanded in Chapters 9 through 11. All implementers should read this chapter so they understand the key concepts of the USB.

### 5.1 Implementer Viewpoints

The USB provides communication services between a host and attached USB devices. However, the simple view an end user sees of attaching one or more USB devices to a host, as in Figure 5-1, is in fact a little more complicated to implement than is indicated by the figure. Different views of the system are required to explain specific USB requirements from the perspective of different implementers. Several important concepts and features must be supported to provide the end user with the reliable operation demanded from today's personal computers. The USB is presented in a layered fashion to ease explanation and allow implementers of particular USB products to focus on the details related to their product.



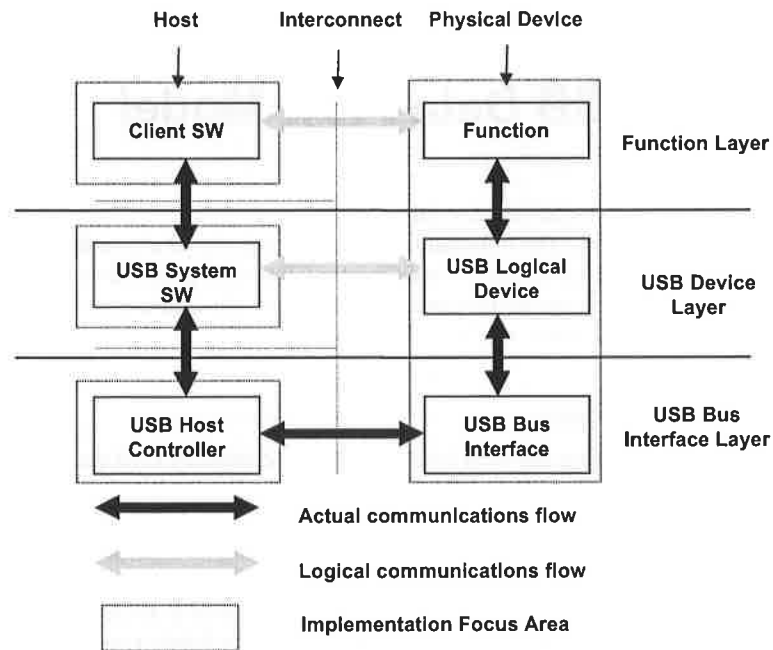
**Figure 5-1. Simple USB Host/Device View**

Figure 5-2 shows a deeper overview of the USB, identifying the different layers of the system that will be described in more detail in the remainder of the specification. In particular, there are four focus implementation areas:

- **USB Physical Device:** A piece of hardware on the end of a USB cable that performs some useful end user function.
- **Client Software:** Software that executes on the host, corresponding to a USB device. This client software is typically supplied with the operating system or provided along with the USB device.
- **USB System Software:** Software that supports the USB in a particular operating system. The USB System Software is typically supplied with the operating system, independently of particular USB devices or client software.
- **USB Host Controller (Host Side Bus Interface):** The hardware and software that allows USB devices to be attached to a host.

There are shared rights and responsibilities between the four USB system components. The remainder of this specification describes the details required to support robust, reliable communication flows between a function and its client.

## Universal Serial Bus Specification Revision 1.1



**Figure 5-2. USB Implementation Areas**

As shown in Figure 5-2, the simple connection of a host to a device requires interaction between a number of layers and entities. The USB Bus Interface layer provides physical/signaling/packet connectivity between the host and a device. The USB Device Layer is the view the USB System Software has for performing generic USB operations with a device. The Function Layer provides additional capabilities to the host via an appropriate matched client software layer. The USB Device and Function layers each have a view of logical communication within their layer that actually uses the USB Bus Interface Layer to accomplish data transfer.

The physical view of USB communication as described in Chapters 6, 7, and 8 is related to the logical communication view presented in Chapters 9 and 10. This chapter describes those key concepts that affect USB implementers and should be read by all before proceeding to the remainder of the specification to find those details most relevant to their product.

To describe and manage USB communication, the following concepts are important:

- **Bus Topology:** Section 5.2 presents the primary physical and logical components of the USB and how they interrelate.
- **Communication Flow Models:** Sections 5.3 through 5.8 describe how communication flows between the host and devices through the USB and defines the four USB transfer types.
- **Bus Access Management:** Section 5.9 describes how bus access is managed within the host to support a broad range of communication flows by USB devices.
- **Special Consideration for Isochronous Transfers:** Section 5.10 presents features of the USB specific to devices requiring isochronous data transfers. Device implementers for non-isochronous devices do not need to read Section 5.10.

## 5.2 Bus Topology

There are four main parts to USB topology:

- Host and Devices: The primary components of a USB system.
- Physical Topology: How USB elements are connected.
- Logical Topology: The roles and responsibilities of the various USB elements and how the USB appears from the perspective of the host and a device.
- Client Software-to-function Relationships: How client software and its related function interfaces on a USB device view each other.

### 5.2.1 USB Host

The host's logical composition is shown in Figure 5-3, and includes the following:

- USB Host Controller
- Aggregate USB System Software (USB Driver, Host Controller Driver, and host software)
- Client.

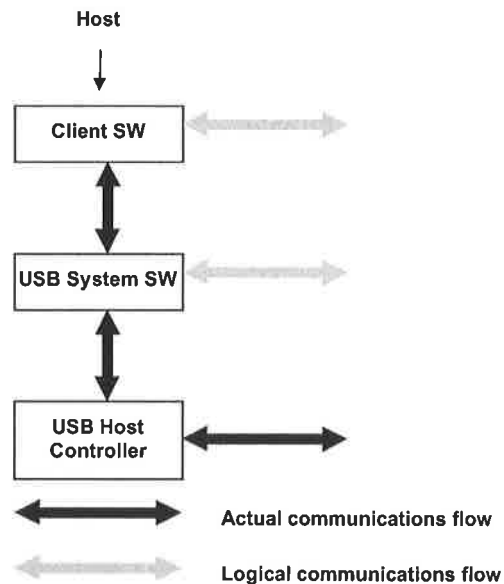


Figure 5-3. Host Composition

The USB host occupies a unique position as the coordinating entity for the USB. In addition to its special physical position, the host has specific responsibilities with regard to the USB and its attached devices. The host controls all access to the USB. A USB device gains access to the bus only by being granted access by the host. The host is also responsible for monitoring the topology of the USB.

For a complete discussion of the host and its duties, refer to Chapter 10.

### 5.2.2 USB Devices

A USB physical device's logical composition is shown in Figure 5-4, and includes the following:

- USB bus interface
- USB logical device
- Function.

USB physical devices provide additional functionality to the host. The types of functionality provided by USB devices vary widely. However, all USB logical devices present the same basic interface to the host. This allows the host to manage the USB-relevant aspects of different USB devices in the same manner.

To assist the host in identifying and configuring USB devices, each device carries and reports configuration-related information. Some of the information reported is common among all logical devices. Other information is specific to the functionality provided by the device. The detailed format of this information varies, depending on the device class of the device.

For a complete discussion of USB devices, refer to Chapter 9.

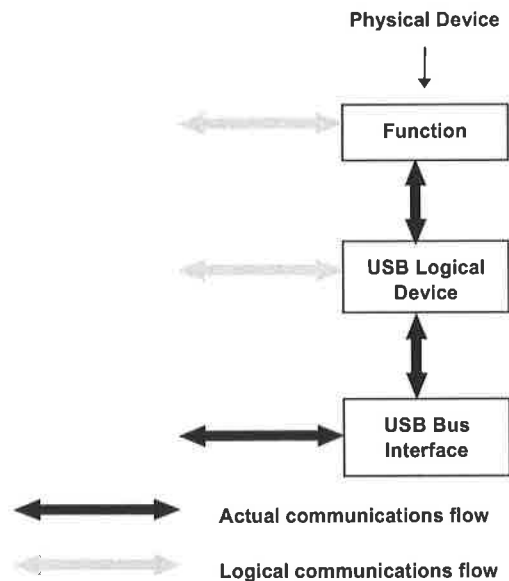


Figure 5-4. Physical Device Composition

### 5.2.3 Physical Bus Topology

Devices on the USB are physically connected to the host via a tiered star topology, as illustrated in Figure 5-5. USB attachment points are provided by a special class of USB device known as a hub. The additional attachment points provided by a hub are called ports. A host includes an embedded hub called the root hub. The host provides one or more attachment points via the root hub. USB devices that provide additional functionality to the host are known as functions. To prevent circular attachments, a tiered ordering is imposed on the star topology of the USB. This results in the tree-like configuration illustrated in Figure 5-5.

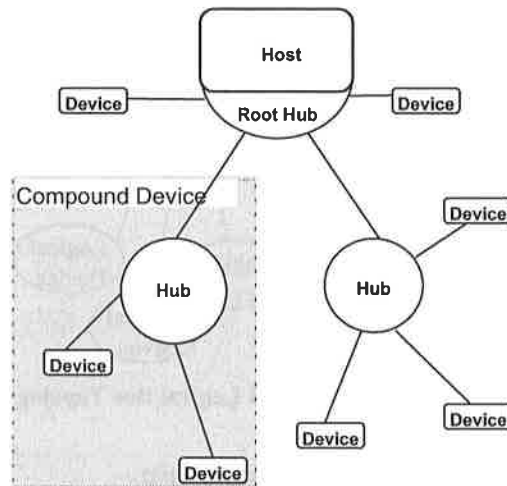


Figure 5-5. USB Physical Bus Topology

Multiple functions may be packaged together in what appears to be a single physical device. For example, a keyboard and a trackball might be combined in a single package. Inside the package, the individual functions are permanently attached to a hub and it is the internal hub that is connected to the USB. When multiple functions are combined with a hub in a single package, they are referred to as a compound device. From the host's perspective, a compound device is the same as a separate hub with multiple functions attached. Figure 5-5 also illustrates a compound device.

### 5.2.4 Logical Bus Topology

While devices physically attach to the USB in a tiered, star topology, the host communicates with each logical device as if it were directly connected to the root port. This creates the logical view illustrated in Figure 5-6 that corresponds to the physical topology shown in Figure 5-5. Hubs are logical devices also, but are not shown in Figure 5-6 to simplify the picture. Even though most host/logical device activities use this logical perspective, the host maintains an awareness of the physical topology to support processing the removal of hubs. When a hub is removed, all of the devices attached to the hub must be removed from the host's view of the logical topology. A more complete discussion of hubs can be found in Chapter 11.

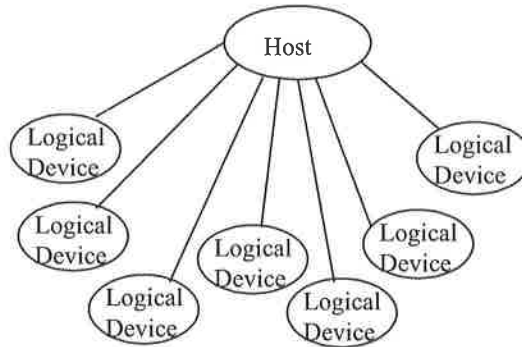


Figure 5-6. USB Logical Bus Topology

### 5.2.5 Client Software-to-function Relationship

Even though the physical and logical topology of the USB reflects the shared nature of the bus, client software (CSw) manipulating a USB function interface is presented with the view that it deals only with its interface(s) of interest. Client software for USB functions must use USB software programming interfaces to manipulate their functions as opposed to directly manipulating their functions via memory or I/O accesses as with other buses (e.g., PCI, EISA, PCMCIA, etc.). During operation, client software should be independent of other devices that may be connected to the USB. This allows the designer of the device and client software to focus on the hardware/software interaction design details. Figure 5-7 illustrates a device designer's perspective of the relationships of client software and USB functions with respect to the USB logical topology of Figure 5-6.

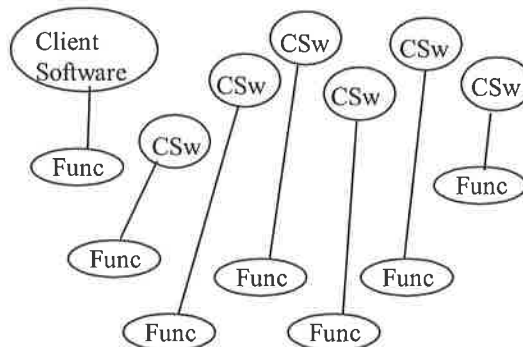


Figure 5-7. Client Software-to-function Relationships

### 5.3 USB Communication Flow

The USB provides a communication service between software on the host and its USB function. Functions can have different communication flow requirements for different client-to-function interactions. The USB provides better overall bus utilization by allowing the separation of the different communication flows to a USB function. Each communication flow makes use of some bus access to accomplish communication between client and function. Each communication flow is terminated at an endpoint on a device. Device endpoints are used to identify aspects of each communication flow.

Figure 5-8 shows a more detailed view of Figure 5-2. The complete definition of the actual communication flows of Figure 5-2 supports the logical device and function layer communication flows. These actual communication flows cross several interface boundaries. Chapters 6 through 8 describe the mechanical, electrical, and protocol interface definitions of the USB “wire.” Chapter 9 describes the USB device programming interface that allows a USB device to be manipulated from the host side of the wire. Chapter 10 describes two host side software interfaces:

- **Host Controller Driver (HCD):** The software interface between the USB Host Controller and USB System Software. This interface allows a range of Host Controller implementations without requiring all host software to be dependent on any particular implementation. One USB Driver can support different Host Controllers without requiring specific knowledge of a Host Controller implementation. A Host Controller implementer provides an HCD implementation that supports the Host Controller.
- **USB Driver (USB D):** The interface between the USB System Software and the client software. This interface provides clients with convenient functions for manipulating USB devices.

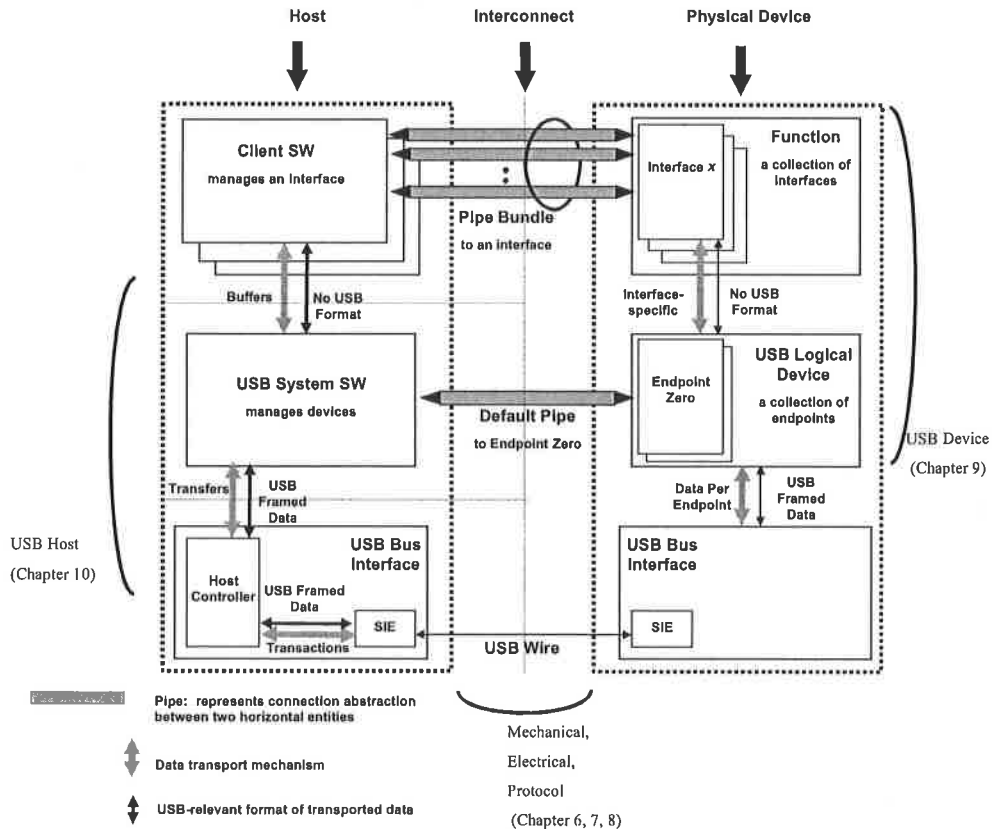


Figure 5-8. USB Host/Device Detailed View



## Universal Serial Bus Specification Revision 1.1

A USB logical device appears to the USB system as a collection of endpoints. Endpoints are grouped into endpoint sets that implement an interface. Interfaces are views to the function. The USB System Software manages the device using the Default Control Pipe. Client software manages an interface using pipe bundles (associated with an endpoint set). Client software requests that data be moved across the USB between a buffer on the host and an endpoint on the USB device. The Host Controller (or USB device, depending on transfer direction) packetizes the data to move it over the USB. The Host Controller also coordinates when bus access is used to move the packet of data over the USB.

Figure 5-9 illustrates how communication flows are carried over pipes between endpoints and host side memory buffers. The following sections describe endpoints, pipes, and communication flows in more detail.

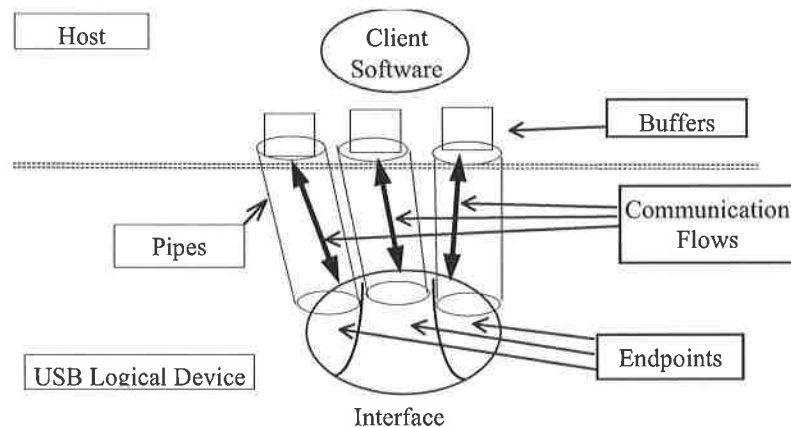


Figure 5-9. USB Communication Flow

Software on the host communicates with a logical device via a set of communication flows. The set of communication flows are selected by the device software/hardware designer(s) to efficiently match the communication requirements of the device to the transfer characteristics provided by the USB.

### 5.3.1 Device Endpoints

An endpoint is a uniquely identifiable portion of a USB device that is the terminus of a communication flow between the host and device. Each USB logical device is composed of a collection of independent endpoints. Each logical device has a unique address assigned by the system at device attachment time. Each endpoint on a device is given at design time a unique device-determined identifier called the endpoint number. Each endpoint has a device-determined direction of data flow. The combination of the device address, endpoint number, and direction allows each endpoint to be uniquely referenced. Each endpoint is a simplex connection that supports data flow in one direction: either input (from device to host) or output (from host to device).

An endpoint has characteristics that determine the type of transfer service required between the endpoint and the client software. Endpoints describe themselves by:

- Their bus access frequency/latency requirements
- Their bandwidth requirements
- Their endpoint number
- The error handling behavior requirements
- Maximum packet size that the endpoint is capable of sending or receiving

- The transfer type for the endpoint (refer to Section 5.4 for details)
- The direction data is transferred between the endpoint and the host.

Endpoints other than those with endpoint number zero are in an unknown state before being configured and may not be accessed by the host before being configured.

### 5.3.1.1 Endpoint Zero Requirements

All USB devices are required to implement a default control method that uses both the input and output endpoints with endpoint number zero. The USB System Software uses this default control method to initialize and generically manipulate the logical device (e.g., to configure the logical device) as the Default Control Pipe (see Section 5.3.2). The Default Control Pipe provides access to the device's configuration information and allows generic USB status and control access. The Default Control Pipe supports control transfers as defined in Section 5.5. The endpoints with endpoint number zero are always accessible once a device is attached, powered, and has received a bus reset.

### 5.3.1.2 Non-endpoint Zero Requirements

Functions can have additional endpoints as required for their implementation. Low-speed functions are limited to two optional endpoints beyond the two required to implement the Default Control Pipe. Full-speed devices can have additional endpoints only limited by the protocol definition (i.e., a maximum of 15 additional input endpoints and 15 additional output endpoints).

Endpoints other than those for the Default Control Pipe cannot be used until the device is configured as a normal part of the device configuration process (refer to Chapter 9).

## 5.3.2 Pipes

A USB pipe is an association between an endpoint on a device and software on the host. Pipes represent the ability to move data between software on the host via a memory buffer and an endpoint on a device. There are two different, mutually exclusive, pipe communication modes:

- Stream: Data moving through a pipe has no USB-defined structure
- Message: Data moving through a pipe has some USB-defined structure.

The USB does not interpret the content of data it delivers through a pipe. Even though a message pipe requires that data be structured according to USB definitions, the content of the data is not interpreted by the USB.

Additionally, pipes have the following associated with them:

- A claim on USB bus access and bandwidth usage.
- A transfer type.
- The associated endpoint's characteristics, such as directionality and maximum data payload sizes. The data payload is the data that is carried in the data field of a data packet within a bus transaction (as defined in Chapter 8).

The pipe that consists of the two endpoints with endpoint number zero is called the Default Control Pipe. This pipe is always available once a device is powered and has received a bus reset. Other pipes come into existence when a USB device is configured. The Default Control Pipe is used by the USB System Software to determine device identification and configuration requirements, and to configure the device. The Default Control Pipe can also be used by device-specific software after the device is configured. The USB System Software retains "ownership" of the Default Control Pipe and mediates use of the pipe by other client software.

## Universal Serial Bus Specification Revision 1.1

A software client normally requests data transfers via I/O Request Packets (IRPs) to a pipe and then either waits or is notified when they are completed. Details about IRPs are defined in an operating system-specific manner. This specification uses the term to simply refer to an identifiable request by a software client to move data between itself (on the host) and an endpoint of a device in an appropriate direction. A software client can cause a pipe to return all outstanding IRPs if it desires. The software client is notified that an IRP has completed when the bus transactions associated with it have completed either successfully or due to errors.

If there are no IRPs pending or in progress for a pipe, the pipe is idle and the Host Controller will take no action with regard to the pipe; i.e., the endpoint for such a pipe will not see any bus transactions directed to it. The only time bus activity is present for a pipe is when IRPs are pending for that pipe.

If a non-isochronous pipe encounters a condition that causes it to send a STALL to the host (refer to Chapter 8) or three bus errors are encountered on any packet of an IRP, the IRP is aborted/retired, all outstanding IRPs are also retired, and no further IRPs are accepted until the software client recovers from the condition (in an implementation-dependent way) and acknowledges the halt or error condition via a USBD call. An appropriate status informs the software client of the specific IRP result for error versus halt (refer to Chapter 10). Isochronous pipe behavior is described in Section 5.6.

An IRP may require multiple data payloads to move the client data over the bus. The data payloads for such a multiple data payload IRP are expected to be of the maximum packet size until the last data payload that contains the remainder of the overall IRP. See the description of each transfer type for more details. For such an IRP, short packets (i.e., less than maximum-sized data payloads) on input that do not completely fill an IRP data buffer can have one of two possible meanings, depending upon the expectations of a client:

- A client can expect a variable-sized amount of data in an IRP. In this case, a short packet that does not fill an IRP data buffer can be used simply as an in-band delimiter to indicate “end of unit of data.” The IRP should be retired without error and the Host Controller should advance to the next IRP.
- A client can expect a specific-sized amount of data. In this case, a short packet that does not fill an IRP data buffer is an indication of an error. The IRP should be retired, the pipe should be stalled, and any pending IRPs associated with the pipe should also be retired.

Because the Host Controller must behave differently in the two cases and cannot know on its own which way to behave for a given IRP, it is possible to indicate per IRP which behavior the client desires.

An endpoint can inform the host that it is busy by responding with NAK. NAKs are not used as a retire condition for returning an IRP to a software client. Any number of NAKs can be encountered during the processing of a given IRP. A NAK response to a transaction does not constitute an error and is not counted as one of the three errors described above.

### 5.3.2.1 Stream Pipes

Stream pipes deliver data in the data packet portion of bus transactions with no USB-required structure on the data content. Data flows in at one end of a stream pipe and out the other end in the same order. Stream pipes are always uni-directional in their communication flow.

Data flowing through a stream pipe is expected to interact with what the USB believes is a single client. The USB System Software is not required to provide synchronization between multiple clients that may be using the same stream pipe. Data presented to a stream pipe is moved through the pipe in sequential order: first-in, first-out.

A stream pipe to a device is bound to a single device endpoint number in the appropriate direction (i.e., corresponding to an IN or OUT token as defined by the protocol layer). The device endpoint number for the opposite direction can be used for some other stream pipe to the device.

Stream pipes support bulk, isochronous, and interrupt transfer types, which are explained in later sections.

### 5.3.2.2 Message Pipes

Message pipes interact with the endpoint in a different manner than stream pipes. First, a request is sent to the USB device from the host. This request is followed by data transfer(s) in the appropriate direction. Finally, a Status stage follows at some later time. In order to accommodate the request/data/status paradigm, message pipes impose a structure on the communication flow that allows commands to be reliably identified and communicated. Message pipes allow communication flow in both directions, although the communication flow may be predominately one-way. The Default Control Pipe is always a message pipe.

The USB System Software ensures that multiple requests are not sent to a message pipe concurrently. A device is required to service only a single message request at a time per message pipe. Multiple software clients on the host can make requests via the Default Control Pipe, but they are sent to the device in a first-in, first-out order. A device can control the flow of information during the Data and Status stages based on its ability to respond to the host transactions (refer to Chapter 8 for more details).

A message pipe will not normally be sent the next message from the host until the current message's processing at the device has been completed. However, there are error conditions whereby a message transfer can be aborted by the host and the message pipe can be sent a new message transfer prematurely (from the device's perspective). From the perspective of the software manipulating a message pipe, an error on some part of an IRP retires the current IRP and all queued IRPs. The software client that requested the IRP is notified of the IRP completion with an appropriate error indication.

A message pipe to a device requires a single device endpoint number in both directions (IN and OUT tokens). The USB does not allow a message pipe to be associated with different endpoint numbers for each direction.

Message pipes support the control transfer type, which is explained in Section 5.5.

## 5.4 Transfer Types

The USB transports data through a pipe between a memory buffer associated with a software client on the host and an endpoint on the USB device. Data transported by message pipes is carried in a USB-defined structure, but the USB allows device-specific structured data to be transported within the USB-defined message data payload. The USB also defines that data moved over the bus is packetized for any pipe (stream or message), but ultimately the formatting and interpretation of the data transported in the data payload of a bus transaction is the responsibility of the client software and function using the pipe. However, the USB provides different transfer types that are optimized to more closely match the service requirements of the client software and function using the pipe. An IRP uses one or more bus transactions to move information between a software client and its function.

Each transfer type determines various characteristics of the communication flow including the following:

- Data format imposed by the USB
- Direction of communication flow
- Packet size constraints
- Bus access constraints
- Latency constraints
- Required data sequences
- Error handling.

The designers of a USB device choose the capabilities for the device's endpoints. When a pipe is established for an endpoint, most of the pipe's transfer characteristics are determined and remain fixed for the lifetime of the pipe. Transfer characteristics that can be modified are described for each transfer type.

The USB defines four transfer types:

- Control Transfers: Bursty, non-periodic, host software-initiated request/response communication, typically used for command/status operations.
- Isochronous Transfers: Periodic, continuous communication between host and device, typically used for time-relevant information. This transfer type also preserves the concept of time encapsulated in the data. This does not imply, however, that the delivery needs of such data is always time-critical.
- Interrupt Transfers: Small-data, low-frequency, bounded-latency communication.
- Bulk Transfers: Non-periodic, large-packet bursty communication, typically used for data that can use any available bandwidth and can also be delayed until bandwidth is available.

Each transfer type is described in detail in the following four major sections. The data for any IRP is carried by the data field of the data packet as described in Section 8.4.3. Chapter 8 also describes details of the protocol that are affected by use of each particular transfer type.

### 5.5 Control Transfers

Control transfers allow access to different parts of a device. Control transfers are intended to support configuration/command/status type communication flows between client software and its function. A control transfer is composed of a Setup bus transaction moving request information from host to function, zero or more Data transactions sending data in the direction indicated by the Setup transaction, and a Status transaction returning status information from function to host. The Status transaction returns “success” when the endpoint has successfully completed processing the requested operation. Section 8.5.2 describes the details of what packets, bus transactions, and transaction sequences are used to accomplish a control transfer. Chapter 9 describes the details of the defined USB command codes.

Each USB device is required to implement the Default Control Pipe as a message pipe. This pipe is used by the USB System Software. The Default Control Pipe provides access to the USB device’s configuration, status, and control information. A function can, but is not required to, provide endpoints for additional control pipes for its own implementation needs.

The USB device framework (refer to Chapter 9) defines standard, device class, or vendor-specific requests that can be used to manipulate a device’s state. Descriptors are also defined that can be used to contain different information on the device. Control transfers provide the transport mechanism to access device descriptors and make requests of a device to manipulate its behavior.

Control transfers are carried only through message pipes. Consequently, data flows using control transfers must adhere to USB data structure definitions as described in Section 5.5.1.

The USB system will make a “best effort” to support delivery of control transfers between the host and devices. A function and its client software cannot request specific bus access frequency or bandwidth for control transfers. The USB System Software may restrict the bus access and bandwidth that a device may desire for control transfers. These restrictions are defined in Section 5.5.3 and Section 5.5.4.

#### 5.5.1 Control Transfer Data Format

The Setup packet has a USB-defined structure that accommodates the minimum set of commands required to enable communication between the host and a device. The structure definition allows vendor-specific extensions for device specific commands. The Data transactions following Setup have a USB-defined structure except when carrying vendor-specific information. The Status transaction also has a USB-defined structure. Specific control transfer Setup/Data definitions are described in Section 8.5.2 and Chapter 9.

## 5.5.2 Control Transfer Direction

Control transfers are supported via bi-directional communication flow over message pipes. As a consequence, when a control pipe is configured, it uses both the input and output endpoint with the specified endpoint number.

## 5.5.3 Control Transfer Packet Size Constraints

An endpoint for control transfers specifies the maximum data payload size that the endpoint can accept from or transmit to the bus. The USB defines the allowable maximum control data payload sizes for full-speed devices to be either 8, 16, 32, or 64 bytes. Low-speed devices are limited to only an eight-byte maximum data payload size. This maximum applies to the data payloads of the Data packets following a Setup; i.e., the size specified is for the data field of the packet as defined in Chapter 8, not including other information that is required by the protocol. A Setup packet is always eight bytes. A control pipe (including the Default Control Pipe) always uses its *wMaxPacketSize* value for data payloads.

An 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 data payload sizes for full-speed control endpoints and only eight-byte maximum data payload sizes for low-speed control endpoints. No Host Controller is required to support larger or smaller maximum data payload sizes.

In order to determine the maximum packet size for the Default Control Pipe, the USB System Software reads the device descriptor. The host will read the first eight bytes of the device descriptor. The device always responds with at least these initial bytes in a single packet. After the host reads the initial part of the device descriptor, it is guaranteed to have read this default pipe's *wMaxPacketSize* field (byte 7 of the device descriptor). It will then allow the correct size for all subsequent transactions. For all other control endpoints, the maximum data payload size is known after configuration so that the USB System Software can ensure that no data payload will be sent to the endpoint that is larger than the supported size. The host will always use a maximum data payload size of at least eight bytes.

An endpoint must always transmit data payloads with a data field less than or equal to the endpoint's *wMaxPacketSize* (refer to Chapter 9). When a control 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.

The Data stage of a control transfer from an endpoint to the host is complete when the endpoint does one of the following:

- Has transferred exactly the amount of data specified during the Setup stage
- Transfers a packet with a payload size less than *wMaxPacketSize* or transfers a zero-length packet.

When a Data stage is complete, the Host Controller advances to the Status stage instead of continuing on with another data transaction. If the Host Controller does not advance to the Status stage when the Data stage is complete, the endpoint halts the pipe as was outlined in Section 5.3.2. If a larger-than-expected data payload is received from the endpoint, the IRP for the control transfer will be aborted/retired.

The Data stage of a control transfer from the host to an endpoint is complete when all of the data has been transferred. If the endpoint receives a larger-than-expected data payload from the host, it halts the pipe.

#### 5.5.4 Control Transfer Bus Access Constraints

Control transfers can be used by full-speed and low-speed USB devices.

An endpoint has no way to indicate a desired bus access frequency for a control pipe. The USB balances the bus access requirements of all control pipes and the specific IRPs that are pending to provide “best effort” delivery of data between client software and functions.

The USB requires that part of each frame be reserved to be available for use by control transfers as follows:

- If the control transfers that are attempted (in an implementation-dependent fashion) consume less than 10% of the frame time, the remaining time can be used to support bulk transfers (refer to Section 5.8).
- A control transfer that has been attempted and needs to be retried can be retried in the current or a future frame; i.e., it is not required to be retried in the same frame.
- If there are more control transfers than reserved time, but there is additional frame time that is not being used for isochronous or interrupt transfers, a Host Controller may move additional control transfers as they are available.
- If there are too many pending control transfers for the available frame time, control transfers are selected to be moved over the bus as appropriate.
- If there are control transfers pending for multiple endpoints, control transfers for the different endpoints are selected according to a fair access policy that is Host Controller implementation-dependent.
- A transaction of a control transfer that is frequently being retried should not be expected to consume an unfair share of the bus time.

These requirements allow control transfers between host and devices to be regularly moved over the bus with “best effort.”

The rate of control transfers to a particular endpoint can be varied by the USB System Software at its discretion. An endpoint and its client software cannot assume a specific rate of service for control transfers. A control endpoint may see zero or more transfers in a single frame. 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 control transfers are requested for other device endpoints.

The bus frequency and frame timing limit the maximum number of successful control transfers within a frame for any USB system to less than 29 full-speed eight-byte data payloads or less than four low-speed eight-byte data payloads. Table 5-1 lists information about different-sized full-speed control transfers and the maximum number of transfers possible in a frame. This table was generated assuming that there is one Data stage transaction and that the Data stage has a zero-length status phase. The table illustrates the possible power of two data payloads less than or equal to the allowable maximum data payload sizes. The table does not include the overhead associated with bit stuffing.

Table 5-1. Full-speed Control Transfer Limits

Protocol Overhead (45 bytes)		(9 SYNC bytes, 9 PID bytes, 6 Endpoint + CRC bytes, 6 CRC bytes, 8 Setup data bytes, and a 7-byte interpacket delay (EOP, etc.))			
Data Payload	Max Bandwidth (bytes/second)	Frame Bandwidth per Transfer	Max Transfers	Bytes Remaining	Bytes/Frame Useful Data
1	32000	3%	32	23	32
2	62000	3%	31	43	62
4	120000	3%	30	30	120
8	224000	4%	28	16	224
16	384000	4%	24	36	384
32	608000	5%	19	37	608
64	832000	7%	13	83	832
Max	1500000				1500

The 10% frame reservation for non-periodic transfers means that in a system with bus time fully allocated, all full-speed control transfers in the system contend for a nominal three control transfers per frame. Because the USB system uses control transfers for configuration purposes in addition to whatever other control transfers other client software may be requesting, a given software client and its function should not expect to be able to make use of this full bandwidth for its own control purposes. Host Controllers are also free to determine how the individual bus transactions for specific control transfers are moved over the bus within and across frames. An endpoint could see all bus transactions for a control transfer within the same frame or spread across several noncontiguous frames. A Host Controller, for various implementation reasons, may not be able to provide the theoretical maximum number of control transfers per frame.

Both full-speed and low-speed control transfers contend for the same available frame time. Low-speed control transfers simply take longer to transfer. Table 5-2 lists information about different-sized low-speed packets and the maximum number of packets possible in a frame. The table does not include the overhead associated with bit stuffing. For both speeds, because a control transfer is composed of several packets, the packets can be spread over several frames to spread the bus time required across several frames.



Table 5-2. Low-speed Control Transfer Limits

Protocol Overhead (46 bytes)					
Data Payload	Max Bandwidth (Approximate)	Frame Bandwidth per Transfer	Max Transfers	Bytes Remaining	Bytes/Frame Useful Data
1	3000	25%	3	46	3
2	6000	26%	3	43	6
4	12000	27%	3	37	12
8	24000	29%	3	25	24
Max	187500				187

### 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 5.10 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 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 System Software does not adjust the maximum data payload size for an isochronous pipe as is the case for a control pipe. An isochronous pipe can simply either be supported or not supported in a given USB system configuration.

The USB limits the maximum data payload size to 1,023 bytes for each isochronous pipe. Table 5-3 lists information about different-sized isochronous transactions and the maximum number of transactions possible in a frame. The table does not include the overhead associated with bit stuffing.

Table 5-3. 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	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

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.

#### 5.6.4 Isochronous Transfer Bus Access Constraints

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

The USB requires that no more than 90% of any frame be allocated for periodic (isochronous and interrupt) transfers.

An endpoint for an isochronous pipe does not include information about bus access frequency. All isochronous pipes normally move exactly one data packet each frame (i.e., every 1ms). Errors on the bus or delays in operating system scheduling of client software can result in no packet being transferred for a 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 two SOF tokens without an intervening data packet for an isochronous endpoint.

The bus frequency and frame timing limit the maximum number of successful isochronous transactions within a frame for any USB system to less than 151 full-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 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 frame. Also, a receiver can determine how much data was lost. Section 5.10 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 frame of the transfer. 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 small amounts of 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. 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 have support for up to 64-byte maximum data payload sizes for full-speed interrupt endpoints and eight bytes or less maximum data payload sizes for low-speed interrupt endpoints. No Host Controller is required to support larger maximum data payload sizes.

## Universal Serial Bus Specification Revision 1.1

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. The USB System Software does not adjust the bus time made available to an interrupt pipe as is the case for a control pipe. An interrupt pipe can simply either be supported or not supported in a given USB system configuration. 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.

### 5.7.4 Interrupt Transfer Bus Access Constraints

Interrupt transfers can be used by full-speed and low-speed devices.

The USB requires that no more than 90% of any frame be allocated for periodic (isochronous and interrupt) transfers.

The bus frequency and frame timing limit the maximum number of successful interrupt transactions within a frame for any USB system to less than 108 full-speed one-byte data payloads or 14 low-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 frame.

Table 5-4 lists information about different sized full-speed interrupt transactions and the maximum number of transactions possible in a frame. Table 5-5 lists similar information for low-speed interrupt transactions. The tables do not include the overhead associated with bit stuffing.

## Universal Serial Bus Specification Revision 1.1

**Table 5-4. Full-speed Interrupt Transaction Limits**

<b>Protocol Overhead (13 bytes)</b>		<b>(3 SYNC bytes, 3 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 3-byte interpacket delay)</b>			
<b>Data Payload</b>	<b>Max Bandwidth</b>	<b>Frame Bandwidth per Transfer</b>	<b>Max Transfers</b>	<b>Bytes Remaining</b>	<b>Bytes/Frame Useful Data</b>
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

An endpoint for an interrupt pipe specifies its desired bus access period. A full-speed endpoint can specify a desired period from 1ms to 255ms. Low-speed endpoints are limited to specifying only 10ms to 255ms. 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 (1ms). 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.

Table 5-5. Low-speed Interrupt Transaction Limits

Protocol Overhead (13 bytes)					
Data Payload	Max Bandwidth (Approximate)	Frame Bandwidth per Transfer	Max Transfers	Bytes Remaining	Bytes/Frame Useful Data
1	13000	7%	13	5	13
2	24000	8%	12	7	24
4	44000	9%	11	0	44
8	64000	11%	8	19	64
Max	187500				187

Interrupt transfers are moved over the USB by accessing an interrupt endpoint every 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. 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 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

Bulk transfers can be used only by full-speed devices.

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 bus time made available for bulk transfers to a particular endpoint can be varied by the USB System Software at its discretion. An endpoint and its client software cannot assume a specific rate of service for bulk



## Universal Serial Bus Specification Revision 1.1

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.

The bus frequency and frame timing limit the maximum number of successful bulk transactions within a frame for any USB system to less than 72 eight-byte data payloads. Table 5-6 lists information about different-sized bulk transactions and the maximum number of transactions possible in a frame. The table does not include the overhead associated with bit stuffing.

**Table 5-6. Bulk Transaction Limits**

<b>Protocol Overhead (13 bytes)</b>		(3 SYNC bytes, 3 PID bytes, 2 Endpoint + CRC bytes, 2 CRC bytes, and a 3-byte interpacket delay)			
<b>Data Payload</b>	<b>Max Bandwidth (bytes/second)</b>	<b>Frame Bandwidth per Transfer</b>	<b>Max Transfers</b>	<b>Bytes Remaining</b>	<b>Bytes/Frame Useful Data</b>
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

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

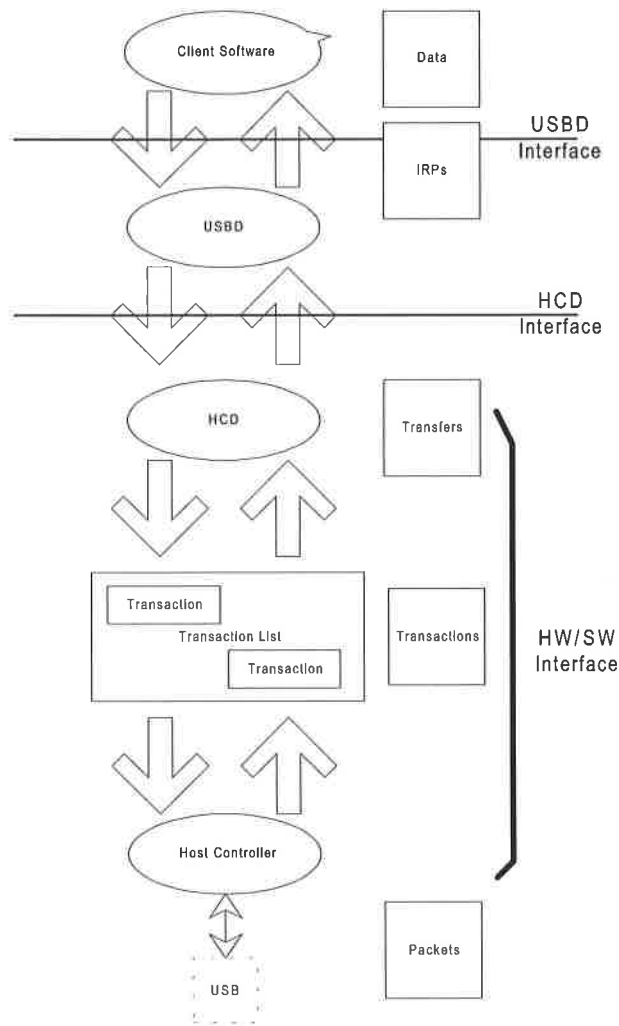
### 5.9.1 Transfer Management

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 5-10 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.

## Universal Serial Bus Specification Revision 1.1



**Figure 5-10. USB Information Conversion From Client Software to Bus**

### 5.9.1.1 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 USB interface.

Some clients may manipulate USB functions via other device class interfaces defined by the operating system and may themselves not make direct USB calls. However, there is always some lowest level client that makes USB calls to pass IRPs to the USB. 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 USB to manipulate the driver's USB function.

## Universal Serial Bus Specification Revision 1.1

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 USBDB interface is defined in Chapter 10.

### 5.9.1.2 USB Driver

The Universal Serial Bus Driver (USBDB) 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 USBDB is involved to ensure that the desired device configuration can be accommodated on the bus. The USBDB 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 USBDB 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 USBDB 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, an bandwidth allocation can be released without changing the device configuration.

The configuration aspects of the USBDB 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.

### 5.9.1.3 Host Controller Driver

The Host Controller Driver (HCD) is responsible for tracking the IRPs in progress and ensuring that USB bandwidth and 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.

### 5.9.1.4 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.

### 5.9.1.5 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.

## Universal Serial Bus Specification Revision 1.1

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.

### 5.9.2 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 (USBSD 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 5-11 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.

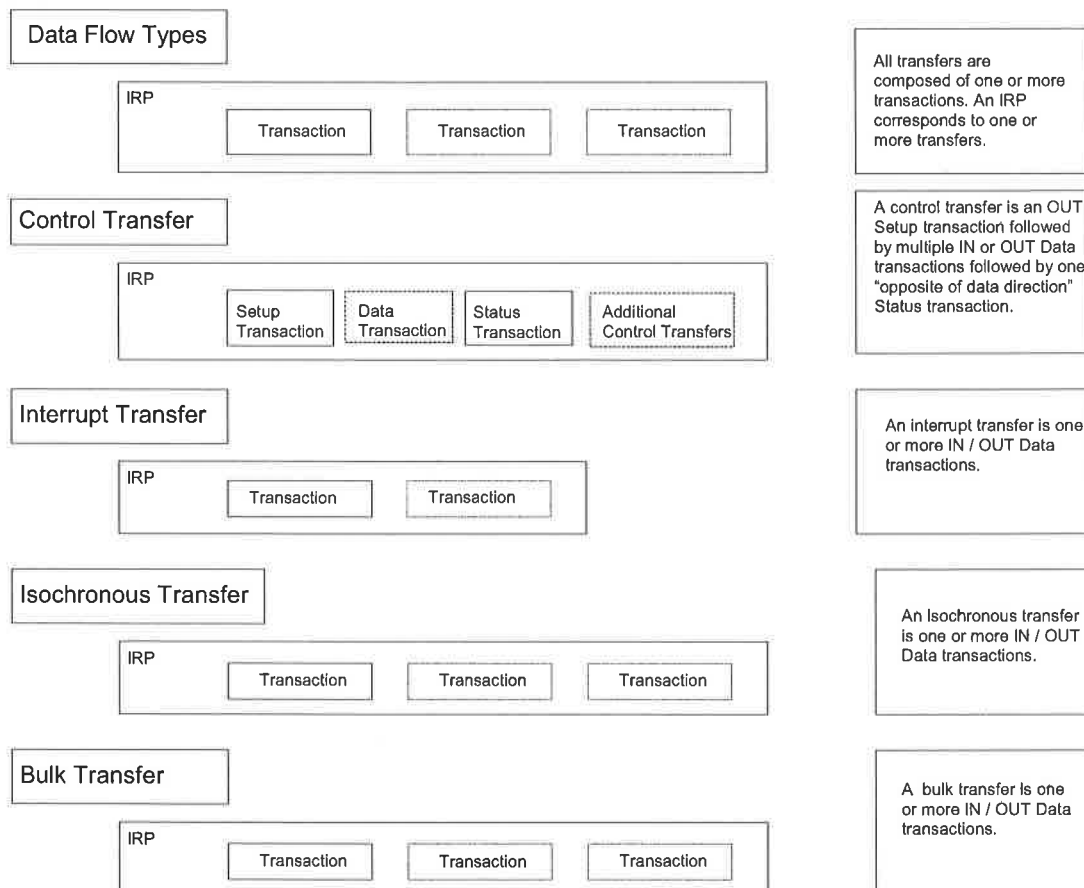


Figure 5-11. Transfers for Communication Flows

## Universal Serial Bus Specification Revision 1.1

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 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 5-12 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 Frame 1, while moving the second transaction of each IRP in opposite order somewhere in 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 frame. Functions cannot depend on seeing transactions within an IRP back-to-back within a frame nor should they depend on not seeing transactions back-to-back within a frame.

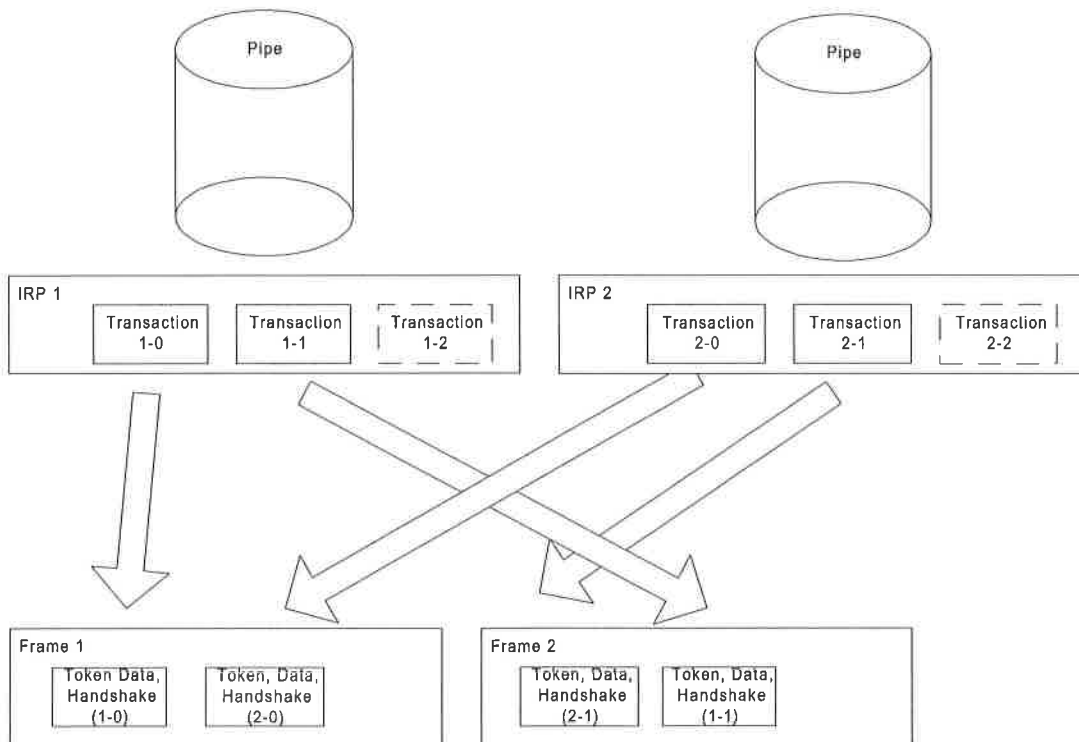


Figure 5-12. Arrangement of IRPs to Transactions/Frames

### 5.9.3 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 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 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 $(1.1667 * 8 * \text{Data\_bc})$

#### Full-speed (Input)

Non-Isynchronous Transfer (Handshake Included)  
 $= 9107 + (83.54 * \text{Floor}(3.167 + \text{BitStuffTime}(\text{Data\_bc}))) + \text{Host\_Delay}$

Isynchronous Transfer (No Handshake)  
 $= 7268 + (83.54 * \text{Floor}(3.167 + \text{BitStuffTime}(\text{Data\_bc}))) + \text{Host\_Delay}$

#### Full-speed (Output)

Non-Isynchronous Transfer (Handshake Included)  
 $= 9107 + (83.54 * \text{Floor}(3.167 + \text{BitStuffTime}(\text{Data\_bc}))) + \text{Host\_Delay}$

Isynchronous Transfer (No Handshake)  
 $= 6265 + (83.54 * \text{Floor}(3.167 + \text{BitStuffTime}(\text{Data\_bc}))) + \text{Host\_Delay}$

#### Low-speed (Input)

$= 64060 + (2 * \text{Hub\_LS\_Setup}) + (676.67 * \text{Floor}(3.167 + \text{BitStuffTime}(\text{Data\_bc}))) + \text{Host\_Delay}$

#### Low-speed (Output)

$= 64107 + (2 * \text{Hub\_LS\_Setup}) + (667.0 * \text{Floor}(3.167 + \text{BitStuffTime}(\text{Data\_bc}))) + \text{Host\_Delay}$

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 5.9.5.

The `Host_Delay` term in the equations is Host Controller- and system-dependent and allows for additional time a Host Controller 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 USB D 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.

### 5.9.4 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 the USB 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 frames into the future they allow a transaction to be requested.

For isochronous pipes, Section 5.10.4 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.

### 5.9.5 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 frame time versus what the 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 frame time to reuse reclaimed bandwidth.

## 5.10 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.

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



## Universal Serial Bus Specification Revision 1.1

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 Frame Clock-to-function Clock Synchronization Options:** How the USB frame clock can relate to a function clock
- **SOF Tracking:** Responsibilities and opportunities of isochronous endpoints with respect to the SOF token and USB 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.

### 5.10.1 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 8kHz mono microphone connected through a mixer driver that sends the input data stream to 44kHz 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 5-13 illustrates this example.

Universal Serial Bus Specification Revision 1.1

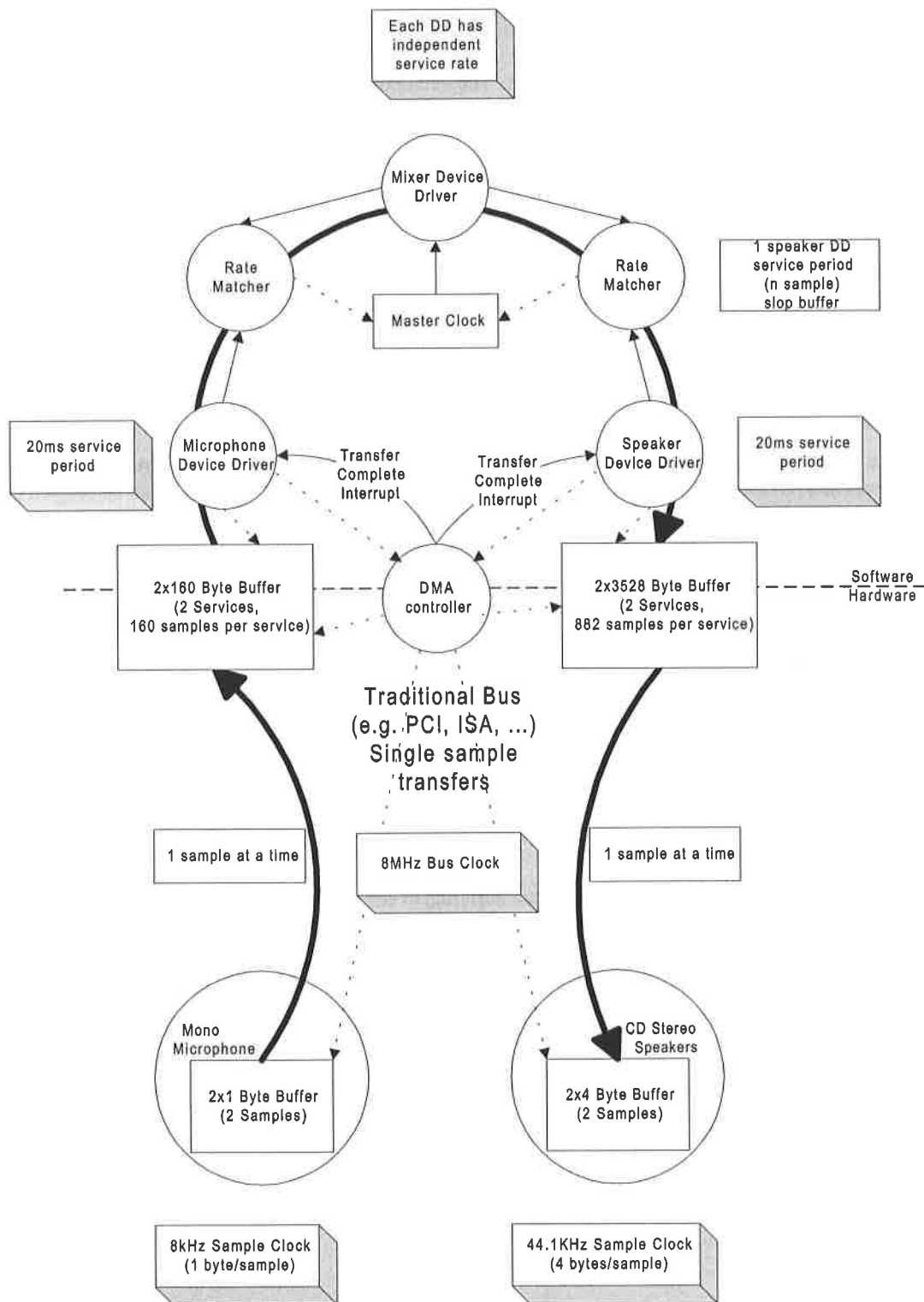


Figure 5-13. Non-USB Isochronous Example

## Universal Serial Bus Specification Revision 1.1

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 20ms. 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 (20ms 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 20ms. The output device consumes 3,528 bytes (882 samples) every 20ms 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.

### 5.10.2 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 1.000ms period (1kHz frequency) and is indicated by the rate of SOF packets on the bus. This clock is somewhat equivalent to the 8MHz 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 5-14 presents an example for a reasonable USB clock environment equivalent to the non-USB example in Figure 5-13.

Universal Serial Bus Specification Revision 1.1

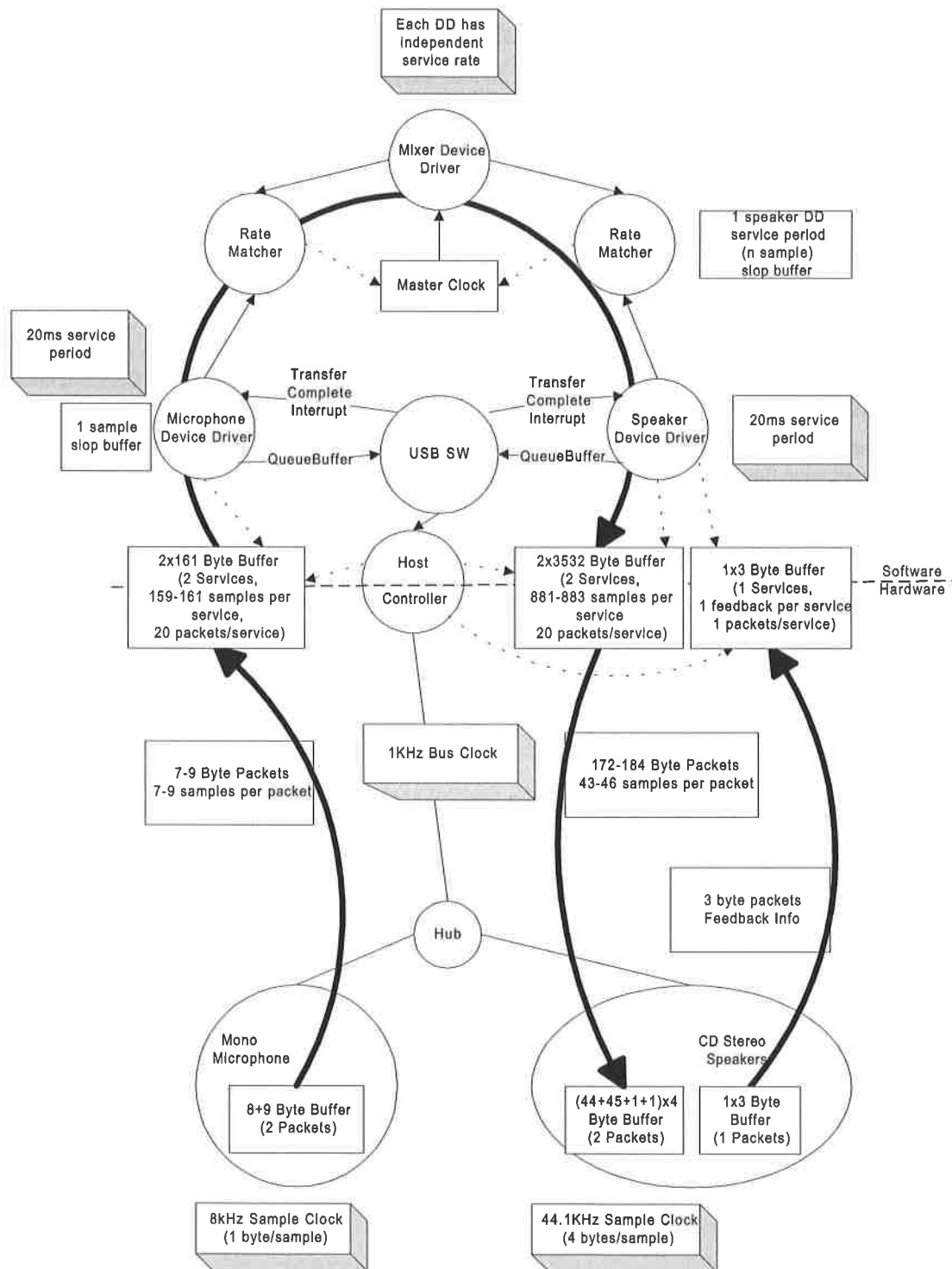


Figure 5-14. USB Isochronous Application

## Universal Serial Bus Specification Revision 1.1

Figure 5-14 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 5-14 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.

### 5.10.3 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 single packet of data is moved exactly once per frame and the frame clock is reasonably precise. Providing the software with this information allows it to adjust the amount of data it processes to the actual frame time that has passed.

### 5.10.4 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.

## Universal Serial Bus Specification Revision 1.1

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.

### 5.10.4.1 Synchronization Type

Three distinct synchronization types are defined. Table 5-7 presents an overview of endpoint synchronization characteristics for both source and sink endpoints. The types are presented in order of increasing capability.

**Table 5-7. Synchronization Characteristics**

	Source	Sink
<b>Asynchronous</b>	Free running $F_s$ Provides implicit feedforward (data stream)	Free running $F_s$ Provides explicit feedback (interrupt pipe)
<b>Synchronous</b>	$F_s$ locked to SOF Uses implicit feedback (SOF)	$F_s$ locked to SOF Uses implicit feedback (SOF)
<b>Adaptive</b>	$F_s$ locked to sink Uses explicit feedback (control pipe)	$F_s$ locked to data flow Uses implicit feedforward (data stream)

#### 5.10.4.1.1 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 (32kHz, 44.1kHz, 48kHz, ...), 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 frame. Asynchronous sink endpoints must provide explicit feedback information to an adaptive driver (refer to Section 5.10.4.2).

## Universal Serial Bus Specification Revision 1.1

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.

Another case arises when there are two or more devices present on the USB that need to have mastership control over SOF generation in order to operate as synchronous devices. This could happen if there were two telephony devices, each locked to a different external clock. One telephony device could be digitally connected to a Private Branch Exchange (PBX) that is not synchronized to the ISDN. The other device could be connected directly to the ISDN. Each device will source or sink data to/from the network side at an externally driven rate. Because only one of the devices can take mastership over the SOF, the other will sink or source data at a rate that is asynchronous to the SOF. This example indicates that every device capable of SOF mastership may be forced to operate as an asynchronous device.

### 5.10.4.1.2 Synchronous

Synchronous endpoints can have their clock system (their notion of time) controlled externally through SOF synchronization. These endpoints must be doing one of the following:

- Slaving their sample clock to the 1ms SOF tick (by means of a programmable PLL).
- Controlling the rate of USB SOF generation so that their data rate becomes automatically locked to SOF. In case these endpoints are not granted SOF mastership, they must degenerate to the asynchronous mode of operation (refer to the asynchronous example).

Synchronous endpoints may source or sink isochronous data streams at either a fixed data rate (single-frequency endpoints), a limited number of data rates (32kHz, 44.1kHz, 48kHz, ...), 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 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 frame. Another possibility is a 64kb/s bit-stream from an ISDN "modem." If the USB SOF generation is locked to the PSTN clock (perhaps through the same ISDN device), the data generation will also be locked to SOF and the endpoint will produce a stable 64kb/s data stream, referenced to the SOF time notion.

### 5.10.4.1.3 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 5.10.4.2) to the source, which allows the source to know the desired data rate of the sink. Adaptive endpoints can communicate with all types of sink endpoints. 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., 8kHz), select between several programmable or auto-detecting data rates (32kHz, 44.1kHz, 48kHz, ...), or may be within one or more ranges (e.g., 5kHz to 12kHz or 44kHz to 49kHz). 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 44.1kHz but can be anything within the operating range of the SRC. Adaptive sinks include such endpoints as high-end digital speakers, headsets, etc.



#### 5.10.4.2 Feedback

An asynchronous sink provides feedback to an adaptive source by indicating accurately what its desired data rate ( $F_f$ ) is, relative to the USB SOF frequency. The required data rate is accurate 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.

The  $F_f$  value consists of a fractional part, in order to get the required resolution with 1kHz frames, and an integer part, which gives the minimum number of samples per frame. Ten bits are required to resolve one sample within a 1kHz frame frequency ( $1000 / 2^{10} = 0.98$ ). This is a ten-bit fraction, represented in unsigned fixed binary point 0.10 format. The integer part needs ten bits ( $2^{10} = 1024$ ) to encode up to 1,023 one-byte samples per frame. The ten-bit integer is represented in unsigned fixed binary point 10.0 format. The combined  $F_f$  value can be coded in unsigned fixed binary point 10.10 format, which fits into three bytes (24 bits). 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. The bit and byte ordering follows the definitions of other multi-byte fields contained in Chapter 8.

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

The sink can determine  $F_f$  by counting cycles of a clock with a frequency of  $F_s * 2^P$  for a period of  $2^{(10-P)}$  frames, where  $P$  is an integer.  $P$  is practically bound to be in the range  $[0,10]$  because there is no point in using a clock slower than  $F_s$ , and no point in trying to update more than once a frame. The counter is read into  $F_f$  and reset every  $2^{(10-P)}$  frames. As long as no clock cycles are skipped, the count will be accurate over the long term. An endpoint needs to implement only the number of counter bits that it requires for its maximum  $F_f$ .

A digital telephony endpoint, for example, will usually derive its 8kHz  $F_s$  by dividing down the 64kHz clock ( $P=3$ ) which it uses to serialize the data stream. The 64kHz clock phase can also give an additional one bit of accuracy, effectively giving  $P=4$ . This would give  $F_f$  updates every  $2^{(10-4)} = 64$  frames. A 13-bit counter would be required to obtain  $F_f$ , with three bits for eight samples per frame, and ten bits for the fractional part. The 13 bits would provide a 3.10 field within the 10.14  $F_f$  value, with the remaining bits set to zero.

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

- $P$  should be in the range  $[1,9]$ .
- 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 ten 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  $2^{(10-P)}$  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.

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.

An adaptive source may obtain the sink data rate information from an adaptive sink that is locked to the same clock as the sink, as would be the case for a two-way speech connection. In this case, the feedback pipe is not needed.

#### **5.10.4.3 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 5-15 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 the USB 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.

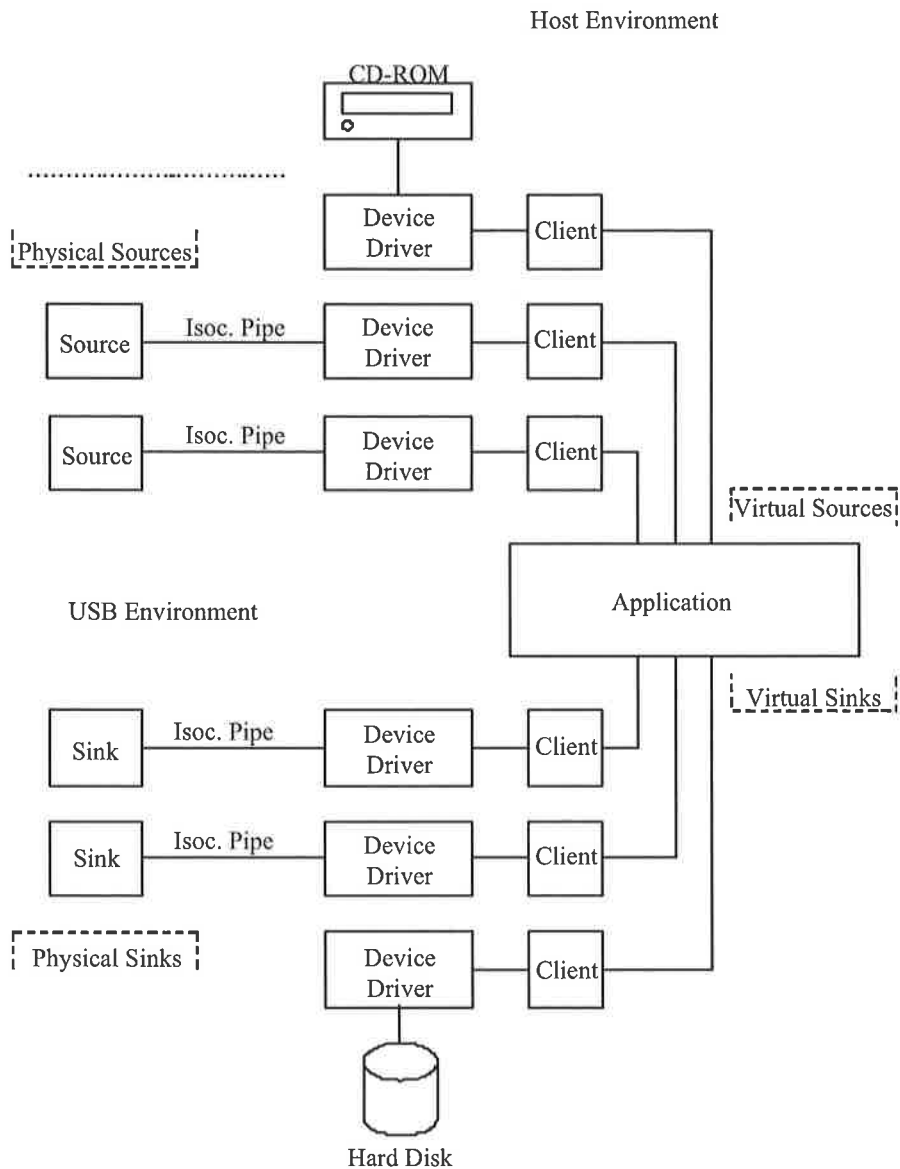


Figure 5-15. Example Source/Sink Connectivity

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.”

## Universal Serial Bus Specification Revision 1.1

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 44.1kHz 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., 10ms) and nominal sample rate of the data (44.1kHz) 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 5-8 indicates at the intersections what actions the application must perform to connect a source endpoint to a sink endpoint.

## Universal Serial Bus Specification Revision 1.1

**Table 5-8. Connection Requirements**

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

**Notes:**

1. Asynchronous RA in the application.  $F_{sj}$  is determined by the source, using the feedforward information embedded in the data stream.  $F_{s0}$  is determined by the sink, based on feedback information from the sink. If nominally  $F_{sj} = F_{s0}$ , 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.  $F_{sj}$  is determined by the source but locked to SOF.  $F_{s0}$  is determined by the sink, based on feedback information from the sink. If nominally  $F_{sj} = F_{s0}$ , 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  $F_{s0}$  falls within the locking range of the adaptive source, a feedthrough connection can be established.  $F_{sj} = F_{s0}$  and both are determined by the asynchronous sink, based on feedback information from the sink. If  $F_{s0}$  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.  $F_{sj}$  is determined by the source.  $F_{s0}$  is determined by the sink and locked to SOF. If nominally  $F_{sj} = F_{s0}$ , 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.  $F_{sj}$  is determined by the source and locked to SOF.  $F_{s0}$  is determined by the sink and locked to SOF. If  $F_{sj} = F_{s0}$ , 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  $F_{sj}$  falls within the locking range of the adaptive sink, a feedthrough connection can be established.  $F_{sj} = F_{s0}$  and both are determined by and locked to the source. If  $F_{sj}$  falls outside the locking range of the adaptive sink, synchronous RA is done in the host to provide an  $F_{s0}$  that is within the locking range of the adaptive sink.
8. If  $F_{sj}$  falls within the locking range of the adaptive sink, a feedthrough connection can be established.  $F_{s0} = F_{sj}$  and both are determined by the source and locked to SOF. If  $F_{sj}$  falls outside the locking range of the adaptive sink, synchronous RA is done in the host to provide an  $F_{s0}$  that is within the locking range of the adaptive sink.
9. The application will use feedback control to set  $F_{s0}$  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.

### 5.10.4.3.1 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.

### 5.10.4.3.2 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.

## 5.10.5 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 frame to frame.

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

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

Figure 5-16 shows the time sequence for a function-to-host transfer (IN process). Data  $D_0$  is accumulated during frame  $F_i$  at time  $\tau_i$ , and transmitted to the host during frame  $F_{i+1}$ . Similarly, for a host-to-function transfer (OUT process), data  $D_0$  is received by the endpoint during frame  $F_{i+1}$  and processed during frame  $F_{i+2}$ .

## Universal Serial Bus Specification Revision 1.1

Time:	$T_i$	$T_{i+1}$	$T_{i+2}$	$T_{i+3}$	...	$T_m$	$T_{m+1}$	...
Frame:	$F_i$	$F_{i+1}$	$F_{i+2}$	$F_{i+3}$	...	$F_m$	$F_{m+1}$	...
Data on Bus:		$D_0$	$D_1$	$D_2$	...	$D_0$	$D_1$	...
OUT Process:			$D_0$	$D_1$	...		$D_0$	...
IN Process	$D_0$	$D_1$	...		$D_0$	...		

**Figure 5-16. Data Prebuffering**

### 5.10.6 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 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 frame. The USB requires that when an isochronous transfer is presented to the Host Controller, it identifies the frame number for the first frame. The Host Controller must not transmit the first transaction before the indicated frame number. Each subsequent transaction in the IRP must be transmitted in succeeding frames. If there are no transactions pending for the current frame, then the Host Controller must not transmit anything for an isochronous pipe. If the indicated frame number has passed, the Host Controller must skip (i.e., not transmit) all transactions until the one corresponding to the current frame is reached.

### 5.10.7 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: exactly one packet per frame, SOF, CRC, and bus transaction timeout.

- Isochronous transfers require exactly one data transaction every frame for normal operation. The USB does not dictate what data is transmitted in each frame. The data transmitter/source determines specifically what data to provide. This regular data-per-frame provides a framework that is fundamental to detecting missing data errors. 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 frame is preceded by an SOF and a receiver can see SOFs on the bus, a receiver can determine that its expected transaction 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 5.10.6.

## Universal Serial Bus Specification Revision 1.1

- 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 frame, then the size is always a known constant. However, in some cases the data size can vary from frame to frame. In this case, the receiver and transmitter have an implementation-dependent mechanism to determine the size of the lost packet.

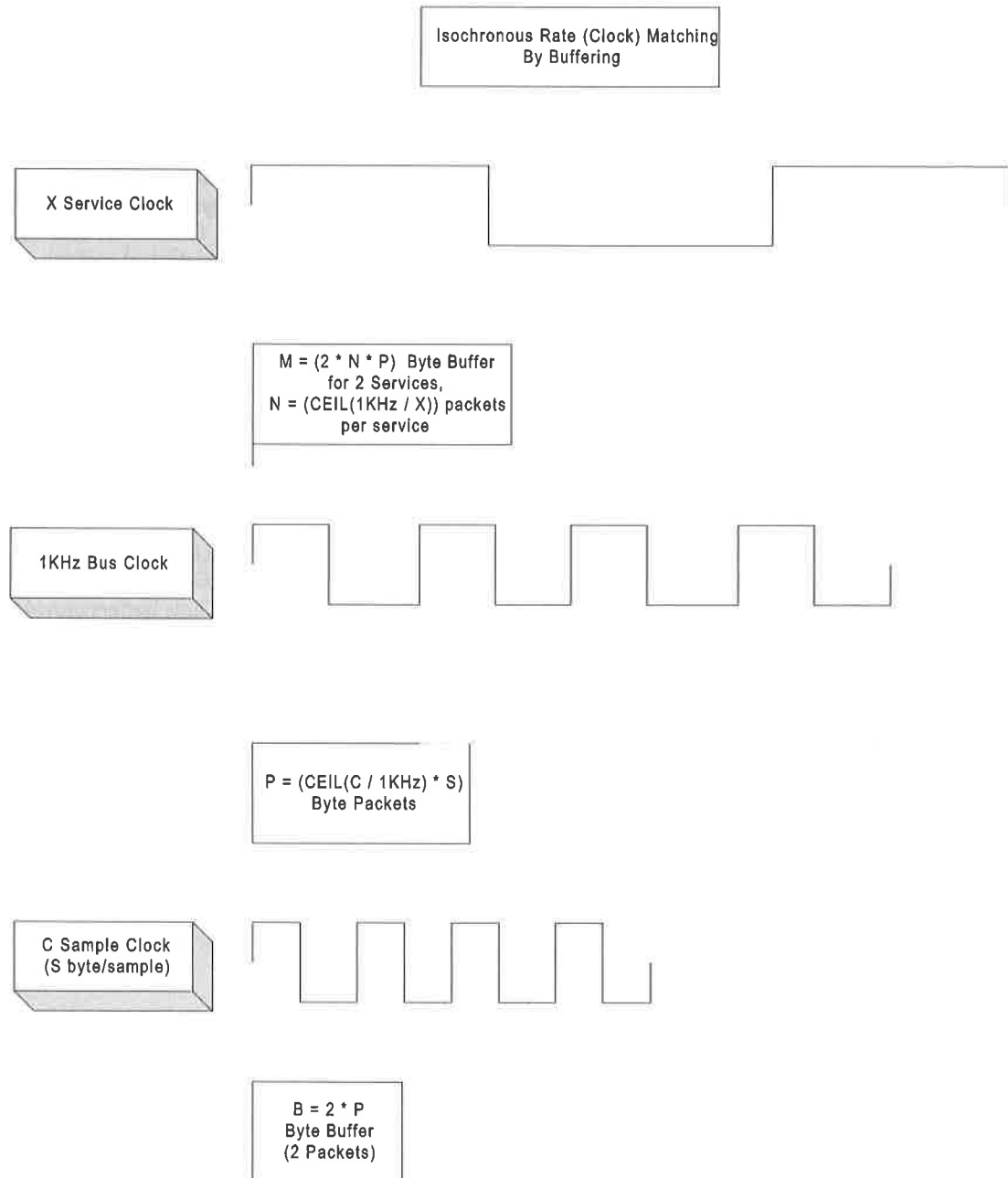
In summary, whether a transaction is actually moved successfully over the bus or not, the transmitter and receiver always advance their data/buffer streams one transaction per frame to keep data-per-time synchronization. The detailed mechanisms described above allow detection, tracking, and reporting 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.

### 5.10.8 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 5-17 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 allow a device and client software design time-determined service clock rate (variable  $X$ ), sample clock rate (variable  $C$ ), and sample size (variable  $S$ ). The USB allows only one transaction per bus clock. 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 5-14 shows actual buffer, packet, and clock values for a typical isochronous example.



## Universal Serial Bus Specification Revision 1.1



**Figure 5-17. 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 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 two speeds. Full-speed, 12 Mb/s, requires the use of a shielded cable with two power conductors and twisted pair signal conductors. Low-speed, 1.5 Mb/s, relaxes the cable requirement. Low-speed cable does not require shielding or 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 insure proper end user connectivity. The “A” connector is the principle means of connecting USB devices. All USB devices must have an “A” connector. 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.

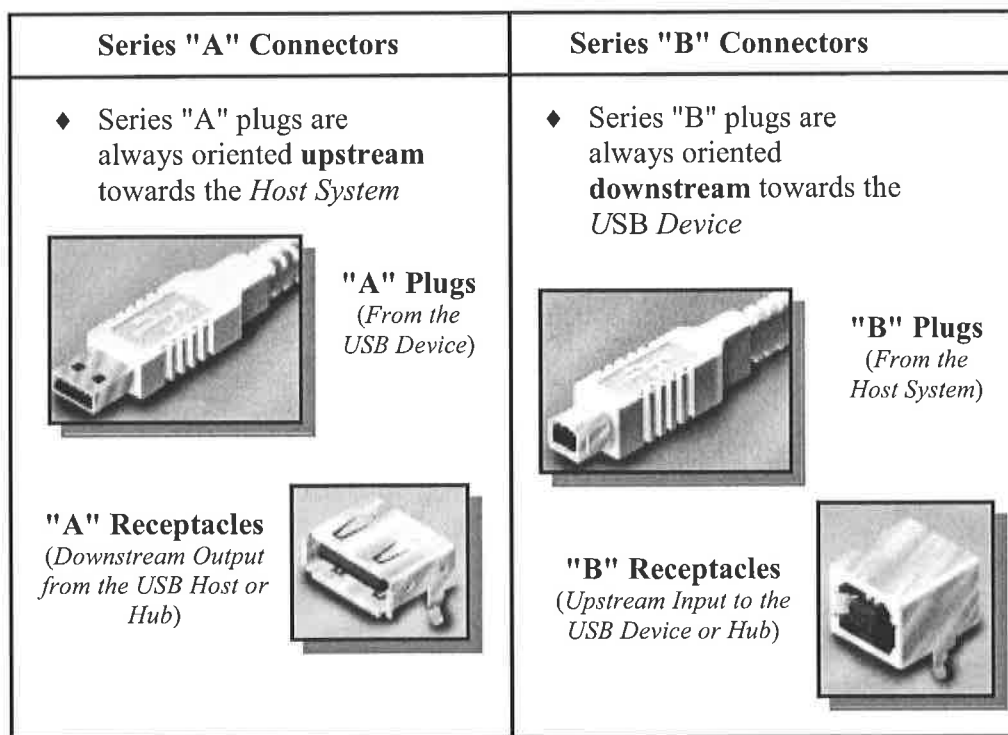


Figure 6-1. Keyed Connector Protocol

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

## Universal Serial Bus Specification Revision 1.1

- 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.

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

Low-speed cable does not require twisted signaling conductors or the overall shield.

### 6.4 Cable Assembly

This specification describes three USB cable assemblies. Detachable cable, Full-speed captive cable, and Low-speed captive cable.

The color used for the cable assembly is vendor specific, recommended colors are White, Grey, or Black.

#### 6.4.1 Detachable Cable Assemblies

Full-speed devices can utilize the “B” connector. This allows the device to have a 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. Detachable cable assemblies may be used only on Full-speed devices. Using a Full-speed detachable cable on a Low-speed device may exceed the maximum Low-speed cable length.

Figure 6-2 illustrates a detachable cable assembly.

Universal Serial Bus Specification Revision 1.1

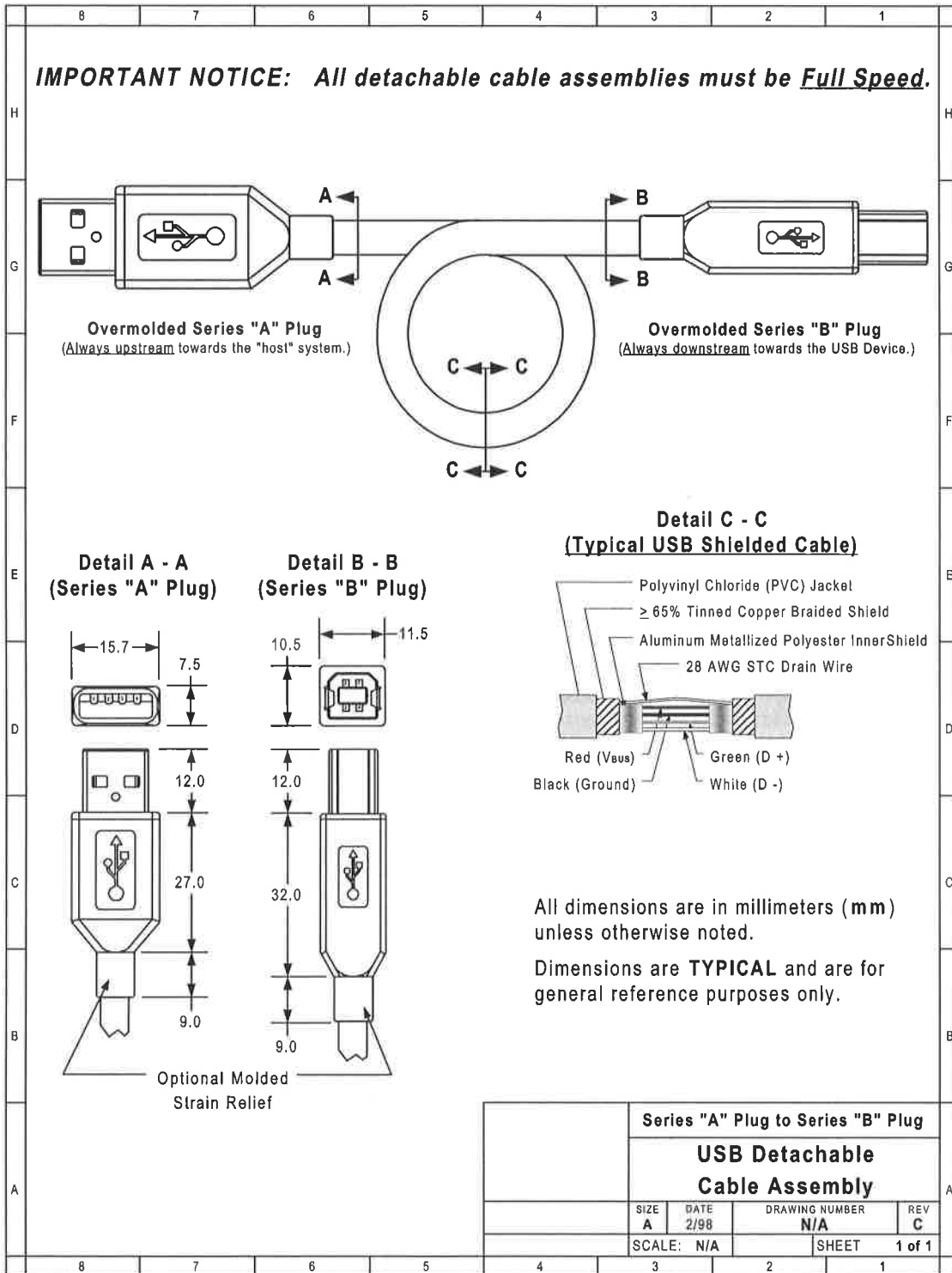


Figure 6-2. USB Detachable Cable Assembly

## Universal Serial Bus Specification Revision 1.1

Detachable Cables 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 Full-speed.
- The cable impedance must match the impedance of the 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. Refer to Section 7.1.17 for details.
- The maximum allowable cable length determined by the cable propagation delay. The USB utilizes an unterminated transmission scheme. Exceeding this limit will cause signaling reflections to interfere with data transmission. Refer to Section 7.1.14 for details.
- Differences in propagation delay between the two signal conductors must be minimized. Refer to Chapter 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 detachable cables, the VBUS requirement is the same as the GND lead.

### 6.4.2 Full-speed Captive Cable Assemblies

Full-speed captive cable assemblies may be used with either Full-speed or Low-speed devices. Assemblies are considered captive if they are provided with a vendor-specific disconnect means. When using a Full-speed captive cable on a Low-speed device the cable must meet all Low-speed requirements.

Figure 6-3 illustrates a Full-speed cable assembly.

Universal Serial Bus Specification Revision 1.1

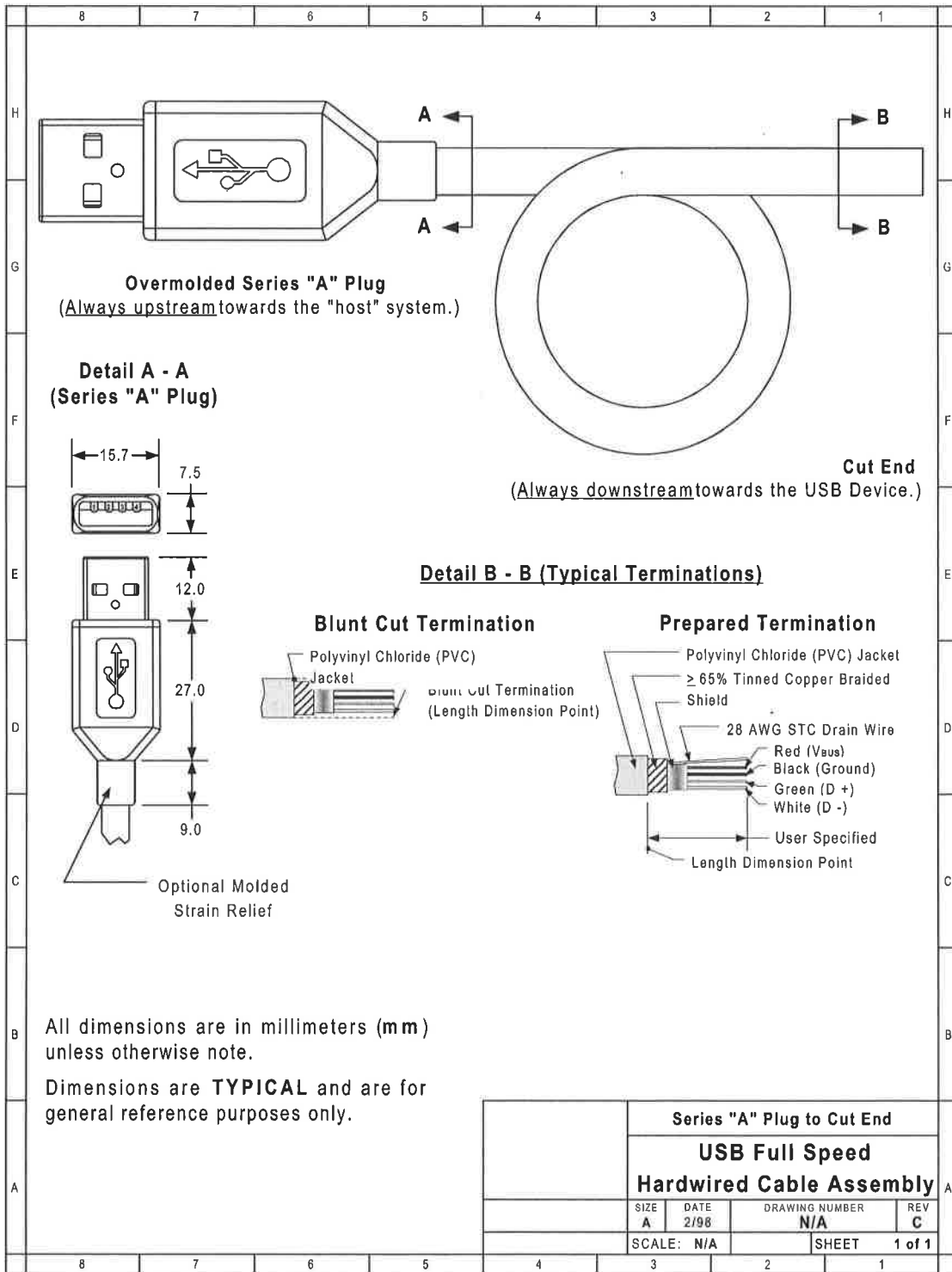


Figure 6-3. USB Full-speed Hardwired Cable Assembly

## Universal Serial Bus Specification Revision 1.1

Full-speed Captive Cables 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 Full-speed.
- The cable impedance must match the impedance of the Full-speed drivers. The drivers are characterized to drive specific cable impedance. Refer to Section 7.1.1 for details.
- The maximum cable length is determined by the attenuation of the signal pair. Refer to Section 7.1.17 for details.
- The maximum cable length is determined by the propagation delay through the cable. The USB utilizes an unterminated transmission scheme, exceeding this limit will cause signaling reflections to interfere with data transmission. Refer to Section 7.1.14 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 disconnect means. Low-speed cable may only be used on Low-speed devices.

Figure 6-4 illustrates a Low-speed cable assembly.

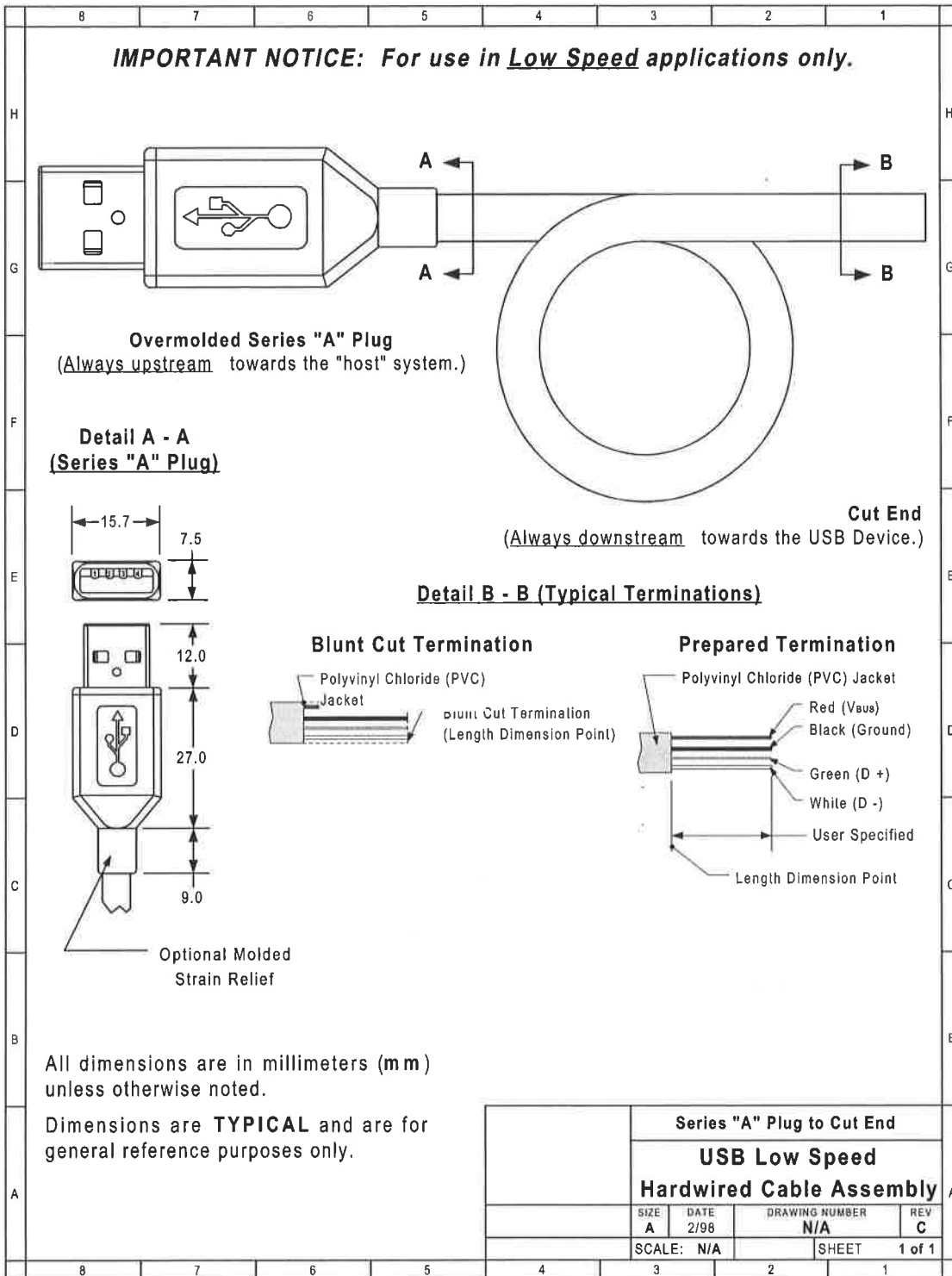


Figure 6-4. USB Low-speed Hardwired Cable Assembly



## Universal Serial Bus Specification Revision 1.1

Low-speed Captive Cables 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 determined by the rise and fall times of Low-speed signaling. This forces Low-speed cable to be significantly shorter than 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.

Non-acceptable cables may work in some situations but they cannot be guaranteed to work in all instances.

- ***Extension cable***  
Cables that provide 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 with both ends terminated in either Series “A” plugs or Series “B” receptacles. This cable 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 busses.
- ***Low-speed detachable cable***  
Detachable cables must be Full-speed. Low-speed devices are prohibited from using detachable cables. Detachable cable is Full-speed rated, using a long 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

Universal Serial Bus Specification Revision 1.1

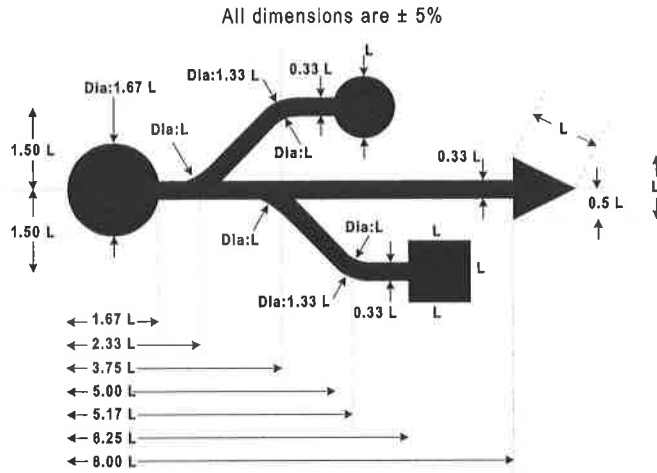


Figure 6-5. USB Icon

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.

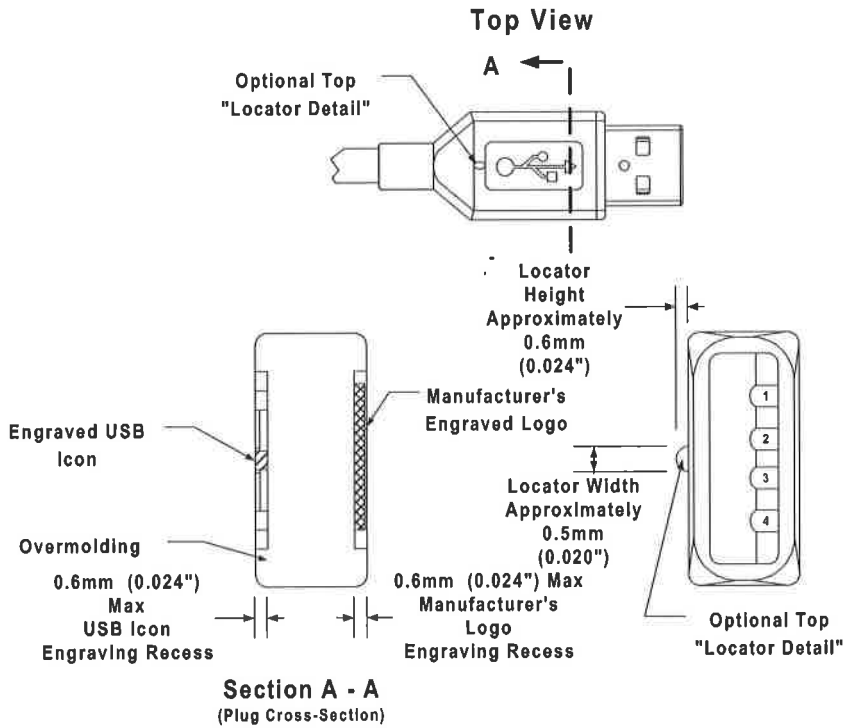


Figure 6-6. Typical USB Plug Orientation

### 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.

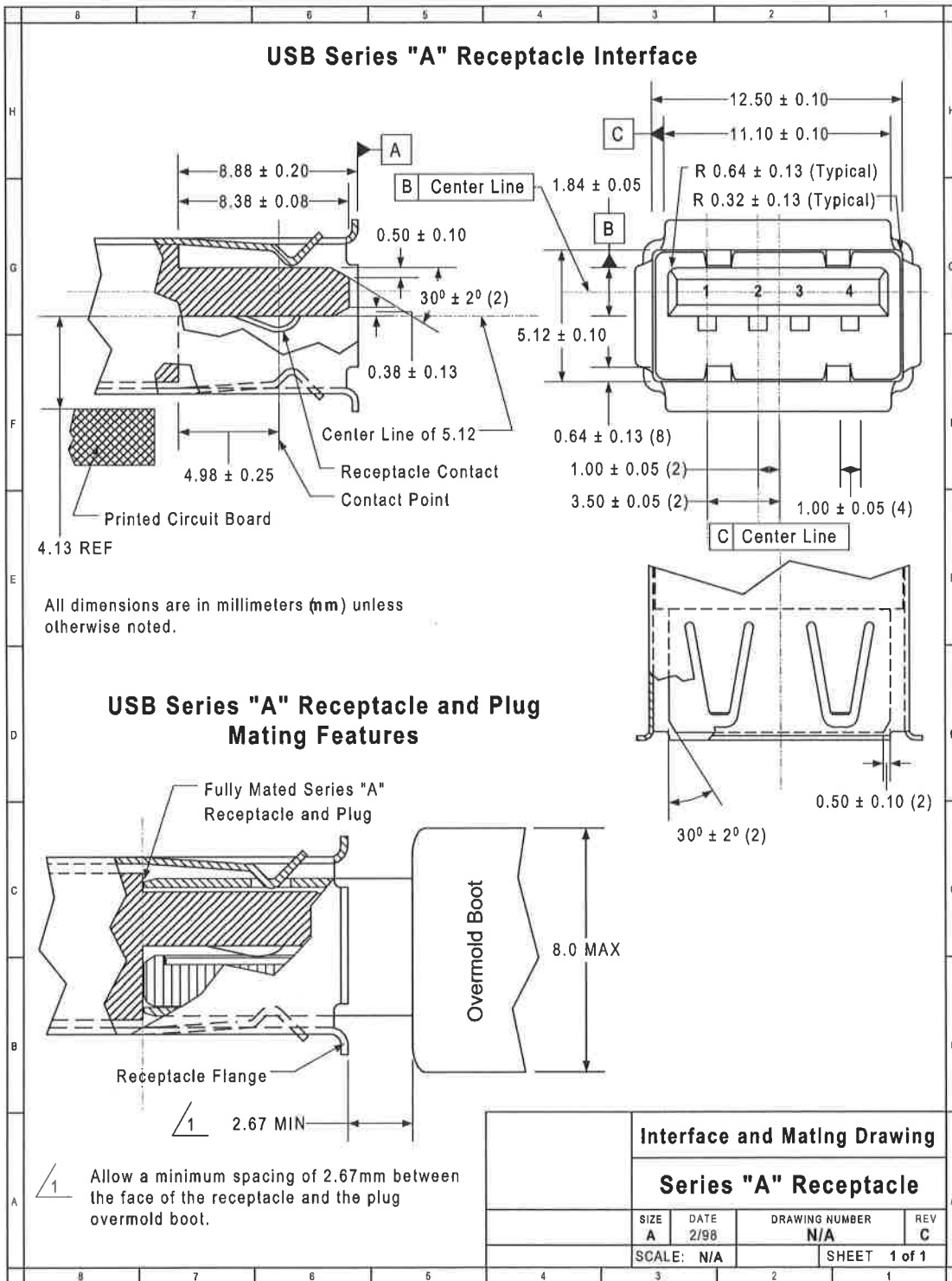


Figure 6-7. USB Series "A" Receptacle Interface and Mating Drawing

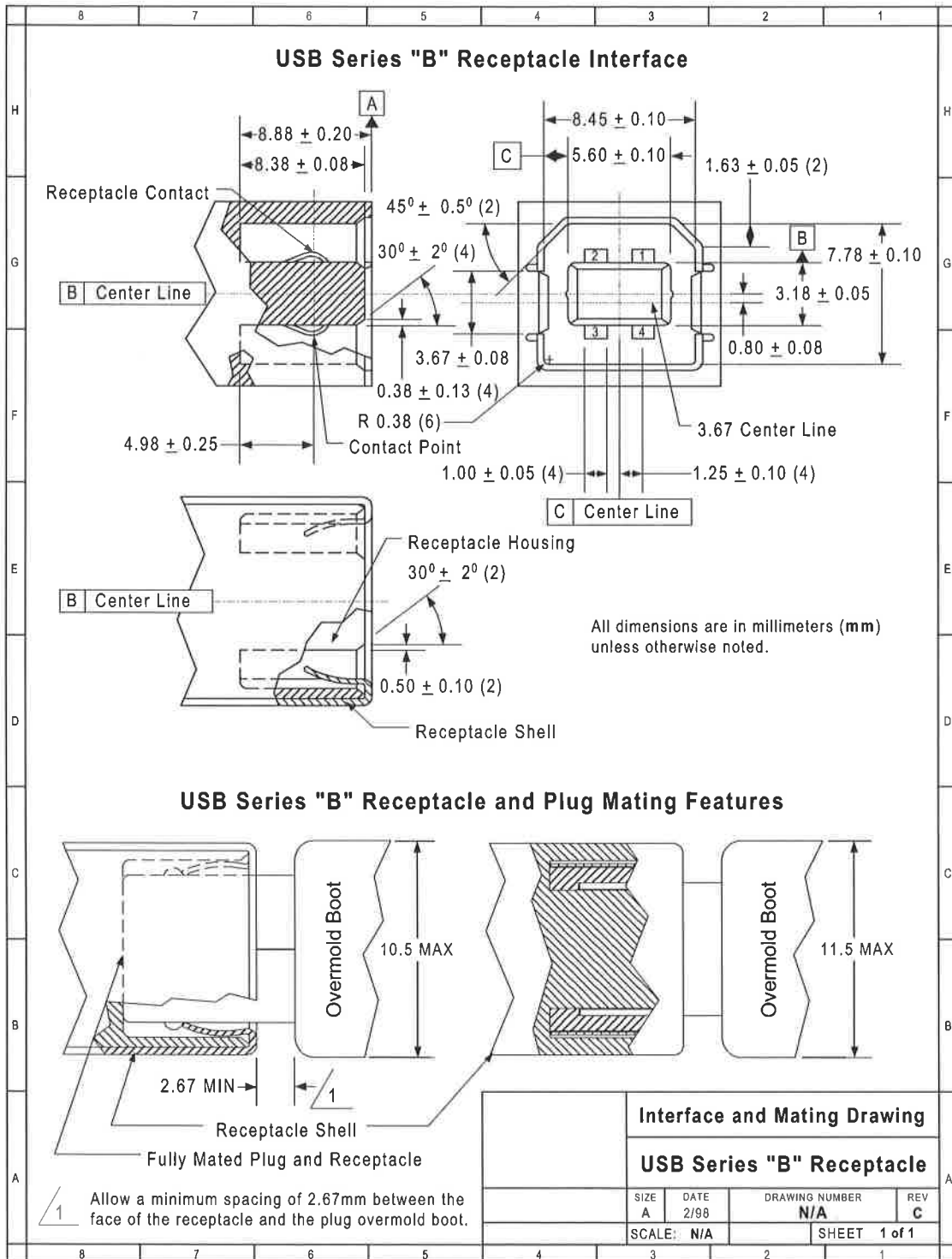


Figure 6-8. USB Series "B" Receptacle Interface and Mating Drawing

### 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, et cetera.

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 the 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.

Universal Serial Bus Specification Revision 1.1

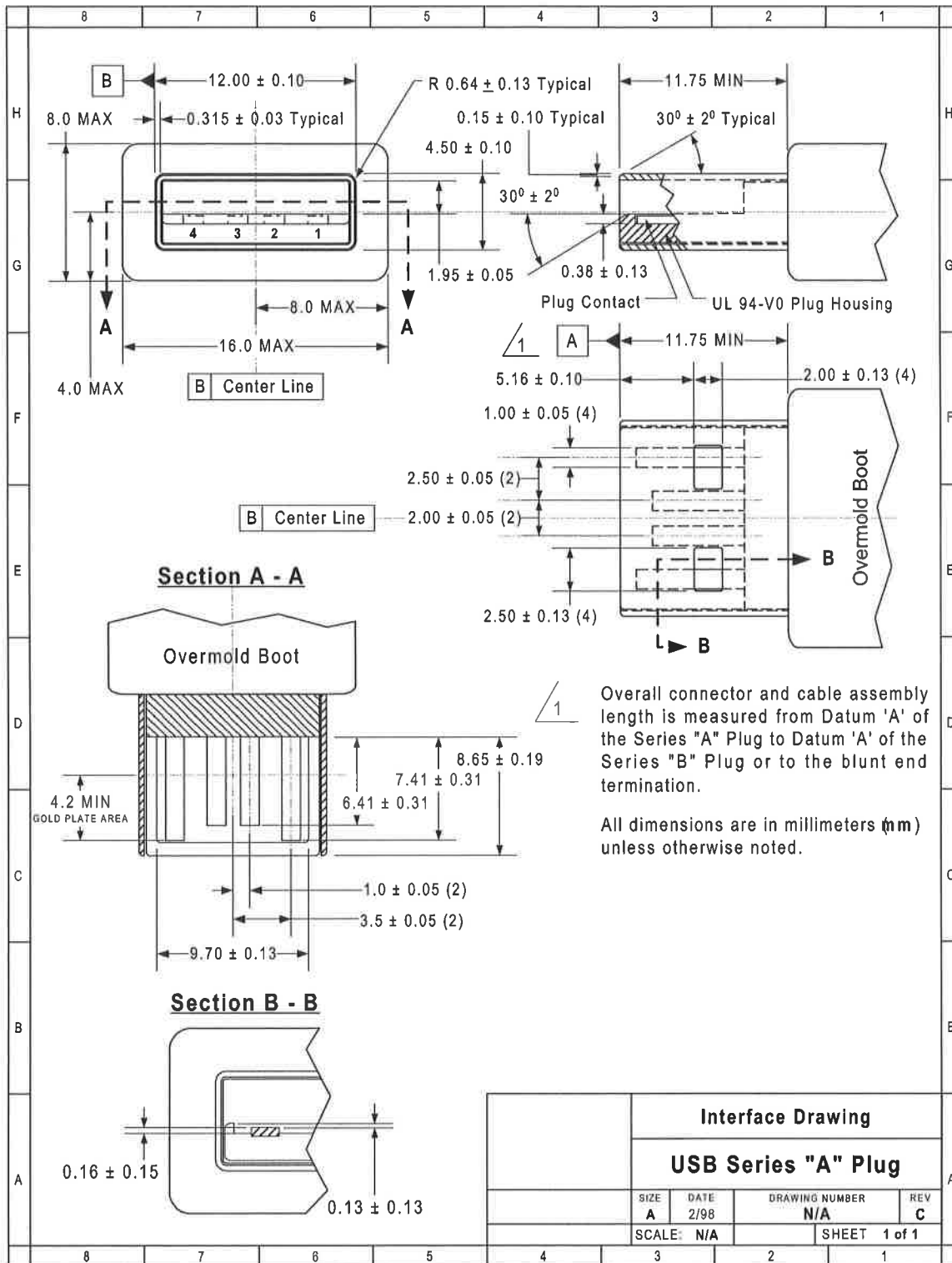


Figure 6-9. USB Series "A" Plug Interface Drawing



Universal Serial Bus Specification Revision 1.1

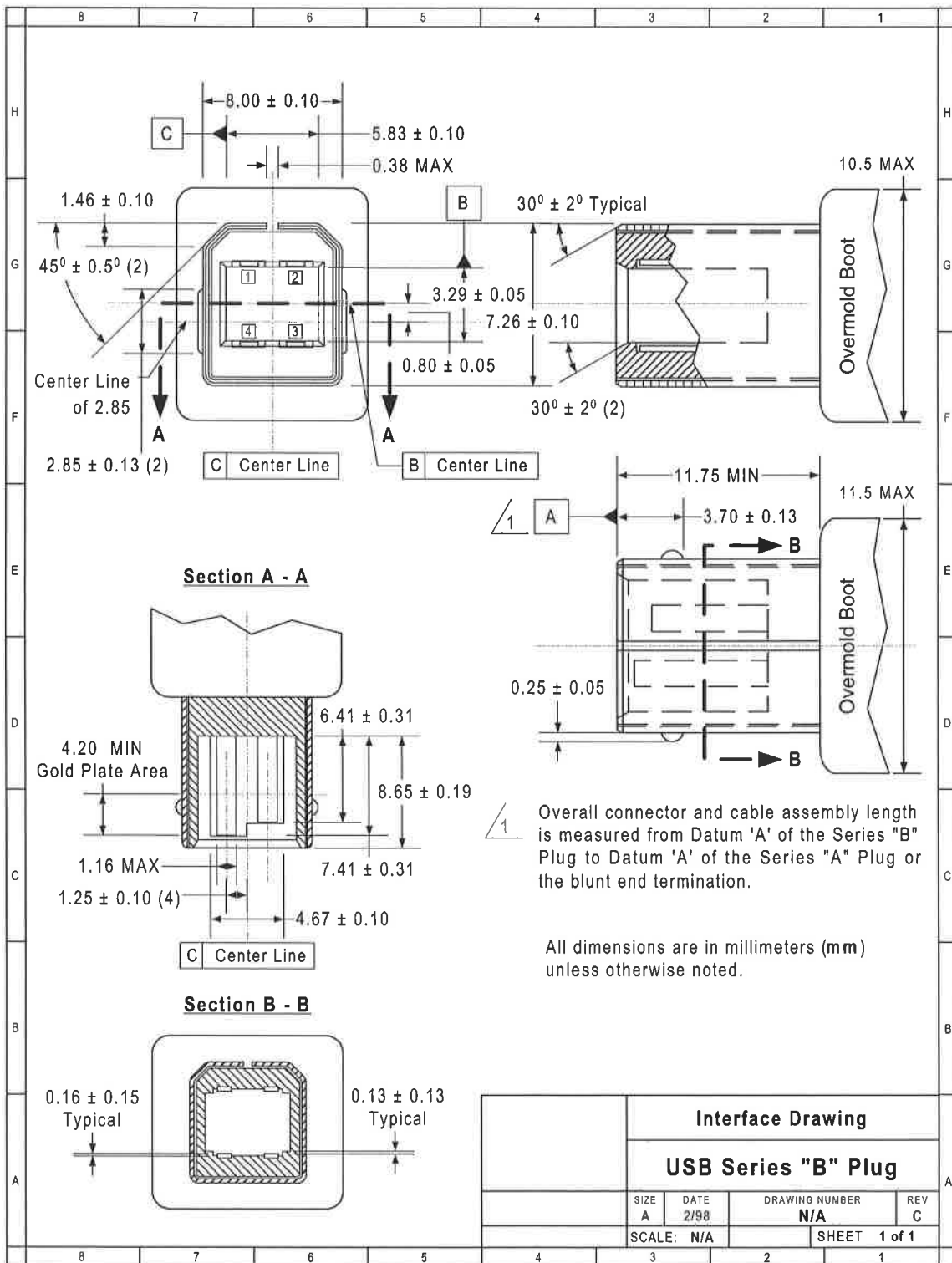


Figure 6-10. USB Series "B" Plug Interface Drawing

#### 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:  $0.30 \pm 0.05$  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.  $0.30 \pm 0.05$  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

Full-speed and Low-speed cables differ in data conductor arrangement and shielding. Low-speed cable does not require twisted data conductors or a shield. Figure 6-11 shows the typical Full-speed cable construction.

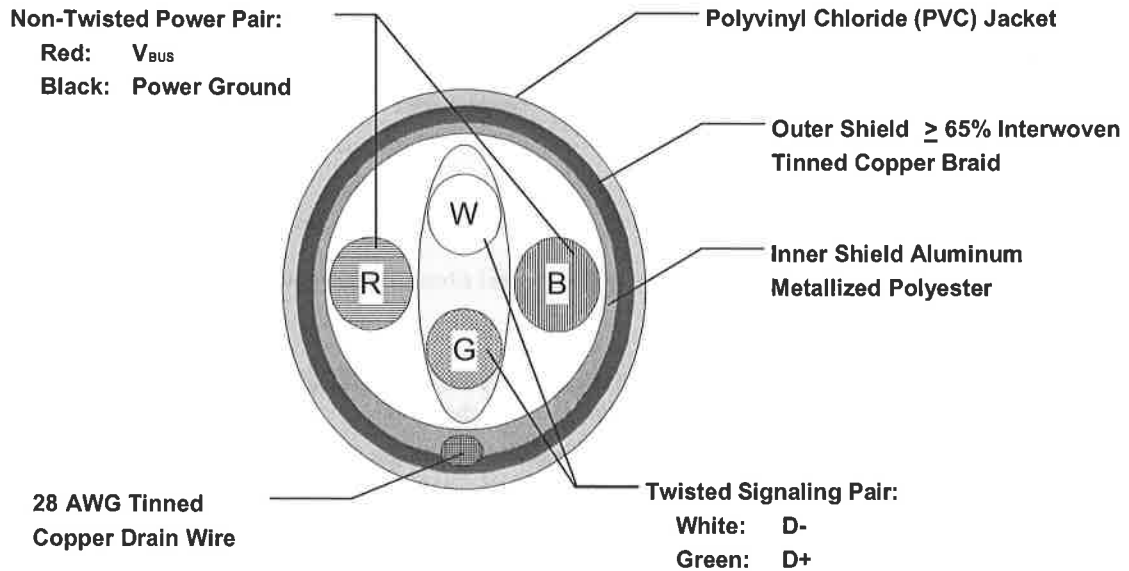


Figure 6-11. Typical Full-speed Cable Construction

### 6.6.1 Description

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,  $\geq 65\%$  tinned copper wire interwoven (braided) outer shield and PVC outer jacket.

Low-speed cable does not require the data pair be twisted or a shield and drain wire.

### 6.6.2 Construction

Raw materials used in the fabrication of this cable shall 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, et cetera, 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 shall 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

## Universal Serial Bus Specification Revision 1.1

Note: Minimum conductor construction shall 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 (not 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 shall be positioned facing out to ensure direct contact with the drain wire.
  - 2. The aluminum metallized inner shield shall over lap by approximately one-quarter turn.

Drain Wire (not 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”) 0.406 mm (0.016”)	7 x 36 19 x 40

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

- A. Coverage Area: Minimum 65%.
- B. Assembly. The interwoven (braided) tinned copper wire outer shield shall encase the aluminum metallized PET shielded power and signal pairs and shall be in direct contact with the drain wire.

Outer Polyvinyl Chloride (PVC) Jacket:

- A. Assembly: The outer PVC jacket shall encase the fully shielded power and signal pairs and shall be in direct contact with the tinned copper outer shield.
- B. Nominal Wall Thickness: 0.64 mm (0.025”).

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

- A. Minimum marking information for Full-speed cable shall 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.

## Universal Serial Bus Specification Revision 1.1

Nominal Fabricated Cable Outer Diameter:

This is a nominal value and may vary slightly from manufacturer to manufacturer as 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 shall be measured at or referenced to +20° C (68° F).

Voltage Rating: 30 Vrms maximum.

Conductor Resistance: Conductor resistance shall 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 shall not exceed five percent (5%) when measured in accordance with ASTM-D-4566 Section 15.

**Table 6-6. Conductor Resistance**

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

### 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**

<b>Test Description</b>	<b>Test Procedure</b>	<b>Performance Requirement</b>
<b>Visual and Dimensional Inspection</b>	<p><b>EIA 364-18</b></p> <p>Visual, dimensional and functional inspection in accordance the USB quality inspection plans.</p>	<p>Must meet or exceed the requirements specified by the most current version of Chapter 6 of the USB Specification.</p>
<b>Insulation Resistance</b>	<p><b>EIA 364-21</b></p> <p>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.</p>	<p>1,000 MΩ minimum.</p>
<b>Dielectric Withstanding Voltage</b>	<p><b>EIA 364-20</b></p> <p>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.</p>	<p>The dielectric must withstand 500 VAC for one minute at sea level.</p>
<b>Low Level Contact Resistance</b>	<p><b>EIA 364-23</b></p> <p>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.</p>	<p>30 mΩ maximum when measured at 20 mV maximum open circuit at 100 mA. Mated test contacts must be in a connector housing.</p>

**Universal Serial Bus Specification Revision 1.1**

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

<b>Test Description</b>	<b>Test Procedure</b>	<b>Performance Requirement</b>
<b>Contact Current Rating</b>	<p><b>EIA 364-70 — Method B</b></p> <p>The object of this test procedure is to detail a standard method to assess the current carrying capacity of mated USB connector contacts.</p>	<p>1.5 A at 250 VAC minimum when measured at an ambient temperature of 25° C. With power applied to the contacts, the <math>\Delta T</math> shall not exceed +30° C at any point in the USB connector under test.</p>
<b>Contact Capacitance</b>	<p><b>EIA 364-30</b></p> <p>The object of this test is to detail a standard method to determine the capacitance between conductive elements of a USB connector.</p>	<p>2 pF maximum unmated per contact</p>
<b>Insertion Force</b>	<p><b>EIA 364-13</b></p> <p>The object of this test is to detail a standard method for determining the mechanical forces required for inserting a USB connector.</p>	<p>35 Newtons maximum at a maximum rate of 12.5 mm (0.492”) per minute</p>
<b>Extraction Force</b>	<p><b>EIA 364-13</b></p> <p>The object of this test is to detail a standard method for determining the mechanical forces required for extracting a USB connector.</p>	<p>10 Newtons minimum at a maximum rate of 12.5 mm (0.492”) per minute</p>
<b>Durability</b>	<p><b>EIA 364-09</b></p> <p>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.</p>	<p>1,500 insertion/extraction cycles at a maximum rate of 200 cycles per hour</p>



**Universal Serial Bus Specification Revision 1.1**

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

<b>Test Description</b>	<b>Test Procedure</b>	<b>Performance Requirement</b>
<b>Cable Pull-Out</b>	<p><b>EIA 364-38</b> <b>Test Condition A</b></p> <p>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.</p>	<p>After the application of a steady state axial load of 40 Newtons for one minute</p>
<b>Physical Shock</b>	<p><b>EIA 364-27</b> <b>Test Condition H</b></p> <p>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.</p>	<p>No discontinuities of 1 <math>\mu</math>S 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</p>
<b>Random Vibration</b>	<p><b>EIA 364-28</b> <b>Test Condition V Test Letter A</b></p> <p>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.</p>	<p>No discontinuities of 1 <math>\mu</math>S or longer duration when mated USB connectors are subjected to 5.35 Gs RMS. 15 minutes in each of three mutually perpendicular planes</p>

**Universal Serial Bus Specification Revision 1.1**

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

<b>Test Description</b>	<b>Test Procedure</b>	<b>Performance Requirement</b>
<b>Thermal Shock</b>	<p><b>EIA 364-32</b> <b>Test Condition I</b></p> <p>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.</p>	<p>10 Cycles <math>-55^{\circ}\text{C}</math> and <math>+85^{\circ}\text{C}</math>. The USB connectors under test must be mated</p>
<b>Humidity Life</b>	<p><b>EIA 364-31</b> <b>Test Condition A Method III</b></p> <p>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.</p>	<p>168 Hours minimum (seven (7) complete cycles). The USB connectors under test shall be tested in accordance with EIA 364-31</p>
<b>Solderability</b>	<p><b>EIA 364-52</b></p> <p>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.</p>	<p>USB contact solder tails shall pass 95% coverage after one hour steam aging as specified in Category 2</p>
<b>Flammability</b>	<p><b>UL 94 V-0</b></p> <p>This procedure is to ensure thermoplastic resin compliance to UL flammability standards.</p>	<p>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, et cetera.</p>

**Universal Serial Bus Specification Revision 1.1**

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

<b>Test Description</b>	<b>Test Procedure</b>	<b>Performance Requirement</b>
<p><b>Cable Impedance</b>  <b>Only required for Full-speed</b></p>	<p>The object of this test is to insure the signal conductors have the proper impedance.</p> <ol style="list-style-type: none"> <li>1. Connect the Time Domain Reflectometer (TDR) outputs to the impedance/delay/skew test fixture (Note 1). Use separate 50 Ohm cables for the plus (or true) and minus (or complement) outputs. Set the TDR head to differential TDR mode.</li> <li>2. Connect the Series "A" plug of the cable to be tested to the test fixture, leaving the other end open-circuited.</li> <li>3. Define a waveform composed of the difference between the true and complement waveforms, to allow measurement of differential impedance.</li> <li>4. Measure the minimum and maximum impedances found between the connector and the open circuited far end of the cable.</li> </ol>	<p>Impedance must be in the range specified in Table 7-9 (Z<sub>0</sub>).</p>

**Universal Serial Bus Specification Revision 1.1**

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

<b>Test Description</b>	<b>Test Procedure</b>	<b>Performance Requirement</b>
<p><b>Signal Pair Attenuation</b>  <b>Only required for Full-speed</b></p>	<p>The object of this test is to insure that adequate signal strength is presented to the receiver to maintain a low error rate.</p> <ol style="list-style-type: none"> <li>1. Connect the Network Analyzer output port (port 1) to the input connector on the attenuation test fixture (Note 2).</li> <li>2. Connect the Series "A" plug of the cable to be tested to the test fixture, leaving the other end open-circuited.</li> <li>3. Calibrate the network analyzer and fixture using the appropriate calibration standards, over the desired frequency range.</li> <li>4. Follow the method listed in Hewlett Packard Application Note 380-2 to measure the open-ended response of the cable.</li> <li>5. Short circuit the Series "B" end (or bare leads end, if a captive cable), and measure the short-circuit response.</li> <li>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.</li> </ol>	<p>Refer to Section 7.1.17 for frequency range and allowable attenuation.</p>

**Universal Serial Bus Specification Revision 1.1**

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

<b>Test Description</b>	<b>Test Procedure</b>	<b>Performance Requirement</b>
<b>Propagation Delay</b>	<p>The purpose of the test is to verify the end to end propagation of the cable.</p> <ol style="list-style-type: none"> <li>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 50Ω cable for each signal, and set the TDR head to differential TDR mode.</li> <li>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.</li> <li>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.</li> <li>4. Remove the short piece of wire and re-measure the propagation delay. Subtract from it the delay of the test fixture measured in the previous step.</li> </ol>	<p><b>Full-speed</b> See Section 7.1.1.1, Section 7.1.4, Section 7.1.16 and Table 7-9 (TFSCBL)</p> <p><b>Low-speed</b> See Section 7.1.1.2, Section 7.1.16 and Table 7-9 (TLSCBL)</p>

**Universal Serial Bus Specification Revision 1.1**

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

<b>Test Description</b>	<b>Test Procedure</b>	<b>Performance Requirement</b>
<b>Propagation Delay Skew</b>	<p>This test insures that the signal on both the D+ and D- lines arrive at the receiver at the same time.</p> <ol style="list-style-type: none"> <li>1. Connect the TDR to the fixture with test sample cable, as in the previous section.</li> <li>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.</li> </ol>	<p>Propagation skew must meet the requirements as listed in Section 7.1.3.</p>
<b>Capacitive Load</b> <b>Only required for Low-speed</b>	<p>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.</p> <ol style="list-style-type: none"> <li>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.</li> <li>2. Connect the series "A" plug to the fixture, with the series "B" end leads open-circuited.</li> <li>3. Set the Impedance Analyzer to a frequency of 100 kHz, to measure the capacitance.</li> </ol>	<p>See Section 7.1.1.2 and Table 7-7 (CLINUA)</p>

**Note 1: 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 50Ω. Coaxial connectors should be provided on the fixture for connection from the TDR.

**Note 2: Attenuation test 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 50Ω output, to the balanced (also known as differential) and likely different impedance loaded presented by the cable. A second transformer or balun

## Universal Serial Bus Specification Revision 1.1

should be used on the other end of the cable under test, to convert the signal back to unbalanced from 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 following drawings describe typical receptacle PCB interfaces. This is included for information purposes only.

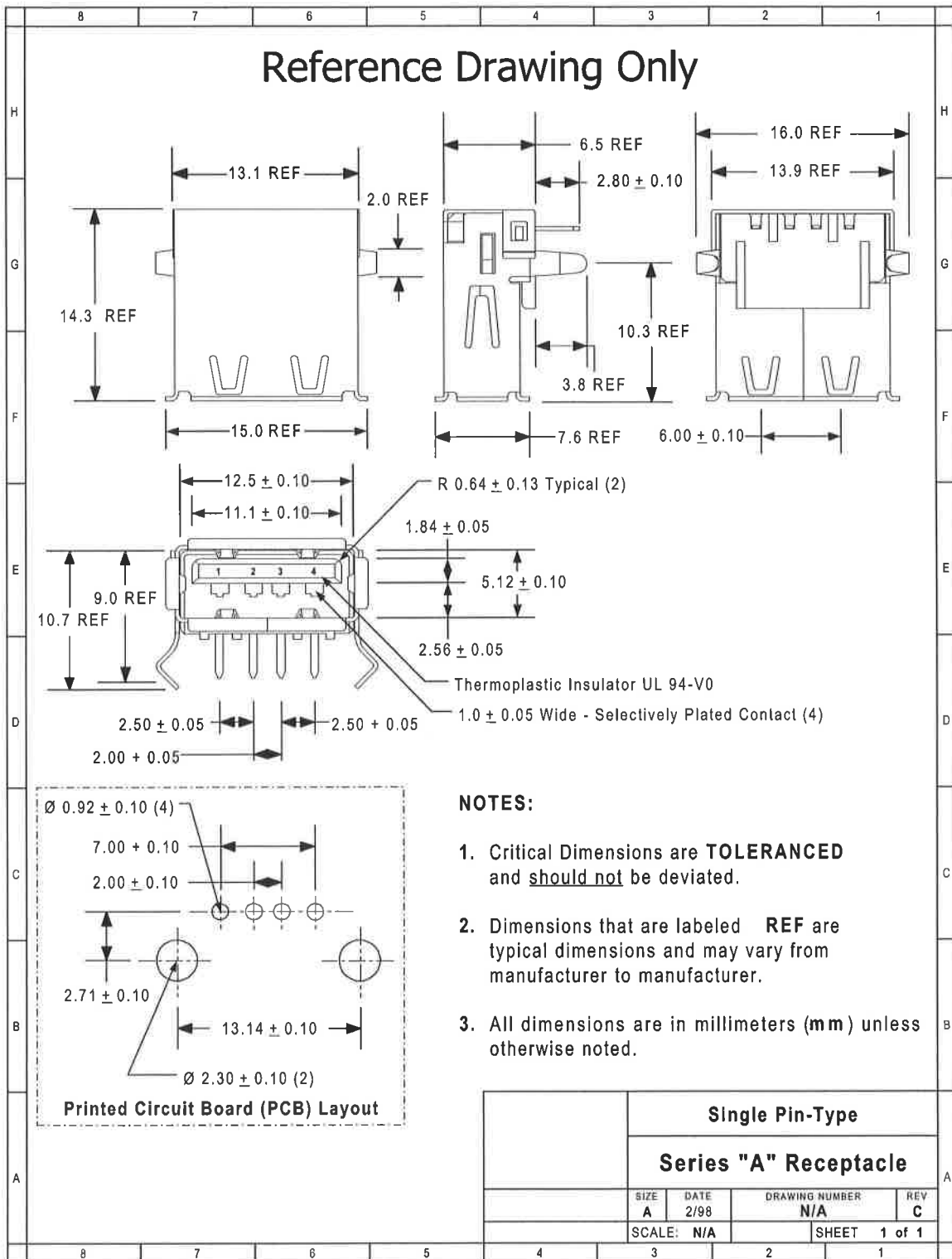


Figure 6-12. Single Pin-Type Series "A" Receptacle





Universal Serial Bus Specification Revision 1.1

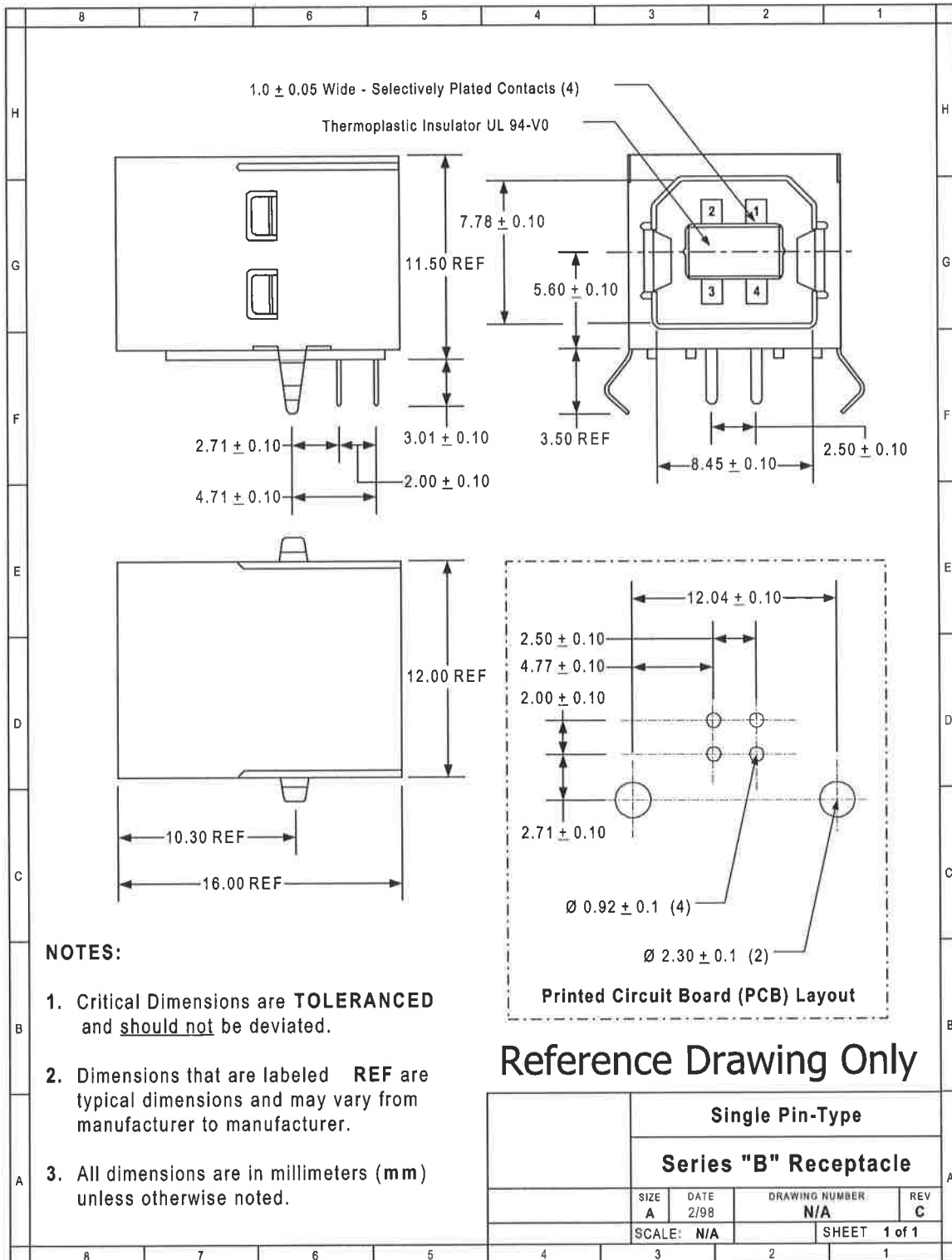


Figure 6-14. Single Pin-Type Series "B" Receptacle



# Chapter 7 Electrical

This chapter describes the electrical specification for the USB. It contains signaling, power distribution, and physical layer specifications.

## 7.1 Signaling

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

### 7.1.1 USB Driver Characteristics

The USB uses a differential output driver to drive the USB data signal onto the USB cable. The static output swing of the driver in its low state must be below  $V_{OL}$  (max) of 0.3V with a 1.5k $\Omega$  load to 3.6V and in its high state must be above the  $V_{OH}$  (min) of 2.8V with a 15k $\Omega$  load to ground as listed in Table 7-5. 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.

USB devices must be capable of withstanding continuous exposure to the waveforms shown in Figure 7-1 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 $\Omega$ . The open-circuit voltage of the source shown in Figure 7-1 is based on the expected worst-case overshoot and undershoot.

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

#### Evaluation Setup

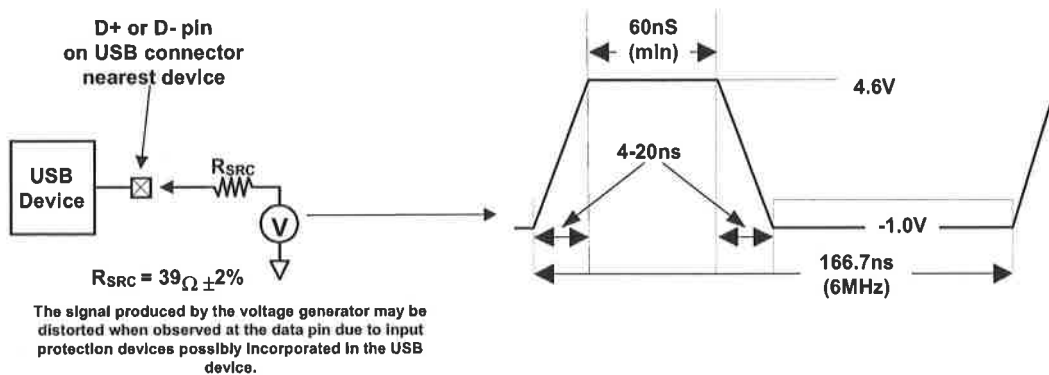


Figure 7-1. Maximum Input Waveforms for USB Signaling

A USB device must be able to withstand a continuous short circuit of D+ and D- to  $V_{BUS}$ , GND, other data line, or the cable shield at the connector. The device must not be damaged when presented with a driving signal that provides a duty cycle of 50% transmit and 50% receive. 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  $V_{BUS}$ .

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

A full-speed USB connection is made through a shielded, twisted pair cable with a characteristic impedance ( $Z_0$ ) of  $90\Omega \pm 15\%$  and a maximum one-way delay of 26ns. The impedance of each of the drivers ( $Z_{DRV}$ ) must be between  $28\Omega$  and  $44\Omega$  i.e. within the grey area in Figure 7-3.

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 7-2. The series resistor  $R_S$  is included in the buffer impedance requirement shown in Figure 7-3. In the rest of the chapter, references to the buffer assume a buffer with the series impedance unless stated otherwise.

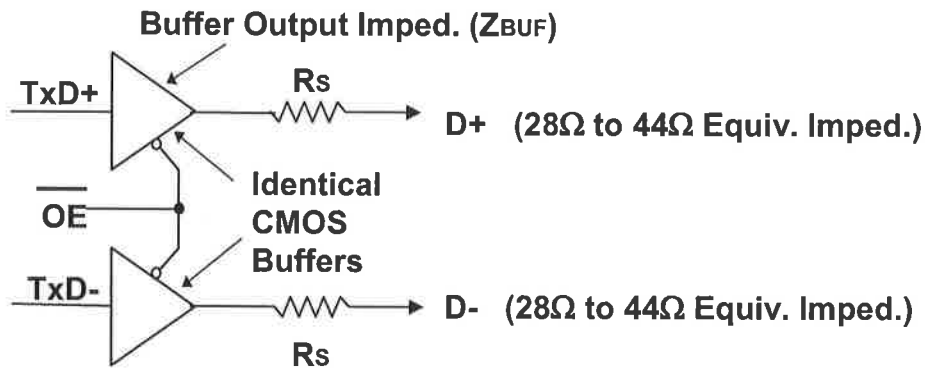


Figure 7-2. Example Full-speed CMOS Driver Circuit

The buffer impedance must be measured for driving high as well as driving low. Figure 7-3 shows the composite V/I characteristics for the full-speed drivers with included series damping resistor ( $R_S$ ). 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  $|6.1V_{OH}|mA$  when driving low and  $-|6.1V_{OH}|mA$  when driving high. 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  $\pm 10.71 * V_{OH}$  mA and the voltage applied to D+/D- need not exceed  $0.3 * V_{OH}$  for the drive low case and need not drop below  $0.7 * V_{OH}$  for the drive high case.

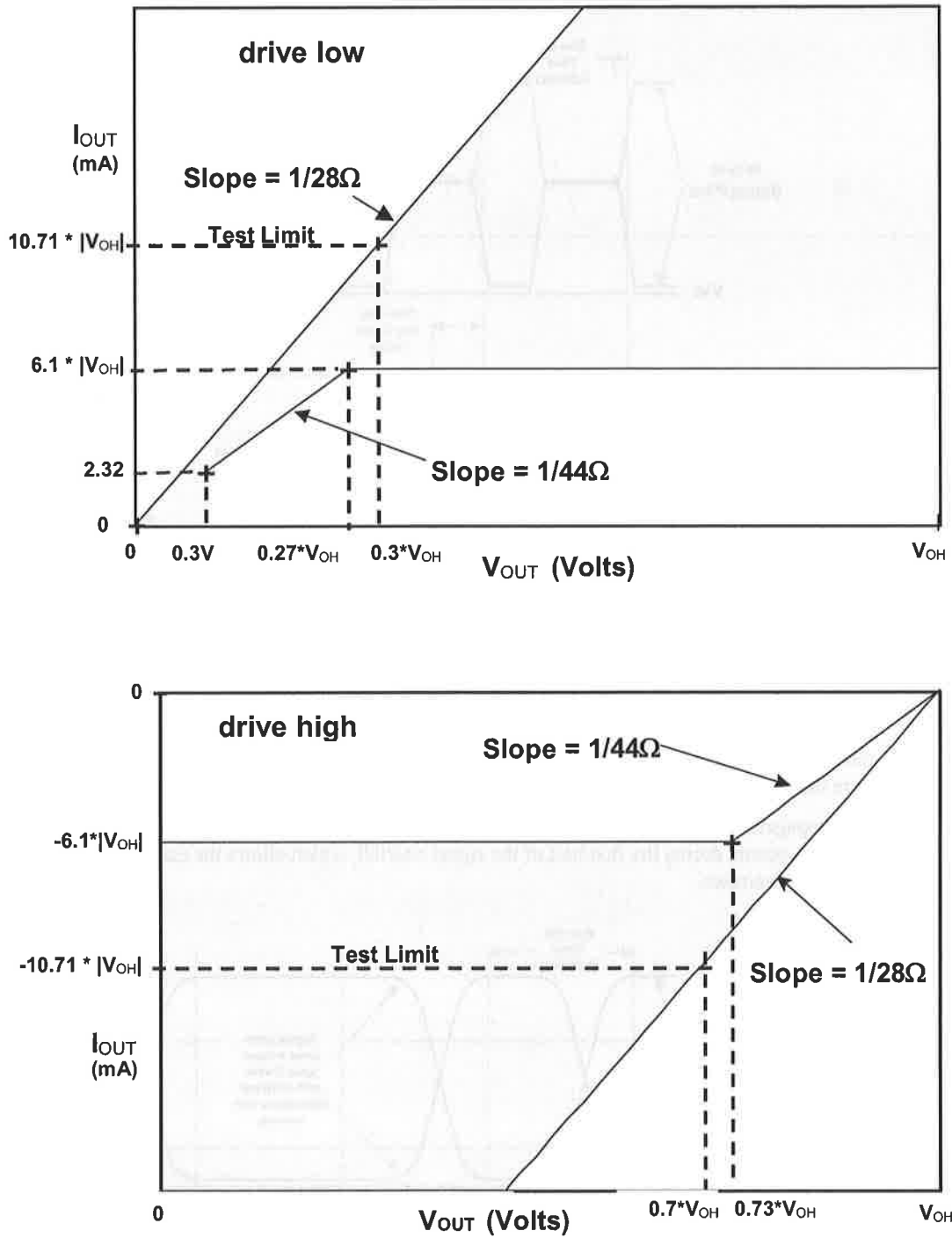


Figure 7-3. Full-speed Buffer V/I Characteristics

Figure 7-4 shows the full-speed driver signal waveforms.

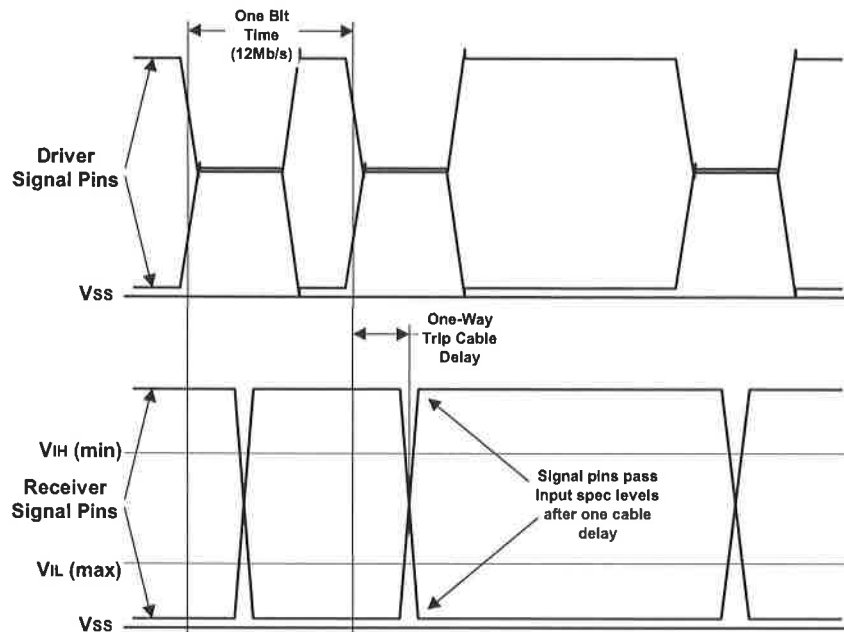


Figure 7-4. Full-speed Signal Waveforms

### 7.1.1.2 Low-speed (1.5Mb/s) Driver Characteristics

A low-speed device must have a captive cable with the Series A connector on the plug end. The combination of the cable and the device must have a single-ended capacitance of no less than 200pF and no more than 450pF on the D+ or D- lines.

The propagation delay (TLSCBL) of a low-speed cable must be less than 18ns. This is to ensure that the reflection occurs during the first half of the signal rise/fall, which allows the cable to be approximated by a lumped capacitance.

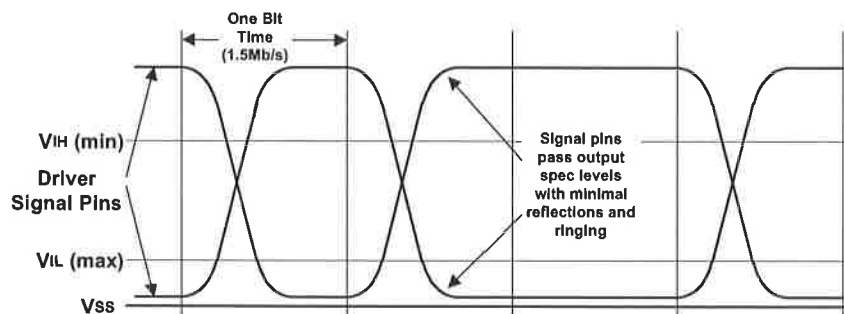


Figure 7-5. Low-speed Driver Signal Waveforms

### 7.1.2 Data Signal Rise and Fall

The output rise time and fall times are measured between 10% and 90% of the signal (Figure 7-6). Rise and fall time requirements apply to differential transitions as well as to transitions between differential and single-ended signaling.

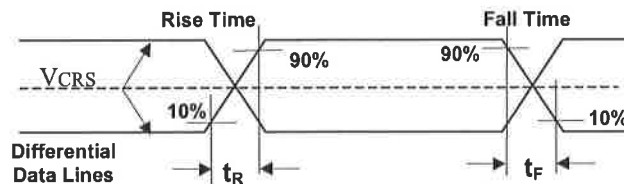
## Universal Serial Bus Specification Revision 1.1

The rise and fall times for full-speed buffers are measured with the load shown in Figure 7-7. The rise and fall times must be between 4ns and 20ns, and matched to within  $\pm 10\%$  to minimize RFI emissions and signal skew. The transitions must be monotonic.

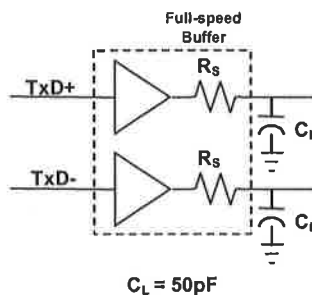
The rise and fall times for low-speed buffers are measured with the load shown in Figure 7-8. The capacitive load shown in Figure 7-8 is representative of the worst-case load allowed by the specification. A downstream port is allowed 150pF of input/output capacitance (C<sub>IND</sub>). A low-speed device (including cable) may have a capacitance of as little as 200pF and as much as 450pF. This gives a range of 200pF to 600pF as the capacitive load that a downstream low-speed buffer might encounter. Upstream buffers on low-speed devices must be designed to drive the capacitance of the attached cable plus an additional 150pF. If a low-speed buffer is designed for an application where the load capacitance is known to fall in a different range, the test load can be adjusted to match the actual application. Low-speed buffers on hosts and hubs that are attached to USB receptacles must be designed for the 200pF to 600pF range. The rise and fall time must be between 75ns and 300ns for any balanced, capacitive test load. In all cases, the edges must be matched to within  $\pm 20\%$  to minimize RFI emissions and signal skew. The transitions must be monotonic.

For both full-speed and low-speed signaling, the crossover voltage (V<sub>CRS</sub>) must be between 1.3V and 2.0V.

This specification does not require matching signal swing matching to any greater degree than described above. However, when signaling, it is preferred that the average voltage on the D+ and D- lines should be constant. This means that the amplitude of the signal swing on both D+ and D- should be the same; the low and high going transition should begin at the same time and change at the same rate; and the crossover voltage should be the same when switching to a J or K. Deviations from signal matching will result in common-mode noise that will radiate and affect the ability of devices and systems to pass tests that are mandated by government agencies.



**Figure 7-6. Data Signal Rise and Fall Time**



**Figure 7-7. Full-speed Load**