Universal Serial Bus Specification

Compaq

Intel

Microsoft

NEC

Revision 1.1
September 23, 1998

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.

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.

Please send comments via electronic mail to techsup@usb.org
For industry information, refer to the USB Implementers Forum web page at http://www.usb.org

Contents

CHAPTER 1 INTRODUCTION 1

1.1 Motivation 1

1.2 Objective of the Specification 1

1.3 Scope of the Document 2

1.4 Document Organization 2

CHAPTER 2 TERMS AND ABBREVIATIONS 3

CHAPTER 3 BACKGROUND 11

3.1 Goals for the Universal Serial Bus 11

3.2 Taxonomy of Application Space 12

3.3 Feature List 12

CHAPTER 4 ARCHITECTURAL OVERVIEW 15

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

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

4.8 USB Devices....21
4.8.1 Device Characterizations....21
4.8.2 Device Descriptions....22

4.9 USB Host: Hardware and Software....24

4.10 Architectural Extensions....24

CHAPTER 5 USB DATA FLOW MODEL....25

5.1 Implementer Viewpoints....25

5.2 Bus Topology....27
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

5.3 USB Communication Flow....31
5.3.1 Device Endpoints....32
5.3.2 Pipes....33

5.4 Transfer Types....35

5.5 Control Transfers....36
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

5.6 Isochronous Transfers....41
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

5.7 Interrupt Transfers....43
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

5.8 Bulk Transfers....46
5.8.1 Bulk Transfer Data Format....47
5.8.2 Bulk Transfer Direction....47
5.8.3 Bulk Transfer Packet Size Constraints....47

5.8.4 Bulk Transfer Bus Access Constraints 47
5.8.5 Bulk Transfer Data Sequences 48

5.9 Bus Access for Transfers 49
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

5.10 Special Considerations for Isochronous Transfers 55
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

CHAPTER 6 MECHANICAL 73

6.1 Architectural Overview 73

6.2 Keyed Connector Protocol 73

6.3 Cable 74

6.4 Cable Assembly 74
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

6.5 Connector Mechanical Configuration and Material Requirements 80
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

6.6 Cable Mechanical Configuration and Material Requirements 90
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

6.7 Electrical, Mechanical and Environmental Compliance Standards 94
6.7.1 Applicable Documents 102

6.8 USB Grounding 102

6.9 PCB Reference Drawings 102

CHAPTER 7 ELECTRICAL

CHAPTER 7 ELECTRICAL 107

7.1 Signaling **107**
- 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

7.2 Power Distribution **134**
- 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

7.3 Physical Layer **141**
- 7.3.1 Regulatory Requirements 142
- 7.3.2 Bus Timing/Electrical Characteristics 142
- 7.3.3 Timing Waveforms 151

CHAPTER 8 PROTOCOL LAYER 155

8.1 Bit Ordering **155**

8.2 SYNC Field **155**

8.3 Packet Field Formats **155**
- 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

8.4 Packet Formats **158**
- 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

8.5 Transaction Formats 162
8.5.1 Bulk Transactions 163
8.5.2 Control Transfers 164
8.5.3 Interrupt Transactions 167
8.5.4 Isochronous Transactions 168

8.6 Data Toggle Synchronization and Retry 168
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

8.7 Error Detection and Recovery 172
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

CHAPTER 9 USB DEVICE FRAMEWORK 175

9.1 USB Device States 175
9.1.1 Visible Device States 175
9.1.2 Bus Enumeration 179

9.2 Generic USB Device Operations 180
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

9.3 USB Device Requests 183
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

9.4 Standard Device Requests 185
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

9.5 Descriptors 196

9.6 Standard USB Descriptor Definitions....196
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

9.7 Device Class Definitions....205
9.7.1 Descriptors....205
9.7.2 Interface(s) and Endpoint Usage....205
9.7.3 Requests....206

CHAPTER 10 USB HOST: HARDWARE AND SOFTWARE 207

10.1 Overview of the USB Host....207
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

10.2 Host Controller Requirements....211
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

10.3 Overview of Software Mechanisms....214
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

10.4 Host Controller Driver....218

10.5 Universal Serial Bus Driver....219
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

10.6 Operating System Environment Guides....227

CHAPTER 11 HUB SPECIFICATION 229

11.1 Overview....229
11.1.1 Hub Architecture....230
11.1.2 Hub Connectivity....230

11.2 Hub Frame Timer........232
11.2.1 Frame Timer Synchronization........233
11.2.2 EOF1 and EOF2 Timing Points........234

11.3 Host Behavior at End-of-Frame........235
11.3.1 Latest Host Packet........235
11.3.2 Packet Nullification........235
11.3.3 Transaction Completion Prediction........236

11.4 Internal Port........237
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

11.5 Downstream Ports........239
11.5.1 Downstream Port State Descriptions........240
11.5.2 Disconnect Detect Timer........243

11.6 Upstream Port........244
11.6.1 Receiver........244
11.6.2 Transmitter........246

11.7 Hub Repeater........249
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

11.8 Bus State Evaluation........251
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

11.9 Suspend and Resume........253

11.10 Hub Reset Behavior........254
11.10.1 Hub Receiving Reset on Upstream Port........254

11.11 Hub Port Power Control........255
11.11.1 Multiple Gangs........255

11.12 Hub I/O Buffer Requirements........256
11.12.1 Pull-up and Pull-down Resistors........256
11.12.2 Edge Rate Control........256

11.13 Hub Controller........256
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

11.14 Hub Configuration........261

11.15 Descriptors 263
11.15.1 Standard Descriptors 263
11.15.2 Class-specific Descriptors 264

11.16 Requests 266
11.16.1 Standard Requests 266
11.16.2 Class-specific Requests 267

INDEX **281**

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" Recptacle 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

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

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

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, Mechnaical 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

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, *wHubStatus* 272
Table 11-14. Hub Change Field, *wHubChange* 272
Table 11-15. Port Status Field, *wPortStatus* 274
Table 11-16. Port Change Field, *wPortChange* 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/adapter 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.

ACK	Handshake packet indicating a positive acknowledgment.
Active Device	A device that is powered and is not in the Suspend state.
Asynchronous Data	Data transferred at irregular intervals with relaxed latency requirements.
Asynchronous RA	The incoming data rate, Fs_i, and the outgoing data rate, Fs_o, of the RA process are independent (i.e., there is no shared master clock). See also Rate Adaptation.
Asynchronous SRC	The incoming sample rate, Fs_i, and outgoing sample rate, Fs_o, of the SRC process are independent (i.e., there is no shared master clock). See also Sample Rate Conversion.
Audio Device	A device that sources or sinks sampled analog data.
AWG#	The measurement of a wire's cross section, as defined by the American Wire Gauge standard.
Babble	Unexpected bus activity that persists beyond a specified point in a frame.
Bandwidth	The amount of data transmitted per unit of time, typically bits per second (b/s) or bytes per second (B/s).
Big Endian	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.
Bit	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).
Bit Stuffing	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/s	Transmission rate expressed in bits per second.
B/s	Transmission rate expressed in bytes per second.
Buffer	Storage used to compensate for a difference in data rates or time of occurrence of events, when transmitting data from one device to another.
Bulk Transfer	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.
Bus Enumeration	Detecting and identifying USB devices.

Byte	A data element that is eight bits in size.
Capabilities	Those attributes of a USB device that are administrated by the host.
Characteristics	Those qualities of a USB device that are unchangeable; for example, the device class is a device characteristic.
Client	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.
Configuring Software	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.
Control Endpoint	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.
Control Pipe	Same as a message pipe.
Control Transfer	One of the four USB transfer types. Control transfers support configuration/command/status type communications between client and function. See also Transfer Type.
CRC	See Cyclic Redundancy Check.
CTI	Computer Telephony Integration.
Cyclic Redundancy Check (CRC)	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.
Default Address	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.
Default Pipe	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.
Device	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. When used as a non-specific reference, a USB device is either a hub or a function.
Device Address	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.

Device Endpoint	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.
Device Resources	Resources provided by UB devices, such as buffer space and endpoints. See also Host Resources and Universal Serial Bus Resources.
Device Software	Software that is responsible for using a USB device. This software may or may not also be responsible for configuring the device for use.
Downstream	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.
Driver	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.
DWORD	Double word. A data element that is two words (i.e., four bytes or 32 bits) in size.
Dynamic Insertion and Removal	The ability to attach and remove devices while the host is in operation.
E^{2}PROM	See Electrically Erasable Programmable Read Only Memory.
EEPROM	See Electrically Erasable Programmable Read Only Memory.
Electrically Erasable Programmable Read Only Memory (EEPROM)	Non-volatile rewritable memory storage technology.
End User	The user of a host.
Endpoint	See Device Endpoint.
Endpoint Address	The combination of an endpoint number and an endpoint direction on a USB device. Each endpoint address supports data transfer in one direction.
Endpoint Direction	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.
Endpoint Number	A four-bit value between 0H and FH, inclusive, associated with an endpoint on a USB device.
EOF	End-of-Frame.
EOP	End-of-Packet.
External Port	See Port.
False EOP	A spurious, usually noise-induced event that is interpreted by a packet receiver as an EOP.
Frame	The time from the start of one SOF token to the start of the subsequent SOF token; consists of a series of transactions.

Frame Pattern	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.
Fs	See Sample Rate.
Full-duplex	Computer data transmission occurring in both directions simultaneously.
Function	A USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers.
Handshake Packet	A packet that acknowledges or rejects a specific condition. For examples, see ACK and NAK.
Host	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.
Host Controller	The host's USB interface.
Host Controller Driver (HCD)	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.
Host Resources	Resources provided by the host, such as buffer space and interrupts. See also Device Resources and Universal Serial Bus Resources.
Hub	A USB device that provides additional connections to the USB.
Hub Tier	The level of connect within a USB network topology, given as the number of hubs through which the data has to flow.
Interrupt Request (IRQ)	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.
Interrupt Transfer	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.
I/O Request Packet	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.
IRP	See I/O Request Packet.
IRQ	See Interrupt Request.
Isochronous Data	A stream of data whose timing is implied by its delivery rate.
Isochronous Device	An entity with isochronous endpoints, as defined in the USB Specification, that sources or sinks sampled analog streams or synchronous data streams.
Isochronous Sink Endpoint	An endpoint that is capable of consuming an isochronous data stream that is sent by the host.
Isochronous Source Endpoint	An endpoint that is capable of producing an isochronous data stream and sending it to the host.
Isochronous Transfer	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.

Jitter	A tendency toward lack of synchronization caused by mechanical or electrical changes. More specifically, the phase shift of digital pulses over a transmission medium.
kb/s	Transmission rate expressed in kilobits per second.
kB/s	Transmission rate expressed in kilobytes per second.
Little Endian	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.
LOA	Loss of bus activity characterized by an SOP without a corresponding EOP.
LSb	Least significant bit.
LSB	Least significant byte.
Mb/s	Transmission rate expressed in megabits per second.
MB/s	Transmission rate expressed in megabytes per second.
Message Pipe	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.
MSb	Most significant bit.
MSB	Most significant byte.
NAK	Handshake packet indicating a negative acknowledgment.
Non Return to Zero Invert (NRZI)	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.
NRZI	See Non Return to Zero Invert.
Object	Host software or data structure representing a USB entity.
Packet	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.
Packet Buffer	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.
Packet ID (PID)	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.
Phase	A token, data, or handshake packet; a transaction has three phases.
Phase Locked Loop (PLL)	A circuit that acts as a phase detector to keep an oscillator in phase with an incoming frequency.
Physical Device	A device that has a physical implementation; e.g., speakers, microphones, and CD players.
PID	See Packet ID.

Pipe	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.
PLL	See Phase Locked Loop.
Polling	Asking multiple devices, one at a time, if they have any data to transmit.
POR	See Power On Reset.
Port	Point of access to or from a system or circuit. For the USB, the point where a USB device is attached.
Power On Reset (POR)	Restoring a storage device, register, or memory to a predetermined state when power is applied.
Programmable Data Rate	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.
Protocol	A specific set of rules, procedures, or conventions relating to format and timing of data transmission between two devices.
RA	See Rate Adaptation.
Rate Adaptation	The process by which an incoming data stream, sampled at Fs_i, is converted to an outgoing data stream, sampled at Fs_o,with a certain loss of quality, determined by the rate adaptation algorithm. Error control mechanisms are required for the process. Fs_i and Fs_o can be different and asynchronous. Fs_i is the input data rate of the RA; Fs_o is the output data rate of the RA.
Request	A request made to a USB device contained within the data portion of a SETUP packet.
Retire	The action of completing service for a transfer and notifying the appropriate software client of the completion.
Root Hub	A USB hub directly attached to the Host Controller. This hub is attached to the host (tier 0).
Root Port	The downstream port on a Root Hub.
Sample	The smallest unit of data on which an endpoint operates; a property of an endpoint.
Sample Rate (Fs)	The number of samples per second, expressed in Hertz (Hz).
Sample Rate Conversion (SRC)	A dedicated implementation of the RA process for use on sampled analog data streams. The error control mechanism is replaced by interpolating techniques.
Service	A procedure provided by a System Programming Interface (SPI).
Service Interval	The period between consecutive requests to a USB endpoint to send or receive data.
Service Jitter	The deviation of service delivery from its scheduled delivery time.
Service Rate	The number of services to a given endpoint per unit time.
SOF	See Start-of-Frame.
SOP	Start-of-Packet.

SPI	See System Programming Interface.
SRC	See Sample Rate Conversion.
Stage	One part of the sequence composing a control transfer; stages include the Setup stage, the Data stage, and the Status stage.
Start-of-Frame (SOF)	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.
Stream Pipe	A pipe that transfers data as a stream of samples with no defined USB structure.
Synchronization Type	A classification that characterizes an isochronous endpoint's capability to connect to other isochronous endpoints.
Synchronous RA	The incoming data rate, Fs_i, and the outgoing data rate, Fs_o, of the RA process are derived from the same master clock. There is a fixed relation between Fs_i and Fs_o.
Synchronous SRC	The incoming sample rate, Fs_i, and outgoing sample rate, Fs_o, of the SRC process are derived from the same master clock. There is a fixed relation between Fs_i and Fs_o.
System Programming Interface (SPI)	A defined interface to services provided by system software.
TDM	See Time Division Multiplexing.
Termination	Passive components attached at the end of cables to prevent signals from being reflected or echoed.
Time Division Multiplexing (TDM)	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.
Timeout	The detection of a lack of bus activity for some predetermined interval.
Token Packet	A type of packet that identifies what transaction is to be performed on the bus.
Transaction	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.
Transfer	One or more bus transactions to move information between a software client and its function.
Transfer Type	Determines the characteristics of the data flow between a software client and its function. Four transfer types are defined: control, interrupt, bulk, and isochronous.
Turn-around Time	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.
USBD	See Universal Serial Bus Driver.
Universal Serial Bus Driver (USBD)	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 Resources	Resources provided by the USB, such as bandwidth and power. See also Device Resources and Host Resources
Upstream	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.
Virtual Device	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.
Word	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.

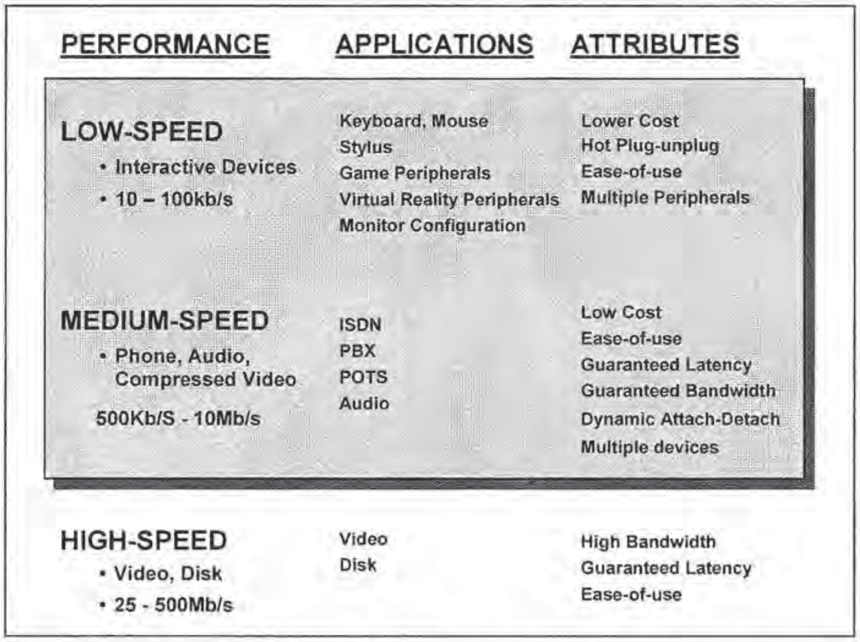
3.1 Goals for the Universal Serial Bus

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.



PERFORMANCE	APPLICATIONS	ATTRIBUTES
LOW-SPEED • Interactive Devices • 10 – 100kb/s	Keyboard, Mouse Stylus Game Peripherals Virtual Reality Peripherals Monitor Configuration	Lower Cost Hot Plug-unplug Ease-of-use Multiple Peripherals
MEDIUM-SPEED • Phone, Audio, Compressed Video 500Kb/S - 10Mb/s	ISDN PBX POTS Audio	Low Cost Ease-of-use Guaranteed Latency Guaranteed Bandwidth Dynamic Attach-Detach Multiple devices
HIGH-SPEED • Video, Disk • 25 - 500Mb/s	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 pointsi 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

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

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.

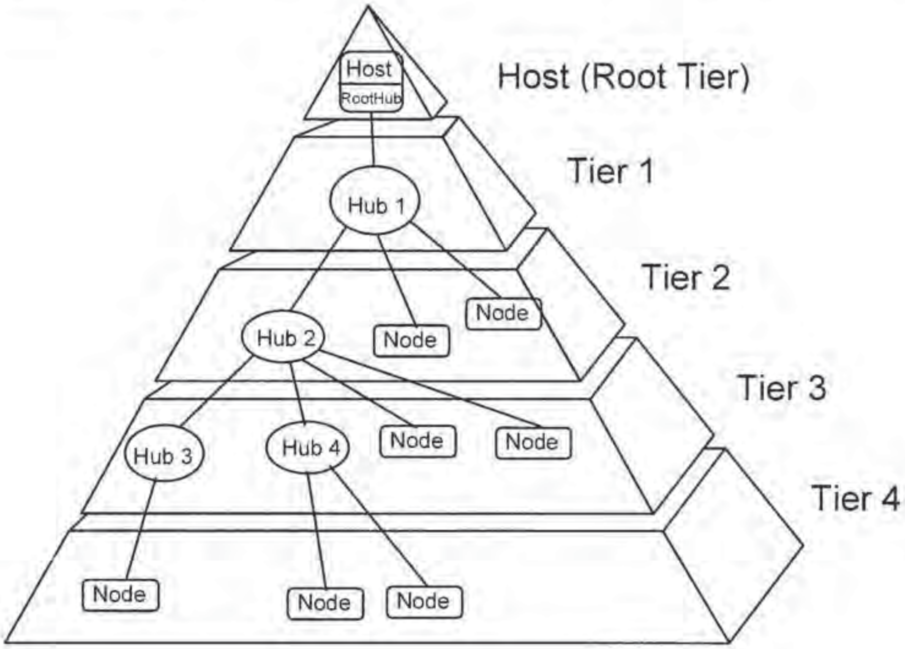


Figure 4-1. Bus Topology

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.

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.

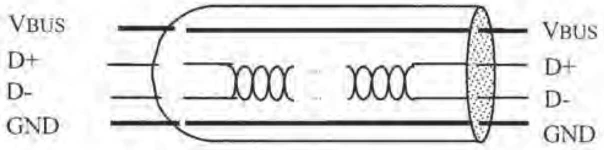


Figure 4-2. USB Cable

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 the 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 though 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 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.

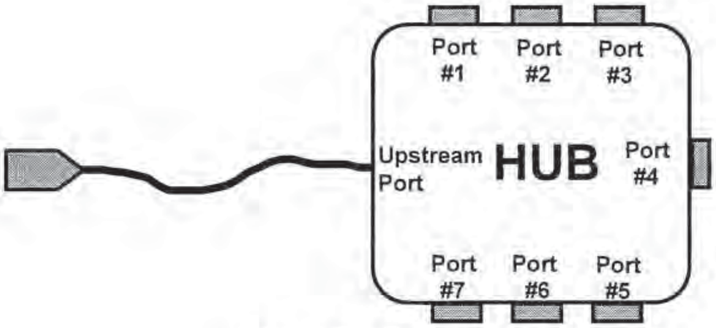


Figure 4-3. A Typical Hub

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

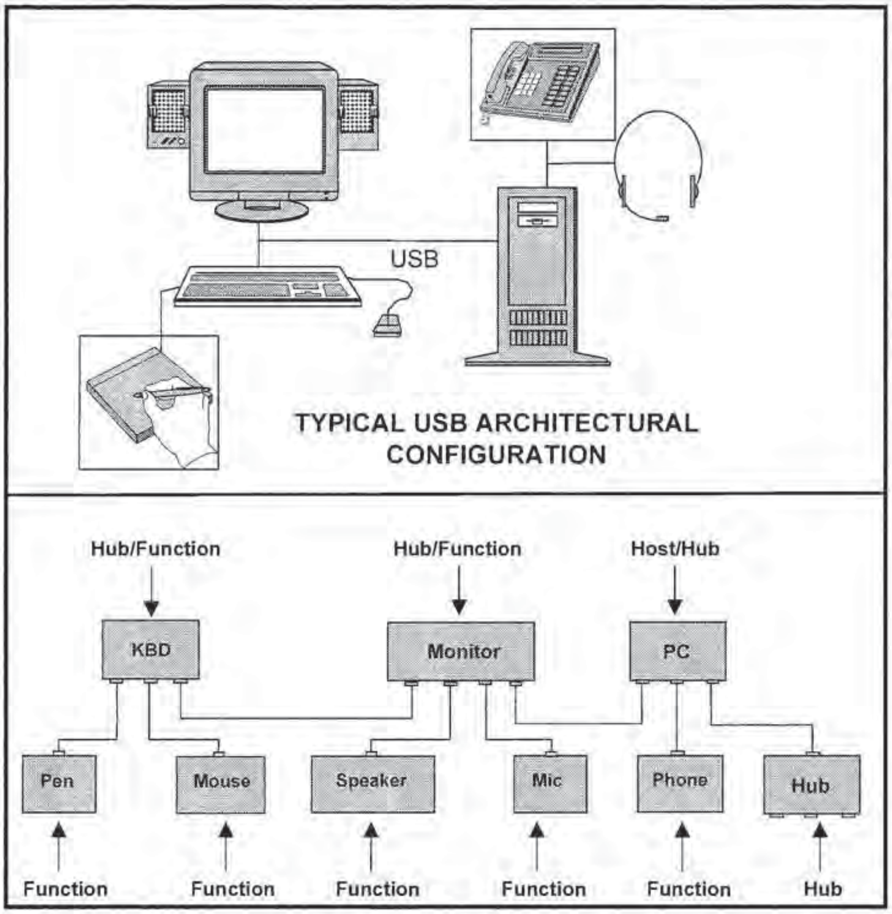


Figure 4-4. Hubs in a Desktop Computer Environment

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.

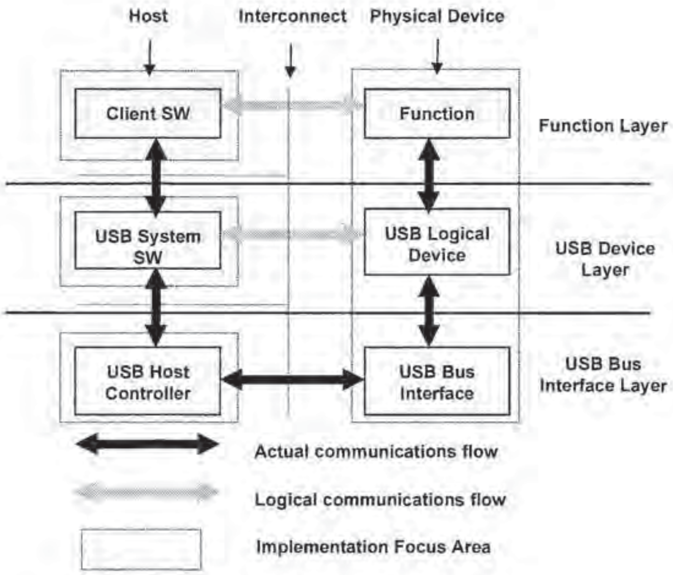


Figure 5-2. USB Implementation Areas

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.

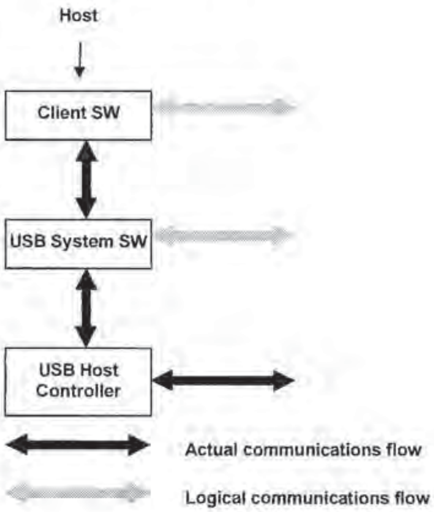


Figure 5-3. Host Composition

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.

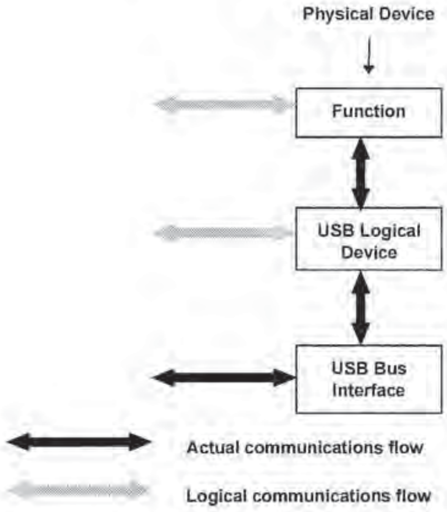


Figure 5-4. Physical Device Composition

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.

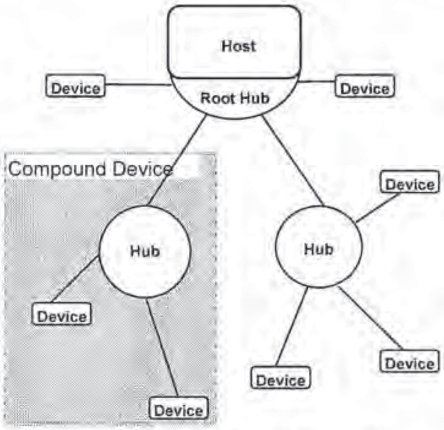


Figure 5-5. USB Physical Bus Topology

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.

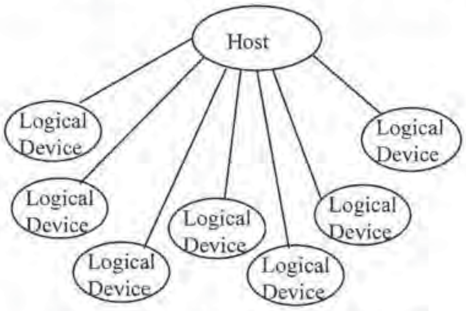


Figure 5-6. USB Logical Bus Topology

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.

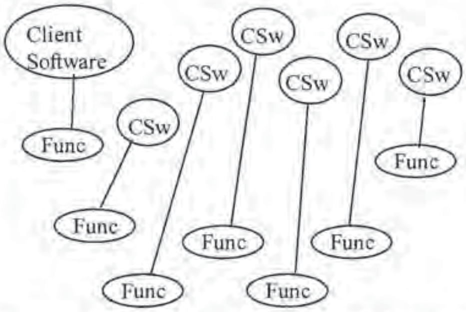


Figure 5-7. Client Software-to-function Relationships

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 (USBD): The interface between the USB System Software and the client software. This interface provides clients with convenient functions for manipulating USB devices.

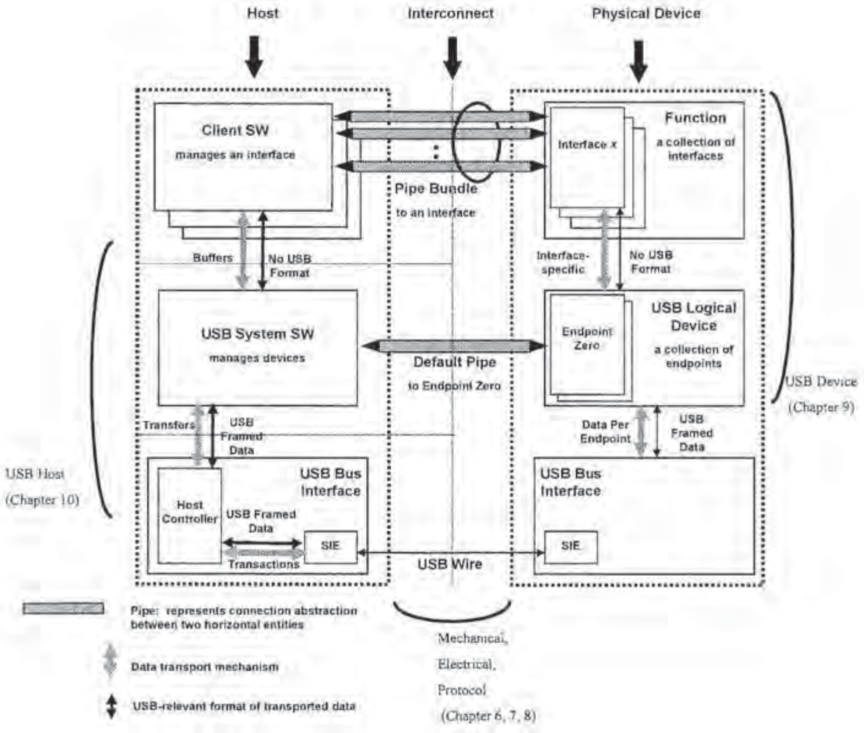


Figure 5-8. USB Host/Device Detailed View

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.

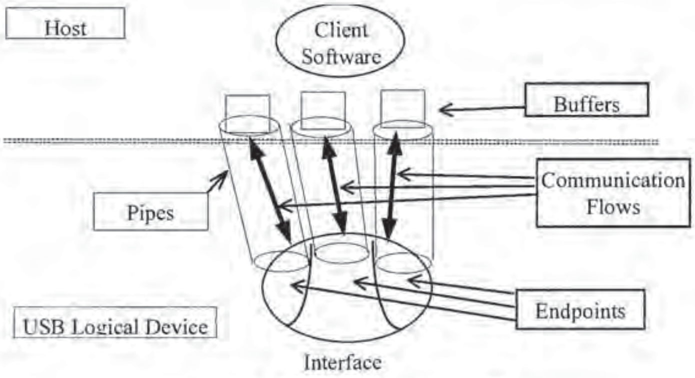


Figure 5-9. USB Communication Flow

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.

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 a 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.

The USB System Software determines the maximum data payload size that will be used for a 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.

Table 5-4. Full-speed Interrupt Transaction Limits

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

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 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

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

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.

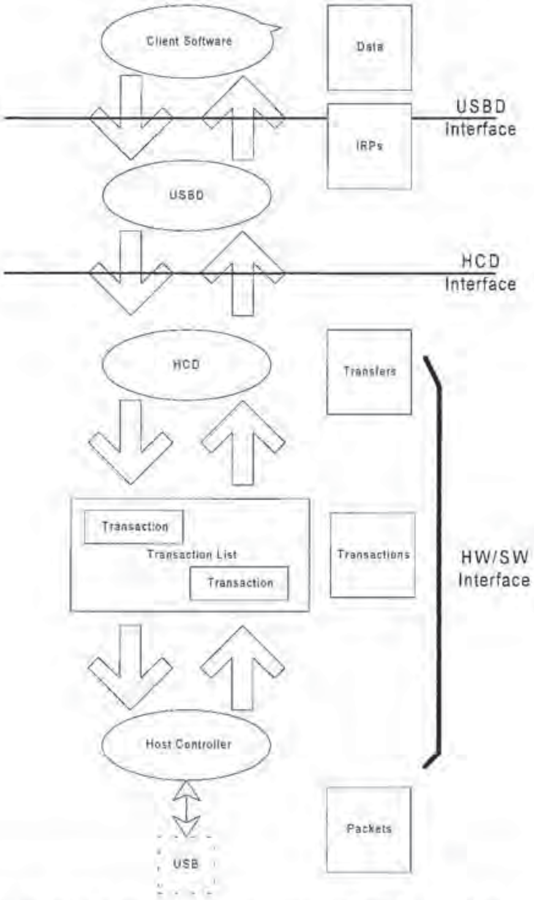


Figure 5-10. USB Information Conversion From Client Software to Bus

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 USBD interface.

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

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

The USBD interface is defined in Chapter 10.

5.9.1.2 USB Driver

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

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

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

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

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

5.9.1.3 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

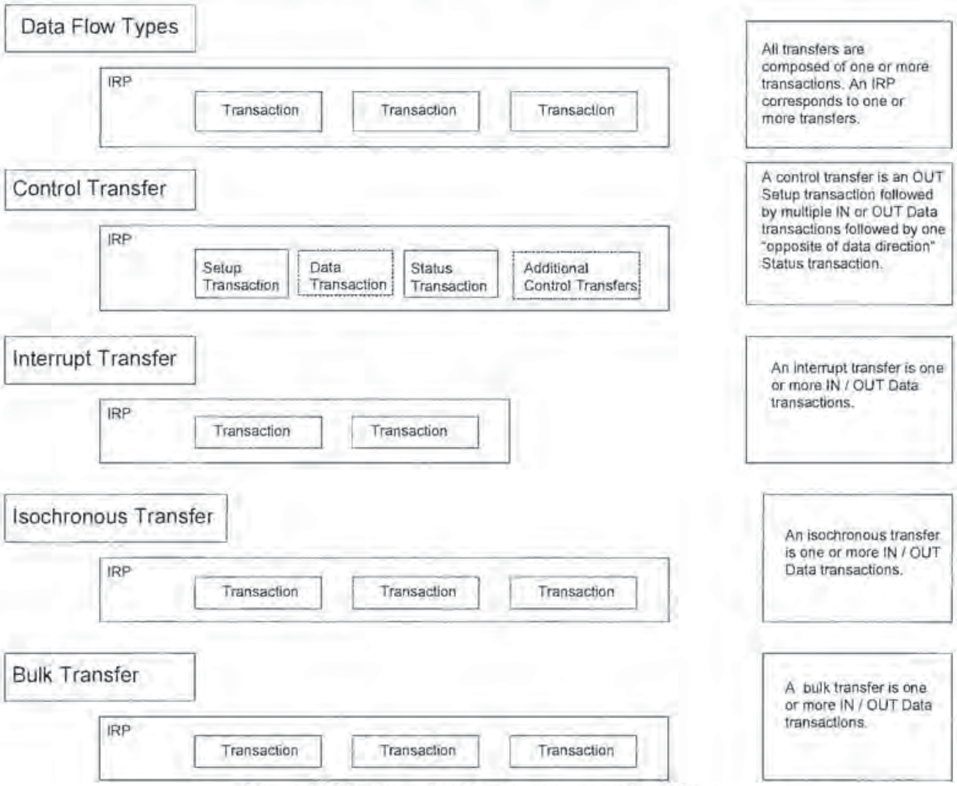
The Host Controller has access to the transaction list and translates it into bus activity. In addition, the Host Controller provides a reporting mechanism whereby the status of a transaction (done, pending, halted, etc.) can be obtained. The Host Controller converts transactions into appropriate implementation-dependent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

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 (USBD and HCD) provides support for matching data movement requirements of a client to packets on the bus. The Host Controller hardware and software uses IRPs to track information about one or more transactions that combine to deliver a transfer of information between the client software and the function. Figure 5-11 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.



All transfers are composed of one or more transactions. An IRP corresponds to one or more transfers.

A control transfer is an OUT Setup transaction followed by multiple IN or OUT Data transactions followed by one "opposite of data direction" Status transaction.

An interrupt transfer is one or more IN / OUT Data transactions.

An isochronous transfer is one or more IN / OUT Data transactions.

A bulk transfer is one or more IN / OUT Data transactions.

Figure 5-11. Transfers for Communication Flows

Even though IRPs track the bus transactions that need to occur to move a specific data flow over the USB, Host Controllers are free to choose how the particular bus transactions are moved over the bus subject to the USB-defined constraints (e.g., exactly one transaction per 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.

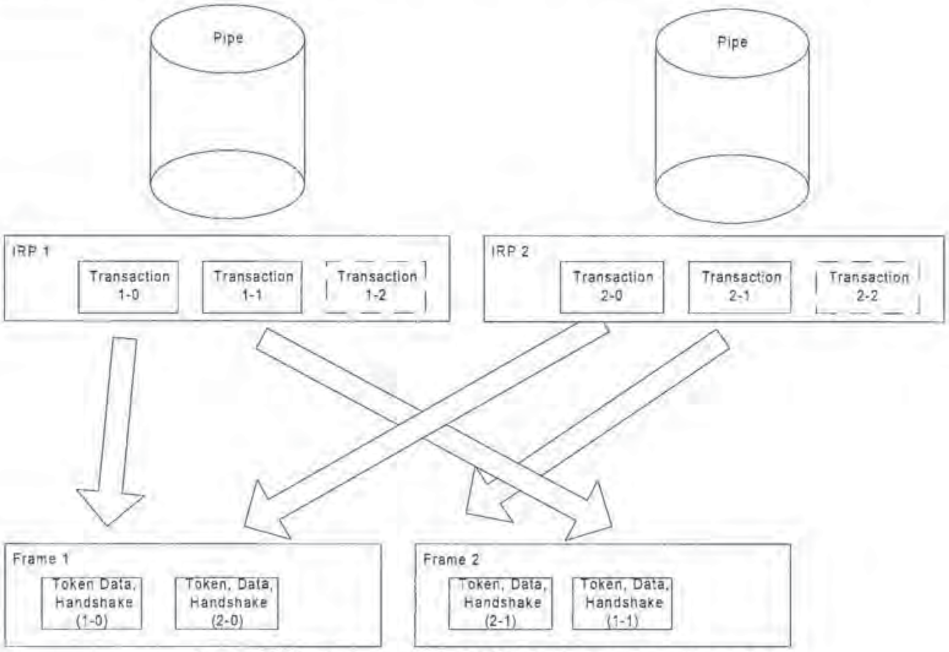


Figure 5-12. Arrangement of IRPs to Transactions/Frames

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*Data_bc)`

```
Full-speed (Input)

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

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

Full-speed (Output)

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

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

Low-speed (Input)

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

Low-speed (Output)

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

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

The actual bus time taken for a given transaction will almost always be less than that calculated because bit stuffing overhead is data-dependent. Worst case bit stuffing is calculated as 1.1667 (7/6) times the raw time (i.e., the BitStuffTime function multiplies the Data_bc by 8*1.1667 in the equations). This means that there will almost always be time unused on the bus (subject to data pattern specifics) after all regularly scheduled transactions have completed. The bus time made available due to less bit stuffing can be reused as discussed in Section 5.9.5.

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 USBD and HCD software working in cooperation. The results of these calculations are used to determine whether a transfer or pipe creation can be supported in a given USB configuration.

5.9.4 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 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.

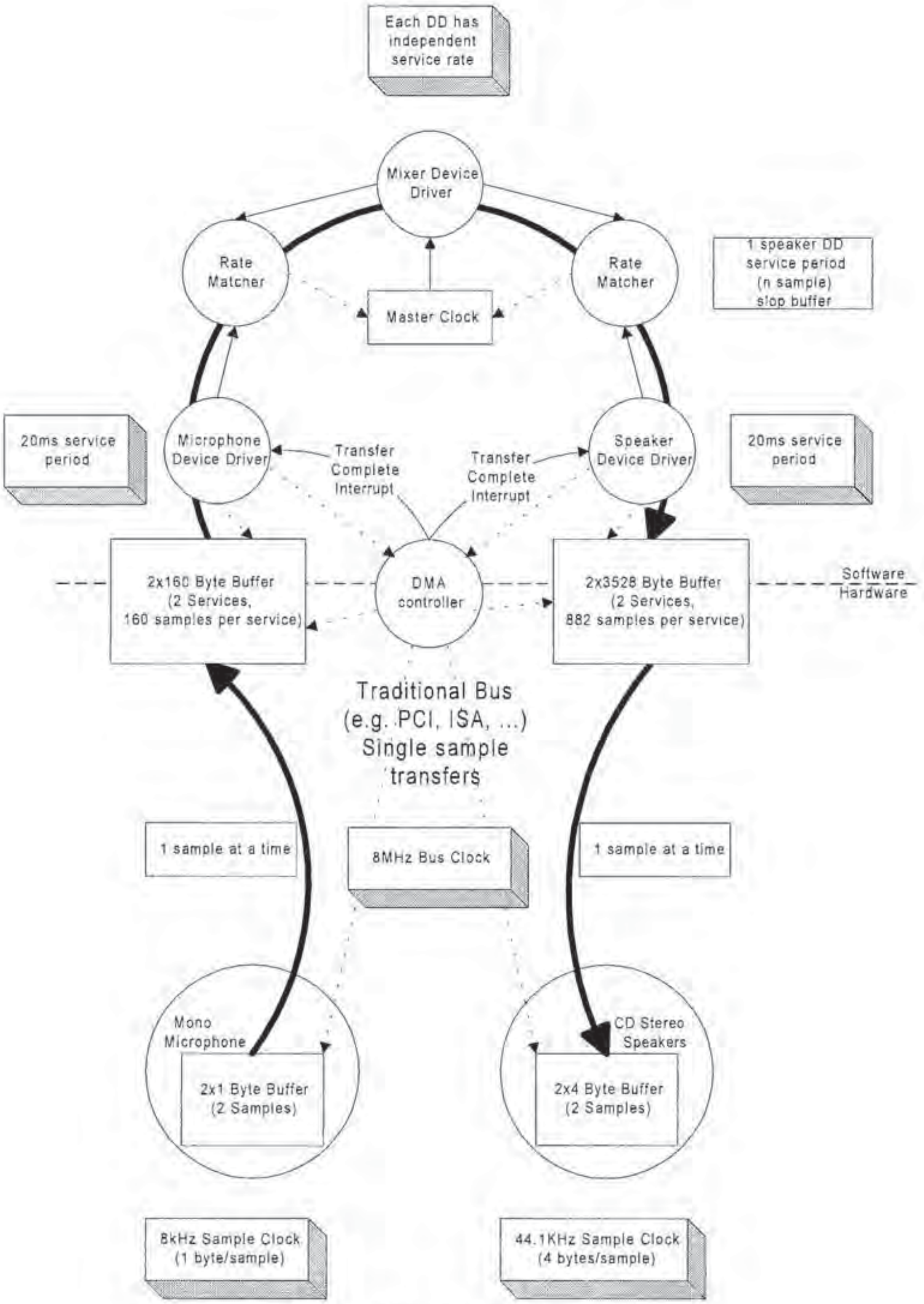


Figure 5-13. Non-USB Isochronous Example

A master clock (which can be provided by software driven from the real time clock) in the PC is used to awaken the mixer to ask the input source for input data and to provide output data to the output sink. In this example, assume it awakens every 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.

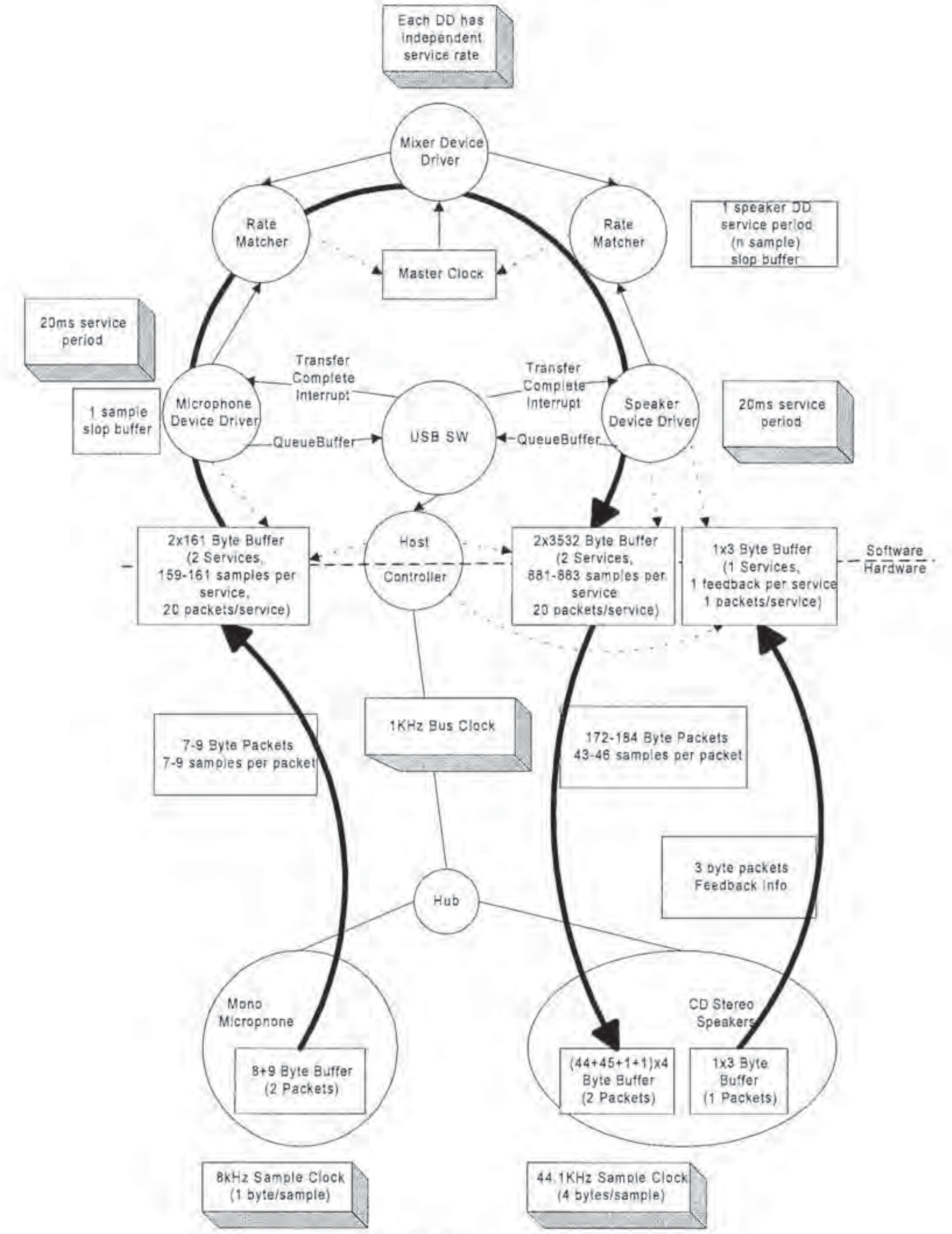


Figure 5-14. USB Isochronous Application

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.

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**
Asynchronous	Free running F_s Provides implicit feedforward (data stream)	Free running F_s Provides explicit feedback (interrupt pipe)
Synchronous	F_s locked to SOF Uses implicit feedback (SOF)	F_s locked to SOF Uses implicit feedback (SOF)
Adaptive	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).

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 (Ff) 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 Ff 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 Ff 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 Ff, 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 Ff 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 Ff over many frames to determine an even more accurate rate, if it needs to.

The sink can determine Ff by counting cycles of a clock with a frequency of Fs * 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 Fs, and no point in trying to update more than once a frame. The counter is read into Ff 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 Ff.

A digital telephony endpoint, for example, will usually derive its 8kHz Fs 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 Ff updates every 2^(10-4) = 64 frames. A 13-bit counter would be required to obtain Ff, 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 Ff 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 Ff is updated. More frequent updates result in a tighter control of the source data rate, which reduces the buffer space required to handle Ff changes.
- P should be less than ten so that Ff 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 Ff value.

Isochronous transfers are used to read Ff from the feedback register. The desired reporting rate for the feedback should be 2^(10-P) frames. Ff will be reported at most once per update period. There is nothing to be gained by reporting the same Ff value more than once per update period. The endpoint may choose to report Ff only if the updated value has changed from the previous Ff 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 Ff. The sink must have sufficient buffer capability to accommodate this. When the sink recognizes this condition, it should adjust the reported Ff 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.

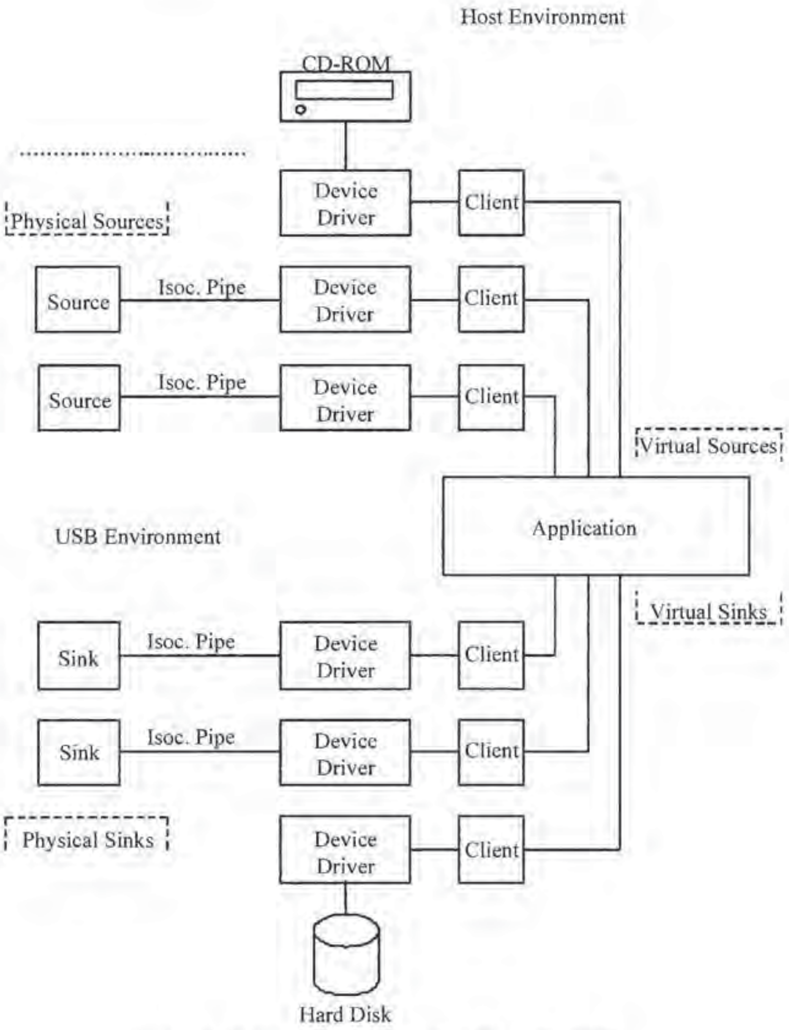


Figure 5-15. Example Source/Sink Connectivity

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."

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.

Table 5-8. Connection Requirements

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

Notes:

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

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

5.10.4.3.1 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 T_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}.

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.

- 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.

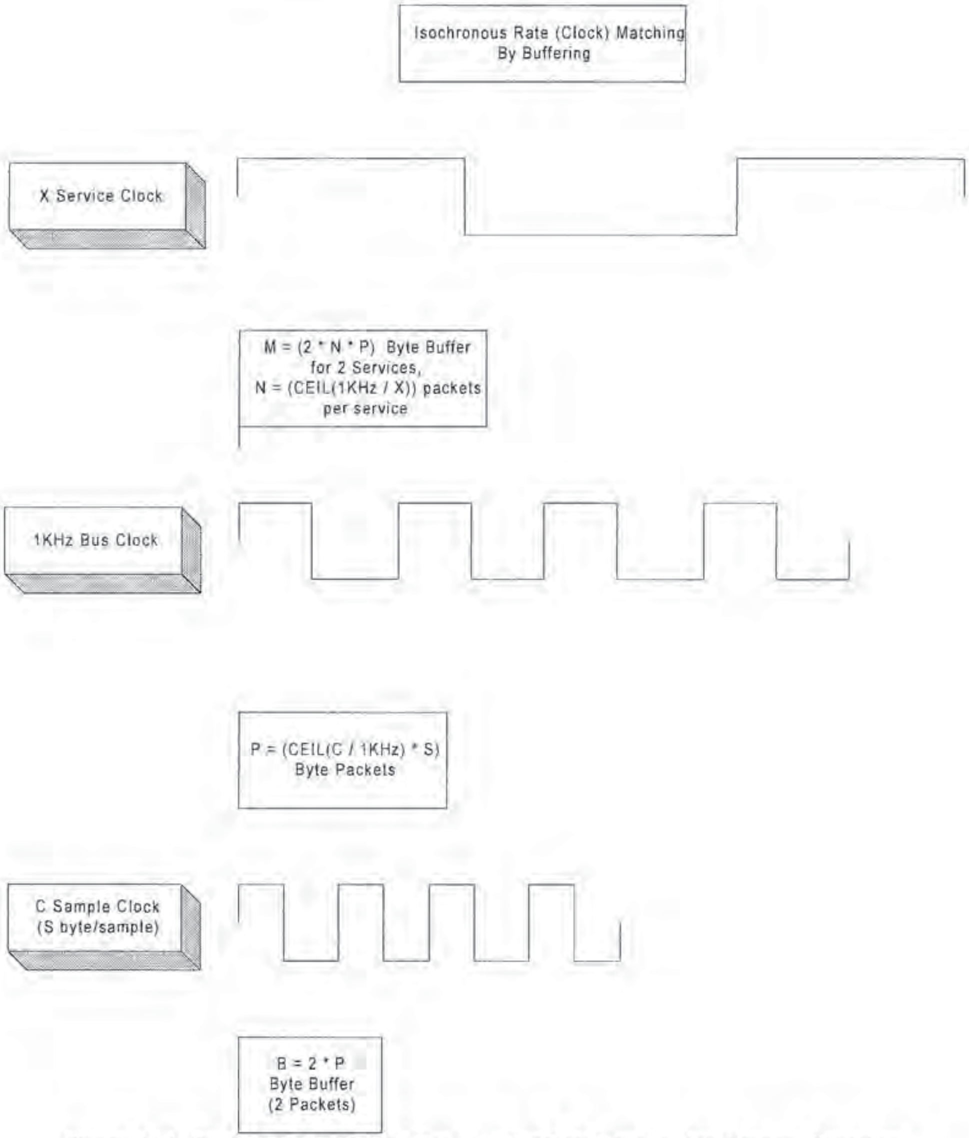


Figure 5-17. Packet and Buffer Size Formulas for Rate-Matched Isochronous Transfers

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.

Series "A" Connectors	Series "B" Connectors
◆ Series "A" plugs are always oriented **upstream** towards the *Host System*  **"A" Plugs** *(From the USB Device)* **"A" Receptacles** *(Downstream Output from the USB Host or Hub)* 	◆ Series "B" plugs are always oriented **downstream** towards the *USB Device*  **"B" Plugs** *(From the Host System)* **"B" Receptacles** *(Upstream Input to the USB Device or Hub)* 

Figure 6-1. Keyed Connector Protocol

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

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

6.3 Cable

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

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.

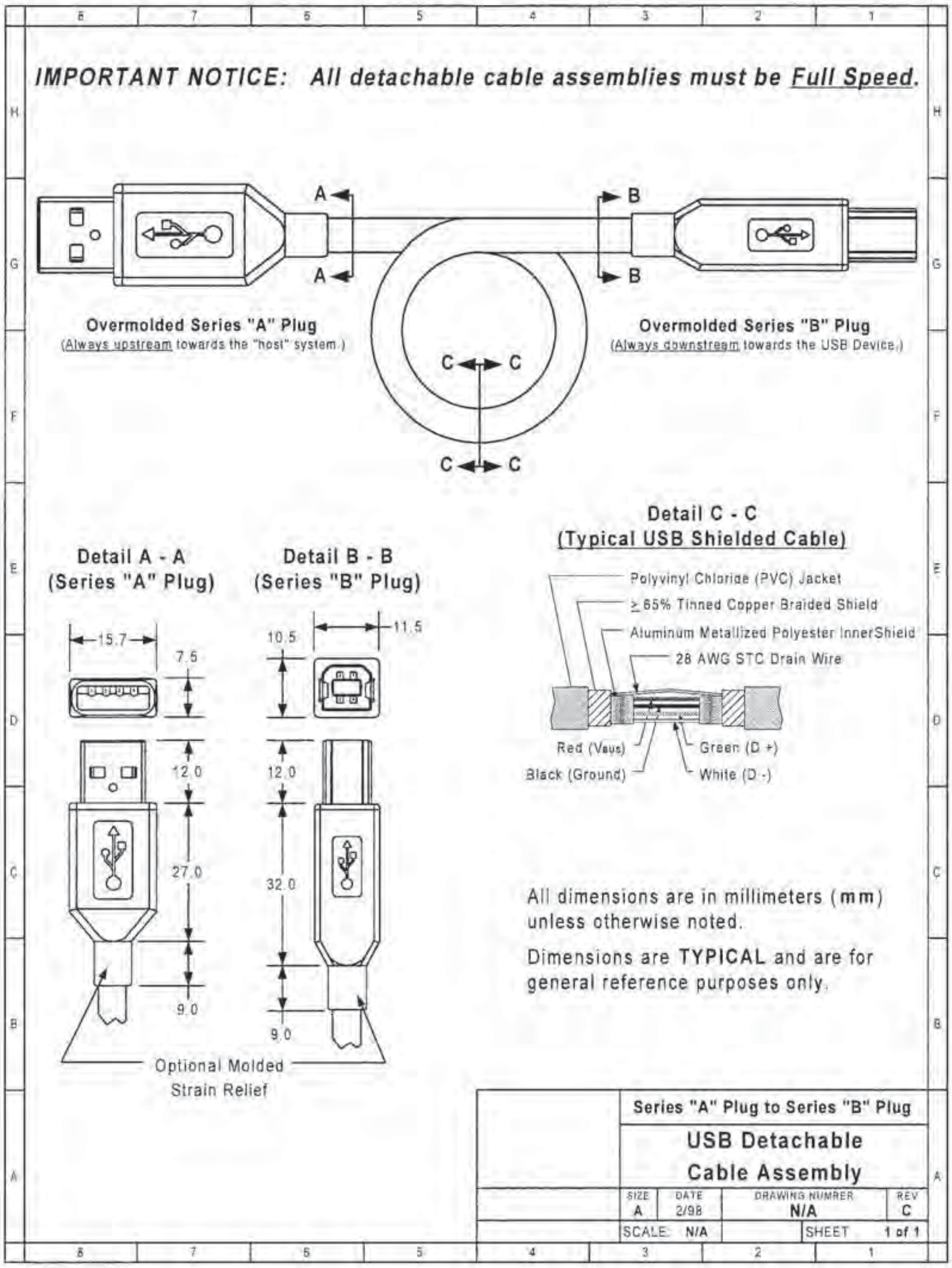


Figure 6-2. USB Detachable Cable Assembly

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.

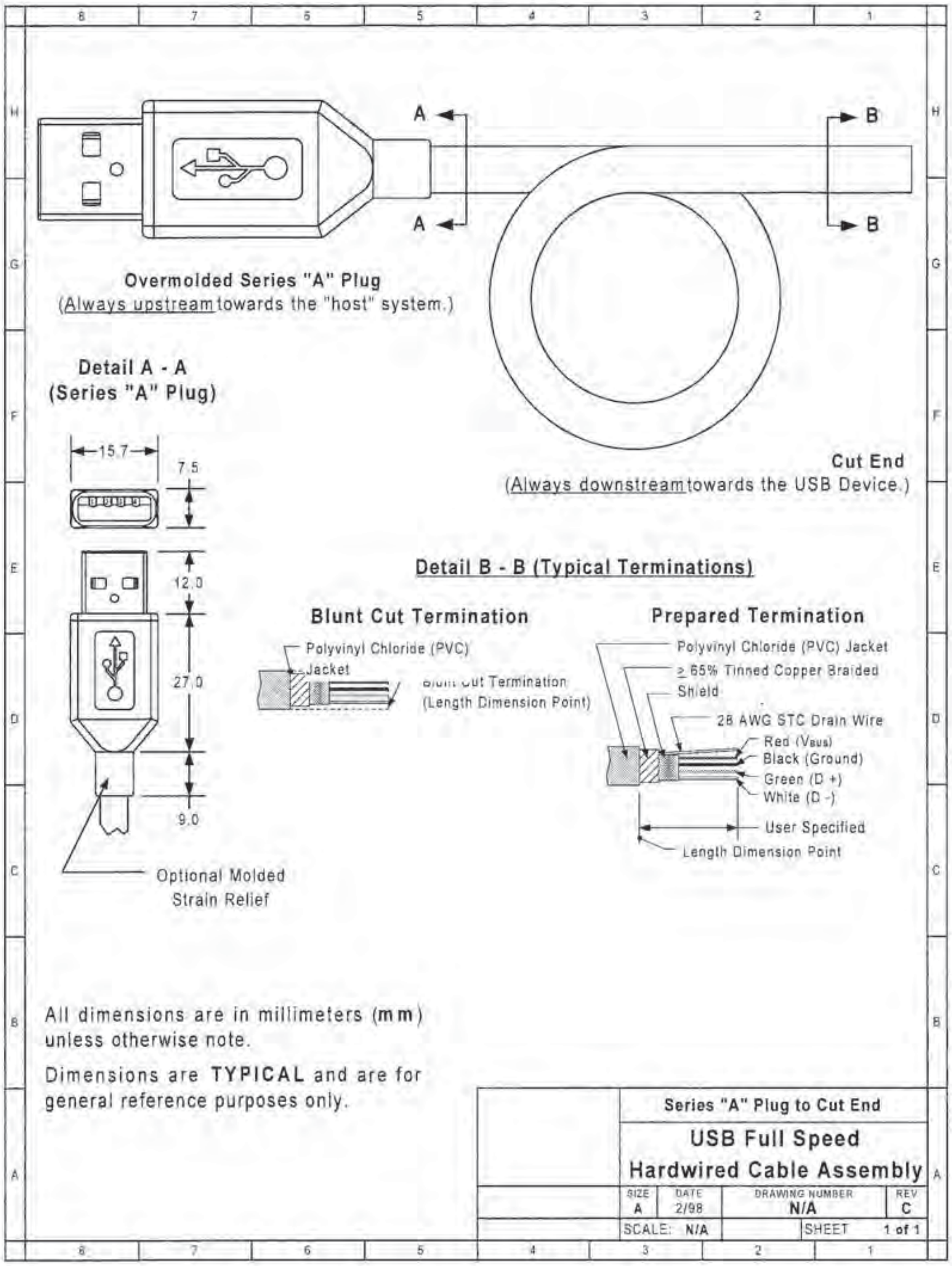


Figure 6-3. USB Full-speed Hardwired Cable Assembly

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 though 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.

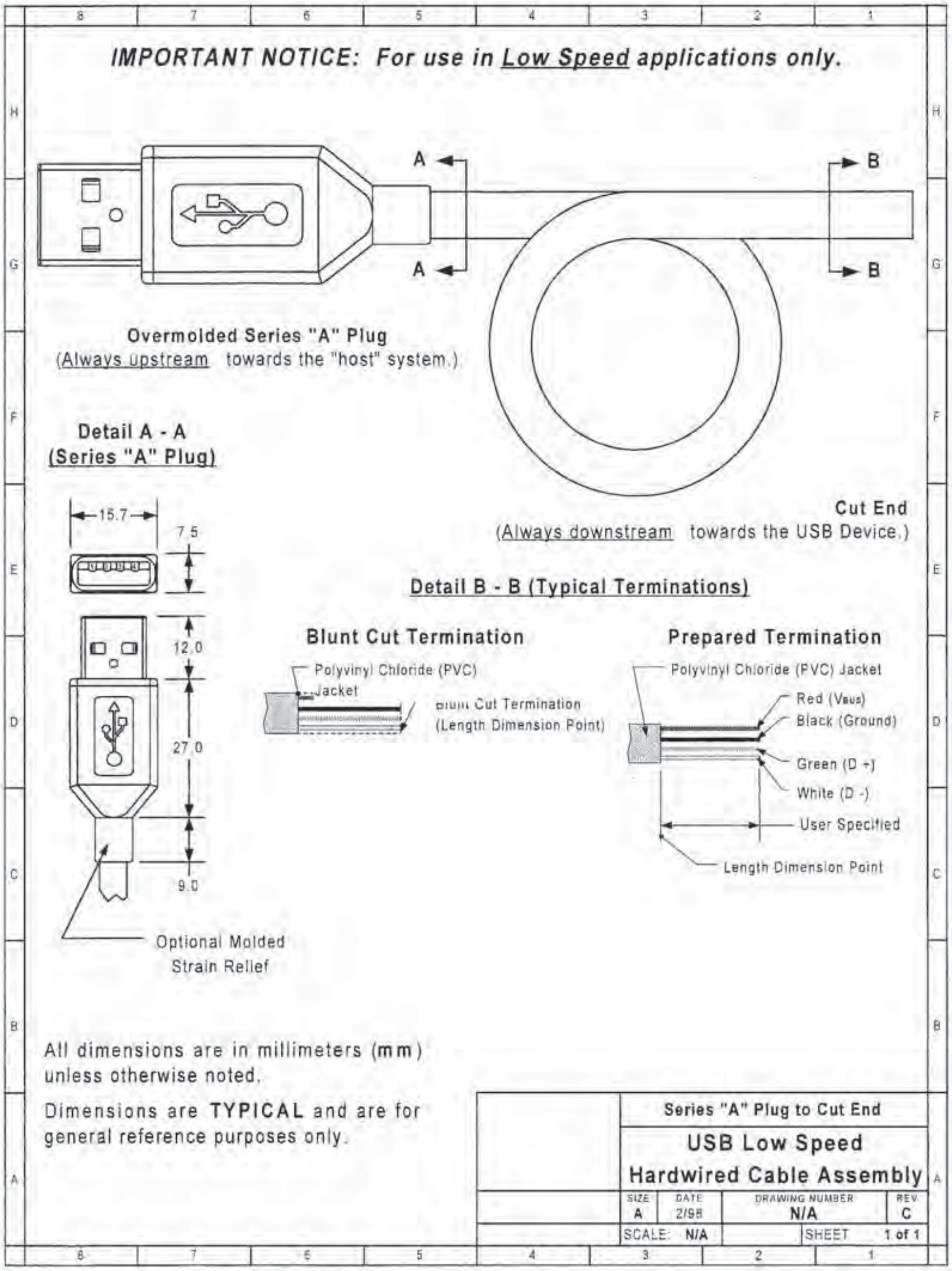


Figure 6-4. USB Low-speed Hardwired Cable Assembly

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 then 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 it 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.

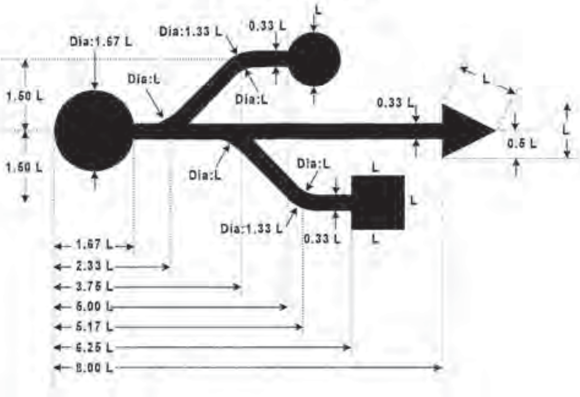
Non-acceptable cables may work in some situations but they cannot be guaranteed to work in all instances.

- ***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



All dimensions are ± 5%

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 Manufacture's logo should not project beyond the overmold surface. The USB Icon is required, while the Manufacture'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.

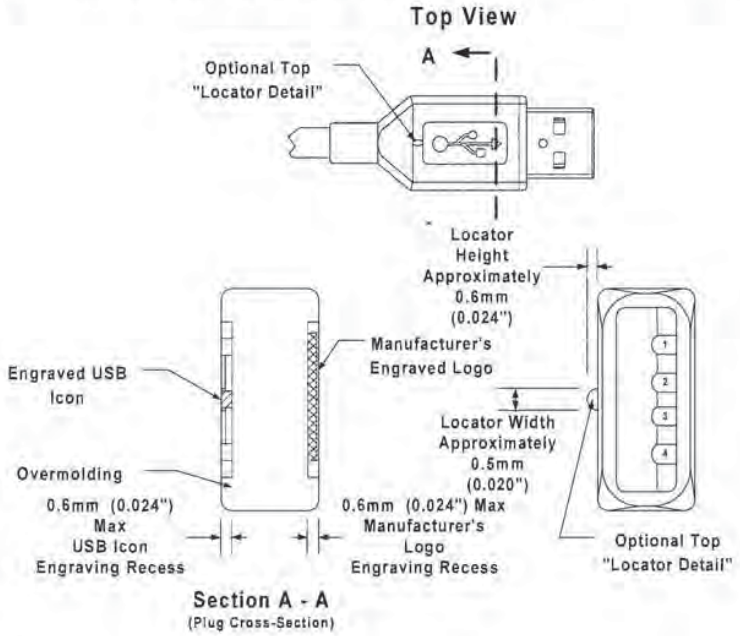


Figure 6-6. Typical USB Plug Orientation

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.

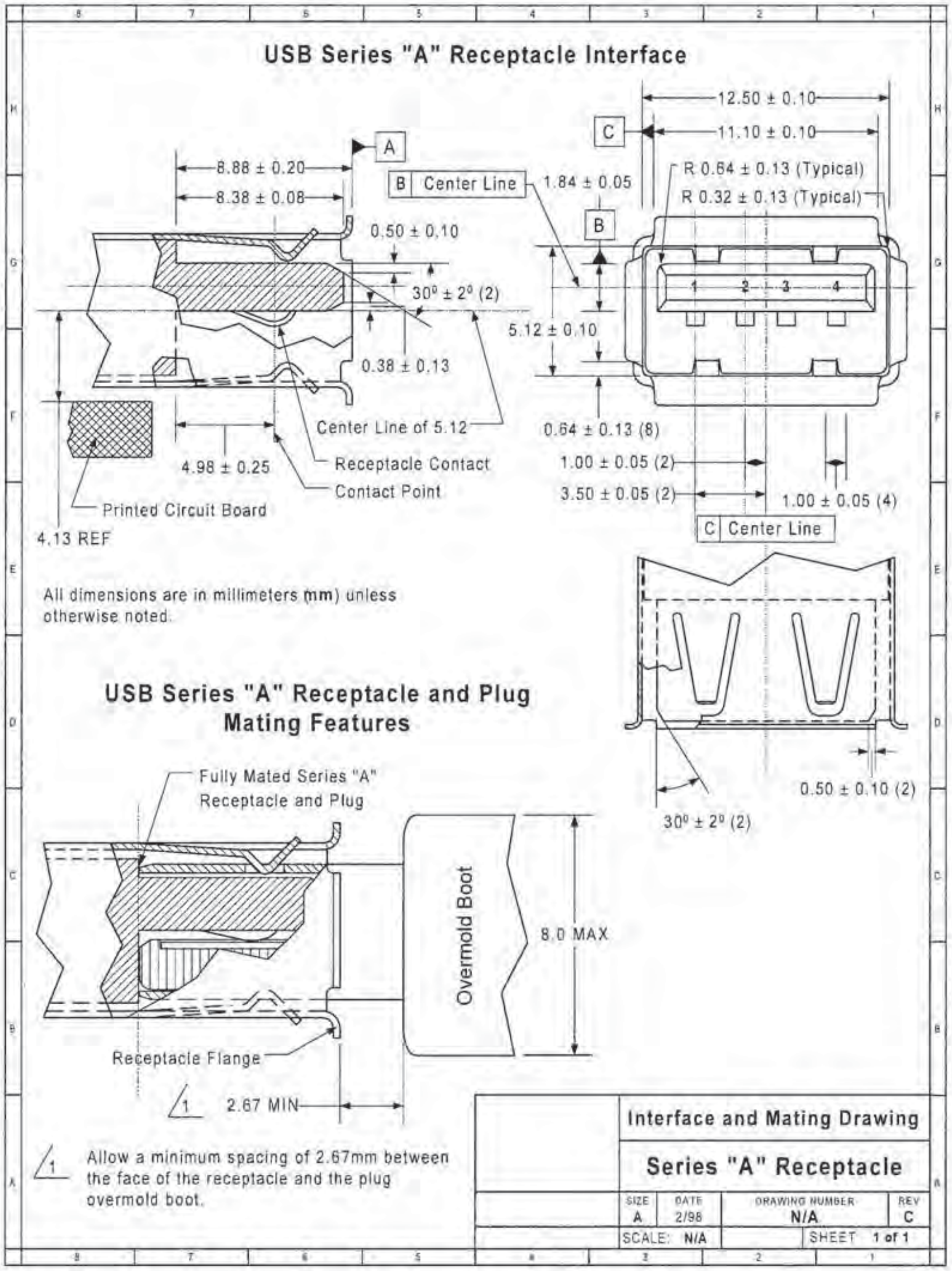


Figure 6-7. USB Series "A" Receptacle Interface and Mating Drawing

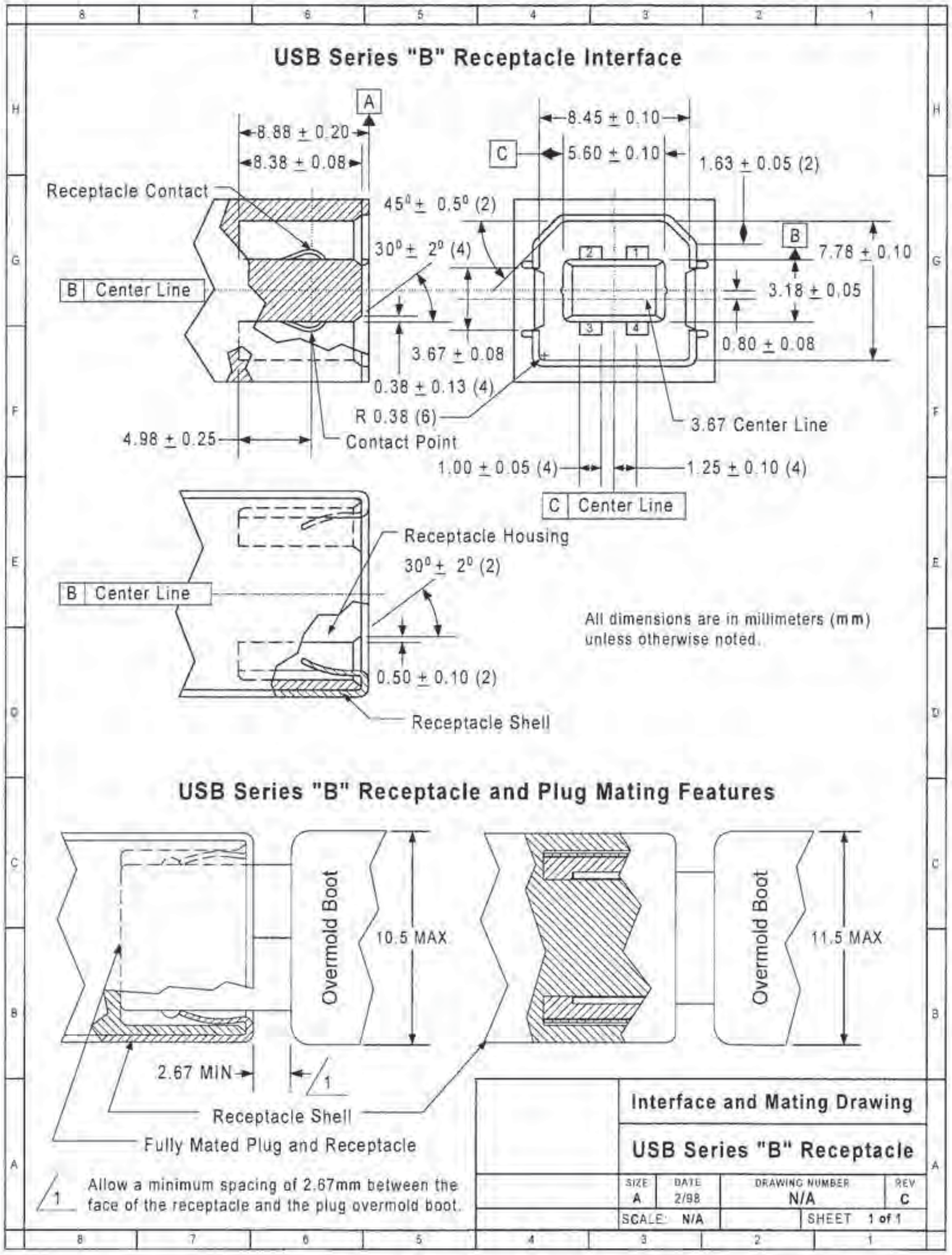


Figure 6-8. USB Series "B" Recptacle Interface and Mating Drawing

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.

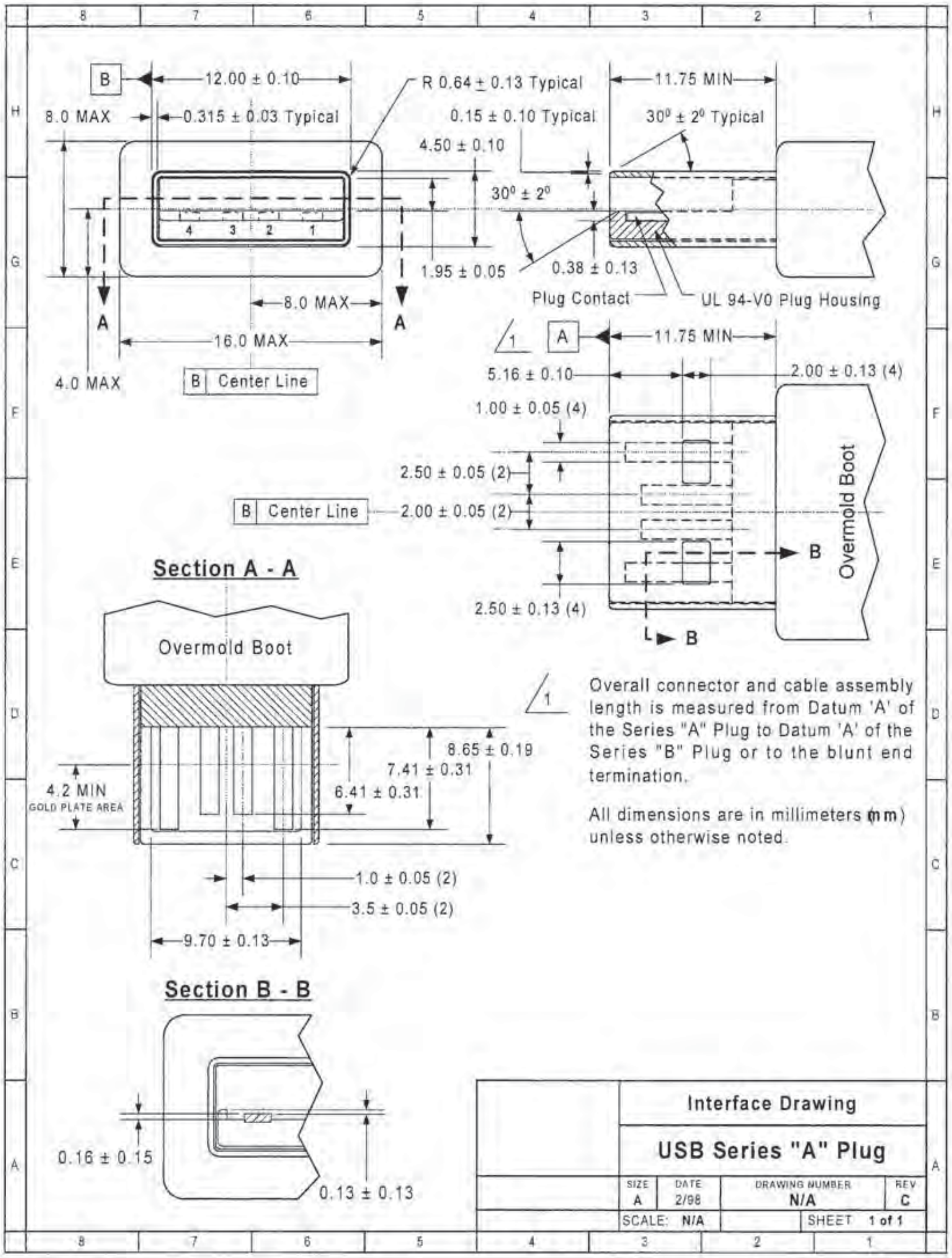
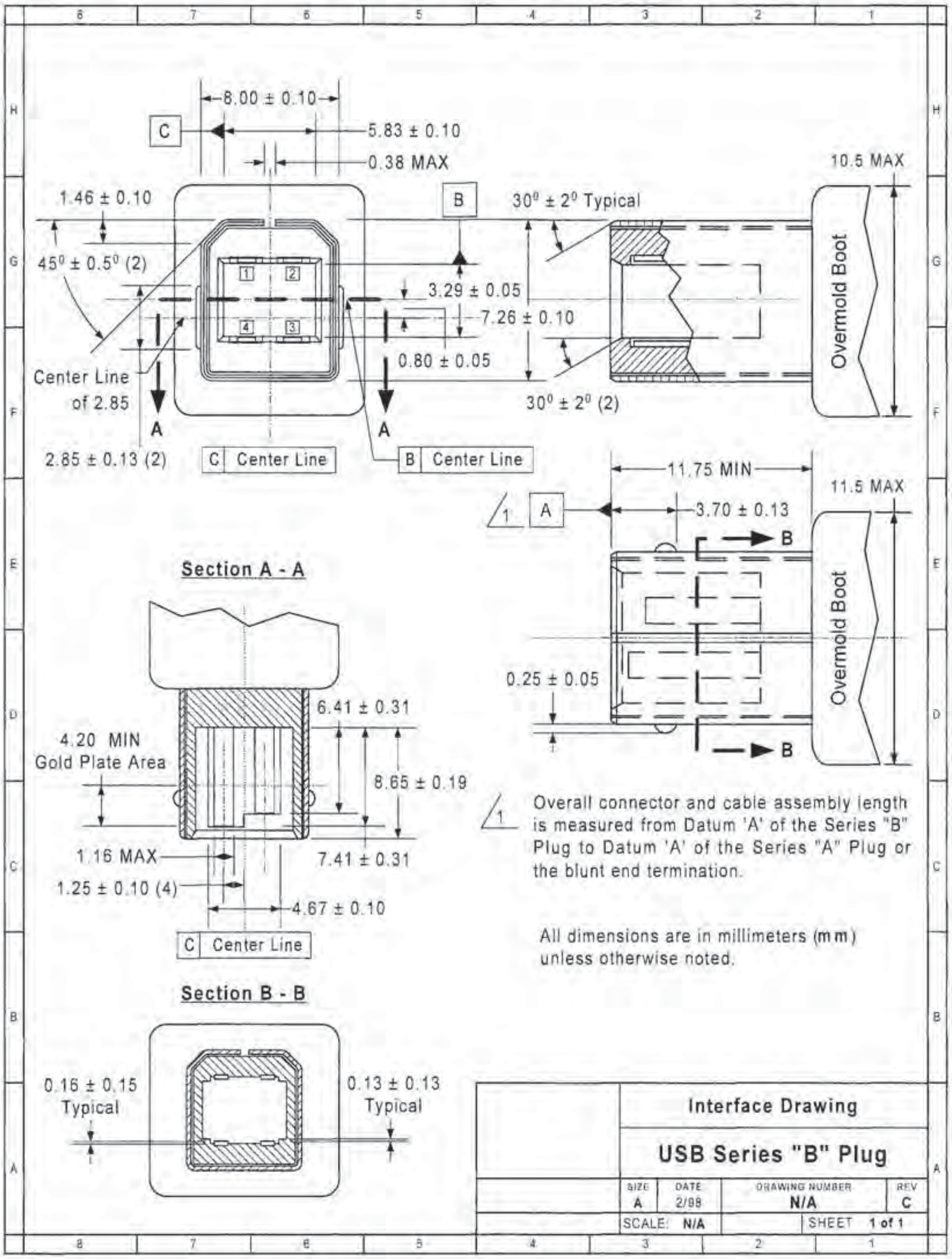


Figure 6-9. USB Series "A" Plug Interface Drawing



1 Overall connector and cable assembly length is measured from Datum 'A' of the Series "B" Plug to Datum 'A' of the Series "A" Plug or the blunt end termination.

All dimensions are in millimeters (mm) unless otherwise noted.

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

6.5.4.1 Plug Injection Molded Thermoplastic Insulator Material

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

Typical Colors: Black, Gray and Natural.

Flammability Characteristics: UL 94-V0 rated.

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

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

6.5.4.2 Plug Shell Materials

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

Plating:

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

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

6.5.4.3 Plug (Male) Contact Materials

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

Plating. Contacts are to be selectively plated.

A. Option I

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

B. Option II

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

C. Option III

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

6.6 Cable Mechanical Configuration and Material Requirements

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.

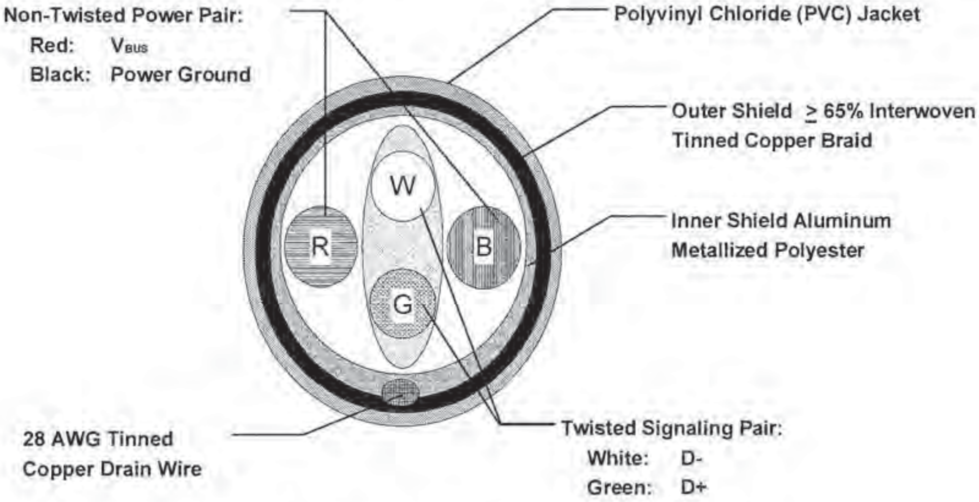


Figure 6-11. Typical Full-speed Cable Construction

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, ≥ 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

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")	7 x 36
	0.406 mm (0.016")	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.

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 (Ω) / 100 Meters Maximum
28	23.20 Ω
26	14.60 Ω
24	9.09 Ω
22	5.74 Ω
20	3.58 Ω

6.6.4 Cable Environmental Characteristics

Temperature Range:

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

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

C. Nominal Temperature Rating: +20° C.

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

6.6.5 Listing

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

6.7 Electrical, Mechanical and Environmental Compliance Standards

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

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

Test Description	Test Procedure	Performance Requirement
Visual and Dimensional Inspection	**EIA 364-18** Visual, dimensional and functional inspection in accordance the USB quality inspection plans.	Must meet or exceed the requirements specified by the most current version of Chapter 6 of the USB Specification.
Insulation Resistance	**EIA 364-21** The object of this test procedure is to detail a standard method to assess the insulation resistance of USB connectors. This test procedure is used to determine the resistance offered by the insulation materials and the various seals of a connector to a DC potential tending to produce a leakage of current through or on the surface of these members.	1,000 MΩ minimum.
Dielectric Withstanding Voltage	**EIA 364-20** The object of this test procedure is to detail a test method to prove that a USB connector can operate safely at its rated voltage and withstand momentary over potentials due to switching, surges and/or other similar phenomena.	The dielectric must withstand 500 VAC for one minute at sea level.
Low Level Contact Resistance	**EIA 364-23** The object of this test is to detail a standard method to measure the electrical resistance across a pair of mated contacts such that the insulating films, if present, will not be broken or asperity melting will not occur.	30 mΩ maximum when measured at 20 mV maximum open circuit at 100 mA. Mated test contacts must be in a connector housing.

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

Test Description	Test Procedure	Performance Requirement
Contact Current Rating	**EIA 364-70 — Method B** The object of this test procedure is to detail a standard method to assess the current carrying capacity of mated USB connector contacts.	1.5 A at 250 VAC minimum when measured at an ambient temperature of 25° C. With power applied to the contacts, the Δ T shall not exceed +30° C at any point in the USB connector under test.
Contact Capacitance	**EIA 364-30** The object of this test is to detail a standard method to determine the capacitance between conductive elements of a USB connector.	2 pF maximum unmated per contact
Insertion Force	**EIA 364-13** The object of this test is to detail a standard method for determining the mechanical forces required for inserting a USB connector.	35 Newtons maximum at a maximum rate of 12.5 mm (0.492") per minute
Extraction Force	**EIA 364-13** The object of this test is to detail a standard method for determining the mechanical forces required for extracting a USB connector.	10 Newtons minimum at a maximum rate of 12.5 mm (0.492") per minute
Durability	**EIA 364-09** The object of this test procedure is to detail a uniform test method for determining the effects caused by subjecting a USB connector to the conditioning action of insertion and extraction, simulating the expected life of the connectors. Durability cycling with a gauge is intended only to produce mechanical stress. Durability performed with mating components is intended to produce both mechanical and wear stress.	1,500 insertion/extraction cycles at a maximum rate of 200 cycles per hour

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

Test Description	Test Procedure	Performance Requirement
Cable Pull-Out	**EIA 364-38** **Test Condition A** The object of this test procedure is to detail a standard method for determining the holding effect of a USB plug cable clamp without causing any detrimental effects upon the cable or connector components when the cable is subjected to inadvertent axial tensile loads.	After the application of a steady state axial load of 40 Newtons for one minute
Physical Shock	**EIA 364-27** **Test Condition H** The object of this test procedure is to detail a standard method to assess the ability of a USB connector to withstand specified severity of mechanical shock.	No discontinuities of 1 µS or longer duration when mated USB connectors are subjected to 11 ms duration 30 Gs half-sine shock pulses. Three shocks in each direction applied along three mutually perpendicular planes for a total of 18 shocks
Random Vibration	**EIA 364-28** **Test Condition V Test Letter A** This test procedure is applicable to USB connectors that may, in service, be subjected to conditions involving vibration. Whether a USB connector has to function during vibration or merely to survive conditions of vibration should be clearly stated by the detailed product specification. In either case, the relevant specification should always prescribe the acceptable performance tolerances.	No discontinuities of 1 µS or longer duration when mated USB connectors are subjected to 5.35 Gs RMS. 15 minutes in each of three mutually perpendicular planes

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

Test Description	Test Procedure	Performance Requirement
Thermal Shock	**EIA 364-32** **Test Condition I** The object of this test is to determine the resistance of a USB connector to exposure at extremes of high and low temperatures and to the shock of alternate exposures to these extremes, simulating the worst case conditions for storage, transportation and application.	10 Cycles -55°C and $+85^{\circ}$C. The USB connectors under test must be mated
Humidity Life	**EIA 364-31** **Test Condition A Method III** The object of this test procedure is to detail a standard test method for the evaluation of the properties of materials used in USB connectors as they are influenced by the effects of high humidity and heat.	168 Hours minimum (seven (7) complete cycles). The USB connectors under test shall be tested in accordance with EIA 364-31
Solderability	**EIA 364-52** The object of this test procedure is to detail a uniform test method for determining USB connector solderability. The test procedure contained herein utilizes the solder dip technique. It is not intended to test or evaluate solder cup, solder eyelet, other hand-soldered type or SMT type terminations.	USB contact solder tails shall pass 95% coverage after one hour steam aging as specified in Category 2
Flammability	**UL 94 V-0** This procedure is to ensure thermoplastic resin compliance to UL flammability standards.	The manufacturer will require its thermoplastic resin vendor to supply a detailed C of C with each resin shipment. The C of C shall clearly show the resin's UL listing number, lot number, date code, et cetera.

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

Test Description	Test Procedure	Performance Requirement
Cable Imedance **Only required for Full-speed**	The object of this test is to insure the signal conductors have the proper impedance. 1. Connect the Time Domain Reflectometer (TDR) outputs to the impedance/delay/skew test fixture (Note 1). Use separate 50 Ohm cables for the plus (or true) and minus (or complement) outputs. Set the TDR head to differential TDR mode. 2. Connect the Series "A" plug of the cable to be tested to the text fixture, leaving the other end open-circuited. 3. Define a waveform composed of the difference between the true and complement waveforms, to allow measurement of differential impedance. 4. Measure the minimum and maximum impedances found between the connector and the open circuited far end of the cable.	Impedance must be in the range specified in Table 7-9 (ZO)

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

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

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

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

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

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

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

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

should be used on the other end of the cable under test, to convert the signal back to unbalanced 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.

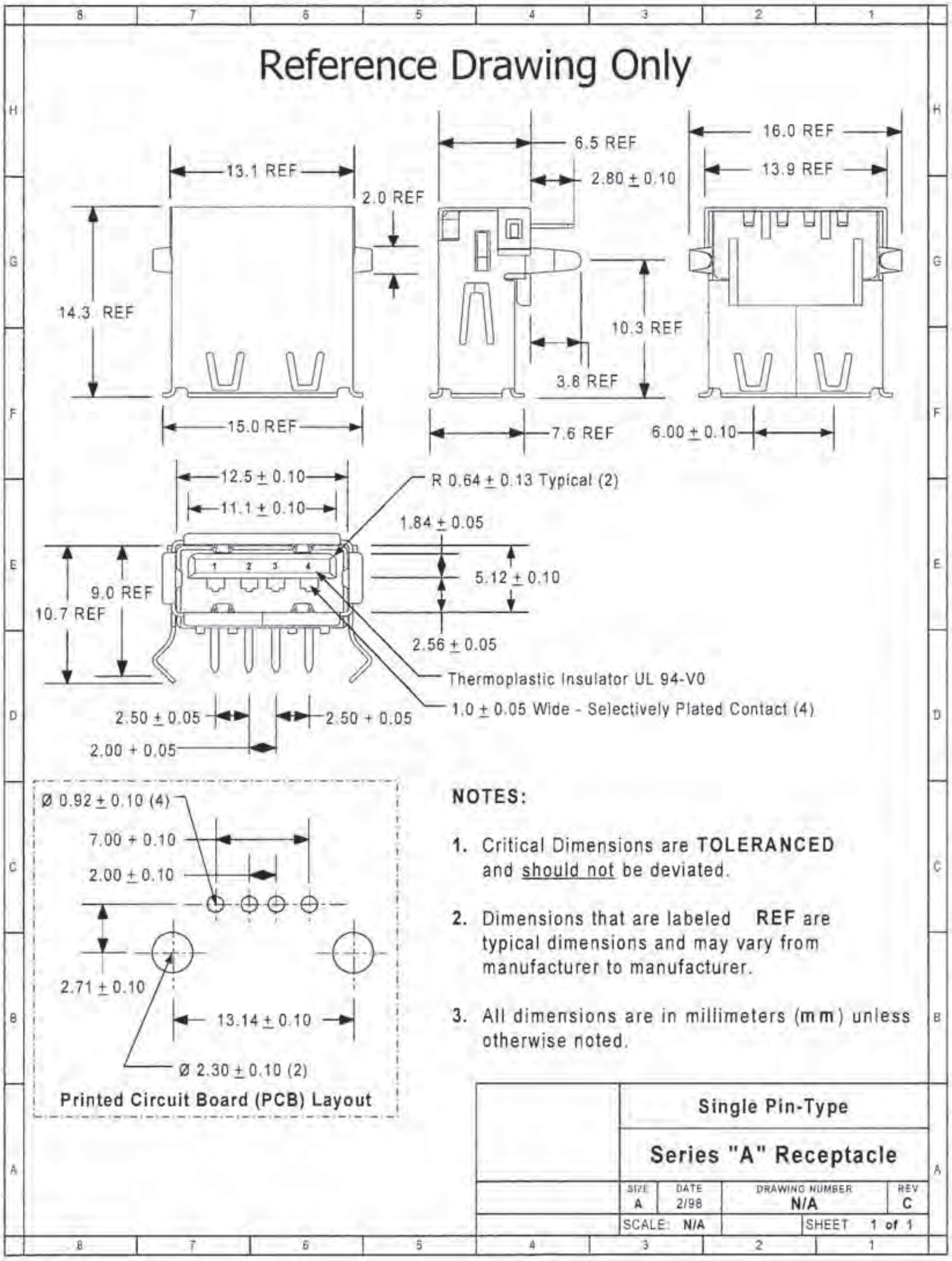
UL STD-94	Test for Flammability of Plastic materials for Parts in Devices and Appliances
UL Subject-444	Communication Cables

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

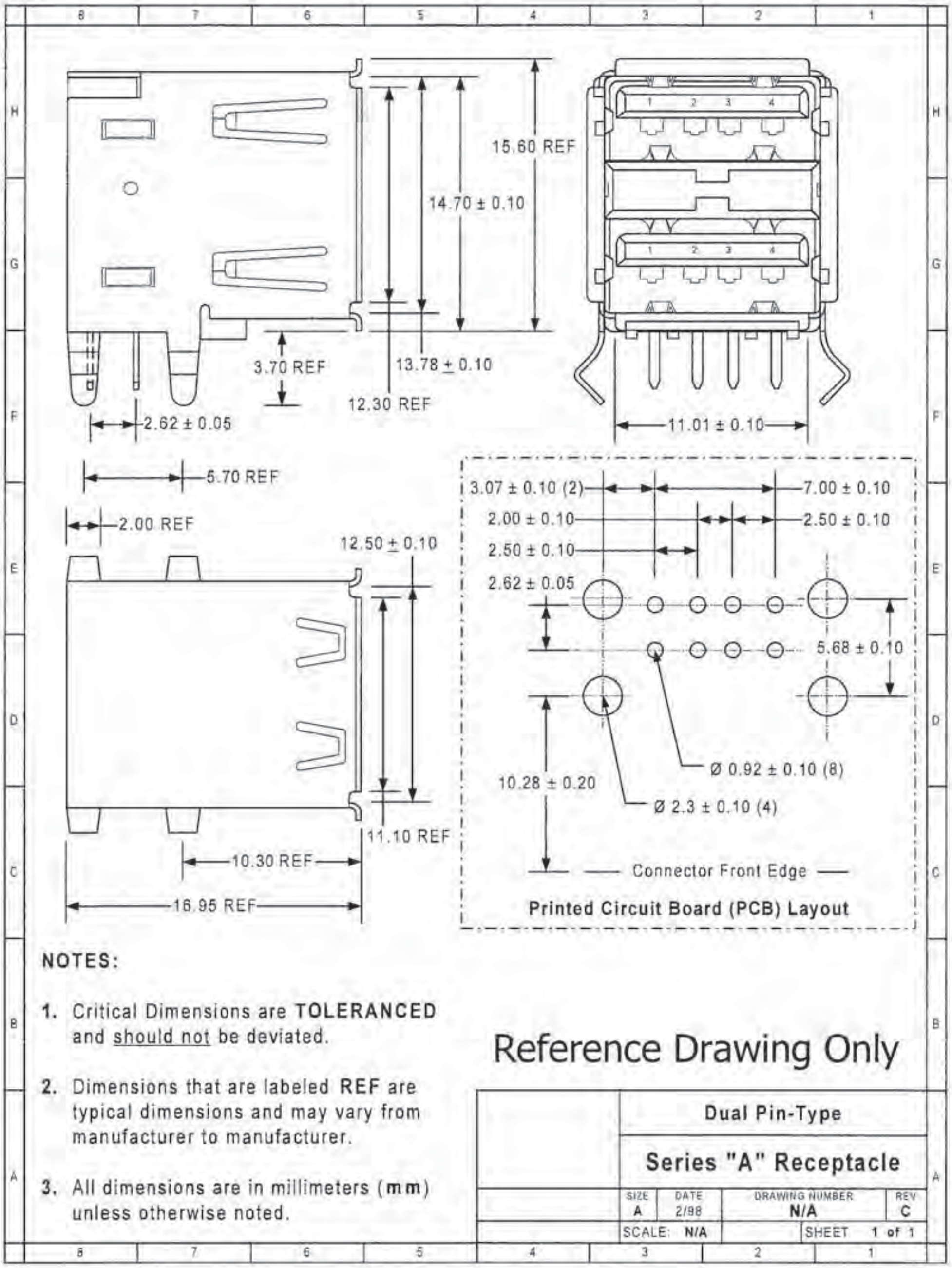
The following drawings describe typical receptacle PCB interfaces. This is included for information purposes only.



NOTES:

1. Critical Dimensions are **TOLERANCED** and <u>should not</u> be deviated.
2. Dimensions that are labeled **REF** are typical dimensions and may vary from manufacturer to manufacturer.
3. All dimensions are in millimeters (mm) unless otherwise noted.

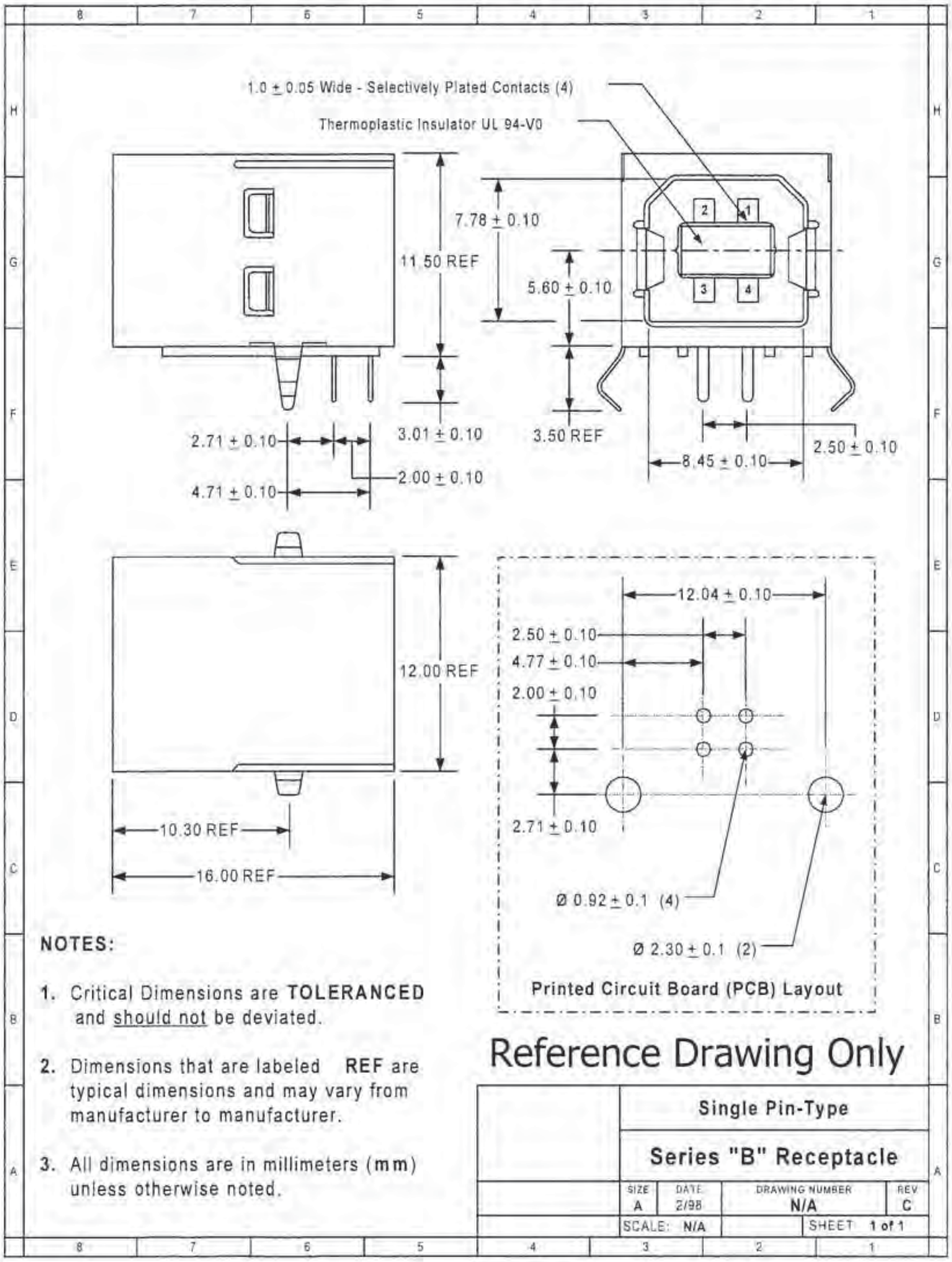
Figure 6-12. Single Pin-Type Series "A" Receptacle



NOTES:

1. Critical Dimensions are **TOLERANCED** and should not be deviated.
2. Dimensions that are labeled **REF** are typical dimensions and may vary from manufacturer to manufacturer.
3. All dimensions are in millimeters (**mm**) unless otherwise noted.

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



NOTES:

1. Critical Dimensions are **TOLERANCED** and <u>should not</u> be deviated.
2. Dimensions that are labeled **REF** are typical dimensions and may vary from manufacturer to manufacturer.
3. All dimensions are in millimeters (**mm**) unless otherwise noted.

Single Pin-Type

Series "B" Receptacle

SIZE	DATE	DRAWING NUMBER	REV
A	2/98	N/A	C

SCALE: N/A

SHEET 1 of 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

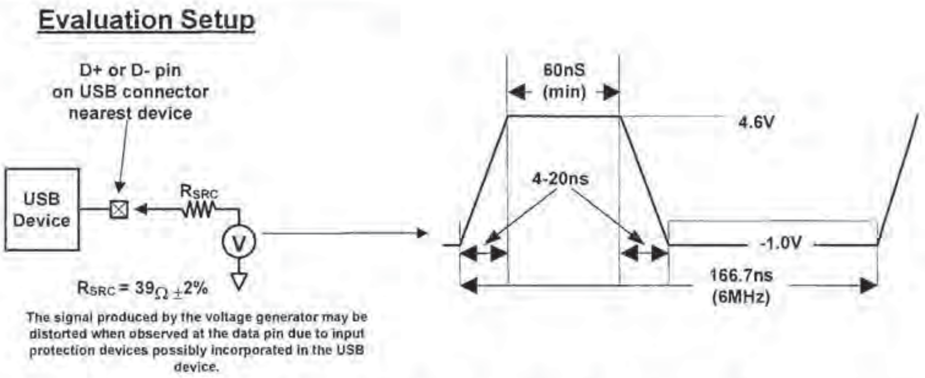
The signaling specification for the USB is described in the following subsections.

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 VOL (max) of 0.3V with a 1.5kΩ load to 3.6V and in its high state must be above the VOH (min) of 2.8V with a 15kΩ 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 39Ω. 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.



The signal produced by the voltage generator may be distorted when observed at the data pin due to input protection devices possibly incorporated in the USB device.

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 VBUS, 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 VBUS.

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 90Ω ±15% and a maximum one-way delay of 26ns. The impedance of each of the drivers (ZDRV) must be between 28Ω and 44Ω 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 RS 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.

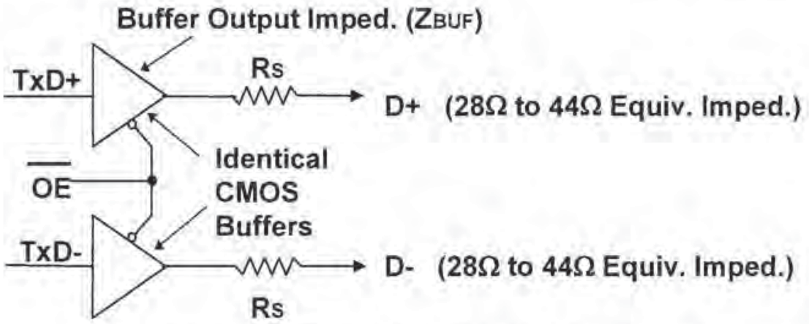


Figure 7-2. Example Full-speed CMOS Driver Circuit

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 (RS). The characteristics are normalized to the steady-state, unloaded output swing of the driver. The normalized driver characteristics are found by dividing the measured voltages and currents by the actual swing of the driver under test. The normalized V/I curve for the driver must fall entirely inside the shaded region. The V/I region is bounded by the minimum driver impedance above and the maximum driver impedance below. The minimum drive region is intersected by a constant current region of |6.1VOH|mA when driving low and -|6.1VOH|mA when driving high. This is the minimum current drive level necessary to ensure that the waveform at the receiver crosses the opposite single-ended switching level on the first reflection.

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

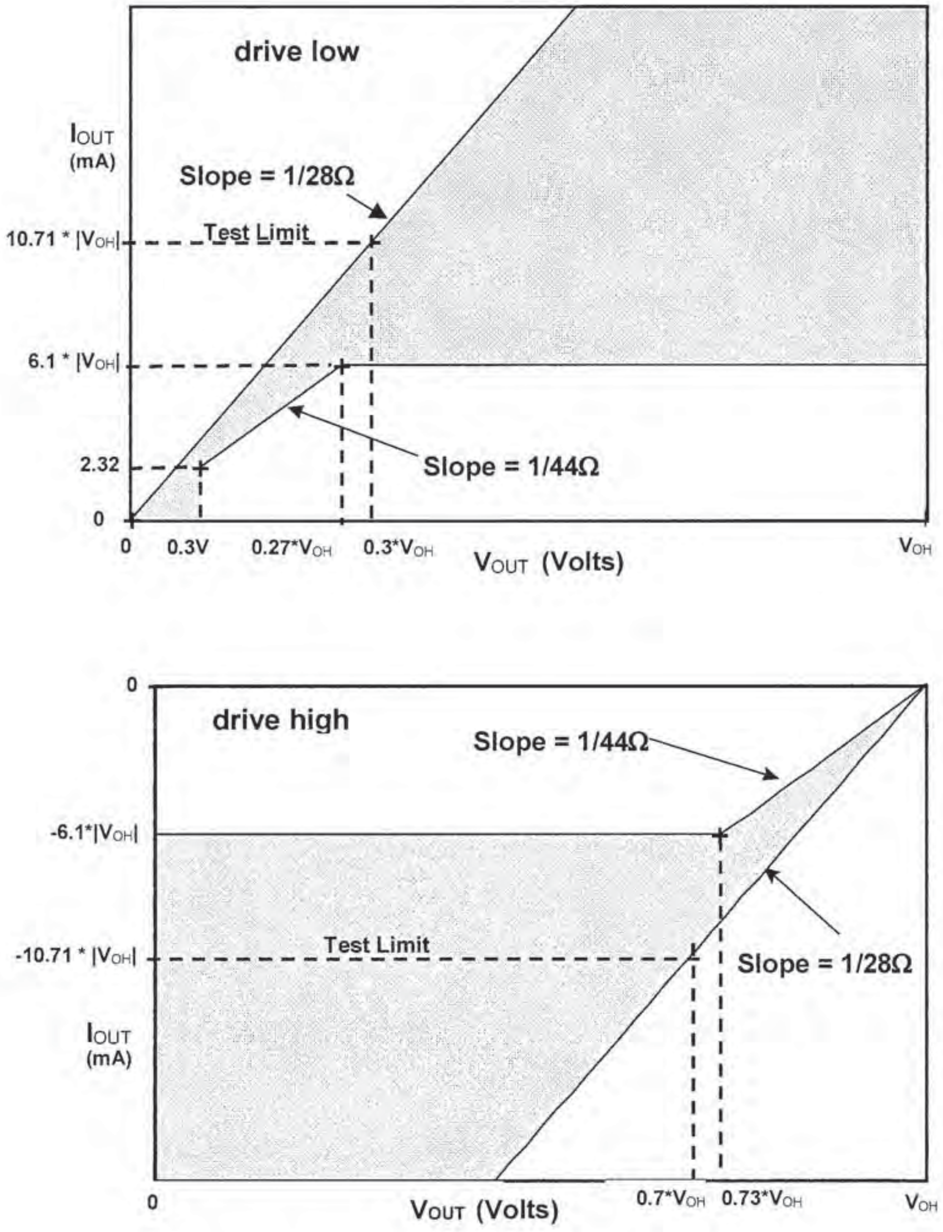


Figure 7-3. Full-speed Buffer V/I Characteristics

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

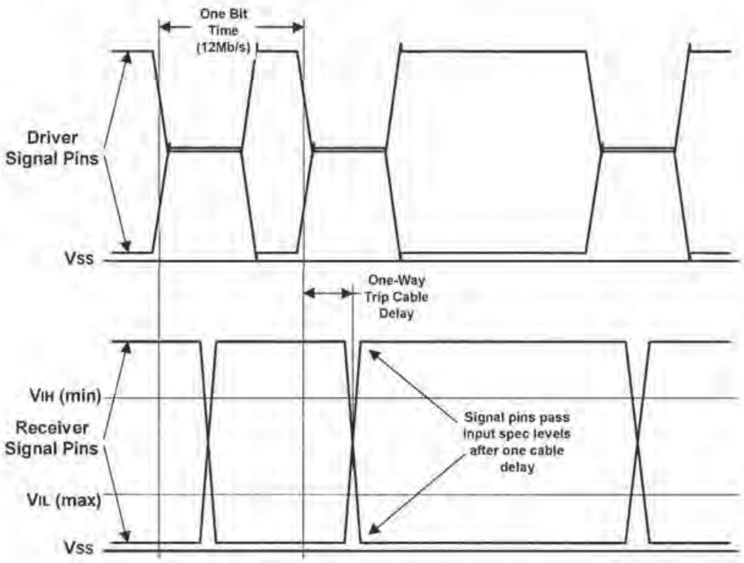


Figure 7-4. Full-speed Signal Waveforms

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.

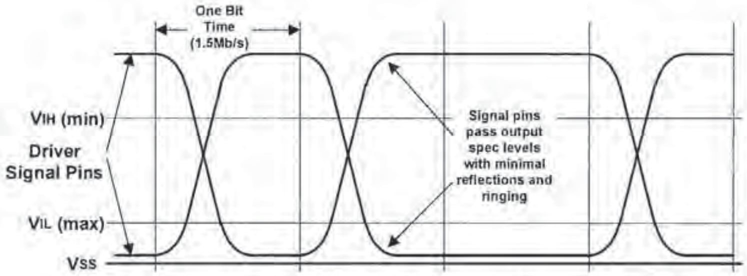


Figure 7-5. Low-speed Driver Signal Waveforms

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.

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 ±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 (CIND). 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 ±20% to minimize RFI emissions and signal skew. The transitions must be monotonic.

For both full-speed and low-speed signaling, the crossover voltage (VCRS) 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.

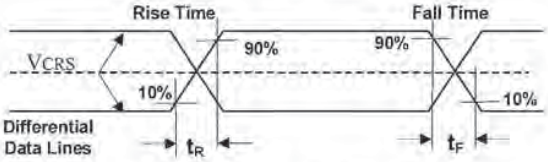


Figure 7-6. Data Signal Rise and Fall Time

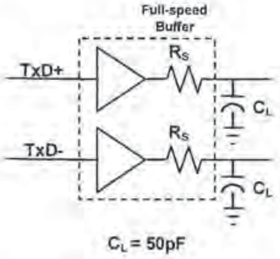
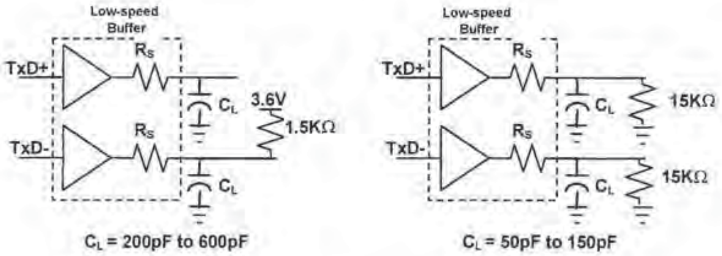


Figure 7-7. Full-speed Load



Low-speed downstream port load

Low-speed upstream port load

Figure 7-8. Low-speed Port Loads

Note: the CL for low-speed port load only represents the range of loading that might be added when the low-speed device is attached to a hub. The low-speed buffer must be designed to drive the load of its attached cable plus CL. A low-speed buffer design that can drive the downstream port test load would be capable of driving any legitimate upstream load.

7.1.2.1 Driver Usage

The upstream ports (towards the host) of all hubs and full-speed functions must use full-speed drivers. The upstream hub port transmits data at both full- and low-speed data rates. However, the signaling always uses full-speed signaling conventions and edge rates (refer to Figure 7-13 Upstream Full Speed Port Transceiver and Table 7-1 Signaling Levels). Transmission of low-speed data does not change the driver's characteristics. The upstream port of low-speed functions must use a low-speed driver.

External downstream ports of all hubs (including the host), are required to be capable of both driver characteristics, such that any type of device can be plugged in to these ports. When the transceiver is in full-speed mode it uses full-speed signaling conventions and edge rates. In low-speed it uses low-speed signaling conventions and edge rates (refer to Figure 7-14 Downstream Port Transceiver and Table 7-1 Signaling Levels).

7.1.3 Cable Skew

The maximum skew introduced by the cable between the differential signaling pair (i.e., D+ and D- (TSKEW)) must be less than 400pS and is measured as described in Section 6.7.

7.1.4 Receiver Characteristics

A differential input receiver must be used to accept the USB data signal. The receiver must feature an input sensitivity (VDI) of at least 200mV when both differential data inputs are in the differential common mode range (VCM) of 0.8V to 2.5V, as shown in Figure 7-9.

In addition to the differential receiver, there must be a single-ended receiver for each of the two data lines. The receivers must have a switching threshold between 0.8V (VIL) and 2.0V (VIH). It is recommended that the single-ended receivers incorporate hysteresis to reduce their sensitivity to noise.

Both D+ and D- may temporarily be less than Vih(min) during differential signal transitions. This period can be up to 14ns (TFST) for full-speed transitions and up to 210ns (TLST) for low-speed transitions. Logic in the receiver must ensure that that this is not interpreted as an SE0.

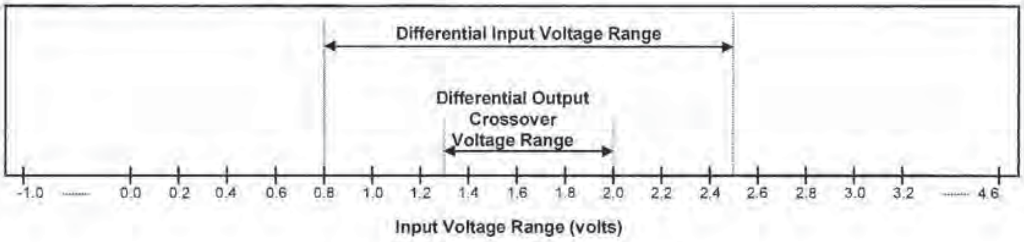


Figure 7-9. Differential Input Sensitivity Range

7.1.5 Device Speed Identification

The USB is terminated at the hub and function ends as shown in Figure 7-10 and Figure 7-11. Full-speed and low-speed devices are differentiated by the position of the pull-up resistor on the downstream end of the cable:

- Full-speed devices are terminated as shown in Figure 7-10 with the pull-up resistor on the D+ line.
- Low-speed devices are terminated as shown in Figure 7-11 with the pull-up resistor on the D- line.
- The pull-down terminators on downstream ports are resistors of 15kΩ ±5% connected to ground.

The design of the pull-up resistor must ensure that the signal levels satisfy the requirements specified in Table 7-1. In order to facilitate bus state evaluation that may be performed at the end of a reset, the design must be able to pull-up D+ or D- from 0V to VIH (min) within the minimum reset relaxation time of 2.5μs. A device that has a detachable cable must use a 1.5kΩ ±5% resistor tied to a voltage source between 3.0V and 3.6V (VTERM) to satisfy these requirements. Devices with captive cables may use alternative termination means. However, the Thevenin resistance of any termination must be no less than 900Ω.

Note: Thevenin resistance of termination does not include the 15kΩ ±5% resistor on host/hub.

The voltage source on the pull-up resistor must be derived from or controlled by the power supplied on the USB cable such that when VBUS is removed, the pull-up resistor does not supply current on the data line to which it is attached.

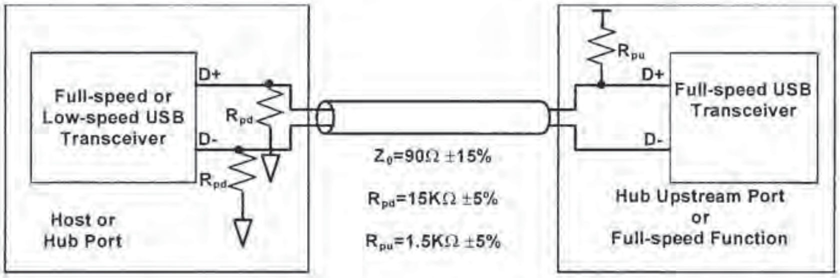


Figure 7-10. Full-speed Device Cable and Resistor Connections

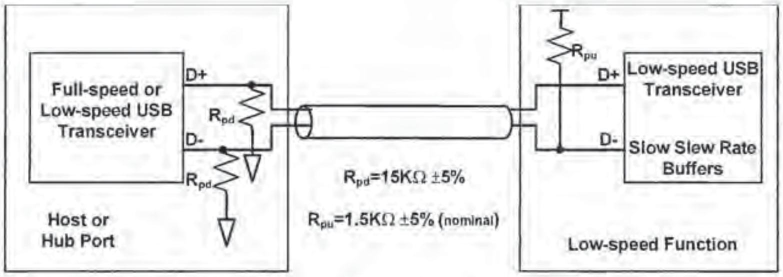


Figure 7-11. Low-speed Device Cable and Resistor Connections

7.1.6 Input Characteristics

The input impedance of D+ or D- without termination should be > 300 kΩ (ZINP). The input capacitance of a port is measured at the connector pins. Upstream and downstream ports are allowed different values of capacitance. The maximum capacitance (differential or single-ended) (CIND) allowed on a downstream port of a hub or host is 150pF on D+ or D-. This is comprised of up to 75pF of lumped capacitance to ground on each line at the transceiver and in the connector, and an additional 75pF capacitance on each conductor in the transmission line between the receptacle and the transceiver. The transmission line between the receptacle and RS must be 90Ω ±15%).

The maximum capacitance on an upstream port of a full-speed device with a detachable cable (CINUB) is 100pF on D+ or D- . This is comprised of up to 75 pF of lumped capacitance to ground on each line at the transceiver and in the connector, and an additional 25pF capacitance on each conductor in the transmission line between the receptacle and the transceiver. The difference in capacitance between D+ and D- must be less than 10%.

For full-speed devices with captive cables, the device itself may have up to 75pF of lumped capacitance to ground on on D+ and D-. The cable accounts for the remainder of the input capacitance .

A low-speed device is required to have a captive cable. The input capacitance of the low-speed device will include the cable. The maximum single-ended or differential input capacitance of a low-speed device is 450pF (CLINUA).

For devices with captive cables, the single-ended input capacitance must be consistent with the termination scheme used. The termination must be able to charge the D+ or D- line from 0V to VIH (min) within 2.5μs. The capacitance on D+/D- includes the single-ended input-capacitance of the device (measured from the pins on the connector on the cable) and the 150pF of input capacitance of the host/hub.

An implementation may use small capacitors at the transceiver for purposes of edge rate control. The sum of the capacitance of the added capacitor (CEDGE) , the transceiver, and the trace connecting capacitor and transceiver to RS must not exceed 75pF (either single-ended or differential) and the capacitance must be balanced to within 10%. The added capacitor, if present, must be placed between the transceiver pins and RS (see Figure 7-12)

Use of ferrite beads on the D+ or D- lines of full-speed devices is discouraged.

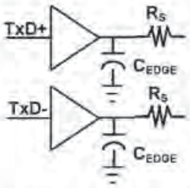


Figure 7-12. Placement of Optional Edge Rate Control Capacitors

7.1.7 Signaling Levels

Table 7-1 summarizes the USB signaling levels. The source is required to drive the levels specified in the second column and the target is required to identify the correct bus state when it sees the levels in the third column. (Target receivers can be more sensitive as long as they are within limits specified in the fourth column).

Table 7-1. Signaling Levels

Bus State	Signaling Levels		
	At originating source connector (at end of bit time)	At final target connector	
		Required	Acceptable
Differential "1"	D+ > V_{OH} (min) and D- < V_{OL} (max)	(D+) - (D-) > 200mV and D+ > V_{IH} (min)	(D+) - (D-) > 200mV
Differential "0"	D- > V_{OH} (min) and D+ < V_{OL} (max)	(D-) - (D+) > 200mV and D- > V_{IH} (min)	(D-) - (D+) > 200mV
Single-ended 0 (SE0)	D+ and D- < V_{OL} (max)	D+ and D- < V_{IL} (max)	D+ and D- < V_{IH} (min)
Data J state:			
Low-speed	Differential "0"	Differential "0"	
Full-speed	Differential "1"	Differential "1"	
Data K state:			
Low-speed	Differential "1"	Differential "1"	
Full-speed	Differential "0"	Differential "0"	
Idle state:	N.A.		
Low-speed		D- > V_{IHZ} (min) and D+ < V_{IL} (max)	D- > V_{IHZ} (min) and D+ < V_{IH} (min)
Full-speed		D+ > V_{IHZ} (min) and D- < V_{IL} (max)	D+ > V_{IHZ} (min) and D- < V_{IH} (min)
Resume state	Data K state	Data K state	
Start-of-Packet (SOP)	Data lines switch from Idle to K state		
End-of-Packet (EOP)[4]	SE0 for approximately 2 bit times[1] followed by a J for 1 bit time[3]	SE0 for ≥ 1 bit time[2] followed by a J state for 1 bit time	SE0 for ≥ 1 bit time[2] followed by a J state
Disconnect (at downstream port)	N.A.	SE0 for ≥2.5μs	
Connect (at downstream port)	N.A.	Idle for ≥2ms	Idle for ≥2.5 μs
Reset	D+ and D- < V_{OL} (max) for ≥10ms	D+ and D- < V_{IL} (max) for ≥10ms	D+ and D- < V_{IL} (max) for ≥2.5 μs

Note 1: The width of EOP is defined in bit times relative to the speed of transmission. (Specification EOP widths are given in Table 7-5 and Table 7-6.)

Note 2: The width of EOP is defined in bit times relative to the device type receiving the EOP. The bit time is approximate.

Note 3: The width of the J state following the EOP is defined in bit times relative to the buffer edge rate. The J state from a low-speed buffer must be a low-speed bit time wide and from a full-speed buffer, a full-speed bit time wide.

Note 4: The keep-alive is a low-speed EOP.

The J and K data states are the two logical levels used to communicate differential data in the system. Differential signaling is measured from the point where the data line signals cross over. Differential data signaling is not concerned with the level at which the signals cross, as long as the crossover voltage meets the requirements in Section 7.1.2. Note that, at the receiver, the Idle and Resume states are logically equivalent to the J and K states respectively.

As shown in Table 7-1, the J and K states for full-speed signaling are inverted from those for low-speed signaling. The sense of data, idle, and resume signaling is set by the type of device that is being attached to a port. If a full-speed device is attached to a port, that segment of the USB uses full-speed signaling conventions (and fast rise and fall times), even if the data being sent across the data lines is at the low-speed data rate. The low-speed signaling conventions shown in Table 7-1 (plus slow rise and fall times) are used only between a low-speed device and the port to which it is attached.

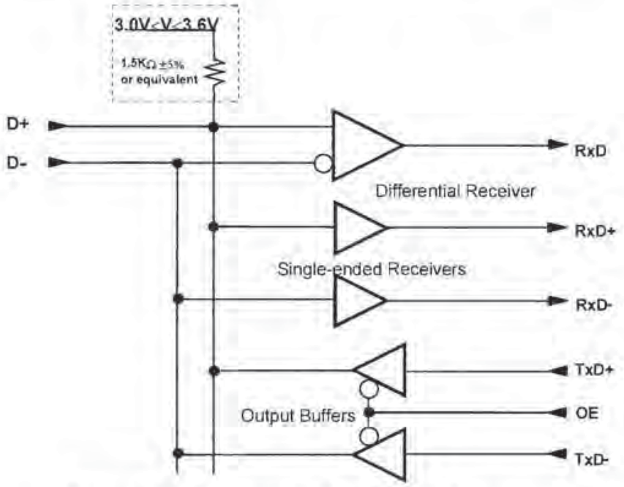


Figure 7-13. Upstream Full-speed Port Transceiver

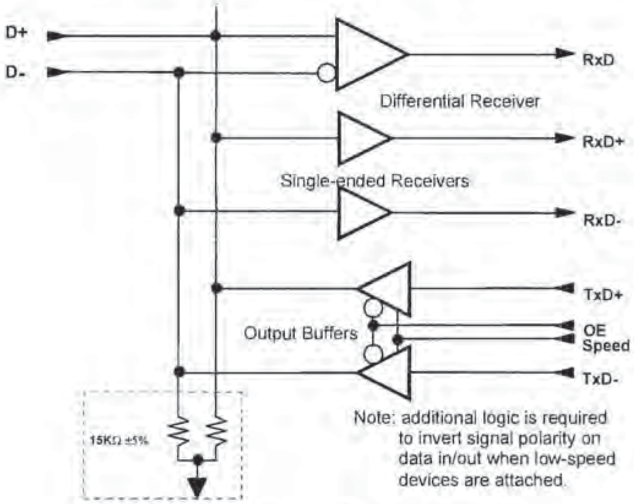


Figure 7-14. Downstream Port Transceiver

7.1.7.1 Connect and Disconnect Signaling

When no function is attached to the downstream port of the host or hub, the pull-down resistors present there will cause both D+ and D- to be pulled below the single-ended low threshold of the host or hub port when that port is not being driven by the hub. This creates an SE0 state on the downstream port. A disconnect condition (TDDIS) is indicated if the host or hub is not driving the data lines and an SE0 persists on a downstream port for more than 2.5μs (see Figure 7-15).

A connect condition (TDCNN) will be detected when the hub detects that one of the data lines is pulled above its VIH threshold for more than 2.5μs (see Figure 7-16 and Figure 7-17).

Hubs may optionally determine the speed of the attached device by sampling the state of the bus immediately before driving SE0 to indicate a reset condition to the device. Alternatively, the hub may float the bus after driving reset and perform bus state evaluation after 2.5μs as shown in Figure 7-18.

All signaling levels given in Table 7-1 are set for this bus segment (and this segment alone) once the speed of the attached device is determined. The mechanics of speed detection are described in Section 11.8.2.

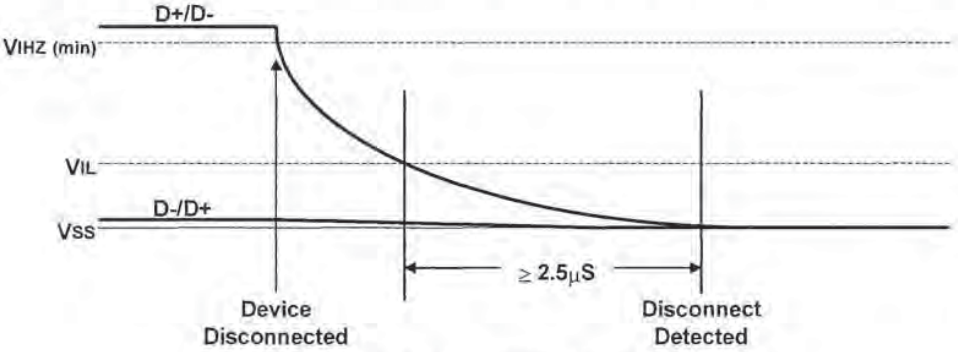


Figure 7-15. Disconnect Detection

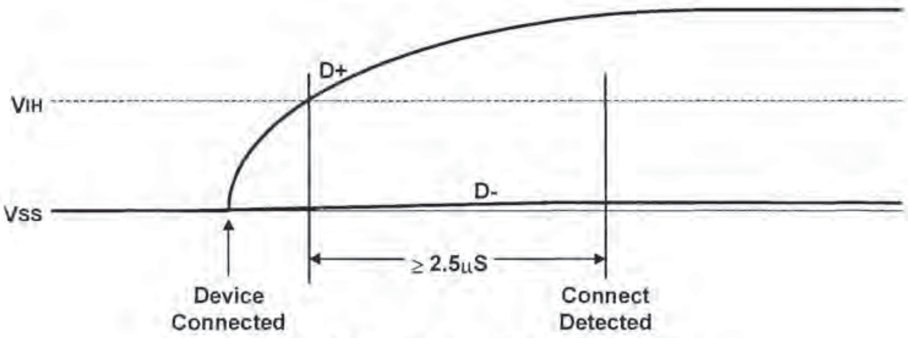


Figure 7-16. Full-speed Device Connect Detection

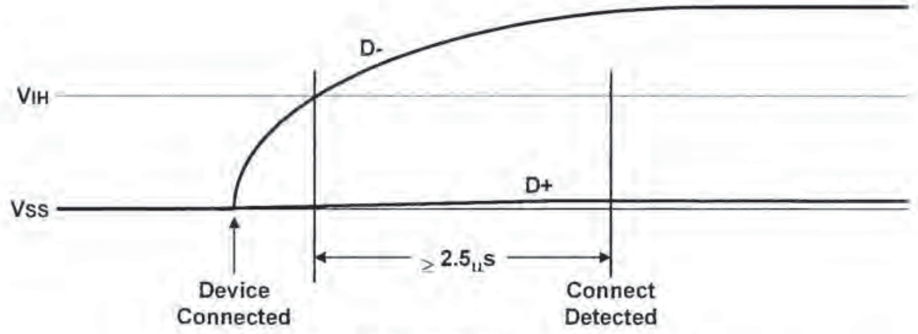


Figure 7-17. Low-speed Device Connect Detection

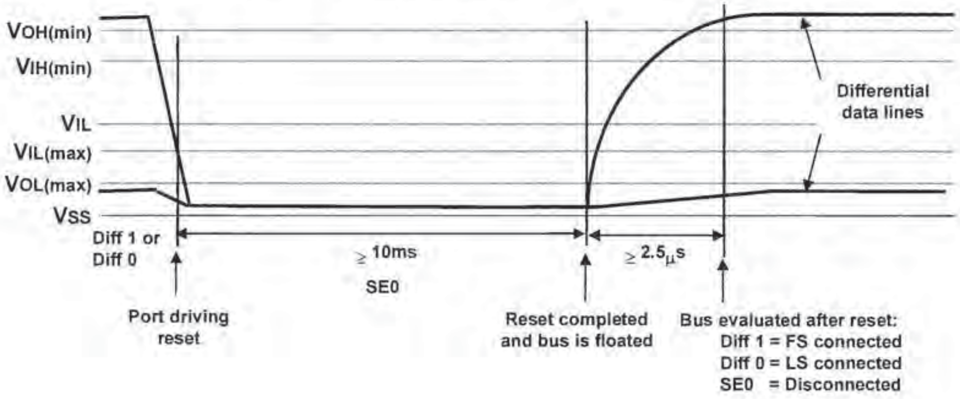


Figure 7-18. Bus State Evaluation after reset (optional)

Because USB components may be hot plugged, and hubs may implement power switching, it is necessary to comprehend the delays between power switching and/or device attach and when the device's internal power has stabilized. Figure 7-19 shows all the events associated with both turning on port power with a device connected and hot-plugging a device. There are six delays and a sequence of events that are defined by this specification.

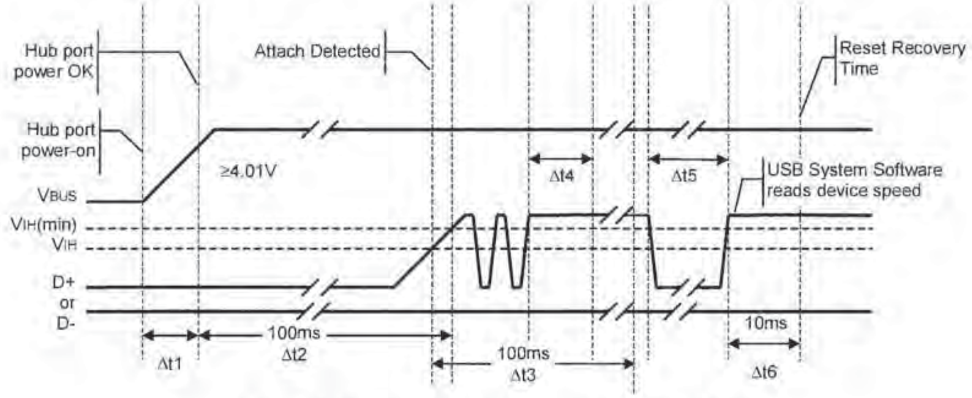


Figure 7-19. Power-on and Connection Events Timing

Δt1 This is the amount of time required for the hub port power switch to operate. This delay is a function of the type of hub port switch. Hubs report this time in the hub descriptor (see Section 11.15.2.1), which can be read via a request to the Hub Controller (see Section 11.16.2.4). If a device were plugged into a non-switched or already-switched on port, Δt1 is equal to zero.

Δt2 (TSIGATT)This is the maximum time from when VBUS is up to valid level (4.01V) to when a device has to signal attach. Δt2 represents the time required for the device's internal power rail to stabilize and for D+ or D- to reach VIH (min) at the hub. Δt2 must be less than 100ms for all hub and device implementations. (This requirement only applies if the device is drawing power from the bus).

Δt3 (TATTDB) This is a debounce interval with a minimum duration of 100ms that is provided by the USB System Software. It ensures that the electrical and mechanical connection is stable before software attempts to reset the attached device. The interval starts when the USB System Software is notified of a connection detection. The interval restarts if there is a disconnect. The debounce interval ensures that power is stable at the device for at least 100ms before any requests will be sent to the device.

Δt4 (T2SUSP) Anytime a device observes no bus activity, it must obey the rules of going into suspend (see Section 7.1.7.4).

Δt5 (TDRST) This is the period of time hubs drive reset to a device. Refer to Section 11.5.1.5 for details.

Δt6 (TRSTRCY) The USB System Software guarantees a minimum of 10ms for reset recovery. Device response to any bus transactions addressed to the default device address during the reset recovery time is undefined.

7.1.7.2 Data Signaling

Data transmission within a packet is done with differential signals.

The start of a packet (SOP) is signaled by the originating port by driving the D+ and D- lines from the Idle state to the opposite logic level (K state). This switch in levels represents the first bit of the SYNC field. Hubs must limit the change in the width of the first bit of SOP when it is retransmitted to less than ±5ns. Distortion can be minimized by matching the nominal data delay through the hub with the output enable delay of the hub.

The SE0 state is used to signal an end-of-packet (EOP). EOP will be signaled by driving D+ and D- to the SE0 state for two bit times followed by driving the lines to the J state for one bit time. The transition from the SE0 to the J state defines the end of the packet at the receiver. The J state is asserted for one bit time and then both the D+ and D- output drivers are placed in their high-impedance state. The bus termination resistors hold the bus in the Idle state. Figure 7-20 shows the signaling for start and end of a packet.

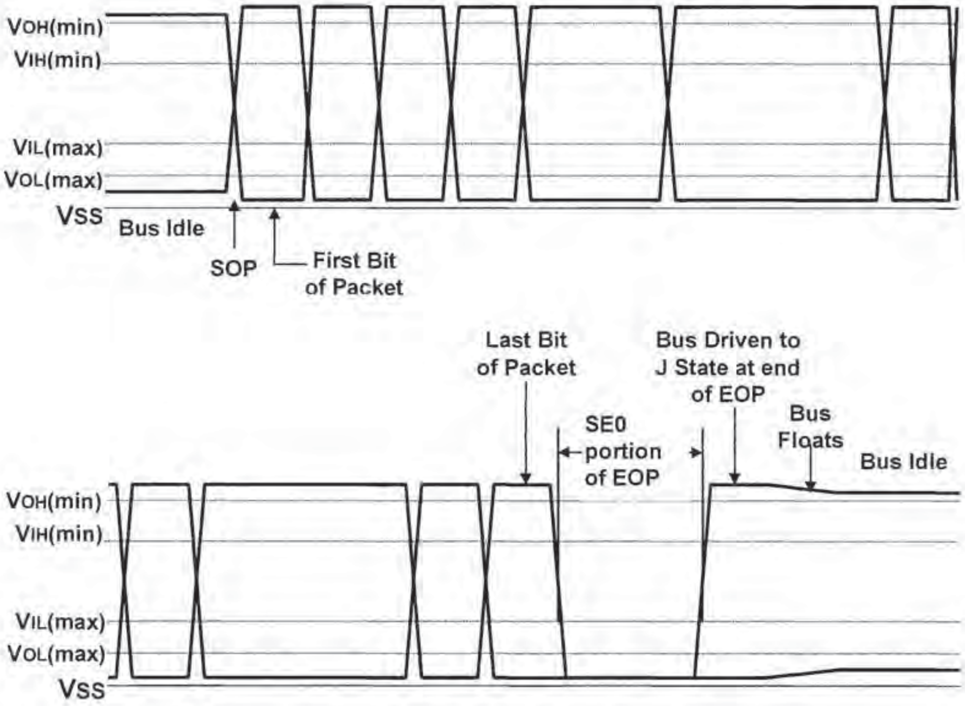


Figure 7-20. Packet Voltage Levels

7.1.7.3 Reset Signaling

A hub signals reset to a downstream port by driving an extended SE0 at the port. After the reset is removed, the device will be in the Default state (refer to Section 9.1).

The reset signaling can be generated on any Hub or Host Controller port by request from the USB System Software. The reset signaling must be driven for a minimum of 10ms (TDRST). After the reset, the hub port will transition to the Enabled state (refer to Section 11.5). Host Controllers and the USB System Software must ensure that resets issued to the root ports drive reset long enough to overwhelm any concurrent resume attempts by downstream devices. Resets from root ports should be nominally 50ms (TDRSTR). It is not required that this be 50ms of continuous Reset signaling. However, if the reset is not continuous, the interval(s) between signaling reset must be less than 3ms (TRHRSI).

A device seeing an SE0 on its upstream port for more than 2.5µs (TDETRST) may treat that signal as a reset. The reset must have taken effect before the reset signaling ends.

Hubs will propagate traffic to a newly reset port after the port is in the Enabled state. The device attached to this port must recognize this bus activity and keep from going into the Suspend state.

Hubs must be able to accept all hub requests and devices must be able to accept a SetAddress() request (refer to Section 11.16.2 and Section 9.4 respectively) after the reset recovery time 10ms (TRSTRCY) after the reset is removed. Failure to accept this request may cause the device not to be recognized by the USB system software. Hubs and devices must complete commands within the times specified in Chapter 9 and Chapter 11.

Reset must wake a device from the Suspend state.

7.1.7.4 Suspending

All devices must support the Suspend state. Devices can go into the Suspend state from any powered state. They begin the transition to the Suspend state after they see a constant Idle state on their upstream bus lines for more than 3.0ms. The device must actually be suspended, drawing only suspend current from the bus after no more than 10ms of bus inactivity on all its ports. Any bus activity on the upstream port will keep a device out of the Suspend state. In the absence of any other bus traffic, the SOF token (refer to Section 8.4.2) will occur once per frame to keep full-speed devices from suspending. In the absence of any low-speed traffic, low-speed devices will see at least one keep-alive (defined in Table 7-1) in every frame in which an SOF occurs, which keeps them from suspending. Hubs generate this keep-alive as described in Section 11.8.4.1.

While in the Suspend state, a device must continue to provide power to its D+ (full-speed) or D- (low-speed) pull-up resistor to maintain an idle so that the upstream hub can maintain the correct connectivity status for the device.

7.1.7.4.1 Global Suspend

Global suspend is used when no communication is desired anywhere on the bus and the entire bus is placed in the Suspend state. The host signals the start of global suspend by ceasing all its transmissions (including the SOF token). As each device on the bus recognizes that the bus is in the Idle state for the appropriate length of time, it goes into the Suspend state.

7.1.7.4.2 Selective Suspend

Segments of the bus can be selectively suspended by sending the command SetPortFeature(PORT_SUSPEND) to the hub port to which that segment is attached. The suspended port will block activity to the suspended bus segment and devices on that segment will go into the Suspend state after the appropriate delay as described above.

Section 11.5 describes the port Suspend state and its interaction with the port state machine. Suspend is further described in Section 11.9.

7.1.7.5 Resume

If a device is in the Suspend state, its operation is resumed when any non-idle signaling is received on its upstream port. Additionally, the device can signal the system to resume operation if its remote wakeup capability has been enabled by the USB System Software. Resume signaling is used by the host or a device to bring a suspended bus segment back to the active condition. Hubs play an important role in the propagation and generation of resume signaling. The following description is an outline of a general global resume sequence. A complete description of the resume sequence, the special cases caused by selective suspend, and the role of the hub are given in Section 11.9.

The host may signal resume (TDRSMDN) at any time. It must send the resume signaling for at least 20ms and then end the resume signaling with a standard, low-speed EOP (two low-speed bit times of SE0 followed by a J). The 20ms of resume signaling ensures that all devices in the network that are enabled to see the resume are awakened. The EOP tears down the connectivity established by the resume signaling and prepares the hubs for normal operation. After resuming the bus, the host must begin sending bus traffic (at least the SOF token) within 3ms to keep the system from going back into the Suspend state.

A device with remote wakeup capability may not generate resume signaling unless the bus has been continuously in the Idle state for 5ms (TWTRSM). This allows the hubs to get into their Suspend state and prepare for propagating resume signaling. The remote wakeup device must hold the resume signaling for at least 1ms but for no more than 15ms (TDRSMUP). At the end of this period, the device stops driving the bus (puts its drivers into the high-impedance state and does not drive the bus to the J state).

If the hub upstream of a remote wakeup device is suspended, it will propagate the resume signaling to its upstream port and to all of its enabled downstream ports, including the port that originally signaled the resume. The hub must begin this rebroadcast (TURSM) of the resume signaling within 100µs of receiving the original resume. The resume signal will propagate in this manner upstream until it reaches the host or a non-suspended hub (refer to Section 11.9), which will reflect the resume downstream and take control of resume timing. This hub is termed the controlling hub. Intermediate hubs (hubs between the resume initiator and the controlling hub) drive resume (TDRSMUP) on their upstream port for at least 1ms during which time they also continue to drive resume on enabled downstream ports. An intermediate hub will stop driving resume on the upstream port and reverse the direction of connectivity from upstream to downstream within 15ms after first asserting resume on its upstream port. When all intermediate hubs have reversed connectivity, resume is being driven from the controlling hub through all intermediate hubs and to all enabled ports. The controlling hub must rebroadcast the resume signaling within 100µs (TURSM) and ensures that resume is signaled for at least 20ms (TDRSMDN). The hub may then begin normal operation by terminating the resume process as described above.

The USB System Software must provide a 10ms resume recovery time (TRSMRCY) during which it will not attempt to access any device connected to the affected (just-activated) bus segment.

Port connects and disconnects can also cause a hub to send a resume signal and awaken the system. These events will cause a hub to send a resume signal only if the hub has been enabled as a remote-wakeup source. Refer to Section 11.4.4 for more details.

Refer to Section 7.2.3 for a description of power control during suspend and resume.

7.1.8 Data Encoding/Decoding

The USB employs NRZI data encoding when transmitting packets. In NRZI encoding, a "1" is represented by no change in level and a "0" is represented by a change in level. Figure 7-21 shows a data stream and the NRZI equivalent. The high level represents the J state on the data lines in this and subsequent figures showing NRZI encoding. A string of zeros causes the NRZI data to toggle each bit time. A string of ones causes long periods with no transitions in the data.

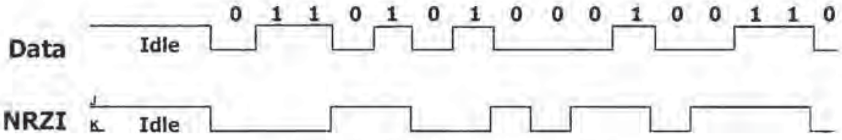


Figure 7-21. NRZI Data Encoding

7.1.9 Bit Stuffing

In order to ensure adequate signal transitions, bit stuffing is employed by the transmitting device when sending a packet on USB (see Figure 7-22 and Figure 7-24). A zero is inserted after every six consecutive ones in the data stream before the data is NRZI encoded, to force a transition in the NRZI data stream. This gives the receiver logic a data transition at least once every seven bit times to guarantee the data and clock lock. Bit stuffing is enabled beginning with the Sync Pattern and throughout the entire transmission. The data "one" that ends the Sync Pattern is counted as the first one in a sequence. Bit stuffing by the transmitter is always enforced, without exception. If required by the bit stuffing rules, a zero bit will be inserted even if it is the last bit before the end-of-packet (EOP) signal.

The receiver must decode the NRZI data, recognize the stuffed bits, and discard them. If the receiver sees seven consecutive ones anywhere in the packet, then a bit stuffing error has occurred and the packet should be ignored. The time interval just before an EOP is a special case. The last data bit before the EOP can become stretched by hub switching skews. This is known as dribble and can lead to the case illustrated in Figure 7-23, which shows where dribble introduces a sixth bit that does not require a bit stuff. Therefore, the receiver must accept a packet for which there are up to six full bit times at the port with no transitions prior to the EOP.

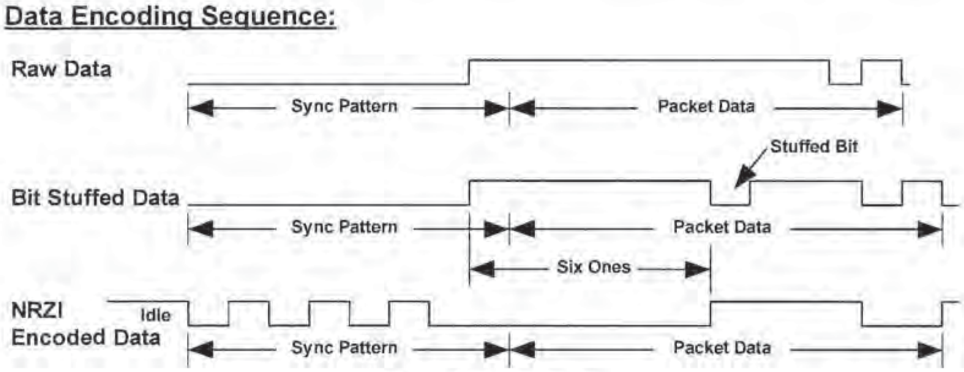


Figure 7-22. Bit Stuffing

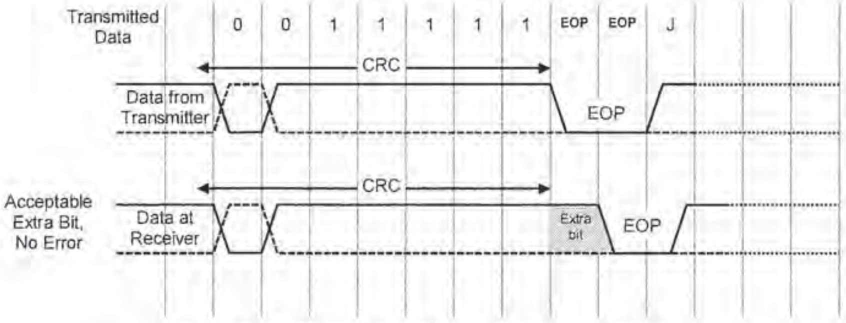


Figure 7-23. Illustration of Extra Bit Preceding EOP

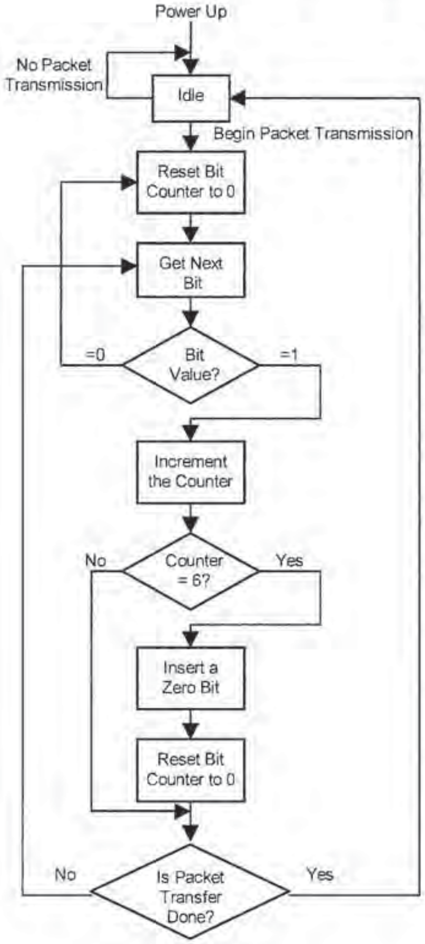


Figure 7-24. Flow Diagram for Bit Stuffing

7.1.10 Sync Pattern

The NRZI bit pattern shown in Figure 7-25 is used as a synchronization pattern and is prefixed to each packet. This pattern is equivalent to a data pattern of seven zeroes followed by a one (80H).

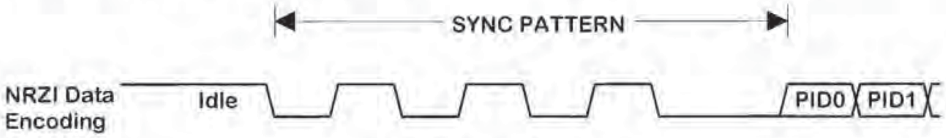


Figure 7-25. Sync Pattern

7.1.11 Data Signaling Rate

The full-speed data rate is nominally 12.000Mb/s. The data-rate tolerance for host, hub, and full-speed functions is ±0.25% (2,500ppm). The accuracy of the Host Controller's data rate must be known and controlled to better than ±0.05% (500ppm). This tolerance includes inaccuracies from all sources:

- Initial frequency accuracy
- Crystal capacitive loading
- Supply voltage on the oscillator
- Temperature
- Aging.

The low-speed data rate is nominally 1.50Mb/s. The permitted data-rate tolerance for low-speed functions is ±1.5% (15,000ppm). This tolerance includes inaccuracies from all sources:

- Initial frequency accuracy
- Crystal capacitive loading
- Supply voltage on the oscillator
- Temperature
- Aging.

This tolerance allows the use of resonators in low cost, low-speed devices.

7.1.12 Frame Interval and Frame Interval Adjustment

The USB defines a frame interval to be 1.000ms ±500ns long. The frame interval is measured from any point in an SOF token in one frame to the same point in the SOF token of the next frame.

The Host Controller must be able to adjust the frame interval. There are two possible components to the frame interval adjustment. If the host's data rate clock is not exactly 12.000Mb/s, then the initial ±0.05% frame interval accuracy can be met by changing the default number of bits per frame from the nominal of 12,000. A Host Controller component that has a range of possible clock-source values may have to make this initial frame count a programmable value. An additional adjustment of ±15 full-speed bit times is required to allow the host to synchronize to an external time reference. During normal bus operation, software may not change the frame interval by more than one full-speed bit time every six frames. If no adjustment is being made, the frame interval repeatability (difference in frame interval between two successive frames) must be less than 0.5 full-speed bit times (TRFI). If an adjustment is being made the frame interval repeatability must be less than 1.5 full-speed bit times (TRFIADJ).

Hubs and certain full-speed functions need to track the frame interval. They also are required to have sufficient frame timing adjustment to compensate for their own frequency tolerance and track the host's ±15 full-speed bit time variability.

7.1.13 Data Source Signaling

This section covers the timing characteristics of data produced and sent from a port (the data source). Section 7.1.14 covers the timing characteristics of data that is transmitted through the Hub Repeater section of a hub. In this section, T_{PERIOD} is defined as the actual period of the data rate that can have a range as defined in Section 7.1.11.

7.1.13.1 Data Source Jitter

The source of data can have some variation (jitter) in the timing of edges of the data transmitted. The time between any set of data transitions is N * T_{PERIOD} ± jitter time, where 'N' is the number of bits between the transitions. The data jitter is measured with the same load used for maximum rise and fall times and is measured at the crossover points of the data lines, as shown in Figure 7-26.

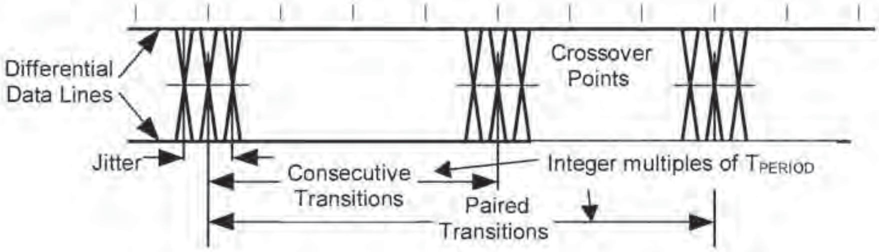


Figure 7-26. Data Jitter Taxonomy

- For full-speed transmissions, the jitter time for any consecutive differential data transitions must be within ±2.0ns and within ±1.0ns for any set of paired (JK-to-next JK transition or KJ-to-next KJ transition) differential data transitions.
- For low-speed transmissions, the jitter time for any consecutive differential data transitions must be within ±25ns and within ±10ns for any set of paired differential data transitions.

These jitter numbers include timing variations due to differential buffer delay and rise and fall time mismatches, internal clock source jitter, and noise and other random effects.

7.1.13.2 EOP Width

The width of the SE0 in the EOP is approximately 2 * T_{PERIOD}. The SE0 width is measured with the same load used for maximum rise and fall times and is measured at the same level as the differential signal crossover points of the data lines (see Figure 7-27).

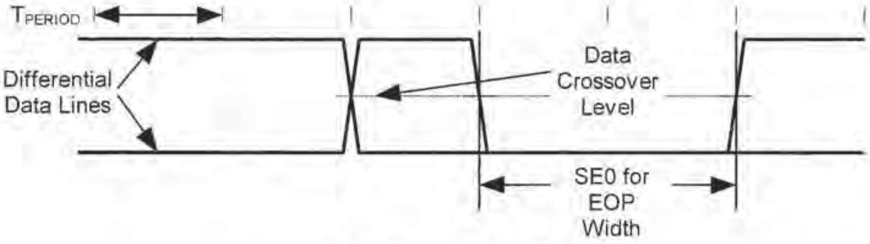


Figure 7-27. SE0 for EOP Width Timing

- For full-speed transmissions, the SE0 for EOP width from the transmitter must be between 160ns and 175ns.
- For low-speed transmissions, the transmitter's SE0 for EOP width must be between 1.25μs and 1.50μs.

These ranges include timing variations due to differential buffer delay and rise and fall time mismatches and to noise and other random effects.

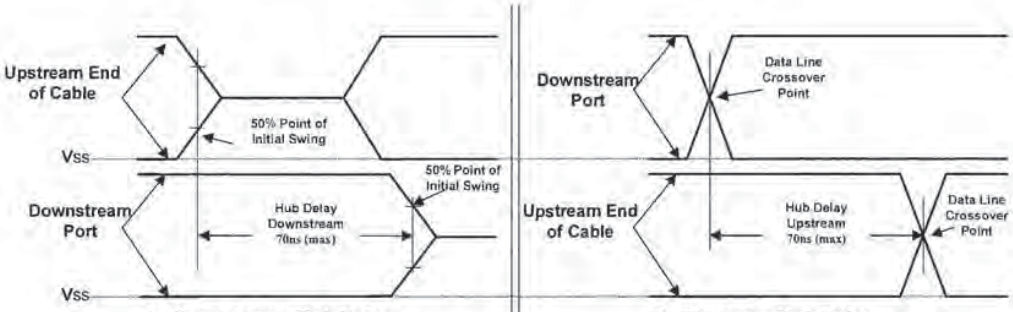
A receiver must accept any valid EOP. Receiver design should note that the single-ended input threshold voltage can be different from the differential crossover voltage and the SE0 transitions will in general be asynchronous to the clock encoded in the NRZI stream.

- A full-speed EOP may have the SE0 interval reduced to as little as 82ns (T_{FEOPR}) and a low-speed SE0 interval may be as short as 670ns (T_{LEOPR}).

A hub may tear down connectivity if it sees an SE0 of at least T_{FST} or T_{LST} followed by a transition to the J state. A hub must tear down connectivity on any valid EOP.

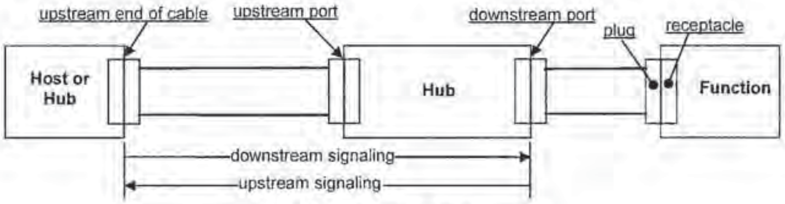
7.1.14 Hub Signaling Timings

The propagation of a full-speed, differential data signal through a hub is shown in Figure 7-28. The downstream signaling is measured without a cable connected to the port and with the load used for measuring rise and fall times. The total delay through the upstream cable and hub electronics must be a maximum of 70ns (T_{HDD1}). If the hub has a USB detachable cable, then the delay (T_{HDD2}) through hub electronics and the associated transmission line must be a maximum of 44ns to allow for a worst-case cable delay of 26ns (T_{FSCBL}). The delay through this hub is measured in both the upstream and downstream directions, as shown in Figure 7-28B, from data line crossover at the input port to data line crossover at the output port.



A. Downstream Hub Delay

B. Upstream Hub Delay



C. Measurement Points

Figure 7-28. Hub Propagation Delay of Full-speed Differential Signals

Low-speed propagation delay for differential signals is measured in the same fashion as for full-speed signaling. The maximum low-speed hub delay is 300ns (TLHDD). This allows for the slower low-speed buffer propagation delay and rise and fall times. It also provides time for the hub to re-clock the low-speed data in the upstream direction.

When the hub acts as a repeater, it must reproduce the received, full-speed signal accurately on its outputs. This means that for differential signals, the propagation delays of a J-to-K state transition must match closely to the delays of a K-to-J state transition. For full-speed propagation, the maximum difference allowed between these two delays (THDJ1) (see Figure 7-28 and Figure 7-42) for a hub plus cable is ±3.0ns. Similarly, the difference in delay between any two J-to-K or K-to-J transitions through a hub (THDJ2) must be less than ±1.0ns. For low-speed propagation in the downstream direction, the corresponding allowable jitter (TLDHJ1) is ±45ns and (TLDHJ2) ±15ns, respectively. For low-speed propagation in the upstream direction, the allowable jitter is ±45ns in both cases (TLUHJ1 and TLUHJ2).

An exception to this case is the skew that can be introduced in the Idle-to-K state transition at SOP (TFSOP and TLSOP) (refer to Section 7.1.7.2). In this case, the delay to the opposite port includes the time to enable the output buffer. However, the delays should be closely matched to the normal hub delay and the maximum additional delay difference over a normal J-to-K transition is ±5.0ns. This limits the maximum distortion of the first bit in the packet.

Note: because of this distortion of the SOP transition relative to the next K-to-J state transition, the first SYNC field bit should not be used to synchronize the receiver to the data stream.

The EOP must be propagated through a hub in the same way as the differential signaling. The propagation delay for sensing an SE0 must be no less than the greater of the J-to-K, or K-to-J differential data delay (to avoid truncating the last data bit in a packet), but not more than 15ns greater than the larger of these differential delays at full-speed and 200ns at low-speed (to prevent creating a bit stuff error at the end of the packet). EOP delays are shown in Figure 7-43.

Because the sense levels for the SE0 state are not at the midpoint of the signal swing, the width of SE0 state will be changed as it passes through each hub. A hub may not change the width of the SE0 state in a full-speed EOP by more than ±15ns (TFHESK), as measured by the difference of the leading edge and trailing edge delays of the SE0 state (see Figure 7-43). An SE0 from a low-speed device has long rise and fall times and is subject to greater skew, but this conditions exists only on the cable from the low-speed device to the port to which it is connected. Thereafter, the signaling uses full-speed buffers and their faster rise and fall times. The SE0 from the low-speed device cannot be changed by more than ±300ns (TLHESK) as it passes through the hub to which the device is connected. This time allows for some signal conditioning in the low-speed port to reduce its sensitivity to noise.

7.1.15 Receiver Data Jitter

The data receivers for all types of devices must be able to properly decode the differential data in the presence of jitter. The more of the bit cell that any data edge can occupy and still be decoded, the more reliable the data transfer will be. Data receivers are required to decode differential data transitions that occur in a window plus and minus a nominal quarter bit cell from the nominal (centered) data edge position. (A simple 4X over-sampling state machine DPLL can be built that satisfies these requirements.) This requirement is derived in Table 7-2 and Table 7-3. The tables assume a worst-case topology of five hubs between the host and device and the worst-case number of seven bits between transitions. The derived numbers are rounded up for ease of specification.

Jitter will be caused by the delay mismatches discussed above and by mismatches in the source and destination data rates (frequencies). The receive data jitter budgets for full- and low-speed are given in Table 7-2 and Table 7-3. These tables give the value and totals for each source of jitter for both consecutive (next) and paired transitions. Note that the jitter component related to the source or destination frequency tolerance has been allocated to the appropriate device (i.e., the source jitter includes bit shifts due to source frequency inaccuracy over the worst-case data transition interval). The output driver jitter can be traded off against the device clock accuracy in a particular implementation as long as the jitter specification is met.

The low-speed jitter budget table has an additional line in it because the jitter introduced by the hub to which the low-speed device is attached is different from all the other devices in the data path. The remaining devices operate with full-speed signaling conventions (though at low-speed data rate).

Table 7-2. Full-speed Jitter Budget

Jitter Source	Full-speed			
	Next Transition		Paired Transition	
	Each (ns)	Total (ns)	Each (ns)	Total (ns)
Source Driver Jitter	2.0	2.0	1.0	1.0
Source Frequency Tolerance (worst-case)	0.21/bit	1.5	0.21/bit	3.0
Source Jitter Total		**3.5**		**4.0**
Hub Jitter	3.0	15.0	1.0	5.0
Jitter Specification		**18.5**		**9.0**
Destination Frequency Tolerance	0.21/bit	1.5	0.21/bit	3.0
Receiver Jitter Budget		**20.0**		**12.0**

Table 7-3. Low-speed Jitter Budget

Jitter Source	Low-speed Upstream			
	Next Transition		Paired Transition	
	Each (ns)	Total (ns)	Each (ns)	Total (ns)
Function Driver Jitter	25.0	25.0	10.0	10.0
Function Frequency Tolerance (worst-case)	10.0/bit	70.0	10.0/bit	140.0
Source (Function) Jitter Total		**95.0**		**150.0**
Hub with Low-speed Device Jitter	45.0	45.0	45.0	45.0
Remaining (full-speed) Hubs' Jitter	3.0	12.0	1.0	4.0
Jitter Specification		**152.0**		**199.0**
Host Frequency Tolerance	1.7/bit	12.0	1.7/bit	24.0
Host Receiver Jitter Budget		**164.0**		**223.0**
	Low-speed Downstream			
	Next Transition		**Paired Transition**	
	Each (ns)	**Total (ns)**	**Each (ns)**	**Total (ns)**
Host Driver Jitter	2.0	2.0	1.0	1.0
Host Frequency Tolerance (worst-case)	1.7/bit	12.0	1.7/bit	24.0
Source (Host) Jitter Total		**14.0**		**25.0**
Hub with Low-speed Device Jitter	45.0	45.0	15.0	15.0
Remaining (full-speed) Hubs' Jitter	3.0	12.0	1.0	4.0
Jitter Spec		**71.0**		**44.0**
Function Frequency Tolerance	10.0/bit	70.0	10.0/bit	140.0
Function Receiver Jitter Budget		**141.0**		**184.0**

Note: this table describes the host transmitting at low-speed data rate using full-speed signaling to a low-speed device through the maximum number of hubs. When the host is directly connected to the low-speed device, then it uses low-speed data rate and low-speed signaling, and the host has to meet the source jitter listed in the "Jitter Specification" row.

7.1.16 Cable Delay

Except (in certain cases) for the SOP, only one data transition is allowed on a USB cable at a time. A full-speed signal edge has to propagate to the far end of the cable, return, and settle within one full-speed bit time. Therefore, the maximum total one-way signal propagation delay allowed is 30ns. The allocation for cable delay is 26ns. A maximum delay of 3ns is allowed from a Host or Hub Controller downstream port to its exterior downstream connector, while a maximum delay of 1ns is allowed from the upstream connector to the upstream port of any device. For a standard USB detachable cable, the cable delay is measured from the Series A connector pins to the Series B connector pins and is no more than 26ns. For other cables, the delay is measured from the series A connector to the point where the cable is connected to the device.

The maximum one-way data delay on a full-speed cable is measured as shown in Figure 7-29.

One-way cable delay for low-speed cables must be less than 18ns. It is measured as shown in Figure 7-30.

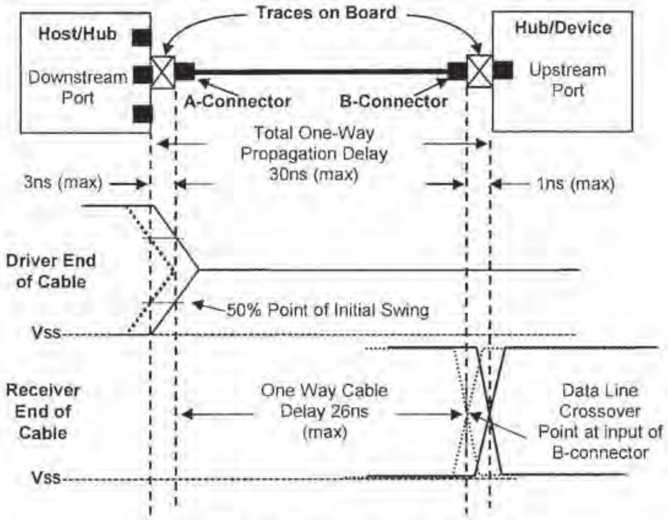
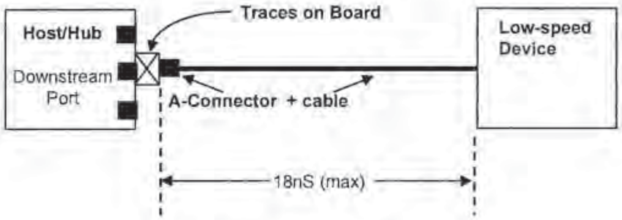


Figure 7-29. Full-speed Cable Delay



One-way Propagation Delay

Figure 7-30. Low-speed Cable Delay

7.1.17 Cable Attenuation

The allowable attenuation of the signal pair (D+, D-) for full speed signaling per cable is listed in Table 7-4. The cable attenuation measurement is defined in Section 6.7.

Table 7-4. Signal Attenuation

Frequency (MHz)	Attenuation (maximum) dB/cable
0.064	0.08
0.256	0.11
0.512	0.13
0.772	0.15
1.000	0.20
4.000	0.39
8.000	0.57
12.000	0.67
24.000	0.95
48.000	1.35
96.000	1.9

7.1.18 Bus Turn-around Time and Inter-packet Delay

Inter-packet delays are measured from the SE0-to-J transition at the end of the EOP to the J-to-K transition that starts the next packet.

A device is required to allow two bit times of inter-packet delay. The delay is measured at the responding device with a bit time defined in terms of the response. This provides adequate time for the device sending the EOP to drive J for one bit time and then turn off its output buffers.

The host must provide at least two bit times of J after the SE0 of an EOP and the start of a new packet (TIPD). If a function is expected to provide a response to a host transmission, the maximum inter-packet delay for a function or hub with a detachable (TRSPIPD1) cable is 6.5 bit times measured at the Series B receptacle. If the device has a captive cable, the inter-packet delay (TRSPIPD2) must be less than 7.5 bit times as measured at the Series-A plug. These timings apply to both full-speed and low-speed devices and the bit times are referenced to the data rate of the packet.

The maximum inter-packet delay for a host response is 7.5 bit times, measured at the host's port pins. There is no maximum inter-packet delay between packets in unrelated transactions.

7.1.19 Maximum End-to-end Signal Delay

A device expecting a response to a transmission will invalidate the transaction if it does not see the start-of-packet (SOP) transition within the timeout period after the end of the transmission (after the SE0-to-J state transition in the EOP). This can occur between an IN token and the following data packet or between a data packet and the handshake packet (refer to Chapter 8). The device expecting the response will not time out before 16 bit times but will timeout before 18 bit times (measured at the data pins of the device from the SE0-to-J transition at the end of the EOP). The host will wait at least 18 bit times for a response to start before it will start a new transaction.

Figure 7-31 depicts the configuration of six signal hops (cables) that results in allowable worst-case signal delay. The maximum propagation delay from the upstream end of a hub's cable to any downstream port connector is 70ns.

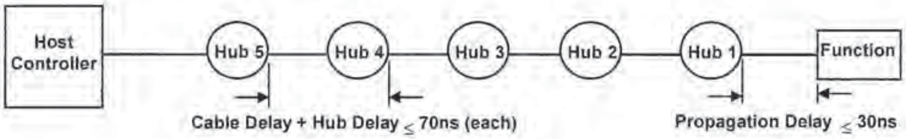


Figure 7-31. Worst-case End to End Signal Delay Model

7.2 Power Distribution

This section describes the USB power distribution specification.

7.2.1 Classes of Devices

The power source and sink requirements of different device classes can be simplified with the introduction of the concept of a unit load. A unit load is defined to be 100mA. The number of unit loads a device can draw is an absolute maximum, not an average over time. A device may be either low-power at one unit load or high-power, consuming up to five unit loads. All devices default to low-power. The transition to high-power is under software control. It is the responsibility of software to ensure adequate power is available before allowing devices to consume high-power.

The USB supports a range of power sourcing and power consuming agents; these include the following:

- **Root port hubs**: Are directly attached to the USB Host Controller. Hub power is derived from the same source as the Host Controller. Systems that obtain operating power externally, either AC or DC must supply at least five unit loads to each port. Such ports are called high-power ports. Battery-powered systems may supply either one or five unit loads. Ports that can supply only one unit load are termed low-power ports.
- **Bus-powered hubs**: Draw all of their power for any internal functions and downstream ports from VBUS on the hub's upstream port. Bus-powered hubs may only draw up to one unit load upon power-up, and five unit loads after configuration. The configuration power is split between allocations to the hub, any non-removable functions and the external ports. External ports in a bus-powered hub can supply only one unit load per port regardless of the current draw on the other ports of that hub. The hub must be able to supply this port current when the hub is in the Active or Suspend state.
- **Self-powered hubs**: Power for the internal functions and downstream ports does not come from VBUS. However, the USB interface of the hub may draw up to one unit load from its upstream VBUS to allow the interface to function when the remainder of the hub is powered down. Hubs that obtain operating power externally (from the USB) must supply five unit loads to each port. Battery-powered hubs may supply either one or five unit loads per port.
- **Low-power bus-powered functions**: All power to these devices comes from VBUS. They may draw no more than one unit load at any time.
- **High-power bus-powered functions**: All power to these devices comes from VBUS. They must draw no more than one unit load upon power-up and may draw up to five unit loads after being configured.

- **Self-powered functions**: May draw up to one unit load from VBUS to allow the USB interface to function when the remainder of the function is powered down. All other power comes from an external (to the USB) source.

No device shall supply (source) current on VBUS at its upstream port at any time. From VBUS on its upstream port, a device may only draw (sink) current. They may not provide power to the pull-up resistor on D+/D- unless VBUS is present (see Section 7.1.5). On power-up, a device needs to ensure that its upstream port is not driving the bus, so that the device is able to receive the reset signaling. Devices must also ensure that the maximum operating current drawn by a device is one unit load, until configured. Any device that draws power from the bus must be able to detect lack of activity on the bus, enter the Suspend state and reduce its current consumption from VBUS (refer to Section 7.2.3 and Section 9.2.5.1).

7.2.1.1 Bus-powered Hubs

Bus-powered hub power requirements can be met with a power control circuit such as the one shown in Figure 7-32. Bus-powered hubs often contain at least one non-removable function. Power is always available to the hub's controller, which permits host access to power management and other configuration registers during the enumeration process. A non-removable function(s) may require that its power be switched, so that upon power-up the entire device (hub and non-removable functions) draws no more than one unit load. Power switching on any non-removable function may be implemented either by removing its power or by shutting off the clock. Switching on the non-removable function is not required if the aggregate power drawn by it and the Hub Controller is less than one unit load. However, as long as the hub port associated with the function is in the Power-off state, the function must be logically reset and the device must appear to be not connected. The total current drawn by a bus-powered device is the sum of the current to the Hub Controller, any non-removable function(s), and the downstream ports.

Figure 7-32 shows the partitioning of power based upon the maximum upstream current draw of five unit loads: one unit load for the Hub Controller and the non-removable function, and one unit load for each of the external downstream ports. If more than four external ports are required, then the hub will need to be self-powered. If the non-removable function(s) and Hub Controller draw more than one unit load, then the number of external ports must be appropriately reduced. Power control to a bus-powered hub may require a regulator. If present, the regulator is always enabled to supply the Hub Controller. The regulator can also power the non-removable functions(s). Inrush current limiting must also be incorporated into the regulator subsystem.

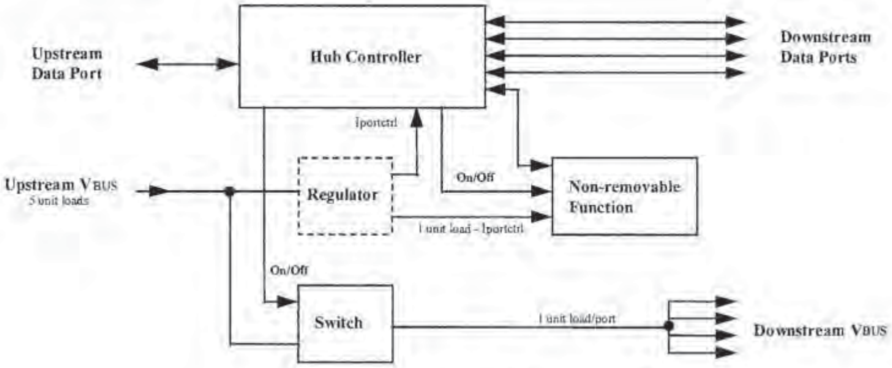


Figure 7-32. Compound Bus-powered Hub

Power to external downstream ports of a bus-powered hub must be switched. The Hub Controller supplies a software controlled on/off signal from the host, which is in the "off" state when the device is powered up or after reset signaling. When switched to the "on" state, the switch implements a soft turn-on function that prevents excessive transient current from being drawn from the upstream port. The voltage drop across the upstream cable, connectors, and switch in a bus-powered hub must not exceed 350mV at maximum rated current.

7.2.1.2 Self-powered Hubs

Self-powered hubs have a local power supply that furnishes power to any non-removable functions and to all downstream ports, as shown in Figure 7-33. Power for the Hub Controller, however, may be supplied from the upstream VBUS (a "hybrid" powered hub) or the local power supply. The advantage of supplying the Hub Controller from the upstream supply is that communication from the host is possible even if the device's power supply remains off. This makes it possible to differentiate between a disconnected and an unpowered device. If the hub draw power for its upstream port from VBUS, it may not draw more than one unit load.

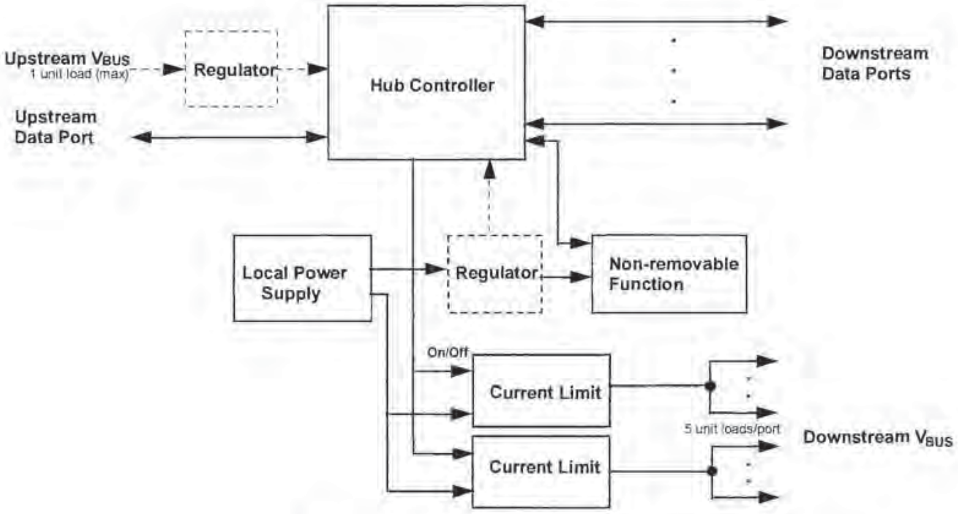


Figure 7-33. Compound Self-powered Hub

The number of ports that can be supported is limited only by the address capability of the hub and the local supply.

Self-powered hubs may experience loss of power. This may be the result of disconnecting the power cord or exhausting the battery. Under these conditions, the hub may force a re-enumeration of itself as a bus-powered hub. This requires the hub to implement port power switching on all external ports. When power is lost, the hub must ensure that upstream current does not exceed low-power. All the rules of a bus-powered hub then apply.

7.2.1.2.1 Over-current Protection

The host and all self-powered hubs must implement over-current protection for safety reasons, and the hub must have a way to detect the over-current condition and report it to the USB software. Should the aggregate current drawn by a gang of downstream ports exceed a preset value, the over-current protection circuit removes or reduces power from all affected downstream ports. The over-current condition is reported through the hub to Host Controller, as described in Section 11.13.5. The preset value cannot exceed 5.0 A and must be sufficiently above the maximum allowable port current such that transient currents (e.g. during power up or dynamic attach or reconfiguration) do not trip the over-current protector.

If an over-current condition occurs on any port, subsequent operation of the USB is not guaranteed, and once the condition is removed, it may be necessary to reinitialize the bus as would be done upon power-up. The over-current limiting mechanism must be resettable without user mechanical intervention. Polymeric PTCs and solid-state switches are examples of methods, which can be used for over-current limiting.

7.2.1.3 Low-power Bus-powered Functions

A low-power function is one that draws up to one unit load from the USB cable when operational. Figure 7-34 shows a typical bus-powered, low-power function, such as a mouse. Low-power regulation can be integrated into the function silicon. Low-power functions must be capable of operating with input VBUS voltages as low as 4.40V, measured at the plug end of the cable.

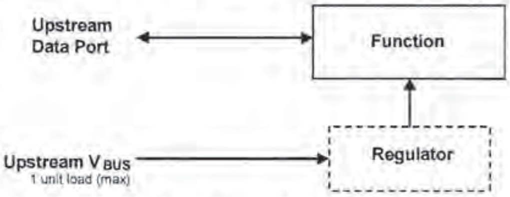


Figure 7-34. Low-power Bus-powered Function

7.2.1.4 High-power Bus-powered Functions

A function is defined as being high-power if, when fully powered, it draws over one but less than five unit loads from the USB cable. A high-power function requires staged switching of power. It must first come up in a reduced power state of less than one unit load. At bus enumeration time, its total power requirements are obtained and compared against the available power budget. If sufficient power exists, the remainder of the function may be powered on. A typical high-power function is shown in Figure 7-35. The function's electronics have been partitioned into two sections. The function controller contains the minimum amount of circuitry necessary to permit enumeration and power budgeting. The remainder of the function resides in the function block. High-power functions must be capable of operating in their low-power (one unit load) mode with an input voltage as low as 4.40V, so that it may be detected and enumerated even when plugged into a bus-powered hub. They must also be capable of operating at full power (up to five unit loads) with a VBUS voltage of 4.75V, measured at the upstream plug end of the cable.

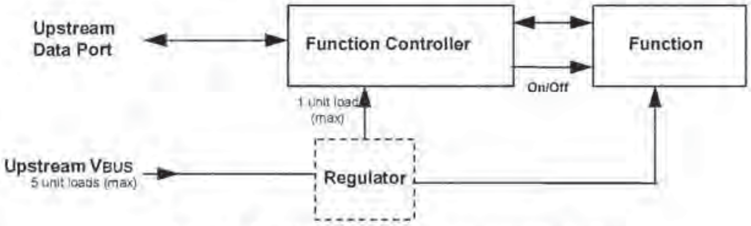


Figure 7-35. High-power Bus-powered Function

7.2.1.5 Self-powered Functions

Figure 7-36 shows a typical self-powered function. The function controller is powered either from the upstream bus via a low-power regulator or from the local power supply. The advantage of the former scheme is that it permits detection and enumeration of a self-powered function whose local power supply is turned off. The maximum upstream power that the function controller can draw is one unit load, and the regulator block must implement inrush current limiting. The amount of power that the function block may draw is limited only by the local power supply. Because the local power supply is not required to power any downstream bus ports, it does not need to implement current limiting, soft start, or power switching.

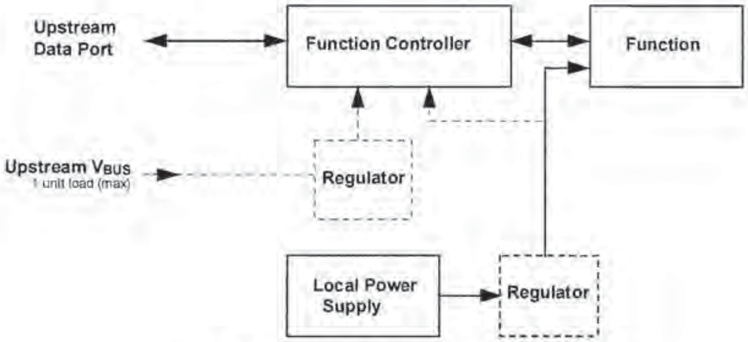


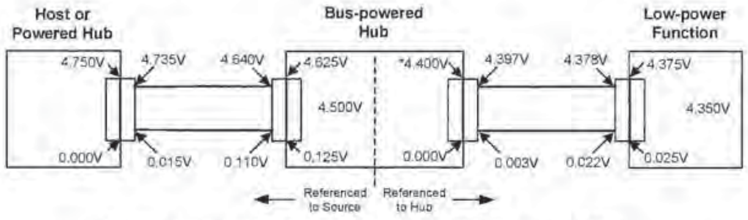
Figure 7-36. Self-powered Function

7.2.2 Voltage Drop Budget

The voltage drop budget is determined from the following:

- The voltage supplied by high-powered hub ports is 4.75V to 5.25V.
- The voltage supplied by low-powered hub ports is 4.4V to 5.25V.
- Bus-powered hubs can have a maximum drop of 350mV from their cable plug (where they attach to a source of power) to their output port connectors (where they supply power).
- The maximum voltage drop (for detachable cables) between the A-series plug and B-series plug on VBUS is 125mV (VBUSD).
- The maximum voltage drop for all cables between upstream and downstream on GND is 125mV (VGNDD).
- All hubs and functions must be able to provide configuration information with as little as 4.40V at the connector end of their upstream cables. Only low-power functions need to be operational with this minimum voltage.
- Functions drawing more than one unit load must operate with a 4.75V minimum input voltage at the connector end of their upstream cables.

Figure 7-37 shows the minimum allowable voltages in a worst-case topology consisting of a bus-powered hub driving a bus-powered function.



*Under transient conditions, supply at hub can drop from 4.400V to 4.070V

Figure 7-37. Worst-case Voltage Drop Topology (Steady State)

7.2.3 Power Control During Suspend/Resume

Suspend current is a function of unit load allocation. All USB devices initially default to low-power. Low-power devices or high-power devices operating at low-power are limited to 500μA of suspend current. If the device is configured for high-power and enabled as a remote wakeup source, it may draw up to 2.5mA during suspend. When computing suspend current, the current from VBUS through the bus pull-up and pull-down resistors must be included. Configured bus-powered hubs may also consume a maximum of 2.5mA, with 500μA allocated to each available external port and the remainder available to the hub and its internal functions. If a hub is not configured, it is operating as a low-power device and must limit its suspend current to 500μA.

While in the Suspend state, a device may briefly draw more than the average current. The amplitude of the current spike cannot exceed the device power allocation 100mA (or 500mA). A maximum of 1.0 second is allowed for an averaging interval. The average current cannot exceed the average suspend current limit (ICCSH or ICCSL, see Table 7-5) during any 1.0s interval (TSUSAVG1). The profile of the current spike is restricted so the transient response of the power supply (which may be an efficient, low-capacity, trickle power supply) is not overwhelmed. The rising edge of the current spike must be no more than 100mA/μs. Downstream ports must be able to absorb the 500mA peak current spike and meet the voltage droop requirements defined for inrush current during dynamic attach (see Section 7.2.4.1). Figure 7-38 illustrates a typical example profile for an averaging interval. If the supply to the pull-up resistor on D+/D- is derived from VBUS, then the suspend current will never go to zero because the pull-up and pull-down resistors will always draw power.

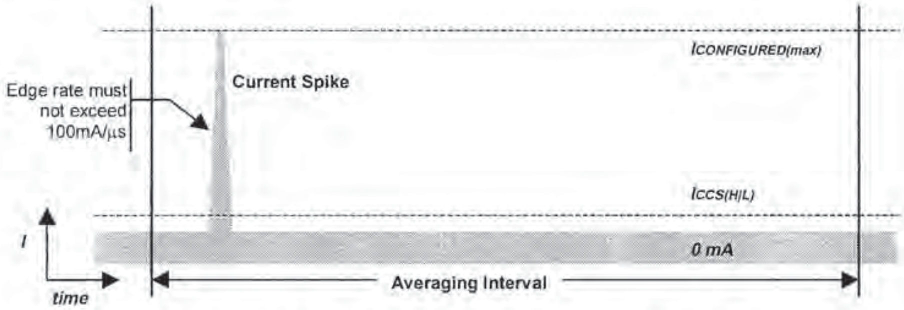


Figure 7-38. Typical Suspend Current Averaging Profile

Devices are responsible for handling the bus voltage reduction due to the inductive and resistive effects of the cable. When a hub is in the Suspend state, it must still be able to provide the maximum current per port (one unit load of current per port for bus-powered hubs and five unit loads per port for self-powered hubs). This is necessary to support remote wakeup-capable devices that will power-up while the remainder of the system is still suspended. Such devices, when enabled to do remote wakeup, must drive resume signaling upstream within 10ms of starting to draw the higher, non-suspend current. Devices not capable of remote wakeup must draw the higher current only when not suspended.

When devices wakeup, either by themselves (remote wakeup) or by seeing resume signaling, they must limit the inrush current on VBUS. The target maximum droop in the hub VBUS is 330mV. The device must have sufficient on-board bypass capacitance or a controlled power-on sequence such that the current drawn from the hub does not exceed the maximum current capability of the port at any time while the device is waking up.

7.2.4 Dynamic Attach and Detach

The act of plugging or unplugging a hub or function must not affect the functionality of another device on other segments of the network. Unplugging a function will stop the transaction between that function and the host. However, the hub to which this function was attached will recover from this condition and will alert the host that the port has been disconnected.

7.2.4.1 Inrush Current Limiting

When a function or hub is plugged into the network, it has a certain amount of on-board capacitance between VBUS and ground. In addition, the regulator on the device may supply current to its output bypass capacitance and to the function as soon as power is applied. Consequently, if no measures are taken to prevent it, there could be a surge of current into the device which might pull the VBUS on the hub below its minimum operating level. Inrush currents can also occur when a high-power function is switched into its high-power mode. This problem must be solved by limiting the inrush current and by providing sufficient capacitance in each hub to prevent the power supplied to the other ports from going out of tolerance. An additional motivation for limiting inrush current is to minimize contact arcing, thereby prolonging connector contact life.

The maximum droop in the hub VBUS is 330mV, or about 10% of the nominal signal swing from the function. In order to meet this requirement, the following conditions must be met:

- The maximum load (CRPB) that can be placed at the downstream end of a cable is 10μF in parallel with 44Ω. The 10μF capacitance represents any bypass capacitor directly connected across the VBUS lines in the function plus any capacitive effects visible through the regulator in the device. The 44Ω resistance represents one unit load of current drawn by the device during connect.
- If more bypass capacitance is required in the device, then the device must incorporate some form of VBUS surge current limiting, such that it matches the characteristics of the above load.
- The hub downstream port VBUS power lines must be bypassed (CHPB) with no less than 120μF of low-ESR capacitance per hub. Standard bypass methods should be used to minimize inductance and resistance between the bypass capacitors and the connectors to reduce droop. The bypass capacitors themselves should have a low dissipation factor to allow decoupling at higher frequencies.

The upstream port of a hub is also required to meet the above requirements. Furthermore, a bus-powered hub must provide additional surge limiting in the form of a soft-start circuit when it enables power to its downstream ports.

A high-power bus-powered device that is switching from a lower power configuration to a higher power configuration must not cause droop > 330 mV on the VBUS at its upstream hub . The device can meet this by ensuring that changes in the capacitive load it presents do not exceed 10μF.

Signal pins are protected from excessive currents during dynamic attach by being recessed in the connector such that the power pins make contact first. This guarantees that the power rails to the downstream device are referenced before the signal pins make contact. In addition, the signal lines are in a high-impedance state during connect, so that no current flows for standard signal levels.

7.2.4.2 Dynamic Detach

When a device is detached from the network with power flowing in the cable, the inductance of the cable will cause a large flyback voltage to occur on the open end of the device cable. This flyback voltage is not destructive. Proper bypass measures on the hub ports will suppress any coupled noise. The frequency range of this noise is inversely dependent on the length of the cable, to a maximum of 60MHz for a one-meter cable. This will require some low capacitance, very low inductance bypass capacitors on each hub port connector. The flyback voltage and the noise it creates is also moderated by the bypass capacitance on the device end of the cable. Also, there must be some minimum capacitance on the device end of the cable to ensure that the inductive flyback on the open end of the cable does not cause the voltage on the device end to reverse polarity. A minimum of 1.0µF is recommended for bypass across VBUS.

7.3 Physical Layer

The physical layer specifications are described in the following subsections.

7.3.1 Regulatory Requirements

All USB devices should be designed to meet the applicable regulatory requirements.

7.3.2 Bus Timing/Electrical Characteristics

Table 7-5. DC Electrical Characteristics

Parameter	Symbol	Conditions	Min.	Max.	Units
Supply Voltage:					
High-power Port	V_{BUS}	Note 2, Section 7.2.1	4.75	5.25	V
Low-power Port	V_{BUS}	Note 2, Section 7.2.1	4.40	5.25	V
Supply Current:					
High-power Hub Port (out)	I_{CCPRT}	Section 7.2.1	500		mA
Low-power Hub Port (out)	I_{CCUPT}	Section 7.2.1	100		mA
High-power Function (in)	I_{CCHPF}	Section 7.2.1		500	mA
Low-power Function (in)	I_{CCLPF}	Section 7.2.1		100	mA
Unconfigured Function/Hub (in)	I_{CCINIT}	Section 7.2.1.4		100	mA
Suspended High-power Device	I_{CCSH}	Section 7.2.3 ; Note 15		2.5	mA
Suspended Low-power Device	I_{CCSL}	Section 7.2.3		500	μA
Input Levels:					
High (driven)	V_{IH}	Note 4, Section 7.1.4	2.0		V
High (floating)	V_{IHZ}	Note 4, Section 7.1.4	2.7	3.6	V
Low	V_{IL}	Note 4, Section 7.1.4		0.8	V
Differential Input Sensitivity	V_{DI}	|(D+)-(D-)|; Figure 7-9; Note 4	0.2		V
Differential Common Mode Range	V_{CM}	Includes V_{DI} range; Figure 7-9; Note 4	0.8	2.5	V
Output Levels:					
Low	V_{OL}	Note 4, 5, Section 7.1.1	0.0	0.3	V
High (Driven)	V_{OH}	Note 4, 6, Section 7.1.1	2.8	3.6	V
Output Signal Crossover Voltage	V_{CRS}	Measured as in Figure 7-6; Note 10	1.3	2.0	V
Decoupling Capacitance:					
Downstream Port Bypass Capacitance (per hub)	C_{HPB}	V_{BUS} to GND, Section 7.2.4.1	120		μF
Upstream Port Bypass Capacitance	C_{RPB}	V_{BUS} to GND; Note 9, Section 7.2.4.1	1.0	10.0	μF

Table 7-5. DC Electrical Characteristics (Continued)

Parameter	Symbol	Conditions	Min.	Max.	Units
Input Capacitance:					
Downstream Port	C_{IND}	Note 2; Section 7.1.6		150	pF
Upstream Port (w/o cable)	C_{INUB}	Note 3; Section 7.1.6		100	pF
Transceiver edge rate control capacitance	C_{EDGE}	Section 7.1.6		75	pF
Terminations:					
Bus Pull-up Resistor on Upstream Port	R_{PU}	1.5kΩ ±5% Section 7.1.5	1.425	1.575	kΩ
Bus Pull-down Resistor on Downstream Port	R_{PD}	15kΩ ±5% Section 7.1.5	14.25	15.75	kΩ
Input impedance exclusive of pullup/pulldown	Z_{INP}	Section 7.1.6	300		kΩ
Termination voltage for upstream port pullup (R_{PU})	V_{TERM}	Section 7.1.5	3.0	3.6	V

Table 7-6. Full-speed Source Electrical Characteristics

Parameter	Symbol	Conditions	Min.	Max.	Units
Driver Characteristics:					
Rise Time	TFR	Figure 7-6; Figure 7-7	4	20	ns
Fall Time	TFF	Figure 7-6; Figure 7-7	4	20	ns
Differential Rise and Fall Time Matching	TFRFM	(TFR/TFF) Note 10, Section 7.1.2	90	111.11	%
Driver Output Resistance	ZDRV	Section 7.1.1.1	28	44	Ω
Clock Timings:					
Full-speed Data Rate	TFDRATE	Average bit rate, Section 7.1.11	11.9700	12.0300	Mb/s
Frame Interval	TFRAME	Section 7.1.12	0.9995	1.0005	ms
Consecutive Frame Interval Jitter	TRFI	No clock adjustment		42	ns
Consecutive Frame Interval Jitter	TRFIADJ	With clock adjustment		126	ns
Full-speed Data Timings:					
Source Jitter Total (including frequency tolerance):		Note 7, 8, 12, 10; Measured as in Figure 7-39;			
To Next Transition	TDJ1		-3.5	3.5	ns
For Paired Transitions	TDJ2		-4	4	ns
Source Jitter for Differential Transition to SE0 Transition	TFDEOP	Note 8; Figure 7-40; Note 11	-2	5	ns
Receiver Jitter:		Note 8; Figure 7-41			
To Next Transition	TJR1		-18.5	18.5	ns
For Paired Transitions	TJR2		-9	9	ns
Source SE0 interval of EOP	TFEOPT	Figure 7-40	160	175	ns
Receiver SE0 interval of EOP	TFEOPR	Note 13; Section 7.1.13.2; Figure 7-40	82		ns
Width of SE0 interval during differential transition	TFST	Section 7.1.4		14	ns

Table 7-7. Low-speed Source Electrical Characteristics

Parameter	Symbol	Conditions	Min.	Max.	Units
Driver Characteristics:					
Transition Time: Rise Time Fall Time	TLR TLF	Measured as in Figure 7-6	75 75	300 300	ns ns
Rise and Fall Time Matching	TLRFM	(TLR/TLF) Note 10	80	125	%
Upstream Port (w/cable, low-speed only)	CLINUA	Note 1; Section 7.1.6	200	450	pF
Clock Timings:					
Low-speed Data Rate	TLDRATE	Section 7.1.11	1.4775	1.5225	Mb/s
Low-speed Data Timings:					
Upstream port source Jitter Total (including frequency tolerance): To Next Transition For Paired Transitions	TUDJ1 TUDJ2	Note 7, 8; Figure 7-39	-95 -150	95 150	ns ns
Upstream port source Jitter for Differential Transition to SE0 Transition	TLDEOP	Note 8; Figure 7-40; Note 11	-40	100	ns
Upstream port differential Receiver Jitter: To Next Transition For Paired Transitions	TDJR1 TDJR2	Note 8; Figure 7-41	-75 -45	75 45	ns ns
Downstream port source Jitter Total (including frequency tolerance): To Next Transition For Paired Transitions	TDDJ1 TDDJ2	Note 7, 8; Figure 7-39	-25 -14	25 14	ns ns
Downstream port source Jitter for Differential Transition to SE0 Transition		Note 8; Figure 7-40; Note 11			ns
Downstream port Differential Receiver Jitter: To Next Transition For Paired Transitions	TUJR1 TUJR2	Note 8; Figure 7-40	-152 -200	152 200	ns ns
Source SE0 interval of EOP	TLEOPT	Figure 7-40	1.25	1.50	µs
Receiver SE0 interval of EOP	TLEOPR	Note 13; Section 7.1.13.2; Figure 7-40	670		ns
Width of SE0 interval during differential transition	TLST	Section 7.1.4		210	ns

Table 7-8. Hub/Repeater Electrical Characteristics

Parameter	Symbol	Conditions	Min.	Max.	Units
Full-speed Hub Characteristics (as measured at connectors):					
Driver Characteristics: (Refer to Table 7-6)		Upstream port and downstream ports configured as full-speed			
Hub Differential Data Delay:		Note 7, 8			
(with cable)	THDD1	Figure 7-42A		70	ns
(without cable)	THDD2	Figure 7-42B		44	ns
Hub Differential Driver Jitter: (including cable)		Note 7, 8; Figure 7-42, Section 7.1.14			
To Next Transition	THDJ1		-3	3	ns
For Paired Transitions	THDJ2		-1	1	ns
Data Bit Width Distortion after SOP	TFSOP	Note 8; Figure 7-42	-5	5	ns
Hub EOP Delay Relative to THDD	TFEOPD	Note 8; Figure 7-43	0	15	ns
Hub EOP Output Width Skew	TFHESK	Note 8; Figure 7-43	-15	15	ns
Low-speed Hub Characteristics (as measured at connectors):					
Driver Characteristics: (Refer to Table 7-7)		Downstream ports configured as low-speed			
Hub Differential Data Delay	TLHDD	Note 7, 8; Figure 7-42		300	ns
Hub Differential Driver Jitter (including cable):		Note 7, 8; Figure 7-42			
Downstream port :					
To Next Transition	TLDHJ1		-45	45	ns
For Paired Transitions	TLDHJ2		-15	15	ns
Upstream port:					
To Next Transition	TLUHJ1		-45	45	ns
For Paired Transitions	TLUHJ2		-45	45	ns
Data Bit Width Distortion after SOP	TLSOP	Note 8; Figure 7-42	-60	60	ns
Hub EOP Delay Relative to THDD	TLEOPD	Note 8; Figure 7-43	0	200	ns
Hub EOP Output Width Skew	TLHESK	Note 8; Figure 7-43	-300	+300	ns

Table 7-9. Cable Characteristics (Note 14)

Parameter	Symbol	Conditions	Min	Max	Units
V_{BUS} Voltage drop for detachable cables	V_{BUSD}	Section 7.2.2		125	mV
GND Voltage drop (for all cables)	V_{GNDD}	Section 7.2.2		125	mV
Differential Cable Impedance (full-speed)	Z_O	(90Ω ±15%);	76.5	103.5	Ω
Cable Delay (one way)		Section 7.1.16			
Full-speed	T_{FSCBL}			26	ns
Low-speed	T_{LSCBL}			18	ns
Cable Skew	T_{SKEW}	Section 7.1.3		400	ps
Unmated Contact Capacitance	C_{UC}	Section 6.7		2	pF

Note 1: Measured at A plug

Note 2: Measured at A receptacle

Note 3: Measured at B receptacle

Note 4: Measured at A or B connector

Note 5: Measured with RL of 1.425kΩ to 3.6V

Note 6: Measured with RL of 14.25kΩ to GND

Note 7: Timing difference between the differential data signals

Note 8: Measured at crossover point of differential data signals

Note 9: The maximum load specification is the maximum effective capacitive load allowed that meets the target V_{BUS} drop of 330mV

Note 10: Excluding the first transition from the Idle state

Note 11: The two transitions should be a (nominal) bit time apart

Note 12: For both transitions of differential signaling

Note 13: Must accept as valid EOP

Note 14: Single-ended capacitance of D+ or D- is the capacitance of D+/D- to all other conductors and, if present, shield in the cable. I.e., to measure the single-ended capacitance of D+, short D-, VBUS, GND and the shield line together and measure the capacitance of D+ to the other conductors.

Note 15: For high power devices (non-hubs) when enabled for remote wakeup

Table 7-10. Hub Event Timings

Event Description	Symbol	Conditions	Min	Max	Unit
Time to detect a downstream port connect event Awake Hub Suspended Hub	TDCNN	Section 11.5 and Section 7.1.7.1	 2.5 2.5	 2000 12000	 µs µs
Time to detect a disconnect event at a downstream port: Awake Hub Suspended Hub	TDDIS	Section 7.1.7.1	 2 2	 2.5 10000.0	 µs µs
Duration of driving resume to a downstream port; Only from a controlling hub	TDRSMDN	Nominal; Section 7.1.7.5 and Section 11.5	20		ms
Time from detecting downstream resume to rebroadcast.	TURSM	Section 7.1.7.5		100	µs
Duration of driving reset to a downstream port	TDRST	Only for a SetPortFeature (PORT_RESET) request; Section 7.1.7.3 and Section 11.5	10	20	ms
Overall duration of driving reset to downstream port, root hub	TDRSTR	only for root hubs; Section 7.1.7.3	50		ms
Maximum interval between reset segments used to create TDRSTR	TRHRSI	only for root hubs; each reset pulse must be of length TDRST; Section 7.1.7.3		3	ms
Time to evaluate device speed after reset	TDSPDEV	*Optional* Section 11.8.2	2.5	1000	µs
Time to detect a long K from upstream	TURLK	Section 11.6.1	2.5	100	µs
Time to detect a long SE0 from upstream	TURLSE0	Section 11.6.1	2.5	10000	µs
Duration of repeating SE0 upstream	TURPSE0	Section 11.6.2		23	FS bit times
Duration of sending SE0 upstream after EOF1	TUDEOP	*Optional* Section 11.6.2		2	FS bit times

Table 7-11. Device Event Timings

Parameter	Symbol	Conditions	Min	Max	Units
Time from internal power good to device pulling D+/D- beyond VIHZ (min) (signaling attach)	TSIGATT	Figure 7-19		100	ms
Debounce interval provided by USB system software after attach	TATTDB	Figure 7-19		100	ms
Maximum time a device can draw power >suspend power when bus is continuously in idle state	T2SUSP	Section 7.1.7.4		10	ms
Maximum duration of suspend averaging interval	TSUSAVGI	Section 7.2.3		1	s
Period of idle bus before device can initiate resume	TWTRSM	Device must be remote-wakeup enabled. Section 7.1.7.5	5		ms
Duration of driving resume upstream	TDRSMUP	Section 7.1.7.5	1	15	ms
Resume Recovery Time	TRSMRCY	Provided by USB System Software; Section 7.1.7.5	10		ms
Time to detect a reset from upstream	TDETRST	Section 7.1.7.3	2.5	10000	μs
Reset Recovery Time	TRSTRCY	Section 7.1.7.3		10	ms
Inter-packet Delay	TIPD	Section 7.1.18	2		bit times
Inter-packet delay for device response w/detachable cable	TRSPIPD1	Section 7.1.18		6.5	bit times
Inter-packet delay for device response w/captive cable	TRSPIPD2	Section 7.1.18		7.5	bit times
SetAddress() Completion Time	TDSETADDR	Section 9.2.6.3		50	ms
Time to complete standard request with no data	TDRQCMPLTND	Section 9.2.6.4		50	ms

Table 7-11. Device Event Timings (Continued)

Parameter	Symbol	Conditions	Min	Max	Units
Time to deliver first and subsequent (except last) data for standard request	TDRETDATA1	Section 9.2.6.4		500	ms
Time to deliver last data for standard request	TDRETDATAN	Section 9.2.6.4		50	ms

7.3.3 Timing Waveforms

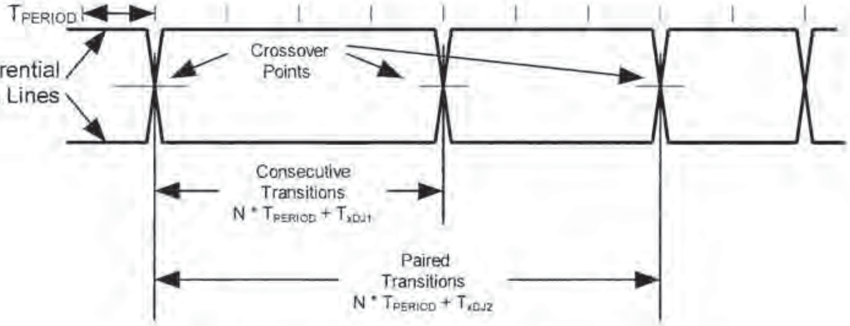


Figure 7-39. Differential Data Jitter

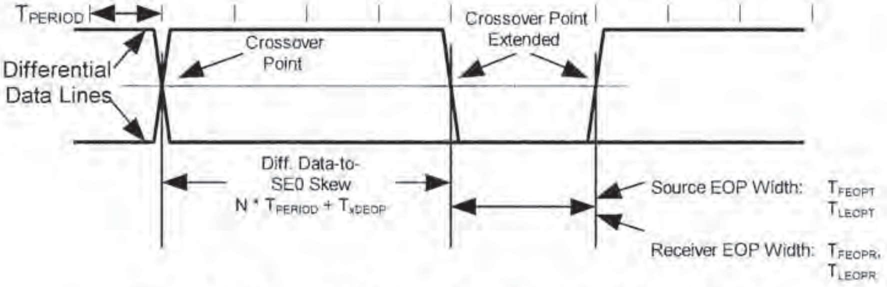


Figure 7-40. Differential-to-EOP Transition Skew and EOP Width

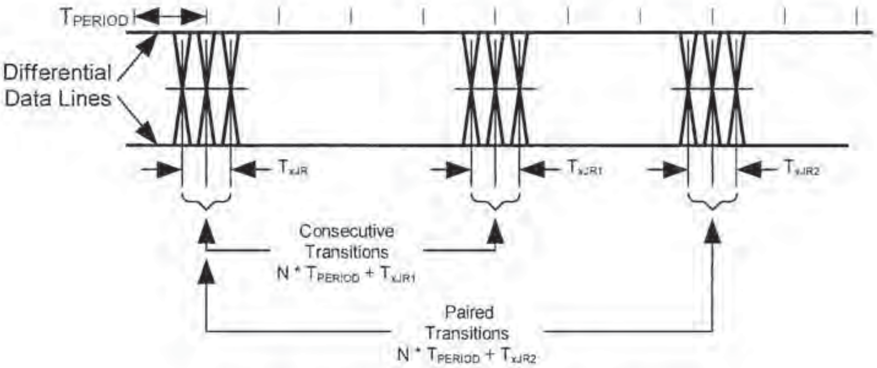
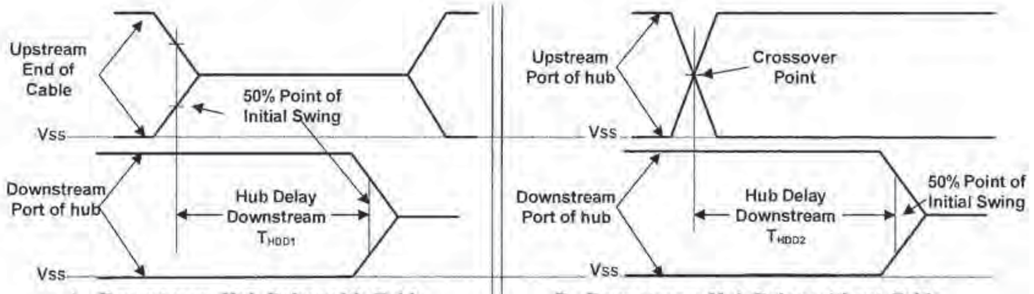
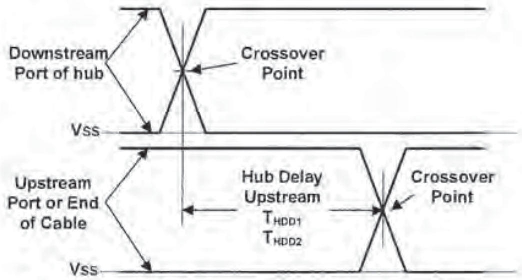


Figure 7-41. Receiver Jitter Tolerance

TPERIOD is the data rate of the receiver that can have the range as defined in Section 7.1.11



A. Downstream Hub Delay with Cable **B. Downstream Hub Delay without Cable**



C. Upstream Hub Delay with or without Cable

Hub Differential Jitter:

$T_{HDJ1} = T_{HDDx}(J) - T_{HDDx}(K)$ or $T_{HDDx}(K) - T_{HDDx}(J)$ **Consecutive Transitions**

$T_{HDJ2} = T_{HDDx}(J) - T_{HDDx}(J)$ or $T_{HDDx}(K) - T_{HDDx}(K)$ **Paired Transitions**

Bit after SOP Width Distortion (same as data jitter for SOP and next J transition):

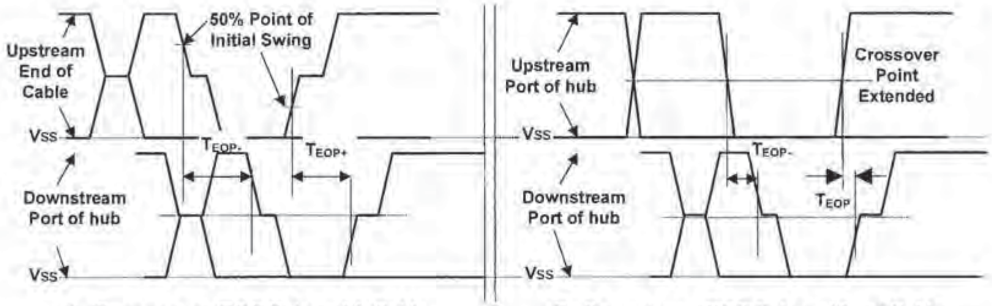
$T_{FSOP} = T_{HDDx}(\text{next J}) - T_{HDDx}(\text{SOP})$

Low-speed timings are determined in the same way for:

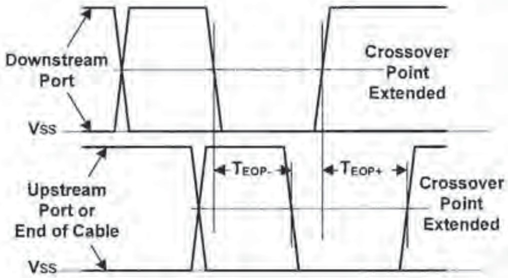
T_{LHDD}, T_{LDHJ1}, T_{LDJH2}, T_{LUHJ1}, T_{LUJH2}, and T_{LSOP}

Figure 7-42. Hub Differential Delay, Differential Jitter, and SOP Distortion

Note: Measurement locations referenced in Figure 7-42 and Figure 7-43 are specified in Figure 7-28



A. Downstream EOP Delay with Cable B. Downstream EOP Delay without Cable



C. Upstream EOP Delay with or Without Cable

EOP Delay:

$T_{FEOPD} = T_{EOPy} - T_{HDDx}$
(T_{EOPy} means that this equation applies to T_{EOP-} and T_{EOP+})

EOP Skew:
$T_{FHESK} = T_{EOP+} - T_{EOP-}$

Low-speed timings are determined in the same way for:
T_{LEOPD} and T_{LHESK}

Figure 7-43. Hub EOP Delay and EOP Skew

Chapter 8
Protocol Layer

This chapter presents a bottom-up view of the USB protocol, starting with field and packet definitions. This is followed by a description of packet transaction formats for different transaction types. Link layer flow control and transaction level fault recovery are then covered. The chapter finishes with a discussion of retry synchronization, babble, and loss of bus activity recovery.

8.1 Bit Ordering

Bits are sent out onto the bus least-significant bit (LSb) first, followed by the next LSb, through to the most-significant bit (MSb) last. In the following diagrams, packets are displayed such that both individual bits and fields are represented (in a left to right reading order) as they would move across the bus.

8.2 SYNC Field

All packets begin with a synchronization (SYNC) field, which is a coded sequence that generates a maximum edge transition density. The SYNC field appears on the bus as IDLE followed by the binary string "KJKJKJKK," in its NRZI encoding. It is used by the input circuitry to align incoming data with the local clock and is defined to be eight bits in length. SYNC serves only as a synchronization mechanism and is not shown in the following packet diagrams (refer to Section 7.1.10). The last two bits in the SYNC field are a marker that is used to identify the end of the SYNC field and, by inference, the start of the PID.

8.3 Packet Field Formats

Field formats for the token, data, and handshake packets are described in the following section. Packet bit definitions are displayed in unencoded data format. The effects of NRZI coding and bit stuffing have been removed for the sake of clarity. All packets have distinct Start- and End-of-Packet delimiters. The Start-of-Packet (SOP) delimeter is part of the SYNC field, and the End-of-Packet (EOP) delimiter is described in Chapter 7.

8.3.1 Packet Identifier Field

A packet identifier (PID) immediately follows the SYNC field of every USB packet. A PID consists of a four-bit packet type field followed by a four-bit check field as shown in Figure 8-1. The PID indicates the type of packet and, by inference, the format of the packet and the type of error detection applied to the packet. The four-bit check field of the PID ensures reliable decoding of the PID so that the remainder of the packet is interpreted correctly. The PID check field is generated by performing a one's complement of the packet type field. A PID error exists if the four PID check bits are not complements of their respective packet identifier bits.

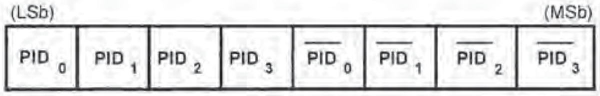


Figure 8-1. PID Format

The host and all functions must perform a complete decoding of all received PID fields. Any PID received with a failed check field or which decodes to a non-defined value is assumed to be corrupted and it, as well as the remainder of the packet, is ignored by the packet receiver. If a function receives an otherwise valid PID for a transaction type or direction that it does not support, the function must not respond. For example, an IN-only endpoint must ignore an OUT token. PID types, codings, and descriptions are listed in Table 8-1.

Table 8-1. PID Types

PID Type	PID Name	PID[3:0]*	Description
Token	OUT	0001B	Address + endpoint number in host-to-function transaction
	IN	1001B	Address + endpoint number in function-to-host transaction
	SOF	0101B	Start-of-Frame marker and frame number
	SETUP	1101B	Address + endpoint number in host-to-function transaction for SETUP to a control pipe
Data	DATA0	0011B	Data packet PID even
	DATA1	1011B	Data packet PID odd
Handshake	ACK	0010B	Receiver accepts error-free data packet
	NAK	1010B	Rx device cannot accept data or Tx device cannot send data
	STALL	1110B	Endpoint is halted or a control pipe request is not supported.
Special	PRE	1100B	Host-issued preamble. Enables downstream bus traffic to low-speed devices.

*Note: PID bits are shown in MSb order. When sent on the USB, the rightmost bit (bit 0) will be sent first.

PIDs are divided into four coding groups: token, data, handshake, and special, with the first two transmitted PID bits (PID<0:1>) indicating which group. This accounts for the distribution of PID codes.

8.3.2 Address Fields

Function endpoints are addressed using two fields: the function address field and the endpoint field. A function needs to fully decode both address and endpoint fields. Address or endpoint aliasing is not permitted, and a mismatch on either field must cause the token to be ignored. Accesses to non-initialized endpoints will also cause the token to be ignored.

8.3.2.1 Address Field

The function address (ADDR) field specifies the function, via its address, that is either the source or destination of a data packet, depending on the value of the token PID. As shown in Figure 8-2, a total of 128 addresses are specified as ADDR<6:0>. The ADDR field is specified for IN, SETUP, and OUT tokens. By definition, each ADDR value defines a single function. Upon reset and power-up, a function's address defaults to a value of zero and must be programmed by the host during the enumeration process. Function address zero is reserved as the default address and may not be assigned to any other use.

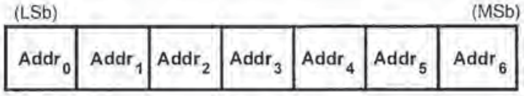


Figure 8-2. ADDR Field

8.3.2.2 Endpoint Field

An additional four-bit endpoint (ENDP) field, shown in Figure 8-3 permits more flexible addressing of functions in which more than one endpoint is required. Except for endpoint address zero, endpoint numbers are function-specific. The endpoint field is defined for IN, SETUP, and OUT token PIDs only. All functions must support a control pipe at endpoint number zero (the Default Control Pipe). Low-speed devices support a maximum of three pipes per function: a control pipe at endpoint number zero plus two additional pipes (either two control pipes, a control pipe and a interrupt endpoint, or two interrupt endpoints). Full-speed functions may support up to the maximum of 16 endpoint numbers of any type.

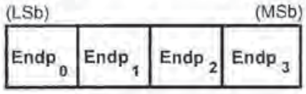


Figure 8-3. Endpoint Field

8.3.3 Frame Number Field

The frame number field is an 11-bit field that is incremented by the host on a per-frame basis. The frame number field rolls over upon reaching its maximum value of 7FFH, and is sent only in SOF tokens at the start of each frame.

8.3.4 Data Field

The data field may range from zero to 1,023 bytes and must be an integral number of bytes. Figure 8-4 shows the format for multiple bytes. Data bits within each byte are shifted out LSb first.

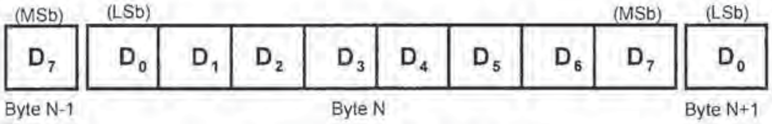


Figure 8-4. Data Field Format

Data packet size varies with the transfer type, as described in Chapter 5.

8.3.5 Cyclic Redundancy Checks

Cyclic redundancy checks (CRCs) are used to protect all non-PID fields in token and data packets. In this context, these fields are considered to be protected fields. The PID is not included in the CRC check of a packet containing a CRC. All CRCs are generated over their respective fields in the transmitter before bit stuffing is performed. Similarly, CRCs are decoded in the receiver after stuffed bits have been removed. Token and data packet CRCs provide 100% coverage for all single- and double-bit errors. A failed CRC is considered to indicate that one or more of the protected fields is corrupted and causes the receiver to ignore those fields, and, in most cases, the entire packet.

For CRC generation and checking, the shift registers in the generator and checker are seeded with an all-ones pattern. For each data bit sent or received, the high order bit of the current remainder is XORed with the data bit and then the remainder is shifted left one bit and the low-order bit set to zero. If the result of that XOR is one, then the remainder is XORed with the generator polynomial.

When the last bit of the checked field is sent, the CRC in the generator is inverted and sent to the checker MSb first. When the last bit of the CRC is received by the checker and no errors have occurred, the remainder will be equal to the polynomial residual.

A CRC error exists if the computed checksum remainder at the end of a packet reception does not match the residual.

Bit stuffing requirements must be met for the CRC, and this includes the need to insert a zero at the end of a CRC if the preceding six bits were all ones.

8.3.5.1 Token CRCs

A five-bit CRC field is provided for tokens and covers the ADDR and ENDP fields of IN, SETUP, and OUT tokens or the time stamp field of an SOF token. The generator polynomial is:

$$G(X) = X^5 + X^2 + 1$$

The binary bit pattern that represents this polynomial is 00101B. If all token bits are received without error, the five-bit residual at the receiver will be 01100B.

8.3.5.2 Data CRCs

The data CRC is a 16-bit polynomial applied over the data field of a data packet. The generating polynomial is:

$$G(X) = X^{16} + X^{15} + X^2 + 1$$

The binary bit pattern that represents this polynomial is 1000000000000101B. If all data and CRC bits are received without error, the 16-bit residual will be 1000000000001101B.

8.4 Packet Formats

This section shows packet formats for token, data, and handshake packets. Fields within a packet are displayed in these figures in the order in which bits are shifted out onto the bus.

8.4.1 Token Packets

Figure 8-5 shows the field formats for a token packet. A token consists of a PID, specifying either IN, OUT, or SETUP packet type; and ADDR and ENDP fields. For OUT and SETUP transactions, the address and endpoint fields uniquely identify the endpoint that will receive the subsequent Data packet. For IN transactions, these fields uniquely identify which endpoint should transmit a Data packet. Only the host can issue token packets. IN PIDs define a Data transaction from a function to the host. OUT and SETUP PIDs define Data transactions from the host to a function.

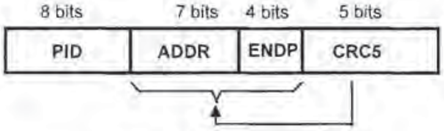


Figure 8-5. Token Format

Token packets have a five-bit CRC that covers the address and endpoint fields as shown above. The CRC does not cover the PID, which has its own check field. Token and SOF packets are delimited by an EOP after three bytes of packet field data. If a packet decodes as an otherwise valid token or SOF but does not terminate with an EOP after three bytes, it must be considered invalid and ignored by the receiver.

8.4.2 Start-of-Frame Packets

Start-of-Frame (SOF) packets are issued by the host at a nominal rate of once every 1.00ms ±0.0005ms. SOF packets consist of a PID indicating packet type followed by an 11-bit frame number field as illustrated in Figure 8-6.

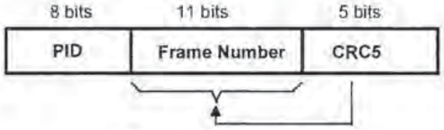


Figure 8-6. SOF Packet

The SOF token comprises the token-only transaction that distributes an SOF marker and accompanying frame number at precisely timed intervals corresponding to the start of each frame. All full-speed functions, including hubs, receive the SOF packet. The SOF token does not cause any receiving function to generate a return packet; therefore, SOF delivery to any given function cannot be guaranteed. The SOF packet delivers two pieces of timing information. A function is informed that an SOF has occurred when it detects the SOF PID. Frame timing sensitive functions, which do not need to keep track of frame number (e.g., a hub), need only decode the SOF PID; they can ignore the frame number and its CRC. If a function needs to track frame number, it must comprehend both the PID and the time stamp. Full-speed devices that have no particular need for bus timing information may ignore the SOF packet.

8.4.3 Data Packets

A data packet consists of a PID, a data field containing zero or more bytes of data, and a CRC as shown in Figure 8-7. There are two types of data packets, identified by differing PIDs: DATA0 and DATA1. Two data packet PIDs are defined to support data toggle synchronization (refer to Section 8.6).

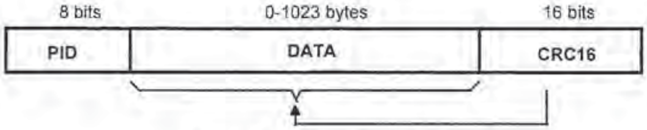


Figure 8-7. Data Packet Format

Data must always be sent in integral numbers of bytes. The data CRC is computed over only the data field in the packet and does not include the PID, which has its own check field.

8.4.4 Handshake Packets

Handshake packets, as shown in Figure 8-8, consist of only a PID. Handshake packets are used to report the status of a data transaction and can return values indicating successful reception of data, command acceptance or rejection, flow control, and halt conditions. Only transaction types that support flow control can return handshakes. Handshakes are always returned in the handshake phase of a transaction and may be returned, instead of data, in the data phase. Handshake packets are delimited by an EOP after one byte of packet field. If a packet decodes as an otherwise valid handshake but does not terminate with an EOP after one byte, it must be considered invalid and ignored by the receiver.

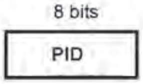


Figure 8-8. Handshake Packet

There are three types of handshake packets:

- **ACK** indicates that the data packet was received without bit stuff or CRC errors over the data field and that the data PID was received correctly. ACK may be issued either when sequence bits match and the receiver can accept data or when sequence bits mismatch and the sender and receiver must resynchronize to each other (refer to Section 8.6 for details). An ACK handshake is applicable only in transactions in which data has been transmitted and where a handshake is expected. ACK can be returned by the host for IN transactions and by a function for OUT or SETUP transactions.
- **NAK** indicates that a function was unable to accept data from the host (OUT) or that a function has no data to transmit to the host (IN). NAK can only be returned by functions in the data phase of IN transactions or the handshake phase of OUT transactions. The host can never issue NAK. NAK is used for flow control purposes to indicate that a function is temporarily unable to transmit or receive data, but will eventually be able to do so without need of host intervention.
- **STALL** is returned by a function in response to an IN token or after the data phase of an OUT transaction (see Figure 8-9 and Figure 8-13). STALL indicates that a function is unable to transmit or receive data, or that a control pipe request is not supported. The host is not permitted to return a STALL under any condition.

The STALL handshake is used by a device in one of two distinct occasions. The first case, known as "functional stall," is when the *Halt* feature associated the endpoint is set. (The *Halt* feature is specified in Chapter 9 of this document.) A special case of the functional stall is the "commanded stall." Commanded stall occurs when the host explicitly sets the endpoint's *Halt* feature, as detailed in Chapter 9. Once a function's endpoint is halted, the function must continue returning STALL until the condition causing the halt has been cleared through host intervention.

The second case, known as "protocol stall," is detailed in Section 8.5.2. Protocol stall is unique to control pipes. Protocol stall differs from functional stall in meaning and duration. A protocol STALL is returned during the Data or Status stage of a control transfer, and the STALL condition terminates at the beginning of the next control transfer (Setup). The remainder of this section refers to the general case of a functional stall.

8.4.5 Handshake Responses

Transmitting and receiving functions must return handshakes based upon an order of precedence detailed in Table 8-2 through Table 8-4. Not all handshakes are allowed, depending on the transaction type and whether the handshake is being issued by a function or the host. Note that if an error occurs during the transmission of the token to the function, the function will not respond with any packets until the next token is received and successfully decoded.

8.4.5.1 Function Response to IN Transactions

Table 8-2 shows the possible responses a function may make in response to an IN token. If the function is unable to send data, due to a halt or a flow control condition, it issues a STALL or NAK handshake, respectively. If the function is able to issue data, it does so. If the received token is corrupted, the function returns no response.

Table 8-2. Function Responses to IN Transactions

Token Received Corrupted	Function Tx Endpoint Halt Feature	Function Can Transmit Data	Action Taken
Yes	Don't care	Don't care	Return no response
No	Set	Don't care	Issue STALL handshake
No	Not set	No	Issue NAK handshake
No	Not set	Yes	Issue data packet

8.4.5.2 Host Response to IN Transactions

Table 8-3 shows the host response to an IN transaction. The host is able to return only one type of handshake: ACK. If the host receives a corrupted data packet, it discards the data and issues no response. If the host cannot accept data from a function, (due to problems such as internal buffer overrun) this condition is considered to be an error and the host returns no response. If the host is able to accept data and the data packet is received error-free, the host accepts the data and issues an ACK handshake.

Table 8-3. Host Responses to IN Transactions

Data Packet Corrupted	Host Can Accept Data	Handshake Returned by Host
Yes	N/A	Discard data, return no response
No	No	Discard data, return no response
No	Yes	Accept data, issue ACK

8.4.5.3 Function Response to an OUT Transaction

Handshake responses for an OUT transaction are shown in Table 8-4. Assuming successful token decode, a function, upon receiving a data packet, may return any one of the three handshake types. If the data packet was corrupted, the function returns no handshake. If the data packet was received error-free and the function's receiving endpoint is halted, the function returns STALL. If the transaction is maintaining sequence bit synchronization and a mismatch is detected (refer to Section 8.6 for details), then the function returns ACK and discards the data. If the function can accept the data and has received the data error-free, it returns ACK. If the function cannot accept the data packet due to flow control reasons, it returns NAK.

Table 8-4. Function Responses to OUT Transactions in Order of Precedence

Data Packet Corrupted	Receiver Halt Feature	Sequence Bits Match	Function Can Accept Data	Handshake Returned by Function
Yes	N/A	N/A	N/A	None
No	Set	N/A	N/A	STALL
No	Not set	No	N/A	ACK
No	Not set	Yes	Yes	ACK
No	Not set	Yes	No	NAK

8.4.5.4 Function Response to a SETUP Transaction

SETUP defines a special type of host-to-function data transaction that permits the host to initialize an endpoint's synchronization bits to those of the host. Upon receiving a SETUP token, a function must accept the data. A function may not respond to a SETUP token with either STALL or NAK and the receiving function must accept the data packet that follows the SETUP token. If a non-control endpoint receives a SETUP token, it must ignore the transaction and return no response.

8.5 Transaction Formats

Packet transaction format varies depending on the endpoint type. There are four endpoint types: bulk, control, interrupt, and isochronous.

8.5.1 Bulk Transactions

Bulk transaction types are characterized by the ability to guarantee error-free delivery of data between the host and a function by means of error detection and retry. Bulk transactions use a three-phase transaction consisting of token, data, and handshake packets as shown in Figure 8-9. Under certain flow control and halt conditions, the data phase may be replaced with a handshake resulting in a two-phase transaction in which no data is transmitted.

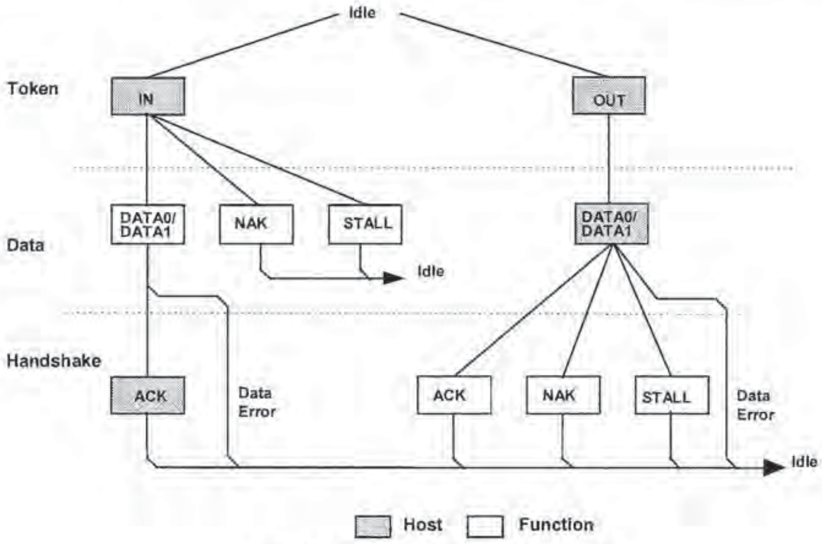


Figure 8-9. Bulk Transaction Format

When the host is ready to receive bulk data, it issues an IN token. The function endpoint responds by returning either a data packet or, should it be unable to return data, a NAK or STALL handshake. NAK indicates that the function is temporarily unable to return data, while STALL indicates that the endpoint is permanently halted and requires USB System Software intervention. If the host receives a valid data packet, it responds with an ACK handshake. If the host detects an error while receiving data, it returns no handshake packet to the function.

When the host is ready to transmit bulk data, it first issues an OUT token packet followed by a data packet. If the data is received without error by the function it will return one of three handshakes:

- ACK indicates that the data packet was received without errors and informs the host that that it may send the next packet in the sequence.
- NAK indicates that the data was received without error but that the host should resend the data because the function was in a temporary condition preventing it from accepting the data (e.g., buffer full).
- If the endpoint was halted, STALL is returned to indicate that the host should not retry the transmission because there is an error condition on the function.

If the data packet was received with a CRC or bit stuff error, no handshake is returned.

Figure 8-10 shows the sequence bit and data PID usage for bulk reads and writes. Data packet synchronization is achieved via use of the data sequence toggle bits and the DATA0/DATA1 PIDs. A bulk endpoint's toggle sequence is initialized to DATA0 when the endpoint experiences any configuration event (configuration events are explained in Sections 9.1.1.5 and 9.4.5). Data toggle on an endpoint is NOT initialized as the direct result of a short packet transfer or the retirement of an IRP.

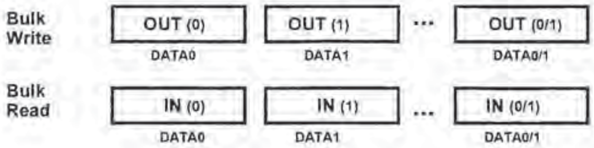


Figure 8-10. Bulk Reads and Writes

The host always initializes the first transaction of a bus transfer to the DATA0 PID with a configuration event. The second transaction uses a DATA1 PID, and successive data transfers alternate for the remainder of the bulk transfer. The data packet transmitter toggles upon receipt of ACK, and the receiver toggles upon receipt and acceptance of a valid data packet (refer to Section 8.6).

8.5.2 Control Transfers

Control transfers minimally have two transaction stages: Setup and Status. A control transfer may optionally contain a Data stage between the Setup and Status stages. During the Setup stage, a SETUP transaction is used to transmit information to the control endpoint of a function. SETUP transactions are similar in format to an OUT, but use a SETUP rather than an OUT PID. Figure 8-11 shows the SETUP transaction format. A SETUP always uses a DATA0 PID for the data field of the SETUP transaction. The function receiving a SETUP must accept the SETUP data and respond with ACK, if the data is corrupted, discard the data and return no handshake.

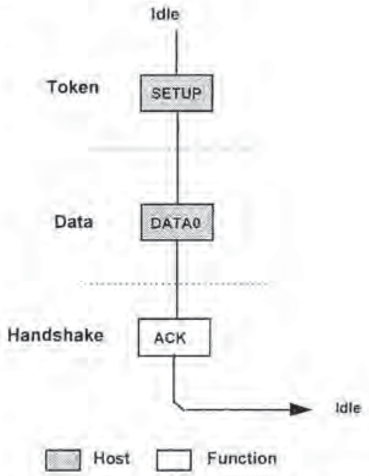


Figure 8-11. Control SETUP Transaction

The Data stage, if present, of a control transfer consists of one or more IN or OUT transactions and follows the same protocol rules as bulk transfers. All the transactions in the Data stage must be in the same direction (i.e., all INs or all OUTs). The amount of data to be sent during the data phase and its direction are specified during the Setup stage. If the amount of data exceeds the prenegotiated data packet size, the data is sent in multiple transactions (INs or OUTs) that carry the maximum packet size. Any remaining data is sent as a residual in the last transaction.

The Status stage of a control transfer is the last operation in the sequence. A Status stage is delineated by a change in direction of data flow from the previous stage and always uses a DATA1 PID. If, for example, the Data stage consists of OUTs, the status is a single IN transaction. If the control sequence has no Data stage, then it consists of a Setup stage followed by a Status stage consisting of an IN transaction. Figure 8-12 shows the transaction order, the data sequence bit value, and the data PID types for control read and write sequences. The sequence bits are displayed in parentheses.

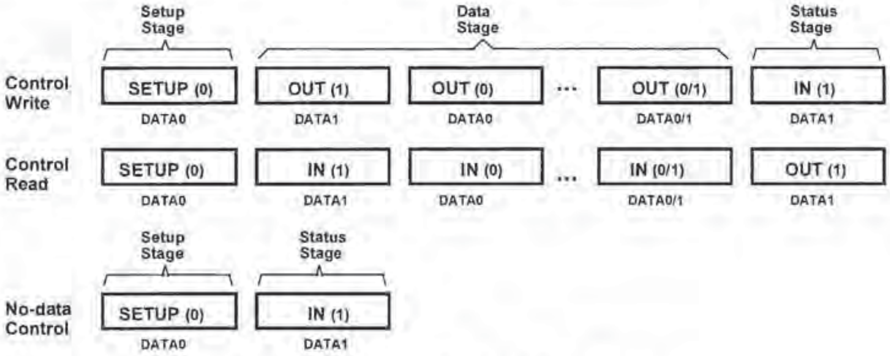


Figure 8-12. Control Read and Write Sequences

When a STALL handshake is sent by a control endpoint in either the Data or Status stages of a control transfer, a STALL handshake must be returned on all succeeding accesses to that endpoint until a SETUP PID is received. The endpoint is not required to return a STALL handshake after it receives a subsequent SETUP PID.

8.5.2.1 Reporting Status Results

The Status stage reports to the host the outcome of the previous Setup and Data stages of the transfer. Three possible results may be returned:

- The command sequence completed successfully.
- The command sequence failed to complete.
- The function is still busy completing command.

Status reporting is always in the function-to-host direction. The Table 8-5 summarizes the type of responses required for each. Control write transfers return status information in the data phase of the Status stage transaction. Control read transfers return status information in the handshake phase of a Status stage transaction, after the host has issued a zero-length data packet during the previous data phase.

Table 8-5. Status Stage Responses

Status Response	Control Write Transfer (sent during data phase)	Control Read Transfer (send during handshake phase)
Function completes	Zero-length data packet	ACK handshake
Function has an error	STALL handshake	STALL handshake
Function is busy	NAK handshake	NAK handshake

For control reads, the host sends an OUT token to the control pipe to initiate the Status stage. The host may only send a zero-length data packet in this phase but the function may accept any length packet as a valid status inquiry. The pipe's handshake response to this data packet indicates the current status. NAK indicates that the function is still processing the command and that the host should continue the Status stage. ACK indicates that the function has completed the command and is ready to accept a new command. STALL indicates that the function has an error that prevents it from completing the command.

For control writes, the host sends an IN token to the control pipe to initiate the Status stage. The function responds with either a handshake or a zero-length data packet to indicate its current status. NAK indicates that the function is still processing the command and that the host should continue the Status stage; return of a zero-length packet indicates normal completion of the command; and STALL indicates that the function cannot complete the command. The function expects the host to respond to the data packet in the Status stage with ACK. If the function does not receive ACK, it remains in the Status stage of the command and will continue to return the zero-length data packet for as long as the host continues to send IN tokens.

If during a Data stage a command pipe is sent more data or is requested to return more data than was indicated in the Setup stage (see Section 8.5.2.2), it should return STALL. If a control pipe returns STALL during the Data stage, there will be no Status stage for that control transfer.

8.5.2.2 Variable-length Data Stage

A control pipe may have a variable-length data phase in which the host request more data than is contained in the specified data structure. When all of the data structure is returned to the host, the function should indicate that the Data stage is ended by returning a packet that is shorter than the *MaxPacketSize* for the pipe. If the data structure is an exact multiple of *wMaxPacketSize* for the pipe, the funtion will return a zero-length packet to indicate the end of the Data stage.

8.5.2.3 Error Handling on the Last Data Transaction

If the ACK handshake on an IN transaction is corrupted, the function and the host will temporarily disagree on whether the transaction was successful. If the transaction is followed by another IN, the toggle retry mechanism will detect the mismatch and recover from the error. If the ACK was on the last IN of a Data stage, the toggle retry mechanism cannot be used and an alternative scheme must be used.

The host that successfully received the data of the last IN will send ACK.. Later, the host will issue an OUT token to start the Status stage of the transfer. If the function did not receive the ACK that ended the Data stage, the function will interpret the start of the Status stage as verification that the host successfully received the data. Control writes do not have this ambiguity. If an ACK handshake on an OUT gets corrupted, the host does not advance to the Status stage and retries the last data instead. A detailed analysis of retry policy is presented in Section 8.6.4.

8.5.2.4 STALL Handshakes Returned by Control Pipes

Control pipes have the unique ability to return a STALL handshake due to function problems in control transfers. If the device is unable to complete a command, it returns a STALL in the Data and/or Status stages of the control transfer. Unlike the case of a functional stall, protocol stall does not indicate an error with the device. The protocol stall condition lasts until the receipt of the next SETUP transaction and the function will return STALL in response to any IN or OUT transaction on the pipe until the SETUP transaction is received. In general, protocol stall indicates that the request or its parameters is not understood by the device and thus provides a mechanism for extending USB requests.

A control pipe may also support functional stall as well, but this is not recommended. This is a degenerative case, because a functional stall on a control pipe indicates that it has lost the ability to communicate with the host. If the control pipe does support functional stall, then it must possess a *Halt* feature, which can be set or cleared by the host. Chapter 9 details how to treat the special case of a *Halt* feature on a control pipe. A well-designed device will associate all of its functions and *Halt* features with non-control endpoints. The control pipes should be reserved for servicing USB requests.

8.5.3 Interrupt Transactions

Interrupt transactions may consist of IN or OUT transfers. Upon receipt of an IN token, a function may return data, NAK, or STALL. If the endpoint has no new interrupt information to return (i.e., no interrupt is pending), the function returns a NAK handshake during the data phase. If the *Halt* feature is set for the interrupt endpoint, the function will return a STALL handshake. If an interrupt is pending, the function returns the interrupt information as a data packet. The host, in response to receipt of the data packet, issues either an ACK handshake if data was received error-free or returns no handshake if the data packet was received corrupted. Figure 8-13 shows the interrupt transaction format.

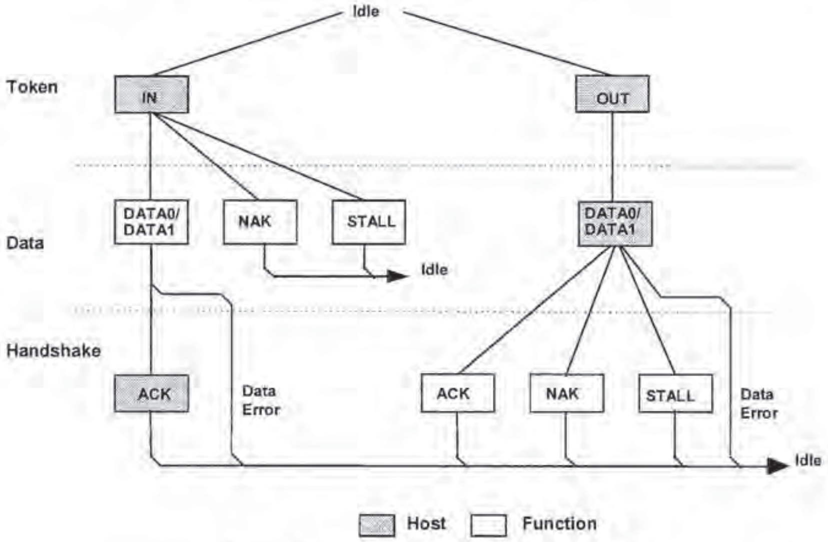


Figure 8-13. Interrupt Transaction Format

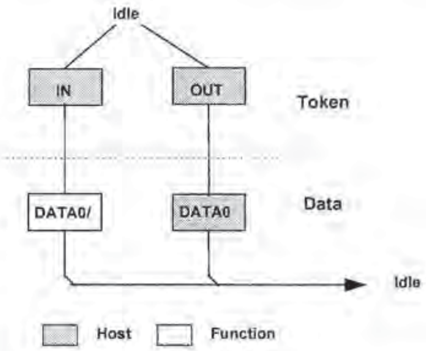
When an endpoint is using the interrupt transfer mechanism for actual interrupt data, the data toggle protocol must be followed. This allows the function to know that the data has been received by the host and the event condition may be cleared. This "guaranteed" delivery of events allows the function to only send the interrupt information until it has been received by the host rather than having to send the interrupt data every time the function is polled and until the USB System Software clears the interrupt condition.

When used in the toggle mode, an interrupt endpoint is initialized to the DATA0 PID by any configuration event on the endpoint and behaves the same as the bulk transactions shown in Figure 8-10.

An interrupt endpoint may also be used to communicate rate feedback information for certain types of isochronous functions. When used in this mode, the data toggle bits should be changed after each data packet is sent to the host without regard to the presence or type of handshake packet. This capability is supported only for interrupt IN endpoints.

8.5.4 Isochronous Transactions

Isochronous (ISO) transactions have a token and data phase, but no handshake phase, as shown in Figure 8-14. The host issues either an IN or an OUT token followed by the data phase in which the endpoint (for INs) or the host (for OUTs) transmits data. ISO transactions do not support a handshake phase or retry capability.



See Note Below

Figure 8-14. Isochronous Transaction Format

Note: a device or Host Controller should be able to accept either DATA0 or DATA1. A device or Host Controller should only send DATA0.

ISO transactions do not support toggle sequencing.

8.6 Data Toggle Synchronization and Retry

The USB provides a mechanism to guarantee data sequence synchronization between data transmitter and receiver across multiple transactions. This mechanism provides a means of guaranteeing that the handshake phase of a transaction was interpreted correctly by both the transmitter and receiver. Synchronization is achieved via use of the DATA0 and DATA1 PIDs and separate data toggle sequence bits for the data transmitter and receiver. Receiver sequence bits toggle only when the receiver is able to accept data and receives an error-free data packet with the correct data PID. Transmitter sequence bits toggle only when the data transmitter receives a valid ACK handshake. The data transmitter and receiver must have their sequence bits synchronized at the start of a transaction. The synchronization mechanism used varies with the transaction type. Data toggle synchronization is not supported for ISO transfers.

8.6.1 Initialization via SETUP Token

Control transfers use the SETUP token for initializing host and function sequence bits. Figure 8-15 shows the host issuing a SETUP packet to a function followed by an OUT transaction. The numbers in the circles represent the transmitter and receiver sequence bits. The function must accept the data and return ACK. When the function accepts the transaction, it must set its sequence bit so that both the host's and function's sequence bits are equal to one at the end of the SETUP transaction.

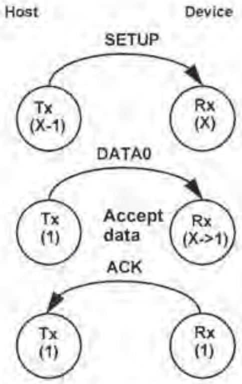


Figure 8-15. SETUP Initialization

8.6.2 Successful Data Transactions

Figure 8-16 shows the case where two successful transactions have occurred. For the data transmitter, this means that it toggles its sequence bit upon receipt of ACK. The receiver toggles its sequence bit only if it receives a valid data packet and the packet's data PID matches the current value of its sequence bit. The transmitter only toggles its sequence bit after it receives and ACK to a data packet.

During each transaction, the receiver compares the transmitter sequence bit (encoded in the data packet PID as either DATA0 or DATA1) with its receiver sequence bit. If data cannot be accepted, the receiver must issue NAK and the sequence bits of both the transmitter and receiver remain unchanged. If data can be accepted and the receiver's sequence bit matches the PID sequence bit, then data is accepted and the sequence bit is toggled. Two-phase transactions in which there is no data packet leave the transmitter and receiver sequence bits unchanged.

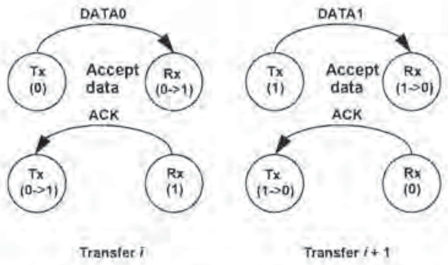


Figure 8-16. Consecutive Transactions

8.6.3 Data Corrupted or Not Accepted

If data cannot be accepted or the received data packet is corrupted, the receiver will issue a NAK or STALL handshake, or timeout, depending on the circumstances, and the receiver will not toggle its sequence bit. Figure 8-17 shows the case where a transaction is NAKed and then retried. Any non-ACK handshake or timeout will generate similar retry behavior. The transmitter, having not received an ACK handshake, will not toggle its sequence bit. As a result, a failed data packet transaction leaves the transmitter's and receiver's sequence bits synchronized and untoggled. The transaction will then be retried and, if successful, will cause both transmitter and receiver sequence bits to toggle.

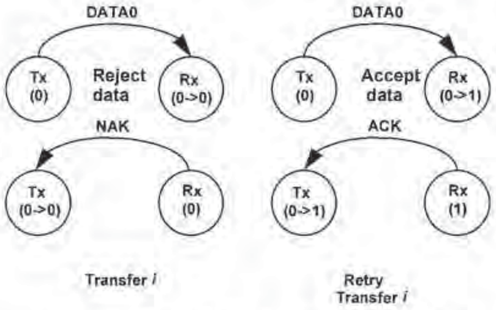


Figure 8-17. NAKed Transaction with Retry

8.6.4 Corrupted ACK Handshake

The transmitter is the last and only agent to know for sure whether a transaction has been successful, due to its receiving an ACK handshake. A lost or corrupted ACK handshake can lead to a temporary loss of synchronization between transmitter and receiver as shown in Figure 8-18. Here the transmitter issues a valid data packet, which is successfully acquired by the receiver; however, the ACK handshake is corrupted.

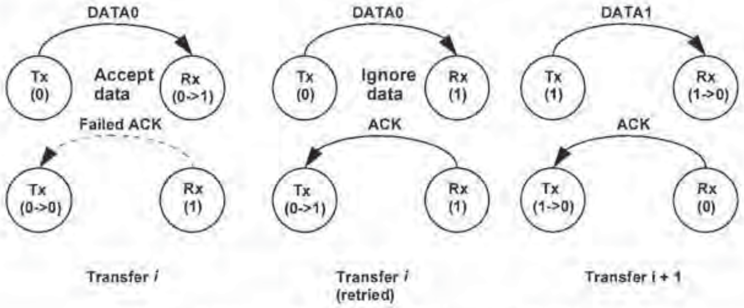


Figure 8-18. Corrupted ACK Handshake with Retry

At the end of transaction *i*, there is a temporary loss of coherency between transmitter and receiver, as evidenced by the mismatch between their respective sequence bits. The receiver has received good data, but the transmitter does not know whether it has successfully sent data. On the next transaction, the transmitter will resend the previous data using the previous DATA0 PID. The receiver's sequence bit and the data PID will not match, so the receiver knows that it has previously accepted this data. Consequently, it discards the incoming data packet and does not toggle its sequence bit. The receiver then issues ACK, which causes the transmitter to regard the retried transaction as successful. Receipt of ACK causes the transmitter to toggle its sequence bit. At the beginning of transaction *i*+1, the sequence bits have toggled and are again synchronized.

The data transmitter must guarantee that any retried data packet is identical (same length and content) as that sent in the original transaction. If the data transmitter is unable, because of problems such as a buffer underrun condition, to transmit the identical amount of data as was in the original data packet, it must abort the transaction by generating a bit stuffing violation. This causes a detectable error at the receiver and guarantees that a partial packet will not be interpreted as a good packet. The transmitter should not try to force an error at the receiver by sending a known bad CRC. A combination of a bad packet with a "bad" CRC may be interpreted by the receiver as a good packet.

8.6.5 Low-speed Transactions

The USB supports signaling at two speeds: full-speed signaling at 12.0Mb/s and low-speed signaling at 1.5Mb/s. Hubs disable downstream bus traffic to all ports to which low-speed devices are attached during full-speed downstream signaling. This is required both for EMI reasons and to prevent any possibility that a low-speed device might misinterpret downstream a full-speed packet as being addressed to it.
Figure 8-19 shows an IN low-speed transaction in which the host issues a token and handshake and receives a data packet.

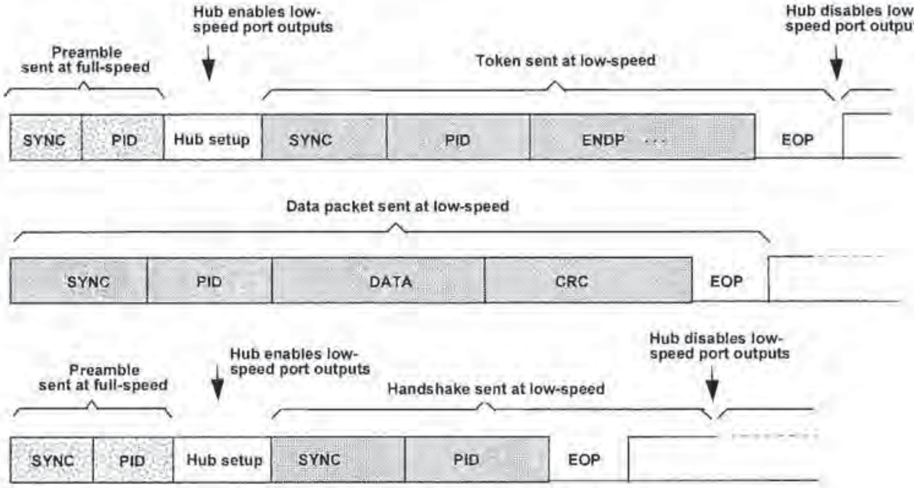


Figure 8-19. Low-speed Transaction

All downstream packets transmitted to low-speed devices require a preamble. The preamble consists of a SYNC followed by a PRE PID, both sent at full-speed. Hubs must comprehend the PRE PID; all other USB devices may ignore it and treat it as undefined. After the end of the preamble PID, the host must wait at least four full-speed bit times during which hubs must complete the process of enabling the repeater function on ports that are connected to low-speed devices. During this hub setup interval, hubs must drive their full-speed and low-speed ports to their respective Idle states. Hubs must be ready to repeat low-speed signaling on low-speed ports before the end of the hub setup interval. Low-speed connectivity rules are summarized below:

1. Low-speed devices are identified during the connection process and the hub ports to which they are connected are identified as low-speed.
2. All downstream low-speed packets must be prefaced with a preamble (sent at full-speed), which turns on the output buffers on low-speed hub ports.

3. Low-speed hub port output buffers are turned off upon receipt of EOP and are not turned on again until a preamble PID is detected.
4. Upstream connectivity is not affected by whether a hub port is full- or low-speed.

Low-speed signaling begins with the host issuing SYNC at low-speed, followed by the remainder of the packet. The end of the packet is identified by an End-of-Packet (EOP), at which time all hubs tear down connectivity and disable any ports to which low-speed devices are connected. Hubs do not switch ports for upstream signaling; low-speed ports remain enabled in the upstream direction for both low-speed and full-speed signaling.

Low-speed and full-speed transactions maintain a high degree of protocol commonality. However, low-speed signaling does have certain limitations which include:

- Data payload is limited to eight bytes, maximum
- Only interrupt and control types of transfers are supported
- The SOF packet is not received by low-speed devices.

8.7 Error Detection and Recovery

The USB permits reliable end-to-end communication in the presence of errors on the physical signaling layer. This includes the ability to reliably detect the vast majority of possible errors and to recover from errors on a transaction-type basis. Control transactions, for example, require a high degree of data reliability; they support end-to-end data integrity using error detection and retry. Isochronous transactions, by virtue of their bandwidth and latency requirements, do not permit retries and must tolerate a higher incidence of uncorrected errors.

8.7.1 Packet Error Categories

The USB employs three error detection mechanisms: bit stuff violations, PID check bits, and CRCs. Bit stuff violations are defined in Section 7.1.9. PID errors are defined in Section 8.3.1. CRC errors are defined in Section 8.3.5.

With the exception of the SOF token, any packet that is received corrupted causes the receiver to ignore it and discard any data or other field information that came with the packet. Table 8-6 lists error detection mechanisms, the types of packets to which they apply, and the appropriate packet receiver response.

Table 8-6. Packet Error Types

Field	Error	Action
PID	PID Check, Bit Stuff	Ignore packet
Address	Bit Stuff, Address CRC	Ignore token
Frame Number	Bit Stuff, Frame Number CRC	Ignore Frame Number field
Data	Bit Stuff, Data CRC	Discard data

8.7.2 Bus Turn-around Timing

Neither the device nor the host will send an indication that a received packet had an error. This absence of positive acknowledgement is considered to be the indication that there was an error. As a consequence of this method of error reporting, the host and USB function need to keep track of how much time has elapsed from when the transmitter completes sending a packet until it begins to receive a response. This time is referred to as the bus turn-around time. The timer starts counting on the SE0-to-'J' transition of the EOP strobe and stops counting when the Idle-to-'K' SOP transition is detected. Both devices and the host require turn-around timers. The device bus turn-around time is defined by the worst case round trip delay plus the maximum device response delay (refer to Section 7.1.18). If a response is not received within this worst case timeout, then the transmitter considers that the packet transmission has failed. USB devices timeout no sooner than 16 bit times and no latter than 18 bit times after the end of the previous EOP. If the host wishes to indicate an error condition via a timeout, it must wait at least 18 bit times before issuing the next token to ensure that all downstream devices have timed out.

As shown in Figure 8-20, the device uses its bus turn-around timer between token and data or data and handshake phases. The host uses its timer between data and handshake or token and data phases.

If the host receives a corrupted data packet, it must wait before sending out the next token. This wait interval guarantees that the host does not attempt to issue a token immediately after a false EOP.

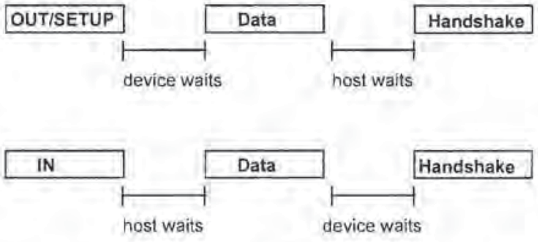


Figure 8-20. Bus Turn-around Timer Usage

8.7.3 False EOPs

False EOPs must be handled in a manner which guarantees that the packet currently in progress completes before the host or any other device attempts to transmit a new packet. If such an event were to occur, it would constitute a bus collision and have the ability to corrupt up to two consecutive transactions. Detection of false EOP relies upon the fact that a packet into which a false EOP has been inserted will appear as a truncated packet with a CRC failure. (The last 16 bits of the packet will have a very low probability of appearing to be a correct CRC.)

The host and devices handle false EOP situations differently. When a device sees a corrupted data packet, it issues no response and waits for the host to send the next token. This scheme guarantees that the device will not attempt to return a handshake while the host may still be transmitting a data packet. If a false EOP has occurred, the host data packet will eventually end, and the device will be able to detect the next token. If a device issues a data packet that gets corrupted with a false EOP, the host will ignore the packet and not issue the handshake. The device, expecting to see a handshake from the host, will timeout.

If the host receives a corrupted data packet, it assumes that a false EOP may have occurred and waits for 16 bit times to see if there is any subsequent upstream traffic. If no bus transitions are detected within the 16 bit interval and the bus remains in the Idle state, the host may issue the next token. Otherwise, the host waits for the device to finish sending the remainder of its packet. Waiting 16 bit times guarantees two conditions:

- The first condition is to make sure that the device has finished sending its packet. This is guaranteed by a timeout interval (with no bus transitions) greater than the worst case six-bit time bit stuff interval.
- The second condition is that the transmitting device's bus turn-around timer must be guaranteed to expire.

Note that the timeout interval is transaction speed sensitive. For full-speed transactions, the host must wait 16 full-speed bit times; for low-speed transactions, it must wait 16 low-speed bit times.

If the host receives a data packet with a valid CRC, it assumes that the packet is complete and need not delay in issuing the next token.

8.7.4 Babble and Loss of Activity Recovery

The USB must be able to detect and recover from conditions which leave it waiting indefinitely for an EOP or which leave the bus in something other than the Idle state at the end of a frame.

- Loss of activity (LOA) is characterized by SOP followed by lack of bus activity (bus remains driven to a 'J' or 'K') and no EOP at the end of a frame.
- Babble is characterized by an SOP followed by the presence of bus activity past the end of a frame.

LOA and babble have the potential to either deadlock the bus or force out the beginning of the next frame. Neither condition is acceptable, and both must be prevented from occurring. As the USB component responsible for controlling connectivity, hubs are responsible for babble/LOA detection and recovery. All USB devices that fail to complete their transmission at the end of a frame are prevented from transmitting past a frame's end by having the nearest hub disable the port to which the offending device is attached. Details of the hub babble/LOA recovery mechanism appear in Section 11.8.1.

Chapter 9
USB Device Framework

A USB device may be divided into three layers:

- The bottom layer is a bus interface that transmits and receives packets.
- The middle layer handles routing data between the bus interface and various endpoints on the device. An endpoint is the ultimate consumer or provider of data. It may be thought of as a source or sink for data.
- The top layer is the functionality provided by the serial bus device; for instance, a mouse or ISDN interface.

This chapter describes the common attributes and operations of the middle layer of a USB device. These attributes and operations are used by the function-specific portions of the device to communicate through the bus interface and ultimately with the host.

9.1 USB Device States

A USB device has several possible states. Some of these states are visible to the USB and the host, while others are internal to the USB device. This section describes those states.

9.1.1 Visible Device States

This section describes USB device states that are externally visible (see Figure 9-1). Table 9-1 summarizes the visible device states.

Note: USB devices perform a reset operation in response to reset signaling on the upstream port. When reset signaling has completed, the USB device is reset.

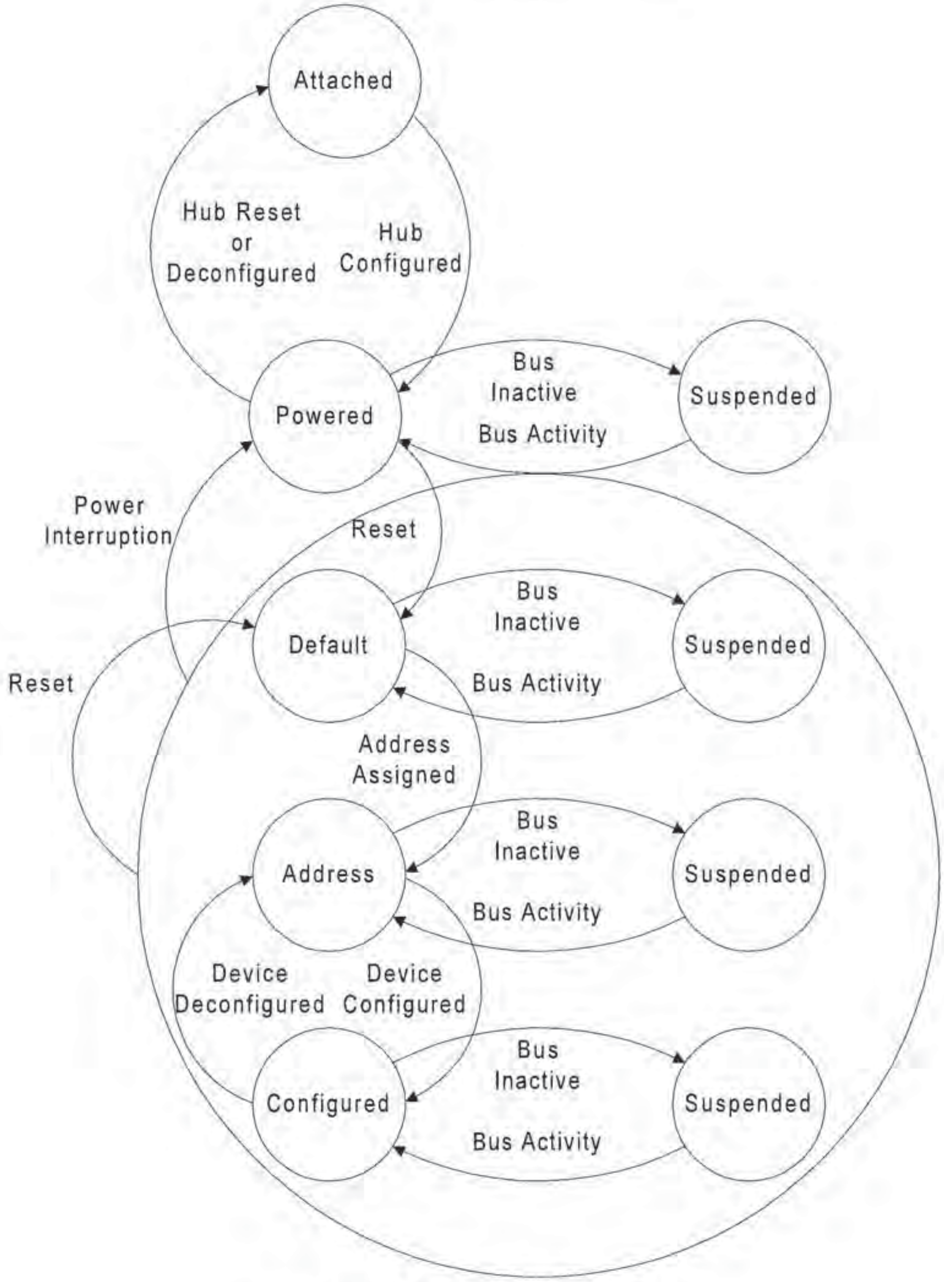


Figure 9-1. Device State Diagram

Table 9-1. Visible Device States

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

9.1.1.1 Attached

A USB device may be attached or detached from the USB. The state of a USB device when it is detached from the USB is not defined by this specification. This specification only addresses required operations and attributes once the device is attached.

9.1.1.2 Powered

USB devices may obtain power from an external source and/or from the USB through the hub to which they are attached. Externally powered USB devices are termed self-powered. Although self-powered devices may already be powered before they are attached to the USB, they are not considered to be in the Powered state until they are attached to the USB and VBUS is applied to the device.

A device may support both self-powered and bus-powered configurations. Some device configurations support either power source. Other device configurations may be available only if the device is self-powered. Devices report their power source capability through the configuration descriptor. The current power source is reported as part of a device's status. Devices may change their power source at any time; e.g., from self- to bus-powered. If a configuration is capable of supporting both power modes, the power maximum reported for that configuration is the maximum the device will draw from VBUS in either mode. The device must observe this maximum, regardless of its mode. If a configuration supports only one power mode and the power source of the device changes, the device will lose its current configuration and address and return to the Powered state. If a device is self-powered and its current configuration requires more than 100mA, then if the device switches to being bus-powered, it must return to the Address state. Self-powered hubs that use VBUS to power the Hub Controller are allowed to remain in the Configured state if local power is lost. Refer to Section 11.14 for details.

A hub port must be powered in order to detect port status changes, including attach and detach. Bus-powered hubs do not provide any downstream power until they are configured, at which point they will provide power as allowed by their configuration and power source. A USB device must be able to be addressed within a specified time period from when power is initially applied (refer to Chapter 7). After an attachment to a port has been detected, the host may enable the port, which will also reset the device attached to the port.

9.1.1.3 Default

After the device has been powered, it must not respond to any bus transactions until it has received a reset from the bus. After receiving a reset, the device is then addressable at the default address.

9.1.1.4 Address

All USB devices use the default address when initially powered or after the device has been reset. Each USB device is assigned a unique address by the host after attachment or after reset. A USB device maintains its assigned address while suspended.

A USB device responds to requests on its default pipe whether the device is currently assigned a unique address or is using the default address.

9.1.1.5 Configured

Before a USB device's function may be used, the device must be configured. From the device's perspective, configuration involves writing a non-zero value to the device configuration register. Configuring a device or changing an alternate setting causes all of the status and configuration values associated with endpoints in the affected interfaces to be set to their default values. This includes setting the data toggle of any endpoint using data toggles to the value DATA0.

9.1.1.6 Suspended

In order to conserve power, USB devices automatically enter the Suspended state when the device has observed no bus traffic for a specified period (refer to Chapter 7). When suspended, the USB device maintains any internal status, including its address and configuration.

All devices must suspend if bus activity has not been observed for the length of time specified in Chapter 7. Attached devices must be prepared to suspend at any time they are powered, whether they have been assigned a non-default address or are configured. Bus activity may cease due to the host entering a suspend mode of its own. In addition, a USB device shall also enter the Suspended state when the hub port it is attached to is disabled. This is referred to as selective suspend.

A USB device exits suspend mode when there is bus activity. A USB device may also request the host to exit suspend mode or selective suspend by using electrical signaling to indicate remote wakeup. The ability of a device to signal remote wakeup is optional. If a USB device is capable of remote wakeup signaling, the device must support the ability of the host to enable and disable this capability. When the device is reset, remote wakeup signaling must be disabled.

9.1.2 Bus Enumeration

When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached to a powered port, the following actions are taken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe (refer to Section 11.13.3 for more information). At this point, the USB device is in the Powered state and the port to which it is attached is disabled.
2. The host determines the exact nature of the change by querying the hub.
3. Now that the host knows the port to which the new device has been attached, the host then waits for at least 100 ms to allow completion of an insertion process and for power at the device to become stable. The host then issues a port enable and reset command to that port. Refer to Section 7.1.7.1 and Figure 7-19 for sequence of events and timings of connection through device reset.
4. The hub maintains the reset signal to that port for 10 ms (See Section 11.5.1.5). When the reset signal is released, the port has been enabled. The USB device is now in the Default state and can draw no more than 100mA from VBUS. All of its registers and state have been reset and it answers to the default address.
5. The host assigns a unique address to the USB device, moving the device to the Address state.
6. Before the USB device receives a unique address, its Default Control Pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
7. The host reads the configuration information from the device by reading each configuration zero to *n*-1, where *n* is the number of configurations. This process may take several milliseconds to complete.
8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the Configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of VBUS power described in its descriptor for the selected configuration. From the device's point of view it is now ready for use.

When the USB device is removed, the hub again sends a notification to the host. Detaching a device disables the port to which it had been attached. Upon receiving the detach notification, the host will update its local topological information.

9.2 Generic USB Device Operations

All USB devices support a common set of operations. This section describes those operations.

9.2.1 Dynamic Attachment and Removal

USB devices may be attached and removed at any time. The hub that provides the attachment point or port is responsible for reporting any change in the state of the port.

The host enables the hub port where the device is attached upon detection of an attachment, which also has the effect of resetting the device. A reset USB device has the following characteristics:

- Responds to the default USB address
- Is not configured
- Is not initially suspended.

When a device is removed from a hub port, the hub disables the port where the device was attached and notifies the host of the removal.

9.2.2 Address Assignment

When a USB device is attached, the host is responsible for assigning a unique address to the device. This is done after the device has been reset by the host and the hub port where the device is attached has been enabled.

9.2.3 Configuration

A USB device must be configured before its function(s) may be used. The host is responsible for configuring a USB device. The host typically requests configuration information from the USB device to determine the device's capabilities.

As part of the configuration process, the host sets the device configuration and, where necessary, selects the appropriate alternate settings for the interfaces.

Within a single configuration, a device may support multiple interfaces. An interface is a related set of endpoints that present a single feature or function of the device to the host. The protocol used to communicate with this related set of endpoints and the purpose of each endpoint within the interface may be specified as part of a device class or vendor-specific definition.

In addition, an interface within a configuration may have alternate settings that redefine the number or characteristics of the associated endpoints. If this is the case, the device shall support the GetInterface() and SetInterface() requests to report or select the current alternative setting for the specified interface.

Within each configuration, each interface descriptor contains fields that identify the interface number and the alternate setting. Interfaces are numbered from zero to one less than the number of concurrent interfaces supported by the configuration. Alternate settings range from zero to one less than the number of alternate settings for a specific interface. The default setting when a device is initially configured is alternate setting zero.

In support of adaptive device drivers that are capable of managing a related group of USB devices, the device and interface descriptors contain *Class*, *SubClass*, and *Protocol* fields. These fields are used to identify the function(s) provided by a USB device and the protocols used to communicate with the function(s) on the device. A class code is assigned to a group of related devices that has been characterized as a part of a USB Class Specification. A class of devices may be further subdivided into subclasses and within a class or subclass a protocol code may define how the Host Software communicates with the device.

Note: the assignment of class, subclass and protocol codes must be coordinated but is beyond the scope of this specification.

9.2.4 Data Transfer

Data may be transferred between a USB device endpoint and the host in one of four ways. Refer to Chapter 5 for the definition of the four types of transfers. An endpoint number may be used for different types of data transfers in different alternate settings. However, once an alternate setting is selected (including the default setting of an interface), a USB device endpoint uses only one data transfer method until a different alternate setting is selected.

9.2.5 Power Management

Power management on USB devices involves the issues described in the following sections.

9.2.5.1 Power Budgeting

USB bus power is a limited resource. During device enumeration, a host evaluates a device's power requirements. If the power requirements of a particular configuration exceed the power available to the device, Host software shall not select that configuration.

USB devices shall limit the power they consume from VBUS to one unit load or less until configured. Suspended devices, whether configured or not, shall limit their bus power consumption as defined in Chapter 7. Depending on the power capabilities of the port to which the device is attached, a USB device may be able to draw up to five unit loads from VBUS after configuration.

9.2.5.2 Remote Wakeup

Remote wakeup allows a suspended USB device to signal a host that may also be suspended. This notifies the host that it should resume from its suspended mode, if necessary, and service the external event that triggered the suspended USB device to signal the host. A USB device reports its ability to support remote wakeup in a configuration descriptor. If a device supports remote wakeup, it must also allow the capability to be enabled and disabled using the standard USB requests.

Remote wakeup is accomplished using electrical signaling described in Section 7.1.7.5.

9.2.6 Request Processing

With the exception of SetAddress() requests (see Section 9.4.6), a device may begin processing of a request as soon as the device returns the ACK following the Setup. The device is expected to "complete" processing of the request before it allows the Status stage to complete successfully. Some requests initiate operations that take many milliseconds to complete. For requests such as this, the device class is required to define a method other than Status stage completion to indicate that the operation has completed. For example, a reset on a hub port takes at least 10 ms to complete. The SetPortFeature(PORT_RESET) (see Chapter 11) request "completes" when the reset on the port is initiated. Completion of the reset operation is signaled when the port's status change is set to indicate that the port is now enabled. This technique prevents the host from having to constantly poll for a completion when it is known that the request will take a relatively long period of time.

9.2.6.1 Request Processing Timing

All devices are expected to handle requests in a timely manner. USB sets an upper limit of 5 seconds as the upper limit for any command to be processed. This limit is not applicable in all instances. The limitations are described in the following sections. It should be noted that the limitations given below are intended to encompass a wide range of implementations. If all devices in a USB system used the maximum allotted time for request processing the user experience would suffer. For this reason, implementations should strive to complete requests in times that are as short as possible.

9.2.6.2 Reset/Resume Recovery Time

After a port is reset or resumed, the USB System Software is expected to provide a "recovery" interval of 10 ms before the device attached to the port is expected to respond to data transfers. The device may ignore any data transfers during the recovery interval.

After the end of the recovery interval (measured from the end of the reset or the end of the EOP at the end of the resume signaling) the device must accept data transfers at any time.

9.2.6.3 Set Address Processing

After the reset/resume recovery interval, if a device receives a SetAddress() request, the device must be able to complete processing of the request and be able to successfully complete the Status stage of the request within 50 ms. In the case of the SetAddress() request, the Status stage successfully completes when the devices sends the zero-length Status packet or when the device sees the ACK in response to the Status stage data packet.

After successful completion of the Status stage, the device is allowed a SetAddress() recovery interval of 2 ms. At the end of this interval, the device must be able to accept Setup packets addressed to the new address. Also, at the end of the recovery interval the device must not respond to tokens sent to the old address (unless, of course, the old and new address is the same.)

9.2.6.4 Standard Device Requests

For standard device requests that require no Data stage, a device must be able to complete the request and be able to successfully complete the Status stage of the request within 50 ms of receipt of the request. This limitation applies to requests to the device, interface, or endpoint.

For standard device requests that require data stage transfer to the host, the device must be able to return the first data packet to the host within 500 ms of receipt of the request. For subsequent data packets, if any, the device must be able to return them within 500 ms of successful completion of the transmission of the previous packet. The device must then be able to successfully complete the status stage within 50 ms after returning the last data packet.

For standard device requests that require a data stage transfer to the device, the 5-second limit applies. This means that the device must be capable of accepting all data packets from the host and successfully completing the Status stage if the host provides the data at the maximum rate at which the device can accept it. Delays between packets introduced by the host add to the time allowed for the device to complete the request.

9.2.6.5 Class-specific Requests

Unless specifically exempted in the class document, all class-specific requests must meet the timing limitations for standard device requests. If a class document provides an exemption, the exemption may only be specified on a request-by-request basis.

A class document may require that a device respond more quickly than is specified in this section. Faster response may be required for standard and class-specific requests.

9.2.7 Request Error

When a request is received by a device that is not defined for the device, is inappropriate for the current setting of the device, or has values that are not compatible with the request, then a Request Error exists.

The device deals with the Request Error by returning a STALL PID in response to the next Data stage transaction or in the Status stage of the message. It is preferred that the STALL PID be returned at the next Data stage transaction, as this avoids unnecessary bus activity.

9.3 USB Device Requests

All USB devices respond to requests from the host on the device's Default Control Pipe. These requests are made using control transfers. The request and the request's parameters are sent to the device in the Setup packet. The host is responsible for establishing the values passed in the fields listed in Table 9-2. Every Setup packet has eight bytes.

Table 9-2. Format of Setup Data

Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bitmap	Characteristics of request: D7: Data transfer direction 0 = Host-to-device 1 = Device-to-host D6...5: Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4...0: Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4...31 = Reserved
1	*bRequest*	1	Value	Specific request (refer to Table 9-3)
2	*wValue*	2	Value	Word-sized field that varies according to request
4	*wIndex*	2	Index or Offset	Word-sized field that varies according to request; typically used to pass an index or offset
6	*wLength*	2	Count	Number of bytes to transfer if there is a Data stage

9.3.1 bmRequestType

This bitmapped field identifies the characteristics of the specific request. In particular, this field identifies the direction of data transfer in the second phase of the control transfer. The state of the *Direction* bit is ignored if the *wLength* field is zero, signifying there is no Data stage.

The USB Specification defines a series of standard requests that all devices must support. These are enumerated in Table 9-3. In addition, a device class may define additional requests. A device vendor may also define requests supported by the device.

Requests may be directed to the device, an interface on the device, or a specific endpoint on a device. This field also specifies the intended recipient of the request. When an interface or endpoint is specified, the *wIndex* field identifies the interface or endpoint.

9.3.2 bRequest

This field specifies the particular request. The *Type* bits in the *bmRequestType* field modify the meaning of this field. This specification defines values for the *bRequest* field only when the bits are reset to zero, indicating a standard request (refer to Table 9-3).

9.3.3 wValue

The contents of this field vary according to the request. It is used to pass a parameter to the device, specific to the request.

9.3.4 wIndex

The contents of this field vary according to the request. It is used to pass a parameter to the device, specific to the request.

The *wIndex* field is often used in requests to specify an endpoint or an interface. Figure 9-2 shows the format of *wIndex* when it is used to specify an endpoint.

D7	D6	D5	D4	D3	D2	D1	D0
Direction	Reserved (Reset to zero)			Endpoint Number			
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

Figure 9-2. wIndex Format when Specifying an Endpoint

The *Direction* bit is set to zero to indicate the OUT endpoint with the specified *Endpoint Number* and to one to indicate the IN endpoint. In the case of a control pipe, the request should have the *Direction* bit set to zero but the device may accept either value of the *Direction* bit.

Figure 9-3 shows the format of *wIndex* when it is used to specify an interface.

D7	D6	D5	D4	D3	D2	D1	D0
Interface Number							
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

Figure 9-3. wIndex Format when Specifying an Interface

9.3.5 wLength

This field specifies the length of the data transferred during the second phase of the control transfer. The direction of data transfer (host-to-device or device-to-host) is indicated by the *Direction* bit of the *bmRequestType* field. If this field is zero, there is no data transfer phase.

On an input request, a device must never return more data than is indicated by the *wLength* value; it may return less. On an output request, *wLength* will always indicate the exact amount of data to be sent by the host. Device behavior is undefined if the host should send more data than is specified in *wLength*.

9.4 Standard Device Requests

This section describes the standard device requests defined for all USB devices. Table 9-3 outlines the standard device requests, while Table 9-4 and Table 9-5 give the standard request codes and descriptor types, respectively.

USB devices must respond to standard device requests, whether the device has been assigned a non-default address or the device is currently configured.

Table 9-3. Standard Device Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Interface
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

Table 9-4. Standard Request Codes

bRequest	Value
GET_STATUS	0
CLEAR_FEATURE	1
Reserved for future use	2
SET_FEATURE	3
Reserved for future use	4
SET_ADDRESS	5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNCH_FRAME	12

Table 9-5. Descriptor Types

Descriptor Types	Value
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5

Feature selectors are used when enabling or setting features, such as remote wakeup, specific to a device, interface, or endpoint. The values for the feature selectors are given in Table 9-6.

Table 9-6. Standard Feature Selectors

Feature Selector	Recipient	Value
DEVICE_REMOTE_WAKEUP	Device	1
ENDPOINT_HALT	Endpoint	0

If an unsupported or invalid request is made to a USB device, the device responds by returning STALL in the Data or Status stage of the request. If the device detects the error in the Setup stage, it is preferred that the device returns STALL at the earlier of the Data or Status stage. Receipt of an unsupported or invalid request does NOT cause the optional *Halt* feature on the control pipe to be set. If for any reason, the device becomes unable to communicate via its Default Control Pipe due to an error condition, the device must be reset to clear the condition and restart the Default Control Pipe.

9.4.1 Clear Feature

This request is used to clear or disable a specific feature.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None

Feature selector values in *wValue* must be appropriate to the recipient. Only device feature selector values may be used when the recipient is a device, only interface feature selector values may be used when the recipient is an interface, and only endpoint feature selector values may be used when the recipient is an endpoint.

Refer to Table 9-6 for a definition of which feature selector values are defined for which recipients.

A ClearFeature() request that references a feature that cannot be cleared, that does not exist, or that references an interface or endpoint that does not exist will cause the device to respond with a Request Error.

If *wLength* is non-zero, then the device behavior is not specified.

Default state:	Device behavior when this request is received while the device is in the Default state is not specified.
Address state:	This request is valid when the device is in the Address state; references to interfaces or to endpoints other than endpoint zero shall cause the device to respond with a Request Error.
Configured state:	This request is valid when the device is in the Configured state.

9.4.2 Get Configuration

This request returns the current device configuration value.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value

If the returned value is zero, the device is not configured.

If *wValue*, *wIndex*, or *wLength* are not as specified above, then the device behavior is not specified.

Default state:	Device behavior when this request is received while the device is in the Default state is not specified.
Address state:	The value zero shall be returned.
Configured state:	The non-zero *bConfigurationValue* of the current configuration shall be returned.

9.4.3 Get Descriptor

This request returns the specified descriptor if the descriptor exists.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID (refer to Section 9.6.5)	Descriptor Length	Descriptor

The *wValue* field specifies the descriptor type in the high byte and the descriptor index in the low byte (refer to Table 9-5). The *wIndex* field specifies the Language ID for string descriptors or is reset to zero for other descriptors. The *wLength* field specifies the number of bytes to return. If the descriptor is longer than the *wLength* field, only the initial bytes of the descriptor are returned. If the descriptor is shorter than the *wLength* field, the device indicates the end of the control transfer by sending a short packet when further data is requested. A short packet is defined as a packet shorter than the maximum payload size or a NULL data packet (refer to Chapter 5).

The standard request to a device supports three types of descriptors: DEVICE, CONFIGURATION, and STRING. A request for a configuration descriptor returns the configuration descriptor, all interface descriptors, and endpoint descriptors for all of the interfaces in a single request. The first interface descriptor follows the configuration descriptor. The endpoint descriptors for the first interface follow the first interface descriptor. If there are additional interfaces, their interface descriptor and endpoint descriptors follow the first interface's endpoint descriptors. Class-specific and/or vendor-specific descriptors follow the standard descriptors they extend or modify.

All devices must provide a device descriptor and at least one configuration descriptor. If a device does not support a requested descriptor, it responds with a Request Error.

Default state:	This is a valid request when the device is in the Default state.
Address state:	This is a valid request when the device is in the Address state.

Configured state: This is a valid request when the device is in the Configured state.

9.4.4 Get Interface

This request returns the selected alternate setting for the specified interface.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Setting

Some USB devices have configurations with interfaces that have mutually exclusive settings. This request allows the host to determine the currently selected alternative setting.

If *wValue* or *wLength* are not as specified above, then the device behavior is not specified.

If the interface specified does not exist, then the device responds with a Request Error.

Default state: Device behavior when this request is received while the device is in the Default state is not specified.

Address state: A Request Error response is given by the device.

Configured state: This is a valid request when the device is in the Configured state.

9.4.5 Get Status

This request returns status for the specified recipient.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status

The *Recipient* bits of the *bmRequestType* field specify the desired recipient. The data returned is the current status of the specified recipient.

If *wValue* or *wLength* are not as specified above, or if *wIndex* is non-zero for a device status request, then the behavior of the device is not specified.

If an interface or an endpoint is specified that does not exist then the device responds with a Request Error.

Default state: Device behavior when this request is received while the device is in the Default state is not specified.

Address state: If an interface or an endpoint other than endpoint zero is specified, then the device responds with a Request Error.

Configured state: If an interface or endpoint that does not exist is specified, then the device responds with a Request Error.

A GetStatus() request to a device returns the information shown in Figure 9-4.

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)						Remote Wakeup	Self Powered
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

Figure 9-4. Information Returned by a GetStatus() Request to a Device

The *Self Powered* field indicates whether the device is currently self-powered. If D0 is reset to zero, the device is bus-powered. If D0 is set to one, the device is self-powered. The *Self Powered* field may not be changed by the SetFeature() or ClearFeature() requests.

The *Remote Wakeup* field indicates whether the device is currently enabled to request remote wakeup. The default mode for devices that support remote wakeup is disabled. If D1 is reset to zero, the ability of the device to signal remote wakeup is disabled. If D1 is set to one, the ability of the device to signal remote wakeup is enabled. The *Remote Wakeup* field can be modified by the SetFeature() and ClearFeature() requests using the DEVICE_REMOTE_WAKEUP feature selector. This field is reset to zero when the device is reset.

A GetStatus() request to an interface returns the information shown in Figure 9-5.

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

Figure 9-5. Information Returned by a GetStatus() Request to a Interface

A GetStatus() request to an endpoint returns the information shown in Figure 9-6.

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							Halt
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

Figure 9-6. Information Returned by a GetStatus() Request to an Endpoint

The *Halt* feature is required to be implemented for all interrupt and bulk endpoint types. If the endpoint is currently halted, then the *Halt* feature is set to one. Otherwise, the *Halt* feature is reset to zero. The *Halt* feature may optionally be set with the SetFeature(ENDPOINT_HALT) request. When set by the SetFeature() request, the endpoint exhibits the same stall behavior as if the field had been set by a hardware condition. If the condition causing a halt has been removed, clearing the *Halt* feature via a ClearFeature(ENDPOINT_HALT) request results in the endpoint no longer returning a STALL. For endpoints using data toggle, regardless of whether an endpoint has the *Halt* feature set, a ClearFeature(ENDPOINT_HALT) request always results in the data toggle being reinitialized to DATA0. The *Halt* feature is reset to zero after either a SetConfiguration() or SetInterface() request even if the requested configuration or interface is the same as the current configuration or interface.

It is neither required nor recommended that the *Halt* feature be implemented for the Default Control Pipe. However, devices may set the *Halt* feature of the Default Control Pipe in order to reflect a functional error condition. If the feature is set to one, the device will return STALL in the Data and Status stages of each standard request to the pipe except GetStatus(), SetFeature(), and ClearFeature() requests. The device need not return STALL for class-specific and vendor-specific requests.

9.4.6 Set Address

This request sets the device address for all future device accesses.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None

The *wValue* field specifies the device address to use for all subsequent accesses.

As noted elsewhere, requests actually may result in up to three stages. In the first stage, the Setup packet is sent to the device. In the optional second stage, data is transferred between the host and the device. In the final stage, status is transferred between the host and the device. The direction of data and status transfer depends on whether the host is sending data to the device or the device is sending data to the host. The Status stage transfer is always in the opposite direction of the Data stage. If there is no Data stage, the Status stage is from the device to the host.

Stages after the initial Setup packet assume the same device address as the Setup packet. The USB device does not change its device address until after the Status stage of this request is completed successfully. Note that this is a difference between this request and all other requests. For all other requests, the operation indicated must be completed before the Status stage.

If the specified device address is greater than 127, or if *wIndex* or *wLength* are non-zero, then the behavior of the device is not specified.

Device response to SetAddress() with a value of 0 is undefined.

Default state:	If the address specified is non-zero, then the device shall enter the Address state; otherwise, the device remains in the Default state (this is not an error condition).
Address state:	If the address specified is zero, then the device shall enter the Default state; otherwise, the device remains in the Address state but uses the newly-specified address.
Configured state:	Device behavior when this request is received while the device is in the Configured state is not specified.

9.4.7 Set Configuration

This request sets the device configuration.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None

The lower byte of the *wValue* field specifies the desired configuration. This configuration value must be zero or match a configuration value from a configuration descriptor. If the configuration value is zero, the device is placed in its Address state. The upper byte of the *wValue* field is reserved.

If *wIndex*, *wLength*, or the upper byte of *wValue* is non-zero, then the behavior of this request is not specified.

Default state:	Device behavior when this request is received while the device is in the Default state is not specified.
Address state:	If the specified configuration value is zero, then the device remains in the Address state. If the specified configuration value matches the configuration value from a configuration descriptor, then that configuration is selected and the device enters the Configured state. Otherwise, the device responds with a Request Error.
Configured state:	If the specified configuration value is zero, then the device enters the Address state. If the specified configuration value matches the configuration value from a configuration descriptor, then that configuration is selected and the device remains in the Configured state. Otherwise, the device responds with aRequest Error.

9.4.8 Set Descriptor

This request may be used to update existing descriptors or new descriptors may be added.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Language ID (refer to Section 9.6.5) or zero	Descriptor Length	Descriptor

The *wValue* field specifies the descriptor type in the high byte and the descriptor index in the low byte (refer to Table 9-5). The *wIndex* field specifies the Language ID for string descriptors or is reset to zero for other descriptors. The *wLength* field specifies the number of bytes to transfer from the host to the device.

If this request is not supported then the device will respond with a Request Error.

Default state:	Device behavior when this request is received while the device is in the Default state is not specified.
Address state:	If supported, this is a valid request when the device is in the Address state.
Configured state:	If supported, this is a valid request when the device is in the Configured state.

9.4.9 Set Feature

This request is used to set or enable a specific feature.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None

Feature selector values in *wValue* must be appropriate to the recipient. Only device feature selector values may be used when the recipient is a device; only interface feature selector values may be used when the recipient is an interface, and only endpoint feature selector values may be used when the recipient is an endpoint.

Refer to Table 9-6 for a definition of which feature selector values are defined for which recipients. A SetFeature() request that references a feature that cannot be set or that does not exist causes a STALL to be returned in the Status stage of the request.

If *wLength* is non-zero, then the behavior of the device is not specified.

If an endpoint or interface is specified that does not exist, then the device responds with a Request Error.

Default state:	Device behavior when this request is received while the device is in the Default state is not specified.
Address state:	If an interface or an endpoint other than endpoint zero is specified, then the device responds with a Request Error.
Configured state:	This is a valid request when the device is in the Configured state.

9.4.10 Set Interface

This request allows the host to select an alternate setting for the specified interface.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000001B	SET_INTERFACE	Alternative Setting	Interface	Zero	None

Some USB devices have configurations with interfaces that have mutually exclusive settings. This request allows the host to select the desired alternate setting. If a device only supports a default setting for the specified interface, then a STALL may be returned in the Status stage of the request.

If the interface or the alternative setting does not exist, then the device responds with a Request Error. If *wLength* is non-zero, then the behavior of the device is not specified.

Default state:	Device behavior when this request is received while the device is in the Default state is not specified.
Address state:	The device shall respond with a Request Error.
Configured state:	This is a valid request when the device is in the Configured state.

9.4.11 Synch Frame

This request is used to set and then report an endpoint's synchronization frame.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

When an endpoint supports isochronous transfers, the endpoint may also require per-frame transfers to vary in size according to a specific pattern. The host and the endpoint must agree on which frame the repeating pattern begins. The number of the frame in which the pattern began is returned to the host. This frame number is the one conveyed to the endpoint by the last SOF prior to the first frame of the pattern. Alternatively, the device may use this request to restart the pattern. In this case, the device would save the frame number in each SOF and return this value in the Data stage of this transfer and restart the pattern on each IN of the Data stage.

This value is only used for isochronous data transfers using implicit pattern synchronization. If *wValue* is non-zero or *wLength* is not two, then the behavior of the device is not specified.

If the specified endpoint does not support this request, then the device will respond with a Request Error.

Default state:	Device behavior when this request is received while the device is in the Default state is not specified.
Address state:	The device shall respond with a Request Error.
Configured state:	This is a valid request when the device is in the Configured state.

9.5 Descriptors

USB devices report their attributes using descriptors. A descriptor is a data structure with a defined format. Each descriptor begins with a byte-wide field that contains the total number of bytes in the descriptor followed by a byte-wide field that identifies the descriptor type.

Using descriptors allows concise storage of the attributes of individual configurations because each configuration may reuse descriptors or portions of descriptors from other configurations that have the same characteristics. In this manner, the descriptors resemble individual data records in a relational database.

Where appropriate, descriptors contain references to string descriptors that provide displayable information describing a descriptor in human-readable form. The inclusion of string descriptors is optional. However, the reference fields within descriptors are mandatory. If a device does not support string descriptors, string reference fields must be reset to zero to indicate no string descriptor is available.

If a descriptor returns with a value in its length field that is less than defined by this specification, the descriptor is invalid and should be rejected by the host. If the descriptor returns with a value in its length field that is greater than defined by this specification, the extra bytes are ignored by the host, but the next descriptor is located using the length returned rather than the length expected.

A device may return class- or vendor-specific descriptors in two ways.

1. If the class or vendor specific descriptors use the same format as standard descriptors (e.g. start with a length byte and followed by a type byte), they may be returned interleaved with standard descriptors in the configuration information returned by a GetDescriptor(Configuration) request. In this case, the class or vendor-specific descriptors typically follow a related standard descriptor they modify or extend.
2. If the class or vendor specifric descriptors are independentof configuration infomrationor use a non-standard format, a GetDescriptor() request specifying the class or vendor specific descriptor type and index may be used to retrieve the descriptor from the device. A class or vendor specification will define the appropriate way to retrieve these descriptors.

9.6 Standard USB Descriptor Definitions

The standard descriptors defined in this specification may only be modified or extended by revision of the Universal Serial Bus Specification.

Note*:* An extension to the USB 1.0 standard endpoint descriptor has been published in Device Class Specification for Audio Devices Revision 1.0. This is the only extension defined outside USB Specification that is allowed. Future revisions of the USB Specification that extend the standard endpoint descriptor will do so as to not conflict with the extension defined in the Audio Device Class Specification Revision 1.0.

9.6.1 Device

A device descriptor describes general information about a USB device. It includes information that applies globally to the device and all of the device's configurations. A USB device has only one device descriptor.

All USB devices have a Default Control Pipe. The maximum packet size of a device's Default Control Pipe is described in the device descriptor. Endpoints specific to a configuration and its interface(s) are described in the configuration descriptor. A configuration and its interface(s) do not include an endpoint descriptor for the Default Control Pipe. Other than the maximum packet size, the characteristics of the Default Control Pipe are defined by this specification and are the same for all USB devices.

The *bNumConfigurations* field identifies the number of configurations the device supports. Table 9-7 shows the standard device descriptor.

Table 9-7. Standard Device Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	DEVICE Descriptor Type
2	*bcdUSB*	2	BCD	USB Specification Release Number in Binary-Coded Decimal (i.e., 2.10 is 210H). This field identifies the release of the USB Specification with which the device and its descriptors are compliant.
4	*bDeviceClass*	1	Class	Class code (assigned by the USB). If this field is reset to zero, each interface within a configuration specifies its own class information and the various interfaces operate independently. If this field is set to a value between 1 and FEH, the device supports different class specifications on different interfaces and the interfaces may not operate independently. This value identifies the class definition used for the aggregate interfaces. (For example, a CD-ROM device with audio and digital data interfaces that require transport control to eject CDs or start them spinning.) If this field is set to FFH, the device class is vendor-specific.
5	*bDeviceSubClass*	1	SubClass	Subclass code (assigned by the USB). These codes are qualified by the value of the *bDeviceClass* field. If the *bDeviceClass* field is reset to zero, this field must also be reset to zero. If the *bDeviceClass* field is not set to FFH, all values are reserved for assignment by the USB.

Table 9-7. Standard Device Descriptor (Continued)

Offset	Field	Size	Value	Description
6	*bDeviceProtocol*	1	Protocol	Protocol code (assigned by the USB). These codes are qualified by the value of the *bDeviceClass* and the *bDeviceSubClass* fields. If a device supports class-specific protocols on a device basis as opposed to an interface basis, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to zero, the device does not use class-specific protocols on a device basis. However, it may use class-specific protocols on an interface basis. If this field is set to FFH, the device uses a vendor-specific protocol on a device basis.
7	*bMaxPacketSize0*	1	Number	Maximum packet size for endpoint zero (only 8, 16, 32, or 64 are valid)
8	*idVendor*	2	ID	Vendor ID (assigned by the USB)
10	*idProduct*	2	ID	Product ID (assigned by the manufacturer)
12	*bcdDevice*	2	BCD	Device release number in binary-coded decimal
14	*iManufacturer*	1	Index	Index of string descriptor describing manufacturer
15	*iProduct*	1	Index	Index of string descriptor describing product
16	*iSerialNumber*	1	Index	Index of string descriptor describing the device's serial number
17	*bNumConfigurations*	1	Number	Number of possible configurations

9.6.2 Configuration

The configuration descriptor describes information about a specific device configuration. The descriptor contains a *bConfigurationValue* field with a value that, when used as a parameter to the SetConfiguration() request, causes the device to assume the described configuration.

The descriptor describes the number of interfaces provided by the configuration. Each interface may operate independently. For example, an ISDN device might be configured with two interfaces, each providing 64kB/s bi-directional channels that have separate data sources or sinks on the host. Another configuration might present the ISDN device as a single interface, bonding the two channels into one 128kB/s bi-directional channel.

When the host requests the configuration descriptor, all related interface and endpoint descriptors are returned (refer to Section 9.4.2).

A USB device has one or more configuration descriptors. Each configuration has one or more interfaces and each interface has zero or more endpoints. An endpoint is not shared among interfaces within a single configuration unless the endpoint is used by alternate settings of the same interface. Endpoints may be shared among interfaces that are part of different configurations without this restriction.

Once configured, devices may support limited adjustments to the configuration. If a particular interface has alternate settings, an alternate may be selected after configuration. Table 9-8 shows the standard configuration descriptor.

Table 9-8. Standard Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	CONFIGURATION Descriptor Type
2	*wTotalLength*	2	Number	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class- or vendor-specific) returned for this configuration.
4	*bNumInterfaces*	1	Number	Number of interfaces supported by this configuration
5	*bConfigurationValue*	1	Number	Value to use as an argument to the SetConfiguration() request to select this configuration
6	*iConfiguration*	1	Index	Index of string descriptor describing this configuration

Table 9-8. Standard Configuration Descriptor (Continued)

Offset	Field	Size	Value	Description
7	*bmAttributes*	1	Bitmap	Configuration characteristics D7: Reserved (set to one) D6: Self-powered D5: Remote Wakeup D4...0: Reserved (reset to zero) D7 is reserved and must be set to one for historical reasons. A device configuration that uses power from the bus and a local source reports a non-zero value in *MaxPower* to indicate the amount of bus power required and sets D6. The actual power source at runtime may be determined using the GetStatus(DEVICE) request (see Section 9.4.5). If a device configuration supports remote wakeup, D5 is set to one.
8	*MaxPower*	1	mA	Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2mA units (i.e., 50 = 100mA). Note: a device configuration reports whether the configuration is bus-powered or self-powered. Device status reports whether the device is currently self-powered. If a device is disconnected from its external power source, it updates device status to indicate that it is no longer self-powered. A device may not increase its power draw from the bus, when it loses its external power source, beyond the amount reported by its configuration. If a device can continue to operate when disconnected from its external power source, it continues to do so. If the device cannot continue to operate, it fails operations it can no longer support. The USB System Software may determine the cause of the failure by checking the status and noting the loss of the device's power source.

9.6.3 Interface

This descriptor describes a specific interface within a configuration. A configuration provides one or more interfaces, each with zero or more endpoint descriptors describing a unique set of endpoints within the configuration. When a configuration supports more than one interface, the endpoints for a particular interface follow the interface descriptor in the data returned by the GetConfiguration() request. An interface descriptor is always returned as part of a configuration descriptor. Interface descriptors cannot be directly accessed with a GetDescriptor() or SetDescriptor() request.

An interface may include alternate settings that allow the endpoints and/or their characteristics to be varied after the device has been configured. The default setting for an interface is always alternate setting zero. The SetInterface() request is used to select an alternate setting or to return to the default setting. The GetInterface() request returns the selected alternate setting.

Alternate settings allow a portion of the device configuration to be varied while other interfaces remain in operation. If a configuration has alternate settings for one or more of its interfaces, a separate interface descriptor and its associated endpoints are included for each setting.

If a device configuration supported a single interface with two alternate settings, the configuration descriptor would be followed by an interface descriptor with the *bInterfaceNumber* and *bAlternateSetting* fields set to zero and then the endpoint descriptors for that setting, followed by another interface descriptor and its associated endpoint descriptors. The second interface descriptor's *bInterfaceNumber* field would also be set to zero, but the *bAlternateSetting* field of the second interface descriptor would be set to one.

If an interface uses only endpoint zero, no endpoint descriptors follow the interface descriptor and the interface identifies a request interface that uses the default pipe attached to endpoint zero. In this case, the *bNumEndpoints* field shall be set to zero.

An interface descriptor never includes endpoint zero in the number of endpoints. Table 9-9 shows the standard interface descriptor.

Table 9-9. Standard Interface Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	INTERFACE Descriptor Type
2	*bInterfaceNumber*	1	Number	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	*bAlternateSetting*	1	Number	Value used to select alternate setting for the interface identified in the prior field
4	*bNumEndpoints*	1	Number	Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses the Default Control Pipe.
5	*bInterfaceClass*	1	Class	Class code (assigned by the USB). A value of zero is reserved for future standardization. If this field is set to FFH, the interface class is vendor-specific. All other values are reserved for assignment by the USB.
6	*bInterfaceSubClass*	1	SubClass	Subclass code (assigned by the USB). These codes are qualified by the value of the *bInterfaceClass* field. If the *bInterfaceClass* field is reset to zero, this field must also be reset to zero. If the *bInterfaceClass* field is not set to FFH, all values are reserved for assignment by the USB.

Table 9-9. Standard Interface Descriptor (Continued)

Offset	Field	Size	Value	Description
7	*bInterfaceProtocol*	1	Protocol	Protocol code (assigned by the USB). These codes are qualified by the value of the *bInterfaceClass* and the *bInterfaceSubClass* fields. If an interface supports class-specific requests, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to zero, the device does not use a class-specific protocol on this interface. If this field is set to FFH, the device uses a vendor-specific protocol for this interface.
8	*iInterface*	1	Index	Index of string descriptor describing this interface

9.6.4 Endpoint

Each endpoint used for an interface has its own descriptor. This descriptor contains the information required by the host to determine the bandwidth requirements of each endpoint. An endpoint descriptor is always returned as part of the configuration information returned by a GetDescriptor(Configuration) request. An endpoint descriptor cannot be directly accessed with a GetDescriptor() or SetDescriptor() request. There is never an endpoint descriptor for endpoint zero. Table 9-10 shows the standard endpoint descriptor.

Table 9-10. Standard Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 3...0: The endpoint number Bit 6...4: Reserved, reset to zero Bit 7: Direction, ignored for control endpoints 0 = OUT endpoint 1 = IN endpoint

Table 9-10. Standard Endpoint Descriptor (Continued)

Offset	Field	Size	Value	Description
3	*bmAttributes*	1	Bitmap	This field describes the endpoint's attributes when it is configured using the *bConfigurationValue*. Bit 1..0: Transfer Type 00 = Control 01 = Isochronous 10 = Bulk 11 = Interrupt All other bits are reserved.
4	*wMaxPacketSize*	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. For isochronous endpoints, this value is used to reserve the bus time in the schedule, required for the per-frame data payloads. The pipe may, on an ongoing basis, actually use less bandwidth than that reserved. The device reports, if necessary, the actual bandwidth used via its normal, non-USB defined mechanisms. For interrupt, bulk, and control endpoints, smaller data payloads may be sent, but will terminate the transfer and may or may not require intervention to restart. Refer to Chapter 5 for more information.
6	*bInterval*	1	Number	Interval for polling endpoint for data transfers. Expressed in milliseconds. This field is ignored for bulk and control endpoints. For isochronous endpoints this field must be set to 1. For interrupt endpoints, this field may range from 1 to 255.

9.6.5 String

String descriptors are optional. As noted previously, if a device does not support string descriptors, all references to string descriptors within device, configuration, and interface descriptors must be reset to zero.

String descriptors use UNICODE encodings as defined by *The Unicode Standard, Worldwide Character Encoding, Version 1.0, Volumes 1 and 2,* The Unicode Consortium, Addison-Wesley Publishing Company, Reading, Massachusetts. The strings in a USB device may support multiple languages. When requesting a string descriptor, the requester specifies the desired language using a sixteen-bit language ID (LANGID) defined by Microsoft for Windows as described in *Developing International Software for Windows 95 and Windows NT,* Nadine Kano, Microsoft Press, Redmond, Washington. String index zero for all languages returns a string descriptor that contains an array of two-byte LANGID codes supported by the device. Table 9-11 shows the LANGID code array. A USB device may omit all string descriptors. USB devices that omit all string descriptors shall not return an array of LANGID codes.

The array of LANGID codes is not NULL-terminated. The size of the array (in bytes) is computed by subtracting two from the value of the first byte of the descriptor.

Table 9-11. Codes Representing Languages Supported by the Device

Offset	Field	Size	Value	Description
0	*bLength*	1	N+2	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	STRING Descriptor Type
2	*wLANGID[0]*	2	Number	LANGID code zero
...	...	...	...	...
N	*wLANGID[x]*	2	Number	LANGID code *x*

The UNICODE string descriptor (shown in Table 9-12) is not NULL-terminated. The string length is computed by subtracting two from the value of the first byte of the descriptor.

Table 9-12. UNICODE String Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	STRING Descriptor Type
2	*bString*	N	Number	UNICODE encoded string

9.7 Device Class Definitions

All devices must support the requests and descriptor definitions described in this chapter. Most devices provide additional requests and, possibly, descriptors for device-specific extensions. In addition, devices may provide extended services that are common to a group of devices. In order to define a class of devices, the following information must be provided to completely define the appearance and behavior of the device class.

9.7.1 Descriptors

If the class requires any specific definition of the standard descriptors, the class definition must include those requirements as part of the class definition. In addition, if the class defines a standard extended set of descriptors, they must also be fully defined in the class definition. Any extended descriptor definitions should follow the approach used for standard descriptors; for example, all descriptors should begin with a length field.

9.7.2 Interface(s) and Endpoint Usage

When a class of devices is standardized, the interfaces used by the devices, including how endpoints are used, must be included in the device class definition. Devices may further extend a class definition with proprietary features as long as they meet the base definition of the class.

9.7.3 Requests

All of the requests specific to the class must be defined.

Chapter 10
USB Host: Hardware and Software

The USB interconnect supports data traffic between a host and a USB device. This chapter describes the host interfaces necessary to facilitate USB communication between a software client, resident on the host, and a function implemented on a device. The implementation described in this chapter is not required. This implementation is provided as an example to illustrate the host system behavior expected by a USB device. A host system may provide a different host software implementation as long as a USB device experiences the shame host behavior.

10.1 Overview of the USB Host

10.1.1 Overview

The basic flow and interrelationships of the USB communications model are shown in Figure 10-1

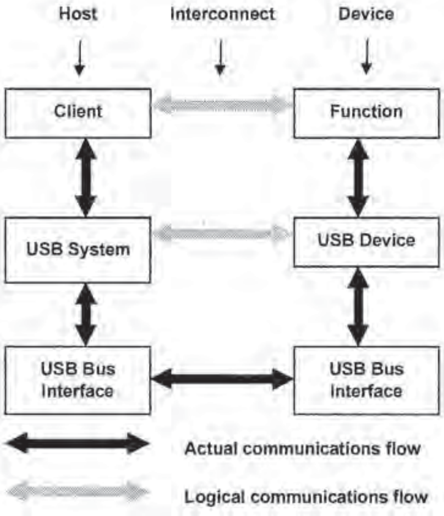


Figure 10-1. Interlayer Communications Model

The host and the device are divided into the distinct layers depicted in Figure 10-1. Vertical arrows indicate the actual communication on the host. The corresponding interfaces on the device are implementation-specific. All communications between the host and device ultimately occur on the physical USB wire. However, there are logical host-device interfaces between each horizontal layer. These communications, between client software resident on the host and the function provided by the device, are typified by a contract based on the needs of the application currently using the device and the capabilities provided by the device.

This client-function interaction creates the requirements for all of the underlying layers and their interfaces.

This chapter describes this model from the point of view of the host and its layers. Figure 10-2 describes, based on the overall view introduced in Chapter 5, the host's view of its communication with the device.

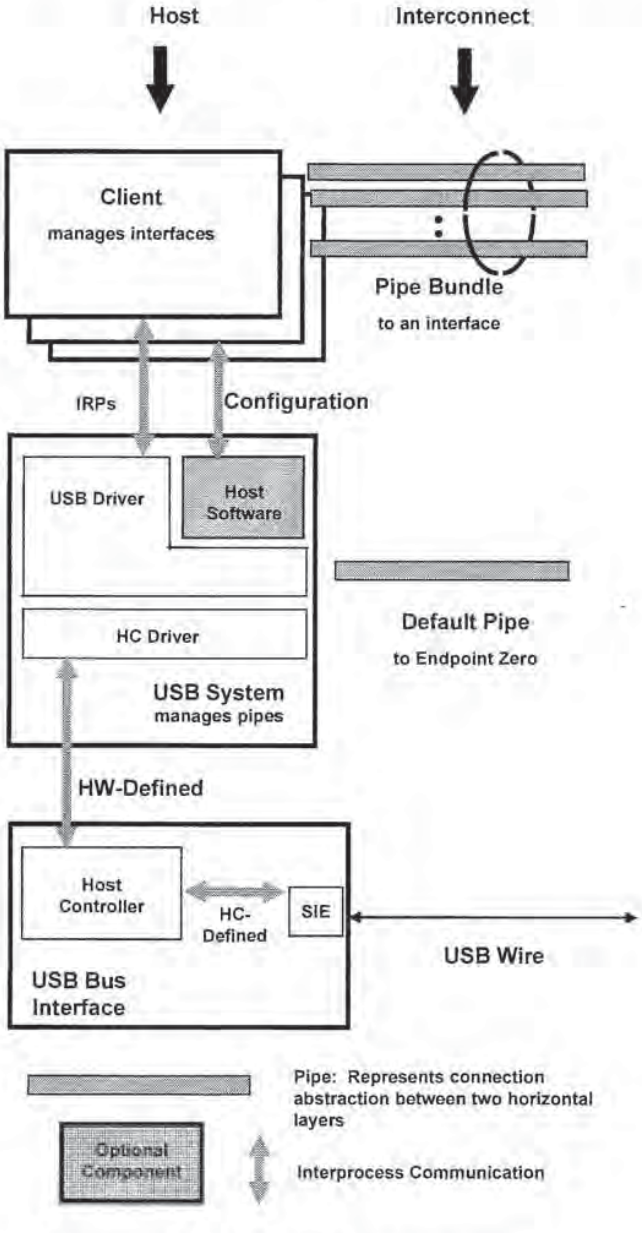


Figure 10-2. Host Communications

There is only one host for each USB. The major layers of a host consist of the following:

- USB bus interface
- USB System
- Client.

The USB bus interface handles interactions for the electrical and protocol layers (refer to Chapter 7 and Chapter 8). From the interconnect point of view, a similar USB bus interface is provided by both the USB device and the host, as exemplified by the Serial Interface Engine (SIE). On the host, however, the USB bus interface has additional responsibilities due to the unique role of the host on the USB and is implemented as the Host Controller. The Host Controller has an integrated root hub providing attachment points to the USB wire.

The USB System uses the Host Controller to manage data transfers between the host and USB devices. The interface between the USB System and the Host Controller is dependent on the hardware definition of the Host Controller. The USB System, in concert with the Host Controller, performs the translation between the client's view of data transfers and the USB transactions appearing on the interconnect. This includes the addition of any USB feature support such as protocol wrappers. The USB System is also responsible for managing USB resources, such as bandwidth and bus power, so that client access to the USB is possible.

The USB System has three basic components:

- Host Controller Driver
- USB Driver
- Host Software.

The Host Controller Driver (HCD) exists to more easily map the various Host Controller implementations into the USB System, such that a client can interact with its device without knowing to which Host Controller the device is connected. The USB Driver (USBD) provides the basic host interface (USBDI) for clients to USB devices. The interface between the HCD and the USBD is known as the Host Controller Driver Interface (HCDI). This interface is never available directly to clients and thus is not defined by the USB Specification. A particular HCDI is, however, defined by each operating system that supports various Host Controller implementations.

The USBD provides data transfer mechanisms in the form of I/O Request Packets (IRPs), which consist of a request to transport data across a specific pipe. In addition to providing data transfer mechanisms, the USBD is responsible for presenting to its clients an abstraction of a USB device that can be manipulated for configuration and state management. As part of this abstraction, the USBD owns the default pipe (see Chapter 5 and Chapter 9) through which all USB devices are accessed for the purposes of standard USB control. This default pipe represents a logical communication between the USBD and the abstraction of a USB device as shown in Figure 10-2.

In some operating systems, additional non-USB System Software is available that provides configuration and loading mechanisms to device drivers. In such operating systems, the device driver shall use the provided interfaces instead of directly accessing the USBDI mechanisms.

The client layer describes all the software entities that are responsible for directly interacting with USB devices. When each device is attached to the system, these clients might interact directly with the peripheral hardware. The shared characteristics of the USB place USB System Software between the client and its device; that is, a client cannot directly access the device's hardware.

Overall, the host layers provide the following capabilities:

- Detecting the attachment and removal of USB devices
- Managing USB standard control flow between the host and USB devices
- Managing data flow between the host and USB devices
- Collecting status and activity statistics
- Controlling the electrical interface between the Host Controller and USB devices, including the provision of a limited amount of power.

The following sections describe these responsibilities and the requirements placed on the USBDI in greater detail. The actual interfaces used for a specific combination of host platform and operating system are described in the appropriate operating system environment guide.

All hubs (see Chapter 11) report internal status changes and their port change status via the status change pipe. This includes a notification of when a USB device is attached to or removed from one of their ports. A USBD client generically known as the hub driver receives these notifications as owner of the hub's Status Change pipe. For device attachments, the hub driver then initiates the device configuration process. In some systems, this hub driver is a part of the host software provided by the operating system for managing devices.

10.1.2 Control Mechanisms

Control information may be passed between the host and a USB device using in-band or out-of-band signaling. In-band signaling mixes control information with data in a pipe outside the awareness of the host. Out-of-band signaling places control information in a separate pipe.

There is a message pipe called the default pipe for each attached USB device. This logical association between a host and a USB device is used for USB standard control flow such as device enumeration and configuration. The default pipe provides a standard interface to all USB devices. The default pipe may also be used for device-specific communications, as mediated by the USBD, which owns the default pipes of all of the USB devices.

A particular USB device may allow the use of additional message pipes to transfer device-specific control information. These pipes use the same communications protocol as the default pipe, but the information transferred is specific to the USB device and is not standardized by the USB Specification.

The USBD supports the sharing of the default pipe, which it owns and uses, with its clients. It also provides access to any other control pipes associated with the device.

10.1.3 Data Flow

The Host Controller is responsible for transferring streams of data between the host and USB devices. These data transfers are treated as a continuous stream of bytes. The USB supports four basic types of data transfers:

- Control transfers
- Isochronous transfers
- Interrupt transfers
- Bulk transfers.

For additional information on transfer types, refer to Chapter 5.

Each device presents one or more interfaces that a client may use to communicate with the device. Each interface is composed of zero or more pipes that individually transfer data between the client and a particular endpoint on the device. The USBD establishes interfaces and pipes at the explicit request of the Host Software. The Host Controller provides service based on parameters provided by the Host Software when the configuration request is made.

A pipe has several characteristics based on the delivery requirements of the data to be transferred. Examples of these characteristics include the following:

- the rate at which data needs to be transferred
- whether data is provided at a steady rate or sporadically
- how long data may be delayed before delivery
- whether the loss of data being transferred is catastrophic.

A USB device endpoint describes the characteristics required for a specific pipe. Endpoints are described as part of a USB device's characterization information. For additional details, refer to Chapter 9.

10.1.4 Collecting Status and Activity Statistics

As a common communicant for all control and data transfers between the host and USB devices, the USB System and the Host Controller are well-positioned to track status and activity information. Such information is provided upon request to the Host Software, allowing that software to manage status and activity information. This specification does not identify any specific information that should be tracked or require any particular format for reporting activity and status information.

10.1.5 Electrical Interface Considerations

The host provides power to USB devices attached to the root hub. The amount of power provided by a port is specified in Chapter 7.

10.2 Host Controller Requirements

In all implementations, Host Controllers perform the same basic duties with regard to the USB and its attached devices. These basic duties are described below.

The Host Controller has requirements from both the host and the USB. The following is a brief overview of the functionality provided. Each capability is discussed in detail in subsequent sections.

State Handling	As a component of the host, the Host Controller reports and manages its states.
Serializer/Deserializer	For data transmitted from the host, the Host Controller converts protocol and data information from its native format to a bit stream transmitted on the USB. For data being received into the host, the reverse operation is performed.
Frame Generation	The Host Controller produces SOF tokens at a period of 1ms.
Data Processing	The Host Controller processes requests for data transmission to and from the host.
Protocol Engine	The Host Controller supports the protocol specified by the USB.
Transmission Error Handling	All Host Controllers exhibit the same behavior when detecting and reacting to the defined error categories.
Remote Wakeup	All host controlers must have the ability to place the bus into the Suspended state and to respond to bus wakeup events.
Root Hub	The root hub provides standard hub function to link the Host Controller to one or more USB ports.

Host System Interface	Provides a high-speed data path between the Host Controller and host system.

The following sections present a more detailed discussion of the required capabilities of the Host Controller.

10.2.1 State Handling

The Host Controller has a series of states that the USB System manages. Additionally, the Host Controller provides the interface to the following two areas of USB-relevant state:

- State change propagation
- Root hub.

The root hub presents to the hub driver the same standard states as other USB devices. The Host Controller supports these states and their transitions for the hub. For detailed discussions of USB states, including their interrelations and transitions, refer to Chapter 9.

The overall state of the Host Controller is inextricably linked with that of the root hub and of the overall USB. Any Host Controller state changes that are visible to attached devices must be reflected in the corresponding device state change information such that the resulting Host Controller and device states are consistent.

USB devices request a wakeup through the use of resume signaling (refer to Chapter 7), devices to return to their configured state. The Host Controller itself may cause a resume event through the same signaling method. The Host Controller must notify the rest of the host of a resume event through a mechanism or mechanisms specific to that system's implementation.

10.2.2 Serializer/Deserializer

The actual transmission of data across the physical USB takes places as a serial bit stream. A Serial Interface Engine (SIE), whether implemented as part of the host or a USB device, handles the serialization and deserialization of USB transmissions. On the host, this SIE is part of the Host Controller.

10.2.3 Frame Generation

It is the Host Controller's responsibility to partition USB time into 1ms quantities called "frames." Frames are created by the Host Controller through issuing Start-of-Frame (SOF) tokens at 1.00ms intervals as shown in Figure 10-3. The SOF token is the first transmission in the frame period. After issuing a SOF token, the Host Controller is free to transmit other transactions for the remainder of the frame period. When the Host Controller is in its normal operating state, SOF tokens must be continuously generated at the 1ms periodic rate, regardless of the other bus activity or lack thereof. If the Host Controller enters a state where it is not providing power on the bus , it must not generate SOFs. When the Host Controller is not generating SOFs, it may enter a power-reduced state.

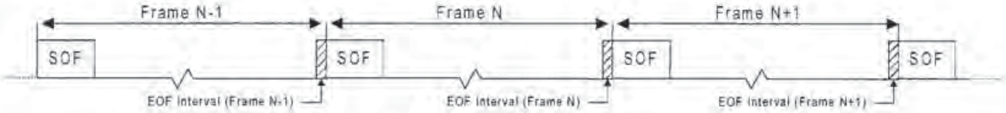


Figure 10-3. Frame Creation

The SOF token holds the highest priority access to the bus. Babble circuitry in hubs electrically isolates any active transmitters during the End-of-Frame (EOF) interval, providing an idle bus for the SOF transmission.

The Host Controller must allow the length of the USB frame to be adjusted by ±1 bit time (refer to Section 10.5.3.2.4). The Host Controller maintains the current frame number that may be read by the USB System.

The following apply to the current frame number:

- Used to uniquely identify one frame from another
- Incremented at the end of every frame period
- Valid through the subsequent frame.

The host transmits the lower 11 bits of the current frame number in each SOF token transmission. When requested from the Host Controller, the current frame number is the frame number in existence at the time the request was fulfilled. The current frame number as returned by the host (Host Controller or HCD) is at least 32 bits, although the Host Controller itself is not required to maintain more than 11 bits.

The Host Controller shall cease transmission during the EOF interval. When the EOF interval begins, any transactions scheduled specifically for the frame that has just passed are retired. If the Host Controller is executing a transaction at the time the EOF interval is encountered, the Host Controller terminates the transaction.

10.2.4 Data Processing

The Host Controller is responsible for receiving data from the USB System and sending it to the USB and for receiving data from the USB and sending it to the USB System. The particular format used for the data communications between the USB System and the Host Controller is implementation specific, within the rules for transfer behavior described in Chapter 5.

10.2.5 Protocol Engine

The Host Controller manages the USB protocol level interface. It inserts the appropriate protocol information for outgoing transmissions. It also strips and interprets, as appropriate, the incoming protocol information.

10.2.6 Transmission Error Handling

The Host Controller must be capable of detecting the following transmission error conditions, which are defined from the host's point of view:

- Timeout conditions after a host-transmitted token or packet. These errors occur when the addressed endpoint is unresponsive or when the structure of the transmission is so badly damaged that the targeted endpoint does not recognize it.
- Data errors resulting in missing or invalid transmissions:
  - The Host Controller sends or receives a packet shorter than that required for the transmission: for example, a transmission extending beyond EOF or a lack of resources available to the Host Controller.
  - An invalid CRC field on a received data packet.

- Protocol errors:
  - An invalid handshake PID, such as a malformed or inappropriate handshake
  - A false EOP
  - A bit stuffing error.

For each bulk, command, and interrupt transaction, the host must maintain an error count tally. Errors result from the conditions described above, not as a result of an endpoint NAKing a request. This value reflects the number of times the transaction has encountered a transmission error. If the error count tally for a given transaction reaches three, the host retires the transfer. When a transfer is retired due to excessive errors, the last error type will be indicated. Isochronous transactions are attempted only once, regardless of outcome, and, therefore, no error count is maintained for this type.

10.2.7 Remote Wakeup

If USB System wishes to place the bus in the Suspended state, it commands the Host Controller to stop all bus traffic, including SOFs. This causes all USB devices to enter the Suspended state. In this state, the USB System may enable the Host Controller to respond to bus wakeup events. This allows the Host Controller to respond to bus wakeup signaling to restart the host system.

10.2.8 Root Hub

The root hub provides the connection between the Host Controller and one or more USB ports. The root hub provides the same functionality as other hubs (See Chapter 11), except that the hardware and software interface between the root hub and the Host Controller is defined by the specific hardware implementation.

10.2.8.1 Port Resets

Section 7.1.7.3 describes the requirements of a hub to ensure all upstream resume attempts are overpowered with a long reset downstream. Root hubs may provide an aggregate reset period of at least 50ms. If the reset duration is controlled in hardware and the hardware timer is <50ms, the USB System can issue several consecutive resets to accumulate a sufficiently long reset to the device.

10.2.9 Host System Interface

The Host Controller provides a high-speed bus-mastering interface to and from main system memory. The physical transfer between memory and the USB wire is performed automatically by the Host Controller. When data buffers need to be filled or emptied, the Host Controller informs the USB System.

10.3 Overview of Software Mechanisms

The HCD and the USBD present software interfaces based on different levels of abstraction. They are, however, expected to operate together in a specified manner to satisfy the overall requirements of the USB System (see Figure 10-2). The requirements for the USB System are expressed primarily as requirements for the USBDI. The division of duties between the USBD and the HCD is not defined. However, the one requirement of the HCDI that must be met is that it supports, in the specified operating system context, multiple Host Controller implementations.

The HCD provides an abstraction of the Host Controller and an abstraction of the Host Controller's view of data transfer across the USB. The USBD provides an abstraction of the USB device and of the data transfers between the client of the USBD and the function on the USB device. Overall, the USB System acts as a facilitator for transmitting data between the client and the function and as a control point for the USB-specific interfaces of the USB device. As part of facilitating data transfer, the USB System provides buffer management capabilities and allows the synchronization of the data transmittal to the needs of the client and the function.

The specific requirements for the USBDI are described later in this chapter. The exact functions that fulfill these requirements are described in the relevant operating system environment guide for the HCDI and the USBDI. The procedures involved in accomplishing data transfers via the USBDI are described in the following sections.

10.3.1 Device Configuration

Different operating system environments perform device configuration using different software components and different sequences of events. The USB System does not assume a specific operating system method. However, there are some basic requirements that must be fulfilled by any USB System implementation. In some operating systems existing host software provides these requirements. In others, the USB System provides the capabilities.

The USB System assumes a specialized client of the USBD, called a hub driver, that acts as a clearinghouse for the addition and removal of devices from a particular hub. Once the hub driver receives such notifications, it will employ additional host software and other USBD clients, in an operating system specific manner, to recognize and configure the device. This model, shown in Figure 10-4, is the basis of the following discussion.

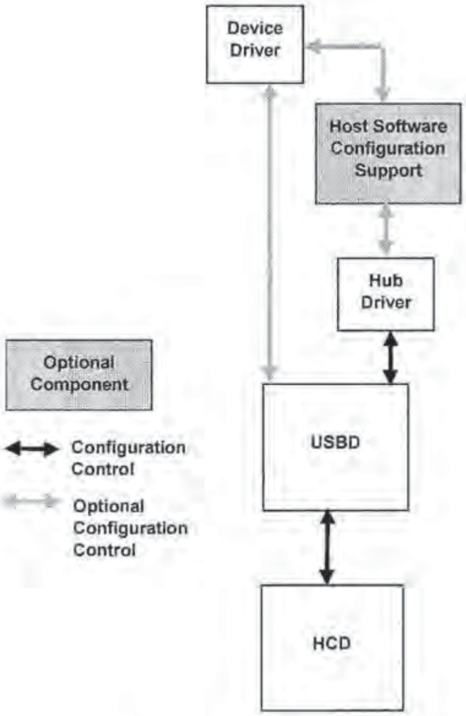


Figure 10-4. Configuration Interactions

When a device is attached, the hub driver receives a notification from the hub detecting the change. The hub driver, using the information provided by the hub, requests a device identifier from the USBD. The USBD in turn sets up the default pipe for that device and returns a device identifier to the hub driver.

The device is now ready to be configured for use. For each device, there are three configurations that must be complete before that device is ready for use:

1. Device Configuration: This includes setting up all of the device's USB parameters and allocating all USB host resources that are visible to the device. This is accomplished by setting the configuration value on the device. A limited set of configuration changes, such as alternate settings, is allowed without totally reconfiguring the device. Once the device is configured, it is, from its point of view, ready for use.
2. USB Configuration: In order to actually create a USBD pipe ready for use by a client, additional USB information, not visible to the device, must be specified by the client. This information, known as the Policy for the pipe, describes how the client will use the pipe. This includes such items as the maximum amount of data the client will transfer with one IRP, the maximum service interval the client will use, the client's notification identification, and so on.
3. Function Configuration: Once configuration types 1 and 2 have been accomplished, the pipe is completely ready for use from the USB's point of view. However, additional vendor- or class-specific setup may be required before the client can actually use the pipe. This configuration is a private matter between the device and the client and is not standardized by the USBD.

The following paragraphs describe the device and USB configuration requirements.

The responsible configuring software performs the actual device configuration. Depending on the particular operating system implementation, the software responsible for configuration can include the following:

- The hub driver
- Other host software
- A device driver.

The configuring software first reads the device descriptor, then requests the description for each possible configuration. It may use the information provided to load a particular client, such as a device driver, which initially interacts with the device. The configuring software, perhaps with input from that device driver, chooses a configuration for the device. Setting the device configuration sets up all of the endpoints on the device and returns a collection of interfaces to be used for data transfer by USBD clients. Each interface is a collection of pipes owned by a single client.

This initial configuration uses the default settings for interfaces and the default bandwidth for each endpoint. A USBD implementation may additionally allow the client to specify alternate interfaces when selecting the initial configuration. The USB System will verify that the resources required for the support of the endpoint are available and, if so, will allocate the bandwidth required. Refer to Section 10.3.2 for a discussion of resource management.

The device is now configured, but the created pipes are not yet ready for use. The USB configuration is accomplished when the client initializes each pipe by setting a Policy to specify how it will interact with the pipe. Among the information specified is the client's maximum service interval and notification information. Among the actions taken by the USB System, as a result of setting the Policy, is determining the amount of buffer working space required beyond the data buffer space provided by the client. The size of the buffers required is based upon the usage chosen by the client and upon the per-transfer needs of the USB System.

The client receives notifications when IRPs complete, successfully or due to errors. The client may also wake up independently of USB notification to check the status of pending IRPs.

The client may also choose to make configuration modifications, such as enabling an alternate setting for an interface or changing the bandwidth allocated to a particular pipe. In order to perform these changes, the interface or pipe, respectively, must be idle.

10.3.2 Resource Management

Whenever a pipe is setup by the USBD for a given endpoint, the USB System must determine if it can support the pipe. The USB System makes this determination based on the requirements stated in the endpoint descriptor. One of the endpoint requirements, which must be supported in order to create a pipe for an endpoint, is the bandwidth necessary for that endpoint's transfers. There are two stages to check for available bandwidth. First the maximum execution time for a transaction is calculated. Then, the frame schedule is consulted to determine if the indicated transaction will fit.

The allocation of the guaranteed bandwidth for isochronous and interrupt pipes, and the determination of whether a particular control or bulk transaction will fit into a given frame, can be determined by a software heuristic in the USB System. If the actual transaction execution time in the Host Controller exceeds the heuristically determined value, the Host Controller is responsible for ensuring that frame integrity is maintained (refer to Section 10.2.3). The following discussion describes the requirements for the USB System heuristic.

In order to determine if bandwidth can be allocated, or if a transaction can be fit into a particular frame, the maximum transaction execution time must be calculated. The calculation of the maximum transaction execution time requires that the following information be provided. (Note that an agent other than the client may provide some of this information.)

- Number of data bytes (*wMaxPacketSize*) to be transmitted.
- Transfer type.
- Depth in the topology. If less precision is allowed, the maximum topology depth may be assumed.

This calculation must include the bit transmission time, the signal propagation delay through the topology, and any implementation-specific delays, such as preparation or recovery time required by the Host Controller itself. Refer to Chapter 5 for examples of formulas that can be used for such calculations.

10.3.3 Data Transfers

The basis for all client-function communication is the interface: a bundle of related pipes associated with a particular USB device.

Exactly one client on the host manages a given interface. The client initializes each pipe of an interface by setting the Policy for that pipe. This includes the maximum amount of data to be transmitted per IRP and the maximum service interval for the pipe. A service interval is stated in milliseconds and describes the interval over which an IRP's data will be transmitted for an isochronous pipe. It describes the polling interval for an interrupt pipe. The client is notified when a specified request is completed. The client manages the size of each IRP such that its duty cycle and latency constraints are maintained. Additional Policy information includes the notification information for the client.

The client provides the buffer space required to hold the transmitted data. The USB System uses the Policy to determine the additional working space it will require.

The client views its data as a contiguous serial stream, which it manages in a similar manner to those streams provided over other types of bus technologies. Internally, the USB System may, depending on its own Policy and any Host Controller constraints, break the client request down into smaller requests to be sent across the USB. However, two requirements must be met whenever the USB System chooses to undertake such division:

- The division of the data stream into smaller chunks is not visible to the client.
- USB samples are not split across bus transactions.

When a client wishes to transfer data, it will send an IRP to the USBD. Depending on the direction of data transfer, a full or empty data buffer will be provided. When the request is complete (successfully or due to an error condition), the IRP and its status is returned to the client. Where relevant, this status is also provided on a per-transaction basis.

10.3.4 Common Data Definitions

In order to allow the client to receive request results as directly as possible from its device, it is desirable to minimize the amount of processing and copying required between the device and the client. To facilitate this, some control aspects of the IRP are standardized such that different layers in the stack may directly use the information provided by the client. The particular format for this data is dependent on the actualization of the USBDI in the operating system. Some data elements may in fact not be directly visible to the client at all, but are generated as a result of the client request.

The following data elements define the relevant information for a request:

- Identification of the pipe associated with the request. Identifying this pipe also describes information such as transfer type for this request.
- Notification identification for the particular client.
- Location and length of data buffer that is to be transmitted or received.
- Completion status for the request. Both the summary status and, as required, detailed per-transaction status must be provided.
- Location and length of working space. This is implementation-ependent.

The actual mechanisms used to communicate requests to the USBD are operating system-pecific. However, beyond the requirements stated above for what request-related information must be available, there are also requirements on how requests will be processed. The basic requirements are described in Chapter 5. Additionally, the USBD provides a mechanism to designate a group of isochronous IRPs for which the transmission of the first transaction of each IRP will occur in the same frame. The USBD also provides a mechanism for designating an uninterruptable set of vendor- or class-specific requests to a default pipe. No other requests to that default pipe, including standard, class, or vendor request may be inserted in the execution flow for such an uninterruptable set. If any request in this set fails, the entire set is retired.

10.4 Host Controller Driver

The Host Controller Driver (HCD) is an abstraction of Host Controller hardware and the Host Controller's view of data transmission over the USB. The HCDI meets the following requirements:

- Provides an abstraction of the Host Controller hardware.
- Provides an abstraction for data transfers by the Host Controller across the USB interconnect.
- Provides an abstraction for the allocation (and de-allocation) of Host Controller resources, to support guaranteed service to USB devices.
- Presents the root hub and its behavior according to the hub class definition. This includes supporting the root hub such that the hub driver interacts with the root hub exactly as it would for any hub. In particular, even though a root hub can be implemented in a combination of hardware and software, the root hub responds initially to the default device address (from a client perspective), returns descriptor information, supports having its device address set, and supports the other hub class requests. However, bus transactions may or may not need to be generated to accomplish this behavior given the close integration possible between the Host Controller and the root hub.

The HCD provides a software interface (HCDI) that implements the required abstractions. The function of the HCD is to provide an abstraction, which hides the details of the Host Controller hardware. Below the Host Controller hardware is the physical USB and all the attached USB devices.

The HCD is the lowest tier in the USB software stack. The HCD has only one client: the Universal Serial Bus Driver (USBD). The USBD maps requests from many clients to the appropriate HCD. A given HCD may manage many Host Controllers.

The HCDI is not directly accessible from a client. Therefore, the specific interface requirements for the HCDI are not discussed here.

10.5 Universal Serial Bus Driver

The USBD provides a collection of mechanisms that operating system components, typically device drivers, use to access USB devices. The only access to a USB device is that provided by the USBD. The USBD implementations are operating system-specific. The mechanisms provided by the USBD are implemented using as appropriate and augmenting as necessary the mechanisms provided by the operating system environment in which the USB runs. The following discussion centers on the basic capabilities required for all USBD implementations. For specifics of the USBD operation within a specific environment, see the relevant operating system environment guide for the USBD. A single instance of the USBD directs accesses to one or more HCDs that in turn connect to one or more Host Controllers. If allowed, how USBD instancing is managed is dependent upon the operating system environment. However, from the client's point of view, the USBD with which the client communicates manages all of the attached USB devices.

10.5.1 USBD Overview

Clients of USBD direct commands to devices or move streams of data to or from pipes. The USBD presents two groups of software mechanisms to clients: command mechanisms and pipe mechanisms.

Command mechanisms allow clients to configure and control USBD operation as well as to configure and generically control a USB device. In particular, command mechanisms provide all access to the device's default pipe.

Pipe mechanisms allow a USBD client to manage device specific data and control transfers. Pipe mechanisms do not allow a client to directly address the device's default pipe.

Figure 10-5 presents an overview of the USBD structure.

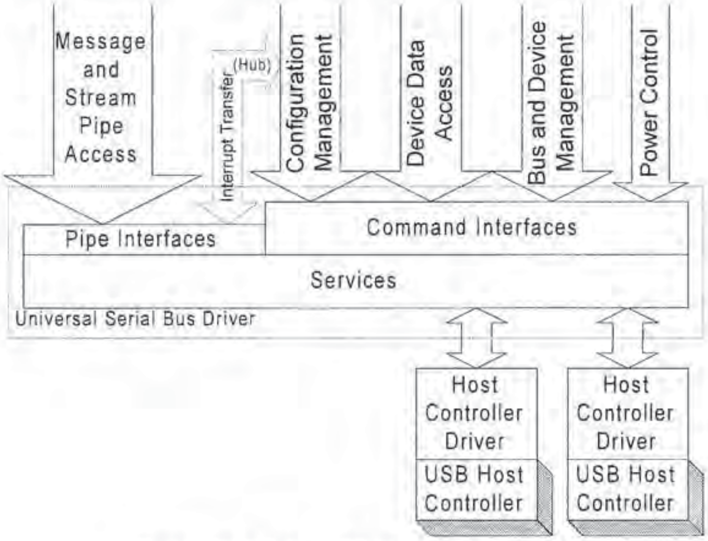


Figure 10-5. Universal Serial Bus Driver Structure

10.5.1.1 USBD Initialization

Specific USBD initialization is operating system-dependent. When a particular USB managed by USBD is initialized, the management information for that USB is also created. Part of this management information is the default address device and its default pipe.

When a device is attached to a USB, it responds to a special address known as the default address (refer to Chapter 9) until its unique address is assigned by the bus enumerator. In order for the USB System to interact with the new device, the default device address and the device's default pipe must be available to the hub driver whenr a device is attached. During device initialization, the default address is changed to a unique address.

10.5.1.2 USBD Pipe Usage

Pipes are the method by which a device endpoint is associated with a Host Software entity. Pipes are owned by exactly one such entity on the host. Although the basic concept of a pipe is the same no matter who the owner, some distinction of capabilities provided to the USBD client occurs between two groups of pipes:

- Default pipes, which are owned and managed by the USBD
- All other pipes, which are owned and managed by clients of the USBD.

Default pipes are never directly accessed by clients, although they are often used to fulfill some part of client requests relayed via command mechanisms.

10.5.1.2.1 Default Pipes

The USBD is responsible for allocating and managing appropriate buffering to support transfers on the default pipe that are not directly visible to the client such as setting a device address. For those transfers that are directly visible to the client, such as sending vendor and class commands or reading a device descriptor, the client must provide the required buffering.

10.5.1.2.2 Client Pipes

Any pipe not owned and managed by the USBD can be owned and managed by a USBD client. From the USBD viewpoint, a single client owns the pipe. In fact, a cooperative group of clients can manage the pipe, provided they behave as a single coordinated entity when using the pipe.

The client is responsible for providing the amount of buffering it needs to service the data transfer rate of the pipe within a service interval attainable by the client. Additional buffering requirements for working space are specified by the USB System.

10.5.1.3 USBD Service Capabilities

The USBD provides services in the following categories:

- Configuration via command mechanisms
- Transfer services via both command and pipe mechanisms
- Event notification
- Status reporting and error recovery.

10.5.2 USBD Command Mechanism Requirements

USBD command mechanisms allow a client generic access to a USB device. Generally, these commands allow the client to make read or write accesses to one of potentially several device data and control spaces. The client provides as little as a device identifier and the relevant data or empty buffer pointer.

USBD command transfers do not require that the USB device be configured. Many of the device configuration facilities provided by the USBD are command transfers.

Following are the specific requirements on the command mechanisms provided.

10.5.2.1 Interface State Control

USBD clients must be able to set a specified interface to any settable pipe state. Setting an interface state results in all of the pipes in that interface moving to that state. Additionally, all of the pipes in an interface may be reset or aborted.

10.5.2.2 Pipe State Control

USBD pipe state has two components:

- Host status
- Reflected endpoint status.

Whenever the pipe status is reported, the value for both components will be identified. The pipe status reflected from the endpoint is the result of the endpoint being in a particular state. The USBD client manages the pipe state as reported by the USBD. For any pipe state reflected from the endpoint, the client must also interact with the endpoint to change the state.

A USBD pipe is in exactly one of the following states:

- Active: The pipe's Policy has been set and the pipe is able to transmit data. The client can query as to whether any IRPs are outstanding for a particular pipe. Pipes for which there are no outstanding IRPs are still considered to be in the Active state as long as they are able to accept new IRPs.
- Halted: An error has occurred on the pipe. This state may also be a reflection of the corresponding Halted endpoint on the device.

A pipe and endpoint are considered active when the device is configured and the pipe and/or endpoint is not stalled. Clients may manipulate pipe state in the following ways:

- Aborting a Pipe: All of the IRPs scheduled for a pipe are retired immediately and returned to the client with a status indicating they have been aborted. Neither the host state nor the reflected endpoint state of the pipe is affected.
- Resetting a Pipe: The pipe's IRPs are aborted. The host state is moved to Active. If the reflected endpoint state needs to be changed, that must be commanded explicitly by the USBD client.
- Clearing a Halted pipe: The pipe's state is cleared from *Halted* to *Active*.
- Halting a pipe: The pipe's state is set to *Halted*.

10.5.2.3 Getting Descriptors

The USBDI must provide a mechanism to retrieve standard device, configuration and string descriptors, as well as any class- or vendor-specific descriptors.

10.5.2.4 Getting Current Configuration Settings

The USBDI must provide a facility to return, for any specified device, the current configuration descriptor. If the device is not configured, no configuration descriptor is returned. This action is equivalent to returning the configuration descriptor for the current configuration by requesting the specific configuration descriptor. It does not, however, require the client to know the identifier for the current configuration. This will return all of the configuration information, including the following:

- All of the configuration descriptor information as stored on the device, including all of the alternate settings for all of the interfaces
- Indicators for which of the alternate settings for interfaces are active
- Pipe handles for endpoints in the active alternate settings for interfaces
- Actual *wMaxPacketSize* values for endpoints in the active alternate settings for interfaces.

Additionally, for any specified pipe, the USBDI must provide a facility to return the *wMaxPacketSize* that is currently being used by the pipe.

10.5.2.5 Adding Devices

The USBDI must provide a mechanism for the hub driver to inform USBD of the addition of a new device to a specified USB and to retrieve the USBD ID of the new USB device. The USBD tasks include assigning the device address and preparing the device's default pipe for use.

10.5.2.6 Removing Devices

The USBDI must provide a facility for the hub driver to inform the USBD that a specific device has been removed.

10.5.2.7 Managing Status

The USBDI must provide a mechanism for obtaining and clearing device-based status, on a device, interface, or pipe basis.

10.5.2.8 Sending Class Commands

This USBDI mechanism is used by a client, typically a class-specific or adaptive driver, to send one or more class-specific commands to a device.

10.5.2.9 Sending Vendor Commands

This USBDI mechanism is used by a client to send one or more vendor-specific commands to a device.

10.5.2.10 Establishing Alternate Settings

The USBDI must provide a mechanism to change the alternate setting for a specified interface. As a result, the pipe handles for the previous setting are released and new pipe handles for the interface are returned. For this request to succeed the interface must be idle; i.e., no data buffers may be queued for any pipes in the interface.

10.5.2.11 Establishing a Configuration

Configuring software requests a configuration by passing a buffer containing the configuration information to the USBD. The USBD requests resources for the endpoints in the configuration, and if all resource requests succeed, the USBD sets the device configuration and returns interface handles with corresponding pipe handles for all of the active endpoints. The default values are used for all alternate settings for interfaces.

Note: the interface implementing the configuration may require specific alternate settings to be identified.

10.5.2.12 Setting Descriptors

For devices supporting this behavior, the USBDI allows existing descriptors to be updated or new descriptors to be added.

10.5.3 USBD Pipe Mechanisms

This part of the USBDI offers clients the highest-speed, lowest overhead data transfer services possible. Higher performance is achieved by shifting some pipe management responsibilities from the USBD to the client. As a result, the pipe mechanisms are implemented at a more primitive level than the data transfer services provided by the USBD command mechanisms. Pipe mechanisms do not allow access to a device's default pipe.

USBD pipe transfers are available only after both the device and USB configuration have completed successfully. At the time the device is configured, the USBD requests the resources required to support all device pipes in the configuration. Clients are allowed to modify the configuration, constrained by whether the specified interface or pipe is idle.

Clients provide full buffers to outgoing pipes and retrieve transfer status information following the completion of a request. The transfer status returned for an outgoing pipe allows the client to determine the success or failure of the transfer.

Clients provide empty buffers to incoming pipes and retrieve the filled buffers and transfer status information from incoming pipes following the completion of a request. The transfer status returned for an incoming pipe allows a client to determine the amount and the quality of the data received.

10.5.3.1 Supported Pipe Types

The four types of pipes supported, based on the four transfer types, are described in the following sections.

10.5.3.1.1 Isochronous Data Transfers

Each buffer queued for an isochronous pipe is required to be viewable as a stream of samples. As with all pipe transfers, the client establishes a Policy for using this isochronous pipe, including the relevant service interval for this client. Lost or missing bytes, which are detected on input, and transmission problems, which are noted on output, are indicated to the client.

The client queues a first buffer, starting the pipe streaming service. To maintain the continuous streaming transfer model used in all isochronous transfers, the client queues an additional buffer before the current buffer is retired.

The USBD is required to be able to provide a sample stream view of the client's data stream. In other words, using the client's specified method of synchronization, the precise packetization of the data is hidden from the client. Additionally, a given transaction is always contained completely within some client data buffer.

For an output pipe, the client provides a buffer of data. The USBD allocates the data across the frames for the service period using the client's chosen method of synchronization.

For an input pipe, the client must provide an empty buffer large enough to hold the maximum number of bytes the client's device will deliver in the service period. Where missing or invalid bytes are indicated, the USBD may leave the space that the bytes would have occupied in place in the buffer and identify the error. One of the consequences of using no synchronization method is that this reserved space is assumed to be the maximum packet size. The buffer-retired notification occurs when the IRP completes. Note that the input buffer need not be full when returned to the client.

The USBD may optionally provide additional views of isochronous data streams. The USBD is also required to be able to provide a packet stream view of the client's data stream.

10.5.3.1.2 Interrupt Transfers

The Interrupt out transfer originates in the client of the USBD and is delivered to the USB device. The Interrupt in transfer originates in a USB device and is delivered to a client of the USBD. The USB System guarantees that the transfers meet the maximum latency specified by the USB endpoint descriptor.

The client queues a buffer large enough to hold the interrupt transfer data (typically a single USB transaction). When all of the data is transferred, or if the error threshold is exceeded, the IRP is returned to the client.

10.5.3.1.3 Bulk Transfers

Bulk transfers may originate either from the device or the client. No periodicity or guaranteed latency is assumed. When all of the data is transferred, or if the error threshold is exceeded, the IRP is returned to the client.

10.5.3.1.4 Control Transfers

All message pipes transfer data in both directions. In all cases, the client outputs a setup stage to the device endpoint. The optional data stage may be either input or output and the final status is always logically presented to the host. For details of the defined message protocol, refer to Chapter 8.

The client prepares a buffer specifying the command phase and any optional data or empty buffer space. The client receives a buffer-retired notification when all phases of the control transfer are complete, or an error notification, if the transfer is aborted due to transmission error.

10.5.3.2 USBD Pipe Mechanism Requirements

The following pipe mechanisms are provided.

10.5.3.2.1 Aborting IRPs

The USBDI must allow IRPs for a particular pipe to be aborted.

10.5.3.2.2 Managing Pipe Policy

The USBDI must allow a client to set and clear the Policy for an individual pipe or for an entire interface. Any IRPs made by the client prior to successfully setting a Policy are rejected by the USBD.

10.5.3.2.3 Queuing IRPs

The USBDI must allow clients to queue IRPs for a given pipe. When IRPs are returned to the client, the request status is also returned. A mechanism is provided by the USBD to identify a group of isochronous IRPs whose first transactions will all occur in the same frame.

10.5.3.2.4 Being a Master Client

The Master Client is allowed to adjust the number of bit times in a frame. This mechanism is used to synchronize the USB to a device, such as an ISDN port. A client requesting master status identifies itself with an interface handle for the device from which it is mastering.

The USBDI must allow a client to request becoming a Master Client for a given USB and to release this capability when it is no longer required. The USB will grant Master Client Status only to a single client. Attempts by other clients to become the Master Client are ignored until the current Master Client relinquishes control. The Master Client may explicitly release master status, or the client's master status will be automatically released when the referenced device is reset or detached.

10.5.4 Managing the USB via the USBD Mechanisms

Using the provided USBD mechanisms, the following general capabilities are supported by any USB System.

10.5.4.1 Configuration Services

Configuration services operate on a per-device basis. The configuring software tells the USBD when to perform device configuration. A hub driver has a special role in device management and provides at least the following capabilities:

- Device attach/detach recognition, driven by an interrupt pipe owned by the hub driver
- Device reset, accomplished by the hub driver by resetting the hub port upstream of the device
- Tells the USBD to perform device address assignment
- Power control.

The USBDI additionally provides the following configuration facilities, which may be used by the hub driver or other configuring software available on the host:

- Device identification and access to configuration information (via access to descriptors on the device)
- Device configuration via command mechanisms.

When the hub driver informs the USBD of a device attachment, the USBD establishes the default pipe for the new device.

10.5.4.1.1 Configuration Management

Configuration management services are provided primarily as a set of specific interface commands that generate USB transactions on the default pipe. The notable exception is the use of an additional interrupt pipe that delivers hub status directly to the hub driver.

Every hub initiates an interrupt transfer when there is a change in the state of one of the hub ports. Generally, the port state change will be the connection or removal of a downstream USB device. (Refer to Chapter 11 for more information.)

10.5.4.1.2 Initial Device Configuration

The device configuration process begins when a hub reports, via its status change pipe, the connection of a new USB device.

Configuration management services allow configuring software to select a USB device configuration from the set of configurations listed in the device. The USBD verifies that adequate power is available and the data transfer rates given for all endpoints in the configuration do not exceed the capabilities of the USB with the current schedule before setting the device configuration.

10.5.4.1.3 Modifying a Device Configuration

Configuration management services allow configuring software to replace a USB device configuration with another configuration from the set of configurations listed in the device. The operation succeeds if adequate power is available and the data transfer rates given for all endpoints in the new configuration fit within the capabilities of the USB with the current schedule. If the new configuration is rejected, the previous configuration remains.

Configuration management services allow configuring software to return a USB device to a Not Configured state.

10.5.4.1.4 Device Removal

Error recovery and/or device removal processing begins when a hub reports via its status change pipe that the USB device has been removed.

10.5.4.2 Bus and Device Management

Bus and Device Management services allow a client to become the Master Client on a USB, and as the Master Client, to adjust the number of bit times in a frame on that bus. A Master Client may add or subtract one bit time to the current USB frame. Adjusting SOFs more frequently than once every 6ms has undefined results.

10.5.4.3 Power Control

There are two cooperating levels of power management for the USB: bus and device level management. This specification provides mechanisms for managing power on the USB bus. Device classes may define class-specific power control capabilities.

All USB devices must support the Suspended state (refer to Chapter 9). The device is placed into the Suspended state via control of the hub port to which the device is attached. Normal device operation ceases in the Suspend State, however, if the device is capable of wakeup signaling and the device is enabled for remote wakeup it may generate resume signaling in response to external events.

The power management system may transition a device to the Suspended state or power-off the device in order to control and conserve power. The USB provides neither requirements nor commands for the device state to be saved and restored across these transitions. Device classes may define class-specific device state save-and-restore capabilities.

The USB System coordinates the interaction between device power states and the Suspended state.

10.5.4.4 Event Notifications

USBD clients receive several kinds of event notifications through a number of sources:

- Completion of an action initiated by a client.
- Interrupt transfers over stream pipes can deliver notice of device events directly to USBD clients. For example, hubs use an interrupt pipe to deliver events corresponding to changes in hub status.
- Event data can be embedded by devices in streams.
- Standard device interface commands, device class commands, vendor-specific commands, and even general control transfers over message pipes can all be used to poll devices for event conditions.

10.5.4.5 Status Reporting and Error Recovery Services

The command and pipe mechanisms both provide status reporting on individual requests as they are invoked and completed.

Additionally, USB device status is available to USBD clients using the command mechanisms.

The USBD provides clients with pipe error recovery mechanisms by allowing pipes to be reset or aborted.

10.5.4.6 Managing Remote Wakeup Devices

The USB System can minimize the resume power consumption of a suspended USB tree. This is accomplished by explicitly enabling devices capable of resume signaling and controlling propagation of resume signaling via selectively suspending and/or disabling hub ports between the device and the nearest self-powered, awake hub.

In some error-recovery scenarios, the USB System will need to re-enumerate sub-trees. The sub-tree may be partially or completely suspended. During error-recovery, the USB System must avoid contention between a device issuing resume signaling and simultaneously driving reset down the port. Avoidance is accomplished via management of the devices' remote wakeup feature and the hubs' port features. The rules are as follows:

- Issue a SetDeviceFeature(DEVICE_REMOTE_WAKEUP) request to the leaf device, only just prior to selectively suspending any port between where the device is connected and the root port (via a SetPortFeature(PORT_SUSPEND) request).
- Do not reset a suspended port that has had a device enabled for remote wakeup without first enabling that port.

10.5.5 Passing USB Preboot Control to the Operating System

A single software driver owns the Host Controller. If the host system implements USB services before the operating system loads, the Host Controller must provide a mechanism that disables access by the preboot software and allows the operating system to gain control. Preboot USB configuration is not passed to the operating system. Once the operating system gains control it is responsible to fully configure the bus. If the operating system provides a mechanism to pass control back to the preboot environment, the bus will be in an unknown state. The preboot software should treat this event as a powerup.

10.6 Operating System Environment Guides

As noted previously, the actual interfaces between the USB System and host software are specific to the host platform and operating system. A companion specification is required for each combination of platform and operating system with USB support. These specifications describe the specific interfaces used to integrate the USB into the host. Each operating system provider for the USB System identifies a compatible Universal USB Specification revision.

Chapter 11
Hub Specification

This chapter describes the architectural requirements for the USB hub. It contains a description of the two principal sub-blocks: the Hub Repeater and the Hub Controller. The chapter also describes the hub's operation for error recovery, reset, and suspend/resume. The second half of the chapter defines hub request behavior and hub descriptors.

The hub specification supplies sufficient additional information to permit an implementer to design a hub that conforms to the USB specification.

11.1 Overview

Hubs provide the electrical interface between USB devices and the host. Hubs are directly responsible for supporting many of the attributes that make USB user friendly and hide its complexity from the user. Listed below are the major aspects of USB functionality that hubs must support:

- Connectivity behavior
- Power management
- Device connect/disconnect detection
- Bus fault detection and recovery
- Full- and low-speed device support.

A hub consists of two components: the Hub Repeater and the Hub Controller. The Hub Repeater is responsible for connectivity setup and tear-down. It also supports exception handling, such as bus fault detection and recovery and connect/disconnect detect. The Hub Controller provides the mechanism for host-to-hub communication. Hub-specific status and control commands permit the host to configure a hub and to monitor and control its individual downstream ports.

11.1.1 Hub Architecture

Figure 11-1 shows a hub and the locations of its upstream and downstream ports. A hub consists of a Hub Repeater section and a Hub Controller section. The Hub Repeater is responsible for managing connectivity on a per-packet basis, while the Hub Controller provides status and control and permits host access to the hub.

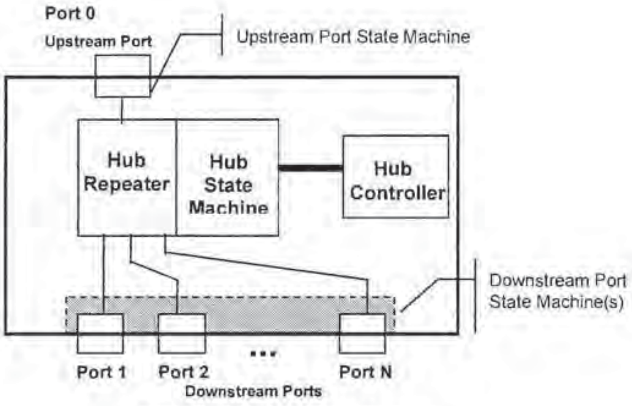


Figure 11-1. Hub Architecture

11.1.2 Hub Connectivity

Hubs display differing connectivity behavior, depending on whether they are propagating packet traffic or resume signaling, or are in the Idle state.

11.1.2.1 Packet Signaling Connectivity

The Hub Repeater contains one port that must always connect in the upstream direction (referred to as the upstream port) and one or more downstream ports. Upstream connectivity is defined as being towards the host, and downstream connectivity is defined as being towards a device. Figure 11-2 shows the packet signaling connectivity behavior for hubs in the upstream and downstream directions. A hub also has an Idle state, during which the hub makes no connectivity. When in the Idle state, all of the hub's ports are in the receive mode waiting for the start of the next packet.

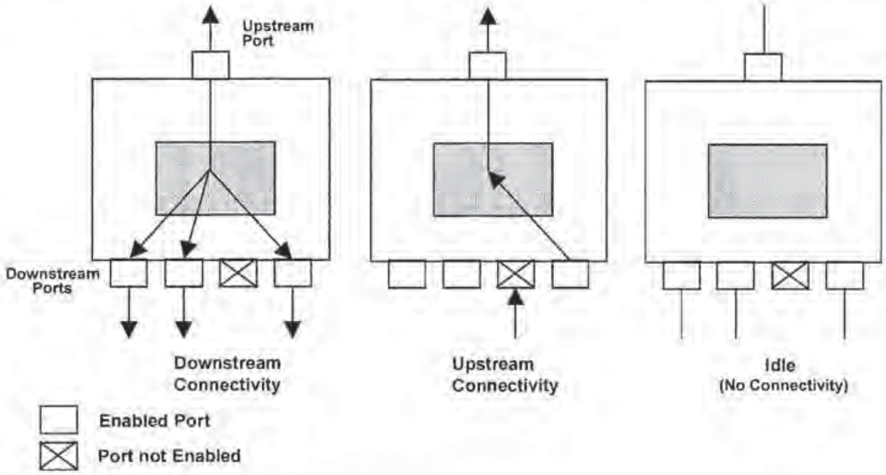


Figure 11-2. Hub Signaling Connectivity

If a downstream hub port is enabled (i.e., in a state where it can propagate signaling through the hub) and the hub detects a Start-of-Packet (SOP) on that port, connectivity is established in an upstream direction to the upstream port of that hub, but not to any other downstream ports. This means that when a device or a hub transmits a packet upstream, only those hubs in line between the transmitting device and the host will see the packet. Refer to Section 11.8.3 for optional behavior when a hub detects simultaneous upstream signaling on more than one port.

In the downstream direction, hubs operate in a broadcast mode. When a hub detects an SOP on its upstream port, it establishes connectivity to all enabled downstream ports. If a port is not enabled, it does not propagate packet signaling downstream.

11.1.2.2 Resume Connectivity

Hubs exhibit different connectivity behaviors for upstream- and downstream-directed resume signaling. A hub that is suspended reflects resume signaling from its upstream port to all of its enabled downstream ports. Figure 11-3 illustrates hub upstream and downstream resume connectivity.

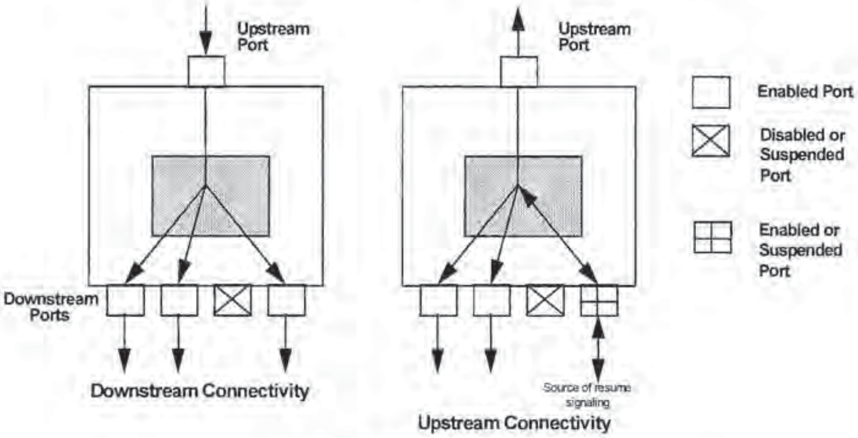


Figure 11-3. Resume Connectivity

If a hub is suspended and detects resume signaling from a selectively suspended or an enabled downstream port, the hub reflects that signaling upstream and to all of its enabled downstream ports, including the port that initiated the resume sequence. Resume signaling is not reflected to disabled or suspended ports. A detailed discussion of resume connectivity appears in Section 11.9.

11.1.2.3 Hub Fault Recovery Mechanisms

Hubs are the essential USB component for establishing connectivity between the host and other devices. It is vital that any connectivity faults, especially those that might result in a deadlock, be detected and prevented from occurring. Hubs need to handle connectivity faults only when they are in the repeater mode.

Hubs must also be able to detect and recover from lost or corrupted packets that are addressed to the Hub Controller. Because the Hub Controller is, in fact, another USB device, it must adhere to the same timeout rules as other USB devices, as described in Chapter 8.

11.2 Hub Frame Timer

Each hub has a frame timer whose timing is derived from the hub's local clock and is synchronized to the host frame period by the host-generated Start-of-Frame (SOF). The frame timer provides timing references that are used to allow the hub to detect a babbling device and prevent the hub from being disabled by the upstream hub. The hub frame timer must track the host frame period and be capable of remaining synchronized with the host even if two consecutive SOF tokens are missed by the hub.

The frame timer must lock to the host's frame timing for worst case tolerances and offsets between the host and hub. The offsets have to accommodate the hub oscillator tolerance (≤ 500ppm) and accuracy (≤ 2500ppm) as well as the host's allowed frame tolerance of ≤ 500ppm. The range of the hub frame timer is:

12,000 * 1±(hub accuracy + hub tolerance + host tolerance)

The host tolerance is allowed to be ±500ppm, meaning that a frame time is between 0.9995ms and 1.0005ms, absolute. If the hub's oscillator is at the limits of its accuracy and tolerance, it can be running at between 11,964,000Hz and 12,036,000Hz. If the host is generating an SOF every 1.0005ms and the hub is running at 12,036,000Hz, then the hub's frame timer will count 12,042 times between each SOF. If the host is generating an SOF every 0.9995ms and the hub is running at 11,964,000Hz, then the hub's frame timer will count 11,958 times between each SOF. If the hub accuracy and tolerance are both zero, the hub frame timer range is ±6 bit times.

11.2.1 Frame Timer Synchronization

A hub's frame timer is clocked by the hub's clock source and is synchronized to SOF packets that are derived from the host's frame timer. After a reset or resume, the hub's frame timer is not synchronized. Whenever the hub receives two consecutive SOF packets, its frame timer should be synchronized. Synchronized is synonymous with lock(ed). A example for a method of constructing a timer that properly synchronizes is as follows.

The hub maintains three timer values: frame timer (down counter), current frame (up counter), and next frame (register). After a reset or resume, a flag is set to indicate that the frame timer is not synchronized.

When the first SOF token is detected, the current frame timer resets and starts counting once per hub bit time. On the next SOF, if the timer has not rolled over, the value in the current frame timer is loaded into the next frame register and into the frame timer. The current frame timer is reset to zero and continues to count and the flag is set to indicate that the frame timer is locked. If the current frame timer has rolled over (exceeded 12,043 – a test at 16,383 is adequate), then an SOF was missed and the frame timer and next frame values are not loaded and the flag indicating that the timer is not synchronized remains set.

Whenever the frame timer counts down to zero, the current value of the next frame register is loaded into the frame timer. When an SOF is detected, and the current frame timer has not rolled over, the value of the current frame timer is loaded into the frame timer and the next frame registers. The current frame timer is then reset to zero and continues to count. If the current frame timer has rolled over, then the value in the next frame register is loaded into the frame timer. This process can cause the frame timer to be updated twice in a single frame: once when the frame timer reaches zero and once when the SOF is detected.

The synchronization circuit described above depends on successfully decoding an SOF packet identifier (PID). This means that the frame timer will be synchronized to a time that is at least 16 bit times into the frame. Each implementation will take some time to react to the SOF decode and set the appropriate timer/counter values. (This reaction time is implementation-dependent but is assumed to be less than four full-speed bit times.) Subsequent sections describe the actions that are controlled by the frame timer. These actions are defined at the EOF1, EOF2, and EOF points, which should nominally be the same points in time throughout the bus. EOF1 and EOF2 are defined in later sections. These sections assume that the hub's frame timer will count to zero at the end of the frame (EOF). The circuitry described above will have the frame timer counting to zero 16-20 bit times after the start of a frame (or end of previous frame). The timings and bit offsets in the later sections should be advanced to account for this offset (add 16-20 bit times to the EOF1 and EOF2 points.)

The frame timer provides a indication to the hub Repeater state machine to indicate that the frame timer has synchronized to SOF and that the frame timer is capable of generating the EOF1 and EOF2 timing points. This signal is important after a global resume because of the possibility that a device may have been detached and a different speed device attached while the host was generating a long resume (several seconds) and the disconnect cannot be detected. A different speed device will bias D+ and D- to appear like a K on the hub which would then be treated as an SOP and, unless inhibited, this SOP would propagate though the resumed hubs. Since the hubs would not have seen any SOF's at this point, the hubs would not be synchronized and, thus, unable to generate the EOF1 and EOF2 timing points. The only recovery from this would be for the host to reset and re-enumerate the section of the bus containing the changed device. This scenario is prevented by inhibiting any downstream port from establishing connectivity until the hub is locked after a resume.

11.2.2 EOF1 and EOF2 Timing Points

The EOF1 and EOF2 are timing points that are derived from the hub's frame timer. These timing points are used to ensure that devices and hubs do not interfere with the proper transmission of the SOF packet from the host. *These timing points have meaning only when the frame timer has been synchronized to the SOF.*

The host and hub frame markers, while all synchronized to the host's SOF, are subject to certain skews that dictate the placement of the EOF points. Figure 11-4 illustrates critical End-of-Frame (EOF) timing points. Table 11-1 summarizes the host and hub EOF timing points.

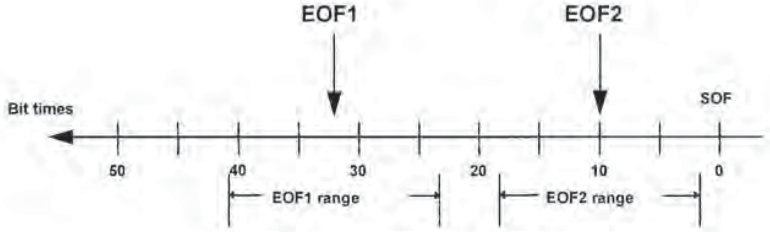


Figure 11-4. EOF Timing Points

At the EOF2 point, any port that has upstream connectivity will be disabled as a babbler. Hubs prevent becoming disabled by sending an End-of-Packet (EOP) to the upstream hub before that hub reaches its EOF2 point (i.e., at EOF1).

Note: a hub is permitted to send the EOP if upstream connectivity is not established at EOF1 time. A hub must send the EOP if connectivity is established from any downstream port at the EOF1 point.

The EOF2 point is defined to occur at least one bit time before the first bit of the SYNC for an SOP. The period allowed for an EOP is four full-speed bit times (the upstream port on a hub is always full-speed.)

Although the hub is synchronized to the SOF, timing skew can accumulate between the host and a hub or between hubs. This timing skew represents the difference between different frame timers on different hubs and the host. The total accumulated skew can be as large as ±9 bit times. This is composed of ±1 bit times per frame of quantization error and ±1 bit per frame of wander. The quantization error occurs when the hub times the interval between SOFs and arrives at a value that is off by a fraction of a bit time but, due to quantization, is rounded to a full bit. Frame wander occurs when the host's frame timer is adjusted by the USB System Software so that the value sampled by the hub in a previous frame differs from the frame interval being used by the host. These values accumulate over multiple frames because SOF packets can be lost and the hub cannot resynchronize its frame timer. This specification allows for the loss of two consecutive SOFs. During this interval the quantization error accumulates to ±3 bit times and the wander accumulates to $\pm 1 \pm 2 \pm 3 = \pm 6$ for a total of ±9 bit times of accumulated skew in three frames. This skew timing affects the placement of the EOF1 and EOF2 points as follows.

Note: although the USB System Software is not allowed to cause the frame interval to change more than one bit time every six frames, the hub skew timing assumes that the frame interval can change one bit time per frame. This cannot be reduced because it would create interoperability problems with hubs designed to previous versions of this specification.

A hub must reach its EOF2 point one bit time before the end of the frame. In order to ensure this, a 9-bit time guard-band must be added so that the EOF2 point is set to occur when the hub's local frame timer reaches 10. A hub must complete its EOP before the hub to which it is attached reaches its EOF2 point. A hub may reach its EOF2 point nine bit times before bit time 10 (at bit time 19 before the SOF). To ensure that the EOP is completed by bit time 19, it must start before bit time 23. To ensure that the hub starts at bit time 23 with respect to another hub, a hub must set its EOF1 point nine bit times ahead of bit time 23 (at bit time 32). If a hub sets its timer to generate an EOP at bit time 32, that EOP may start as much as 9 bit times early (at bit time 41).

Table 11-1. Hub and Host EOF Timing Points

Description	Nominal Number of Bits from Start of SOF	Notes
EOF1	32	End-of-Frame point #1
EOF2	10	End-of-Frame point #2

11.3 Host Behavior at End-of-Frame

It is the responsibility of the USB host controller (the host) to not provoke a response from a device if the response would cause the device to be sending a packet at the EOF2 point. Furthermore, because a hub will terminate an upstream directed packet when the hub reaches its EOF1 point, the host should not start a transaction if a response from the device (data or handshake) would be pending or in process when a hub reaches its EOF1 point. The implications of these limitations are described in the following sections.

In defining the timing points below, the last bit interval in a frame is designated as bit time zero. Bit times in a frame that occur before the last have values that increase the further they are from bit time zero (earlier bit times have higher numbers). These bit time designations are used for convenience only and are not intended to imply a particular implementation. The only requirement of an implementation is that the relative bit time values be preserved.

11.3.1 Latest Host Packet

Hubs are allowed to send an EOP on their upstream ports at the EOF1 point if there is no downstream-directed traffic in progress at that time. To prevent potential contention, the host is not allowed to start a packet if connectivity will not be established on all connections before a hub reaches its EOF1 point. This means that the host must not start a packet after bit time 42.

Note: although there is as much as a six-bit time delay between the time the host starts a packet and all connections are established, this time need not be added to the packet start time as this phase delay exists for the SOF packet as well, causing all hub frame timers to be phase delayed with respect to the host by the propagation delay. There is only one bit time of phase delay between any two adjacent hubs and this has been accounted for in the skew calculations.

11.3.2 Packet Nullification

If a device is sending a packet (data or handshake) when a hub in the device's upstream path reaches its EOF1 point, the hub will send a full-speed EOP. Any packet that is truncated by a hub must be discarded.

A host implementation may discard any packet that is being received at bit time 41. Alternatively, a host implementation may attempt to maximize bus utilization by accepting a packet if the packet is predicted to start at or before bit time 41.

11.3.3 Transaction Completion Prediction

A device can send two types of packets: data and handshake. A handshake packet is always exactly 16 bit times long (sync byte plus PID byte.) The time from the end of a packet from the host until the first bit of the handshake must be seen at the host is 17 bit times. This gives a total allocation of 35 bit times from the end of data packet from the root (start of EOP) until it is predicted that the handshake will be completed (start of EOP) from the device. Therefore, if the host is sending a data packet for which the device can return a handshake (anything other than an isochronous packet), then if the host completes the data packet and starts sending EOP before bit time 76, then the host can predict that the device will complete the handshake and start the EOP for the handshake on or before bit time 41. For a low-speed device, the 36 bit times from start of EOP from root to start of EOP from the device are low-speed bit times, which convert 1 to eight into full-speed bit times. Therefore, if the host completes the low-speed data packet by bit time 329, then the low-speed device can be predicted to complete the handshake before bit time 41

Note: if the host cannot accept a full-speed EOP as a valid end of a low-speed packet, then the low-speed EOP will need to complete before bit time 41, which will add 13 full-speed bit times to the low-speed handshake time.

As the host approaches the end of the frame, it must ensure that it does not require a device to send a handshake if that handshake can't be completed before bit time 41. The host expects to receive a handshake after any valid, non-isochronous data packet. Therefore, if the host is sending a non-isochronous data packet when it reaches bit time 76 (329 for low-speed), then the host should start an abnormal termination sequence to ensure that the device will not try to respond. This abnormal termination sequence consists of 7 consecutive bits of 1 followed by an EOP. The abnormal termination sequence is sent at the speed of the current packet.

If the host is preparing to send an IN token, it may not send the token if the predicted packet from the device would not complete by bit time 41. The maximum valid length of the response from the device is known by the host and should be used in the prediction calculation. For a full-speed packet, the maximum interval between the start of the IN token and the end of a data packet is:

token_length + (packet_length + header + CRC) * 7/6 + 18

Where *token_length* is 34 bit times, *packet_length* is the maximum number of data bits in the packet, *header* is eight bits of sync and eight bits of PID, and CRC is 16 bits. The 7/6 multiplier accounts for the absolute worst case bit-stuff on the packet and the 18 extra bits allow for worst case turn-around delay. For a low-speed device, the same calculation applies but the result must be multiplied by 8 to convert to full-speed bit times and an additional 20 full-speed bit times must be added to account for the low-speed prefix. This gives the maximum number of bit times between the start of the IN token and the end of the data packet, so the token cannot be sent if this number of bit times does not exist before the earliest EOF1 point (bit time 41). (E.g., take the results of the above calculation and add 41. If the number of bits left in the frame is less than this value, the token may not be sent.)

The host is allowed to use a more conservative algorithm than the one given above for deciding whether or not to start a transaction. The calculation might also include the time required for the host to send the handshake when one is required, as there is no benefit in starting a transfer if the handshake cannot be completed.

11.4 Internal Port

The internal port is the connection between the Hub Controller and the Hub Repeater. Besides conveying the serial data to/from the Hub Controller, the internal port is the source of certain resume signals. Figure 11-5 illustrates the internal port state machine; Table 11-2 defines the internal port signals and events.

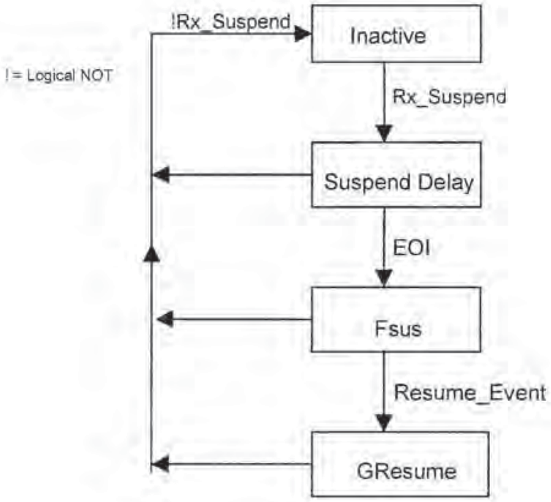


Figure 11-5. Internal Port State Machine

Table 11-2. Internal Port Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
EOI	Internal	End of timed interval
Rx_Suspend	Receiver	Receiver is in the Suspend state
Resume_Event	Hub Controller	A resume condition exists in the Hub Controller

11.4.1 Inactive

This state is entered whenever the Receiver is not in the Suspend state.

11.4.2 Suspend Delay

This state is entered from the Inactive state when the Receiver transitions to the Suspend state.

This is a timed state with a 2ms interval.

11.4.3 Full Suspend (Fsus)

This state is entered when the Suspend Delay interval expires.

11.4.4 Generate Resume (GResume)

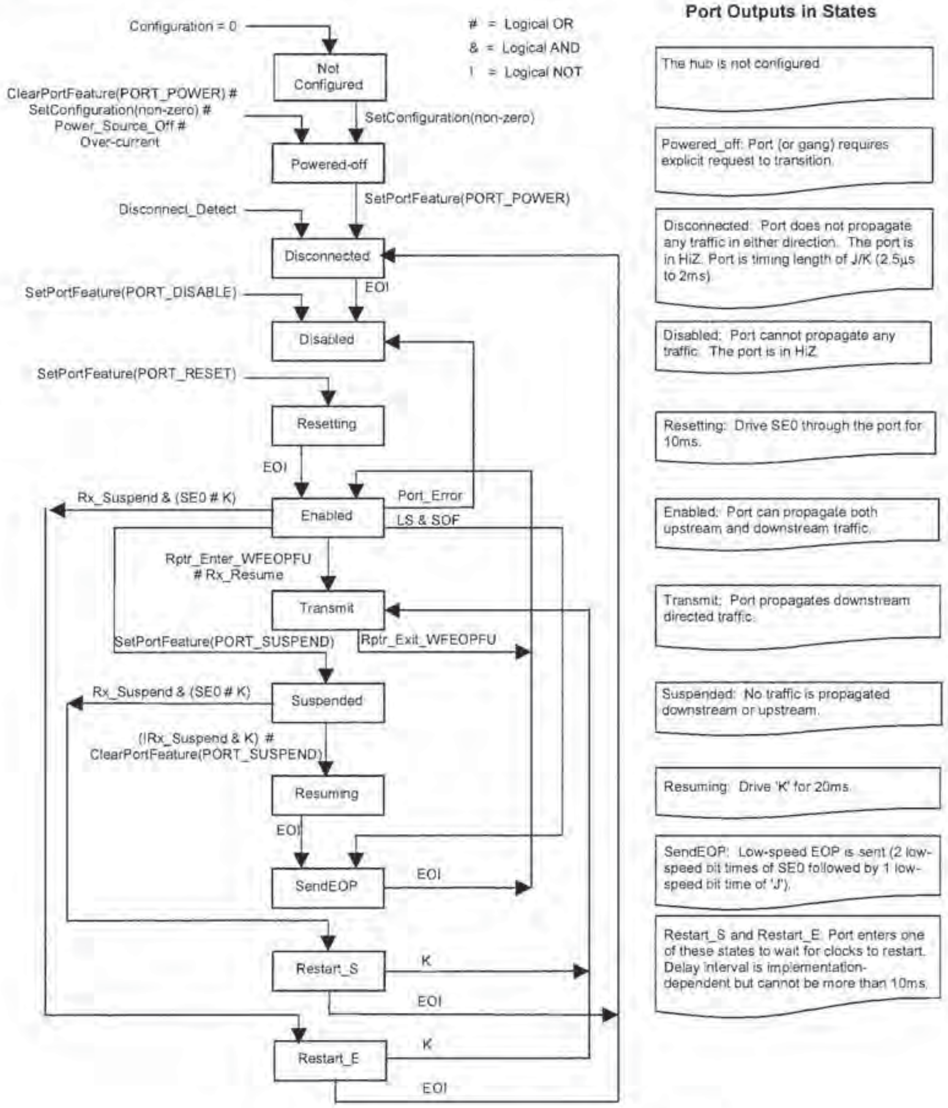
This state is entered from the Fsus state when a resume condition exists in the Hub Controller. A resume condition exists if the C_PORT_SUSPEND bit is set in any port or if the hub is enabled as a wakeup source and any bit is set in a Port Change field or the Hub Change field (as described in Table 11-14 and Table 11-10, respectively).

In this state, the internal port generates signaling to emulate an SOP_FD to the Hub Repeater.

11.5 Downstream Ports

The following sections provide a functional description of a state machine that exhibits the correct behavior for a downstream port on a hub.

Figure 11-6 is an illustration of the downstream port state machine. The events and signals are defined in Table 11-3. Each of the states is described in Section 11.5.1. In the diagram below, some of the entry conditions into states are shown without origin. These conditions have multiple origin states and the individual transitions lines are not shown so that the diagram can be simplified. The description of the entered state indicates from which states the transition is applicable.



The hub is not configured

Powered_off: Port (or gang) requires explicit request to transition.

Disconnected: Port does not propagate any traffic in either direction. The port is in HiZ. Port is timing length of J/K (2.5µs to 2ms).

Disabled: Port cannot propagate any traffic. The port is in HiZ.

Resetting: Drive SE0 through the port for 10ms.

Enabled: Port can propagate both upstream and downstream traffic.

Transmit: Port propagates downstream directed traffic.

Suspended: No traffic is propagated downstream or upstream.

Resuming: Drive 'K' for 20ms.

SendEOP: Low-speed EOP is sent (2 low-speed bit times of SE0 followed by 1 low-speed bit time of 'J').

Restart_S and Restart_E: Port enters one of these states to wait for clocks to restart. Delay interval is implementation-dependent but cannot be more than 10ms.

Figure 11-6. Downstream Hub Port State Machine

Table 11-3. Downstream Hub Port Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
Power_source_off	Implementation-dependent	Power to the port not available due to over-current or termination of source power (e.g., external power removed)
Over-current	Hub Controller	Over-current condition exists on the hub or the port
EOI	Internal	End of a timed interval or sequence
SE0	Internal	SE0 received on port
Disconnect_Detect	Internal	Long SE0 detected on port (See Section 11.5.2)
LS	Hub Controller	Low-speed device attached to this port
SOF	Hub Controller	SOF token received
J	Internal	'J' received on port
K	Internal	'K' received on port
Rx_Resume	Receiver	Upstream Receiver in Resume state
Rx_Suspend	Receiver	Upstream Receiver in Suspend state
Rptr_Exit_WFEOPFU	Hub Repeater	Hub Repeater exits the WFEOPFU state
Rptr_Enter_WFEOPFU	Hub Repeater	Hub Repeater enters the WFEOPFU state
Port_Error	Internal	Error condition detected (see Section 11.8.1)
Configuration = 0	Hub Controller	Hub controller's configuration value is zero

11.5.1 Downstream Port State Descriptions

11.5.1.1 Not Configured

A port transitions to and remains in this state whenever the value of the hub configuration is zero. While the port is in this state, the hub will drive an SE0 on the port (this behavior is optional on root hubs). No other active signaling takes place on the port when it is in this state.

11.5.1.2 Powered-off

This state is supported for all hubs.

A port transitions to this state in any of the following situations:

- From any state except Not Configured when the hub receives a ClearPortFeature(PORT_POWER) request for this port
- From any state when the hub receives a SetConfiguration() request with a configuration value other than zero
- From any state except Not Configured when power is lost to the port or an over-current condition exists.

A port will enter this state due to an over-current condition on another port if that over-current condition may have caused the power supplied to this port to drop below specified limits for port power (see Section 7.2.1.2.1 and Section 7.2.4.1).

If a hub was configured while the hub was self-powered, then if external power is lost the hub must place all ports in the Powered-off state. If the hub is configured while bus powered, then the hub need not change port status if the hub switched to externally applied power. However, if external power is subsequently lost, the hub must place ports in the Powered-off state.

In this state, the port's differential and single-ended transmitters and receivers are disabled.

Control of power to the port is covered in Section 11.11.

11.5.1.3 Disconnected

A port transitions to this state in any of the following situations:

- from the Powered-off state when the hub receives a SetPortFeature(PORT_POWER) request
- from any state except the Not Configured and Powered-off states when the port's disconnect timer times out
- from the Restart_S or Restart_E state at the end of the restart interval.

In the Disconnected state, the port's differential transmitter and receiver are disabled and only connection detection is possible.

This is a timed state. While in this state, the timer is reset as long as the port's signal lines are in the SE0 state. If another signaling state is detected, the timer starts. Unless the hub is suspended with clocks stopped, this timer's duration is 2.5μs to 2ms.

If the hub is suspended with its remote wakeup feature enabled then on a transition from the SE0 state on a Disconnected port the hub will start its clocks and time this event. The hub must be able to start its clocks and time this event within 12ms of the transition. If a hub does not have its remote wakeup feature enabled, then transitions on a port that is in the Disconnected state are ignored until the hub is resumed.

11.5.1.4 Disabled

A port transitions to this state in any of the following situations:

- From the Disconnected state when the timer expires indicating a connection is detected on the port
- From any but the Powered-off, Disconnected, or SenseSE0 states on receipt of a ClearPortFeature(PORT_ENABLE) request
- From the Enabled state when an error condition is detected on the port

A port in the Disabled state will not propagate signaling in either the upstream or the downstream direction. While in this state, the duration of any SE0 received on the port is timed.

11.5.1.5 Resetting

Unless it is in the Powered-off or Disconnected states, a port transitions to the Resetting state upon receipt of a SetPortFeature(PORT_RESET) request. The hub drives SE0 on the port during this timed interval. The duration of the Resetting state is nominally 10ms to 20ms (10ms is preferred).

11.5.1.6 Enabled

A port transitions to this state in any of the following situations:

- At the end of the Resetting state
- From the Transmit state when the Hub Repeater exits the WFEOPFU state
- From the Suspended state if the upstream Receiver is in the Suspend state when a 'K' is detected on the port
- At the end of the SendEOP state.

While in this state, the output of the port's differential receiver is available to the Hub Repeater so that 'J'-to-'K' transitions can establish upstream connectivity.

11.5.1.7 Transmit

For full- and low-speed ports this state is entered in either of the following situations:

- from the Enabled state if the upstream Receiver is in the Resume state
- immediately from the Restart_S or Restart_E state if a 'K' is detected on the port.

For a full-speed port, this state is entered from the Enabled state on the transition of the Hub Repeater to the WFEOPFU state. While in this state, the port will transmit the data that is received on the upstream port.

For a low-speed port, this state is entered from the Enabled state if a full-speed PRE PID is received on the upstream port. While in this state, the port will retransmit the data that is received on the upstream port (after proper inversion).

11.5.1.8 Suspended

A port enters the Suspended state from the Enabled state when it receives a SetPortFeature(PORT_SUSPEND) request. While a port is in the Suspended state, the port's differential transmitter is disabled.

An implementation is allowed to have a SE0 ‘noise’ filter for a port that is in the suspended state. This filter can time the length of SE0 and, if the length of the SE0 is shorter than 2.5μs, the port may remain in this state. However, this filter may not be used if the hub is suspended and the clocks are stopped. Rather, if the hub is suspended with its clocks stopped, a transition to SE0 on a suspended port must cause the port to immediately transition to the Restart_S state. This is to insure that the attached device is not reset and placed at the default address without having the hub disable the port.

11.5.1.9 Resuming

A port enters this state from the Suspended state in either of the following situations:

- If a 'K' is detected on the port and the Receiver is not in the Suspend state

- When a ClearPortFeature(PORT_SUSPEND) request is received.

This is a timed state with a nominal duration of 20ms (the interval may be longer under the conditions described in the note below). While in this state, the hub drives a 'K' on the port.

Note: a single timer is allowed to be used to time both the Resetting interval and the Resuming interval and that timer may be shared among multiple ports. When shared, the timer is reset when a port enters the Resuming state or the Resetting state. If shared, it may not be shared among more than ten ports as the cumulative delay could exceed the amount of time required to replace a device and a disconnect could be missed.

11.5.1.10 SendEOP

This state is entered from the Resuming state if the 20ms timer expires. It is also entered from the Enabled state when an SOF (or other FS token) is received and a low-speed devices is attached to this port. In this state, the hub will send a low-speed EOP (two low-speed bits times of SE0 followed by one low-speed bit times of J). At the end of the EOP, the state ends.

Since the transmitted EOP should be of fixed length, the SendEOP timer, if shared, should not be reset. If the hub implementation shares the SendEOP timing circuits between ports, then the Resuming state should not end until an SOF (or other FS token) has been received (see Section 11.8.4.1 for Keep-alive generation rules).

11.5.1.11 Restart_S/Restart_E

A port enters the Restart_S state from the Suspended state or enters the Restart_E state from the Enabled state when an SE0 or 'K' is seen at the port and the Receiver is in the Suspended state.

These states are needed to ensure that a transient SE0, which may be seen at the start of resume signaling, does not cause the port to be disabled.

In these states, the port continuously monitors the bus state and exits to the Transmit state immediately on seeing the K state. In this case, the port completes its transition to the Transmit state within 100µs after entering the Restart_S or Restart_E state. If the bus state is not 'K', the port transitions to the Disconnected state. This transition should happen within 10ms of entering the Restart_S or Restart_E state.

11.5.2 Disconnect Detect Timer

Each port is required to have a disconnect timer. This timer is used to constantly monitor the ports single-ended receivers to detect a disconnect event. The reason for constant monitoring is that a noise event on the bus can cause the attached device to detect a reset condition on the bus after 2.5µs of SE0 on the bus. If the hub does not place the port in the disconnect state before the device resets, then the device can be at in the Default Address state with the port enabled. This can cause systems errors that are very difficult to isolate and correct.

This timer should be reset whenever the D+ and D- lines on the port are not in the SE0 state or when the port is not in the Enabled, Suspended, or Disabled states. This timer may have a timeout that is a short as 1.994µs (2.0µs - 3000ppm) but should not be longer than 2.508µs (+3000 ppm). When this timer expires, it generates the Disconnect_Detect signal to the port state machine.

11.6 Upstream Port

The upstream port has four components: transmitter, transmitter state machine, receiver and receiver state machine. The transmitter and its state machine are the Transmitter, while the receiver and its state machine are the Receiver. Both the transmitter and receiver have differential and single-ended components. The differential transmitter and receiver can send/receive 'J' or 'K' to/from the bus while the single-ended components are used to send/receive SE0, suspend, and resume signaling. In this section, when it is necessary to differentiate the signals sent/received by the differential component of the transmitter/receiver from those of the single-ended components, DJ and DK will be used to denote the differential signal and SJ, SK and SE0 will be used for the single-ended signals.

It is assumed that the differential transmitter and receiver are turned off during suspend to minimize power consumption. The single-ended components are left on at all times, as they will take minimal power.

11.6.1 Receiver

The receiver state machine is responsible for monitoring the signaling state of the upstream connection to detect long-term signaling events such as bus reset, resume, and suspend. Figure 11-7 illustrates the state transition diagram. Table 11-4 defines the signals and events referenced in Figure 11-7.

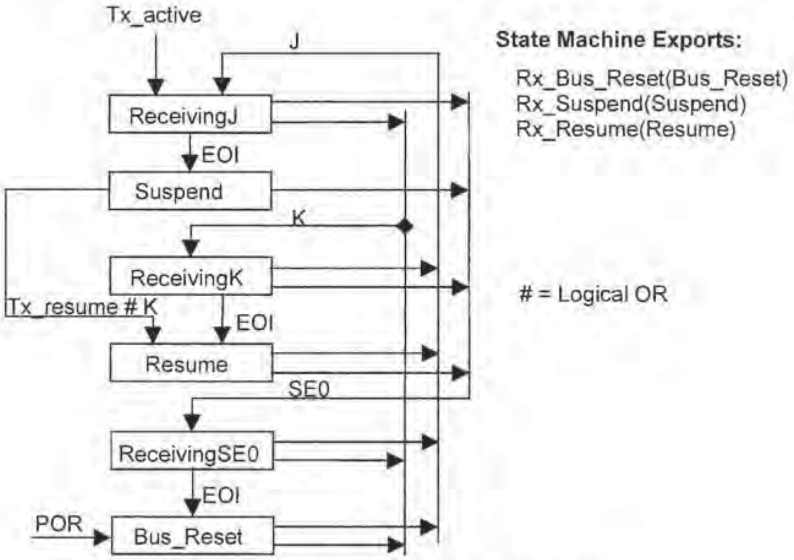


Figure 11-7. Upstream Port Receiver State Machine

Table 11-4. Upstream Hub Port Receiver Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
Tx_active	Transmitter	Transmitter in the Active state
J	Internal	Receiving a 'J' (IDLE) on the upstream port
EOI	Internal	End of timed interval
K	Internal	Receiving a 'K' on the upstream port
Tx_resume	Transmitter	Transmitter is in the Sresume state
SE0	Internal	Receiving an SE0 on the upstream port
POR	Implementation-dependent	Power_On_Reset

11.6.1.1 ReceivingJ

This state is entered from any state except the Suspend state if the receiver detects an SJ (or Idle) condition on the bus or while the Transmitter is in the Active state.

This is a timed state with an interval of 3ms. The timer is reset each time this state is entered.

The timer only advances if the Transmitter is in the Inactive state.

11.6.1.2 Suspend

This state is entered if the 3ms timer expires in the ReceivingJ state. When the Receiver enters this state, the Hub Controller starts a 2ms timer. If that timer expires while the Receiver is still in this state, then the Hub Controller is suspended. When the Hub Controller is suspended, it may generate resume signaling.

11.6.1.3 ReceivingK

This state is entered from any state except the Resume state when the receiver detects an SK condition on the bus and the Hub Repeater is in the WFSOP or WFSOPFU state. This is a timed state with a duration of 2.5μs to 100μs. The timer is reset each time this state starts.

11.6.1.4 Resume

This state is entered from the ReceivingK state when the timer expires.

This state is also entered from the Suspend state while the Transmitter is in the Sresume state or if there is a transition to the K state on the upstream port.

If the hub enters this state when its timing reference is not available, the hub may remain in this state until the hub's timing reference becomes stable. If this state is being held pending stabilization of the hub's clock, the Receiver should provide a K to the repeater for propagation to the downstream ports. When clocks are stable, the Receiver should repeat the incoming signals.

Note: constraints on hub behavior after reset require that the hub be able to start clocks and get them stable in less than 10ms.

11.6.1.5 ReceivingSE0

This state is entered from any state except Bus_Reset when the receiver detects an SE0 condition and the Hub Repeater is in the WFSOP or WFSOPFU state. This is a timed state. The minimum interval for this state is 2.5μs. The maximum depends on the hub but this interval must timeout early enough such that if the width of the SE0 on the upstream port is only 10ms, the Receiver will enter the Bus_Reset state with sufficient time remaining in the 10ms interval for the hub to complete its reset processing. Furthermore, if the hub is suspended when the Receiver enters this state, the hub must be able to start its clocks, time this interval, and complete its reset processing within 10ms. It is preferred that this interval be as long as possible given the constraints listed here. This will provide for the maximum immunity to noise on the upstream port and reduce the probability that the device will reset in the presence of noise before the upstream hub disables the port.

The timer is reset each time this state starts.

11.6.1.6 Bus_Reset

This state is entered from the ReceivingSE0 state when the timer expires. As long as the port continues to receive SE0, the Receiver will remain in this state.

This state is also entered while power-on-reset (POR) is being generated by the hub's local circuitry. The state machine cannot exit this state while POR is active.

11.6.2 Transmitter

This state machine is used to monitor the upstream port while the Hub Repeater has connectivity in the upstream direction. The purpose of this monitoring activity is to prevent propagation of erroneous indications in the upstream direction. In particular, this machine prevents babble and disconnect events on the downstream ports of this hub from propagating and causing this hub to be disabled or disconnected by the hub to which it is attached. Figure 11-8 is the transmitter state transition diagram. Table 11-5 defines the signals and events referenced in Figure 11-8.

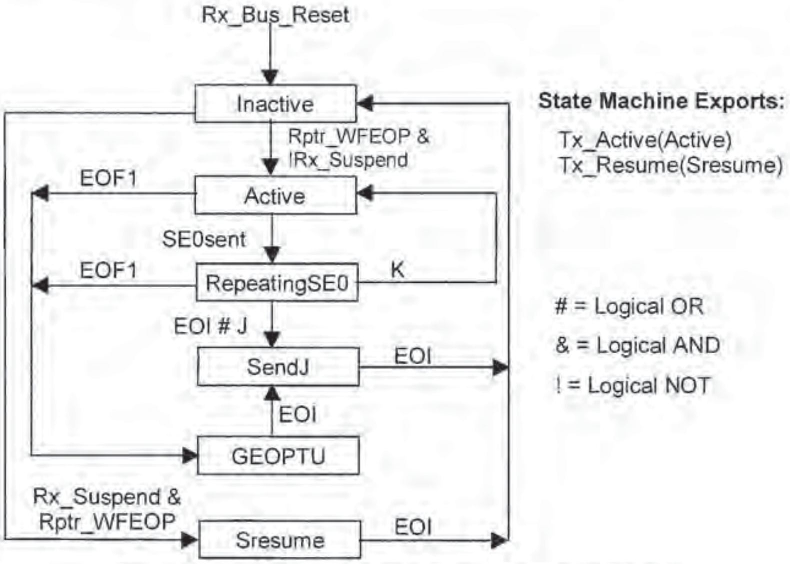


Figure 11-8. Upstream Hub Port Transmitter State Machine

Table 11-5. Upstream Hub Port Transmit Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
Rx_Bus_Reset	Receiver	Receiver is in the Bus_Reset state
EOF1	Frame Timer	Hub frame time has reached the EOF1 point or is between EOF1 and the end of the frame
J	Internal	Transmitter transitions to sending a 'J' and transmits a 'J'
Rptr_WFEOP	Hub Repeater	Hub Repeater is in the WFOEP state
K	Internal	Transmitter transmits a 'K'
SE0sent	Internal	At least one bit time of SE0 has been sent through the transmitter
Rx_Suspend	Receiver	Receiver is in Suspend state
EOI	Internal	End of timed interval

11.6.2.1 Inactive

This state is entered at the end of the SendJ state or while the Receiver is in the Bus_Reset state. This state is also entered at the end of the Sresume state. While the transmitter is in this state, both the differential and single-ended transmit circuits are disabled and placed in their high-impedance state.

11.6.2.2 Active

This state is entered from the Inactive state when the Hub Repeater transitions to the WFEOP state. This state is entered from the RepeatingSE0 state if the first transition after the SE0 is not to the J state. In this state, the data from a downstream port is repeated and transmitted on the upstream port.

11.6.2.3 RepeatingSE0

The port enters this state from the Active state when one bit time of SE0 has been sent on the upstream port. While in this state, the transmitter is still active and downstream signaling is repeated on the port. This is a timed state with a duration of 23 full-speed bit times.

11.6.2.4 SendJ

The port enters this state from the RepeatingSE0 state if either the bit timer reaches 23 or the repeated signaling changes from SE0 to 'J'. This state is also entered at the end of the GEOPTU state. This state lasts for one full-speed bit time. During this state, the hub drives an SJ on the port.

11.6.2.5 Generate End of Packet Towards Upstream Port (GEOPTU)

The port enters this state from the Active or RepeatingSEO state if the frame timer reaches the EOF1 point.

In this state, the port transmits SE0 for two full-speed bit times.

11.6.2.6 Send Resume (Sresume)

The port enters this state from the Inactive state if the Receiver is in the Suspend state and the Hub Repeater transitions to the WFEOP state. This indicates that a downstream device (or the port to the Hub Controller) has generated resume signaling, causing upstream connectivity to be established.

On entering this state, the hub will restart clocks if they had been turned off during the Suspend state. While in this state, the Transmitter will drive a 'K' on the upstream port. While the Transmitter is in this state, the Receiver is held in the Resume state. While in the Resume state, all downstream ports that are in the Enable state are placed in the Transmit state and the resume on this port is transmitted to those downstream ports.

The port stays in this state for at least 1ms but for no more than 15ms.

11.7 Hub Repeater

The Hub Repeater provides the following functions:

- Sets up and tears down connectivity on packet boundaries
- Ensures orderly entry into and out of the Suspend state, including proper handling of remote wakeups

The state machine in Figure 11-9 shows the states and transitions needed to implement the Hub Repeater. Table 11-6 defines the Hub Repeater signals and events. The following sections describe the states and the transitions.

Several of the state transitions below will occur when an EOP is detected. When such a transition is indicated, the transition does not occur until after the hub has repeated the SE0-to-'J' transition and has driven 'J' for at least one bit time (bit time is determined by the speed of the port.)

Some of the transitions are triggered by an SOP. Transitions of this type occur as soon as the hub detects the 'J'-to-'K' transition, ensuring that the initial edge of the SYNC field is preserved.

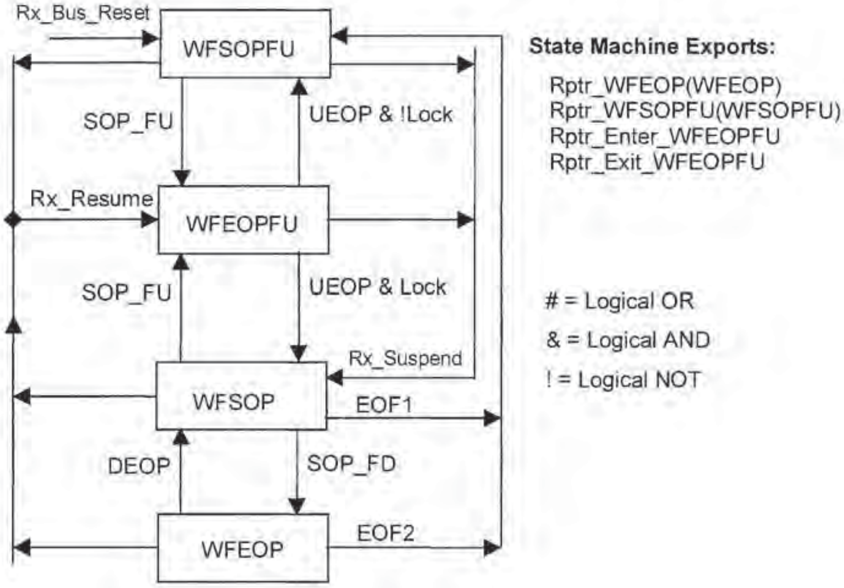


Figure 11-9. Hub Repeater State Machine

Table 11-6. Hub Repeater Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
Rx_Bus_Reset	Receiver	Receiver is in the Bus_Reset state
UEOP	Internal	EOP received from the upstream port
DEOP	Internal	Generated when the Transmitter enters the SendJ state
EOF1	Frame Timer	Frame timer is at the EOF1 point or between EOF1 and End-of-Frame
EOF2	Frame Timer	Frame timer is at the EOF2 point or between EOF2 and End-of-Frame
Lock	Frame Timer	Frame timer is locked
Rx_Suspend	Receiver	Receiver is in the Suspend state
Rx_Resume	Receiver	Receiver is in the Resume state
SOP_FD	Internal	SOP received from downstream port or Hub Controller. Generated on the transition from the Idle to K state on a port.
SOP_FU	Internal	SOP received from upstream port. Generated on the transition from the Idle to K state on the upstream port.

11.7.1 Wait for Start of Packet from Upstream Port (WFSOPFU)

This state is entered in either of the following situations:

- From any other state when the upstream Receiver is in the Bus_Reset state
- From the WFSOP state if the frame timer is at or has passed the EOF1 point
- From the WFEOP state at the EOF2 point.
- From the WFEOPFU if the frame timer is not synchronized (locked) when an EOP is received on the upstream port.

In this state, the hub is waiting for an SOP on the upstream port and transitions on downstream ports are ignored by the Hub Repeater. While the Hub Repeater is in this state, connectivity is not established.

This state is used during the End-of-Frame (past the EOF1 point) to ensure that the hub will be able to receive the SOF when it is sent by the host.

11.7.2 Wait for End of Packet from Upstream Port (WFEOPFU)

The hub enters this state if the hub is in the WFSOP or WFSOPFU state and an SOP is detected on the upstream port. The hub also enters this state from the WFSOP, WFSOPFU, or WFEOP states when the Receiver enters the Resume state.

While in this state, connectivity is established from the upstream port to all enabled downstream ports. Downstream ports that are in the Enabled state are placed in the Transmit state on the transition to this state.

11.7.3 Wait for Start of Packet (WFSOP)

This state is entered in any of the following situations:

- From the WFEOPstate when an EOP is detected from the downstream port
- From the WFEOPFU state if the frame timer is synchronized (locked) when an EOP is received from upstream
- From the WFSOPFU or WFEOPFU states when the upstream Receiver transitions to the Suspend state.

A hub in this state is waiting for an SOP on the upstream port or any downstream port that is in the Enabled state. While the Hub Repeater is in this state, connectivity is not established.

11.7.4 Wait for End of Packet (WFEOP)

This state is entered from the WFSOP state when an SOP is received from a downstream port in the Enabled state.

In this state, the hub has connectivity established in the upstream direction and the signaling received on an enabled downstream port is repeated and driven on the upstream port. The upstream Transmitter is placed in the Active state on the transition to this state.

If the Hub Repeater is in this state when the EOF2 point is reached, the downstream port for which connectivity is established is disabled as a babble port.

Note: the Transmitter will send an EOP at EOF1 but the Hub Repeater stays in this state until the device sends an EOP or the EOF2 point is reached.

11.8 Bus State Evaluation

A hub is required to evaluate the state of the connection on a port in order to make appropriate port state transitions. This section describes the appropriate times and means for several of these evaluations.

11.8.1 Port Error

A Port Error can occurs on a port that is in the Enabled state. A Port Error condition exists when::

- The hub is in the WFEOP state with connectivity established upstream from the port when the frame timer reaches the EOF2 point.
- At the EOF2 point the Hub Repeater is in the WFSOPFU state and there is other than an Idle/J state on the port.

If upstream-directed connectivity is established when the frame timer reaches the EOF1 point, the upstream Transmitter will generate a full-speed EOP to prevent the hub from being disabled by the upstream hub. The connected port is then disabled if it has not ended the packet and returned to the Idle state before the frame timer reaches the EOF2 point.

11.8.2 Speed Detection

The speed of an attached device is determined by the placement of a pull-up resistor on the device (see Section 7.1.5). When a device is attached, the hub is expected to detect the speed of the device by sensing the Bus Idle state. Due to connect and start-up transients, the hub may not be able to reliably determine the speed of the device until the transients have ended. The USB System Software is required to "debounce" the connection and provide a delay between the time a connection is detected and the device is used (see Section 7.1.7.1). At the end of the debounce interval, the device is expected to have placed its upstream port in the Idle state and be able to react to reset signaling. The USB System Software must send a SetPortFeature(PORT_RESET) request to the port to enable the port and make the attached device ready for use. This provides a convenient time for the hub to evaluate the speed of the device attached to the port. Speed detection can be done at the beginning of the port reset as the port leaves the Disabled state or at the end of the port reset between the end of the Resetting state and the start of the Enabled state.

If an implementation chooses to do speed evaluation on entry to the Resetting state from the Disabled state, it will set the PORT_LOW_SPEED status according to the condition of the D+ and D- lines at that time. (Note: if both D+ and D- are high at this time, the hub may stay in the Disabled state and set the C_PORT_ENABLE bit to indicate that the hub could not determine the speed of the device. Otherwise the hub should assume that the device is low-speed.) This determines the speed of the device and the Idle/J state for the port. The hub will then drive an SE0 for the duration of the Resetting state timer. At the end of the Resetting state, the hub will drive the lines to the J state that is appropriate for the speed of the attached device and transition to the Enabled state.

Note: because the SendEOP state also exits to the Enabled state, an implementation might exit the Resetting state to the SendEOP state without driving the 'J' and then let the SendEOP circuit complete the operation.

If an implementation chooses to do speed evaluation on exit from the Resetting state, then it will need an additional state called the Speed_eval state. At the end of the Resetting state, the hub will float the D+ and D- lines and allow the lines to settle to the Idle state appropriate for the attached device. At the end of the Speed_eval state, the hub will set the PORT_LOW_SPEED status as appropriate. The Speed_eval state must last for at least 2.5μs but no longer than 1ms. It is possible that the port will detect a disconnect condition during the speed evaluation. If so, the port transitions to the Disconnected state and will not enter the Enabled state.

11.8.3 Collision

If the Hub Repeater is in the WFEOP state and an SOP is detected on another enabled port, a Collision condition exists. There are two allowed behaviors for the hub in this instance.

The first, and preferred, behavior is to 'garble' the message so that the host can detect the problem. The hub garbles the message by transmitting a 'K' on the upstream port. This 'K' should persist until packet traffic from all downstream ports ends. The hub should use the last EOP to terminate the garbled packet. babble detection is enabled during this garbled message.

A second behavior is to block the second packet and, when the first message ends, return the hub to the WFSOPFU or WFSOP state as appropriate. If the second stream is still active, the hub may reestablish connectivity upstream. This method is not preferred, as it does not convey the problem to the host. Additionally, if the second stream causes the hub to reestablish upstream connectivity as the host is trying to establish downstream connectivity, additional packets can be lost and the host cannot properly associate the problem.

11.8.4 Full- versus Low-speed Behavior

The upstream connection of a hub must always be a full-speed connection. All downstream ports of a hub that are attached to USB connectors must be able to support both full-speed and low-speed devices. When low-speed data is sent or received through a hub's upstream connection, the signaling is full-speed even though the bit times are low-speed.

Full-speed signaling must not be transmitted to low-speed ports.

If a port is detected to be attached to a low-speed device, the hub port's output buffers are configured to operate at the slow slew rate (75-300ns), and the port will not propagate downstream-directed packets unless they are prefaced with a PRE PID. When a hub receives a PRE PID, it must enable the drivers on the enabled, low-speed ports within four bit times of receiving the last bit of the PID.

Note: when the driver is turned on, the upstream port will be in the 'J' state and the downstream ports should be driven to the same state.

Low-speed data follows the PID and is propagated to both low and full-speed devices. Hubs continue to propagate downstream signaling to all enabled ports until a downstream EOP is detected, at which time all output drivers are turned off.

Full-speed devices will not misinterpret low-speed traffic because no low-speed data pattern can generate a valid full-speed PID.

When a low-speed device transmits, it does not preface its data packet with a PRE PID. Hubs will propagate upstream-directed packets of any speed using full-speed signaling polarity and edge rates.

For both upstream and downstream low-speed data, the hub is responsible for inverting the polarity of the data before transmitting to/from a low-speed port.

Although a low-speed device will send a low-speed EOP to properly terminate a packet, a hub may truncate a low-speed packet at the EOF1 point with a full-speed EOP. Thus, hubs must always be able to tear down connectivity in response to a full-speed EOP regardless of the data rate of the packet.

Because of the slow transitions on low-speed ports, when the D+ and D- signal lines are switching between the 'J' and 'K', they may both be below 2.0V for a period of time that is longer than a full-speed bit time. A hub must ensure that these slow transitions do not result in termination of connectivity and must not result in an SE0 being sent upstream.

11.8.4.1 Low-speed Keep-alive

All hub ports to which low-speed devices are connected must generate a low-speed keep-alive strobe, generated at the beginning of the frame, which consists of a valid low-speed EOP (described in Section 7.1.13.2). The strobe must be generated at least once in each frame in which an SOF is received from the host. This strobe is used to prevent low-speed devices from suspending if there is no other low-speed traffic on the bus. The hub can generate the keep-alive on any valid full-speed token packet. The following rules for generation of a low-speed keep-alive must be adhered to:

- A keep-alive must minimally be derived from each SOF. It is recommended that a keep-alive be generated on any valid full-speed token.
- The keep-alive must start by the eighth bit after the PID of the full-speed token.

11.9 Suspend and Resume

Hubs must support suspend and resume both as a USB device and in terms of propagating suspend and resume signaling. Hubs support both global and selective suspend and resume. Global and selective suspend are defined in Section 7.1.7.4. Global suspend/resume refers to the entire bus being suspended or resumed without affecting any hub's downstream port states; selective suspend/resume refers to a downstream port of a hub being suspended or resumed without affecting the hub state. Global suspend/resume is implemented through the root port(s) at the host. Selective suspend/resume is implemented via requests to a hub. Device-initiated resume is called remote-wakeup (see Section 7.1.7.5).

Figure 11-10 shows the timing relationships for an example remote-wakeup sequence. This example illustrates a device initiating resume signaling through a suspended hub ('B') to an awake hub ('A'). Hub 'A' in this example times and completes the resume sequence and is the "Controlling Hub". The timings and events are defined in Section 7.1.7.5.

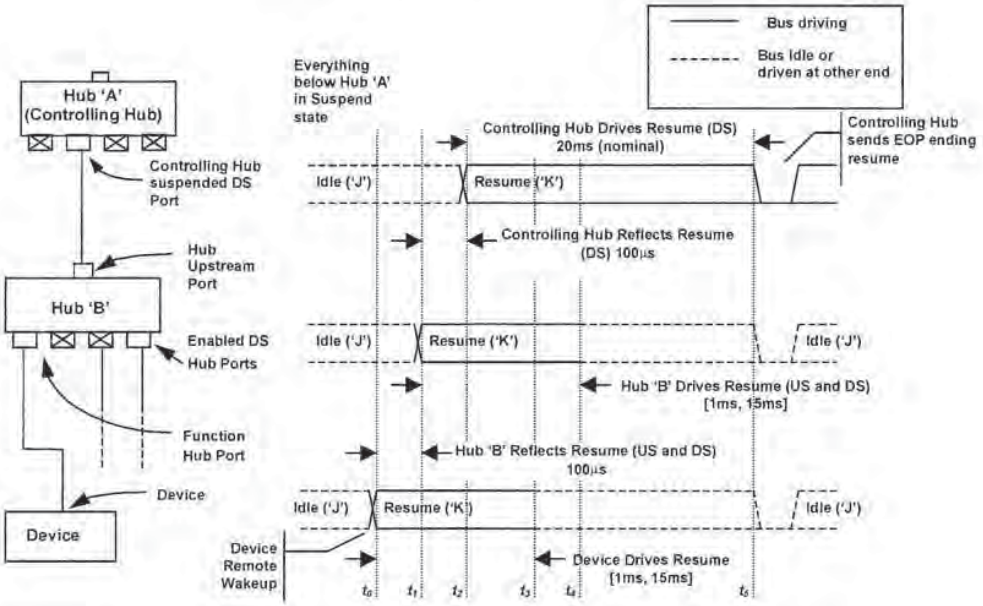


Figure 11-10. Example Remote-Wakeup Resume Signaling

Here is an explanation of what happens at each t_n:

t_0 Suspended device initiates remote-wakeup by driving a 'K' on the data lines.

t_1 Suspended hub 'B' detects the 'K' on its downstream port, wakes up enough within 100µs to reflect the resume upstream and down through all enabled ports.

t_2 Hub 'A' is not suspended (implication is that the port at which 'B' is attached is selectively suspended), detects the 'K' on the selectively suspended port where 'B' is attached, and reflects the resume signal back to 'B' within 100µs.

t_3 Device ceases driving 'K' upstream.

t_4 Hub 'B' ceases driving 'K' upstream and down all enabled ports and begins repeating upstream signaling to all enabled downstream ports.

t_5 Hub 'A' completes resume sequence, after appropriate timing interval, by driving a low-speed EOP downstream.

The hub reflection time is much smaller than the minimum duration a USB device will drive resume upstream. This relationship guarantees that resume will be propagated upstream and downstream without any gaps.

11.10 Hub Reset Behavior

The following sections describe hub reset behavior and its interactions with resume, attach detect, and power-on.

11.10.1 Hub Receiving Reset on Upstream Port

Reset signaling to a hub is defined only in the downstream direction, which is at the hub's upstream port. A hub may start its reset sequence if it detects 2.5µs or more of continuous SE0 signaling and must complete its reset sequence by the end of the reset signaling.

Note: the 2.5μs lower limit is set by a need to prevent low-speed EOP strobes from being interpreted as reset.

A suspended hub must interpret the start of reset as a wakeup event; it must be awake and have completed its reset sequence by the end of reset signaling.

After completion of the reset sequence, a hub is in the following state:

- Hub Controller default address is 0
- Hub status change bits are set to zero
- Hub Repeater is in the WFSOPFU state
- Transmitter is in the Inactive state
- Downstream ports are in the Not Configured state and SE0 driven on all downstream ports.

11.11 Hub Port Power Control

Self-powered hubs may have power switches that control delivery of power downstream ports but it is not required. Bus-powered hubs are required to have power switches. A hub with power switches can switch power to all ports as a group/gang, to each port individually, or have an arbitrary number of gangs of one or more ports .

A hub indicates whether or not it supports power switching by the setting of the Logical Power Switching Mode field in *wHubCharacteristics*.If a hub supports per-port power switching, then the power to a port is turned on when a SetPortFeature(PORT_POWER) request is received for the port. Port power is turned off when the port is in the Powered-off or Not Configured states. If a hub supports ganged power switching, then the power to all ports in a gang is turned on when any port in a gang receives a SetPortFeature(PORT_POWER) request. The power to a gang is not turned off unless all ports in a gang are in the Powered-off or Not Configured states. Note, the power to a port is not turned on by a SetPortFeature(PORT_POWER) if both C_HUB_LOCAL_POWER and Local Power Status (in *wHubStatus*) are set to 1B at the time when the request is executed and the PORT_POWER feature would be turned on.

Although a self-powered hub is not required to implement power switching, the hub must support the Powered-off state for all ports. Additionally, the hub must implement the *PortPwrCtrlMask* (all bits set to 1b) even though the hub has no power switches that can be controlled by the USB System Software.

Note: to ensure compatibility with previous versions of USB software, hubs must implement the Logical Power Switching Mode field in *wHubCharacteristics*. This is because some versions of SW will not use the SetPortFeature() request if the hub indicates in *wHubCharacteristics* that the port does not support port power switching. Otherwise, the Logical Power Switching Mode field in *wHubCharacteristics* would have become redundant as of this version of the specification.

The setting of the Logical Power Switching Mode for hubs with no power switches should reflect the manner in which over-current is reported. For example, if the hub reports over-current conditions on a per-port basis, then the Logical Power Switching Mode should be set to indicate that power switching is controlled on a per-port basis.

For a hub with no power switches, *bPwrOn2PwrGood* should be set to zero.

11.11.1 Multiple Gangs

A hub may implement any number of power and/or over-current gangs. A hub that implements more than one over-current and/or power switching gang must set both the Logical Power Switching Mode and the

Over-current Reporting Mode to indicate that power switching and over-current reporting are on a per port basis (these fields are in *wHubCharacteristics*.) Also, all bits in *PortPwrCtrlMask* must be set to 1b.

When an over-current condition occurs on an over-current protection device, the over-current is signaled on all ports that are protected by that device. When the over-current is signaled, all the ports in the group are placed in the Powered-off state, and the C_PORT_OVER-CURRENT field is set to 1B on all the ports. When port status is read from any port in the group, the PORT_OVER-CURRENT field will be set to 1b as long as the over-current condition exists. The C_PORT_OVER-CURRENT field must be cleared in each port individually.

When multiple ports share a power switch, setting PORT_POWER on any port in the group will cause the power to all ports in the group to turn on. It will not, however, cause the other ports in that group to leave the Powered-off state. When all the ports in a group are in the Powered-off state or the hub is not configured, the power to the ports is turned off.

If a hub implements both power switching and over-current, it is not necessary for the over-current groups to be the same as the power switching groups.

If an over-current condition occurs and power switches are present, then all power switches associated with an over-current protection circuit must be turned off. If multiple over-current protection devices are associate with a single power switch then that switch will be turned off when any of the over-current protection circuits indicates an over-current condition.

11.12 Hub I/O Buffer Requirements

All hub ports must be able to detect and generate all the bus signaling states described in Table 7-1. This requires that hub be able to independently drive and monitor the D+ and D- outputs on each of its ports. Each hub port must have single-ended receivers and transmitters on the D+ and D- lines as well as a differential receiver and transmitter. Details on voltage levels and drive requirements appear in Chapter 7.

11.12.1 Pull-up and Pull-down Resistors

Hubs, and the devices to which they connect, use a combination of pull-up and pull-down resistors to control D+ and D- in the absence of their being actively driven. These resistors establish voltage levels used to signal connect and disconnect and maintain the data lines at their idle values when not being actively driven. Each hub downstream port requires a pull-down resistor (R_{pd}) on each data line; the hub upstream port requires a pull-up resistor (R_{pu}) on its D+ line. Values for R_{pu} and R_{pd} appear in Chapter 7.

11.12.2 Edge Rate Control

Downstream hub ports must support transmission and reception of both low-speed and full-speed edge rates. The respective signaling specifications are given in Chapter 7. Edge rate on a downstream port must be selectable, based upon whether a downstream device was detected as being full-speed or low-speed. The hub upstream port always uses full-speed signaling, and its output buffers always operate with full-speed edge rates and signal polarities.

11.13 Hub Controller

The Hub Controller is logically organized as shown in Figure 11-11.

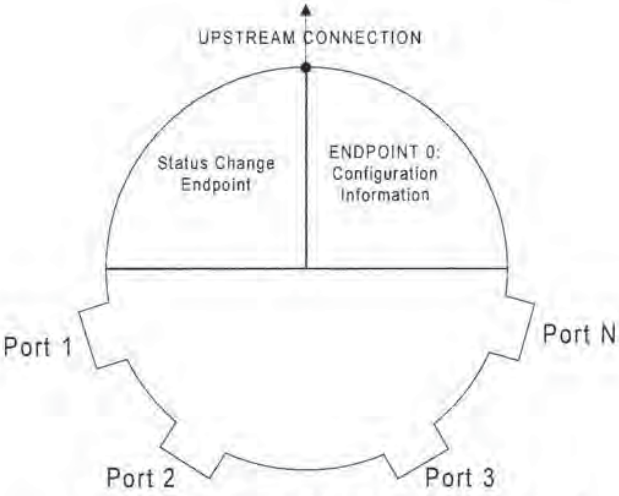


Figure 11-11. Example Hub Controller Organization

11.13.1 Endpoint Organization

The Hub Class defines one additional endpoint beyond Default Control Pipe, which is required for all devices: the Status Change endpoint. The host system receives port and hub status change notifications through the Status Change endpoint. The Status Change endpoint is an interrupt endpoint. If no hub or port status change bits are set, then the hub returns an NAK when the Status Change endpoint is polled. When a status change bit is set, the hub responds with data, as shown in Section 11.13.4, indicating the entity (hub or port) with a change bit set. The USB System Software can use this data to determine which status registers to access in order to determine the exact cause of the status change interrupt.

11.13.2 Hub Information Architecture and Operation

Figure 11-12 shows how status, status change, and control information relate to device states. Hub descriptors and Hub/Port Status and Control are accessible through the Default Control Pipe. The Hub descriptors may be read at any time. When a hub detects a change on a port or when the hub changes its own state, the Status Change endpoint transfers data to the host in the form specified in Section 11.13.4.

Hub or port status change bits can be set because of hardware or software events. When set, these bits remain set until cleared directly by the USB System Software through a ClearPortFeature() request or by a hub reset. While a change bit is set, the hub continues to report a status change when polled until all change bits have been cleared by the USB System Software.

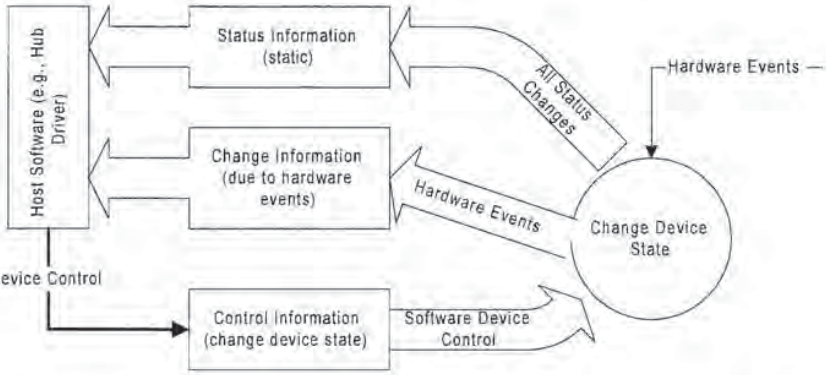


Figure 11-12. Relationship of Status, Status Change, and Control Information to Device States

The USB System Software uses the interrupt pipe associated with the Status Change endpoint to detect changes in hub and port status.

11.13.3 Port Change Information Processing

Hubs report a port's status through port commands on a per-port basis. The USB System Software acknowledges a port change by clearing the change state corresponding to the status change reported by the hub. The acknowledgment clears the change state for that port so future data transfers to the Status Change endpoint do not report the previous event. This allows the process to repeat for further changes (see Figure 11-13.)

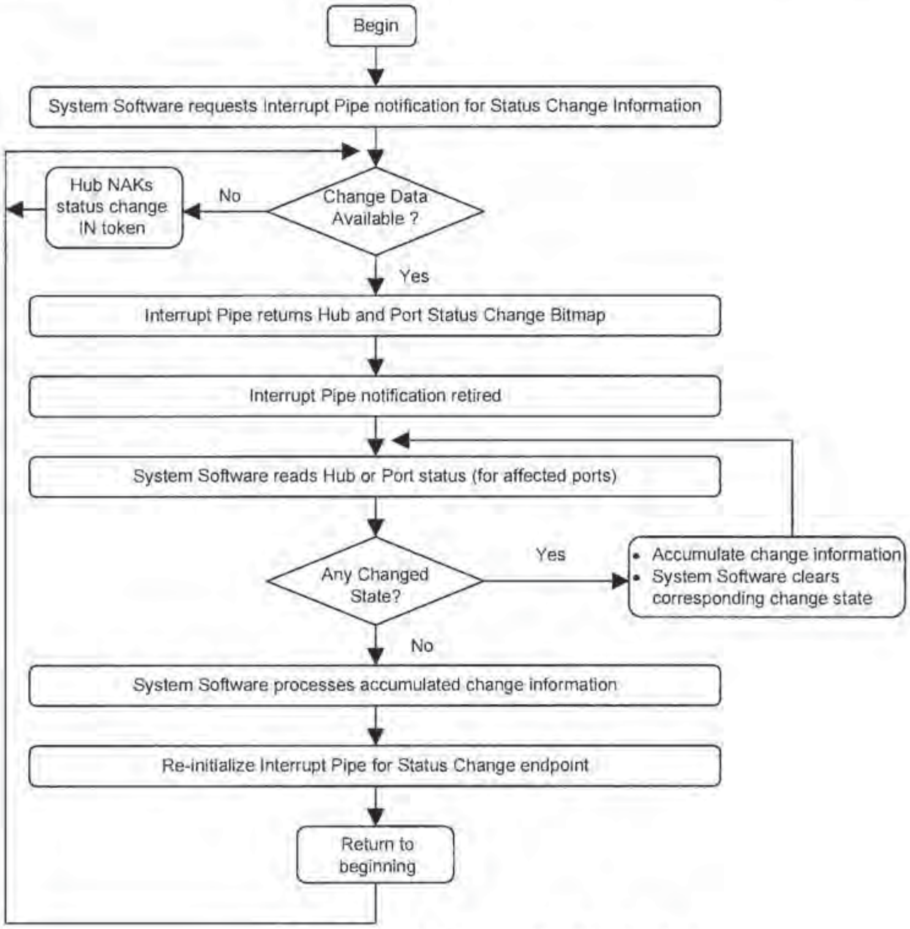


Figure 11-13. Port Status Handling Method

11.13.4 Hub and Port Status Change Bitmap

The Hub and Port Status Change Bitmap, shown in Figure 11-14, indicates whether the hub or a port has experienced a status change. This bitmap also indicates which port(s) have had a change in status. The hub returns this value on the Status Change endpoint. Hubs report this value in byte-increments. That is, if a hub has six ports, it returns a byte quantity and reports a zero in the invalid port number field locations. the USB System Software is aware of the number of ports on a hub (this is reported in the hub descriptor) and decodes the Hub and Port Status Change Bitmap accordingly. The hub reports any changes in hub status in bit zero of the Hub and Port Status Change Bitmap.

The Hub and Port Status Change Bitmap size varies from a minimum size of one byte. Hubs report only as many bits as there are ports on the hub, subject to the byte-granularity requirement (i.e., round up to the nearest byte).

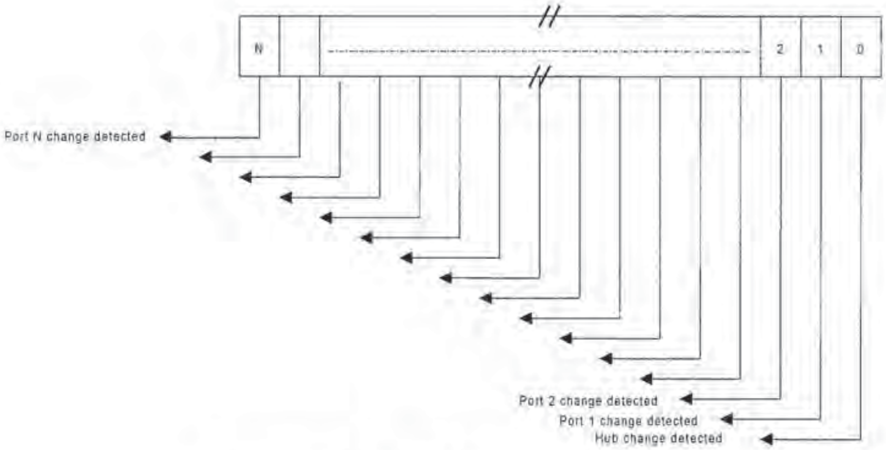


Figure 11-14. Hub and Port Status Change Bitmap

Any time the Status Change endpoint is polled by the host controller and any of the Status Changed bits are non-zero, the Hub and Port Status Change Bitmap is returned. Figure 11-15 shows an example creation mechanism for hub and port change bits.

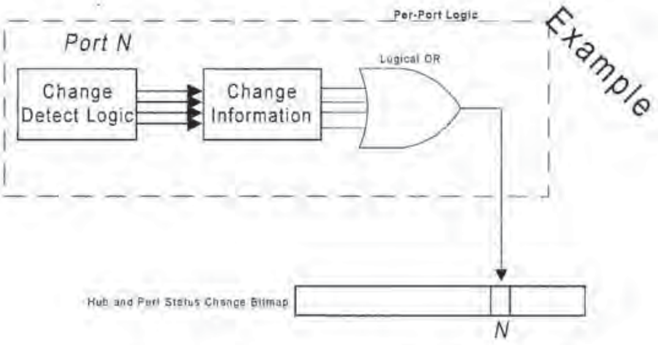


Figure 11-15. Example Hub and Port Change Bit Sampling

11.13.5 Over-current Reporting and Recovery

USB devices must be designed to meet applicable safety standards. Usually, this will mean that a self-powered hub implement current limiting on its downstream ports. If an over-current condition occurs, it causes a status and state change in one or more ports. This change is reported to the USB System Software so that it can take corrective action.

A hub may be designed to report over-current as either a port or a hub event. The hub descriptor field *wHubCharacteristics* is used to indicate the reporting capabilities of a particular hub (see Section 11.15.2). The over-current status bit in the hub or port status field indicates the state of the over-current detection when the status is returned. The over-current status change bit in the Hub or Port Change field indicates if the over-current status has changed.

When a hub experiences an over-current condition, it must place all affected ports in the Powered-off state. If a hub has per-port power switching and per-port current limiting, an over-current on one port may still cause the power on another port to fall below specified minimums. In this case, the affected port is placed in the Powered-off state and C_PORT_OVER_CURRENT is set for the port, but PORT_OVER_CURRENT is not set. If the hub has over-current detection on a hub basis, then an over-current condition on the hub will cause all ports to enter the Powered-off state. However, in this case, neither C_PORT_OVER_CURRENT nor PORT_OVER_CURRENT is set for the affected ports.

Host recovery actions for an over-current event should include the following:

1. Host gets change notification from hub with over-current event.
2. Host extracts appropriate hub or port change information (depending on the information in the change bitmap).
3. Host waits for over-current status bit to be cleared to 0.
4. Host cycles power on to all of the necessary ports (e.g., issues a SetPortFeature(PORT_POWER) request for each port).
5. Host re-enumerates all affected ports.

11.14 Hub Configuration

Hubs are configured through the standard USB device configuration commands. A hub that is not configured behaves like any other device that is not configured with respect to power requirements and addressing. If a hub implements power switching, no power is provided to the downstream ports while the hub is not configured. Configuring a hub enables the Status Change endpoint. The USB System Software may then issue commands to the hub to switch port power on and off at appropriate times.

The USB System Software examines hub descriptor information to determine the hub's characteristics. By examining the hub's characteristics, the USB System Software ensures that illegal power topologies are not allowed by not powering on the hub's ports if doing so would violate the USB power topology. The device status and configuration information can be used to determine whether the hub should be used as a bus or self-powered device. Table 11-7 summarizes the information and how it can be used to determine the current power requirements of the hub.

Table 11-7. Hub Power Operating Mode Summary

Configuration Descriptor		Hub Device Status (Self Power)	Explanation
MaxPower	bmAttributes (Self Powered)		
0	0	N/A	N/A This is an illegal set of information.
0	1	0	N/A A device which is only self-powered, but does not have local power cannot connect to the Bus and communicate.

Table 11-7. Hub Power Operating Mode Summary (Continued)

Configuration Descriptor		Hub Device Status (Self Power)	Explanation
MaxPower	bmAttributes (Self Powered)		
0	1	1	Self-powered only hub and local power supply is good. Hub status also indicates local power good, see Section 11.16.2.5. Hub functionality is valid anywhere depth restriction is not violated.
> 0	0	N/A	Bus-powered only hub. Downstream ports may not be powered unless allowed in current topology. Hub device status reporting Self Powered is meaningless in combination of a zeroed *bmAttributes.Self-Powered*.
> 0	1	0	This hub is capable of both self- and bus-powered operating modes. It is currently only available as a bus-powered hub.
> 0	1	1	This hub is capable of both self- and bus-powered operating modes. It is currently available as a self-powered hub.

A self-powered hub has a local power supply, but may optionally draw one unit load from its upstream connection. This allows the interface to function when local power is not available (see Section 7.2.1.2). When local power is removed (either a hub-wide over-current condition or local supply is off), a hub of this type remains in the Configured state but transitions all ports (whether removable or non-removable) to the Powered-off state. While local power is off, all port status and change information read as zero and all SetPortFeature() requests are ignored (request is treated as a no-operation). The hub will use the Status Change endpoint to notify the USB System Software of the hub event (see Section 11.16.2.5 for details on hub status).

The *MaxPower* field in the configuration descriptor is used to report to the system the maximum power the hub will draw from VBUS when the configuration is selected. For bus-powered hubs, the reported value must not include the power for any of external downstream ports. The external devices attaching to the hub will report their individual power requirements.

A compound device may power both the hub electronics and the permanently attached devices from VBUS. The entire load may be reported in the hubs' configuration descriptor with the permanently attached devices each reporting self-powered, with zero *MaxPower* in their respective configuration descriptors.

11.15 Descriptors

Hub descriptors are derived from the general USB device framework. Hub descriptors define a hub device and the ports on that hub. The host accesses hub descriptors through the hub's default pipe.

The USB specification (refer to Chapter 9) defines the following descriptors:

- Device
- Configuration
- Interface
- Endpoint
- String (optional).

The hub class defines additional descriptors (refer to Section 11.15.2). In addition, vendor-specific descriptors are allowed in the USB device framework. Hubs support standard USB device commands as defined in Chapter 9.

11.15.1 Standard Descriptors

The hub class pre-defines certain fields in standard USB descriptors. Other fields are either implementation-dependent or not applicable to this class.

Note: for the descriptors and fields shown below, the bits in a field are organized in a little-endian fashion; that is, bit location 0 is the least significant bit and bit location 7 is the most significant bit of a byte value.

Device Descriptor

bDeviceClass = HUB_CLASSCODE (09H)
bDeviceSubClass = 0

Interface Descriptor

bNumEndpoints = 1
bInterfaceClass = HUB_CLASSCODE (09H)
bInterfaceSubClass = 0
bInterfaceProtocol = 0

Configuration Descriptor

MaxPower = The maximum amount of bus power the hub will consume in this configuration

Endpoint Descriptor (for Status Change Endpoint)

bEndpointAddress = Implementation-dependent; Bit 7: Direction = In(1)
wMaxPacketSize = Implementation-dependent
bmAttributes = Transfer Type = Interrupt (00000111B)
bInterval = FFH (Maximum allowable interval)

The hub class driver retrieves a device configuration from the USB System Software using the GetDescriptor() device request. The only endpoint descriptor that is returned by the GetDescriptor() request is the Status Change endpoint descriptor.

11.15.2 Class-specific Descriptors

11.15.2.1 Hub Descriptor

Table 11-8 outlines the various fields contained by the hub descriptor.

Table 11-8. Hub Descriptor

Offset	Field	Size	Description
0	*bDescLength*	1	Number of bytes in this descriptor, including this byte
1	*bDescriptorType*	1	Descriptor Type, value: 29H for hub descriptor
2	*bNbrPorts*	1	Number of downstream ports that this hub supports
3	*wHubCharacteristics*		D1...D0: Logical Power Switching Mode 00: Ganged power switching (all ports' power at once) 01: Individual port power switching 1X: Reserved. Used only on 1.0 compliant hubs that implement no power switching. D2: Identifies a Compound Device 0: Hub is not part of a compound device 1: Hub is part of a compound device D4...D3: Over-current Protection Mode 00: Global Over-current Protection. The hub reports over-current as a summation of all ports' current draw, without a breakdown of individual port over-current status. 01: Individual Port Over-current Protection. The hub reports over-current on a per-port basis. Each port has an over-current indicator. 1X: No Over-current Protection. This option is allowed only for bus-powered hubs that do not implement over-current protection. D15...D5: Reserved
5	*bPwrOn2PwrGood*	1	Time (in 2ms intervals) from the time the power-on sequence begins on a port until power is good on that port. The USB System Software uses this value to determine how long to wait before accessing a powered-on port.
6	*bHubContrCurrent*	1	Maximum current requirements of the Hub Controller electronics in mA.

Table 11-8. Hub Descriptor (Continued)

Offset	Field	Size	Description
7	*DeviceRemovable*	Variable, depending on number of ports on hub	Indicates if a port has a removable device attached. This field is reported on byte-granularity. Within a byte, if no port exists for a given location, the field representing the port characteristics returns 0. Bit value definition: 0B - Device is removable 1B - Device is non-removable This is a bitmap corresponding to the individual ports on the hub: Bit 0: Reserved for future use Bit 1: Port 1 Bit 2: Port 2 .... Bit *n*: Port *n* (implementation-dependent, up to a maximum of 255 ports).
Variable	*PortPwrCtrlMask*	Variable, depending on number of ports on hub	This field exists for reasons of compatibility with software written for 1.0 compliant devices. All bits in this field should be set to 1B. This field has one bit for each port on the hub with additional pad bits, if necessary, to make the number of bits in the field an integer multiple of 8.

11.16 Requests

11.16.1 Standard Requests

Hubs have tighter constraints on request processing timing than specified in Section 9.2.6 for standard devices because they are crucial to the 'time to availability' of all devices attached to USB. The worst case request timing requirements are listed below (apply to both Standard and Hub Class requests):.

1. Completion time for requests with no data stage: 50 ms
2. Completion times for standard requests with data stage(s)
    - Time from setup packet to first data stage: 50 ms
    - Time between each subsequent data stage: 50 ms
    - Time between last data stage and status stage: 50 ms

As hubs play such a crucial role in bus enumeration, it is recommended that hubs average response times be less than 5ms for all requests.

Table 11-9 outlines the various standard device requests.

Table 11-9. Hub Responses to Standard Device Requests

bRequest	Hub Response
CLEAR_FEATURE	Standard
GET_CONFIGURATION	Standard
GET_DESCRIPTOR	Standard
GET_INTERFACE	Undefined. Hubs are allowed to support only one interface
GET_STATUS	Standard
SET_ADDRESS	Standard
SET_CONFIGURATION	Standard
SET_DESCRIPTOR	Optional
SET_FEATURE	Standard
SET_INTERFACE	Undefined. Hubs are allowed to support only one interface
SYNCH_FRAME	Undefined. Hubs are not allowed to have isochronous endpoints

Optional requests that are not implemented shall return a STALL in the Data stage or Status stage of the request.

11.16.2 Class-specific Requests

The hub class defines requests to which hubs respond, as outlined in Table 11-10. Table 11-11 defines the hub class request codes. All requests in the table below except for GetBusState() and SetHubDescriptor() are mandatory.

Table 11-10. Hub Class Requests

Request	bmRequestType	bRequest	wValue	wIndex	wLength	Data
ClearHubFeature	00100000B	CLEAR_ FEATURE	Feature Selector	Zero	Zero	None
ClearPortFeature	00100011B	CLEAR_ FEATURE	Feature Selector	Port	Zero	None
GetBusState	10100011B	GET_ STATE	Zero	Port	One	Per-Port Bus State
GetHubDescriptor	10100000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
GetHubStatus	10100000B	GET_ STATUS	Zero	Zero	Four	Hub Status and Change Indicators
GetPortStatus	10100011B	GET_ STATUS	Zero	Port	Four	Port Status and Change Indicators
SetHubDescriptor	00100000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
SetHubFeature	00100000B	SET_ FEATURE	Feature Selector	Zero	Zero	None
SetPortFeature	00100011B	SET_ FEATURE	Feature Selector	Port	Zero	None

Table 11-11. Hub Class Request Codes

bRequest	Value
GET_ STATUS	0
CLEAR_ FEATURE	1
GET_STATE	2
SET_ FEATURE	3
Reserved for future use	*4-5*
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7

Table 11-12 gives the valid feature selectors for the hub class. See Section 11.16.2.5 and Section 11.16.2.6 for a description of the features.

Table 11-12. Hub Class Feature Selectors

	Recipient	Value
C_HUB_LOCAL_POWER	Hub	0
C_HUB_OVER_CURRENT	Hub	1
PORT_CONNECTION	Port	0
PORT_ENABLE	Port	1
PORT_SUSPEND	Port	2
PORT_OVER_CURRENT	Port	3
PORT_RESET	Port	4
PORT_POWER	Port	8
PORT_LOW_SPEED	Port	9
C_PORT_CONNECTION	Port	16
C_PORT_ENABLE	Port	17
C_PORT_SUSPEND	Port	18
C_PORT_OVER_CURRENT	Port	19
C_PORT_RESET	Port	20

11.16.2.1 Clear Hub Feature

This request resets a value reported in the hub status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100000B	CLEAR_ FEATURE	Feature Selector	Zero	Zero	None

Clearing a feature disables that feature; refer to Table 11-12 for the feature selector definitions that apply to the hub as a recipient. If the feature selector is associated with a change indicator, clearing that indicator acknowledges the change. This request format is used to clear either the C_HUB_LOCAL_POWER or C_HUB_OVER_CURRENT features.

It is a Request Error if *wValue* is not a feature selector listed in Table 11-12 or if *wIndex* or *wLength* are not as specified above.

If the hub is not configured, the hub's response to this request is undefined.

11.16.2.2 Clear Port Feature

This request resets a value reported in the port status.

bmRequestType	Brequest	wValue	wIndex	wLength	Data
00100011B	CLEAR_ FEATURE	Feature Selector	Port	Zero	None

The port number must be a valid port number for that hub, greater than zero.

Clearing a feature disables that feature or starts a process associated with the feature; refer to Table 11-12 for the feature selector definitions. If the feature selector is associated with a change indicator, clearing that indicator acknowledges the change. This request format is used to clear the following features:

- PORT_ENABLE
- PORT_SUSPEND
- PORT_POWER
- C_PORT_CONNECTION
- C_PORT_RESET
- C_PORT_ENABLE
- C_PORT_SUSPEND
- C_PORT_OVER_CURRENT.

Clearing the PORT_SUSPEND feature causes a host-initiated resume on the specified port. If the port is not in the Suspended state, the hub should treat this request as a functional no-operation.

Clearing the PORT_ENABLE feature causes the port to beplaced in the Disabled state. If the port is in the Powered-off state, the hub should treat this request as a functional no-operation.

Clearing the PORT_POWER feature causes the port to be placed in the Powered-off state and may, subject to the constraints due to the hub's method of power switching, result in power being removed from the port. Refer to Section 11.11 on rules for how this request is used with ports that are gang-powered.

It is a Request Error if *wValue* is not a feature selector listed in Table 11-12, if *wIndex* specifies a port that doesn't exist, or if *wLength* is not as specified above. It is not an error for this request to try to clear a feature that is already cleared (hub should treat as a function no-operation).

If the hub is not configured, the hub's response to this request is undefined.

11.16.2.3 Get Bus State

This is an optional per-port diagnostic request that returns the bus state value, as sampled at the last EOF2 point.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100011B	GET_ STATE	Zero	Port	One	Per-Port Bus State

The port number must be a valid port number for that hub, greater than zero. If an invalid port number is specified or if *wValue* or *wLength* are not as specified above, then the hub shall return a STALL in the Data stage of the request (aborting the Status stage).

Hubs may implement an optional diagnostic aid to facilitate system debug. Hubs implement this aid through this optional request. This diagnostic feature provides a glimpse of the USB bus state as sampled at the last EOF2 sample point.

Hubs that implement this diagnostic feature should store the bus state at each EOF2 state in preparation for a potential request in the following USB frame.

The data returned is bitmapped in the following manner:

Bit 0:	The value of the D- signal
Bit 1:	The value of the D+ signal
Bits 2-7:	Reserved for future use and are reset to zero.

The hub must be able to return the bus state in the Data stage transaction within the frame in which the request was received. If the hub does not receive ACK for the data packet, the device is not required to return the same data packet if the host continues with the Data stage. Rather, the hub will always return the bus state at the immediately prior EOF2 sample point along with the DATA0 PID.

Hubs that do not implement this request shall return a STALL in the Data stage of the request (aborting the Status stage).

If the hub is not configured, the hub's response to this request is undefined.

11.16.2.4 Get Hub Descriptor

This request returns the hub descriptor.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero	Descriptor Length	Descriptor

The GetDescriptor() request for the hub class descriptor follows the same usage model as that of the standard GetDescriptor() request (refer to Chapter 9). The standard hub descriptor is denoted by using the value *bDescriptorType* defined in Section 11.15.2.1. All hubs are required to implement one hub descriptor, with descriptor index zero.

If *wLength* is larger than the actual length of the descriptor, then only the actual length is returned. If *wLength* is less than the actual length of the descriptor, then only the first *wLength* bytes of the descriptor are returned; this is not considered an error even if *wLength* is zero.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

If the hub is not configured, the hub's response to this request is undefined.

11.16.2.5 Get Hub Status

This request returns the current hub status and the states that have changed since the previous acknowledgment.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100000B	GET_ STATUS	Zero	Zero	Four	Hub Status and Change Indicators

The first word of data contains *wHubStatus* (refer to Table 11-13). The second word of data contains *wHubChange* (refer to Table 11-14).

It is a Request Error if *wValue*, *wIndex*, or *wLength* are other than as specified above.

If the hub is not configured, the hub's response to this request is undefined.

Table 11-13. Hub Status Field, *wHubStatus*

Bit	Description
0	**Local Power Source:** This is the source of the local power supply. This field indicates whether hub power (for other than the SIE) is being provided by an external source or from the USB. . This field allows the USB System Software to determine the amount of power available from a hub to downstream devices. 0 = Local power supply good 1 = Local power supply lost (inactive)
1	**Over-current Indicator:** If the hub supports over-current reporting on a hub basis, this field indicates that the sum of all the ports' current has exceeded the specified maximum and all ports have been places in the Powered-off state. If the hub reports over-current on a per-port basis or has no over-current detection capabilities, this field is always zero. For more details on over-current protection, see Section 7.2.1.2.1. 0 = No over-current condition currently exists 1 = A hub over-current condition exists
2-15	**Reserved** These bits return 0 when read.

There are no defined feature selector values for these status bits and they can neither be set nor cleared by the USB System Software.

Table 11-14. Hub Change Field, *wHubChange*

Bit	Description
0	**Local Power Status Change:** (C_HUB_LOCAL_POWER) This field indicates that a change has occurred in the hub's Local Power Source field in *wHubStatus*. This field is initialized to zero when the hub receives a bus reset. 0 = No change has occurred to Local Power Status 1 = Local Power Status has changed
1	**Over-Current Indicator Change**: (C_HUB_OVER_CURRENT) This field indicates if a change has occurred in the Over-Current field in *wHubStatus*. This field is initialized to zero when the hub receives a bus reset. 0 = No change has occurred to the Over-Current Indicator 1 = Over-Current Indicator has changed
2-15	**Reserved** These bits return 0 when read.

Hubs may allow setting of these change bits with SetHubFeature() requests in order to support diagnostics. If the hub does not support setting of these bits, it should either treat the SetHubFeature() request as a Request Error or as a functional no-operation. When set, these bits may be cleared by a ClearHubFeature() request. A request to set a feature that is already set or to clear a feature that is already clear has no effect and the hub will not fail the request.

11.16.2.6 Get Port Status

This request returns the current port status and the current value of the port status change bits.

bmRequestType	BRequest	wValue	wIndex	wLength	Data
10100011B	GET_STATUS	Zero	Port	Four	Port Status and Change Indicators

The port number must be a valid port number for that hub, greater than zero.

The first word of data contains *wPortStatus* (refer to Table 11-15). The second word of data contains *wPortChange* (refer to Table 11-14).

The bit locations in the *wPortStatus* and *wPortChange* fields correspond in a one-to-one fashion where applicable.

It is a Request Error if *wValue* or *wLength* are other than as specified above or if *wIndex* specifies a port that does not exist.

If the hub is not configured, the behavior of the hub in response to this request is undefined.

11.16.2.6.1 Port Status Bits

Table 11-15. Port Status Field, *wPortStatus*

Bit	Description
0	**Current Connect Status:** (PORT_CONNECTION) This field reflects whether or not a device is currently connected to this port. 0 = No device is present 1 = A device is present on this port
1	**Port Enabled/Disabled:** (PORT_ENABLE) Ports can be enabled by the USB System Software only. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by the USB System Software. 0 = Port is disabled 1 = Port is enabled
2	**Suspend:** (PORT_SUSPEND) This field indicates whether or not the device on this port is suspended. Setting this field causes the device to suspend by not propagating bus traffic downstreamThis field my be reset by a request or by resume signaling from the device attached to the port. 0 = Not suspended 1 = Suspended or resuming
3	**Over-current Indicator:** (PORT_OVER_CURRENT) If the hub reports over-current conditions on a per-port basis, this field will indicate that that the current drain on the port exceeds the specified maximum.For more details, see Section 7.2.1.2.1. 0 = All no over-current condition exists on this port 1 = An over-current condition exists on this port.
4	**Reset:** (PORT_RESET) This field is set when the host wishes to reset the attached device. It remains set until the reset signaling is turned off by the hub. 0 = Reset signaling not asserted 1 = Reset signaling asserted
5-7	**Reserved** These bits return 0 when read.
8	**Port Power:** (PORT_POWER) This field reflects a port's logical, power control state. Because hubs can implement different methods of port power switching, this field may or may not represent whether power is applied to the port. The device descriptor reports the type of power switching implemented by the hub. 0 = This port is in the Powered-off state 1 = This port is not in the Powered-off state
9	**Low Speed Device Attached:** (PORT_LOW_SPEED) This is relevant only if a device is attached. 0 = Full-speed device attached to this port 1 = Low-speed device attached to this port
10-15	**Reserved** These bits return 0 when read.

11.16.2.6.1.1 PORT_CONNECTION

When the Port Power bit is one, this bit indicates whether or not a device is attached. This field reads as one when a device is attached; it reads as zero when no device is attached. This bit is reset to zero when the port is in the Powered-off state or the Disconnected states. It is set to one when the port is in the Powered state, a device attach is detected (see Section 7.1.7.1) and the port transitions from the Disconnected state to the Disabled state.

SetPortFeature(PORT_CONNECTION) and ClearPortFeature(PORT_CONNECTION) requests shall not be used by the USB System Software and must be treated as no-operation requests by hubs.

11.16.2.6.1.2 PORT_ENABLE

This bit is set when the port is allowed to send or receive packet data or resume signaling.

This bit may be set only as a result of a SetPortFeature(PORT_RESET) request. When the hub exits the Resetting state or, if present, the Speed_eval state, this bit is set and bus traffic may be transmitted to the port. This bit may be cleared as the result of any of the following:

- The port being in the Powered-off state
- Receipt of a ClearPortFeature(PORT_ENABLE) request
- Port_Error detection
- Disconnect detection
- When the port enters the Resetting state as a result of receiving the SetPortFeature(PORT_RESET) request.

The hub response to a SetPortFeature(PORT_ENABLE) request is not specified. The preferred behavior is that the hub respond with a Request Error. This may not be used by the USB System Software. The ClearPortFeature(PORT_ENABLE) request is supported as specified in Section 11.5.1.4.

11.16.2.6.1.3 PORT_SUSPEND

This bit is set to one when the port is selectively suspended by the USB System Software. While this bit is set, the hub does not propagate downstream-directed traffic to this port, but the hub will respond to resume signaling from the port. This bit can be set only if the port's PORT_ENABLE bit is set and the hub receives a SetPortFeature(PORT_SUSPEND) request. This bit is cleared to zero on the transition from the SendEOP state to the Enabled state, or on the transition from the Restart_S state to the Transmit state, or on any event that causes the PORT_ENABLE bit to be cleared while the PORT_SUSPEND bit is set.

The SetPortFeature(PORT_SUSPEND) request may be issued by the USB System Software at any time but will have an effect only as specified in Section 11.5.

11.16.2.6.1.4 PORT_OVER-CURRENT

This bit is set to one while an over-current condition exists on the port. This bit is cleared when an over-current condition does not exist on the port.

If the voltage on this port is affected by an over-current condition on another port then this bit is set and remains set until the over-current condition on the affecting port is removed. When the over-current condition on the affecting port is removed, this bit is reset to zero if an over-current condition does not exist on this port.

Over-current protection is required on self-powered hubs (it is optional on bus-powered hubs) as outlined in Section 7.2.1.2.1.

The SetPortFeature(PORT_OVER_CURRENT) and ClearPortFeature(PORT_OVER_CURRENT) requests shall not be used by the USB System Software and may be treated as no-operation requests by hubs.

11.16.2.6.1.5 PORT_RESET

This bit is set while the port is in the Resetting state. A SetPortFeature(PORT_RESET) request will initiate the Resetting state if the conditions in Section 11.5.1.5 are met. This bit is set to zero while the port is in the Powered-off state.

The ClearPortFeature(PORT_RESET) request shall not be used by the USB System Software and may be treated as a no-operation request by hubs.

11.16.2.6.1.6 PORT_POWER

This bit reflects the current power state of a port. This bit is implemented on all ports whether or not actual port power switching devices are present.

While this bit is zero, the port is in the Powered-off state. Similarly, anything that causes this port to go to the Power-off state will cause this bit to be set to zero.

A SetPortFeature(PORT_POWER) will set this bit to one unless both C_HUB_LOCAL_POWER and Local Power Status (in *wHubStatus*) are set to one in which case the request is treated as a functional no-operation.

This bit may be cleared under the following curcumstances:

- Hub receives a ClearPortFeature(PORT_POWER).
- An over-current condition exists on the port.
- An over-current condition on another port causes the power on this port to be shut off.

The SetPortFeature(PORT_POWER) and ClearPortFeature(PORT_POWER) requests may be issued by the USB System Software whenever the port is not in the Not Configured state, but will have an effect only as specified in Section 11.11.

11.16.2.6.1.7 PORT_LOW_SPEED

This bit has meaning only when the PORT_ENABLE bit is set. This bit is set to one if the attached device is low-speed.

The SetPortFeature(PORT_LOW_SPEED) and ClearPortFeature(PORT_LOW_SPEED) requests shall not be used by the USB System Software and may be treated as no-operation requests by hubs.

11.16.2.6.2 Port Status Change Bits

Port status change bits are used to indicate changes in port status bits that are not the direct result of requests. Port status change bits can be cleared with a ClearPortFeature() request or by a hub reset. Hubs may allow setting of the status change bits with a SetPortFeature() request for diagnostic purposes. If a hub does not support setting of the status change bits, it may either treat the request as a Request Error or as a functional no-operation. Table 11-16 describes the various bits in the *wPortChange* field.

Table 11-16. Port Change Field, *wPortChange*

Bit	Description
0	**Connect Status Change:** (C_PORT_CONNECTION) Indicates a change has occurred in the port's Current Connect Status. The hub device sets this field as described in Section 11.16.2.6.2.1. 0 = No change has occurred to Current Connect status 1 = Current Connect status has changed
1	**Port Enable/Disable Change:** (C_PORT_ENABLE) This field is set to one when a port is disabled because of a Port_Error condition (see Section 11.8.1).
2	**Suspend Change:** (C_PORT_SUSPEND) This field indicates a change in the host-visible suspend state of the attached device. It indicates the device has transitioned out of the Suspend stateThis field is set only when the entire resume process has completed. That is, the hub has ceased signaling resume on this port. 0 = No change 1 = Resume complete
3	**Over-Current Indicator Change:** (C_PORT_OVER_CURRENT) This field applies only to hubs that report over-current conditions on a per-port basis (as reported in the hub descriptor). 0 = No change has occurred to Over-Current Indicator 1 = Over-Current Indicator has changed If the hub does not report over-current on a per-port basis, then this field is always zero.
4	**Reset Change:** (C_PORT_RESET) This field is set when reset processing on this port is complete. 0 = No change 1 = Reset complete
5-15	**Reserved** These bits return 0 when read.

11.16.2.6.2.1 C_PORT_CONNECTION

This bit is set when the PORT_CONNECTION bit changes because of an attach or detach detect event (see Section 7.1.7.1). This bit will be cleared to zero by a ClearPortFeature(C_PORT_CONNECTION) request or while the port is in the Powered-off state.

11.16.2.6.2.2 C_PORT_ENABLE.

This bit is set when the PORT_ENABLE bit changes from one to zero as a result of a Port Error condition (see Section 11.8.1.). This bit is not set on any other changes to PORT_ENABLE.

This bit may be set if, on a SetPortFeature(PORT_RESET) the port stays in the Disabled state because an invalid idle state exists on the bus (see Section 11.8.2).

This bit will be cleared by a ClearPortFeature(C_PORT_ENABLE) request or while the port is in the Powered-off state.

11.16.2.6.2.3 C_PORT_SUSPEND

This bit is set on the following transitions:

- on transition from the Resuming state to the SendEOP state
- on transition from the Restart_S state to the Transmit state.

This bit will be cleared by a ClearPortFeature(C_PORT_SUSPEND) request, or while the port is in the Powered-off state.

11.16.2.6.2.4 C_PORT_OVER-CURRENT

This bit is set when the PORT_OVER_CURRENT bit changes from zero to one or from one to zero. This bit is also set if the port is placed in the Powered-off state due to an over-current condition on another port.

This bit will be cleared when the port is in the Not Configured state or by a ClearPortFeature(C_PORT_OVER-CURRENT) request.

11.16.2.6.2.5 C_PORT_RESET

This bit is set when the port <u>transitions</u> from the Resetting state (or, if present, the Speed_eval state) to the Enabled state.

This bit will be cleared by a ClearPortFeature(C_PORT_RESET) request, or while the port is in the Powered-off state.

11.16.2.7 Set Hub Descriptor

This request overwrites the hub descriptor.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero	Descriptor Length	Descriptor

The SetDescriptor request for the hub class descriptor follows the same usage model as that of the standard SetDescriptor request (refer to Chapter 9). The standard hub descriptor is denoted by using the value *bDescriptorType* defined in Section 11.15.2.1. All hubs are required to implement one hub descriptor, with descriptor index zero.

This request is optional. This request writes data to a class-specific descriptor. The host provides the data that is to be transferred to the hub during the data transfer phase of the control transaction. This request writes the entire hub descriptor at once.

Hubs must buffer all the bytes received from this request to ensure that the entire descriptor has been successfully transmitted from the host. Upon successful completion of the bus transfer, the hub updates the contents of the specified descriptor.

It is a Request Error if *wIndex* is not zero or if *wLength* does not match the amount of data sent by the host. Hubs that do not support this request respond with a STALL during the Data stage of the request.

If the hub is not configured, the hub's response to this request is undefined.

11.16.2.8 Set Hub Feature

This request sets a value reported in the hub status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100000B	SET_ FEATURE	Feature Selector	Zero	Zero	None

Setting a feature enables that feature; refer to Table 11-12for the feature selector definitions that apply to the hub as recipient. Change indicators may not be acknowledged using this request.

It is a Request Error if *wValue* is not a feature selector listed in Table 11-12 or if *wIndex* or *wLength* are not as specified above.

If the hub is not configured, the hub's response to this request is undefined.

11.16.2.9 Set Port Feature

This request sets a value reported in the port status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100011B	SET_ FEATURE	Feature Selector	Port	Zero	None

The port number must be a valid port number for that hub, greater than zero.

Setting a feature enables that feature or starts a process associated with that feature; see Table 11-12 for the feature selector definitions that apply to a port as a recipient. Change indicators may not be acknowledged using this request. Features that can be set with this request are:

- PORT_RESET
- PORT_SUSPEND
- PORT_POWER
- C_PORT_CONNECTION*
- C_PORT_RESET*
- C_PORT_ENABLE*
- C_PORT_SUSPEND*
- C_PORT_OVER_CURRENT*.

*denotes features that are not required to be set by this request.

Setting the PORT_SUSPEND feature causes bus traffic to cease on that port and, consequently, the device to suspend. Setting the reset feature PORT_RESET causes the hub to signal reset on that port. When the reset signaling is complete, the hub sets the C_PORT_RESET change indicator and immediately enables the port. Refer to Section 11.10 for a complete discussion of host-initiated reset behavior. Also see Section 11.16.2.6.1 for further details.

The hub must meet the following requirements:

- If the port is in the Powered-off state, the hub must treat a SetPortFeature(PORT_RESET) request as a functional no-operation.
- If the port is not in the Enabled or Transmitting state, the hub must treat a SetPortFeature(PORT_SUSPEND) request as a functional no-operation.
- If the port is not in the Powered-off state, the hub must treat a SetPortFeature(PORT_POWER) request as a functional no-operation.

It is a Request Error if *wValue* is not a feature selector listed in Table 11-12, if *wIndex* specifies a port that doen't exist, or if *wLength* is not as specified above.

If the hub is not configured, the hub's response to this request is undefined.

Index

4X over-sampling state machine DPLLs, 7.1.15

A

abnormal termination sequences, 11.3.3
aborting/retiring transfers
  aborting control transfers, 5.5.5
  client role in, 10.5.2.2
  conditions for, 5.3.2
  message pipes and, 5.3.2.2
  packet size and, 5.5.3
  USBDI role, 10.5.3.2.1
access frequency of control pipes, 5.5.4
Acknowledge packet. *See* ACKs
ACKs, 8.3.1 *Table 8-1*
  in bulk transfers, 8.5.1
  in control transfers, 8.5.2, 8.5.2.1
  corrupted ACK handshake, 8.5.2.3, 8.6.4
  in data toggle, 8.6, 8.6.1, 8.6.2
  defined, 2.0 *glossary*
  function response to OUT transactions, 8.4.5.3
  host response to IN transactions, 8.4.5.2
  overview, 8.4.4
  in request processing, 9.2.6
A connectors. *See* Series "A" and "B" connectors
active devices, defined, 2.0 *glossary*
active pipes, 10.5.2.2
Active state (transmitter state machine), 11.6.2, 11.6.2.2
adaptive endpoints
  connection requirements, 5.10.4.3
  feedback for isochronous transfers, 5.10.4.2
  overview, 5.10.4.1.3
adding devices. *See* dynamic insertion and removal
Address device state
  bus enumeration process, 9.1.2
  overview, 9.1.1.4
  standard device requests, 9.4.1:9.4.11
  visible device state table, 9.1.1 *Table 9-1*
addresses
  Address device state, 9.1.1 *Table 9-1*, 9.1.1.4, 9.1.2, 9.4.1:9.4.11
  aliasing, 8.3.2
  assignment
    after dynamic insertion or removal, 4.6.3
    bus enumeration, 2.0 *glossary*, 4.6.3, 9.1.2
    device initialization, 10.5.1.1
    operations overview, 9.2.2
    re-enumerating sub-trees, 10.5.4.6
    staged power switching in functions and, 7.2.1.4
    time limits for completing, 9.2.6.3
    USB System Software role, 4.9
  endpoint addresses, 5.3.1, 9.6.4
  SetAddress() request, 9.4.6
address fields
  address field (ADDR), 8.3.2.1, 8.3.5.1, 8.4.1
  endpoint field (ENDP), 8.3.2.2, 8.3.5.1, 8.4.1
  packet address fields, 8.3.2:8.3.2.2
ADDR field
  overview, 8.3.2.1
  token CRCs, 8.3.5.1
  in token packets, 8.4.1
aging, data-rate inaccuracies and, 7.1.11
aliasing addresses, 8.3.2
allocating bit times in handshake packets, 11.3.3
allocating buffers. *See* buffers
allocating USB bandwidth
  transfer management, 5.9.1:5.9.1.5
  USB System role, 10.3.2
alternate settings for interfaces
  configuration requirements, 10.3.1
  GetInterface() request, 9.4.4
  in interface descriptors, 9.6.3
  SetInterface() request, 9.4.10
  USBDI mechanisms, 10.5.2.10
  USB support for, 9.2.3
American National Standard/Electronic Industries Association, 6.7.1
American Standard Test Materials, 6.7.1
ANSI/EIA-364-C (12/94), 6.7.1
applications
  in source-to-sink connectivity, 5.10.4.3
  USB suitability for, 3.3
application space taxonomy, 3.2
architectural overview of USB
  architectural extensions, 4.10
  bus protocol, 4.4
  bus topology, 4.1.1
  data flow types, 4.7, 4.7:4.7.5
  hub architecture, 4.8.2.1, 11.1.1, 11.13.2
  mechanical and electrical specifications, 4.2:4.2.2, 6.1
  physical interface, 4.2, 4.2:4.2.2
  power, 4.3, 4.3:4.3.2
  robustness and error handling, 4.5, 4.5:4.5.2
  system configuration, 4.6, 4.6:4.6.3
  USB devices, 4.1.1.2, 4.8, 4.8:4.8.2.2
  USB host, 4.1.1.1, 4.9
  USB system description, 4.1, 4.1:4.1.1.2
assigning addresses. *See* addresses; bus enumeration
ASTM-D-4565, 6.6.3, 6.7.1
ASTM-D-4566, 6.6.3, 6.7.1
asynchronous data transfers, 2.0 *glossary*, 4.9
asynchronous endpoints
  connection requirements, 5.10.4.3
  feedback for isochronous transfers, 5.10.4.2
  overview, 5.10.4.1.1
asynchronous RA, 2.0 *glossary*, 5.10.4.3. *See also* RA (rate adaptation)
asynchronous SRC, 2.0 *glossary*. *See also* SRC
Attached device state
  in bus enumeration process, 9.1.2
  overview, 9.1.1.1
  visible device state table, 9.1.1 *Table 9-1*
attaching devices. *See* dynamic insertion and removal

attenuation, 7.1.17
attributes of devices in configuration descriptors, 9.6.2
attributes of endpoints in endpoint descriptors, 9.6.4
audio connectivity, 5.10.4.3.1
Audio Device Class Specification Revision 1.0, 9.6
audio devices, defined, 2.0 *glossary*
available time in frames
  bulk transfers and, 5.8.4
  bus bandwidth reclamation, 5.9.5
  control transfers and, 5.5.4
  interrupt transfer bus access constraints, 5.7.4
  isochronous transfers and, 5.6, 5.6.4
AWG, 2.0 *glossary*, 6.6.2

B

babble
  defined, 2.0 *glossary*
  EOF2 timing points and, 11.2.2
  error detection and recovery, 8.7.4
background of USB development, 3.1:3.3
bAlternateSetting field (interface descriptors), 9.6.3
bandwidth
  allocating for pipes, 4.4, 4.7.5
  bandwidth reclamation, 5.9.5
  defined, 2.0 *glossary*
  transfer management, 4.7.5, 5.9.1:5.9.1.5, 10.3.2
  USB system role in, 10.3.2
battery-powered hubs, 7.2.1
bcdDevice field (device descriptors), 9.6.1
bcdUSB field (device descriptors), 9.6.1
bConfigurationValue field (configuration descriptors), 9.6.2
B connectors. *See* Series "A" and "B" connectors
bDescLength field (hub descriptors), 11.15.2.1
bDescriptorType field
  configuration descriptors, 9.6.2
  device descriptors, 9.6.1
  endpoint descriptors, 9.6.4
  hub descriptors, 11.15.2.1, 11.16.2.4, 11.16.2.7
  interface descriptors, 9.6.3
  string descriptors, 9.6.5
bDeviceClass field, 9.6.1, 11.15.1
bDeviceProtocol field (device descriptors), 9.6.1
bDeviceSubClass field, 9.6.1, 11.15.1
bEndpointAddress field, 9.6.4, 11.15.1
bHubContrCurrent field (hub descriptors), 11.15.2.1
bi-directional communication flow, 5.6.2, 5.8.2
big endian, defined, 2.0 *glossary*
bInterfaceClass field, 9.6.3, 11.15.1
bInterfaceNumber field (interface descriptors), 9.6.3
bInterfaceProtocol field, 9.6.3, 11.15.1
bInterfaceSubClass field, 9.6.3, 11.15.1
bInterval field, 9.6.4, 11.15.1
bit cells, decoding, 7.1.15
bitmaps of hub and port status changes, 11.13.4
bit ordering, 8.1
bits, defined, 2.0 *glossary*
bit stuffing
  bit stuff violations, 8.7.1
  calculating transaction times, 5.9.3
  defined, 2.0 *glossary*
  overview, 7.1.9
bit times
  bit time designations, 11.3
  bit time zero, 11.3
  in transaction completion prediction, 11.3.3
bLength field
  configuration descriptors, 9.6.2
  device descriptors, 9.6.1
  endpoint descriptors, 9.6.4
  interface descriptors, 9.6.3
  string descriptors, 9.6.5
blocking packets in Collision conditions, 11.8.3
blunt cut termination, 6.4.2, 6.4.3
bmAttributes field
  configuration descriptors, 9.6.2
  endpoint descriptors, 9.6.4
  hub descriptors, 11.14, 11.15.1
bMaxPacketSize0 field (device descriptors), 9.6.1
bmRequestType field
  hub class requests, 11.16.2
  overview, 9.3.1
  Setup data format, 9.3
  standard device requests, 9.4
bNbrPorts field (hub descriptors), 11.15.2.1
bNumConfigurations field (device descriptors), 9.6.1
bNumEndpoints field, 9.6.3, 11.15.1
bNumInterfaces field (configuration descriptors), 9.6.2
bPwrOn2PwrGood field, 11.11, 11.15.2.1
bRequest field
  hub class requests, 11.16.2
  overview, 9.3.2
  Setup data format, 9.3
  standard device requests, 9.4
  standard hub requests, 11.16.1
broadcast mode of hub operation, 11.1.2.1
B/S or b/S, defined, 2.0 *glossary*
bString field (string descriptors), 9.6.5
buffers
  buffer impedance, 7.1.1.1
  calculating sizes in functions and software, 5.9.4
  client pipes and, 10.5.1.2.2
  client role in, 10.3.3, 10.5.3
  defined, 2.0 *glossary*
  endpoint buffer size, 4.4
  hub I/O buffer requirements, 11.12:11.12.2
  identifying location and length, 10.3.4
  interrupt transfers and, 5.7.3
  isochronous transfers and, 5.10.4.2
  non-USB isochronous application, 5.10.1
  packet buffers, 2.0 *glossary*
  prebuffering data, 5.10.5
  rate matching and, 5.10.8
  rise and fall times for full-speed buffers, 7.1.2
  USBD role in allocating, 10.5.1.2.1
bulk transfers
  bus access constraints, 5.8.4
  data format, 5.8.1
  data sequences, 5.8.5
  defined, 2.0 *glossary*, 5.4
  direction, 5.8.2
  overview, 4.7.2, 5.8
  packet size, 5.8.3, 9.6.4
  transaction format, 8.5.1

bulk transfers *(continued)*
  transaction organization within IRPs, 5.9.2
  USBD pipe mechanism responsibilities, 10.5.3.1.3
bus access for transfers
  bulk transfer constraints, 5.8.4
  bus bandwidth reclamation, 5.9.5
  calculating buffer sizes, 5.9.4
  calculating bus transaction times, 5.9.3
  client software role in, 5.9.1.1
  control transfer constraints, 5.5.4
  HCD role in, 5.9.1.3
  Host Controller role in, 5.9.1.5
  interrupt transfer constraints, 5.7.4
  isochronous transfer constraints, 5.6.4
  transaction list, 5.9.1.4
  transaction tracking, 5.9.2
  transfer management, 5.9.1:5.9.1.5
  transfer type overview, 5.4
  USBD role in, 5.9.1.2
bus clock, 5.10.2, 5.10.3, 5.10.8
bus enumeration
  defined, 2.0 *glossary*
  device initialization, 10.5.1.1
  overview, 4.6.3, 9.1.2
  re-enumerating sub-trees, 10.5.4.6
  staged power switching in functions, 7.2.1.4
  USB System Software role, 4.9
Bus Management services, 10.5.4.2
bus-powered devices and functions
  configuration descriptors, 9.6.2
  defined, 4.3.1
  device states, 9.1.1.2
  high-power bus-powered functions, 7.2.1.4
  low-power bus-powered functions, 7.2.1.3
  power budgeting, 9.2.5.1
bus-powered hubs
  configuration, 11.14
  defined, 4.3.1, 7.2.1
  device states, 9.1.1.2
  overview, 7.2.1.1
  power switching, 11.11
  voltage drop budget, 7.2.2
bus protocol overview, 4.4
Bus_Reset receiver state, 11.6.1, 11.6.1.6
bus states
  detecting, 11.16.2.3
  evaluating after reset, 7.1.7.1
  global suspend, 7.1.7.4.1
  Host Controller role in state handling, 10.2.1
  signaling levels and, 7.1.7
bus timing/electrical characteristics, 7.3.2
bus topology, 5.2:5.2.5
  client-software-to-function relationship, 5.2.5
  defined, 4.1
  devices, 5.2.2
  hosts, 5.2.1
  illustrated, 4.1.1
  logical bus topology, 5.2.4
  physical bus topology, 5.2.3
bus transaction timeout in isochronous transfers, 5.10.7
bus turn-around time, 2.0 glossary, 7.1.18, 8.7.2
bypass capacitors, 7.2.4.1, 7.2.4.2
bytes, defined, 2.0 *glossary*

C

cable assemblies, 6.4:6.4.4
cable attenuation, 7.1.17
cable delay
  electrical characteristics, 7.3.2 *Table* 7-9
  full-speed cables, 6.4.2
  hub differential delay, differential jitter, and SOP distortion, 7.3.3 *Figure* 7-42
  hub EOP delay and EOP skew, 7.3.3 *Figure* 7-43
  hub signaling timings, 7.1.14
  inter-packet delay and, 7.1.18
  low-speed cables, 6.4.3, 7.1.1.2
  overview, 7.1.16
  propagation delay, 6.4.1, 6.7 *Table 6-7*, 7.1.1.2
  skew delay, 6.7 *Table 6-7*, 7.1.3, 7.3.3 *Figure* 7-43
cables
  attenuation, 7.1.17
  cable assemblies, 6.4:6.4.4
  cable delay (*See* cable delay)
  captive cables
    full-speed captive cable assemblies, 6.4.2
    inter-packet delay and, 7.1.18
    low-speed captive cable assemblies, 6.4.3
    maximum capacitance, 7.1.6
    termination, 7.1.5
  color choices, 6.4
  construction, 6.6.2
  description, 6.6.1
  detachable cables
    cable delay, 7.1.16
    connectors and, 6.2
    detachable cable assemblies, 6.4.1
    inter-packet delay and, 7.1.18
    low-speed detachable cables, 6.4.4
    maximum capacitance, 7.1.6
    termination, 7.1.5
    voltage drop budget, 7.2.2
  electrical characteristics and standards, 4.2.1, 6.6.3, 6.7, 7.3.2 *Table* 7-9
  end-to-end signal delay, 7.1.19
  environmental characteristics, 6.6.4, 6.7
  flyback voltage, 7.2.4.2
  full-speed cables, 6.4.2
  impedance, 6.4.1, 6.4.2, 6.7 *Table 6-7*
  input capacitance, 7.1.6
  length, 6.4.1, 6.4.2, 6.4.3
  listing, 6.6.5
  low-speed cables, 6.4.3, 6.4.4
  mechanical configuration and material requirements, 6.6:6.6.5, 6.7
  overview, 6.3
  prohibited cable assemblies, 6.4.4
  pull-out standards, 6.7 *Table* 6-7
  shielding, 6.6, 6.6.1
  termination, 7.1.5
  voltage drop budget, 7.2.2

- calculations
  - buffering for rate matching, 5.10.8
  - buffer sizes in functions and software, 5.9.4
  - bus transaction times, 5.9.3
- capabilities, defined, 2.0 *glossary*
- capacitance
  - after dynamic attach, 7.2.4.1
  - decoupling capacitance, 7.3.2 *Table* 7-5
  - input capacitance, 7.1.6, 7.3.2 *Table* 7-5
  - low-speed buffers, 7.1.1.2, 7.1.2
  - low-speed cable capacitive loads, 6.4.3
  - optional edge rate control capacitors, 7.1.6
  - pull-up resistors and, 7.1.5
  - single-ended capacitance, 7.1.1.2
  - small capacitors, 7.1.6
  - target maximum droop and, 7.2.4.1
  - unmated contact capacitance, 7.3.2 *Table* 7-9
- capacitive load, 6.7 *Table 6-7*
- captive cables
  - full-speed captive cable assemblies, 6.4.2
  - inter-packet delay and, 7.1.18
  - low-speed captive cable assemblies, 6.4.3
  - maximum capacitance, 7.1.6
  - termination, 7.1.5
- change bits
  - device states, 11.13.2
  - hub and port status change bitmap, 11.13.4
  - hub status, 11.16.2.5
  - over-current status change bits, 11.13.5
  - port status change bits, 11.16.2.6.2:11.16.2.6.2.5
  - Status Change endpoint defined, 11.13.1
- change propagation, host state handling of, 10.2.1
- characteristics of devices, 2.0 *glossary*, 9.6.2
- C_HUB_LOCAL_POWER, 11.11, 11.16.2, 11.16.2.1, 11.16.2.5, 11.16.2.6.1.6
- C_HUB_OVER_CURRENT, 11.16.2, 11.16.2.1
- C_HUB_OVER_POWER, 11.16.2.5
- classes of devices. *See* device classes
- Class field, 9.2.3, 9.6.3
- class-specific descriptors, 9.5, 11.15.2.1
- class-specific requests
  - hub class-specific requests, 11.16.2:11.16.2.9
  - time limits for completing, 9.2.6.5
  - USBDI mechanisms, 10.5.2.8
- ClearFeature() request, CLEAR_FEATURE
  - ClearHubFeature() request, 11.16.2.1
  - ClearPortFeature() request, 11.16.2.2
  - endpoint status and, 9.4.5
  - hub class requests, 11.16.2
  - hub requests, 11.16.1
  - overview, 9.4.1
  - standard device request codes, 9.4
- ClearHubFeature() request
  - clearing hub features, 11.16.2.5
  - hub class requests, 11.16.2
  - hub class-specific requests, 11.16.2.1
- clearing pipes, 10.5.2.2
- ClearPortFeature() request
  - clearing status change bits, 11.13.2, 11.16.2.6.2
  - C_PORT_CONNECTION, 11.16.2.6.2.1
  - C_PORT_ENABLE, 11.16.2.6.2.2
  - C_PORT_OVER-CURRENT, 11.16.2.6.2.4
- ClearPortFeature() request *(continued)*
  - C_PORT_RESET, 11.16.2.6.2.5
  - C_PORT_SUSPEND, 11.16.2.6.2.3
  - hub class requests, 11.16.2, 11.16.2.2
  - PORT_CONNECTION, 11.16.2.6.1.1
  - PORT_ENABLE, 11.5.1.4, 11.16.2.6.1.2
  - PORT_LOW_SPEED, 11.16.2.6.1.7
  - PORT_OVER_CURRENT, 11.16.2.6.1.4
  - PORT_POWER, 11.5.1.2, 11.16.2.6.1.6
  - PORT_RESET, 11.16.2.6.1.5
  - PORT_SUSPEND, 11.5.1.9
- client pipes, 10.5.1.2.2
- client software
  - in bus topology, 5.2, 5.2.1, 5.2.5
  - client software-to-function relationships, 5.2, 5.2.5
  - in communication flow, 5.3
  - control transfers and, 5.5
  - defined, 2.0 *glossary*
  - as implementation focus area, 5.1
  - Master Client status, 10.5.3.2.4, 10.5.4.2
  - notification identification, 10.3.4
  - role in configuration, 10.3.1
  - role in data transfers, 10.3.3
  - service clock and, 5.10.2
  - in source-to-sink connectivity, 5.10.4.3
  - in transfer management, 5.9.1, 5.9.1.1
- clock model
  - buffering for rate matching, 5.10.8
  - bus clock, 5.10.2
  - clock encoding scheme in electrical specifications overview, 4.2.1
  - clock synchronization, 5.10.3
  - clock-to-clock phase differences, 5.10.3
  - defined, 5.10
  - hub clock source, 11.2.1
  - in non-USB isochronous application, 5.10.1
  - overview, 5.10.2
  - sample clock, 5.10.2
  - service clock, 5.10.2
  - using SOF tokens as clocks, 5.10.5
- clock timings, 7.3.2 *Table 7-6*, 7.3.2 *Table 7-7*
- CMOS driver circuit, 7.1.1.1
- codes. *See specific types of codes*
- Collision conditions, 11.8.3
- color choices
  - cables, 6.4
  - plugs, 6.5.4.1
  - receptacles, 6.5.3.1
- commanded stalls, 8.4.4
- commands. *See* requests
- common mode range for differential input sensitivity, 7.1.4
- Communication Cables (UL Subject-444), 6.7.1
- *Communication Cables (UL Subject-444)*, 6.6.5
- communication flow, 5.3:5.3.2.2
- completed operations, 9.2.6
- completed transactions, 11.3.3
- completion times for hub requests, 11.16.1
- compound devices
  - bus-powered hubs, 7.2.1.1
  - in bus topology, 5.2.3
  - defined, 4.8.2.2

- compound devices *(continued)*
  - hub descriptors for, 11.15.2.1
  - power configuration, 11.14
  - self-powered hubs, 7.2.1.2
- conductor resistance unbalance, 6.6.3
- conductors
  - mechanical specifications, 4.2.2
  - power and signal conductors in cables, 6.3, 6.6.2
  - resistance, 6.6.3
- configuration
  - bus enumeration, 4.6.3, 9.1.2
  - configuration management, 10.5.4.1.1
  - Configured device state, 9.1.1.5
  - control transfers and, 5.5.4
  - descriptors, 9.4.3, 9.5, 9.6.1:9.6.4, 11.15.1 (*See also* descriptors)
  - device attachment, 4.6.1
  - device configuration, 10.3.1
  - device removal, 4.6.2, 10.5.4.1.4
  - function configuration, 10.3.1
  - hubs, 11.14
  - information in device characteristics, 4.8.1
  - initial device configuration, 10.5.4.1.2
  - interrupt transfers and, 5.7.4
  - modifying device configuration, 10.5.4.1.3
  - multiple configurations, 9.6.1
  - multiple interfaces, 9.2.3
  - operations overview, 9.2.3
  - power distribution and, 7.2.1
  - remote wakeup capabilities, 9.2.5.2
  - requests
    - configuration requests, 5.9.1.2
    - GetConfiguration() request, 9.4.2
    - SetConfiguration() request, 9.4.7
  - required configurations before usage, 10.3.1
  - USB configuration, 10.3.1
  - USBDI mechanisms for getting current settings, 10.5.2.4
  - USBD role in, 5.9.1.2, 10.5.4.1:10.5.4.1.4
- Configuration = 0 signal/event, 11.5
- configuration descriptors, 9.4.3, 11.15.1
- Configured device state
  - in bus enumeration process, 9.1.2
  - overview, 9.1.1.5
  - standard device requests and, 9.4.1:9.4.11
  - visible device state table, 9.1.1 *Table 9-1*
- configuring software, defined, 2.0 *glossary*
- Connect bus state, 7.1.7, 7.1.7.1
- connecting devices. *See* dynamic insertion and removal
- connection status, 11.16.2.6.2, 11.16.2.6.2.1
- connectivity
  - audio connectivity, 5.10.4.3.1
  - hub fault recovery mechanisms, 11.1.2.3
  - Hub Repeater responsibilities, 11.1
  - hubs, 11.1, 11.1.2:11.1.2.3
  - packet signaling connectivity, 11.1.2.1
  - resume connectivity, 11.1.2.2
  - source/sink connectivity, 5.10.4.3
  - synchronous data connectivity, 5.10.4.3.2
- connectors
  - input capacitance, 7.1.6
  - inrush current and, 7.2.4.1
  - interface and mating drawings, 6.5.3, 6.5.4
  - keyed connector protocol, 6.2
  - mechanical configuration and material requirements, 4.2.2, 6.5:6.5.4.3
  - orientation, 6.5.1
  - Series "A" and Series "B" plugs, 6.5.4
  - Series "A" and Series "B" receptacles, 6.5.3
  - standards for, 6.7
  - termination data, 6.5.2
  - USB Icon, 6.5
- construction, cable, 6.6.2
- contact arcing, minimizing, 7.2.4.1
- contact capacitance standards, 6.7 *Table 6-7*
- contact current rating standards, 6.7 *Table 6-7*
- contact materials, 6.5.3.3, 6.5.4.3
- control endpoints, 2.0 *glossary*. *See also* control transfers
- controlling hubs, defined, 7.1.7.5
- control mechanisms
  - device states and control information, 11.13.2
  - Host Controller control flow management, 4.9
  - of USB host, 10.1.2
- control pipes, 2.0 *glossary*. *See also* control transfers; message pipes; pipes
- control transfers
  - bus access constraints, 5.5.4
  - control pipes in device characteristics, 4.8.1
  - data format, 5.5.1
  - data sequences, 5.5.5
  - defined, 2.0 *glossary*, 5.4
  - device requests, 9.3
  - direction, 5.5.2
  - error handling on last data transaction, 8.5.2.3
  - full-speed limits, 5.5.4 *Table 5-1*
  - low-speed limits, 5.5.4 *Table 5-1*
  - overview, 4.7.1, 5.5
  - packet size, 5.5.3, 9.6.4
  - protocol stalls, 8.4.4
  - reporting status results, 8.5.2.1
  - stages, 2.0 *glossary*, 5.5
  - STALL handshakes returned by control pipes, 8.5.2.4
  - transaction format, 8.5.2
  - transaction organization within IRPs, 5.9.2
  - USBD pipe mechanism responsibilities, 10.5.3.1.4
  - variable-length data stage, 8.5.2.2
- corrupted transfers and requests
  - in control transfers, 8.5.2
  - corrupted ACK handshake, 8.5.2.3, 8.6.4
  - corrupted CRCs, 10.2.6
  - corrupted IN tokens, 8.4.5.1
  - corrupted PIDs, 8.3.1
  - corrupted SOF packets in isochronous transfers, 5.10.6
  - in data toggle, 8.6.3
  - error detection and recovery, 8.7:8.7.4
  - function response to OUT transactions, 8.4.5.3
  - host response to IN transactions, 8.4.5.2
  - NAK or STALL handshake, 8.6.3

- costs of implementation, 3.3
- C_PORT_CONNECTION
  - clearing, 11.16.2.2
  - defined, 11.16.2.6.2.1
  - hub class feature selectors, 11.16.2
  - Port Change field, 11.16.2.6.2
  - SetPortFeature() request, 11.16.2.9
- C_PORT_ENABLE
  - ClearPortFeature() request, 11.16.2.2
  - defined, 11.16.2.6.2.2
  - hub class feature selectors, 11.16.2
  - Port Change field, 11.16.2.6.2
  - SetPortFeature() request, 11.16.2.9
- C_PORT_OVER_CURRENT
  - clearing, 11.16.2.2
  - defined, 11.16.2.6.2.4
  - hub class feature selectors, 11.16.2
  - over-current conditions, 11.11.1, 11.13.5
  - Port Change field, 11.16.2.6.2
  - SetPortFeature() request, 11.16.2.9
- C_PORT_RESET
  - clearing, 11.16.2.2
  - defined, 11.16.2.6.2.5
  - hub class feature selectors, 11.16.2
  - Port Change field, 11.16.2.6.2
  - SetPortFeature() request, 11.16.2.9
- C_PORT_SUSPEND
  - clearing, 11.16.2.2
  - defined, 11.16.2.6.2.3
  - hub class feature selectors, 11.16.2
  - Port Change field, 11.16.2.6.2
  - resume conditions and, 11.4.4
  - SetPortFeature() request, 11.16.2.9
- CRCs
  - bit stuffing, 7.1.9
  - in bulk transfers, 8.5.1
  - corrupted CRCs, 10.2.6
  - in data packets, 8.3.5.2, 8.4.3
  - defined, 2.0 *glossary*
  - in error detection, 8.7.1
  - overview, 8.3.5
  - protection in isochronous transfers, 5.10.7
  - resending, 8.6.4
  - in token packets, 8.3.5.1, 8.4.1
  - USB robustness and, 4.5, 4.5.1
- cross-over points of data lines, 7.1.13.2
- cross-over voltage in signaling, 7.1.2
- crystal capacitive loading, 7.1.11
- CTI, 2.0 *glossary*, 3.1
- current
  - current averaging profile, 7.2.3
  - current spikes during suspend/resume, 7.2.3
  - supply current, 7.3.2 *Table 7-5*
- current frame in hub timing, 11.2.1
- current limiting
  - bus-powered hubs, 7.2.1.1
  - dynamic attach and detach, 7.2.4.1
  - in over-current conditions, 11.13.5
  - power control during suspend/resume, 7.2.3
  - remote wakeup and, 7.2.3
  - self-powered functions, 7.2.1.5
- cyclic redundancy check. *See* CRCs

D

- D+ or D- lines
  - average voltage, 7.1.2
  - signaling levels and, 7.1.7
  - signal termination, 7.1.5
  - during signal transitions, 7.1.4
  - single-ended capacitance, 7.1.1.2
  - standardized contact terminating assignments, 6.5.2
- data
  - data defined, 5.10.4
  - data encoding/decoding, 7.1.8
  - data prebuffering, 5.10.5
  - data processing role of Host Controller, 10.2.4
- DATA0/DATA1 PIDs, 8.3.1 *Table 8-1*
  - in bulk transfers, 5.8.5, 8.5.1
  - comparing sequence bits, 8.6.2
  - in control transfers, 8.5.2
  - in data packets, 8.4.3
  - in interrupt transactions, 8.5.3
  - interrupt transfers and, 5.7.5
  - synchronization and, 8.6
- data field in packets, 8.3.4, 8.4.3
- data flow model. *See* transfers
- data flow types. *See* transfer types
- data formats
  - bulk transfers, 5.8.1
  - control transfers, 5.5.1
  - interrupt transfers, 5.7.1
  - isochronous transfers, 5.6.1, 5.10.4
  - overview, 5.4
- Data J state. *See* J bus state
- Data K bus state. *See* K bus state
- data packets
  - bus protocol overview, 4.4
  - data CRCs, 8.3.5.2
  - in isochronous transfers, 8.5.4
  - overview, 8.4.3
  - packet field formats, 8.3:8.3.5.2
- data payload
  - bulk transfer data packets, 5.8.3
  - calculating transaction times, 5.9.3
  - defined, 5.3.2
  - interrupt transfers, 5.7.3
  - isochronous transfers, 5.6.3
  - packet size constraints, 5.5.3, 5.6.3
- data PIDs, 8.3.1 *Table 8-1*. *See also* DATA0/DATA1 PIDs
- data rates
  - adaptive endpoints, 5.10.4.1.3
  - asynchronous endpoints, 5.10.4.1.1
  - in buffering calculations, 5.10.8
  - data-rate tolerance, 7.1.11
  - defined, 5.10.4
  - in electrical specifications overview, 4.2.1
  - feedback for isochronous transfers, 5.10.4.2
  - full-speed source electrical characteristics, 7.3.2 *Table 7-6*
  - low-speed source electrical characteristics, 7.3.2 *Table 7-7*
  - overview, 7.1.11

data rates *(continued)*
  sample clock and, 5.10.2
  synchronous endpoints, 5.10.4.1.2
data retry indicators in control transfers, 5.5.5
data sequences
  bulk transfers, 5.8.5
  control transfers, 5.5.5
  interrupt transfers, 5.7.5
  isochronous transfers, 5.6.5
data signaling, 7.1.7.2
data signal rise and fall time. *See* rise and fall times
data source jitter, 7.1.13.1, 7.1.15
data source signaling, 7.1.13
Data stage
  in control transfers, 5.5, 5.5.5, 8.5.2
  error handling on last data transaction, 8.5.2.3
  length of data, 9.3.5
  packet size constraints, 5.5.3
  variable-length data stages, 8.5.2.2
data toggle
  bulk transfers, 5.8.5
  in bulk transfers, 8.5.1
  corrupted ACK handshake, 8.6.4
  data corrupted or not accepted, 8.6.3
  in data packets, 8.4.3
  initialization via SETUP token, 8.6.1
  in interrupt transactions, 8.5.3
  interrupt transfers and, 5.7.5
  low-speed transactions, 8.6.5
  overview, 8.6
  successful data transactions, 8.6.2
data transfers. *See* data packets; Data stage; transfers
DC electrical characteristics, 7.3.2 *Table 7-5*
debounce intervals in connection events, 7.1.7.1
debouncing connections, 11.8.2
decoupling capacitance, 7.3.2 *Table 7-5*
default addresses of devices, 2.0 *glossary*, 9.1.1.4, 10.5.1.1
Default bus state, 7.1.7.3
Default Control Pipe
  in bus enumeration process, 9.1.2
  in communication flow, 5.3
  control transfer packet size constraints, 5.5.3
  defined, 4.4, 5.3.2
  endpoint zero requirements, 5.3.1.1
  as message pipe, 5.3.2.2
  size description in descriptors, 9.6.1
Default device state
  overview, 9.1.1.3
  standard device requests and, 9.4.1:9.4.11
  visible device state table, 9.1.1 *Table 9-1*
default pipes, 2.0 *glossary*, 10.5.1.2.1
delays. *See* cable delay; differential delay; propagation delay
delivery rates in isochronous transfers, 4.7.4
DEOP signal/event, 11.7
descriptors
  accessing, 11.15.1
  in bus enumeration process, 9.1.2
  class-specific descriptors, 9.5, 11.15.2.1
  configuration descriptors, 9.6.2, 10.3.1, 10.5.2.4
descriptors *(continued)*
  control transfers and, 5.5, 5.5.3
  defined, 9.5
  device class definitions, 9.7, 9.7.1
  device descriptors, 9.4 *Table 9-5*, 9.6.1:9.6.5
  endpoint descriptors, 9.6.4
  getting descriptors, 9.4.3, 10.5.2.3
  hub descriptors, 11.15:11.15.2.1, 11.16.2.4, 11.16.2.7
  interface descriptors, 9.2.3, 9.6.3
  isochronous transfer capabilities, 5.10
  listing remote wakeup capabilities, 9.2.5.2
  overview, 9.5:9.7.3
  setting descriptors, 9.4.8, 10.5.2.12
  string descriptors, 9.6.5
  USBDI mechanisms for getting descriptors, 10.5.2.3
  vendor-specific descriptors, 9.5
deserialization of transmissions, 10.2.2
detachable cables
  cable delay, 7.1.16
  connectors and, 6.2
  detachable cable assemblies, 6.4.1
  inter-packet delay and, 7.1.18
  low-speed detachable cables, 6.4.4
  maximum capacitance, 7.1.6
  termination, 7.1.5
  voltage drop budget, 7.2.2
detached devices, 9.1.1.1, 9.1.2
detaching devices. *See* dynamic insertion and removal
detecting bus state, 11.16.2.3
detecting connect and disconnect conditions, 7.1.7.1
detecting errors. *See* error detection and handling
detecting hub and port status changes, 11.13.2, 11.13.3, 11.13.4
detecting over-current conditions, 7.2.1.2.1
detecting speed of devices. *See* speed detection
Developing International Software for Windows 95 and Windows NT, 9.6.5
device addresses, 2.0 *glossary*. *See also* addresses; devices
device classes. *See also* USB device framework
  class codes, 9.2.3
  defined, 4.8
  descriptors, 9.2.3, 9.6.1, 9.7
  device characteristics, 4.8.1
  device class definitions, 9.7
  getting class-specific descriptors, 9.5
  hub class-specific requests, 11.16.2:11.16.2.9
  interfaces and endpoint usage, 9.7.2
  requests, 9.7.3
  standard, class, and vendor information, 4.8.1
Device Class Specification for Audio Devices Revision 1.0, 9.6
device descriptors
  descriptor types, 9.4 *Table 9-5*
  device class descriptors, 9.2.3, 9.7
  GetDescriptor() request, 9.4.3
  getting class-specific descriptors, 9.5
  hubs, 11.15.1

device descriptors *(continued)*
  overview, 9.6.1
  standard definitions, 9.6.1:9.6.5
device drivers, 5.10.4.3, 10.3.1
device endpoints, 2.0 *glossary*, 5.3.1.1. *See also* endpoints
device-initiated resume. *See* remote wakeup
Device layer
  descriptors, 9.5:9.7.3
  device states, 9.1:9.1.2
  generic USB device operations, 9.2:9.2.7
  standard device requests, 9.4:9.4.11
  in USB device framework, 9
  USB device requests, 9.3:9.3.5
Device Management services, 10.5.4.2
Device release numbers, 9.6.1
DEVICE_REMOTE_WAKEUP, 9.4 *Table 9-6*
DeviceRemovable field (hub descriptors), 11.15.2.1
device resources, 2.0 *glossary*. *See also* buffers; endpoints
devices. *See also* USB device framework
  address assignment, 9.1.2, 9.2.2
  characteristics and configuration (*See also* device descriptors)
    configuration, 4.8.2.2, 9.2.3
    data-rate tolerance, 7.1.11
    descriptors, 9.5:9.7.3, 9.6.1
    device characteristics, 4.8.1
    device classes, 4.8, 9.7
    device descriptions, 4.8.2:4.8.2.1
    device speed, 7.1.5, 7.1.7.1, 11.8.2
    host role in configuration, 10.3.1
    optional endpoints, 5.3.1.2
    USBD role in configuration, 10.5.4.1:10.5.4.1.4
  data transfer, 9.2.4
    communication flow requirements, 5.3
    control transfers and, 5.5
    detailed communication flow illustrated, 5.3
    differing bus access for transfers, 5.9
    jitter budget table, 7.1.15
    response to IN transactions, 8.4.5.1
    response to OUT transactions, 8.4.5.3
    response to SETUP transactions, 8.4.5.4
    role in bulk transfers, 8.5.1
  device event timings, 7.3.2 *Table 7-11*
  devices defined, 2.0 *glossary*
  dynamic attach and detach, 9.2.1
    power distribution, 7.2.4:7.2.4.2
    removing, 10.5.2.6, 10.5.4.1.4
    USBDI mechanisms, 10.5.2.5, 10.5.2.6
  generic USB device operations, 9.2:9.2.7
  power distribution, 7.2.1, 9.2.5
    bus-powered devices, 4.3.1, 7.2.1.1
    dynamic attach and detach, 7.2.4:7.2.4.2
    high-power bus-powered functions, 7.2.1.4
    low-power bus-powered functions, 7.2.1.3
    power supply and, 4.3.1
    self-powered devices, 4.3.1, 7.2.1.2, 7.2.1.5
    suspend/resume conditions, 7.2.3
    voltage drop budget, 7.2.2
devices *(continued)*
  requests
    host communication, 10.1.1
    request errors, 9.2.7
    request processing, 9.2.6:9.2.6.5
    standard device requests, 9.4:9.4.11
    USB device requests, 9.3:9.3.5
  status
    device states, 9.1:9.1.2, 11.13.2
    getting device status, 9.4.5
    getting port status, 11.16.2.6.1.1
  types of devices
    compound devices, 4.8.2.2
    functions, 4.8.2.2
    hubs, 4.8.2.1
    mapping physical and virtual devices, 5.10.4.3
    virtual devices, 2.0 *glossary*
  in USB topology, 4.1.1.2, 5.2, 5.2.2, 9.0
device software, defined, 2.0 *glossary*
diagnostic requests, 11.16.2.3
diameter of cables, 6.6.2
dielectric withstanding voltage standards, 6.7 *Table 6-7*
Differential 1 bus state, 7.1.7
Differential 2 bus state, 7.1.7
differential data jitter, 7.3.3 *Figure 7-39*, 7.3.3 *Figure 7-42*
differential delay, 7.3.2 *Table 7-8*, 7.3.3 *Figure 7-42*
differential-ended components in upstream ports, 11.6
differential input receivers, 7.1.4, 7.1.6
differential output drivers, USBD as, 7.1.1
differential signaling, 7.1.7, 7.1.7.2
differential-to-EOP transition skew, 7.3.3 *Figure 7-40*
dimensional inspection standards, 6.7 *Table 6-7*
Direction bit, 9.3.1, 9.3.4
direction of communication flow, 5.4
  bmRequestType field, 9.3.1
  bulk transfers, 5.8.2
  bus protocol overview, 4.4
  control transfers, 5.5.2
  interrupt transfers, 5.7.2
  isochronous transfers, 5.6.2
disabled ports, 11.5, 11.5.1.4, 11.16.2.6.1, 11.16.2.6.2
Disabled state, 11.5, 11.5.1.4
disabling features, 9.4.1
discarding packets, 11.3.2
Disconnect_Detect signal/event, 11.5, 11.5.2
Disconnected state
  connect and disconnect signaling, 7.1.7.1
  downstream ports, 11.5, 11.5.1.3
  signaling levels and, 7.1.7
disconnecting devices. *See* dynamic insertion and removal
disconnect timer, 11.5.2
distortion, minimizing in SOP, 7.1.7.2
documents, applicable standards, 6.7.1
down counters in hub timing, 11.2.1

downstream ports and hubs
  downstream connectivity defined, 11.1.2.1
  downstream defined, 2.0 *glossary*
  downstream plugs, 6.2
  downstream ports defined, 4.8.2.1
  downstream port state machine, 11.5
  full- or low-speed drivers and, 7.1.2.1
  in hub architecture, 11.1.1
  hub delay, 7.3.3 *Figure 7-42*
  hub descriptors, 11.15.2.1
  hub EOP delay and EOP skew, 7.3.3 *Figure 7-43*
  input capacitance, 7.1.6
  jitter, 7.3.2 *Table 7-7*
  port state descriptions, 11.5.1:11.5.1.11
  pull-up and pull-down resistors, 11.12.1
  signaling delays, 7.1.14
  transceivers, 7.1.7
drain wires, 6.5.2, 6.6.1, 6.6.2
dribble, defined, 7.1.9
drift, 5.10.1, 5.10.3
driver characteristics
  full-speed (12Mb/S) driver characteristics, 7.1.1.1
  full-speed source electrical characteristics, 7.3.2 *Table 7-6*
  low-speed (1.5Mb/S) driver characteristics, 7.1.1.2
  low-speed source electrical characteristics, 7.3.2 *Table 7-7*
  overview, 7.1.1
drivers
  defined, 2.0 *glossary*
  role in configuration, 10.3.1
  in source-to-sink connectivity, 5.10.4.3
droop, 7.2.3, 7.2.4.1
dual pin-type receptacles, 6.9
durability standards, 6.7 *Table 6-7*
DWORD, defined, 2.0 *glossary*
dynamic insertion and removal, 9.2.1
  attaching devices, 4.6.1
  defined, 2.0 *glossary*
  detecting insertion and removal, 4.9, 9.2.1
  Hub Repeater responsibilities, 11.1
  hub support for, 11.1
  power control, 7.2.3, 7.2.4:7.2.4.2
  power-on and connection events timing, 7.1.7.1
  removing devices, 4.6.2
  USB robustness and, 4.5

E

E2PROM defined, 2.0 *glossary*
ease-of-use considerations, 1.1
edges of signals
  cable delay, 7.1.16
  data source jitter, 7.1.13.1
  edge transition density, 8.2
  hubs and edge rate control, 11.12.2
  optional edge rate control capacitors, 7.1.6
EEPROM, defined, 2.0 *glossary*
Electrical Connector/Socket Test Procedures, 6.7.1
Electrically Erasable Programmable Read Only Memory (EEPROM), 2.0 *glossary*
Electrical Performance Properties of Insulation and Jacket for Telecommunication Wire and Cable, 6.7.1
electrical specifications, 6.1, 7
  applicable documents, 6.7.1
  bus timing/electrical characteristics, 7.3.2
  cables, 6.3, 6.4:6.4.4, 6.6:6.6.5
  connectors, 6.2, 6.5:6.5.4.3
  overview, 4.2.1, 6
  PCB reference drawings, 6.9
  physical layer specifications, 7.3:7.3.3
  power distribution, 7.2.3, 7.2.4:7.2.4.2, 7.2:7.2.1.5
  signaling, 7.1:7.1.19
  standards for, 6.7, 7.3.1
  timing waveforms, 7.3.3
  USB grounding, 6.8
embedded hubs, 4.8.2.2, 5.2.3
EMI, USB grounding and, 6.8
enabled ports
  connectivity and, 11.1.2.1
  downstream ports, 11.5, 11.5.1.6
  getting port status, 11.16.2.6.1
  PORT_ENABLE bit, 11.16.2.6.1.2
  port status change bits, 11.16.2.6.2
Enabled state, 11.5, 11.5.1.6
encoding data, 7.1.8
End-of-Frame (EOF). *See* EOFs
End-of-Packet (EOP). *See* EOPs
End-of-Packet bus state, 7.1.7, 7.1.7.2
end-of-packet delimiter. *See* EOPs
ENDP field, 8.3.2.2, 8.3.5.1, 8.4.1
endpoint addresses, 2.0 *glossary*, 5.3.1, 9.6.4
endpoint descriptors, 9.4.3, 9.6.1, 9.6.3, 9.6.4
endpoint direction, defined, 2.0 *glossary*
endpoint field (ENDP), 8.3.2.2, 8.3.5.1, 8.4.1
ENDPOINT_HALT, 9.4 *Table 9-6*
endpoint numbers, 2.0 *glossary*, 5.3.1
endpoints
  addresses, 9.6.4
  characteristics, 5.3.1
  description in descriptors, 9.4.3, 9.6.1, 9.6.3, 9.6.4
  in device class definitions, 9.7.2
  direction of flow, 5.3.1
  endpoint address field, 8.3.2.2
  endpoint aliasing, 8.3.2
  endpoint zero requirements, 4.8.1, 5.3.1.1, 5.3.1.2, 5.3.2
  getting endpoint status, 9.4.5
  Hub Controller endpoint organization, 11.13.1
  in interfaces, 9.2.3, 9.6.2, 9.6.3
  logical devices as collections of endpoints, 5.3
  message pipes and, 5.3.2.2
  non-endpoint zero requirements, 5.3.1.2
  overview, 5.3.1
  pipes and, 4.4, 5.3.2
  programmable data rates, 2.0 *glossary*
  reflected endpoint status, 10.5.2.2
  role in data transfers, 4.7
  samples, 2.0 *glossary*
  specifying in *wIndex* field, 9.3.4
  stream pipes and, 5.3.2.1
  synchronization frame, 9.4.11

endpoint synchronization type, 5.10.4, 5.10.4.1
endpoint zero
  Default Control Pipe and, 5.3.2
  in device characteristics, 4.8.1
  non-endpoint zero requirements, 5.3.1.2
  requirements, 5.3.1.1
end-to-end signal delay, 7.1.19
end users, 2.0 *glossary*, 3.3
enumeration. *See* bus enumeration
environmental characteristics for cables, 6.6.4
environmental compliance standards, 6.7
EOF1 or EOF2 signal/event
  frame timer and, 11.2.1, 11.2.2
  host behavior at end-of-frame, 11.3
  in Hub Repeater state machine, 11.7
  in transmitter state machine, 11.6.2
EOFs
  defined, 2.0 *glossary*
  in frame timer synchronization, 11.2.1
  host behavior at end-of-frame, 11.3:11.3.3
  Host Controller frame generation, 10.2.3
  in transaction completion prediction, 11.3.3
EOI signal/event
  in downstream port state machine, 11.5
  in internal port state machine, 11.4
  in receiver state machine, 11.6.1
  in transmitter state machine, 11.6.2
EOP bus state, 7.1.7, 7.1.7.2
EOPs
  defined, 2.0 *glossary*
  differential-to-EOP transition skew and EOP width, 7.3.3 *Figure 7-40*
  EOP delimiter, 8.3
  EOP width, 7.1.13.2, 7.3.3 *Figure 7-40*
  error detection through bus turn-around timing, 8.7.2
  extra bits and, 7.1.9
  false EOPs, 2.0 *glossary*, 8.7.3
  handshake packets and, 8.4.4
  hub EOP delay and EOP skew, 7.3.3 *Figure 7-43*
  hub/repeater electrical characteristics, 7.3.2 *Table 7-8*
  hub signaling at EOF1, 11.3.1
  intervals between IN token and EOP, 11.3.3
  propagation delays, 7.1.14
equations
  buffering for rate matching, 5.10.8
  buffer sizes in functions and software, 5.9.4
  bus transaction times, 5.9.3
error detection and handling. *See also* corrupted transfers and requests
  babble and loss of activity recovery, 8.7.4
  bit stuff violations, 8.7.1
  bulk transfers and, 5.8.5, 8.5.1
  bus turn-around timing, 8.7.2
  control transfers and, 5.5.5, 8.5.2.1
  corrupted ACK handshake, 8.5.2.3, 8.6.4
  corrupted SOF packets in isochronous transfers, 5.10.6
  CRCs, 8.3.5, 8.7.1
  data corrupted or not accepted, 8.6.3
  error count tally, 10.2.6
error detection and handling *(continued)*
  error handling for transfers, 5.4
  error handling on last data transaction, 8.5.2.3
  false EOPs, 2.0 *glossary*, 8.7.3
  Host Controller role in, 10.2.6
  Hub Repeater responsibilities, 11.1
  hub role in, 11.1.2.3
  interrupt transfers and, 5.7.5
  isochronous transfers and, 5.6.4, 5.6.5, 5.10.7
  overview, 8.7
  packet error categories, 8.7.1
  PID check bits, 8.7.1
  Port Error conditions, 11.8.1
  Request Errors, 9.2.7
  sample size and, 5.10.8
  short packets and error conditions, 5.3.2
  synchronous data connectivity, 5.10.4.3.2
  USBD role in, 10.5.4.5
  USB robustness and, 4.5.1, 4.5.2
ESD, USB grounding and, 6.8
event notifications, USBD and, 10.5.4.4
exception handling. *See* error detection and handling
extended descriptor definitions, 9.7.1
extensibility of USB architecture, 4.10
extension cables, 6.4.4
externally-powered hubs, 7.2.1. *See also* self-powered hubs
extraction force standards, 6.7 *Table 6-7*

F

failed data transactions, 8.6.3
false EOPs, 2.0 *glossary*, 8.7.3
fault detection. *See* error detection and handling
features
  hub class feature selectors, 11.16.2
  SetFeature() request, 9.4.9
  setting hub features, 11.16.2.8
  standard feature selectors, 9.4 *Table 9-6*
feedback for isochronous transfers, 5.10.4.2
ferrite beads, 7.1.6
fields. *See names of specific fields*
flammability
  cables, 6.6.4
  Series "A" and Series "B" plugs, 6.5.4.1
  Series "A" and Series "B" receptacles, 6.5.3.1
  standards, 6.7 *Table 6-7*
flexibility of USB devices, 3.3
flow control mechanisms
  in bus protocol overview, 4.4
  handshake packets and, 8.4.4
  USB robustness and, 4.5
flyback voltage, 7.2.4.2
format of USB device requests, 9.3
formulas
  buffering for rate matching, 5.10.8
  buffer sizes in functions and software, 5.9.4
  bus transaction times, 5.9.3
frame clock, 5.10.3

- frame intervals
  - frame interval adjustment, 7.1.12
  - full-speed source electrical characteristics, 7.3.2 *Table 7-6*
  - low-speed source electrical characteristics, 7.3.2 *Table 7-7*
  - repeatability, 7.1.12
- frame number field, 8.3.3, 8.4.2
- frame numbers, 5.10.6, 8.3.3, 8.4.2, 10.2.3
- frame pattern, defined, 2.0 *glossary*
- frames. *See also* frame timers
  - allocating bandwidth, 4.7.5, 5.9.1:5.9.1.5, 10.3.2
  - available time in frames, 5.5.4, 5.6, 5.6.4, 5.7.4, 5.8.4, 5.9.5
  - bandwidth reclamation, 5.9.5
  - bit time zero, 11.3
  - control transfer reserved portions, 5.5.4
  - defined, 2.0 *glossary*
  - frame generation responsibilities of Host Controller, 10.2.3
  - frame intervals, 7.1.12, 7.3.2 *Table 7-6*, 7.3.2 *Table 7-7*
  - frame number field, 8.3.3, 8.4.2
  - frame numbers, 5.10.6, 8.3.3, 8.4.2, 10.2.3
  - frame timers, 11.2:11.2.2
  - frame wander defined, 11.2.2
  - host behavior at end-of-frame, 11.3
  - interrupt transfer limitations, 5.7.4
  - isochronous transfer limitations, 5.6.3, 5.6.4
  - Master Client status, 10.5.3.2.4, 10.5.4.2
  - one transaction per frame in isochronous transfers, 5.10.7
  - organization of transactions within, 5.9.2
  - samples per frame in isochronous transfers, 5.10.4.2
  - SOF packets, 8.4.2
  - SOF tracking, 5.10.6
- frame timers
  - frame wander, 11.2.2
  - hub frame timer, 11.2:11.2.2
  - timing skew, 11.2.2
- frame wander, defined, 11.2.2
- frequency-locked clocks, 5.10.3
- Fs. *See* SRC
- Fsus state, 11.4, 11.4.3
- full-duplex, defined, 2.0 *glossary*
- full-speed buffers, 7.1.2
- full-speed cables
  - cable delay, 7.1.16
  - captive cable assemblies, 6.4.2
  - construction, 6.6, 6.6.2
  - description, 6.6.1
  - listing, 6.6.5
  - specifications, 6.3
  - standards for, 6.6.3, 6.6.4, 6.7
- full-speed driver characteristics (12Mb/S), 7.1.1.1
- full-speed functions and hubs
  - bulk transfers and, 5.8.4
  - cable and resistor connections, 7.1.5
  - connect detection, 7.1.7.1
  - control transfers and, 5.5.3, 5.5.4, 5.5.4 *Table 5-1*
  - data-rate tolerance, 7.1.11
- full-speed functions and hubs *(continued)*
  - detachable cables and, 6.4.1
  - full-speed source electrical characteristics, 7.3.2 *Table 7-6*
  - full- *vs.* low-speed port behavior, 11.8.4
  - getting port status, 11.16.2.6.1
  - hub/repeater electrical characteristics, 7.3.2 *Table 7-8*
  - hub support for, 11.1
  - input capacitance, 7.1.6
  - interrupt transfers and, 5.7.3, 5.7.4 *Table 5-4*
  - isochronous transfers and, 5.6.4
  - optional endpoints, 5.3.1.2
  - signal termination, 7.1.5
  - SOF PID and, 8.4.2
  - speed detection and, 11.8.2
  - Transmit state and, 11.5.1.7
- full-speed signaling
  - calculating transaction times, 5.9.3
  - data rates, 4.2.1
  - data source jitter, 7.1.13.1
  - edge rates, 11.12.2
  - EOP width, 7.1.13.2
  - full-speed loads, 7.1.2
  - intervals between IN token and EOP, 11.3.3
  - J and K states, 7.1.7
  - jitter budget table, 7.1.15
  - propagation delays, 7.1.14
- Full Suspend (Fsus) state, 11.4, 11.4.3
- function address field (ADDR), 8.3.2.1
- functional stall, 8.4.4, 8.5.2.4
- Function layer
  - detailed communication flow, 5.3
  - illustrated, 5.1
  - interlayer communications model, 10.1.1
- functions. *See also* devices
  - address assignment, 9.1.2, 9.2.2
  - characteristics and configuration (*See also* device descriptors)
    - configuration, 4.8.2.2, 9.2.3
    - data-rate tolerance, 7.1.11
    - descriptors, 9.5:9.7.3, 9.6.1
    - device characteristics, 4.8.1
    - device classes, 4.8, 9.7
    - device speed, 7.1.7.1, 11.8.2
    - host role in configuration, 10.3.1
    - optional endpoints, 5.3.1.2
  - data transfer, 9.2.4
    - communication flow requirements, 5.3
    - control transfers and, 5.5
    - detailed communication flow illustrated, 5.3
    - differing bus access for transfers, 5.9
    - jitter budget table, 7.1.15
    - response to IN transactions, 8.4.5.1
    - response to OUT transactions, 8.4.5.3
    - response to SETUP transactions, 8.4.5.4
    - role in bulk transfers, 8.5.1
  - device event timings, 7.3.2 *Table 7-11*
  - devices defined, 2.0 *glossary*

  - dynamic attach and detach, 9.2.1
    - power distribution, 7.2.4:7.2.4.2
    - removing, 10.5.2.6, 10.5.4.1.4
    - USB mechanisms, 10.5.2.5, 10.5.2.6
  - generic USB device operations, 9.2:9.2.7
  - overview, 4.8.2.2
  - power distribution, 7.2.1, 9.2.5
    - bus-powered devices, 4.3.1, 7.2.1.1
    - dynamic attach and detach, 7.2.4:7.2.4.2
    - high-power bus-powered functions, 7.2.1.4
    - low-power bus-powered functions, 7.2.1.3
    - power supply and, 4.3.1
    - self-powered functions, 7.2.1.2, 7.2.1.5
    - suspend/resume conditions, 7.2.3
    - voltage drop budget, 7.2.2
  - requests
    - host communication with, 10.1.1
    - request errors, 9.2.7
    - request processing, 9.2.6:9.2.6.5
    - standard device requests, 9.4:9.4.11
    - USB device requests, 9.3:9.3.5
  - status, 9.1:9.1.2, 9.4.5
  - types of devices
    - compound devices, 4.8.2.2
    - functions, 4.8.2.2
    - hubs, 4.8.2.1
    - mapping physical and virtual devices, 5.10.4.3
    - virtual devices, 2.0 *glossary*
  - in USB topology, 4.1.1.2, 5.2.2, 5.2.3, 9.0
- function-to-host transfers. *See* IN PID

G

- gang-mode power control, 11.15.2.1
- garbling messages in Collision conditions, 11.8.3
- Generate End of Packet Towards Upstream Port state (GEOPTU), 11.6.2, 11.6.2.5
- Generate Resume state, 11.4, 11.4.4
- generic USB device operations, 9.2:9.2.7
- GEOPTU state, 11.6.2, 11.6.2.5
- GetBusState() request, 11.16.2, 11.16.2.3
- GetConfiguration() request, GET_CONFIGURATION
  - hub requests, 11.16.1
  - overview, 9.4.2
  - returning interface descriptors, 9.6.3
  - standard device request codes, 9.4
- GetDescriptor() request, GET_DESCRIPTOR, 11.15.1
  - endpoint descriptors, 9.6.4
  - GetDescriptor(CONFIGURATION) request, 9.5, 9.6.4
  - GetHubDescriptor() request, 11.16.2.4
  - hub class requests, 11.16.2
  - hub descriptors, 11.16.2.4
  - hub requests, 11.16.1
  - interface descriptors, 9.6.3
  - overview, 9.4.3
  - standard device request codes, 9.4
- GetHubDescriptor() request, 11.16.2, 11.16.2.4
- GetHubStatus() request, 11.16.2, 11.16.2.5
- GET_INTERFACE
  - hub requests, 11.16.1
  - overview, 9.4.4
  - standard device request codes, 9.4
- GetInterface() request
  - alternate settings for interfaces, 9.2.3
  - interface descriptors, 9.6.3
  - overview, 9.4.4
- GetPortStatus() request, 11.16.2, 11.16.2.6:11.16.2.6.2.5
- GET_STATE, 11.16.2, 11.16.2.3
- GetStatus() request, GET_STATUS, 9.4.5
  - GetHubStatus() request, 11.16.2.5
  - GetPortStatus() request, 11.16.2.6
  - hub class requests, 11.16.2
  - overview, 9.4.5
  - standard device request codes, 9.4
- global resumes
  - frame timer synchronization, 11.2.1
  - hub support, 11.9
  - signaling, 7.1.7.5
- global suspend, 7.1.7.4.1, 11.9
- glossary, 2.0
- GND leads
  - cable electrical characteristics, 7.3.2 *Table 7-9*
  - captive cable assemblies, 6.4.2, 6.4.3
  - detachable cables, 6.4.1
  - electrical specifications overview, 4.2.1
  - standardized contact terminating assignments, 6.5.2
- GResume state, 11.4, 11.4.4
- grounding, 6.8

H

- halted pipes, 10.5.2.2
- *Halt* feature
  - bulk transfers, 5.8.5
  - control transfers, 5.5.5, 8.5.2.4
  - functional stalls, 8.4.4
  - GetStatus() request, 9.4.5
  - interrupt transfers, 5.7.5, 8.5.3
  - isochronous transfers, 5.6.5
  - responses to standard device requests, 9.4
- handshakes, 8.3.1 *Table 8-1*. *See also* ACKs; NAKs; STALLs
  - ACK PID, 8.3.1 *Table 8-1*
  - bulk transfers, 8.5.1
  - bus protocol overview, 4.4
  - defined, 2.0 *glossary*
  - function response to IN transactions, 8.4.5.1
  - function response to OUT transactions, 8.4.5.3
  - function response to SETUP transactions, 8.4.5.4
  - handshake responses, 8.4.5, 8.4.5:8.4.5.4
  - host response to IN transactions, 8.4.5.2
  - isochronous transfers, 5.6.5, 5.10.7
  - NAK PID, 8.3.1 *Table 8-1*
  - overview, 8.4.4
  - packet field formats, 8.3:8.3.5.2
  - STALL PID, 8.3.1 *Table 8-1*
  - total allocation of bit times, 11.3.3
- hardwired cable assemblies, 6.4.2

- HCD (Host Controller Driver)
  - defined, 2.0 *glossary*, 5.3
  - HCDI (Host Controller Driver Interface), 10.1.1, 10.4
  - overview, 10.4
  - software interface overview, 10.3
  - in transfer management, 5.9.1, 5.9.1.3
  - in USB topology, 5.2.1, 10.1.1
- HCDI (Host Controller Driver Interface), 10.1.1, 10.4
- high-powered devices
  - bus-powered functions, 7.2.1, 7.2.1.4
  - high-power ports, 7.2.1
  - voltage drop budget, 7.2.2
- host, 10
  - in bus topology, 4.1.1.1, 5.2, 5.2.1
  - collecting status and activity statistics, 10.1.4
  - components, 10.1.1
  - control mechanisms, 10.1.2
    - EOF1 and EOF2 timing points, 11.2.2
    - host behavior at end-of-frame, 11.3:11.3.3
    - host-to-hub communications, 11.1
    - resource management, 10.3.2
    - responsibilities and capabilities, 10.1.1
    - role in assigning addresses, 9.2.2
    - role in configuration, 9.2.3, 10.3.1
    - synchronizing hub frame timer to host frame period, 11.2
  - data flow, 10.1.3
    - common data definitions, 10.3.4
    - data-rate tolerance, 7.1.11
    - data transfer mechanisms, 10.1.3, 10.3.3
    - detailed communication flow illustrated, 5.3
    - host response to IN transactions, 8.4.5.2
    - interlayer communications model, 10.1.1
    - role in bulk transfers, 8.5.1
  - defined, 2.0 *glossary*, 4.9
  - electrical considerations, 10.1.5
    - jitter budget table, 7.1.15
    - over-current protection, 7.2.1.2.1
    - over-current recovery, 11.13.5
  - hardware and software, 10.0
  - Host Controller Driver (HCD), 10.4 (*See also* HCD)
  - Host Controller responsibilities, 4.9, 10.2 (*See also* Host Controller)
  - host tolerance, hub frame timer and, 11.2
  - operating system environment guides, 10.6
  - overview of USB Host, 10.1:10.1.5
  - power management overview, 4.3.2
  - software mechanisms, 10.3:10.3.4
  - status in USBD pipe state, 10.5.2.2
  - Universal Serial Bus Driver (USBD), 10.5:10.5.5 (*See also* USBD (USB Driver))
  - USB System Software responsibilities, 4.9 (*See also* USB System Software)
- Host Controller, 4.9
  - in bus topology, 5.2.1
  - calculating buffer sizes in functions and software, 5.9.4
  - data transfer mechanisms, 10.1.3
    - bulk transfers, 5.8.3
    - control transfers, 5.5.3, 5.5.4
    - data processing, 10.2.4
- Host Controller
  - data transfer mechanisms *(continued)*
    - data-rate tolerance, 7.1.11
    - interrupt transfers, 5.7.3
    - role in transfer management, 5.9.1, 5.9.1.5
    - tracking transactions, 5.9.2
    - transaction list, 5.9.1.4
    - transmission error handling, 10.2.6
  - defined, 2.0 *glossary*, 4.9
  - frame generation, 10.2.3
  - HCD and HCDI overview, 10.4 (*See also* HCD; HCDI)
  - host behavior at end-of-frame, 11.3
  - host-system interface, 10.2.9
  - as implementation focus area, 5.1
  - implemented in USB Bus interface, 10.1.1
  - multiple Host Controllers, 4.10
  - passing preboot control to operating system, 10.5.5
  - port resets, 10.2.8.1
  - protocol engine, 10.2.5
  - remote wakeup and, 10.2.7
  - requirements, 10.2
  - root hub and, 10.2.8
  - serializer/deserializer, 10.2.2
  - state handling, 10.2.1
  - status and activity monitoring, 10.1.4
  - USB System interaction, 10.1.1
- Host Controller Driver. *See* HCD (Host Controller Driver)
- Host Controller Driver Interface (HCDI), 10.1.1, 10.4
- host resources, 2.0 *glossary*
- host side bus interface. *See* Host Controller
- host software
  - in bus topology, 5.2.1
  - as component of USB System, 10.1.1
  - pipes and, 10.5.1.2
  - status and activity monitoring, 10.1.4
- host-to-function transfers. *See* OUT PID
- hot plugging. *See* dynamic insertion and removal
- Hub Change field, 11.4.4, 11.16.2.5
- hub class definitions
  - additional endpoints, 11.13.1
  - feature selectors, 11.16.2
  - request codes, 11.16.2
  - root hub and, 10.4
- Hub Controller, 11.13:11.13.15
  - control commands, 11.1
  - defined, 4.8.2.1
  - endpoint organization, 11.13.1
  - hub and port change information processing, 11.13.3, 11.13.4
  - in hub architecture, 11.1.1, 11.13.2
  - internal port connection, 11.4
  - over-current reporting and recovery, 11.13.5
  - power distribution and, 7.2.1.1
  - role in host-to-hub communications, 11.1
  - status commands, 11.1
- hub descriptors, 11.13.2
- hub drivers, 10.3.1
- hub frame timer, 11.2:11.2.2
- hub oscillator tolerance, 11.2

- Hub Repeater
  - Collision conditions, 11.8.3
  - connectivity setup and tear-down, 11.1
  - defined, 4.8.2.1
  - dynamic insertion and removal, 11.1
  - electrical characteristics, 7.3.2 *Table 7-8*
  - fault detection and recovery, 11.1
  - in hub architecture, 11.1.1
  - hub signaling timings, 7.1.14
  - internal port connection, 11.4
  - packet signaling connectivity, 11.1.2.1
  - repeater state descriptions, 11.2.1, 11.7:11.7.4
  - Wait for End of Packet (WFEOP), 11.7.4
  - Wait for End of Packet from Upstream Port state (WFEOPFU), 11.7.2
  - Wait for Start of Packet (WFSOP), 11.7.3
  - Wait for Start of Packet from Upstream Port state (WFSOPFU), 11.7.1
- Hub Repeater state machine, 11.2.1, 11.7
- hubs, 11. *See also* Hub Controller; Hub Repeater; ports
  - architecture, 4.1.1.2, 11.1
  - bus states
    - bus state evaluation, 11.8:11.8.4.1
    - collision, 11.8.3
    - connect/disconnect detection, 11.1
    - full- *vs.* low-speed behavior, 11.8.4
    - getting bus state, 11.16.2.3
    - low-speed keep-alive, 7.1.7, 11.8.4.1
    - port error, 11.8.1
    - reset behavior, 11.10, 11.10.1
    - speed detection, 11.8.2
    - status change detection, 11.13.2, 11.13.4
  - in bus topology, 5.2.3, 5.2.4
  - characteristics and configuration, 11.14
    - clearing features, 11.16.2.5
    - data-rate tolerance, 7.1.11
    - descriptors, 11.13.2, 11.15, 11.15:11.15.2.1, 11.16.2.7
    - full- *vs.* low-speed behavior, 11.8.4
    - input capacitance, 7.1.6
    - speed detection of devices, 11.8.2
    - typical configuration illustrated, 4.8.2.1
  - connectivity behavior, 11.1, 11.1.2:11.1.2.3
  - controlling hubs, 7.1.7.5
  - defined, 2.0 *glossary*
  - downstream ports, 11.5:11.5.1.11
  - dynamic insertion and removal role, 4.6.1, 4.6.2
  - embedded hubs, 4.8.2.2
  - fault detection and recovery, 11.1
  - Hub Controller, 4.8.2.1, 11.1, 11.13:11.13.15 (*See also* Hub Controller)
  - hub drivers, 10.3.1
  - Hub Repeater, 4.8.2.1, 7.3.2 *Table 7-8*, 11.1, 11.7:11.7.4 (*See also* Hub Repeater)
  - hub tier, 2.0 *glossary*
  - intermediate hubs, 7.1.7.5
  - internal ports, 11.4:11.4.4
  - I/O buffer requirements, 11.12:11.12.2
  - overview, 4.8.2.1, 11.1:11.1.2.3
- hubs *(continued)*
  - power management, 11.1
    - bus-powered hubs, 4.3.1, 7.2.1.1
    - hub port power control, 11.11
    - multiple gangs, 11.11.1
    - over-current reporting and recovery, 11.13.5
    - power source and sink requirements, 7.2.1
    - self-powered hubs, 7.2.1.2
    - surge limiting, 7.2.4.1
  - requests, 11.16:11.16.2.9
  - root hubs, 2.0 *glossary*
  - signaling and timing
    - edge rate control, 11.12.2
    - host behavior at end-of-frame, 11.3:11.3.3
    - hub differential delay, 7.3.3 *Figure 7-42*
    - hub EOP delay and EOP skew, 7.3.3 *Figure 7-43*
    - hub event timings, 7.3.2 *Table 7-10*
    - hub frame timer, 11.2, 11.2:11.2.2
    - hub oscillator tolerance, 11.2
    - hub signaling timings, 7.1.14
    - hub switching skews, 7.1.9
    - jitter budget table, 7.1.15
    - low-speed keep-alive strobe, 7.1.7, 11.8.4.1
    - power-on and connection events timing, 7.1.7.1
    - reset signaling, 7.1.7.3
    - resume signaling, 7.1.7.5
    - signaling delays, 7.1.14
    - suspend and resume signaling, 11.9
    - tracking frame intervals, 7.1.12
  - upstream ports, 11.6, 11.6:11.6.2.6
- Hub State Machine, 11.1.1
- Hub Status field, 11.16.2.5
- humidity life standards, 6.7 *Table 6-7*
- hybrid powered hubs, 7.2.1.2
- hysteresis in single-ended receivers, 7.1.4

I

- *iConfiguration* field (configuration descriptors), 9.6.2
- Icon for USB plugs and receptacles, 6.5, 6.5.1
- Idle bus state
  - data signaling overview, 7.1.7.2
  - hub connectivity and, 11.1.2.1
  - Idle-to-K state transition, 7.1.14
  - NRZI data encoding, 7.1.8
  - signaling levels and, 7.1.7
- idle pipes, 5.3.2, 10.5.2.2
- *idProduct* field (device descriptors), 9.6.1
- *idVendor* field (device descriptors), 9.6.1
- *iInterface* field (interface descriptors), 9.6.3
- *iManufacturer* field (device descriptors), 9.6.1
- impedance
  - cable impedance tests, 6.7 *Table 6-7*
  - detachable cable assemblies, 6.4.1
  - differential cable impedance, 7.3.2 *Table 7-9*
  - full-speed captive cable assemblies, 6.4.2
  - full-speed connections, 7.1.1.1
  - input impedance of ports, 7.1.6
  - zero impedance voltage sources, 7.1.1
- implementer viewpoints of data flow models, 5.1
- Inactive state, 11.4, 11.4.1, 11.6.2, 11.6.2.1
- in-band signaling, 10.1.2

initial frequency inaccuracies, 7.1.11
initialization of USBD, 10.5.1.1
injection molded thermoplastic insulator material, 6.5.3.1, 6.5.4.1
inner shielding in cables, 6.6.2
IN PID, 8.3.1 *Table 8-1*
  ACK handshake and, 8.4.4
  ADDR field, 8.3.2.1
  bit times and, 11.3.3
  bulk transfers, 8.5.1
  control transfers, 8.5.2, 8.5.2.1
  ENDP field, 8.3.2.2
  error handling on last data transaction, 8.5.2.3
  function response to, 8.4.5.1
  host response to, 8.4.5.2
  interrupt transactions, 8.5.3
  intervals between IN token and EOP, 11.3.3
  isochronous transfers, 8.5.4
  low-speed transactions, 8.6.5
  NAK handshake and, 8.4.4
  prebuffering data, 5.10.5
  STALL handshake and, 8.4.4
  token CRCs, 8.3.5.1
  token packets, 8.4.1
input capacitance, 7.1.2, 7.3.2 *Table 7-5*
input characteristics (signaling), 7.1.6
input impedance of ports, 7.1.6
input levels, 7.3.2 *Table 7-5*
inputs (Series "B" receptacles), 6.2
input sensitivity of differential input receivers, 7.1.4
inrush current limiting
  bus-powered hubs, 7.2.1.1
  dynamic attach and detach, 7.2.4.1
  remote wakeup and, 7.2.3
  self-powered functions, 7.2.1.5
  suspend/resume and, 7.2.3
inserting devices. *See* dynamic insertion and removal
insertion force standards, 6.7 *Table 6-7*
instancing of USBD, 10.5
insulation
  cables, 6.6.2
  insulator materials, 6.5.3.1, 6.5.4.1
  resistance standards, 6.7 *Table 6-7*
interconnect model, 4.1, 5.10.4.3
interface class codes, 9.6.3
interface descriptors
  GetDescriptor() request, 9.4.3
  hubs, 11.15.1
  overview, 9.6.3
interface numbers, 9.2.3, 9.6.3
interfaces
  alternate interfaces, 9.2.3, 10.5.2.10
  alternate settings, 9.6.3
  in configuration descriptors, 9.6.2
  defined, 9.2.3
  in device class definitions, 9.7.2
  as endpoint sets, 5.3
  getting interface status, 9.4.4, 9.4.5
  host-system interface, 10.2.9
  interface class codes, 9.6.3
  interface descriptors, 9.4.3, 9.6.3, 11.15.1
  interface numbers, 9.2.3, 9.6.3
interfaces *(continued)*
  interface subclass codes, 9.6.3
  plug interface and mating drawings, 6.5.4
  setting interface state, 9.4.10, 10.5.2.1
  specifying in *wIndex* field, 9.3.4
interfaces of plugs, 6.5.3
interface state control, 10.5.2.1
interface subclass codes, 9.6.3
interlayer communications model, 4.1, 10.1.1
intermediate hubs, 7.1.7.5
internal clock source jitter, 7.1.13.1
internal ports
  Full Suspend (Fsus) state, 11.4.3
  Generate Resume (GResume) state, 11.4.4
  Inactive state, 11.4.1
  Suspend Delay state, 11.4.2
internal port state machine, 11.4
inter-packet delay, 7.1.18
interrupt endpoints, 11.13.1
interrupt requests, defined, 2.0 *glossary*
interrupt transfers
  bus access constraints, 5.7.4
  data format, 5.7.1
  data sequences, 5.7.5
  defined, 2.0 *glossary*, 5.4
  direction, 5.7.2
  full-speed transfer limits, 5.7.4 *Table 5-4*
  low-speed transfer limits, 5.7.4 *Table 5-4*
  overview, 4.7.3, 5.7
  packet size, 5.7.3, 9.6.4
  transaction format, 8.5.3
  transaction organization within IRPs, 5.9.2
  USBD pipe mechanism responsibilities, 10.5.3.1.2
intervals
  debounce intervals in connection events, 7.1.7.1
  frame intervals, 7.1.12, 7.3.2 *Table 7-6*, 7.3.2 *Table 7-7*
  Resetting state and Resuming state intervals, 11.5.1.9
  resume and recovery intervals for devices, 9.2.6.2
  service and polling intervals, 2.0 *glossary*, 9.6.4, 10.3.3
  timeout intervals, 8.7.3
  between IN token and EOP, 11.3.3
interwoven tinned copper wire, 6.6.2
I/O buffers. *See* buffers
I/O Request Packets. *See* IRPs
iProduct field (device descriptors), 9.6.1
IRPs. *See also* requests; transfers
  aborting/retiring, 5.3.2, 10.5.3.2.1
  class-specific requests, 9.2.6.5
  client software role in, 5.9.1.1
  defined, 2.0 *glossary*, 5.3.2
  HCD tracking of, 5.9.1.3
  multiple data payloads in, 5.3.2
  pipes and, 5.3.2
  queuing IRPs, 10.5.3.2.3
  request processing overview, 9.2.6
  reset/resume recovery time, 9.2.6.2
  set address processing, 9.2.6.3
  STALLS and, 5.3.2
  standard device requests, 9.2.6.4, 9.4:9.4.11

- IRPs *(continued)*
  - timing, 9.2.6.1, 9.2.6.3, 9.2.6.4
  - transaction organization within IRPs, 5.9.2
  - USB device requests, 9.3:9.3.5
  - USBD role in, 10.1.1
- IRQs, defined, 2.0 *glossary*
- *iSerialNumber* field (device descriptors), 9.6.1
- isochronous data, defined, 2.0 *glossary*
- isochronous devices, defined, 2.0 *glossary*
- isochronous sink endpoints, defined, 2.0 *glossary*
- isochronous source endpoints, defined, 2.0 *glossary*
- isochronous transfers
  - buffering for rate matching, 5.10.8
  - bus access constraints, 5.6.4
  - clock model, 5.10.2
  - clock synchronization, 5.10.3
  - connectivity, 5.10.4.3
  - data format, 5.6.1
  - data prebuffering, 5.10.5
  - data sequences, 5.6.5
  - defined, 2.0 *glossary*, 5.4
  - direction, 5.6.2
  - endpoint synchronization frame, 9.4.11
  - error handling, 5.10.7
  - feedback, 5.10.4.2
  - illustrated, 5.10.2
  - isochronous device framework, 5.10.4
  - non-USB example isochronous application, 5.10.1
  - overview, 4.7.4, 5.6
  - packet size, 5.6.3, 9.6.4
  - SOF tracking, 5.10.6
  - special considerations, 5.10:5.10.8
  - synchronization types, 5.10.4.1.1, 5.10.4.1.2, 5.10.4.1.3
  - transaction format, 8.5.4
  - transaction organization within IRPs, 5.9.2
  - USBD pipe mechanism responsibilities, 10.5.3.1.1
  - USB features and, 3.3
  - USB System Software role, 4.9
- ISO transfers. *See* isochronous transfers
- ITCW (interwoven tinned copper wire), 6.6.2

J

- jacketing in cables, 6.6.2
- J bus state
  - data signaling overview, 7.1.7.2
  - J-to-K state transition, 7.1.14
  - NRZI data encoding, 7.1.8
  - signaling levels and, 7.1.7
- jitter
  - clock jitter, 5.10.3
  - data source jitter, 7.1.13.1
  - defined, 2.0 *glossary*
  - differential data jitter, 7.3.3 *Figure 7-39*
  - differential jitter, 7.3.3 *Figure 7-42*
  - full-speed source electrical characteristics, 7.3.2 *Table 7-6*
  - hub/repeater electrical characteristics, 7.3.2 *Table 7-8*
  - internal clock source jitter, 7.1.13.1
  - low-speed source electrical characteristics, 7.3.2 *Table 7-7*
- jitter (continued)
  - in non-USB isochronous application, 5.10.1
  - output driver jitter, 7.1.15
  - receiver data jitter, 7.1.15, 7.3.3 *Figure 7-41*
  - service jitter, 2.0 *glossary*
- J signal/event
  - differential transmissions, 11.6
  - downstream port state machine, 11.5
  - receiver state machine, 11.6.1
  - transmitter state machine, 11.6.2

K

- kB/S and kb/S, defined, 2.0 *glossary*
- K bus state
  - data signaling overview, 7.1.7.2
  - Idle-to-K state transition, 7.1.14
  - K-to-J state transition, 7.1.14
  - NRZI data encoding, 7.1.8
  - signaling levels and, 7.1.7
- keep-alive strobe, 7.1.7, 7.1.7.4, 11.8.4.1
- keyed connector protocol, 6.2
- K signal/event
  - differential transmissions, 11.6
  - downstream port state machine, 11.5
  - receiver state machine, 11.6.1
  - transmitter state machine, 11.6.2

L

- LANGID code array, 9.6.5
- language IDs in string descriptors, 9.6.5
- latency constraints for transfers, 5.4
- latest host packet, 11.3.1
- least-significant bit, 2.0 *glossary*, 8.1
- length of cables, 6.4.1, 6.4.2, 6.4.3
- listing, UL listing for cables, 6.6.5
- little endian, defined, 2.0 *glossary*
- LOA, 2.0 *glossary*, 8.7.4
- load capacitance, 6.4.3, 6.7 *Table 6-7*
- Local Power Source field, 11.16.2.5
- Local Power Status Change field, 11.16.2.5
- Local Power Status field, 11.16.2.6.1.6
- local power supplies, 7.2.1.2, 7.2.1.5
- locking hub frame timer, 11.2.1
- Lock signal/event, 11.7
- logical bus topology, 5.2, 5.2.4
- logical devices
  - in bus topology, 5.2.2
  - as collections of endpoints, 5.3
  - detailed communication flow illustrated, 5.3
  - unique addresses and endpoints, 5.3.1
- Logical Power Switching Mode field, 11.11, 11.11.1, 11.15.2.1
- logo location on connectors, 6.5.1
- LOI, 6.5.3.1, 6.5.4.1
- loss of bus activity. *See* LOA
- low level contact resistance standards, 6.7 *Table 6-7*
- low-power bus-powered functions, 7.2.1, 7.2.1.3
- low-power hubs, 7.2.2
- low-power ports, 7.2.1
- low-speed buffers, 7.1.2

low-speed cables
  cable environmental characteristics, 6.6.4
  captive cable assemblies, 6.4.3
  configuration overview, 6.6
  construction, 6.6.2
  description, 6.6.1
  detachable cables, 6.4.4
  listing, 6.6.5
  specifications, 6.3
  standards for, 6.6.3, 6.7
low-speed driver characteristics (1.5Mb/S), 7.1.1.2
low-speed functions and hubs
  cable and resistor connections, 7.1.5
  connect detection, 7.1.7.1
  control transfers and, 5.5.3, 5.5.4, 5.5.4 *Table 5-1*
  data-rate tolerance, 7.1.11
  detachable cables and, 6.4.1
  detecting, 11.16.2.6.1.7
  full-speed captive cable assemblies, 6.4.2
  full- *vs.* low-speed port behavior, 11.8.4
  getting port status, 11.16.2.6.1
  hub/repeater electrical characteristics, 7.3.2 *Table 7-8*
  hub support for, 11.1
  input capacitance, 7.1.6
  interrupt transfers and, 5.7.3, 5.7.4 *Table 5-4*
  low-speed captive cable assemblies, 6.4.3
  optional endpoints, 5.3.1.2
  signal termination, 7.1.5
  speed detection and, 11.8.2
  Transmit state and, 11.5.1.7
low-speed keep-alive strobe, 7.1.7, 7.1.7.4, 11.8.4.1
low-speed signaling
  calculating transaction times, 5.9.3
  data rates, 4.2.1
  data source jitter, 7.1.13.1
  data toggle synchronization and retry, 8.6.5
  edge rates, 11.12.2
  EOP width, 7.1.13.2
  intervals between IN token and EOP, 11.3.3
  J and K states, 7.1.7
  jitter budget table, 7.1.15
  low-speed loads, 7.1.2
  propagation delays, 7.1.14
  transactions illustrated, 8.6.5
low-speed source electrical characteristics, 7.3.2 *Table 7-7*
LSb and LSB
  in bit ordering, 8.1
  defined, 2.0 *glossary*
LS signal/event, 11.5

M

male plug contact materials, 6.5.4.3
Manufacturer's logo location, 6.5.1
Manufacturer's names in device descriptors, 9.6.1
mapping physical and virtual devices, 5.10.4.3
marking on cables, 6.6.2
Master Client status, 5.10.4.1.1, 10.5.3.2.4, 10.5.4.2
master clock, 5.10.1
mastership control over SOF, 5.10.4.1.1, 10.5.3.2.4, 10.5.4.2
material requirements
  cables, 6.6:6.6.5
  connectors, 6.5:6.5.4.3
mating area materials, 6.5.3.3, 6.5.4.3
MaxPower field, 9.6.2, 11.14
MB/S, defined, 2.0 *glossary*
Mb/S, defined, 2.0 *glossary*
mechanical specifications, 6, 6.1
  applicable documents, 6.7.1
  architectural overview, 6.1
  cable assembly, 6.4:6.4.4
  cables, 6.3, 6.6:6.6.5
  connectors, 6.2, 6.5:6.5.4.3
  overview, 4.2.2, 6
  PCB reference drawings, 6.9
  standards for, 6.7, 6.7.1
  USB grounding, 6.8
message pipes. *See also* control pipes; pipes
  in bus protocol overview, 4.4
  defined, 2.0 *glossary*, 5.3.2
  overview, 5.3.2.2
microphone non-USB isochronous application, 5.10.1
microphone USB isochronous application, 5.10.2
modifying device configuration, 10.5.4.1.3
monotonic transitions, 7.1.2
most-significant bit, 2.0 *glossary*, 8.1
MSb and MSB, 2.0 *glossary*, 8.1
multiple gangs, hubs and, 11.11.1

N

NAKs
  in bulk transfers, 8.5.1
  busy endpoints, 5.3.2
  in control transfers, 8.5.2.1
  data corrupted or not accepted, 8.6.3
  defined, 2.0 *glossary*
  function response to IN transactions, 8.4.5.1
  function response to OUT transactions, 8.4.5.3
  in interrupt transfers, 5.7.4, 8.5.3
  overview, 8.3.1 *Table 8-1*, 8.4.4
NEC Article 800 for communications cables, 6.6.4
next frame in hub timing, 11.2.1
No Acknowledge packet. *See* NAKs
nominal cable diameter, 6.6.2
nominal cable temperatures, 6.6.4
nominal twist ratio in signal pair, 6.6.2
non-acceptable cables, 6.4.4
Non Return to Zero Invert. *See* NRZI encoding
non-twisted power pair in cables, 6.6.1, 6.6.2
non-USB isochronous application, 5.10.1
Not Configured state, 11.5, 11.5.1.1
NRZI encoding, 7.1.8
  bit stuffing, 7.1.9
  defined, 2.0 *glossary*
  in electrical specifications overview, 4.2.1
  sync pattern, 7.1.10

O

objects, defined, 2.0 *glossary*
offsets between host and hub, 11.2
one-way propagation delay, 7.1.16, 7.3.2 *Table 7-6*, 7.3.2 *Table 7-7*

open architecture, USB development and, 1.2
open-circuit voltage, 7.1.1
operating systems
  companion specifications, 10.6
  device configuration, 10.3.1
  interaction with USBD, 10.5
  passing preboot control to, 10.5.5
operating temperatures for cables, 6.6.4
operations, generic USB device operations, 9.2:9.2.7
optional edge rate control capacitors, 7.1.6
out-of-band signaling, 10.1.2
OUT PID, 8.3.1 *Table 8-1*
  ACK handshake and, 8.4.4
  ADDR field, 8.3.2.1
  in bulk transfers, 8.5.1
  in control transfers, 8.5.2, 8.5.2.1
  in data toggle, 8.6.1
  ENDP field, 8.3.2.2
  function response to OUT transactions, 8.4.5.3
  in interrupt transfers, 8.5.3
  in isochronous transfers, 8.5.4
  NAK handshake and, 8.4.4
  prebuffering data, 5.10.5
  STALL handshake and, 8.4.4
  token CRCs, 8.3.5.1
  in token packets, 8.4.1
output driver jitter, 7.1.15
output levels, 7.3.2 *Table 7-5*
output receptacles, 6.2
output rise and fall times, 7.1.2
outside plating, 6.5.3.2, 6.5.4.2
over-current conditions
  C_PORT_OVER-CURRENT bit, 11.16.2.6.2.4
  getting port status, 11.16.2.6.1
  over-current gangs, 11.11.1
  over-current protection in self-powered hubs, 7.2.1.2.1
  PORT_OVER-CURRENT bit, 11.16.2.6.1.4
  port status change bits, 11.16.2.6.2
  protection mode descriptors, 11.15.2.1
  reporting and recovery, 11.13.5
  signaling, 11.11.1
Over-Current Indicator Change field, 11.16.2.5
Over-Current Indicator field, 11.16.2.5
Over-current Reporting Mode field, 11.11.1
Over-current signal/event, 11.5
over-sampling state machine DPLLs, 7.1.15
oxygen index, 6.5.3.1, 6.5.4.1

P

packet buffers, defined, 2.0 *glossary*
packet field formats, 8.3
  address fields, 8.3.2:8.3.2.2
  cyclic redundancy checks (CRC), 8.3.5:8.3.5.2
  data field, 8.3.4, 8.4.3
  frame number field, 8.3.3, 8.4.2
  packet identifier field, 8.3.1 (*See also* PIDs)
packet formats
  data packets, 4.4, 8.3.5.2, 8.3:8.3.5.2, 8.4.3, 8.5.1, 8.5.4
  handshake packets, 8.4.4 (*See also* handshakes)
  handshake responses, 8.4.5:8.4.5.4
packet formats *(continued)*
  overview, 8.4
  packet field formats, 8.3:8.3.5.2
  SOF packets, 5.10.2, 5.10.4.1.1, 5.10.4.1.2, 8.4.2
  token packets, 4.4, 8.3.5.1, 8.4.1, 8.5.1, 8.5.4
packet identifier field (PID). *See* PIDs
packet IDs. *See* PIDs
packet nullification, 11.3.2
packets. *See also* packet field formats; packet formats; packet size
  bit stuffing, 7.1.9
  blocking in Collision conditions, 11.8.3
  bus protocol overview, 4.4
  data packets defined, 4.4
  data signaling overview, 7.1.7.2
  defined, 2.0 *glossary*
  error detection and recovery, 8.7:8.7.4
  handshake packets defined, 4.4
  hub connectivity and, 11.1.2.1
  inter-packet delay, 7.1.18
  one transaction per frame in isochronous transfers, 5.10.7
  packet field formats, 8.3:8.3.5.2
  packet formats, 8.4:8.4.5.4
  packet nullification, 11.3.2
  packet size (*See* packet size)
  packet voltage levels, 7.1.7.2
  short packets, 5.3.2
  splitting samples across packets, 5.10.8
  SYNC field, 8.2
  token packets defined, 4.4
  transaction formats, 8.5:8.5.4
packet size
  in buffering calculations, 5.10.8
  bulk transfer constraints, 5.8.3
  characteristics for transfers, 5.4
  control transfer constraints, 5.5.3
  determining missing packet size, 5.10.7
  in device descriptors, 9.6.1
  in endpoint descriptors, 9.6.4
  interrupt transfer constraints, 5.7.3
  isochronous transfer constraints, 5.6.3
packet voltage levels, 7.1.7.2
partitioning of power, 7.2.1.1
pattern synchronization, 9.4.11
PBT (polybutylene terephthalate), 6.5.3.1, 6.5.4.1
PCB reference drawings, 6.9
PC industry, USB and, 3.3
PC-to-telephone interconnects, 1.1
performance criteria for electrical, mechanical and environmental compliance, 6.7
period transfers. *See* interrupt transfers; isochronous transfers
peripheral devices, 4.8.2.2. *See also* devices; functions
per-port current limiting, 11.13.5
per-port power switching, 11.11, 11.13.5
PET (polyethylene terephthalate), 6.5.3.1, 6.5.4.1
phase delay for SOF packets, 11.3.1
phase differences, clock synchronization and, 5.10.3
phase-locked clocks, 5.10.3
Phase Locked Loop, defined, 2.0 *glossary*

phases, defined, 2.0 *glossary*
Physical and Environmental Performance Properties of Insulation and Jacket for Telecommunication Wire and Cable, 6.7.1
physical bus topology, 5.2, 5.2.3, 6.1
physical devices
  connectivity illustrated, 5.10.4.3
  defined, 2.0 *glossary*
  as implementation focus area, 5.1
  logical components in bus topology, 5.2.2
physical interface, 4.2:4.2.2
physical shock standards, 6.7 *Table 6-7*
PID errors, defined, 8.3.1
PIDs
  corrupted PIDs, 8.3.1
  in data packets, 8.4.3
  data PIDs
    DATA0/DATA1 PIDs, 8.3.1 *Table 8-1*
      bulk transfers, 5.8.5
      in bulk transfers, 8.5.1
      comparing sequence bits, 8.6.2
      in control transfers, 8.5.2
      in data packets, 8.4.3
      in interrupt transactions, 8.5.3
      interrupt transfers and, 5.7.5
      synchronization and, 8.6
  defined, 2.0 *glossary*
  full- *vs.* low-speed port behavior, 11.8.4
  in handshake packets, 8.4.4
  handshake PIDs, 8.3.1 *Table 8-1*
    ACK PID
      in bulk transfers, 8.5.1
      in control transfers, 8.5.2, 8.5.2.1
      corrupted ACK handshake, 8.5.2.3, 8.6.4
      in data toggle, 8.6, 8.6.1, 8.6.2
      defined, 2.0 *glossary*
      function response to OUT transactions, 8.4.5.3
      host response to IN transactions, 8.4.5.2
      overview, 8.3.1 *Table 8-1*, 8.4.4
      in request processing, 9.2.6
    NAK PID
      in bulk transfers, 8.5.1
      busy endpoints, 5.3.2
      in control transfers, 8.5.2.1
      data corrupted or not accepted, 8.6.3
      defined, 2.0 *glossary*
      function response to IN transactions, 8.4.5.1
      function response to OUT transactions, 8.4.5.3
      in interrupt transfers, 5.7.4, 8.5.3
      overview, 8.3.1 *Table 8-1*, 8.4.4
    STALL PID
      in bulk transfers, 5.8.5, 8.5.1
      in control transfers, 8.5.2.1
      data corrupted or not accepted, 8.6.3
      functional and commanded stalls, 8.4.4
      function response to IN transactions, 8.4.5.1
      function response to OUT transactions, 8.4.5.3
      in interrupt transfers, 5.7.5, 8.5.3
      overview, 8.3.1 *Table 8-1*, 8.4.4
PIDs
  handshake PIDs
    STALL PID *(continued)*
      protocol stalls, 8.4.4
      Request Error responses, 9.2.7
      responses to standard device requests, 9.4
      returned by control pipes, 8.5.2.4
  overview, 8.3.1
  PID check bits, 8.7.1
  PID errors, 8.3.1
  special PIDs
    PRE PID
      inter-packet delays and, 7.1.18
      low-speed port behavior and, 11.8.4
      low-speed transactions, 8.6.5
      overview, 8.3.1 *Table 8-1*
      Transmit state and, 11.5.1.7
  in start-of-frame packets, 8.4.2
  in token packets, 8.4.1
  token PIDs, 8.3.1 *Table 8-1*
    OUT PID
      ACK handshake and, 8.4.4
      ADDR field, 8.3.2.1
      in bulk transfers, 8.5.1
      in control transfers, 8.5.2, 8.5.2.1
      in data toggle, 8.6.1
      ENDP field, 8.3.2.2
      function response to OUT transactions, 8.4.5.3
      in interrupt transfers, 8.5.3
      in isochronous transfers, 8.5.4
      NAK handshake and, 8.4.4
      overview, 8.3.1 *Table 8-1*
      prebuffering data, 5.10.5
      STALL handshake and, 8.4.4
      token CRCs, 8.3.5.1
      in token packets, 8.4.1
    IN PID
      ACK handshake and, 8.4.4
      ADDR field, 8.3.2.1
      bit times and, 11.3.3
      bulk transfers, 8.5.1
      control transfers, 8.5.2, 8.5.2.1
      ENDP field, 8.3.2.2
      error handling on last data transaction, 8.5.2.3
      function response to, 8.4.5.1
      host response to, 8.4.5.2
      interrupt transactions, 8.5.3
      intervals between IN token and EOP, 11.3.3
      isochronous transfers, 8.5.4
      low-speed transactions, 8.6.5
      NAK handshake and, 8.4.4
      overview, 8.3.1 *Table 8-1*
      prebuffering data, 5.10.5
      STALL handshake and, 8.4.4
      token CRCs, 8.3.5.1
      token packets, 8.4.1

PIDs
  token PIDs *(continued)*
    SETUP PID
      ACK handshake and, 8.4.4
      ADDR field, 8.3.2.1
      in control transfers, 8.5.2
      in data toggle, 8.6.1
      ENDP field, 8.3.2.2
      function response to SETUP transactions, 8.4.5.4
      overview, 8.3.1 *Table 8-1*
      token CRCs, 8.3.5.1
      in token packets, 8.4.1
    SOF PID (*See also* SOFs)
      frame number field, 8.3.3
      overview, 8.3.1 *Table 8-1*
      start-of-frame packets, 8.4.2
pins
  dual pin-type receptacles, 6.9
  inrush current and, 7.2.4.1
  single pin-type receptacles, 6.9
pipes
  aborting or resetting, 10.5.2.2, 10.5.3.2.1
  active, stalled, or idle status, 10.5.2.2
  allocating bandwidth for, 4.7.5
  characteristics and transfer types, 5.4
  client pipes, 10.5.1.2.2
  data transfer mechanisms, 10.1.3
  Default Control Pipe, 4.4 (*See also* Default Control Pipe)
  default pipes, 10.5.1.2.1
  defined, 2.0 *glossary*, 4.4
  in device characteristics, 4.8.1
  identification, 10.3.4
  Master Client status, 5.10.4.1.1, 10.5.3.2.4, 10.5.4.2
  overview, 5.3.2, 10.5.1, 10.5.3
  pipe state control, 10.5.2.2
  pipe usage, 10.5.1.2
  Policies, 10.3.1, 10.3.3, 10.5.3.2.2
  queuing IRPs, 10.5.3.2.3
  role in data transfers, 4.7
  service and polling intervals, 10.3.3
  stream and message pipes, 4.4, 5.3.2, 5.3.2.1, 5.3.2.2
  supported pipe types, 10.5.3.1:10.5.3.1.4
  USBD pipe mechanism responsibilities, 10.5.1:10.5.3.2.4
  USB robustness and, 4.5
pipe state control, 10.5.2.2
pipe status, 10.5.2.2
plating
  plug contact materials, 6.5.4.3
  plug shell materials, 6.5.4.2
  receptacle contact materials, 6.5.3.3
  receptacle shell materials, 6.5.3.2
PLL, defined, 2.0 *glossary*
plugs
  interface and mating drawings, 6.5.3
  keyed connector protocol, 6.2
  materials, 6.5.4.1, 6.5.4.2, 6.5.4.3
  orientation, 6.5.1
plugs *(continued)*
  Series "A" and Series "B" plugs, 6.5.4
  standards for, 6.7
  termination data, 6.5.2
  USB Icon, 6.5, 6.5.1
Policies
  defined, 10.3.1
  setting, 10.3.3
  USBDI mechanisms, 10.5.3.2.2
polling
  defined, 2.0 *glossary*
  endpoints, 9.6.4
  setting intervals for pipes, 10.3.3
polybutylene terephthalate (PBT), 6.5.3.1, 6.5.4.1
polyethylene terephthalate (PET), 6.5.3.1, 6.5.4.1
POR signal/event
  Bus_Reset state and, 11.6.1.6
  defined, 2.0 *glossary*
  in receiver state machine, 11.6.1
Port Change field, 11.4.4, 11.16.2.6.2
PORT_CONNECTION
  defined, 11.16.2.6.1.1
  hub class feature selectors, 11.16.2
  Port Status field, 11.16.2.6.1
PORT_ENABLE
  clearing, 11.16.2.2
  defined, 11.16.2.6.1.2
  hub class feature selectors, 11.16.2
  Port Error condition, 11.16.2.6.2.2
  Port Status field, 11.16.2.6.1
Port Error condition, 11.8.1, 11.16.2.6.2.2
Port_Error signal/event, 11.5
PORT_LOW_SPEED
  defined, 11.16.2.6.1.7
  hub class feature selectors, 11.16.2
  Port Status field, 11.16.2.6.1
  speed detection and, 11.8.2
PORT_OVER_CURRENT
  defined, 11.16.2.6.1.4
  hub class feature selectors, 11.16.2
  over-current conditions and, 11.11.1, 11.13.5
  Port Status field, 11.16.2.6.1
PORT_POWER
  clearing, 11.16.2.2
  defined, 11.16.2.6.1.6
  hub class feature selectors, 11.16.2
  Port Status field, 11.16.2.6.1
  SetPortFeature() request, 11.16.2.9
  shared power switching and, 11.11.1
PortPwrCtrlMask field
  hub descriptors for, 11.15.2.1
  multiple gangs and, 11.11.1
  power switching settings, 11.11
PORT_RESET
  defined, 11.16.2.6.1.5
  hub class feature selectors, 11.16.2
  Port Status field, 11.16.2.6.1
  SetPortFeature() request, 11.16.2.9

ports. *See also* hubs
  in bus topology, 5.2.3
  clearing features, 11.16.2.2
  data source signaling, 7.1.13
  defined, 4.8.2.1
  disconnect timer, 11.5.2
  downstream ports, 4.8.2.1, 11.5:11.5.1.11
  full- *vs.* low-speed port behavior, 11.8.4
  getting bus state, 11.16.2.3
  in hub architecture, 11.1.1
  hub configuration and, 11.14
  hub descriptors, 11.15:11.15.2.1
  hub internal ports, 11.4:11.4.4
  input capacitance, 7.1.6
  over-current reporting and recovery, 11.13.5
  port expansion considerations in USB development, 1.1
  power control, 11.11
  resetting, 10.2.8.1
  root ports, 2.0 *glossary*
  setting port features, 11.16.2.9
  status
    bus state evaluation, 11.8:11.8.4.1
    detecting status changes, 11.13.2, 11.13.3
    hub and port status change bitmap, 11.13.4
    port status bits, 11.16.2.6.1:11.16.2.6.2.5
  upstream ports, 4.8.2.1, 11.6:11.6.2.6
port status change bits, 11.16.2.6.1:11.16.2.6.2.5
Port Status field, 11.16.2.6.1
PORT_SUSPEND
  clearing, 11.16.2.2
  defined, 11.16.2.6.1.3
  hub class feature selectors, 11.16.2
  Port Status field, 11.16.2.6.1
  SetPortFeature() request, 11.16.2.9
power budgeting, 7.2.1.4, 9.2.5.1
power conductors in cables, 6.3
power control circuits in bus-powered hubs, 7.2.1.1
power distribution and management. *See also* over-current conditions; power switching
  classes of devices, 7.2.1:7.2.1.5
    bus-powered devices or hubs, 4.3.1, 7.2.1.1
    high-power bus-powered functions, 7.2.1.4
    low-power bus-powered functions, 7.2.1.3
    self-powered devices or hubs, 4.3.1, 7.2.1.2, 7.2.1.5
  configuration characteristics
    hub descriptors for power-on sequence, 11.15.2.1
    information in device characteristics, 4.8.1
    power consumption in configuration descriptors, 9.6.2
    power source capability in configuration, 9.1.1.2
  dynamic attach and detach, 7.2.3, 7.2.4:7.2.4.2
  Host Controller role in, 4.9
  host role in, 10.1.5
  hub support for, 11.1
  loss of power, 7.2.1.2
  over-current conditions, 7.2.1.2.1, 11.13.5 (*See also* over-current conditions)
  overview, 4.3.1, 4.3.2, 9.2.5
  power budgeting, 7.2.1.4, 9.2.5.1
power distribution and management *(continued)*
  power status
    control during suspend/resume, 7.2.3
    device states, 9.1.1.2
    port power states, 11.16.2.6.1.6, 11.16.2.9
  power switching, 11.11, 11.11.1
  remote wakeup, 7.2.3, 9.2.5.2
  USB System role, 10.5.4.3
  USB System Software role, 4.9
  voltage drop budget, 7.2.2
Powered device state, 9.1.1 *Table 9-1*, 9.1.1.2
Powered Off state, 11.5, 11.5.1.2
powered-on ports, 11.16.2.9
Power On Reset. *See* POR signal/event
power-on sequence, 11.15.2.1
power pair construction, 6.6.2
power pins, 7.2.4.1
Power_source_off signal/event, 11.5
power switching
  bus-powered hubs, 7.2.1.1
  getting port status, 11.16.2.6.1
  hub descriptors for, 11.15.2.1
  hub port power control, 11.11
  power-on and connection events timing, 7.1.7.1
  power switching gangs, 11.11.1
  staged power switching, 7.2.1.4
preamble packet. *See* PRE PID
preboot control, passing to operating systems, 10.5.5
prebuffering data, 5.10.5
prepared termination, 6.4.2, 6.4.3
PRE PID, 8.3.1 *Table 8-1*
  inter-packet delays and, 7.1.18
  low-speed port behavior and, 11.8.4
  low-speed transactions, 8.6.5
  Transmit state and, 11.5.1.7
product descriptions in device descriptors, 9.6.1
Product IDs in device descriptors, 9.6.1
programmable data rate, defined, 2.0 *glossary*
prohibited cable assemblies, 6.4.4
propagation delay
  cable delay, 7.1.16
  detachable cables, 6.4.1
  end-to-end signal delay, 7.1.19
  full- and low-speed signals, 7.1.14
  full-speed cables, 6.4.2
  full-speed source electrical characteristics, 7.3.2 *Table 7-6*
  low-speed cables, 6.4.3, 7.1.1.2
  low-speed source electrical characteristics, 7.3.2 *Table 7-7*
  tests, 6.7 *Table 6-7*
propagation delay skew, 6.7 *Table 6-7*
protected fields in packets, 8.3.5
protocol codes
  defined, 9.2.3
  in device descriptors, 9.6.1
  in interface descriptors, 9.6.3
protocol engine requirements of Host Controller, 10.2.5
protocol errors, detecting, 10.2.6
Protocol field, 9.2.3

Protocol layer, 8
  bit ordering, 8.1
  bus protocol, 4.4
  data toggle synchronization and retry, 8.6:8.6.5
  error detection and recovery, 8.7:8.7.4
  packet field formats, 8.3:8.3.5.2
  packet formats, 8.4:8.4.5.4
  SYNC field, 8.2
  transaction formats, 8.5:8.5.4
protocols
  defined, 2.0 *glossary*
  protocol codes, 9.2.3, 9.6.1, 9.6.3
protocol stall, 8.4.4, 8.5.2.4
pull-up and pull-down resistors
  buffer impedance measurement, 7.1.1.1
  hub requirements, 11.12.1
  power control during suspend/resume, 7.1.7.4, 7.2.3
  signal termination, 7.1.5

Q

queuing IRPs, 10.5.3.2.3

R

RA (rate adaptation)
  asynchronous RA, 2.0 *glossary*
  audio connectivity and, 5.10.4.3.1
  defined, 2.0 *glossary*
  in source-to-sink connectivity, 5.10.4.3
  synchronous data connectivity, 5.10.4.3.2
  synchronous RA, 2.0 *glossary*
random vibration standards, 6.7 *Table 6-7*
rate adaptation. *See* RA (rate adaptation)
rate matchers
  asynchronous endpoints, 5.10.4.1.1
  buffering for rate matching, 5.10.8
  client software role, 5.10.4.3
  in non-USB isochronous application, 5.10.1
  synchronous endpoints, 5.10.4.1.2
ratings, full-speed, 6.4.1, 6.4.2
read/write sequences
  in bulk transfers, 8.5.1
  in control transfers, 8.5.2, 8.5.2.1
real-time clock, 5.10.1
real-time data transfers. *See* isochronous transfers
receivers
  receive phase of signaling, 7.1.1
  receiver characteristics, 7.1.4
  receiver jitter, 7.1.15, 7.3.2 *Table 7-6*, 7.3.2 *Table 7-7*, 7.3.3 *Figure 7-41*
  receiver sequence bits, 8.6, 8.6.2
  receiver state descriptions, 11.6.1
  receiver state machine, 11.6, 11.6.1
receiver state machine, 11.6, 11.6.1
ReceivingJ state, 11.6.1, 11.6.1.1
ReceivingK state, 11.6.1, 11.6.1.3
ReceivingSE0 state, 11.6.1, 11.6.1.5
receptacles
  interface and mating drawings, 6.5.3
  keyed connector protocol, 6.2
  materials, 6.5.3.1, 6.5.3.2, 6.5.3.3
  PCB reference drawings, 6.9
receptacles *(continued)*
  Series "A" and Series "B" receptacles, 6.5.3
  standards for, 6.7
  termination data, 6.5.2
  USB Icon, 6.5, 6.5.1
Recipient bits, 9.4.5
recovering from errors. *See* error detection and handling
recovery intervals for devices, 9.2.6.2
re-enumerating sub-trees, 10.5.4.6
reflected endpoint status, 10.5.2.2
registers in hub timing, 11.2.1
regulators in bus-powered hubs, 7.2.1.1
regulatory requirements for USB devices, 7.3.1
reliable delivery in isochronous transfers, 5.10
remote wakeup
  in configuration descriptors, 9.6.2
  Host Controller role, 10.2.7
  inrush current and, 7.2.3
  overview, 9.2.5.2
  resume signaling, 7.1.7.5, 9.1.1.6
  timing relationships, 11.9
  USB System role in, 10.5.4.6
Remote Wakeup field, 9.4.5
removable devices, 11.15.2.1
removing devices. *See* dynamic insertion and removal
RepeatingSE0 state, 11.6.2, 11.6.2.3
replacing configuration information, 10.5.4.1.3
reporting rates for feedback, 5.10.4.2
request codes, 9.4 *Table 9-4*, 11.16.2
Request Errors, 9.2.7
requests. *See also* PIDs; *names of specific requests*
  bRequest field, 9.3.2
  class-specific requests, 9.2.6.5, 10.5.2.8, 11.16:11.16.2.9
  completion times for hub requests, 11.16.1
  control transfers and, 5.5
  defined, 2.0 *glossary*
  in device class definitions, 9.7.3
  hub standard and class-specific requests, 11.16:11.16.2.9
  information requirements for, 10.3.4
  overview, 9.2.6
  port status reporting, 11.13.3
  request processing timing, 9.2.6.1
  reset/resume recovery time, 9.2.6.2
  set address processing, 9.2.6.3
  standard device requests, 9.2.6.4, 9.4:9.4.11
  standard feature selectors, 9.4 *Table 9-6*
  standard hub requests, 11.16:11.16.2.9
  standard request codes, 9.4 *Table 9-4*
  USBD command mechanisms, 10.5.2:10.5.2.12
  USB device requests, 9.3:9.3.5
  vendor-specific requests, 10.5.2.9
required data sequences for transfers, 5.4
reserved portions of frames, 5.5.4
Reset bus state
  downstream ports, 11.5, 11.5.1.5
  in power-on and connection events, 7.1.7.1
  reset signaling, 7.1.7.3
  signaling levels and, 7.1.7

- reset condition
  - in bus enumeration process, 9.1.2
  - C_PORT_RESET bit, 11.16.2.6.2.5
  - Default device state and, 9.1.1.3
  - device characteristics, 9.2.1
  - getting port status, 11.16.2.6.1
  - hub reset behavior, 11.10
  - PORT_RESET bit, 11.16.2.6.1.5
  - port status change bits, 11.16.2.6.2
  - remote wakeup and, 10.5.4.6
  - reset recovery time, 7.1.7.3, 9.2.6.2
  - resetting pipes, 10.5.2.2
  - SetPortFeature(PORT_RESET) request, 11.16.2.9
  - USB System and, 10.2.8.1
- reset signaling, 7.1.7.3
- resistance ratings, 6.6.3
- resistors
  - pull-up and pull-down resistors, 7.1.1.1, 7.1.5, 7.1.7.4, 7.2.3, 11.12.1
  - series damping resistors, 7.1.1.1
- resonators, data-rate tolerance and, 7.1.11
- resource management, USB System role in, 10.3.2
- Restart_S/Restart_E states, 11.5, 11.5.1.11
- Resume bus state
  - downstream ports, 11.5, 11.5.1.9
  - overview, 7.1.7.5
  - receivers, 11.6.1, 11.6.1.4
  - reset signaling and, 7.1.7.3
  - signaling levels and, 7.1.7
- Resume_Event signal/event, 11.4
- resume intervals for devices, 9.2.6.2
- resume signaling
  - hub support, 11.1.2.2, 11.9
  - power control during suspend/resume, 7.2.3
  - remote wakeup and, 10.5.4.6
  - resume conditions in Hub Controller, 11.4.4
  - single-ended transmissions, 11.6
- retire, defined, 2.0 *glossary*
- retiring IRPs. *See* aborting/retiring transfers
- RFI, USB grounding and, 6.8
- rise and fall times
  - data source jitter, 7.1.13.1
  - driver usage, 7.1.2.1
  - full-speed connections, 7.1.1.1
  - full-speed source electrical characteristics, 7.3.2 *Table 7-6*
  - low-speed source electrical characteristics, 7.3.2 *Table 7-7*
  - overview, 7.1.2
  - SE0 from low-speed devices, 7.1.14
- robustness of USB, 3.3, 4.5:4.5.2
- root hub
  - in bus topology, 5.2.3
  - defined, 2.0 *glossary*
  - HCDI presentation of, 10.4
  - Host Controller and, 10.2.8, 10.2.8.1
  - state handling, 10.2.1
- root port hub, defined, 7.2.1
- root ports, 2.0 *glossary*, 11.9
- Rptr_Enter_WFEOPFU signal/event, 11.5
- Rptr_Exit_WFEOPFU signal/event, 11.5
- Rptr_WFEOP signal/event, 11.6.2
- Rx_Bus_Reset signal/event, 11.6.2, 11.7
- Rx_Resume signal/event, 11.5, 11.7
- Rx_Suspend signal/event, 11.4, 11.5, 11.6.2, 11.7

S

- sample clock
  - buffering for rate matching, 5.10.8
  - defined, 5.10.2
  - synchronous endpoints, 5.10.4.1.2
- sampled analog devices, 5.10.4
- Sample Rate Conversion. *See* SRC
- samples
  - defined, 2.0 *glossary*
  - sample size in buffering calculations, 5.10.8
  - samples per frame in isochronous transfers, 5.10.4.2
- scheduling
  - access to USB interconnect, 4.1
  - transaction schedule in bus protocol overview, 4.4
- SE0sent signal/event, 11.6.2
- SE0 signal/event
  - in data signaling, 7.1.7.2
  - downstream port state machine, 11.5
  - Not Configured state, 11.5.1.1
  - propagation delays, 7.1.14
  - pull-down resistors and, 7.1.7.1
  - receiver state machine, 11.6.1
  - reset signaling, 7.1.7.3
  - SE0 interval of EOP, 7.3.2 *Table 7-6*, 7.3.2 *Table 7-7*
  - signaling levels and, 7.1.7
  - single-ended transmissions, 11.6
- SE0 width, 7.1.13.2, 7.1.14, 7.3.2 *Table 7-6*, 7.3.2 *Table 7-7*
- selective resume signaling, 11.9
- selective suspend signaling
  - defined, 9.1.1.6
  - hub support, 11.9
  - overview, 7.1.7.4.2
- self-powered devices and functions
  - configuration descriptors, 9.6.2
  - defined, 4.3.1, 7.2.1
  - device states, 9.1.1.2
  - overview, 7.2.1.5
- Self Powered field, 9.4.5
- self-powered hubs
  - configuration, 11.14
  - defined, 7.2.1
  - device states, 9.1.1.2
  - over-current protection, 7.2.1.2.1
  - overview, 7.2.1.2
  - power switching, 11.11
- self-recovery, USB robustness and, 4.5
- SendEOP state, 11.5, 11.5.1.10
- SendJ state, 11.6.2, 11.6.2.4
- Send Resume state (Sresume), 11.6.2, 11.6.2.6
- sequence of transactions in frames, 5.9.2
- Serial Interface Engine (SIE), 10.1.1, 10.2.2
- serializer/deserializer, 10.2.2
- serial numbers in device descriptors, 9.6.1

Series "A" and "B" connectors
  detachable cables and, 6.4.1
  keyed connector protocol, 6.2
  plugs
    injection molded thermoplastic insulator material, 6.5.4.1
    interface drawings, 6.5.4
    plug (male) contact materials, 6.5.4.3
    plug shell materials, 6.5.4.2
  receptacles
    injection molded thermoplastic insulator material, 6.5.3.1
    interface and mating drawings, 6.5.3
    PCB reference drawings, 6.9
    receptacle contact materials, 6.5.3.3
    receptacle shell materials, 6.5.3.2
  standards for, 6.7
  USB Icon, 6.5, 6.5.1
series damping resistors, 7.1.1.1
service, defined, 2.0 *glossary*
service clock, 5.10.2, 5.10.8
service intervals, 2.0 *glossary*, 10.3.3
service jitter, defined, 2.0 *glossary*
service periods of data, 5.10.1
service rates, defined, 2.0 *glossary*
SET_ADDRESS
  hub requests, 11.16.1
  overview, 9.4.6
  standard device request codes, 9.4
SetAddress() request
  overview, 9.4.6
  reset recovery time and, 7.1.7.3
  time limits for completing processing, 9.2.6.3
SetConfiguration() request, SET_CONFIGURATION
  hub requests, 11.16.1
  overview, 9.4.7
  Powered-off state and, 11.5.1.2
  setting configuration in descriptors, 9.6.2
  standard device request codes, 9.4
SetDescriptor() request, SET_DESCRIPTOR
  getting endpoint descriptors, 9.6.4
  hub class requests, 11.16.2
  hub requests, 11.16.1
  interface descriptors and, 9.6.3
  overview, 9.4.8
  SetHubDescriptor() request, 11.16.2.7
  standard device request codes, 9.4
SetDeviceFeature(DEVICE_REMOTE_WAKEUP) request, 10.5.4.6
SetFeature() request, SET_FEATURE, 9.4.5, 9.4.9
  hub class requests, 11.16.2
  hub requests, 11.16.1
  overview, 9.4.9
  SetHubFeature() request, 11.16.2.8
  SetPortFeature() request, 11.16.2.9
  standard device request codes, 9.4
SetHubDescriptor() request, 11.16.2, 11.16.2.7
SetHubFeature() request, 11.16.2, 11.16.2.5, 11.16.2.8
SetInterface() request, SET_INTERFACE, 9.2.3, 9.4, 9.4.10, 9.6.3, 11.16.1
SetPortFeature() request
  hub class requests, 11.16.2, 11.16.2.9
  PORT_CONNECTION, 11.16.2.6.1.1
  PORT_ENABLE, 11.16.2.6.1.2
  PORT_LOW_SPEED, 11.16.2.6.1.7
  PORT_OVER_CURRENT, 11.16.2.6.1.4
  PORT_POWER
    Disconnected state and, 11.5.1.3
    port power settings, 11.11
    port power states, 11.16.2.6.1.6
    requirements, 11.16.2.9
  PORT_RESET
    completion, 9.2.6
    C_PORT_ENABLE bit, 11.16.2.6.2.2
    evaluating device speed during, 11.8.2
    initiating port reset, 11.16.2.6.1.5
    in port enabling, 11.16.2.6.1.2
    requirements, 11.16.2.9
    Resetting state and, 11.5.1.5
  PORT_SUSPEND, 10.5.4.6
    requirements, 11.16.2.9
    selective suspend, 7.1.7.4.2
    suspending ports, 11.5.1.8, 11.16.2.6.1.3
  power-off conditions and, 11.14
SETUP PID, 8.3.1 *Table 8-1*
  ACK handshake and, 8.4.4
  ADDR field, 8.3.2.1
  in control transfers, 8.5.2
  in data toggle, 8.6.1
  ENDP field, 8.3.2.2
  function response to, 8.4.5.4
  token CRCs, 8.3.5.1
  in token packets, 8.4.1
Setup stage
  in control transfer data sequences, 5.5.5
  in control transfers, 5.5, 8.5.2
  data format for USB device requests, 9.3
shell
  conductors, 6.5.2
  plug shell materials, 6.5.4.2
  receptacle shell materials, 6.5.3.2
shielding
  grounding, 6.8
  low-speed and full-speed cables, 6.6
  outer and inner cable shielding, 6.6.1
  shielded cables illustrated, 6.4.1
  standardized contact terminating assignments, 6.5.2
short circuits, USB withstanding capabilities, 7.1.1
short packets
  defined, 9.4.3
  detecting, 10.2.6
  multiple data payloads and, 5.3.2
SIE (Serial Interface Engine), 10.1.1, 10.2.2
signal conductors in cables, 6.3
signal edges. *See* edges of signals
signaling
  bit stuffing, 7.1.9
  cable attenuation, 7.1.17
  connect and disconnect signaling, 7.1.7.1
  data encoding/decoding, 7.1.8

signaling *(continued)*
  data rate, 7.1.11
  data signaling, 7.1.7.2
  delay
    bus turn-around time and inter-packet delay, 7.1.18
    cable delay, 7.1.16
    cable skew delay, 7.1.3
    maximum end-to-end signal delay, 7.1.19
  frame interval and frame interval adjustment, 7.1.12
  hub signaling timings, 7.1.14
  in-band and out-of-band, 10.1.2
  input characteristics, 7.1.6
  jitter, 7.1.13.1, 7.1.15 (*See also* jitter)
  low-speed (1.5Mb/S) driver characteristics, 7.1.1.2
  overview, 7.1
  receiver characteristics, 7.1.4, 7.1.15
  reset signaling, 7.1.7.3
  resume signaling, 7.1.7.5
  rise and fall time, 7.1.2, 7.1.2.1
  signal attenuation, 7.1.17
  signal edges (*See* edges of signals)
  signaling levels, 7.1.7:7.1.7.5
  signal integrity, 4.5
  signal termination, 7.1.5
  source signaling, 7.1.13, 7.1.13.1, 7.1.13.2
  suspend signaling, 7.1.7.4
  sync pattern, 7.1.10
  USB driver characteristics, 7.1.1
signal matching, 7.1.2
signal pair attenuation, 6.4.1, 6.7 *Table 6-7*
signal pair construction, 6.6.2
signal pins, 7.2.4.1
signal swing, 7.1.2
signal termination, 7.1.5
Single-ended 0 bus state (SEO)
  in data signaling, 7.1.7.2
  pull-down resistors and, 7.1.7.1
  reset signaling, 7.1.7.3
  signaling levels and, 7.1.7
single-ended capacitance, 7.1.1.2
single-ended components in upstream ports, 11.6
single-ended receivers, 7.1.4, 7.1.6
single pin-type receptacles, 6.9
sink endpoints
  adaptive sink endpoints, 5.10.4.1.3
  audio connectivity, 5.10.4.3.1
  connectivity overview, 5.10.4.3
  feedback for isochronous transfers, 5.10.4.2
  synchronization types, 5.10.4.1
  synchronous data connectivity, 5.10.4.3.2
skew
  cable skew delay, 6.7 *Table 6-7*, 7.1.3, 7.3.2 *Table 7-9*
  differential-to-EOP transition skew, 7.3.3 *Figure 7-40*
  hub EOP delay and EOP skew, 7.3.3 *Figure 7-43*
  hub/repeater electrical characteristics, 7.3.2 *Table 7-8*
  hub switching skew, 7.1.9
  Idle-to-K state transition, 7.1.14
skew *(continued)*
  minimizing signal skew, 7.1.1
  timing skew accumulating, 11.2.2
slips in synchronous data, 5.10.4.3.2
small capacitors, 7.1.6
SOF PID, 8.3.1 *Table 8-1*
  frame number field, 8.3.3
  start-of-frame packets, 8.4.2
SOFs
  bus clock and, 5.10.2
  defined, 2.0 *glossary*
  in downstream port state machine, 11.5
  error recovery in isochronous transfers, 5.10.7
  frame intervals, 7.1.12
  frame timer synchronization, 11.2, 11.2.1
  Host Controller frame generation, 10.2.3
  loss of consecutive SOFs, 11.2.2
  mastership control and synchronization, 5.10.4.1.1, 5.10.4.1.2
  overview, 8.4.2
  tracking, 5.10.6, 5.10.7
  using as clocks, 5.10.5
soft-start circuits, 7.2.4.1
software interfaces. *See* client software; HCDI; host; USBDI; USB System software
solderability standards, 6.7 *Table 6-7*
solder tails, 6.5.3.3, 6.5.4.3
SOP bus state, 7.1.7, 7.1.7.2
SOP_FD signal/event
  generating, 11.4.4
  in Hub Repeater state machine, 11.7
SOP_FU signal/event, 11.7
SOPs, 8.3
  error detection through bus turn-around timing, 8.7.2
  frame timer synchronization and, 11.2.1
  Idle-to-K state transition, 7.1.14
  SOP distortion, 7.3.3 *Figure 7-42*
  timeout periods and, 7.1.19
source endpoints
  adaptive source endpoints, 5.10.4.1.3
  audio connectivity, 5.10.4.3.1
  connectivity overview, 5.10.4.3
  feedback for isochronous transfers, 5.10.4.2
  synchronization types, 5.10.4.1
  synchronous data connectivity, 5.10.4.3.2
source jitter, 7.3.2 *Table 7-6*, 7.3.2 *Table 7-7*
source/sink connectivity, 5.10.4.3
special PIDs (PRE PID), 8.3.1 *Table 8-1*
  inter-packet delays and, 7.1.18
  low-speed port behavior and, 11.8.4
  low-speed transactions, 8.6.5
  Transmit state and, 11.5.1.7
specific-sized data payloads, 5.3.2
speed detection
  attached devices, 11.8.2
  detecting low-speed functions and hubs, 11.16.2.6.1.7
  detecting speed of devices, 7.1.7.1
  termination and, 7.1.5
Speed_eval state, 11.8.2
SPI, defined, 2.0 *glossary*

splitting sample across packets, 5.10.8
SRC
  asynchronous SRC, 2.0 *glossary*
  audio connectivity and, 5.10.4.3.1
  defined, 2.0 *glossary*
  synchronous SRC, 2.0 *glossary*
Sresume state, 11.6.2, 11.6.2.6
staged power switching, 7.2.1.4
stages in control transfers, defined, 2.0 *glossary, 5.5*. *See also* Data stage; Setup stage; Status stage
STALLs, 8.3.1 *Table 8-1*
  in bulk transfers, 5.8.5, 8.5.1
  in control transfers, 8.5.2.1
  data corrupted or not accepted, 8.6.3
  functional and commanded stalls, 8.4.4
  function response to IN transactions, 8.4.5.1
  function response to OUT transactions, 8.4.5.3
  in interrupt transfers, 5.7.5, 8.5.3
  overview, 8.4.4
  protocol stalls, 8.4.4
  Request Error responses, 9.2.7
  responses to standard device requests, 9.4
  returned by control pipes, 8.5.2.4
standard device information, 4.8.1
standard device requests, 9.2.6.4, 9.4:9.4.11, 11.16.1
standards (applicable documents), 6.7.1
standard USB descriptor definitions, 9.6.1:9.6.5
Start-of-Frame. *See* SOFs
Start-of-Packet. *See* SOPs
Start-of-Packet bus state (SOP), 7.1.7, 7.1.7.2
start-of-packet delimiter. *See* SOPs
star topology, 5.2.3
state handling. *See* bus states; status
static output swing of USBD, 7.1.1
status. *See also* status change bits
  device states, 10.5.2.7, 11.13.2
  Host Controller role in, 4.9
  host's role in monitoring status and activity, 10.1.4
  hub and port status change bitmap, 11.13.4
  hub status, 11.16.2.5
  notification of completion status, 10.3.4
  port change information processing, 11.13.3
  port status, 11.16.2.6:11.16.2.6.2.5
  port status change bits, 11.16.2.6.2:11.16.2.6.2.5
  USBD event notifications, 10.5.4.4
  USBD status reporting and error recovery, 10.5.4.5
status change bits. *See also* Status Change endpoint
  detecting changes, 11.13.2
  device states, 11.13.2
  hub and port status change bitmap, 11.13.4
  hub status, 11.16.2.5
  over-current status change bits, 11.13.5
  port status change bits, 11.16.2.6.2:11.16.2.6.2.5
Status Change endpoint
  defined, 11.13.1
  device states and, 11.13.2
  hub and port status change bitmap, 11.13.4
  hub configuration and, 11.14
  hub descriptors, 11.15.1
Status stage
  in control transfers, 5.5, 5.5.5, 8.5.2
  reporting status results, 8.5.2.1
storage temperatures for cables, 6.6.4
stranded tinned conductors, 6.6.2
streaming real time transfers. *See* isochronous transfers
stream pipes
  bulk transfers and, 5.8.2
  in bus protocol overview, 4.4
  defined, 2.0 *glossary*, 5.3.2
  interrupt transfers and, 5.7.2
  isochronous transfers and, 5.6.2
  overview, 5.3.2.1
string descriptors
  GetDescriptor() request, 9.4.3
  as optional, 9.5
  overview, 9.6.5
stuffed bits. *See* bit stuffing
subclasses
  device subclass codes, 9.2.3, 9.6.1
  interface subclass codes, 9.2.3, 9.6.3
SubClass field, 9.2.3
substrate materials
  plug contact materials, 6.5.4.3
  plug shell materials, 6.5.4.2
  receptacle contact materials, 6.5.3.3
  receptacle shell materials, 6.5.3.2
successful transfers, 8.6.2, 10.3.4
supply current, 7.3.2 *Table 7-5*
supply voltage
  DC electrical characteristics, 7.3.2 *Table 7-5*
  oscillators, 7.1.11
surge limiting, 7.2.4.1
Suspend bus state
  global suspend, 7.1.7.4.1
  overview, 7.1.7.4
  power control during suspend/resume, 7.2.3
  reset signaling, 7.1.7.3
  resume signaling, 7.1.7.5
  selective suspend, 7.1.7.4.2
Suspend Delay state, 11.4, 11.4.2
suspended devices
  global suspend, 7.1.7.4.1
  hub support for suspend signaling, 11.9
  power control during suspend/resume, 7.2.3
  power-on and connection events, 7.1.7.1
  remote wakeup, 9.2.5.2, 10.2.7, 10.5.4.6
  resume signaling, 7.1.7.5
  selective suspend, 7.1.7.4.2
  single-ended transmissions, 11.6
  Suspend bus state, 7.1.7.4
  Suspended device state, 9.1.1.6
suspended hubs
  hub reset behavior, 11.10.1
  resume signaling and, 11.1.2.2
suspended ports
  C_PORT_SUSPEND, 11.16.2.6.2.3
  getting port status, 11.16.2.6.1
  port status change bits, 11.16.2.6.2
  PORT_SUSPEND, 11.16.2.6.1.3, 11.16.2.9

Suspended state, 9.1.1 *Table 9-1*, 9.1.1.6, 11.5, 11.5.1.8
Suspend state, 11.6.1, 11.6.1.2
switching thresholds for single-ended receivers, 7.1.4
SYNC field
  in data signaling, 7.1.7.2
  in electrical specifications overview, 4.2.1
  overview, 8.2
SynchFrame() request, SYNCH_FRAME, 9.4, 9.4.11, 11.16.1
synchronization. *See also* synchronization types
  clock synchronization, 5.10.3
  data-per-time synchronization, 5.10.7
  data toggle synchronization, 8.4.3, 8.6:8.6.5
  endpoint synchronization frame, 9.4.11
  frame timer synchronization, 11.2, 11.2.1
  jitter, 2.0 *glossary* (*See also* jitter)
  physical and virtual devices, 5.10.4.3
  SYNC field, 8.2
  sync pattern, 7.1.10
  transmitter and receiver synchronization in isochronous transfers, 5.10
synchronization types
  adaptive, 5.10.4.1.3
  asynchronous, 5.10.4.1.1
  defined, 2.0 *glossary*, 5.10.4
  overview, 5.10.4.1
  synchronous, 5.10.4.1.2
synchronous data connectivity, 5.10.4.3.2
synchronous data devices, 5.10.4
synchronous endpoints, 5.10.4.1.2, 5.10.4.3
synchronous RA, 2.0 *glossary*, 5.10.4.3
synchronous SRC, 2.0 *glossary*
sync pattern, 7.1.9, 7.1.10
system configuration. *See* configuration
System Programming Interface, defined, 2.0 *glossary*
system software. *See* USB System Software

T

taxonomy of application space, 3.2
TDM, defined, 2.0 *glossary*
telephone interconnects, 1.1
temperature
  data-rate inaccuracies and, 7.1.11
  ranges for cables, 6.6.4
termination
  blunt cut and prepared termination, 6.4.2, 6.4.3
  DC electrical characteristics, 7.3.2 *Table 7-5*
  defined, 2.0 *glossary*
  detachable cable assemblies, 6.4.1
  electrical specifications overview, 4.2.1
  full-speed captive cable assemblies, 6.4.2
  low-speed captive cable assemblies, 6.4.3
  signal termination, 7.1.5
  USB topology rules, 6.4.4
termination data, 6.5.2
test criteria for electrical, mechanical and environmental compliance, 6.7
Test for Flammability of Plastic Materials for Parts in Devices and Appliances, 6.7.1
thermal shock standards, 6.7 *Table 6-7*
Thevenin resistance, 7.1.5
tiered star topology, 5.2.3
Time Division Multiplexing (TDM), 2.0 *glossary*
timed states
  Disconnected state, 11.5.1.3
  Resuming state, 11.5.1.9
timeout
  bus transaction timeout, 5.10.7
  defined, 2.0 *glossary*
  detecting timeout conditions, 10.2.6
  timeout intervals in error detection, 8.7.3
timing. *See also* cable delay; propagation delay; skew; synchronization; timing waveforms
  bus timing/electrical characteristics, 7.3.2
  bus transaction time calculations, 5.9.3
  bus turn-around timing, 8.7.2
  clock model, 5.10.2
  clock synchronization, 5.10.3
  completion times for hub requests, 11.16.1
  current frame timer, 11.2.1
  data source signaling, 7.1.13
  device event timings, 7.3.2 *Table 7-11*
  frame interval adjustments, 7.1.12
  frame timer, 11.2.1
  hub event timings, 7.3.2 *Table 7-10*
  hub frame timer, 11.2:11.2.2
  hub signaling timings, 7.1.14
  isochronous transfer feedback, 5.10.4.2
  isochronous transfer importance, 5.10
  next frame timer, 11.2.1
  in non-USB isochronous application, 5.10.1
  port disconnect timer, 11.5.2
  power-on and connection events timing, 7.1.7.1
  remote wakeup timing relationships, 11.9
  request processing timing, 9.2.6.1
  Resetting state and Resuming state intervals, 11.5.1.9
  SE0 for EOP width timing, 7.1.13.2
  skew accumulating between host and hub, 11.2.2
  SOF PID timing information, 8.4.2
  SOF tokens as clocks, 5.10.5
  synchronization types, 5.10.4.1
timing waveforms, 7.3.3
  differential data jitter, 7.3.3 *Figure 7-39*
  differential-to-EOP transition skew and EOP width, 7.3.3 *Figure 7-40*
  hub differential delay, differential jitter, and SOP distortion, 7.3.3 *Figure 7-42*
  hub EOP delay and EOP skew, 7.3.3 *Figure 7-43*
  receiver jitter tolerance, 7.3.3 *Figure 7-41*
toggle mode. *See* data toggle
token packets
  in bulk transfers, 8.5.1
  bus protocol overview, 4.4
  CRCs, 8.3.5.1
  defined, 2.0 *glossary*
  in isochronous transfers, 8.5.4
  overview, 8.4.1
  packet field formats, 8.3:8.3.5.2
token PIDs, 8.3.1 *Table 8-1*. *See also* IN PID; OUT PID; SETUP PID; SOF PID

topology
  bus topology, 4.1, 4.1.1, 5.2:5.2.5
  hub tiers defined, 2.0 *glossary*
transaction completion prediction, 11.3.3
transaction list
  defined, 5.9.1.4
  HCD role in, 5.9.1.3
  Host Controller and, 5.9.1.5
transactions
  allocating bandwidth for, 5.9.1:5.9.1.5, 10.3.2
  buffer size calculations, 5.9.4
  bus protocol overview, 4.4
  defined, 2.0 *glossary*
  error detection and recovery, 8.7:8.7.4
  organization within IRPs, 5.9.2
  timeout, 5.10.7
  tracking transactions, 5.9.2
  transaction completion prediction, 11.3.3
  transaction formats
    bulk transfers, 5.8.4, 8.5.1
    control transfers, 5.5.4, 8.5.2:8.5.2.3
    interrupt transactions, 8.5.3
    isochronous transfers, 5.6.3, 5.10.6, 5.10.7, 8.5.4
    overview, 8.5
  transaction list, 5.9.1.3, 5.9.1.4, 5.9.1.5
  transaction schedule, 4.4
  transaction time calculations, 5.9.3
transfer management, 5.9.1:5.9.1.5
  allocating bandwidth, overview, 4.7.5, 5.9.1.1
  client software, 5.9.1.1
  HCD, 5.9.1.3
  Host Controller, 5.9.1.5
  illustrated, 5.9.1
  transaction list, 5.9.1.4
  USB driver, 5.9.1.2
  USB System, 10.3.2
transfers, 5.0. *See also* transactions; *names of specific transfer types (i.e., bulk transfers)*
  bulk transfers, 2.0 *glossary*, 4.7.2, 5.8:5.8.5, 8.5.1
  bus access for transfers, 5.9:5.9.5
    bus bandwidth reclamation, 5.9.5
    calculating buffer sizes in functions and software, 5.9.4
    calculating bus transaction times, 5.9.3
    transaction tracking, 5.9.2
    transfer management, 5.9.1:5.9.1.5
  bus protocol overview, 4.4
  bus topology, 5.2:5.2.5
  communication flow, 4.1, 5.3:5.3.2.2
  control transfers, 4.7.1, 5.5:5.5.5, 8.5.2:8.5.2.4
  data prebuffering, 5.10.5
  data signaling overview, 7.1.7.2
  defined, 2.0 *glossary*
  error detection and recovery, 8.7:8.7.4
  Host Controller responsibilities, 4.9, 10.1.3
  hub connectivity and, 11.1.2.1
  implementer viewpoints, 5.1
  interrupt transfers, 2.0 *glossary*, 4.7.3, 5.7:5.7.5, 8.5.3
  isochronous transfers, 2.0 *glossary*, 4.7.4, 5.6:5.6.5, 5.10:5.10.8, 8.5.4
transfers *(continued)*
  operations overview, 9.2.4
  organization of transactions within frames, 5.9.2
  overview, 5.0
  period transfers, 5.6.4
  power management, 9.2.5
  request processing, 9.2.6:9.2.6.5
  standard device requests, 9.4:9.4.11
  time limits for completing, 9.2.6.4, 9.2.6.5
  transaction formats, 8.5:8.5.4
  transfer types, 4.7:4.7.5, 5.4:5.8.5
  USB device requests, 9.3:9.3.5
  USBD role in, 10.1.1, 10.5.3:10.5.3.2.4
  USB System role in, 10.3.3
transfer types. *See also* transactions; transfers; *names of specific transfer types (i.e., bulk transfers)*
  allocating USB bandwidth, 4.7.5
  bulk transfers, 2.0 *glossary*, 4.7.2, 5.8
  in calculating transaction times, 5.9.3
  control transfers, 4.7.1, 5.5
  interrupt transfers, 2.0 *glossary*, 4.7.3, 5.7
  isochronous transfers, 2.0 *glossary*, 4.7.4, 5.6
  for message pipes, 5.3.2.2
  overview, 4.7:4.7.5, 5.4
  pipes and, 4.4
  for stream pipes, 5.3.2.1
  transfer types defined, 2.0 *glossary*
transmit phase of signaling, 7.1.1
Transmit state, 11.5, 11.5.1.7
transmitters
  Active state, 11.6.2.2
  Generate End of Packet Towards Upstream Port state (GEOPTU), 11.6.2.5
  Inactive state, 11.6.2.1
  RepeatingSE0 state, 11.6.2.3
  SendJ state, 11.6.2.4
  Send Resume state (Sresume), 11.6.2.6
  transmitter data jitter, 7.1.13.1
  transmitter sequence bits, 8.6, 8.6.2
  transmitter state descriptions, 11.6.2
  transmitter state machine, 11.6, 11.6.2
transmitter state machine, 11.6, 11.6.2
truncated packets, 11.3.2
turn-around times
  defined, 2.0 *glossary*
  error detection, 8.7.2
  overview, 7.1.18
turning power on for ports, 11.11
twisted data pair in cables, 6.6.1
Tx_active signal/event, 11.6.1
Tx_resume signal/event, 11.6.1

U

UEOP signal/event, 11.7
UL listing for cables, 6.6.5
UL STD-94, 6.7.1
UL Subject-444, 6.6.5, 6.7.1
unacceptable cables, 6.4.4

underplating
  plug contact materials, 6.5.4.3
  plug shell materials, 6.5.4.2
  receptacle contact materials, 6.5.3.3
  receptacle shell materials, 6.5.3.2
Underwriter's Laboratory, Inc., 6.6.5, 6.7.1
The Unicode Standard, Worldwide Character Encoding, 9.6.5
UNICODE string descriptors, 9.6.5
unique addresses
  assigning after dynamic insertion or removal, 4.6.3
  device initialization, 10.5.1.1
  operations overview, 9.2.2
  SetAddress() request, 9.4.6
  time limits for completing addressing, 9.2.6.3
Universal Serial Bus
  architectural extensions, 4.10
  bus protocol, 4.4
  clock model, 5.10, 5.10.2
  components, 5.1
  configuration, 4.6:4.6.3, 10.3.1
  data flow and transfers, 4.7:4.7.5, 5.1:5.10.8
  description, 4.1:4.1.1.2
  feature list, 3.3
  goals, 3.1
  host hardware and software, 4.9, 10.2:10.6
  hubs, 11.1:11.16
  mechanical and electrical specifications, 6.1:6.9, 7.1:7.1.19, 7.3:7.3.3
  motivation for development, 1.1
  physical interface, 4.2:4.2.2
  power distribution, 4.3:4.3.2, 7.2:7.2.4.2
  protocol layer, 8.1:8.7
  range of USB data traffic workloads, 3.2
  robustness and error detection/recovery, 4.5:4.5.2
  USB device framework, 9.1:9.7.3
  USB devices, 4.8:4.8.2.2
  USB schedule, 4.1
Universal Serial Bus Driver. *See* USBD (USB Driver)
Universal Serial Bus Resources, 2.0 *glossary*
up counters in hub timing, 11.2.1
upgrade paths, 3.3
upstream plugs, 6.2
upstream ports and hubs
  defined, 4.8.2.1
  full-speed drivers and, 7.1.2.1
  full-speed port transceiver, 7.1.7
  hub architecture, 11.1.1
  hub EOP delay and EOP skew, 7.3.3 *Figure 7-43*
  input capacitance, 7.1.6
  jitter, 7.3.2 *Table 7-7*
  low-speed source electrical characteristics, 7.3.2 *Table 7-7*
  pull-up and pull-down resistors, 11.12.1
  receivers, 11.6.1:11.6.1.6
  reset on upstream port, 11.10.1
  signaling delays, 7.1.14
  transmitters, 11.6.2:11.6.2.6
  upstream connectivity defined, 11.1.2.1
  upstream defined, 2.0 *glossary*
  upstream hub delay, 7.3.3 *Figure 7-42*
USB. *See* Universal Serial Bus
USB Bus Interface layer
  in bus topology, 5.2.2
  detailed communication flow illustrated, 5.3
  Host Controller implementation, 10.1.1
  illustrated, 5.1
  interlayer communications model, 10.1.1
USBD (USB Driver). *See also* USBDI (USB Driver Interface)
  in bus topology, 5.2.1
  command mechanisms, 10.5.1:10.5.2.12
  as component of USB System, 10.1.1
  configuration and, 10.3.1
  control mechanisms, 10.1.2
  data transfer mechanisms, 10.1.3
  defined, 2.0 *glossary*, 5.3, 10.5
  driver characteristics, 7.1.1
  full-and low-speed drivers, 7.1.1.1, 7.1.1.2, 7.1.2.1
  HCD interaction with, 10.4
  hub drivers, 10.3.1
  initialization, 10.5.1.1
  overview, 10.5.1
  passing preboot control to operating system, 10.5.5
  pipe mechanisms, 5.9.1.2, 10.5.1:10.5.3.2.4
  request data format mechanisms, 10.3.4
  service capabilities, 10.5.1.3
  software interface overview, 10.3
  in transfer management, 5.9.1, 5.9.1.2
  USB System and, 10.5.4:10.5.4.6
USB device framework, 9, 9.7.2
  descriptors, 9.5:9.7.3
  device class definitions, 9.7:9.7.3
  generic USB device operations, 9.2:9.2.7
    address assignment, 9.2.2
    configuration, 9.2.3
    data transfer, 9.2.4
    dynamic attachment and removal, 9.2.1
    power management, 9.2.5
    request error, 9.2.7
    request processing, 9.2.6:9.2.6.5
  standard descriptor definitions, 9.6:9.6.4
  standard device requests, 9.4:9.4.11
  USB device requests, 9.3:9.3.5
  USB device states, 9.1:9.1.2
USB Device layer
  detailed communication flow illustrated, 5.3
  illustrated, 5.1
  interlayer communications model, 10.1.1
USB devices. *See* devices
USBDI (USB Driver Interface)
  adding devices, 10.5.2.5
  alternate interface mechanisms, 10.5.2.10
  getting descriptors, 10.5.2.3
  granting Master Client status, 10.5.3.2.4, 10.5.4.2
  removing devices, 10.5.2.6
  role in request data format, 10.3.4
  sending class commands, 10.5.2.8
  sending vendor commands, 10.5.2.9
  setting descriptors, 10.5.2.12
  software interface overview, 10.3
USB host. *See* host
USB host controller. *See* Host Controller
USB Icon, 6.5, 6.5.1

USB interconnect model, 4.1, 5.10.4.3
USB Logical Devices. *See* logical devices
USB Physical Devices. *See* physical devices
USB schedule, 4.1
USB Specification Release Number, 9.6.1
USB System. *See also* HCD; host software; USBD
  allocating bandwidth, 10.3.2
  buffers and, 10.2.9
  data transfer role, 10.3.3
  HCD component, 10.1.1
  Host Controller interaction, 10.1.1
  host software component, 10.1.1
  power management, 10.5.4.3
  remote wakeup, 10.2.7, 10.5.4.6
  software interface overview, 10.3
  state handling, 10.2.1
  status and activity monitoring, 10.1.4
  USBD component, 10.1.1
USB System Software
  asynchronous data transfers, 4.9
  bus enumeration, 4.9
  in bus topology, 5.2.1
  in communication flow, 5.3
  detecting hub and port status changes, 11.13.2
  as implementation focus area, 5.1
  interrupt transfer support, 5.7.3
  isochronous transfer support, 4.9
  power management, 4.9
  role, 4.9

V

variable-length data stages, 8.5.2.2
variable-sized data payloads, 5.3.2
VBus leads
  bypassing, 7.2.4.1
  cable electrical characteristics, 7.3.2 *Table 7-9*
  detachable cables, 6.4.1
  in electrical specifications overview, 4.2.1
  full-speed captive cable assemblies, 6.4.2
  low-speed captive cable assemblies, 6.4.3
  standardized contact terminating assignments, 6.5.2
  upstream port power supply and, 7.2.1
Vendor IDs in device descriptors, 9.6.1
vendor information in device characteristics, 4.8.1
vendor-specific descriptors, 9.5
vendor-specific requests, 10.5.2.9
V/I characteristics of full-speed connections, 7.1.1.1
virtual devices, 2.0 *glossary*, 5.10.4.3
visible device states, 9.1.1
visual inspection standards, 6.7 *Table 6-7*
voltage
  average voltage on D+/D- lines, 7.1.2
  cross-over voltage in signaling, 7.1.2
  droop, 7.2.3, 7.2.4.1
  flyback voltage, 7.2.4.2
  full-speed connections, 7.1.1.1
  open-circuit voltage, 7.1.1
  ratings for cables, 6.6.3
  reduction due to cable resistive effects, 7.2.3
  supply voltage, 7.3.2 *Table 7-5*
  voltage drop budget, 7.2.2
voltage *(continued)*
  voltage drops, 7.2.1.1
  voltage drop topology, 7.2.2
  zero impedance voltage sources, 7.1.1

W

Wait for End of Packet from Upstream Port state (WFEOPFU), 11.7, 11.7.2
Wait for End of Packet (WFEOP) state, 11.7, 11.7.4
Wait for Start of Packet from Upstream Port state (WFSOPFU), 11.7, 11.7.1
Wait for Start of Packet (WFSOP) state, 11.7, 11.7.3
wander, defined, 11.2.2
waveforms
  differential data jitter, 7.3.3 *Figure 7-39*
  differential-to-EOP transition skew and EOP width, 7.3.3 *Figure 7-40*
  full-speed driver signal waveforms, 7.1.1.1
  hub differential delay, differential jitter, and SOP distortion, 7.3.3 *Figure 7-42*
  hub EOP delay and EOP skew, 7.3.3 *Figure 7-43*
  maximum input waveforms for signaling, 7.1.1
  receiver jitter tolerance, 7.3.3 *Figure 7-41*
WFEOPFU state, 11.5.1.6, 11.7, 11.7.2
WFEOP state, 11.7, 11.7.4
WFSOPFU state, 11.7, 11.7.1
WFSOP state, 11.7, 11.7.3
wHubChange field, 11.16.2.5
wHubCharacteristics field
  hub descriptor, 11.15.2.1
  multiple gangs and, 11.11.1
  over-current reporting, 11.13.5
  power switching settings, 11.11
wHubStatus field, 11.16.2.5
wIndex field
  hub class requests, 11.16.2
  overview, 9.3.4
  Setup data format, 9.3
  standard device requests, 9.4
wire gauge in cables, 6.6.2
wire insulation in cables, 6.6.2
wiring assignments for conductors, 6.5.2
wLANGID[] field (string descriptors), 9.6.5
wLength field
  hub class requests, 11.16.2
  overview, 9.3.5
  Setup data format, 9.3
  standard device requests, 9.4
wMaxPacketSize field
  bulk transfers and, 5.8.3
  control transfer packet size, 5.5.3
  endpoint descriptors, 9.6.4
  hub descriptors, 11.15.1
  interrupt transfer packet size, 5.7.3
  variable-length data stages, 8.5.2.2
words, defined, 2.0 *glossary*
working space, location and length of, 10.3.4
worst case bit stuffing, 5.9.3
wPortChange field, 11.16.2.6, 11.16.2.6.2
wPortStatus field, 11.16.2.6, 11.16.2.6.1
wTotalLength field (configuration descriptors), 9.6.2

wValue field
    hub class requests, 11.16.2
    overview, 9.3.3
    Setup data format, 9.3
    standard device requests, 9.4

Z

zero impedance voltage sources, 7.1.1