Universal Serial Bus Specification

Compaq

Hewlett-Packard

Intel

Lucent

Microsoft

NEC

Philips

Revision 2.0

April 27, 2000

Microchip Ex. 1018, p. 1
Microchip v. Aptiv
IPR2024-00494

Scope of this Revision

The 2.0 revision of the specification is intended for product design. Every attempt has been made to ensure a consistent and implementable specification. Implementations should ensure compliance with this revision.

Revision History

Revision	Issue Date	Comments
0.7	November 11, 1994	Supersedes 0.6e.
0.8	December 30, 1994	Revisions to Chapters 3-8, 10, and 11. Added appendixes.
0.9	April 13, 1995	Revisions to all the chapters.
0.99	August 25, 1995	Revisions to all the chapters.
1.0 FDR	November 13, 1995	Revisions to Chapters 1, 2, 5-11.
1.0	January 15, 1996	Edits to Chapters 5, 6, 7, 8, 9, 10, and 11 for consistency.
1.1	September 23, 1998	Updates to all chapters to fix problems identified.
2.0 (draft 0.79)	October 5, 1999	Revisions to chapters 5, 7, 8, 9, 11 to add high speed.
2.0 (draft 0.9)	December 21, 1999	Revisions to all chapters to add high speed.
2.0	April 27, 2000	Revisions for high-speed mode.

Universal Serial Bus Specification
Copyright © 2000, Compaq Computer Corporation,
Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc,
Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V.
All rights reserved.

INTELLECTUAL PROPERTY DISCLAIMER

THIS SPECIFICATION IS PROVIDED TO YOU "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE. THE AUTHORS OF THIS SPECIFICATION DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY PROPRIETARY RIGHTS, RELATING TO USE OR IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. THE PROVISION OF THIS SPECIFICATION TO YOU DOES NOT PROVIDE YOU WITH ANY LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS.

All product names are trademarks, registered trademarks, or servicemarks of their respective owners.

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

Microchip Ex. 1018, p. 2
Microchip v. Aptiv
IPR2024-00494

Acknowledgement of USB 2.0 Technical Contribution

The authors of this specification would like to recognize the following people who participated in the USB 2.0 Promoter Group technical working groups. We would also like to thank others in the USB 2.0 Promoter companies and throughout the industry who contributed to the development of this specification.

Hub Working Group

John Garney	Intel Corporation (Chair/Editor)
Ken Stufflebeam	Compaq Computer Corporation
David Wooten	Compaq Computer Corporation
Matt Nieberger	Hewlett-Packard Company
John Howard	Intel Corporation
Venkat Iyer	Intel Corporation
Steve McGowan	Intel Corporation
Geert Knapen	Royal Philips Electronics
Zong Liang Wu	Royal Philips Electronics
Jim Clee	Lucent Technologies Inc
Jim Guziak	Lucent Technologies Inc
Dave Thompson	Lucent Technologies Inc
John Fuller	Microsoft Corporation
Nathan Sherman	Microsoft Corporation
Mark Williams	Microsoft Corporation
Nobuo Furuya	NEC Corporation
Toshimi Sakurai	NEC Corporation
Moto Sato	NEC Corporation
Katsuya Suzuki	NEC Corporation

Electrical Working Group

Jon Lueker	Intel Corporation (Chair/Editor)
David Wooten	Compaq Computer Corporation
Matt Nieberger	Hewlett-Packard Company
Larry Taugher	Hewlett-Packard Company
Venkat Iyer	Intel Corporation
Steve McGowan	Intel Corporation
Mike Pennell	Intel Corporation
Todd West	Intel Corporation
Gerrit den Besten	Royal Philips Electronics
Marq Kole	Royal Philips Electronics
Zong Liang Wu	Royal Philips Electronics
Jim Clee	Lucent Technologies Inc
Jim Guziak	Lucent Technologies Inc
Par Parikh	Lucent Technologies Inc
Dave Thompson	Lucent Technologies Inc
Ed Giaimo	Microsoft Corporation
Mark Williams	Microsoft Corporation
Toshihiko Ohtani	NEC Corporation
Kugao Ouchi	NEC Corporation
Katsuya Suzuki	NEC Corporation
Toshio Tasaki	NEC Corporation

Microchip Ex. 1018, p. 3
Microchip v. Aptiv
IPR2024-00494

Microchip Ex. 1018, p. 4
Microchip v. Aptiv
IPR2024-00494

Contents

CHAPTER 1 INTRODUCTION

1.1 Motivation 1

1.2 Objective of the Specification 1

1.3 Scope of the Document 2

1.4 USB Product Compliance 2

1.5 Document Organization 2

CHAPTER 2 TERMS AND ABBREVIATIONS

CHAPTER 3 BACKGROUND

3.1 Goals for the Universal Serial Bus 11

3.2 Taxonomy of Application Space 12

3.3 Feature List 13

CHAPTER 4 ARCHITECTURAL OVERVIEW

4.1 USB System Description 15
4.1.1 Bus Topology 16

4.2 Physical Interface 17
4.2.1 Electrical 17
4.2.2 Mechanical 18

4.3 Power 18
4.3.1 Power Distribution 18
4.3.2 Power Management 18

4.4 Bus Protocol 18

4.5 Robustness 19
4.5.1 Error Detection 19
4.5.2 Error Handling 19

4.6 System Configuration 19
4.6.1 Attachment of USB Devices 20
4.6.2 Removal of USB Devices 20
4.6.3 Bus Enumeration 20

Microchip Ex. 1018, p. 5
Microchip v. Aptiv
IPR2024-00494

4.7 Data Flow Types 20
4.7.1 Control Transfers 21
4.7.2 Bulk Transfers 21
4.7.3 Interrupt Transfers 21
4.7.4 Isochronous Transfers 21
4.7.5 Allocating USB Bandwidth 21

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

4.9 USB Host: Hardware and Software 24

4.10 Architectural Extensions 24

CHAPTER 5 USB DATA FLOW MODEL

5.1 Implementer Viewpoints 25

5.2 Bus Topology 27
5.2.1 USB Host 27
5.2.2 USB Devices 28
5.2.3 Physical Bus Topology 29
5.2.4 Logical Bus Topology 30
5.2.5 Client Software-to-function Relationship 31

5.3 USB Communication Flow 31
5.3.1 Device Endpoints 33
5.3.2 Pipes 34
5.3.3 Frames and Microframes 36

5.4 Transfer Types 36
5.4.1 Table Calculation Examples 37

5.5 Control Transfers 38
5.5.1 Control Transfer Data Format 38
5.5.2 Control Transfer Direction 39
5.5.3 Control Transfer Packet Size Constraints 39
5.5.4 Control Transfer Bus Access Constraints 40
5.5.5 Control Transfer Data Sequences 43

5.6 Isochronous Transfers 44
5.6.1 Isochronous Transfer Data Format 44
5.6.2 Isochronous Transfer Direction 44
5.6.3 Isochronous Transfer Packet Size Constraints 44
5.6.4 Isochronous Transfer Bus Access Constraints 47
5.6.5 Isochronous Transfer Data Sequences 47

5.7 Interrupt Transfers 48
5.7.1 Interrupt Transfer Data Format 48
5.7.2 Interrupt Transfer Direction 48
5.7.3 Interrupt Transfer Packet Size Constraints 48
5.7.4 Interrupt Transfer Bus Access Constraints 49
5.7.5 Interrupt Transfer Data Sequences 52

Microchip Ex. 1018, p. 6
Microchip v. Aptiv
IPR2024-00494

5.8 Bulk Transfers 52
5.8.1 Bulk Transfer Data Format 52
5.8.2 Bulk Transfer Direction 52
5.8.3 Bulk Transfer Packet Size Constraints 53
5.8.4 Bulk Transfer Bus Access Constraints 53
5.8.5 Bulk Transfer Data Sequences 55

5.9 High-Speed, High Bandwidth Endpoints 56
5.9.1 High Bandwidth Interrupt Endpoints 56
5.9.2 High Bandwidth Isochronous Endpoints 57

5.10 Split Transactions 58

5.11 Bus Access for Transfers 58
5.11.1 Transfer Management 59
5.11.2 Transaction Tracking 61
5.11.3 Calculating Bus Transaction Times 63
5.11.4 Calculating Buffer Sizes in Functions and Software 65
5.11.5 Bus Bandwidth Reclamation 65

5.12 Special Considerations for Isochronous Transfers 65
5.12.1 Example Non-USB Isochronous Application 66
5.12.2 USB Clock Model 69
5.12.3 Clock Synchronization 71
5.12.4 Isochronous Devices 71
5.12.5 Data Prebuffering 80
5.12.6 SOF Tracking 81
5.12.7 Error Handling 81
5.12.8 Buffering for Rate Matching 82

CHAPTER 6 MECHANICAL

6.1 Architectural Overview 85

6.2 Keyed Connector Protocol 85

6.3 Cable 86

6.4 Cable Assembly 86
6.4.1 Standard Detachable Cable Assemblies 86
6.4.2 High-/full-speed Captive Cable Assemblies 88
6.4.3 Low-speed Captive Cable Assemblies 90
6.4.4 Prohibited Cable Assemblies 92

6.5 Connector Mechanical Configuration and Material Requirements 93
6.5.1 USB Icon Location 93
6.5.2 USB Connector Termination Data 94
6.5.3 Series “A” and Series “B” Receptacles 94
6.5.4 Series “A” and Series “B” Plugs 98

Microchip Ex. 1018, p. 7
Microchip v. Aptiv
IPR2024-00494

6.6 Cable Mechanical Configuration and Material Requirements 102
6.6.1 Description 102
6.6.2 Construction 103
6.6.3 Electrical Characteristics 105
6.1.4 Cable Environmental Characteristics 106
6.1.5 Listing 106

6.7 Electrical, Mechanical, and Environmental Compliance Standards 106
6.7.1 Applicable Documents 114

6.8 USB Grounding 114

6.9 PCB Reference Drawings 114

CHAPTER 7 ELECTRICAL

7.1 Signaling 119
7.1.1 USB Driver Characteristics 123
7.1.2 Data Signal Rise and Fall, Eye Patterns 129
7.1.3 Cable Skew 139
7.1.4 Receiver Characteristics 139
7.1.5 Device Speed Identification 141
7.1.6 Input Characteristics 142
7.1.7 Signaling Levels 144
7.1.8 Data Encoding/Decoding 157
7.1.9 Bit Stuffing 157
7.1.10 Sync Pattern 159
7.1.11 Data Signaling Rate 159
7.1.12 Frame Interval 159
7.1.13 Data Source Signaling 160
7.1.14 Hub Signaling Timings 162
7.1.15 Receiver Data Jitter 164
7.1.16 Cable Delay 165
7.1.17 Cable Attenuation 167
7.1.18 Bus Turn-around Time and Inter-packet Delay 168
7.1.19 Maximum End-to-end Signal Delay 168
7.1.20 Test Mode Support 169

7.2 Power Distribution 171
7.2.1 Classes of Devices 171
7.2.2 Voltage Drop Budget 175
7.2.3 Power Control During Suspend/Resume 176
7.2.4 Dynamic Attach and Detach 177

7.3 Physical Layer 178
7.3.1 Regulatory Requirements 178
7.3.2 Bus Timing/Electrical Characteristics 178
7.3.3 Timing Waveforms 191

Microchip Ex. 1018, p. 8
Microchip v. Aptiv
IPR2024-00494

CHAPTER 8 PROTOCOL LAYER

8.1 Byte/Bit Ordering 195

8.2 SYNC Field 195

8.3 Packet Field Formats 195
8.3.1 Packet Identifier Field 195
8.3.2 Address Fields 197
8.3.3 Frame Number Field 197
8.3.4 Data Field 197
8.3.5 Cyclic Redundancy Checks 198

8.4 Packet Formats 199
8.4.1 Token Packets 199
8.4.2 Split Transaction Special Token Packets 199
8.4.3 Start-of-Frame Packets 204
8.4.4 Data Packets 206
8.4.5 Handshake Packets 206
8.4.6 Handshake Responses 207

8.5 Transaction Packet Sequences 209
8.5.1 NAK Limiting via Ping Flow Control 217
8.5.2 Bulk Transactions 221
8.5.3 Control Transfers 225
8.5.4 Interrupt Transactions 228
8.5.5 Isochronous Transactions 229

8.6 Data Toggle Synchronization and Retry 232
8.6.1 Initialization via SETUP Token 233
8.6.2 Successful Data Transactions 233
8.6.3 Data Corrupted or Not Accepted 233
8.6.4 Corrupted ACK Handshake 234
8.6.5 Low-speed Transactions 235

8.7 Error Detection and Recovery 236
8.7.1 Packet Error Categories 236
8.7.2 Bus Turn-around Timing 237
8.7.3 False EOPs 237
8.7.4 Babble and Loss of Activity Recovery 238

Microchip Ex. 1018, p. 9
Microchip v. Aptiv
IPR2024-00494

CHAPTER 9 USB DEVICE FRAMEWORK

9.1 USB Device States....239
9.1.1 Visible Device States....239
9.1.2 Bus Enumeration....243

9.2 Generic USB Device Operations....244
9.2.1 Dynamic Attachment and Removal....244
9.2.2 Address Assignment....244
9.2.3 Configuration....244
9.2.4 Data Transfer....245
9.2.5 Power Management....245
9.2.6 Request Processing....245
9.2.7 Request Error....247

9.3 USB Device Requests....248
9.3.1 bmRequestType....248
9.3.2 bRequest....249
9.3.3 wValue....249
9.3.4 wIndex....249
9.3.5 wLength....249

9.4 Standard Device Requests....250
9.4.1 Clear Feature....252
9.4.2 Get Configuration....253
9.4.3 Get Descriptor....253
9.4.4 Get Interface....254
9.4.5 Get Status....254
9.4.6 Set Address....256
9.4.7 Set Configuration....257
9.4.8 Set Descriptor....257
9.4.9 Set Feature....258
9.4.10 Set Interface....259
9.4.11 Synch Frame....260

9.5 Descriptors....260

9.6 Standard USB Descriptor Definitions....261
9.6.1 Device....261
9.6.2 Device_Qualifier....264
9.6.3 Configuration....264
9.6.4 Other_Speed_Configuration....266
9.6.5 Interface....267
9.6.6 Endpoint....269
9.6.7 String....273

9.7 Device Class Definitions....274
9.7.1 Descriptors....274
9.7.2 Interface(s) and Endpoint Usage....274
9.7.3 Requests....274

Microchip Ex. 1018, p. 10
Microchip v. Aptiv
IPR2024-00494

CHAPTER 10 USB HOST: HARDWARE AND SOFTWARE

10.1 Overview of the USB Host 275
10.1.1 Overview 275
10.1.2 Control Mechanisms 278
10.1.3 Data Flow 278
10.1.4 Collecting Status and Activity Statistics 279
10.1.5 Electrical Interface Considerations 279

10.2 Host Controller Requirements 279
10.2.1 State Handling 280
10.2.2 Serializer/Deserializer 280
10.2.3 Frame and Microframe Generation 280
10.2.4 Data Processing 281
10.2.5 Protocol Engine 281
10.2.6 Transmission Error Handling 282
10.2.7 Remote Wakeup 282
10.2.8 Root Hub 282
10.2.9 Host System Interface 283

10.3 Overview of Software Mechanisms 283
10.3.1 Device Configuration 283
10.3.2 Resource Management 285
10.3.3 Data Transfers 286
10.3.4 Common Data Definitions 286

10.4 Host Controller Driver 287

10.5 Universal Serial Bus Driver 287
10.5.1 USBD Overview 288
10.5.2 USBD Command Mechanism Requirements 289
10.5.3 USBD Pipe Mechanisms 291
10.5.4 Managing the USB via the USBD Mechanisms 293
10.5.5 Passing USB Preboot Control to the Operating System 295

10.6 Operating System Environment Guides 296

CHAPTER 11 HUB SPECIFICATION

11.1 Overview 297
11.1.1 Hub Architecture 297
11.1.2 Hub Connectivity 298

11.2 Hub Frame/Microframe Timer 300
11.2.1 High-speed Microframe Timer Range 300
11.2.2 Full-speed Frame Timer Range 301
11.2.3 Frame/Microframe Timer Synchronization 301
11.2.4 Microframe Jitter Related to Frame Jitter 303
11.2.5 EOF1 and EOF2 Timing Points 303

Microchip Ex. 1018, p. 11
Microchip v. Aptiv
IPR2024-00494

11.3 Host Behavior at End-of-Frame 306
11.3.1 Full-/low-speed Latest Host Packet 306
11.3.2 Full-/low-speed Packet Nullification 306
11.3.3 Full-/low-speed Transaction Completion Prediction 306

11.4 Internal Port 307
11.4.1 Inactive 308
11.4.2 Suspend Delay 308
11.4.3 Full Suspend (Fsus) 308
11.4.4 Generate Resume (GResume) 308

11.5 Downstream Facing Ports 309
11.5.1 Downstream Facing Port State Descriptions 312
11.5.2 Disconnect Detect Timer 315
11.5.3 Port Indicator 316

11.6 Upstream Facing Port 318
11.6.1 Full-speed 318
11.6.2 High-speed 318
11.6.3 Receiver 318
11.6.4 Transmitter 322

11.7 Hub Repeater 324
11.7.1 High-speed Packet Connectivity 324
11.7.2 Hub Repeater State Machine 327
11.7.3 Wait for Start of Packet from Upstream Port (WFSOPFU) 329
11.7.4 Wait for End of Packet from Upstream Port (WFEOPFU) 330
11.7.5 Wait for Start of Packet (WFSOP) 330
11.7.6 Wait for End of Packet (WFEOP) 330

11.8 Bus State Evaluation 330
11.8.1 Port Error 330
11.8.2 Speed Detection 331
11.8.3 Collision 331
11.8.4 Low-speed Port Behavior 331

11.9 Suspend and Resume 332

11.10 Hub Reset Behavior 334

11.11 Hub Port Power Control 335
11.11.1 Multiple Gangs 335

11.12 Hub Controller 336
11.12.1 Endpoint Organization 336
11.12.2 Hub Information Architecture and Operation 337
11.12.3 Port Change Information Processing 337
11.12.4 Hub and Port Status Change Bitmap 338
11.12.5 Over-current Reporting and Recovery 339
11.12.6 Enumeration Handling 340

11.13 Hub Configuration 340

Microchip Ex. 1018, p. 12
Microchip v. Aptiv
IPR2024-00494

11.14 Transaction Translator 342
11.14.1 Overview 342
11.14.2 Transaction Translator Scheduling 344

11.15 Split Transaction Notation Information 346

11.16 Common Split Transaction State Machines 349
11.16.1 Host Controller State Machine 350
11.16.2 Transaction Translator State Machine 354

11.17 Bulk/Control Transaction Translation Overview 360
11.17.1 Bulk/Control Split Transaction Sequences 360
11.17.2 Bulk/Control Split Transaction State Machines 366
11.17.3 Bulk/Control Sequencing 371
11.17.4 Bulk/Control Buffering Requirements 372
11.17.5 Other Bulk/Control Details 372

11.18 Periodic Split Transaction Pipelining and Buffer Management 372
11.18.1 Best Case Full-Speed Budget 373
11.18.2 TT Microframe Pipeline 373
11.18.3 Generation of Full-speed Frames 374
11.18.4 Host Split Transaction Scheduling Requirements 374
11.18.5 TT Response Generation 378
11.18.6 TT Periodic Transaction Handling Requirements 379
11.18.7 TT Transaction Tracking 380
11.18.8 TT Complete-split Transaction State Searching 381

11.19 Approximate TT Buffer Space Required 382

11.20 Interrupt Transaction Translation Overview 382
11.20.1 Interrupt Split Transaction Sequences 383
11.20.2 Interrupt Split Transaction State Machines 386
11.20.3 Interrupt OUT Sequencing 392
11.20.4 Interrupt IN Sequencing 393

11.21 Isochronous Transaction Translation Overview 394
11.21.1 Isochronous Split Transaction Sequences 395
11.21.2 Isochronous Split Transaction State Machines 398
11.21.3 Isochronous OUT Sequencing 403
11.21.4 Isochronous IN Sequencing 404

11.22 TT Error Handling 404
11.22.1 Loss of TT Synchronization With HS SOFs 404
11.22.2 TT Frame and Microframe Timer Synchronization Requirements 405

11.23 Descriptors 407
11.23.1 Standard Descriptors for Hub Class 407
11.23.2 Class-specific Descriptors 417

11.24 Requests 419
11.24.1 Standard Requests 419
11.24.2 Class-specific Requests 420

Microchip Ex. 1018, p. 13
Microchip v. Aptiv
IPR2024-00494

APPENDIX A TRANSACTION EXAMPLES

A.1 Bulk/Control OUT and SETUP Transaction Examples....439

A.2 Bulk/Control IN Transaction Examples....464

A.3 Interrupt OUT Transaction Examples....489

A.4 Interrupt IN Transaction Examples....509

A.5 Isochronous OUT SpAppendix A Transaction Examples

APPENDIX B EXAMPLE DECLARATIONS FOR STATE MACHINES

B.1 Global Declarations....555

B.2 Host Controller Declarations....558

B.3 Transaction Translator Declarations....560

APPENDIX C RESET PROTOCOL STATE DIAGRAMS

C.1 Downstream Facing Port State Diagram....565

C.2 Upstream Facing Port State Diagram....567

C.2.1 Reset From Suspended State....567

C.2.2 Reset From Full-speed Non-suspended State....570

C.2.3 Reset From High-speed Non-suspended State....570

C.2.4 Reset Handshake....570

INDEX

Microchip Ex. 1018, p. 14
Microchip v. Aptiv
IPR2024-00494

Figures

Figure 3-1. Application Space Taxonomy 12
Figure 4-1. Bus Topology 16
Figure 4-2. USB Cable 17
Figure 4-3. A Typical Hub 23
Figure 4-4. Hubs in a Desktop Computer Environment 23
Figure 5-1. Simple USB Host/Device View 25
Figure 5-2. USB Implementation Areas 26
Figure 5-3. Host Composition 27
Figure 5-4. Physical Device Composition 28
Figure 5-5. USB Physical Bus Topology 29
Figure 5-6. Multiple Full-speed Buses in a High-speed System 30
Figure 5-7. USB Logical Bus Topology 30
Figure 5-8. Client Software-to-function Relationships 31
Figure 5-9. USB Host/Device Detailed View 32
Figure 5-10. USB Communication Flow 33
Figure 5-11. Data Phase PID Sequence for Isochronous IN High Bandwidth Endpoints 57
Figure 5-12. Data Phase PID Sequence for Isochronous OUT High Bandwidth Endpoints 58
Figure 5-13. USB Information Conversion From Client Software to Bus 59
Figure 5-14. Transfers for Communication Flows 62
Figure 5-15. Arrangement of IRPs to Transactions/(Micro)frames 63
Figure 5-16. Non-USB Isochronous Example 67
Figure 5-17. USB Full-speed Isochronous Application 70
Figure 5-18. Example Source/Sink Connectivity 77
Figure 5-19. Data Prebuffering 81
Figure 5-20. Packet and Buffer Size Formulas for Rate-matched Isochronous Transfers 83
Figure 6-1. Keyed Connector Protocol 85
Figure 6-2. USB Standard Detachable Cable Assembly 87
Figure 6-3. USB High-/full-speed Hardwired Cable Assembly 89
Figure 6-4. USB Low-speed Hardwired Cable Assembly 91
Figure 6-5. USB Icon 93
Figure 6-6. Typical USB Plug Orientation 93
Figure 6-7. USB Series "A" Receptacle Interface and Mating Drawing 95
Figure 6-8. USB Series "B" Receptacle Interface and Mating Drawing 96

Microchip Ex. 1018, p. 15
Microchip v. Aptiv
IPR2024-00494

Figure 6-9. USB Series "A" Plug Interface Drawing....99
Figure 6-10. USB Series “B” Plug Interface Drawing....100
Figure 6-11. Typical High-/full-speed Cable Construction....102
Figure 6-12. Single Pin-type Series "A" Receptacle....115
Figure 6-13. Dual Pin-type Series "A" Receptacle....116
Figure 6-14. Single Pin-type Series "B" Receptacle....117
Figure 7-1. Example High-speed Capable Transceiver Circuit....120
Figure 7-2. Maximum Input Waveforms for USB Signaling....124
Figure 7-3. Example Full-speed CMOS Driver Circuit (non High-speed capable)....125
Figure 7-4. Full-speed Buffer V/I Characteristics....126
Figure 7-5. Full-speed Buffer V/I Characteristics for High-speed Capable Transceiver....127
Figure 7-6. Full-speed Signal Waveforms....128
Figure 7-7. Low-speed Driver Signal Waveforms....128
Figure 7-8. Data Signal Rise and Fall Time....130
Figure 7-9. Full-speed Load....130
Figure 7-10. Low-speed Port Loads....131
Figure 7-11. Measurement Planes....131
Figure 7-12. Transmitter/Receiver Test Fixture....132
Figure 7-13. Template 1....133
Figure 7-14. Template 2....134
Figure 7-15. Template 3....135
Figure 7-16. Template 4....136
Figure 7-17. Template 5....137
Figure 7-18. Template 6....138
Figure 7-19. Differential Input Sensitivity Range for Low-/full-speed....140
Figure 7-20. Full-speed Device Cable and Resistor Connections....141
Figure 7-21. Low-speed Device Cable and Resistor Connections....141
Figure 7-22. Placement of Optional Edge Rate Control Capacitors for Low-/full-speed....143
Figure 7-23. Diagram for High-speed Loading Equivalent Circuit....143
Figure 7-24. Upstream Facing Full-speed Port Transceiver....146
Figure 7-25. Downstream Facing Low-/full-speed Port Transceiver....146
Figure 7-26. Low-/full-speed Disconnect Detection....149
Figure 7-27. Full-/high-speed Device Connect Detection....149
Figure 7-28. Low-speed Device Connect Detection....150
Figure 7-29. Power-on and Connection Events Timing....150
Figure 7-30. Low-/full-speed Packet Voltage Levels....152
Figure 7-31. NRZI Data Encoding....157

Microchip Ex. 1018, p. 16
Microchip v. Aptiv
IPR2024-00494

Figure 7-32. Bit Stuffing 157
Figure 7-33. Illustration of Extra Bit Preceding EOP (Full-/low-speed) 158
Figure 7-34. Flow Diagram for Bit Stuffing 158
Figure 7-35. Sync Pattern (Low-/full-speed) 159
Figure 7-36. Data Jitter Taxonomy 160
Figure 7-37. SE0 for EOP Width Timing 161
Figure 7-38. Hub Propagation Delay of Full-speed Differential Signals 162
Figure 7-39. Full-speed Cable Delay 166
Figure 7-40. Low-speed Cable Delay 166
Figure 7-41. Worst-case End-to-end Signal Delay Model for Low-/full-speed 169
Figure 7-42. Compound Bus-powered Hub 172
Figure 7-43. Compound Self-powered Hub 173
Figure 7-44. Low-power Bus-powered Function 174
Figure 7-45. High-power Bus-powered Function 174
Figure 7-46. Self-powered Function 175
Figure 7-47. Worst-case Voltage Drop Topology (Steady State) 175
Figure 7-48. Typical Suspend Current Averaging Profile 176
Figure 7-49. Differential Data Jitter for Low-/full-speed 191
Figure 7-50. Differential-to-EOP Transition Skew and EOP Width for Low-/full-speed 191
Figure 7-51. Receiver Jitter Tolerance for Low-/full-speed 191
Figure 7-52. Hub Differential Delay, Differential Jitter, and SOP Distortion for Low-/full-speed 192
Figure 7-53. Hub EOP Delay and EOP Skew for Low-/full-speed 193
Figure 8-1. PID Format 195
Figure 8-2. ADDR Field 197
Figure 8-3. Endpoint Field 197
Figure 8-4. Data Field Format 198
Figure 8-5. Token Format 199
Figure 8-6. Packets in a Start-split Transaction 200
Figure 8-7. Packets in a Complete-split Transaction 200
Figure 8-8. Relationship of Interrupt IN Transaction to High-speed Split Transaction 201
Figure 8-9. Relationship of Interrupt OUT Transaction to High-speed Split OUT Transaction 202
Figure 8-10. Start-split (SSPLIT) Token 202
Figure 8-11. Port Field 203
Figure 8-12. Complete-split (CSPLIT) Transaction Token 204
Figure 8-13. SOF Packet 204
Figure 8-14. Relationship between Frames and Microframes 205
Figure 8-15. Data Packet Format 206

Microchip Ex. 1018, p. 17
Microchip v. Aptiv
IPR2024-00494

Figure 8-16. Handshake Packet 206
Figure 8-17. Legend for State Machines 210
Figure 8-18. State Machine Context Overview 211
Figure 8-19. Host Controller Top Level Transaction State Machine Hierarchy Overview 211
Figure 8-20. Host Controller Non-split Transaction State Machine Hierarchy Overview 212
Figure 8-21. Device Transaction State Machine Hierarchy Overview 212
Figure 8-22. Device Top Level State Machine 213
Figure 8-23. Device_process_Trans State Machine 213
Figure 8-24. Dev_do_OUT State Machine 214
Figure 8-25. Dev_do_IN State Machine 215
Figure 8-26. HC_Do_nonsplit State Machine 216
Figure 8-27. Host High-speed Bulk OUT/Control Ping State Machine 218
Figure 8-28. Dev_HS_ping State Machine 219
Figure 8-29. Device High-speed Bulk OUT /Control State Machine 220
Figure 8-30. Bulk Transaction Format 221
Figure 8-31. Bulk/Control/Interrupt OUT Transaction Host State Machine 222
Figure 8-32. Bulk/Control/Interrupt OUT Transaction Device State Machine 223
Figure 8-33. Bulk/Control/Interrupt IN Transaction Host State Machine 224
Figure 8-34. Bulk/Control/Interrupt IN Transaction Device State Machine 225
Figure 8-35. Bulk Reads and Writes 225
Figure 8-36. Control SETUP Transaction 226
Figure 8-37. Control Read and Write Sequences 226
Figure 8-38. Interrupt Transaction Format 229
Figure 8-39. Isochronous Transaction Format 229
Figure 8-40. Isochronous OUT Transaction Host State Machine 230
Figure 8-41. Isochronous OUT Transaction Device State Machine 231
Figure 8-42. Isochronous IN Transaction Host State Machine 231
Figure 8-43. Isochronous IN Transaction Device State Machine 232
Figure 8-44. SETUP Initialization 233
Figure 8-45. Consecutive Transactions 233
Figure 8-46. NAKed Transaction with Retry 234
Figure 8-47. Corrupted ACK Handshake with Retry 234
Figure 8-48. Low-speed Transaction 235
Figure 8-49. Bus Turn-around Timer Usage 237
Figure 9-1. Device State Diagram 240
Figure 9-2. wIndex Format when Specifying an Endpoint 249
Figure 9-3. wIndex Format when Specifying an Interface 249

Microchip Ex. 1018, p. 18
Microchip v. Aptiv
IPR2024-00494

Figure 9-4. Information Returned by a GetStatus() Request to a Device 255
Figure 9-5. Information Returned by a GetStatus() Request to an Interface....255
Figure 9-6. Information Returned by a GetStatus() Request to an Endpoint 256
Figure 9-7. Example of Feedback Endpoint Numbers....272
Figure 9-8. Example of Feedback Endpoint Relationships....272
Figure 10-1. Interlayer Communications Model....275
Figure 10-2. Host Communications....276
Figure 10-3. Frame and Microframe Creation 281
Figure 10-4. Configuration Interactions....284
Figure 10-5. Universal Serial Bus Driver Structure....288
Figure 11-1. Hub Architecture....298
Figure 11-2. Hub Signaling Connectivity 299
Figure 11-3. Resume Connectivity 299
Figure 11-4. Example High-speed EOF Offsets Due to Propagation Delay Without EOF Advancement 302
Figure 11-5. Example High-speed EOF Offsets Due to Propagation Delay With EOF Advancement....302
Figure 11-6. High-speed EOF2 Timing Point....304
Figure 11-7. High-speed EOF1 Timing Point....304
Figure 11-8. Full-speed EOF Timing Points....304
Figure 11-9. Internal Port State Machine....308
Figure 11-10. Downstream Facing Hub Port State Machine 310
Figure 11-11. Port Indicator State Diagram....317
Figure 11-12. Upstream Facing Port Receiver State Machine....319
Figure 11-13. Upstream Facing Port Transmitter State Machine 322
Figure 11-14. Example Hub Repeater Organization....324
Figure 11-15. High-speed Port Selector State Machine....326
Figure 11-16. Hub Repeater State Machine....328
Figure 11-17. Example Remote-wakeup Resume Signaling With Full-/low-speed Device 333
Figure 11-18. Example Remote-wakeup Resume Signaling With High-speed Device 334
Figure 11-19. Example Hub Controller Organization....336
Figure 11-20. Relationship of Status, Status Change, and Control Information to Device States 337
Figure 11-21. Port Status Handling Method 338
Figure 11-22. Hub and Port Status Change Bitmap....339
Figure 11-23. Example Hub and Port Change Bit Sampling 339
Figure 11-24. Transaction Translator Overview....342
Figure 11-25. Periodic and Non-periodic Buffer Sections of TT....343
Figure 11-26. TT Microframe Pipeline for Periodic Split Transactions 344
Figure 11-27. TT Nonperiodic Buffering....345

Microchip Ex. 1018, p. 19
Microchip v. Aptiv
IPR2024-00494

Figure 11-28. Example Full-/low-speed Handler Scheduling for Start-splits 346
Figure 11-29. Flow Sequence Legend 346
Figure 11-30. Legend for State Machines 347
Figure 11-31. State Machine Context Overview 348
Figure 11-32. Host Controller Split Transaction State Machine Hierarchy Overview 349
Figure 11-33. Transaction Translator State Machine Hierarchy Overview 350
Figure 11-34. Host Controller 350
Figure 11-35. HC_Process_Command 351
Figure 11-36. HC_Do_Start 352
Figure 11-37. HC_Do_Complete 353
Figure 11-38. Transaction Translator 354
Figure 11-39. TT_Process_Packet 355
Figure 11-40. TT_Do_Start 356
Figure 11-41. TT_Do_Complete 357
Figure 11-42. TT_BulkSS 357
Figure 11-43. TT_BulkCS 358
Figure 11-44. TT_IntSS 358
Figure 11-45. TT_IntCS 359
Figure 11-46. TT_IsochSS 359
Figure 11-47. Sample Algorithm for Compare_buffs 361
Figure 11-48. Bulk/Control OUT Start-split Transaction Sequence 362
Figure 11-49. Bulk/Control OUT Complete-split Transaction Sequence 363
Figure 11-50. Bulk/Control IN Start-split Transaction Sequence 364
Figure 11-51. Bulk/Control IN Complete-split Transaction Sequence 365
Figure 11-52. Bulk/Control OUT Start-split Transaction Host State Machine 366
Figure 11-53. Bulk/Control OUT Complete-split Transaction Host State Machine 367
Figure 11-54. Bulk/Control OUT Start-split Transaction TT State Machine 368
Figure 11-55. Bulk/Control OUT Complete-split Transaction TT State Machine 368
Figure 11-56. Bulk/Control IN Start-split Transaction Host State Machine 369
Figure 11-57. Bulk/Control IN Complete-split Transaction Host State Machine 370
Figure 11-58. Bulk/Control IN Start-split Transaction TT State Machine 371
Figure 11-59. Bulk/Control IN Complete-split Transaction TT State Machine 371
Figure 11-60. Best Case Budgeted Full-speed Wire Time With No Bit Stuffing 373
Figure 11-61. Scheduling of TT Microframe Pipeline 374
Figure 11-62. Isochronous OUT Example That Avoids a Start-split-end With Zero Data 375
Figure 11-63. End of Frame TT Pipeline Scheduling Example 376
Figure 11-64. Isochronous IN Complete-split Schedule Example at $L=Y_6$ 377

Microchip Ex. 1018, p. 20
Microchip v. Aptiv
IPR2024-00494

Figure 11-65. Isochronous IN Complete-split Schedule Example at L=Y_7 377
Figure 11-66. Microframe Pipeline 380
Figure 11-67. Advance_Pipeline Pseudocode 381
Figure 11-68. Interrupt OUT Start-split Transaction Sequence 383
Figure 11-69. Interrupt OUT Complete-split Transaction Sequence 384
Figure 11-70. Interrupt IN Start-split Transaction Sequence 385
Figure 11-71. Interrupt IN Complete-split Transaction Sequence 385
Figure 11-72. Interrupt OUT Start-split Transaction Host State Machine 386
Figure 11-73. Interrupt OUT Complete-split Transaction Host State Machine 387
Figure 11-74. Interrupt OUT Start-split Transaction TT State Machine 388
Figure 11-75. Interrupt OUT Complete-split Transaction TT State Machine 389
Figure 11-76. Interrupt IN Start-split Transaction Host State Machine 389
Figure 11-77. Interrupt IN Complete-split Transaction Host State Machine 390
Figure 11-78. HC_Data_or_Error State Machine 391
Figure 11-79. Interrupt IN Start-split Transaction TT State Machine 391
Figure 11-80. Interrupt IN Complete-split Transaction TT State Machine 392
Figure 11-81. Example of CRC16 Handling for Interrupt OUT 393
Figure 11-82. Example of CRC16 Handling for Interrupt IN 394
Figure 11-83. Isochronous OUT Start-split Transaction Sequence 395
Figure 11-84. Isochronous IN Start-split Transaction Sequence 396
Figure 11-85. Isochronous IN Complete-split Transaction Sequence 397
Figure 11-86. Isochronous OUT Start-split Transaction Host State Machine 398
Figure 11-87. Isochronous OUT Start-split Transaction TT State Machine 399
Figure 11-88. Isochronous IN Start-split Transaction Host State Machine 400
Figure 11-89. Isochronous IN Complete-split Transaction Host State Machine 401
Figure 11-90. Isochronous IN Start-split Transaction TT State Machine 402
Figure 11-91. Isochronous IN Complete-split Transaction TT State Machine 402
Figure 11-92. Example of CRC16 Isochronous OUT Data Packet Handling 403
Figure 11-93. Example of CRC16 Isochronous IN Data Packet Handling 404
Figure 11-94. Example Frame/Microframe Synchronization Events 406
Figure A-1. Normal No Smash 441
Figure A-2. Normal HS DATA0/1 Smash 442
Figure A-3. Normal HS DATA0/1 3 Strikes Smash 443
Figure A-4. Normal HS ACK(S) Smash(case 1) 444
Figure A-5. Normal HS ACK(S) Smash(case 2) 445
Figure A-6. Normal HS ACK(S) 3 Strikes Smash 446
Figure A-7. Normal HS CSPLIT Smash 447

Microchip Ex. 1018, p. 21
Microchip v. Aptiv
IPR2024-00494

Figure A-8. Normal HS CSPLIT 3 Strikes Smash....448
Figure A-9. Normal HS ACK(C) Smash....449
Figure A-10. Normal S ACK(C) 3 Strikes Smash....450
Figure A-11. Normal FS/LS DATA0/1 Smash....451
Figure A-12. Normal FS/LS DATA0/1 3 Strikes Smash....452
Figure A-13. Normal FS/LS ACK Smash....453
Figure A-14. Normal FS/LS ACK 3 Strikes Smash....454
Figure A-15. No buffer Available No Smash (HS NAK(S))....455
Figure A-16. No Buffer Available HS NAK(S) Smash....456
Figure A-17. No Buffer Available HS NAK(S) 3 Strikes Smash....457
Figure A-18. CS Earlier No Smash (HS NYET)....458
Figure A-19. CS Earlier HS NYET Smash(case 1)....459
Figure A-20. CS Earlier HS NYET Smash(case 2)....460
Figure A-21. CS Earlier HS NYET 3 Strikes Smash....461
Figure A-22. Device Busy No Smash(FS/LS NAK)....462
Figure A-23. Device Stall No Smash(FS/LS STALL)....463
Figure A-24. Normal No Smash....466
Figure A-25. Normal HS SSPLIT Smash....467
Figure A-26. Normal SSPLIT 3 Strikes Smash....468
Figure A-27. Normal HS ACK(S) Smash(case 1)....469
Figure A-28. Normal HS ACK(S) Smash(case 2)....470
Figure A-29. Normal HS ACK(S) 3 Strikes Smash....471
Figure A-30. Normal HS CSPLIT Smash....472
Figure A-31. Normal HS CSPLIT 3 Strikes Smash....473
Figure A-32. Normal HS DATA0/1 Smash....474
Figure A-33. Normal HS DATA0/1 3 Strikes Smash....475
Figure A-34. Normal FS/LS IN Smash....476
Figure A-35. Normal FS/LS IN 3 Strikes Smash....477
Figure A-36. Normal FS/LS DATA0/1 Smash....478
Figure A-37. Normal FS/LS DATA0/1 3 Strikes Smash....479
Figure A-38. Normal FS/LS ACK Smash....480
Figure A-39. No Buffer Available No Smash(HS NAK(S))....481
Figure A-40. No Buffer Available HS NAK(S) Smash....482
Figure A-41. No Buffer Available HS NAK(S) 3 Strikes Smash....483
Figure A-42. CS Earlier No Smash(HS NYET)....484
Figure A-43. CS Earlier HS NYET Smash(case 1)....485
Figure A-44. CS Earlier HS NYET Smash(case 2)....486

Microchip Ex. 1018, p. 22
Microchip v. Aptiv
IPR2024-00494

Figure A-45. Device Busy No Smash(FS/LS NAK)....487
Figure A-46. Device Stall No Smash(FS/LS STALL)....488
Figure A-47. Normal No Smash(FS/LS Handshake Packet is Done by M+1)....492
Figure A-48. Normal HS DATA0/1 Smash....493
Figure A-49. Normal HS CSPLIT Smash....494
Figure A-50. Normal HS CSPLIT 3 Strikes Smash....495
Figure A-51. Normal HS ACK(C) Smash....496
Figure A-52. Normal HS ACK(C) 3 Strikes Smash....497
Figure A-53. Normal FS/LS DATA0/1 Smash....498
Figure A-54. Normal FS/LS ACK Smash....499
Figure A-55. Searching No Smash....500
Figure A-56. CS Earlier No Smash(HS NYET and FS/LS Handshake Packet is Done by M+2)....501
Figure A-57. CS Earlier No Smash(HS NYET and FS/LS Handshake Packet is Done by M+3)....502
Figure A-58. CS Earlier HS NYET Smash....503
Figure A-59. CS Earlier HS NYET 3 Strikes Smash....504
Figure A-60. Abort and Free Abort(FS/LS Transaction is Continued at End of M+3)....505
Figure A-61. Abort and Free Free(FS/LS Transaction is not Started at End of M+3)....506
Figure A-62. Device Busy No Smash(FS/LS NAK)....507
Figure A-63. Device Stall No Smash(FS/LS STALL)....508
Figure A-64. Normal No Smash(FS/LS Data Packet is on M+1)....512
Figure A-65. Normal HS SSPLIT Smash....513
Figure A-66. Normal HS CSPLIT Smash....514
Figure A-67. Normal HS CSPLIT 3 Strikes Smash....515
Figure A-68. Normal HS DATA0/1 Smash....516
Figure A-69. Normal HS DATA0/1 3 Strikes Smash....517
Figure A-70. Normal FS/LS IN Smash....518
Figure A-71. Normal FS/LS DATA0/1 Smash....519
Figure A-72. Normal FS/LS ACK Smash....520
Figure A-73. Searching No Smash....521
Figure A-74. CS Earlier No Smash(HS MDATA and FS/LS Data Packet is on M+1 and M+2)....522
Figure A-75. CS Earlier No Smash(HS NYET and FS/LS Data Packet is on M+2)....523
Figure A-76. CS Earlier No Smash(HS NYET and MDATA and FS/LS Data Packet is on M+2 and M+3)....524
Figure A-77. CS Earlier No Smash(HS NYET and FS/LS Data Packet is on M+3)....525
Figure A-78. CS Earlier HS NYET Smash....526
Figure A-79. CS Earlier HS NYET 3 Strikes Smash....527
Figure A-80. Abort and Free Abort(HS NYET and FS/LS Transaction is Continued at End of M+3)....528
Figure A-81. Abort and Free Free(HS NYET and FS/LS Transaction is not Started at End of M+3)....529

Microchip Ex. 1018, p. 23
Microchip v. Aptiv
IPR2024-00494

Figure A-82. Device Busy No Smash(FS/LS NAK) 530
Figure A-83. Device Stall No Smash(FS/LS STALL) 531
Figure C-1. Downstream Facing Port Reset Protocol State Diagram 566
Figure C-2. Upstream Facing Port Reset Detection State Diagram 568
Figure C-3. Upstream Facing Port Reset Handshake State Diagram 569

Microchip Ex. 1018, p. 24
Microchip v. Aptiv
IPR2024-00494

Tables

Table 5-1. Low-speed Control Transfer Limits 41
Table 5-2. Full-speed Control Transfer Limits 42
Table 5-3. High-speed Control Transfer Limits 43
Table 5-4. Full-speed Isochronous Transaction Limits 45
Table 5-5. High-speed Isochronous Transaction Limits 46
Table 5-6. Low-speed Interrupt Transaction Limits 49
Table 5-7. Full-speed Interrupt Transaction Limits 50
Table 5-8. High-speed Interrupt Transaction Limits 51
Table 5-9. Full-speed Bulk Transaction Limits 54
Table 5-10. High-speed Bulk Transaction Limits 55
Table 5-11. *wMaxPacketSize* Field of Endpoint Descriptor 56
Table 5-12. Synchronization Characteristics 72
Table 5-13. Connection Requirements 79
Table 6-1. USB Connector Termination Assignment 94
Table 6-2. Power Pair 103
Table 6-3. Signal Pair 104
Table 6-4. Drain Wire Signal Pair 104
Table 6-5. Nominal Cable Diameter 105
Table 6-6. Conductor Resistance 105
Table 6-7. USB Electrical, Mechanical, and Environmental Compliance Standards 106
Table 7-1. Description of Functional Elements in the Example Shown in Figure 7-1 122
Table 7-2. Low-/full-speed Signaling Levels 145
Table 7-3. High-speed Signaling Levels 147
Table 7-4. Full-speed Jitter Budget 164
Table 7-5. Low-speed Jitter Budget 165
Table 7-6. Maximum Allowable Cable Loss 167
Table 7-7. DC Electrical Characteristics 178
Table 7-8. High-speed Source Electrical Characteristics 180
Table 7-9. Full-speed Source Electrical Characteristics 181
Table 7-10. Low-speed Source Electrical Characteristics 182
Table 7-11. Hub/Repeater Electrical Characteristics 183
Table 7-12. Cable Characteristics (Note 14) 185
Table 7-13. Hub Event Timings 186
Table 7-14. Device Event Timings 188

Microchip Ex. 1018, p. 25
Microchip v. Aptiv
IPR2024-00494

Table 8-1. PID Types....196
Table 8-2. Isochronous OUT Payload Continuation Encoding....203
Table 8-3. Endpoint Type Values in Split Special Token....204
Table 8-4. Function Responses to IN Transactions....208
Table 8-5. Host Responses to IN Transactions....208
Table 8-6. Function Responses to OUT Transactions in Order of Precedence....209
Table 8-7. Status Stage Responses....227
Table 8-8. Packet Error Types....236
Table 9-1. Visible Device States....241
Table 9-2. Format of Setup Data....248
Table 9-3. Standard Device Requests....250
Table 9-4. Standard Request Codes....251
Table 9-5. Descriptor Types....251
Table 9-6. Standard Feature Selectors....252
Table 9-7. Test Mode Selectors....259
Table 9-8. Standard Device Descriptor....262
Table 9-9. Device_Qualifier Descriptor....264
Table 9-10. Standard Configuration Descriptor....265
Table 9-11. Other_Speed_Configuration Descriptor....267
Table 9-12. Standard Interface Descriptor....268
Table 9-13. Standard Endpoint Descriptor....269
Table 9-14. Allowed wMaxPacketSize Values for Different Numbers of Transactions per Microframe....273
Table 9-15. String Descriptor Zero, Specifying Languages Supported by the Device....273
Table 9-16. UNICODE String Descriptor....274
Table 11-1. High-speed Microframe Timer Range Contributions....300
Table 11-2. Full-speed Frame Timer Range Contributions....301
Table 11-3. Hub and Host EOF1/EOF2 Timing Points....303
Table 11-4. Internal Port Signal/Event Definitions....308
Table 11-5. Downstream Facing Port Signal/Event Definitions....311
Table 11-6. Automatic Port State to Port Indicator Color Mapping....316
Table 11-7. Port Indicator Color Definitions....317
Table 11-8. Upstream Facing Port Receiver Signal/Event Definitions....320
Table 11-9. Upstream Facing Port Transmit Signal/Event Definitions....323
Table 11-10. High-speed Port Selector Signal/Event Definitions....326
Table 11-11. Hub Repeater Signal/Event Definitions....329
Table 11-12. Hub Power Operating Mode Summary....341
Table 11-13. Hub Descriptor....417

Microchip Ex. 1018, p. 26
Microchip v. Aptiv
IPR2024-00494

Table 11-14. Hub Responses to Standard Device Requests 419
Table 11-15. Hub Class Requests 420
Table 11-16. Hub Class Request Codes 421
Table 11-17. Hub Class Feature Selectors 421
Table 11-18. wValue Field for Clear_TT_Buffer 424
Table 11-19. Hub Status Field, *wHubStatus* 425
Table 11-20. Hub Change Field, *wHubChange* 426
Table 11-21. Port Status Field, *wPortStatus* 427
Table 11-22. Port Change Field, *wPortChange* 431
Table 11-23. Format of Returned TT State 432
Table 11-24. Test Mode Selector Codes 436
Table 11-25. Port Indicator Selector Codes 437

Microchip Ex. 1018, p. 27
Microchip v. Aptiv
IPR2024-00494

Microchip Ex. 1018, p. 28
Microchip v. Aptiv
IPR2024-00494

Chapter 1
Introduction

1.1 Motivation

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

- **Connection of the PC to the telephone**
  It is well understood that the merge of computing and communication will be the basis for the next generation of productivity applications. The movement of machine-oriented and human-oriented data types from one location or environment to another depends on ubiquitous and cheap connectivity. Unfortunately, the computing and communication industries have evolved independently. The USB provides a ubiquitous link that can be used across a wide range of PC-to-telephone interconnects.

- **Ease-of-use**
  The lack of flexibility in reconfiguring the PC has been acknowledged as the Achilles' heel to its further deployment. The combination of user-friendly graphical interfaces and the hardware and software mechanisms associated with new-generation bus architectures have made computers less confrontational and easier to reconfigure. However, from the end user's point of view, the PC's I/O interfaces, such as serial/parallel ports, keyboard/mouse/joystick interfaces, etc., do not have the attributes of plug-and-play.

- **Port expansion**
  The addition of external peripherals continues to be constrained by port availability. The lack of a bi-directional, low-cost, low-to-mid speed peripheral bus has held back the creative proliferation of peripherals such as telephone/fax/modem adapters, answering machines, scanners, PDA's, keyboards, mice, etc. Existing interconnects are optimized for one or two point products. As each new function or capability is added to the PC, a new interface has been defined to address this need.

The more recent motivation for USB 2.0 stems from the fact that PCs have increasingly higher performance and are capable of processing vast amounts of data. At the same time, PC peripherals have added more performance and functionality. User applications such as digital imaging demand a high performance connection between the PC and these increasingly sophisticated peripherals. USB 2.0 addresses this need by adding a third transfer rate of 480 Mb/s to the 12 Mb/s and 1.5 Mb/s originally defined for USB. USB 2.0 is a natural evolution of USB, delivering the desired bandwidth increase while preserving the original motivations for USB and maintaining full compatibility with existing peripherals.

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

1.2 Objective of the Specification

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

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

Microchip Ex. 1018, p. 29
Microchip v. Aptiv
IPR2024-00494

1.3 Scope of the Document

The specification is primarily targeted to peripheral developers and system OEMs, but provides valuable information for platform operating system/ BIOS/ device driver, adapter IHVs/ISVs, and platform/adapter controller vendors. This specification can be used for developing new products and associated software.

1.4 USB Product Compliance

Adopters of the USB 2.0 specification have signed the USB 2.0 Adopters Agreement, which provides them access to a reciprocal royalty-free license from the Promoters and other Adopters to certain intellectual property contained in products that are compliant with the USB 2.0 specification. Adopters can demonstrate compliance with the specification through the testing program as defined by the USB Implementers Forum. Products that demonstrate compliance with the specification will be granted certain rights to use the USB Implementers Forum logo as defined in the logo license.

1.5 Document Organization

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

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

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

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

Microchip Ex. 1018, p. 30
Microchip v. Aptiv
IPR2024-00494

Chapter 2
Terms and Abbreviations

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

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

Microchip Ex. 1018, p. 31
Microchip v. Aptiv
IPR2024-00494

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

Microchip Ex. 1018, p. 32
Microchip v. Aptiv
IPR2024-00494

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

Microchip Ex. 1018, p. 33
Microchip v. Aptiv
IPR2024-00494

Frame	A 1 millisecond time base established on full-/low-speed buses.
Frame Pattern	A sequence of frames that exhibit a repeating pattern in the number of samples transmitted per frame. For a 44.1 kHz audio transfer, the frame pattern could be nine frames containing 44 samples followed by one frame containing 45 samples.
Fs	See sample rate.
Full-duplex	Computer data transmission occurring in both directions simultaneously.
Full-speed	USB operation at 12 Mb/s. See also low-speed and high-speed.
Function	A USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers.
Handshake Packet	A packet that acknowledges or rejects a specific condition. For examples, see ACK and NAK.
High-bandwidth endpoint	A high-speed device endpoint that transfers more than 1024 bytes and less than 3073 bytes per microframe.
High-speed	USB operation at 480 Mb/s. See also low-speed and full-speed.
Host	The host computer system where the USB Host Controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.
Host Controller	The host's USB interface.
Host Controller Driver (HCD)	The USB software layer that abstracts the Host Controller hardware. The Host Controller Driver provides an SPI for interaction with a Host Controller. The Host Controller Driver hides the specifics of the Host Controller hardware implementation.
Host Resources	Resources provided by the host, such as buffer space and interrupts. See also Device Resources and Universal Serial Bus Resources.
Hub	A USB device that provides additional connections to the USB.
Hub Tier	One plus the number of USB links in a communication path between the host and a function. See Figure 4-1.
Interrupt Request (IRQ)	A hardware signal that allows a device to request attention from a host. The host typically invokes an interrupt service routine to handle the condition that caused the request.
Interrupt Transfer	One of the four USB transfer types. Interrupt transfer characteristics are small data, non-periodic, low-frequency, and bounded-latency. Interrupt transfers are typically used to handle service needs. See also transfer type.
I/O Request Packet	An identifiable request by a software client to move data between itself (on the host) and an endpoint of a device in an appropriate direction.
IRP	See I/O Request Packet.
IRQ	See Interrupt Request.
Isochronous Data	A stream of data whose timing is implied by its delivery rate.
Isochronous Device	An entity with isochronous endpoints, as defined in the USB Specification, that sources or sinks sampled analog streams or synchronous data streams.

Microchip Ex. 1018, p. 34
Microchip v. Aptiv
IPR2024-00494

Isochronous Sink Endpoint	An endpoint that is capable of consuming an isochronous data stream that is sent by the host.
Isochronous Source Endpoint	An endpoint that is capable of producing an isochronous data stream and sending it to the host.
Isochronous Transfer	One of the four USB transfer types. Isochronous transfers are used when working with isochronous data. Isochronous transfers provide periodic, continuous communication between host and device. See also transfer type.
Jitter	A tendency toward lack of synchronization caused by mechanical or electrical changes. More specifically, the phase shift of digital pulses over a transmission medium.
kb/s	Transmission rate expressed in kilobits per second.
kB/s	Transmission rate expressed in kilobytes per second.
Little Endian	Method of storing data that places the least significant byte of multiple-byte values at lower storage addresses. For example, a 16-bit integer stored in little endian format places the least significant byte at the lower address and the most significant byte at the next address. See also big endian.
LOA	Loss of bus activity characterized by an SOP without a corresponding EOP.
Low-speed	USB operation at 1.5 Mb/s. See also full-speed and high-speed.
LSb	Least significant bit.
LSB	Least significant byte.
Mb/s	Transmission rate expressed in megabits per second.
MB/s	Transmission rate expressed in megabytes per second.
Message Pipe	A bi-directional pipe that transfers data using a request/data/status paradigm. The data has an imposed structure that allows requests to be reliably identified and communicated.
Microframe	A 125 microsecond time base established on high-speed buses.
MSb	Most significant bit.
MSB	Most significant byte.
NAK	Handshake packet indicating a negative acknowledgment.
Non Return to Zero Invert (NRZI)	A method of encoding serial data in which ones and zeroes are represented by opposite and alternating high and low voltages where there is no return to zero (reference) voltage between encoded bits. Eliminates the need for clock pulses.
NRZI	See Non Return to Zero Invert.
Object	Host software or data structure representing a USB entity.
Packet	A bundle of data organized in a group for transmission. Packets typically contain three elements: control information (e.g., source, destination, and length), the data to be transferred, and error detection and correction bits.
Packet Buffer	The logical buffer used by a USB device for sending or receiving a single packet. This determines the maximum packet size the device can send or receive.

Microchip Ex. 1018, p. 35
Microchip v. Aptiv
IPR2024-00494

Packet ID (PID)	A field in a USB packet that indicates the type of packet, and by inference, the format of the packet and the type of error detection applied to the packet.
Phase	A token, data, or handshake packet. A transaction has three phases.
Phase Locked Loop (PLL)	A circuit that acts as a phase detector to keep an oscillator in phase with an incoming frequency.
Physical Device	A device that has a physical implementation; e.g., speakers, microphones, and CD players.
PID	See Packet ID.
Pipe	A logical abstraction representing the association between an endpoint on a device and software on the host. A pipe has several attributes; for example, a pipe may transfer data as streams (stream pipe) or messages (message pipe). See also stream pipe and message pipe.
PLL	See Phase Locked Loop.
Polling	Asking multiple devices, one at a time, if they have any data to transmit.
POR	See Power On Reset.
Port	Point of access to or from a system or circuit. For the USB, the point where a USB device is attached.
Power On Reset (POR)	Restoring a storage device, register, or memory to a predetermined state when power is applied.
Programmable Data Rate	Either a fixed data rate (single-frequency endpoints), a limited number of data rates (32 kHz, 44.1 kHz, 48 kHz, …), or a continuously programmable data rate. The exact programming capabilities of an endpoint must be reported in the appropriate class-specific endpoint descriptors.
Protocol	A specific set of rules, procedures, or conventions relating to format and timing of data transmission between two devices.
RA	See rate adaptation.
Rate Adaptation	The process by which an incoming data stream, sampled at Fs_i, is converted to an outgoing data stream, sampled at Fs_o,with a certain loss of quality, determined by the rate adaptation algorithm. Error control mechanisms are required for the process. Fs_i and Fs_o can be different and asynchronous. Fs_i is the input data rate of the RA; Fs_o is the output data rate of the RA.
Request	A request made to a USB device contained within the data portion of a SETUP packet.
Retire	The action of completing service for a transfer and notifying the appropriate software client of the completion.
Root Hub	A USB hub directly attached to the Host Controller. This hub (tier 1) is attached to the host.
Root Port	The downstream port on a Root Hub.
Sample	The smallest unit of data on which an endpoint operates; a property of an endpoint.
Sample Rate (Fs)	The number of samples per second, expressed in Hertz (Hz).

Microchip Ex. 1018, p. 36
Microchip v. Aptiv
IPR2024-00494

Sample Rate Conversion (SRC)	A dedicated implementation of the RA process for use on sampled analog data streams. The error control mechanism is replaced by interpolating techniques.
Service	A procedure provided by a System Programming Interface (SPI).
Service Interval	The period between consecutive requests to a USB endpoint to send or receive data.
Service Jitter	The deviation of service delivery from its scheduled delivery time.
Service Rate	The number of services to a given endpoint per unit time.
SOF	See Start-of-Frame.
SOP	Start-of-Packet.
SPI	See System Programming Interface.
Split transaction	A transaction type supported by host controllers and hubs. This transaction type allows full- and low-speed devices to be attached to hubs operating at high-speed.
SRC	See Sample Rate Conversion.
Stage	One part of the sequence composing a control transfer; stages include the Setup stage, the Data stage, and the Status stage.
Start-of-Frame (SOF)	The first transaction in each (micro)frame. An SOF allows endpoints to identify the start of the (micro)frame and synchronize internal endpoint clocks to the host.
Stream Pipe	A pipe that transfers data as a stream of samples with no defined USB structure.
Synchronization Type	A classification that characterizes an isochronous endpoint's capability to connect to other isochronous endpoints.
Synchronous RA	The incoming data rate, Fs_i, and the outgoing data rate, Fs_o, of the RA process are derived from the same master clock. There is a fixed relation between Fs_i and Fs_o.
Synchronous SRC	The incoming sample rate, Fs_i, and outgoing sample rate, Fs_o, of the SRC process are derived from the same master clock. There is a fixed relation between Fs_i and Fs_o.
System Programming Interface (SPI)	A defined interface to services provided by system software.
TDM	See Time Division Multiplexing.
TDR	See Time Domain Reflectometer.
Termination	Passive components attached at the end of cables to prevent signals from being reflected or echoed.
Time Division Multiplexing (TDM)	A method of transmitting multiple signals (data, voice, and/or video) simultaneously over one communications medium by interleaving a piece of each signal one after another.
Time Domain Reflectometer (TDR)	An instrument capable of measuring impedance characteristics of the USB signal lines.

Microchip Ex. 1018, p. 37
Microchip v. Aptiv
IPR2024-00494

Timeout	The detection of a lack of bus activity for some predetermined interval.
Token Packet	A type of packet that identifies what transaction is to be performed on the bus.
Transaction	The delivery of service to an endpoint; consists of a token packet, optional data packet, and optional handshake packet. Specific packets are allowed/required based on the transaction type.
Transaction translator	A functional component of a USB hub. The Transaction Translator responds to special high-speed transactions and translates them to full/low-speed transactions with full/low-speed devices attached on downstream facing ports.
Transfer	One or more bus transactions to move information between a software client and its function.
Transfer Type	Determines the characteristics of the data flow between a software client and its function. Four standard transfer types are defined: control, interrupt, bulk, and isochronous.
Turn-around Time	The time a device needs to wait to begin transmitting a packet after a packet has been received to prevent collisions on the USB. This time is based on the length and propagation delay characteristics of the cable and the location of the transmitting device in relation to other devices on the USB.
Universal Serial Bus Driver (USBD)	The host resident software entity responsible for providing common services to clients that are manipulating one or more functions on one or more Host Controllers.
Universal Serial Bus Resources	Resources provided by the USB, such as bandwidth and power. See also Device Resources and Host Resources.
Upstream	The direction of data flow towards the host. An upstream port is the port on a device electrically closest to the host that generates upstream data traffic from the hub. Upstream ports receive downstream data traffic.
USBD	See Universal Serial Bus Driver.
USB-IF	USB Implementers Forum, Inc. is a nonprofit corporation formed to facilitate the development of USB compliant products and promote the technology.
Virtual Device	A device that is represented by a software interface layer. An example of a virtual device is a hard disk with its associated device driver and client software that makes it able to reproduce an audio .WAV file.
Word	A data element that is two bytes (16 bits) in size.

Microchip Ex. 1018, p. 38
Microchip v. Aptiv
IPR2024-00494

Chapter 3
Background

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

3.1 Goals for the Universal Serial Bus

The USB is specified to be an industry-standard extension to the PC architecture with a focus on PC peripherals that enable consumer and business applications. The following criteria were applied in defining the architecture for the USB:

- Ease-of-use for PC peripheral expansion
- Low-cost solution that supports transfer rates up to 480 Mb/s
- Full support for real-time data for voice, audio, and video
- Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging
- Integration in commodity device technology
- Comprehension of various PC configurations and form factors
- Provision of a standard interface capable of quick diffusion into product
- Enabling new classes of devices that augment the PC's capability
- Full backward compatibility of USB 2.0 for devices built to previous versions of the specification

Microchip Ex. 1018, p. 39
Microchip v. Aptiv
IPR2024-00494

3.2 Taxonomy of Application Space

Figure 3-1 describes a taxonomy for the range of data traffic workloads that can be serviced over a USB. As can be seen, a 480 Mb/s bus comprehends the high-speed, full-speed, and low-speed data ranges. Typically, high-speed and full-speed data types may be isochronous, while low-speed data comes from interactive devices. The USB is primarily a PC bus but can be readily applied to other host-centric computing devices. The software architecture allows for future extension of the USB by providing support for multiple USB Host Controllers.

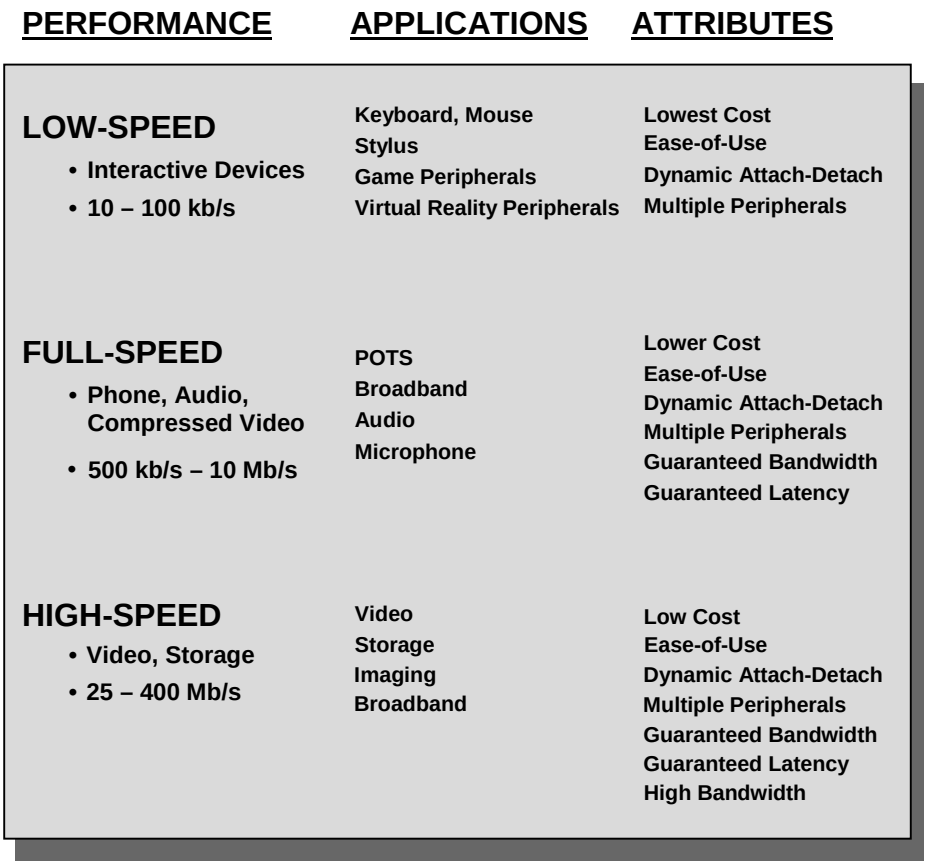


Figure 3-1. Application Space Taxonomy

Microchip Ex. 1018, p. 40
Microchip v. Aptiv
IPR2024-00494

3.3 Feature List

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

Easy to use for end user

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

Wide range of workloads and applications

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

Isochronous bandwidth

- Guaranteed bandwidth and low latencies appropriate for telephony, audio, video, etc.

Flexibility

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

Robustness

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

Synergy with PC industry

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

Microchip Ex. 1018, p. 41
Microchip v. Aptiv
IPR2024-00494

Low-cost implementation

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

Upgrade path

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

Microchip Ex. 1018, p. 42
Microchip v. Aptiv
IPR2024-00494

Chapter 4
Architectural Overview

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

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

4.1 USB System Description

A USB system is described by three definitional areas:

- USB interconnect
- USB devices
- USB host

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

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

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

Microchip Ex. 1018, p. 43
Microchip v. Aptiv
IPR2024-00494

4.1.1 Bus Topology

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

Due to timing constraints allowed for hub and cable propagation times, the maximum number of tiers allowed is seven (including the root tier). Note that in seven tiers, five non-root hubs maximum can be supported in a communication path between the host and any device. A compound device (see Figure 4-1) occupies two tiers; therefore, it cannot be enabled if attached at tier level seven. Only functions can be enabled in tier seven.

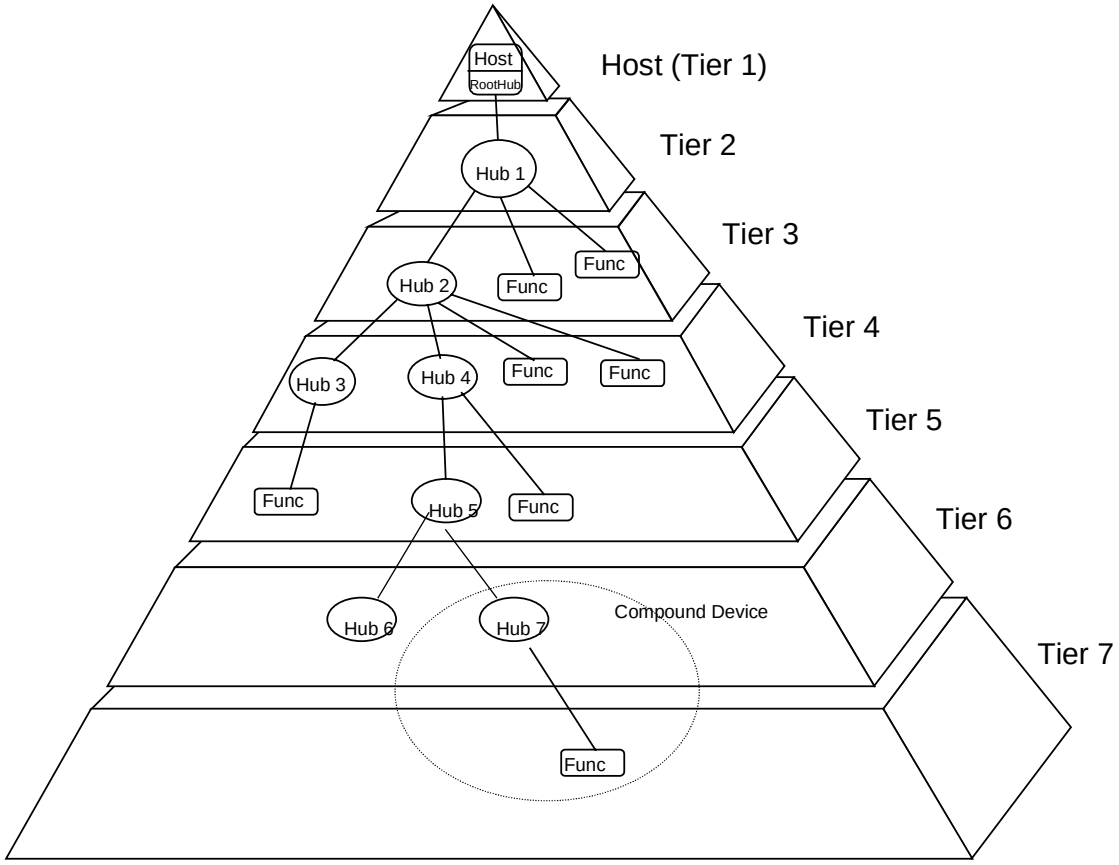


Figure 4-1. Bus Topology

4.1.1.1 USB Host

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

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

Microchip Ex. 1018, p. 44
Microchip v. Aptiv
IPR2024-00494

4.1.1.2 USB Devices

USB devices are one of the following:

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

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

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

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

4.2 Physical Interface

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

4.2.1 Electrical

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

There are three data rates:

- The USB high-speed signaling bit rate is 480 Mb/s.
- The USB full-speed signaling bit rate is 12 Mb/s.
- A limited capability low-speed signaling mode is also defined at 1.5 Mb/s.

USB 2.0 host controllers and hubs provide capabilities so that full-speed and low-speed data can be transmitted at high-speed between the host controller and the hub, but transmitted between the hub and the device at full-speed or low-speed. This capability minimizes the impact that full-speed and low-speed devices have upon the bandwidth available for high-speed devices.

The low-speed mode is defined to support a limited number of low-bandwidth devices, such as mice, because more general use would degrade bus utilization.

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

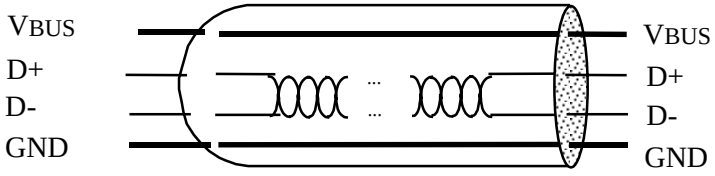


Figure 4-2. USB Cable

Microchip Ex. 1018, p. 45
Microchip v. Aptiv
IPR2024-00494

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

4.2.2 Mechanical

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

4.3 Power

The specification covers two aspects of power:

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

4.3.1 Power Distribution

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

4.3.2 Power Management

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

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

4.4 Bus Protocol

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

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

Microchip Ex. 1018, p. 46
Microchip v. Aptiv
IPR2024-00494

data to transfer. The destination, in general, responds with a handshake packet indicating whether the transfer was successful.

Some bus transactions between host controllers and hubs involve the transmission of four packets. These types of transactions are used to manage the data transfers between the host and full-/low- speed devices.

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

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

4.5 Robustness

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

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

4.5.1 Error Detection

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

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

4.5.2 Error Handling

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

4.6 System Configuration

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

Microchip Ex. 1018, p. 47
Microchip v. Aptiv
IPR2024-00494

4.6.1 Attachment of USB Devices

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

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

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

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

4.6.2 Removal of USB Devices

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

4.6.3 Bus Enumeration

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

4.7 Data Flow Types

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

The USB architecture comprehends four basic types of data transfers:

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

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

Microchip Ex. 1018, p. 48
Microchip v. Aptiv
IPR2024-00494

4.7.1 Control Transfers

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

4.7.2 Bulk Transfers

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

4.7.3 Interrupt Transfers

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

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

4.7.4 Isochronous Transfers

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

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

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

4.7.5 Allocating USB Bandwidth

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

The USB's bandwidth capacity can be allocated among many different data streams. This allows a wide range of devices to be attached to the USB. Further, different device bit rates, with a wide dynamic range, can be concurrently supported.

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

Microchip Ex. 1018, p. 49
Microchip v. Aptiv
IPR2024-00494

4.8 USB Devices

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

4.8.1 Device Characterizations

All USB devices are accessed by a USB address that is assigned when the device is attached and enumerated. Each USB device additionally supports one or more pipes through which the host may communicate with the device. All USB devices must support a specially designated pipe at endpoint zero to which the USB device's USB control pipe will be attached. All USB devices support a common access mechanism for accessing information through this control pipe.

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

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

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

4.8.2 Device Descriptions

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

4.8.2.1 Hubs

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

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

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

A USB 2.0 hub consists of three portions: the Hub Controller, the Hub Repeater, and the Transaction Translator. The Hub Repeater is a protocol-controlled switch between the upstream port and downstream ports. It also has hardware support for reset and suspend/resume signaling. The Host Controller provides the communication to/from the host. Hub-specific status and control commands permit the host to configure a hub and to monitor and control its ports. The Transaction Translator provides the mechanisms that support full-/low-speed devices behind the hub, while transmitting all device data between the host and the hub at high-speed.

Microchip Ex. 1018, p. 50
Microchip v. Aptiv
IPR2024-00494

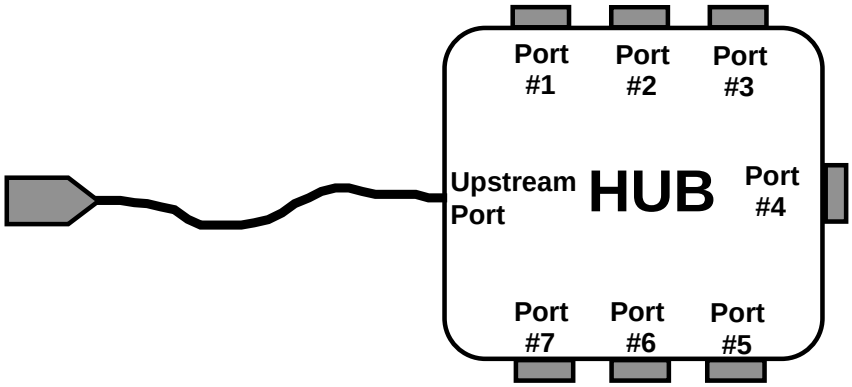


Figure 4-3. A Typical Hub

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

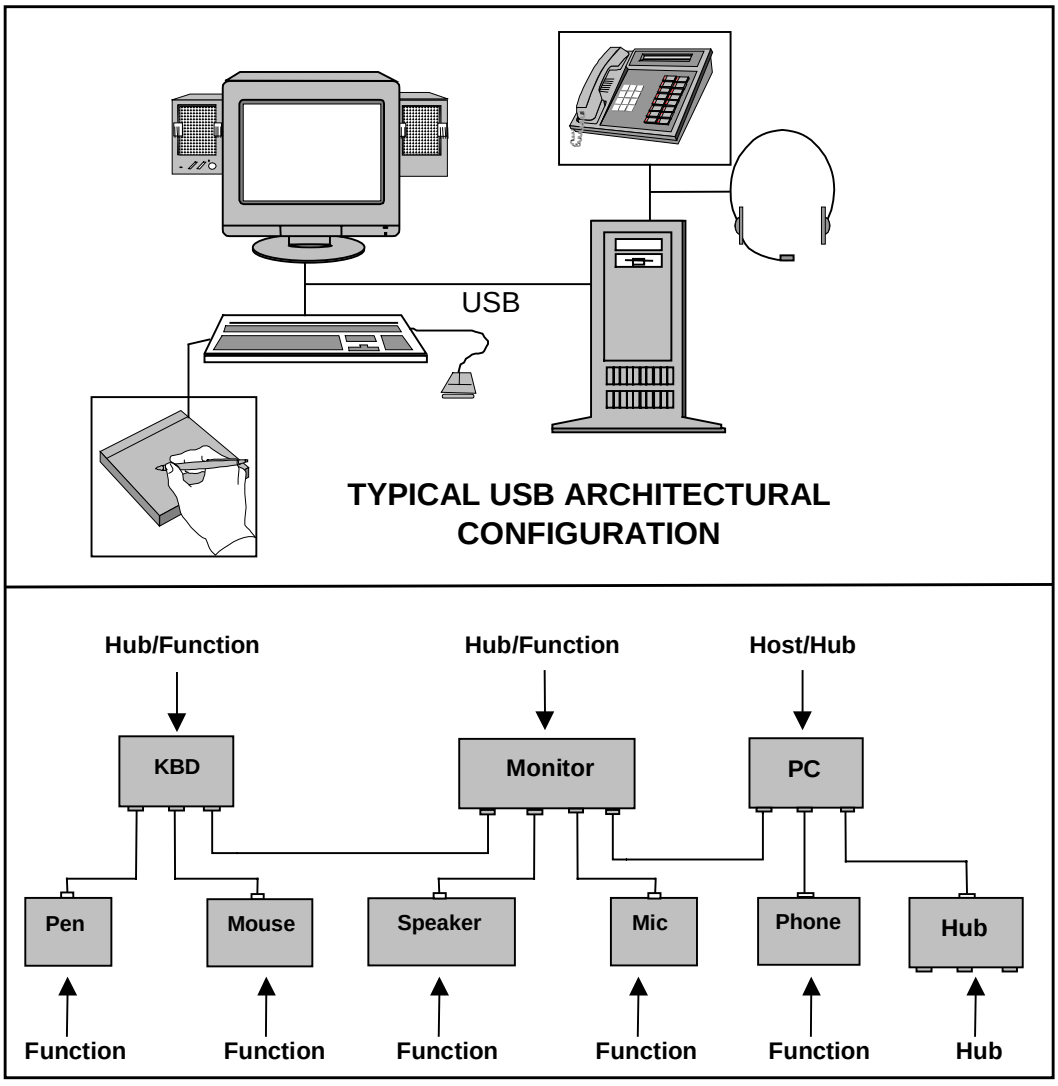


Figure 4-4. Hubs in a Desktop Computer Environment

Microchip Ex. 1018, p. 51
Microchip v. Aptiv
IPR2024-00494

4.8.2.2 Functions

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

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

Examples of functions include the following:

- A human interface device such as a mouse, keyboard, tablet, or game controller
- An imaging device such as a scanner, printer, or camera
- A mass storage device such as a CD-ROM drive, floppy drive, or DVD drive

4.9 USB Host: Hardware and Software

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

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

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

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

4.10 Architectural Extensions

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

Microchip Ex. 1018, p. 52
Microchip v. Aptiv
IPR2024-00494

Chapter 5
USB Data Flow Model

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

5.1 Implementer Viewpoints

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

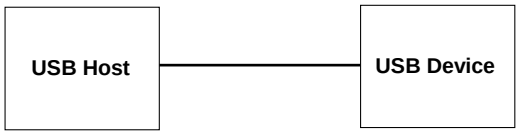


Figure 5-1. Simple USB Host/Device View

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

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

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

Microchip Ex. 1018, p. 53
Microchip v. Aptiv
IPR2024-00494

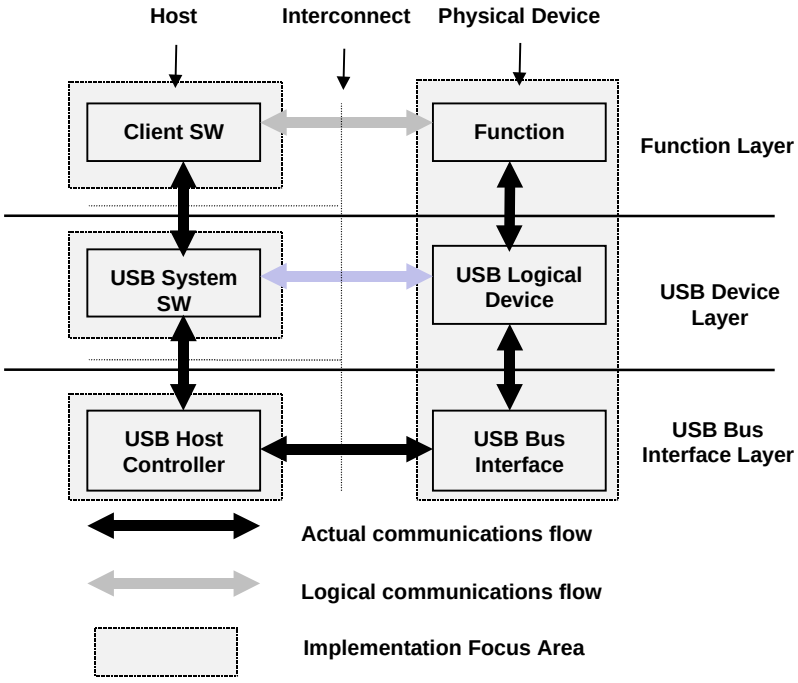


Figure 5-2. USB Implementation Areas

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

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

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

- Bus Topology: Section 5.2 presents the primary physical and logical components of the USB and how they interrelate.
- Communication Flow Models: Sections 5.3 through 5.8 describe how communication flows between the host and devices through the USB and defines the four USB transfer types.
- Bus Access Management: Section 5.11 describes how bus access is managed within the host to support a broad range of communication flows by USB devices.
- Special Consideration for Isochronous Transfers: Section 5.12 presents features of the USB specific to devices requiring isochronous data transfers. Device implementers for non-isochronous devices do not need to read Section 5.12.

Microchip Ex. 1018, p. 54
Microchip v. Aptiv
IPR2024-00494

5.2 Bus Topology

There are four main parts to USB topology:

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

5.2.1 USB Host

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

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

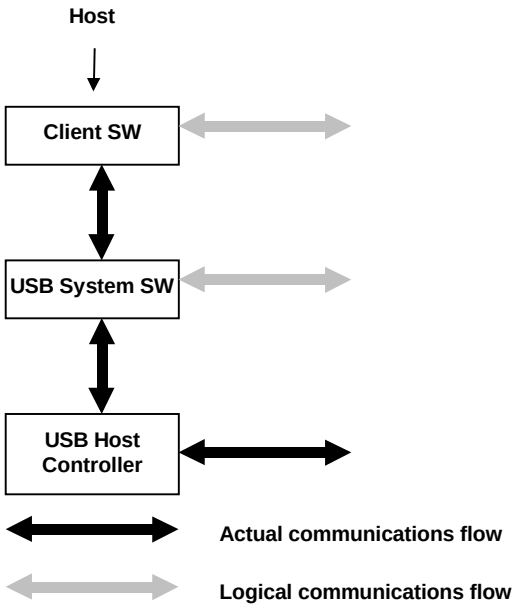


Figure 5-3. Host Composition

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

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

Microchip Ex. 1018, p. 55
Microchip v. Aptiv
IPR2024-00494

5.2.2 USB Devices

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

- USB bus interface
- USB logical device
- Function

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

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

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

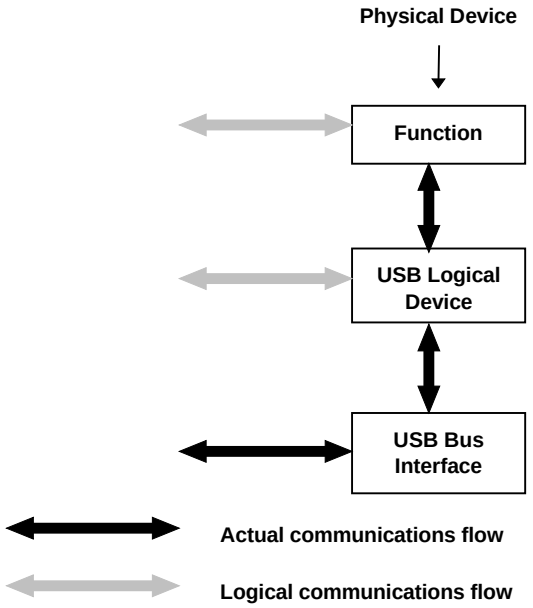


Figure 5-4. Physical Device Composition

Microchip Ex. 1018, p. 56
Microchip v. Aptiv
IPR2024-00494

5.2.3 Physical Bus Topology

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

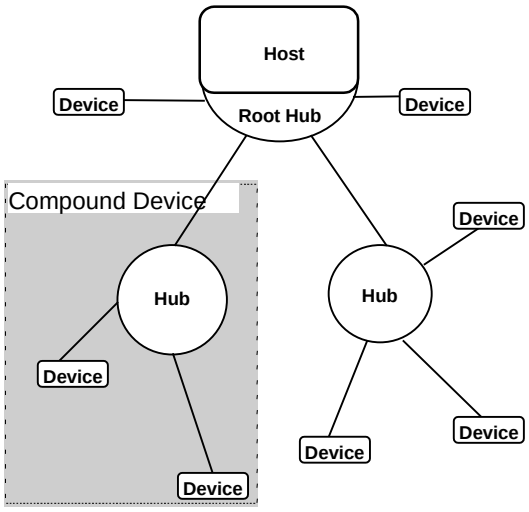


Figure 5-5. USB Physical Bus Topology

Multiple functions may be packaged together in what appears to be a single physical device. For example, a keyboard and a trackball might be combined in a single package. Inside the package, the individual functions are permanently attached to a hub and it is the internal hub that is connected to the USB. When multiple functions are combined with a hub in a single package, they are referred to as a compound device. The hub and each function attached to the hub within the compound device is assigned its own device address. A device that has multiple interfaces controlled independently of each other is referred to as a composite device. A composite device has only a single device address. From the host's perspective, a compound device is the same as a separate hub with multiple functions attached. Figure 5-5 also illustrates a compound device.

Microchip Ex. 1018, p. 57
Microchip v. Aptiv
IPR2024-00494

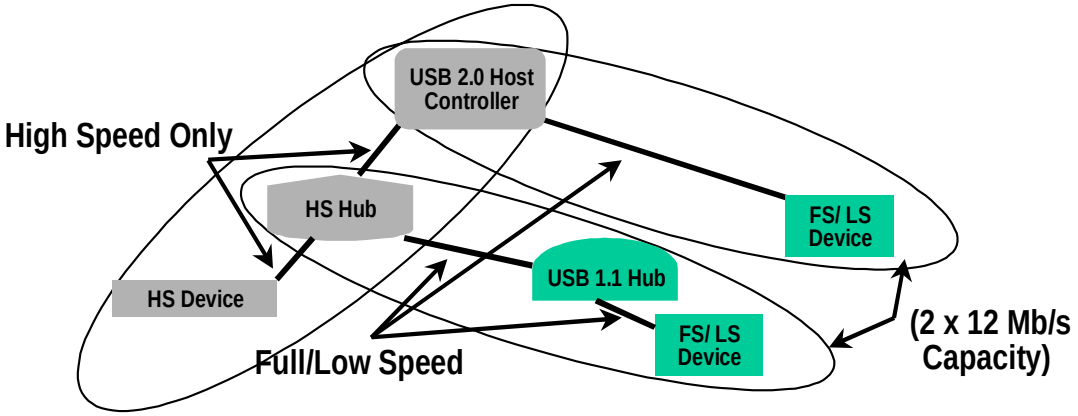


Figure 5-6. Multiple Full-speed Buses in a High-speed System

The hub plays a special role in a high-speed system. The hub isolates the full-/low-speed signaling environment from the high-speed signaling environment. Figure 5-6 shows a hub operating in high speed supporting a high-speed attached device. The hub also allows USB1.1 hubs to attach and operate at full-/low-speed along with other full-/low-speed only devices. The host controller also directly supports attaching full-/low-speed only devices. Chapter 11 describes the details of how the hub accomplishes the isolation of the two signaling environments.

Each high-speed operating hub essentially adds one (or more) additional full-/low-speed buses; i.e., each hub supports additional (optionally multiple) 12 Mb/s of USB full-/low-speed bandwidth. This allows more full-/low-speed buses to be attached without requiring additional host controllers in a system. Even though there can be several 12 Mb/s full-/low-speed buses, there are only at most 127 USB devices attached to any single host controller.

5.2.4 Logical Bus Topology

While devices physically attach to the USB in a tiered, star topology, the host communicates with each logical device as if it were directly connected to the root port. This creates the logical view illustrated in Figure 5-7 that corresponds to the physical topology shown in Figure 5-5. Hubs are logical devices also but are not shown in Figure 5-7 to simplify the picture. Even though most host/logical device activities use this logical perspective, the host maintains an awareness of the physical topology to support processing the removal of hubs. When a hub is removed, all of the devices attached to the hub must be removed from the host's view of the logical topology. A more complete discussion of hubs can be found in Chapter 11.

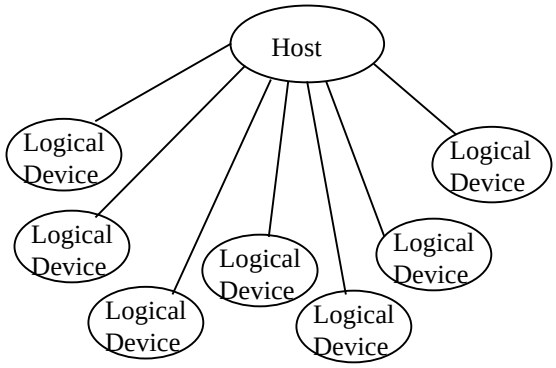


Figure 5-7. USB Logical Bus Topology

Microchip Ex. 1018, p. 58
Microchip v. Aptiv
IPR2024-00494

5.2.5 Client Software-to-function Relationship

Even though the physical and logical topology of the USB reflects the shared nature of the bus, client software (CSw) manipulating a USB function interface is presented with the view that it deals only with its interface(s) of interest. Client software for USB functions must use USB software programming interfaces to manipulate their functions as opposed to directly manipulating their functions via memory or I/O accesses as with other buses (e.g., PCI, EISA, PCMCIA, etc.). During operation, client software should be independent of other devices that may be connected to the USB. This allows the designer of the device and client software to focus on the hardware/software interaction design details. Figure 5-8 illustrates a device designer's perspective of the relationships of client software and USB functions with respect to the USB logical topology of Figure 5-7.

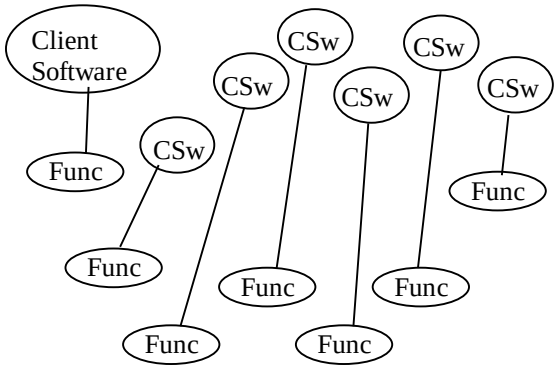


Figure 5-8. Client Software-to-function Relationships

5.3 USB Communication Flow

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

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

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

Microchip Ex. 1018, p. 59
Microchip v. Aptiv
IPR2024-00494

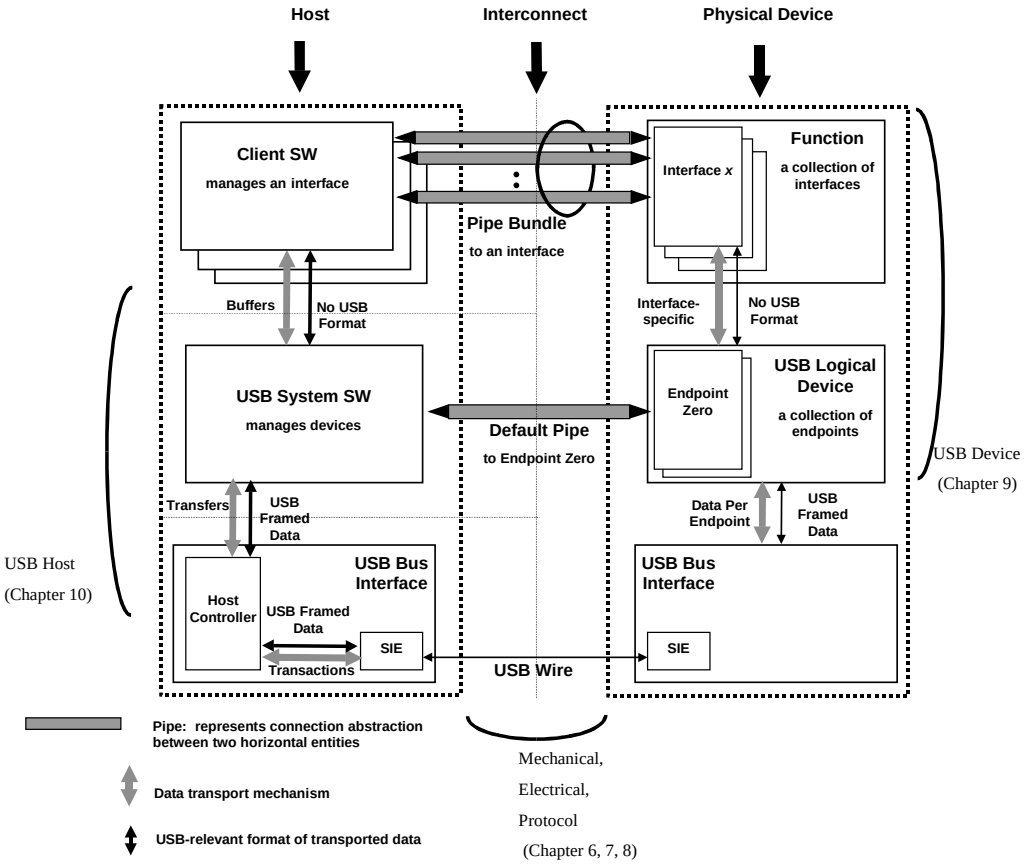


Figure 5-9. USB Host/Device Detailed View

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

Microchip Ex. 1018, p. 60
Microchip v. Aptiv
IPR2024-00494

Figure 5-10 illustrates how communication flows are carried over pipes between endpoints and host side memory buffers. The following sections describe endpoints, pipes, and communication flows in more detail.

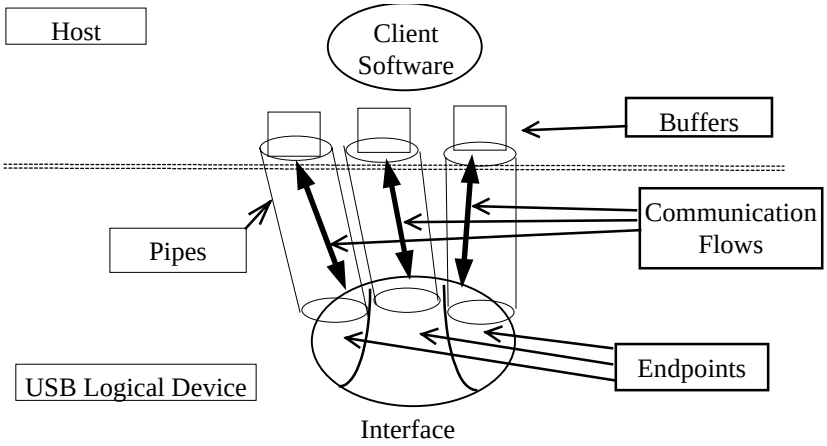


Figure 5-10. USB Communication Flow

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

5.3.1 Device Endpoints

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

An endpoint has characteristics that determine the type of transfer service required between the endpoint and the client software. An endpoint describes itself by:

- Bus access frequency/latency requirement
- Bandwidth requirement
- Endpoint number
- Error handling behavior requirements
- Maximum packet size that the endpoint is capable of sending or receiving
- The transfer type for the endpoint (refer to Section 5.4 for details)
- The direction in which data is transferred between the endpoint and the host

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

Microchip Ex. 1018, p. 61
Microchip v. Aptiv
IPR2024-00494

5.3.1.1 Endpoint Zero Requirements

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

A USB device that is capable of operating at high-speed must have a minimum level of support for operating at full-speed. When the device is attached to a hub operating in full-speed, the device must:

- Be able to reset successfully at full-speed
- Respond successfully to standard requests: set_address, set_configuration, get_descriptor for device and configuration descriptors, and return appropriate information

The high-speed device may or may not be able to support its intended functionality when operating at full-speed.

5.3.1.2 Non-endpoint Zero Requirements

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

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

5.3.2 Pipes

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

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

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

Additionally, pipes have the following associated with them:

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

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

Microchip Ex. 1018, p. 62
Microchip v. Aptiv
IPR2024-00494

Software retains “ownership” of the Default Control Pipe and mediates use of the pipe by other client software.

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

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

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

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

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

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

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

5.3.2.1 Stream Pipes

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

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

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

Microchip Ex. 1018, p. 63
Microchip v. Aptiv
IPR2024-00494

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

5.3.2.2 Message Pipes

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

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

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

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

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

5.3.3 Frames and Microframes

USB establishes a 1 millisecond time base called a frame on a full-/low-speed bus and a 125 μs time base called a microframe on a high-speed bus. A (micro)frame can contain several transactions. Each transfer type defines what transactions are allowed within a (micro)frame for an endpoint. Isochronous and interrupt endpoints are given opportunities to the bus every N (micro)frames. The values of N and other details about isochronous and interrupt transfers are described in Sections 5.6 and 5.7.

5.4 Transfer Types

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

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

- Data format imposed by the USB
- Direction of communication flow
- Packet size constraints

Microchip Ex. 1018, p. 64
Microchip v. Aptiv
IPR2024-00494

- Bus access constraints
- Latency constraints
- Required data sequences
- Error handling

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

The USB defines four transfer types:

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

Each transfer type is described in detail in the following four major sections. The data for any IRP is carried by the data field of the data packet as described in Section 8.3.4. Chapter 8 also describes details of the protocol that are affected by use of each particular transfer type.

5.4.1 Table Calculation Examples

The following sections describe each of the USB transfer types. In these sections, there are tables that illustrate the maximum number of transactions that can be expected to be contained in a (micro)frame. These tables can be used to determine the maximum performance behavior possible for a specific transfer type. Actual performance may vary with specific system implementation details.

Each table shows:

- The protocol overhead required for the specific transfer type (and speed)
- For some sample data payload sizes:
    - The maximum sustained bandwidth possible for this case
    - The percentage of a (micro)frame that each transaction requires
    - The maximum number of transactions in a (micro)frame for the specific case
    - The remaining bytes in a (micro)frame that would not be required for the specific case
    - The total number of data bytes transported in a single (micro)frame for the specific case

A transaction of a particular transfer type typically requires multiple packets. The protocol overhead for each transaction includes:

- A SYNC field for each packet: either 8 bits (full-/low-speed) or 32 bits (high-speed)
- A PID byte for each packet: includes PID and PID invert (check) bits
- An EOP for each packet: 3 bits (full-/low-speed) or 8 bits (high-speed)
- In a token packet, the endpoint number, device address, and CRC5 fields (16 bits total)

Microchip Ex. 1018, p. 65
Microchip v. Aptiv
IPR2024-00494

- In a data packet, CRC16 fields (16 bits total)
- In a data packet, any data field (8 bits per byte)
- For transaction with multiple packets, the inter packet gap or bus turnaround time required.

For these calculations, there is assumed to be no bit-stuffing required.

Using the low speed interrupt OUT as an example, there are 5 packets in the transaction:

- A PRE special packet
- A token packet
- A PRE special packet
- A data packet
- A handshake packet

There is one bus turnaround between the data and handshake packets. The protocol overhead is therefore:

5 SYNC, 5 PID, Endpoint + CRC5, CRC16, 5 EOPs and interpacket delay (one bus turnaround, 1 delay between packets, and 2 hub setup times).

5.5 Control Transfers

Control transfers allow access to different parts of a device. Control transfers are intended to support configuration/command/status type communication flows between client software and its function. A control transfer is composed of a Setup bus transaction moving request information from host to function, zero or more Data transactions sending data in the direction indicated by the Setup transaction, and a Status transaction returning status information from function to host. The Status transaction returns “success” when the endpoint has successfully completed processing the requested operation. Section 8.5.3 describes the details of what packets, bus transactions, and transaction sequences are used to accomplish a control transfer. Chapter 9 describes the details of the defined USB command codes.

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

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

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

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

5.5.1 Control Transfer Data Format

The Setup packet has a USB-defined structure that accommodates the minimum set of commands required to enable communication between the host and a device. The structure definition allows vendor-specific extensions for device specific commands. The Data transactions following Setup have a USB-defined structure except when carrying vendor-specific information. The Status transaction also has a USB-defined structure. Specific control transfer Setup/Data definitions are described in Section 8.5.3 and Chapter 9.

Microchip Ex. 1018, p. 66
Microchip v. Aptiv
IPR2024-00494

5.5.2 Control Transfer Direction

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

5.5.3 Control Transfer Packet Size Constraints

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

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

All Host Controllers are required to have support for 8-, 16-, 32-, and 64-byte maximum data payload sizes for full-speed control endpoints, only 8-byte maximum data payload sizes for low-speed control endpoints, and only 64-byte maximum data payload size for high-speed control endpoints. No Host Controller is required to support larger or smaller maximum data payload sizes.

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

An endpoint must always transmit data payloads with a data field less than or equal to the endpoint's *wMaxPacketSize* (refer to Chapter 9). When a control transfer involves more data than can fit in one data payload of the currently established maximum size, all data payloads are required to be maximum-sized except for the last data payload, which will contain the remaining data.

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

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

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

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

Microchip Ex. 1018, p. 67
Microchip v. Aptiv
IPR2024-00494

5.5.4 Control Transfer Bus Access Constraints

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

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

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

- If the control transfers that are attempted (in an implementation-dependent fashion) consume less than 10% of the frame time for full-/low-speed endpoints or less than 20% of a microframe for high-speed endpoints, the remaining time can be used to support bulk transfers (refer to Section 5.8).
- A control transfer that has been attempted and needs to be retried can be retried in the current or a future (micro)frame; i.e., it is not required to be retried in the same (micro)frame.
- If there are more control transfers than reserved time, but there is additional (micro)frame time that is not being used for isochronous or interrupt transfers, a Host Controller may move additional control transfers as they are available.
- If there are too many pending control transfers for the available (micro)frame time, control transfers are selected to be moved over the bus as appropriate.
- If there are control transfers pending for multiple endpoints, control transfers for the different endpoints are selected according to a fair access policy that is Host Controller implementation-dependent.
- A transaction of a control transfer that is frequently being retried should not be expected to consume an unfair share of the bus time.

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

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

The USB System Software can, at its discretion, vary the rate of control transfers to a particular endpoint. An endpoint and its client software cannot assume a specific rate of service for control transfers. A control endpoint may see zero or more transfers in a single (micro)frame. Bus time made available to a software client and its endpoint can be changed as other devices are inserted into and removed from the system or also as control transfers are requested for other device endpoints.

The bus frequency and (micro)frame timing limit the maximum number of successful control transfers within a (micro)frame for any USB system. For full-/low-speed buses, the number of successful control transfers per frame is limited to less than 29 full-speed eight-byte data payloads or less than four low-speed eight-byte data payloads. For high-speed buses, the number of control transfers is limited to less than 32 high-speed 64-byte data payloads per microframe.

Table 5-1 lists information about different-sized low-speed packets and the maximum number of packets possible in a frame. The table does not include the overhead associated with bit stuffing.

Microchip Ex. 1018, p. 68
Microchip v. Aptiv
IPR2024-00494

Table 5-1. Low-speed Control Transfer Limits

	Protocol Overhead (63 bytes)		(15 SYNC bytes, 15 PID bytes, 6 Endpoint + CRC bytes, 6 CRC bytes, 8 Setup data bytes, and a 13-byte interpacket delay (EOP, etc.))			
	Data Payload	**Max Bandwidth (bytes/second)**	**Frame Bandwidth per Transfer**	**Max Transfers**	**Bytes Remaining**	**Bytes/Frame Useful Data**
	1	3000	26%	3	40	3
	2	6000	27%	3	37	6
	4	12000	28%	3	31	12
	8	24000	30%	3	19	24
Max		187500				187

For all speeds, because a control transfer is composed of several packets, the packets can be spread over several (micro)frames to spread the bus time required across several (micro)frames.

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

For high-speed endpoints, the 20% microframe reservation for non-periodic transfers means that all high speed control transfers are contending for nominally six control transfers per microframe. High-speed control transfers contend for microframe time along with split-transactions (see Sections 11.15-11.21 for more information about split transactions) for full- and low-speed control transfers. Both full-speed and low-speed control transfers contend for the same available frame time. However, high-speed control transfers for some endpoints can occur simultaneously with full- and low-speed control transfers for other endpoints. Low-speed control transfers simply take longer to transfer.

Microchip Ex. 1018, p. 69
Microchip v. Aptiv
IPR2024-00494

Table 5-2 lists information about different-sized full-speed control transfers and the maximum number of transfers possible in a frame. This table was generated assuming that there is one Data stage transaction and that the Data stage has a zero-length status phase. The table illustrates the possible power of two data payloads less than or equal to the allowable maximum data payload sizes. The table does not include the overhead associated with bit stuffing.

Table 5-2. Full-speed Control Transfer Limits

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

Microchip Ex. 1018, p. 70
Microchip v. Aptiv
IPR2024-00494

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

Table 5-3. High-speed Control Transfer Limits

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

5.5.5 Control Transfer Data Sequences

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

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

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

Microchip Ex. 1018, p. 71
Microchip v. Aptiv
IPR2024-00494

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

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

5.6 Isochronous Transfers

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

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

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

5.6.1 Isochronous Transfer Data Format

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

5.6.2 Isochronous Transfer Direction

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

5.6.3 Isochronous Transfer Packet Size Constraints

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

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

Microchip Ex. 1018, p. 72
Microchip v. Aptiv
IPR2024-00494

Table 5-4. Full-speed Isochronous Transaction Limits

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

Microchip Ex. 1018, p. 73
Microchip v. Aptiv
IPR2024-00494

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

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

Table 5-5. High-speed Isochronous Transaction Limits

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

Microchip Ex. 1018, p. 74
Microchip v. Aptiv
IPR2024-00494

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

5.6.4 Isochronous Transfer Bus Access Constraints

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

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

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

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

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

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

5.6.5 Isochronous Transfer Data Sequences

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

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

Microchip Ex. 1018, p. 75
Microchip v. Aptiv
IPR2024-00494

5.7 Interrupt Transfers

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

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

5.7.1 Interrupt Transfer Data Format

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

5.7.2 Interrupt Transfer Direction

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

5.7.3 Interrupt Transfer Packet Size Constraints

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

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

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

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

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

Microchip Ex. 1018, p. 76
Microchip v. Aptiv
IPR2024-00494

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

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

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

5.7.4 Interrupt Transfer Bus Access Constraints

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

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

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

Table 5-6. Low-speed Interrupt Transaction Limits

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

Microchip Ex. 1018, p. 77
Microchip v. Aptiv
IPR2024-00494

Table 5-7. Full-speed Interrupt Transaction Limits

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

Microchip Ex. 1018, p. 78
Microchip v. Aptiv
IPR2024-00494

Table 5-8. High-speed Interrupt Transaction Limits

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

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

Microchip Ex. 1018, p. 79
Microchip v. Aptiv
IPR2024-00494

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

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

5.7.5 Interrupt Transfer Data Sequences

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

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

5.8 Bulk Transfers

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

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

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

5.8.1 Bulk Transfer Data Format

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

5.8.2 Bulk Transfer Direction

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

Microchip Ex. 1018, p. 80
Microchip v. Aptiv
IPR2024-00494

5.8.3 Bulk Transfer Packet Size Constraints

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

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

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

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

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

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

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

5.8.4 Bulk Transfer Bus Access Constraints

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

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

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

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

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

Microchip Ex. 1018, p. 81
Microchip v. Aptiv
IPR2024-00494

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

Table 5-9. Full-speed Bulk Transaction Limits

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

Microchip Ex. 1018, p. 82
Microchip v. Aptiv
IPR2024-00494

Table 5-10. High-speed Bulk Transaction Limits

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

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

5.8.5 Bulk Transfer Data Sequences

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

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

Microchip Ex. 1018, p. 83
Microchip v. Aptiv
IPR2024-00494

5.9 High-Speed, High Bandwidth Endpoints

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

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

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

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

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

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

5.9.1 High Bandwidth Interrupt Endpoints

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

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

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

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

Microchip Ex. 1018, p. 84
Microchip v. Aptiv
IPR2024-00494

5.9.2 High Bandwidth Isochronous Endpoints

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

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

No retries are ever done for isochronous endpoints.

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

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

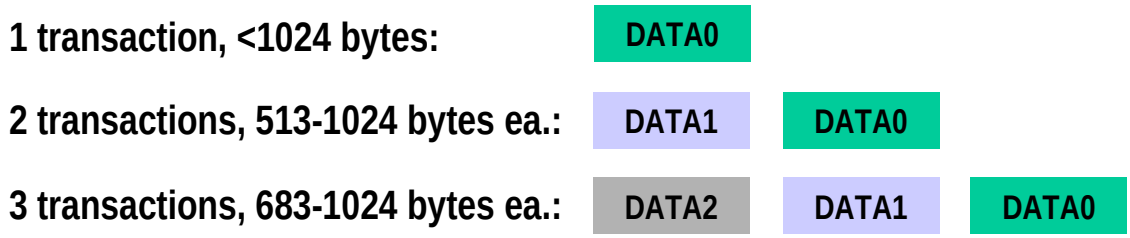


Figure 5-11. Data Phase PID Sequence for Isochronous IN High Bandwidth Endpoints

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

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

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

Microchip Ex. 1018, p. 85
Microchip v. Aptiv
IPR2024-00494

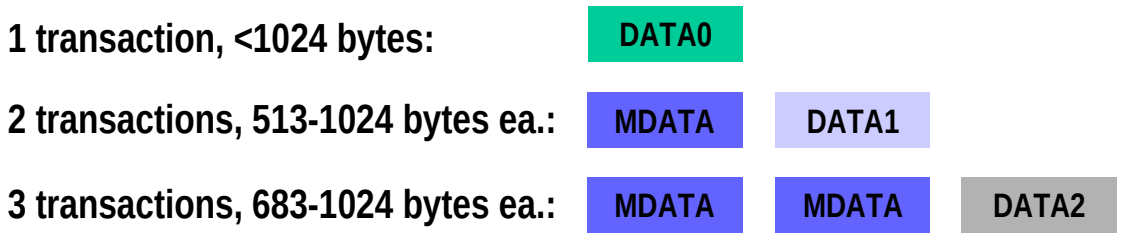


Figure 5-12. Data Phase PID Sequence for Isochronous OUT High Bandwidth Endpoints

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

5.10 Split Transactions

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

5.11 Bus Access for Transfers

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

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

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

Microchip Ex. 1018, p. 86
Microchip v. Aptiv
IPR2024-00494

5.11.1 Transfer Management

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

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

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

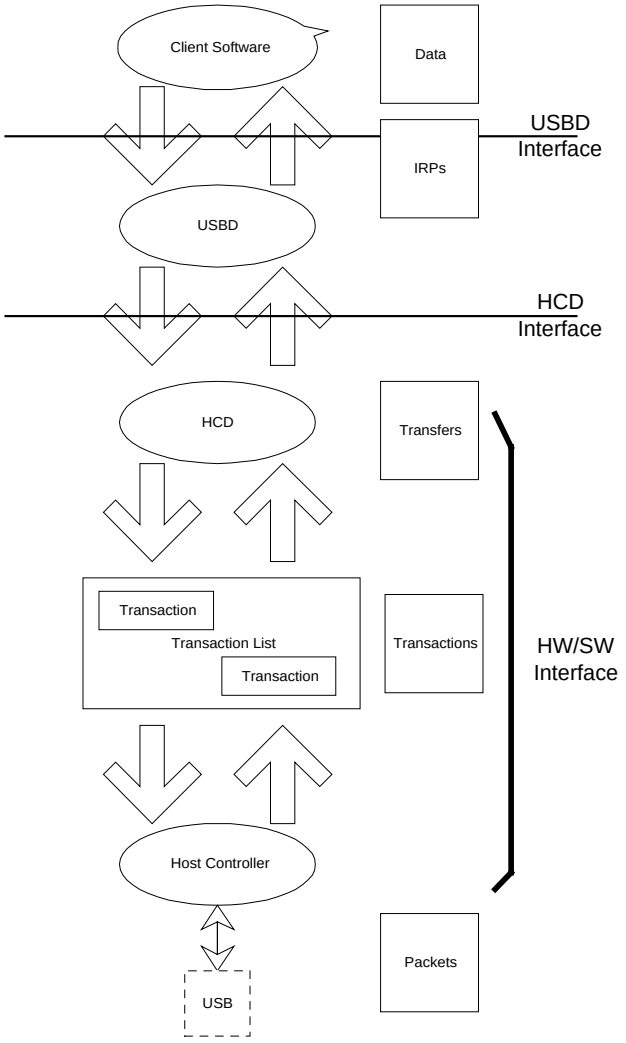


Figure 5-13. USB Information Conversion From Client Software to Bus

Microchip Ex. 1018, p. 87
Microchip v. Aptiv
IPR2024-00494

5.11.1.1 Client Software

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

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

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

The USBD interface is defined in Chapter 10.

5.11.1.2 USB Driver

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

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

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

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

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

5.11.1.3 Host Controller Driver

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

IRPs are defined in an operating system-dependent manner.

Microchip Ex. 1018, p. 88
Microchip v. Aptiv
IPR2024-00494

5.11.1.4 Transaction List

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

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

5.11.1.5 Host Controller

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

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

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

5.11.2 Transaction Tracking

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

Microchip Ex. 1018, p. 89
Microchip v. Aptiv
IPR2024-00494

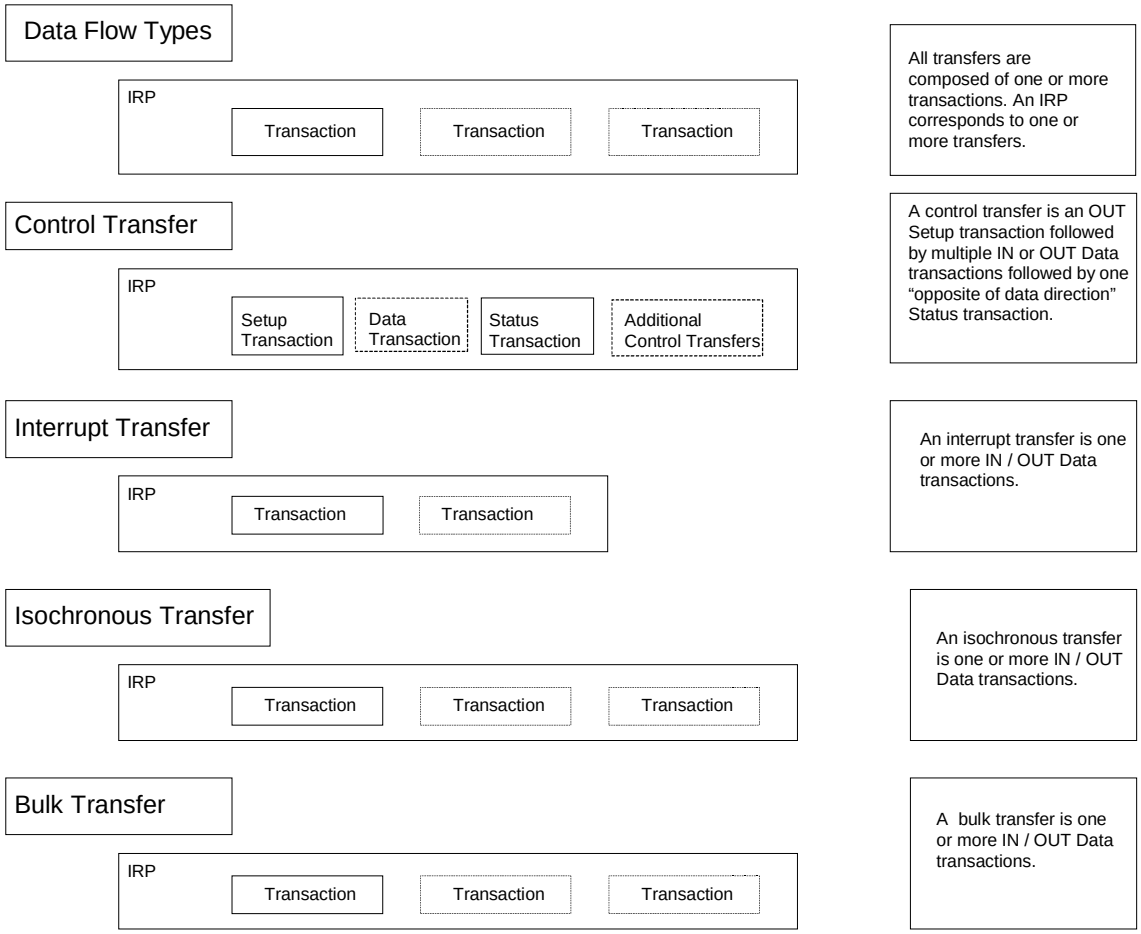


Figure 5-14. Transfers for Communication Flows

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

Microchip Ex. 1018, p. 90
Microchip v. Aptiv
IPR2024-00494

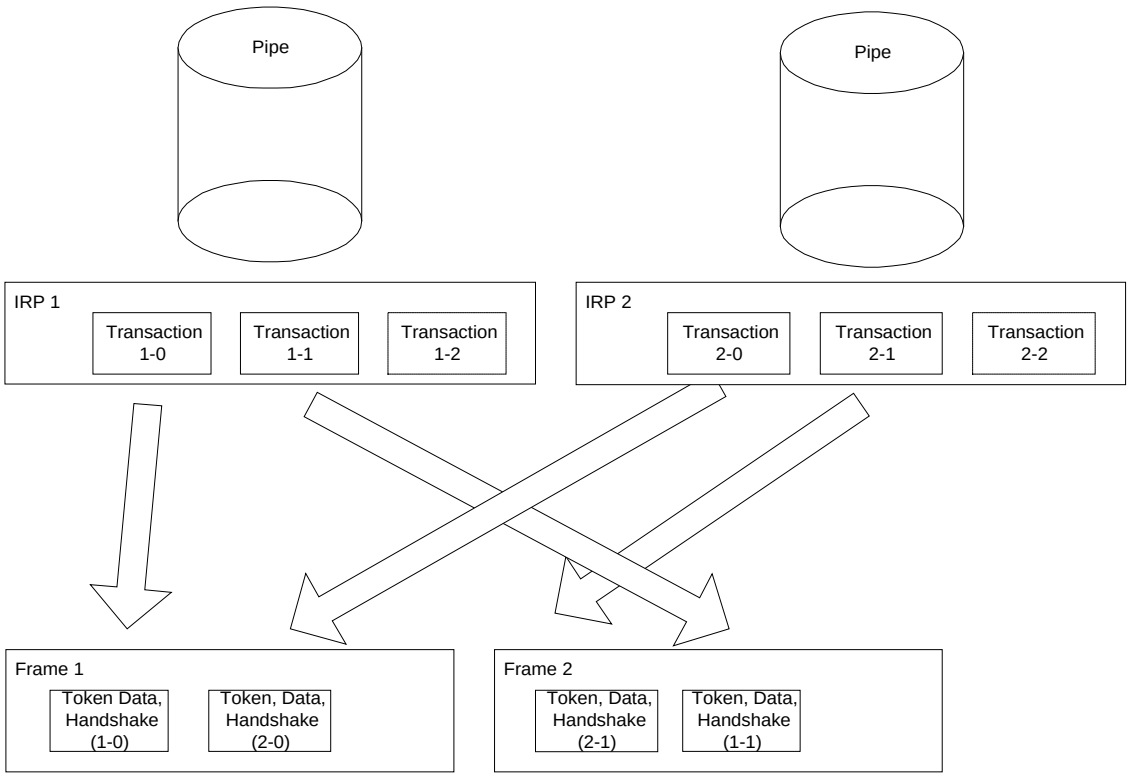


Figure 5-15. Arrangement of IRPs to Transactions/(Micro)frames

5.11.3 Calculating Bus Transaction Times

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

KEY:

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

Microchip Ex. 1018, p. 91
Microchip v. Aptiv
IPR2024-00494

High-speed (Input)

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

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

High-speed (Output)

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


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

Full-speed (Input)

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

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

Full-speed (Output)

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


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

Low-speed (Input)

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

Low-speed (Output)

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

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

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

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

Microchip Ex. 1018, p. 92
Microchip v. Aptiv
IPR2024-00494

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

5.11.4 Calculating Buffer Sizes in Functions and Software

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

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

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

5.11.5 Bus Bandwidth Reclamation

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

5.12 Special Considerations for Isochronous Transfers

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

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

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

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

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

Microchip Ex. 1018, p. 93
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

The remainder of the section presents the following key concepts:

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

5.12.1 Example Non-USB Isochronous Application

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

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

Microchip Ex. 1018, p. 94
Microchip v. Aptiv
IPR2024-00494

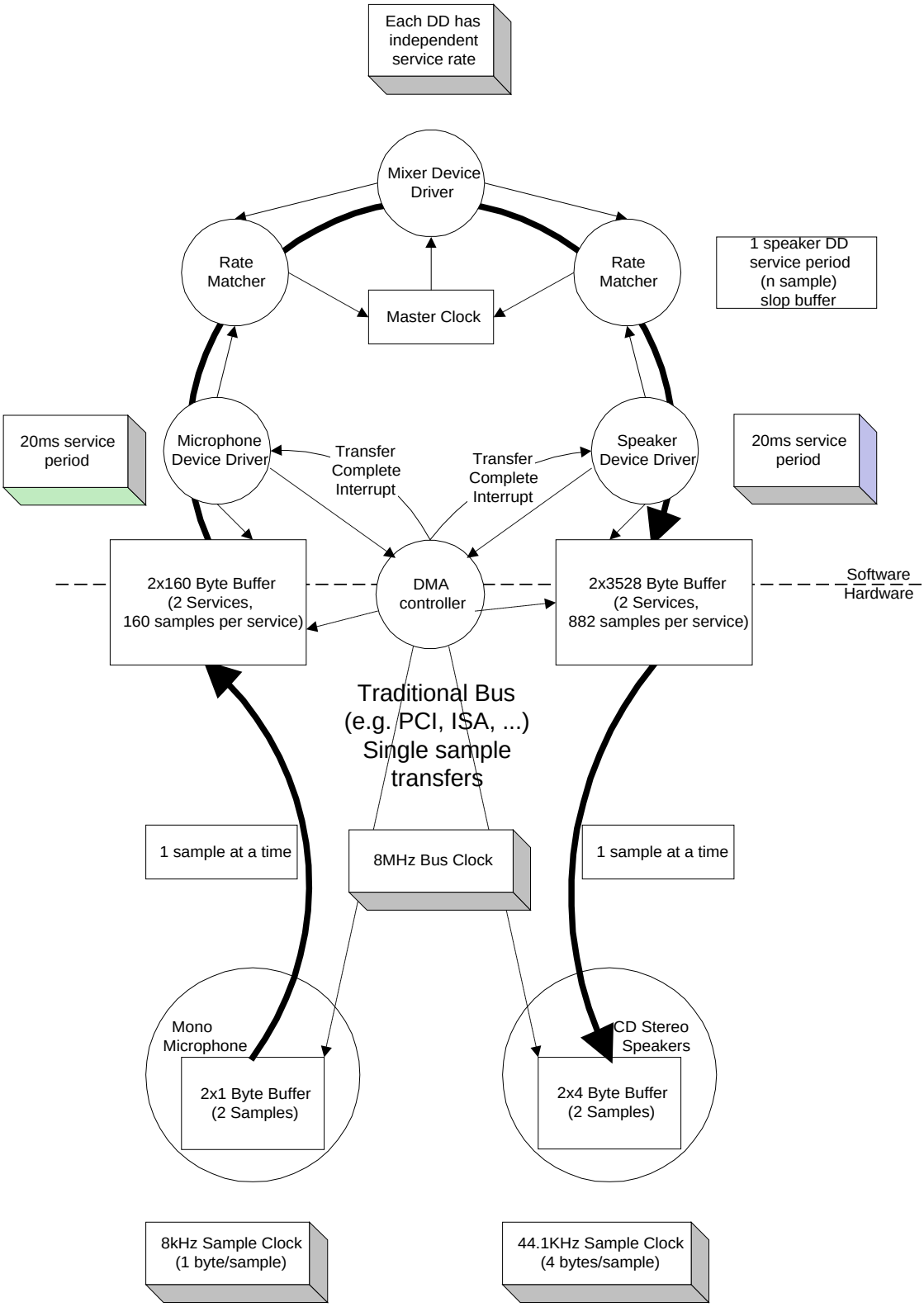


Figure 5-16. Non-USB Isochronous Example

Microchip Ex. 1018, p. 95
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

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

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

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

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

Microchip Ex. 1018, p. 96
Microchip v. Aptiv
IPR2024-00494

5.12.2 USB Clock Model

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

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

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

Microchip Ex. 1018, p. 97
Microchip v. Aptiv
IPR2024-00494

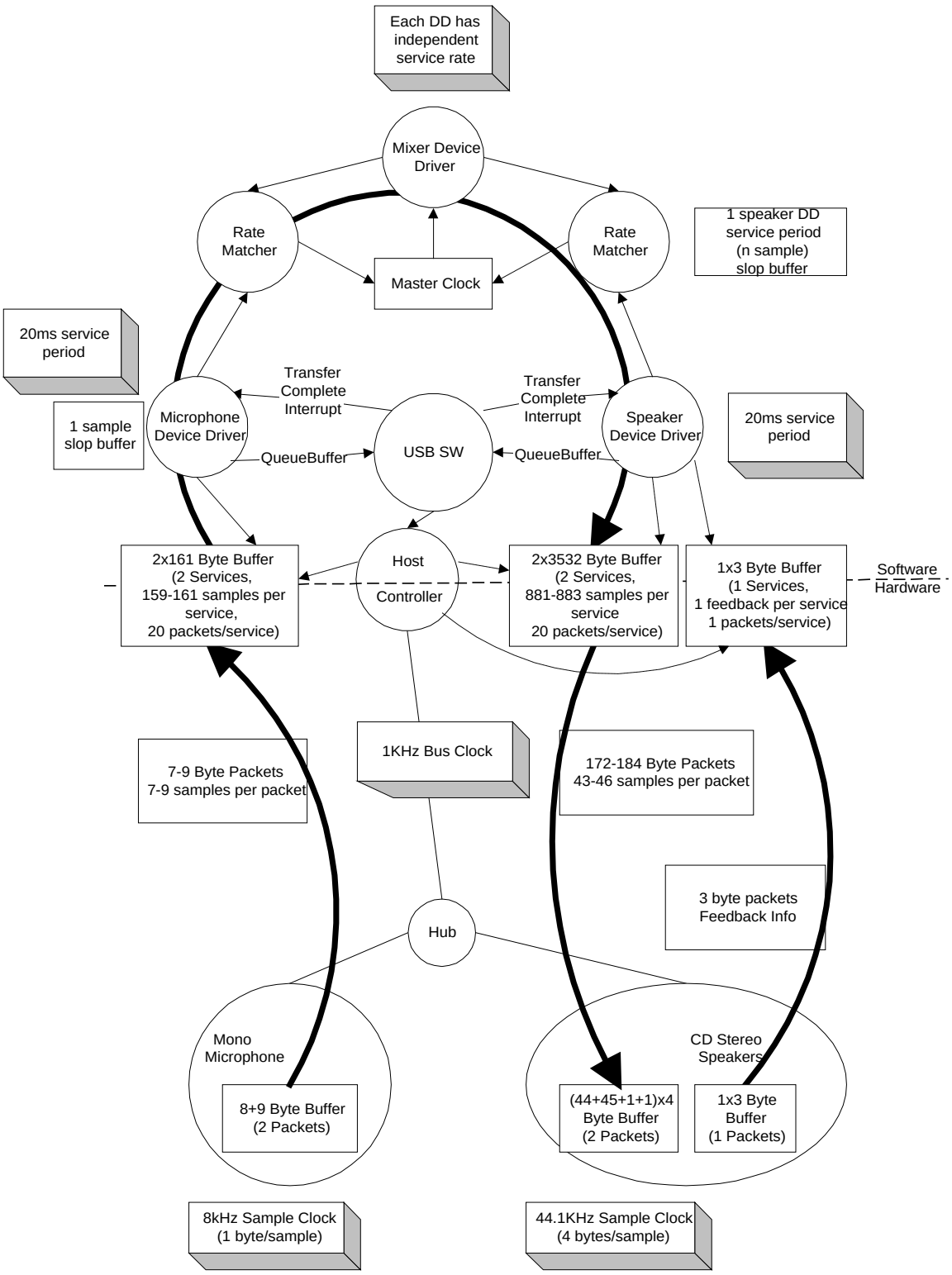


Figure 5-17. USB Full-speed Isochronous Application

Microchip Ex. 1018, p. 98
Microchip v. Aptiv
IPR2024-00494

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

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

5.12.3 Clock Synchronization

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

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

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

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

5.12.4 Isochronous Devices

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

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

Microchip Ex. 1018, p. 99
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

5.12.4.1 Synchronization Type

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

Table 5-12. Synchronization Characteristics

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

5.12.4.1.1 Asynchronous

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

Microchip Ex. 1018, p. 100
Microchip v. Aptiv
IPR2024-00494

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

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

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

5.12.4.1.2 Synchronous

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

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

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

5.12.4.1.3 Adaptive

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

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

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

5.12.4.2 Feedback

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

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

Microchip Ex. 1018, p. 101
Microchip v. Aptiv
IPR2024-00494

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

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

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

$$F_m = F_s * 2^P$$

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

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

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

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

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

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

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

Microchip Ex. 1018, p. 102
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

5.12.4.3 Implicit Feedback

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

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

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

Two cases can arise:

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

Microchip Ex. 1018, p. 103
Microchip v. Aptiv
IPR2024-00494

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

5.12.4.4 Connectivity

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

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

Microchip Ex. 1018, p. 104
Microchip v. Aptiv
IPR2024-00494

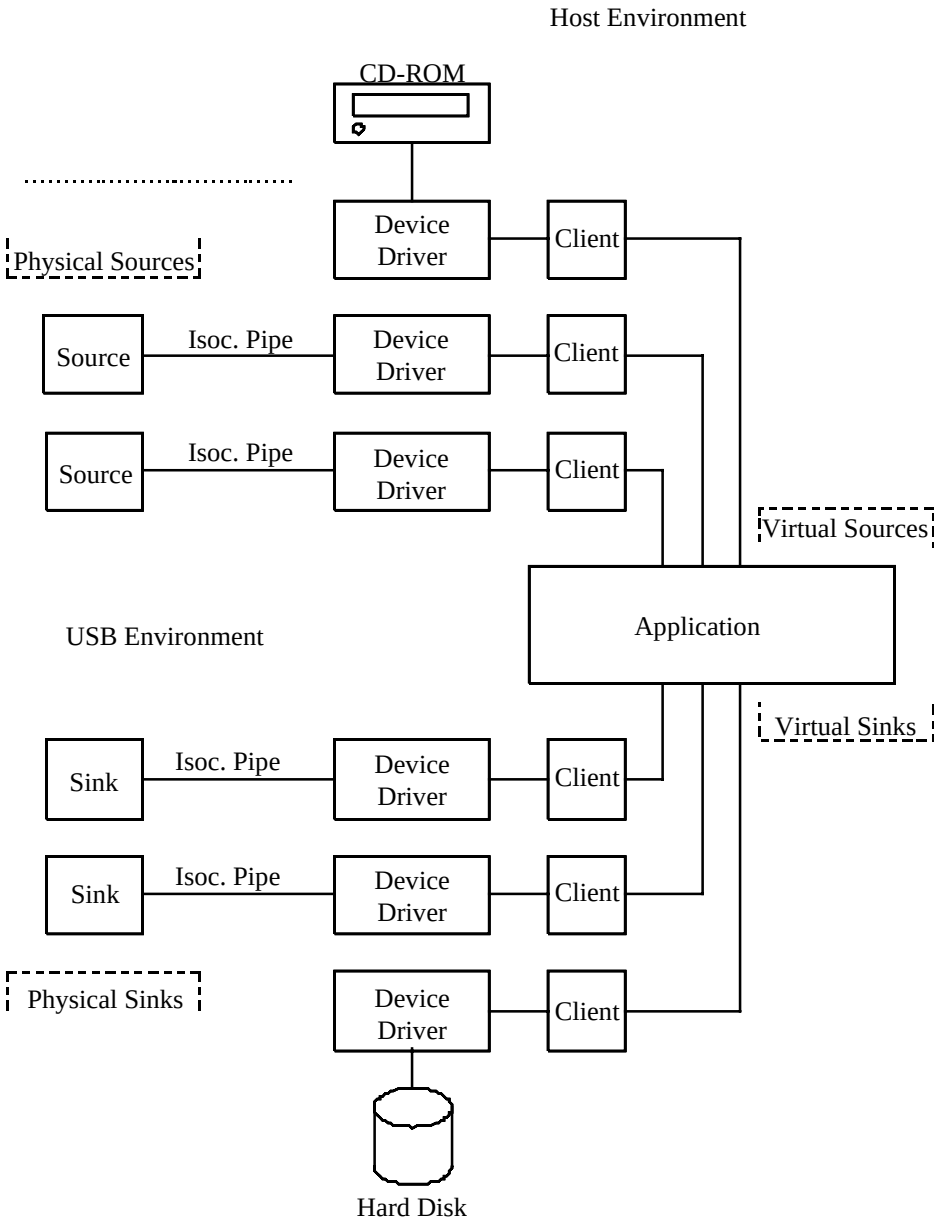


Figure 5-18. Example Source/Sink Connectivity

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

Microchip Ex. 1018, p. 105
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

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

Microchip Ex. 1018, p. 106
Microchip v. Aptiv
IPR2024-00494

Table 5-13. Connection Requirements

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

Notes:

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

Microchip Ex. 1018, p. 107
Microchip v. Aptiv
IPR2024-00494

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

5.12.4.4.1 Audio Connectivity

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

5.12.4.4.2 Synchronous Data Connectivity

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

5.12.5 Data Prebuffering

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

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

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

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

Microchip Ex. 1018, p. 108
Microchip v. Aptiv
IPR2024-00494

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

Time:	T_i	T_{i+1}	T_{i+2}	T_{i+3}	------	------	T_m	T_{m+1}	------
(Micro)Frame:	F_i	F_{i+1}	F_{i+2}	M_{i+3}	------	------	F_m	F_{m+1}	------
Data on Bus:	------	D_0	D_1	D_2	------	------	D_0	D_1	------
OUT Process:	------	------	D_0	D_1	------	------	------	D_0	------
IN Process:	D_0	D_1	------	------	------	D_0	------	------	------

Figure 5-19. Data Prebuffering

5.12.6 SOF Tracking

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

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

5.12.7 Error Handling

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

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

Microchip Ex. 1018, p. 109
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

5.12.8 Buffering for Rate Matching

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

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

Microchip Ex. 1018, p. 110
Microchip v. Aptiv
IPR2024-00494

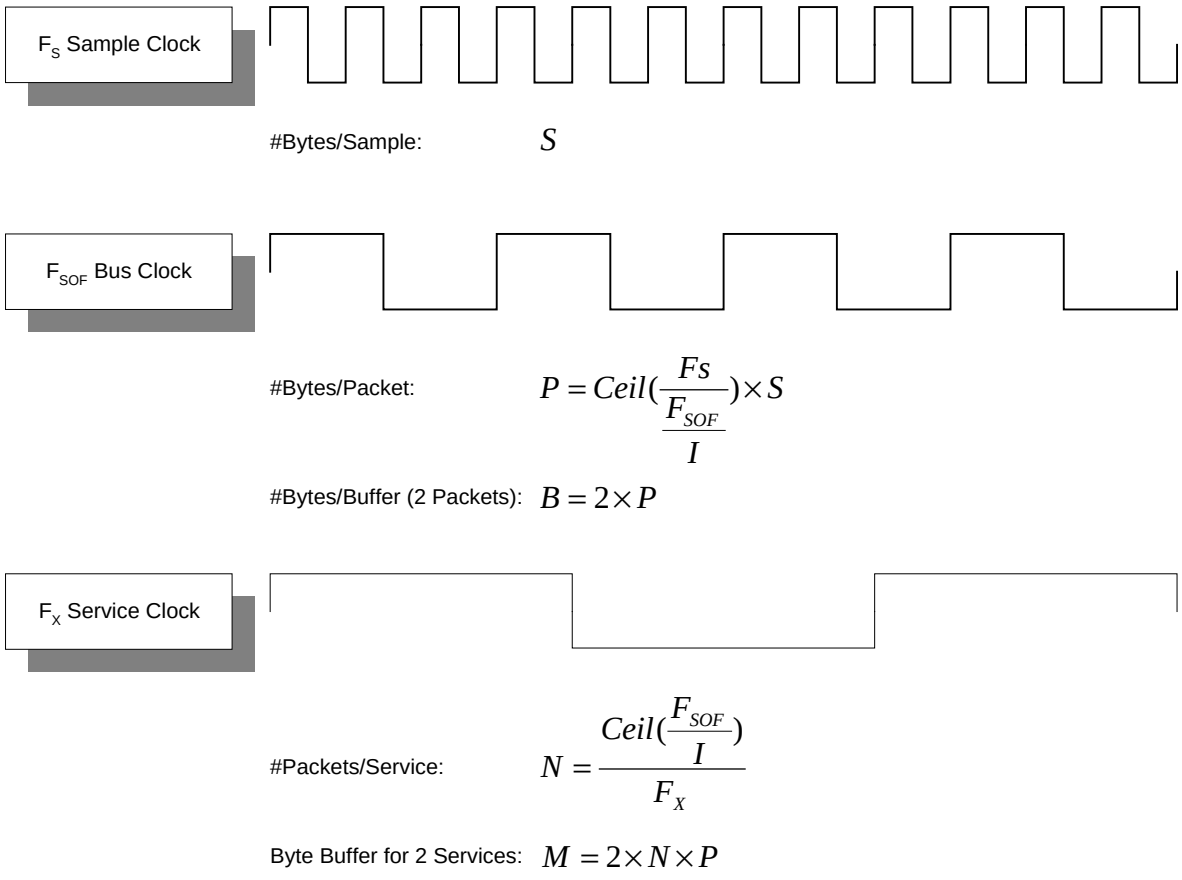


Figure 5-20. Packet and Buffer Size Formulas for Rate-matched Isochronous Transfers

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

Microchip Ex. 1018, p. 111
Microchip v. Aptiv
IPR2024-00494

Microchip Ex. 1018, p. 112
Microchip v. Aptiv
IPR2024-00494

Chapter 6
Mechanical

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

6.1 Architectural Overview

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

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

6.2 Keyed Connector Protocol

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

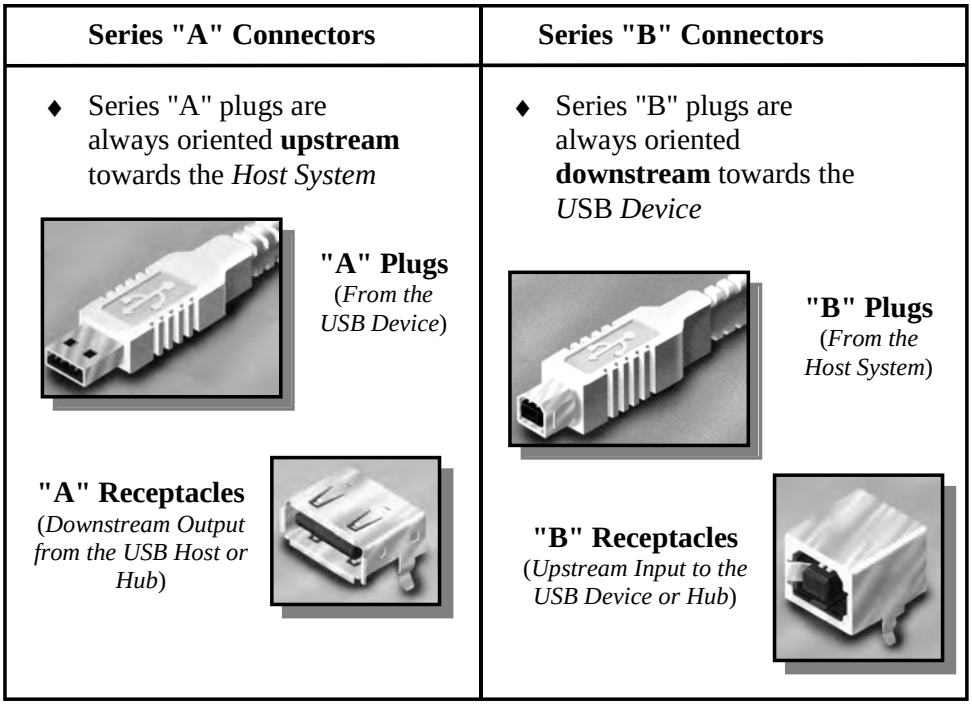


Figure 6-1. Keyed Connector Protocol

Microchip Ex. 1018, p. 113
Microchip v. Aptiv
IPR2024-00494

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

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

6.3 Cable

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

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

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

6.4 Cable Assembly

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

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

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

6.4.1 Standard Detachable Cable Assemblies

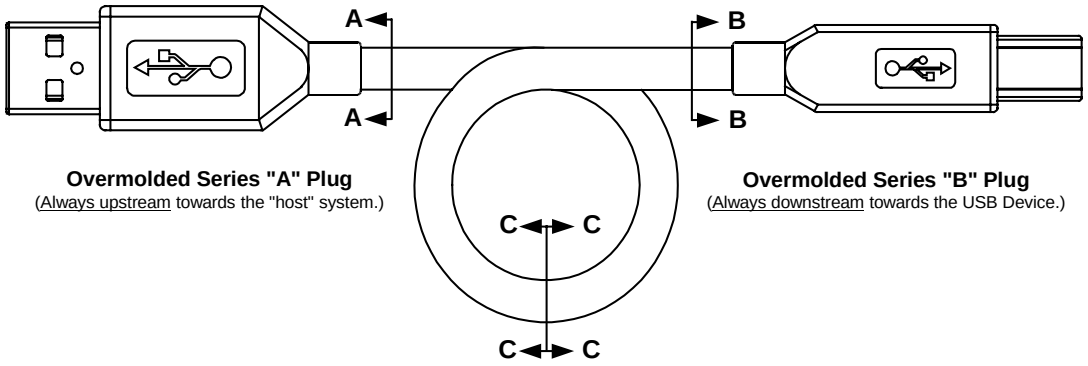
High-speed and full-speed devices can utilize the “B” connector. This allows the device to have a standard detachable USB cable. This eliminates the need to build the device with a hardwired cable and minimizes end user problems if cable replacement is necessary.

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

Figure 6-2 illustrates a standard detachable cable assembly.

Microchip Ex. 1018, p. 114
Microchip v. Aptiv
IPR2024-00494

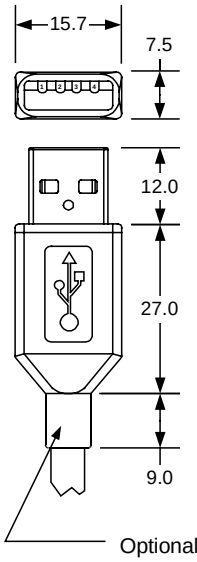
IMPORTANT NOTICE: All standard detachable cable assemblies must be high-/full-speed.



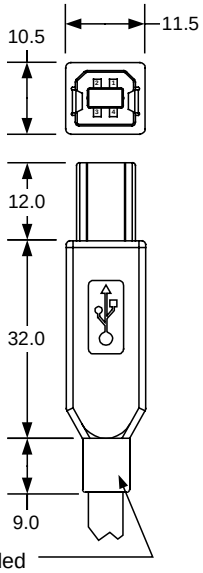
Overmolded Series "A" Plug
(Always upstream towards the "host" system.)

Overmolded Series "B" Plug
(Always downstream towards the USB Device.)

Detail A - A
(Series "A" Plug)

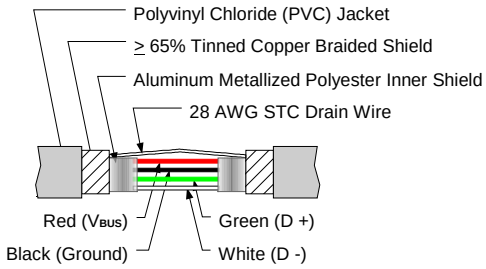


Detail B - B
(Series "B" Plug)



Optional Molded
Strain Relief

Detail C - C
(Typical USB Shielded Cable)



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

Dimensions are **TYPICAL** and are for general reference purposes only.

	Series "A" Plug to Series "B" Plug
	USB Standard Detachable Cable Assembly
SIZE **A** / DATE 2/98 / DRAWING NUMBER **N/A** / REV **C**	
SCALE: **N/A**	SHEET **1 of 1**

Figure 6-2. USB Standard Detachable Cable Assembly

Microchip Ex. 1018, p. 115
Microchip v. Aptiv
IPR2024-00494

Standard detachable cable assemblies must meet the following electrical requirements:

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

6.4.2 High-/full-speed Captive Cable Assemblies

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

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

Microchip Ex. 1018, p. 116
Microchip v. Aptiv
IPR2024-00494

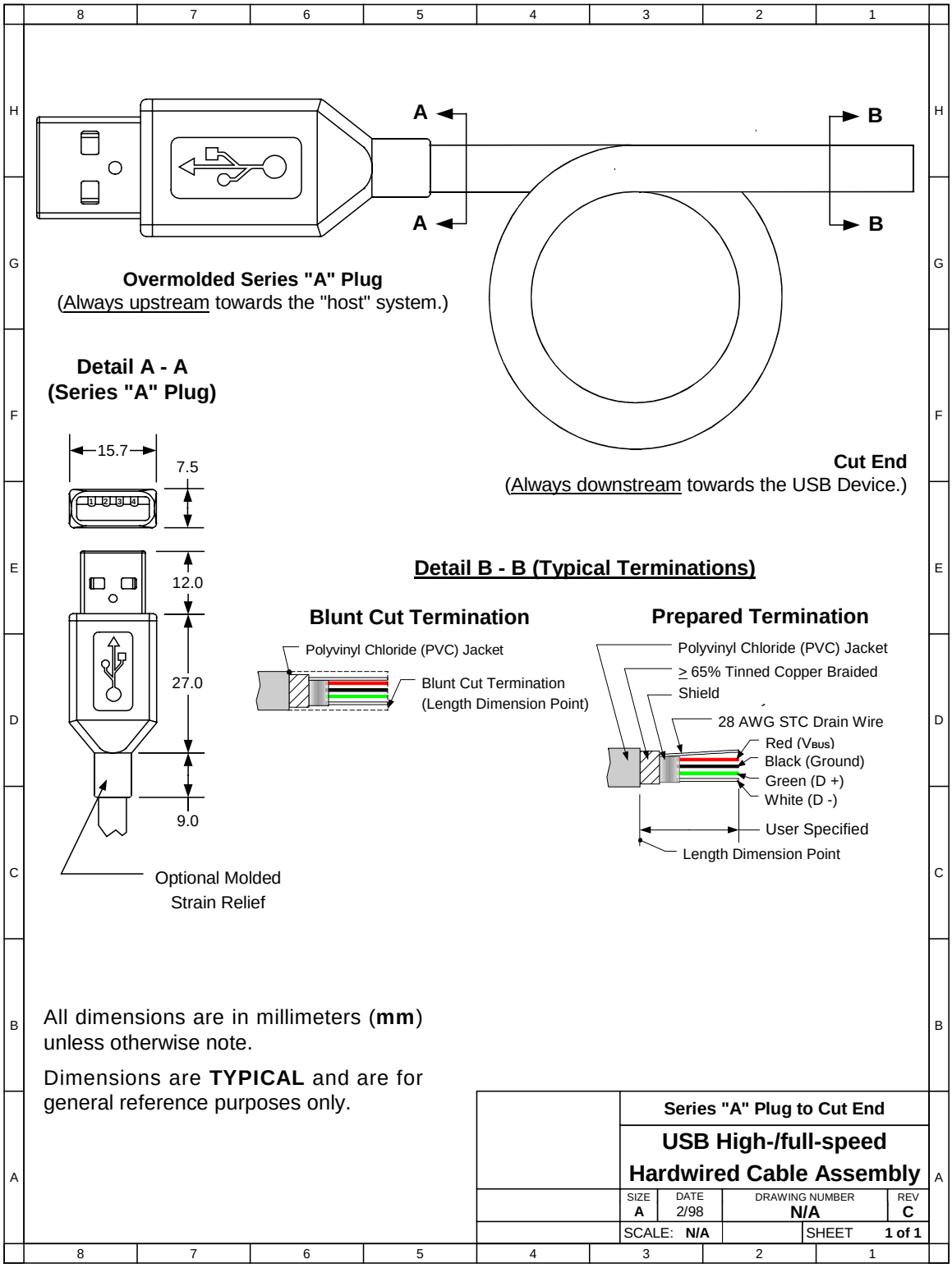


Figure 6-3. USB High-/full-speed Hardwired Cable Assembly

Microchip Ex. 1018, p. 117
Microchip v. Aptiv
IPR2024-00494

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

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

6.4.3 Low-speed Captive Cable Assemblies

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

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

Microchip Ex. 1018, p. 118
Microchip v. Aptiv
IPR2024-00494

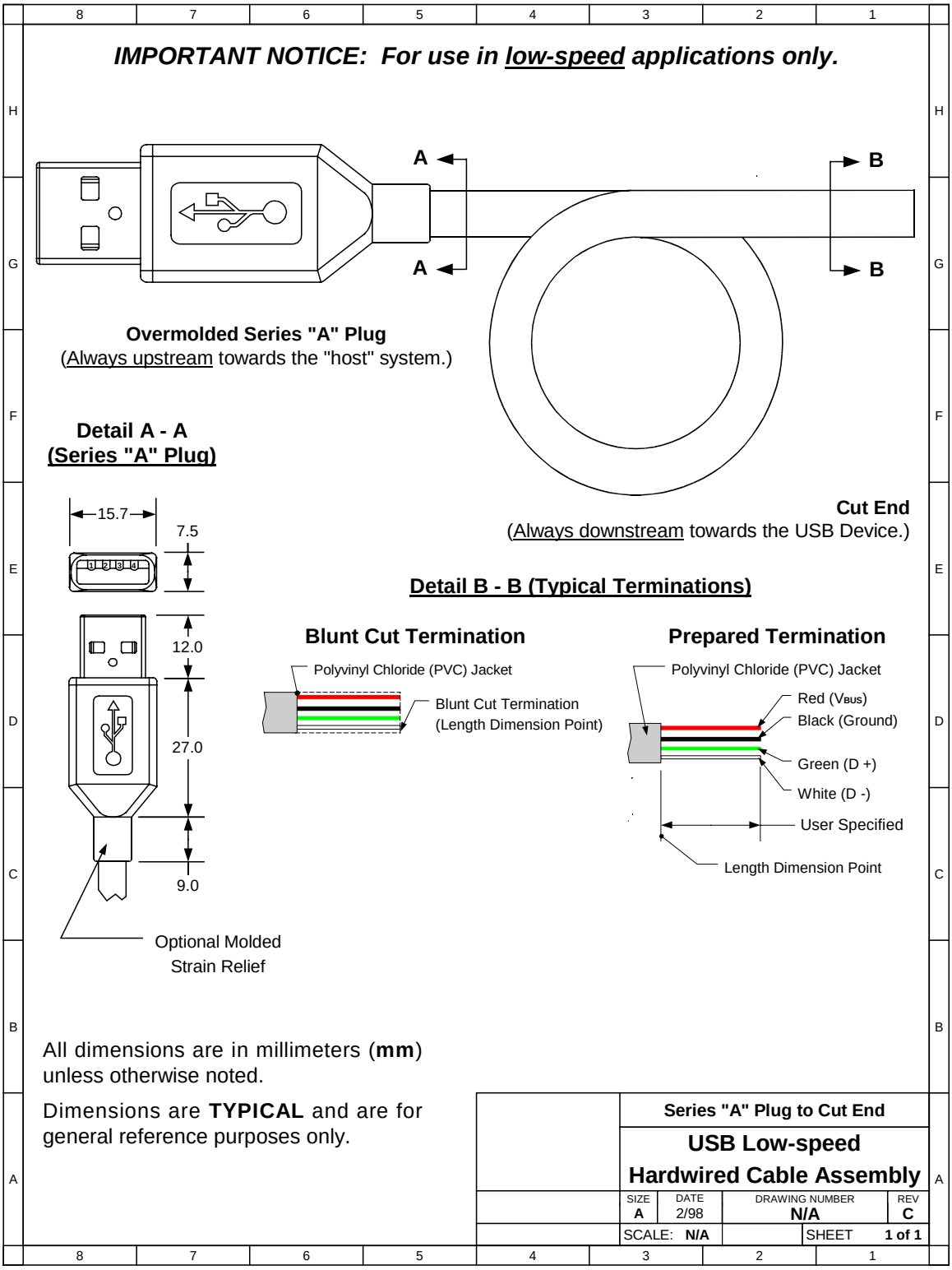


Figure 6-4. USB Low-speed Hardwired Cable Assembly

Microchip Ex. 1018, p. 119
Microchip v. Aptiv
IPR2024-00494

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

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

6.4.4 Prohibited Cable Assemblies

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

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

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

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

Microchip Ex. 1018, p. 120
Microchip v. Aptiv
IPR2024-00494

6.5 Connector Mechanical Configuration and Material Requirements

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

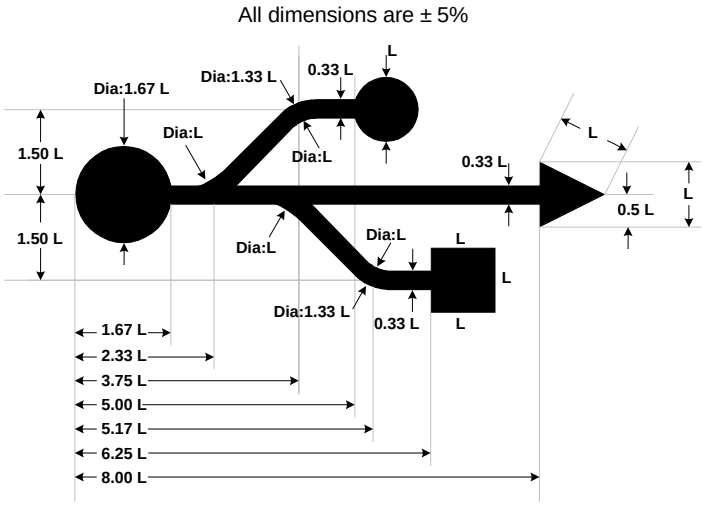


Figure 6-5. USB Icon

6.5.1 USB Icon Location

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

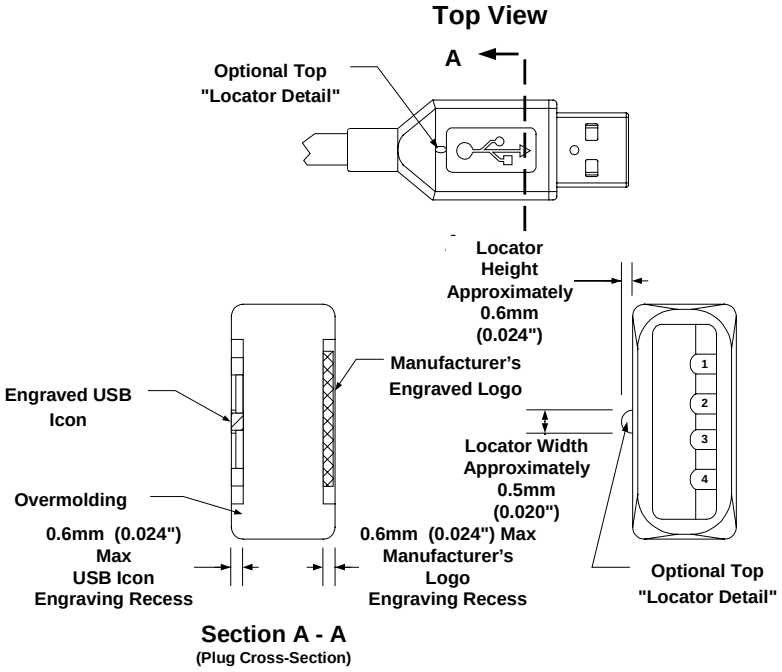


Figure 6-6. Typical USB Plug Orientation

Microchip Ex. 1018, p. 121
Microchip v. Aptiv
IPR2024-00494

6.5.2 USB Connector Termination Data

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

Table 6-1. USB Connector Termination Assignment

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

6.5.3 Series “A” and Series “B” Receptacles

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

Microchip Ex. 1018, p. 122
Microchip v. Aptiv
IPR2024-00494

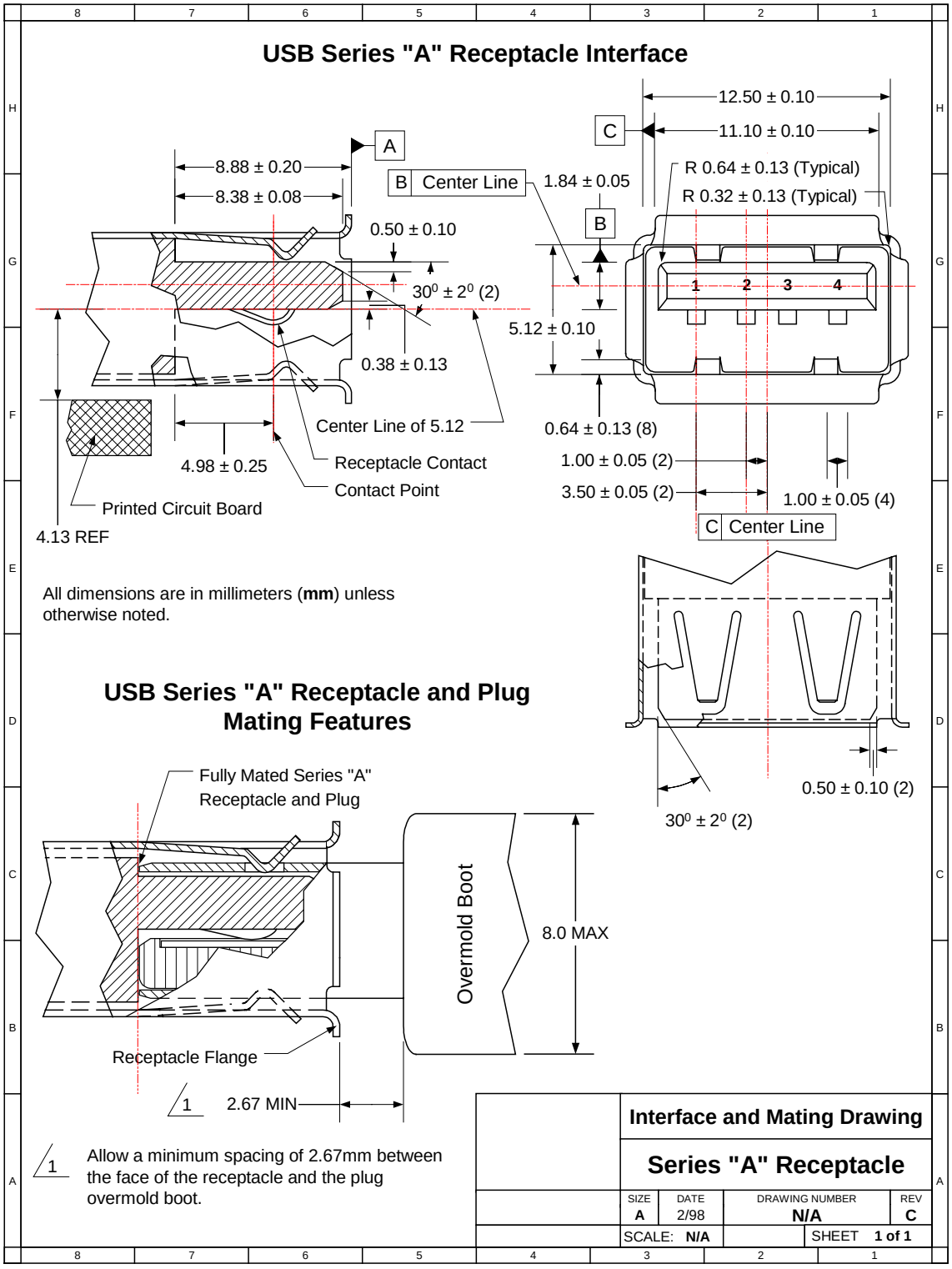


Figure 6-7. USB Series "A" Receptacle Interface and Mating Drawing

Microchip Ex. 1018, p. 123
Microchip v. Aptiv
IPR2024-00494

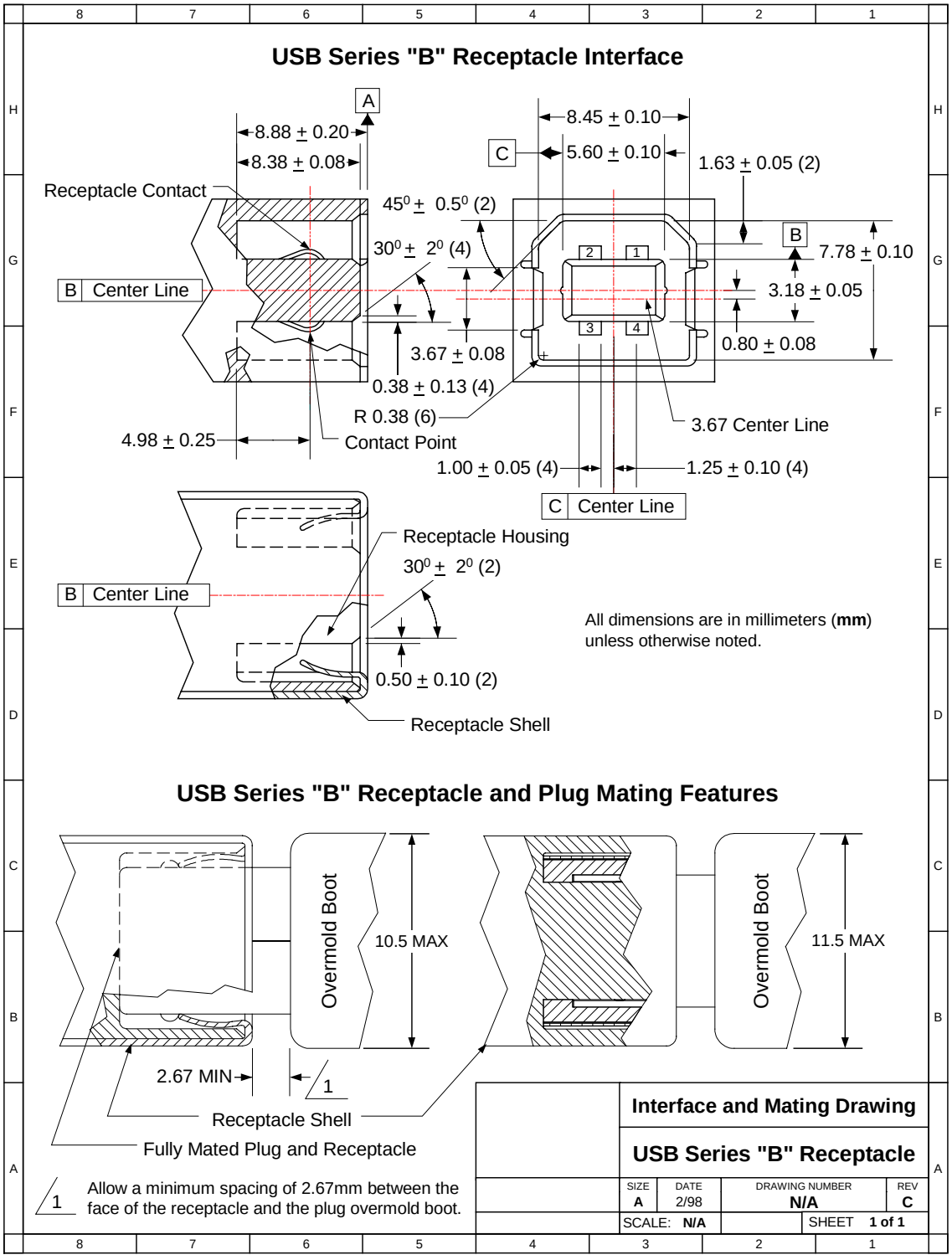


Figure 6-8. USB Series "B" Receptacle Interface and Mating Drawing

Microchip Ex. 1018, p. 124
Microchip v. Aptiv
IPR2024-00494

6.5.3.1 Receptacle Injection Molded Thermoplastic Insulator Material

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

Typical Colors: Black, gray, and natural.

Flammability Characteristics: UL 94-V0 rated.

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

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

6.5.3.2 Receptacle Shell Materials

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

Plating:

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

6.5.3.3 Receptacle Contact Materials

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

Plating: Contacts are to be selectively plated.

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

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

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

Microchip Ex. 1018, p. 125
Microchip v. Aptiv
IPR2024-00494

6.5.4 Series “A” and Series “B” Plugs

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

Microchip Ex. 1018, p. 126
Microchip v. Aptiv
IPR2024-00494

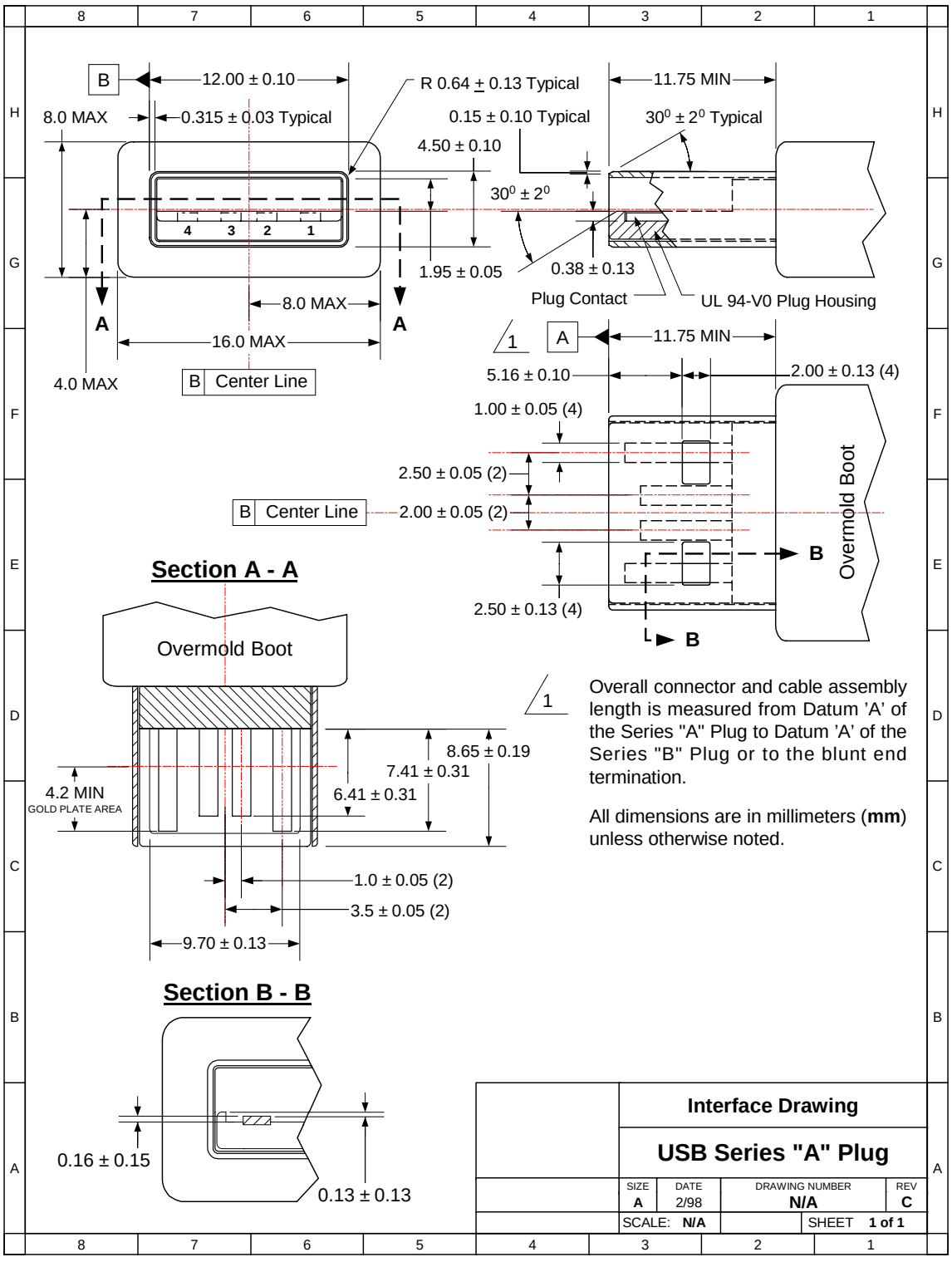


Figure 6-9. USB Series "A" Plug Interface Drawing

Microchip Ex. 1018, p. 127
Microchip v. Aptiv
IPR2024-00494

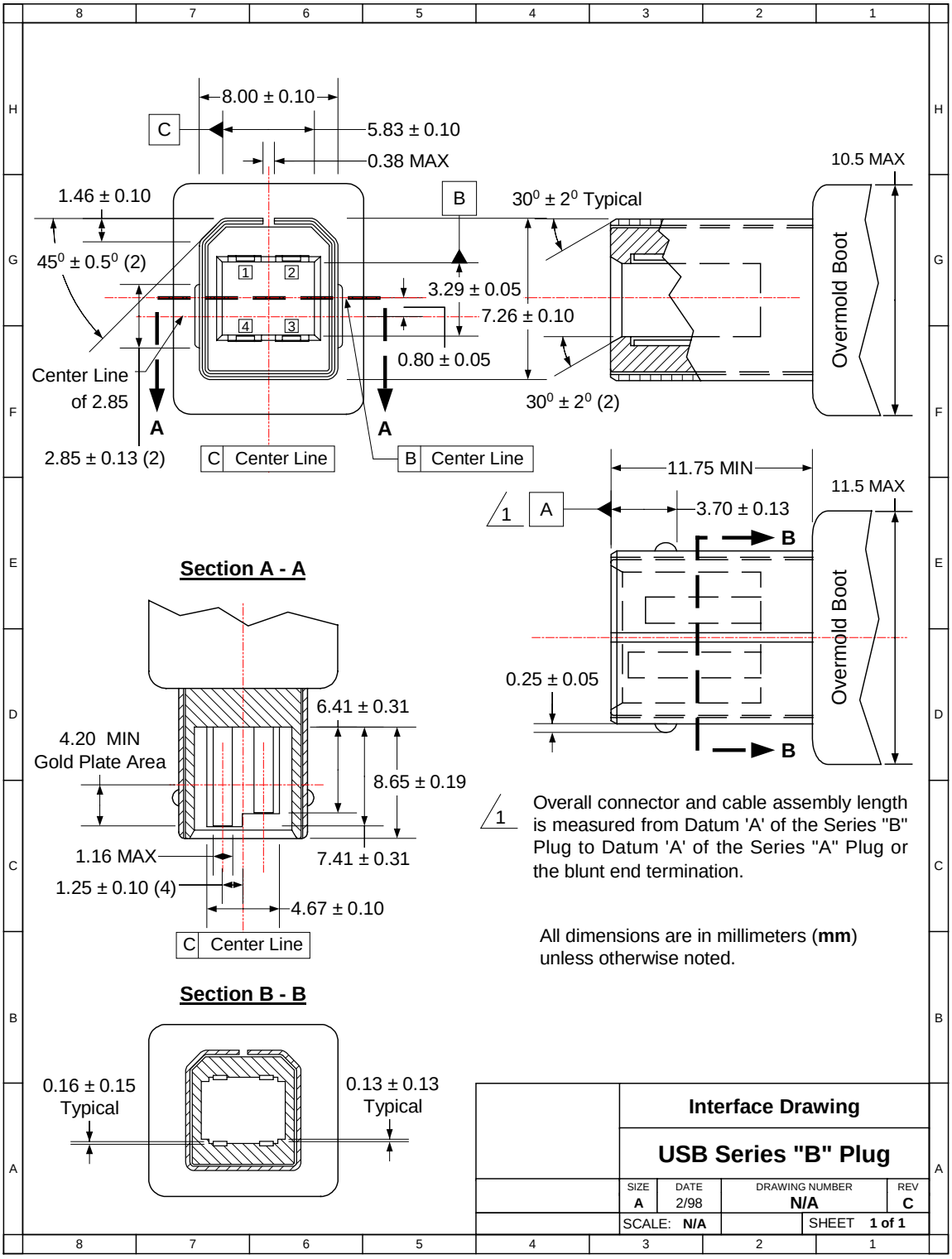


Figure 6-10. USB Series "B" Plug Interface Drawing

Microchip Ex. 1018, p. 128
Microchip v. Aptiv
IPR2024-00494

6.5.4.1 Plug Injection Molded Thermoplastic Insulator Material

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

Typical Colors: Black, gray, and natural.

Flammability Characteristics: UL 94-V0 rated.

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

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

6.5.4.2 Plug Shell Materials

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

Plating:

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

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

6.5.4.3 Plug (Male) Contact Materials

Substrate Material: 0.30 $\pm$ 0.05 mm half-hard phosphor bronze.

Plating: Contacts are to be selectively plated.

A. Option I

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

B. Option II

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

C. Option III

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

Microchip Ex. 1018, p. 129
Microchip v. Aptiv
IPR2024-00494

6.6 Cable Mechanical Configuration and Material Requirements

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

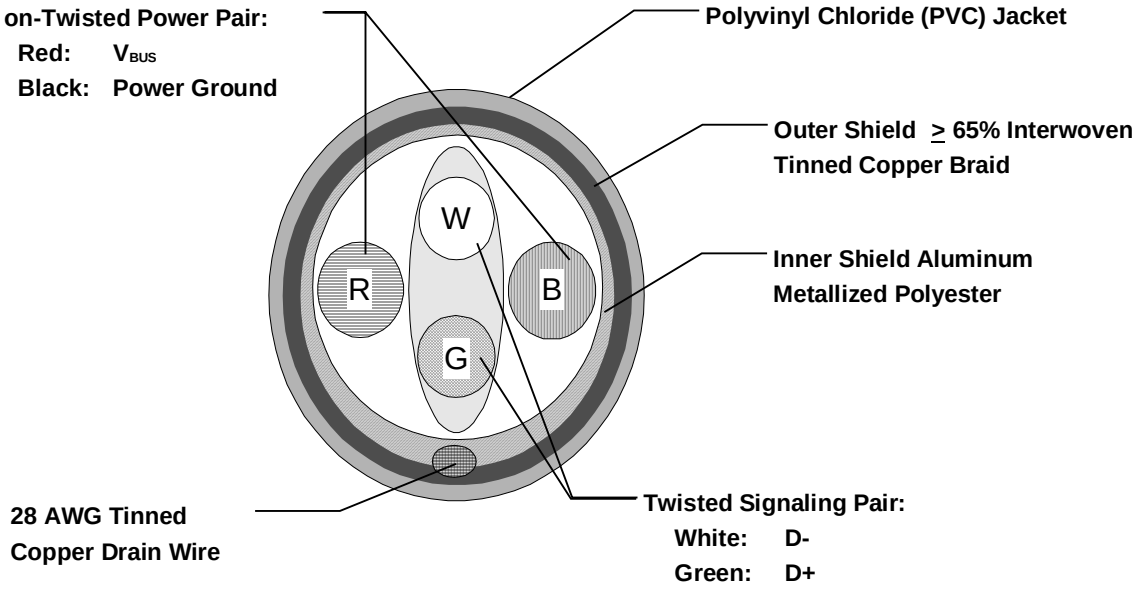


Figure 6-11. Typical High-/full-speed Cable Construction

6.6.1 Description

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

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

Microchip Ex. 1018, p. 130
Microchip v. Aptiv
IPR2024-00494

6.6.2 Construction

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

Table 6-2. Power Pair

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

Note: Minimum conductor construction must be stranded tinned copper.

Non-Twisted Power Pair:

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

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

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

Signal Pair:

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

Microchip Ex. 1018, p. 131
Microchip v. Aptiv
IPR2024-00494

Table 6-3. Signal Pair

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

Note: Minimum conductor construction must be stranded tinned copper.

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

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

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

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

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

B. Metallizing: Vacuum deposited aluminum

C. Assembly:

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

Drain Wire (required for low-speed):

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

Table 6-4. Drain Wire Signal Pair

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

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

A. Coverage Area: Minimum 65%.

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

Outer Polyvinyl Chloride (PVC) Jacket:

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

Microchip Ex. 1018, p. 132
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

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

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

Nominal Fabricated Cable Outer Diameter:

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

Table 6-5. Nominal Cable Diameter

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

6.6.3 Electrical Characteristics

All electrical characteristics must be measured at or referenced to +20 ºC (68 ºF).

Voltage Rating: 30 V rms maximum.

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

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

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

Table 6-6. Conductor Resistance

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

Microchip Ex. 1018, p. 133
Microchip v. Aptiv
IPR2024-00494

6.6.4 Cable Environmental Characteristics

Temperature Range:

A. Operating Temperature Range: 0 ºC to +50 ºC

B. Storage Temperature Range: -20 ºC to +60 ºC

C. Nominal Temperature Rating: +20 ºC

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

6.6.5 Listing

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

6.7 Electrical, Mechanical, and Environmental Compliance Standards

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

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

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

Microchip Ex. 1018, p. 134
Microchip v. Aptiv
IPR2024-00494

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

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

Microchip Ex. 1018, p. 135
Microchip v. Aptiv
IPR2024-00494

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

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

Microchip Ex. 1018, p. 136
Microchip v. Aptiv
IPR2024-00494

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

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

Microchip Ex. 1018, p. 137
Microchip v. Aptiv
IPR2024-00494

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

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

Microchip Ex. 1018, p. 138
Microchip v. Aptiv
IPR2024-00494

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

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

Microchip Ex. 1018, p. 139
Microchip v. Aptiv
IPR2024-00494

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

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

Microchip Ex. 1018, p. 140
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

Microchip Ex. 1018, p. 141
Microchip v. Aptiv
IPR2024-00494

6.7.1 Applicable Documents

American National Standard/Electronic Industries Association

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

American Standard Test Materials

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

Underwriters' Laboratory, Inc.

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

6.8 USB Grounding

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

6.9 PCB Reference Drawings

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

Microchip Ex. 1018, p. 142
Microchip v. Aptiv
IPR2024-00494

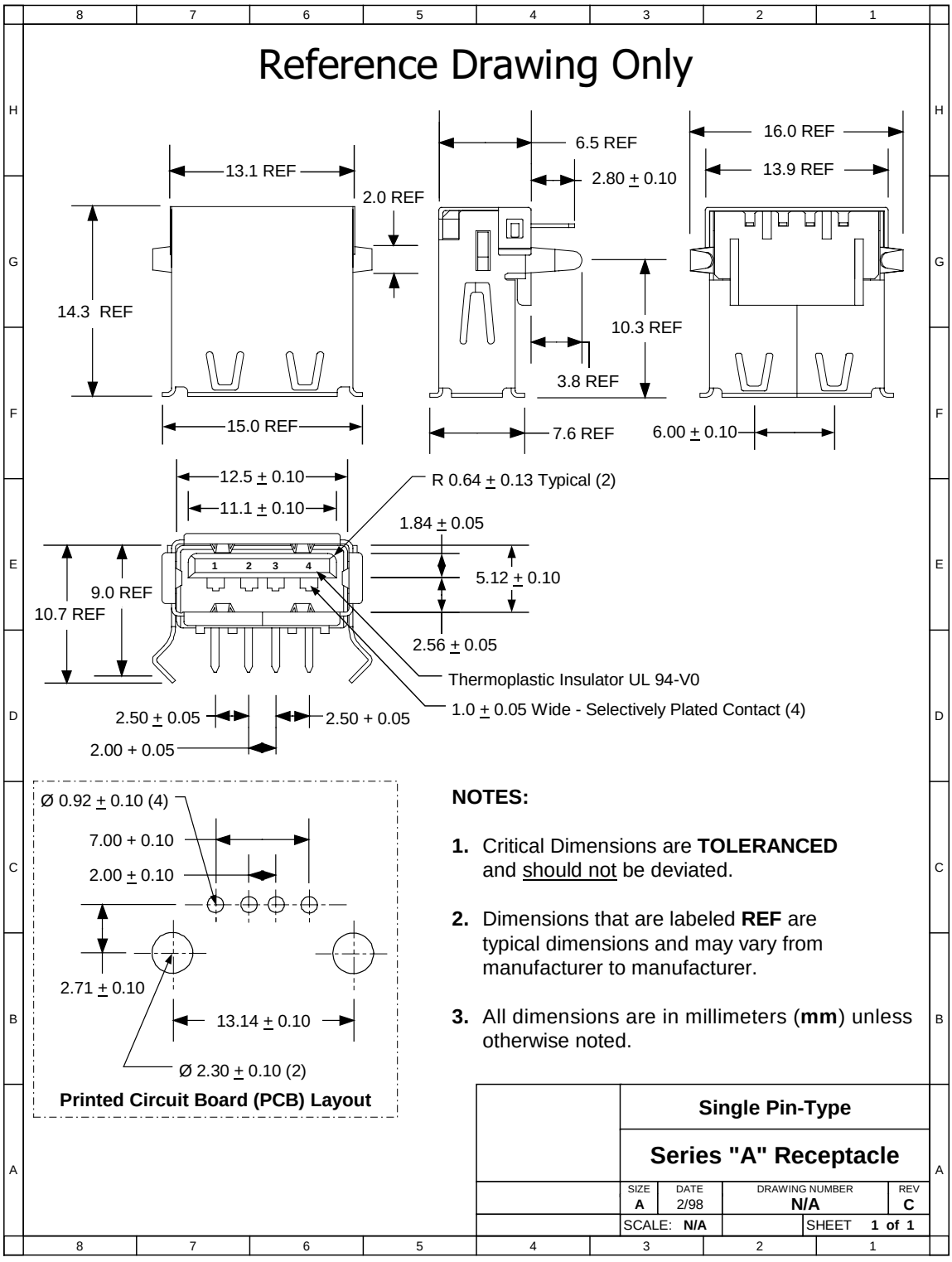


Figure 6-12. Single Pin-type Series "A" Receptacle

Microchip Ex. 1018, p. 143
Microchip v. Aptiv
IPR2024-00494

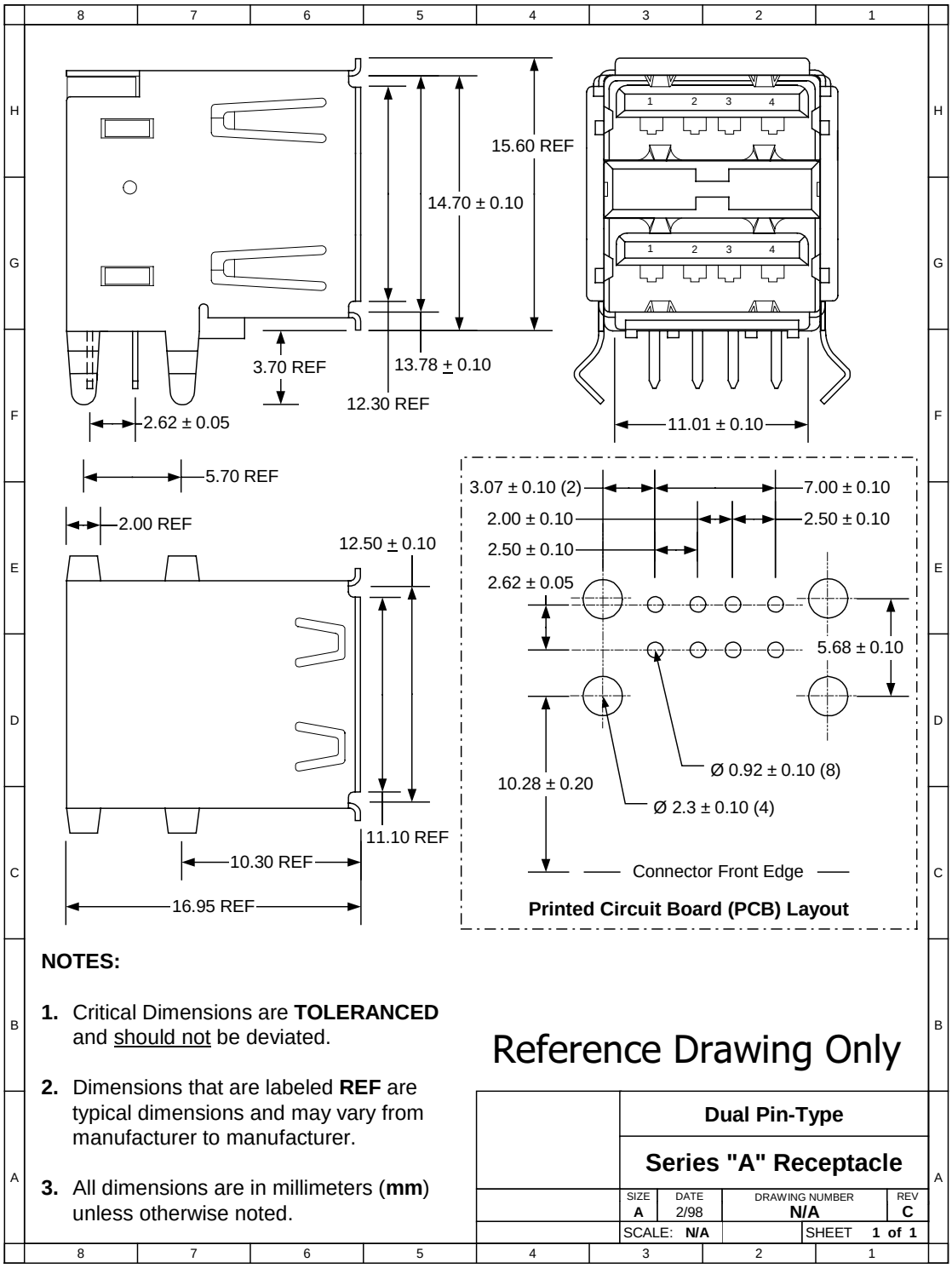


Figure 6-13. Dual Pin-type Series "A" Receptacle

Microchip Ex. 1018, p. 144
Microchip v. Aptiv
IPR2024-00494

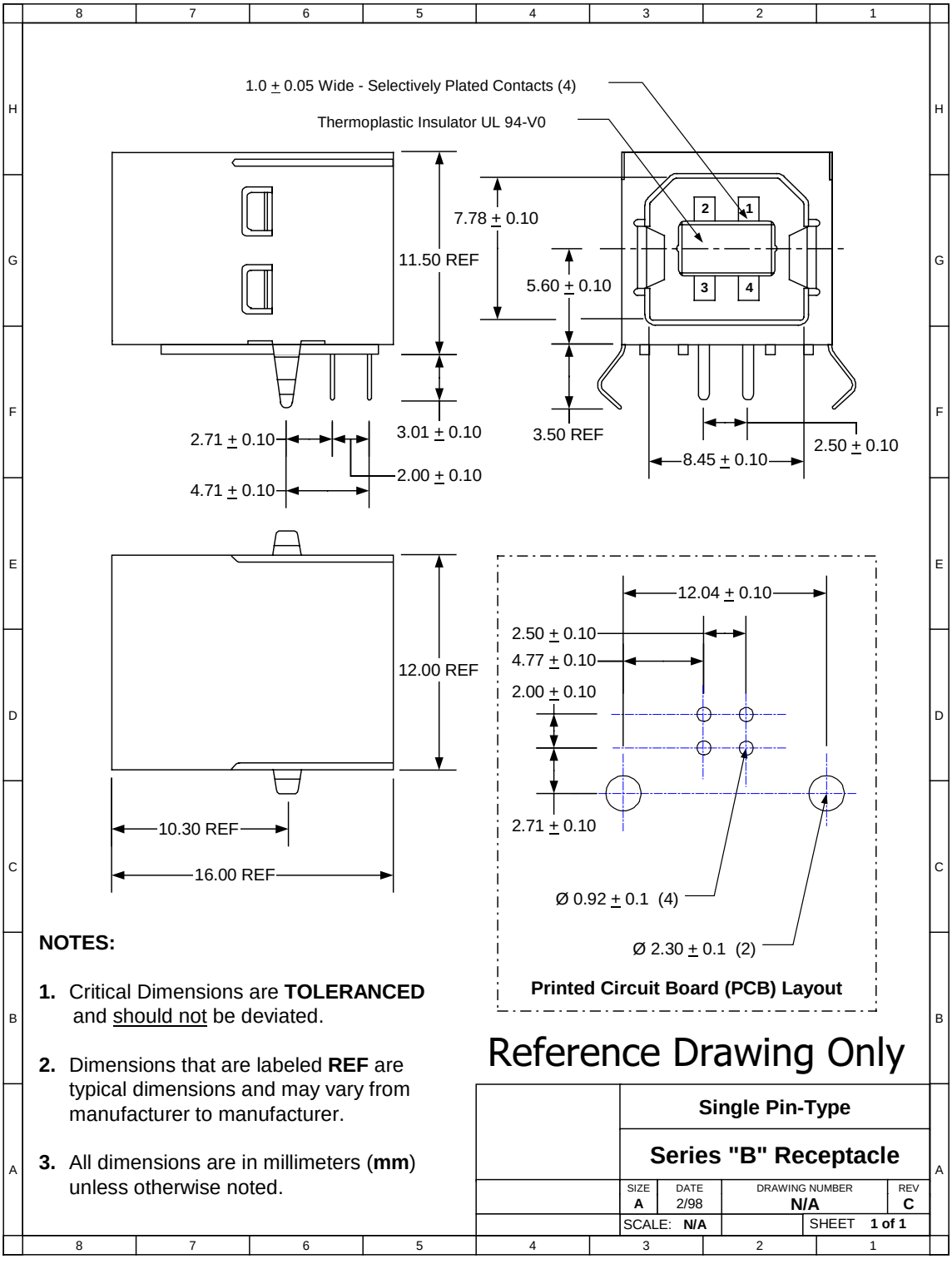


Figure 6-14. Single Pin-type Series "B" Receptacle

Microchip Ex. 1018, p. 145
Microchip v. Aptiv
IPR2024-00494

Microchip Ex. 1018, p. 146
Microchip v. Aptiv
IPR2024-00494

Chapter 7
Electrical

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

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

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

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

7.1 Signaling

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

Overview of High-speed Signaling

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

Microchip Ex. 1018, p. 147
Microchip v. Aptiv
IPR2024-00494

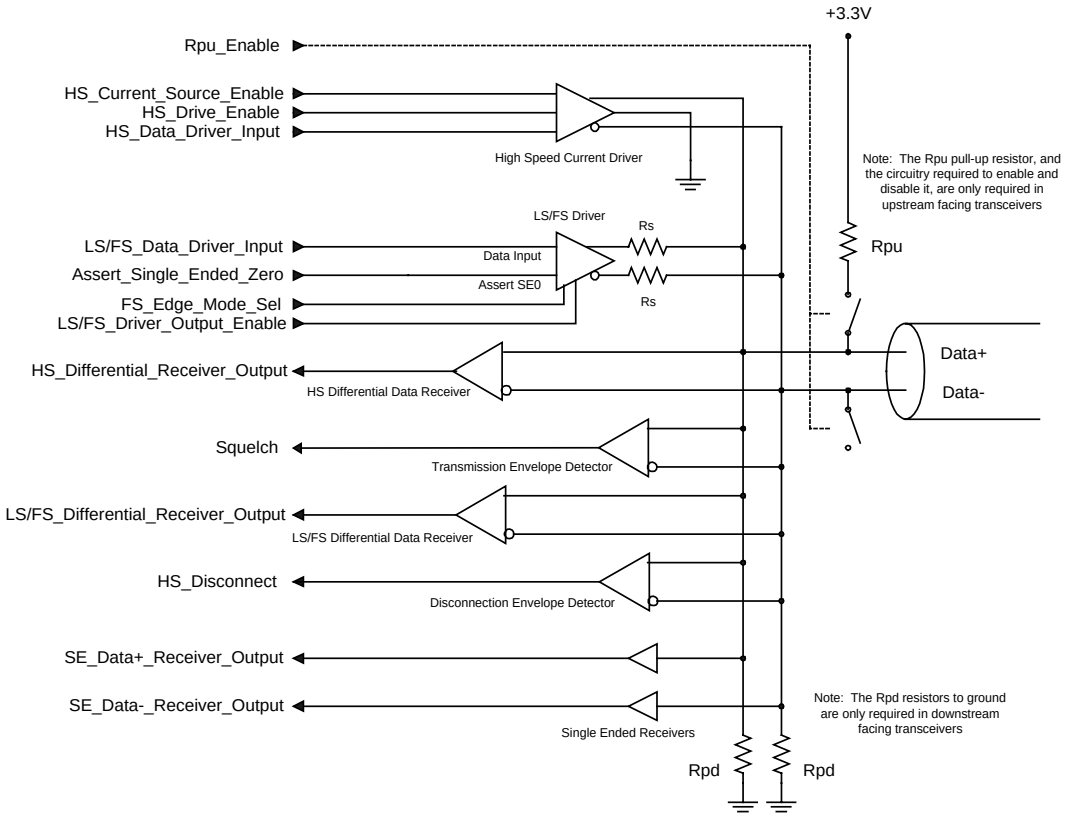


Figure 7-1. Example High-speed Capable Transceiver Circuit

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

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

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

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

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

Microchip Ex. 1018, p. 148
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

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

Microchip Ex. 1018, p. 149
Microchip v. Aptiv
IPR2024-00494

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

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

Microchip Ex. 1018, p. 150
Microchip v. Aptiv
IPR2024-00494

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

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

7.1.1 USB Driver Characteristics

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

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

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

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

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

Microchip Ex. 1018, p. 151
Microchip v. Aptiv
IPR2024-00494

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

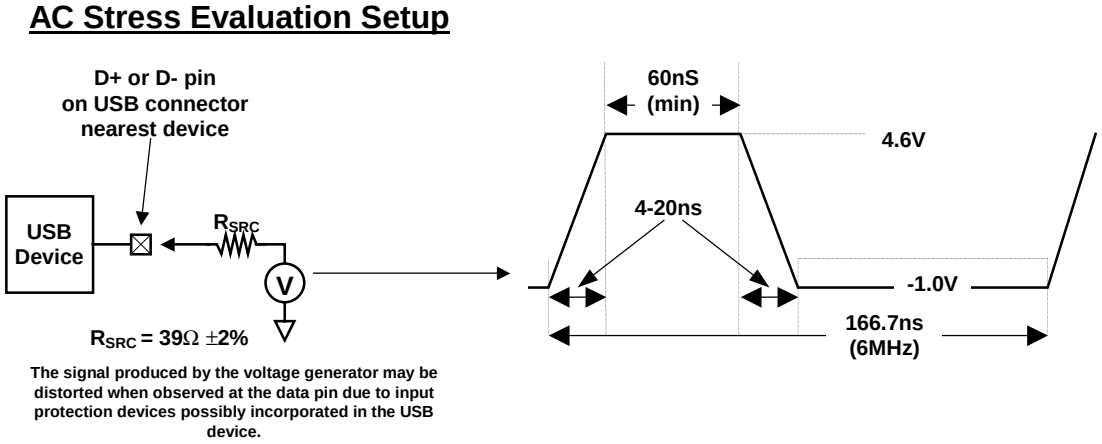


Figure 7-2. Maximum Input Waveforms for USB Signaling

Short Circuit Withstand

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

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

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

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

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

Microchip Ex. 1018, p. 152
Microchip v. Aptiv
IPR2024-00494

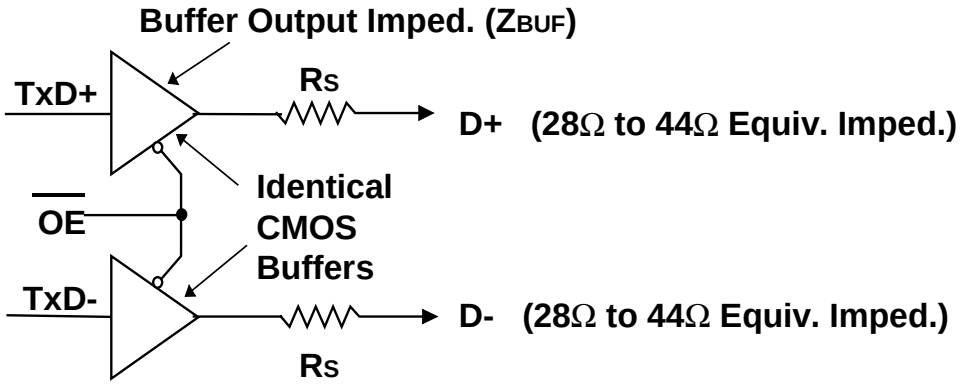


Figure 7-3. Example Full-speed CMOS Driver Circuit (non High-speed capable)

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

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

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

Full-speed Buffers in High-speed Capable Transceivers

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

Microchip Ex. 1018, p. 153
Microchip v. Aptiv
IPR2024-00494

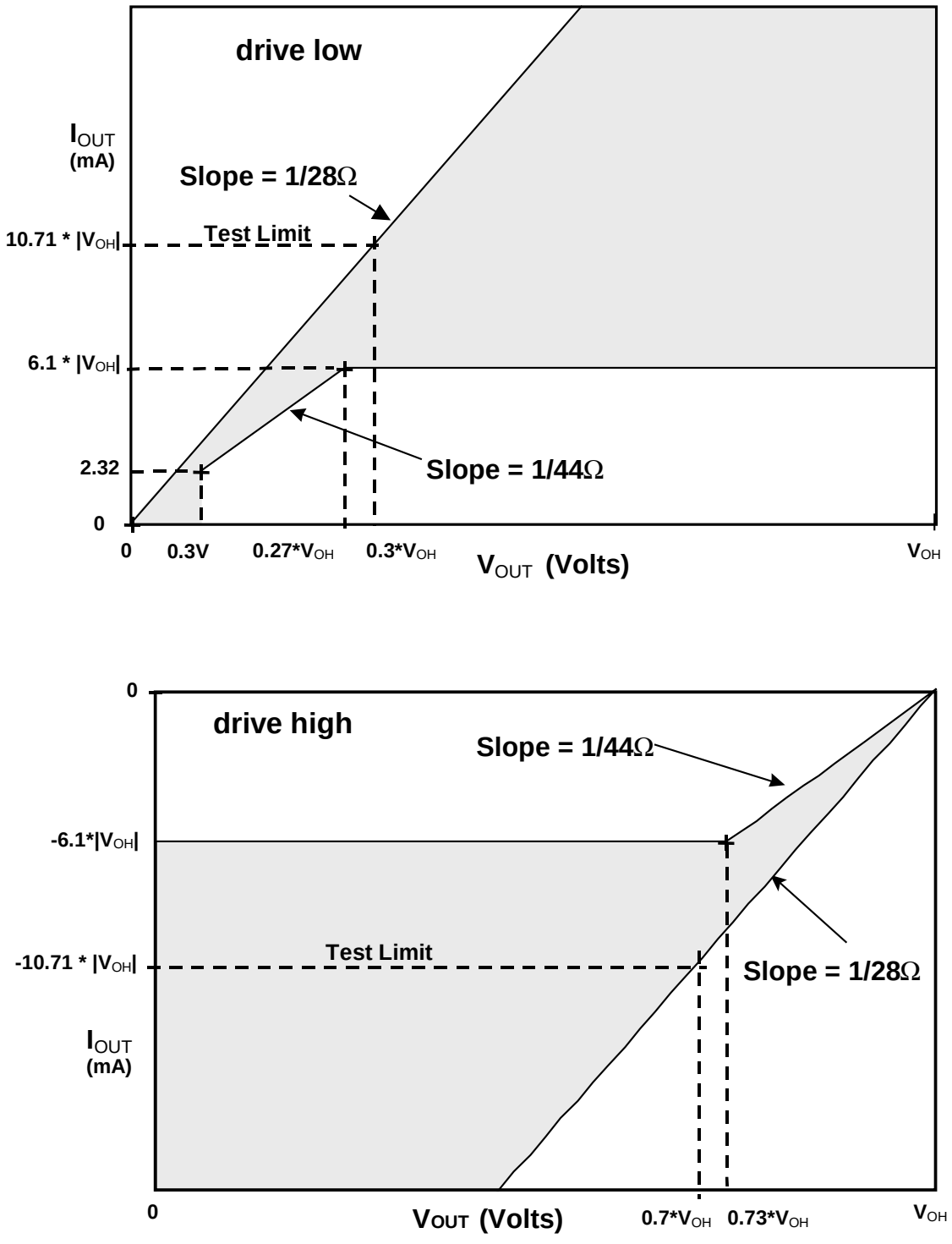


Figure 7-4. Full-speed Buffer V/I Characteristics

Microchip Ex. 1018, p. 154
Microchip v. Aptiv
IPR2024-00494

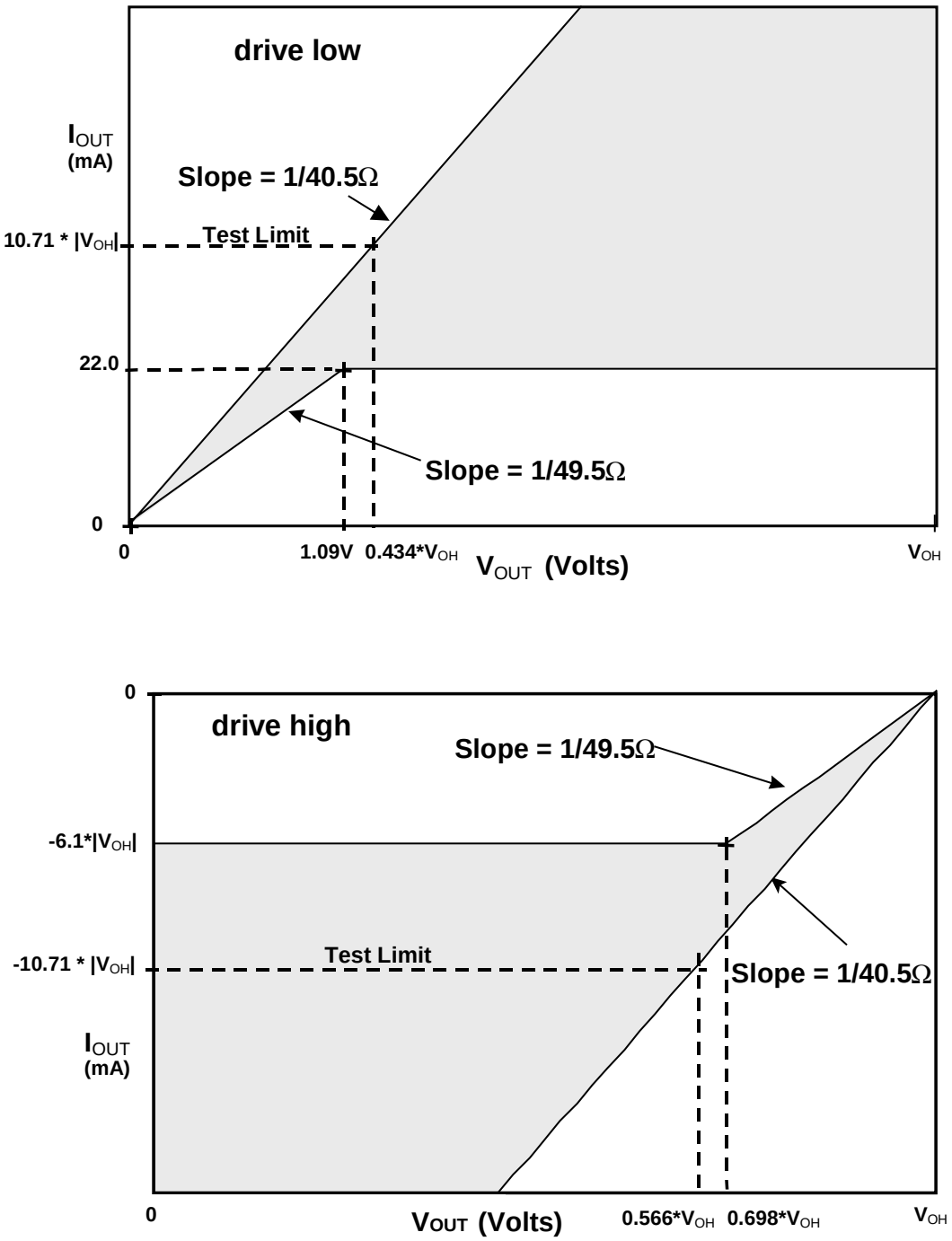


Figure 7-5. Full-speed Buffer V/I Characteristics for High-speed Capable Transceiver

Microchip Ex. 1018, p. 155
Microchip v. Aptiv
IPR2024-00494

Figure 7-6 shows the full-speed driver signal waveforms.

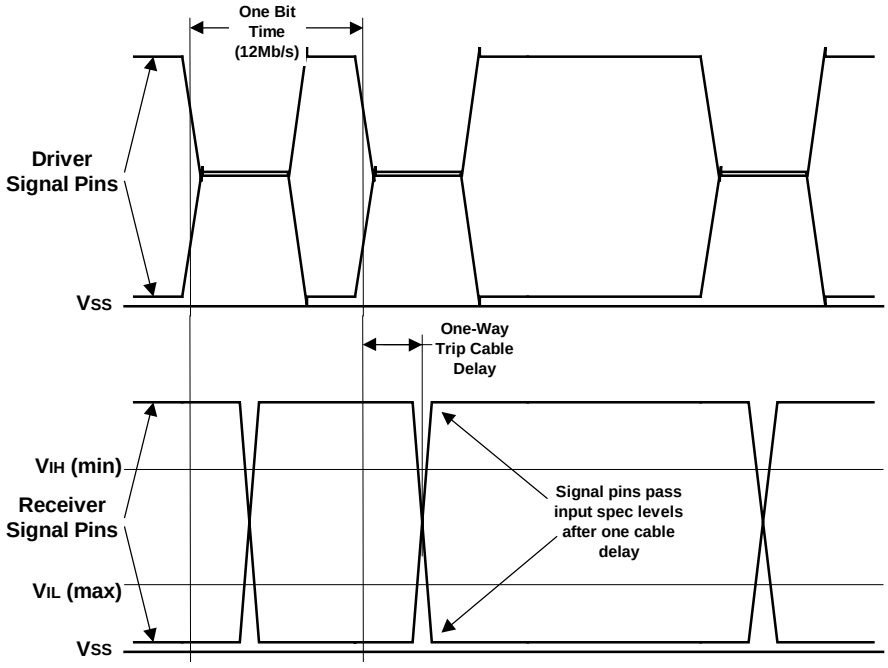


Figure 7-6. Full-speed Signal Waveforms

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

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

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

Figure 7-7 shows the low-speed driver signal waveforms.

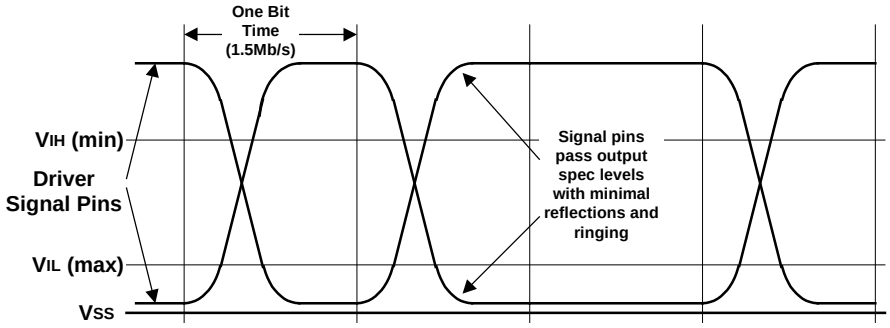


Figure 7-7. Low-speed Driver Signal Waveforms

Microchip Ex. 1018, p. 156
Microchip v. Aptiv
IPR2024-00494

7.1.1.3 High-speed (480 Mb/s) Driver Characteristics

A high-speed USB connection is made through a shielded, twisted pair cable with a differential characteristic impedance (Z_0) of 90 Ω ±15%, a common mode impedance (Z_{CM}) of 30 Ω ±30%, and a maximum one-way delay of 26 ns (T_{FSCBL}). The D+ and D- circuit board traces which run between a transceiver and its associated connector should also have a nominal differential impedance of 90 Ω, and together they may add an additional 4 ns of delay between the transceivers. (See Section 7.1.6 for details on impedance specifications of boards and transceivers.) The differential output impedance of a high-speed capable driver is required to be 90 Ω ±10%. When either the D+ or D- lines are driven high, V_{HSOH} (the high-speed mode high-level output voltage driven on a data line with a precision 45 Ω load to GND) must be 400 mV ±10%. On a line which is not driven, either because the transceiver is not transmitting or because the opposite line is being driven high, V_{HSOL} (the high-speed mode low-level output voltage driven on a data line with a 45 Ω load to GND) must be 0 V ± 10 mV.

Note: Unless indicated otherwise, all voltage measurements are to be made with respect to the local circuit ground.

Note: This specification requires that a high-speed capable transceiver operating in full-speed or low-speed mode must have a driver impedance (Z_{HSDRV}) of 45 Ω ±10%. It is recommended that the driver impedances be matched to within 5 Ω within a transceiver. For upstream facing transceivers which do not support high-speed mode, the driver output impedance (Z_{DRV}) must fall within the range of 28 Ω to 44 Ω.

On downstream facing ports, R_{PD} resistors (15 kΩ ±5%) must be connected from D+ and D- to ground.

When a high-speed capable transceiver transitions to high-speed mode, the high-speed idle state is achieved by driving SE0 with the low-/full-speed drivers at each end of the link (so as to provide the required terminations), and by disconnecting the D+ pull-up resistor in the upstream facing transceiver.

In the preferred embodiment, a transceiver activates its high-speed current driver only when transmitting high-speed signals. This is a potential design challenge, however, since the signal amplitude and timing specifications must be met even on the first symbol within a packet. As a less efficient alternative, a transceiver may cause its high-speed current source to be continually active while in high-speed mode. When the transceiver is not transmitting, the current may be directed into the device ground rather than through the current steering switch which is used for data signaling. In the example circuit, steering the current to ground is accomplished by setting HS_Drive_Enable low.

In CMOS implementations, the driver impedance will typically be realized by the combination of the driver's intrinsic output impedance and R_S. To optimally control Z_{HSDRV} and to minimize parasitics, it is preferred the driver impedance be minimized (under 5 Ω) and the balance of the 45 Ω should be contributed by the R_S component.

When a transceiver operating in high-speed mode transmits, the transmit current is directed into either the D+ or D- data line. A J is asserted by directing the current to the D+ line, a K by directing it to the D- line.

When each of the data lines is terminated with a 45 Ω resistor to the device ground, the effective load resistance on each side is 22.5 Ω. Therefore, the line into which the drive current is being directed rises to 17.78 ma * 22.5 Ω or 400 mV (nominal). The other line remains at the device ground voltage. When the current is directed to the opposite line, these voltages are reversed.

7.1.2 Data Signal Rise and Fall, Eye Patterns

The following sections specify the data signal rise and fall times for full-speed and low-speed signaling, and the rise time and eye patterns for high-speed signaling.

Microchip Ex. 1018, p. 157
Microchip v. Aptiv
IPR2024-00494

7.1.2.1 Low-speed and Full-speed Data Signal Rise and Fall

For low-speed and full-speed, the output rise time and fall times are measured between 10% and 90% of the signal (Figure 7-8). Rise and fall time requirements apply to differential transitions as well as to transitions between differential and single-ended signaling.

The rise and fall times for full-speed buffers are measured with the load shown in Figure 7-9. The rise and fall times must be between 4 ns and 20 ns and matched to within ±10% to minimize RFI emissions and signal skew. The transitions must be monotonic.

The rise and fall times for low-speed buffers are measured with the load shown in Figure 7-10. The capacitive load shown in Figure 7-10 is representative of the worst-case load allowed by the specification. A downstream facing transceiver is allowed 150 pF of input/output capacitance (C_{IND}). A low-speed device (including cable) may have a capacitance of as little as 200 pF and as much as 450 pF. This gives a range of 200 pF to 600 pF as the capacitive load that a downstream facing low-speed buffer might encounter. Upstream facing buffers on low-speed devices must be designed to drive the capacitance of the attached cable plus an additional 150 pF. If a low-speed buffer is designed for an application where the load capacitance is known to fall in a different range, the test load can be adjusted to match the actual application. Low-speed buffers on hosts and hubs that are attached to USB receptacles must be designed for the 200 pF to 600 pF range. The rise and fall time must be between 75 ns and 300 ns for any balanced, capacitive test load. In all cases, the edges must be matched to within ±20% to minimize RFI emissions and signal skew. The transitions must be monotonic.

For both full-speed and low-speed signaling, the crossover voltage (V_{CRS}) must be between 1.3 V and 2.0 V.

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

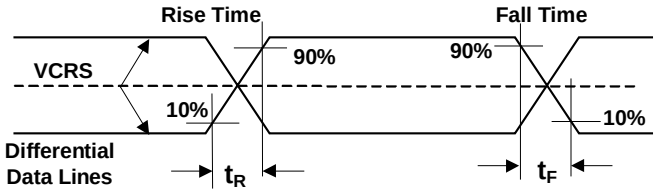


Figure 7-8. Data Signal Rise and Fall Time

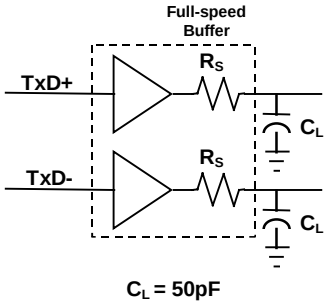


Figure 7-9. Full-speed Load

Microchip Ex. 1018, p. 158
Microchip v. Aptiv
IPR2024-00494

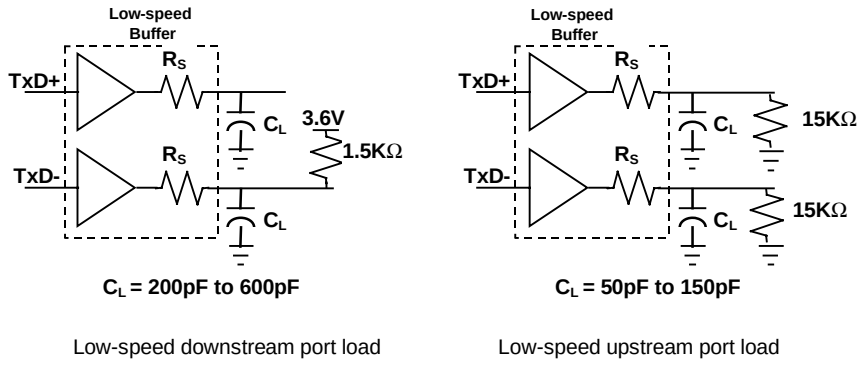


Figure 7-10. Low-speed Port Loads

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

7.1.2.2 High-speed Signaling Eye Patterns and Rise and Fall Time

The following specifications apply to high-speed mode signaling. All bits, including the first and last bit of a packet, must meet the following eye pattern requirements for timing and amplitude.

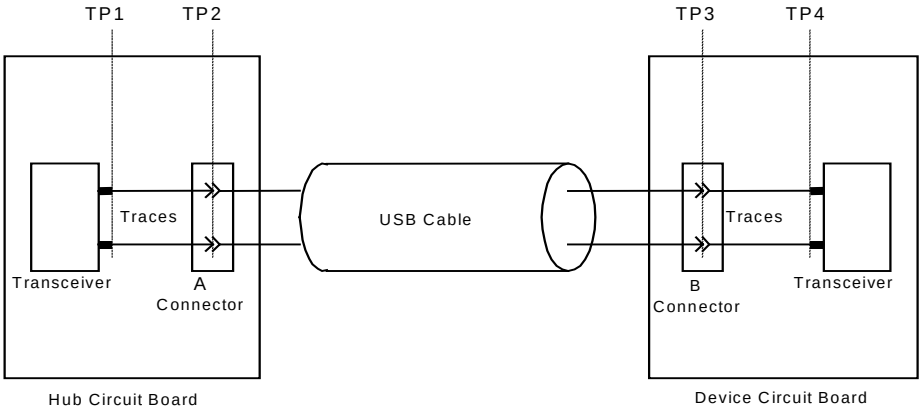


Figure 7-11. Measurement Planes

Figure 7-11 defines four test planes which will be referenced in this section. TP1 and TP4 are the points where the transceiver IC pins are soldered to the hub and device circuit boards, respectively. TP2 is at the mated pins of the A connector, and TP3 is at the mated pins of the B connector (or, in the case of a captive cable, where the cable is attached to the circuit board). The following differential eye pattern templates specify transmit waveform and receive sensitivity requirements at various points and under various conditions.

When testing high-speed transmitters and receivers, measurements are made with the Transmitter/Receiver Test Fixture shown in Figure 7-12. In either case, the fixture is attached to the USB connector closest to the transceiver being tested.

Microchip Ex. 1018, p. 159
Microchip v. Aptiv
IPR2024-00494

Transmitter Test Attenuation: Voltage at Scope Inputs = 0.760 * Voltage at Transmitter Outputs
Receiver Test Attenuation: Voltage at Receiver Inputs = 0.684 * Voltage at Data Generator Outputs

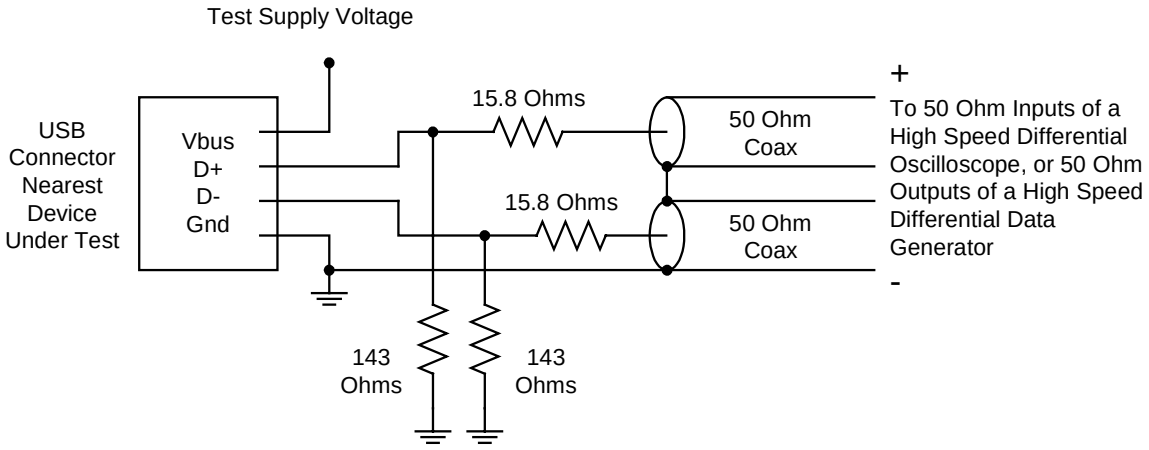


Figure 7-12. Transmitter/Receiver Test Fixture

Note: When testing the upstream facing port of a device, VBUS must be provided from the time the device is placed in the appropriate test mode until the test is completed. This requirement will likely necessitate additional switching functionality in the test fixture (for example, to switch the D+ and D- lines between the host controller and the test instrument). Such additions must have minimal impact on the high frequency measurement results.

Transmit eye patterns specify the minimum and maximum limits, as well as limits on timing jitter, within which a driver must drive signals at each of the specified test planes. Receive eye patterns specify the minimum and maximum limits, as well as limits on timing jitter, within which a receiver must recover data.

Conformance to Templates 1, 2, 3, and 4 is required for USB 2.0 hubs and devices:

Template 1: Transmit waveform requirements for hub measured at TP2, and for device (without a captive cable) measured at TP3

Template 2: Transmit waveform requirements for device (with a captive cable) measured at TP2

Template 3: Receiver sensitivity requirements for device (with a captive cable) when signal is applied at TP2

Template 4: Receiver sensitivity requirements for device (without a captive cable) when signal is applied at TP3, and for hub when signal is applied at TP2

Templates 5 and 6 are recommended guidelines for designers:

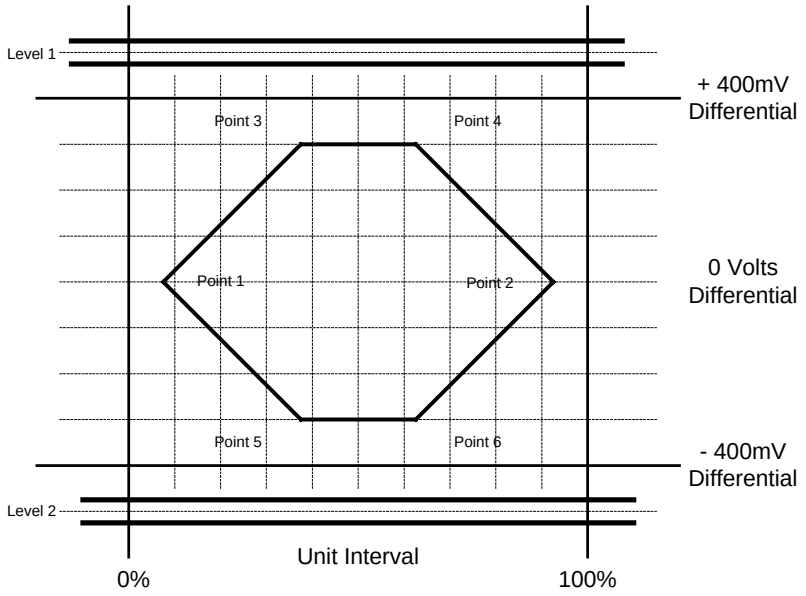
Template 5: Transmit waveform requirements for hub transceiver measured at TP1, and for device transceiver measured at TP4

Template 6: Receiver sensitivity requirements for device transceiver when signal is applied at TP4, and for hub transceiver at when signal is applied at TP1

Microchip Ex. 1018, p. 160
Microchip v. Aptiv
IPR2024-00494

Template 1

Figure 7-13 shows the transmit waveform requirements for a hub measured at TP2, and for a device (without a captive cable) measured at TP3.



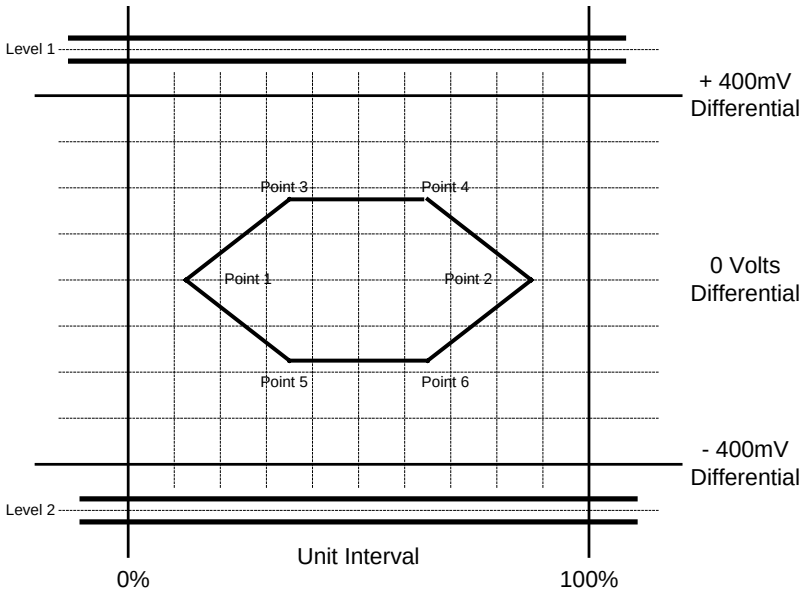
	Voltage Level (D+ - D-)	Time (% of Unit Interval)
Level 1	525 mV in UI following a transition, 475 mV in all others	N/A
Level 2	-525 mV in UI following a transition, -475 in all others	N/A
Point 1	0 V	7.5% UI
Point 2	0 V	92.5% UI
Point 3	300 mV	37.5% UI
Point 4	300 mV	62.5% UI
Point 5	-300 mV	37.5% UI
Point 6	-300 mV	62.5% UI

Figure 7-13. Template 1

Microchip Ex. 1018, p. 161
Microchip v. Aptiv
IPR2024-00494

Template 2

Figure 7-14 shows transmit waveform requirements for a device (with a captive cable) measured at TP2.



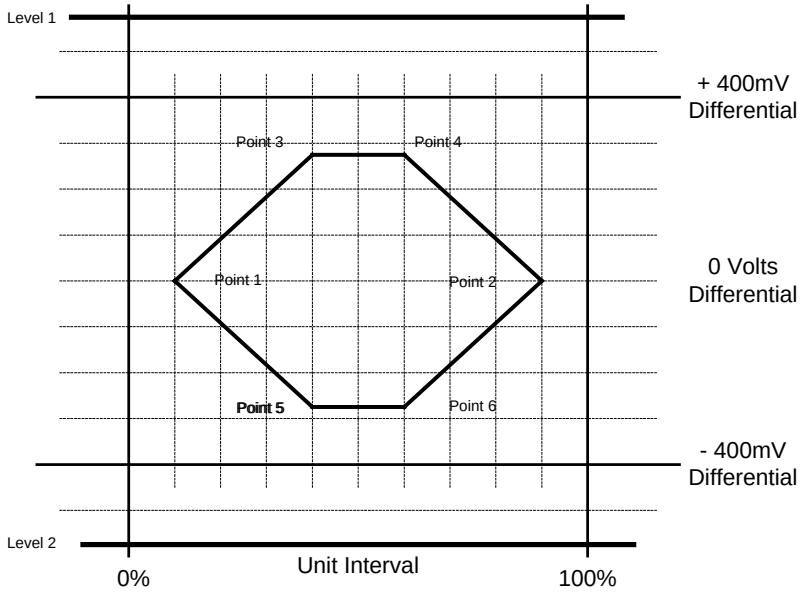
	Voltage Level (D+ - D-)	Time (% of Unit Interval)
Level 1	525 mV in UI following a transition, 475 mV in all others	N/A
Level 2	-525 mV in UI following a transition, -475 in all others	N/A
Point 1	0 V	12.5% UI
Point 2	0 V	87.5% UI
Point 3	175 mV	35% UI
Point 4	175 mV	65% UI
Point 5	-175 mV	35% UI
Point 6	-175 mV	65% UI

Figure 7-14. Template 2

Microchip Ex. 1018, p. 162
Microchip v. Aptiv
IPR2024-00494

Template 3

Figure 7-15 shows receiver sensitivity requirements for a device (with a captive cable) when a signal is applied at TP2.



	Voltage Level (D+ - D-)	Time (% of Unit Interval)
Level 1	575 mV	N/A
Level 2	-575 mV	N/A
Point 1	0 V	10% UI
Point 2	0 V	90% UI
Point 3	275 mV	40% UI
Point 4	275 mV	60% UI
Point 5	-275 mV	40% UI
Point 6	-275 mV	60% UI

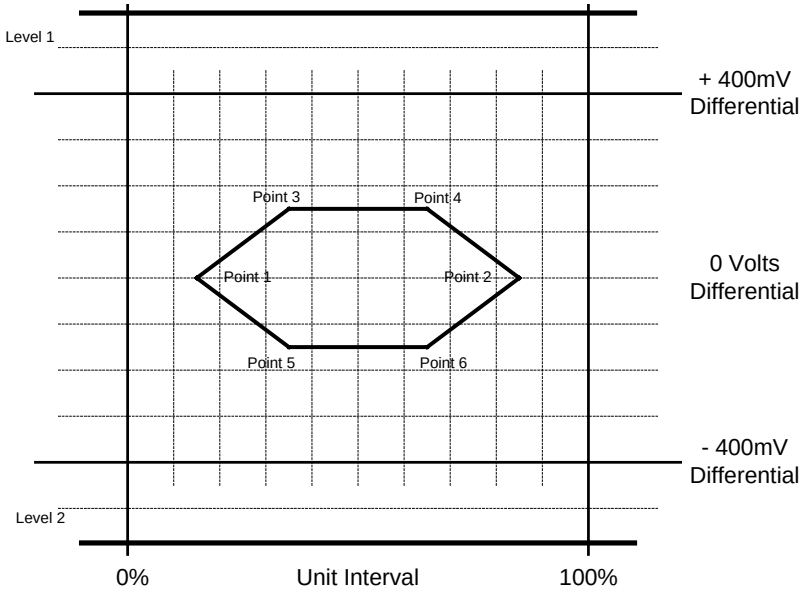
Figure 7-15. Template 3

Note: This eye is intended to specify differential data receiver sensitivity requirements. Levels 1 and 2 are outside the Disconnect Threshold values, but disconnection is detected at the source (after a minimum of 32 bit times without any transitions), not at the target receiver.

Microchip Ex. 1018, p. 163
Microchip v. Aptiv
IPR2024-00494

Template 4

Figure 7-16 shows receiver sensitivity requirements for a device (without a captive cable) when signal is applied at TP3, and for a hub when a signal is applied at TP2.



	Voltage Level (D+ - D-)	Time (% of Unit Interval)
Level 1	575 mV	N/A
Level 2	-575 mV	N/A
Point 1	0 V	15% UI
Point 2	0 V	85% UI
Point 3	150 mV	35% UI
Point 4	150 mV	65% UI
Point 5	-150 mV	35% UI
Point 6	-150 mV	65% UI

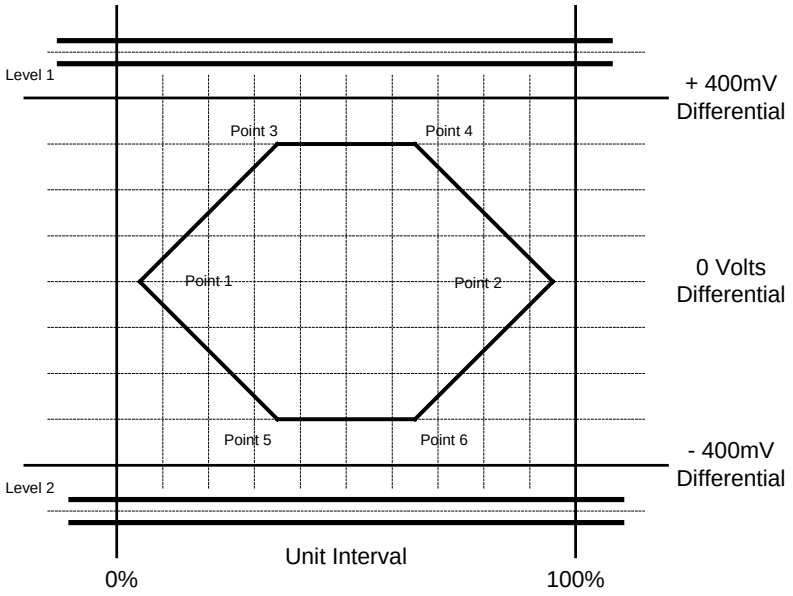
Figure 7-16. Template 4

Note: This eye is intended to specify differential data receiver sensitivity requirements. Levels 1 and 2 are outside the Disconnect Threshold values, but disconnection is detected at the source (after a minimum of 32 bit times without any transitions), not at the target receiver.

Microchip Ex. 1018, p. 164
Microchip v. Aptiv
IPR2024-00494

Template 5

Figure 7-17 shows transmit waveform requirements for a hub transceiver measured at TP1 and for a device transceiver measured at TP4.



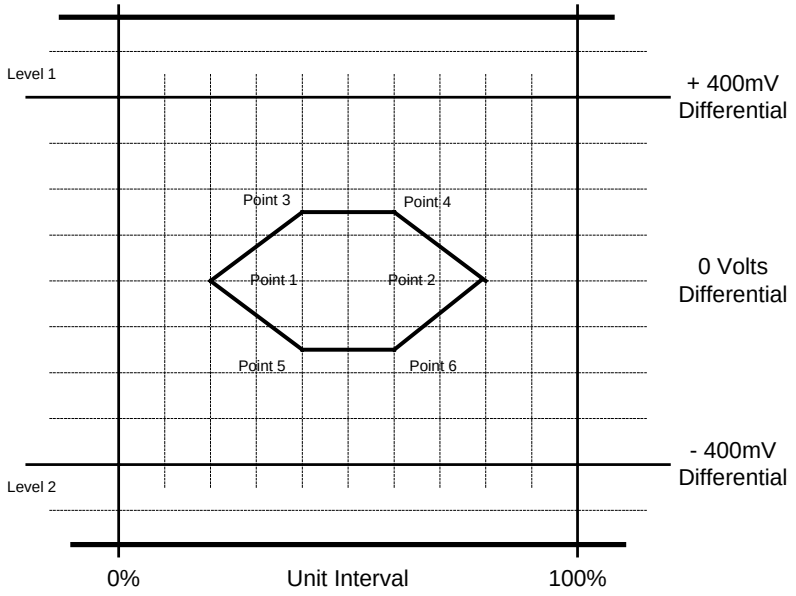
	Voltage Level (D+ - D-)	Time (% of Unit Interval)
Level 1	525 mV in UI following a transition, 475 mV in all others	N/A
Level 2	-525 mV in UI following a transition, -475 in all others	N/A
Point 1	0 V	5% UI
Point 2	0 V	95% UI
Point 3	300 mV	35% UI
Point 4	300 mV	65% UI
Point 5	-300 mV	35% UI
Point 6	-300 mV	65% UI

Figure 7-17. Template 5

Microchip Ex. 1018, p. 165
Microchip v. Aptiv
IPR2024-00494

Template 6

Figure 7-18 shows receiver sensitivity requirements for a device transceiver when a signal is applied at TP4 and for a hub transceiver when a signal is applied at TP1.



	Voltage Level (D+ - D-)	Time (% of Unit Interval)
Level 1	575 mV	N/A
Level 2	-575 mV	N/A
Point 1	0 V	20% UI
Point 2	0 V	80% UI
Point 3	150 mV	40% UI
Point 4	150 mV	60% UI
Point 5	-150 mV	40% UI
Point 6	-150 mV	60% UI

Figure 7-18. Template 6

Note: This eye is intended to specify differential data receiver sensitivity requirements. Levels 1 and 2 are outside the Disconnect Threshold values, but disconnection is detected at the source (after a minimum of 32 bit times without any transitions), not at the target receiver.

Microchip Ex. 1018, p. 166
Microchip v. Aptiv
IPR2024-00494

High-speed Signaling Rise and Fall Times

The transition time of a high-speed driver must not be less than the specified minimum allowable differential rise and fall time (T_{HSR} and T_{HSF}). Transition times are measured when driving a reference load of 45 Ω to ground on D+ and D-. Figure 7-12 shows a recommended "Transmitter Test Fixture" for performing these measurements.

For a hub, or for a device with detachable cable, the 10% to 90% high-speed differential rise and fall times must be 500 ps or longer when measured at the A or B receptacles (respectively).

For a device with a captive cable assembly, it is a recommended design guideline that the 10% to 90% high-speed differential rise and fall times must be 500 ps or longer when measured at the point where the cable is attached to the device circuit board.

It is required that high-speed data transitions be monotonic over the minimum vertical openings specified in the preceding eye pattern templates.

7.1.2.3 Driver Usage

The upstream facing ports of functions must use one and only one of the following three driver configurations:

1. Low-speed – Low-speed drivers only
2. Full-speed – Full-speed drivers only
3. Full-/high-speed – Combination full-speed and high-speed drivers

Upstream facing USB 2.0 hub ports must use full-/high-speed drivers. Such ports must be capable of transmitting data at low-speed and full-speed rates with full-speed signaling, and at the high-speed rate using high-speed signaling. Downstream facing ports (including the host) must support low-speed, full-speed, and high-speed signaling, and must be able to transmit data at each of the three associated data rates.

In this section, there is reference to a situation in which high-speed operation is "disallowed." This topic is discussed in depth in Chapter 11 of this specification. In brief, a high-speed capable hub's downstream facing ports are "high-speed disallowed" if the hub is unable to establish a high-speed connection on its upstream facing port. For example, this would be the case for the downstream facing ports of a high-speed capable hub when the hub is connected to a USB 1.1 host controller.

When a full-/high-speed device is attached to a pre-USB 2.0 hub, or to a hub port which is high-speed disallowed, it is required to behave as a full-speed only device. When a full-/high-speed device is attached to a USB 2.0 hub which is not high-speed disallowed, it must operate with high-speed signaling and data rate.

7.1.3 Cable Skew

The maximum skew introduced by the cable between the differential signaling pair (i.e., D+ and D- (T_{SKEW})) must be less than 100 ps and is measured as described in Section 6.7.

7.1.4 Receiver Characteristics

This section discusses the receiver characteristics for low-speed, full-speed, and full-/high-speed transceivers.

7.1.4.1 Low-speed and Full-speed Receiver Characteristics

A differential input receiver must be used to accept the USB data signal. The receiver must feature an input sensitivity (V_{DI}) of at least 200 mV when both differential data inputs are in the differential common mode range (V_{CM}) of 0.8 V to 2.5 V, as shown in Figure 7-19.

Microchip Ex. 1018, p. 167
Microchip v. Aptiv
IPR2024-00494

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

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

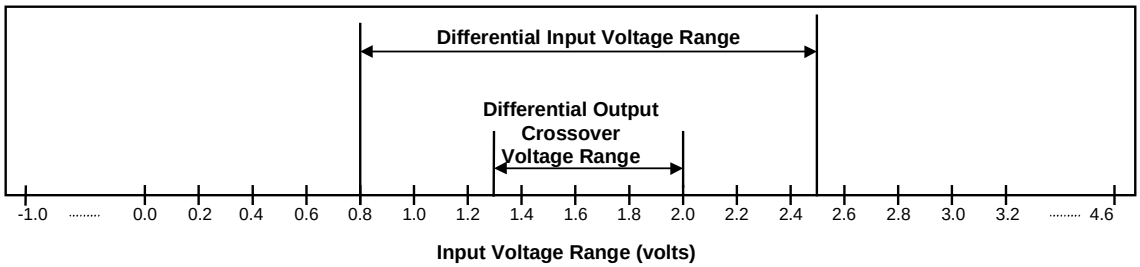


Figure 7-19. Differential Input Sensitivity Range for Low-/full-speed

7.1.4.2 High-speed Receiver Characteristics

A high-speed capable transceiver receiver must conform to the receiver characteristics specifications called out in Section 7.1.4.1 when receiving in low-speed or full-speed modes.

As shown in Figure 7-1, a high-speed capable transceiver which is operating in high-speed mode "listens" for an incoming serial data stream with the high-speed differential data receiver and the transmission envelope detector. Additionally, a downstream facing high-speed capable transceiver monitors the amplitude of the differential voltage on the lines with the disconnection envelope detector.

When receiving in high-speed mode, the differential receiver must be able to reliably receive signals that conform to the Receiver Eye Pattern templates shown in Section 7.1.2. Additionally, it is a strongly recommended guideline that a high-speed receiver should be able to reliably receive such signals in the presence of a common mode voltage component (V_{HSCM}) over the range of –50 mV to 500 mV (the nominal common mode component of high-speed signaling is 200 mV). Low frequency chirp J and K signaling, which occurs during the Reset handshake, should be reliably received with a common mode voltage range of –50 mV to 600 mV.

Reception of data is qualified by the output of the transmission envelope detector. The receiver must disable data recovery when the signal falls below the high-speed squelch level (V_{HSSQ}) defined in Table 7-3. (Detector must indicate squelch when the magnitude of the differential voltage envelope is ≤ 100 mV, and must not indicate squelch if the amplitude of differential voltage envelope is ≥ 150 mV.) Squelch detection must be done with a differential envelope detector, such as the one shown in Figure 7-1. The envelope detector used to detect the squelch state must incorporate a filtering mechanism that prevents indication of squelch during differential data crossovers.

The definition of a high-speed packet's SYNC pattern, together with the requirements for high-speed hub repeaters, guarantee that a receiver will see at least 12 bits of SYNC (KJKJKJKJKJKK) followed by the data portion of the packet. This means that the combination of squelch response time, DLL lock time, and end of SYNC detection must occur within 12 bit times. This is required to assure that the first bit of the packet payload will be received correctly.

In the case of a downstream facing port, a high-speed capable transceiver must include a differential envelope detector that indicates when the signal on the data exceeds the high-speed Disconnect level (V_{HSDSC}) as defined in Table 7-3. (The detector must not indicate that the disconnection threshold has been exceeded if the differential signal amplitude is ≤525 mV, and must indicate that the threshold has been exceeded if the differential signal amplitude is ≥625 mV.)

Microchip Ex. 1018, p. 168
Microchip v. Aptiv
IPR2024-00494

When sampled at the appropriate time, this detector provides indication that the device has been disconnected. The details of how the disconnection envelope detector is used are described in Section 7.1.7.3.

7.1.5 Device Speed Identification

The following sections specify the speed identification mechanisms for low-speed, full-speed, and high-speed.

7.1.5.1 Low-/Full-speed Device Speed Identification

The USB is terminated at the hub and function ends as shown in Figure 7-20 and Figure 7-21. Full-speed and low-speed devices are differentiated by the position of the pull-up resistor on the downstream end of the cable:

- Full-speed devices are terminated as shown in Figure 7-20 with the pull-up resistor on the D+ line.
- Low-speed devices are terminated as shown in Figure 7-21 with the pull-up resistor on the D- line.
- The pull-down terminators on downstream facing ports are resistors of 15 kΩ ±5% connected to ground.

The design of the pull-up resistor must ensure that the signal levels satisfy the requirements specified in Table 7-2. In order to facilitate bus state evaluation that may be performed at the end of a reset, the design must be able to pull-up D+ or D- from 0 V to VIH (min) within the minimum reset relaxation time of 2.5 μs. A device that has a detachable cable must use a 1.5 kΩ ±5% resistor tied to a voltage source between 3.0 V and 3.6 V (VTERM) to satisfy these requirements. Devices with captive cables may use alternative termination means. However, the Thevenin resistance of any termination must be no less than 900 Ω.

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

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

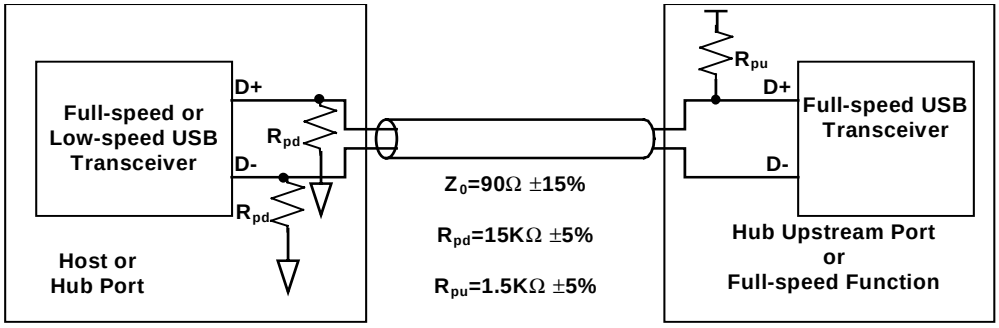


Figure 7-20. Full-speed Device Cable and Resistor Connections

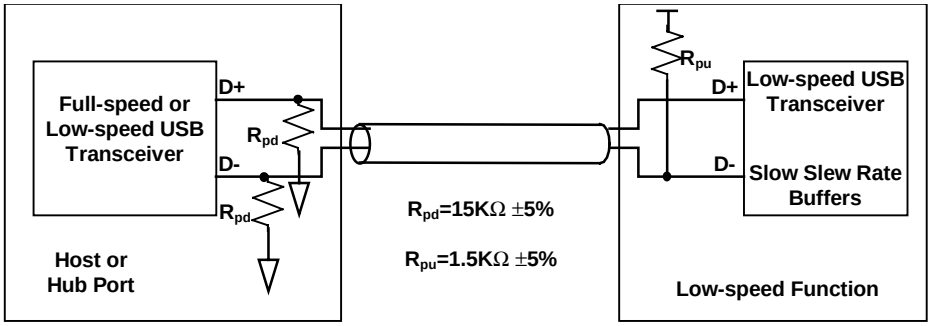


Figure 7-21. Low-speed Device Cable and Resistor Connections

Microchip Ex. 1018, p. 169
Microchip v. Aptiv
IPR2024-00494

7.1.5.2 High-speed Device Speed Identification

The high-speed Reset and Detection mechanisms follow the behavioral model for low-/full-speed. When reset is complete, the link must be operating in its appropriate signaling mode (low-speed, full-speed, or high-speed as governed by the preceding usage rules), and the speed indication bits in the port status register will correctly report this mode. Software need only initiate the assertion of reset and read the port status register upon notification of reset completion.

High-speed capable devices initially attach as full-speed devices. This means that for high-speed capable upstream facing ports, R_{PU} (1.5 kΩ ±5%) must be connected from D+ to the 3.3 V supply (as shown in Figure 7-1) through a switch which can be opened under SW control.

After the initial attachment, high-speed capable transceivers engage in a low level protocol during reset to establish a high-speed link and to indicate high-speed operation in the appropriate port status register. This protocol is described in Section 7.1.7.5.

7.1.6 Input Characteristics

The following sections describe the input characteristics for transceivers operating in low-speed, full-speed, and high-speed modes.

7.1.6.1 Low-speed and Full-speed Input Characteristics

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

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

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

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

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

An implementation may use small capacitors at the transceiver for purposes of edge rate control. The sum of the capacitance of the added capacitor (C_{EDGE}), the transceiver, and the trace connecting capacitor and transceiver to R_S must not exceed 75 pF (either single-ended or differential) and the capacitance must be balanced to within 10%. The added capacitor, if present, must be placed between the transceiver pins and R_S (see Figure 7-22).

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

Microchip Ex. 1018, p. 170
Microchip v. Aptiv
IPR2024-00494

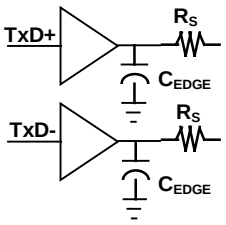


Figure 7-22. Placement of Optional Edge Rate Control Capacitors for Low-/full-speed

7.1.6.2 High-speed Input Characteristics

Figure 7-23 shows the simple equivalent loading circuit of a USB device operating in high-speed receive mode.

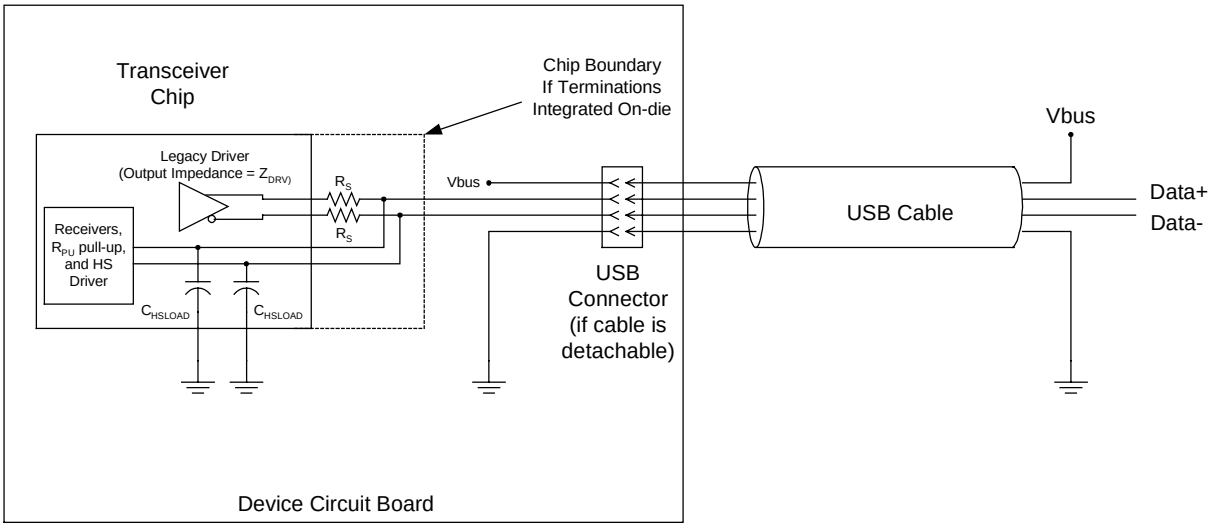


Figure 7-23. Diagram for High-speed Loading Equivalent Circuit

When operating in high-speed signaling mode, a transceiver must meet the following loading specifications:

1. DC output voltage and resistance specifications
2. TDR loading specification

Additionally, it is strongly recommended that a transceiver component operating in high-speed signaling mode should meet the following lumped capacitance guideline.

The use of ferrites on high-speed data lines is strongly discouraged.

DC output voltage and resistance specifications – A transceiver that is in high-speed mode must present a DC load on each of the data lines nominally equivalent to 45 Ω to ground. The actual resistance, Z_{HSDRV}, must be 40.5 Ω ≤ Z_{HSDRV} ≤ 49.5 Ω. The output voltage in the high-speed idle state (V_{HSTERM}) is specified in Table 7-3

TDR loading specification – The AC loading specifications of a transceiver in the high-speed idle state are specified in terms of differential TDR (Time Domain Reflectometer) measurements.

These measurements govern the maximum allowable transmission line discontinuities for the port connector, the interconnect leading from the connector to the transceiver, the transceiver package, and the transceiver IC itself. In the special case of a high-speed capable device with a captive cable, the transmission line discontinuities of the cable assembly are also governed.

Microchip Ex. 1018, p. 171
Microchip v. Aptiv
IPR2024-00494

The following specifications must be met with the incident rise time of the differential TDR set to 400 ps. It is important to note that all times are "as displayed" on the TDR and are hence "round trip times."

Termination Impedance (Z_{HSTERM}) is measured on the TDR trace at a specific measurement time following the connector reference time. The connector reference time is determined by disconnecting the TDR connection from the port connector and noting the time of the open circuit step. For an A connector, the measurement time is 8 ns after the connector reference location. For a B connector, the measurement time is 4 ns after the connector reference location. The differential termination impedance must be:

$$80\ \Omega \leq Z_{HSTERM} \leq 100\ \Omega$$

Through Impedance (Z_{HSTHRU}) is the impedance measured from 500 ps before the connector reference location until the time governed by the Termination impedance specification.

$$70\ \Omega \leq Z_{HSTHRU} \leq 110\ \Omega$$

In the Exception Window (a sliding 1.4 ns window inside the Through Impedance time window), the differential impedance may exceed the Through limits. No single excursion, however, may exceed the Through limits for more than twice the TDR rise time (400 ps).

In the special case of a high-speed capable device with a captive cable, the same specifications must be met, but the TDR measurements must be made through the captive cable assembly. Determination of the connector reference time can be more difficult in this case, since the cable may not be readily removable from the port being tested. It is left to the tester of a specific device to determine the connector reference location by whatever means are available.

Lumped capacitance guideline for the transceiver component

When characterizing a transceiver chip as an isolated component, the measurement can be performed effectively at the chip boundary shown in Figure 7-23 without USB connectors or cables. Parasitic capacitance of the test fixture can be corrected by measuring the capacitance of the fixture itself and subtracting this reading from the reading taken with the transceiver inserted. If the terminations are off-chip, discrete R_S resistors should be in place during the measurements, and measurements should be taken on the "connector side" of the resistors. The transceiver should be in Test_SE0_NAK mode during testing.

Capacitance measurements are taken from each of the data lines to ground while the other line is left open. The instrument used to perform this measurement must be able to determine the effective capacitance to ground in the presence of the parallel effective resistance to ground.

Capacitance to Ground on each line: $C_{HSLOAD} \leq 10$ pF
Matching of Capacitances to Ground: ≤ 1.0 pF

The guideline is to allow no more than 5.0 pF for the transceiver die itself and no more than an additional 5 pF for the package. The differential capacitance across the transceiver inputs should be no more than 5.0 pF

7.1.7 Signaling Levels

The following sections specify signaling levels for low-speed, full-speed, and high-speed operation.

7.1.7.1 Low-/Full-speed Signaling Levels

Table 7-2 summarizes the USB signaling levels. The source is required to drive the levels specified in the second column, and the target is required to identify the correct bus state when it sees the levels in the third column. (Target receivers can be more sensitive as long as they are within limits specified in the fourth column.)

Microchip Ex. 1018, p. 172
Microchip v. Aptiv
IPR2024-00494

Table 7-2. Low-/full-speed Signaling Levels

Bus State	Signaling Levels		
	At originating source connector (at end of bit time)	At final target connector	
		Required	Acceptable
Differential "1"	D+ > V_{OH} (min) and D- < V_{OL} (max)	(D+) - (D-) > 200 mV and D+ > V_{IH} (min)	(D+) - (D-) > 200 mV
Differential "0"	D- > V_{OH} (min) and D+ < V_{OL} (max)	(D-) - (D+) > 200 mV and D- > V_{IH} (min)	(D-) - (D+) > 200 mV
Single-ended 0 (SE0)	D+ and D- < V_{OL} (max)	D+ and D- < V_{IL} (max)	D+ and D- < V_{IH} (min)
Single-ended 1 (SE1)	D+ and D- > V_{OSE1}(min)	D+ and D- > V_{IL} (max)	
Data J state: Low-speed Full-speed	 Differential "0" Differential "1"	 Differential "0" Differential "1"	
Data K state: Low-speed Full-speed	 Differential "1" Differential "0"	 Differential "1" Differential "0"	
Idle state: Low-speed Full-speed	NA	 D- > V_{IHZ} (min) and D+ < V_{IL} (max) D+ > V_{IHZ} (min) and D- < V_{IL} (max)	 D- > V_{IHZ} (min) and D+ < V_{IH} (min) D+ > V_{IHZ} (min) and D- < V_{IH} (min)
Resume state	Data K state	Data K state	
Start-of-Packet (SOP)	Data lines switch from Idle to K state		
End-of-Packet (EOP)[4]	SE0 for approximately 2 bit times[1] followed by a J for 1 bit time[3]	SE0 for ≥ 1 bit time[2] followed by a J state for 1 bit time	SE0 for ≥ 1 bit time[2] followed by a J state
Disconnect (at downstream port)	NA	SE0 for ≥2.5 μs	
Connect (at downstream port)	NA	Idle for ≥2 ms	Idle for ≥2.5 μs
Reset	D+ and D- < VOL (max) for ≥10ms	D+ and D- < VIL (max) for ≥10 ms	D+ and D- < VIL (max) for ≥2.5 μs

Note 1: The width of EOP is defined in bit times relative to the speed of transmission. (Specification EOP widths are given in Table 7-7 and Table 7-8.)

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

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

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

Microchip Ex. 1018, p. 173
Microchip v. Aptiv
IPR2024-00494

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

As shown in Table 7-2, the J and K states for full-speed signaling are inverted from those for low-speed signaling. The sense of data, idle, and resume signaling is set by the type of device that is being attached to a port. If a full-speed device is attached to a port, that segment of the USB uses full-speed signaling conventions (and fast rise and fall times), even if the data being sent across the data lines is at the low-speed data rate. The low-speed signaling conventions shown in Table 7-2 (plus slow rise and fall times) are used only between a low-speed device and the port to which it is attached.

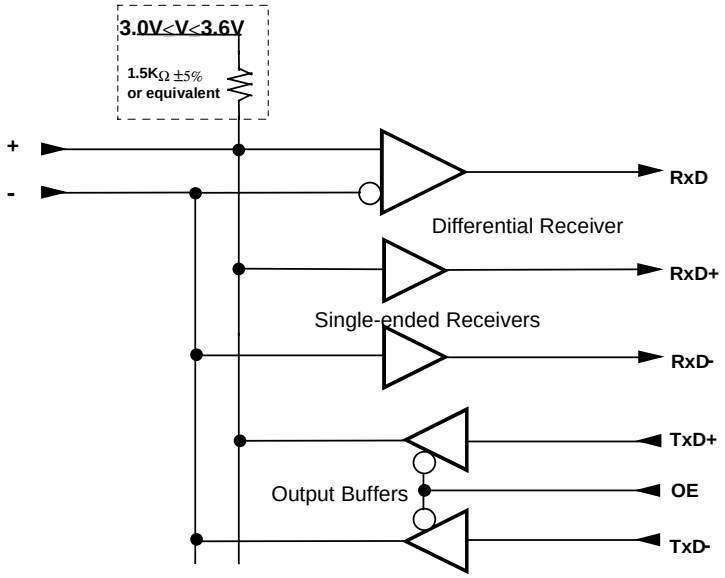


Figure 7-24. Upstream Facing Full-speed Port Transceiver

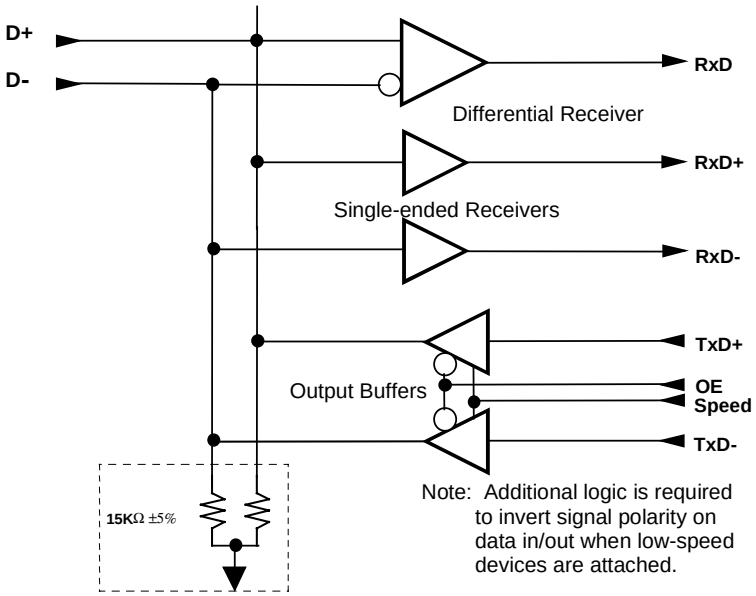


Figure 7-25. Downstream Facing Low-/full-speed Port Transceiver

Microchip Ex. 1018, p. 174
Microchip v. Aptiv
IPR2024-00494

7.1.7.2 Full-/High-speed Signaling Levels

The high-speed signaling voltage specifications in Table 7-3 must be met when measuring at the connector closest to the transceiver, using precision 45 Ω load resistors to the device ground as reference loads. All voltage measurements are taken with respect to the local device ground.

Table 7-3. High-speed Signaling Levels

Bus State	Required Signaling Level at Source Connector	Required Signaling Level at Target Connector
High-speed Differential "1"	**DC Levels:** V_{HSOH} (min) ≤ D+ ≤ V_{HSOH} (max) V_{HSOL} (min) ≤ D- ≤ V_{HSOL} (max) See Note 1. **AC Differential Levels:** A transmitter must conform to the eye pattern templates called out in Section 7.1.2. See Note 2.	**AC Differential Levels** The signal at the target connector must be recoverable, as defined by the eye pattern templates called out in Section 7.1.2. See Note 2.
High-speed Differential "0"	**DC Levels:** V_{HSOH} (min) ≤ D- ≤ V_{HSOH} (max) V_{HSOL} (min) ≤ D+ ≤ V_{HSOL} (max) See Note 1. **AC Differential Levels:** A transmitter must conform to the eye pattern templates called out in Section 7.1.2. See Note 2.	**AC Differential Levels:** The signal at the target connector must be recoverable, as defined by the eye pattern templates called out in Section 7.1.2. See Note 2.
High-speed J State	High-speed Differential "1"	High-speed Differential "1"
High-speed K State	High-speed Differential "0"	High-speed Differential "0"

Microchip Ex. 1018, p. 175
Microchip v. Aptiv
IPR2024-00494

Table 7-3. High-speed Signaling Levels (Continued)

Chirp J State (differential voltage; applies only during reset when both hub and device are high-speed capable)	**DC Levels:** V_{CHIRPJ} (min) ≤ (D+ - D-) ≤ V_{CHIRPJ} (max)	**AC Differential Levels** The differential signal at the target connector must be ≥ 300 mV
Chirp K State (differential voltage; applies only during reset when both hub and device are high-speed capable)	**DC Levels:** V_{CHIRPK} (min) ≤ (D+ - D-) ≤ V_{CHIRPK} (max)	**AC Differential Levels** The differential signal at the target connector must be ≤ -300 mV
High-speed Squelch State	NA	V_{HSSQ} - Receiver must indicate squelch when magnitude of differential voltage is ≤100 mV; receiver must not indicate squelch if magnitude of differential voltage is ≥150 mV. See Note 3.
High-speed Idle State	NA	**DC Levels:** V_{HSOI} min ≤ (D+, D-) ≤ V_{HSOI} max See Note 1. **AC Differential Levels:** Magnitude of differential voltage is ≤ 100 mV See Note 3.
Start of High-speed Packet (HSSOP)	Data lines switch from high-speed Idle to high-speed J or high-speed K state.	
End of High-speed Packet (HSEOP)	Data lines switch from high-speed J or K to high-speed Idle state.	
High-speed Disconnect State (at downstream facing port)	NA	V_{HSDSC} - Downstream facing port must not indicate device disconnection if differential voltage is ≤ 525 mV, and must indicate device disconnection when magnitude of differential voltage is ≥ 625 mV, at the sample time discussed in Section 7.1.7.3.

Note 1: Measured with a 45 Ω resistor to ground at each data line, using test modes Test_J and Test_K

Note 2: Measured using test mode Test_Packet with fixture shown in Figure 7-12

Note 3: Measured with fixture shown in Figure 7-12, using test mode SE0_NACK

Note 4: A high-speed driver must never “intentionally” generate a signal in which both D+ and D- are driven to a level above 200 mV. The current-steering design of a high-speed driver should naturally preclude this possibility.

Microchip Ex. 1018, p. 176
Microchip v. Aptiv
IPR2024-00494

7.1.7.3 Connect and Disconnect Signaling

When no function is attached to the downstream facing port of a host or hub in low-/full-speed, the pull-down resistors present there will cause both D+ and D- to be pulled below the single-ended low threshold of the host or hub transceiver when that port is not being driven by the hub. This creates an SE0 state on the downstream facing port. A disconnect condition is indicated if the host or hub is not driving the data lines and an SE0 persists on a downstream facing port for more than TDDIS (see Figure 7-26). The specifications for TDDIS and TDCNN are defined in Table 7-13.

A connect condition will be detected when the hub detects that one of the data lines is pulled above its VIH threshold for more than TDCNN (see Figure 7-27 and Figure 7-28).

Hubs must determine the speed of the attached device by sampling the state of the bus immediately before driving SE0 to indicate a reset condition to the device.

All signaling levels given in Table 7-2 are set for this bus segment (and this segment alone) once the speed of the attached device is determined. The mechanics of speed detection are described in Section 11.8.2.

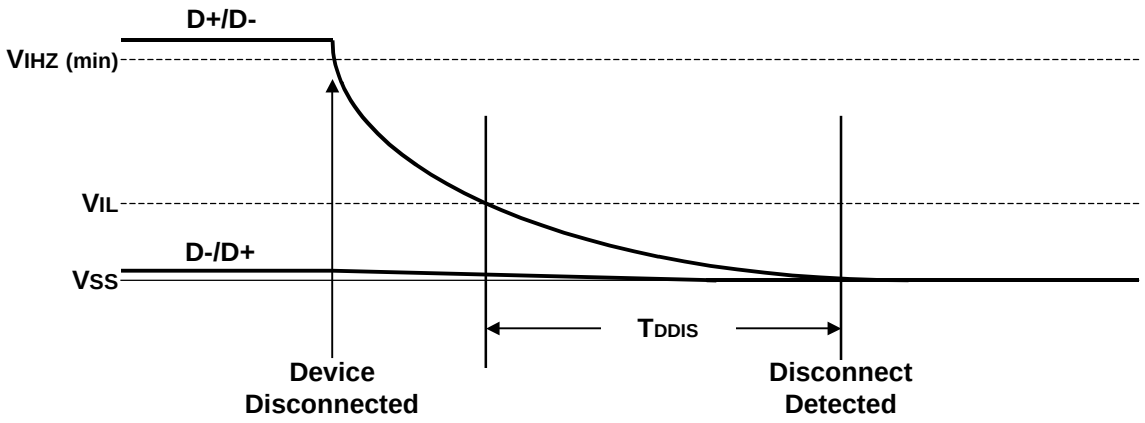


Figure 7-26. Low-/full-speed Disconnect Detection

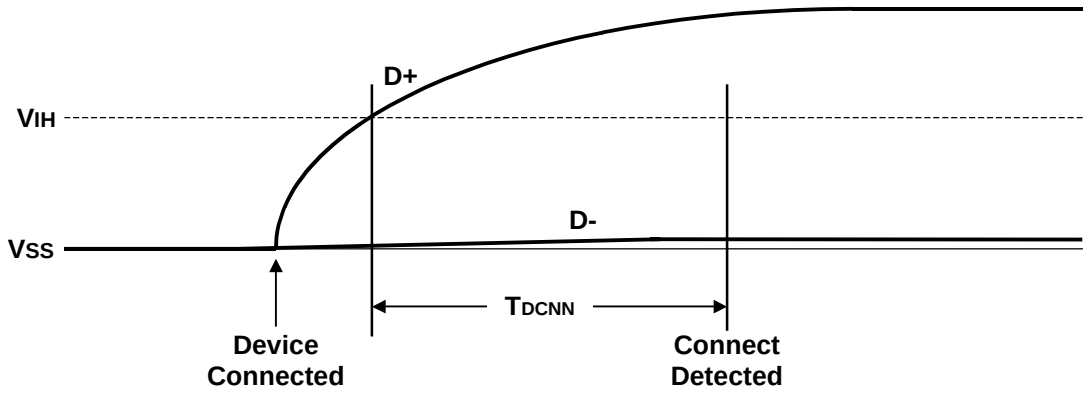


Figure 7-27. Full-/high-speed Device Connect Detection

Microchip Ex. 1018, p. 177
Microchip v. Aptiv
IPR2024-00494

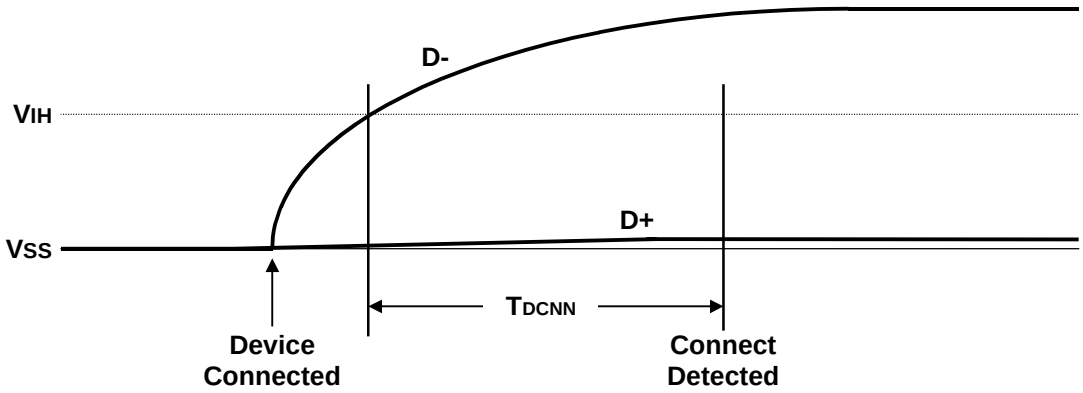


Figure 7-28. Low-speed Device Connect Detection

Because USB components may be hot plugged, and hubs may implement power switching, it is necessary to comprehend the delays between power switching and/or device attach and when the device's internal power has stabilized. Figure 7-29 shows all the events associated with both turning on port power with a device connected and hot-plugging a device. There are six delays and a sequence of events that are defined by this specification.

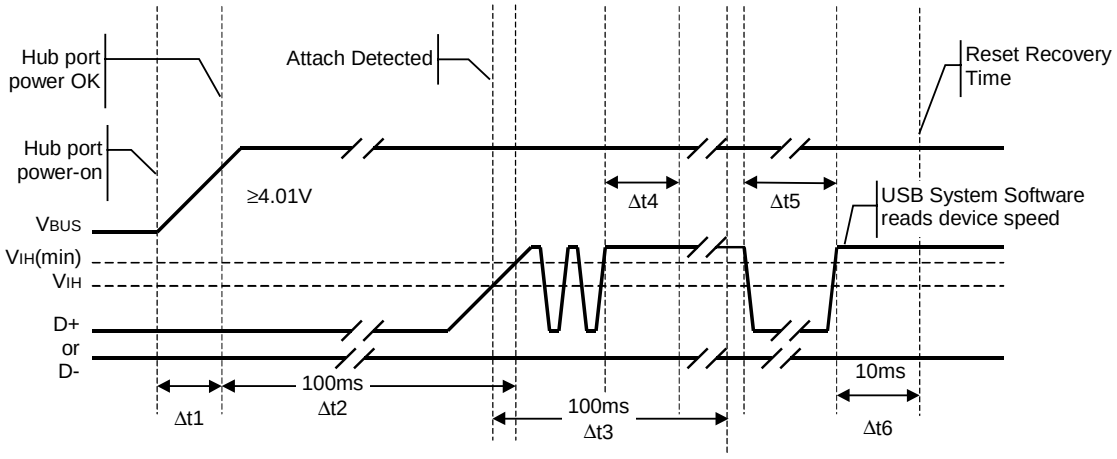


Figure 7-29. Power-on and Connection Events Timing

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

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

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

Δt4 (T2SUSP) Anytime a device observes no bus activity, it must obey the rules of going into suspend (see Section 7.1.7.6).

Microchip Ex. 1018, p. 178
Microchip v. Aptiv
IPR2024-00494

Δt5 (TDRST) This is the period of time hubs drive reset to a device. Refer to Section 7.1.7.5 and Section 11.5.1.5 for details.

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

High-speed capable devices must initially attach as full-speed devices and must comply with all full-speed connection requirements. A high-speed capable downstream facing port must correctly detect the attachment of low-speed and full-speed devices and must also comply with all low-speed and full-speed connection behaviors.

Transition to high-speed signaling is accomplished by means of a low level electrical protocol which occurs during Reset. This protocol is specified in Section 7.1.7.5.

A downstream facing transceiver operating in high-speed mode detects disconnection of a high-speed device by sensing the doubling in differential signal amplitude across the D+ and D- lines that can occur when the device terminations are removed. The Disconnection Envelope Detector output goes high when the downstream facing transceiver transmits and positive reflections from the open line return with a phase which is additive with the transceiver driver signal. Signals with differential amplitudes ≥ 625 mV must reliably activate the Disconnection Envelope Detector. Signals with differential amplitudes ≤ 525 mV must never activate the Disconnection Envelope Detector.

To assure that this additive effect occurs and is of sufficient duration to be detected, the EOP at the end of a high-speed SOF is lengthened to a continuous string of 40 bits without any transitions, as discussed in Section 7.1.13.2. This length is sufficient to guarantee that the voltage at the downstream facing port's connector will double, since the maximum allowable round trip signal delay is 30 bit times.

When a downstream facing port is transmitting in high-speed mode and detects that it has sent 32 bits without a transition, the disconnection envelope detector's output must be sampled once during transmission of the next 8 bits at the transceiver output. (In the absence of bus errors, the next 8 bits will not include a transition.) If the sample indicates that the disconnection detection threshold has been exceeded, the downstream facing port must indicate that the high-speed device has been disconnected. See Section 11.12.4.

7.1.7.4 Data Signaling

Data transmission within a packet is done with differential signals.

7.1.7.4.1 Low-/Full-Speed Signaling

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

The SE0 state is used to signal an end-of-packet (EOP). EOP will be signaled by driving D+ and D- to the SE0 state for two bit times followed by driving the lines to the J state for one bit time. The transition from the SE0 to the J state defines the end of the packet at the receiver. The J state is asserted for one bit time and then both the D+ and D- output drivers are placed in their high-impedance state. The bus termination resistors hold the bus in the Idle state. Figure 7-30 shows the signaling for start and end of a packet.

Microchip Ex. 1018, p. 179
Microchip v. Aptiv
IPR2024-00494

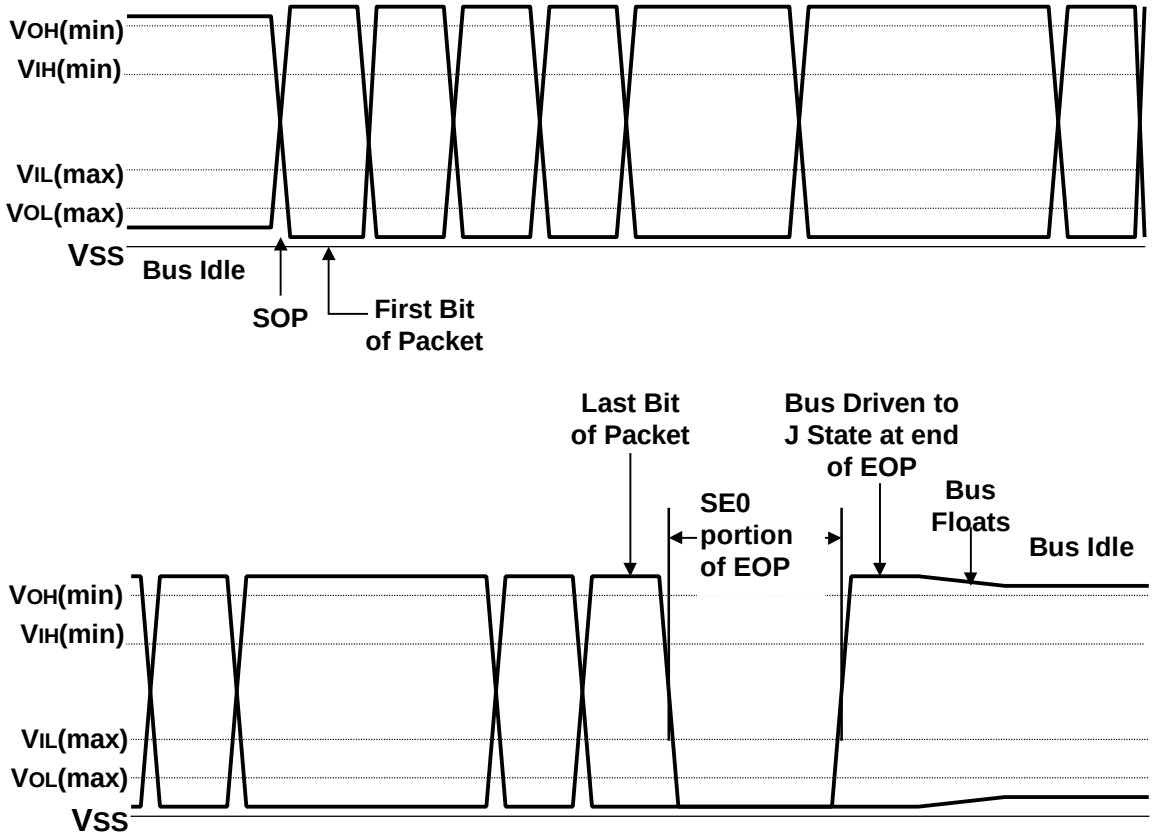


Figure 7-30. Low-/full-speed Packet Voltage Levels

7.1.7.4.2 High-speed Signaling

The high-speed Idle state is when both lines are nominally at GND.

The source of the packet signals the Start of Packet (SOP) in high-speed mode by driving the D+ and D- lines from the high-speed Idle state to the K state. This K is the first symbol of the SYNC pattern (NRZI sequence KJKJKJKJ KJKJKJKJ KJKJKJKJ KJKJKJKK) as described in Section 7.1.10.

The high-speed End of Packet (EOP) begins with a transition from the last symbol before the EOP to the opposite symbol. This opposite symbol is the first symbol in the EOP pattern (NRZ 01111111 with bit stuffing disabled) as described in Section 7.1.13.2. Upon completion of the EOP pattern, the driver ceases to inject current into the D+ or D- lines, and the lines return to the high-speed Idle state. The high-speed SOF EOP is a special case. This SOF EOP is 40 symbols without a transition (rather than 8 for a non-SOF packet).

The fact that the first symbol in the EOP pattern forces a transition simplifies the process of determining precisely which is the last bit in the packet prior to the EOP delimiter.

Microchip Ex. 1018, p. 180
Microchip v. Aptiv
IPR2024-00494

7.1.7.5 Reset Signaling

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

The reset signaling can be generated on any Hub or Host Controller port by request from the USB System Software. The reset signaling must be driven for a minimum of 10ms (TDRST). After the reset, the hub port will transition to the Enabled state (refer to Section 11.5).

As an additional requirement, Host Controllers and the USB System Software must ensure that resets issued to the root ports drive reset long enough to overwhelm any concurrent resume attempts by downstream devices. It is required that resets from root ports have a duration of at least 50 ms (TDRSTR). It is not required that this be 50 ms of continuous Reset signaling. However, if the reset is not continuous, the interval(s) between reset signaling must be less than 3 ms (TRHRSI), and the duration of each SE0 assertion must be at least 10 ms (TDRST).

A device operating in low-/full-speed mode that sees an SE0 on its upstream facing port for more than 2.5 μs (TDETRST) may treat that signal as a reset. The reset must have taken effect before the reset signaling ends.

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

Hubs must be able to accept all hub requests and devices must be able to accept a SetAddress() request (refer to Section 11.24.2 and Section 9.4 respectively) after the reset recovery time 10 ms (TRSTRCY) after the reset is removed. Failure to accept this request may cause the device not to be recognized by the USB system software. Hubs and devices must complete commands within the times specified in Chapter 9 and Chapter 11.

Reset must wake a device from the Suspend state.

It is required that a high-speed capable device can be reset while in the Powered, Default, Address, Configured, or Suspended states shown in Figure 9-1. The reset signaling is compatible with low-/full-speed reset. This means that a hub must successfully reset any device (even USB 1.X devices), and a device must be successfully reset by any hub (even USB1.X hubs).

If, and only if, a high-speed capable device is reset by a high-speed capable hub which is not high-speed disallowed, both hub and device must be operating in the default state in high-speed signaling mode at the end of reset. The hub port status register must indicate that the port is in high-speed signaling mode. This requirement is met by having such a device and such a hub engage in a low level protocol during the reset signaling time. The protocol is defined in such a way that USB 1.X devices will not be disrupted from their normal reset behaviors.

Note: Because the downstream facing port will not be in Transmit state during the Reset Protocol, high-speed Chirp signaling levels will not provoke disconnect detection. (Refer to Section 7.1.7.3 and Section 11.5.1.7.)

Reset Protocol for high-speed capable hubs and devices

1. The hub checks to make sure the attached device is not low-speed. (A low-speed device is not allowed to support high-speed operation. If the hub determines that it is attached to a low-speed device, it does not conduct the following high-speed detection protocol during reset.)

2. The hub drives SE0. In this description of the Reset Protocol and High-speed Detection Handshake, the start of SE0 is referred to as time T0.

Microchip Ex. 1018, p. 181
Microchip v. Aptiv
IPR2024-00494

3. The device detects assertion of SE0.

    a) If the device is being reset from suspend, then the device begins a high-speed detection handshake after the detection of SE0 for no less than 2.5 µs (TFILTSE0). Since a suspended device will generally have its clock oscillator disabled, the detection of SE0 will cause the oscillator to be restarted. The clock must be useable (although not necessarily settled to 500 ppm accuracy) in time to detect the high-speed hub chirp as described in Step 8.
    b) If the device is being reset from a non-suspended full-speed state, then the device begins a high-speed detection handshake after the detection of SE0 for no less than 2.5 µs and no more than 3.0 ms (TWTRSTFS).
    c) If the device is being reset from a non-suspended high-speed state, then the device must wait no less than 3.0 ms and no more than 3.125 ms (TWTREV) before reverting to full-speed. Reversion to full-speed is accomplished by removing the high-speed termination and reconnecting the D+ pull-up resistor. The device samples the bus state, and checks for SE0 (reset as opposed to suspend), no less than 100 µs and no more than 875 µs (TWTRSTHS) after starting reversion to full-speed. If SE0 (reset) is detected, then the device begins a high-speed detection handshake.

High-speed Detection Handshake (not performed if low-speed device detected by hub):

Note: In the following handshake, both the hub and device are required to detect Chirp J's and K's of specified minimum durations. It is strongly recommended that "gaps" in these Chirp signals as short as 16 high-speed bit times should restart the duration timers.

4. The high-speed device leaves the D+ pull-up resistor connected, leaves the high-speed terminations disabled, and drives the high-speed signaling current into the D- line. This creates a Chirp K on the bus. The device chirp must last no less than 1.0 ms (TUCH) and must end no more than 7.0 ms (TUCHEND) after high-speed Reset time T0.

5. The hub must detect the device chirp after it has seen assertion of the Chirp K for no less than 2.5 µs (TFILT). If the hub does not detect a device chirp, it must continue the assertion of SE0 until the end of reset.

6. No more than 100 µs (TWTDCH) after the bus leaves the Chirp K state, the hub must begin to send an alternating sequence of Chirp K's and Chirp J's. There must be no Idle states on the bus between the J's and K's. This sequence must continue until a time (TDCHSE0) no more than 500 µs before and no less than 100 µs before the end of Reset. (This will guarantee that the bus remains active, preventing the device from entering the high-speed Suspend state.) Each individual Chirp K and Chirp J must last no less than 40 µs and no more than 60 µs (TDCHBIT).

7. After completing the hub chirp sequence, the hub asserts SE0 until end of Reset. At the end of reset, the hub must transition to the high-speed Enabled state without causing any transitions on the data lines.

8. After the device completes its chirp, it looks for the high-speed hub chirp. At a minimum, the device is required to see the sequence Chirp K-J-K-J-K-J in order to detect a valid hub chirp. Each individual Chirp K and Chirp J must be detected for no less than 2.5 µs (TFILT).

    a) If the device detects the sequence Chirp K-J-K-J-K-J, then no more than 500 µs (TWTHS) after detection, the device is required to disconnect the D+ pull-up resistor, enable the high-speed terminations, and enter the high-speed Default state.
    b) If the device has not detected the sequence Chirp K-J-K-J-K-J by a time no less than 1.0 ms and no more than 2.5 ms (TWTFS) after completing its own chirp, then the device is required to revert to the full-speed Default state and wait for the end of Reset.

7.1.7.6 Suspending

All devices must support the Suspend state. Devices can go into the Suspend state from any powered state. They begin the transition to the Suspend state after they see a constant Idle state on their upstream facing bus lines for more than 3.0 ms. The device must actually be suspended, drawing only suspend current from the bus after no more than 10 ms of bus inactivity on all its ports. Any bus activity on the upstream facing port will keep

Microchip Ex. 1018, p. 182
Microchip v. Aptiv
IPR2024-00494

a device out of the Suspend state. In the absence of any other bus traffic, the SOF token (refer to Section 8.4.3) will occur once per (micro)frame to keep full-/high-speed devices from suspending. In the absence of any low-speed traffic, low-speed devices will see at least one keep-alive (defined in Table 7-2) in every frame in which an SOF occurs, which keeps them from suspending. Hubs generate this keep-alive as described in Section 11.8.4.1.

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

Additional Requirements for High-speed Capable Devices

From the perspective of a device operating in high-speed mode, a Reset and a Suspend are initially indistinguishable, so the first part of the device response is the same as for a Reset. When a device operating in high-speed mode detects that the data lines have been in the high-speed Idle state for at least 3.0 ms, it must revert to the full-speed configuration no later than 3.125 ms (TWTREV) after the start of the idle state. Reversion to full-speed is accomplished by disconnecting its termination resistors and reconnecting its D+ pull-up resistor. No earlier than 100 µs and no later than 875 µs (TWTRSTHS) after reverting to full-speed, the device must sample the state of the line. If the state is a full-speed J, the device continues with the Suspend process. (SE0 would have indicated that the downstream facing port was driving reset, and the device would have gone into the "High-speed Detection Handshake" as described in Section 7.1.7.5.)

A device or downstream facing port which is suspended from high-speed operation actually transitions to full-speed signaling during the suspend process, but is required to remember that it was operating in high-speed mode when suspended. When the resume occurs, the device or downstream facing transceiver must revert to high-speed as discussed in Section 7.1.7.7 without the need for a reset.

7.1.7.6.1 Global Suspend

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

After 3.0 ms of continuous idle state, a downstream facing transceiver operating in high-speed must revert to the full-speed idle configuration (high-speed terminations disabled), but it does not enable full-speed disconnect detection until 1.0 ms later. This is to make sure that the device has returned to the full-speed Idle state prior to the enabling of full-speed disconnect detection, thereby preventing an unintended disconnect detection. After re-enabling the full-speed disconnect detection mechanism, the hub continues with the suspend process.

7.1.7.6.2 Selective Suspend

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

When a downstream facing port operating in high-speed mode receives the SetPortFeature(PORT_SUSPEND) command, the port immediately reverts to the full-speed Idle state and blocks any activity to the suspend segment. Full-speed disconnect detection is disabled until the port has been in full-speed idle for 4.0 ms. This prevents an unintended disconnect detection. After re-enabling the full-speed disconnect detection mechanism, the hub continues with the suspend process.

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

Microchip Ex. 1018, p. 183
Microchip v. Aptiv
IPR2024-00494

7.1.7.7 Resume

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

The host may signal resume (TDRSMDN) at any time. It must send the resume signaling for at least 20 ms and then end the resume signaling in one of two ways, depending on the speed at which its port was operating when it was suspended. If the port was in low-/full-speed when suspended, the resume signaling must be ended with a standard, low-speed EOP (two low-speed bit times of SE0 followed by a J). If the port was operating in high-speed when it was suspended, the resume signaling must be ended with a transition to the high-speed idle state.

The 20 ms of resume signaling ensures that all devices in the network that are enabled to see the resume are awakened. The connectivity established by the resume signaling is torn down by the End of Resume, which prepares the hubs for normal operation. After resuming the bus, the host must begin sending bus traffic (at least the SOF token) within 3 ms of the start of the idle state to keep the system from going back into the Suspend state.

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

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

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

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

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

If the hub port and device were operating in high-speed prior to suspend, they are required to "remember" that they were previously operating in high-speed, and they must transition back to high-speed operation, without arbitration, within two low-speed bit times of the K to SE0 transition. The inactivity timers must be started two low-speed bit times after the K to SE0 transition. Note that the transition from SE0 to J which would normally

Microchip Ex. 1018, p. 184
Microchip v. Aptiv
IPR2024-00494

occur at the end of full-speed resume signaling is omitted if the link was operating in high-speed at the time when it was suspended.

It is required that the host begin sending SOF's in time to prevent the high-speed tree from suspending.

7.1.8 Data Encoding/Decoding

The USB employs NRZI data encoding when transmitting packets. In NRZI encoding, a "1" is represented by no change in level and a "0" is represented by a change in level. Figure 7-31 shows a data stream and the NRZI equivalent. The high level represents the J state on the data lines in this and subsequent figures showing NRZI encoding. A string of zeros causes the NRZI data to toggle each bit time. A string of ones causes long periods with no transitions in the data.

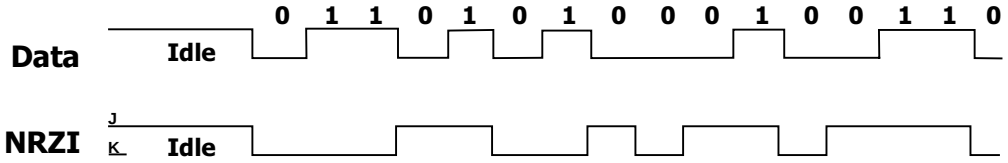


Figure 7-31. NRZI Data Encoding

7.1.9 Bit Stuffing

In order to ensure adequate signal transitions, bit stuffing is employed by the transmitting device when sending a packet on USB (see Figure 7-32 and Figure 7-34). A zero is inserted after every six consecutive ones in the data stream before the data is NRZI encoded, to force a transition in the NRZI data stream. This gives the receiver logic a data transition at least once every seven bit times to guarantee the data and clock lock. Bit stuffing is enabled beginning with the Sync Pattern. The data "one" that ends the Sync Pattern is counted as the first one in a sequence. Bit stuffing by the transmitter is always enforced, except during high-speed EOP. If required by the bit stuffing rules, a zero bit will be inserted even if it is the last bit before the end-of-packet (EOP) signal.

The receiver must decode the NRZI data, recognize the stuffed bits, and discard them.

7.1.9.1 Full-/low-speed

Full-/low-speed signaling uses bit stuffing throughout the packet without exception. If the receiver sees seven consecutive ones anywhere in the packet, then a bit stuffing error has occurred and the packet should be ignored. The time interval just before an EOP is a special case. The last data bit before the EOP can become stretched by hub switching skews. This is known as dribble and can lead to the case illustrated in Figure 7-33, which shows where dribble introduces a sixth bit that does not require a bit stuff. Therefore, the receiver must accept a packet for which there are up to six full bit times at the port with no transitions prior to the EOP.

Data Encoding Sequence:

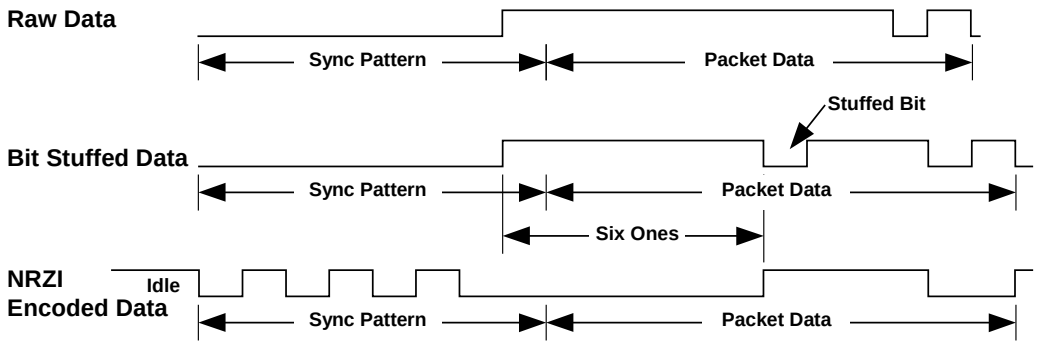


Figure 7-32. Bit Stuffing

Microchip Ex. 1018, p. 185
Microchip v. Aptiv
IPR2024-00494

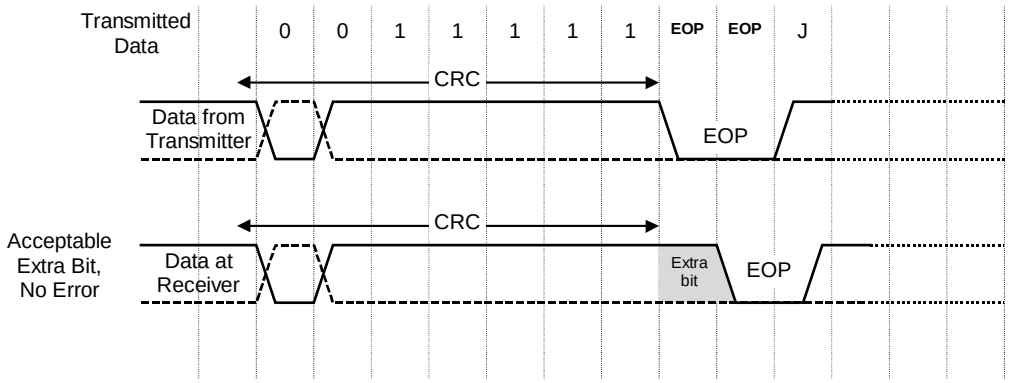


Figure 7-33. Illustration of Extra Bit Preceding EOP (Full-/low-speed)

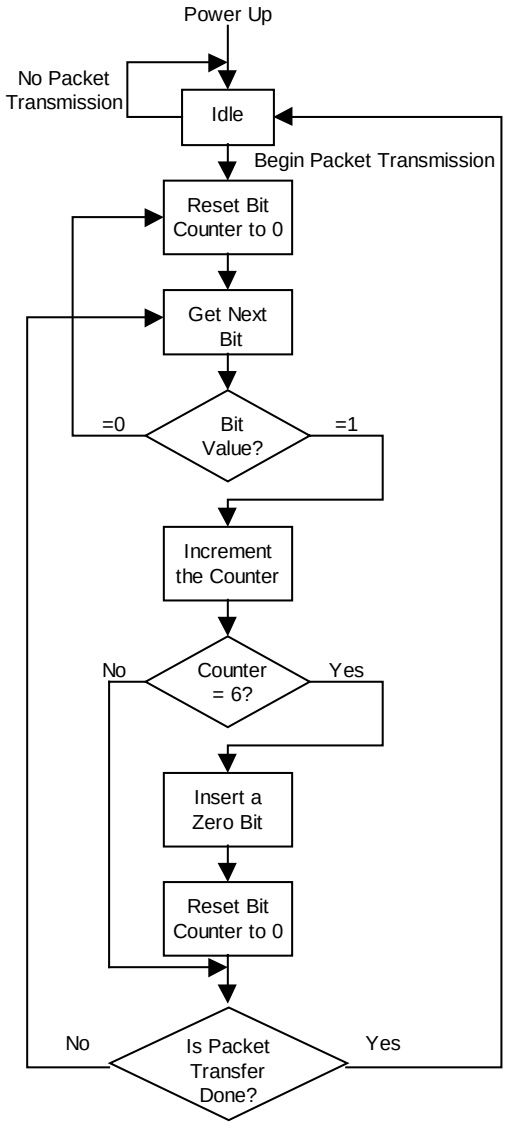


Figure 7-34. Flow Diagram for Bit Stuffing

Microchip Ex. 1018, p. 186
Microchip v. Aptiv
IPR2024-00494

7.1.9.2 High-Speed

High-speed signaling uses bit stuffing throughout the packet, with the exception of the intentional bit stuff errors used in the high-speed EOP as described in Section 7.1.13.2.

7.1.10 Sync Pattern

The SYNC pattern used for low-/full-speed transmission is required to be 3 KJ pairs followed by 2 K's for a total of eight symbols. Figure 7-35 shows the NRZI bit pattern, which is prefixed to each low-/full-speed packet.

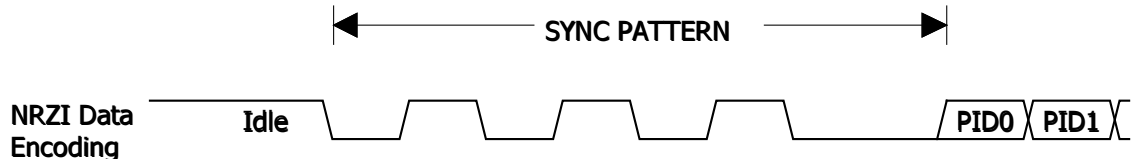


Figure 7-35. Sync Pattern (Low-/full-speed)

The SYNC pattern used for high-speed transmission is required to be 15 KJ pairs followed by 2 K's, for a total of 32 symbols. Hubs are allowed to drop up to 4 bits from the start of the SYNC pattern when repeating packets. Hubs must not corrupt any repeated bits of the SYNC field, however. Thus, after being repeated by 5 hubs, a packet's SYNC field may be as short as 12 bits.

7.1.11 Data Signaling Rate

The high-speed data rate (THSDRAT) is nominally 480.00 Mb/s, with a required bit rate accuracy of ±500 ppm. For hosts, hubs, and high-speed capable functions, the required data-rate accuracy when transmitting at any speed is ±0.05% (500 ppm). The full-speed rate for such hubs and functions is TFDRATHS. The low-speed rate for such hubs is TLDRATHS (a low-speed function must not support high-speed).

The full-speed data rate is nominally 12.000 Mb/s. For full-speed only functions, the required data-rate when transmitting (TFDRATE) is 12.000 Mb/s ±0.25% (2,500 ppm).

The low-speed data rate is nominally 1.50 Mb/s. For low-speed functions, the required data-rate when transmitting (TLDRATE) is 1.50 Mb/s ±1.5% (15,000 ppm). This allows the use of resonators in low cost, low-speed devices.

Hosts and hubs must be able to receive data from any compliant low-speed, full-speed, or high-speed source. High-speed capable functions must be able to receive data from any compliant full-speed or high-speed source. Full-speed only functions must be able to receive data from any compliant full-speed source. Low-speed only functions must be able to receive data from any compliant low-speed source.

The above accuracy numbers include contributions from all sources:

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

7.1.12 Frame Interval

The USB defines a frame interval (TFRAME) to be 1.000 ms ±500 ns long. The USB defines a microframe interval (THSFRAM) to be 125.0 µs ±62.5 ns long. The (micro)frame interval is measured from any point in an SOF token in one (micro)frame to the same point in the SOF token of the next (micro)frame.

Microchip Ex. 1018, p. 187
Microchip v. Aptiv
IPR2024-00494

Since the Host Controller and hubs must meet clock accuracy specification of ±0.05%, they will automatically meet the frame interval requirements without the need for adjustment.

The frame interval repeatability, TRFI (difference in frame interval between two successive frames), must be less than 0.5 full-speed bit times. The microframe interval repeatability, THSRFI (difference in the microframe interval between two successive microframes, measured at the host), must be less than 4 high-speed bit times. Each hub may introduce at most 4 additional high-speed bits of microframe jitter.

Hubs and certain full-/high-speed functions need to track the (micro)frame interval. They also are required to have sufficient frame timing adjustment to compensate for their own frequency inaccuracy.

7.1.13 Data Source Signaling

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

7.1.13.1 Data Source Jitter

This section describes the maximum allowable data source jitter for low-speed, full-speed, and high-speed signaling.

7.1.13.1.1 Low-/full-speed Data Source Jitter

The source of data can have some variation (jitter) in the timing of edges of the data transmitted. The time between any set of data transitions is (N * TPERIOD) ± jitter time, where 'N' is the number of bits between the transitions. The data jitter is measured with the same load used for maximum rise and fall times and is measured at the crossover points of the data lines, as shown in Figure 7-36.

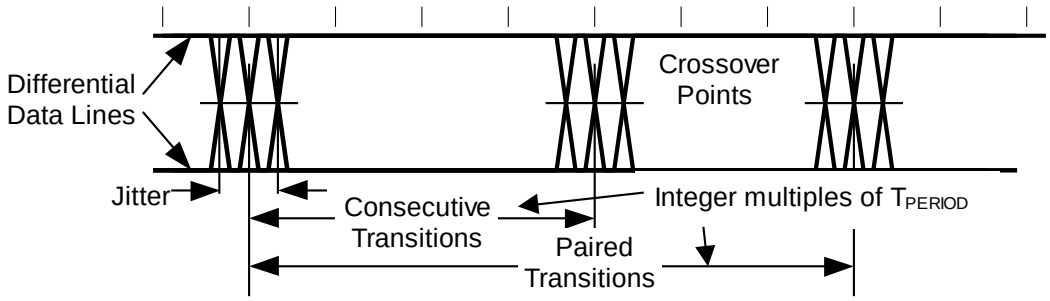


Figure 7-36. Data Jitter Taxonomy

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

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

7.1.13.1.2 High-speed Data Source Jitter

High-speed data within a single packet must be transmitted with no more jitter than is allowed by the eye patterns defined in Section 7.1.2 when measured over a sliding window of 480 high-speed bit times.

Microchip Ex. 1018, p. 188
Microchip v. Aptiv
IPR2024-00494

7.1.13.2 EOP Width

This section describes low-speed, full-speed, and high-speed EOP width.

7.1.13.2.1 Low-/full-speed EOP

The width of the SE0 in the EOP is approximately 2 * TPERIOD. The SE0 width is measured with the same load used for maximum rise and fall times and is measured at the same level as the differential signal crossover points of the data lines (see Figure 7-37).

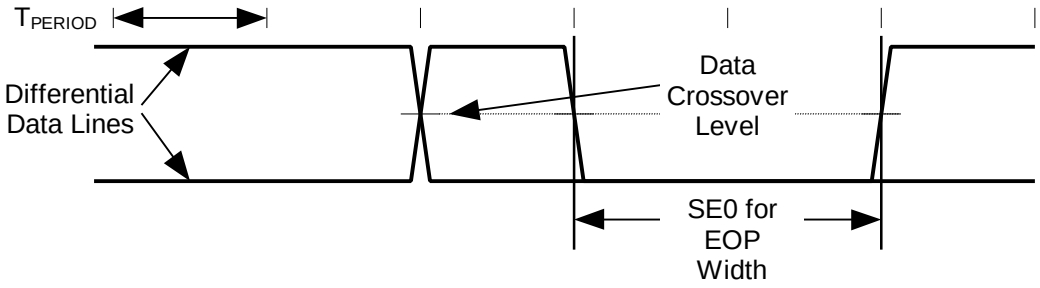


Figure 7-37. SE0 for EOP Width Timing

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

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

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

- A full-speed EOP may have the SE0 interval reduced to as little as 82 ns (TFEOPR) and a low-speed SE0 interval may be as short as 670 ns (TLEOPR).

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

7.1.13.2.2 High-speed EOP

In high-speed signaling, a bit stuff error is intentionally generated to indicate EOP. A receiver is required to interpret any bit stuff error as an EOP.

For high-speed packets other than SOF's, the transmitted EOP delimiter is required to be an NRZ byte of 01111111 without bit stuffing. For example, if the last symbol prior to the EOP field is a J, this would lead to an EOP of KKKKKKKK.

For high-speed SOF's, the transmitted EOP delimiter is required to be 5 NRZ bytes without bit stuffing, consisting of 01111111 11111111 11111111 11111111 11111111. Thus if the last bit prior to the EOP field is a J, this would lead to 40 K's on the wire, at the end of which the lines must return to the high-speed Idle state. This extra EOP length is of no significance to a receiver; it is used for disconnect detection as discussed in Section 7.1.7.3.

A hub may add at most 4 random bits to the end of the EOP field when repeating a packet. Thus after 5 repeaters, a packet can have up to 20 random bits following the EOP field. A hub, however, must not corrupt any of the 8 (or 40 in the case of a SOF) required bits of the EOP field.

Microchip Ex. 1018, p. 189
Microchip v. Aptiv
IPR2024-00494

7.1.14 Hub Signaling Timings

This section describes low-speed, full-speed, and high-speed hub signaling timings.

7.1.14.1 Low-/full-speed Hub Signaling Timings

The propagation of a full-speed, differential data signal through a hub is shown in Figure 7-38. The downstream signaling is measured without a cable connected to the port and with the load used for measuring rise and fall times. The total delay through the upstream cable and hub electronics must be a maximum of 70 ns (T_{HDD1}). If the hub has a detachable USB cable, then the delay (T_{HDD2}) through hub electronics and the associated transmission line must be a maximum of 44 ns to allow for a maximum cable delay of 26 ns (T_{FSCBL}). The delay through this hub is measured in both the upstream and downstream directions, as shown in Figure 7-38B, from data line crossover at the input port to data line crossover at the output port.

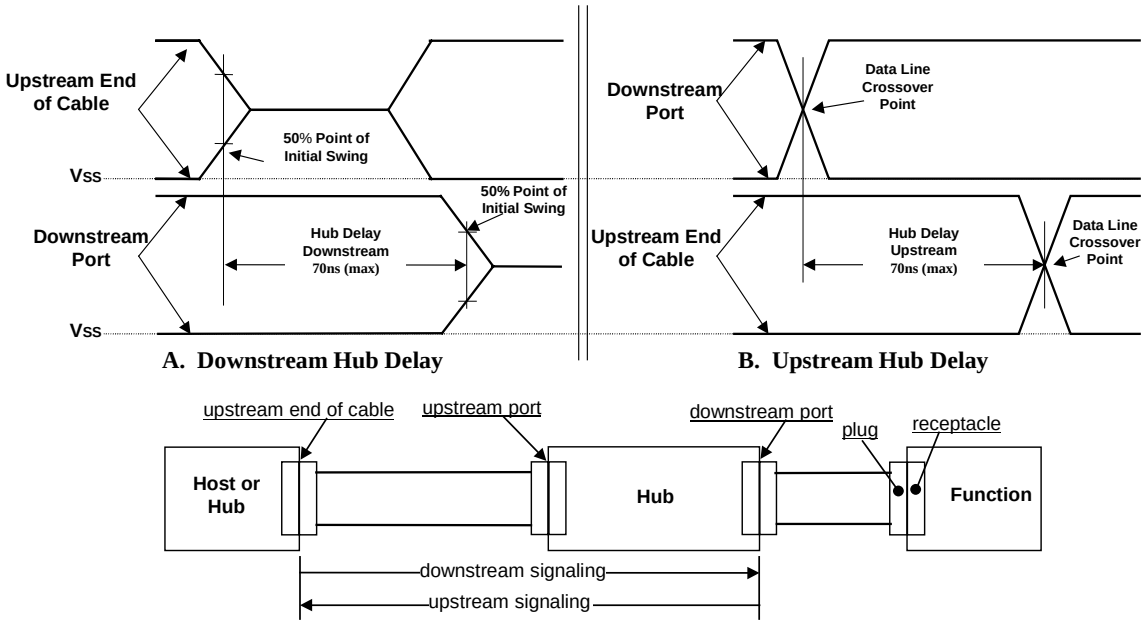


Figure 7-38. Hub Propagation Delay of Full-speed Differential Signals

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

When the hub acts as a repeater, it must reproduce the received, full-speed signal accurately on its outputs. This means that for differential signals, the propagation delays of a J-to-K state transition must match closely to the delays of a K-to-J state transition. For full-speed propagation, the maximum difference allowed between these two delays (T_{HDJ1}) (see Figure 7-38 and Figure 7-52) for a hub plus cable is ±3.0 ns. Similarly, the difference in delay between any two J-to-K or K-to-J transitions through a hub (T_{HDJ2}) must be less than ±1.0 ns. For low-speed propagation in the downstream direction, the corresponding allowable jitter (T_{LDHJ1}) is ±45 ns and (T_{LDHJ2}) ±15 ns, respectively. For low-speed propagation in the upstream direction, the allowable jitter is ±45 ns in both cases (T_{LUHJ1} and T_{LUHJ2}).

An exception to this case is the skew that can be introduced in the Idle-to-K state transition at SOP (T_{FSOP} and T_{LSOP}) (refer to Section 7.1.7.4). In this case, the delay to the opposite port includes the time to enable the output buffer. However, the delays should be closely matched to the normal hub delay and the maximum

Microchip Ex. 1018, p. 190
Microchip v. Aptiv
IPR2024-00494

additional delay difference over a normal J-to-K transition is ±5.0 ns. This limits the maximum distortion of the first bit in the packet.

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

The EOP must be propagated through a hub in the same way as the differential signaling. The propagation delay for sensing an SE0 must be no less than the greater of the J-to-K or K-to-J differential data delay (to avoid truncating the last data bit in a packet), but not more than 15 ns greater than the larger of these differential delays at full-speed and 200 ns at low-speed (to prevent creating a bit stuff error at the end of the packet). EOP delays are shown in Figure 7-53.

Because the sense levels for the SE0 state are not at the midpoint of the signal swing, the width of SE0 state will be changed as it passes through each hub. A hub may not change the width of the SE0 state in a full-speed EOP by more than ±15 ns (T_{FHESK}), as measured by the difference of the leading edge and trailing edge delays of the SE0 state (see Figure 7-53). An SE0 from a low-speed device has long rise and fall times and is subject to greater skew, but these conditions exist only on the cable from the low-speed device to the port to which it is connected. Thereafter, the signaling uses full-speed buffers and their faster rise and fall times. The SE0 from the low-speed device cannot be changed by more than ±300 ns (T_{LHESK}) as it passes through the hub to which the device is connected. This time allows for some signal conditioning in the low-speed transceiver to reduce its sensitivity to noise.

7.1.14.2 High-speed Hub Signaling Timings

When a hub acts as a repeater for high-speed data, the delay of the hub (T_{HSHDD}) must not exceed 36 high-speed bit times plus 4 ns (the trace delays allowed for the hub circuit board). This delay is measured from the last bit of the SYNC field at the input connector to the last bit of the SYNC field at the output connector.

A high-speed hub repeater must digitally resynchronize the buffered data, so there is no allowance for cumulative jitter (within a single packet) as a high-speed packet passes through multiple repeater stages. Within a single packet, the jitter must not exceed the eye pattern templates defined in Section 7.1.2 over a sliding window of 480 high-speed bit times.

Due to the data synchronization process, the propagation delay of a hub repeater is allowed to vary at most 5 high-speed bit times (T_{HSHDV}). The delay including this allowed variation must not exceed 36 high-speed bit times plus 4 ns. (This allows for some uncertainty as to when an incoming packet arrives at the hub with respect to the phase of the synchronization clock.)

Microchip Ex. 1018, p. 191
Microchip v. Aptiv
IPR2024-00494

7.1.15 Receiver Data Jitter

This section describes low-speed, full-speed, and high-speed receiver data jitter.

7.1.15.1 Low-/full-speed Receiver Data Jitter

The data receivers for all types of devices must be able to properly decode the differential data in the presence of jitter. The more of the bit cell that any data edge can occupy and still be decoded, the more reliable the data transfer will be. Data receivers are required to decode differential data transitions that occur in a window plus and minus a nominal quarter bit cell from the nominal (centered) data edge position. (A simple 4X over-sampling state machine DPLL can be built that satisfies these requirements.) This requirement is derived in Table 7-4 and Table 7-5. The tables assume a worst-case topology of five hubs between the host and device and the worst-case number of seven bits between transitions. The derived numbers are rounded up for ease of specification.

Jitter will be caused by the delay mismatches discussed above and by mismatches in the source and destination data rates (frequencies). The receive data jitter budgets for full- and low-speed are given in Table 7-4 and Table 7-5. These tables give the value and totals for each source of jitter for both consecutive (next) and paired transitions. Note that the jitter component related to the source or destination frequency tolerance has been allocated to the appropriate device (i.e., the source jitter includes bit shifts due to source frequency inaccuracy over the worst-case data transition interval). The output driver jitter can be traded off against the device clock accuracy in a particular implementation as long as the jitter specification is met.

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

Table 7-4. Full-speed Jitter Budget

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

Microchip Ex. 1018, p. 192
Microchip v. Aptiv
IPR2024-00494

Table 7-5. Low-speed Jitter Budget

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

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

7.1.15.2 High-speed Receiver Data Jitter

A high-speed capable receiver must reliably recover high-speed data when the waveforms at its inputs conform to the receiver sensitivity eye pattern templates. The templates, which are called out in Section 7.1.2.2, specify the horizontal and vertical eye pattern opening over a 480 bit time sliding window over the duration of a packet. Thus, for example, a high-speed receiver within a function must reliably recover data with a peak to peak jitter of 30%, measured at its B receptacle (as described by Template 4).

Such conformance is tested using Test Mode Test_Packet, as defined in Section 7.1.20.

It is a recommended design guideline that a receiver's BER should be <= 10^{-12} when the receiver sensitivity requirement is met.

7.1.16 Cable Delay

The maximum total one-way signal propagation delay allowed is 30 ns. The allocation for cable delay is 26 ns. A maximum delay of 3 ns is allowed from a Host or Hub Controller downstream facing transceiver to its exterior downstream facing connector, while a maximum delay of 1 ns is allowed from the upstream facing connector to the upstream facing transceiver of any device. For a standard USB detachable cable, the cable

Microchip Ex. 1018, p. 193
Microchip v. Aptiv
IPR2024-00494

delay is measured from the Series A connector pins to the Series B connector pins and is no more than 26 ns. For other cables, the delay is measured from the series A connector to the point where the cable is connected to the device. The cable delay must also be less than 5.2 ns per meter.

The maximum one-way data delay on a full-speed cable is measured as shown in Figure 7-39.

One-way cable delay for low-speed cables must be less than 18 ns. It is measured as shown in Figure 7-40.

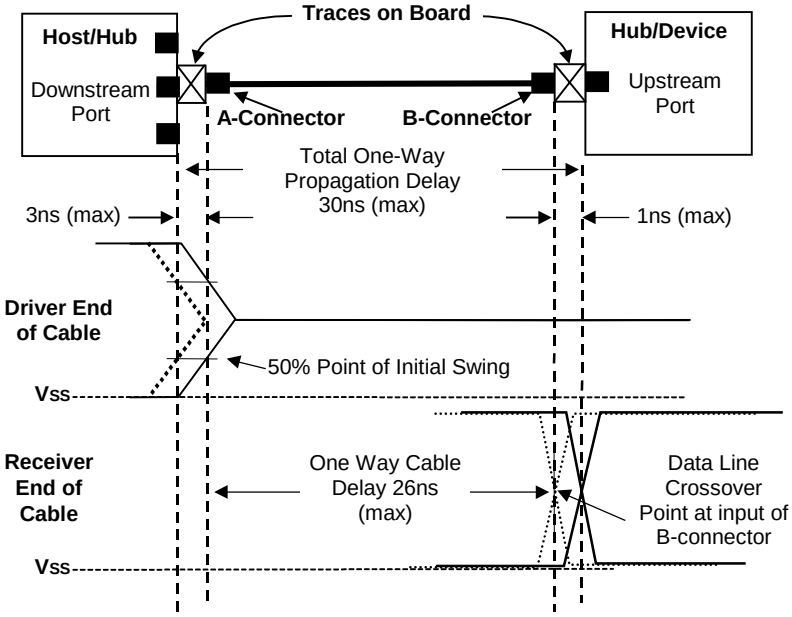
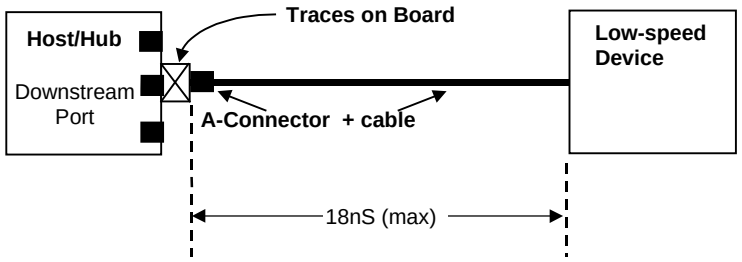


Figure 7-39. Full-speed Cable Delay



One-way Propagation Delay

Figure 7-40. Low-speed Cable Delay

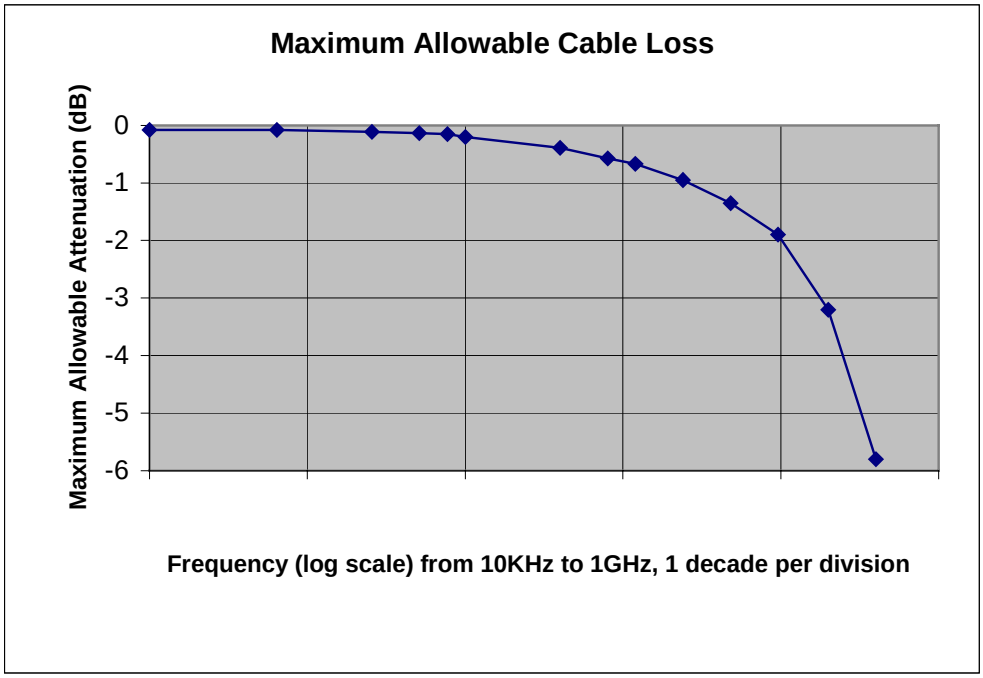
Microchip Ex. 1018, p. 194
Microchip v. Aptiv
IPR2024-00494

7.1.17 Cable Attenuation

USB cables must not exceed the loss figures shown in Table 7-6. Between the frequencies called out in the table, the cable loss should be no more than is shown in the accompanying graph.

Table 7-6. Maximum Allowable Cable Loss

Frequency (MHz)	Attenuation (maximum) dB/cable
0.064	0.08
0.256	0.11
0.512	0.13
0.772	0.15
1.000	0.20
4.000	0.39
8.000	0.57
12.000	0.67
24.000	0.95
48.000	1.35
96.000	1.9
200.00	3.2
400.00	5.8



Microchip Ex. 1018, p. 195
Microchip v. Aptiv
IPR2024-00494

7.1.18 Bus Turn-around Time and Inter-packet Delay

This section describes low-speed, full-speed, and high-speed bus turn-around time and inter-packet delay.

7.1.18.1 Low-/Full-Speed Bus Turn-around Time and Inter-packet Delay

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

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

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

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

7.1.18.2 High-Speed Bus Turn-around Time and Inter-packet Delay

High-speed inter-packet delays are measured from time when the line returns to a high-speed Idle State at the end of one packet to when the line leaves the high-speed Idle State at the start of the next packet.

When transmitting after receiving a packet, hosts and devices must provide an inter-packet delay of at least 8 bit times (THSIPDOD) measured at their A or B connectors (receptacles or plugs).

Additionally, if a host is transmitting two packets in a row, the inter-packet delay must be a minimum of 88 bit times (THSIPDSD), measured at the host's A receptacle. This will guarantee an inter-packet delay of at least 32 bit times at all devices (when receiving back to back packets). The maximum inter-packet delay provided by a host is 192 bit times within a transaction (THSRSPIPD1) measured at the A receptacle. When a host responds to a packet from a device, it will provide an inter-packet delay of at most 192 bit times measured at the A receptacle. There is no maximum inter-packet delay between packets in unrelated transactions.

When a device with a detachable cable responds to a packet from a host, it will provide an inter-packet delay of at most 192 bit times measured at the B receptacle. If the device has a captive cable, it will provide an inter-packet delay of at most 192 bit times plus 52 ns (2 times the max cable length) measured at the cable's A plug (THSRSPIPD2).

7.1.19 Maximum End-to-end Signal Delay

This section describes low-speed, full-speed, and high-speed end-to-end delay.

7.1.19.1 Low-/full-speed End-to-end Signal Delay

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

Microchip Ex. 1018, p. 196
Microchip v. Aptiv
IPR2024-00494

Figure 7-41 depicts the configuration of six signal hops (cables) that results in allowable worst-case signal delay. The maximum propagation delay from the upstream end of a hub's cable to any downstream facing connector on that hub is 70 ns.

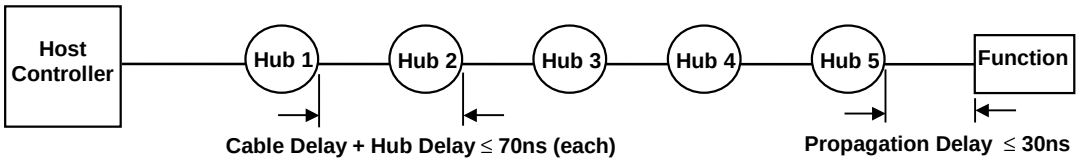


Figure 7-41. Worst-case End-to-end Signal Delay Model for Low-/full-speed

7.1.19.2 High-Speed End-to-end Delay

A high-speed host or device expecting a response to a transmission must not timeout the transaction if the inter-packet delay is less than 736 bit times, and it must timeout the transaction if no signaling is seen within 816 bit times.

These timeout limits allow a response to be seen even for the worst-case round trip signal delay. In high-speed mode, the worst-case round trip signal delay model is the sum of the following components:

12 max length cable delays (6 cables)	= 312 ns	
10 max delay hubs (5 hubs)	= 40 ns +	360 bit times
1 max device response time	=	192 bit times
Worst-case round trip delay	= 352 ns +	552 bit times = 721 bit times

7.1.20 Test Mode Support

To facilitate compliance testing, host controllers, hubs, and high-speed capable functions must support the following test modes:

- Test mode Test_SE0_NAK: Upon command, a port's transceiver must enter the high-speed receive mode and remain in that mode until the exit action is taken. This enables the testing of output impedance, low level output voltage, and loading characteristics. In addition, while in this mode, upstream facing ports (and only upstream facing ports) must respond to any IN token packet with a NAK handshake (only if the packet CRC is determined to be correct) within the normal allowed device response time. This enables testing of the device squelch level circuitry and, additionally, provides a general purpose stimulus/response test for basic functional testing.
- Test mode Test_J: Upon command, a port's transceiver must enter the high-speed J state and remain in that state until the exit action is taken. This enables the testing of the high output drive level on the D+ line.
- Test mode Test_K: Upon command, a port's transceiver must enter the high-speed K state and remain in that state until the exit action is taken. This enables the testing of the high output drive level on the D- line.
- Test mode Test_Packet: Upon command, a port must repetitively transmit the following test packet until the exit action is taken. This enables the testing of rise and fall times, eye patterns, jitter, and any other dynamic waveform specifications.

  The test packet is made up by concatenating the following strings. (Note: For J/K NRZI data, and for NRZ data, the bit on the left is the first one transmitted. "S" indicates that a bit stuff occurs, which inserts an "extra" NRZI data bit. "* N" is used to indicate N occurrences of a string of bits or symbols.)

Microchip Ex. 1018, p. 197
Microchip v. Aptiv
IPR2024-00494

NRZI Symbols (Fields)	NRZ Bit Strings	Number of NRZ Bits
{KJ * 15}, KK (SYNC)	{00000000 * 3}, 00000001	32
KKJKJKKK (DATA0 PID)	11000011	8
JKJKJKJK * 9	00000000 * 9	72
JJKKJJKK * 8	01010101 * 8	64
JJJJKKKK * 8	01110111 * 8	64
JJJJJJKKKKKKK * 8	0, {111111S *15}, 111111	97
JJJJJJK * 8	S, 111111S, {0111111S * 7}	55
{JKKKKKKK * 10}, JK	00111111, {S0111111 * 9}, S0	72
JJJKKKJJKKKKJKKK (CRC16)	0110110101110011	16
JJJJJJJ (EOP)	01111111	8

A port in Test_Packet mode must send this packet repetitively. The inter-packet timing must be no less than the minimum allowable inter-packet gap as defined in Section 7.1.18 and no greater than 125 µs.

- Test mode Test_Force_Enable: Upon command, downstream facing hub ports (and only downstream facing hub ports) must be enabled in high-speed mode, even if there is no device attached. Packets arriving at the hub's upstream facing port must be repeated on the port which is in this test mode. This enables testing of the hub's disconnect detection; the disconnect detect bit can be polled while varying the loading on the port, allowing the disconnect detection threshold voltage to be measured.

Test Mode Entry and Exit

Test mode of a port is entered by using a device specific standard request (for an upstream facing port) or a port specific hub class request (for a downstream facing port). The device standard request SetFeature(TEST_MODE) is defined in Section 9.4.9. The hub class request SetPortFeature(PORT_TEST) is defined in Section 11.24.2.13. All high-speed capable devices/hubs must support these requests. These requests are not supported for non-high-speed devices.

The transition to test mode must be complete no later than 3 ms after the completion of the status stage of the request.

For an upstream facing port, the exit action is to power cycle the device. For a downstream facing port, the exit action is to reset the hub, as defined in Section 11.24.2.13.

Microchip Ex. 1018, p. 198
Microchip v. Aptiv
IPR2024-00494

7.2 Power Distribution

This section describes the USB power distribution specification.

7.2.1 Classes of Devices

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

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

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

No device shall supply (source) current on VBUS at its upstream facing port at any time. From VBUS on its upstream facing port, a device may only draw (sink) current. They may not provide power to the pull-up resistor on D+/D- unless VBUS is present (see Section 7.1.5). When VBUS is removed, the device must remove power from the D+/D- pull-up resistor within 10 seconds. On power-up, a device needs to ensure that its upstream facing port is not driving the bus, so that the device is able to receive the reset signaling. Devices must also ensure that the maximum operating current drawn by a device is one unit load, until configured. Any device that draws power from the bus must be able to detect lack of activity on the bus, enter the Suspend state, and reduce its current consumption from VBUS (refer to Section 7.2.3 and Section 9.2.5.1).

Microchip Ex. 1018, p. 199
Microchip v. Aptiv
IPR2024-00494

7.2.1.1 Bus-powered Hubs

Bus-powered hub power requirements can be met with a power control circuit such as the one shown in Figure 7-42. Bus-powered hubs often contain at least one non-removable function. Power is always available to the hub's controller, which permits host access to power management and other configuration registers during the enumeration process. A non-removable function(s) may require that its power be switched, so that upon power-up, the entire device (hub and non-removable functions) draws no more than one unit load. Power switching on any non-removable function may be implemented either by removing its power or by shutting off the clock. Switching on the non-removable function is not required if the aggregate power drawn by it and the Hub Controller is less than one unit load. However, as long as the hub port associated with the function is in the Power-off state, the function must be logically reset and the device must appear to be not connected. The total current drawn by a bus-powered device is the sum of the current to the Hub Controller, any non-removable function(s), and the downstream facing ports.

Figure 7-42 shows the partitioning of power based upon the maximum current draw (from upstream) of five unit loads: one unit load for the Hub Controller and the non-removable function and one unit load for each of the external downstream facing ports. If more than four external ports are required, then the hub will need to be self-powered. If the non-removable function(s) and Hub Controller draw more than one unit load, then the number of external ports must be appropriately reduced. Power control to a bus-powered hub may require a regulator. If present, the regulator is always enabled to supply the Hub Controller. The regulator can also power the non-removable functions(s). Inrush current limiting must also be incorporated into the regulator subsystem.

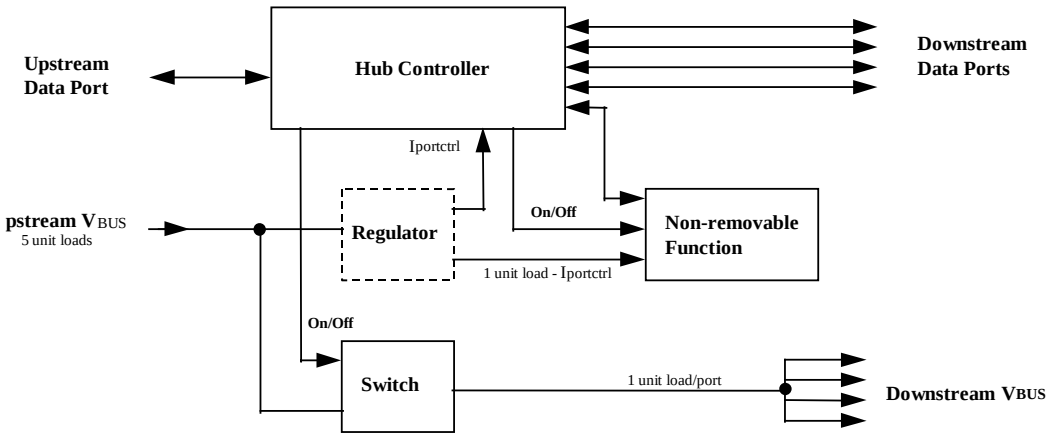


Figure 7-42. Compound Bus-powered Hub

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

7.2.1.2 Self-powered Hubs

Self-powered hubs have a local power supply that furnishes power to any non-removable functions and to all downstream facing ports, as shown in Figure 7-43. Power for the Hub Controller, however, may be supplied from the upstream VBUS (a "hybrid" powered hub) or the local power supply. The advantage of supplying the Hub Controller from the upstream supply is that communication from the host is possible even if the device's power supply remains off. This makes it possible to differentiate between a disconnected and an unpowered device. If the hub draws power for its upstream facing port from VBUS, it may not draw more than one unit load.

Microchip Ex. 1018, p. 200
Microchip v. Aptiv
IPR2024-00494

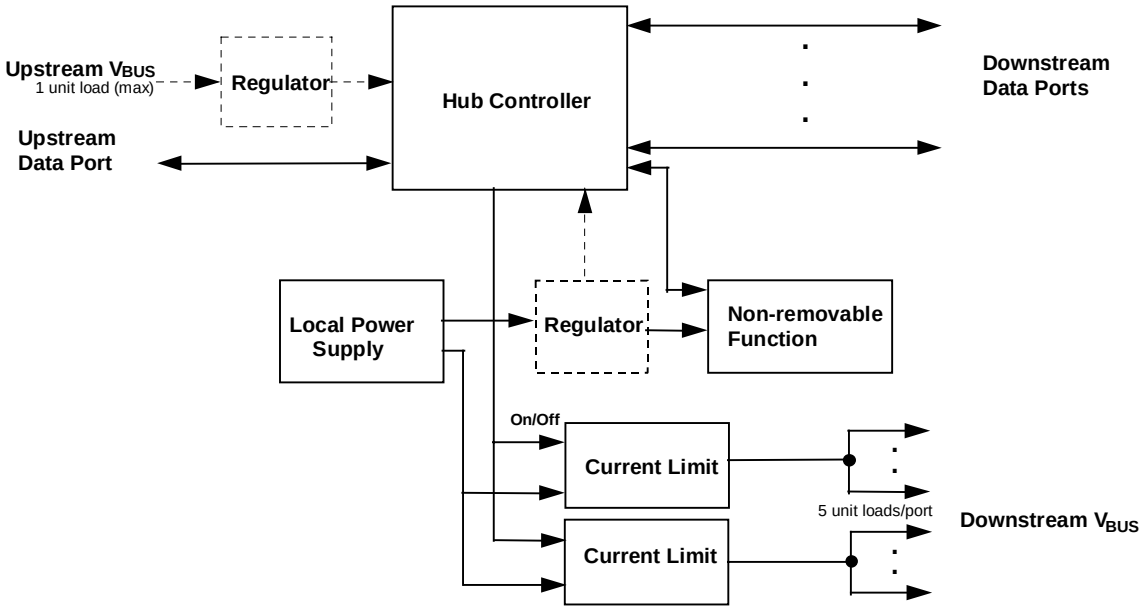


Figure 7-43. Compound Self-powered Hub

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

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

7.2.1.2.1 Over-current Protection

The host and all self-powered hubs must implement over-current protection for safety reasons, and the hub must have a way to detect the over-current condition and report it to the USB software. Should the aggregate current drawn by a gang of downstream facing ports exceed a preset value, the over-current protection circuit removes or reduces power from all affected downstream facing ports. The over-current condition is reported through the hub to Host Controller, as described in Section 11.12.5. The preset value cannot exceed 5.0 A and must be sufficiently above the maximum allowable port current such that transient currents (e.g., during power up or dynamic attach or reconfiguration) do not trip the over-current protector. If an over-current condition occurs on any port, subsequent operation of the USB is not guaranteed, and once the condition is removed, it may be necessary to reinitialize the bus as would be done upon power-up. The over-current limiting mechanism must be resettable without user mechanical intervention. Polymeric PTCs and solid-state switches are examples of methods, which can be used for over-current limiting.

7.2.1.3 Low-power Bus-powered Functions

A low-power function is one that draws up to one unit load from the USB cable when operational. Figure 7-44 shows a typical bus-powered, low-power function, such as a mouse. Low-power regulation can be integrated into the function silicon. Low-power functions must be capable of operating with input VBUS voltages as low as 4.40 V, measured at the plug end of the cable.

Microchip Ex. 1018, p. 201
Microchip v. Aptiv
IPR2024-00494

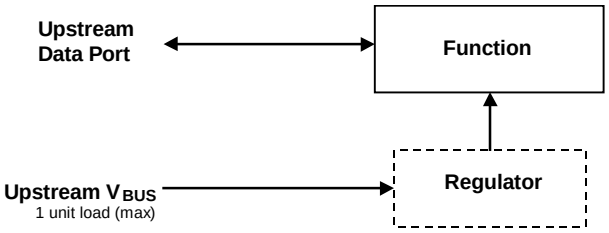


Figure 7-44. Low-power Bus-powered Function

7.2.1.4 High-power Bus-powered Functions

A function is defined as being high-power if, when fully powered, it draws over one but no more than five unit loads from the USB cable. A high-power function requires staged switching of power. It must first come up in a reduced power state of less than one unit load. At bus enumeration time, its total power requirements are obtained and compared against the available power budget. If sufficient power exists, the remainder of the function may be powered on. A typical high-power function is shown in Figure 7-45. The function's electronics have been partitioned into two sections. The function controller contains the minimum amount of circuitry necessary to permit enumeration and power budgeting. The remainder of the function resides in the function block. High-power functions must be capable of operating in their low-power (one unit load) mode with an input voltage as low as 4.40 V, so that it may be detected and enumerated even when plugged into a bus-powered hub. They must also be capable of operating at full power (up to five unit loads) with a VBUS voltage of 4.75 V, measured at the upstream plug end of the cable.

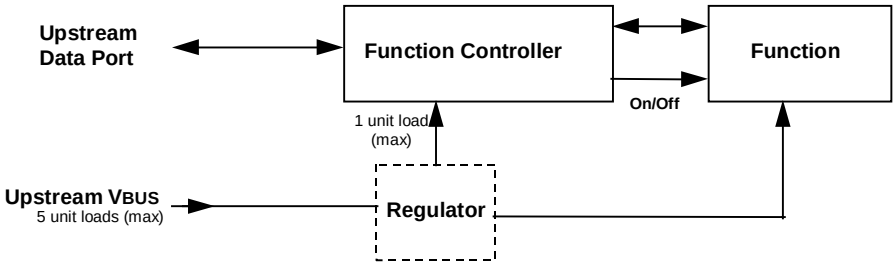


Figure 7-45. High-power Bus-powered Function

7.2.1.5 Self-powered Functions

Figure 7-46 shows a typical self-powered function. The function controller is powered either from the upstream bus via a low-power regulator or from the local power supply. The advantage of the former scheme is that it permits detection and enumeration of a self-powered function whose local power supply is turned off. The maximum upstream power that the function controller can draw is one unit load, and the regulator block must implement inrush current limiting. The amount of power that the function block may draw is limited only by the local power supply. Because the local power supply is not required to power any downstream bus ports, it does not need to implement current limiting, soft start, or power switching.

Microchip Ex. 1018, p. 202
Microchip v. Aptiv
IPR2024-00494

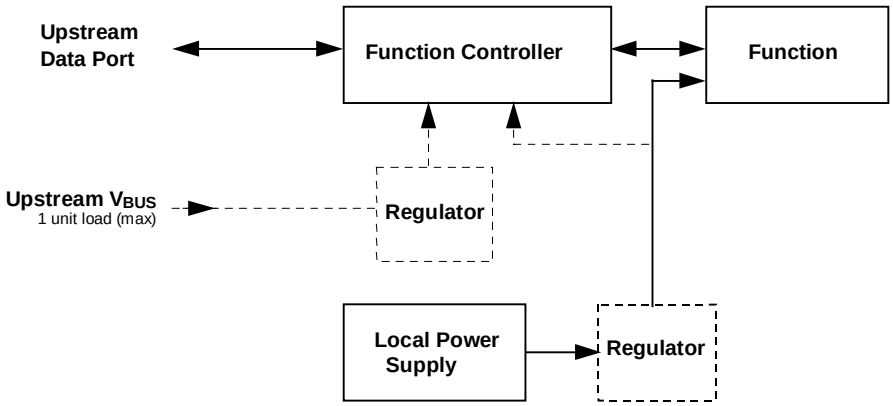


Figure 7-46. Self-powered Function

7.2.2 Voltage Drop Budget

The voltage drop budget is determined from the following:

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

Figure 7-47 shows the minimum allowable voltages in a worst-case topology consisting of a bus-powered hub driving a bus-powered function.

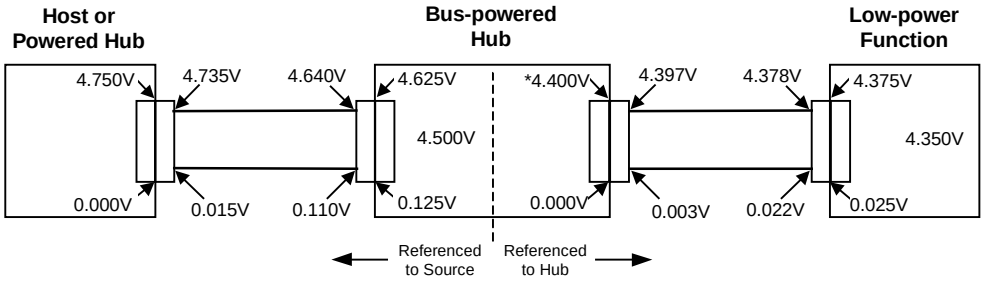


Figure 7-47. Worst-case Voltage Drop Topology (Steady State)

Microchip Ex. 1018, p. 203
Microchip v. Aptiv
IPR2024-00494

7.2.3 Power Control During Suspend/Resume

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

While in the Suspend state, a device may briefly draw more than the average current. The amplitude of the current spike cannot exceed the device power allocation 100 mA (or 500 mA). A maximum of 1.0 second is allowed for an averaging interval. The average current cannot exceed the average suspend current limit (ICCSH or ICCSL, see Table 7-7) during any 1.0-second interval (TSUSAVG1). The profile of the current spike is restricted so the transient response of the power supply (which may be an efficient, low-capacity, trickle power supply) is not overwhelmed. The rising edge of the current spike must be no more than 100 mA/µs. Downstream facing ports must be able to absorb the 500 mA peak current spike and meet the voltage droop requirements defined for inrush current during dynamic attach (see Section 7.2.4.1). Figure 7-48 illustrates a typical example profile for an averaging interval. If the supply to the pull-up resistor on D+/D- is derived from VBUS, then the suspend current will never go to zero because the pull-up and pull-down resistors will always draw power.

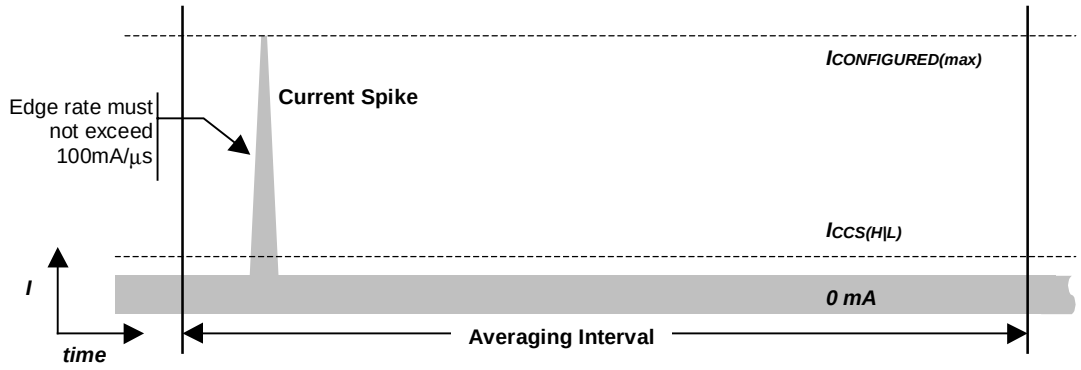


Figure 7-48. Typical Suspend Current Averaging Profile

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

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

Microchip Ex. 1018, p. 204
Microchip v. Aptiv
IPR2024-00494

7.2.4 Dynamic Attach and Detach

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

7.2.4.1 Inrush Current Limiting

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

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

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

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

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

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

7.2.4.2 Dynamic Detach

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

Microchip Ex. 1018, p. 205
Microchip v. Aptiv
IPR2024-00494

inductive flyback on the open end of the cable does not cause the voltage on the device end to reverse polarity. A minimum of 1.0 µF is recommended for bypass across V_{BUS}.

7.3 Physical Layer

The physical layer specifications are described in the following subsections.

7.3.1 Regulatory Requirements

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

7.3.2 Bus Timing/Electrical Characteristics

Table 7-7. DC Electrical Characteristics

Parameter	Symbol	Conditions	Min.	Max.	Units
Supply Voltage:					
High-power Port	V_{BUS}	Note 2, Section 7.2.1	4.75	5.25	V
Low-power Port	V_{BUS}	Note 2, Section 7.2.1	4.40	5.25	V
Supply Current:					
High-power Hub Port (out)	I_{CCPRT}	Section 7.2.1	500		mA
Low-power Hub Port (out)	I_{CCUPT}	Section 7.2.1	100		mA
High-power Function (in)	I_{CCHPF}	Section 7.2.1		500	mA
Low-power Function (in)	I_{CCLPF}	Section 7.2.1		100	mA
Unconfigured Function/Hub (in)	I_{CCINIT}	Section 7.2.1.4		100	mA
Suspended High-power Device	I_{CCSH}	Section 7.2.3; Note 15		2.5	mA
Suspended Low-power Device	I_{CCSL}	Section 7.2.3		500	µA
Input Levels for Low-/full-speed:					
High (driven)	V_{IH}	Note 4, Section 7.1.4	2.0		V
High (floating)	V_{IHZ}	Note 4, Section 7.1.4	2.7	3.6	V
Low	V_{IL}	Note 4, Section 7.1.4		0.8	V
Differential Input Sensitivity	V_{DI}	|(D+)-(D-)|; Figure 7-19; Note 4	0.2		V
Differential Common Mode Range	V_{CM}	Includes V_{DI} range; Figure 7-19; Note 4	0.8	2.5	V
Input Levels for High-speed:					
High-speed squelch detection threshold (differential signal amplitude)	V_{HSSQ}	Section 7.1.7.2 (specification refers to differential signal amplitude)	100	150	mV

Microchip Ex. 1018, p. 206
Microchip v. Aptiv
IPR2024-00494

Table 7-7. DC Electrical Characteristics (Continued)

Parameter	Symbol	Conditions	Min.	Max.	Units
High speed disconnect detection threshold (differential signal amplitude)	V_{HSDSC}	Section 7.1.7.2 (specification refers to differential signal amplitude)	525	625	mV
High-speed differential input signaling levels		Section 7.1.7.2 Specified by eye pattern templates			
High-speed data signaling common mode voltage range (guideline for receiver)	V_{HSCM}	Section 7.1.4.2	-50	500	mV
Output Levels for Low-/full-speed:					
Low	V_{OL}	Note 4, 5, Section 7.1.1	0.0	0.3	V
High (Driven)	V_{OH}	Note 4, 6, Section 7.1.1	2.8	3.6	V
SE1	V_{OSE1}	Section 7.1.1	0.8		V
Output Signal Crossover Voltage	V_{CRS}	Measured as in Figure 7-8; Note 10	1.3	2.0	V
Output Levels for High-speed:					
High-speed idle level	V_{HSOI}	Section 7.1.7.2	-10.0	10.0	mV
High-speed data signaling high	V_{HSOH}	Section 7.1.7.2	360	440	mV
High-speed data signaling low	V_{HSOL}	Section 7.1.7.2	-10.0	10.0	mV
Chirp J level (differential voltage)	V_{CHIRPJ}	Section 7.1.7.2	700	1100	mV
Chirp K level (differential voltage)	V_{CHIRPK}	Section 7.1.7.2	-900	-500	mV
Decoupling Capacitance:					
Downstream Facing Port Bypass Capacitance (per hub)	C_{HPB}	VBUS to GND, Section 7.2.4.1	120		µF
Upstream Facing Port Bypass Capacitance	C_{RPB}	VBUS to GND; Note 9, Section 7.2.4.1	1.0	10.0	µF
Input Capacitance for Low-/full-speed:					
Downstream Facing Port	C_{IND}	Note 2; Section 7.1.6.1		150	pF
Upstream Facing Port (w/o cable)	C_{INUB}	Note 3; Section 7.1.6.1		100	pF
Transceiver edge rate control capacitance	C_{EDGE}	Section 7.1.6.1		75	pF

Microchip Ex. 1018, p. 207
Microchip v. Aptiv
IPR2024-00494

Table 7-7. DC Electrical Characteristics (Continued)

Parameter	Symbol	Conditions	Min.	Max.	Units
Input Impedance for High-speed:					
TDR spec for high-speed termination		Section 7.1.6.2			
Terminations:					
Bus Pull-up Resistor on Upstream Facing Port	R_{PU}	1.5 kΩ ±5% Section 7.1.5	1.425	1.575	kΩ
Bus Pull-down Resistor on Downstream Facing Port	R_{PD}	15 kΩ ±5% Section 7.1.5	14.25	15.75	kΩ
Input impedance exclusive of pullup/pulldown (for low-/full-speed)	Z_{INP}	Section 7.1.6	300		kΩ
Termination voltage for upstream facing port pullup (R_{PU})	V_{TERM}	Section 7.1.5	3.0	3.6	V
Terminations in High-speed:					
Termination voltage in high-speed	V_{HSTERM}	Section 7.1.6.2	-10	10	mV

Table 7-8. High-speed Source Electrical Characteristics

Parameter	Symbol	Conditions	Min.	Max.	Units
Driver Characteristics:					
Rise Time (10% - 90%)	T_{HSR}	Section 7.1.2	500		ps
Fall Time (10% - 90%)	T_{HSF}	Section 7.1.2	500		ps
Driver waveform requirements		Specified by eye pattern templates in Section 7.1.2			
Driver Output Resistance (which also serves as high-speed termination)	Z_{HSDRV}	Section 7.1.1.1	40.5	49.5	Ω
Clock Timings:					
High-speed Data Rate	T_{HSDRAT}	Section 7.1.11	479.760	480.240	Mb/s
Microframe Interval	T_{HSFRAM}	Section 7.1.12	124.9375	125.0625	µs
Consecutive Microframe Interval Difference	T_{HSRFI}	Section 7.1.12		4 high-speed bit times	
High-speed Data Timings:					
Data source jitter		Source and receiver jitter specified by the eye pattern templates in Section 7.1.2.2			
Receiver jitter tolerance					

Microchip Ex. 1018, p. 208
Microchip v. Aptiv
IPR2024-00494

Table 7-9. Full-speed Source Electrical Characteristics

Parameter	Symbol	Conditions	Min.	Max.	Units
Driver Characteristics:					
Rise Time	T_{FR}	Figure 7-8; Figure 7-9	4	20	ns
Fall Time	T_{FF}	Figure 7-8; Figure 7-9	4	20	ns
Differential Rise and Fall Time Matching	T_{FRFM}	(T_{FR}/T_{FF}) Note 10, Section 7.1.2	90	111.11	%
Driver Output Resistance for driver which is not high-speed capable	Z_{DRV}	Section 7.1.1.1	28	44	Ω
Clock Timings:					
Full-speed Data Rate for hubs and devices which are high-speed capable	$T_{FDRATHS}$	Average bit rate, Section 7.1.11	11.9940	12.0060	Mb/s
Full-speed Data Rate for devices which are not high-speed capable	T_{FDRATE}	Average bit rate, Section 7.1.11	11.9700	12.0300	Mb/s
Frame Interval	T_{FRAME}	Section 7.1.12	0.9995	1.0005	ms
Consecutive Frame Interval Jitter	T_{RFI}	No clock adjustment Section 7.1.12		42	ns
Full-speed Data Timings:					
Source Jitter Total (including frequency tolerance): To Next Transition For Paired Transitions	 T_{DJ1} T_{DJ2}	Note 7, 8, 12, 10; Measured as in Figure 7-49;	 -3.5 -4	 3.5 4	 ns ns
Source Jitter for Differential Transition to SE0 Transition	T_{FDEOP}	Note 8; Figure 7-50; Note 11	-2	5	ns
Receiver Jitter: To Next Transition For Paired Transitions	 T_{JR1} T_{JR2}	Note 8; Figure 7-51	 -18.5 -9	 18.5 9	 ns ns
Source SE0 interval of EOP	T_{FEOPT}	Figure 7-50	160	175	ns
Receiver SE0 interval of EOP	T_{FEOPR}	Note 13; Section 7.1.13.2; Figure 7-50	82		ns
Width of SE0 interval during differential transition	T_{FST}	Section 7.1.4		14	ns

Microchip Ex. 1018, p. 209
Microchip v. Aptiv
IPR2024-00494

Table 7-10. Low-speed Source Electrical Characteristics

Parameter	Symbol	Conditions	Min.	Max.	Units
Driver Characteristics:					
Transition Time:					
Rise Time	T_{LR}	Measured as in Figure 7-8	75	300	ns
Fall Time	T_{LF}		75	300	ns
Rise and Fall Time Matching	T_{LRFM}	(T_{LR}/T_{LF}) Note 10	80	125	%
Upstream Facing Port (w/cable, low-speed only)	C_{LINUA}	Note 1; Section 7.1.6	200	450	pF
Clock Timings:					
Low-speed Data Rate for hubs which are high-speed capable	$T_{LDRATHS}$	Section 7.1.11	1.49925	1.50075	Mb/s
Low-speed Data Rate for devices which are not high-speed capable	T_{LDRATE}	Section 7.1.11	1.4775	1.5225	Mb/s
Low-speed Data Timings:					
Upstream facing port source Jitter Total (including frequency tolerance):		Note 7, 8; Figure 7-49			
To Next Transition	T_{UDJ1}		-95	95	ns
For Paired Transitions	T_{UDJ2}		-150	150	ns
Upstream facing port source Jitter for Differential Transition to SE0 Transition	T_{LDEOP}	Note 8; Figure 7-50; Note 11	-40	100	ns
Upstream facing port differential Receiver Jitter:		Note 8; Figure 7-51			
To Next Transition	T_{DJR1}		-75	75	ns
For Paired Transitions	T_{DJR2}		-45	45	ns
Downstream facing port source Jitter Total (including frequency tolerance):		Note 7, 8; Figure 7-49			
To Next Transition	T_{DDJ1}		-25	25	ns
For Paired Transitions	T_{DDJ2}		-14	14	ns
Downstream facing port source Jitter for Differential Transition to SE0 Transition		Note 8; Figure 7-50; Note 11			ns
Downstream facing port Differential Receiver Jitter:		Note 8; Figure 7-50			
To Next Transition	T_{UJR1}		-152	152	ns
For Paired Transitions	T_{UJR2}		-200	200	ns

Microchip Ex. 1018, p. 210
Microchip v. Aptiv
IPR2024-00494

Table 7-10. Low-speed Source Electrical Characteristics (Continued)

Parameter	Symbol	Conditions	Min.	Max.	Units
Source SE0 interval of EOP	T_{LEOPT}	Figure 7-50	1.25	1.50	µs
Receiver SE0 interval of EOP	T_{LEOPR}	Note 13; Section 7.1.13.2; Figure 7-50	670		ns
Width of SE0 interval during differential transition	T_{LST}	Section 7.1.4		210	ns

Table 7-11. Hub/Repeater Electrical Characteristics

Parameter	Symbol	Conditions	Min.	Max.	Units
Full-speed Hub Characteristics (as measured at connectors):					
Driver Characteristics: (Refer to Table 7-9)		Upstream facing port and downstream facing ports configured as full-speed			
Hub Differential Data Delay:		Note 7, 8			
(with cable)	T_{HDD1}	Figure 7-52A		70	ns
(without cable)	T_{HDD2}	Figure 7-52B		44	ns
Hub Differential Driver Jitter: (including cable)		Note 7, 8; Figure 7-52, Section 7.1.14			
To Next Transition	T_{HDJ1}		-3	3	ns
For Paired Transitions	T_{HDJ2}		-1	1	ns
Data Bit Width Distortion after SOP	T_{FSOP}	Note 8; Figure 7-52	-5	5	ns
Hub EOP Delay Relative to T_{HDD}	T_{FEOPD}	Note 8; Figure 7-53	0	15	ns
Hub EOP Output Width Skew	T_{FHESK}	Note 8; Figure 7-53	-15	15	ns
Low-speed Hub Characteristics (as measured at connectors):					
Driver Characteristics: (Refer to Table 7-10)		Downstream facing ports configured as low-speed			
Hub Differential Data Delay	T_{LHDD}	Note 7, 8; Figure 7-52		300	ns
Hub Differential Driver Jitter (including cable):		Note 7, 8; Figure 7-52			
Downstream facing port :					
To Next Transition	T_{LDHJ1}		-45	45	ns
For Paired Transitions	T_{LDHJ2}		-15	15	ns
Upstream facing port:					
To Next Transition	T_{LUHJ1}		-45	45	ns
For Paired Transitions	T_{LUHJ2}		-45	45	ns
Data Bit Width Distortion after SOP	T_{LSOP}	Note 8; Figure 7-52	-60	60	ns
Hub EOP Delay Relative to T_{HDD}	T_{LEOPD}	Note 8; Figure 7-53	0	200	ns
Hub EOP Output Width Skew	T_{LHESK}	Note 8; Figure 7-53	-300	+300	ns

Microchip Ex. 1018, p. 211
Microchip v. Aptiv
IPR2024-00494

Table 7-11. Hub/Repeater Electrical Characteristics (Continued)

Parameter	Symbol	Conditions	Min.	Max.	Units
High-speed Hub Characteristics (as measured at connectors):					
Driver Characteristics: (Refer to Table 7-8)		Upstream facing port and downstream facing ports configured as high-speed			
Hub Data Delay (without cable):	T_{HSHDD}	Section 7.1.14.2		36 high-speed bit times + 4 ns	
Hub Data Jitter:		Specified by eye patterns in Section 7.1.2.2			
Hub Delay Variation Range:	T_{HSHDV}	Section 7.1.14.2		5 high-speed bit times	

Microchip Ex. 1018, p. 212
Microchip v. Aptiv
IPR2024-00494

Table 7-12. Cable Characteristics (Note 14)

Parameter	Symbol	Conditions	Min	Max	Units
V_{BUS} Voltage drop for detachable cables	V_{BUSD}	Section 7.2.2		125	mV
GND Voltage drop (for all cables)	V_{GNDD}	Section 7.2.2		125	mV
Differential Cable Impedance (full-/high-speed)	Z_O	(90 Ω ±15%);	76.5	103.5	Ω
Common mode cable impedance (full-/high-speed)	Z_{CM}	(30 Ω ±30%);	21.0	39.0	Ω
Cable Delay (one way) Full-/high-speed Low-speed	 T_{FSCBL} T_{LSCBL}	Section 7.1.16		 26 18	 ns ns
Cable Skew	T_{SKEW}	Section 7.1.3		100	ps
Unmated Contact Capacitance	C_{UC}	Section 6.7		2	pF
Cable loss		Specified by table and graph in Section 7.1.17			

Note 1: Measured at A plug.
Note 2: Measured at A receptacle.
Note 3: Measured at B receptacle.
Note 4: Measured at A or B connector.
Note 5: Measured with RL of 1.425 kΩ to 3.6 V.
Note 6: Measured with RL of 14.25 kΩ to GND.
Note 7: Timing difference between the differential data signals.
Note 8: Measured at crossover point of differential data signals.
Note 9: The maximum load specification is the maximum effective capacitive load allowed that meets the target V_{BUS} drop of 330 mV.
Note 10: Excluding the first transition from the Idle state.
Note 11: The two transitions should be a (nominal) bit time apart.
Note 12: For both transitions of differential signaling.
Note 13: Must accept as valid EOP.
Note 14: Single-ended capacitance of D+ or D- is the capacitance of D+/D- to all other conductors and, if present, shield in the cable. That is, to measure the single-ended capacitance of D+, short D-, VBUS, GND, and the shield line together and measure the capacitance of D+ to the other conductors.
Note 15: For high power devices (non-hubs) when enabled for remote wakeup.

Microchip Ex. 1018, p. 213
Microchip v. Aptiv
IPR2024-00494

Table 7-13. Hub Event Timings

Event Description	Symbol	Conditions	Min	Max	Unit
Time to detect a downstream facing port connect event Awake Hub Suspended Hub	TDCNN	Section 11.5 and Section 7.1.7.3	 2.5 2.5	 2000 12000	 µs µs
Time to detect a disconnect event at a hub's downstream facing port	TDDIS	Section 7.1.7.3	2	2.5	µs
Duration of driving resume to a downstream port; only from a controlling hub	TDRSMDN	Nominal; Section 7.1.7.7 and Section 11.5	20		ms
Time from detecting downstream resume to rebroadcast	TURSM	Section 7.1.7.7		1.0	ms
Duration of driving reset to a downstream facing port	TDRST	Only for a SetPortFeature (PORT_RESET) request; Section 7.1.7.5 and Section 11.5	10	20	ms
Overall duration of driving reset to downstream facing port, root hub	TDRSTR	Only for root hubs; Section 7.1.7.5	50		ms
Maximum interval between reset segments used to create TDRSTR	TRHRSI	Only for root hubs; each reset pulse must be of length TDRST; Section 7.1.7.5		3	ms
Time to detect a long K from upstream	TURLK	Section 11.6	2.5	100	µs
Time to detect a long SE0 from upstream	TURLSE0	Section 11.6	2.5	10000	µs
Duration of repeating SE0 upstream (for low-/full-speed repeater)	TURPSE0	Section 11.6		23	FS bit times
Duration of sending SE0 upstream after EOF1 (for low-/full-speed repeater)	TUDEOP	*Optional* Section 11.6		2	FS bit times
Inter-packet Delay (for high-speed) for packets traveling in same direction	THSIPDSD	Section 7.1.18.2	88		bit times
Inter-packet Delay (for high-speed) for packets traveling in opposite direction	THSIPDOD	Section 7.1.18.2	8		bit times

Microchip Ex. 1018, p. 214
Microchip v. Aptiv
IPR2024-00494

Table 7-13. Hub Event Timings (Continued)

Event Description	Symbol	Conditions	Min	Max	Unit
Inter-packet delay for device/root hub response w/detachable cable for high-speed	$T_{HSRSPIPD1}$	Section 7.1.18.2		192	bit times
Reset Handshake Protocol:					
Time for which a Chirp J or Chirp K must be continuously detected (filtered) by hub or device during Reset handshake	T_{FILT}	Section 7.1.7.5	2.5		µs
Time after end of device Chirp K by which hub must start driving first Chirp K in the hub's chirp sequence	T_{WTDCH}	Section 7.1.7.5		100	µs
Time for which each individual Chirp J or Chirp K in the chirp sequence is driven downstream by hub during reset	T_{DCHBIT}	Section 7.1.7.5	40	60	µs
Time before end of reset by which a hub must end its downstream chirp sequence	T_{DCHSE0}	Section 7.1.7.5	100	500	µs

Microchip Ex. 1018, p. 215
Microchip v. Aptiv
IPR2024-00494

Table 7-14. Device Event Timings

Parameter	Symbol	Conditions	Min	Max	Units
Time from internal power good to device pulling D+/D- beyond V_{IHZ} (min) (signaling attach)	T_{SIGATT}	Figure 7-29		100	ms
Debounce interval provided by USB system software after attach	T_{ATTDB}	Figure 7-29		100	ms
Maximum time a device can draw power >suspend power when bus is continuously in idle state	T_{2SUSP}	Section 7.1.7.6		10	ms
Maximum duration of suspend averaging interval	$T_{SUSAVGI}$	Section 7.2.3		1	s
Period of idle bus before device can initiate resume	T_{WTRSM}	Device must be remote-wakeup enabled Section 7.1.7.5	5		ms
Duration of driving resume upstream	T_{DRSMUP}	Section 7.1.7.7	1	15	ms
Resume Recovery Time	T_{RSMRCY}	Provided by USB System Software; Section 7.1.7.7	10		ms
Time to detect a reset from upstream for non high-speed capable devices	T_{DETRST}	Section 7.1.7.5	2.5	10000	µs
Reset Recovery Time	T_{RSTRCY}	Section 7.1.7.5		10	ms
Inter-packet Delay (for low-/full-speed)	T_{IPD}	Section 7.1.18	2		bit times
Inter-packet delay for device response w/detachable cable for low-/full-speed	$T_{RSPIPD1}$	Section 7.1.18		6.5	bit times
Inter-packet delay for device response w/captive cable for low-/full-speed	$T_{RSPIPD2}$	Section 7.1.18		7.5	bit times

Microchip Ex. 1018, p. 216
Microchip v. Aptiv
IPR2024-00494

Table 7-14. Device Event Timings (Continued)

Parameter	Symbol	Conditions	Min	Max	Units
SetAddress() Completion Time	TDSETADDR	Section 9.2.6.3		50	ms
Time to complete standard request with no data	TDRQCMPLTND	Section 9.2.6.4		50	ms
Time to deliver first and subsequent (except last) data for standard request	TDRETDATA1	Section 9.2.6.4		500	ms
Time to deliver last data for standard request	TDRETDATAN	Section 9.2.6.4		50	ms
Inter-packet delay for device response w/captive cable (high-speed)	THSRSPIPD2	Section 7.1.18.2		192 bit times + 52 ns	
SetAddress() Completion Time	TDSETADDR	Section 9.2.6.3		50	ms
Time to complete standard request with no data	TDRQCMPLTND	Section 9.2.6.4		50	ms
Reset Handshake Protocol:					
Time for which a suspended high-speed capable device must see a continuous SE0 before beginning the high-speed detection handshake	TFILTSE0	Section 7.1.7.5	2.5		µs
Time a high-speed capable device operating in non-suspended full-speed must wait after start of SE0 before beginning the high-speed detection handshake	TWTRSTFS	Section 7.1.7.5	2.5	3000	µs
Time a high-speed capable device operating in high-speed must wait after start of SE0 before reverting to full-speed	TWTREV	Section 7.1.7.5	3.0	3.125	ms
Time a device must wait after reverting to full-speed before sampling the bus state for SE0 and beginning the high-speed detection handshake	TWTRSTHS	Section 7.1.7.5	100	875	µs

Microchip Ex. 1018, p. 217
Microchip v. Aptiv
IPR2024-00494

Table 7-14. Device Event Timings (Continued)

Parameter	Symbol	Conditions	Min	Max	Units
Minimum duration of a Chirp K from a high-speed capable device within the reset protocol	TUCH	Section 7.1.7.5	1.0		ms
Time after start of SE0 by which a high-speed capable device is required to have completed its Chirp K within the reset protocol	TUCHEND	Section 7.1.7.5		7.0	ms
Time between detection of downstream chirp and entering high-speed state	TWTHS	Section 7.1.7.5		500	µs
Time after end of upstream chirp at which device reverts to full-speed default state if no downstream chirp is detected	TWTFS	Section 7.1.7.5	1.0	2.5	ms

Microchip Ex. 1018, p. 218
Microchip v. Aptiv
IPR2024-00494

7.3.3 Timing Waveforms

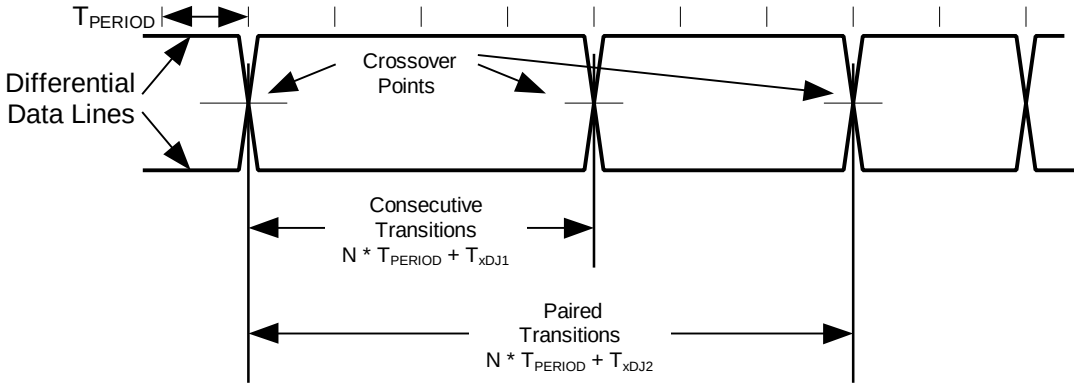


Figure 7-49. Differential Data Jitter for Low-/full-speed

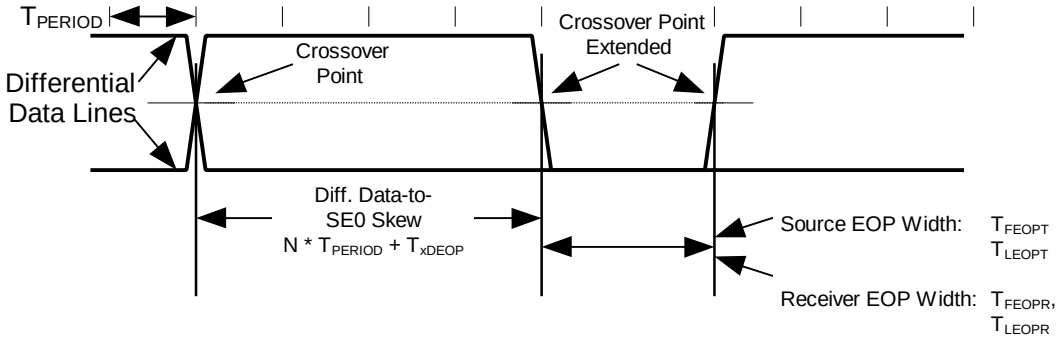


Figure 7-50. Differential-to-EOP Transition Skew and EOP Width for Low-/full-speed

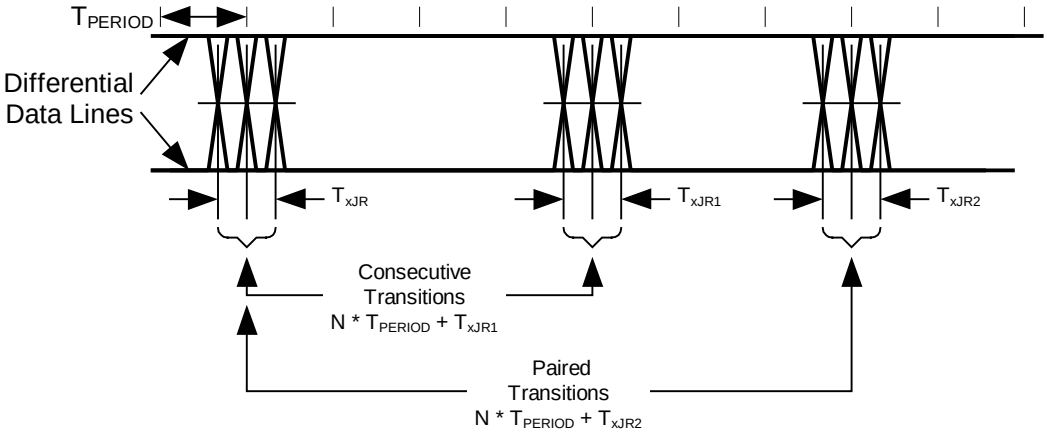
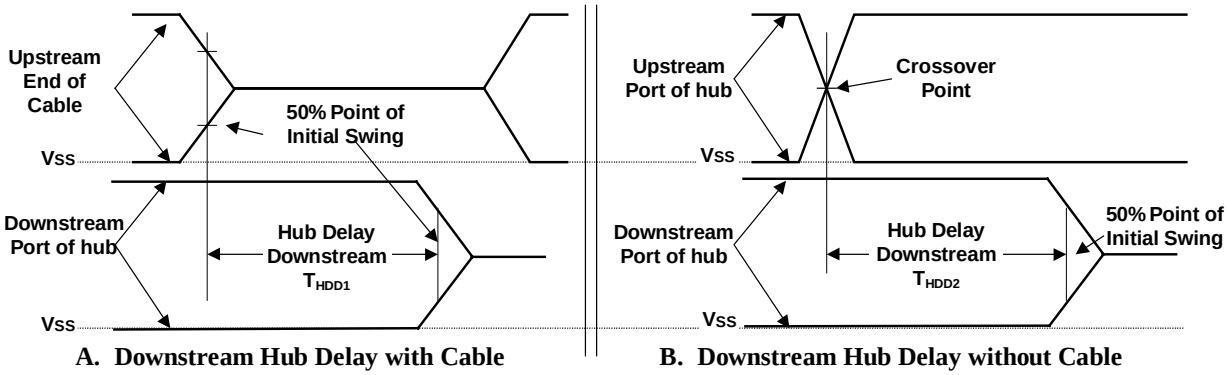


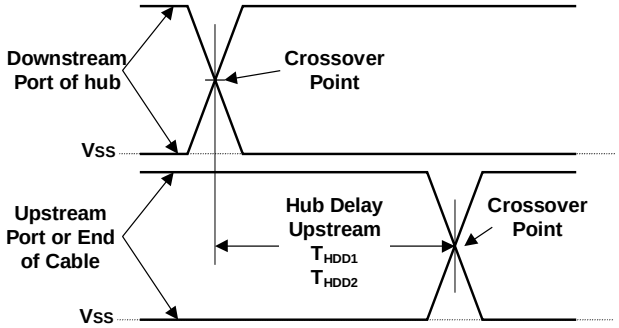
Figure 7-51. Receiver Jitter Tolerance for Low-/full-speed

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

Microchip Ex. 1018, p. 219
Microchip v. Aptiv
IPR2024-00494



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



C. Upstream Hub Delay with or without Cable

Hub Differential Jitter:

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

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

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

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

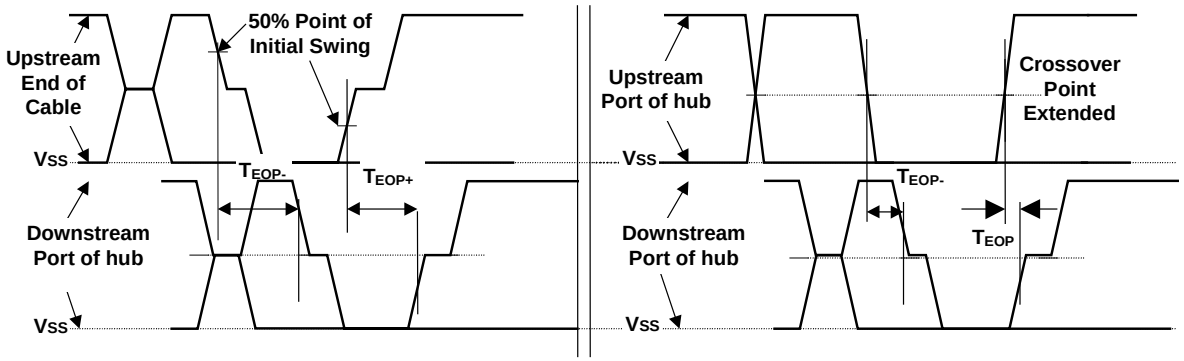
Low-speed timings are determined in the same way for:

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

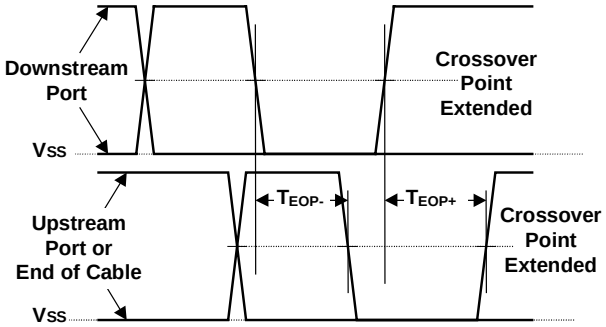
Figure 7-52. Hub Differential Delay, Differential Jitter, and SOP Distortion for Low-/full-speed

Measurement locations referenced in Figure 7-52 and Figure 7-53 are specified in Figure 7-38.

Microchip Ex. 1018, p. 220
Microchip v. Aptiv
IPR2024-00494



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



C. Upstream EOP Delay with or Without Cable

EOP Delay:

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

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

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

Figure 7-53. Hub EOP Delay and EOP Skew for Low-/full-speed

Microchip Ex. 1018, p. 221
Microchip v. Aptiv
IPR2024-00494

Microchip Ex. 1018, p. 222
Microchip v. Aptiv
IPR2024-00494

Chapter 8
Protocol Layer

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

8.1 Byte/Bit Ordering

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

Multiple byte fields in standard descriptors, requests, and responses are interpreted as and moved over the bus in little-endian order, i.e., LSB to MSB.

8.2 SYNC Field

All packets begin with a synchronization (SYNC) field, which is a coded sequence that generates a maximum edge transition density. It is used by the input circuitry to align incoming data with the local clock. A SYNC from an initial transmitter is defined to be eight bits in length for full/low-speed and 32 bits for high-speed. Received SYNC fields may be shorter as described in Chapter 7. SYNC serves only as a synchronization mechanism and is not shown in the following packet diagrams (refer to Section 7.1.10). The last two bits in the SYNC field are a marker that is used to identify the end of the SYNC field and, by inference, the start of the PID.

8.3 Packet Field Formats

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

8.3.1 Packet Identifier Field

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

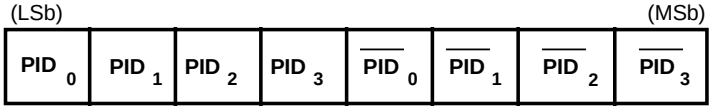


Figure 8-1. PID Format

The host and all functions must perform a complete decoding of all received PID fields. Any PID received with a failed check field or which decodes to a non-defined value is assumed to be corrupted and it, as well

Microchip Ex. 1018, p. 223
Microchip v. Aptiv
IPR2024-00494

as the remainder of the packet, is ignored by the packet receiver. If a function receives an otherwise valid PID for a transaction type or direction that it does not support, the function must not respond. For example, an IN-only endpoint must ignore an OUT token. PID types, codings, and descriptions are listed in Table 8-1.

Table 8-1. PID Types

PID Type	PID Name	PID<3:0>*	Description
Token	OUT	0001B	Address + endpoint number in host-to-function transaction
	IN	1001B	Address + endpoint number in function-to-host transaction
	SOF	0101B	Start-of-Frame marker and frame number
	SETUP	1101B	Address + endpoint number in host-to-function transaction for SETUP to a control pipe
Data	DATA0	0011B	Data packet PID even
	DATA1	1011B	Data packet PID odd
	DATA2	0111B	Data packet PID high-speed, high bandwidth isochronous transaction in a microframe (see Section 5.9.2 for more information)
	MDATA	1111B	Data packet PID high-speed for split and high bandwidth isochronous transactions (see Sections 5.9.2, 11.20, and 11.21 for more information)
Handshake	ACK	0010B	Receiver accepts error-free data packet
	NAK	1010B	Receiving device cannot accept data or transmitting device cannot send data
	STALL	1110B	Endpoint is halted or a control pipe request is not supported
	NYET	0110B	No response yet from receiver (see Sections 8.5.1 and 11.17-11.21)
Special	PRE	1100B	(Token) Host-issued preamble. Enables downstream bus traffic to low-speed devices.
	ERR	1100B	(Handshake) Split Transaction Error Handshake (reuses PRE value)
	SPLIT	1000B	(Token) High-speed Split Transaction Token (see Section 8.4.2)
	PING	0100B	(Token) High-speed flow control probe for a bulk/control endpoint (see Section 8.5.1)
	Reserved	0000B	Reserved PID

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

Microchip Ex. 1018, p. 224
Microchip v. Aptiv
IPR2024-00494

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

8.3.2 Address Fields

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

8.3.2.1 Address Field

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

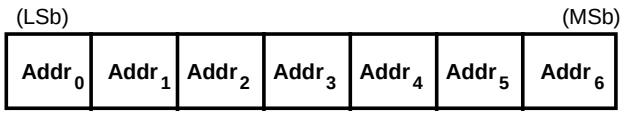


Figure 8-2. ADDR Field

8.3.2.2 Endpoint Field

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

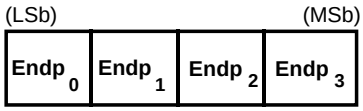


Figure 8-3. Endpoint Field

8.3.3 Frame Number Field

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

8.3.4 Data Field

The data field may range from zero to 1,024 bytes and must be an integral number of bytes. Figure 8-4 shows the format for multiple bytes. Data bits within each byte are shifted out LSb first.

Microchip Ex. 1018, p. 225
Microchip v. Aptiv
IPR2024-00494

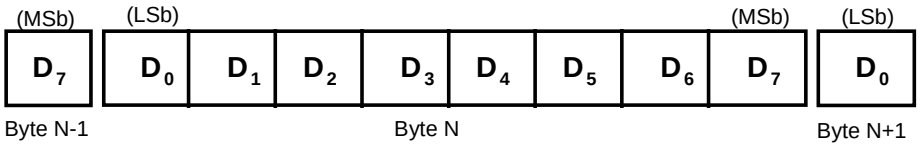


Figure 8-4. Data Field Format

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

8.3.5 Cyclic Redundancy Checks

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

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

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

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

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

8.3.5.1 Token CRCs

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

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

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

8.3.5.2 Data CRCs

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

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

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

Microchip Ex. 1018, p. 226
Microchip v. Aptiv
IPR2024-00494

8.4 Packet Formats

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

8.4.1 Token Packets

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

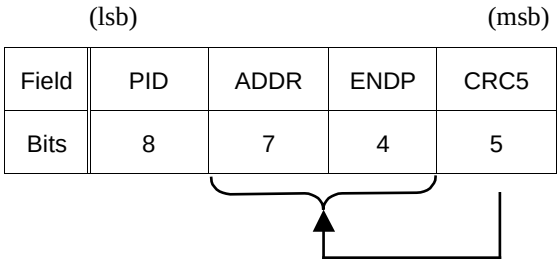


Figure 8-5. Token Format

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

8.4.2 Split Transaction Special Token Packets

USB defines a special token for split transactions: SPLIT. This is a 4 byte token packet compared to other normal 3 byte token packets. The split transaction token packet provides additional transaction types with additional transaction specific information. The split transaction token is used to support split transactions between the host controller communicating with a hub operating at high speed with full-/low-speed devices to some of its downstream facing ports. There are two split transactions defined that use the SPLIT special token: a start-split transaction (SSPLIT) and a complete-split transaction (CSPLIT). A field in the SPLIT special token, described in the following sections, indicates the specific split transaction.

8.4.2.1 Split Transactions

A high-speed split transaction is used only between the host controller and a hub when the hub has full-/low-speed devices attached to it. This high-speed split transaction is used to initiate a full-/low-speed transaction via the hub and some full-/low-speed device endpoint. The high-speed split transaction also allows the completion status of the full-/low-speed transaction to be retrieved from the hub. This approach allows the host controller to start a full-/low-speed transaction via a high-speed transaction and then continue with other high-speed transactions without having to wait for the full-/low-speed transaction to proceed/complete at the slower speed. See Chapter 11 for more details about the state machines and transaction definitions of split transactions.

A high-speed split transaction has two parts: a start-split and a complete-split. Split transactions are only defined to be used between the host controller and a hub. No other high-speed or full-/low-speed devices ever use split transactions.

Microchip Ex. 1018, p. 227
Microchip v. Aptiv
IPR2024-00494

Figure 8-6 shows the packets composing a generic start-split transaction. There are two packets in the token phase: the SPLIT special token and a full-/low-speed token. Depending on the direction of data transfer and whether a handshake is defined for the transaction type, the token phase is optionally followed by a data packet and a handshake packet. Start split transactions can consist of 2, 3, or 4 packets as determined by the specific transfer type and data direction.

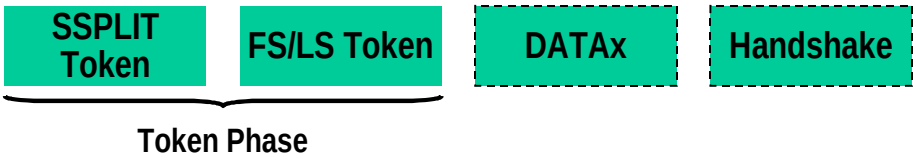


Figure 8-6. Packets in a Start-split Transaction

Figure 8-7 shows the packets composing a generic complete-split transaction. There are two packets in the token phase: the SPLIT special token and a full-/low-speed token. A data or handshake packet follows the token phase packets in the complete-split depending on the data transfer direction and specific transaction type. Complete split transactions can consist of 2 or 3 packets as determined by the specific transfer type and data direction.

Figure 8-7. Packets in a Complete-split Transaction

The results of a split transaction are returned by a complete-split transaction. Figure 8-8 shows this conceptual “conversion” for an example interrupt IN transfer type. The host issues a start-split (indicated with 1) to the hub and then can proceed with other high-speed transactions. The start-split causes the hub to issue a full-/low-speed IN token sometime later (indicated by 2). The device responds to the IN token (in this example) with a data packet and the hub responds with a handshake to the device. Finally, the host sometime later issues a complete-split (indicated by 3) to retrieve the data provided by the device. Note that in the example, the hub provided the full-/low-speed handshake (ACK in this example) to the device endpoint before the complete-split, and the complete-split did not provide a high-speed handshake to the hub.

Microchip Ex. 1018, p. 228
Microchip v. Aptiv
IPR2024-00494

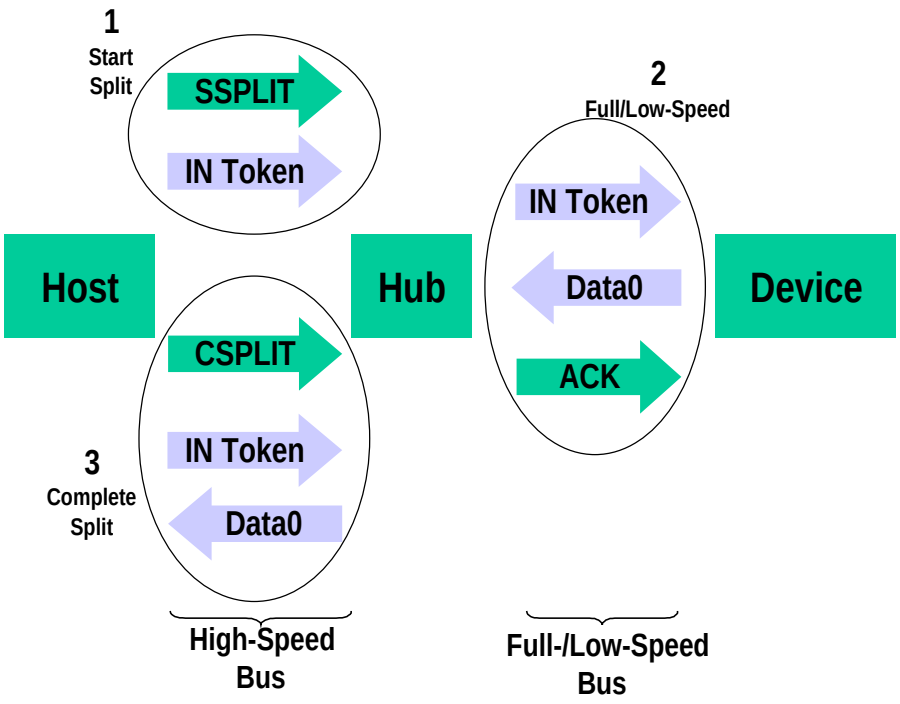


Figure 8-8. Relationship of Interrupt IN Transaction to High-speed Split Transaction

A normal full-/low-speed OUT transaction is similarly conceptually "converted" into start-split and complete-split transactions. Figure 8-9 shows this "conversion" for an example interrupt OUT transfer type. The host issues a start-split transaction consisting of a SSPLIT special token, an OUT token, and a DATA packet. The hub sometime later issues the OUT token and DATA packet on the full-/low-speed bus. The device responds with a handshake. Sometime later, the host issues the complete-split transaction and the hub responds with the results (either full-/low-speed data or handshake) provided by the device.

Microchip Ex. 1018, p. 229
Microchip v. Aptiv
IPR2024-00494

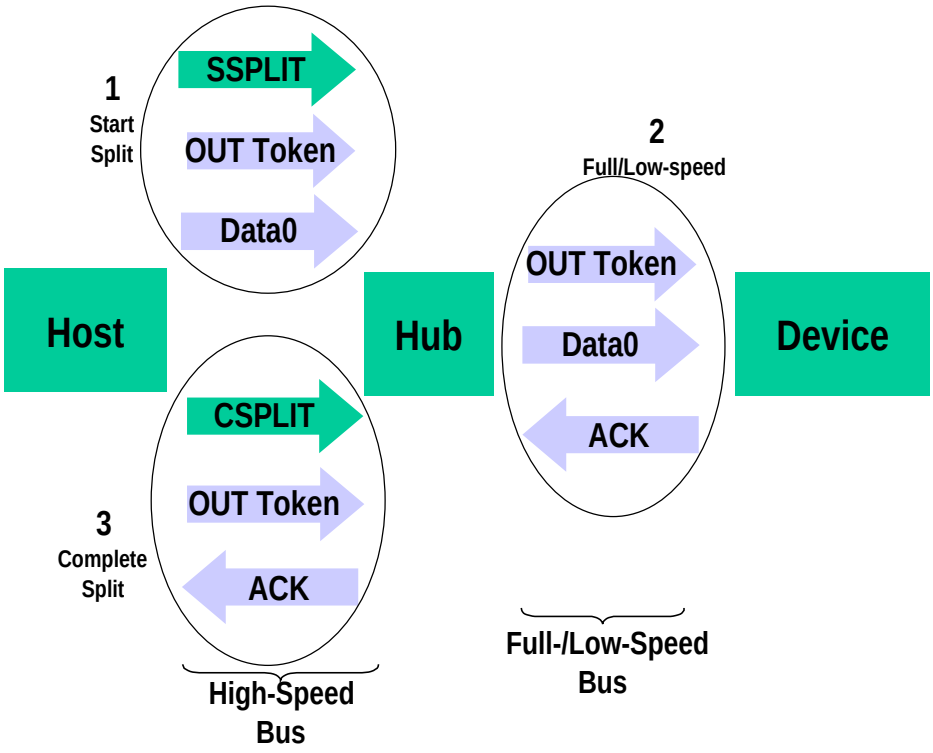


Figure 8-9. Relationship of Interrupt OUT Transaction to High-speed Split OUT Transaction

The next two sections describe the fields composing the detailed start- and complete-split token packets. Figure 8-10 and Figure 8-12 show the fields in the split-transaction token packet. The SPLIT special token follows the general token format and starts with a PID field (after a SYNC) and ends with a CRC5 field (and EOP). Start-split and complete-split token packets are both 4 bytes long. SPLIT transactions must only originate from the host. The start-split token is defined in Section 8.4.2.2 and the complete-split token is defined in Section 8.4.2.3.

8.4.2.2 Start-Split Transaction Token

(lsb) (msb)

Field	SPLIT PID	Hub Addr	SC	Port	S	E	ET	CRC5
Bits	8	7	1	7	1	1	2	5

Figure 8-10. Start-split (SSPLIT) Token

The Hub addr field contains the USB device address of the hub supporting the specified full-/low-speed device for this full-/low-speed transaction. This field has the same definition as the ADDR field definition in Section 8.3.2.1.

A SPLIT special token packet with the SC (Start/Complete) field set to zero indicates that this is a start-split transaction (SSPLIT).

The Port field contains the port number of the target hub for which this full-/low-speed transaction is destined. As shown in Figure 8-11, a total of 128 ports are specified as PORT<6:0>. The host must correctly set the port field for single and multiple TT hub implementations. A single TT hub implementation may ignore the port field.

Microchip Ex. 1018, p. 230
Microchip v. Aptiv
IPR2024-00494

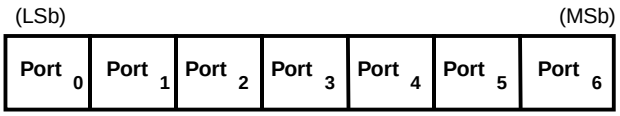


Figure 8-11. Port Field

The S (Speed) field specifies the speed for this interrupt or control transaction as follows:

- 0 – Full speed
- 1 – Low speed

For bulk IN/OUT and isochronous IN start-splits, the S field must be set to zero. For bulk/control IN/OUT, interrupt IN/OUT, and isochronous IN start-splits, the E field must be set to zero.

For full-speed isochronous OUT start-splits, the S[1] (Start) and E (End) fields specify how the high-speed data payload corresponds to data for a full-speed data packet as shown in Table 8-2.

Table 8-2. Isochronous OUT Payload Continuation Encoding

S	E	High-speed to Full-speed Data Relation
0	0	High-speed data is the middle of the full-speed data payload
0	1	High-speed data is the end of the full-speed data payload
1	0	High-speed data is the beginning of the full-speed data payload
1	1	High-speed data is all of the full-speed data payload.

Isochronous OUT start-split transactions use these encodings to allow the hub to detect various error cases due to lack of receiving start-split transactions for an endpoint with a data payload that requires multiple start-splits. For example, a large full-speed data payload may require three start-split transactions: a start-split/beginning, a start-split/middle and a start-split/end. If any of these transactions is not received by the hub, it will either ignore the full-speed transaction (if the start-split/beginning is not received), or it will force an error for the corresponding full-speed transaction (if one of the other two transactions are not received). Other error conditions can be detected by not receiving a start-split during a microframe.

The ET (Endpoint Type) field specifies the endpoint type of the full-/low-speed transaction as shown in Table 8-3.

[1] The S bit can be reused for these encodings since isochronous transactions must not be low speed.

Microchip Ex. 1018, p. 231
Microchip v. Aptiv
IPR2024-00494

Table 8-3. Endpoint Type Values in Split Special Token

ET value (msb:lsb)	Endpoint Type
00	Control
01	Isochronous
10	Bulk
11	Interrupt

This field tells the hub which split transaction state machine to use for this full-/low-speed transaction.

The full-/low-speed device address and endpoint number information is contained in the normal token packet that follows the SPLIT special token packet.

8.4.2.3 Complete-Split Transaction Token

(lsb) (msb)

Field	SPLIT PID	Hub Addr	SC	Port	S	U	ET	CRC5
Bits	8	7	1	7	1	1	2	5

Figure 8-12. Complete-split (CSPLIT) Transaction Token

A SPLIT special token packet with the SC field set to one indicates that this is a complete-split transaction (CSPLIT).

The U bit is reserved/unused and must be reset to zero(0B).

The other fields of the complete-split token packet have the same definitions as for the start-split token packet.

8.4.3 Start-of-Frame Packets

Start-of-Frame (SOF) packets are issued by the host at a nominal rate of once every 1.00 ms ±0.0005 ms for a full-speed bus and 125 µs ±0.0625 µs for a high-speed bus. SOF packets consist of a PID indicating packet type followed by an 11-bit frame number field as illustrated in Figure 8-13.

(lsb) (msb)

Field	PID	FrameNumber	CRC5
Bits	8	11	5

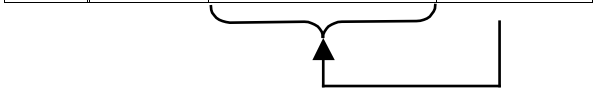


Figure 8-13. SOF Packet

The SOF token comprises the token-only transaction that distributes an SOF marker and accompanying frame number at precisely timed intervals corresponding to the start of each frame. All high-speed and full-speed functions, including hubs, receive the SOF packet. The SOF token does not cause any receiving function to generate a return packet; therefore, SOF delivery to any given function cannot be guaranteed.

Microchip Ex. 1018, p. 232
Microchip v. Aptiv
IPR2024-00494

The SOF packet delivers two pieces of timing information. A function is informed that an SOF has occurred when it detects the SOF PID. Frame timing sensitive functions, that do not need to keep track of frame number (e.g., a full-speed operating hub), need only decode the SOF PID; they can ignore the frame number and its CRC. If a function needs to track frame number, it must comprehend both the PID and the time stamp. Full-speed devices that have no particular need for bus timing information may ignore the SOF packet.

8.4.3.1 USB Frames and Microframes

USB defines a full-speed 1 ms frame time indicated by a Start Of Frame (SOF) packet each and every 1ms period with defined jitter tolerances. USB also defines a high-speed microframe with a 125 µs frame time with related jitter tolerances (See Chapter 7). SOF packets are generated (by the host controller or hub transaction translator) every 1ms for full-speed links. SOF packets are also generated after the next seven 125 µs periods for high-speed links.

Figure 8-14 shows the relationship between microframes and frames.

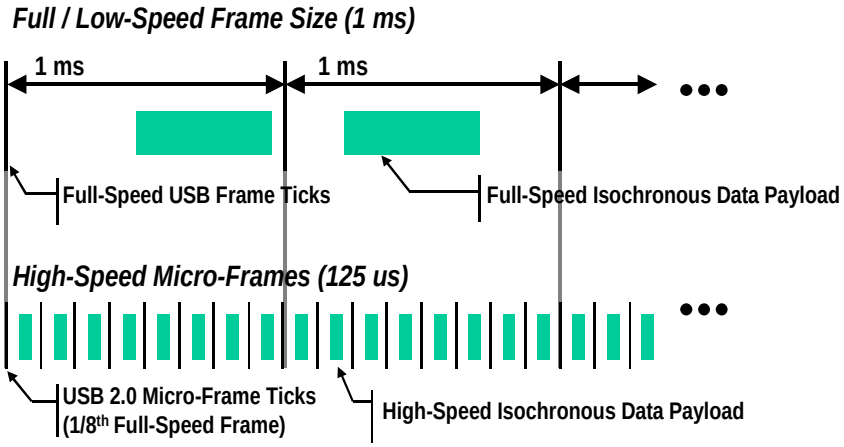


Figure 8-14. Relationship between Frames and Microframes

High-speed devices see an SOF packet with the same frame number eight times (every 125 µs) during each 1 ms period. If desired, a high-speed device can locally determine a particular microframe "number" by detecting the SOF that had a different frame number than the previous SOF and treating that as the zeroth microframe. The next seven SOFs with the same frame number can be treated as microframes 1 through 7.

Microchip Ex. 1018, p. 233
Microchip v. Aptiv
IPR2024-00494

8.4.4 Data Packets

A data packet consists of a PID, a data field containing zero or more bytes of data, and a CRC as shown in Figure 8-15. There are four types of data packets, identified by differing PIDs: DATA0, DATA1, DATA2 and MDATA. Two data packet PIDs (DATA0 and DATA1) are defined to support data toggle synchronization (refer to Section 8.6). All four data PIDs are used in data PID sequencing for high bandwidth high-speed isochronous endpoints (refer to Section 5.9). Three data PIDs (MDATA, DATA0, DATA1) are used in split transactions (refer to Sections 11.17-11.21).

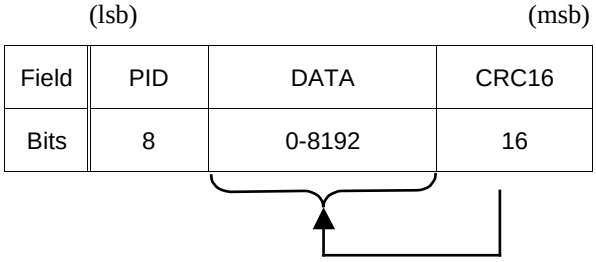


Figure 8-15. Data Packet Format

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

The maximum data payload size allowed for low-speed devices is 8 bytes. The maximum data payload size for full-speed devices is 1023. The maximum data payload size for high-speed devices is 1024 bytes.

8.4.5 Handshake Packets

Handshake packets, as shown in Figure 8-16, consist of only a PID. Handshake packets are used to report the status of a data transaction and can return values indicating successful reception of data, command acceptance or rejection, flow control, and halt conditions. Only transaction types that support flow control can return handshakes. Handshakes are always returned in the handshake phase of a transaction and may be returned, instead of data, in the data phase. Handshake packets are delimited by an EOP after one byte of packet field. If a packet decodes as an otherwise valid handshake but does not terminate with an EOP after one byte, it must be considered invalid and ignored by the receiver.

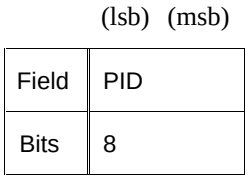


Figure 8-16. Handshake Packet

There are four types of handshake packets and one special handshake packet:

- **ACK** indicates that the data packet was received without bit stuff or CRC errors over the data field and that the data PID was received correctly. ACK may be issued either when sequence bits match and the receiver can accept data or when sequence bits mismatch and the sender and receiver must resynchronize to each other (refer to Section 8.6 for details). An ACK handshake is applicable only in transactions in which data has been transmitted and where a handshake is expected. ACK can be returned by the host for IN transactions and by a function for OUT, SETUP, or PING transactions.
- **NAK** indicates that a function was unable to accept data from the host (OUT) or that a function has no data to transmit to the host (IN). NAK can only be returned by functions in the data phase of IN transactions or the handshake phase of OUT or PING transactions. The host can never issue NAK.

Microchip Ex. 1018, p. 234
Microchip v. Aptiv
IPR2024-00494

NAK is used for flow control purposes to indicate that a function is temporarily unable to transmit or receive data, but will eventually be able to do so without need of host intervention.

- **STALL** is returned by a function in response to an IN token or after the data phase of an OUT or in response to a PING transaction (see Figure 8-30 and Figure 8-38). STALL indicates that a function is unable to transmit or receive data, or that a control pipe request is not supported. The state of a function after returning a STALL (for any endpoint except the default endpoint) is undefined. The host is not permitted to return a STALL under any condition.

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

  The second case, known as "protocol stall," is detailed in Section 8.5.3. Protocol stall is unique to control pipes. Protocol stall differs from functional stall in meaning and duration. A protocol STALL is returned during the Data or Status stage of a control transfer, and the STALL condition terminates at the beginning of the next control transfer (Setup). The remainder of this section refers to the general case of a functional stall.

- **NYET** is a high-speed only handshake that is returned in two circumstances. It is returned by a high-speed endpoint as part of the PING protocol described later in this chapter. NYET may also be returned by a hub in response to a split-transaction when the full-/low-speed transaction has not yet been completed or the hub is otherwise not able to handle the split-transaction. See Chapter 11 for more details.

- **ERR** is a high-speed only handshake that is returned to allow a high-speed hub to report an error on a full-/low-speed bus. It is only returned by a high-speed hub as part of the split transaction protocol. See Chapter 11 for more details.

8.4.6 Handshake Responses

Transmitting and receiving functions must return handshakes based upon an order of precedence detailed in Table 8-4 through Table 8-6. Not all handshakes are allowed, depending on the transaction type and whether the handshake is being issued by a function or the host. Note that if an error occurs during the transmission of the token to the function, the function will not respond with any packets until the next token is received and successfully decoded.

8.4.6.1 Function Response to IN Transactions

Table 8-4 shows the possible responses a function may make in response to an IN token. If the function is unable to send data, due to a halt or a flow control condition, it issues a STALL or NAK handshake, respectively. If the function is able to issue data, it does so. If the received token is corrupted, the function returns no response.

Microchip Ex. 1018, p. 235
Microchip v. Aptiv
IPR2024-00494

Table 8-4. Function Responses to IN Transactions

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

8.4.6.2 Host Response to IN Transactions

Table 8-5 shows the host response to an IN transaction. The host is able to return only one type of handshake: ACK. If the host receives a corrupted data packet, it discards the data and issues no response. If the host cannot accept data from a function, (due to problems such as internal buffer overrun) this condition is considered to be an error and the host returns no response. If the host is able to accept data and the data packet is received error-free, the host accepts the data and issues an ACK handshake.

Table 8-5. Host Responses to IN Transactions

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

8.4.6.3 Function Response to an OUT Transaction

Handshake responses for an OUT transaction are shown in Table 8-6. Assuming successful token decode, a function, upon receiving a data packet, may return any one of the three handshake types. If the data packet was corrupted, the function returns no handshake. If the data packet was received error-free and the function's receiving endpoint is halted, the function returns STALL. If the transaction is maintaining sequence bit synchronization and a mismatch is detected (refer to Section 8.6 for details), then the function returns ACK and discards the data. If the function can accept the data and has received the data error-free, it returns ACK. If the function cannot accept the data packet due to flow control reasons, it returns NAK.

Microchip Ex. 1018, p. 236
Microchip v. Aptiv
IPR2024-00494

Table 8-6. Function Responses to OUT Transactions in Order of Precedence

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

8.4.6.4 Function Response to a SETUP Transaction

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

8.5 Transaction Packet Sequences

The packets that comprise a transaction varies depending on the endpoint type. There are four endpoint types: bulk, control, interrupt, and isochronous.

A host controller and device each require different state machines to correctly sequence each type of transaction. Figures in the following sections show state machines that define the correct sequencing of packets within a transaction of each type. The diagrams should not be taken as a required implementation, but to specify the required behavior.

Figure 8-17 shows the legend for the state machine diagrams. A circle with a three-line border indicates a reference to another (hierarchical) state machine. A circle with a two-line border indicates an initial state. A circle with a single-line border represents a simple state.

Microchip Ex. 1018, p. 237
Microchip v. Aptiv
IPR2024-00494

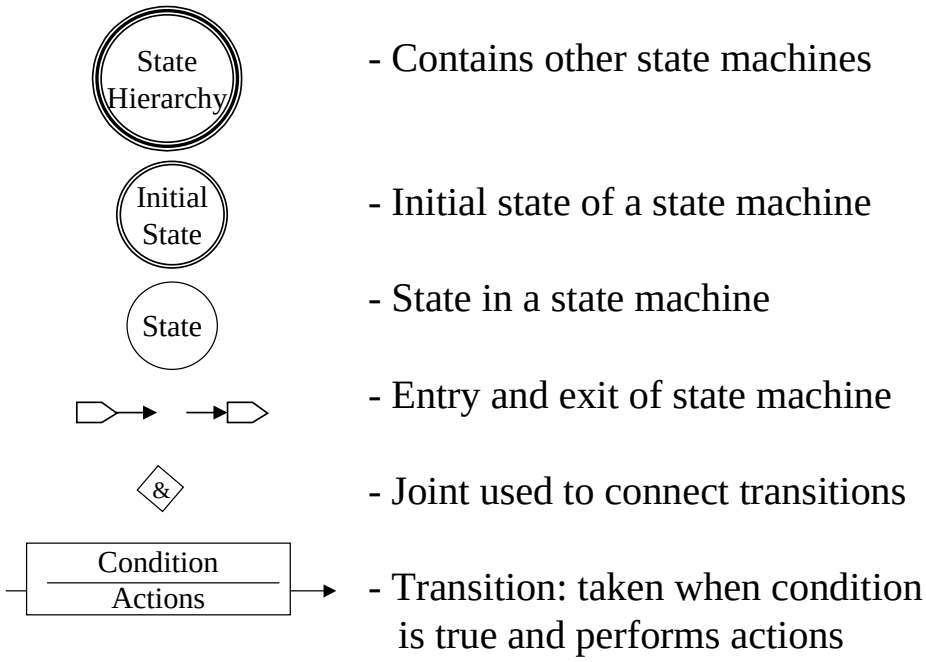


Figure 8-17. Legend for State Machines

The "tab" shapes with arrows are the entry or exit (respectively in the legend) to/from the state machine. The entry/exit relates to another state in a state machine at a higher level in the state machine hierarchy.

A diamond (joint) is used to join several transitions to a common point. A joint allows a single input transition with multiple output transitions or multiple input transitions and a single output transition. All conditions on the transitions of a path involving a joint must be true for the path to be taken. A path is simply a sequence of transitions involving one or more joints.

A transition is labeled with a block with a line in the middle separating the (upper) condition and the (lower) actions. The condition is required to be true to take the transition. The syntax for actions and conditions is VHDL. The actions are performed if the transition is taken. A circle includes a name in bold and optionally one or more actions that are performed upon entry to the state.

The host controller and device state machines are in a context as shown in Figure 8-18. The host controller determines the next transaction to run for an endpoint and issues a command (HC_cmd) to the host controller state machines. This causes the host controller state machines to issue one or more packets to move over the downstream bus (HSD1).

The device receives these packets from the bus (HSD2), reacts to the received packet, and interacts with its function(s) via the state of the corresponding endpoint (in the EP_array). Then the device may respond with a packet on the upstream bus (HSU1). The host controller state machines can receive a packet from the bus (HSU2) and provide a result of the transaction back to the host controller (HC_resp). The details of what packets are sent on the bus is determined by the transfer type for the endpoint and what bus activity the state machines observe.

The state machines are presented in a hierarchical form. Figure 8-19 shows the top level state machines for the host controller. The non-split transactions are presented in the remainder of this chapter. The split transaction state machines (HC_Do_start and HC_Do_complete) are described and shown in Chapter 11.

Microchip Ex. 1018, p. 238
Microchip v. Aptiv
IPR2024-00494

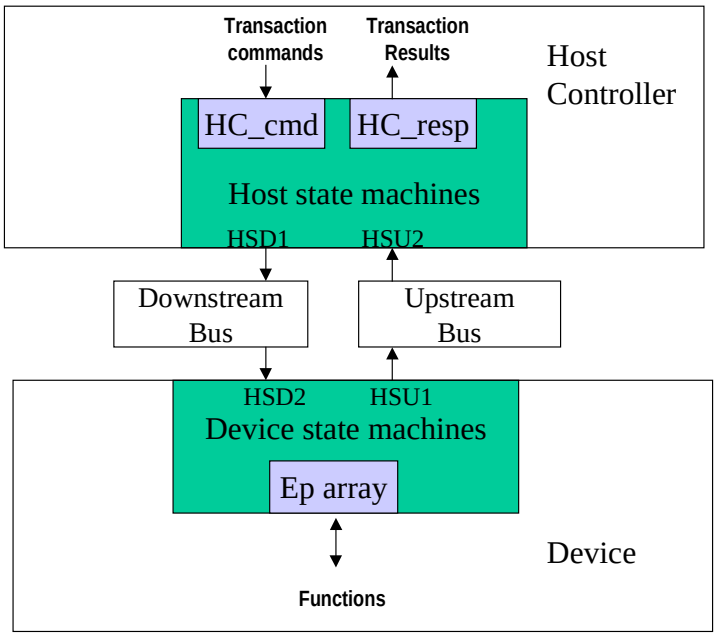


Figure 8-18. State Machine Context Overview

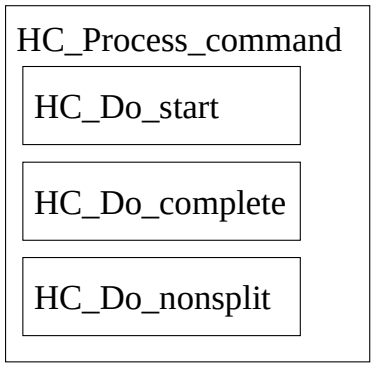


Figure 8-19. Host Controller Top Level Transaction State Machine Hierarchy Overview

The host controller state machines are located in the host controller. The host controller causes packets to be issued downstream (labeled as HSD1) and it receives upstream packets (labeled as HSU2).

The device state machines are located in the device. The device causes packets to be issued upstream (labeled as HSU1) and it receives downstream packets (labeled as HSD2).

The host controller has commands that tell it what transaction to issue next for an endpoint. The host controller tracks transactions for several endpoints. The host controller state machines sequence to determine what the host controller needs to do next for the current endpoint. The device has a state for each of its endpoints. The device state machines sequence to determine what reaction the device has to a transaction.

The appendix includes some declarations that were used in constructing the state machines and may be useful in understanding additional details of the state machines. There are several pseudo-code procedures and functions for conditions and actions. Simple descriptions of them are also included in the appendix.

Figure 8-20 shows an overview of the overall state machine hierarchy for the host controller for the non-split transaction types. Figure 8-21 shows the hierarchy of the device state machines. The state machines

Microchip Ex. 1018, p. 239
Microchip v. Aptiv
IPR2024-00494

common to endpoint types are presented first. The lowest level endpoint type specific state machines are presented in each following endpoint type section.

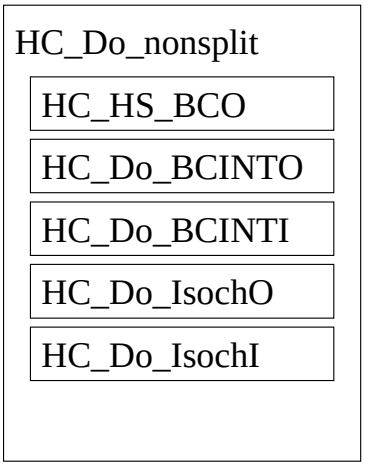


Figure 8-20. Host Controller Non-split Transaction State Machine Hierarchy Overview

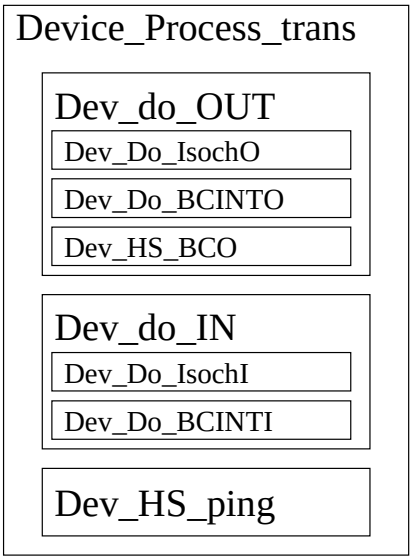


Figure 8-21. Device Transaction State Machine Hierarchy Overview

Microchip Ex. 1018, p. 240
Microchip v. Aptiv
IPR2024-00494

Global Actions **Concurrent Statements** **Architecture Declarations** **Signals Status** **State Register Statements**

Package List

ieee std_logic_1164
ieee numeric_std
usb2statemachines behav_package

```
SIGNAL  SCOPE DEFAULT
hsu1    OUT   (BULK,NAK,0,0,ok,in_dir,TRUE,ALLDATA,FALSE,FA
device  INT   '0'
token   INT   '0'
```

Process Declarations

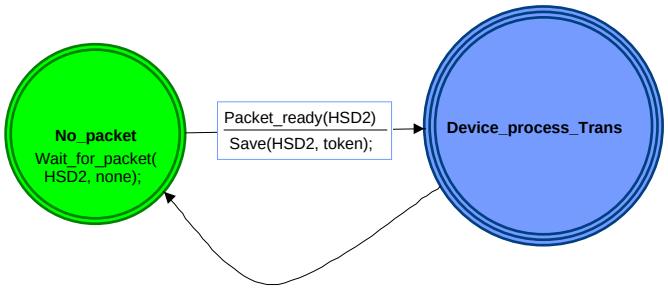


Figure 8-22. Device Top Level State Machine

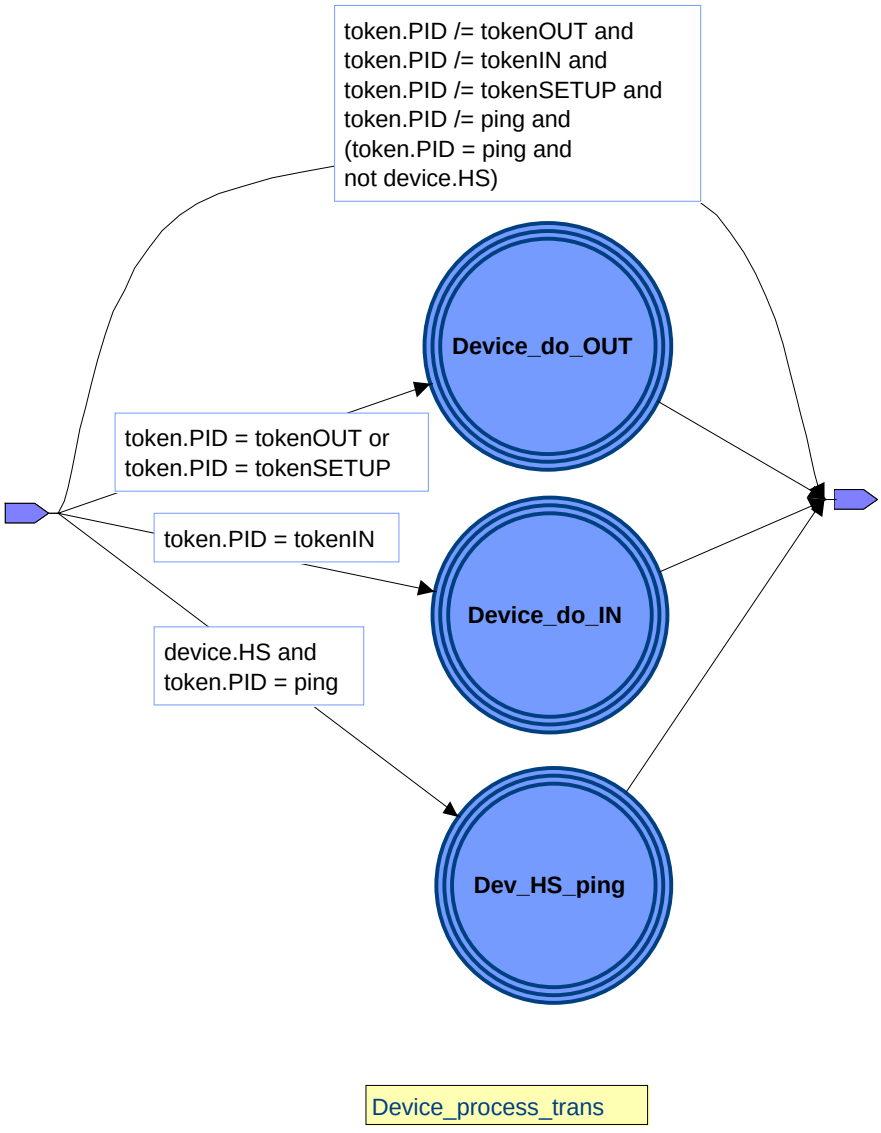


Figure 8-23. Device_process_Trans State Machine

Microchip Ex. 1018, p. 241
Microchip v. Aptiv
IPR2024-00494

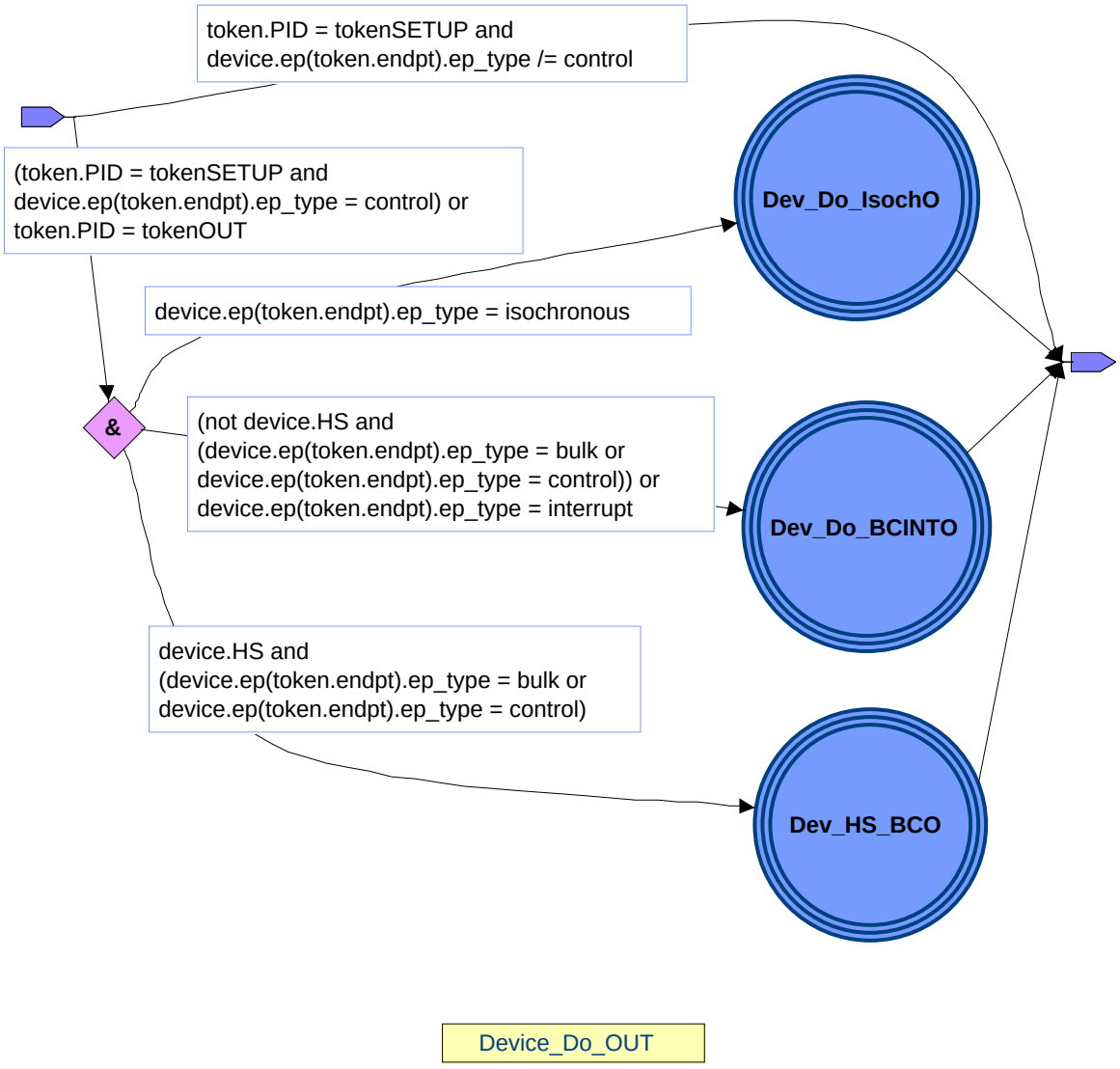


Figure 8-24. Dev_do_OUT State Machine

Microchip Ex. 1018, p. 242
Microchip v. Aptiv
IPR2024-00494

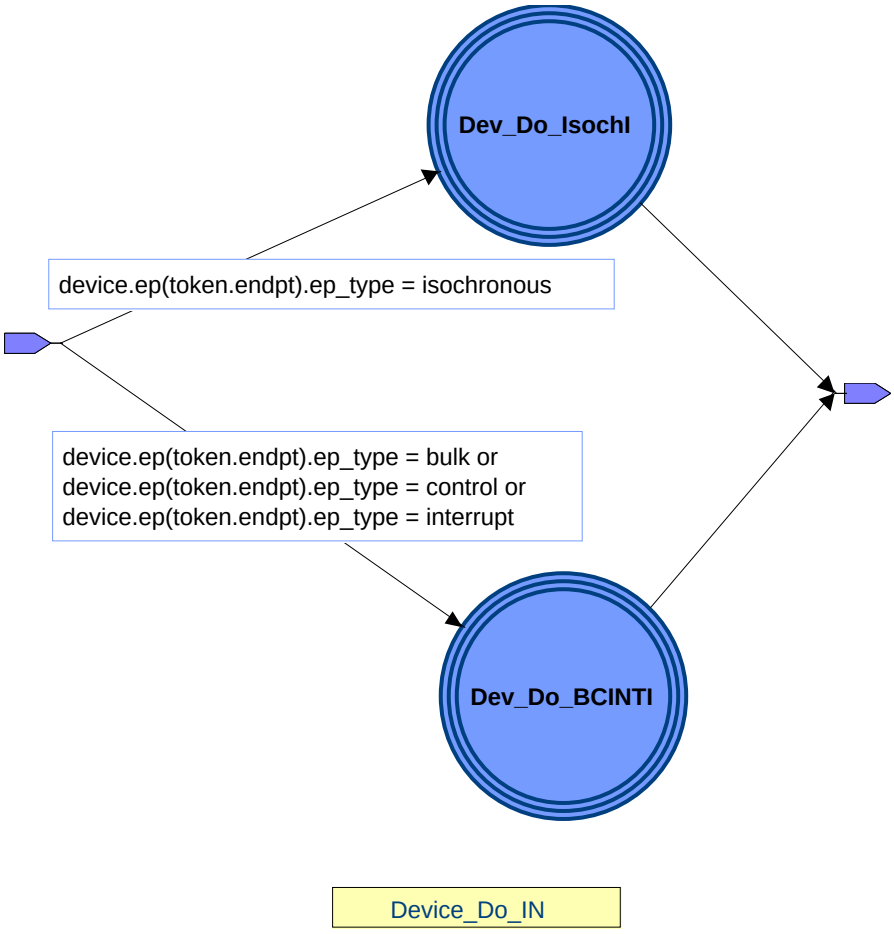


Figure 8-25. Dev_do_IN State Machine

Microchip Ex. 1018, p. 243
Microchip v. Aptiv
IPR2024-00494

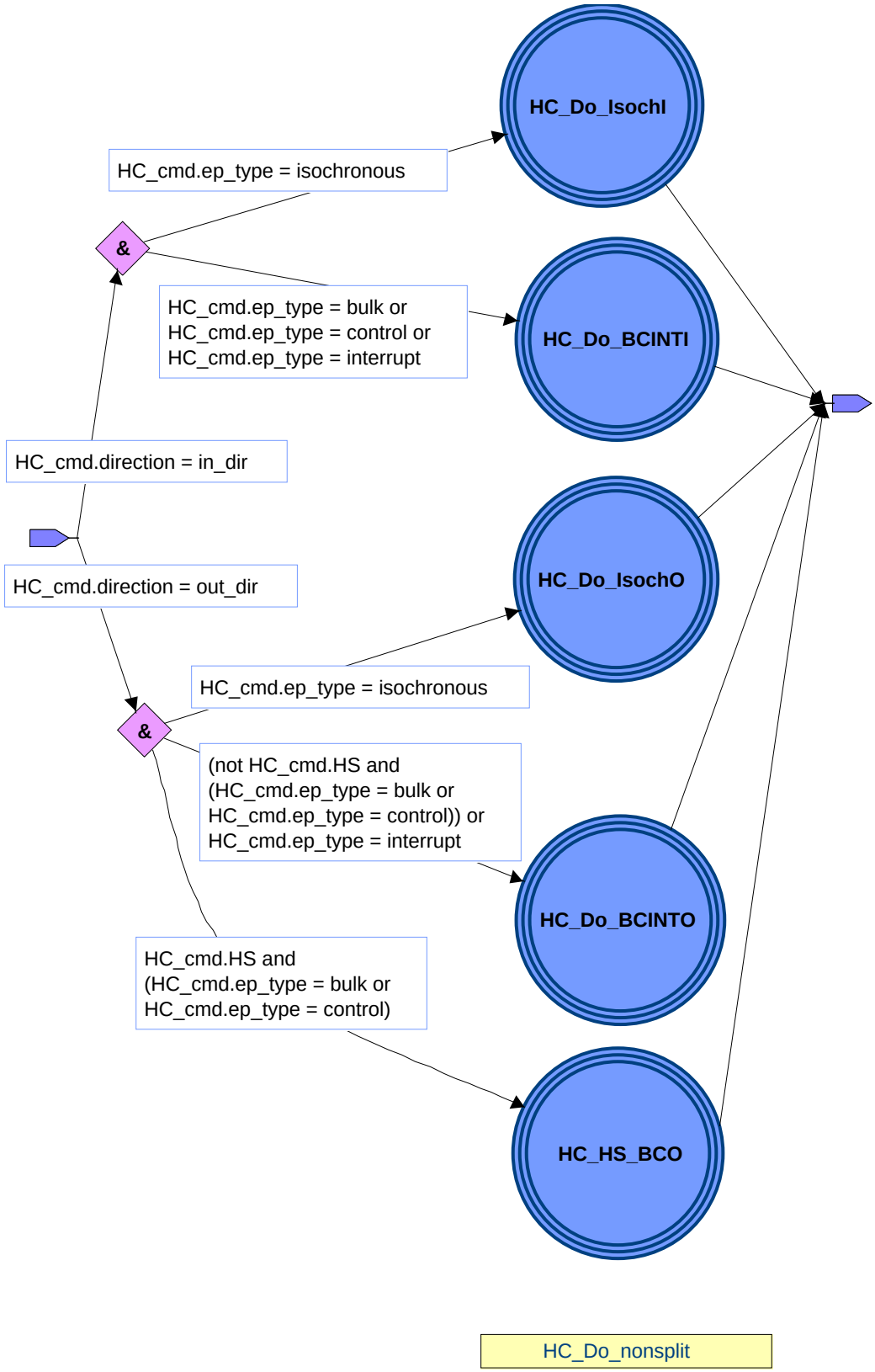


Figure 8-26. HC_Do_nonsplit State Machine

Microchip Ex. 1018, p. 244
Microchip v. Aptiv
IPR2024-00494

8.5.1 NAK Limiting via Ping Flow Control

Full-/low-speed devices can have bulk/control endpoints that take time to process their data and, therefore, respond to OUT transactions with a NAK handshake. This handshake response indicates that the endpoint did not accept the data because it did not have space for the data. The host controller is expected to retry the transaction at some future time when the endpoint has space available. Unfortunately, by the time the endpoint NAKs, most of the full-/low-speed bus time for the transaction had been used. This means that the full-/low-speed bus has poor utilization when there is a high frequency of NAK'd OUT transactions.

High-speed devices must support an improved NAK mechanism for Bulk OUT and Control endpoints and transactions. Control endpoints must support this protocol for an OUT transaction in the data and status stages. The control Setup stage must not support the PING protocol.

This mechanism allows the device to tell the host controller whether it has sufficient endpoint space for the next OUT transaction. If the device endpoint does not have space, the host controller can choose to delay a transaction attempt for this endpoint and instead try some other transaction. This can lead to improved bus utilization. The mechanism avoids using bus time to send data until the host controller knows that the endpoint has space for the data.

The host controller queries the high-speed device endpoint with a PING special token. The PING special token packet is a normal token packet as shown in Figure 8-5. The endpoint either responds to the PING with a NAK or an ACK handshake.

A NAK handshake indicates that the endpoint does not have space for a *wMaxPacketSize* data payload. The host controller will retry the PING at some future time to query the endpoint again. A device can respond to a PING with a NAK for long periods of time. A NAK response is not a reason for the host controller to retire a transfer request. If a device responds with a NAK in a (micro)frame, the host controller may choose to issue the next transaction in the next *bInterval* specified for the endpoint. However, the device must be prepared to receive PINGs as sequential transactions, e.g., one immediately after the other.

An ACK handshake indicates the endpoint has space for a *wMaxPacketSize* data payload. The host controller must generate an OUT transaction with a DATA phase as the next transaction to the endpoint. The host controller may generate other transactions to other devices or endpoints before the OUT/DATA transaction for this endpoint.

If the endpoint responds to the OUT/DATA transaction with an ACK handshake, this means the endpoint accepted the data successfully and has room for another *wMaxPacketSize* data payload. The host controller continues with OUT/DATA transactions (which are not required to be the next transactions on the bus) as long as it has transactions to generate.

If the endpoint instead responds to the OUT/DATA transaction with a NYET handshake, this means that the endpoint accepted the data but does not have room for another *wMaxPacketSize* data payload. The host controller must return to using a PING token until the endpoint indicates it has space.

Microchip Ex. 1018, p. 245
Microchip v. Aptiv
IPR2024-00494

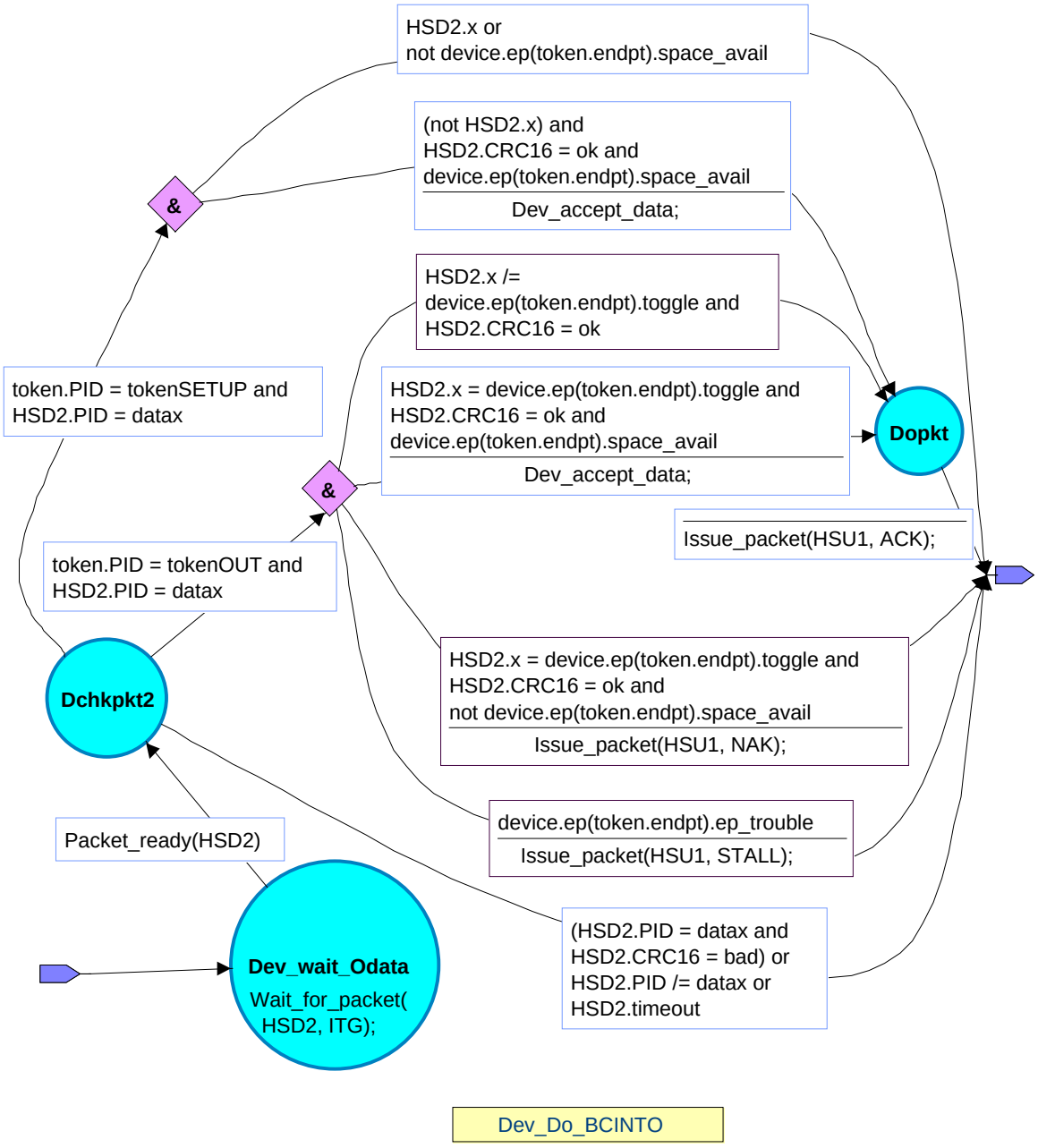


Figure 8-27. Host High-speed Bulk OUT/Control Ping State Machine

8.5.1.1 NAK Responses to OUT/DATA During PING Protocol

The endpoint may also respond to the OUT/DATA transaction with a NAK handshake. This means that the endpoint did not accept the data and does not have space for a *wMaxPacketSize* data payload at this time. The host controller must return to using a PING token until the endpoint indicates it has space.

A NAK response is expected to be an unusual occurrence. A high-speed bulk/control endpoint must specify its maximum NAK rate in its endpoint descriptor. The endpoint is allowed to NAK at most one time each *bInterval* period. A NAK suggests that the endpoint responded to a previous OUT or PING with an inappropriate handshake, or that the endpoint transitioned into a state where it (temporarily) could not

Microchip Ex. 1018, p. 246
Microchip v. Aptiv
IPR2024-00494

accept data. An endpoint can use a *bInterval* of zero to indicate that it never NAKs. An endpoint must always be able to accept a PING from the host, even if it never NAKs.

If a timeout occurs after the data phase, the host must return to using a PING token. Note that a transition back to the PING state does not affect the data toggle state of the transaction data phase.

Figure 8-27 shows the host controller state machine for the interactions and transitions between PING and OUT/DATA tokens and the allowed ACK, NAK, and NYET handshakes for the PING mechanism.

Figure 8-29 shows the device endpoint state machine for PING based on the buffer space the endpoint has available.

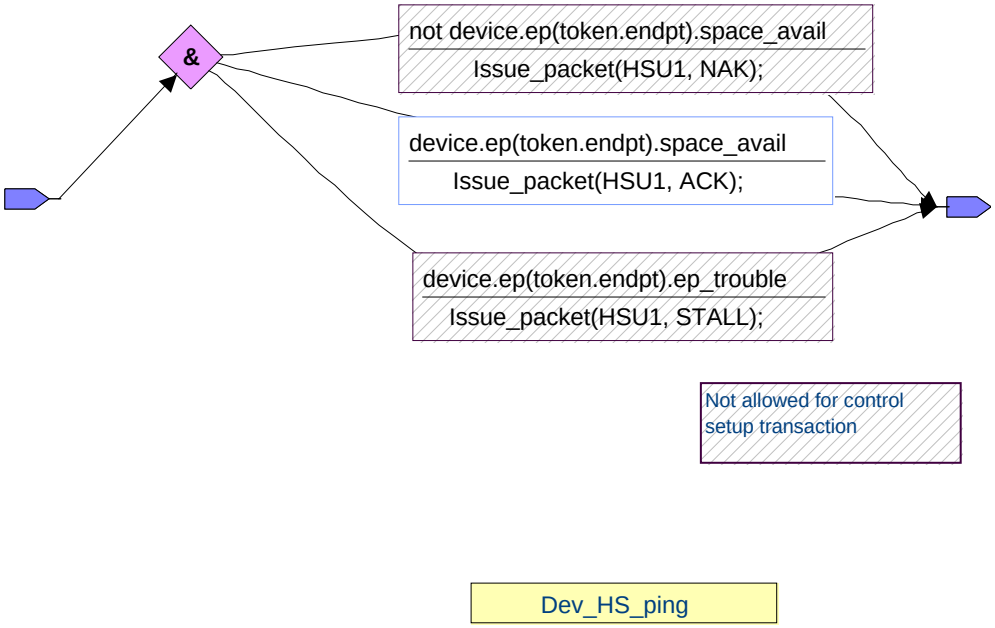


Figure 8-28. Dev_HS_ping State Machine

Microchip Ex. 1018, p. 247
Microchip v. Aptiv
IPR2024-00494

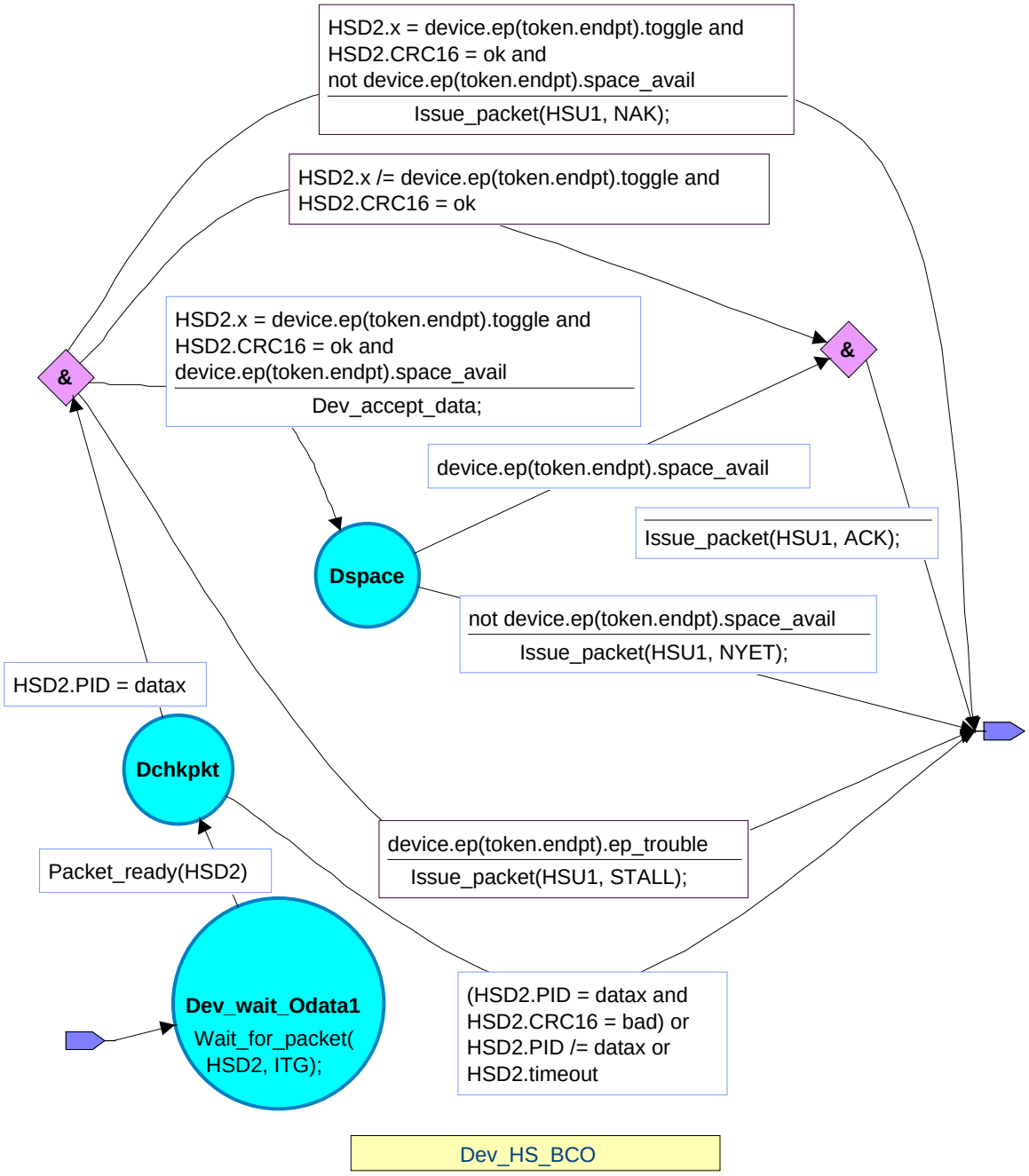


Figure 8-29. Device High-speed Bulk OUT /Control State Machine

Full-/low-speed devices/endpoints must not support the PING protocol. Host controllers must not support the PING protocol for full-/low-speed devices.

Note: The PING protocol is also not included as part of the split-transaction protocol definition. Some split-transactions have equivalent flow control without using PING. Other split-transactions will not benefit from PING as defined. In any case, split-transactions that can return a NAK handshake have small data payloads which should have minor high-speed bus impact. Hubs must support PING on their control endpoint, but PING is not defined for the split-transactions that are used to communicate with full-/low-speed devices supported by a hub.

Microchip Ex. 1018, p. 248
Microchip v. Aptiv
IPR2024-00494

8.5.2 Bulk Transactions

Bulk transaction types are characterized by the ability to guarantee error-free delivery of data between the host and a function by means of error detection and retry. Bulk transactions use a three-phase transaction consisting of token, data, and handshake packets as shown in Figure 8-30. Under certain flow control and halt conditions, the data phase may be replaced with a handshake resulting in a two-phase transaction in which no data is transmitted. The PING and NYET packets must only be used with devices operating at high-speed.

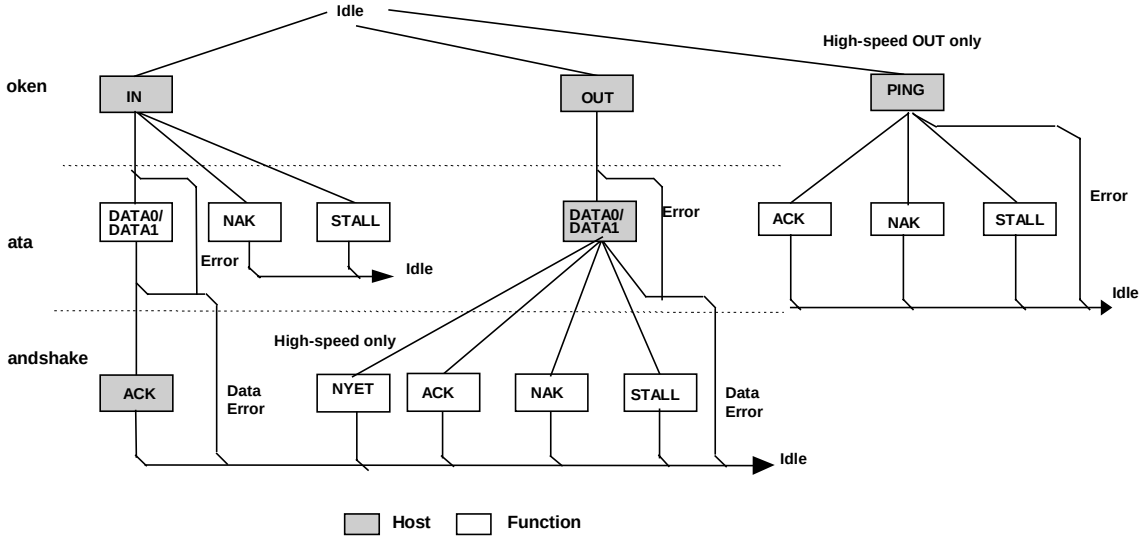


Figure 8-30. Bulk Transaction Format

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

When the host is ready to transmit bulk data, it first issues an OUT token packet followed by a data packet (or PING special token packet, see Section 8.5.1). If the data is received without error by the function, it will return one of three (or four including NYET, for a device operating at high-speed) handshakes:

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

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

Figure 8-31 and Figure 8-32 show the host and device state machines respectively for bulk, control, and interrupt OUT full/low-speed transactions. Figure 8-27, Figure 8-28, and Figure 8-29 show the state machines for high-speed transactions. Figure 8-33 and Figure 8-34 show the host and device state machines respectively for bulk, control, and interrupt IN transactions.

Microchip Ex. 1018, p. 249
Microchip v. Aptiv
IPR2024-00494

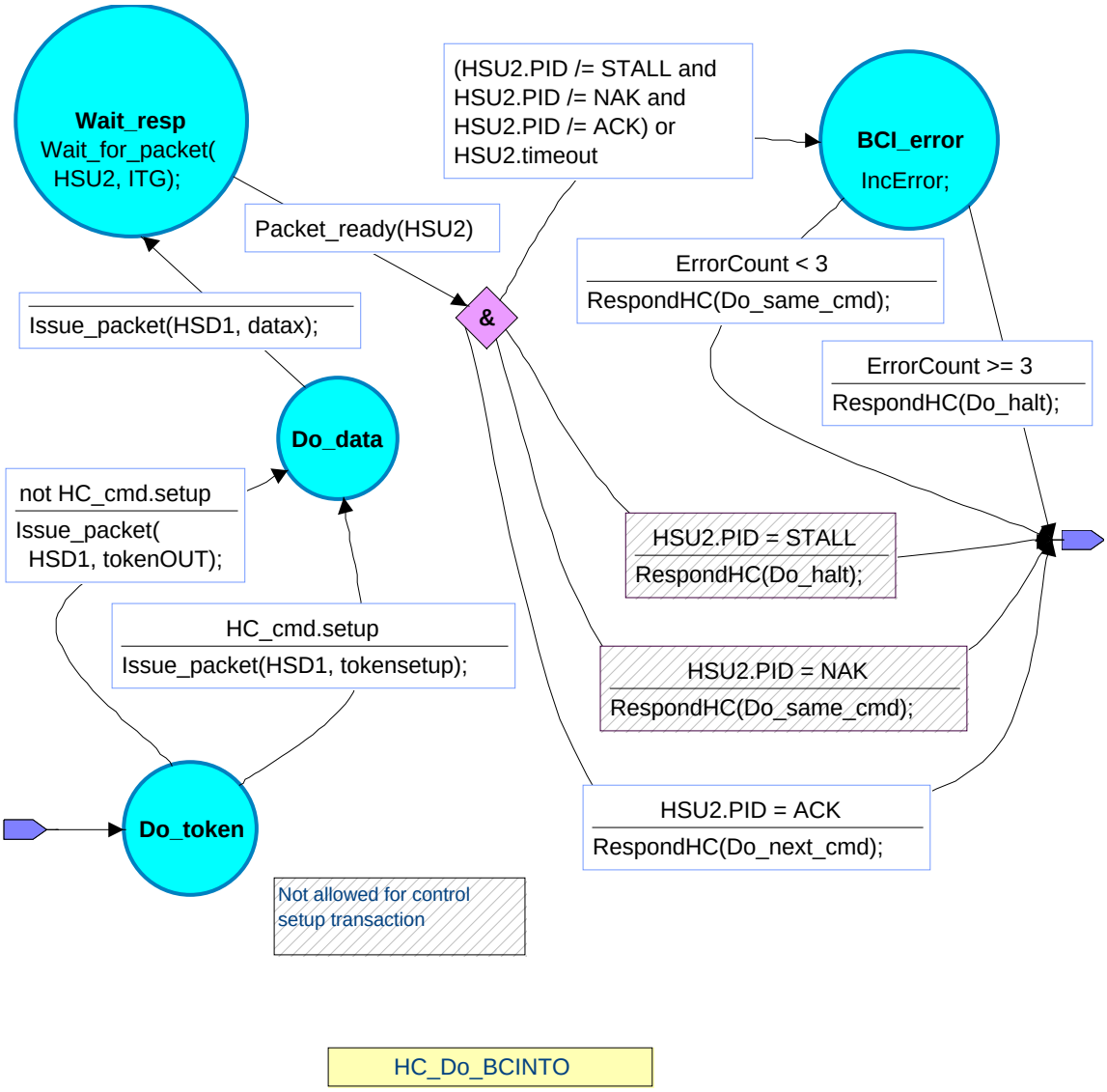


Figure 8-31. Bulk/Control/Interrupt OUT Transaction Host State Machine

Microchip Ex. 1018, p. 250
Microchip v. Aptiv
IPR2024-00494

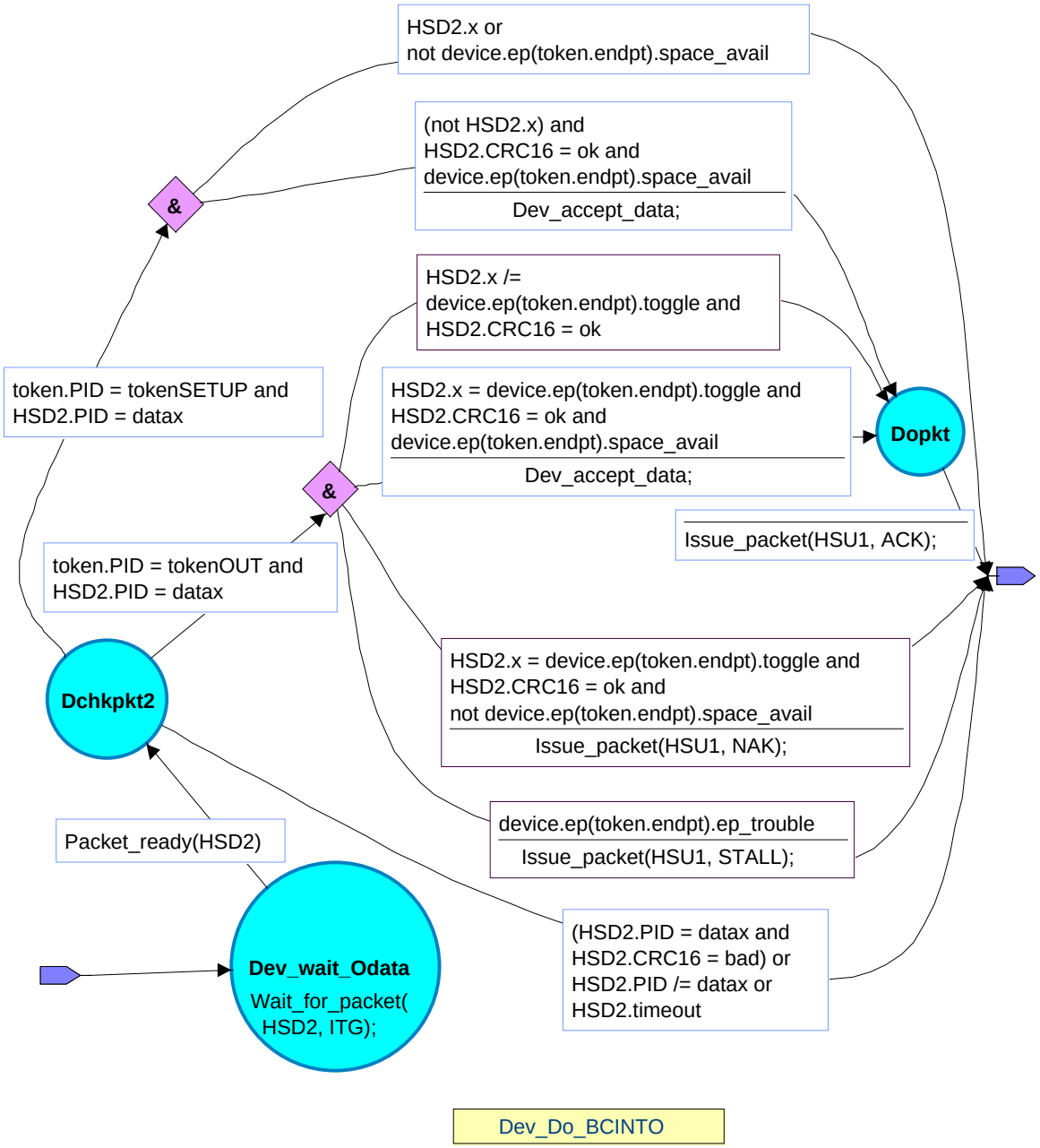


Figure 8-32. Bulk/Control/Interrupt OUT Transaction Device State Machine

Microchip Ex. 1018, p. 251
Microchip v. Aptiv
IPR2024-00494

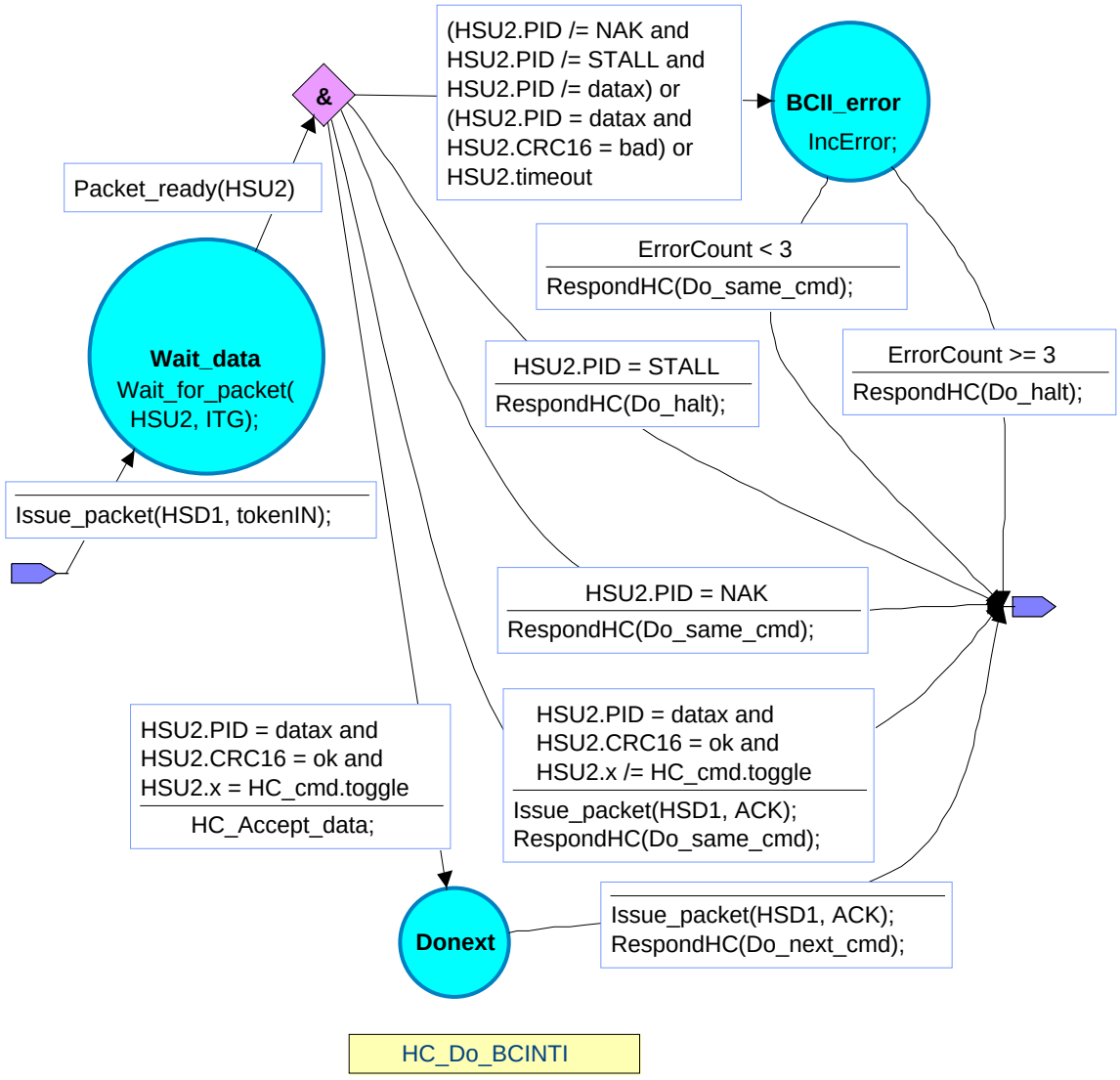


Figure 8-33. Bulk/Control/Interrupt IN Transaction Host State Machine

Microchip Ex. 1018, p. 252
Microchip v. Aptiv
IPR2024-00494

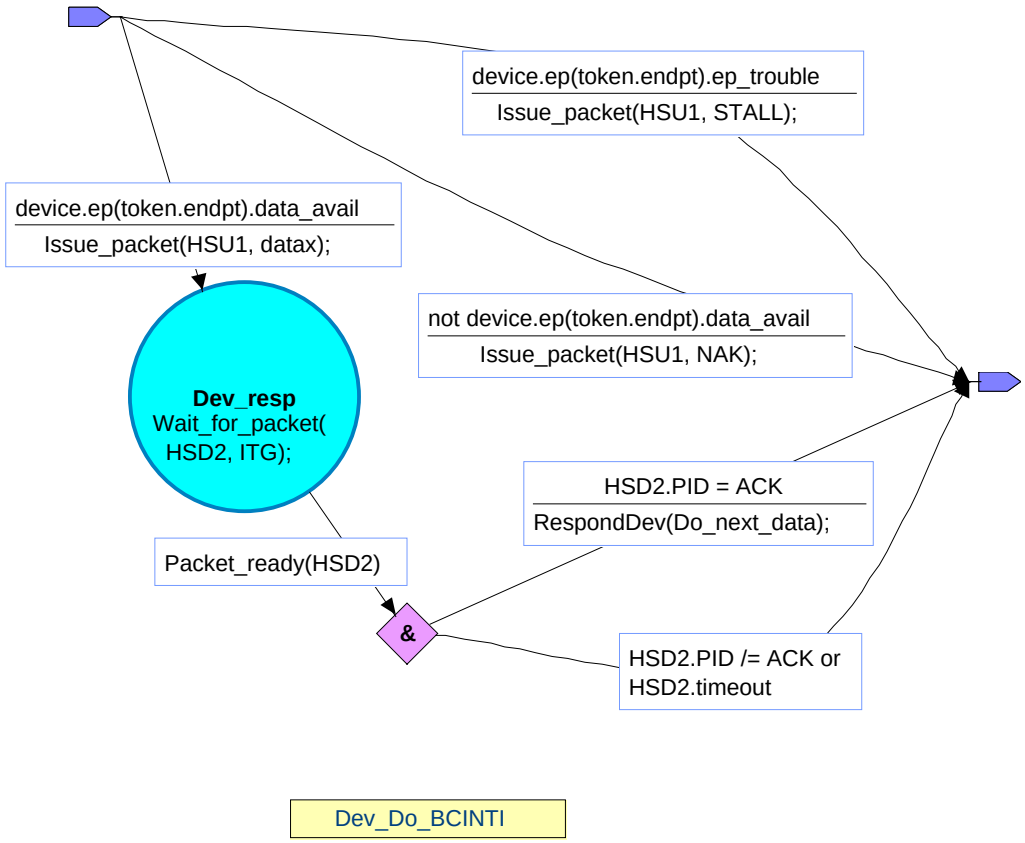


Figure 8-34. Bulk/Control/Interrupt IN Transaction Device State Machine

Figure 8-35 shows the sequence bit and data PID usage for bulk reads and writes. Data packet synchronization is achieved via use of the data sequence toggle bits and the DATA0/DATA1 PIDs. A bulk endpoint's toggle sequence is initialized to DATA0 when the endpoint experiences any configuration event (configuration events are explained in Sections 9.1.1.5 and 9.4.5). Data toggle on an endpoint is NOT initialized as the direct result of a short packet transfer or the retirement of an IRP.

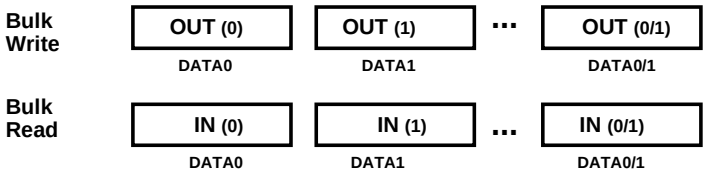


Figure 8-35. Bulk Reads and Writes

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

8.5.3 Control Transfers

Control transfers minimally have two transaction stages: Setup and Status. A control transfer may optionally contain a Data stage between the Setup and Status stages. During the Setup stage, a SETUP transaction is used to transmit information to the control endpoint of a function. SETUP transactions are similar in format to an OUT but use a SETUP rather than an OUT PID. Figure 8-36 shows the SETUP transaction format. A SETUP always uses a DATA0 PID for the data field of the SETUP transaction. The

Microchip Ex. 1018, p. 253
Microchip v. Aptiv
IPR2024-00494

function receiving a SETUP must accept the SETUP data and respond with ACK; if the data is corrupted, discard the data and return no handshake.

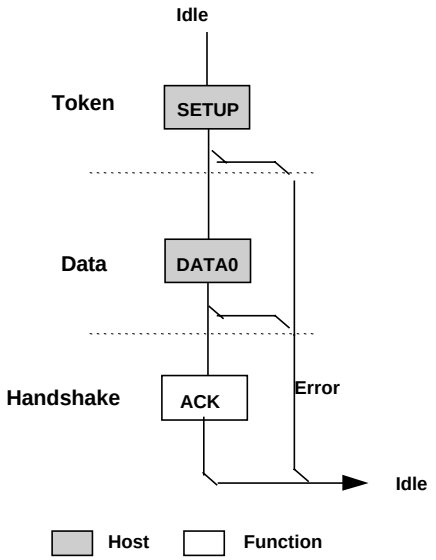


Figure 8-36. Control SETUP Transaction

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

The Status stage of a control transfer is the last transaction in the sequence. The status stage transactions follow the same protocol sequence as bulk transactions. Status stage for devices operating at high-speed also includes the PING protocol. A Status stage is delineated by a change in direction of data flow from the previous stage and always uses a DATA1 PID. If, for example, the Data stage consists of OUTs, the status is a single IN transaction. If the control sequence has no Data stage, then it consists of a Setup stage followed by a Status stage consisting of an IN transaction.

Figure 8-37 shows the transaction order, the data sequence bit value, and the data PID types for control read and write sequences. The sequence bits are displayed in parentheses.

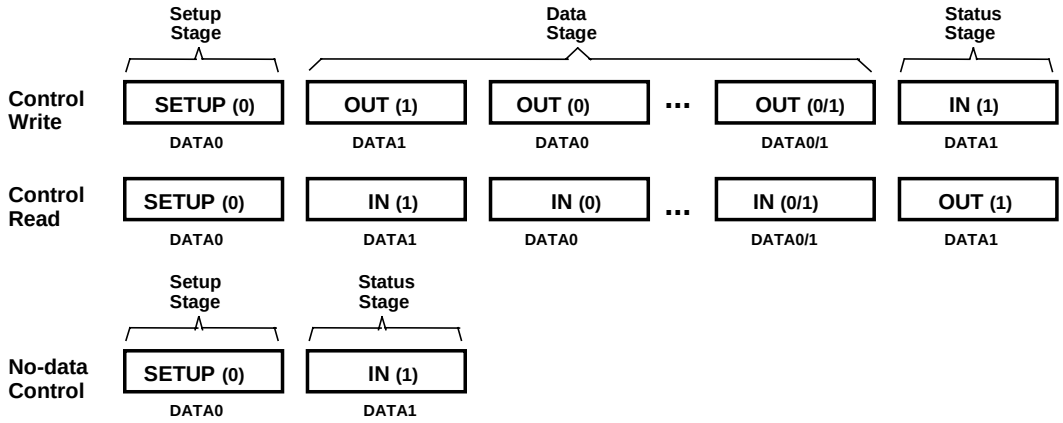


Figure 8-37. Control Read and Write Sequences

Microchip Ex. 1018, p. 254
Microchip v. Aptiv
IPR2024-00494

When a STALL handshake is sent by a control endpoint in either the Data or Status stages of a control transfer, a STALL handshake must be returned on all succeeding accesses to that endpoint until a SETUP PID is received. The endpoint is not required to return a STALL handshake after it receives a subsequent SETUP PID. For the default endpoint, if an ACK handshake is returned for the SETUP transaction, the host expects that the endpoint has automatically recovered from the condition that caused the STALL and the endpoint must operate normally.

8.5.3.1 Reporting Status Results

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

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

Status reporting is always in the function-to-host direction. Table 8-7 summarizes the type of responses required for each. Control write transfers return status information in the data phase of the Status stage transaction. Control read transfers return status information in the handshake phase of a Status stage transaction, after the host has issued a zero-length data packet during the previous data phase.

Table 8-7. Status Stage Responses

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

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

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

If during a Data stage a command pipe is sent more data or is requested to return more data than was indicated in the Setup stage (see Section 8.5.3.2), it should return STALL. If a control pipe returns STALL during the Data stage, there will be no Status stage for that control transfer.

Microchip Ex. 1018, p. 255
Microchip v. Aptiv
IPR2024-00494

8.5.3.2 Variable-length Data Stage

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

8.5.3.3 Error Handling on the Last Data Transaction

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

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

8.5.3.4 STALL Handshakes Returned by Control Pipes

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

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

8.5.4 Interrupt Transactions

Interrupt transactions may consist of IN or OUT transfers. Upon receipt of an IN token, a function may return data, NAK, or STALL. If the endpoint has no new interrupt information to return (i.e., no interrupt is pending), the function returns a NAK handshake during the data phase. If the *Halt* feature is set for the interrupt endpoint, the function will return a STALL handshake. If an interrupt is pending, the function returns the interrupt information as a data packet. The host, in response to receipt of the data packet, issues either an ACK handshake if data was received error-free or returns no handshake if the data packet was received corrupted. Figure 8-38 shows the interrupt transaction format.

Section 5.9.1 contains additional information about high-speed, high-bandwidth interrupt endpoints. Such endpoints use multiple transactions in a microframe as defined in that section. Each transaction for a high-bandwidth endpoint follows the transaction format shown in Figure 8-38.

Microchip Ex. 1018, p. 256
Microchip v. Aptiv
IPR2024-00494

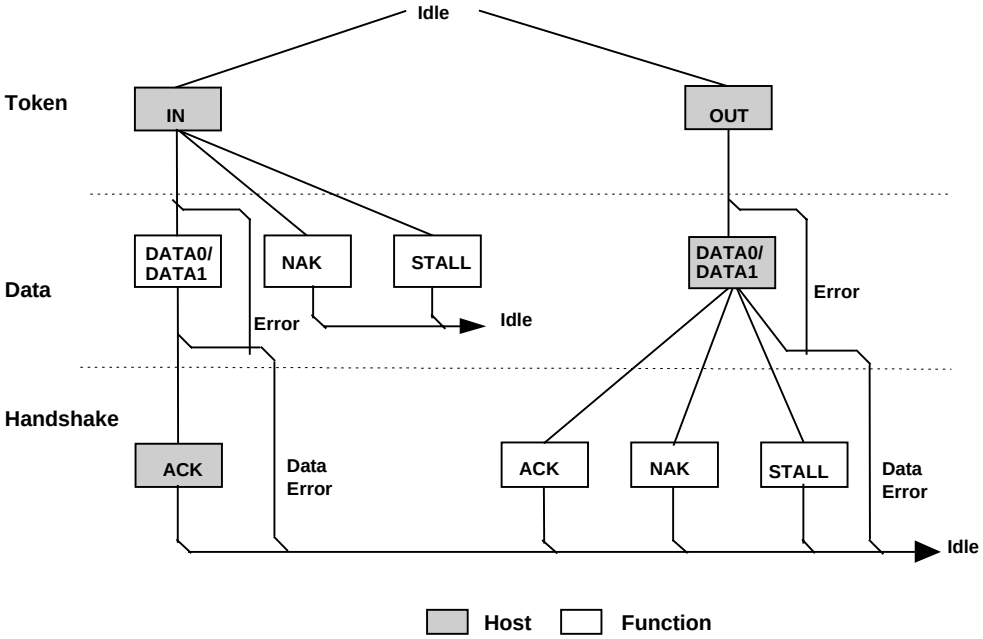


Figure 8-38. Interrupt Transaction Format

When an endpoint is using the interrupt transfer mechanism for actual interrupt data, the data toggle protocol must be followed. This allows the function to know that the data has been received by the host and the event condition may be cleared. This "guaranteed" delivery of events allows the function to only send the interrupt information until it has been received by the host rather than having to send the interrupt data every time the function is polled and until the USB System Software clears the interrupt condition. When used in the toggle mode, an interrupt endpoint is initialized to the DATA0 PID by any configuration event on the endpoint and behaves the same as the bulk transactions shown in Figure 8-35.

8.5.5 Isochronous Transactions

Isochronous transactions have a token and data phase, but no handshake phase, as shown in Figure 8-39. The host issues either an IN or an OUT token followed by the data phase in which the endpoint (for INs) or the host (for OUTs) transmits data. Isochronous transactions do not support a handshake phase or retry capability.

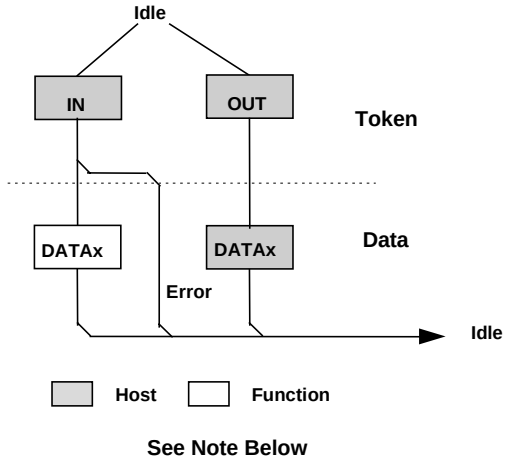


Figure 8-39. Isochronous Transaction Format

Microchip Ex. 1018, p. 257
Microchip v. Aptiv
IPR2024-00494

Note: A full-speed device or Host Controller should be able to accept either DATA0 or DATA1 PIDs in data packets. A full-speed device or Host Controller should only send DATA0 PIDs in data packets. A high-speed Host Controller must be able to accept and send DATA0, DATA1, DATA2, or MDATA PIDs in data packets. A high-speed device with at most 1 transaction per microframe must only send DATA0 PIDs in data packets. A high-speed device with high-bandwith endpoints (e.g., one that has more than 1 transaction per microframe) must be able to accept and/or send DATA0, DATA1, DATA2, or MDATA PIDs in data packets.

Full-speed isochronous transactions do not support toggle sequencing. High-speed isochronous transactions with a single transaction per microframe do not support toggle sequencing. High bandwidth, high-speed isochronous transactions support data PID sequencing (see Section 5.9.1 for more details).

Figure 8-40 and Figure 8-41 show the host and device state machines respectively for isochronous OUT transactions. Figure 8-42 and Figure 8-43 show the host and device state machines respectively for isochronous IN transactions.

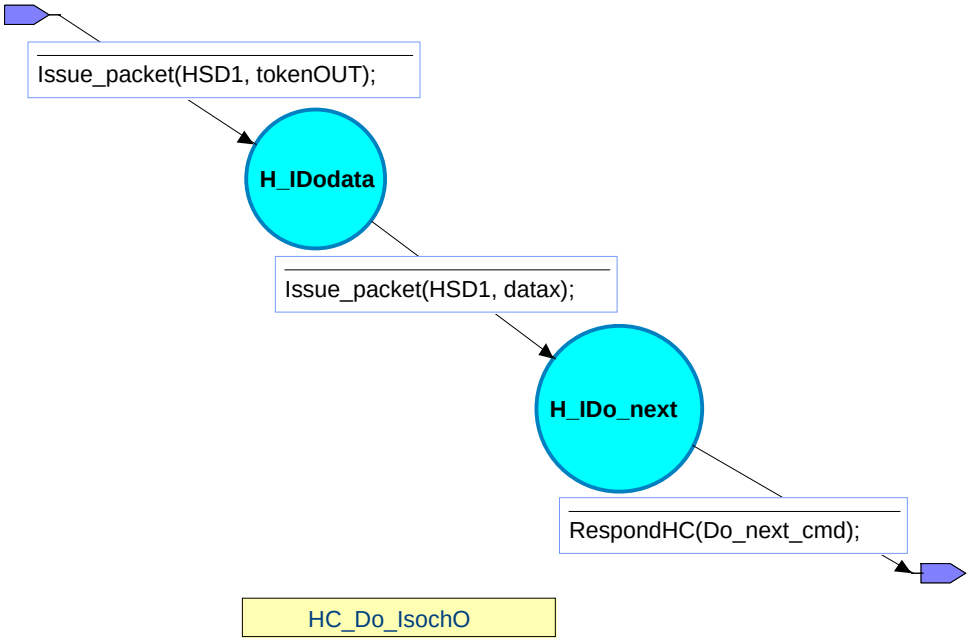


Figure 8-40. Isochronous OUT Transaction Host State Machine

Microchip Ex. 1018, p. 258
Microchip v. Aptiv
IPR2024-00494

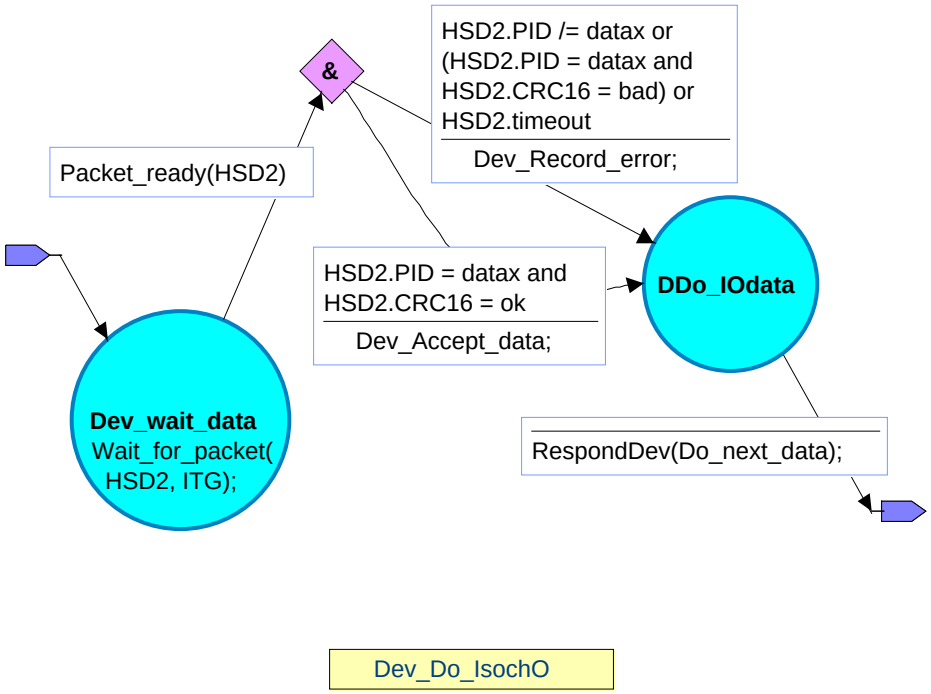


Figure 8-41. Isochronous OUT Transaction Device State Machine

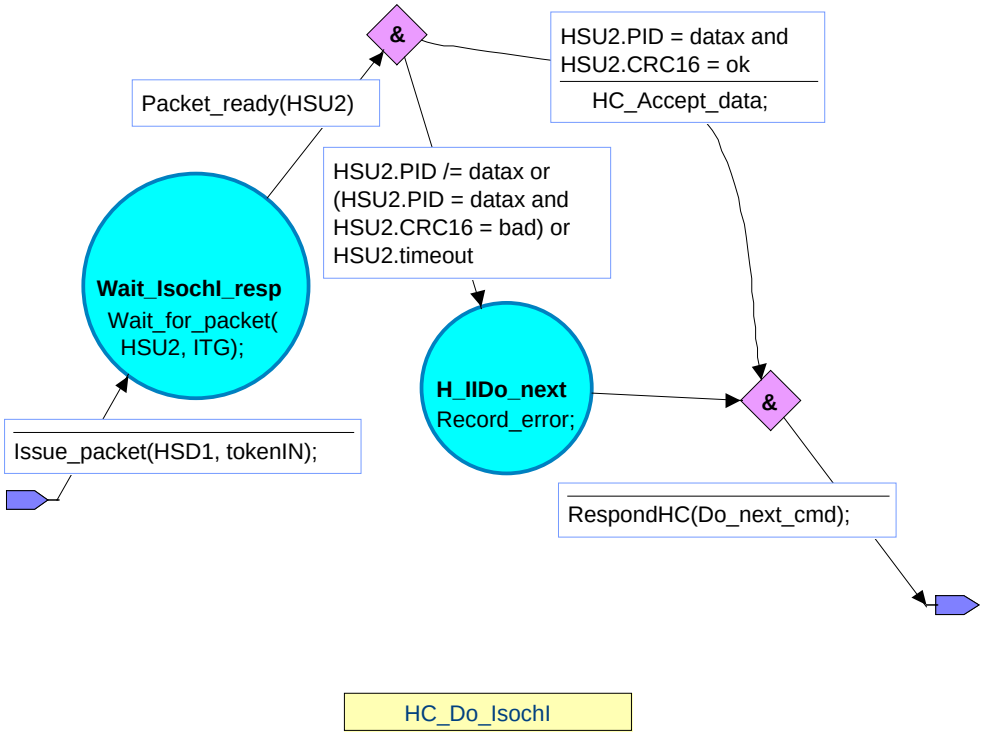


Figure 8-42. Isochronous IN Transaction Host State Machine

Microchip Ex. 1018, p. 259
Microchip v. Aptiv
IPR2024-00494

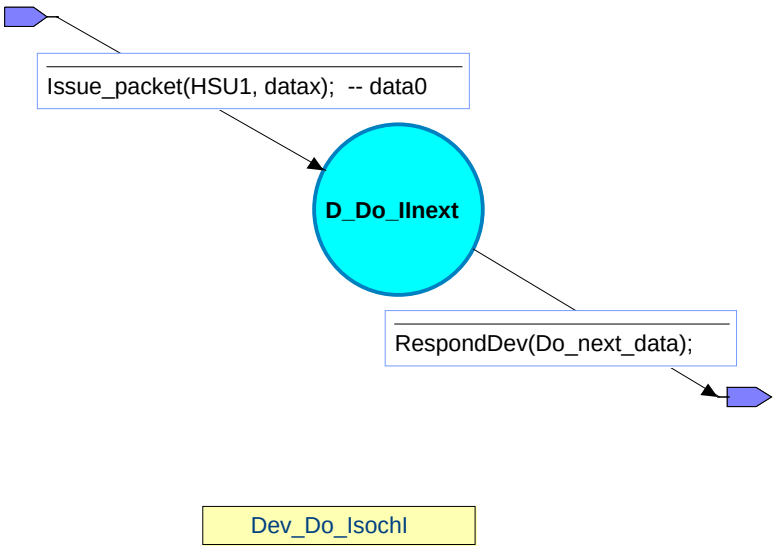


Figure 8-43. Isochronous IN Transaction Device State Machine

8.6 Data Toggle Synchronization and Retry

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

The state machines contained in this chapter and in Chapter 11 describe data toggle synchronization in a more compact form. Instead of explicitly identifying DATA0 and DATA1, it uses a value “DATAx” to represent either/both DATA0/DATA1 PIDs. In some cases where the specific data PID is important, another variable labeled “x” is used that has the value 0 for DATA0 and 1 for DATA1.

High-speed, high-bandwidth isochronous and interrupt endpoints support a similar but different data synchronization technique called data PID sequencing. That technique is used instead of data toggle synchronization. Section 5.9.1 defines data PID sequencing.

Microchip Ex. 1018, p. 260
Microchip v. Aptiv
IPR2024-00494

8.6.1 Initialization via SETUP Token

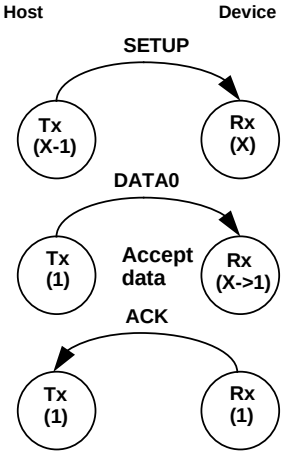


Figure 8-44. SETUP Initialization

Control transfers use the SETUP token for initializing host and function sequence bits. Figure 8-44 shows the host issuing a SETUP packet to a function followed by an OUT transaction. The numbers in the circles represent the transmitter and receiver sequence bits. The function must accept the data and return ACK. When the function accepts the transaction, it must set its sequence bit so that both the host's and function's sequence bits are equal to one at the end of the SETUP transaction.

8.6.2 Successful Data Transactions

Figure 8-45 shows the case where two successful transactions have occurred. For the data transmitter, this means that it toggles its sequence bit upon receipt of ACK. The receiver toggles its sequence bit only if it receives a valid data packet and the packet's data PID matches the current value of its sequence bit. The transmitter only toggles its sequence bit after it receives an ACK to a data packet.

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

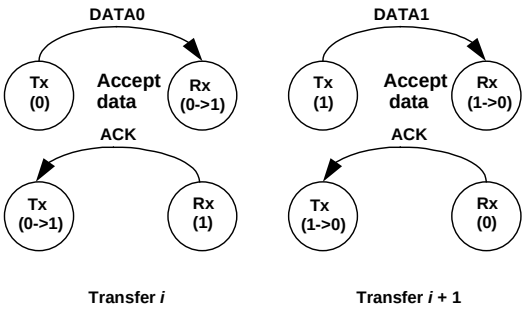


Figure 8-45. Consecutive Transactions

8.6.3 Data Corrupted or Not Accepted

If data cannot be accepted or the received data packet is corrupted, the receiver will issue a NAK or STALL handshake, or timeout, depending on the circumstances, and the receiver will not toggle its sequence bit.

Microchip Ex. 1018, p. 261
Microchip v. Aptiv
IPR2024-00494

Figure 8-46 shows the case where a transaction is NAKed and then retried. Any non-ACK handshake or timeout will generate similar retry behavior. The transmitter, having not received an ACK handshake, will not toggle its sequence bit. As a result, a failed data packet transaction leaves the transmitter's and receiver's sequence bits synchronized and untoggled. The transaction will then be retried and, if successful, will cause both transmitter and receiver sequence bits to toggle.

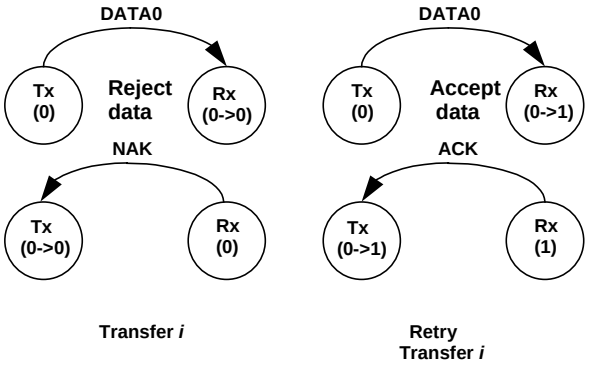


Figure 8-46. NAKed Transaction with Retry

8.6.4 Corrupted ACK Handshake

The transmitter is the last and only agent to know for sure whether a transaction has been successful, due to its receiving an ACK handshake. A lost or corrupted ACK handshake can lead to a temporary loss of synchronization between transmitter and receiver as shown in Figure 8-47. Here the transmitter issues a valid data packet, which is successfully acquired by the receiver; however, the ACK handshake is corrupted.

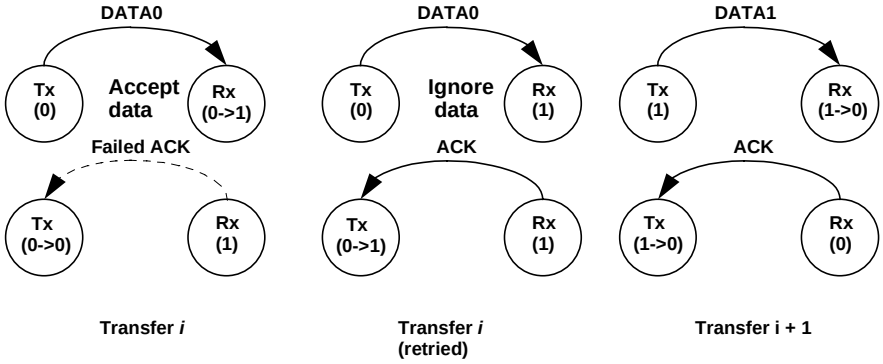


Figure 8-47. Corrupted ACK Handshake with Retry

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

The data transmitter must guarantee that any retried data packet is identical (same length and content) as that sent in the original transaction. If the data transmitter is unable, because of problems such as a buffer underrun condition, to transmit the identical amount of data as was in the original data packet, it must abort

Microchip Ex. 1018, p. 262
Microchip v. Aptiv
IPR2024-00494

the transaction by generating a bit stuffing violation for full-/low-speed. An error for high-speed must be forced by taking the currently calculated CRC and complementing it before transmitting it. This causes a detectable error at the receiver and guarantees that a partial packet will not be interpreted as a good packet. The transmitter should not try to force an error at the receiver by sending a constant known bad CRC. A combination of a bad packet with a "bad" CRC may be interpreted by the receiver as a good packet.

8.6.5 Low-speed Transactions

The USB supports signaling at three speeds: high-speed signaling at 480 Mb/s, full-speed signaling at 12.0 Mb/s, and low-speed signaling at 1.5 Mb/s. Hubs isolate high-speed signaling from full-/low-speed signaling environments.

Within a full-/low-speed signaling environment, hubs disable downstream bus traffic to all ports to which low-speed devices are attached during full-speed downstream signaling. This is required both for EMI reasons and to prevent any possibility that a low-speed device might misinterpret downstream a full-speed packet as being addressed to it.

Figure 8-48 shows an IN low-speed transaction in which the host (or TT) issues a token and handshake and receives a data packet.

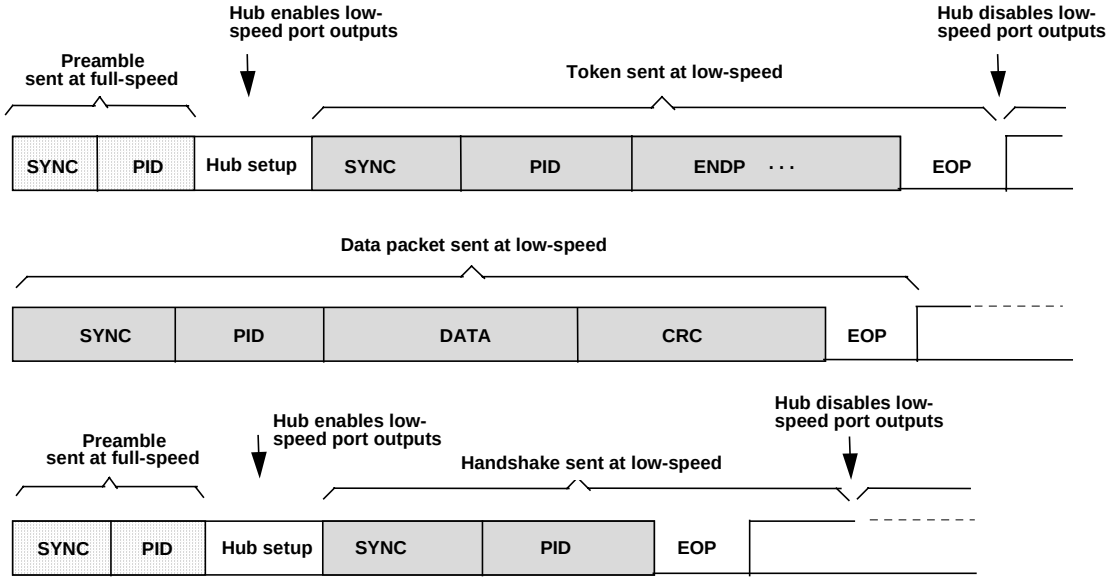


Figure 8-48. Low-speed Transaction

All downstream packets transmitted to low-speed devices within a full-/low-speed signaling environment require a preamble. Preambles are never used in a high-speed signaling environment. The preamble consists of a SYNC followed by a PRE PID, both sent at full-speed. Hubs must comprehend the PRE PID; all other USB devices may ignore it and treat it as undefined. At the end of the preamble PID, the host (or TT) drives the bus to the Idle state for at least one full-speed bit time. This Idle period on the bus is termed the hub setup interval and lasts for at least four full-speed bit times. During this hub setup interval, hubs must drive their full-speed and low-speed ports to their respective Idle states. Hubs must be ready to repeat low-speed signaling on low-speed ports before the end of the hub setup interval. Low-speed connectivity rules are summarized below:

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

Microchip Ex. 1018, p. 263
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

8.7 Error Detection and Recovery

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

8.7.1 Packet Error Categories

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

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

Table 8-8. Packet Error Types

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

Microchip Ex. 1018, p. 264
Microchip v. Aptiv
IPR2024-00494

8.7.2 Bus Turn-around Timing

Neither the device nor the host will send an indication that a received packet had an error. This absence of positive acknowledgement is considered to be the indication that there was an error. As a consequence of this method of error reporting, the host and USB function need to keep track of how much time has elapsed from when the transmitter completes sending a packet until it begins to receive a response packet. This time is referred to as the bus turn-around time. Devices and hosts require turn-around timers to measure this time.

For full-/low-speed transactions, the timer starts counting on the SE0-to-'J' transition of the EOP strobe and stops counting when the Idle-to-'K' SOP transition is detected. For high-speed transactions, the timer starts counting when the data lines return to the squelch level and stops counting when the data lines leave the squelch level.

The device bus turn-around time is defined by the worst case round trip delay plus the maximum device response delay (refer to Sections 7.1.18 and 7.1.19 for specific bus turn-around times). If a response is not received within this worst case timeout, then the transmitter considers that the packet transmission has failed.

Timeout is used and interpreted as a transaction error condition for many transfer types. If the host wishes to indicate an error condition for a transaction via a timeout, it must wait the full bus turn-around time before issuing the next token to ensure that all downstream devices have timed out.

As shown in Figure 8-49, the device uses its bus turn-around timer between token and data or data and handshake phases. The host uses its timer between data and handshake or token and data phases.

If the host receives a corrupted data packet, it may require additional wait time before sending out the next token. This additional wait interval guarantees that the host properly handles false EOPs.

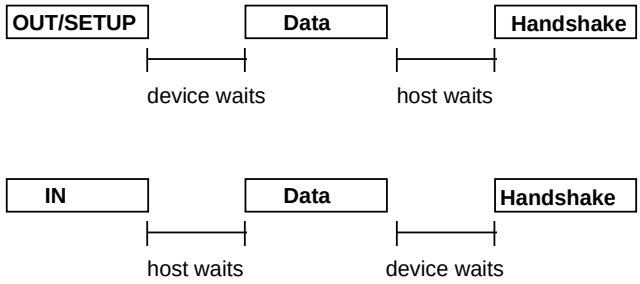


Figure 8-49. Bus Turn-around Timer Usage

8.7.3 False EOPs

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

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

Microchip Ex. 1018, p. 265
Microchip v. Aptiv
IPR2024-00494

packet and not issue the handshake. The device, expecting to see a handshake from the host, will timeout the transaction.

If the host receives a corrupted full-/low-speed data packet, it assumes that a false EOP may have occurred and waits for 16 bit times to see if there is any subsequent upstream traffic. If no bus transitions are detected within the 16 bit interval and the bus remains in the Idle state, the host may issue the next token.

Otherwise, the host waits for the device to finish sending the remainder of its full-/low-speed packet. Waiting 16 bit times guarantees two conditions:

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

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

If the host receives a corrupted high-speed data packet, it ignores any data until the data lines return to the squelch level before issuing the next token. For high-speed transactions, the host does not need to wait additional time (beyond the normal inter-transaction gap time) after the data lines return to the squelch level.

If the host receives a data packet with a valid CRC, it assumes that the packet is complete and requires no additional delay (beyond normal inter-transaction gap time) in issuing the next token.

8.7.4 Babble and Loss of Activity Recovery

The USB must be able to detect and recover from conditions which leave it waiting indefinitely for a full-/low-speed EOP or which leave the bus in something other than the Idle state at the end of a (micro)frame.

- Full-/low-speed loss of activity (LOA) is characterized by an SOP followed by lack of bus activity (bus remains driven to a 'J' or 'K') and no EOP at the end of a frame.
- Full-/low-speed babble is characterized by an SOP followed by the presence of bus activity past the end of a frame.
- High-speed babble/LOA is characterized by the data lines being at an unsquelched level at the end of a microframe.

LOA and babble have the potential to either deadlock the bus or delay the beginning of the next (micro)frame. Neither condition is acceptable, and both must be prevented from occurring. As the USB component responsible for controlling connectivity, hubs are responsible for babble/LOA detection and recovery. All USB devices that fail to complete their transmission at the end of a (micro)frame are prevented from transmitting past a (micro)frame's end by having the nearest hub disable the port to which the offending device is attached. Details of the hub babble/LOA recovery mechanism appear in Section 11.2.5.

Microchip Ex. 1018, p. 266
Microchip v. Aptiv
IPR2024-00494

Chapter 9
USB Device Framework

A USB device may be divided into three layers:

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

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

9.1 USB Device States

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

9.1.1 Visible Device States

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

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

Microchip Ex. 1018, p. 267
Microchip v. Aptiv
IPR2024-00494

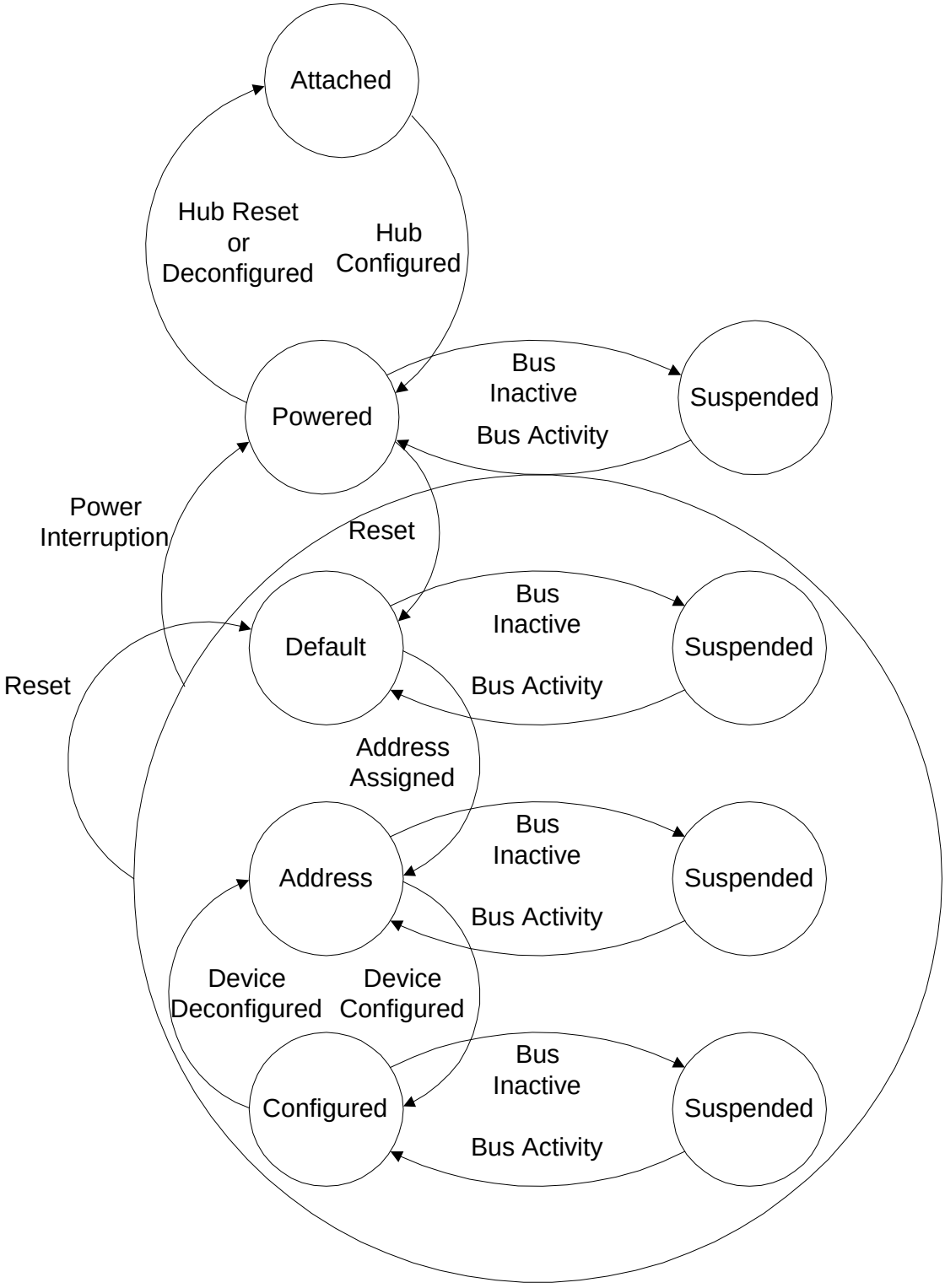


Figure 9-1. Device State Diagram

Microchip Ex. 1018, p. 268
Microchip v. Aptiv
IPR2024-00494

Table 9-1. Visible Device States

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

Microchip Ex. 1018, p. 269
Microchip v. Aptiv
IPR2024-00494

9.1.1.1 Attached

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

9.1.1.2 Powered

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

A device may support both self-powered and bus-powered configurations. Some device configurations support either power source. Other device configurations may be available only if the device is self-powered. Devices report their power source capability through the configuration descriptor. The current power source is reported as part of a device's status. Devices may change their power source at any time, e.g., from self- to bus-powered. If a configuration is capable of supporting both power modes, the power maximum reported for that configuration is the maximum the device will draw from VBUS in either mode. The device must observe this maximum, regardless of its mode. If a configuration supports only one power mode and the power source of the device changes, the device will lose its current configuration and address and return to the Powered state. If a device is self-powered and its current configuration requires more than 100 mA, then if the device switches to being bus-powered, it must return to the Address state. Self-powered hubs that use VBUS to power the Hub Controller are allowed to remain in the Configured state if local power is lost. Refer to Section 11.13 for details.

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

9.1.1.3 Default

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

When the reset process is complete, the USB device is operating at the correct speed (i.e., low-/full-/high-speed). The speed selection for low- and full-speed is determined by the device termination resistors. A device that is capable of high-speed operation determines whether it will operate at high-speed as a part of the reset process (see Chapter 7 for more details).

A device capable of high-speed operation must reset successfully at full-speed when in an electrical environment that is operating at full-speed. After the device is successfully reset, the device must also respond successfully to device and configuration descriptor requests and return appropriate information. The device may or may not be able to support its intended functionality when operating at full-speed.

9.1.1.4 Address

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

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

Microchip Ex. 1018, p. 270
Microchip v. Aptiv
IPR2024-00494

9.1.1.5 Configured

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

9.1.1.6 Suspended

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

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

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

9.1.2 Bus Enumeration

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

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe (refer to Section 11.12.3 for more information). At this point, the USB device is in the Powered state and the port to which it is attached is disabled.
2. The host determines the exact nature of the change by querying the hub.
3. Now that the host knows the port to which the new device has been attached, the host then waits for at least 100 ms to allow completion of an insertion process and for power at the device to become stable. The host then issues a port enable and reset command to that port. Refer to Section 7.1.7.5 for sequence of events and timings of connection through device reset.
4. The hub performs the required reset processing for that port (see Section 11.5.1.5). When the reset signal is released, the port has been enabled. The USB device is now in the Default state and can draw no more than 100 mA from VBUS. All of its registers and state have been reset and it answers to the default address.
5. The host assigns a unique address to the USB device, moving the device to the Address state.
6. Before the USB device receives a unique address, its Default Control Pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
7. The host reads the configuration information from the device by reading each configuration zero to *n*-1, where *n* is the number of configurations. This process may take several milliseconds to complete.

Microchip Ex. 1018, p. 271
Microchip v. Aptiv
IPR2024-00494

8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the Configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of VBUS power described in its descriptor for the selected configuration. From the device's point of view, it is now ready for use.

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

9.2 Generic USB Device Operations

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

9.2.1 Dynamic Attachment and Removal

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

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

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

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

9.2.2 Address Assignment

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

9.2.3 Configuration

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

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

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

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

Within each configuration, each interface descriptor contains fields that identify the interface number and the alternate setting. Interfaces are numbered from zero to one less than the number of concurrent interfaces supported by the configuration. Alternate settings range from zero to one less than the number of alternate

Microchip Ex. 1018, p. 272
Microchip v. Aptiv
IPR2024-00494

settings for a specific interface. The default setting when a device is initially configured is alternate setting zero.

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

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

9.2.4 Data Transfer

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

9.2.5 Power Management

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

9.2.5.1 Power Budgeting

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

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

9.2.5.2 Remote Wakeup

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

Remote wakeup is accomplished using electrical signaling described in Section 7.1.7.7.

9.2.6 Request Processing

With the exception of SetAddress() requests (see Section 9.4.6), a device may begin processing of a request as soon as the device returns the ACK following the Setup. The device is expected to "complete" processing of the request before it allows the Status stage to complete successfully. Some requests initiate operations that take many milliseconds to complete. For requests such as this, the device class is required to define a method other than Status stage completion to indicate that the operation has completed. For example, a reset on a hub port takes at least 10 ms to complete. The SetPortFeature(PORT_RESET) (see Chapter 11) request "completes" when the reset on the port is initiated. Completion of the reset operation is

Microchip Ex. 1018, p. 273
Microchip v. Aptiv
IPR2024-00494

signaled when the port's status change is set to indicate that the port is now enabled. This technique prevents the host from having to constantly poll for a completion when it is known that the request will take a relatively long period of time.

9.2.6.1 Request Processing Timing

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

9.2.6.2 Reset/Resume Recovery Time

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

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

9.2.6.3 Set Address Processing

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

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

9.2.6.4 Standard Device Requests

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

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

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

Microchip Ex. 1018, p. 274
Microchip v. Aptiv
IPR2024-00494

9.2.6.5 Class-specific Requests

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

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

9.2.6.6 Speed Dependent Descriptors

A device capable of operation at high-speed can operate in either full- or high-speed. The device always knows its operational speed due to having to manage its transceivers correctly as part of reset processing (See Chapter 7 for more details on reset). A device also operates at a single speed after completing the reset sequence. In particular, there is no speed switch during normal operation. However, a high-speed capable device may have configurations that are speed dependent. That is, it may have some configurations that are only possible when operating at high-speed or some that are only possible when operating at full-speed. High-speed capable devices must support reporting their speed dependent configurations.

A high-speed capable device responds with descriptor information that is valid for the current operating speed. For example, when a device is asked for configuration descriptors, it only returns those for the current operating speed (e.g., full speed). However, there must be a way to determine the capabilities for both high- and full-speed operation.

Two descriptors allow a high-speed capable device to report configuration information about the other operating speed. The two descriptors are: the (other_speed) device_qualifier descriptor and the other_speed_configuration descriptor. These two descriptors are retrieved by the host by using the GetDescriptor request with the corresponding descriptor type values.

Note: These descriptors are not retrieved unless the host explicitly issues the corresponding GetDescriptor requests. If these two requests are not issued, the device would simply appear to be a single speed device.

Devices that are high-speed capable must set the version number in the *bcdUSB* field of their descriptors to 0200H. This indicates that such devices support the other_speed requests defined by USB 2.0. A device with descriptor version numbers less than 0200H should cause a Request Error response (see next section) if it receives these other_speed requests. A USB 1.x device (i.e., one with a device descriptor version less than 0200H) should not be issued the other_speed requests.

9.2.7 Request Error

When a request is received by a device that is not defined for the device, is inappropriate for the current setting of the device, or has values that are not compatible with the request, then a Request Error exists. The device deals with the Request Error by returning a STALL PID in response to the next Data stage transaction or in the Status stage of the message. It is preferred that the STALL PID be returned at the next Data stage transaction, as this avoids unnecessary bus activity.

Microchip Ex. 1018, p. 275
Microchip v. Aptiv
IPR2024-00494

9.3 USB Device Requests

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

Table 9-2. Format of Setup Data

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

9.3.1 bmRequestType

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

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

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

Microchip Ex. 1018, p. 276
Microchip v. Aptiv
IPR2024-00494

9.3.2 bRequest

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

9.3.3 wValue

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

9.3.4 wIndex

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

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

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

Figure 9-2. wIndex Format when Specifying an Endpoint

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

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

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

Figure 9-3. wIndex Format when Specifying an Interface

9.3.5 wLength

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

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

Microchip Ex. 1018, p. 277
Microchip v. Aptiv
IPR2024-00494

9.4 Standard Device Requests

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

USB devices must respond to standard device requests, even if the device has not yet been assigned an address or has not been configured.

Table 9-3. Standard Device Requests

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

Microchip Ex. 1018, p. 278
Microchip v. Aptiv
IPR2024-00494

Table 9-4. Standard Request Codes

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

Table 9-5. Descriptor Types

Descriptor Types	Value
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5
DEVICE_QUALIFIER	6
OTHER_SPEED_CONFIGURATION	7
INTERFACE_POWER[1]	8

[1] The INTERFACE_POWER descriptor is defined in the current revision of the *USB Interface Power Management Specification*.

Microchip Ex. 1018, p. 279
Microchip v. Aptiv
IPR2024-00494

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

Table 9-6. Standard Feature Selectors

Feature Selector	Recipient	Value
DEVICE_REMOTE_WAKEUP	Device	1
ENDPOINT_HALT	Endpoint	0
TEST_MODE	Device	2

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

9.4.1 Clear Feature

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

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

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

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

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

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

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

Address state: This request is valid when the device is in the Address state; references to interfaces or to endpoints other than endpoint zero shall cause the device to respond with a Request Error.

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

Note: The Test_Mode feature cannot be cleared by the ClearFeature() request.

Microchip Ex. 1018, p. 280
Microchip v. Aptiv
IPR2024-00494

9.4.2 Get Configuration

This request returns the current device configuration value.

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

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

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

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

Address state: The value zero must be returned.

Configured state: The non-zero *bConfigurationValue* of the current configuration must be returned.

9.4.3 Get Descriptor

This request returns the specified descriptor if the descriptor exists.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID (refer to Section 9.6.7)	Descriptor Length	Descriptor

The *wValue* field specifies the descriptor type in the high byte (refer to Table 9-5) and the descriptor index in the low byte. The descriptor index is used to select a specific descriptor (only for configuration and string descriptors) when several descriptors of the same type are implemented in a device. For example, a device can implement several configuration descriptors. For other standard descriptors that can be retrieved via a GetDescriptor() request, a descriptor index of zero must be used. The range of values used for a descriptor index is from 0 to one less than the number of descriptors of that type implemented by the device.

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

The standard request to a device supports three types of descriptors: device (also device_qualifier), configuration (also other_speed_configuration), and string. A high-speed capable device supports the device_qualifier descriptor to return information about the device for the speed at which it is not operating (including *wMaxPacketSize* for the default endpoint and the number of configurations for the other speed). The other_speed_configuration returns information in the same structure as a configuration descriptor, but for a configuration if the device were operating at the other speed. A request for a configuration descriptor returns the configuration descriptor, all interface descriptors, and endpoint descriptors for all of the

Microchip Ex. 1018, p. 281
Microchip v. Aptiv
IPR2024-00494

interfaces in a single request. The first interface descriptor follows the configuration descriptor. The endpoint descriptors for the first interface follow the first interface descriptor. If there are additional interfaces, their interface descriptor and endpoint descriptors follow the first interface's endpoint descriptors. Class-specific and/or vendor-specific descriptors follow the standard descriptors they extend or modify.

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

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

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

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

9.4.4 Get Interface

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

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

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

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

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

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

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

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

9.4.5 Get Status

This request returns status for the specified recipient.

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

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

Microchip Ex. 1018, p. 282
Microchip v. Aptiv
IPR2024-00494

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

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

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

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

Configured state: If an interface or endpoint that does not exist is specified, then the device responds with a Request Error.

A GetStatus() request to a device returns the information shown in Figure 9-4.

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)						Remote Wakeup	Self Powered
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

Figure 9-4. Information Returned by a GetStatus() Request to a Device

The *Self Powered* field indicates whether the device is currently self-powered. If D0 is reset to zero, the device is bus-powered. If D0 is set to one, the device is self-powered. The *Self Powered* field may not be changed by the SetFeature() or ClearFeature() requests.

The *Remote Wakeup* field indicates whether the device is currently enabled to request remote wakeup. The default mode for devices that support remote wakeup is disabled. If D1 is reset to zero, the ability of the device to signal remote wakeup is disabled. If D1 is set to one, the ability of the device to signal remote wakeup is enabled. The *Remote Wakeup* field can be modified by the SetFeature() and ClearFeature() requests using the DEVICE_REMOTE_WAKEUP feature selector. This field is reset to zero when the device is reset.

A GetStatus() request to an interface returns the information shown in Figure 9-5.

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Reserved (Reset to zero)							

Figure 9-5. Information Returned by a GetStatus() Request to an Interface

Microchip Ex. 1018, p. 283
Microchip v. Aptiv
IPR2024-00494

A GetStatus() request to an endpoint returns the information shown in Figure 9-6.

D7	D6	D5	D4	D3	D2	D1	D0
Reserved (Reset to zero)							Halt
D15	D14	D13	D12	D11	D10	D9	D8
Reserved (Reset to zero)							

Figure 9-6. Information Returned by a GetStatus() Request to an Endpoint

The *Halt* feature is required to be implemented for all interrupt and bulk endpoint types. If the endpoint is currently halted, then the *Halt* feature is set to one. Otherwise, the *Halt* feature is reset to zero. The *Halt* feature may optionally be set with the SetFeature(ENDPOINT_HALT) request. When set by the SetFeature() request, the endpoint exhibits the same stall behavior as if the field had been set by a hardware condition. If the condition causing a halt has been removed, clearing the *Halt* feature via a ClearFeature(ENDPOINT_HALT) request results in the endpoint no longer returning a STALL. For endpoints using data toggle, regardless of whether an endpoint has the *Halt* feature set, a ClearFeature(ENDPOINT_HALT) request always results in the data toggle being reinitialized to DATA0. The *Halt* feature is reset to zero after either a SetConfiguration() or SetInterface() request even if the requested configuration or interface is the same as the current configuration or interface.

It is neither required nor recommended that the *Halt* feature be implemented for the Default Control Pipe. However, devices may set the *Halt* feature of the Default Control Pipe in order to reflect a functional error condition. If the feature is set to one, the device will return STALL in the Data and Status stages of each standard request to the pipe except GetStatus(), SetFeature(), and ClearFeature() requests. The device need not return STALL for class-specific and vendor-specific requests.

9.4.6 Set Address

This request sets the device address for all future device accesses.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None

The *wValue* field specifies the device address to use for all subsequent accesses.

As noted elsewhere, requests actually may result in up to three stages. In the first stage, the Setup packet is sent to the device. In the optional second stage, data is transferred between the host and the device. In the final stage, status is transferred between the host and the device. The direction of data and status transfer depends on whether the host is sending data to the device or the device is sending data to the host. The Status stage transfer is always in the opposite direction of the Data stage. If there is no Data stage, the Status stage is from the device to the host.

Stages after the initial Setup packet assume the same device address as the Setup packet. The USB device does not change its device address until after the Status stage of this request is completed successfully. Note that this is a difference between this request and all other requests. For all other requests, the operation indicated must be completed before the Status stage.

If the specified device address is greater than 127, or if *wIndex* or *wLength* are non-zero, then the behavior of the device is not specified.

Microchip Ex. 1018, p. 284
Microchip v. Aptiv
IPR2024-00494

Device response to SetAddress() with a value of 0 is undefined.

Default state: If the address specified is non-zero, then the device shall enter the Address state; otherwise, the device remains in the Default state (this is not an error condition).

Address state: If the address specified is zero, then the device shall enter the Default state; otherwise, the device remains in the Address state but uses the newly-specified address.

Configured state: Device behavior when this request is received while the device is in the Configured state is not specified.

9.4.7 Set Configuration

This request sets the device configuration.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None

The lower byte of the *wValue* field specifies the desired configuration. This configuration value must be zero or match a configuration value from a configuration descriptor. If the configuration value is zero, the device is placed in its Address state. The upper byte of the *wValue* field is reserved.

If *wIndex*, *wLength*, or the upper byte of *wValue* is non-zero, then the behavior of this request is not specified.

Default state: Device behavior when this request is received while the device is in the Default state is not specified.

Address state: If the specified configuration value is zero, then the device remains in the Address state. If the specified configuration value matches the configuration value from a configuration descriptor, then that configuration is selected and the device enters the Configured state. Otherwise, the device responds with a Request Error.

Configured state: If the specified configuration value is zero, then the device enters the Address state. If the specified configuration value matches the configuration value from a configuration descriptor, then that configuration is selected and the device remains in the Configured state. Otherwise, the device responds with a Request Error.

9.4.8 Set Descriptor

This request is optional and may be used to update existing descriptors or new descriptors may be added.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Language ID (refer to Section 9.6.7) or zero	Descriptor Length	Descriptor

Microchip Ex. 1018, p. 285
Microchip v. Aptiv
IPR2024-00494

The *wValue* field specifies the descriptor type in the high byte (refer to Table 9-5) and the descriptor index in the low byte. The descriptor index is used to select a specific descriptor (only for configuration and string descriptors) when several descriptors of the same type are implemented in a device. For example, a device can implement several configuration descriptors. For other standard descriptors that can be set via a SetDescriptor() request, a descriptor index of zero must be used. The range of values used for a descriptor index is from 0 to one less than the number of descriptors of that type implemented by the device.

The *wIndex* field specifies the Language ID for string descriptors or is reset to zero for other descriptors. The *wLength* field specifies the number of bytes to transfer from the host to the device.

The only allowed values for descriptor type are device, configuration, and string descriptor types.

If this request is not supported, the device will respond with a Request Error.

Default state: Device behavior when this request is received while the device is in the Default state is not specified.

Address state: If supported, this is a valid request when the device is in the Address state.

Configured state: If supported, this is a valid request when the device is in the Configured state.

9.4.9 Set Feature

This request is used to set or enable a specific feature.

bmRequestType	bRequest	wValue	wIndex		wLength	Data
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Test Selector	Zero Interface Endpoint	Zero	None

Feature selector values in *wValue* must be appropriate to the recipient. Only device feature selector values may be used when the recipient is a device; only interface feature selector values may be used when the recipient is an interface, and only endpoint feature selector values may be used when the recipient is an endpoint.

Refer to Table 9-6 for a definition of which feature selector values are defined for which recipients.

The TEST_MODE feature is only defined for a device recipient (i.e., bmRequestType = 0) and the lower byte of wIndex must be zero. Setting the TEST_MODE feature puts the device upstream facing port into test mode. The device will respond with a request error if the request contains an invalid test selector. The transition to test mode must be complete no later than 3 ms after the completion of the status stage of the request. The transition to test mode of an upstream facing port must not happen until after the status stage of the request. The power to the device must be cycled to exit test mode of an upstream facing port of a device. See Section 7.1.20 for definitions of each test mode. A device must support the TEST_MODE feature when in the Default, Address or Configured high-speed device states.

A SetFeature() request that references a feature that cannot be set or that does not exist causes a STALL to be returned in the Status stage of the request.

Microchip Ex. 1018, p. 286
Microchip v. Aptiv
IPR2024-00494

Table 9-7. Test Mode Selectors

Value	Description
00H	Reserved
01H	Test_J
02H	Test_K
03H	Test_SE0_NAK
04H	Test_Packet
05H	Test_Force_Enable
06H-3FH	Reserved for standard test selectors
3FH-BFH	Reserved
C0H-FFH	Reserved for vendor-specific test modes.

If the feature selector is *TEST_MODE*, then the most significant byte of *wIndex* is used to specify the specific test mode. The recipient of a SetFeature(TEST_MODE…) must be the device; i.e., the lower byte of *wIndex* must be zero and the *bmRequestType* must be set to zero. The device must have its power cycled to exit test mode. The valid test mode selectors are listed in Table 9-7. See Section 7.1.20 for more information about the specific test modes.

If *wLength* is non-zero, then the behavior of the device is not specified.

If an endpoint or interface is specified that does not exist, then the device responds with a Request Error.

Default state: A device must be able to accept a SetFeature(TEST_MODE, TEST_SELECTOR) request when in the Default State. Device behavior for other SetFeature requests while the device is in the Default state is not specified.

Address state: If an interface or an endpoint other than endpoint zero is specified, then the device responds with a Request Error.

Configured state: This is a valid request when the device is in the Configured state.

9.4.10 Set Interface

This request allows the host to select an alternate setting for the specified interface.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None

Some USB devices have configurations with interfaces that have mutually exclusive settings. This request allows the host to select the desired alternate setting. If a device only supports a default setting for the specified interface, then a STALL may be returned in the Status stage of the request. This request cannot be used to change the set of configured interfaces (the SetConfiguration() request must be used instead).

If the interface or the alternate setting does not exist, then the device responds with a Request Error. If *wLength* is non-zero, then the behavior of the device is not specified.

Microchip Ex. 1018, p. 287
Microchip v. Aptiv
IPR2024-00494

Default state: Device behavior when this request is received while the device is in the Default state is not specified.

Address state: The device must respond with a Request Error.

Configured state: This is a valid request when the device is in the Configured state.

9.4.11 Synch Frame

This request is used to set and then report an endpoint's synchronization frame.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

When an endpoint supports isochronous transfers, the endpoint may also require per-frame transfers to vary in size according to a specific pattern. The host and the endpoint must agree on which frame the repeating pattern begins. The number of the frame in which the pattern began is returned to the host.

If a high-speed device supports the Synch Frame request, it must internally synchronize itself to the zeroth microframe and have a time notion of classic frame. Only the frame number is used to synchronize and reported by the device endpoint (i.e., no microframe number). The endpoint must synchronize to the zeroth microframe.

This value is only used for isochronous data transfers using implicit pattern synchronization. If *wValue* is non-zero or *wLength* is not two, then the behavior of the device is not specified.

If the specified endpoint does not support this request, then the device will respond with a Request Error.

Default state: Device behavior when this request is received while the device is in the Default state is not specified.

Address state: The device shall respond with a Request Error.

Configured state: This is a valid request when the device is in the Configured state.

9.5 Descriptors

USB devices report their attributes using descriptors. A descriptor is a data structure with a defined format. Each descriptor begins with a byte-wide field that contains the total number of bytes in the descriptor followed by a byte-wide field that identifies the descriptor type.

Using descriptors allows concise storage of the attributes of individual configurations because each configuration may reuse descriptors or portions of descriptors from other configurations that have the same characteristics. In this manner, the descriptors resemble individual data records in a relational database.

Where appropriate, descriptors contain references to string descriptors that provide displayable information describing a descriptor in human-readable form. The inclusion of string descriptors is optional. However, the reference fields within descriptors are mandatory. If a device does not support string descriptors, string reference fields must be reset to zero to indicate no string descriptor is available.

If a descriptor returns with a value in its length field that is less than defined by this specification, the descriptor is invalid and should be rejected by the host. If the descriptor returns with a value in its length

Microchip Ex. 1018, p. 288
Microchip v. Aptiv
IPR2024-00494

field that is greater than defined by this specification, the extra bytes are ignored by the host, but the next descriptor is located using the length returned rather than the length expected.

A device may return class- or vendor-specific descriptors in two ways:

1. If the class or vendor specific descriptors use the same format as standard descriptors (e.g., start with a length byte and followed by a type byte), they must be returned interleaved with standard descriptors in the configuration information returned by a GetDescriptor(Configuration) request. In this case, the class or vendor-specific descriptors must follow a related standard descriptor they modify or extend.
2. If the class or vendor specific descriptors are independent of configuration information or use a non-standard format, a GetDescriptor() request specifying the class or vendor specific descriptor type and index may be used to retrieve the descriptor from the device. A class or vendor specification will define the appropriate way to retrieve these descriptors.

9.6 Standard USB Descriptor Definitions

The standard descriptors defined in this specification may only be modified or extended by revision of the Universal Serial Bus Specification.

Note: An extension to the USB 1.0 standard endpoint descriptor has been published in Device Class Specification for Audio Devices Revision 1.0. This is the only extension defined outside USB Specification that is allowed. Future revisions of the USB Specification that extend the standard endpoint descriptor will do so as to not conflict with the extension defined in the Audio Device Class Specification Revision 1.0.

9.6.1 Device

A device descriptor describes general information about a USB device. It includes information that applies globally to the device and all of the device's configurations. A USB device has only one device descriptor.

A high-speed capable device that has different device information for full-speed and high-speed must also have a device_qualifier descriptor (see Section 9.6.2).

The DEVICE descriptor of a high-speed capable device has a version number of 2.0 (0200H). If the device is full-speed only or low-speed only, this version number indicates that it will respond correctly to a request for the device_qualifier desciptor (i.e., it will respond with a request error).

The *bcdUSB* field contains a BCD version number. The value of the *bcdUSB* field is 0xJJMN for version JJ.M.N (JJ – major version number, M – minor version number, N – sub-minor version number), e.g., version 2.1.3 is represented with value 0x0213 and version 2.0 is represented with a value of 0x0200.

The *bNumConfigurations* field indicates the number of configurations at the current operating speed. Configurations for the other operating speed are not included in the count. If there are specific configurations of the device for specific speeds, the *bNumConfigurations* field only reflects the number of configurations for a single speed, not the total number of configurations for both speeds.

If the device is operating at high-speed, the *bMaxPacketSize0* field must be 64 indicating a 64 byte maximum packet. High-speed operation does not allow other maximum packet sizes for the control endpoint (endpoint 0).

All USB devices have a Default Control Pipe. The maximum packet size of a device's Default Control Pipe is described in the device descriptor. Endpoints specific to a configuration and its interface(s) are described in the configuration descriptor. A configuration and its interface(s) do not include an endpoint descriptor for the Default Control Pipe. Other than the maximum packet size, the characteristics of the Default Control Pipe are defined by this specification and are the same for all USB devices.

The *bNumConfigurations* field identifies the number of configurations the device supports. Table 9-8 shows the standard device descriptor.

Microchip Ex. 1018, p. 289
Microchip v. Aptiv
IPR2024-00494

Table 9-8. Standard Device Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	DEVICE Descriptor Type
2	*bcdUSB*	2	BCD	USB Specification Release Number in Binary-Coded Decimal (i.e., 2.10 is 210H). This field identifies the release of the USB Specification with which the device and its descriptors are compliant.
4	*bDeviceClass*	1	Class	Class code (assigned by the USB-IF). If this field is reset to zero, each interface within a configuration specifies its own class information and the various interfaces operate independently. If this field is set to a value between 1 and FEH, the device supports different class specifications on different interfaces and the interfaces may not operate independently. This value identifies the class definition used for the aggregate interfaces. If this field is set to FFH, the device class is vendor-specific.
5	*bDeviceSubClass*	1	SubClass	Subclass code (assigned by the USB-IF). These codes are qualified by the value of the *bDeviceClass* field. If the *bDeviceClass* field is reset to zero, this field must also be reset to zero. If the *bDeviceClass* field is not set to FFH, all values are reserved for assignment by the USB-IF.

Microchip Ex. 1018, p. 290
Microchip v. Aptiv
IPR2024-00494

Table 9-8. Standard Device Descriptor (Continued)

Offset	Field	Size	Value	Description
6	*bDeviceProtocol*	1	Protocol	Protocol code (assigned by the USB-IF). These codes are qualified by the value of the *bDeviceClass* and the *bDeviceSubClass* fields. If a device supports class-specific protocols on a device basis as opposed to an interface basis, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to zero, the device does not use class-specific protocols on a device basis. However, it may use class-specific protocols on an interface basis. If this field is set to FFH, the device uses a vendor-specific protocol on a device basis.
7	*bMaxPacketSize0*	1	Number	Maximum packet size for endpoint zero (only 8, 16, 32, or 64 are valid)
8	*idVendor*	2	ID	Vendor ID (assigned by the USB-IF)
10	*idProduct*	2	ID	Product ID (assigned by the manufacturer)
12	*bcdDevice*	2	BCD	Device release number in binary-coded decimal
14	*iManufacturer*	1	Index	Index of string descriptor describing manufacturer
15	*iProduct*	1	Index	Index of string descriptor describing product
16	*iSerialNumber*	1	Index	Index of string descriptor describing the device's serial number
17	*bNumConfigurations*	1	Number	Number of possible configurations

Microchip Ex. 1018, p. 291
Microchip v. Aptiv
IPR2024-00494

9.6.2 Device_Qualifier

The device_qualifier descriptor describes information about a high-speed capable device that would change if the device were operating at the other speed. For example, if the device is currently operating at full-speed, the device_qualifier returns information about how it would operate at high-speed and vice-versa. Table 9-9 shows the fields of the device_qualifier descriptor.

Table 9-9. Device_Qualifier Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of descriptor
1	*bDescriptorType*	1	Constant	Device Qualifier Type
2	*bcdUSB*	2	BCD	USB specification version number (e.g., 0200H for V2.00)
4	*bDeviceClass*	1	Class	Class Code
5	*bDeviceSubClass*	1	SubClass	SubClass Code
6	*bDeviceProtocol*	1	Protocol	Protocol Code
7	*bMaxPacketSize0*	1	Number	Maximum packet size for other speed
8	*bNumConfigurations*	1	Number	Number of Other-speed Configurations
9	*bReserved*	1	Zero	Reserved for future use, must be zero

The vendor, product, device, manufacturer, product, and serialnumber fields of the standard device descriptor are not included in this descriptor since that information is constant for a device for all supported speeds. The version number for this descriptor must be at least 2.0 (0200H).

The host accesses this descriptor using the GetDescriptor() request. The descriptor type in the GetDescriptor() request is set to device_qualifier (see Table 9-5).

If a full-speed only device (with a device descriptor version number equal to 0200H) receives a GetDescriptor() request for a device_qualifier, it must respond with a request error. The host must not make a request for an other_speed_configuration descriptor unless it first successfully retrieves the device_qualifier descriptor.

9.6.3 Configuration

The configuration descriptor describes information about a specific device configuration. The descriptor contains a *bConfigurationValue* field with a value that, when used as a parameter to the SetConfiguration() request, causes the device to assume the described configuration.

The descriptor describes the number of interfaces provided by the configuration. Each interface may operate independently. For example, an ISDN device might be configured with two interfaces, each providing 64 Kb/s bi-directional channels that have separate data sources or sinks on the host. Another configuration might present the ISDN device as a single interface, bonding the two channels into one 128 Kb/s bi-directional channel.

When the host requests the configuration descriptor, all related interface and endpoint descriptors are returned (refer to Section 9.4.3).

Microchip Ex. 1018, p. 292
Microchip v. Aptiv
IPR2024-00494

A USB device has one or more configuration descriptors. Each configuration has one or more interfaces and each interface has zero or more endpoints. An endpoint is not shared among interfaces within a single configuration unless the endpoint is used by alternate settings of the same interface. Endpoints may be shared among interfaces that are part of different configurations without this restriction.

Once configured, devices may support limited adjustments to the configuration. If a particular interface has alternate settings, an alternate may be selected after configuration. Table 9-10 shows the standard configuration descriptor.

Table 9-10. Standard Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	CONFIGURATION Descriptor Type
2	*wTotalLength*	2	Number	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint, and class- or vendor-specific) returned for this configuration.
4	*bNumInterfaces*	1	Number	Number of interfaces supported by this configuration
5	*bConfigurationValue*	1	Number	Value to use as an argument to the SetConfiguration() request to select this configuration
6	*iConfiguration*	1	Index	Index of string descriptor describing this configuration

Microchip Ex. 1018, p. 293
Microchip v. Aptiv
IPR2024-00494

Table 9-10. Standard Configuration Descriptor (Continued)

Offset	Field	Size	Value	Description
7	*bmAttributes*	1	Bitmap	Configuration characteristics D7: Reserved (set to one) D6: Self-powered D5: Remote Wakeup D4...0: Reserved (reset to zero) D7 is reserved and must be set to one for historical reasons. A device configuration that uses power from the bus and a local source reports a non-zero value in *bMaxPower* to indicate the amount of bus power required and sets D6. The actual power source at runtime may be determined using the GetStatus(DEVICE) request (see Section 9.4.5). If a device configuration supports remote wakeup, D5 is set to one.
8	*bMaxPower*	1	mA	Maximum power consumption of the 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). Note: A device configuration reports whether the configuration is bus-powered or self-powered. Device status reports whether the device is currently self-powered. If a device is disconnected from its external power source, it updates device status to indicate that it is no longer self-powered. A device may not increase its power draw from the bus, when it loses its external power source, beyond the amount reported by its configuration. If a device can continue to operate when disconnected from its external power source, it continues to do so. If the device cannot continue to operate, it fails operations it can no longer support. The USB System Software may determine the cause of the failure by checking the status and noting the loss of the device's power source.

9.6.4 Other_Speed_Configuration

The other_speed_configuration descriptor shown in Table 9-11 describes a configuration of a high-speed capable device if it were operating at its other possible speed. The structure of the other_speed_configuration is identical to a configuration descriptor.

Microchip Ex. 1018, p. 294
Microchip v. Aptiv
IPR2024-00494

Table 9-11. Other_Speed_Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of descriptor
1	*bDescriptorType*	1	Constant	Other_speed_Configuration Type
2	*wTotalLength*	2	Number	Total length of data returned
4	*bNumInterfaces*	1	Number	Number of interfaces supported by this speed configuration
5	*bConfigurationValue*	1	Number	Value to use to select configuration
6	*iConfiguration*	1	Index	Index of string descriptor
7	*bmAttributes*	1	Bitmap	Same as Configuration descriptor
8	*bMaxPower*	1	mA	Same as Configuration descriptor

The host accesses this descriptor using the GetDescriptor() request. The descriptor type in the GetDescriptor() request is set to other_speed_configuration (see Table 9-5).

9.6.5 Interface

The interface descriptor describes a specific interface within a configuration. A configuration provides one or more interfaces, each with zero or more endpoint descriptors describing a unique set of endpoints within the configuration. When a configuration supports more than one interface, the endpoint descriptors for a particular interface follow the interface descriptor in the data returned by the GetConfiguration() request. An interface descriptor is always returned as part of a configuration descriptor. Interface descriptors cannot be directly accessed with a GetDescriptor() or SetDescriptor() request.

An interface may include alternate settings that allow the endpoints and/or their characteristics to be varied after the device has been configured. The default setting for an interface is always alternate setting zero. The SetInterface() request is used to select an alternate setting or to return to the default setting. The GetInterface() request returns the selected alternate setting.

Alternate settings allow a portion of the device configuration to be varied while other interfaces remain in operation. If a configuration has alternate settings for one or more of its interfaces, a separate interface descriptor and its associated endpoints are included for each setting.

If a device configuration supported a single interface with two alternate settings, the configuration descriptor would be followed by an interface descriptor with the *bInterfaceNumber* and *bAlternateSetting* fields set to zero and then the endpoint descriptors for that setting, followed by another interface descriptor and its associated endpoint descriptors. The second interface descriptor's *bInterfaceNumber* field would also be set to zero, but the *bAlternateSetting* field of the second interface descriptor would be set to one.

If an interface uses only endpoint zero, no endpoint descriptors follow the interface descriptor. In this case, the *bNumEndpoints* field must be set to zero.

An interface descriptor never includes endpoint zero in the number of endpoints. Table 9-12 shows the standard interface descriptor.

Microchip Ex. 1018, p. 295
Microchip v. Aptiv
IPR2024-00494

Table 9-12. Standard Interface Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	INTERFACE Descriptor Type
2	*bInterfaceNumber*	1	Number	Number of this interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	*bAlternateSetting*	1	Number	Value used to select this alternate setting for the interface identified in the prior field
4	*bNumEndpoints*	1	Number	Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses the Default Control Pipe.
5	*bInterfaceClass*	1	Class	Class code (assigned by the USB-IF). A value of zero is reserved for future standardization. If this field is set to FFH, the interface class is vendor-specific. All other values are reserved for assignment by the USB-IF.
6	*bInterfaceSubClass*	1	SubClass	Subclass code (assigned by the USB-IF). These codes are qualified by the value of the *bInterfaceClass* field. If the *bInterfaceClass* field is reset to zero, this field must also be reset to zero. If the *bInterfaceClass* field is not set to FFH, all values are reserved for assignment by the USB-IF.

Microchip Ex. 1018, p. 296
Microchip v. Aptiv
IPR2024-00494

Table 9-12. Standard Interface Descriptor (Continued)

Offset	Field	Size	Value	Description
7	*bInterfaceProtocol*	1	Protocol	Protocol code (assigned by the USB). These codes are qualified by the value of the *bInterfaceClass* and the *bInterfaceSubClass* fields. If an interface supports class-specific requests, this code identifies the protocols that the device uses as defined by the specification of the device class. If this field is reset to zero, the device does not use a class-specific protocol on this interface. If this field is set to FFH, the device uses a vendor-specific protocol for this interface.
8	*iInterface*	1	Index	Index of string descriptor describing this interface

9.6.6 Endpoint

Each endpoint used for an interface has its own descriptor. This descriptor contains the information required by the host to determine the bandwidth requirements of each endpoint. An endpoint descriptor is always returned as part of the configuration information returned by a GetDescriptor(Configuration) request. An endpoint descriptor cannot be directly accessed with a GetDescriptor() or SetDescriptor() request. There is never an endpoint descriptor for endpoint zero. Table 9-13 shows the standard endpoint descriptor.

Table 9-13. Standard Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 3...0: The endpoint number Bit 6...4: Reserved, reset to zero Bit 7: Direction, ignored for control endpoints 0 = OUT endpoint 1 = IN endpoint

Microchip Ex. 1018, p. 297
Microchip v. Aptiv
IPR2024-00494

Table 9-13. Standard Endpoint Descriptor (Continued)

Offset	Field	Size	Value	Description
3	*bmAttributes*	1	Bitmap	This field describes the endpoint's attributes when it is configured using the *bConfigurationValue*. Bits 1..0: Transfer Type 00 = Control 01 = Isochronous 10 = Bulk 11 = Interrupt If not an isochronous endpoint, bits 5..2 are reserved and must be set to zero. If isochronous, they are defined as follows: 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 endpoint 11 = Reserved Refer to Chapter 5 for more information. All other bits are reserved and must be reset to zero. Reserved bits must be ignored by the host.

Microchip Ex. 1018, p. 298
Microchip v. Aptiv
IPR2024-00494

Table 9-13. Standard Endpoint Descriptor (Continued)

Offset	Field	Size	Value	Description
4	*wMaxPacketSize*	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. For isochronous endpoints, this value is used to reserve the bus time in the schedule, required for the per-(micro)frame data payloads. The pipe may, on an ongoing basis, actually use less bandwidth than that reserved. The device reports, if necessary, the actual bandwidth used via its normal, non-USB defined mechanisms. For all endpoints, bits 10..0 specify the maximum packet size (in bytes). For high-speed isochronous and interrupt endpoints: Bits 12..11 specify the number of additional transaction opportunities per microframe: 00 = None (1 transaction per microframe) 01 = 1 additional (2 per microframe) 10 = 2 additional (3 per microframe) 11 = Reserved Bits 15..13 are reserved and must be set to zero. Refer to Chapter 5 for more information.
6	*bInterval*	1	Number	Interval for polling endpoint for data transfers. Expressed in frames or microframes depending on the device operating speed (i.e., either 1 millisecond or 125 µs units). For full-/high-speed isochronous endpoints, this value must be in the range from 1 to 16. The *bInterval* value is used as the exponent for a $2^{\text{bInterval-1}}$ value; e.g., a *bInterval* of 4 means a period of 8 (2^{4-1}). For full-/low-speed interrupt endpoints, the value of this field may be from 1 to 255. For high-speed interrupt endpoints, the *bInterval* value is used as the exponent for a $2^{\text{bInterval-1}}$ value; e.g., a *bInterval* of 4 means a period of 8 (2^{4-1}). This value must be from 1 to 16. For high-speed bulk/control OUT endpoints, the *bInterval* must specify the maximum NAK rate of the endpoint. A value of 0 indicates the endpoint never NAKs. Other values indicate at most 1 NAK each *bInterval* number of microframes. This value must be in the range from 0 to 255. See Chapter 5 description of periods for more detail.

The *bmAttributes* field provides information about the endpoint's Transfer Type (bits 1..0) and Synchronization Type (bits 3..2). In addition, the Usage Type bit (bits 5..4) indicate whether this is an endpoint used for normal data transfers (bits 5..4=00B), whether it is used to convey explicit feedback information for one or more data endpoints (bits 5..4=01B) or whether it is a data endpoint that also serves

Microchip Ex. 1018, p. 299
Microchip v. Aptiv
IPR2024-00494

as an implicit feedback endpoint for one or more data endpoints (bits 5..4=10B). Bits 5..2 are only meaningful for isochronous endpoints and must be reset to zero for all other transfer types.

If the endpoint is used as an explicit feedback endpoint (bits 5..4=01B), then the Transfer Type must be set to isochronous (bits1..0 = 01B) and the Synchronization Type must be set to No Synchronization (bits 3..2=00B).

A feedback endpoint (explicit or implicit) needs to be associated with one (or more) isochronous data endpoints to which it provides feedback service. The association is based on endpoint number matching. A feedback endpoint always has the opposite direction from the data endpoint(s) it services. If multiple data endpoints are to be serviced by the same feedback endpoint, the data endpoints must have ascending ordered–but not necessarily consecutive–endpoint numbers. The first data endpoint and the feedback endpoint must have the same endpoint number (and opposite direction). This ensures that a data endpoint can uniquely identify its feedback endpoint by searching for the first feedback endpoint that has an endpoint number equal or less than its own endpoint number.

Example: Consider the extreme case where there is a need for five groups of OUT asynchronous isochronous endpoints and at the same time four groups of IN adaptive isochronous endpoints. Each group needs a separate feedback endpoint and the groups are composed as shown in Figure 9-7.

OUT Group	Nr of OUT Endpoints	IN Group	Nr of IN Endpoints
1	1	6	1
2	2	7	2
3	2	8	3
4	3	9	4
5	3		

Figure 9-7. Example of Feedback Endpoint Numbers

The endpoint numbers can be intertwined as illustrated in Figure 9-8.

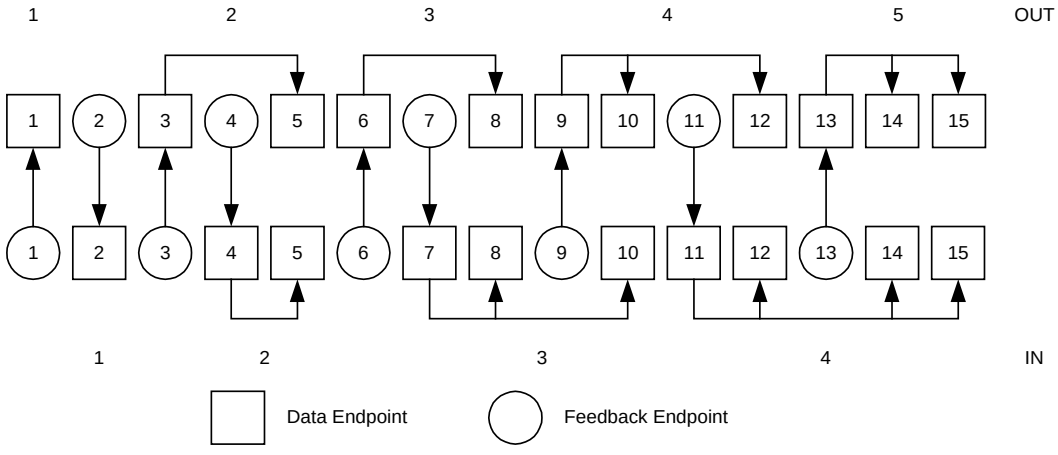


Figure 9-8. Example of Feedback Endpoint Relationships

Microchip Ex. 1018, p. 300
Microchip v. Aptiv
IPR2024-00494

High-speed isochronous and interrupt endpoints use bits 12..11 of *wMaxPacketSize* to specify multiple transactions for each microframe specified by *bInterval*. If bits 12..11 of *wMaxPacketSize* are zero, the maximum packet size for the endpoint can be any allowed value (as defined in Chapter 5). If bits 12..11 of *wMaxPacketSize* are not zero (0), the allowed values for *wMaxPacketSize* bits 10..0 are limited as shown in Table 9-14.

Table 9-14. Allowed wMaxPacketSize Values for Different Numbers of Transactions per Microframe

***wMaxPacketSize* bits 12..11**	***wMaxPacketSize* bits 10..0 Values Allowed**
00	1 – 1024
01	513 – 1024
10	683 – 1024
11	N/A; reserved

For high-speed bulk and control OUT endpoints, the *bInterval* field is only used for compliance purposes; the host controller is not required to change its behavior based on the value in this field.

9.6.7 String

String descriptors are optional. As noted previously, if a device does not support string descriptors, all references to string descriptors within device, configuration, and interface descriptors must be reset to zero.

String descriptors use UNICODE encodings as defined by *The Unicode Standard, Worldwide Character Encoding, Version 3.0*, The Unicode Consortium, Addison-Wesley Publishing Company, Reading, Massachusetts (URL: http://www.unicode.com). The strings in a USB device may support multiple languages. When requesting a string descriptor, the requester specifies the desired language using a sixteen-bit language ID (LANGID) defined by the USB-IF. The list of currently defined USB LANGIDs can be found at http://www.usb.org/developers/docs.html. String index zero for all languages returns a string descriptor that contains an array of two-byte LANGID codes supported by the device. Table 9-15 shows the LANGID code array. A USB device may omit all string descriptors. USB devices that omit all string descriptors must not return an array of LANGID codes.

The array of LANGID codes is not NULL-terminated. The size of the array (in bytes) is computed by subtracting two from the value of the first byte of the descriptor.

Table 9-15. String Descriptor Zero, Specifying Languages Supported by the Device

Offset	Field	Size	Value	Description
0	*bLength*	1	N+2	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	STRING Descriptor Type
2	*wLANGID[0]*	2	Number	LANGID code zero
...	...	...	...	...
N	*wLANGID[x]*	2	Number	LANGID code *x*

Microchip Ex. 1018, p. 301
Microchip v. Aptiv
IPR2024-00494

The UNICODE string descriptor (shown in Table 9-16) is not NULL-terminated. The string length is computed by subtracting two from the value of the first byte of the descriptor.

Table 9-16. UNICODE String Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	STRING Descriptor Type
2	*bString*	N	Number	UNICODE encoded string

9.7 Device Class Definitions

All devices must support the requests and descriptor definitions described in this chapter. Most devices provide additional requests and, possibly, descriptors for device-specific extensions. In addition, devices may provide extended services that are common to a group of devices. In order to define a class of devices, the following information must be provided to completely define the appearance and behavior of the device class.

9.7.1 Descriptors

If the class requires any specific definition of the standard descriptors, the class definition must include those requirements as part of the class definition. In addition, if the class defines a standard extended set of descriptors, they must also be fully defined in the class definition. Any extended descriptor definitions must follow the approach used for standard descriptors; for example, all descriptors must begin with a length field.

9.7.2 Interface(s) and Endpoint Usage

When a class of devices is standardized, the interfaces used by the devices, including how endpoints are used, must be included in the device class definition. Devices may further extend a class definition with proprietary features as long as they meet the base definition of the class.

9.7.3 Requests

All of the requests specific to the class must be defined.

Microchip Ex. 1018, p. 302
Microchip v. Aptiv
IPR2024-00494

Chapter 10
USB Host: Hardware and Software

The USB interconnect supports data traffic between a host and a USB device. This chapter describes the host interfaces necessary to facilitate USB communication between a software client, resident on the host, and a function implemented on a device. The implementation described in this chapter is not required. This implementation is provided as an example to illustrate the host system behavior expected by a USB device. A host system may provide a different host software implementation as long as a USB device experiences the same host behavior.

10.1 Overview of the USB Host

10.1.1 Overview

The basic flow and interrelationships of the USB communications model are shown in Figure 10-1.

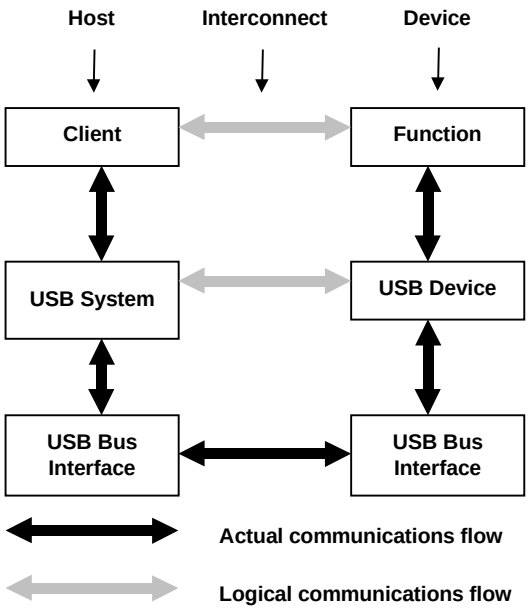


Figure 10-1. Interlayer Communications Model

The host and the device are divided into the distinct layers depicted in Figure 10-1. Vertical arrows indicate the actual communication on the host. The corresponding interfaces on the device are implementation-specific. All communications between the host and device ultimately occur on the physical USB wire. However, there are logical host-device interfaces between each horizontal layer. These communications, between client software resident on the host and the function provided by the device, are typified by a contract based on the needs of the application currently using the device and the capabilities provided by the device.

This client-function interaction creates the requirements for all of the underlying layers and their interfaces.

Microchip Ex. 1018, p. 303
Microchip v. Aptiv
IPR2024-00494

This chapter describes this model from the point of view of the host and its layers. Figure 10-2 illustrates, based on the overall view introduced in Chapter 5, the host's view of its communication with the device.

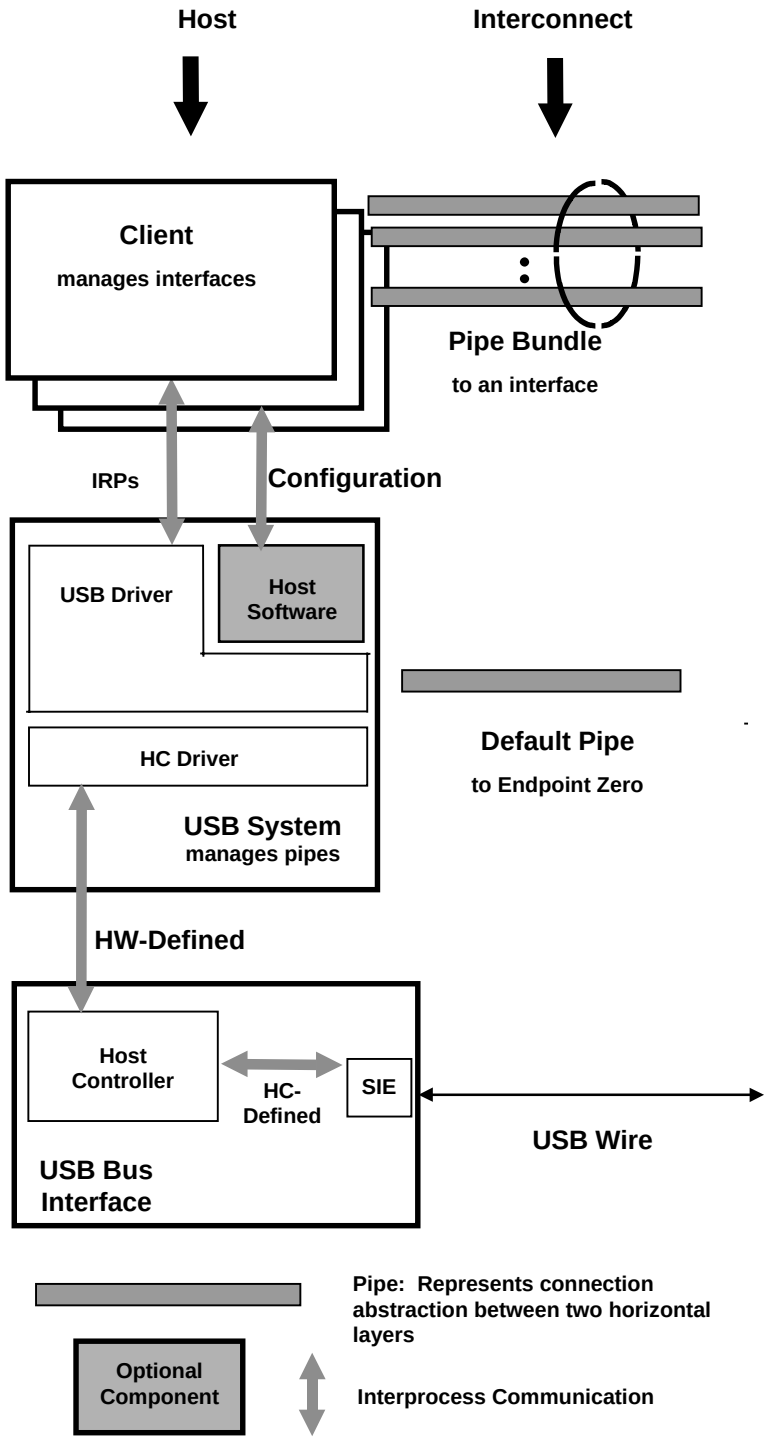


Figure 10-2. Host Communications

Microchip Ex. 1018, p. 304
Microchip v. Aptiv
IPR2024-00494

There is only one host for each USB. The major layers of a host consist of the following:

- USB bus interface
- USB System
- Client

The USB bus interface handles interactions for the electrical and protocol layers (refer to Chapter 7 and Chapter 8). From the interconnect point of view, a similar USB bus interface is provided by both the USB device and the host, as exemplified by the Serial Interface Engine (SIE). On the host, however, the USB bus interface has additional responsibilities due to the unique role of the host on the USB and is implemented as the Host Controller. The Host Controller has an integrated root hub providing attachment points to the USB wire.

The USB System uses the Host Controller to manage data transfers between the host and USB devices. The interface between the USB System and the Host Controller is dependent on the hardware definition of the Host Controller. The USB System, in concert with the Host Controller, performs the translation between the client's view of data transfers and the USB transactions appearing on the interconnect. This includes the addition of any USB feature support such as protocol wrappers. The USB System is also responsible for managing USB resources, such as bandwidth and bus power, so that client access to the USB is possible.

The USB System has three basic components:

- Host Controller Driver
- USB Driver
- Host Software

The Host Controller Driver (HCD) exists to more easily map the various Host Controller implementations into the USB System, such that a client can interact with its device without knowing to which Host Controller the device is connected. The USB Driver (USBD) provides the basic host interface (USBDI) for clients to USB devices. The interface between the HCD and the USBD is known as the Host Controller Driver Interface (HCDI). This interface is never available directly to clients and thus is not defined by the USB Specification. A particular HCDI is, however, defined by each operating system that supports various Host Controller implementations.

The USBD provides data transfer mechanisms in the form of I/O Request Packets (IRPs), which consist of a request to transport data across a specific pipe. In addition to providing data transfer mechanisms, the USBD is responsible for presenting to its clients an abstraction of a USB device that can be manipulated for configuration and state management. As part of this abstraction, the USBD owns the default pipe (see Chapter 5 and Chapter 9) through which all USB devices are accessed for the purposes of standard USB control. This default pipe represents a logical communication between the USBD and the abstraction of a USB device as shown in Figure 10-2.

In some operating systems, additional non-USB System Software is available that provides configuration and loading mechanisms to device drivers. In such operating systems, the device driver shall use the provided interfaces instead of directly accessing the USBDI mechanisms.

The client layer describes all the software entities that are responsible for directly interacting with USB devices. When each device is attached to the system, these clients might interact directly with the peripheral hardware. The shared characteristics of the USB place USB System Software between the client and its device; that is, a client cannot directly access the device's hardware.

Microchip Ex. 1018, p. 305
Microchip v. Aptiv
IPR2024-00494

Overall, the host layers provide the following capabilities:

- Detecting the attachment and removal of USB devices
- Managing USB standard control flow between the host and USB devices
- Managing data flow between the host and USB devices
- Collecting status and activity statistics
- Controlling the electrical interface between the Host Controller and USB devices, including the provision of a limited amount of power

The following sections describe these responsibilities and the requirements placed on the USBDI in greater detail. The actual interfaces used for a specific combination of host platform and operating system are described in the appropriate operating system environment guide.

All hubs (see Chapter 11) report internal status changes and their port change status via the status change pipe. This includes a notification of when a USB device is attached to or removed from one of their ports. A USBD client generically known as the hub driver receives these notifications as owner of the hub's Status Change pipe. For device attachments, the hub driver then initiates the device configuration process. In some systems, this hub driver is a part of the host software provided by the operating system for managing devices.

10.1.2 Control Mechanisms

Control information may be passed between the host and a USB device using in-band or out-of-band signaling. In-band signaling mixes control information with data in a pipe outside the awareness of the host. Out-of-band signaling places control information in a separate pipe.

There is a message pipe called the default pipe for each attached USB device. This logical association between a host and a USB device is used for USB standard control flow such as device enumeration and configuration. The default pipe provides a standard interface to all USB devices. The default pipe may also be used for device-specific communications, as mediated by the USBD, which owns the default pipes of all of the USB devices.

A particular USB device may allow the use of additional message pipes to transfer device-specific control information. These pipes use the same communications protocol as the default pipe, but the information transferred is specific to the USB device and is not standardized by the USB Specification.

The USBD supports the sharing of the default pipe, which it owns and uses, with its clients. It also provides access to any other control pipes associated with the device.

10.1.3 Data Flow

The Host Controller is responsible for transferring streams of data between the host and USB devices. These data transfers are treated as a continuous stream of bytes. The USB supports four basic types of data transfers:

- Control transfers
- Isochronous transfers
- Interrupt transfers
- Bulk transfers

For additional information on transfer types, refer to Chapter 5.

Each device presents one or more interfaces that a client may use to communicate with the device. Each interface is composed of zero or more pipes that individually transfer data between the client and a particular endpoint on the device. The USBD establishes interfaces and pipes at the explicit request of the Host Software. The Host Controller provides service based on parameters provided by the Host Software when the configuration request is made.

Microchip Ex. 1018, p. 306
Microchip v. Aptiv
IPR2024-00494

A pipe has several characteristics based on the delivery requirements of the data to be transferred. Examples of these characteristics include the following:

- The rate at which data needs to be transferred
- Whether data is provided at a steady rate or sporadically
- How long data may be delayed before delivery
- Whether the loss of data being transferred is catastrophic

A USB device endpoint describes the characteristics required for a specific pipe. Endpoints are described as part of a USB device's characterization information. For additional details, refer to Chapter 9.

10.1.4 Collecting Status and Activity Statistics

As a common communicant for all control and data transfers between the host and USB devices, the USB System and the Host Controller are well-positioned to track status and activity information. Such information is provided upon request to the Host Software, allowing that software to manage status and activity information. This specification does not identify any specific information that should be tracked or require any particular format for reporting activity and status information.

10.1.5 Electrical Interface Considerations

The host provides power to USB devices attached to the root hub. The amount of power provided by a port is specified in Chapter 7.

10.2 Host Controller Requirements

In all implementations, Host Controllers perform the same basic duties with regard to the USB and its attached devices. These basic duties are described below.

The Host Controller has requirements from both the host and the USB. The following is a brief overview of the functionality provided. Each capability is discussed in detail in subsequent sections.

State Handling	As a component of the host, the Host Controller reports and manages its states.
Serializer/Deserializer	For data transmitted from the host, the Host Controller converts protocol and data information from its native format to a bit stream transmitted on the USB. For data being received into the host, the reverse operation is performed.
(micro)frame Generation	The Host Controller produces SOF tokens at a period of 1 ms when operating with full-speed devices, and at a period of 125 µs when operating with high-speed devices.
Data Processing	The Host Controller processes requests for data transmission to and from the host.
Protocol Engine	The Host Controller supports the protocol specified by the USB.
Transmission Error Handling	All Host Controllers exhibit the same behavior when detecting and reacting to the defined error categories.
Remote Wakeup	All Host Controllers must have the ability to place the bus into the Suspended state and to respond to bus wakeup events.

Microchip Ex. 1018, p. 307
Microchip v. Aptiv
IPR2024-00494

Root Hub	The root hub provides standard hub function to link the Host Controller to one or more USB ports.
Host System Interface	Provides a high-speed data path between the Host Controller and host system.

The following sections present a more detailed discussion of the required capabilities of the Host Controller.

10.2.1 State Handling

The Host Controller has a series of states that the USB System manages. Additionally, the Host Controller provides the interface to the following two areas of USB-relevant state:

- State change propagation
- Root hub

The root hub presents to the hub driver the same standard states as other USB devices. The Host Controller supports these states and their transitions for the hub. For detailed discussions of USB states, including their interrelations and transitions, refer to Chapter 9.

The overall state of the Host Controller is inextricably linked with that of the root hub and of the overall USB. Any Host Controller state changes that are visible to attached devices must be reflected in the corresponding device state change information such that the resulting Host Controller and device states are consistent.

USB devices request a wakeup through the use of resume signaling (refer to Chapter 7). The Host Controller must notify the rest of the host of a resume event through a mechanism or mechanisms specific to that system's implementation. The Host Controller itself may cause a resume event through the same signaling method.

10.2.2 Serializer/Deserializer

The actual transmission of data across the physical USB takes places as a serial bit stream. A Serial Interface Engine (SIE), whether implemented as part of the host or a USB device, handles the serialization and deserialization of USB transmissions. On the host, this SIE is part of the Host Controller.

10.2.3 Frame and Microframe Generation

It is the Host Controller's responsibility to partition USB time into quantities called "frames" when operating with full-speed devices, and "microframes" when operating with high-speed devices. Frames and microframes are created by the Host Controller through issuing Start-of-Frame (SOF) tokens as shown in Figure 10-3. The SOF token is the first transmission in the (micro)frame period. Host controllers operating with high-speed devices generate SOF tokens at 125 µs intervals. Host controllers operating with full-speed devices generate SOF tokens at 1.00 ms intervals. After issuing an SOF token, the Host Controller is free to transmit other transactions for the remainder of the (micro)frame period. When the Host Controller is in its normal operating state, SOF tokens must be continuously generated at appropriate periodic rate, regardless of other bus activity or lack thereof. If the Host Controller enters a state where it is not providing power on the bus, it must not generate SOFs. When the Host Controller is not generating SOFs, it may enter a power-reduced state.

Microchip Ex. 1018, p. 308
Microchip v. Aptiv
IPR2024-00494

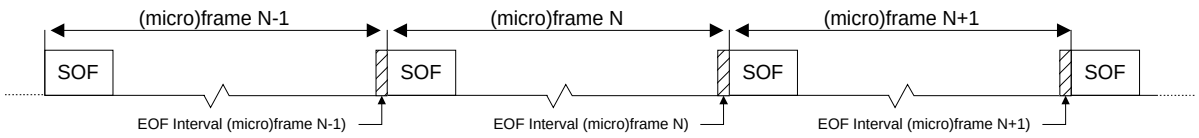


Figure 10-3. Frame and Microframe Creation

The SOF token holds the highest priority access to the bus. Babble circuitry in hubs electrically isolates any active transmitters during the End-of-microframe or End-of-Frame (EOF) interval, providing an idle bus for the SOF transmission.

The Host Controller maintains the current (micro)frame number that may be read by the USB System.

The following apply to the current (micro)frame number maintained by the host:

- Used to uniquely identify one (micro)frame from another
- Incremented at the end of every (micro)frame period
- Valid through the subsequent (micro)frame

Host controllers operating with full-speed devices maintain a current frame number (at least 11 bits) that increments at a 1 ms period. The host transmits the lower 11 bits of the current frame number in each SOF token transmission.

Host controllers operating with high-speed devices maintain a current microframe number (at least 14 bits) that increments at a 125 µs period. The host transmits bits 3 through 13 of the current microframe number in each SOF token transmission. This results in the same SOF packet value being transmitted for eight consecutive microframes before the SOF packet value increments.

When requested from the Host Controller, the current (micro)frame number is the (micro)frame number in existence at the time the request was fulfilled. The current (micro)frame number as returned by the host (Host Controller or HCD) is at least 32 bits, although the Host Controller itself is not required to maintain more than 11 bits when operating with full-speed devices or 14 bits when operating with high-speed devices.

The Host Controller shall cease transmission during the EOF interval. When the EOF interval begins, any transactions scheduled specifically for the (micro)frame that has just passed are retired. If the Host Controller is executing a transaction at the time the EOF interval is encountered, the Host Controller terminates the transaction.

10.2.4 Data Processing

The Host Controller is responsible for receiving data from the USB System and sending it to the USB and for receiving data from the USB and sending it to the USB System. The particular format used for the data communications between the USB System and the Host Controller is implementation specific, within the rules for transfer behavior described in Chapter 5.

10.2.5 Protocol Engine

The Host Controller manages the USB protocol level interface. It inserts the appropriate protocol information for outgoing transmissions. It also strips and interprets, as appropriate, the incoming protocol information.

Microchip Ex. 1018, p. 309
Microchip v. Aptiv
IPR2024-00494

10.2.6 Transmission Error Handling

The Host Controller must be capable of detecting the following transmission error conditions, which are defined from the host's point of view:

- Timeout conditions after a host-transmitted token or packet. These errors occur when the addressed endpoint is unresponsive or when the structure of the transmission is so badly damaged that the targeted endpoint does not recognize it.
- Data errors resulting in missing or invalid transmissions:
  - The Host Controller is unable to completely send or receive a packet for host specific reasons, for example, a transmission extending beyond EOF or a lack of resources available to the Host Controller.
  - An invalid CRC field on a received data packet.
- Protocol errors:
  - An invalid handshake PID, such as a malformed or inappropriate handshake
  - A false EOP
  - A bit stuffing error

For each bulk, control, and interrupt transaction, the host must maintain an error count tally. Errors result from the conditions described above, not as a result of an endpoint NAKing a request. This value reflects the number of times the transaction has encountered a transmission error. It is recommended that the error count not be incremented when there was an error due to host specific reasons (buffer underrun or overrun), and that whenever a transaction does not encounter a transmission error, the error count is reset to zero.

If the error count for a given transaction reaches three, the host retires the transfer. When a transfer is retired due to excessive errors, the last error type must be indicated. Isochronous transactions are attempted only once, regardless of outcome, and, therefore, no error count is maintained for this type.

10.2.7 Remote Wakeup

If USB System wishes to place the bus in the Suspended state, it commands the Host Controller to stop all bus traffic, including SOFs. This causes all USB devices to enter the Suspended state. In this state, the USB System may enable the Host Controller to respond to bus wakeup events. This allows the Host Controller to respond to bus wakeup signaling to restart the host system.

10.2.8 Root Hub

The root hub provides the connection between the Host Controller and one or more USB ports. The root hub provides the same functionality for dealing with USB topology as other hubs (see Chapter 11), except that the hardware and software interface between the root hub and the Host Controller is defined by the specific hardware implementation.

10.2.8.1 Port Resets

Section 7.1.7.5 describes the requirements of a hub to ensure all upstream resume attempts are overpowered with a long reset downstream. Root hubs must provide an aggregate reset period of at least 50 ms. If the reset duration is controlled in hardware and the hardware timer is <50 ms, the USB System can issue several consecutive resets to accumulate the specified reset duration as described in Section 7.1.7.5.

Microchip Ex. 1018, p. 310
Microchip v. Aptiv
IPR2024-00494

10.2.9 Host System Interface

The Host Controller provides a high-speed bus-mastering interface to and from main system memory. The physical transfer between memory and the USB wire is performed automatically by the Host Controller. When data buffers need to be filled or emptied, the Host Controller informs the USB System.

10.3 Overview of Software Mechanisms

The HCD and the USBD present software interfaces based on different levels of abstraction. They are, however, expected to operate together in a specified manner to satisfy the overall requirements of the USB System (see Figure 10-2). The requirements for the USB System are expressed primarily as requirements for the USBDI. The division of duties between the USBD and the HCD is not defined. However, the one requirement of the HCDI that must be met is that it supports, in the specified operating system context, multiple Host Controller implementations.

The HCD provides an abstraction of the Host Controller and an abstraction of the Host Controller's view of data transfer across the USB. The USBD provides an abstraction of the USB device and of the data transfers between the client of the USBD and the function on the USB device. Overall, the USB System acts as a facilitator for transmitting data between the client and the function and as a control point for the USB-specific interfaces of the USB device. As part of facilitating data transfer, the USB System provides buffer management capabilities and allows the synchronization of the data transmittal to the needs of the client and the function.

The specific requirements for the USBDI are described later in this chapter. The exact functions that fulfill these requirements are described in the relevant operating system environment guide for the HCDI and the USBDI. The procedures involved in accomplishing data transfers via the USBDI are described in the following sections.

10.3.1 Device Configuration

Different operating system environments perform device configuration using different software components and different sequences of events. The USB System does not assume a specific operating system method. However, there are some basic requirements that must be fulfilled by any USB System implementation. In some operating systems, existing host software provides these requirements. In others, the USB System provides the capabilities.

The USB System assumes a specialized client of the USBD, called a hub driver, that acts as a clearinghouse for the addition and removal of devices from a particular hub. Once the hub driver receives such notifications, it will employ additional host software and other USBD clients, in an operating system specific manner, to recognize and configure the device. This model, shown in Figure 10-4, is the basis of the following discussion.

Microchip Ex. 1018, p. 311
Microchip v. Aptiv
IPR2024-00494

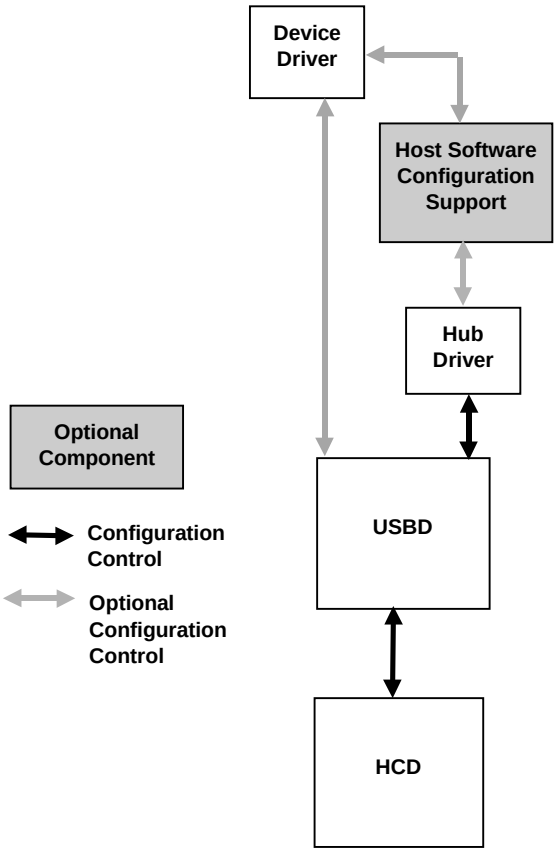


Figure 10-4. Configuration Interactions

When a device is attached, the hub driver receives a notification from the hub detecting the change. The hub driver, using the information provided by the hub, requests a device identifier from the USBD. The USBD in turn sets up the default pipe for that device and returns a device identifier to the hub driver.

The device is now ready to be configured for use. For each device, there are three configurations that must be complete before that device is ready for use:

1. Device Configuration: This includes setting up all of the device's USB parameters and allocating all USB host resources that are visible to the device. This is accomplished by setting the configuration value on the device. A limited set of configuration changes, such as alternate settings, is allowed without totally reconfiguring the device. Once the device is configured, it is, from its point of view, ready for use.
2. USB Configuration: In order to actually create a USBD pipe ready for use by a client, additional USB information, not visible to the device, must be specified by the client. This information, known as the Policy for the pipe, describes how the client will use the pipe. This includes such items as the maximum amount of data the client will transfer with one IRP, the maximum service interval the client will use, the client's notification identification, and so on.
3. Function Configuration: Once configuration types 1 and 2 have been accomplished, the pipe is completely ready for use from the USB's point of view. However, additional vendor- or class-specific setup may be required before the client can actually use the pipe. This configuration is a private matter between the device and the client and is not standardized by the USBD.

Microchip Ex. 1018, p. 312
Microchip v. Aptiv
IPR2024-00494

The following paragraphs describe the device and USB configuration requirements.

The responsible configuring software performs the actual device configuration. Depending on the particular operating system implementation, the software responsible for configuration can include the following:

- The hub driver
- Other host software
- A device driver

The configuring software first reads the device descriptor, then requests the description for each possible configuration. It may use the information provided to load a particular client, such as a device driver, which initially interacts with the device. The configuring software, perhaps with input from that device driver, chooses a configuration for the device. Setting the device configuration sets up all of the endpoints on the device and returns a collection of interfaces to be used for data transfer by USBD clients. Each interface is a collection of pipes owned by a single client.

This initial configuration uses the default settings for interfaces and the default bandwidth for each endpoint. A USBD implementation may additionally allow the client to specify alternate interfaces when selecting the initial configuration. The USB System will verify that the resources required for the support of the endpoint are available and, if so, will allocate the bandwidth required. Refer to Section 10.3.2 for a discussion of resource management.

The device is now configured, but the created pipes are not yet ready for use. The USB configuration is accomplished when the client initializes each pipe by setting a Policy to specify how it will interact with the pipe. Among the information specified is the client's maximum service interval and notification information. Among the actions taken by the USB System, as a result of setting the Policy, is determining the amount of buffer working space required beyond the data buffer space provided by the client. The size of the buffers required is based upon the usage chosen by the client and upon the per-transfer needs of the USB System.

The client receives notifications when IRPs complete, successfully or due to errors. The client may also wake up independently of USB notification to check the status of pending IRPs.

The client may also choose to make configuration modifications, such as enabling an alternate setting for an interface or changing the bandwidth allocated to a particular pipe. In order to perform these changes, the interface or pipe, respectively, must be idle.

10.3.2 Resource Management

Whenever a pipe is setup by the USBD for a given endpoint, the USB System must determine if it can support the pipe. The USB System makes this determination based on the requirements stated in the endpoint descriptor. One of the endpoint requirements, which must be supported in order to create a pipe for an endpoint, is the bandwidth necessary for that endpoint's transfers. There are two stages to check for available bandwidth. First the maximum execution time for a transaction is calculated. Then the (micro)frame schedule is consulted to determine if the indicated transaction will fit.

The allocation of the guaranteed bandwidth for isochronous and interrupt pipes, and the determination of whether a particular control or bulk transaction will fit into a given (micro)frame, can be determined by a software heuristic in the USB System. If the actual transaction execution time in the Host Controller exceeds the heuristically determined value, the Host Controller is responsible for ensuring that (micro)frame integrity is maintained (refer to Section 10.2.3). The following discussion describes the requirements for the USB System heuristic.

Microchip Ex. 1018, p. 313
Microchip v. Aptiv
IPR2024-00494

In order to determine if bandwidth can be allocated, or if a transaction can be fit into a particular (micro)frame, the maximum transaction execution time must be calculated. The calculation of the maximum transaction execution time requires that the following information be provided: (Note that an agent other than the client may provide some of this information.)

- Number of data bytes (*wMaxPacketSize*) to be transmitted.
- Transfer type.
- Depth in the topology. If less precision is allowed, the maximum topology depth may be assumed.

This calculation must include the bit transmission time, the signal propagation delay through the topology, and any implementation-specific delays, such as preparation or recovery time required by the Host Controller itself. Refer to Chapter 5 for examples of formulas that can be used for such calculations.

10.3.3 Data Transfers

The basis for all client-function communication is the interface: a bundle of related pipes associated with a particular USB device.

Exactly one client on the host manages a given interface. The client initializes each pipe of an interface by setting the Policy for that pipe. This includes the maximum amount of data to be transmitted per IRP and the maximum service interval for the pipe. A service interval is stated in milliseconds and describes the interval over which an IRP's data will be transmitted for an isochronous pipe. It describes the polling interval for an interrupt pipe. The client is notified when a specified request is completed. The client manages the size of each IRP such that its duty cycle and latency constraints are maintained. Additional Policy information includes the notification information for the client.

The client provides the buffer space required to hold the transmitted data. The USB System uses the Policy to determine the additional working space it will require.

The client views its data as a contiguous serial stream, which it manages in a similar manner to those streams provided over other types of bus technologies. Internally, the USB System may, depending on its own Policy and any Host Controller constraints, break the client request down into smaller requests to be sent across the USB. However, two requirements must be met whenever the USB System chooses to undertake such division:

- The division of the data stream into smaller chunks is not visible to the client.
- USB samples are not split across bus transactions.

When a client wishes to transfer data, it will send an IRP to the USBD. Depending on the direction of data transfer, a full or empty data buffer will be provided. When the request is complete (successfully or due to an error condition), the IRP and its status is returned to the client. Where relevant, this status is also provided on a per-transaction basis.

10.3.4 Common Data Definitions

In order to allow the client to receive request results as directly as possible from its device, it is desirable to minimize the amount of processing and copying required between the device and the client. To facilitate this, some control aspects of the IRP are standardized such that different layers in the stack may directly use the information provided by the client. The particular format for this data is dependent on the actualization of the USBDI in the operating system. Some data elements may in fact not be directly visible to the client at all but are generated as a result of the client request.

Microchip Ex. 1018, p. 314
Microchip v. Aptiv
IPR2024-00494

The following data elements define the relevant information for a request:

- Identification of the pipe associated with the request. Identifying this pipe also describes information such as transfer type for this request.
- Notification identification for the particular client.
- Location and length of data buffer that is to be transmitted or received.
- Completion status for the request. Both the summary status and, as required, detailed per-transaction status must be provided.
- Location and length of working space. This is implementation-dependent.

The actual mechanisms used to communicate requests to the USBD are operating system-specific. However, beyond the requirements stated above for what request-related information must be available, there are also requirements on how requests will be processed. The basic requirements are described in Chapter 5. Additionally, the USBD provides a mechanism to designate a group of isochronous IRPs for which the transmission of the first transaction of each IRP will occur in the same (micro)frame. The USBD also provides a mechanism for designating an uninterruptable set of vendor- or class-specific requests to a default pipe. No other requests to that default pipe, including standard, class, or vendor request may be inserted in the execution flow for such an uninterruptable set. If any request in this set fails, the entire set is retired.

10.4 Host Controller Driver

The Host Controller Driver (HCD) is an abstraction of Host Controller hardware and the Host Controller's view of data transmission over the USB. The HCDI meets the following requirements:

- Provides an abstraction of the Host Controller hardware.
- Provides an abstraction for data transfers by the Host Controller across the USB interconnect.
- Provides an abstraction for the allocation (and de-allocation) of Host Controller resources to support guaranteed service to USB devices.
- Presents the root hub and its behavior according to the hub class definition. This includes supporting the root hub such that the hub driver interacts with the root hub exactly as it would for any hub. In particular, even though a root hub can be implemented in a combination of hardware and software, the root hub responds initially to the default device address (from a client perspective), returns descriptor information, supports having its device address set, and supports the other hub class requests. However, bus transactions may or may not need to be generated to accomplish this behavior given the close integration possible between the Host Controller and the root hub.

The HCD provides a software interface (HCDI) that implements the required abstractions. The function of the HCD is to provide an abstraction, which hides the details of the Host Controller hardware. Below the Host Controller hardware is the physical USB and all the attached USB devices.

The HCD is the lowest tier in the USB software stack. The HCD has only one client: the Universal Serial Bus Driver (USBD). The USBD maps requests from many clients to the appropriate HCD. A given HCD may manage many Host Controllers.

The HCDI is not directly accessible from a client. Therefore, the specific interface requirements for the HCDI are not discussed here.

10.5 Universal Serial Bus Driver

The USBD provides a collection of mechanisms that operating system components, typically device drivers, use to access USB devices. The only access to a USB device is that provided by the USBD. The USBD implementations are operating system-specific. The mechanisms provided by the USBD are implemented, using as appropriate and augmenting as necessary, the mechanisms provided by the operating system environment in which the USB runs. The following discussion centers on the basic capabilities

Microchip Ex. 1018, p. 315
Microchip v. Aptiv
IPR2024-00494

required for all USBD implementations. For specifics of the USBD operation within a specific environment, see the relevant operating system environment guide for the USBD. A single instance of the USBD directs accesses to one or more HCDs that in turn connect to one or more Host Controllers. If allowed, how USBD instancing is managed is dependent upon the operating system environment. However, from the client's point of view, the USBD with which the client communicates manages all of the attached USB devices.

10.5.1 USBD Overview

Clients of USBD direct commands to devices or move streams of data to or from pipes. The USBD presents two groups of software mechanisms to clients: command mechanisms and pipe mechanisms.

Command mechanisms allow clients to configure and control USBD operation as well as to configure and generically control a USB device. In particular, command mechanisms provide all access to the device's default pipe.

Pipe mechanisms allow a USBD client to manage device specific data and control transfers. Pipe mechanisms do not allow a client to directly address the device's default pipe.

Figure 10-5 presents an overview of the USBD structure.

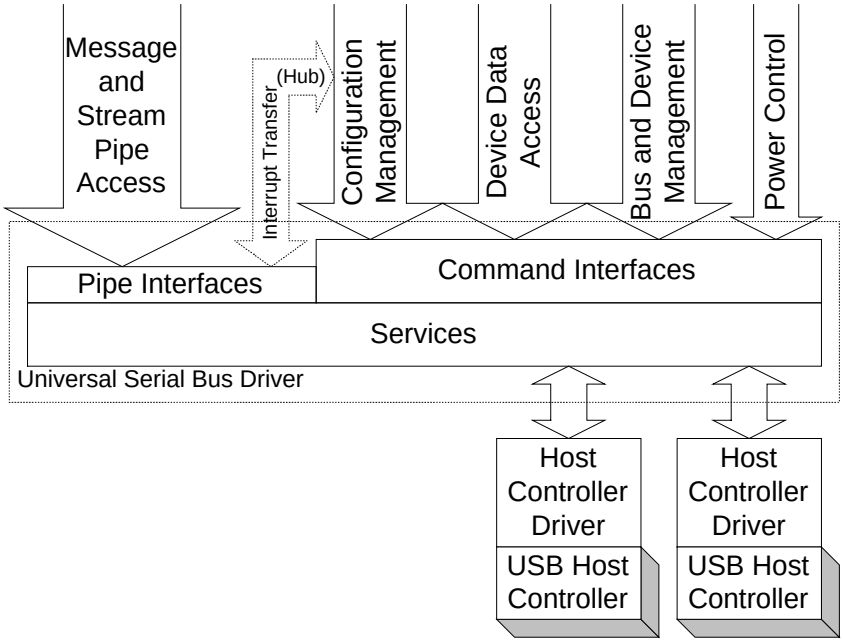


Figure 10-5. Universal Serial Bus Driver Structure

10.5.1.1 USBD Initialization

Specific USBD initialization is operating system-dependent. When a particular USB managed by USBD is initialized, the management information for that USB is also created. Part of this management information is the default address device and its default pipe used to communicate to a newly reset device.

When a device is attached to a USB, it responds to a special address known as the default address (refer to Chapter 9) until its unique address is assigned by the bus enumerator. In order for the USB System to interact with the new device, the default device address and the device's default pipe must be available to the hub driver when a device is attached. During device initialization, the default address is changed to a unique address.

Microchip Ex. 1018, p. 316
Microchip v. Aptiv
IPR2024-00494

10.5.1.2 USBD Pipe Usage

Pipes are the method by which a device endpoint is associated with a Host Software entity. Pipes are owned by exactly one such entity on the host. Although the basic concept of a pipe is the same no matter who the owner, some distinction of capabilities provided to the USBD client occurs between two groups of pipes:

- Default pipes, which are owned and managed by the USBD
- All other pipes, which are owned and managed by clients of the USBD

Default pipes are never directly accessed by clients, although they are often used to fulfill some part of client requests relayed via command mechanisms.

10.5.1.2.1 Default Pipes

The USBD is responsible for allocating and managing appropriate buffering to support transfers on the default pipe that are not directly visible to the client such as setting a device address. For those transfers that are directly visible to the client, such as sending vendor and class commands or reading a device descriptor, the client must provide the required buffering.

10.5.1.2.2 Client Pipes

Any pipe not owned and managed by the USBD can be owned and managed by a USBD client. From the USBD viewpoint, a single client owns the pipe. In fact, a cooperative group of clients can manage the pipe, provided they behave as a single coordinated entity when using the pipe.

The client is responsible for providing the amount of buffering it needs to service the data transfer rate of the pipe within a service interval attainable by the client. Additional buffering requirements for working space are specified by the USB System.

10.5.1.3 USBD Service Capabilities

The USBD provides services in the following categories:

- Configuration via command mechanisms
- Transfer services via both command and pipe mechanisms
- Event notification
- Status reporting and error recovery

10.5.2 USBD Command Mechanism Requirements

USBD command mechanisms allow a client generic access to a USB device. Generally, these commands allow the client to make read or write accesses to one of potentially several device data and control spaces. The client provides as little as a device identifier and the relevant data or empty buffer pointer.

USBD command transfers do not require that the USB device be configured. Many of the device configuration facilities provided by the USBD are command transfers.

Following are the specific requirements on the command mechanisms provided.

10.5.2.1 Interface State Control

USBD clients must be able to set a specified interface to any settable pipe state. Setting an interface state results in all of the pipes in that interface moving to that state. Additionally, all of the pipes in an interface may be reset or aborted.

Microchip Ex. 1018, p. 317
Microchip v. Aptiv
IPR2024-00494

10.5.2.2 Pipe State Control

USBD pipe state has two components:

- Host status
- Reflected endpoint status

Whenever the pipe status is reported, the value for both components will be identified. The pipe status reflected from the endpoint is the result of the endpoint being in a particular state. The USBD client manages the pipe state as reported by the USBD. For any pipe state reflected from the endpoint, the client must also interact with the endpoint to change the state.

A USBD pipe is in exactly one of the following states:

- Active: The pipe's Policy has been set and the pipe is able to transmit data. The client can query as to whether any IRPs are outstanding for a particular pipe. Pipes for which there are no outstanding IRPs are still considered to be in the Active state as long as they are able to accept new IRPs.
- Halted: An error has occurred on the pipe. This state may also be a reflection of the corresponding Halted endpoint on the device.

A pipe and endpoint are considered active when the device is configured and the pipe and/or endpoint is not stalled. Clients may manipulate pipe state in the following ways:

- Aborting a Pipe: All of the IRPs scheduled for a pipe are retired immediately and returned to the client with a status indicating they have been aborted. Neither the host state nor the reflected endpoint state of the pipe is affected.
- Resetting a Pipe: The pipe's IRPs are aborted. The host state is moved to Active. If the reflected endpoint state needs to be changed, that must be commanded explicitly by the USBD client.
- Clearing a Halted pipe: The pipe's state is cleared from *Halted* to *Active*.
- Halting a Pipe: The pipe's state is set to *Halted*.

10.5.2.3 Getting Descriptors

The USBDI must provide a mechanism to retrieve standard device, configuration, and string descriptors, as well as any class- or vendor-specific descriptors.

10.5.2.4 Getting Current Configuration Settings

The USBDI must provide a facility to return, for any specified device, the current configuration descriptor. If the device is not configured, no configuration descriptor is returned. This action is equivalent to returning the configuration descriptor for the current configuration by requesting the specific configuration descriptor. It does not, however, require the client to know the identifier for the current configuration. This will return all of the configuration information, including the following:

- All of the configuration descriptor information as stored on the device, including all of the alternate settings for all of the interfaces
- Indicators for which of the alternate settings for interfaces are active
- Pipe handles for endpoints in the active alternate settings for interfaces
- Actual *wMaxPacketSize* values for endpoints in the active alternate settings for interfaces

Additionally, for any specified pipe, the USBDI must provide a facility to return the *wMaxPacketSize* that is currently being used by the pipe.

Microchip Ex. 1018, p. 318
Microchip v. Aptiv
IPR2024-00494

10.5.2.5 Adding Devices

The USBDI must provide a mechanism for the hub driver to inform USBD of the addition of a new device to a specified USB and to retrieve the USBD ID of the new USB device. The USBD tasks include assigning the device address and preparing the device's default pipe for use.

10.5.2.6 Removing Devices

The USBDI must provide a facility for the hub driver to inform the USBD that a specific device has been removed.

10.5.2.7 Managing Status

The USBDI must provide a mechanism for obtaining and clearing device-based status on a device, interface, or pipe basis.

10.5.2.8 Sending Class Commands

This USBDI mechanism is used by a client, typically a class-specific or adaptive driver, to send one or more class-specific commands to a device.

10.5.2.9 Sending Vendor Commands

This USBDI mechanism is used by a client to send one or more vendor-specific commands to a device.

10.5.2.10 Establishing Alternate Settings

The USBDI must provide a mechanism to change the alternate setting for a specified interface. As a result, the pipe handles for the previous setting are released and new pipe handles for the interface are returned. For this request to succeed, the interface must be idle; i.e., no data buffers may be queued for any pipes in the interface.

10.5.2.11 Establishing a Configuration

Configuring software requests a configuration by passing a buffer containing the configuration information to the USBD. The USBD requests resources for the endpoints in the configuration, and if all resource requests succeed, the USBD sets the device configuration and returns interface handles with corresponding pipe handles for all of the active endpoints. The default values are used for all alternate settings for interfaces.

Note: The interface implementing the configuration may require specific alternate settings to be identified.

10.5.2.12 Setting Descriptors

For devices supporting this behavior, the USBDI allows existing descriptors to be updated or new descriptors to be added.

10.5.3 USBD Pipe Mechanisms

This part of the USBDI offers clients the highest-speed, lowest overhead data transfer services possible. Higher performance is achieved by shifting some pipe management responsibilities from the USBD to the client. As a result, the pipe mechanisms are implemented at a more primitive level than the data transfer services provided by the USBD command mechanisms. Pipe mechanisms do not allow access to a device's default pipe.

USBD pipe transfers are available only after both the device and USB configuration have completed successfully. At the time the device is configured, the USBD requests the resources required to support all

Microchip Ex. 1018, p. 319
Microchip v. Aptiv
IPR2024-00494

device pipes in the configuration. Clients are allowed to modify the configuration, constrained by whether the specified interface or pipe is idle.

Clients provide full buffers to outgoing pipes and retrieve transfer status information following the completion of a request. The transfer status returned for an outgoing pipe allows the client to determine the success or failure of the transfer.

Clients provide empty buffers to incoming pipes and retrieve the filled buffers and transfer status information from incoming pipes following the completion of a request. The transfer status returned for an incoming pipe allows a client to determine the amount and the quality of the data received.

10.5.3.1 Supported Pipe Types

The four types of pipes supported, based on the four transfer types, are described in the following sections.

10.5.3.1.1 Isochronous Data Transfers

Each buffer queued for an isochronous pipe is required to be viewable as a stream of samples. As with all pipe transfers, the client establishes a Policy for using this isochronous pipe, including the relevant service interval for this client. Lost or missing bytes, which are detected on input, and transmission problems, which are noted on output, are indicated to the client.

The client queues a first buffer, starting the pipe streaming service. To maintain the continuous streaming transfer model used in all isochronous transfers, the client queues an additional buffer before the current buffer is retired.

The USBD is required to be able to provide a sample stream view of the client's data stream. In other words, using the client's specified method of synchronization, the precise packetization of the data is hidden from the client. Additionally, a given transaction is always contained completely within some client data buffer.

For an output pipe, the client provides a buffer of data. The USBD allocates the data across the (micro)frames for the service period using the client's chosen method of synchronization.

For an input pipe, the client must provide an empty buffer large enough to hold the maximum number of bytes the client's device will deliver in the service period. Where missing or invalid bytes are indicated, the USBD may leave the space that the bytes would have occupied in place in the buffer and identify the error. One of the consequences of using no synchronization method is that this reserved space is assumed to be the maximum packet size. The buffer-retired notification occurs when the IRP completes. Note that the input buffer need not be full when returned to the client.

The USBD may optionally provide additional views of isochronous data streams. The USBD is also required to be able to provide a packet stream view of the client's data stream.

10.5.3.1.2 Interrupt Transfers

The Interrupt out transfer originates in the client of the USBD and is delivered to the USB device. The Interrupt in transfer originates in a USB device and is delivered to a client of the USBD. The USB System guarantees that the transfers meet the maximum latency specified by the USB endpoint descriptor.

The client queues a buffer large enough to hold the interrupt transfer data (typically a single USB transaction). When all of the data is transferred, or if the error threshold is exceeded, the IRP is returned to the client.

10.5.3.1.3 Bulk Transfers

Bulk transfers may originate either from the device or the client. No periodicity or guaranteed latency is assumed. When all of the data is transferred, or if the error threshold is exceeded, the IRP is returned to the client.

Microchip Ex. 1018, p. 320
Microchip v. Aptiv
IPR2024-00494

10.5.3.1.4 Control Transfers

All message pipes transfer data in both directions. In all cases, the client outputs a setup stage to the device endpoint. The optional data stage may be either input or output and the final status is always logically presented to the host. For details of the defined message protocol, refer to Chapter 8.

The client prepares a buffer specifying the command phase and any optional data or empty buffer space. The client receives a buffer-retired notification when all phases of the control transfer are complete, or an error notification, if the transfer is aborted due to transmission error.

10.5.3.2 USBD Pipe Mechanism Requirements

The following pipe mechanisms are provided.

10.5.3.2.1 Aborting IRPs

The USBDI must allow IRPs for a particular pipe to be aborted.

10.5.3.2.2 Managing Pipe Policy

The USBDI must allow a client to set and clear the Policy for an individual pipe or for an entire interface. Any IRPs made by the client prior to successfully setting a Policy are rejected by the USBD.

10.5.3.2.3 Queuing IRPs

The USBDI must allow clients to queue IRPs for a given pipe. When IRPs are returned to the client, the request status is also returned. A mechanism is provided by the USBD to identify a group of isochronous IRPs whose first transactions will all occur in the same (micro)frame.

10.5.4 Managing the USB via the USBD Mechanisms

Using the provided USBD mechanisms, the following general capabilities are supported by any USB System.

10.5.4.1 Configuration Services

Configuration services operate on a per-device basis. The configuring software tells the USBD when to perform device configuration. A hub driver has a special role in device management and provides at least the following capabilities:

- Device attach/detach recognition, driven by an interrupt pipe owned by the hub driver
- Device reset, accomplished by the hub driver by resetting the hub port upstream of the device
- Tells the USBD to perform device address assignment
- Power control

The USBDI additionally provides the following configuration facilities, which may be used by the hub driver or other configuring software available on the host:

- Device identification and access to configuration information (via access to descriptors on the device)
- Device configuration via command mechanisms

When the hub driver informs the USBD of a device attachment, the USBD establishes the default pipe for the new device.

Microchip Ex. 1018, p. 321
Microchip v. Aptiv
IPR2024-00494

10.5.4.1.1 Configuration Management

Configuration management services are provided primarily as a set of specific interface commands that generate USB transactions on the default pipe. The notable exception is the use of an additional interrupt pipe that delivers hub status directly to the hub driver.

Every hub initiates an interrupt transfer when there is a change in the state of one of the hub ports. Generally, the port state change will be the connection or removal of a downstream USB device. (Refer to Chapter 11 for more information.)

10.5.4.1.2 Initial Device Configuration

The device configuration process begins when a hub reports, via its status change pipe, the connection of a new USB device.

Configuration management services allow configuring software to select a USB device configuration from the set of configurations listed in the device. The USBD verifies that adequate power is available and the data transfer rates given for all endpoints in the configuration do not exceed the capabilities of the USB with the current schedule before setting the device configuration.

10.5.4.1.3 Modifying a Device Configuration

Configuration management services allow configuring software to replace a USB device configuration with another configuration from the set of configurations listed in the device. The operation succeeds if adequate power is available and the data transfer rates given for all endpoints in the new configuration fit within the capabilities of the USB with the current schedule. If the new configuration is rejected, the previous configuration remains.

Configuration management services allow configuring software to return a USB device to a Not Configured state.

10.5.4.1.4 Device Removal

Error recovery and/or device removal processing begins when a hub reports via its status change pipe that the USB device has been removed.

10.5.4.2 Power Control

There are two cooperating levels of power management for the USB: bus and device level management. This specification provides mechanisms for managing power on the USB bus. Device classes may define class-specific power control capabilities.

All USB devices must support the Suspended state (refer to Chapter 9). The device is placed into the Suspended state via control of the hub port to which the device is attached. Normal device operation ceases in the Suspend State; however, if the device is capable of wakeup signaling and the device is enabled for remote wakeup, it may generate resume signaling in response to external events.

The power management system may transition a device to the Suspended state or power-off the device in order to control and conserve power. The USB provides neither requirements nor commands for the device state to be saved and restored across these transitions. Device classes may define class-specific device state save-and-restore capabilities.

The USB System coordinates the interaction between device power states and the Suspended state.

It is recommended that while a device is not being used by the system (i.e., no transactions are being transmitted to or from the device besides SOF tokens), the device be suspended as soon as possible by selectively suspending the port to which the device is attached. Suspending inactive devices reduces reliability issues due to high currents passing through a transceiver operating in high-speed mode in the presence of short circuit conditions described in Section 7.1.1. Some of these short circuit conditions are not detectable in the absence of transactions to the device. Suspending the unused device will place the

Microchip Ex. 1018, p. 322
Microchip v. Aptiv
IPR2024-00494

transceiver interface into full-speed mode which has a greater reliability in the presence of short circuit conditions.

10.5.4.3 Event Notifications

USBD clients receive several kinds of event notifications through a number of sources:

- Completion of an action initiated by a client.
- Interrupt transfers over stream pipes can deliver notice of device events directly to USBD clients. For example, hubs use an interrupt pipe to deliver events corresponding to changes in hub status.
- Event data can be embedded by devices in streams.
- Standard device interface commands, device class commands, vendor-specific commands, and even general control transfers over message pipes can all be used to poll devices for event conditions.

10.5.4.4 Status Reporting and Error Recovery Services

The command and pipe mechanisms both provide status reporting on individual requests as they are invoked and completed.

Additionally, USB device status is available to USBD clients using the command mechanisms.

The USBD provides clients with pipe error recovery mechanisms by allowing pipes to be reset or aborted.

10.5.4.5 Managing Remote Wakeup Devices

The USB System can minimize the resume power consumption of a suspended USB tree. This is accomplished by explicitly enabling devices capable of resume signaling and controlling propagation of resume signaling via selectively suspending and/or disabling hub ports between the device and the nearest self-powered, awake hub.

In some error-recovery scenarios, the USB System will need to re-enumerate sub-trees. The sub-tree may be partially or completely suspended. During error-recovery, the USB System must avoid contention between a device issuing resume signaling and simultaneously driving reset down the port. Avoidance is accomplished via management of the devices' remote wakeup feature and the hubs' port features. The rules are as follows:

- Issue a SetDeviceFeature(DEVICE_REMOTE_WAKEUP) request to the leaf device, only just prior to selectively suspending any port between where the device is connected and the root port (via a SetPortFeature(PORT_SUSPEND) request).
- Do not reset a suspended port that has had a device enabled for remote wakeup without first enabling that port.
- Verify that after a remote wakeup, the devices in the subtree affected by the remote wakeup are still present. This will typically be done as part of determining which potential remote wakeup device was the source of the wakeup. This should be done to ensure that a suspended device is not disconnected (and possibly reconnected) or reset (e.g., by noise) during a suspend/resume process.

10.5.5 Passing USB Preboot Control to the Operating System

A single software driver owns the Host Controller. If the host system implements USB services before the operating system loads, the Host Controller must provide a mechanism that disables access by the preboot software and allows the operating system to gain control. Preboot USB configuration is not passed to the operating system. Once the operating system gains control, it is responsible to fully configure the bus. If the operating system provides a mechanism to pass control back to the preboot environment, the bus will be in an unknown state. The preboot software should treat this event as a powerup.

Microchip Ex. 1018, p. 323
Microchip v. Aptiv
IPR2024-00494

10.6 Operating System Environment Guides

As noted previously, the actual interfaces between the USB System and host software are specific to the host platform and operating system. A companion specification is required for each combination of platform and operating system with USB support. These specifications describe the specific interfaces used to integrate the USB into the host. Each operating system provider for the USB System identifies a compatible Universal USB Specification revision.

Microchip Ex. 1018, p. 324
Microchip v. Aptiv
IPR2024-00494

Chapter 11
Hub Specification

This chapter describes the architectural requirements for the USB hub. It contains a description of the three principal sub-blocks: the Hub Repeater, the Hub Controller, and the Transaction Translator. The chapter also describes the hub's operation for error recovery, reset, and suspend/resume. The second half of the chapter defines hub request behavior and hub descriptors.

The hub specification supplies sufficient additional information to permit an implementer to design a hub that conforms to the USB specification.

11.1 Overview

Hubs provide the electrical interface between USB devices and the host. Hubs are directly responsible for supporting many of the attributes that make USB user friendly and hide its complexity from the user. Listed below are the major aspects of USB functionality that hubs must support:

- Connectivity behavior
- Power management
- Device connect/disconnect detection
- Bus fault detection and recovery
- High-, full-, and low-speed device support

A hub consists of three components: the Hub Repeater, the Hub Controller, and the Transaction Translator. The Hub Repeater is responsible for connectivity setup and tear-down. It also supports exception handling, such as bus fault detection and recovery and connect/disconnect detect. The Hub Controller provides the mechanism for host-to-hub communication. Hub-specific status and control commands permit the host to configure a hub and to monitor and control its individual downstream facing ports. The Transaction Translator responds to high-speed split transactions and translates them to full-/low-speed transactions with full-/low-speed devices attached on downstream facing ports.

11.1.1 Hub Architecture

Figure 11-1 shows a hub and the locations of its upstream and downstream facing ports. A hub consists of a Hub Repeater section, a Hub Controller section, and a Transaction Translator section. The hub must operate at high-speed when its upstream facing port is connected at high-speed. The hub must operate at full-speed when its upstream facing port is connected at full-speed.

The Hub Repeater is responsible for managing connectivity between upstream and downstream facing ports which are operating at the same speed. The Hub Repeater supports full-/low-speed connectivity and high-speed connectivity. The Hub Controller provides status and control and permits host access to the hub. The Transaction Translator takes high-speed split transactions and translates them to full-/low-speed transactions when the hub is operating at high-speed and has full-/low-speed devices attached. The operating speed of a device attached on a downstream facing port determines whether the Routing Logic connects a port to the Transaction Translator or hub repeater sections.

Microchip Ex. 1018, p. 325
Microchip v. Aptiv
IPR2024-00494

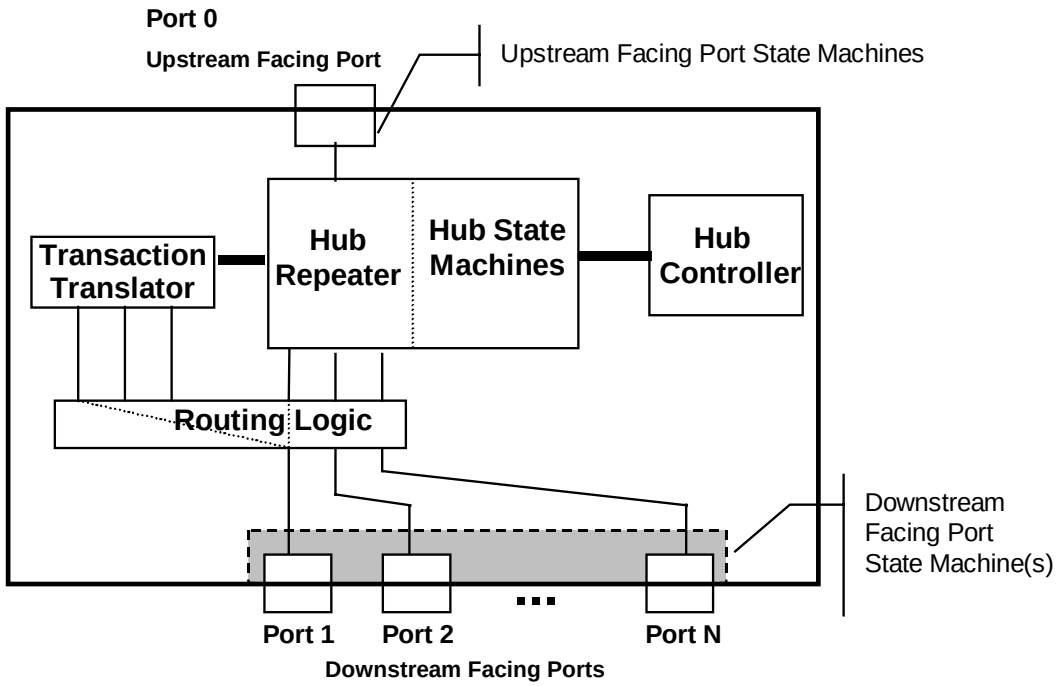


Figure 11-1. Hub Architecture

When a hub's upstream facing port is attached to an electrical environment that is operating at full-/low-speed, the hub's high-speed functionality is disallowed. This means that the hub will only operate at full-/low-speed and the transaction translator and high-speed repeater will not operate. In this electrical environment, the hub repeater must operate as a full-/low-speed repeater and the routing logic connects ports to the hub repeater.

When the hub upstream facing port is attached to an electrical environment that is operating at high-speed, the full-/low-speed hub repeater is not operational. In this electrical environment when a high-speed device is attached on downstream facing port, the routing logic will connect the port to the hub repeater and the hub repeater must operate as a high-speed repeater. In this case, when a full-/low-speed device is attached on a downstream facing port, the routing logic must connect the port to the transaction translator.

11.1.2 Hub Connectivity

Hubs exhibit different connectivity behavior depending on whether they are propagating packet traffic, or resume signaling, or are in the Idle state.

11.1.2.1 Packet Signaling Connectivity

The Hub Repeater contains one port that must always connect in the upstream direction (referred to as the upstream facing port) and one or more downstream facing ports. Upstream connectivity is defined as being towards the host, and downstream connectivity is defined as being towards a device. Figure 11-2 shows the packet signaling connectivity behavior for hubs in the upstream and downstream directions. A hub also has an Idle state, during which the hub makes no connectivity. When in the Idle state, all of the hub's ports are in the receive mode waiting for the start of the next packet.

Microchip Ex. 1018, p. 326
Microchip v. Aptiv
IPR2024-00494

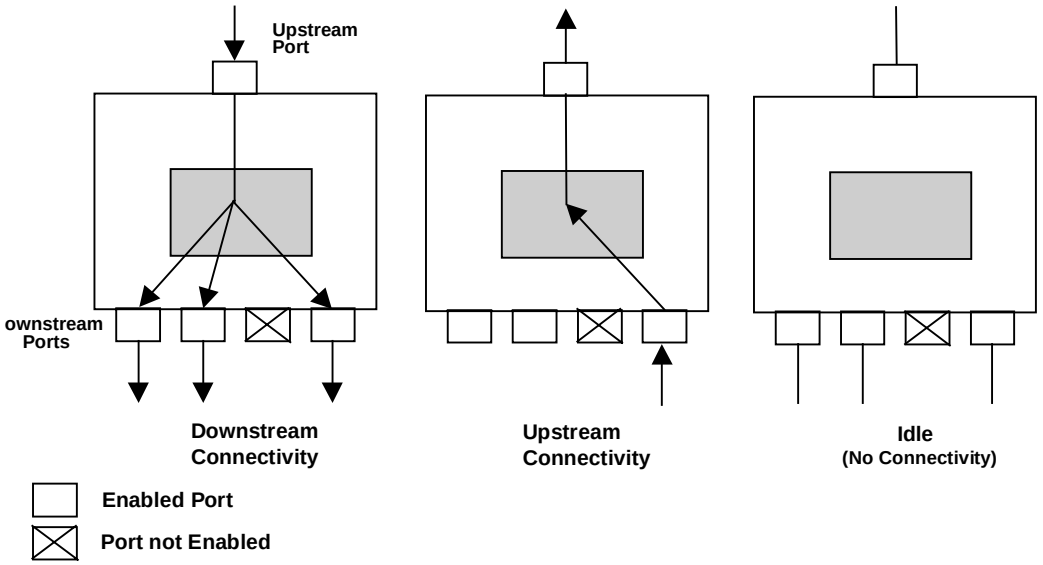


Figure 11-2. Hub Signaling Connectivity

If a downstream facing port is enabled (i.e., in a state where it can propagate signaling through the hub), and the hub detects the start of a packet on that port, connectivity is established in an upstream direction to the upstream facing port of that hub, but not to any other downstream facing ports. This means that when a device or a hub transmits a packet upstream, only those hubs in line between the transmitting device and the host will see the packet. Refer to Section 11.8.3 for optional behavior when a hub detects simultaneous upstream signaling on more than one port.

In the downstream direction, hubs operate in a broadcast mode. When a hub detects the start of a packet on its upstream facing port, it establishes connectivity to all enabled downstream facing ports. If a port is not enabled, it does not propagate packet signaling downstream.

11.1.2.2 Resume Connectivity

Hubs exhibit different connectivity behaviors for upstream- and downstream-directed resume signaling. A hub that is suspended reflects resume signaling from its upstream facing port to all of its enabled downstream facing ports. Figure 11-3 illustrates hub upstream and downstream resume connectivity.

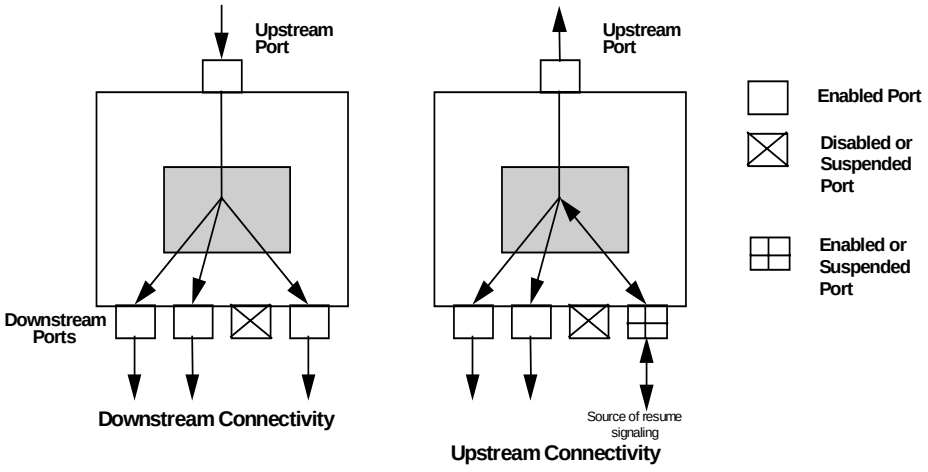


Figure 11-3. Resume Connectivity

Microchip Ex. 1018, p. 327
Microchip v. Aptiv
IPR2024-00494

If a hub is suspended and detects resume signaling from a selectively suspended or an enabled downstream facing port, the hub reflects that signaling upstream and to all of its enabled downstream facing ports, including the port that initiated the resume sequence. Resume signaling is not reflected to disabled or suspended ports. A detailed discussion of resume connectivity appears in Section 11.9.

11.1.2.3 Hub Fault Recovery Mechanisms

Hubs are the essential USB component for establishing connectivity between the host and other devices. It is vital that any connectivity faults, especially those that might result in a deadlock, be detected and prevented from occurring. Hubs need to handle connectivity faults only when they are in the repeater mode.

Hubs must also be able to detect and recover from lost or corrupted packets that are addressed to the Hub Controller. Because the Hub Controller is, in fact, another USB device, it must adhere to the same timeout rules as other USB devices, as described in Chapter 8.

11.2 Hub Frame/Microframe Timer

Each hub has a (micro)frame timer whose timing is derived from the hub's local clock and is synchronized to the host (micro)frame period by the host-generated Start-of-(micro)frame (SOF). The (micro)frame timer provides timing references that are used to allow the hub to detect a babbling device and prevent the hub from being disabled by the upstream hub. The hub (micro)frame timer must track the host (micro)frame period and be capable of remaining synchronized with the host even if two consecutive SOF tokens are missed by the hub.

The (micro)frame timer must lock to the host's (micro)frame timing for worst case clock accuracies and timing offsets between the host and hub. There are specific requirements for hubs when their upstream facing port is operating at high-speed and full-speed.

11.2.1 High-speed Microframe Timer Range

The range for a microframe timer must be from 59904 to 60096 high-speed bits.

The nominal microframe interval is 60000 high-speed bit times. The hub microframe timer range specified above is 60000 +/- 96 high-speed bit times in order to accommodate host accuracy, hub accuracy, repeater jittter, and hub quantization. The +/-96 full-speed bit time variation is calculated in Table 11-2.

Table 11-1. High-speed Microframe Timer Range Contributions

Source of Variation	Variation (ppm)	Variation (bits) Over One Microframe Interval	Comment
Host accuracy	+/- 500	+/- 30	
Hub accuracy	+/- 500	+/- 30	
Host jitter		+/- 2	
Hub chain jitter		+/- 20	Four hubs in series upstream of hub; 0 to 5 bits of jitter per hub
Quantization		+/-14	Bits need to round total variation to multiple of 16

Microchip Ex. 1018, p. 328
Microchip v. Aptiv
IPR2024-00494

11.2.2 Full-speed Frame Timer Range

The range of the frame timer must be from 11958 to 12042 full-speed bits.

The nominal frame interval is 12000 full-speed bit times. The hub frame timer range specified above is 12000 +/- 42 full-speed bit times in order to accommodate host accuracy and hub accuracy. The +/-42 full-speed bit time variation is calculated in Table 11-2.

Table 11-2. Full-speed Frame Timer Range Contributions

Source of Variation	Variation (ppm)	Variation (bits) Over One Frame Interval	Comment
Host accuracy	+/- 500	+/- 6	
Hub accuracy	+/- 3000	+/- 36	+/-6 bits due to hub accuracy (500 ppm) +/-30 bits due to 1.x parent hub accuracy (2500 ppm)

11.2.3 Frame/Microframe Timer Synchronization

A hub's (micro)frame timer is clocked by the hub's clock source and is synchronized to SOF packets that are derived from the host's (micro)frame timer. After a reset or resume, the hub's (micro)frame timer is not synchronized. Whenever the hub receives two consecutive SOF packets, its (micro)frame timer must be synchronized. Synchronized is synonymous with lock(ed). An example for a method of constructing a timer that properly synchronizes is as follows.

11.2.3.1 Example (Micro)frame Timer Synchronization Method

The hub maintains three timer values: (micro)frame timer (down counter), current (micro)frame (up counter), and next (micro)frame (register). After a reset or resume, a flag is set to indicate that the (micro)frame timer is not synchronized.

When the first SOF token is detected, the current (micro)frame timer resets and starts counting once per hub bit time. On the next SOF, if the timer has not rolled over, the value in the current (micro)frame timer is loaded into the next (micro)frame register and into the (micro)frame timer. The current (micro)frame timer is reset to zero and continues to count and the flag is set to indicate that the (micro)frame timer is locked. The (micro)frame timer rolls over when the count exceeds 60096 for high-speed or 12042 for full-speed (a test at 65535 for high-speed or 16383 for full-speed is adequate). If the current (micro)frame timer has rolled over, then an SOF was missed and the (micro)frame timer and next (micro)frame values are not loaded. When an SOF is missed, the flag indicating that the timer is not synchronized remains set.

Whenever the (micro)frame timer counts down to zero, the current value of the next (micro)frame register is loaded into the (micro)frame timer. When an SOF is detected, and the current (micro)frame timer has not rolled over, the value of the current (micro)frame timer is loaded into the (micro)frame timer and the next (micro)frame registers. The current (micro)frame timer is then reset to zero and continues to count. If the current (micro)frame timer has rolled over, then the value in the next (micro)frame register is loaded into the (micro)frame timer. This process can cause the (micro)frame timer to be updated twice in a single (micro)frame: once when the (micro)frame timer reaches zero and once when the SOF is detected.

Microchip Ex. 1018, p. 329
Microchip v. Aptiv
IPR2024-00494

11.2.3.2 EOF Advancement

The hub must advance its EOF points based on its SOF decode time in order to ensure that in the tiered topology, hubs farther away from the host will always have later EOF points than hubs nearer to the host. The magnitude of advance is implementation-dependent; the possible range of advance is derived below.

The synchronization circuit described above depends on successfully decoding an SOF packet identifier (PID). This means that the (micro)frame timer will be synchronized to a time that is later than the synchronization point in the SOF packet: later by at least 40 bit times for high-speed or 16 bit times for full-speed. Each implementation also takes some time to react to the SOF decode and set the appropriate timer/counter values. This reaction time is implementation-dependent but is assumed to be less than 192 bit times for high-speed and four bit times for full-speed. Subsequent sections describe the actions that are controlled by the (micro)frame timer. These actions are defined at the EOF1, EOF2, and EOF. EOF1 and EOF2 are defined in later sections. These sections assume that the hub's (micro)frame timer will count to zero at the end of the (micro)frame (EOF). The circuitry described above will have the (micro)frame timer counting to zero after 40 to 192 for high-speed bit times or 16-20 full-speed bit times after the start of a (micro)frame (or end of previous (micro)frame). The timings and bit offsets in the later sections must be advanced to account for this delay (i.e., add 40-192 for high-speed or 16-20 bit times for full-speed to the EOF1 and EOF2 points).

Advancing the EOF points by the processing delay ensures that the spread between the EOFs is only due to the propagation delay. For example, for high-speed, the maximum spread between 2 EOF points anywhere on the USB is less than 216 bits (144 + 72). 144 bit times are due to 36 bit times of max latency through 4 repeaters. 72 bit times are due to five maximum cable and interconnect delays of 30 ns each. As can be seen in Figure 11-4 without EOF advancement, a hub with a larger tier number could have an EOF occuring earlier than a hub with a smaller tier number. In Figure 11-5 with EOF advancement ensures that in the tiered topology, hubs with larger tier numbers always have later EOF points than hubs with smaller tier numbers. Note: 13 bit times in the figures is an example maximum cable delay (approximately 30 ns).

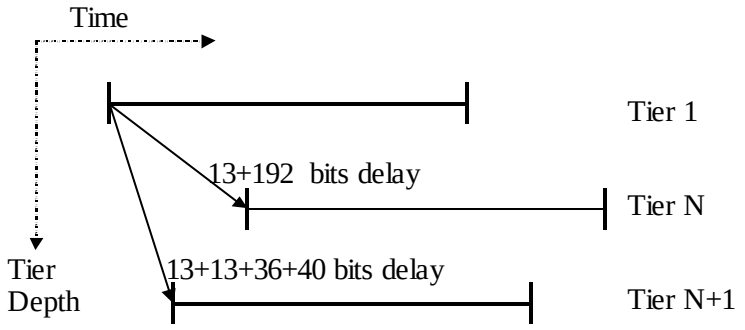


Figure 11-4. Example High-speed EOF Offsets Due to Propagation Delay Without EOF Advancement

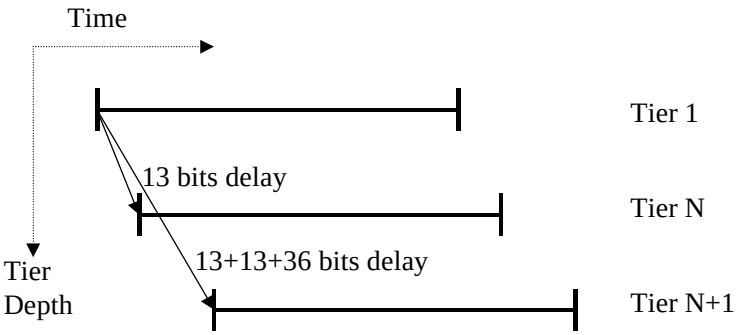


Figure 11-5. Example High-speed EOF Offsets Due to Propagation Delay With EOF Advancement

Microchip Ex. 1018, p. 330
Microchip v. Aptiv
IPR2024-00494

11.2.3.3 Effect of Synchronization on Repeater Behavior

The (micro)frame timer provides an indication to the hub Repeater state machine that the (micro)frame timer has synchronized to SOF and that the (micro)frame timer is capable of generating the EOF1 and EOF2 timing points. This signal is important after a global resume because of the possibility that a full-/low-speed device may have been detached, and a low-/full-speed device attached while the host was generating a long resume (several seconds) and the disconnect cannot be detected. The new device will bias D+ and D- to appear like a K on the hub which would then be treated as an SOP and, unless inhibited, this SOP would propagate though the resumed hubs. Since the hubs would not have seen any SOFs at this point, the hubs would not be synchronized and, thus, unable to generate the EOF1 and EOF2 timing points. The only recovery from this would be for the host to reset and re-enumerate the section of the bus containing the changed device. This scenario is prevented by inhibiting any downstream facing port from establishing connectivity until the hub is locked after a resume.

11.2.4 Microframe Jitter Related to Frame Jitter

The period between the SOFs from the Transaction Translator must not vary by more than +/- 42 ns. The microframe timer count must be used by the Transaction Translator to generate SOFs to full-speed devices (and keepalives to low-speed devices) connected to it.

The SOF received at the upstream facing port of the hub is repeated with a local clock. The frequency of this clock may be a divided version of the bit rate. This could result in a quantization error and microframe-to-microframe jitter. The microframe-to-microframe jitter of a hub repeater must be between 0 and 5 bit times. This means that the latency through the repeater of consecutive SOFs must differ by less than 5 bits. A hub may register the SOF for internal use, e.g., microframe synchronization. This requires SOF PID detection. The circuitry used for internal registering of the SOF must have a jitter which is less than or equal to 16 bits. This means that the microframe timer count values between consecutive equally spaced SOFs must differ by less than or equal to 16 bits. The host controller frequency may drift over the period of a microframe resulting in microframe period jitter. The host controller source jitter for SOFs must be less than 4 bits. This means that the consecutive periods between SOFs must differ by less than 4 bits. These requirements ensure that the microframe period at the end of five hub tiers will have a jitter of less than 40 bits (4 from host controller + 4*5 from hub repeaters + 16 from the internal SOF registering). This means that the consecutive periods between SOFs as measured at any microframe timer will differ by less than 40 bits (83.3 ns at 480 Mbs). This is less than the +/- 42 ns variation allowed.

11.2.5 EOF1 and EOF2 Timing Points

The EOF1 and EOF2 are timing points that are derived from the hub's (micro)frame timer. Table 11-3 specifies the required host and hub EOF timing points for high-speed and full-speed operation.

Table 11-3. Hub and Host EOF1/EOF2 Timing Points

Label	Bit Times Before EOF for High-speed	Bit Times Before EOF for Full-speed	Notes
EOF1	560	32	End-of-(micro)frame point #1
EOF2	64	10	End-of-(micro)frame point #2

These timing points are used to ensure that devices and hubs do not interfere with the proper transmission of the SOF packet from the host. *These timing points have meaning only when the (micro)frame timer has been synchronized to the SOF.*

The host and hub (micro)frame markers, while all synchronized to the host's SOF, are subject to certain skews that dictate the placement of the EOF points. Figure 11-6 illustrates EOF2 timing point for high-

Microchip Ex. 1018, p. 331
Microchip v. Aptiv
IPR2024-00494

speed operation. Figure 11-7 illustrates the EOF1 high-speed timing point. The numbers in the figures are in high-speed bit times.

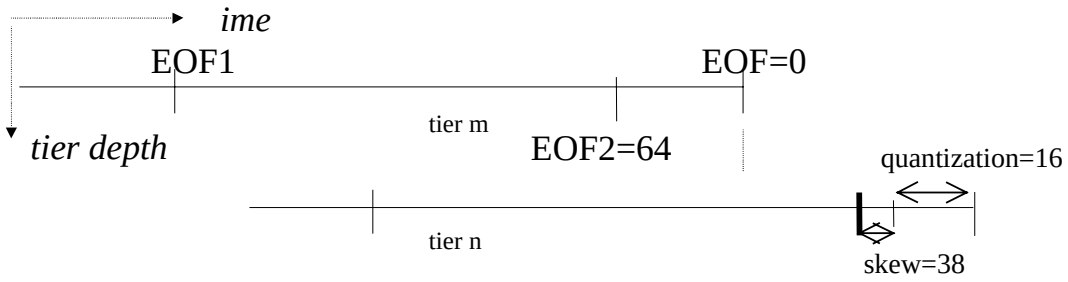


Figure 11-6. High-speed EOF2 Timing Point

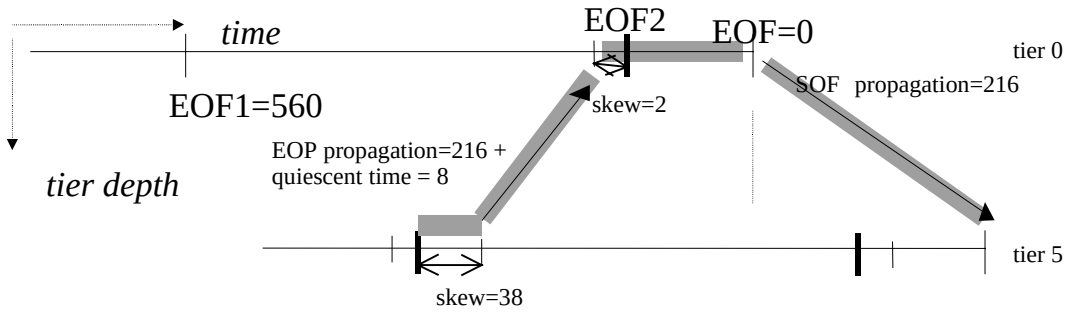


Figure 11-7. High-speed EOF1 Timing Point

At the EOF2 point, any port that has upstream connectivity will be disabled as a babbler. Hubs operating as a full-/low-speed repeater prevent becoming disabled by sending an end of packet to the upstream hub before that hub reaches its EOF2 point (i.e., at EOF1).

Figure 11-8 illustrates EOF timing points for full-/low-speed repeater operation.

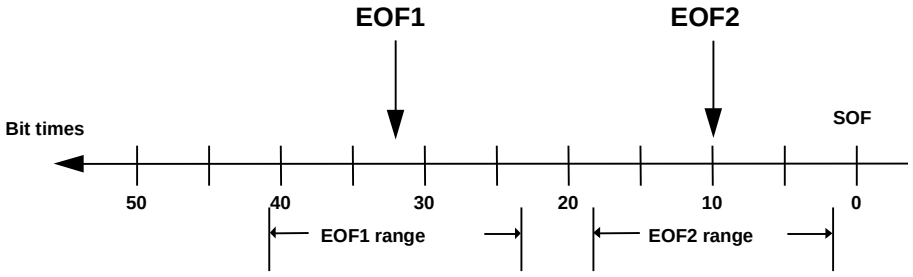


Figure 11-8. Full-speed EOF Timing Points

The hub operating as a full-/low-speed repeater is permitted to send the EOP if upstream connectivity is not established at EOF1 time. A full-speed repeater must send the EOP if connectivity is established from any downstream facing port at the EOF1 point.

A high-speed repeater must tear down upstream connectivity at the EOF1 point.

A high-speed repeater must tear down connectivity after the bus returns to the Idle state and the Elasticity buffer is emptied (as described in Section 11.7.2) rather than on decoding an EOP pattern as in full-/low-speed. Therefore, abrupt end of signaling (i.e, without a high-speed EOP) may cause malformed packets, and this must not affect repeater operation. The host controller design must be capable of processing such packets correctly.

Microchip Ex. 1018, p. 332
Microchip v. Aptiv
IPR2024-00494

11.2.5.1 High-speed EOF1 and EOF2 Timing Points

The EOF2 point is 64 bit times before EOF as shown in Figure 11-6, and the EOF1 point is 560 bit times before EOF as shown in Figure 11-7.

Although the hub is synchronized to the SOF, timing skew can accumulate between the host and a hub or between hubs. This timing skew represents the difference between different microframe timers on different hubs and the host. The total accumulated skew can be as much as 38 bit times. This is composed of ±2 bit times of (micro)frame host source jitter and 0 to 36 bit times of repeater jitter as derived earlier. This skew timing affects the placement of the EOF1 and EOF2 points.

Note: The hub skew timing assumes that the microframe interval will not be changed by the host after the microframe timers have synchronized.

EOF skew can be from –2 to + 38 bits, so all EOFs are within 256 bits (216 bits of EOF propagation delay + 40 bits of EOF skew) of each other.

Note: The EOF2 point is based on 16 bit times for quantization + 38 bit times of skew; therefore, the EOF2 point needs to located at least 54 bit times before EOF. The EOF2 point is set at 64 bit times to allow babble detection to be done with a divided (by 16) version of the bit clock. An upstream-directed packet ending before EOF1 must reach every upstream hub/host before it gets to its EOF2 point. This is achieved if the EOF1 point is located at least 544 bits before any upstream EOF (64 bits of EOF2 offset + 216 bits of EOP propagation delay + 8 bits of idle time + 216 bits of SOF propagation delay + 38 bits of EOF1 skew + 2 bits of EOF2 skew). The EOF1 point is set at 560 bit times to allow using a divided (by 16) version of the bit clock.

11.2.5.2 Full-speed EOF1 and EOF2 Timing Points

When the hub operates as a full-/low-speed repeater, the EOF1 point is 10 bit times before EOF and EOF1 is 32 bit times before EOF as shown in Figure 11-8.

The EOF2 point is defined to occur at least one bit time before the first bit of the SYNC for an SOP. The period allowed for an EOP is four full-speed bit times (the upstream facing port on a hub is always full-speed).

Although the hub is synchronized to the SOF, timing skew can accumulate between the host and a hub or between hubs. This timing skew represents the difference between different frame timers on different hubs and the host. The total accumulated skew can be as large as ±9 bit times. This is composed of ±1 bit times per frame of quantization error and ±1 bit per frame of wander. The quantization error occurs when the hub times the interval between SOFs and arrives at a value that is off by a fraction of a bit time but, due to quantization, is rounded to a full bit. Frame wander occurs when the host's frame timer is adjusted by the USB System Software so that the value sampled by the hub in a previous frame differs from the frame interval being used by the host. (Note: Such adjustment was permitted in the USB 1.0 and 1.1 specification but is no longer permitted.) These values accumulate over multiple frames because SOF packets can be lost and the hub cannot resynchronize its frame timer. This specification allows for the loss of two consecutive SOFs. During this interval, the quantization error accumulates to ±3 bit times, and the wander accumulates to ±1 ± 2 ± 3 = ±6 for a total of ±9 bit times of accumulated skew in three frames. This skew timing affects the placement of the EOF1 and EOF2 points as follows.

A hub must reach its EOF2 point one bit time before the end of the frame. In order to ensure this, a 9-bit time guard-band must be added so that the EOF2 point is set to occur when the hub's local frame timer reaches 10. A hub must complete its EOP before the hub to which it is attached reaches its EOF2 point. A hub may reach its EOF2 point nine bit times before bit time 10 (at bit time 19 before the SOF). To ensure that the EOP is completed by bit time 19, it must start before bit time 23. To ensure that the hub starts at bit time 23 with respect to another hub, a hub must set its EOF1 point nine bit times ahead of bit time 23 (at bit time 32). If a hub sets its timer to generate an EOP at bit time 32, that EOP may start as much as 9 bit times early (at bit time 41).

Microchip Ex. 1018, p. 333
Microchip v. Aptiv
IPR2024-00494

11.3 Host Behavior at End-of-Frame

It is the responsibility of the USB host controller (the host) to not provoke a response from a device if the response would cause the device to be sending a packet at the EOF2 point. Furthermore, because a hub will terminate an upstream directed packet when the hub reaches its EOF1 point, the host should not start a transaction if a response from the device (data or handshake) would be pending or in process when a hub reaches its EOF1 point. The implications of these limitations are described in the following sections.

Note: The above requirements can be met if the host controller ensures that the last transaction will complete by its EOF1. The time consumed by a transaction (and consequently the latest start time of the transaction) can be evaluated by accumulating the various delay components in the transaction. The packet lengths should include all fields and account for bit-stuffing overhead as described in Chapter 7 and Chapter 8. Formulae for calculating transaction times are located in Section 5.11.3.

In defining the timing points below, the last bit interval in a (micro)frame is designated as bit time zero. Bit times in a (micro)frame that occur before the last have values that increase the further they are from bit time zero (earlier bit times have higher numbers). These bit time designations are used for convenience only and are not intended to imply a particular implementation. The only requirement of an implementation is that the relative time relationships be preserved.

Host controllers issuing high-speed transactions on a high-speed bus must meet the above requirements. Host controllers issuing full-/low-speed transactions on a full-/low-speed bus may also use the following three behaviors near EOF.

11.3.1 Full-/low-speed Latest Host Packet

Hubs are allowed to send an EOP on their upstream facing ports at the EOF1 point if there is no downstream-directed traffic in progress at that time. To prevent potential contention, the host is not allowed to start a packet if connectivity will not be established on all connections before a hub reaches its EOF1 point. This means that the host must not start a packet after bit time 42.

Note: Although there is as much as a six-bit time delay between the time the host starts a packet and all connections are established, this time need not be added to the packet start time as this phase delay exists for the SOF packet as well, causing all hub frame timers to be phase delayed with respect to the host by the propagation delay. There is only one bit time of phase delay between any two adjacent hubs and this has been accounted for in the skew calculations.

11.3.2 Full-/low-speed Packet Nullification

If a device is sending a packet (data or handshake) when a hub in the device's upstream path reaches its EOF1 point, the hub will send a full-speed EOP. Any packet that is truncated by a hub must be discarded.

A host implementation may discard any packet that is being received at bit time 41. Alternatively, a host implementation may attempt to maximize bus utilization by accepting a packet if the packet is predicted to start at or before bit time 41.

11.3.3 Full-/low-speed Transaction Completion Prediction

A device can send two types of packets: data and handshake. A handshake packet is always exactly 16 bit times long (sync byte plus PID byte.) The time from the end of a packet from the host until the first bit of the handshake must be seen at the host is 17 bit times. This gives a total allocation of 35 bit times from the end of data packet from the root (start of EOP) until it is predicted that the handshake will be completed (start of EOP) from the device. Therefore, if the host is sending a data packet for which the device can return a handshake (anything other than an isochronous packet), then if the host completes the data packet and starts sending EOP before bit time 76, then the host can predict that the device will complete the handshake and start the EOP for the handshake on or before bit time 41. For a low-speed device, the 36 bit times from start of EOP from root to start of EOP from the device are low-speed bit times, which convert 1

Microchip Ex. 1018, p. 334
Microchip v. Aptiv
IPR2024-00494

to 8 into full-speed bit times. Therefore, if the host completes the low-speed data packet by bit time 329, then the low-speed device can be predicted to complete the handshake before bit time 41.

Note: If the host cannot accept a full-speed EOP as a valid end of a low-speed packet, then the low-speed EOP will need to complete before bit time 41, which will add 13 full-speed bit times to the low-speed handshake time.

As the host approaches the end of the frame, it must ensure that it does not require a device to send a handshake if that handshake cannot be completed before bit time 41. The host expects to receive a handshake after any valid, non-isochronous data packet. Therefore, if the host is sending a non-isochronous data packet when it reaches bit time 76 (329 for low-speed), then the host should start an abnormal termination sequence to ensure that the device will not try to respond. This abnormal termination sequence consists of 7 consecutive (non-bitstuffed) bits of 1 followed by an EOP. The abnormal termination sequence is sent at the speed of the current packet. Note: The intent of this sequence is to force a bitstuffing violation (and possibly other errors) at the receiver.

If the host is preparing to send an IN token, it may not send the token if the predicted packet from the device would not complete by bit time 41. The maximum valid length of the response from the device is known by the host and should be used in the prediction calculation. For a full-speed packet, the maximum interval between the start of the IN token and the end of a data packet is:

token_length + (packet_length + header + CRC) * 7/6 + 18

Where *token_length* is 34 bit times, *packet_length* is the maximum number of data bits in the packet, *header* is eight bits of sync and eight bits of PID, and CRC is 16 bits. The 7/6 multiplier accounts for the absolute worst case bit-stuff on the packet, and the 18 extra bits allow for worst case turn-around delay. For a low-speed device, the same calculation applies, but the result must be multiplied by 8 to convert to full-speed bit times, and an additional 20 full-speed bit times must be added to account for the low-speed prefix. This gives the maximum number of bit times between the start of the IN token and the end of the data packet, so the token cannot be sent if this number of bit times does not exist before the earliest EOF1 point (bit time 41). (For example, take the results of the above calculation and add 41. If the number of bits left in the frame is less than this value, the token may not be sent.)

The host is allowed to use a more conservative algorithm than the one given above for deciding whether or not to start a transaction. The calculation might also include the time required for the host to send the handshake when one is required, as there is no benefit in starting a transfer if the handshake cannot be completed.

11.4 Internal Port

The internal port is the connection between the Hub Controller and the Hub Repeater. Besides conveying the serial data to/from the Hub Controller, the internal port is the source of certain resume signals. Figure 11-9 illustrates the internal port state machine; Table 11-4 defines the internal port signals and events.

Microchip Ex. 1018, p. 335
Microchip v. Aptiv
IPR2024-00494

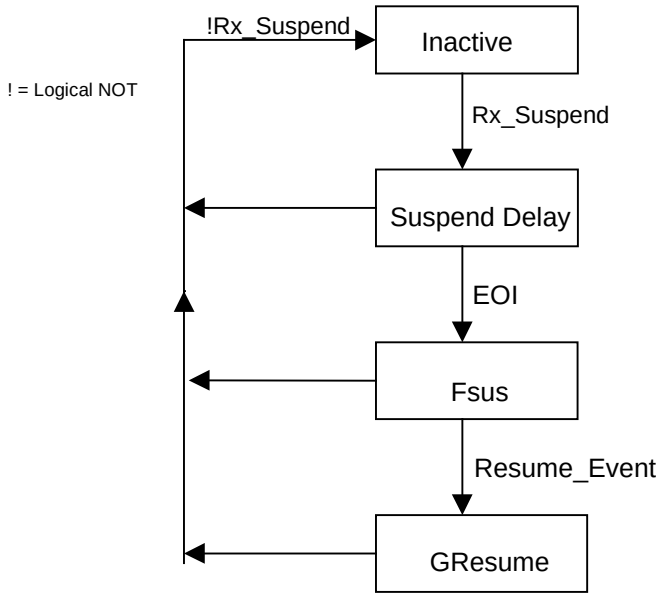


Figure 11-9. Internal Port State Machine

Table 11-4. Internal Port Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
EOI	Internal	End of timed interval
Rx_Suspend	Receiver	Receiver is in the Suspend state
Resume_Event	Hub Controller	A resume condition exists in the Hub Controller

11.4.1 Inactive

This state is entered whenever the Receiver is not in the Suspend state.

11.4.2 Suspend Delay

This state is entered from the Inactive state when the Receiver transitions to the Suspend state.

This is a timed state with a 2 ms interval.

11.4.3 Full Suspend (Fsus)

This state is entered when the Suspend Delay interval expires.

11.4.4 Generate Resume (GResume)

This state is entered from the Fsus state when a resume condition exists in the Hub Controller. A resume condition exists if the C_PORT_SUSPEND bit is set in any port, or if the hub is enabled as a wakeup source and any bit is set in a Port Change field or the Hub Change field (as described in Figures 11-22 and 11-20, respectively).

In this state, the internal port generates signaling to emulate an SOP_FD to the Hub Repeater.

Microchip Ex. 1018, p. 336
Microchip v. Aptiv
IPR2024-00494

11.5 Downstream Facing Ports

The following sections provide a functional description of a state machine that exhibits the correct behavior for a downstream facing port.

Figure 11-10 is an illustration of the downstream facing port state machine. The events and signals are defined in Table 11-5. Each of the states is described in Section 11.5.1. In the diagram below, some of the entry conditions into states are shown without origin. These conditions have multiple origin states and the individual transitions lines are not shown so that the diagram can be simplified. The description of the entered state indicates from which states the transition is applicable.

Note: For the root hub, the signals from the upstream facing port state machines are implementation dependent.

Microchip Ex. 1018, p. 337
Microchip v. Aptiv
IPR2024-00494

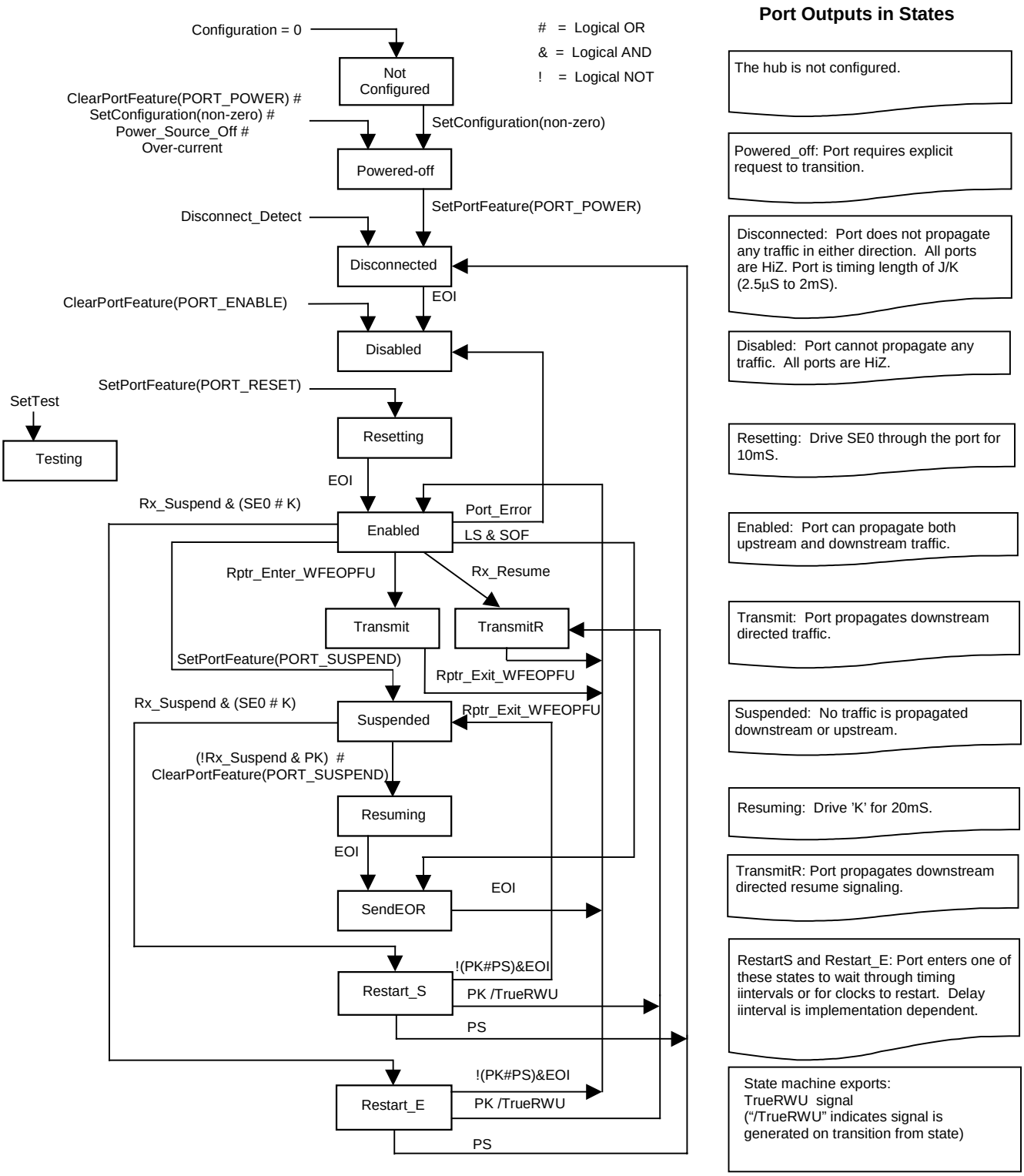


Figure 11-10. Downstream Facing Hub Port State Machine

Microchip Ex. 1018, p. 338
Microchip v. Aptiv
IPR2024-00494

Table 11-5. Downstream Facing Port Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
Power_source_off	Implementation-dependent	Power to the port not available due to over-current or termination of source power (e.g., external power removed)
Over-current	Hub Controller	Over-current condition exists on the hub or the port
EOI	Internal	End of a timed interval or sequence
SE0	Internal	SE0 received on port
Disconnect_Detect	Internal	Disconnect seen at port
LS	Hub Controller	Low-speed device attached to this port
SOF	Hub Controller	SOF token received
TrueRWU	Internal	K lasting for at least TDDIS (see Table 7-13)
PK	Internal	K lasting for at least TDDIS
PS	Internal	SE0 lasting for at least TDDIS
K	Internal	'K' received on port
Rx_Resume	Receiver	Upstream Receiver in Resume state
Rx_Suspend	Receiver	Upstream Receiver in Suspend state
Rptr_Exit_WFEOPFU	Hub Repeater	Hub Repeater exits the WFEOPFU state
Rptr_Enter_WFEOPFU	Hub Repeater	Hub Repeater enters the WFEOPFU state
Port_Error	Internal	Error condition detected (see Section 11.8.1)
SetTest	Hub Controller	Logical OR of SetPortFeature(Test_SE0_NAK), SetPortFeature(Test_J), SetPortFeature(Test_K), SetPortFeature(Test_PRBS), SetPortFeature(Test_Force_Enable)
Configuration = 0	Hub Controller	Hub controller's configuration value is zero

Microchip Ex. 1018, p. 339
Microchip v. Aptiv
IPR2024-00494

11.5.1 Downstream Facing Port State Descriptions

11.5.1.1 Not Configured

A port transitions to and remains in this state whenever the value of the hub configuration is zero. While the port is in this state, the hub will drive an SE0 on the port (this behavior is optional on root hubs). No other active signaling takes place on the port when it is in this state.

11.5.1.2 Powered-off

This state is supported for all hubs.

A port transitions to this state in any of the following situations:

- From any state except Not Configured when the hub receives a ClearPortFeature(PORT_POWER) request for this port
- From any state when the hub receives a SetConfiguration() request with a configuration value other than zero
- From any state except Not Configured when power is lost to the port or an over-current condition exists

A port will enter this state due to an over-current condition on another port if that over-current condition may have caused the power supplied to this port to drop below specified limits for port power (see Section 7.2.1.2.1 and Section 7.2.4.1).

If a hub was configured while the hub was self-powered, and then if external power is lost, the hub must place all ports in the Powered-off state. If the hub is configured while bus powered, then the hub need not change port status if the hub switched to externally applied power. However, if external power is subsequently lost, the hub must place ports in the Powered-off state.

In this state, the port's differential and single-ended transmitters and receivers are disabled.

Control of power to the port is covered in Section 11.11.

11.5.1.3 Disconnected

A port transitions to this state in any of the following situations:

- From the Powered-off state when the hub receives a SetPortFeature(PORT_POWER) request
- From any state except the Not Configured and Powered-off states when the port's disconnect timer times out
- From the Restart_S or Restart_E state at the end of the restart interval

In the Disconnected state, the port's differential transmitter and receiver are disabled and only connection detection is possible.

This is a timed state. While in this state, the timer is reset as long as the port's signal lines are in the SE0 or SE1 state. If another signaling state is detected, the timer starts. Unless the hub is suspended with clocks stopped, this timer's duration is 2.5 μs to 2 ms.

If the hub is suspended with its remote wakeup feature enabled, then on a transition to any state other than the SE0 state or SE1 state on a Disconnected port, the hub will start its clocks and time this event. The hub must be able to start its clocks and time this event within 12 ms of the transition. If a hub does not have its remote wakeup feature enabled, then transitions on a port that is in the Disconnected state are ignored until the hub is resumed.

Microchip Ex. 1018, p. 340
Microchip v. Aptiv
IPR2024-00494

11.5.1.4 Disabled

A port transitions to this state in any of the following situations:

- From the Disconnected state when the timer expires indicating a connection is detected on the port
- From any but the Powered-off, Disconnected, or Not Configured states on receipt of a ClearPortFeature(PORT_ENABLE) request
- From the Enabled state when an error condition is detected on the port

A port in the Disabled state will not propagate signaling in either the upstream or the downstream direction. While in this state, the duration of any SE0 received on the port is timed. If the port is using high-speed terminations when it enters this state, it switches to full-speed terminations. The port must not perform normal disconnect detection until at least 4 ms after entering this state.

11.5.1.5 Resetting

Unless it is in the Powered-off or Disconnected states, a port transitions to the Resetting state upon receipt of a SetPortFeature(PORT_RESET) request. The hub drives SE0 on the port during this timed interval. The duration of the Resetting state is nominally 10 ms to 20 ms (10 ms is preferred).

A hub in high-speed operation will use the high-speed terminations of the port when in this state.

11.5.1.6 Enabled

A port transitions to this state in any of the following situations:

- At the end of the Resetting state
- From the Transmit state or the TransmitR state when the Hub Repeater exits the WFEOPFU state
- From the Suspended state if the upstream Receiver is in the Suspend state when a 'K' is detected on the port
- At the end of the SendEOR state
- From the Restart_E state when a persistent K or persistent SE0 has not been seen within 900 µs of entering that state

While in this state, the output of the port's differential receiver is available to the Hub Repeater so that appropriate signaling transitions can establish upstream connectivity.

A port which is using high-speed terminations in this state switches to full-speed terminations on Rx_Suspend (i.e., when the hub is suspended). The port must not perform normal disconnect detection until at least 1 ms after Rx_Suspend becomes active.

11.5.1.7 Transmit

This state is entered from the Enabled state on the transition of the Hub Repeater to the WFEOPFU state. While in this state, the port will transmit the data that is received on the upstream facing port.

For a low-speed port, this state is entered from the Enabled state if a full-speed PRE PID is received on the upstream facing port. While in this state, the port will retransmit the data that is received on the upstream facing port (after proper inversion).

In high-speed, this state is used for testing for disconnect at the port. The disconnect detection circuit is enabled after 32 bits of the same signaling level ('J' or 'K') have been transmitted down the port.

Note: Because of the timing skew in the repeater path to the downstream facing ports, all downstream facing ports may not be enabled for disconnect detection at the same instant in time.

Microchip Ex. 1018, p. 341
Microchip v. Aptiv
IPR2024-00494

11.5.1.8 TransmitR

This state is entered in either of the following situations:

- From the Enabled state if the upstream Receiver is in the Resume state
- From the Restart_S or Restart_E state if a PK is detected on the port

When in this state, the port repeats the resume 'K' at the upstream facing port to the downstream facing port. Depending on the speed of the port, two behaviors are possible on the K->SE0 transition at the upstream facing port at the end of the resume.

- Upstream facing port high-speed and downstream facing port full-/low-speed: After the K->SE0 transition, the port drives SE0 for 16 to 18 full-speed bit times followed by driving J for at least one full-speed bit time. Note: The timer in the Resume state of the upstream port receiver state machine which generates EOITR can be used to time this requirement at the downstream facing port(s). The pullup resistor and the latency of the Transaction Translator(TT) results in this Idle state being maintained for at least one low-speed bit time ensuring that a device sees the same end of resume behavior below the TT as it would below a USB 1.x hub.
- Upstream facing port and downstream facing port are the same speed: port continues to repeat the signaling which follows the K->SE0 transition.

A port operating in high-speed reverts to its high-speed terminations within 18 full-speed bit times after the K->SE0 transition as described in Section 7.1.7.7.

11.5.1.9 Suspended

A port enters the Suspended state:

- From the Enabled state when it receives a SetPortFeature(PORT_SUSPEND) request
- From the Restart_S state when a persistent K or persistent SE0 has not been seen within 900 µs of entering that state

While a port is in the Suspended state, the port's differential transmitter is disabled. A high-speed port reverts from high-speed to full-speed terminations but its speed status continues to be high-speed. The port must not perform normal disconnect detection until at least 4 ms after entering this state.

An implementation must have a K/SE0 'noise' filter for a port that is in the suspended state. This filter can time the length of K/SE0 and, if the length of the K/SE0 is shorter than TDDIS, the port must remain in this state. If the hub is suspended with its clocks stopped, a transition to K/SE0 on a suspended port must cause the port to immediately transition to the Restart_S state.

11.5.1.10 Resuming

A port enters this state from the Suspended state in either of the following situations:

- If a 'K' is detected on the port and persists for at least 2.5 µs and the Receiver is not in the Suspended state. The transition from the Suspended state must happen within 900 µs of the J->K transition.
- When a ClearPortFeature(PORT_SUSPEND) request is received.

This is a timed state with a nominal duration of 20 ms (the interval may be longer under the conditions described in the note below). While in this state, the hub drives a 'K' on the port.

Note: A single timer is allowed to be used to time both the Resetting interval and the Resuming interval and that timer may be shared among multiple ports. When shared, the timer is reset when a port enters the Resuming state or the Resetting state. If shared, it may not be shared among more than ten ports as the cumulative delay could exceed the amount of time required to replace a device and a disconnect could be missed.

Microchip Ex. 1018, p. 342
Microchip v. Aptiv
IPR2024-00494

11.5.1.11 SendEOR

This state is entered from the Resuming state if the 20 ms timer expires. It is also entered from the Enabled state when an SOF (or other FS token) is received and a low-speed device is attached to this port.

This is a timed state which lasts for three low-speed bit times.

In this state, if the port is high-speed it will drive the bus to the Idle state for three low-speed bit times and then exit from this state to the Enabled state. It must also revert to its high-speed terminations within 18 full-speed bit times after the K->SE0 transition as described in Section 7.1.7.7.

If the port is full-speed or low-speed, the port must drive two low-speed bit times of SE0 followed by one low-speed bit time of Idle state and then exit from this state to the Enabled state.

Since the driven SE0 period should be of fixed length, the SendEOR timer, if shared, should not be reset. If the hub implementation shares the SendEOR timing circuits between ports, then for a port with a low-speed device attached, the Resuming state should not end until an SOF (or other FS token) has been received (see Section 11.8.4.1 for Keep-alive generation rules).

11.5.1.12 Restart_S

A port enters the Restart_S state from the Suspended state when an SE0 or 'K' is seen at the port and the Receiver is in the Suspended state.

In this state, the port continuously monitors the bus state. If the bus is in the 'K' state for at least TDDIS, the port sets the C_PORT_SUSPEND bit, exits to the TransmitR, and generates a signal to the repeater called 'TrueRWU'. If the bus is in the 'SE0' state for at least TDDIS, the port exits to the Disconnected state. Either of these transitions must happen within 900 µs after entering the Restart_S state; otherwise, the port must transition back to the Suspended state.

11.5.1.13 Restart_E

A port enters the Restart_E state from the Enabled state when an 'SE0' or 'K' is seen at the port and the Receiver is in the Suspended state.

In this state, the port continuously monitors the bus state. If the bus is in the 'K' state for at least TDDIS, the port exits to the TransmitR state and generates a signal to the repeater called 'TrueRWU'. If the bus is in the 'SE0' state for at least TDDIS, the port exits to the Disconnected state. Either of these transitions must happen within 900 µs after entering the Restart_E state; otherwise the port must transition back to the Enabled state.

11.5.1.14 Testing

A port transitions to this state from any state when the port sees SetTest.

While in this state, the port executes the host command as decoded by the hub controller. If the command was a SetPortFeature(PORT_TEST, Test_Force_Enable), the port supports packet connectivity in the downstream direction in a manner identical to that when the port is in the Enabled state.

11.5.2 Disconnect Detect Timer

11.5.2.1 High-speed Disconnect Detection

High-speed disconnect detection is described in Section 7.1.7.3.

Microchip Ex. 1018, p. 343
Microchip v. Aptiv
IPR2024-00494

11.5.2.2 Full-/low-speed Disconnect Detection

Each port is required to have a timer used for detecting disconnect when a full-/low-speed device is attached to the port. This timer is used to constantly monitor the port's single-ended receivers to detect a disconnect event. The reason for constant monitoring is that a noise event on the bus can cause the attached device to detect a reset condition on the bus after 2.5 µs of SE0 or SE1 on the bus. If the hub does not place the port in the disconnect state before the device resets, then the device can be at the Default Address state with the port enabled. This can cause systems errors that are very difficult to isolate and correct.

This timer must be reset whenever the D+ and D- lines on the port are not in the SE0 or SE1 state or when the port is not in the Enabled, Suspended, Disabled, Restart-E, or Restart_S states. This timer must be reset for 4ms upon entry to the Suspended and Disabled states. This timer times an interval TDDIS. The range of TDDIS is 2.0 µs to 2.5 as defined in Table 7-13. When this timer expires, it generates the Disconnect_Detect signal to the port state machine.

This timer can also be used for filtering the K/SE0 signal in the Suspended, Restart_E, or Restart_S states as described in Section 11.5.1.

11.5.3 Port Indicator

Each downstream facing port of a hub can support an optional status indicator. The presence of indicators for downstream facing ports is specified by bit 7 of the *wHubCharacteristics* field of the hub class descriptor. Each port's indicator must be located in a position that obviously associates the indicator with the port. The indicator provides two colors: green and amber. This can be implemented as physically one LED with two color capability or two separate LEDs. A combination of hardware and software control is used to inform the user of the current status of the port or the device attached to the port and to guide the user through problem resolution. Colors and blinking are used to provide information to the user.

An external hub must automatically control the color of the indicator as specified in Figure 11-11. Automatic port indicator setting support for root hubs may be implemented with either hardware or software. The port indicator color selector value is zero (indicating automatic control) when the hub transitions to the configured device state. When the hub is suspended or not configured, port indicators must be off.

Table 11-6 identifies the mapping of color to port state when the port indicators are automatically controlled.

Table 11-6. Automatic Port State to Port Indicator Color Mapping

Power Switching	Downstream Facing Hub Port State			
	Powered-off	**Disconnected, Disabled, Not Configured, Resetting, Testing**	**Enabled, Transmit, or TransmitR**	**Suspended, Resuming, SendEOR, Restart_E, or Restart_S**
With	Off or amber if due to an over-current condition	Off	Green	Off
Without	Off	Off or amber if due to an over-current condition	Green	Off

Microchip Ex. 1018, p. 344
Microchip v. Aptiv
IPR2024-00494

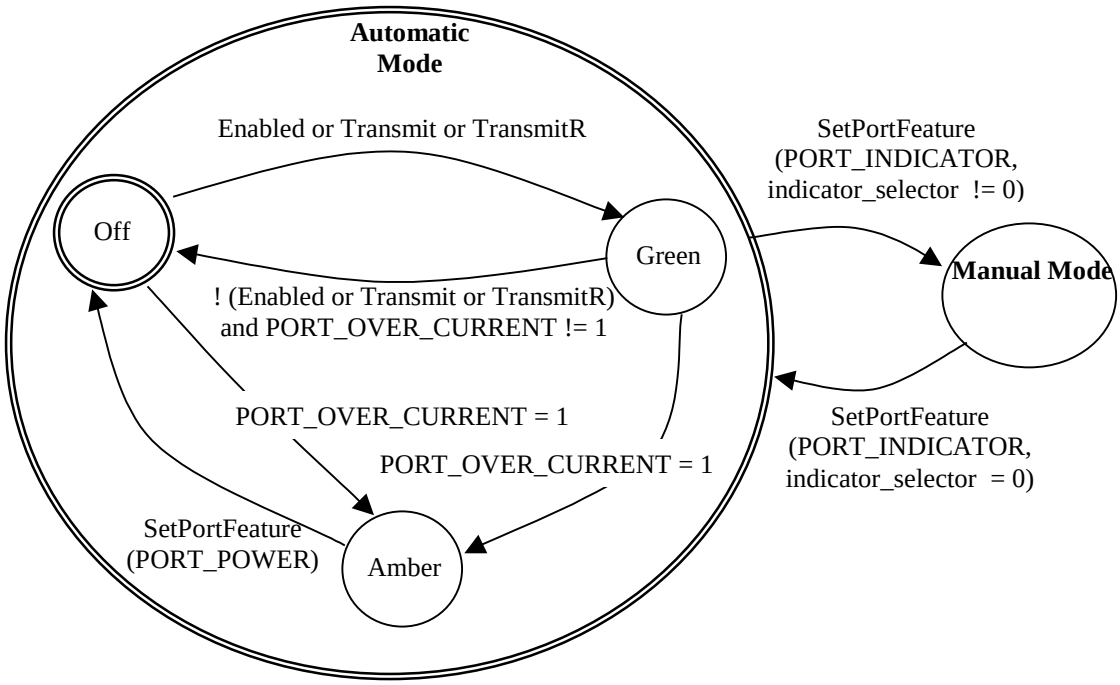


Figure 11-11. Port Indicator State Diagram

In **Manual Mode** the color of a port indicator (Amber, Green, or Off) is set by a system software USB Hub class request. In **Automatic Mode** the color of a port indicator is set by the port state information.

Table 11-7 defines port state as understood by the user.

Table 11-7. Port Indicator Color Definitions

Color	Definition
Off	Not operational
Amber	Error condition
Green	Fully operational
Blinking Off/Green	Software attention
Blinking Off/Amber	Hardware attention
Blinking Green/Amber	Reserved

Note that the indicators reflect the status of the port, not necessarily the device attached to it. Blinking of the indicator is used to draw the user's attention to the port, irrespective of its color.

Microchip Ex. 1018, p. 345
Microchip v. Aptiv
IPR2024-00494

Port indicators allow control by software. Host software forces the state of the indicator to draw attention to the port or to indicate the current state of the port.

See Section 11.24.2.7.1.10 for the specification of indicator requests.

11.5.3.1 Labeling

USB system software uses port numbers to reference an individual port with a ClearPortFeature or SetPortFeature request. If a vendor provides a labeling to identify individual downstream facing ports, then each port connector must be labeled with their respective port number.

11.6 Upstream Facing Port

The upstream facing port has four components: transmitter, transmitter state machine, receiver, and receiver state machine. The transmitter and its state machine are the Transmitter, while the receiver and its state machine are the Receiver. The Transmitter and Receiver operate in high-speed and full-speed depending on the current hub configuration.

11.6.1 Full-speed

Both the transmitter and receiver have differential and single-ended components. The differential transmitter and receiver can send/receive 'J' or 'K' to/from the bus while the single-ended components are used to send/receive SE0, suspend, and resume signaling. The single-ended components are also used to receive SE1. In this section, when it is necessary to differentiate the signals sent/received by the differential component of the transmitter/receiver from those of the single-ended components, DJ and DK will be used to denote the differential signal, while SJ, SK, SE0, and SE1 will be used for the single-ended signals.

When the Hub Repeater has connectivity in the upstream direction, the transmitter must not send or propagate SE1 signaling. Instead, the SE1 must be propagated as a DJ.

11.6.2 High-speed

Both the transmitter and receiver have differential components only. These signals are called HJ and HK. The HS_Idle state is the idle state of the bus in high-speed.

It is assumed that the differential transmitter and receiver are turned off during suspend to minimize power consumption. The single-ended components are left on at all times, as they will take minimal power.

11.6.3 Receiver

The receiver state machine is responsible for monitoring the signaling state of the upstream connection to detect long-term signaling events such as bus reset, resume, and suspend. This state machine details the operation of the device state diagram shown in Figure 9-1 in the Default, Address, Configured, and Suspended state. The Suspend, Resume, and ReceivingSE0 states are only used when the upstream facing port is operating in full-speed mode with full-speed terminations. The ReceivingIS, ReceivingHJ, and ReceivingHK states are only used when the upstream facing port is operating in high-speed mode with high-speed terminations; so these states are categorized as the HS (high-speed) states, and all other states are categorized as nonHS in the description below.

Microchip Ex. 1018, p. 346
Microchip v. Aptiv
IPR2024-00494

Figure 11-12 illustrates the state transition diagram.

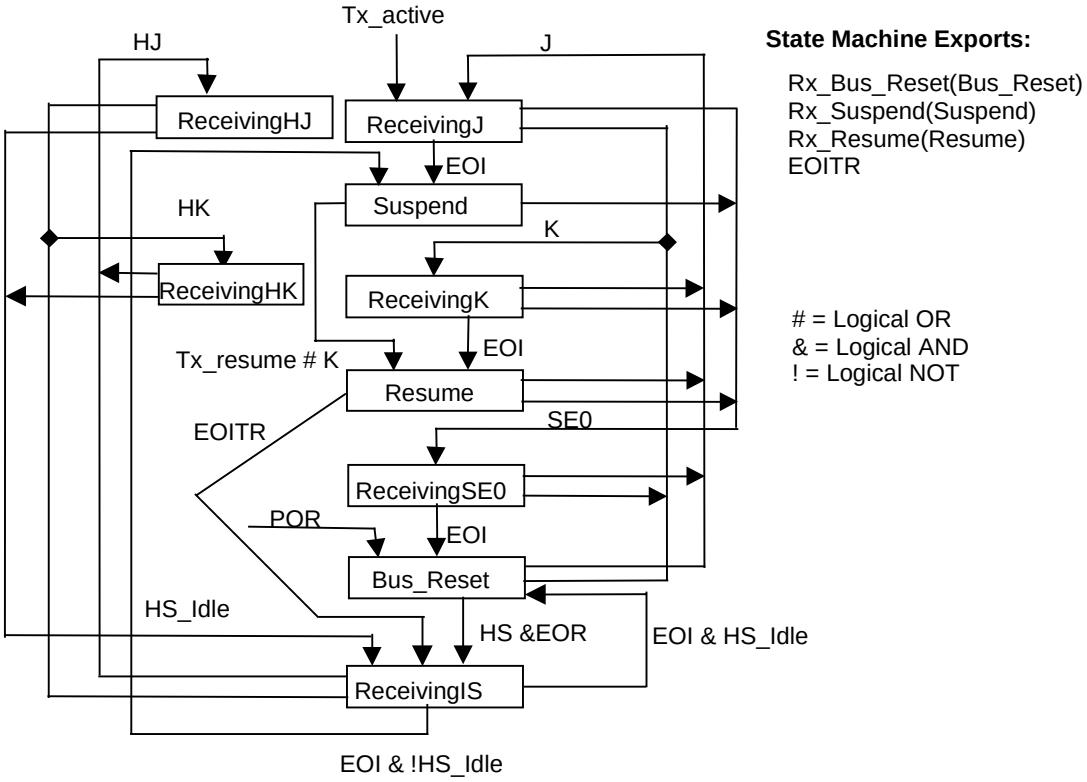


Figure 11-12. Upstream Facing Port Receiver State Machine

Table 11-8 defines the signals and events referenced in the figures.

Microchip Ex. 1018, p. 347
Microchip v. Aptiv
IPR2024-00494

Table 11-8. Upstream Facing Port Receiver Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
HS	Internal	Port is operating in high-speed
Tx_active	Transmitter	Transmitter in the Active state
J	Internal	Receiving a 'J' (IDLE) or an ‘SE1’ on the upstream facing port
HJ	Internal	Receiving an HJ on the upstream facing port
EOI	Internal	End of timed interval
EOITR	Internal	Generated 24 full-speed bit times after the K->SE0 transition at the end of resume
HK, K	Internal	Receiving an HK, 'K' on the upstream facing port
Tx_resume	Transmitter	Transmitter is in the Sresume state
HS_Idle	Internal	Receiving an Idle state on the high-speed upstream facing port
SE0	Internal	Receiving an SE0 on the full-speed upstream facing port
EOR	Internal	End of Reset signaling from upstream
POR	Implementation-dependent	Power_On_Reset

11.6.3.1 ReceivingIS

This state is entered

- From the ReceivingHJ or ReceivingHK state when a SE0 is seen at the port and the port is in high-speed operation
- From the Resume state when a EOITR is seen and the port is in high-speed operation
- From the Bus Reset state at the End of Reset signaling from upstream when the port is in high-speed operation

This is a timed state with an interval of 3 ms. The timer is reset each time this state is entered.

11.6.3.2 ReceivingHJ

This state is entered from an HS state when a HJ is seen on the bus.

11.6.3.3 ReceivingJ

This state is entered from a nonHS state except the Suspend state if the receiver detects an SJ (or Idle) or SE1 condition on the bus or while the Transmitter is in the Active state.

This is a timed state with an interval of 3 ms. The timer is reset each time this state is entered.

The timer only advances if the Transmitter is in the Inactive state.

Microchip Ex. 1018, p. 348
Microchip v. Aptiv
IPR2024-00494

11.6.3.4 Suspend

This state is entered when:

- The 3 ms timer expires in the ReceivingJ
- The 3 ms timer expires in the ReceivingIS state and the port has removed its high-speed terminations and connected its D+ pull-up resistor and the resulting bus state is not SE0.

When the Receiver enters this state, the Hub Controller starts a 2 ms timer. If that timer expires while the Receiver is still in this state, then the Hub Controller is suspended. When the Hub Controller is suspended, it may generate resume signaling.

11.6.3.5 ReceivingHK

This state is entered from an HS state when a HK is seen on the bus.

11.6.3.6 ReceivingK

This state is entered from any nonHS state except the Resume state when the receiver detects an SK condition on the bus and the Hub Repeater is in the WFSOP or WFSOPFU state.

This is a timed state with a duration of 2.5 μs to 100 μs. The timer is reset each time this state starts.

11.6.3.7 Resume

This state is entered:

- From the ReceivingK state when the timer expires
- From the Suspend state while the Transmitter is in the Sresume state or if there is a transition to the K state on the upstream facing port

If the hub enters this state when its timing reference is not available, the hub may remain in this state until the hub's timing reference becomes stable (timing references must stabilize in less than 10 ms). If this state is being held pending stabilization of the hub's clock, the Receiver must provide a K to the repeater for propagation to the downstream facing ports. When clocks are stable, the Receiver must repeat the incoming signals.

Note: Hub timing references will be stable in less than 10 ms since reset requirements already specify that they be stable in less than 10 ms and a hub must support reset from suspend.

11.6.3.8 ReceivingSE0

This state is entered from any nonHS state except Bus_Reset when the receiver detects an SE0 condition and the Hub Repeater is in the WFSOP or WFSOPFU state.

This is a timed state. The minimum interval for this state is 2.5 μs. The maximum depends on the hub but this interval must timeout early enough such that if the width of the SE0 on the upstream facing port is only 10 ms, the Receiver will enter the Bus_Reset state with sufficient time remaining in the 10 ms interval for the hub to complete its reset processing. Furthermore, if the hub is suspended when the Receiver enters this state, the hub must be able to start its clocks, time this interval, and complete its reset (chirp) protocol and processing in the Bus_Reset state within 10 ms. It is preferred that this interval be as long as possible given the constraints listed here. This will provide for the maximum immunity to noise on the upstream facing port and reduce the probability that the device will reset in the presence of noise before the upstream hub disables the port.

The timer is reset each time this state starts.

Microchip Ex. 1018, p. 349
Microchip v. Aptiv
IPR2024-00494

11.6.3.9 Bus_Reset

This state is entered:

- From the ReceivingSE0 state when the timer expires. As long as the port continues to receive SE0, the Receiver will remain in this state.
- This state is also entered while power-on-reset (POR) is being generated by the hub's local circuitry. The state machine cannot exit this state while POR is active.
- The 3 ms timer expires in the ReceivingIS state and the port has removed its high-speed terminations and connected its D+ pull-up resistor and the resulting bus state is still SE0.

In this state, a high-speed capable port will implement the chirp signaling, handshake, and timing protocol as described in Section 7.1.7.5.

11.6.4 Transmitter

This state machine is used to monitor the upstream facing port while the Hub Repeater has connectivity in the upstream direction. The purpose of this monitoring activity is to prevent propagation of erroneous indications in the upstream direction. In particular, this machine prevents babble and disconnect events on the downstream facing ports of this hub from propagating and causing this hub to be disabled or disconnected by the hub to which it is attached. Figure 11-13 is the transmitter state transition diagram. Table 11-9 defines the signals and events referenced in Figure 11-13.

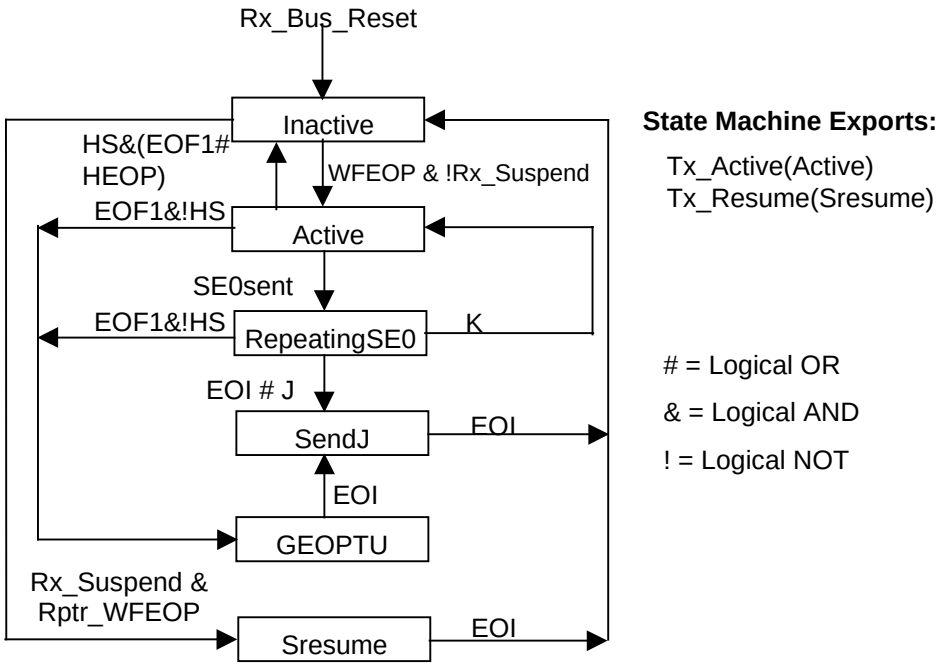


Figure 11-13. Upstream Facing Port Transmitter State Machine

Microchip Ex. 1018, p. 350
Microchip v. Aptiv
IPR2024-00494

Table 11-9. Upstream Facing Port Transmit Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
Rx_Bus_Reset	Receiver	Receiver is in the Bus_Reset state
EOF1	(micro)frame Timer	Hub (micro)frame time has reached the EOF1 point or is between EOF1 and the end of the (micro)frame
J	Internal	Transmitter transitions to sending a 'J' and transmits a 'J'
Rptr_WFEOP	Hub Repeater	Hub Repeater is in the WFOEP state
K	Internal	Transmitter transmits a 'K'
SE0sent	Internal	At least one bit time of SE0 has been sent through the transmitter
Rx_Suspend	Receiver	Receiver is in Suspend state
HEOP	Repeater	Completion of packet transmission in upstream direction
HS	Internal	Upstream facing port is operating as high-speed port
EOI	Internal	End of timed interval

11.6.4.1 Inactive

This state is entered at the end of the SendJ state or while the Receiver is in the Bus_Reset state. This state is also entered at the end of the Sresume state. While the transmitter is in this state, both the differential and single-ended transmit circuits are disabled and placed in their high-impedance state.

When port is operating as a high-speed port, this state is entered from the Active state at EOF1 or after an HEOP from downstream.

11.6.4.2 Active

This state is entered from the Inactive state when the Hub Repeater transitions to the WFEOP state. This state is entered from the RepeatingSE0 state if the first transition after the SE0 is not to the J state. In this state, the data from a downstream facing port is repeated and transmitted on the upstream facing port.

11.6.4.3 RepeatingSE0

The port enters this state from the Active state when one bit time of SE0 has been sent on the upstream facing port. While in this state, the transmitter is still active and downstream signaling is repeated on the port. This is a timed state with a duration of 23 full-speed bit times.

11.6.4.4 SendJ

The port enters this state from the RepeatingSE0 state if either the bit timer reaches 23 or the repeated signaling changes from SE0 to 'J' or 'SE1'. This state is also entered at the end of the GEOPTU state. This state lasts for one full-speed bit time. During this state, the hub drives an SJ on the port.

Microchip Ex. 1018, p. 351
Microchip v. Aptiv
IPR2024-00494

11.6.4.5 Generate End of Packet Towards Upstream Port (GEOPTU)

The port enters this state from the Active or RepeatingSEO state if the frame timer reaches the EOF1 point.

In this state, the port transmits SE0 for two full-speed bit times.

11.6.4.6 Send Resume (Sresume)

The port enters this state from the Inactive state if the Receiver is in the Suspend state and the Hub Repeater transitions to the WFEOP state. This indicates that a downstream device (or the port to the Hub Controller) has generated resume signaling causing upstream connectivity to be established.

On entering this state, the hub will restart clocks if they had been turned off during the Suspend state. While in this state, the Transmitter will drive a 'K' on the upstream facing port. While the Transmitter is in this state, the Receiver is held in the Resume state. While the Receiver is in the Resume state, all downstream facing ports that are in the Enabled state are placed in the TransmitR state and the resume on this port is transmitted to those downstream facing ports.

The port stays in this state for at least 1 ms but for no more than 15 ms.

11.7 Hub Repeater

The Hub Repeater provides the following functions:

- Sets up and tears down connectivity on packet boundaries
- Ensures orderly entry into and out of the Suspend state, including proper handling of remote wakeups

11.7.1 High-speed Packet Connectivity

High-speed packet repeaters must reclock the packets in both directions. Reclocking means that the repeater extracts the data from the received stream and retransmits the stream using its own local clock. This is necessary in order to keep the jitter seen at a receiver within acceptable limits (see Chapter 7 for definition and limits on jitter).

Reclocking creates several requirements which can be best understood with the example repeater signal path shown in Figure 11-14.

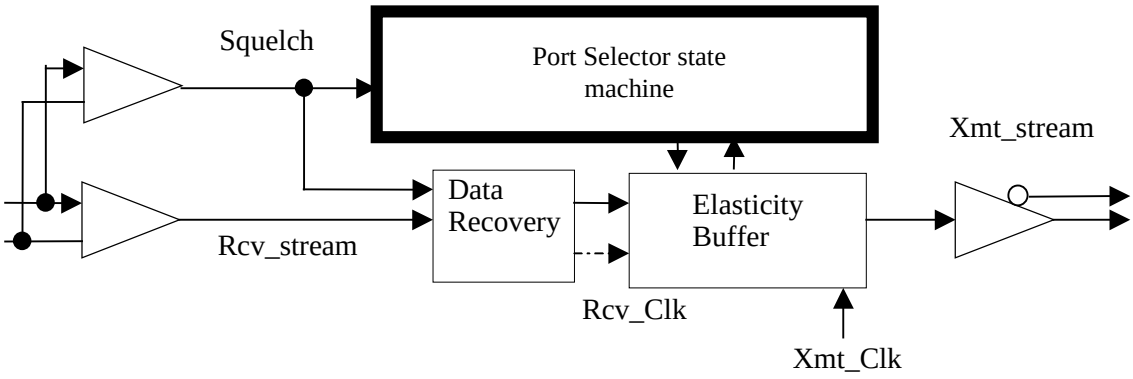


Figure 11-14. Example Hub Repeater Organization

Microchip Ex. 1018, p. 352
Microchip v. Aptiv
IPR2024-00494

11.7.1.1 Squelch Circuit

Because of squelch detection, the initial bits of the SYNC field may not be seen in the rest of the repeater. At most, 4 bits of the SYNC field may be sacrificed in the entire repeater path.

The squelch circuit may take at most 4 bit times to disable the repeater after the bus returns to the Idle state. This results in bits being added after the end of the packet. This is also known as EOP dribble and up to 4 random bits may get added after the packet by the entire repeater path.

11.7.1.2 Data Recovery Unit

The data recovery unit extracts the receive clock and receive data from this stream. Note that this is a conceptual model only; actual implementations (e.g., DLL) may achieve the reclocking by the local clock without separation of the receive clock and data.

11.7.1.3 Elasticity Buffer

The half-depth of the elasticity buffer in the repeater must be at least 12 bits.

The total latency of a packet through a repeater must be less than 36 bit times. This includes the latency through the elasticity buffer.

The elasticity buffer is used to handle the difference in frequency between the receive clock and the local clock and works as follows. The elasticity buffer is primed (filled with at least 12 bits) by the receive clock before the data is clocked out of it by the transmit clock. If the transmit clock is faster than the receive clock, the buffer will get emptied more quickly than it gets filled. If the transmit clock is slower, the buffer will get emptied slower than it gets filled. If the half-depth of the buffer is chosen to be equal to the maximum difference in clock rate over the length of a packet, bits will not be lost or added to the packet. The half-depth is calculated as follows.

The clock tolerance allowed is 500 ppm. This takes into account the effect of voltage, temperature, aging, etc. So the received clock and the local clock could be different by 1000 ppm. The longest packet has a data payload of 1 Kbytes. The maximum length of a packet is computed by adding the length of all the fields and assuming maximum bit-stuffing. This maximum length is 9644 bits (9624 bits of packet + 20 bits of EOP dribble). This means that when the repeater is clocking out a packet with its local clock, it could get ahead of or fall behind the receive clock by 9.644 bits (1000 ppm*9644). This calculation yields 10 bits. The half-depth of the elasticity buffer in the repeater must be at least 12 bits to provide system timing margin.

11.7.1.4 High-Speed Port Selector State Machine

This state machine is used to establish connectivity on a valid packet and to keep the repeater from establishing connectivity from a port which is seeing noise. This state machine must implement the behavior shown in Figure 11-15. (Note: This state machine may be implemented on a per-port or per-hub basis.)

Microchip Ex. 1018, p. 353
Microchip v. Aptiv
IPR2024-00494

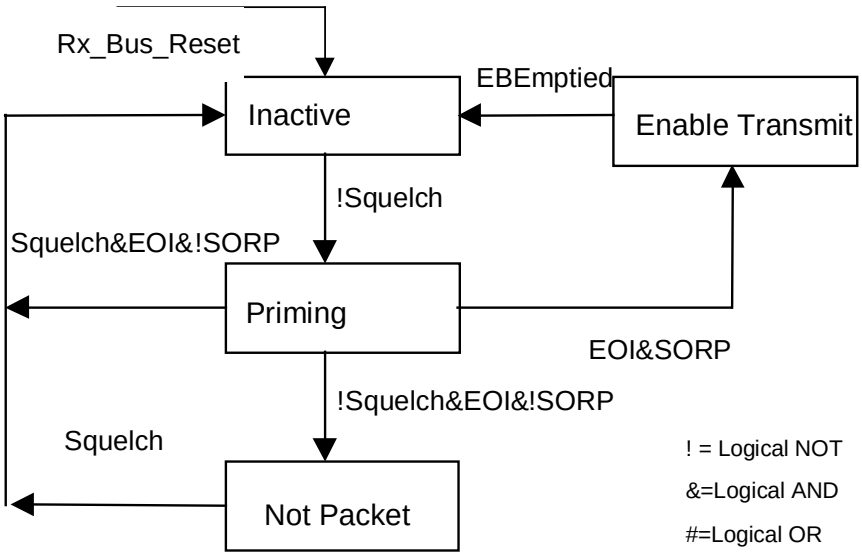


Figure 11-15. High-speed Port Selector State Machine

Table 11-10. High-speed Port Selector Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
Rx_Bus_Reset	Internal	Receiver is in the Bus_reset state.
EBEmptied	Internal	All bits accumulated in the elasticity buffer have been transmitted.
EOI	Internal	End of interval of time needed for priming elasticity buffer
Squelch	Internal	Bus is in squelch state
SORP	Internal	Start Of Repeating Pattern; a 'JKJK' or 'KJKJ' pattern has been seen in data in elasticity buffer.

11.7.1.4.1 Inactive

This state is entered

- From the Enable Transmit state when all the bits accumulated in the elasticity buffer have been transmitted
- From the Priming state if squelch is seen and the elasticity buffer is primed without a SORP being seen
- From the Not Packet state when the squelch circuit indicates a squelch state on the port
- From on any state on Rx_Bus_Reset

11.7.1.4.2 Priming

This state is entered from the Inactive state when the squelch circuit indicates that valid signal levels have been observed at the port. This is a timed state and the priming interval is the time needed for the implementation to fill the elasticity buffer with at least 12 bits.

Microchip Ex. 1018, p. 354
Microchip v. Aptiv
IPR2024-00494

11.7.1.4.3 Enable Transmit

This state is entered from the Priming state when the Elasticity buffer priming interval has elapsed and the bits in the elasticity buffer include the SORP pattern.

In this state, the state machine generates a signal “start of high-speed packet” (SOHP) to the repeater state machine which allows the repeater to establish connectivity from this port to the upstream facing port (or downstream facing ports).

11.7.1.4.4 Not Packet

This state is entered from the Priming state when the Elasticity buffer priming interval has elapsed, and the bits in the elasticity buffer do not include the SORP pattern, and the squelch signal is not active.

11.7.2 Hub Repeater State Machine

The Hub repeater state machine in Figure 11-16 shows the states and transitions needed to implement the Hub Repeater. Table 11-11 defines the Hub Repeater signals and events. The following sections describe the states and the transitions.

11.7.2.1 High-speed Repeater Operation

Connectivity is setup on SOHP and torn down on HEOP. (HEOP is either the EBemptied signal from the port selector state machine ‘OR’ the EOI signal which causes the transition out of the SendEOR state in downstream facing port state machine.) Several of the state transitions below will occur when the HEOP is seen. When such a transition is indicated, the transition does not occur until after the hub has repeated the last bit in the elasticity buffer. Some of the transitions are triggered by an SOHP. Transitions of this type occur as soon as the hub detects the SOHP from the port selector state machine ensuring that a valid packet start has been seen.

11.7.2.2 Full-/low-speed Repeater Operation

Connectivity is setup on SOP and torn down on EOP. Several of the state transitions below will occur when the EOP is seen. When such a transition is indicated, the transition does not occur until after the hub has repeated the SE0-to-'J' transition and has driven 'J' for at least one bit time (bit time is determined by the speed of the port.) Some of the transitions are triggered by an SOP. Transitions of this type occur as soon as the hub detects the 'J'-to-'K' transition, ensuring that the initial edge of the SYNC field is preserved.

Microchip Ex. 1018, p. 355
Microchip v. Aptiv
IPR2024-00494

11.7.2.3 Repeater State Machine

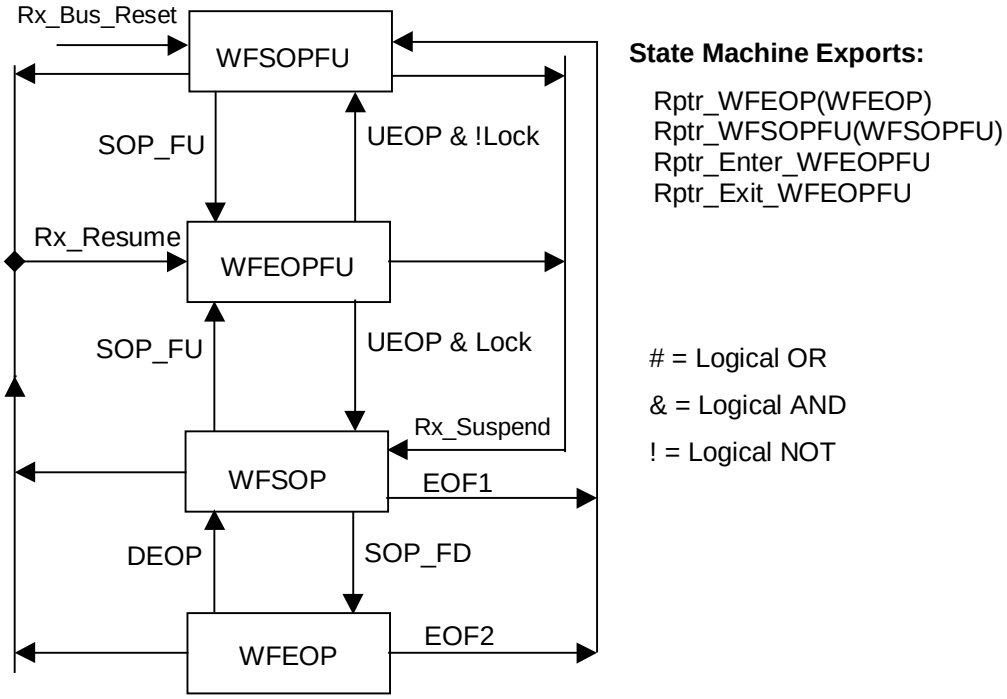


Figure 11-16. Hub Repeater State Machine

Microchip Ex. 1018, p. 356
Microchip v. Aptiv
IPR2024-00494

Table 11-11. Hub Repeater Signal/Event Definitions

Signal/Event Name	Event/Signal Source	Description
Rx_Bus_Reset	Receiver	Receiver is in the Bus_Reset state
HEOP	Internal (Port selector, Downstream port, Upstream port receiver)	Three sources of HEOP: EBEmptied signal from port selector state machine OR transition at EOI from SendEOR state in downstream facing port state machine OR EOITR from upstream facing port receiver state machine
UEOP	Internal	(HEOP)EOP received from the upstream facing port
DEOP	Internal	Generated when the Transmitter enters the (Inactive) SendJ state
EOF1	(Micro)frame Timer	(micro)frame timer is at the EOF1 point or between EOF1 and End-of-(micro)frame
EOF2	(Micro)frame Timer	(micro)frame timer is at the EOF2 point or between EOF2 and End-of-(micro)frame
Lock	(Micro)frame Timer	(micro)frame timer is locked
Rx_Suspend	Receiver	Receiver is in the Suspend state
Rx_Resume	Receiver	Receiver is in the Resume state
SOP_FD	Internal	(SOHP)SOP received from downstream facing port or Hub Controller. Generated (after SOHP identified) on the transition from the Idle to K state on a port.
SOP_FU	Internal	(SOHP)SOP received from upstream facing port. Generated (after SOHP identified) on the transition from the Idle to K state on the upstream facing port.

11.7.3 Wait for Start of Packet from Upstream Port (WFSOPFU)

This state is entered in either of the following situations:

- From any other state when the upstream Receiver is in the Bus_Reset state
- From the WFSOP state if the (micro)frame timer is at or has passed the EOF1 point
- From the WFEOP state at the EOF2 point
- From the WFEOPFU if the (micro)frame timer is not synchronized (locked) when an (HEOP)EOP is received on the upstream facing port

In this state, the hub is waiting for an (SOHP)SOP on the upstream facing port, and transitions on downstream facing ports are ignored by the Hub Repeater. While the Hub Repeater is in this state, connectivity is not established.

Microchip Ex. 1018, p. 357
Microchip v. Aptiv
IPR2024-00494

This state is used during the End-of-(micro)frame (past the EOF1 point) to ensure that the hub will be able to receive the SOF when it is sent by the host.

11.7.4 Wait for End of Packet from Upstream Port (WFEOPFU)

The hub enters this state if the hub is in the WFSOP or WFSOPFU state and an (SOHP)SOP is detected on the upstream facing port. The hub also enters this state from the WFSOP, WFSOPFU, or WFEOP states when the Receiver enters the Resume state.

While in this state, connectivity is established from the upstream facing port to all enabled downstream facing ports. Downstream facing ports that are in the Enabled state are placed in the Transmit state on the transition to this state.

11.7.5 Wait for Start of Packet (WFSOP)

This state is entered in any of the following situations:

- From the WFEOP state when an (HEOP)EOP is detected from the downstream facing port
- From the WFEOPFU state if the (micro)frame timer is synchronized (locked) when an (HEOP)EOP is received from upstream
- From the WFSOPFU or WFEOPFU states when the upstream Receiver transitions to the Suspend state

A hub in this state is waiting for an (SOHP)SOP on the upstream facing port or any downstream facing port that is in the Enabled state. While the Hub Repeater is in this state, connectivity is not established.

11.7.6 Wait for End of Packet (WFEOP)

This state is entered from the WFSOP state when an (SOHP)SOP is received from a downstream facing port in the Enabled state.

In this state, the hub has connectivity established in the upstream direction and the signaling received on an enabled downstream facing port is repeated and driven on the upstream facing port. The upstream Transmitter is placed in the Active state on the transition to this state.

If the Hub Repeater is in this state when the EOF2 point is reached, the downstream facing port for which connectivity is established is disabled as a babble port.

Note: The full-speed Transmitter will send an EOP at EOF1, but the Repeater stays in this state until the device sends an (HEOP)EOP or the EOF2 point is reached.

11.8 Bus State Evaluation

A hub is required to evaluate the state of the connection on a port in order to make appropriate port state transitions. This section describes the appropriate times and means for several of these evaluations.

11.8.1 Port Error

A Port Error can occur on a downstream facing port that is in the Enabled state. A Port Error condition exists when:

- The hub is in the WFEOP state with connectivity established upstream from the port when the (micro)frame timer reaches the EOF2 point.
- At the EOF2 point, the Hub Repeater is in the WFSOPFU state, and there is other than Idle state on the port.

Microchip Ex. 1018, p. 358
Microchip v. Aptiv
IPR2024-00494

If upstream-directed connectivity is established when the (micro)frame timer reaches the EOF1 point, the upstream Transmitter will (return to Inactive state) generate a full-speed EOP to prevent the hub from being disabled by the upstream hub. The connected port is then disabled if it has not ended the packet and returned to the Idle state before the (micro)frame timer reaches the EOF2 point.

11.8.2 Speed Detection

At the end of reset, the bus is in the Idle state for the speed recorded in the port status register. Speed detection is described in Section 7.1.7.5.

If the device connected at the downstream facing port is high-speed, the repeater (rather than the Transaction Translator) is used to signal between this port and the upstream facing port.

Due to connect and start-up transients, the hub may not be able to reliably determine the speed of the device until the transients have ended. The USB System Software is required to "debounce" the connection and provide a delay between the time a connection is detected and the device is used (see Section 7.1.7.3). At the end of the debounce interval, the device is expected to have placed its upstream facing port in the Idle state and be able to react to reset signaling. The USB System Software must send a SetPortFeature(PORT_RESET) request to the port to enable the port and make the attached device ready for use.

The downstream facing port monitors the state of the D+ and D- lines to determine if the connected device is low-speed. If so, the PORT_LOW_SPEED status bit is set to one to indicate a low-speed device. If not, the PORT_LOW_SPEED status bit is set to zero to indicate a full-/high-speed device. Upon exit from the reset process, the hub must set the PORT_HIGH_SPEED status bit according to the detected speed. The downstream facing port performs the required reset processing as defined in Section 7.1.7.5. At the end of the Resetting state, the hub will return the bus to the Idle state that is appropriate for the speed of the attached device and transition to the Enabled state.

11.8.3 Collision

If the Hub Repeater is in the WFEOP state and an (SOHP)SOP is detected on another enabled port, a Collision condition exists. There are two allowed behaviors for the hub in this instance. In either case, connectivity teardown at EOF1 and babble detection at EOF2 is required.

The first, and preferred, behavior is to ‘garble’ the message so that the host can detect the problem. The hub garbles the message by transmitting a (‘J’ or) 'K' on the upstream facing port. This (‘J’ or) 'K' should persist until packet traffic from all downstream facing ports ends. The hub should use the last (‘J’ or ‘K’) EOP to terminate the garbled packet. Babble detection is enabled during this garbled message.

A second behavior is to block the second packet and, when the first message ends, return the hub to the WFSOPFU or WFSOP state as appropriate. If the second stream is still active, the hub may reestablish connectivity upstream. This method is not preferred, as it does not convey the problem to the host. Additionally, if the second stream causes the hub to reestablish upstream connectivity as the host is trying to establish downstream connectivity, additional packets can be lost and the host cannot properly associate the problem.

Note: In high-speed repeaters, use of the SOHP to detect collisions would need replication of the datapath shown in Figure 11-14 at every port. The unsquelch signal at a port can be used instead of the SOHP to detect collisions; in this case, the second behavior (blocking) described above must be used.

11.8.4 Low-speed Port Behavior

When a hub is configured for full-/low-speed operation, low-speed data is sent or received through the hub’s upstream facing port at full-speed signaling even though the bit times are low-speed.

Full-speed signaling must not be transmitted to low-speed ports.

Microchip Ex. 1018, p. 359
Microchip v. Aptiv
IPR2024-00494

If a port is detected to be attached to a low-speed device, the hub port's output buffers are configured to operate at the slow slew rate (75-300 ns), and the port will not propagate downstream-directed packets unless they are prefaced with a PRE PID. When a PRE PID is received, the 'J' state must be driven on enabled low-speed ports within four bit times of receiving the last bit of the PRE PID.

Low-speed data follows the PID and is propagated to both low- and full-speed devices. Hubs continue to propagate downstream signaling to all enabled ports until a downstream EOP is detected, at which time all output drivers are turned off.

Full-speed devices will not misinterpret low-speed traffic because no low-speed data pattern can generate a valid full-speed PID.

When a low-speed device transmits, it does not preface its data packet with a PRE PID. Hubs will propagate upstream-directed packets of full-/low-speed using full-speed signaling polarity and edge rates.

For both upstream and downstream low-speed data, the hub is responsible for inverting the polarity of the data before transmitting to/from a low-speed port.

Although a low-speed device will send a low-speed EOP to properly terminate a packet, a hub may truncate a low-speed packet at the EOF1 point with a full-speed EOP. Thus, hubs must always be able to tear down connectivity in response to a full-speed EOP regardless of the data rate of the packet.

Because of the slow transitions on low-speed ports, when the D+ and D- signal lines are switching between the 'J' and 'K', they may both be below 2.0 V for a period of time that is longer than a full-speed bit time. A hub must ensure that these slow transitions do not result in termination of connectivity and must not result in an SE0 being sent upstream.

11.8.4.1 Low-speed Keep-alive

All hub ports to which low-speed devices are connected must generate a low-speed keep-alive strobe, generated at the beginning of the frame, which consists of a valid low-speed EOP (described in Section 7.1.13.2). The strobe must be generated at least once in each frame in which an SOF is received. This strobe is used to prevent low-speed devices from suspending if there is no other low-speed traffic on the bus. The hub can generate the keep-alive on any valid full-speed token packet. The following rules for generation of a low-speed keep-alive must be adhered to:

- A keep-alive must minimally be derived from each SOF. It is recommended that a keep-alive be generated on any valid full-speed token.
- The keep-alive must start by the eighth bit after the PID of the full-speed token.

11.9 Suspend and Resume

Hubs must support suspend and resume both as a USB device and in terms of propagating suspend and resume signaling. Hubs support both global and selective suspend and resume. Global and selective suspend are defined in Section 7.1.7.6. Global suspend/resume refers to the entire bus being suspended or resumed without affecting any hub's downstream facing port states; selective suspend/resume refers to a downstream facing port of a hub being suspended or resumed without affecting the hub state. Global suspend/resume is implemented through the root port(s) at the host. Selective suspend/resume is implemented via requests to a hub. Device-initiated resume is called remote-wakeup (see Section 7.1.7.7).

If the hub upstream facing port is in (high-speed) full-speed, the required behavior is the same as that for a function with upstream facing port in (high-speed) full-speed and is described in Chapter 7.

When a downstream facing port operating at high-speed goes into the Suspended state, it switches to full-speed terminations but continues to have high-speed port status. In response to a remote wakeup or selective resume, this port will drive full-speed 'K' throughout its Resuming state. The requirements and timings are the same as for full-speed ports and described below. At the end of this signaling, the bus will

Microchip Ex. 1018, p. 360
Microchip v. Aptiv
IPR2024-00494

be returned to the high-speed Idle state (using the SendEOR state). After this, the port will return to the Enabled state. The high-speed status of the port is maintained throughout the suspend-resume cycle.

Figure 11-17 and Figure 11-18 show the timing relationships for an example remote-wakeup sequence. This example illustrates a device initiating resume signaling through a suspended hub ('B') to an awake hub ('A'). Hub 'A' in this example times and completes the resume sequence and is the "Controlling Hub". The timings and events are defined in Section 7.1.7.7.

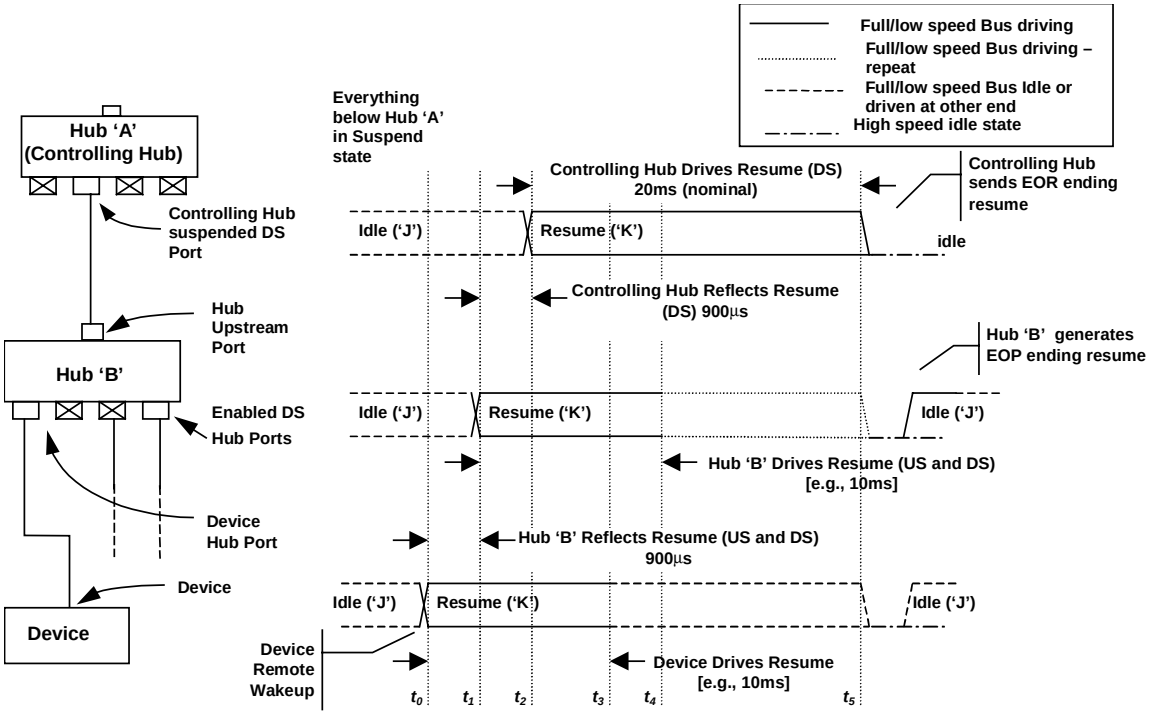


Figure 11-17. Example Remote-wakeup Resume Signaling With Full-/low-speed Device

Microchip Ex. 1018, p. 361
Microchip v. Aptiv
IPR2024-00494

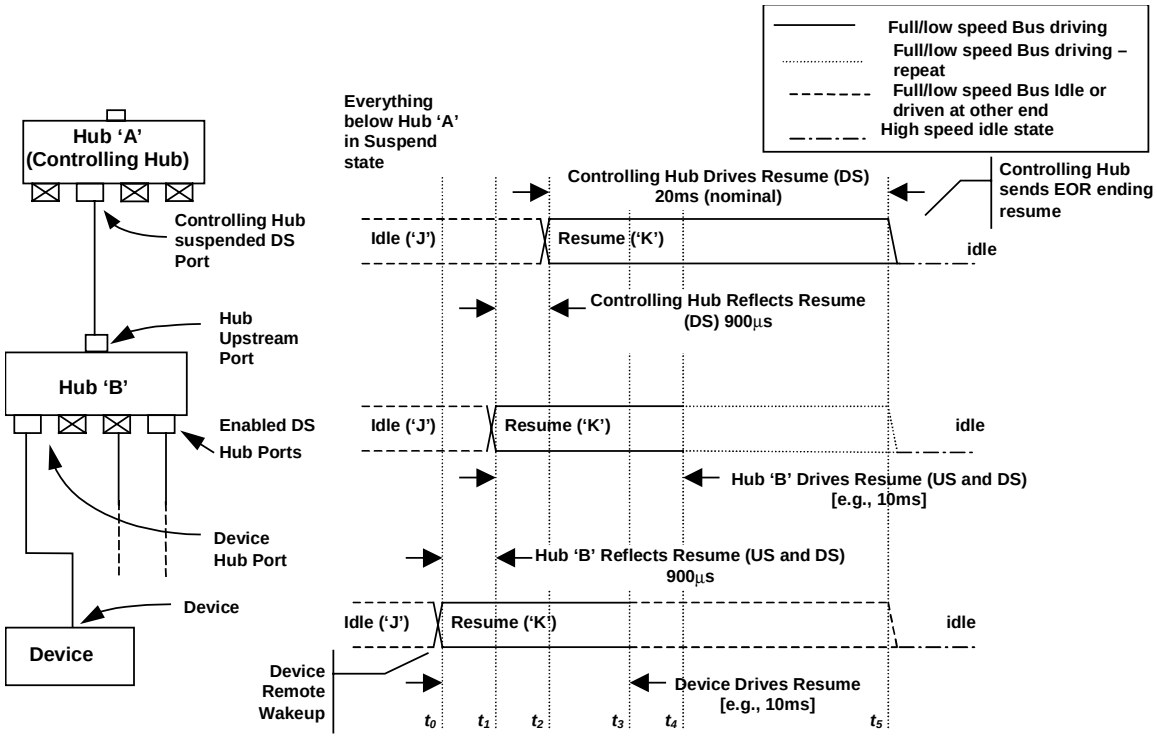


Figure 11-18. Example Remote-wakeup Resume Signaling With High-speed Device

Here is an explanation of what happens at each t_n:

***t*$_0$** Suspended device initiates remote-wakeup by driving a 'K' on the data lines.

***t*$_1$** Suspended hub 'B' detects the 'K' on its downstream facing port and wakes up enough within 900 µs to filter and then reflect the resume upstream and down through all enabled ports.

***t*$_2$** Hub 'A' is not suspended (implication is that the port at which 'B' is attached is selectively suspended), detects the 'K' on the selectively suspended port where 'B' is attached, and filters and then reflects the resume signal back to 'B' within 900 µs.

***t*$_3$** Device ceases driving 'K' upstream.

***t*$_4$** Hub 'B' ceases driving 'K' upstream and down all enabled ports and begins repeating upstream signaling to all enabled downstream facing ports.

***t*$_5$** Hub 'A' completes resume sequence, after appropriate timing interval, by driving a speed-appropriate end of resume downstream. (End of resume will be an Idle state for a high-speed device or a low-speed EOP for a full-/low-speed device.)

The hub reflection time is much smaller than the minimum duration a USB device will drive resume upstream. This relationship guarantees that resume will be propagated upstream and downstream without any gaps.

11.10 Hub Reset Behavior

Reset signaling to a hub is defined only in the downstream direction, which is at the hub's upstream facing port. Reset signaling required of the hub is described in Section 7.1.7.5.

A suspended hub must interpret the start of reset as a wakeup event; it must be awake and have completed its reset sequence by the end of reset signaling.

Microchip Ex. 1018, p. 362
Microchip v. Aptiv
IPR2024-00494

After completion of the reset sequence, a hub is in the following state:

- Hub Controller default address is 0.
- Hub status change bits are set to zero.
- Hub Repeater is in the WFSOPFU state.
- Transmitter is in the Inactive state.
- Downstream facing ports are in the Not Configured state and SE0 driven on all downstream facing ports.

11.11 Hub Port Power Control

Self-powered hubs may have power switches that control delivery of power downstream facing ports but it is not required. Bus-powered hubs are required to have power switches. A hub with power switches can switch power to all ports as a group/gang, to each port individually, or have an arbitrary number of gangs of one or more ports.

A hub indicates whether or not it supports power switching by the setting of the Logical Power Switching Mode field in *wHubCharacteristics*. If a hub supports per-port power switching, then the power to a port is turned on when a SetPortFeature(PORT_POWER) request is received for the port. Port power is turned off when the port is in the Powered-off or Not Configured states. If a hub supports ganged power switching, then the power to all ports in a gang is turned on when any port in a gang receives a SetPortFeature(PORT_POWER) request. The power to a gang is not turned off unless all ports in a gang are in the Powered-off or Not Configured states. Note, the power to a port is not turned on by a SetPortFeature(PORT_POWER) if both C_HUB_LOCAL_POWER and Local Power Status (in *wHubStatus*) are set to 1B at the time when the request is executed and the PORT_POWER feature would be turned on.

Although a self-powered hub is not required to implement power switching, the hub must support the Powered-off state for all ports. Additionally, the hub must implement the *PortPwrCtrlMask* (all bits set to 1B) even though the hub has no power switches that can be controlled by the USB System Software.

Note: To ensure compatibility with previous versions of USB Software, hubs must implement the Logical Power Switching Mode field in *wHubCharacteristics*. This is because some versions of SW will not use the SetPortFeature() request if the hub indicates in *wHubCharacteristics* that the port does not support port power switching. Otherwise, the Logical Power Switching Mode field in *wHubCharacteristics* would have become redundant as of this version of the specification.

The setting of the Logical Power Switching Mode for hubs with no power switches should reflect the manner in which over-current is reported. For example, if the hub reports over-current conditions on a per-port basis, then the Logical Power Switching Mode should be set to indicate that power switching is controlled on a per-port basis.

For a hub with no power switches, *bPwrOn2PwrGood* must be set to zero.

11.11.1 Multiple Gangs

A hub may implement any number of power and/or over-current gangs. A hub that implements more than one over-current and/or power switching gang must set both the Logical Power Switching Mode and the Over-current Reporting Mode to indicate that power switching and over-current reporting are on a per port basis (these fields are in *wHubCharacteristics*). Also, all bits in *PortPwrCtrlMask* must be set to 1B.

When an over-current condition occurs on an over-current protection device, the over-current is signaled on all ports that are protected by that device. When the over-current is signaled, all the ports in the group are placed in the Powered-off state, and the C_PORT_OVER-CURRENT field is set to 1B on all the ports. When port status is read from any port in the group, the PORT_OVER-CURRENT field will be set to 1B as

Microchip Ex. 1018, p. 363
Microchip v. Aptiv
IPR2024-00494

long as the over-current condition exists. The C_PORT_OVER-CURRENT field must be cleared in each port individually.

When multiple ports share a power switch, setting PORT_POWER on any port in the group will cause the power to all ports in the group to turn on. It will not, however, cause the other ports in that group to leave the Powered-off state. When all the ports in a group are in the Powered-off state or the hub is not configured, the power to the ports is turned off.

If a hub implements both power switching and over-current, it is not necessary for the over-current groups to be the same as the power switching groups.

If an over-current condition occurs and power switches are present, then all power switches associated with an over-current protection circuit must be turned off. If multiple over-current protection devices are associated with a single power switch then that switch will be turned off when any of the over-current protection circuits indicates an over-current condition.

11.12 Hub Controller

The Hub Controller is logically organized as shown in Figure 11-19.

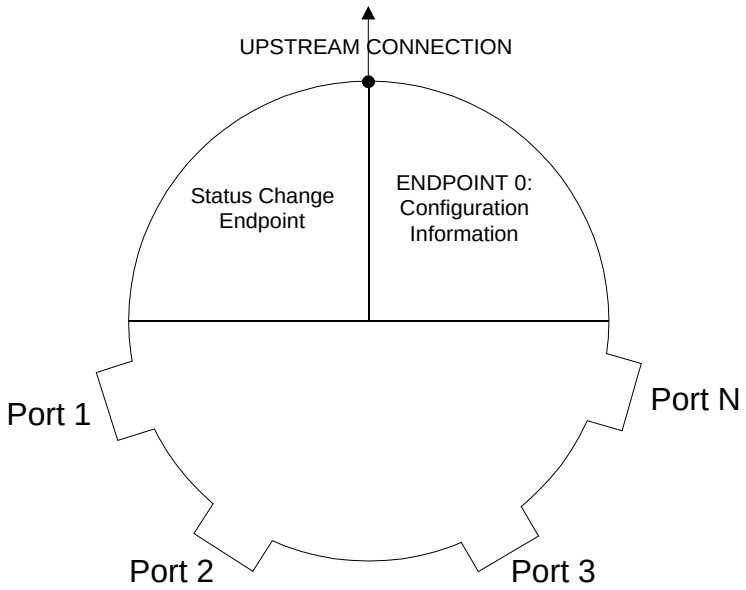


Figure 11-19. Example Hub Controller Organization

11.12.1 Endpoint Organization

The Hub Class defines one additional endpoint beyond Default Control Pipe, which is required for all hubs: the Status Change endpoint. The host system receives port and hub status change notifications through the Status Change endpoint. The Status Change endpoint is an interrupt endpoint. If no hub or port status change bits are set, then the hub returns an NAK when the Status Change endpoint is polled. When a status change bit is set, the hub responds with data, as shown in Section 11.12.4, indicating the entity (hub or port) with a change bit set. The USB System Software can use this data to determine which status registers to access in order to determine the exact cause of the status change interrupt.

Microchip Ex. 1018, p. 364
Microchip v. Aptiv
IPR2024-00494

11.12.2 Hub Information Architecture and Operation

Figure 11-20 shows how status, status change, and control information relate to device states. Hub descriptors and Hub/Port Status and Control are accessible through the Default Control Pipe. The Hub descriptors may be read at any time. When a hub detects a change on a port or when the hub changes its own state, the Status Change endpoint transfers data to the host in the form specified in Section 11.12.4.

Hub or port status change bits can be set because of hardware or Software events. When set, these bits remain set until cleared directly by the USB System Software through a ClearPortFeature() request or by a hub reset. While a change bit is set, the hub continues to report a status change when polled until all change bits have been cleared by the USB System Software.

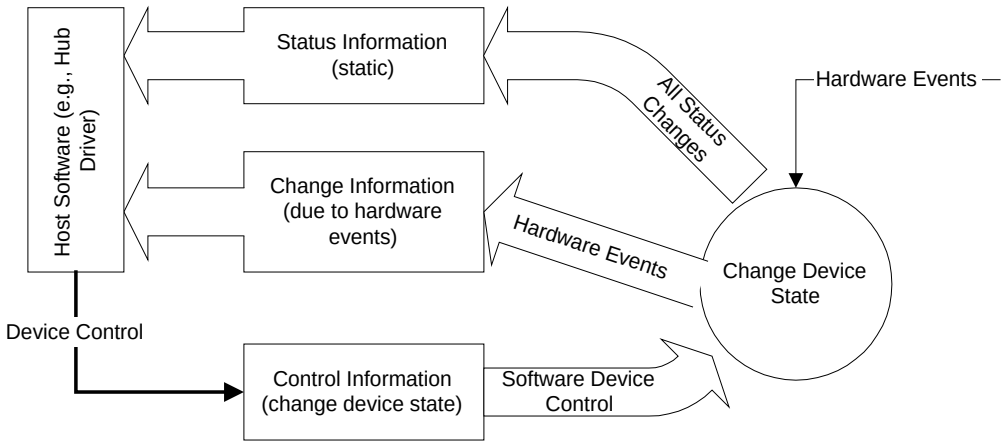


Figure 11-20. Relationship of Status, Status Change, and Control Information to Device States

The USB System Software uses the interrupt pipe associated with the Status Change endpoint to detect changes in hub and port status.

11.12.3 Port Change Information Processing

Hubs report a port's status through port commands on a per-port basis. The USB System Software acknowledges a port change by clearing the change state corresponding to the status change reported by the hub. The acknowledgment clears the change state for that port so future data transfers to the Status Change endpoint do not report the previous event. This allows the process to repeat for further changes (see Figure 11-21).

Microchip Ex. 1018, p. 365
Microchip v. Aptiv
IPR2024-00494

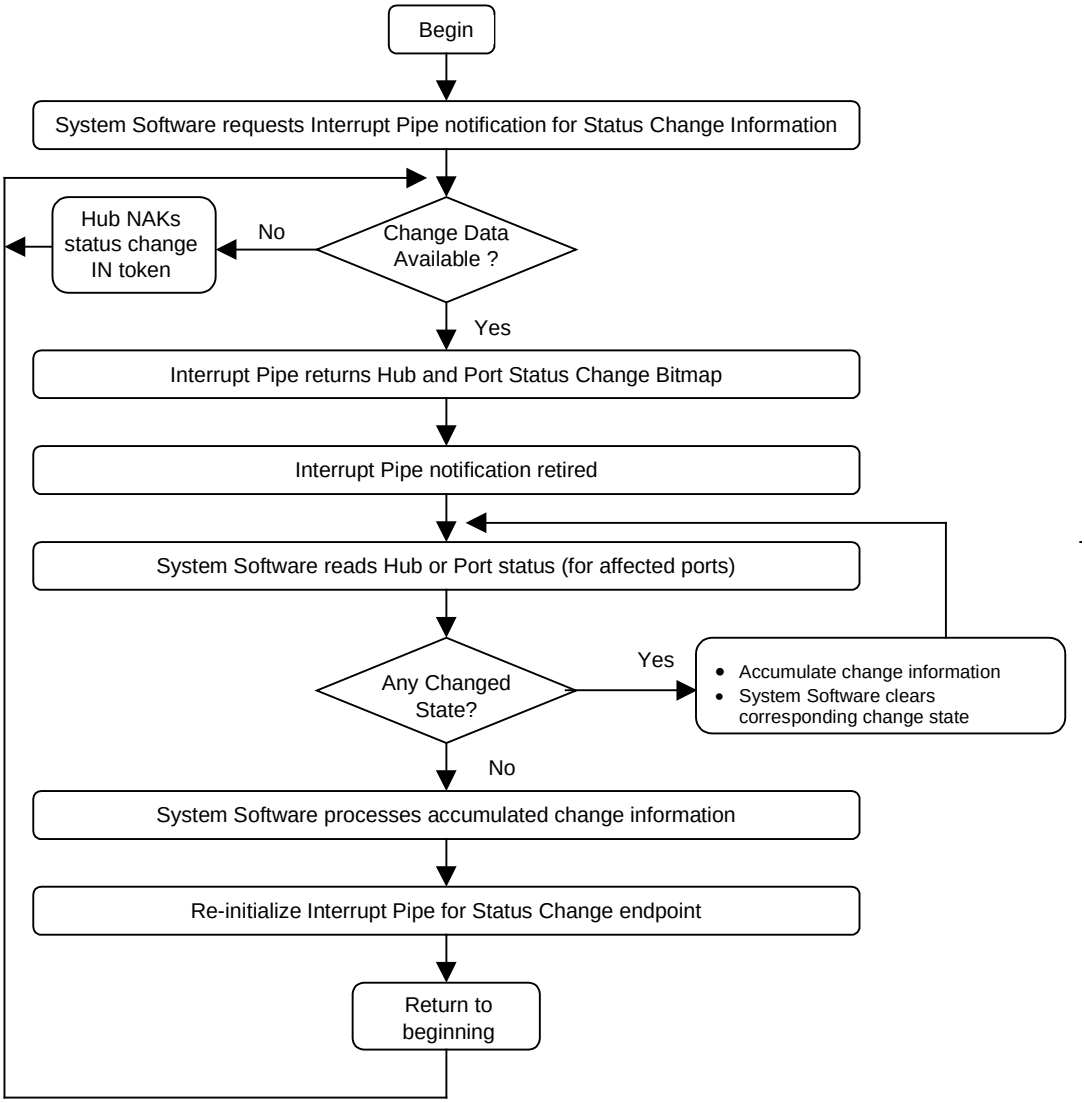


Figure 11-21. Port Status Handling Method

11.12.4 Hub and Port Status Change Bitmap

The Hub and Port Status Change Bitmap, shown in Figure 11-22, indicates whether the hub or a port has experienced a status change. This bitmap also indicates which port(s) has had a change in status. The hub returns this value on the Status Change endpoint. Hubs report this value in byte-increments. That is, if a hub has six ports, it returns a byte quantity, and reports a zero in the invalid port number field locations. The USB System Software is aware of the number of ports on a hub (this is reported in the hub descriptor) and decodes the Hub and Port Status Change Bitmap accordingly. The hub reports any changes in hub status in bit zero of the Hub and Port Status Change Bitmap.

The Hub and Port Status Change Bitmap size varies from a minimum size of one byte. Hubs report only as many bits as there are ports on the hub, subject to the byte-granularity requirement (i.e., round up to the nearest byte).

Microchip Ex. 1018, p. 366
Microchip v. Aptiv
IPR2024-00494

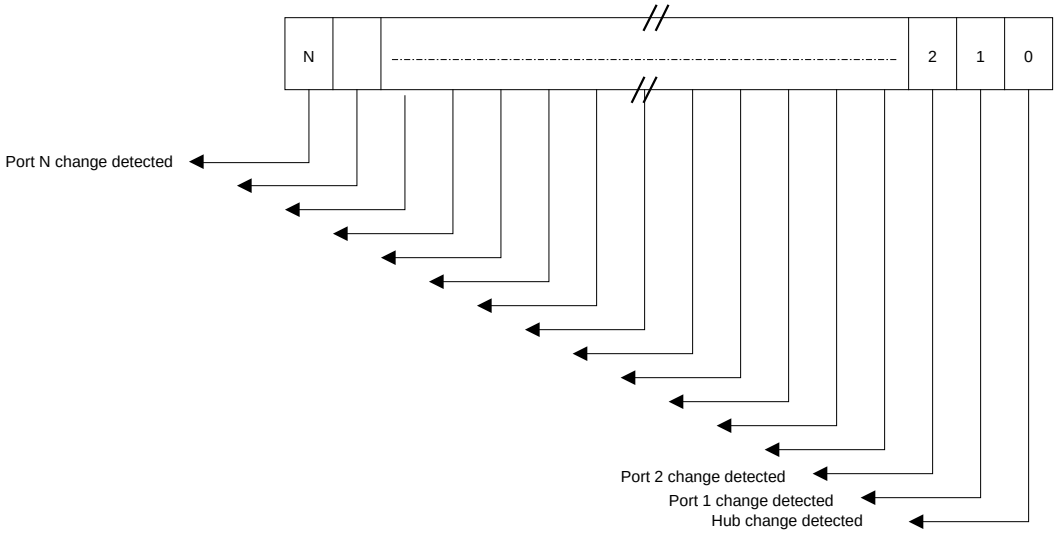


Figure 11-22. Hub and Port Status Change Bitmap

Any time the Status Change endpoint is polled by the host controller and any of the Status Changed bits are non-zero, the Hub and Port Status Change Bitmap is returned. Figure 11-23 shows an example creation mechanism for hub and port change bits.

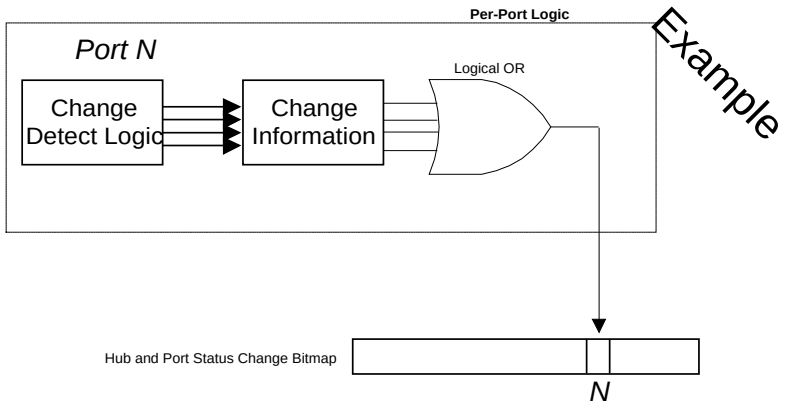


Figure 11-23. Example Hub and Port Change Bit Sampling

11.12.5 Over-current Reporting and Recovery

USB devices must be designed to meet applicable safety standards. Usually, this will mean that a self-powered hub implement current limiting on its downstream facing ports. If an over-current condition occurs, it causes a status and state change in one or more ports. This change is reported to the USB System Software so that it can take corrective action.

A hub may be designed to report over-current as either a port or a hub event. The hub descriptor field *wHubCharacteristics* is used to indicate the reporting capabilities of a particular hub (see Section 11.23.2). The over-current status bit in the hub or port status field indicates the state of the over-current detection when the status is returned. The over-current status change bit in the Hub or Port Change field indicates if the over-current status has changed.

When a hub experiences an over-current condition, it must place all affected ports in the Powered-off state. If a hub has per-port power switching and per-port current limiting, an over-current on one port may still

Microchip Ex. 1018, p. 367
Microchip v. Aptiv
IPR2024-00494

cause the power on another port to fall below specified minimums. In this case, the affected port is placed in the Powered-off state and C_PORT_OVER_CURRENT is set for the port, but PORT_OVER_CURRENT is not set. If the hub has over-current detection on a hub basis, then an over-current condition on the hub will cause all ports to enter the Powered-off state. However, in this case, neither C_PORT_OVER_CURRENT nor PORT_OVER_CURRENT is set for the affected ports.

Host recovery actions for an over-current event should include the following:

1. Host gets change notification from hub with over-current event.
2. Host extracts appropriate hub or port change information (depending on the information in the change bitmap).
3. Host waits for over-current status bit to be cleared to 0.
4. Host cycles power on to all of the necessary ports (e.g., issues a SetPortFeature(PORT_POWER) request for each port).
5. Host re-enumerates all affected ports.

11.12.6 Enumeration Handling

The hub device class commands are used to manipulate its downstream facing port state. When a device is attached, the device attach event is detected by the hub and reported on the status change interrupt. The host will accept the status change report and request a SetPortFeature(PORT_RESET) on the port. As part of the bus reset sequence, a speed detect is performed by the hub's port hardware.

The Get_Status(PORT) request invoked by the host will return a "not PORT_LOW_SPEED and PORT_HIGH_SPEED" indication for a downstream facing port operating at high-speed. The Get_Status(PORT) will report "PORT_LOW_SPEED" for a downstream facing port operating at low-speed. The Get_Status(PORT) will report "not PORT_LOW_SPEED and not PORT_HIGH_SPEED" for a downstream facing port operating at full-speed.

When the device is detached from the port, the port reports the status change through the status change endpoint and the port will be reconnected to the high-speed repeater. Then the process is ready to be repeated on the next device attach detect.

11.13 Hub Configuration

Hubs are configured through the standard USB device configuration commands. A hub that is not configured behaves like any other device that is not configured with respect to power requirements and addressing. If a hub implements power switching, no power is provided to the downstream facing ports while the hub is not configured. Configuring a hub enables the Status Change endpoint. The USB System Software may then issue commands to the hub to switch port power on and off at appropriate times.

The USB System Software examines hub descriptor information to determine the hub's characteristics. By examining the hub's characteristics, the USB System Software ensures that illegal power topologies are not allowed by not powering on the hub's ports if doing so would violate the USB power topology. The device status and configuration information can be used to determine whether the hub should be used as a bus or self-powered device. Table 11-12 summarizes the information and how it can be used to determine the current power requirements of the hub.

Microchip Ex. 1018, p. 368
Microchip v. Aptiv
IPR2024-00494

Table 11-12. Hub Power Operating Mode Summary

Configuration Descriptor		Hub Device Status (Self Power)	Explanation
MaxPower	bmAttributes (Self Powered)		
0	0	N/A	N/A This is an illegal set of information.
0	1	0	N/A A device which is only self-powered, but does not have local power cannot connect to the bus and communicate.
0	1	1	Self-powered only hub and local power supply is good. Hub status also indicates local power good, see Section 11.16.2.5. Hub functionality is valid anywhere depth restriction is not violated.
> 0	0	N/A	Bus-powered only hub. Downstream facing ports may not be powered unless allowed in current topology. Hub device status reporting Self Powered is meaningless in combination of a zeroed *bmAttributes.Self-Powered*.
> 0	1	0	This hub is capable of both self- and bus-powered operating modes. It is currently only available as a bus-powered hub.
> 0	1	1	This hub is capable of both self- and bus-powered operating modes. It is currently available as a self-powered hub.

A self-powered hub has a local power supply, but may optionally draw one unit load from its upstream connection. This allows the interface to function when local power is not available (see Section 7.2.1.2). When local power is removed (either a hub-wide over-current condition or local supply is off), a hub of this type remains in the Configured state but transitions all ports (whether removable or non-removable) to the Powered-off state. While local power is off, all port status and change information read as zero and all SetPortFeature() requests are ignored (request is treated as a no-operation). The hub will use the Status Change endpoint to notify the USB System Software of the hub event (see Section 11.24.2.6 for details on hub status).

The *MaxPower* field in the configuration descriptor is used to report to the system the maximum power the hub will draw from VBUS when the configuration is selected. For bus-powered hubs, the reported value must not include the power for any of external downstream facing ports. The external devices attaching to the hub will report their individual power requirements.

A compound device may power both the hub electronics and the permanently attached devices from VBUS. The entire load may be reported in the hubs' configuration descriptor with the permanently attached devices each reporting self-powered, with zero *MaxPower* in their respective configuration descriptors.

Microchip Ex. 1018, p. 369
Microchip v. Aptiv
IPR2024-00494

11.14 Transaction Translator

A hub has a special responsibility when it is operating in high-speed and has full-/low-speed devices connected on downstream facing ports. In this case, the hub must isolate the high-speed signaling environment from the full-/low-speed signaling environment. This function is performed by the Transaction Translator (TT) portion of the hub.

This section defines the required behavior of the transaction translator.

11.14.1 Overview

Figure 11-24 shows an overview of the Transaction Translator. The TT is responsible for participating in high-speed split transactions on the high-speed bus via its upstream facing port and issuing corresponding full-/low-speed transactions on its downstream facing ports that are operating at full-/low-speed. The TT acts as a high-speed function on the high-speed bus and performs the role of a host controller for its downstream facing ports that are operating at full-/low-speed. The TT includes a high-speed handler to deal with high-speed transactions. The TT also includes a full-/low-speed handler that performs the role of a host controller on the downstream facing ports that are operating at full-/low-speed.

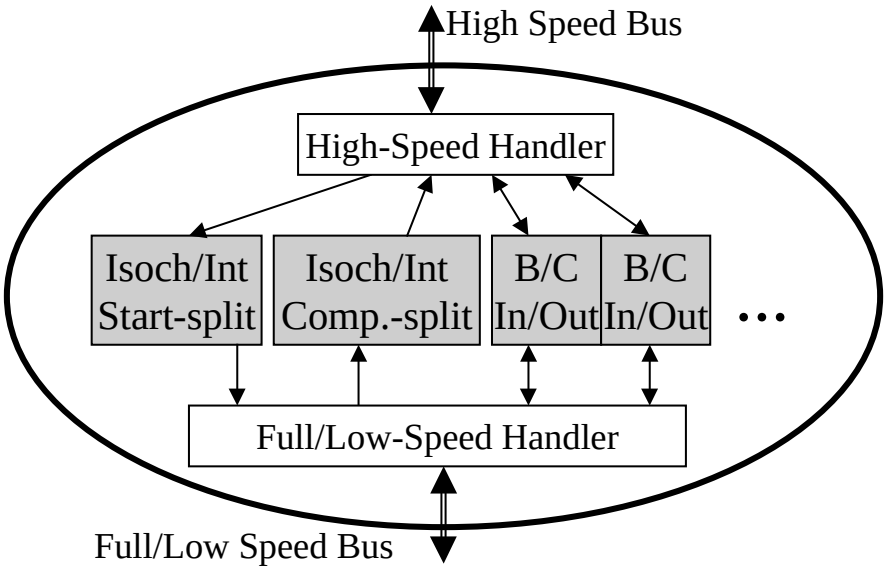


Figure 11-24. Transaction Translator Overview

The TT has buffers (shown in gray in the figure) to hold transactions that are in progress and tracks the state of each buffered transaction as it is processed by the TT. The buffers provide the connection between the high-speed and full-/low-speed handlers. The state tracking the TT does for each transaction depends on the specific USB transfer type of the transaction (i.e., bulk, control, interrupt, isochronous). The high-speed handler accepts high-speed start-split transactions or responds to high-speed complete-split transactions. The high-speed handler places the start-split transactions in local buffers for the full-/low-speed handler's use.

The buffered start-split transactions provide the full-/low-speed handler with the information that allows it to issue corresponding full-/low-speed transactions to full-/low-speed devices attached on downstream facing ports. The full-/low-speed handler buffers the results of these full-/low-speed transactions so that they can be returned with a corresponding complete-split transaction on the high-speed bus.

The general conversion between full-/low-speed transactions and the corresponding high-speed split transaction protocol is described in Section 8.4.2. More details about the specific transfer types for split transactions are described later in this chapter.

Microchip Ex. 1018, p. 370
Microchip v. Aptiv
IPR2024-00494

The high-speed handler of the TT operates independently of the full-/low-speed handler. Both handlers use the local transaction buffers to exchange information where required.

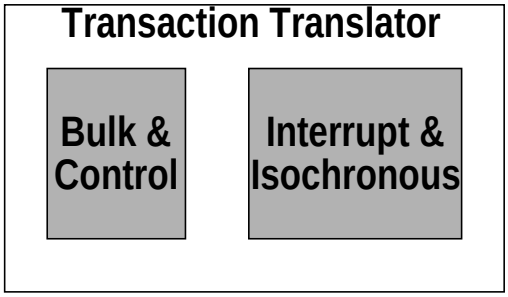


Figure 11-25. Periodic and Non-periodic Buffer Sections of TT

The TT has two buffer and state tracking sections (shown in gray in Figure 11-24 and Figure 11-25): periodic (for isochronous/interrupt full-/low-speed transactions) and non-periodic (for bulk/control full-/low-speed transactions). The requirements on the TT for these two buffer and state tracking sections are different. Each will be described in turn later in this chapter.

11.14.1.1 Data Handling Between High-speed and Full-/low-speed

The host converts transfer requests involving a full-/low-speed device into corresponding high-speed split transactions to the TT to which the device is attached.

Low-speed Preamble(PRE) packets are never used on the high-speed bus to indicate a low-speed transaction. Instead, a low-speed transaction is encoded in the split transaction token.

The host can have a single schedule of the transactions that need to be issued to devices. This single schedule can be used to hold both high-speed transactions and high-speed split transactions used for communicating with full-/low-speed devices.

11.14.1.2 Host Controller and TT Split Transactions

The host controller uses the split transaction protocol for initiating full-/low-speed transactions via the TT and then determining the completion status of the full-/low-speed transaction. This approach allows the host controller to start a full-/low-speed transaction and then continue with other high-speed transactions while avoiding having to wait for the slower transaction to proceed/complete at its speed. A high-speed split transaction has two parts: a start-split and a complete-split. Split transactions are only used between the host controller and a hub. No other high-/full-/low-speed devices ever participate in split transactions.

When the host controller sends a start-split transaction at high-speed, the split transaction is addressed to the TT for that device. That TT will accept the transaction and buffer it locally. The high-speed handler responds with an appropriate handshake to inform the host controller that the transaction has been accepted. Not all split transactions have a handshake phase to the start-split. The start-split transactions are kept temporarily in a TT transaction buffer.

The full-/low-speed handler processes start-split periodic transactions stored in the periodic transaction buffer (in order) as the downstream full-/low-speed bus is ready for the “next” transaction. The full-/low-speed handler accepts any result information from the downstream bus (in response to the full-/low-speed transaction) and accumulates it in a local buffer for later transmission to the host controller.

At an appropriate future time, the host controller sends a high-speed complete-split transaction to retrieve the status/data/result for appropriate full-/low-speed transactions. The high-speed handler checks this high-speed complete-split transaction with the response at the head of the appropriate local transaction buffer and responds accordingly. The specific split transaction sequences are defined for each USB transfer type in later sections.

Microchip Ex. 1018, p. 371
Microchip v. Aptiv
IPR2024-00494

11.14.1.3 Multiple Transaction Translators

A hub has two choices for organizing transaction translators (TTs). A hub can have one TT for all downstream facing ports that have full-/low-speed devices attached or the hub can have one TT for each downstream facing port. The hub must report its organization in the hub class descriptor.

11.14.2 Transaction Translator Scheduling

As the high-speed handler accepts start-splits, the full-/low-speed transaction information and data for OUTs or the transaction information for INs accumulate in buffers awaiting their service on the downstream bus. The host manages the periodic TT transaction buffers differently than the non-periodic transaction buffers.

11.14.2.1 TT Isochronous/Interrupt (Periodic) Transaction Buffering

Periodic transactions have strict timing requirements to meet on a full-/low-speed bus (as defined by the specific endpoint and transfer type). Therefore, transactions must move across the high-speed bus, through the TT, across the full-/low-speed bus, back through the TT, and onto the high-speed bus in a timely fashion. An overview of the microframe pipeline of buffering in the TT is shown in Figure 11-26. A transaction begins as a start-split on the high-speed bus, is accepted by the high-speed handler, and is stored in the start-split transaction buffer. The full-/low-speed handler uses the next start-split transaction at the head of the start-split transaction buffer when it is time to issue the next periodic full-/low-speed transaction on the downstream bus. The results of the transaction are accumulated in the complete-split transaction buffer. The TT responds to a complete-split from the host and extracts the appropriate response from the complete-split transaction buffer. This completes the flow for a periodic transaction through the TT. This is called the periodic transaction pipeline.

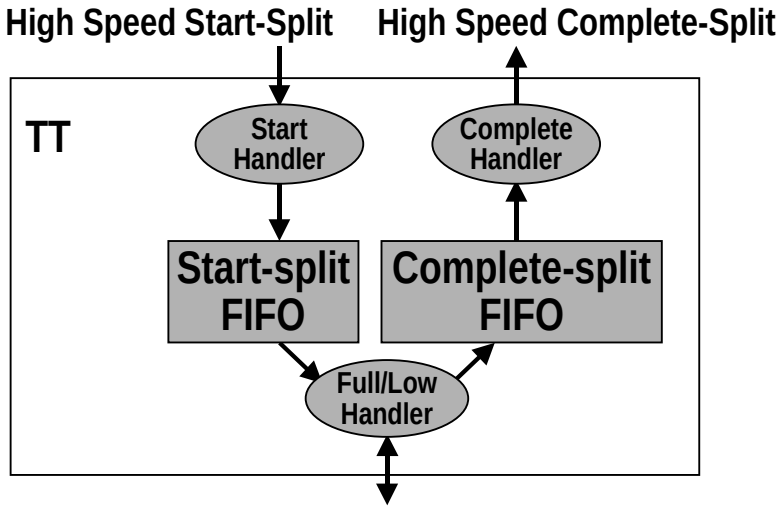


Figure 11-26. TT Microframe Pipeline for Periodic Split Transactions

The TT implements a traditional pipeline of transactions with its periodic transaction buffers. There is separate buffer space for start-splits and complete-splits. The host is responsible for filling the start-split transaction buffer and draining the complete-split transaction buffer. The host software manages the host controller to cause high-speed split transactions at the correct times to avoid over/under runs in the TT periodic transaction buffers. The host controller sends data “just in time” for full-/low-speed OUTs and retrieves response data from full-/low-speed INs to ensure that the periodic transaction buffer space required in the TT is the minimum possible. See Section 11.18 for more detailed information.

USB strictly defines the timing requirements of periodic transactions and the isochronous transport capabilities of the high-speed and full-/low-speed buses. This allows the host to accurately predict when

Microchip Ex. 1018, p. 372
Microchip v. Aptiv
IPR2024-00494

data for periodic transactions must be moved on both the full-/low-speed and high-speed buses, whenever a client requests a data transfer with a full-/low-speed periodic endpoint. Therefore, the host can "pipeline" data to/from the TT so that it moves in a timely manner with its target endpoint. Once the configuration of a full-/low-speed device with periodic endpoints is set, the host streams data to/from the TT to keep the device's endpoints operating normally.

11.14.2.2 TT Bulk/Control (Non-Periodic) Transaction Buffering

Non-periodic transactions have no timing requirements, but the TT supports the maximum full-/low-speed throughput allowed. A TT provides a few transaction buffers for bulk/control full-/low-speed transactions. The host and TT use simple flow control (NAK) mechanisms to manage the bulk/control non-periodic transaction buffers. The host issues a start-split transaction, and if there is available buffer space, the TT accepts the transaction. The full-/low-speed handler uses the buffered information to issue the downstream full-/low-speed transaction and then uses the same buffer to hold any results (e.g., handshake or data or timeout). The buffer is then emptied with a corresponding high-speed complete-split and the process continues. Figure 11-27 shows an example overview of a TT that has two bulk/control buffers.

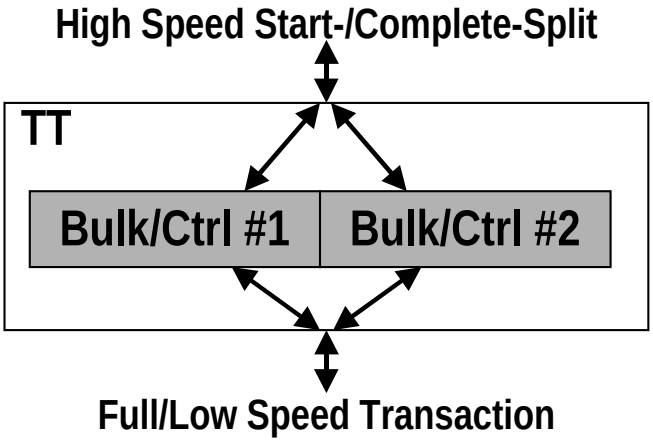


Figure 11-27. TT Nonperiodic Buffering

11.14.2.3 Full-/low-speed Handler Transaction Scheduling

The full-/low-speed handler uses a simple, scheduled priority scheme to service pending transactions on the downstream bus. Whenever the full-/low-speed handler finishes a transaction on the downstream bus, it takes the next start-split transaction from the start-split periodic transaction buffer (if any). If there are no available start-split periodic transactions in the buffer, the full-/low-speed handler may attempt a bulk/control transaction. If there are start-split transactions pending in the bulk/control buffer(s) and there is sufficient time left in the full-/low-speed 1 ms frame to complete the transaction, the full-/low-speed handler issues one of the bulk/control transactions (in round robin order). Figure 11-28 shows pseudo code for the full-/low-speed handler start-split transaction scheduling algorithm.

The TT also sequences the transaction pipeline based on the high-speed microframe timer to ensure that it does not start full-/low-speed periodic transactions too early or too late. The "Advance_pipeline" procedure in the pseudo code is used to keep the TT advancing the microframe "pipeline". This procedure is described in more detail later in Figure 11-67.

Microchip Ex. 1018, p. 373
Microchip v. Aptiv
IPR2024-00494

```
While (1) loop
   While (not end of microframe) loop
      -- process next start-split transaction
      If available periodic start-split transaction then
         Process next full-/low-speed periodic transaction
      Else if (available bulk/control transaction) and
              (fits in full-/low-speed 1 ms frame) then
         Process one transaction
      End if
   End loop

   Advance_Pipeline();  -- see description in Figure 11-67(below)
End loop
```

Figure 11-28. Example Full-/low-speed Handler Scheduling for Start-splits

As described earlier in this chapter, the TT derives the downstream bus's 1 ms SOF timer from the high-speed 125 µs microframe. This means that the host and the TT have the same 1 ms frame time for all TTs. Given the strict relationship between frames and the zeroth microframe, there is no need to have any explicit timing information carried in the periodic split transactions sent to the TT. See Section 11.18 for more information.

11.15 Split Transaction Notation Information

The following sections describe the details of the transaction phases and flow sequences of split transactions for the different USB transfer types: bulk/control, interrupt, and isochronous. Each description also shows detailed example host and TT state machines to achieve the required transaction definitions. The diagrams should not be taken as a required implementation, but to specify the required behavior. Appendix A includes example high-speed and full-speed transaction sequences with different results to clarify the relationships between the host controller, the TT, and a full-speed endpoint.

Low-speed is not discussed in detail since beyond the handling of the PRE packet (which is defined in Chapter 8), there are no packet sequencing differences between low- and full-speed.

For each data transfer direction, reference figures also show the possible flow sequences for the start-split and the complete-split portion of each split transaction transfer type.

The transitions on the flow sequence figures have labels that correspond to the transitions in the host and TT state machines. These labels are also included in the examples in Appendix A. The three character labels are of the form: < S | C >< T | D | H | E ><number>. S indicates that this is a start-split label. C indicates that this is a complete-split label. T indicates token phase; D indicates data phase; H indicates handshake phase; E indicates an error case. The number simply distinguishes different labels of the same case/phase in the same split transaction part.

The flow sequence figures further identify the visibility of transitions according to the legend in Figure 11-29. The flow sequences also include some indication of states required in the host or TT or actions taken. The legend shown in Figure 11-29 indicates how these are identified.

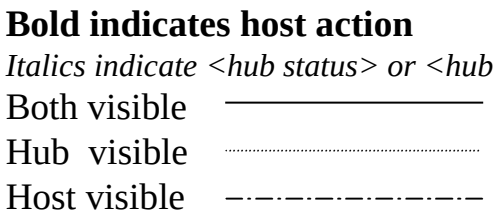


Figure 11-29. Flow Sequence Legend

Figure 11-30 shows the legend for the state machine diagrams. A circle with a three line border indicates a reference to another (hierarchical) state machine. A circle with a two line border indicates an initial state. A circle with a single line border is a simple state.

Microchip Ex. 1018, p. 374
Microchip v. Aptiv
IPR2024-00494

A diamond (joint) is used to join several transitions to a common point. A joint allows a single input transition with multiple output transitions or multiple input transitions and a single output transition. All conditions on the transitions of a path involving a joint must be true for the path to be taken. A path is simply a sequence of transitions involving one or more joints.

A transition is labeled with a block with a line in the middle separating the (upper) condition and the (lower) actions. The condition is required to be true to take the transition. The actions are performed if the transition is taken. The syntax for actions and conditions is VHDL. A circle includes a name in bold and optionally one or more actions that are performed upon entry to the state.

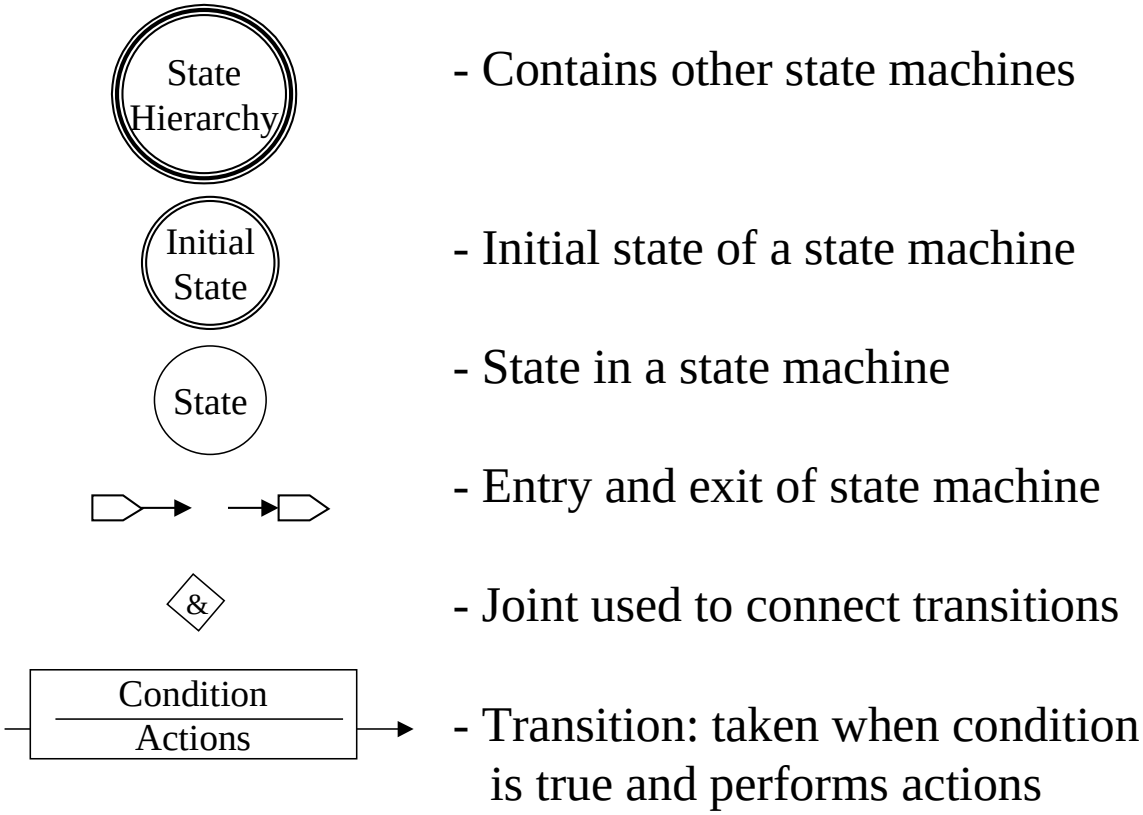


Figure 11-30. Legend for State Machines

The descriptions of the split transactions for the four transfer types refer to the status of the full-/low-speed transaction on the bus downstream of the TT. This status is used by the high-speed handler to determine its response to a complete-split transaction. The status is only visible within a TT implementation and is used in the specification purely for ease of explanation. The defined status values are:

- Ready – The transaction has completed on the downstream facing full-/low-speed bus with the result as follows:
  - Ready/NAK – A NAK handshake was received.
  - Ready /trans_err – The full-/low-speed transaction experienced a error in the transaction. Possible errors are: PID to PID_invert bits check failure, CRC5 check failure, incorrect PID, timeout, CRC16 check failure, incorrect packet length, bitstuffing error, false EOP.
  - Ready /ACK – An ACK handshake was received.
  - Ready /Stall – A STALL handshake was received.
  - Ready /Data – A data packet was received and the CRC check passed. (bulk/control IN).

Microchip Ex. 1018, p. 375
Microchip v. Aptiv
IPR2024-00494

- Ready /lastdata – A data packet was finished being received. (isochronous/interrupt IN).
- Ready /moredata – A data packet was being received when the microframe timer occurred (isochronous/interrupt IN).

- Old – A complete-split has been received by the high-speed handler for a transaction that previously had a “ready” status. The possible status results are the same as for the Ready status. This is the initial state for a buffer before it has been used for a transaction.
- Pending – The transaction is waiting to be completed on the downstream facing full-/low-speed bus.

The figures use “old/x” and “ready/x” to indicate any of the old or ready status respectively.

The split transaction state machines in the remainder of this chapter are presented in the context of Figure 11-31. The host controller state machines are located in the host controller. The host controller causes packets to be issued downstream (labeled as HSD1) and it receives upstream packets (labeled as HSU2).

The transaction translator state machines are located in the TT. The TT causes packets to be issued upstream (labeled as HSU1) and it receives downstream packets (labeled as HSD2).

The host controller has commands that tell it what split transaction to issue next for an endpoint. The host controller tracks transactions for several endpoints. The TT has state in buffers that track transactions for several endpoints.

Appendix B includes some declarations that were used in constructing the state machines and may be useful in understanding additional details of the state machines. There are several pseudo-code procedures and functions for conditions and actions. Simple descriptions of them are also included in Appendix B.

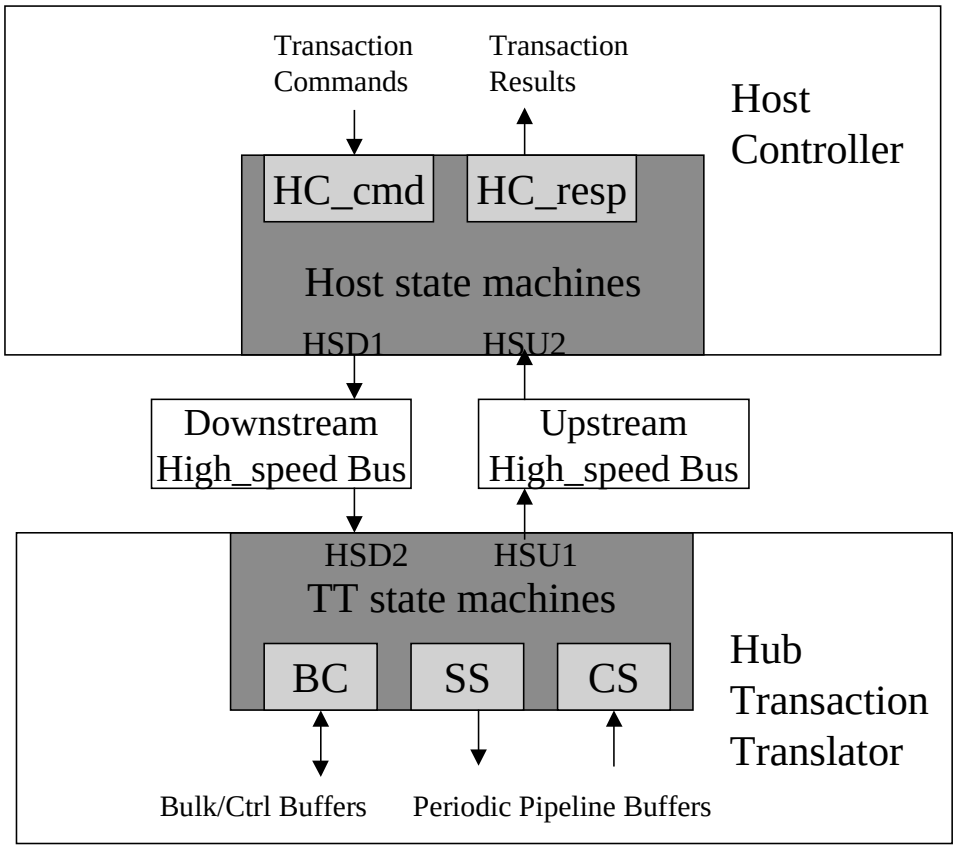


Figure 11-31. State Machine Context Overview

Microchip Ex. 1018, p. 376
Microchip v. Aptiv
IPR2024-00494

11.16 Common Split Transaction State Machines

There are several state machines common to all the specific split transaction types. These state machines are used in the host controller and transaction translator to determine the specific split transaction type (e.g., interrupt OUT start-split vs. bulk IN complete-split). An overview of the host controller state machine hierarchy is shown in Figure 11-32. The overview of the transaction translator state machine hierarchy is shown in Figure 11-33. Each of the labeled boxes in the figures show an individual state machine. Boxes contained in another box indicate a state machine contained within another state machine. All the state machines except the lowest level ones are shown in the remaining figures in this section. The lowest level state machines are shown in later sections describing the specific split transaction type.

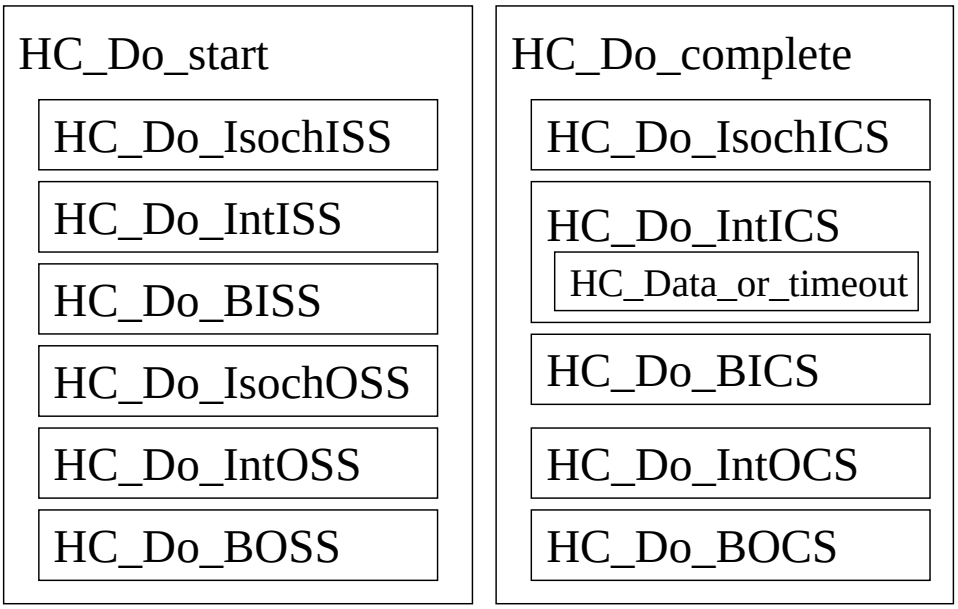


Figure 11-32. Host Controller Split Transaction State Machine Hierarchy Overview

Microchip Ex. 1018, p. 377
Microchip v. Aptiv
IPR2024-00494

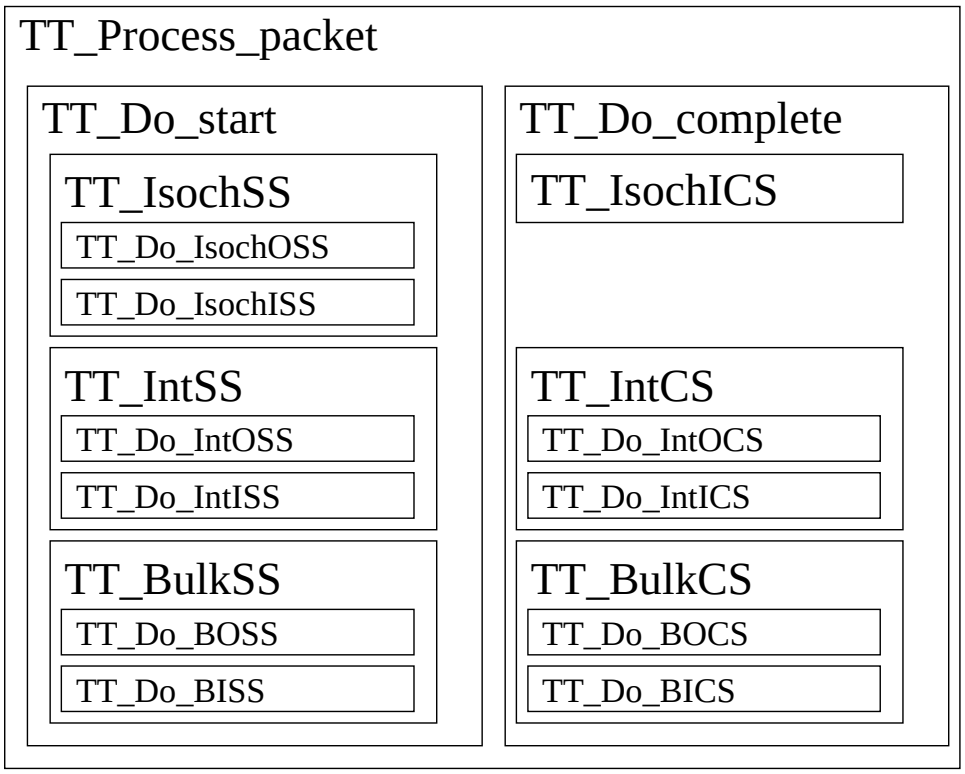


Figure 11-33. Transaction Translator State Machine Hierarchy Overview

11.16.1 Host Controller State Machine

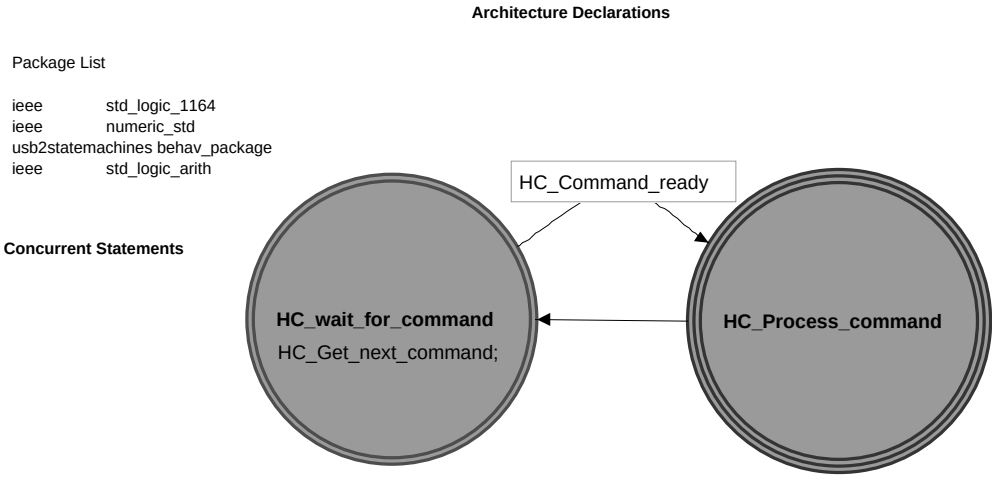


Figure 11-34. Host Controller

Microchip Ex. 1018, p. 378
Microchip v. Aptiv
IPR2024-00494

11.16.1.1 HC_Process_command State Machine

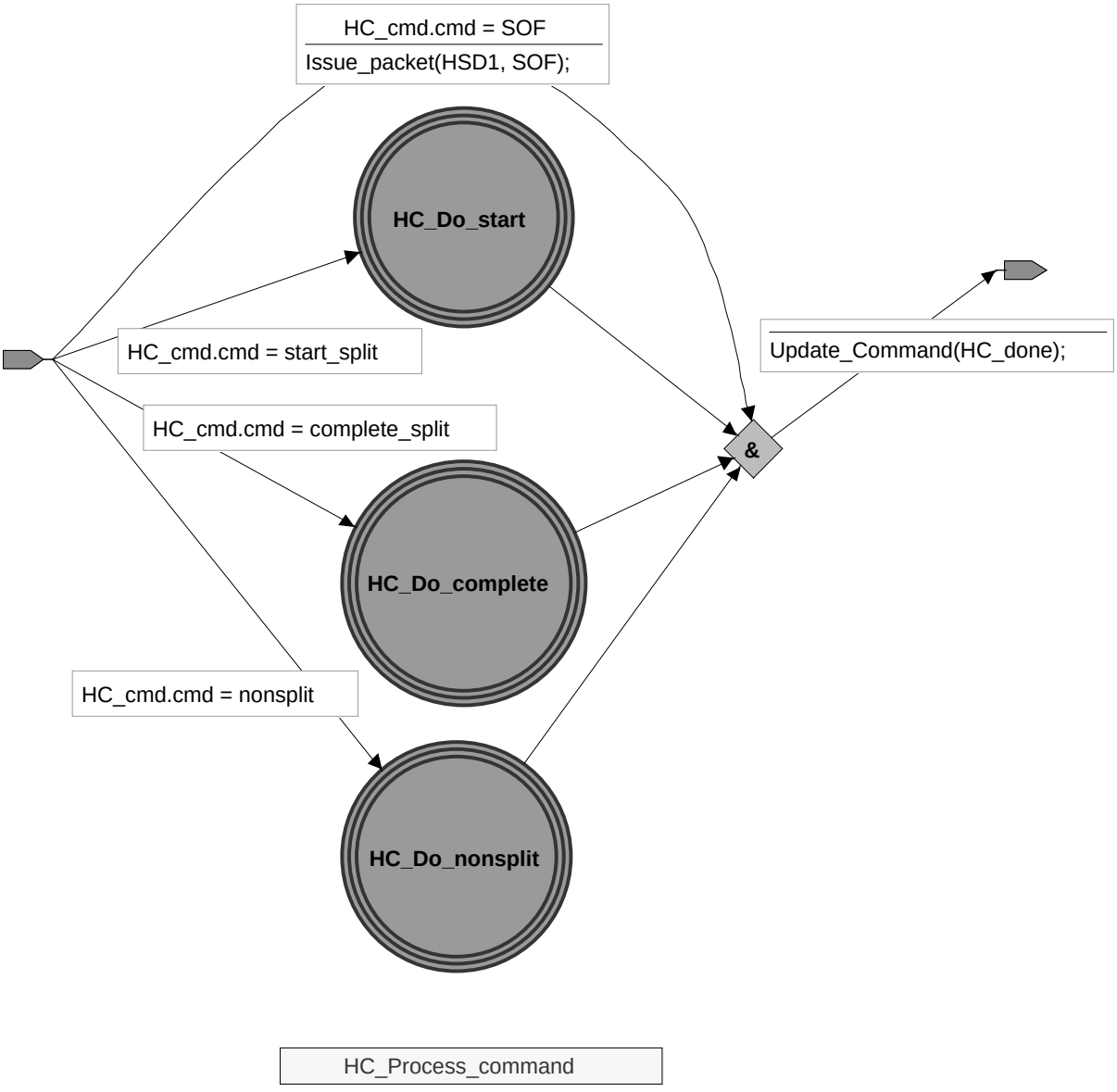


Figure 11-35. HC_Process_Command

Microchip Ex. 1018, p. 379
Microchip v. Aptiv
IPR2024-00494

11.16.1.1.1 HC_Do_start State Machine

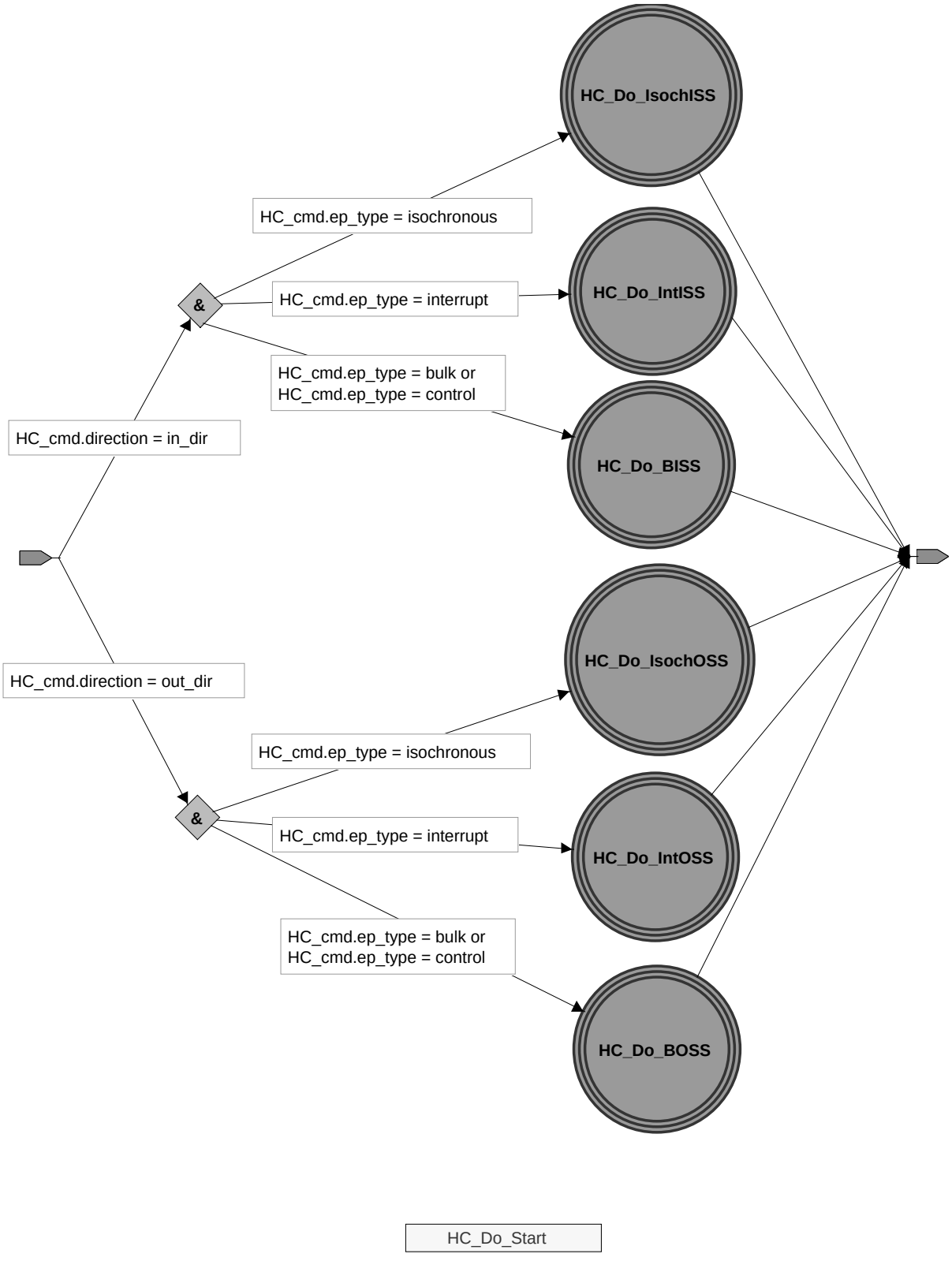


Figure 11-36. HC_Do_Start

Microchip Ex. 1018, p. 380
Microchip v. Aptiv
IPR2024-00494

11.16.1.1.2 HC_Do_complete State Machine

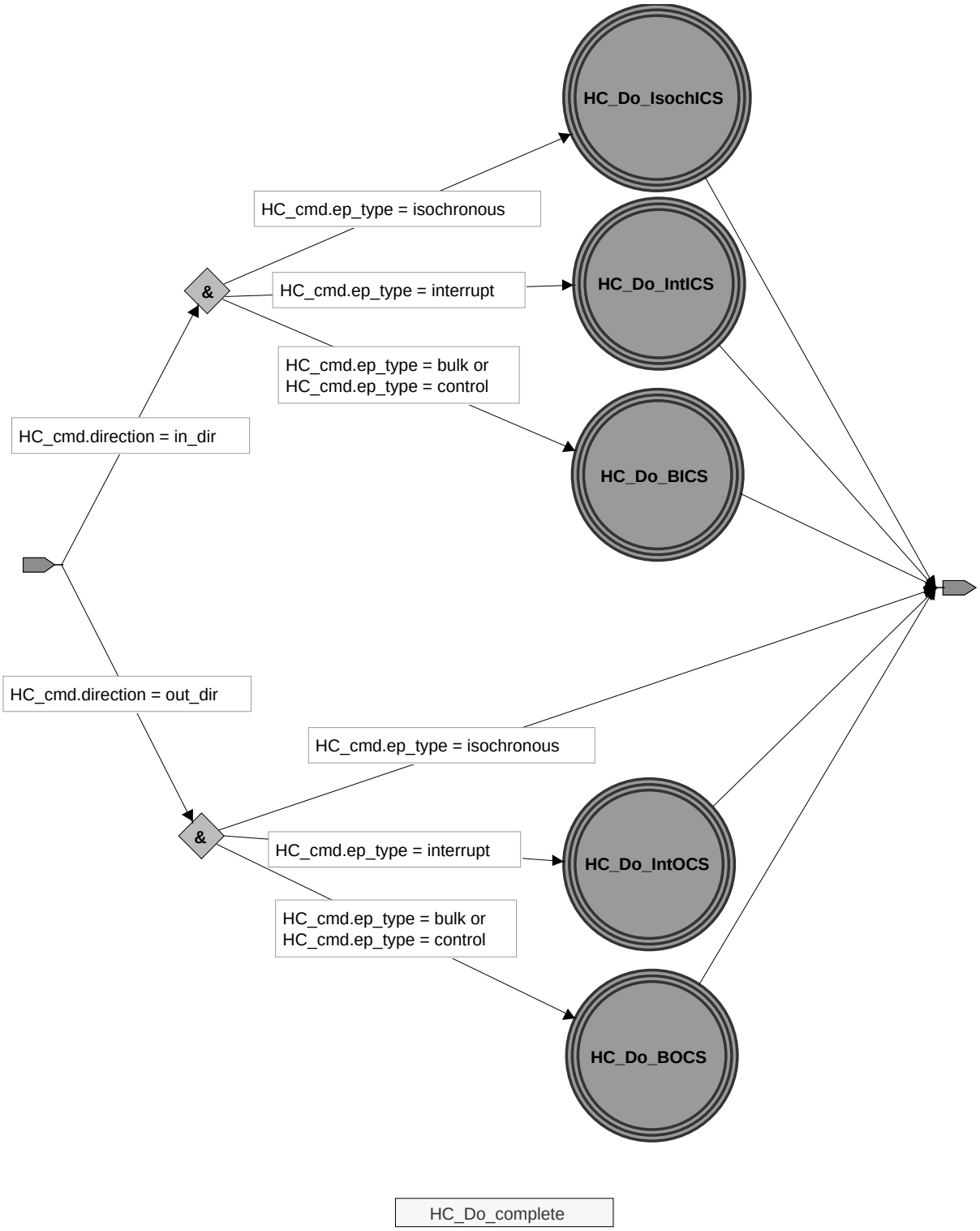


Figure 11-37. HC_Do_Complete

Microchip Ex. 1018, p. 381
Microchip v. Aptiv
IPR2024-00494

11.16.2 Transaction Translator State Machine

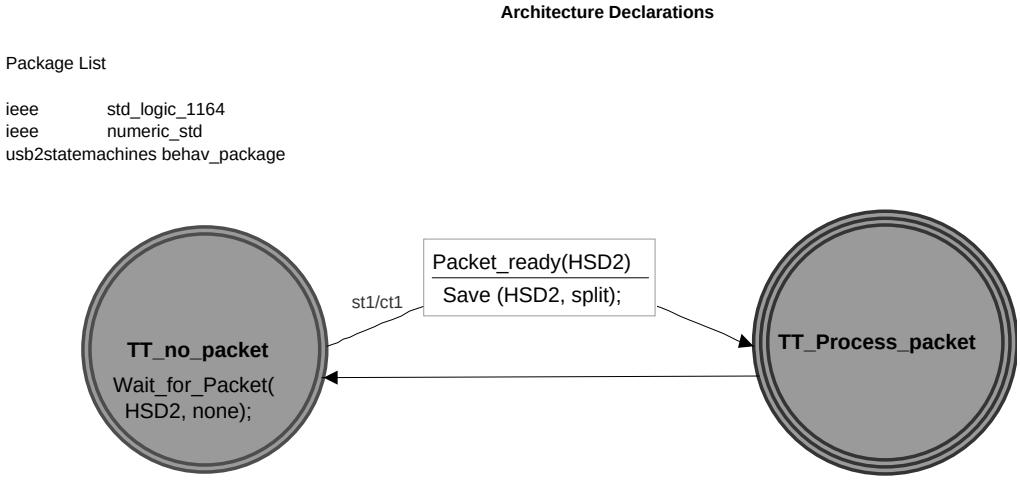


Figure 11-38. Transaction Translator

Microchip Ex. 1018, p. 382
Microchip v. Aptiv
IPR2024-00494

11.16.2.1 TT_Process_packet State Machine

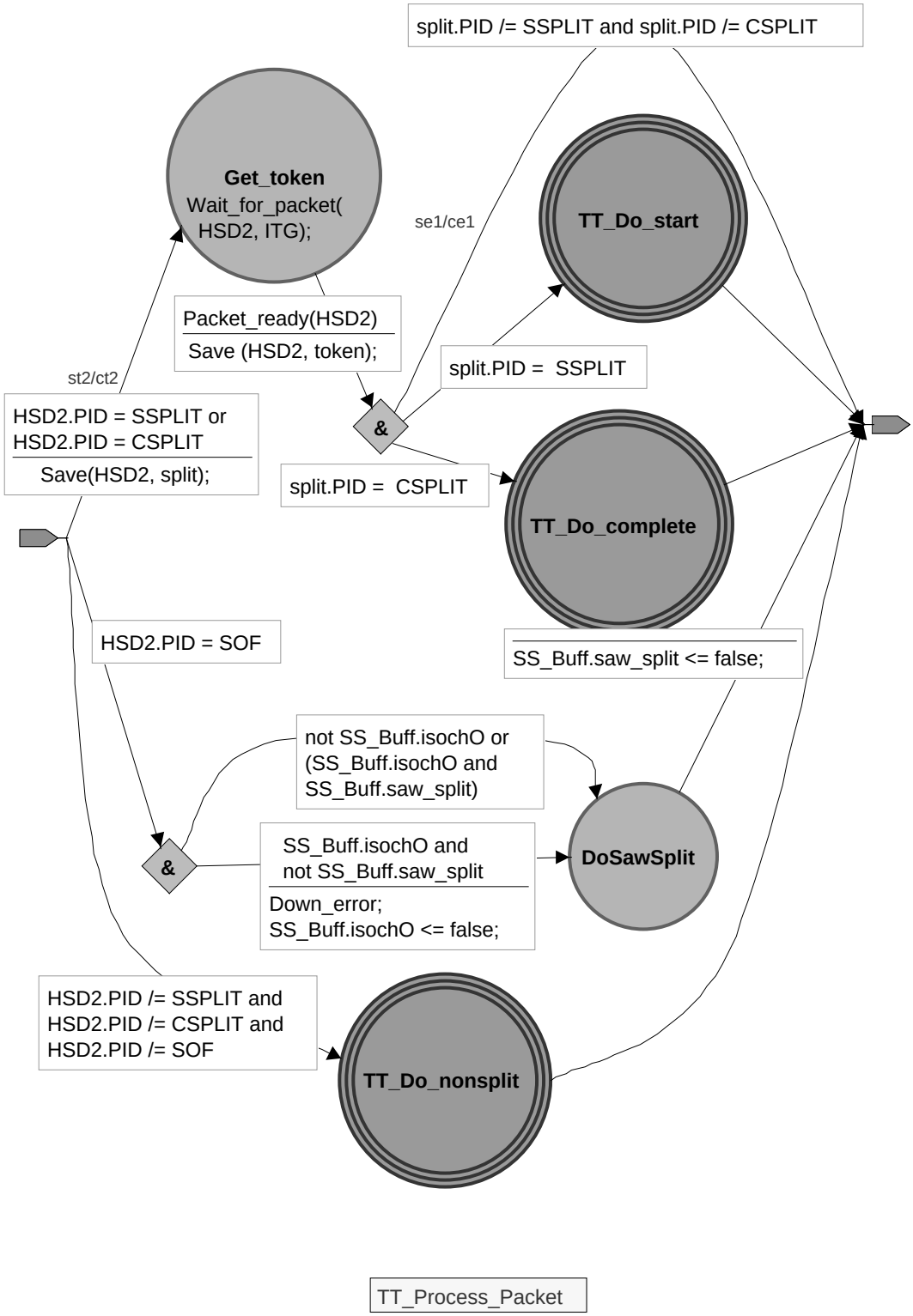


Figure 11-39. TT_Process_Packet

Microchip Ex. 1018, p. 383
Microchip v. Aptiv
IPR2024-00494

11.16.2.1.1 TT_Do_Start State Machine

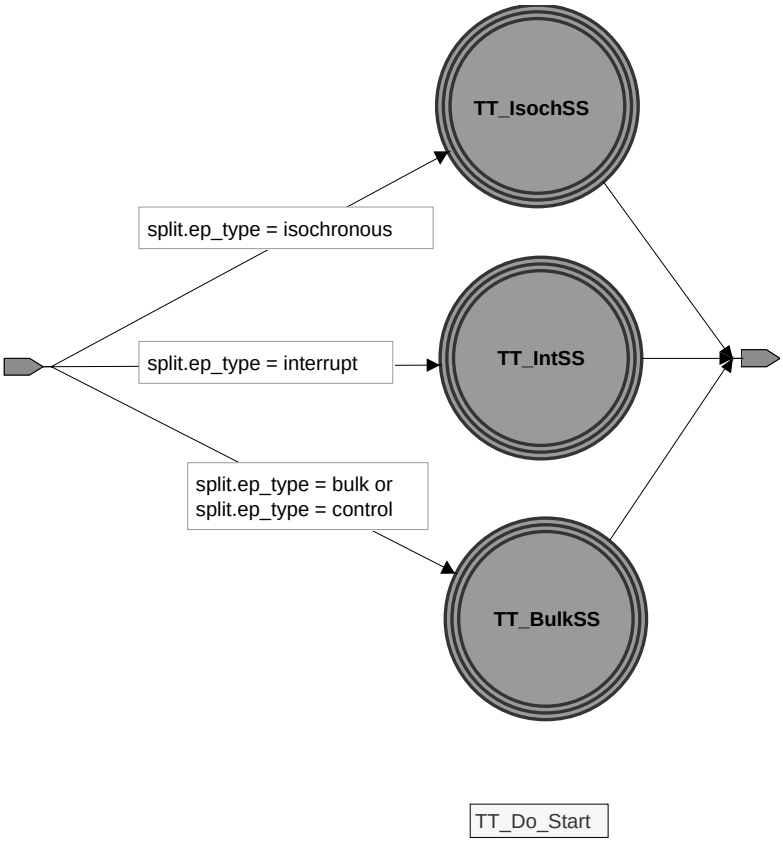


Figure 11-40. TT_Do_Start

Microchip Ex. 1018, p. 384
Microchip v. Aptiv
IPR2024-00494

11.16.2.1.2 TT_Do_Complete State Machine

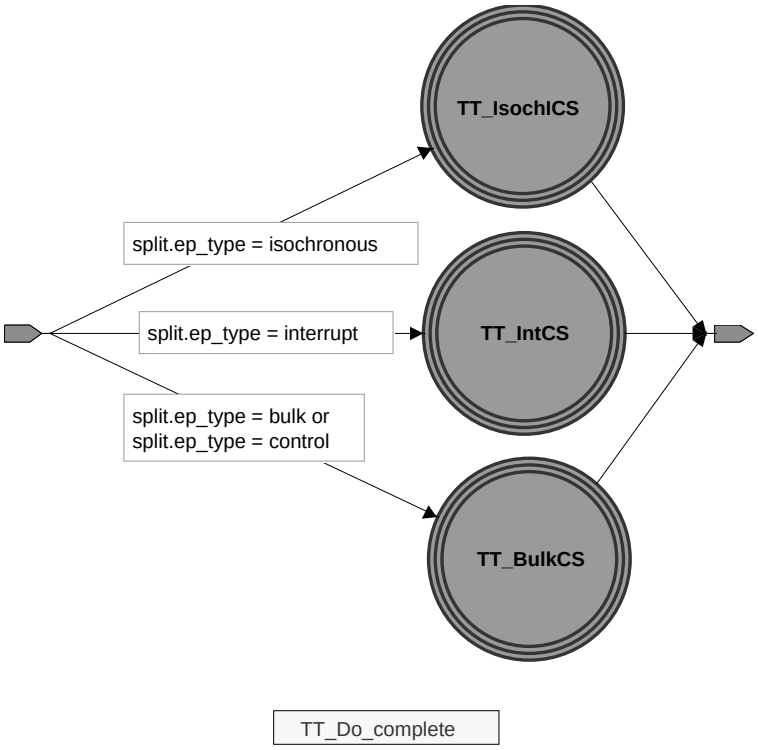


Figure 11-41. TT_Do_Complete

11.16.2.1.3 TT_BulkSS State Machine

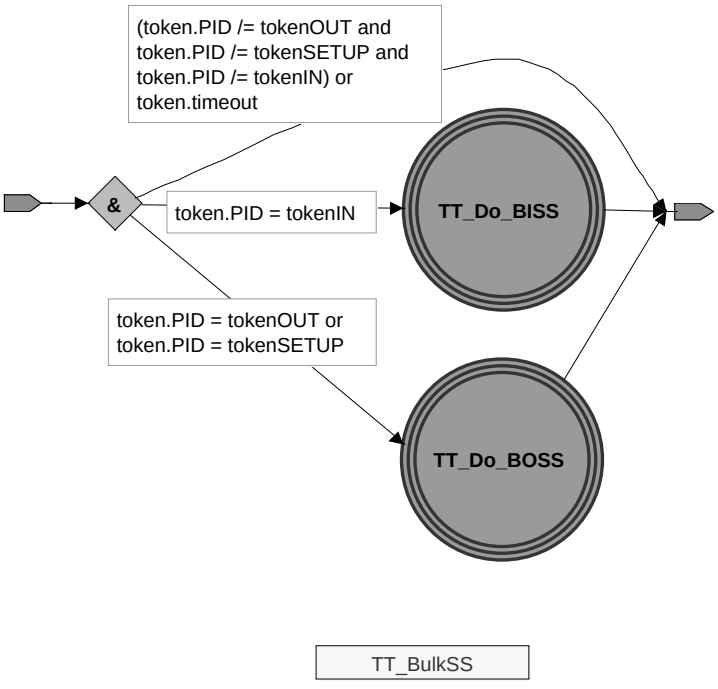


Figure 11-42. TT_BulkSS

Microchip Ex. 1018, p. 385
Microchip v. Aptiv
IPR2024-00494

11.16.2.1.4 TT_BulkCS State Machine

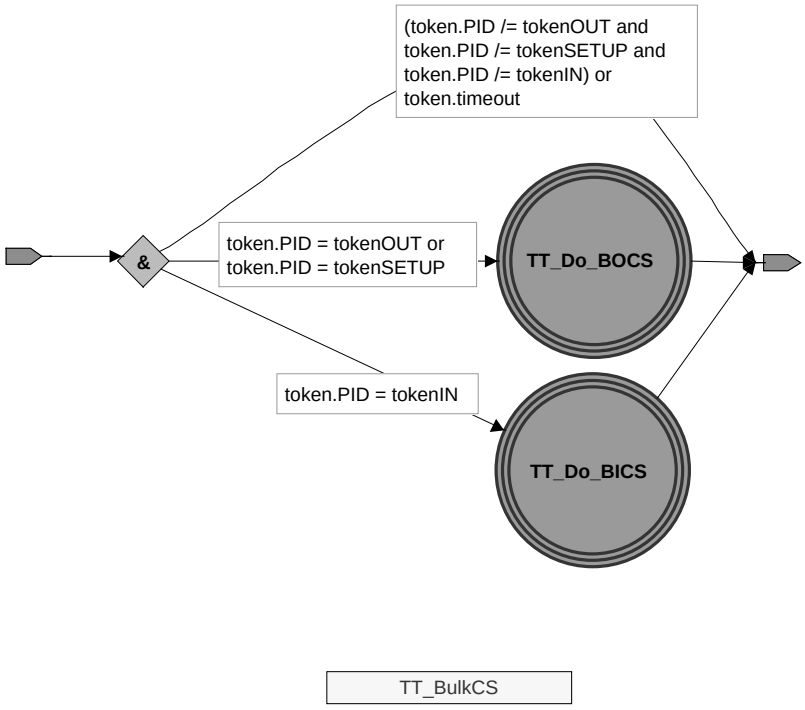


Figure 11-43. TT_BulkCS

11.16.2.1.5 TT_IntSS State Machine

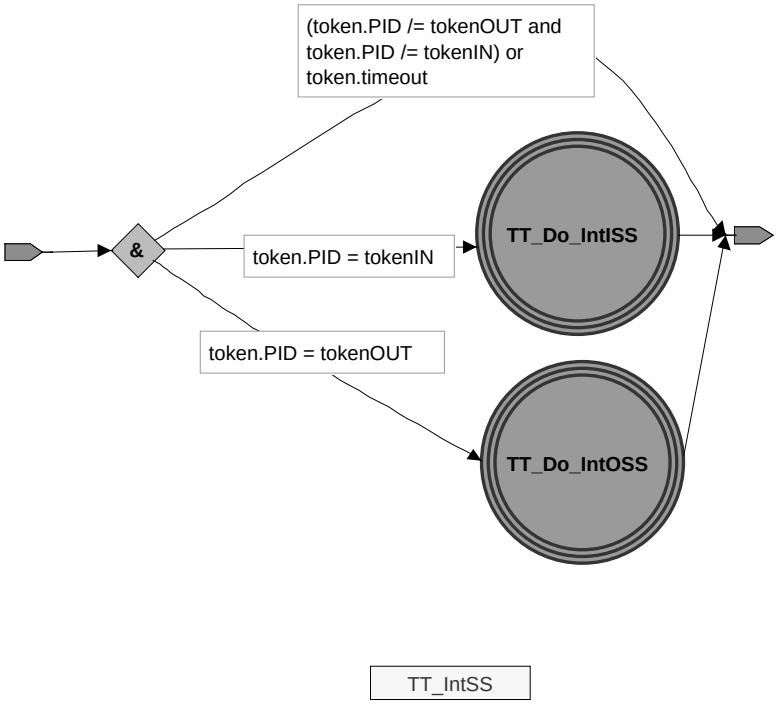


Figure 11-44. TT_IntSS

Microchip Ex. 1018, p. 386
Microchip v. Aptiv
IPR2024-00494

11.16.2.1.6 TT_IntCS State Machine

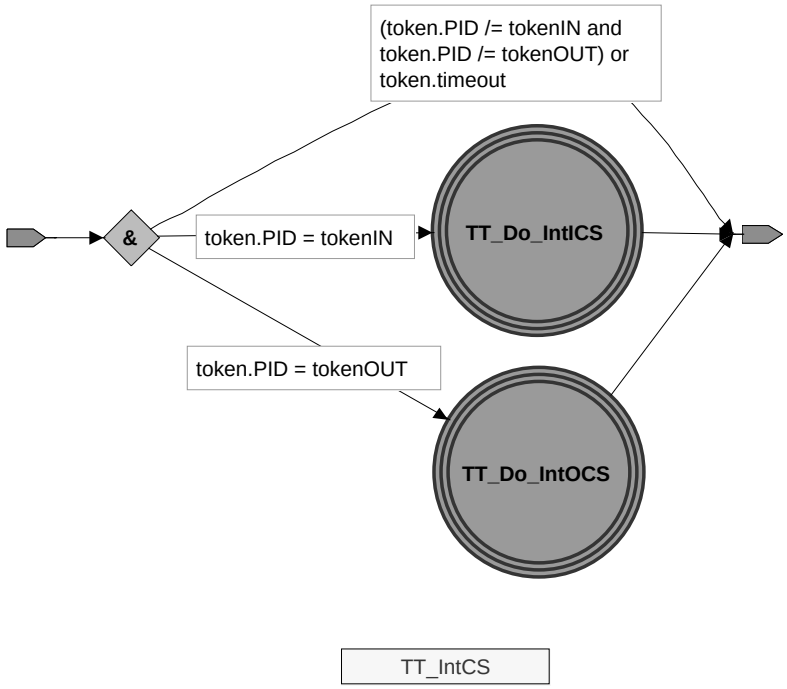


Figure 11-45. TT_IntCS

11.16.2.1.7 TT_IsochSS State Machine

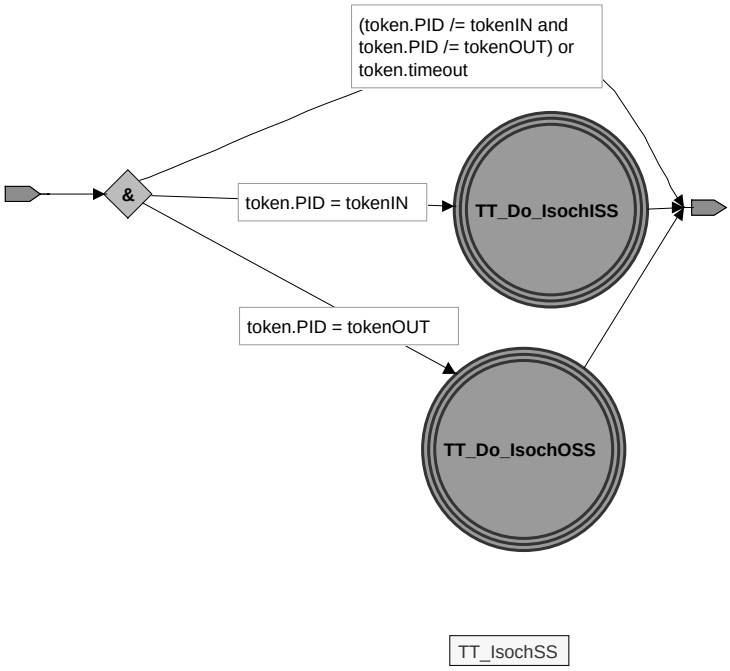


Figure 11-46. TT_IsochSS

Microchip Ex. 1018, p. 387
Microchip v. Aptiv
IPR2024-00494

11.17 Bulk/Control Transaction Translation Overview

Each TT must have at least two bulk/control transaction buffers. Each buffer holds the information for a start- or complete-split transaction and represents a single full-/low-speed transaction that is awaiting (or has completed) transfer on the downstream bus. The buffer is used to hold the transaction information from the start-split (and data for an OUT) and then the handshake/result of the full-/low-speed transaction (and data for an IN). This buffer is filled and emptied by split transactions from the high-speed bus via the high-speed handler. The buffer is also updated by the full-/low-speed handler while the transaction is in progress on the downstream bus.

The high-speed handler must accept a start-split transaction from the host controller for a bulk/control endpoint whenever the high-speed handler has appropriate space in a bulk/control buffer.

The host controller attempts a start-split transaction according to its bulk/control high-speed transaction schedule. As soon as the high-speed handler responds to a complete-split transaction with the results from the corresponding buffer, the next start-split for some (possibly other) full-/low-speed endpoint can be saved in the buffer.

There is no method to control the start-split transaction accepted next by the high-speed handler. Sequencing of start-split transactions is simply determined by available TT buffer space and the current state of the host controller schedule (e.g., which start-split transaction is next that the host controller tries as a normal part of processing high-speed transactions).

The host controller does not need to segregate split transaction bulk (or control) transactions from high-speed bulk (control) transactions when building its schedule. The host controller is required to track whether a transaction is a normal high-speed transaction or a high-speed split transaction.

The following sections describe the details of the transaction phases, flow sequences, and state machines for split transactions used to support full-/low-speed bulk and control OUT and IN transactions. There are only minor differences between bulk and control split transactions. In the figures, some areas are shaded to indicate that they do not apply for control transactions.

11.17.1 Bulk/Control Split Transaction Sequences

The state machine figures show the transitions required for high-speed split transactions for full-/low-speed bulk/control transfer types for a single endpoint. These figures must not be interpreted as showing any particular specific timing. They define the required sequencing behavior of different packets of a bulk/control split transaction. In particular, other high-speed or split transactions for other endpoints occur before or after these split transaction sequences.

Figure 11-47 shows a sample code algorithm that describes the behavior of the transitions labeled with *Is_new_SS*, *Is_old_SS* and *Is_no_space* shown in the figures for both bulk/control IN and OUT start-split transactions buffered in the TT for any endpoint. This algorithm ensures that the TT only buffers a single bulk/control split transaction for any endpoint. The complete-split protocol definition requires an endpoint has only a single result buffered in the TT at any time. Note that the “buffer match” test is different for bulk and control endpoints. A buffer match test for a bulk transaction must include the direction of the transaction in the test since bulk endpoints are unidirectional. A control transaction must not use direction as part of the match test.

Microchip Ex. 1018, p. 388
Microchip v. Aptiv
IPR2024-00494

```
procedure Compare_buffs IS
      variable match:boolean:=FALSE;
begin
--
--  Is_new_SS is true when BC_buff.status == NEW_SS
--  Is_old_SS is true when BC_buff.status == OLD_SS
--  Is_no_space is true when BC_buff.status == NO_SPACE
--
       -- Assume nospace and intialize index to 0.
       BC_buff.status := NO_SPACE;
       BC_buff.index  := 0;

       FOR i IN 0 to num_buffs-1 LOOP
           IF NOT match THEN
               -- Re-use buffer with same Device Address/End point.
               IF (token.endpt = cam(i).store.endpt AND
                   token.dev_addr = cam(i).store.dev_addr AND
                   ((token.direction = cam(i).store.direction AND
                     split.ep_type /= CONTROL) OR
                    split.ep_type = CONTROL)) THEN

                 -- If The buffer is already pending/ready this must be a retry.
                 IF (cam(i).match.state = READY OR cam(i).match.state = PENDING) THEN
                     BC_buff.status := OLD_SS;
                 ELSE
                     BC_buff.status := NEW_SS;
                 END IF;
                 BC_buff.index := i;
                 match := TRUE;

               -- Otherwise use the buffer if it's old.
               ELSIF (cam(i).match.state = OLD) THEN
                 BC_buff.status := NEW_SS;
                 BC_buff.index := i;
               END IF;
           END IF;
       END LOOP;

    end Compare_buffs;
```

Figure 11-47. Sample Algorithm for Compare_buffs

Figure 11-48 shows the sequence of packets for a start-split transaction for the full-/low-speed bulk OUT transfer type. The block labeled SSPLIT represents a split transaction token packet as described in Chapter 8. It is followed by an OUT token packet (or SETUP token packet for a control setup transaction). If the high-speed handler times out after the SSPLIT or OUT token packets, and does not receive the following OUT/SETUP or DATA0/1 packets, it will not respond with a handshake as indicated by the dotted line transitions labeled “se1” or “se2”. This causes the host to subsequently see a transaction error (timeout) (labeled “se2” and indicated with a dashed line). If the high-speed handler receives the DATA0/1 packet and it fails the CRC check, it takes the transition “se2” which causes the host to timeout and follow the “se2” transition.

Microchip Ex. 1018, p. 389
Microchip v. Aptiv
IPR2024-00494

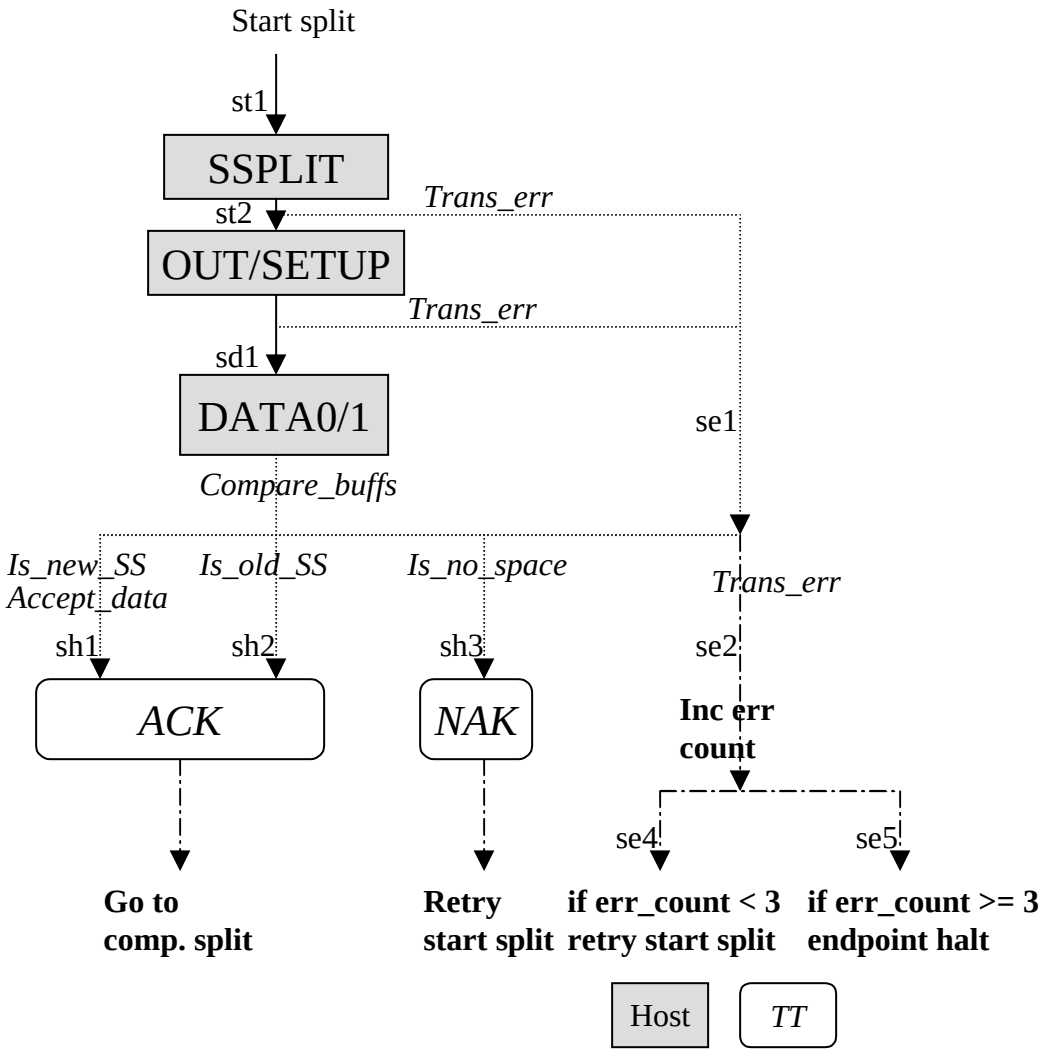


Figure 11-48. Bulk/Control OUT Start-split Transaction Sequence

The host must keep retrying the start-split for this endpoint until the err_count reaches three for this endpoint before continuing on to some other start-split for this endpoint. However, the host can issue other start-splits for other endpoints before it retries the start-split for this endpoint. The err_count is used to count how many errors have been experienced during attempts to issue a particular transaction for a particular endpoint.

If there is no space in the transaction buffers to hold the start-split, the high-speed handler responds with a NAK via transition “sh3”. This will cause the host to retry this start-split at some future time based on its normal schedule. The host does not increase its err_count for a NAK handshake response. Once the host has received a NAK response to a start-split, it can skip other start-splits for this TT for bulk/control endpoints until it finishes a bulk/control complete-split.

If there is buffer space for the start-split, the high-speed handler takes transition “sh1” and responds with an ACK. This tells the host it must try a complete-split the next time it attempts to process a transaction for this full-/low-speed endpoint. After receiving an ACK handshake, the host must not issue a further start-split for this endpoint until the corresponding complete-split has been completed.

If the high-speed handler already has a start-split for this full-/low-speed endpoint pending or ready, it follows transition “sh2” and also responds with an ACK, but ignores the data. This handles the case where

Microchip Ex. 1018, p. 390
Microchip v. Aptiv
IPR2024-00494

an ACK handshake was smashed and missed by the host controller and now the host controller is retrying the start-split; e.g., a high-speed handler transition of “sh1” but a host transition of “se2”.

In the host controller error cases, the host controller implements the “three strikes and you’re out” mechanism. That is, it increments an error count (err_count) and, if the count is less than three (transition “se4”), it will retry the transaction. If the err_count is greater or equal to three (transition “se5”), the host controller does endpoint halt processing and does not retry the transaction. If for some reason, a host memory or non-USB bus delay (e.g.,a system memory “hold off”) occurs that causes the transaction to not be completed normally, the err_count must not be incremented. Whenever a transaction completes normally, the err_count is reset to zero.

The high-speed handler in the TT has no immediate knowledge of what the host sees, so the “se2”, “se4”, and “se5” transitions show only host visibility.

This packet flow sequence showing the interactions between the host and hub is also represented by host and high-speed handler state machine diagrams in the next section. Those state machine diagrams use the same labels to correlate transitions between the two representations of the split transaction rules.

Figure 11-49 shows the corresponding flow sequence for the complete-split transaction for the full-/low-speed bulk/control OUT transfer type. The notation “ready/x” or “old/x” indicates that the transaction status of the split transaction is any of the ready or old states. After a full-/low-speed transaction is run on the downstream bus, the transaction status is updated to reflect the result of the transaction. The possible result status is: nak, stall, ack. The “x” means any of the NAK, ACK, STALL full-/low-speed transaction status results. Each status result reflects the handshake response from the full-/low-speed transaction.

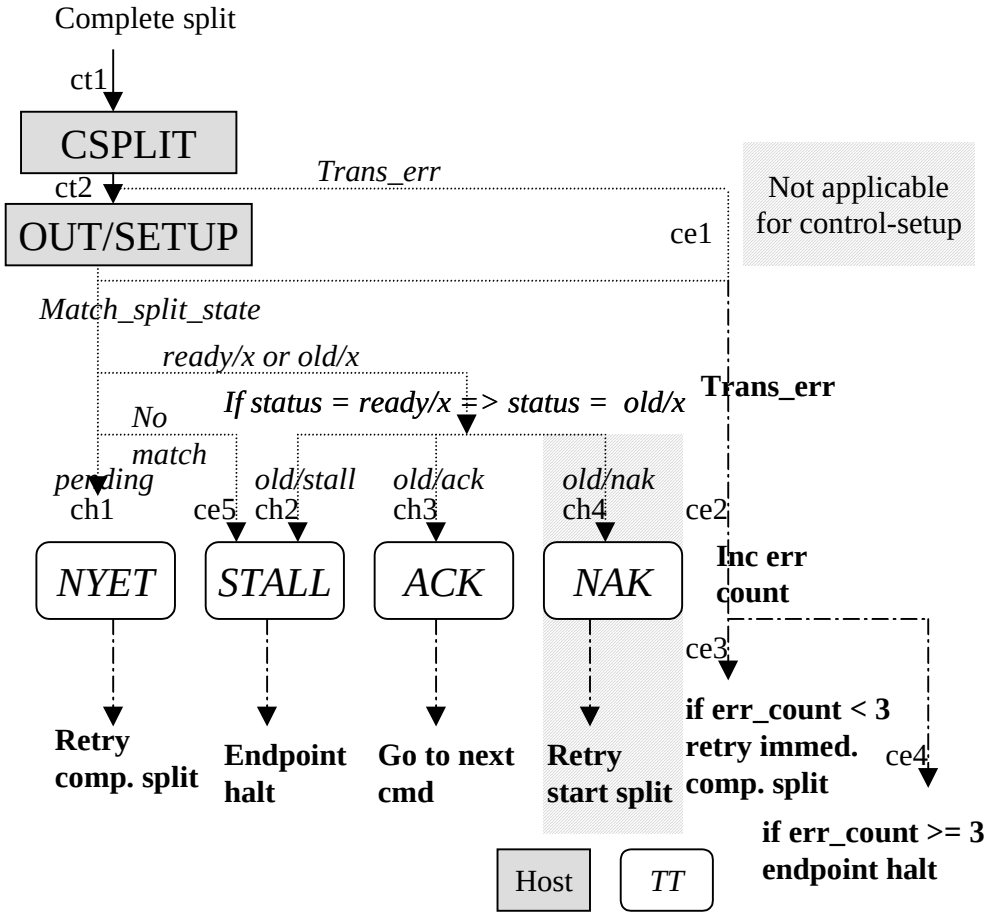


Figure 11-49. Bulk/Control OUT Complete-split Transaction Sequence

Microchip Ex. 1018, p. 391
Microchip v. Aptiv
IPR2024-00494

There is no timeout response status for a transaction because the full-/low-speed handler must perform a local retry of a full-/low-speed bulk or control transaction that experiences a transaction error. It locally implements a “three strikes and you’re out” retry mechanism. This means that the full-/low-speed transaction will resolve to one of a NAK, STALL or ACK handshake results. If the transaction experiences a transaction error three times, the full-/low-speed handler will reflect this as a stall status result. The full-/low-speed handler must not do a local retry of the transaction in response to an ACK, NAK, or STALL handshake.

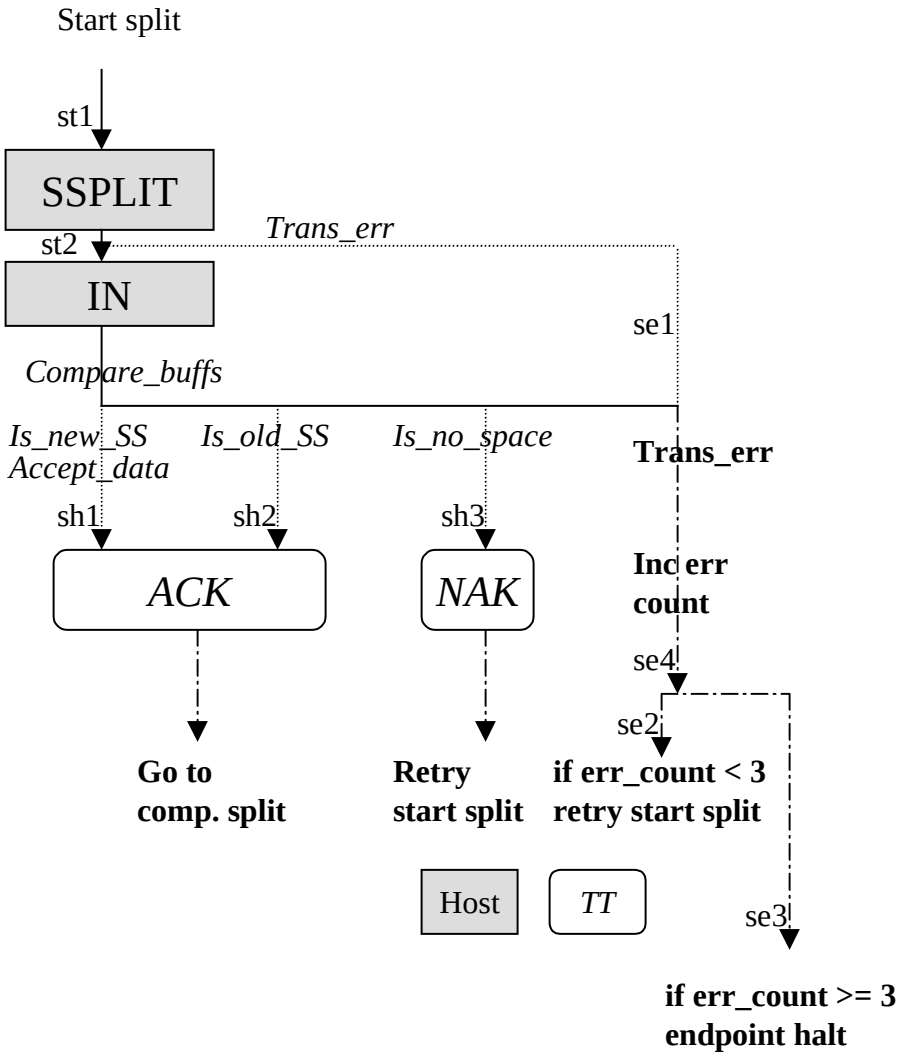


Figure 11-50. Bulk/Control IN Start-split Transaction Sequence

If the high-speed handler receives the complete-split token packet (and the token packet) while the full-/low-speed transaction has not been completed (e.g., the transaction status is “pending”), the high-speed handler responds with a NYET handshake. This causes the host to retry the complete-split for this endpoint some time in the future.

If the high-speed handler receives a complete-split token packet (and the token packet) and finds no local buffer with a corresponding transaction, the TT responds with a STALL to indicate a protocol violation.

Once the full-/low-speed handler has finished a full-/low-speed transaction, it changes the transaction status from pending to ready and saves the transaction result. This allows the high-speed handler to respond to the complete-split transaction with something besides NYET. Once the high-speed handler has seen a

Microchip Ex. 1018, p. 392
Microchip v. Aptiv
IPR2024-00494

complete-split, it changes the transaction status from ready/x to old/x. This allows the high-speed handler to reuse its local buffer for some other bulk/control transaction after this complete-split is finished.

If the host times out the transaction or does not receive a valid handshake, it immediately retries the complete-split before going on to any other bulk/control transactions for this TT. The normal "three strikes" mechanism applies here also for the host; i.e., the err_count is incremented. If for some reason, a host memory or non-USB bus delay (e.g., a system memory "hold off") occurs that causes the transaction to not be completed normally, the err_count must not be incremented.

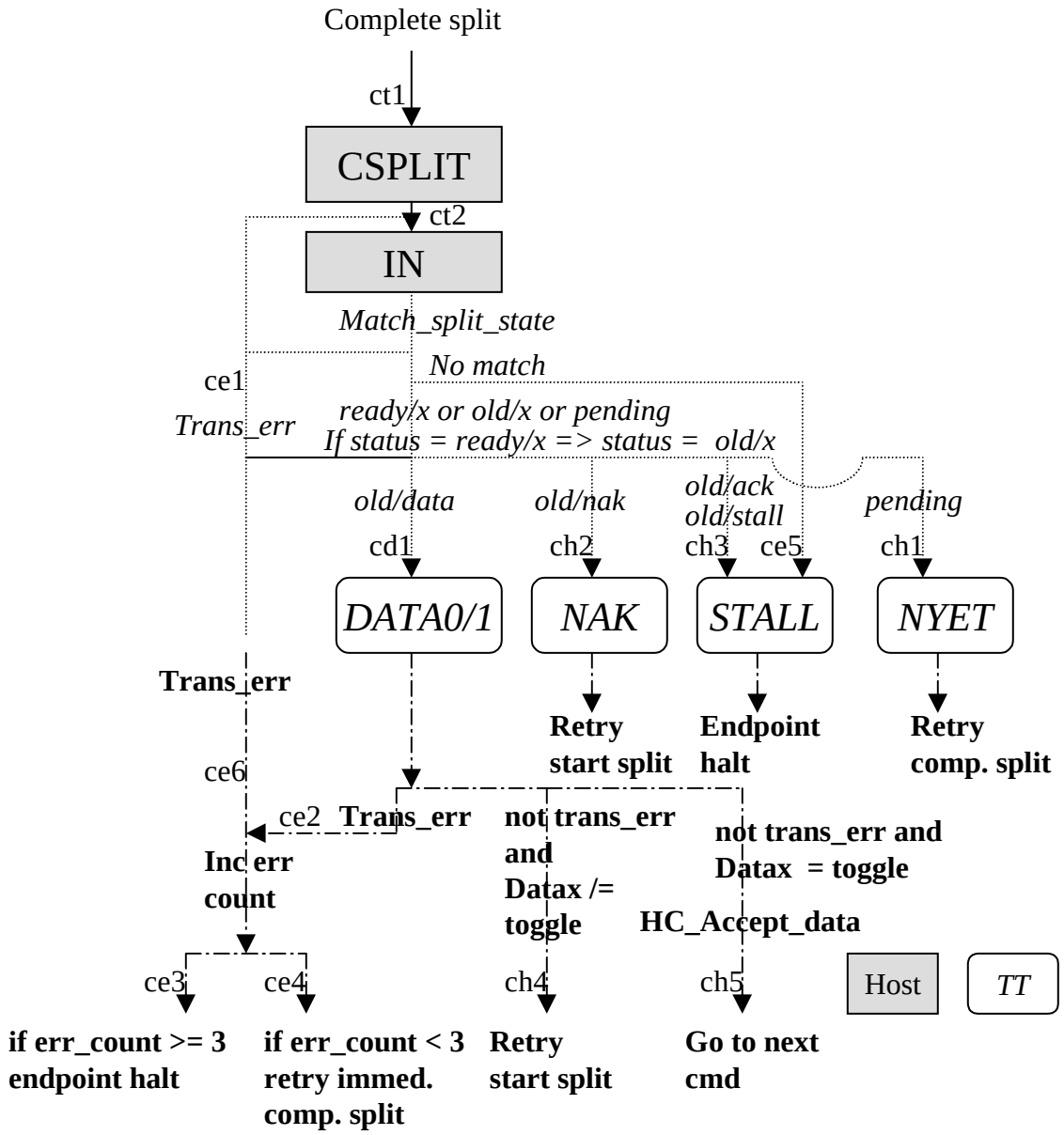


Figure 11-51. Bulk/Control IN Complete-split Transaction Sequence

If the host receives a STALL handshake, it performs endpoint halt processing and will not issue any more split transactions for this full-/low-speed endpoint until the halt condition is removed.

If the host receives an ACK, it records the results of the full-/low-speed transaction and advances to the next split transaction for this endpoint. The next transaction will be issued at some time in the future according to normal scheduling rules.

Microchip Ex. 1018, p. 393
Microchip v. Aptiv
IPR2024-00494

If the host receives a NAK, it will retry the start-split transaction for this endpoint at some time in the future according to normal scheduling rules. The host must not increment the err_count in this case.

The host must keep retrying the current start-split until the err_count reaches three for this endpoint before proceeding to the next split transaction for this endpoint. However, the host can issue other start-splits for other endpoints before it retries the start-split for this endpoint.

After the host receives a NAK, ACK, or STALL handshake in response to a complete-split transaction, it may subsequently issue a start-split transaction for the same endpoint. The host may choose to instead issue a start-split transaction for a different endpoint that is not awaiting a complete-split response.

The shaded case shown in the figure indicates that a control setup transaction should never encounter a NAK response since that is not allowed for full-/low-speed transactions.

Figure 11-50 and Figure 11-51 show the corresponding flow sequences for bulk/control IN split transactions.

11.17.2 Bulk/Control Split Transaction State Machines

The host and TT state machines for bulk/control IN and OUT split transactions are shown in the following figures. The transitions for these state machines are labeled the same as in the flow sequence figures.

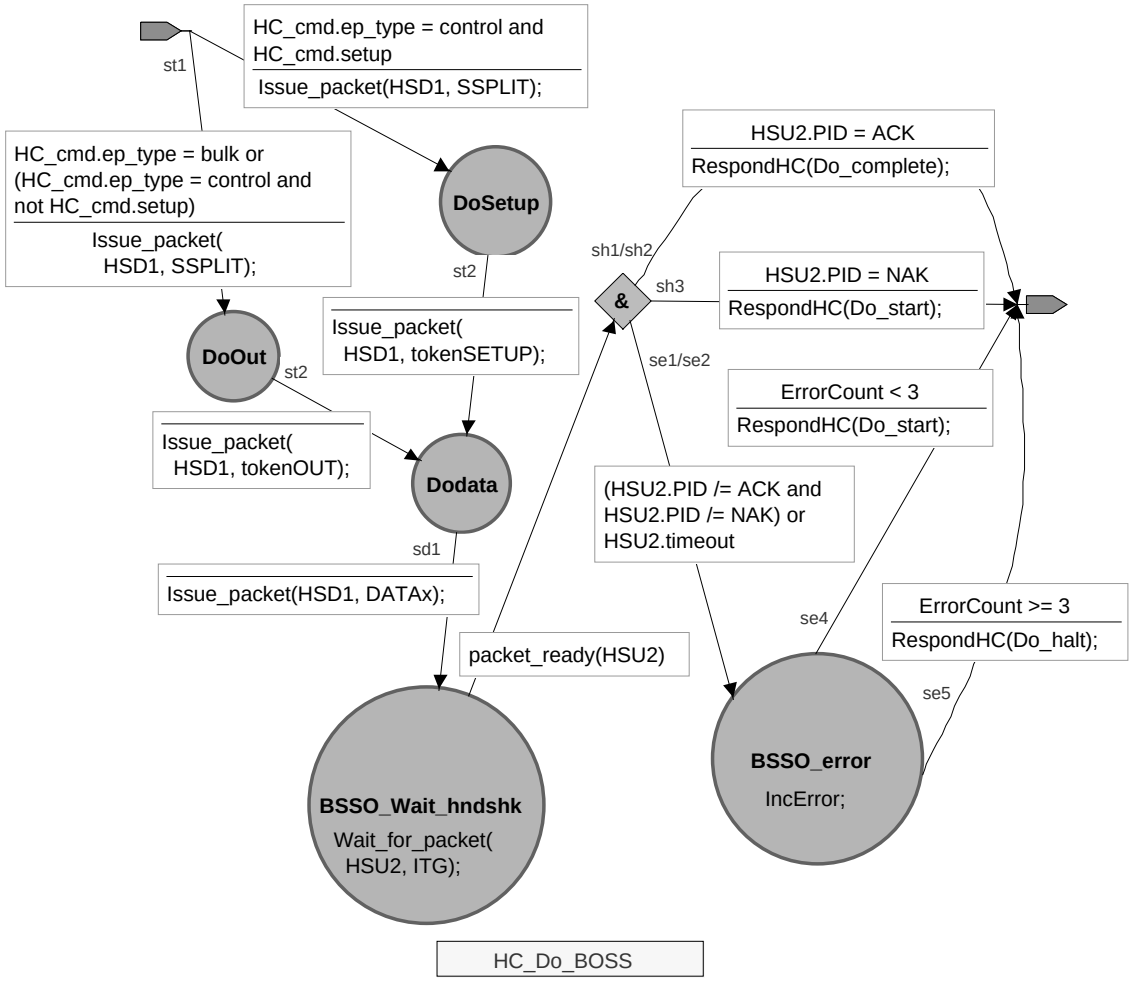


Figure 11-52. Bulk/Control OUT Start-split Transaction Host State Machine

Microchip Ex. 1018, p. 394
Microchip v. Aptiv
IPR2024-00494

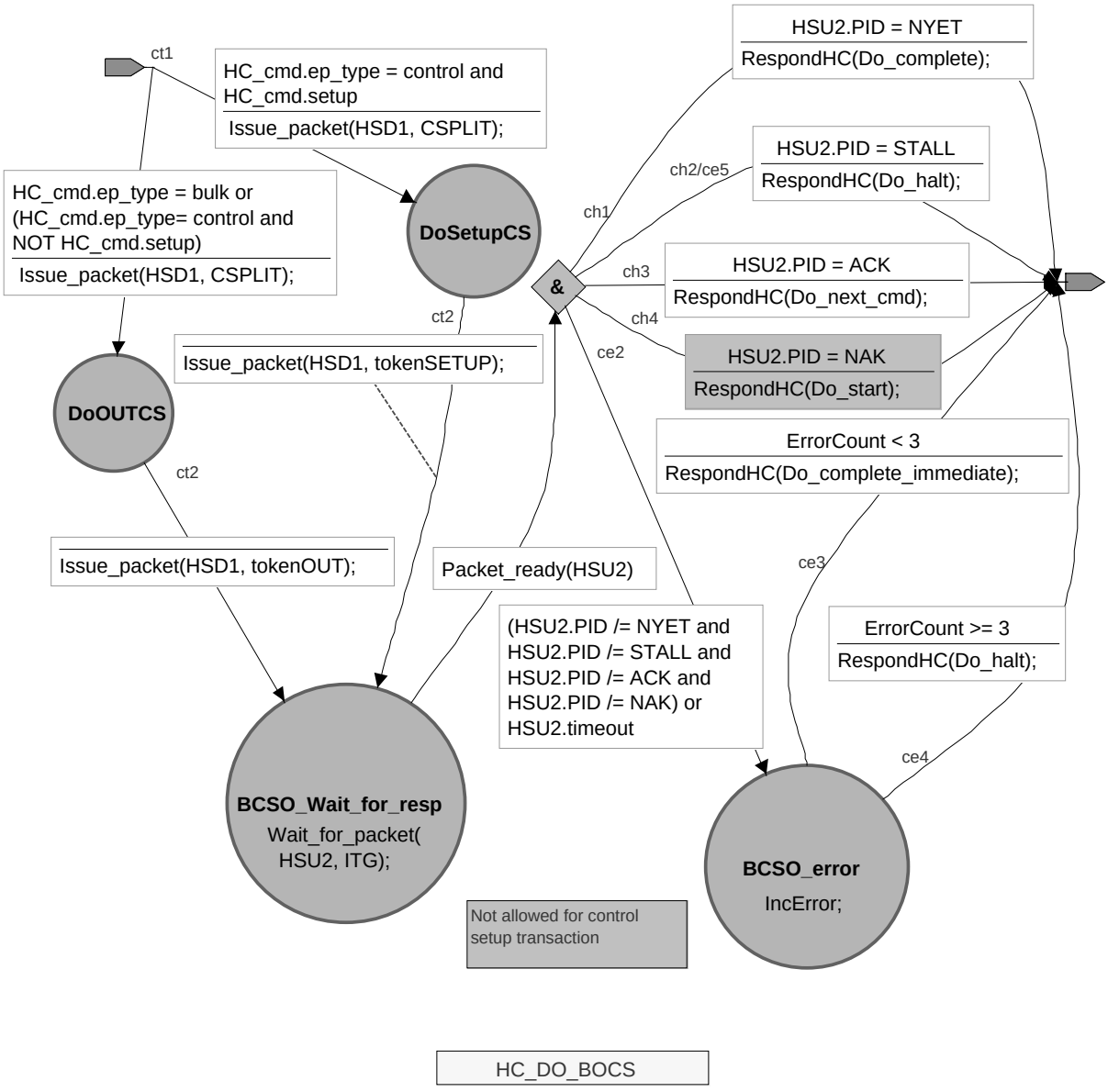


Figure 11-53. Bulk/Control OUT Complete-split Transaction Host State Machine

Microchip Ex. 1018, p. 395
Microchip v. Aptiv
IPR2024-00494

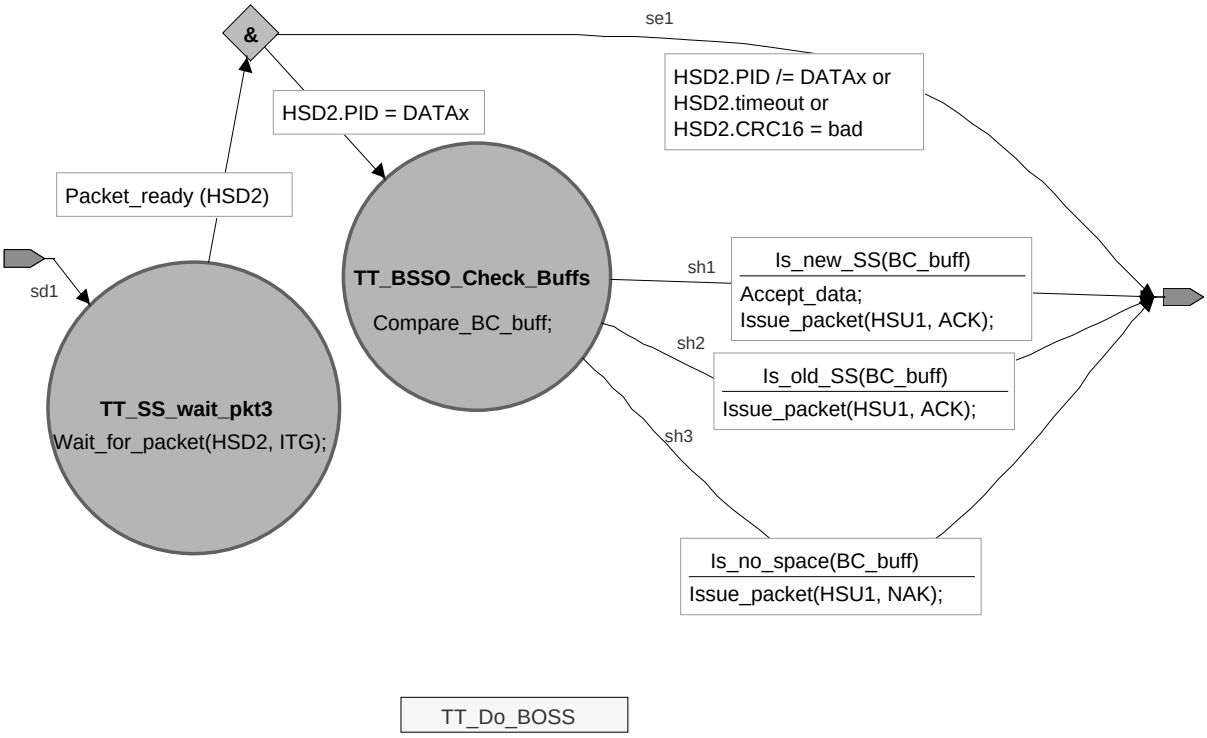


Figure 11-54. Bulk/Control OUT Start-split Transaction TT State Machine

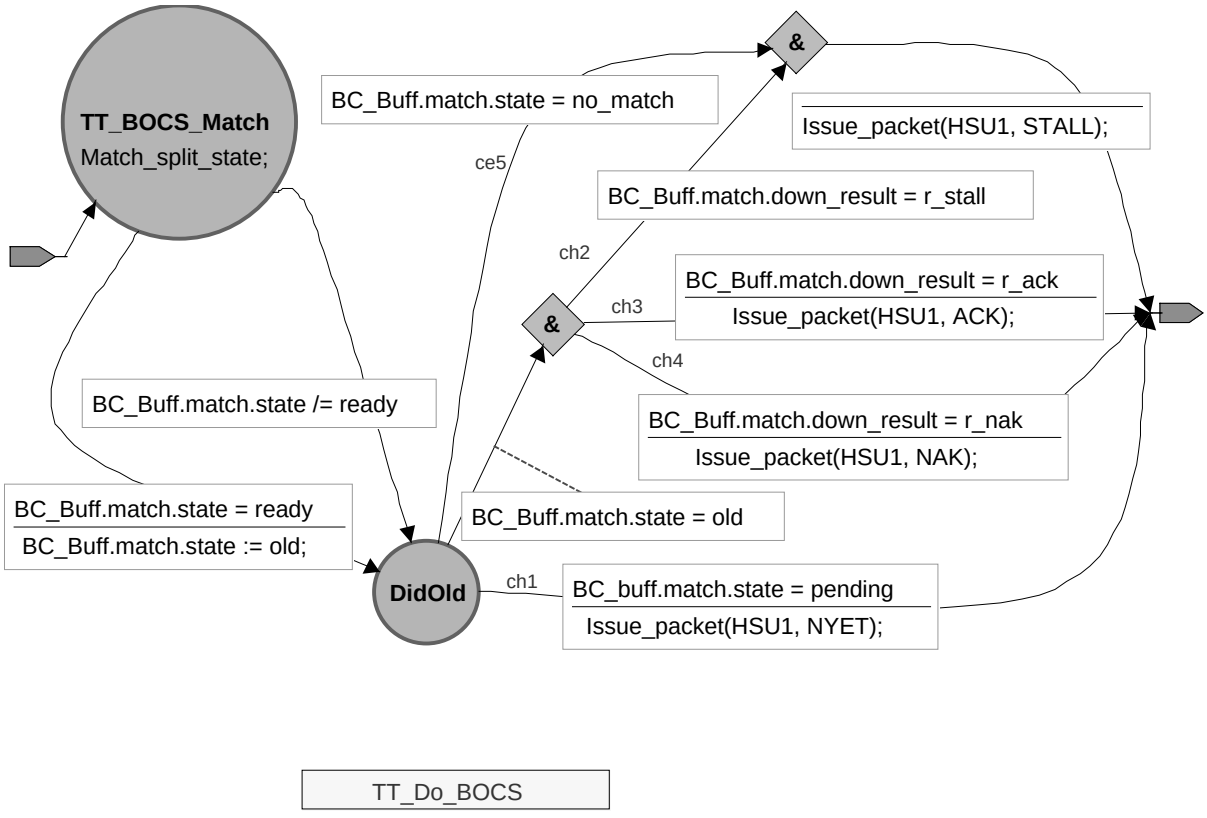


Figure 11-55. Bulk/Control OUT Complete-split Transaction TT State Machine

Microchip Ex. 1018, p. 396
Microchip v. Aptiv
IPR2024-00494

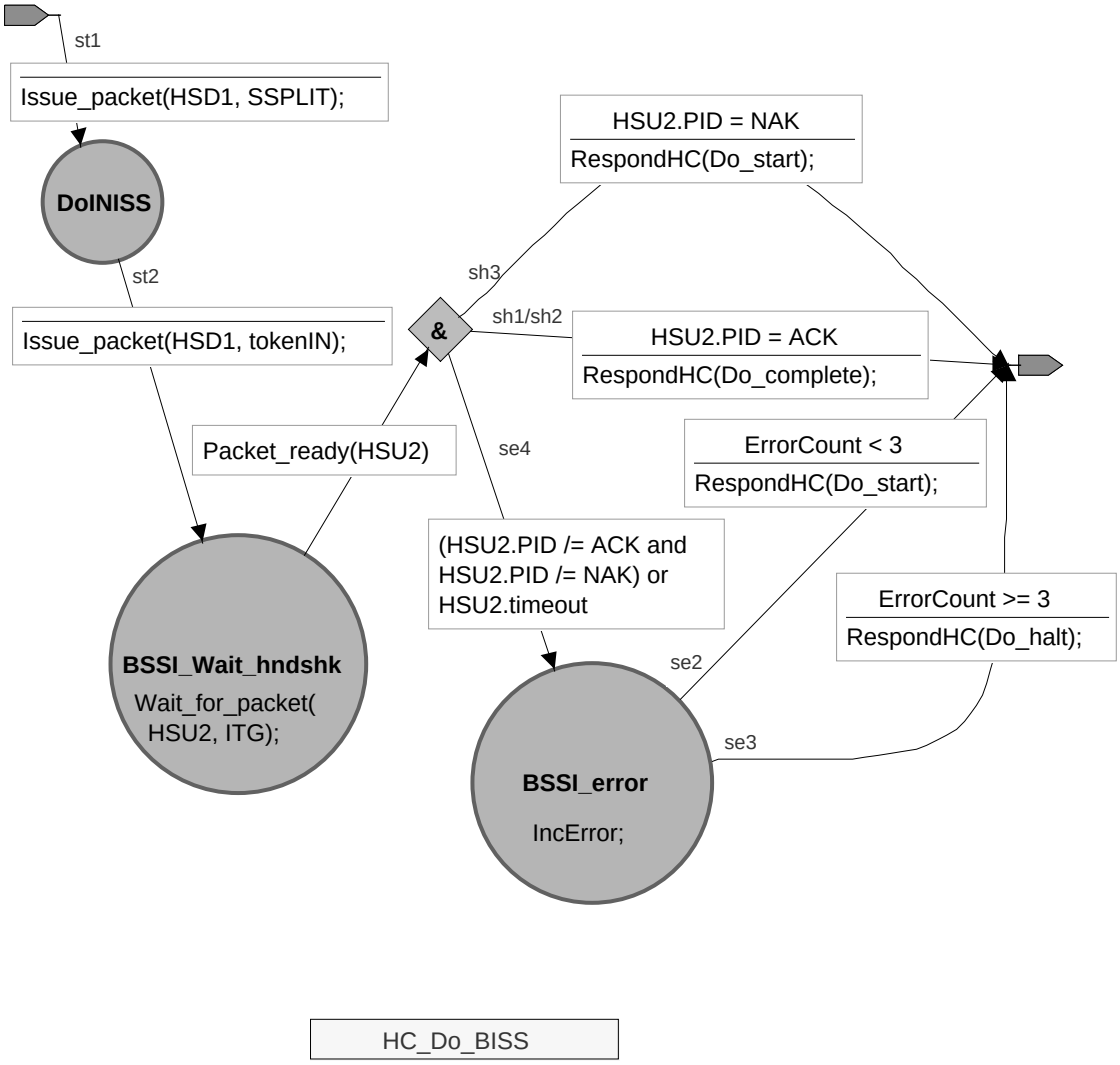


Figure 11-56. Bulk/Control IN Start-split Transaction Host State Machine

Microchip Ex. 1018, p. 397
Microchip v. Aptiv
IPR2024-00494

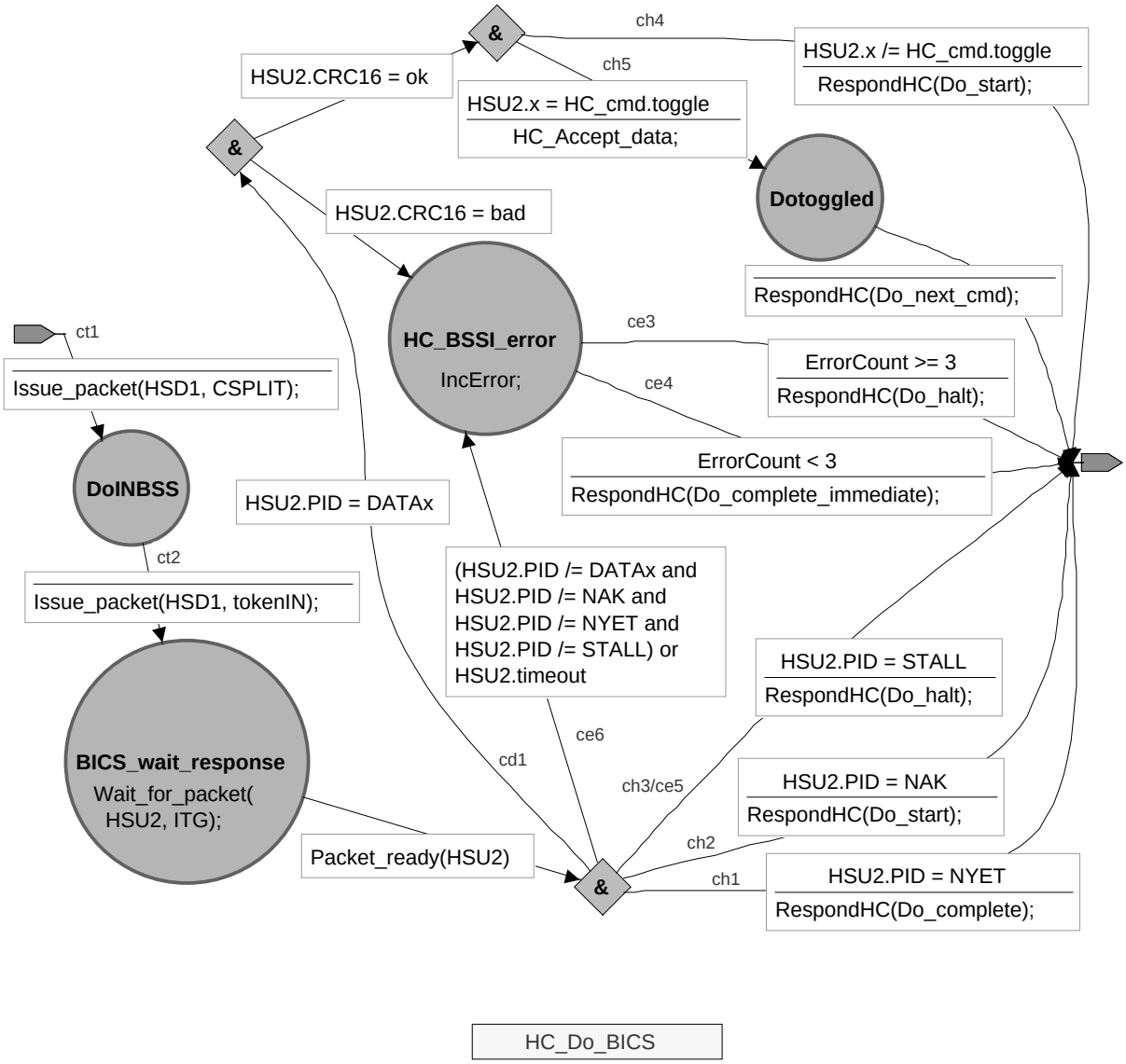


Figure 11-57. Bulk/Control IN Complete-split Transaction Host State Machine

Microchip Ex. 1018, p. 398
Microchip v. Aptiv
IPR2024-00494

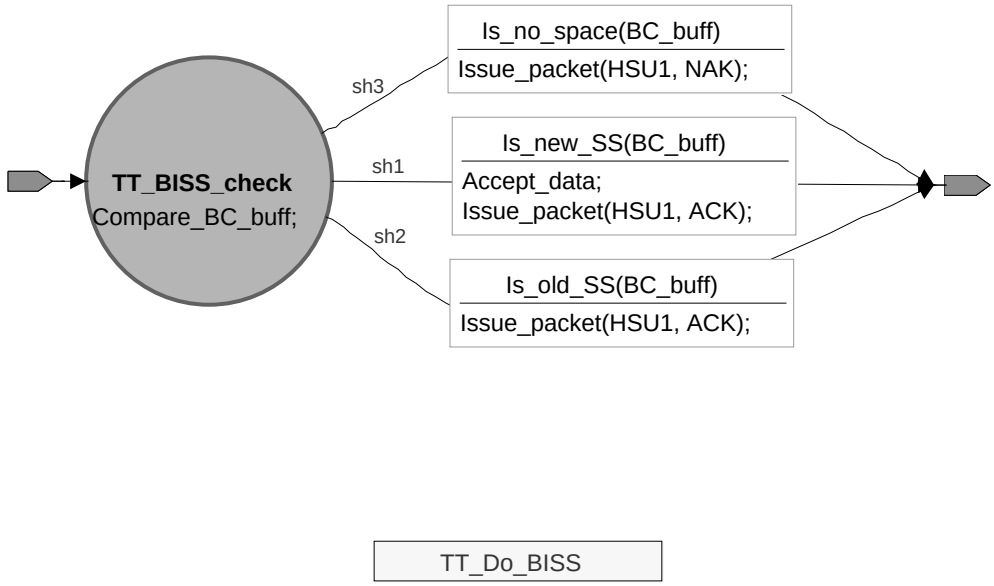


Figure 11-58. Bulk/Control IN Start-split Transaction TT State Machine

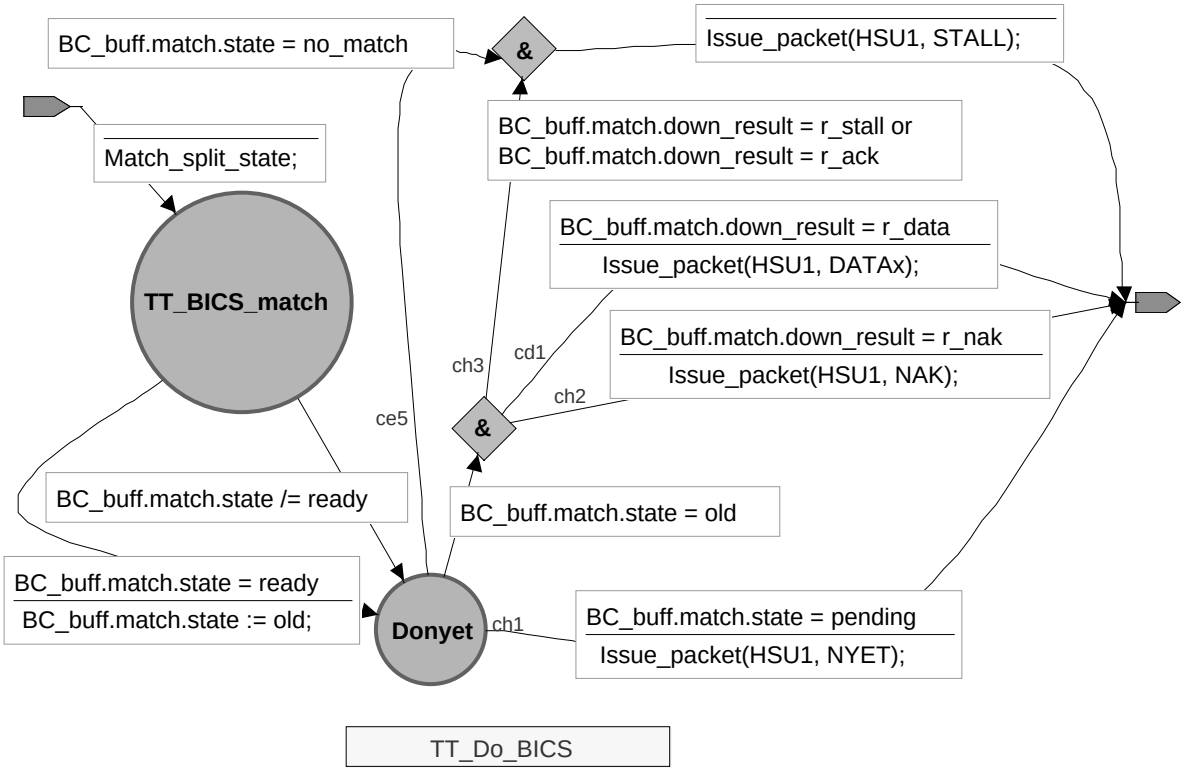


Figure 11-59. Bulk/Control IN Complete-split Transaction TT State Machine

11.17.3 Bulk/Control Sequencing

Once the high-speed handler has received a start-split for an endpoint and saved it in a local buffer, it responds with an ACK split transaction handshake. This tells the host controller to do a complete-split transaction next time this endpoint is polled.

Microchip Ex. 1018, p. 399
Microchip v. Aptiv
IPR2024-00494

As soon as possible (subject to scheduling rules described previously), the full-/low-speed handler issues the full-/low-speed transaction and saves the handshake status (for OUT) or data/handshake status (for IN) in the same buffer.

Some time later (according to the host controller schedule), this endpoint will be polled for the complete-split transaction. The high-speed handler responds to the complete-split to return the full-/low-speed endpoint status for this transaction (as recorded in the buffer). If the host controller polls for the complete-split transaction for this endpoint before the full-/low-speed handler has finished processing this transaction on the downstream bus, the high-speed handler responds with a NYET handshake. This tells the host controller that the transaction is not yet complete. In this case, the host controller will retry the complete-split again at some later time.

When the full-/low-speed handler finally finishes the full-/low-speed transaction, it saves the data/status in the buffer to be ready for the next host controller complete-split transaction for this endpoint. When the host sends the complete-split, the high-speed handler responds with the indicated data/status as recorded in the buffer. The buffer transaction status is updated from ready to old so the high-speed handler is ready for either a retry or a new start-split transaction for this (or some other) full-/low-speed endpoint.

If there is an error on the complete-split transaction, the host controller will retry the complete-split transaction for this bulk/control endpoint “immediately” before proceeding to some other bulk/control split transaction. The host controller may issue other periodic split transactions or other non-split transactions before doing this complete-split transaction retry.

If there is a bulk/control transaction in progress on the downstream facing bus when the EOF time occurs, the TT must adhere to the definition in Section 11.3 for its behavior on the downstream facing bus. This will cause an increase in the error count for this transaction. The normal retry rules will determine if the transaction will be retried or not on the downstream facing bus.

11.17.4 Bulk/Control Buffering Requirements

The TT must provide at least two transactions of non-periodic buffering to allow the TT to deliver maximum full-/low-speed throughput on a downstream bus when the high-speed bus is idle.

As the high-speed bus becomes busier, the throughput possible on downstream full-/low-speed buses will decrease.

A TT may provide more than two transactions of non-periodic buffering and this can improve throughput for downstream buses for specific combinations of device configurations.

11.17.5 Other Bulk/Control Details

When a bulk/control split transaction fails, it can leave the associated TT transaction buffer in a busy (ready/x) state. This buffer state will not allow the buffer to be reused for other bulk/control split transactions. Therefore, as part of endpoint halt processing for full-/low-speed endpoints connected via a TT, the host software must use the Clear_TT_Buffer request to the TT to ensure that the buffer is not in the busy state.

Appendix A shows examples of packet sequences for full-/low-speed bulk/control transactions and their relationship with start-splits and complete-splits in various normal and error conditions.

11.18 Periodic Split Transaction Pipelining and Buffer Management

There are requirements on the behavior of the host and the TT to ensure that the microframe pipeline correctly sequences full-/low-speed isochronous/interrupt transactions on downstream facing full-/low-speed buses. The host must determine the microframes in which a start-split and complete-split transaction must be issued on high-speed to correctly sequence a corresponding full-/low-speed transaction on the downstream facing bus. This is called “scheduling” the split transactions.

Microchip Ex. 1018, p. 400
Microchip v. Aptiv
IPR2024-00494

In the following descriptions, the 8 microframes within each full-speed (1 ms.) frame are referred to as microframe Y_0, Y_1, Y_2, …, Y_7. This notation means that the first microframe of each full-speed frame is labeled Y_0. The second microframe is labeled Y_1, etc. The last microframe of each full-speed frame is labeled Y_7. The labels repeat for each full-speed frame.

This section describes details of the microframe pipeline that affect both full-speed isochronous and full-/low-speed interrupt transactions. Then the split transaction rules for interrupt and isochronous are described.

Bulk/control transactions are not scheduled with this mechanism. They are handled as described in the previous section.

11.18.1 Best Case Full-Speed Budget

A microframe of time allows at most 187.5 raw bytes of signaling on a full-speed bus. In order to estimate when full-/low-speed transactions appear on a downstream bus, the host must calculate a best case full-speed budget. This budget tracks in which microframes a full-/low-speed transaction appears. The best case full-speed budget assumes that 188 full-speed bytes occur in each microframe. Figure 11-60 shows how a 1 ms frame subdivided into microframes of budget time. This estimate assumes that no bit stuffing occurs to lengthen the time required to move transactions over the bus.

The maximum number of bytes in a 1 ms frame is calculated as:

1157 maximum_periodic_bytes_per_frame = 12 Mb/s * 1 ms / 8 bits_per_byte *

6 data_bits / 7 bit-stuffed_data_bits * 90% maximum_periodic_data_per_frame

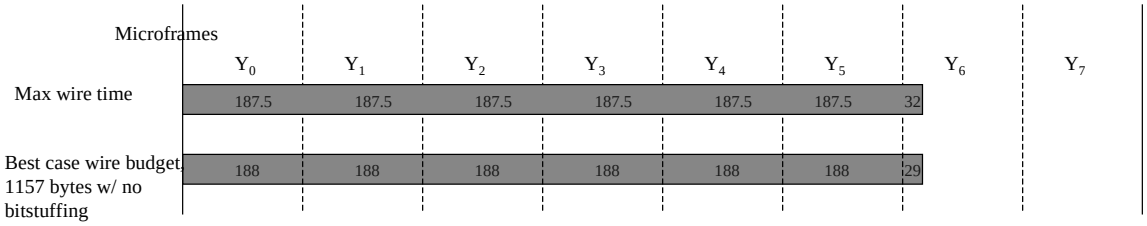


Figure 11-60. Best Case Budgeted Full-speed Wire Time With No Bit Stuffing

11.18.2 TT Microframe Pipeline

The TT implements a microframe pipeline of split transactions in support of a full-/low-speed bus. Start-split transactions are scheduled a microframe before the earliest time that their corresponding full-/low-speed transaction is expected to start. Complete-split transactions are scheduled in microframes that the full-/low-speed transaction can finish.

When a full-/low-speed device is attached to the bus and configured, the host assigns some time on the full-/low-speed bus at some budgeted time, based on the endpoint requirements of the configured device.

The effects of bit stuffing can delay when the full-/low-speed transaction actually runs. The results of other previous full-/low-speed transactions can cause the transaction to run earlier or later on the full-/low-speed bus.

The host always uses the maximum data payload size for a full-/low-speed endpoint in doing its budgeting. It does not attempt to schedule the actual data payloads that may be used in specific transactions to full-/low-speed endpoints. The host must include the maximum duration interpacket gap, bus turnaround times, and “TT think time”. The TT requires some time to proceed to the next full-/low-speed transaction. This time is called the “TT think time” and is specified in the hub descriptor field *wHubCharacteristics* bit 5 and 6.

Microchip Ex. 1018, p. 401
Microchip v. Aptiv
IPR2024-00494

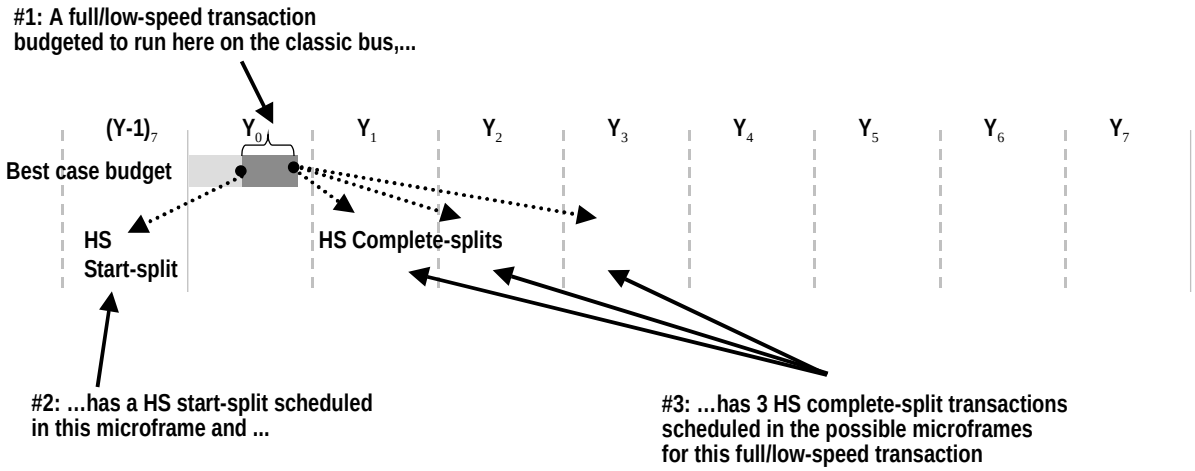


Figure 11-61. Scheduling of TT Microframe Pipeline

Figure 11-61 shows an example of a new endpoint that is assigned some portion of a full-/low-speed frame and where its start- and complete-splits are generally scheduled. The act of assigning some portion of the full-/low-speed frame to a particular transaction is called determining the budget for the transaction. More precise rules for scheduling and budgeting are presented later. The start-split for this example transaction is scheduled in microframe $Y\text{-}1_7$, the transaction is budgeted to run in microframe Y_0, and complete-splits are scheduled for microframes Y_1, Y_2, and Y_3. Section 11.18.4 describes the scheduling rules more completely.

The host must determine precisely when start- and complete- splits are scheduled to avoid overruns or underruns in the periodic transaction buffers provided by the TT.

11.18.3 Generation of Full-speed Frames

The TT must generate SOFs on the full-speed bus to establish the 1 ms frame clock within the defined jitter tolerances for full-speed devices. The TT has its own frame clock that is synchronized to the microframe SOFs on the high-speed bus. The SOF that reflects a change in the frame number it carries is identified as the zeroth microframe SOF. The zeroth high-speed microframe SOF corresponds to the full-speed SOF on the TT's downstream facing bus. The TT must adhere to all timing/jitter requirements of a host controller related to frames as defined in other parts of this specification.

The TT must stop issuing full-speed SOFs after it detects 250 μs of high-speed idle. This is required to ensure that the full-/low-speed downstream facing bus enters suspend no more than 250 μs after the high-speed bus enters suspend.

The TT must generate a full-speed SOF on the downstream facing bus based on its frame timer. The generation of the full-speed SOF must occur within +/-3 full-speed bit time from the occurrence of the zeroth high-speed SOF. See Section 11.22.1 for more information about TT SOF generation.

11.18.4 Host Split Transaction Scheduling Requirements

Scheduling of split transactions is done by the host (typically in software) based on a best-case estimate of how the full-/low-speed transactions can be run on the downstream facing bus. This best-case estimate is called the best case budget. The host is free to issue the split transactions anytime within the scheduled microframe, but each split transaction must be issued sometime within the scheduled microframe. This description of the scheduling requirements applies to the split transactions for a single full-/low-speed transaction at a time.

1. The host must never schedule a start-split in microframe Y_6. Some error conditions may result in the host controller erroneously issuing a start-split in this microframe. The TT response to this start-split is undefined.

Microchip Ex. 1018, p. 402
Microchip v. Aptiv
IPR2024-00494

2. The host must compute the start-split schedule by determining the best case budget for the transaction and:

   a. For isochronous OUT full-speed transactions, for each microframe in which the transaction is budgeted, the host must schedule a 188 (or the remaining data size) data byte start-split transaction. The start-split transaction must be scheduled in the microframe before the data is budgeted to begin on the full-speed bus. The start-split transactions must use the beginning/middle/end/all split transaction token encodings corresponding to the piece of the full-speed data that is being sent on the high-speed bus. For example, if only a single start-split is required, an "all" encoding is used. If multiple start-splits are required, a "beginning" encoding is used for the first start-split and an "end" encoding is used for the final start-split. If there are more than two start-splits required, the additional start-splits that are not the first or last use a "middle" encoding. A zero length full-speed data payload must only be scheduled with an "all" start-split. A start-split transaction for a beginning, middle, or end start-split must always have a non-zero length data payload.
   Figure 11-62 shows an example of an isochronous OUT that would appear to have budgeted a zero length data payload in a start-split (end). This example instead must be scheduled with a start-split(all) transaction.

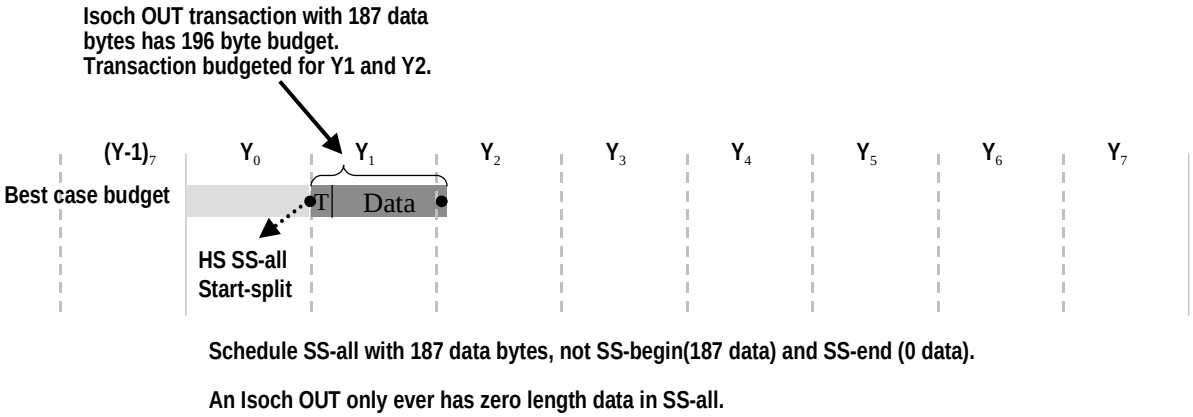


Figure 11-62. Isochronous OUT Example That Avoids a Start-split-end With Zero Data

   b. For isochronous IN and interrupt IN/OUT full-/low-speed transactions, a single start-split must be scheduled in the microframe before the transaction is budgeted to start on the full-/low-speed bus.

3. The host never schedules more than one complete-split in any microframe for the same full-/low-speed transaction.

   a. For isochronous OUT full-speed transactions, the host must never schedule a complete-split. The TT response to a complete-split for an isochronous OUT is undefined.

   b. For interrupt IN/OUT full-/low-speed transactions, the host must schedule a complete-split transaction in each of the two microframes following the first microframe in which the full-/low-speed transaction is budgeted. An additional complete-split must also be scheduled in the third following microframe unless the full-/low-speed transaction was budgeted to start in microframe Y_6. Figure 11-63 shows an example with only two complete-splits.

Microchip Ex. 1018, p. 403
Microchip v. Aptiv
IPR2024-00494

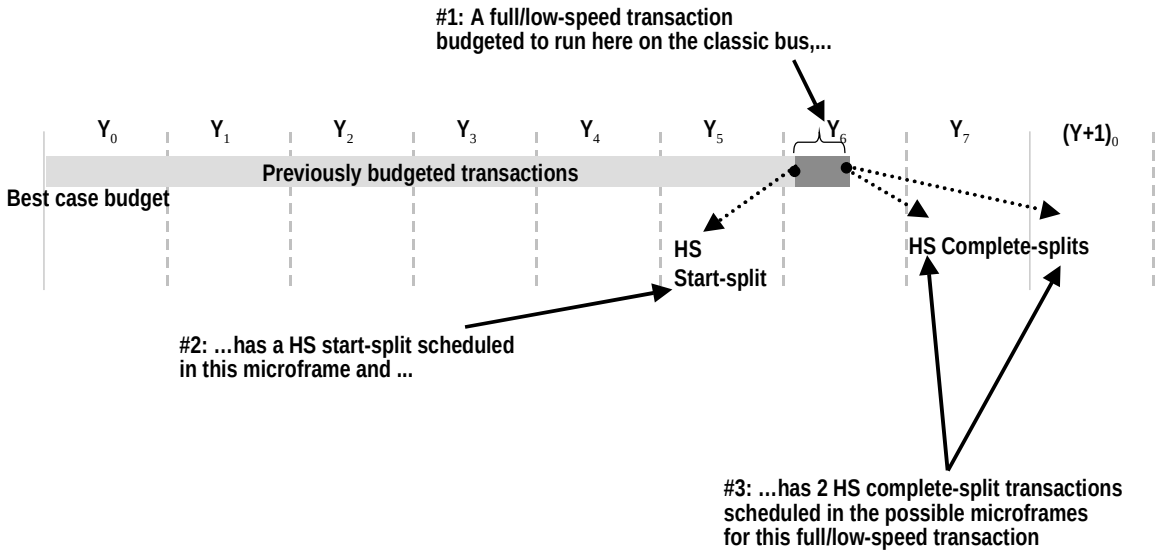


Figure 11-63. End of Frame TT Pipeline Scheduling Example

c. For isochronous IN full-speed transactions, for each microframe in which the full-speed transaction is budgeted, a complete-split must be scheduled for each following microframe. Also, determine the last microframe in which a complete-split is scheduled, call it L. If L is less than Y_6, schedule additional complete-splits in microframe L+1 and L+2.

   If L is equal to Y_6, schedule one complete-split in microframe Y_7. Also, schedule one complete-split in microframe Y_0 of the next frame, unless the full-speed transaction was budgeted to start in microframe Y_0.

   If L is equal to Y_7, schedule one complete-split in microframe Y_0 of the next frame, unless the full-speed transaction was budgeted to start in microframe Y_0. Figure 11-64 and Figure 11-65 show examples of the cases for L= Y_6 and L=Y_7.

Microchip Ex. 1018, p. 404
Microchip v. Aptiv
IPR2024-00494

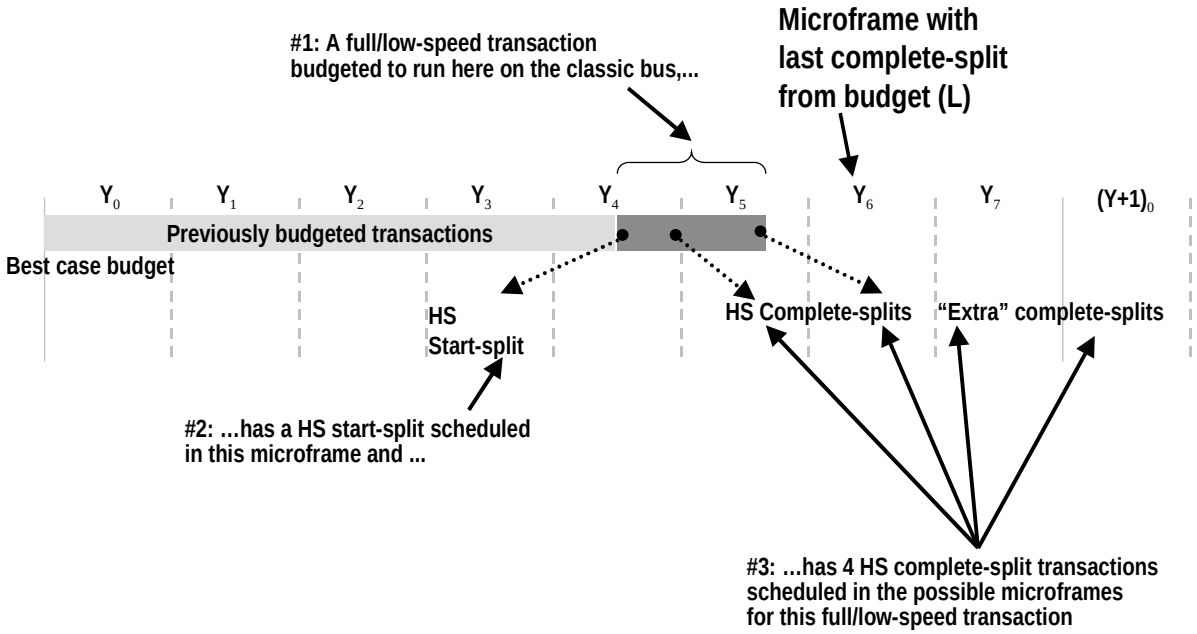


Figure 11-64. Isochronous IN Complete-split Schedule Example at L=Y_6

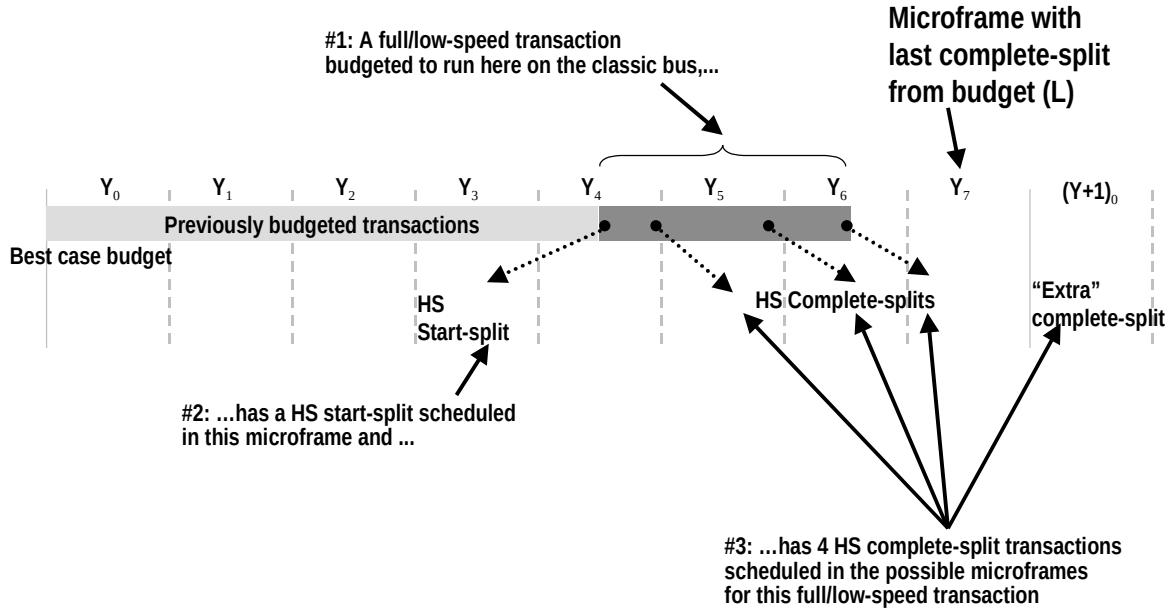


Figure 11-65. Isochronous IN Complete-split Schedule Example at L=Y_7

4. The host must never issue more than 16 start-splits in any high-speed microframe for any TT.
5. The host must only issue a split transaction in the microframe in which it was scheduled.
6. As precisely identified in the flow sequence and state machine figures, the host controller must immediately retry a complete-split after a high-speed transaction error ("trans_err").

Microchip Ex. 1018, p. 405
Microchip v. Aptiv
IPR2024-00494

The “pattern” of split transactions scheduled for a full-/low-speed transaction can be computed once when each endpoint is configured. Then the pattern does not change unless some change occurs to the collection of currently configured full-/low-speed endpoints attached via a TT.

Finally, for all periodic endpoints that have split transactions scheduled within a particular microframe, the host must issue complete-split transactions in the same relative order as the corresponding start-split transactions were issued.

11.18.5 TT Response Generation

The approach used for full-speed isochronous INs and interrupt INs/OUTs ensures that there is always an opportunity for the TT to return data/results whenever it has something to return from the full-/low-speed transaction. Then whenever the full-/low-speed handler starts the full-/low-speed transaction, it simply accumulates the results in each microframe and then returns it in response to a complete-split from the host. The TT acts similar to an isochronous device in that it uses the microframe boundary to "carve up" the full-/low-speed data to be returned to the host. The TT does not do any computation on how much data to return at what time. In response to the "next" high-speed complete-split, the TT simply returns the endpoint data it has received from the full-/low-speed bus in a microframe.

Whenever the TT has data to return in response to a complete-split for an interrupt full-/low-speed or isochronous full-speed transaction, it uses either a DATA0/1 or MDATA for the data packet PID.

If the full-/low-speed handler completes the full-/low-speed isochronous/interrupt IN transaction during a microframe with a valid CRC16, it uses the DATA0/1 PID for the data packet of the complete-split transaction. This indicates that this is the last data of the full-/low-speed transaction. A DATA0 PID is always used for isochronous transactions. For interrupt transactions, a DATA0/1 PID is used corresponding to the full-/low-speed data packet PID received.

If the full-/low-speed handler completes the full-/low-speed isochronous/interrupt IN transaction during a microframe with a bad CRC16, it uses the ERR response to the complete-split transaction and does not return the data received from the full-/low-speed device.

If the TT is still receiving data on the downstream facing bus at the microframe boundary, the TT will respond with either an MDATA PID or a NYET for the corresponding complete-split. If the TT has received more than two bytes of the data field of the full-/low-speed data packet, it will respond with an MDATA PID. Further, the data packet that will be returned in the complete-split must contain the data received from the full-/low-speed device minus the last two bytes. The last two bytes must not be included since they could be the CRC16 field, but the TT will not know this until the next microframe. The CRC16 field received from the full-/low-speed device is never returned in a complete-split data packet for isochronous/interrupt transactions. If less than three data bytes of the full-/low-speed data packet have been received at the end of a microframe, the TT must respond with a NYET to the corresponding high-speed complete-split. Both of these responses indicate to the host that more data is being received and another complete-split transaction is required.

When the host controller receives a DATA0/1 PID for interrupt or isochronous IN complete-splits (and ACK, NAK, STALL, ERR for interrupt IN/OUT complete-splits), it stops issuing any remaining complete-splits that might be scheduled for that endpoint for this full-/low-speed transaction.

If the TT has not started the full-/low-speed transaction when it receives a complete-split, the TT will not find an entry in the complete-split pipeline stage. When this happens, the protocol state machines show that the TT responds with a NYET (e.g., the “no match” case). This NYET response tells the host that there are no results available currently, but the host should continue with other scheduled split transactions for this endpoint in subsequent microframes.

In general, there will be two (or more) complete-split transactions scheduled for a periodic endpoint. However, for interrupt endpoints, the maximum size of the full-/low-speed transaction guarantees that it can never require more than two complete-split transactions. Two complete-split transactions are only required when the transaction spans a microframe boundary. In cases where the full-/low-speed transaction actually

Microchip Ex. 1018, p. 406
Microchip v. Aptiv
IPR2024-00494

starts and completes in the same microframe, only a single complete-split will return data; any other earlier complete-splits will have a NYET response.

For isochronous IN transactions, more complete-split transactions may be scheduled based on the length of the full-speed transaction. A full-speed isochronous IN transaction can be up to 1023 data bytes, which can require portions of up to 8 microframes of time on the downstream facing bus (with the worst alignment in the frame and worst case bit stuffing). Such a maximum sized full-speed transaction can require 8 complete-split transactions. If the device generates less data, the host will stop issuing complete-splits after the one that returns the final data from the device for a frame.

11.18.6 TT Periodic Transaction Handling Requirements

The TT has two methods it must use to react to timing related events that affect the microframe pipeline: current transaction abort and freeing pending start-splits. These methods must be used to manage the microframe pipeline.

The TT must also react (as described in Section 11.22.1) when its microframe or frame timer loses synchronization with the high-speed bus.

The TT must not issue too many full-/low-speed transactions in any microframe.

Each of these requirements are described below.

11.18.6.1 Abort of Current Transaction

When a current transaction is in progress on the downstream facing bus and it is no longer appropriate for the TT to continue the transaction, the transaction is "aborted."

The TT full-/low-speed handler must abort the current full-/low-speed transaction:

1. For all periodic transaction types, if the full-speed frame EOF time occurs
2. If the transaction is an interrupt transaction and the start-split for the transaction was received in some microframe (call it X) and the TT microframe timer indicates the X+4 microframe

Note that no additional abort handling is required for isochronous transactions besides the generic IN/OUT handling described below. Abort has different processing requirements with regards to the downstream facing bus for IN and OUT transactions. For any type of transaction, the TT must not generate a complete-split response for an aborted transaction; e.g., no entry is made in the complete-split pipeline stage for an aborted transaction.

1. At the time the TT decides to abort an IN transaction, the TT must not issue the handshake packet for the transaction if the handshake has not already been started on the downstream facing bus. The TT may choose to not issue the IN token packet, if possible. If the transaction is in the data phase (e.g., in the middle of the target device generated DATA packet), the TT simply awaits the completion of that packet and ignores any data received and must not respond with a full-/low-speed handshake. The TT must not make an entry in the complete-split pipeline stage. This processing will cause a NYET response to the corresponding complete-split on the high-speed bus.
2. At the time the TT decides to abort an OUT transaction, the TT may choose to not issue the TOKEN or DATA packets, if possible. If the TT is in the middle of the DATA packet, it must stop issuing data bytes as soon as possible and force a bit-stuffing error on the downstream facing bus. In any case, the TT must not make an entry in the complete-split pipeline stage. This processing will cause a NYET response to the corresponding complete-split on the high-speed bus.

11.18.6.2 Free of Pending Start-splits

A start-split can be buffered in the start-split pipeline stage that is no longer appropriate to cause a full-/low-speed transaction on the downstream facing bus. Such a start-split transaction must be "freed" from the

Microchip Ex. 1018, p. 407
Microchip v. Aptiv
IPR2024-00494

start-split pipeline stage. This means the start-split is simply ignored by the TT and the TT must respond to a corresponding complete-split with a NYET. For example, no entry is made in the complete-split pipeline stage for the freed start-split.

A start-split in the start-split pipeline must be freed:

1. If the full-speed frame EOF time occurs, except for start-splits received in $(Y\text{-}1)_7$
2. If the start-split transaction was received in some microframe (call it X) and the TT microframe timer indicates the X+4 microframe

If the TT receives a periodic start-split transaction in microframe Y_6, its behavior is undefined. This is a host scheduling error.

11.18.6.3 Maximum Full-/low-speed Transactions per Microframe

The TT must not start a full-/low-speed transaction unless it has space available in the complete-split pipeline stage to hold the results of the transaction. If there is not enough space, the TT must wait to issue the transaction until there is enough space. The maximum number of normally operating full-speed transactions that can ever be completed in a microframe is 16.

11.18.7 TT Transaction Tracking

Figure 11-66 shows the TT microframe pipeline of transactions. The 8 high-speed microframes that compose a full-/low-speed frame are labeled with Y_0 through Y_7 assuming the microframe timer has occurred at the point in time shown by the arrow (e.g., time "NOW").

As shown in the figure, a start-split high-speed transaction that the high-speed handler receives in microframe Y_0 (e.g., a start-split "B") can run on the full-/low-speed bus during microframe times Y_1 or Y_2 or Y_3. This variation in starting on the full-/low-speed bus is due to bit stuffing and bulk/control reclamation that can occur on the full-/low-speed bus. Once the full-/low-speed transaction finishes, its complete-split transactions (if they are required) will run on the high-speed bus during microframes Y_2, Y_3, or Y_4.

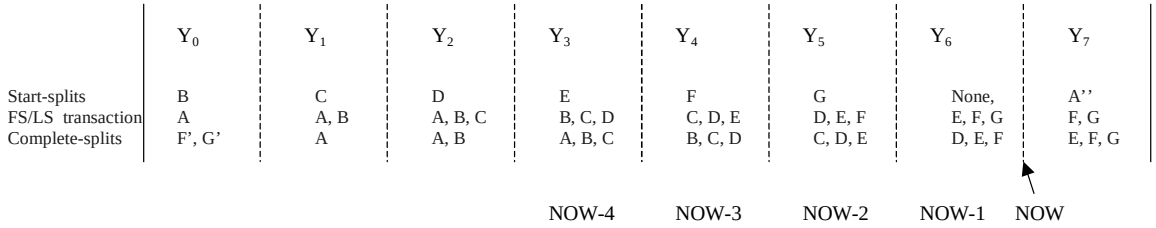


Figure 11-66. Microframe Pipeline

When the microframe timer indicates a new microframe, the high-speed handler must mark any start-splits in the start-split pipeline stage it received in the previous microframe as "pending" so that they can be processed on the full-/low-speed bus as appropriate. This prevents the full-/low-speed transactions from running on the downstream bus too early.

At the beginning of each microframe (call it "NOW"), the high-speed handler must free (as defined in Section 11.18.6.2) any start-split transactions from the start-split pipeline stage that are still pending from microframe NOW-4 (or earlier) and ignore them. If the transaction is in progress on the downstream facing bus, the transaction must be aborted (with full-/low-speed methods as defined in Chapter 8). This is described in more detail in the previous sections. This ensures that even if the full-/low-speed bus has encountered a babble condition on the bus (or other delay condition), the TT keeps its periodic transaction pipeline running on time (e.g., transactions do not run too late). This also ensures that when the last scheduled complete-split transaction is received by the TT, the full-/low-speed transaction has been completed (either successfully or by being aborted).

Microchip Ex. 1018, p. 408
Microchip v. Aptiv
IPR2024-00494

Finally, at the beginning of each microframe, the high-speed handler must change any complete-split transaction responses in the complete-split pipeline stage from microframe NOW-2 to the free state so that their space can be reused for responses in this microframe.

This algorithm is shown in pseudo code in Figure 11-67. This pseudo-code corresponds to the Advance_pipeline procedure identified previously.

```
-- Clean up start-split state in case full-/low-speed bus fell behind
while start-splits in pending state received by TT before microframe-4 loop
   Free start-split entry
End loop

-- Clean up complete-split pipeline in case no complete-splits were received
While complete-split transaction states from (microframe-2) loop
   Free complete-split response transaction entry
End loop

-- Enable full-/low-speed transactions received in previous microframe
While start-split transactions from (previous_microframe) loop
   Set start-split entry to pending status
End loop
```

Figure 11-67. Advance_Pipeline Pseudocode

11.18.8 TT Complete-split Transaction State Searching

A host must issue complete-split transactions in a microframe for a set of full-/low-speed endpoints in the same relative order as the start-splits were issued in a microframe for this TT. However, errors on start- or complete-splits can cause the high-speed handler to receive a complete-split transaction that does not "match" the expected next transaction according to the TT's transaction pipeline.

The TT has a pipeline of complete-split transaction state that it is expecting to use to respond to complete-split transactions. Normally the host will issue the complete-split that the high-speed handler is expecting next and the complete-split will correspond to the entry at the front of the complete-split pipeline.

However, when errors occur, the complete-split transaction that the high-speed handler receives might not match the entry at the front of the complete-split pipeline. This can happen for example, when a start-split is damaged on the high-speed bus and the high-speed handler does not receive it successfully. Or the high-speed handler might have a match, but the matching entry is located after the state for other expected complete-splits that the high-speed handler did not receive (due to complete-split errors on the high-speed bus).

The high-speed handler must respond to a complete-split transaction with the results of a full-/low-speed transaction that it has completed. This means that the high-speed handler must search to find the correct state tracking entry among several possible complete-split response entries. This searching takes time. The high-speed handler only needs to search the complete-split responses accumulated during the previous microframe. There only needs to be at most 1 microframe of complete-split response entries; the microframe of responses that have already been accumulated and are awaiting to be returned via high-speed complete-splits.

The split transaction protocol is defined to allow the high-speed handler to timeout the first high-speed complete-split transaction while it is searching for the correct response. This allows the high-speed handler time to complete its search and respond correctly to the next (retried) complete-split.

The following interrupt and isochronous flow sequence figures show these cases with the transitions labeled "Search not complete in time" and "No split response found".

The high-speed handler matches the complete-split transaction with the correct entry in the complete-split pipeline stage and advances the pipeline appropriately. There are five cases the TT must handle correctly:

1. If the high-speed complete-split token and first entry of the complete-split pipeline match, the high-speed handler responds with the indicated data/status. This case occurs the first time the TT receives a complete-split.

Microchip Ex. 1018, p. 409
Microchip v. Aptiv
IPR2024-00494

2. Same as above, but this is a retry of a complete-split that the TT has already received due to the host controller not receiving the (previous) response information.
3. If the complete-split transaction matches some other entry in the complete-split pipeline besides the first, the high-speed handler advances the complete-split pipeline (e.g., frees response information for previous complete-split entries) and responds with the information for the matching entry. This case can happen due to normal or missed previous complete-split transactions. An example abnormal case could be that the host controller was unsuccessful in issuing a complete-split transaction to the high-speed handler and has done endpoint halt processing for that endpoint. This means the next complete-split will not match the first entry of the complete-split pipeline stage.
4. The high-speed handler can also receive a complete-split before it has started a full-/low-speed transaction. If there is not an entry in the complete-split pipeline, the high-speed handler responds with a NYET handshake to inform the host that it has no status information. When the host issues the last scheduled complete-split for this endpoint for this frame, it must interpret the NYET as an error condition. This stimulates the normal “three strikes” error handling. If there have been more than three errors, the host halts this endpoint. If there have been less than three errors, the host continues processing the scheduled transactions of this endpoint (e.g., a start-split will be issued as the next transaction for this endpoint at the next scheduled time for this endpoint). Note that a NYET response is possible in this case due to a transaction error on the start-split or a host (or TT) scheduling error.
5. The high-speed handler can timeout its first high-speed complete-split transaction while it is searching the complete-split pipeline stage for a matching entry. However, the high-speed handler must respond correctly to the subsequent complete-split transaction. If the high-speed handler did not respond correctly for an interrupt IN after it had acknowledged the full-/low-speed transaction, the endpoint software and the device would lose data synchronization and more catastrophic errors could occur.

The host controller must issue the complete-split transactions in the same relative order as the original corresponding start-split transactions.

11.19 Approximate TT Buffer Space Required

A transaction translator requires buffer and state tracking space for its periodic and non-periodic portions.

The TT microframe pipeline requires less than:

- 752 data bytes for the start-split stage
- 2x 188 data bytes for the complete-split stage
- 16x 4x transaction status (<4 bytes for each transaction) for start-split stage
- 16x 2x transaction status (<4 bytes for each transaction) for complete-split stage

There are, at most, 4 microframes of buffering required for the start-split stage of the pipeline and, at most, 2 microframes of buffering for the complete-split stage of the pipeline. There are, at most, 16 full-speed (minimum sized) transactions possible in any microframe.

The non-periodic portion of the TT requires at least:

- 2x (64 data + 4 transaction status) bytes

Different implementations may require more or less buffering and state tracking space.

11.20 Interrupt Transaction Translation Overview

The flow sequence and state machine figures show the transitions required for high-speed split transactions for full-/low-speed interrupt transfer types for a single endpoint. These figures must not be interpreted as showing any particular specific timing. In particular, high-speed or full-/low-speed transactions for other endpoints may occur before or after these split transactions. Specific details are described as appropriate.

Microchip Ex. 1018, p. 410
Microchip v. Aptiv
IPR2024-00494

In contrast to bulk/control processing, the full-/low-speed handler must not do local retry processing on the full-/low-speed bus in response to a transaction error for full-/low-speed interrupt transactions.

11.20.1 Interrupt Split Transaction Sequences

The interrupt IN and OUT flow sequence figures use the same notation and have descriptions similar to the bulk/control figures.

In contrast to bulk/control processing, the full-speed handler must not do local retry processing on the full-speed bus in response to a transaction errors (including timeout) of an interrupt transaction.

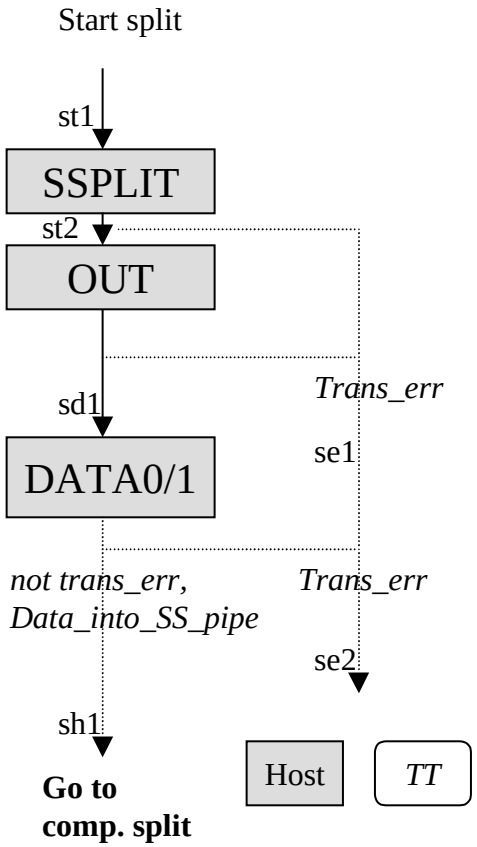


Figure 11-68. Interrupt OUT Start-split Transaction Sequence

Microchip Ex. 1018, p. 411
Microchip v. Aptiv
IPR2024-00494

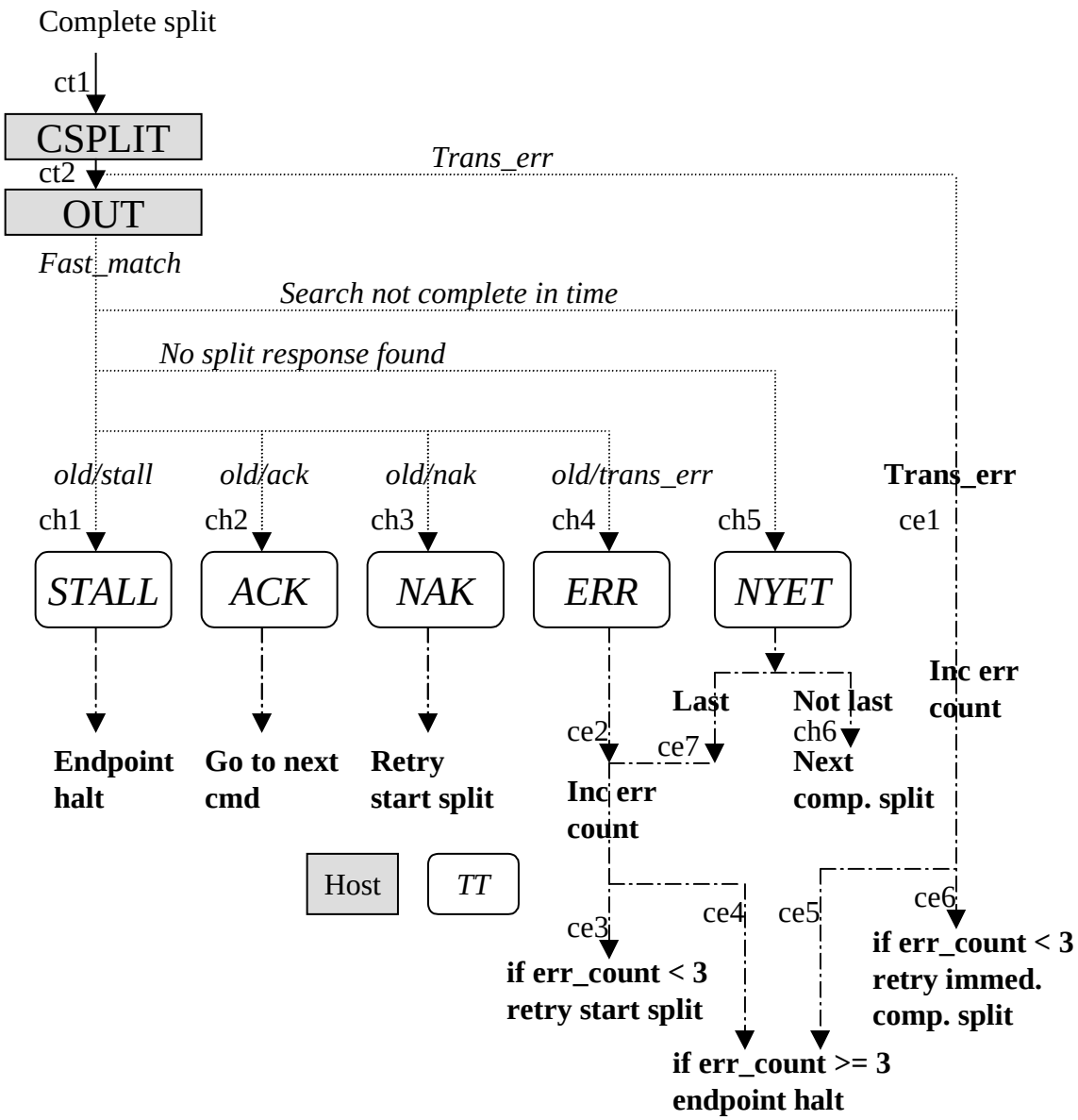


Figure 11-69. Interrupt OUT Complete-split Transaction Sequence

Microchip Ex. 1018, p. 412
Microchip v. Aptiv
IPR2024-00494

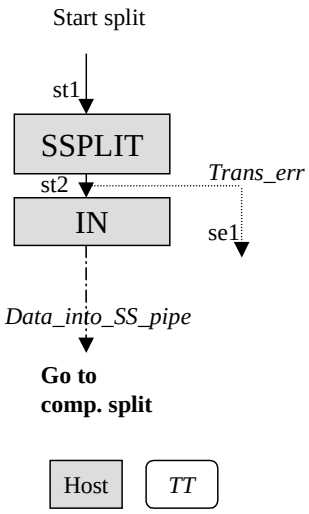


Figure 11-70. Interrupt IN Start-split Transaction Sequence

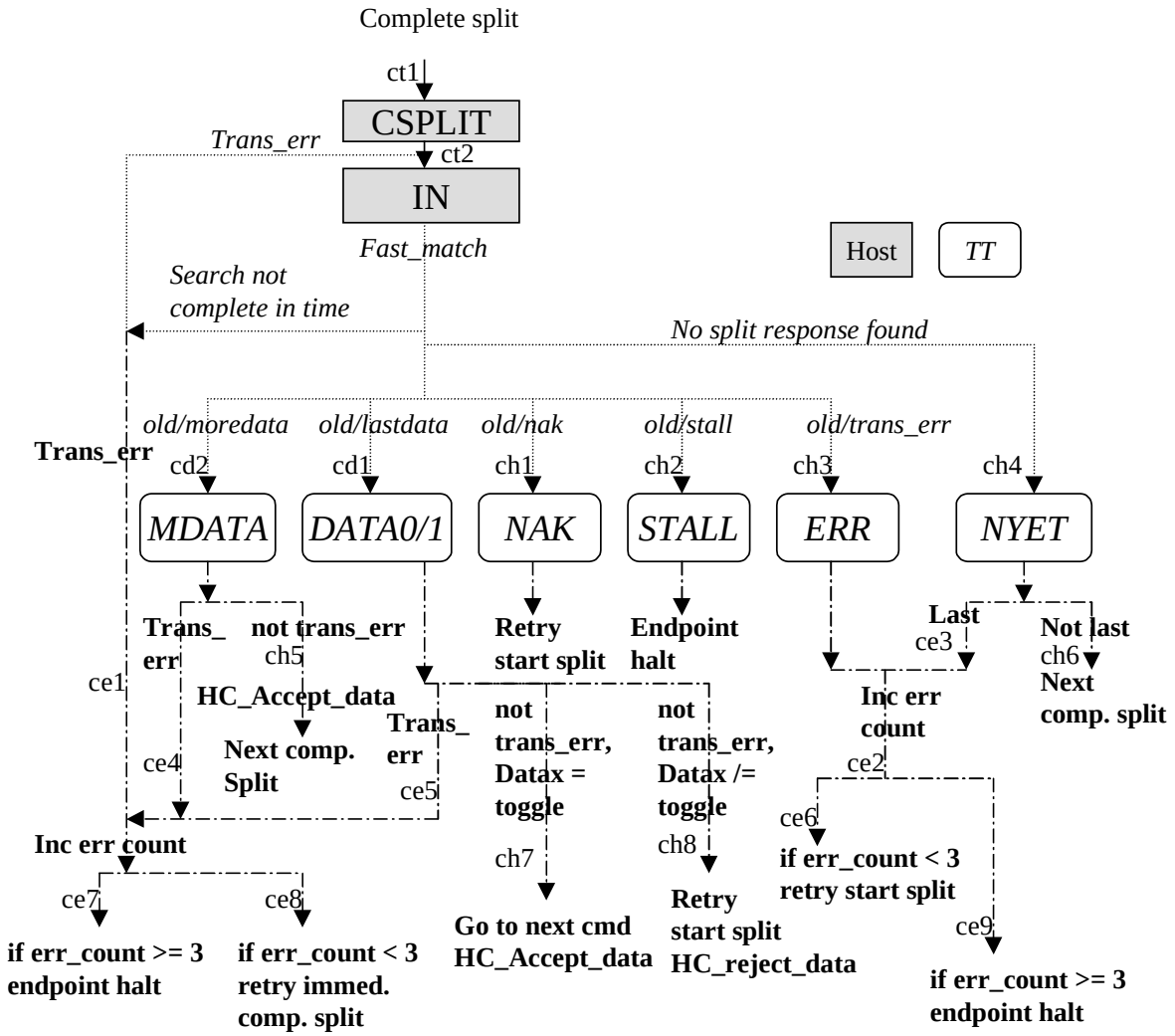


Figure 11-71. Interrupt IN Complete-split Transaction Sequence

Microchip Ex. 1018, p. 413
Microchip v. Aptiv
IPR2024-00494

11.20.2 Interrupt Split Transaction State Machines

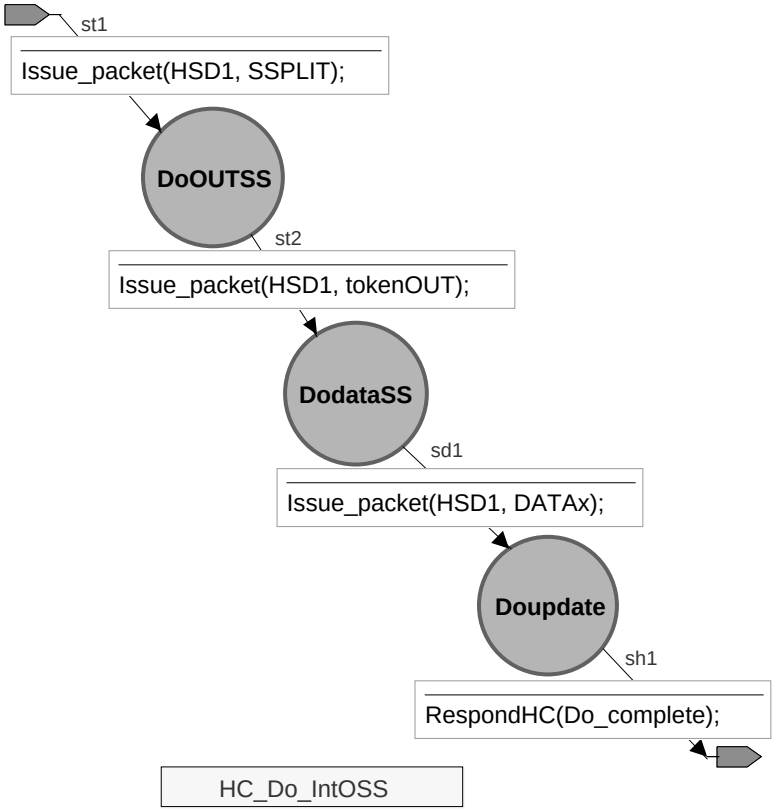


Figure 11-72. Interrupt OUT Start-split Transaction Host State Machine

Microchip Ex. 1018, p. 414
Microchip v. Aptiv
IPR2024-00494

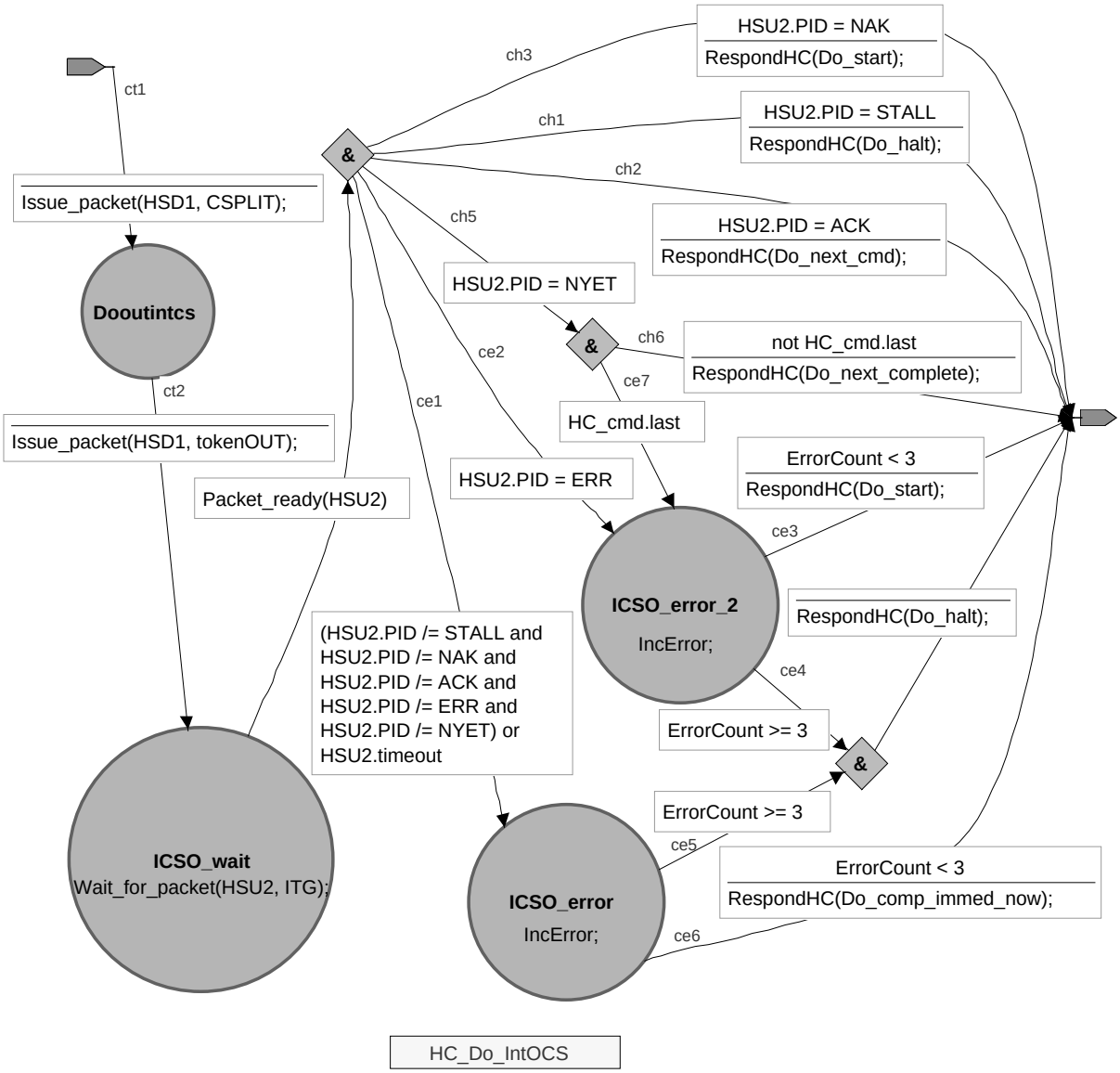


Figure 11-73. Interrupt OUT Complete-split Transaction Host State Machine

Microchip Ex. 1018, p. 415
Microchip v. Aptiv
IPR2024-00494

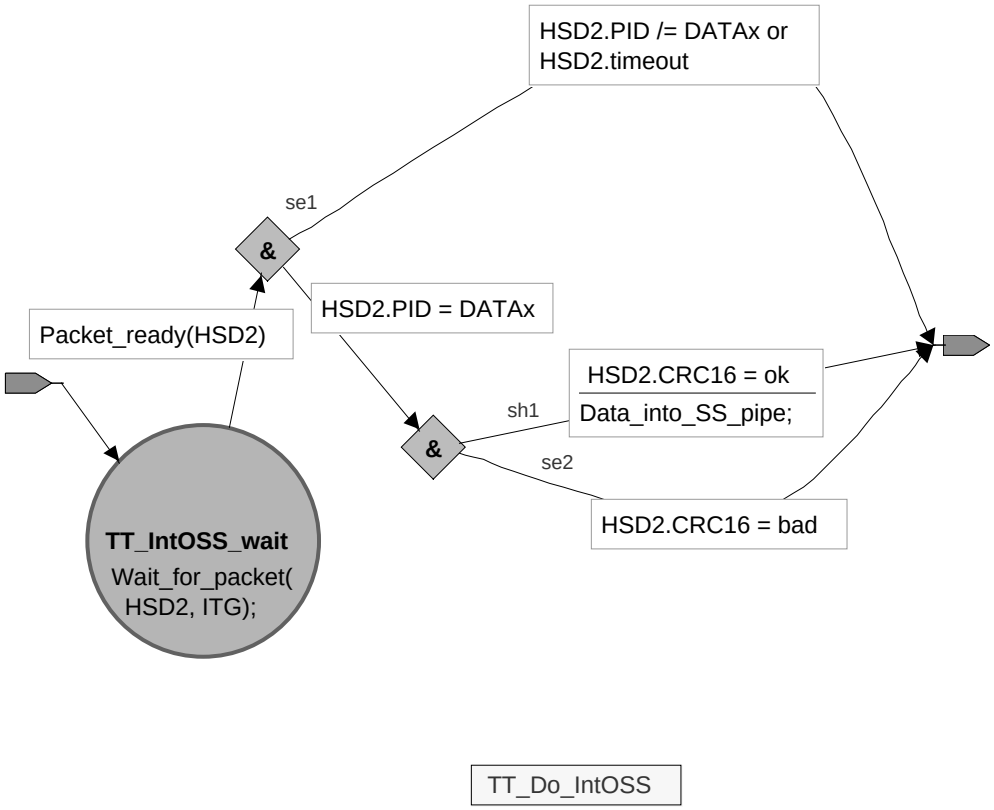


Figure 11-74. Interrupt OUT Start-split Transaction TT State Machine

Microchip Ex. 1018, p. 416
Microchip v. Aptiv
IPR2024-00494

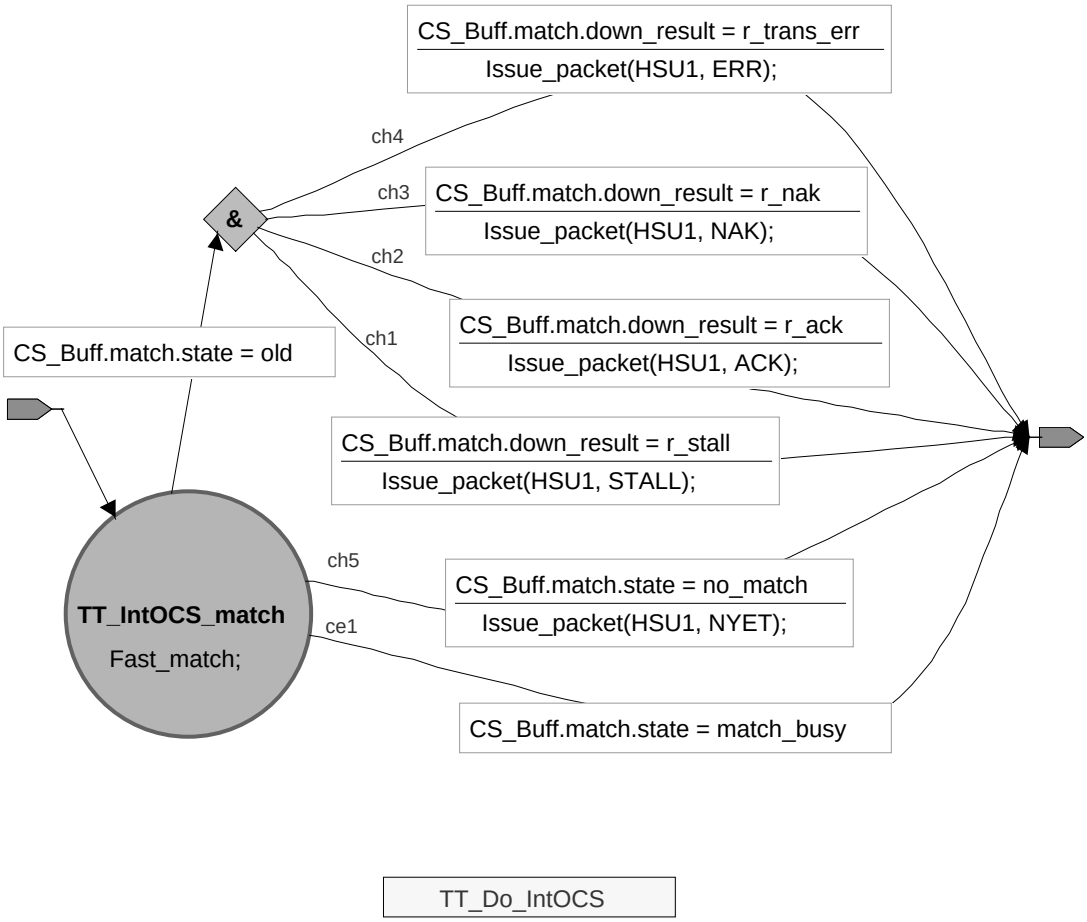


Figure 11-75. Interrupt OUT Complete-split Transaction TT State Machine

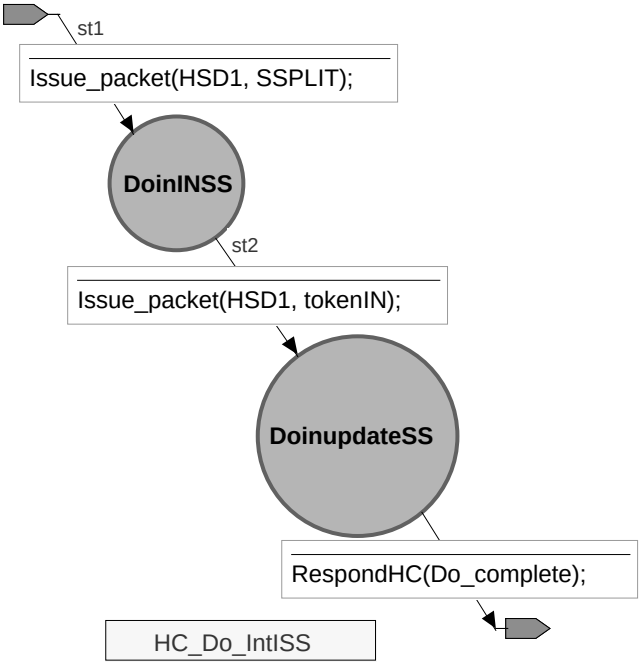


Figure 11-76. Interrupt IN Start-split Transaction Host State Machine

Microchip Ex. 1018, p. 417
Microchip v. Aptiv
IPR2024-00494

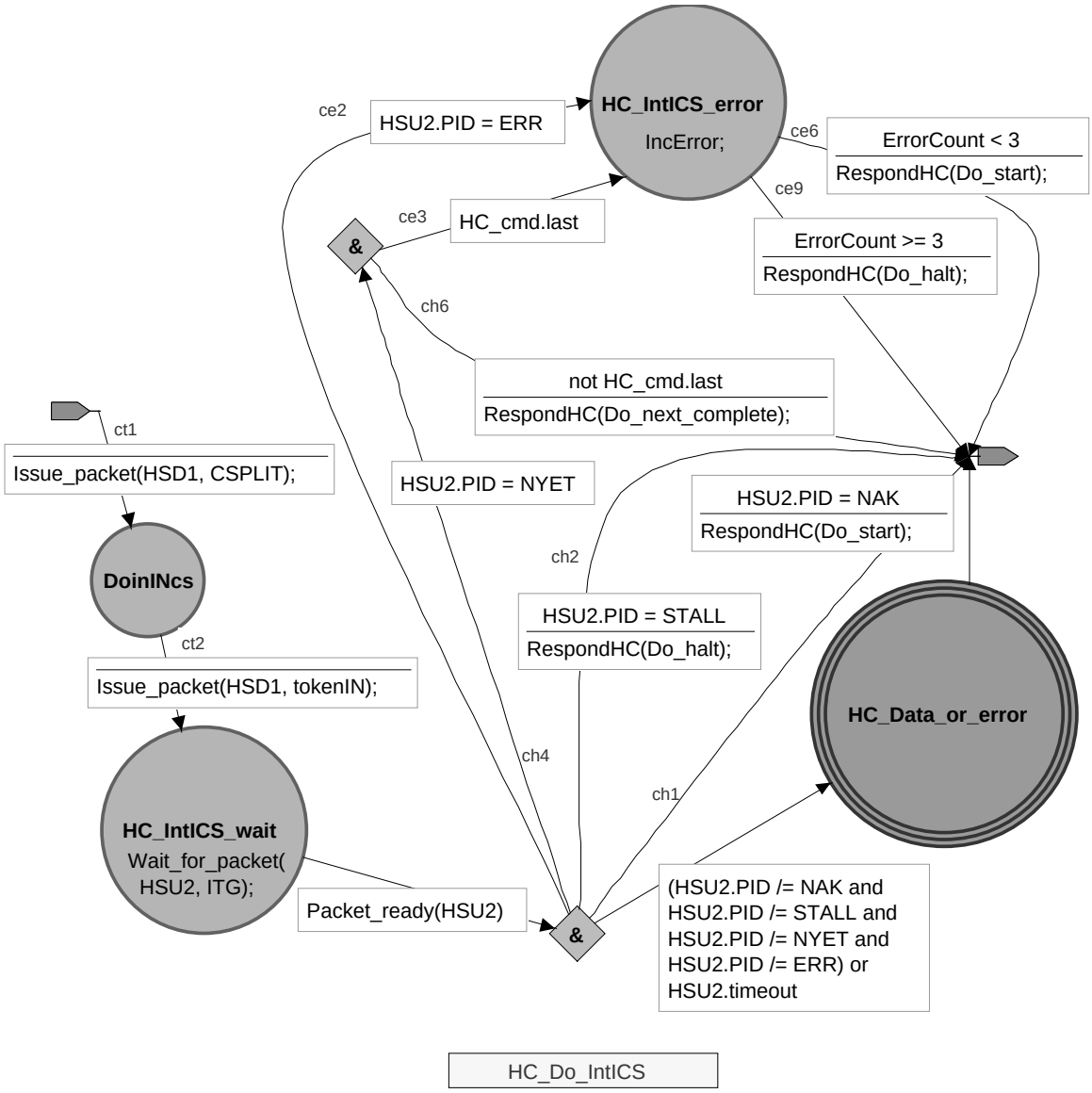


Figure 11-77. Interrupt IN Complete-split Transaction Host State Machine

Microchip Ex. 1018, p. 418
Microchip v. Aptiv
IPR2024-00494

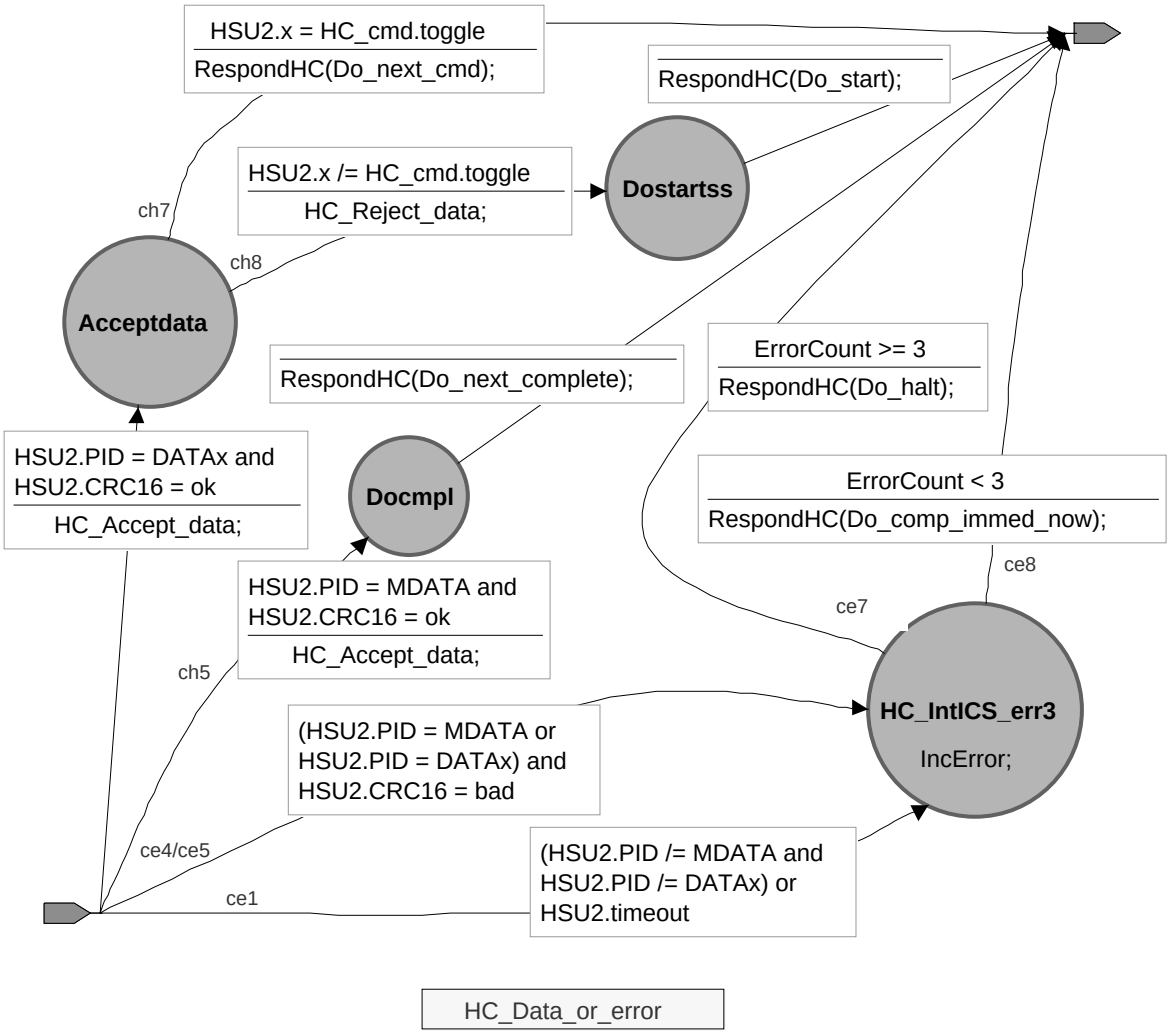


Figure 11-78. HC_Data_or_Error State Machine

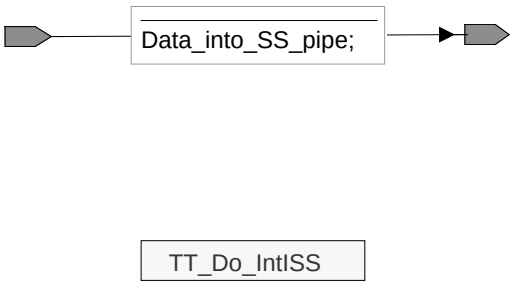


Figure 11-79. Interrupt IN Start-split Transaction TT State Machine

Microchip Ex. 1018, p. 419
Microchip v. Aptiv
IPR2024-00494

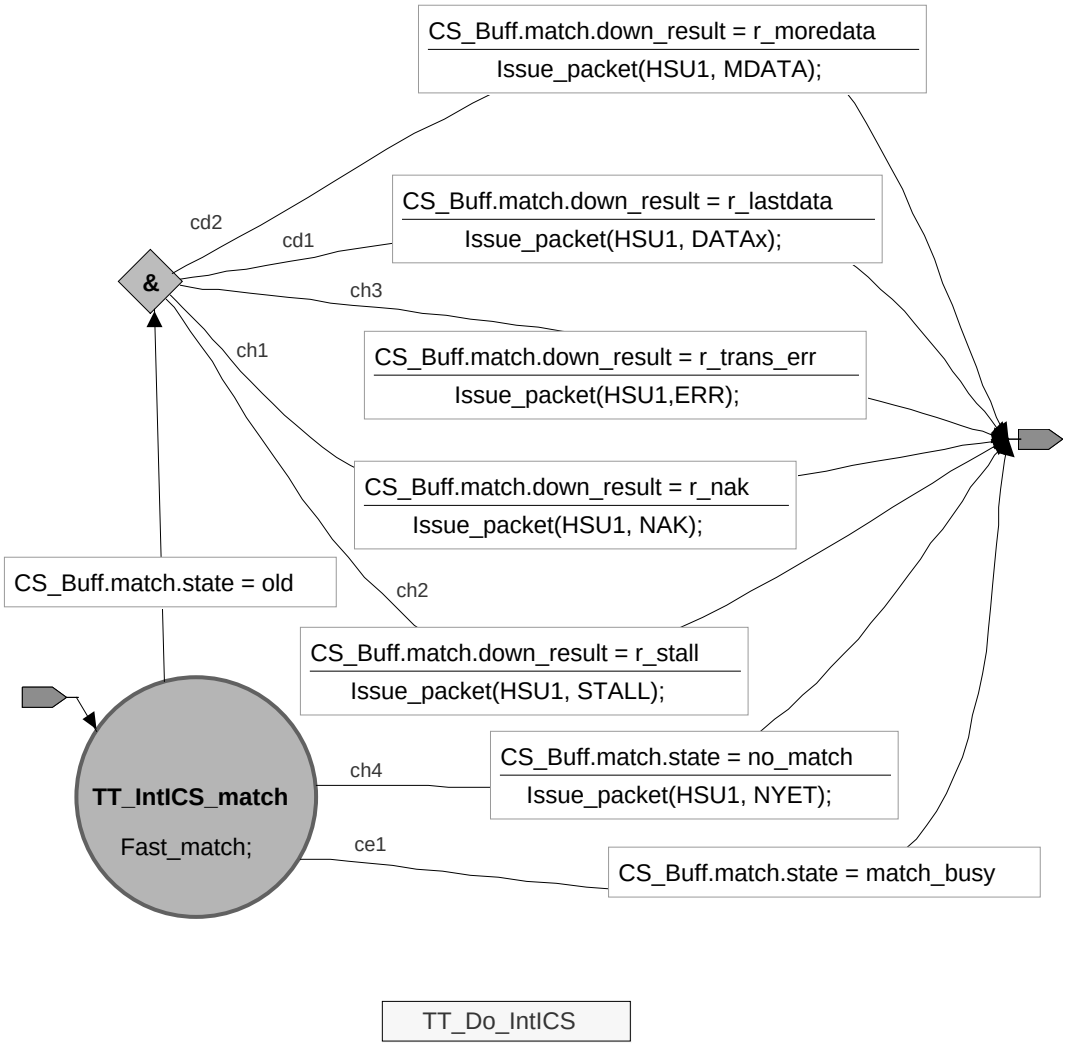


Figure 11-80. Interrupt IN Complete-split Transaction TT State Machine

11.20.3 Interrupt OUT Sequencing

Interrupt OUT split transactions are scheduled by the host controller as normal high-speed transactions with the start- and complete-splits scheduled as described previously.

When there are several full-/low-speed transactions allocated for a given microframe, they are saved by the high-speed handler in the TT in the start-split pipeline stage. The start-splits are saved in the order they are received until the end of the microframe. At the end of the microframe, these transactions are available to be issued by the full-/low-speed handler on the full-/low-speed bus in the order they were received.

In a following microframe (as described previously), the full-/low-speed handler issues the transactions that had been saved in the start-split pipeline stage on the downstream facing full-/low-speed bus. Some transactions could be leftover from a previous microframe since the high-speed schedule was built assuming best case bit stuffing and the full-/low-speed transactions could be taking longer on the full-/low-speed bus. As the full-/low-speed handler issues transactions on the downstream facing full-/low-speed bus, it saves the results in the periodic complete-split pipeline stage and then advances to the next transaction in the start-split pipeline.

In a following microframe (as described previously), the host controller issues a high-speed complete-split transaction and the TT responds appropriately.

Microchip Ex. 1018, p. 420
Microchip v. Aptiv
IPR2024-00494

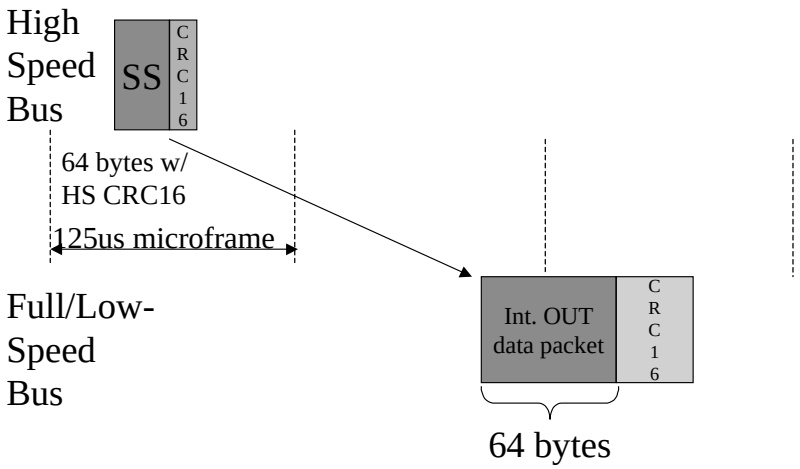


Figure 11-81. Example of CRC16 Handling for Interrupt OUT

The start-split transaction for an interrupt OUT transaction includes a normal CRC16 field for the high-speed data packet of the data phase of the start-split transaction. However, the data payload of the data packet contains only the data payload of the corresponding full-/low-speed data packet; i.e., there is only a single CRC16 in the data packet of the start-split transaction. The TT high-speed handler must check the CRC on the start-split and ignore the start-split if there is a failure in the CRC check of the data packet. If the start-split has a CRC check failure, the full-speed transaction must not be started on the downstream bus. Figure 11-81 shows an example of the CRC16 handling for an interrupt OUT transaction and its start-split.

11.20.4 Interrupt IN Sequencing

When the high-speed handler receives an interrupt start-split transaction, it saves the packet in the start-split pipeline stage. In this fashion, it accumulates some number of start-split transactions for a following microframe.

At the beginning of the next microframe (as described previously), these transactions are available to be issued by the full-/low-speed handler on the downstream full-/low-speed bus in the order they were saved in the start-split pipeline stage. The full-/low-speed handler issues each transaction on the downstream facing bus. The full-/low-speed handler responds to the full-/low-speed transaction with an appropriate handshake as described in Chapter 8. The full-/low-speed handler saves the results of the transaction (data, NAK, STALL, trans_err) in the complete-split pipeline stage.

During following microframes, the host controller issues high-speed complete-split transactions to retrieve the data/handshake from the high-speed handler. When the high-speed handler receives s complete-split transaction, the TT returns whatever data it has received during a microframe. If the full-/low-speed transaction was started and completed in a single microframe, the TT returns all the data for the transaction in the complete-split response occurring in the following microframe. If the full-/low-speed CRC check passes, the appropriate DATA0/1 PID for the data packet is used. If the full-/low-speed CRC check fails, an ERR handshake is used and there is no data packet as part of the complete-split transaction.

If the full-/low-speed transaction spanned a microframe, the TT requires two complete-splits (in two subsequent microframes) to return all the data for the full-/low-speed transaction. The data packet PID for the first complete-split must be an MDATA to tell the host controller that another complete-split is required for this endpoint. This MDATA response is made without performing a CRC check (since the CRC16 field has not yet been received on the full-/low-speed bus). The complete-split in the next microframe must use a DATA0/1 PID if the CRC check passes. If the CRC check fails, an ERR handshake response is made instead and there is no data packet as part of the complete-split transaction. Since full-speed interrupt transactions are limited to 64 data bytes or less (and low-speed interrupt transactions are limited to 8 data

Microchip Ex. 1018, p. 421
Microchip v. Aptiv
IPR2024-00494

bytes or less), no full-/low-speed interrupt transaction can span more than a single microframe boundary; i.e., no more than two microframes are ever required to complete the transaction.

The complete-split transaction for an interrupt IN transaction must not include the CRC16 field received from the full-/low-speed data packet (i.e., only a high-speed CRC16 field is used in split transactions). The TT must use a high-speed CRC16 on each complete-split data packet. If the full-speed handler detects a failed CRC check, it must use an ERR handshake response in the complete-split transaction to reflect that error to the high-speed host controller. The host controller must check the CRC16 on each returned complete-split data packet. A CRC failure (or ERR handshake) on any (partial) complete-split is reflected as a CRC failure on the total full-/low-speed transaction. This means that for a case where a full-/low-speed interrupt spans a microframe boundary, the host controller can accept the first complete-split without errors, then the second complete-split can indicate that the data from the first complete-split must be rejected as if it were never received by the host controller. Figure 11-82 shows an example of an interrupt IN and its CRC16 handling with corresponding complete-split responses.

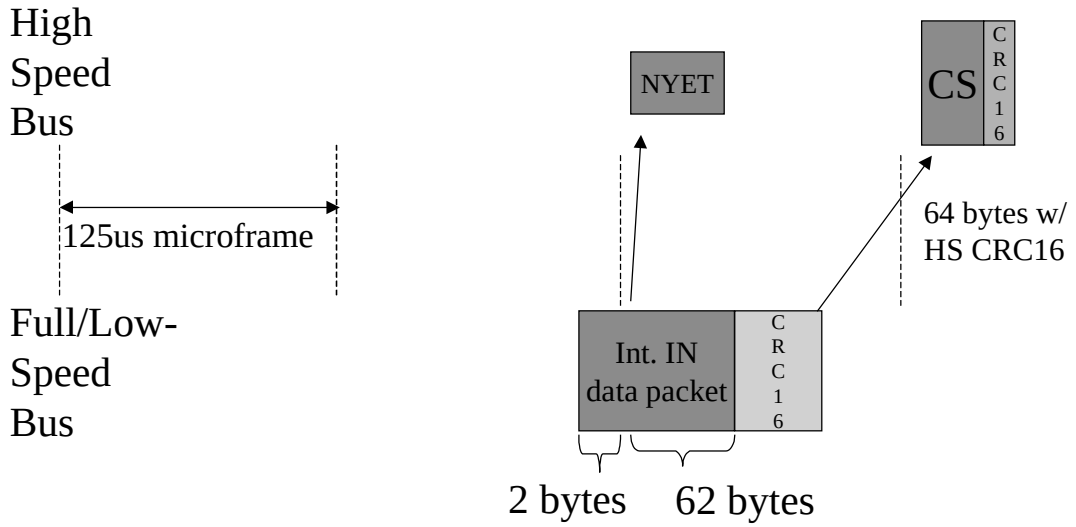


Figure 11-82. Example of CRC16 Handling for Interrupt IN

11.21 Isochronous Transaction Translation Overview

Isochronous split transactions are handled by the host by scheduling start- and complete-split transactions as described previously. Isochronous IN split transactions have more than two schedule entries:

- One entry for the start-split transaction in the microframe before the earliest the full-speed transaction can occur
- Other entries for the complete-splits in microframes after the data can occur on the full-speed bus (similar to interrupt IN scheduling)

Furthermore, isochronous transactions are split into microframe sized pieces; e.g., a 300 byte full-speed transaction is budgeted multiple high-speed split transactions to move data to/from the TT. This allows any alignment of the data for each microframe.

Full-speed isochronous OUT transactions issued by a TT do not have corresponding complete-split transactions. They must only have start-split transaction(s).

The host controller must preserve the same order for the complete-split transactions (as for the start-split transactions) for IN handling.

Microchip Ex. 1018, p. 422
Microchip v. Aptiv
IPR2024-00494

Isochronous INs have start- and complete- split transactions. The “first” high-speed split transaction for a full-speed endpoint is always a start-split transaction and the second (and others as required) is always a complete-split no matter what the high-speed handler of the TT responds.

The full-/low-speed handler recombines OUT data in its local buffers to recreate the single full-speed data transaction and handle the microframe error cases. The full-/low-speed handler splits IN response data on microframe boundaries.

Microframe buffers always advance no matter what the interactions with the host controller or the full-speed handler.

11.21.1 Isochronous Split Transaction Sequences

The flow sequence and state machine figures show the transitions required for high-speed split transactions for a full-speed isochronous transfer type for a single endpoint. These figures must not be interpreted as showing any particular specific timing. In particular, high-speed or full-speed transactions for other endpoints may occur before or after these split transactions. Specific details are described as appropriate.

In contrast to bulk/control processing, the full-speed handler must not do local retry processing on the full-speed bus in response to transaction errors (including timeout) of an isochronous transaction.

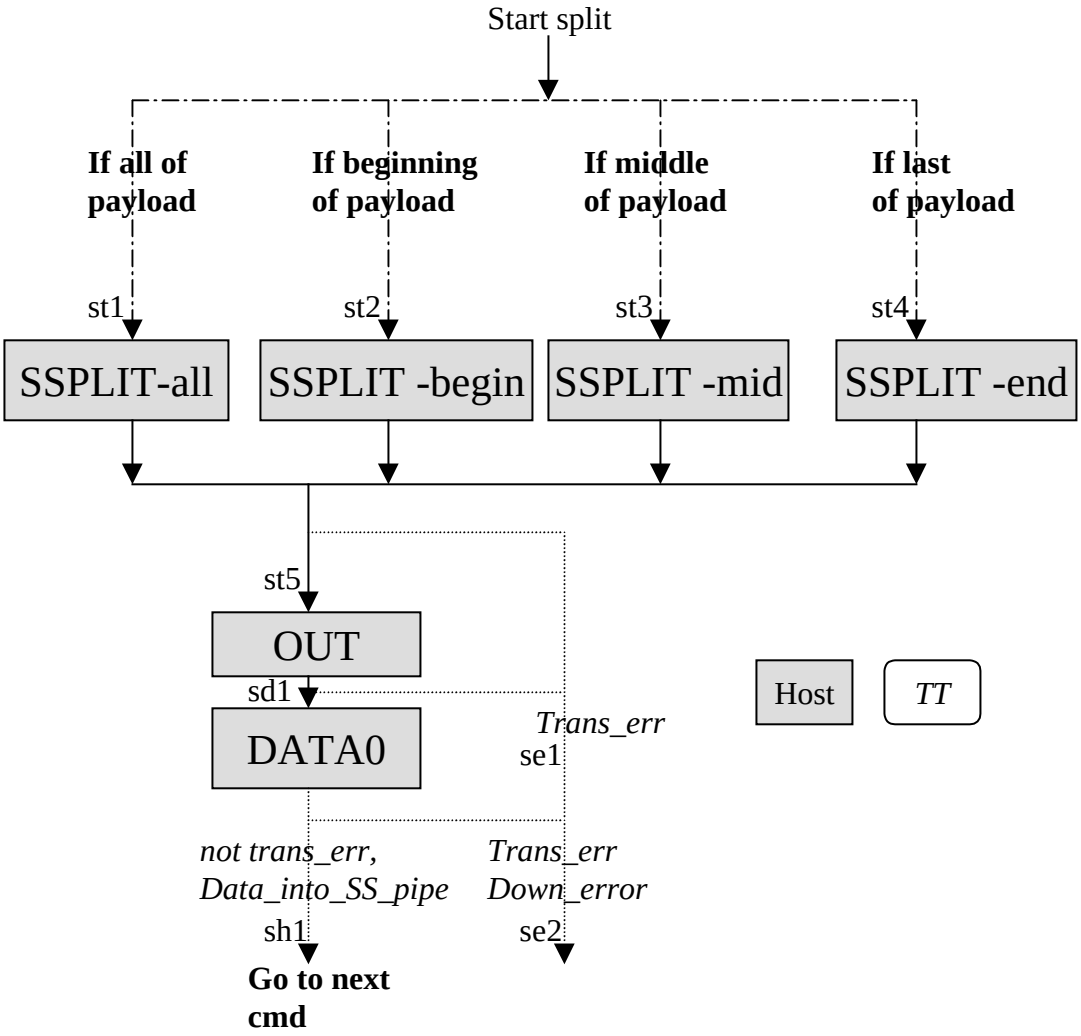


Figure 11-83. Isochronous OUT Start-split Transaction Sequence

Microchip Ex. 1018, p. 423
Microchip v. Aptiv
IPR2024-00494

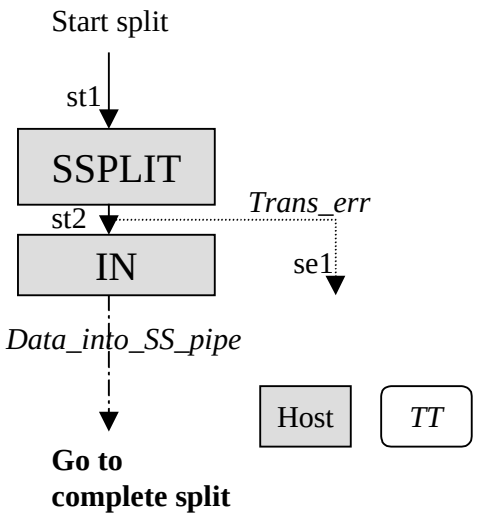


Figure 11-84. Isochronous IN Start-split Transaction Sequence

In Figure 11-85, the high-speed handler returns an ERR handshake for a “transaction error” of the full-speed transaction.

The high-speed handler returns an NYET handshake when it cannot find a matching entry in the complete-split pipeline stage. This handles the case where the host controller issued the first high-speed complete-split transaction, but the full-/low-speed handler has not started the transaction yet or has not yet received data back from the full-speed device. This can be due to a delay from starting previous full-speed transactions.

The transition labeled "TAdvance" indicates that the host advances to the next transaction for this full-speed endpoint.

The transition labeled "DAdvance" indicates that the host advances to the next data area of the current transaction for the current full-speed endpoint.

Microchip Ex. 1018, p. 424
Microchip v. Aptiv
IPR2024-00494

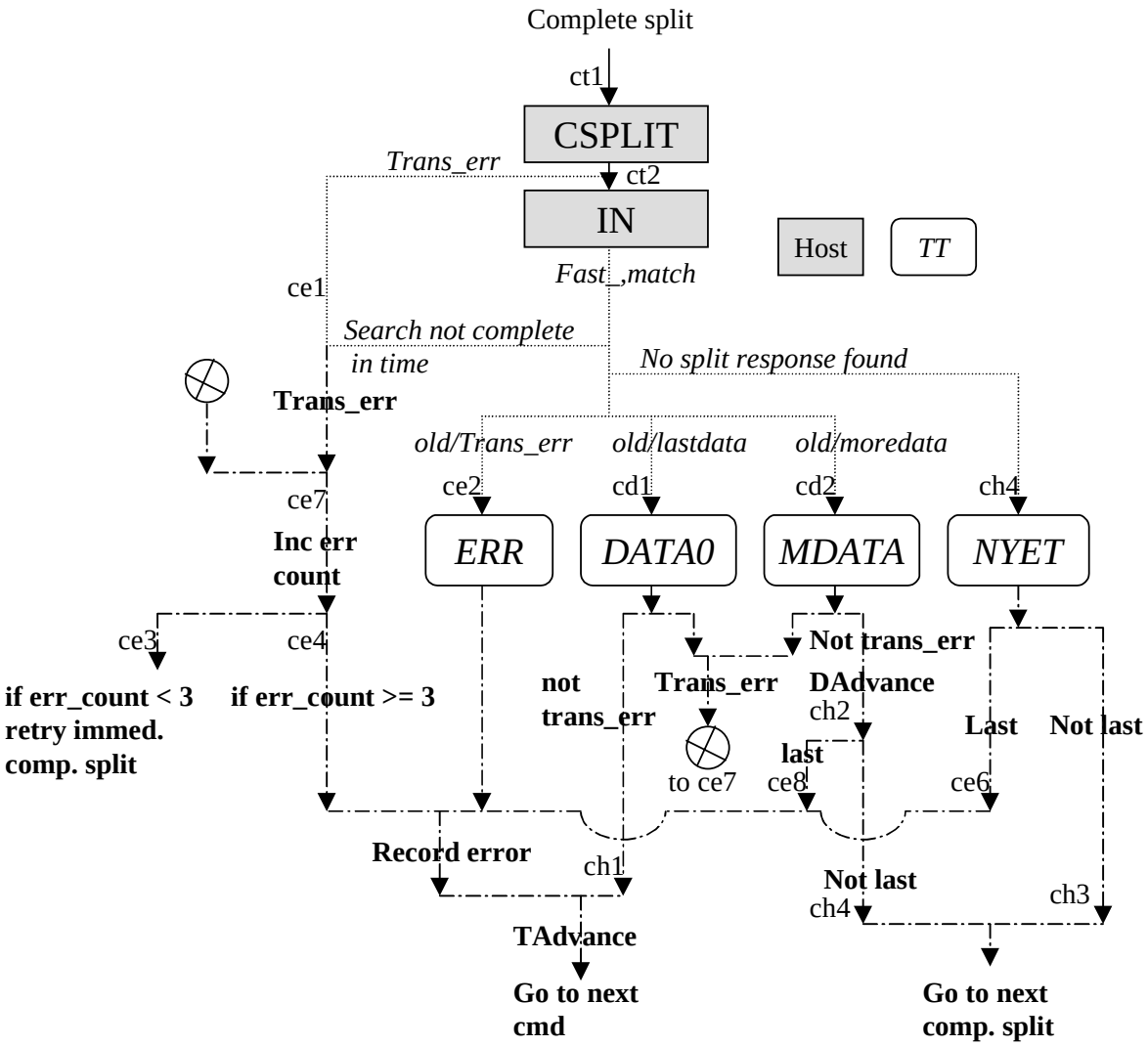


Figure 11-85. Isochronous IN Complete-split Transaction Sequence

Microchip Ex. 1018, p. 425
Microchip v. Aptiv
IPR2024-00494

11.21.2 Isochronous Split Transaction State Machines

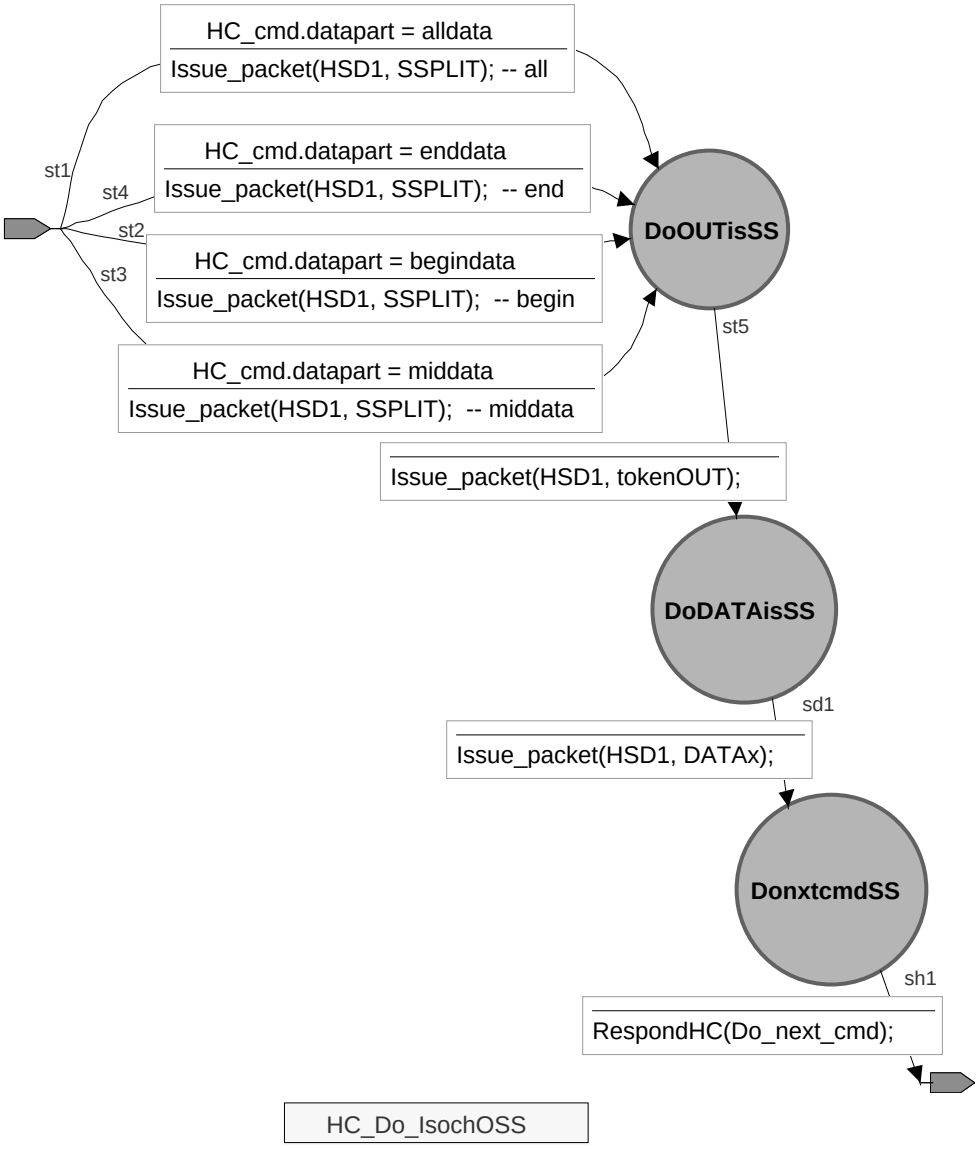


Figure 11-86. Isochronous OUT Start-split Transaction Host State Machine

Microchip Ex. 1018, p. 426
Microchip v. Aptiv
IPR2024-00494

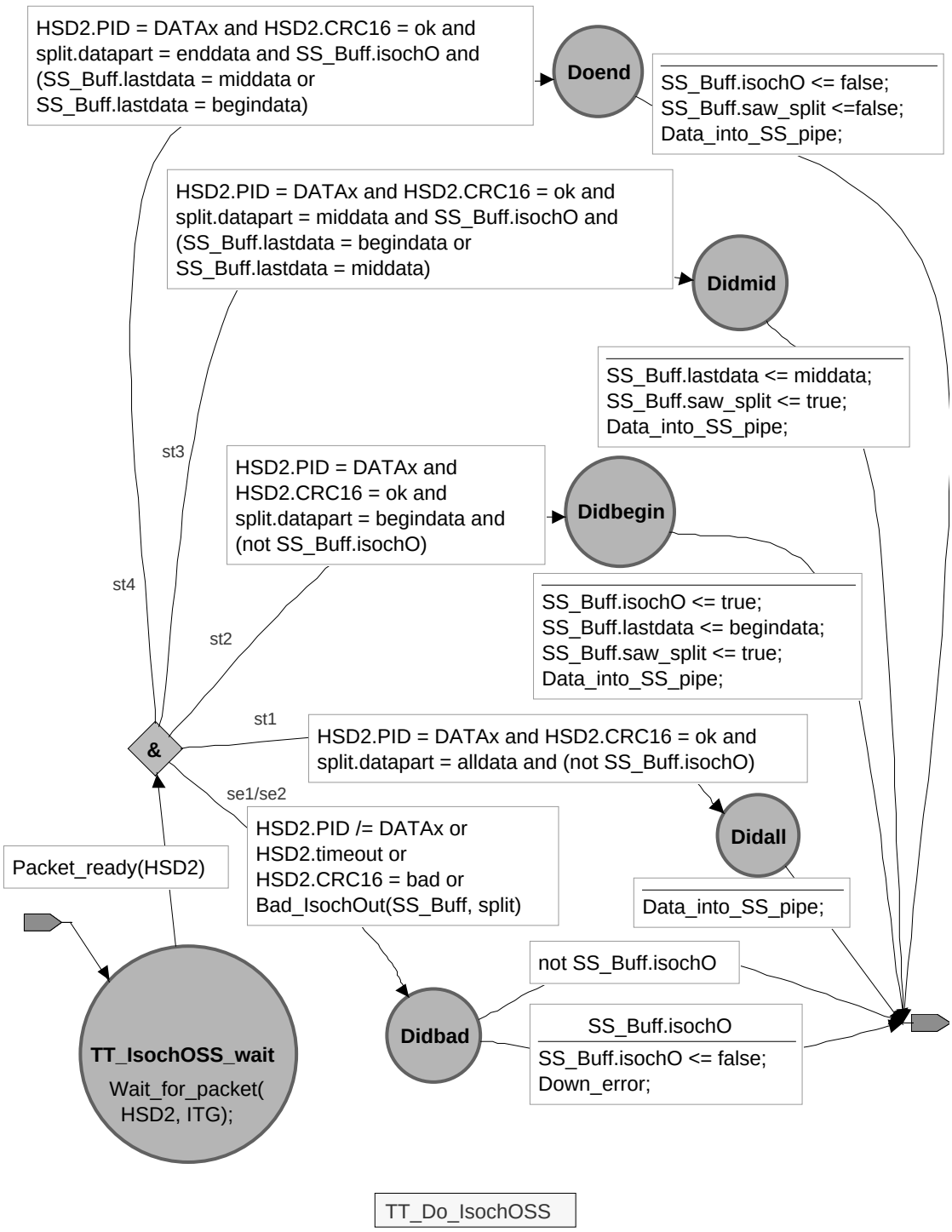


Figure 11-87. Isochronous OUT Start-split Transaction TT State Machine

There is a condition in Figure 11-87 on transition se1/se2 labeled “Bad_IsochOut”. This condition is true when none of the conditions on transitions st1 through st4 are true. The action labeled “Down_error” records an error to be indicated on the downstream facing full-speed bus for the transaction corresponding to this start-split.

Microchip Ex. 1018, p. 427
Microchip v. Aptiv
IPR2024-00494

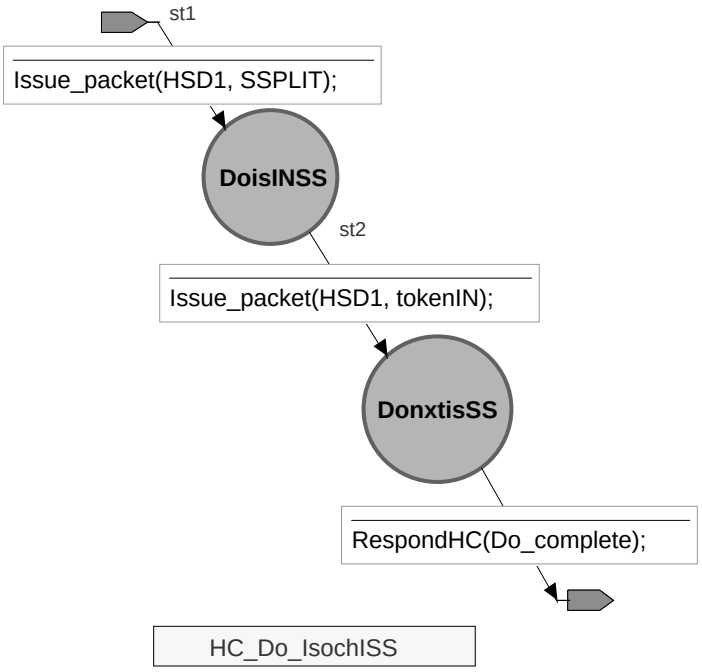


Figure 11-88. Isochronous IN Start-split Transaction Host State Machine

Microchip Ex. 1018, p. 428
Microchip v. Aptiv
IPR2024-00494

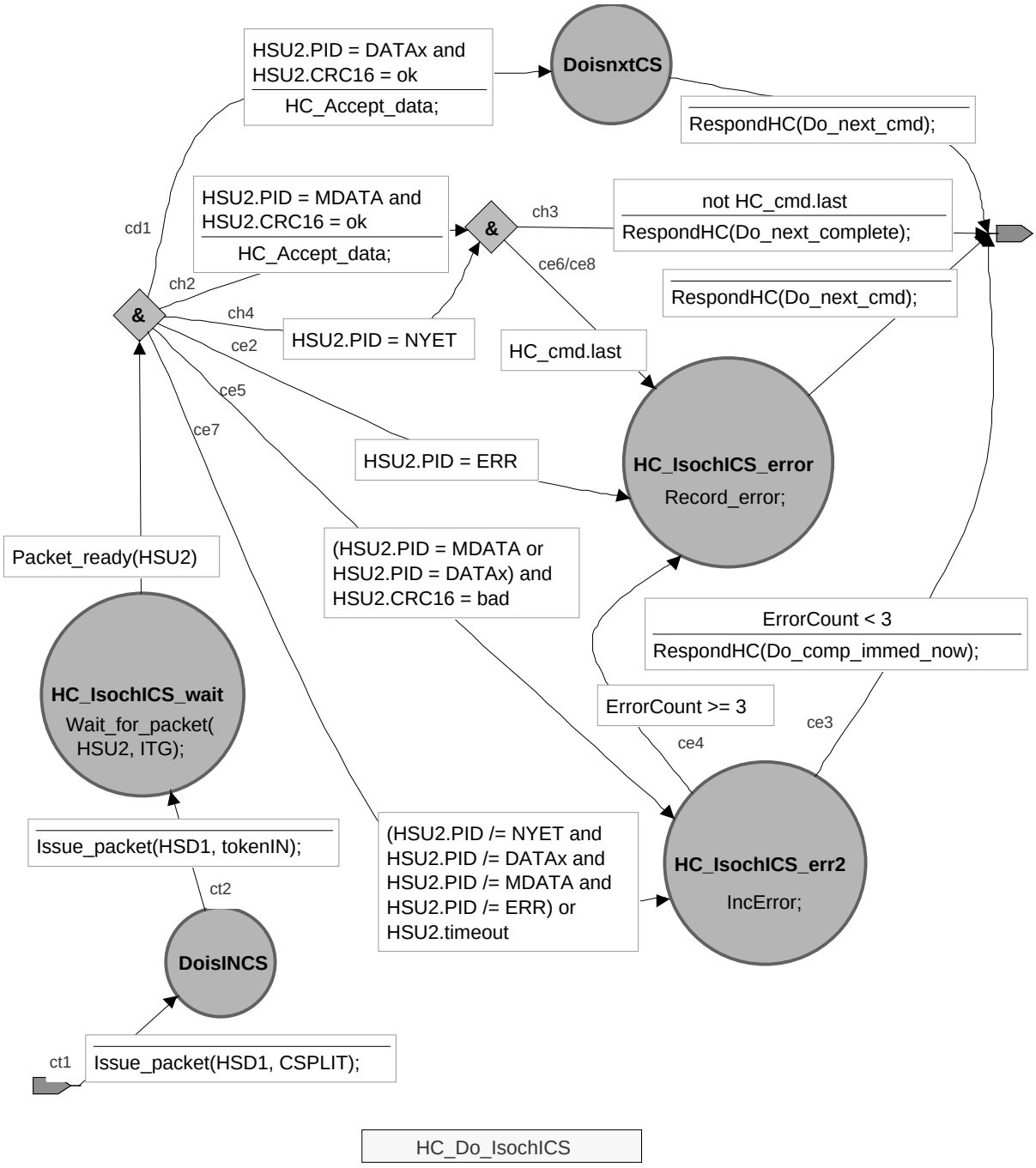


Figure 11-89. Isochronous IN Complete-split Transaction Host State Machine

In Figure 11-89, the transition “ce8” occurs when the high-speed handler responds with an MDATA to indicate there is more data for the full-speed transaction, but the host controller knows that this is the last scheduled complete-split for this endpoint for this frame. If a DATA0 response from the high-speed handler is not received before the last scheduled complete-split, the host controller records an error and proceeds to the next transaction for this endpoint (in the next frame).

Microchip Ex. 1018, p. 429
Microchip v. Aptiv
IPR2024-00494

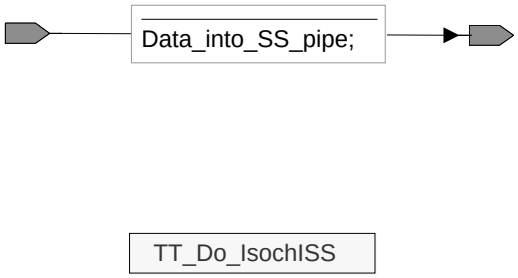


Figure 11-90. Isochronous IN Start-split Transaction TT State Machine

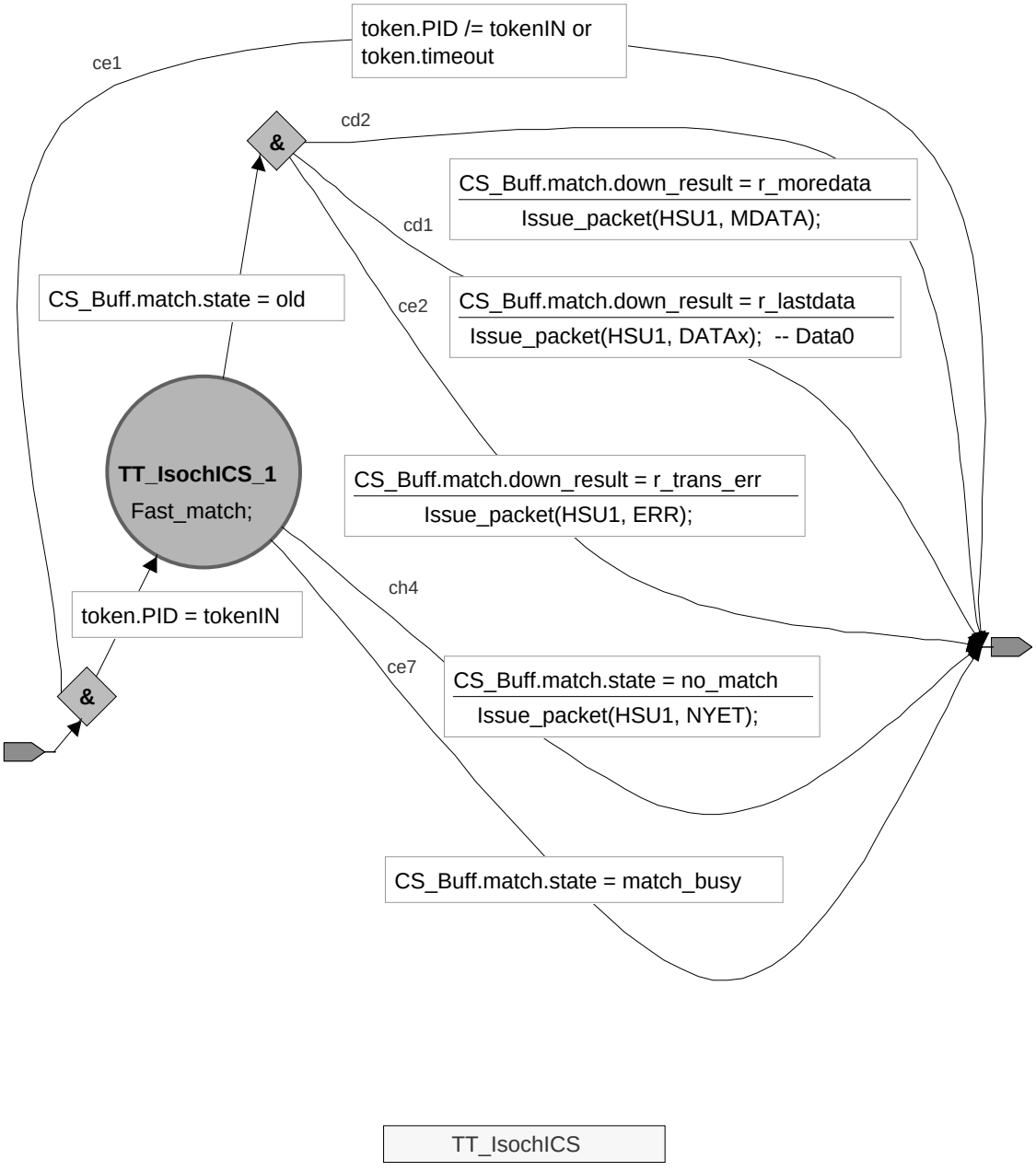


Figure 11-91. Isochronous IN Complete-split Transaction TT State Machine

Microchip Ex. 1018, p. 430
Microchip v. Aptiv
IPR2024-00494

11.21.3 Isochronous OUT Sequencing

The host controller and TT must ensure that errors that can occur in split transactions of an isochronous full-speed transaction translate into a detectable error. For isochronous OUT split transactions, once the high-speed handler has received an “SSPLIT-begin” start-split transaction token packet, the high-speed handler must track start-split transactions that are received for this endpoint. The high-speed handler must track that a start-split transaction is received each and every microframe until an “SSPLIT-end” split transaction token packet is received for this endpoint. If a microframe passes without the high-speed handler receiving a start-split for this full-speed endpoint, it must ensure that the full-speed handler forces a bitstuff error on the full-speed transaction. Any subsequent “SPLIT-middle” or “SPLIT-end” start-splits for the same endpoint must be ignored until the next non “SPLIT-middle” and non “SPLIT-end” is received (for any endpoint supported by this TT).

The start-split transaction for an isochronous OUT transaction must not include the CRC16 field for the full-speed data packet. For a full-speed transaction, the host would compute the CRC16 of the data packet for the full data packet (e.g., a 1023 byte data packet uses a single CRC16 field that is computed once by the host controller). For a split transaction, any isochronous OUT full-speed transaction is subdivided into multiple start-splits, each with a data payload of 188 bytes or less. For each of these start-splits, the host computes a high-speed CRC16 field for each start-split data packet. The TT high-speed handler must check each high-speed CRC16 value on each start-split. The TT full-speed handler must locally generate the CRC16 value for the complete full-speed data packet. Figure 11-92 shows an example of a full-speed isochronous OUT packet and the high-speed start-splits with their CRC16 fields.

If there is a CRC check failure on the high-speed start-split, the high-speed handler must indicate to the full-speed handler that there was an error in the start-split for the full-speed transaction. If the transaction has been indicated as having a CRC failure (or if there is a missed start-split), the full-speed handler uses the defined mechanism for forcing a downstream corrupted packet. If the first start-split has a CRC check failure, the full-speed transaction must not be started on the downstream bus.

Additional high-speed start-split transactions for the same endpoint must be ignored after a CRC check fails, until the high-speed handler receives either an “SSPLIT-end” start-split transaction token packet for that endpoint or a start-split for a different endpoint.

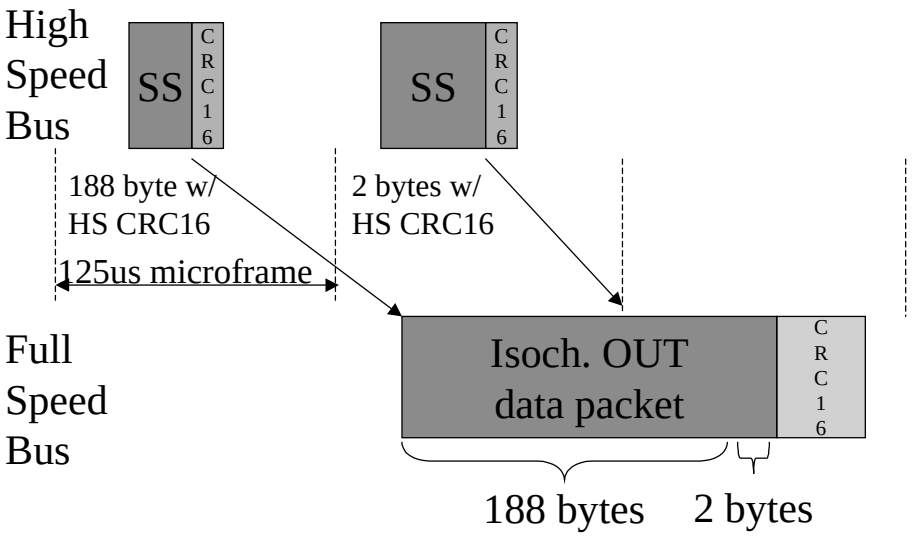


Figure 11-92. Example of CRC16 Isochronous OUT Data Packet Handling

Microchip Ex. 1018, p. 431
Microchip v. Aptiv
IPR2024-00494

11.21.4 Isochronous IN Sequencing

The complete-split transaction for an isochronous IN transaction must not include the CRC16 field for the full-speed data packet (e.g., only a high-speed CRC16 field is used in split transactions). The TT must not pass the full-speed value received from the device and instead only use high-speed CRC16 values for complete-split transactions. If the full-speed handler detects a failed CRC check at the end of the data packet (e.g., after potentially several complete-split transactions on high-speed), the handler must use an ERR handshake response to reflect that error to the high-speed host controller. The host controller must check the CRC16 on each returned high-speed complete-split. A CRC failure (or ERR handshake) on any (partial) complete-split is reflected by the host controller as a CRC failure on the total full-speed transaction. Figure 11-93 shows an example of the relationships of the full-speed data packet and the high-speed complete-splits and their CRC16 fields.

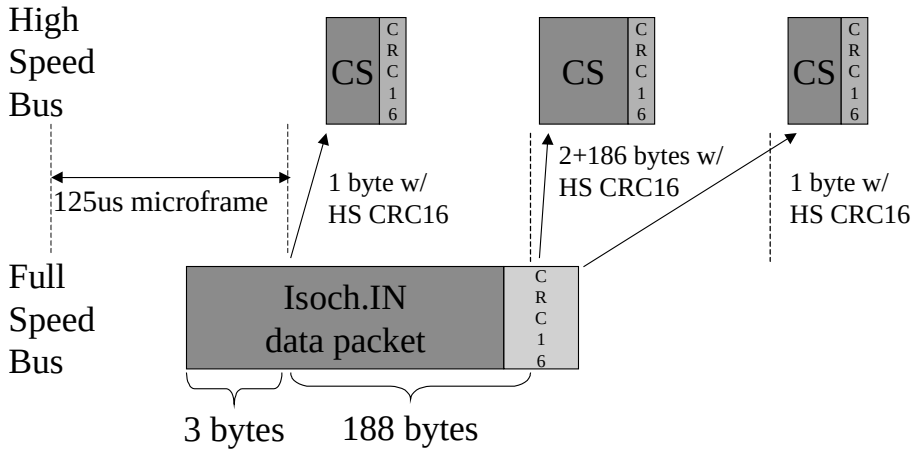


Figure 11-93. Example of CRC16 Isochronous IN Data Packet Handling

11.22 TT Error Handling

The TT has the same requirements for handling errors as a host controller or hub. In particular:

- If the TT is receiving a packet at EOF2 of the downstream facing bus, it must disable the downstream facing port that is currently transmitting.
- If the TT is transmitting a packet near EOF1 of the downstream facing bus, it must force an abnormal termination sequence as defined in Section 11.3.3 and stop transmitting.
- If the TT is going to transmit a non-periodic full-/low-speed transaction, it must determine that there is sufficient time remaining before EOF1 to complete the transaction. This determination is based on normal sequencing of the packets in the transaction. Since the TT has no information about data payload size for INs, it must use the maximum allowed size allowed for the transfer type in its determination. Periodic transactions do not need to be included in this test since the microframe pipeline is maintained separately.

11.22.1 Loss of TT Synchronization With HS SOFs

The hub has a timer it uses for (micro)frame maintenance. It has a 1 ms frame timer when operating at full-/low-speed for enforcing EOF with downstream connected devices. It has a 125 µs microframe timer when operating at high-speed for enforcing EOF with high-speed devices. It also uses the 125 µs microframe timer to create a 1 ms frame timer for enforcing EOF with downstream full-/low-speed devices when operating at high-speed. The hub (micro)frame timer must always stay synchronized with host generated SOFs to keep the bus operating correctly

Microchip Ex. 1018, p. 432
Microchip v. Aptiv
IPR2024-00494

In normal hub repeater (full- or high-speed) operation (e.g., not involving a TT), the (micro)frame timer loses synchronization whenever it has missed SOFs for three consecutive microframes. While timer synchronization is lost, the hub does not establish upstream connectivity. Downstream connectivity is established normally, even when timer synchronization is lost. When the timer is synchronized, the hub allows upstream connectivity to be established when required. The hub is responsible for ensuring that there is no signaling being repeated/transmitted upstream from a device after the EOF2 point in any (micro)frame. The hub must not establish upstream connectivity if it has lost (micro)frame timer synchronization since it no longer knows accurately where the EOF2 point is.

11.22.2 TT Frame and Microframe Timer Synchronization Requirements

When the hub is operating at high-speed and has full-/low-speed devices connected on its downstream facing ports (e.g., a TT is active), the hub has additional responsibilities beyond enforcement of the (high-speed) EOF2 point on its upstream facing port in every microframe. The TT must also generate full-speed SOFs downstream and ensure that the TT operates correctly (in bridging high-speed and full-/low-speed operation).

A high-speed operating hub synchronizes its microframe timer to 125 µs SOFs. However, in order to generate full-speed downstream SOFs, it must also have a 1 ms frame timer. It generates this 1 ms frame timer by recognizing zeroth microframe SOFs, e.g., a high-speed SOF when the frame number value changes compared to SOF of the immediately previous microframe.

In order to create the 1 ms frame timer, the hub must successfully receive a zeroth microframe SOF after its microframe timer is synchronized. In order to recognize a zeroth microframe SOF, the hub must successfully receive SOFs for two consecutive microframes where the frame number increments by 1 (mod 2^11). When the hub has done this, it knows that the second SOF is a zeroth microframe SOF and thereby establishes a 1 ms frame timer starting time. Note that a hub can synchronize both timers with as few as two SOFs if the SOFs are for microframe 7 and microframe 0, i.e., if the second SOF is a zeroth microframe SOF.

Once the hub has synchronized its 1 ms frame timer, it can keep that timer synchronized as long as it keeps its 125 µs microframe timer synchronized (since it knows that every 8 microframes from the zeroth microframe SOF is a 1 ms frame). In particular, the hub can keep its frame timer synchronized even if it misses zeroth microframe SOFs (as long as the microframe timer stays synchronized).

So in summary, the hub can synchronize its 125 µs microframe timer after receiving SOFs of two consecutive microframes. It synchronizes its 1 ms frame timer when it receives a zeroth microframe SOF (and the microframe timer is synchronized). The 125 µs microframe timer loses synchronization after three SOFs for consecutive microframes have been missed. This also causes the 1 ms frame timer to lose synchronization at the same time.

The TT must only generate full-speed SOFs downstream when its 1 ms frame timer is synchronized.

Correct internal operation of the TT is dependent on both timers. The TT must accurately know when microframes occur to enforce its microframe pipeline abort/free rules. It knows this based on a synchronized microframe timer (for generally incrementing the microframe number) and a synchronized frame timer (to know when the zeroth microframe occurs).

Since loss of microframe timer synchronization immediately causes loss of frame timer synchronization, the TT stops normal operation once the microframe timer loses synchronization. In an error free environment, microframe timer synchronization can be restored after receiving the two SOFs for the next two consecutive microframes (e.g., synchronization is restored at least 250 µs after synchronization loss). As long as SOFs are not missed, frame timer synchronization will be restored in less than 1 ms after microframe synchronization. Note that frame timer synchronization can be restored in a high-speed operating case in much less time (0.250-1.250 ms) than the 2-3 ms required in full-speed operation. Once the frame timer is synchronized, SOFs can be issued on downstream facing full-speed ports for the beginning of the next frame.

Microchip Ex. 1018, p. 433
Microchip v. Aptiv
IPR2024-00494

Once the hub detects loss of microframe timer synchronization, its TT(s):

1. Must respond to periodic complete-splits with any responses buffered in the periodic pipeline (only good for at most 1 microframe of complete-splits).
2. Must abort any buffered periodic start-split transactions in the periodic pipeline.
3. Must ignore any high-speed periodic start-splits.
4. Must stop issuing full-speed SOFs on downstream facing full-speed ports (and low-speed keep-alives on low-speed ports).
5. Must not start issuing subsequent periodic full-/low-speed transactions on downstream facing full-/low-speed ports.
6. Must respond to high-speed start-split bulk/control transactions.
7. Buffered bulk/control results must respond to high-speed complete-split transactions.
8. Pending bulk/control transactions must not be issued to full-/low-speed downstream facing ports. The TT buffers used to hold bulk/control transactions must be preserved until the microframe timer is re-synchronized. (Or until a Clear_TT_Buffer request is received for the transaction).

Note that in any case a TT must not issue transactions of any speed on downstream facing ports when its upstream facing port is suspended.

A TT only restores normal operation on downstream facing full-/low-speed ports after both microframe and frame timers are synchronized. Figure Figure 11-94 summarizes the relationship between high-speed SOFs and the TT frame and microframe timer synchronization requirements on start-splits.

For suspend sequencing of a hub, a hub will first lose microframe/frame timer synchronization at the same time. This will cause its TT(s) to stop issuing SOFs (which should be the only transactions keeping the downstream facing full-/low-speed ports out of suspend). Then the hub (along with any downstream devices) will enter suspend.

Upon a resume, the hub will first restore its microframe timer synchronization (after high-speed transactions continue). Then in less than 1 ms (assuming no errors), the frame timer will be synchronized and the TT can start normal operation (including SOFs/keep-alives on downstream facing full-/low-speed ports).

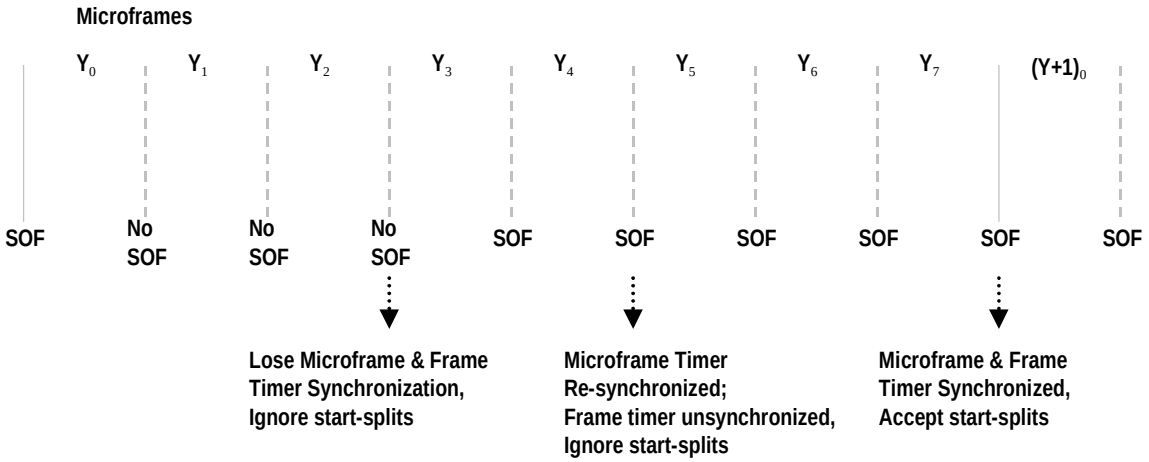


Figure 11-94. Example Frame/Microframe Synchronization Events

Microchip Ex. 1018, p. 434
Microchip v. Aptiv
IPR2024-00494

11.23 Descriptors

Hub descriptors are derived from the general USB device framework. Hub descriptors define a hub device and the ports on that hub. The host accesses hub descriptors through the hub's default pipe.

The USB specification (refer to Chapter 9) defines the following descriptors:

- Device
- Configuration
- Interface
- Endpoint
- String (optional)

The hub class defines additional descriptors (refer to Section 11.23.2). In addition, vendor-specific descriptors are allowed in the USB device framework. Hubs support standard USB device commands as defined in Chapter 9.

11.23.1 Standard Descriptors for Hub Class

The hub class pre-defines certain fields in standard USB descriptors. Other fields are either implementation-dependent or not applicable to this class.

A hub returns different descriptors based on whether it is operating at high-speed or full-/low-speed. A hub can report three different sets of the descriptors: one descriptor set for full-/low-speed operation and two sets for high-speed operation.

A hub operating at full-/low-speed has a device descriptor with a bDeviceProtocol field set to zero(0) and an interface descriptor with a bInterfaceProtocol field set to zero(0). The rest of the descriptors are the same for all speeds.

A hub operating at high-speed can have one of two TT organizations: single TT or multiple TT. All hubs must support the single TT organization. A multiple TT hub has an additional interface descriptor (with a corresponding endpoint descriptor). The first set of descriptors shown below must be provided by all hubs. A hub that has a single TT must set the bDeviceProtocol field of the device descriptor to one(1) and the interface descriptor bInterfaceProtocol field set to 0.

A multiple TT hub must set the bDeviceProtocol field of the device descriptor to two (2). The first interface descriptor has the bInterfaceProtocol field set to one(1). Such a hub also has a second interface descriptor where the bInterfaceProtocol is set to two(2). When the hub is configured with an interface protocol of one(1), it will operate as a single TT organized hub. When the hub is configured with an interface protocol of two(2), it will operate as a multiple TT organized hub. The TT organization must not be changed while the hub has full-/low-speed transactions in progress.

Microchip Ex. 1018, p. 435
Microchip v. Aptiv
IPR2024-00494

Note: For the descriptors and fields shown below, the bits in a field are organized in a little-endian fashion; that is, bit location 0 is the least significant bit and bit location 7 is the most significant bit of a byte value.

Full-/Low-speed Operating Hub

Device Descriptor (full-speed information):

bLength	12H
bDescriptorType	1
bcdUSB	0200H
bDeviceClass	HUB_CLASSCODE (09H)
bDeviceSubClass	0
bDeviceProtocol	0
bMaxPacketSize0	64
bNumConfigurations	1

Device_Qualifier Descriptor (high-speed information):

bLength	0AH
bDescriptorType	6
bcdUSB	200H
bDeviceClass	HUB_CLASSCODE (09H)
bDeviceSubClass	0
bDeviceProtocol	1 (for single TT) or 2 (for multiple TT)
bMaxPacketSize0	64
bNumConfigurations	1

Configuration Descriptor (full-speed information):

bLength	09H
bDescriptorType	2
wTotalLength	N
bNumInterfaces	1
bConfigurationValue	X
iConfiguration	Y
bmAttributes	Z
bMaxPower	The maximum amount of bus power the hub will consume in full-/low-speed configuration

Microchip Ex. 1018, p. 436
Microchip v. Aptiv
IPR2024-00494

Interface Descriptor:

bLength	09H
bDescriptorType	4
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	HUB_CLASSCODE (09H)
bInterfaceSubClass	0
bInterfaceProtocol	0
iInterface	i

Endpoint Descriptor (for Status Change Endpoint):

bLength	07H
bDescriptorType	5
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	Implementation-dependent
bInterval	FFH (Maximum allowable interval)

Other_Speed_Configuration Descriptor (High-speed information):

bLength	09H
bDescriptorType	7
wTotalLength	N
bNumInterfaces	1 (for single TT) or 2 (for multiple TT)
bConfigurationValue	X
iConfiguration	Y
bmAttributes	Z
bMaxPower	The maximum amount of bus power the hub will consume in high-speed configuration

Microchip Ex. 1018, p. 437
Microchip v. Aptiv
IPR2024-00494

Interface Descriptor:

bLength	09H
bDescriptorType	4
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	HUB_CLASSCODE (09H)
bInterfaceSubClass	0
bInterfaceProtocol	0 (for single TT) 1 (for multiple TT)
iInterface	i

Endpoint Descriptor (for Status Change Endpoint):

bLength	07H
bDescriptorType	5
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	Implementation-dependent
bInterval	FFH (Maximum allowable interval)

Interface Descriptor (present if multiple TT hub):

bLength	09H
bDescriptorType	4
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	HUB_CLASSCODE (09H)
bInterfaceSubClass	0
bInterfaceProtocol	2
iInterface	i

Microchip Ex. 1018, p. 438
Microchip v. Aptiv
IPR2024-00494

Endpoint Descriptor (present if multiple TT hub):

bLength	07H
bDescriptorType	5
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	Implementation-dependent
bInterval	FFH (Maximum allowable interval)

High-speed Operating Hub with Single TT

Device Descriptor (High-speed information):

bLength	12H
bDescriptorType	1
bcdUSB	200H
bDeviceClass	HUB_CLASSCODE (09H)
bDeviceSubClass	0
bDeviceProtocol	1
bMaxPacketSize0	64
bNumConfigurations	1

Device_Qualifier Descriptor (full-speed information):

bLength	0AH
bDescriptorType	6
bcdUSB	200H
bDeviceClass	HUB_CLASSCODE (09H)
bDeviceSubClass	0
bDeviceProtocol	0
bMaxPacketSize0	64
bNumConfigurations	1

Microchip Ex. 1018, p. 439
Microchip v. Aptiv
IPR2024-00494

Configuration Descriptor (high-speed information):

bLength	09H
bDescriptorType	2
wTotalLength	N
bNumInterfaces	1
bConfigurationValue	X
iConfiguration	Y
bmAttributes	Z
bMaxPower	The maximum amount of bus power the hub will consume in this configuration

Interface Descriptor:

bLength	09H
bDescriptorType	4
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	HUB_CLASSCODE (09H)
bInterfaceSubClass	0
bInterfaceProtocol	0 (single TT)
iInterface	i

Endpoint Descriptor (for Status Change Endpoint):

bLength	07H
bDescriptorType	5
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	Implementation-dependent
bInterval	FFH (Maximum allowable interval)

Microchip Ex. 1018, p. 440
Microchip v. Aptiv
IPR2024-00494

Other_Speed_Configuration Descriptor (full-speed information):

bLength	09H
bDescriptorType	7
wTotalLength	N
bNumInterfaces	1
bConfigurationValue	X
iConfiguration	Y
bmAttributes	Z
bMaxPower	The maximum amount of bus power the hub will consume in high-speed configuration

Interface Descriptor:

bLength	09H
bDescriptorType	4
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	HUB_CLASSCODE (09H)
bInterfaceSubClass	0
bInterfaceProtocol	0
iInterface	i

Endpoint Descriptor (for Status Change Endpoint):

bLength	07H
bDescriptorType	5
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	Implementation-dependent
bInterval	FFH (Maximum allowable interval)

Microchip Ex. 1018, p. 441
Microchip v. Aptiv
IPR2024-00494

High-speed Operating Hub with Multiple TTs

Device Descriptor (High-speed information):

bLength	12H
bDescriptorType	1
bcdUSB	200H
bDeviceClass	HUB_CLASSCODE (09H)
bDeviceSubClass	0
bDeviceProtocol	2 (multiple TTs)
bMaxPacketSize0	64
bNumConfigurations	1

Device_Qualifier Descriptor (full-speed information):

bLength	0AH
bDescriptorType	6
bcdUSB	200H
bDeviceClass	HUB_CLASSCODE (09H)
bDeviceSubClass	0
bDeviceProtocol	0
bMaxPacketSize0	64
bNumConfigurations	1

Configuration Descriptor (high-speed information):

bLength	09H
bDescriptorType	2
wTotalLength	N
bNumInterfaces	1
bConfigurationValue	X
iConfiguration	Y
bmAttributes	Z
bMaxPower	The maximum amount of bus power the hub will consume in this configuration

Microchip Ex. 1018, p. 442
Microchip v. Aptiv
IPR2024-00494

Interface Descriptor:

bLength	09H
bDescriptorType	4
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	HUB_CLASSCODE (09H)
bInterfaceSubClass	0
bInterfaceProtocol	1 (single TT)
iInterface	i

Endpoint Descriptor (for Status Change Endpoint):

bLength	07H
bDescriptorType	5
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	Implementation-dependent
bInterval	FFH (Maximum allowable interval)

Interface Descriptor:

bLength	09H
bDescriptorType	4
bInterfaceNumber	0
bAlternateSetting	1
bNumEndpoints	1
bInterfaceClass	HUB_CLASSCODE (09H)
bInterfaceSubClass	0
bInterfaceProtocol	2 (multiple TTs)
iInterface	i

Microchip Ex. 1018, p. 443
Microchip v. Aptiv
IPR2024-00494

Endpoint Descriptor:

bLength	07H
bDescriptorType	5
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	Implementation-dependent
bInterval	FFH (Maximum allowable interval)

Other_Speed_Configuration Descriptor (full-speed information):

bLength	09H
bDescriptorType	7
wTotalLength	N
bNumInterfaces	1
bConfigurationValue	X
iConfiguration	Y
bmAttributes	Z
bMaxPower	The maximum amount of bus power the hub will consume in high-speed configuration

Interface Descriptor:

bLength	09H
bDescriptorType	4
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	HUB_CLASSCODE (09H)
bInterfaceSubClass	0
bInterfaceProtocol	0
iInterface	i

Microchip Ex. 1018, p. 444
Microchip v. Aptiv
IPR2024-00494

Endpoint Descriptor (for Status Change Endpoint):

bLength	07H
bDescriptorType	5
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	Implementation-dependent
bInterval	FFH (Maximum allowable interval)

The hub class driver retrieves a device configuration from the USB System Software using the GetDescriptor() device request. The only endpoint descriptor that is returned by the GetDescriptor() request is the Status Change endpoint descriptor.

11.23.2 Class-specific Descriptors

11.23.2.1 Hub Descriptor

Table 11-13 outlines the various fields contained by the hub descriptor.

Table 11-13. Hub Descriptor

Offset	Field	Size	Description
0	*bDescLength*	1	Number of bytes in this descriptor, including this byte
1	*bDescriptorType*	1	Descriptor Type, value: 29H for hub descriptor
2	*bNbrPorts*	1	Number of downstream facing ports that this hub supports
3	*wHubCharacteristics*	2	D1...D0: Logical Power Switching Mode 00: Ganged power switching (all ports' power at once) 01: Individual port power switching 1X: Reserved. Used only on 1.0 compliant hubs that implement no power switching D2: Identifies a Compound Device 0: Hub is not part of a compound device. 1: Hub is part of a compound device. D4...D3: Over-current Protection Mode 00: Global Over-current Protection. The hub reports over-current as a summation of all ports' current draw, without a breakdown of individual port over-current status. 01: Individual Port Over-current Protection. The hub reports over-current on a per-port basis. Each port has an over-current status. 1X: No Over-current Protection. This option is allowed only for bus-powered hubs that do not implement over-current protection.

Microchip Ex. 1018, p. 445
Microchip v. Aptiv
IPR2024-00494

Offset	Field	Size	Description
			D6...D5: TT Think TIme 00: TT requires at most 8 FS bit times of inter transaction 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. See Section 11.5.3. D15...D8: Reserved
5	*bPwrOn2PwrGood*	1	Time (in 2 ms intervals) from the time the power-on sequence begins on a port until power is good on that port. The USB System Software uses this value to determine how long to wait before accessing a powered-on port.
6	*bHubContrCurrent*	1	Maximum current requirements of the Hub Controller electronics in mA.
7	*DeviceRemovable*	Variable, depending on number of ports on hub	Indicates if a port has a removable device attached. This field is reported on byte-granularity. Within a byte, if no port exists for a given location, the field representing the port characteristics returns 0. Bit value definition: 0B - Device is removable. 1B - Device is non-removable This is a bitmap corresponding to the individual ports on the hub: Bit 0: Reserved for future use. Bit 1: Port 1 Bit 2: Port 2 .... Bit *n*: Port *n* (implementation-dependent, up to a maximum of 255 ports).
Variable	*PortPwrCtrlMask*	Variable, depending on number of ports on hub	This field exists for reasons of compatibility with software written for 1.0 compliant devices. All bits in this field should be set to 1B. This field has one bit for each port on the hub with additional pad bits, if necessary, to make the number of bits in the field an integer multiple of 8.

Microchip Ex. 1018, p. 446
Microchip v. Aptiv
IPR2024-00494

11.24 Requests

11.24.1 Standard Requests

Hubs have tighter constraints on request processing timing than specified in Section 9.2.6 for standard devices because they are crucial to the "time to availability" of all devices attached to USB. The worst case request timing requirements are listed below (apply to both Standard and Hub Class requests):

1. Completion time for requests with no data stage: 50 ms
2. Completion times for standard requests with data stage(s)
   - Time from setup packet to first data stage: 50 ms
   - Time between each subsequent data stage: 50 ms
   - Time between last data stage and status stage: 50 ms

As hubs play such a crucial role in bus enumeration, it is recommended that hubs average response times be less than 5 ms for all requests.

Table 11-14 outlines the various standard device requests.

Table 11-14. Hub Responses to Standard Device Requests

bRequest	Hub Response
CLEAR_FEATURE	Standard
GET_CONFIGURATION	Standard
GET_DESCRIPTOR	Standard
GET_INTERFACE	Undefined. Hubs are allowed to support only one interface.
GET_STATUS	Standard
SET_ADDRESS	Standard
SET_CONFIGURATION	Standard
SET_DESCRIPTOR	Optional
SET_FEATURE	Standard
SET_INTERFACE	Undefined. Hubs are allowed to support only one interface.
SYNCH_FRAME	Undefined. Hubs are not allowed to have isochronous endpoints.

Optional requests that are not implemented shall return a STALL in the Data stage or Status stage of the request.

Microchip Ex. 1018, p. 447
Microchip v. Aptiv
IPR2024-00494

11.24.2 Class-specific Requests

The hub class defines requests to which hubs respond, as outlined in Table 11-15. Table 11-16 defines the hub class request codes. All requests in the table below except SetHubDescriptor() are mandatory.

Table 11-15. Hub Class Requests

Request	bmRequestType	bRequest	wValue	wIndex	wLength	Data
ClearHubFeature	00100000B	CLEAR_ FEATURE	Feature Selector	Zero	Zero	None
ClearPortFeature	00100011B	CLEAR_ FEATURE	Feature Selector	Selector, Port	Zero	None
ClearTTBuffer	00100011B	CLEAR_TT_BUFFER	Dev_Addr, EP_Num	TT_port	Zero	None
GetHubDescriptor	10100000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
GetHubStatus	10100000B	GET_ STATUS	Zero	Zero	Four	Hub Status and Change Status
GetPortStatus	10100011B	GET_ STATUS	Zero	Port	Four	Port Status and Change Status
ResetTT	00100011B	RESET_TT	Zero	Port	Zero	None
SetHubDescriptor	00100000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
SetHubFeature	00100000B	SET_ FEATURE	Feature Selector	Zero	Zero	None
SetPortFeature	00100011B	SET_ FEATURE	Feature Selector	Selector, Port	Zero	None
GetTTState	10100011B	GET_TT_STATE	TT_Flags	Port	TT State Length	TT State
StopTT	00100011B	STOP_TT	Zero	Port	Zero	None

Microchip Ex. 1018, p. 448
Microchip v. Aptiv
IPR2024-00494

Table 11-16. Hub Class Request Codes

bRequest	Value
GET_ STATUS	0
CLEAR_ FEATURE	1
RESERVED (used in previous specifications for GET_STATE)	2
SET_ FEATURE	3
Reserved for future use	*4-5*
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
CLEAR_TT_BUFFER	8
RESET_TT	9
GET_TT_STATE	10
STOP_TT	11

Table 11-17 gives the valid feature selectors for the hub class. See Section 11.24.2.6 and Section 11.24.2.7 for a description of the features.

Table 11-17. Hub Class Feature Selectors

	Recipient	Value
C_HUB_LOCAL_POWER	Hub	0
C_HUB_OVER_CURRENT	Hub	1
PORT_CONNECTION	Port	0
PORT_ENABLE	Port	1
PORT_SUSPEND	Port	2
PORT_OVER_CURRENT	Port	3
PORT_RESET	Port	4

Microchip Ex. 1018, p. 449
Microchip v. Aptiv
IPR2024-00494

Table 11-17. Hub Class Feature Selectors (continued)

	Recipient	Value
PORT_POWER	Port	8
PORT_LOW_SPEED	Port	9
C_PORT_CONNECTION	Port	16
C_PORT_ENABLE	Port	17
C_PORT_SUSPEND	Port	18
C_PORT_OVER_CURRENT	Port	19
C_PORT_RESET	Port	20
PORT_TEST	Port	21
PORT_INDICATOR	Port	22

11.24.2.1 Clear Hub Feature

This request resets a value reported in the hub status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100000B	CLEAR_ FEATURE	Feature Selector	Zero	Zero	None

Clearing a feature disables that feature; refer to Table 11-17 for the feature selector definitions that apply to the hub as a recipient. If the feature selector is associated with a status change, clearing that status change acknowledges the change. This request format is used to clear either the C_HUB_LOCAL_POWER or C_HUB_OVER_CURRENT features.

It is a Request Error if *wValue* is not a feature selector listed in Table 11-17 or if *wIndex* or *wLength* are not as specified above.

If the hub is not configured, the hub's response to this request is undefined.

11.24.2.2 Clear Port Feature

This request resets a value reported in the port status.

bmRequestType	bRequest	wValue	wIndex		wLength	Data
00100011B	CLEAR_ FEATURE	Feature Selector	Selector	Port	Zero	None

The port number must be a valid port number for that hub, greater than zero. The port field is located in bits 7..0 of the *wIndex* field.

Microchip Ex. 1018, p. 450
Microchip v. Aptiv
IPR2024-00494

Clearing a feature disables that feature or starts a process associated with the feature; refer to Table 11-17 for the feature selector definitions. If the feature selector is associated with a status change, clearing that status change acknowledges the change. This request format is used to clear the following features:

- PORT_ENABLE
- PORT_SUSPEND
- PORT_POWER
- PORT_INDICATOR
- C_PORT_CONNECTION
- C_PORT_RESET
- C_PORT_ENABLE
- C_PORT_SUSPEND
- C_PORT_OVER_CURRENT

Clearing the PORT_SUSPEND feature causes a host-initiated resume on the specified port. If the port is not in the Suspended state, the hub should treat this request as a functional no-operation.

Clearing the PORT_ENABLE feature causes the port to be placed in the Disabled state. If the port is in the Powered-off state, the hub should treat this request as a functional no-operation.

Clearing the PORT_POWER feature causes the port to be placed in the Powered-off state and may, subject to the constraints due to the hub's method of power switching, result in power being removed from the port. Refer to Section 11.11 on rules for how this request is used with ports that are gang-powered.

The selector field identifies the port indicator selector when clearing a port indicator. The selector field is in bits 15..8 of the *wIndex* field.

It is a Request Error if *wValue* is not a feature selector listed in Table 11-17, if *wIndex* specifies a port that does not exist, or if *wLength* is not as specified above. It is not an error for this request to try to clear a feature that is already cleared (hub should treat as a functional no-operation).

If the hub is not configured, the hub's response to this request is undefined.

11.24.2.3 Clear TT Buffer

This request clears the state of a Transaction Translator(TT) bulk/control buffer after it has been left in a busy state due to high-speed errors. This request is only defined for non-periodic endpoints; e.g., if it is issued for a periodic endpoint, the response is undefined. After successful completion of this request, the buffer can again be used by the TT with high-speed split transactions for full-/low-speed transactions to attached full-/low-speed devices.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100011B	CLEAR_TT_BUFFER	Device_Address, Endpoint_Number	TT_port	Zero	None

If the hub supports a TT per port, then wIndex must specify the port number of the TT that encountered the high-speed errors (e.g., with the busy TT buffer). If the hub provides only a single TT, then *wIndex* must be set to one.

Microchip Ex. 1018, p. 451
Microchip v. Aptiv
IPR2024-00494

The device_address, endpoint_number, and endpoint_type of the full-/low-speed endpoint (as specified in the corresponding split transaction) that may have a busy TT buffer must be specified in the wValue field. The specific bit fields used are shown in Table 11-18.

It is a Request Error if *wIndex* specifies a port that does not exist, or if *wLength* is not as specified above. It is not an error for this request to try to clear a buffer for a transaction that is not buffered by the TT (should treat as a functional no-operation).

If the hub is not configured, the hub's response to this request is undefined.

Table 11-18. wValue Field for Clear_TT_Buffer

Bits	Field
3..0	Endpoint Number
10..4	Device Address
12..11	Endpoint Type
14..13	Reserved, must be zero
15	Direction, 1 = IN, 0 = OUT

11.24.2.4 Get Bus State

Previous versions of USB defined a GetBusState request. This request is no longer defined.

11.24.2.5 Get Hub Descriptor

This request returns the hub descriptor.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero	Descriptor Length	Descriptor

The GetDescriptor() request for the hub class descriptor follows the same usage model as that of the standard GetDescriptor() request (refer to Chapter 9). The standard hub descriptor is denoted by using the value *bDescriptorType* defined in Section 11.23.2.1. All hubs are required to implement one hub descriptor, with descriptor index zero.

If *wLength* is larger than the actual length of the descriptor, then only the actual length is returned. If *wLength* is less than the actual length of the descriptor, then only the first *wLength* bytes of the descriptor are returned; this is not considered an error even if *wLength* is zero.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

If the hub is not configured, the hub's response to this request is undefined.

Microchip Ex. 1018, p. 452
Microchip v. Aptiv
IPR2024-00494

11.24.2.6 Get Hub Status

This request returns the current hub status and the states that have changed since the previous acknowledgment.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100000B	GET_ STATUS	Zero	Zero	Four	Hub Status and Change Status

The first word of data contains *wHubStatus* (refer to Table 11-19). The second word of data contains *wHubChange* (refer to Table 11-20).

It is a Request Error if *wValue*, *wIndex*, or *wLength* are other than as specified above.

If the hub is not configured, the hub's response to this request is undefined.

Table 11-19. Hub Status Field, *wHubStatus*

Bit	Description
0	**Local Power Source:** This is the source of the local power supply. This field indicates whether hub power (for other than the SIE) is being provided by an external source or from the USB. This field allows the USB System Software to determine the amount of power available from a hub to downstream devices. 0 = Local power supply good 1 = Local power supply lost (inactive)
1	**Over-current:** If the hub supports over-current reporting on a hub basis, this field indicates that the sum of all the ports' current has exceeded the specified maximum and all ports have been placed in the Powered-off state. If the hub reports over-current on a per-port basis or has no over-current detection capabilities, this field is always zero. For more details on over-current protection, see Section 7.2.1.2.1. 0 = No over-current condition currently exists. 1 = A hub over-current condition exists.
2-15	**Reserved** These bits return 0 when read.

There are no defined feature selector values for these status bits and they can neither be set nor cleared by the USB System Software.

Microchip Ex. 1018, p. 453
Microchip v. Aptiv
IPR2024-00494

Table 11-20. Hub Change Field, *wHubChange*

Bit	Description
0	**Local Power Status Change:** (C_HUB_LOCAL_POWER) This field indicates that a change has occurred in the hub's Local Power Source field in *wHubStatus*. This field is initialized to zero when the hub receives a bus reset. 0 = No change has occurred to Local Power Status. 1 = Local Power Status has changed.
1	**Over-Current Change**: (C_HUB_OVER_CURRENT) This field indicates if a change has occurred in the Over-Current field in *wHubStatus*. This field is initialized to zero when the hub receives a bus reset. 0 = No change has occurred to the Over-Current Status. 1 = Over-Current Status has changed.
2-15	**Reserved** These bits return 0 when read.

Hubs may allow setting of these change bits with SetHubFeature() requests in order to support diagnostics. If the hub does not support setting of these bits, it should either treat the SetHubFeature() request as a Request Error or as a functional no-operation. When set, these bits may be cleared by a ClearHubFeature() request. A request to set a feature that is already set or to clear a feature that is already clear has no effect and the hub will not fail the request.

11.24.2.7 Get Port Status

This request returns the current port status and the current value of the port status change bits.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100011B	GET_STATUS	Zero	Port	Four	Port Status and Change Status

The port number must be a valid port number for that hub, greater than zero.

The first word of data contains *wPortStatus* (refer to Table 11-21). The second word of data contains *wPortChange* (refer to Table 11-20).

The bit locations in the *wPortStatus* and *wPortChange* fields correspond in a one-to-one fashion where applicable.

It is a Request Error if *wValue* or *wLength* are other than as specified above or if *wIndex* specifies a port that does not exist.

If the hub is not configured, the behavior of the hub in response to this request is undefined.

Microchip Ex. 1018, p. 454
Microchip v. Aptiv
IPR2024-00494

11.24.2.7.1 Port Status Bits

Table 11-21. Port Status Field, *wPortStatus*

Bit	Description
0	**Current Connect Status:** (PORT_CONNECTION) This field reflects whether or not a device is currently connected to this port. 0 = No device is present. 1 = A device is present on this port.
1	**Port Enabled/Disabled:** (PORT_ENABLE) Ports can be enabled by the USB System Software only. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by the USB System Software. 0 = Port is disabled. 1 = Port is enabled.
2	**Suspend:** (PORT_SUSPEND) This field indicates whether or not the device on this port is suspended. Setting this field causes the device to suspend by not propagating bus traffic downstream. This field may be reset by a request or by resume signaling from the device attached to the port. 0 = Not suspended. 1 = Suspended or resuming.
3	**Over-current:** (PORT_OVER_CURRENT) If the hub reports over-current conditions on a per-port basis, this field will indicate that the current drain on the port exceeds the specified maximum. For more details, see Section 7.2.1.2.1. 0 = All no over-current condition exists on this port. 1 = An over-current condition exists on this port.
4	**Reset**: (PORT_RESET) This field is set when the host wishes to reset the attached device. It remains set until the reset signaling is turned off by the hub. 0 = Reset signaling not asserted. 1 = Reset signaling asserted.
5-7	**Reserved** These bits return 0 when read.
8	**Port Power:** (PORT_POWER) This field reflects a port's logical, power control state. Because hubs can implement different methods of port power switching, this field may or may not represent whether power is applied to the port. The device descriptor reports the type of power switching implemented by the hub. 0 = This port is in the Powered-off state. 1 = This port is not in the Powered-off state.
9	**Low- Speed Device Attached:** (PORT_LOW_SPEED) This is relevant only if a device is attached. 0 = Full-speed or High-speed device attached to this port (determined by bit 10). 1 = Low-speed device attached to this port.
10	**High-speed Device Attached:** (PORT_HIGH_SPEED) This is relevant only if a device is attached. 0 = Full-speed device attached to this port. 1 = High-speed device attached to this port.
11	**Port Test Mode** : (PORT_TEST) This field reflects the status of the port's test mode. Software uses the SetPortFeature() and ClearPortFeature() requests to manipulate the port test mode. 0 = This port is not in the Port Test Mode. 1 = This port is in Port Test Mode.
12	**Port Indicator Control**: (PORT_INDICATOR) This field is set to reflect software control of the port indicator. For more details see Sections 11.5.3, 11.24.2.7.1.10, and 11.24.2.13. 0 = Port indicator displays default colors. 1 = Port indicator displays software controlled color.
13-15	**Reserved** These bits return 0 when read.

Microchip Ex. 1018, p. 455
Microchip v. Aptiv
IPR2024-00494

11.24.2.7.1.1 PORT_CONNECTION

When the Port Power bit is one, this bit indicates whether or not a device is attached. This field reads as one when a device is attached; it reads as zero when no device is attached. This bit is reset to zero when the port is in the Powered-off state or the Disconnected states. It is set to one when the port is in the Powered state, a device attach is detected (see Section 7.1.7.3), and the port transitions from the Disconnected state to the Disabled state.

SetPortFeature(PORT_CONNECTION) and ClearPortFeature(PORT_CONNECTION) requests shall not be used by the USB System Software and must be treated as no-operation requests by hubs.

11.24.2.7.1.2 PORT_ENABLE

This bit is set when the port is allowed to send or receive packet data or resume signaling.

This bit may be set only as a result of a SetPortFeature(PORT_RESET) request. When the hub exits the Resetting state or, if present, the Speed_eval state, this bit is set and bus traffic may be transmitted to the port. This bit may be cleared as the result of any of the following:

- The port being in the Powered-off state
- Receipt of a ClearPortFeature(PORT_ENABLE) request
- Port_Error detection
- Disconnect detection
- When the port enters the Resetting state as a result of receiving the SetPortFeature(PORT_RESET) request

The hub response to a SetPortFeature(PORT_ENABLE) request is not specified. The preferred behavior is that the hub respond with a Request Error. This may not be used by the USB System Software. The ClearPortFeature(PORT_ENABLE) request is supported as specified in Section 11.5.1.4.

11.24.2.7.1.3 PORT_SUSPEND

This bit is set to one when the port is selectively suspended by the USB System Software. While this bit is set, the hub does not propagate downstream-directed traffic to this port, but the hub will respond to resume signaling from the port. This bit can be set only if the port's PORT_ENABLE bit is set and the hub receives a SetPortFeature(PORT_SUSPEND) request. This bit is cleared to zero on the transition from the SendEOP state to the Enabled state, or on the transition from the Restart_S state to the Transmit state, or on any event that causes the PORT_ENABLE bit to be cleared while the PORT_SUSPEND bit is set.

The SetPortFeature(PORT_SUSPEND) request may be issued by the USB System Software at any time but will have an effect only as specified in Section 11.5.

11.24.2.7.1.4 PORT_OVER-CURRENT

This bit is set to one while an over-current condition exists on the port. This bit is cleared when an over-current condition does not exist on the port.

If the voltage on this port is affected by an over-current condition on another port, this bit is set and remains set until the over-current condition on the affecting port is removed. When the over-current condition on the affecting port is removed, this bit is reset to zero if an over-current condition does not exist on this port.

Over-current protection is required on self-powered hubs (it is optional on bus-powered hubs) as outlined in Section 7.2.1.2.1.

The SetPortFeature(PORT_OVER_CURRENT) and ClearPortFeature(PORT_OVER_CURRENT) requests shall not be used by the USB System Software and may be treated as no-operation requests by hubs.

Microchip Ex. 1018, p. 456
Microchip v. Aptiv
IPR2024-00494

11.24.2.7.1.5 PORT_RESET

This bit is set while the port is in the Resetting state. A SetPortFeature(PORT_RESET) request will initiate the Resetting state if the conditions in Section 11.5.1.5 are met. This bit is set to zero while the port is in the Powered-off state.

The ClearPortFeature(PORT_RESET) request shall not be used by the USB System Software and may be treated as a no-operation request by hubs.

11.24.2.7.1.6 PORT_POWER

This bit reflects the current power state of a port. This bit is implemented on all ports whether or not actual port power switching devices are present.

While this bit is zero, the port is in the Powered-off state. Similarly, anything that causes this port to go to the Power-off state will cause this bit to be set to zero.

A SetPortFeature(PORT_POWER) will set this bit to one unless both C_HUB_LOCAL_POWER and Local Power Status (in *wHubStatus*) are set to one in which case the request is treated as a functional no-operation.

This bit may be cleared under the following curcumstances:

- Hub receives a ClearPortFeature(PORT_POWER).
- An over-current condition exists on the port.
- An over-current condition on another port causes the power on this port to be shut off.

The SetPortFeature(PORT_POWER) and ClearPortFeature(PORT_POWER) requests may be issued by the USB System Software whenever the port is not in the Not Configured state, but will have an effect only as specified in Section 11.11.

11.24.2.7.1.7 PORT_LOW_SPEED

This bit has meaning only when the PORT_ENABLE bit is set. This bit is set to one if the attached device is low-speed. If this bit is set to zero, the attached device is either full- or high-speed as determined by bit 10 (PORT_HIGH_SPEED, see below).

The SetPortFeature(PORT_LOW_SPEED) and ClearPortFeature(PORT_LOW_SPEED) requests shall not be used by the USB System Software and may be treated as no-operation requests by hubs.

11.24.2.7.1.8 PORT_HIGH_SPEED

This bit has meaning only when the PORT_ENABLE bit is set and the PORT_LOW_SPEED bit is set to zero. This bit is set to one if the attached device is high-speed. The bit is set to zero if the attached device is full-speed.

The SetPortFeature(PORT_HIGH_SPEED) and ClearPortFeature(PORT_HIGH_SPEED) requests shall not be used by the USB System Software and may be treated as no-operation requests by hubs.

11.24.2.7.1.9 PORT_TEST

When the Port Test Mode bit is set to a one (1B), the port is in test mode. The specific test mode is specified in the SetPortFeature(PORT_TEST) request by the test selector. The hub provides no standard mechanism to report the specific test mode; therefore, system software must track which test selector was used. Refer to Section 7.1.20 for details on each test mode. See Section 11.24.2.13 for more information about using SetPortFeature to control test mode.

Microchip Ex. 1018, p. 457
Microchip v. Aptiv
IPR2024-00494

This field may only be set as a result of a SetPortFeature(PORT_TEST) request. A port PORT_TEST request is only valid to a port that is in the Disabled, Disconnected, or Suspended states. If the port is not in one of these states, the hub must respond with a request error.

This field may be cleared as a result of resetting the hub.

11.24.2.7.1.10 PORT_INDICATOR

When the Port Indicator port status is set to a (1B), the port indicator selector is non-zero. The specific indicator mode is specified in the SetPortFeature(PORT_INDICATOR) request by the indicator selector. The GetPortStatus(PORT_INDICATOR) provides no standard mechanism to report a specific indicator mode; therefore, system software must track which indicator mode was used. Refer to Sections 11.5.3 and 11.24.2.13 for details on each indicator mode.

This field may only be set as a result of a SetPortFeature(PORT_INDICATOR) request.

This field may be cleared as a result of a SetPortFeature(PORT_INDICATOR) request with Indicator Selector = Default or a ClearPortFeature(PORT_INDICATOR) request.

This feature must be set when host software detects an error on a port that requires user intervention. This feature must be utilized by system software if it determines that any of the following conditions are true:

- A high power device is plugged into a low power port.
- A defective device is plugged into a port (Babble conditions, excessive errors, etc.).
- An overcurrent condition occurs which causes software or hardware to set the indicator.

The PORT_OVER_CURRENT status bit will set the default port indicator color to amber. Setting the PORT_POWER feature, sets the indicator to off.

This feature is also used when host software determines that a port requires user attention. Many error conditions can be resolved if the user moves a device from one port to another that has the proper capabilities.

A typical scenario is when a user plugs a high power device in to a bus-powered hub. If there is an available high power port, then the user can be directed to move the device from the low-power port to the high power port.

1. Host software would cycle the PORT_INDICATOR feature of the low-power port to blink the indicator and display a message to the user to unplug the device from the port with the blinking indicator.
2. Using the C_PORT_CONNECTION status change feature, host software can determine when the user physically removed the device from the low-power port.
3. Host software would next issue a ClearPortFeature(PORT_INDICATOR) to the low-power port (restoring the default color), begin cycling the PORT_INDICATOR of the high-power port, and display a message telling the user to plug the device into the port with the blinking indicator.
4. Using the C_PORT_CONNECTION status change feature host software can determine when the user physically inserted the device onto the high power port.

Host software must cycle the PORT_INDICATOR feature to blink the current color at approximately 0.5 Hz rate with a 30-50% duty cycle.

Microchip Ex. 1018, p. 458
Microchip v. Aptiv
IPR2024-00494

11.24.2.7.2 Port Status Change Bits

Port status change bits are used to indicate changes in port status bits that are not the direct result of requests. Port status change bits can be cleared with a ClearPortFeature() request or by a hub reset. Hubs may allow setting of the status change bits with a SetPortFeature() request for diagnostic purposes. If a hub does not support setting of the status change bits, it may either treat the request as a Request Error or as a functional no-operation. Table 11-22 describes the various bits in the *wPortChange* field.

Table 11-22. Port Change Field, *wPortChange*

Bit	Description
0	**Connect Status Change:** (C_PORT_CONNECTION) Indicates a change has occurred in the port's Current Connect Status. The hub device sets this field as described in Section 11.24.2.7.2.1. 0 = No change has occurred to Current Connect status. 1 = Current Connect status has changed.
1	**Port Enable/Disable Change:** (C_PORT_ENABLE) This field is set to one when a port is disabled because of a Port_Error condition (see Section 11.8.1).
2	**Suspend Change:** (C_PORT_SUSPEND) This field indicates a change in the host-visible suspend state of the attached device. It indicates the device has transitioned out of the Suspend state. This field is set only when the entire resume process has completed. That is, the hub has ceased signaling resume on this port. 0 = No change. 1 = Resume complete.
3	**Over-Current Indicator Change:** (C_PORT_OVER_CURRENT) This field applies only to hubs that report over-current conditions on a per-port basis (as reported in the hub descriptor). 0 = No change has occurred to Over-Current Indicator. 1 = Over-Current Indicator has changed. If the hub does not report over-current on a per-port basis, then this field is always zero.
4	**Reset Change:** (C_PORT_RESET) This field is set when reset processing on this port is complete. 0 = No change. 1 = Reset complete.
5-15	**Reserved** These bits return 0 when read.

11.24.2.7.2.1 C_PORT_CONNECTION

This bit is set when the PORT_CONNECTION bit changes because of an attach or detach detect event (see Section 7.1.7.3). This bit will be cleared to zero by a ClearPortFeature(C_PORT_CONNECTION) request or while the port is in the Powered-off state.

11.24.2.7.2.2 C_PORT_ENABLE

This bit is set when the PORT_ENABLE bit changes from one to zero as a result of a Port Error condition (see Section 11.8.1). This bit is not set on any other changes to PORT_ENABLE.

This bit may be set if, on a SetPortFeature(PORT_RESET), the port stays in the Disabled state because an invalid idle state exists on the bus (see Section 11.8.2).

This bit will be cleared by a ClearPortFeature(C_PORT_ENABLE) request or while the port is in the Powered-off state.

Microchip Ex. 1018, p. 459
Microchip v. Aptiv
IPR2024-00494

11.24.2.7.2.3 C_PORT_SUSPEND

This bit is set on the following transitions:

- On transition from the Resuming state to the SendEOP state
- On transition from the Restart_S state to the Transmit state

This bit will be cleared by a ClearPortFeature(C_PORT_SUSPEND) request, or while the port is in the Powered-off state.

11.24.2.7.2.4 C_PORT_OVER-CURRENT

This bit is set when the PORT_OVER_CURRENT bit changes from zero to one or from one to zero. This bit is also set if the port is placed in the Powered-off state due to an over-current condition on another port.

This bit will be cleared when the port is in the Not Configured state or by a ClearPortFeature(C_PORT_OVER_CURRENT) request.

11.24.2.7.2.5 C_PORT_RESET

This bit is set when the port transitions from the Resetting state (or, if present, the Speed_eval state) to the Enabled state.

This bit will be cleared by a ClearPortFeature(C_PORT_RESET) request, or while the port is in the Powered-off state.

11.24.2.8 Get_TT_State

This request returns the internal state of the transaction translator in a vendor specific format. A TT receiving this request must have first been stopped via the Stop_TT request. This request is provided for debugging purposes.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100011B	GET_TT_STATE	TT_Flags	TT_Port	TT State Length	TT State

The TT_Flags bits 7..0 are reserved for future USB definition and must be set to zero. The TT_Flags bits 15..8 are for vendor specific usage.

The TT state returned in the data stage of the control transfer for this request is shown in Table 11-23.

Table 11-23. Format of Returned TT State

Offset	Field	Size (bytes)	Comments
0	TT_Return_Flags	4	Bits 15..0 are reserved for future USB definition and must be set to zero. Bits 31..16 are for vendor specific usage.
4	TT_specific_state	Implementation dependent	

Microchip Ex. 1018, p. 460
Microchip v. Aptiv
IPR2024-00494

If the hub supports multiple TTs, then *wIndex* must specify the port number of the TT that will return TT_state. If the hub provides only a single TT, then Port must be set to one.

The state of the TT after processing this request is undefined.

It is a Request Error, if *wIndex* specifies a port that does not exist. If *wLength* is larger than the actual length of this request, then only the actual length is returned. If *wLength* is less than the actual length of this request, then only the first *wLength* bytes of this request are returned; this is not considered an error even if *wLength* is zero.

If the hub is not configured, the hub's response to this request is undefined.

11.24.2.9 Reset_TT

This request returns the transaction translator in a hub to a known state.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100011B	RESET_TT	Zero	TT_Port	Zero	None

Under some circumstances, a Transaction Translator (TT) in a hub may be in an unknown state such that it is no longer functioning correctly. The Reset_TT request allows the TT to be returned to the state it is in immediately after the hub is configured. Reset_TT only resets the TT internal data structures (buffers) and pipeline and its related state machines. After the reset is completed, the TT can resume its normal operation. Reset of the TT is de-coupled from the other parts of the hub (including downstream facing ports of the hub, the hub repeater, the hub controller, etc). Other parts of the hub are not reset and can continue their normal operation. The downstream facing ports are not reset, so that when the TT resumes its normal operation, the corresponding attached devices continue to work; i.e., a new enumeration process is not required. The working of downstream FS/LS devices are disrupted only during the reset time of the TT to which they belong.

If the hub supports multiple TTs, then *wIndex* must specify the port number of the TT that is to be reset. If the hub provides only a single TT, then Port must be set to one. For a single TT Hub, the Hub can ignore the Port number.

It is a Request Error, if *wIndex* specifies a port that does not exist, or if *wLength* is not as specified above.

If the hub is not configured, the hub's response to this request is undefined.

11.24.2.10 Set Hub Descriptor

This request overwrites the hub descriptor.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero	Descriptor Length	Descriptor

The SetDescriptor request for the hub class descriptor follows the same usage model as that of the standard SetDescriptor request (refer to Chapter 9). The standard hub descriptor is denoted by using the value *bDescriptorType* defined in Section 11.23.2.1. All hubs are required to implement one hub descriptor with descriptor index zero.

Microchip Ex. 1018, p. 461
Microchip v. Aptiv
IPR2024-00494

This request is optional. This request writes data to a class-specific descriptor. The host provides the data that is to be transferred to the hub during the data transfer phase of the control transaction. This request writes the entire hub descriptor at once.

Hubs must buffer all the bytes received from this request to ensure that the entire descriptor has been successfully transmitted from the host. Upon successful completion of the bus transfer, the hub updates the contents of the specified descriptor.

It is a Request Error if *wIndex* is not zero or if *wLength* does not match the amount of data sent by the host. Hubs that do not support this request respond with a STALL during the Data stage of the request.

If the hub is not configured, the hub's response to this request is undefined.

11.24.2.11 Stop_TT

This request stops the normal execution of the transaction translator so that the internal TT state can be retrieved via Get_TT_State. This request is provided for debugging purposes.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100011B	STOP_TT	Zero	TT_Port	Zero	None

The only standardized method to restart a TT after a Stop_TT request is via the Reset_TT request.

If the hub supports multiple TTs, then *wIndex* must specify the port number of the TT that is being stopped. If the hub provides only a single TT, then Port must be set to one. For a single TT Hub, the Hub can ignore the Port number.

It is a Request Error, if *wIndex* specifies a port that does not exist, or if *wLength* is not as specified above.

If the hub is not configured, the hub's response to this request is undefined.

11.24.2.12 Set Hub Feature

This request sets a value reported in the hub status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100000B	SET_ FEATURE	Feature Selector	Zero	Zero	None

Setting a feature enables that feature; refer to Table 11-17 for the feature selector definitions that apply to the hub as recipient. Status changes may not be acknowledged using this request.

It is a Request Error if *wValue* is not a feature selector listed in Table 11-17 or if *wIndex* or *wLength* are not as specified above.

If the hub is not configured, the hub's response to this request is undefined.

Microchip Ex. 1018, p. 462
Microchip v. Aptiv
IPR2024-00494

11.24.2.13 Set Port Feature

This request sets a value reported in the port status.

bmRequestType	bRequest	wValue	wIndex		wLength	Data
00100011B	SET_ FEATURE	Feature Selector	Selector	Port	Zero	None

The port number must be a valid port number for that hub, greater than zero. The port number is in the least significant byte (bits 7..0) of the *wIndex* field. The most significant byte of *wIndex* is zero, except when the feature selector is PORT_TEST.

Setting a feature enables that feature or starts a process associated with that feature; see Table 11-17 for the feature selector definitions that apply to a port as a recipient. Status change may not be acknowledged using this request. Features that can be set with this request are:

- PORT_RESET
- PORT_SUSPEND
- PORT_POWER
- PORT_TEST
- PORT_INDICATOR
- C_PORT_CONNECTION*
- C_PORT_RESET*
- C_PORT_ENABLE*
- C_PORT_SUSPEND*
- C_PORT_OVER_CURRENT*

* Denotes features that are not required to be set by this request

Setting the PORT_SUSPEND feature causes bus traffic to cease on that port and, consequently, the device to suspend. Setting the reset feature PORT_RESET causes the hub to signal reset on that port. When the reset signaling is complete, the hub sets the C_PORT_RESET status change and immediately enables the port. Also see Section 11.24.2.7.1 for further details.

When the feature selector is PORT_TEST, the most significant byte (bits 15..8) of the *wIndex* field is the selector identifying the specific test mode. Table 11-24 lists the test selector definitions. Refer to Section 7.1.20 for definitions of each test mode. Test mode of a downstream facing port can only be used in a well defined sequence of hub states. This sequence is defined as follows:

1) All enabled downstream facing ports of the hub containing the port to be tested must be (selectively) suspended via the SetPortFeature(PORT_SUSPEND) request. Each downstream facing port of the hub must be in the disabled, disconnected, or suspended state (see Figure 11-9).

2) A SetPortFeature(PORT_TEST) request must be issued to the downstream facing port to be tested. Only a single downstream facing port can be in test_mode at a time. The transition to test mode must be complete no later than 3 ms after the completion of the status stage of the request.

3) The downstream facing port under test can now be tested.

4) During test_mode, a port disconnect or resume status change on one of the suspended ports (not including the port under test) must cause a status change (C_PORT_CONNECTION or C_PORT_SUSPEND) report (See Section 11.12.3 and 11.24.2.7.2) from the hub. Note: Other

Microchip Ex. 1018, p. 463
Microchip v. Aptiv
IPR2024-00494

status changes may or may not be supported in a hub with a downstream facing port in test mode. The reporting of these status changes can allow a test application to restore normal operation of a root hub without requiring a non-USB keyboard or mouse for user input. For example. a USB device attached to the root hub can be disconnected to notify the test application to restore normal root hub operation.

5) During test_mode, the state of the hub downstream facing ports must not be changed by the host (i.e., hub class requests other than the Get_Port_Status() request must not be issued by the host). Note: The hub must also be reset before a SetPortFeature(PORT_TEST) can be used to place the port into another test mode.

6) After the test is completed, the hub (with the port under test) must be reset by the host or user. This must be accomplished by manipulating the port of the parent hub to which the hub under test is attached. This manipulation can consist of one of the following:

   a) Issuing a SetPortFeature(PORT_RESET) to port of the parent hub to which the hub under test is attached.

   b) Issuing a ClearPortFeature(PORT_POWER) and SetPortFeature(PORT_POWER) to cycle power of a parent hub that supports per port power control.

   c) Disconnecting and re-connecting the hub under test from its parent hub port.

   d) For a root hub under test, a reset of the Host Controller may be required as there is no parent hub of the root hub.

7) Behavior of the hub under test and its downstream facing ports is undefined if these requirements are not met.

Table 11-24. Test Mode Selector Codes

Value	Test Mode Description
0H	Reserved
1H	Test_J
2H	Test_K
3H	Test_SE0_NAK
4H	Test_Packet
5H	Test_Force_Enable
06H-3FH	Reserved for Standard Test selections
40H-BFH	Reserved
C0H-FFH	Reserved for Vendor-Unique test selections

Microchip Ex. 1018, p. 464
Microchip v. Aptiv
IPR2024-00494

When the feature selector is PORT_INDICATOR, the most significant byte of the *wIndex* field is the selector identifying the specific indicator mode. Table 11-25 lists the indicator selector definitions. Refer to Sections 11.5.3 and 11.24.2.7.1.10 for indicator details. The hub will respond with a request error if the request contains an invalid indicator selector.

Table 11-25. Port Indicator Selector Codes

Value	Port Indicator Color	Port Indicator Mode
0	Color set automatically, as defined in Table 11-6	Automatic
1	Amber	Manual
2	Green	
3	Off	
4-FFH	Reserved	Reserved

The hub must meet the following requirements:

- If the port is in the Powered-off state, the hub must treat a SetPortFeature(PORT_RESET) request as a functional no-operation.
- If the port is not in the Enabled or Transmitting state, the hub must treat a SetPortFeature(PORT_SUSPEND) request as a functional no-operation.
- If the port is not in the Powered-off state, the hub must treat a SetPortFeature(PORT_POWER) request as a functional no-operation.

It is a Request Error if *wValue* is not a feature selector listed in Table 11-17, if *wIndex* specifies a port that does not exist, or if *wLength* is not as specified above.

If the hub is not configured, the hub's response to this request is undefined.

Microchip Ex. 1018, p. 465
Microchip v. Aptiv
IPR2024-00494

Microchip Ex. 1018, p. 466
Microchip v. Aptiv
IPR2024-00494

Appendix A
Transaction Examples

This appendix contains transaction examples for different split transaction cases. The cases are for bulk/control OUT and SETUP, bulk/control IN, interrupt OUT, interrupt IN, isochronous OUT, and isochronous IN.

A.1 Bulk/Control OUT and SETUP Transaction Examples

Legend:

(S): Start Split

(C): Complete Split

Summary of cases for bulk/control OUT and SETUP transaction

- Normal cases

Case	Reference Figure	Similar Figure
No smash	Figure A-1	
HS SSPLIT smash		Figure A-2
HS SSPLIT 3 strikes smash		Figure A-3
HS OUT/SETUP(S) smash		Figure A-2
HS OUT/SETUP(S) 3 strikes smash		Figure A-3
HS DATA0/1 smash	Figure A-2	
HS DATA0/1 3 strikes smash	Figure A-3	
HS ACK(S) smash	Figure A-4 Figure A-5	
HS ACK(S) 3 strikes smash	Figure A-6	
HS CSPLIT smash	Figure A-7	
HS CSPLIT 3 strikes smash	Figure A-8	
HS OUT/SETUP(C) smash		Figure A-7
HS OUT/SETUP(C) 3 strikes smash		Figure A-8

Microchip Ex. 1018, p. 467
Microchip v. Aptiv
IPR2024-00494

HS ACK(C) smash	Figure A-9	
HS ACK(C) 3 strikes smash	Figure A-10	
FS/LS OUT/SETUP smash		Figure A-11
FS/LS OUT/SETUP 3 strikes smash		Figure A-12
FS/LS DATA0/1 smash	Figure A-11	
FS/LS DATA0/1 3 strikes smash	Figure A-12	
FS/LS ACK smash	Figure A-13	
FS/LS ACK 3 strikes smash	Figure A-14	

- No buffer(on hub) avallable cases

Case	Reference Figure	Similar Figure
No smash(HS NAK(S))	Figure A-15	
HS NAK(S) smash	Figure A-16	
HS NAK(S) 3 strikes smash	Figure A-17	

- CS(Complete-split transaction) earlier cases

Case	Reference Figure	Similar Figure
No smash(HS NYET)	Figure A-18	
HS NYET smash	Figure A-19 Figure A-20	
HS NYET 3 strikes smash	Figure A-21	

- Device busy cases

Case	Reference Figure	Similar Figure
No smash(HS NAK(C))	Figure A-22	
HS NAK(C) smash		Figure A-9

Microchip Ex. 1018, p. 468
Microchip v. Aptiv
IPR2024-00494

HS NAK(C) 3 strikes smash		Figure A-10
FS/LS NAK smash		Figure A-13
FS/LS NAK 3 strikes smash		Figure A-14

- Device stall cases

Case	Reference Figure	Similar Figure
No smash	Figure A-23	
HS STALL(C) smash		Figure A-9
HS STALL(C) 3 strikes smash		Figure A-10
FS/LS STALL smash		Figure A-13
FS/LS STALL 3 strikes smash		Figure A-14

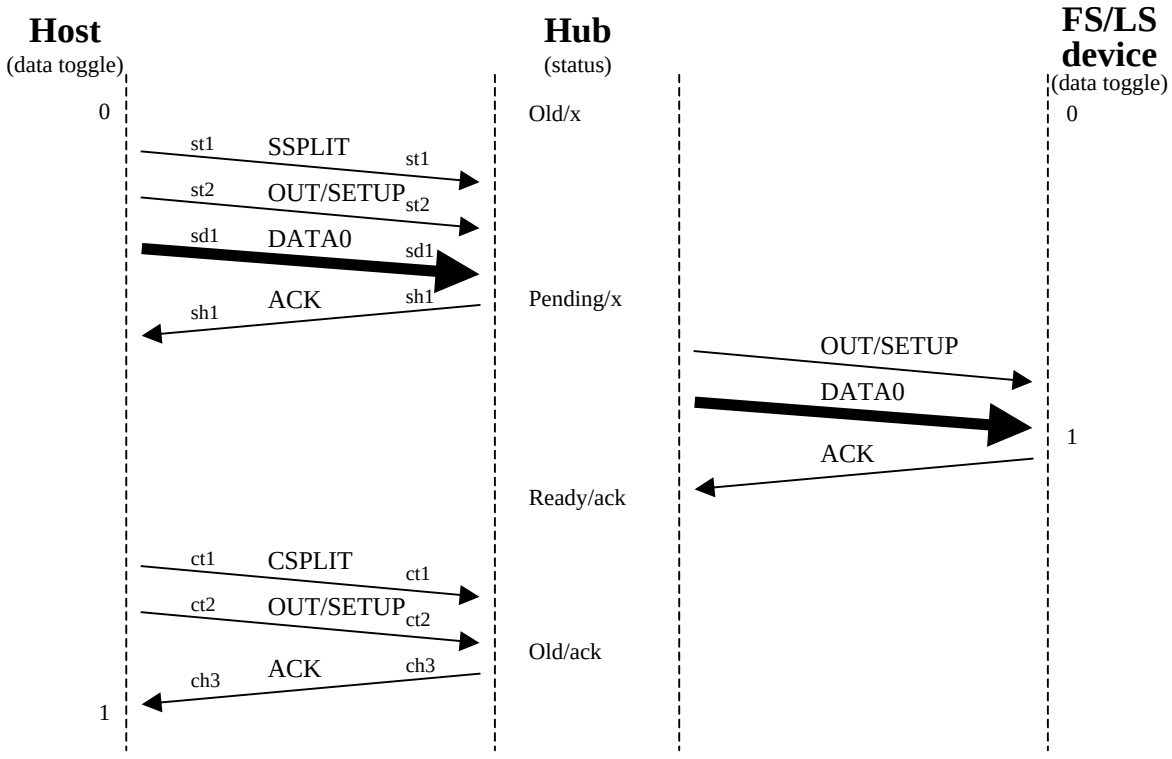


Figure A-1. Normal No Smash

Microchip Ex. 1018, p. 469
Microchip v. Aptiv
IPR2024-00494

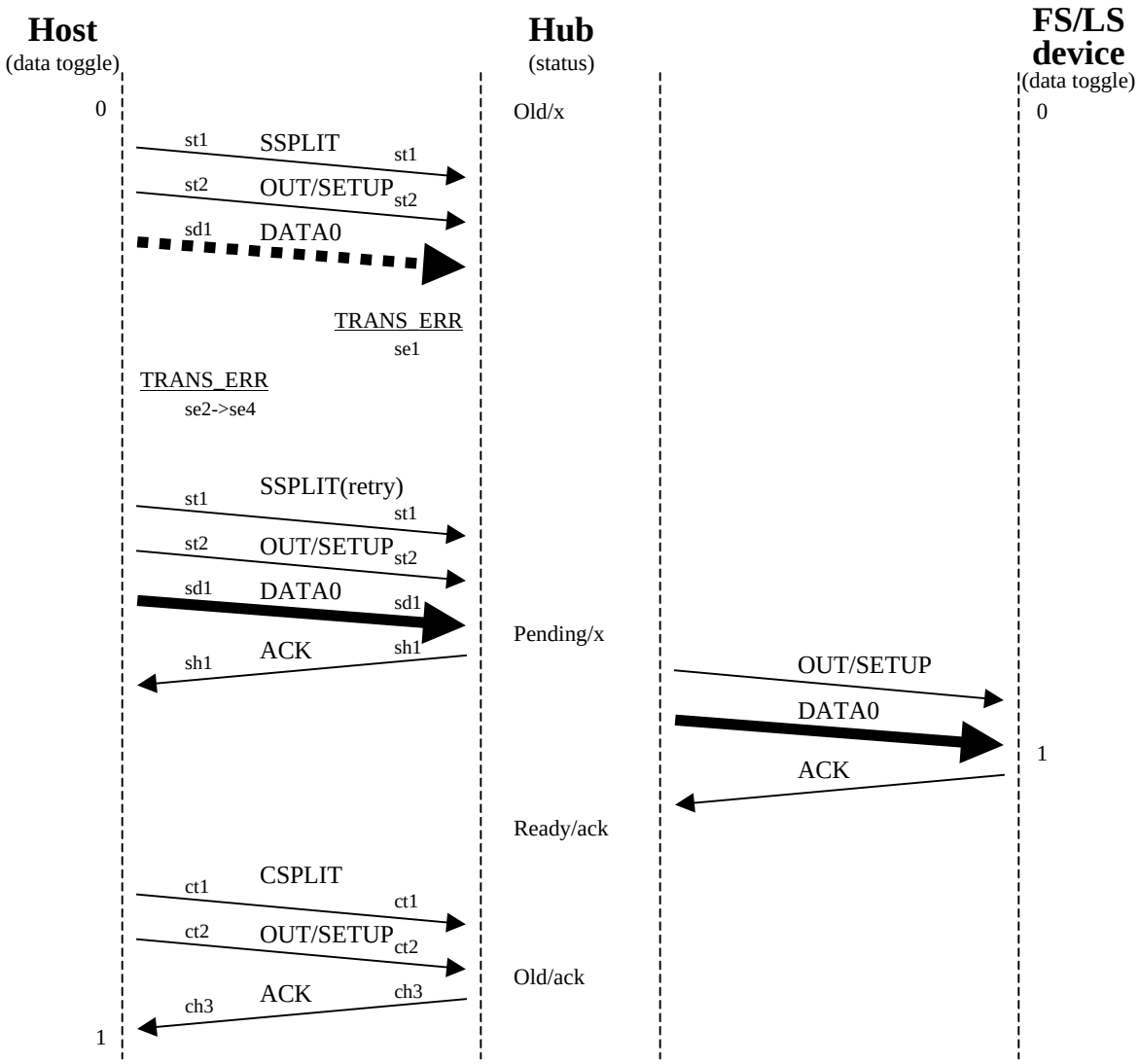


Figure A-2. Normal HS DATA0/1 Smash

Microchip Ex. 1018, p. 470
Microchip v. Aptiv
IPR2024-00494

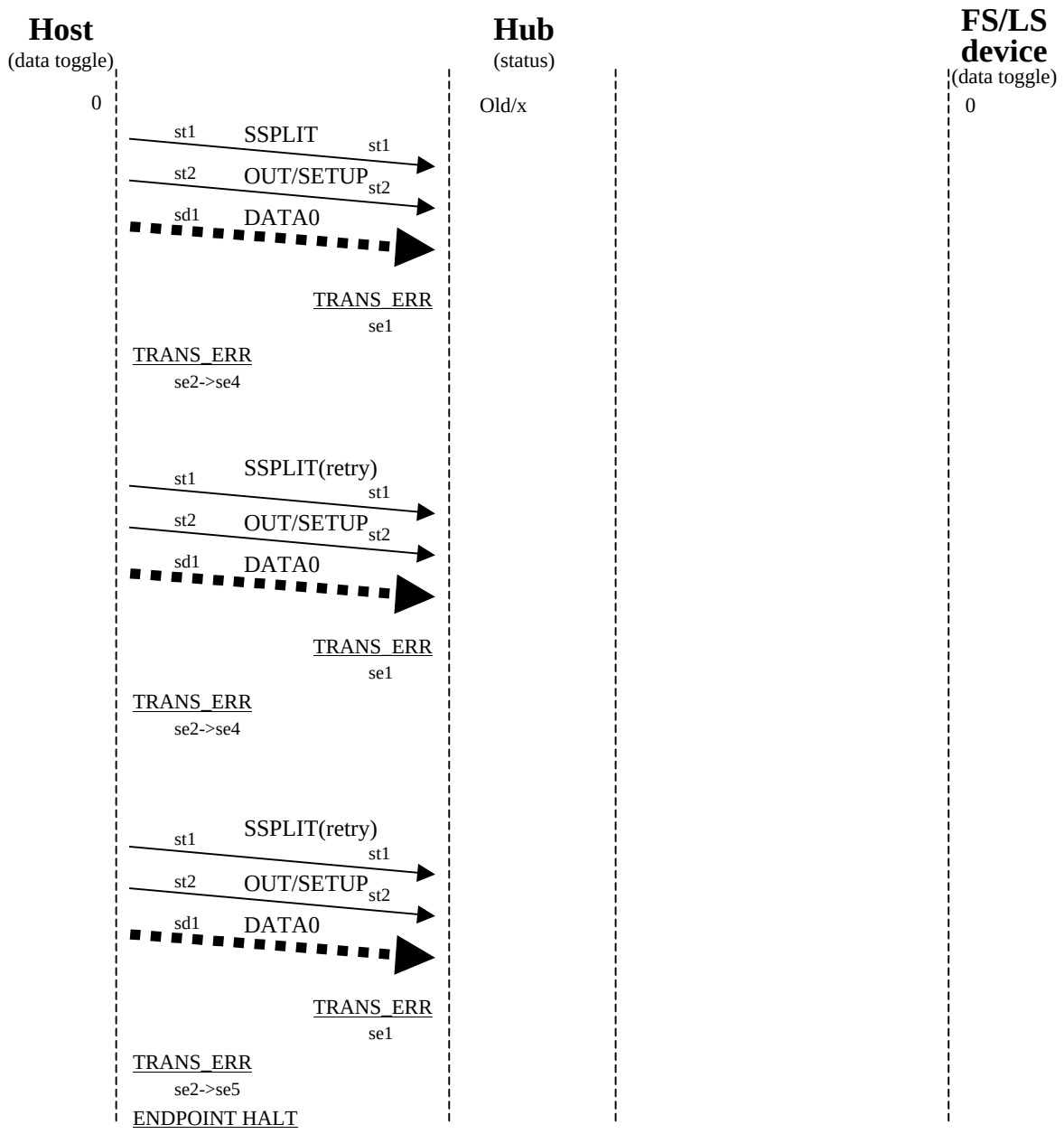


Figure A-3. Normal HS DATA0/1 3 Strikes Smash

Microchip Ex. 1018, p. 471
Microchip v. Aptiv
IPR2024-00494

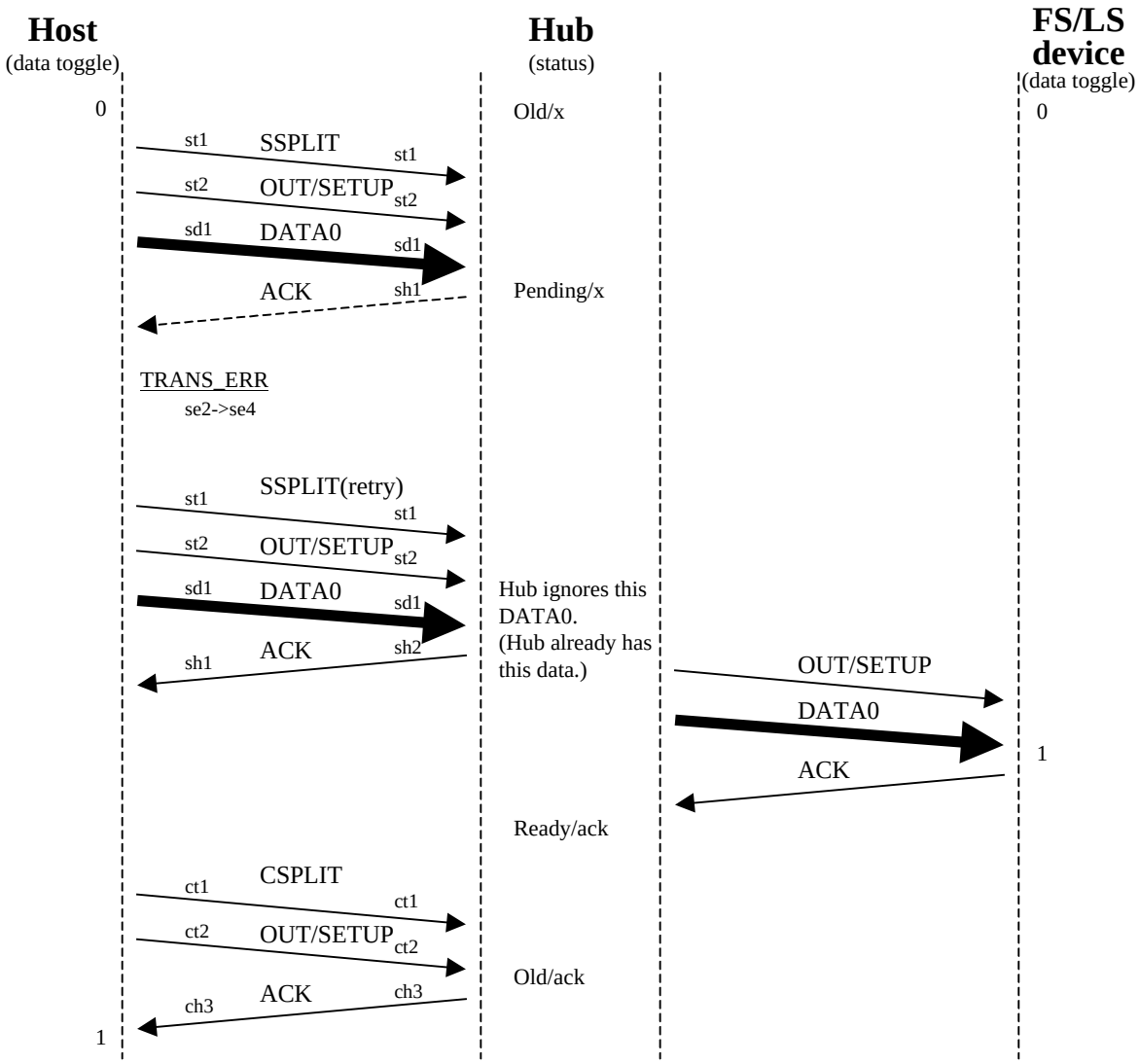


Figure A-4. Normal HS ACK(S) Smash(case 1)

Microchip Ex. 1018, p. 472
Microchip v. Aptiv
IPR2024-00494

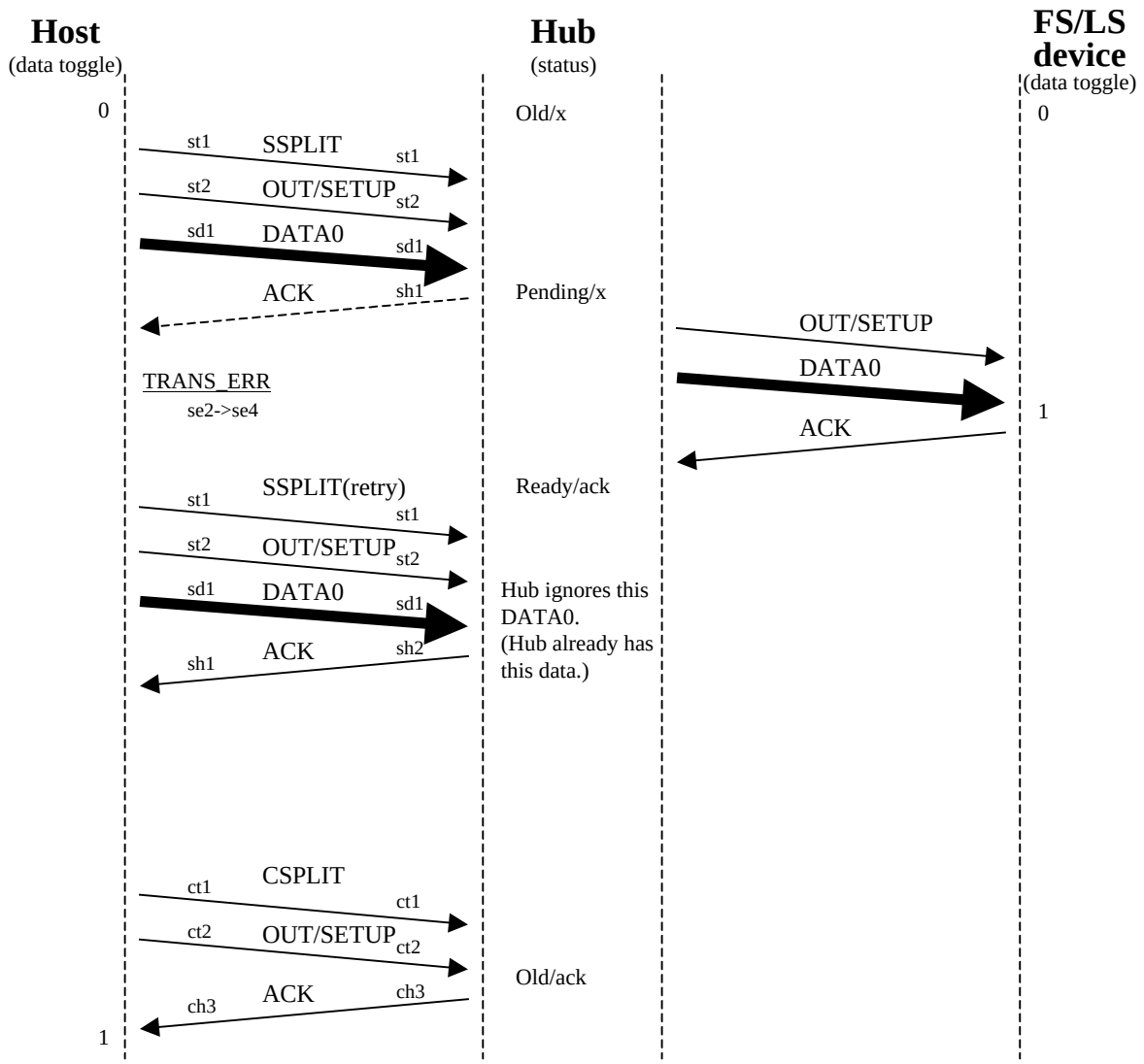


Figure A-5. Normal HS ACK(S) Smash(case 2)

Microchip Ex. 1018, p. 473
Microchip v. Aptiv
IPR2024-00494

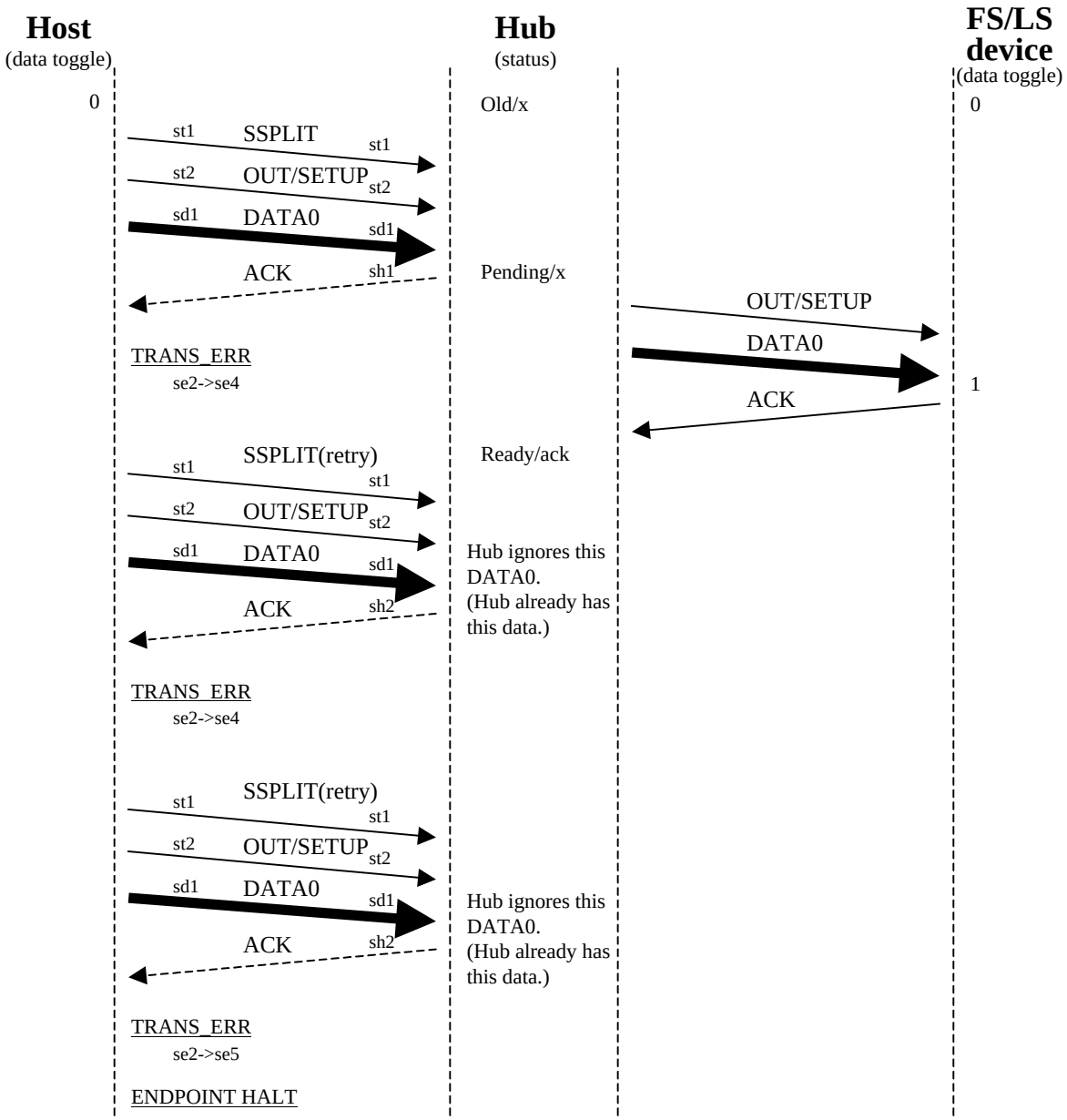


Figure A-6. Normal HS ACK(S) 3 Strikes Smash

Microchip Ex. 1018, p. 474
Microchip v. Aptiv
IPR2024-00494

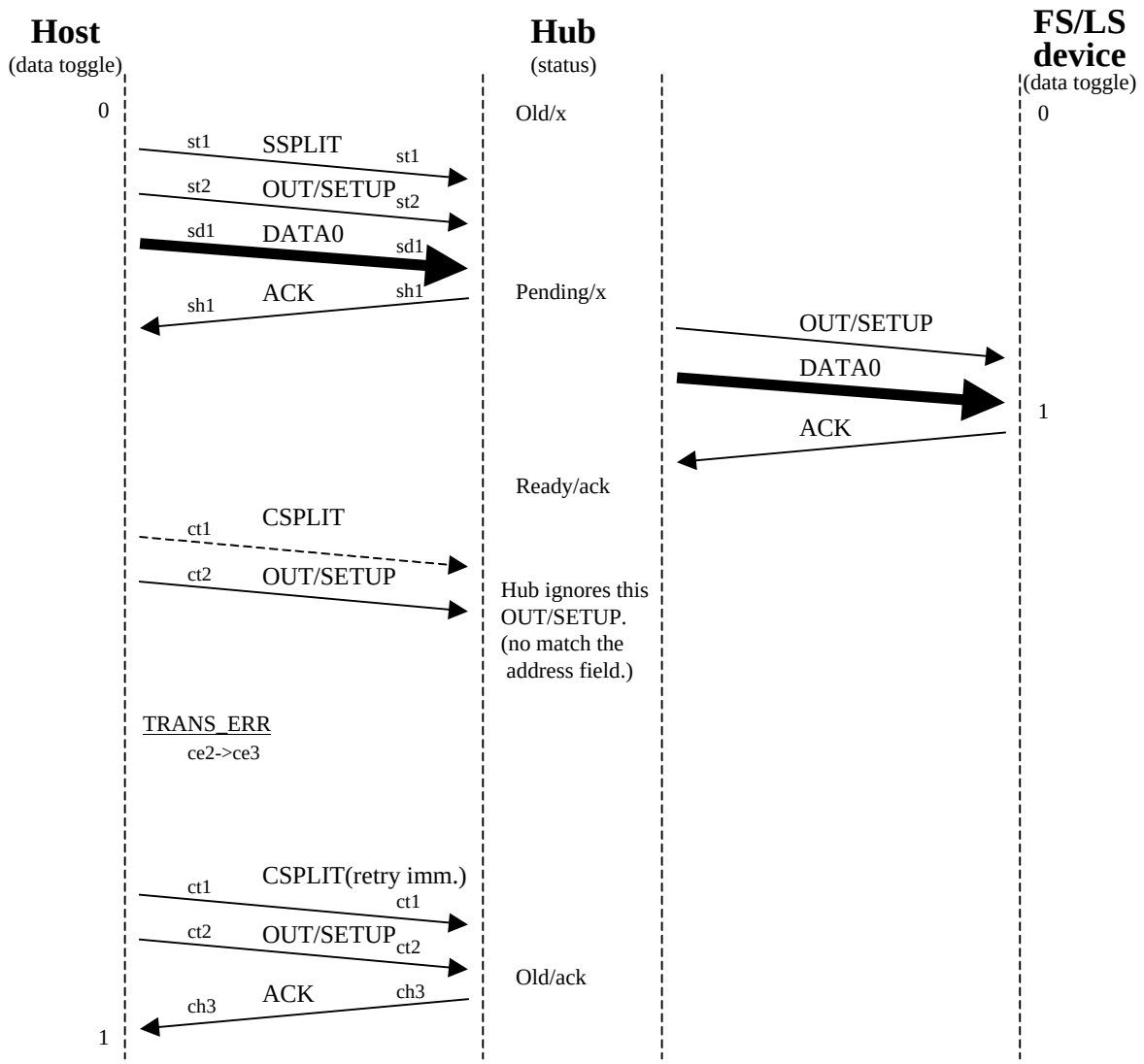


Figure A-7. Normal HS CSPLIT Smash

Microchip Ex. 1018, p. 475
Microchip v. Aptiv
IPR2024-00494

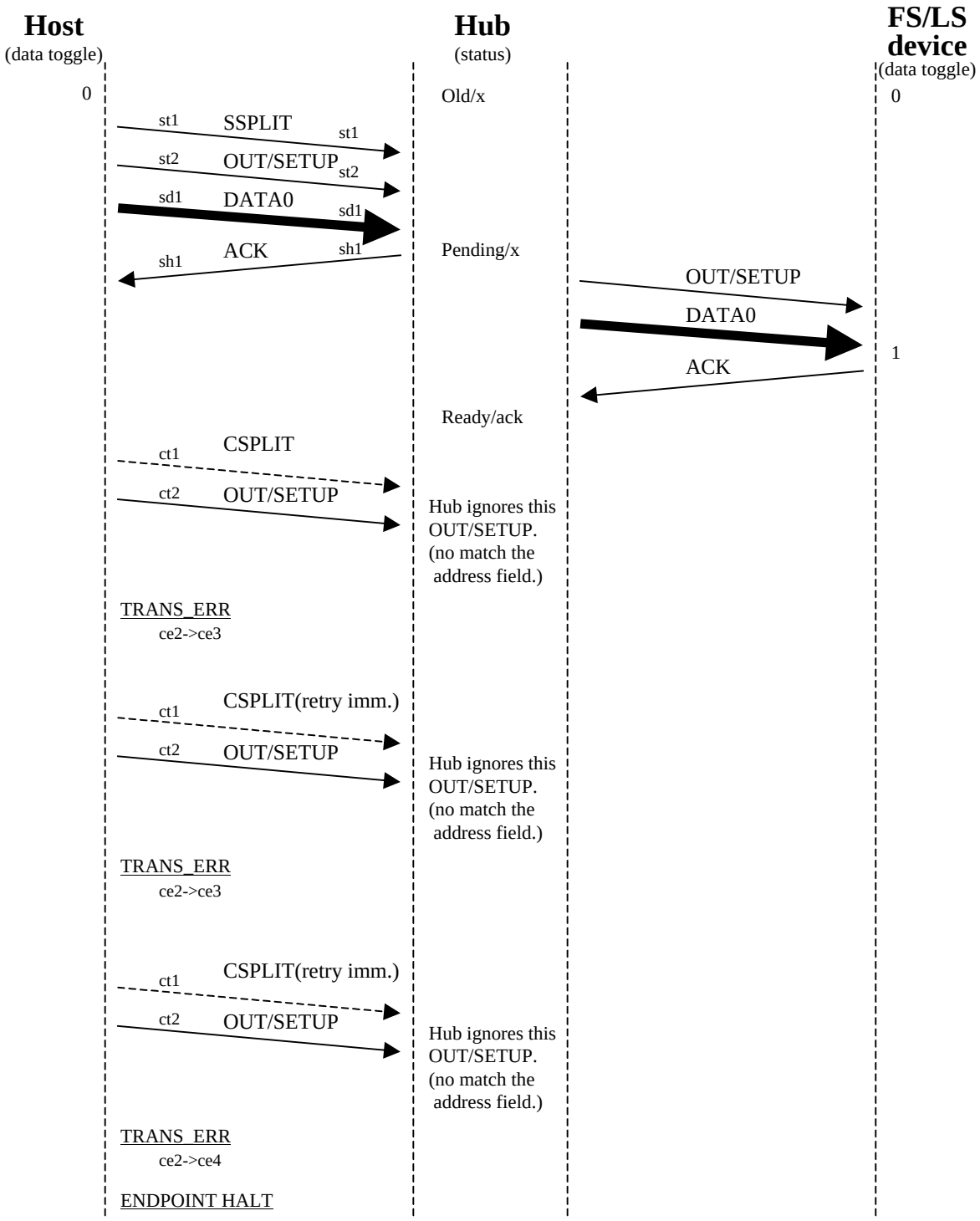


Figure A-8. Normal HS CSPLIT 3 Strikes Smash

Microchip Ex. 1018, p. 476
Microchip v. Aptiv
IPR2024-00494

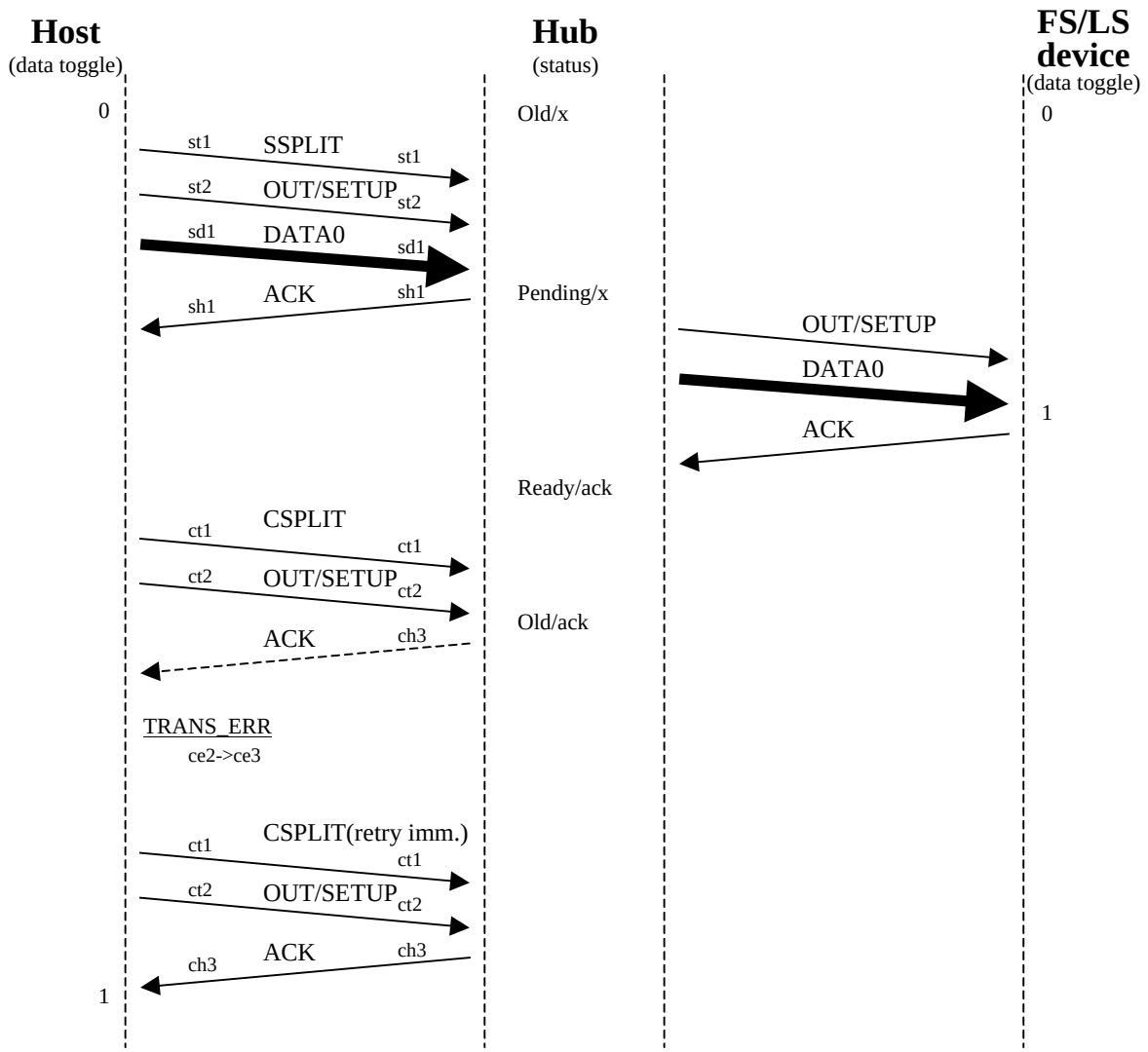


Figure A-9. Normal HS ACK(C) Smash

Microchip Ex. 1018, p. 477
Microchip v. Aptiv
IPR2024-00494

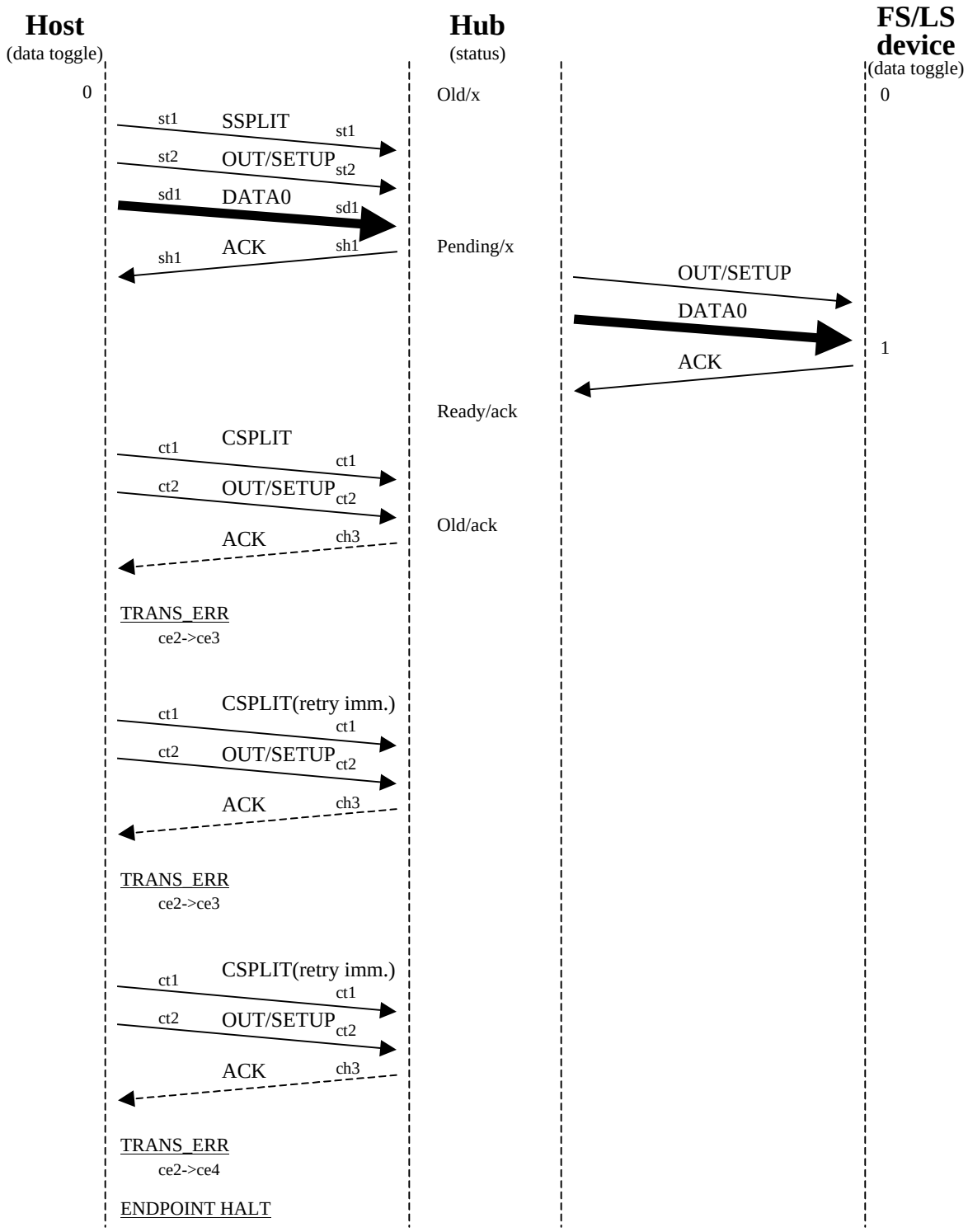


Figure A-10. Normal S ACK(C) 3 Strikes Smash

Microchip Ex. 1018, p. 478
Microchip v. Aptiv
IPR2024-00494

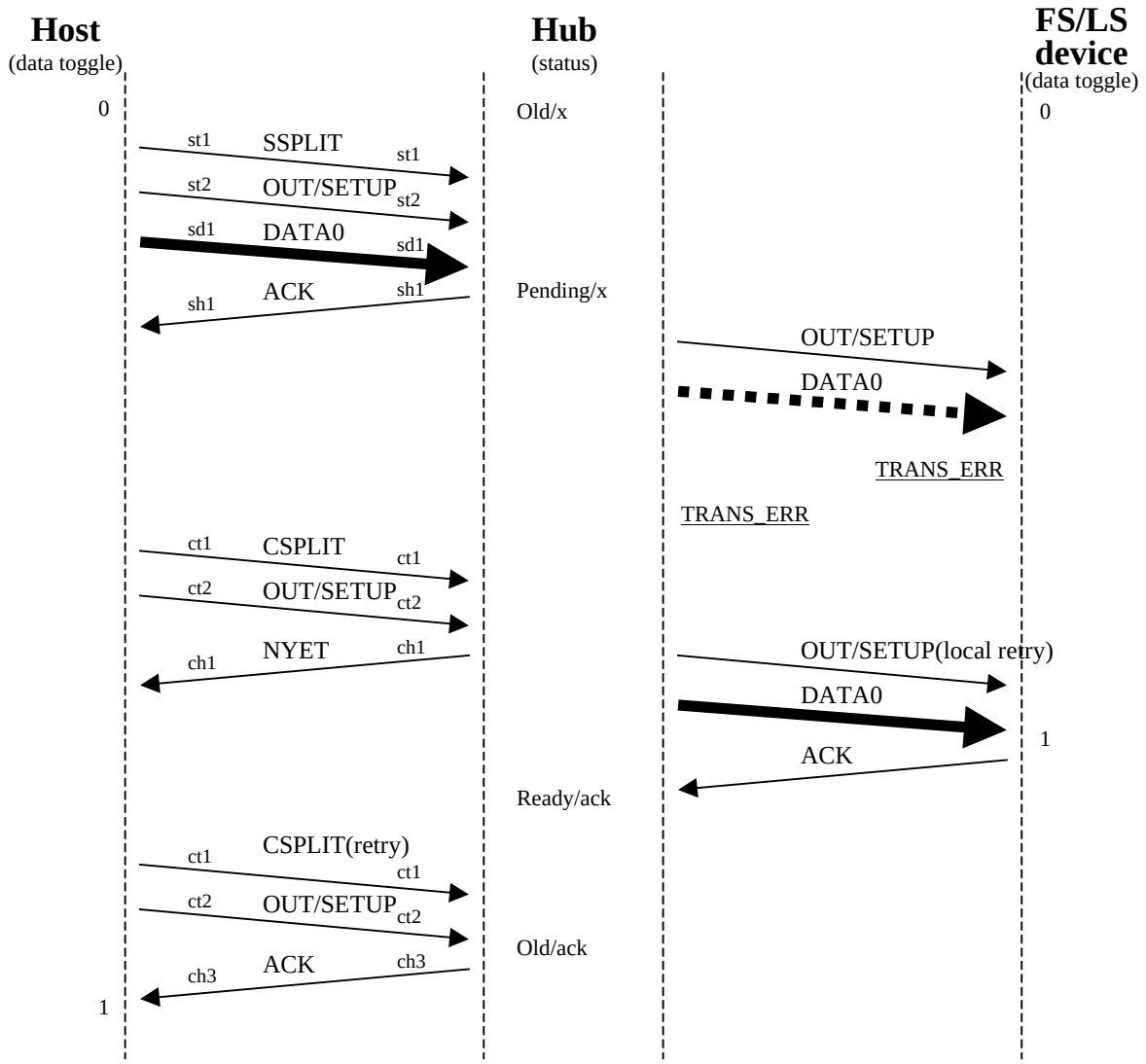


Figure A-11. Normal FS/LS DATA0/1 Smash

Microchip Ex. 1018, p. 479
Microchip v. Aptiv
IPR2024-00494

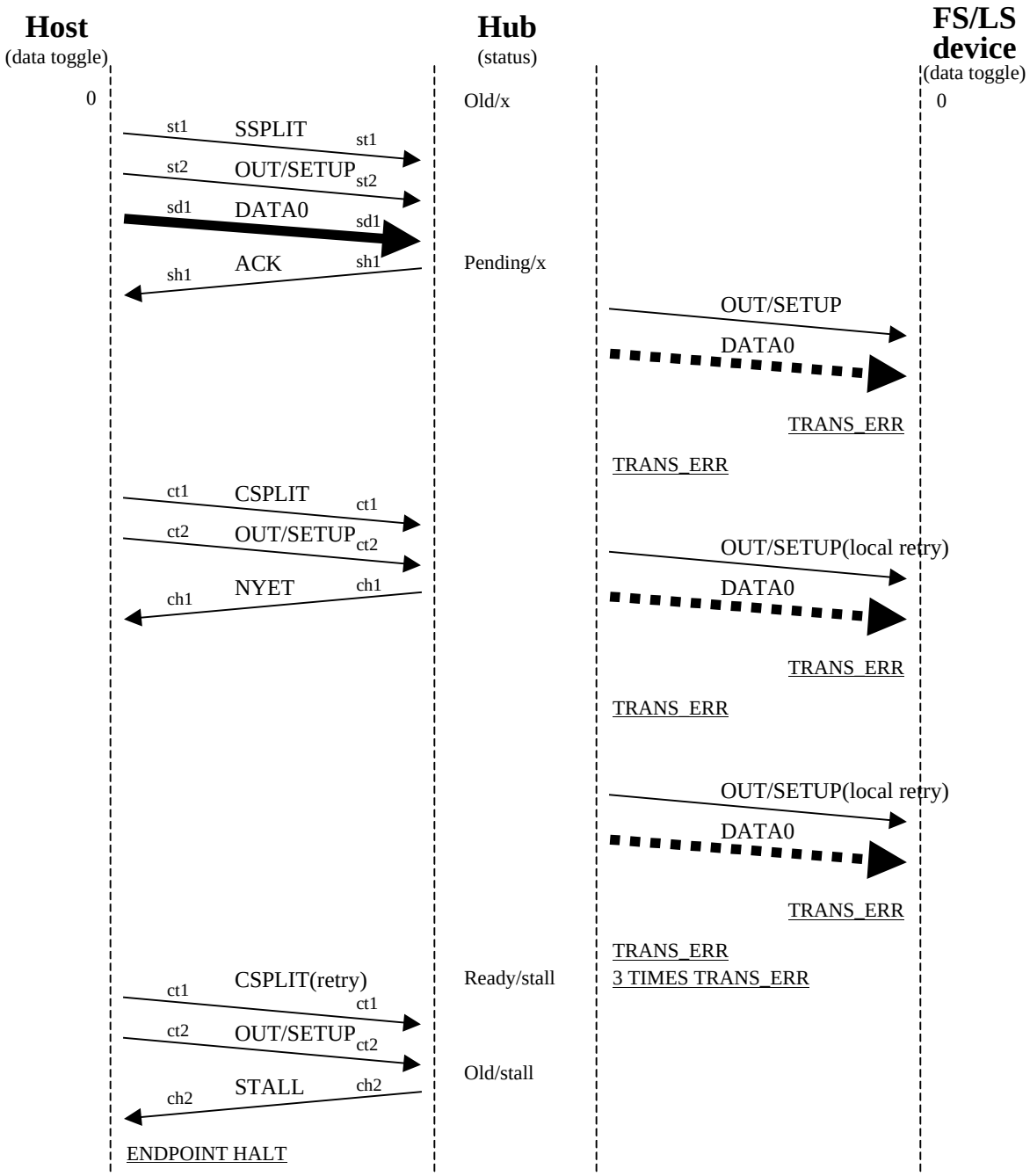


Figure A-12. Normal FS/LS DATA0/1 3 Strikes Smash

Microchip Ex. 1018, p. 480
Microchip v. Aptiv
IPR2024-00494

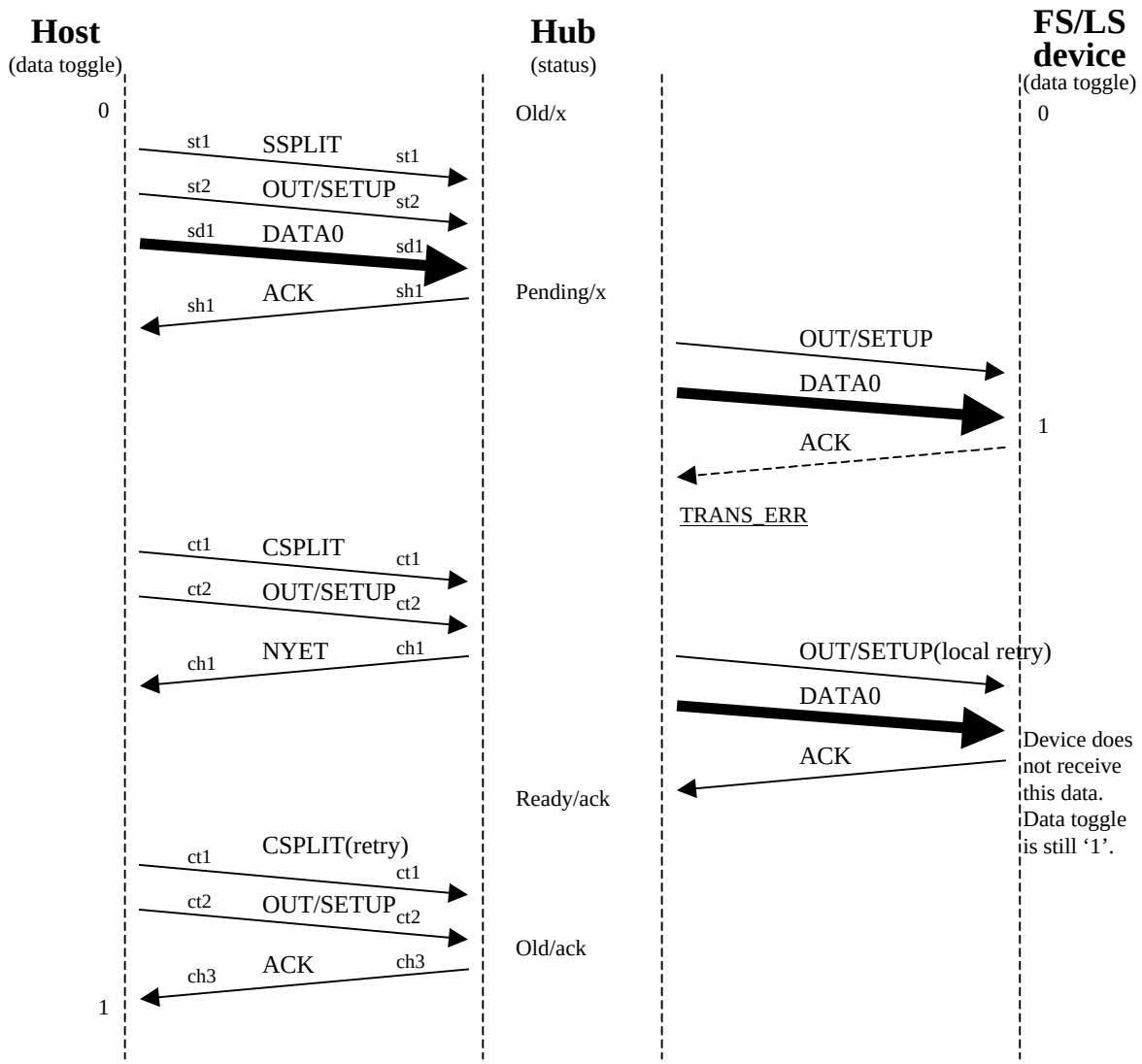


Figure A-13. Normal FS/LS ACK Smash

Microchip Ex. 1018, p. 481
Microchip v. Aptiv
IPR2024-00494

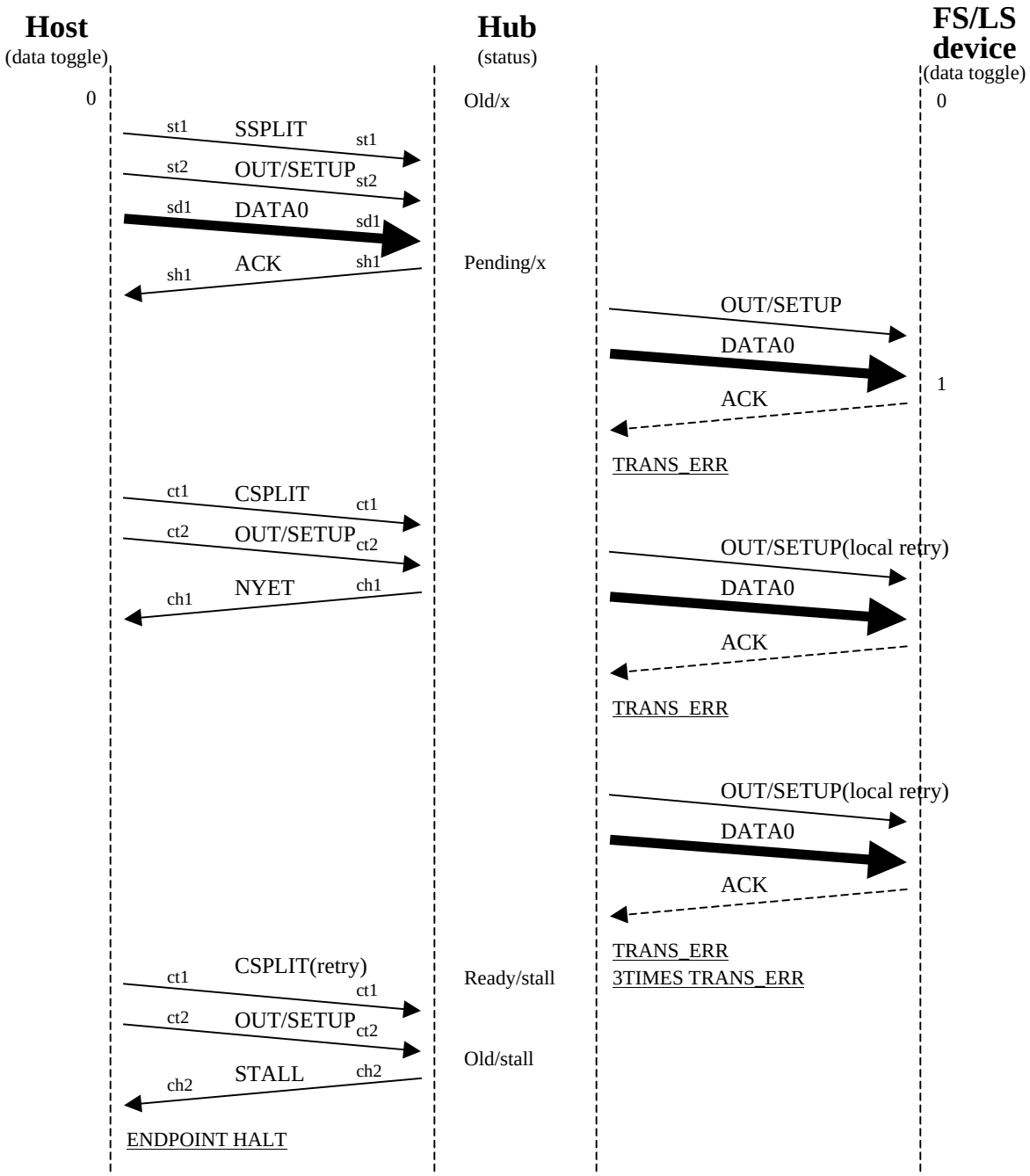


Figure A-14. Normal FS/LS ACK 3 Strikes Smash

Microchip Ex. 1018, p. 482
Microchip v. Aptiv
IPR2024-00494

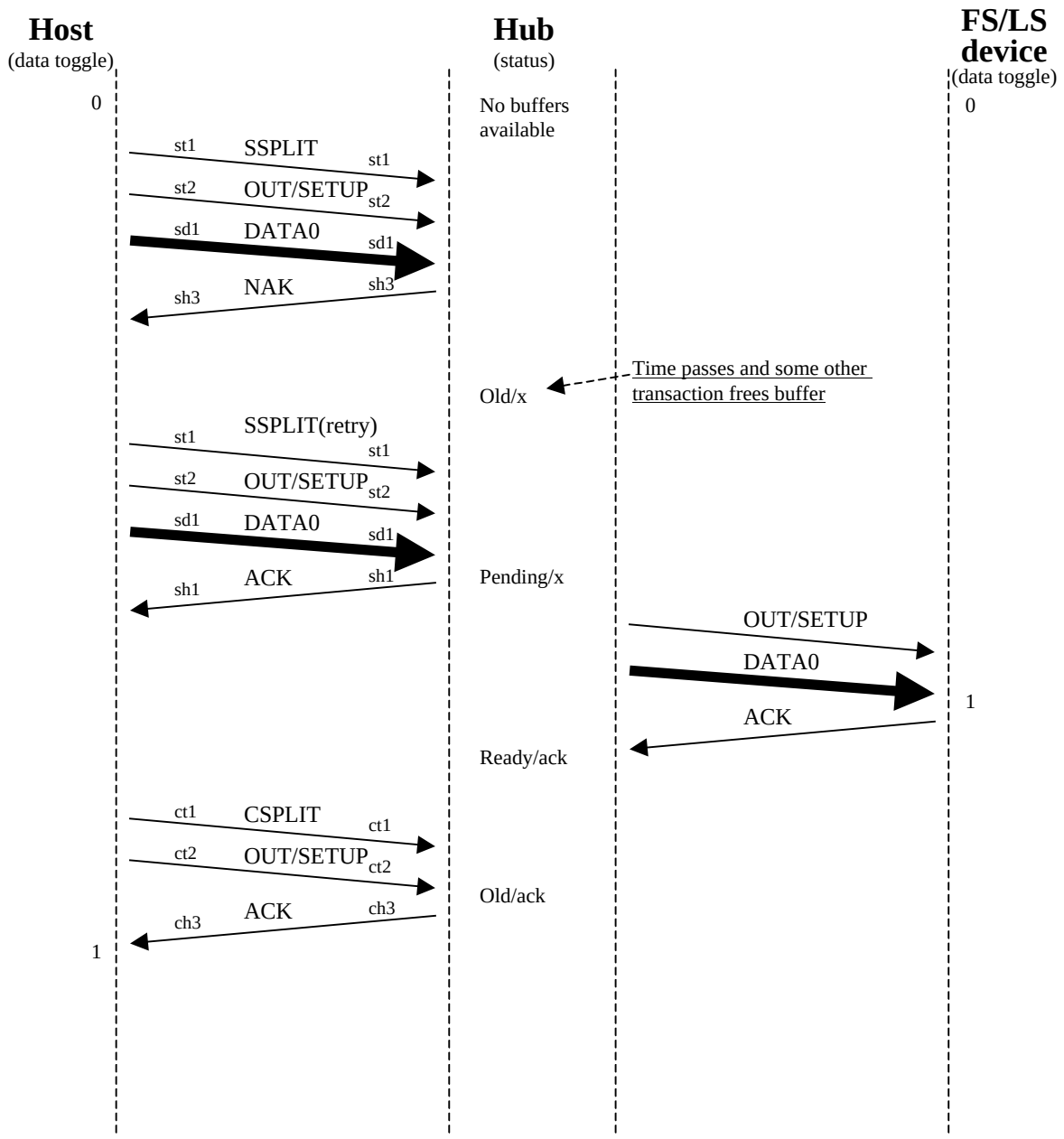


Figure A-15. No buffer Available No Smash (HS NAK(S))

Microchip Ex. 1018, p. 483
Microchip v. Aptiv
IPR2024-00494

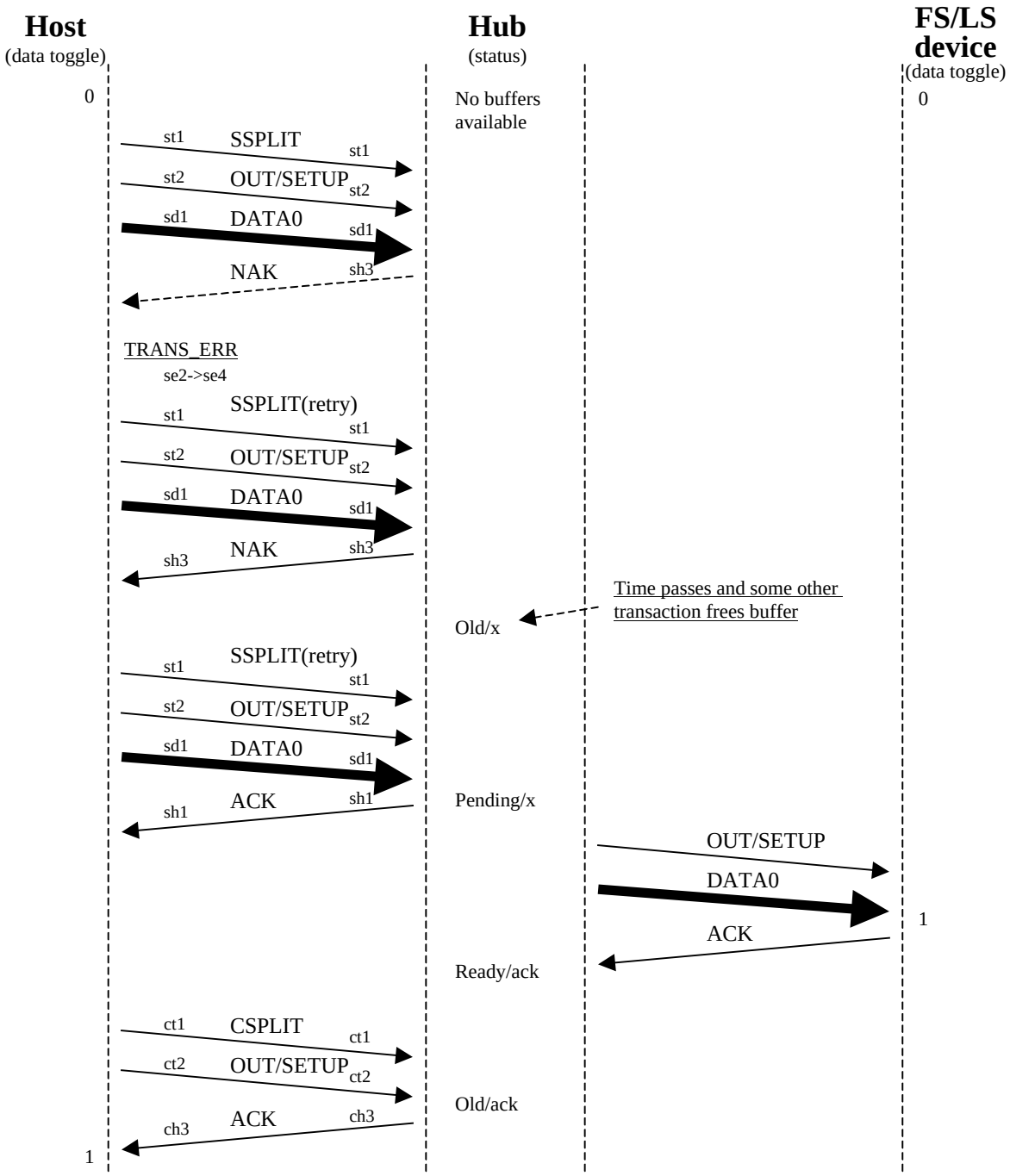


Figure A-16. No Buffer Available HS NAK(S) Smash

Microchip Ex. 1018, p. 484
Microchip v. Aptiv
IPR2024-00494

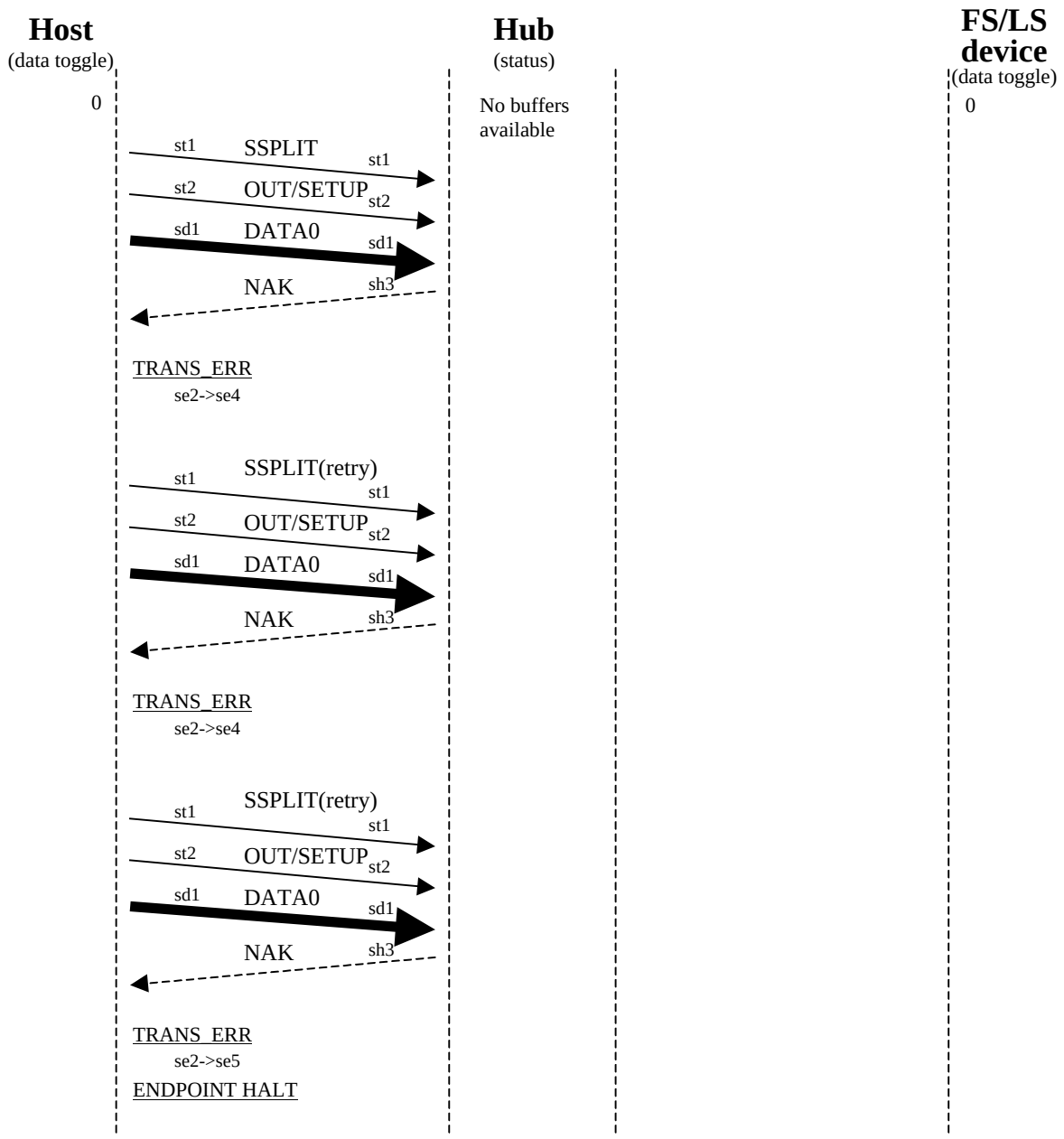


Figure A-17. No Buffer Available HS NAK(S) 3 Strikes Smash

Microchip Ex. 1018, p. 485
Microchip v. Aptiv
IPR2024-00494

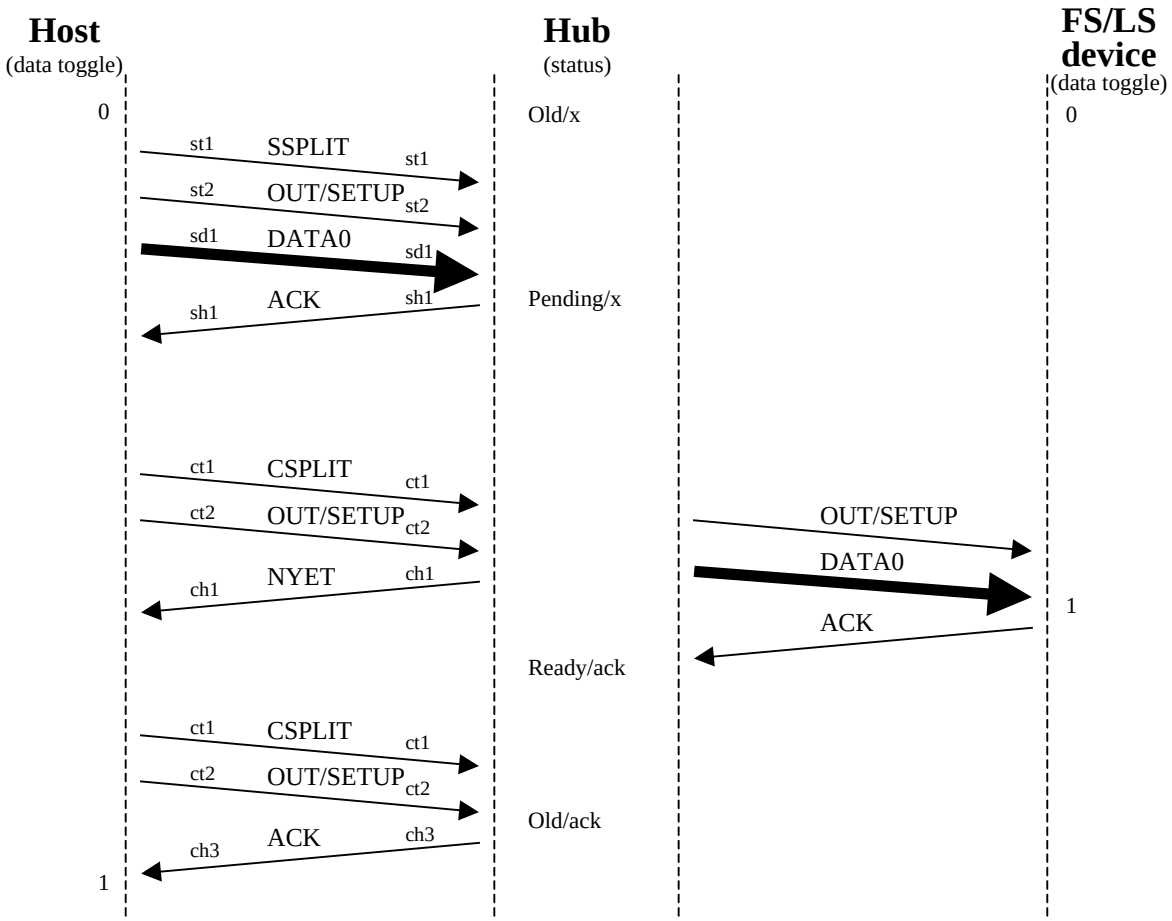


Figure A-18. CS Earlier No Smash (HS NYET)

Microchip Ex. 1018, p. 486
Microchip v. Aptiv
IPR2024-00494

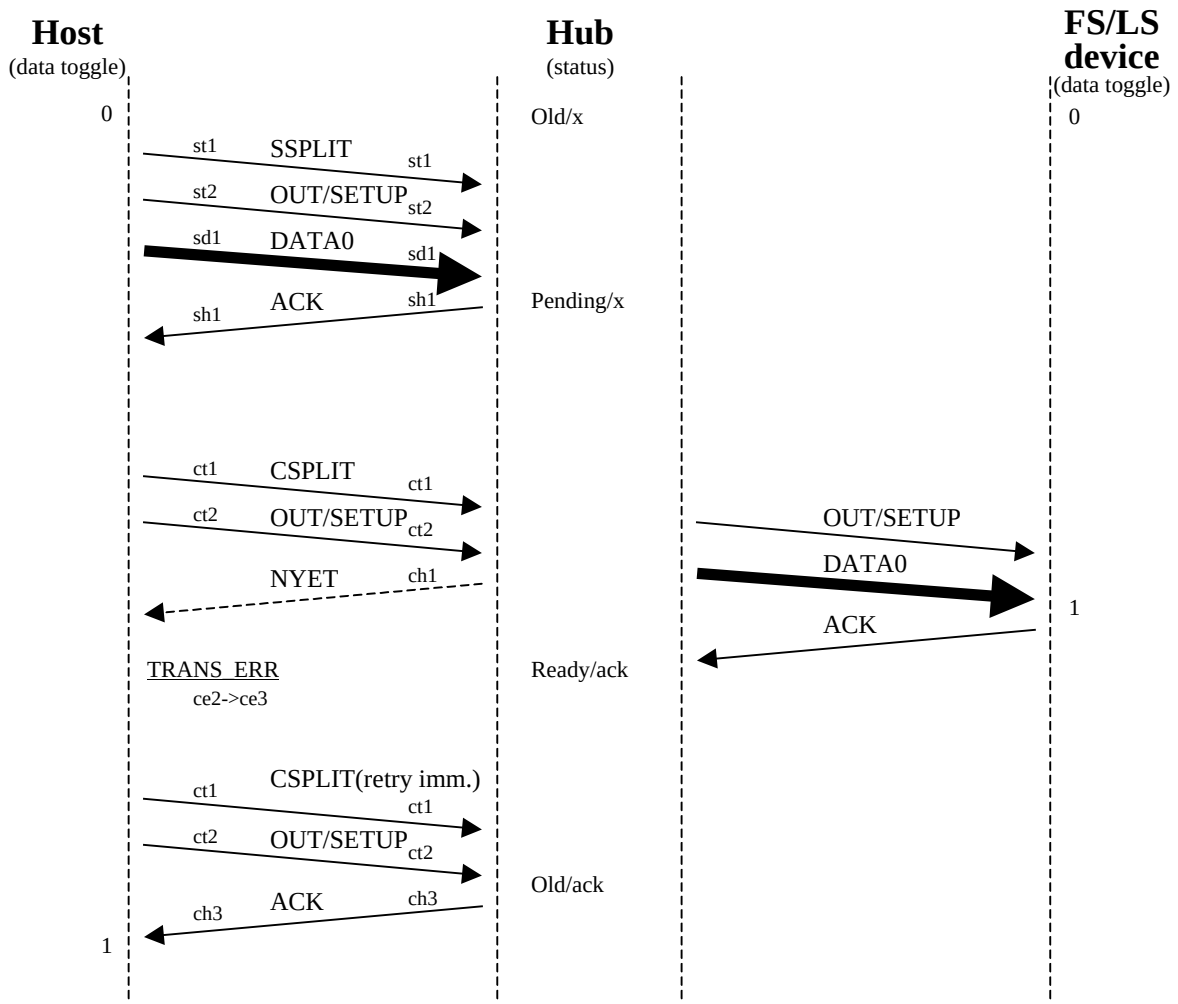


Figure A-19. CS Earlier HS NYET Smash(case 1)

Microchip Ex. 1018, p. 487
Microchip v. Aptiv
IPR2024-00494

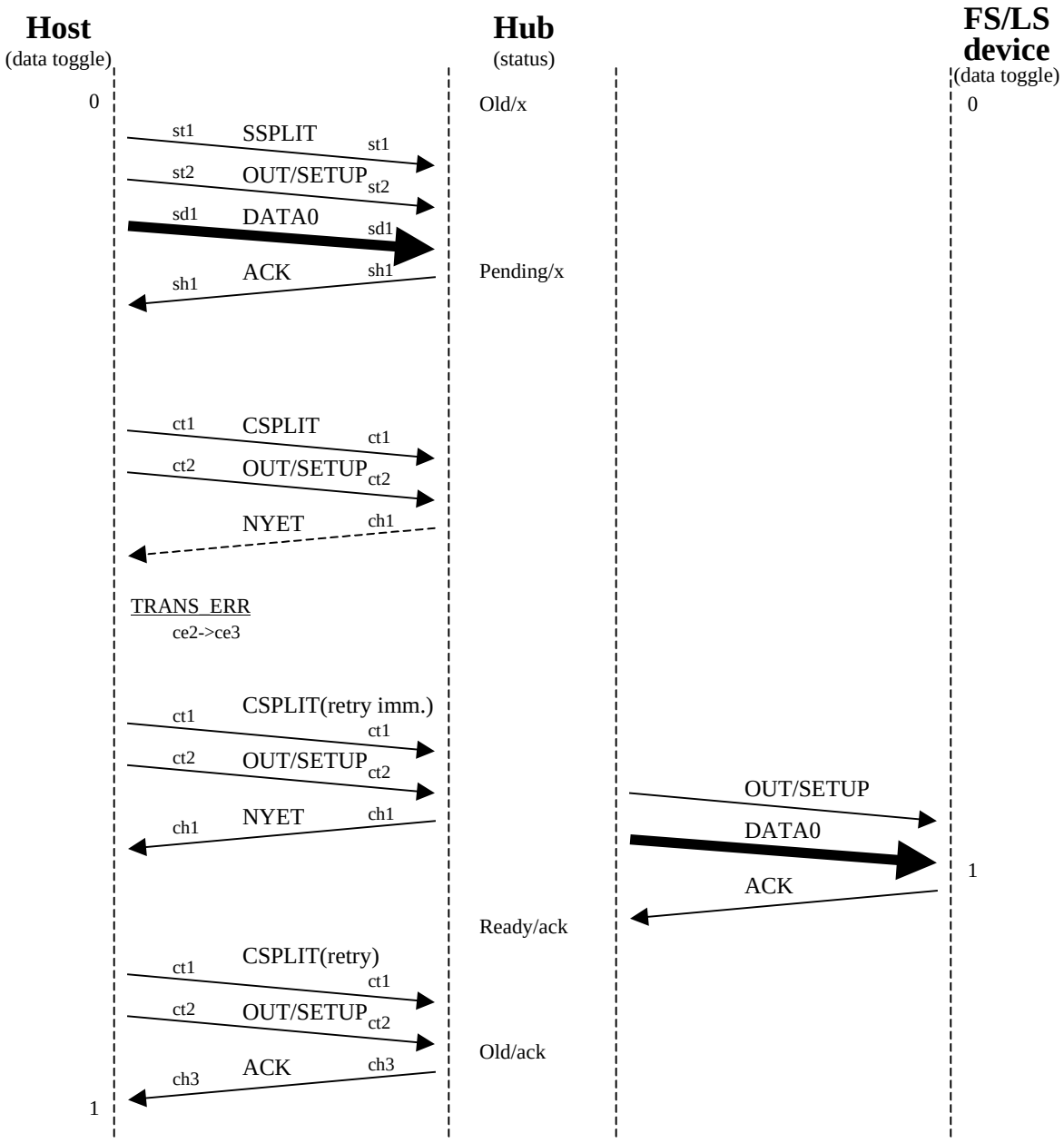


Figure A-20. CS Earlier HS NYET Smash(case 2)

Microchip Ex. 1018, p. 488
Microchip v. Aptiv
IPR2024-00494

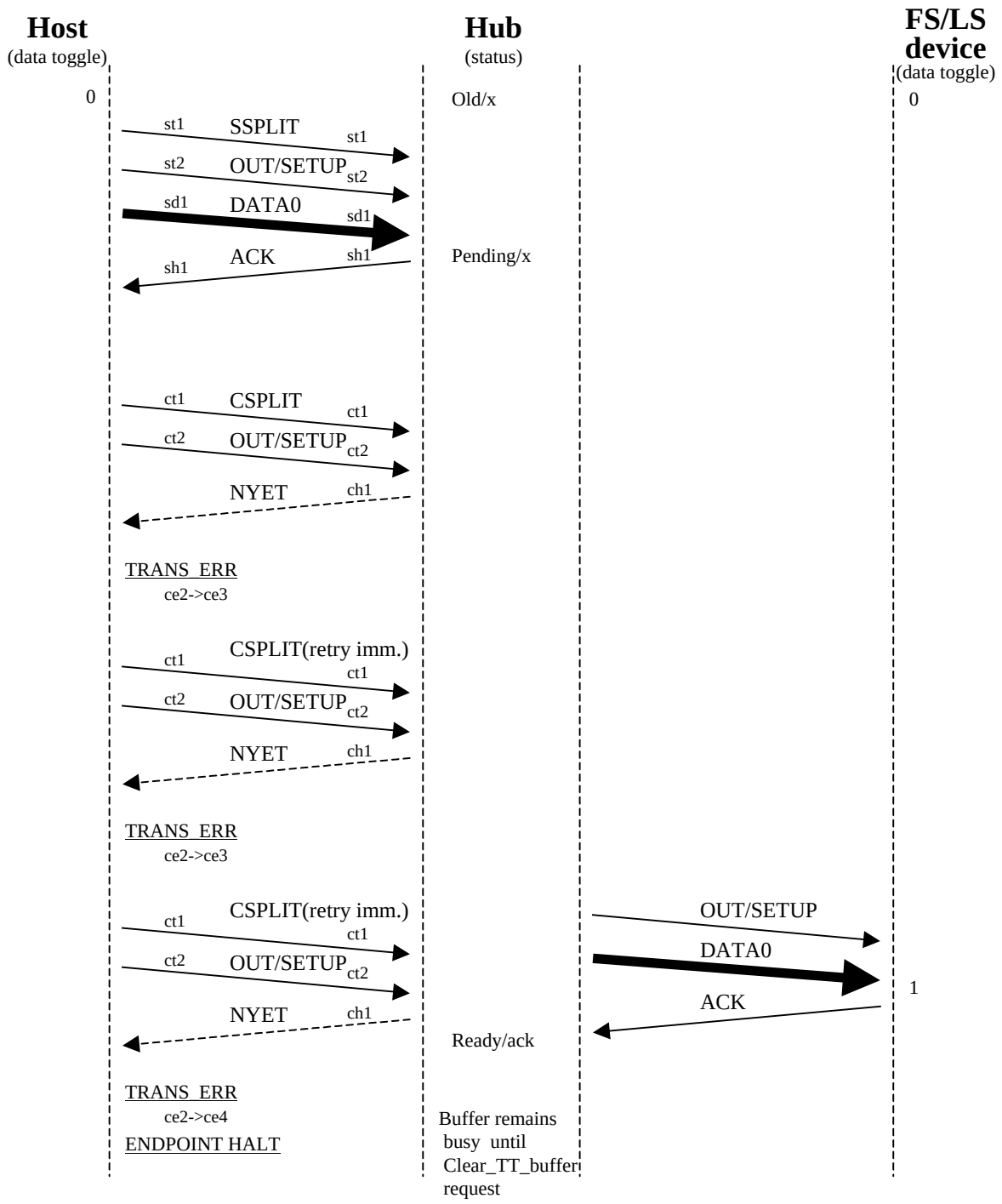


Figure A-21. CS Earlier HS NYET 3 Strikes Smash

Microchip Ex. 1018, p. 489
Microchip v. Aptiv
IPR2024-00494

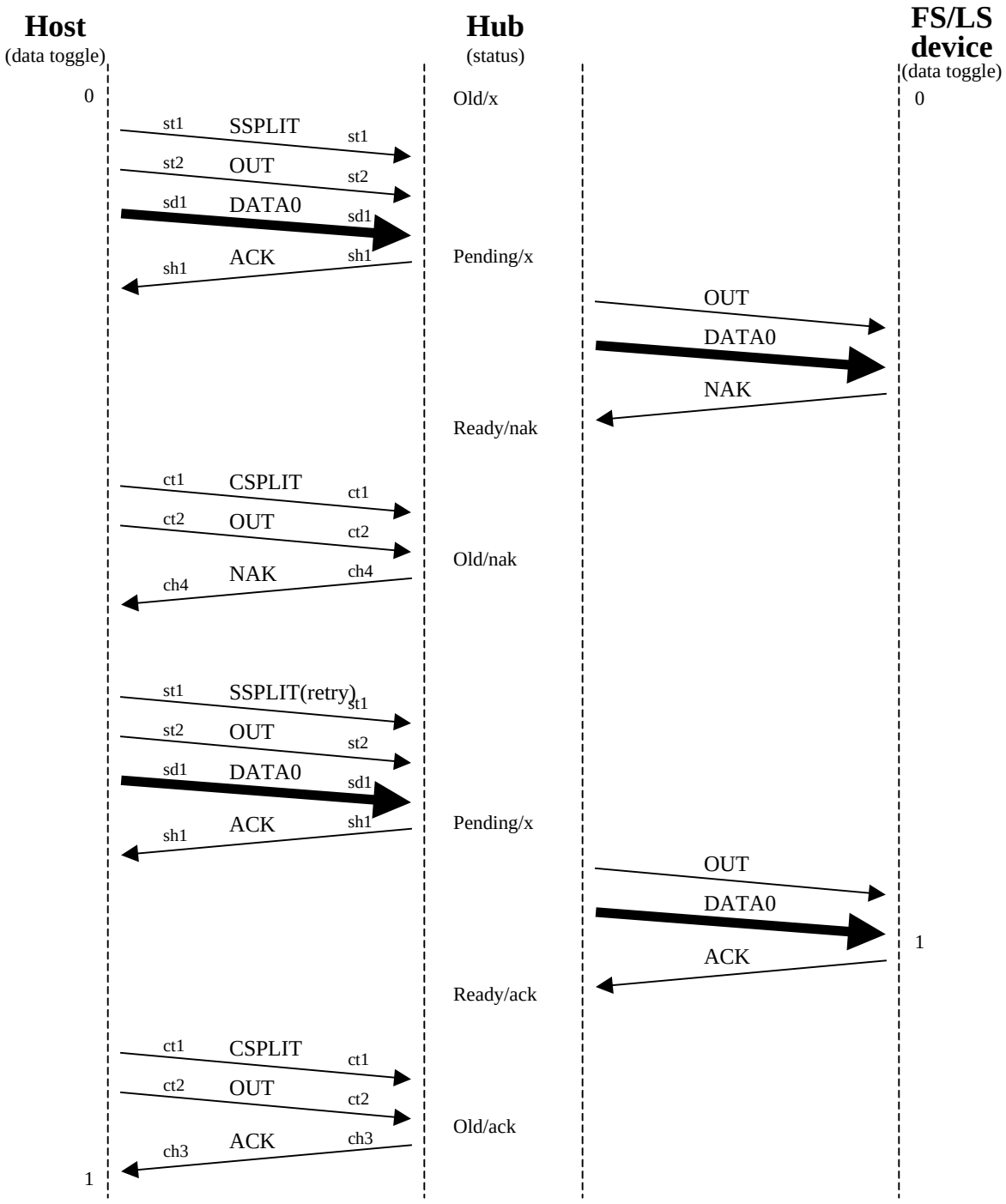


Figure A-22. Device Busy No Smash(FS/LS NAK)

Microchip Ex. 1018, p. 490
Microchip v. Aptiv
IPR2024-00494

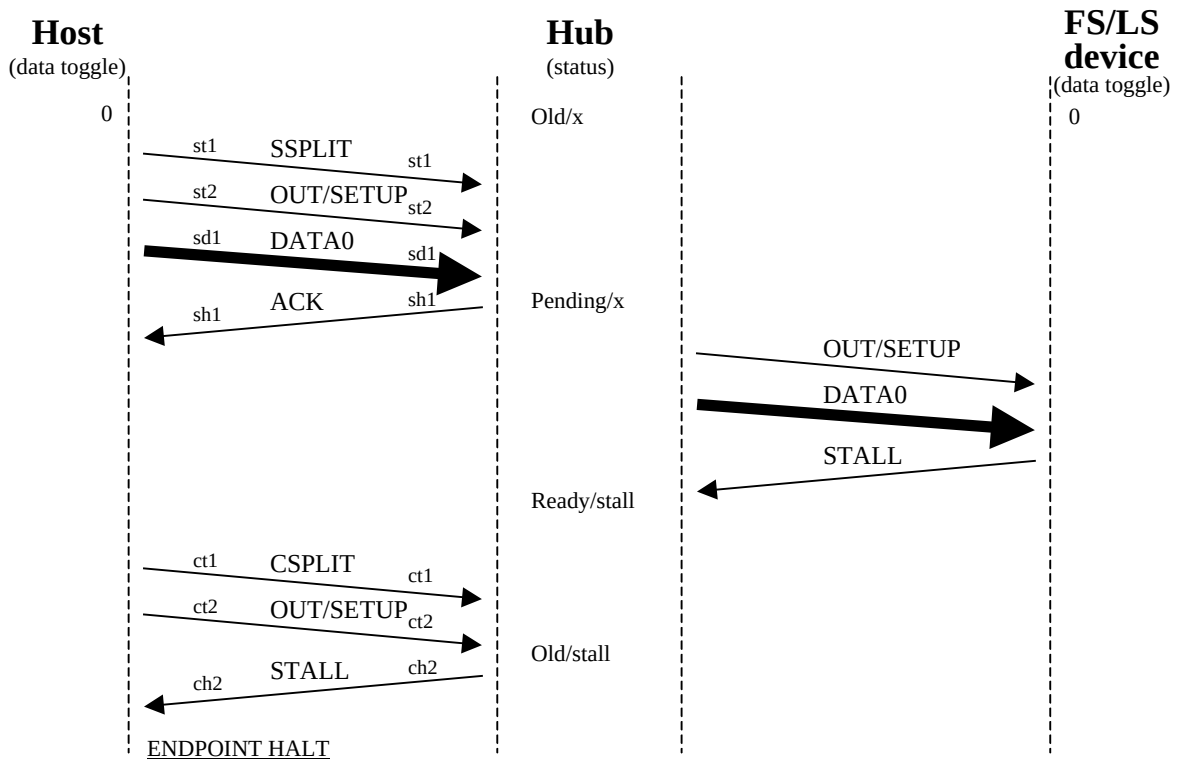


Figure A-23. Device Stall No Smash(FS/LS STALL)

Microchip Ex. 1018, p. 491
Microchip v. Aptiv
IPR2024-00494

A.2 Bulk/Control IN Transaction Examples

Legend:

(S): Start Split

(C): Complete Split

Summary of cases for bulk/control IN transaction

- Normal cases

Case	Reference figure	Similar figure
No smash	Figure A-24	
HS SSPLIT smash	Figure A-25	
HS SSPLIT 3 strikes smash	Figure A-26	
HS IN(S) smash		Figure A-25
HS IN(S) 3 strikes smash		Figure A-26
HS ACK(S) smash	Figure A-27 Figure A-28	
HS ACK(S) 3 strikes smash	Figure A-29	
HS CSPLIT smash	Figure A-30	
HS CSPLIT 3 strikes smash	Figure A-31	
HS IN(C) smash		Figure A-30
HS IN(C) 3 strikes smash		Figure A-31
HS DATA0/1 smash	Figure A-32	
HS DATA0/1 3 strikes smash	Figure A-33	
FS/LS IN smash	Figure A-34	
FS/LS IN 3 strikes smash	Figure A-35	
FS/LS DATA0/1 smash	Figure A-36	
FS/LS DATA0/1 3 strikes smash	Figure A-37	

Microchip Ex. 1018, p. 492
Microchip v. Aptiv
IPR2024-00494

FS/LS ACK smash	Figure A-38	
FS/LS ACK 3 strikes smash	No figure	

- No buffer(on hub) avallable cases

Case	Reference figure	Similar figure
No smash(HS NAK(S))	Figure A-39	
HS NAK(S) smash	Figure A-40	
HS NAK(S) 3 strikes smash	Figure A-41	

- CS(Complete-split transaction) earlier cases

Case	Reference figure	Similar figure
No smash(HS NYET)	Figure A-42	
HS NYET smash	Figure A-43 Figure A-44	
HS NYET 3 strikes smash	No figure	

- Device busy cases

Case	Reference figure	Similar figure
No smash(HS NAK(C))	Figure A-45	
HS NAK(C) smash		Figure A-32
HS NAK(C) 3 strikes smash		Figure A-33
FS/LS NAK smash		Figure A-36
FS/LS NAK 3 strikes smash		Figure A-37

Microchip Ex. 1018, p. 493
Microchip v. Aptiv
IPR2024-00494

- Device stall cases

Case	Reference figure	Similar figure
No smash	Figure A-46	
HS STALL(C) smash		Figure A-32
HS STALL(C) 3 strikes smash		Figure A-33
FS/LS STALL smash		Figure A-36
FS/LS STALL 3 strikes smash		Figure A-37

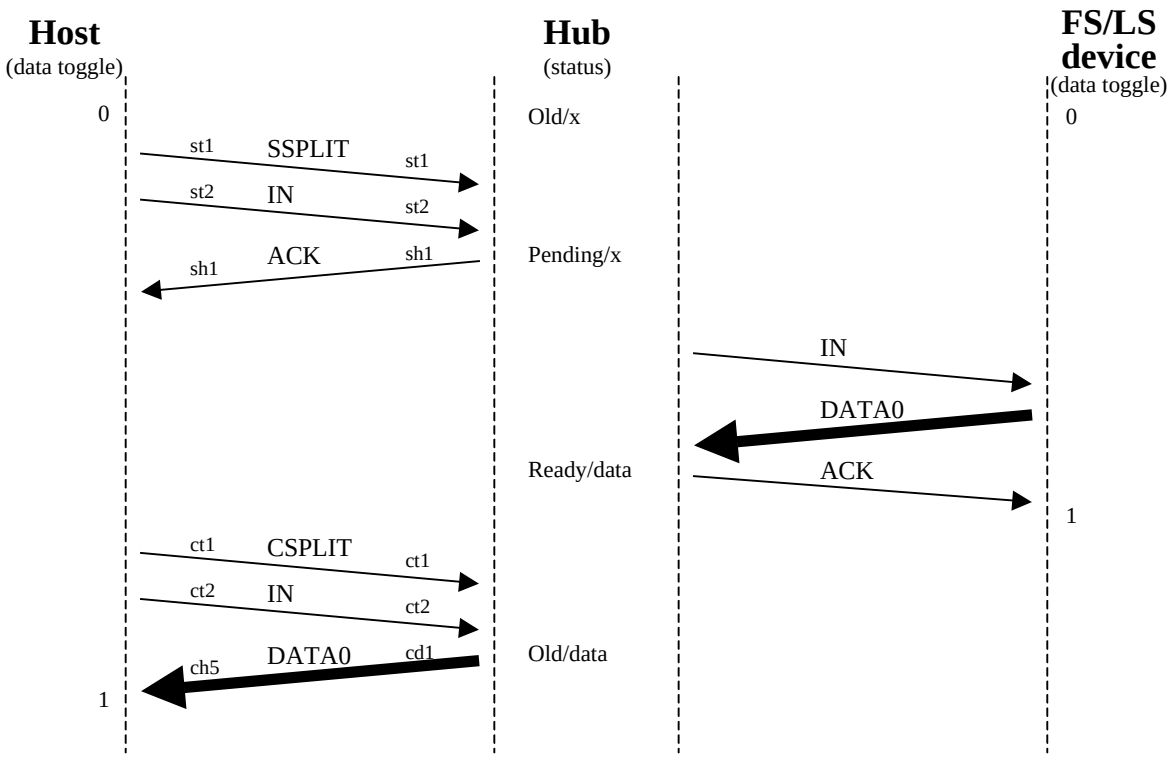


Figure A-24. Normal No Smash

Microchip Ex. 1018, p. 494
Microchip v. Aptiv
IPR2024-00494

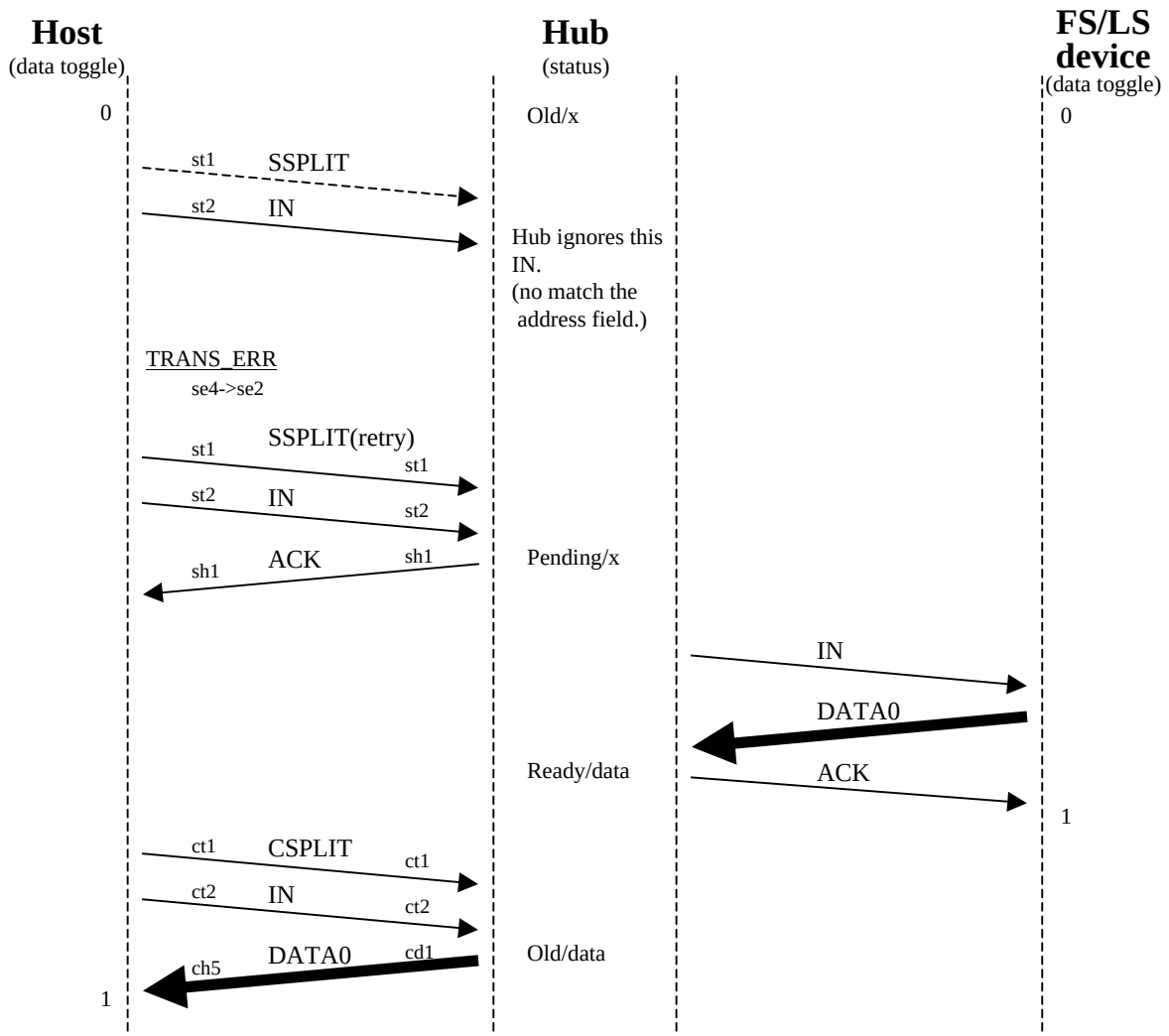


Figure A-25. Normal HS SSPLIT Smash

Microchip Ex. 1018, p. 495
Microchip v. Aptiv
IPR2024-00494

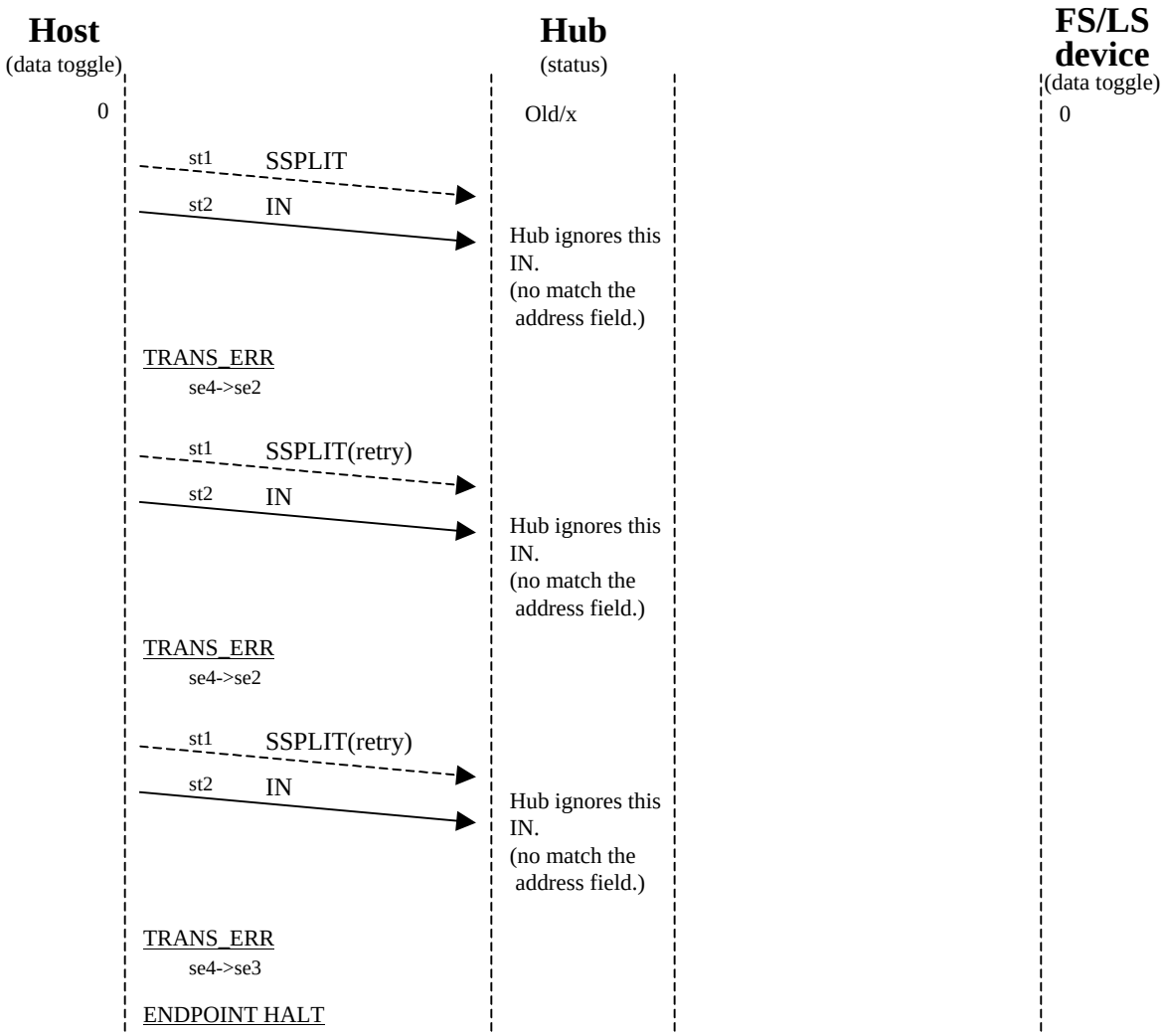


Figure A-26. Normal SSPLIT 3 Strikes Smash

Microchip Ex. 1018, p. 496
Microchip v. Aptiv
IPR2024-00494

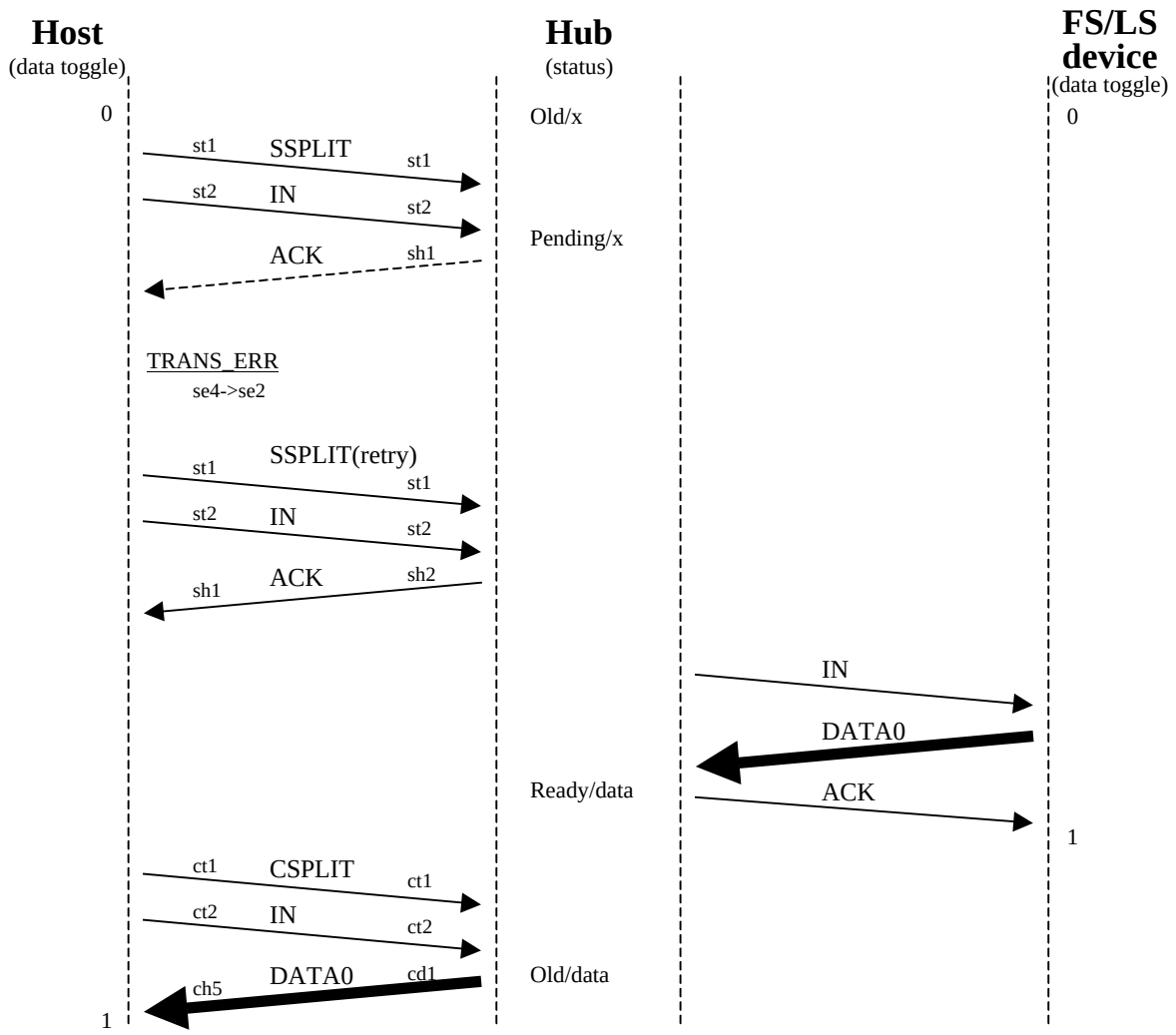


Figure A-27. Normal HS ACK(S) Smash(case 1)

Microchip Ex. 1018, p. 497
Microchip v. Aptiv
IPR2024-00494

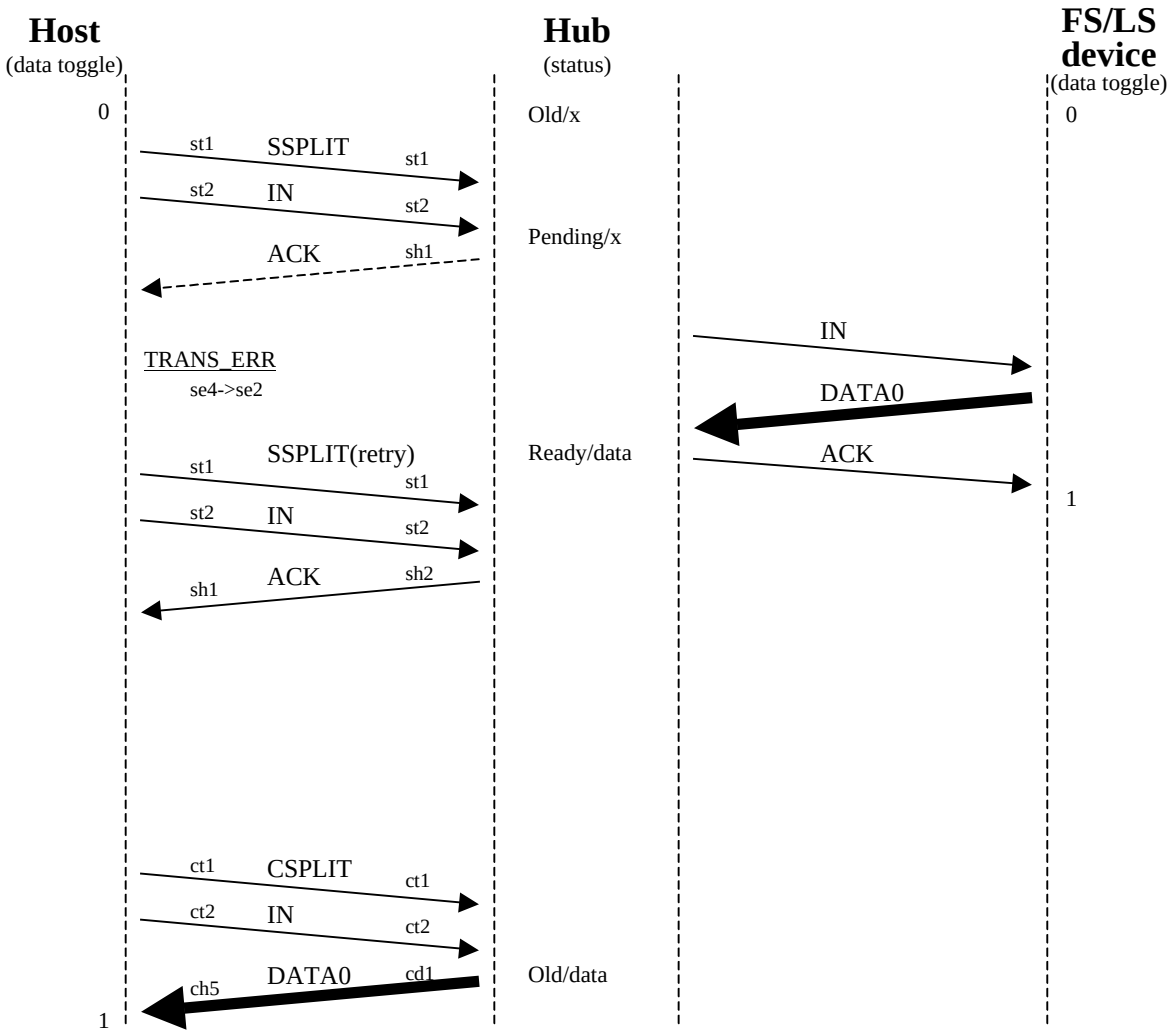


Figure A-28. Normal HS ACK(S) Smash(case 2)

Microchip Ex. 1018, p. 498
Microchip v. Aptiv
IPR2024-00494

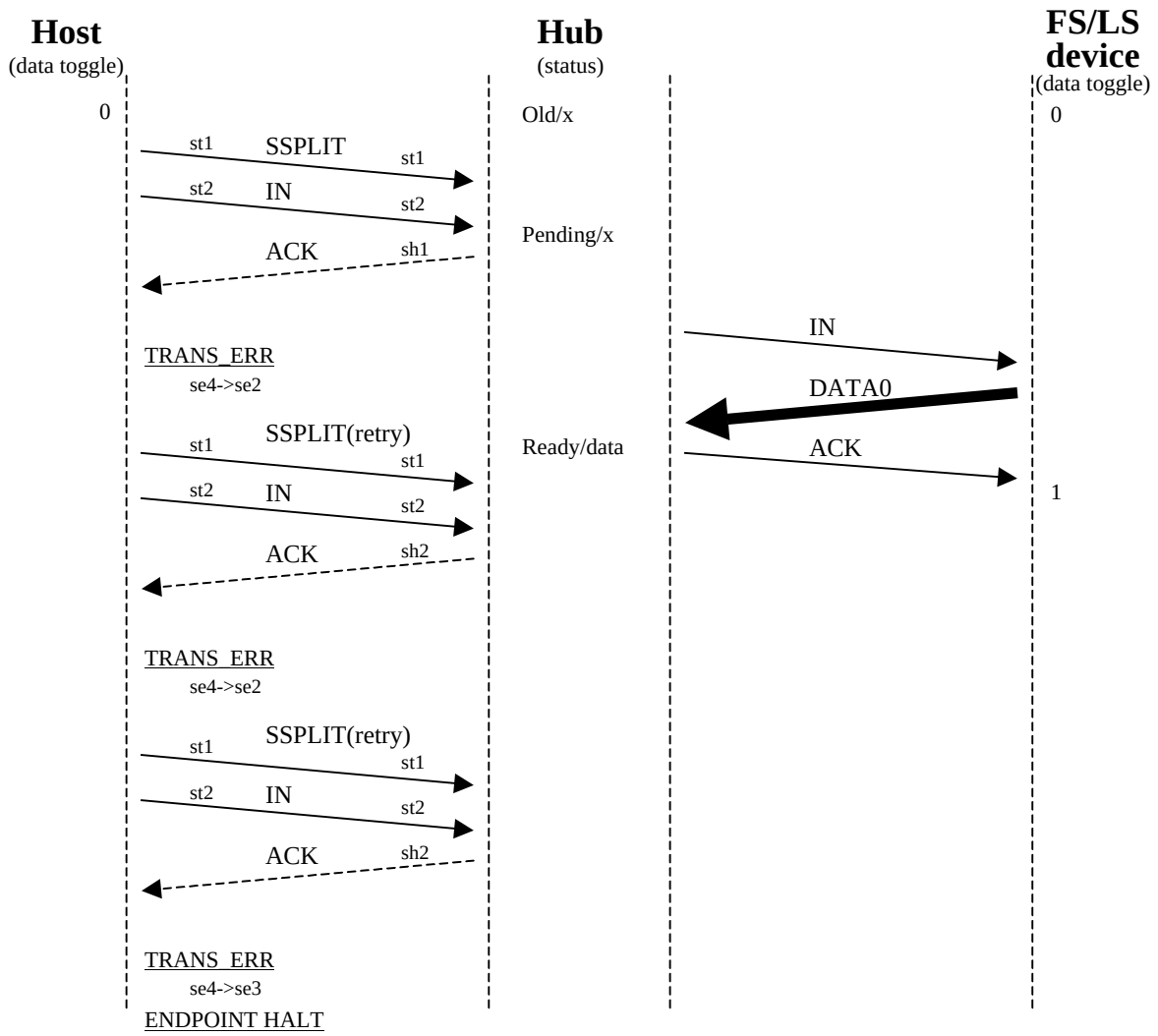


Figure A-29. Normal HS ACK(S) 3 Strikes Smash

Microchip Ex. 1018, p. 499
Microchip v. Aptiv
IPR2024-00494

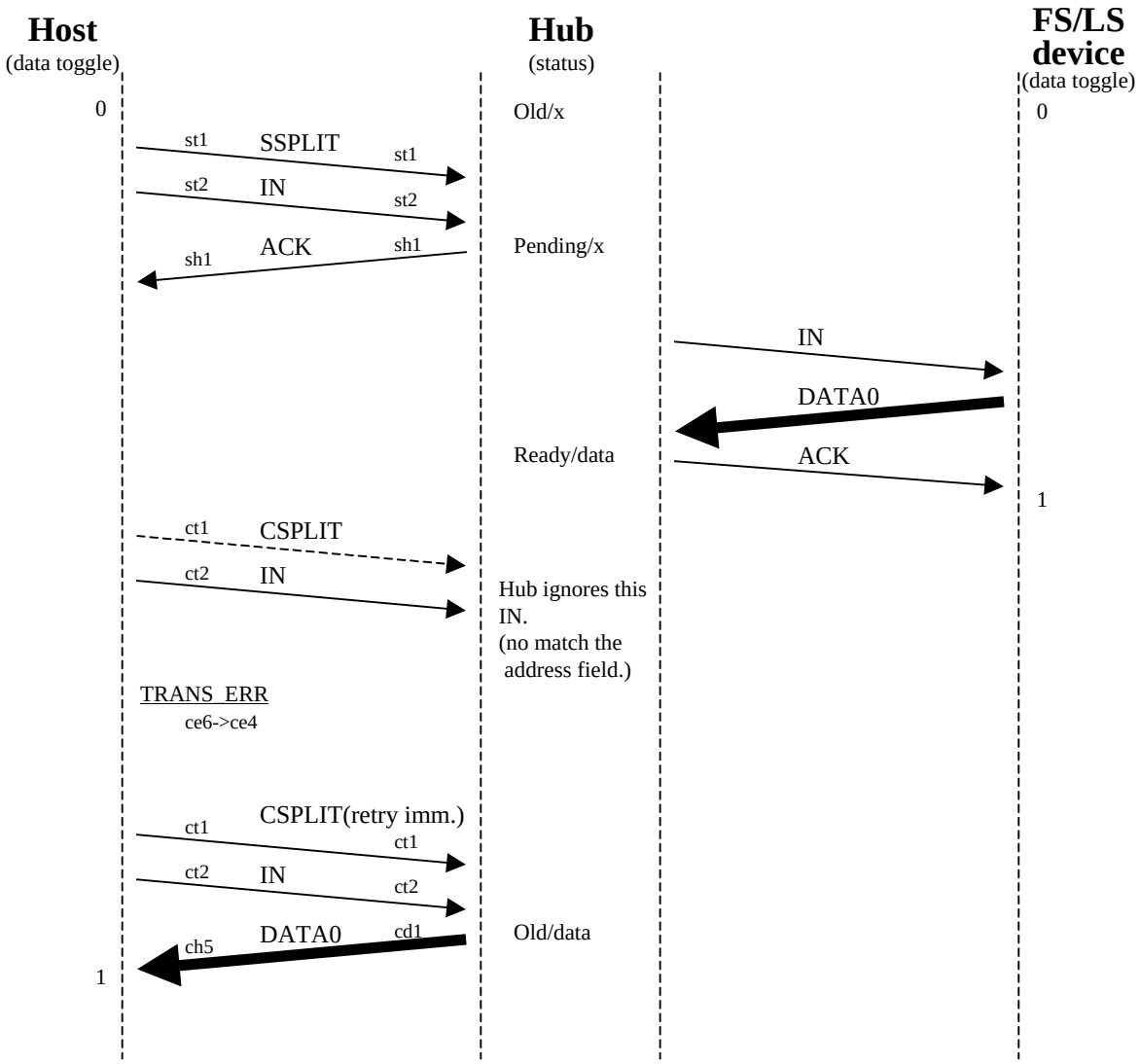


Figure A-30. Normal HS CSPLIT Smash

Microchip Ex. 1018, p. 500
Microchip v. Aptiv
IPR2024-00494

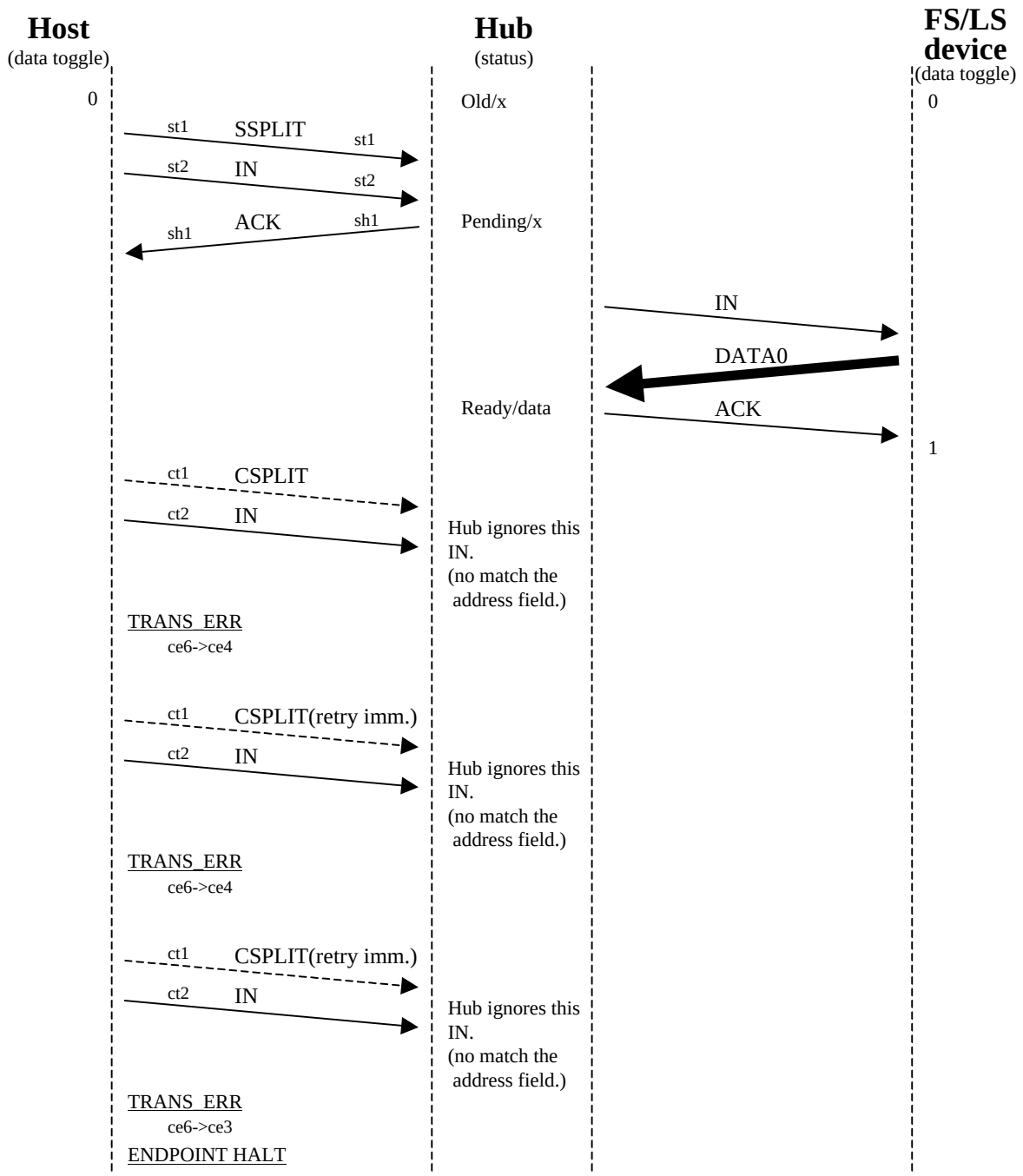


Figure A-31. Normal HS CSPLIT 3 Strikes Smash

Microchip Ex. 1018, p. 501
Microchip v. Aptiv
IPR2024-00494

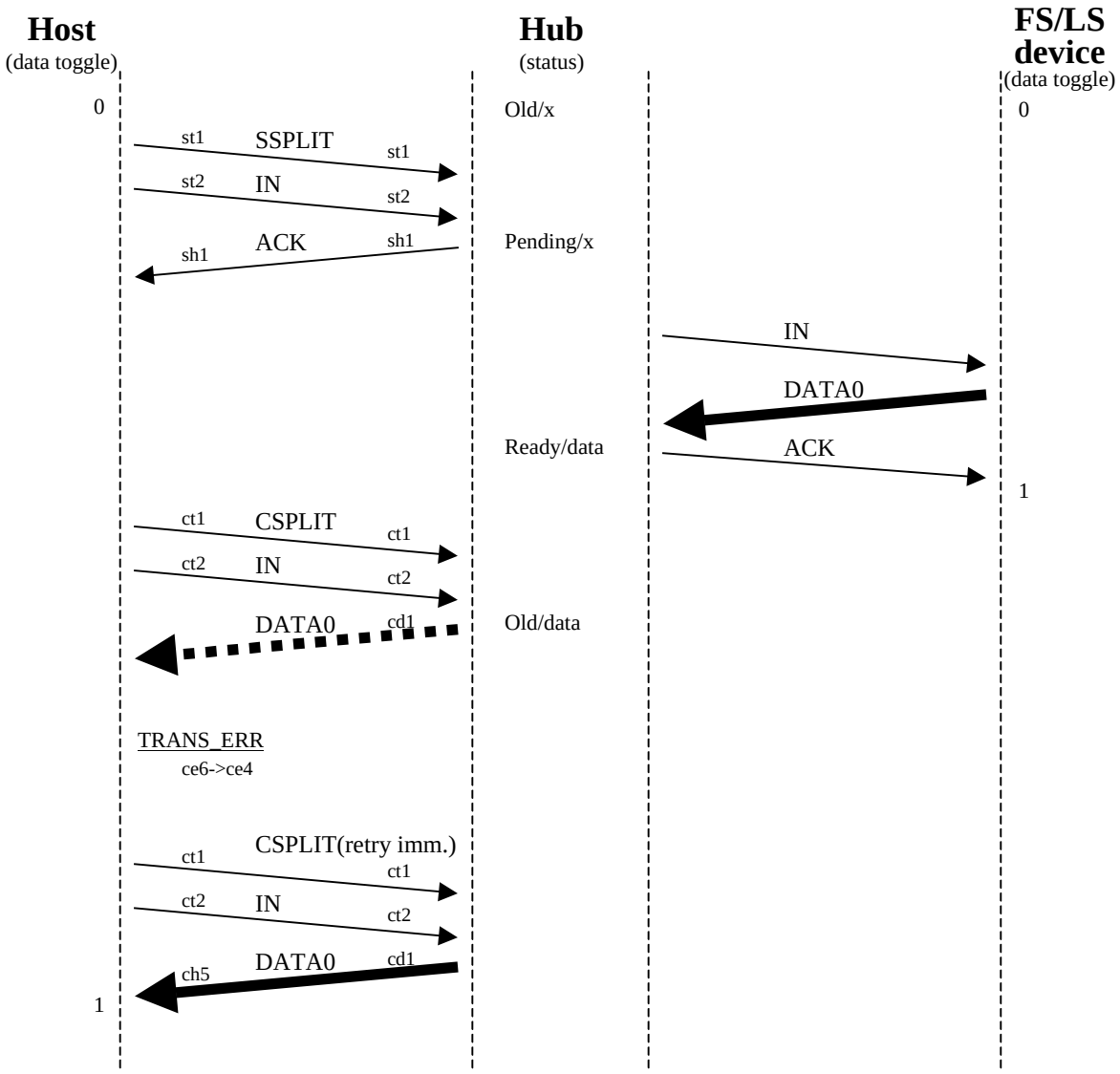


Figure A-32. Normal HS DATA0/1 Smash

Microchip Ex. 1018, p. 502
Microchip v. Aptiv
IPR2024-00494

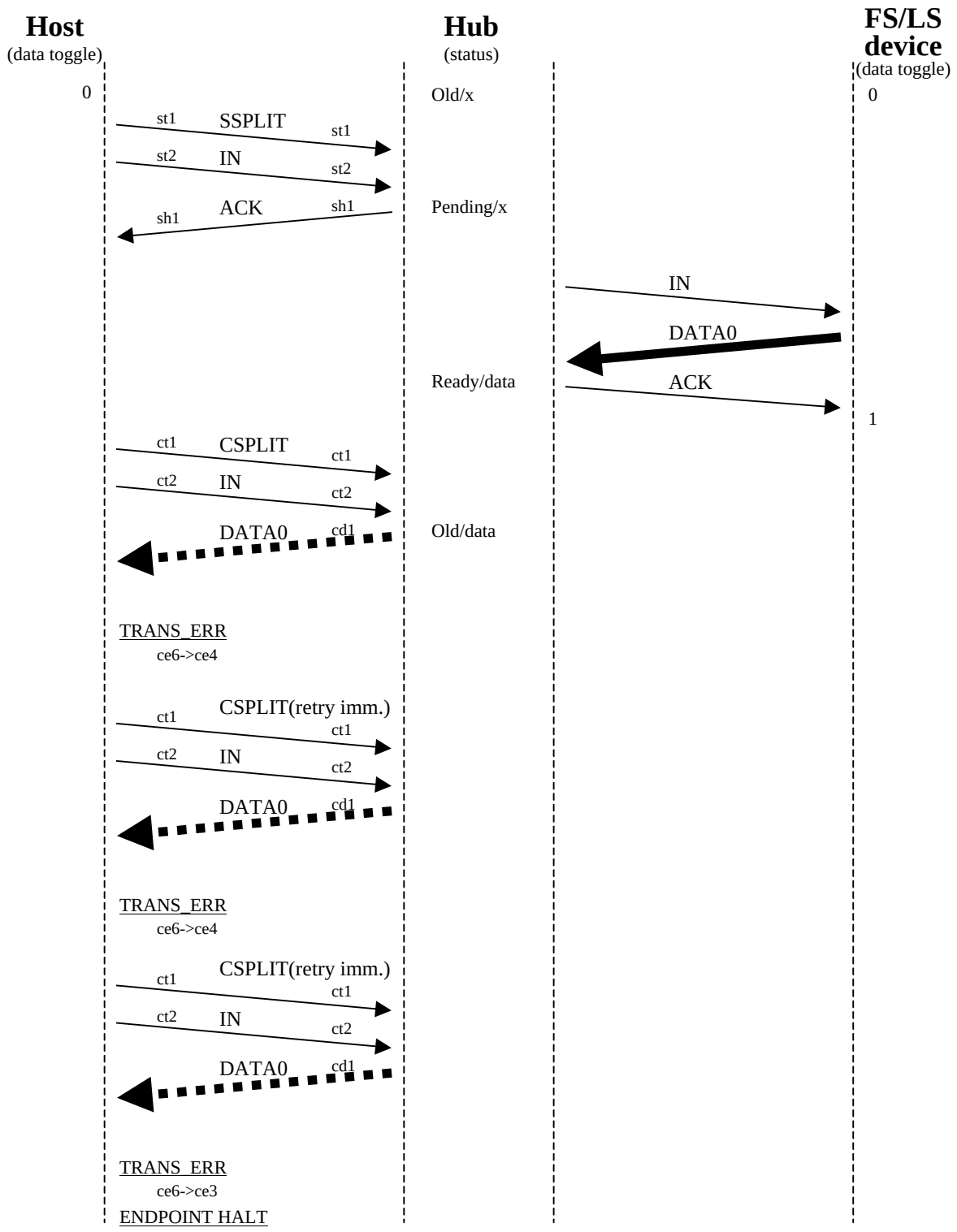


Figure A-33. Normal HS DATA0/1 3 Strikes Smash

Microchip Ex. 1018, p. 503
Microchip v. Aptiv
IPR2024-00494

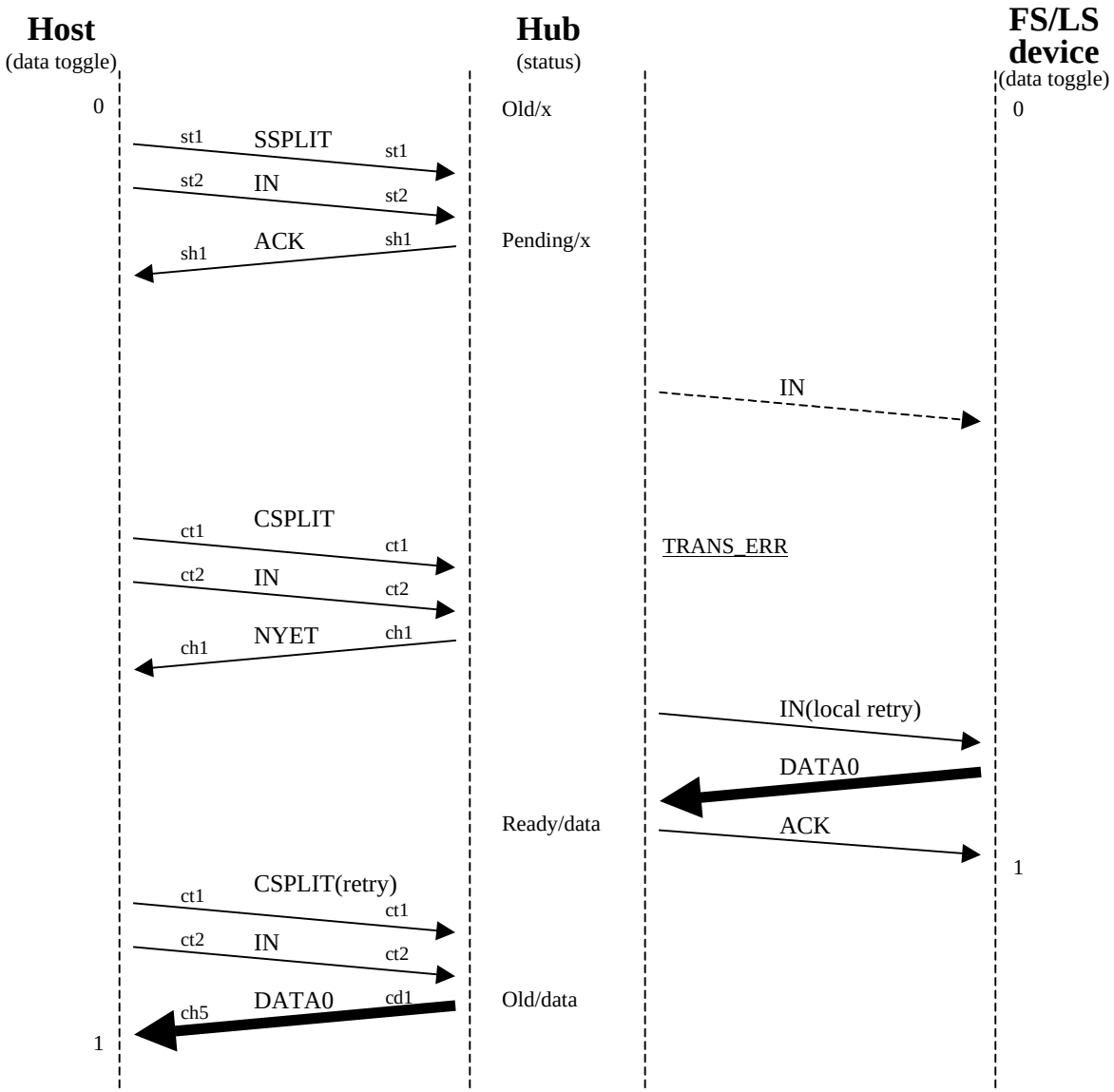


Figure A-34. Normal FS/LS IN Smash

Microchip Ex. 1018, p. 504
Microchip v. Aptiv
IPR2024-00494

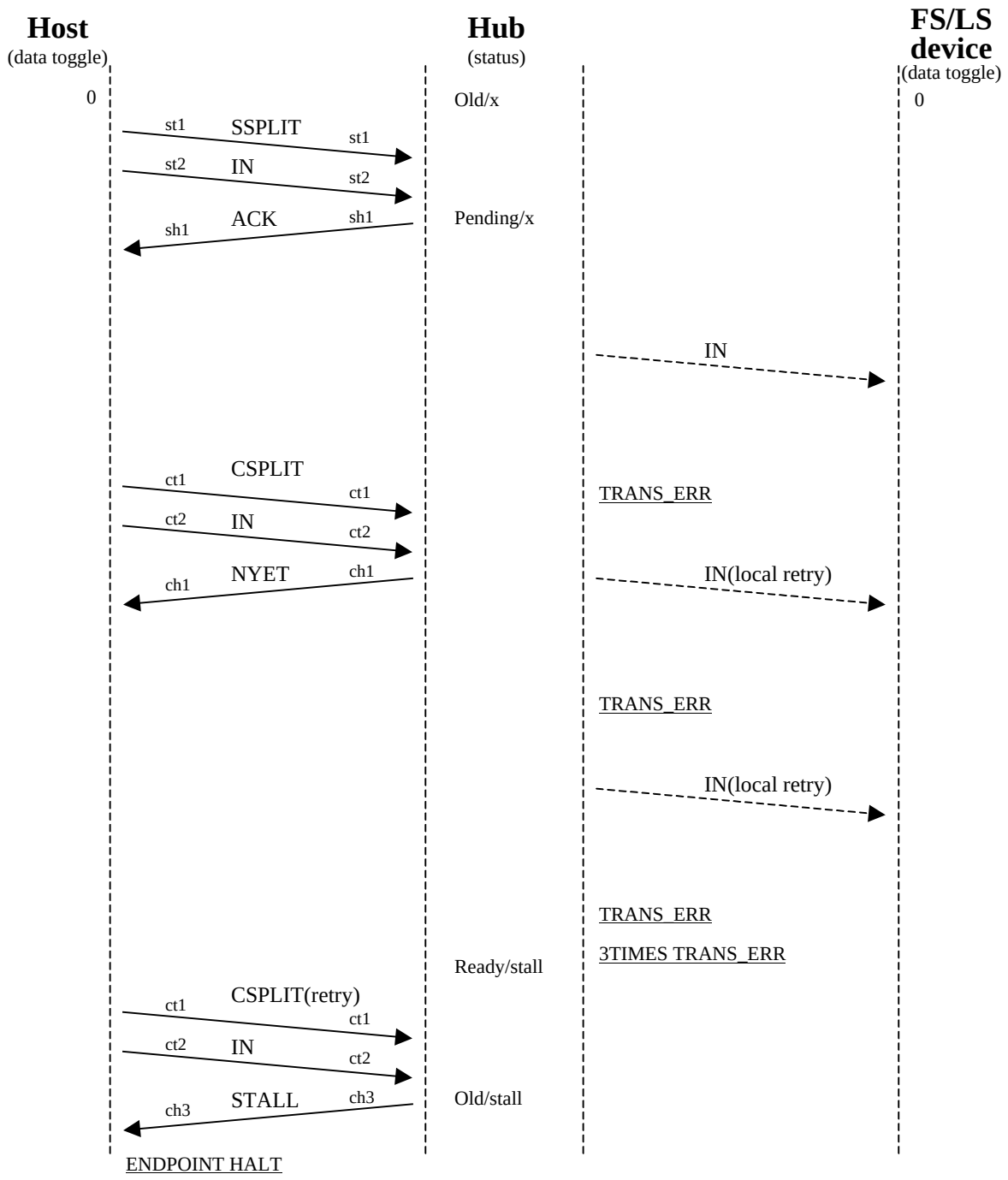


Figure A-35. Normal FS/LS IN 3 Strikes Smash

Microchip Ex. 1018, p. 505
Microchip v. Aptiv
IPR2024-00494

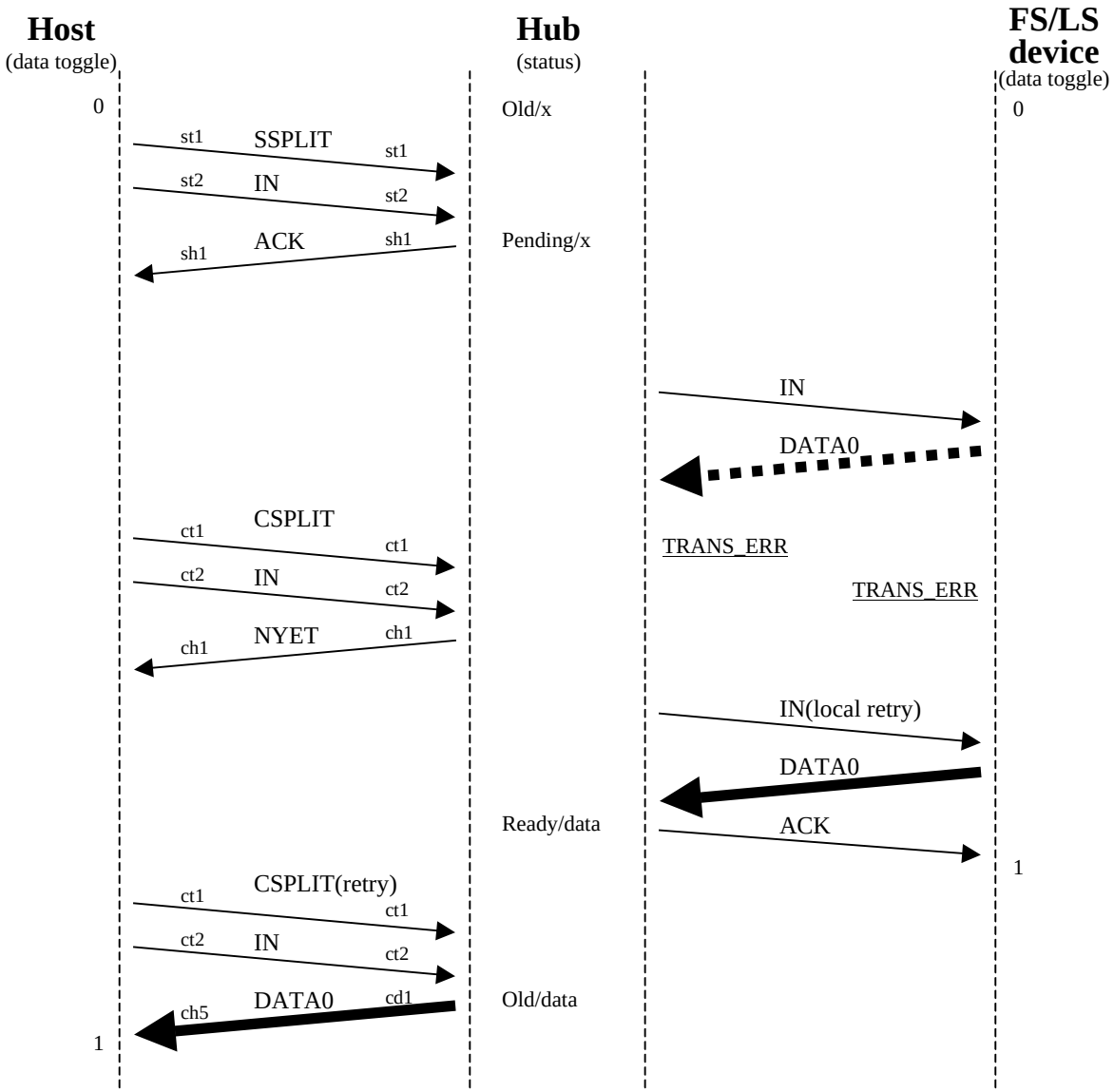


Figure A-36. Normal FS/LS DATA0/1 Smash

Microchip Ex. 1018, p. 506
Microchip v. Aptiv
IPR2024-00494

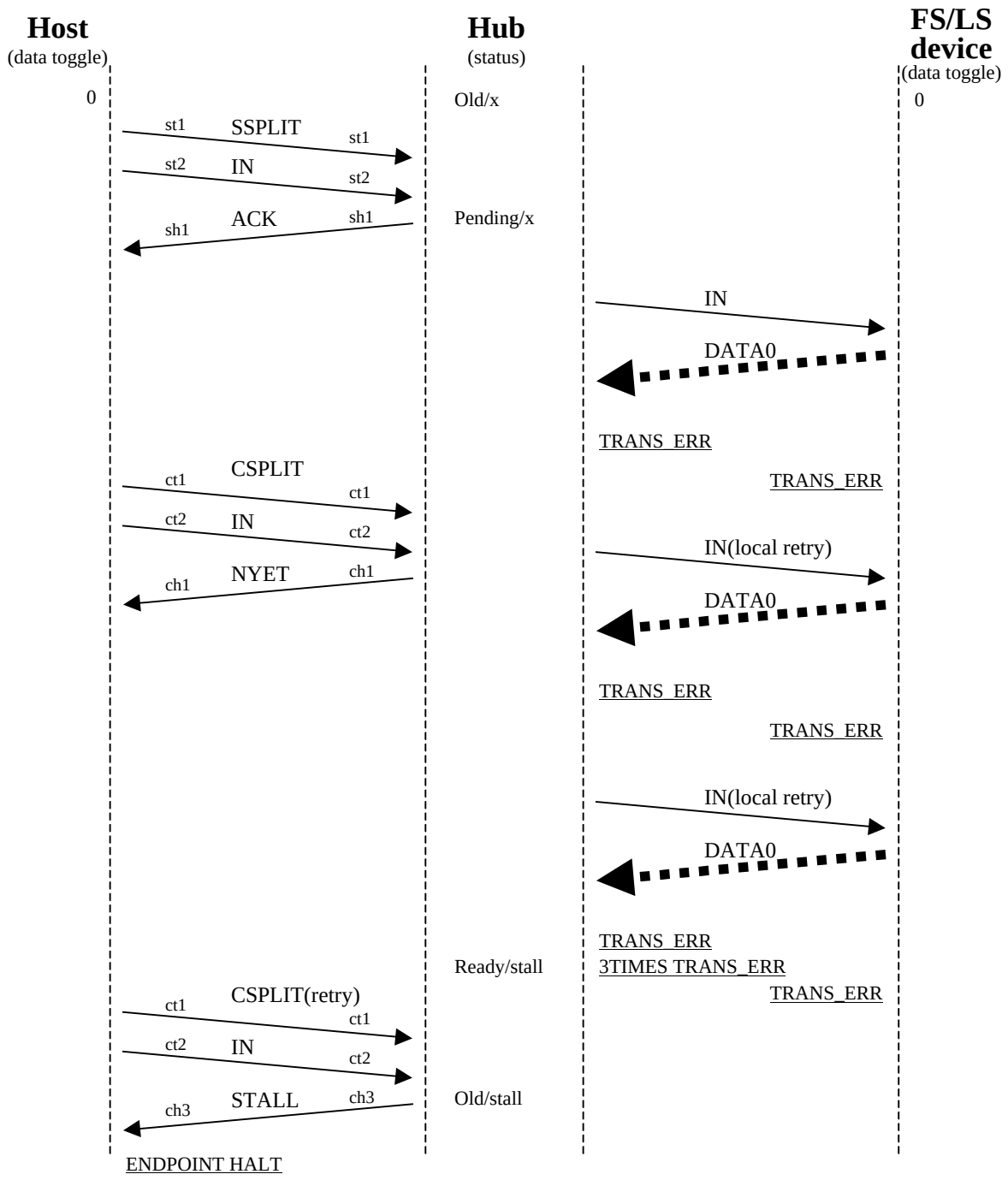


Figure A-37. Normal FS/LS DATA0/1 3 Strikes Smash

Microchip Ex. 1018, p. 507
Microchip v. Aptiv
IPR2024-00494

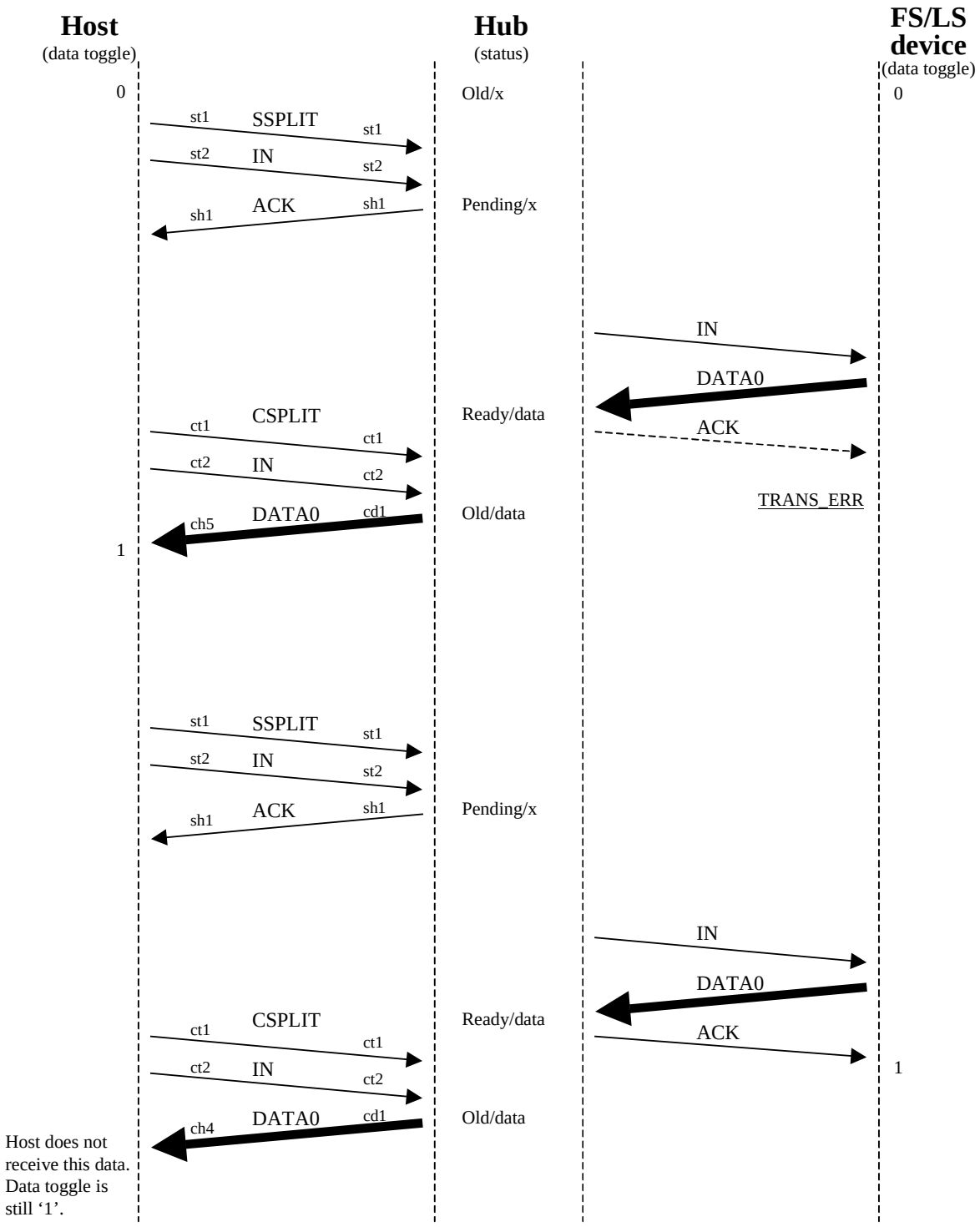


Figure A-38. Normal FS/LS ACK Smash

Microchip Ex. 1018, p. 508
Microchip v. Aptiv
IPR2024-00494

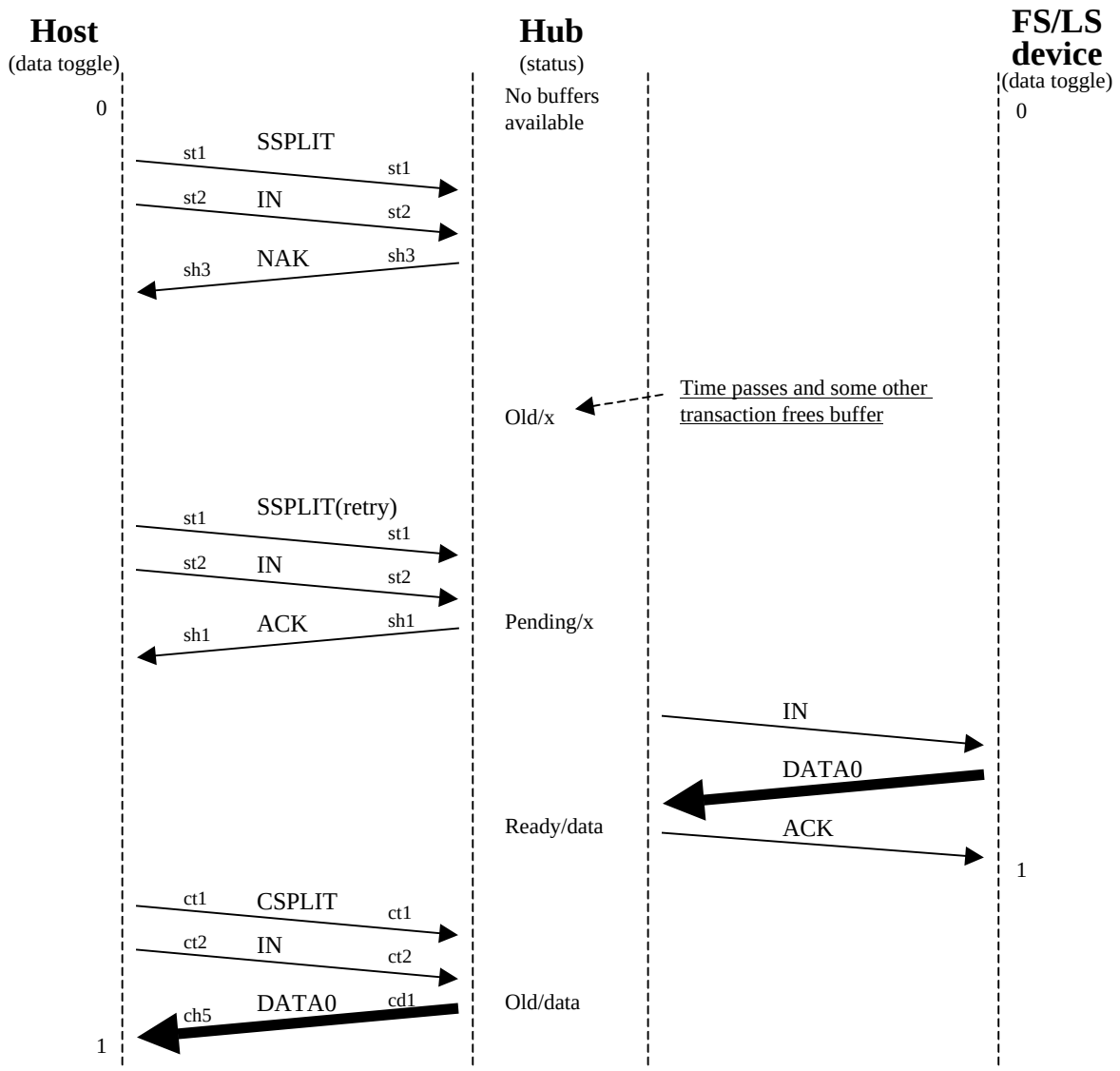


Figure A-39. No Buffer Available No Smash(HS NAK(S))

Microchip Ex. 1018, p. 509
Microchip v. Aptiv
IPR2024-00494

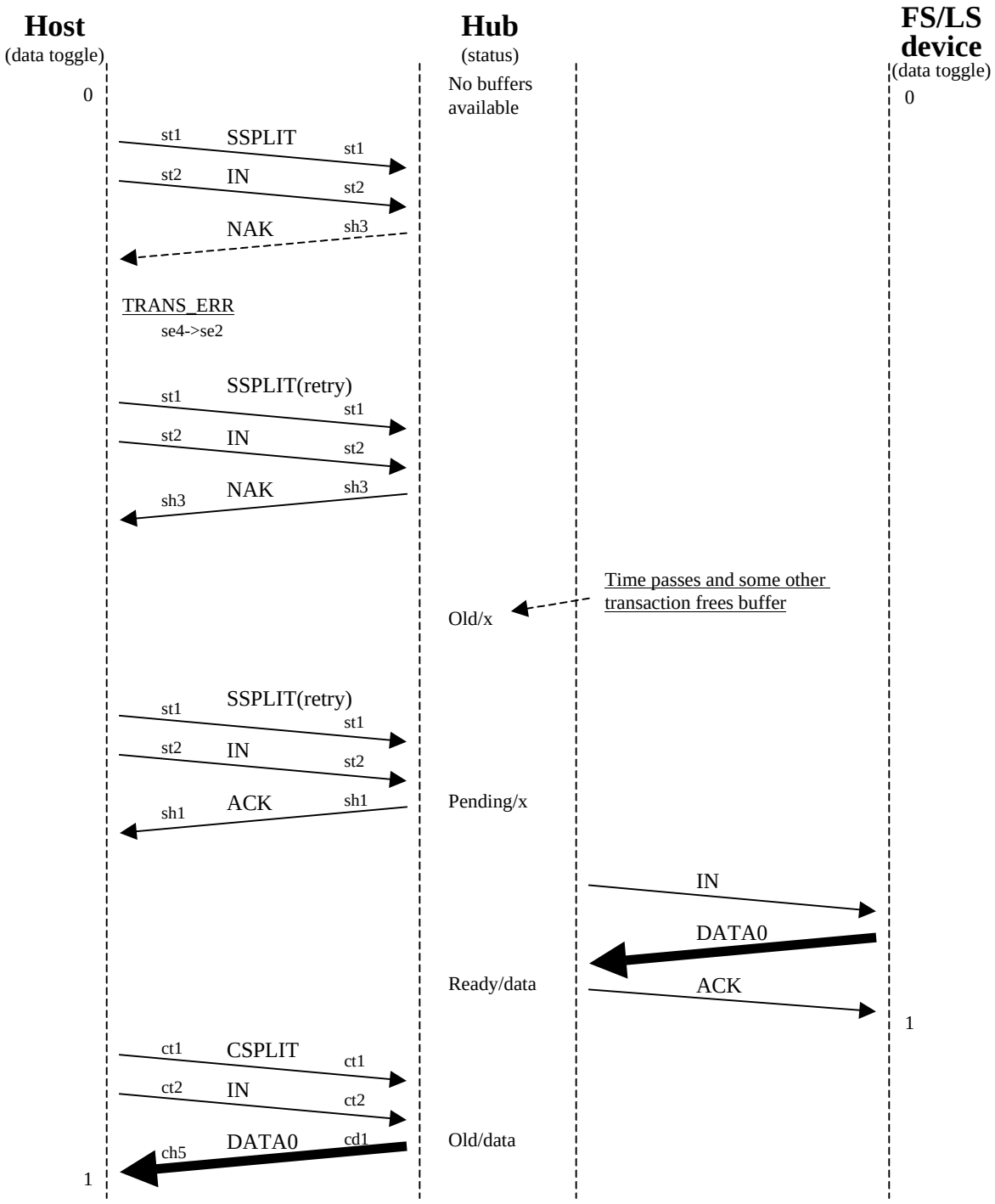


Figure A-40. No Buffer Available HS NAK(S) Smash

Microchip Ex. 1018, p. 510
Microchip v. Aptiv
IPR2024-00494

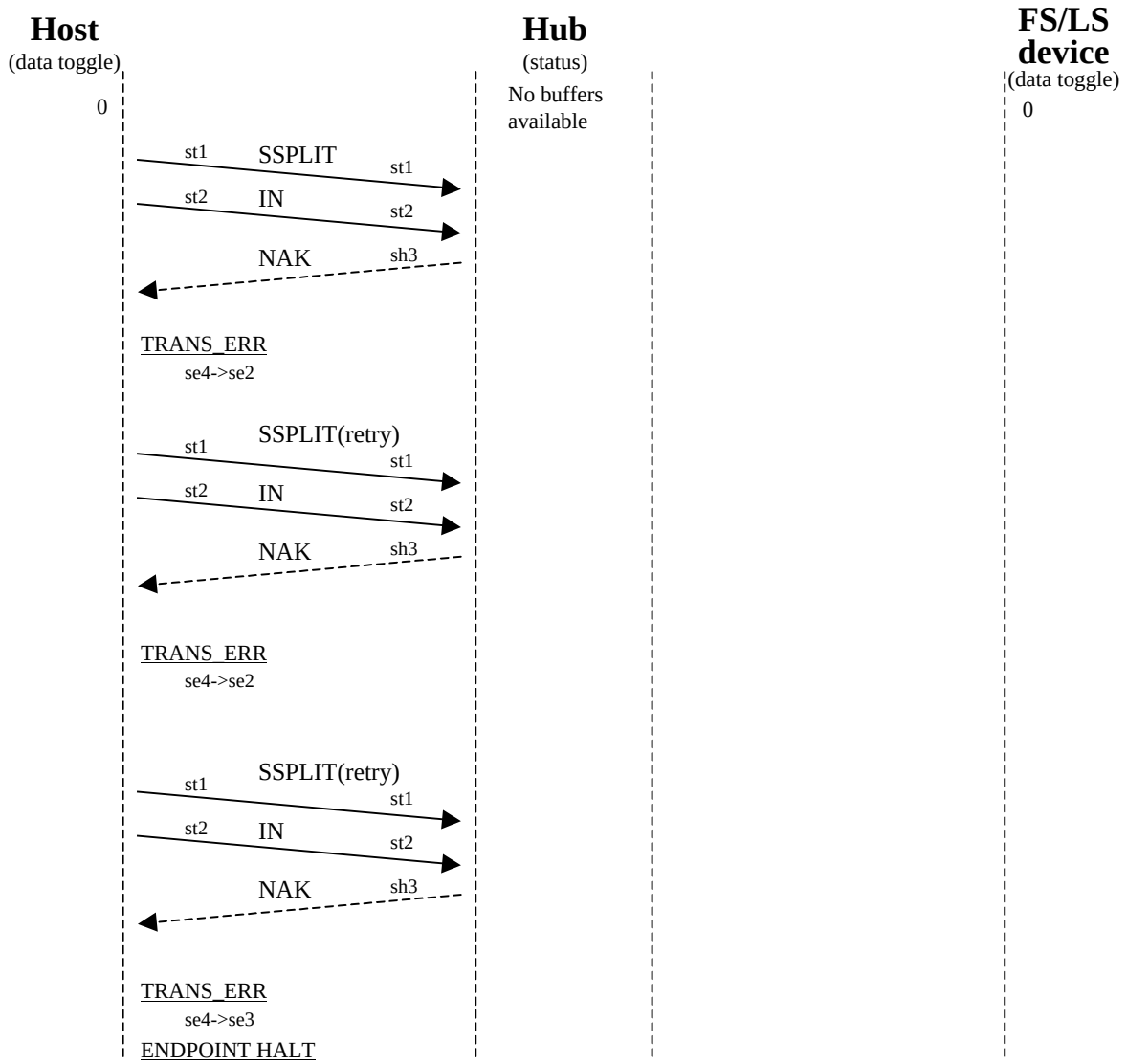


Figure A-41. No Buffer Available HS NAK(S) 3 Strikes Smash

Microchip Ex. 1018, p. 511
Microchip v. Aptiv
IPR2024-00494

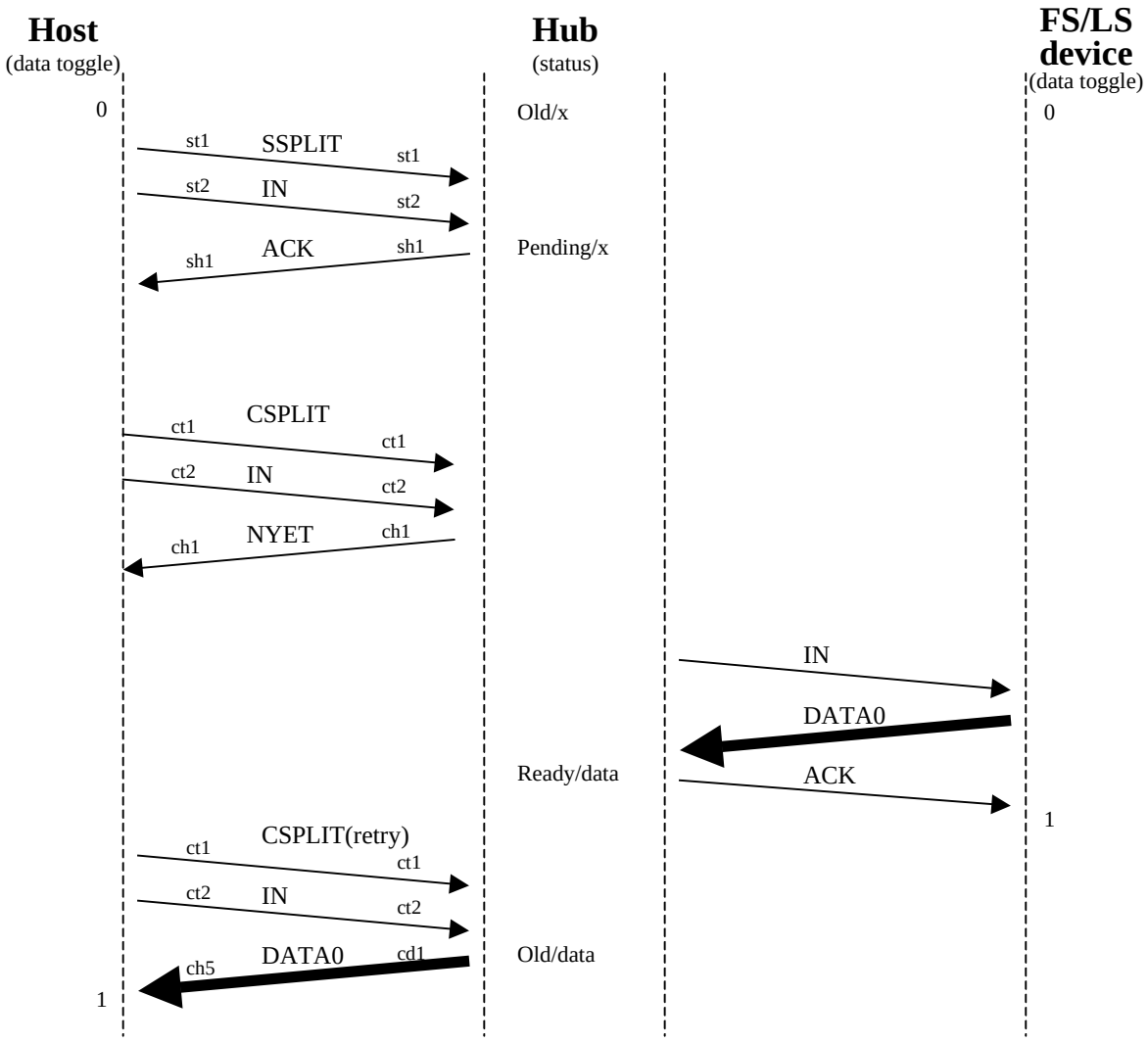


Figure A-42. CS Earlier No Smash(HS NYET)

Microchip Ex. 1018, p. 512
Microchip v. Aptiv
IPR2024-00494

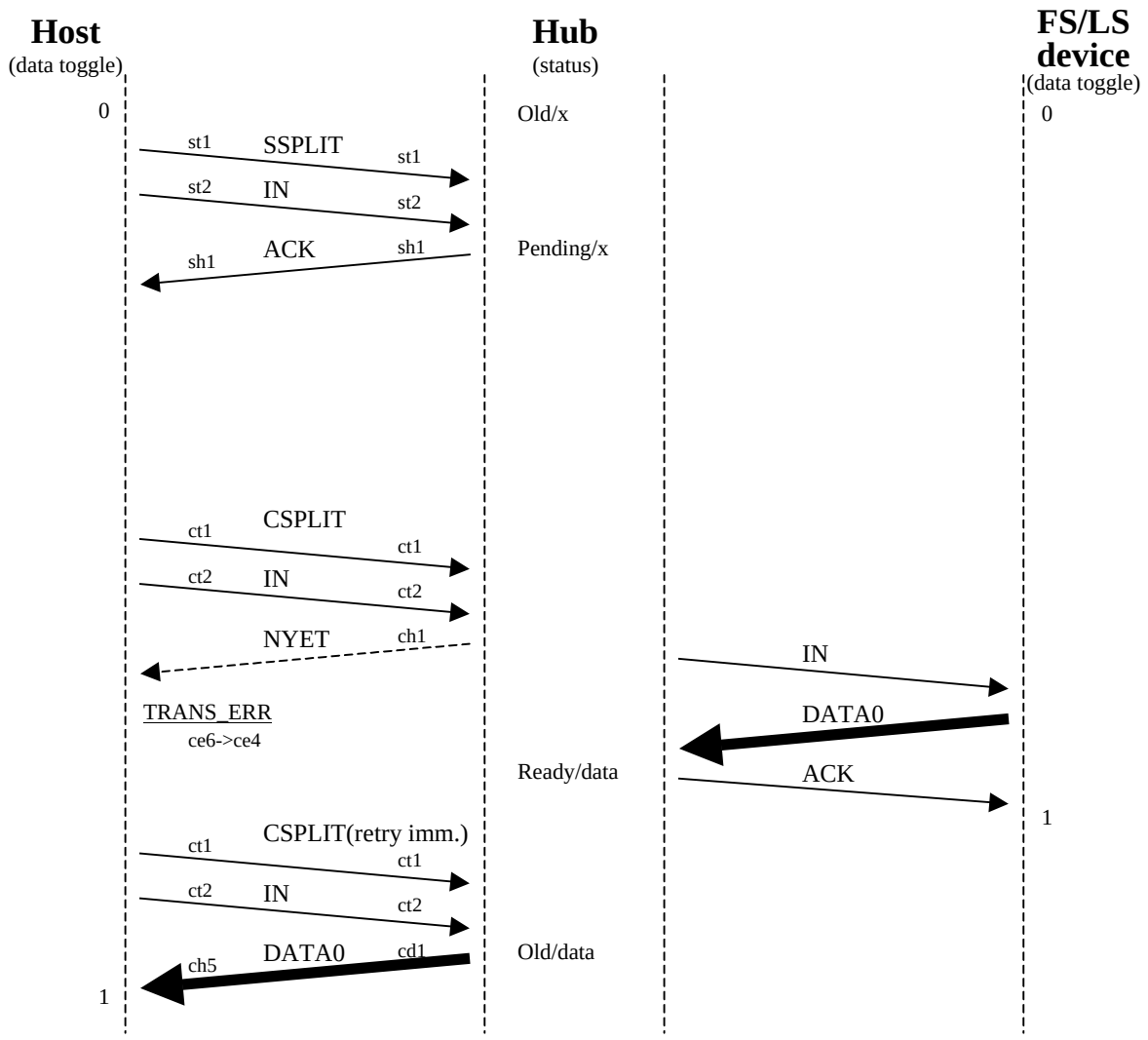


Figure A-43. CS Earlier HS NYET Smash(case 1)

Microchip Ex. 1018, p. 513
Microchip v. Aptiv
IPR2024-00494

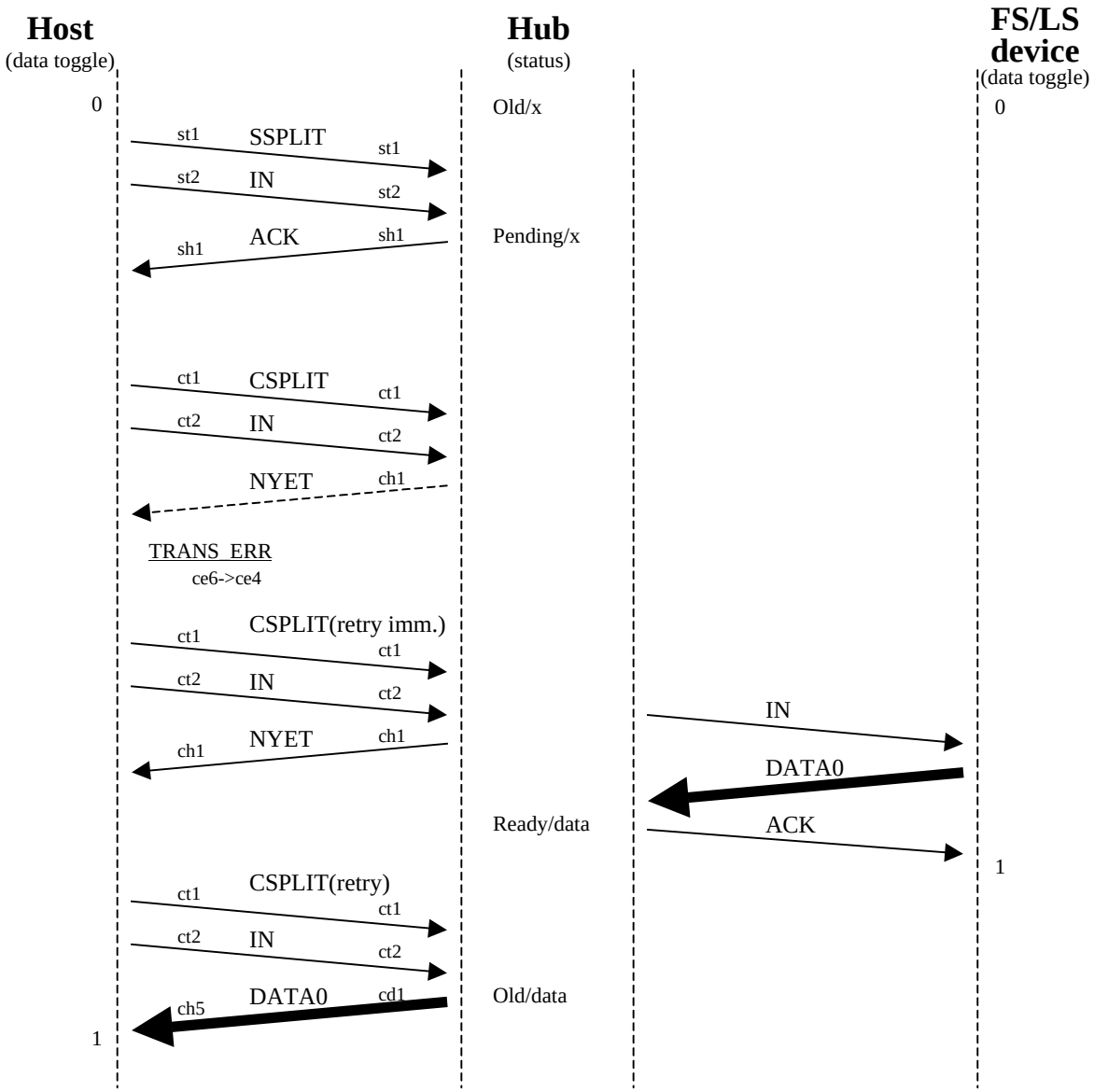


Figure A-44. CS Earlier HS NYET Smash(case 2)

Microchip Ex. 1018, p. 514
Microchip v. Aptiv
IPR2024-00494

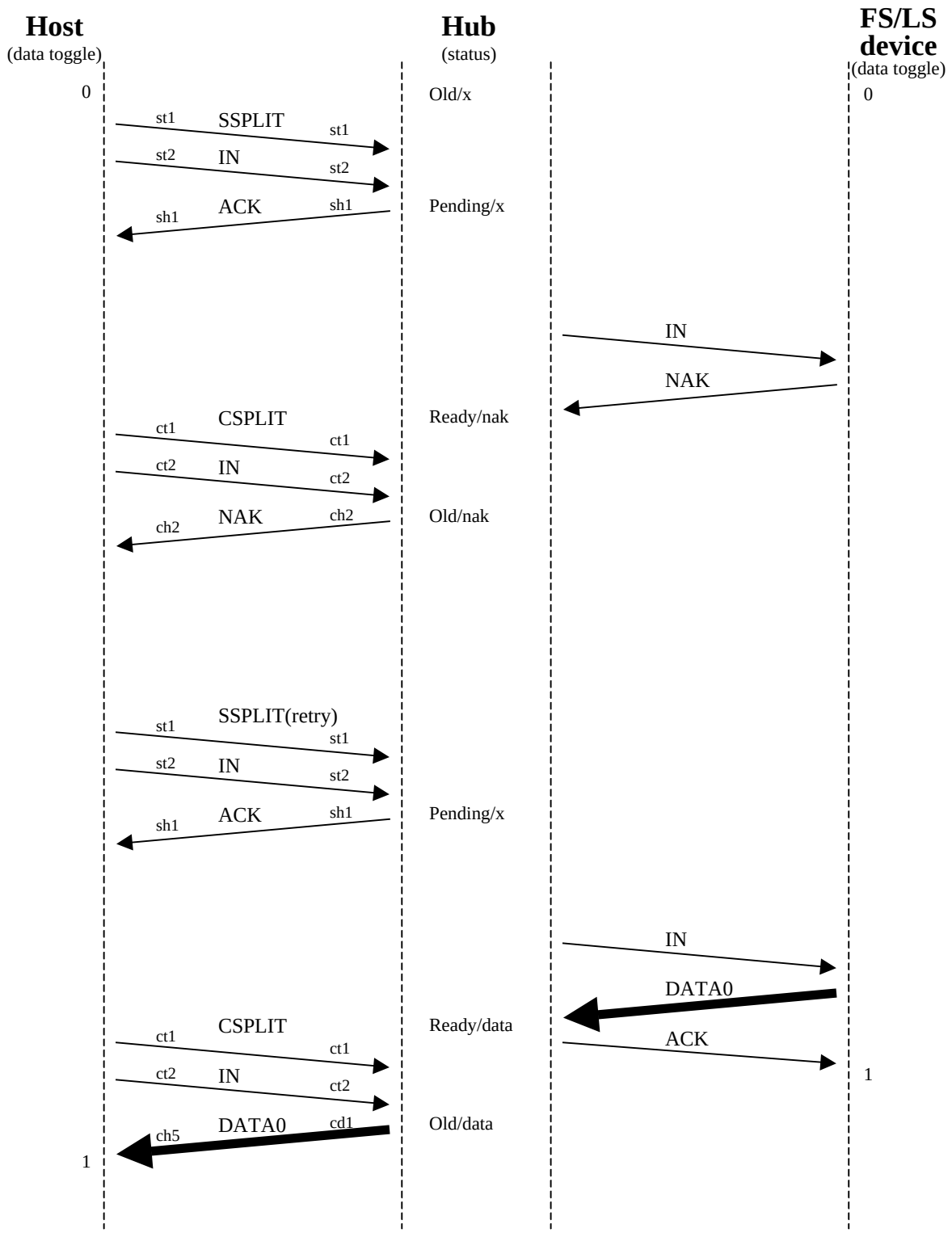


Figure A-45. Device Busy No Smash(FS/LS NAK)

Microchip Ex. 1018, p. 515
Microchip v. Aptiv
IPR2024-00494

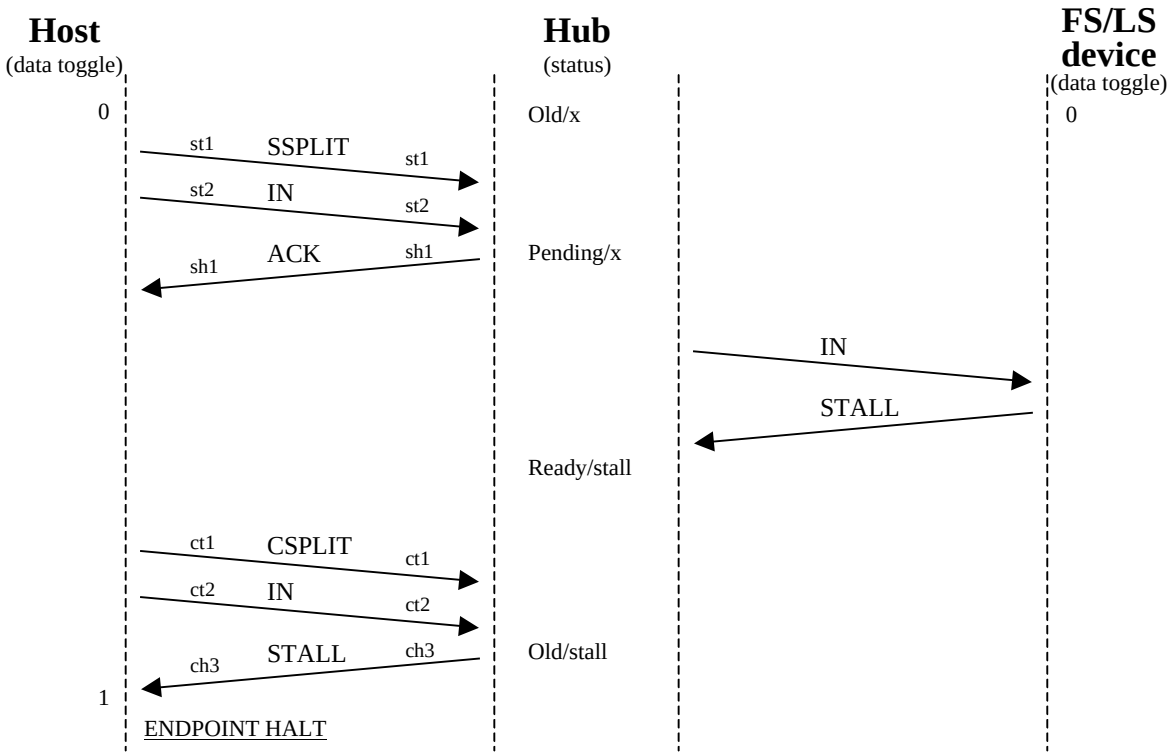


Figure A-46. Device Stall No Smash(FS/LS STALL)

Microchip Ex. 1018, p. 516
Microchip v. Aptiv
IPR2024-00494

A.3 Interrupt OUT Transaction Examples

Legend:

(S): Start Split

(C): Complete Split

Summary of cases for Interrupt OUT transaction

- Normal cases

Case	Reference Figure	Similar Figure
No smash (FS/LS handshake packet is done by M+1)	Figure A-47	
HS SSPLIT smash		Figure A-48
HS SSPLIT 3 strikes smash	No figure	
HS OUT(S) smash		Figure A-48
HS OUT(S) 3 strikes smash	No figure	
HS DATA0/1 smash	Figure A-48	
HS DATA0/1 3 strikes smash	No figure	
HS CSPLIT smash	Figure A-49	
HS CSPLIT 3 strikes smash	Figure A-50	
HS OUT(C) smash		Figure A-49
HS OUT(C) 3 strikes smash		Figure A-50
HS ACK(C) smash	Figure A-51	
HS ACK(C) 3 strikes smash	Figure A-52	
FS/LS OUT smash		Figure A-53
FS/LS OUT 3 strikes smash	No figure	
FS/LS DATA0/1 smash	Figure A-53	
FS/LS DATA0/1 3 strikes smash	No figure	
FS/LS ACK smash	Figure A-54	

Microchip Ex. 1018, p. 517
Microchip v. Aptiv
IPR2024-00494

FS/LS ACK 3 strikes smash	No figure	

- Searcing

Case	Reference Figure	Similar Figure
No smash	Figure A-55	

- CS(Complete-split transaction) earlier cases

Case	Reference Figure	Similar Figure
No smash (HS NYETand FS/LS handshake packet is done by M+2)	Figure A-56	
No smash(HS NYET and FS/LS handshake packet is done by M+3)	Figure A-57	
HS NYET smash	Figure A-58	
HS NYET 3 strikes smash	Figure A-59	

- Abort and Free cases

Case	Reference Figure	Similar Figure
No smash and abort (HS NYETand FS/LS transaction is continued at end of M+3)	Figure A-60	
No smash and free(HS NYETand FS/LS transaction is not started at end of M+3)	Figure A-61	

- FS/LS transaction error cases

Case	Reference Figure	Similar Figure
HS ERR smash		Figure A-51
HS ERR 3 strikes smash		Figure A-52

- Device busy cases

Microchip Ex. 1018, p. 518
Microchip v. Aptiv
IPR2024-00494

Case	Reference Figure	Similar Figure
No smash(HS NAK(C))	Figure A-62	
HS NAK(C) smash		Figure A-51
HS NAK(C) 3 strikes smash		Figure A-52
FS/LS NAK smash		Figure A-53
FS/LS NAK 3 strikes smash	No figure	

- Device stall cases

Case	Reference Figure	Similar Figure
No smash	Figure A-63	
HS STALL(C) smash		Figure A-51
HS STALL(C) 3 strikes smash		Figure A-52
FS/LS STALL smash		Figure A-53
FS/LS STALL 3 strikes smash	No figure	

Microchip Ex. 1018, p. 519
Microchip v. Aptiv
IPR2024-00494

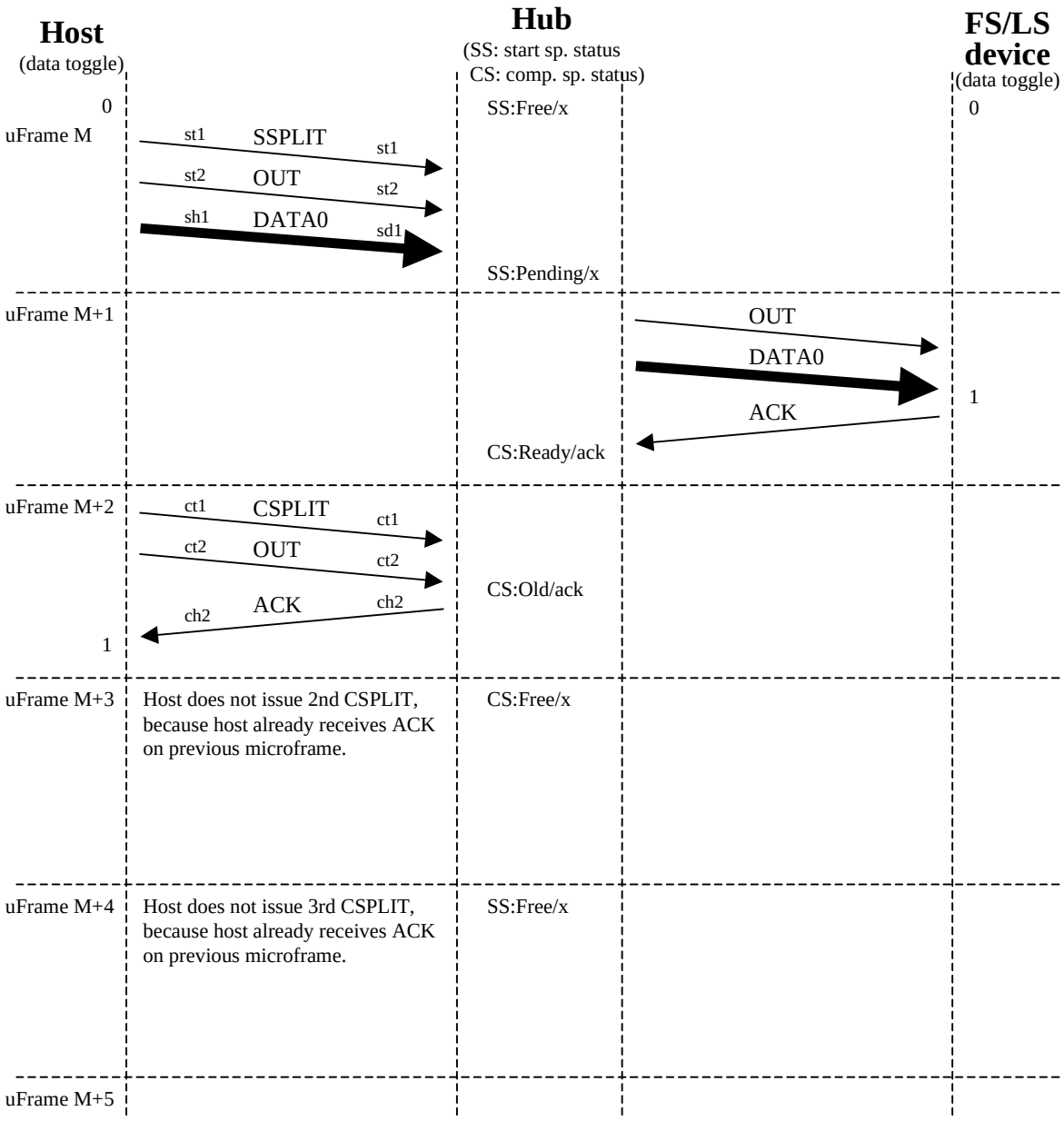


Figure A-47. Normal No Smash(FS/LS Handshake Packet is Done by M+1)

Microchip Ex. 1018, p. 520
Microchip v. Aptiv
IPR2024-00494

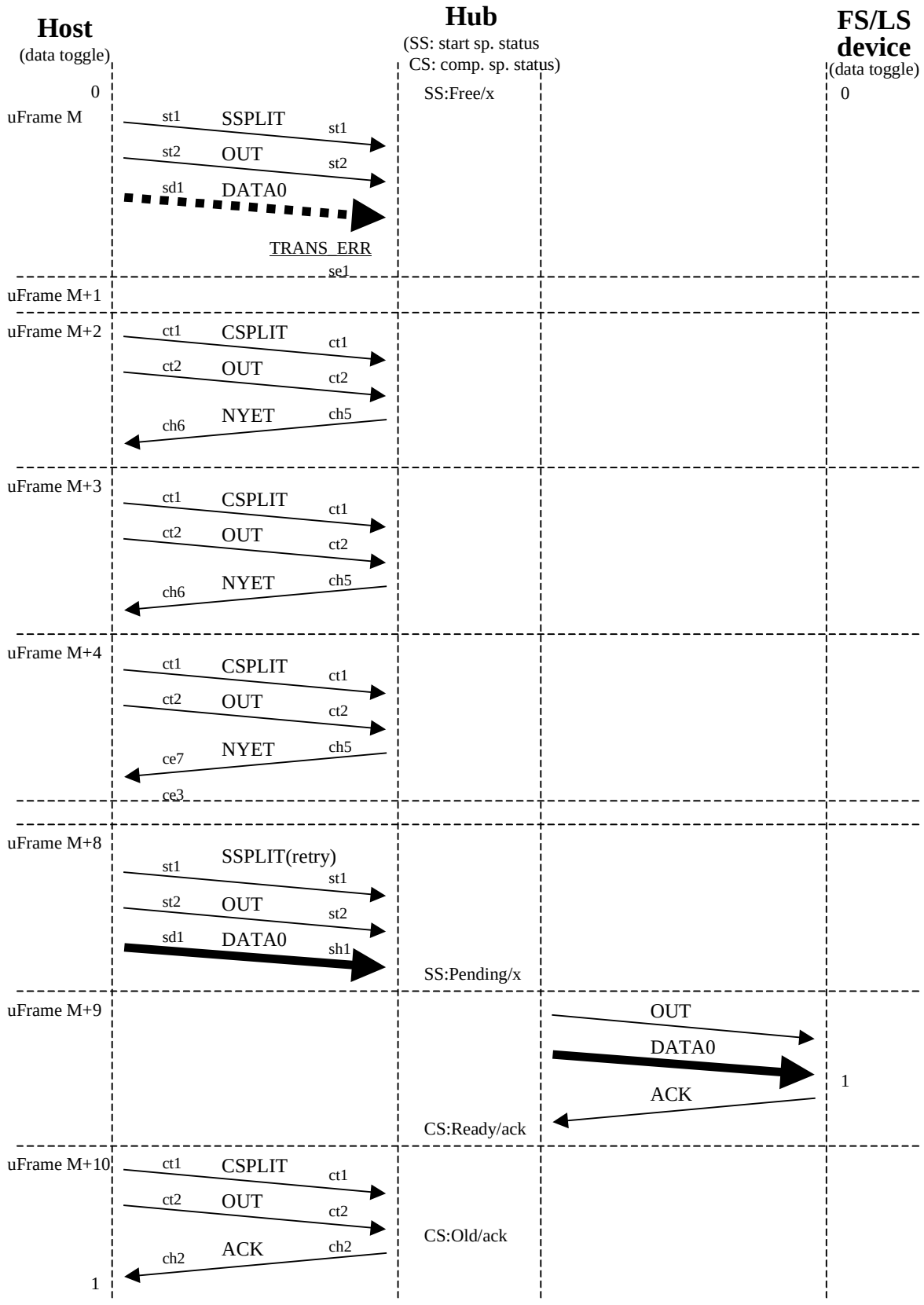


Figure A-48. Normal HS DATA0/1 Smash

Microchip Ex. 1018, p. 521
Microchip v. Aptiv
IPR2024-00494

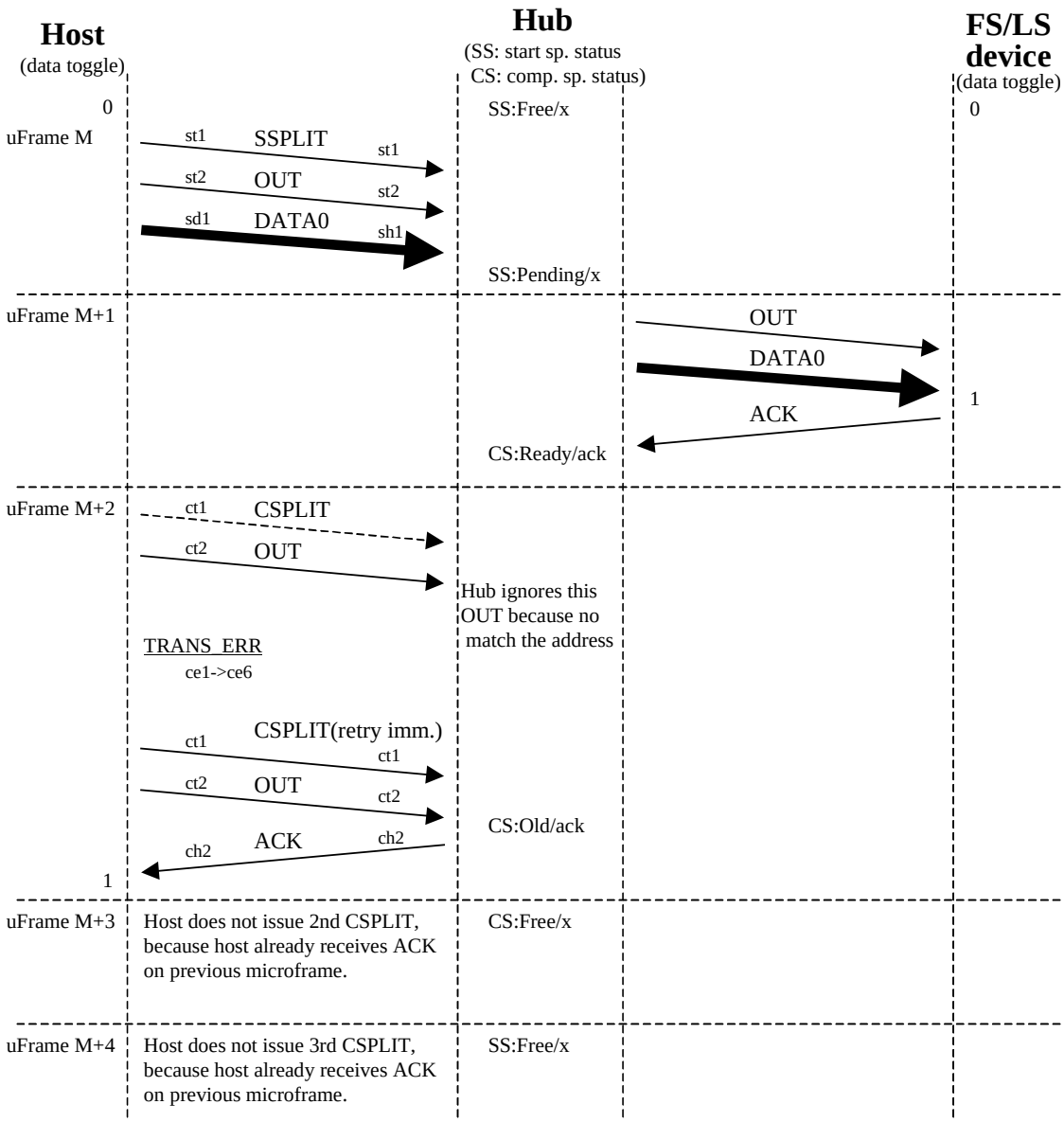


Figure A-49. Normal HS CSPLIT Smash

Microchip Ex. 1018, p. 522
Microchip v. Aptiv
IPR2024-00494

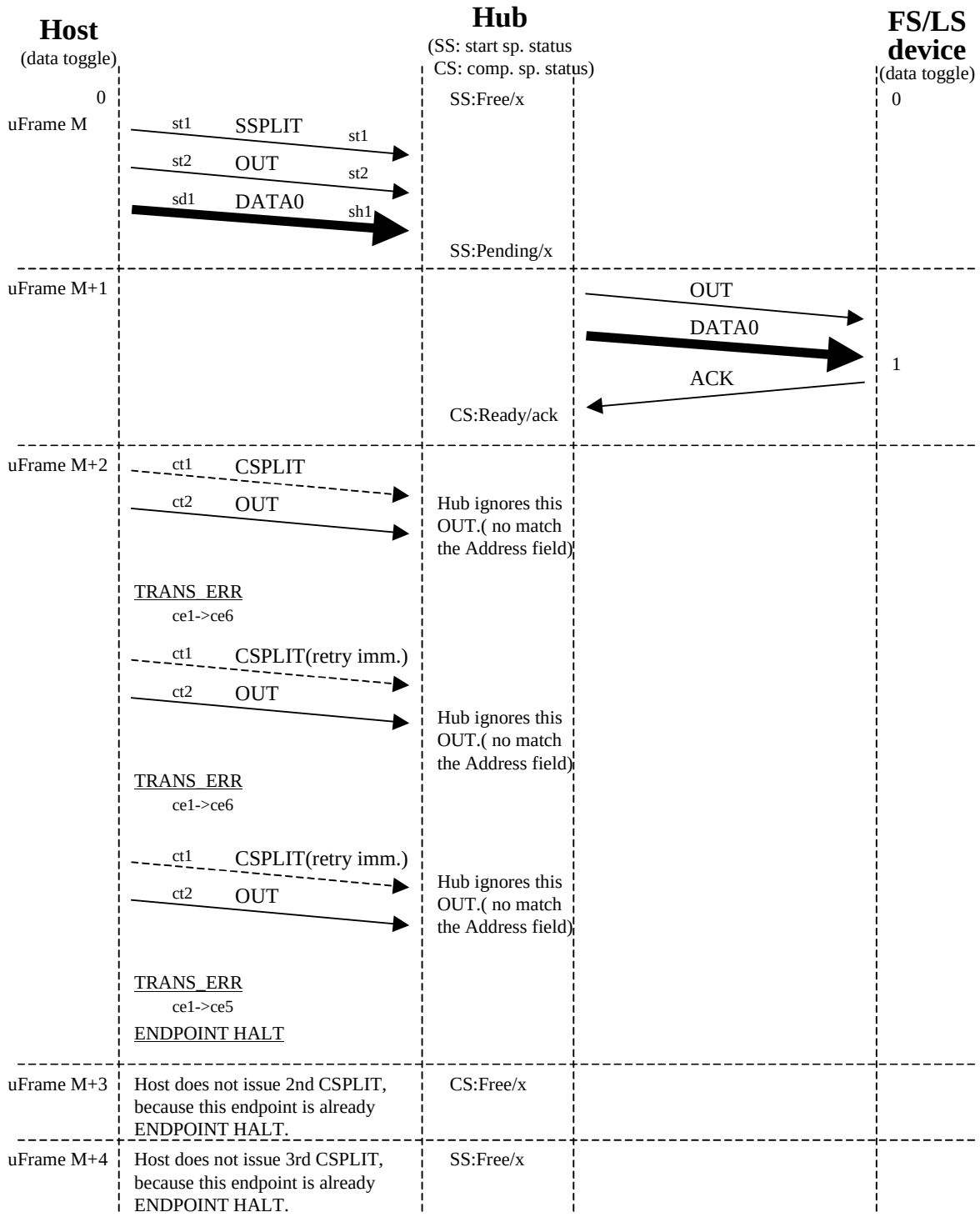


Figure A-50. Normal HS CSPLIT 3 Strikes Smash

Microchip Ex. 1018, p. 523
Microchip v. Aptiv
IPR2024-00494

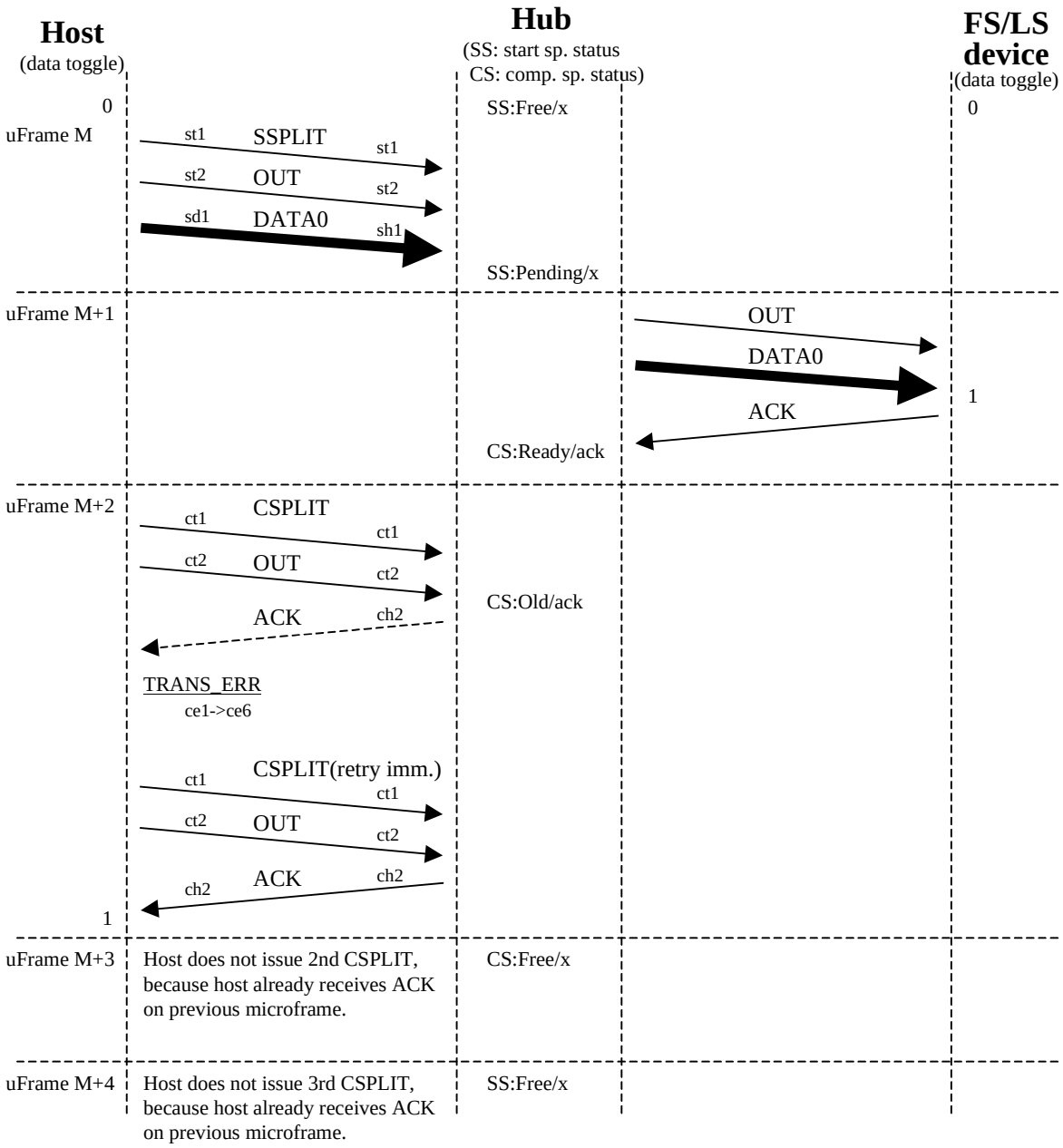


Figure A-51. Normal HS ACK(C) Smash

Microchip Ex. 1018, p. 524
Microchip v. Aptiv
IPR2024-00494

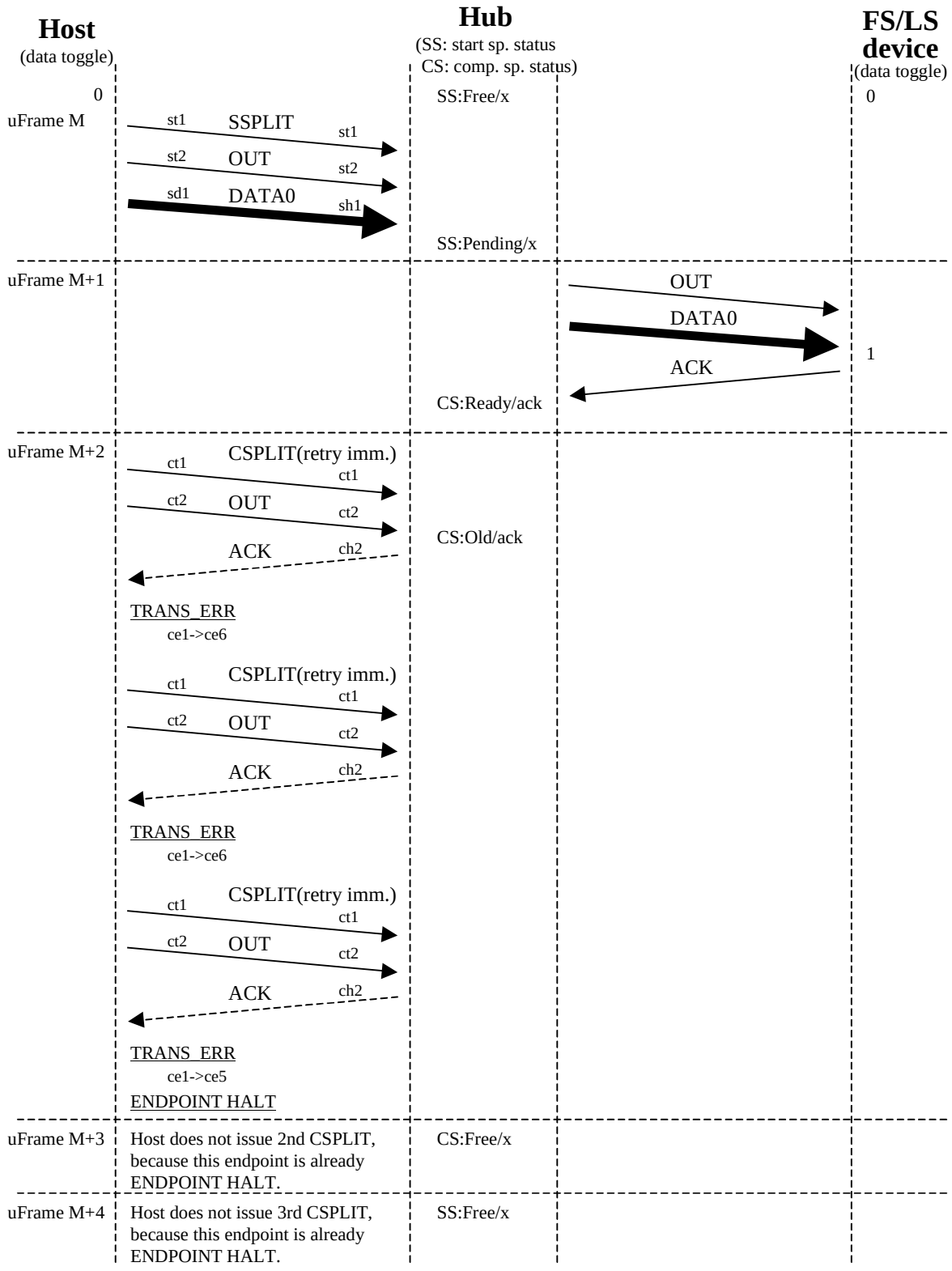


Figure A-52. Normal HS ACK(C) 3 Strikes Smash

Microchip Ex. 1018, p. 525
Microchip v. Aptiv
IPR2024-00494

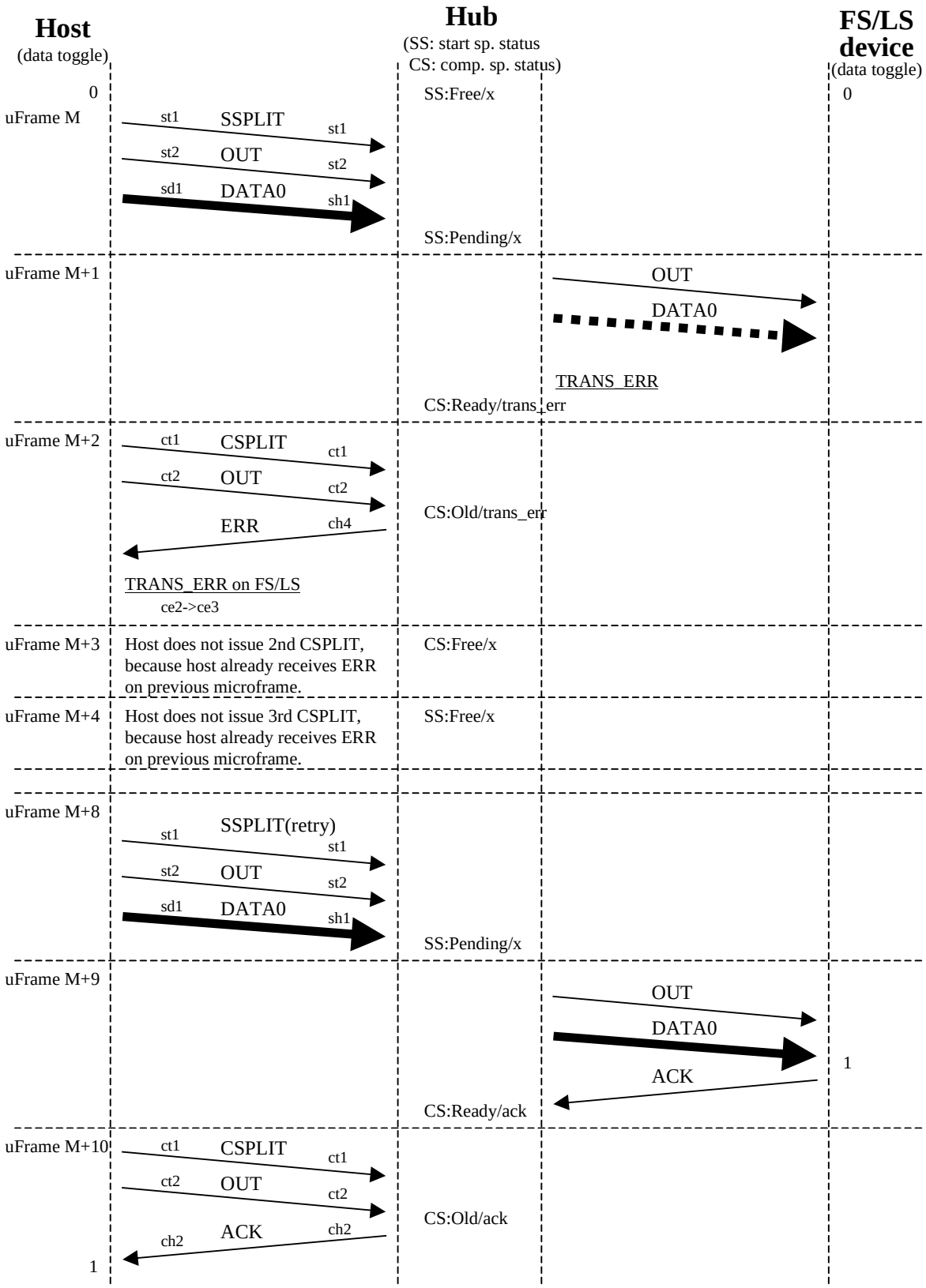


Figure A-53. Normal FS/LS DATA0/1 Smash

Microchip Ex. 1018, p. 526
Microchip v. Aptiv
IPR2024-00494

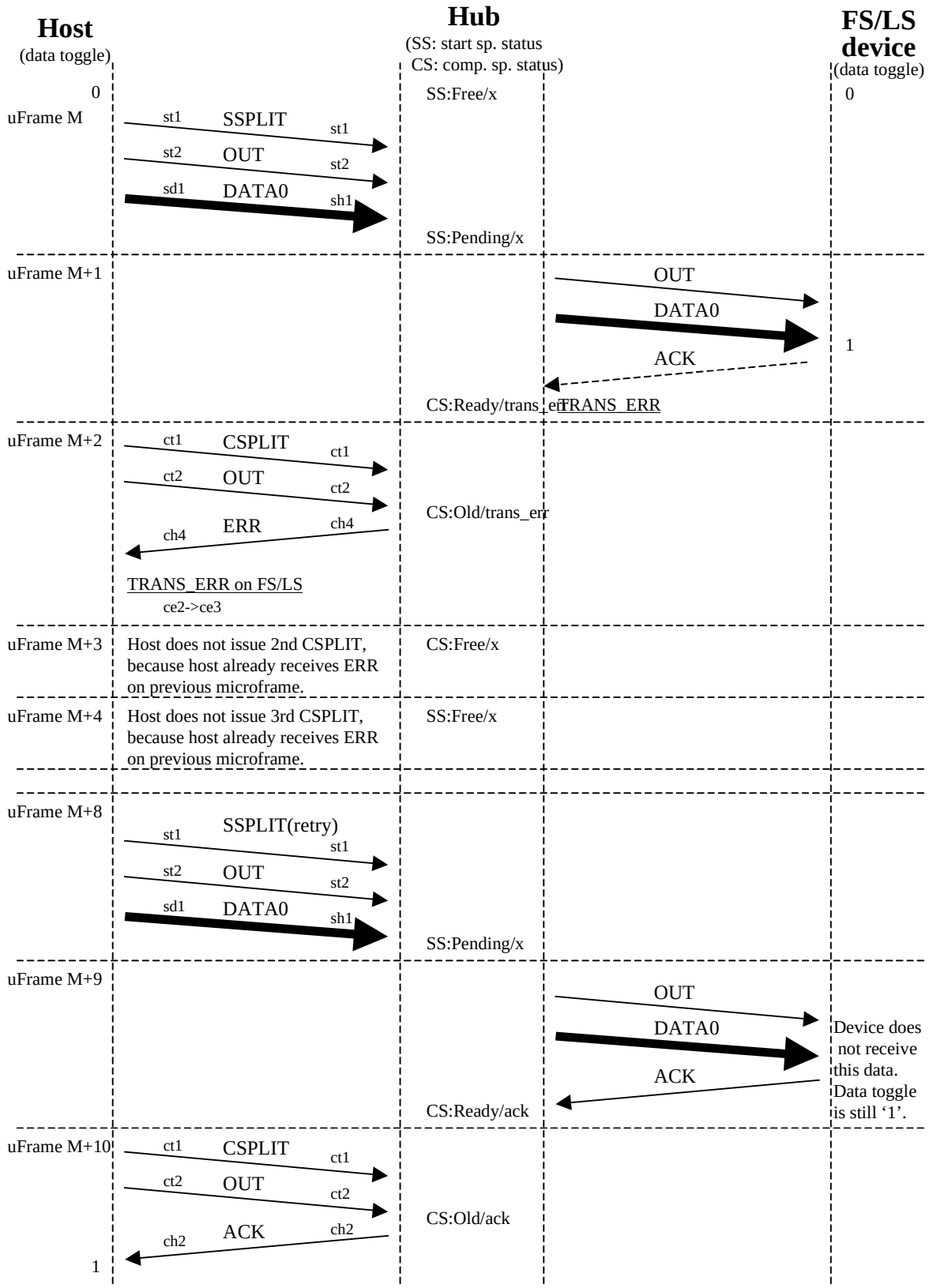


Figure A-54. Normal FS/LS ACK Smash

Microchip Ex. 1018, p. 527
Microchip v. Aptiv
IPR2024-00494

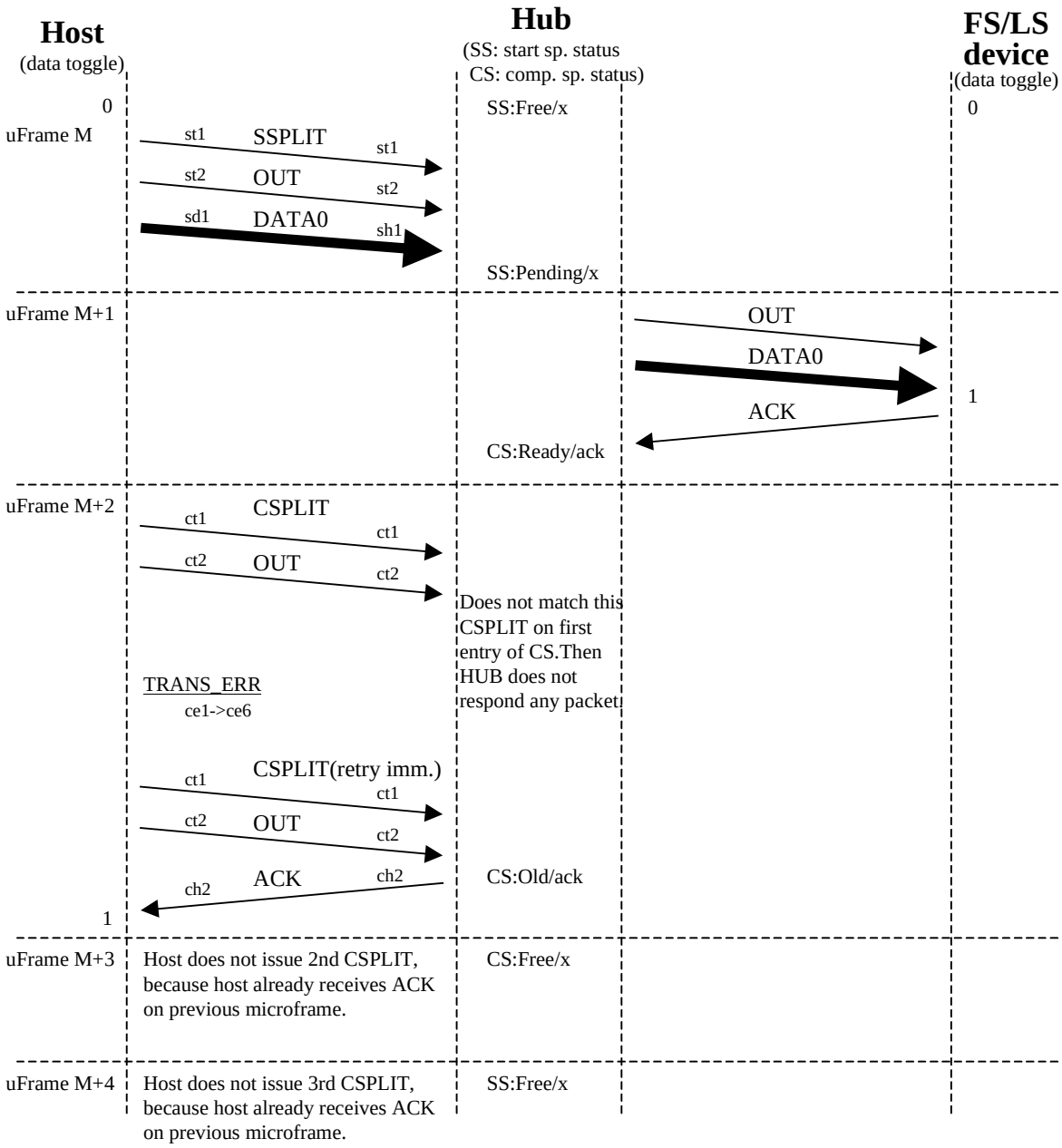


Figure A-55. Searching No Smash

Microchip Ex. 1018, p. 528
Microchip v. Aptiv
IPR2024-00494

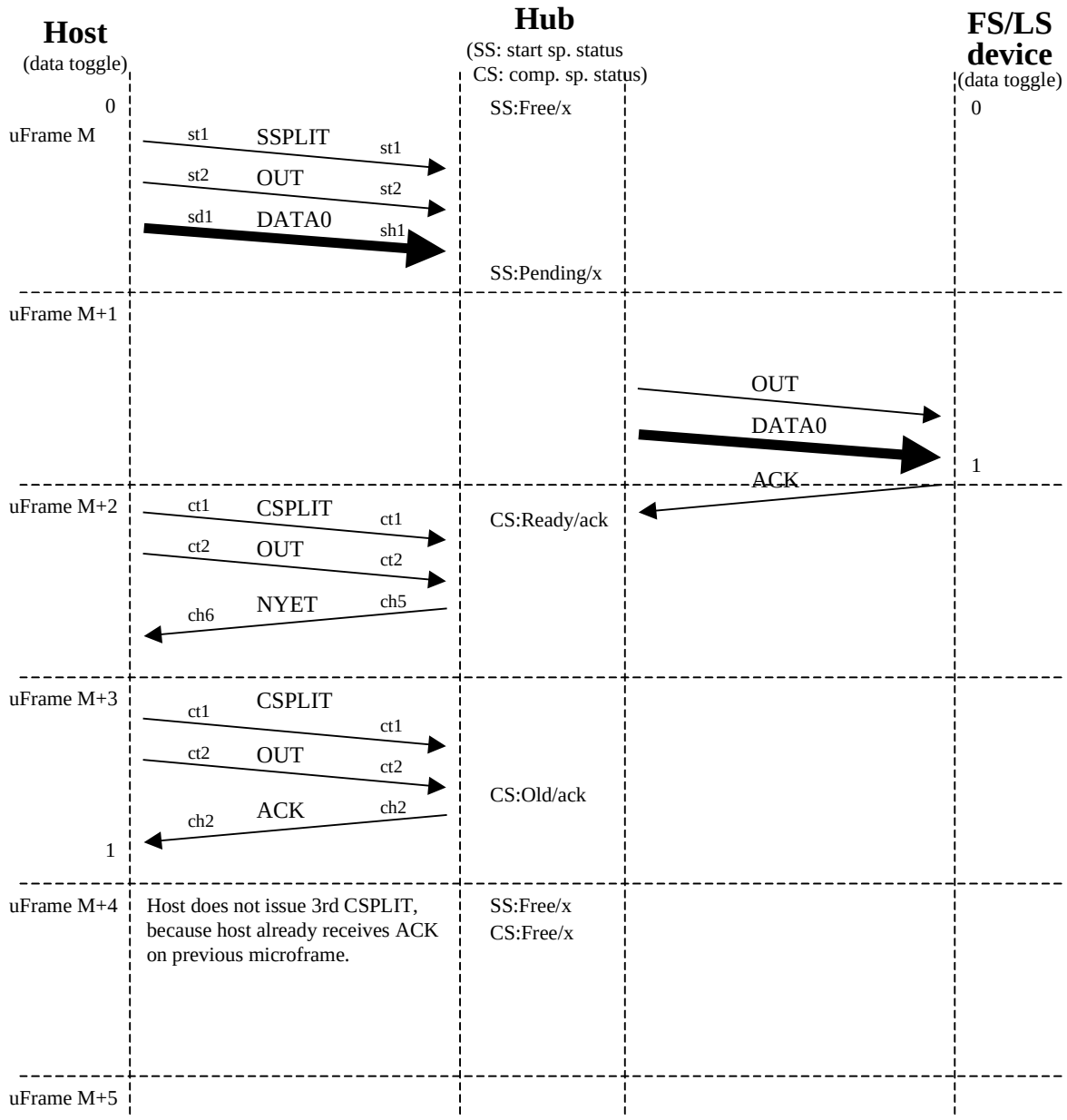


Figure A-56. CS Earlier No Smash(HS NYET and FS/LS Handshake Packet is Done by M+2)

Microchip Ex. 1018, p. 529
Microchip v. Aptiv
IPR2024-00494

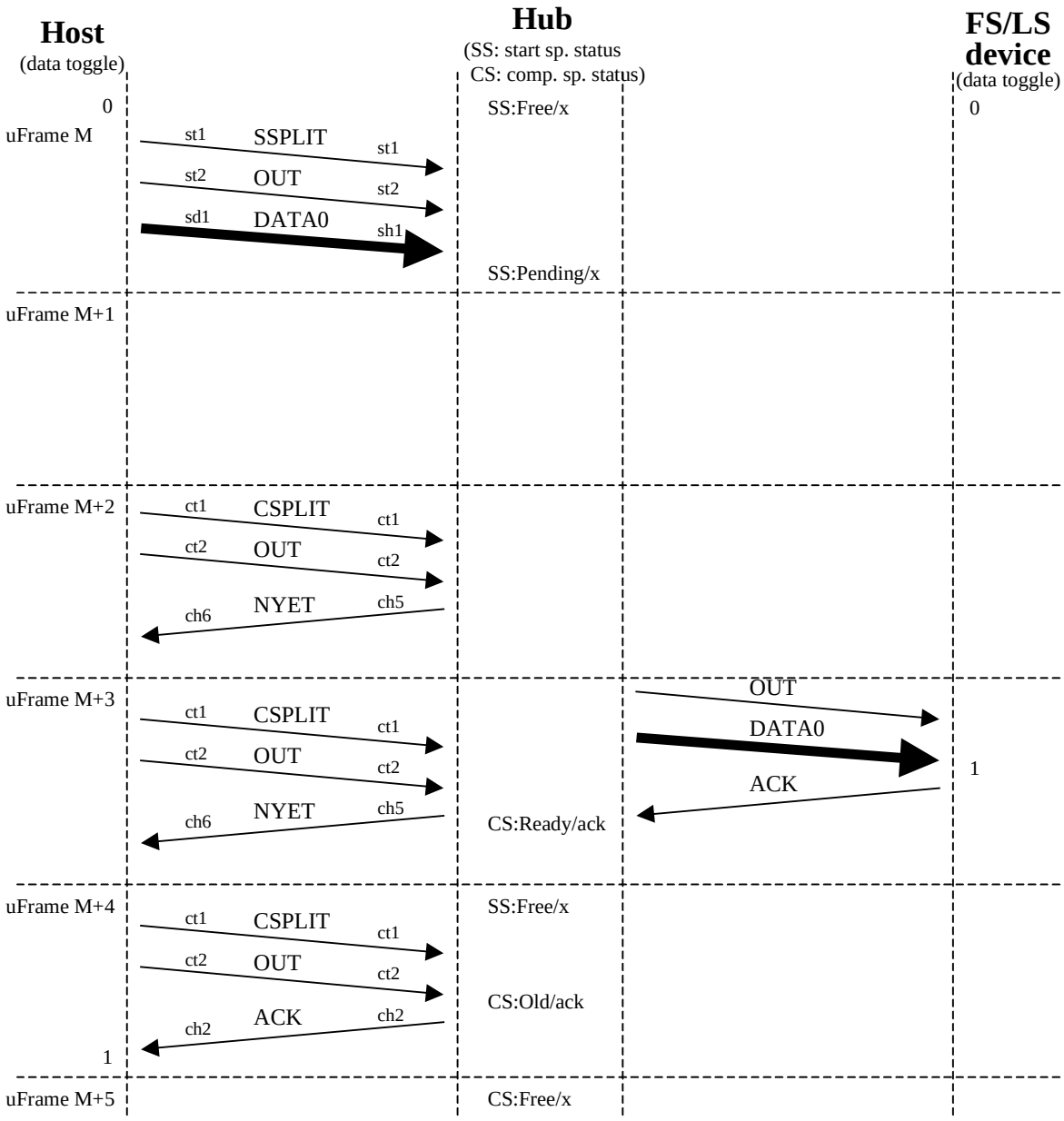


Figure A-57. CS Earlier No Smash(HS NYET and FS/LS Handshake Packet is Done by M+3)

Microchip Ex. 1018, p. 530
Microchip v. Aptiv
IPR2024-00494

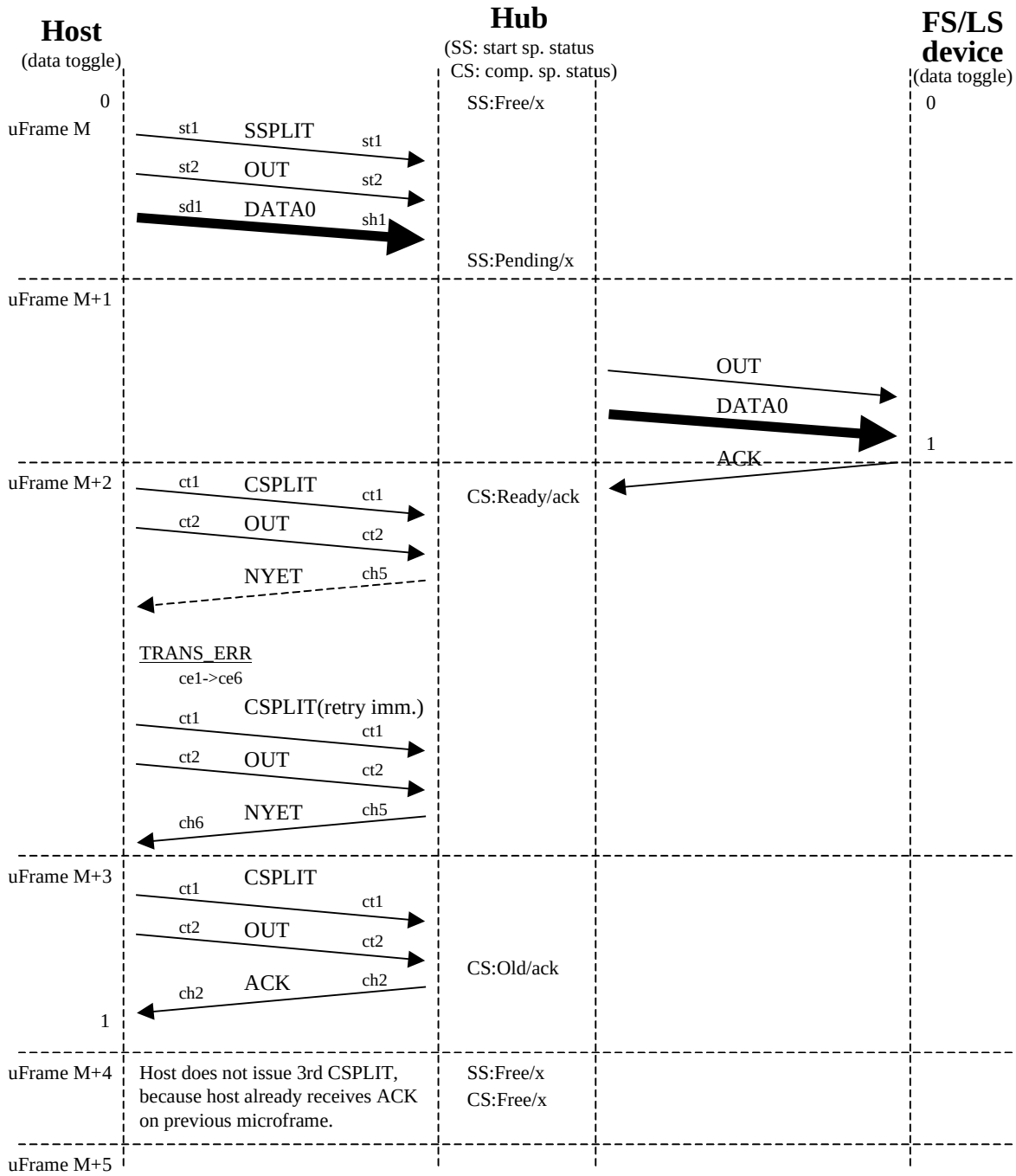


Figure A-58. CS Earlier HS NYET Smash

Microchip Ex. 1018, p. 531
Microchip v. Aptiv
IPR2024-00494

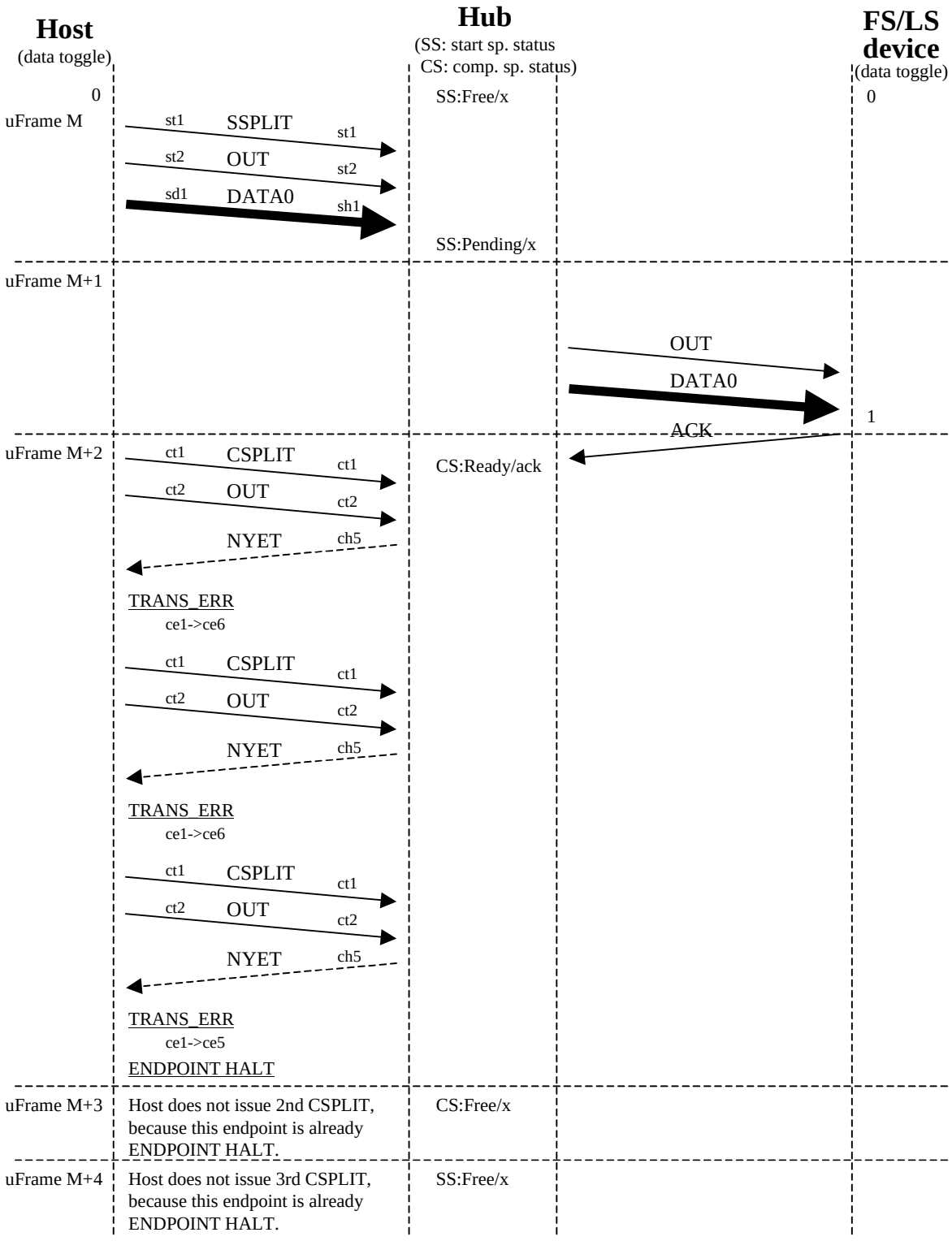


Figure A-59. CS Earlier HS NYET 3 Strikes Smash

Microchip Ex. 1018, p. 532
Microchip v. Aptiv
IPR2024-00494

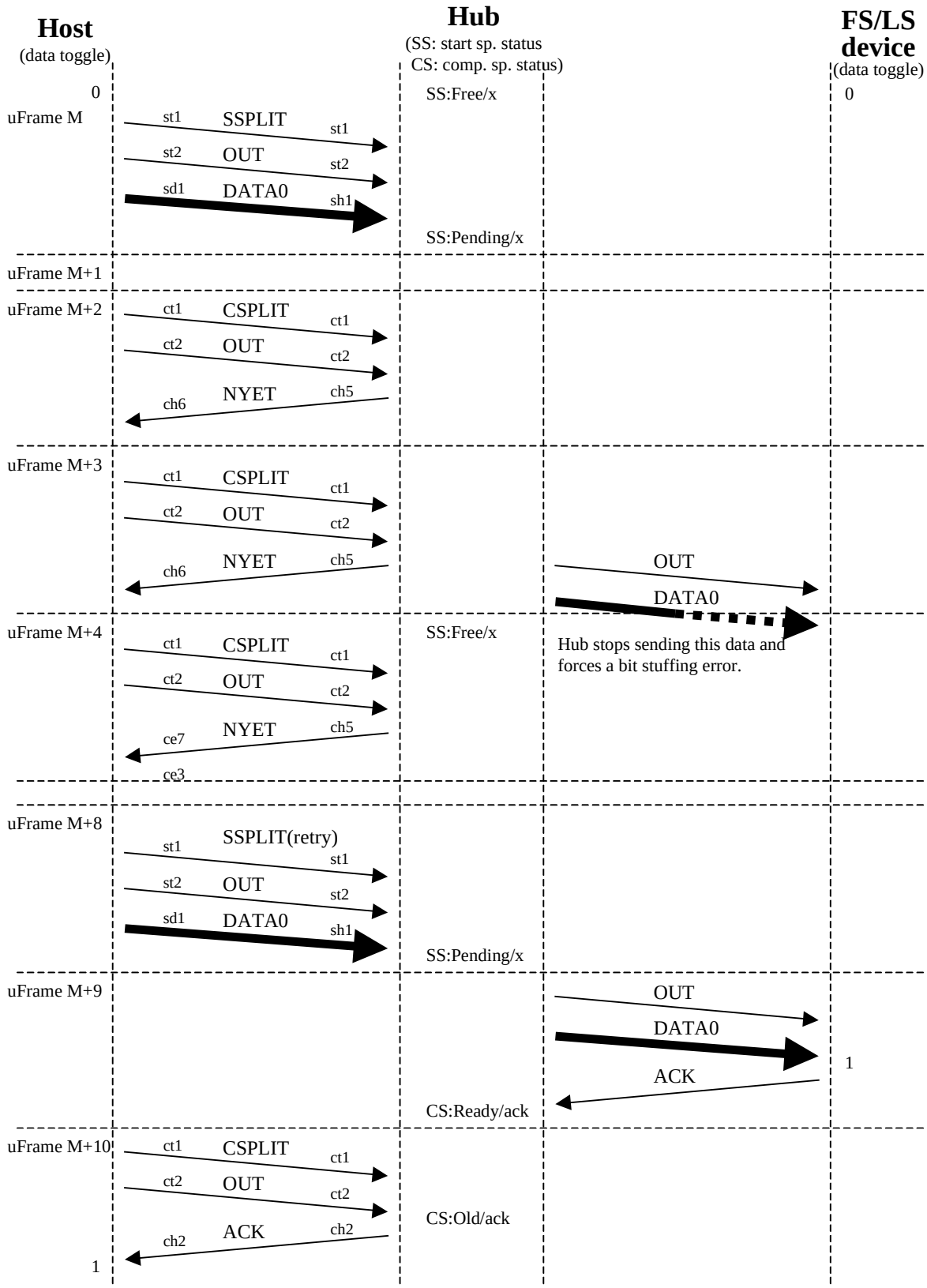


Figure A-60. Abort and Free Abort(FS/LS Transaction is Continued at End of M+3)

Microchip Ex. 1018, p. 533
Microchip v. Aptiv
IPR2024-00494

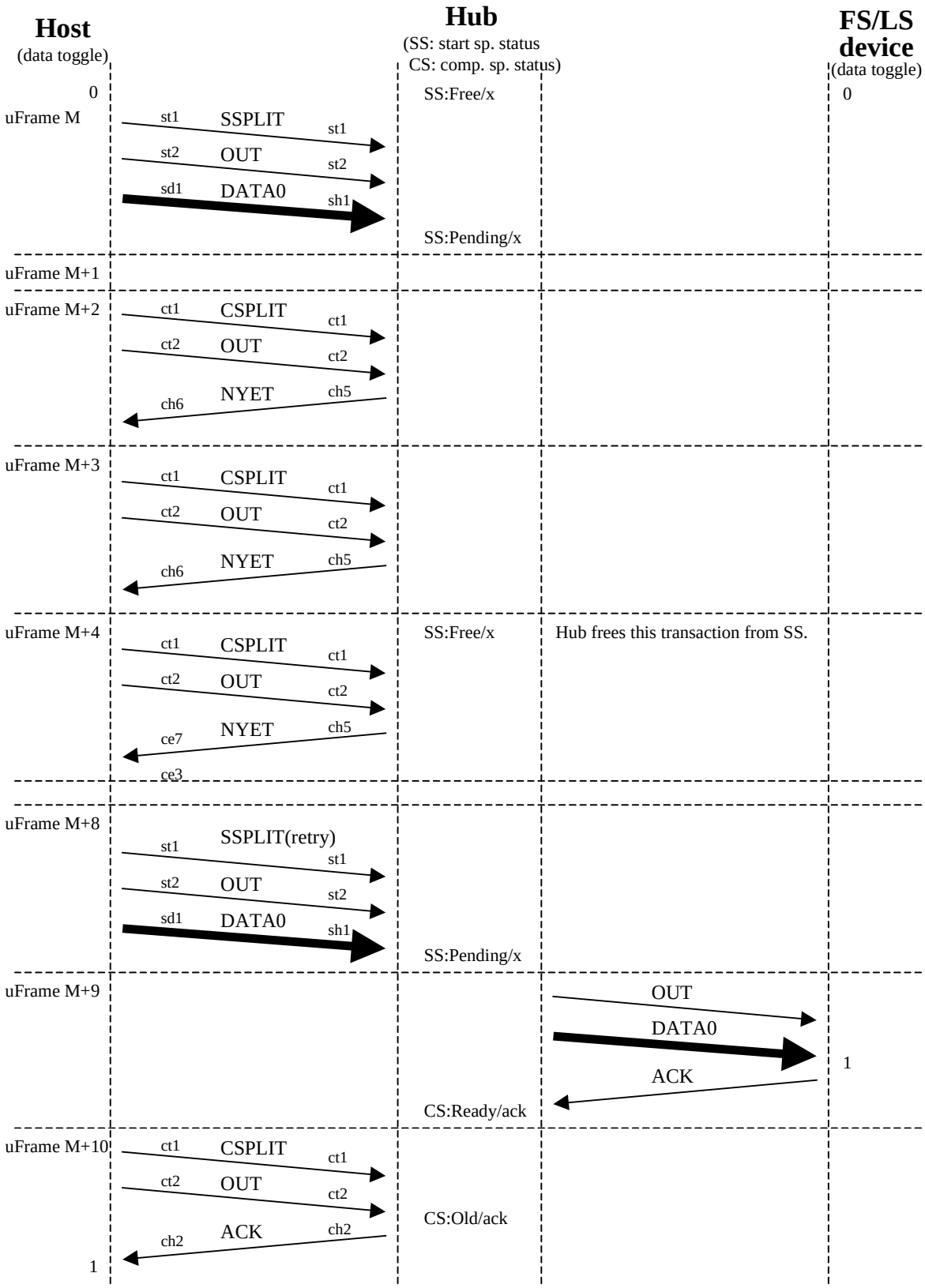


Figure A-61. Abort and Free Free(FS/LS Transaction is not Started at End of M+3)

Microchip Ex. 1018, p. 534
Microchip v. Aptiv
IPR2024-00494

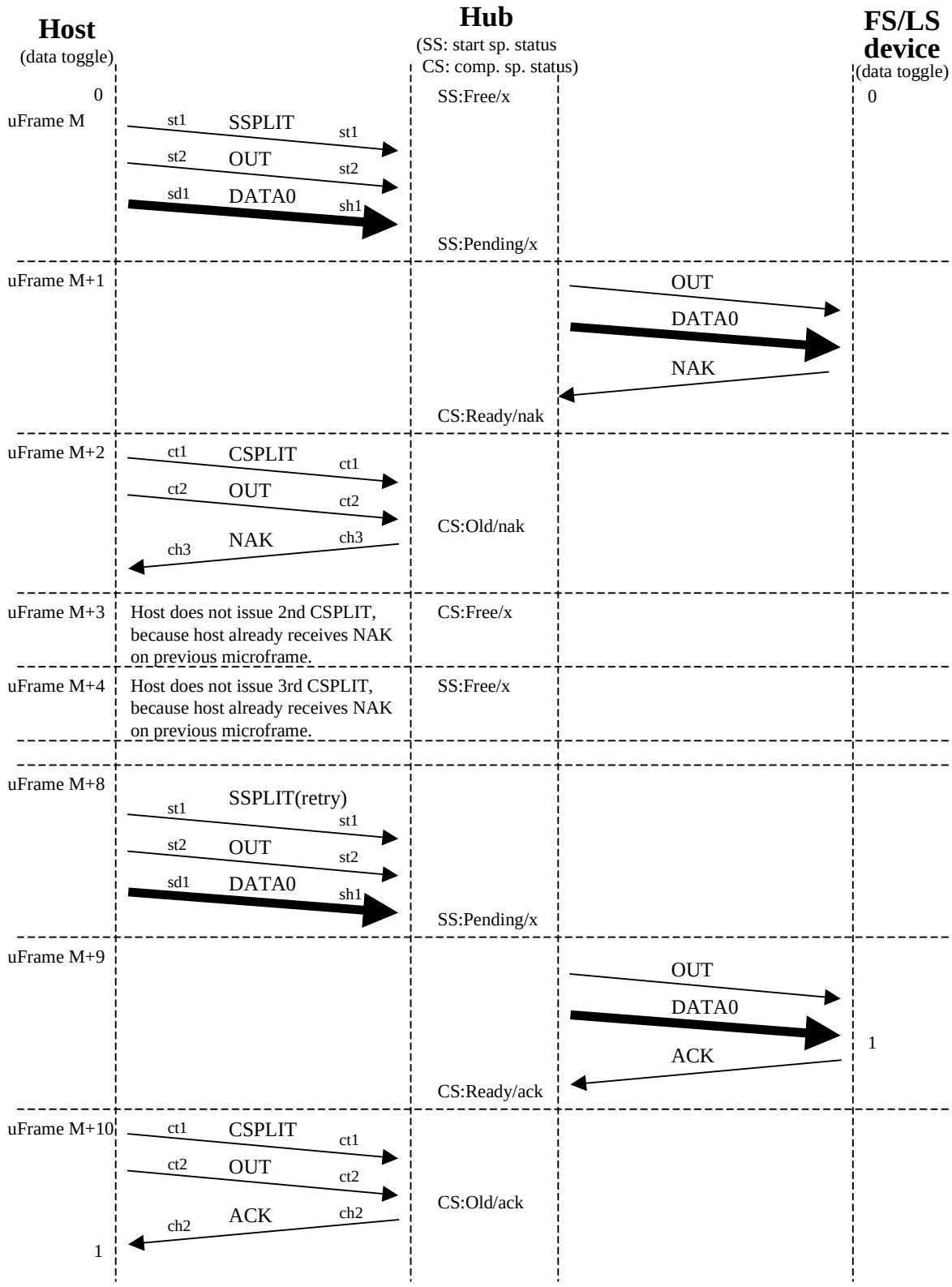


Figure A-62. Device Busy No Smash(FS/LS NAK)

Microchip Ex. 1018, p. 535
Microchip v. Aptiv
IPR2024-00494

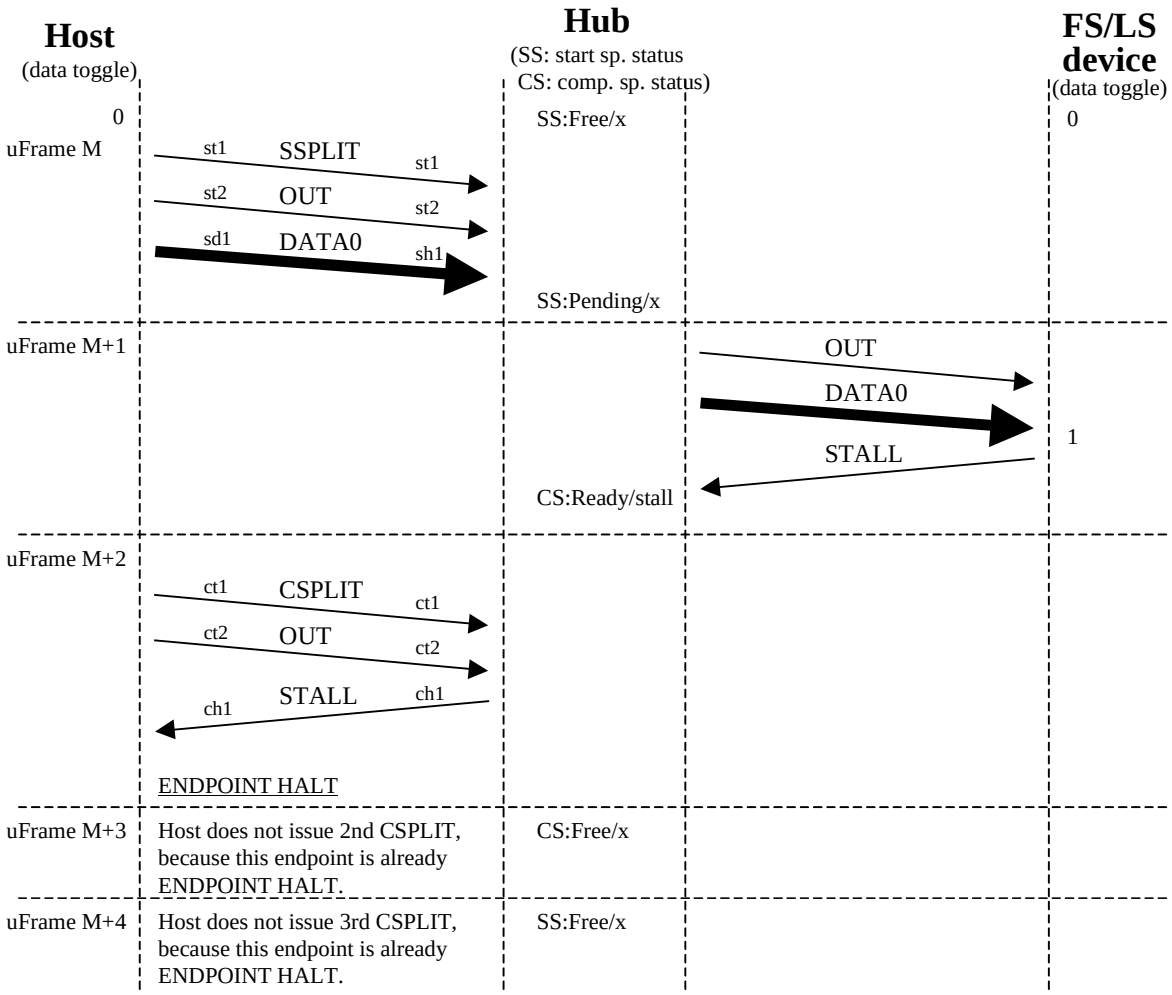


Figure A-63. Device Stall No Smash(FS/LS STALL)

Microchip Ex. 1018, p. 536
Microchip v. Aptiv
IPR2024-00494

A.4 Interrupt IN Transaction Examples

Legend:

(S): Start Split

(C): Complete Split

Summary of cases for Interrupt OUT transaction

- Normal cases

Case	Reference Figure	Similar Figure
No smash (FS/LS data packet is on M+1)	Figure A-64	
HS SSPLIT smash	Figure A-65	
HS SSPLIT 3 strikes smash	No figure	
HS IN(S) smash		Figure A-65
HS IN(S) 3 strikes smash	No figure	
HS CSPLIT smash	Figure A-66	
HS CSPLIT 3 strikes smash	Figure A-67	
HS IN(C) smash		Figure A-66
HS IN(C) 3 strikes smash		Figure A-67
HS DATA0/1 smash	Figure A-68	
HS DATA0/1 3 strikes smash	Figure A-69	
FS/LS IN smash	Figure A-70	
FS/LS IN 3 strikes smash	No figure	
FS/LS DATA0/1 smash	Figure A-71	
FS/LS DATA0/1 3 strikes smash	No figure	
FS/LS ACK smash	Figure A-72	
FS/LS ACK 3 strikes smash	No figure	

Microchip Ex. 1018, p. 537
Microchip v. Aptiv
IPR2024-00494

- Searcing

Case	Reference Figure	Similar Figure
No smash	Figure A-73	

- CS(Complete-split transaction) earlier cases

Case	Reference Figure	Similar Figure
No smash (HS MDATA and FS/LS transaction is on M+1 and M+2)	Figure A-74	
No smash (HS NYET and FS/LS transaction is on M+2)	Figure A-75	
No smash (HS NYET and MDATA and FS/LS transaction is on M+2 and M+3)	Figure A-76	
No smash (HS NYET and FS/LS transaction is on M+3)	Figure A-77	
HS NYET smash	Figure A-78	
HS NYET 3 strikes smash	Figure A-79	

- Abort and Free cases

Case	Reference Figure	Similar Figure
No smash and abort (HS NYETand FS/LS transaction is continued at end of M+3)	Figure A-80	
No smash and free(HS NYETand FS/LS transaction is not started at end of M+3)	Figure A-81	

- FS/LS transaction error cases

Case	Reference Figure	Similar Figure
HS ERR smash		Figure A-68
HS ERR 3 strikes smash		Figure A-69

Microchip Ex. 1018, p. 538
Microchip v. Aptiv
IPR2024-00494

- Device busy cases

Case	Reference Figure	Similar Figure
No smash(HS NAK(C))	Figure A-82	
HS NAK(C) smash		Figure A-68
HS NAK(C) 3 strikes smash		Figure A-69
FS/LS NAK smash		Figure A-71
FS/LS NAK 3 strikes smash	No figure	

- Device stall cases

Case	Reference Figure	Similar Figure
No smash	Figure A-83	
HS STALL(C) smash		Figure A-68
HS STALL(C) 3 strikes smash		Figure A-69
FS/LS STALL smash		Figure A-71
FS/LS STALL 3 strikes smash	No figure	

Microchip Ex. 1018, p. 539
Microchip v. Aptiv
IPR2024-00494

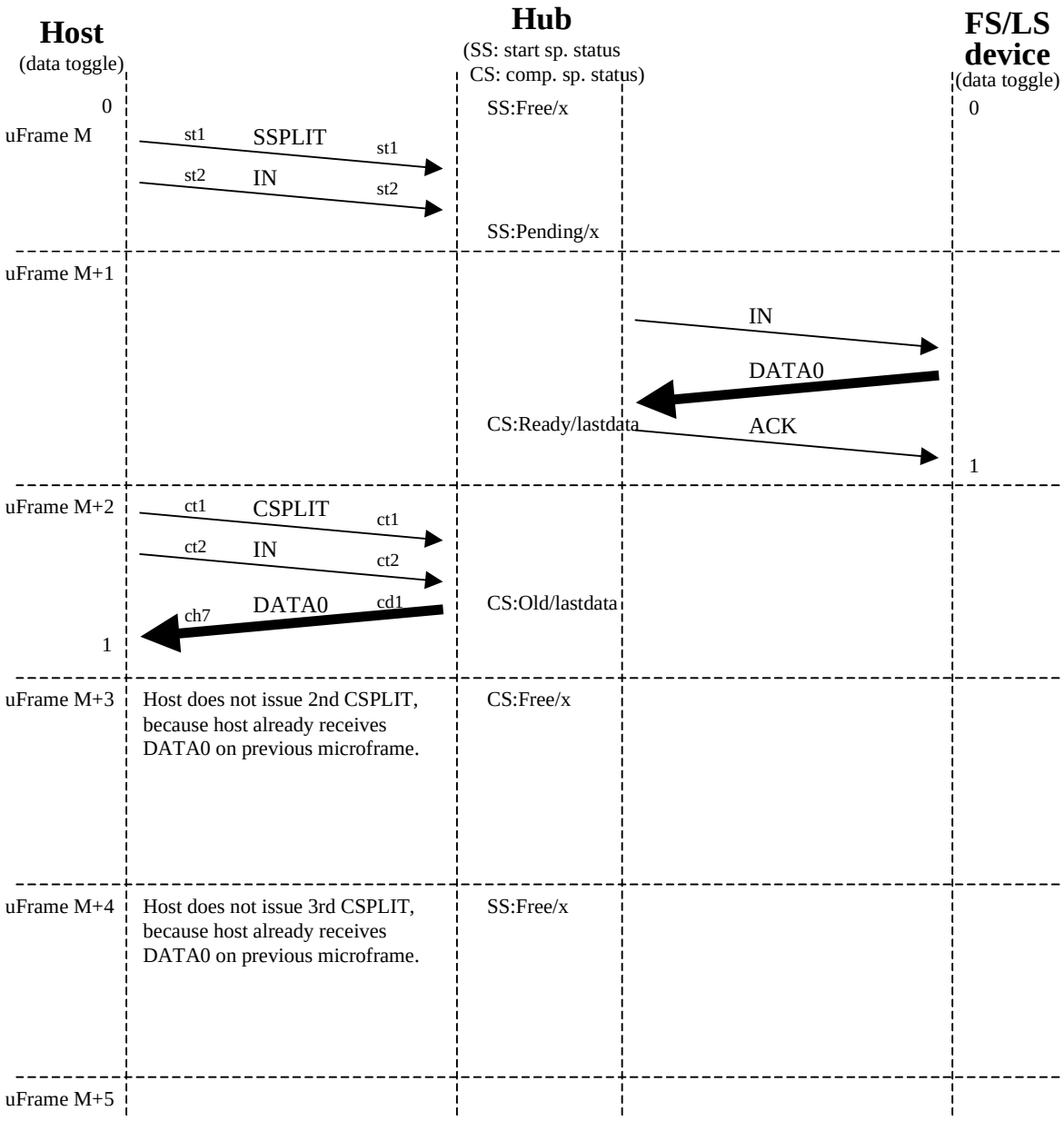


Figure A-64. Normal No Smash(FS/LS Data Packet is on M+1)

Microchip Ex. 1018, p. 540
Microchip v. Aptiv
IPR2024-00494

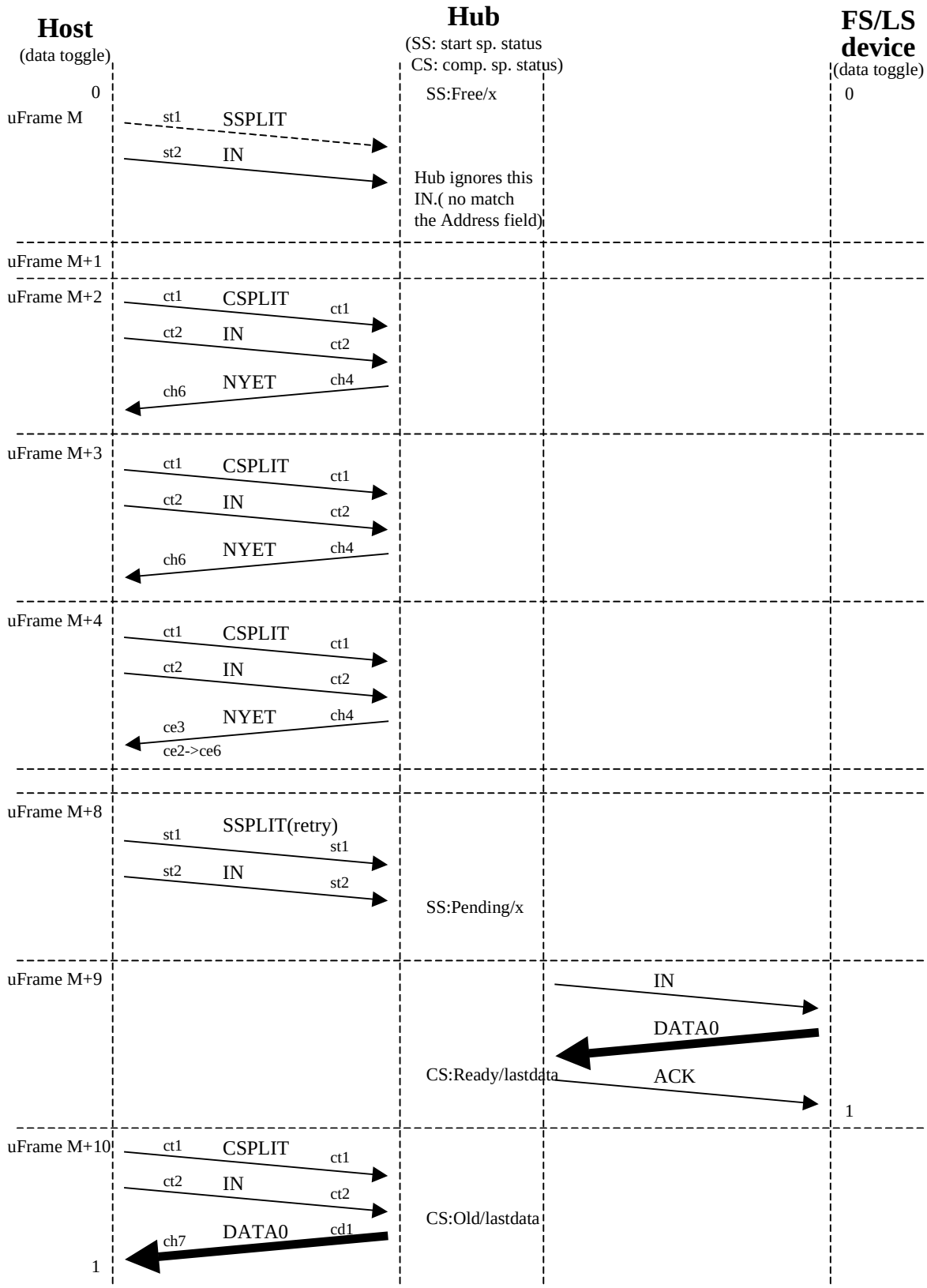


Figure A-65. Normal HS SSPLIT Smash

Microchip Ex. 1018, p. 541
Microchip v. Aptiv
IPR2024-00494

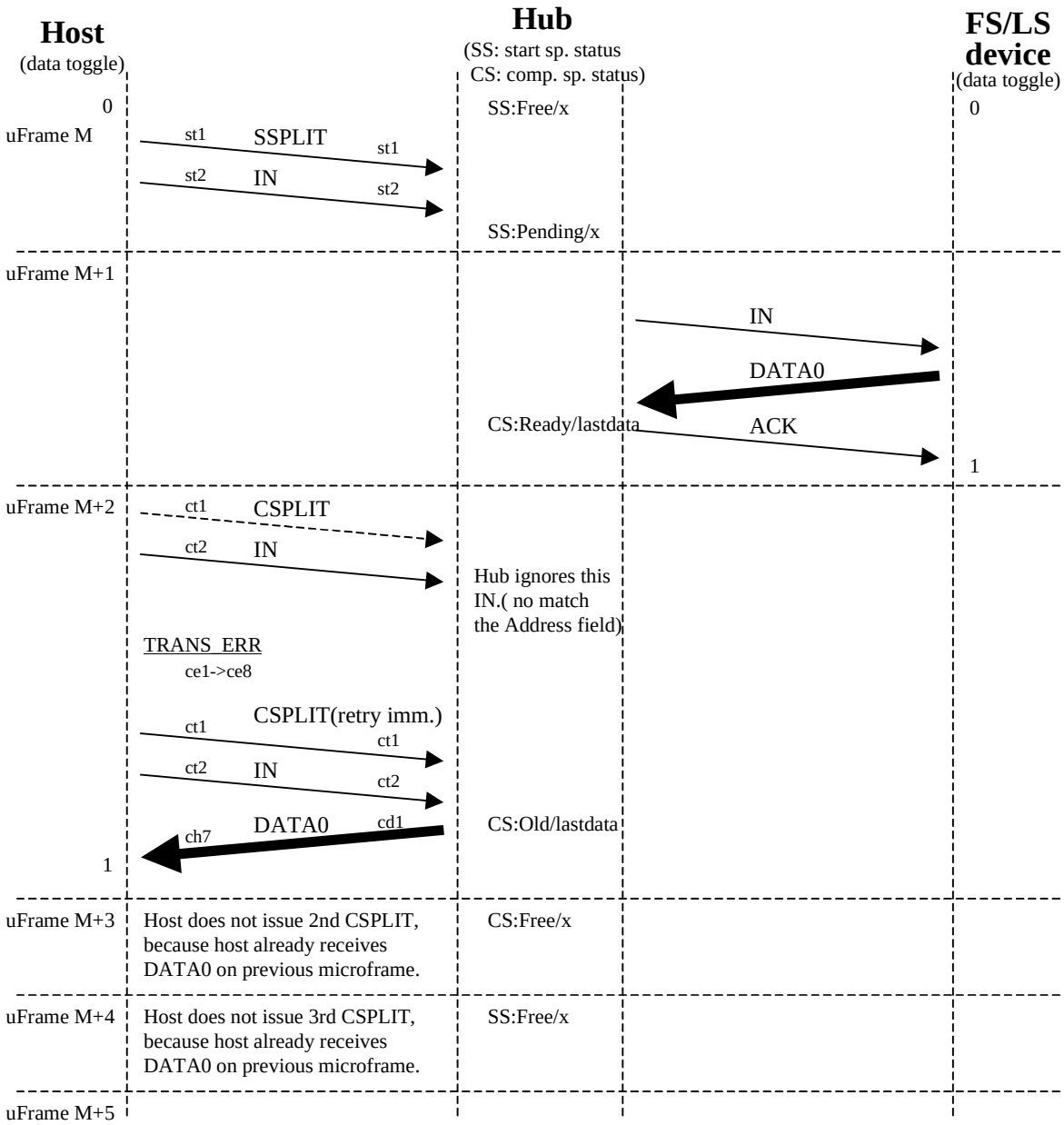


Figure A-66. Normal HS CSPLIT Smash

Microchip Ex. 1018, p. 542
Microchip v. Aptiv
IPR2024-00494

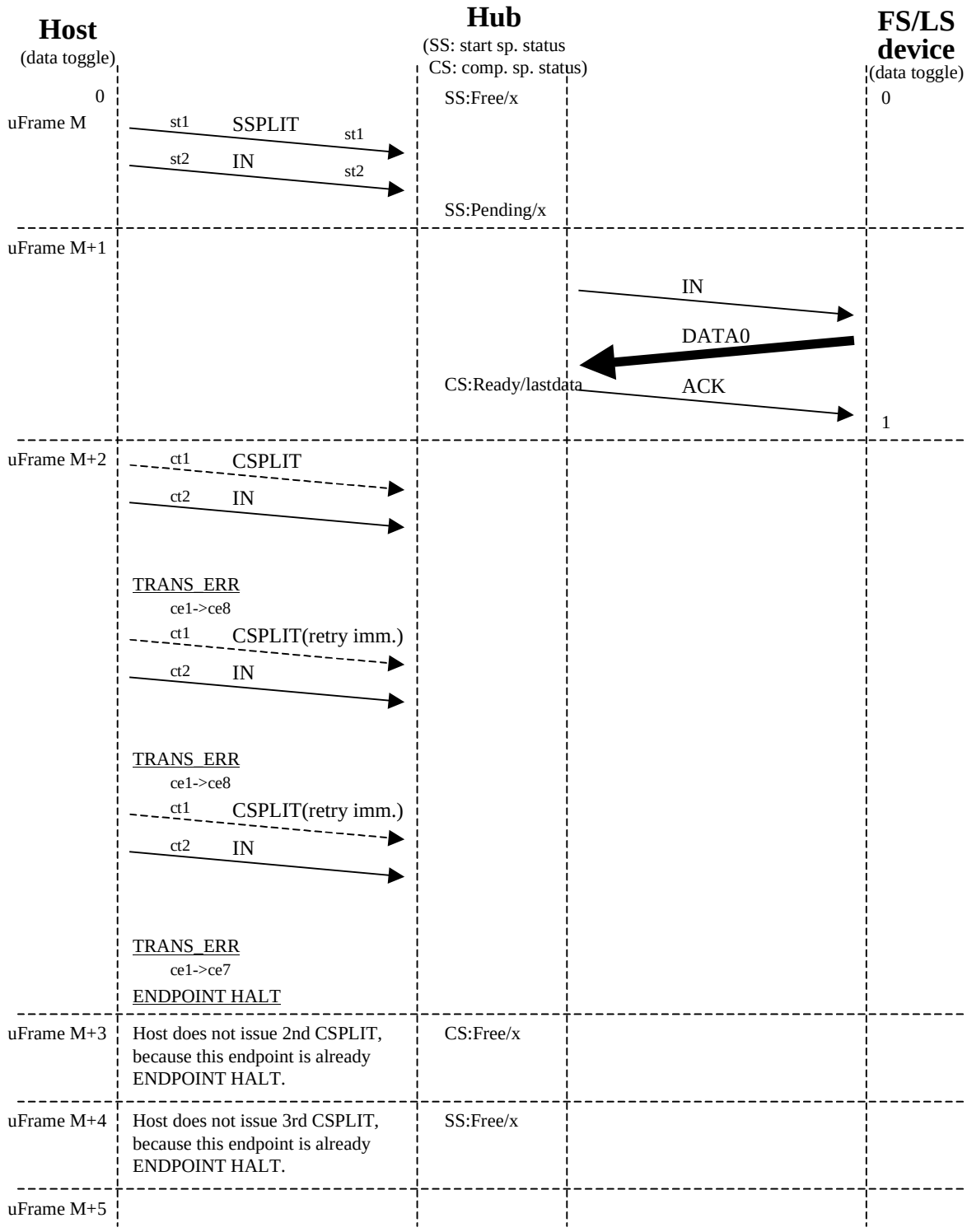


Figure A-67. Normal HS CSPLIT 3 Strikes Smash

Microchip Ex. 1018, p. 543
Microchip v. Aptiv
IPR2024-00494

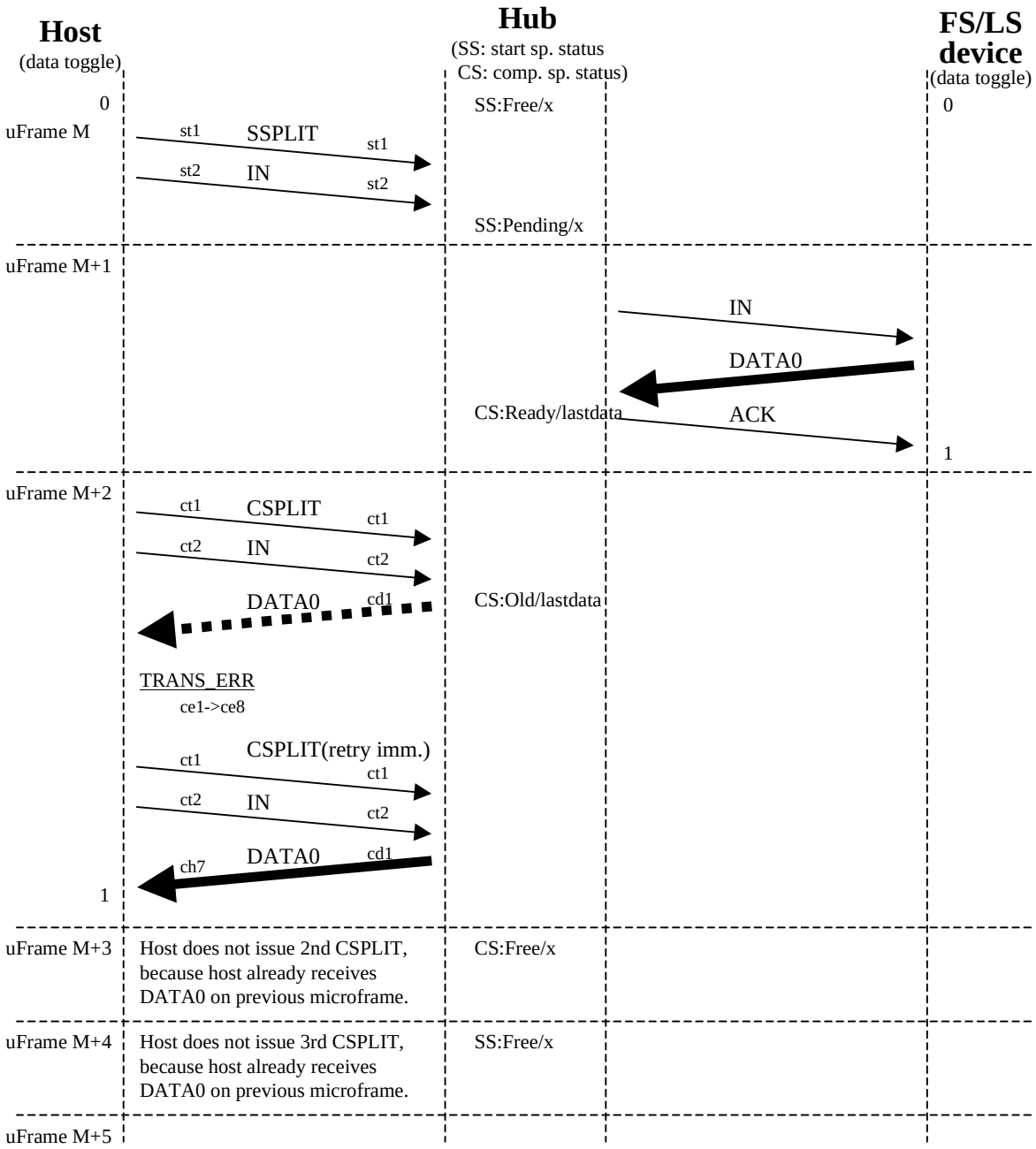


Figure A-68. Normal HS DATA0/1 Smash

Microchip Ex. 1018, p. 544
Microchip v. Aptiv
IPR2024-00494

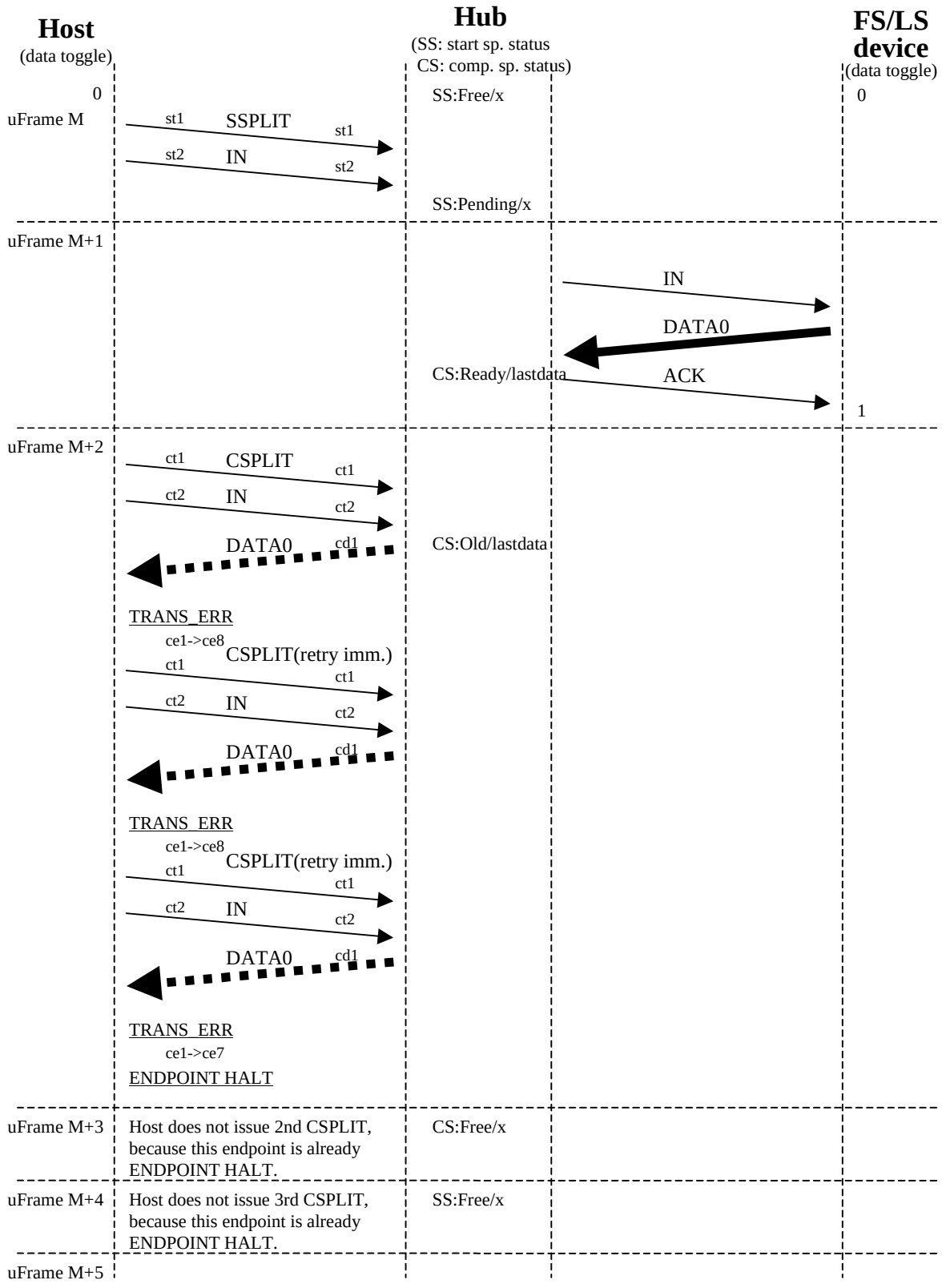


Figure A-69. Normal HS DATA0/1 3 Strikes Smash

Microchip Ex. 1018, p. 545
Microchip v. Aptiv
IPR2024-00494

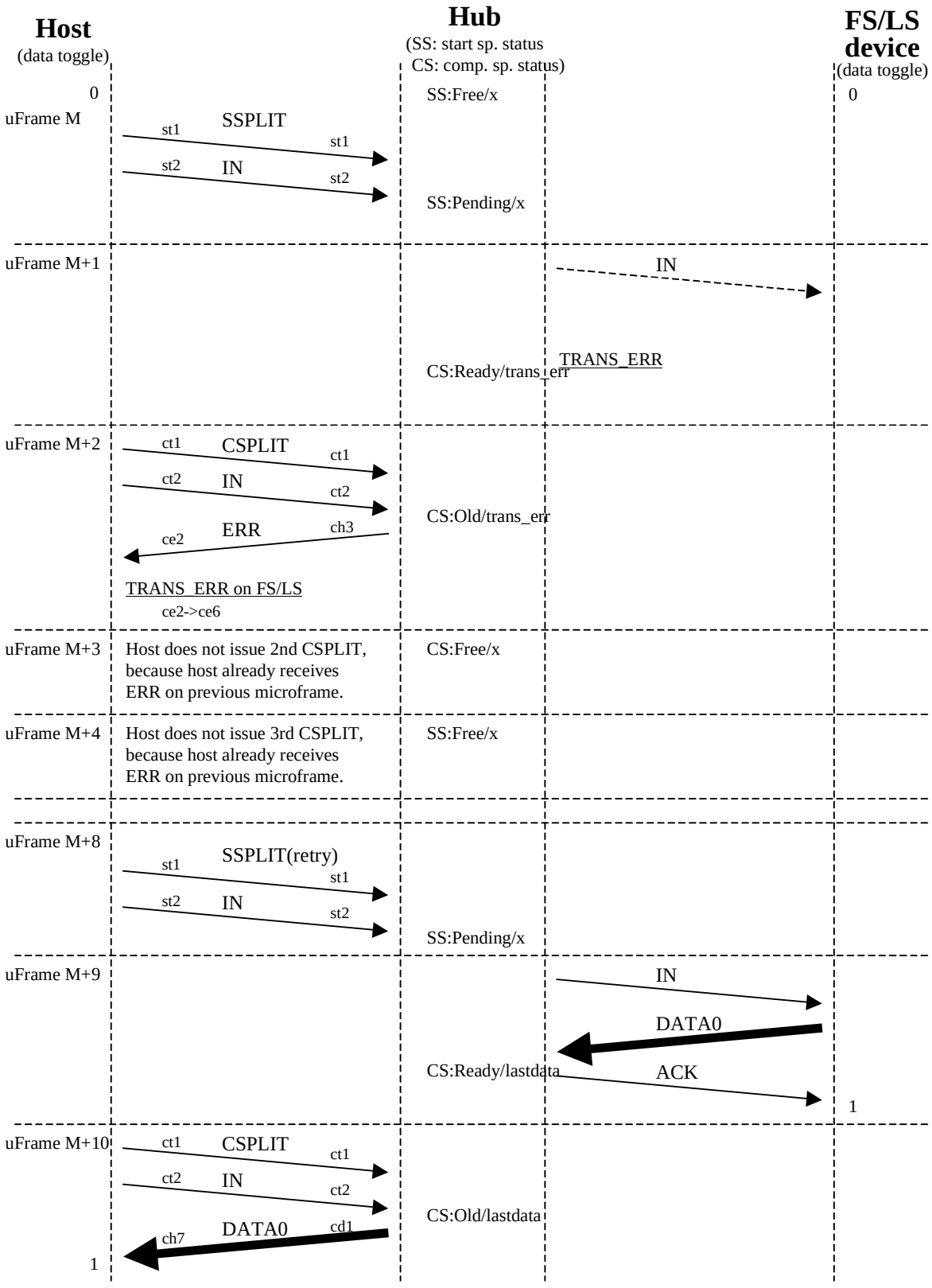


Figure A-70. Normal FS/LS IN Smash

Microchip Ex. 1018, p. 546
Microchip v. Aptiv
IPR2024-00494

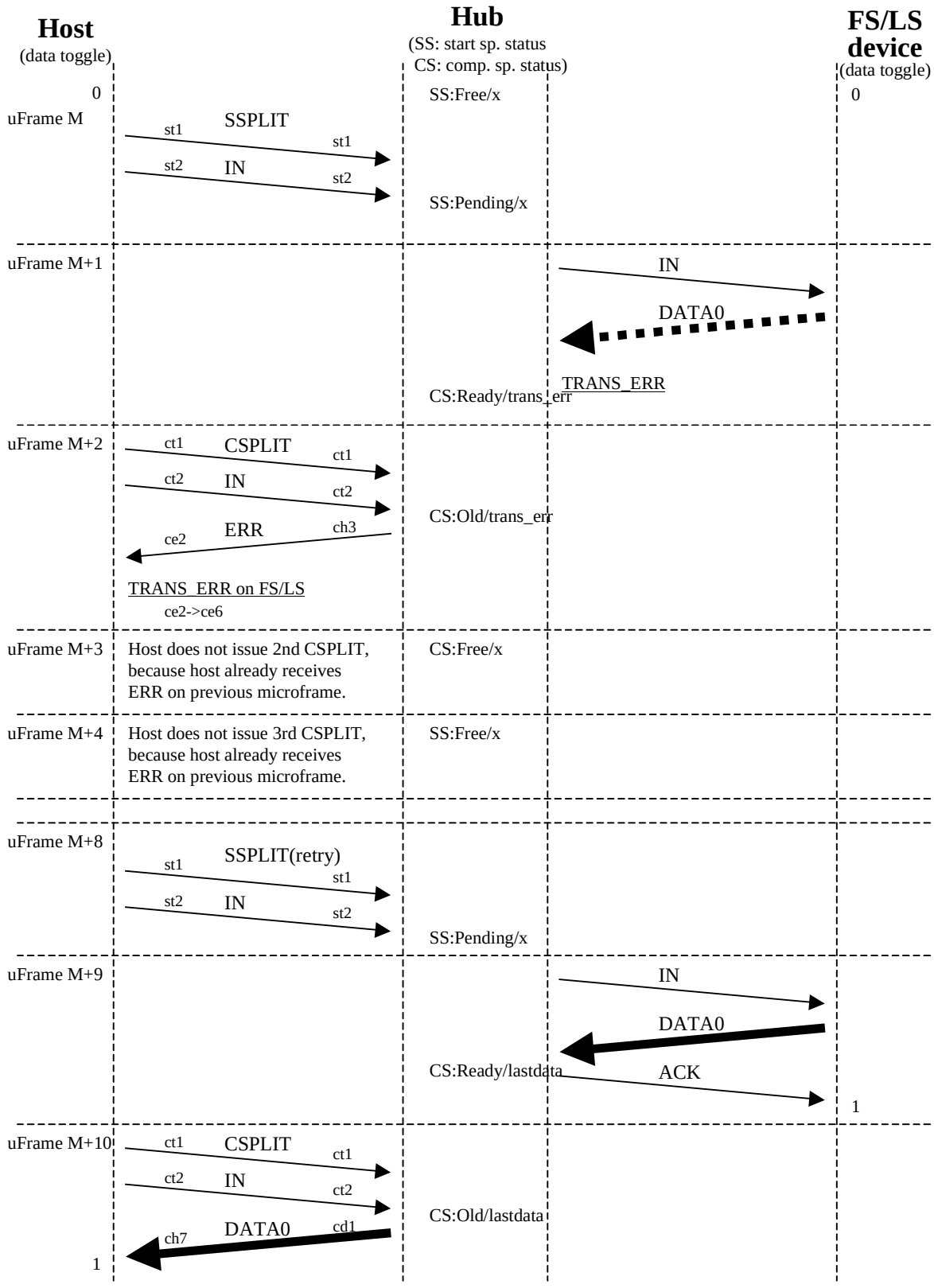


Figure A-71. Normal FS/LS DATA0/1 Smash

Microchip Ex. 1018, p. 547
Microchip v. Aptiv
IPR2024-00494

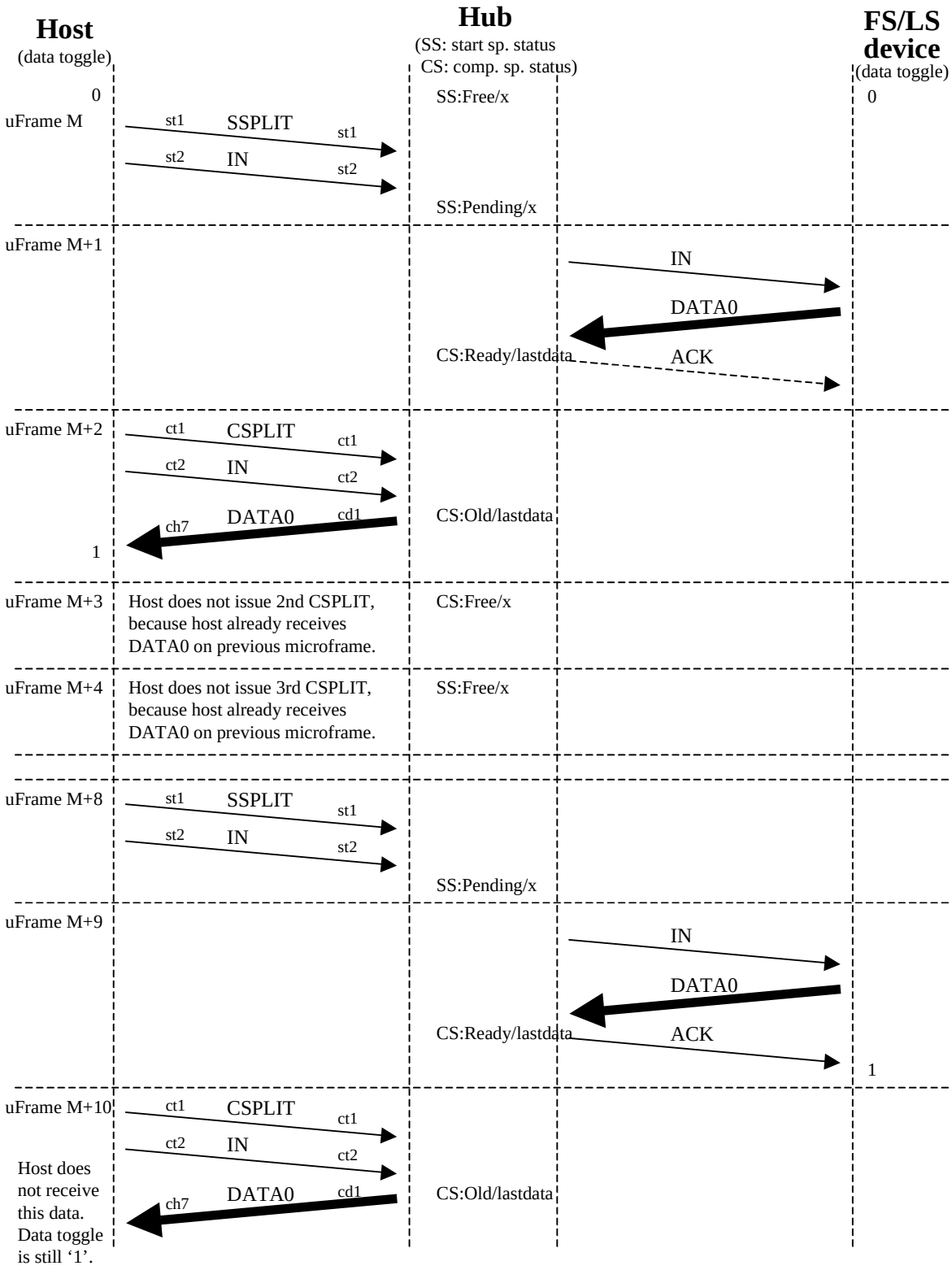


Figure A-72. Normal FS/LS ACK Smash

Microchip Ex. 1018, p. 548
Microchip v. Aptiv
IPR2024-00494

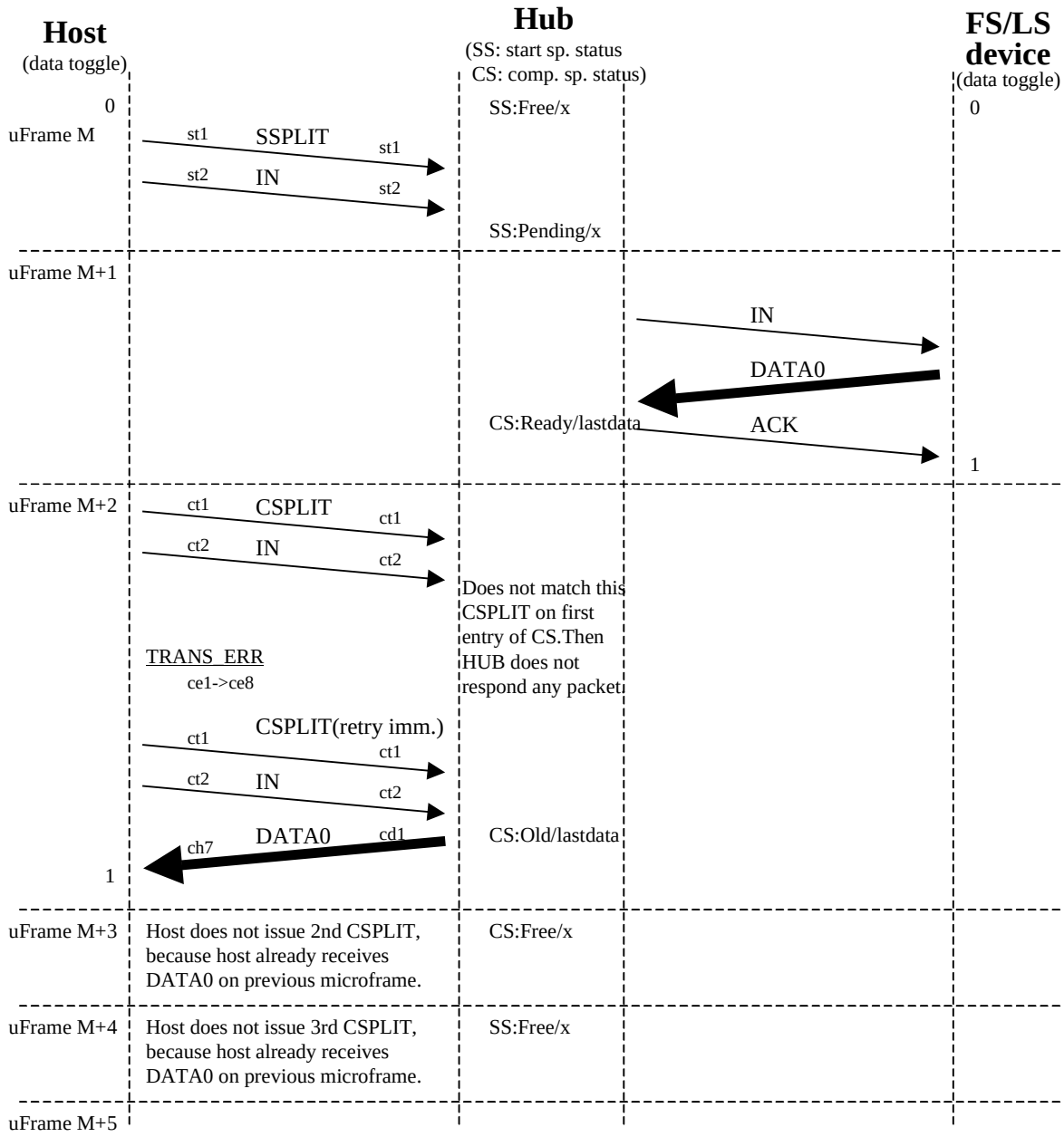


Figure A-73. Searching No Smash

Microchip Ex. 1018, p. 549
Microchip v. Aptiv
IPR2024-00494

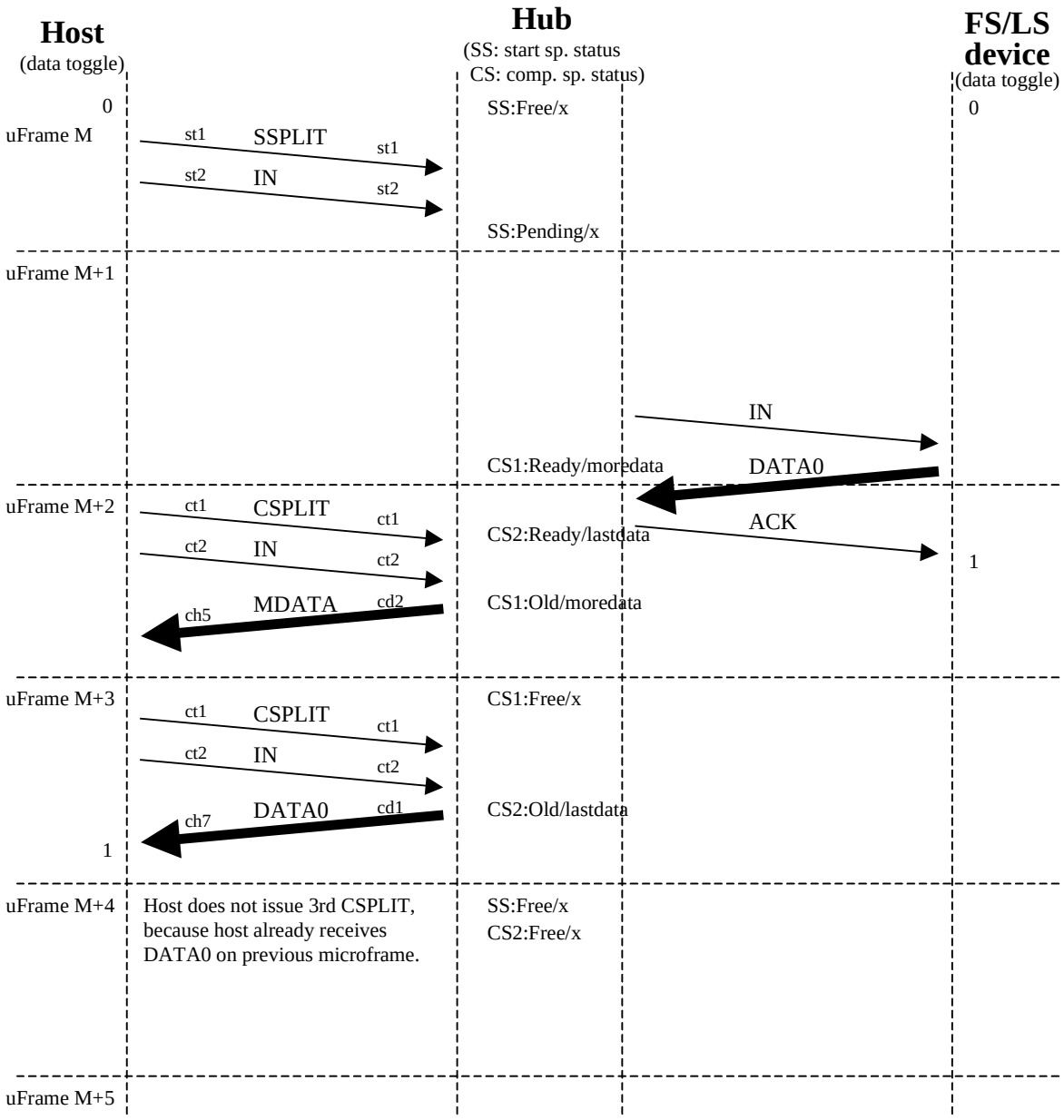


Figure A-74. CS Earlier No Smash(HS MDATA and FS/LS Data Packet is on M+1 and M+2)

Microchip Ex. 1018, p. 550
Microchip v. Aptiv
IPR2024-00494

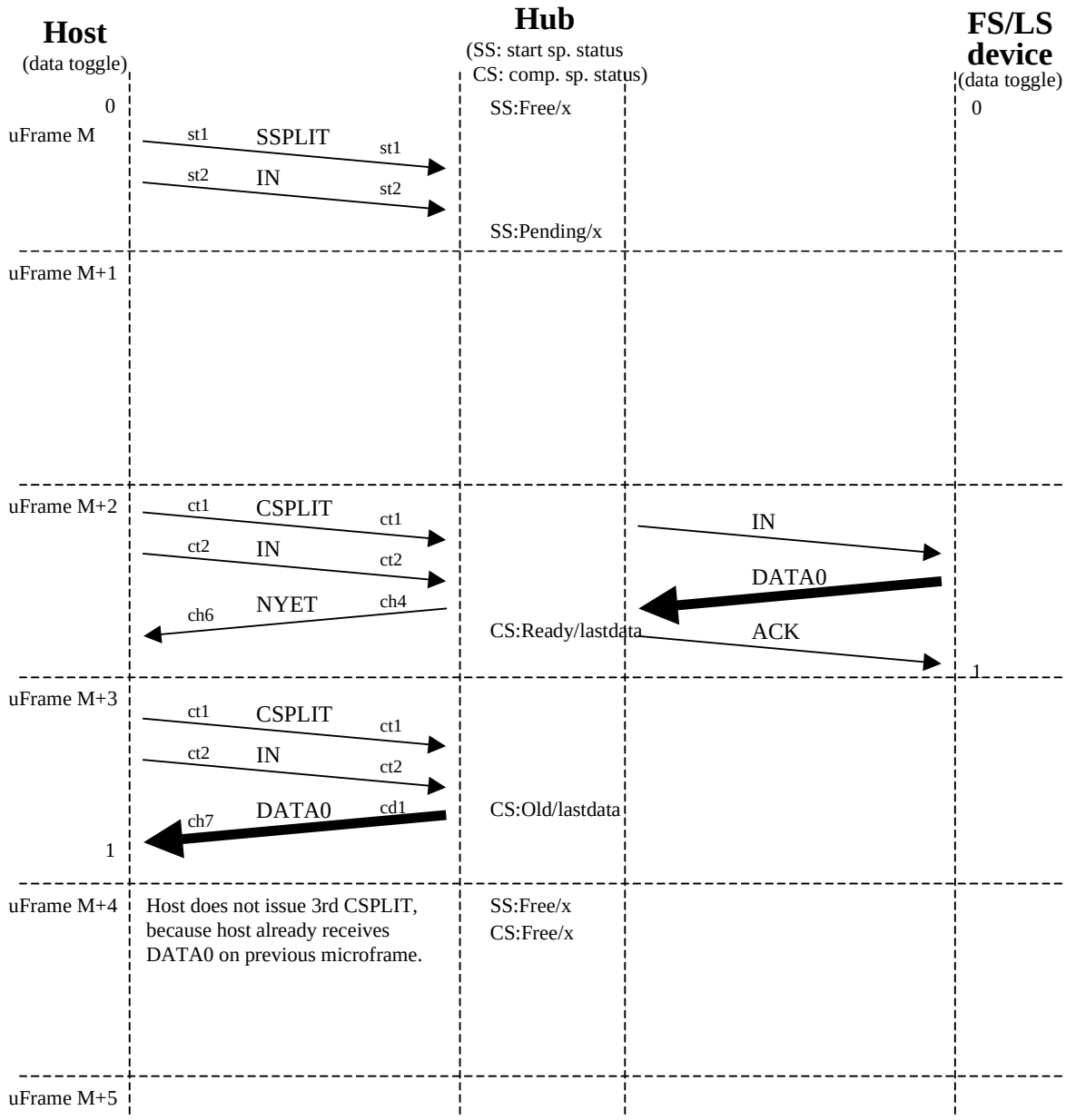


Figure A-75. CS Earlier No Smash(HS NYET and FS/LS Data Packet is on M+2)

Microchip Ex. 1018, p. 551
Microchip v. Aptiv
IPR2024-00494

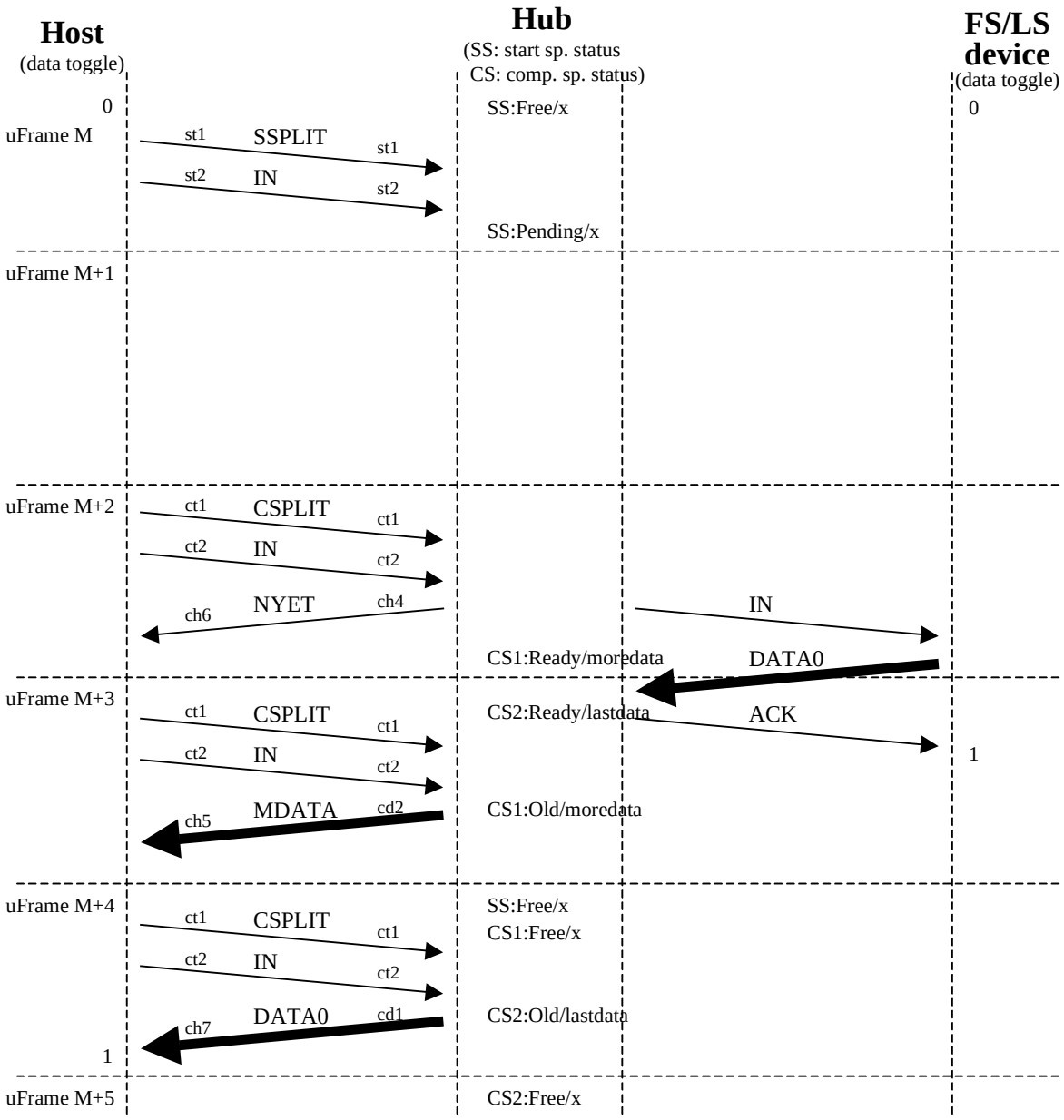


Figure A-76. CS Earlier No Smash(HS NYET and MDATA and FS/LS Data Packet is on M+2 and M+3)

Microchip Ex. 1018, p. 552
Microchip v. Aptiv
IPR2024-00494

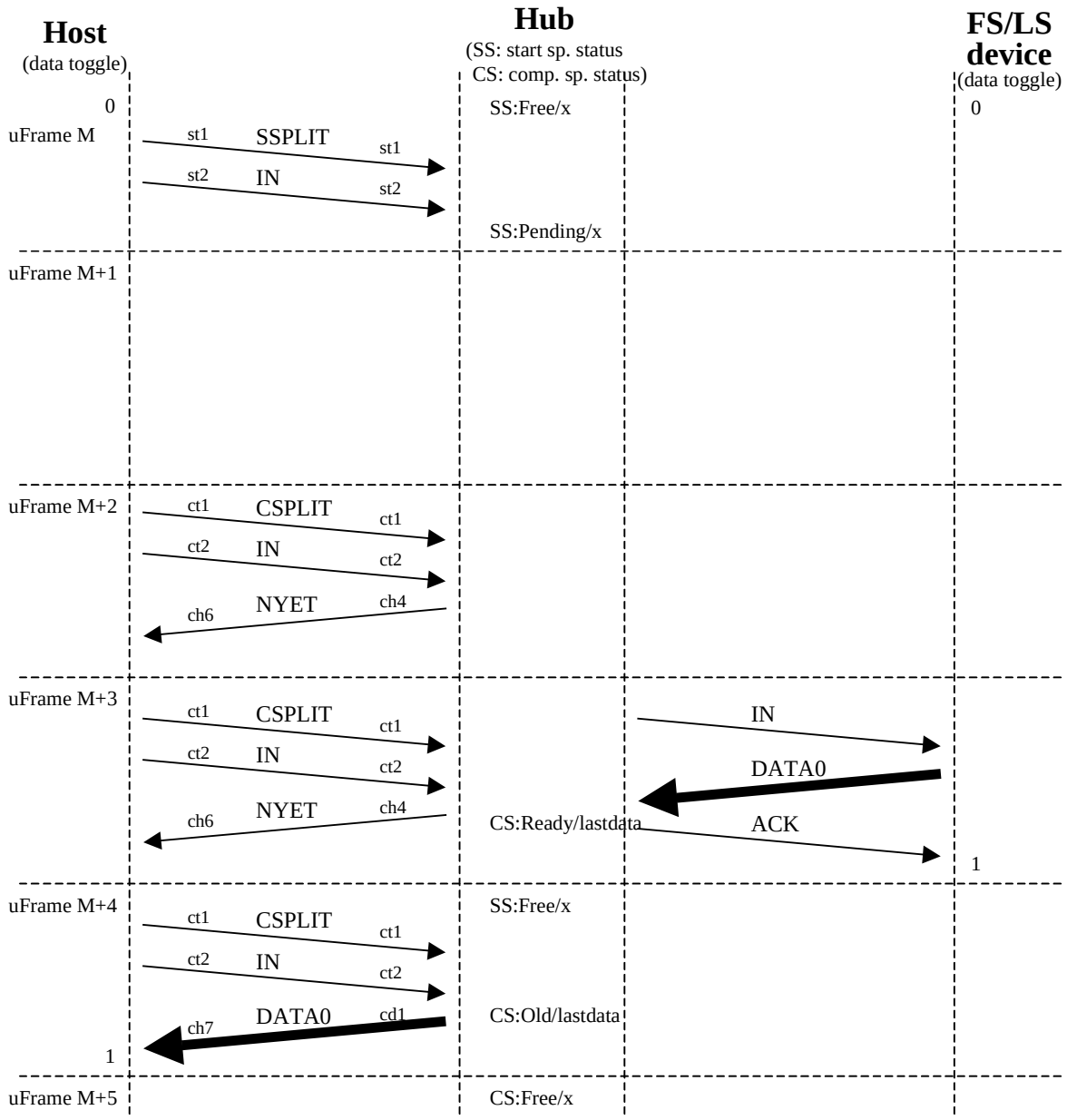


Figure A-77. CS Earlier No Smash(HS NYET and FS/LS Data Packet is on M+3)

Microchip Ex. 1018, p. 553
Microchip v. Aptiv
IPR2024-00494

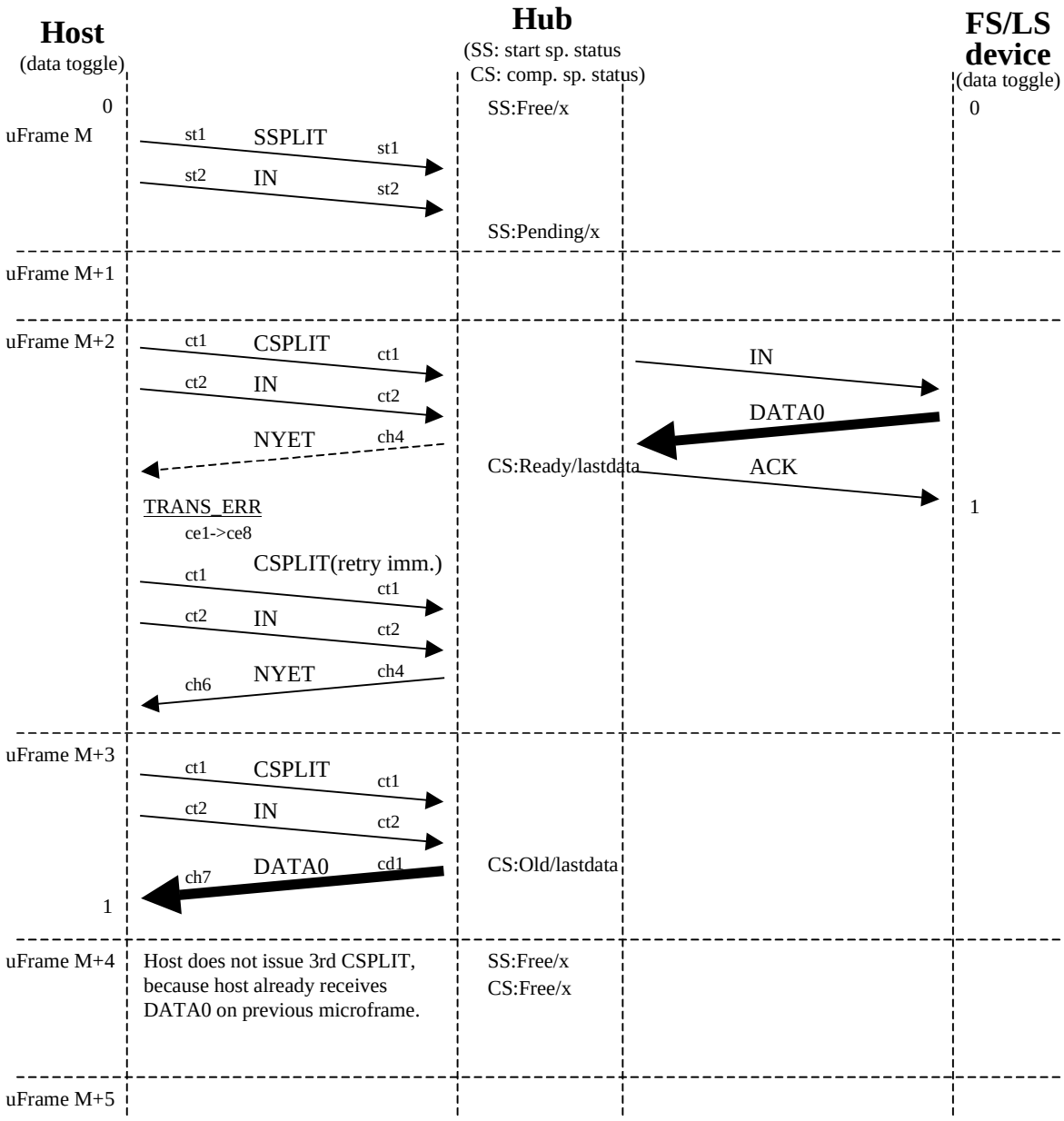


Figure A-78. CS Earlier HS NYET Smash

Microchip Ex. 1018, p. 554
Microchip v. Aptiv
IPR2024-00494

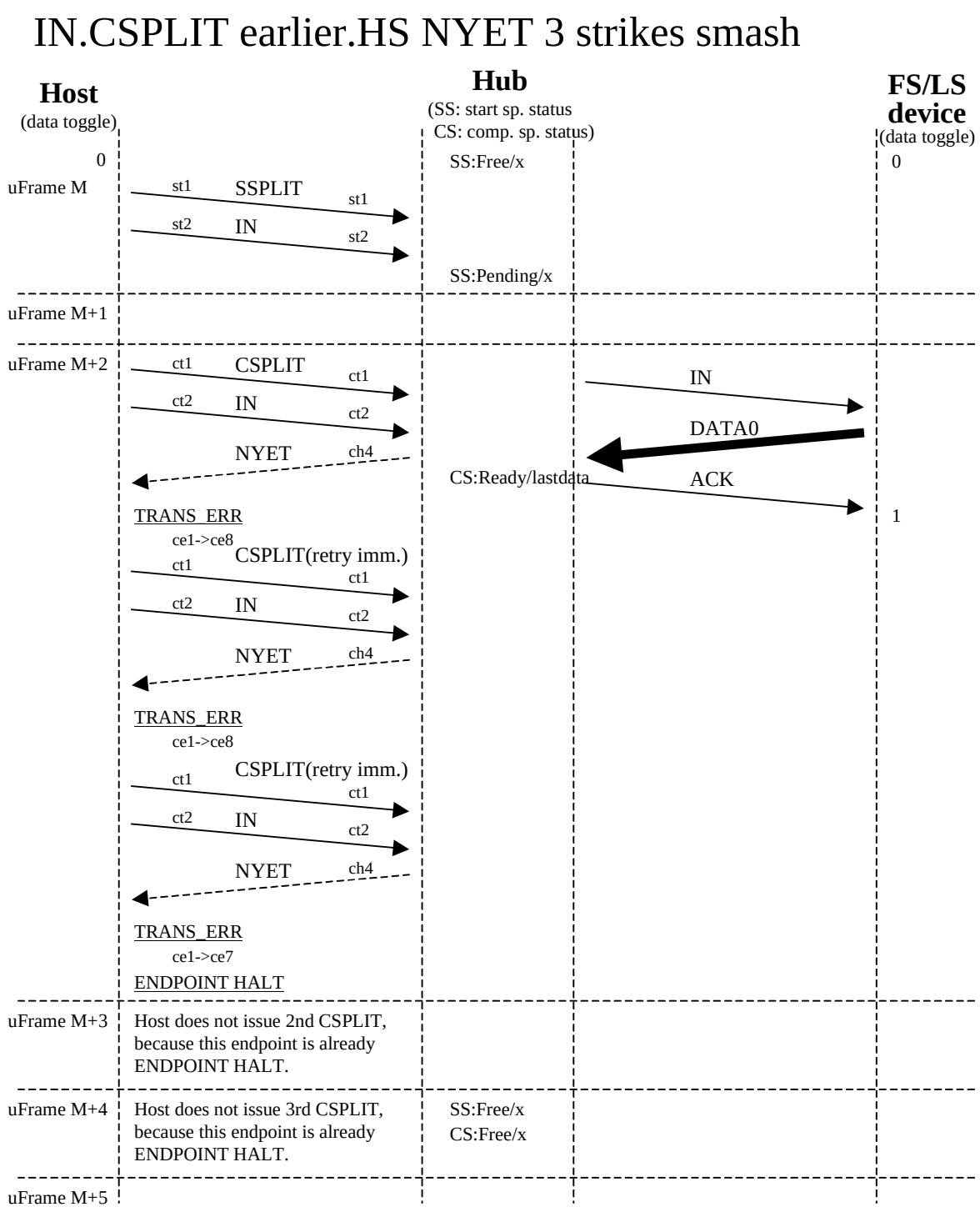


Figure A-79. CS Earlier HS NYET 3 Strikes Smash

Microchip Ex. 1018, p. 555
Microchip v. Aptiv
IPR2024-00494

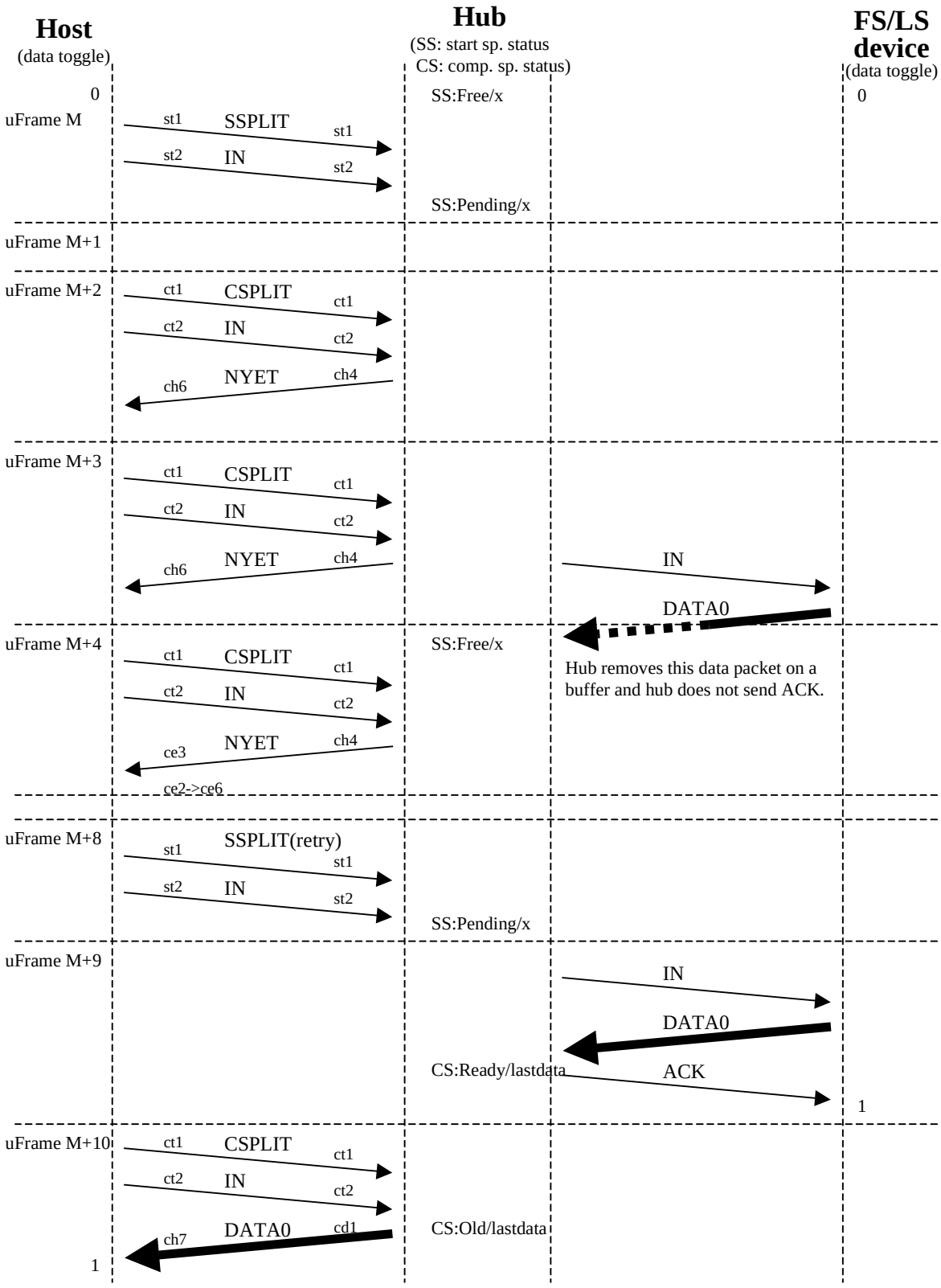


Figure A-80. Abort and Free Abort(HS NYET and FS/LS Transaction is Continued at End of M+3)

Microchip Ex. 1018, p. 556
Microchip v. Aptiv
IPR2024-00494

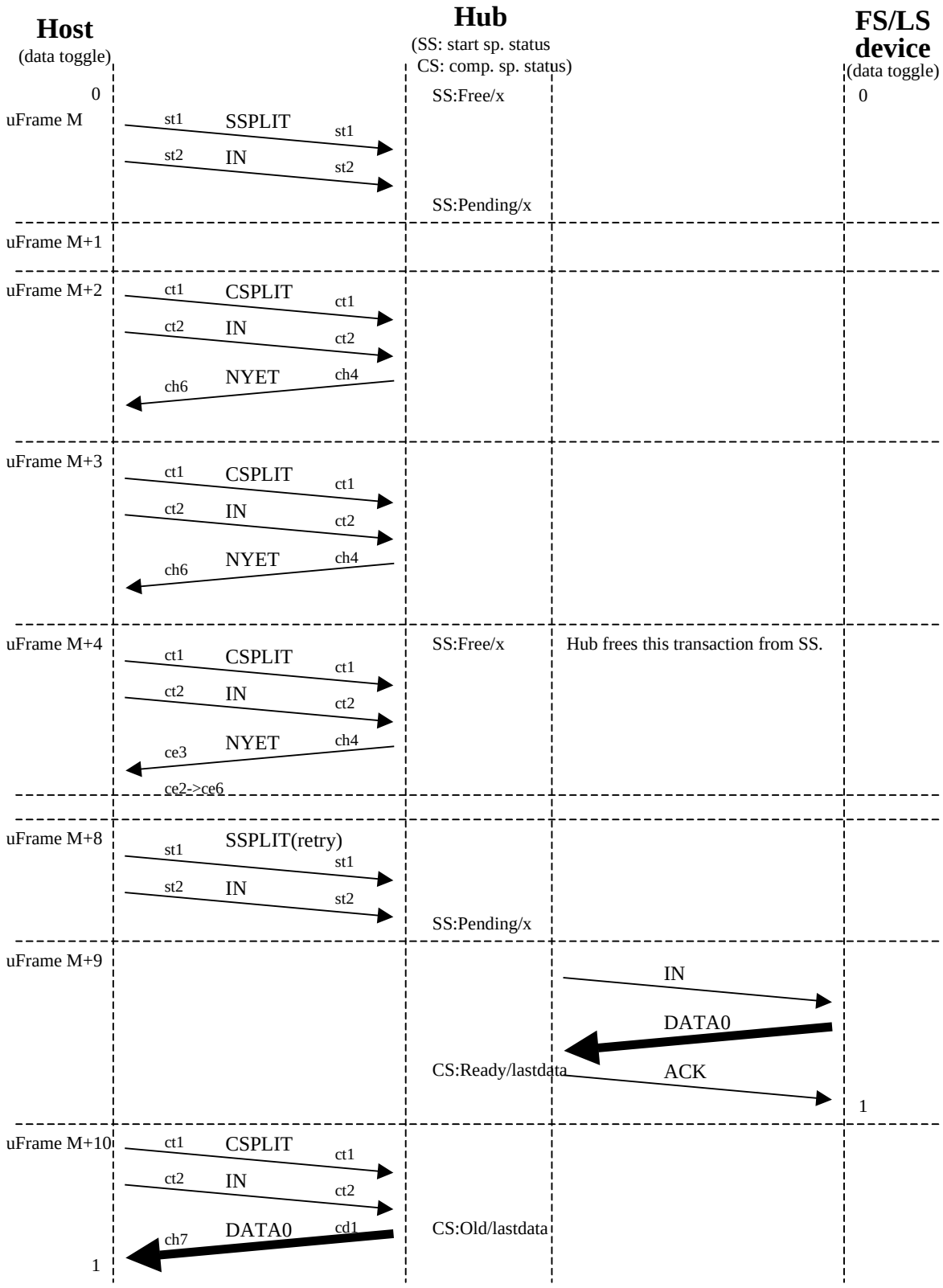


Figure A-81. Abort and Free Free(HS NYET and FS/LS Transaction is not Started at End of M+3)

Microchip Ex. 1018, p. 557
Microchip v. Aptiv
IPR2024-00494

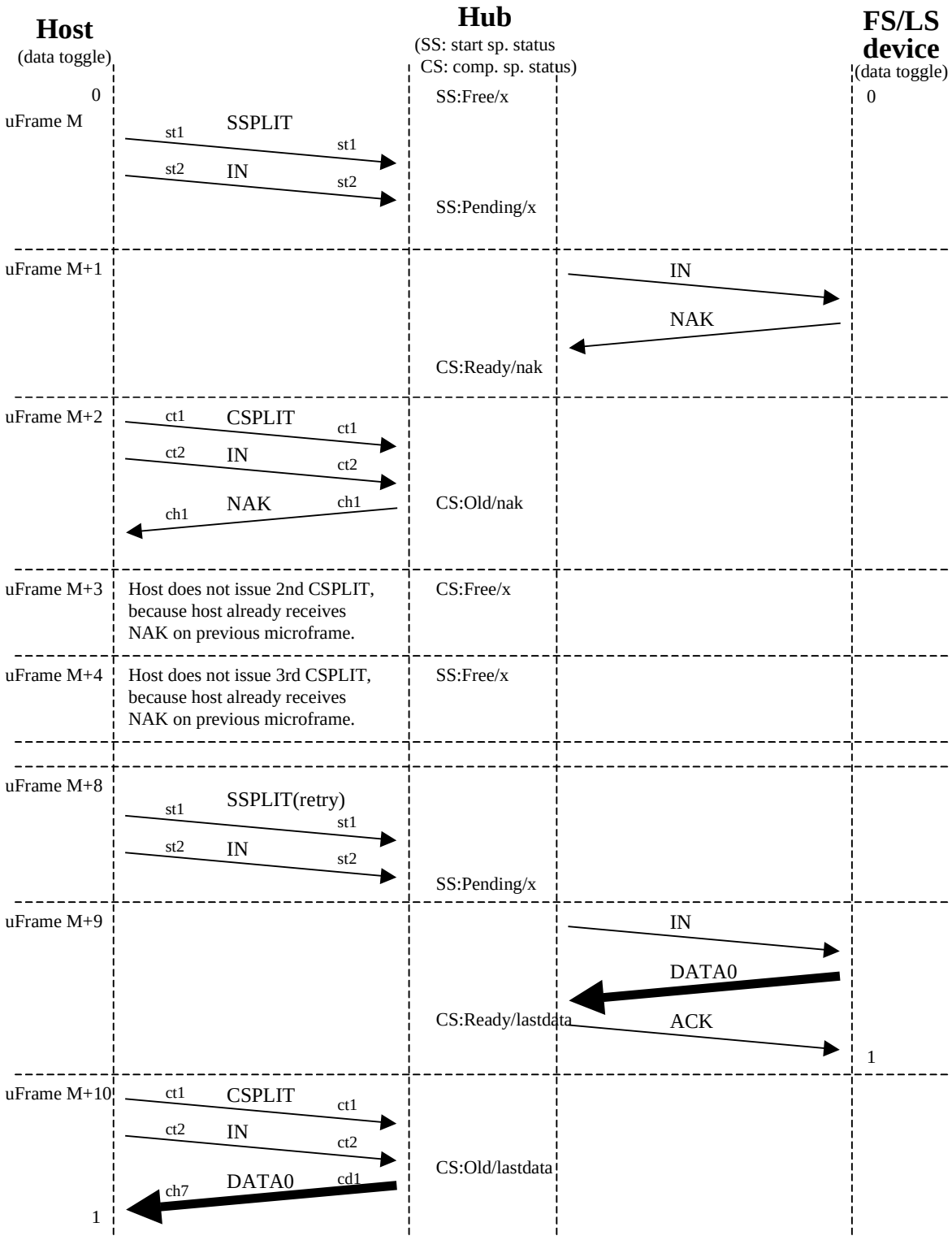


Figure A-82. Device Busy No Smash(FS/LS NAK)

Microchip Ex. 1018, p. 558
Microchip v. Aptiv
IPR2024-00494

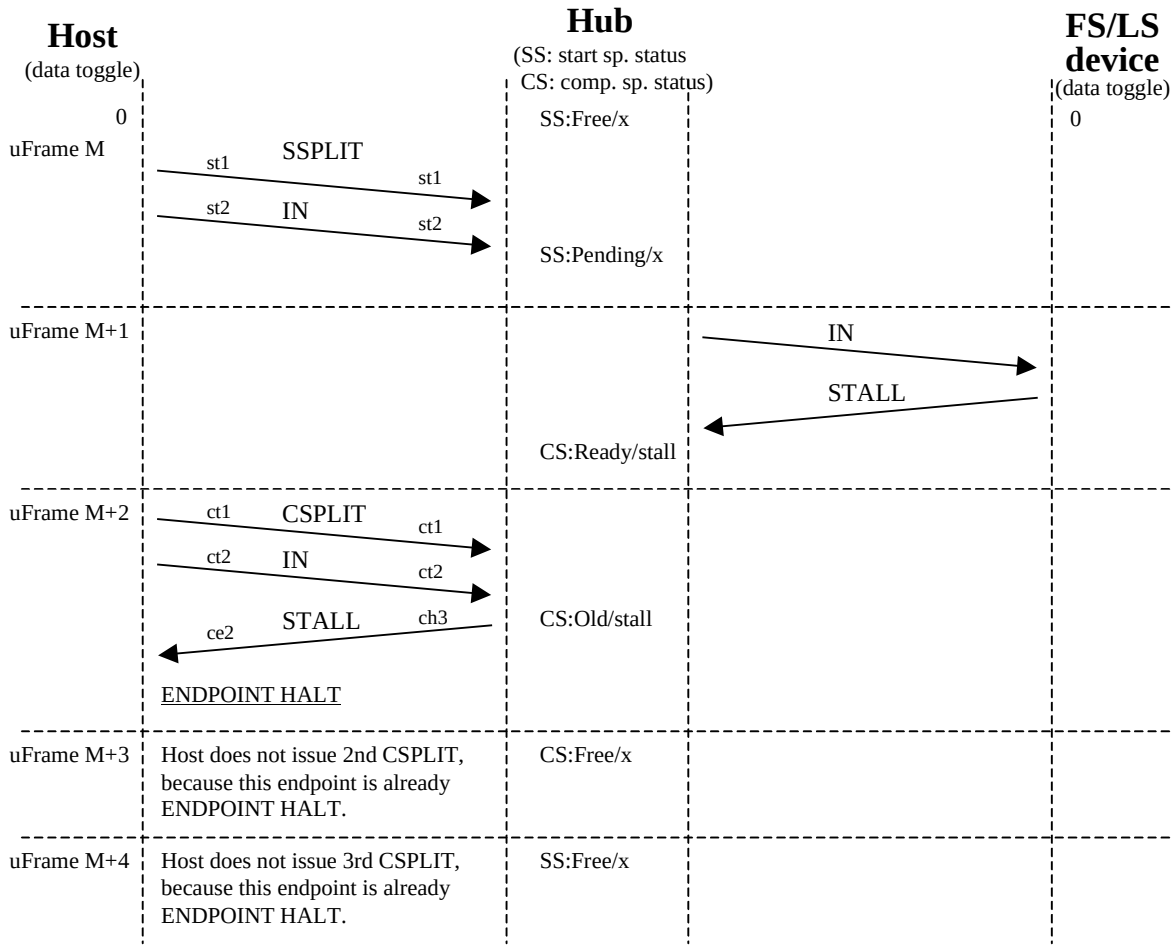


Figure A-83. Device Stall No Smash(FS/LS STALL)

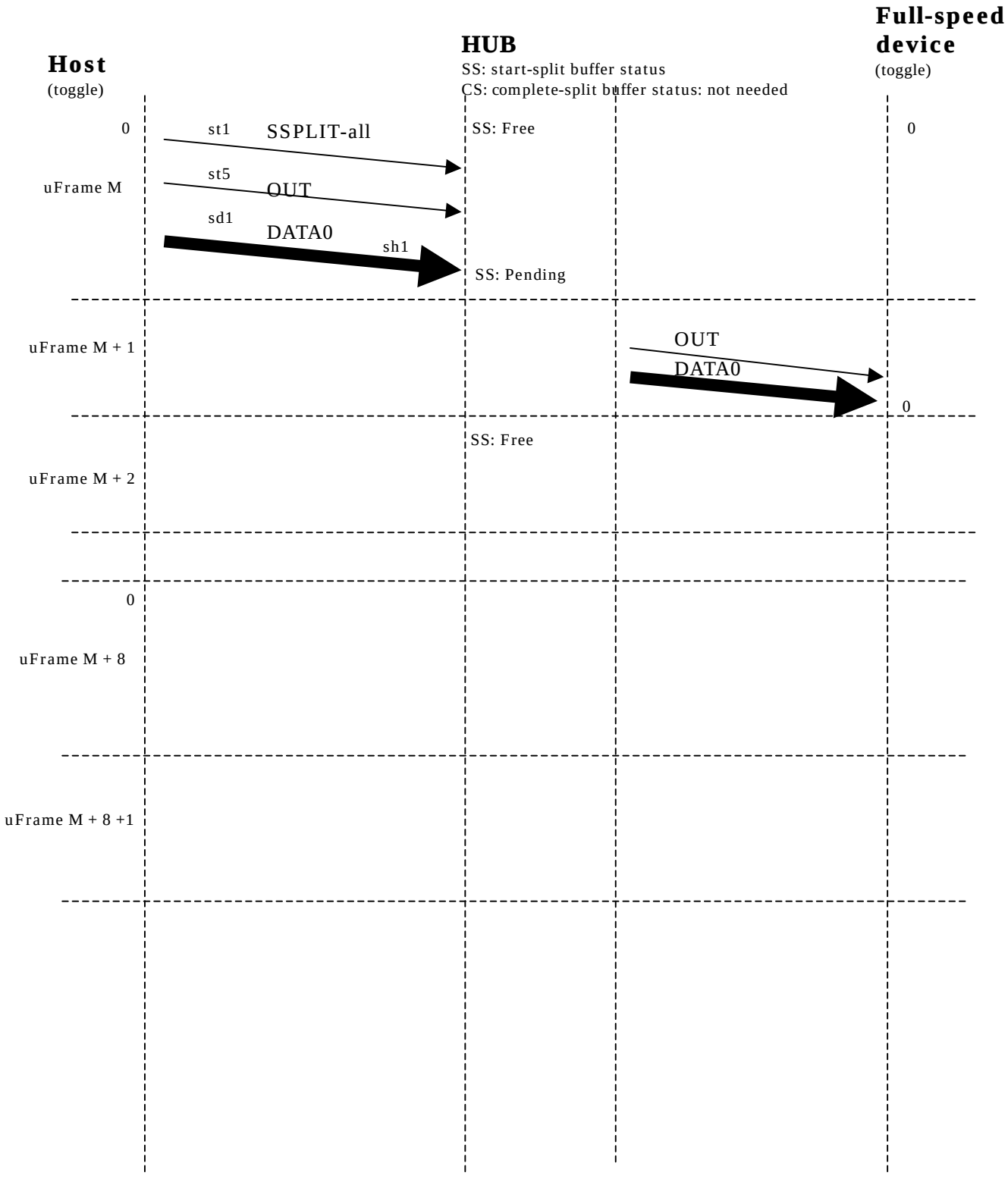
Microchip Ex. 1018, p. 559
Microchip v. Aptiv
IPR2024-00494

A.5 Isochronous OUT Split-transaction Examples

Case	Reference Figure
Normal: small payload (<=188)	1
Normal: large payload (> 188)	2
HS SSPLIT-all corrupted, HS OUT corrupted	3
HS DATA0 corrupted (small payload)	4
HS SSPLIT-begin corrupted	5
HS OUT after the HS SSPLIT-begin is corrupted	6
HS DATA0 corrupted (large payload)	7
HS SSPLIT-mid or OUT or DATA0 corrupted	8

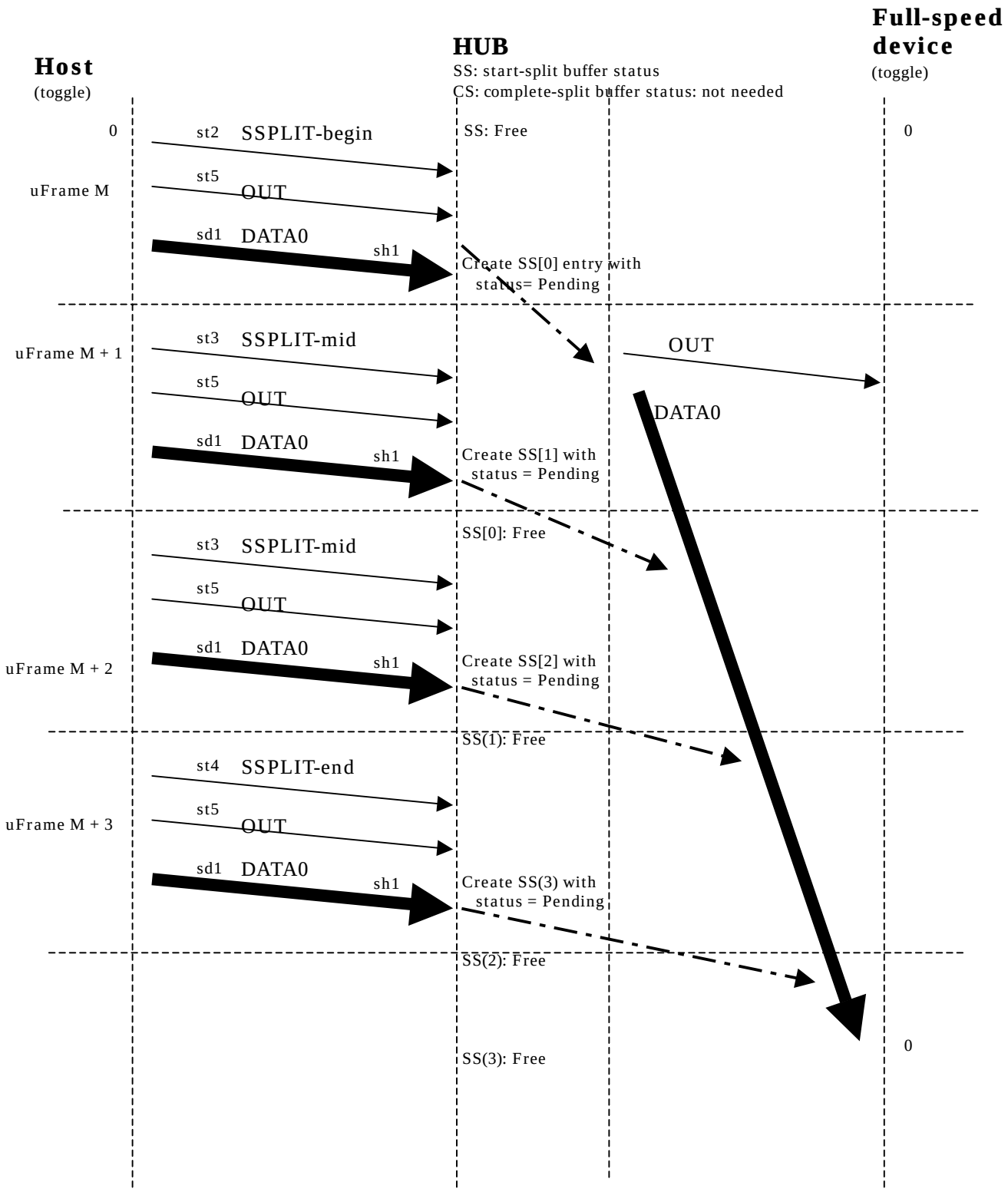
Microchip Ex. 1018, p. 560
Microchip v. Aptiv
IPR2024-00494

1) Normal. Payload <= 188 bytes:



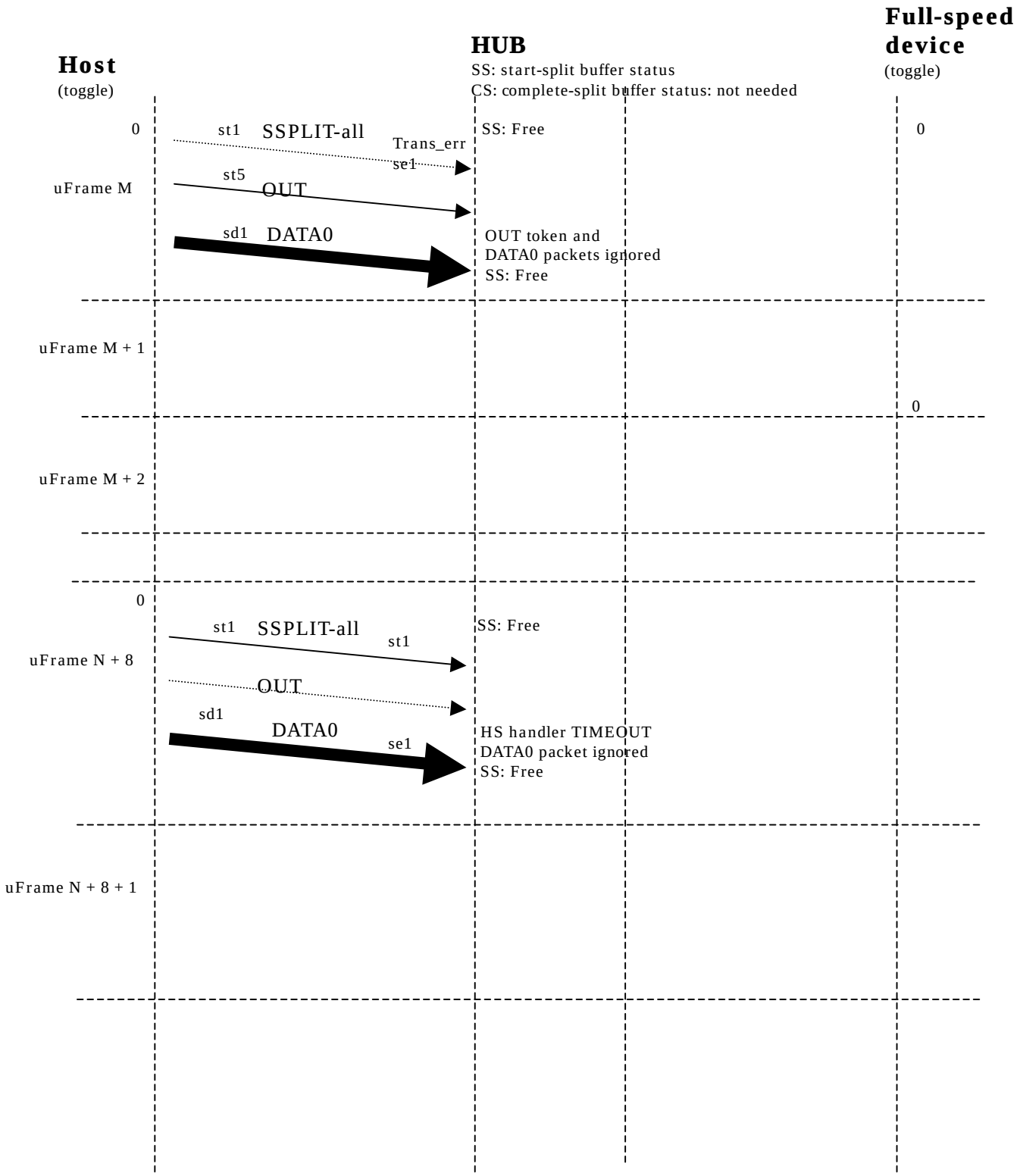
Microchip Ex. 1018, p. 561
Microchip v. Aptiv
IPR2024-00494

2) Normal. Payload > 188 Bytes



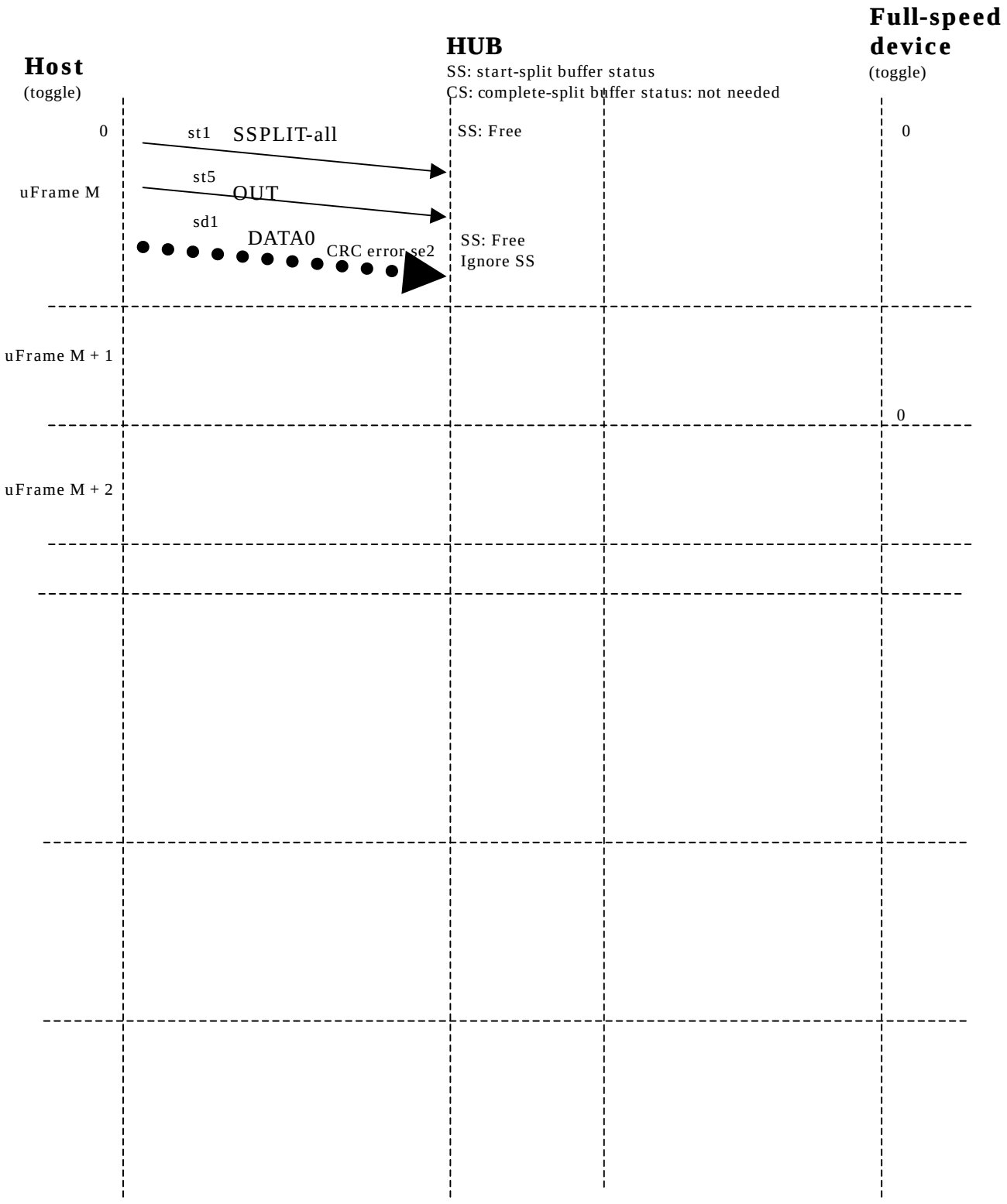
Microchip Ex. 1018, p. 562
Microchip v. Aptiv
IPR2024-00494

3) HS SSPLIT-all corrupted (missing or CRC error etc.) HS OUT corrupted



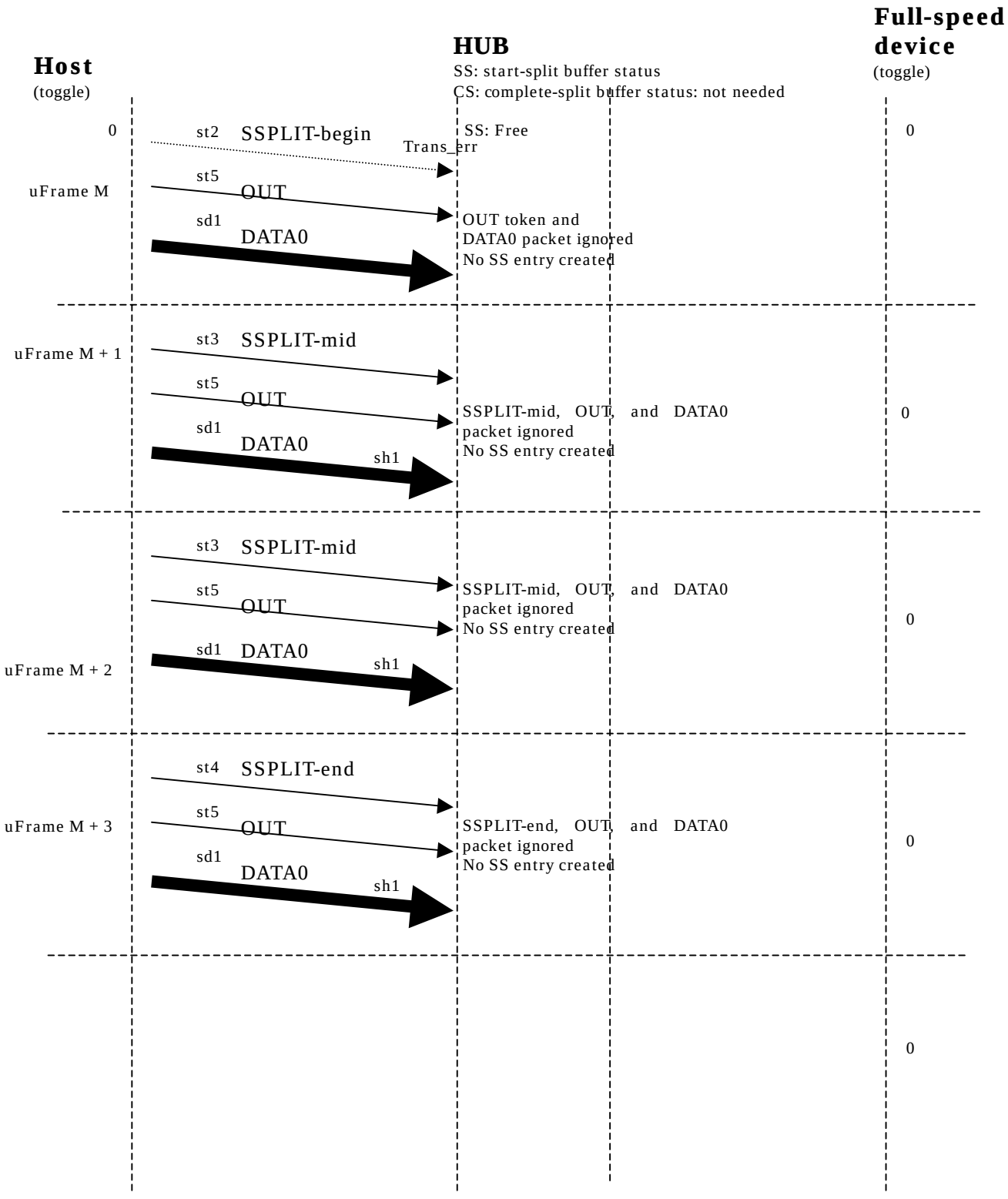
Microchip Ex. 1018, p. 563
Microchip v. Aptiv
IPR2024-00494

4) HS DATA0 corrupted



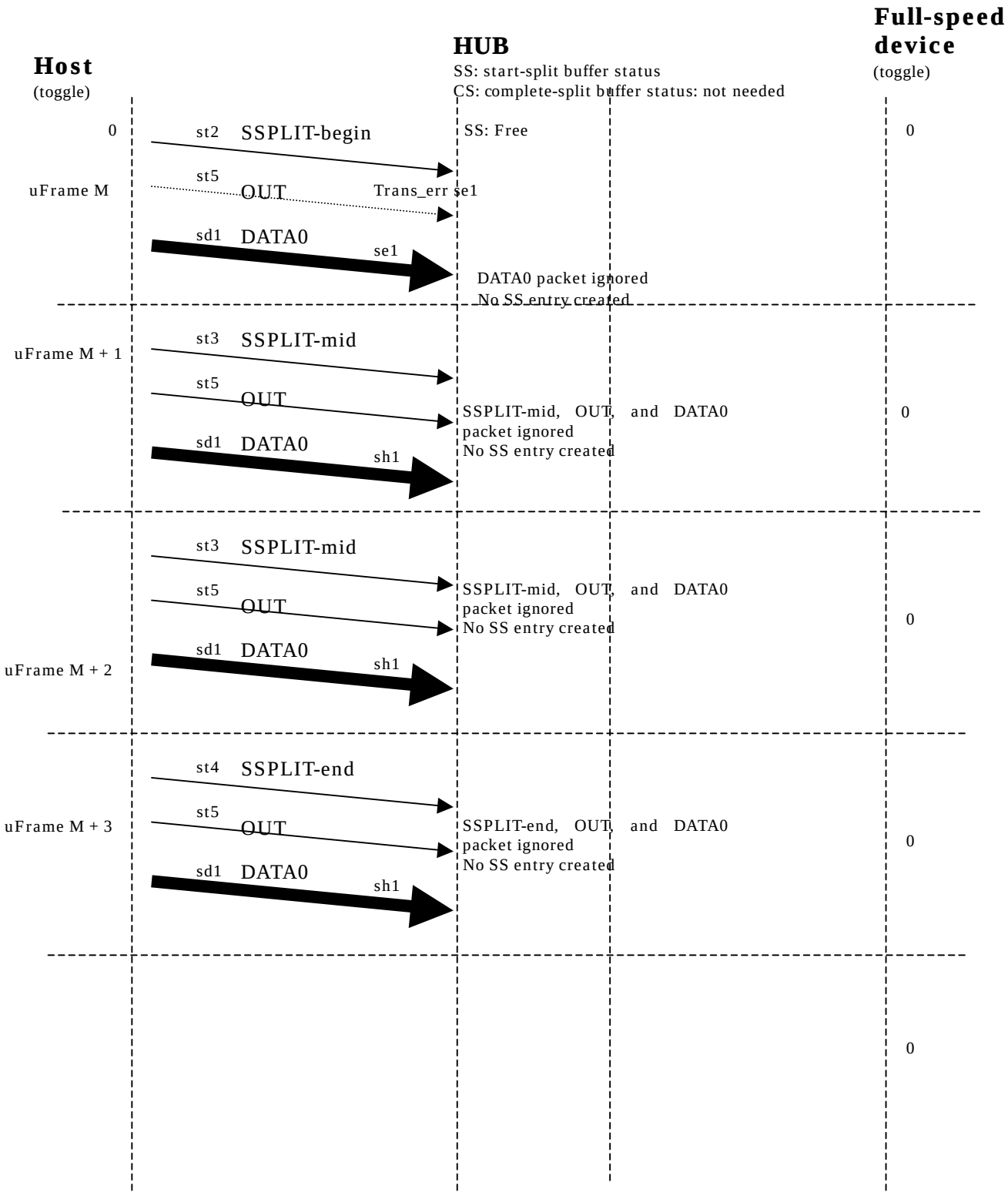
Microchip Ex. 1018, p. 564
Microchip v. Aptiv
IPR2024-00494

5) HS SSPLIT-begin corrupted (missing or CRC error etc.)



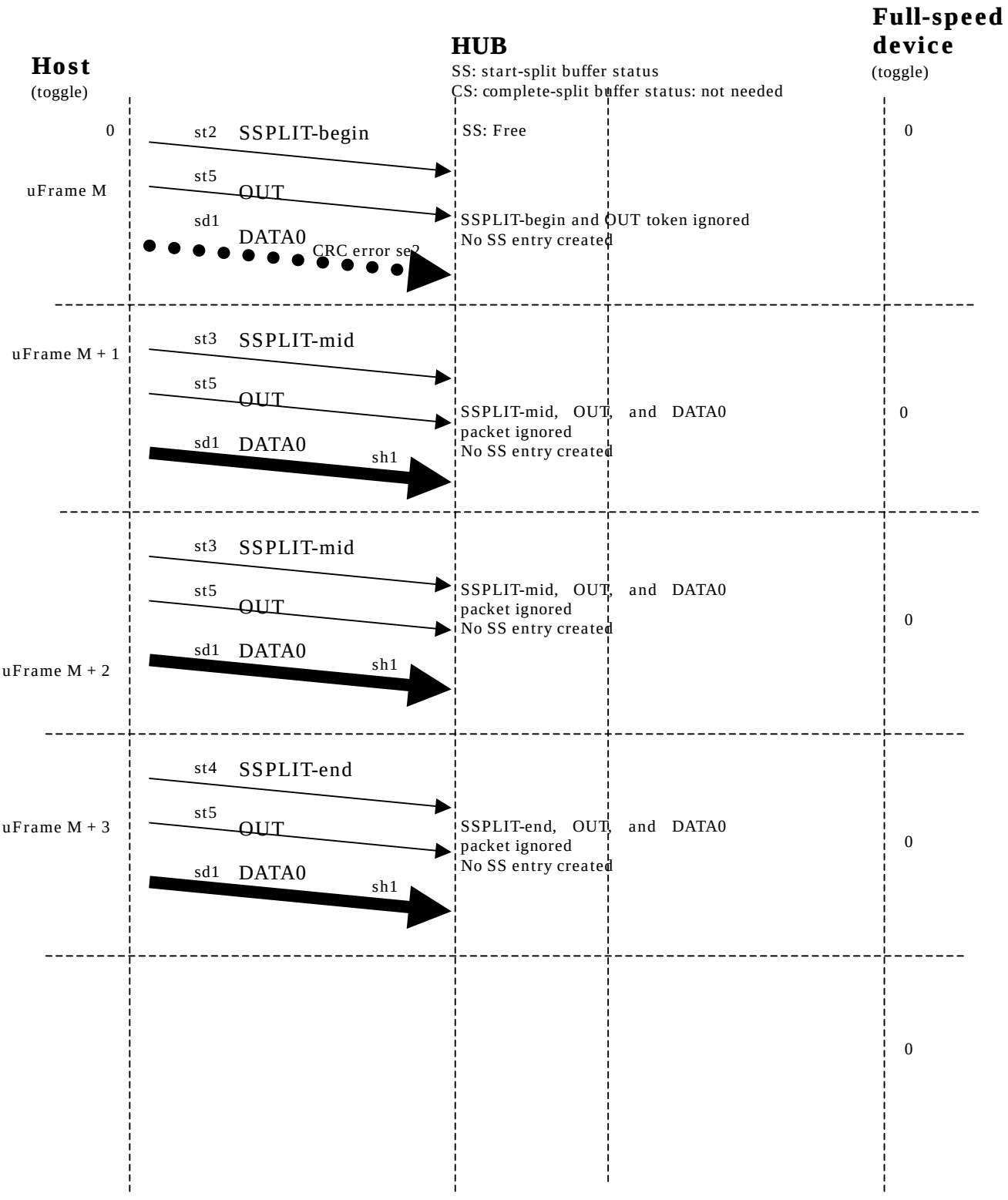
Microchip Ex. 1018, p. 565
Microchip v. Aptiv
IPR2024-00494

6) HS OUT after the HS SSPLIT-begin is corrupted



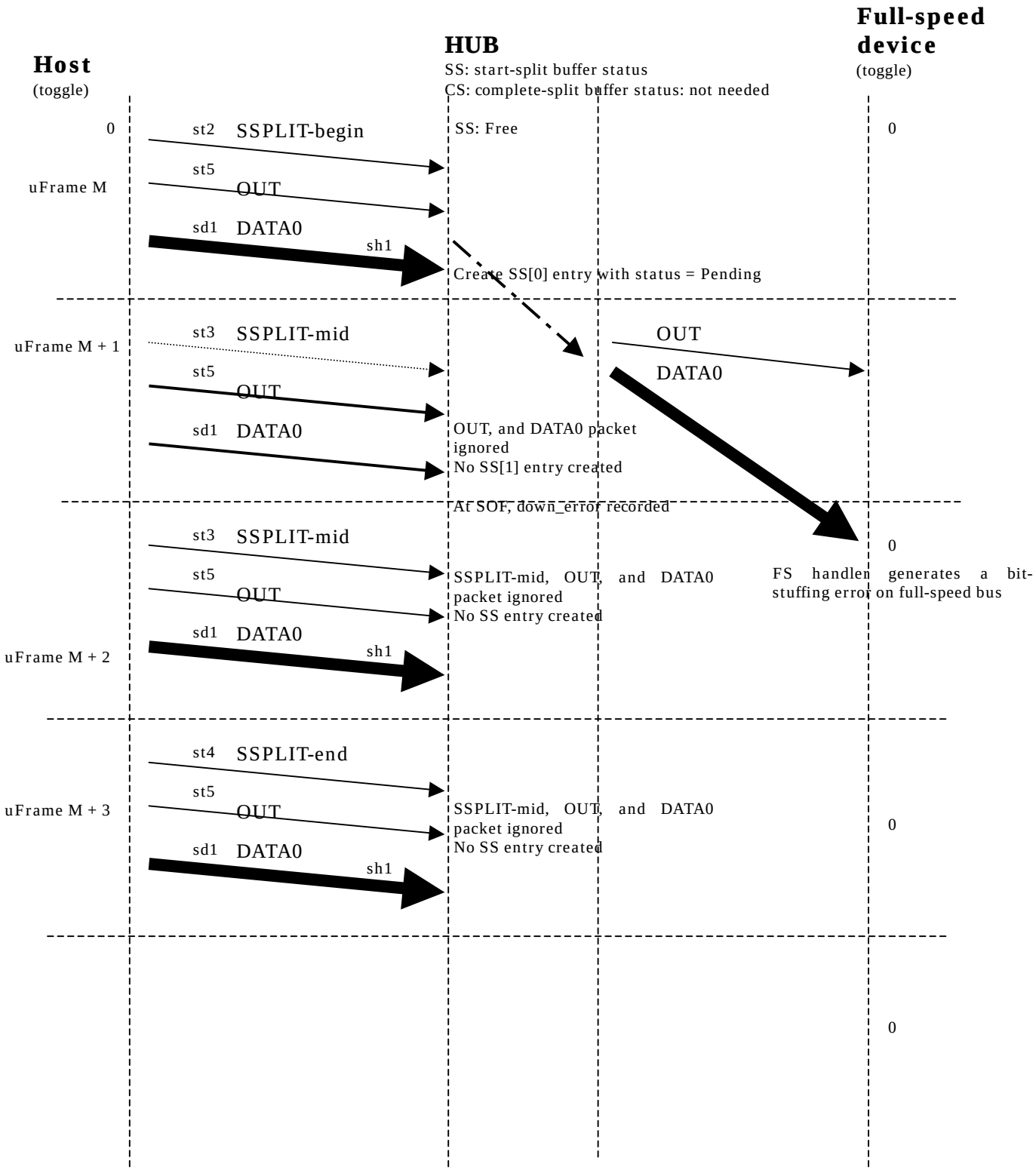
Microchip Ex. 1018, p. 566
Microchip v. Aptiv
IPR2024-00494

7) HS DATA0 corrupted



Microchip Ex. 1018, p. 567
Microchip v. Aptiv
IPR2024-00494

8) HS SSPLIT-mid or OUT token or DATA0 packet after it is corrupted



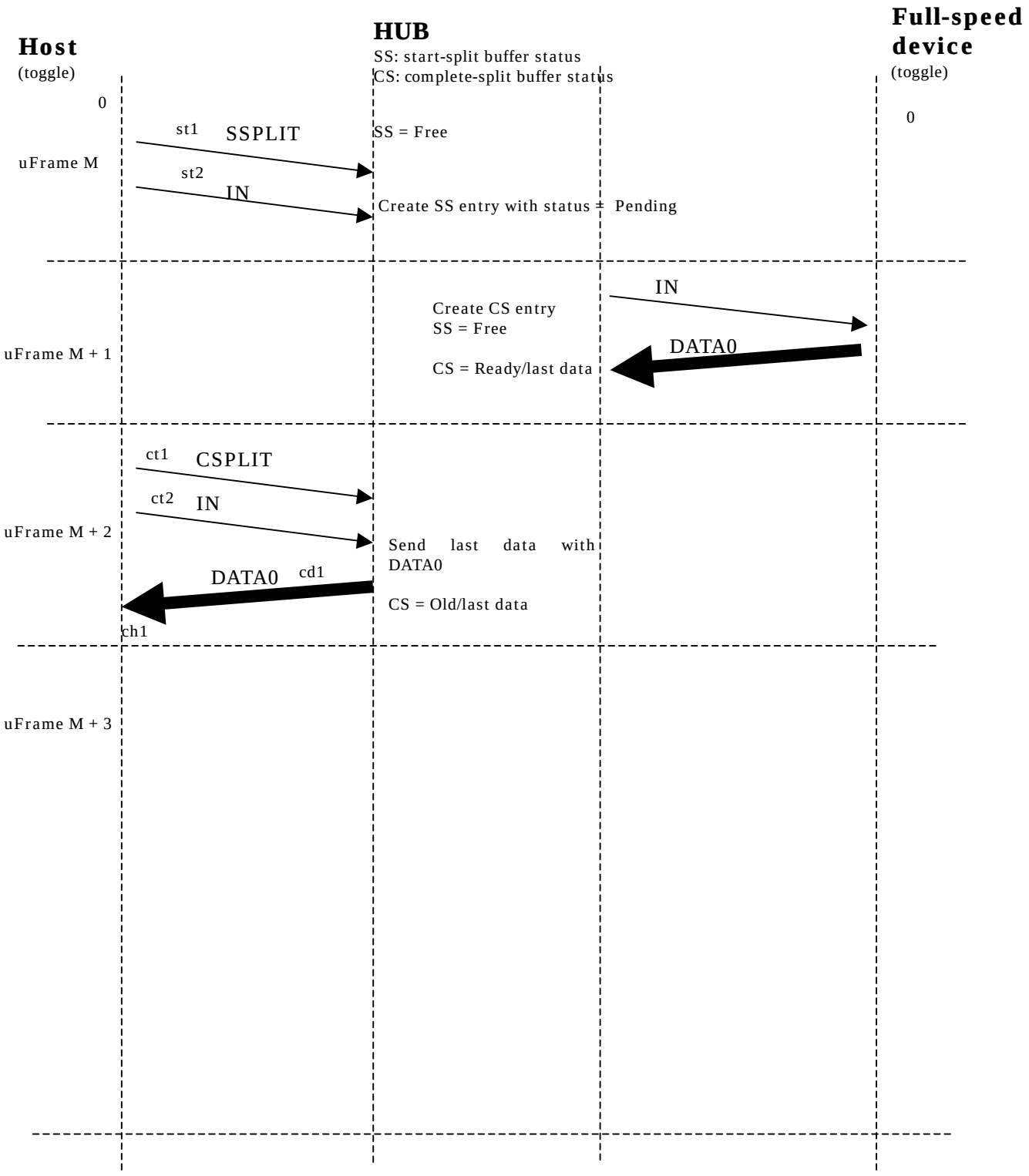
Microchip Ex. 1018, p. 568
Microchip v. Aptiv
IPR2024-00494

A.6 Isochronous IN Split-transaction Examples

Case	Reference Figure
Normal: full-speed bus transaction does not cross microframe boundary	1
Normal: full-speed bus transaction crosses microframe boundary	2
HS SSPLIT corrupted	3
IN after HS SSPLIT corrupted	4
HS CSPLIT corrupted	5
Consecutive HS CSPLIT corrupted	6
HS IN corrupted	7
Consecutive HS IN corrupted	8
HS data corrupted (case 1)	9
HS data corrupted (case 2)	10
TT has more data than HS expects	11
HS CS too early (full-speed data not available yet)	12
Full-speed timeout or CRC error	13

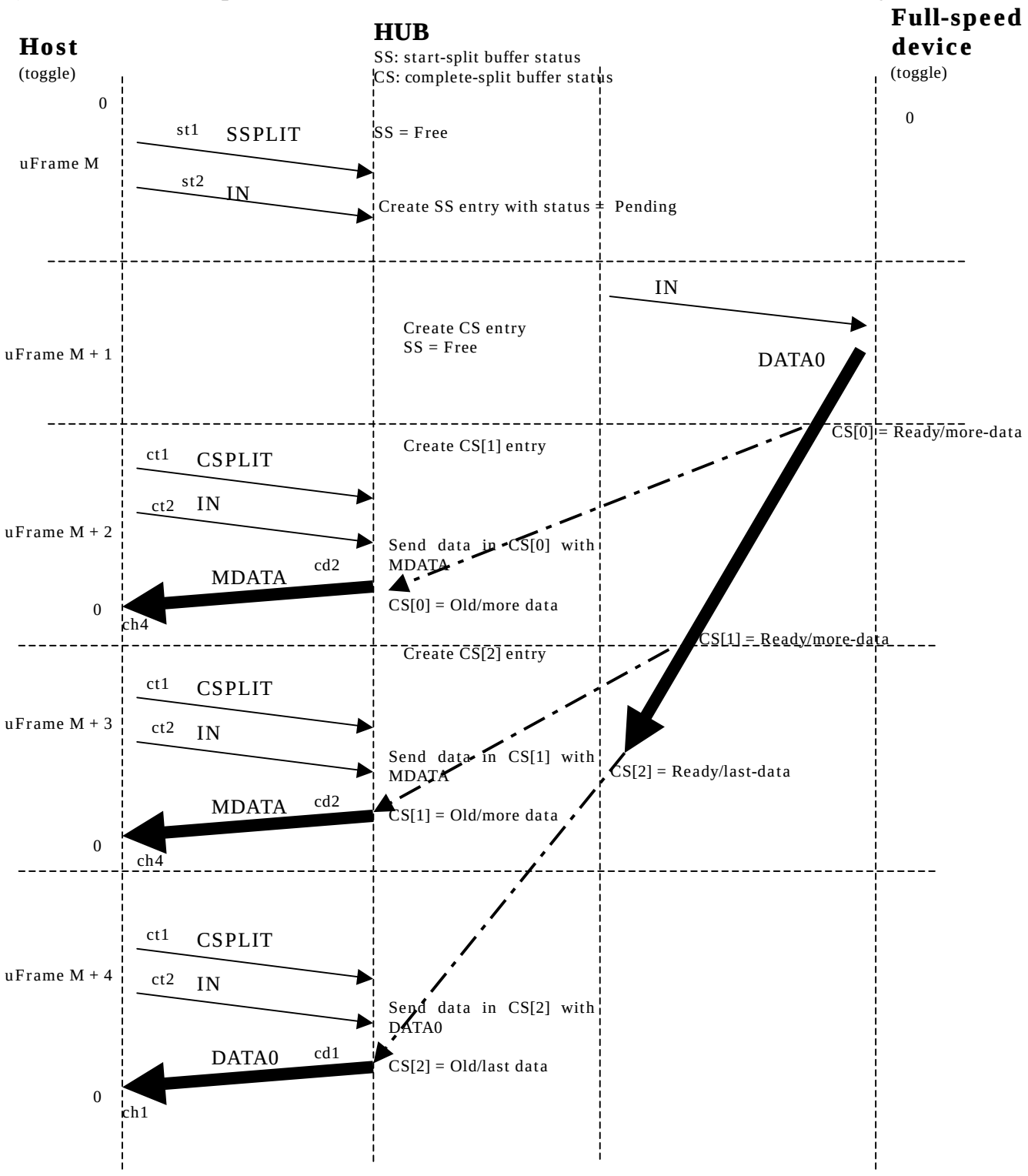
Microchip Ex. 1018, p. 569
Microchip v. Aptiv
IPR2024-00494

1) Normal: full-speed bus transaction does not cross microframe boundary



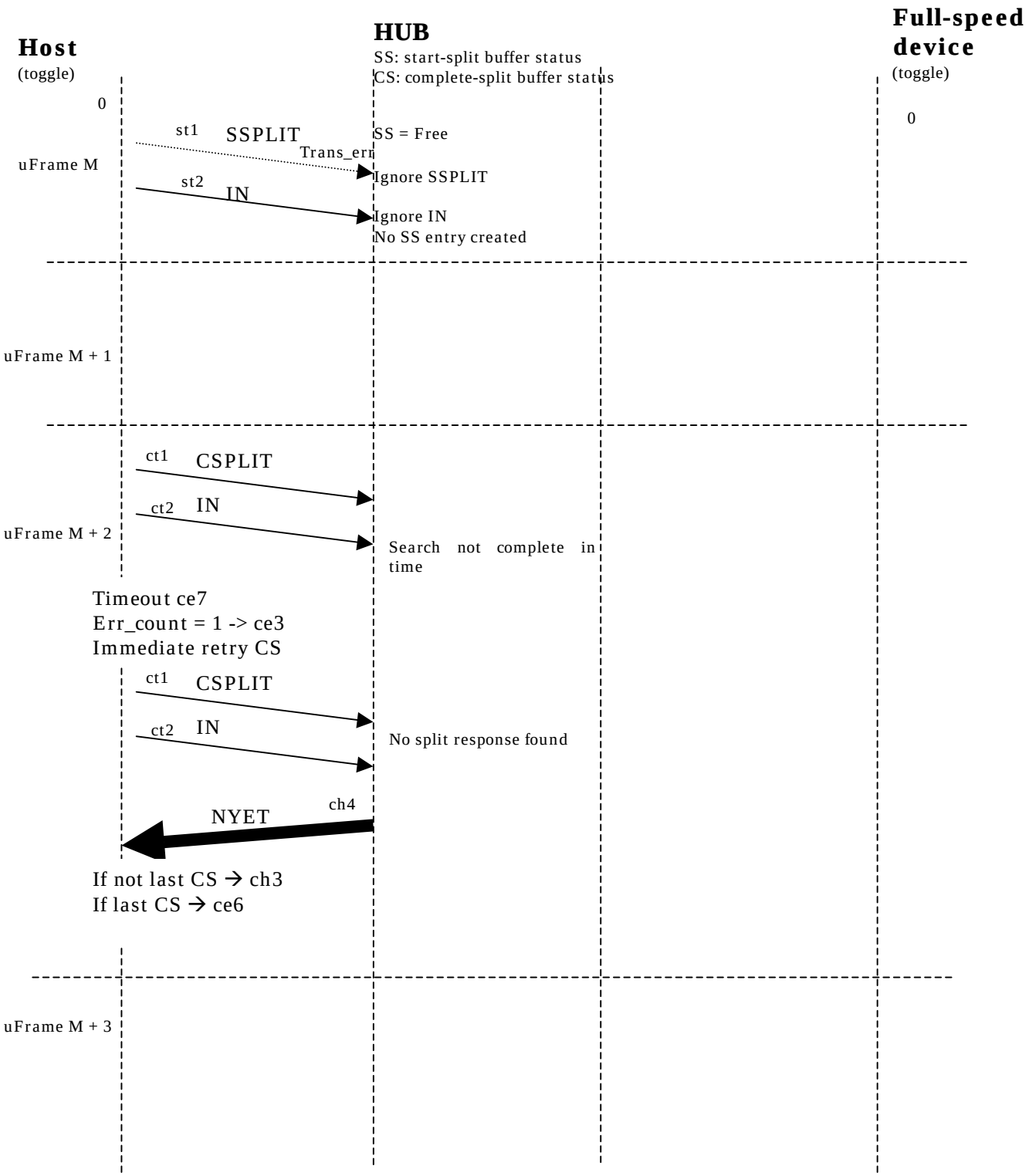
Microchip Ex. 1018, p. 570
Microchip v. Aptiv
IPR2024-00494

2) Normal: full-speed bus transaction crosses microframe boundary



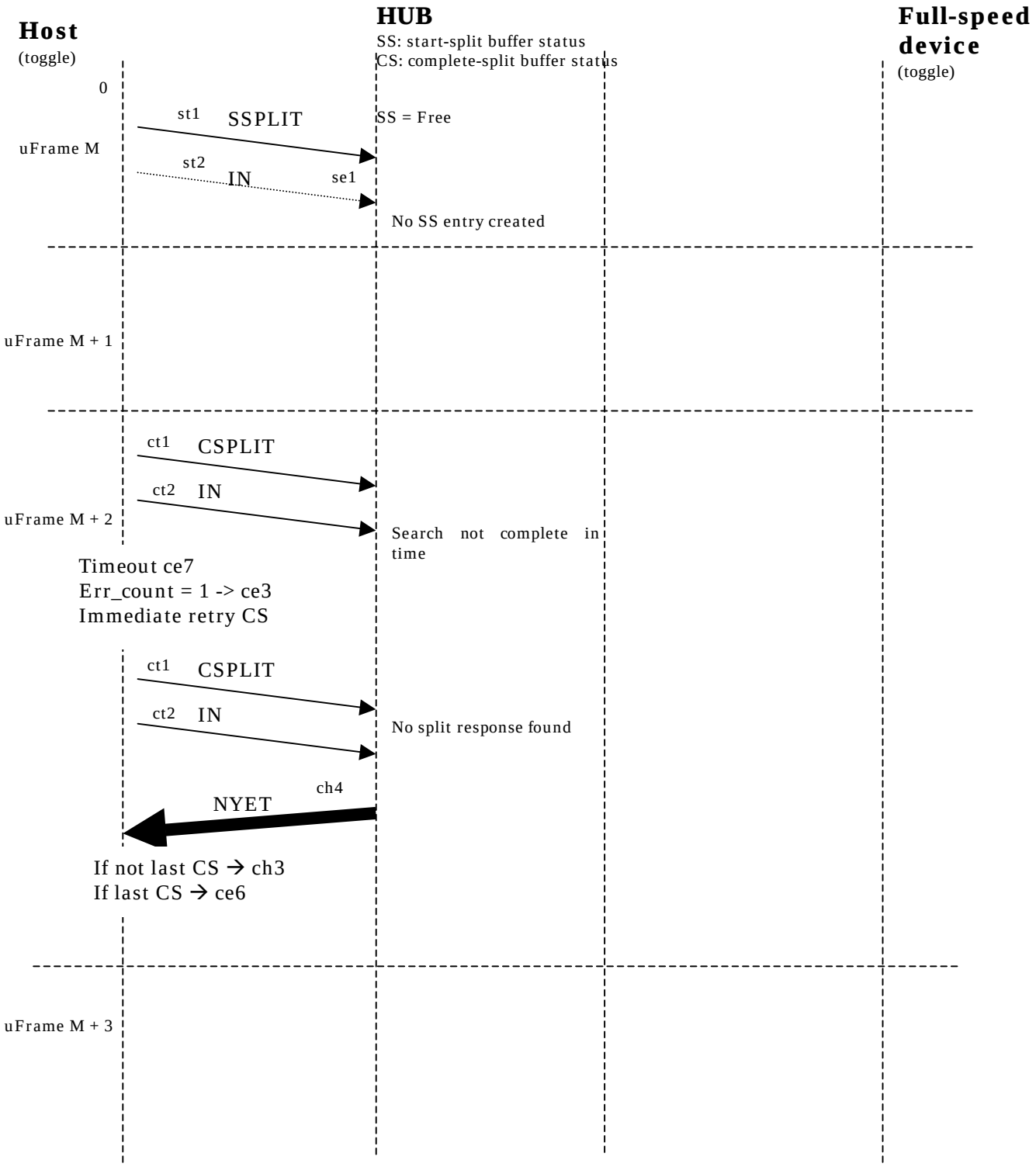
Microchip Ex. 1018, p. 571
Microchip v. Aptiv
IPR2024-00494

3) HS SSPLIT corrupted



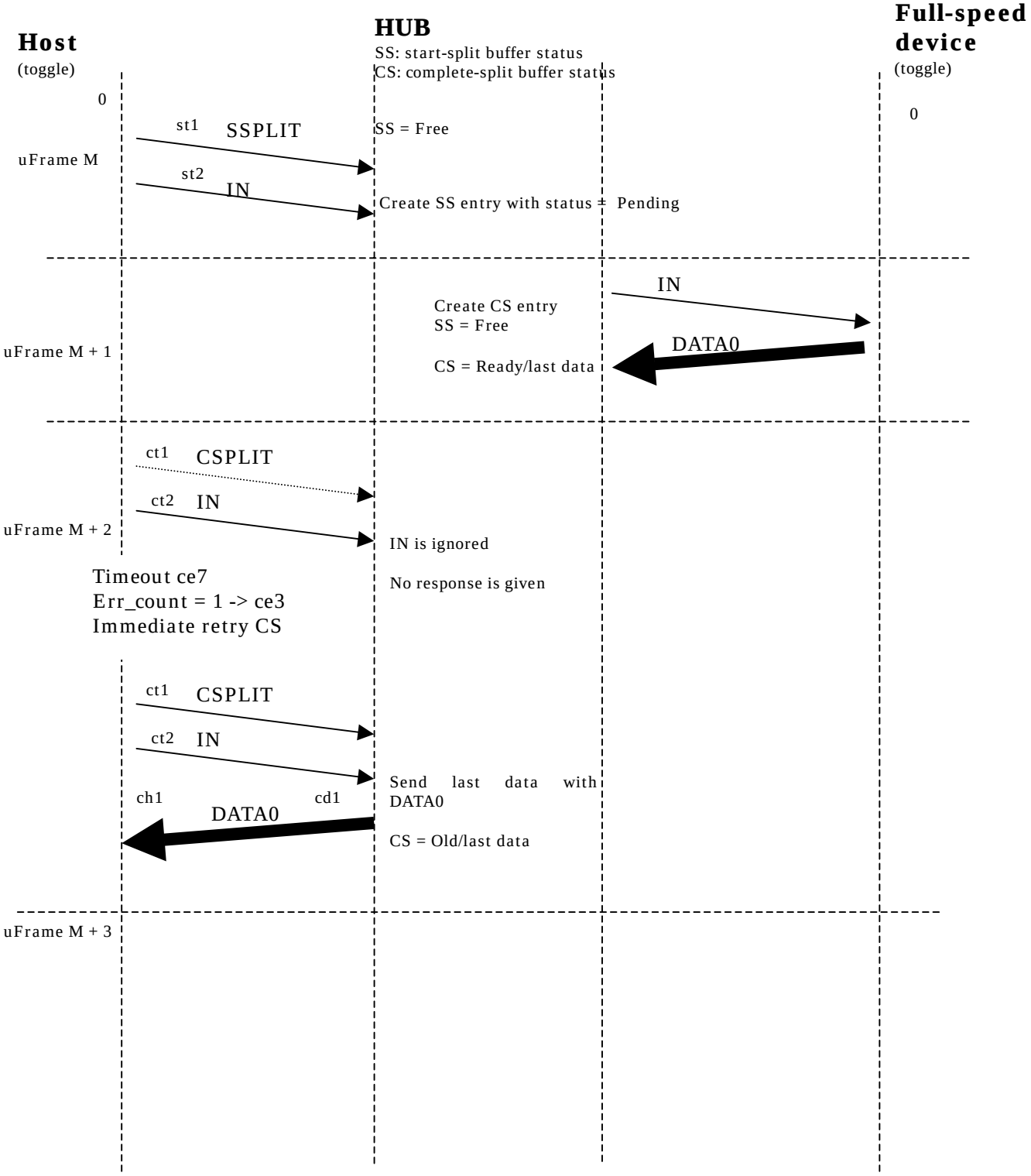
Microchip Ex. 1018, p. 572
Microchip v. Aptiv
IPR2024-00494

4) IN after HS SSPLIT corrupted



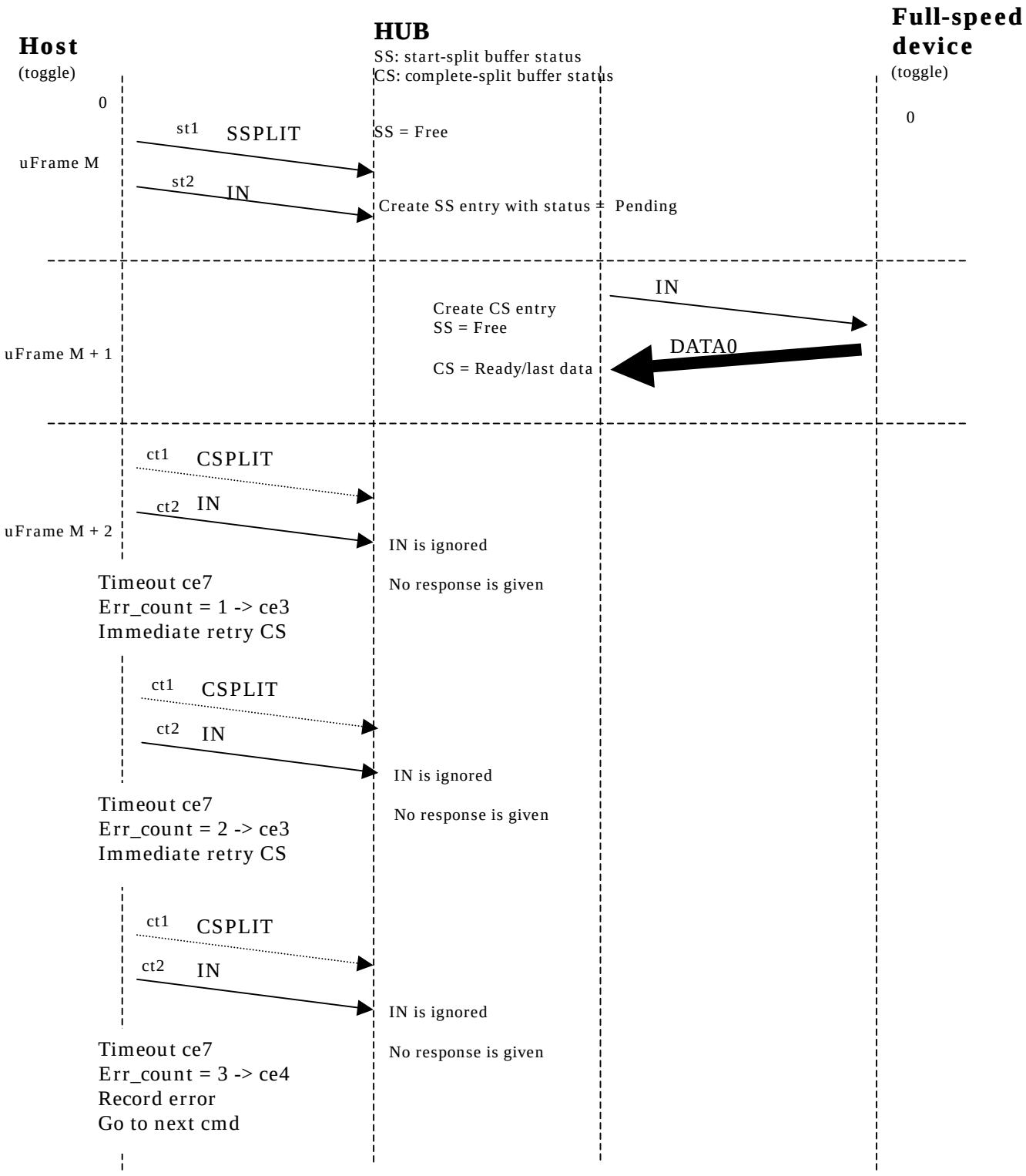
Microchip Ex. 1018, p. 573
Microchip v. Aptiv
IPR2024-00494

5) HS CSPLIT corrupted



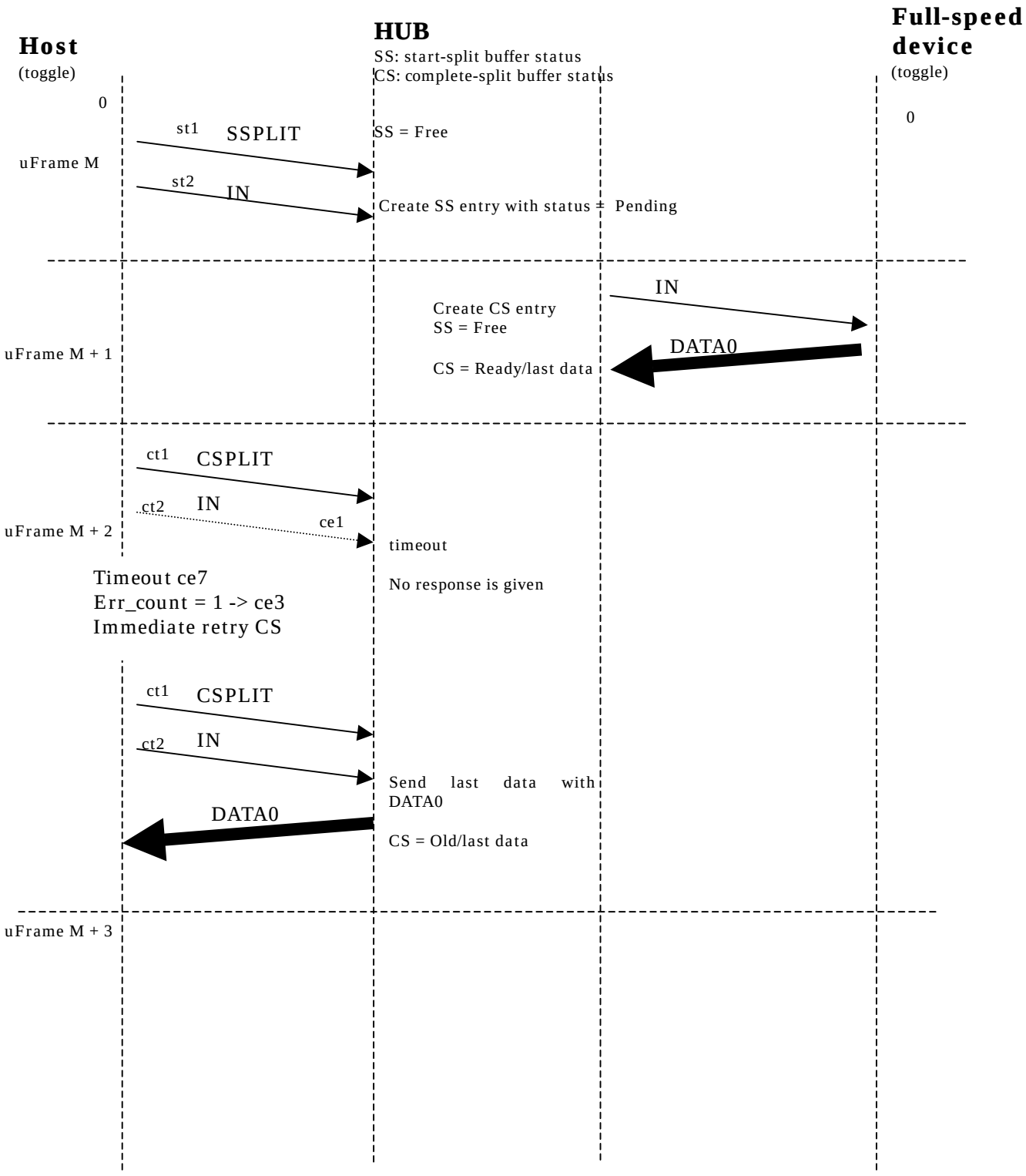
Microchip Ex. 1018, p. 574
Microchip v. Aptiv
IPR2024-00494

6) Consecutive HS CSPLIT corrupted



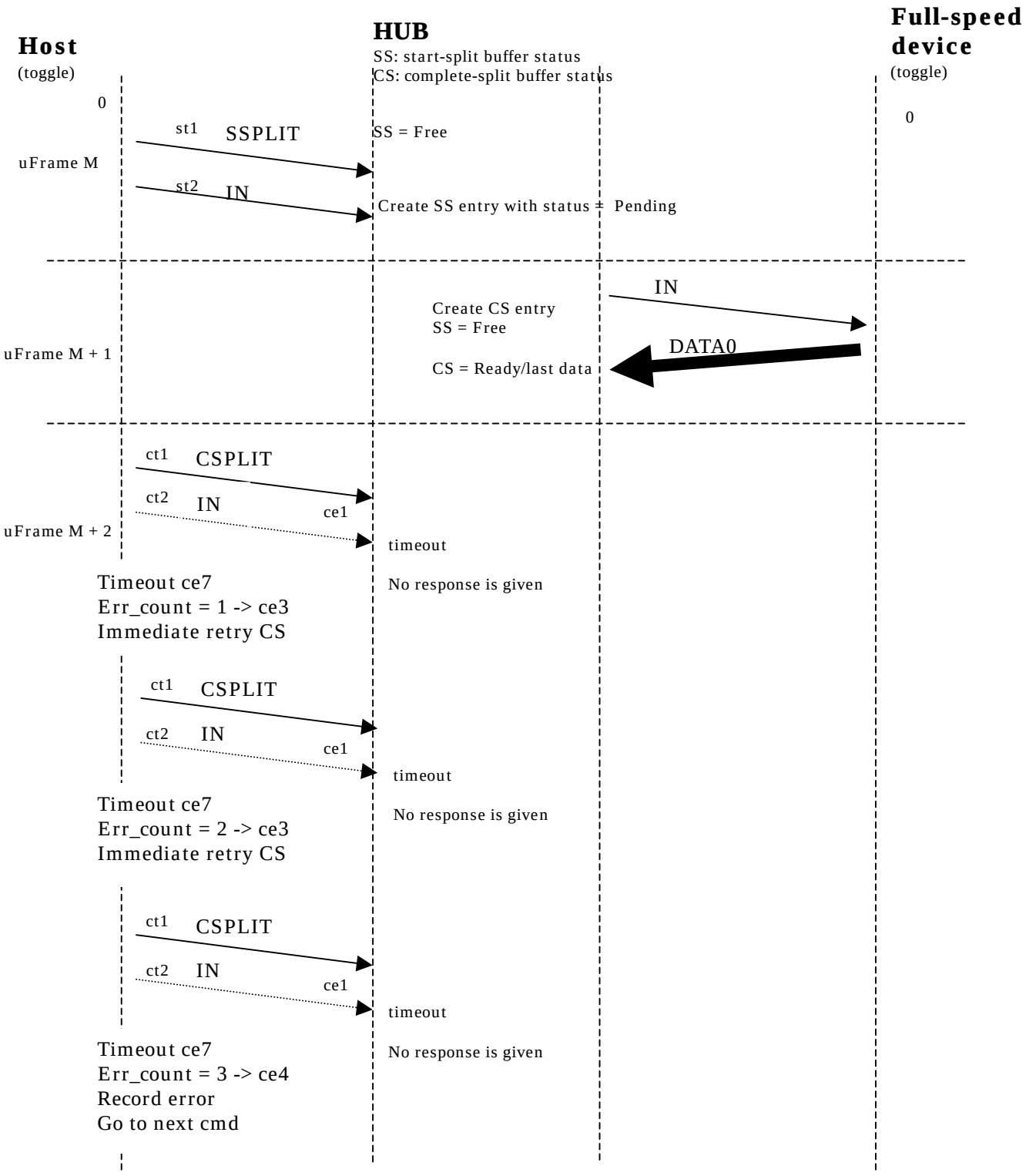
Microchip Ex. 1018, p. 575
Microchip v. Aptiv
IPR2024-00494

7) HS IN corrupted



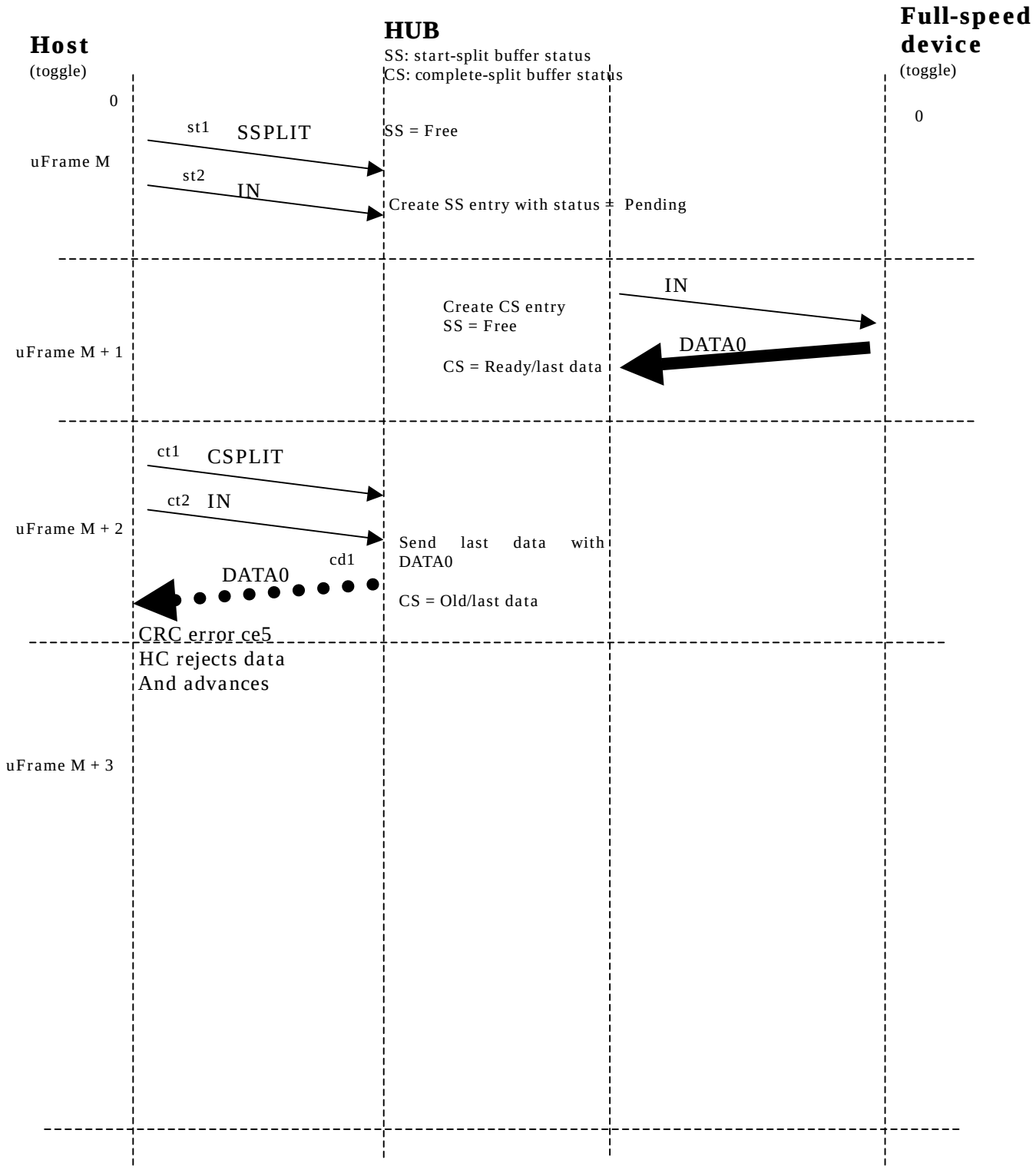
Microchip Ex. 1018, p. 576
Microchip v. Aptiv
IPR2024-00494

8) Consecutive HS IN corrupted



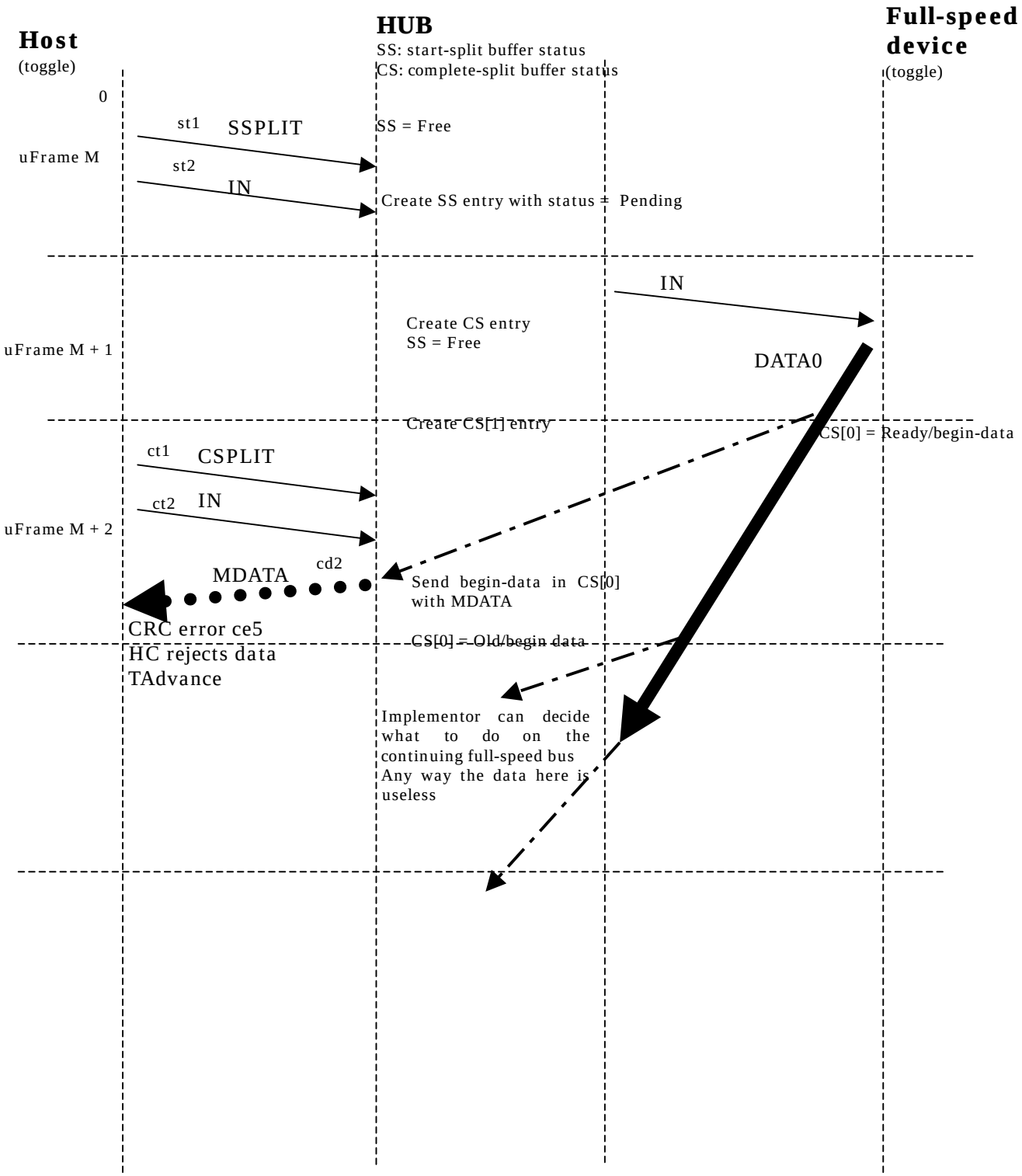
Microchip Ex. 1018, p. 577
Microchip v. Aptiv
IPR2024-00494

9) HS data corrupted (case 1)



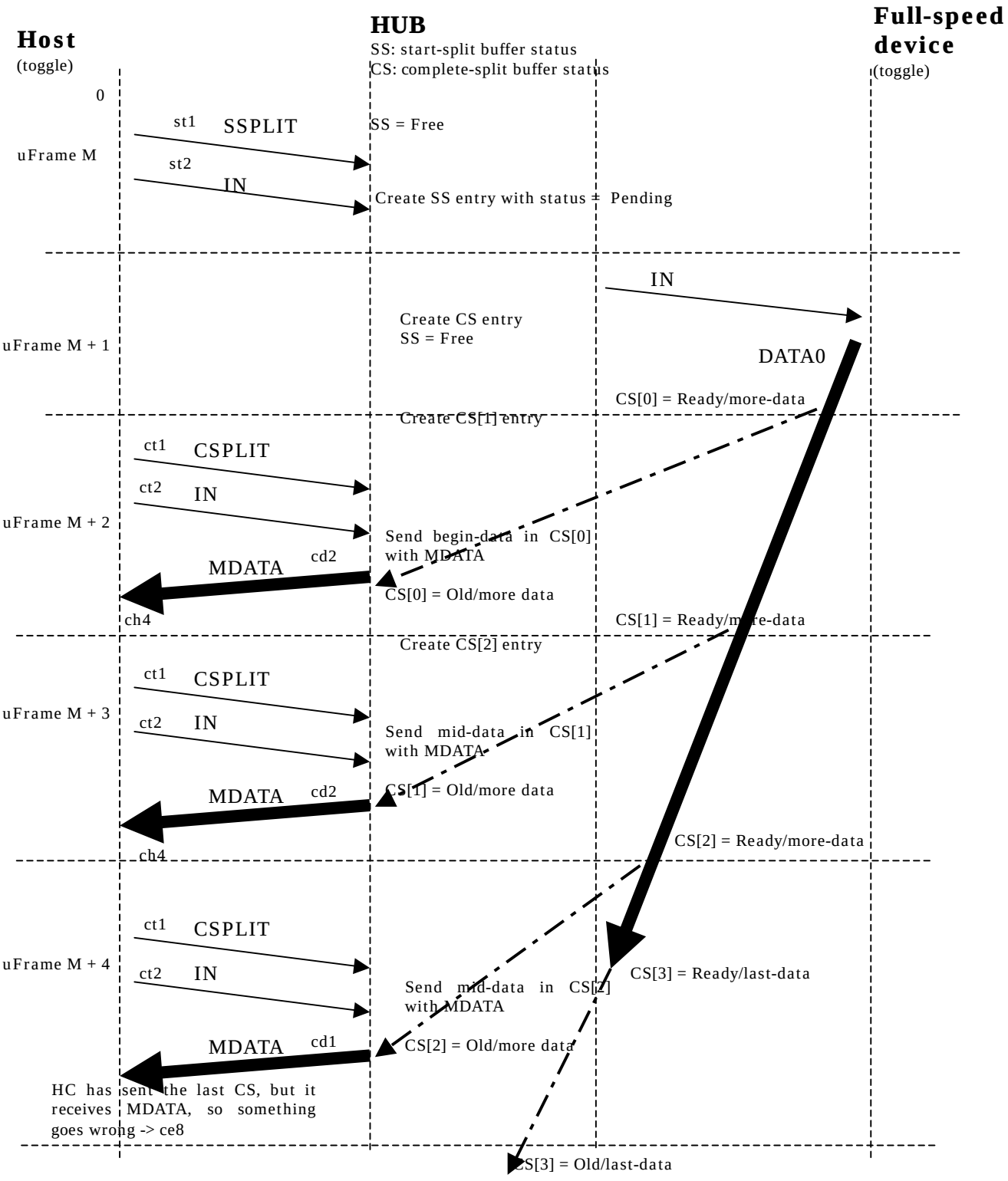
Microchip Ex. 1018, p. 578
Microchip v. Aptiv
IPR2024-00494

10) HS data corrupted (case 2)



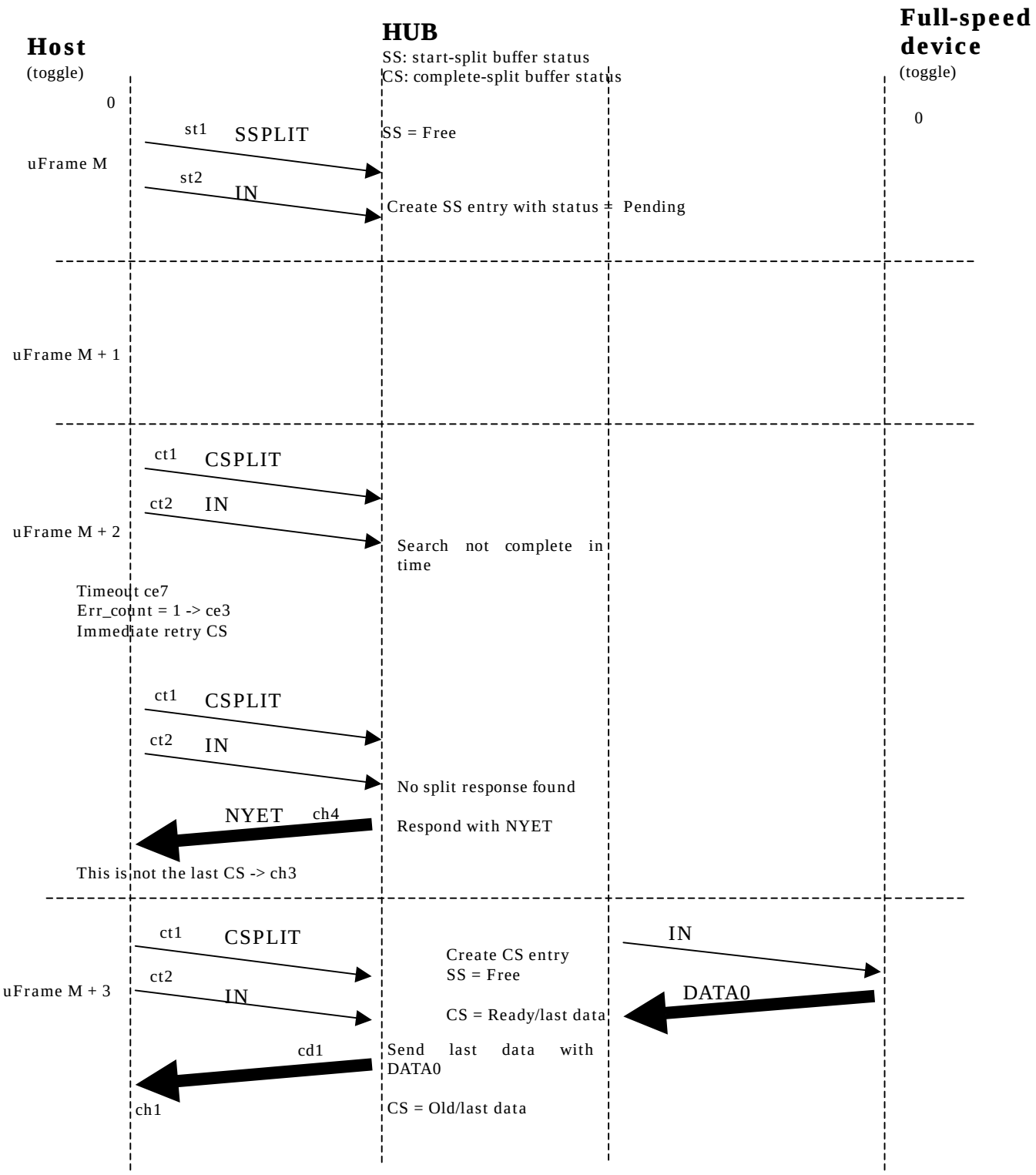
Microchip Ex. 1018, p. 579
Microchip v. Aptiv
IPR2024-00494

11) TT has more data than HC expects



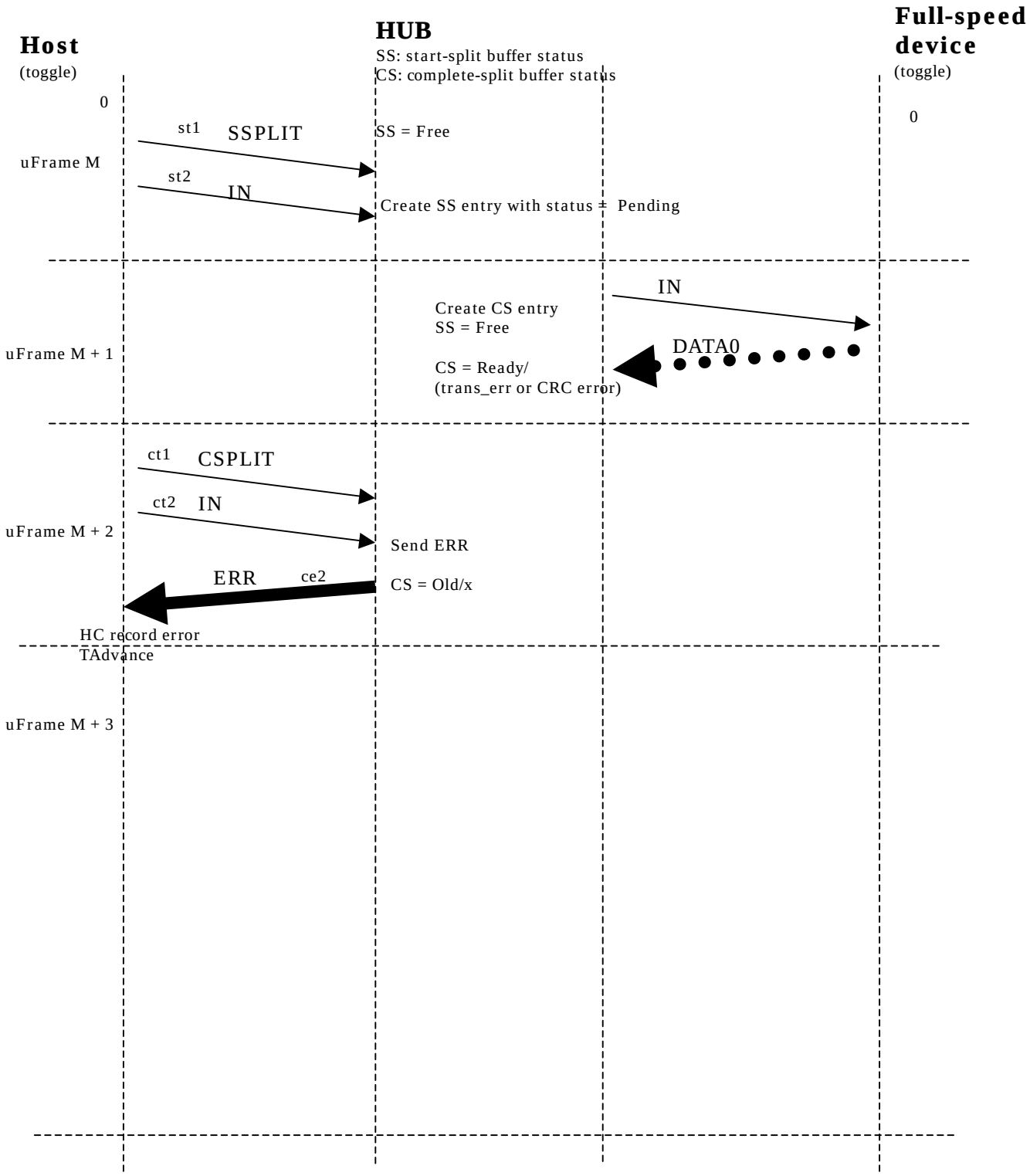
Microchip Ex. 1018, p. 580
Microchip v. Aptiv
IPR2024-00494

12) HS CS too early (full-speed data not available yet)



Microchip Ex. 1018, p. 581
Microchip v. Aptiv
IPR2024-00494

13) Full-speed timeout or CRC error



Microchip Ex. 1018, p. 582
Microchip v. Aptiv
IPR2024-00494

Appendix B
Example Declarations for State Machines

This appendix contains example declarations used in the construction of the state machines in Chapters 8 and 11. These declarations may help in understanding some aspects of the state machines. There are three sets of declarations: global declarations, host controller specific declarations, and transaction translator declarations.

B.1 Global Declarations

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

PACKAGE behav_package IS

    CONSTANT FIFO_DEPTH  : INTEGER := 3;     -- Size of bulk buffer.
                                             -- Determines how many outstanding
                                             -- Split transactions are allowed.

    CONSTANT ERROR_INJECT_DEPTH  : INTEGER := 16;     -- Size of Error Inject FIFO.

    TYPE ep_types IS (bulk, control, isochronous, interrupt);  -- endpoint types

    TYPE directions IS (in_dir, out_dir);  -- data transfer directions

    TYPE pids IS (NAK, ACK, STALL,         -- possible packet PIDs
      tokenIN, tokenOUT, tokenSETUP,
      SOF, ping,
      MDATA,
      DATAx,                      -- represents both DATA0 and DATA1
      CSPLIT, SSPLIT,
      NYET, ERR,
      TRANS_ERR);               -- pseudo PIDs for error cases

    TYPE cmds IS (start_split, complete_split, nonsplit, SOF);       -- HC commands

    TYPE data_choices IS (alldata, begindata, enddata, middata);
      -- isochronous data part for an HC command

    TYPE HCresponses IS (      -- what HC should do next for this command
      do_start,                -- do start-split transaction
      do_complete,             -- do complete-split transaction
      do_complete_immediate,
          -- do complete-split immediately before doing a different transaction
      do_halt,
          -- do endpoint halt processing for the endpoint of this command
      do_next_cmd,             -- do next command for this endpoint
                               -- advance data pointer appropriately
      do_same_cmd,             -- do same command over again
      do_comp_immed_now,
          -- do complete-split imediately within same microframe
      do_next_complete,
          -- do next complete-split in next microframe (periodic)
      do_next_ping,
      do_ping,
      do_out,
      do_idle                  -- Response not active - Used for Simulation
    );

    TYPE Devresponses IS (
      do_next_data,
      do_nothing
    );
```


Microchip Ex. 1018, p. 583
Microchip v. Aptiv
IPR2024-00494

```
TYPE waits IS (
  ITG,                    -- wait up to an inter packet (intra transaction) gap
                          --  for the next packet.
  none);                  -- wait forever for next packet

TYPE CRCs IS (bad, ok);

TYPE states IS (old, pending, ready, no_match, match_busy);
-- states of a buffer

TYPE results IS (          -- full/low speed transaction result in a buffer
        r_ack,
        r_nak,
        r_trans_err,
        r_stall,
        r_badcrc,
        r_lastdata,
        r_moredata,
        r_data);

TYPE epinfo_rec IS RECORD
        space_avail     : boolean;
        data_avail      : boolean;
        ep_type         : ep_types;
        ep_trouble      : boolean;
        toggle          : boolean;
        END RECORD;

TYPE epinfo_array IS ARRAY(1 DOWNTO 0) OF epinfo_rec;

TYPE device_rec IS RECORD
        ep              : epinfo_array;
        HS             : BOOLEAN;
        END RECORD;


TYPE match_rec IS RECORD  -- result of matching a high-speed complete-split
        state           : states;
        down_result     : results;
        END RECORD;

TYPE HS_bus_rec IS RECORD
-- partial high speed transaction state from a high speed bus
        ep_type         : ep_types;
        PID             : pids;
        dev_addr        : INTEGER RANGE 0 TO 127;
        endpt           : INTEGER RANGE 0 TO 15;
        CRC16           : CRCs;
        direction       : directions;
        x               : boolean;
        datapart        : data_choices;
        ready           : boolean;
        timeout         : boolean;
        END RECORD;

TYPE command_rec IS RECORD          -- command state that the HC must act upon
        ep_type         : ep_types;
        cmd             : cmds;
        setup           : boolean;        -- true is control setup
        ping            : boolean;
        HS              : boolean;
        dev_addr        : INTEGER RANGE 0 TO 127;
        endpt           : INTEGER RANGE 0 TO 15;
        CRC16           : CRCs;
        direction       : directions;
        datapart        : data_choices;
        toggle            : boolean;
        last              : boolean;
        END RECORD;
```


Microchip Ex. 1018, p. 584
Microchip v. Aptiv
IPR2024-00494

```
    TYPE bc_buf_status IS (OLD,NU,NOSPACE);     -- Responses from Compare_BC_buff.

    TYPE BC_buff_rec IS RECORD       -- (partial) state of a bulk/control buffer
            match           : match_rec;
            index           : INTEGER RANGE 0 TO (FIFO_DEPTH-1);
            status          : bc_buf_status;
            END RECORD;

    TYPE CS_buff_rec IS RECORD
      -- (partial) state of a periodic complete-split buffer
            match           : match_rec;
            store           : hs_bus_rec;
            END RECORD;

    TYPE SS_buff_rec IS RECORD
            saw_split: boolean;
            isochO:    boolean;  -- was the last transaction an isochronous OUT SS
            lastdata:  data_choices;
              -- if isochO is true, then what was the last data portion
            END RECORD;

    TYPE cam_rec IS RECORD       -- Information stored in the bulk/control Buffer.
            store           : hs_bus_rec;
            match           : match_rec;
            END RECORD;

    TYPE phases IS (SPLIT, TOKEN, DATA);        -- Error Inject phases.

    TYPE err_inject_rec IS RECORD               -- Error Injection FIFO record.
            phase           : phases;
            timeout         : boolean;
            crc             : CRCs;
            pid             : boolean;
            END RECORD;

    TYPE err_inject_type IS ARRAY((ERROR_INJECT_DEPTH - 1) DOWNTO 0)
      of err_inject_rec;

    TYPE cam_type IS ARRAY((FIFO_DEPTH - 1) DOWNTO 0) OF cam_rec;

    --returns true when there is a packet ready to receive from a bus

    FUNCTION Packet_ready(HS_bus_in: HS_bus_rec) RETURN boolean;

    -- wait until there is a packet ready on a bus
    PROCEDURE Wait_for_packet(HS_bus_in: HS_bus_rec; wait_type: waits);

    PROCEDURE RespondDev(dr: devresponses);

    PROCEDURE HC_Accept_data;

    PROCEDURE HC_Reject_data;

    PROCEDURE Dev_Accept_data;

    PROCEDURE Dev_Record_error;
END behav_package;
```


Microchip Ex. 1018, p. 585
Microchip v. Aptiv
IPR2024-00494

B.2 Host Controller Declarations

```
    shared VARIABLE ErrorCount     :  integer :=0;
    shared VARIABLE HC_response_v  : HCresponses;
    shared VARIABLE rd_ptr         : integer RANGE 0 TO (ERROR_INJECT_DEPTH-1) := 0;
    SIGNAL HSU2_ready              : boolean;
    SIGNAL HC_command_ready        : boolean := FALSE;
    SIGNAL HC_cmd                  : command_rec;
    SIGNAL HCresponse              : HCresponses;
    SIGNAL err_inject_fifo         : err_inject_type;
    SIGNAL wr_ptr                  : integer RANGE 0 TO (ERROR_INJECT_DEPTH-1) := 0;

    ------------------------------------------------------------------------------
    -- Issue a packet onto the HS bus.
    ------------------------------------------------------------------------------
    PROCEDURE Issue_packet(SIGNAL HS_bus_out : OUT HS_bus_rec;
                                pid        : pids) IS
    BEGIN

        HS_bus_out.ep_type  <= HC_cmd.ep_type;
        HS_bus_out.endpt    <= HC_cmd.endpt;
        HS_bus_out.dev_addr <= HC_cmd.dev_addr;
        HS_bus_out.direction <= HC_cmd.direction;
        HS_bus_out.datapart <= HC_cmd.datapart;

        HS_bus_out.x        <= HC_cmd.toggle;     -- ????

        -- Check for Error injection when FIFO is not empty.
        IF (wr_ptr /= rd_ptr) THEN

            -- Insert an error during SPLIT phase ?
            IF ((err_inject_fifo(rd_ptr).phase = SPLIT AND (pid = SSPLIT OR pid =
CSPLIT)) OR
            -- Insert an error during Token phase ?
                (err_inject_fifo(rd_ptr).phase = TOKEN AND
                  (pid = tokenIN OR pid = tokenOUT OR pid = tokenSETUP)) OR
            -- Insert an error during Data phase ?
                (err_inject_fifo(rd_ptr).phase = DATA AND (pid = MDATA OR pid = DATAx)))
THEN

                HS_bus_out.crc16   <= err_inject_fifo(rd_ptr).crc;
                HS_bus_out.timeout <= err_inject_fifo(rd_ptr).timeout;
                IF (err_inject_fifo(rd_ptr).pid) THEN
                    HS_bus_out.pid     <= TRANS_ERR;
                ELSE
                    HS_bus_out.pid      <= pid;
                END IF;

                -- Update read pointer.
                IF (rd_ptr = (ERROR_INJECT_DEPTH-1)) THEN
                   rd_ptr := 0;
                ELSE
                    rd_ptr := rd_ptr + 1;
                END IF;
            ELSE
                -- Otherwise issue packet with no errors.
                HS_bus_out.crc16    <= ok;
                HS_bus_out.timeout  <= FALSE;
                HS_bus_out.pid      <= pid;

            END IF;

        -- Otherwise issue packet with no errors.
        ELSE
            HS_bus_out.crc16    <= ok;
            HS_bus_out.timeout  <= FALSE;
            HS_bus_out.pid      <= pid;
        END IF;
        HS_bus_out.ready    <= TRUE;
        HS_bus_out.ready    <= FALSE after 500 ps;
```


Microchip Ex. 1018, p. 586
Microchip v. Aptiv
IPR2024-00494

```
END Issue_packet;

-------------------------------------------------------------------------------
-- Get next command for HC to execute.
-- NOT USED FOR THIS IMPLEMENTATION !!!
-------------------------------------------------------------------------------
PROCEDURE HC_Get_next_command IS
BEGIN
END;

-------------------------------------------------------------------------------
-- Tells HC what happened to this command.
-------------------------------------------------------------------------------
PROCEDURE RespondHC (HCresponse : HCresponses) IS
BEGIN
    HC_response_v := HCresponse;
END;

-------------------------------------------------------------------------------
-- Update command status for the next time the command will be executed by HC.
-- NOT USED FOR THIS IMPLEMENTATION !!!
-------------------------------------------------------------------------------
PROCEDURE Update_command (SIGNAL HCdone : OUT boolean) IS
BEGIN
    HCdone <= TRUE;
END;

-------------------------------------------------------------------------------
-- Increment "3 strikes" error count for endpoint transaction.
-------------------------------------------------------------------------------
PROCEDURE IncError IS
BEGIN
    ErrorCount := ErrorCount + 1;
END;

-------------------------------------------------------------------------------
-- Record Error for current command.
-- NOT USED FOR THIS IMPLEMENTATION !!!
-------------------------------------------------------------------------------
PROCEDURE Record_error IS
BEGIN
    ErrorCount := 0;

END;
```


Microchip Ex. 1018, p. 587
Microchip v. Aptiv
IPR2024-00494

B.3 Transaction Translator Declarations

```
shared VARIABLE cam          : cam_type;         -- TT buffer.
shared VARIABLE BC_buff      : BC_buff_rec;
shared VARIABLE CS_Buff      : CS_buff_rec;
shared VARIABLE rd_ptr       : integer RANGE 0 TO (ERROR_INJECT_DEPTH-1) := 0;
shared VARIABLE derror_v     : boolean;
shared VARIABLE ss_avail_v   : boolean;
shared VARIABLE periodic     : boolean := FALSE;
shared VARIABLE error_time   :time := 1000000000 ns;
SIGNAL split                 : HS_bus_rec;       -- Stored Shared Split Token
SIGNAL token                 : HS_bus_rec;       -- Stored Token
SIGNAL SS_Buff               : SS_buff_rec;
SIGNAL CS_Buff_sig           : CS_buff_rec;
SIGNAL mem                   : cam_rec;
SIGNAL memwrite              : boolean;
SIGNAL err_inject_fifo       : err_inject_type;
SIGNAL wr_ptr                : integer RANGE 0 TO (ERROR_INJECT_DEPTH-1) := 0;
SIGNAL derror                : boolean;
SIGNAL ss_avail              : boolean;

-------------------------------------------------------------------------------
-- Is_no_space - Returns true when there is no space in the Bulk/Control buffers
--               for the current start-split.
-------------------------------------------------------------------------------
function Is_no_space(BC_buff: BC_buff_rec) return boolean is
  variable result:boolean:=FALSE;
begin
  IF (BC_buff.status = NOSPACE) THEN
      result := TRUE;
  END IF;
  return result;
end Is_no_space;

-------------------------------------------------------------------------------
-- Is_new_SS - Returns true when the current high speed start-split is new.
-------------------------------------------------------------------------------
function Is_new_SS(BC_buff: BC_buff_rec) return boolean is
  variable result:boolean:=FALSE;
begin
  IF (BC_buff.status = NU) THEN
      result := TRUE;
  END IF;
  return result;
end Is_new_SS;

-------------------------------------------------------------------------------
-- IS_old_SS - Returns true when the current high speed start-split is a retry.
-------------------------------------------------------------------------------
function Is_old_SS(BC_buff: BC_buff_rec) return boolean is
  variable result:boolean:=FALSE;
begin
  IF (BC_buff.status = OLD) THEN
      result := TRUE;
  END IF;
  return result;
end Is_old_SS;

-------------------------------------------------------------------------------
-- Issue_packet - Issue a packet onto the HS bus.
-------------------------------------------------------------------------------
procedure Issue_packet(signal HS_bus_out : out HS_bus_rec;
                               pid       : pids) IS
begin

  -- Setup HS packet based on whether its periodic or bulk.
    IF (periodic = TRUE) THEN
        HS_bus_out.ep_type      <= CS_Buff.store.ep_type;
        HS_bus_out.endpt        <= CS_Buff.store.endpt;
        HS_bus_out.dev_addr     <= CS_Buff.store.dev_addr;
```


Microchip Ex. 1018, p. 588
Microchip v. Aptiv
IPR2024-00494

```
        HS_bus_out.direction    <= CS_Buff.store.direction;
        HS_bus_out.datapart     <= CS_Buff.store.datapart;
        HS_bus_out.x            <= CS_Buff.store.x;     -- ????
    ELSE
        HS_bus_out.ep_type      <= cam(BC_buff.index).store.ep_type;
        HS_bus_out.endpt        <= cam(BC_buff.index).store.endpt;
        HS_bus_out.dev_addr     <= cam(BC_buff.index).store.dev_addr;
        HS_bus_out.direction    <= cam(BC_buff.index).store.direction;
        HS_bus_out.datapart     <= cam(BC_buff.index).store.datapart;
        HS_bus_out.x            <= cam(BC_buff.index).store.x;     -- ????

        -- Update bulk/control with state information which may have been updated
        -- by the complete-split state machines.
        cam(BC_buff.index).match.state := BC_buff.match.state;
    END IF;


    -- Check for Error injection when FIFO is not empty.
    IF (wr_ptr /= rd_ptr) THEN

        HS_bus_out.crc16   <= err_inject_fifo(rd_ptr).crc;
        HS_bus_out.timeout <= err_inject_fifo(rd_ptr).timeout;
        IF (err_inject_fifo(rd_ptr).pid) THEN
            HS_bus_out.pid     <= TRANS_ERR;
        ELSE
            HS_bus_out.pid      <= pid;
        END IF;

        --IF (now > error_time ) THEN
            -- Update read pointer.
            IF (rd_ptr = (ERROR_INJECT_DEPTH-1)) THEN
               rd_ptr := 0;
            ELSE
                rd_ptr := (rd_ptr + 1);
            END IF;
        --END IF;

        error_time := now;

    -- Otherwise issue packet with no errors.
    ELSE
        HS_bus_out.crc16    <= ok;
        HS_bus_out.timeout  <= FALSE;
        HS_bus_out.pid      <= pid;
    END IF;

    HS_bus_out.ready        <= TRUE;
    HS_bus_out.ready        <= FALSE after 500 ps;


end Issue_packet;

-- returns true when wrong combination of split start and last isoch out transaction
FUNCTION Bad_IsochOut (SS_Buff : SS_Buff_rec;
                       split : HS_bus_rec) RETURN boolean IS
  VARIABLE result:boolean:=FALSE;
BEGIN

  result := ((split.datapart = enddata OR split.datapart = middata) AND
              NOT(SS_Buff.lastdata = begindata OR SS_Buff.lastdata = middata)) OR
            ((split.datapart = begindata OR split.datapart = alldata) AND
              SS_Buff.isochO) OR
            ((split.datapart = middata OR split.datapart = enddata) AND NOT
              SS_Buff.isochO);

  RETURN result;
END Bad_IsochOut;

--------------------------------------------------------------------------------
-- Save - Save the Packet for use later.
--------------------------------------------------------------------------------
```


Microchip Ex. 1018, p. 589
Microchip v. Aptiv
IPR2024-00494

```
procedure Save(hs_bus_in : IN HS_bus_rec;
               SIGNAL hs_bus_out: OUT HS_bus_rec) IS
begin
    hs_bus_out <= hs_bus_in;
end Save;

--------------------------------------------------------------------------------
--  Compare_BC_buff - This procedure is used to look at the BC buffer to determine
--                    whether the packet should be stored. Compare_BC_buff will
--                    initialize BC_buff with the buffer location information.
--------------------------------------------------------------------------------
procedure Compare_BC_buff IS
  variable match:boolean:=FALSE;
begin

   -- Assume nospace and intialize index to 0.
   BC_buff.status := NOSPACE;
   BC_buff.index  := 0;

   FOR i IN 0 to FIFO_DEPTH-1 LOOP
       IF NOT match THEN
           -- Re-use buffer with same Device Address/End point.
           IF (token.endpt = cam(i).store.endpt AND
                token.dev_addr = cam(i).store.dev_addr AND
               ((token.direction = cam(i).store.direction AND
                 split.ep_type /= CONTROL) OR
               split.ep_type = CONTROL)) THEN

             -- If The buffer is already pending/ready this must be a retry.
             IF (cam(i).match.state = READY OR cam(i).match.state = PENDING) THEN
                 BC_buff.status := OLD;
             ELSE
                 BC_buff.status := NU;
             END IF;
             BC_buff.index := i;
             match := TRUE;

           -- Otherwise use the buffer if it's old.
           ELSIF (cam(i).match.state = OLD) THEN
             BC_buff.status := NU;
             BC_buff.index := i;
           END IF;
       END IF;
   END LOOP;

   BC_buff.match.state := cam(BC_buff.index).match.state;

end Compare_BC_buff;

--------------------------------------------------------------------------------
-- Accept_data - Store start-split into bulk/control buffer. Index is setup
--               in a previous call to Compare_BC_buff.
--------------------------------------------------------------------------------
procedure Accept_data IS
begin
    cam(BC_buff.index).store := token;
    cam(BC_buff.index).match.state := PENDING;
    BC_buff.match.state := PENDING;
end Accept_data;

--------------------------------------------------------------------------------
-- Match_split_state - This procedure finds the BC buffer location which matches
--                     the current complete-split.
--------------------------------------------------------------------------------
procedure Match_split_state IS
  variable match:boolean:=FALSE;
begin
   BC_buff.match.state  := NO_MATCH;
   BC_buff.index  := 0;

   FOR i IN 0 to FIFO_DEPTH-1 LOOP
```


Microchip Ex. 1018, p. 590
Microchip v. Aptiv
IPR2024-00494

```
       IF NOT match THEN
           -- Is this the buffer used for the start-split
           --  corresponding to this complete-split?
           -- If it is... store information into BC_buff and
           -- indicate match was found.

           IF (token.endpt = cam(i).store.endpt AND
               token.dev_addr = cam(i).store.dev_addr AND
               token.direction = cam(i).store.direction) THEN

             BC_buff.match.state       := cam(i).match.state;
             BC_buff.match.down_result := cam(i).match.down_result;
             BC_buff.index := i;
             match := TRUE;
           END IF;
       END IF;
   END LOOP;

   periodic := FALSE;                -- Setup Issue Packet.

end Match_split_state;

-------------------------------------------------------------------------------
-- Record an error in the SS pipeline for fowarding on the downstream bus.
-------------------------------------------------------------------------------
PROCEDURE Down_error IS
BEGIN
    derror_v := TRUE;
END Down_error;

-------------------------------------------------------------------------------
--
-------------------------------------------------------------------------------
procedure Data_into_SS_pipe IS
begin
    CS_Buff.match.state := MATCH_BUSY;
    ss_avail_v := TRUE;
end Data_into_SS_pipe;

-------------------------------------------------------------------------------
--
-------------------------------------------------------------------------------
procedure Fast_match IS
begin

   periodic := TRUE;                -- Setup Issue Packet.
end Fast_match;
```


Microchip Ex. 1018, p. 591
Microchip v. Aptiv
IPR2024-00494

Microchip Ex. 1018, p. 592
Microchip v. Aptiv
IPR2024-00494

Appendix C
Reset Protocol State Diagrams

This appendix presents state diagrams that provide implementation examples for the reset protocol as described in Section 7.1.7.5. These state diagrams should be considered as an example to guide implementers; the description of the reset protocol and the high-speed reset handshake in Section 7.1.7.5 is the complete required behavior. By necessity, state diagrams incorporate some implementation dependent parts that, although describing the reset protocol correctly, can also be implemented in a different way yielding similar behavior.

Any timer used in these state diagrams should have a resolution that allows it to always keep to the allowed time frame. For instance, if a timer times out between a time T_{TIMER}(min) and T_{TIMER}(max), the timer should have a minimal resolution of at least 1 clocktick in the range of T_{TIMER}. In a number of places, a time T_{TIMER} is mentioned in a state diagram; while in the tables in Section 7.3, a range is given for this time. In that case, the time represents a chosen value in the range such that it is at least 1 clocktick of the associated timer away from the upper boundary of that range. Under these conditions, a state in the state diagrams will never miss a branch because the associated timer overstepped the time-out condition.

In the state diagrams in this appendix, a timer can be either Run, Started, or Cleared. If a timer is Run, it will update itself every clocktick. If a timer is Cleared, it is stopped and its contents are reset to zero. A timer that is Started is first cleared and then immediately run. Stopping of a timer is never done explicitly in the state diagrams.

C.1 Downstream Facing Port State Diagram

This section describes the reset protocol state diagram for the downstream facing port.

The state diagram shown in Figure C-1 shows all the necessary and required behavior of a downstream facing port in case of a reset. As this is the initiating party in the reset protocol, the hub enters the Resetting state through a request from the host (the SetPortFeature(PORT_RESET) command). The downstream facing port then drives an SE0 to initiate the reset and at the same time starts a timer T0 to time the whole reset procedure.

If the attached device is low-speed, then the only way that reset ends is when the timer T0 times out (T_{DRST}) and the bus returns to idle. Whether a device is low-speed is determined prior to entering the Resetting state in the status bit PORT_LOW_SPEED. This is described in more detail in Section 11.8.2. When reset has completed, the hub enters the low-speed Enabled state.

If the attached device is full-speed and not high-speed capable, it will end reset when timer T0 expires (T_{DRST}) and the hub has not detected a valid upstream chirp (continuous Chirp K). It will then enter the full-speed enabled state.

Last, if the attached device is high-speed capable, it will send back an upstream chirp some time after the SE0 has been asserted on the bus. The actual time before the upstream chirp starts depends on whether the attached device was suspended or awake at the time the reset started. The loop between the blocks with “Clear timer T1” and “Run timer T1” represents the 2.5 µs (T_{FILT}) filtering the reset protocol asks for.

Note: The timer T1 is required to be reset after an interruption of 16 high-speed bit-times of the continuous Chirp K that makes up the upstream chirp. It may be reset by any shorter interruption.

If the filtering of the upstream chirp takes too much time, the downstream facing port may not be able to finish its downstream chirp in time to be able to end the reset procedure in time. Therefore, when timer T0 reaches beyond the time T_{UCHEND} (time to detect an upstream chirp), the hub is put in a wait state, which it leaves after the timer has timed out the complete reset protocol (T_{DRST}). It will then enter the full-speed enabled state.

Microchip Ex. 1018, p. 593
Microchip v. Aptiv
IPR2024-00494

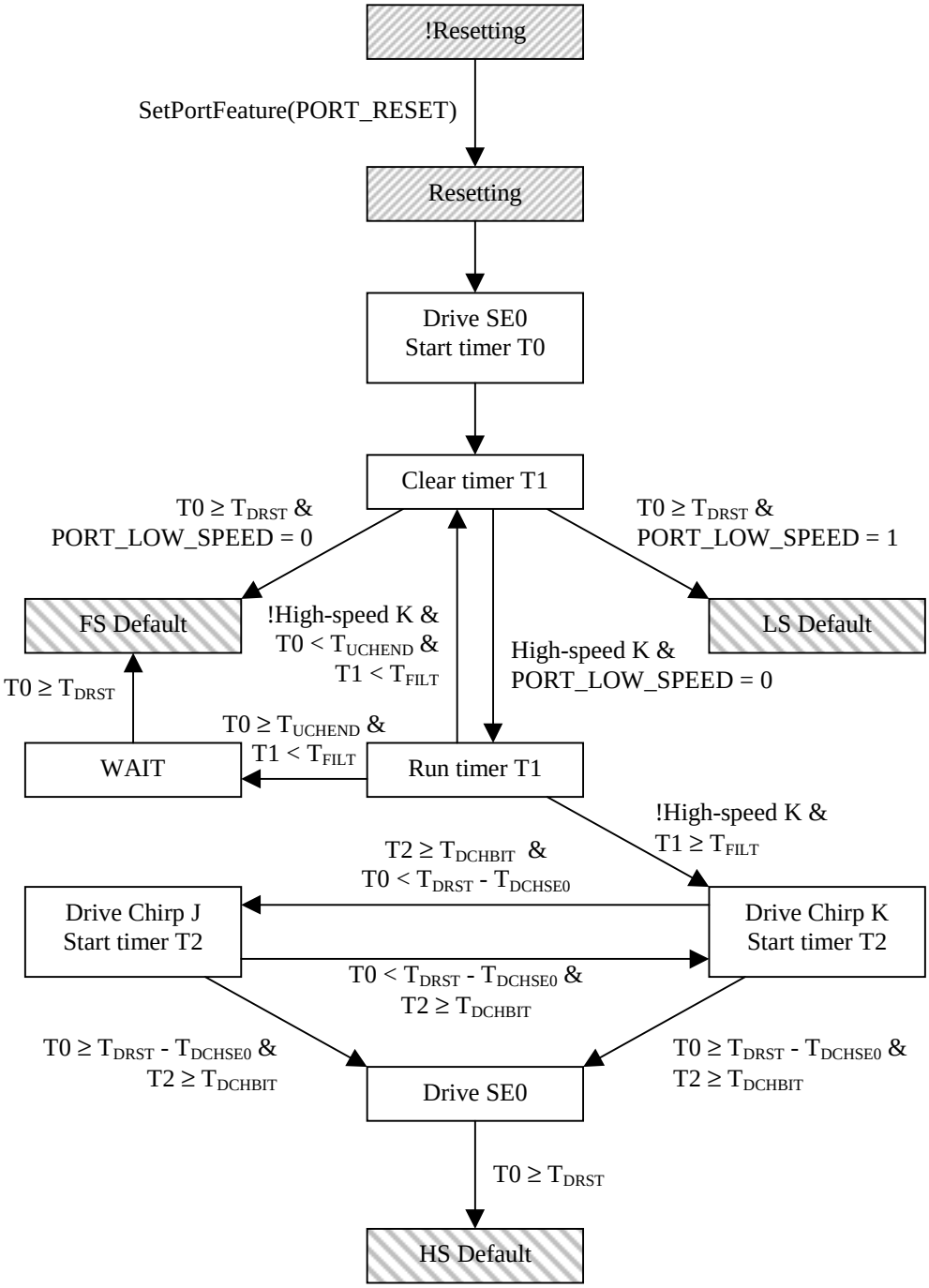


Figure C-1. Downstream Facing Port Reset Protocol State Diagram

When the downstream-facing port has successfully detected an upstream chirp, it will start transmitting the downstream chirp as soon as it has seen the bus leave the Chirp K state. This end of the upstream chirp will return the bus to the SE0 state. So immediately (actually within 100 µs (T_{WTDCH}) after the end of the upstream chirp according to Section 7.1.7.5), the hub drives a Chirp K for 40 to 60 µs (T_{DCHBIT}), then a Chirp J for 40 to 60 µs, then a Chirp K, etc. It continues with this alternating sequence until timer T0 has come within 100 to 500 µs (T_{DCHSE0}) of the end of reset (T_{DRST}). When this time is reached, the downstream-facing port finishes the

Microchip Ex. 1018, p. 594
Microchip v. Aptiv
IPR2024-00494

40 to 60 µs of continuous signaling it was busy with when the timer T0 exceeds the value of $T_{DRST} - T_{DCHSE0}$ before driving SE0 until the end of reset.

C.2 Upstream Facing Port State Diagram

This section describes the reset protocol state diagrams for the upstream facing port. The state diagram for the upstream facing port is more complicated than the diagram for the downstream facing port as the device can be in any possible state when it receives a reset signal. Therefore, the state diagram has been split into two parts:

- The reset detection state diagram which describes the way a device reacts to reset signaling on its upstream facing port (see Figure C-2)
- The reset handshake state diagram that explains how a high-speed capable device performs a handshake procedure with the hub upstream to communicate each others high-speed capabilities and have both enter a high-speed state at the end of reset (see Figure C-3)

Therefore, all of these states must be covered in the diagram. Also, the fact that for a high-speed capable device a suspend is initially indistinguishable from a reset requires that the state diagram for the upstream facing port addresses the suspend procedure as well.

At the start of the reset, we can be any possible state, but we can collect them into three groups, where each group is handled differently, but all states in the same group handle reset in the same way. The states are as follows:

- Suspended
- Powered, FS Default, FS Address, and FS Configured
- HS Default, HS Address, and HS Configured

These groups of states correspond to an identical list of possibilities as described in Section 7.1.7.5 under item 3 of the reset protocol.

C.2.1 Reset From Suspended State

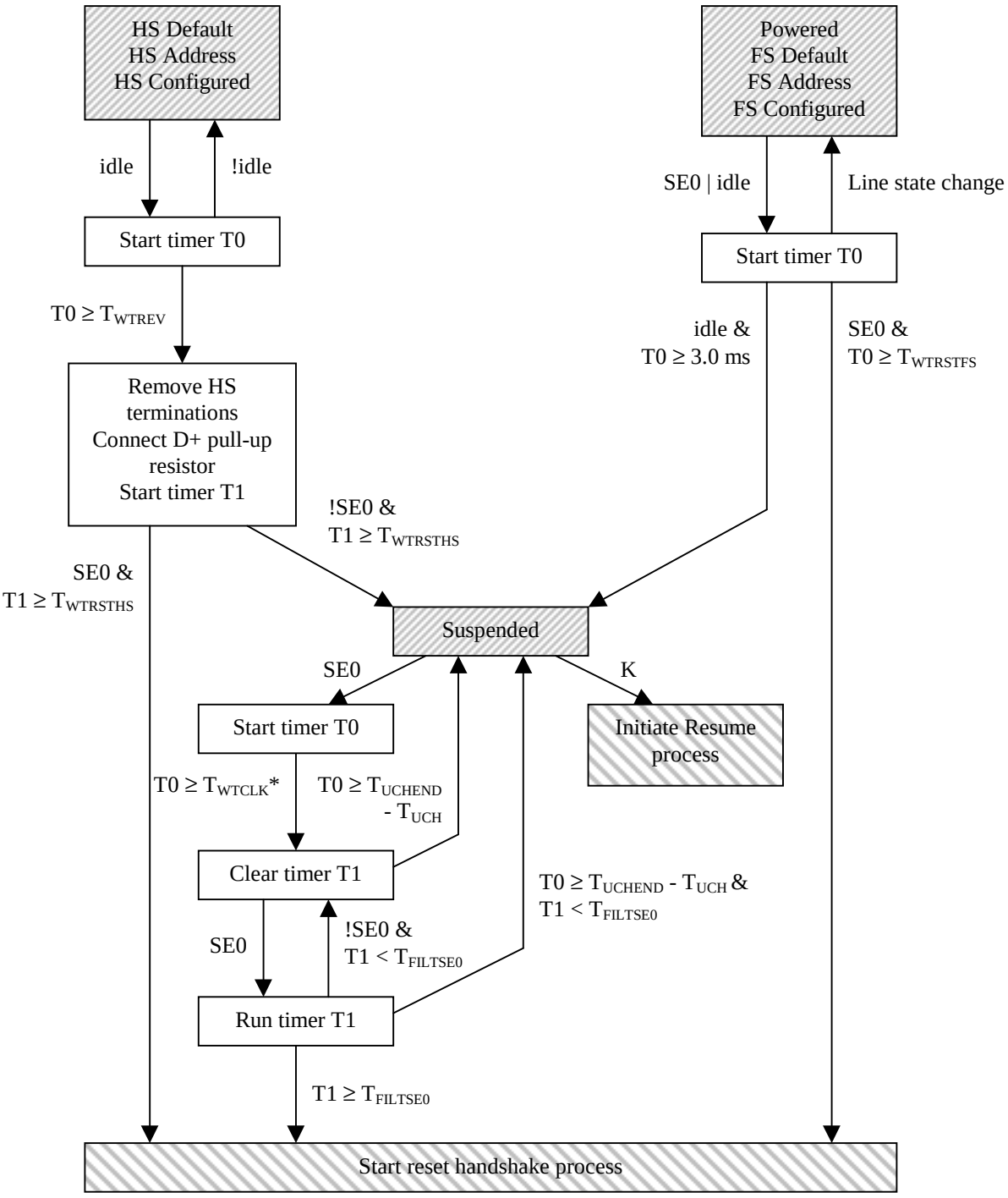
As can be seen from Figure C-2, the device wakes up from the Suspended state as soon as it sees a K or an SE0 on the bus. A J would be indistinguishable from idle on the bus that a suspended device sees normally. On seeing a K, the device will initiate a resume process. For the details of this process, see Section 7.1.7.7. On seeing an SE0, the device could enter the reset handshake procedure, so it starts timer T0.

The actual reset handshake is only started after seeing a continuous assertion of SE0 for at least 2.5 µs ($T_{FILTSE0}$). The loop between the blocks with “Clear timer T1” and “Run timer T1” represents this filtering. If the device has not detected a continuous SE0 before timer T0 exceeds the value of $T_{UCHEND} - T_{UCH}$, the device goes back into the Suspended state.

A device coming from suspend most probably had its high-speed clock stopped to meet the power requirements for a suspended device (see Section 7.2.3). Therefore, it may take some time to let the clock settle to a level of operation where it is able to perform the reset detection and handshake with enough precision. In the state diagram, a time symbol T_{WTCLK} is used to have the device wait for a stable clock. This symbol is not part of the USB 2.0 specification and does not appear in Chapter 7. It is an implementation specific detail of the reset detection state diagram for the upstream facing port, where it is marked with a asterisk (*). T_{WTCLK} should have a value somewhere between 0 and 5.0 ms. This allows at least 1.0 ms time to detect the continuous SE0.

If the device has seen an SE0 signal on the bus for at least $T_{FILTSE0}$, then it can safely assume to have detected a reset and can start the reset handshake.

Microchip Ex. 1018, p. 595
Microchip v. Aptiv
IPR2024-00494



(*) **Note:** T_{WTCLK} is a symbol that is only used in this state diagram. It is not part of the USB 2.0 specification and does not appear in Chapter 7. It is an implementation specific detail of this state diagram. See Section C.2.1 for a detailed description.

Figure C-2. Upstream Facing Port Reset Detection State Diagram

Microchip Ex. 1018, p. 596
Microchip v. Aptiv
IPR2024-00494

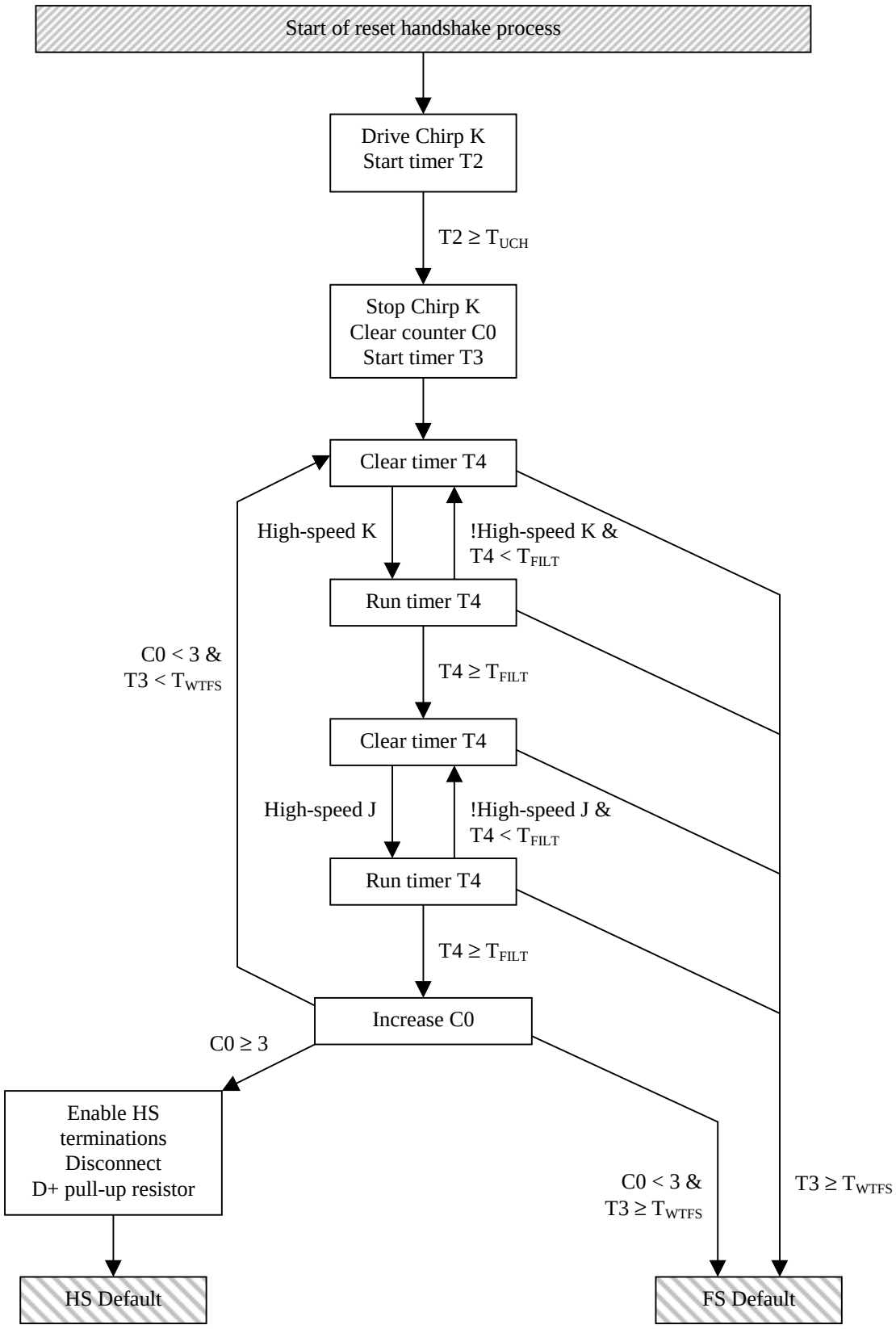


Figure C-3. Upstream Facing Port Reset Handshake State Diagram

Microchip Ex. 1018, p. 597
Microchip v. Aptiv
IPR2024-00494

C.2.2 Reset From Full-speed Non-suspended State

Timer T0 is started when seeing an SE0 or idle state from a full-speed Non-suspended state.

If a J (idle) is detected and the timer T0 exceeds the value of 3.0 ms while no change has been detected in the state of the bus, the device is suspended.

If an SE0 is detected and the timer T0 times out the value of $T_{WTRSTFS}$ (between 2.5 µs minimum and 3.0 ms maximum) while no change has been detected in the SE0 state, the device can start the reset handshake. On any line state change, the device aborts the detection of reset or suspend from upstream and returns to its previous state.

C.2.3 Reset From High-speed Non-suspended State

Timer T0 is started when seeing a high-speed idle on the bus from a high-speed Non-suspended state. If anything else than idle is detected on the bus, the device aborts detection of a reset and returns to its previous state. When timer T0 exceeds the value of T_{WTREV} (between 3.0 ms minimally and 3.125 ms maximally), the device reverts to full-speed by switching off its high-speed terminations and connecting the D+ pull-up resistor to the D+ line.

The reset protocol allows some time for debouncing and settling of the lines in the new state ($T_{WTRSTHS}$). After this time, the line should be sampled to see whether the device should be suspended (on detecting a full-speed idle) or reset (on detecting SE0).

If an idle was detected, the device should suspend; if an SE0 was detected, the device can start the reset handshake.

If something other than an idle or an SE0, in other words, a K, was detected, the device will also enter the suspended state. However, on seeing the K, the device will immediately resume, effectively returning to the high-speed state.

C.2.4 Reset Handshake

At this point, the behavior of devices has become independent of the initial state they were in when the reset started. The reset handshake is started by the device, when it sends an upstream chirp that is at least 1.0 ms long and stops before the timer T0 hits the 7.0 ms mark. Note: This is the same timer T0 that was started in the reset detection state diagram in Figure C-2.

A choice of implementation is available here. The one presented in the state diagram in Figure C-3 is where a timer T2 is started when the Chirp K is asserted to time the minimum required duration of the upstream chirp. The Chirp K is stopped when timer T2 exceeds the value of T_{UCH}. Another approach would be to wait until the timer T0 exceeds the value of T_{UCHEND}, before ending the upstream chirp. Both conform to the requirements of the reset protocol in Section 7.1.7.5, and the choice may depend on the particular application.

As soon as the upstream chirp has ended, the device starts listening for the downstream chirp. In order to detect at least a K-J-K-J-K-J pattern, it first starts looking for a continuously asserted Chirp K. The method employed in this state diagram is counting the number of K-J transitions. Here K and J are actually Chirp K and Chirp J, respectively, asserted continuously for at least 2.5 µs (T_{FILT}).

Continuous assertion is determined by the loop between the “Clear timer T4” and “Run timer T4”. This is similar to the method used in the downstream facing port state diagram in Figure C-1 to detect the upstream chirp. After this, a continuous Chirp J is detected in the same manner, most likely, even using the same hardware. Now we have detected one K-J transition. Until we have detected three K-J transitions in the same way, we will not revert to high-speed.

The whole procedure of detecting the downstream chirp is timed by timer T3 which requires the device to perform the detection of the K-J-K-J-K-J for at least 1.0 ms, but at most 2.5 ms. If the device is unable to detect a sufficient number of K-J transitions before the timer T3 times out at T_{WTFS}, the device enters the full-speed default state. Reset ends when the bus state changes from SE0 to idle. The time T_{WTFS} is given a wide range to

Microchip Ex. 1018, p. 598
Microchip v. Aptiv
IPR2024-00494

allow sufficient leverage for a device which has awoke from suspend to use its (possible not yet stable) clock to time this duration reliably.

Reversion to high-speed when the device has detected the K-J-K-J-K-J pattern is accomplished by enabling the high-speed terminations and disconnecting the pull-up resistor from the D+-line. According to Section 7.1.7.5, you may wait up to 500 µs before actually reverting to high-speed, but in this state diagram, this reversion is done immediately after detection of three K-J transitions. After this switching of terminations and pull-up, the device enters the high-speed Default state. The end of reset is signified by the first packet that is received, most likely an SOF packet.

Microchip Ex. 1018, p. 599
Microchip v. Aptiv
IPR2024-00494

Microchip Ex. 1018, p. 600
Microchip v. Aptiv
IPR2024-00494

Index

0th microframe, 9.4.11, 11.14.2.3, 11.18.3, 11.22.2
"3 strikes and you're out" mechanism, 11.17.1
4X over-sampling state machine DPLLs, 7.1.15.1

A

abnormal termination sequences, 11.3.3
aborting/retiring transfers
  aborting control transfers, 5.5.5
  after loss of synchronization, 11.22.2
  client role in, 10.5.2.2
  conditions for, 5.3.2
  message pipes and, 5.3.2.2
  packet size and, 5.5.3
  Transaction Translator's role, 11.18.6, 11.18.6.1
  USBDI role, 10.5.3.2.1
access frequency of control pipes, 5.5.4
Acknowledge packet. *See* ACKs
ACKs, 8.3.1 *Table 8-1*
  in bulk transfers, 8.5.2, 11.17.1
  in control transfers, 8.5.3, 8.5.3.1, 11.17.1
  corrupted ACK handshake, 8.5.3.3, 8.6.4
  in data toggle, 8.6, 8.6.1, 8.6.2
  defined, 2.0 *glossary*
  function response to OUT transactions, 8.4.6.3
  host response to IN transactions, 8.4.6.2
  overview, 8.4.5
  PING flow control and OUT transactions, 8.5.1, 8.5.1.1
  Ready/ACK status, 11.15
  in request processing, 9.2.6
AC loading specifications, 7.1.6.2
A connectors. *See* Series "A" and "B" connectors
AC stress evaluative setup, 7.1.1
actions in state machines, 8.5, 11.15
active devices, defined, 2.0 *glossary*
active pipes, 10.5.2.2
adaptive endpoints
  connection requirements, 5.12.4.4
  feedback for isochronous transfers, 5.12.4.2
  overview, 5.12.4.1.3
adding devices. *See* dynamic insertion and removal
Address device state
  bus enumeration process, 9.1.2
  overview, 9.1.1.4
  standard device requests, 9.4.1 to 9.4.11
  visible device state table, 9.1.1 *Table 9-1*
addresses
  Address device state, 9.1.1.4, 9.1.1 *Table 9-1,* 9.1.2, 9.4.1 to 9.4.11
  aliasing, 8.3.2
  assignment
    after dynamic insertion or removal, 4.6.3
    bus enumeration, 2.0 *glossary,* 4.6.3, 9.1.2
    device initialization, 10.5.1.1
    operations overview, 9.2.2
    re-enumerating sub-trees, 10.5.4.5
    staged power switching in functions and, 7.2.1.4
    time limits for completing, 9.2.6.3
    USB System Software role, 4.9
  endpoint addresses, 5.3.1, 9.6.6
  SetAddress() request, 9.4.6
address fields
  address field (ADDR), 8.3.2.1, 8.3.5.1, 8.4.1, 8.4.2.2
  endpoint field (ENDP), 8.3.2.2, 8.3.5.1, 8.4.1
  Hub address field, 8.4.2.2
  packet address fields, 8.3.2 to 8.3.2.2
ADDR field
  overview, 8.3.2.1
  token CRCs, 8.3.5.1
  in token packets, 8.4.1
Adopters Agreement, 1.4
advancing pipeline pseudocode, 11.18.7
aging, data-rate inaccuracies and, 7.1.11
aliasing addresses, 8.3.2
"all" encoding, 11.18.4
allocating bit times in handshake packets, 11.3.3
allocating buffers. *See* buffers
allocating USB bandwidth
  transfer management, 5.11.1 to 5.11.1.5
  USB System role, 10.3.2
alternate settings for interfaces
  configuration requirements, 10.3.1
  GetInterface() request, 9.4.4
  in interface descriptors, 9.6.5
  SetInterface() request, 9.4.10
  USBDI mechanisms, 10.5.2.10
  USB support for, 9.2.3
American National Standard/Electronic Industries Association, 6.7.1
American Standard Test Materials, 6.7.1
ANSI/EIA-364-C (12/94), 6.7.1
applications
  in source-to-sink connectivity, 5.12.4.4
  USB suitability for, 3.3

Microchip Ex. 1018, p. 601
Microchip v. Aptiv
IPR2024-00494

architectural overview of USB
architectural extensions, 4.10
bus protocol, 4.4
bus topology, 4.1.1
data flow types, 4.7 to 4.7.5
hub architecture, 4.8.2.1, 11.1.1, 11.12.2
mechanical and electrical specifications, 4.2 to 4.2.2, 6.1
physical interface, 4.2 to 4.2.2
power, 4.3 to 4.3.2
robustness and error handling, 4.5 to 4.5.2
system configuration, 4.6 to 4.6.3
USB devices, 4.1.1.2, 4.8 to 4.8.2.2
USB host, 4.1.1.1, 4.9
USB system description, 4.1 to 4.1.1.2
assigning addresses. *See* addresses; bus enumeration
ASTM-D-4565, 6.6.3, 6.7.1
ASTM-D-4566, 6.6.3, 6.7.1
asynchronous data transfers, 2.0 *glossary,* 4.9
asynchronous endpoints
connection requirements, 5.12.4.4
feedback for isochronous transfers, 5.12.4.2
overview, 5.12.4.1.1
asynchronous RA, 2.0 *glossary,* 5.12.4.4. *See also* RA (rate adaptation)
asynchronous SRC, 2.0 *glossary*. *See also* SRC
Attached device state
in bus enumeration process, 9.1.2
overview, 9.1.1.1
visible device state table, 9.1.1 *Table 9-1*
attaching devices. *See* dynamic insertion and removal
attenuation, 7.1.17
attributes of devices in configuration descriptors, 9.6.3
attributes of endpoints in endpoint descriptors, 9.6.6
audio connectivity, 5.12.4.4.1
Audio Device Class Specification Revision 1.0, 9.6
audio devices, defined, 2.0 *glossary*
automatic port color indicators, 11.5.3
available time in frames and microframes
bulk transfers and, 5.8.4
bus bandwidth reclamation, 5.11.5
control transfers and, 5.5.4
interrupt transfer bus access constraints, 5.7.4
isochronous transfers and, 5.6, 5.6.4
AWG, 2.0 *glossary,* 6.6.2

B

babble
Collision conditions and detection, 11.8.3
defined, 2.0 *glossary*
EOF2 timing points and, 11.2.5
babble *(continued)*
EOF and babble detection, 11.2.5.1
error detection and recovery, 8.7.4
transaction tracking and, 11.18.7
background of USB development, 3.1 to 3.3
backwards compatibility of USB 2.0, 3.1
bAlternateSetting field (interface descriptors), 9.6.5, 11.23.1
bandwidth
allocating for pipes, 4.4, 4.7.5
bandwidth reclamation, 5.11.5
defined, 2.0 *glossary*
transfer management, 4.7.5, 5.11.1 to 5.11.1.5, 10.3.2
USB system role in, 10.3.2
battery-powered hubs, 7.2.1
bcdDevice field (device descriptors), 9.6.1
bcdUSB field (device descriptors), 9.2.6.6, 9.6.1, 11.23.1
bcdUSB field (device qualifier descriptors), 9.6.2, 11.23.1
bConfigurationValue field
configuration descriptors, 9.6.3, 11.23.1
other speed configuration descriptors, 9.6.4, 11.23.1
B connectors. *See* Series "A" and "B" connectors
bDescLength field (hub descriptors), 11.23.2.1
bDescriptorType field
configuration descriptors, 9.6.3, 11.23.1
device descriptors, 9.6.1, 11.23.1
device qualifier descriptors, 9.6.2, 11.23.1
endpoint descriptors, 9.6.6, 11.23.1
hub descriptors, 11.23.2.1, 11.24.2.5, 11.24.2.10
interface descriptors, 9.6.5, 11.23.1
other speed configuration descriptors, 9.6.4, 11.23.1
string descriptors, 9.6.7
bDeviceClass field
device descriptors, 9.6.1, 11.23.1
device qualifier descriptors, 9.6.2, 11.23.1
bDeviceProtocol field
device descriptors, 9.6.1, 11.23.1
device qualifier descriptors, 9.6.2, 11.23.1
bDeviceSubClass field
device descriptors, 9.6.1, 11.23.1
device qualifier descriptors, 9.6.2, 11.23.1
"beginning" encoding, 11.18.4
bEndpointAddress field (endpoint descriptors), 9.6.6, 11.23.1
best case full-speed budgets, 11.18.1, 11.18.4
bHubContrCurrent field (hub descriptors), 11.23.2.1
bi-directional communication flow, 5.6.2, 5.8.2
big endian, defined, 2.0 *glossary*

Microchip Ex. 1018, p. 602
Microchip v. Aptiv
IPR2024-00494

bInterfaceClass field (interface descriptors), 9.6.5, 11.23.1
bInterfaceNumber field (interface descriptors), 9.6.5, 11.23.1
bInterfaceProtocol field (interface descriptors), 9.6.5, 11.23.1
bInterfaceSubClass field (interface descriptors), 9.6.5, 11.23.1
bInterval field (endpoint descriptors), 9.6.6, 11.23.1
bit cells, decoding, 7.1.15.1
bitmaps of hub and port status changes, 11.12.4
bit ordering, 8.1
bits, defined, 2.0 *glossary*
bit stuffing
  bit stuffing errors, 11.3.3, 11.15, 11.22
  bit stuff violations, 8.7.1
  calculating transaction times, 5.11.3
  defined, 2.0 *glossary*
  high-speed signaling and, 7.1
  microframe pipeline and, 11.18.2
  overview, 7.1.9
bit times
  bit time designations, 11.3
  bit time zero, 11.3
  before EOF, 11.2.5
  in transaction completion prediction, 11.3.3
bLength field
  configuration descriptors, 9.6.3, 11.23.1
  device descriptors, 9.6.1, 11.23.1
  device qualifier descriptors, 9.6.2, 11.23.1
  endpoint descriptors, 9.6.6, 11.23.1
  interface descriptors, 9.6.5, 11.23.1
  other speed configuration descriptors, 9.6.4, 11.23.1
  string descriptors, 9.6.7
blinking indicators. *See* indicators
blocking packets in Collision conditions, 11.8.3
blunt cut termination, 6.4.2, 6.4.3
bmAttributes field
  configuration descriptors, 9.6.3, 11.23.1
  endpoint descriptors, 9.6.6, 11.23.1
  hub descriptors, 11.13
  other speed configuration descriptors, 9.6.4, 11.23.1
bMaxPacketSize0 field
  device descriptors, 9.6.1, 11.23.1
  device qualifier descriptors, 9.6.2, 11.23.1
bMaxPower field, 9.6.3
  configuration descriptors, 11.23.1
  other speed configuration descriptors, 9.6.4, 11.23.1
bmRequestType field
  hub class requests, 11.24.2
  overview, 9.3.1
  Setup data format, 9.3
bmRequestType field *(continued)*
  standard device requests, 9.4
bNbrPorts field (hub descriptors), 11.23.2.1
bNumConfigurations field
  device descriptors, 9.6.1, 11.23.1
  device qualifier descriptors, 9.6.2, 11.23.1
bNumEndpoints field (interface descriptors), 9.6.5, 11.23.1
bNumInterfaces field
  configuration descriptors, 9.6.3, 11.23.1
  other speed configuration descriptors, 9.6.4, 11.23.1
bPwrOn2PwrGood field, 11.11, 11.23.2.1
bRequest field
  hub class requests, 11.24.2
  overview, 9.3.2
  Setup data format, 9.3
  standard device requests, 9.4
  standard hub requests, 11.24.1
bReserved field (device qualifier descriptor), 9.6.2
broadcast mode of hub operation, 11.1.2.1
B/S or b/S, defined, 2.0 *glossary*
bString field (string descriptors), 9.6.7
budgets, best case full-speed budget, 11.18.1, 11.18.4
buffers
  buffer impedance, 7.1.1.1
  buffer match tests, 11.17.1
  bulk/control transfer buffering requirements, 11.17.4
  calculating sizes in functions and software, 5.11.4
  clearing, 11.17.5, 11.24.2.3
  client pipes and, 10.5.1.2.2
  client role in, 10.3.3, 10.5.3
  defined, 2.0 *glossary*
  elasticity buffer, 11.7.1.3
  endpoint buffer size, 4.4
  identifying location and length, 10.3.4
  interrupt transfers and, 5.7.3
  isochronous transfers and, 5.12.4.2
  non-periodic transaction buffers, 11.14.1, 11.14.2.2, 11.17, 11.17.4
  non-USB isochronous application, 5.12.1
  packet buffers, 2.0 *glossary*
  periodic transaction buffers, 11.14.2.1
  prebuffering data, 5.12.5
  rate matching and, 5.12.8
  rise and fall times for full-speed buffers, 7.1.2.1

Microchip Ex. 1018, p. 603
Microchip v. Aptiv
IPR2024-00494

buffers *(continued)*
Transaction Translator buffers
overview, 11.14.1
resetting, 11.24.2.9
space required, 11.19
underrun or overrun states and error counts, 10.2.6
USBD role in allocating, 10.5.1.2.1
bulk transfers. *See also* non-periodic transactions
buffering requirements, 11.14.2.2, 11.17.4
bus access constraints, 5.8.4
data format, 5.8.1
data sequences, 5.8.5
defined, 2.0 *glossary,* 5.4
direction, 5.8.2
failures, 11.17.5
NAK rates for endpoints, 9.6.6
non-periodic transactions, 11.17 to 11.17.5
overview, 4.7.2, 5.8
packet size, 5.8.3, 9.6.6
scheduling, 11.14.2.2
split transaction examples, A.1, A.2
split transaction notation for, 11.15
state machines, 8.5.1, 8.5.1.1, 8.5.2, 11.17.2
transaction format, 8.5.2
transaction organization within IRPs, 5.11.2
USBD pipe mechanism responsibilities, 10.5.3.1.3
bus access for transfers
bulk transfer constraints, 5.8.4
bus access periods, 5.12.8
bus bandwidth reclamation, 5.11.5
calculating buffer sizes, 5.11.4
calculating bus transaction times, 5.11.3
client software role in, 5.11.1.1
control transfer constraints, 5.5.4
HCD role in, 5.11.1.3
Host Controller role in, 5.11.1.5
interrupt transfer constraints, 5.7.4
isochronous transfer constraints, 5.6.4
transaction list, 5.11.1.4
transaction tracking, 5.11.2
transfer management, 5.1.1 to 5.11.1.5
transfer type overview, 5.4
USBD role in, 5.11.1.2
bus clock, 5.12.2, 5.12.3, 5.12.8
bus enumeration
defined, 2.0 *glossary*
device initialization, 10.5.1.1
enumeration handling, 11.12.6
overview, 4.6.3, 9.1.2
re-enumerating sub-trees, 10.5.4.5
staged power switching in functions, 7.2.1.4
USB System Software role, 4.9
bus-powered devices and functions
configuration descriptors, 9.6.3
defined, 4.3.1
device states, 9.1.1.2
high-power bus-powered functions, 7.2.1.4
low-power bus-powered functions, 7.2.1.3
power budgeting, 9.2.5.1
bus-powered hubs
configuration, 11.13
defined, 4.3.1, 7.2.1
device states, 9.1.1.2
overview, 7.2.1.1
power switching, 11.11
voltage drop budget, 7.2.2
bus protocol overview, 4.4
Bus_Reset receiver state, 11.6.3, 11.6.3.9
bus states
evaluating after reset, 7.1.7.3
global suspend, 7.1.7.6.1
Host Controller role in state handling, 10.2.1
signaling levels and, 7.1.7.1, 7.1.7.2
Transaction Translator tracking, 11.14.1
bus timing/electrical characteristics, 7.3.2
bus topology, 5.2 to 5.2.5
client-software-to-function relationship, 5.2.5
defined, 4.1
devices, 5.2.2
hosts, 5.2.1
illustrated, 4.1.1
logical bus topology, 5.2.4
physical bus topology, 5.2.3
bus transaction timeout in isochronous transfers, 5.12.7
bus turn-around time, 2.0 glossary, 7.1.18 to 7.1.18.2, 8.7.2, 11.18.2
busy (ready/x) state, 11.17.5
bypass capacitors, 7.2.4.1, 7.2.4.2
bytes, defined, 2.0 *glossary*

C

cable assemblies, 6.4 to 6.4.4
cable attenuation, 7.1.17
cable delay
electrical characteristics, 7.3.2 *Table 7-12*
high-/full-speed cables, 6.4.2
hub differential delay, differential jitter, and SOP distortion, 7.3.3 *Figure 7-52*
hub EOP delay and EOP skew, 7.3.3 *Figure 7-53*
hub signaling timings, 7.1.14.1
inter-packet delay and, 7.1.18.1
low-speed cables, 6.4.3, 7.1.1.2
overview, 7.1.16
propagation delay, 6.4.1, 6.7 *Table 6-7,* 7.1.1.2

Microchip Ex. 1018, p. 604
Microchip v. Aptiv
IPR2024-00494

cable delay *(continued)*
skew delay, 6.7 *Table 6-7,* 7.1.3, 7.3.3 *Figure 7-53*
cables
attenuation, 7.1.17
cable assemblies, 6.4 to 6.4.4
cable delay (*See* cable delay)
captive cables
high-/full-speed captive cable assemblies, 6.4.2
inter-packet delay and, 7.1.18.1
low-speed captive cable assemblies, 6.4.3
maximum capacitance, 7.1.6.1
termination, 7.1.5.1
color choices, 6.4
construction, 6.6.2
description, 6.6.1
detachable cables
cable delay, 7.1.16
connectors and, 6.2
detachable cable assemblies, 6.4.1
inter-packet delay and, 7.1.18.1
low-speed detachable cables, 6.4.4
maximum capacitance, 7.1.6.1
termination, 7.1.5.1
voltage drop budget, 7.2.2
electrical characteristics and standards, 4.2.1, 6.6.3, 6.7, 7.3.2 *Table 7-12*
end-to-end signal delay, 7.1.19.1
environmental characteristics, 6.6.4, 6.7
flyback voltage, 7.2.4.2
high-/full-speed cables, 6.4.2
impedance, 6.4.1, 6.4.2, 6.7 *Table 6-7*
input capacitance, 7.1.6.1
length, 6.4.1, 6.4.2, 6.4.3
listing, 6.6.5
low-speed cables, 6.4.3, 6.4.4, 7.1.1.2
mechanical configuration and material requirements, 6.6 to 6.6.5, 6.7
overview, 6.3
prohibited cable assemblies, 6.4.4
pull-out standards, 6.7 *Table* 6-7
shielding, 6.6, 6.6.1
termination, 7.1.5.1
voltage drop budget, 7.2.2
calculations
buffering for rate matching, 5.12.8
buffer sizes in functions and software, 5.11.4
bus transaction times, 5.11.3
capabilities, defined, 2.0 *glossary*
capacitance
after dynamic attach, 7.2.4.1
decoupling capacitance, 7.3.2 *Table* 7-7
input capacitance, 7.1.6.1, 7.3.2 *Table* 7-7
low-speed buffers, 7.1.1.2, 7.1.2.1
low-speed cable capacitive loads, 6.4.3
capacitance *(continued)*
lumped capacitance guidelines for transceivers, 7.1.6.2
optional edge rate control capacitors, 7.1.6.1
pull-up resistors and, 7.1.5.1
single-ended capacitance, 7.1.1.2
small capacitors, 7.1.6.1
target maximum droop and, 7.2.4.1
unmated contact capacitance, 7.3.2 *Table* 7-12
capacitive load, 6.7 *Table 6-7*
captive cables
high-/full-speed captive cable assemblies, 6.4.2
inter-packet delay and, 7.1.18.1
low-speed captive cable assemblies, 6.4.3
maximum capacitance, 7.1.6.1
rise and fall times, 7.1.2.1, 7.1.2.2
TDR measurements and, 7.1.6.2
termination, 7.1.5.1
change bits
device states, 11.12.2
hub and port status change bitmap, 11.12.4
hub status, 11.24.2.6
over-current status change bits, 11.12.5
port status change bits, 11.24.2.7.2 to 11.24.2.7.2.5
Status Change endpoint defined, 11.12.1
change propagation, host state handling of, 10.2.1
characteristics of devices, 2.0 *glossary,* 9.6.3, 9.6.4
Chirp J and K bus states, 7.1.4.2, 7.1.7.2, 7.1.7.5, C.1, C.2.4
C_HUB_LOCAL_POWER, 11.11, 11.24.2, 11.24.2.1, 11.24.2.6, 11.24.2.7.1.6
C_HUB_OVER_CURRENT, 11.24.2, 11.24.2.1
C_HUB_OVER_POWER, 11.24.2.6
classes of devices. *See* device classes
Class field, 9.2.3, 9.6.5
class-specific descriptors, 9.5, 11.23.2.1
class-specific requests
hub class-specific requests, 11.24.2 to 11.24.2.13
time limits for completing, 9.2.6.5
USBDI mechanisms, 10.5.2.8
Cleared timer status, C.0
ClearFeature() request, CLEAR_FEATURE
ClearHubFeature() request, 11.24.2.1
ClearPortFeature() request, 11.24.2.2
endpoint status and, 9.4.5
hub class requests, 11.24.2
hub requests, 11.24.1
overview, 9.4.1
standard device request codes, 9.4

Microchip Ex. 1018, p. 605
Microchip v. Aptiv
IPR2024-00494

ClearHubFeature() request
clearing hub features, 11.24.2.6
hub class requests, 11.24.2
hub class-specific requests, 11.24.2.1
clearing pipes, 10.5.2.2
ClearPortFeature() request
clearing status change bits, 11.12.2, 11.24.2.7.2
C_PORT_CONNECTION, 11.24.2.7.2.1
C_PORT_ENABLE, 11.24.2.7.2.2
C_PORT_OVER-CURRENT, 11.24.2.7.2.4
C_PORT_RESET, 11.24.2.7.2.5
C_PORT_SUSPEND, 11.24.2.7.2.3
hub class requests, 11.24.2, 11.24.2.2
PORT_CONNECTION, 11.24.2.7.1.1
PORT_ENABLE, 11.5.1.4, 11.24.2.7.1.2
PORT_HIGH_SPEED, 11.24.2.7.1.8
PORT_INDICATOR, 11.24.2.2, 11.24.2.7.1.10
PORT_LOW_SPEED, 11.24.2.7.1.7
PORT_OVER_CURRENT, 11.24.2.7.1.4
PORT_POWER, 11.24.2.13
PORT_POWER, 11.5.1.2, 11.24.2.7.1.6
PORT_RESET, 11.24.2.7.1.5
PORT_SUSPEND, 11.5.1.10
ClearTTBuffer() request, CLEAR_TT_BUFFER
checking for busy state, 11.17.5
hub class-specific requests, 11.24.2, 11.24.2.3
client pipes, 10.5.1.2.2
client software
in bus topology, 5.2, 5.2.1, 5.2.5
client software-to-function relationships, 5.2, 5.2.5
in communication flow, 5.3
control transfers and, 5.5
defined, 2.0 *glossary*
as implementation focus area, 5.1
notification identification, 10.3.4
role in configuration, 10.3.1
role in data transfers, 10.3.3
service clock and, 5.12.2
in source-to-sink connectivity, 5.12.4.4
in transfer management, 5.11.1, 5.11.1.1
clock model
buffering for rate matching, 5.12.8
bus clock, 5.12.2
clock encoding scheme in electrical specifications overview, 4.2.1
clock synchronization, 5.12.3
clock-to-clock phase differences, 5.12.3
clock tolerance, 11.7.1.3
defined, 5.12
frame clocks, 11.18.3
hub clock source, 11.2.3
in non-USB isochronous application, 5.12.1
overview, 5.12.2
clock model *(continued)*
receive clock, 11.7.1.2, 11.7.1.3
sample clock, 5.12.2
service clock, 5.12.2
transmit clock, 11.7.1.3
using SOF tokens as clocks, 5.12.5
clock timings, 7.3.2 *Table 7-8,* 7.3.2 *Table 7-9,* 7.3.2 *Table 7-10*
CMOS driver circuit, 7.1.1.1
CMOS implementations, 7.1.1.3
codes. *See specific types of codes*
Collision conditions, 11.8.3
color choices
cables, 6.4
indicator lights on devices, 11.5.3 to 11.5.3.1
plugs, 6.5.4.1
receptacles, 6.5.3.1
commanded stalls, 8.4.5
commands. *See* requests
common mode range for differential input sensitivity, 7.1.4.1
Communication Cables (UL Subject-444), 6.6.5, 6.7.1
communication flow, 5.3 to 5.3.3
Compare_BC_buff algorithm, 11.17.1
completed operations, 9.2.6
completed transactions, 11.3.3
complete-split transactions
buffering, 11.14.2.1, 11.17
bulk/control transactions, 11.17, 11.17.1
CSPLIT transaction tokens, 8.4.2.3
defined, 11.14.1.2
isochronous transactions, 11.21
notation for, 11.15
overview, 11.14.1
scheduling, 11.14.2.1, 11.18.4
space for, 11.18.6.3
split transaction overview, 8.4.2, 8.4.2.1
TT state searching, 11.18.8
completion times for hub requests, 11.24.1
composite devices, 5.2.3
compound devices
bus-powered hubs, 7.2.1.1
in bus topology, 5.2.3
defined, 4.8.2.2
hub descriptors for, 11.23.2.1
power configuration, 11.13
self-powered hubs, 7.2.1.2
conditions in state machine transitions, 8.5, 11.15
conductor resistance unbalance, 6.6.3
conductors
mechanical specifications, 4.2.2
power and signal conductors in cables, 6.3, 6.6.2
resistance, 6.6.3

Microchip Ex. 1018, p. 606
Microchip v. Aptiv
IPR2024-00494

configuration
bus enumeration, 4.6.3, 9.1.2
configuration management, 10.5.4.1.1
Configured device state, 9.1.1.5
control transfers and, 5.5.4
descriptors, 5.3.1.1, 9.4.3, 9.5, 9.6.1 to 9.6.4, 11.23.1 (*See also* descriptors)
device attachment, 4.6.1
device configuration, 10.3.1
device removal, 4.6.2, 10.5.4.1.4
function configuration, 10.3.1
hubs, 11.13
information in device characteristics, 4.8.1
initial device configuration, 10.5.4.1.2
interrupt transfers and, 5.7.4
modifying device configuration, 10.5.4.1.3
multiple configurations, 9.6.1
multiple interfaces, 9.2.3
operations overview, 9.2.3
other-speed configurations, 9.6.2
power distribution and, 7.2.1
remote wakeup capabilities, 9.2.5.2
requests
configuration requests, 5.11.1.2
GetConfiguration() request, 9.4.2
SetConfiguration() request, 9.4.7
required configurations before usage, 10.3.1
USB configuration, 10.3.1
USBDI mechanisms for getting current settings, 10.5.2.4
USBD role in, 5.11.1.2, 10.5.4.1 to 10.5.4.1.4
Configuration = 0 signal/event, 11.5 *Table 11-5*
CONFIGURATION descriptor, 9.4 *Table 9-5*
configuration descriptors, 9.4.3, 9.6.4, 11.23.1
Configured device state
in bus enumeration process, 9.1.2
overview, 9.1.1.5
standard device requests and, 9.4.1 to 9.4.11
visible device state table, 9.1.1 *Table 9-1*
configuring software, defined, 2.0 *glossary*
Connect bus state, 7.1.7.1, 7.1.7.3
connecting devices. *See* dynamic insertion and removal
connection status, 11.24.2.7.2, 11.24.2.7.2.1
connectivity
audio connectivity, 5.12.4.4.1
hub fault recovery mechanisms, 11.1.2.3
Hub Repeater responsibilities, 11.1
hubs, 11.1, 11.1.2 to 11.1.2.3
packet signaling connectivity, 11.1.2.1
resume connectivity, 11.1.2.2
source/sink connectivity, 5.12.4.4
synchronous data connectivity, 5.12.4.4.2
tearing down, 11.2.5
connectors
input capacitance, 7.1.6.1
inrush current and, 7.2.4.1
interface and mating drawings, 6.5.3, 6.5.4
keyed connector protocol, 6.2
mechanical configuration and material requirements, 4.2.2, 6.5 to 6.5.4.3
orientation, 6.5.1
reference times, 7.1.6.2
Series "A" and Series "B" plugs, 6.5.4
Series "A" and Series "B" receptacles, 6.5.3
standards for, 6.7
termination data, 6.5.2
USB Icon, 6.5
construction, cable, 6.6.2
contact arcing, minimizing, 7.2.4.1
contact capacitance standards, 6.7 *Table 6-7*
contact current rating standards, 6.7 *Table 6-7*
contact materials, 6.5.3.3, 6.5.4.3
control endpoints, 2.0 *glossary*. *See also* control transfers
controlling hubs, defined, 7.1.7.7
control mechanisms
device states and control information, 11.12.2
Host Controller control flow management, 4.9
of USB host, 10.1.2
control pipes, 2.0 *glossary*. *See also* control transfers; message pipes; pipes
control transfers. *See also* non-periodic transactions
buffering, 11.14.2.2, 11.17.4
bus access constraints, 5.5.4
control pipes in device characteristics, 4.8.1
data format, 5.5.1
data sequences, 5.5.5
defined, 2.0 *glossary,* 5.4
device requests, 9.3
direction, 5.5.2
error handling on last data transaction, 8.5.3.3
failures, 11.17.5
full-speed limits, 5.5.4 *Table 5-2*
high-speed limits, 5.5.4 *Table 5-3*
low-speed limits, 5.5.4 *Table 5-1*
NAK rates for endpoints, 9.6.6
non-periodic transactions, 11.17 to 11.17.5
overview, 4.7.1, 5.5
packet size, 5.5.3, 9.6.6
protocol stalls, 8.4.5
reporting status results, 8.5.3.1
scheduling, 11.14.2.2
simultaneous transfers, 5.5.4
split transaction examples, A.1, A.2
split transaction notation for, 11.15
stages, 2.0 *glossary,* 5.5
STALL handshakes returned by control pipes, 8.5.3.4

Microchip Ex. 1018, p. 607
Microchip v. Aptiv
IPR2024-00494

control transfers *(continued)*
state machines, 8.5.1, 8.5.1.1, 8.5.2, 11.17.2
transaction format, 8.5.3
transaction organization within IRPs, 5.11.2
USBD pipe mechanism responsibilities, 10.5.3.1.4
variable-length data stage, 8.5.3.2
converting split transactions, 11.14.1
corrupted transfers and requests
in control transfers, 8.5.3
corrupted ACK handshake, 8.5.3.3, 8.6.4
corrupted CRCs, 10.2.6
corrupted IN tokens, 8.4.6.1
corrupted PIDs, 8.3.1
corrupted SOF packets in isochronous transfers, 5.12.6
in data toggle, 8.6.3
error detection and recovery, 8.7 to 8.7.4
function response to OUT transactions, 8.4.6.3
host response to IN transactions, 8.4.6.2
NAK or STALL handshake, 8.6.3
costs of implementation, 3.3
C_PORT_CONNECTION
clearing, 11.24.2.2
defined, 11.24.2.7.2.1
hub class feature selectors, 11.24.2
Port Change field, 11.24.2.7.2
port status changes, 11.24.2.7.1.10
SetPortFeature() request, 11.24.2.13
C_PORT_ENABLE
ClearPortFeature() request, 11.24.2.2
defined, 11.24.2.7.2.2
hub class feature selectors, 11.24.2
Port Change field, 11.24.2.7.2
SetPortFeature() request, 11.24.2.13
C_PORT_OVER_CURRENT
clearing, 11.24.2.2
defined, 11.24.2.7.2.4
hub class feature selectors, 11.24.2
over-current conditions, 11.11.1, 11.12.5
Port Change field, 11.24.2.7.2
SetPortFeature() request, 11.24.2.13
C_PORT_RESET
clearing, 11.24.2.2
defined, 11.24.2.7.2.5
hub class feature selectors, 11.24.2
Port Change field, 11.24.2.7.2
SetPortFeature() request, 11.24.2.13
C_PORT_SUSPEND
clearing, 11.24.2.2
defined, 11.24.2.7.2.3
hub class feature selectors, 11.24.2
Port Change field, 11.24.2.7.2
resume conditions and, 11.4.4
SetPortFeature() request, 11.24.2.13
CRCs
in bulk transfers, 8.5.2
corrupted CRCs, 10.2.6
CRC16 handling, 11.15, 11.18.5, 11.20.3, 11.20.4, 11.21.3, 11.21.4
CRC check failures, 11.15, 11.20.3, 11.20.4, 11.21.3, 11.21.4
in data packets, 8.3.5.2, 8.4.4
defined, 2.0 *glossary*
in error detection, 8.7.1
overview, 8.3.5
protection in isochronous transfers, 5.12.7
resending, 8.6.4
in token packets, 8.3.5.1, 8.4.1
USB robustness and, 4.5, 4.5.1
cross-over points of data lines, 7.1.13.2.1
cross-over voltage in signaling, 7.1.2.1
crystal capacitive loading, 7.1.11
CSPLIT (complete-split transactions). *See* complete-split transactions
CTI, 2.0 *glossary,* 3.1
current
current averaging profile, 7.2.3
current spikes during suspend/resume, 7.2.3
high-speed current driver, 7.1 *Table 7-1*
high-speed signaling and, 7.1.1.3
supply current, 7.3.2 *Table 7-7*
current frame in hub timing, 11.2.3.1
current limiting
bus-powered hubs, 7.2.1.1
dynamic attach and detach, 7.2.4.1
in over-current conditions, 11.12.5
power control during suspend/resume, 7.2.3
remote wakeup and, 7.2.3
self-powered functions, 7.2.1.5
cyclic redundancy check. *See* CRCs

D

D+ or D- lines
average voltage, 7.1.2.1
high-speed signaling and, 7.1, 7.1.1.3
impedance, 7.1.6.1
pull-up resistors and, 7.1
signaling levels and, 7.1.7.1
signal termination, 7.1.5.1
during signal transitions, 7.1.4.1
single-ended capacitance, 7.1.1.2
standardized contact terminating assignments, 6.5.2
test mode, 7.1.20
data
data defined, 5.12.4
data encoding/decoding, 7.1.8
data prebuffering, 5.12.5
data processing role of Host Controller, 10.2.4

Microchip Ex. 1018, p. 608
Microchip v. Aptiv
IPR2024-00494

DATA0/DATA1/DATA2 PIDs
in bulk transfers, 5.8.5, 8.5.2
comparing sequence bits, 8.6.2
in control transfers, 8.5.3
in data packets, 8.4.4
high-bandwidth transactions and, 5.9.1, 5.9.2
high-speed DATA2 PIDs, 8.3.1 *Table 8-1*
in interrupt transactions, 5.7.5, 8.5.4, 11.20.4
synchronization and, 8.6
Transaction Translator response generation, 11.18.5
data field in packets, 8.3.4, 8.4.4
data flow model. *See* transfers
data flow types. *See* transfer types
data formats. *See also specific types of transfers*
bulk transfers, 5.8.1
control transfers, 5.5.1
interrupt transfers, 5.7.1
isochronous transfers, 5.6.1, 5.12.4
overview, 5.4
Data J state. *See* J bus state
Data K bus state. *See* K bus state
data packets
bus protocol overview, 4.4
data CRCs, 8.3.5.2
in isochronous transfers, 8.5.5
packet field formats, 8.3 to 8.3.5.2
packet overview, 8.4.4
spreading over several frames, 5.5.4
data payload
bulk transfers, 5.8.3
calculating transaction times, 5.11.3
defined, 5.3.2
interrupt transfers, 5.7.3
isochronous transfers, 5.6.3
maximum sizes, 8.4.4
non-zero data payload, 5.6.3
packet size constraints, 5.5.3, 5.6.3
data phases
aborting, 11.18.6.1
transaction notation for, 11.15
data PIDs. *See* DATA0/DATA1/DATA2 PIDs; DATA0/DATA1 PIDs; MDATA PIDs
data rates
adaptive endpoints, 5.12.4.1.3
asynchronous endpoints, 5.12.4.1.1
in buffering calculations, 5.12.8
data-rate tolerance, 7.1.11
defined, 5.12.4
in electrical specifications overview, 4.2.1
feedback for isochronous transfers, 5.12.4.2
full-speed source electrical characteristics, 7.3.2 *Table 7-9*
high-speed source electrical characteristics, 7.3.2 *Table 7-8*
data rates *(continued)*
low-speed source electrical characteristics, 7.3.2 *Table 7-10*
overview, 7.1.11
sample clock and, 5.12.2
synchronous endpoints, 5.12.4.1.2
data recovery unit, 11.7.1.2
data retry indicators in control transfers, 5.5.5
data sequences
bulk transfers, 5.8.5
control transfers, 5.5.5
interrupt transfers, 5.7.5
isochronous transfers, 5.6.5
data signaling, 7.1.7.4 to 7.1.7.4.2
data signal rise and fall time. *See* rise and fall times
data source jitter, 7.1.13.1 to 7.1.13.1.2, 7.1.14.2, 7.1.15.1
data source signaling, 7.1.13 to 7.1.13.2.2
Data stage
in control transfers, 5.5, 5.5.5, 8.5.3
error handling on last data transaction, 8.5.3.3
length of data, 9.3.5
packet size constraints, 5.5.3
variable-length data stages, 8.5.3.2
data toggle
bulk transfers, 5.8.5
in bulk transfers, 8.5.2
corrupted ACK handshake, 8.6.4
data corrupted or not accepted, 8.6.3
in data packets, 8.4.4
data toggle sequencing, 8.5.5
high bandwidth transactions and, 5.9.1
initialization via SETUP token, 8.6.1
in interrupt transactions, 8.5.4
interrupt transfers and, 5.7.5
low-speed transactions, 8.6.5
overview, 8.6
successful data transactions, 8.6.2
data transfers. *See* data packets; Data stage; transfers
DC electrical characteristics, 7.3.2 *Table 7-7*
DC output voltage specifications, 7.1.6.2
DC resistance of plugs, 6.6.3
debounce intervals in connection events, 7.1.7.3
debouncing connections, 11.8.2
declarations in state machines
global declarations, B.1
Host Controller declarations, B.2
Transaction Translator declarations, B.3
decoupling capacitance, 7.3.2 *Table 7-7*
default addresses of devices, 2.0 *glossary,* 9.1.1.4, 10.5.1.1
Default bus state, 7.1.7.5

Microchip Ex. 1018, p. 609
Microchip v. Aptiv
IPR2024-00494

Default Control Pipe
in bus enumeration process, 9.1.2
in communication flow, 5.3
control transfer packet size constraints, 5.5.3
defined, 4.4, 5.3.2
endpoint zero requirements, 5.3.1.1
as message pipe, 5.3.2.2
size description in descriptors, 9.6.1
Default device state
overview, 9.1.1.3
standard device requests and, 9.4.1 to 9.4.11
visible device state table, 9.1.1 *Table 9-1*
default pipes, 2.0 *glossary,* 10.5.1.2.1
delays. *See* cable delay; differential delay; propagation delay
delivery rates in isochronous transfers, 4.7.4
DEOP signal/event, 11.7.2.3 *Table 11-11*
descriptor index, 9.4.3, 9.4.8
descriptors
accessing, 11.23.1
in bus enumeration process, 9.1.2
class-specific descriptors, 9.5, 11.23.2.1
configuration descriptors, 9.6.3, 9.6.4, 10.3.1, 10.5.2.4
control transfers and, 5.5, 5.5.3
defined, 9.5
descriptor index, 9.4.3, 9.4.8
device class definitions, 9.7, 9.7.1
device descriptors, 9.4 *Table 9-5,* 9.6.1 to 9.6.5
endpoint descriptors, 9.6.6
getting descriptors, 9.4.3, 10.5.2.3
hub descriptors, 11.23 to 11.23.2.1, 11.24.2.5, 11.24.2.10
interface descriptors, 9.2.3, 9.6.5
isochronous transfer capabilities, 5.12
listing remote wakeup capabilities, 9.2.5.2
other speed configuration descriptor, 9.6.4
overview, 9.5 to 9.7.3
setting descriptors, 5.3.1.1, 9.4.8, 10.5.2.12
speed dependent descriptors, 9.2.6.6, 9.6.4
string descriptors, 9.6.7
USBDI mechanisms for getting descriptors, 10.5.2.3
vendor-specific descriptors, 9.5
deserialization of transmissions, 10.2.2
detachable cables
cable delay, 7.1.16
connectors and, 6.2
detachable cable assemblies, 6.4.1
inter-packet delay and, 7.1.18.1
low-speed detachable cables, 6.4.4
maximum capacitance, 7.1.6.1
termination, 7.1.5.1
voltage drop budget, 7.2.2
detached devices, 9.1.1.1, 9.1.2
detaching devices. *See* dynamic insertion and removal
detecting connect and disconnect conditions, 7.1.7.3, 7.1.20
detecting errors. *See* error detection and handling
detecting hub and port status changes, 7.1.7.5, 11.12.2, 11.12.3, 11.12.4
detecting over-current conditions, 7.2.1.2.1
detecting speed of devices. *See* speed detection
Detection mechanism, 7.1.5.2
Dev_Do_BCINTI state machine, 8.5.2 *Figure 8-34*
Dev_Do_BCINTO state machine, 8.5.2 *Figure 8-32*
Dev_Do_IN state machine, 8.5 *Figure 8-25*
Dev_Do_IsochI state machine, 8.5.5 *Figure 8-43*
Dev_Do_IsochO state machine, 8.5.5 *Figure 8-41*
Dev_Do_OUT state machine, 8.5 *Figure 8-24*
Dev_HS_BCO state machine, 8.5.1.1 *Figure 8-29*
Dev_HS_ping state machine, 8.5.1.1 *Figure 8-28*
device addresses, 2.0 *glossary*. *See also* addresses; devices
device classes. *See also* USB device framework
class codes, 9.2.3
defined, 4.8
descriptors, 9.2.3, 9.6.1, 9.7
device characteristics, 4.8.1
device class definitions, 9.7
device qualifier descriptors, 9.6.2
getting class-specific descriptors, 9.5
hub class-specific requests, 11.24.2 to 11.24.2.13
interfaces and endpoint usage, 9.7.2
requests, 9.7.3
standard, class, and vendor information, 4.8.1
Device Class Specification for Audio Devices Revision 1.0, 9.6
DEVICE descriptor, 9.4 *Table 9-5*
device descriptors
descriptor types, 9.4 *Table 9-5*
device class descriptors, 9.2.3, 9.7
device qualifier descriptors, 9.6.2
GetDescriptor() request, 9.4.3
getting class-specific descriptors, 9.5
hubs, 11.23.1
overview, 9.6.1
speed dependent descriptors, 9.2.6.6
standard definitions, 9.6.1 to 9.6.5
device drivers, 5.12.4.4, 10.3.1
device endpoints, 2.0 *glossary,* 5.3.1.1. *See also* endpoints
device-initiated resume. *See* remote wakeup

Microchip Ex. 1018, p. 610
Microchip v. Aptiv
IPR2024-00494

Device layer
descriptors, 9.5 to 9.7.3
device states, 9.1 to 9.1.2
generic USB device operations, 9.2 to 9.2.7
standard device requests, 9.4 to 9.4.11
in USB device framework, 9
USB device requests, 9.3 to 9.3.5
Device_Process_trans state machine, 8.5 *Figure 8-23*
device qualifier descriptors, 9.2.6.6, 9.4.3, 9.4 *Table 9-5,* 9.6.1, 9.6.2
Device release numbers, 9.6.1
DEVICE_REMOTE_WAKEUP, 9.4 *Table 9-6*
DeviceRemovable field (hub descriptors), 11.23.2.1
device resources, 2.0 *glossary*. *See also* buffers; endpoints
devices. *See also* USB device framework
address assignment, 9.1.2, 9.2.2
characteristics and configuration (*See also* device descriptors)
configuration, 4.8.2.2, 9.2.3
data-rate tolerance, 7.1.11
descriptors, 9.5 to 9.7.3, 9.6.1
device characteristics, 4.8.1
device classes, 4.8, 9.7
device descriptions, 4.8.2 to 4.8.2.1
device speed, 7.1.5 to 7.1.5.2, 7.1.7.3, 11.8.2
host role in configuration, 10.3.1
optional endpoints, 5.3.1.2
USBD role in configuration, 10.5.4.1 to 10.5.4.1.4
data transfer, 9.2.4
communication flow requirements, 5.3
control transfers and, 5.5
detailed communication flow illustrated, 5.3
differing bus access for transfers, 5.11
jitter budget table, 7.1.15.1
PING flow control, 8.5.1, 8.5.1.1
response to IN transactions, 8.4.6.1
response to OUT transactions, 8.4.6.3
response to SETUP transactions, 8.4.6.4
role in bulk transfers, 8.5.2
device event timings, 7.3.2 *Table 7-14*
devices defined, 2.0 *glossary*
device state machines, 8.5
dynamic attach and detach, 9.2.1
power distribution, 7.2.4 to 7.2.4.2
removing, 10.5.2.6, 10.5.4.1.4
USBDI mechanisms, 10.5.2.5, 10.5.2.6
generic USB device operations, 9.2 to 9.2.7
port indicators, 11.5.3 to 11.5.3.1
devices *(continued)*
power distribution, 7.2.1, 9.2.5
bus-powered devices, 4.3.1, 7.2.1.1
dynamic attach and detach, 7.2.4 to 7.2.4.2
high-power bus-powered functions, 7.2.1.4
low-power bus-powered functions, 7.2.1.3
power supply and, 4.3.1
self-powered devices, 4.3.1, 7.2.1.2, 7.2.1.5
suspend/resume conditions, 7.2.3
voltage drop budget, 7.2.2
requests
host communication, 10.1.1
request errors, 9.2.7
request processing, 9.2.6 to 9.2.6.6
standard device requests, 9.4 to 9.4.11
USB device requests, 9.3 to 9.3.5
state machines, 8.5, 8.5.2, 8.5.5
status
device states, 9.1 to 9.1.2, 11.12.2
getting device status, 9.4.5
getting port status, 11.24.2.7.1.1
subtree devices after wakeup, 10.5.4.5
turn-around timers, 8.7.2
types of devices
composite devices, 5.2.3
compound devices, 4.8.2.2, 5.2.3
functions, 4.8.2.2
hubs, 4.8.2.1
mapping physical and virtual devices, 5.12.4.4
virtual devices, 2.0 *glossary*
in USB topology, 4.1.1.2, 5.2, 5.2.2, 9.0
device software, defined, 2.0 *glossary*
device state machines, 8.5. *See also specific state machines under* Dev_
diameter of cables, 6.6.2
diamond symbols in state machines, 8.5, 11.15
dielectric withstanding voltage standards, 6.7 *Table 6-7*
Differential 0 bus state, 7.1.7.2
Differential 1 bus state, 7.1.7.1, 7.1.7.2
Differential 2 bus state, 7.1.7.1
differential data jitter, 7.3.3 *Figure 7-49,* 7.3.3 *Figure 7-52*
differential delay, 7.3.2 *Table 7-11,* 7.3.3 *Figure 7-52*
differential-ended components in upstream ports, 11.6.1, 11.6.2
differential envelope detectors, 7.1
differential input receivers, 1, 7.1, 7.1.4.1, 7.1.6, 7.1 *Table 7-1*
differential output drivers, USBD as, 7.1.1
differential signaling, 7.1.7.1, 7.1.7.2, 7.1.7.4.1
differential termination impedance, 7.1.6.2
differential-to-EOP transition skew, 7.3.3 *Figure 7-50*

Microchip Ex. 1018, p. 611
Microchip v. Aptiv
IPR2024-00494

dimensional inspection standards, 6.7 *Table 6-7*
Direction bit, 9.3.1, 9.3.4
direction of communication flow, 5.4
  bmRequestType field, 9.3.1
  bulk transfers, 5.8.2
  bus protocol overview, 4.4
  control transfers, 5.5.2
  interrupt transfers, 5.7.2
  isochronous transfers, 5.6.2
disabled ports, 11.5, 11.5.1.4, 11.24.2.7.1, 11.24.2.7.2
Disabled state, 11.5, 11.5.1.4
disabling features, 9.4.1
discarding packets, 11.3.2
Disconnect_Detect signal/event, 11.5.2, 11.5 *Table 11-5*
Disconnected state
  connect and disconnect signaling, 7.1.7.3
  detecting, 7.1, 7.1.4.2, 7.1.20
  downstream ports, 11.5, 11.5.1.3
  signaling levels and, 7.1.7.1, 7.1.7.2
disconnecting devices. *See* dynamic insertion and removal
disconnection envelope detectors, 7.1.7.3, 7.1 *Table 7-1*
disconnect timer, 11.5.2
distortion, minimizing in SOP, 7.1.7.4.1
DLL lock, 7.1
documents, applicable standards, 6.7.1
down counters in hub timing, 11.2.3.1
downstream facing ports and hubs
  Disconnect state detection, 7.1
  downstream connectivity defined, 11.1.2.1
  downstream defined, 2.0 *glossary*
  downstream facing port state machine, 11.5
  downstream plugs, 6.2
  downstream ports defined, 4.8.2.1
  driver speed and, 7.1.2.3
  enumeration handling, 11.12.6
  high-speed driver characteristics and, 7.1.1.3
  high-speed signaling and, 7.1.7.6.1, 7.1.7.6.2, 11.1.1
  in hub architecture, 11.1.1
  hub delay, 7.3.3 *Figure 7-52*
  hub descriptors, 11.23.2.1
  hub EOP delay and EOP skew, 7.3.3 *Figure 7-53*
  input capacitance, 7.1.6.1
  jitter, 7.3.2 *Table 7-10*
  multiple Transaction Translators, 11.14.1.3
  port state descriptions, 11.5.1 to 11.5.1.14
  reset state machines, C.1
  signaling delays, 7.1.14.1
  signaling speeds, 7.1
  status changes, 11.12.6
  test mode support, 7.1.20
downstream facing ports and hubs *(continued)*
  transceivers, 7.1, 7.1.7.1, 7.1.7.2
downstream facing transceivers, high-speed signaling and, 7, 7.1
downstream packets (HSD1), 8.5, 11.15
drain wires, 6.5.2, 6.6.1, 6.6.2
dribble, defined, 7.1.9.1
drift, 5.12.1, 5.12.3
driver characteristics
  full-speed driver characteristics, 7.1.1.1
  full-speed source electrical characteristics, 7.3.2 *Table 7-9*
  high-speed driver characteristics, 7.1.1.3
  high-speed source electrical characteristics, 7.3.2 *Table 7-8*
  low-speed driver characteristics, 7.1.1.2, 7.1 *Table 7-1*
  low-speed source electrical characteristics, 7.3.2 *Table 7-10*
  overview, 7.1.1
drivers
  defined, 2.0 *glossary*
  role in configuration, 10.3.1
  in source-to-sink connectivity, 5.12.4.4
droop, 7.2.3, 7.2.4.1
dual pin-type receptacles, 6.9
durability standards, 6.7 *Table 6-7*
DWORD, defined, 2.0 *glossary*
dynamic insertion and removal, 9.2.1
  attaching devices, 4.6.1
  defined, 2.0 *glossary*
  detecting insertion and removal, 4.9, 9.2.1
  Hub Repeater responsibilities, 11.1
  hub support for, 11.1
  power control, 7.2.3, 7.2.4 to 7.2.4.2
  power-on and connection events timing, 7.1.7.3
  removing devices, 4.6.2
  USB robustness and, 4.5

E

E field (End), 8.4.2.2
E2PROM defined, 2.0 *glossary*
ease-of-use considerations, 1.1
EBEmptied signal/event, 11.7.1.4 *Table 11-10*
edges of signals
  cable delay, 7.1.16
  data source jitter, 7.1.13.1.1
  edge transition density, 8.2
  optional edge rate control capacitors, 7.1.6.1
EEPROM, defined, 2.0 *glossary*
elasticity buffer, 11.7.1.3
Electrical Connector/Socket Test Procedures, 6.7.1
Electrically Erasable Programmable Read Only Memory (EEPROM), 2.0 *glossary*

Microchip Ex. 1018, p. 612
Microchip v. Aptiv
IPR2024-00494

Electrical Performance Properties of Insulation and Jacket for Telecommunication Wire and Cable, 6.7.1
electrical specifications, 6.1, 7
applicable documents, 6.7.1
bus timing/electrical characteristics, 7.3.2
cables, 6.3, 6.4 to 6.4.4, 6.6 to 6.6.5
connectors, 6.2, 6.5 to 6.5.4.3
overview, 4.2.1, 6
PCB reference drawings, 6.9
physical layer specifications, 7.3 to 7.3.3
power distribution, 7.2 to 7.2.1.5, 7.2.3, 7.2.4 to 7.2.4.2
signaling, 7.1 to 7.1.20
standards for, 6.7, 7.3.1
timing waveforms, 7.3.3
USB grounding, 6.8
embedded hubs, 4.8.2.2, 5.2.3
EMI, USB grounding and, 6.8
enabled ports
connectivity and, 11.1.2.1
downstream ports, 11.5, 11.5.1.6
getting port status, 11.24.2.7.1
PORT_ENABLE bit, 11.24.2.7.1.2
port status change bits, 11.24.2.7.2
Enabled state, 11.5, 11.5.1.6
Enable Transmit state, 11.7.1.4.3
encoding data, 7.1.8, 11.18.4
"end" encoding, 11.18.4
End field (E), 8.4.2.2
End-of-Frame (EOF). *See* EOFs
End of High-speed Packet (HSEOP), 7.1.7.2, 7.1.7.4.2
End-of-Packet (EOP). *See* EOPs
End-of-Packet bus state, 7.1.7.1, 7.1.7.2, 7.1.7.4.1, 7.1.7.4.2
end-of-packet delimiter. *See* EOPs
ENDP field, 8.3.2.2, 8.3.5.1, 8.4.1
endpoint addresses, 2.0 *glossary,* 5.3.1, 9.6.6
ENDPOINT descriptor, 9.4 *Table 9-5*
endpoint descriptors, 9.4.3, 9.6.1, 9.6.5, 9.6.6
endpoint direction, defined, 2.0 *glossary*
endpoint field (ENDP), 8.3.2.2, 8.3.5.1, 8.4.1
ENDPOINT_HALT, 9.4 *Table 9-6*
endpoint numbers, 2.0 *glossary,* 5.3.1
endpoints
addresses, 9.6.6
characteristics, 5.3.1
description in descriptors, 9.4.3, 9.6.1, 9.6.5, 9.6.6
in device class definitions, 9.7.2
direction of flow, 5.3.1
endpoint address field, 8.3.2.2
endpoint aliasing, 8.3.2
endpoint zero requirements, 4.8.1, 5.3.1.1, 5.3.1.2, 5.3.2
endpoints *(continued)*
explicit feedback endpoints, 9.6.5, 9.6.6
getting endpoint status, 9.4.5
high-bandwidth endpoints, 2.0 *glossary,* 5.7.4
high-speed signaling attributes, 9.6.6
Hub Controller endpoint organization, 11.12.1
in interfaces, 9.2.3, 9.6.3, 9.6.5
logical devices as collections of endpoints, 5.3
message pipes and, 5.3.2.2
non-endpoint zero requirements, 5.3.1.2
number matching, 9.6.6
overview, 5.3.1
pipes and, 4.4, 5.3.2
programmable data rates, 2.0 *glossary*
reflected endpoint status, 10.5.2.2
role in data transfers, 4.7
samples, 2.0 *glossary*
specifying in *wIndex* field, 9.3.4
state machines, 8.5
stream pipes and, 5.3.2.1
synchronization frame, 9.4.11
Transfer Types, Synchronization Types, and Usage Types, 9.6.6
endpoint synchronization type, 5.12.4, 5.12.4.1
Endpoint Type field (ET), 8.4.2.2
endpoint type field (ET), 8.4.2.2
endpoint zero
Default Control Pipe and, 5.3.2
in device characteristics, 4.8.1
non-endpoint zero requirements, 5.3.1.2
requirements, 5.3.1.1
end-to-end signal delay, 7.1.19 to 7.1.19.2
end users, 2.0 *glossary,* 3.3
entering test mode, 7.1.20
entry points into state machines, 8.5
enumeration. *See* bus enumeration
envelope detectors, 2.0 *glossary,* 7.1, 7.1.4.2, 7.1.7.3, 7.1 *Table 7-1*
environmental characteristics for cables, 6.6.4
environmental compliance standards, 6.7
EOF1 or EOF2 signal/event
frame and microframe timers, 11.2.3.2, 11.2.5 to 11.2.5.2
host behavior at end-of-frame, 11.3
in Hub Repeater state machine, 11.7.2.3 *Table 11-11*
in transmitter state machine, 11.6.4 *Table 11-9*

Microchip Ex. 1018, p. 613
Microchip v. Aptiv
IPR2024-00494

EOFs
advancing, 11.2.3.2
defined, 2.0 *glossary*
in frame and microframe timer synchronization, 11.2.3.2
host behavior at end-of-frame, 11.3 to 11.3.3
Host Controller frame and microframe generation, 10.2.3
in transaction completion prediction, 11.3.3
EOI signal/event
defined, 11.7.1.4 *Table 11-10*
in downstream port state machine, 11.5 *Table 11-5*
in internal port state machine, 11.4
in receiver state machine, 11.6.3 *Table 11-8*
in transmitter state machine, 11.6.4
EOP bus state, 7.1.7.1, 7.1.7.2, 7.1.7.4.1, 7.1.7.4.2
EOPs
defined, 2.0 *glossary*
differential-to-EOP transition skew and EOP width, 7.3.3 *Figure 7-50*
EOP delimiter, 8.3
EOP dribble defined, 11.7.1.1
EOP width, 7.1.13.2 to 7.1.13.2.2, 7.3.3 *Figure 7-50*
error detection through bus turn-around timing, 8.7.2
extra bits and, 7.1.9, 7.1.9.1
false EOPs, 2.0 *glossary,* 8.7.3, 11.15
handshake packets and, 8.4.5
high-speed signaling and, 7.1
hub EOP delay and EOP skew, 7.3.3 *Figure 7-53*
hub/repeater electrical characteristics, 7.3.2 *Table 7-11*
hub signaling at EOF1, 11.3.1
intervals between IN token and EOP, 11.3.3
propagation delays, 7.1.14.1
EOR signal/event, 11.6.3 *Table 11-8*
equations
buffering for rate matching, 5.12.8
buffer sizes in functions and software, 5.11.4
bus transaction times, 5.11.3
ERR handshake
interrupt transactions, 11.20.4
isochronous transactions, 11.21.1, 11.21.4
Transaction Translator response generation, 11.18.5
error detection and handling. *See also* corrupted transfers and requests
"3 strikes and you're out" mechanism, 11.17.1
babble and loss of activity recovery, 8.7.4
bit stuff violations, 8.7.1
bulk transfers and, 5.8.5, 8.5.2
error detection and handling. *(Continued)*
bus turn-around timing, 8.7.2
busy (ready/x) state, 11.17.5
control transfers and, 5.5.5, 8.5.3.1
corrupted ACK handshake, 8.5.3.3, 8.6.4
corrupted SOF packets in isochronous transfers, 5.12.6
CRCs, 8.3.5, 8.7.1, 11.15, 11.20.3, 11.20.4, 11.21.3, 11.21.4
data corrupted or not accepted, 8.6.3
error count tally, 10.2.6, 11.17.1
error handling for transfers, 5.4
error handling on last data transaction, 8.5.3.3
false EOPs, 2.0 *glossary,* 8.7.3
HC_Data_or_error state machine, 11.20.2
high bandwidth transactions, 5.9.2
Host Controller role in, 10.2.6
Hub Repeater responsibilities, 11.1
hub role in, 11.1.2.3
interrupt transfers and, 5.7.5
isochronous transfers and, 5.6.4, 5.6.5, 5.12.7
notation for error cases, 11.15
overview, 8.7
packet error categories, 8.7.1
periodic transactions, 11.18.4
PID check bits, 8.7.1
Port Error conditions, 11.8.1
port indicators, 11.5.3 to 11.5.3.1
Request Errors, 9.2.7
sample size and, 5.12.8
short packets and error conditions, 5.3.2
split transaction sequencing, 11.21.3
status values for, 11.15
synchronous data connectivity, 5.12.4.4.2
timeouts, 8.7.2, 11.17.1
Transaction Translator error handling, 11.22
USBD role in, 10.5.4.4
USB robustness and, 4.5.1, 4.5.2
ERR PID, 8.3.1 *Table 8-1,* 8.4.5
ESD, USB grounding and, 6.8
ET field (Endpoint Type), 8.4.2.2
event notifications, USBD and, 10.5.4.3
example declarations in state machines, B.1, B.2, B.3
exception handling. *See* error detection and handling
Exception Window, 7.1.6.2
exiting test mode, 7.1.20
exit points from state machines, 8.5
explicit feedback endpoints, 9.6.5, 9.6.6
extended descriptor definitions, 9.7.1
extensibility of USB architecture, 4.10
extension cable assemblies, 6.4.4
externally-powered hubs, 7.2.1. *See also* self-powered hubs
extraction force standards, 6.7 *Table 6-7*

Microchip Ex. 1018, p. 614
Microchip v. Aptiv
IPR2024-00494

eye pattern templates
defined, 2.0 *glossary*
error rates and jitter tolerance, 7.1.14.2, 7.1.15.2
high-speed receiver characteristics and, 7.1.4.2
overview, 7.1.2.2
transmit eye patterns, 7.1, 7.1.2

F

failed data transactions, 8.6.3
false EOPs, 2.0 *glossary,* 8.7.3, 11.15
fault detection. *See* error detection and handling
features
hub class feature selectors, 11.24.2
SetFeature() request, 9.4.9
setting hub features, 11.24.2.12
standard feature selectors, 9.4 *Table 9-6*
feedback endpoints, 9.6.6
feedback for isochronous transfers, 5.12.4.2, 5.12.4.3, 9.6.5
ferrite beads, 7.1.6.2
fields. *See names of specific fields*
flammability
cables, 6.6.4
Series "A" and Series "B" plugs, 6.5.4.1
Series "A" and Series "B" receptacles, 6.5.3.1
standards, 6.7 *Table 6-7*
flexibility of USB devices, 3.3
flow control mechanisms
in bus protocol overview, 4.4
handshake packets and, 8.4.5
non-periodic transactions, 11.14.2.2
USB robustness and, 4.5
flow sequences
non-periodic transactions, 11.17 to 11.17.5
periodic transactions, 11.18 to 11.18.8, 11.20 to 11.20.4, 11.21.1
split transaction notation for, 11.15
flyback voltage, 7.2.4.2
format of USB device requests, 9.3
formulas
buffering for rate matching, 5.12.8
buffer sizes in functions and software, 5.11.4
bus transaction times, 5.11.3
frame and microframe intervals
full-speed source electrical characteristics, 7.3.2 *Table 7-9*
high-speed source electrical characteristics, 7.3.2 *Table 7-8*
low-speed source electrical characteristics, 7.3.2 *Table 7-10*
repeatability, 7.1.12
frame and microframe numbers
buffering for rate matching, 5.12.8
frame and microframe number field, 8.4.3
frame number field, 8.3.3
frame numbers, 8.3.3
generating frames and microframes, 10.2.3
illustrated, 8.4.3.1
SOF tracking, 5.12.6
frame and microframe timers
frame wander, 11.2.5.2
hub frame timer, 11.2 to 11.2.5.2
timing skew, 11.2.5.1 to 11.2.5.2
TT loss of synchronization, 11.22.1
frame clocks, 5.12.3, 5.12.4.1.2, 11.18.3
frame pattern, defined, 2.0 *glossary*
frames and microframes. *See also* frame and microframe timers
allocating bandwidth, 4.7.5, 5.11.1 to 5.11.1.5, 10.3.2
available time in frames and microframes, 5.5.4, 5.6, 5.6.4, 5.7.4, 5.8.4, 5.11.5
babble and loss of activity recovery, 8.7.4
bandwidth reclamation, 5.11.5
best case full-speed budgets, 11.18.1, 11.18.4
bit time zero, 11.3
clock tracking and microframe SOFs, 5.12.4.1.2
control transfer reserved portions, 5.5.4
data prebuffering and, 5.12.5
defined, 2.0 *glossary,* 5.3.3
error handling in transfers, 5.12.7
frame and microframe intervals, 7.1.12, 7.3.2 *Table 7-8,* 7.3.2 *Table 7-9,* 7.3.2 *Table 7-10*
frame and microframe numbers (*See* frame and microframe numbers)
frame and microframe timer ranges, 11.2.1 to 11.2.2
frame wander, defined, 11.2.5.2
generation role of Host Controller, 10.2.3
generation role of Transaction Translator, 11.18.3
host behavior at end-of-frame, 11.3
interrupt transfer limitations, 5.7.4
isochronous transactions, 5.6.3, 5.6.4, 5.12.4.2, 8.5.5
jitter, 11.2.4
maximum allowable transactions, 5.4.1, 11.18.6.3
microframe numbers, 8.4.3.1

Microchip Ex. 1018, p. 615
Microchip v. Aptiv
IPR2024-00494

frames and microframes *(Continued)*
microframe pipelines
buffer space, 11.19
clearing and aborting transactions, 11.18.6
defined, 11.18.2
periodic split transactions, 11.14.2.1, 11.18
resetting, 11.24.2.9
transaction tracking, 11.18.7
multiple transactions, 5.6.4, 5.7.4, 5.9, 5.9.2, 9.6.6
organization of transactions within, 5.11.2
overview, 8.4.3.1
samples per frame in isochronous transfers, 5.12.4.2
SOF packets, 8.4.3
SOF tracking, 5.12.6
split transactions and, 5.10
synch frame requests, 9.4.11
timers, 11.2 to 11.2.5.2
timer synchronization, 11.2.3 to 11.2.3.3, 11.22.1
toggle sequencing, 8.5.5
zeroth microframe, 9.4.11, 11.14.2.3
freeing pending start-splits, 11.18.6.2
frequency-locked clocks, 5.12.3
Fs. *See* SRC
Fsus state, 11.4, 11.4.3
full-duplex, defined, 2.0 *glossary*
full-speed buffers, 7.1.2.1
full-speed cables. *See* high-/full-speed cables
full-speed driver characteristics, 7.1.1.1, 7.1 *Table 7-1*
full-speed functions and hubs
bulk transfers and, 5.8.4
cable and resistor connections, 7.1.5.1
connect detection, 7.1.7.3
control transfers and, 5.5.3, 5.5.4, 5.5.4 *Table 5-2*
data-rate tolerance, 7.1.11
defined, 2.0 *glossary*
detachable cables and, 6.4.1
full-speed port transceiver, 7.1.7.1
full-speed source electrical characteristics, 7.3.2 *Table 7-9*
full- *vs.* low-speed port behavior, 11.8.4
getting port status, 11.24.2.7.1
hub/repeater electrical characteristics, 7.3.2 *Table 7-11*
hub support for, 11.1
input capacitance, 7.1.6.1
interrupt transfers and, 5.7.3, 5.7.4 *Table 5-7*
isochronous transfers and, 5.6.4
maximum data payload, 8.4.4
optional endpoints, 5.3.1.2
in physical bus topology, 5.2.3
reset states and, C.2.2
full-speed functions and hubs *(Continued)*
sampling rates, 5.12.4.2
signal termination, 7.1.5.1
SOF PID and, 8.4.3
speed detection and, 11.8.2
Transmit state and, 11.5.1.7
full-speed signaling
babble and loss of activity recovery, 8.7.4
best case full-speed budgets, 11.18.1, 11.18.4
bus transactions and, 4.4
calculating transaction times, 5.11.3
data rates, 4.2.1
data signaling overview, 7.1.7.4.1
data source jitter, 7.1.13.1.1
defined, 2.0 *glossary*
differential receivers, 7.1 *Table 7-1*
downstream and upstream facing ports, 7.1
driver characteristics, 7.1.1
driver requirements, 7.1.2.3
endpoint zero requirements, 5.3.1.1
EOF timing points, 11.2.5.2
EOP width, 7.1.13.2.1
errors, 8.6.4
frame timer ranges, 11.2.2
full-speed loads, 7.1.2.1
high-speed devices operating at full-speed, 5.3.1.1
host behavior at end-of-frame, 11.3 to 11.3.3
hub class descriptors and, 11.23.1
intervals between IN token and EOP, 11.3.3
isochronous transaction limits, 5.6.3
J and K states, 7.1.7.1
jitter budget table, 7.1.15.1
propagation delays, 7.1.14.1
receiver characteristics, 7.1.4.1
reset signaling, 7.1.7.5
sampling rates, 5.12.4.2
scheduling, 11.14.2.3
speed detection, 9.1.1.3
Transaction Translator and, 4.8.2.1, 11.18.3, 11.18.5
Full Suspend (Fsus) state, 11.4, 11.4.3
function address field (ADDR), 8.3.2.1, 8.4.2.2
functional stall, 8.4.5, 8.5.3.4
Function layer
detailed communication flow, 5.3
illustrated, 5.1
interlayer communications model, 10.1.1

Microchip Ex. 1018, p. 616
Microchip v. Aptiv
IPR2024-00494

functions. *See also* devices; full-speed functions and hubs; high-speed functions and hubs; low-speed functions and hubs
address assignment, 9.1.2, 9.2.2
characteristics and configuration (*See also* device descriptors)
configuration, 4.8.2.2, 9.2.3
data-rate tolerance, 7.1.11
descriptors, 9.5 to 9.7.3, 9.6.1
device characteristics, 4.8.1
device classes, 4.8, 9.7
device speed, 7.1.7.3, 11.8.2
host role in configuration, 10.3.1
optional endpoints, 5.3.1.2
data transfer
communication flow requirements, 5.3
control transfers and, 5.5
detailed communication flow illustrated, 5.3
differing bus access for transfers, 5.11
jitter budget table, 7.1.15.1
overview, 9.2.4
PING flow control and OUT transactions, 8.5.1, 8.5.1.1
response to IN transactions, 8.4.6.1
response to OUT transactions, 8.4.6.3
response to SETUP transactions, 8.4.6.4
role in bulk transfers, 8.5.2
device event timings, 7.3.2 *Table 7-14*
devices defined, 2.0 *glossary*
dynamic attach and detach, 9.2.1
power distribution, 7.2.4 to 7.2.4.2
removing, 10.5.2.6, 10.5.4.1.4
USB mechanisms, 10.5.2.5, 10.5.2.6
generic USB device operations, 9.2 to 9.2.7
overview, 4.8.2.2
power distribution, 7.2.1, 9.2.5
bus-powered devices, 4.3.1, 7.2.1.1
dynamic attach and detach, 7.2.4 to 7.2.4.2
high-power bus-powered functions, 7.2.1.4
low-power bus-powered functions, 7.2.1.3
power supply and, 4.3.1
self-powered functions, 7.2.1.2, 7.2.1.5
suspend/resume conditions, 7.2.3
voltage drop budget, 7.2.2
requests
host communication with, 10.1.1
request errors, 9.2.7
request processing, 9.2.6 to 9.2.6.6
standard device requests, 9.4 to 9.4.11
USB device requests, 9.3 to 9.3.5
status, 9.1 to 9.1.2, 9.4.5
types of devices
compound devices, 4.8.2.2
functions, 4.8.2.2
mapping physical and virtual devices, 5.12.4.4
virtual devices, 2.0 *glossary*
in USB topology, 4.1.1.2, 5.2.2, 5.2.3, 9.0
function-to-host transfers. *See* IN PID

G

gang-mode power control, 11.23.2.1
garbling messages in Collision conditions, 11.8.3
Generate End of Packet Towards Upstream Port state (GEOPTU), 11.6.4, 11.6.4.5
Generate Resume state, 11.4, 11.4.4
generic USB device operations, 9.2 to 9.2.7
GEOPTU state, 11.6.4, 11.6.4.5
GetConfiguration() request, GET_CONFIGURATION
hub requests, 11.24.1
overview, 9.4.2
returning interface descriptors, 9.6.5
standard device request codes, 9.4
GetDescriptor() request, GET_DESCRIPTOR, 11.23.1
device_qualifer descriptors, 9.6.2
endpoint descriptors, 9.6.6
GetDescriptor(CONFIGURATION) request, 9.5, 9.6.6
GetHubDescriptor() request, 11.24.2.5
hub class requests, 11.24.2
hub descriptors, 11.24.2.5
hub requests, 11.24.1
interface descriptors, 9.6.5
other_speed_configuration descriptors, 9.6.4
overview, 9.4.3
standard device request codes, 9.4
GetHubDescriptor() request, 11.24.2, 11.24.2.5
GetHubStatus() request, 11.24.2, 11.24.2.6
GetInterface() request, GET_INTERFACE
alternate settings for interfaces, 9.2.3
hub requests, 11.24.1
interface descriptors, 9.6.5
overview, 9.4.4
standard device request codes, 9.4
GetPortStatus() request
class-specific requests, 11.24.2
overview, 11.24.2.7 to 11.24.2.7.2.5
PORT_INDICATOR, 11.24.2.7.1.10
during test mode, 11.24.2.13
GET_STATE, 11.24.2

Microchip Ex. 1018, p. 617
Microchip v. Aptiv
IPR2024-00494

GetStatus() request, GET_STATUS
GetHubStatus() request, 11.24.2.6
GetPortStatus() request, 11.24.2.7
hub class requests, 11.24.2
overview, 9.4.5
PORT, 11.12.6
standard device request codes, 9.4
GetTTState() request, GET_TT_STATE
hub class requests, 11.24.2
overview, 11.24.2.8
STOP_TT, 11.24.2.11
global declarations in state machines, B.1
global resumes
frame and microframe timer synchronization, 11.2.3.3
hub support, 11.9
signaling, 7.1.7.7
global suspend, 7.1.7.6.1, 11.9
glossary, 2.0
GND leads
cable electrical characteristics, 7.3.2 *Table 7-12*
captive cable assemblies, 6.4.2, 6.4.3
detachable cables, 6.4.1
electrical specifications overview, 4.2.1
standardized contact terminating assignments, 6.5.2
GResume state, 11.4, 11.4.4
grounding, 6.8

H

halted pipes, 10.5.2.2
Halt feature
bulk transfers, 5.8.5
control transfers, 5.5.5, 8.5.3.4
functional stalls, 8.4.5
GetStatus() request, 9.4.5
interrupt transfers, 5.7.5, 8.5.4
isochronous transfers, 5.6.5
responses to standard device requests, 9.4
handshakes. *See also* ACKs; NAKs; STALLs
ACK PID, 8.3.1 *Table 8-1*
bulk transfers, 8.5.2
bus protocol overview, 4.4
defined, 2.0 *glossary*
detection handshakes, 7.1.7.5, 7.1.7.6
function response to IN transactions, 8.4.6.1
function response to OUT transactions, 8.4.6.3
function response to SETUP transactions, 8.4.6.4
handshake responses, 8.4.6 to 8.4.6.4
host response to IN transactions, 8.4.6.2
isochronous transfers, 5.6.5, 5.12.7
NAK PID, 5.9.1, 8.3.1 *Table 8-1,* 8.5.1, 8.5.1.1
NYET PID, 8.3.1 *Table 8-1,* 8.4.5, 8.5.1, 8.5.2
handshakes *(Continued)*
overview, 8.3.1 *Table 8-1,* 8.4.5
packet field formats, 8.3 to 8.3.5.2
PING flow control and OUT transactions, 8.5.1, 8.5.1.1
STALL PID, 8.3.1 *Table 8-1*
total allocation of bit times, 11.3.3
transaction notation for, 11.15
hardwired cable assemblies, 6.4.2
HCD (Host Controller Driver)
defined, 2.0 *glossary,* 5.3
HCDI (Host Controller Driver Interface), 10.1.1, 10.4
overview, 10.4
software interface overview, 10.3
in transfer management, 5.11.1, 5.11.1.3
in USB topology, 5.2.1, 10.1.1
HC_Data_or_error state machine, 11.20.2
HCDI (Host Controller Driver Interface), 10.1.1, 10.4
HC_Do_BCINTI state machine, 8.5.2 *Figure 8-33*
HC_Do_BCINTO state machine, 8.5.2 *Figure 8-31*
HC_Do_BICS state machine, 11.17.2
HC_Do_BISS state machine, 11.17.2
HC_Do_BOCS state machine, 11.17.2
HC_Do_BOSS state machine, 11.17.2
HC_Do_complete state machine, 11.16.1.1.2
HC_Do_IntICS state machine, 11.20.2
HC_Do_IntISS state machine, 11.20.2
HC_Do_intOCS state machine, 11.20.2
HC_Do_IntOSS state machine, 11.20.2
HC_Do_IsochICS state machine, 11.21.2
HC_Do_IsochISS state machine, 11.21.2
HC_Do_IsochI state machine, 8.5.5 *Figure 8-42*
HC_Do_IsochOSS state machine, 11.21.2
HC_Do_IsochO state machine, 8.5.5 *Figure 8-40*
HC_Do_nonsplit state machine, 8.5 *Figure 8-26*
HC_Do_Start state machine, 11.16.1.1.1
HC_HS_BCO state machine, 8.5.1 *Figure 8-27*
HC_Process_command state machine, 11.16.1.1
HEOP signal/event, 11.6.4 *Table 11-9,* 11.7.2.1, 11.7.2.3 *Table 11-11*
hierarchical state machines, 8.5, 11.15
high-bandwidth endpoints, 2.0 *glossary,* 5.7.4, 5.9 to 5.9.2, 5.12.3
high-bandwidth transactions, 5.12.7, 8.5.5

Microchip Ex. 1018, p. 618
Microchip v. Aptiv
IPR2024-00494

high-/full-speed cables
cable delay, 7.1.16
cable impedance tests, 6.7 *Table 6-7*
captive cable assemblies, 6.4.2
construction, 6.6, 6.6.2
description, 6.6.1
listing, 6.6.5
signal pair attenuation, 6.7 *Table 6-7*
specifications, 6.3
standards for, 6.6.3, 6.6.4, 6.7
high-powered devices
bus-powered functions, 7.2.1, 7.2.1.4
high-power ports, 7.2.1
voltage drop budget, 7.2.2
High-speed Detection Handshake, 7.1.7.5, 7.1.7.6
high-speed driver characteristics, 7.1 *Table 7-1*
high-speed functions and hubs
in application space taxonomy, 3.2
bulk transfers, 5.8.4
connect detection, 7.1.7.3
control transfers, 5.5.3, 5.5.4, 5.5.4 *Table 5-3*
data signaling rates, 7.1.11
defined, 2.0 *glossary*
detachable cables, 6.4.1
device_qualifier descriptors, 9.6.1, 9.6.2
frame and microframe numbers, 8.4.3.1
full-speed operation, 5.3.1.1
getting port status, 11.24.2.7.1
high-speed repeaters, 11.2.5, 11.7.2.1
high-speed source electrical characteristics, 7.3.2 *Table 7-8*
hub/repeater electrical characteristics, 7.3.2 *Table 7-11*
input capacitance, 7.1.6.2
interrupt transfers, 5.7.3, 5.7.4 *Table 5-8*
isochronous transfers, 5.6.4
maximum data payload, 8.4.4
NAK mechanisms, 8.5.1, 8.5.1.1
other_speed_configuration descriptors, 9.6.4
performance, 1.1
in physical bus topology, 5.2.3
port selector state machine, 11.7.1.4 to 11.7.1.4.4
reset signaling, 7.1.7.5
reset state machines, C.2.3
sampling rates, 5.12.4.2
signal termination, 7.1
SOF PID and, 8.4.3, 8.4.3.1
speed detection and, 7.1.5.2, 11.8.2
test mode support, 7.1.20
high-speed port selector state machine, 11.7.1.4 to 11.7.1.4.4
high-speed signaling
bit stuffing, 7.1.9.2
bus transactions and, 4.4
calculating transaction times, 5.11.3
Chirp J and K states, 7.1.4.2, 7.1.7.2
control transfers, 5.5.4
current drivers, 7.1 *Table 7-1*
data handling, 11.14.1.1
data rates, 4.2.1
data signaling overview, 7.1.7.4.2
defined, 2.0 *glossary*
Differential 0/Differential 1 bus states, 7.1.7.2
differential data receivers, 7.1 *Table 7-1*
"disallowed" situations, 7.1.2.3
Disconnect state, 7.1.7.2
downstream and upstream facing ports, 7.1
driver characteristics, 7.1.1, 7.1.1.1, 7.1.1.3, 7.1.2.3
End of High-speed Packet (HSEOP), 7.1.7.2, 7.1.7.4.2
endpoints
attributes, 9.6.6
endpoint zero requirements, 5.3.1.1
high-bandwidth endpoints, 2.0 *glossary,* 5.9 to 5.9.2
EOF timing points, 11.2.5.1
EOP width, 7.1.13.2.2
error detection, 8.7.3, 8.7.4
error handling, 5.12.7, 8.6.4
eye patterns, 7.1.2.2
frame and microframe generation, 10.2.3
full-speed signaling and, 5.3.1.1, 7.1.1.1
high-speed devices operating at full-speed, 5.3.1.1
hub architecture and, 11.1.1
hub class descriptors and, 11.23.1
Idle state, 7.1.7.2
isochronous transaction limits, 5.6.3
J and K states, 7.1.7.2, 7.1.7.4.2
jitter, 7.1.13.1.2, 7.1.15.2
microframes and, 5.3.3, 11.2.1
overview, 7.1
packet repeaters, 11.7.1 to 11.7.1.4.4
PING flow control protocol and, 5.5.4, 5.8.4
propagation delays, 7.1.14.2
receiver characteristics, 7.1.4.2
reset signaling, 7.1.7.5, 7.1.7.6
resume signaling, 7.1.7.7, 11.9
sampling rates, 5.12.4.2
signaling levels, 7.1.7.2
split transactions, 5.10, 8.4.2 to 8.4.2.3

Microchip Ex. 1018, p. 619
Microchip v. Aptiv
IPR2024-00494

high-speed signaling *(Continued)*
Squelch state, 7.1.7.2
Start of High-speed Packet (HSSOP), 7.1.7.4.2
Start of high-speed Packet (HSSOP), 7.1.7.2
Suspended state and, 7.1.7.6, 11.9
synch frame requests, 9.4.11
timer range, 11.2.1
toggle sequencing, 8.5.5
Transaction Translator, 4.8.2.1, 11.1, 11.14 to 11.14.2.3
types of transactions, 11.17
HJ signal/event, 11.6.3 *Table 11-8*
HK signal/event, 11.6.3 *Table 11-8*
host, 10
in bus topology, 4.1.1.1, 5.2, 5.2.1
collecting status and activity statistics, 10.1.4
components, 10.1.1
control mechanisms, 10.1.2
EOF1 and EOF2 timing points, 11.2.5 to 11.2.5.2
host behavior at end-of-frame, 11.3 to 11.3.3
host-to-hub communications, 11.1
resource management, 10.3.2
responsibilities and capabilities, 10.1.1
role in assigning addresses, 9.2.2
role in configuration, 9.2.3, 10.3.1
synchronizing hub (micro)frame timer to host (micro)frame period, 11.2
data flow, 10.1.3
common data definitions, 10.3.4
data-rate tolerance, 7.1.11
data transfer mechanisms, 10.1.3, 10.3.3
detailed communication flow illustrated, 5.3
host response to IN transactions, 8.4.6.2
interlayer communications model, 10.1.1
role in bulk transfers, 8.5.2
defined, 2.0 *glossary,* 4.9
electrical considerations, 10.1.5
jitter budget table, 7.1.15.1
over-current protection, 7.2.1.2.1
over-current recovery, 11.12.5
hardware and software, 10.0
host accuracy variations, 11.2.1 to 11.2.2
Host Controller Driver (HCD), 10.4 (*See also* HCD)
Host Controller responsibilities, 4.9, 10.2 (*See also* Host Controller)
host jitter, 11.2.1
host tolerance, hub (micro)frame timer and, 11.2
operating system environment guides, 10.6
overview of USB Host, 10.1 to 10.1.5
power management overview, 4.3.2
software mechanisms, 10.3 to 10.3.4
host *(Continued)*
split transaction scheduling, 11.18.4
state machines, 8.5.1, 8.5.2, 8.5.5
status in USBD pipe state, 10.5.2.2
turn-around timers, 8.7.2
Universal Serial Bus Driver (USBD), 10.5 to 10.5.5 (*See also* USBD (USB Driver))
USB System Software responsibilities, 4.9 (*See also* USB System Software)
Host Controller, 4.9
best case full-speed budgets, 11.18.1, 11.18.4
in bus topology, 5.2.1
calculating buffer sizes in functions and software, 5.11.4
data transfer mechanisms, 10.1.3
bulk transfers, 5.8.3, 11.17 to 11.17.5
control transfers, 5.5.3, 5.5.4, 11.17 to 11.17.5
data processing, 10.2.4
data-rate tolerance, 7.1.11
high bandwidth transfers, 5.9.1, 5.9.2, 11.20.2
interrupt transfers, 5.7.3, 5.9.1
isochronous transfers, 5.6.3, 5.9.2, 11.21.2
non-periodic transactions, 11.17 to 11.17.5
periodic transactions, 11.18 to 11.18.8
role in transfer management, 5.11.1, 5.11.1.5
split transactions, 11.16.1 to 11.16.1.1.2
tracking transactions, 5.11.2
transaction list, 5.11.1.4
transmission error handling, 10.2.6
declarations in state machines, B.2
defined, 2.0 *glossary,* 4.9
frame and microframe generation, 10.2.3
HCD and HCDI overview, 10.4 (*See also* HCD; HCDI)
host behavior at end-of-frame, 11.3
host-system interface, 10.2.9
as implementation focus area, 5.1
implemented in USB Bus interface, 10.1.1
multiple Host Controllers, 4.10
passing preboot control to operating system, 10.5.5
port resets, 10.2.8.1
protocol engine, 10.2.5
remote wakeup and, 10.2.7
requirements, 10.2
root hub and, 10.2.8
serializer/deserializer, 10.2.2
state handling, 10.2.1
state machines, 8.5, 11.16 to 11.16.1.1.2, 11.17.2, 11.20.2, 11.21.2, B.2
status and activity monitoring, 10.1.4

Microchip Ex. 1018, p. 620
Microchip v. Aptiv
IPR2024-00494

Host Controller *(Continued)*
test mode support, 7.1.20
Transaction Translator and, 11.14.1, 11.14.1.2
USB System interaction, 10.1.1
Host Controller Driver. *See* HCD (Host Controller Driver)
Host Controller Driver Interface (HCDI), 10.1.1, 10.4
host resources, 2.0 *glossary*
host side bus interface. *See* Host Controller
host software
in bus topology, 5.2.1
as component of USB System, 10.1.1
pipes and, 10.5.1.2
status and activity monitoring, 10.1.4
host-to-function transfers. *See* OUT PID
hot plugging. *See* dynamic insertion and removal
HSD1 packets (downstream packets), 8.5, 11.15
HS_Drive_Enable, 7.1.1.3
HSEOP (End of High-speed Packet), 7.1.7.2, 7.1.7.4.2
HS_Idle signal/event, 11.6.2, 11.6.3 *Table 11-8*
HS signal/event, 11.6.3 *Table 11-8,* 11.6.4 *Table 11-9*
HSSOP (Start of High-speed Packet), 7.1.7.2, 7.1.7.4.2
HSU2 packets (upstream packets), 8.5, 11.15
Hub address field, 8.4.2.2
Hub Change field, 11.4.4, 11.24.2.6
hub class definitions
additional endpoints, 11.12.1
feature selectors, 11.24.2
request codes, 11.24.2
root hub and, 10.4
Hub Controller, 11.12 to 11.12.6
control commands, 11.1
defined, 4.8.2.1
endpoint organization, 11.12.1
hub and port change information processing, 11.12.3, 11.12.4
in hub architecture, 11.1.1, 11.12.2
internal port connection, 11.4
over-current reporting and recovery, 11.12.5
power distribution and, 7.2.1.1
role in host-to-hub communications, 11.1
status commands, 11.1
hub descriptors, 11.12.2
Hub Repeater
Collision conditions, 11.8.3
connectivity setup and tear-down, 11.1
data recovery unit, 11.7.1.2
defined, 4.8.2.1
dynamic insertion and removal, 11.1
elasticity buffer, 11.7.1.3
electrical characteristics, 7.3.2 *Table 7-11*
fault detection and recovery, 11.1
high-speed packet repeaters, 11.7.1 to 11.7.1.4.4
in hub architecture, 11.1.1
hub signaling timings, 7.1.14.1
internal port connection, 11.4
packet signaling connectivity, 11.1.2.1
repeater state descriptions, 11.2.3.3, 11.7 to 11.7.6
squelch circuit, 11.7.1.1
Wait for End of Packet (WFEOP), 11.7.6
Wait for End of Packet from Upstream Port state (WFEOPFU), 11.7.4
Wait for Start of Packet (WFSOP), 11.7.5
Wait for Start of Packet from Upstream Port state (WFSOPFU), 11.7.3
Hub Repeater state machine, 11.2.3.3, 11.7.2
hubs, 11. *See also* Hub Controller; Hub Repeater; ports
accuracy variations, 11.2.1 to 11.2.2
architecture, 4.1.1.2, 11.1
bus states
bus state evaluation, 11.8 to 11.8.4.1
collision, 11.8.3
connect/disconnect detection, 11.1
full- *vs.* low-speed behavior, 11.8.4
low-speed keep-alive, 7.1.7.1, 11.8.4.1
port error, 11.8.1
reset behavior, 11.10
speed detection, 11.8.2
status change detection, 7.1.7.5, 11.12.2, 11.12.4
in bus topology, 5.2.3, 5.2.4
characteristics and configuration, 11.13
clearing features, 11.24.2.6
data-rate tolerance, 7.1.11
descriptors, 11.12.2, 11.23 to 11.23.2.1, 11.24.2.10
full- *vs.* low-speed behavior, 11.8.4
input capacitance, 7.1.6.1
speed detection of devices, 11.8.2
typical configuration illustrated, 4.8.2.1
connectivity behavior, 11.1, 11.1.2 to 11.1.2.3
controlling hubs, 7.1.7.7
defined, 2.0 *glossary*
downstream ports, 11.5 to 11.5.1.15
dynamic insertion and removal role, 4.6.1, 4.6.2
embedded hubs, 4.8.2.2
enumeration handling, 11.12.6
fault detection and recovery, 11.1
Hub Controller, 4.8.2.1, 11.1, 11.12 to 11.12.6 (*See also* Hub Controller)
hub drivers, 10.3.1
Hub Repeater, 4.8.2.1, 7.3.2 *Table 7-11,* 11.1, 11.7 to 11.7.6 (*See also* Hub Repeater)
hub tier, 2.0 *glossary*

Microchip Ex. 1018, p. 621
Microchip v. Aptiv
IPR2024-00494

hubs *(Continued)*
intermediate hubs, 7.1.7.7
internal ports, 11.4 to 11.4.4
labeling on port connectors, 11.5.3.1
overview, 4.8.2.1, 11.1 to 11.1.2.3
port indicators, 11.5.3 to 11.5.3.1
power management, 11.1
bus-powered hubs, 4.3.1, 7.2.1.1
hub port power control, 11.11
multiple gangs, 11.11.1
over-current reporting and recovery, 11.12.5
power source and sink requirements, 7.2.1
self-powered hubs, 7.2.1.2
surge limiting, 7.2.4.1
requests, 11.24 to 11.24.2.13
root hubs, 2.0 *glossary*
signaling and timing
high-speed rise and fall times, 7.1.2.2
high-speed signal isolation, 8.6.5
high-speed support, 7
host behavior at end-of-frame, 11.3 to 11.3.3
hub chain jitter, 11.2.1
hub differential delay, 7.3.3 *Figure 7-52*
hub EOP delay and EOP skew, 7.3.3 *Figure 7-53*
hub event timings, 7.3.2 *Table 7-13*
hub frame timer, 11.2 to 11.2.5.2
hub (micro)frame timer, 11.2
hub signaling timings, 7.1.14 to 7.1.14.2
hub switching skews, 7.1.9.1
jitter budget table, 7.1.15.1
low-speed keep-alive strobe, 7.1.7.1, 11.8.4.1
power-on and connection events timing, 7.1.7.3
reset signaling, 7.1.7.5
resume signaling, 7.1.7.7
signaling delays, 7.1.14.1
suspend and resume signaling, 11.9
tracking frame and microframe intervals, 7.1.12
sync pattern, 7.1.10
test mode support, 7.1.20
Transaction Translator, 4.8.2.1, 11.1, 11.14 to 11.14.2.3
upstream ports, 11.6 to 11.6.4.6
Hub State Machine, 11.1.1
Hub Status field, 11.24.2.6
humidity life standards, 6.7 *Table 6-7*
hybrid powered hubs, 7.2.1.2
hysteresis in single-ended receivers, 7.1.4.1

I

iConfiguration field
configuration descriptors, 9.6.3, 11.23.1
other speed configuration descriptors, 9.6.4, 11.23.1
Icon for USB plugs and receptacles, 6.5, 6.5.1
Idle bus state
data signaling overview, 7.1.7.4.1, 7.1.7.4.2
downstream facing ports in high-speed, 7.1.7.6.1, 7.1.7.6.2
high-speed driver characteristics and, 7.1.1.3
high-speed signaling, 7.1
high-speed TDR measurements, 7.1.6.2
hub connectivity and, 11.1.2.1
Idle-to-K state transition, 7.1.14.1
NRZI data encoding, 7.1.8
signaling levels and, 7.1.7.1, 7.1.7.2
idle pipes, 5.3.2, 10.5.2.2
idProduct field (device descriptors), 9.6.1
idVendor field (device descriptors), 9.6.1
iInterface field (interface descriptors), 9.6.5, 11.23.1
iManufacturer field (device descriptors), 9.6.1
impedance
cable impedance tests, 6.7 *Table 6-7*
detachable cable assemblies, 6.4.1
differential cable impedance, 7.1, 7.3.2 *Table 7-12*
Exception Window, 7.1.6.2
full-speed connections, 7.1.1.1
high-/full-speed captive cable assemblies, 6.4.2
high-speed driver characteristics, 7.1.1.3
input impedance, 7.3.2 *Table 7-7*
input impedance of ports, 7.1.6.1
Termination Impedance, 7.1.6.2
test mode, 7.1.20
Through Impedance, 7.1.6.2
zero impedance voltage sources, 7.1.1
Implementers Forum, 1.4, 2.0 *glossary*
implementer viewpoints of data flow models, 5.1
implicit feedback for transfers, 5.12.4.3, 9.6.5
Inactive state, 11.4, 11.4.1, 11.6.4, 11.6.4.1, 11.7.1.4.1
in-band signaling, 10.1.2
incident rise times, 7.1.6.2
indicators
descriptors, 11.23.2.1
lights on devices, 11.5.3 to 11.5.3.1
port status indicators, 11.24.2.7.1, 11.24.2.7.1.10
selectors, 11.24.2.12

Microchip Ex. 1018, p. 622
Microchip v. Aptiv
IPR2024-00494

initial frequency inaccuracies, 7.1.11
initialization of USBD, 10.5.1.1
initial states, 8.5, 11.15
injection molded thermoplastic insulator material, 6.5.3.1, 6.5.4.1
inner shielding in cables, 6.6.2
IN PID, 8.3.1 *Table 8-1*
  aborting IN transactions, 11.18.6.1
  ACK handshake and, 8.4.5
  ADDR field, 8.3.2.1
  bit times and, 11.3.3
  bulk transfers, 8.5.2, 8.5.2 *Figure 8-33,* 8.5.2 *Figure 8-34,* 11.17.1, 11.17.2
  complete-split flow sequence, 11.20.1
  control transfers, 8.5.2 *Figure 8-33,* 8.5.2 *Figure 8-34,* 8.5.3, 8.5.3.1, 11.17.1, 11.17.2
  ENDP field, 8.3.2.2
  error handling on last data transaction, 8.5.3.3
  function response to, 8.4.6.1
  high bandwidth transactions and, 5.9.2
  host response to, 8.4.6.2
  interrupt transfers, 8.5.2 *Figure 8-33,* 8.5.2 *Figure 8-34,* 8.5.4, 11.20.4
  intervals between IN token and EOP, 11.3.3
  isochronous transfers, 8.5.5, 8.5.5 *Figure 8-42,* 8.5.5 *Figure 8-43,* 11.21 to 11.21.4
  low-speed transactions, 8.6.5
  NAK handshake and, 8.4.5
  prebuffering data, 5.12.5
  scheduling IN transactions, 11.18.4
  split transaction conversion, 8.4.2.1
  split transaction examples, A.2, A.4, A.6
  STALL handshake and, 8.4.5
  start-split flow sequence, 11.20.1
  state machines
    bulk/control state machines, 8.5.2 *Figure 8-33,* 8.5.2 *Figure 8-34,* 11.17.2
    interrupt state machines, 8.5.2 *Figure 8-33,* 8.5.2 *Figure 8-34,* 11.20.2
    isochronous state machines, 8.5.5 *Figure 8-42,* 8.5.5 *Figure 8-43,* 11.21.2
  token CRCs, 8.3.5.1
  token packets, 8.4.1
  Transaction Translator response generation, 11.18.5
input capacitance, 7.1.2.1, 7.3.2 *Table 7-7*
input characteristics (signaling), 7.1.6.1
input impedance, 7.3.2 *Table 7-7*
input impedance of ports, 7.1.6.1
input levels, 7.3.2 *Table 7-7*
inputs (Series "B" receptacles), 6.2
input sensitivity of differential input receivers, 7.1.4.1
input transitions in state machines, 8.5
inrush current limiting
  bus-powered hubs, 7.2.1.1
  dynamic attach and detach, 7.2.4.1
  remote wakeup and, 7.2.3
  self-powered functions, 7.2.1.5
  suspend/resume and, 7.2.3
inserting devices. *See* dynamic insertion and removal
insertion force standards, 6.7 *Table 6-7*
instancing of USBD, 10.5
insulation
  cables, 6.6.2
  insulator materials, 6.5.3.1, 6.5.4.1
  resistance standards, 6.7 *Table 6-7*
interconnect model, 4.1, 5.12.4.4
interface class codes, 9.6.5
INTERFACE descriptor, 9.4 *Table 9-5*
interface descriptors
  GetDescriptor() request, 9.4.3
  hubs, 11.23.1
  INTERFACE_POWER descriptor, 9.4 *Table 9-5*
  overview, 9.6.5
interface numbers, 9.2.3, 9.6.5
INTERFACE_POWER descriptor, 9.4 *Table 9-5*
interfaces
  alternate interfaces, 9.2.3, 10.5.2.10
  alternate settings, 9.6.5
  in configuration descriptors, 9.6.3, 9.6.4
  defined, 9.2.3
  in device class definitions, 9.7.2
  as endpoint sets, 5.3
  getting interface status, 9.4.4, 9.4.5
  host-system interface, 10.2.9
  interface class codes, 9.6.5
  interface descriptors, 9.4.3, 9.6.5, 11.23.1
  interface numbers, 9.2.3, 9.6.5
  interface subclass codes, 9.6.5
  plug interface and mating drawings, 6.5.4
  setting interface state, 9.4.10, 10.5.2.1
  specifying in *wIndex* field, 9.3.4
interfaces of plugs, 6.5.3
interface state control, 10.5.2.1
interface subclass codes, 9.6.5
interlayer communications model, 4.1, 10.1.1
intermediate hubs, 7.1.7.7
internal clock source jitter, 7.1.13.1.1
internal ports
  Full Suspend (Fsus) state, 11.4.3
  Generate Resume (GResume) state, 11.4.4
  Inactive state, 11.4.1
  Suspend Delay state, 11.4.2
internal port state machine, 11.4
interpacket delay, 7.1.18 to 7.1.18.2
interpacket gaps, 11.18.2

Microchip Ex. 1018, p. 623
Microchip v. Aptiv
IPR2024-00494

interrupt endpoints, 11.12.1
interrupt requests, defined, 2.0 *glossary*
interrupt transfers. *See also* periodic transactions
  aborting, 11.18.6.1
  bus access constraints, 5.7.4
  data format, 5.7.1
  data sequences, 5.7.5
  defined, 2.0 *glossary,* 5.4
  direction, 5.7.2
  flow sequences, 11.18.8, 11.20 to 11.20.4
  full-speed transfer limits, 5.7.4
  high- bandwidth endpoints, 5.9.1
  high-speed transfer limits, 5.7.4 *Table 5-8*
  low-speed transfer limits, 5.7.4
  multiple transactions and, 9.6.6
  overview, 4.7, 4.7.3, 5.7
  packet size, 5.7.3, 9.6.6
  periodic transactions, 11.18 to 11.18.8
  required bus access period, 5.7.4
  scheduling and buffering, 11.14.2.1
  sequencing, 11.20.3, 11.20.4
  split transactions
    examples, A.3, A.4
    host handling of, 5.10
    notation for, 11.15
  state machines, 8.5.2, 11.20 to 11.20.4
  transaction format, 8.5.4
  transaction organization within IRPs, 5.11.2
  Transaction Translator response generation, 11.18.5
  USBD pipe mechanism responsibilities, 10.5.3.1.2
intervals
  debounce intervals in connection events, 7.1.7.3
  frame and microframe intervals, 7.1.12, 7.3.2 *Table 7-8,* 7.3.2 *Table 7-9,* 7.3.2 *Table 7-10*
  Resetting state and Resuming state intervals, 11.5.1.10
  resume and recovery intervals for devices, 9.2.6.2
  service and polling intervals, 2.0 *glossary,* 9.6.6, 10.3.3
  timeout intervals, 8.7.3
  between IN token and EOP, 11.3.3
interwoven tinned copper wire, 6.6.2
IN transactions. *See* IN PID
I/O buffers. *See* buffers
I/O Request Packets. *See* IRPs
iProduct field (device descriptors), 9.6.1
IRPs. *See also* requests; transfers
  aborting/retiring, 5.3.2, 10.5.3.2.1
  class-specific requests, 9.2.6.5
  client software role in, 5.11.1.1
  defined, 2.0 *glossary,* 5.3.2
  HCD tracking of, 5.11.1.3
  multiple data payloads in, 5.3.2
  pipes and, 5.3.2
  queuing IRPs, 10.5.3.2.3
  request processing overview, 9.2.6
  reset/resume recovery time, 9.2.6.2
  set address processing, 9.2.6.3
  STALLS and, 5.3.2
  standard device requests, 9.2.6.4, 9.4 to 9.4.11
  timing, 9.2.6.1, 9.2.6.3, 9.2.6.4
  transaction organization within IRPs, 5.11.2
  USB device requests, 9.3 to 9.3.5
  USBD role in, 10.1.1
IRQs, defined, 2.0 *glossary*
iSerialNumber field (device descriptors), 9.6.1
isochronous data, defined, 2.0 *glossary*
isochronous devices, defined, 2.0 *glossary*
isochronous sink endpoints, defined, 2.0 *glossary*
isochronous source endpoints, defined, 2.0 *glossary*
isochronous transactions. *See also* periodic transactions
isochronous transfers
  buffering, 5.12.8, 11.14.2.1
  bus access constraints, 5.6.4
  clock model, 5.12.2
  clock synchronization, 5.12.3
  connectivity, 5.12.4.4
  data format, 5.6.1
  data prebuffering, 5.12.5
  data sequences, 5.6.5
  defined, 2.0 *glossary,* 5.4
  direction, 5.6.2
  endpoint attributes, 9.6.6
  endpoint synchronization frame, 9.4.11
  error handling, 5.12.7
  feedback, 5.12.4.2
  high-bandwidth endpoints, 2.0 *glossary,* 5.9.2
  illustrated, 5.12.2
  implicit feedback, 5.12.4.3, 9.6.5
  isochronous device framework, 5.12.4
  multiple transactions and, 9.6.6
  non-USB example isochronous application, 5.12.1
  non-zero data payload, 5.6.3

Microchip Ex. 1018, p. 624
Microchip v. Aptiv
IPR2024-00494

isochronous transfers *(Continued)*
overview, 4.7.4, 5.6, 11.21 to 11.21.4
packet size, 5.6.3, 9.6.6
periodic transactions, 11.18 to 11.18.8
rate matching, 5.12.8
scheduling, 11.14.2.1
sequencing, 11.21.3, 11.21.4
SOF tracking, 5.12.6
special considerations, 5.12 to 5.12.8
specifying required bus access period, 5.6.4
split transactions
examples, A.5, A.6
flow sequences, 11.18.8
host controller and, 5.10
notation for, 11.15
state machines, 8.5.5, 11.21 to 11.21.4
synchronization types, 5.12.4.1.1, 5.12.4.1.2, 5.12.4.1.3
transaction format, 8.5.5
transaction organization within IRPs, 5.11.2
Transaction Translator response generation, 11.18.5
USBD pipe mechanism responsibilities, 10.5.3.1.1
USB features and, 3.3
USB System Software role, 4.9
ISO transfers. *See* isochronous transfers
ITCW (interwoven tinned copper wire), 6.6.2

J

J bus state
data signaling overview, 7.1.7.4.1
high-speed signaling and, 7.1, 7.1.1.3
J-to-K state transition, 7.1.14.1
NRZI data encoding, 7.1.8
signaling levels and, 7.1.7.1, 7.1.7.2
test mode, 7.1.20
J signal/event
differential transmissions, 11.6.1
receiver state machine, 11.6.3 *Table 11-8*
transmission sequence, 7
transmitter state machine, 11.6.4 *Table 11-9*
jacketing in cables, 6.6.2
jitter
clock jitter, 5.12.3
cumulative jitter in high-speed signaling, 7.1.14.2
data source jitter, 7.1.13.1 to 7.1.13.1.2
defined, 2.0 *glossary*
differential data jitter, 7.3.3 *Figure 7-49*
differential jitter, 7.3.3 *Figure 7-52*
frame and microframe jitter, 11.2.4
full-speed source electrical characteristics, 7.3.2 *Table 7-9*
high-speed source electrical characteristics, 7.3.2 *Table 7-8*
jitter *(Continued)*
host jitter, 11.2.1
hub chain jitter, 11.2.1
hub/repeater electrical characteristics, 7.3.2 *Table 7-11*
internal clock source jitter, 7.1.13.1.1
low-speed source electrical characteristics, 7.3.2 *Table 7-10*
in non-USB isochronous application, 5.12.1
output driver jitter, 7.1.15.1
receiver data jitter, 7.1.15 to 7.1.15.2, 7.3.3 *Figure 7-51*
service jitter, 2.0 *glossary*
test mode, 7.1.20
Transaction Translator frame clock and, 11.18.3
joint symbols in state machine transitions, 8.5, 11.15

K

K bus state
data signaling overview, 7.1.7.4.1
high speed signaling and, 7.1.1.3
high-speed signaling and, 7.1
Idle-to-K state transition, 7.1.14.1
K-to-J state transition, 7.1.14.1
NRZI data encoding, 7.1.8
signaling levels and, 7.1.7.1, 7.1.7.2, 7.1.7.4.2
test mode, 7.1.20
K signal/event
differential transmissions, 11.6.1
downstream port state machine, 11.5 *Table 11-5*
receiver state machine, 11.6.3 *Table 11-8*
transmission sequence, 7
transmitter state machine, 11.6.4 *Table 11-9*
kB/S and kb/S, defined, 2.0 *glossary*
keep-alive strobe, 7.1.7.1, 7.1.7.6, 11.8.4.1
keyed connector protocol, 6.2

L

labeling on port connectors, 11.5.3.1
LANGID code array, 9.6.7
language IDs in string descriptors, 9.6.7
latency
constraints for transfers, 5.4
packets, 11.7.1.3
latest host packet, 11.3.1
least-significant bit (LSb), 2.0 *glossary,* 8.1
least-significant byte (LSB), 2.0 *glossary,* 8.1
length of cables, 6.4.1, 6.4.2, 6.4.3
listing, UL listing for cables, 6.6.5
little endian order, 2.0 *glossary,* 8.1
LOA, 2.0 *glossary,* 8.7.4
load capacitance, 6.4.3, 6.7 *Table 6-7*
Local Power Source field, 11.24.2.6

Microchip Ex. 1018, p. 625
Microchip v. Aptiv
IPR2024-00494

Local Power Status Change field, 11.24.2.6
Local Power Status field, 11.24.2.7.1.6
local power supplies, 7.2.1.2, 7.2.1.5
locking hub frame timer, 11.2.3
Lock signal/event, 11.7.2.3 *Table 11-11*
logical bus topology, 5.2, 5.2.4
logical devices
  in bus topology, 5.2.2
  as collections of endpoints, 5.3
  detailed communication flow illustrated, 5.3
  unique addresses and endpoints, 5.3.1
Logical Power Switching Mode field, 11.11, 11.11.1, 11.23.2.1
logo location on connectors, 6.5.1
LOI, 6.5.3.1, 6.5.4.1
loss, cable, 7.1.17
loss of bus activity. *See* LOA
loss of synchronization, 11.22.1
low level contact resistance standards, 6.7 *Table 6-7*
low-power bus-powered functions, 7.2.1, 7.2.1.3
low-power hubs, 7.2.2
low-power ports, 7.2.1
low-speed buffers, 7.1.2.1
low-speed cables
  cable environmental characteristics, 6.6.4
  captive cable assemblies, 6.4.3
  configuration overview, 6.6
  construction, 6.6.2
  description, 6.6.1
  detachable cables, 6.4.4
  listing, 6.6.5
  specifications, 6.3
  standards for, 6.6.3, 6.7
low-speed driver characteristics, 7.1.1.2, 7.1 *Table 7-1*
low-speed functions and hubs
  cable and resistor connections, 7.1.5.1
  connect detection, 7.1.7.3
  control transfers and, 5.5.3, 5.5.4, 5.5.4 *Table 5-1*
  data-rate tolerance, 7.1.11
  defined, 2.0 *glossary*
  detachable cables and, 6.4.1
  detecting, 11.24.2.7.1.7
  full- *vs.* low-speed port behavior, 11.8.4
  getting port status, 11.24.2.7.1
  high-/full-speed captive cable assemblies, 6.4.2
  hub/repeater electrical characteristics, 7.3.2 *Table 7-11*
  hub support for, 11.1
  input capacitance, 7.1.6.1
  interrupt transfers and, 5.7.3, 5.7.4 *Table 5-6*
  low-speed captive cable assemblies, 6.4.3
  maximum data payload, 8.4.4
low-speed functions and hubs *(Continued)*
  optional endpoints, 5.3.1.2
  in physical bus topology, 5.2.3
  signal termination, 7.1.5.1
  speed detection and, 11.8.2
  Transmit state and, 11.5.1.7
low-speed keep-alive strobe, 7.1.7.1, 7.1.7.6, 11.8.4.1
low-speed signaling
  babble and loss of activity recovery, 8.7.4
  bus transactions and, 4.4
  calculating transaction times, 5.11.3
  data rates, 4.2.1
  data signaling overview, 7.1.7.4.1
  data source jitter, 7.1.13.1.1
  data toggle synchronization and retry, 8.6.5
  defined, 2.0 *glossary*
  differential receivers, 7.1 *Table 7-1*
  downstream and upstream facing ports, 7.1
  driver characteristics, 7.1.1, 7.1.2.3, 7.1 *Table 7-1*
  EOP width, 7.1.13.2.1
  errors, 8.6.4
  host behavior at end-of-frame, 11.3 to 11.3.3
  hub class descriptors and, 11.23.1
  intervals between IN token and EOP, 11.3.3
  J and K states, 7.1.7.1
  jitter budget table, 7.1.15.1
  low-speed loads, 7.1.2.1
  propagation delays, 7.1.14.1
  receiver characteristics, 7.1.4.1
  reset signaling, 7.1.7.5
  scheduling, 11.14.2.3
  speed detection, 9.1.1.3
  transactions illustrated, 8.6.5
  Transaction Translator, 4.8.2.1, 11.18.5
low-speed source electrical characteristics, 7.3.2 *Table 7-10*
LSb and LSB
  in bit ordering, 8.1
  defined, 2.0 *glossary*
LS signal/event, 11.5
lumped capacitance guidelines for transceivers, 7.1.6.2

M

male plug contact materials, 6.5.4.3
manual port color indicators, 11.5.3
Manufacturer's logo location, 6.5.1
Manufacturer's names in device descriptors, 9.6.1
mapping physical and virtual devices, 5.12.4.4
marking on cables, 6.6.2
master clock, 5.12.1

Microchip Ex. 1018, p. 626
Microchip v. Aptiv
IPR2024-00494

material requirements
cables, 6.6 to 6.6.5
connectors, 6.5 to 6.5.4.3
mating area materials, 6.5.3.3, 6.5.4.3
MaxPower field, 9.6.4, 11.13
MB/S, defined, 2.0 *glossary*
Mb/S, defined, 2.0 *glossary*
MDATA PID
in data packets, 8.4.4
defined, 8.3.1 *Table 8-1*
high-bandwidth endpoints and, 2.0 *glossary,* 5.9.2
interrupt IN sequencing, 11.20.4
Transaction Translator response generation, 11.18.5
measurement planes in speed signaling eye patterns, 7.1.2.2
mechanical specifications, 6, 6.1
applicable documents, 6.7.1
architectural overview, 6.1
cable assembly, 6.4 to 6.4.4
cables, 6.3, 6.6 to 6.6.5
connectors, 6.2, 6.5 to 6.5.4.3
overview, 4.2.2, 6
PCB reference drawings, 6.9
standards for, 6.7, 6.7.1
USB grounding, 6.8
message pipes. *See also* control pipes; pipes
in bus protocol overview, 4.4
defined, 2.0 *glossary,* 5.3.2
overview, 5.3.2.2
microframe pipelines. *See under* frames and microframes
microframes. *See* frames and microframes
microframe timers. *See* frame and microframe timers
microphone non-USB isochronous application, 5.12.1
microphone USB isochronous application, 5.12.2
"middle" encoding, 11.18.4
modifying device configuration, 10.5.4.1.3
monotonic transitions, 7.1.2.1, 7.1.2.2
most-significant bit (MSb), 2.0 *glossary,* 8.1
most-significant byte (MSB), 2.0 *glossary,* 8.1, 11.24.2.13
MSb and MSB, 2.0 *glossary,* 8.1
multiple gangs, hubs and, 11.11.1
multiple Transaction Translators, 11.14.1.3, 11.23.1, 11.24.2.8

N

NAKs
in bulk transfers, 8.5.2, 11.17.1
busy endpoints, 5.3.2
in control transfers, 8.5.3.1, 11.17.1
data corrupted or not accepted, 8.6.3
defined, 2.0 *glossary*
function response to IN transactions, 8.4.6.1
function response to OUT transactions, 8.4.6.3
high bandwidth transactions and, 5.9.1
in interrupt transfers, 5.7.4, 8.5.4, 11.20.4
NAK limiting via Ping flow control, 8.5.1, 8.5.1.1
non-periodic transactions, 11.14.2.2, 11.17.1
overview, 8.3.1 *Table 8-1,* 8.4.5
Ready/NAK status, 11.15
test mode, 7.1.20
NEC Article 800 for communications cables, 6.6.4
next frame in hub timing, 11.2.3.1
No Acknowledge packet. *See* NAKs
nominal cable diameter, 6.6.2
nominal cable temperatures, 6.6.4
nominal twist ratio in signal pair, 6.6.2
non-acceptable cables, 6.4.4
non-periodic transactions
buffers, 11.14.1, 11.14.2.2, 11.19
failures, 11.17.5
overview, 11.17 to 11.17.5
scheduling, 11.14.2.2
Non Return to Zero Invert. *See* NRZI encoding
non-twisted power pair in cables, 6.6.1, 6.6.2
non-USB isochronous application, 5.12.1
non-zero data payloads, 5.6.3
Not Configured state, 11.5, 11.5.1.1
Not Packet state, 11.7.1.4.4
NRZI encoding, 7.1.8
bit stuffing, 7.1.9
defined, 2.0 *glossary*
in electrical specifications overview, 4.2.1
high-speed signaling and, 7.1
sync pattern, 7.1.7.4.2, 7.1.10
NYET handshake
aborting, 11.18.6.1
bulk/control transactions, 8.5.2, 11.17.1
defined, 8.3.1 *Table 8-1,* 8.4.5
isochronous transactions, 11.21.1
Ping flow control and, 8.5.1
Transaction Translator response generation, 11.18.5

Microchip Ex. 1018, p. 627
Microchip v. Aptiv
IPR2024-00494

O

objects, defined, 2.0 *glossary*
offsets between host and hub, 11.2
Old status, 11.15
one-way propagation delay, 7.1.16, 7.3.2 *Table 7-9,* 7.3.2 *Table 7-10*
open architecture, USB development and, 1.2
open-circuit voltage, 7.1.1
operating systems
  companion specifications, 10.6
  device configuration, 10.3.1
  interaction with USBD, 10.5
  passing preboot control to, 10.5.5
operating temperatures for cables, 6.6.4
operations, generic USB device operations, 9.2 to 9.2.7
optional edge rate control capacitors, 7.1.6.1
OTHER_SPEED_CONFIGURATION descriptor, 9.2.6.6, 9.4 *Table 9-5*, 9.4.3, 9.6.2, 9.6.4
(other speed) device_qualifier descriptor, 9.2.6.6
other_speed requests, 9.2.6.6
out-of-band signaling, 10.1.2
OUT PID, 8.3.1 *Table 8-1*
  aborting OUT transactions, 11.18.6.1
  ACK handshake and, 8.4.5
  ADDR field, 8.3.2.1
  in bulk transfers, 8.5.2, 8.5.2 *Figure 8-31,* 8.5.2 *Figure 8-32,* 11.17.1, 11.17.2
  complete-split flow sequence, 11.20.1
  in control transfers, 8.5.2 *Figure 8-31,* 8.5.2 *Figure 8-32,* 8.5.3, 8.5.3.1, 11.17.1, 11.17.2
  in data toggle, 8.6.1
  ENDP field, 8.3.2.2
  function response to OUT transactions, 8.4.6.3
  high bandwidth transactions and, 5.9.2
  high-speed PING flow control protocol, 5.8.4
  in interrupt transfers, 8.5.2 *Figure 8-31,* 8.5.2 *Figure 8-32,* 8.5.4, 11.20.3
  in isochronous transfers, 8.5.5, 8.5.5 *Figure 8-40,* 8.5.5 *Figure 8-41,* 11.21 to 11.21.4
  NAK handshake and, 8.4.5
  OUT/DATA transactions, 8.5.1, 8.5.1.1
  PING flow control and OUT transactions, 8.5.1, 8.5.1.1
  prebuffering data, 5.12.5
  scheduling OUT transactions, 11.18.4
  split transaction conversion, 8.4.2.1, 8.4.2.2
  split transaction examples, A.1, A.3, A.5
  STALL handshake and, 8.4.5
  start-split flow sequence, 11.20.1
OUT PID *(Continued)*
  state machines
    bulk/control state machines, 8.5.2 *Figure 8-31,* 8.5.2 *Figure 8-32,* 11.17.2
    interrupt state machines, 8.5.2 *Figure 8-31,* 8.5.2 *Figure 8-32,* 11.20.2
    isochronous state machines, 8.5.5 *Figure 8-40,* 8.5.5 *Figure 8-41,* 11.21.2
  token CRCs, 8.3.5.1
  in token packets, 8.4.1
  Transaction Translator response generation, 11.18.5
output driver jitter, 7.1.15.1
output levels, 7.3.2 *Table 7-7*
output receptacles, 6.2
output rise and fall times, 7.1.2.1
output transitions in state machines, 8.5
outside plating, 6.5.3.2, 6.5.4.2
over-current conditions
  C_PORT_OVER-CURRENT bit, 11.24.2.7.2.4
  getting port status, 11.24.2.7.1
  over-current gangs, 11.11.1
  over-current protection in self-powered hubs, 7.2.1.2.1
  port indicators, 11.5.3
  PORT_OVER-CURRENT bit, 11.24.2.7.1.4
  port status change bits, 11.24.2.7.2
  protection mode descriptors, 11.23.2.1
  reporting and recovery, 11.12.5
  signaling, 11.11.1
Over-current Indicator Change field, 11.24.2.6
Over-current Indicator field, 11.24.2.6
Over-current Reporting Mode field, 11.11.1
Over-current signal/event, 11.5 *Table 11-5*
over-sampling state machine DPLLs, 7.1.15.1
oxygen index, 6.5.3.1, 6.5.4.1

P

packet buffers, defined, 2.0 *glossary*
packet field formats, 8.3
  address fields, 8.3.2 to 8.3.2.2
  cyclic redundancy checks (CRC), 8.3.5 to 8.3.5.2
  data field, 8.3.4, 8.4.4
  frame number field, 8.3.3, 8.4.3
  packet identifier field, 8.3.1 (*See also* PIDs)

Microchip Ex. 1018, p. 628
Microchip v. Aptiv
IPR2024-00494

packet formats
data packets, 4.4, 8.3 to 8.3.5.2, 8.4.4, 8.5.2, 8.5.5
handshake packets, 8.4.5 (*See also* handshakes)
handshake responses, 8.4.6 to 8.4.6.4
overview, 8.4
packet field formats, 8.3 to 8.3.5.2
SOF packets, 5.12.2, 5.12.4.1.1, 5.12.4.1.2, 8.4.3
token packets, 4.4, 8.3.5.1, 8.4.1, 8.5.2, 8.5.5
packet identifier field (PID). *See* PIDs
packet IDs. *See* PIDs
packet nullification, 11.3.2
packets. *See also* packet field formats; packet formats; packet size
bit stuffing, 7.1.9 to 7.1.9.2
blocking in Collision conditions, 11.8.3
bus protocol overview, 4.4
data packets defined, 4.4
data signaling overview, 7.1.7.4 to 7.1.7.4.2
defined, 2.0 *glossary*
error detection and recovery, 8.7 to 8.7.4
handshake packets defined, 4.4
high-speed packet repeaters, 11.7.1 to 11.7.1.4.4
hub connectivity and, 11.1.2.1
inter-packet delay, 7.1.18 to 7.1.18.2
one transaction per frame in isochronous transfers, 5.12.7
packet field formats, 8.3 to 8.3.5.2
packet formats, 8.4 to 8.4.6.4
packet nullification, 11.3.2
packet size (*See* packet size)
packet voltage levels, 7.1.7.4.1
short packets, 5.3.2
splitting samples across packets, 5.12.8
SYNC field, 8.2
test mode packets, 7.1.20
token packets defined, 4.4
total latency, 11.7.1.3
transaction formats, 8.5 to 8.5.5
packet size
in buffering calculations, 5.12.8
bulk transfer constraints, 5.8.3
characteristics for transfers, 5.4
control transfer constraints, 5.5.3
determining missing packet size, 5.12.7
in device descriptors, 9.6.1
in device_qualifer descriptor, 9.6.2
in endpoint descriptors, 9.6.6
interrupt transfer constraints, 5.7.3
isochronous transfer constraints, 5.6.3
packet voltage levels, 7.1.7.4.1
parasitic loading, 7.1, 7.1.1.3, 7.1.6.2
partitioning of power, 7.2.1.1
paths, notations for, 11.15
pattern synchronization, 9.4.11
PBT (polybutylene terephthalate), 6.5.3.1, 6.5.4.1
PCB reference drawings, 6.9
PC industry, USB and, 3.3
PC-to-telephone interconnects, 1.1
pending start-splits, 11.18.6.2
Pending status, 11.15, 11.17.1
pending transactions, 11.18.6
performance criteria for electrical, mechanical and environmental compliance, 6.7
periodic buffer sections, 11.14.1
periodic transactions. *See also* interrupt transfers; isochronous transfers
after loss of synchronication, 11.22.2
buffer space required, 11.14.1, 11.19
handling requirements, 11.18.6 to 11.18.6.3
interrupt transaction sequencing, 11.20.3, 11.20.4
IN transaction sequencing, 11.20.4, 11.21.4
isochronous split transactions, 11.21 to 11.21.4
OUT transaction sequencing, 11.20.3, 11.21.3
overview, 11.18 to 11.18.8
periodic transaction pipelines, 11.14.2.1
scheduling and buffering, 11.14.2.1
split transaction flow sequences, 11.18.8
peripheral devices, 4.8.2.2. *See also* devices; functions
per-port current limiting, 11.12.5
per-port power switching, 11.11, 11.12.5
PET (polyethylene terephthalate), 6.5.3.1, 6.5.4.1
phase delay for SOF packets, 11.3.1
phase differences, clock synchronization and, 5.12.3
phase-locked clocks, 5.12.3
Phase Locked Loop, defined, 2.0 *glossary*
phases, defined, 2.0 *glossary*
Physical and Environmental Performance Properties of Insulation and Jacket for Telecommunication Wire and Cable, 6.7.1
physical bus topology, 5.2, 5.2.3, 6.1
physical devices
connectivity illustrated, 5.12.4.4
defined, 2.0 *glossary*
as implementation focus area, 5.1
logical components in bus topology, 5.2.2
physical interface, 4.2 to 4.2.2
physical shock standards, 6.7 *Table 6-7*
PID errors, defined, 8.3.1

Microchip Ex. 1018, p. 629
Microchip v. Aptiv
IPR2024-00494

PIDs
corrupted PIDs, 8.3.1
in data packets, 8.4.4
data PIDs
DATA0/DATA1/DATA2 PIDs
in bulk transfers, 5.8.5, 8.5.2
comparing sequence bits, 8.6.2
in control transfers, 8.5.3
in data packets, 8.4.4
high-bandwidth transactions and, 5.9.1, 5.9.2
high-speed data PIDs, 8.3.1 *Table 8-1*
in interrupt transactions, 5.7.5, 8.5.4, 11.20.4
synchronization and, 8.6
Transaction Translator response generation, 11.18.5
error handling in high-speed transactions, 5.12.7
high bandwidth transactions and, 5.9.2
MDATA PIDs
in data packets, 8.4.4
defined, 8.3.1 *Table 8-1*
high-bandwidth endpoints and, 5.9.2
interrupt IN sequencing, 11.20.4
Transaction Translator response generation, 11.18.5
defined, 2.0 *glossary*
full- *vs.* low-speed port behavior, 11.8.4
in handshake packets, 8.4.5
handshake PIDs, 8.3.1 *Table 8-1*
ACK PIDs
in bulk transfers, 8.5.2, 11.17.1
in control transfers, 8.5.3, 8.5.3.1, 11.17.1
corrupted ACK handshake, 8.5.3.3, 8.6.4
in data toggle, 8.6, 8.6.1, 8.6.2
defined, 2.0 *glossary*
function response to OUT transactions, 8.4.6.3
host response to IN transactions, 8.4.6.2
overview, 8.3.1 *Table 8-1,* 8.4.5
PING flow control and OUT transactions, 8.5.1, 8.5.1.1
Ready/ACK status, 11.15
in request processing, 9.2.6
NAK PIDs
in bulk transfers, 8.5.2, 11.17.1
busy endpoints, 5.3.2
in control transfers, 8.5.3.1, 11.17.1
data corrupted or not accepted, 8.6.3
defined, 2.0 *glossary*
function response to IN transactions, 8.4.6.1
function response to OUT transactions, 8.4.6.3

PIDs *(Continued)*
handshake PIDs
NAK PIDs
high bandwidth transactions and, 5.9.1
in interrupt transfers, 5.7.4, 8.5.4, 11.20.4
NAK limiting via Ping flow control, 8.5.1, 8.5.1.1
non-periodic transactions, 11.14.2.2, 11.17.1
overview, 8.3.1 *Table 8-1,* 8.4.5
Ready/NAK status, 11.15
test mode, 7.1.20
NYET PIDs
aborting, 11.18.6.1
bulk/control transactions, 8.5.2, 11.17.1
defined, 8.3.1 *Table 8-1,* 8.4.5
isochronous transactions, 11.21.1
Ping flow control and, 8.5.1
Transaction Translator response generation, 11.18.5
STALL PIDs
in bulk transfers, 5.8.5, 8.5.2, 11.17.1
in control transfers, 8.5.3.1, 11.17.1
data corrupted or not accepted, 8.6.3
functional and commanded stalls, 8.4.5
function response to IN transactions, 8.4.6.1
function response to OUT transactions, 8.4.6.3
in interrupt transfers, 5.7.5, 8.5.4, 11.20.4
overview, 8.3.1 *Table 8-1,* 8.4.5
protocol stalls, 8.4.5
Ready/Stall status, 11.15
Request Error responses, 9.2.7
responses to standard device requests, 9.4
returned by control pipes, 8.5.3.4
incorrect PID errors, 11.15
overview, 8.3.1
PID check bits, 8.7.1
PID errors, 8.3.1
special PIDs
ERR PIDs, 8.3.1 *Table 8-1,* 8.4.5
PING PIDs, 8.3.1 *Table 8-1,* 8.3.2.2, 8.3.5.1, 8.4.1, 8.4.5, 8.5.2, 8.5.3.1
PRE PIDs
inter-packet delays and, 7.1.18.1
low-speed port behavior and, 11.8.4
low-speed transactions, 8.6.5
overview, 8.3.1 *Table 8-1*
Transmit state and, 11.5.1.7
Reserved PIDs, 8.3.1 *Table 8-1*
SPLIT PIDs, 8.3.1 *Table 8-1,* 8.3.2.1, 8.3.5.1, 8.4.2 to 8.4.2.3
in start-of-frame packets, 8.4.3
in token packets, 8.4.1

Microchip Ex. 1018, p. 630
Microchip v. Aptiv
IPR2024-00494

PIDs *(Continued)*
token PIDs, 8.3.1 *Table 8-1*
OUT PIDs
aborting OUT transactions, 11.18.6.1
ACK handshake and, 8.4.5
ADDR field, 8.3.2.1
in bulk transfers, 8.5.2, 8.5.2 *Figure 8-31,* 8.5.2 *Figure 8-32,* 11.17.1, 11.17.2
complete-split flow sequence, 11.20.1
in control transfers, 8.5.2 *Figure 8-31,* 8.5.2 *Figure 8-32,* 8.5.3, 8.5.3.1, 11.17.1, 11.17.2
in data toggle, 8.6.1
ENDP field, 8.3.2.2
function response to OUT transactions, 8.4.6.3
high bandwidth transactions and, 5.9.2
high-speed PING flow control protocol, 5.8.4
in interrupt transfers, 8.5.2 *Figure 8-31,* 8.5.2 *Figure 8-32,* 8.5.4, 11.20.3
in isochronous transfers, 8.5.5, 8.5.5 *Figure 8-40,* 8.5.5 *Figure 8-41,* 11.21 to 11.21.4
NAK handshake and, 8.4.5
OUT/DATA transactions, 8.5.1, 8.5.1.1
overview, 8.3.1 *Table 8-1*
PING flow control and OUT transactions, 8.5.1, 8.5.1.1
prebuffering data, 5.12.5
scheduling OUT transactions, 11.18.4
split transaction conversion, 8.4.2.1, 8.4.2.2
split transaction examples, A.1, A.3, A.5
STALL handshake and, 8.4.5
start-split flow sequence, 11.20.1
state machines, 8.5.2, 8.5.5, 11.17.2, 11.20.2, 11.21.2
token CRCs, 8.3.5.1
in token packets, 8.4.1
Transaction Translator response generation, 11.18.5
IN PIDs
aborting IN transactions, 11.18.6.1
ACK handshake and, 8.4.5
ADDR field, 8.3.2.1
bit times and, 11.3.3
bulk transfers, 8.5.2, 8.5.2 *Figure 8-33,* 8.5.2 *Figure 8-34,* 11.17.1, 11.17.2
complete-split flow sequence, 11.20.1
control transfers, 8.5.2 *Figure 8-33,* 8.5.2 *Figure 8-34,* 8.5.3, 8.5.3.1, 11.17.1, 11.17.2
ENDP field, 8.3.2.2
error handling on last data transaction, 8.5.3.3
PIDs *(Continued)*
token PIDs
IN PIDs
function response to, 8.4.6.1
high bandwidth transactions and, 5.9.2
host response to, 8.4.6.2
interrupt transactions, 8.5.4
interrupt transfers, 8.5.2 *Figure 8-33,* 8.5.2 *Figure 8-34,* 11.20.4
intervals between IN token and EOP, 11.3.3
isochronous transfers, 8.5.5, 8.5.5 *Figure 8-42,* 8.5.5 *Figure 8-43,* 11.21 to 11.21.4
low-speed transactions, 8.6.5
NAK handshake and, 8.4.5
overview, 8.3.1 *Table 8-1*
prebuffering data, 5.12.5
scheduling IN transactions, 11.18.4
split transaction conversion, 8.4.2.1
split transaction examples, A.2, A.4, A.6
STALL handshake and, 8.4.5
start-split flow sequence, 11.20.1
state machines, 8.5.2, 8.5.5, 11.17.2, 11.20.2, 11.21.2
token CRCs, 8.3.5.1
token packets, 8.4.1
Transaction Translator response generation, 11.18.5
SETUP PIDs
ACK handshake and, 8.4.5
ADDR field, 8.3.2.1
in control transfers, 8.5.3
in data toggle, 8.6.1
ENDP field, 8.3.2.2
function response to SETUP transactions, 8.4.6.4
overview, 8.3.1 *Table 8-1*
token CRCs, 8.3.5.1
in token packets, 8.4.1
SOF PIDs (*See also* SOFs)
frame number field, 8.3.3
frames and microframes, 8.4.3.1
overview, 8.3.1 *Table 8-1*
start-of-frame packets, 8.4.3
PID to PID_invert bits check failure, 11.15
PING flow control protocol
high-speed signaling, 5.8.4
high-speed signaling and, 5.5.4
as limited to high-speed transactions, 8.5.1
NAK limiting and, 8.5.1, 8.5.1.1
NYET handshake and, 8.4.5
PING PIDs, 8.3.1 *Table 8-1,* 8.3.2.2, 8.3.5.1, 8.4.1, 8.4.5, 8.5.2, 8.5.3.1

Microchip Ex. 1018, p. 631
Microchip v. Aptiv
IPR2024-00494

pins
dual pin-type receptacles, 6.9
inrush current and, 7.2.4.1
single pin-type receptacles, 6.9
pipes
aborting or resetting, 10.5.2.2, 10.5.3.2.1
active, stalled, or idle status, 10.5.2.2
allocating bandwidth for, 4.7.5
characteristics and transfer types, 5.4
client pipes, 10.5.1.2.2
data transfer mechanisms, 10.1.3
Default Control Pipe, 4.4 (*See also* Default Control Pipe)
default pipes, 10.5.1.2.1
defined, 2.0 *glossary,* 4.4
in device characteristics, 4.8.1
identification, 10.3.4
overview, 5.3.2, 10.5.1, 10.5.3
pipe state control, 10.5.2.2
pipe usage, 10.5.1.2
Policies, 10.3.1, 10.3.3, 10.5.3.2.2
queuing IRPs, 10.5.3.2.3
role in data transfers, 4.7
service and polling intervals, 10.3.3
stream and message pipes, 4.4, 5.3.2, 5.3.2.1, 5.3.2.2
supported pipe types, 10.5.3.1 to 10.5.3.1.4
USBD pipe mechanism responsibilities, 10.5.1 to 10.5.3.2.4
USB robustness and, 4.5
pipe state control, 10.5.2.2
pipe status, 10.5.2.2
PK signal/event, 11.5 *Table 11-5*
plating
plug contact materials, 6.5.4.3
plug shell materials, 6.5.4.2
receptacle contact materials, 6.5.3.3
receptacle shell materials, 6.5.3.2
PLL, defined, 2.0 *glossary*
plugs
DC resistance, 6.6.3
interface and mating drawings, 6.5.3
keyed connector protocol, 6.2
materials, 6.5.4.1, 6.5.4.2, 6.5.4.3
orientation, 6.5.1
Series "A" and Series "B" plugs, 6.5.4
standards for, 6.7
termination data, 6.5.2
USB Icon, 6.5, 6.5.1
Policies
defined, 10.3.1
setting, 10.3.3
USBDI mechanisms, 10.5.3.2.2
polling
defined, 2.0 *glossary*
endpoints, 9.6.6
setting intervals for pipes, 10.3.3
polybutylene terephthalate (PBT), 6.5.3.1, 6.5.4.1
polyethylene terephthalate (PET), 6.5.3.1, 6.5.4.1
POR signal/event
Bus_Reset state and, 11.6.3.9
defined, 2.0 *glossary*
in receiver state machine, 11.6.3 *Table 11-8*
Port Change field, 11.4.4, 11.24.2.7.2
PORT_CONNECTION
defined, 11.24.2.7.1.1
hub class feature selectors, 11.24.2
Port Status field, 11.24.2.7.1
PORT_ENABLE
clearing, 11.24.2.2
defined, 11.24.2.7.1.2
hub class feature selectors, 11.24.2
Port Error condition, 11.24.2.7.2.2
PORT_HIGH_SPEED and, 11.24.2.7.1.8
Port Status field, 11.24.2.7.1
Port Error condition, 11.8.1, 11.24.2.7.2.2
Port_Error signal/event, 11.5 *Table 11-5*
Port field, 8.4.2.2
PORT_HIGH_SPEED
Port Status field, 11.24.2.7.1
speed detection and, 11.8.2, 11.24.2.7.1.8
PORT_INDICATOR
clearing port features, 11.24.2.2
getting indicator status, 11.24.2.7.1.10, 11.24.2.13
hub class feature selectors, 11.24.2
indicator selectors, 11.24.2.12
Port Status field, 11.24.2.7.1
port indicators
descriptors, 11.23.2.1
lights on devices, 11.5.3 to 11.5.3.1
port status indicators, 11.24.2.7.1, 11.24.2.7.1.10
selectors, 11.24.2.12
PORT_LOW_SPEED
defined, 11.24.2.7.1.7
hub class feature selectors, 11.24.2
Port Status field, 11.24.2.7.1
speed detection and, 11.8.2, 11.24.2.7.1.8
PORT_OVER_CURRENT
defined, 11.24.2.7.1.4
hub class feature selectors, 11.24.2
over-current conditions and, 11.11.1, 11.12.5
port indicators, 11.24.2.7.1.10
Port Status field, 11.24.2.7.1

Microchip Ex. 1018, p. 632
Microchip v. Aptiv
IPR2024-00494

PORT_POWER
clearing, 11.24.2.2
defined, 11.24.2.7.1.6
hub class feature selectors, 11.24.2
Port Status field, 11.24.2.7.1
SetPortFeature() request, 11.24.2.13
shared power switching and, 11.11.1
PortPwrCtrlMask field
hub descriptors for, 11.23.2.1
multiple gangs and, 11.11.1
power switching settings, 11.11
PORT_RESET
defined, 11.24.2.7.1.5
hub class feature selectors, 11.24.2
Port Status field, 11.24.2.7.1
SetPortFeature() request, 11.24.2.13
ports. *See also* hubs
in bus topology, 5.2.3
clearing features, 11.24.2.2
data source signaling, 7.1.13 to 7.1.13.2.2
defined, 4.8.2.1
disconnect timer, 11.5.2
downstream facing ports, 4.8.2.1, 11.5 to 11.5.1.15
full- *vs.* low-speed port behavior, 11.8.4
in hub architecture, 11.1.1
hub configuration and, 11.13
hub descriptors, 11.23 to 11.23.2.1
hub internal ports, 11.4 to 11.4.4
indicators, 11.5.3 to 11.5.3.1, 11.24.2.7.1, 11.24.2.7.1.10
indicator selectors, 11.24.2.12
input capacitance, 7.1.6.1
over-current reporting and recovery, 11.12.5
port expansion considerations in USB development, 1.1
port selector state machine, 11.7.1.4 to 11.7.1.4.4
power control, 11.11
resetting, 10.2.8.1
root ports, 2.0 *glossary*
setting port features, 11.24.2.13
signal speed support, 7
status
bus state evaluation, 11.8 to 11.8.4.1
detecting status changes, 7.1.7.5, 11.12.2, 11.12.3
hub and port status change bitmap, 11.12.4
port status bits, 11.24.2.7.1 to 11.24.2.7.2.5
testing mode, 11.24.2.7.1.9, 11.24.2.13
upstream facing ports, 4.8.2.1, 11.6 to 11.6.4.6
port selector state machine, 11.7.1.4 to 11.7.1.4.4
port status change bits, 11.24.2.7.1 to 11.24.2.7.2.5
Port Status field, 11.24.2.7.1
PORT_SUSPEND
clearing, 11.24.2.2
defined, 11.24.2.7.1.3
hub class feature selectors, 11.24.2
Port Status field, 11.24.2.7.1
SetPortFeature() request, 11.24.2.13
PORT_TEST
hub class feature selectors, 11.24.2
overview, 11.24.2.7.1.9
Port Status field, 11.24.2.7.1
specific test modes, 11.24.2.13
power budgeting, 7.2.1.4, 9.2.5.1
power conductors in cables, 6.3
power control circuits in bus-powered hubs, 7.2.1.1
power distribution and management. *See also* over-current conditions; power switching
classes of devices, 7.2.1 to 7.2.1.5
bus-powered devices or hubs, 4.3.1, 7.2.1.1
high-power bus-powered functions, 7.2.1.4
low-power bus-powered functions, 7.2.1.3
self-powered devices or hubs, 4.3.1, 7.2.1.2, 7.2.1.5
configuration characteristics
hub descriptors for power-on sequence, 11.23.2.1
information in device characteristics, 4.8.1
power consumption in configuration descriptors, 9.6.3
power source capability in configuration, 9.1.1.2
dynamic attach and detach, 7.2.3, 7.2.4 to 7.2.4.2
Host Controller role in, 4.9
host role in, 10.1.5
hub support for, 11.1
loss of power, 7.2.1.2
over-current conditions, 7.2.1.2.1, 11.12.5 (*See also* over-current conditions)
overview, 4.3.1, 4.3.2, 9.2.5
power budgeting, 7.2.1.4, 9.2.5.1
power status
control during suspend/resume, 7.2.3
device states, 9.1.1.2
port power states, 11.24.2.7.1.6, 11.24.2.13
power switching, 11.11, 11.11.1
remote wakeup, 7.2.3, 9.2.5.2
USB System role, 10.5.4.2
USB System Software role, 4.9
voltage drop budget, 7.2.2
Powered device state, 9.1.1.2, 9.1.1 *Table 9-1*
Powered Off state, 11.5, 11.5.1.2
powered-on ports, 11.24.2.13
Power On Reset. *See* POR signal/event
power-on sequence, 11.23.2.1

Microchip Ex. 1018, p. 633
Microchip v. Aptiv
IPR2024-00494

power pair construction, 6.6.2
power pins, 7.2.4.1
Power_source_off signal/event, 11.5 *Table 11-5*
power switching
  bus-powered hubs, 7.2.1.1
  getting port status, 11.24.2.7.1
  hub descriptors for, 11.23.2.1
  hub port power control, 11.11
  power-on and connection events timing, 7.1.7.3
  power switching gangs, 11.11.1
  staged power switching, 7.2.1.4
preamble packet. *See* PRE PID
preambles, 8.6.5
preboot control, passing to operating systems, 10.5.5
prebuffering data, 5.12.5
prepared termination, 6.4.2, 6.4.3
PRE PID, 8.3.1 *Table 8-1*
  inter-packet delays and, 7.1.18.1
  low-speed port behavior and, 11.8.4
  low-speed transactions, 8.6.5
  overview, 8.3.1 *Table 8-1*
  Transmit state and, 11.5.1.7
priming elasticity buffer, 11.7.1.3
Priming state, 11.7.1.4.2
product descriptions in device descriptors, 9.6.1
Product IDs in device descriptors, 9.6.1
programmable data rate, defined, 2.0 *glossary*
prohibited cable assemblies, 6.4.4
Promoters, USB Implementers Forum, 1.4, 2.0 *glossary*
propagation delay
  cable delay, 7.1.16
  detachable cables, 6.4.1
  end-to-end signal delay, 7.1.19 to 7.1.19.2
  EOF point advancement and, 11.2.3.2
  full- and low-speed signals, 7.1.14.1
  full-speed source electrical characteristics, 7.3.2 *Table 7-9*
  high-/full-speed cables, 6.4.2
  high-speed signaling, 7.1.1.3, 7.1.14.2
  low-speed cables, 6.4.3, 7.1.1.2
  low-speed source electrical characteristics, 7.3.2 *Table 7-10*
  tests, 6.7 *Table 6-7*
propagation delay skew, 6.7 *Table 6-7*
protected fields in packets, 8.3.5
protocol codes
  defined, 9.2.3
  in device descriptors, 9.6.1
  in device qualifer descriptors, 9.6.2
  in interface descriptors, 9.6.5
protocol engine requirements of Host Controller, 10.2.5
protocol errors, detecting, 10.2.6
Protocol field, 9.2.3
Protocol layer, 8
  bit ordering, 8.1
  bus protocol, 4.4
  data toggle synchronization and retry, 8.6 to 8.6.5
  error detection and recovery, 8.7 to 8.7.4
  packet field formats, 8.3 to 8.3.5.2
  packet formats, 8.4 to 8.4.6.4
  SYNC field, 8.2
  transaction formats, 8.5 to 8.5.5
protocols
  defined, 2.0 *glossary*
  protocol codes, 9.2.3, 9.6.1, 9.6.2, 9.6.5
protocol stall, 8.4.5, 8.5.3.4
PS signal/event, 11.5 *Table 11-5*
pull-up and pull-down resistors
  buffer impedance measurement, 7.1.1.1
  high-speed signaling and, 7.1
  power control during suspend/resume, 7.1.7.6, 7.2.3
  signaling speeds and, 7.1 *Table 7-1*
  signal termination, 7.1.5.1

Q

quantization in high-speed microframe timer range, 11.2.1
queuing IRPs, 10.5.3.2.3

R

RA (rate adaptation)
  asynchronous RA, 2.0 *glossary*
  audio connectivity and, 5.12.4.4.1
  defined, 2.0 *glossary*
  in source-to-sink connectivity, 5.12.4.4
  synchronous data connectivity, 5.12.4.4.2
  synchronous RA, 2.0 *glossary*
random vibration standards, 6.7 *Table 6-7*
rate adaptation. *See* RA (rate adaptation)
rate matchers
  asynchronous endpoints, 5.12.4.1.1
  buffering for rate matching, 5.12.8
  client software role, 5.12.4.4
  in non-USB isochronous application, 5.12.1
  synchronous endpoints, 5.12.4.1.2
ratings, full-speed, 6.4.1, 6.4.2
read/write sequences
  in bulk transfers, 8.5.2
  in control transfers, 8.5.3, 8.5.3.1
Ready/ACK status, 11.15
Ready/Data status, 11.15
Ready/lastdata status, 11.15
Ready/moredata status, 11.15
Ready/NAK status, 11.15
Ready/Stall status, 11.15
Ready status, 11.15

Microchip Ex. 1018, p. 634
Microchip v. Aptiv
IPR2024-00494

Ready/trans-err status, 11.15
real-time clock, 5.12.1
real-time data transfers. *See* isochronous transfers
receive clock, 11.7.1.2, 11.7.1.3
receiver eye pattern templates. *See* eye pattern templates
receivers
  differential data receivers, 7.1 *Table 7-1*
  receive phase of signaling, 7.1.1
  receiver characteristics, 7.1.4 to 7.1.4.2
  receiver jitter, 7.1.15 to 7.1.15.2, 7.3.2 *Table 7-9,* 7.3.2 *Table 7-10,* 7.3.3 *Figure 7-51*
  receiver sequence bits, 8.6, 8.6.2
  receiver state machine, 11.6, 11.6.3
  sensitivity requirements, 7.1.2.2 *Figure 7-15,* 7.1.2.2 *Figure 7-16,* 7.1.2.2 *Figure 7-18*
  single- ended receivers, 7.1 *Table 7-1*
  squelch detection, 7.1, 7.1.4.2
ReceivingHJ state, 11.6.3.2
ReceivingHK state, 11.6.3.5
ReceivingIS state, 11.6.3.1
ReceivingJ state, 11.6.3, 11.6.3.3
ReceivingK state, 11.6.3, 11.6.3.6
ReceivingSE0 state, 11.6.3, 11.6.3.8
receptacles
  interface and mating drawings, 6.5.3
  keyed connector protocol, 6.2
  materials, 6.5.3.1, 6.5.3.2, 6.5.3.3
  PCB reference drawings, 6.9
  Series "A" and Series "B" receptacles, 6.5.3
  standards for, 6.7
  termination data, 6.5.2
  USB Icon, 6.5, 6.5.1
Recipient bits, 9.4.5
reclocking, defined, 11.7.1
recovering from errors. *See* error detection and handling
recovery intervals for devices, 9.2.6.2
re-enumerating sub-trees, 10.5.4.5
reflected endpoint status, 10.5.2.2
registers in hub timing, 11.2.3.1
regulators in bus-powered hubs, 7.2.1.1
regulatory compliance, 7.0
regulatory requirements for USB devices, 7.3.1
reliable delivery in isochronous transfers, 5.12
remote wakeup
  in configuration descriptors, 9.6.3
  Host Controller role, 10.2.7
  inrush current and, 7.2.3
  overview, 9.2.5.2
  resume signaling, 7.1.7.7, 9.1.1.6
  timing relationships, 11.9
  USB System role in, 10.5.4.5
Remote Wakeup field, 9.4.5
removable devices, 11.23.2.1
removing devices. *See* dynamic insertion and removal
RepeatingSE0 state, 11.6.4, 11.6.4.3
replacing configuration information, 10.5.4.1.3
reporting rates for feedback, 5.12.4.2
request codes, 9.4 *Table 9-4,* 11.24.2
Request Errors, 9.2.7
requests. *See also* PIDs; *names of specific requests*
  bRequest field, 9.3.2
  class-specific requests, 9.2.6.5, 10.5.2.8, 11.24 to 11.24.2.13
  completion times for hub requests, 11.24.1
  control transfers and, 5.5
  defined, 2.0 *glossary*
  in device class definitions, 9.7.3
  hub standard and class-specific requests, 11.24 to 11.24.2.13
  information requirements for, 10.3.4
  overview, 9.2.6
  port status reporting, 11.12.3
  request processing timing, 9.2.6.1
  reset/resume recovery time, 9.2.6.2
  set address processing, 9.2.6.3
  standard device requests, 9.2.6.4, 9.4 to 9.4.11
  standard feature selectors, 9.4 *Table 9-6*
  standard hub requests, 11.24 to 11.24.2.13
  standard request codes, 9.4 *Table 9-4*
  USBD command mechanisms, 10.5.2 to 10.5.2.12
  USB device requests, 9.3 to 9.3.5
  vendor-specific requests, 10.5.2.9
required data sequences for transfers, 5.4
Reserved PID, 8.3.1 *Table 8-1*
reserved portions of frames, 5.5.4
Reserved test mode, 9.4.9
Reset bus state
  downstream ports, 11.5, 11.5.1.5
  high- and full-speed operations, 5.3.1.1
  high-speed detection and, 7.1.5.2
  high-speed signaling and, 7.1.7.6
  in power-on and connection events, 7.1.7.3
  reset signaling, 7.1.7.5
  signaling levels and, 7.1.7.1
reset condition
  in bus enumeration process, 9.1.2
  C_PORT_RESET bit, 11.24.2.7.2.5
  Default device state and, 9.1.1.3
  device characteristics, 9.2.1
  getting port status, 11.24.2.7.1
  hub reset behavior, 11.10
  PORT_RESET bit, 11.24.2.7.1.5
  port status change bits, 11.24.2.7.2
  remote wakeup and, 10.5.4.5
  reset handshake, C.2.4

Microchip Ex. 1018, p. 635
Microchip v. Aptiv
IPR2024-00494

reset condition *(Continued)*
reset recovery time, 7.1.7.5, 9.2.6.2
reset signaling, 7.1.7.5
resetting pipes, 10.5.2.2
SetPortFeature(PORT_RESET) request, 11.24.2.13
state machine diagams, C.0
USB System and, 10.2.8.1
reset devices, communicating with, 10.5.1.1
reset handshake, C.2.4
Resetting state, 11.5.1.5
ResetTT() request, RESET_TT, 11.24.2, 11.24.2.9
resistance ratings, 6.6.3
resistors
high-speed signaling and, 7.1
pull-up and pull-down resistors, 7.1.1.1, 7.1.5.1, 7.1.7.6, 7.2.3
series damping resistors, 7.1.1.1
speed detection and, 9.1.1.3
resonators, data-rate tolerance and, 7.1.11
resource management, USB System role in, 10.3.2
Restart_E state, 11.5, 11.5.1.13
Restart_S state, 11.5, 11.5.1.12
Resume bus state
downstream ports, 11.5, 11.5.1.10
overview, 7.1.7.7
receivers, 11.6.3, 11.6.3.7
reset signaling and, 7.1.7.5
signaling levels and, 7.1.7.1
Resume_Event signal/event, 11.4
resume intervals for devices, 9.2.6.2
resume signaling
after loss of synchronication, 11.22.2
hub support, 11.1.2.2, 11.9
power control during suspend/resume, 7.2.3
remote wakeup and, 10.5.4.5
resume conditions in Hub Controller, 11.4.4
single-ended transmissions, 11.6.1
retire, defined, 2.0 *glossary*
retiring IRPs. *See* aborting/retiring transfers
RFI, USB grounding and, 6.8
rise and fall times
data source jitter, 7.1.13.1.1
full-speed connections, 7.1.1.1
full-speed source electrical characteristics, 7.3.2 *Table 7-9*
high-speed signaling, 7.1.2.2
high-speed source electrical characteristics, 7.3.2 *Table 7-8*
low-speed source electrical characteristics, 7.3.2 *Table 7-10*
overview, 7.1.2.1 to 7.1.2.2
SE0 from low-speed devices, 7.1.14.1
testing, 7.1.20
robustness of USB, 3.3, 4.5 to 4.5.2
root hub
in bus topology, 5.2.3
defined, 2.0 *glossary*
HCDI presentation of, 10.4
Host Controller and, 10.2.8, 10.2.8.1
state handling, 10.2.1
root port hub, defined, 7.2.1
root ports, 2.0 *glossary,* 11.9
round trip times, 7.1.6.2
Rptr_Enter_WFEOPFU signal/event, 11.5 *Table 11-5*
Rptr_Exit_WFEOPFU signal/event, 11.5 *Table 11-5*
Rptr_WFEOP signal/event, 11.6.4 *Table 11-9*
Run timer status, C.0
Rx_Bus_Reset signal/event, 11.6.4 *Table 11-9,* 11.7.1.4 *Table 11-10,* 11.7.2.3 *Table 11-11*
Rx_Resume signal/event, 11.5 *Table 11-5,* 11.7.2.3 *Table 11-11*
Rx_Suspend signal/event, 11.4, 11.5 *Table 11-5,* 11.6.4 *Table 11-9,* 11.7.2.3 *Table 11-11*

S

S field (Start), 8.4.2.2
sample clock
buffering for rate matching, 5.12.8
defined, 5.12.2
synchronous endpoints, 5.12.4.1.2
sampled analog devices, 5.12.4
sample declarations in state machines, B.1, B.2, B.3
Sample Rate Conversion. *See* SRC
samples
defined, 2.0 *glossary*
sample size in buffering calculations, 5.12.8
samples per (micro)frame in isochronous transfers, 5.12.4.2
SC field (Start/Complete field), 8.4.2.2, 8.4.2.3
scheduling
access to USB interconnect, 4.1
host split transaction scheduling, 11.18.4
microframe pipeline scheduling, 11.18.2
periodic split transaction scheduling, 11.18
transaction schedule in bus protocol overview, 4.4
Transaction Translator transaction scheduling, 11.14.2 to 11.14.2.3
SE0sent signal/event, 11.6.4 *Table 11-9*

Microchip Ex. 1018, p. 636
Microchip v. Aptiv
IPR2024-00494

SE0 signal/event
in data signaling, 7.1.7.4.1
downstream port state machine, 11.5 *Table 11-5*
Not Configured state, 11.5.1.1
propagation delays, 7.1.14.1
pull-down resistors and, 7.1.7.3
receiver state machine, 11.6.3 *Table 11-8*
reset signaling, 7.1.7.5
SE0 interval of EOP, 7.3.2 *Table 7-9,* 7.3.2 *Table 7-10*
signaling levels and, 7.1.7.1
single-ended transmissions, 11.6.1
test mode, 7.1.20
SE0 width, 7.1.13.2.1, 7.1.14.1, 7.3.2 *Table 7-9,* 7.3.2 *Table 7-10*
SE1 signal/event, 7.1.1, 11.6.1
selective resume signaling, 11.9
selective suspend signaling
defined, 9.1.1.6
hub support, 11.9
overview, 7.1.7.6.2
self-powered devices and functions
configuration descriptors, 9.6.3
defined, 4.3.1, 7.2.1
device states, 9.1.1.2
overview, 7.2.1.5
Self Powered field, 9.4.5
self-powered hubs
configuration, 11.13
defined, 7.2.1
device states, 9.1.1.2
over-current protection, 7.2.1.2.1
overview, 7.2.1.2
power switching, 11.11
self-recovery, USB robustness and, 4.5
SendEOR state, 11.5, 11.5.1.11
SendJ state, 11.6.4, 11.6.4.4
Send Resume state (Sresume), 11.6.4, 11.6.4.6
sequence of transactions in frames, 5.11.2
Serial Interface Engine (SIE), 10.1.1, 10.2.2
serializer/deserializer, 10.2.2
serial numbers in device descriptors, 9.6.1
Series "A" and "B" connectors
detachable cables and, 6.4.1
keyed connector protocol, 6.2
plugs
injection molded thermoplastic insulator material, 6.5.4.1
interface drawings, 6.5.4
plug (male) contact materials, 6.5.4.3
plug shell materials, 6.5.4.2
Series "A" and "B" connectors *(Continued)*
receptacles
injection molded thermoplastic insulator material, 6.5.3.1
interface and mating drawings, 6.5.3
PCB reference drawings, 6.9
receptacle contact materials, 6.5.3.3
receptacle shell materials, 6.5.3.2
standards for, 6.7
USB Icon, 6.5, 6.5.1
series damping resistors, 7.1.1.1
service, defined, 2.0 *glossary*
service clock, 5.12.2, 5.12.8
service intervals, 2.0 *glossary,* 10.3.3
service jitter, defined, 2.0 *glossary*
service periods of data, 5.12.1
service rates, defined, 2.0 *glossary*
SetAddress() request, SET_ADDRESS
hub requests, 11.24.1
overview, 9.4.6
reset recovery time and, 7.1.7.5
standard device request codes, 9.4
time limits for completing processing, 9.2.6.3
SetConfiguration() request, SET_CONFIGURATION
hub requests, 11.24.1
overview, 9.4.7
Powered-off state and, 11.5.1.2
setting configuration in descriptors, 9.1.1.5, 9.6.3
standard device request codes, 9.4
SetDescriptor() request, SET_DESCRIPTOR
getting endpoint descriptors, 9.6.6
hub class requests, 11.24.2
hub requests, 11.24.1
interface descriptors and, 9.6.5
overview, 9.4.8
SetHubDescriptor() request, 11.24.2.10
standard device request codes, 9.4
SetDeviceFeature(DEVICE_REMOTE_WAKEUP) request, 10.5.4.5
SetFeature() request, SET_FEATURE, 9.4.5, 9.4.9
hub class requests, 11.24.2
hub requests, 11.24.1
overview, 9.4.9
SetHubFeature() request, 11.24.2.12
SetPortFeature() request, 11.24.2.13
standard device request codes, 9.4
TEST_MODE, 7.1.20, 9.4.9
TEST_SELECTOR, 9.4.9

Microchip Ex. 1018, p. 637
Microchip v. Aptiv
IPR2024-00494

SetHubDescriptor() request, 11.24.2, 11.24.2.10
SetHubFeature() request, 11.24.2, 11.24.2.6, 11.24.2.12
SetInterface() request, SET_INTERFACE, 9.2.3, 9.4, 9.4.10, 9.6.5, 11.24.1
SetPortFeature() request
  hub class requests, 11.24.2, 11.24.2.13
  PORT_CONNECTION, 11.24.2.7.1.1
  PORT_ENABLE, 11.24.2.7.1.2
  PORT_HIGH_SPEED, 11.24.2.7.1.8
  PORT_INDICATOR, 11.24.2.7.1.10, 11.24.2.12
  PORT_LOW_SPEED, 11.24.2.7.1.7
  PORT_OVER_CURRENT, 11.24.2.7.1.4
  PORT_POWER
    Disconnected state and, 11.5.1.3
    port power settings, 11.11
    port power states, 11.24.2.7.1.6, 11.24.2.13
    requirements, 11.24.2.13
  PORT_RESET
    completion, 9.2.6
    C_PORT_ENABLE bit, 11.24.2.7.2.2
    evaluating device speed during, 11.8.2
    initiating port reset, 11.24.2.7.1.5, 11.24.2.13
    in port enabling, 11.24.2.7.1.2
    requirements, 11.24.2.13
    Resetting state and, 11.5.1.5
  PORT_SUSPEND, 10.5.4.5
    selective suspend, 7.1.7.6.2
    suspending ports, 11.5.1.9, 11.24.2.7.1.3
  PORT_TEST, 11.24.2.7.1.9, 11.24.2.13
  power-off conditions and, 11.13
  TEST_MODE, 7.1.20
SetTest signal/event, 11.5 *Table 11-5*
SETUP PID, 8.3.1 *Table 8-1*
  ACK handshake and, 8.4.5
  ADDR field, 8.3.2.1
  in control transfers, 8.5.3
  in data toggle, 8.6.1
  ENDP field, 8.3.2.2
  function response to, 8.4.6.4
  overview, 8.3.1 *Table 8-1*
  split transaction examples, A.1
  token CRCs, 8.3.5.1
  in token packets, 8.4.1
Setup stage
  in control transfer data sequences, 5.5.5
  in control transfers, 5.5, 8.5.3
  data format for USB device requests, 9.3
SETUP transactions. *See* SETUP PID
shell
  conductors, 6.5.2
  plug shell materials, 6.5.4.2
  receptacle shell materials, 6.5.3.2
shielding
  grounding, 6.8
  low-speed and high-/full-speed cables, 6.6
  outer and inner cable shielding, 6.6.1
  shielded cables illustrated, 6.4.1
  standardized contact terminating assignments, 6.5.2
short circuits, USB withstanding capabilities, 7.1.1
short packets
  defined, 9.4.3
  detecting, 10.2.6
  multiple data payloads and, 5.3.2
SIE (Serial Interface Engine), 10.1.1, 10.2.2
signal conductors in cables, 6.3
signal edges. *See* edges of signals
signaling
  bit stuffing, 7.1.9 to 7.1.9.2
  cable attenuation, 7.1.17
  connect and disconnect signaling, 7.1.7.3
  data encoding/decoding, 7.1.8
  data rate, 7.1.11
  data signaling, 7.1.7.4 to 7.1.7.4.2
  delay
    bus turn-around time and inter-packet delay, 7.1.18 to 7.1.18.2
    cable delay, 7.1.16
    cable skew delay, 7.1.3
    maximum end-to-end signal delay, 7.1.19 to 7.1.19.2
  high-speed driver characteristics, 7.1.1.3
  hub signaling timings, 7.1.14 to 7.1.14.2
  in-band and out-of-band, 10.1.2
  input characteristics, 7.1.6.1
  jitter, 7.1.13.1 to 7.1.13.1.2, 7.1.15 to 7.1.15.2 (*See also* jitter)
  low-speed (1.5Mb/S) driver characteristics, 7.1.1.2
  (micro)frame intervals and repeatability, 7.1.12
  overview, 7.1
  receiver characteristics, 7.1.4 to 7.1.4.2, 7.1.15.1
  reset signaling, 7.1.7.5
  resume signaling, 7.1.7.7
  rise and fall time, 7.1.2.1 to 7.1.2.2
  signal attenuation, 7.1.17
  signal edges (*See* edges of signals)
  signaling levels, 7.1.7 to 7.1.7.5
  signal integrity, 4.5
  signal termination, 7.1.5.1
  source signaling, 7.1.13 to 7.1.13.2.2
  suspend signaling, 7.1.7.6
  sync pattern, 7.1.10
  USB driver characteristics, 7.1.1

Microchip Ex. 1018, p. 638
Microchip v. Aptiv
IPR2024-00494

signal matching, 7.1.2.1
signal pair attenuation, 6.4.1, 6.7 *Table 6-7*
signal pair construction, 6.6.2
signal pins, 7.2.4.1
signal swing, 7.1.2.1
signal termination, 7.1, 7.1.5.1
simple states, notation for, 11.15
Single-ended 0 bus state (SEO)
  in data signaling, 7.1.7.4.1
  pull-down resistors and, 7.1.7.3
  reset signaling, 7.1.7.5
  signaling levels and, 7.1.7.1
  test mode, 7.1.20
single-ended capacitance, 7.1.1.2
single-ended components in upstream ports, 11.6.1
single-ended receivers, 7.1.4.1, 7.1.6.1, 7.1 *Table 7-1*
"single packets" and split transactions, 5.12.3
single pin-type receptacles, 6.9
sink endpoints
  adaptive sink endpoints, 5.12.4.1.3
  audio connectivity, 5.12.4.4.1
  connectivity overview, 5.12.4.4
  feedback for isochronous transfers, 5.12.4.2
  synchronization types, 5.12.4.1
  synchronous data connectivity, 5.12.4.4.2
skew
  cable skew delay, 6.7 *Table 6-7,* 7.1.3, 7.3.2 *Table 7-12*
  differential-to-EOP transition skew, 7.3.3 *Figure 7-50*
  hub EOP delay and EOP skew, 7.3.3 *Figure 7-53*
  hub/repeater electrical characteristics, 7.3.2 *Table 7-11*
  hub switching skew, 7.1.9.1
  Idle-to-K state transition, 7.1.14.1
  minimizing signal skew, 7.1.1
  timing skew accumulation, 11.2.5.1 to 11.2.5.2
slips in synchronous data, 5.12.4.4.2
small capacitors, 7.1.6.1
SOF PID, 8.3.1 *Table 8-1*
  frame number field, 8.3.3
  frames and microframes, 8.4.3.1
  start-of-frame packets, 8.4.3
SOFs
  after loss of synchronication, 11.22.2
  bus clock and, 5.12.2
  defined, 2.0 *glossary*
  in downstream port state machine, 11.5
  error recovery in isochronous transfers, 5.12.7
  frame and microframe intervals, 7.1.12
SOFs *(Continued)*
  frame and microframe timer synchronization, 11.2, 11.2.3 to 11.2.3.3
  frame clock tracking and microframe SOFs, 5.12.4.1.2
  high-speed SOF in connect detection, 7.1.7.3
  Host Controller frame and microframe generation, 10.2.3
  loss of consecutive SOFs, 11.2.5
  loss of TT synchronization, 11.22.1
  microframe synchronization and, 11.2.4
  overview, 8.4.3
  tracking, 5.12.6, 5.12.7
  using as clocks, 5.12.5
soft-start circuits, 7.2.4.1
software interfaces. *See* client software; HCDI; host; USBDI; USB System software
SOHP, 11.7.2.1
solderability standards, 6.7 *Table 6-7*
solder tails, 6.5.3.3, 6.5.4.3
SOP bus state, 7.1.7.1, 7.1.7.2, 7.1.7.4.1, 7.1.7.4.2
SOP_FD signal/event
  generating, 11.4.4
  in Hub Repeater state machine, 11.7.2.3 *Table 11-11*
SOP_FU signal/event, 11.7.2.3 *Table 11-11*
SOPs, 8.3
  error detection through bus turn-around timing, 8.7.2
  frame and microframe timer synchronization, 11.2.3 to 11.2.3.3
  Idle-to-K state transition, 7.1.14.1
  SOP distortion, 7.3.3 *Figure 7-52*
  timeout periods and, 7.1.19.1
SORP signal/event, 11.7.1.4 *Table 11-10*
source endpoints
  adaptive source endpoints, 5.12.4.1.3
  audio connectivity, 5.12.4.4.1
  connectivity overview, 5.12.4.4
  feedback for isochronous transfers, 5.12.4.2
  synchronization types, 5.12.4.1
  synchronous data connectivity, 5.12.4.4.2
source jitter, 7.3.2 *Table 7-8,* 7.3.2 *Table 7-9,* 7.3.2 *Table 7-10*
source/sink connectivity, 5.12.4.4
special PIDs
  ERR PID, 8.3.1 *Table 8-1,* 8.4.5
  PING PID, 8.3.1 *Table 8-1,* 8.3.2.2, 8.3.5.1, 8.4.1, 8.4.5, 8.5.2, 8.5.3.1

Microchip Ex. 1018, p. 639
Microchip v. Aptiv
IPR2024-00494

special PIDs *(Continued)*
PRE PID
defined, 8.3.1 *Table 8-1*
inter-packet delays and, 7.1.18.1
low-speed port behavior and, 11.8.4
low-speed transactions, 8.6.5
Transmit state and, 11.5.1.7
Reserved PID, 8.3.1 *Table 8-1*
SPLIT PID, 8.3.1 *Table 8-1,* 8.3.2.1, 8.3.5.1, 8.4.2 to 8.4.2.3
specific-sized data payloads, 5.3.2
speed
downstream facing ports and hubs, 7.1
high-speed devices operating at full-speed, 5.3.1.1
hubs and signaling speeds, 11.1.1
measurement planes in speed signaling eye patterns, 7.1.2.2
pull-up and pull-down resistors and, 7.1 *Table 7-1*
speed dependent descriptors, 9.2.6.6
upstream facing ports and hubs, 7.1
speed detection
attached devices, 11.8.2
detecting low-speed functions and hubs, 11.24.2.7.1.7
detecting speed of devices, 7.1.7.3
other_speed_configuration descriptor, 9.6.4
PORT_HIGH_SPEED, 11.24.2.7.1.8
reset condition and Default device state, 9.1.1.3
speed indication bits, 7.1.5.2
termination and, 7.1.5.1
SPI, defined, 2.0 *glossary*
SPLIT PID, 8.3.1 *Table 8-1,* 8.3.2.1, 8.3.5.1, 8.4.2 to 8.4.2.3
splitting sample across packets, 5.12.8
split transactions. *See also* complete-split transactions; start-split transactions
best case full-speed budgets, 11.18.1, 11.18.4
bulk/control transactions
control transfers, 5.5.4
IN examples, A.2
OUT and SETUP examples, A.1
sequencing, 11.17.3
state machines, 11.17.2
data handling, 11.14.1.1
data packet types, 8.4.4
defined, 2.0 *glossary,* 5.10
failures, 11.17.5
host controller and, 11.14.1.2
host scheduling, 11.18.4
split transactions *(Continued)*
IN transactions
bulk/control examples, A.2
interrupt examples, A.4
interrupt transaction sequencing, 11.20.4
isochronous examples, A.6
isochronous transaction sequencing, 11.21.4
interrupt transactions
IN examples, A.4
flow sequences and state machines, 11.20 to 11.20.4
OUT examples, A.3
isochronous transactions
IN examples, A.6
OUT examples, A.5
overview, 11.21 to 11.21.4
microframe pipeline, 11.18.2
non-periodic transactions
IN examples, A.2
OUT and SETUP examples, A.1
overview, 11.17 to 11.17.5
sequencing, 11.17.3
state machines, 11.17.2
notation for, 11.15
OUT transactions
bulk/control examples, A.1
interrupt examples, A.3
interrupt sequencing, 11.20.3
isochronous examples, A.5
isochronous sequencing, 11.21.3
periodic transactions
interrupt IN examples, A.4
interrupt OUT examples, A.3
interrupt transaction state machines, 11.20 to 11.20.4
isochronous IN examples, A.6
isochronous OUT examples, A.5
isochronous transaction state machines, 11.21 to 11.21.4
overview, 11.18 to 11.18.8
SETUP transactions, A.1
"single packets" and, 5.12.3
state machines
bulk/control state machines, 11.17.2
interrupt state machines, 11.20 to 11.20.4
isochronous transaction state machines, 11.21 to 11.21.4
overview, 11.16
split transaction state machines, 8.5
token packets, 8.4.2 to 8.4.2.3
Transaction Translator, 11.1.1, 11.14.1

Microchip Ex. 1018, p. 640
Microchip v. Aptiv
IPR2024-00494

squelch circuit, 7.1.20, 11.7.1.1
squelch detection
  error detection and, 8.7.3, 8.7.4
  turn-around timing and, 8.7.2
Squelch signal/event, 11.7.1.4 *Table 11-10*
Squelch state, 7.1, 7.1.4.2, 7.1.7.2
SRC
  asynchronous SRC, 2.0 *glossary*
  audio connectivity and, 5.12.4.4.1
  defined, 2.0 *glossary*
  synchronous SRC, 2.0 *glossary*
Sresume state, 11.6.4, 11.6.4.6
SSPLIT. *See* start-split transactions (SSPLIT)
staged power switching, 7.2.1.4
stages in control transfers, defined, 2.0 *glossary, 5.5. See also* Data stage; Setup stage; Status stage
STALLs, 8.3.1 *Table 8-1*
  in bulk transfers, 5.8.5, 8.5.2, 11.17.1
  in control transfers, 8.5.3.1, 11.17.1
  data corrupted or not accepted, 8.6.3
  functional and commanded stalls, 8.4.5
  function response to IN transactions, 8.4.6.1
  function response to OUT transactions, 8.4.6.3
  in interrupt transfers, 5.7.5, 8.5.4, 11.20.4
  overview, 8.4.5
  protocol stalls, 8.4.5
  Ready/Stall status, 11.15
  Request Error responses, 9.2.7
  responses to standard device requests, 9.4
  returned by control pipes, 8.5.3.4
standard device information, 4.8.1
standard device requests, 9.2.6.4, 9.4 to 9.4.11, 11.24.1
standards (applicable documents), 6.7.1
standard USB descriptor definitions, 9.6.1 to 9.6.5
Start/Complete field (SC), 8.4.2.2
Started timer status, C.0
Start field (S), 8.4.2.2
Start-of-Frame. *See* SOFs
Start of High-speed Packet (HSSOP), 7.1.7.2, 7.1.7.4.2
Start-of-Packet. *See* SOPs
Start-of-Packet bus state (SOP), 7.1.7.1, 7.1.7.2, 7.1.7.4.1, 7.1.7.4.2
start-of-packet delimiter. *See* SOPs
star topology, 5.2.3
start-split transactions (SSPLIT)
  after loss of synchronication, 11.22.2
  buffering, 11.14.2.1, 11.14.2.2, 11.14.2.3, 11.17
  bulk/control split transactions, 11.17, 11.17.1
  defined, 8.4.2, 11.14.1.2
  freeing pending transactions, 11.18.6.2
start-split transactions *(Continued)*
  isochronous transactions, 11.21
  notation for, 11.15
  overview, 11.14.1
  scheduling, 11.14.2.1, 11.14.2.2, 11.14.2.3, 11.18.4
  split transaction overview, 8.4.2.1
  SSPLIT token, 8.4.2.2
  tracking, 11.18.7
state handling. *See* bus states; status
state machines. *See also names of specific state machines under Dev_, HC_, and TT_*
  actions in, 8.5, 11.15
  bulk/control transaction state machines, 8.5.2, 11.17.2
  conditions in, 8.5
  device state machines, 8.5, 8.5.2, 8.5.5
  diamond symbols in, 8.5, 11.15
  downstream facing port state machines, 11.5
  endpoint state machines, 8.5
  example declarations, B.1, B.2, B.3
  Host Controller state machines, 8.5, 11.16 to 11.16.1.1.2, 11.17.2, 11.20.2, 11.21.2
  host state machines, 8.5.1, 8.5.2, 8.5.5
  Hub Repeater state machine, 11.2.3.3, 11.7.2, 11.7.2.3 *Table 11-11*
  Hub state machine, 11.1.1
  initial states in, 8.5
  input transitions in, 8.5
  internal port state machine, 11.4
  interrupt transaction state machines, 8.5.2, 8.5.2 *Figure 8-33,* 8.5.2 *Figure 8-34,* 11.20 to 11.20.4
  isochronous transaction state machines, 8.5.5 *Figure 8-42,* 8.5.5 *Figure 8-43,* 11.21.2
  notation in, 8.5, 11.15
  output transitions in, 8.5
  over-sampling state machine DPLLs, 7.1.15.1
  overview, 8.5
  port indicator colors, 11.5.3
  port selector state machine, 11.7.1.4 to 11.7.1.4.4
  receiver state machine, 11.6, 11.6.3 *Table 11-8*
  reset protocol diagrams, C.0
  resetting TT state machines, 11.24.2.9
  split transaction state machine overview, 11.16
  state hierarchy, 8.5
  states defined, 8.5
  Transaction Translator state machines, 11.16, 11.16.2 to 11.16.2.1.7, 11.24.2.9
  transitions in, 8.5
  transmitter state machine, 11.6, 11.6.4, 11.6.4.2, 11.6.4 *Table 11-9*
static output swing of USBD, 7.1.1

Microchip Ex. 1018, p. 641
Microchip v. Aptiv
IPR2024-00494

status. *See also* status change bits
device states, 10.5.2.7, 11.12.2
Host Controller role in, 4.9
host's role in monitoring status and activity, 10.1.4
hub and port status change bitmap, 11.12.4
hub and port status changes, 7.1.7.5, 11.12.6
hub status, 11.24.2.6
notification of completion status, 10.3.4
port change information processing, 11.12.3
port indicators, 11.5.3 to 11.5.3.1
port status change bits, 11.24.2.7.2 to 11.24.2.7.2.5
USBD event notifications, 10.5.4.3
USBD status reporting and error recovery, 10.5.4.4
status change bits. *See also* Status Change endpoint
detecting changes, 11.12.2
device states, 11.12.2
hub and port status change bitmap, 11.12.4
hub status, 11.24.2.6
over-current status change bits, 11.12.5
port status change bits, 11.24.2.7.2 to 11.24.2.7.2.5
Status Change endpoint
defined, 11.12.1
device states and, 11.12.2
hub and port status change bitmap, 11.12.4
hub configuration and, 11.13
hub descriptors, 11.23.1
Status stage
in control transfers, 5.5, 5.5.5, 8.5.3
reporting status results, 8.5.3.1
StopTT() request, STOP_TT
hub class requests, 11.24.2
overview, 11.24.2.11
storage temperatures for cables, 6.6.4
stranded tinned conductors, 6.6.2
streaming real time transfers. *See* isochronous transfers
stream pipes
bulk transfers and, 5.8.2
in bus protocol overview, 4.4
defined, 2.0 *glossary,* 5.3.2
interrupt transfers and, 5.7.2
isochronous transfers and, 5.6.2
overview, 5.3.2.1
STRING descriptor, 9.4 *Table 9-5*
string descriptors
GetDescriptor() request, 9.4.3
as optional, 9.5
overview, 9.6.7
stuffed bits. *See* bit stuffing
subclasses
device_qualifer descriptor codes, 9.6.2
device subclass codes, 9.2.3, 9.6.1
interface subclass codes, 9.2.3, 9.6.5
SubClass field, 9.2.3
substrate materials
plug contact materials, 6.5.4.3
plug shell materials, 6.5.4.2
receptacle contact materials, 6.5.3.3
receptacle shell materials, 6.5.3.2
subtree devices after wakeup, 10.5.4.5
successful transfers, 8.6.2, 10.3.4
supply current, 7.3.2 *Table 7-7*
supply voltage
DC electrical characteristics, 7.3.2 *Table 7-7*
oscillators, 7.1.11
surge limiting, 7.2.4.1
Suspend bus state
global suspend, 7.1.7.6.1
overview, 7.1.7.6
power control during suspend/resume, 7.2.3
reset signaling, 7.1.7.5
resume signaling, 7.1.7.7
selective suspend, 7.1.7.6.2
Suspend Delay state, 11.4, 11.4.2
suspended devices
global suspend, 7.1.7.6.1
hub support for suspend signaling, 11.9
power control during suspend/resume, 7.2.3
power-on and connection events, 7.1.7.3
remote wakeup, 9.2.5.2, 10.2.7, 10.5.4.5
reset state machines, C.2.1
resume signaling, 7.1.7.7
selective suspend, 7.1.7.6.2
single-ended transmissions, 11.6.1
Suspend bus state, 7.1.7.6
Suspended device state, 9.1.1.6
suspended hubs
hub reset behavior, 11.10
resume signaling and, 11.1.2.2
suspended ports
C_PORT_SUSPEND, 11.24.2.7.2.3
getting port status, 11.24.2.7.1
port status change bits, 11.24.2.7.2
PORT_SUSPEND, 11.24.2.7.1.3, 11.24.2.13
Suspended state, 9.1.1.6, 9.1.1 *Table 9-1,* 11.5, 11.5.1.9. *See also* Suspend bus state; Suspend state
suspend sequencing, 11.22.2
Suspend state, 11.6.3, 11.6.3.4
switching thresholds for single-ended receivers, 7.1.4.1

Microchip Ex. 1018, p. 642
Microchip v. Aptiv
IPR2024-00494

SYNC field
in data signaling, 7.1.7.4.1
in electrical specifications overview, 4.2.1
high-speed signaling and, 7.1
overview, 8.2
squelch detection and, 11.7.1.1
SynchFrame() request, SYNCH_FRAME, 9.4, 9.4.11, 11.24.1
synchronization. *See also* synchronization types
clock synchronization, 5.12.3
data-per-time synchronization, 5.12.7
data toggle synchronization, 8.4.4, 8.6 to 8.6.5
endpoint synchronization frame, 9.4.11
frame and microframe timer synchronization, 11.2, 11.2.3 to 11.2.3.3
jitter, 2.0 *glossary* (*See also* jitter)
physical and virtual devices, 5.12.4.4
SYNC field, 8.2
sync pattern, 7.1.10
Transaction Translator loss of synchronization, 11.18.6, 11.22.1
transmitter and receiver synchronization in isochronous transfers, 5.12
synchronization types
adaptive, 5.12.4.1.3
asynchronous, 5.12.4.1.1
defined, 2.0 *glossary,* 5.12.4
endpoints and, 9.6.6
overview, 5.12.4.1
synchronous, 5.12.4.1.2
synchronous data connectivity, 5.12.4.4.2
synchronous data devices, 5.12.4
synchronous endpoints, 5.12.4.1.2, 5.12.4.4
synchronous RA, 2.0 *glossary,* 5.12.4.4
synchronous SRC, 2.0 *glossary*
sync pattern, 7.1.7.4.2, 7.1.9, 7.1.10
system configuration. *See* configuration
System Programming Interface, defined, 2.0 *glossary*
system software. *See* USB System Software

T

TDM, defined, 2.0 *glossary*
TDR loading specification, 2.0 *glossary,* 7.1.6.2
telephone interconnects, 1.1
temperature
data-rate inaccuracies and, 7.1.11
ranges for cables, 6.6.4
templates. *See* receiver eye pattern templates
termination
blunt cut and prepared termination, 6.4.2, 6.4.3
DC electrical characteristics, 7.3.2 *Table 7-7*
defined, 2.0 *glossary*
detachable cable assemblies, 6.4.1
electrical specifications overview, 4.2.1
termination *(Continued)*
high-/full-speed captive cable assemblies, 6.4.2
low-speed captive cable assemblies, 6.4.3
signal termination, 7.1, 7.1.5.1
USB topology rules, 6.4.4
termination data, 6.5.2
Termination Impedance, 7.1.6.2
test criteria for electrical, mechanical and environmental compliance, 6.7
Test for Flammability of Plastic Materials for Parts in Devices and Appliances, 6.7.1
Testing state, 11.5.1.14
Test_J test mode, 7.1.20, 9.4.9, 11.24.2.13
Test_K test mode, 7.1.20, 9.4.9, 11.24.2.13
test mode, 7.1.20, 9.4.9, 11.24.2.7.1.9, 11.24.2.13
TEST_MODE, 7.1.20, 9.4.9, 9.4 *Table 9-6*
Test_Packet test mode, 7.1.20, 9.4.9, 11.24.2.13
test planes in high-speed signaling, 7.1.2.2
Test_SE0_NAK test mode, 7.1.20, 9.4.9, 11.24.2.13
TEST_SELECTOR, 9.4.9
thermal shock standards, 6.7 *Table 6-7*
Thevenin resistance, 7.1.5.1
"think time," 11.18.2, 11.23.2.1
"three strikes and you're out" mechanism, 11.17.1
Through Impedance, 7.1.6.2
tiered topology
EOF point advancement and, 11.2.3.2
tiered star topology, 5.2.3
tiers in bus typology, 4.1.1
Time Division Multiplexing (TDM), 2.0 *glossary*
Time Domain Reflectometer loading specification, 2.0 *glossary,* 7.1.6.2
timed states
Disconnected state, 11.5.1.3
Resuming state, 11.5.1.10
timeout
bus transaction timeout, 5.12.7
defined, 2.0 *glossary*
detecting timeout conditions, 10.2.6
high bandwidth transactions and, 5.9.1
split transaction flow sequences, 11.18.8
timeout intervals in error detection, 8.7.2, 8.7.3
timeouts, 11.15, 11.17.1
timing. *See also* cable delay; propagation delay; skew; synchronization; timing waveforms
bus timing/electrical characteristics, 7.3.2
bus transaction time calculations, 5.11.3
bus turn-around timing, 8.7.2
clock model, 5.12.2
clock synchronization, 5.12.3
completion times for hub requests, 11.24.1

Microchip Ex. 1018, p. 643
Microchip v. Aptiv
IPR2024-00494

timing *(Continued)*
current frame timer, 11.2.3.1
data source signaling, 7.1.13 to 7.1.13.2.2
device event timings, 7.3.2 *Table 7-14*
frame and microframe intervals, 7.1.12
frame and microframe timers, 11.2.3 to 11.2.3.3
hub event timings, 7.3.2 *Table 7-13*
hub frame timer, 11.2 to 11.2.5.2
hub signaling timings, 7.1.14 to 7.1.14.2
isochronous transfer feedback, 5.12.4.2
isochronous transfer importance, 5.12
low, full, and high-speed turn-around timing, 8.7.2
next frame timer, 11.2.3.1
in non-USB isochronous application, 5.12.1
port disconnect timer, 11.5.2
power-on and connection events timing, 7.1.7.3
remote wakeup timing relationships, 11.9
request processing timing, 9.2.6.1
Resetting state and Resuming state intervals, 11.5.1.10
Run, Clear, and Started timer status, C.0
SE0 for EOP width timing, 7.1.13.2.1
skew accumulating between host and hub, 11.2.5.1 to 11.2.5.2
SOF PID timing information, 8.4.3
SOF tokens as clocks, 5.12.5
synchronization types, 5.12.4.1
timing waveforms, 7.3.3
differential data jitter, 7.3.3 *Figure 7-49*
differential-to-EOP transition skew and EOP width, 7.3.3 *Figure 7-50*
hub differential delay, differential jitter, and SOP distortion, 7.3.3 *Figure 7-52*
hub EOP delay and EOP skew, 7.3.3 *Figure 7-53*
receiver jitter tolerance, 7.3.3 *Figure 7-51*
toggle mode. *See* data toggle
toggle sequencing, 8.5.5
token packets
in bulk transfers, 8.5.2
bus protocol overview, 4.4
CRCs, 8.3.5.1
defined, 2.0 *glossary*
in isochronous transfers, 8.5.5
overview, 8.4.1
packet field formats, 8.3 to 8.3.5.2
split transaction token packets, 8.4.2 to 8.4.2.3
token phases, notation for, 11.15
token PIDs, 8.3.1 *Table 8-1*. *See also* IN PID; OUT PID; SETUP PID; SOF PID
topology
bus topology, 4.1, 4.1.1, 5.2 to 5.2.5
EOF point advancement and, 11.2.3.2
hub tiers defined, 2.0 *glossary*
trace delays, 7.1.14.2
tracking transactions, 11.18.7
transaction completion prediction, 11.3.3
transaction list
defined, 5.11.1.4
HCD role in, 5.11.1.3
Host Controller and, 5.11.1.5
transactions. *See also specific types of transactions*
aborting, 11.18.6, 11.18.6.1
allocating bandwidth for, 5.11.1 to 5.11.1.5, 10.3.2
buffer size calculations, 5.11.4
bus protocol overview, 4.4
defined, 2.0 *glossary*
error detection and recovery, 8.7 to 8.7.4
maximum allowable transactions per microframe, 5.4.1, 11.18.6.3
multiple transactions in microframes, 5.9, 5.9.2
organization within IRPs, 5.11.2
packet sequences, 8.5
pending, 11.18.6
PING flow control protocol and, 5.5.4
scheduling, 4.4, 11.14.2 to 11.14.2.3
split transactions, 5.5.4, 8.4.2 to 8.4.2.3, A.1, A.2, A.3, A.4
state machine overview, 8.5
timeout, 5.12.7
tracking transactions, 5.11.2
transaction completion prediction, 11.3.3
transaction formats
bulk transfers, 5.8.4, 8.5.2, A.1, A.2
control transfers, 5.5.4, 8.5.3 to 8.5.3.4, A.1, A.2
IN transactions, A.2, A.4, A.6
interrupt transfers, 8.5.4, A.3, A.4
isochronous transfers, 5.6.3, 5.12.6, 5.12.7, 8.5.5, A.5, A.6
non-periodic transactions, 11.17 to 11.17.5
OUT transactions, A.1, A.3, A.5
overview, 8.5
periodic and non-periodic transactions, 11.14.1, 11.18 to 11.18.8, 11.22.1
SETUP transactions, A.1
transaction list, 5.11.1.3, 5.11.1.4, 5.11.1.5
transaction time calculations, 5.11.3
Transaction Translator, 4.8.2.1, 11.18.7

Microchip Ex. 1018, p. 644
Microchip v. Aptiv
IPR2024-00494

Transaction Translator
aborting transactions, 11.18.6.1
buffers
buffer space required, 11.17.4, 11.19
clearing buffers, 11.17.5, 11.24.2.3
periodic and non-periodic buffer sections, 11.14.1
complete-split state searching, 11.18.8
data handling, 11.14.1.1
defined, 2.0 *glossary,* 4.8.2.1, 11.1
delay in bus times, 5.11.3
error handling, 11.22
frame and microframe jitter, 11.2.4
freeing pending start-splits, 11.18.6.2
full-speed frame generation, 11.18.3
GET_TT_STATE, 11.24.2.8
host controller and, 11.14.1.2
hub architecture and, 11.1.1
hub class descriptors and, 11.23.1
loss of synchronization, 11.18.6, 11.22.1
low-speed signaling, 8.6.5
microframe pipelines and, 11.18.2
multiple TTs, 11.14.1.3, 11.23.1, 11.24.2.8
resetting, 11.24.2.9
response generation, 11.18.5
scheduling, 11.14.2 to 11.14.2.3
split transaction notation, 11.15
state machines
bulk/control state machines, 11.17.2
declarations, B.3
interrupt transaction state machines, 11.20.2
isochronous transaction state machines, 11.21.2
overview, 11.16, 11.16.2 to 11.16.2.1.7
stopping normal execution, 11.24.2.11
"think time," 11.18.2, 11.23.2.1
in transactions
bulk/control transactions, 11.17.2, 11.17.4
interrupt transactions, 11.20.2
isochronous transactions, 11.21.2, 11.21 to 11.21.4
non-periodic transactions, 11.17 to 11.17.5
periodic transactions, 11.18 to 11.18.8, 11.22.1
transaction tracking, 11.18.7
transceivers
downstream facing ports and hubs, 7.1.4.2, 7.1.7.1
full- and high-speed signaling, 7.1, 7.1.1.1
lumped capacitance guidelines for transceivers, 7.1.6.2
transfer management, 5.11.1 to 5.11.1.5
allocating bandwidth, overview, 4.7.5, 5.11.1.1
client software, 5.11.1.1
HCD, 5.11.1.3
Host Controller, 5.11.1.5
illustrated, 5.11.1
transaction list, 5.11.1.4
USB driver, 5.11.1.2
USB System, 10.3.2
transfers, 5.0. *See also* transactions; *names of specific transfer types (i.e., bulk transfers)*
bulk transfers, 2.0 *glossary,* 4.7.2, 5.8 to 5.8.5, 8.5.2
bus access for transfers, 5.11 to 5.11.5
bus bandwidth reclamation, 5.11.5
calculating buffer sizes in functions and software, 5.11.4
calculating bus transaction times, 5.11.3
transaction tracking, 5.11.2
transfer management, 5.11.1 to 5.11.1.5
bus protocol overview, 4.4
bus topology, 5.2 to 5.2.5
communication flow, 4.1, 5.3 to 5.3.3
control transfers, 4.7.1, 5.5 to 5.5.5, 8.5.2, 8.5.3 to 8.5.3.4
data prebuffering, 5.12.5
data signaling overview, 7.1.7.4 to 7.1.7.4.2
defined, 2.0 *glossary*
error detection and recovery, 8.7 to 8.7.4
frames and microframes, 5.3.3
high-bandwidth transfers, 5.9.1, 5.9.2
high-speed transfer rates in 2.0, 1.1
Host Controller responsibilities, 4.9, 10.1.3
hub connectivity and, 11.1.2.1
implementer viewpoints, 5.1
interrupt transfers, 2.0 *glossary,* 4.7, 4.7.3, 5.7 to 5.7.5, 5.9.1, 8.5.2, 8.5.4
isochronous transfers, 2.0 *glossary,* 4.7.4, 5.6 to 5.6.5, 5.9.2, 5.12 to 5.12.8, 8.5.5
operations overview, 9.2.4
organization of transactions within frames, 5.11.2
overview, 5.0
periodic transfers, 5.6.4, 5.7.4
power management, 9.2.5
request processing, 9.2.6 to 9.2.6.6
standard device requests, 9.4 to 9.4.11
time limits for completing, 9.2.6.4, 9.2.6.5
transaction formats, 8.5 to 8.5.5
transfer types, 4.7 to 4.7.5, 5.4 to 5.8.5
USB device requests, 9.3 to 9.3.5
USBD role in, 10.1.1, 10.5.3 to 10.5.3.2.3
USB System role in, 10.3.3

Microchip Ex. 1018, p. 645
Microchip v. Aptiv
IPR2024-00494

transfer types. *See also* transactions; transfers; *names of specific transfer types (i.e., bulk transfers)*
allocating USB bandwidth, 4.7.5
bulk transfers, 2.0 *glossary,* 4.7.2, 5.8
in calculating transaction times, 5.11.3
control transfers, 4.7.1, 5.5
endpoint field indicators, 9.6.6
high-bandwidth transfers, 5.9.1, 5.9.2
interrupt transfers, 2.0 *glossary,* 4.7.3, 5.7
isochronous transfers, 2.0 *glossary,* 4.7.4, 5.6
for message pipes, 5.3.2.2
overview, 4.7 to 4.7.5, 5.4
pipes and, 4.4
split transactions and, 5.10
for stream pipes, 5.3.2.1
transfer types defined, 2.0 *glossary*
transitions in state machines, 8.5, 11.15
transmission envelope detectors, 7.1.4.2, 7.1 *Table 7-1*
transmit clock, 11.7.1.3
transmit eye patterns, 7.1, 7.1.2
transmit phase of signaling, 7.1.1
TransmitR state, 11.5.1.8
Transmit state, 11.5, 11.5.1.7
Transmitter/Receiver Test Fixture, 7.1.2.2 *Figure 7-12*
transmitters
Active state, 11.6.4.2
Generate End of Packet Towards Upstream Port state (GEOPTU), 11.6.4.5
Inactive state, 11.6.4.1
RepeatingSE0 state, 11.6.4.3
SendJ state, 11.6.4.4
Send Resume state (Sresume), 11.6.4.6
transmitter data jitter, 7.1.13.1.1
transmitter sequence bits, 8.6, 8.6.2
transmitter state descriptions, 11.6.4
transmitter state machine, 11.6, 11.6.4
transmitter state machine, 11.6, 11.6.4
transmit waveform requirements, 7.1.2.2 *Figure 7-13,* 7.1.2.2 *Figure 7-14,* 7.1.2.2 *Figure 7-17*
TrueRWU signal/event, 11.5 *Figure 11-10,* 11.5 *Table 11-5*
truncated packets, 11.3.2
TT. *See* Transaction Translator
TT_BulkCS state machine, 11.16.2.1.4
TT_BulkSS state machine, 11.16.2.1.3
TT_Do_BICS state machine, 11.17.2
TT_Do_BISS state machine, 11.17.2
TT_Do_BOCS state machine, 11.17.2
TT_Do_BOSS state machine, 11.17.2
TT_Do_complete state machine, 11.16.2.1.2
TT_Do_IntICS state machine, 11.20.2
TT_Do_IntISS state machine, 11.20.2
TT_Do_IntOCS state machine, 11.20.2
TT_Do_IntOSS state machine, 11.20.2
TT_Do_IsochISS state machine, 11.21.2
TT_Do_IsochOSS state machine, 11.21.2
TT_Do_Start state machine, 11.16.2.1.1
TT_Flags bits, 11.24.2.8
TT_IntCS state machine, 11.16.2.1.6
TT_IntSS state machine, 11.16.2.1.5
TT_IsochICS state machine, 11.21.2
TT_IsochSS state machine, 11.16.2.1.7
TT_Process_Packet state machine, 11.16.2.1
TT_Return_Flags field, 11.24.2.8
TT_specific_state field, 11.24.2.8
turn-around times
defined, 2.0 *glossary*
error detection, 8.7.2
overview, 7.1.18 to 7.1.18.2
turning power on for ports, 11.11
twisted data pair in cables, 6.6.1
Tx_active signal/event, 11.6.3 *Table 11-8*
Tx_resume signal/event, 11.6.3 *Table 11-8*

U

UEOP signal/event, 11.7.2.3 *Table 11-11*
UL listing for cables, 6.6.5
UL STD-94, 6.7.1
UL Subject-444, 6.6.5, 6.7.1
unacceptable cables, 6.4.4
underplating
plug contact materials, 6.5.4.3
plug shell materials, 6.5.4.2
receptacle contact materials, 6.5.3.3
receptacle shell materials, 6.5.3.2
Underwriter's Laboratory, Inc., 6.6.5, 6.7.1
The Unicode Standard, Worldwide Character Encoding, 9.6.7
UNICODE string descriptors, 9.6.7
unique addresses
assigning after dynamic insertion or removal, 4.6.3
device initialization, 10.5.1.1
operations overview, 9.2.2
SetAddress() request, 9.4.6
time limits for completing addressing, 9.2.6.3
Universal Serial Bus
architectural extensions, 4.10
backwards compatibility, 3.1
bus protocol, 4.4
clock model, 5.12, 5.12.2
components, 5.1
configuration, 4.6 to 4.6.3, 10.3.1
data flow and transfers, 4.7 to 4.7.5, 5.1 to 5.10.8
description, 4.1 to 4.1.1.2
feature list, 3.3
goals, 3.1

Microchip Ex. 1018, p. 646
Microchip v. Aptiv
IPR2024-00494

Universal Serial Bus *(Continued)*
high-speed applications, 3.2
host hardware and software, 4.9, 10.2 to 10.6
hubs, 11.1 to 11.16
mechanical and electrical specifications, 6.1 to 6.9, 7.1 to 7.1.20, 7.3 to 7.3.3
motivation for development, 1.1
physical interface, 4.2 to 4.2.2
power distribution, 4.3 to 4.3.2, 7.2 to 7.2.4.2
protocol layer, 8.1 to 8.7
range of USB data traffic workloads, 3.2
robustness and error detection/recovery, 4.5 to 4.5.2
USB device framework, 9.1 to 9.7.3
USB devices, 4.8 to 4.8.2.2
USB schedule, 4.1
Universal Serial Bus Driver. *See* USBD (USB Driver)
Universal Serial Bus Resources, 2.0 *glossary*
up counters in hub timing, 11.2.3.1
upgrade paths, 3.3
upstream facing ports and hubs
defined, 4.8.2.1
driver speed and, 7.1.2.3
full-speed port transceiver, 7.1, 7.1.7.1
high-speed detection, 7.1.5.2
high-speed signaling and, 11.1.1
hub architecture, 11.1.1
hub EOP delay and EOP skew, 7.3.3 *Figure 7-53*
input capacitance, 7.1.6.1
jitter, 7.3.2 *Table 7-10*
low-speed source electrical characteristics, 7.3.2 *Table 7-10*
receivers, 11.6.3 to 11.6.3.9
reset on upstream port, 11.10
reset state machines, C.2
signaling delays, 7.1.14.1
signaling speeds and, 7.1
test mode support, 7.1.20
transmitters, 11.6 to 11.6.4.6
upstream connectivity defined, 11.1.2.1
upstream defined, 2.0 *glossary*
upstream hub delay, 7.3.3 *Figure 7-52*
upstream facing transceivers, signaling speeds and, 7, 7.1
upstream packets (HSU2), 8.5
upstream plugs, 6.2
Usage Types, 9.6.6
USB. *See* Universal Serial Bus
USB 2.0 Adopters Agreement, 1.4
USB Bus Interface layer
in bus topology, 5.2.2
detailed communication flow illustrated, 5.3
Host Controller implementation, 10.1.1
illustrated, 5.1
interlayer communications model, 10.1.1
USBD (USB Driver). *See also* USBDI (USB Driver Interface)
in bus topology, 5.2.1
command mechanisms, 10.5.1 to 10.5.2.12
as component of USB System, 10.1.1
configuration and, 10.3.1
control mechanisms, 10.1.2
data transfer mechanisms, 10.1.3
defined, 2.0 *glossary,* 5.3, 10.5
driver characteristics, 7.1.1
driver speed and, 7.1.2.3
full-and low-speed drivers, 7.1.1.1, 7.1.1.2
HCD interaction with, 10.4
hub drivers, 10.3.1
initialization, 10.5.1.1
overview, 10.5.1
passing preboot control to operating system, 10.5.5
pipe mechanisms, 5.11.1.2, 10.5.1 to 10.5.3.2.4
request data format mechanisms, 10.3.4
service capabilities, 10.5.1.3
software interface overview, 10.3
in transfer management, 5.11.1, 5.11.1.2
USB System and, 10.5.4 to 10.5.4.5
USB device framework, 9, 9.7.2
descriptors, 9.5 to 9.7.3
device class definitions, 9.7 to 9.7.3
generic USB device operations, 9.2 to 9.2.7
address assignment, 9.2.2
configuration, 9.2.3
data transfer, 9.2.4
dynamic attachment and removal, 9.2.1
power management, 9.2.5
request error, 9.2.7
request processing, 9.2.6 to 9.2.6.6
standard descriptor definitions, 9.6 to 9.6.7
standard device requests, 9.4 to 9.4.11
USB device requests, 9.3 to 9.3.5
USB device states, 9.1 to 9.1.2
USB Device layer
detailed communication flow illustrated, 5.3
illustrated, 5.1
interlayer communications model, 10.1.1
USB devices. *See* devices

Microchip Ex. 1018, p. 647
Microchip v. Aptiv
IPR2024-00494

USBDI (USB Driver Interface)
adding devices, 10.5.2.5
alternate interface mechanisms, 10.5.2.10
getting descriptors, 10.5.2.3
removing devices, 10.5.2.6
role in request data format, 10.3.4
sending class commands, 10.5.2.8
sending vendor commands, 10.5.2.9
setting descriptors, 10.5.2.12
software interface overview, 10.3
USB host. *See* host
USB host controller. *See* Host Controller
USB Icon, 6.5, 6.5.1
USB-IF (USB Implementers Forum, Inc.), 1.4, 2.0 *glossary*
USB Implementers Forum, 1.4, 2.0 *glossary*
USB interconnect model, 4.1, 5.12.4.4
USB Logical Devices. *See* logical devices
USB Physical Devices. *See* physical devices
USB schedule, 4.1
USB Specification Release Number, 9.6.1
USB System. *See also* HCD; host software; USBD
allocating bandwidth, 10.3.2
buffers and, 10.2.9
data transfer role, 10.3.3
HCD component, 10.1.1
Host Controller interaction, 10.1.1
host software component, 10.1.1
power management, 10.5.4.2
remote wakeup, 10.2.7, 10.5.4.5
software interface overview, 10.3
state handling, 10.2.1
status and activity monitoring, 10.1.4
USBD component, 10.1.1
USB System Software
asynchronous data transfers, 4.9
bus enumeration, 4.9
in bus topology, 5.2.1
in communication flow, 5.3
detecting hub and port status changes, 11.12.2
as implementation focus area, 5.1
interrupt transfer support, 5.7.3
isochronous transfer support, 4.9
power management, 4.9
role, 4.9

V

variable-length data stages, 8.5.3.2
variable-sized data payloads, 5.3.2
VBus leads
bypassing, 7.2.4.1
cable electrical characteristics, 7.3.2 *Table 7-12*
detachable cables, 6.4.1
VBus leads *(Continued)*
in electrical specifications overview, 4.2.1
high-/full-speed captive cable assemblies, 6.4.2
low-speed captive cable assemblies, 6.4.3
standardized contact terminating assignments, 6.5.2
upstream port power supply and, 7.2.1
Vendor IDs in device descriptors, 9.6.1
vendor information in device characteristics, 4.8.1
vendor-specific descriptors, 9.5
vendor-specific requests, 10.5.2.9
version numbers in device descriptors, 9.6.1
version numbers in device_qualifer descriptor, 9.6.2
VHDL syntax, 11.15
V/I characteristics of full-speed connections, 7.1.1.1
virtual devices, 2.0 *glossary,* 5.12.4.4
visible device states, 9.1.1
visual inspection standards, 6.7 *Table 6-7*
voltage
average voltage on D+/D- lines, 7.1.2.1
cross-over voltage in signaling, 7.1.2.1
DC output voltage specifications, 7.1.6.2
droop, 7.2.3, 7.2.4.1
flyback voltage, 7.2.4.2
full-speed connections, 7.1.1.1
high-speed signaling and, 7.1
open-circuit voltage, 7.1.1
ratings for cables, 6.6.3
reduction due to cable resistive effects, 7.2.3
reversing in high-speed signaling, 7.1.1.3
supply voltage, 7.3.2 *Table 7-7*
test mode, 7.1.20
voltage drop budget, 7.2.2
voltage drops, 7.2.1.1
voltage drop topology, 7.2.2
zero impedance voltage sources, 7.1.1

W

Wait for End of Packet from Upstream Port state (WFEOPFU), 11.7.2.3 *Figure 11-16,* 11.7.4
Wait for End of Packet (WFEOP) state, 11.7.2.3 *Figure 11-16,* 11.7.6
Wait for Start of Packet from Upstream Port state (WFSOPFU), 11.7.2.3 *Figure 11-16,* 11.7.3
Wait for Start of Packet (WFSOP) state, 11.7.2.3 *Figure 11-16,* 11.7.5
wander, defined, 11.2.5.2

Microchip Ex. 1018, p. 648
Microchip v. Aptiv
IPR2024-00494

waveforms
differential data jitter, 7.3.3 *Figure 7-49*
differential-to-EOP transition skew and EOP width, 7.3.3 *Figure 7-50*
full-speed driver signal waveforms, 7.1.1.1
hub differential delay, differential jitter, and SOP distortion, 7.3.3 *Figure 7-52*
hub EOP delay and EOP skew, 7.3.3 *Figure 7-53*
maximum input waveforms for signaling, 7.1.1
receiver jitter tolerance, 7.3.3 *Figure 7-51*
testing, 7.1.20
WFEOPFU state, 11.5.1.6, 11.7.2.3 *Figure 11-16,* 11.7.4
WFEOP state, 11.7.2.3 *Figure 11-16,* 11.7.6
WFSOPFU state, 11.7.2.3 *Figure 11-16,* 11.7.3
WFSOP state, 11.7.2.3 *Figure 11-16,* 11.7.5
wHubChange field, 11.24.2.6
wHubCharacteristics field
hub descriptor, 11.23.2.1
multiple gangs and, 11.11.1
over-current reporting, 11.12.5
port indicator status, 11.5.3
power switching settings, 11.11
wHubStatus field, 11.24.2.6
wIndex field
hub class requests, 11.24.2
overview, 9.3.4
Setup data format, 9.3
standard device requests, 9.4
wire gauge in cables, 6.6.2
wire insulation in cables, 6.6.2
wiring assignments for conductors, 6.5.2
wLANGID[] field (string descriptors), 9.6.7
wLength field
hub class requests, 11.24.2
overview, 9.3.5
Setup data format, 9.3
standard device requests, 9.4
wMaxPacketSize field
bulk transfers and, 5.8.3
control transfer packet size, 5.5.3
endpoint descriptors, 9.6.6, 11.23.1
high bandwidth endpoints and, 5.9
interrupt transfer packet size, 5.7.3
variable-length data stages, 8.5.3.2
words, defined, 2.0 *glossary*
working space, location and length of, 10.3.4
worst-case bit stuffing, 5.11.3
worst-case signal delay, 7.1.17.1, 7.1.17.2
wPortChange field, 11.24.2.7, 11.24.2.7.2
wPortStatus field, 11.24.2.7, 11.24.2.7.1
wTotalLength field
configuration descriptors, 9.6.3, 11.23.1
other speed configuration descriptors, 9.6.4, 11.23.1
wValue field
hub class requests, 11.24.2
overview, 9.3.3
Setup data format, 9.3
standard device requests, 9.4

Z

zero impedance voltage sources, 7.1.1
zeroth microframe, 9.4.11, 11.14.2.3, 11.18.3, 11.22.2

Microchip Ex. 1018, p. 649
Microchip v. Aptiv
IPR2024-00494

Microchip Ex. 1018, p. 650
Microchip v. Aptiv
IPR2024-00494