11.21.2 Isochronous Split Transaction State Machines

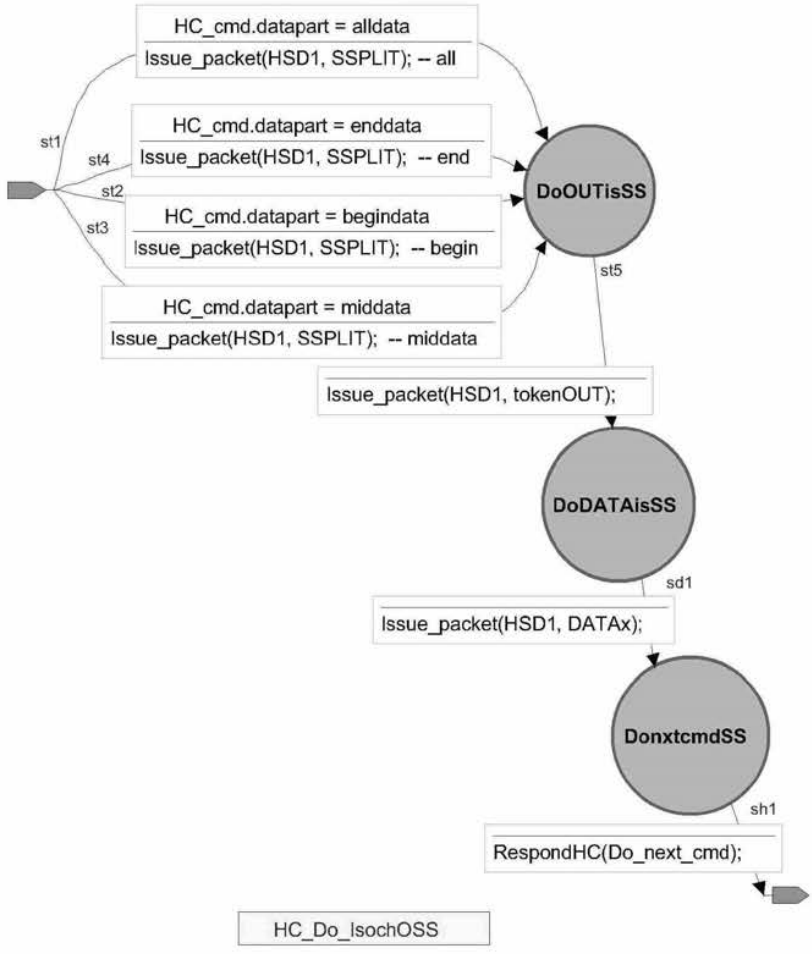


Figure 11-86. Isochronous OUT Start-split Transaction Host State Machine

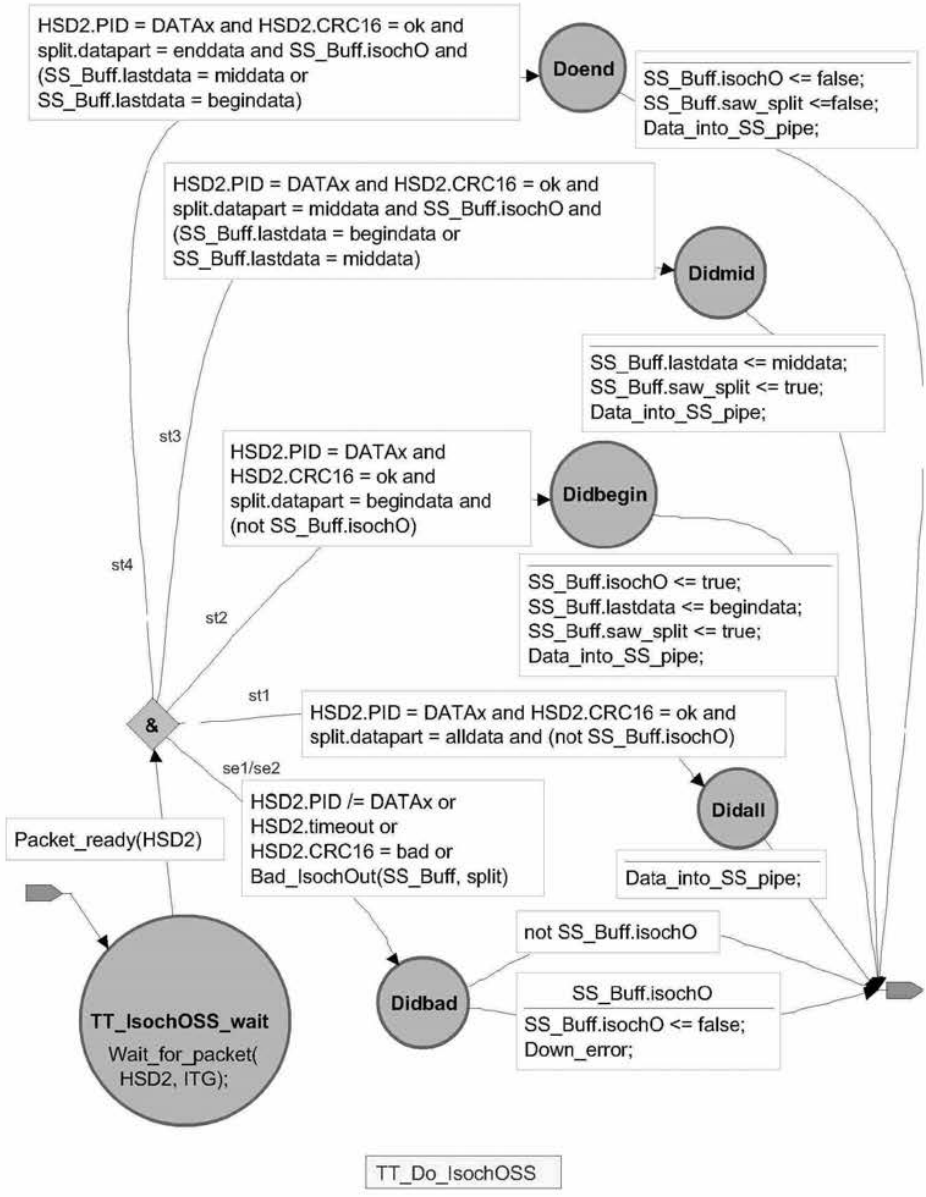


Figure 11-87. Isochronous OUT Start-split Transaction TT State Machine

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.

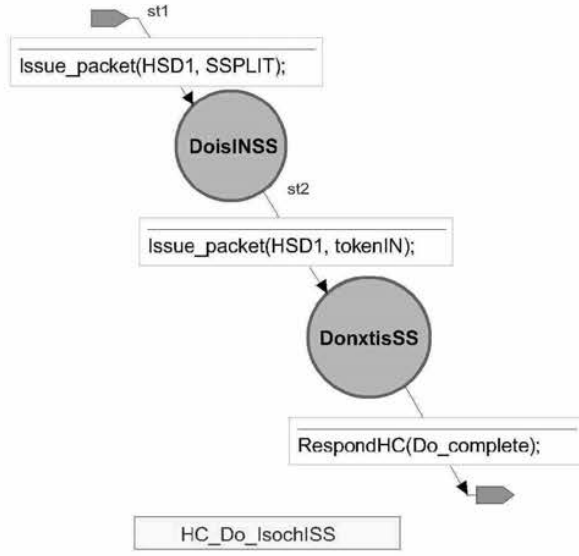


Figure 11-88. Isochronous IN Start-split Transaction Host State Machine

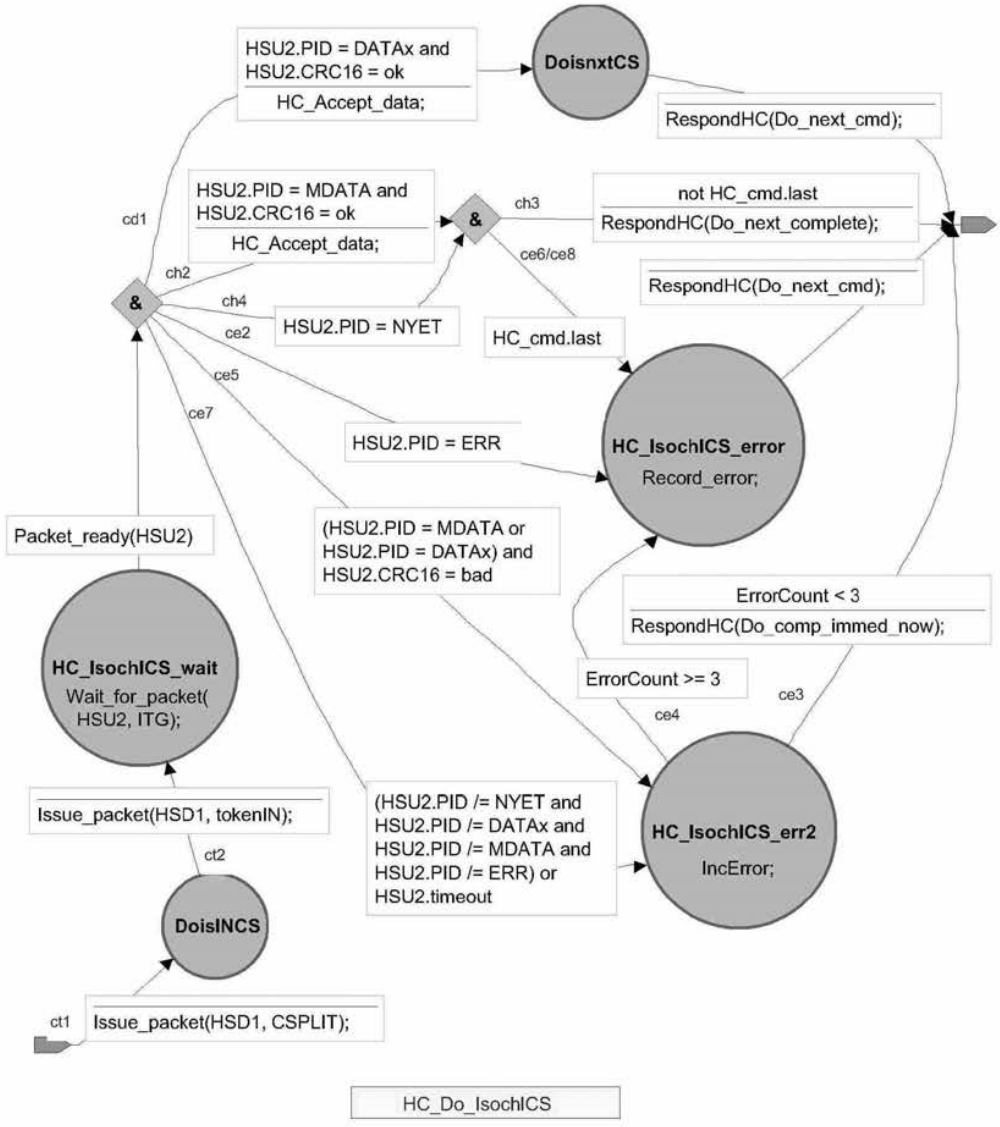
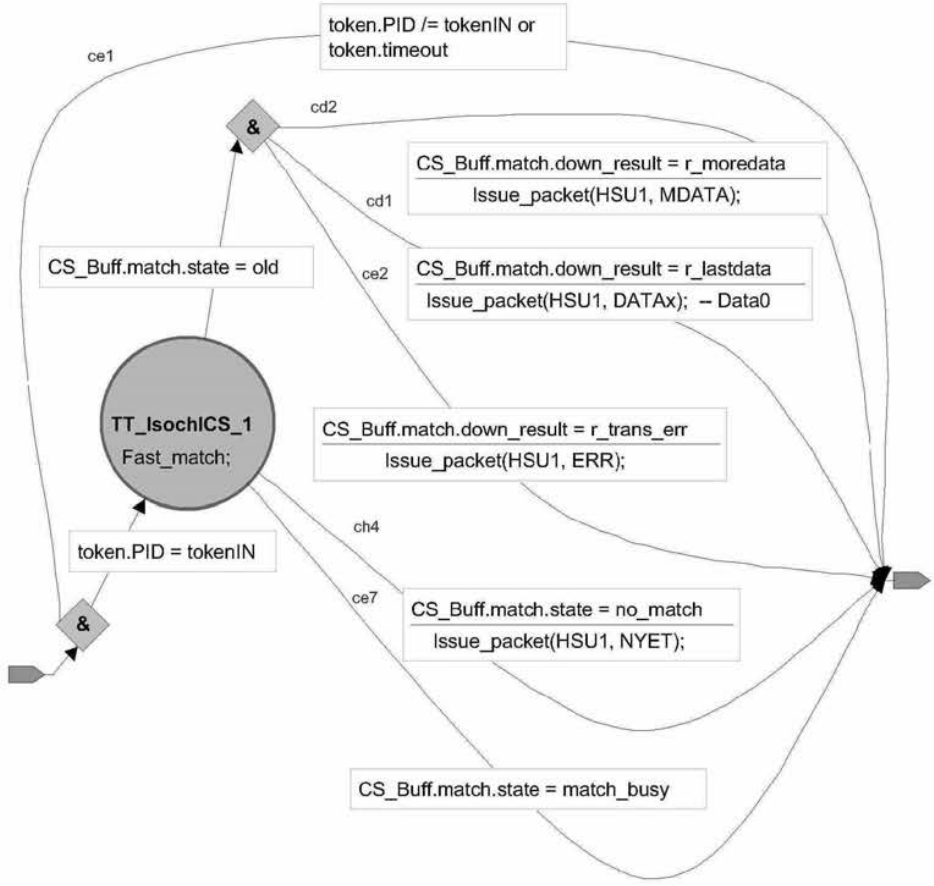


Figure 11-89. Isochronous IN Complete-split Transaction Host State Machine

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).

TT_Do_IsochISS

Figure 11-90. Isochronous IN Start-split Transaction TT State Machine



TT_IsochICS

Figure 11-91. Isochronous IN Complete-split Transaction TT State Machine

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.

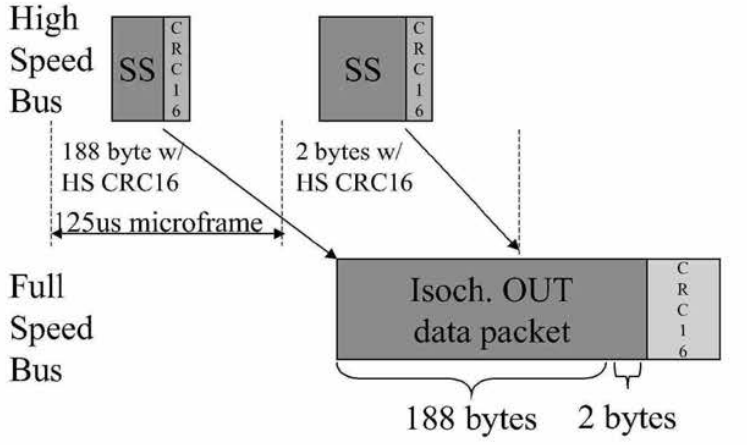


Figure 11-92. Example of CRC16 Isochronous OUT Data Packet Handling

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.

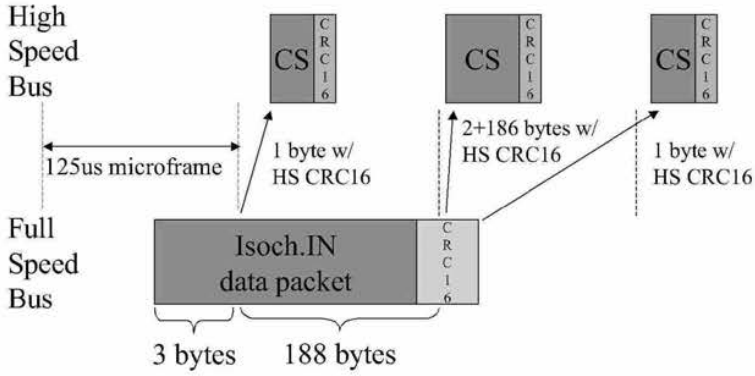


Figure 11-93. Example of CRC16 Isochronous IN Data Packet Handling

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

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.

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).

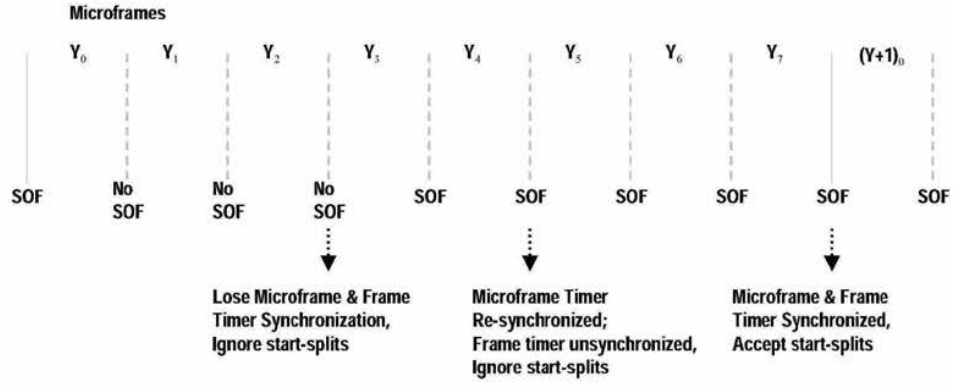


Figure 11-94. Example Frame/Microframe Synchronization Events

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.

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

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

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

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

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)

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)

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

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

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

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.

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.

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.

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

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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	

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.

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.

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

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

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.

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

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

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

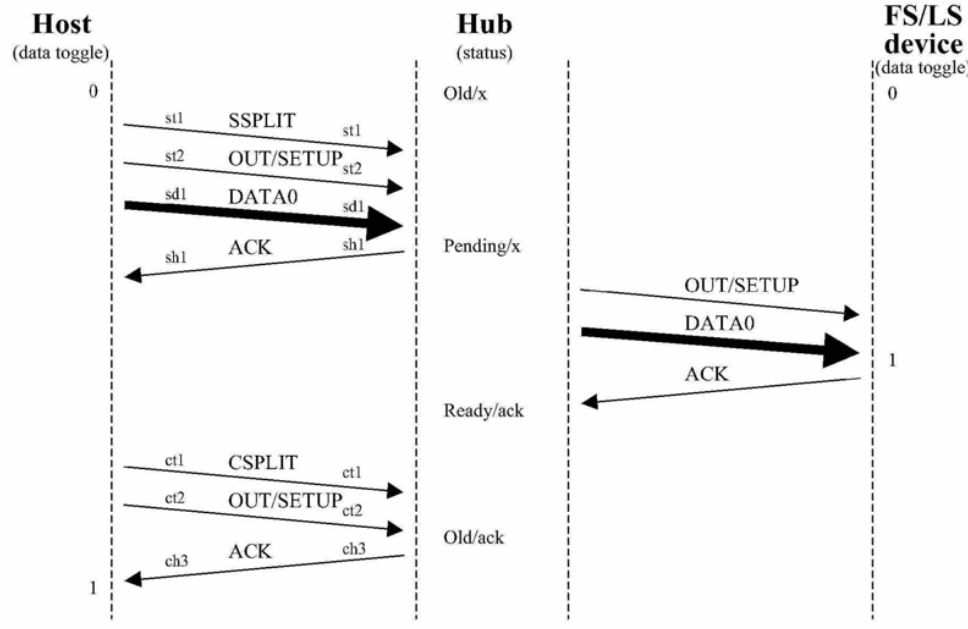


Figure A-1. Normal No Smash

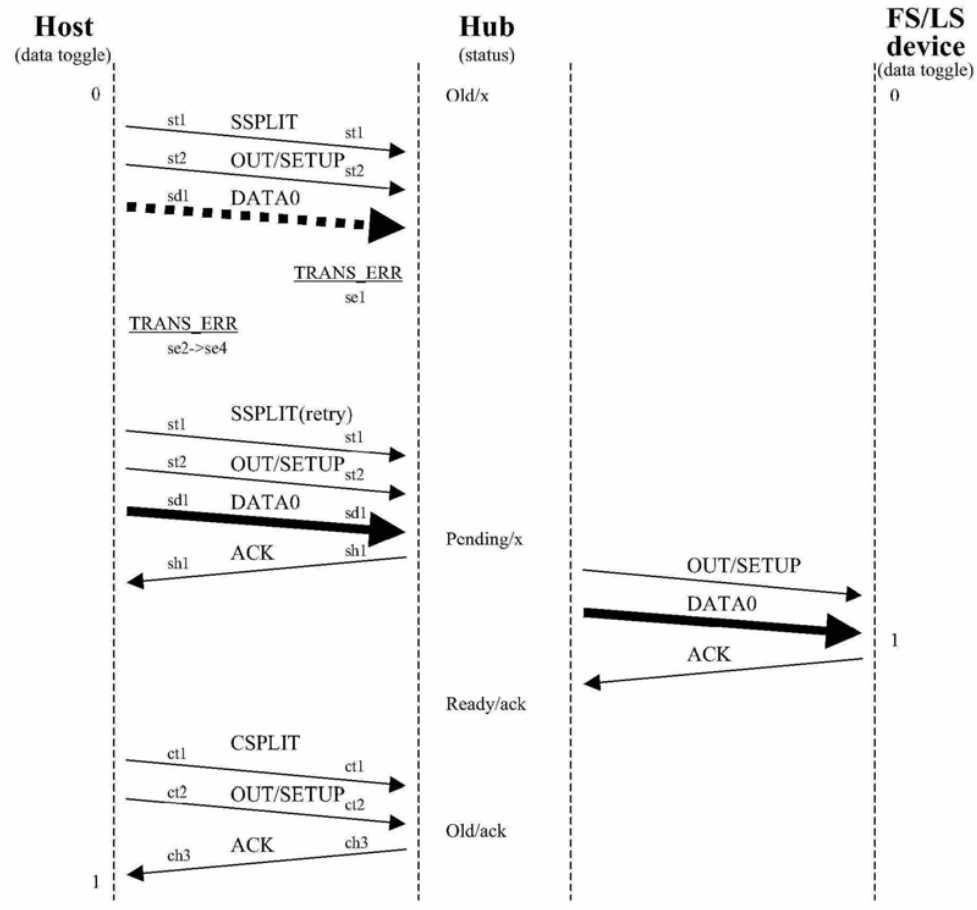


Figure A-2. Normal HS DATA0/1 Smash

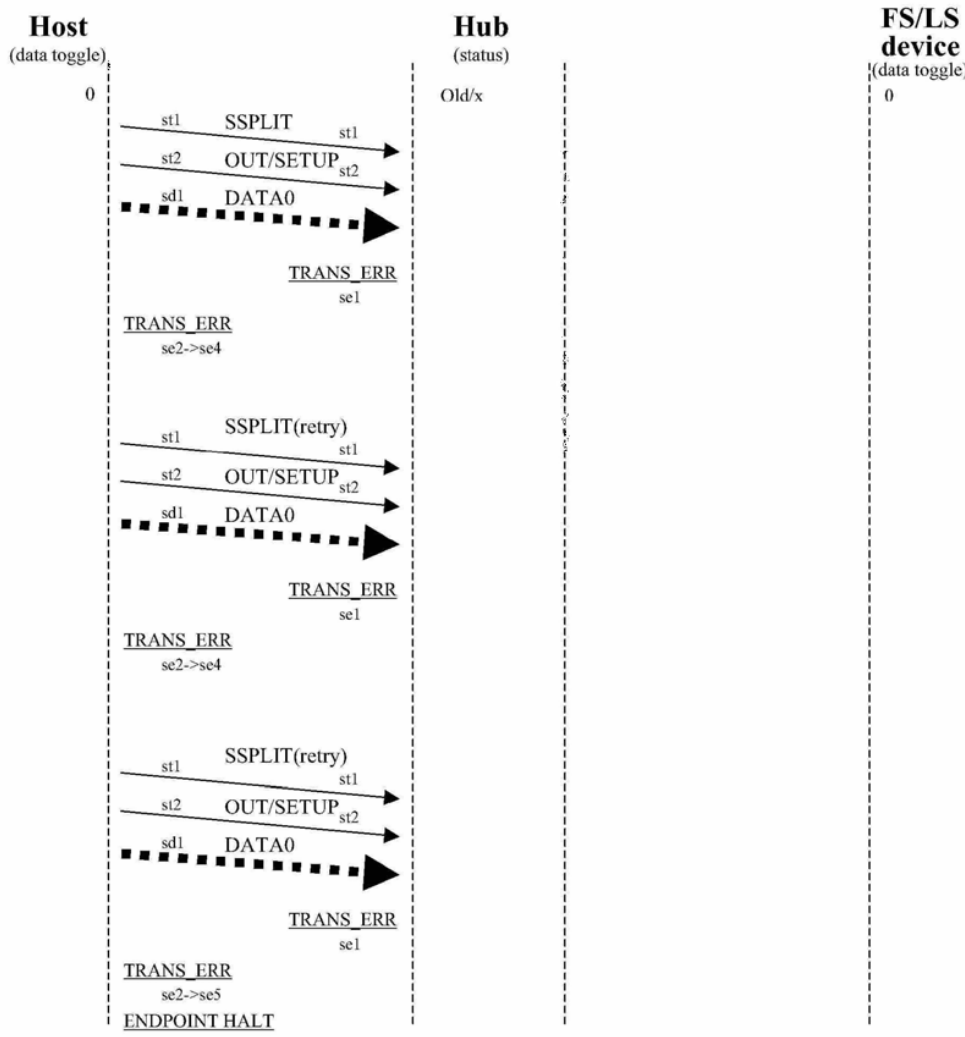


Figure A-3. Normal HS DATA0/1 3 Strikes Smash

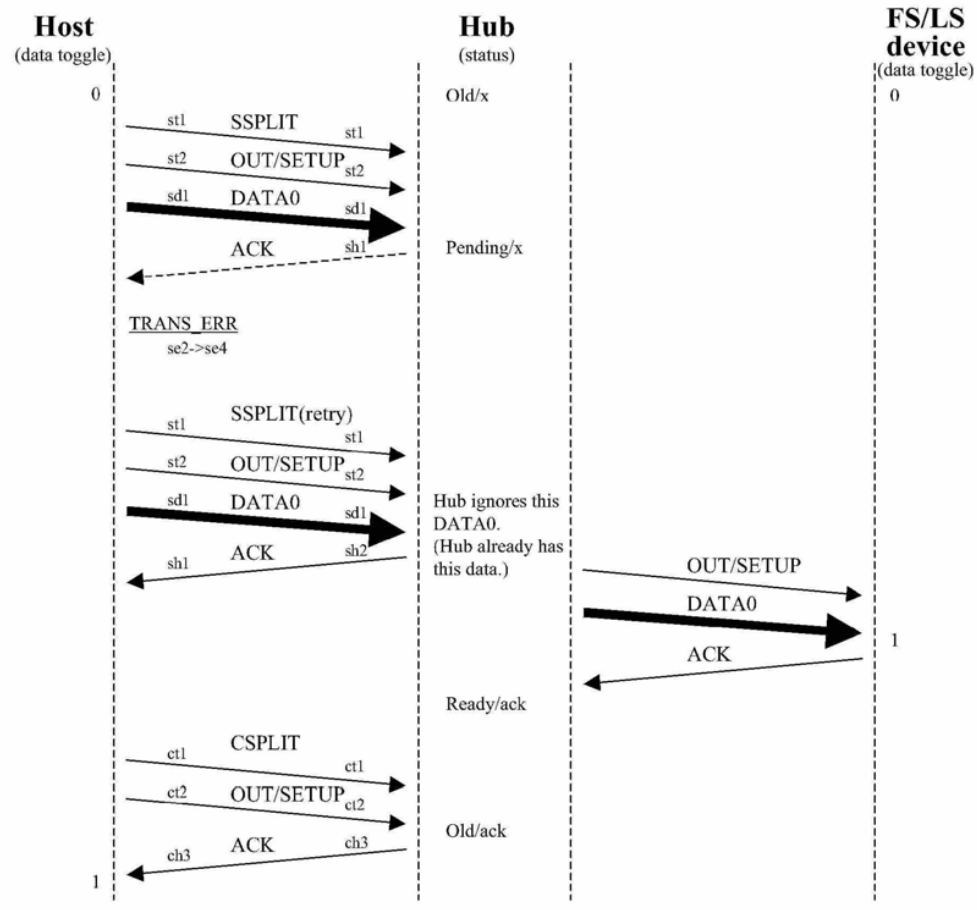


Figure A-4. Normal HS ACK(S) Smash(case 1)

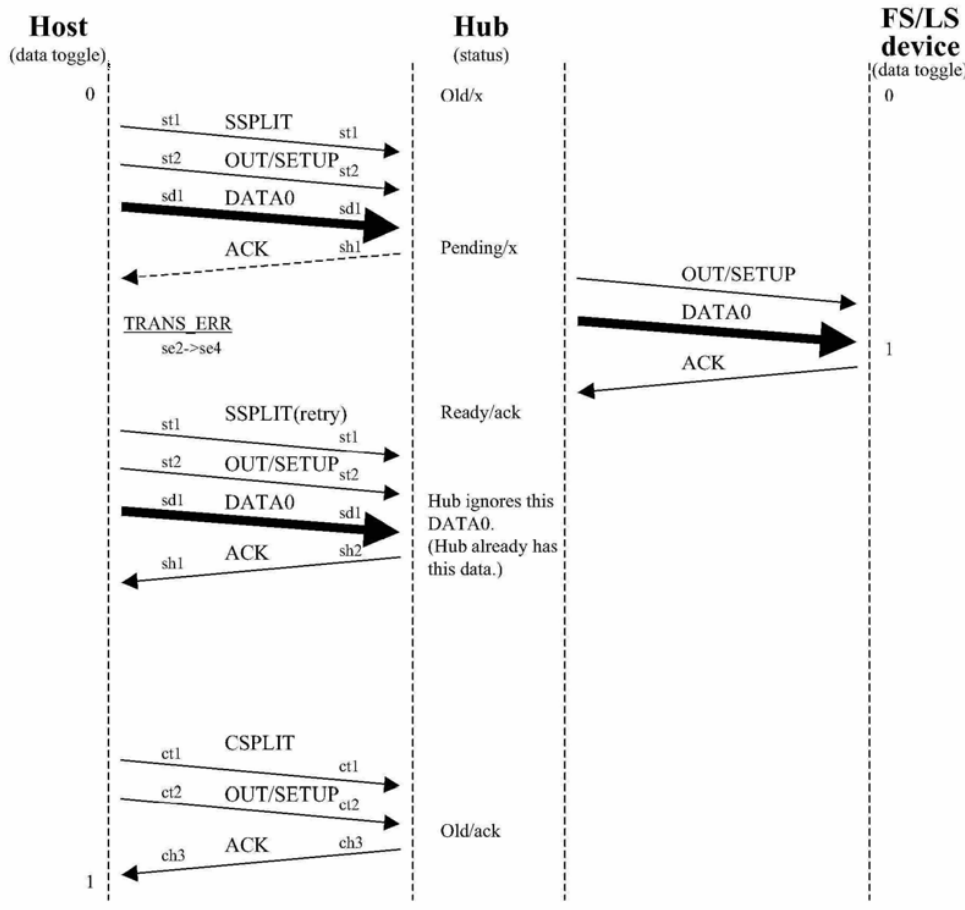


Figure A-5. Normal HS ACK(S) Smash(case 2)

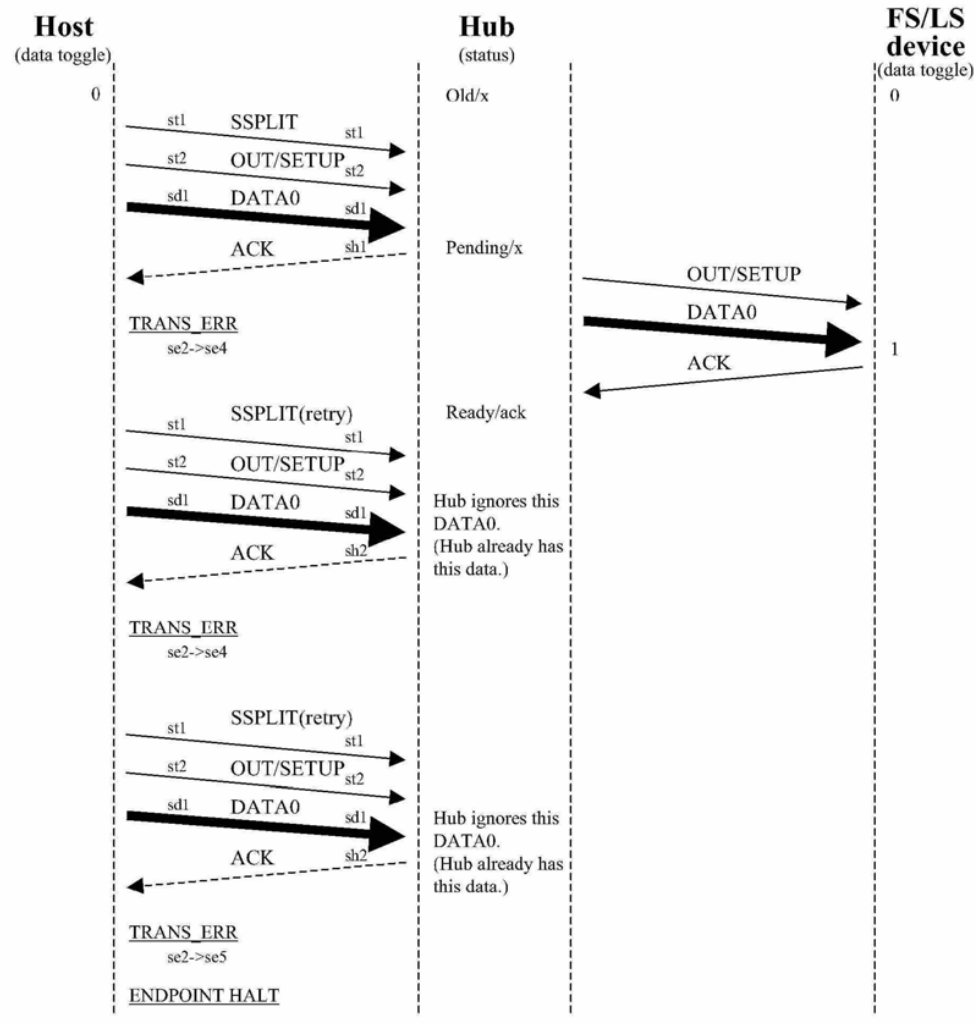


Figure A-6. Normal HS ACK(S) 3 Strikes Smash

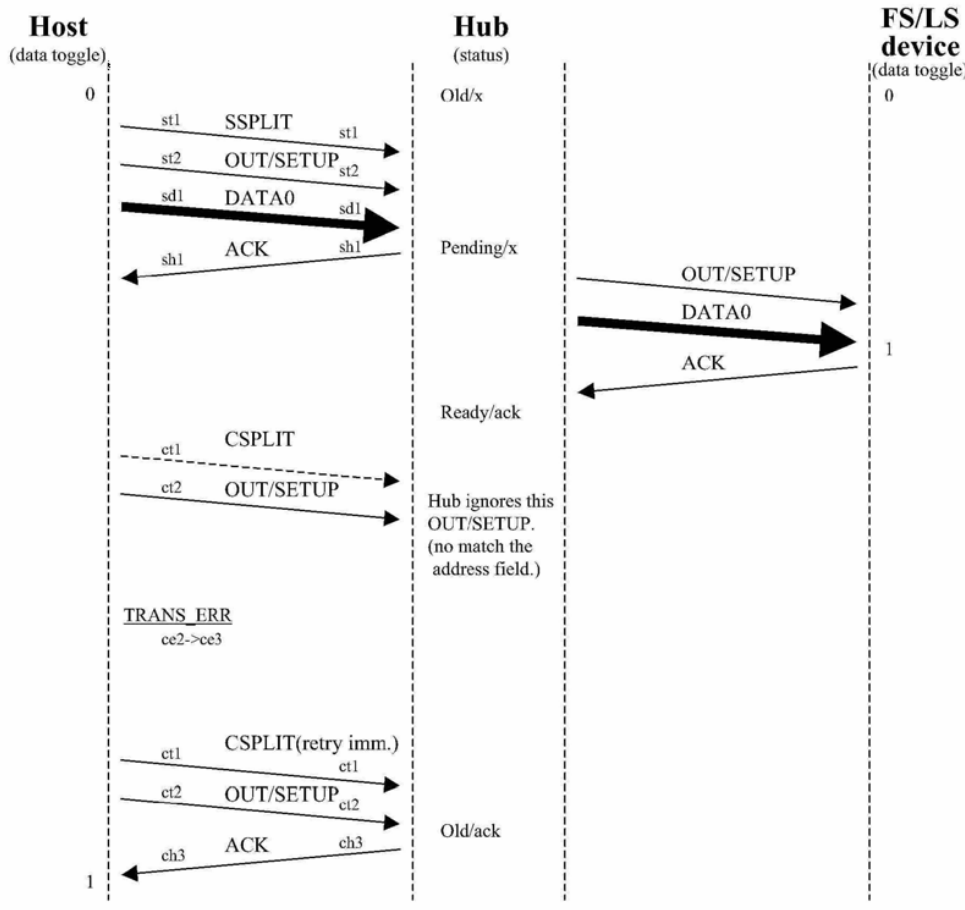


Figure A-7. Normal HS CSPLIT Smash

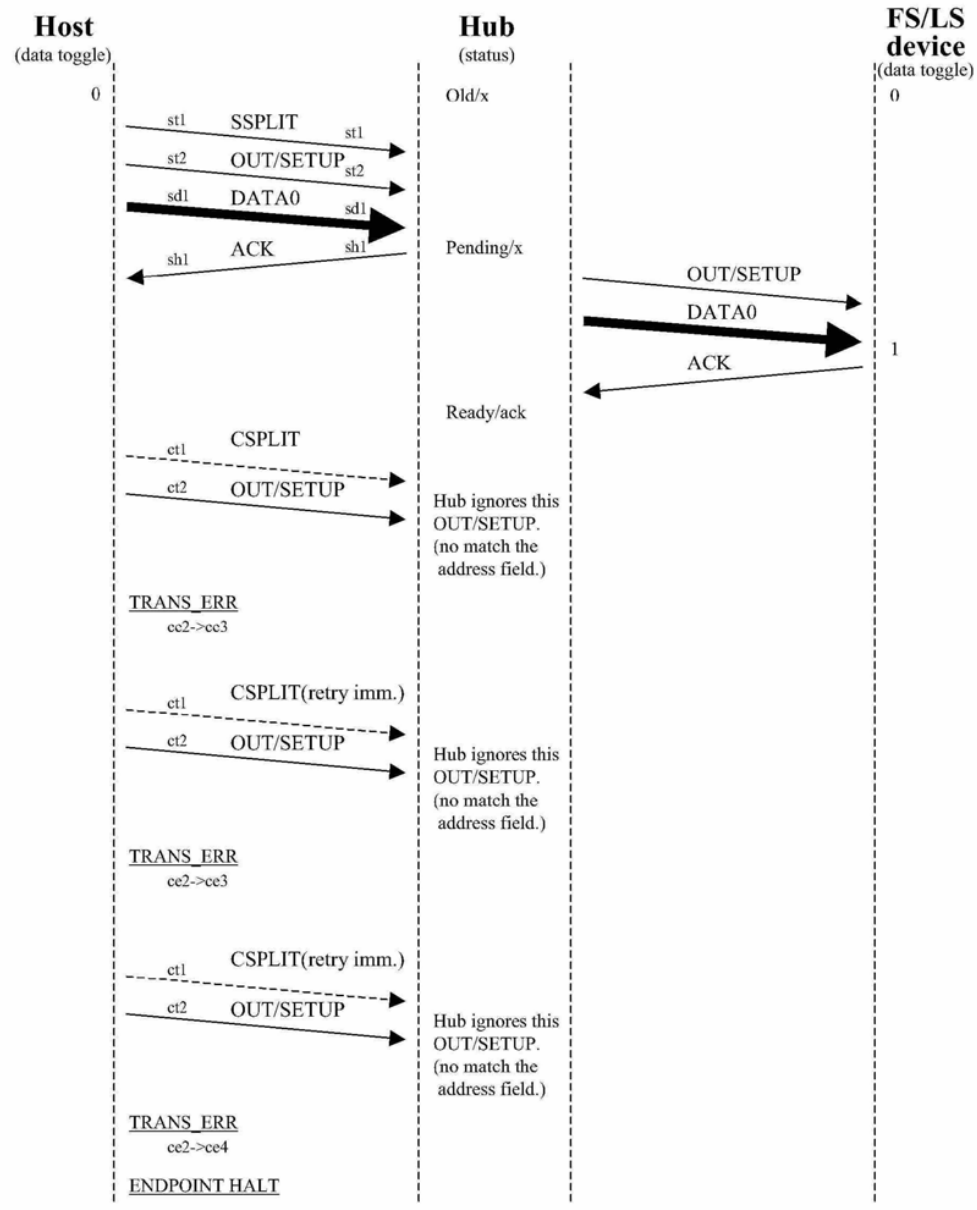


Figure A-8. Normal HS CSPLIT 3 Strikes Smash

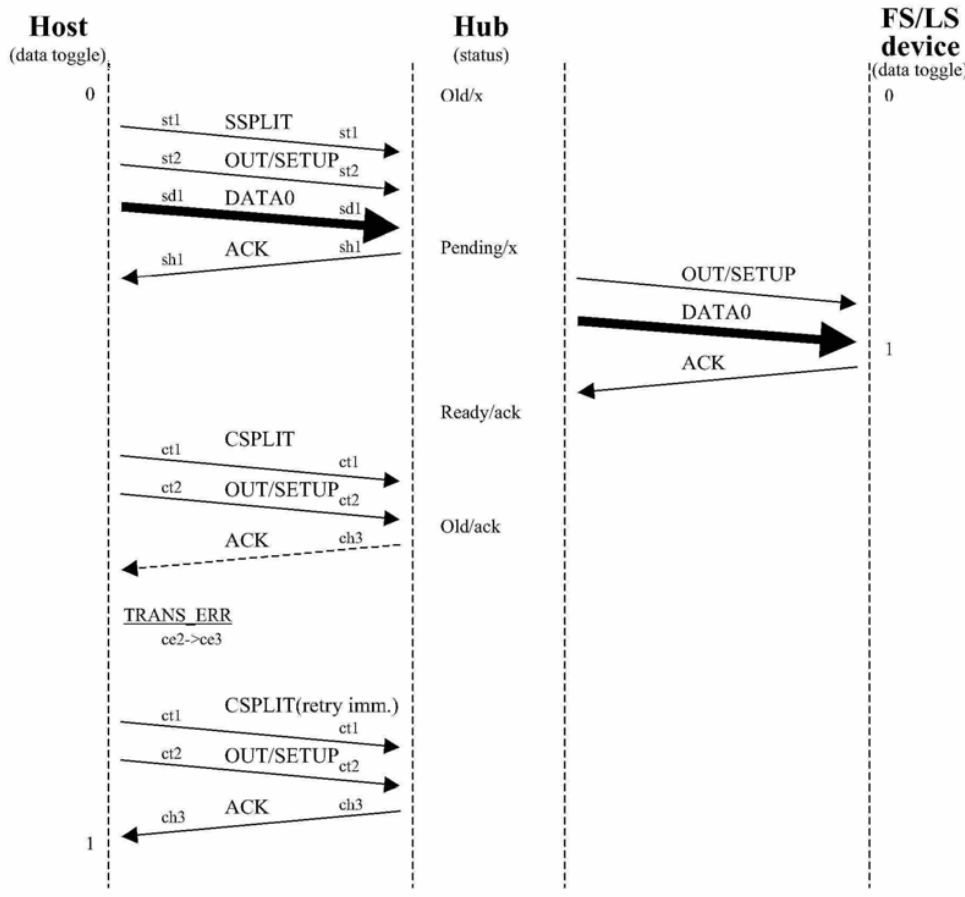


Figure A-9. Normal HS ACK(C) Smash

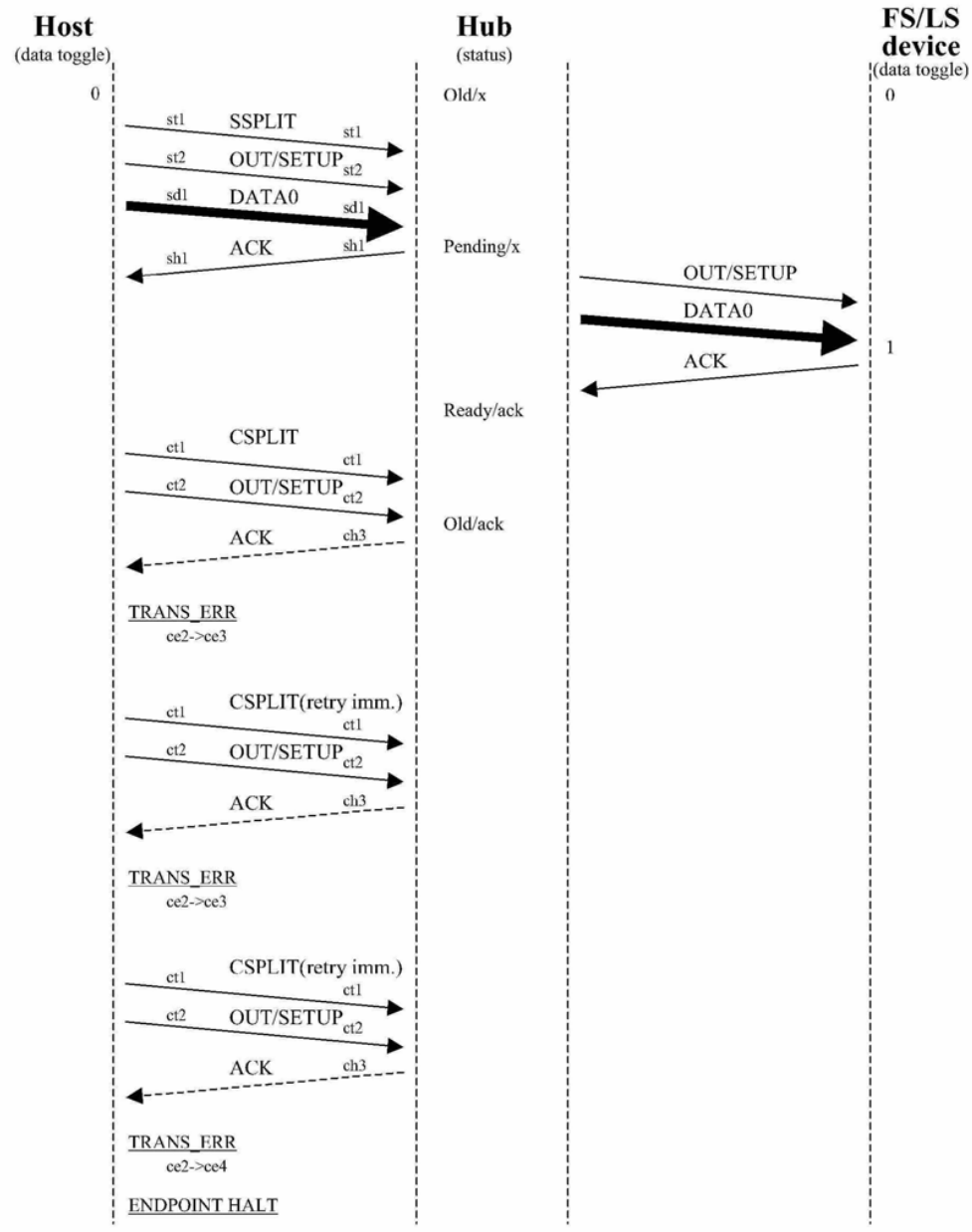


Figure A-10. Normal S ACK(C) 3 Strikes Smash

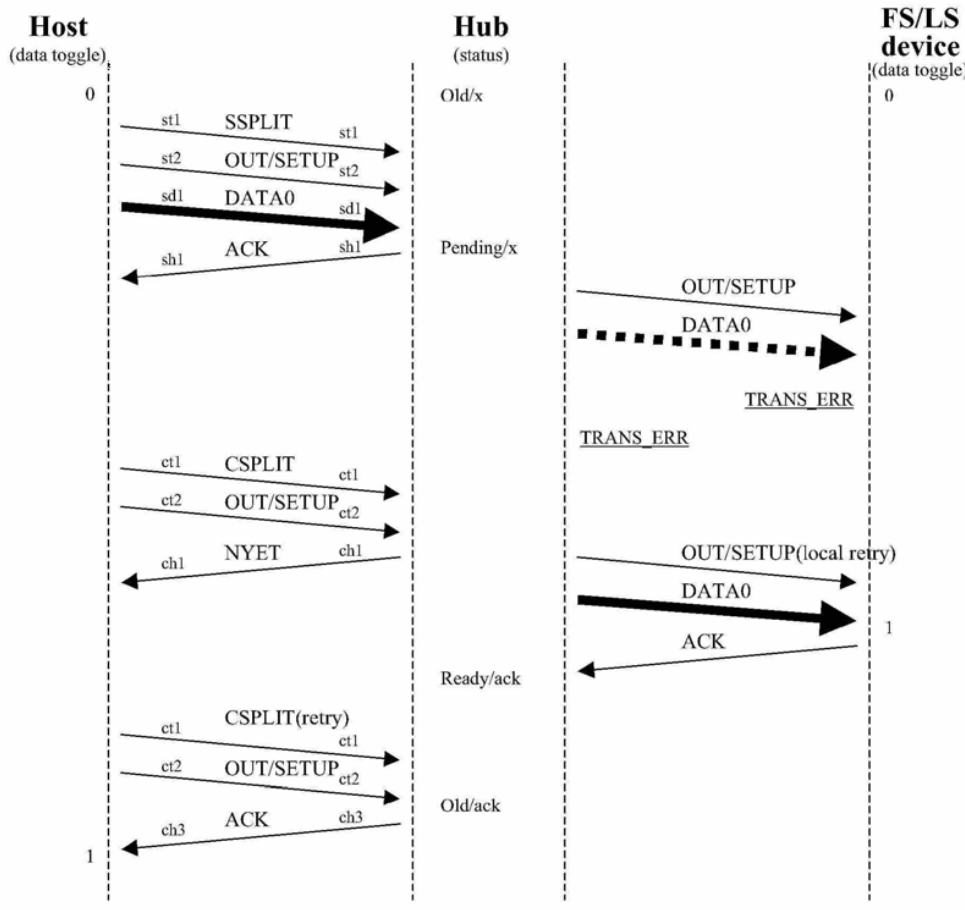


Figure A-11. Normal FS/LS DATA0/1 Smash

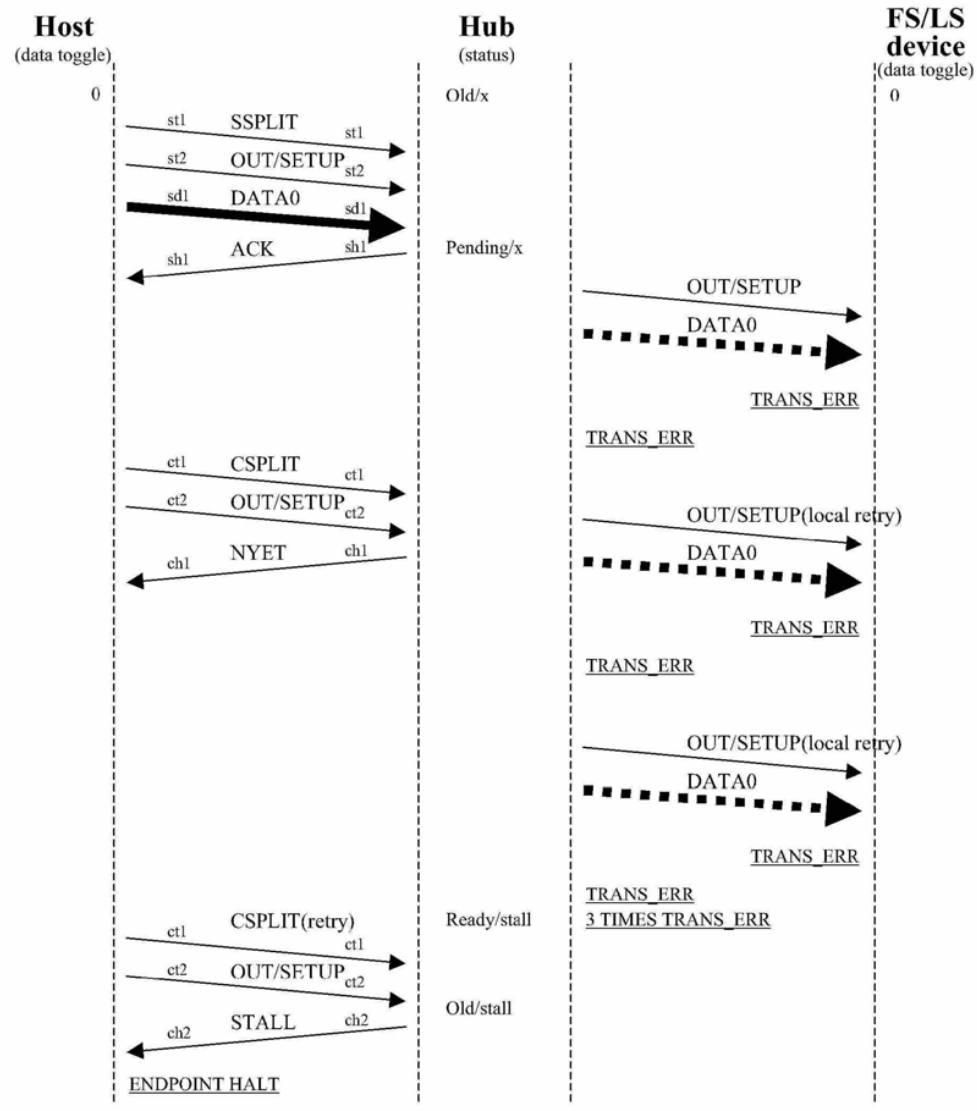


Figure A-12. Normal FS/LS DATA0/1 3 Strikes Smash

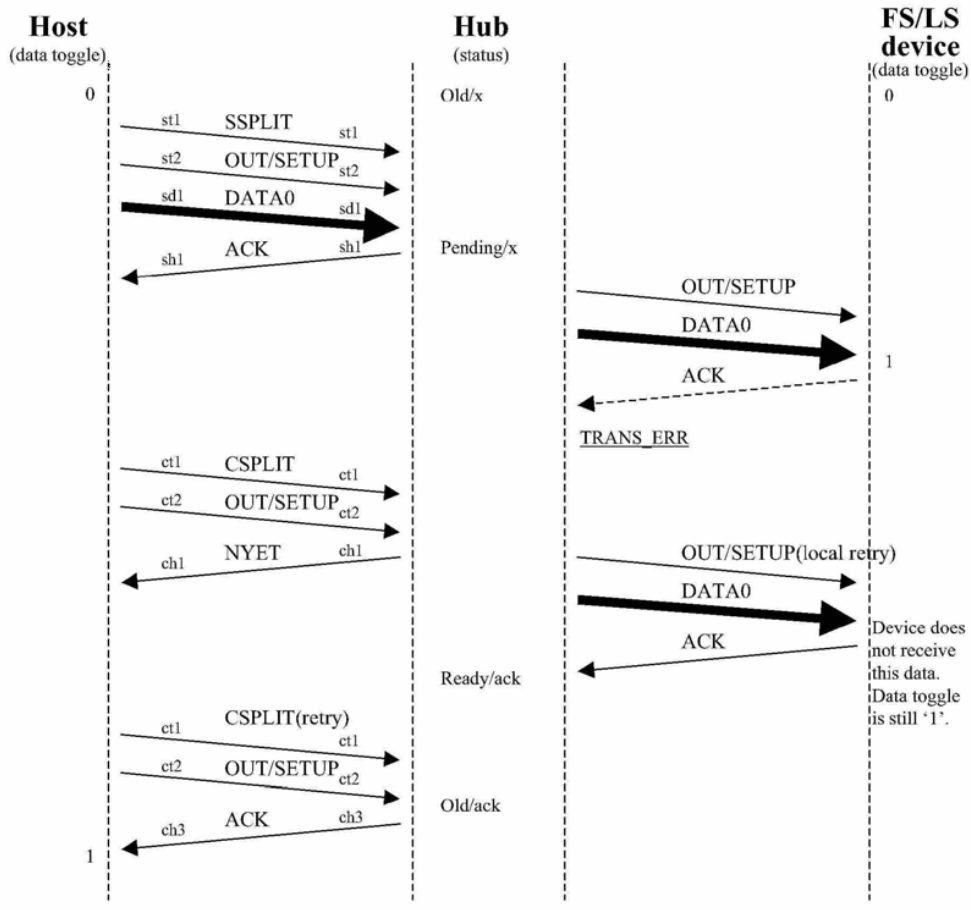


Figure A-13. Normal FS/LS ACK Smash

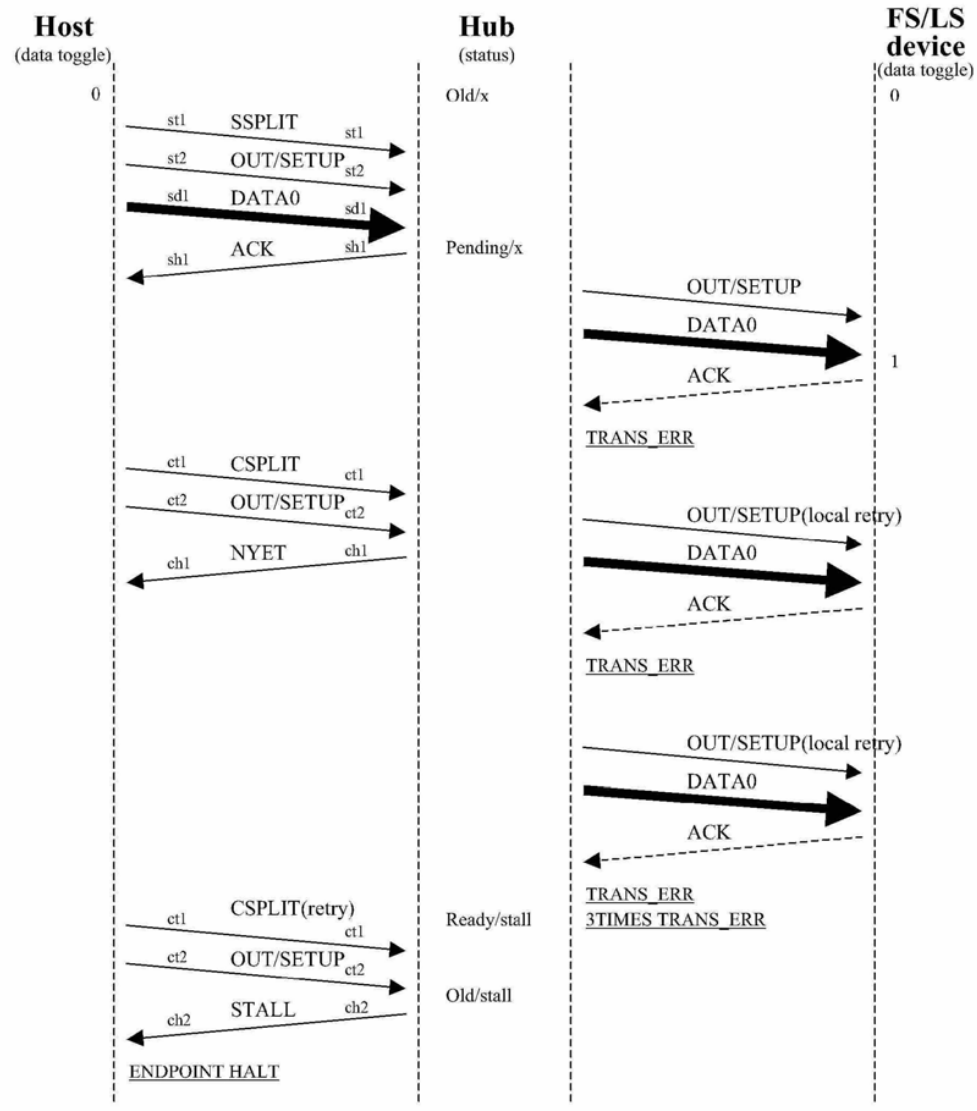


Figure A-14. Normal FS/LS ACK 3 Strikes Smash

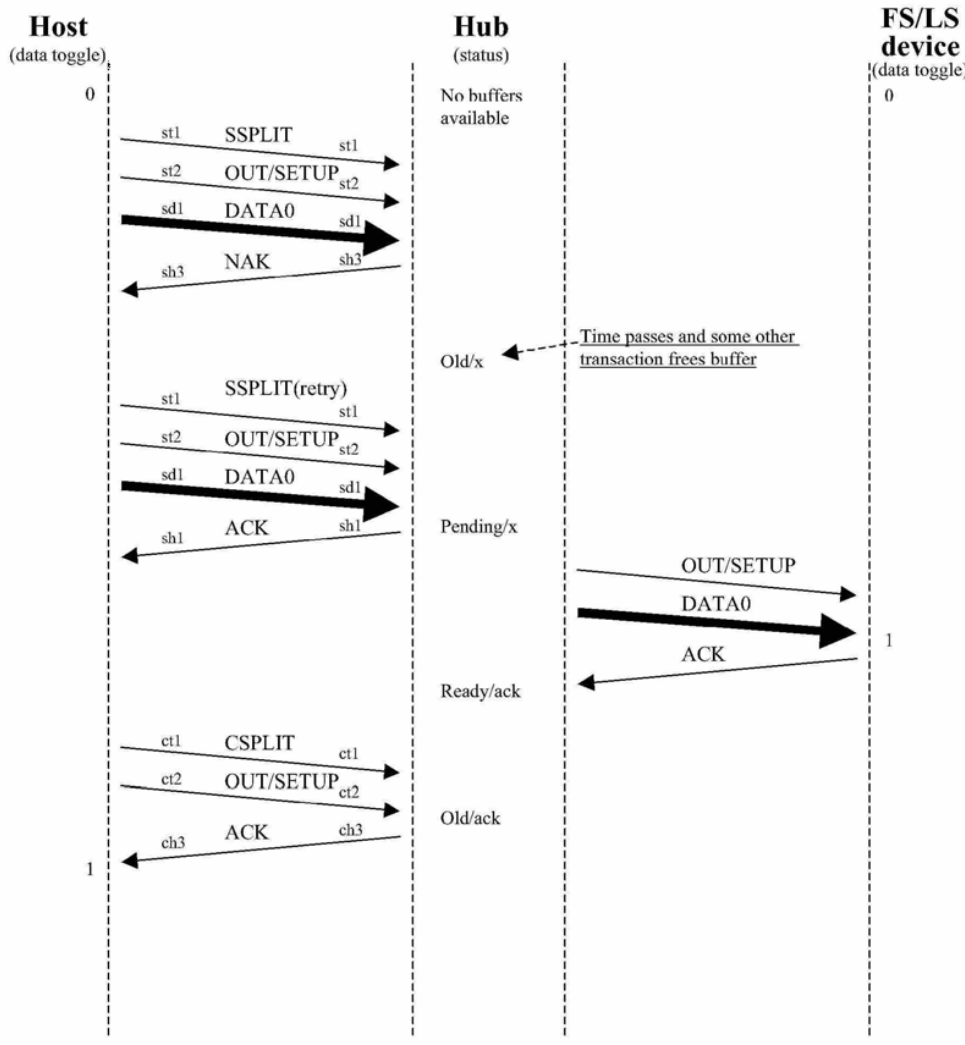


Figure A-15. No buffer Available No Smash (HS NAK(S))

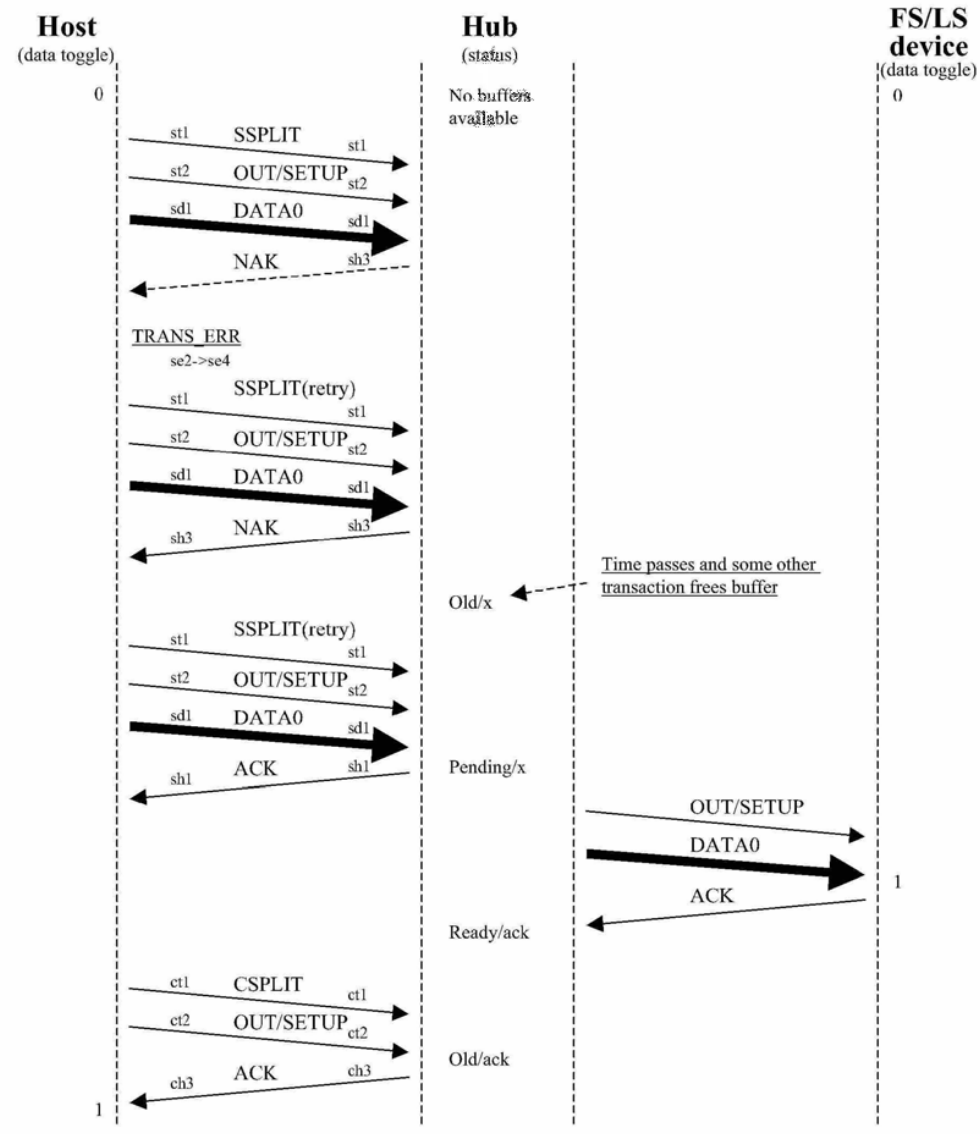


Figure A-16. No Buffer Available HS NAK(S) Smash

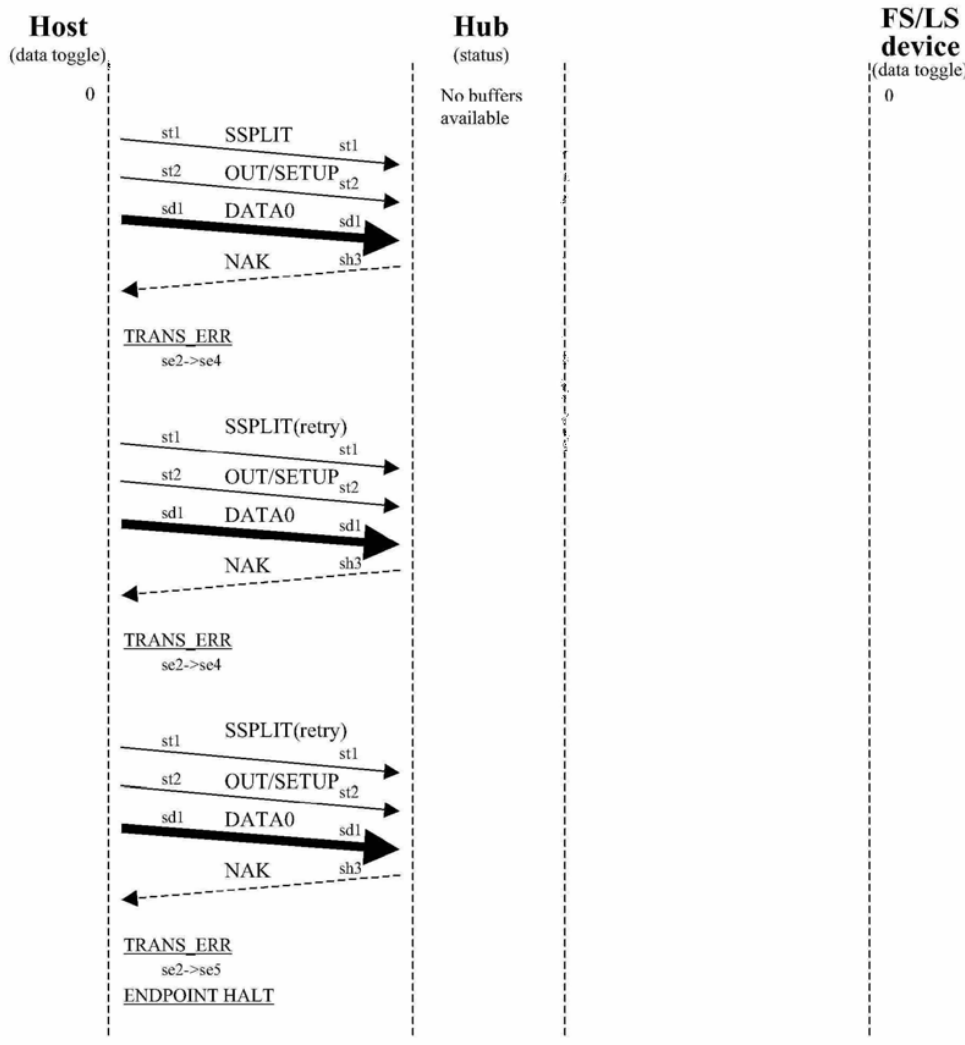


Figure A-17. No Buffer Available HS NAK(S) 3 Strikes Smash

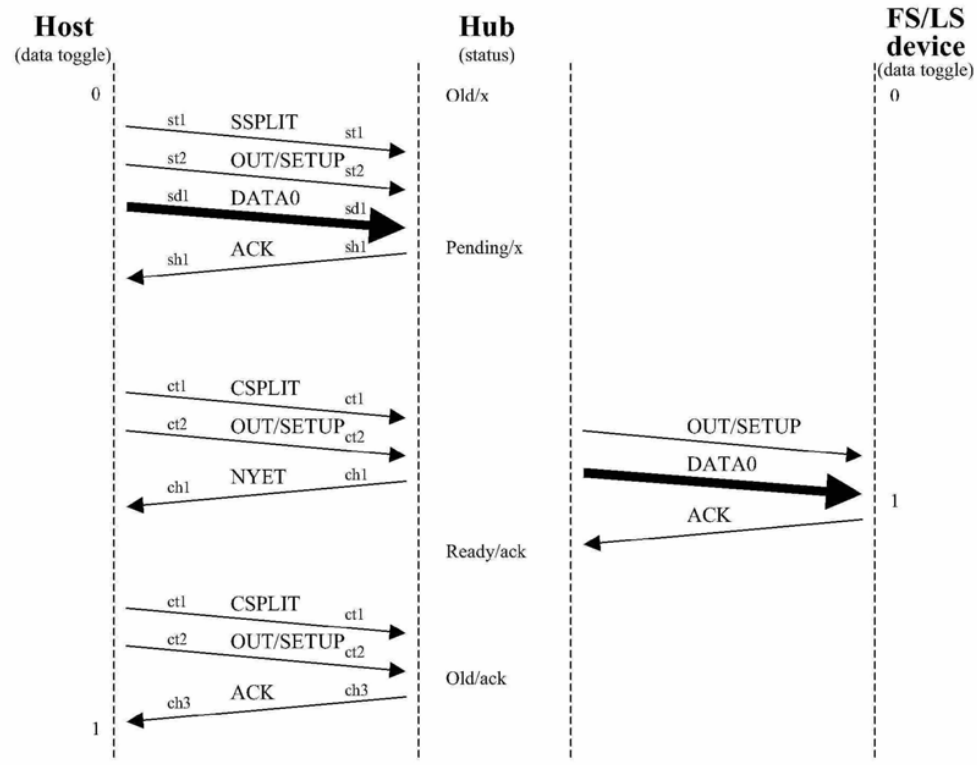


Figure A-18. CS Earlier No Smash (HS NYET)

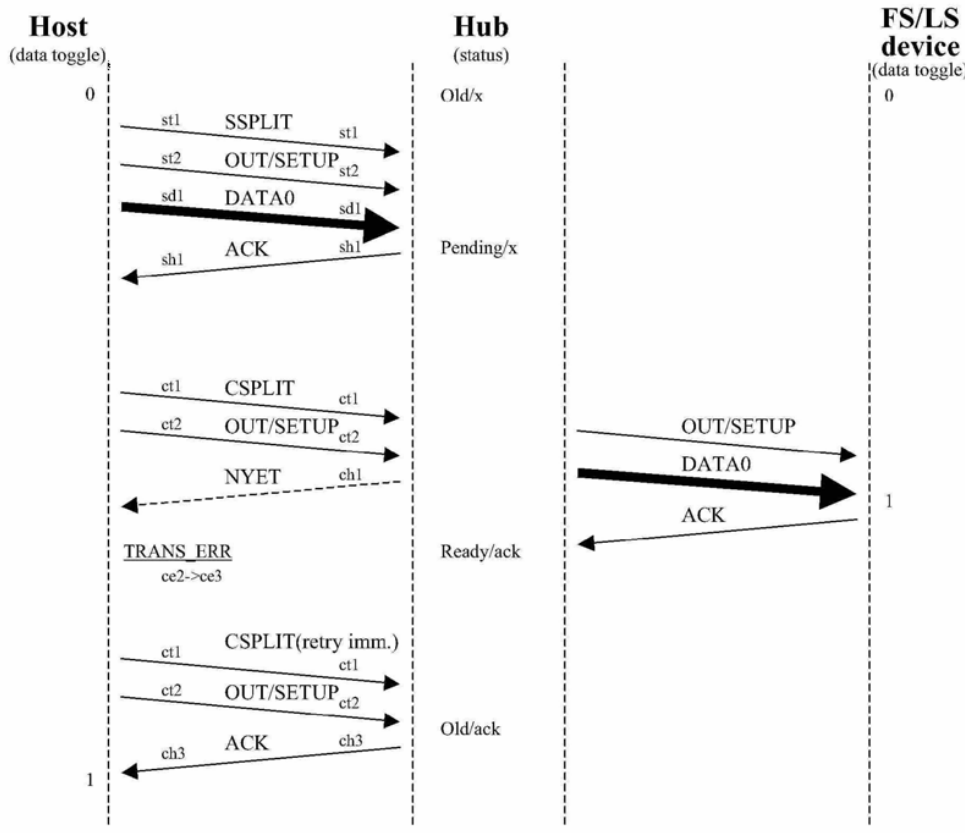


Figure A-19. CS Earlier HS NYET Smash(case 1)

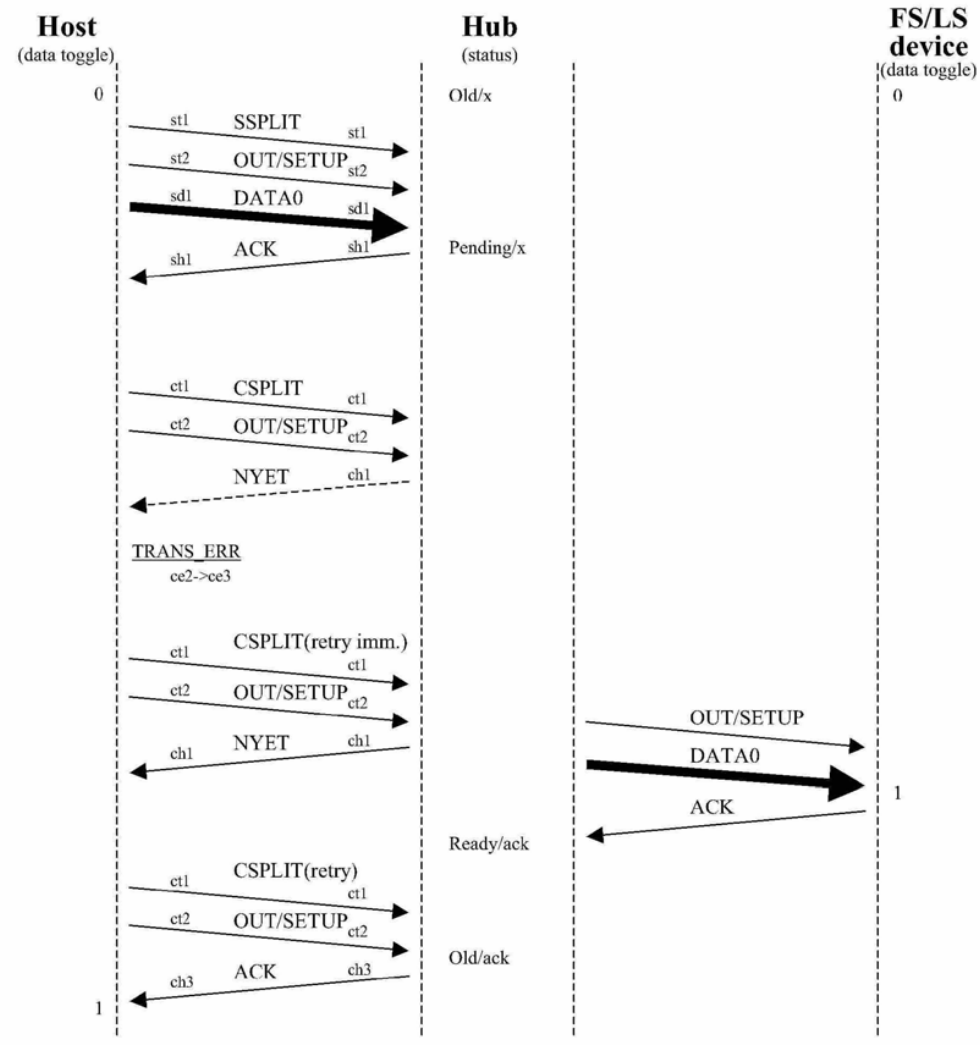


Figure A-20. CS Earlier HS NYET Smash(case 2)

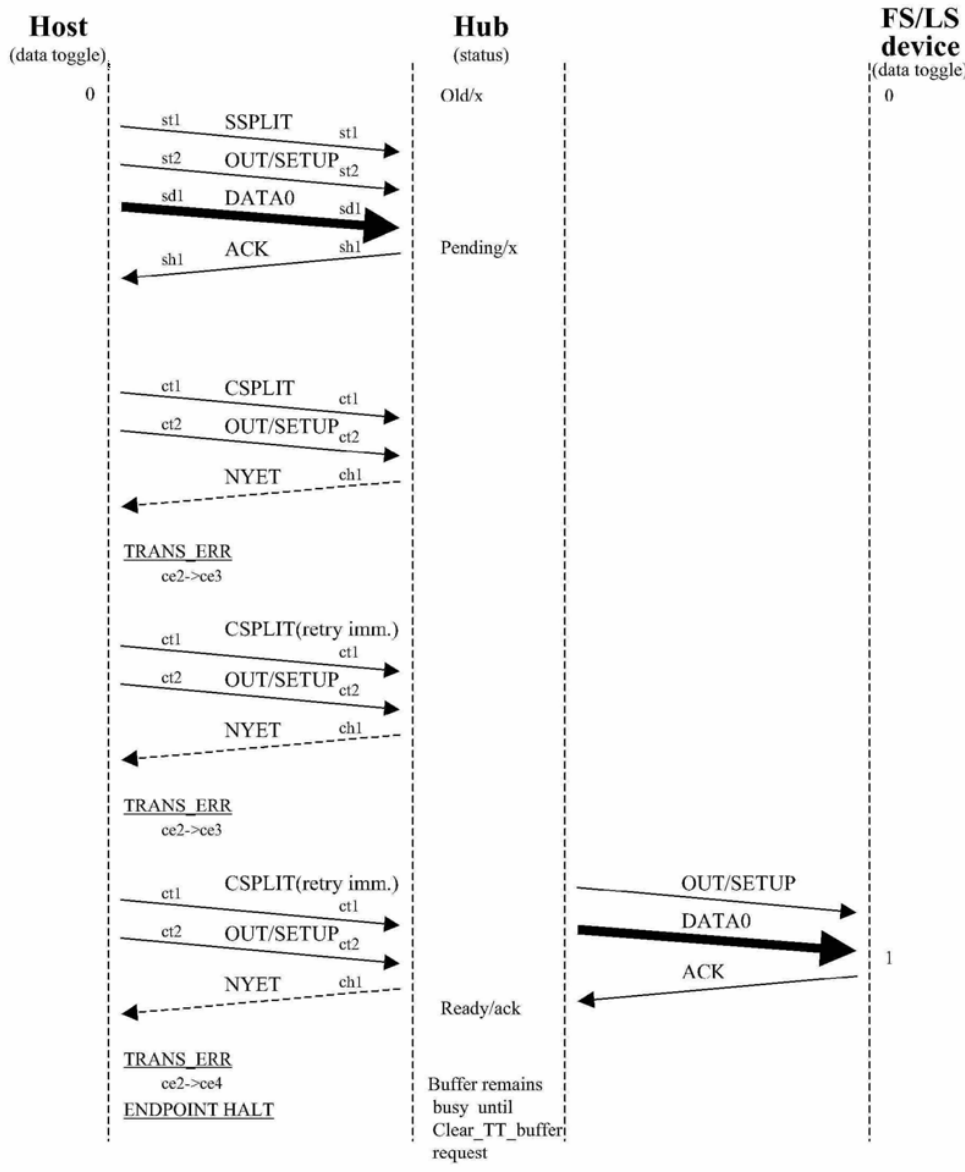


Figure A-21. CS Earlier HS NYET 3 Strikes Smash

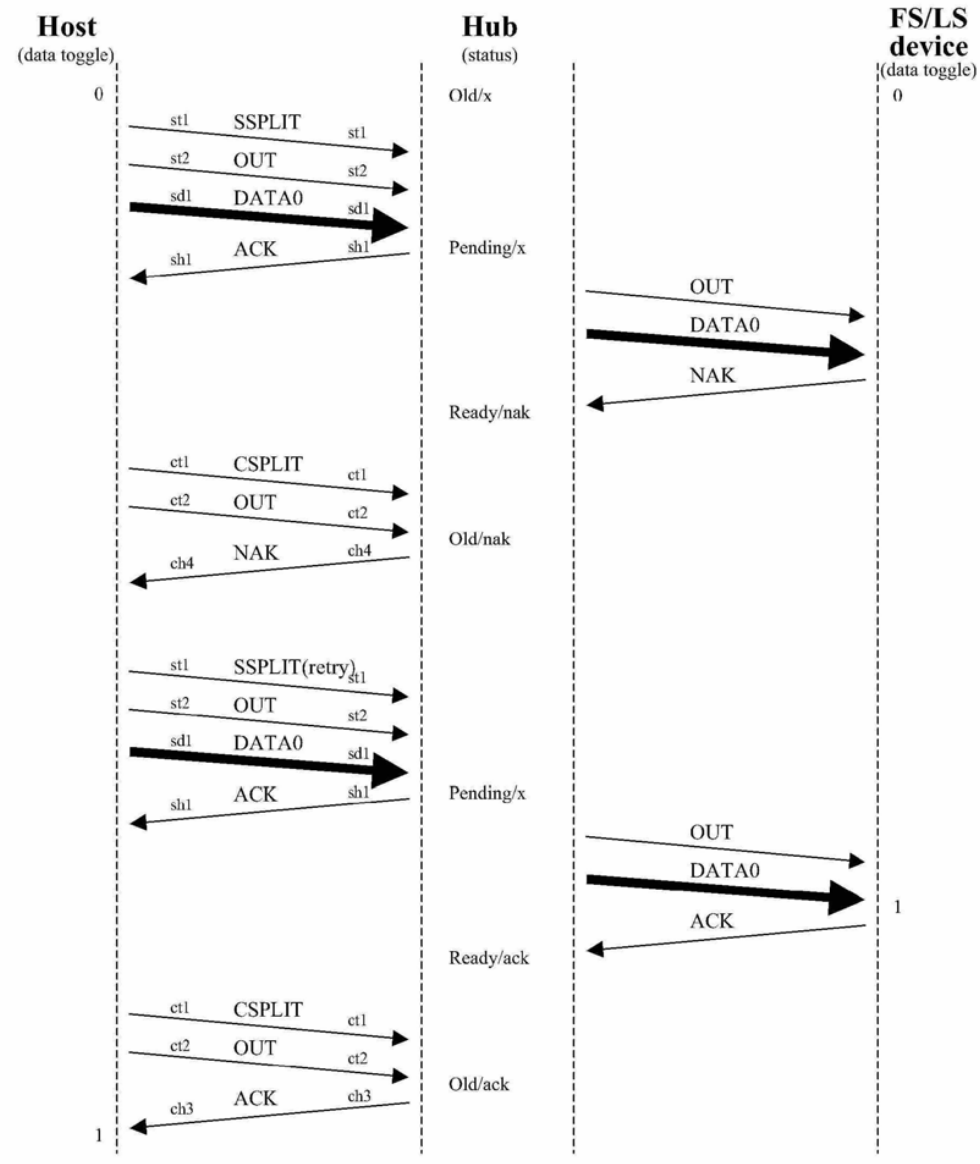


Figure A-22. Device Busy No Smash(FS/LS NAK)

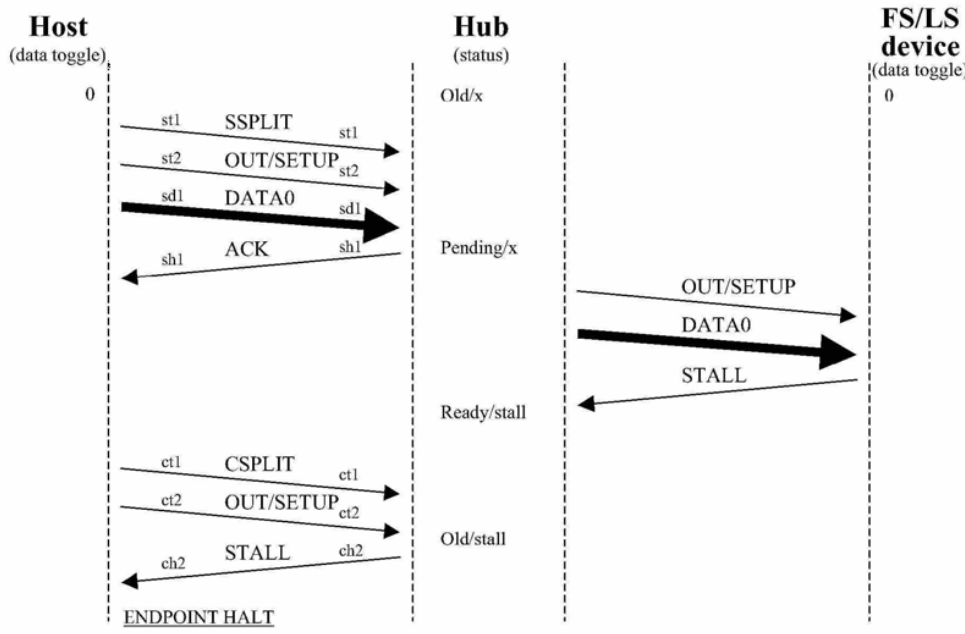


Figure A-23. Device Stall No Smash(FS/LS STALL)

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	

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

- 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

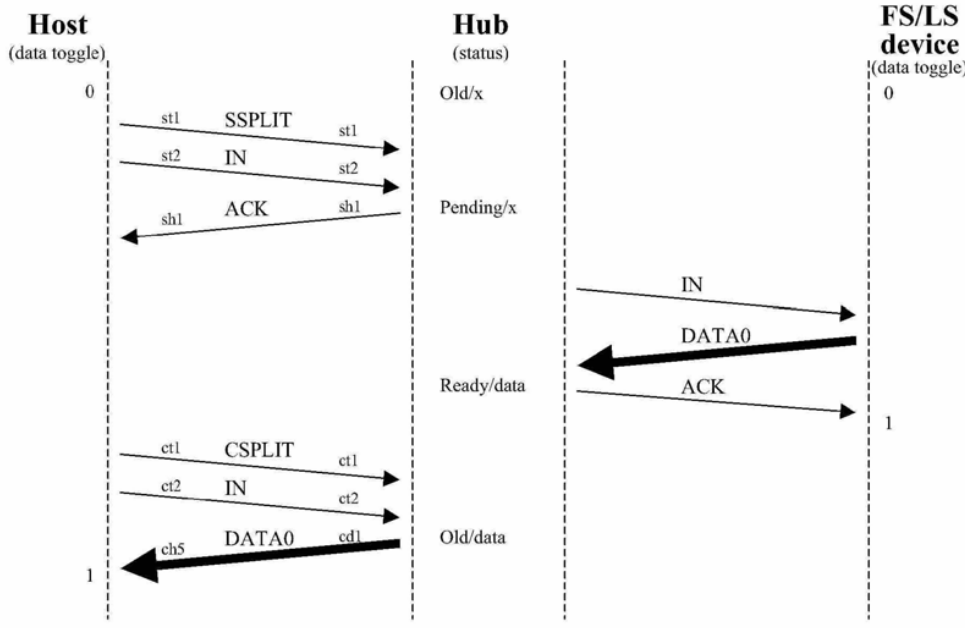


Figure A-24. Normal No Smash

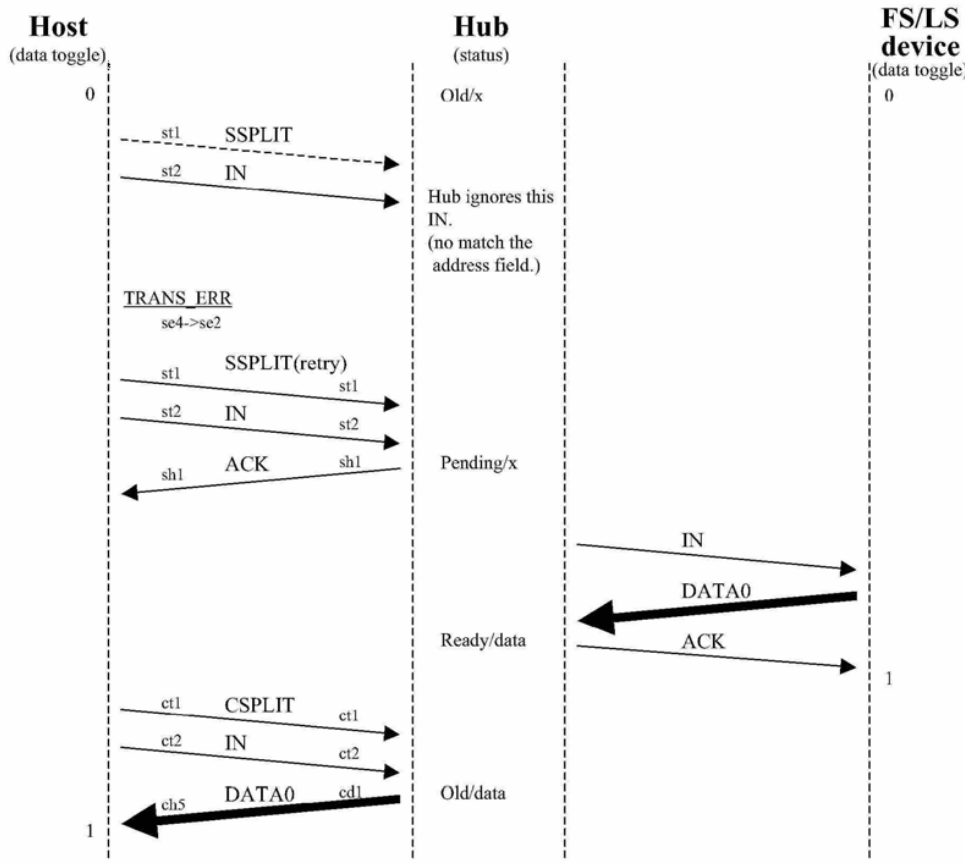


Figure A-25. Normal HS SSPLIT Smash

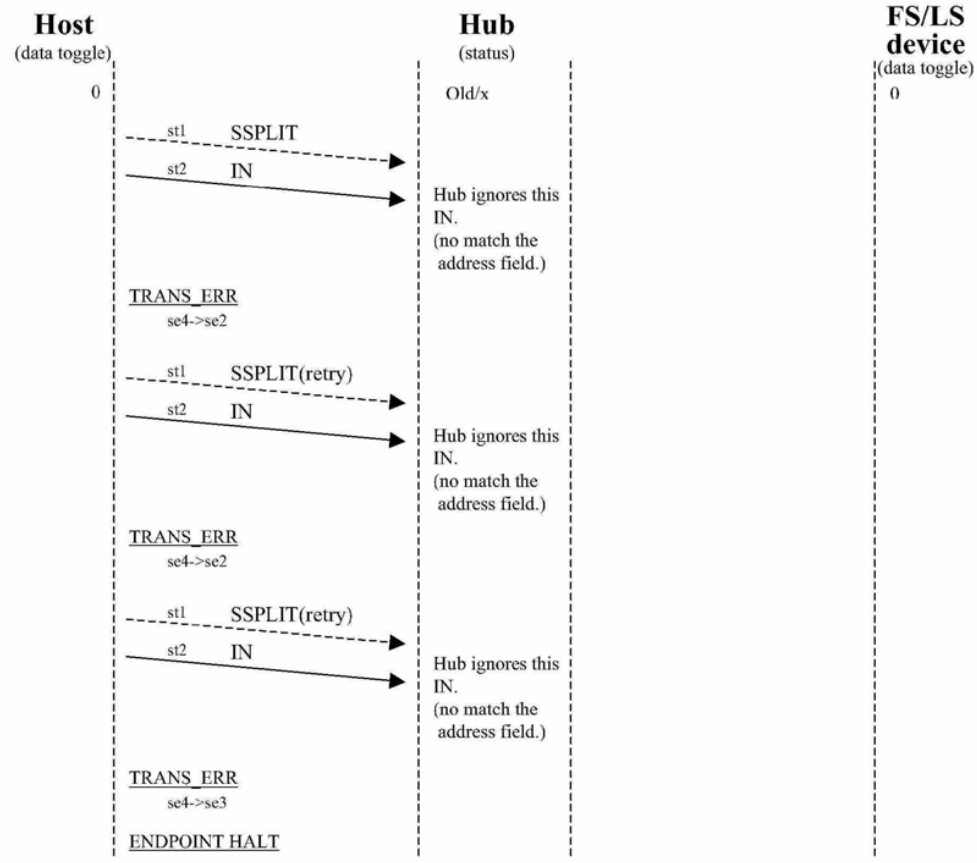


Figure A-26. Normal SSPLIT 3 Strikes Smash

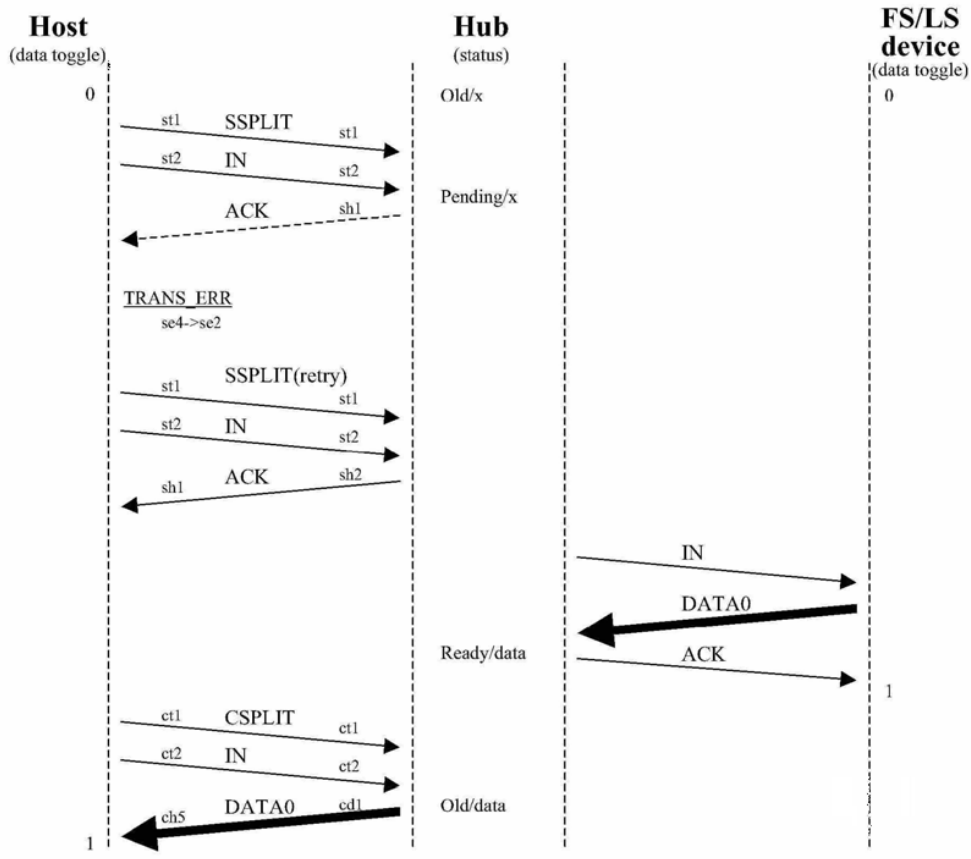


Figure A-27. Normal HS ACK(S) Smash(case 1)

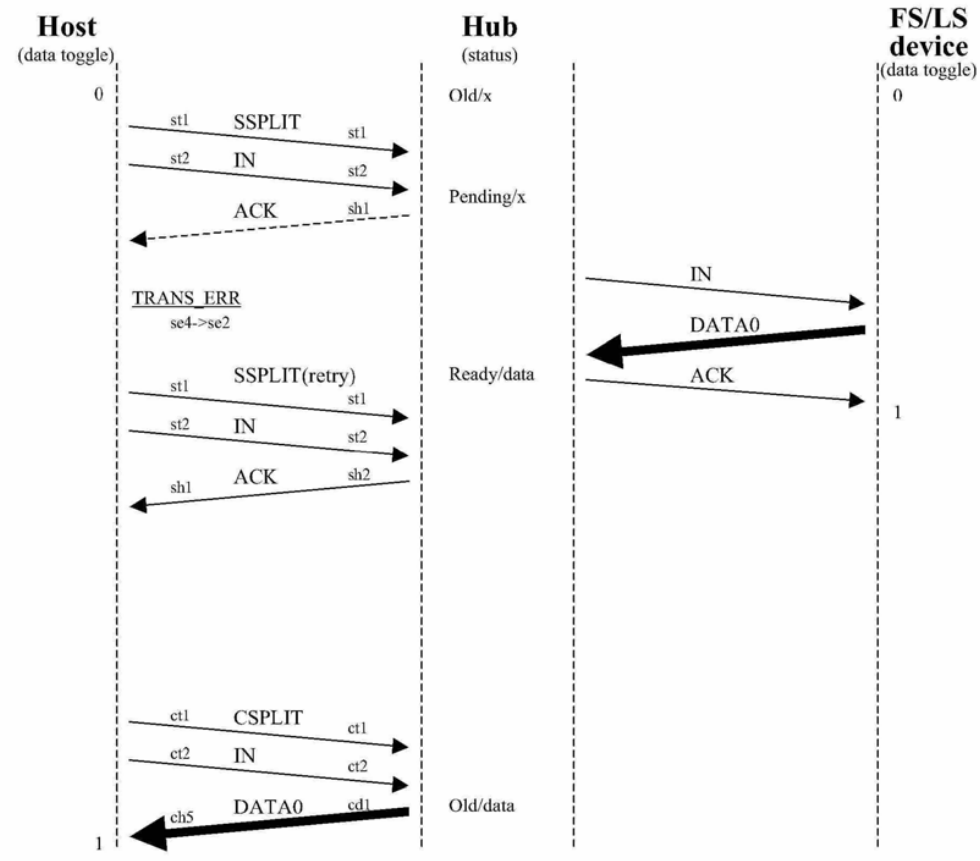


Figure A-28. Normal HS ACK(S) Smash(case 2)

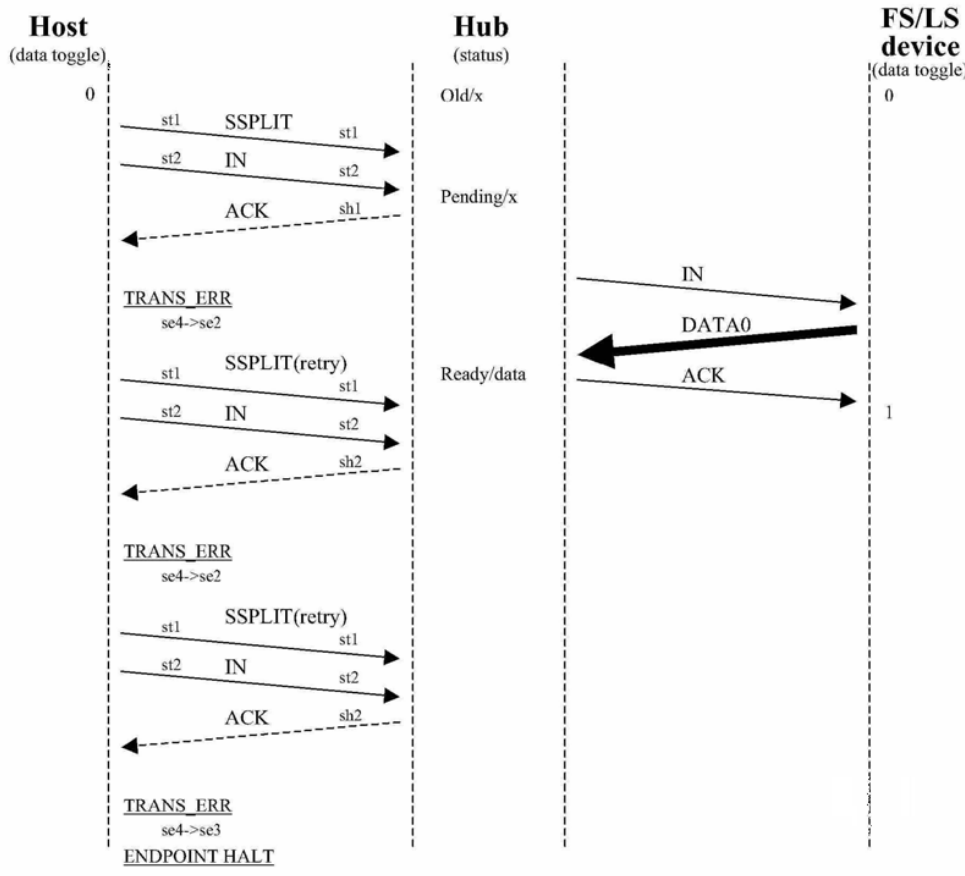


Figure A-29. Normal HS ACK(S) 3 Strikes Smash

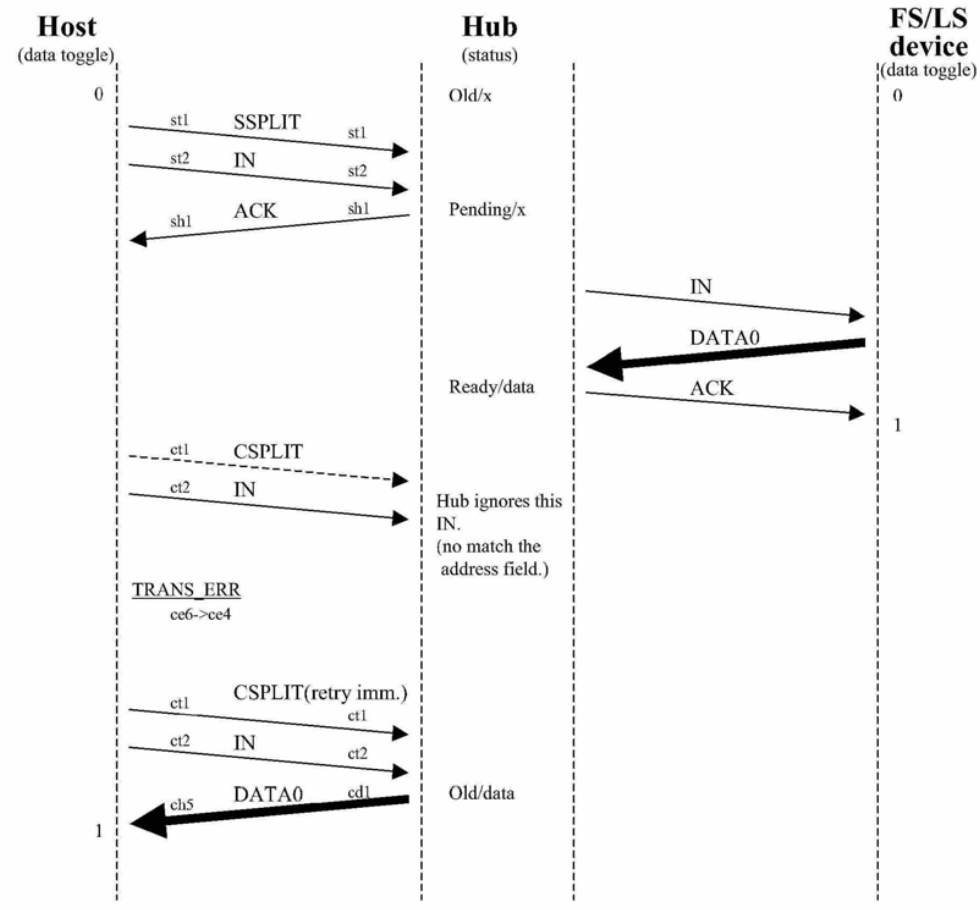


Figure A-30. Normal HS CSPLIT Smash

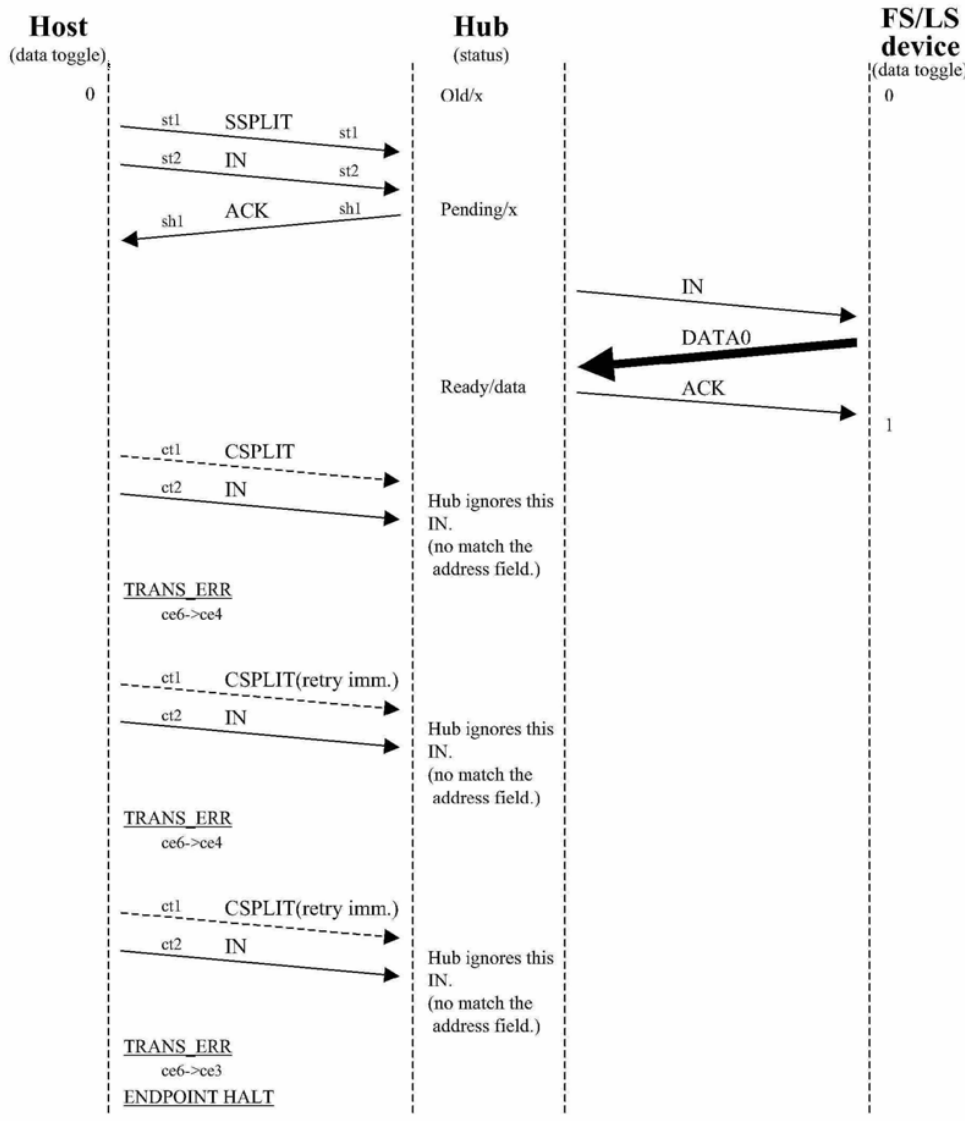


Figure A-31. Normal HS CSPLIT 3 Strikes Smash

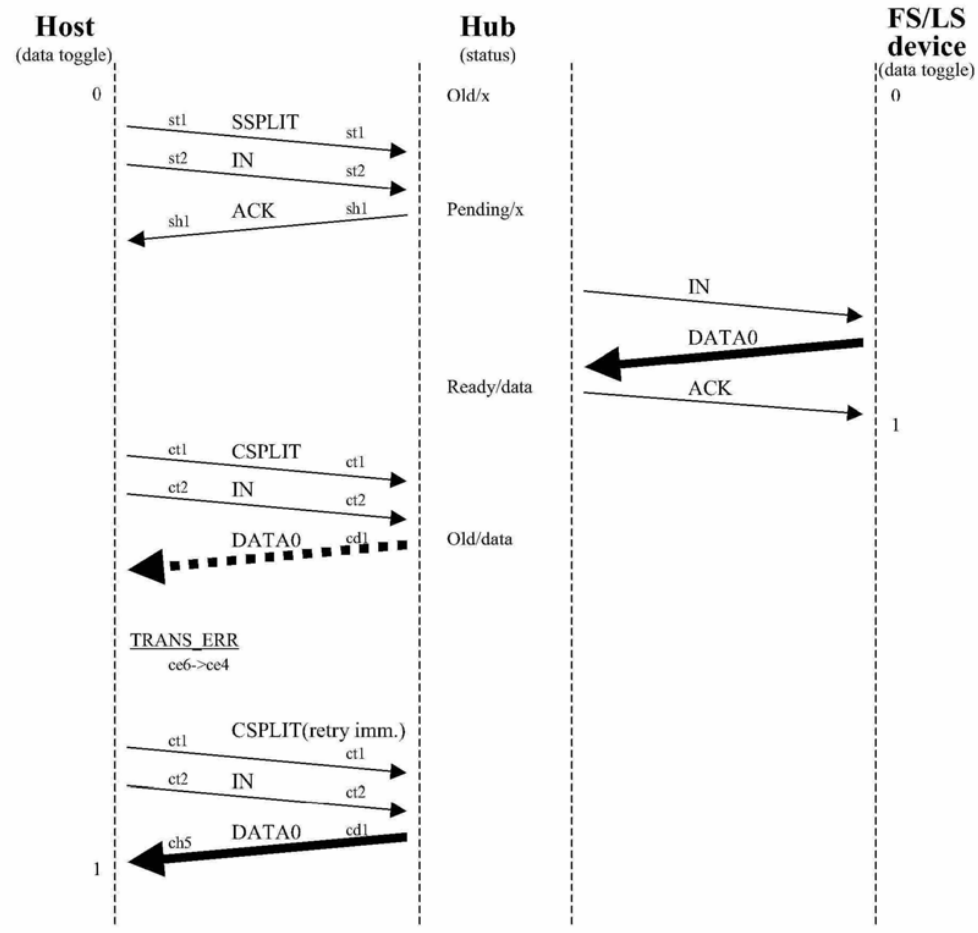


Figure A-32. Normal HS DATA0/1 Smash

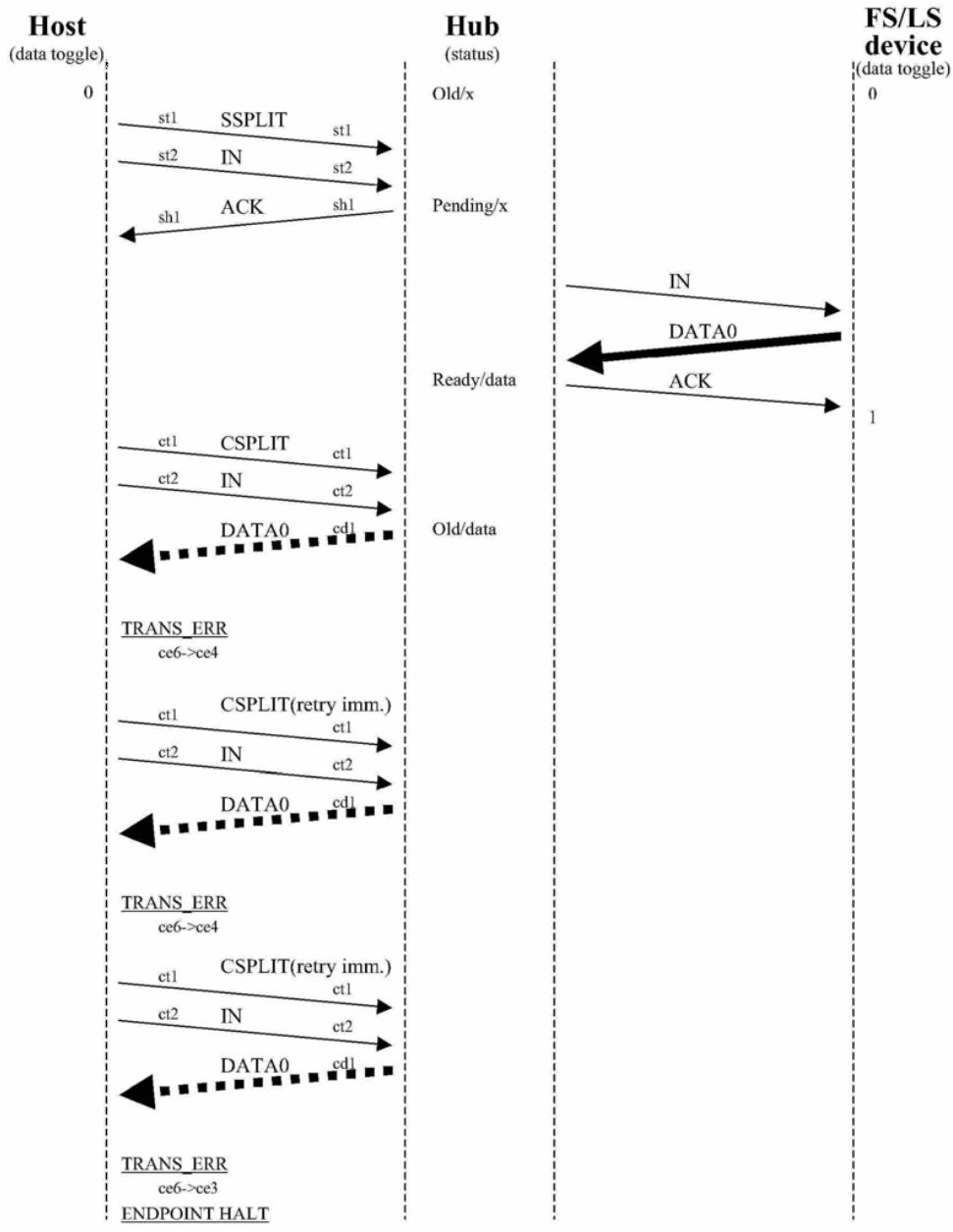


Figure A-33. Normal HS DATA0/1 3 Strikes Smash

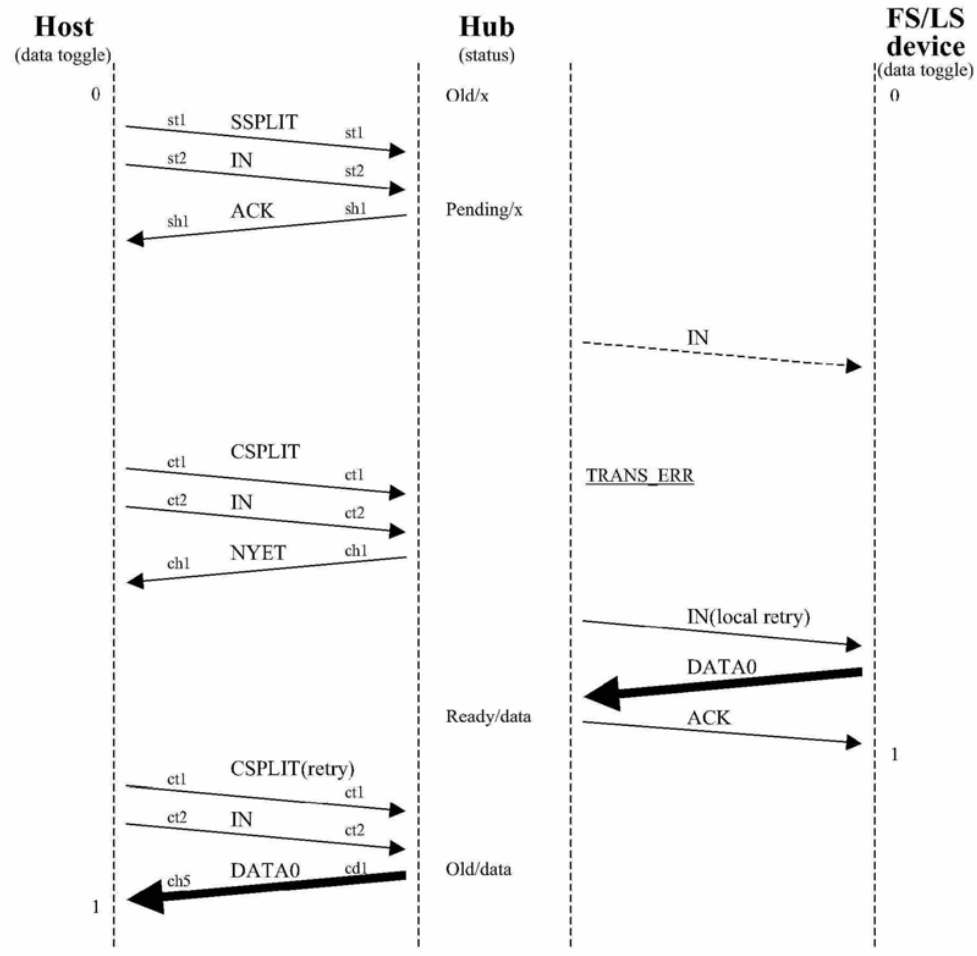


Figure A-34. Normal FS/LS IN Smash

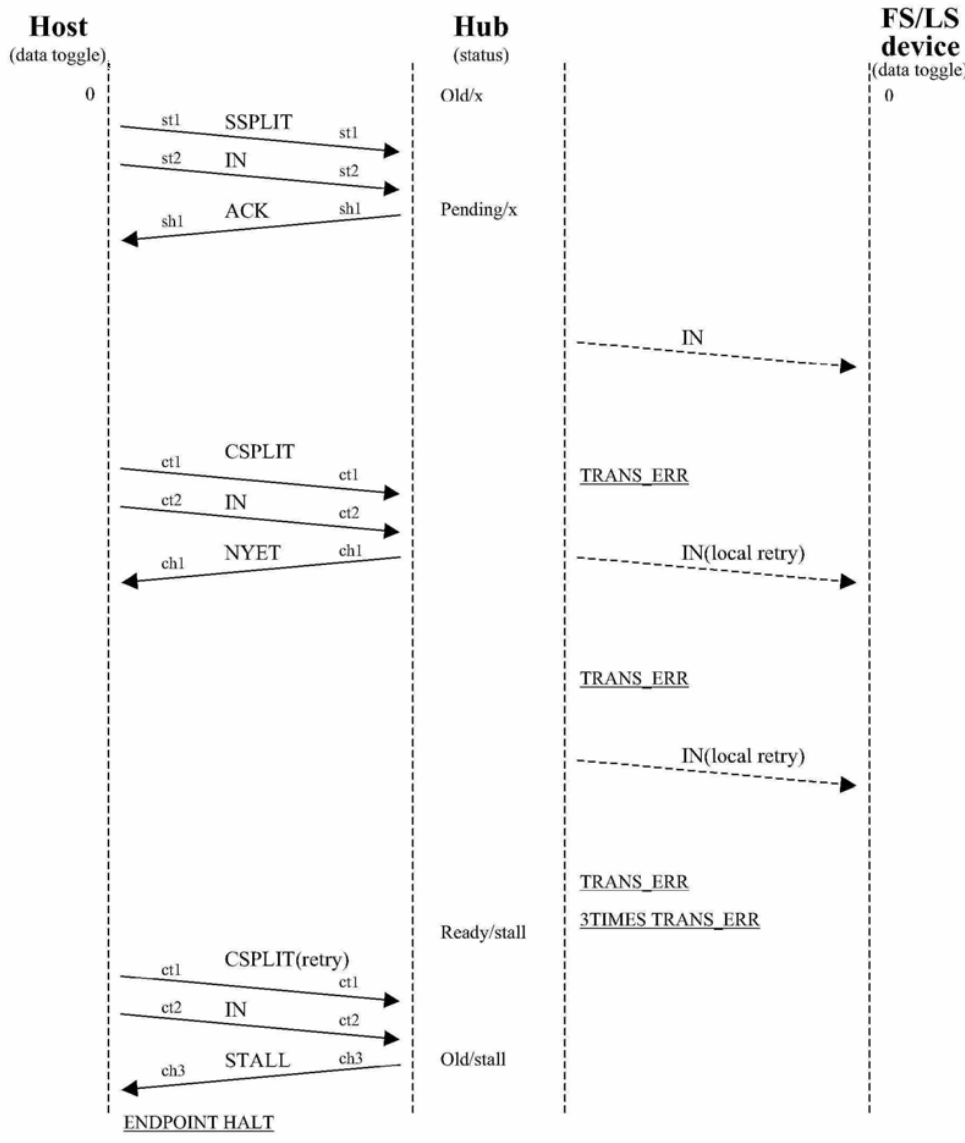


Figure A-35. Normal FS/LS IN 3 Strikes Smash

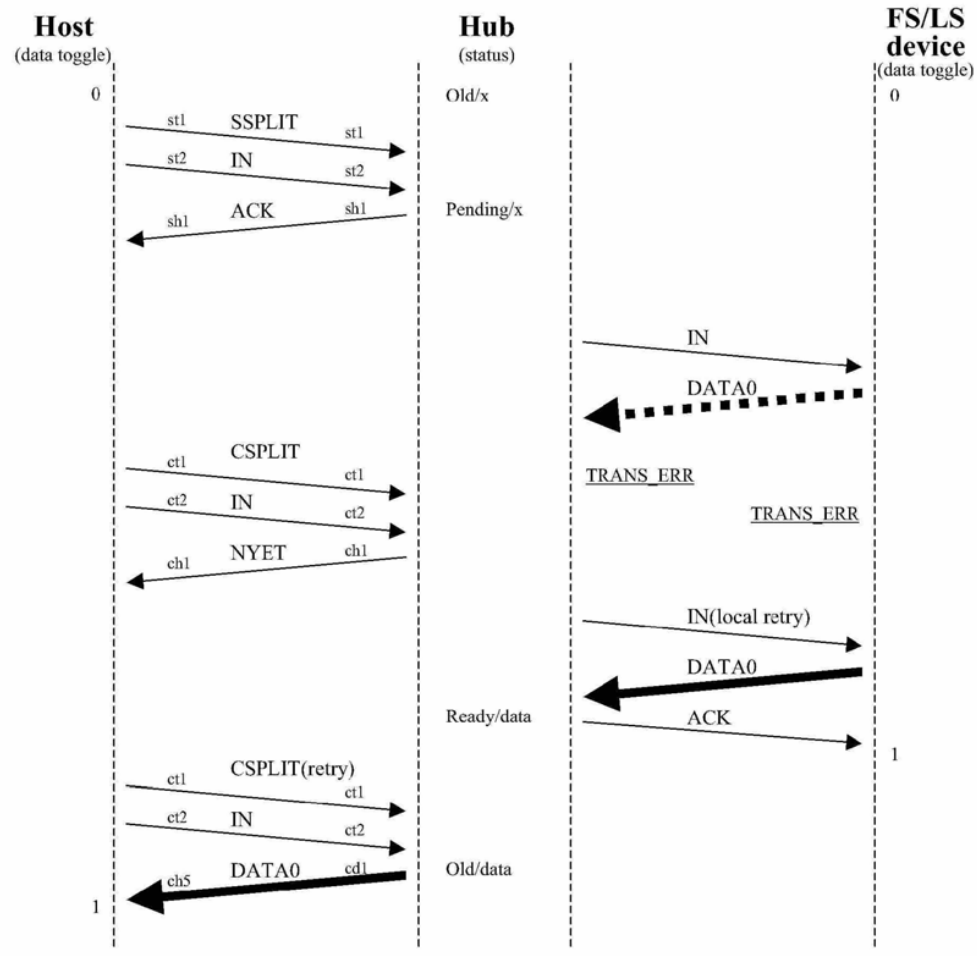


Figure A-36. Normal FS/LS DATA0/1 Smash

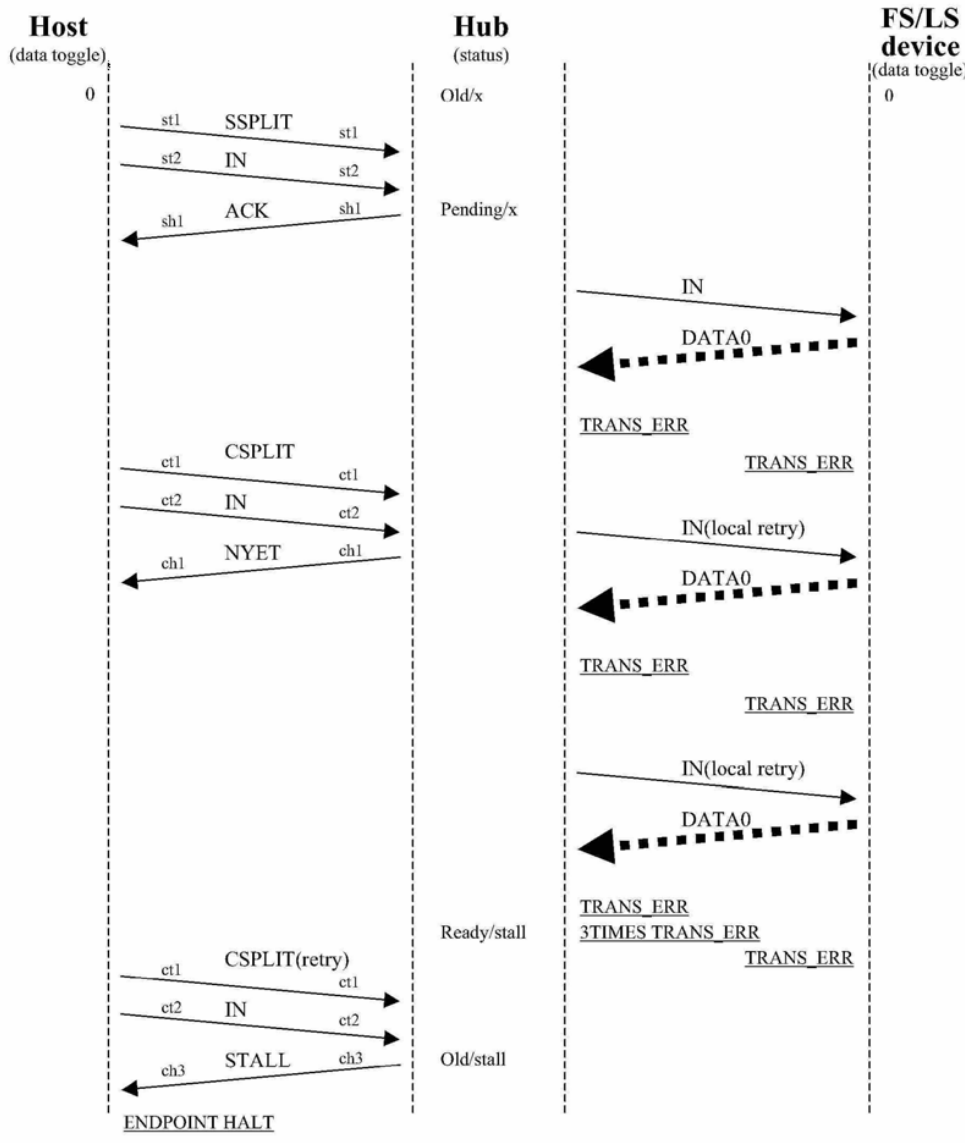


Figure A-37. Normal FS/LS DATA0/1 3 Strikes Smash

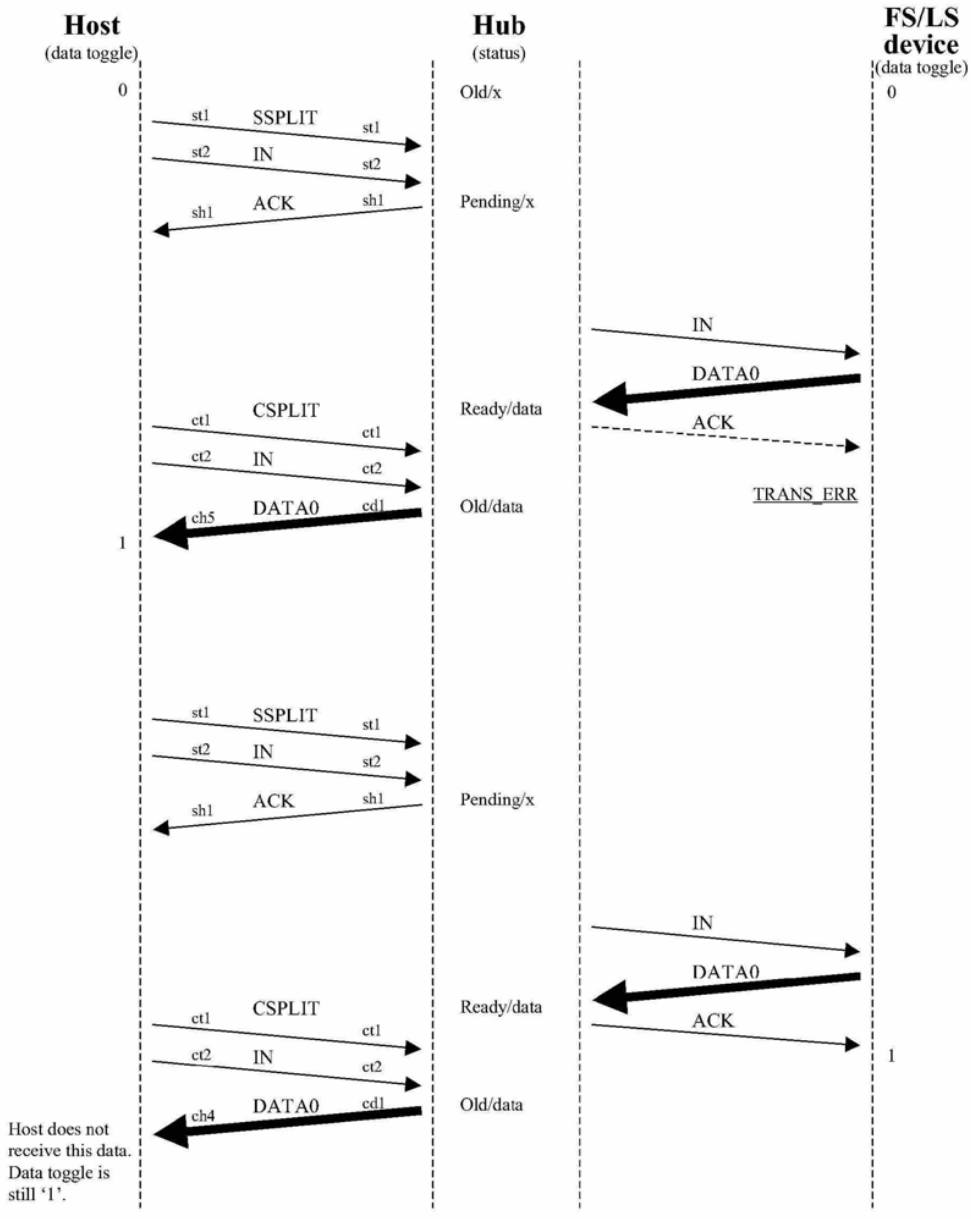


Figure A-38. Normal FS/LS ACK Smash

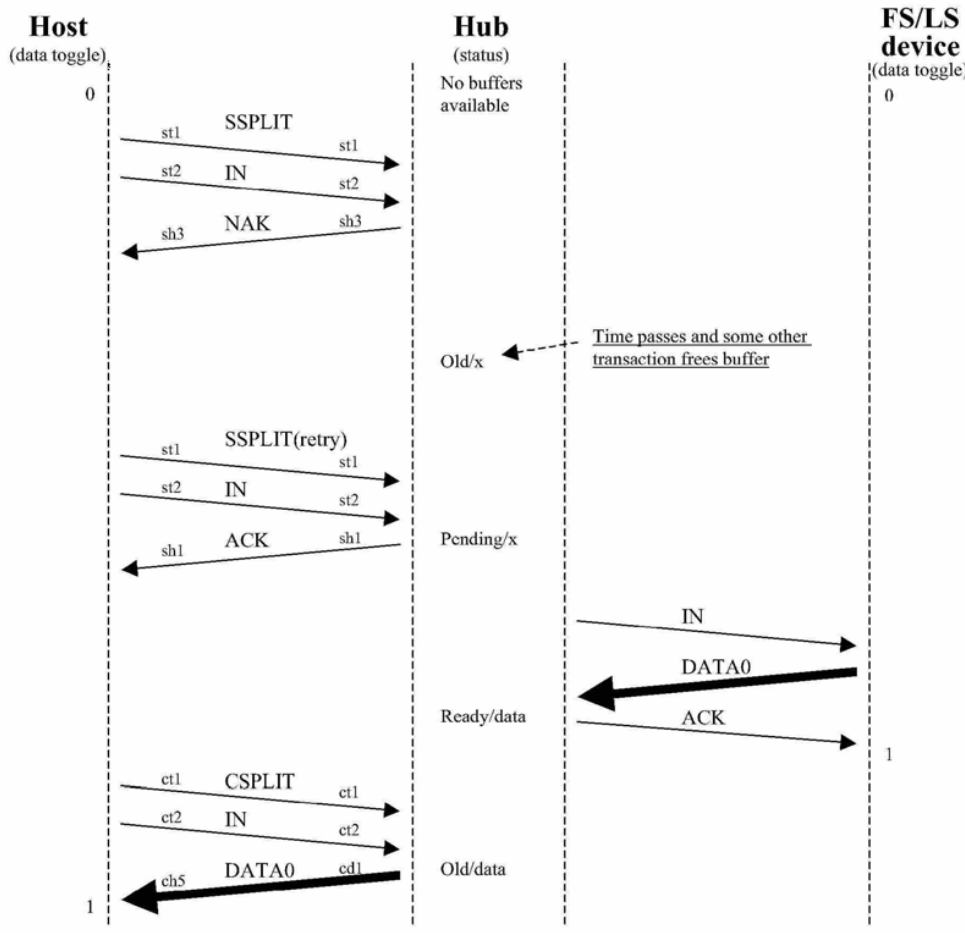


Figure A-39. No Buffer Available No Smash(HS NAK(S))

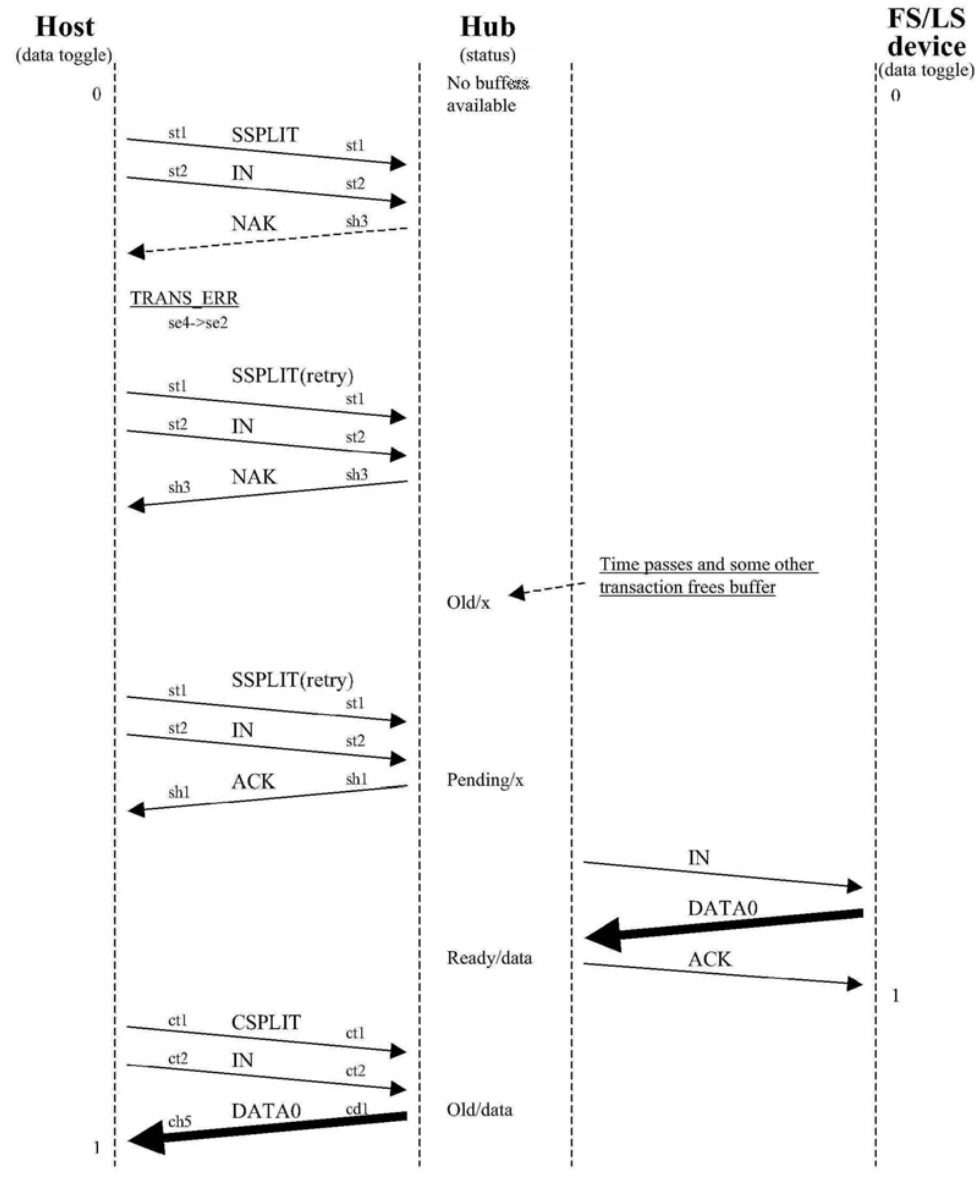


Figure A-40. No Buffer Available HS NAK(S) Smash

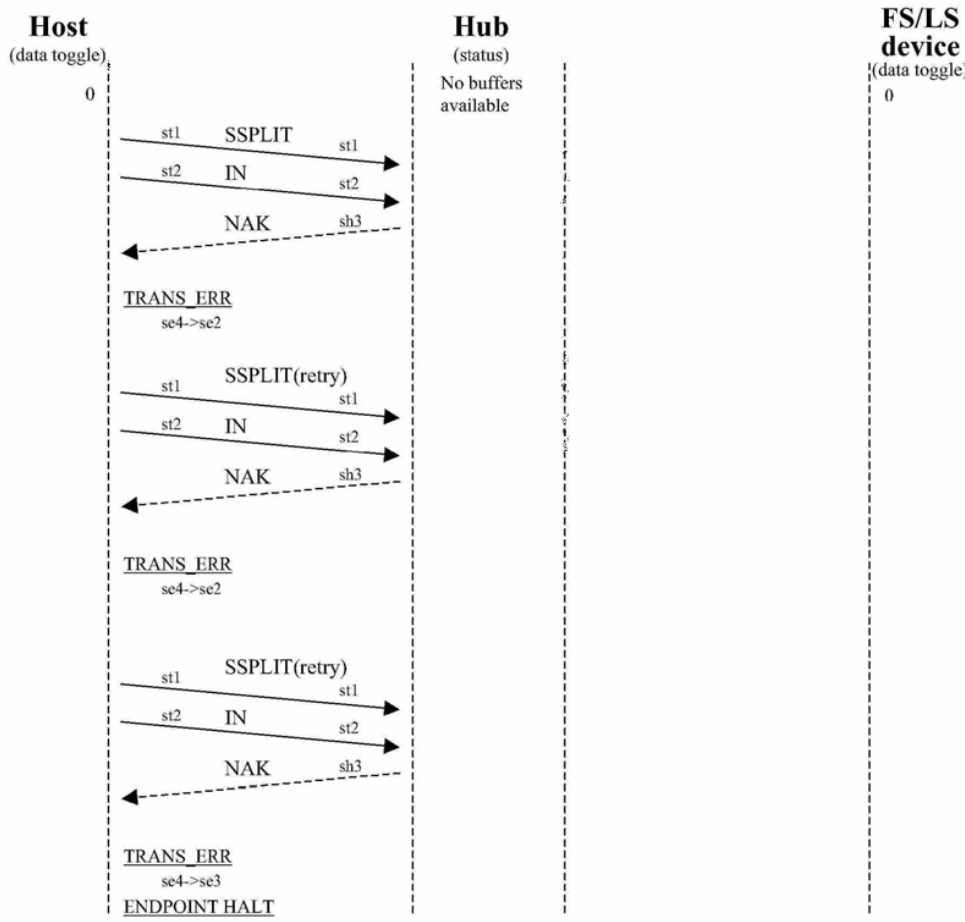


Figure A-41. No Buffer Available HS NAK(S) 3 Strikes Smash

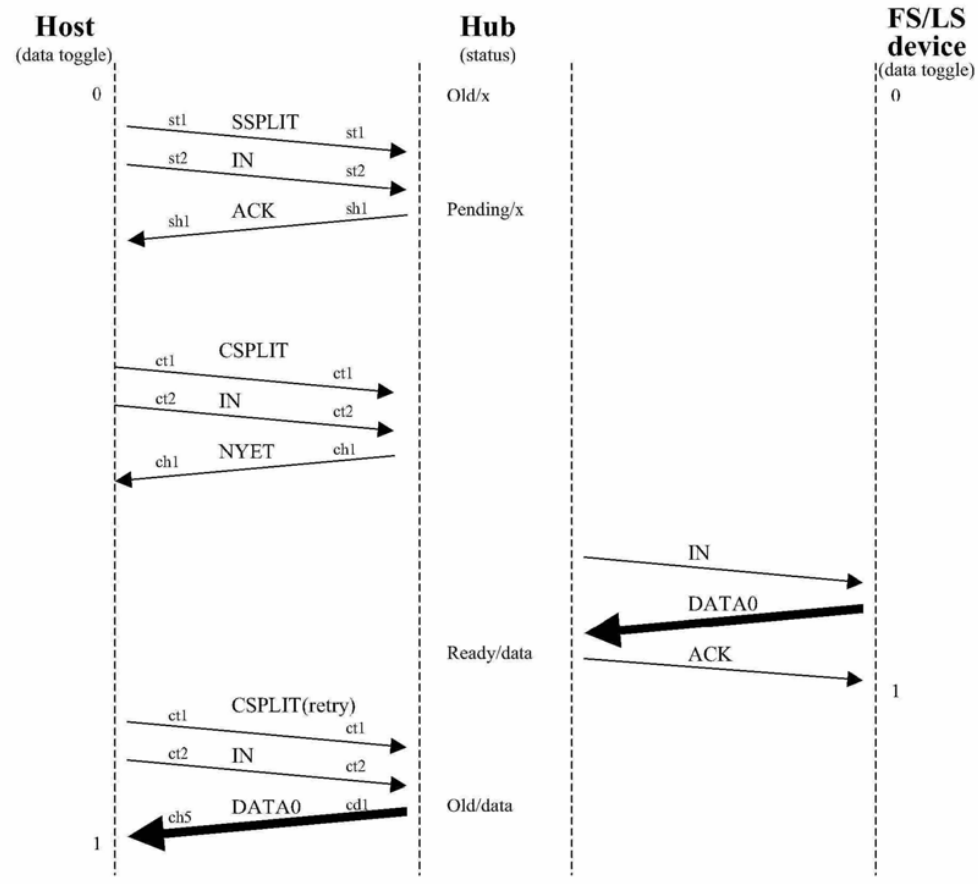


Figure A-42. CS Earlier No Smash(HS NYET)

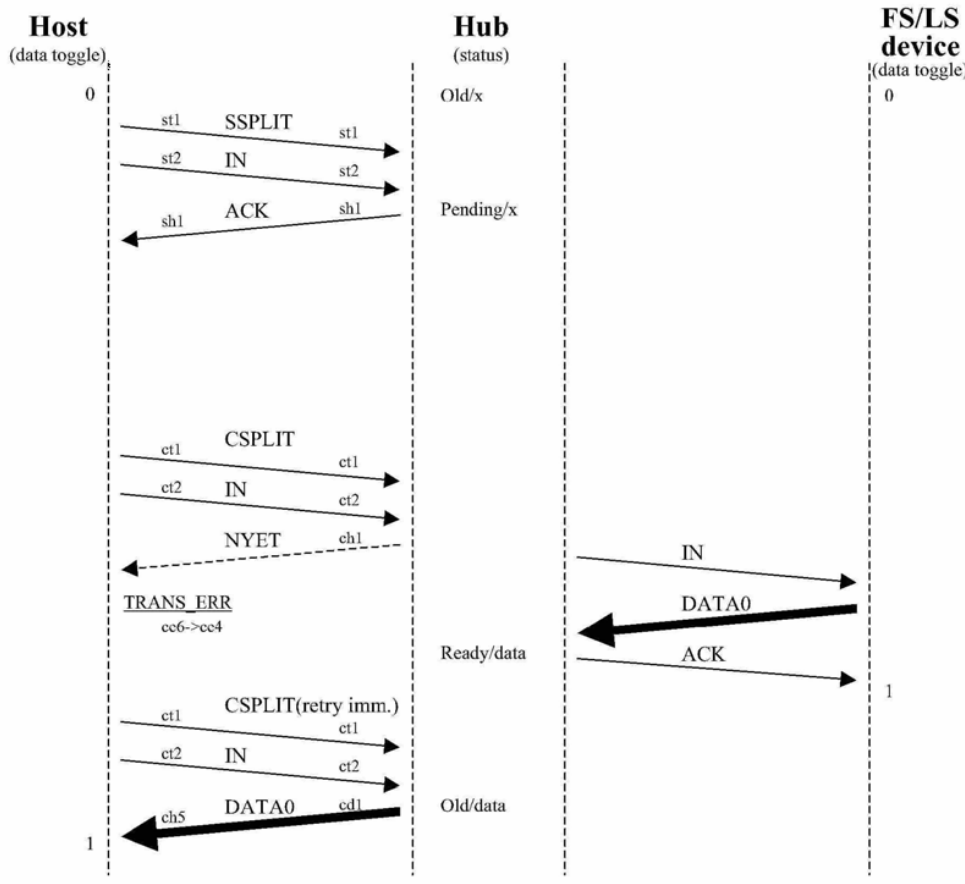


Figure A-43. CS Earlier HS NYET Smash(case 1)

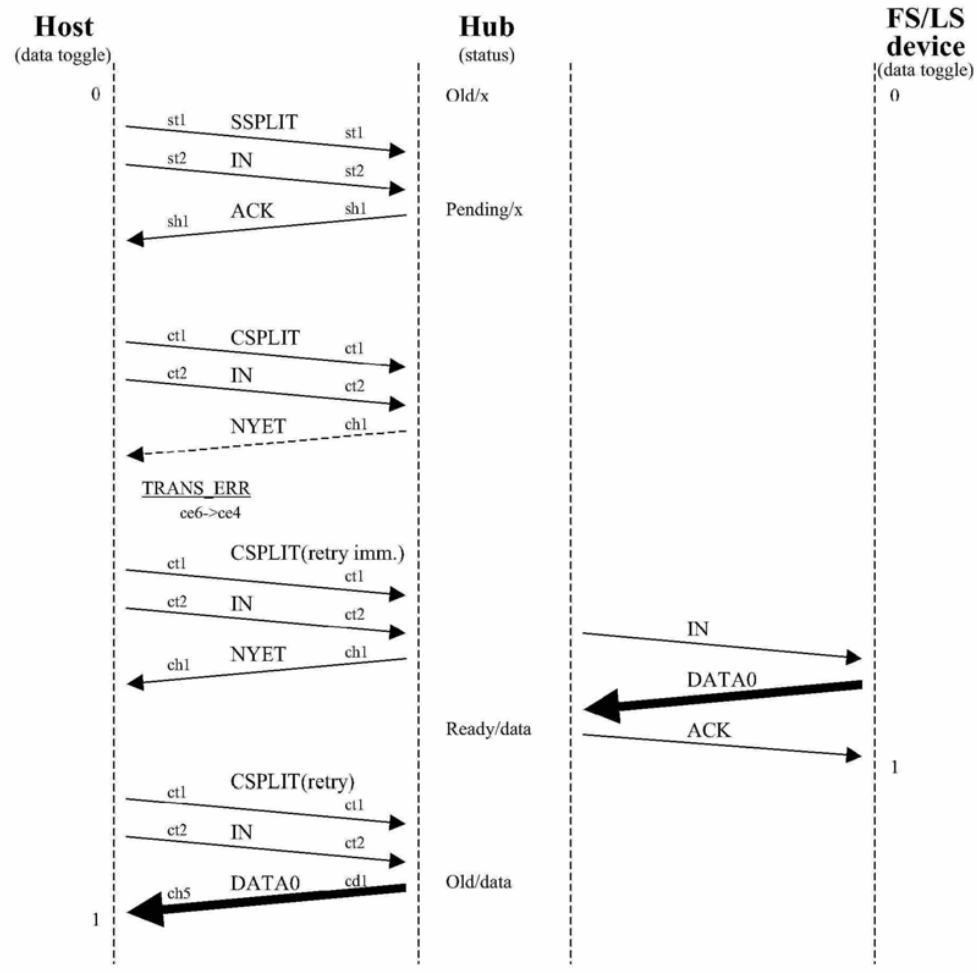


Figure A-44. CS Earlier HS NYET Smash(case 2)

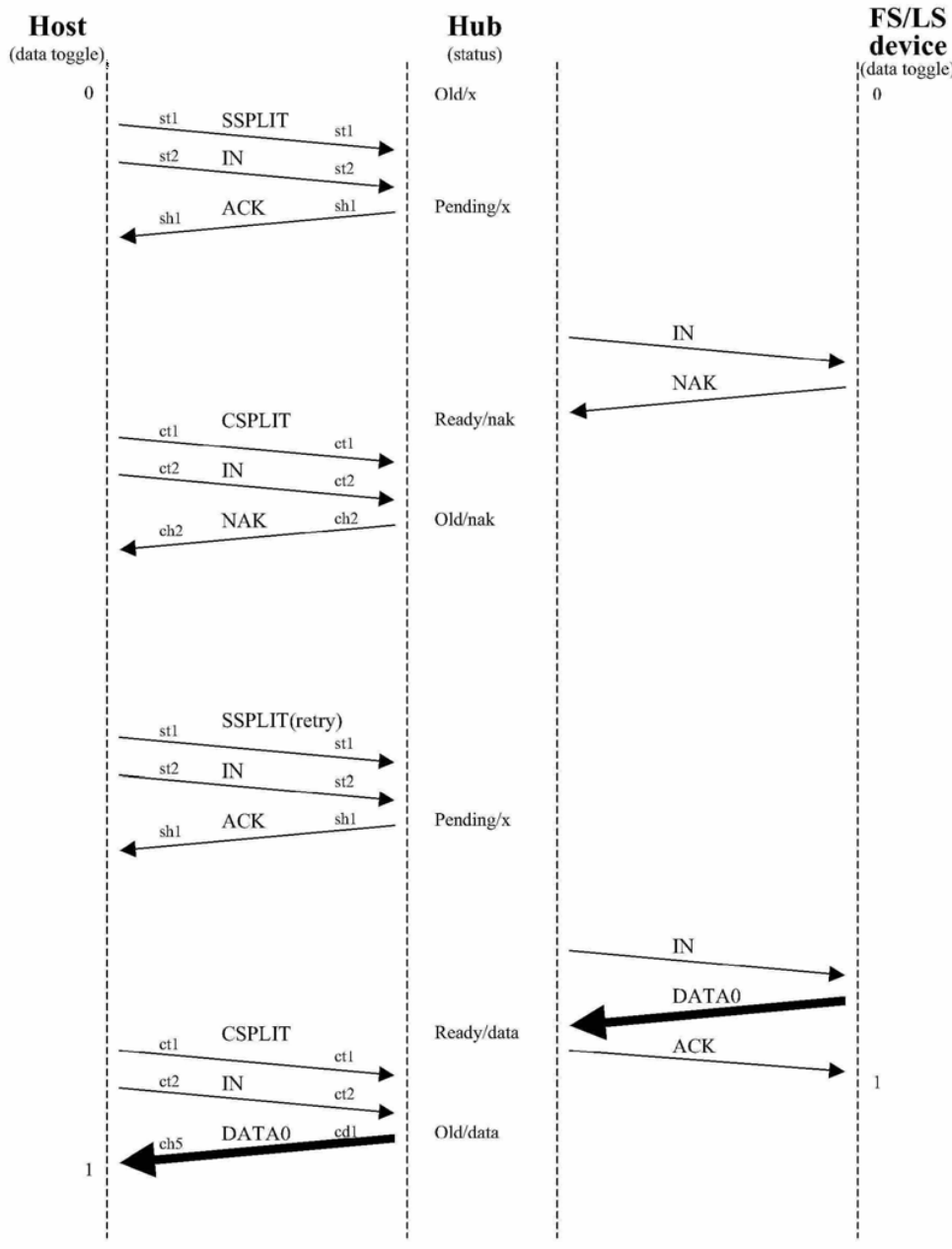


Figure A-45. Device Busy No Smash(FS/LS NAK)