

11.20.2 Interrupt Split Transaction State Machines

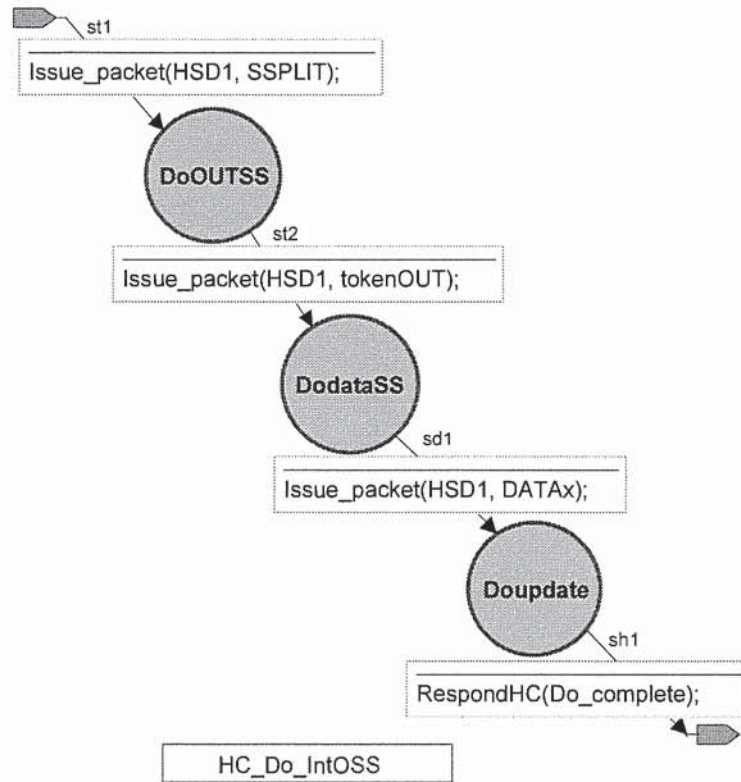


Figure 11-72. Interrupt OUT Start-split Transaction Host State Machine

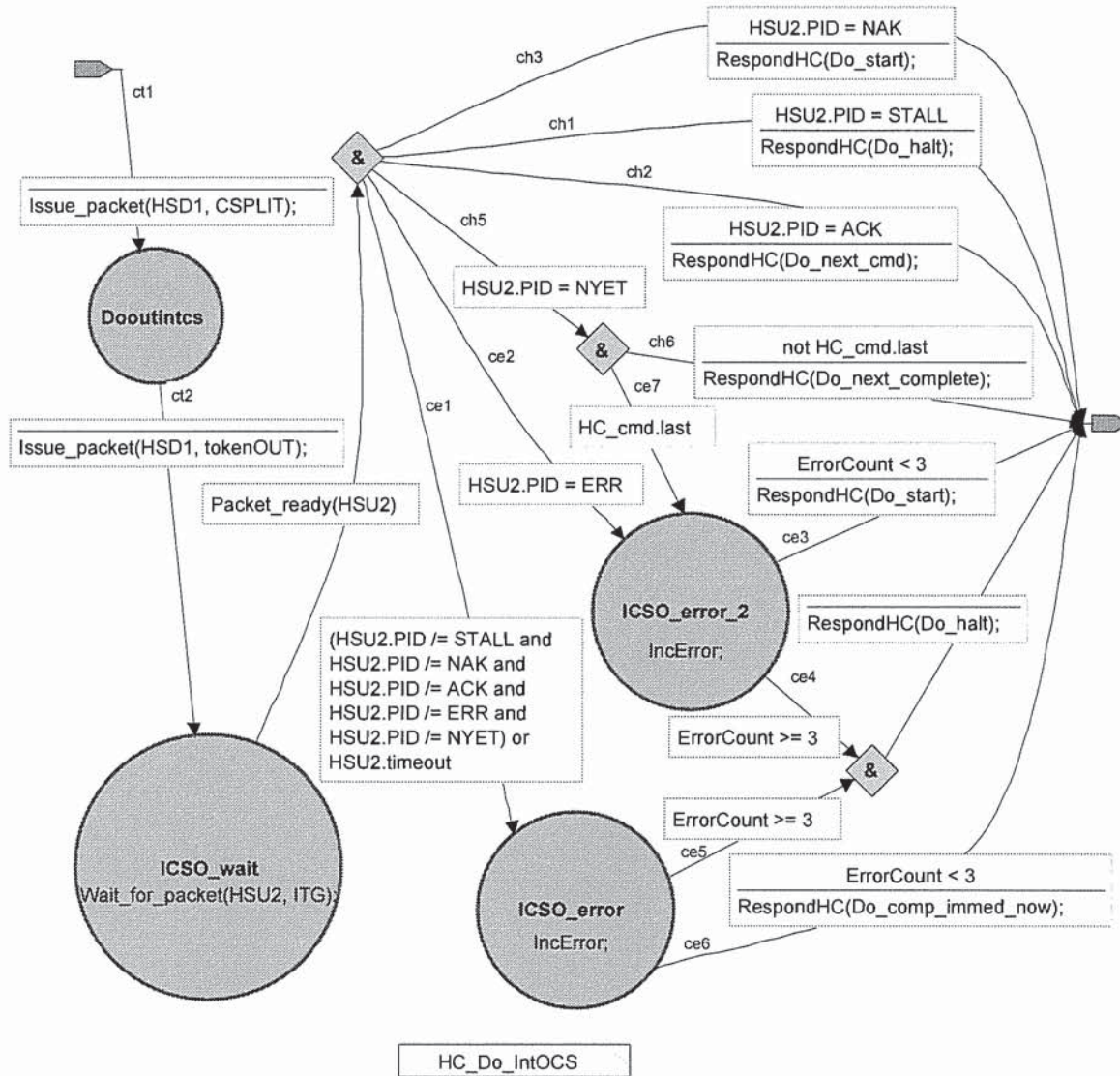


Figure 11-73. Interrupt OUT Complete-split Transaction Host State Machine

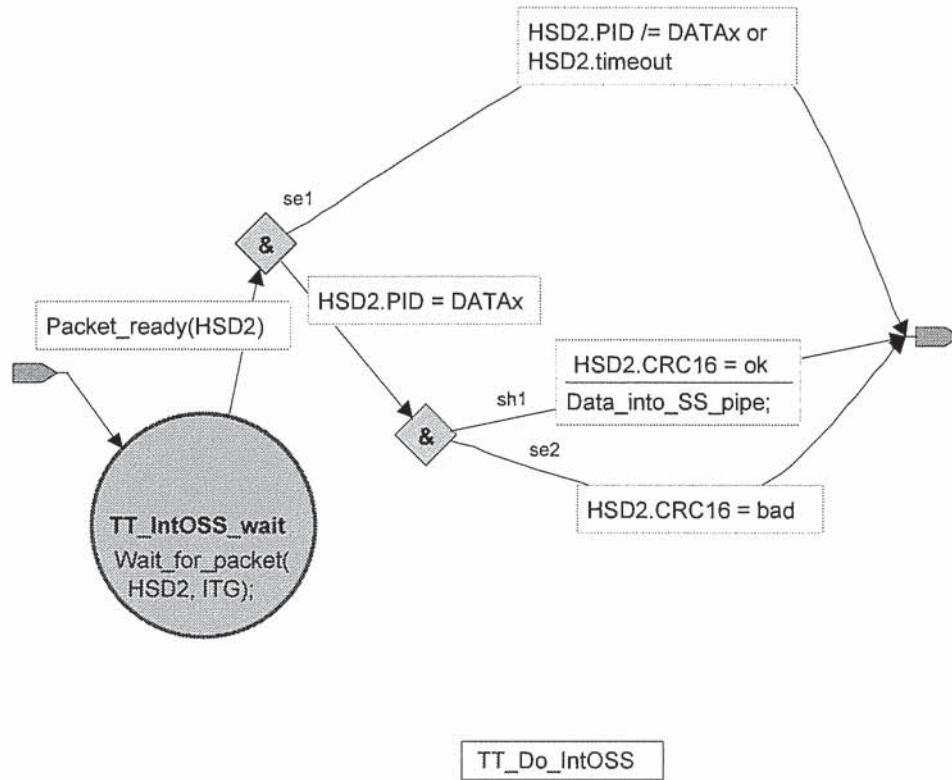


Figure 11-74. Interrupt OUT Start-split Transaction TT State Machine

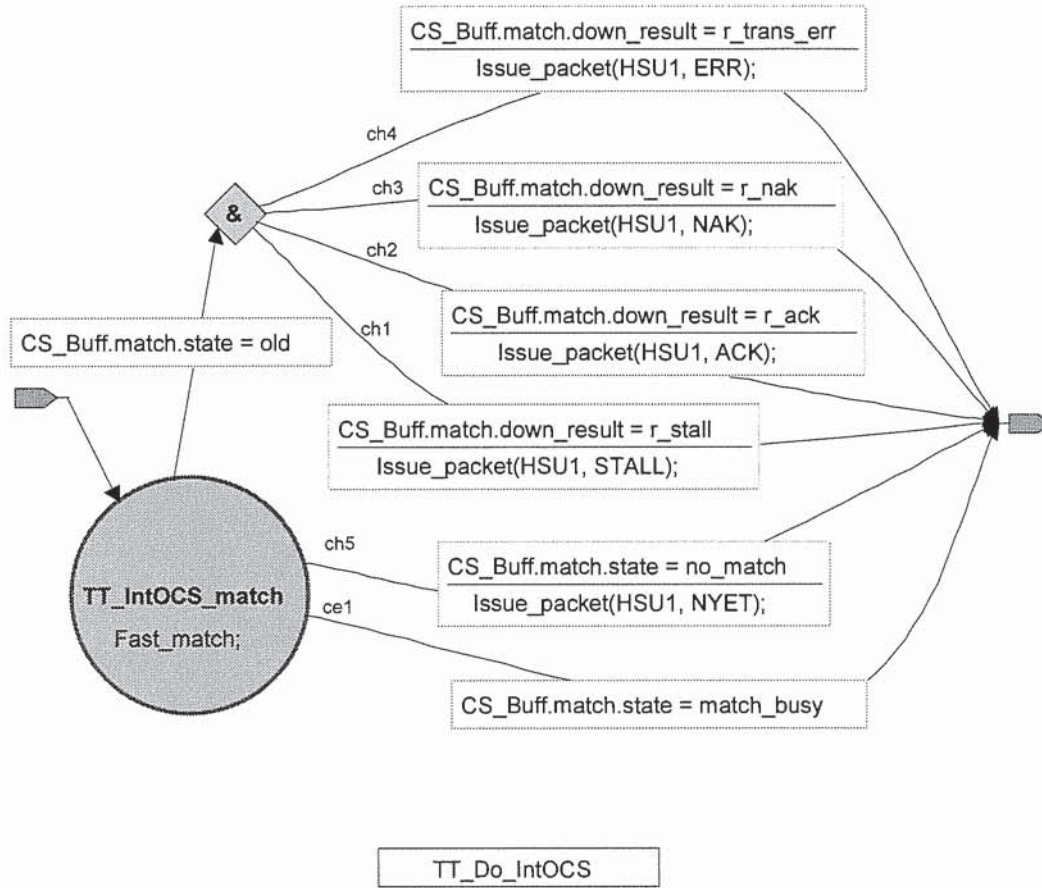


Figure 11-75. Interrupt OUT Complete-split Transaction TT State Machine

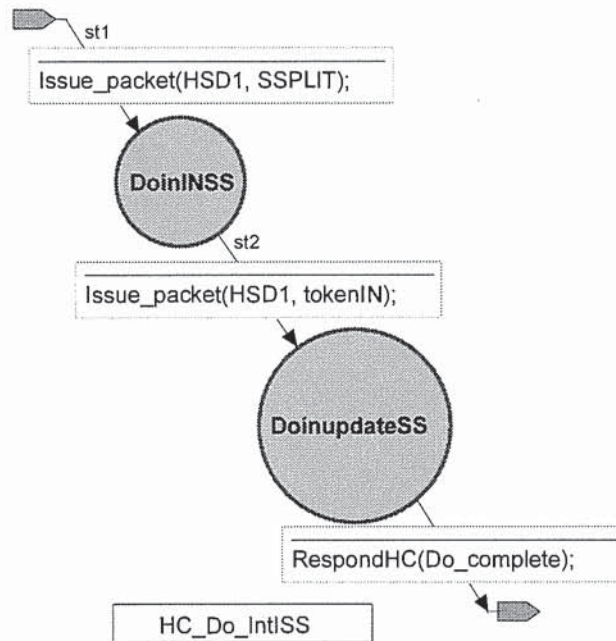


Figure 11-76. Interrupt IN Start-split Transaction Host State Machine

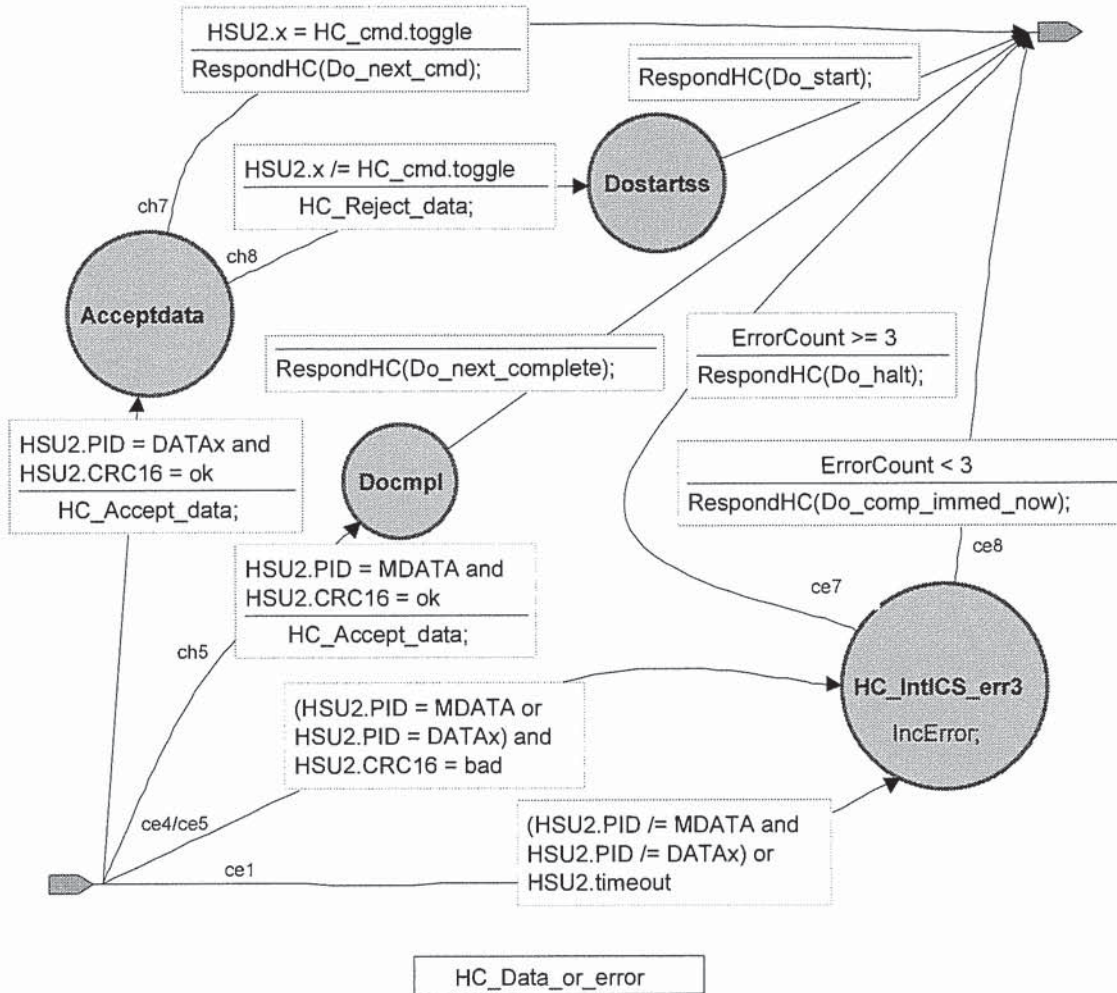


Figure 11-78. HC_Data_or_Error State Machine

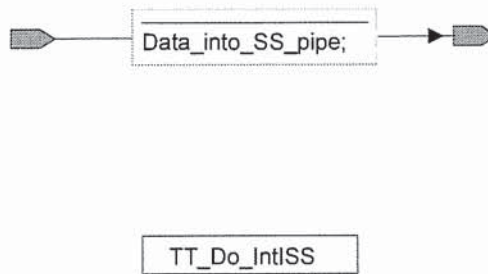


Figure 11-79. Interrupt IN Start-split Transaction TT State Machine

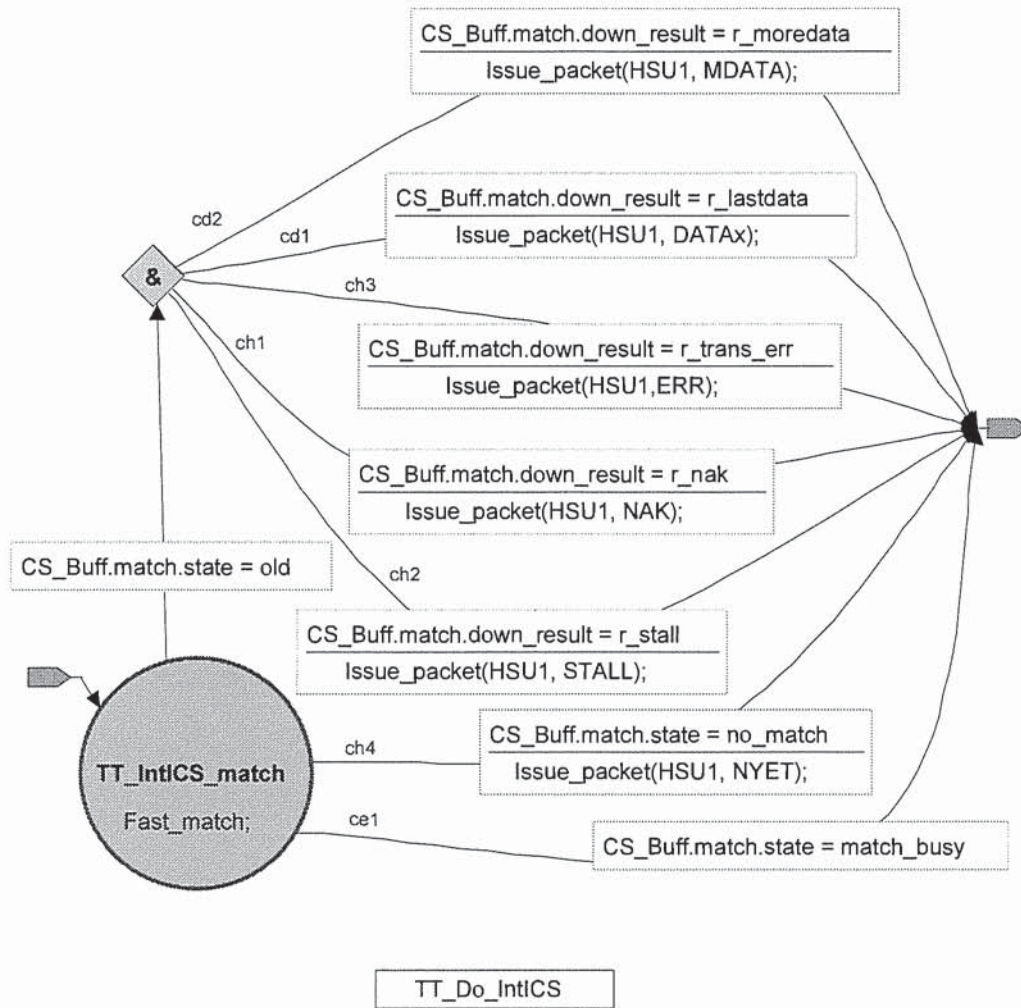


Figure 11-80. Interrupt IN Complete-split Transaction TT State Machine

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.

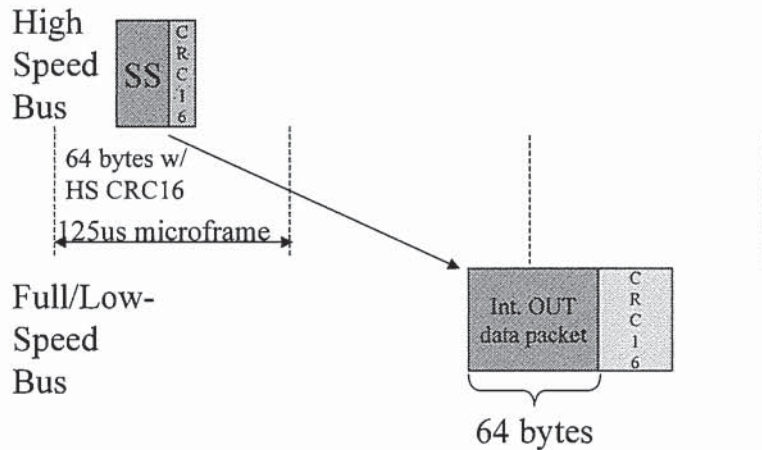


Figure 11-81. Example of CRC16 Handling for Interrupt OUT

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 a 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/I 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/I 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

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.

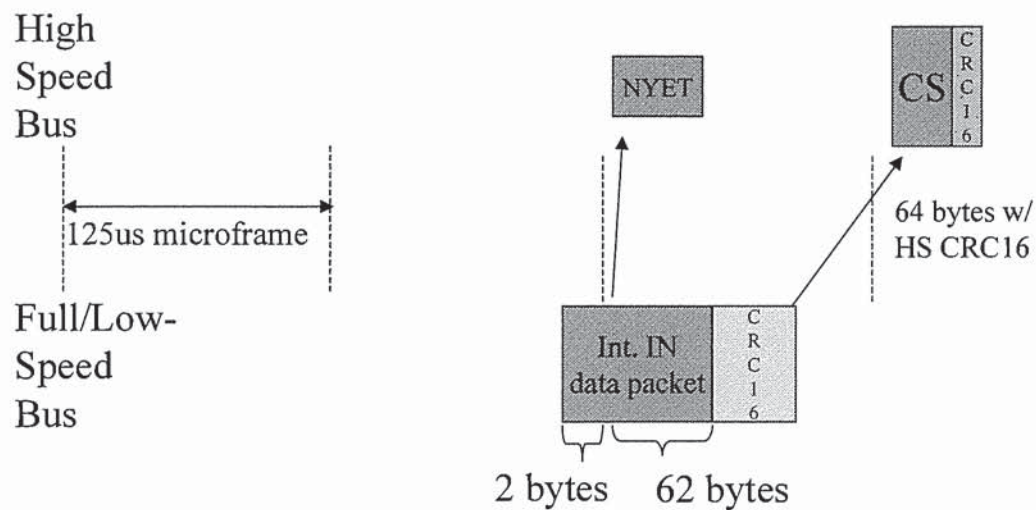


Figure 11-82. Example of CRC16 Handling for Interrupt IN

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.

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.

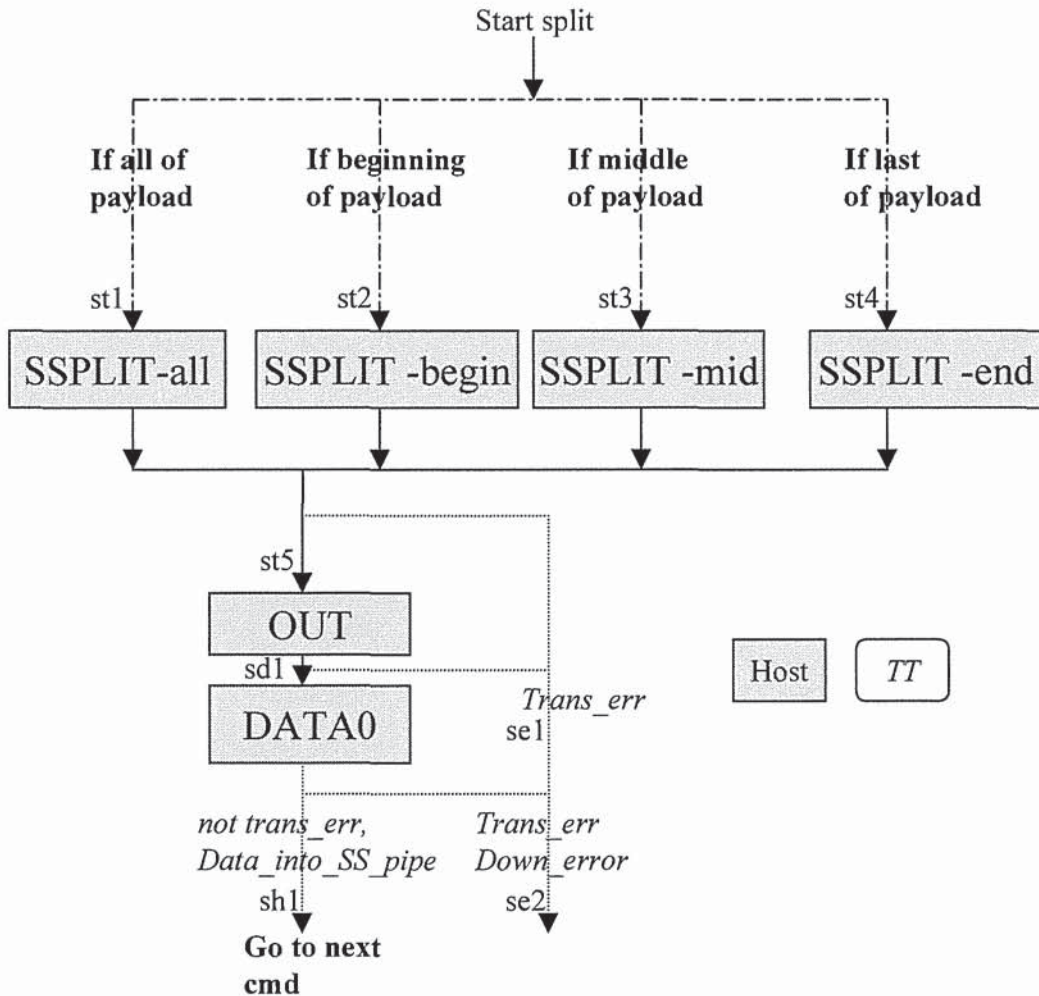


Figure 11-83. Isochronous OUT Start-split Transaction Sequence

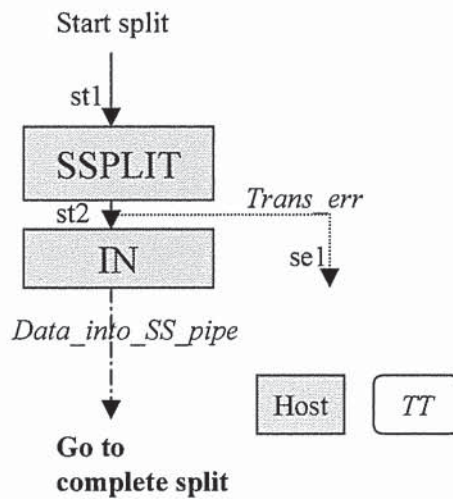


Figure 11-84. Isochronous IN Start-split Transaction Sequence

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.

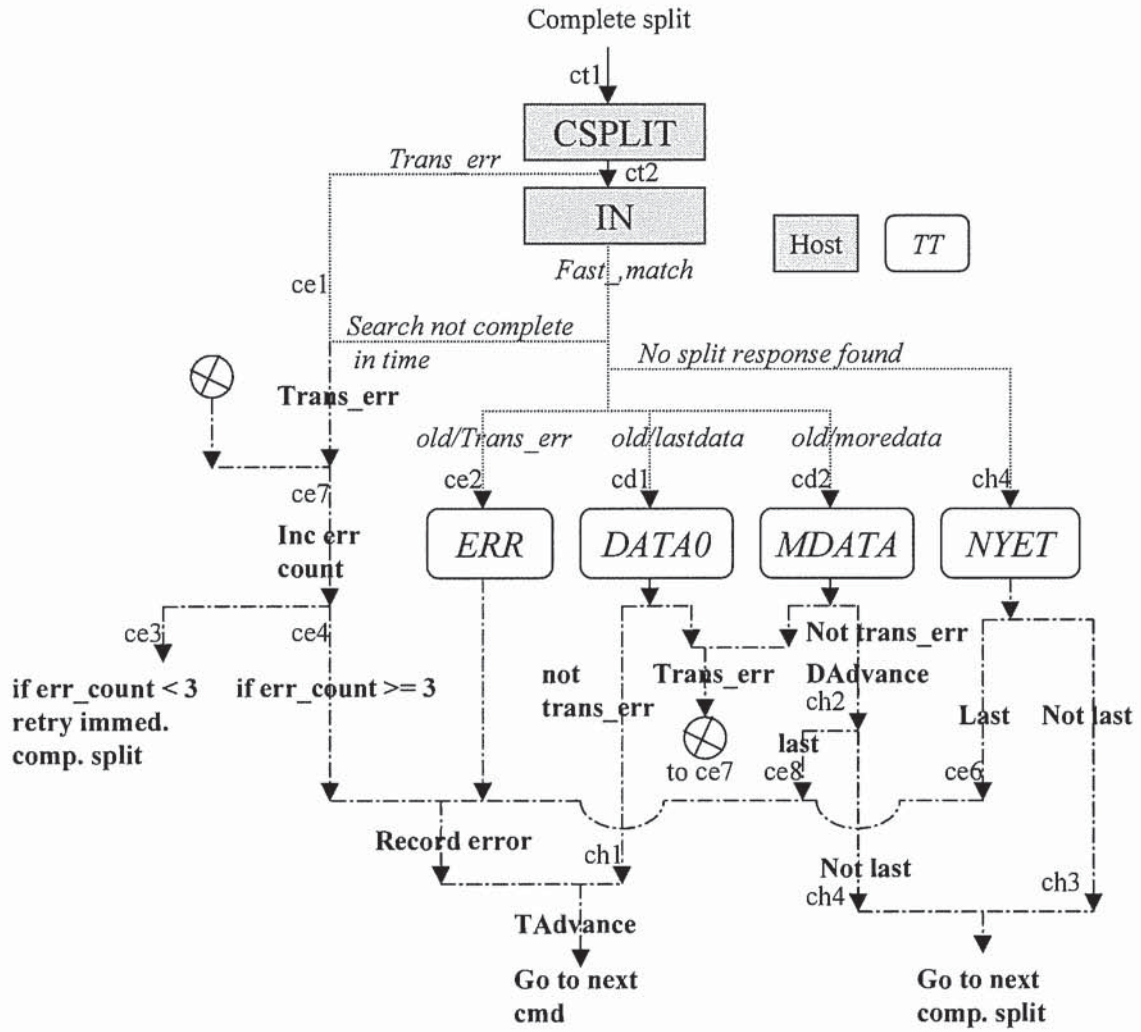


Figure 11-85. Isochronous IN Complete-split Transaction Sequence

11.21.2 Isochronous Split Transaction State Machines

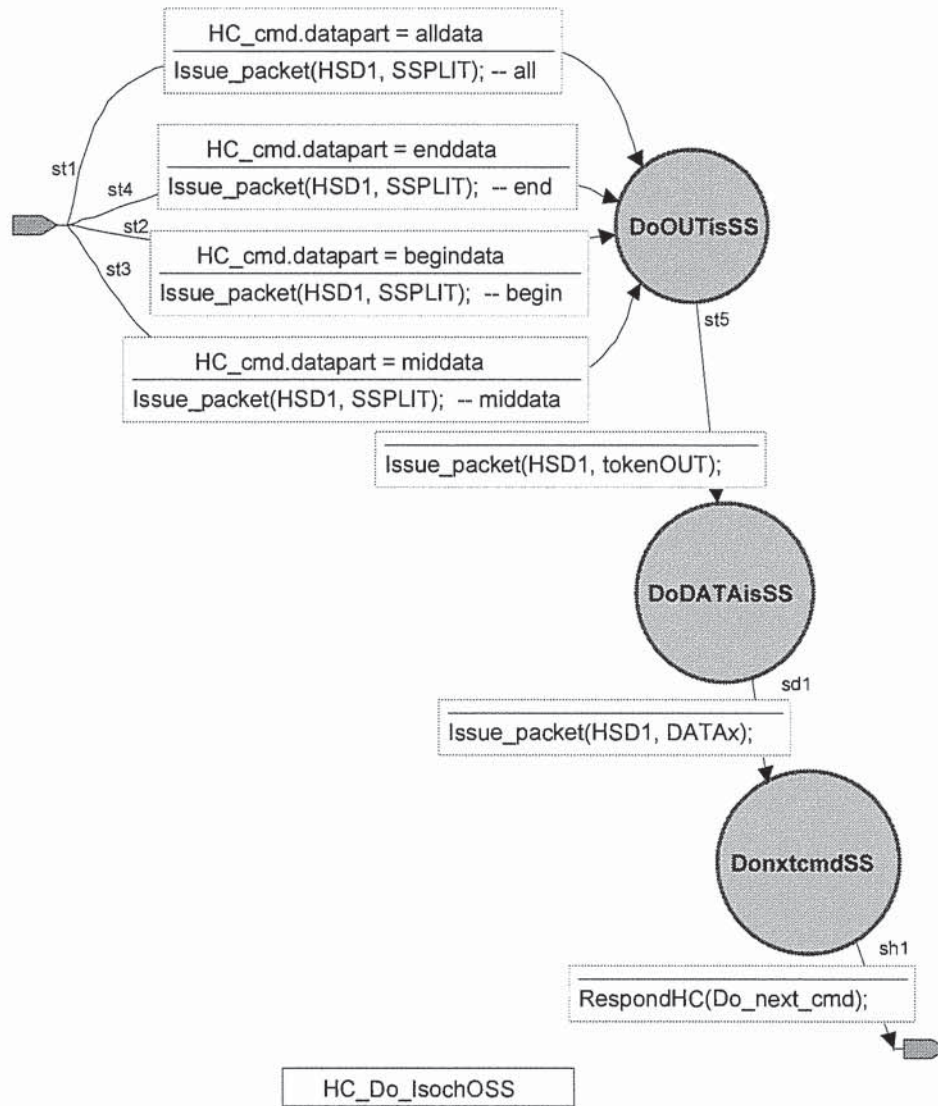


Figure 11-86. Isochronous OUT Start-split Transaction Host State Machine

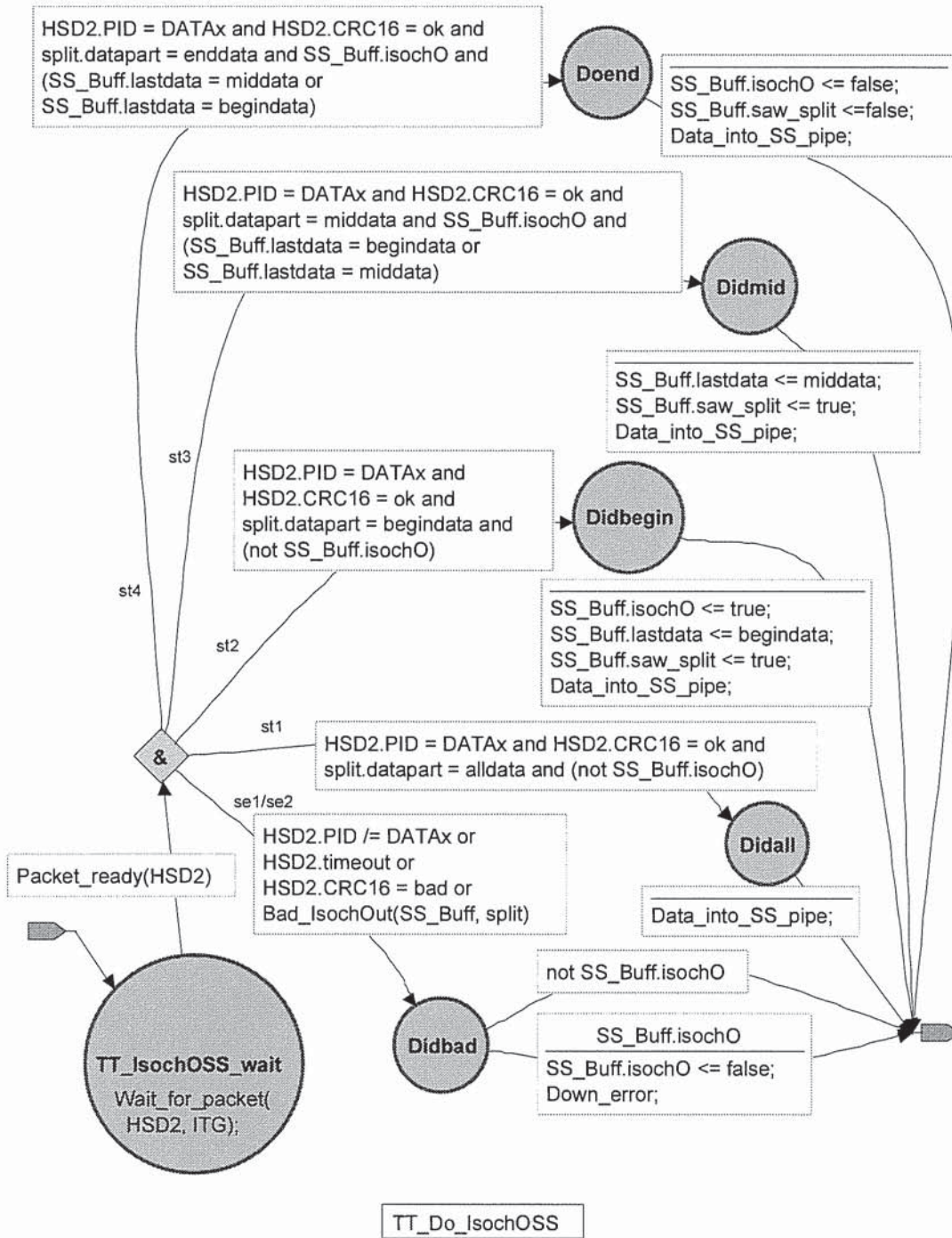


Figure 11-87. Isochronous OUT Start-split Transaction TT State Machine

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.

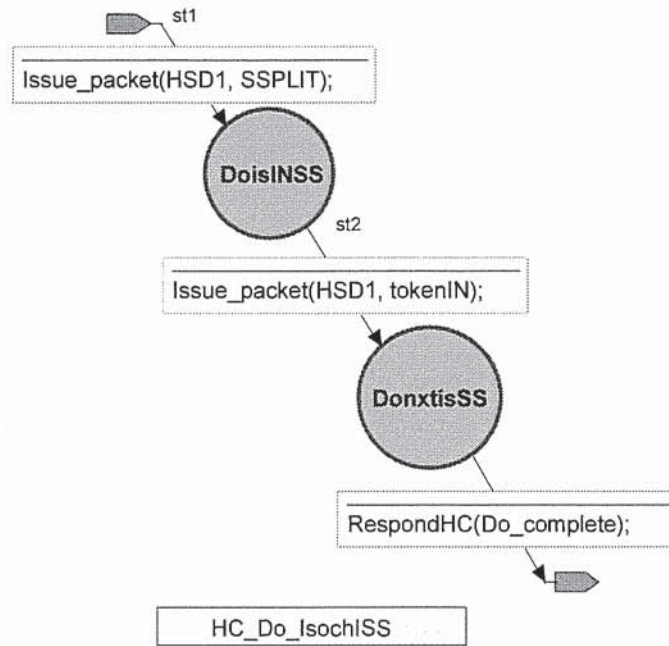


Figure 11-88. Isochronous IN Start-split Transaction Host State Machine

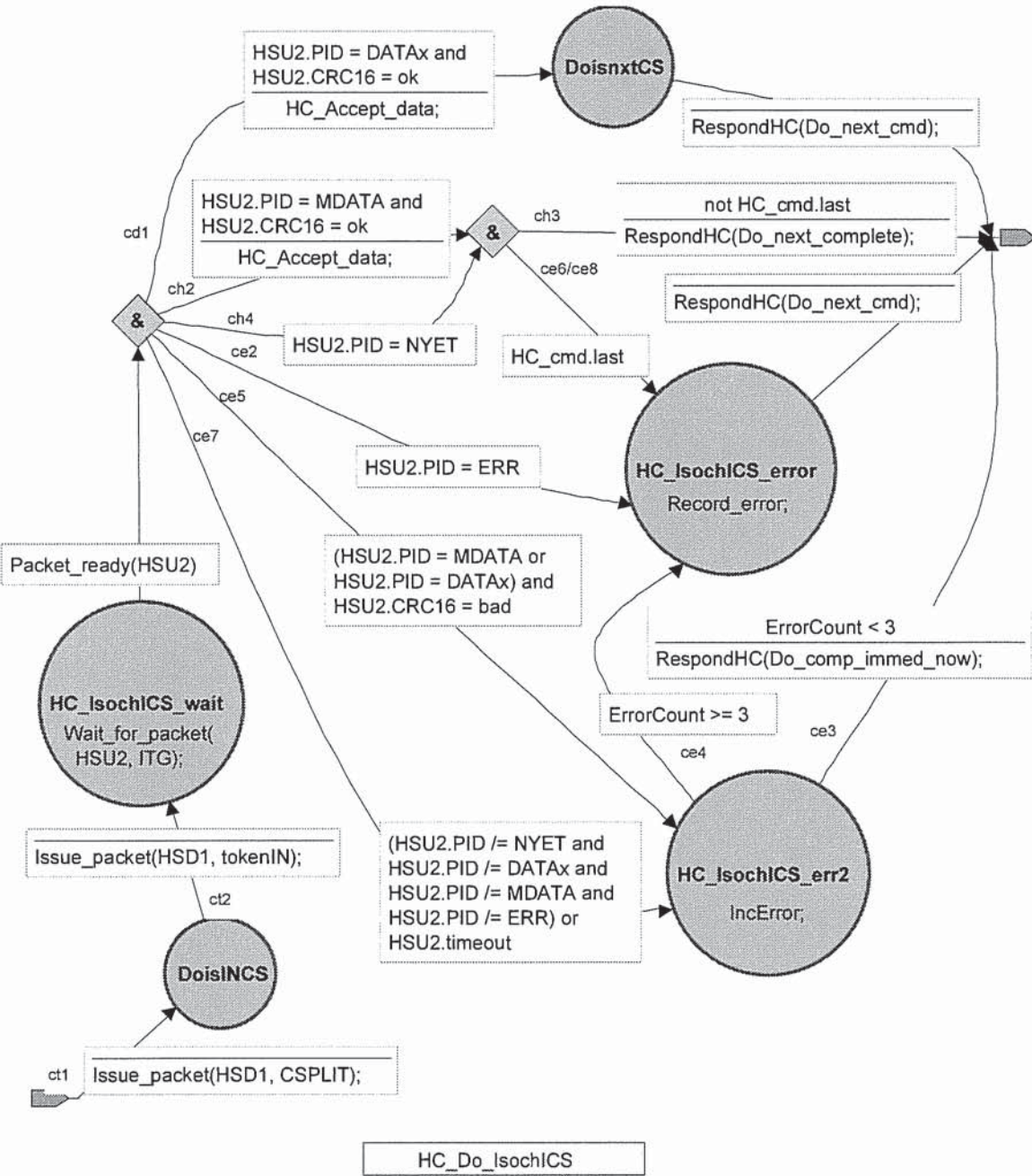
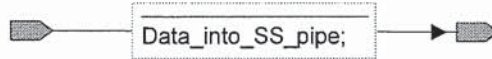


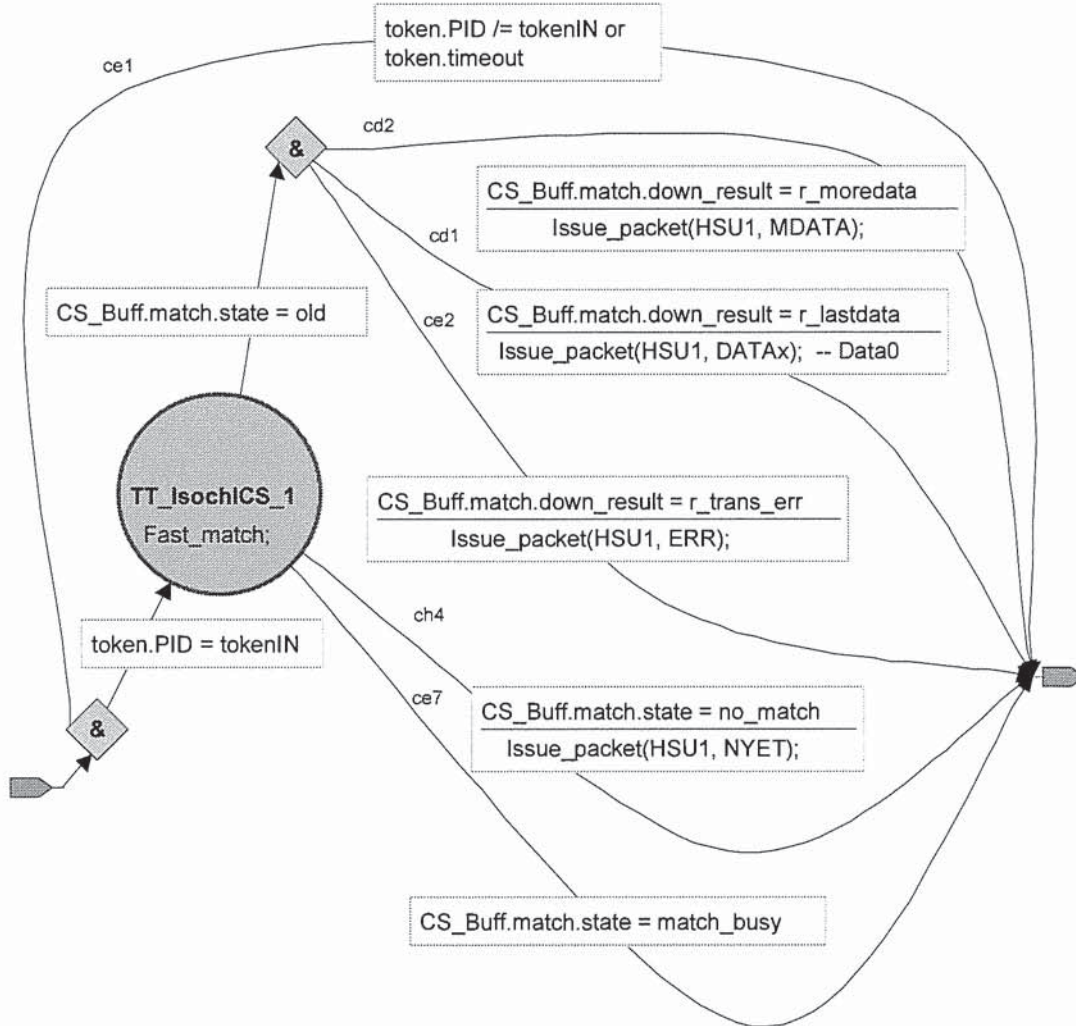
Figure 11-89. Isochronous IN Complete-split Transaction Host State Machine

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.

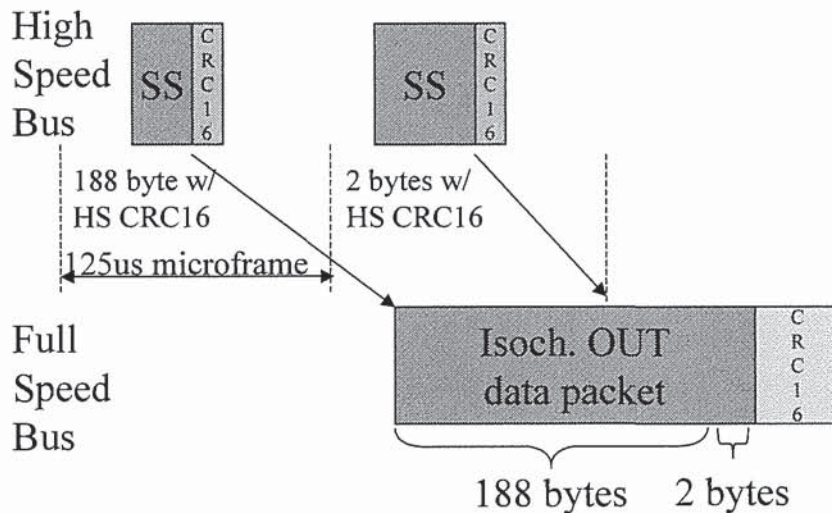


Figure 11-92. Example of CRC16 Isochronous OUT Data Packet Handling

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.

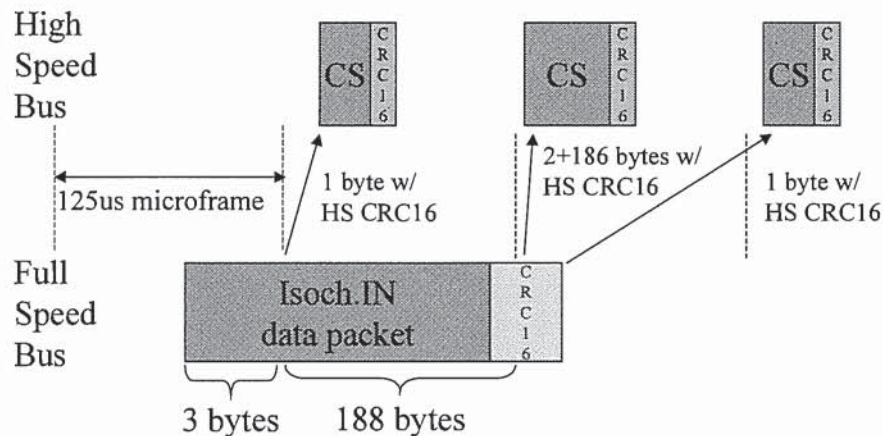


Figure 11-93. Example of CRC16 Isochronous IN Data Packet Handling

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 μ s microframe timer when operating at high-speed for enforcing EOF with high-speed devices. It also uses the 125 μ s 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 μ s 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 μ s 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 μ s 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 μ s 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 μ s 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).

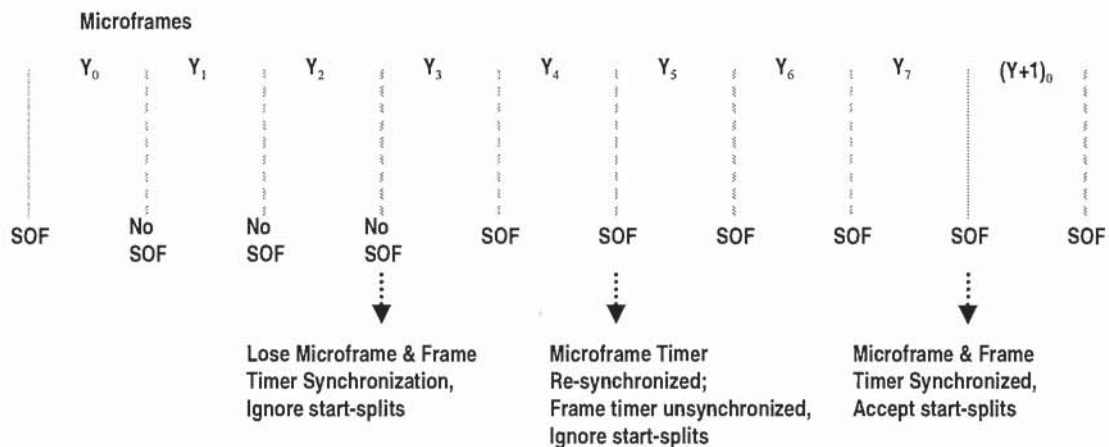


Figure 11-94. Example Frame/Microframe Synchronization Events

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):

<i>bLength</i>	12H
<i>bDescriptorType</i>	1
<i>bcdUSB</i>	0200H
<i>bDeviceClass</i>	HUB_CLASSCODE (09H)
<i>bDeviceSubClass</i>	0
<i>bDeviceProtocol</i>	0
<i>bMaxPacketSize0</i>	64
<i>bNumConfigurations</i>	1

Device_Qualifier Descriptor (high-speed information):

<i>bLength</i>	0AH
<i>bDescriptorType</i>	6
<i>bcdUSB</i>	200H
<i>bDeviceClass</i>	HUB_CLASSCODE (09H)
<i>bDeviceSubClass</i>	0
<i>bDeviceProtocol</i>	1 (for single TT) or 2 (for multiple TT)
<i>bMaxPacketSize0</i>	64
<i>bNumConfigurations</i>	1

Configuration Descriptor (full-speed information):

<i>bLength</i>	09H
<i>bDescriptorType</i>	2
<i>wTotalLength</i>	N
<i>bNumInterfaces</i>	1
<i>bConfigurationValue</i>	X
<i>iConfiguration</i>	Y
<i>bmAttributes</i>	Z
<i>bMaxPower</i>	The maximum amount of bus power the hub will consume in full-/low-speed configuration

Interface Descriptor:

<i>bLength</i>	09H
<i>bDescriptorType</i>	4
<i>bInterfaceNumber</i>	0
<i>bAlternateSetting</i>	0
<i>bNumEndpoints</i>	1
<i>bInterfaceClass</i>	HUB_CLASSCODE (09H)
<i>bInterfaceSubClass</i>	0
<i>bInterfaceProtocol</i>	0
<i>iInterface</i>	i

Endpoint Descriptor (for Status Change Endpoint):

<i>bLength</i>	07H
<i>bDescriptorType</i>	5
<i>bEndpointAddress</i>	Implementation-dependent; Bit 7: Direction = In(1)
<i>bmAttributes</i>	Transfer Type = Interrupt (00000011B)
<i>wMaxPacketSize</i>	Implementation-dependent
<i>bInterval</i>	FFH (Maximum allowable interval)

Other_Speed_Configuration Descriptor (High-speed information):

<i>bLength</i>	09H
<i>bDescriptorType</i>	7
<i>wTotalLength</i>	N
<i>bNumInterfaces</i>	1 (for single TT) or 2 (for multiple TT)
<i>bConfigurationValue</i>	X
<i>iConfiguration</i>	Y
<i>bmAttributes</i>	Z
<i>bMaxPower</i>	The maximum amount of bus power the hub will consume in high-speed configuration

Interface Descriptor:

<i>bLength</i>	09H
<i>bDescriptorType</i>	4
<i>bInterfaceNumber</i>	0
<i>bAlternateSetting</i>	0
<i>bNumEndpoints</i>	1
<i>bInterfaceClass</i>	HUB_CLASSCODE (09H)
<i>bInterfaceSubClass</i>	0
<i>bInterfaceProtocol</i>	0 (for single TT) 1 (for multiple TT)
<i>iInterface</i>	i

Endpoint Descriptor (for Status Change Endpoint):

<i>bLength</i>	07H
<i>bDescriptorType</i>	5
<i>bEndpointAddress</i>	Implementation-dependent; Bit 7: Direction = In(1)
<i>bmAttributes</i>	Transfer Type = Interrupt (00000011B)
<i>wMaxPacketSize</i>	Implementation-dependent
<i>bInterval</i>	FFH (Maximum allowable interval)

Interface Descriptor (present if multiple TT hub):

<i>bLength</i>	09H
<i>bDescriptorType</i>	4
<i>bInterfaceNumber</i>	0
<i>bAlternateSetting</i>	0
<i>bNumEndpoints</i>	1
<i>bInterfaceClass</i>	HUB_CLASSCODE (09H)
<i>bInterfaceSubClass</i>	0
<i>bInterfaceProtocol</i>	2
<i>iInterface</i>	i

Endpoint Descriptor (present if multiple TT hub):

<i>bLength</i>	07H
<i>bDescriptorType</i>	5
<i>bEndpointAddress</i>	Implementation-dependent; Bit 7: Direction = In(1)
<i>bmAttributes</i>	Transfer Type = Interrupt (00000011B)
<i>wMaxPacketSize</i>	Implementation-dependent
<i>bInterval</i>	FFH (Maximum allowable interval)

High-speed Operating Hub with Single TT

Device Descriptor (High-speed information):

<i>bLength</i>	12H
<i>bDescriptorType</i>	1
<i>bcdUSB</i>	200H
<i>bDeviceClass</i>	HUB_CLASSCODE (09H)
<i>bDeviceSubClass</i>	0
<i>bDeviceProtocol</i>	1
<i>bMaxPacketSize0</i>	64
<i>bNumConfigurations</i>	1

Device_Qualifier Descriptor (full-speed information):

<i>bLength</i>	0AH
<i>bDescriptorType</i>	6
<i>bcdUSB</i>	200H
<i>bDeviceClass</i>	HUB_CLASSCODE (09H)
<i>bDeviceSubClass</i>	0
<i>bDeviceProtocol</i>	0
<i>bMaxPacketSize0</i>	64
<i>bNumConfigurations</i>	1

Configuration Descriptor (high-speed information):

<i>bLength</i>	09H
<i>bDescriptorType</i>	2
<i>wTotalLength</i>	N
<i>bNumInterfaces</i>	1
<i>bConfigurationValue</i>	X
<i>iConfiguration</i>	Y
<i>bmAttributes</i>	Z
<i>bMaxPower</i>	The maximum amount of bus power the hub will consume in this configuration

Interface Descriptor:

<i>bLength</i>	09H
<i>bDescriptorType</i>	4
<i>bInterfaceNumber</i>	0
<i>bAlternateSetting</i>	0
<i>bNumEndpoints</i>	1
<i>bInterfaceClass</i>	HUB_CLASSCODE (09H)
<i>bInterfaceSubClass</i>	0
<i>bInterfaceProtocol</i>	0 (single TT)
<i>iInterface</i>	i

Endpoint Descriptor (for Status Change Endpoint):

<i>bLength</i>	07H
<i>bDescriptorType</i>	5
<i>bEndpointAddress</i>	Implementation-dependent; Bit 7: Direction = In(1)
<i>bmAttributes</i>	Transfer Type = Interrupt (00000011B)
<i>wMaxPacketSize</i>	Implementation-dependent
<i>bInterval</i>	FFH (Maximum allowable interval)

Universal Serial Bus Specification Revision 2.0

Other_Speed_Configuration Descriptor (full-speed information):

<i>bLength</i>	09H
<i>bDescriptorType</i>	7
<i>wTotalLength</i>	N
<i>bNumInterfaces</i>	1
<i>bConfigurationValue</i>	X
<i>iConfiguration</i>	Y
<i>bmAttributes</i>	Z
<i>bMaxPower</i>	The maximum amount of bus power the hub will consume in high-speed configuration

Interface Descriptor:

<i>bLength</i>	09H
<i>bDescriptorType</i>	4
<i>bInterfaceNumber</i>	0
<i>bAlternateSetting</i>	0
<i>bNumEndpoints</i>	1
<i>bInterfaceClass</i>	HUB_CLASSCODE (09H)
<i>bInterfaceSubClass</i>	0
<i>bInterfaceProtocol</i>	0
<i>iInterface</i>	i

Endpoint Descriptor (for Status Change Endpoint):

<i>bLength</i>	07H
<i>bDescriptorType</i>	5
<i>bEndpointAddress</i>	Implementation-dependent; Bit 7: Direction = In(1)
<i>bmAttributes</i>	Transfer Type = Interrupt (00000011B)
<i>wMaxPacketSize</i>	Implementation-dependent
<i>bInterval</i>	FFH (Maximum allowable interval)

High-speed Operating Hub with Multiple TTs

Device Descriptor (High-speed information):

<i>bLength</i>	12H
<i>bDescriptorType</i>	1
<i>bcdUSB</i>	200H
<i>bDeviceClass</i>	HUB_CLASSCODE (09H)
<i>bDeviceSubClass</i>	0
<i>bDeviceProtocol</i>	2 (multiple TTs)
<i>bMaxPacketSize0</i>	64
<i>bNumConfigurations</i>	1

Device_Qualifier Descriptor (full-speed information):

<i>bLength</i>	0AH
<i>bDescriptorType</i>	6
<i>bcdUSB</i>	200H
<i>bDeviceClass</i>	HUB_CLASSCODE (09H)
<i>bDeviceSubClass</i>	0
<i>bDeviceProtocol</i>	0
<i>bMaxPacketSize0</i>	64
<i>bNumConfigurations</i>	1

Configuration Descriptor (high-speed information):

<i>bLength</i>	09H
<i>bDescriptorType</i>	2
<i>wTotalLength</i>	N
<i>bNumInterfaces</i>	1
<i>bConfigurationValue</i>	X
<i>iConfiguration</i>	Y
<i>bmAttributes</i>	Z
<i>bMaxPower</i>	The maximum amount of bus power the hub will consume in this configuration

Interface Descriptor:

<i>bLength</i>	09H
<i>bDescriptorType</i>	4
<i>bInterfaceNumber</i>	0
<i>bAlternateSetting</i>	0
<i>bNumEndpoints</i>	1
<i>bInterfaceClass</i>	HUB_CLASSCODE (09H)
<i>bInterfaceSubClass</i>	0
<i>bInterfaceProtocol</i>	1 (single TT)
<i>iInterface</i>	i

Endpoint Descriptor (for Status Change Endpoint):

<i>bLength</i>	07H
<i>bDescriptorType</i>	5
<i>bEndpointAddress</i>	Implementation-dependent; Bit 7: Direction = In(1)
<i>bmAttributes</i>	Transfer Type = Interrupt (00000011B)
<i>wMaxPacketSize</i>	Implementation-dependent
<i>bInterval</i>	FFH (Maximum allowable interval)

Interface Descriptor:

<i>bLength</i>	09H
<i>bDescriptorType</i>	4
<i>bInterfaceNumber</i>	0
<i>bAlternateSetting</i>	1
<i>bNumEndpoints</i>	1
<i>bInterfaceClass</i>	HUB_CLASSCODE (09H)
<i>bInterfaceSubClass</i>	0
<i>bInterfaceProtocol</i>	2 (multiple TTs)
<i>iInterface</i>	i

Endpoint Descriptor:

<i>bLength</i>	07H
<i>bDescriptorType</i>	5
<i>bEndpointAddress</i>	Implementation-dependent; Bit 7: Direction = In(1)
<i>bmAttributes</i>	Transfer Type = Interrupt (00000011B)
<i>wMaxPacketSize</i>	Implementation-dependent
<i>bInterval</i>	FFH (Maximum allowable interval)

Other_Speed_Configuration Descriptor (full-speed information):

<i>bLength</i>	09H
<i>bDescriptorType</i>	7
<i>wTotalLength</i>	N
<i>bNumInterfaces</i>	1
<i>bConfigurationValue</i>	X
<i>iConfiguration</i>	Y
<i>bmAttributes</i>	Z
<i>bMaxPower</i>	The maximum amount of bus power the hub will consume in high-speed configuration

Interface Descriptor:

<i>bLength</i>	09H
<i>bDescriptorType</i>	4
<i>bInterfaceNumber</i>	0
<i>bAlternateSetting</i>	0
<i>bNumEndpoints</i>	1
<i>bInterfaceClass</i>	HUB_CLASSCODE (09H)
<i>bInterfaceSubClass</i>	0
<i>bInterfaceProtocol</i>	0
<i>iInterface</i>	i

Endpoint Descriptor (for Status Change Endpoint):

<i>bLength</i>	07H
<i>bDescriptorType</i>	5
<i>bEndpointAddress</i>	Implementation-dependent; Bit 7: Direction = In(1)
<i>bmAttributes</i>	Transfer Type = Interrupt (00000011B)
<i>wMaxPacketSize</i>	Implementation-dependent
<i>bInterval</i>	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	<i>bDescLength</i>	1	Number of bytes in this descriptor, including this byte
1	<i>bDescriptorType</i>	1	Descriptor Type, value: 29H for hub descriptor
2	<i>bNbrPorts</i>	1	Number of downstream facing ports that this hub supports
3	<i>wHubCharacteristics</i>	2	<p>D1...D0: Logical Power Switching Mode</p> <ul style="list-style-type: none"> 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 <p>D2: Identifies a Compound Device</p> <ul style="list-style-type: none"> 0: Hub is not part of a compound device. 1: Hub is part of a compound device. <p>D4...D3: Over-current Protection Mode</p> <ul style="list-style-type: none"> 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.

Universal Serial Bus Specification Revision 2.0

Offset	Field	Size	Description
			<p>D6...D5: TT Think Time</p> <p>00: TT requires at most 8 FS bit times of inter transaction gap on a full-/low-speed downstream bus.</p> <p>01: TT requires at most 16 FS bit times.</p> <p>10: TT requires at most 24 FS bit times.</p> <p>11: TT requires at most 32 FS bit times.</p> <p>D7: Port Indicators Supported</p> <p>0: Port Indicators are not supported on its downstream facing ports and the PORT_INDICATOR request has no effect.</p> <p>1: Port Indicators are supported on its downstream facing ports and the PORT_INDICATOR request controls the indicators. See Section 11.5.3.</p> <p>D15...D8: Reserved</p>
5	<i>bPwrOn2PwrGood</i>	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	<i>bHubContrCurrent</i>	1	Maximum current requirements of the Hub Controller electronics in mA.
7	<i>DeviceRemovable</i>	Variable, depending on number of ports on hub	<p>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.</p> <p>Bit value definition:</p> <p>0B - Device is removable.</p> <p>1B - Device is non-removable</p> <p>This is a bitmap corresponding to the individual ports on the hub:</p> <p>Bit 0: Reserved for future use.</p> <p>Bit 1: Port 1</p> <p>Bit 2: Port 2</p> <p>....</p> <p>Bit <i>n</i>: Port <i>n</i> (implementation-dependent, up to a maximum of 255 ports).</p>
Variable	<i>PortPwrCtrlMask</i>	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
<i>Reserved for future use</i>	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	<p>Local Power Source: This is the source of the local power supply.</p> <p>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.</p> <p>0 = Local power supply good 1 = Local power supply lost (inactive)</p>
1	<p>Over-current:</p> <p>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.</p> <p>0 = No over-current condition currently exists. 1 = A hub over-current condition exists.</p>
2-15	<p>Reserved</p> <p>These bits return 0 when read.</p>

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	<p>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 <i>wHubStatus</i>.</p> <p>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.</p>
1	<p>Over-Current Change: (C_HUB_OVER_CURRENT) This field indicates if a change has occurred in the Over-Current field in <i>wHubStatus</i>.</p> <p>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.</p>
2-15	<p>Reserved These bits return 0 when read.</p>

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 circumstances:

- 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	<p>Connect Status Change: (<code>C_PORT_CONNECTION</code>) 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.</p> <p>0 = No change has occurred to Current Connect status. 1 = Current Connect status has changed.</p>
1	<p>Port Enable/Disable Change: (<code>C_PORT_ENABLE</code>) This field is set to one when a port is disabled because of a <code>Port_Error</code> condition (see Section 11.8.1).</p>
2	<p>Suspend Change: (<code>C_PORT_SUSPEND</code>) 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.</p> <p>0 = No change. 1 = Resume complete.</p>
3	<p>Over-Current Indicator Change: (<code>C_PORT_OVER_CURRENT</code>) This field applies only to hubs that report over-current conditions on a per-port basis (as reported in the hub descriptor).</p> <p>0 = No change has occurred to Over-Current Indicator. 1 = Over-Current Indicator has changed.</p> <p>If the hub does not report over-current on a per-port basis, then this field is always zero.</p>
4	<p>Reset Change: (<code>C_PORT_RESET</code>) This field is set when reset processing on this port is complete.</p> <p>0 = No change. 1 = Reset complete.</p>
5-15	<p>Reserved These bits return 0 when read.</p>

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

Universal Serial Bus Specification Revision 2.0

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

Universal Serial Bus Specification Revision 2.0

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) available 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

Universal Serial Bus Specification Revision 2.0

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

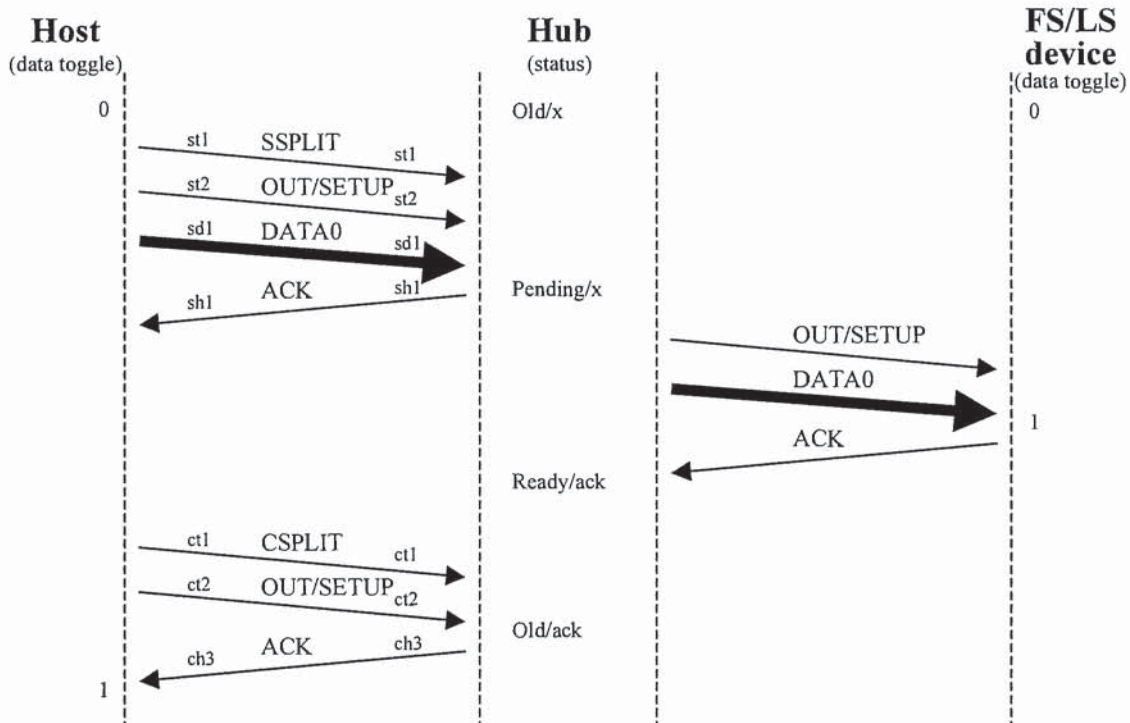


Figure A-1. Normal No Smash

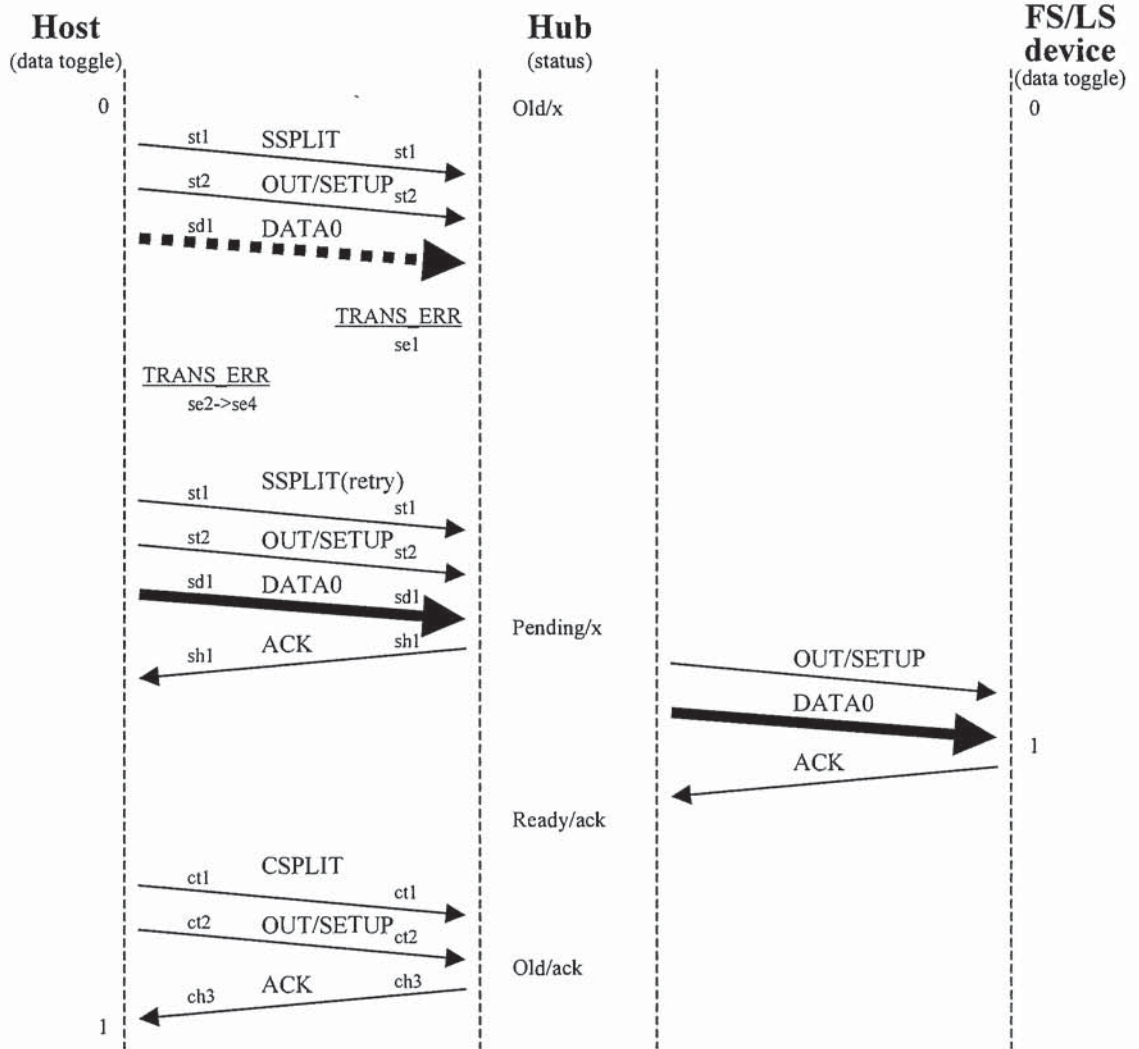


Figure A-2. Normal HS DATA0/1 Smash

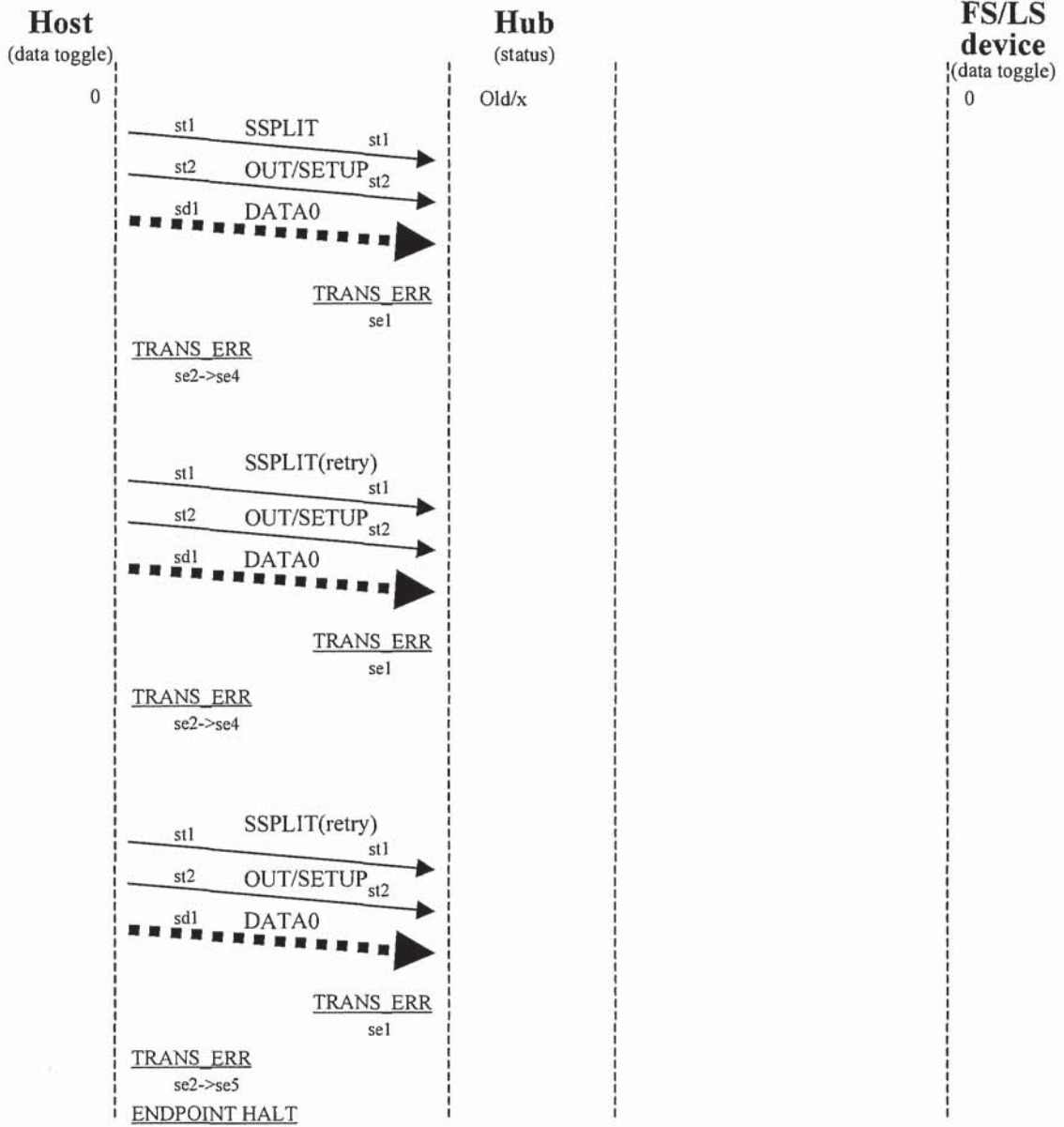


Figure A-3. Normal HS DATA0/1 3 Strikes Smash

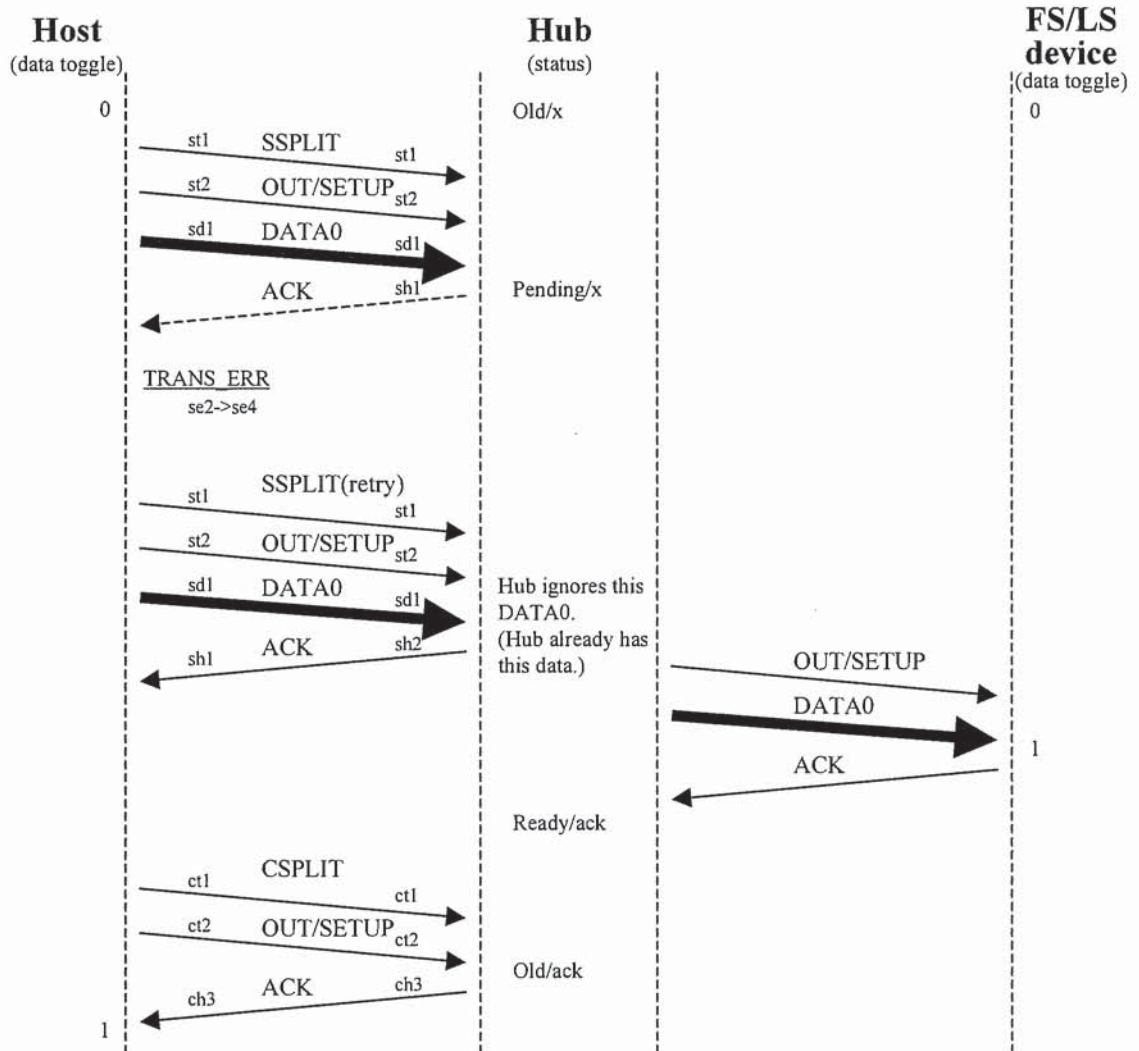


Figure A-4. Normal HS ACK(S) Smash(case 1)

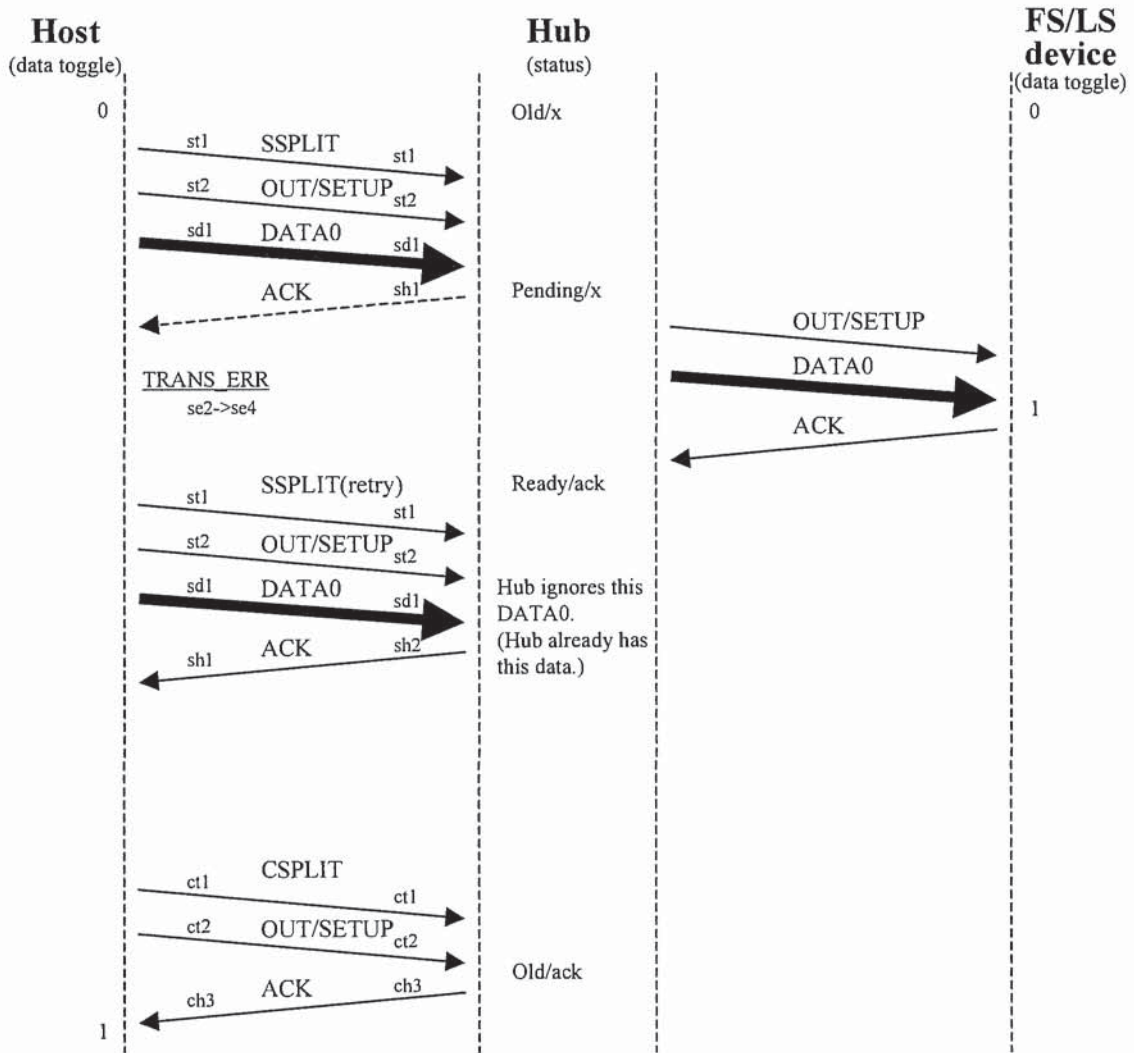


Figure A-5. Normal HS ACK(S) Smash(case 2)

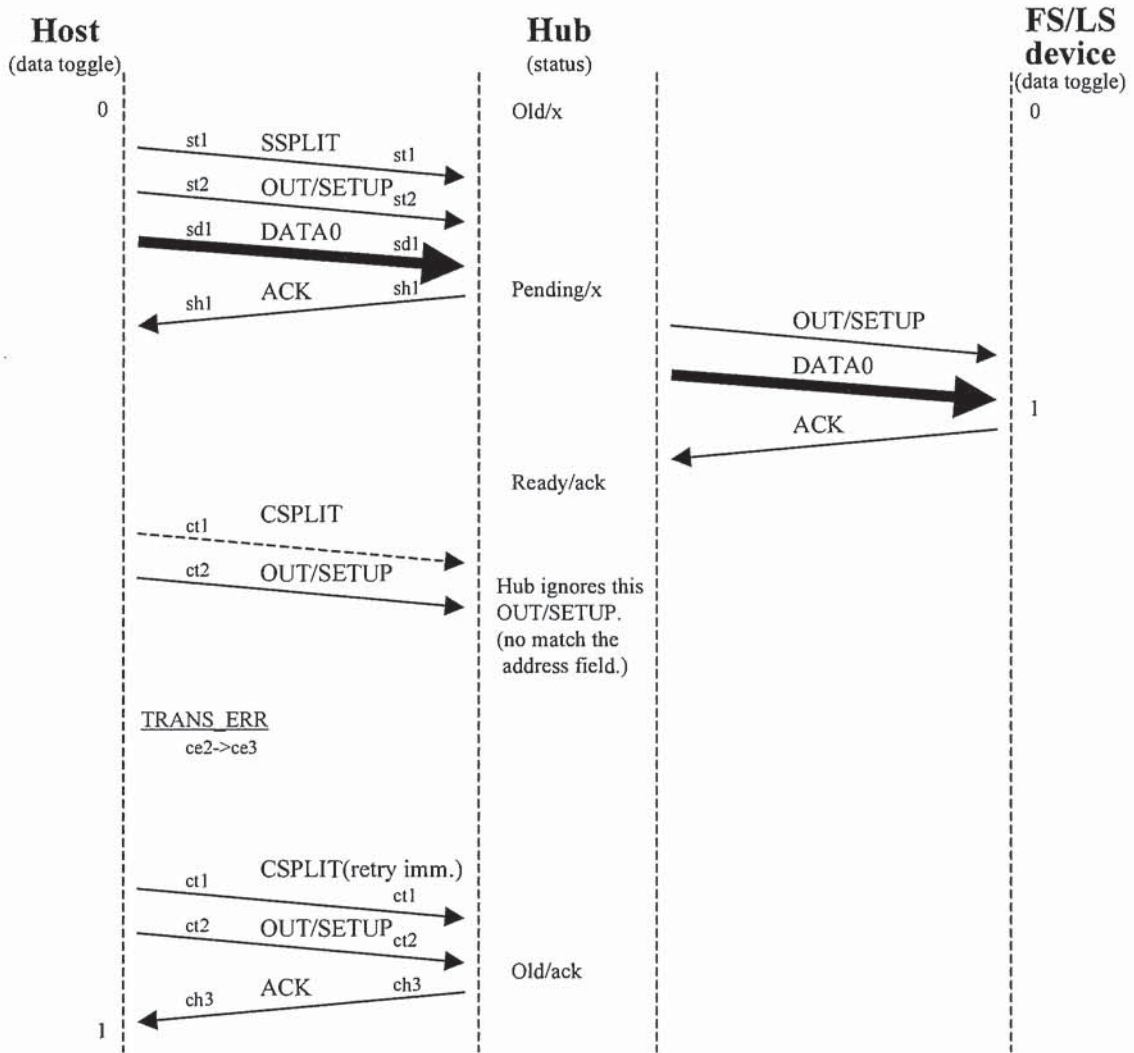


Figure A-7. Normal HS CSPLIT Smash

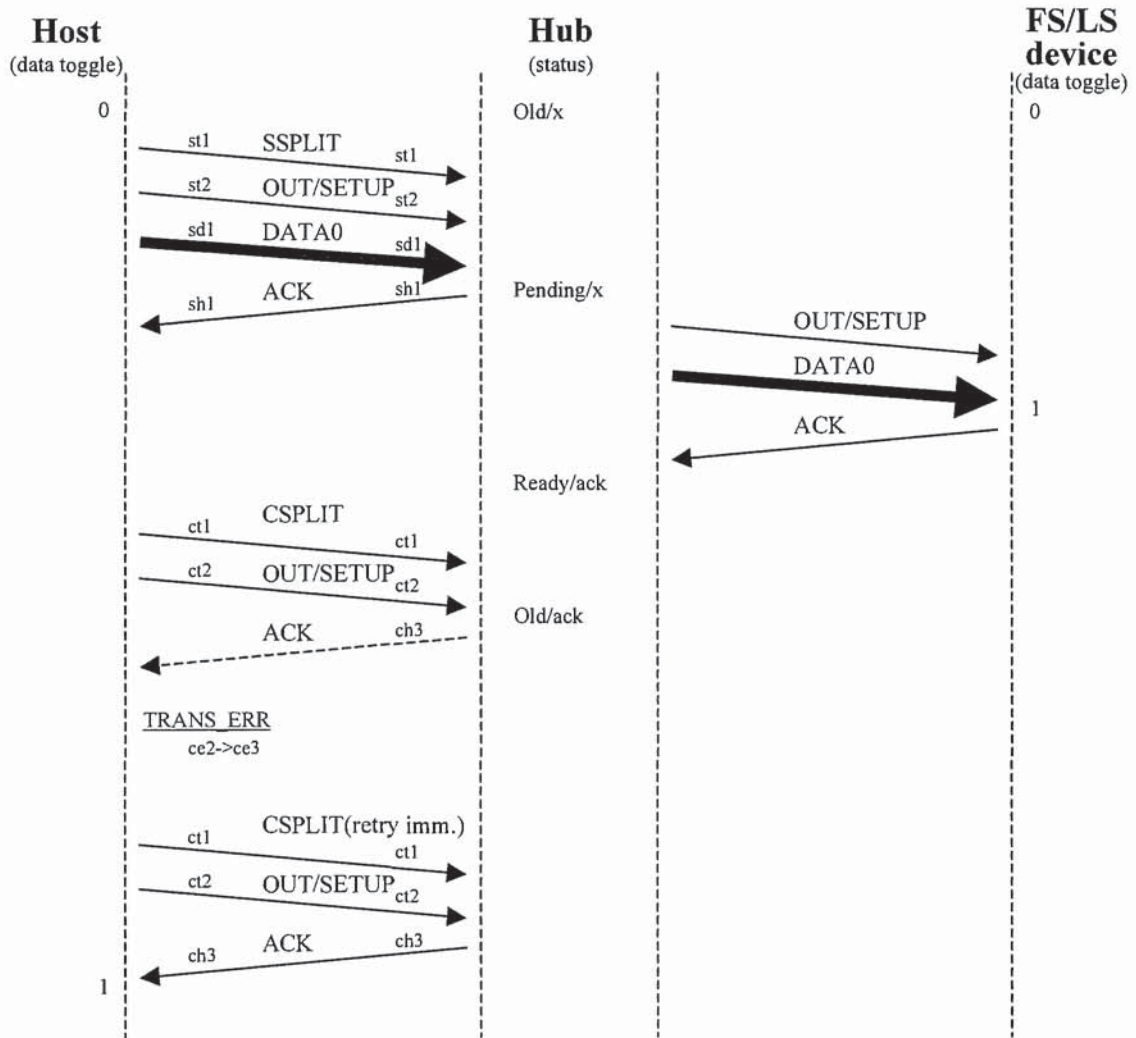


Figure A-9. Normal HS ACK(C) Smash

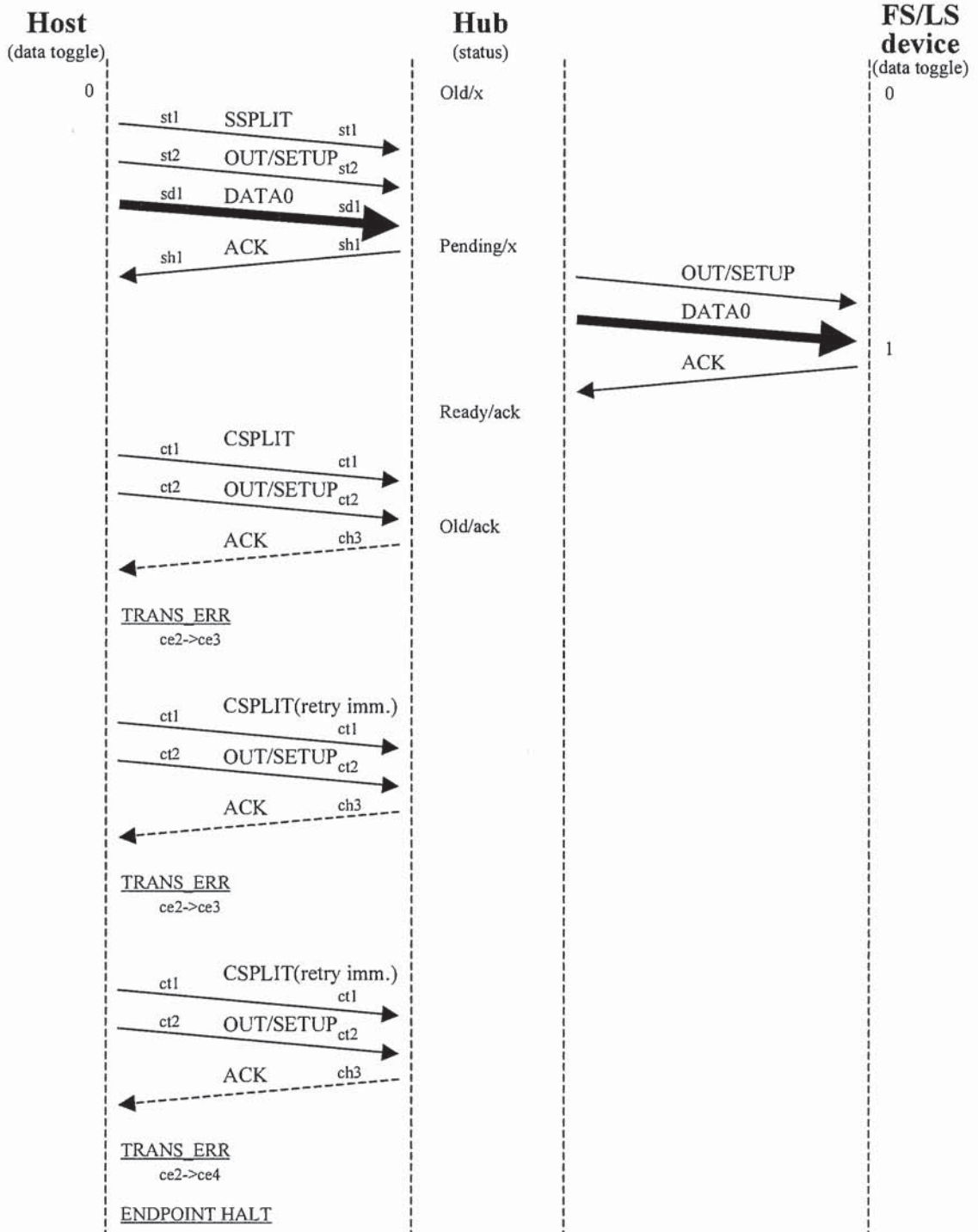


Figure A-10. Normal S ACK(C) 3 Strikes Smash

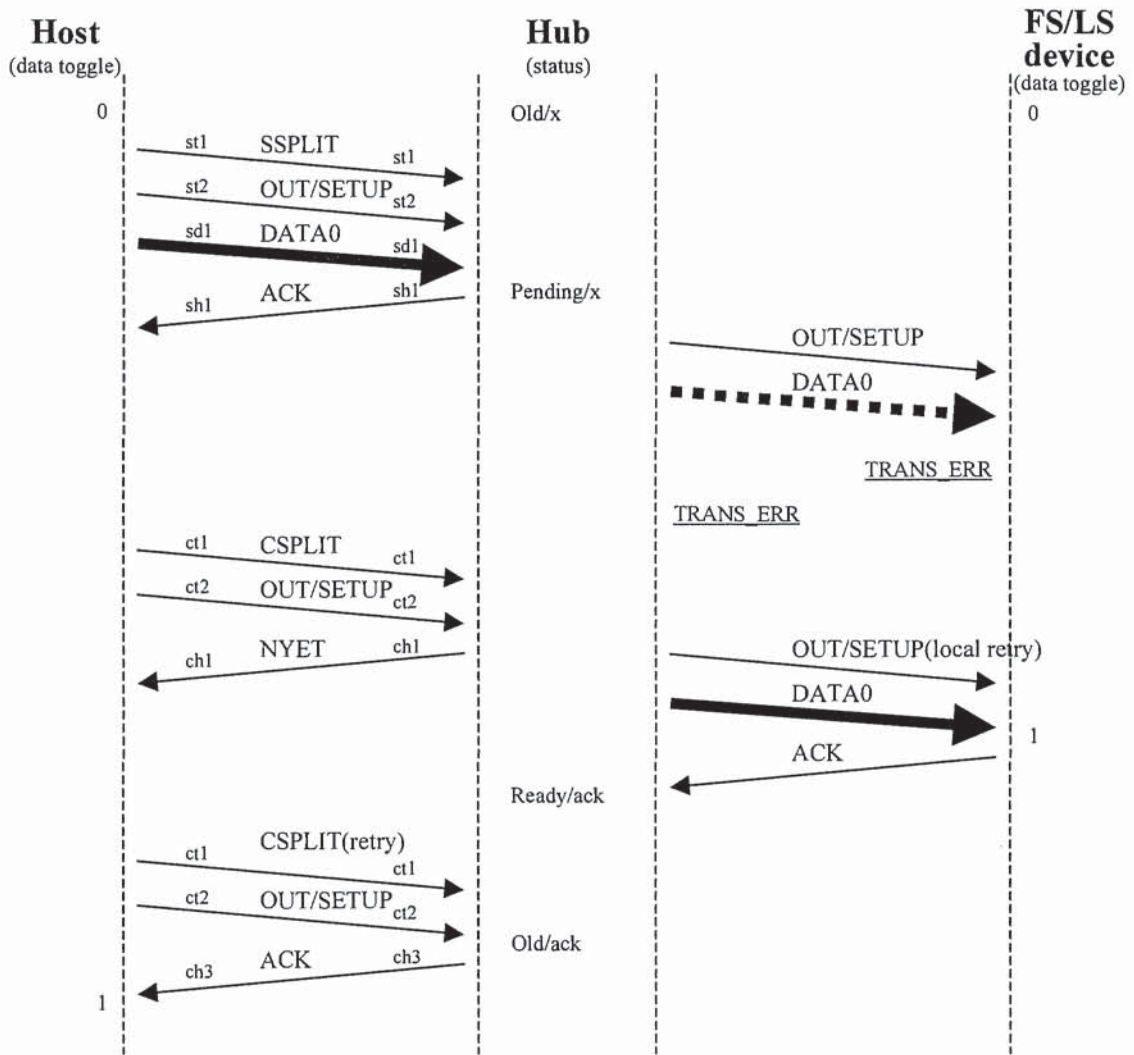


Figure A-11. Normal FS/LS DATA0/1 Smash

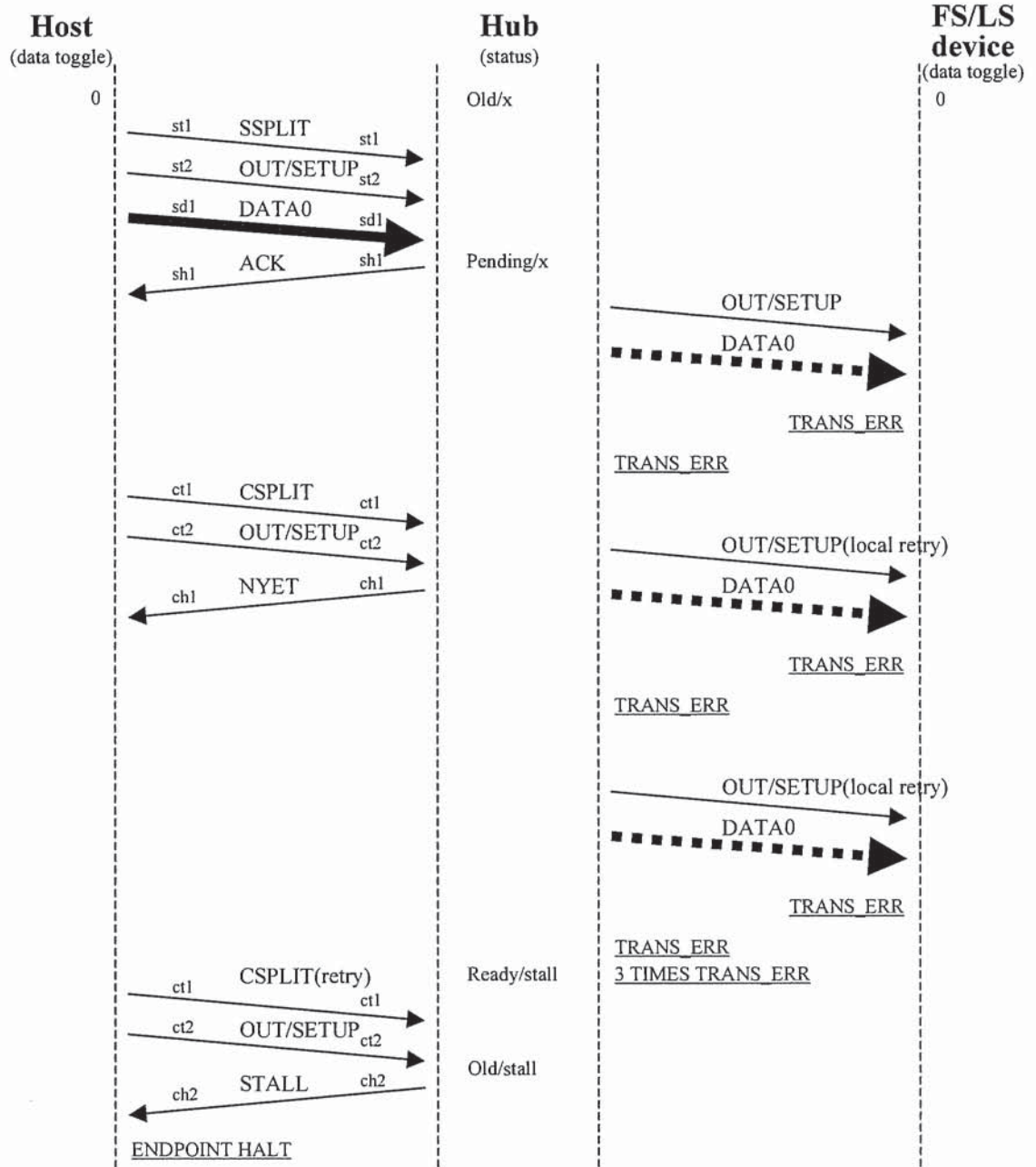


Figure A-12. Normal FS/LS DATA0/1 3 Strikes Smash

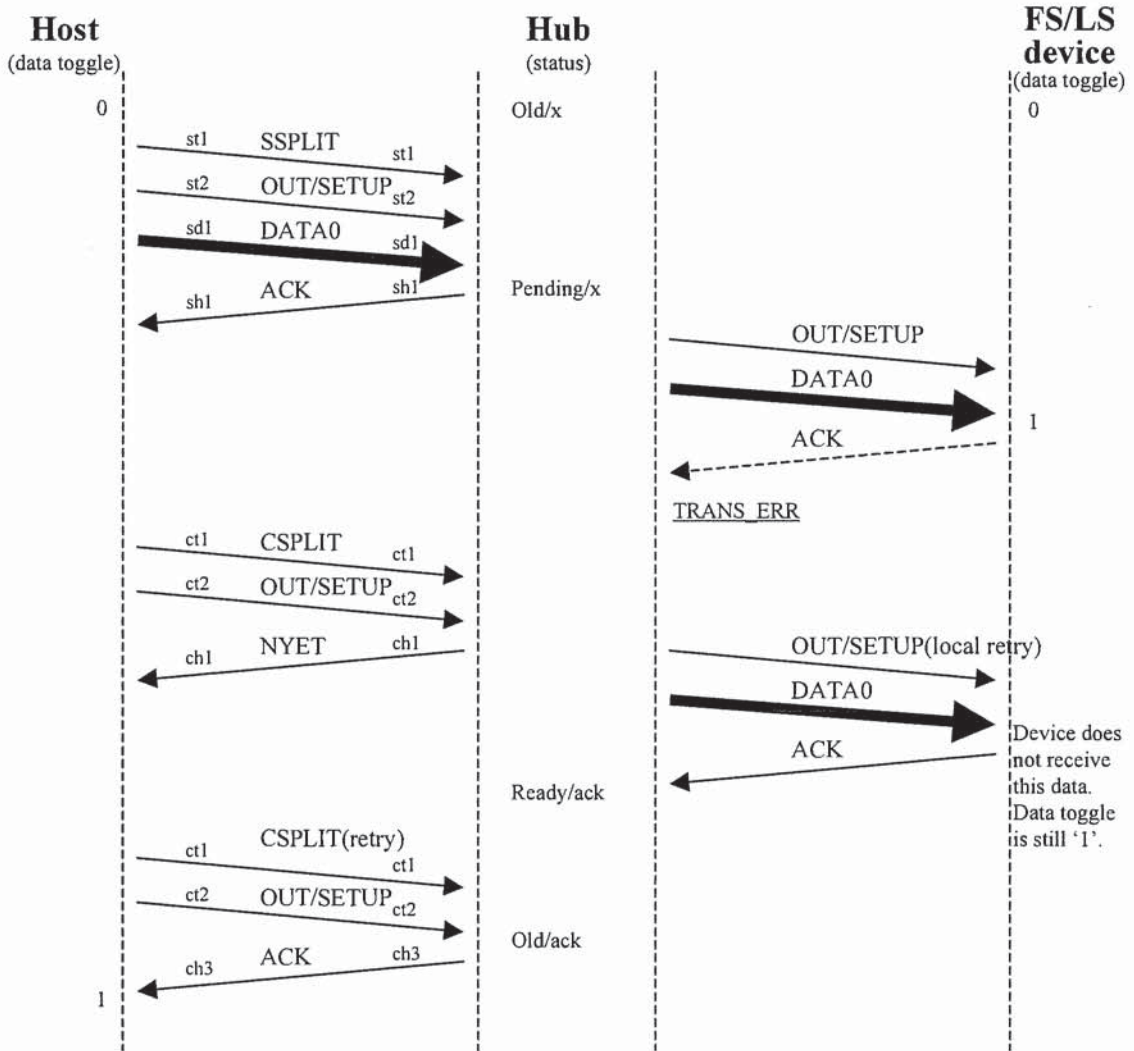


Figure A-13. Normal FS/LS ACK Smash

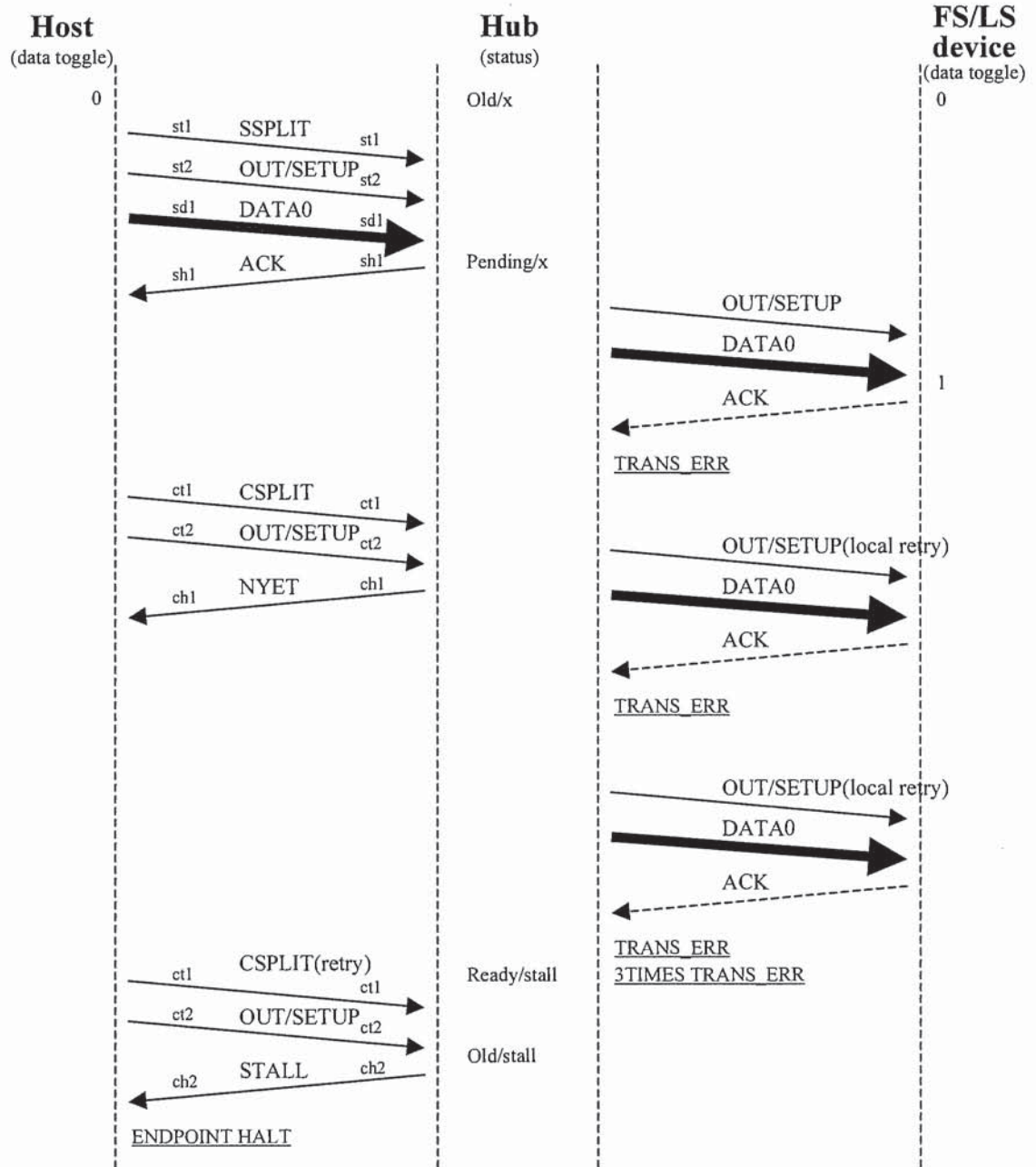


Figure A-14. Normal FS/LS ACK 3 Strikes Smash

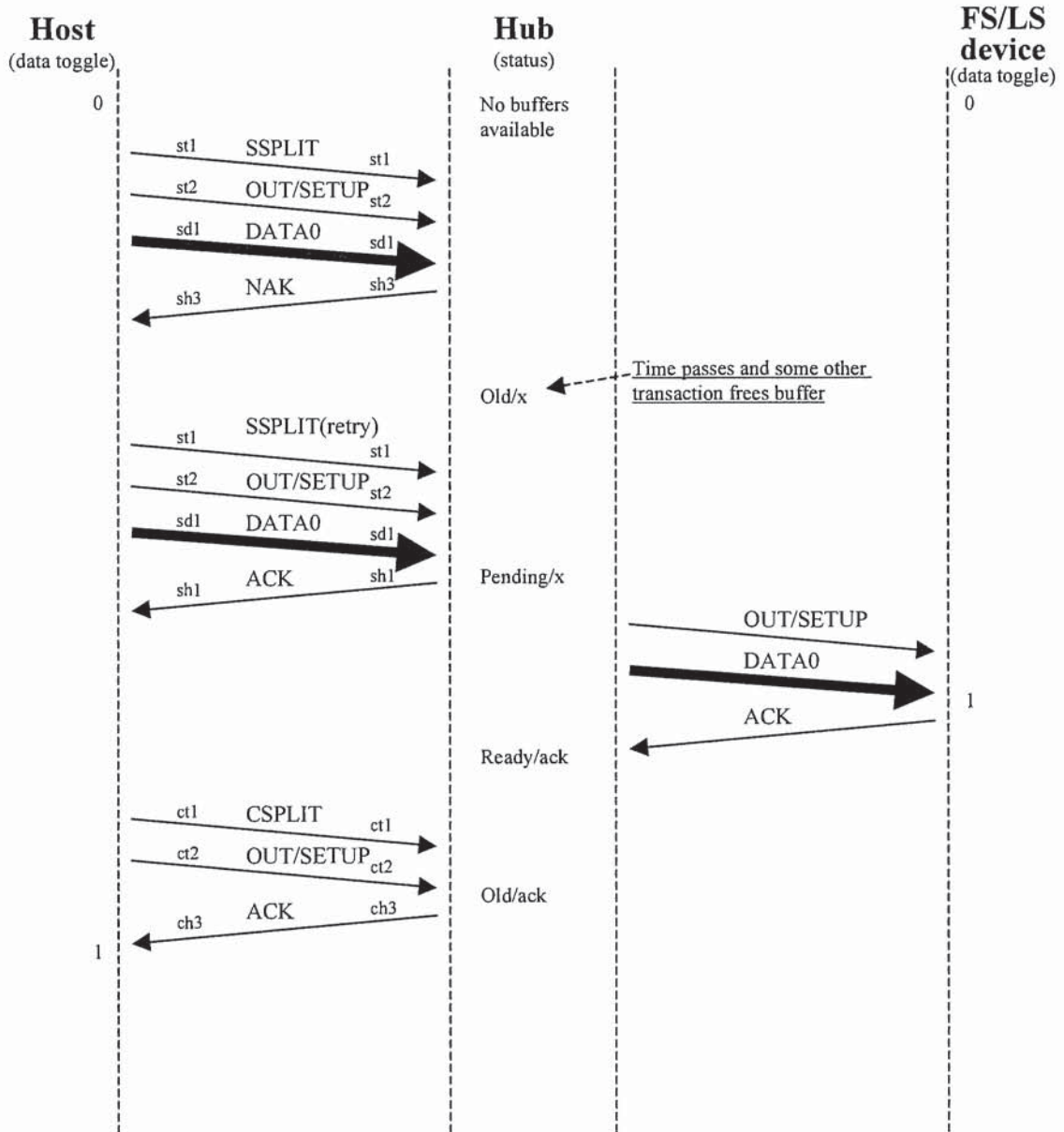


Figure A-15. No buffer Available No Smash (HS NAK(S))

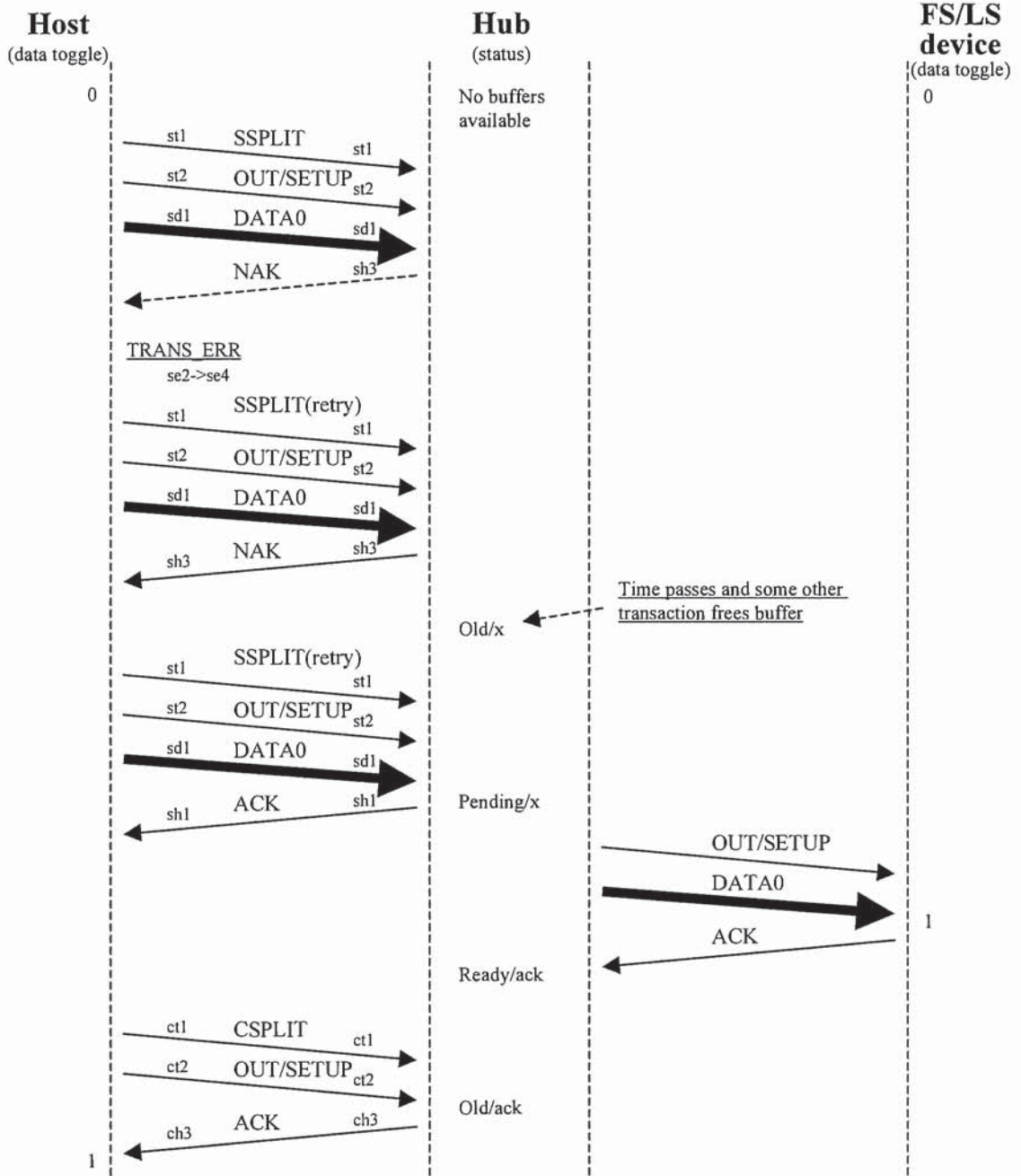


Figure A-16. No Buffer Available HS NAK(S) Smash

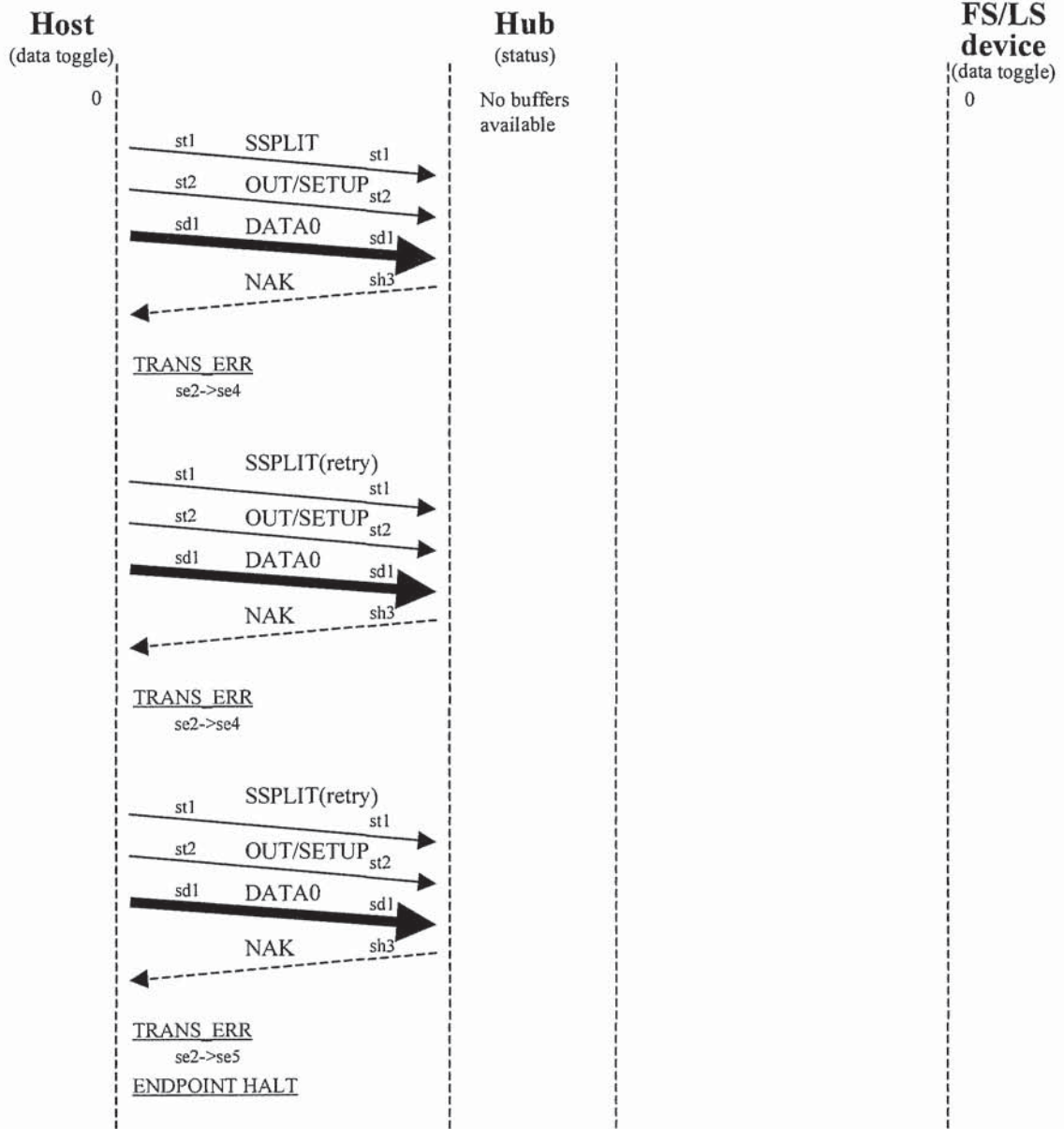


Figure A-17. No Buffer Available HS NAK(S) 3 Strikes Smash

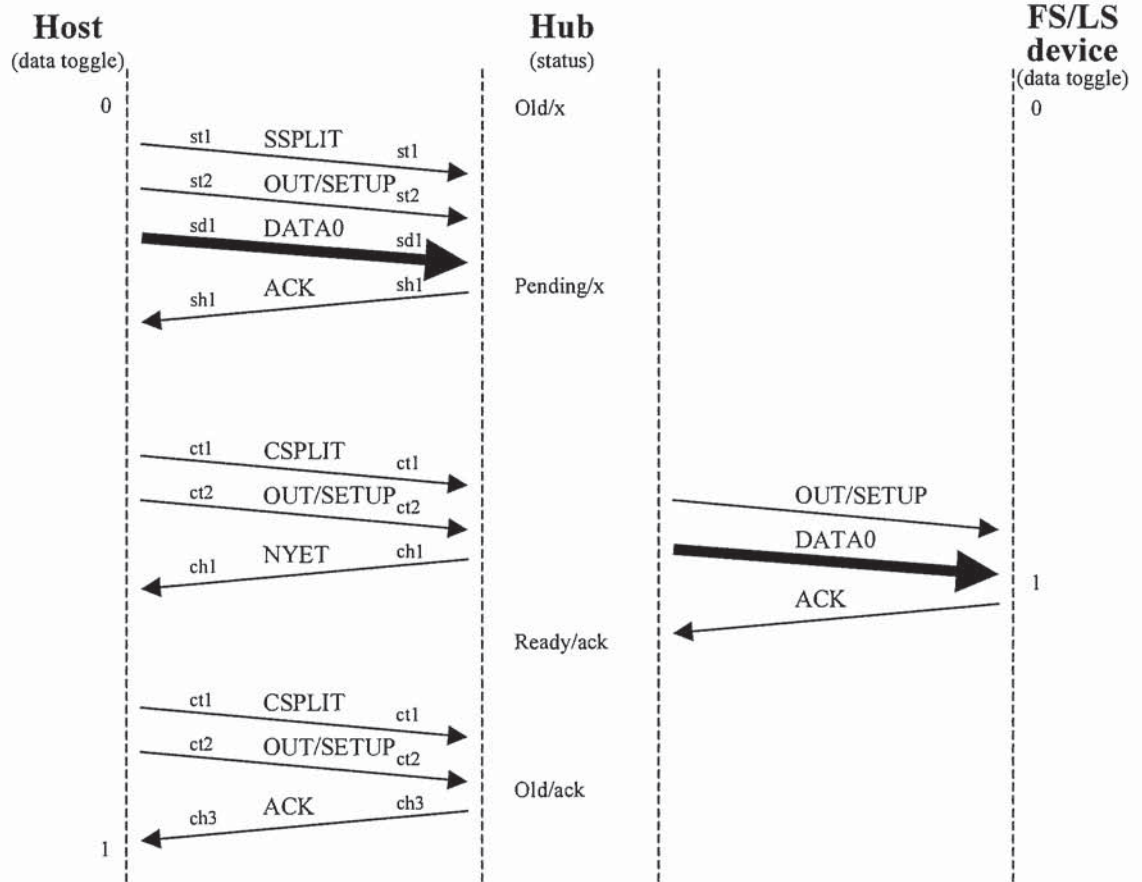


Figure A-18. CS Earlier No Smash (HS NYET)

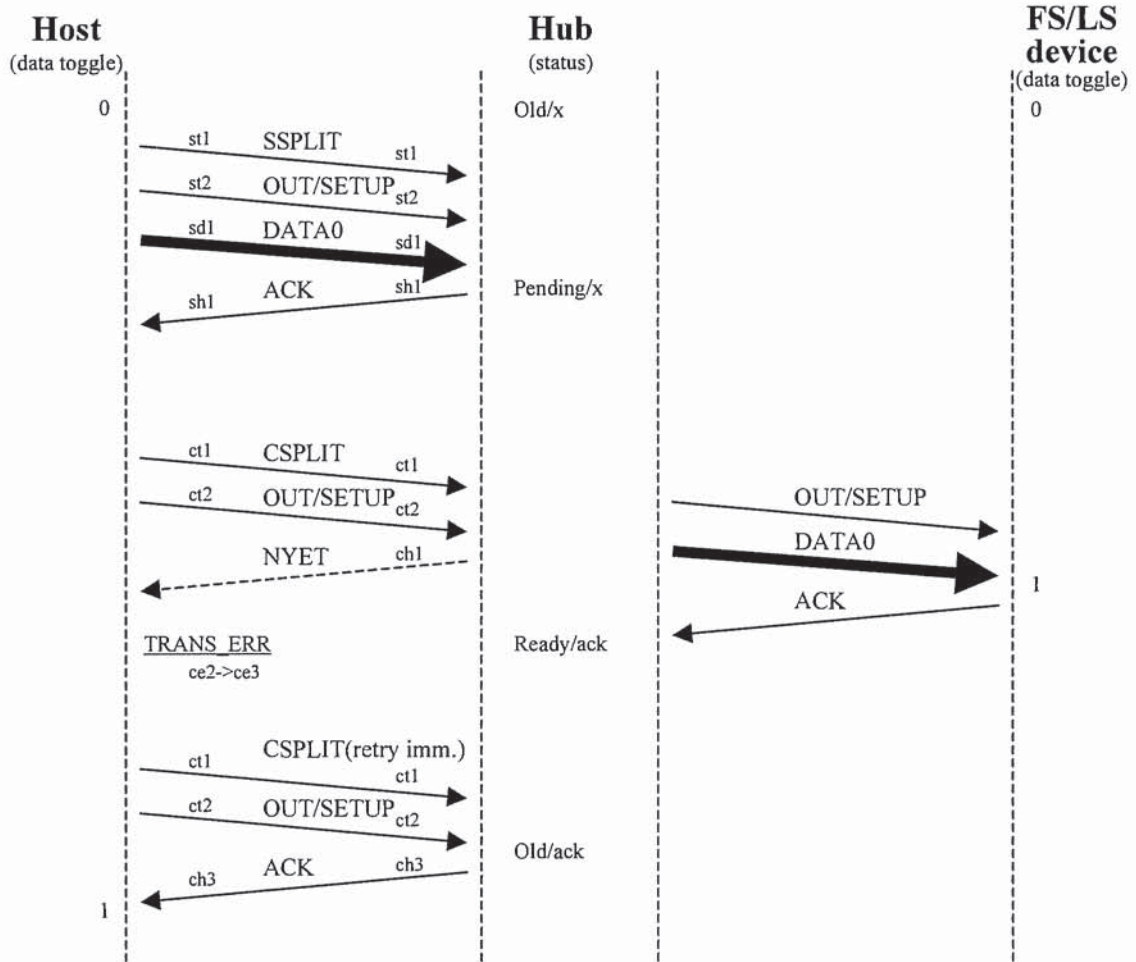


Figure A-19. CS Earlier HS NYET Smash(case 1)

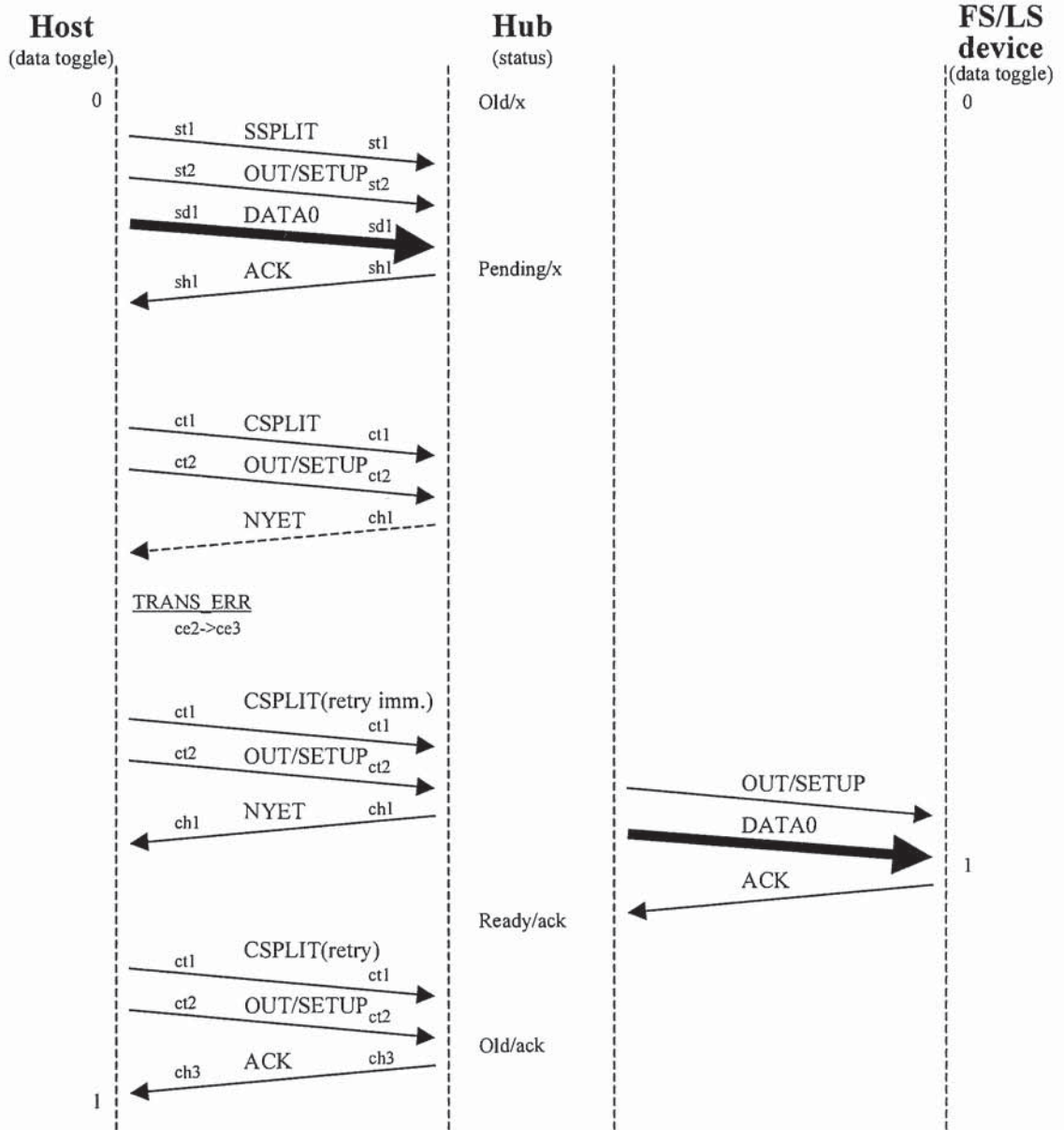


Figure A-20. CS Earlier HS NYET Smash(case 2)

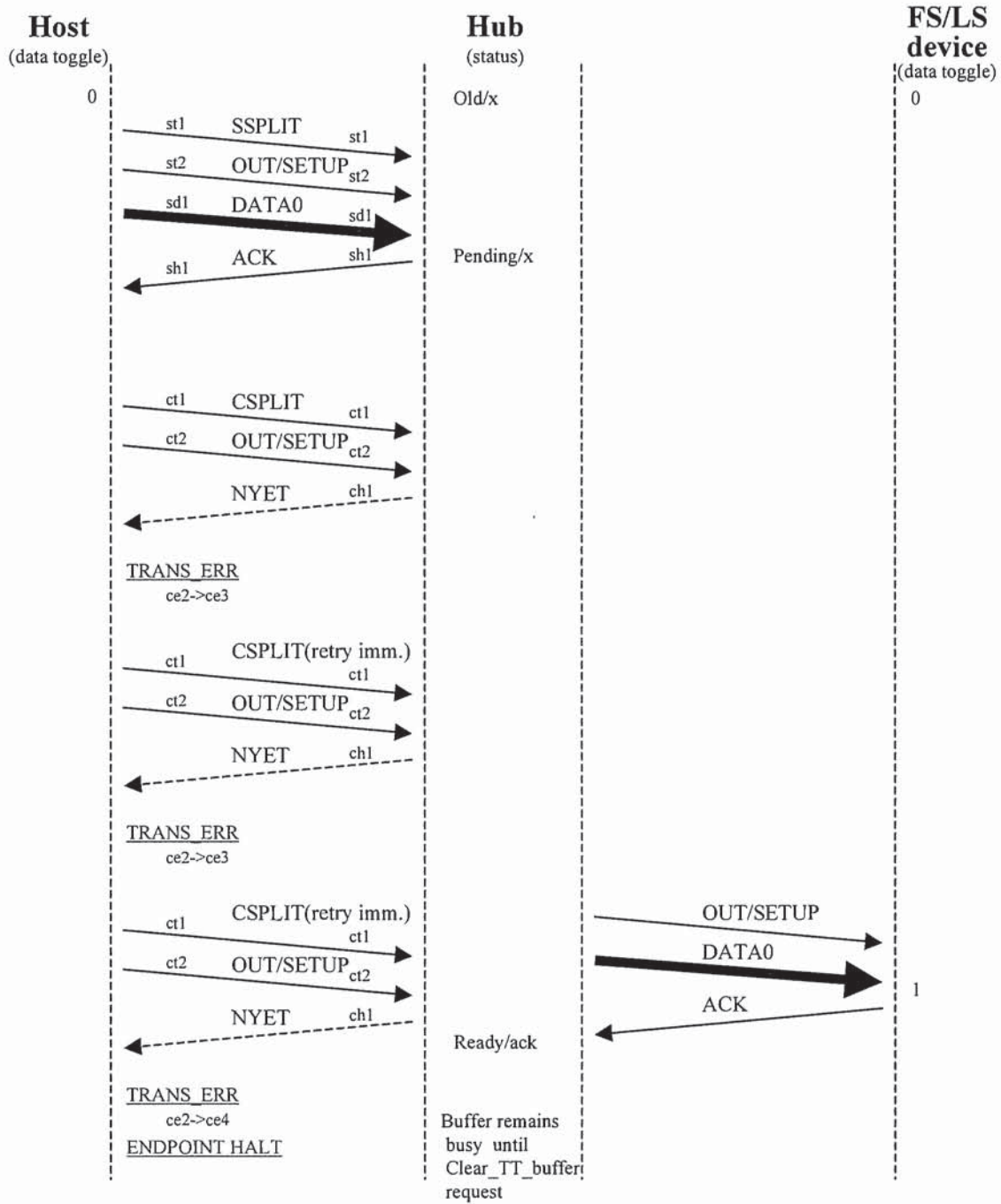


Figure A-21. CS Earlier HS NYET 3 Strikes Smash

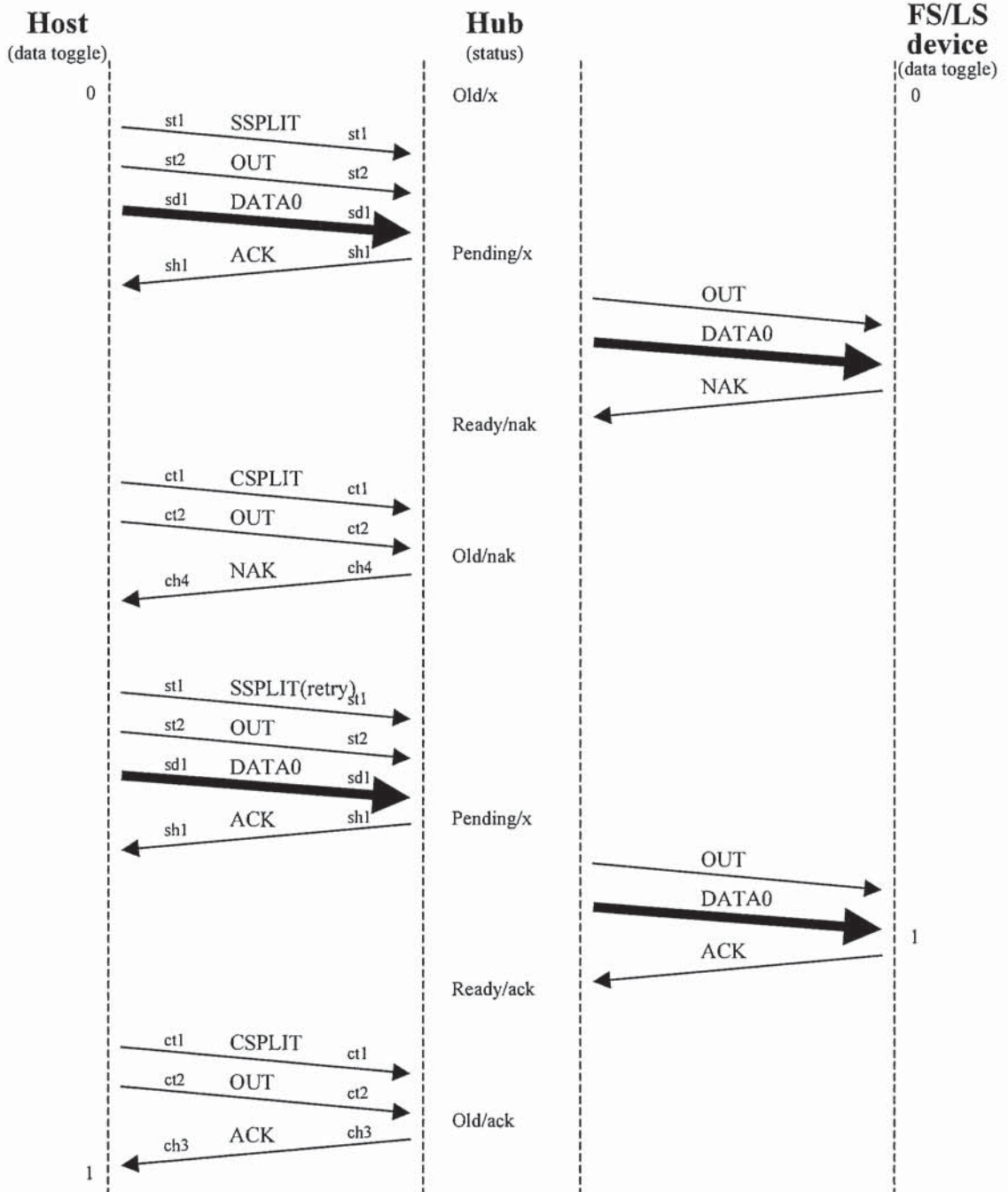


Figure A-22. Device Busy No Smash(FS/LS NAK)

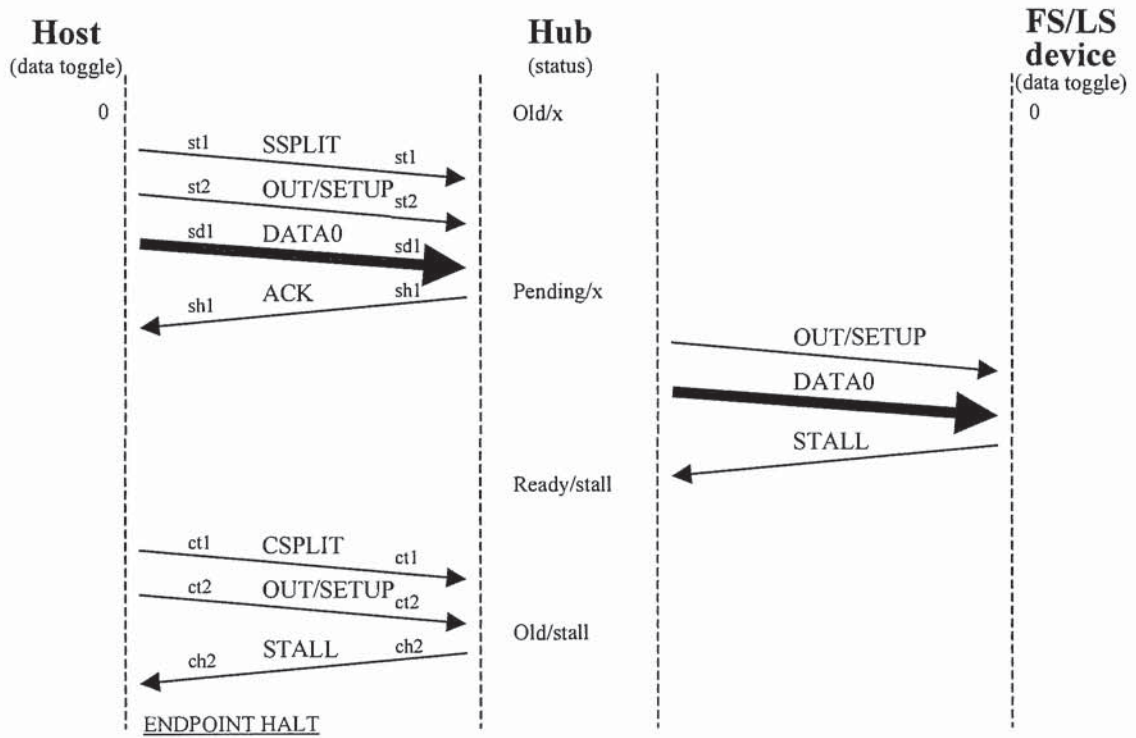


Figure A-23. Device Stall No Smash(FS/LS STALL)

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	

Universal Serial Bus Specification Revision 2.0

FS/LS ACK smash	Figure A-38	
FS/LS ACK 3 strikes smash	No figure	

- No buffer(on hub) available 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

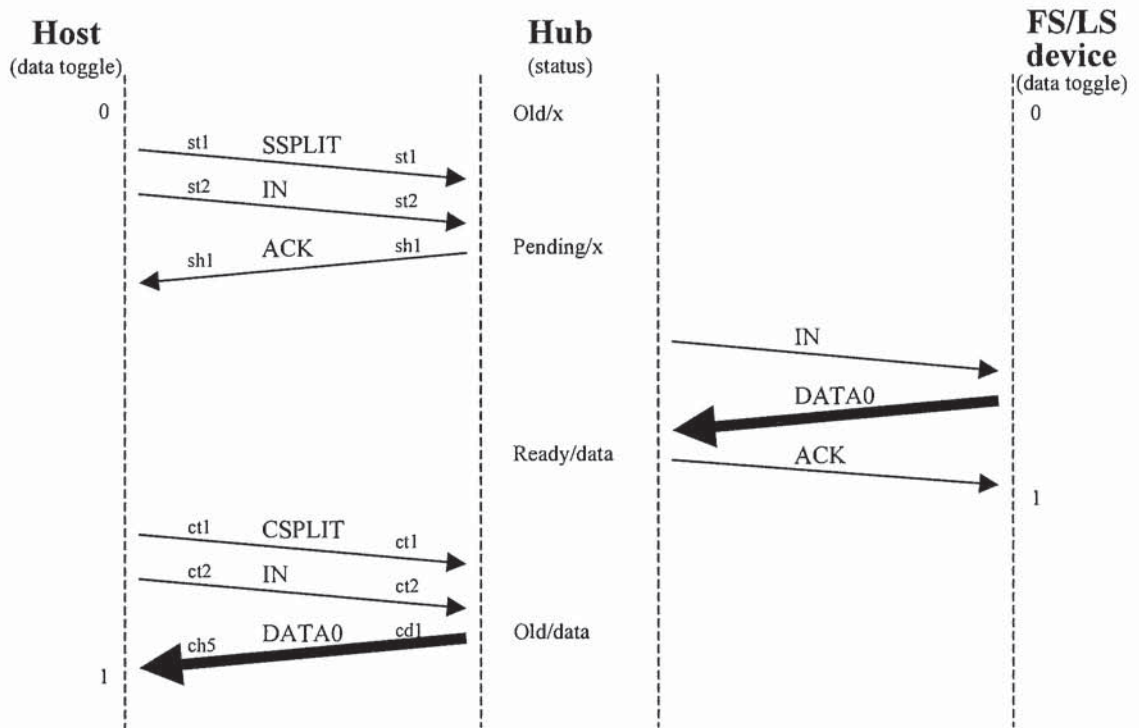


Figure A-24. Normal No Smash

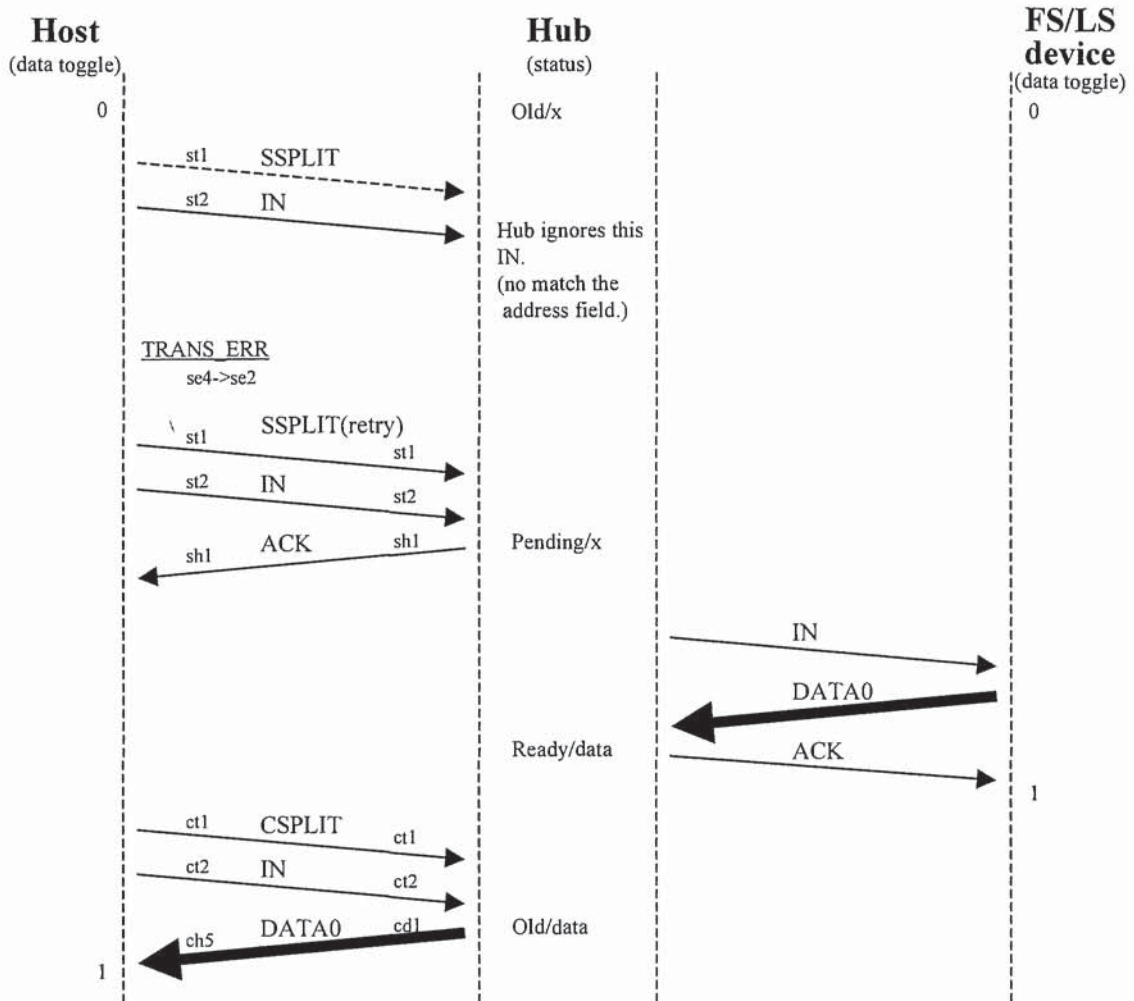


Figure A-25. Normal HS SSPLIT Smash

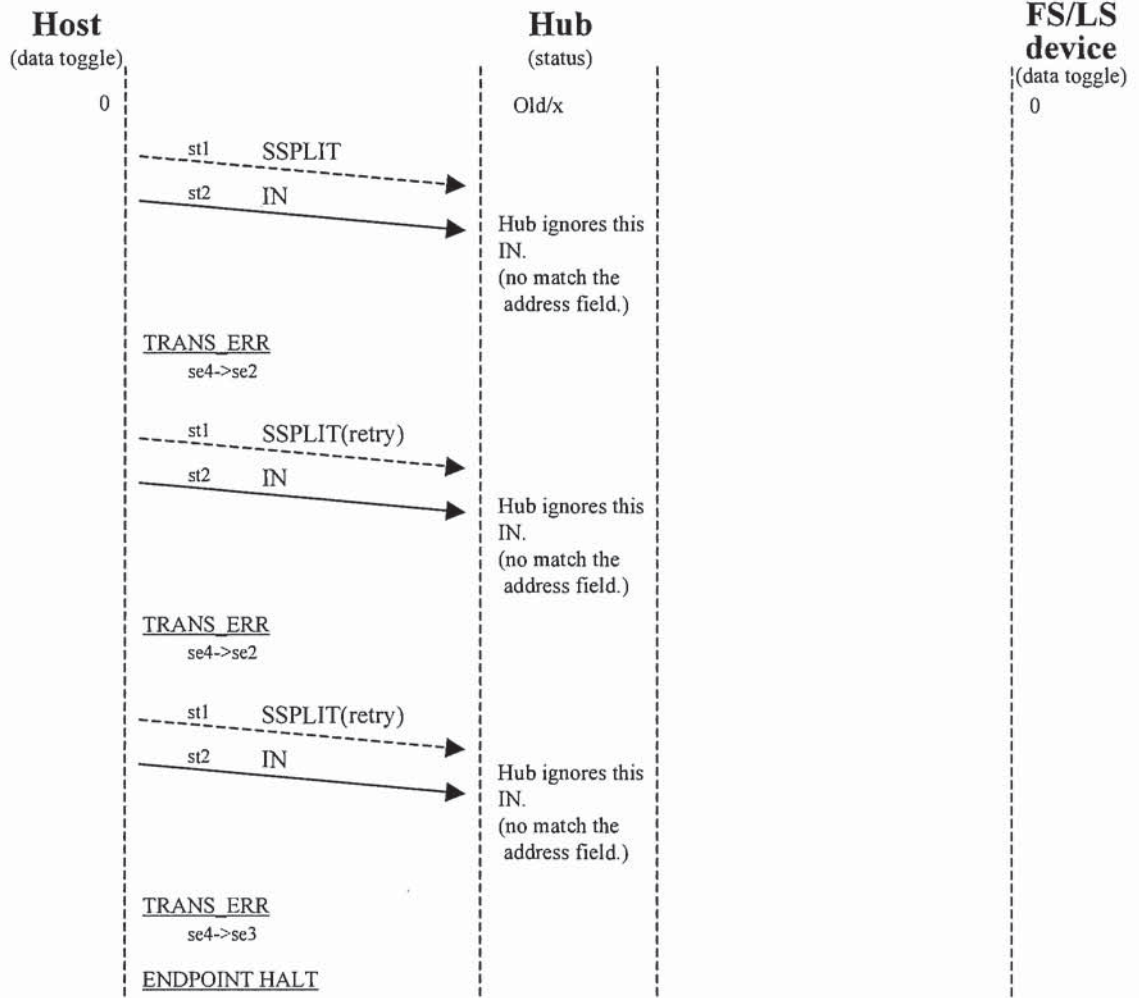


Figure A-26. Normal SSPLIT 3 Strikes Smash

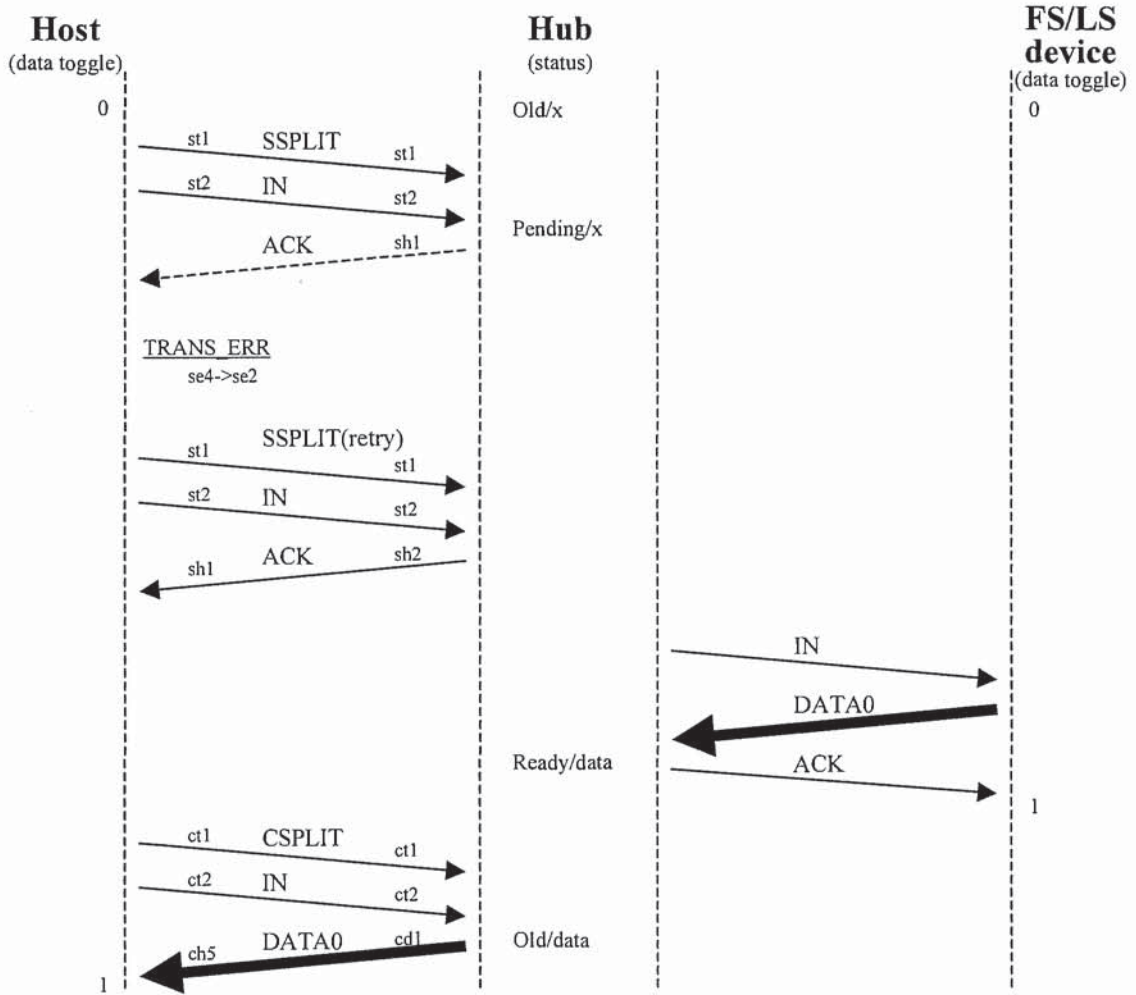


Figure A-27. Normal HS ACK(S) Smash(case 1)

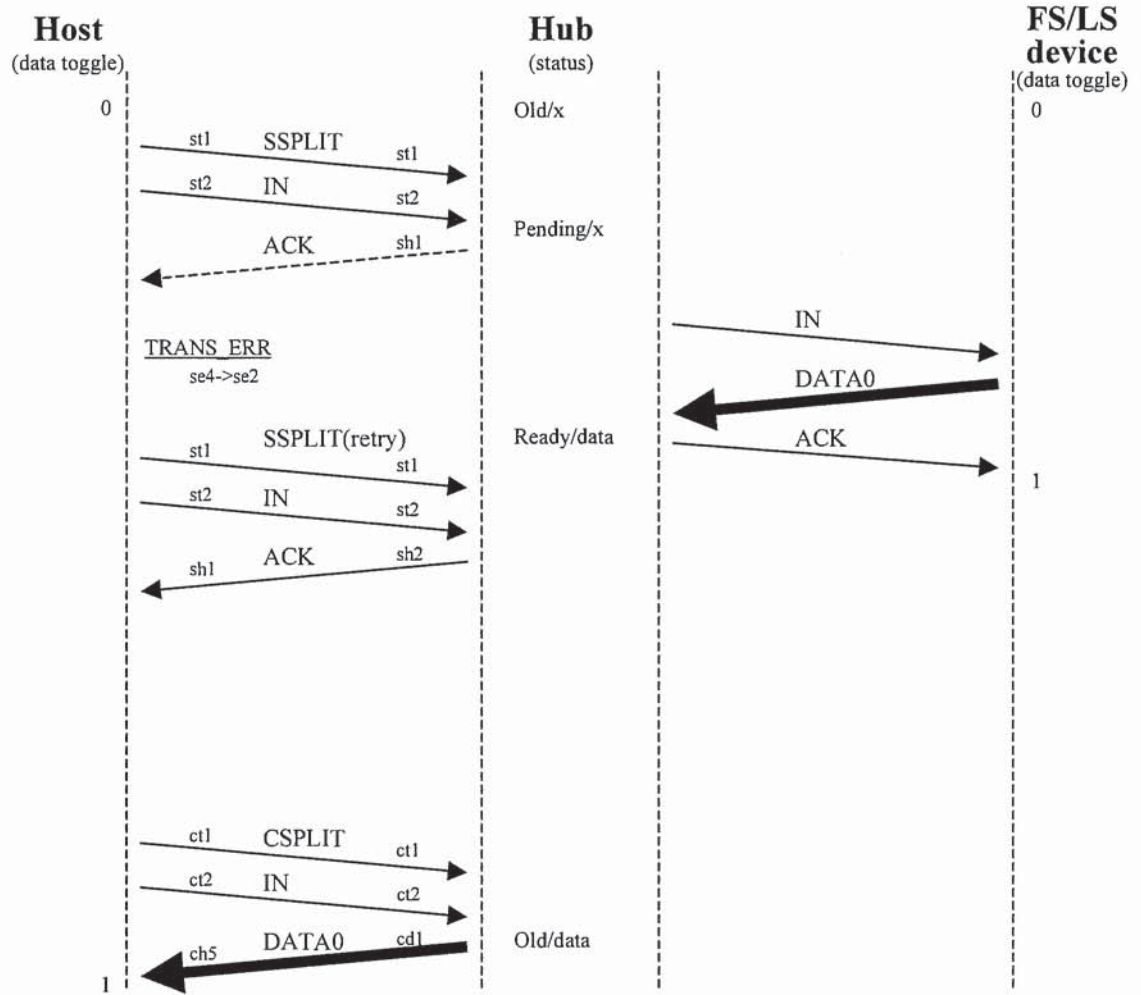


Figure A-28. Normal HS ACK(S) Smash(case 2)

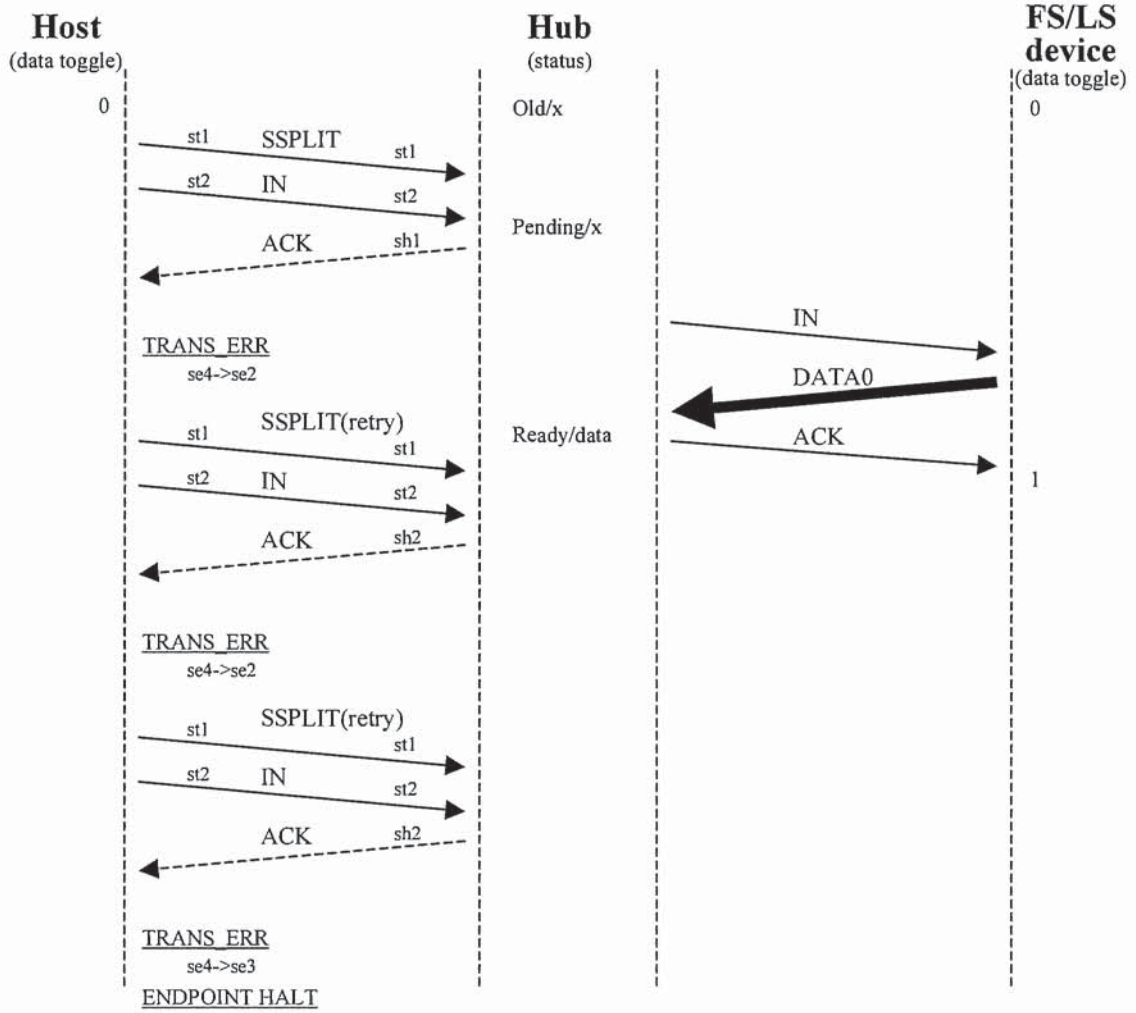


Figure A-29. Normal HS ACK(S) 3 Strikes Smash

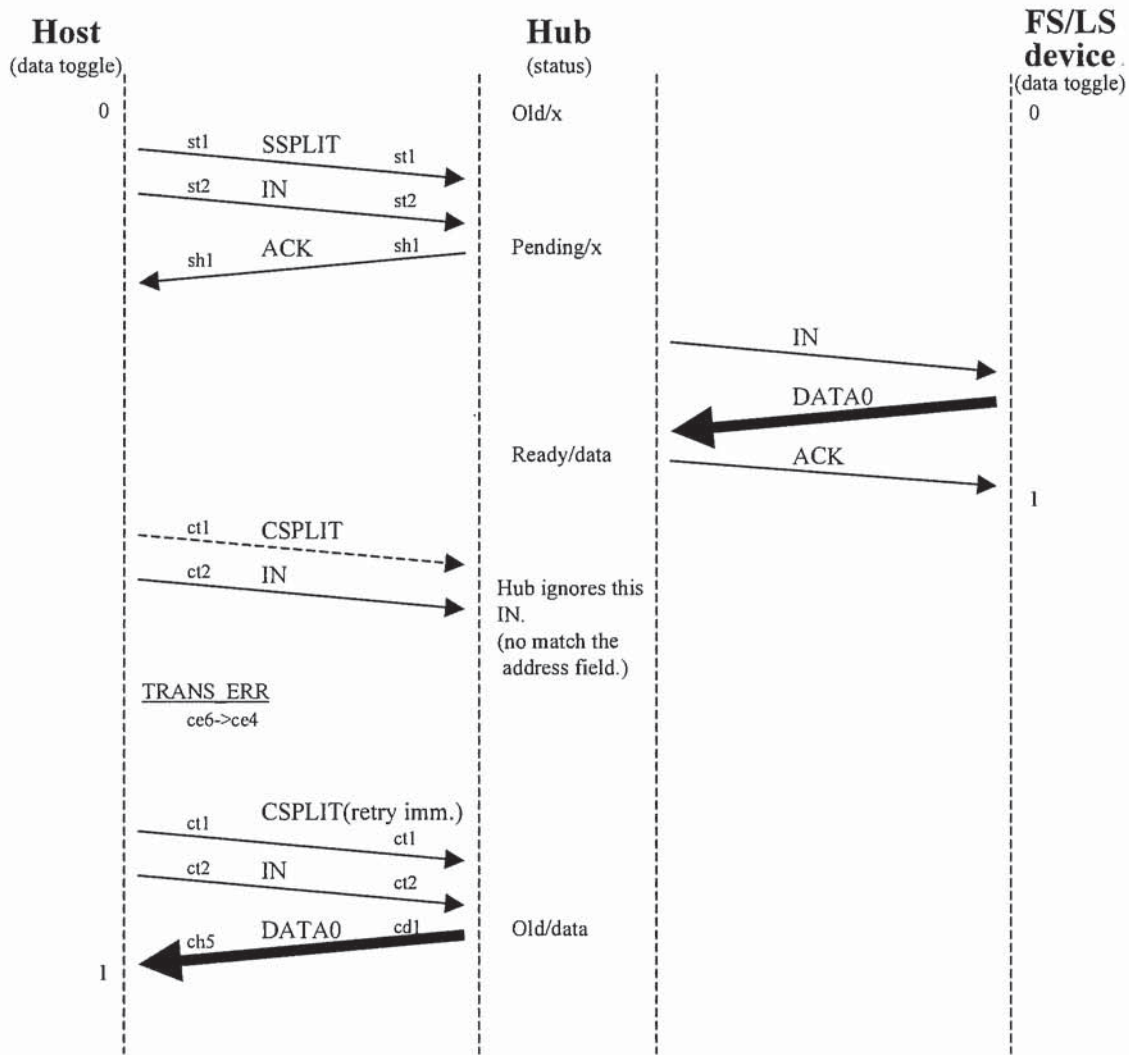


Figure A-30. Normal HS CSPLIT Smash

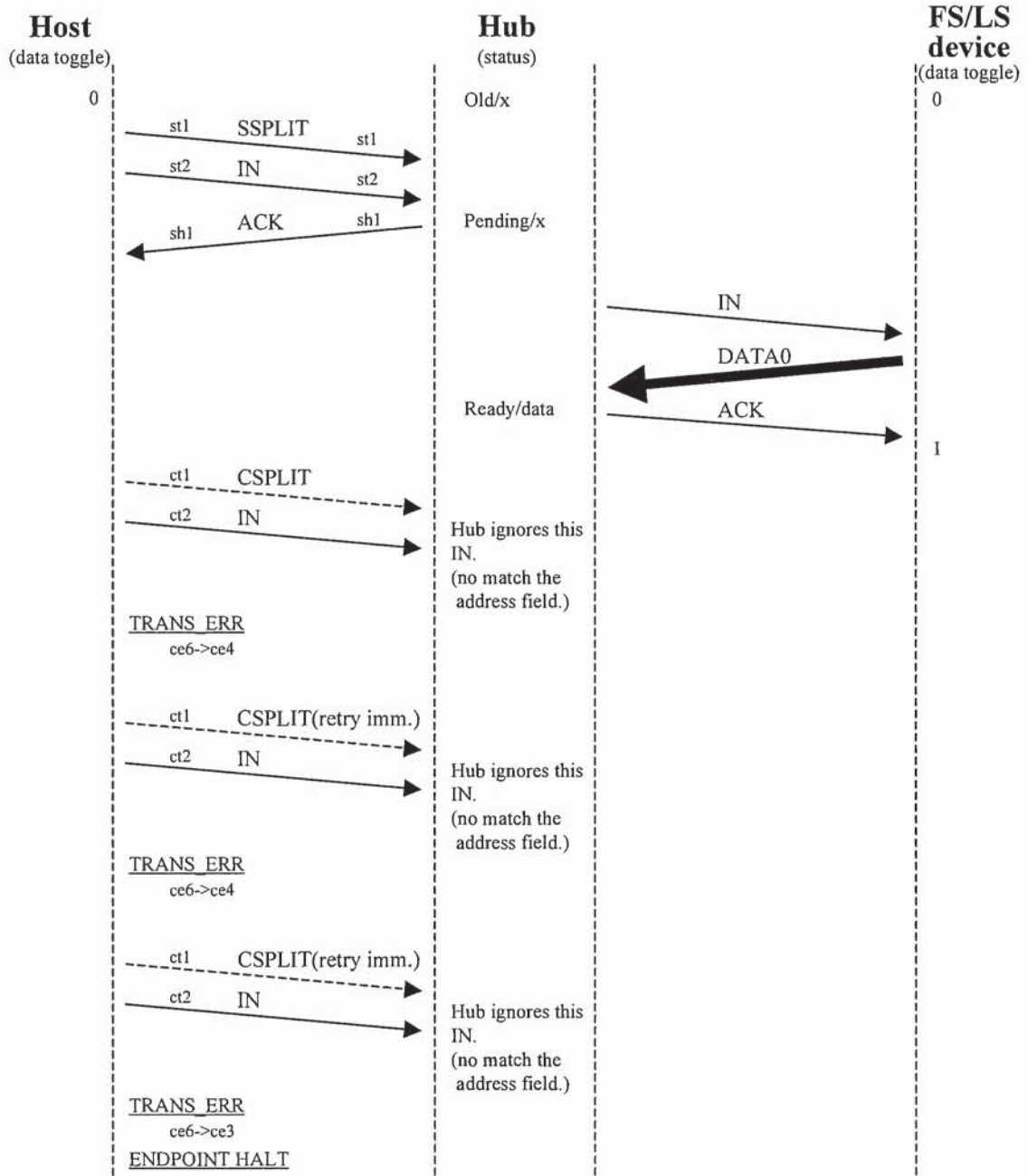


Figure A-31. Normal HS CSPLIT 3 Strikes Smash

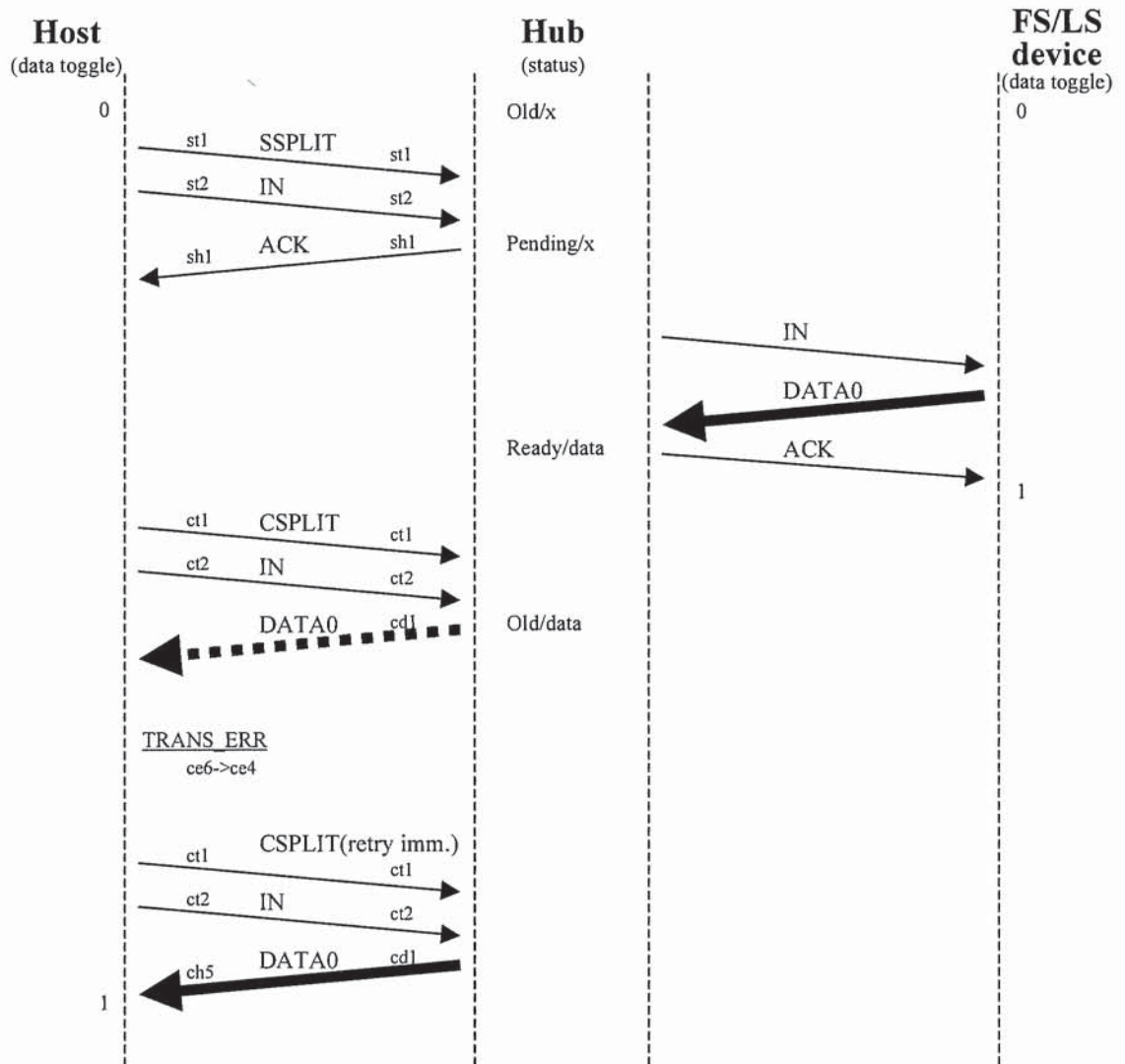


Figure A-32. Normal HS DATA0/1 Smash

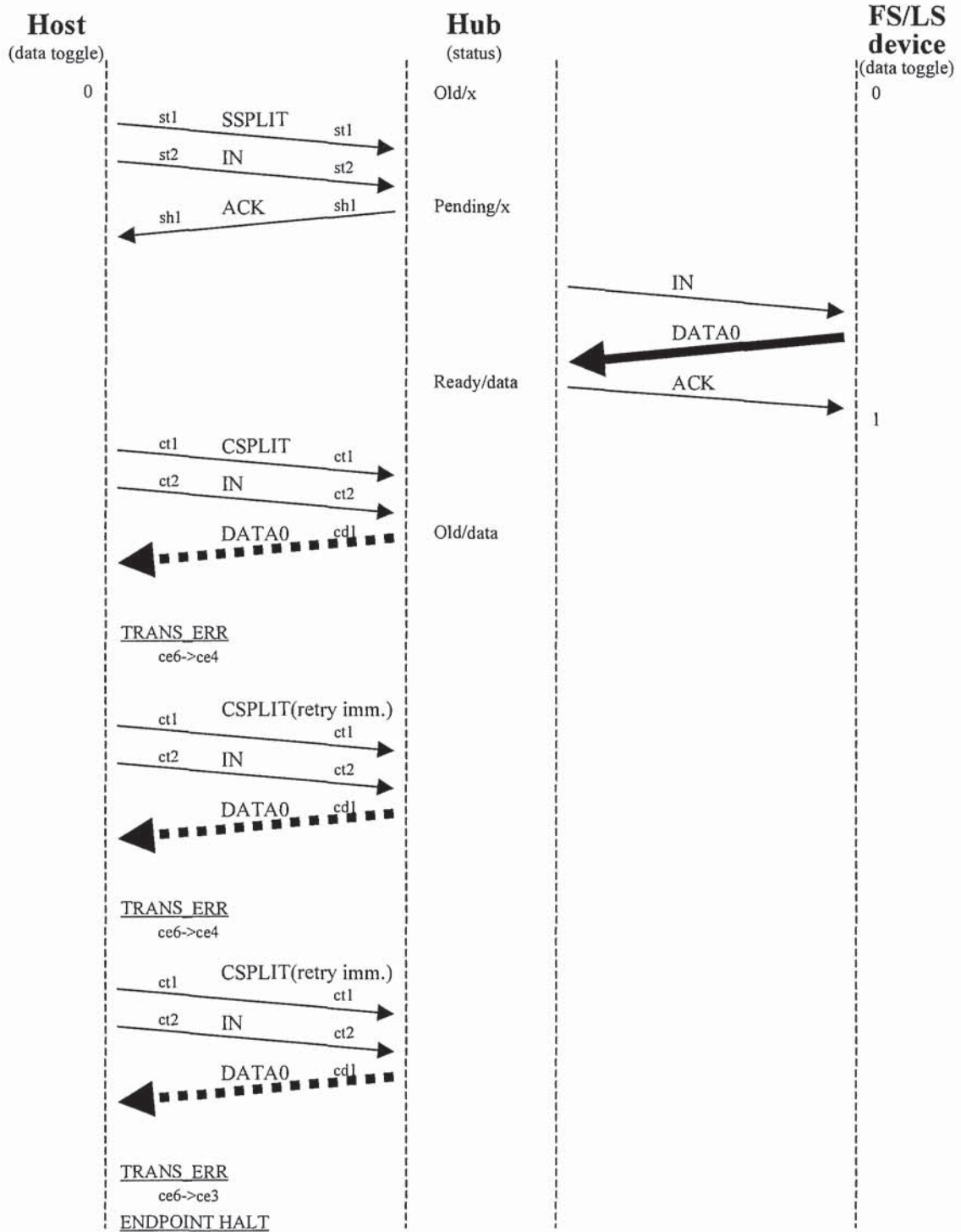


Figure A-33. Normal HS DATA0/1 3 Strikes Smash

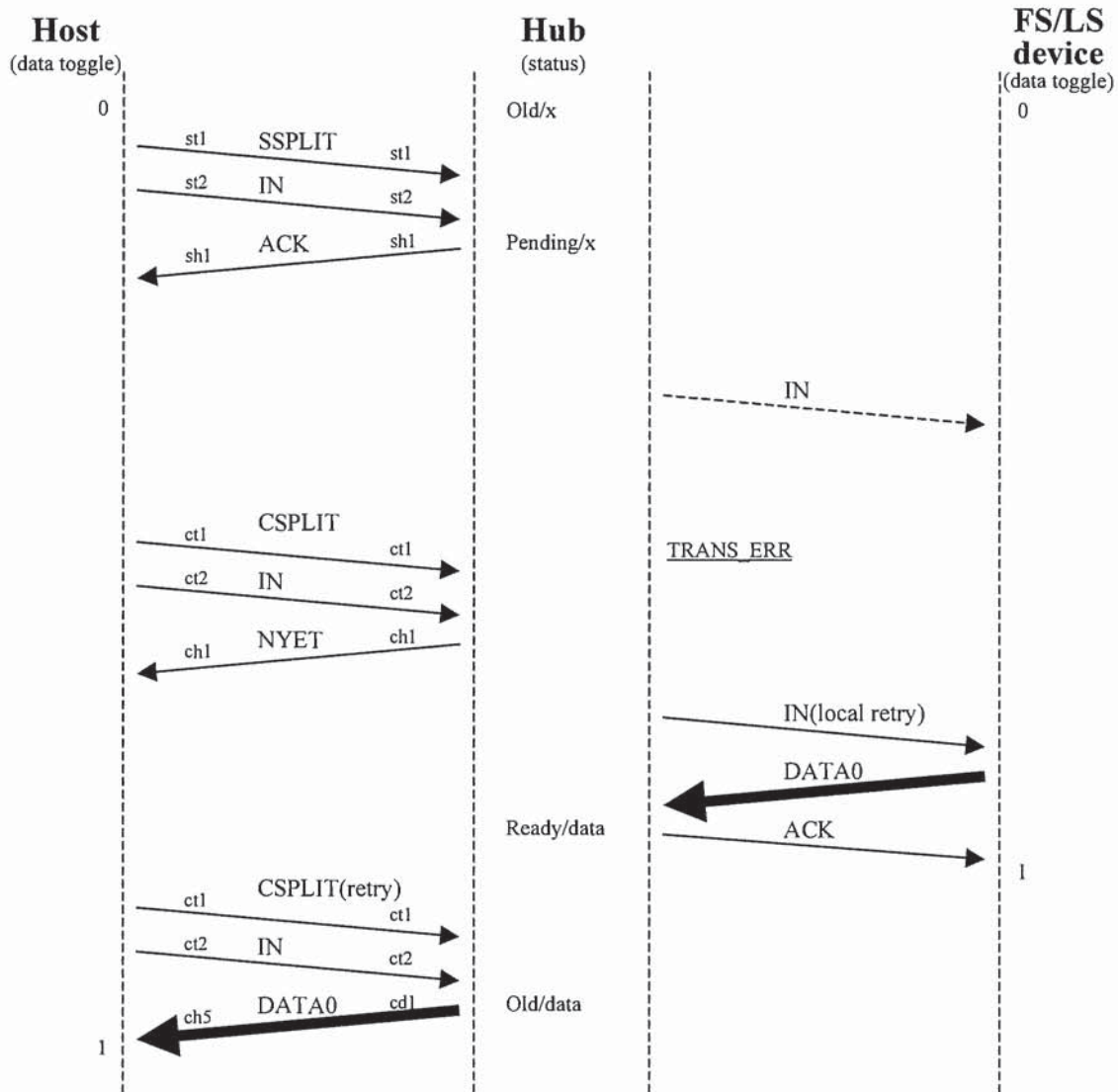


Figure A-34. Normal FS/LS IN Smash

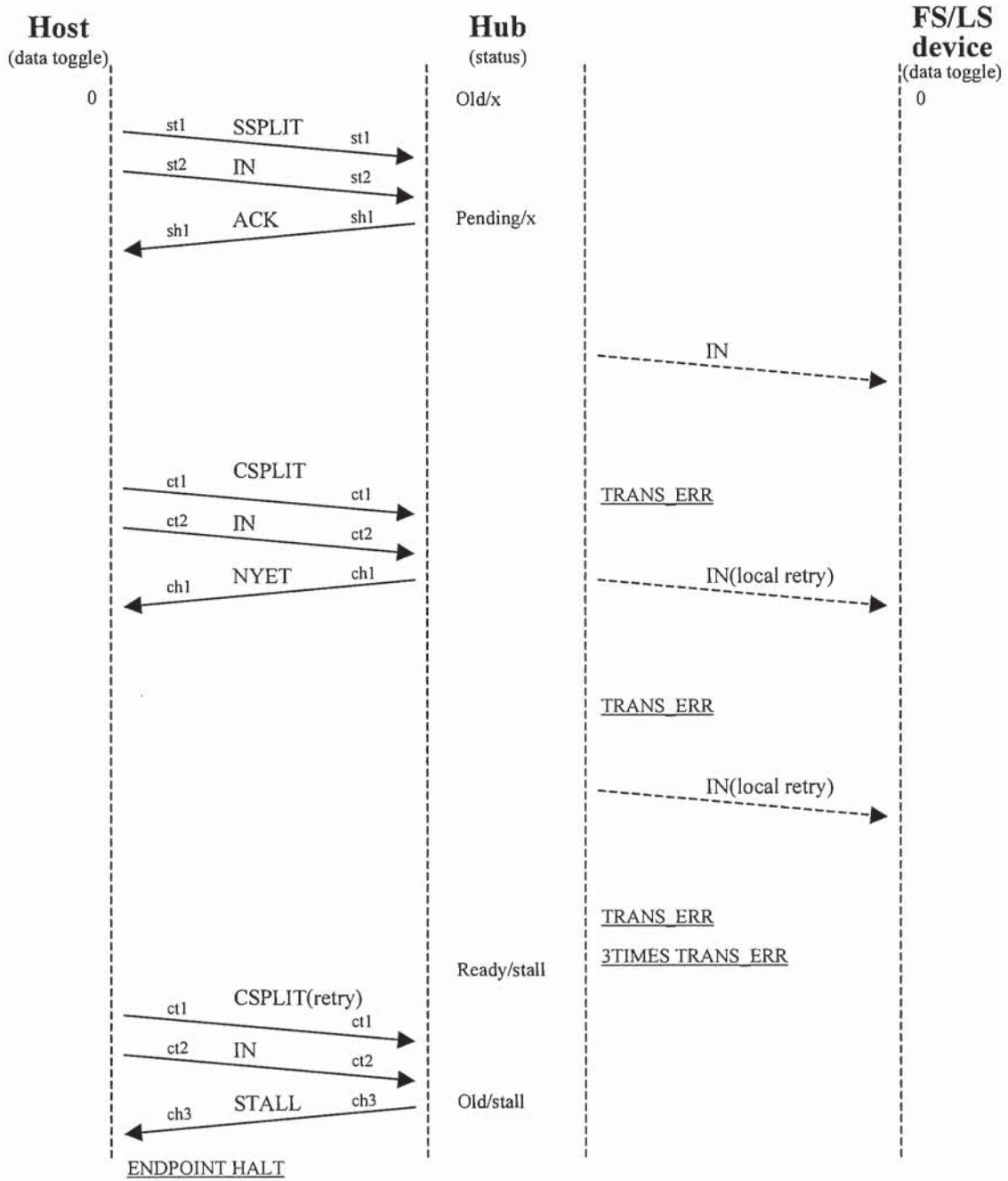


Figure A-35. Normal FS/LS IN 3 Strikes Smash

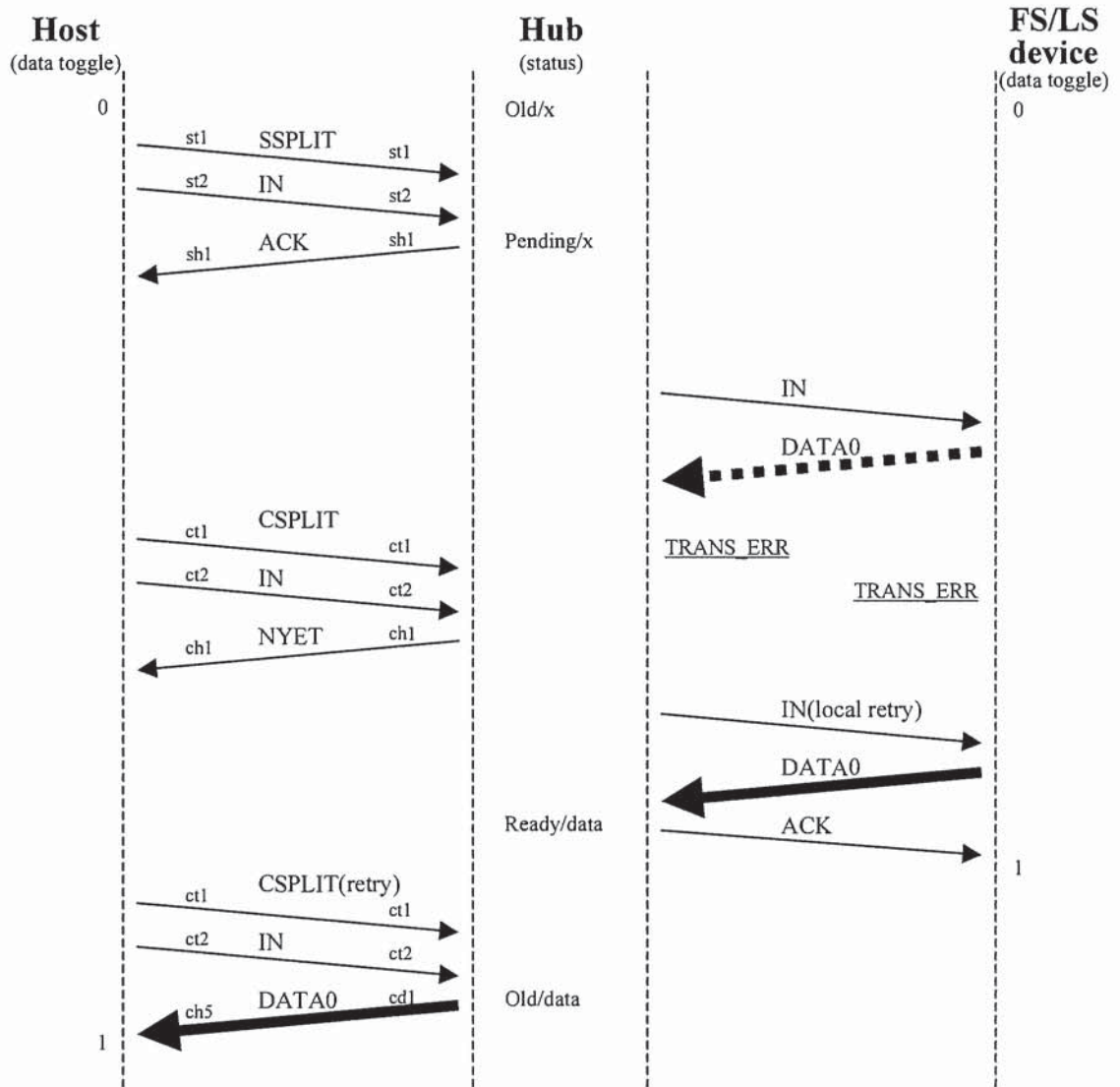


Figure A-36. Normal FS/LS DATA0/1 Smash

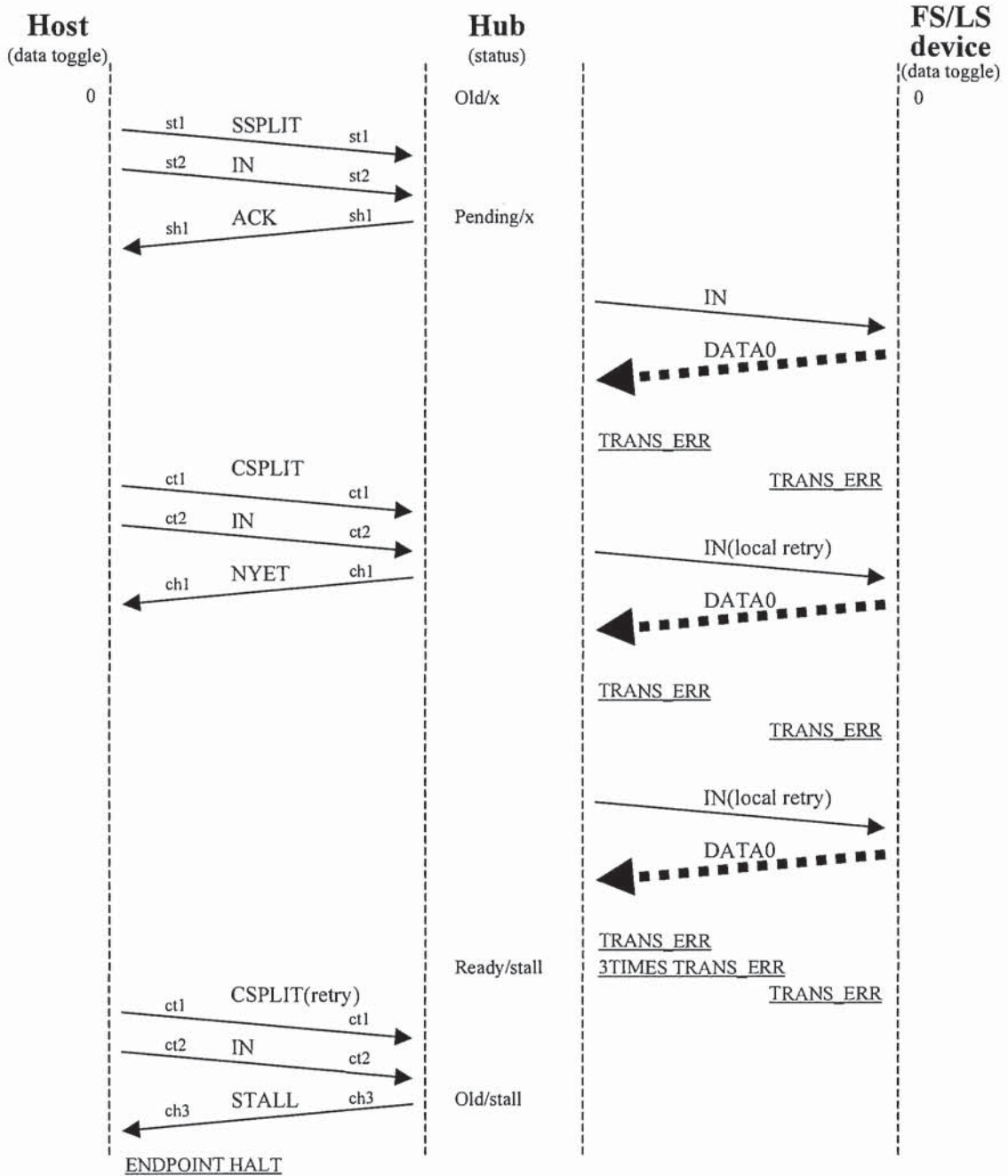


Figure A-37. Normal FS/LS DATA0/1 3 Strikes Smash

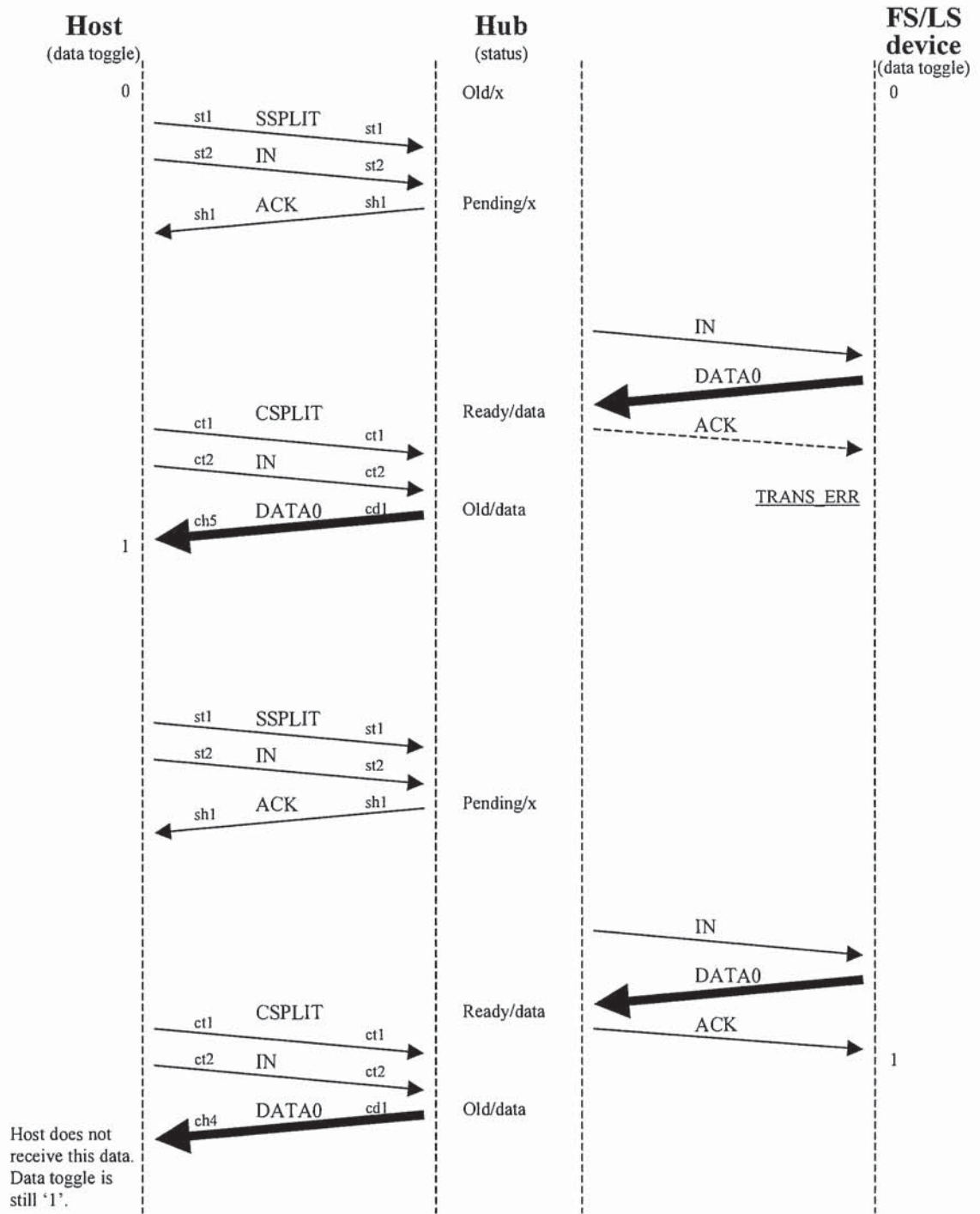


Figure A-38. Normal FS/LS ACK Smash

Universal Serial Bus Specification Revision 2.0

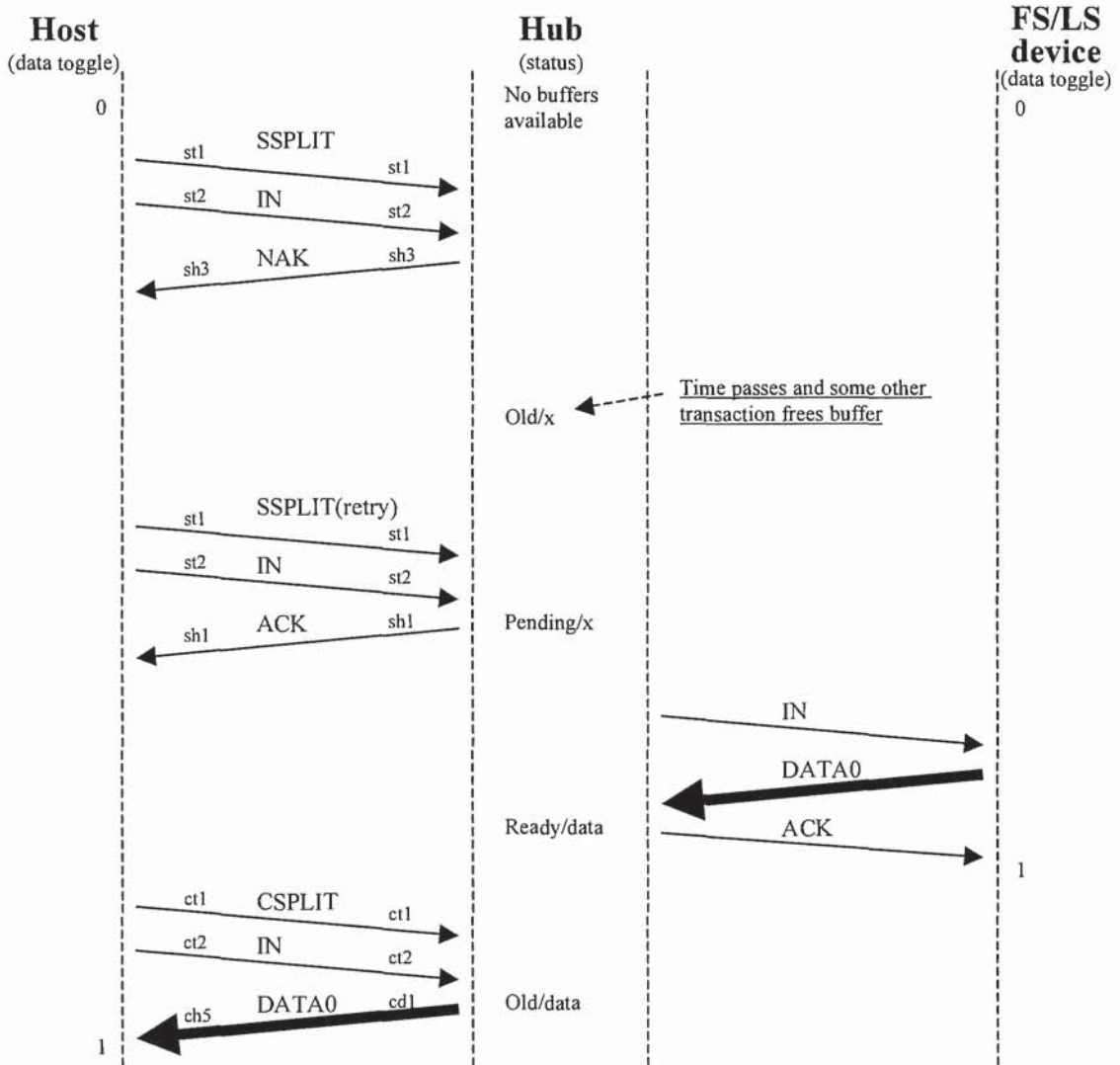


Figure A-39. No Buffer Available No Smash(HS NAK(S))

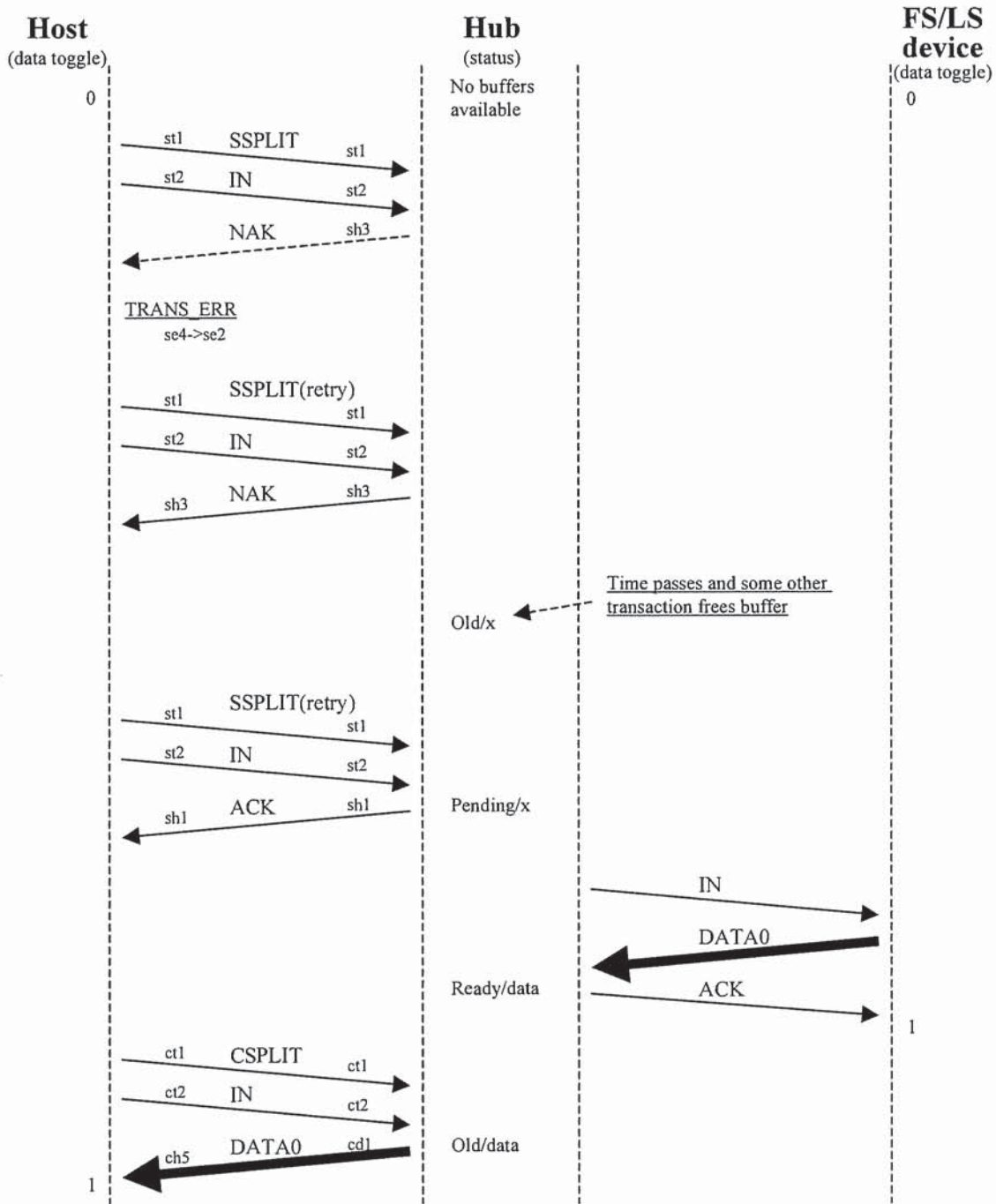


Figure A-40. No Buffer Available HS NAK(S) Smash

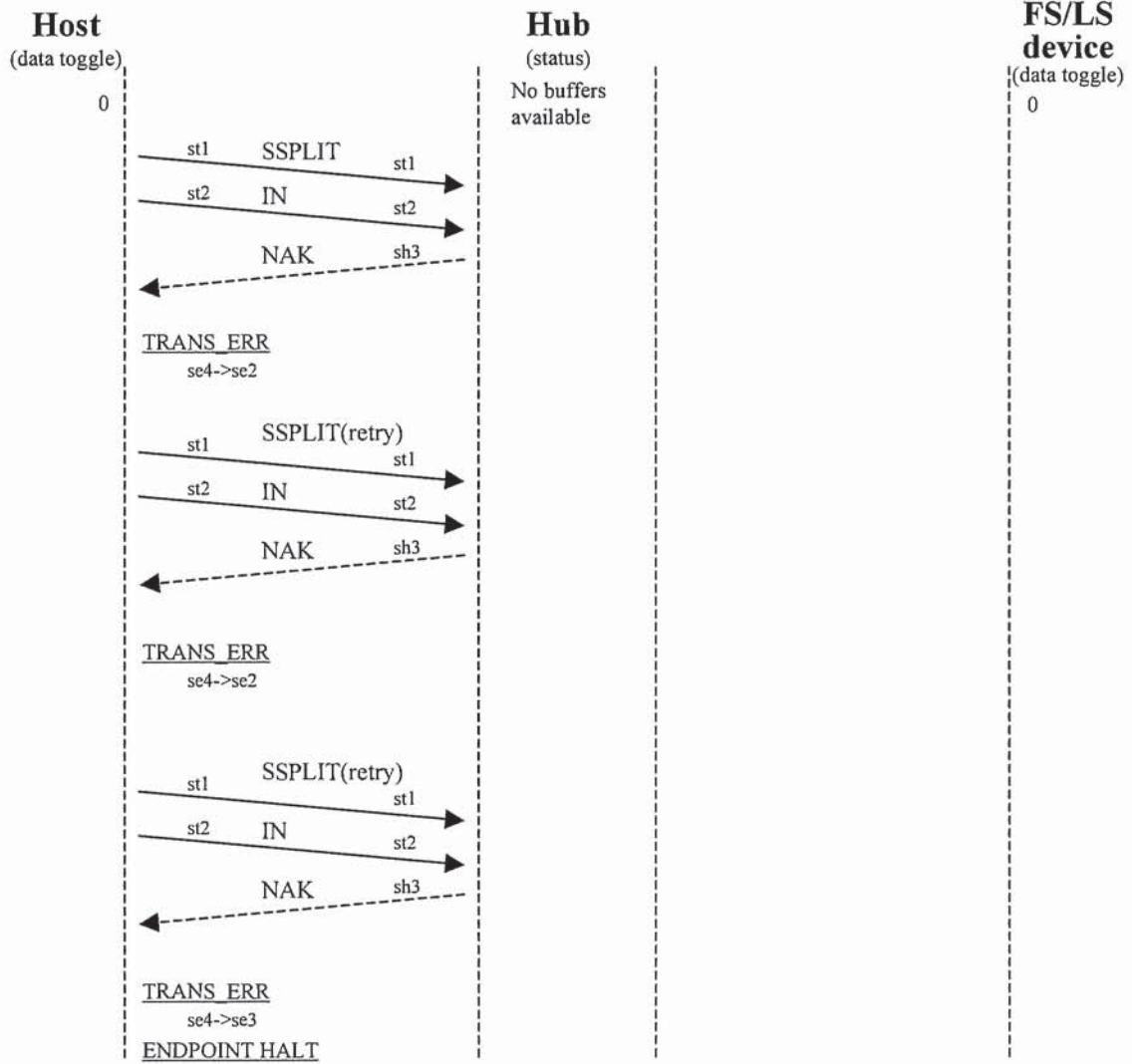


Figure A-41. No Buffer Available HS NAK(S) 3 Strikes Smash

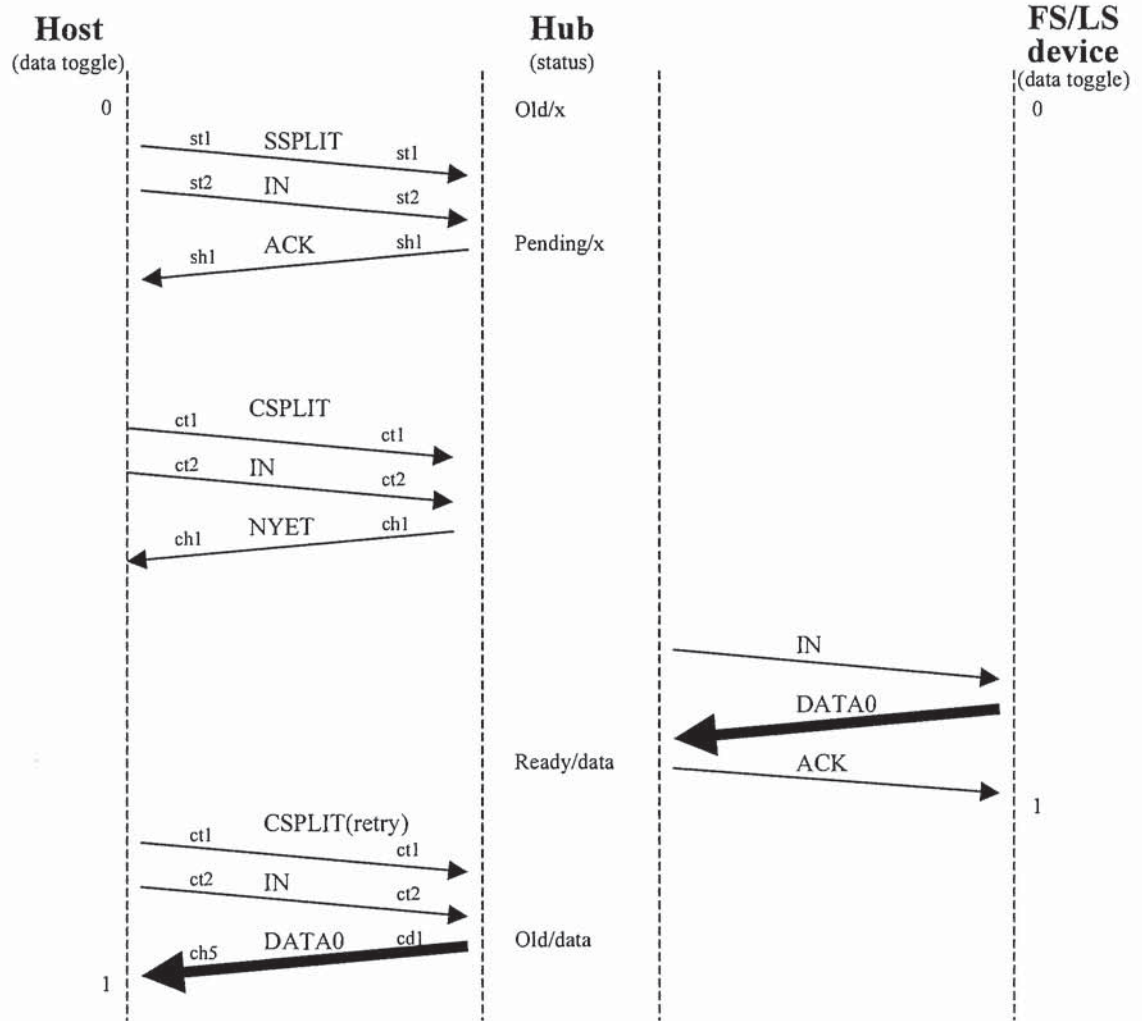


Figure A-42. CS Earlier No Smash(HS NYET)

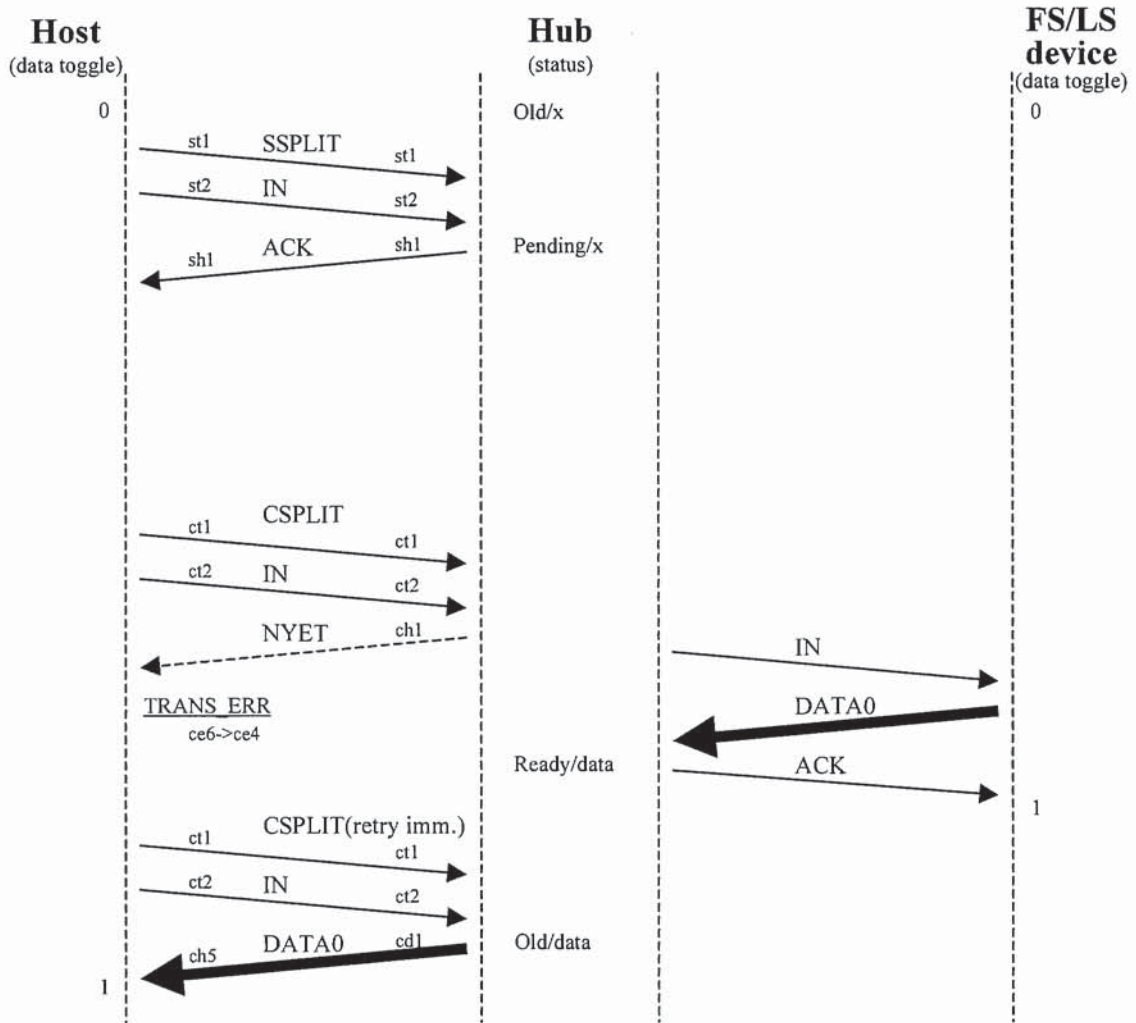


Figure A-43. CS Earlier HS NYET Smash(case 1)

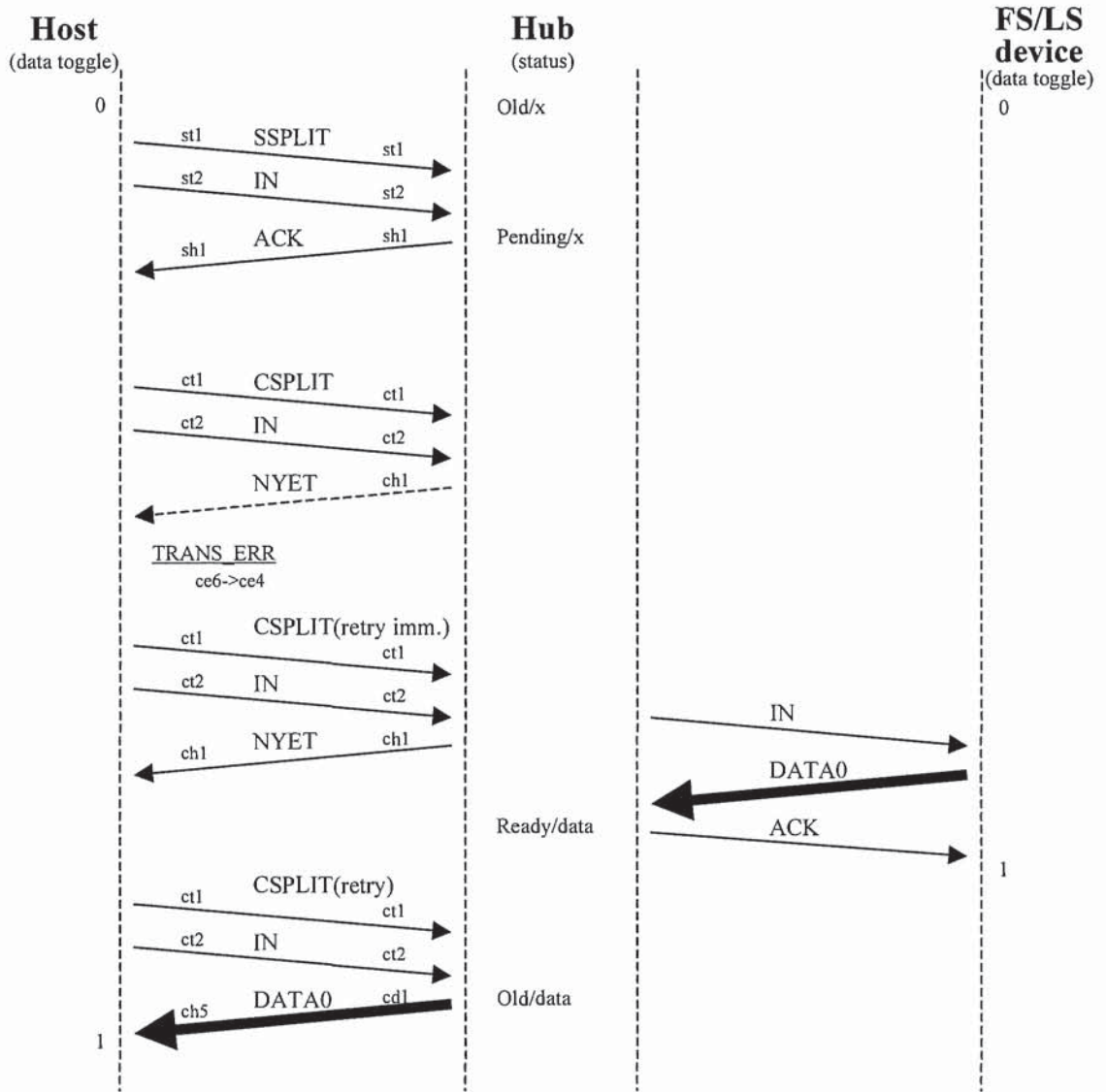


Figure A-44. CS Earlier HS NYET Smash(case 2)

Universal Serial Bus Specification Revision 2.0

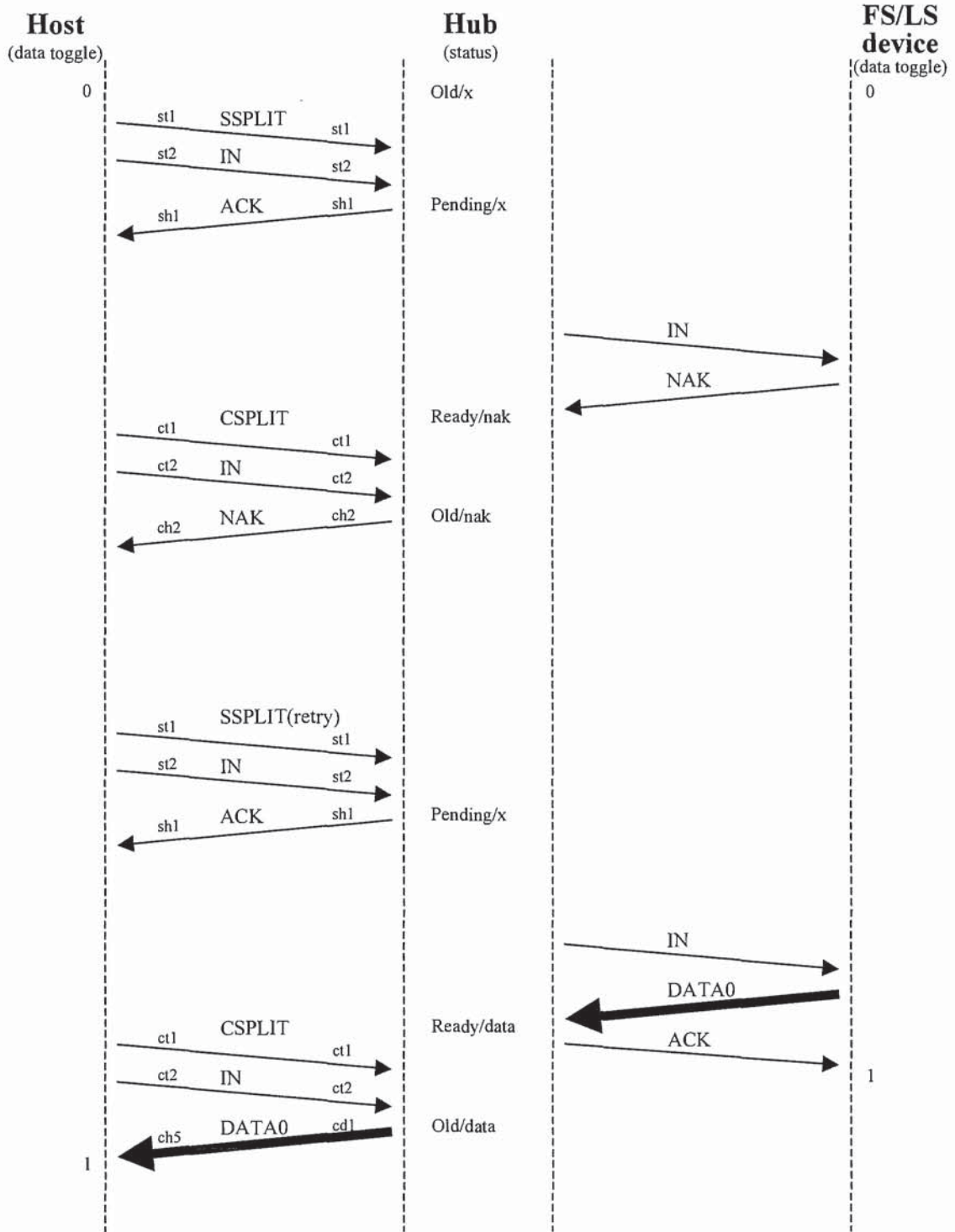


Figure A-45. Device Busy No Smash(FS/LS NAK)

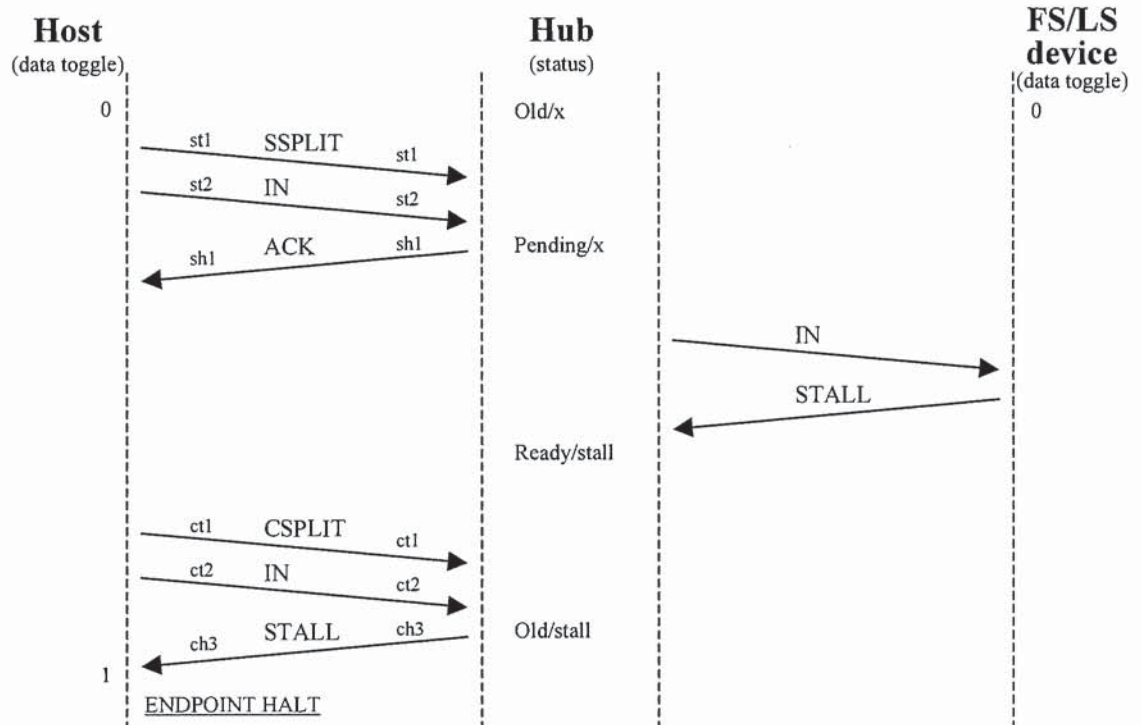


Figure A-46. Device Stall No Smash(FS/LS STALL)

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	

Universal Serial Bus Specification Revision 2.0

FS/LS ACK 3 strikes smash	No figure	
---------------------------	-----------	--

- Searching

Case	Reference Figure	Similar Figure
No smash	Figure A-55	

- CS(Complete-split transaction) earlier cases

Case	Reference Figure	Similar Figure
No smash (HS NYET and 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 NYET and FS/LS transaction is continued at end of M+3)	Figure A-60	
No smash and free(HS NYET and 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

Universal Serial Bus Specification Revision 2.0

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	

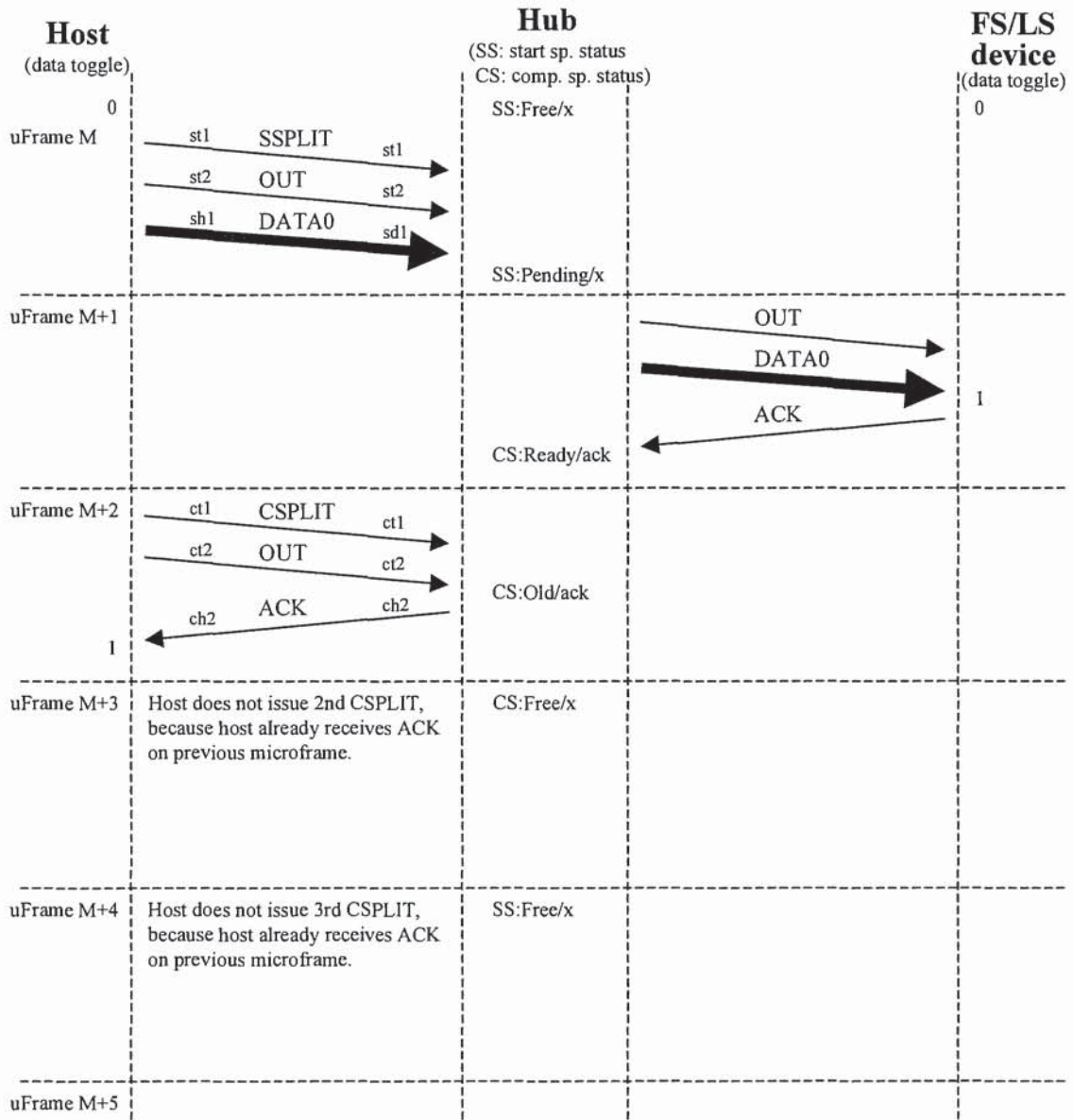


Figure A-47. Normal No Smash(FS/LS Handshake Packet is Done by M+1)

Universal Serial Bus Specification Revision 2.0

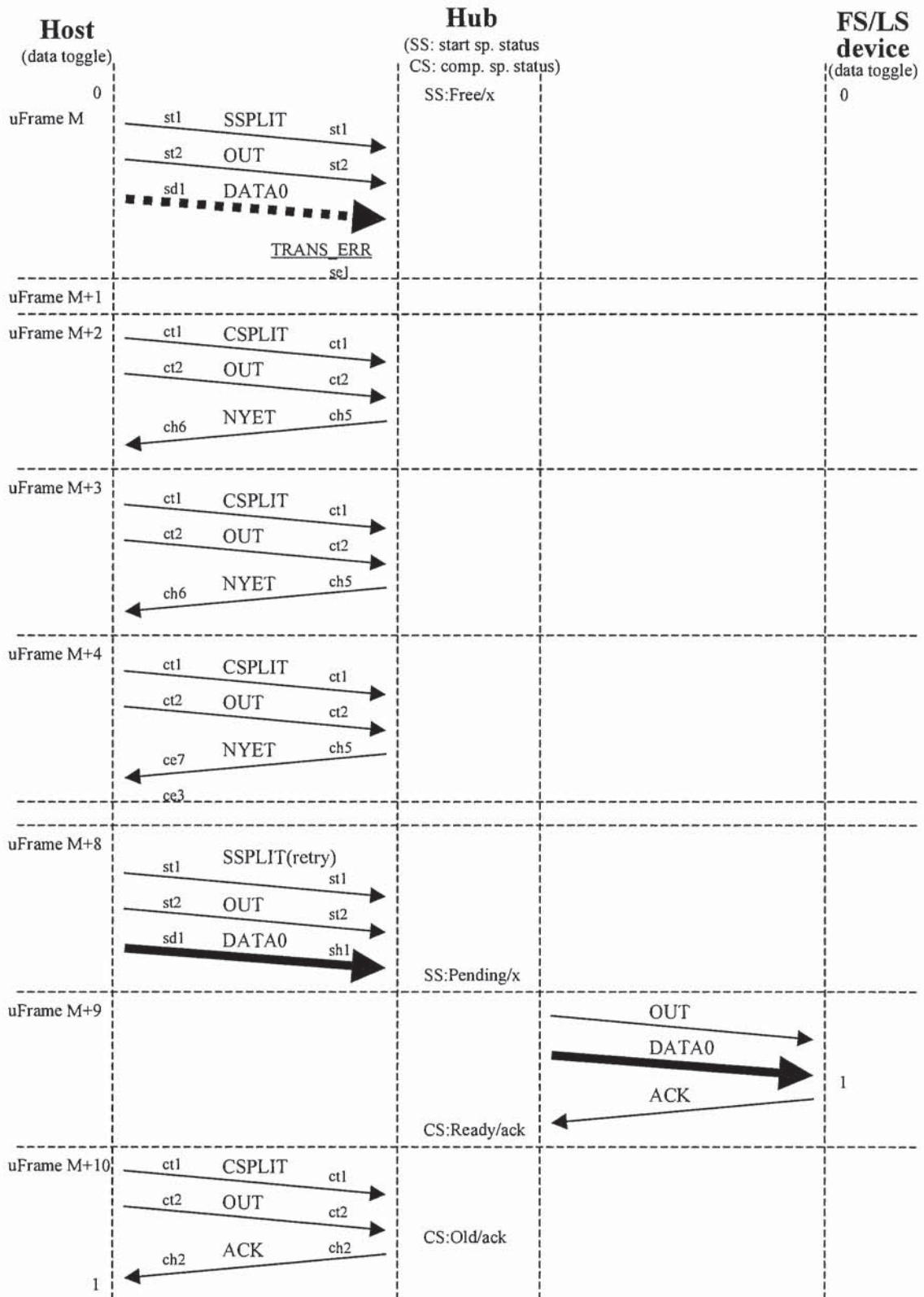


Figure A-48. Normal HS DATA0/1 Smash

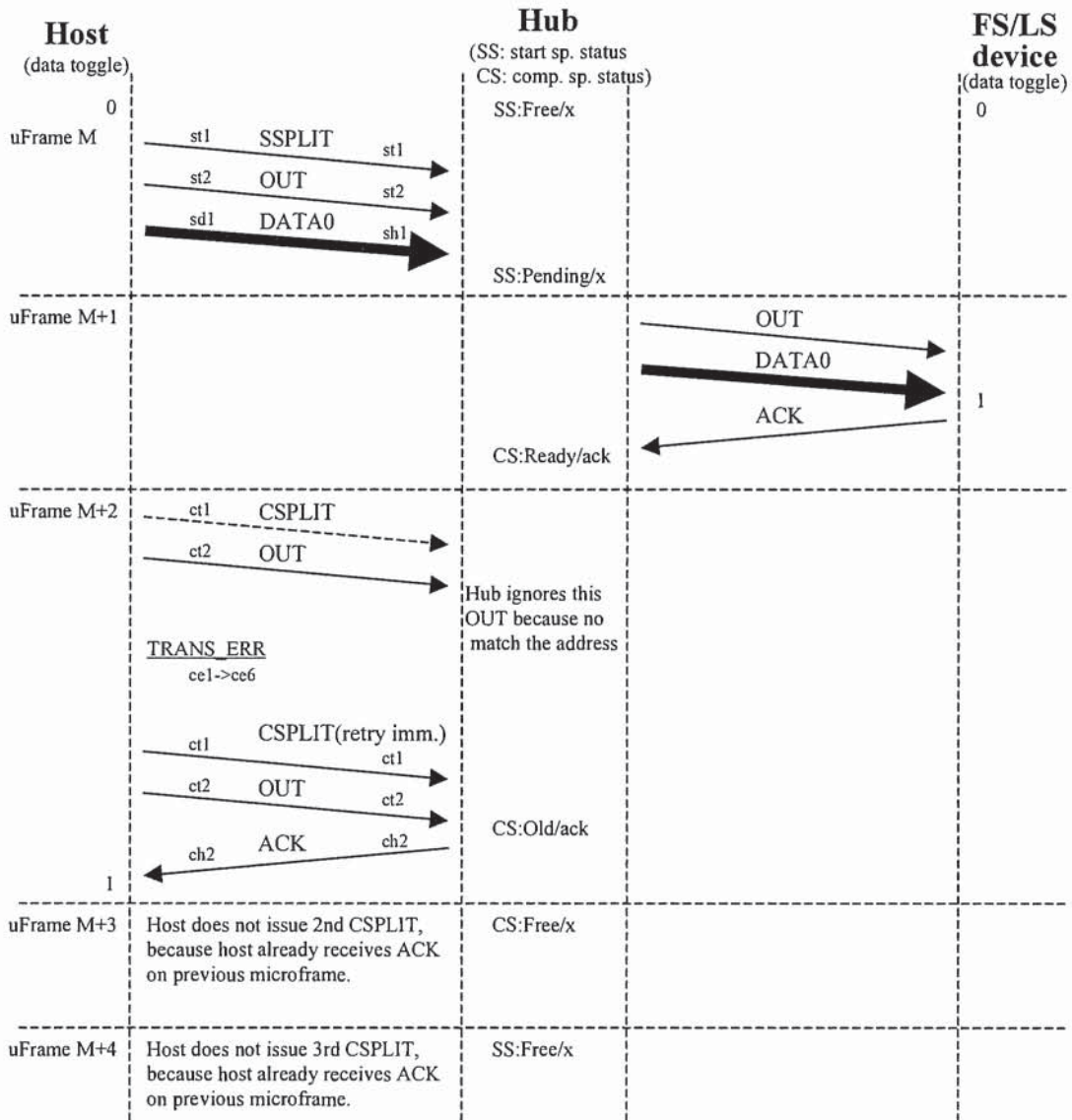


Figure A-49. Normal HS CSPLIT Smash

Universal Serial Bus Specification Revision 2.0

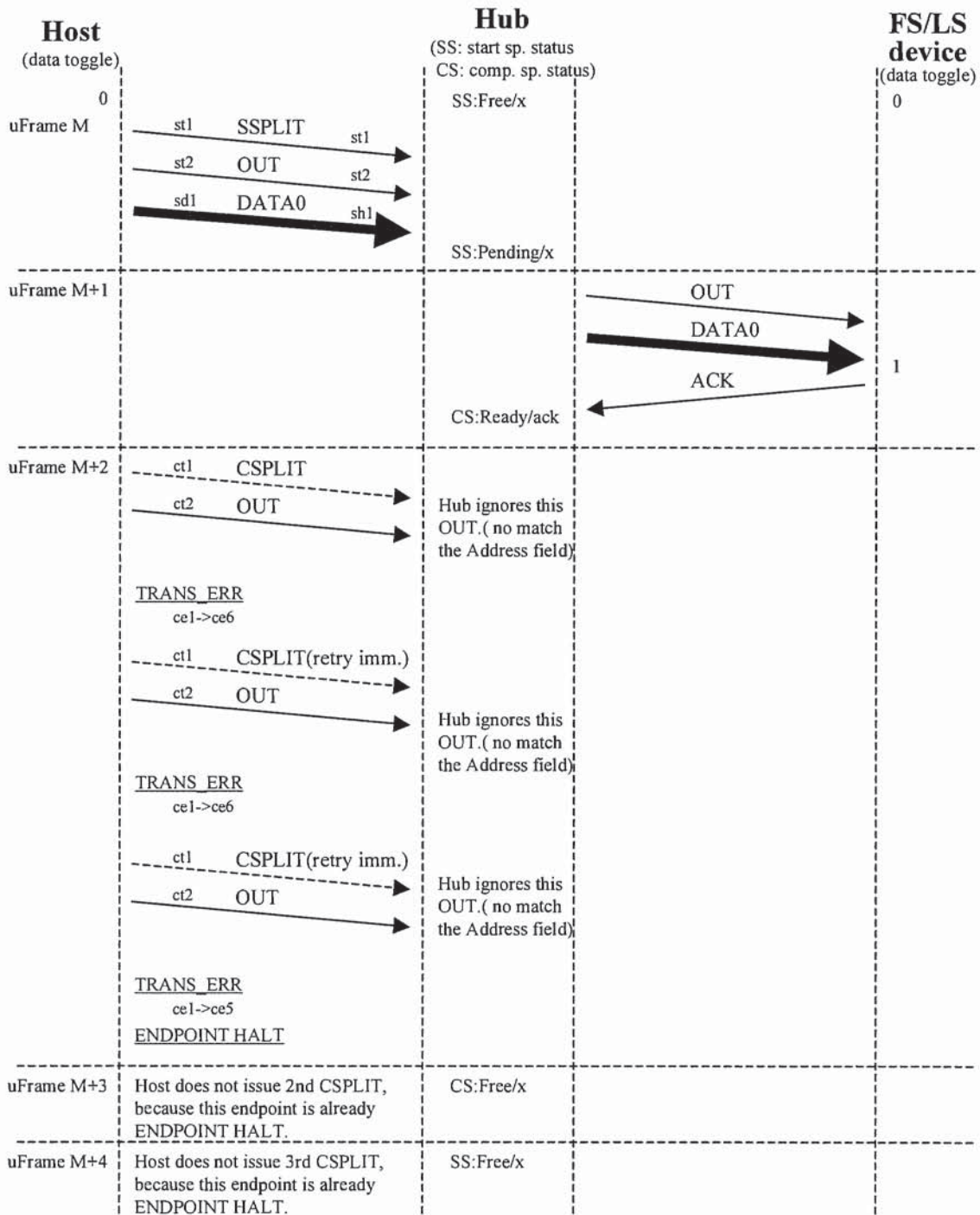


Figure A-50. Normal HS CSPLIT 3 Strikes Smash

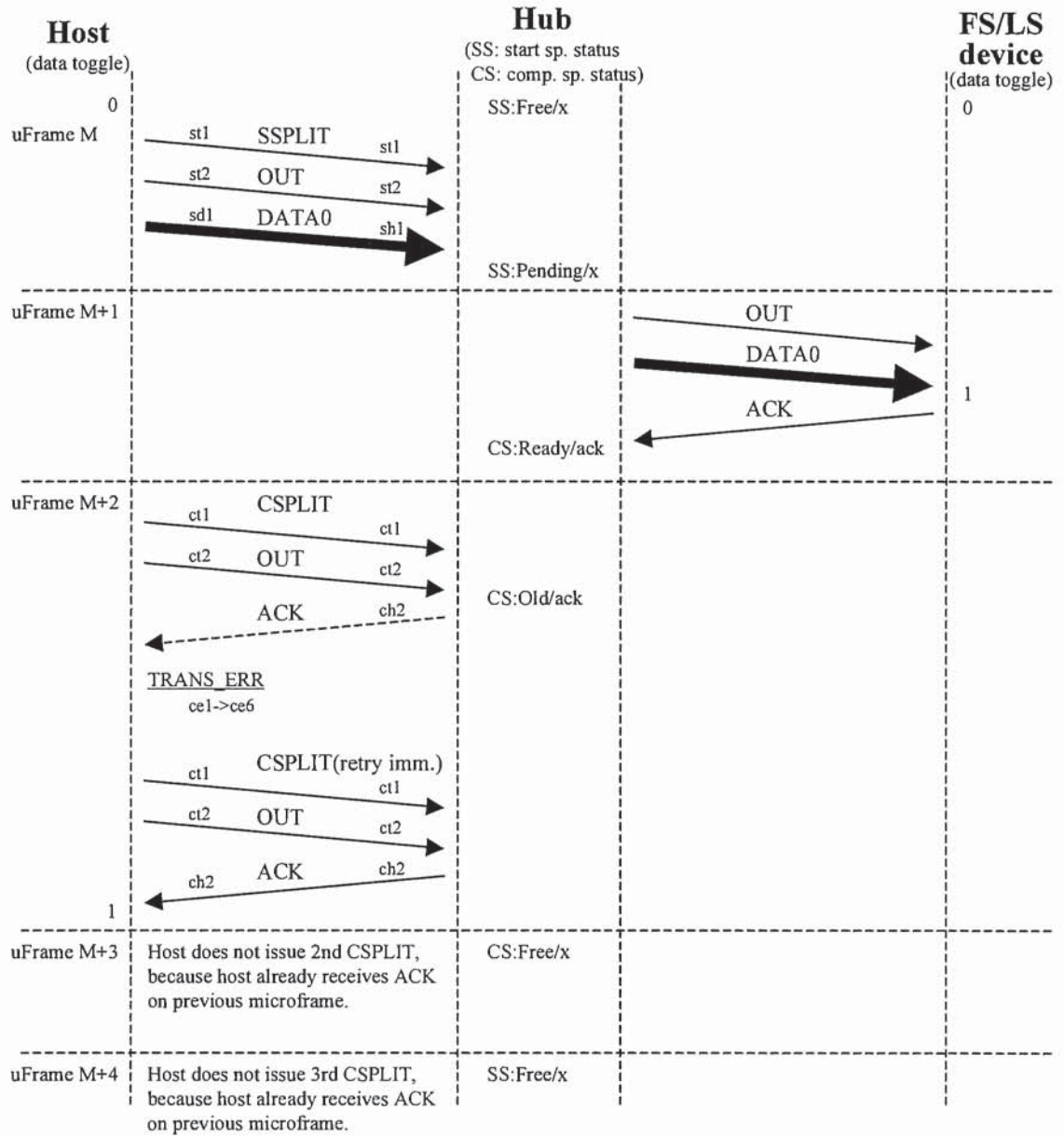


Figure A-51. Normal HS ACK(C) Smash

Universal Serial Bus Specification Revision 2.0

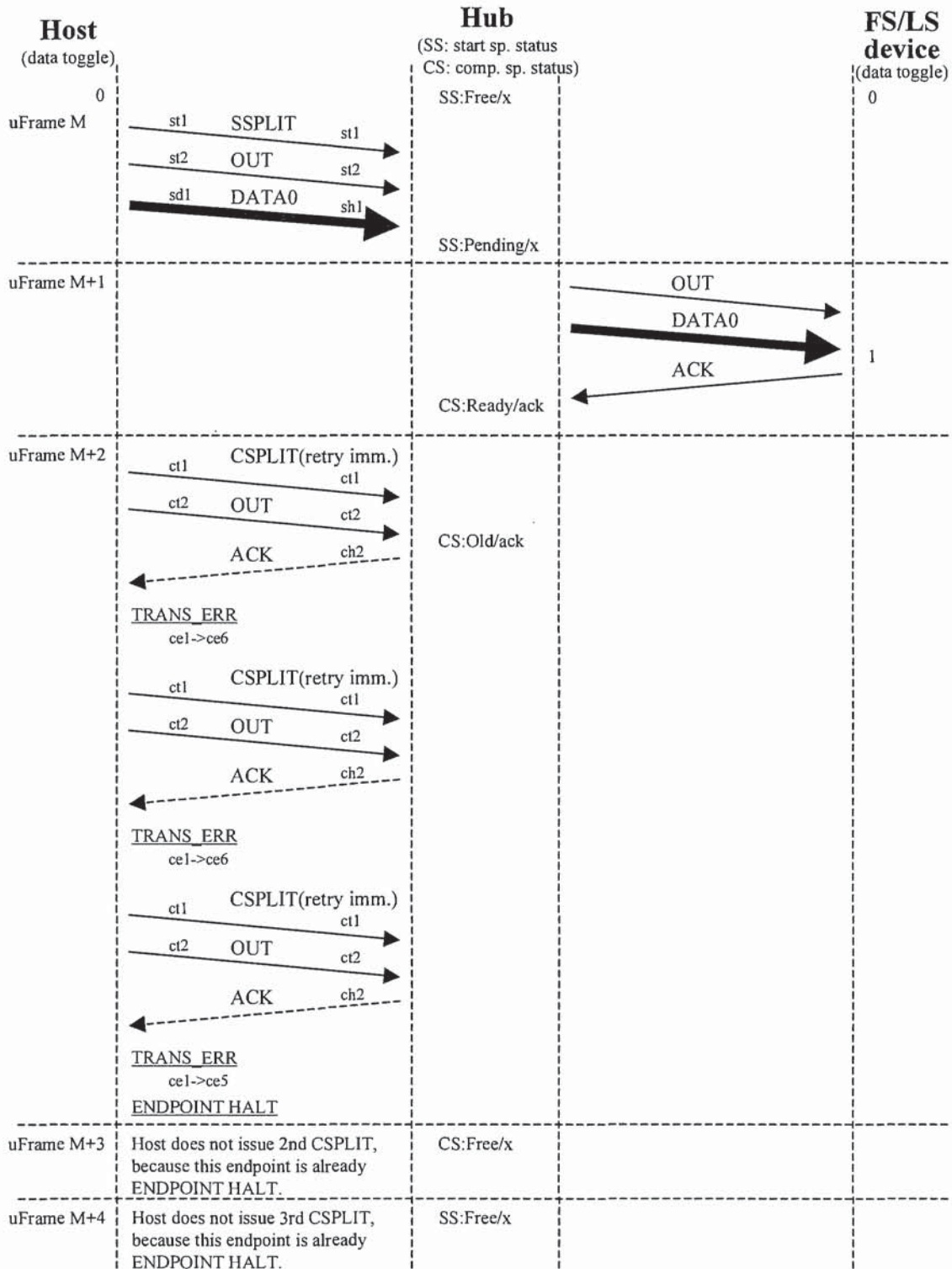


Figure A-52. Normal HS ACK(C) 3 Strikes Smash

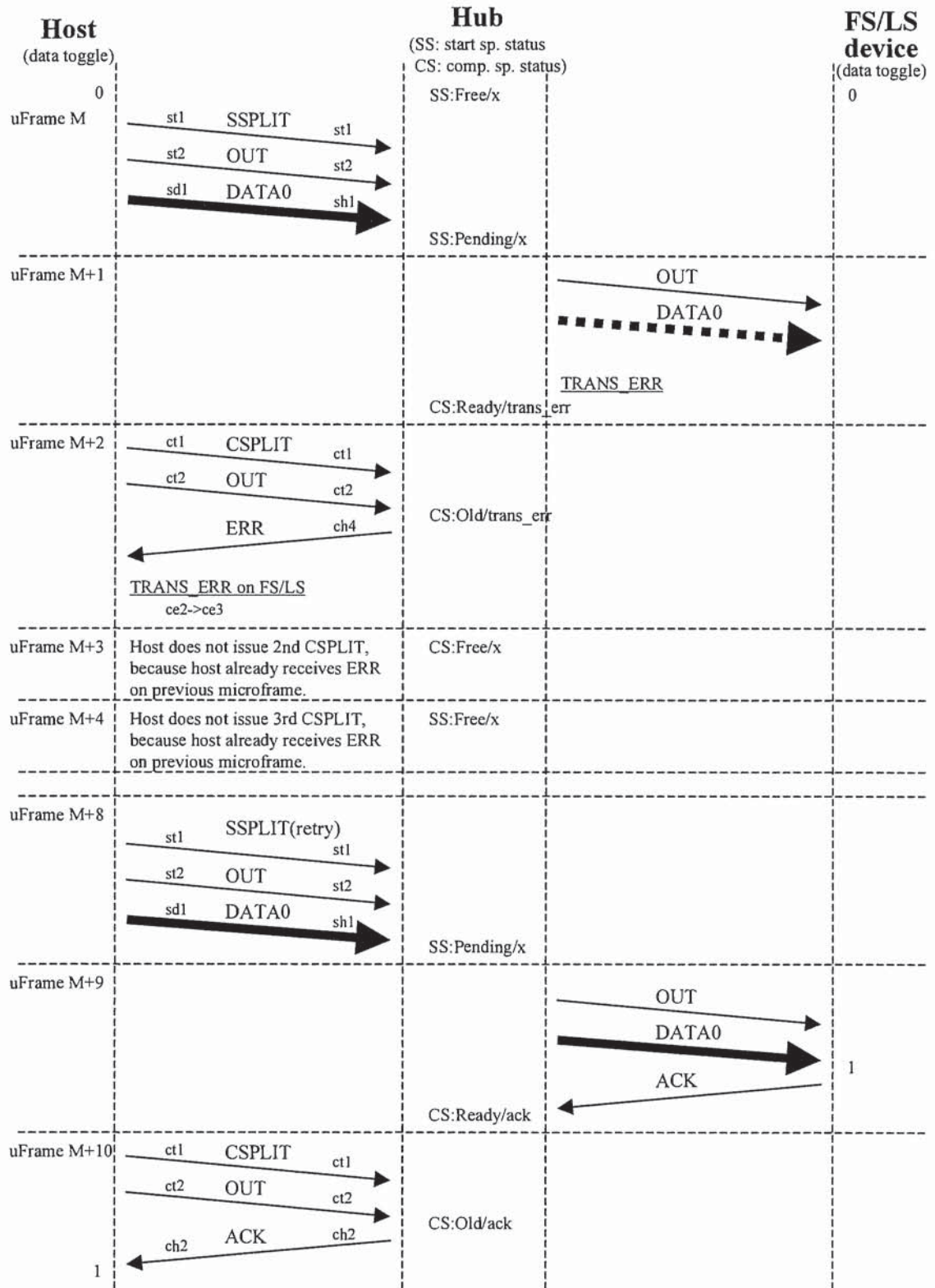


Figure A-53. Normal FS/LS DATA0/1 Smash

Universal Serial Bus Specification Revision 2.0

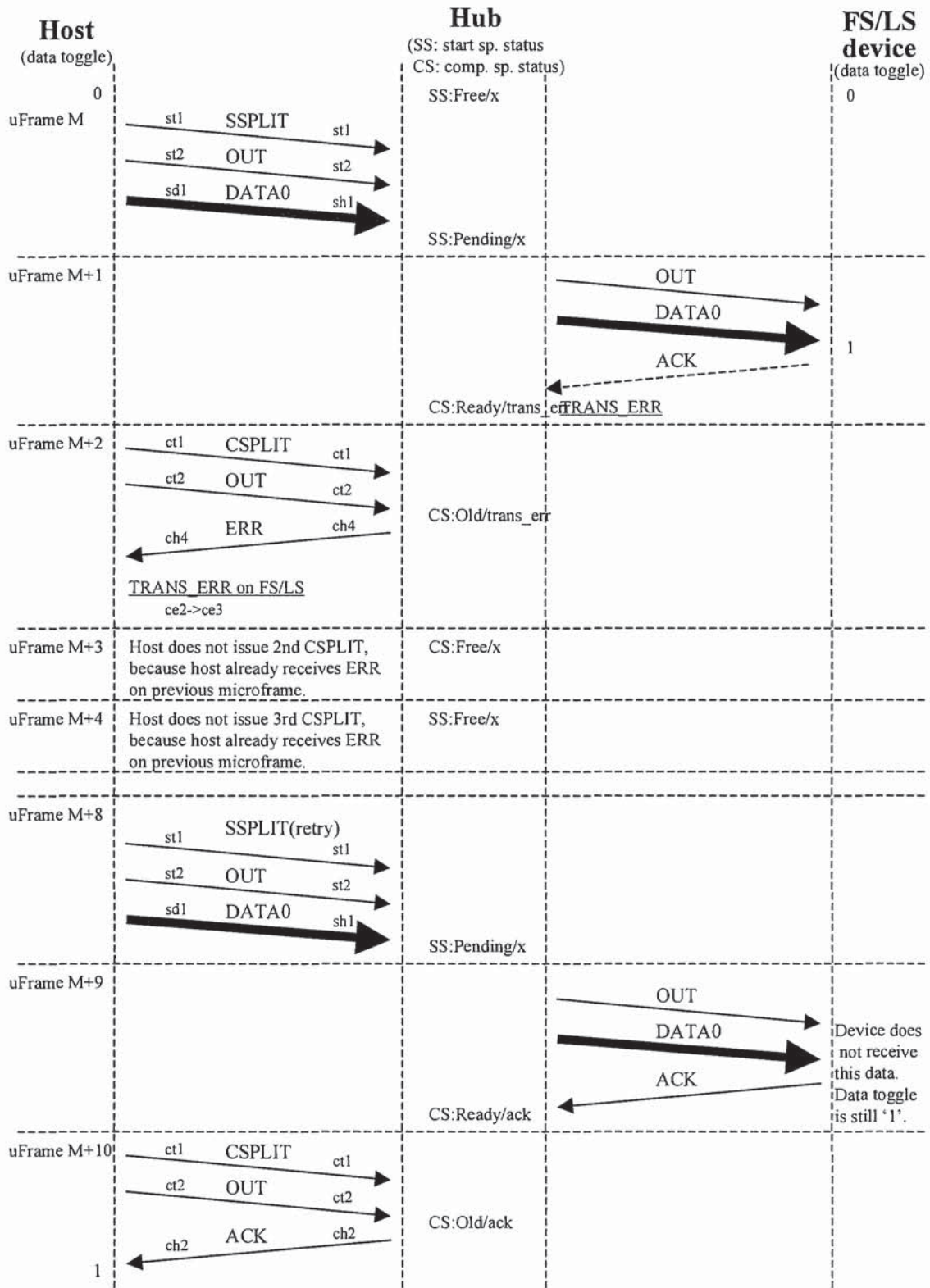


Figure A-54. Normal FS/LS ACK Smash

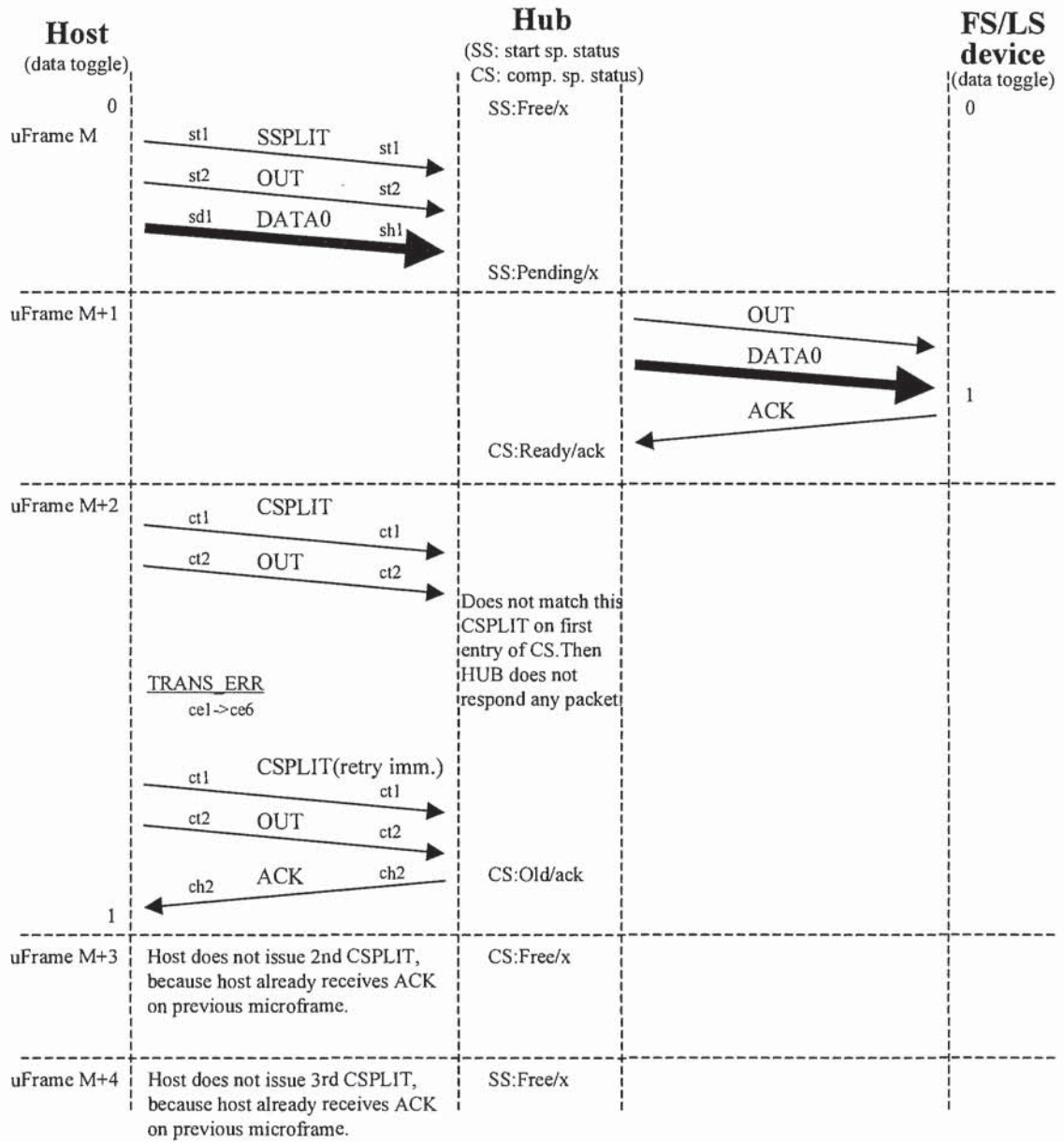


Figure A-55. Searching No Smash

Universal Serial Bus Specification Revision 2.0

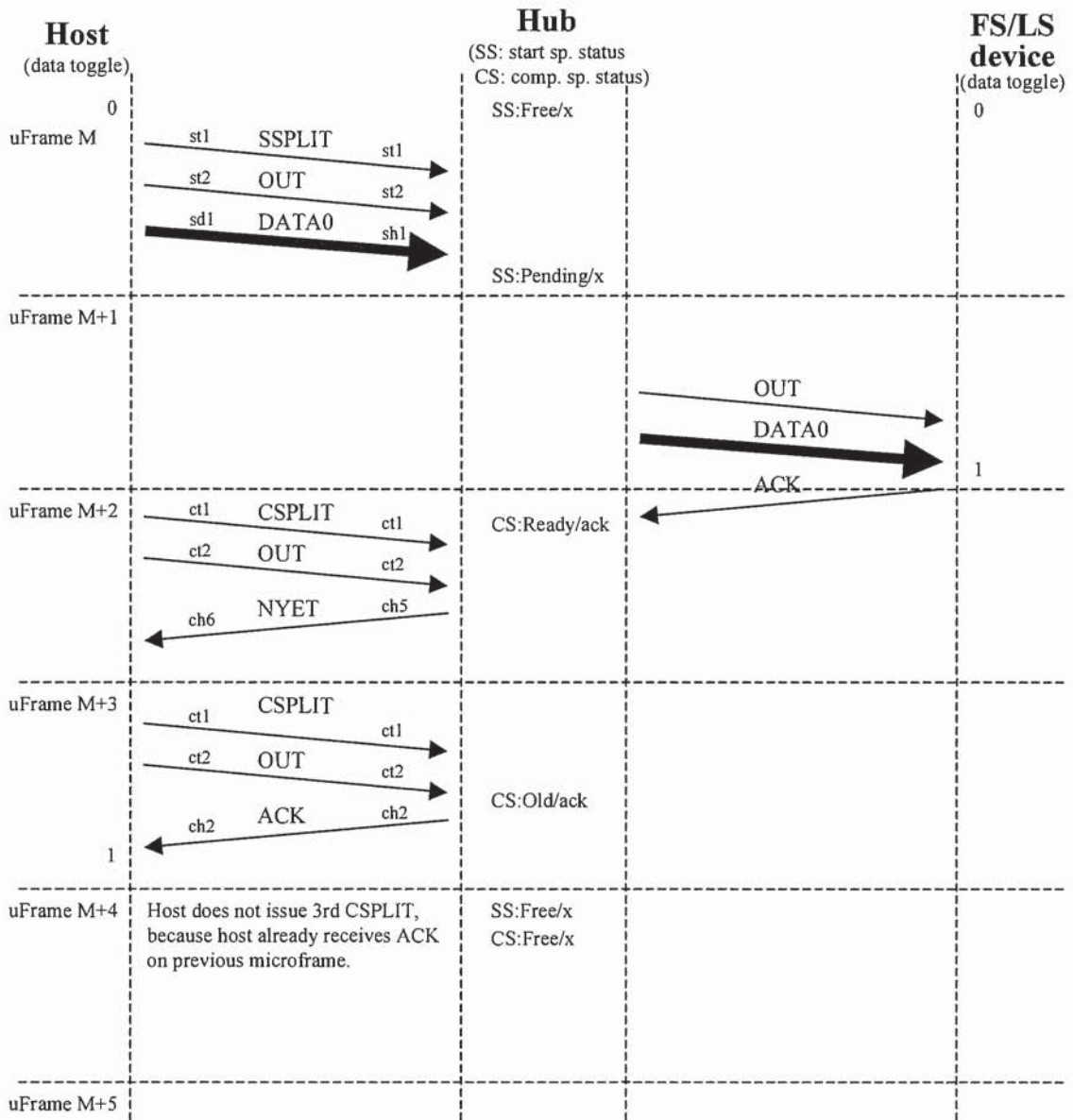


Figure A-56. CS Earlier No Smash(HS NYET and FS/LS Handshake Packet is Done by M+2)

Universal Serial Bus Specification Revision 2.0

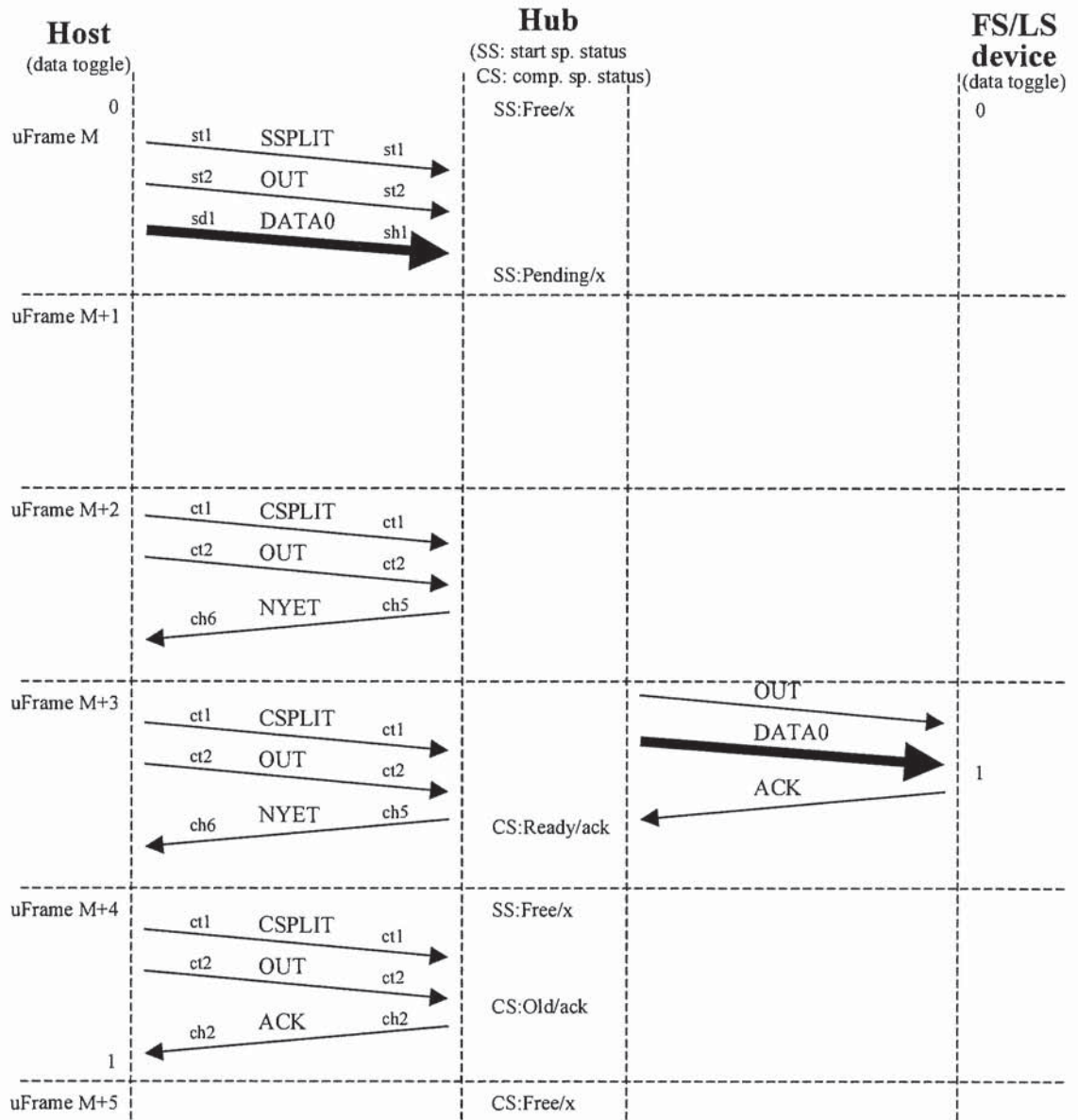


Figure A-57. CS Earlier No Smash(HS NYET and FS/LS Handshake Packet is Done by M+3)

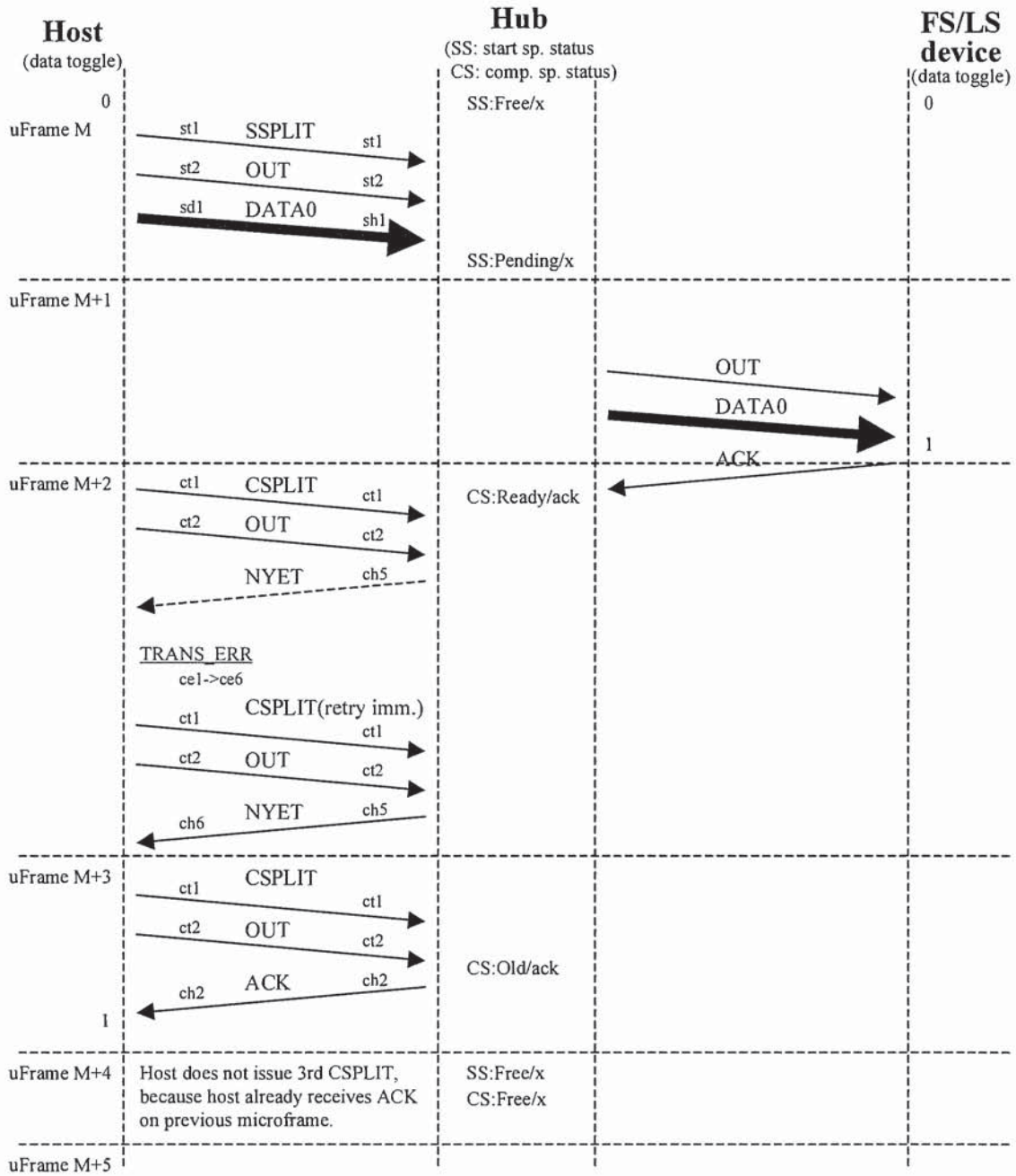


Figure A-58. CS Earlier HS NYET Smash

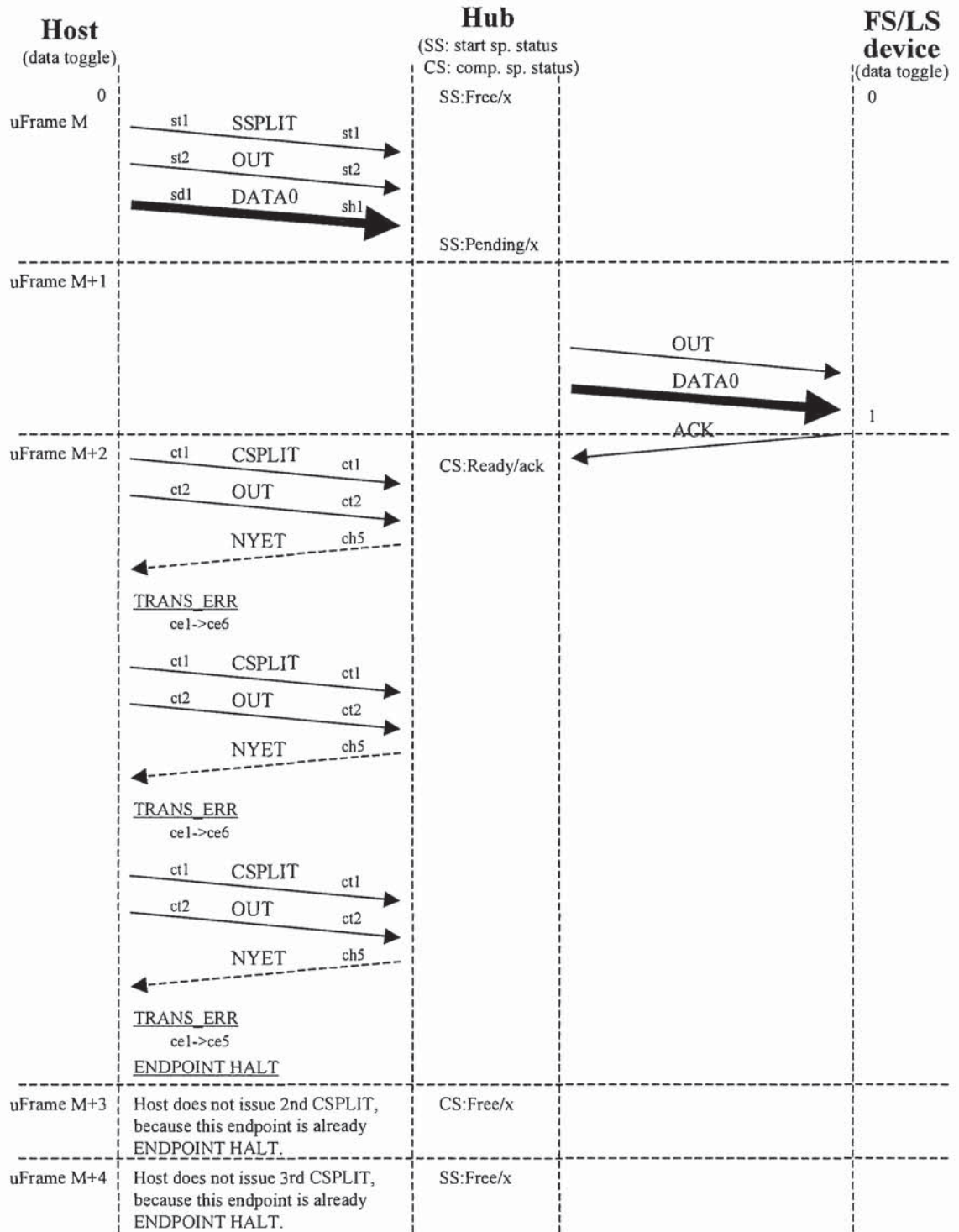


Figure A-59. CS Earlier HS NYET 3 Strikes Smash

Universal Serial Bus Specification Revision 2.0

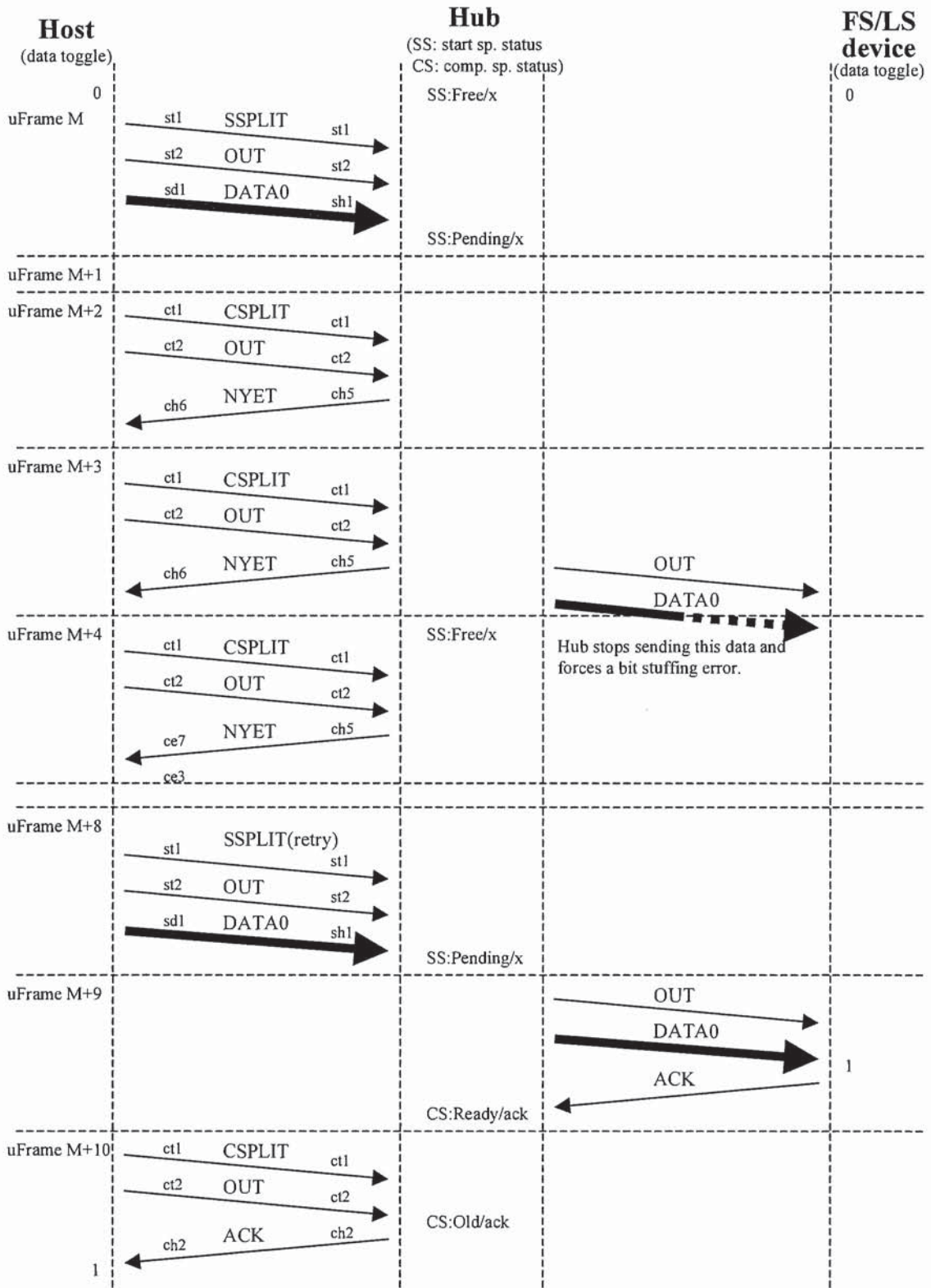


Figure A-60. Abort and Free Abort(FS/LS Transaction is Continued at End of M+3)

Universal Serial Bus Specification Revision 2.0

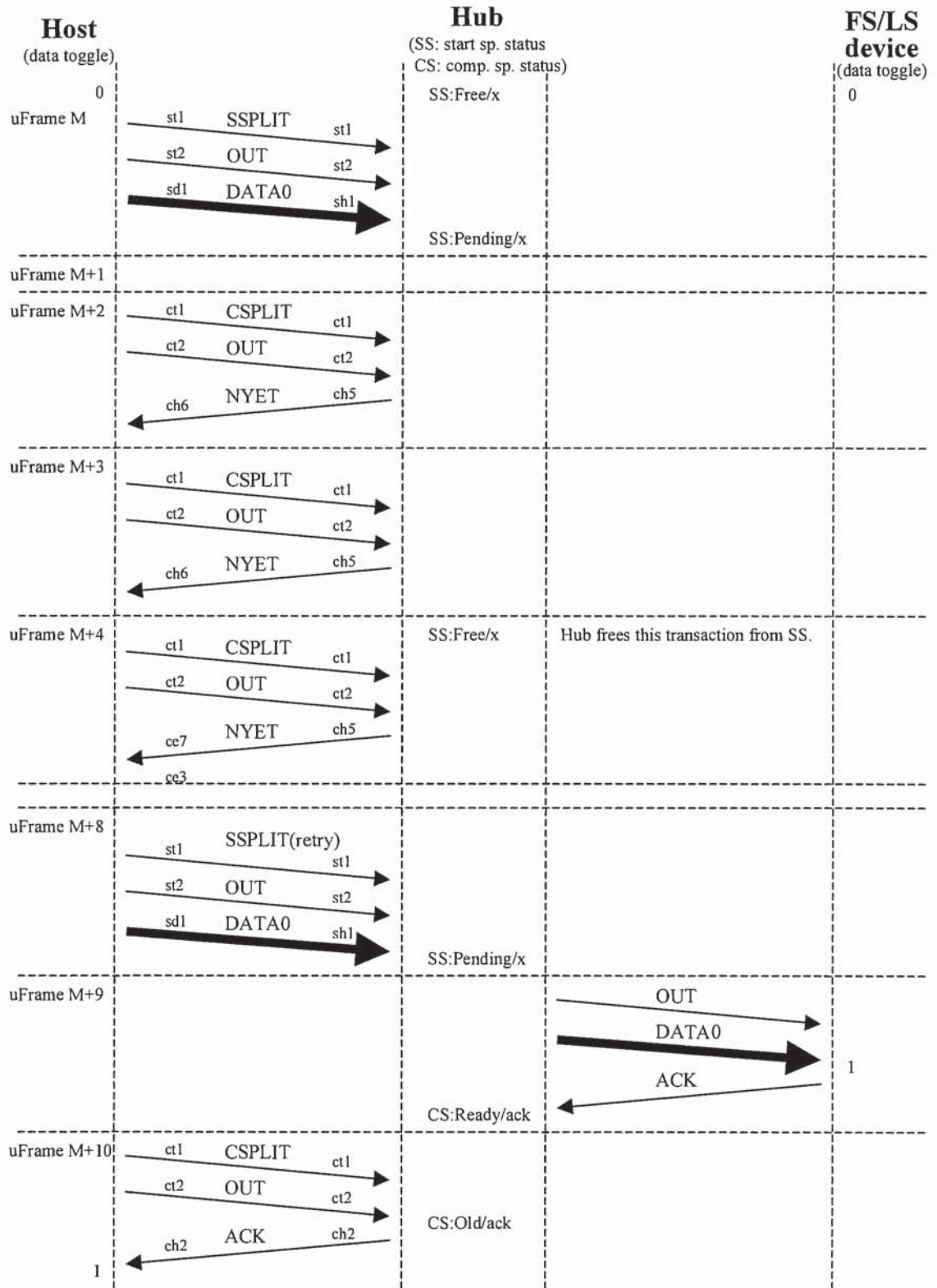


Figure A-61. Abort and Free Free(FS/LS Transaction is not Started at End of M+3)

Universal Serial Bus Specification Revision 2.0

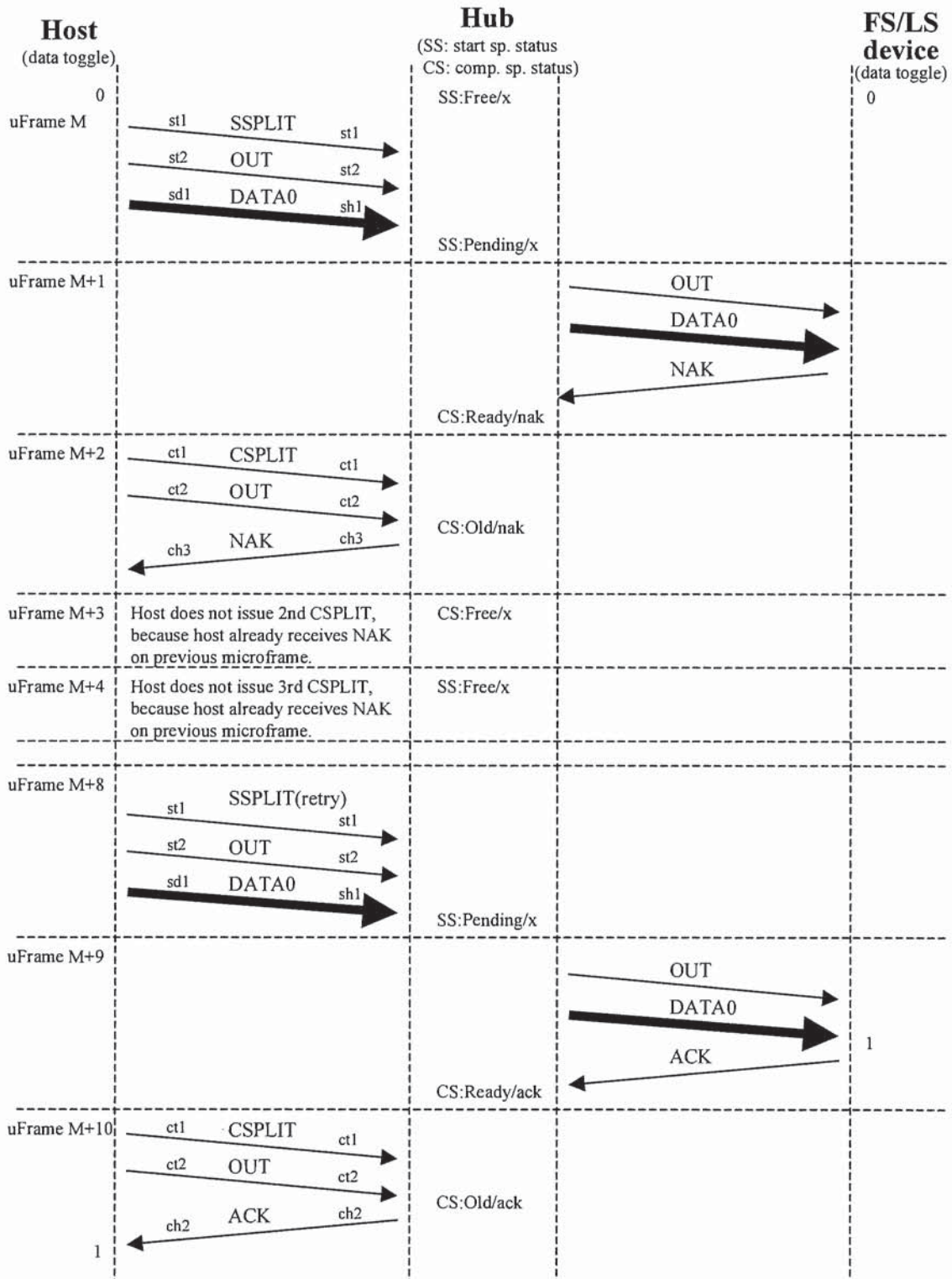


Figure A-62. Device Busy No Smash(FS/LS NAK)

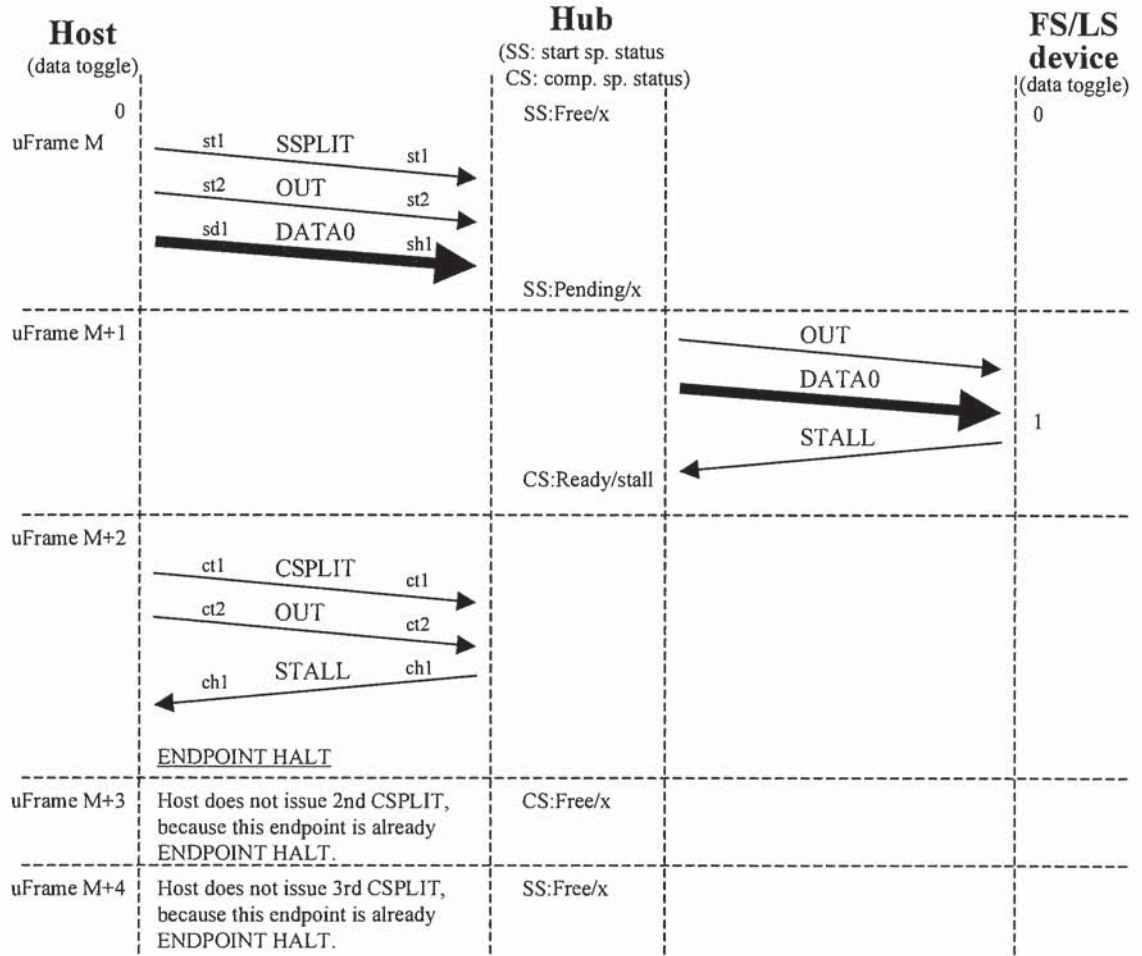


Figure A-63. Device Stall No Smash(FS/LS STALL)

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	

Universal Serial Bus Specification Revision 2.0

- Searching

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 NYET and FS/LS transaction is continued at end of M+3)	Figure A-80	
No smash and free (HS NYET and 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

Universal Serial Bus Specification Revision 2.0

- 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	

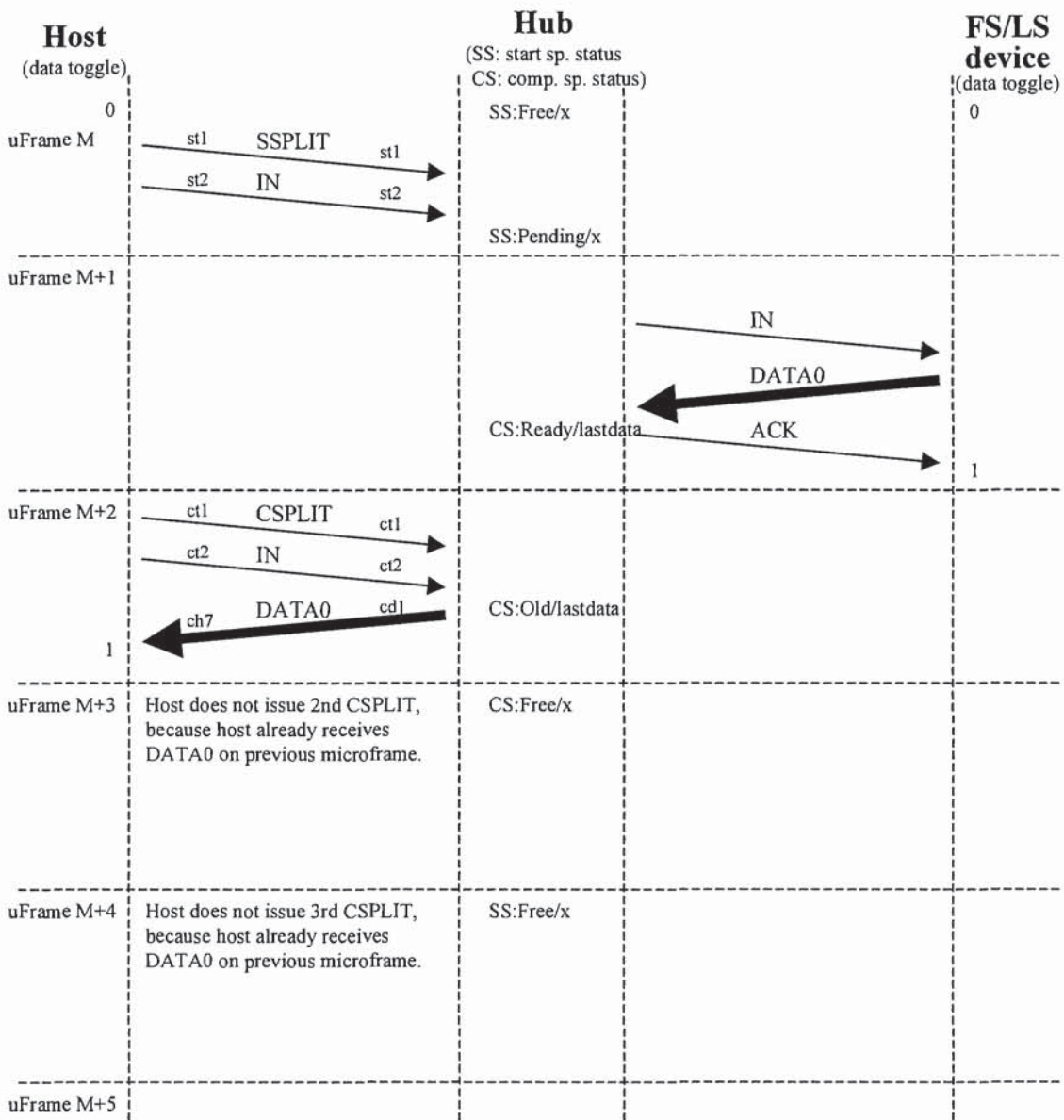


Figure A-64. Normal No Smash(FS/LS Data Packet is on M+1)

Universal Serial Bus Specification Revision 2.0

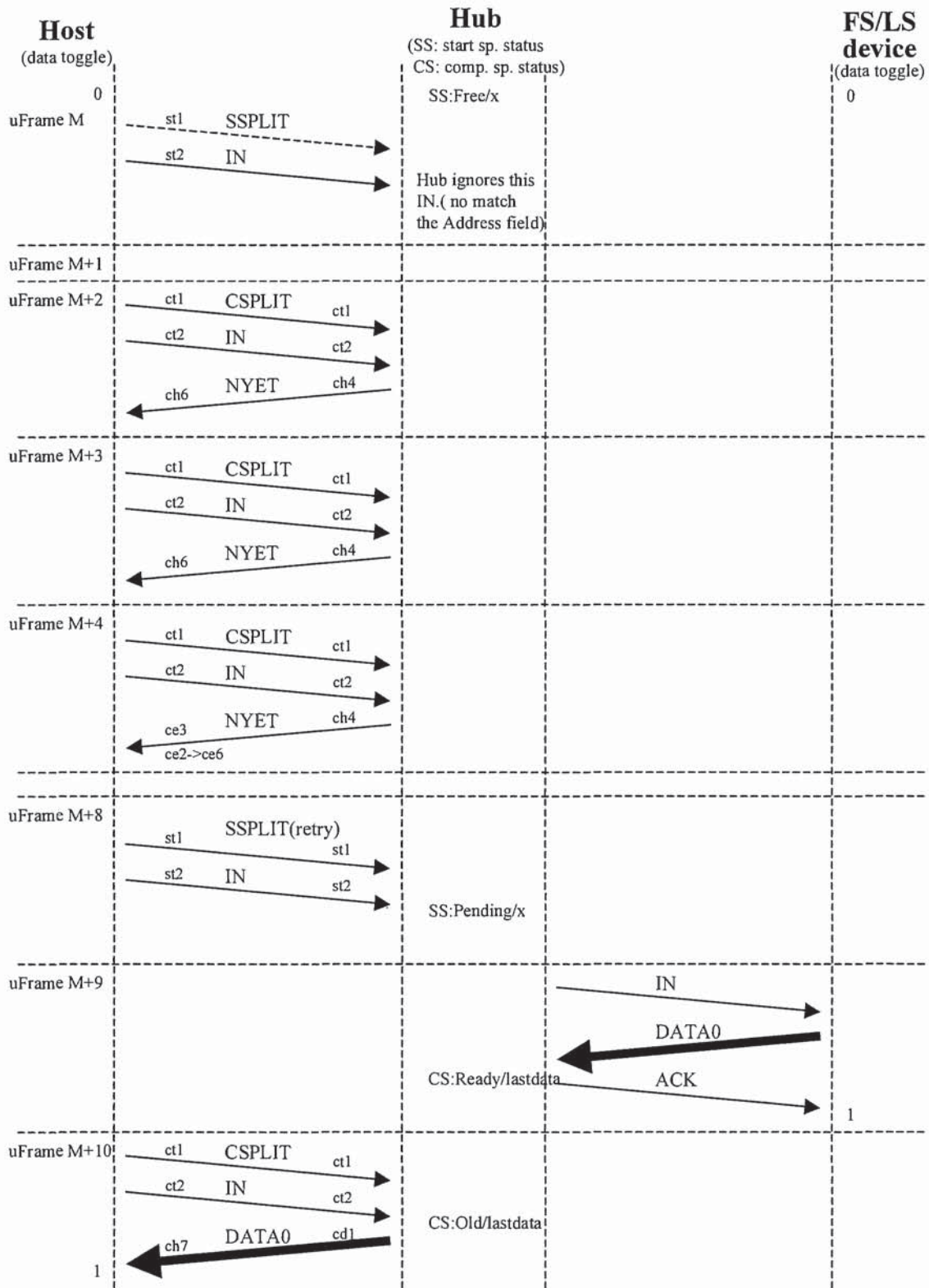


Figure A-65. Normal HS SSPLIT Smash

Universal Serial Bus Specification Revision 2.0

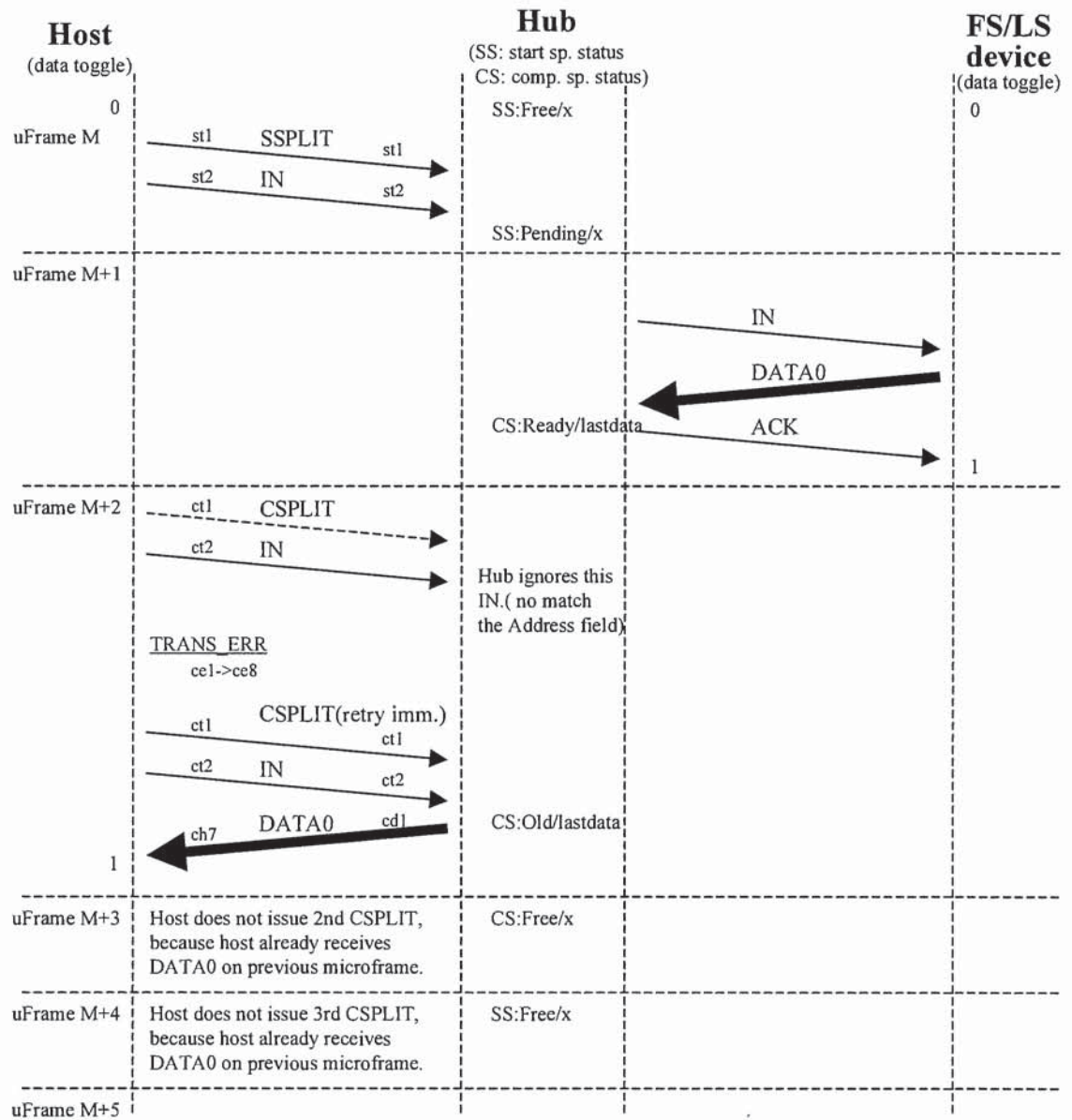


Figure A-66. Normal HS CSPLIT Smash

Universal Serial Bus Specification Revision 2.0

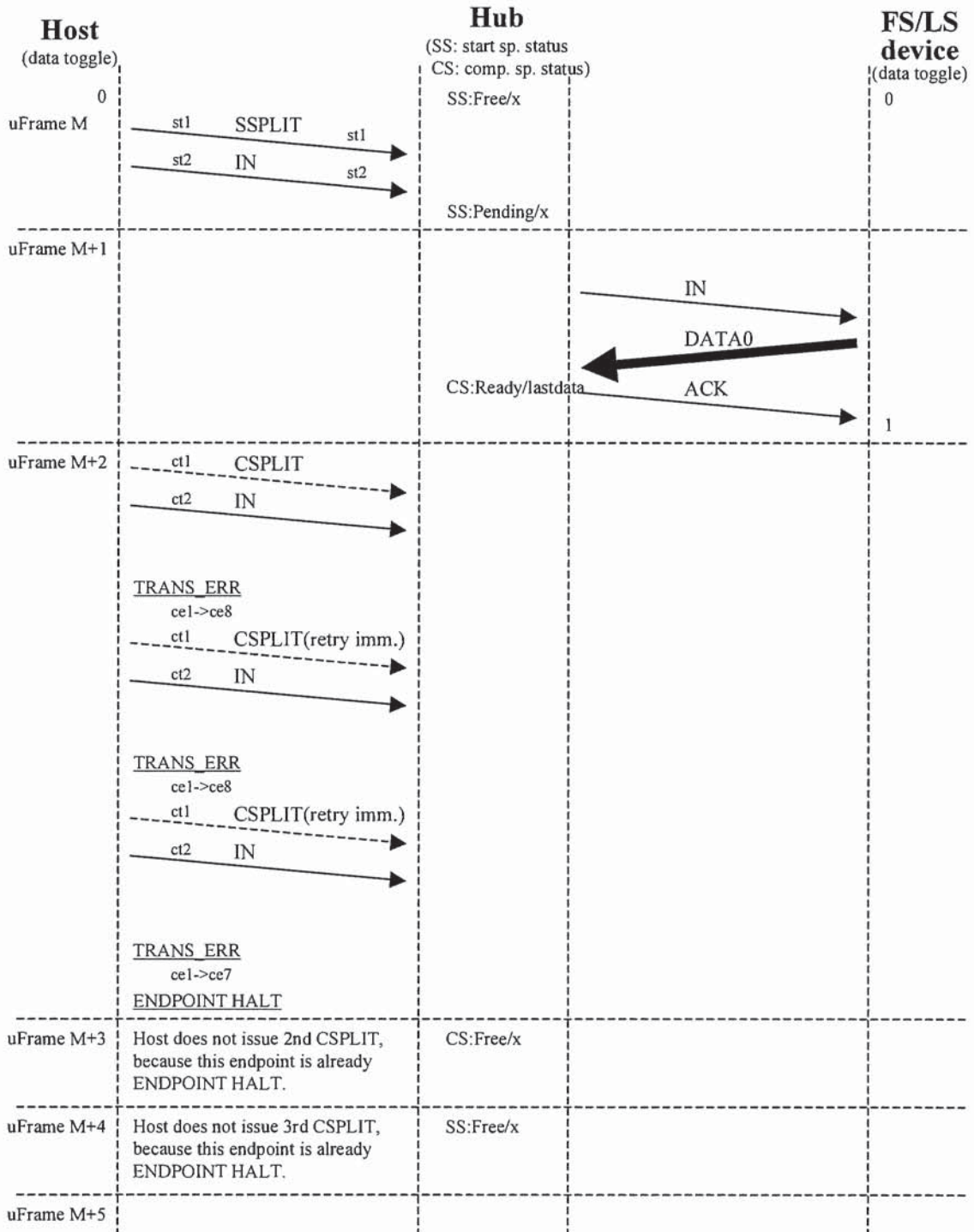


Figure A-67. Normal HS CSPLIT 3 Strikes Smash

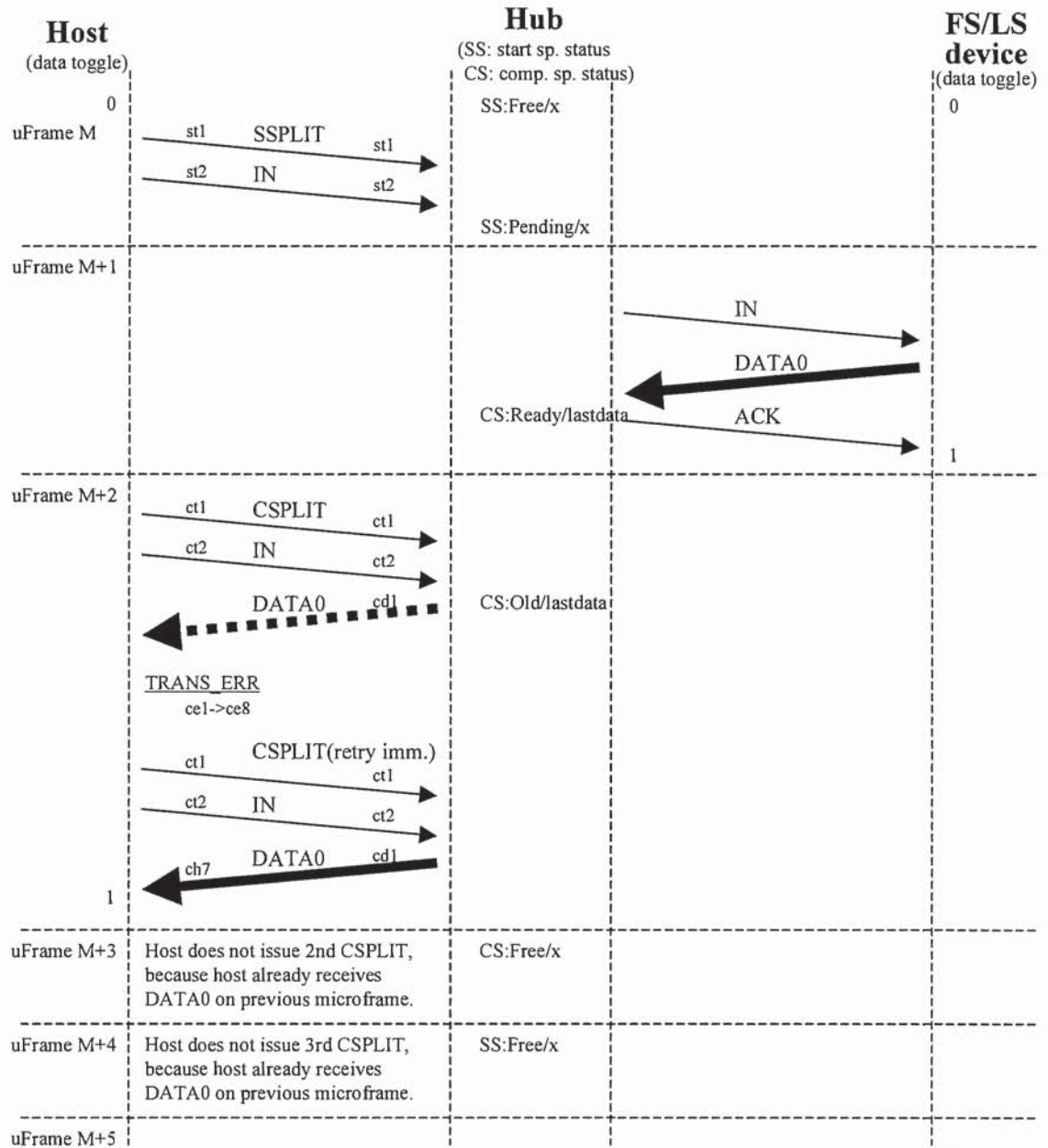


Figure A-68. Normal HS DATA0/1 Smash

Universal Serial Bus Specification Revision 2.0

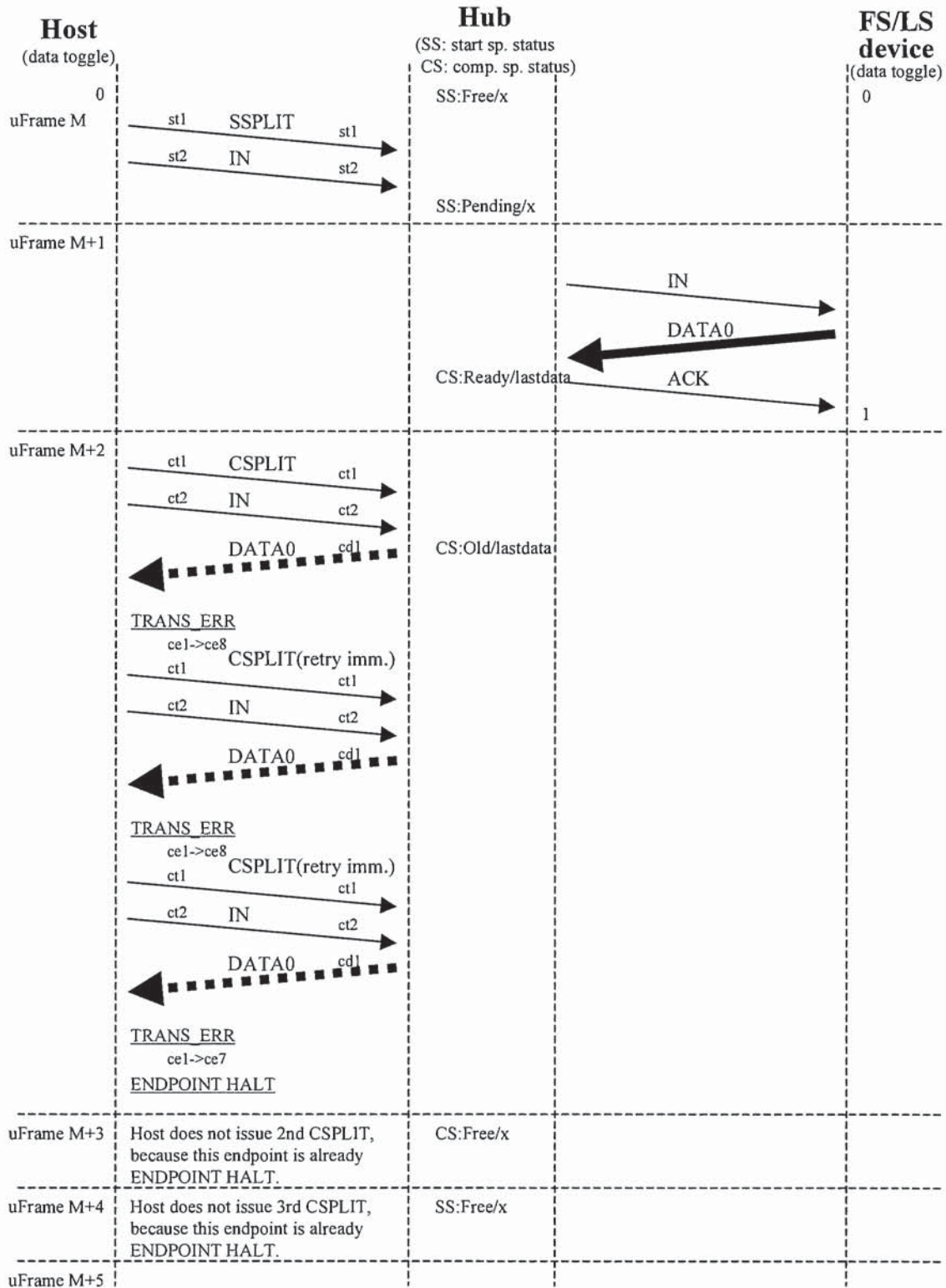


Figure A-69. Normal HS DATA0/1 3 Strikes Smash

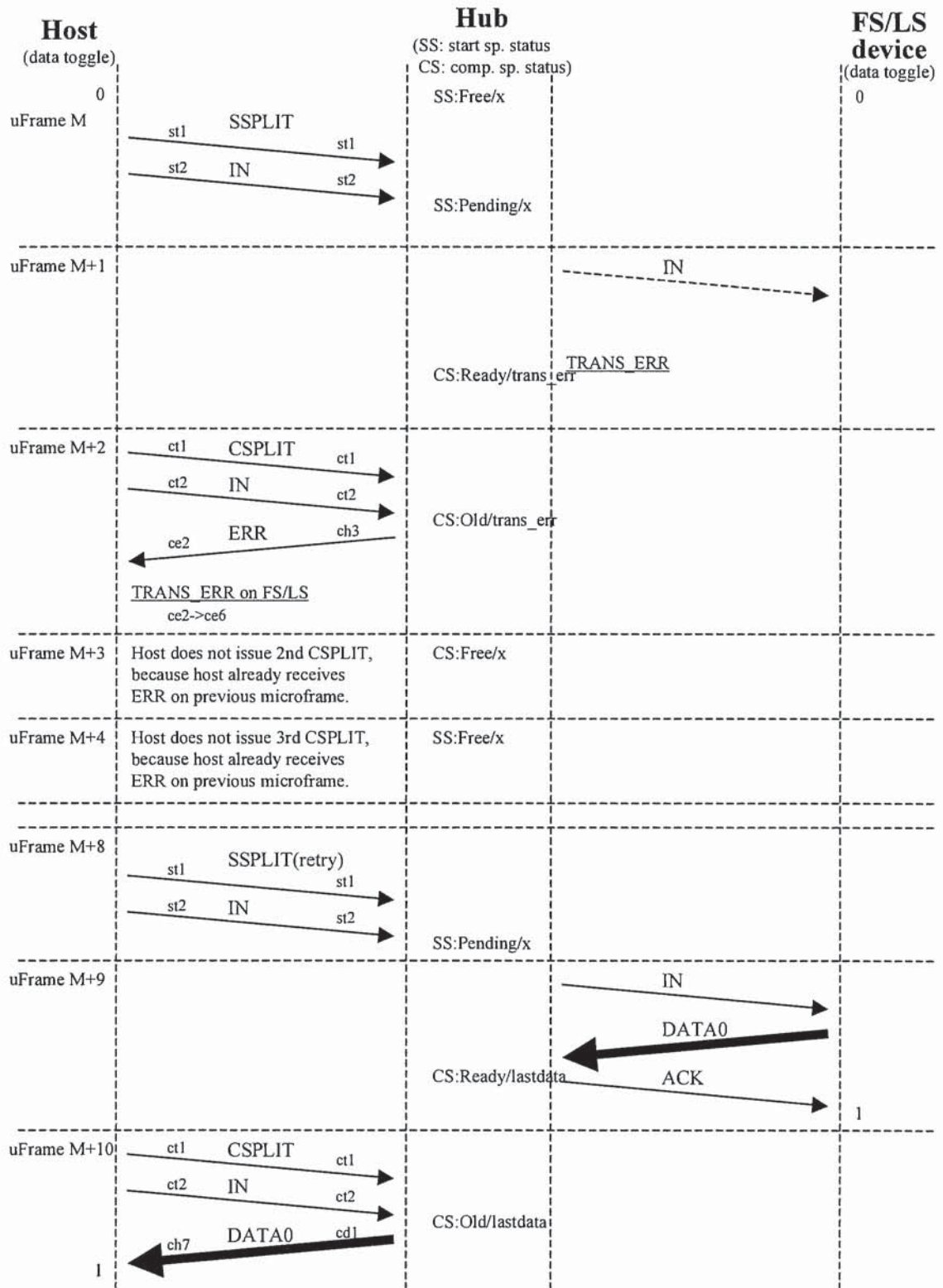


Figure A-70. Normal FS/LS IN Smash

Universal Serial Bus Specification Revision 2.0

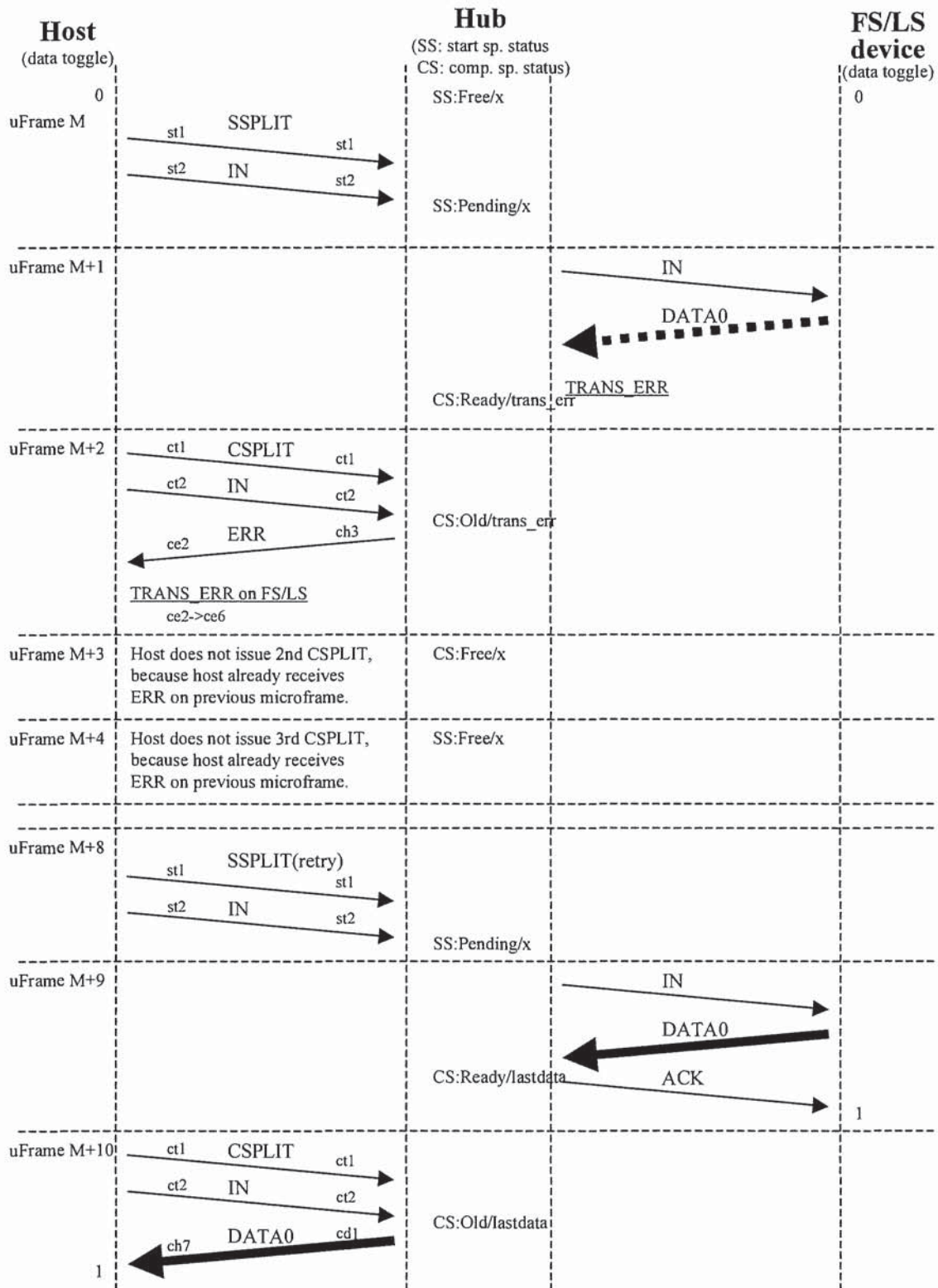


Figure A-71. Normal FS/LS DATA0/1 Smash

Universal Serial Bus Specification Revision 2.0

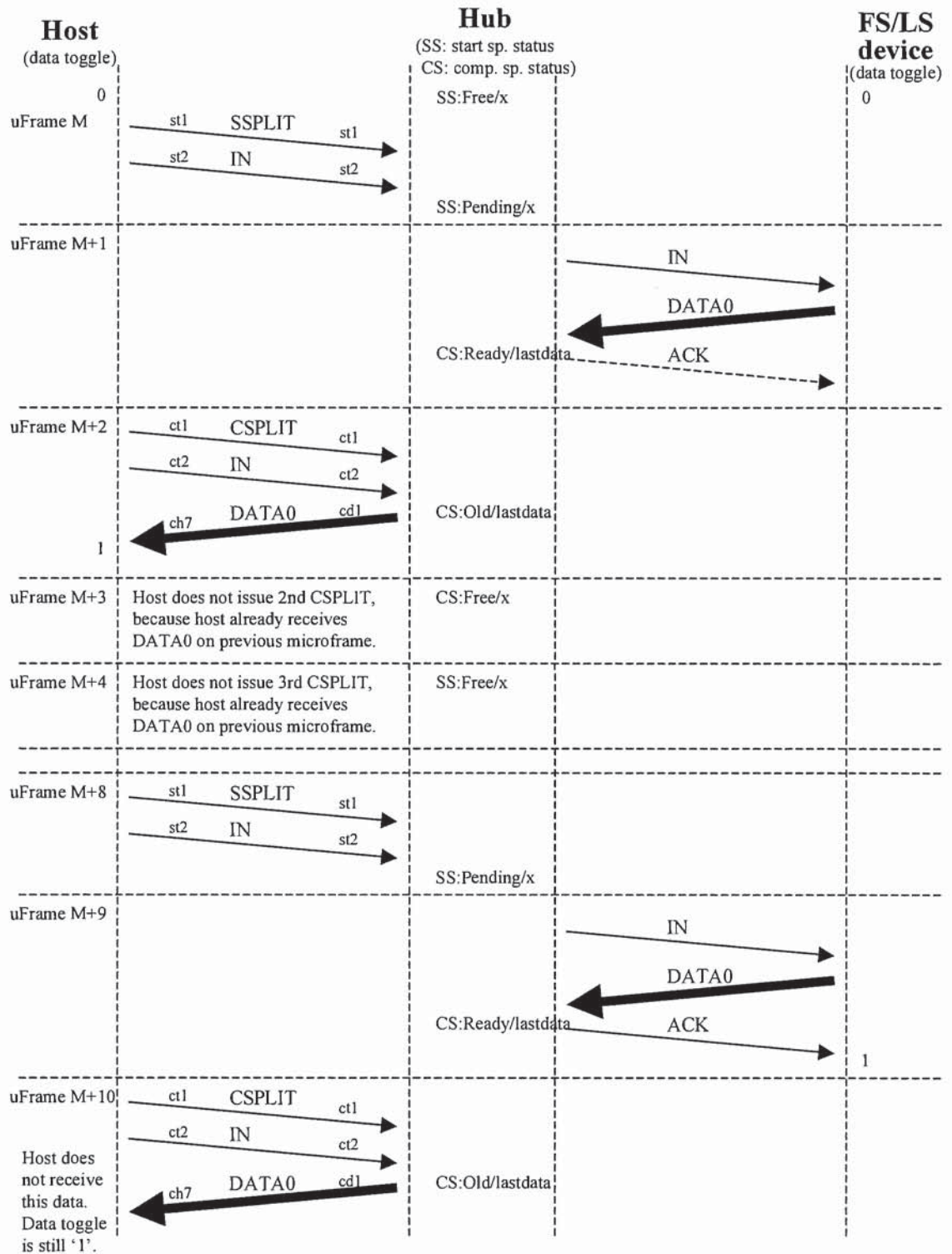


Figure A-72. Normal FS/LS ACK Smash

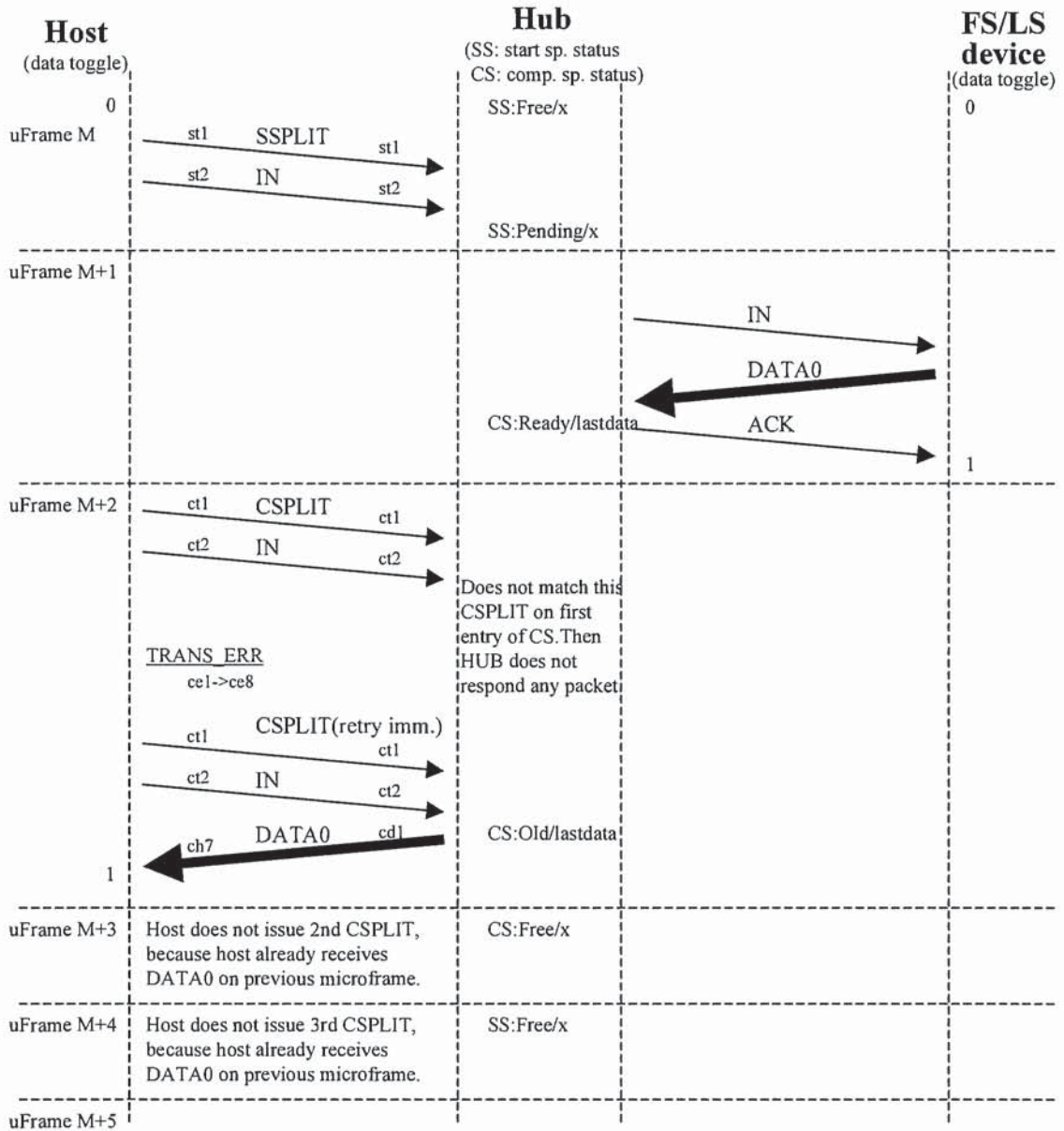


Figure A-73. Searching No Smash

Universal Serial Bus Specification Revision 2.0

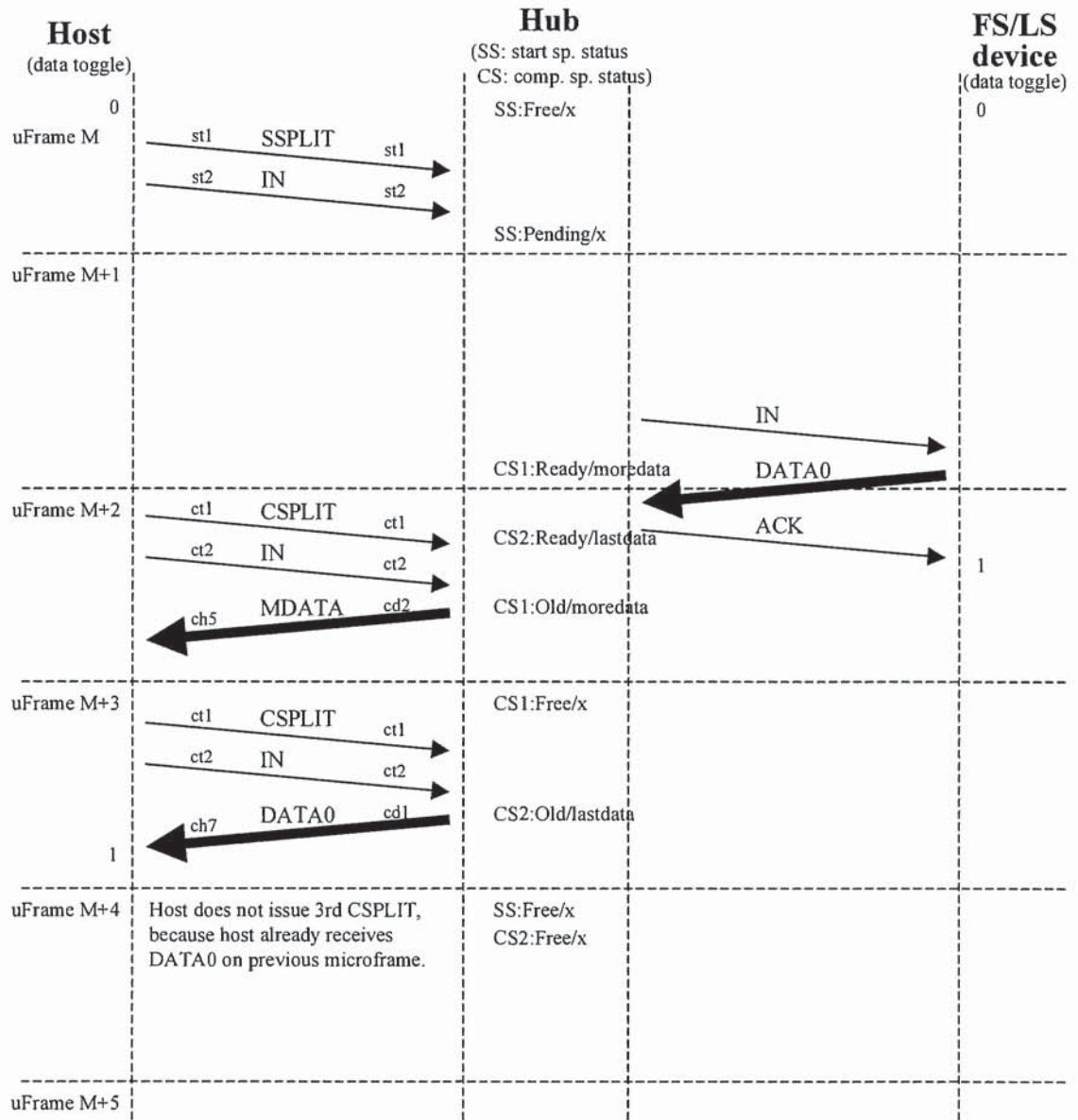


Figure A-74. CS Earlier No Smash(HS MDATA and FS/LS Data Packet is on M+1 and M+2)

Universal Serial Bus Specification Revision 2.0

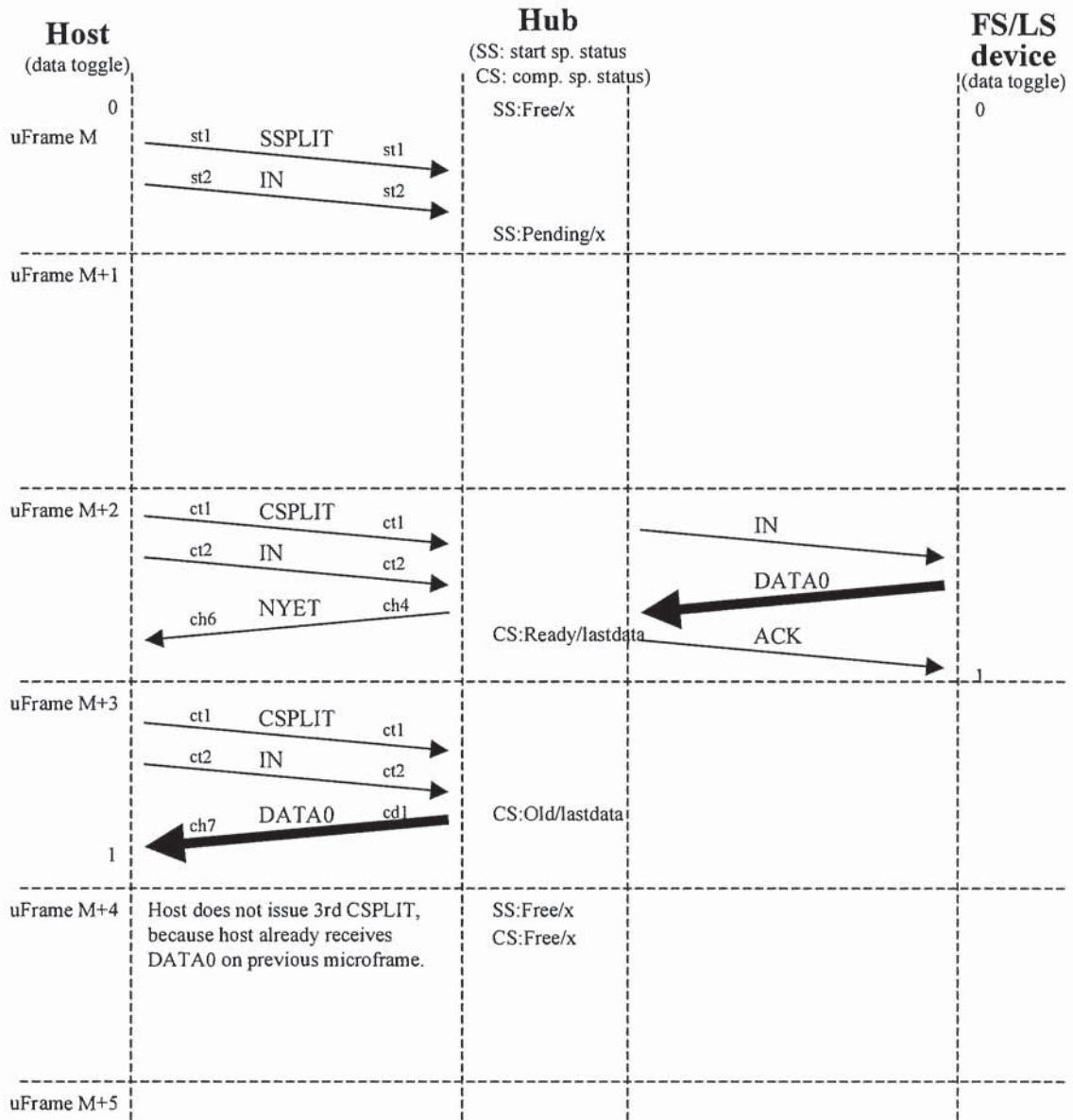


Figure A-75. CS Earlier No Smash(HS NYET and FS/LS Data Packet is on M+2)

Universal Serial Bus Specification Revision 2.0

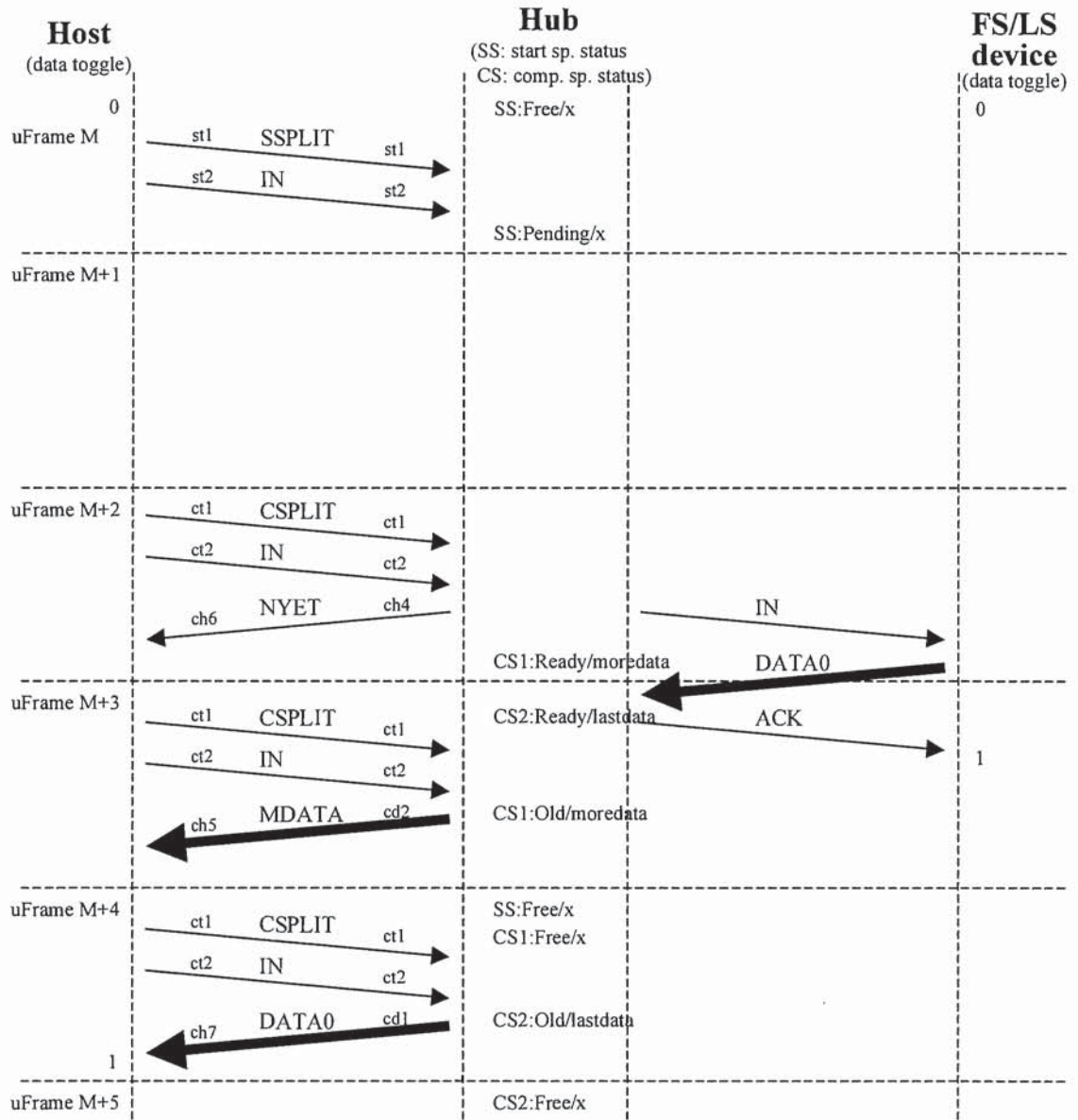


Figure A-76. CS Earlier No Smash(HS NYET and MDATA and FS/LS Data Packet is on M+2 and M+3)

Universal Serial Bus Specification Revision 2.0

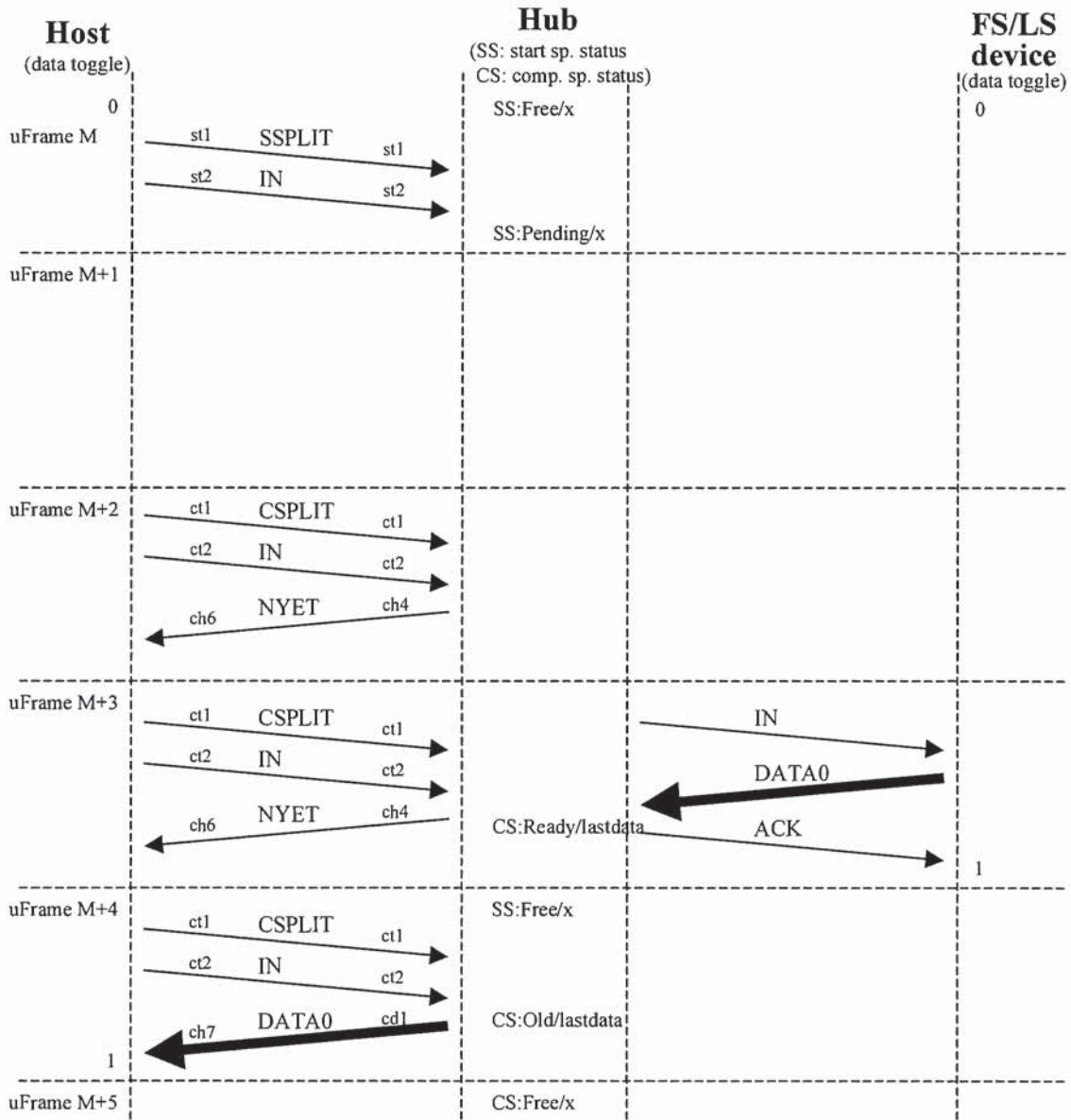


Figure A-77. CS Earlier No Smash(HS NYET and FS/LS Data Packet is on M+3)

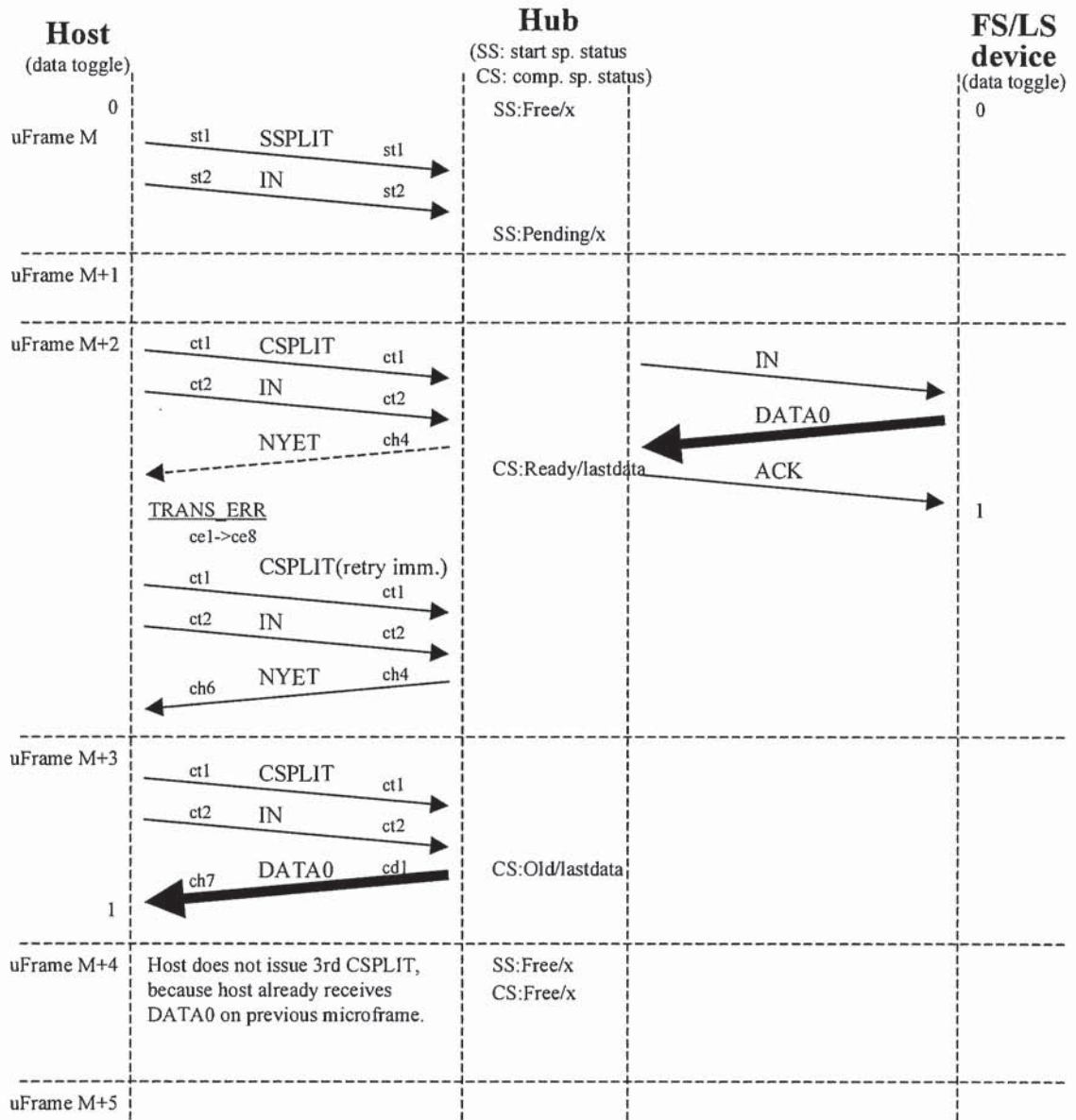


Figure A-78. CS Earlier HS NYET Smash

IN.CSPLIT earlier.HS NYET 3 strikes smash

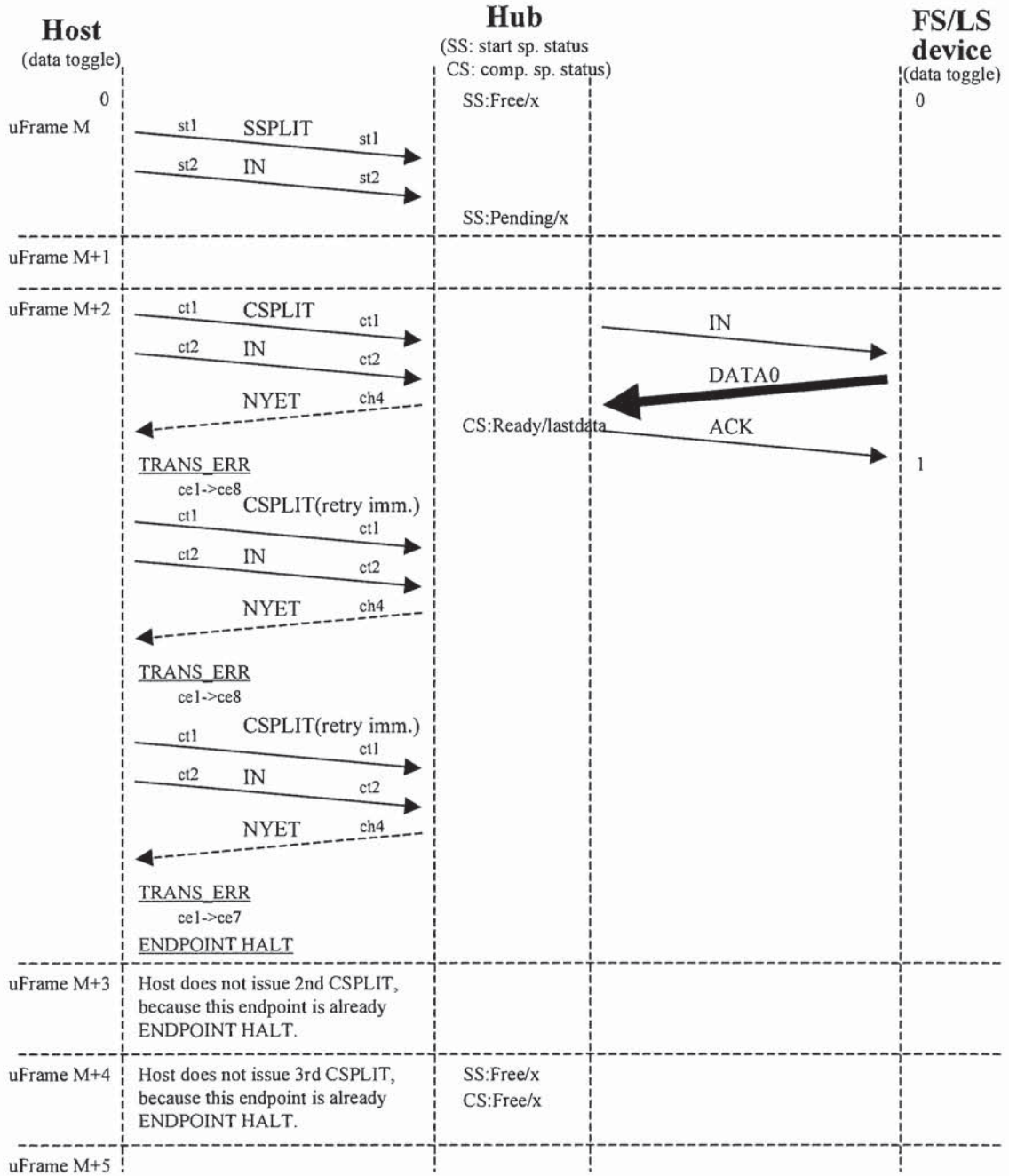


Figure A-79. CS Earlier HS NYET 3 Strikes Smash

Universal Serial Bus Specification Revision 2.0

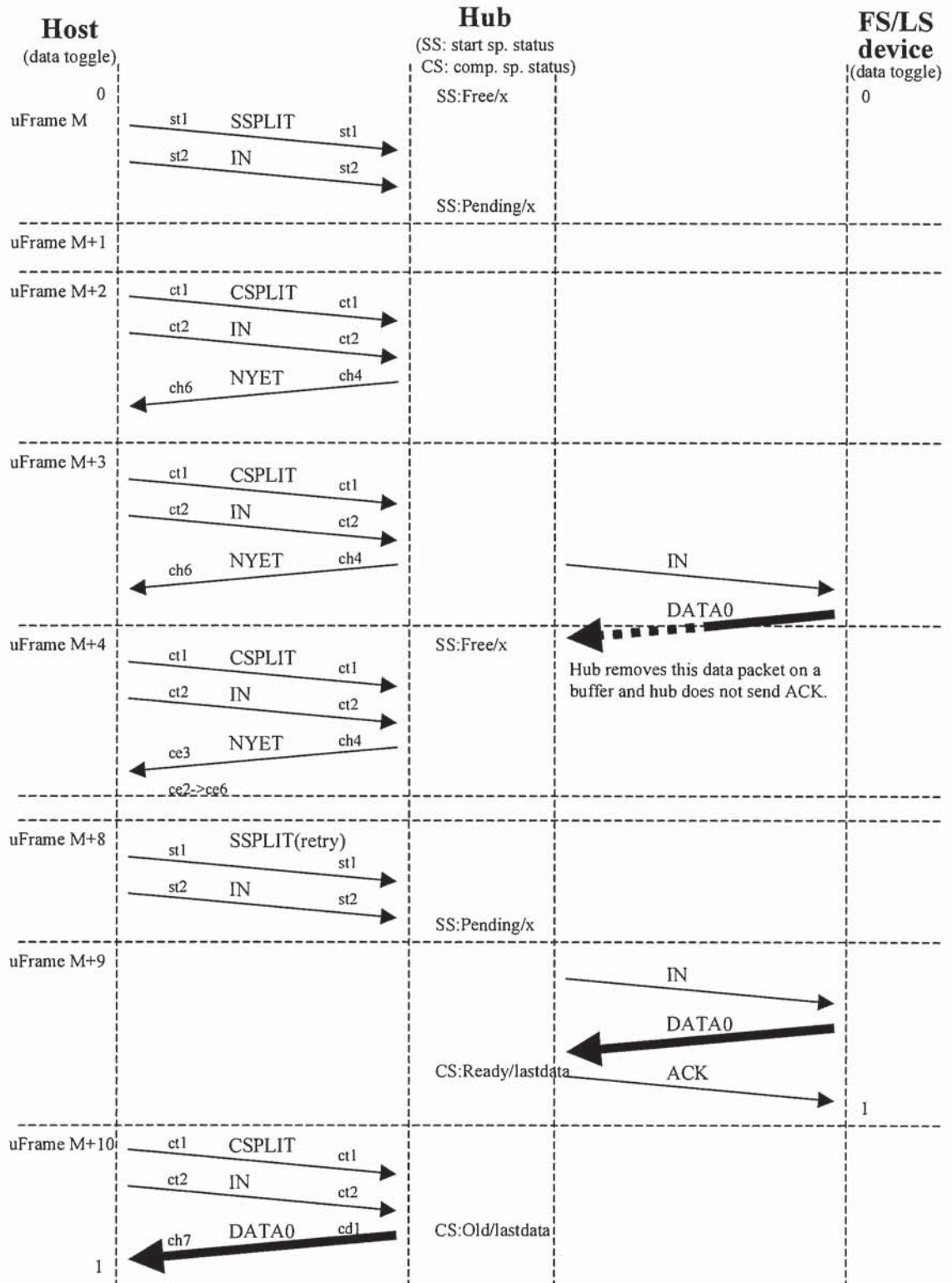


Figure A-80. Abort and Free Abort(HS NYET and FS/LS Transaction is Continued at End of M+3)

Universal Serial Bus Specification Revision 2.0

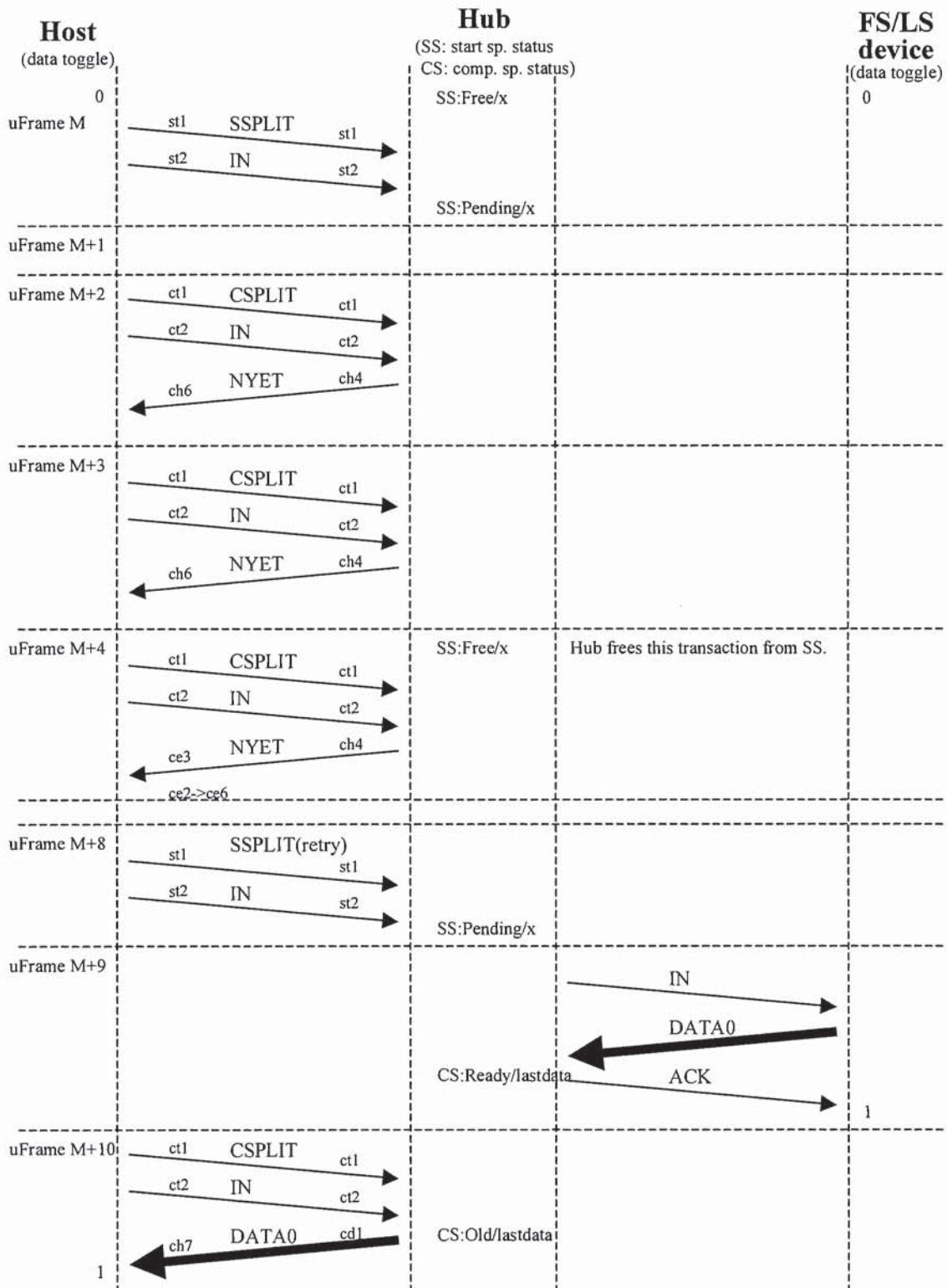


Figure A-81. Abort and Free Free(HS NYET and FS/LS Transaction is not Started at End of M+3)

Universal Serial Bus Specification Revision 2.0

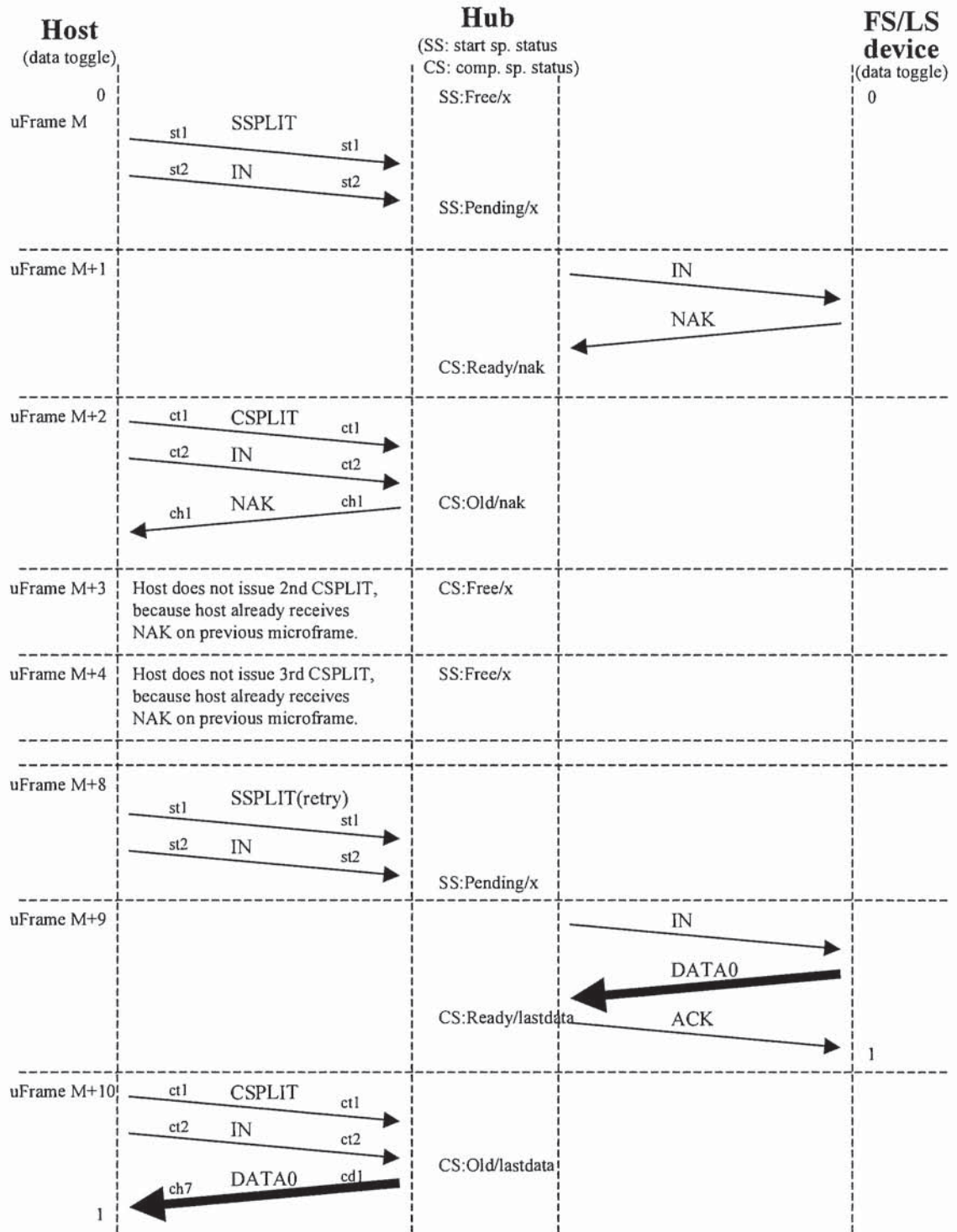


Figure A-82. Device Busy No Smash(FS/LS NAK)

Universal Serial Bus Specification Revision 2.0

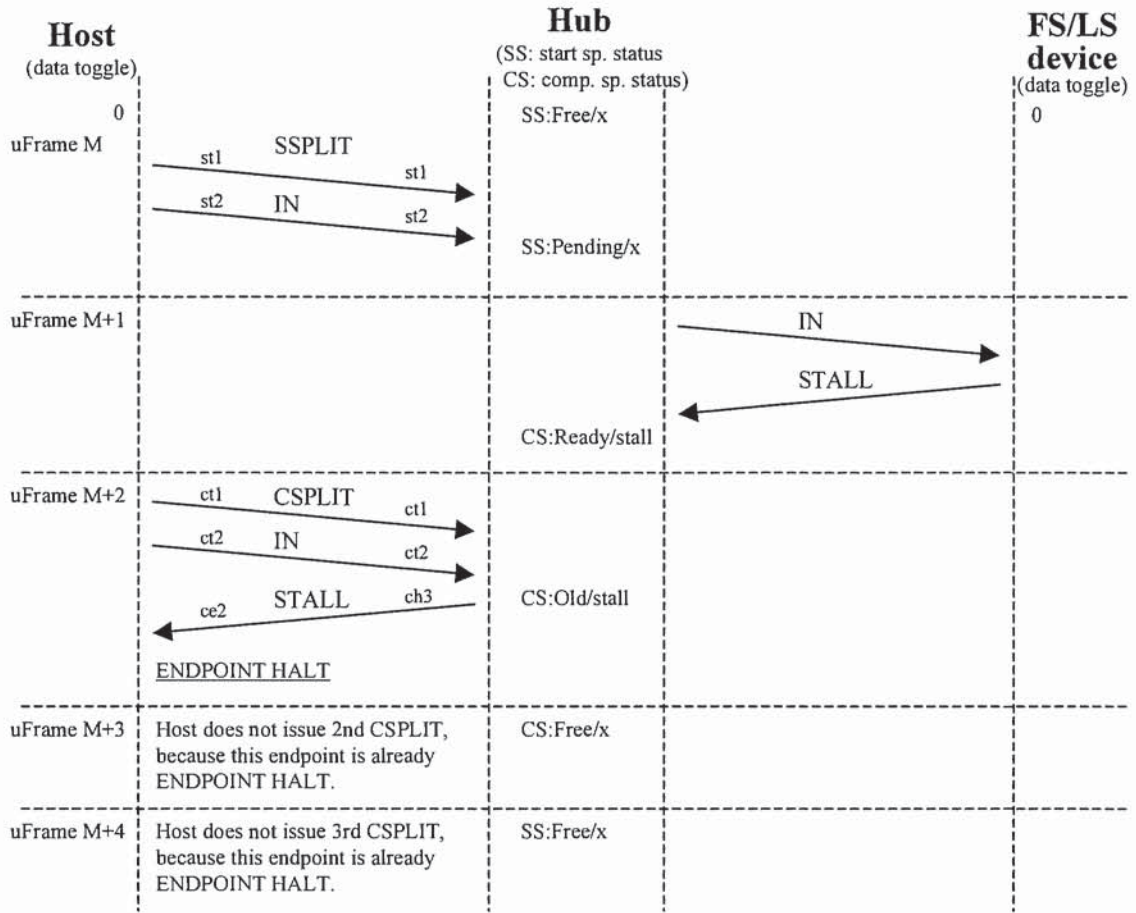
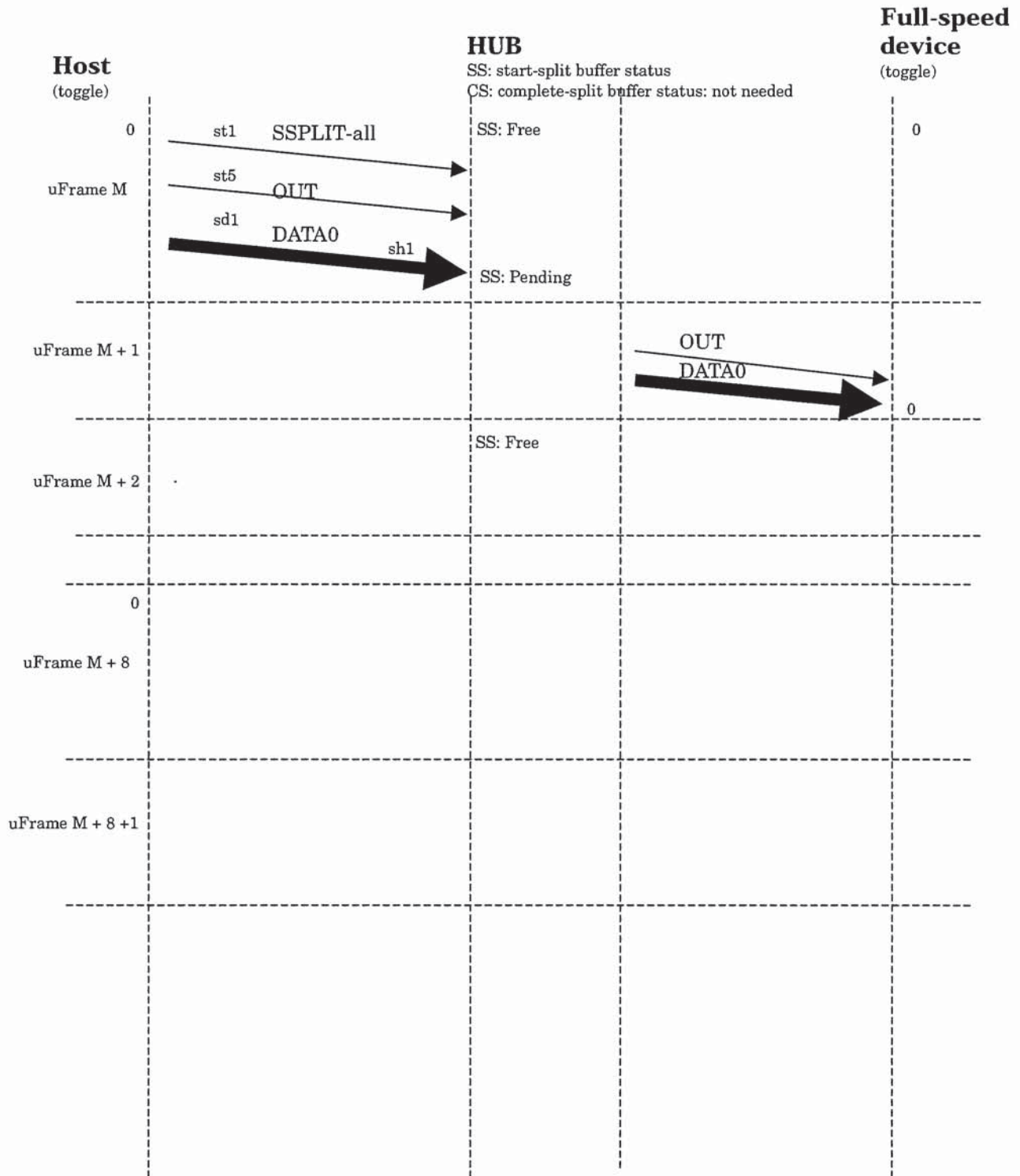


Figure A-83. Device Stall No Smash(FS/LS STALL)

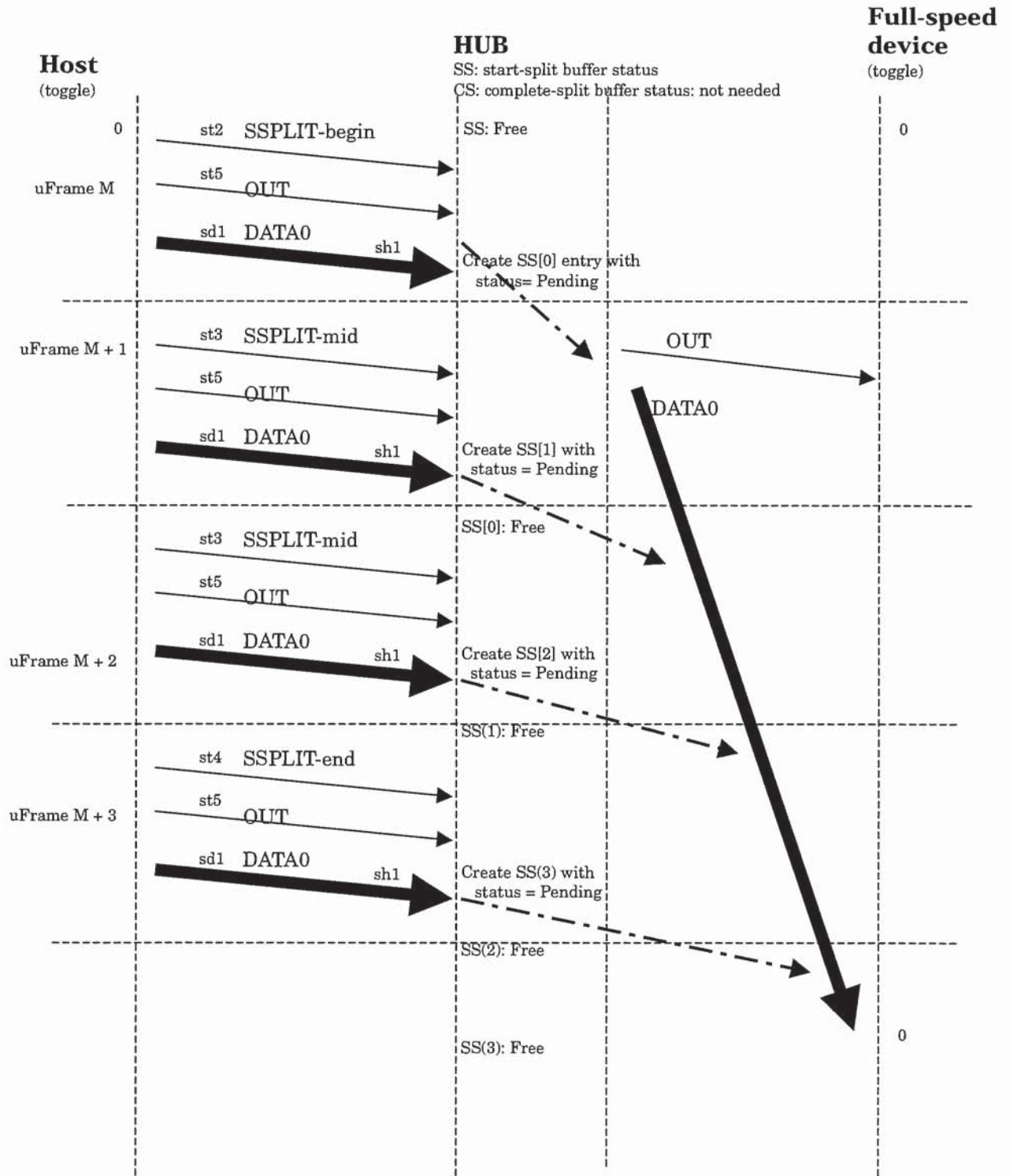
A.5 Isochronous OUT Split-transaction Examples

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

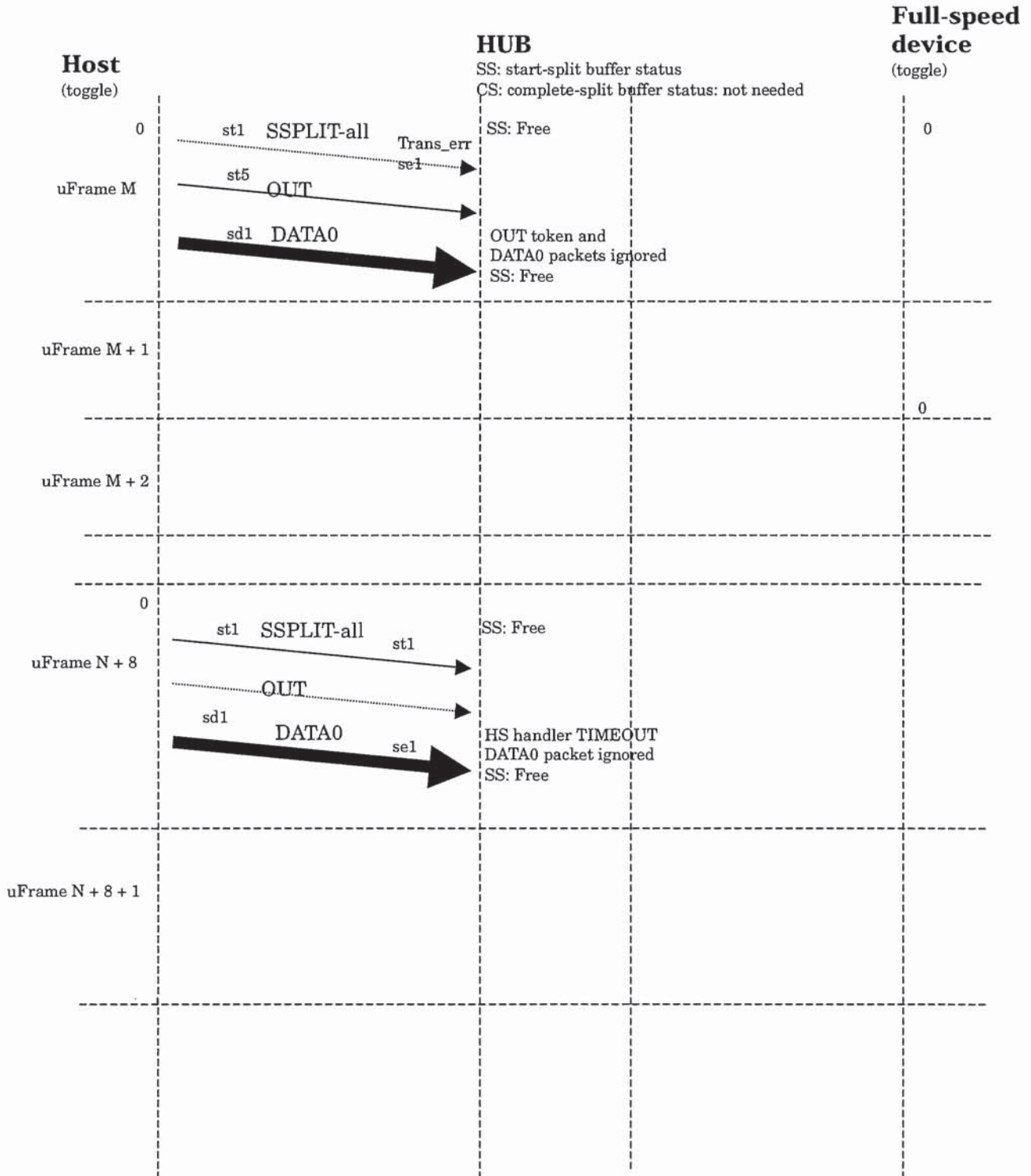
1) Normal. Payload <= 188 bytes:



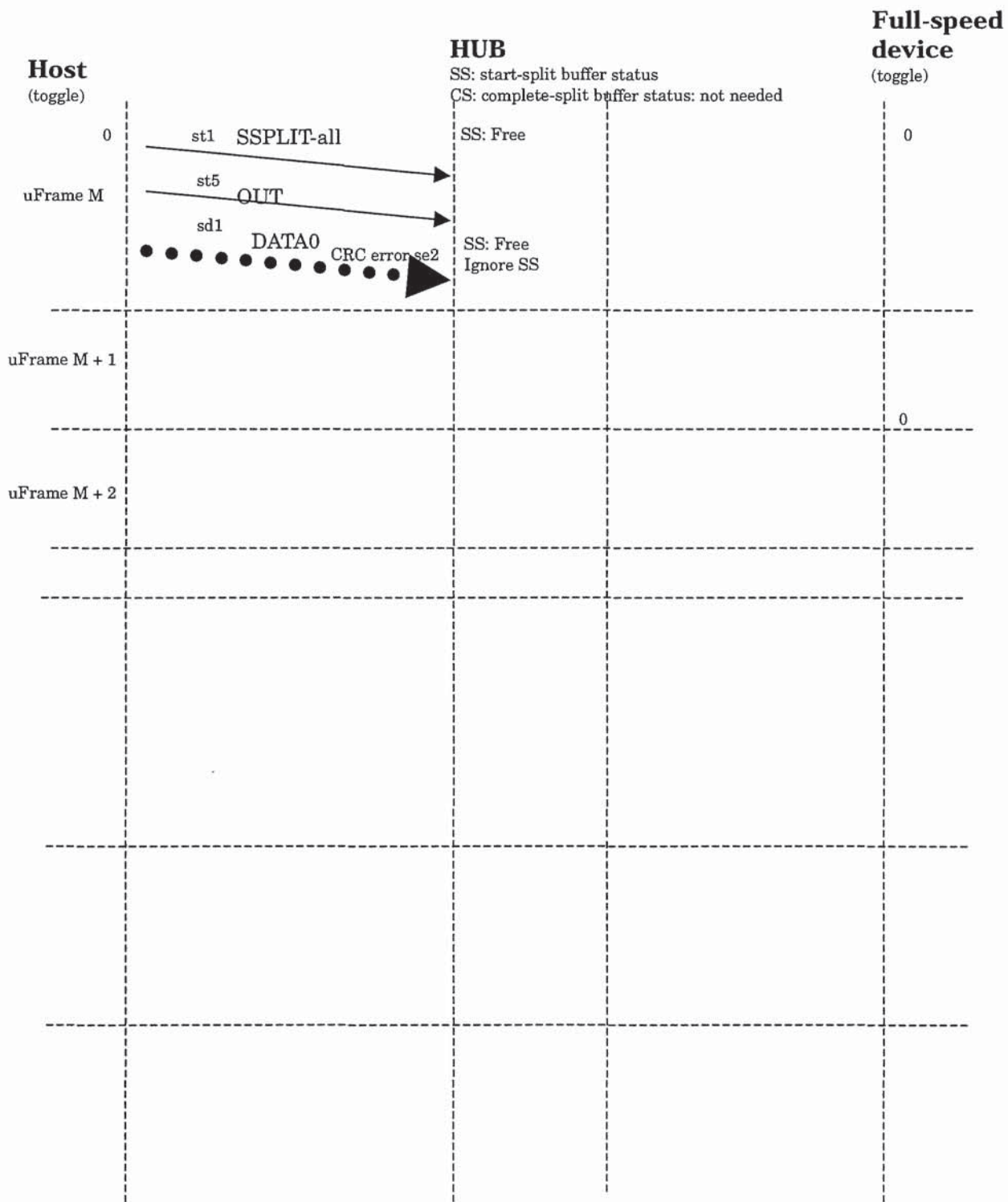
2) Normal. Payload > 188 Bytes



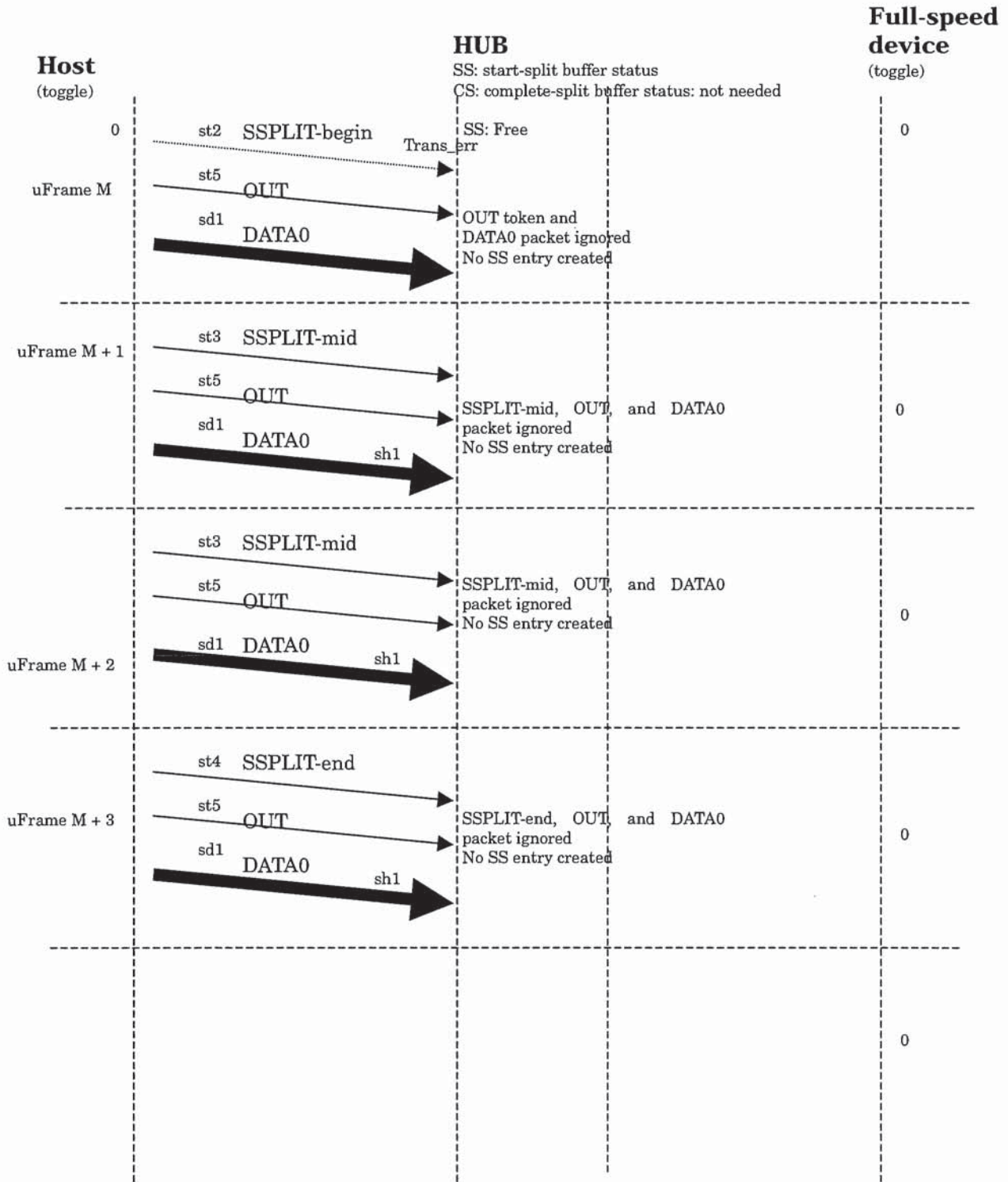
3) HS SSPLIT-all corrupted (missing or CRC error etc.)
 HS OUT corrupted



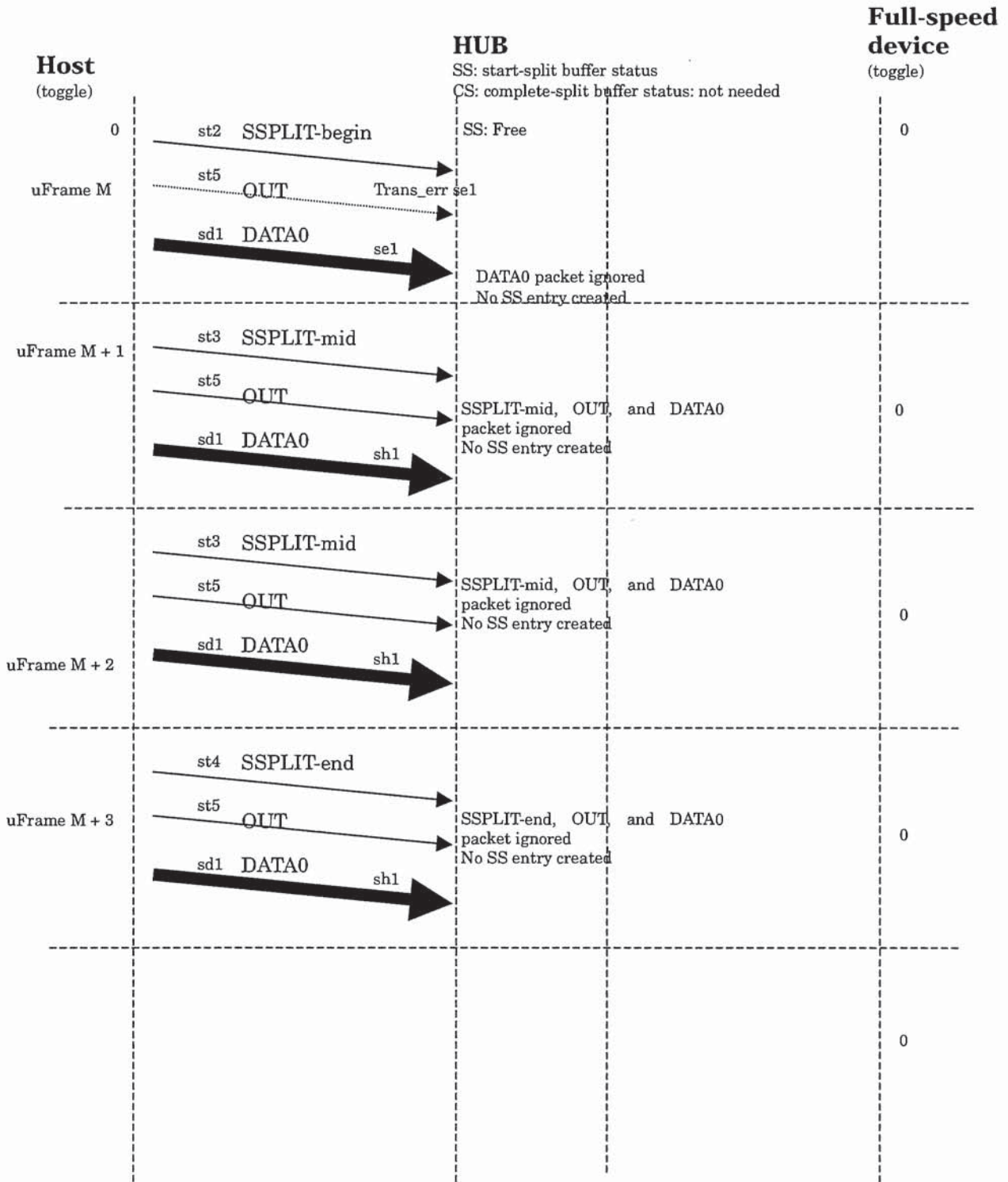
4) HS DATA0 corrupted



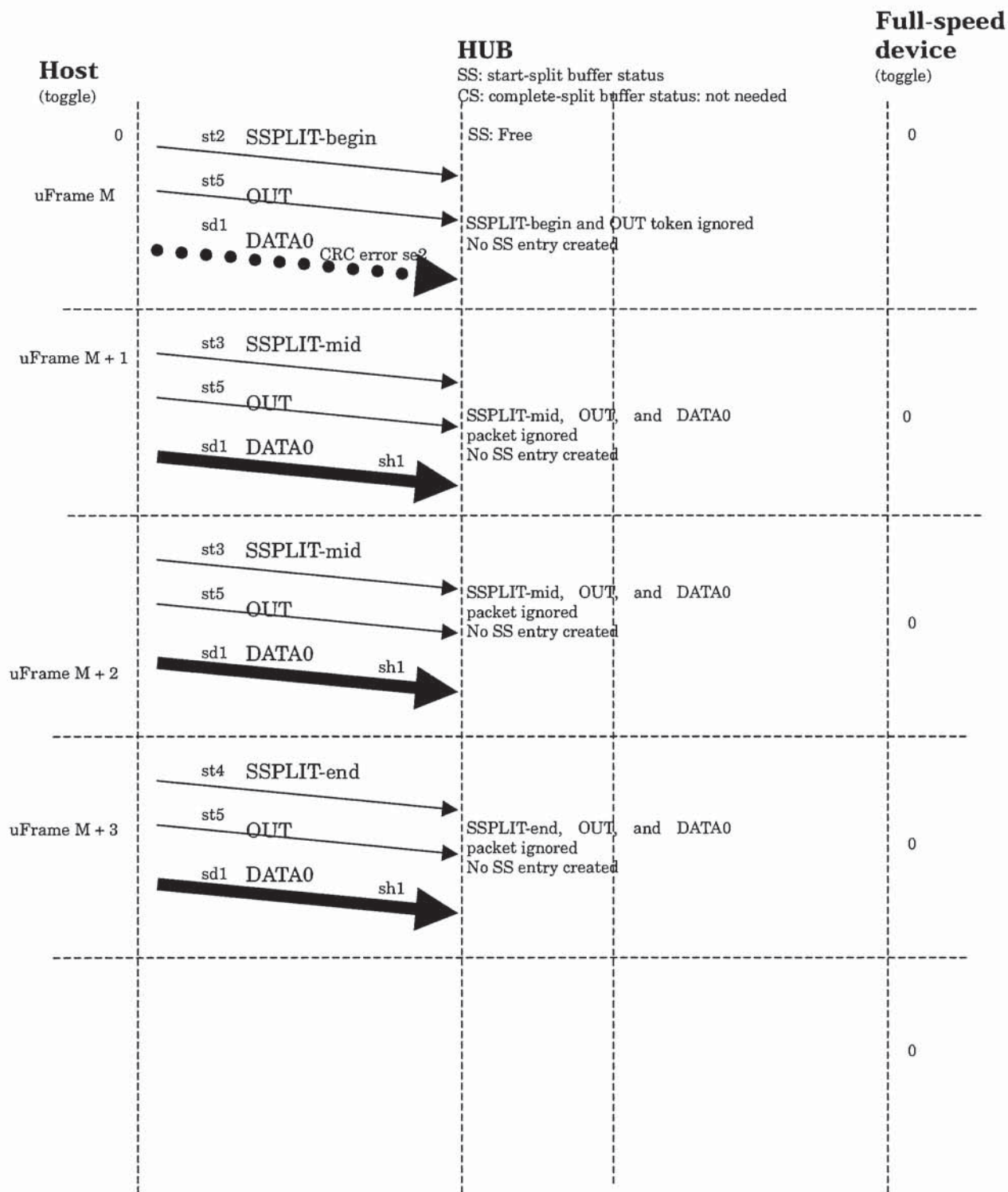
5) HS SSPLIT-begin corrupted (missing or CRC error etc.)



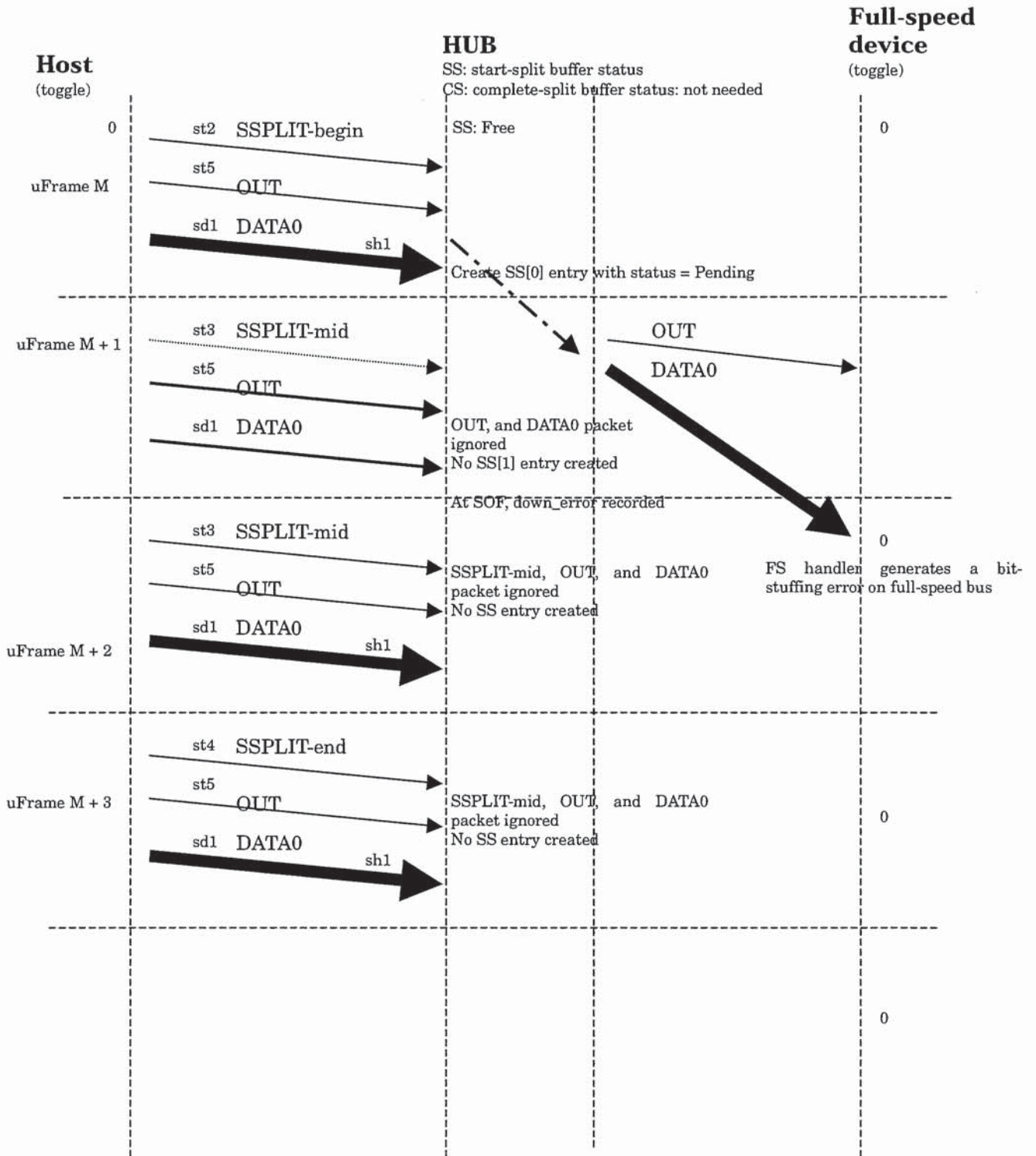
6) HS OUT after the HS SSPLIT-begin is corrupted



7) HS DATA0 corrupted



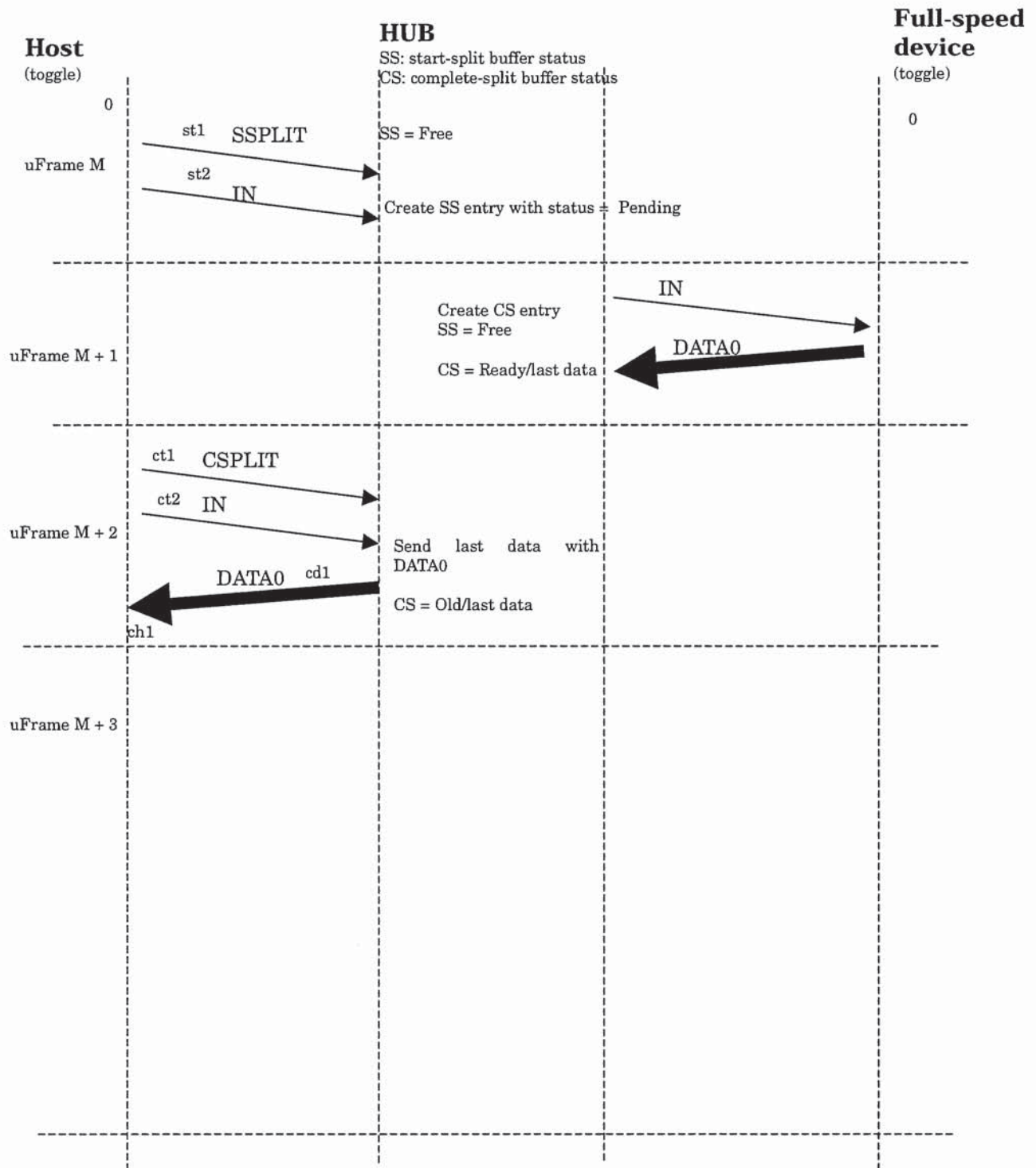
8) HS SSPLIT-mid or OUT token or DATA0 packet after it is corrupted



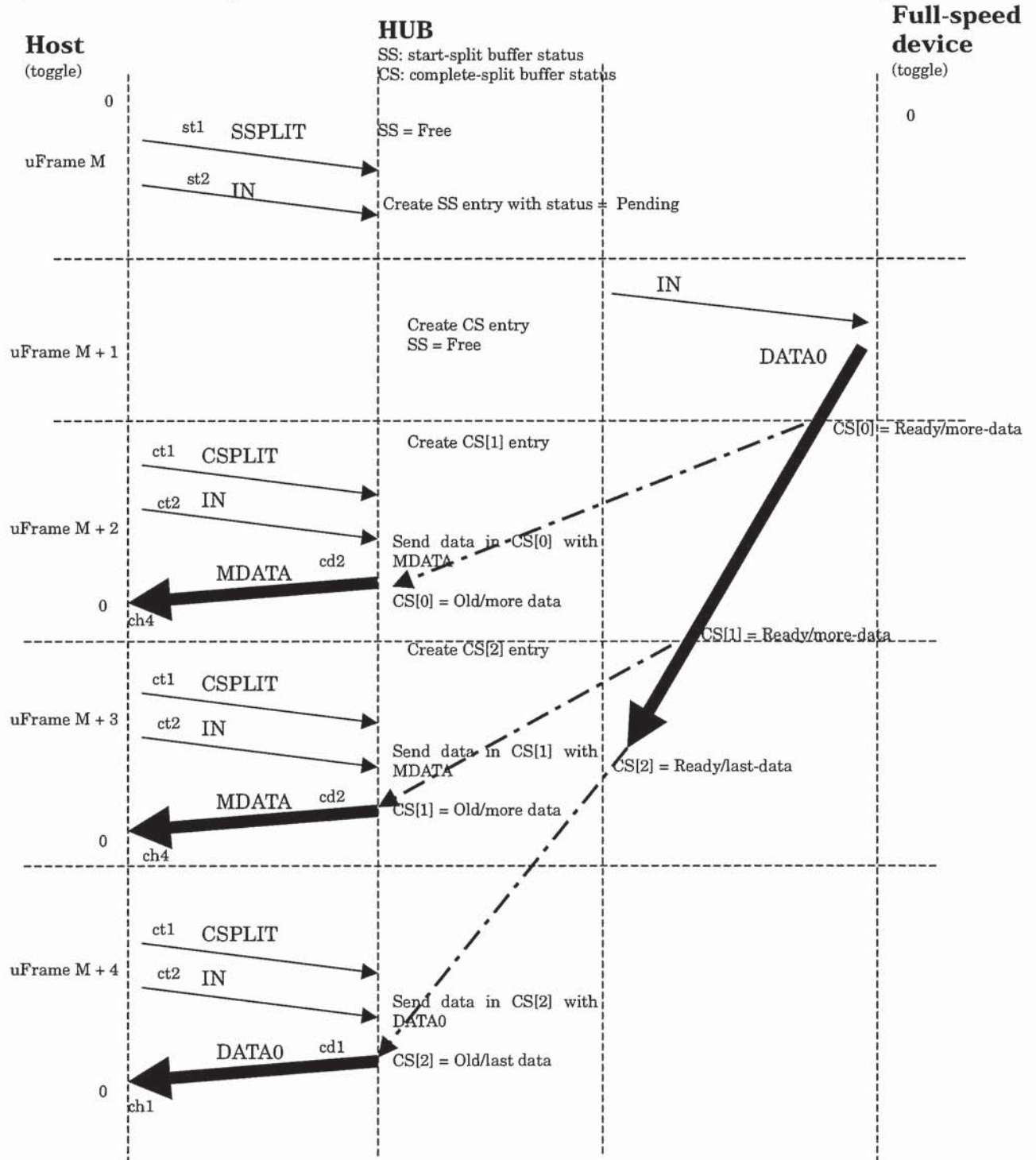
A.6 Isochronous IN Split-transaction Examples

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

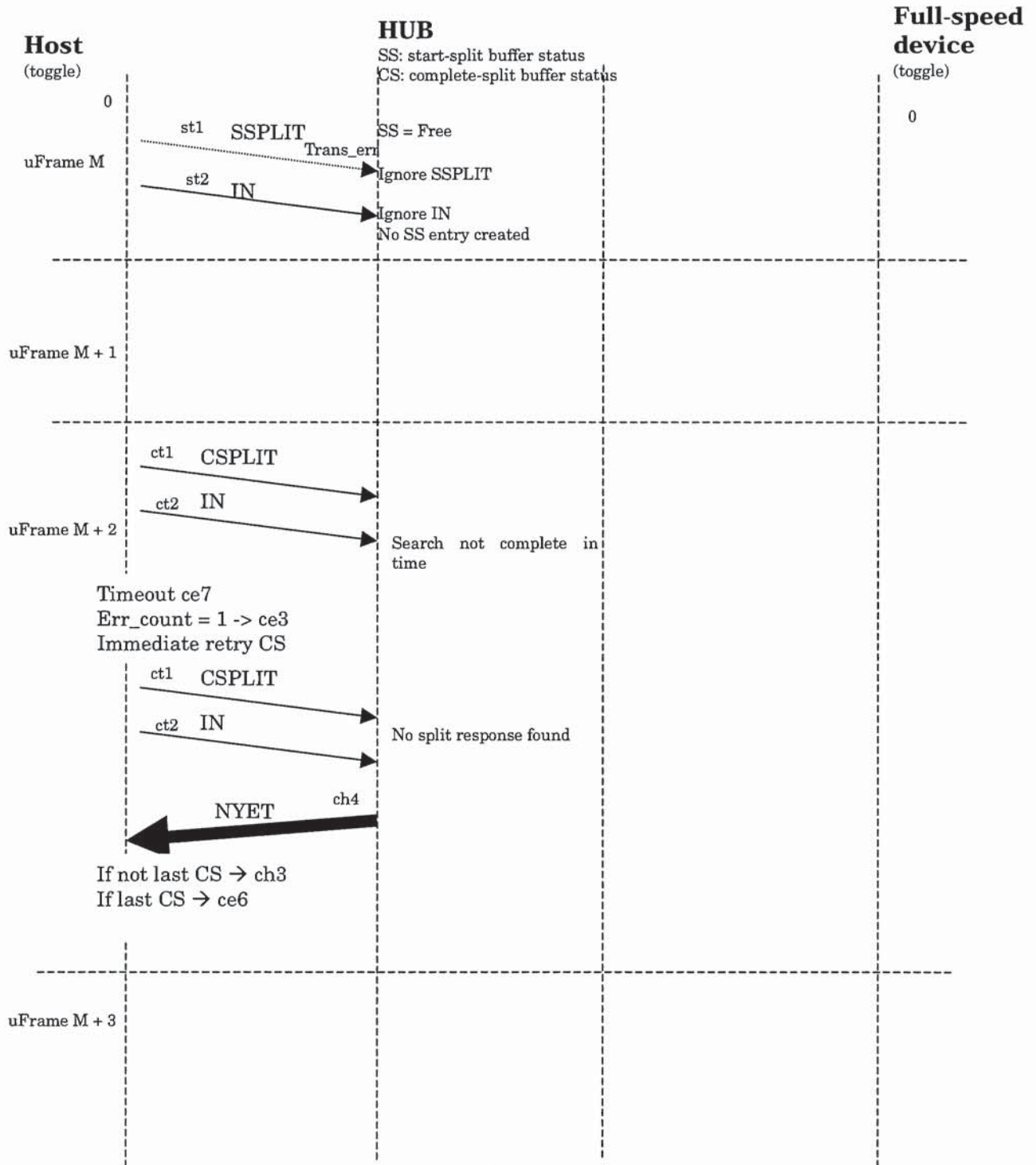
1) Normal: full-speed bus transaction does not cross microframe boundary



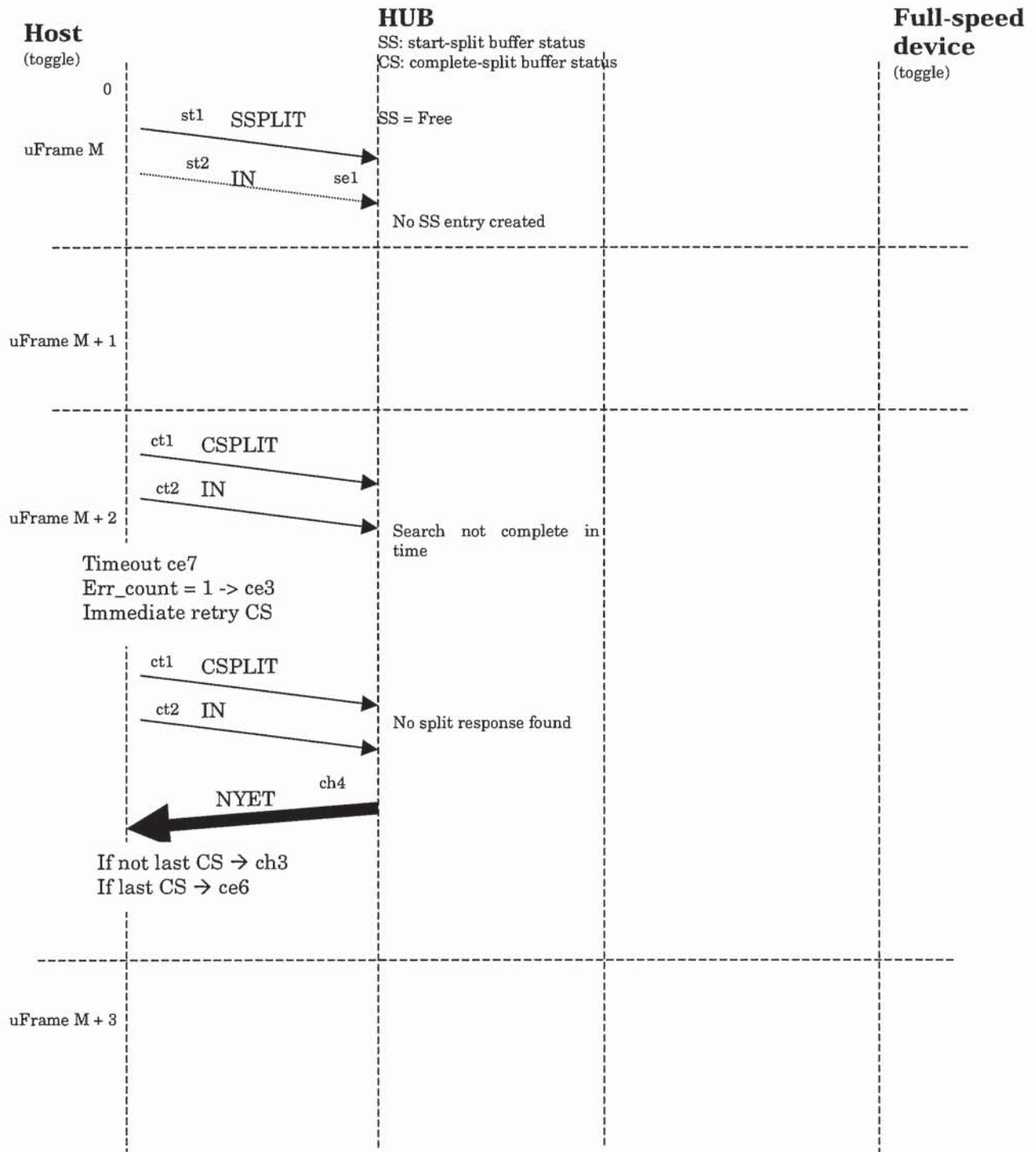
2) Normal: full-speed bus transaction crosses microframe boundary



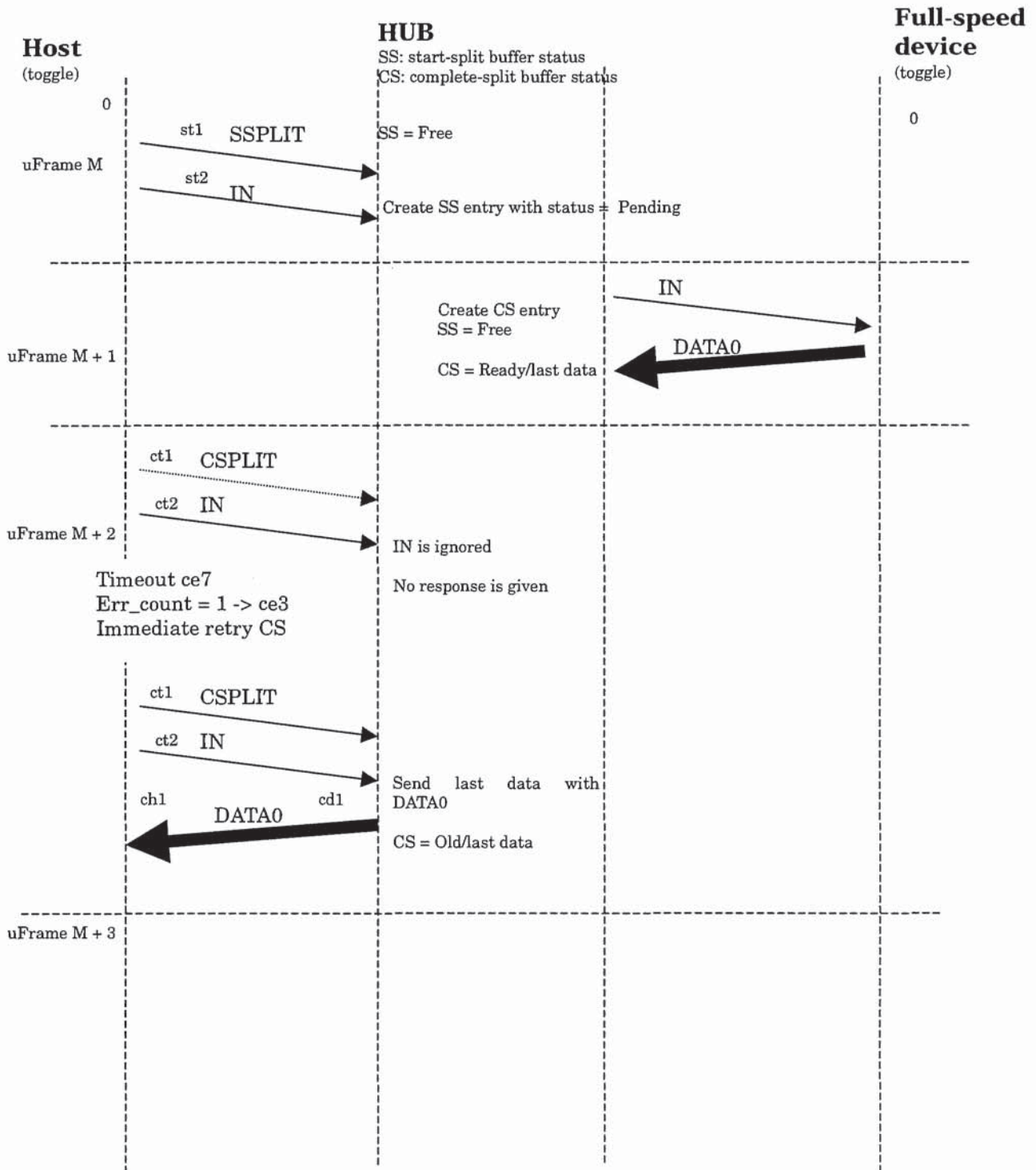
3) HS SSPLIT corrupted



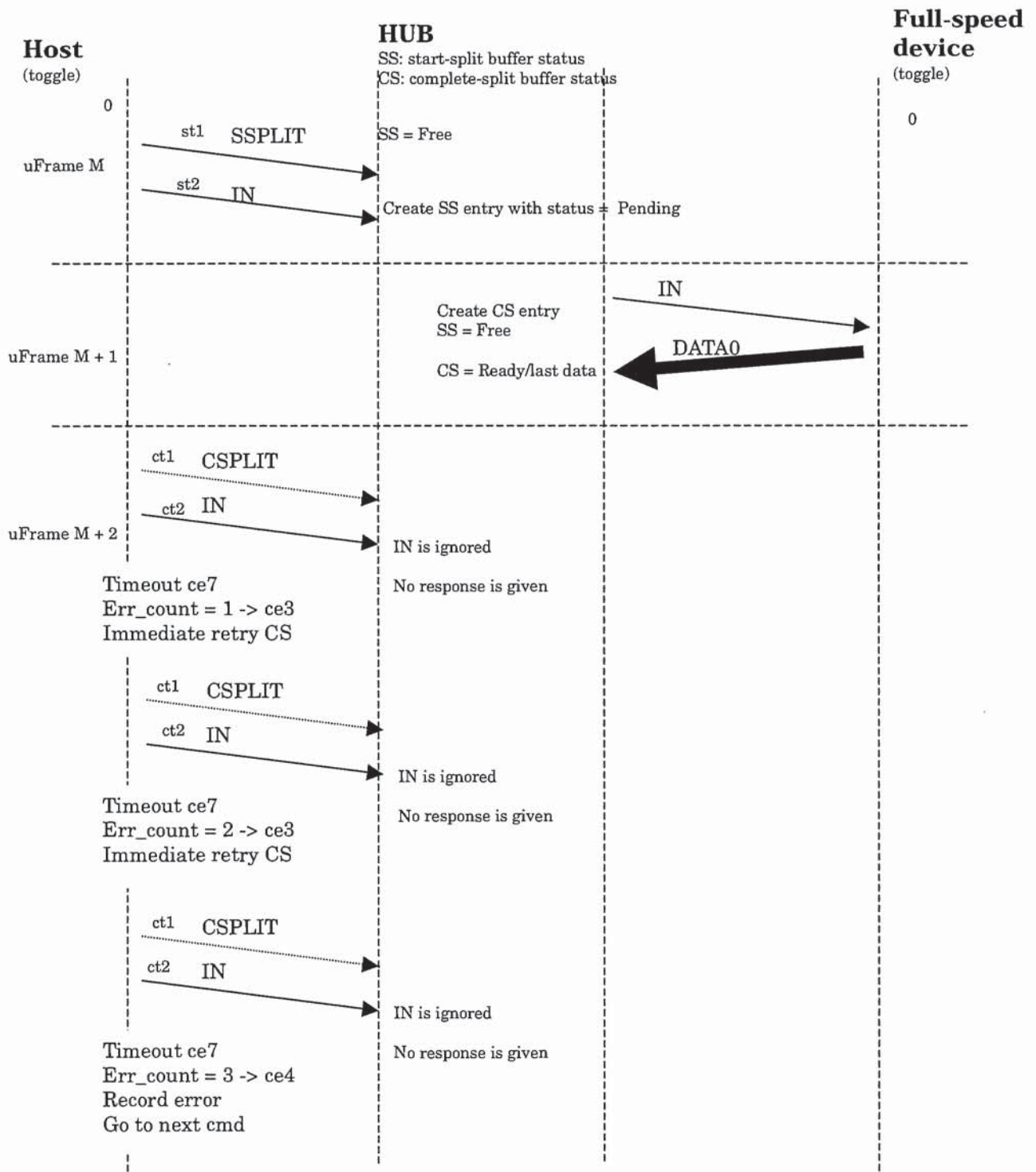
4) IN after HS SSPLIT corrupted



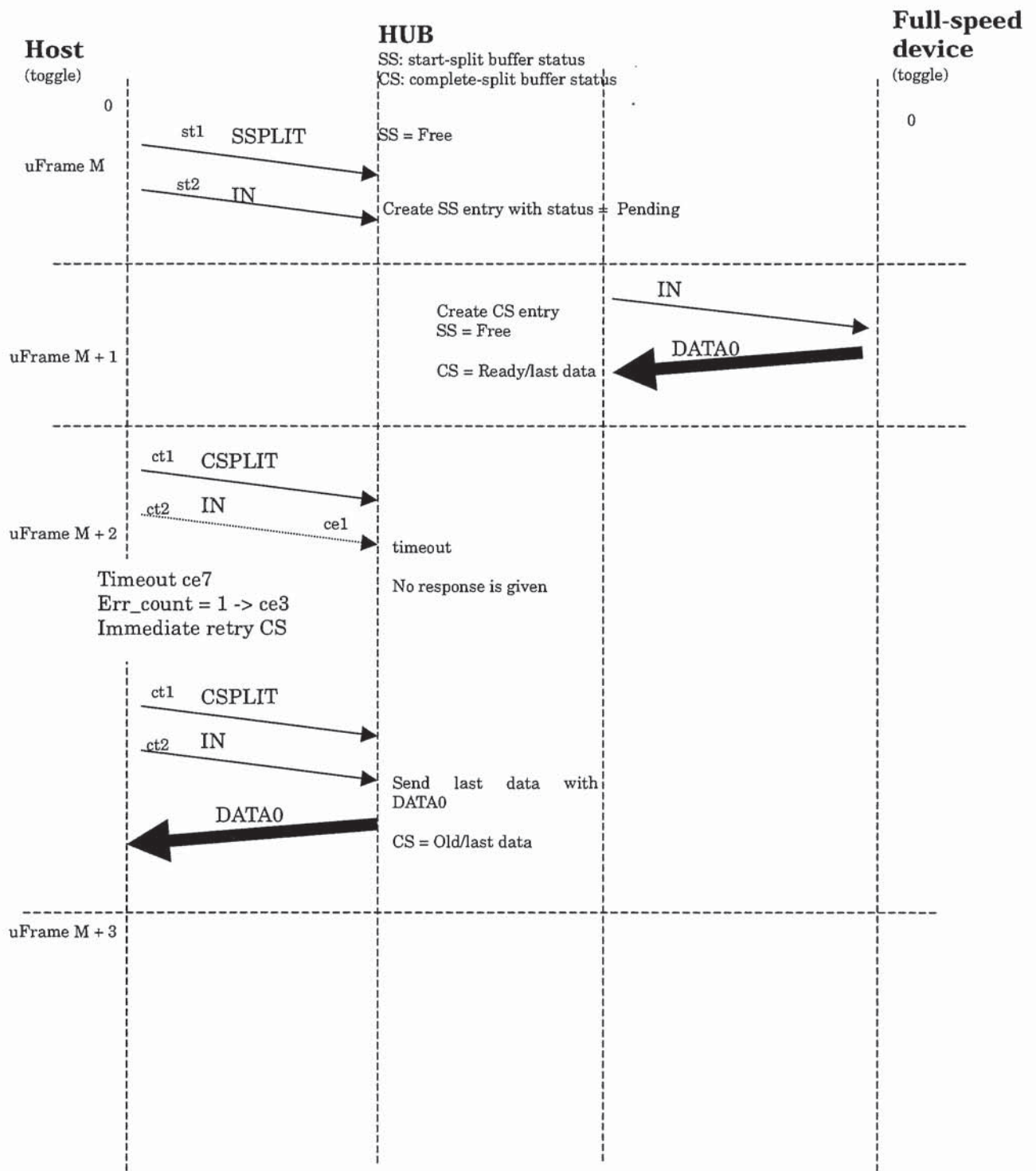
5) HS CSPLIT corrupted



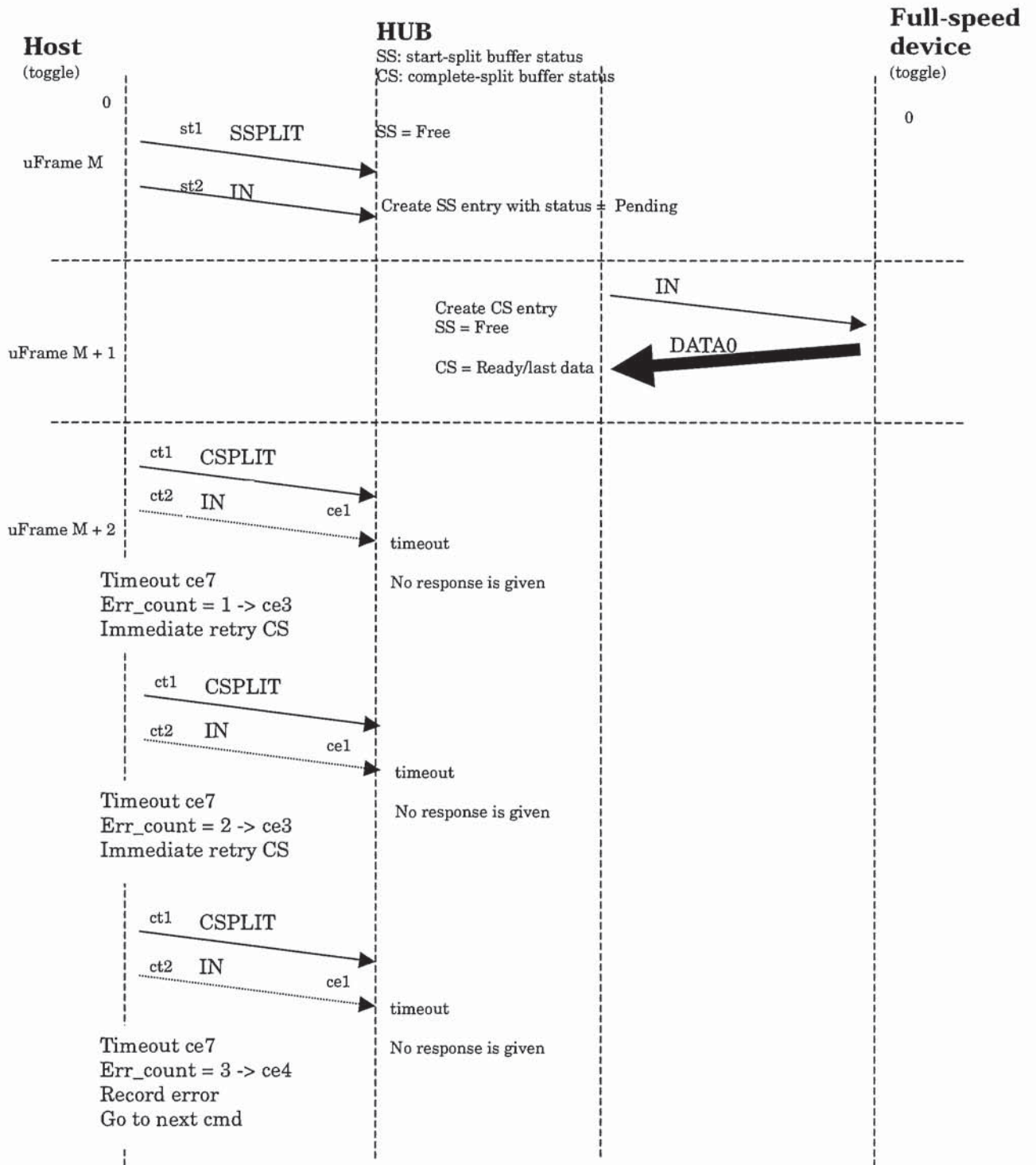
6) Consecutive HS CSPLIT corrupted



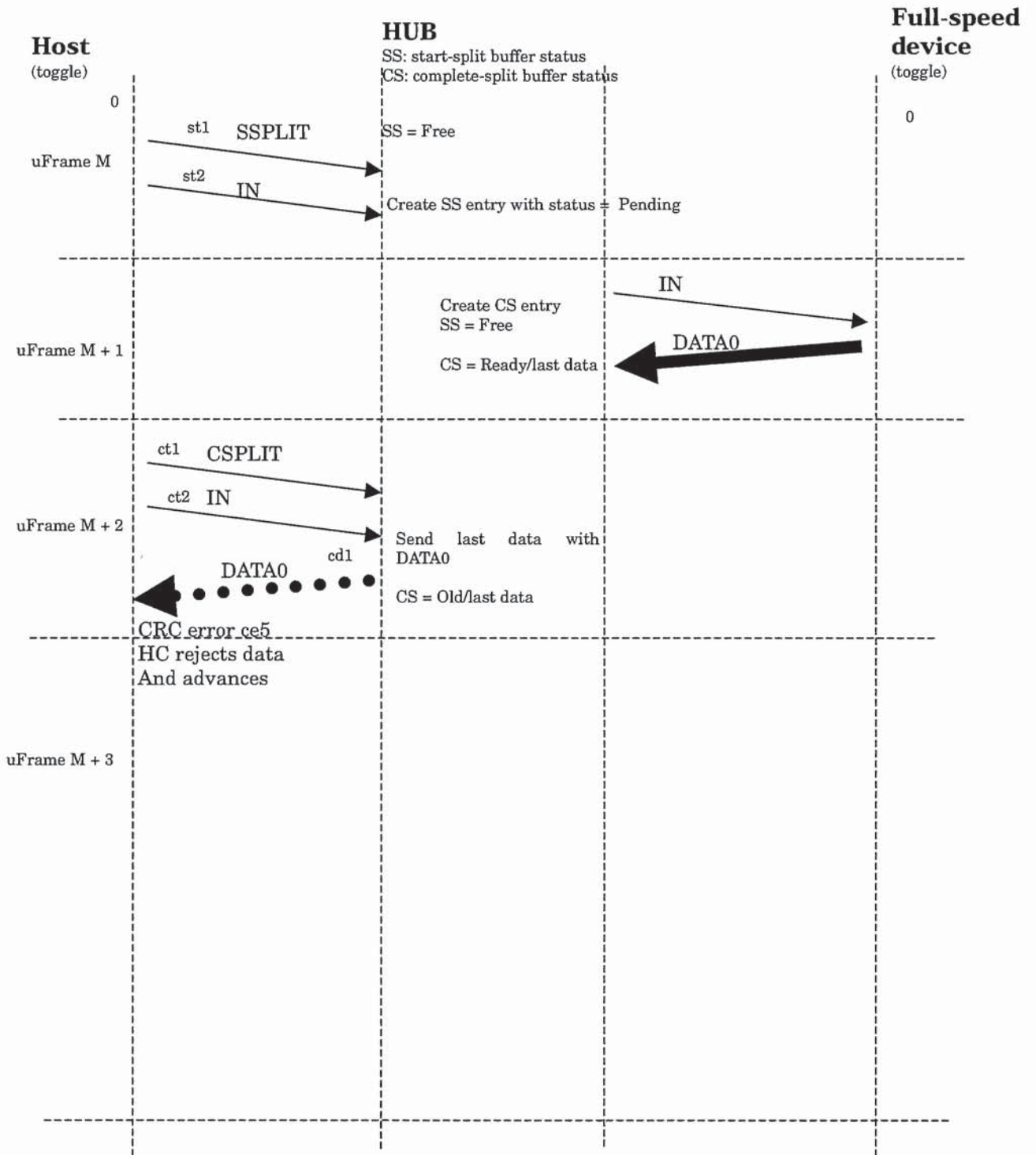
7) HS IN corrupted



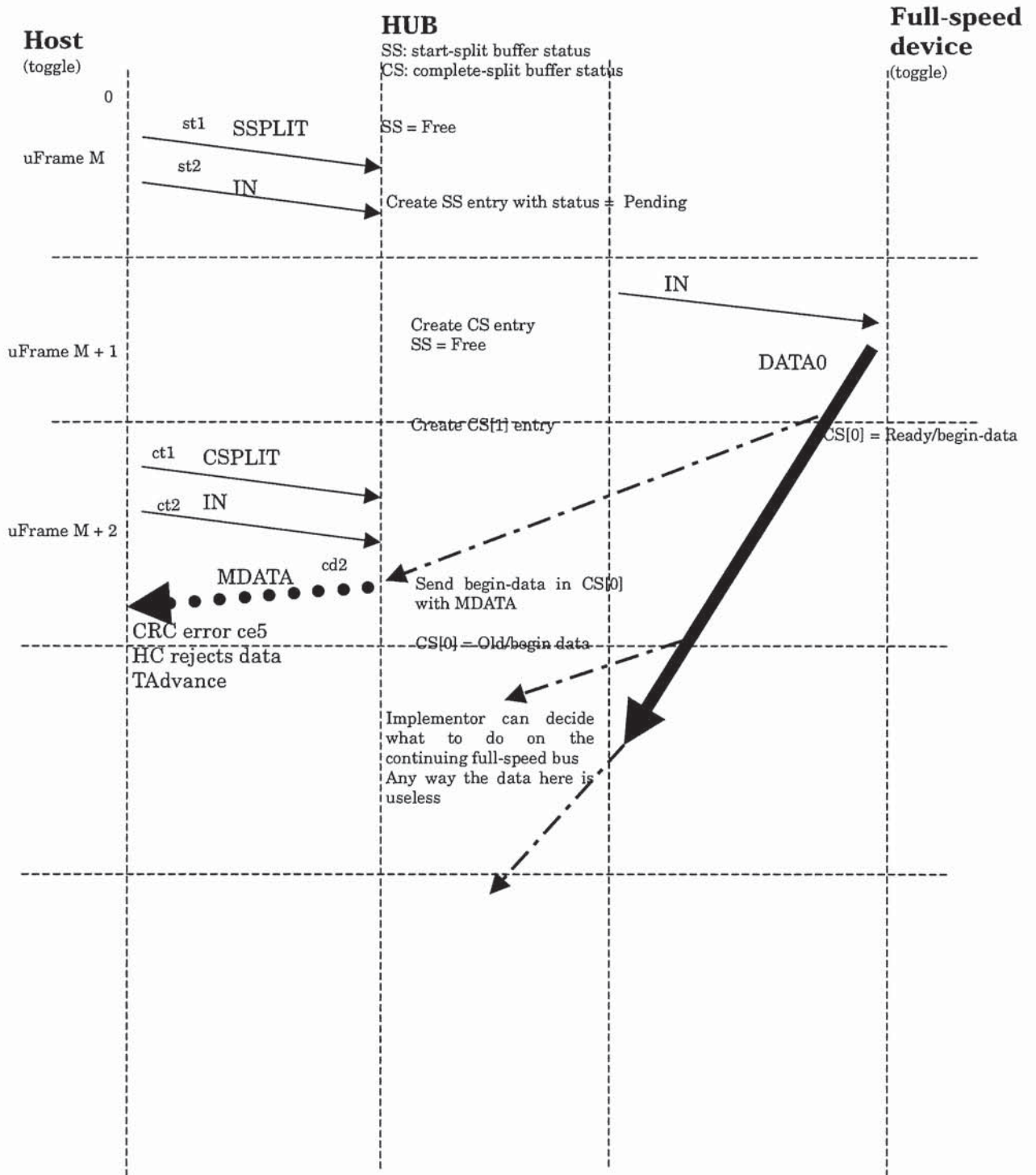
8) Consecutive HS IN corrupted



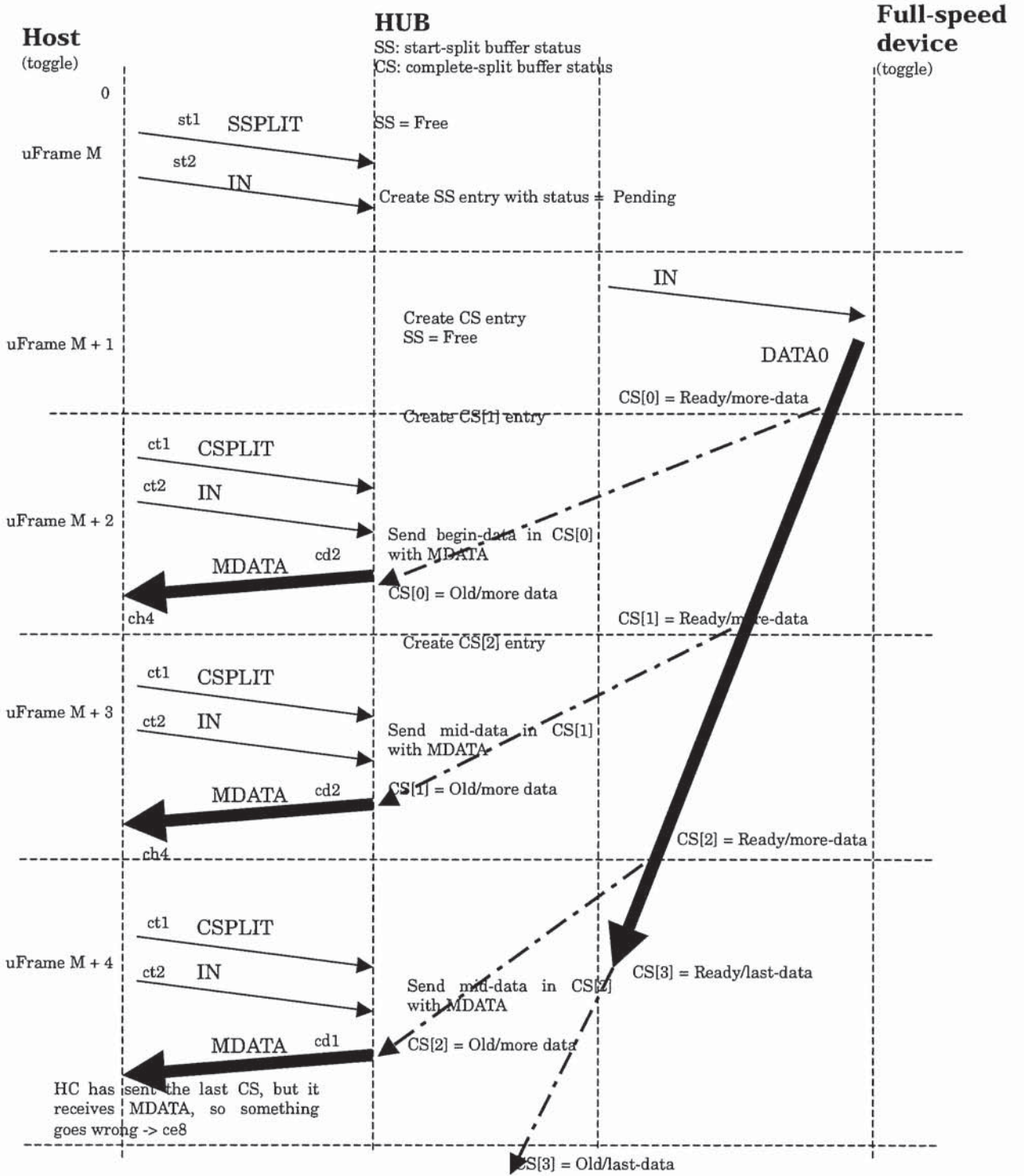
9) HS data corrupted (case 1)



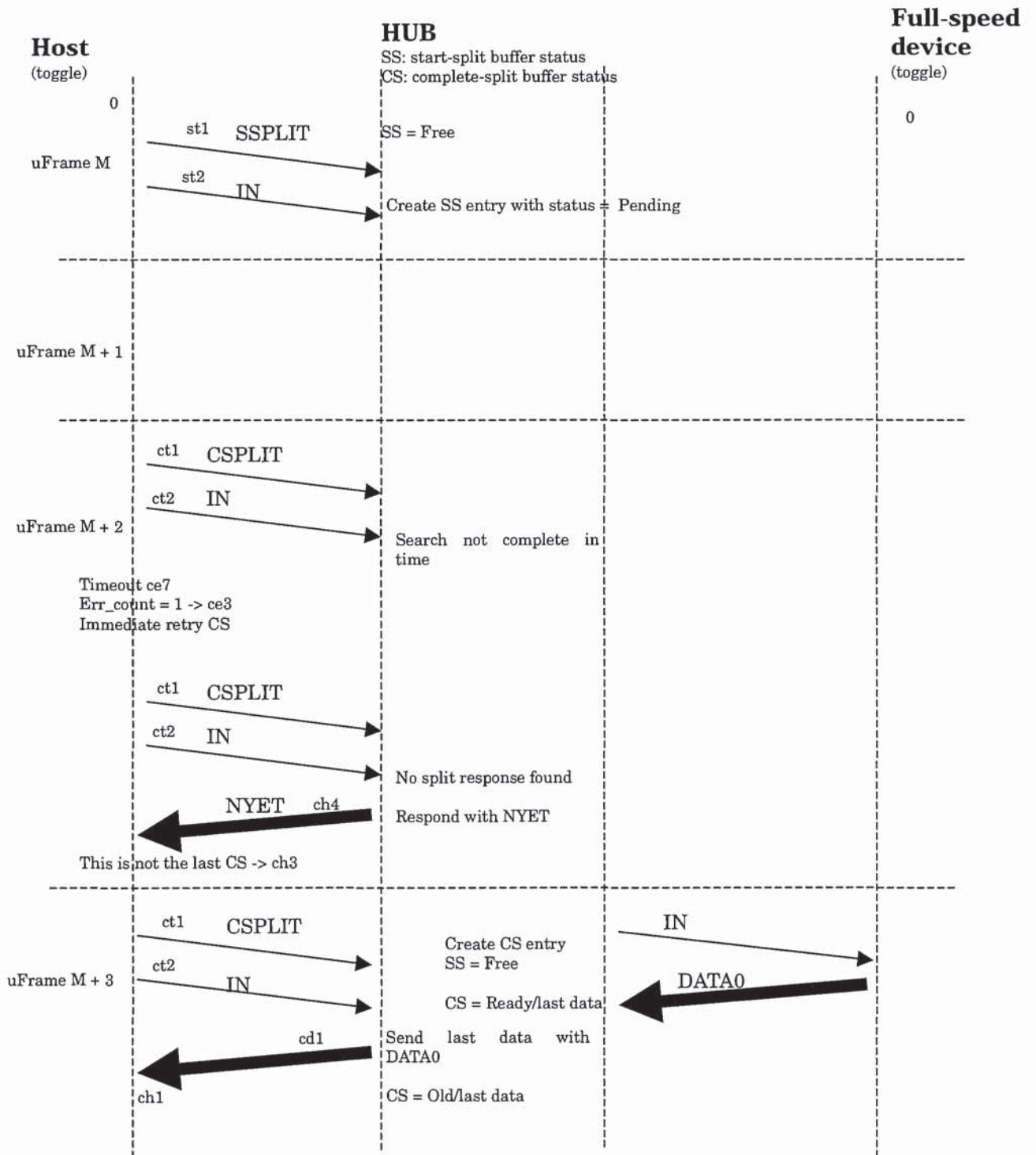
10) HS data corrupted (case 2)



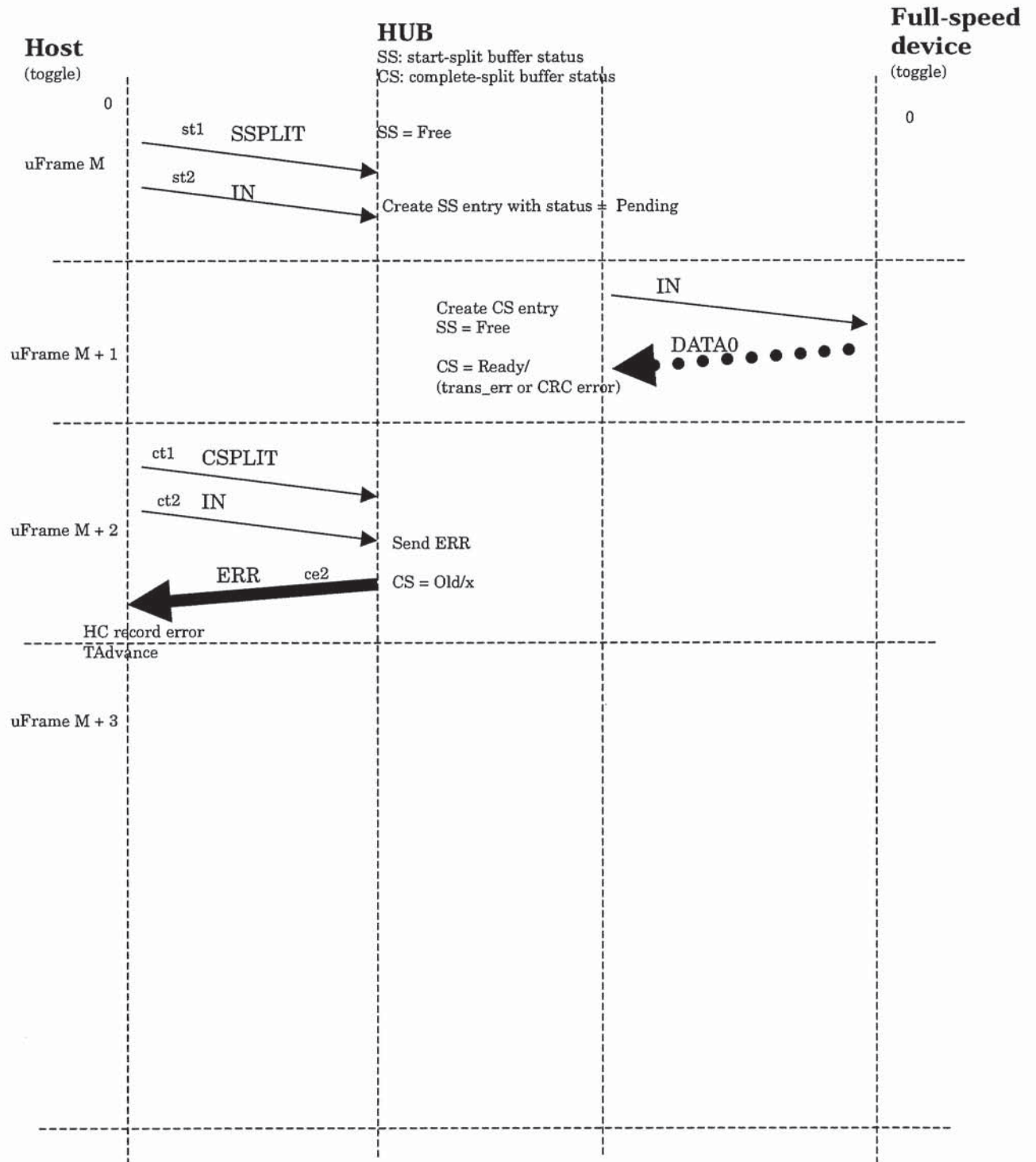
11) TT has more data than HC expects



12) HS CS too early (full-speed data not available yet)



13) Full-speed timeout or CRC error



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 immediately 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
    );

```

Universal Serial Bus Specification Revision 2.0

```

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 DOWNT0 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;

```

Universal Serial Bus Specification Revision 2.0

```
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;
  isoch0:   boolean; -- was the last transaction an isochronous OUT SS
  lastdata: data_choices;
  -- if isoch0 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) DOWNT0 0)
  of err_inject_rec;

TYPE cam_type IS ARRAY((FIFO_DEPTH - 1) DOWNT0 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;
```

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;

```

Universal Serial Bus Specification Revision 2.0

```
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;
```

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;
    END IF;

```

Universal Serial Bus Specification Revision 2.0

```

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.
-----

```


Universal Serial Bus Specification Revision 2.0

```
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 initialize 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
```

Universal Serial Bus Specification Revision 2.0

```
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 forwarding 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;
```


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 $2.5 \mu s$ (T_{FLT}) 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.

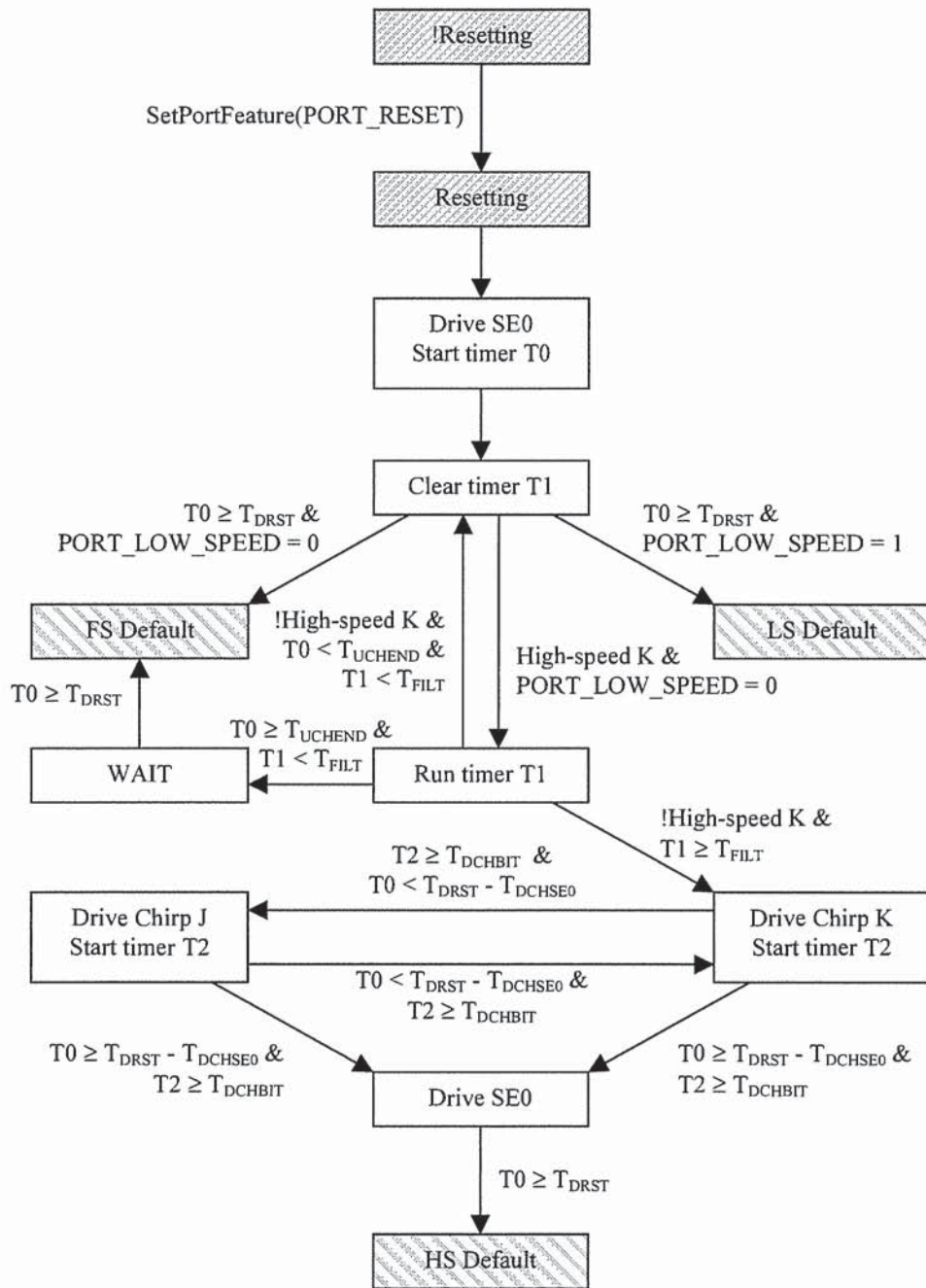


Figure C-1. Downstream Facing Port Reset Protocol State Diagram

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

40 to 60 μ s of continuous signaling it was busy with when the timer T_0 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.

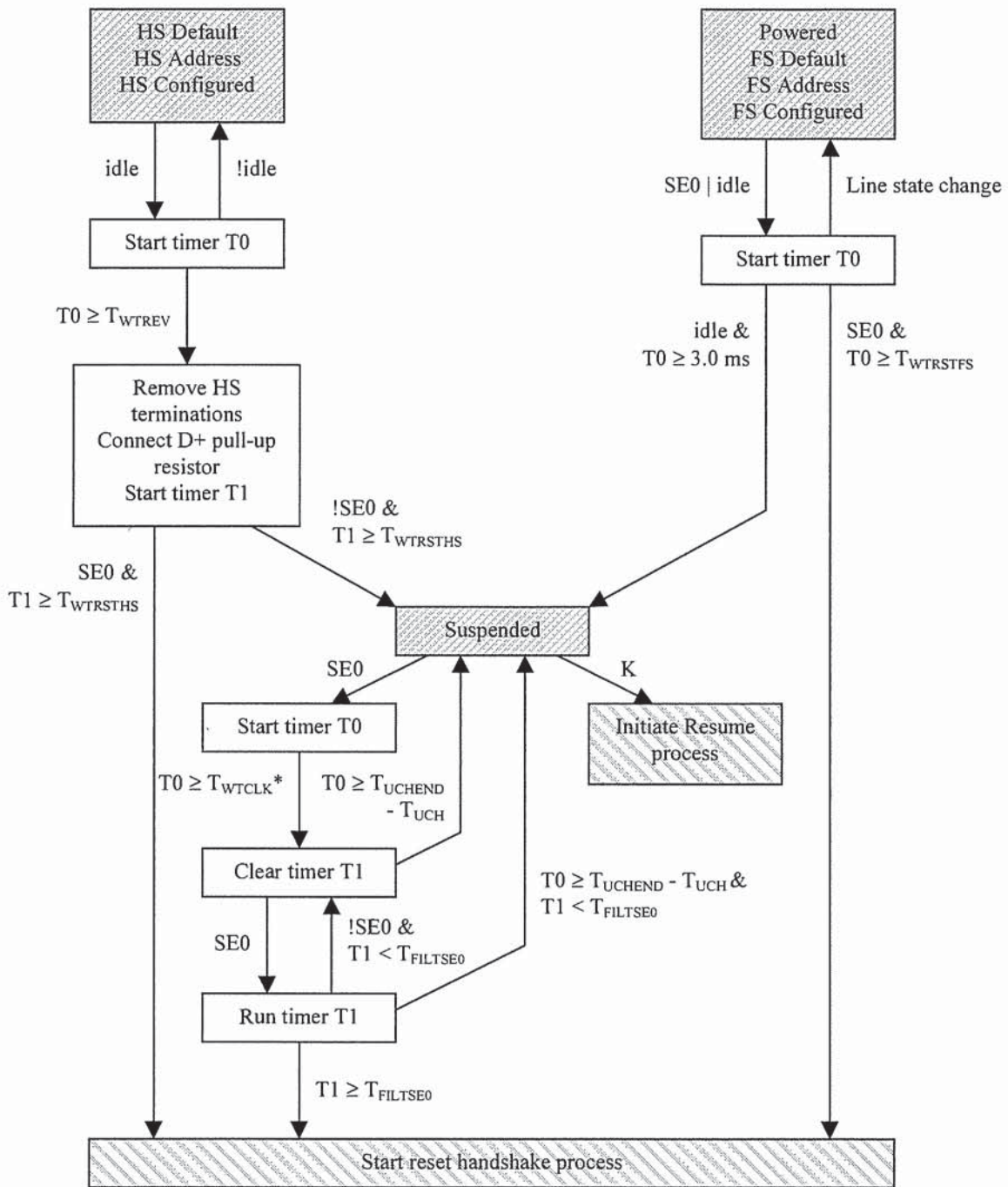
C.2.1 Reset From Suspended State

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 T_0 .

The actual reset handshake is only started after seeing a continuous assertion of $SE0$ for at least 2.5 μ s ($T_{FILTSE0}$). 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 T_0 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 an 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.



(*) **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

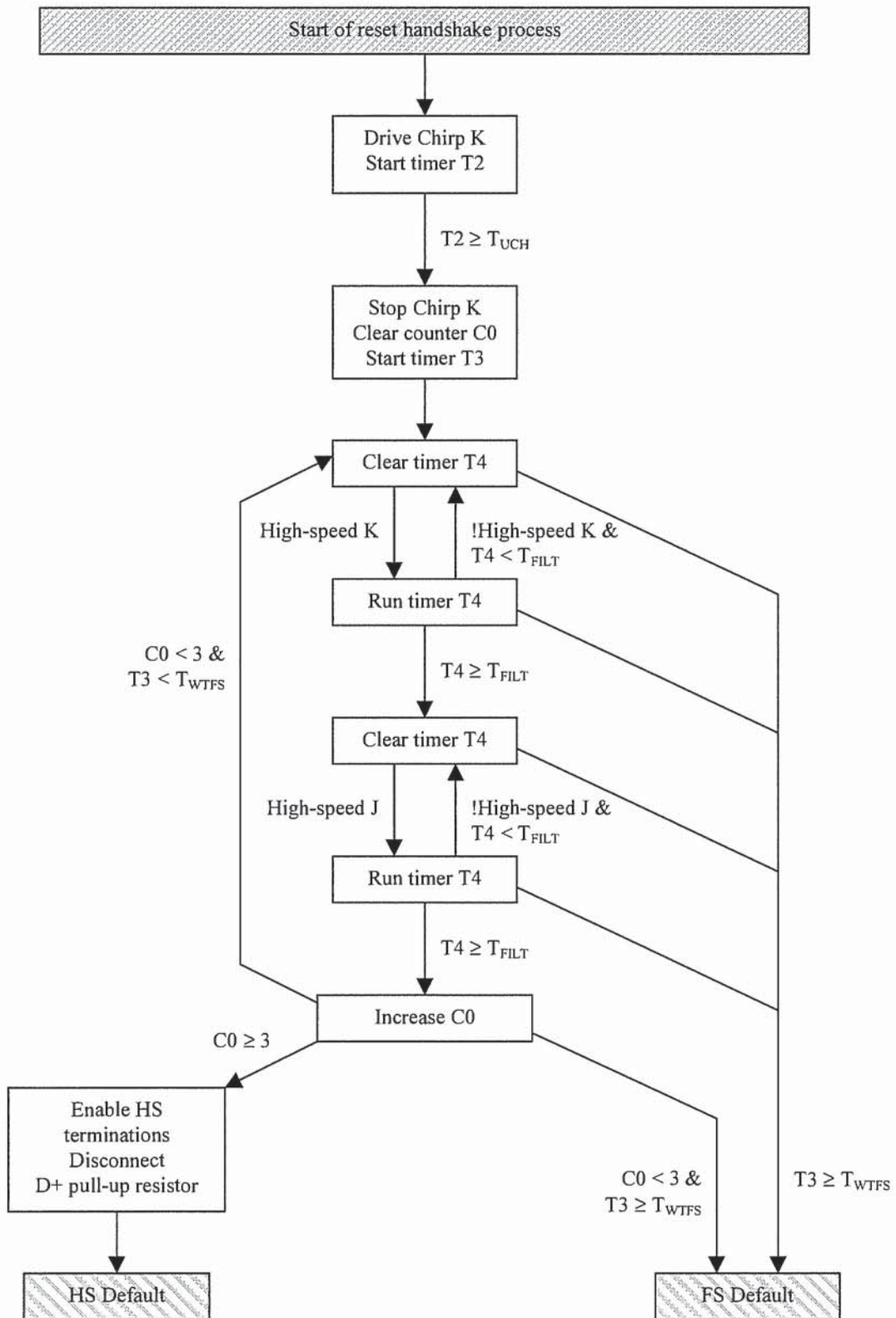


Figure C-3. Upstream Facing Port Reset Handshake State Diagram

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 μ s 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 $T_{WTRSTHS}$ (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 μ s (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_{WTF3} , the device enters the full-speed default state. Reset ends when the bus state changes from SE0 to idle. The time T_{WTF3} is given a wide range to

Universal Serial Bus Specification Revision 2.0

allow sufficient leverage for a device which has awoken from suspend to use its (possibly 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 μ s 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.

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 DLLs, 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

USBID 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

USBID 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

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*

- 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

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
 - USB 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
 - USB 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
 - USB 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

- 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

- 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

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
- USBID mechanisms for getting current settings, 10.5.2.4
- USBID 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

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
 USB 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

- 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

- 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
 - USBID 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_Isochl 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

- 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
 - USB 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
 - USBID 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, USB 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*