

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 NYETand FS/LS handshake packet is done by M+2)	Figure A-56	
No smash(HS NYET and FS/LS handshake packet is done by M+3)	Figure A-57	
HS NYET smash	Figure A-58	
HS NYET 3 strikes smash	Figure A-59	

- Abort and Free cases

Case	Reference Figure	Similar Figure
No smash and abort (HS NYETand FS/LS transaction is continued at end of M+3)	Figure A-60	
No smash and free(HS NYETand FS/LS transaction is not started at end of M+3)	Figure A-61	

- FS/LS transaction error cases

Case	Reference Figure	Similar Figure
HS ERR smash		Figure A-51
HS ERR 3 strikes smash		Figure A-52

- Device busy cases

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	

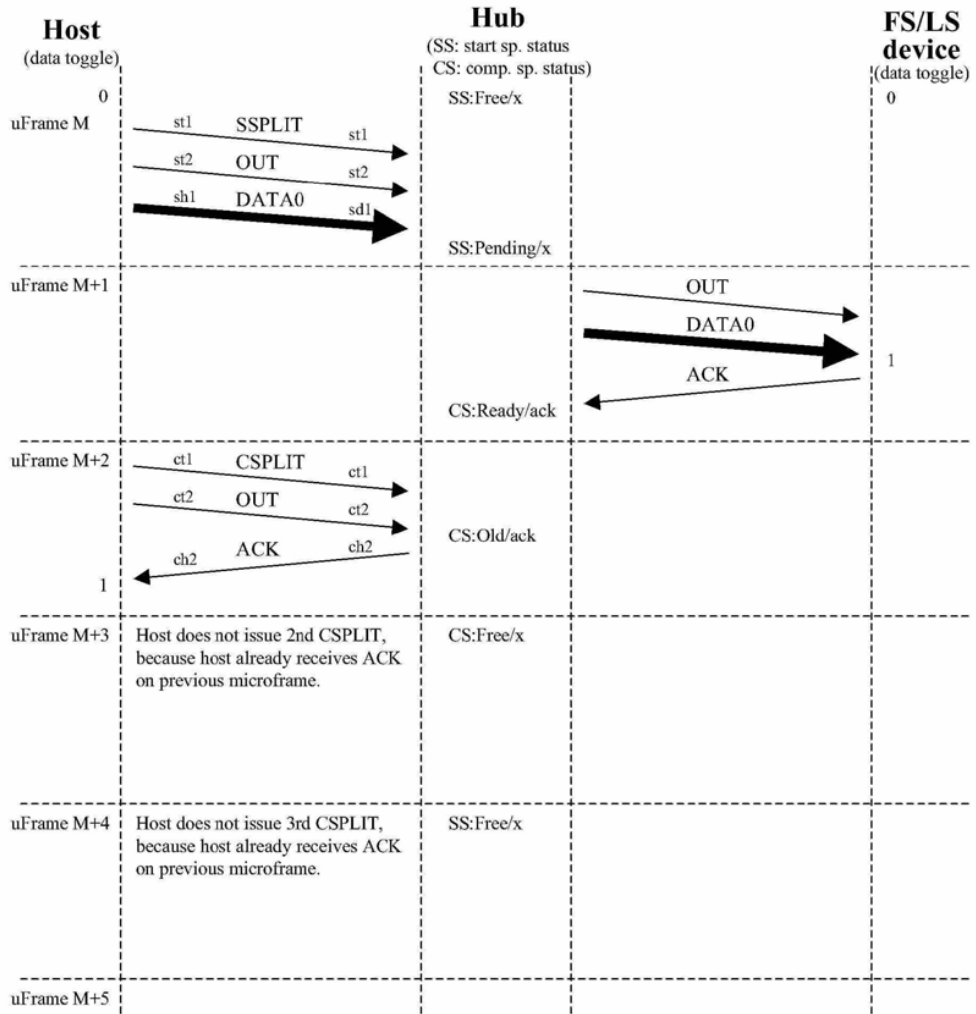


Figure A-47. Normal No Smash(FS/LS Handshake Packet is Done by M+1)

Universal Serial Bus Specification Revision 2.0

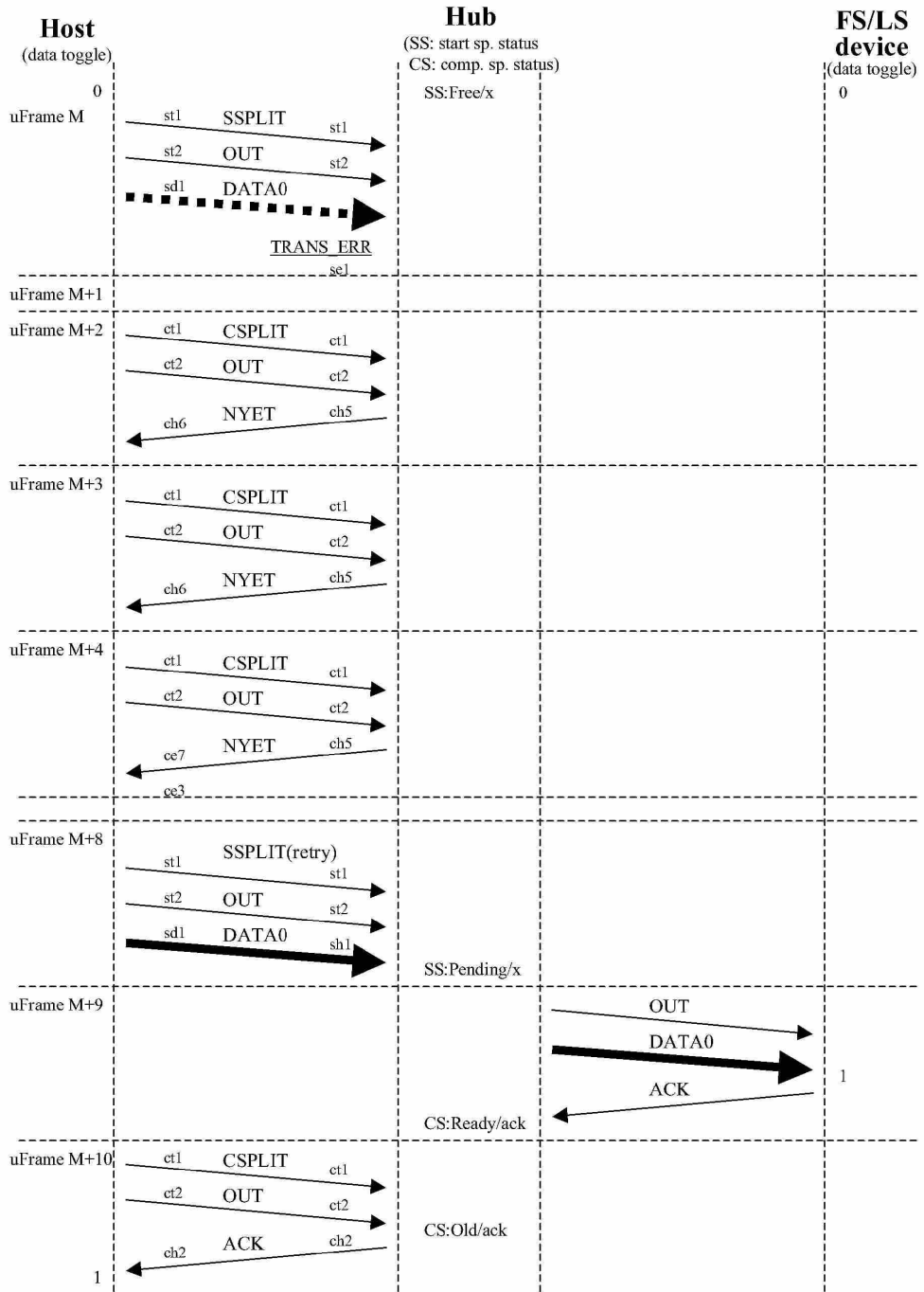


Figure A-48. Normal HS DATA0/1 Smash

Universal Serial Bus Specification Revision 2.0

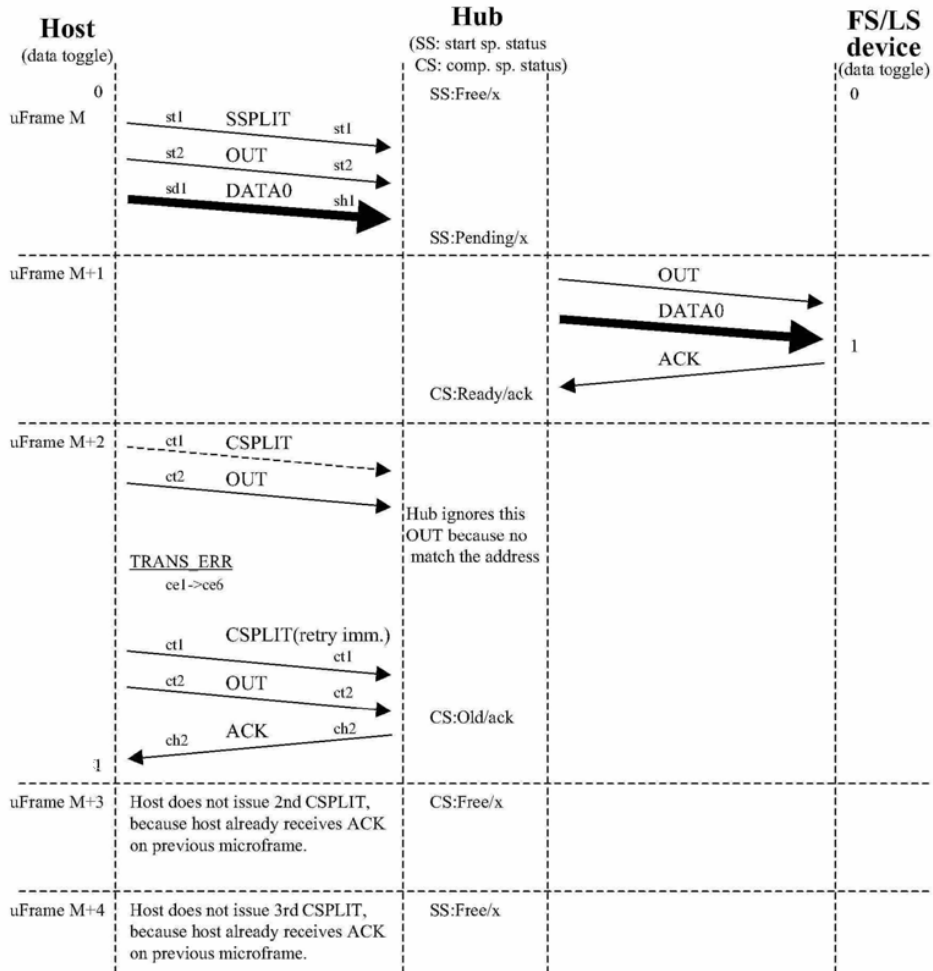


Figure A-49. Normal HS CSPLIT Smash

Universal Serial Bus Specification Revision 2.0

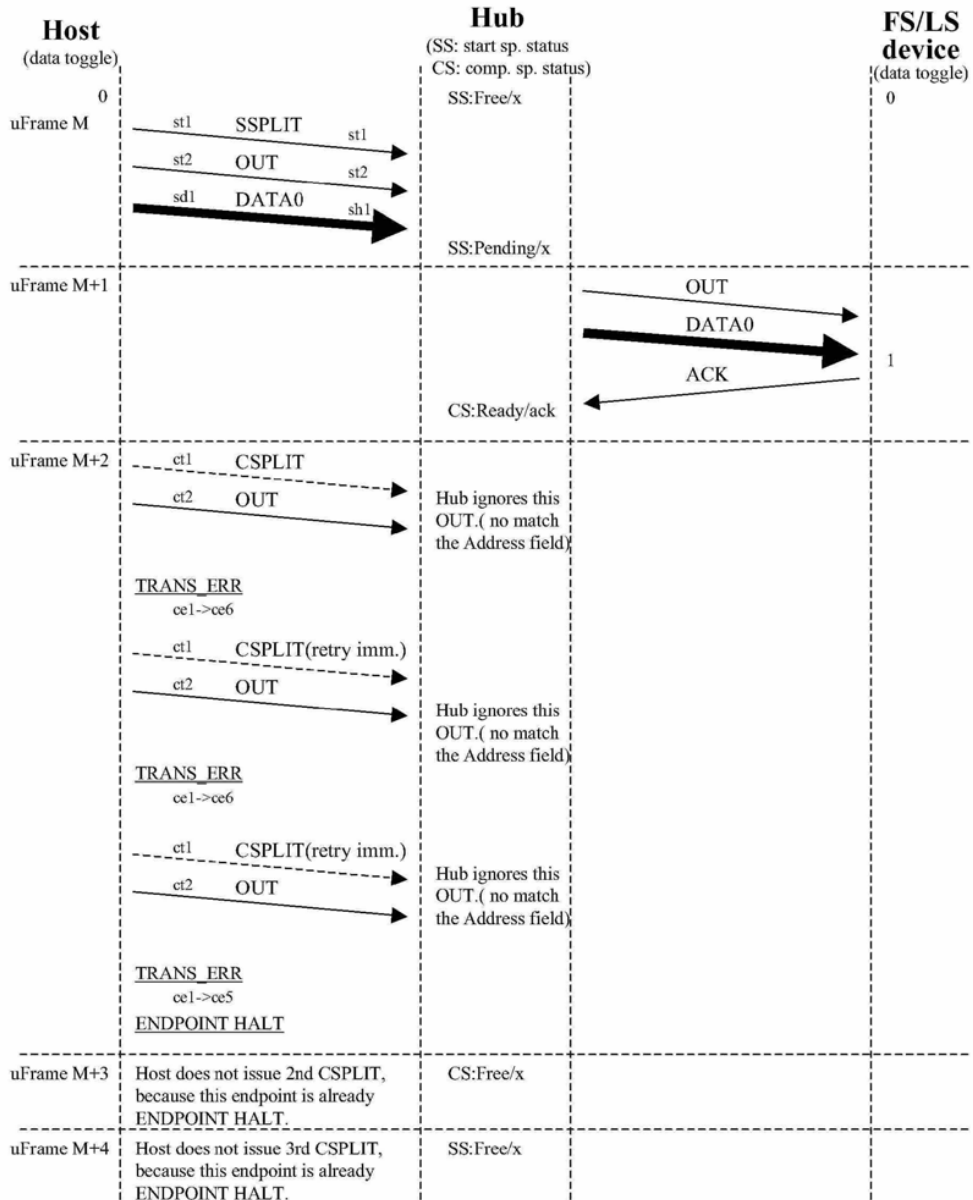


Figure A-50. Normal HS CSPLIT 3 Strikes Smash

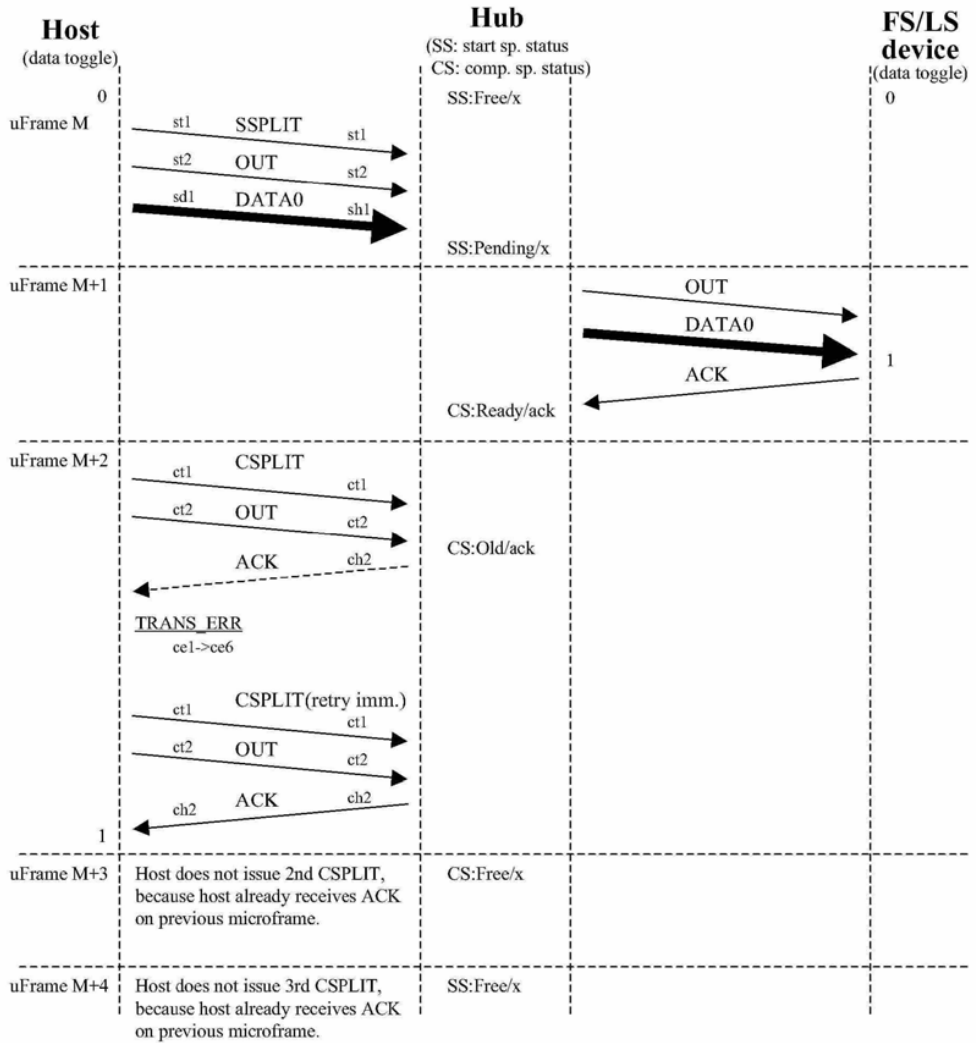


Figure A-51. Normal HS ACK(C) Smash

Universal Serial Bus Specification Revision 2.0

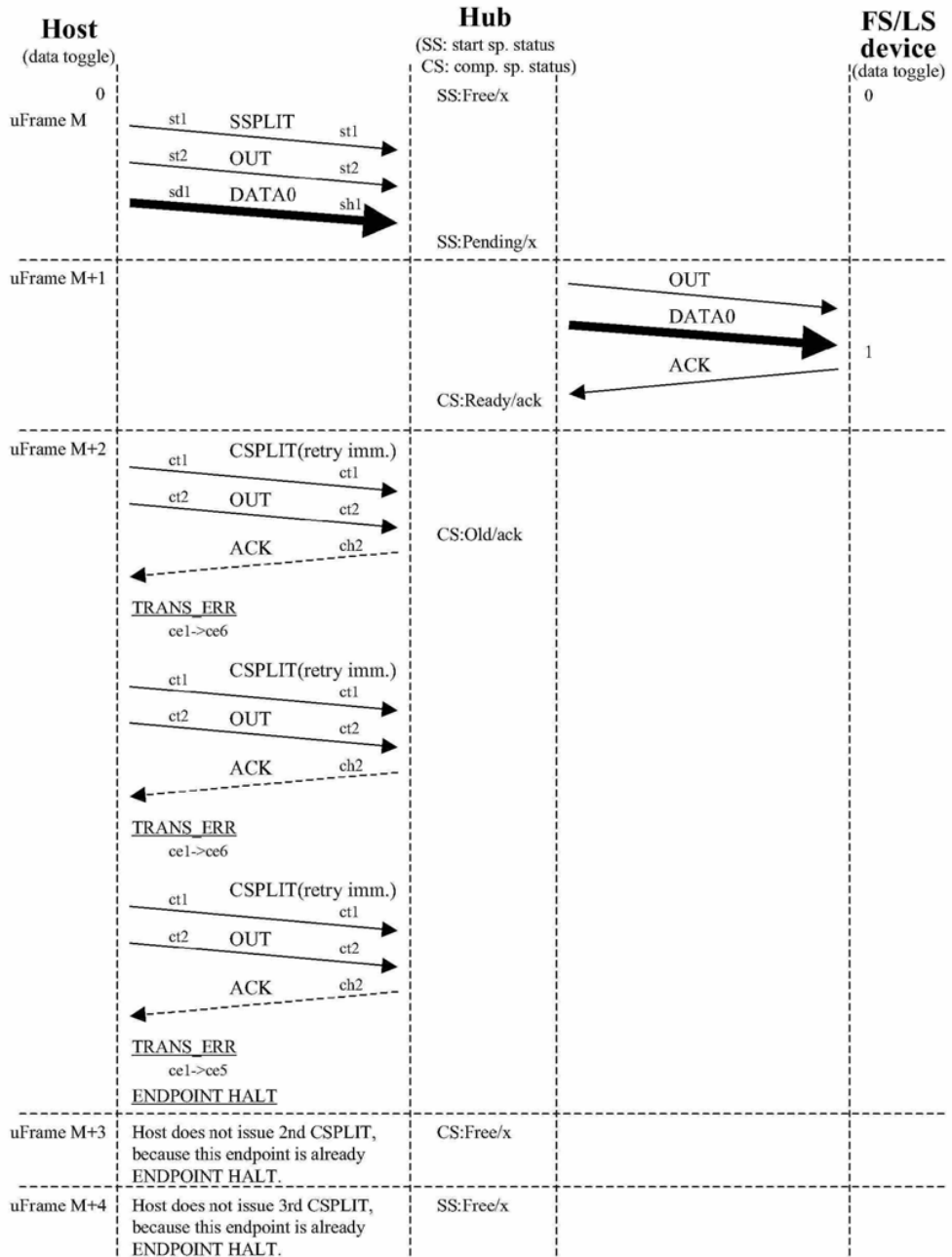


Figure A-52. Normal HS ACK(C) 3 Strikes Smash

Universal Serial Bus Specification Revision 2.0

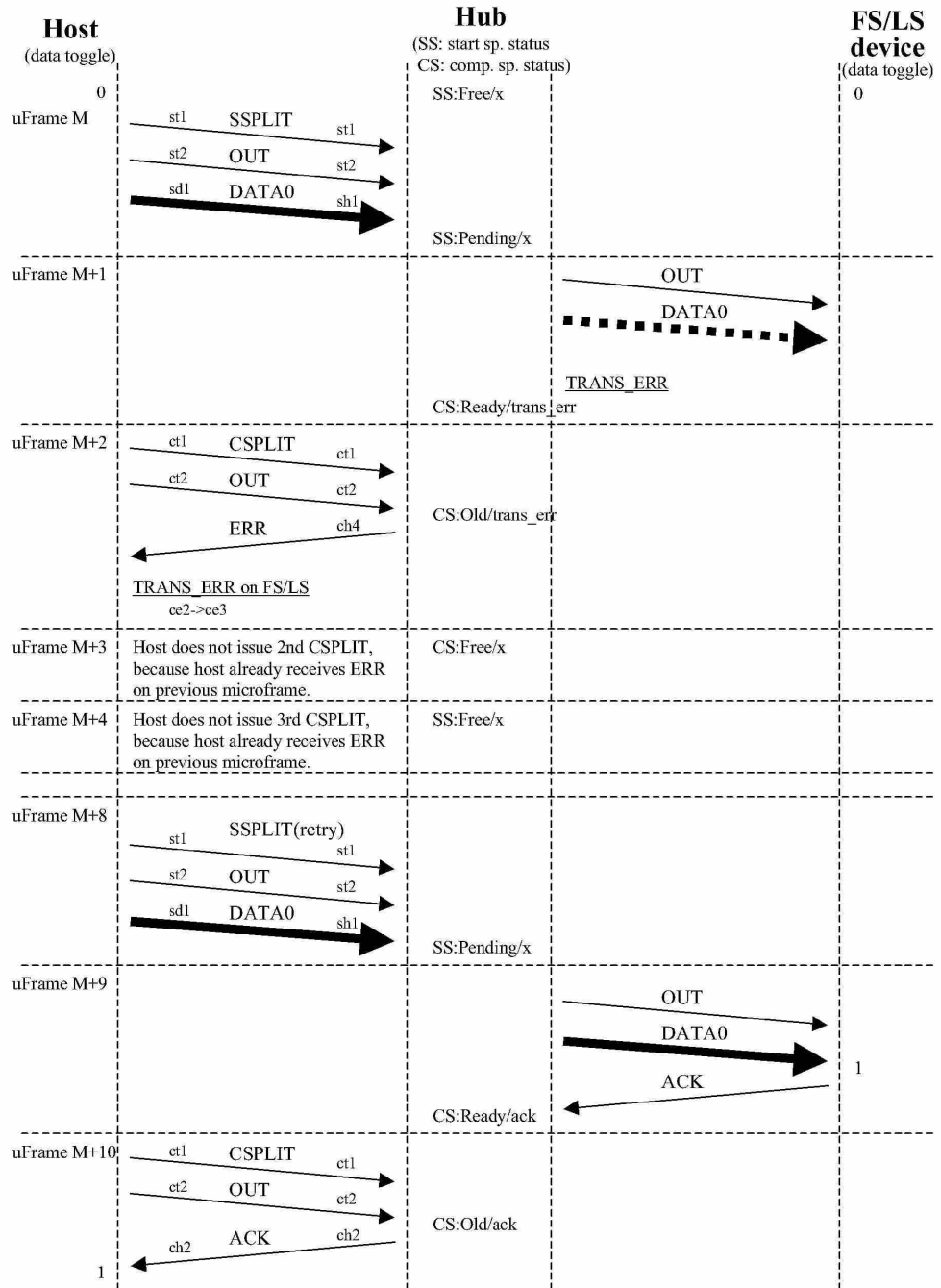


Figure A-53. Normal FS/LS DATA0/1 Smash

Universal Serial Bus Specification Revision 2.0

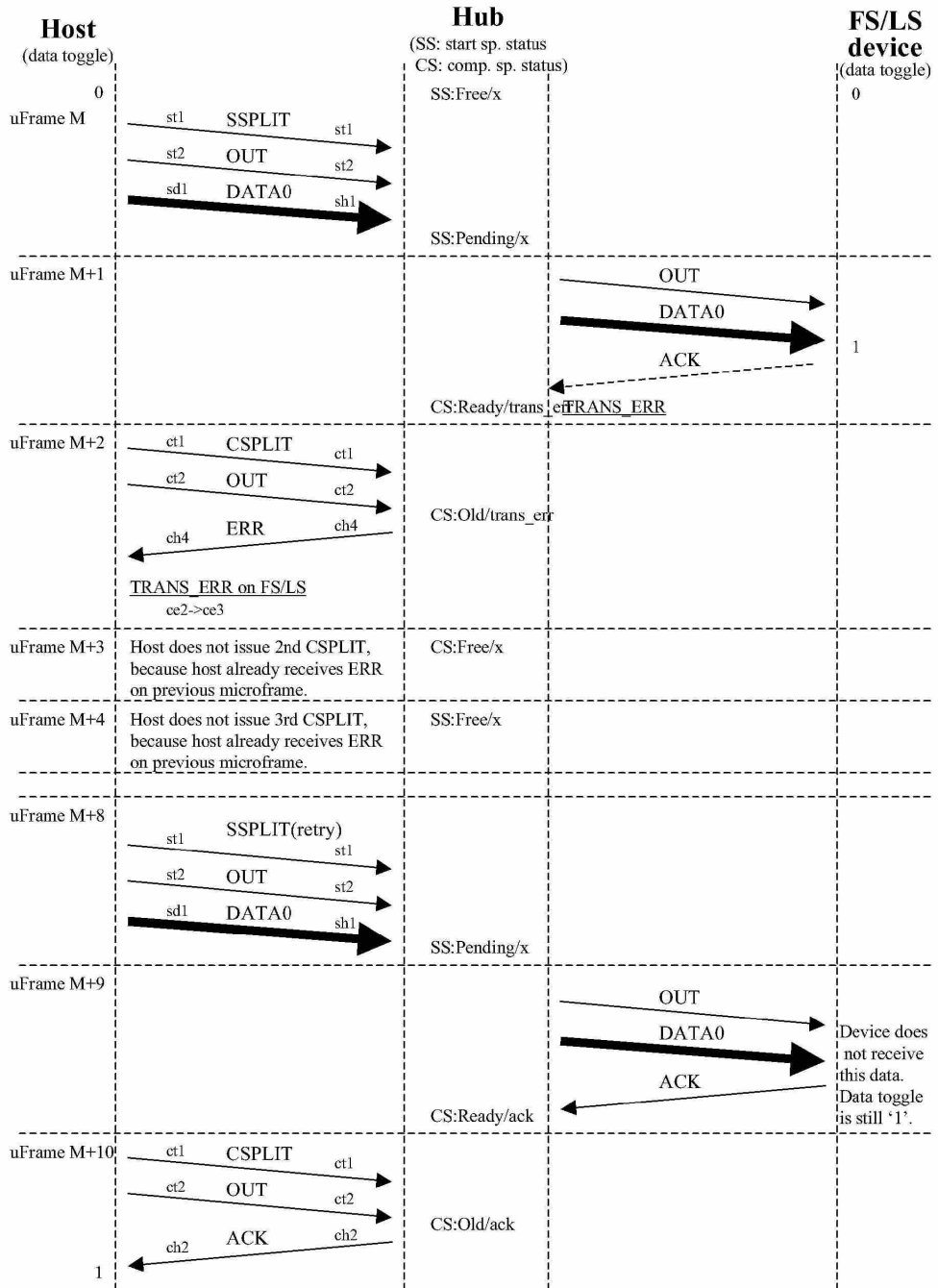


Figure A-54. Normal FS/LS ACK Smash

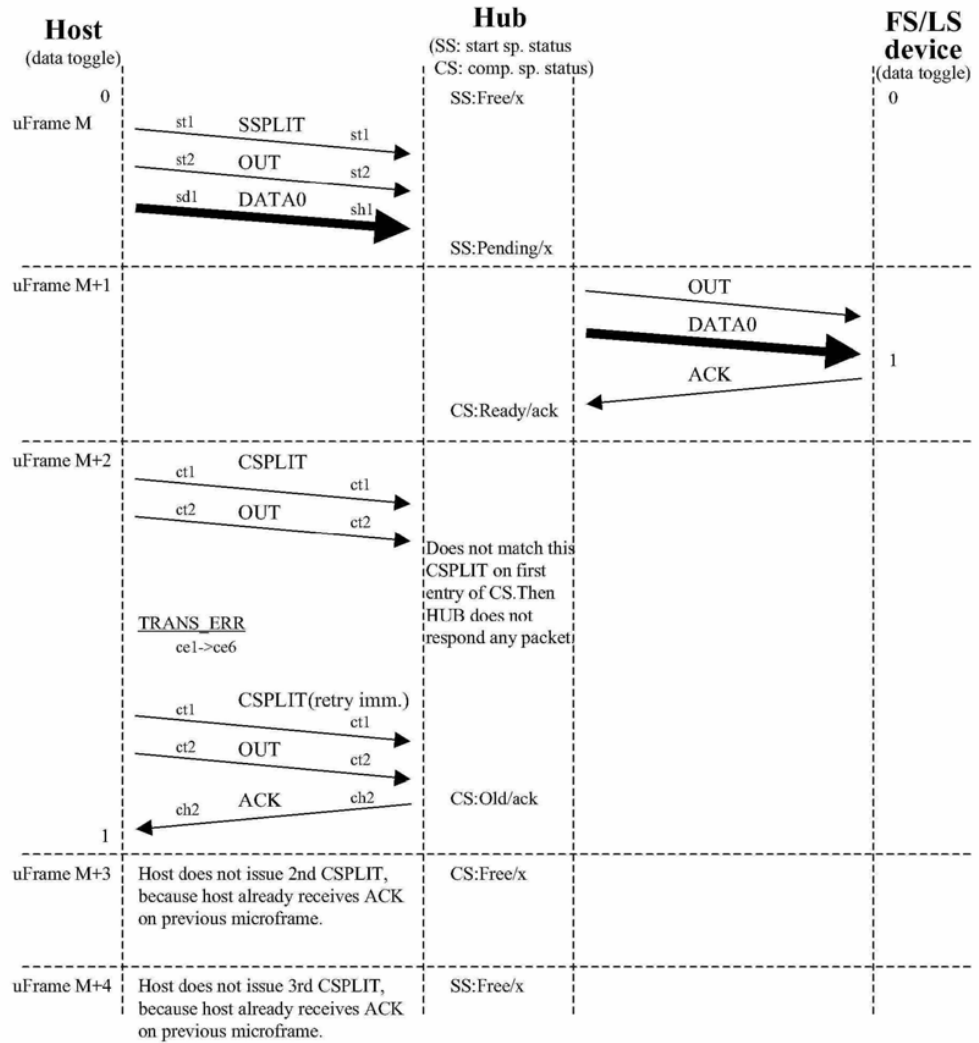


Figure A-55. Searching No Smash

Universal Serial Bus Specification Revision 2.0

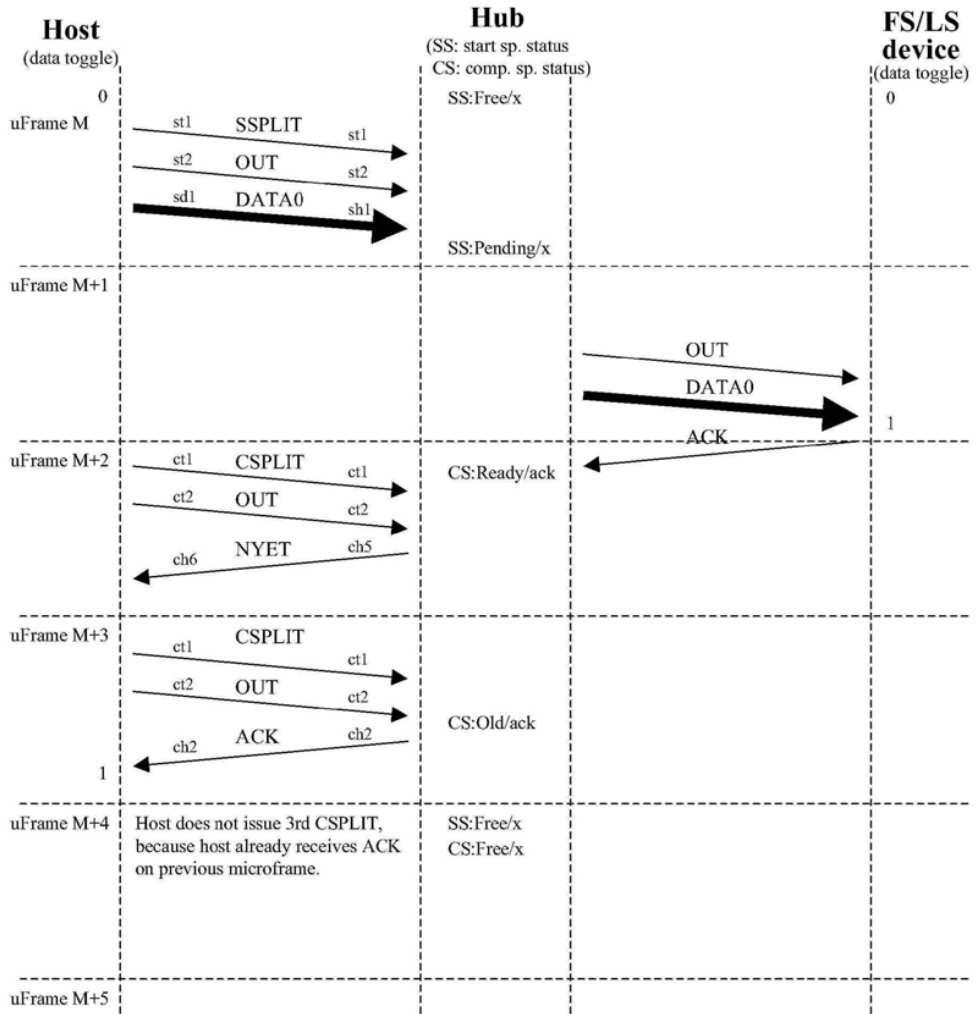


Figure A-56. CS Earlier No Smash(HS NYET and FS/LS Handshake Packet is Done by M+2)

Universal Serial Bus Specification Revision 2.0

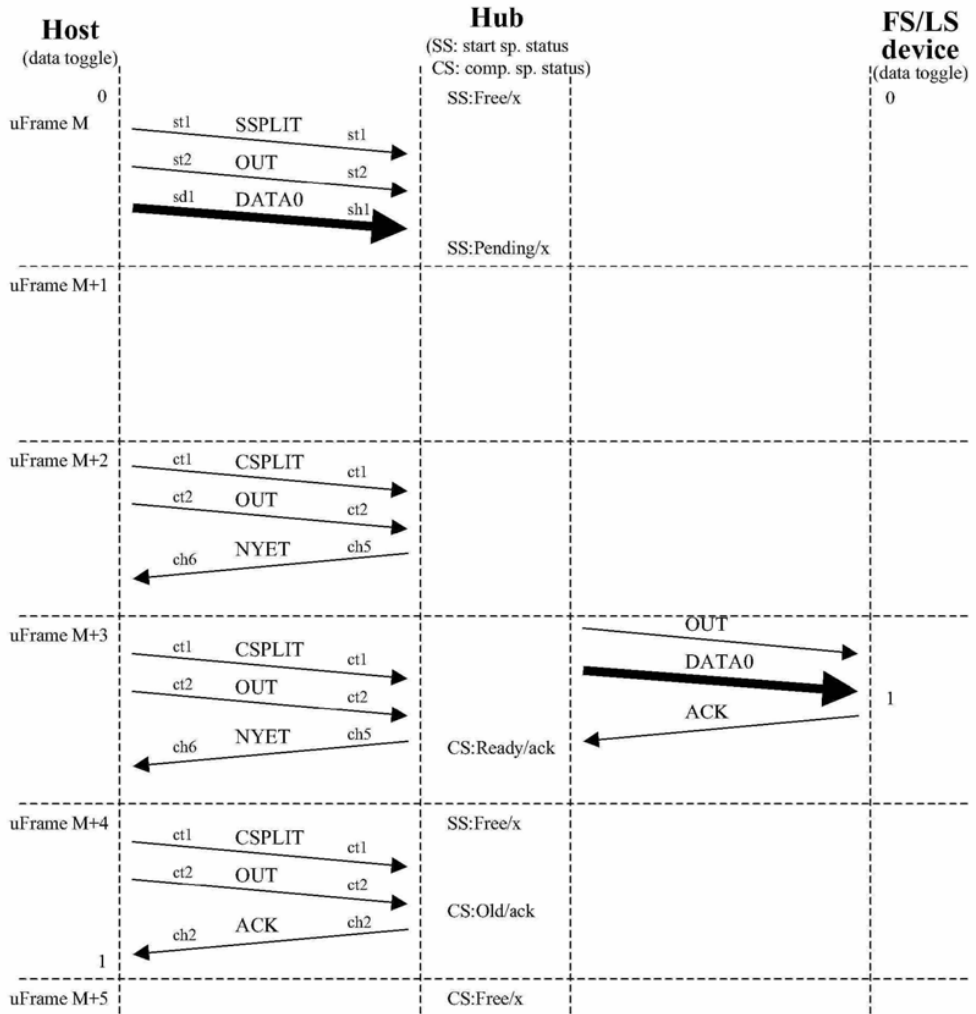


Figure A-57. CS Earlier No Smash(HS NYET and FS/LS Handshake Packet is Done by M+3)

Universal Serial Bus Specification Revision 2.0

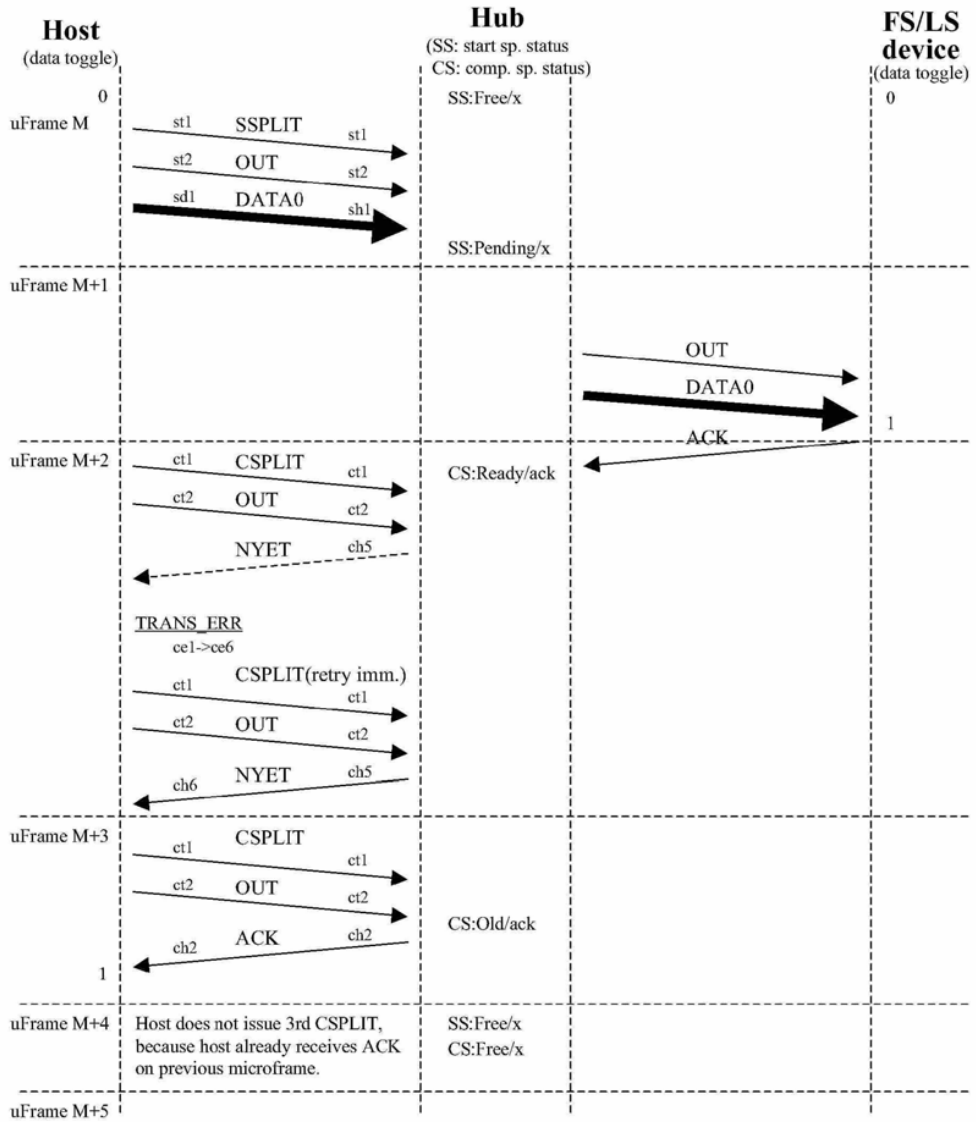


Figure A-58. CS Earlier HS NYET Smash

Universal Serial Bus Specification Revision 2.0

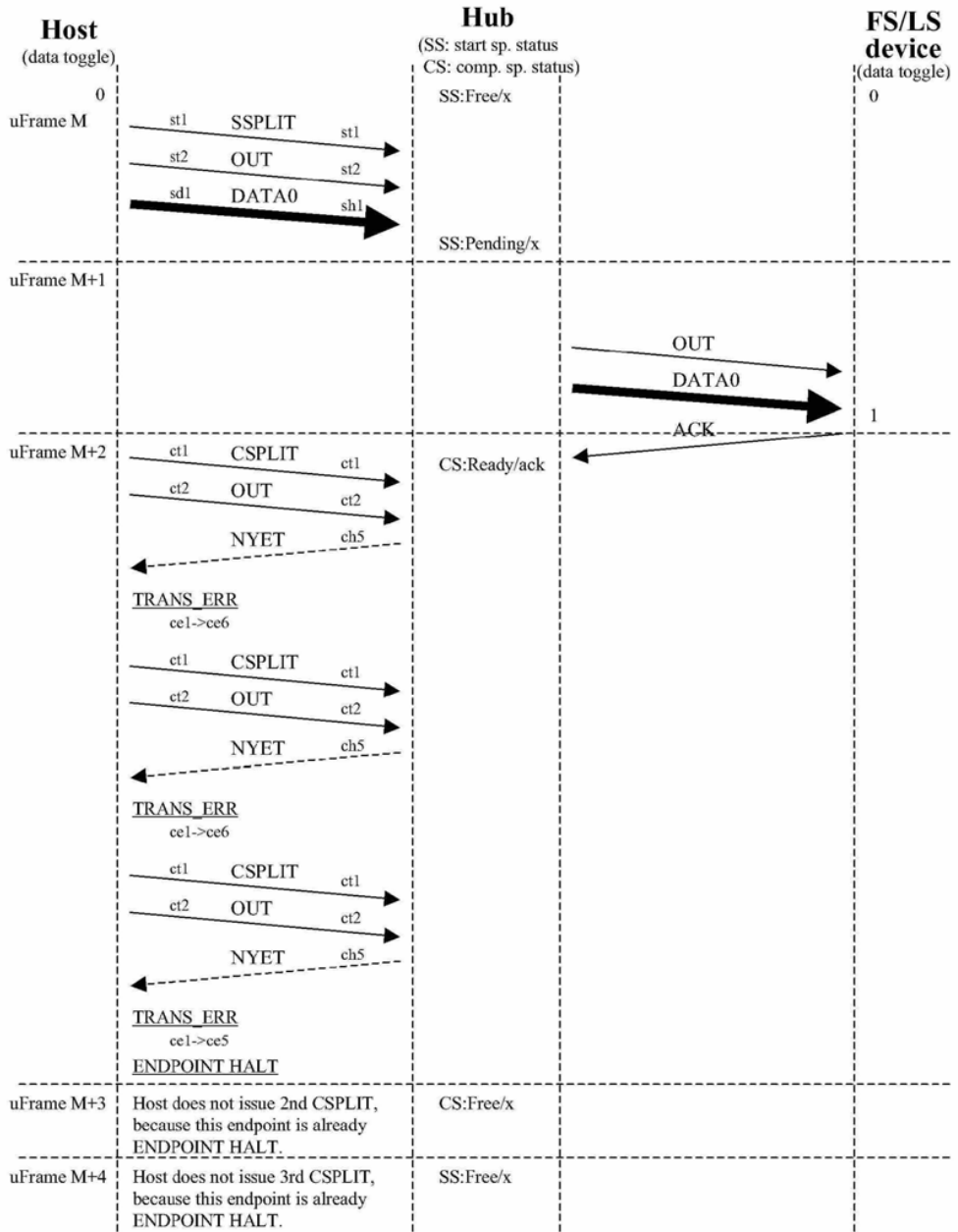


Figure A-59. CS Earlier HS NYET 3 Strikes Smash

Universal Serial Bus Specification Revision 2.0

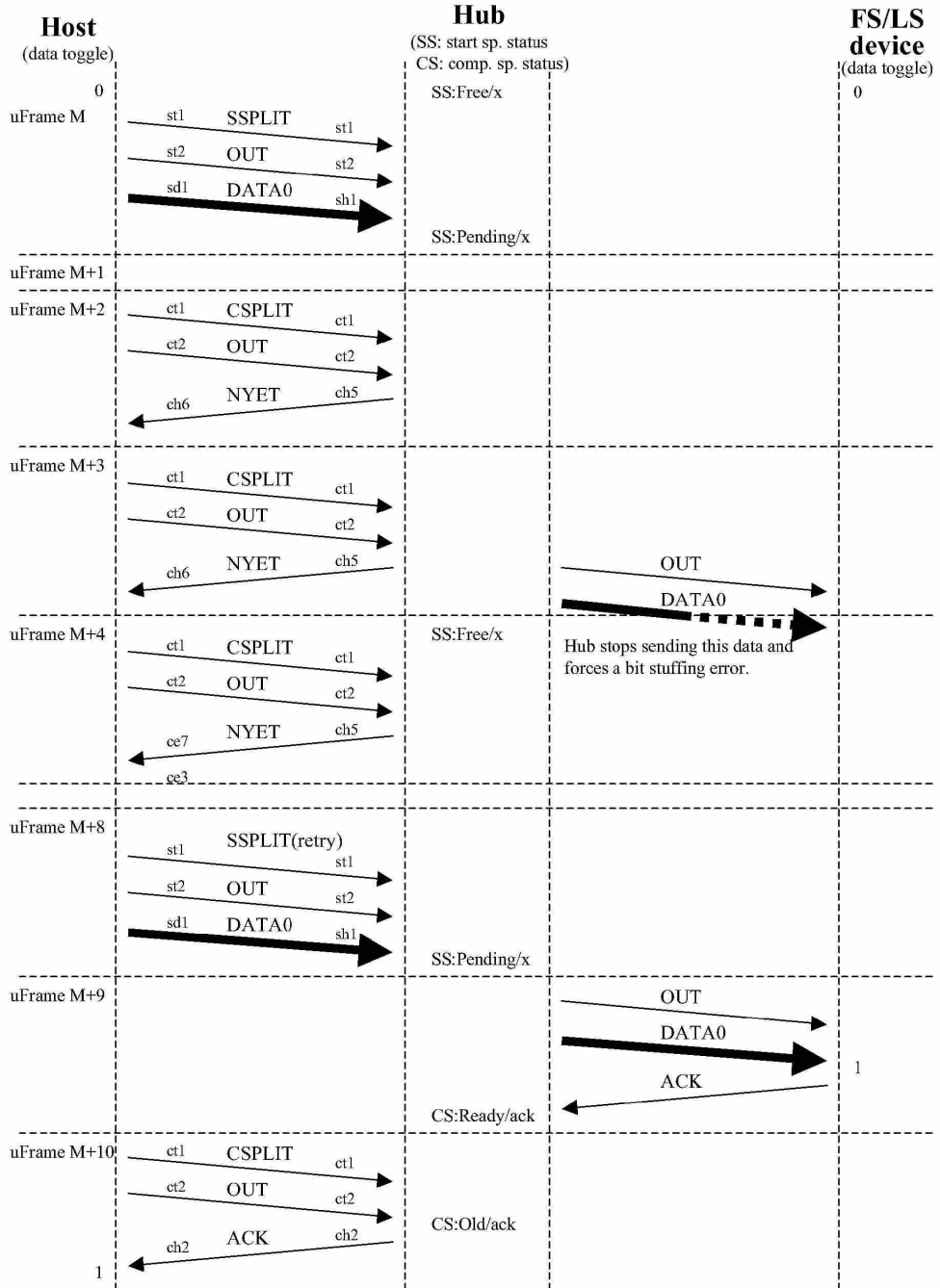


Figure A-60. Abort and Free Abort(FS/LS Transaction is Continued at End of M+3)

Universal Serial Bus Specification Revision 2.0

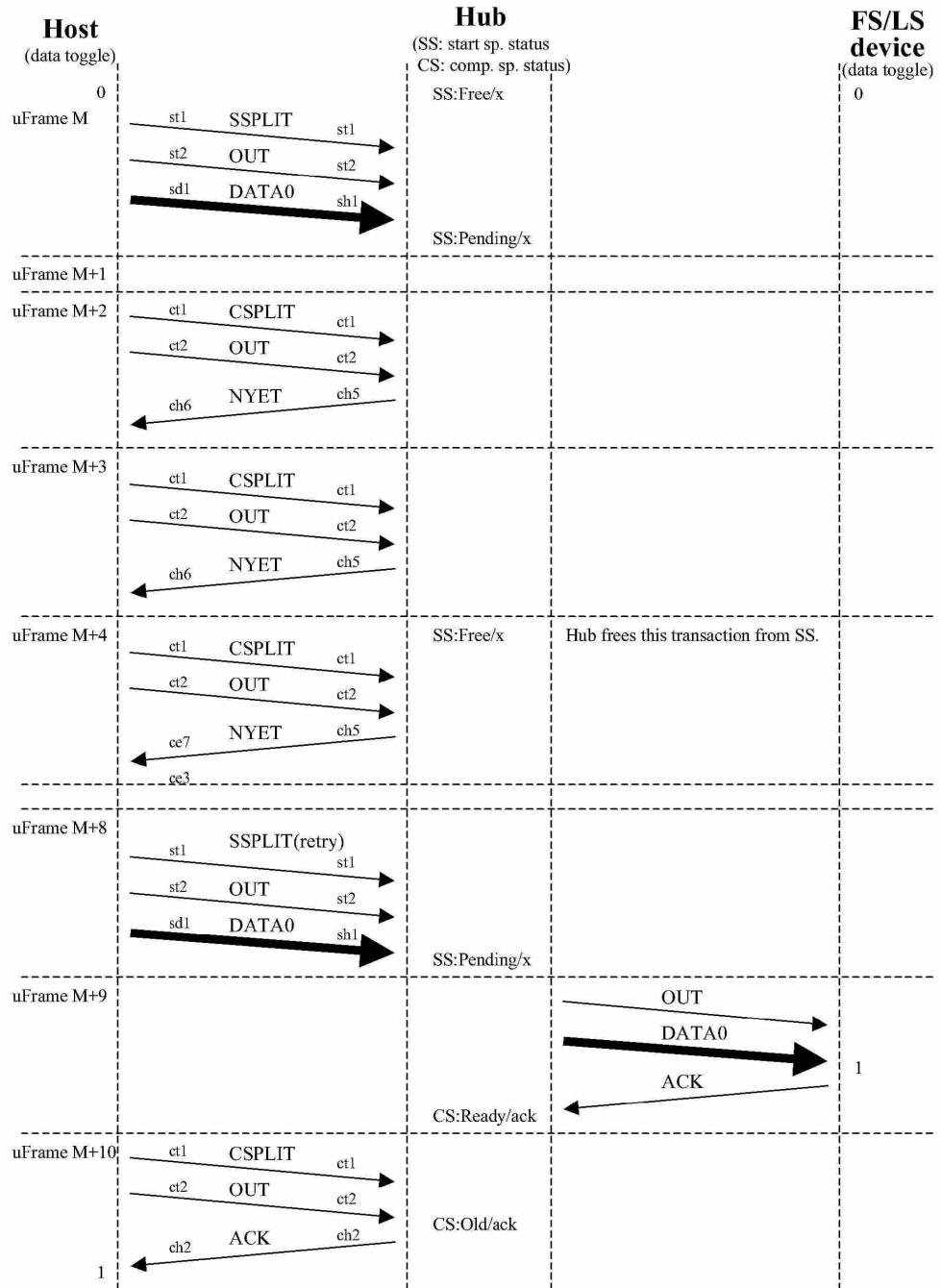


Figure A-61. Abort and Free Free(FS/LS Transaction is not Started at End of M+3)

Universal Serial Bus Specification Revision 2.0

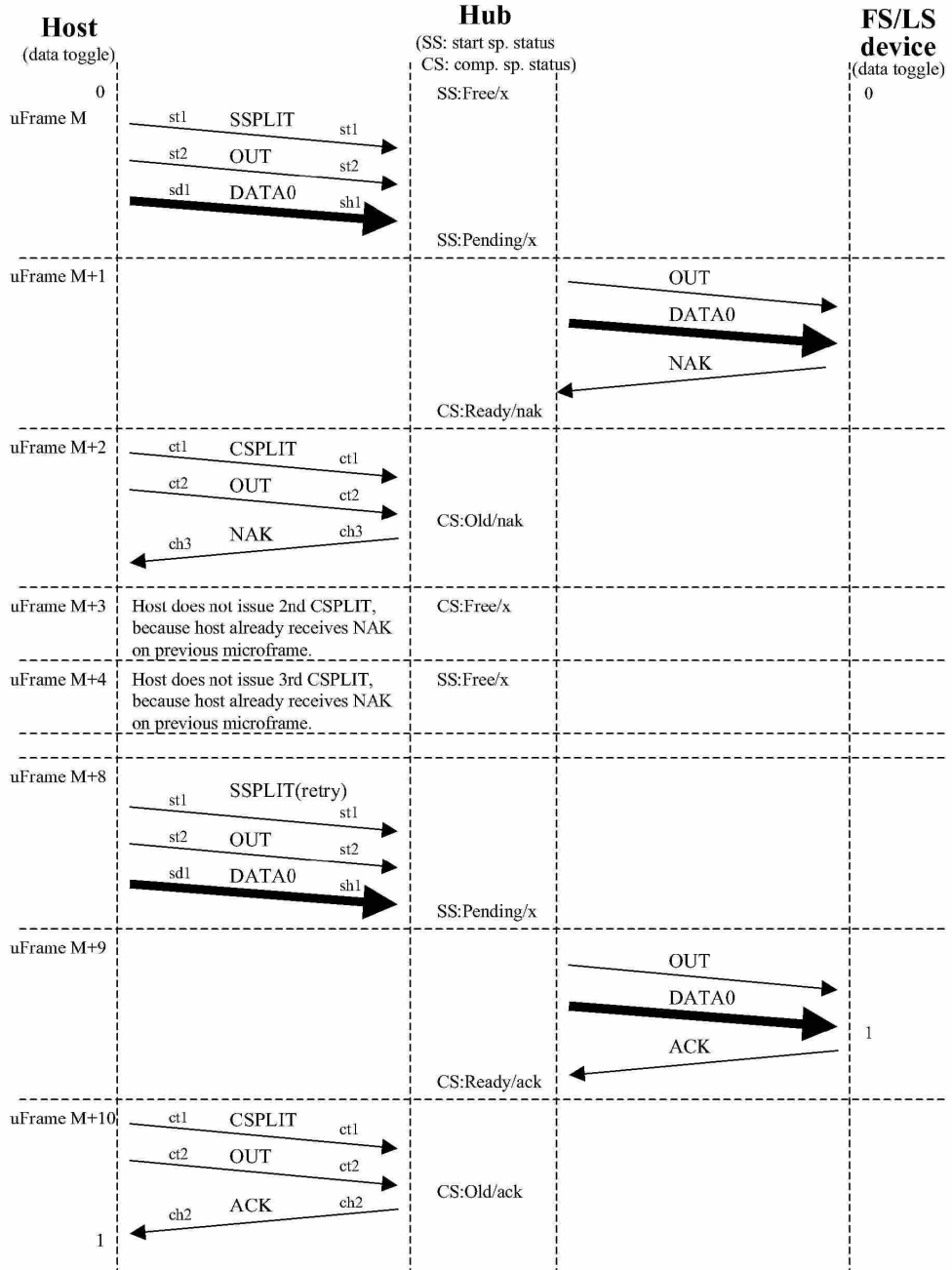


Figure A-62. Device Busy No Smash(FS/LS NAK)

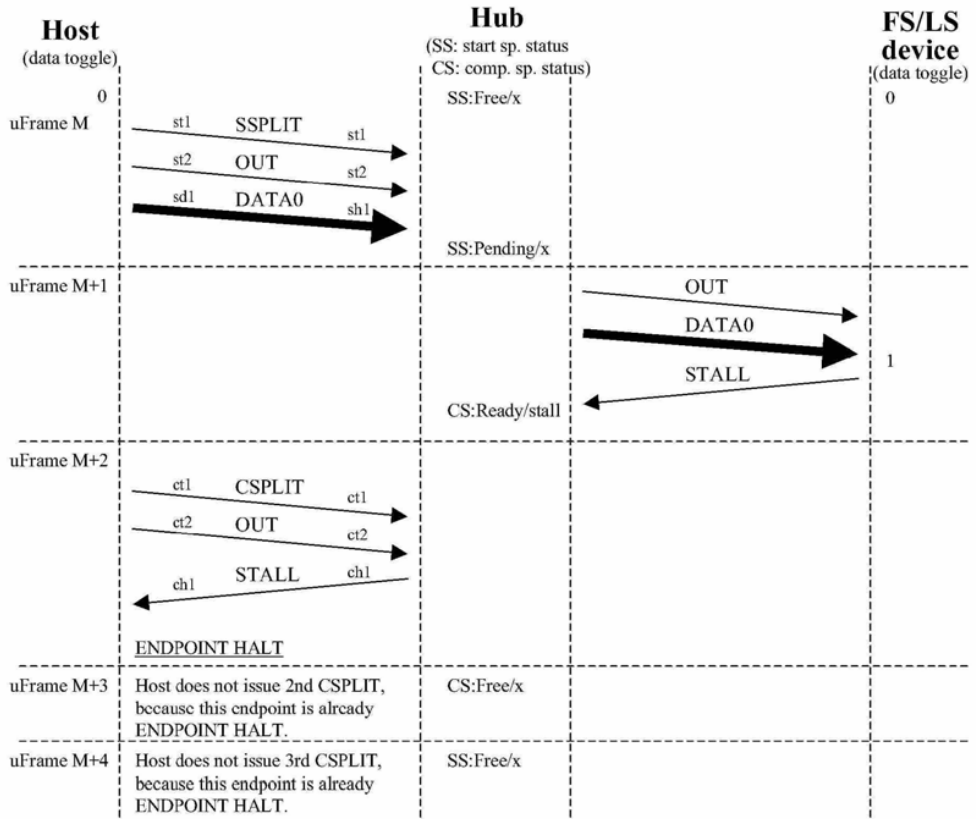


Figure A-63. Device Stall No Smash(FS/LS STALL)

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	

Universal Serial Bus Specification Revision 2.0

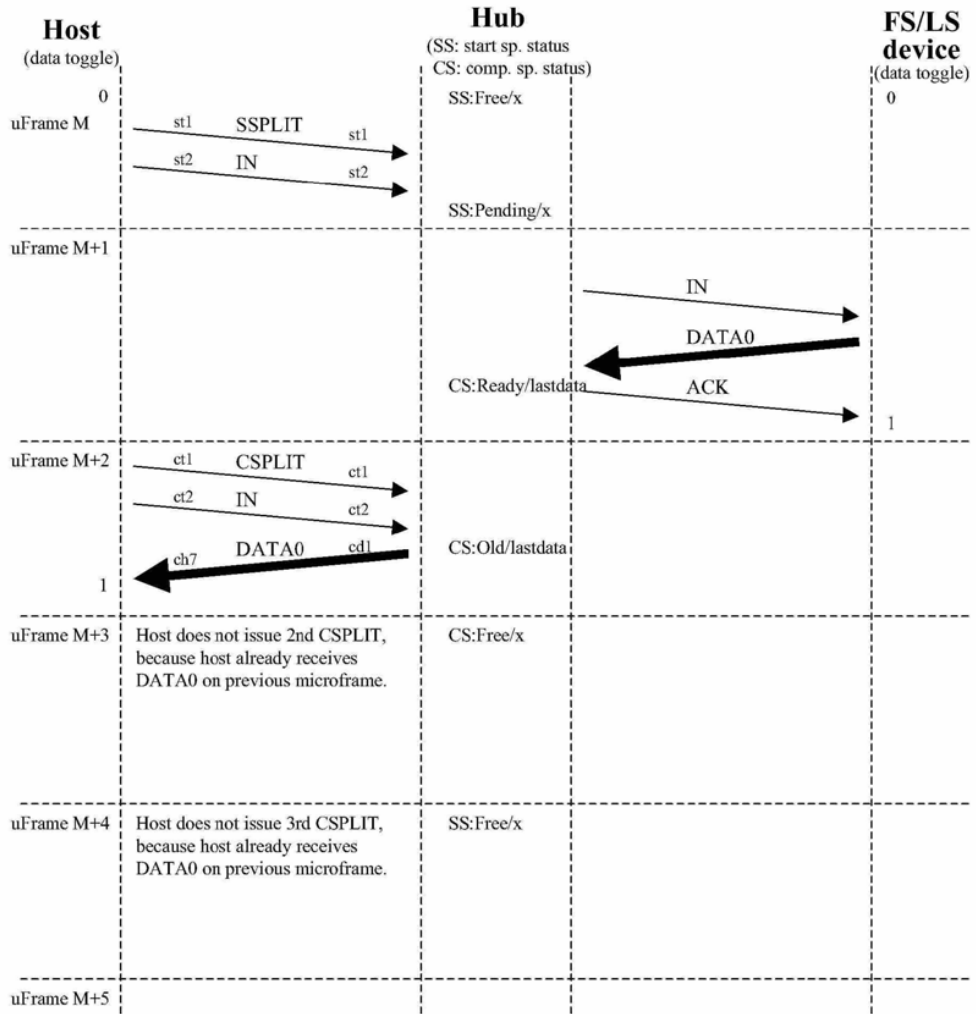


Figure A-64. Normal No Smash(FS/LS Data Packet is on M+1)

Universal Serial Bus Specification Revision 2.0

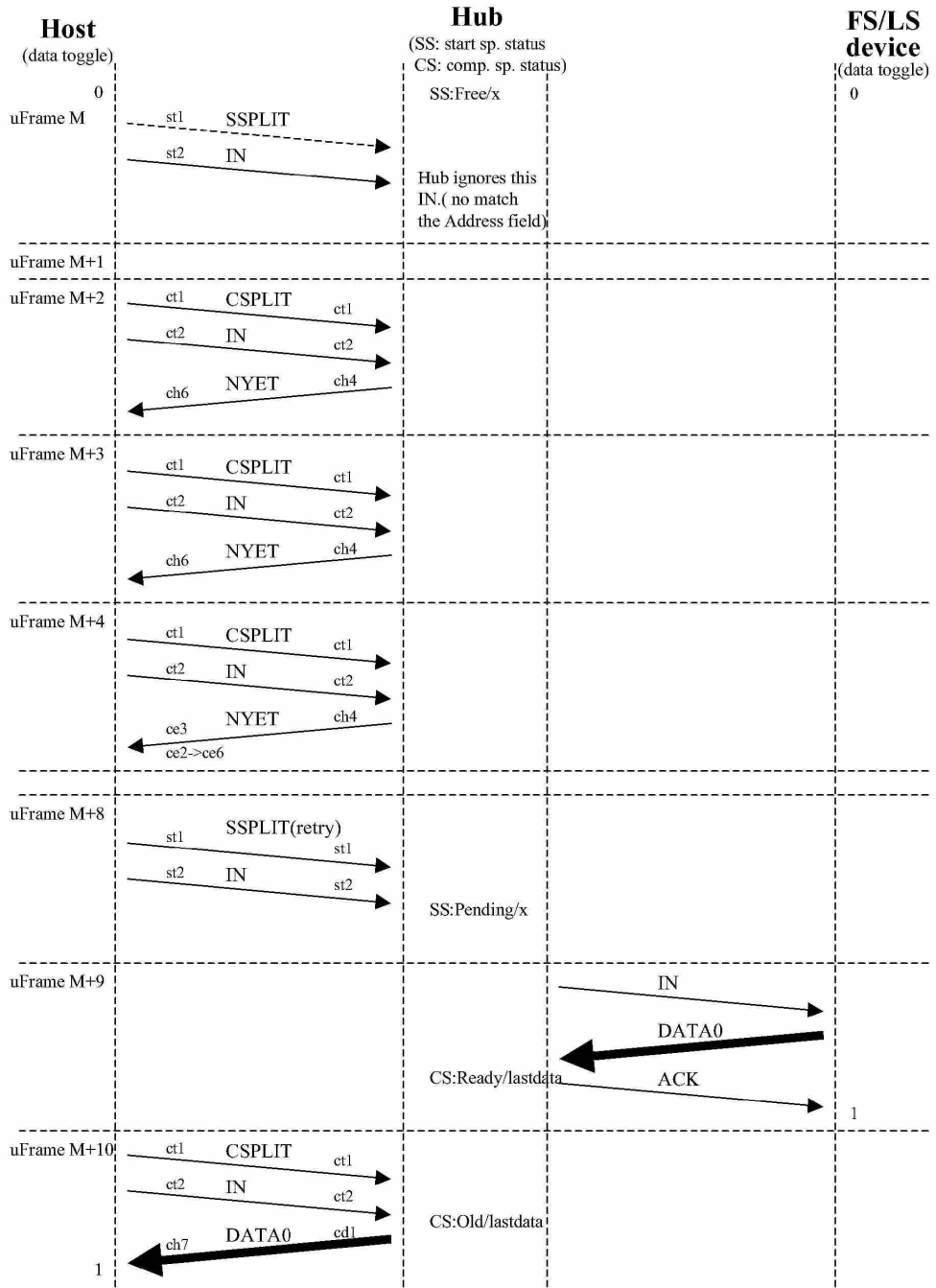


Figure A-65. Normal HS SSPLIT Smash

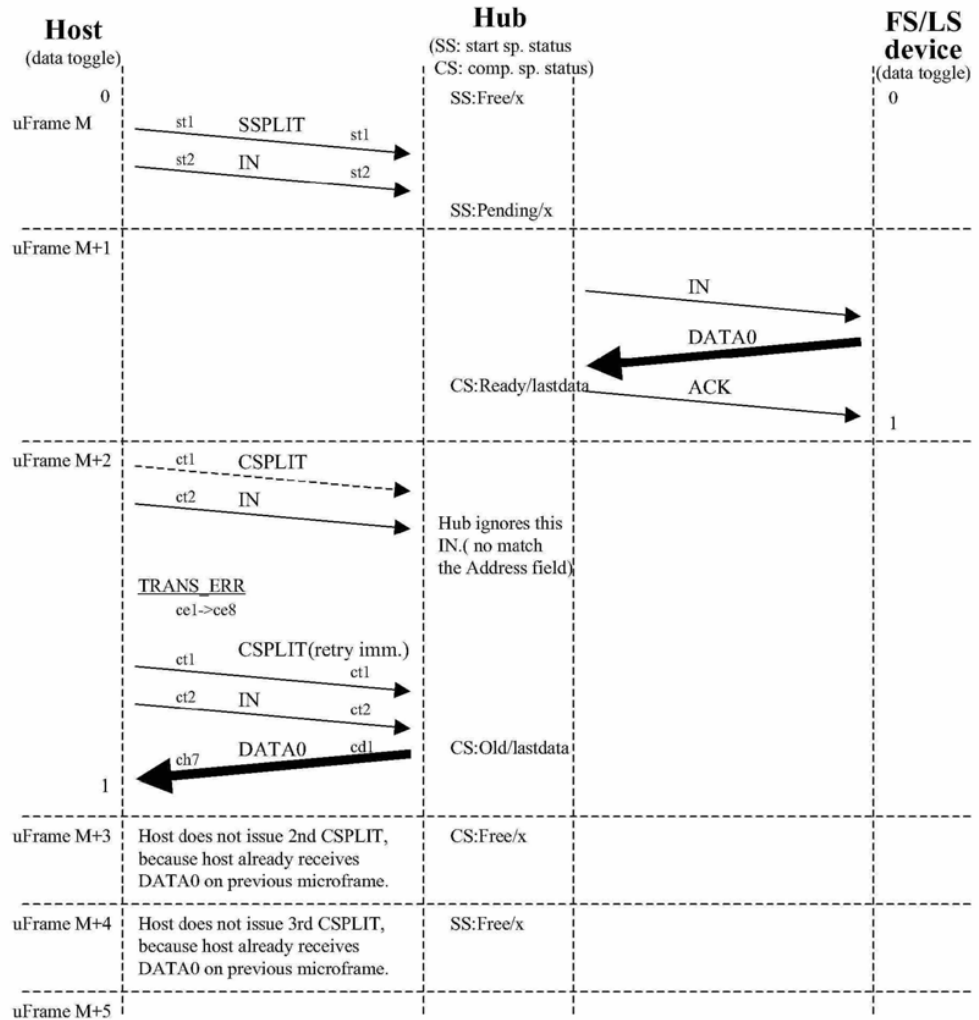


Figure A-66. Normal HS CSPLIT Smash

Universal Serial Bus Specification Revision 2.0

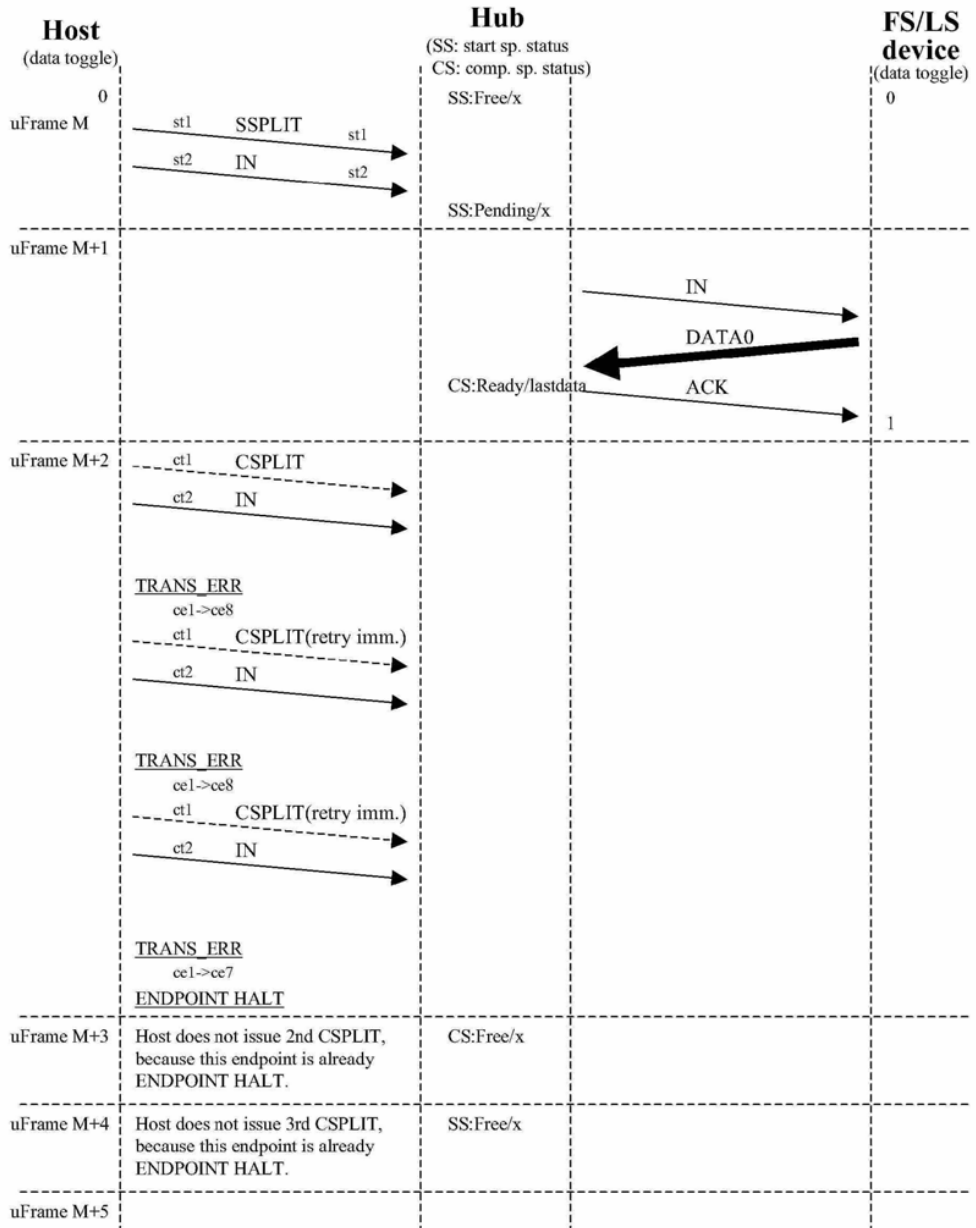


Figure A-67. Normal HS CSPLIT 3 Strikes Smash

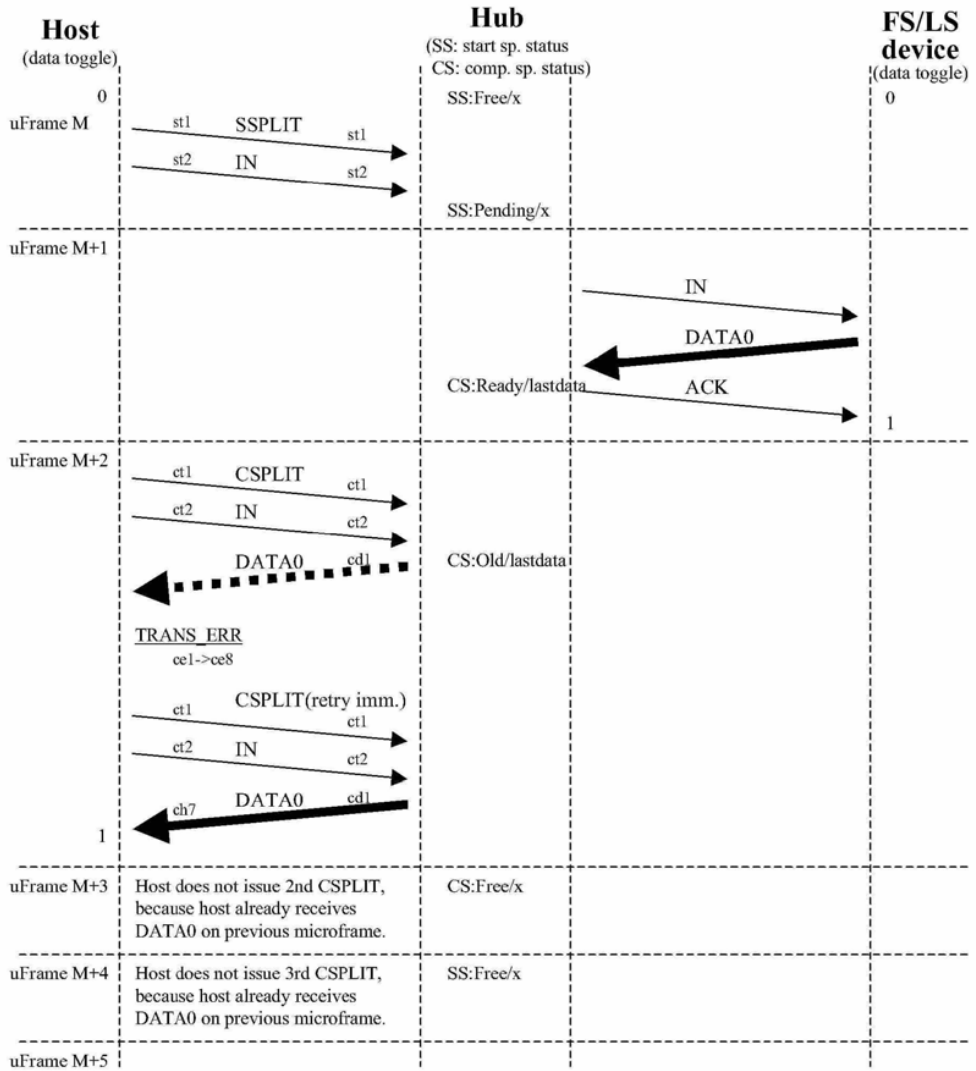


Figure A-68. Normal HS DATA0/1 Smash

Universal Serial Bus Specification Revision 2.0

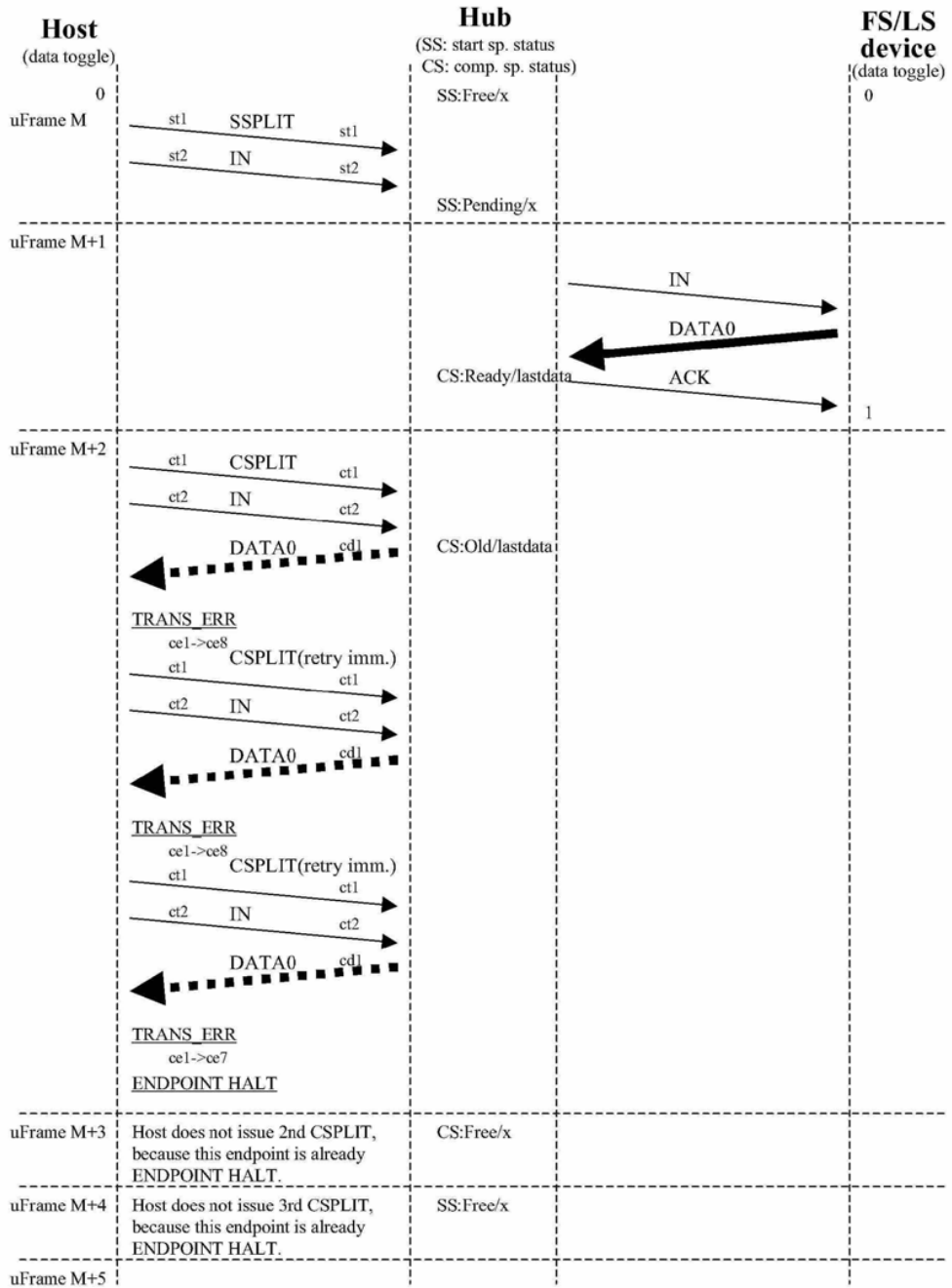


Figure A-69. Normal HS DATA0/1 3 Strikes Smash

Universal Serial Bus Specification Revision 2.0

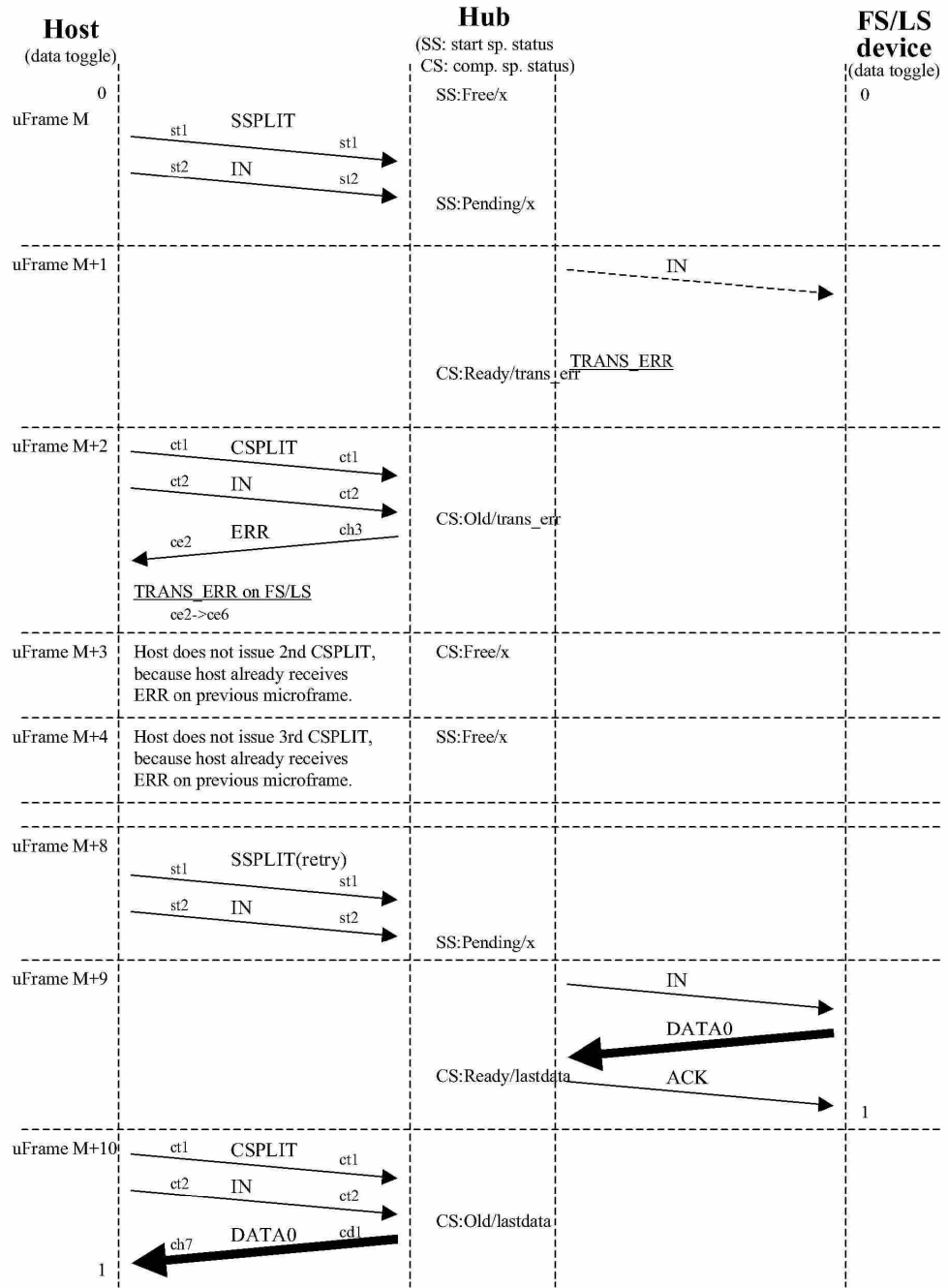


Figure A-70. Normal FS/LS IN Smash

Universal Serial Bus Specification Revision 2.0

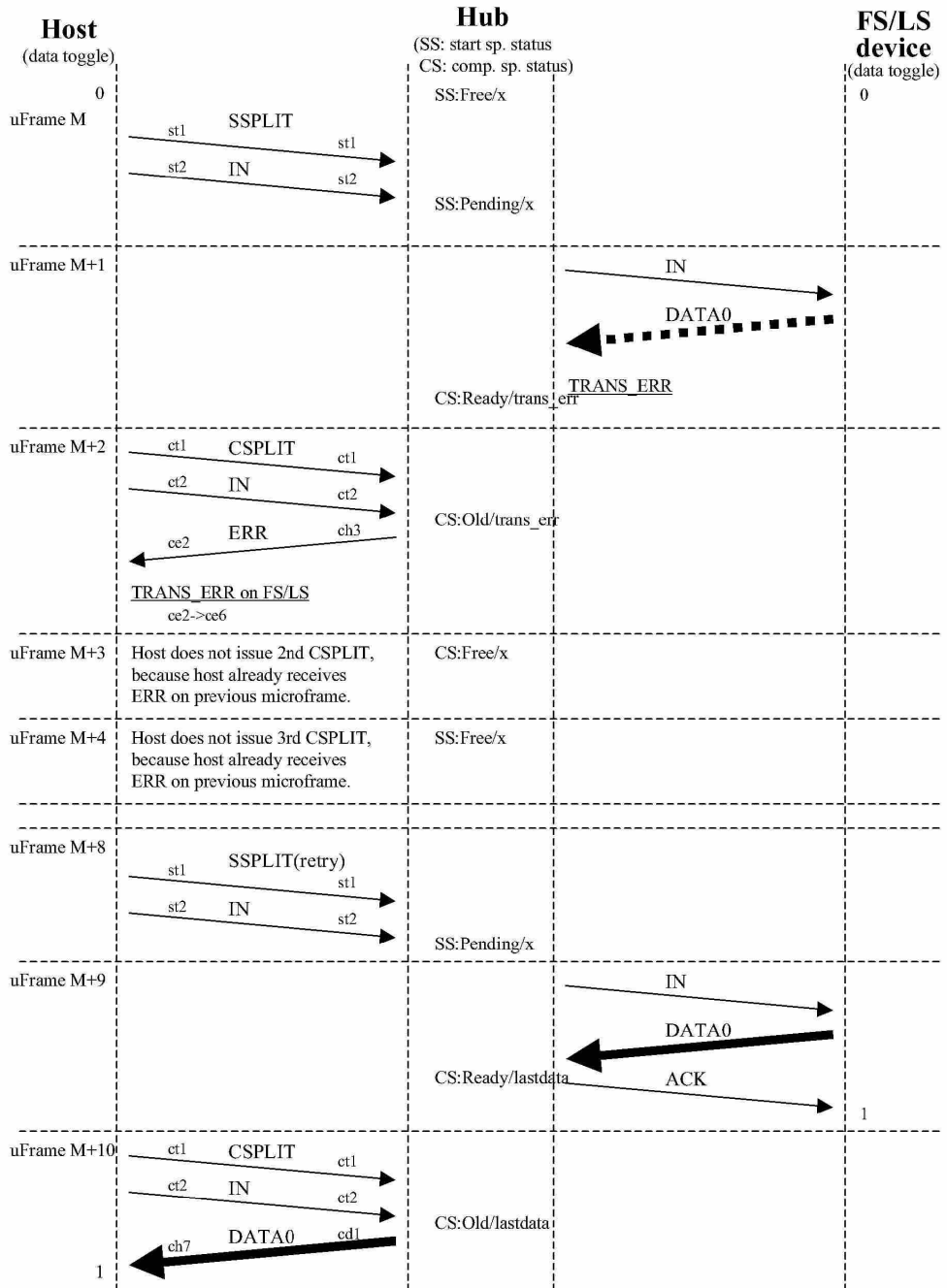


Figure A-71. Normal FS/LS DATA0/1 Smash

Universal Serial Bus Specification Revision 2.0

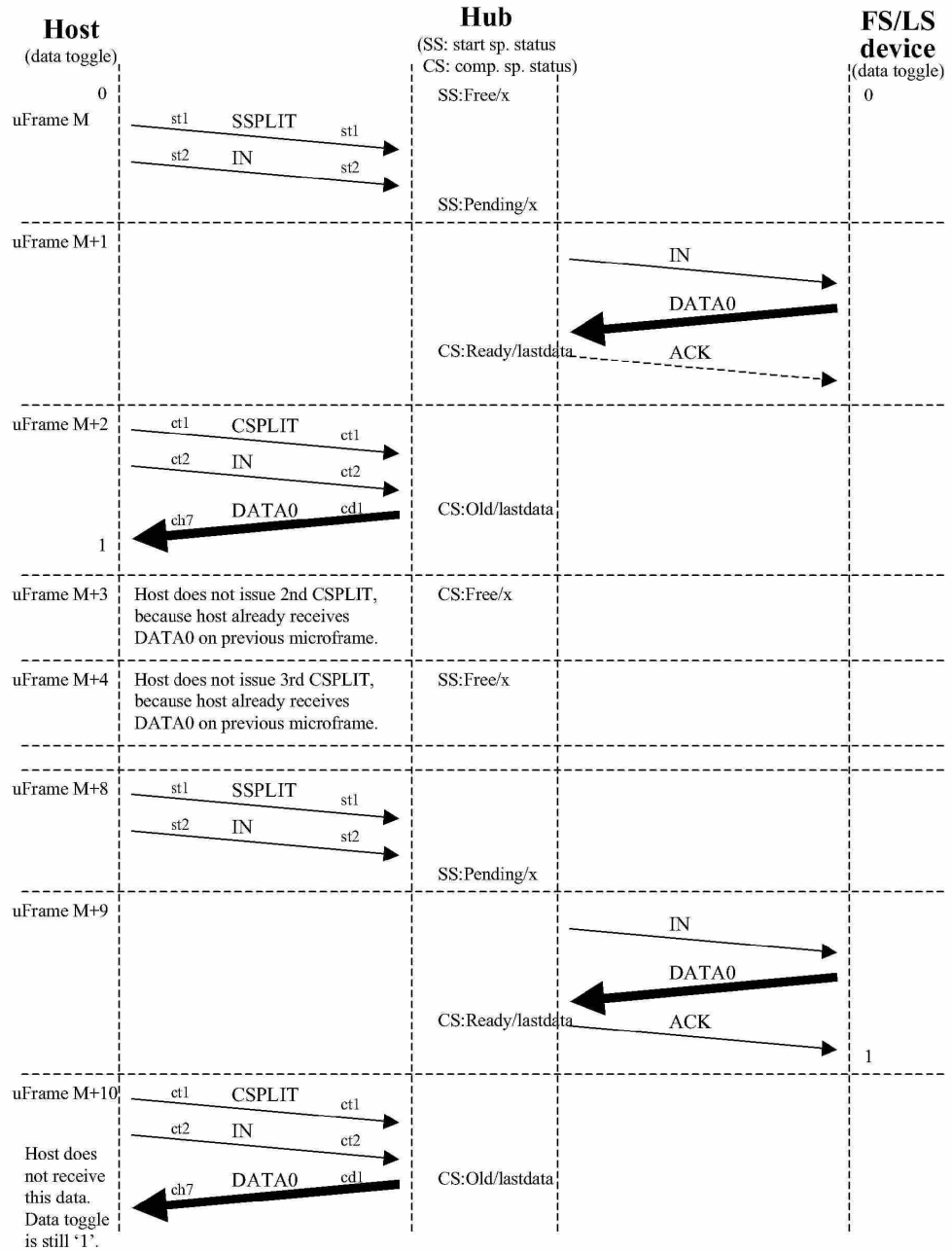


Figure A-72. Normal FS/LS ACK Smash

Universal Serial Bus Specification Revision 2.0

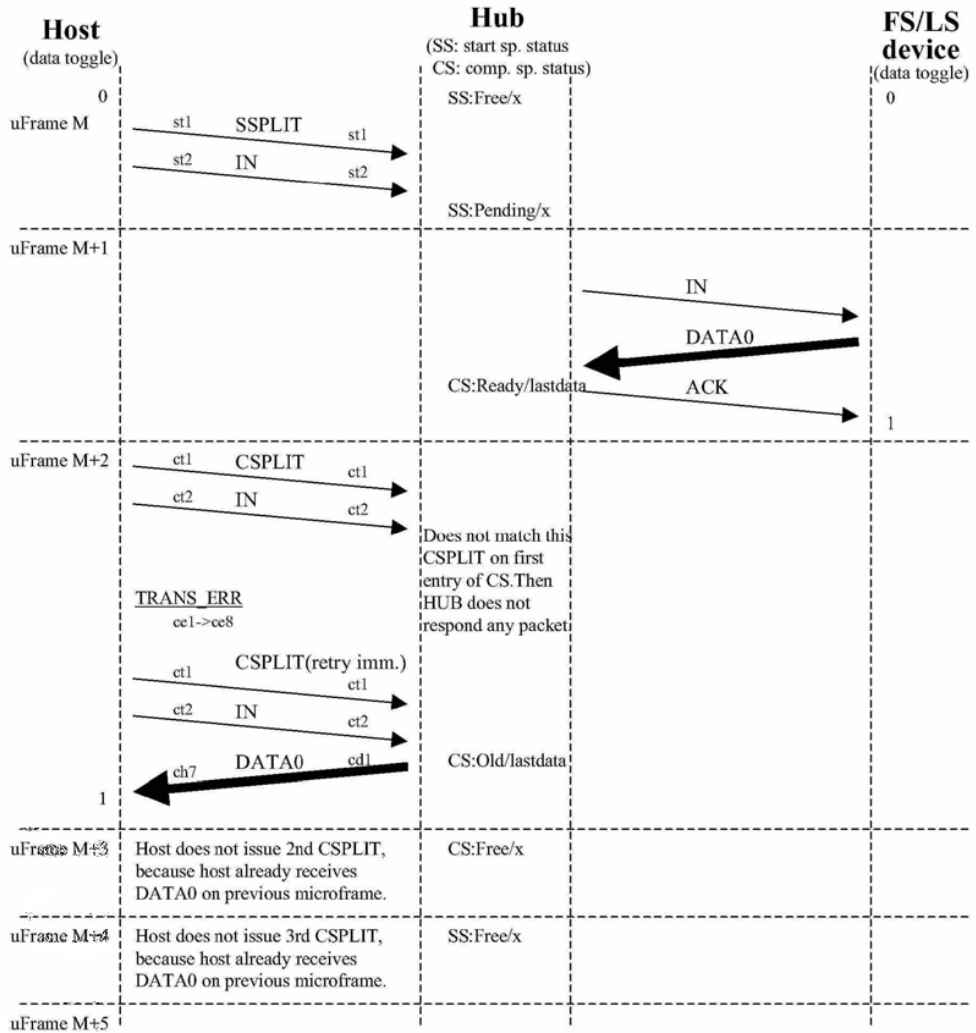


Figure A-73. Searching No Smash

Universal Serial Bus Specification Revision 2.0

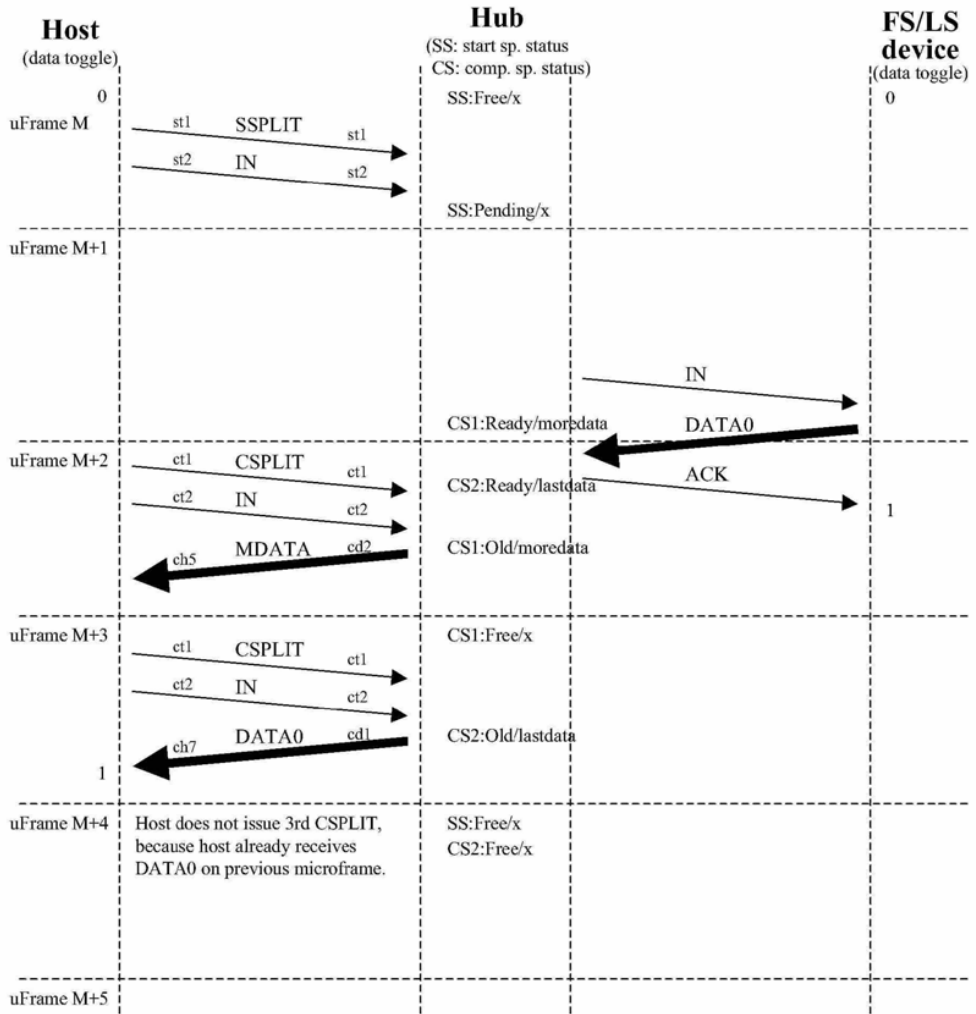


Figure A-74. CS Earlier No Smash(HS MDATA and FS/LS Data Packet is on M+1 and M+2)

Universal Serial Bus Specification Revision 2.0

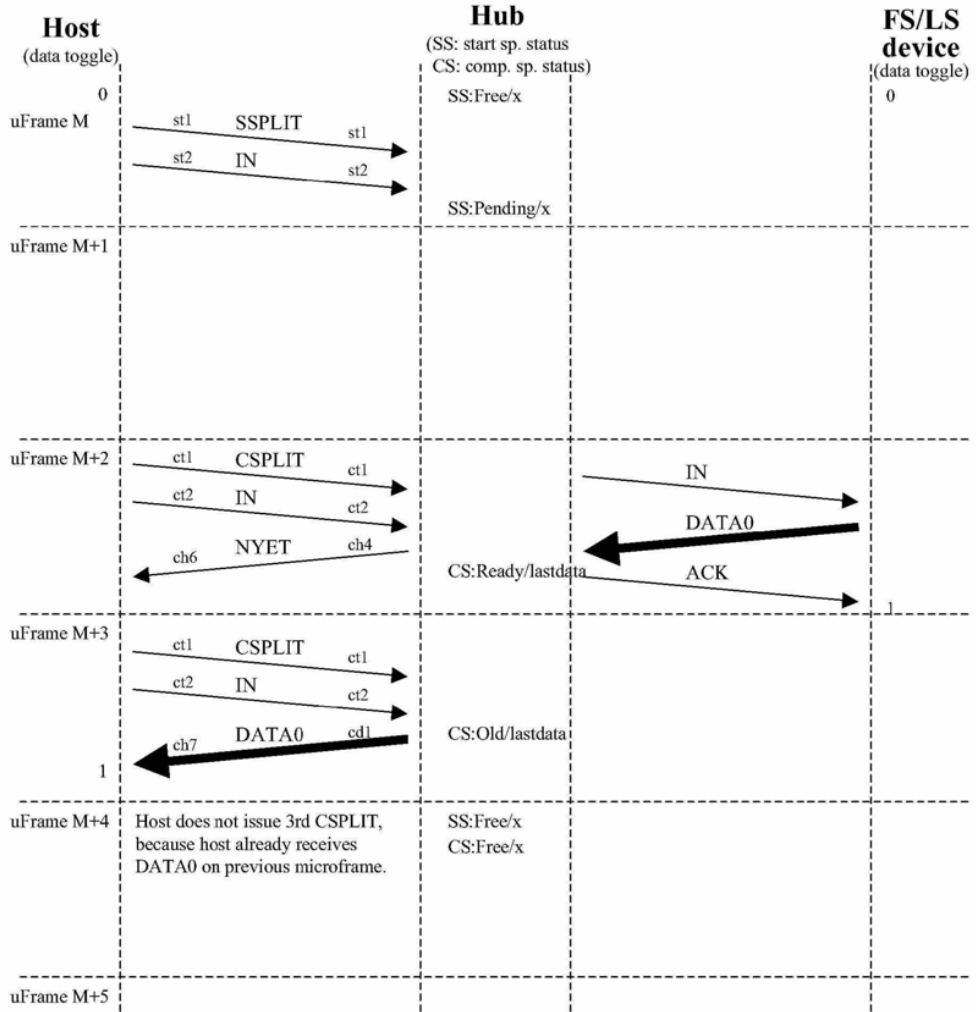


Figure A-75. CS Earlier No Smash(HS NYET and FS/LS Data Packet is on M+2)

Universal Serial Bus Specification Revision 2.0

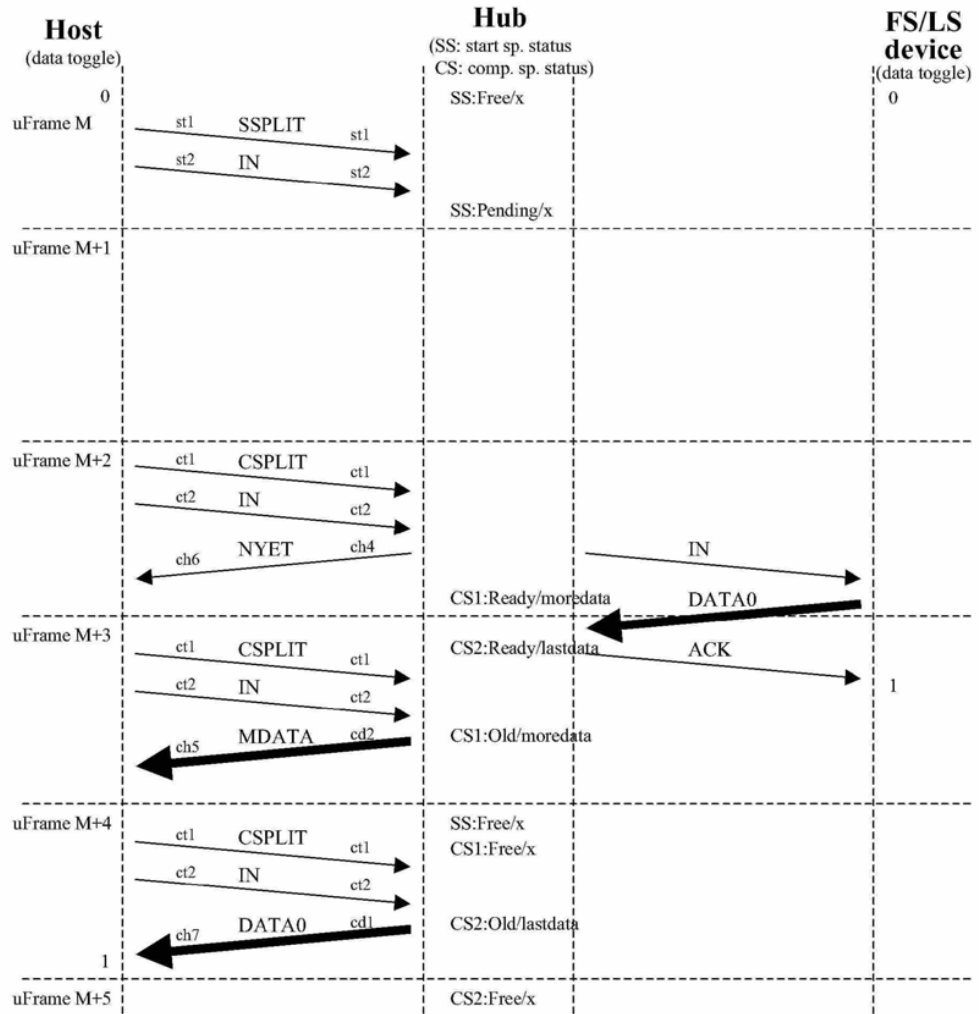


Figure A-76. CS Earlier No Smash(HS NYET and MDATA and FS/LS Data Packet is on M+2 and M+3)

Universal Serial Bus Specification Revision 2.0

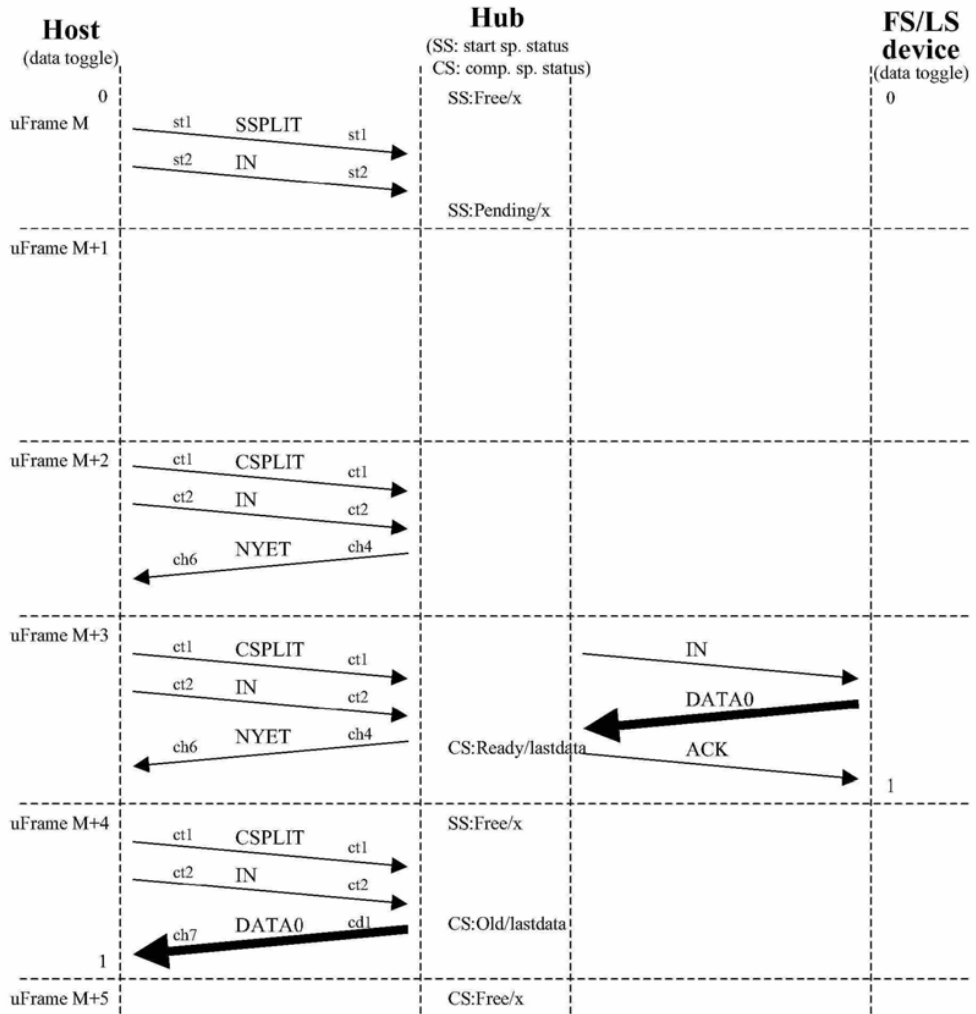


Figure A-77. CS Earlier No Smash(HS NYET and FS/LS Data Packet is on M+3)

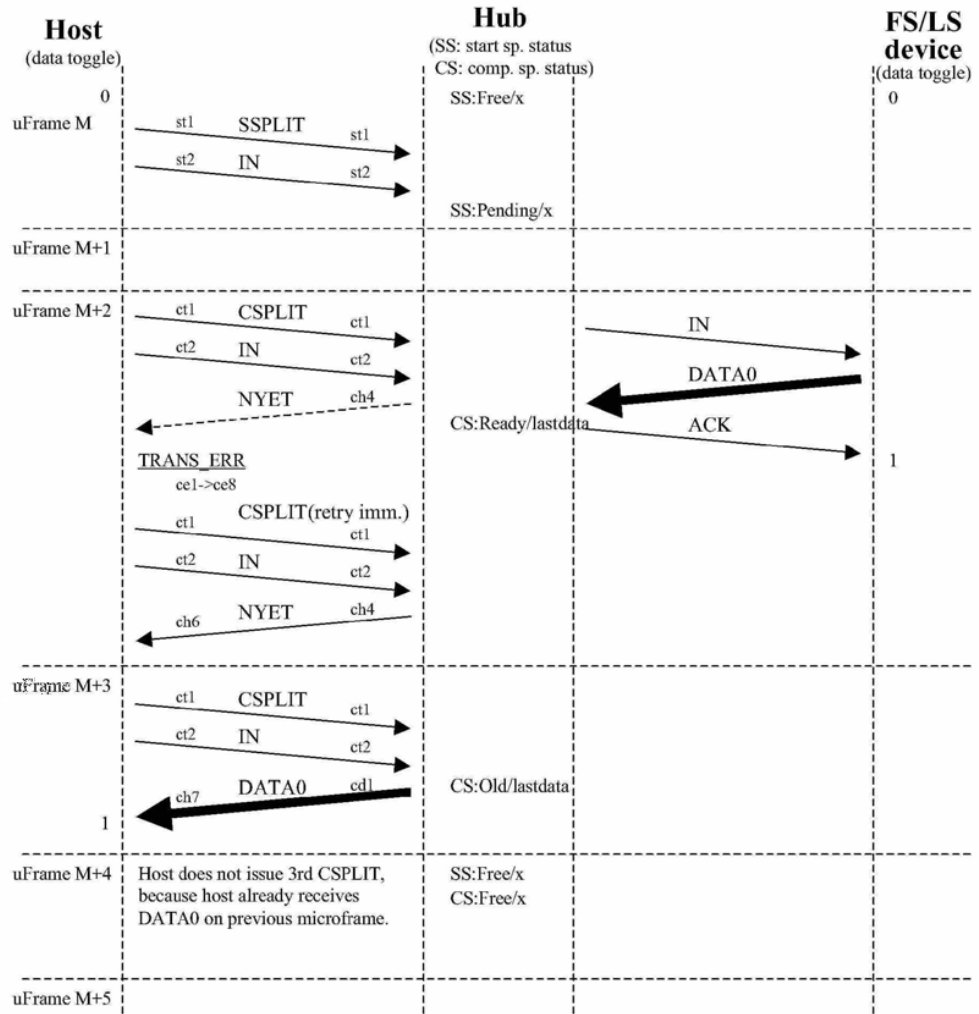


Figure A-78. CS Earlier HS NYET Smash

IN.CSPLIT earlier.HS NYET 3 strikes smash

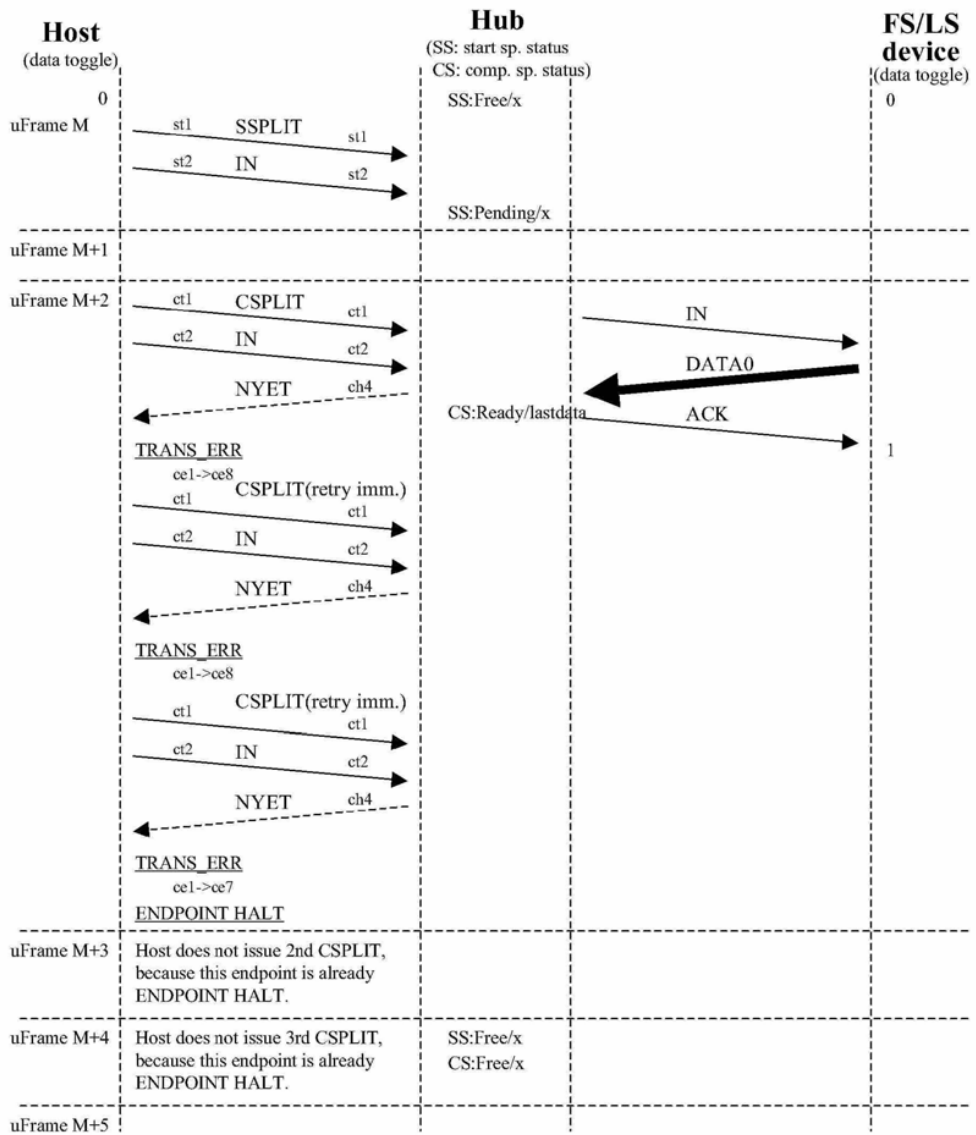


Figure A-79. CS Earlier HS NYET 3 Strikes Smash

Universal Serial Bus Specification Revision 2.0

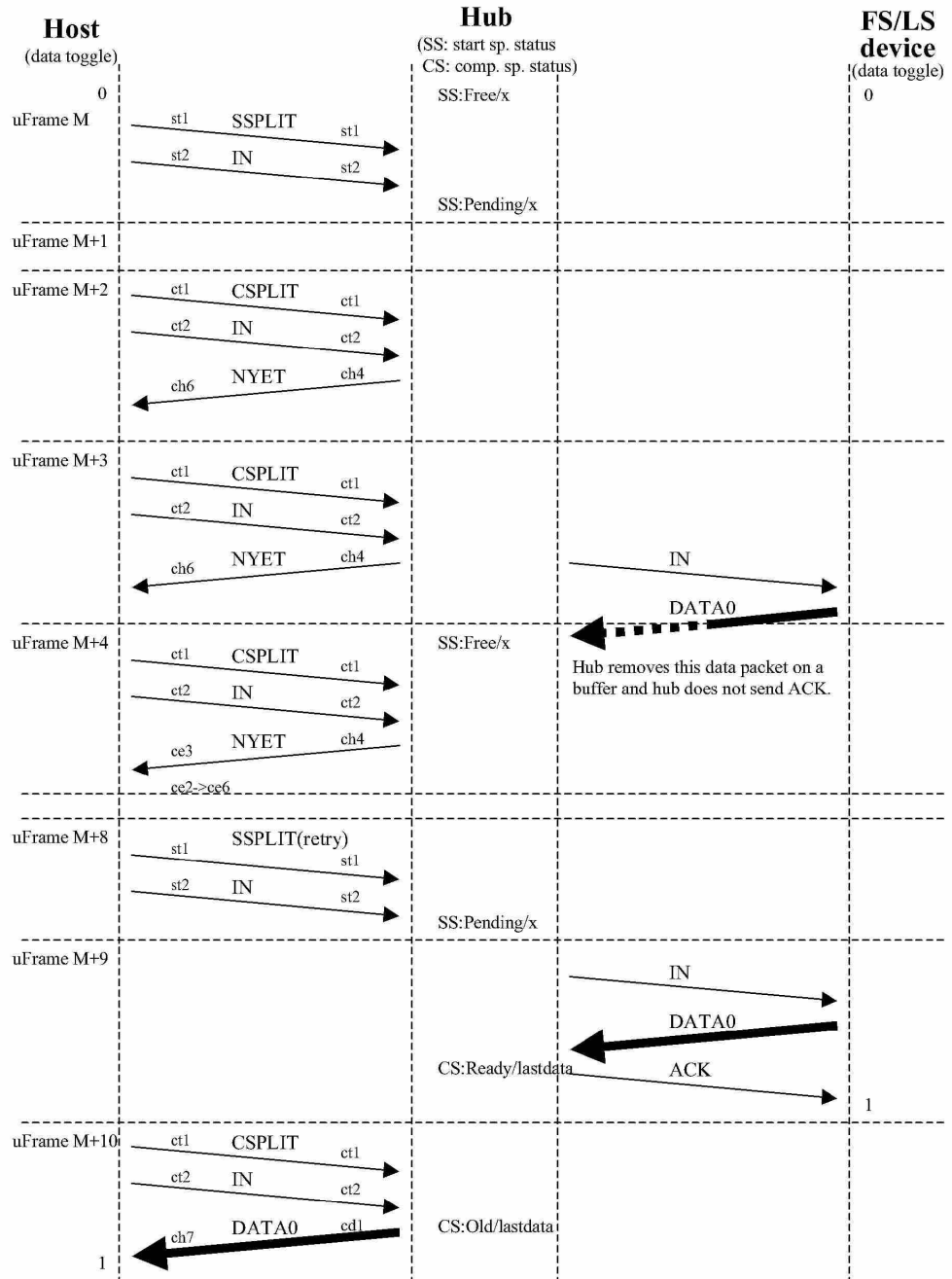


Figure A-80. Abort and Free Abort(HS NYET and FS/LS Transaction is Continued at End of M+3)

Universal Serial Bus Specification Revision 2.0

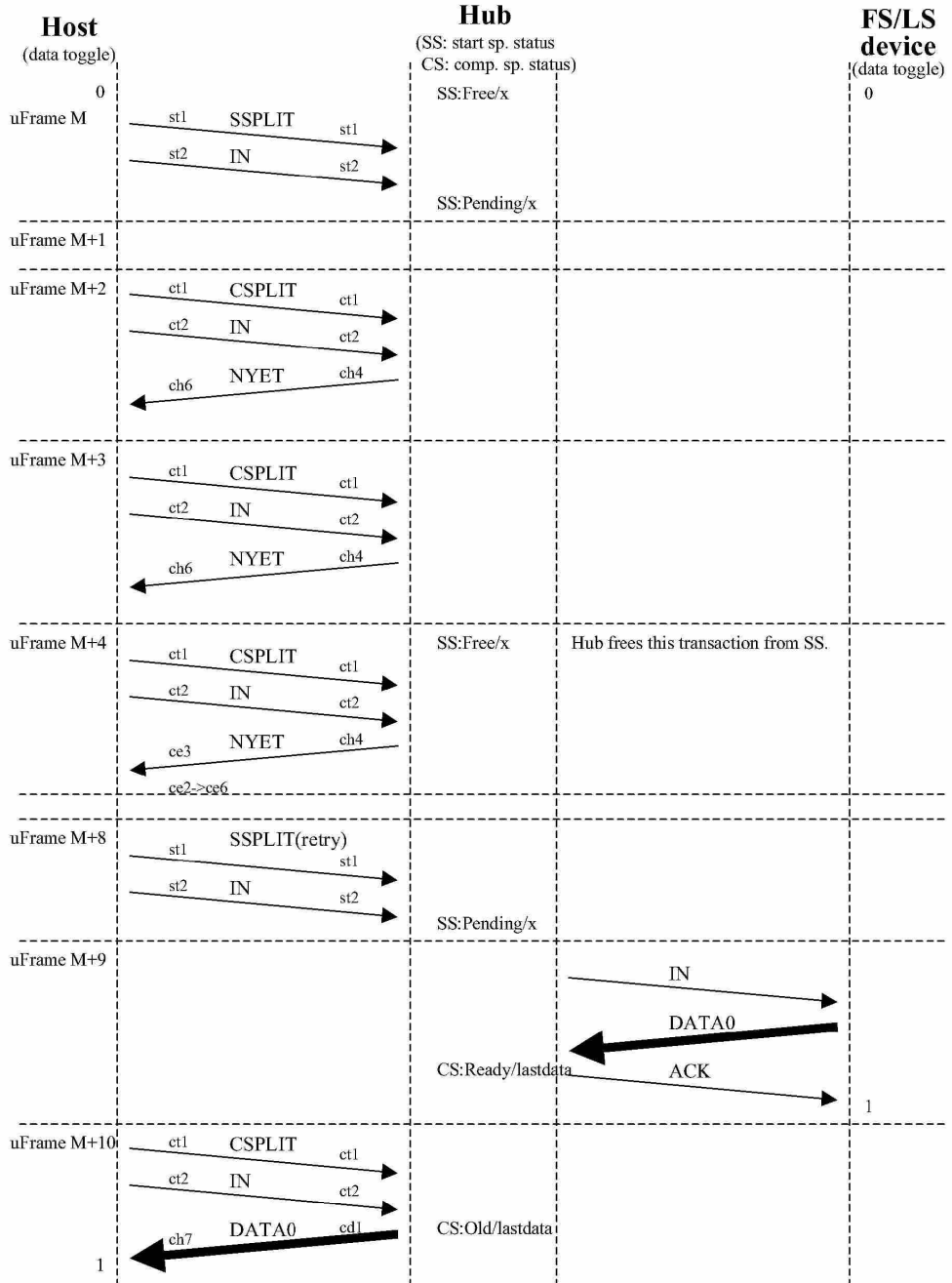


Figure A-81. Abort and Free Free(HS NYET and FS/LS Transaction is not Started at End of M+3)

Universal Serial Bus Specification Revision 2.0

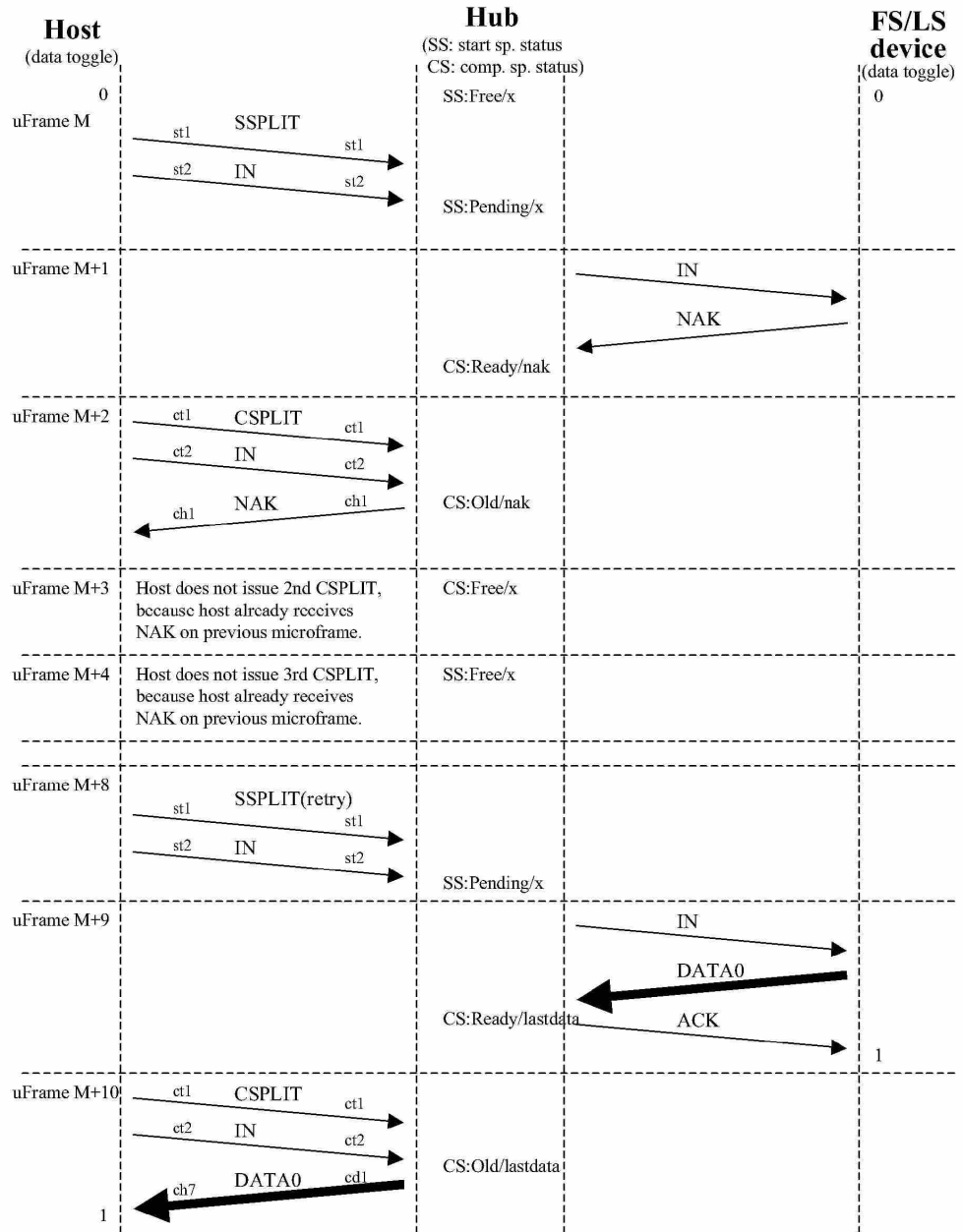


Figure A-82. Device Busy No Smash(FS/LS NAK)

Universal Serial Bus Specification Revision 2.0

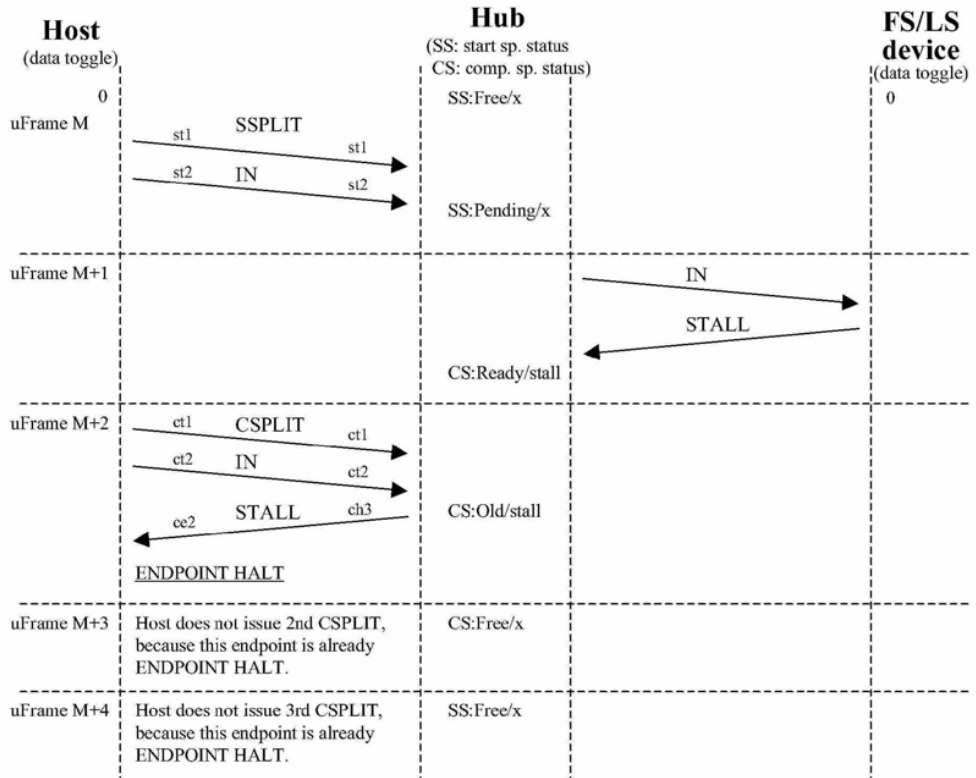
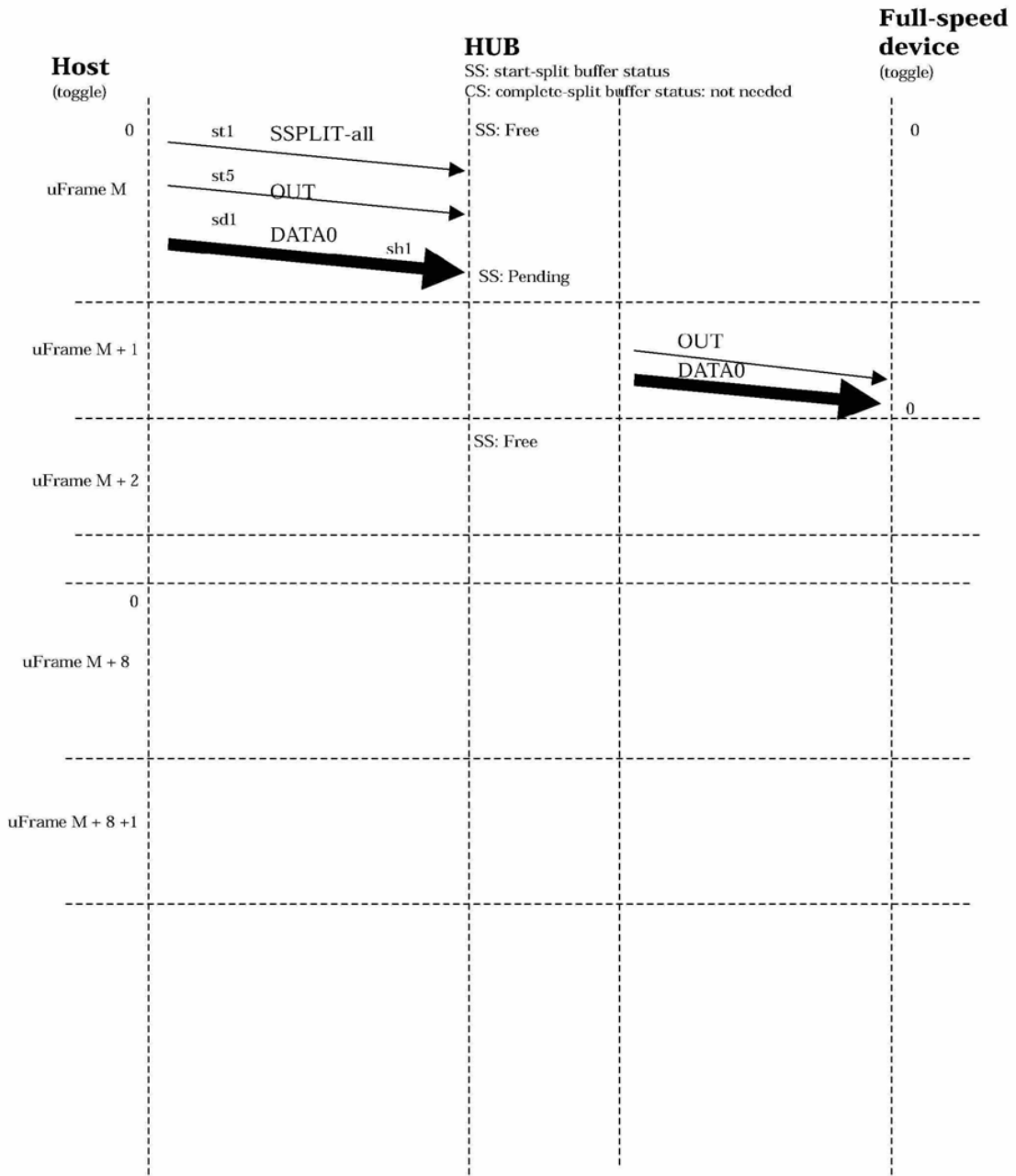


Figure A-83. Device Stall No Smash(FS/LS STALL)

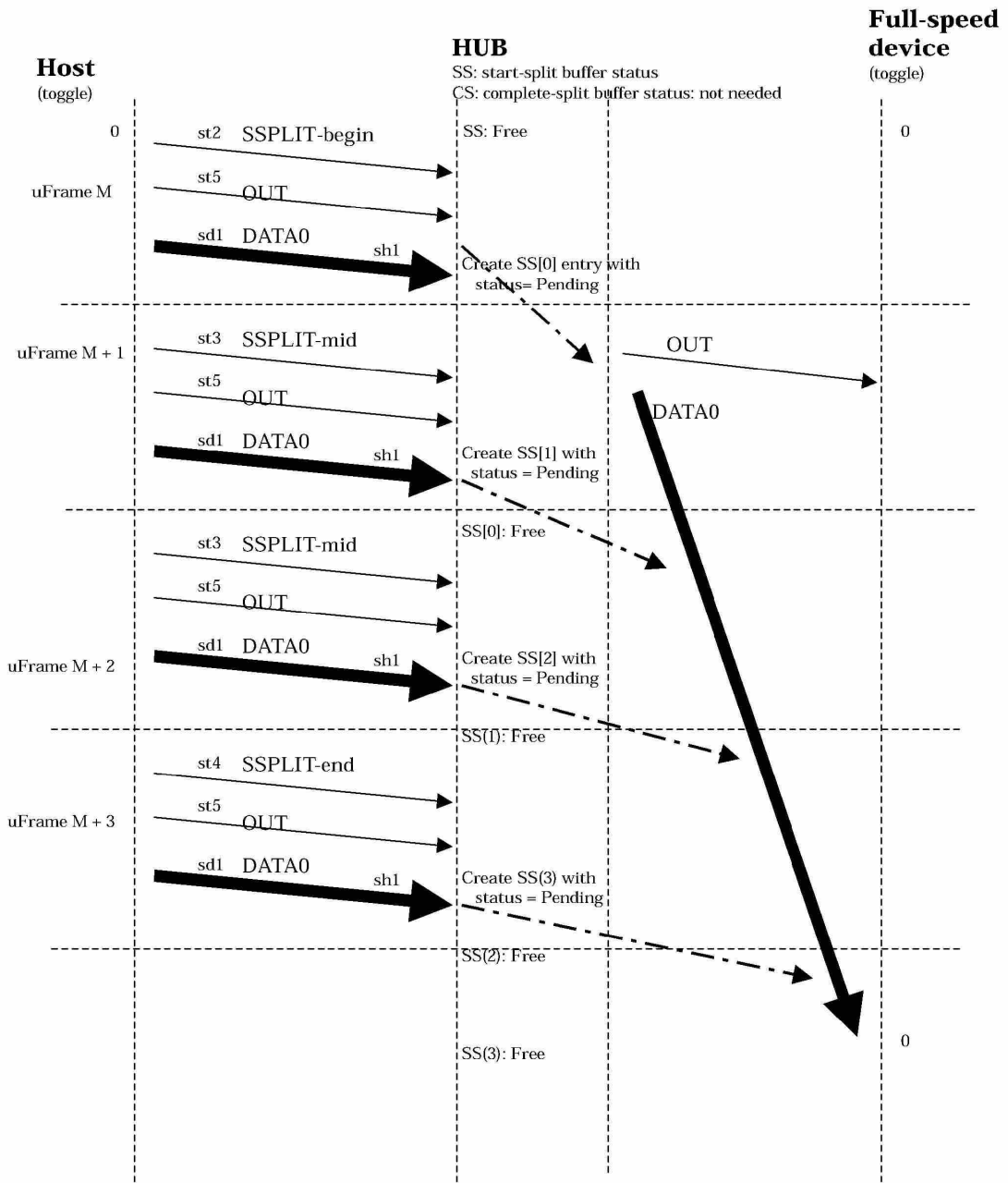
A.5 Isochronous OUT Split-transaction Examples

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

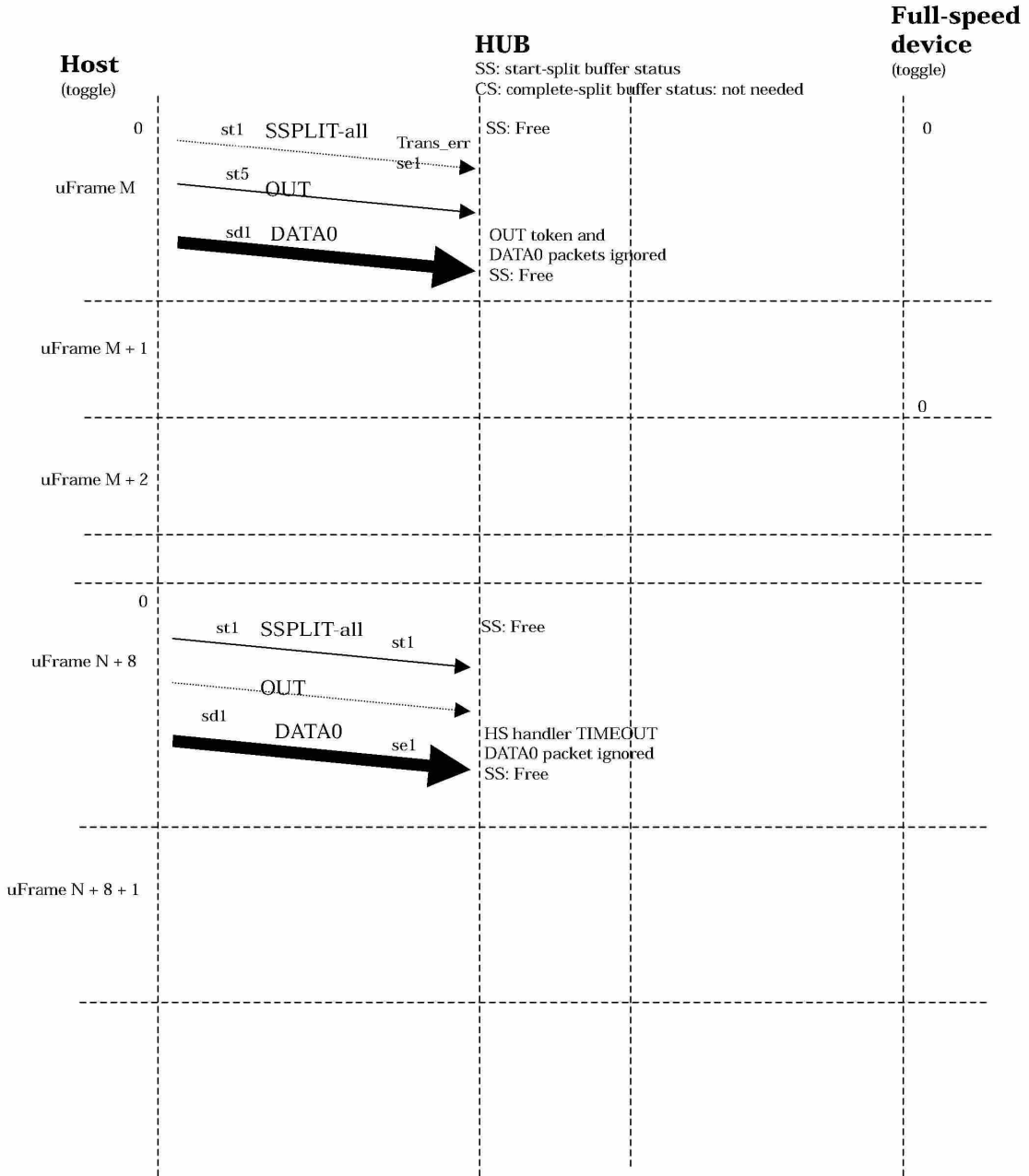
1) Normal. Payload <= 188 bytes:



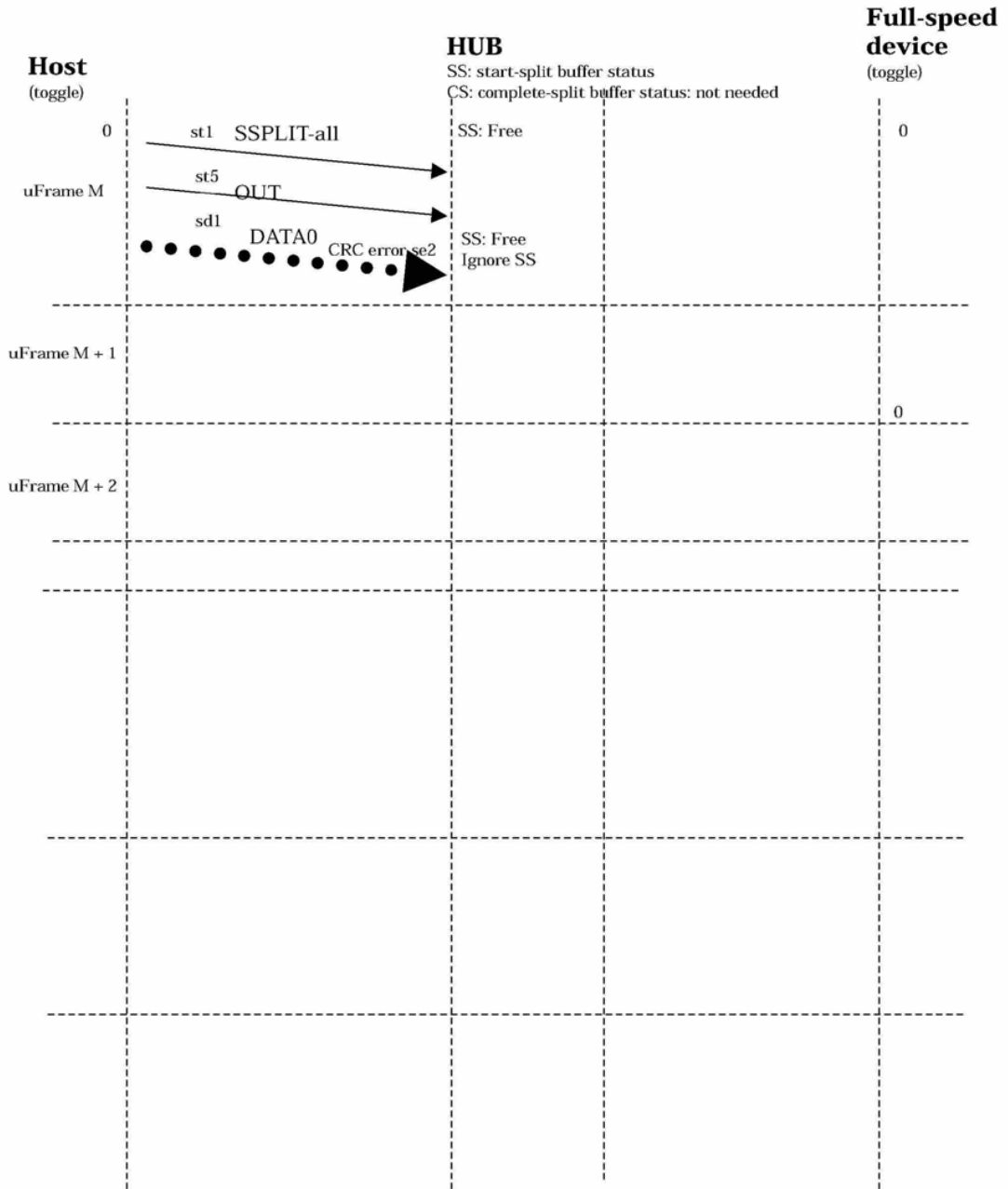
2) Normal. Payload > 188 Bytes



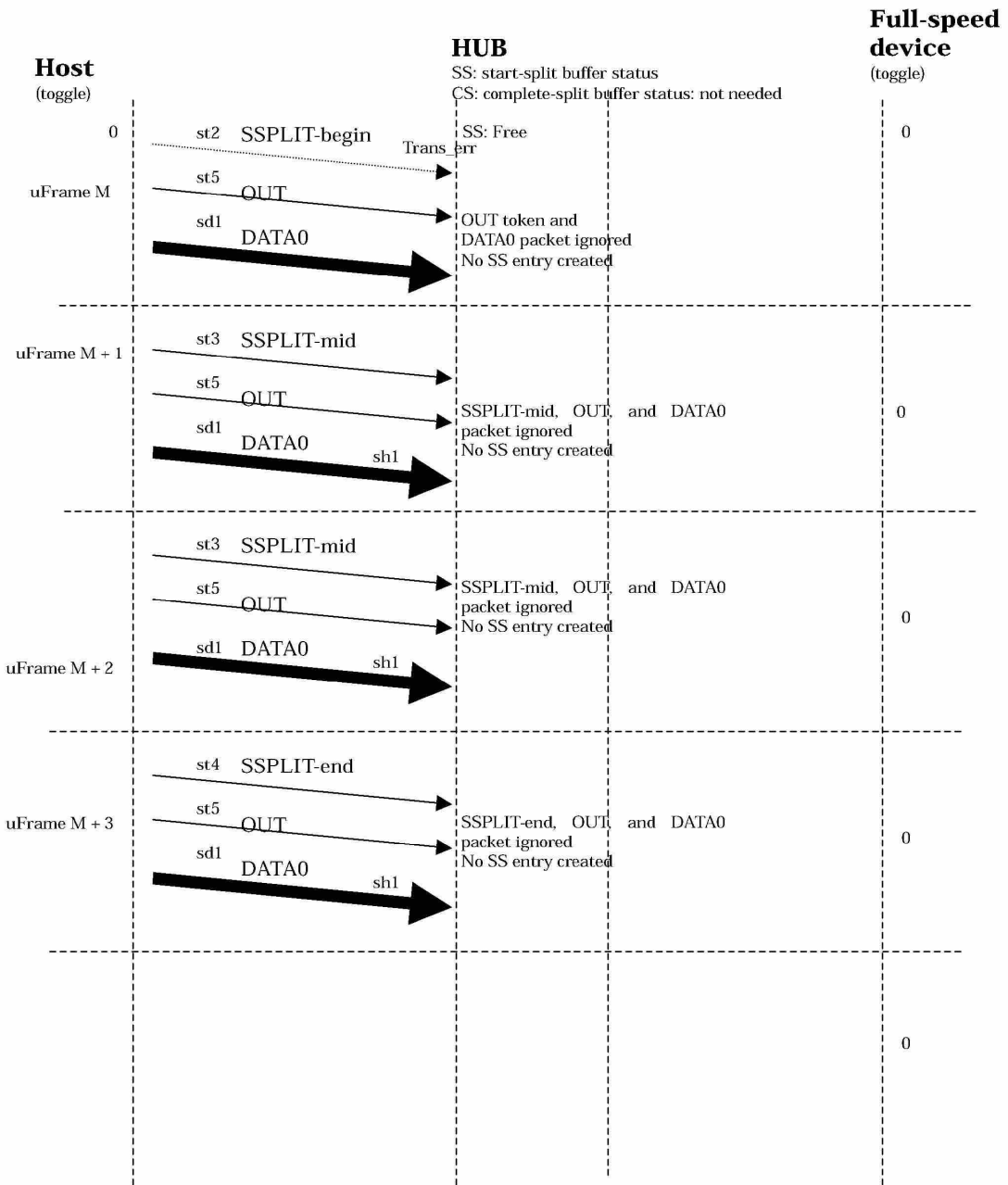
3) HS SSPLIT-all corrupted (missing or CRC error etc.)
 HS OUT corrupted



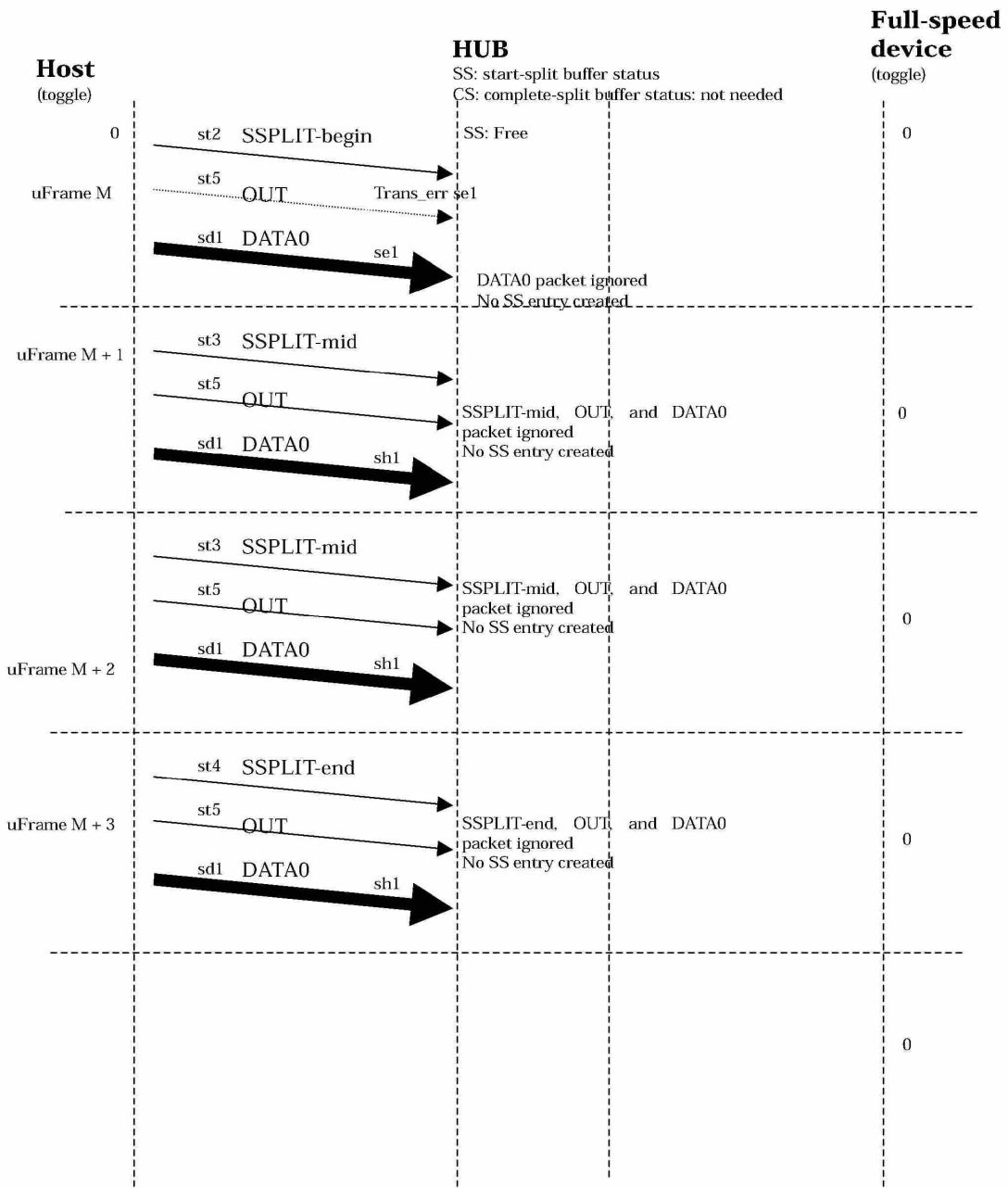
4) HS DATA0 corrupted



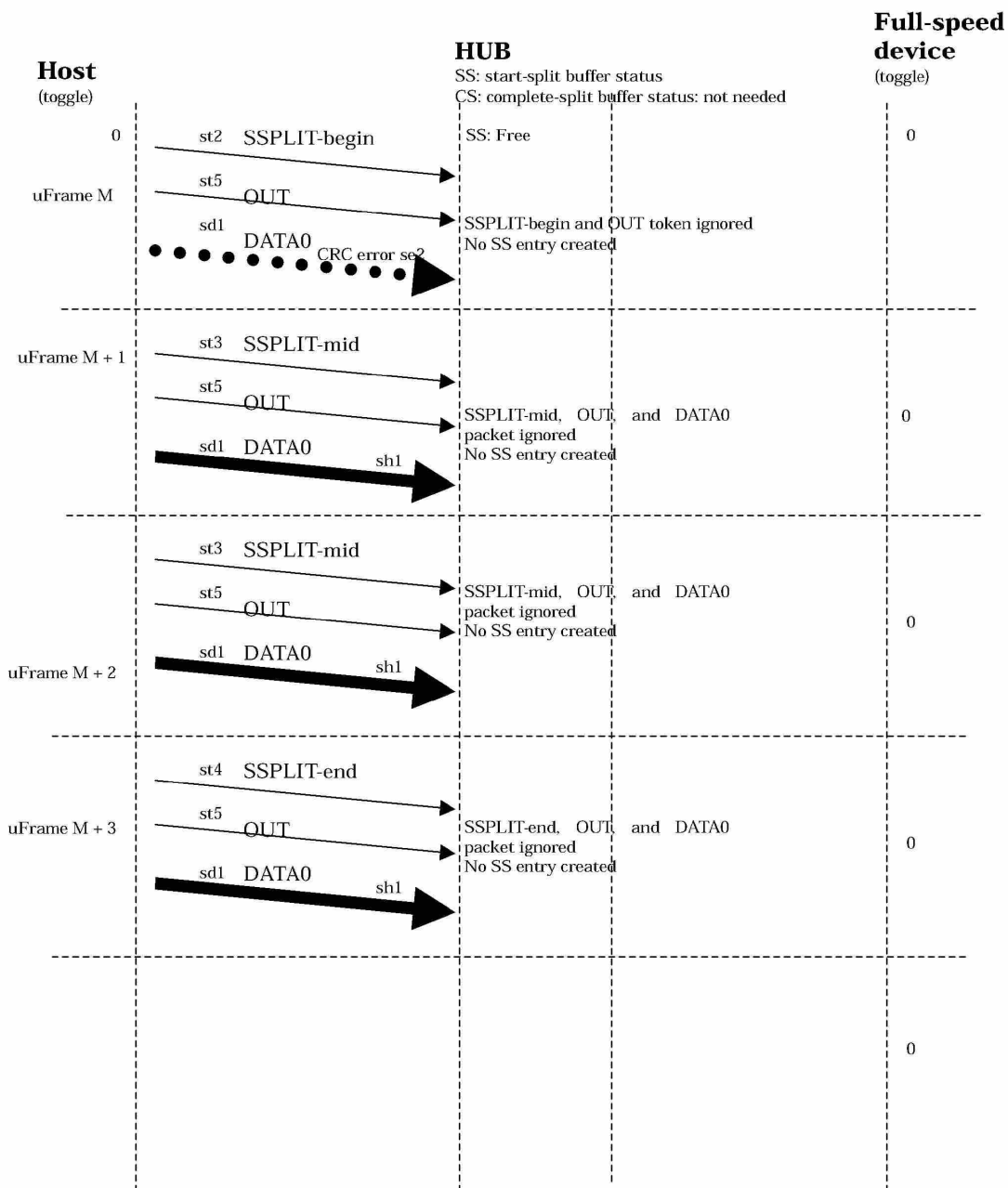
5) HS SSPLIT-begin corrupted (missing or CRC error etc.)



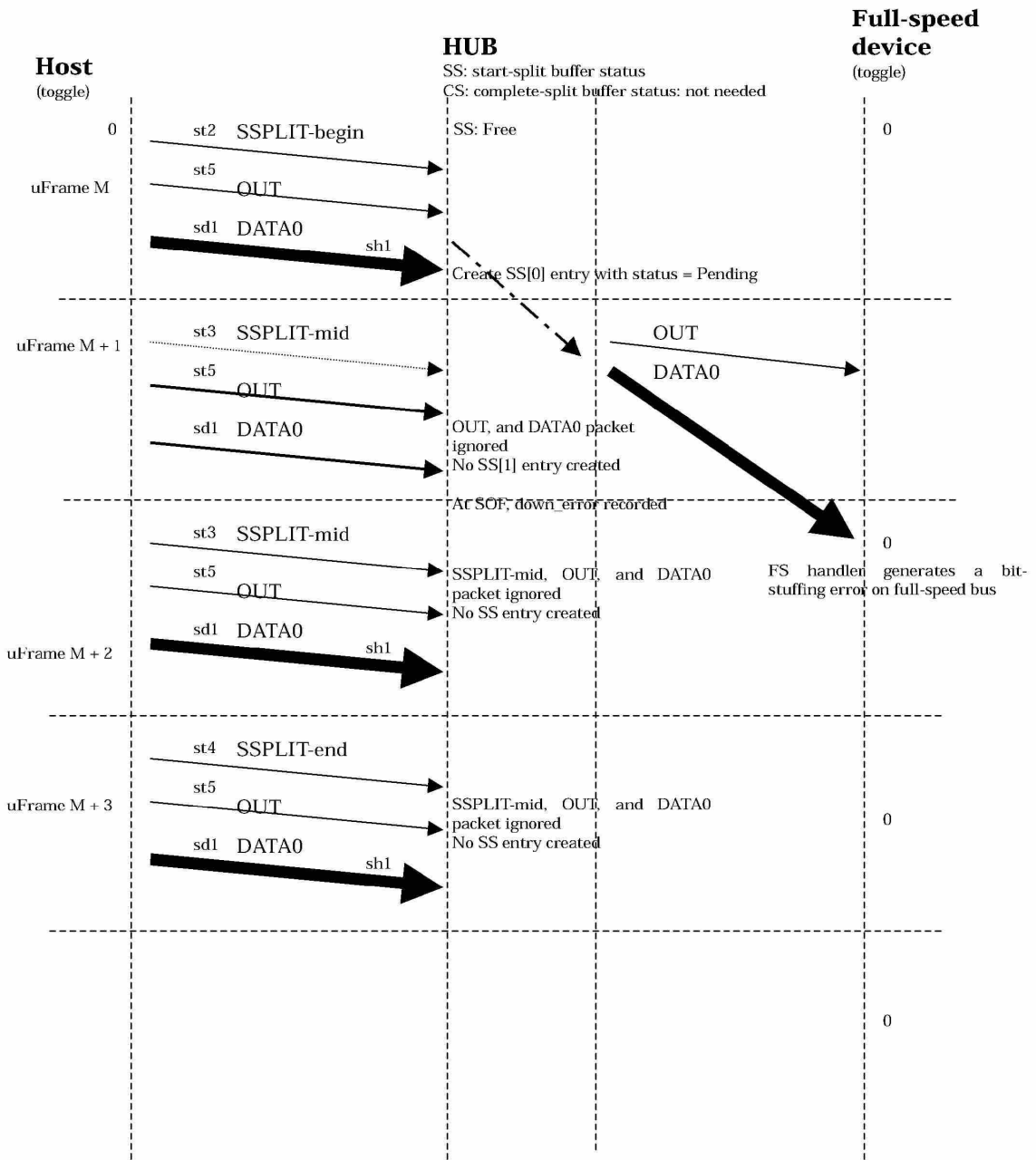
6) HS OUT after the HS SSPLIT-begin is corrupted



7) HS DATA0 corrupted



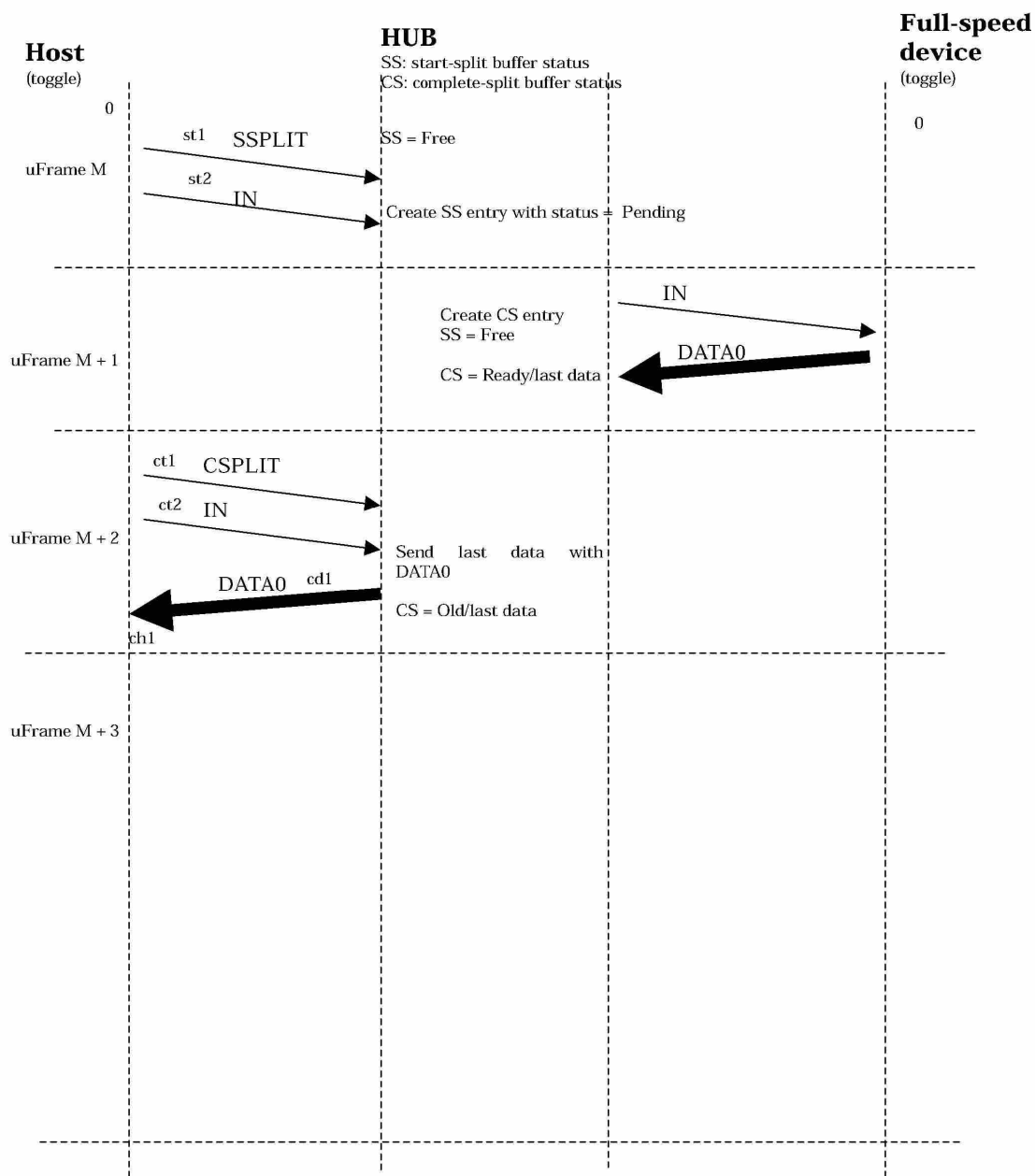
8) HS SSPLIT-mid or OUT token or DATA0 packet after it is corrupted



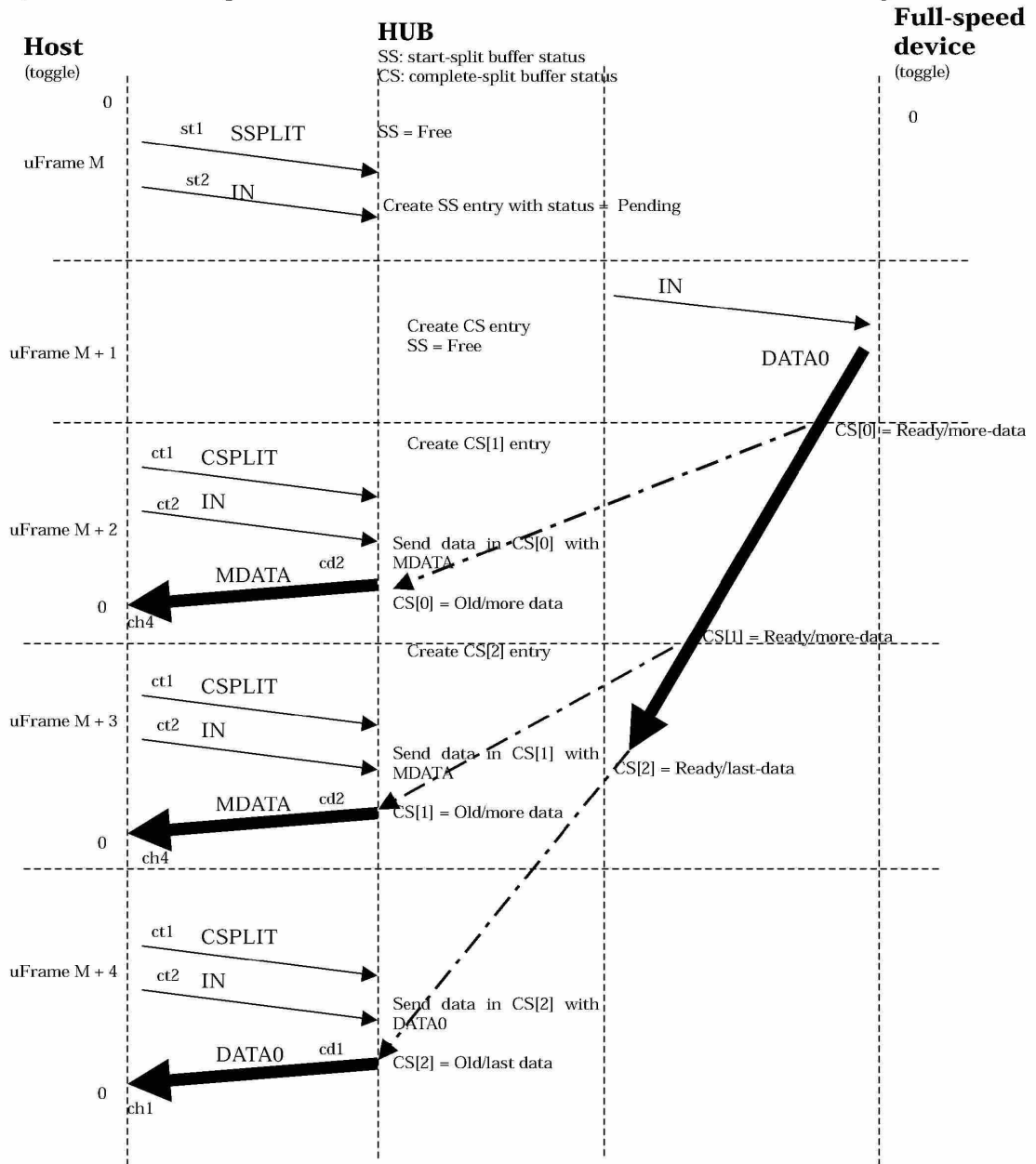
A.6 Isochronous IN Split-transaction Examples

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

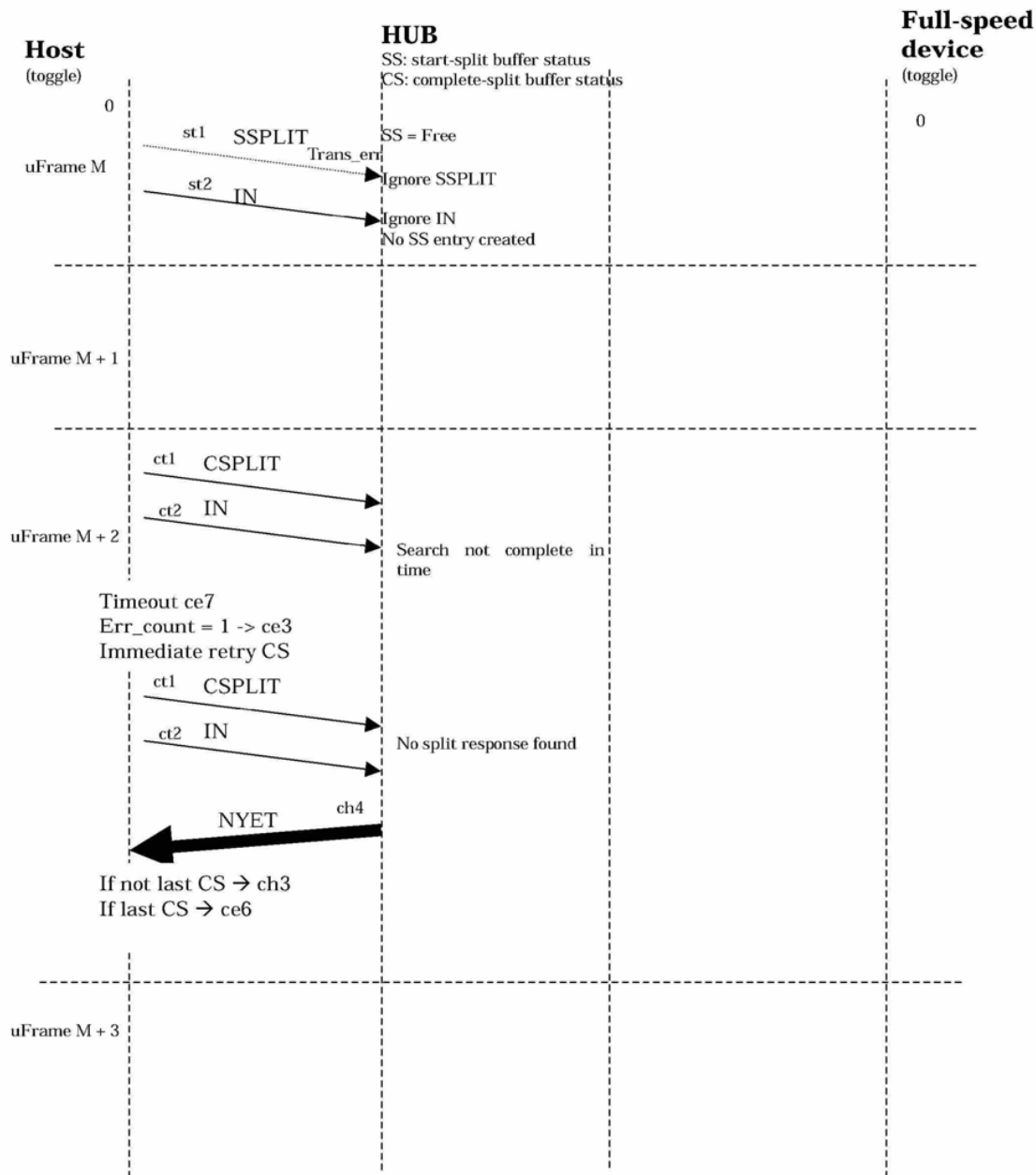
1) Normal: full-speed bus transaction does not cross microframe boundary



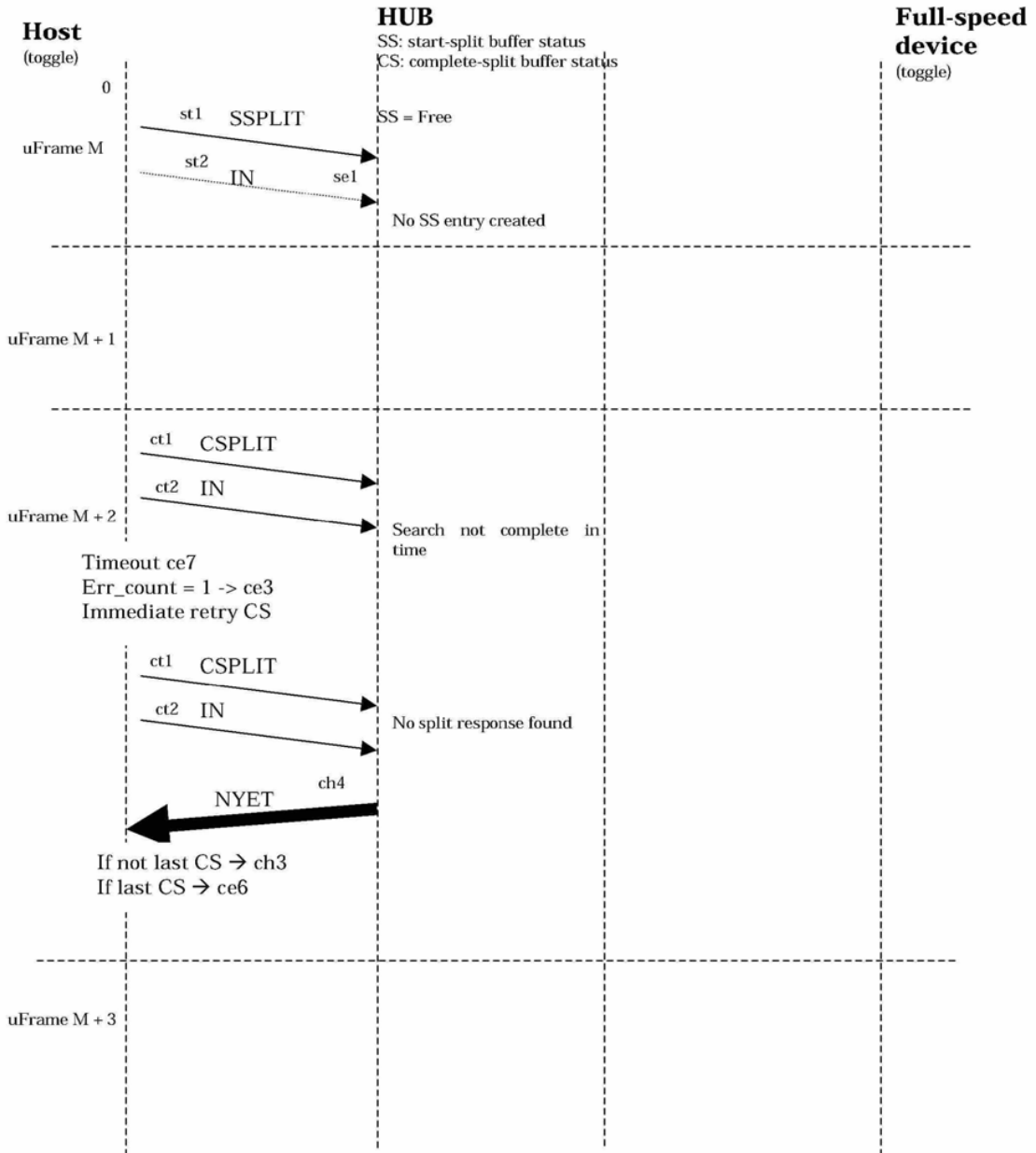
2) Normal: full-speed bus transaction crosses microframe boundary



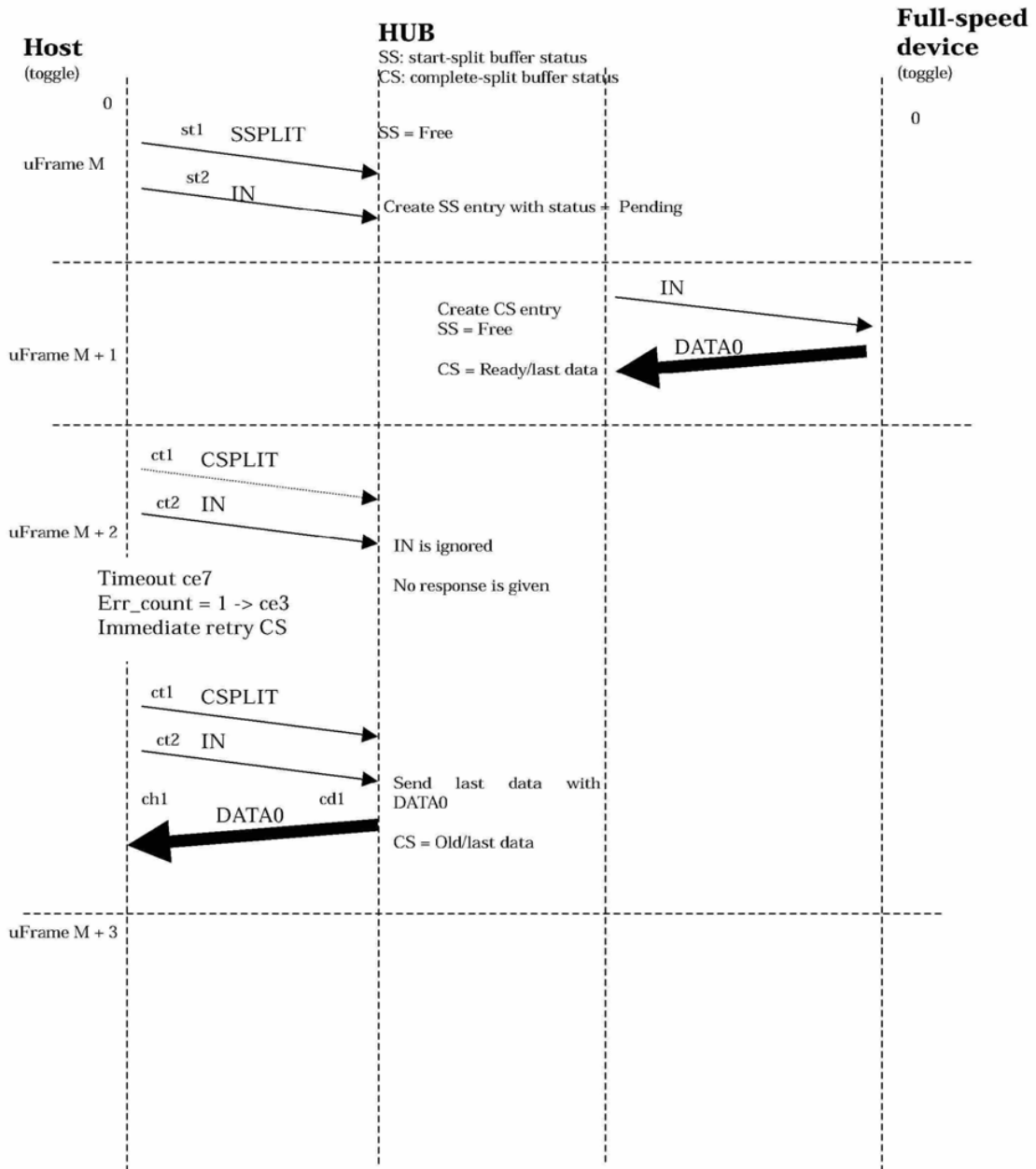
3) HS SSPLIT corrupted



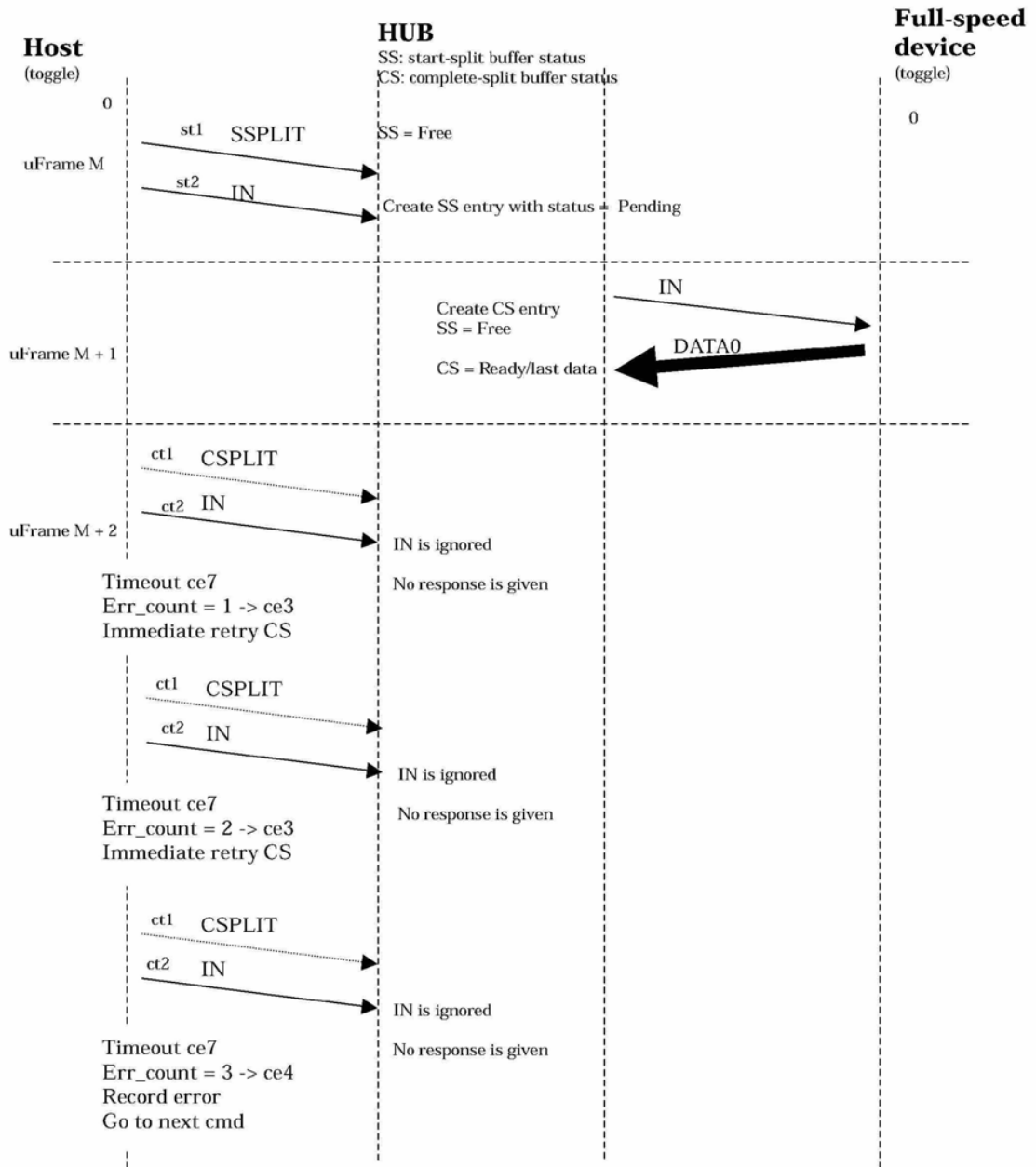
4) IN after HS SSPLIT corrupted



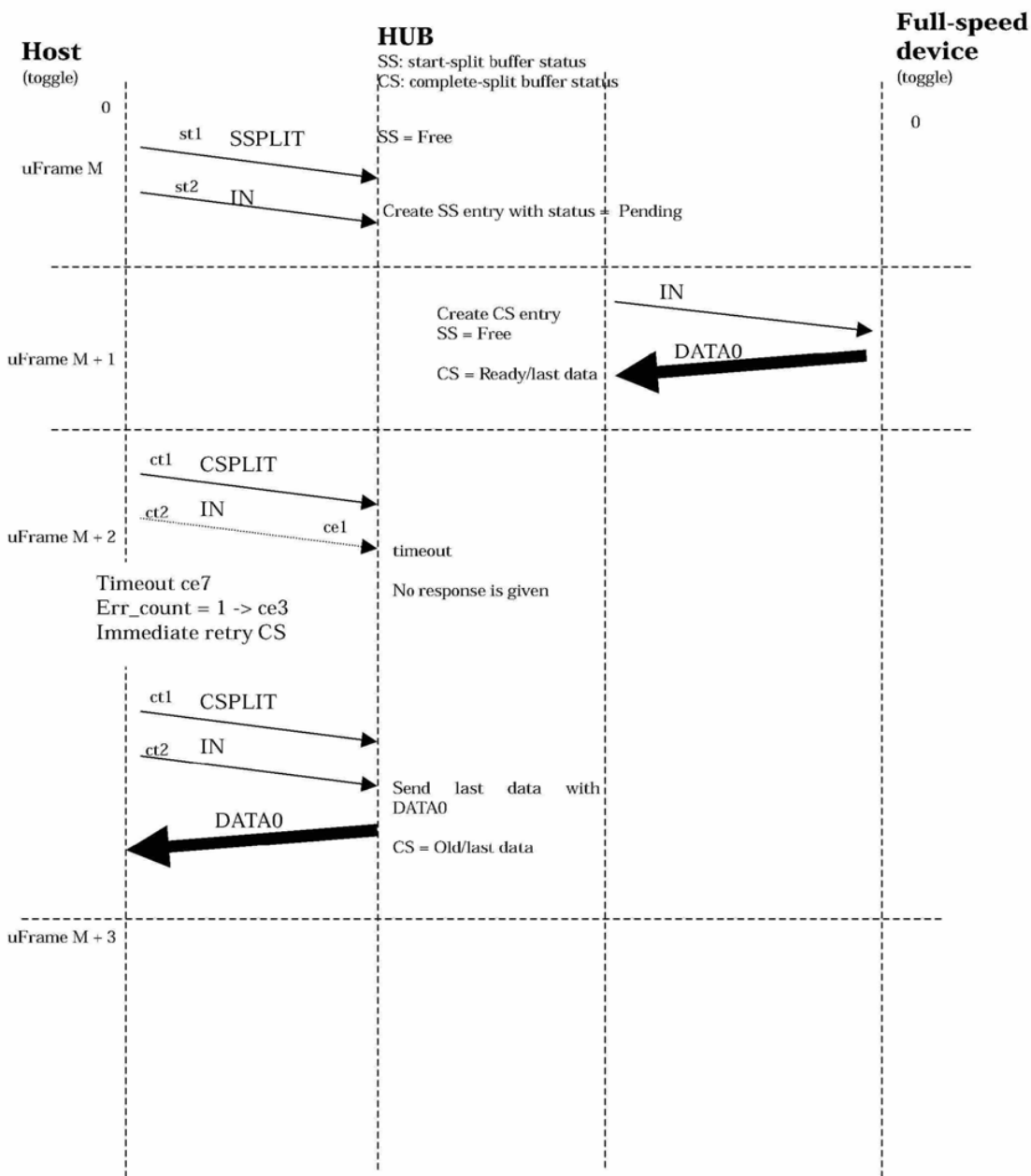
5) HS CSPLIT corrupted



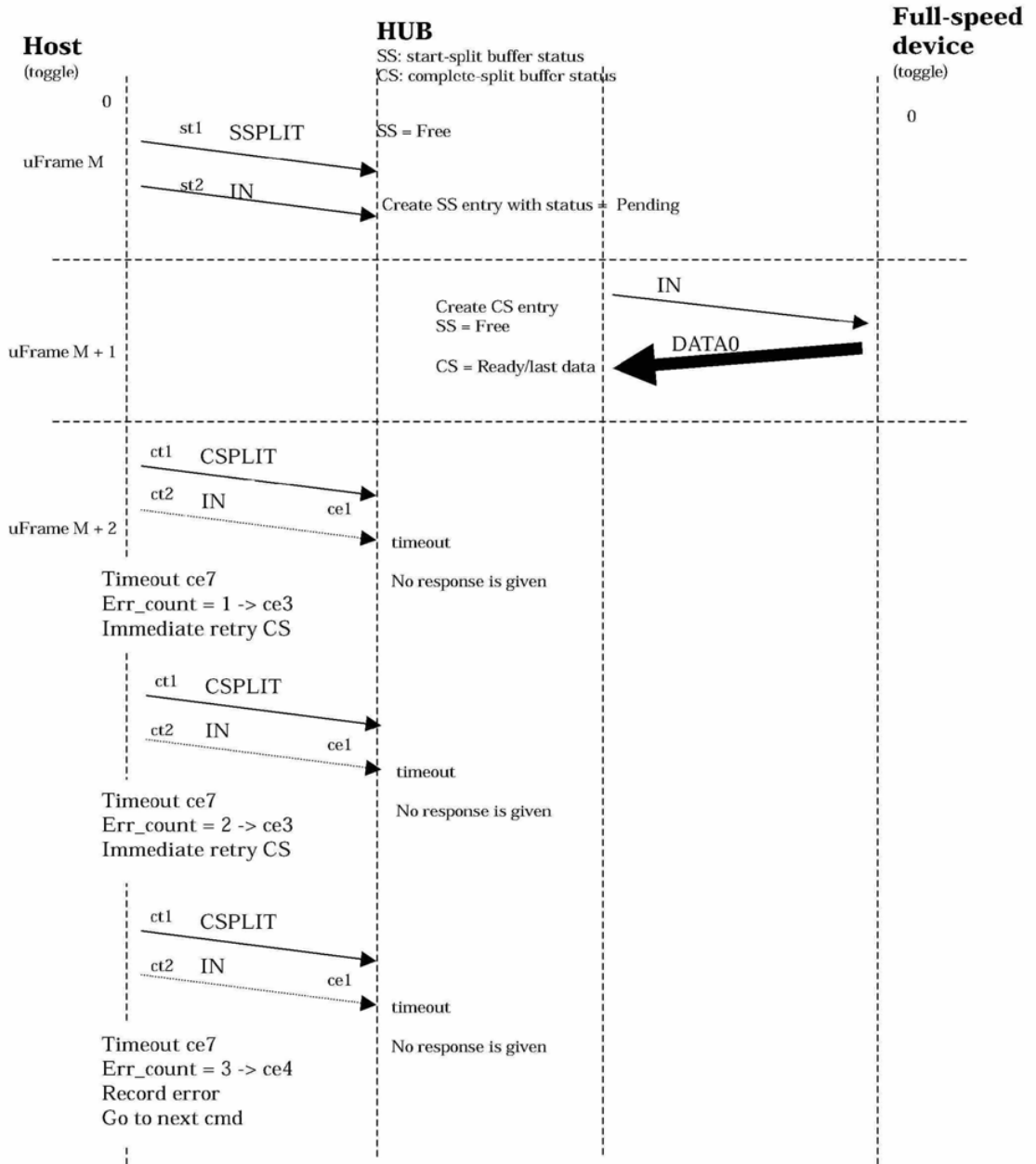
6) Consecutive HS CSPLIT corrupted



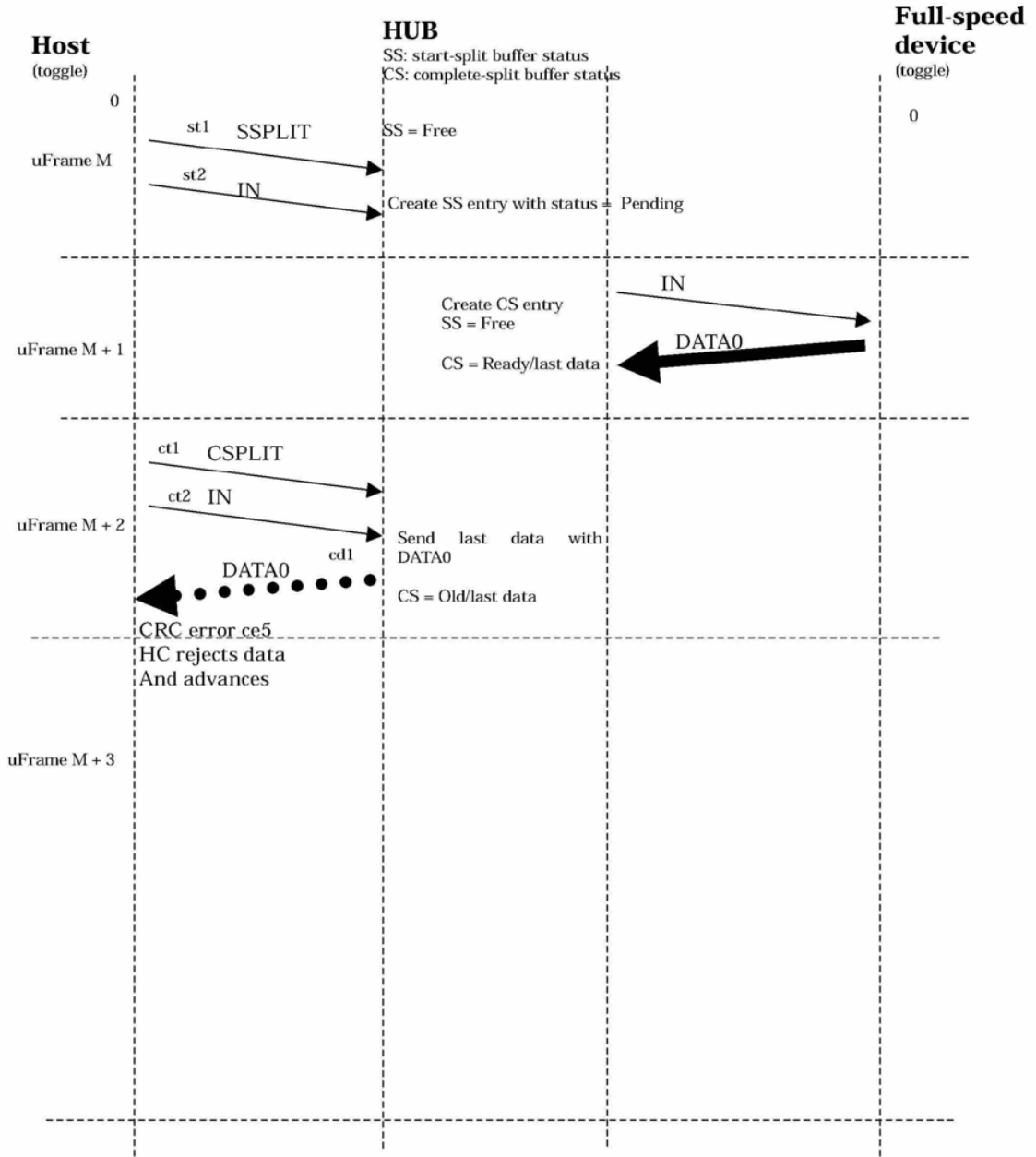
7) HS IN corrupted



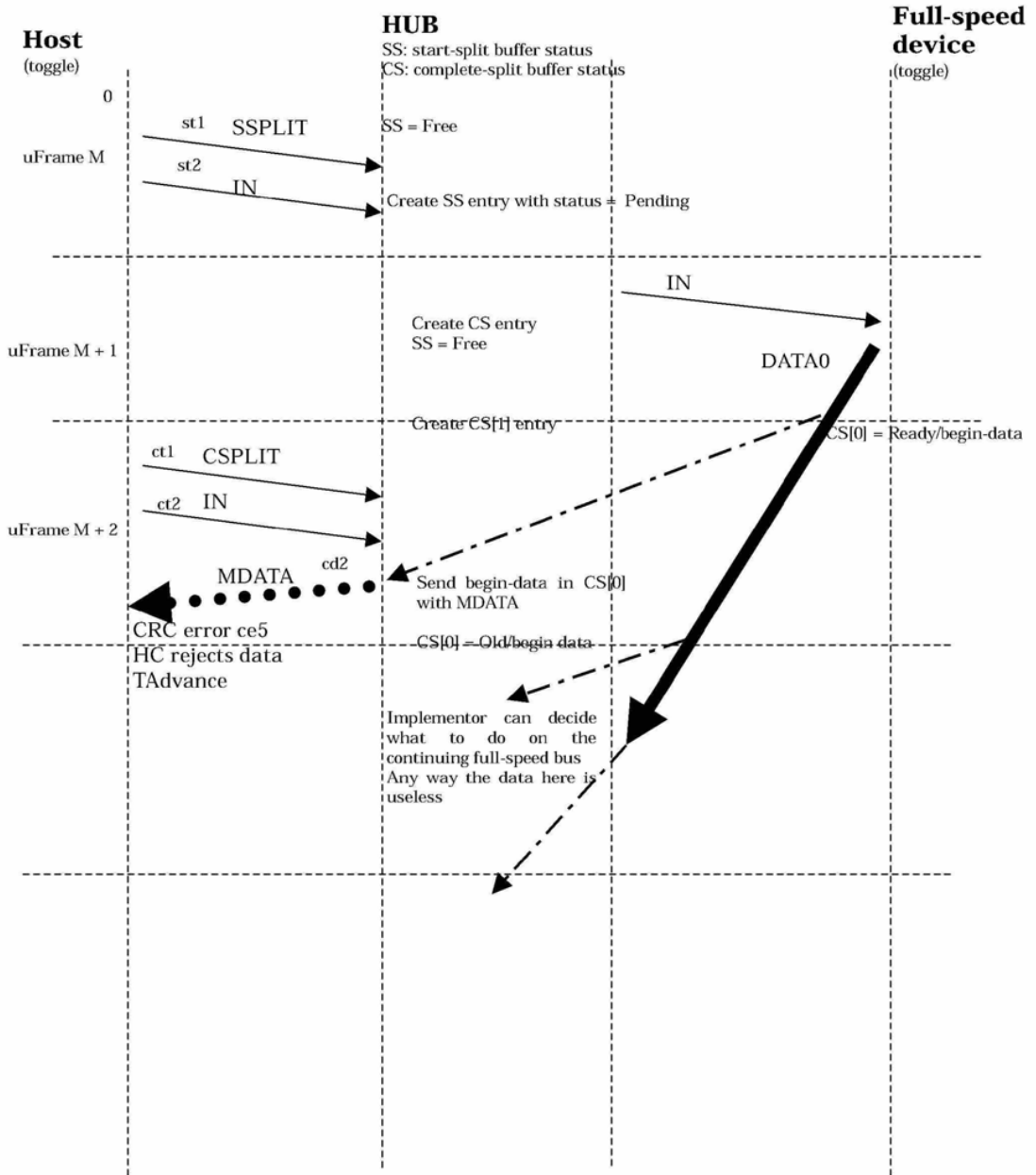
8) Consecutive HS IN corrupted



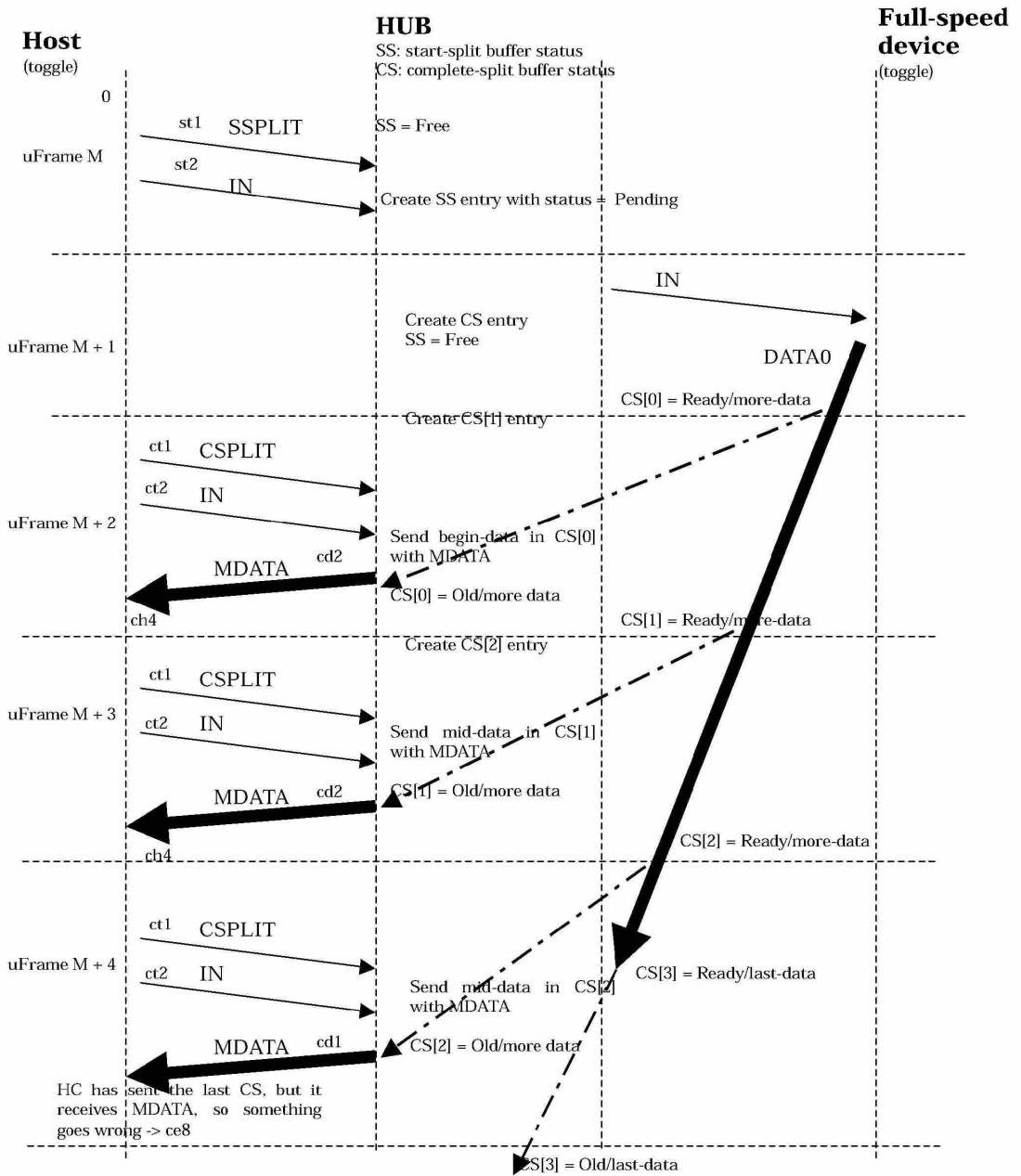
9) HS data corrupted (case 1)



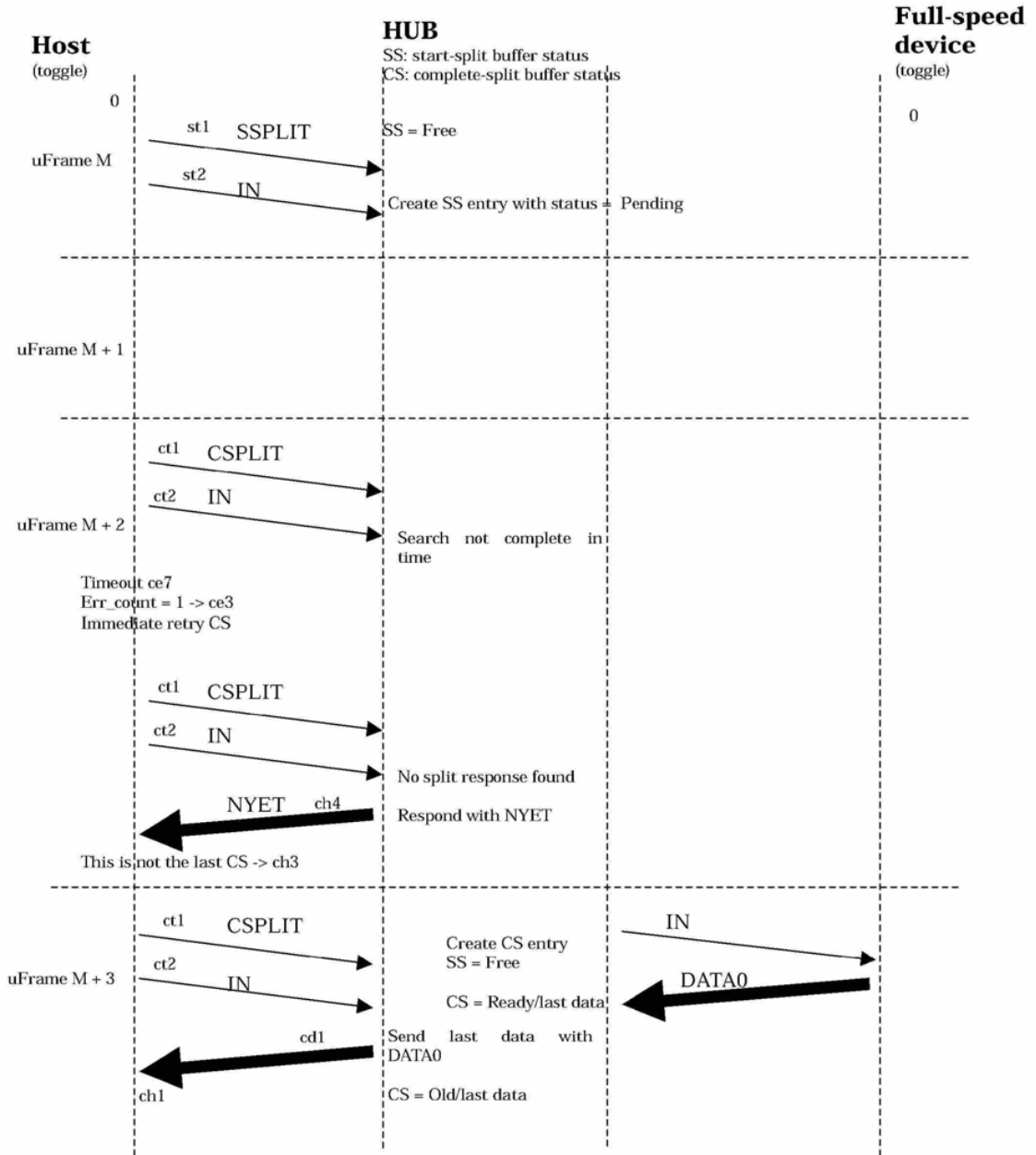
10) HS data corrupted (case 2)



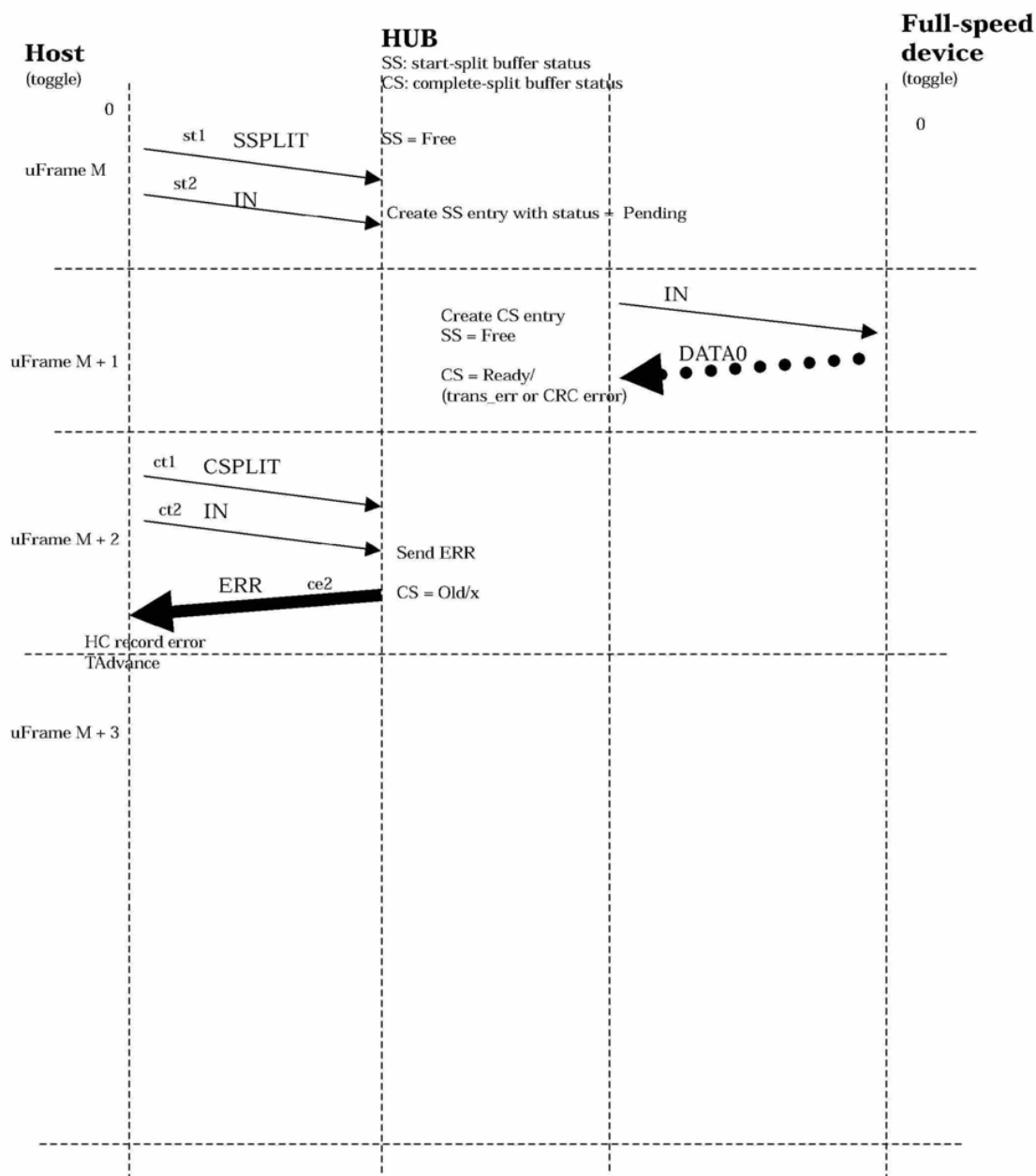
11) TT has more data than HC expects



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;
    
```

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.isoch0) OR
((split.datapart = middata OR split.datapart = enddata) AND NOT
SS_Buff.isoch0);

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 $T_{\text{TIMER}}(\text{min})$ and $T_{\text{TIMER}}(\text{max})$, 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 $2.5 \mu\text{s}$ (T_{FILT}) filtering the reset protocol asks for.

Note: The timer T1 is required to be reset after an interruption of 16 high-speed bit-times of the continuous Chirp K that makes up the upstream chirp. It may be reset by any shorter interruption.

If the filtering of the upstream chirp takes too much time, the downstream facing port may not be able to finish its downstream chirp in time to be able to end the reset procedure in time. Therefore, when timer T0 reaches beyond the time T_{UCHEND} (time to detect an upstream chirp), the hub is put in a wait state, which it leaves after the timer has timed out the complete reset protocol (T_{DRST}). It will then enter the full-speed enabled state.

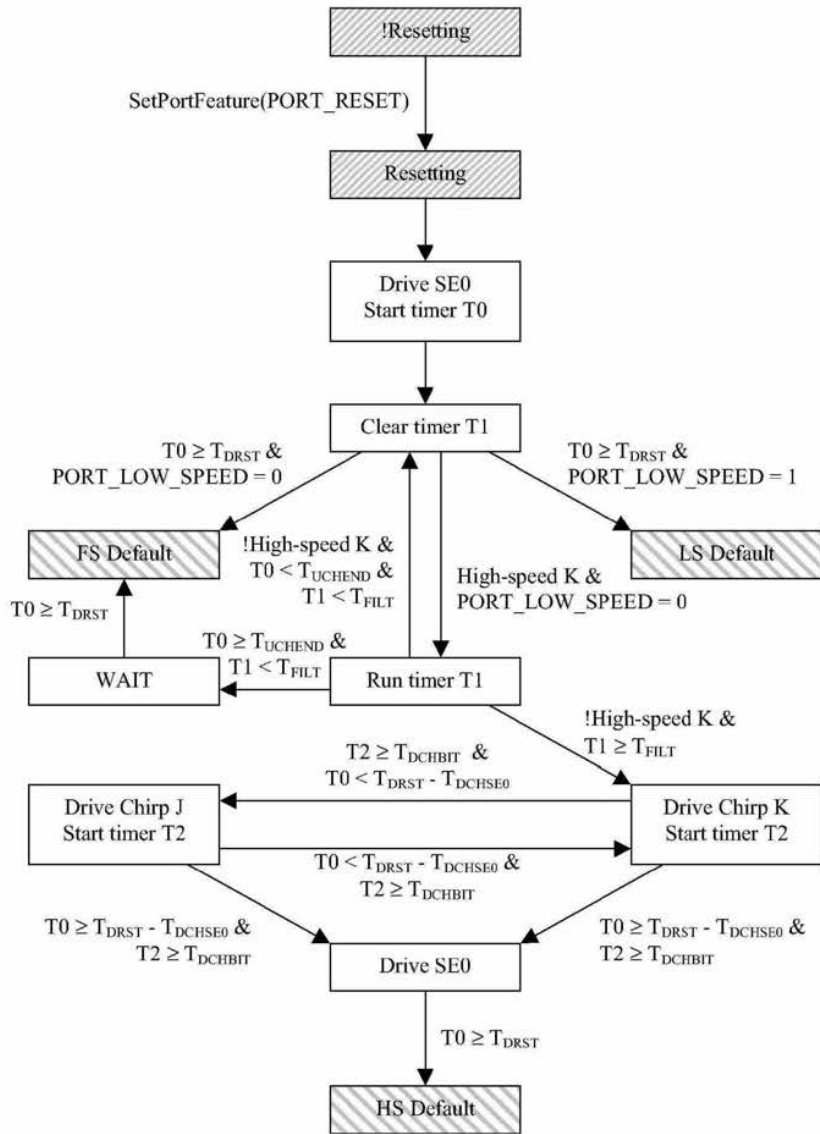


Figure C-1. Downstream Facing Port Reset Protocol State Diagram

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.

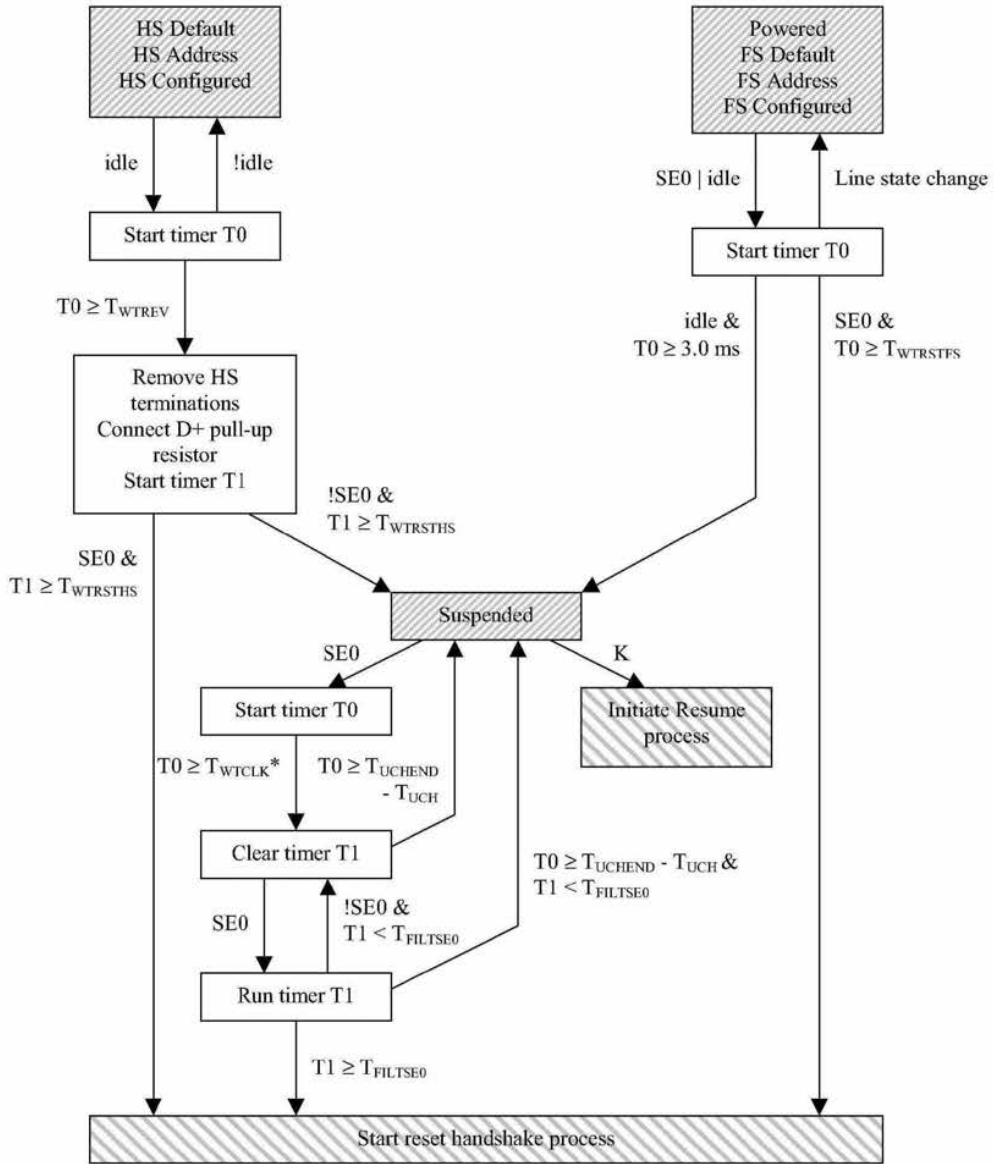
C.2.1 Reset From Suspended State

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 T_1 " and "Run timer T_1 " 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

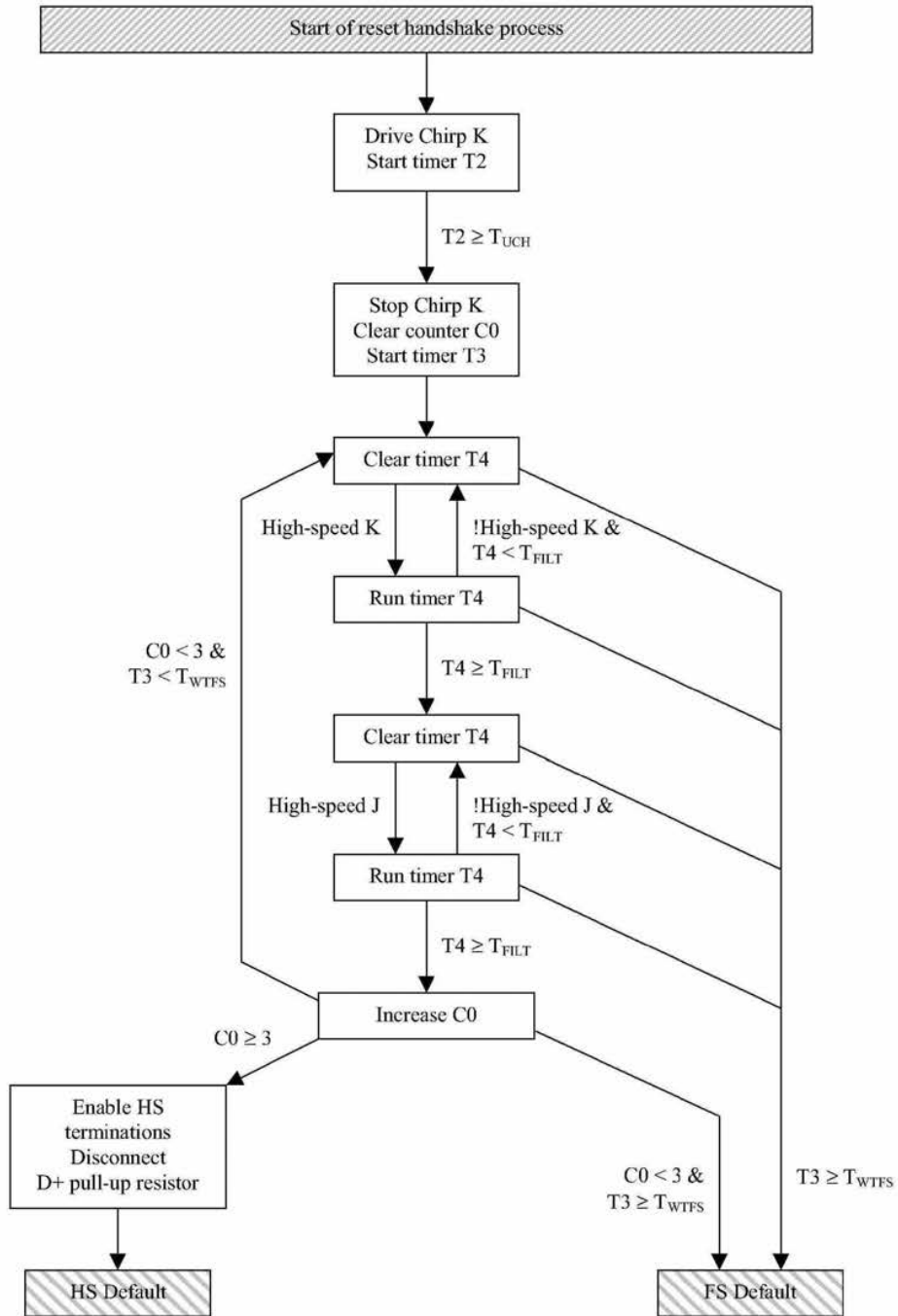


Figure C-3. Upstream Facing Port Reset Handshake State Diagram

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_{WTREV} (between 3.0 ms minimally and 3.125 ms maximally), the device reverts to full-speed by switching off its high-speed terminations and connecting the D+ pull-up resistor to the D+ line.

The reset protocol allows some time for debouncing and settling of the lines in the new state ($T_{WTRSTHS}$). After this time, the line should be sampled to see whether the device should be suspended (on detecting a full-speed idle) or reset (on detecting SE0).

If an idle was detected, the device should suspend; if an SE0 was detected, the device can start the reset handshake.

If something other than an idle or an SE0, in other words, a K, was detected, the device will also enter the suspended state. However, on seeing the K, the device will immediately resume, effectively returning to the high-speed state.

C.2.4 Reset Handshake

At this point, the behavior of devices has become independent of the initial state they were in when the reset started. The reset handshake is started by the device, when it sends an upstream chirp that is at least 1.0 ms long and stops before the timer T0 hits the 7.0 ms mark. Note: This is the same timer T0 that was started in the reset detection state diagram in Figure C-2.

A choice of implementation is available here. The one presented in the state diagram in Figure C-3 is where a timer T2 is started when the Chirp K is asserted to time the minimum required duration of the upstream chirp. The Chirp K is stopped when timer T2 exceeds the value of T_{UCH} . Another approach would be to wait until the timer T0 exceeds the value of T_{UCHEND} , before ending the upstream chirp. Both conform to the requirements of the reset protocol in Section 7.1.7.5, and the choice may depend on the particular application.

As soon as the upstream chirp has ended, the device starts listening for the downstream chirp. In order to detect at least a K-J-K-J-K-J pattern, it first starts looking for a continuously asserted Chirp K. The method employed in this state diagram is counting the number of K-J transitions. Here K and J are actually Chirp K and Chirp J, respectively, asserted continuously for at least 2.5 μ s (T_{FIL}).

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 DPLLs, 7.1.15.1
- A**
- abnormal termination sequences, 11.3.3
- aborting/retiring transfers
 - aborting control transfers, 5.5.5
 - after loss of synchronization, 11.22.2
 - client role in, 10.5.2.2
 - conditions for, 5.3.2
 - message pipes and, 5.3.2.2
 - packet size and, 5.5.3
 - Transaction Translator's role, 11.18.6, 11.18.6.1
 - 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