EHCI Compliance Test Specification
Enhanced Host Controller Interface Specification for Universal Serial Bus

Date: July 2, 2003

Revision: 1.1

Scope of this Revision

The 1.1 revision of the specification is intended to describe the testing to be applied to the Enhanced Host Controller Interface Specification for USB, revision 1.0.

Revision History

Revision	Issue Date	Comments
0.9	1/21/01	First cut that includes most everything. Needs focused review.
0.91	2/26/01	Folded in review comments from first round of review.
1.0	4/24/01	Final revision, targeted to 0.95 revision of the EHCI Specification for USB.
1.1	7/2/03	Added Assertions and Test descriptions targeted for 1.0 revision of the EHCI specification.

Significant Contributors:

John S. Howard	Intel Corporation	Nobuo Furuya	NEC
Abdul Rahman Ismail	Intel Corporation	Bart Vertenten	Philips
Gloria Villasenor	Intel Corporation		

Please send comments via electronic mail to: ehcisupport@intel.com

Table of Contents

1. Introduction

This document provides the compliance criteria and test descriptions for USB2.0 host controllers that conform to the Enhanced Host Controller Interface (EHCI) Specification, Revision 0.95. It is relevant for anyone building an EHCI host controller. These criteria address the functional requirements of an EHCI host controller. The electrical compliance requirements for host controllers are defined by the USB-IF. The document is divided into two major areas. The first is compliance criteria and the second is test descriptions.

Compliance criteria are provided as a list of assertions that describe specific characteristics or behaviors that must be met. Each assertion provides a reference to the specification or other document from where the assertion was derived. In addition, each assertion provides a reference to the specific test description(s) where the assertion is tested. For the reader's convenience, a cross-reference between the test assertions and test descriptions is provided in Appendix A.

Test descriptions provide a high level overview of the tests that are performed to check the compliance criteria. The descriptions are provided with enough detail so that a reader can understand what a test does. The descriptions do not describe the actual step-by-step procedure to perform the test.

2. Test Assertions

2.1 Host Controller Registers

Section 2.

Num	Assertion
2.1.1	The host controller must return zeros for all reserved fields in all defined memory-mapped and configuration space registers when read. **Specification Ref:** 3 (assumed from interpretation of Reserved) **Test Description:** TD.1.4

2.1.1 PCI Configuration Registers

PCI Configuration Registers (USB) - Section 2.1

Num	Assertion
2.1.2	Power Management – The host controller must implement PCI Power Management as per the PCI Bus Power Management Spec Rev 1.1 Class code – The host controller must return 0x0C0320 SBRN – The host controller must return the serial bus spec. release number **Specification Ref:** 2.1.1, 2.1.2, 2.1.4 **Test Description:** TD.1.4, TD.10.1, TD.10.2, TD.10.3, TD.10.4, TD.10.5

Frame Length Adjustment Register - Section 2.1.5

Num	Assertion
2.1.3	Incrementing the FLADJ register by one should increase the SOF cycle time by 16 high speed bit times. NOTE : Changes to FLADJ should be done only when the host controller is halted (i.e. when the halted bit is set in the USBSTS register). **Specification Ref:** 2.1.5 **Test Description:** TD.1.9

Port Wake Capability Register - Section 2.1.6

Num	Assertion
2.1.4	If the low-order bit (bit zero) is a one, then bits 15:1 must be writable. **Specification Ref:** 2.1.6 **Test Description:** TD.1.11

2.1.2 Structural Host Controller Parameters

Mapping Structural Parameters to Implemented Features - Section 2.2.3

Num	Assertion
2.1.5	PORTSC field **Port Indicator Control** must be functional (with externally available indicators per port) when HCSPARAMS field **Port Indicators** is a one. **Specification Ref:** 2.2.3 **Test Description:** TD.2.1
2.1.6	A Debug Port must be present and functional if the HCSPARAMS field **Debug Port Number** is nonzero and the value of **Debug Port Number** be less than or equal to the value reported in the field **N_PORTS** **Specification Ref:** 2.2.3 **Test Description:** Not-Tested
2.1.7	The host controller implementation must have the same number of companion host controllers as reported in the HCSPARAMS field **N_CC**. **Specification Ref:** 2.2.3, 4.2 **Test Description:** TD.1.6
2.1.8	If HCSPARAMS field **Port Routing Rules** is a zero, then the implementation must map the first **N_PCC** ports to the lowest numbered function companion controller, the next **N_PCC** ports to the next lowest number function controller, etc. **Specification Ref:** 2.2.3, 4.2 **Test Description:** TD.1.6
2.1.9	If HCSPARAMS field **Port Routing Rules** is a one, then the implementation must map the ports to companion controllers as specified in the HCSP-PORTROUTE register. **Specification Ref:** 2.2.3, 4.2 **Test Description:** TD.1.6
2.1.10	PORTSC field **Port Power** must be functional (with externally controllable power per port) when HCSPARAMS field **Port Power Control** is a one. **Specification Ref:** 2.2.3**,** **Test Description:** TD.2.6

Mapping Structural Parameters to Implemented Features - Section 2.2.3

Num Assertion

Host Controller Capability Parameters - Section 2.2.4

Note: the Isochronous Scheduling Threshold assertions are in Section 2.3.

2.1.11 USBCMD field **Frame List Size** must be modifiable when HCCPARAMS field **Programmable Frame List Flag** is a one.

Specification Ref: 2.3.1

Test Description: TD.1.8

2.1.12 If the HCCPARAMS field **64-bit Addressing Capability** is a zero, then the host controller must support the 32-bit versions of the host controller data structures (as defined in Section 3 of the EHCI Specification).

Specification Ref: 2.2.4, 3

Test Description: TD.1.10

2.1.13 If the HCCPARAMS field **64-bit Addressing Capability** is a one, then the host controller must support only the 64-bit versions of the host controller data structures (as defined in Appendix B of the EHCI Specification).

Specification Ref: 2.2.4, Appendix B

Test Description: TD.1.2

2.1.3 USBCMD — Section 2.3.1

Ensure that the Op-registers are at Cap-Register offset + Cap-Register LENGTH

USBCMD Register – Section 2.3.1

Num Assertion

2.1.14 After a host controller reset, via the USBCMD register bit *HCReset*, all operational registers and fields must be at their default values.

Specification Ref: 2.3.1

Test Description: TD.1.4

2.1.15 The USBCMD register must be located at the DWORD-aligned offset calculated by:

Memory-Mapped I/O Base Address + CAPLENGTH.

Specification Ref: 2.3.1

Test Description: TD.1.4

2.1.16 The *Interrupt Threshold Control* field in the USBCMD register must be updateable with the values listed in the EHCI specification.

Specification Ref: 2.3.1

Test Description: TD.3.8

2.1.17 The host controller must not execute from the Asynchronous Schedule unless the USBCMD register field *Asynchronous Schedule Enable* is a one and the USBSTS register field *Asynchronous Schedule Status* is a one.

Specification Ref: 2.3.1, 4.8

Test Description: TD.3.8

2.1.18 The host controller must not execute from the Periodic Schedule unless the USBCMD register field *Periodic Schedule Enable* is a one and the USBSTS register field *Periodic Schedule Status* is a one.

Specification Ref: 2.3.1, 4.6

Test Description: TD.3.8

USBCMD Register – Section 2.3.1

Num	Assertion
2.1.19	The host controller must properly service the default size of periodic frame list as encoded in the USBCMD register field *Frame List Size*. **Specification Ref:** 2.3.1 **Test Description:** TD.1.8
2.1.20	In the event USBCMD register field *Frame List Size* is adjustable, then the host controller must properly service every encoding size of frame list. **Specification Ref:** 2.3.1, 4.6 **Test Description:** TD.1.8
2.1.21	The host controller must not drive a reset condition on its downstream ports while the USBCMD register *HCRESET* field is a one. **Specification Ref:** 2.3.1 **Test Description:** TD.1.5
2.1.22	The host controller must not change its PCI configuration register settings when the host controller is reset via the USBCMD register *HCRESET* field. The host controller should reset the RESET bit on completion of reset. **Specification Ref:** 2 **Test Description:** TD.1.4
2.1.23	The host controller will propagate SOFs down enabled ports when the USBCMD register *Run/Stop* field is a one and the USBSTS register *HCHalted* field is a zero. **Specification Ref:** 4.13 **Test Description:** TD.2.7
2.1.24	The host controller can execute transactions from enabled schedules (Asynchronous and Periodic) only when USBCMD register *Run/Stop* field is a one and the USBSTS register *HCHalted* field is a zero. **Specification Ref:** 2.3.1 **Test Description:** TD.1.8

2.1.4 USBSTS — Section 2.3.2

USBSTS Register – Section 2.3.2

Num	Assertion
2.1.25	The host controller must set USBSTS register field *HCHalted* bit to a one within 16 micro-frames after system software sets USBCMD register field *Run/Stop* to a zero. **Specification Ref:** 2.3.2, **Test Description:** TD.1.5
2.1.26	The host controller must set the USBSTS register *Frame List Rollover* field to a one when the FRAME LIST INDEX register field rolls from 0x1FFF to zero when the USBCMD register *Frame List Size* field value is 0x0. **Specification Ref:** 2.3.2 **Test Description:** TD.1.8
2.1.27	The host controller must set the USBSTS register *Frame List Rollover* field to a one when the FRAME LIST INDEX register field rolls from 0x0FFF to zero when the USBCMD register *Frame List Size* field value is 0x1. **Specification Ref:** 2.3.2 **Test Description:** TD.1.8

USBSTS Register – Section 2.3.2

Num	Assertion
2.1.28	The host controller must set the USBSTS register *Frame List Rollover* field to a one when the FRAME LIST INDEX register field rolls from 0x07FF to zero when the USBCMD register *Frame List Size* field value is 0x2. **Specification Ref:** 2.3.2 **Test Description:** TD.1.8
2.1.29	The USBSTS register field *Frame List Rollover* field may only be cleared by system software by writing a one to this field. **Specification Ref:** 2.3.2 **Test Description:** TD.1.8
2.1.30	The host controller must set the USBSTS register *Port Change Detect* field to a one when any PORTSC register's *Connect Status Change* bit transitions from a zero to a one. **Specification Ref:** 2.3.2 **Test Description:** TD.2.6, TD.2.2
2.1.31	The host controller must set the USBSTS register *Port Change Detect* field to a one when any PORTSC register's *Port Enabled/Disable Change* bit transitions from a zero to a one. **Specification Ref:** 2.3.2 **Test Description:** TD.2.2
2.1.32	The host controller must set the USBSTS register *Port Change Detect* field to a one when any PORTSC register's *Over-current Change* bit transitions from a zero to a one. **Specification Ref:** 2.3.2 **Test Description:** Not-Tested
2.1.33	The host controller must set the USBSTS register *Port Change Detect* field to a one when any PORTSC register's *Force Port Resume* bit transitions from a zero to a one as a result of a J-K transition detected on a suspended port. A suspended port is any PORTSC register whose **Suspend** and **Enabled** bits is a one. **Specification Ref:** 2.3.2 **Test Description:** TD.2.2
2.1.34	The host controller must set the USBSTS register **USBERRINT** field to a one if the result of a bus transaction causes the **CERR** field in the associated queue TD to decrement to a zero. **Specification Ref:** 2.3.2, 4.15.1.1, 4.15.1.1.1 **Test Description:** TD.3.9
2.1.35	The host controller must set the USBSTS register **USBERRINT** field to a one if the host controller detected a CRC error during a DATA IN associated with the execution of an iTD or siTD. **Specification Ref:** 2.3.2, 4.15.1.1, 4.15.1.1.1 **Test Description:** TD.3.9
2.1.36	The host controller must set the USBSTS register **USBERRINT** field to a one if the host controller detected a TIMEOUT error during a transaction associated with the execution of an iTD or siTD. **Specification Ref:** 2.3.2, 4.15.1.1, 4.15.1.1.1 **Test Description:** TD.3.9

USBSTS Register – Section 2.3.2

Num	Assertion
2.1.37	The host controller must set the USBSTS register **USBERRINT** field to a one if the host controller detected a BAD PID error during the reception of a response packet associated with the execution of an iTD or siTD. **Specification Ref:** 2.3.2, 4.15.1.1, 4.15.1.1.1 **Test Description:** TD.3.9
2.1.38	The host controller must set the USBSTS register **USBERRINT** field to a one if the host controller detects packet babble during the reception of a data response from the device. **Specification Ref:** 2.3.2, 4.15.1.1, 4.15.1.1.1 **Test Description:** TD.3.9

2.1.5 USBINTR — Section 2.3.3

USBINTR Register – Section 2.3.3

Num	Assertion
2.1.39	The host controller must set the USBSTS register USBINTR field to a one when a iTD's active transaction record's *IOC* field is a one and the host has a non-error transaction completion that causes the transaction record's *Active* bit transition to a zero. **Specification Ref:** 2.3.3, 3.3.2, 4.15.1.2 **Test Description:** TD.3.8
2.1.40	The host controller must set the USBSTS register USBINTR field to a one when an siTD's *IOC* field is a one and the host has a non-error transaction completion that causes the siTD's *Active* bit transition to a zero. **Specification Ref:** 2.3.3, 3.4.3, 4.15.1.2 **Test Description:** TD.3.8
2.1.41	The host controller must set the USBSTS register USBINTR field to a one when a Queue Head's *IOC* field is a one and the host has a non-error transaction completion that causes the Queue Head's *Active* bit transition to a zero. **Specification Ref:** 2.3.3, 3.5.3, 4.15.1.2 **Test Description:** TD.3.8
2.1.42	The host controller must set the USBSTS register USBINTR field to a one when a QueueTD's *IOC* field is a one and the host has a non-error transaction completion on an IN Transaction where the data received in the transaction was less than expected (e.g. short packet). **Specification Ref:** 2.3.3, 3.5.3, 4.15.1.2, 4.15.1.3 **Test Description:** TD.3.8
2.1.43	The host controller must assert an interrupt if and only if the USBINTR register *Interrupt on Async Advance Enable* bit is a one AND USBSTS register *Interrupt on Async Advance* bit is a one. **Specification Ref:** 2.3.3 **Test Description:** TD.3.5

USBINTR Register – Section 2.3.3

Num	Assertion
2.1.44	The host controller must assert an interrupt if and only if the USBINTR register *Host System Error Enable* bit is a one AND USBSTS register *Host System Error* bit is a one. **Specification Ref:** 2.3.3 **Test Description:** Not-Tested
2.1.45	The host controller must assert an interrupt if and only if the USBINTR register *Frame List Rollover Enable* bit is a one ANDUSBSTS register *Frame List Rollover* bit is a one. **Specification Ref:** 2.3.3 **Test Description:** TD.1.8
2.1.46	The host controller must assert an interrupt if and only if the USBINTR register *Port Change Interrupt Enable* bit is a one AND USBSTS register *Port Change Detect* bit is a one. **Specification Ref:** 2.3.3 **Test Description:** TD.2.2
2.1.47	The host controller must assert an interrupt if and only if the USBINTR register *USB Error Interrupt Enable* bit is a one AND USBSTS register *USB Error Interrupt* bit is a one. **Specification Ref:** 2.3.3 **Test Description:** TD.3.9
2.1.48	The host controller must assert an interrupt if and only if the USBINTR register *USB Interrupt Enable* bit is a one AND USBSTS register *USB Interrupt* bit is a one. **Specification Ref:** 2.3.3 **Test Description:** TD.3.8

2.1.6 FRINDEX — Section 2.3.4

FRINDEX Register – Section 2.3.4

Num	Assertion
2.1.49	The SOF value on the USB must ONLY change whenever the FRINDEX value bits [2:0] increment from a zero to a one. **Specification Ref:** 2.3.4, 4.5 **Test Description:** TD.3.1
2.1.50	Both software and the host controller use the FRINDEX register for processing the Periodic Schedule. The host controller must not increment and update FRINDEX until the hardware has at least completed processing from the Periodic Schedule. **Specification Ref:** 2.3.4, 4.5 **Test Description:** TD.3.1, TD.3.23

2.1.7 CTRLDSEGMENT — Section 2.3.5

CTRLDSSEGMENT Register – Section 2.3.5

Num	Assertion
2.1.51	When HCCPARAMS register **64-bit Addressing Capability** field is a zero, then CTRLDSSEGMENT register must be read only and return zero when read. **Specification Ref:** 2.3.5 **Test Description:** TD.1.4
2.1.52	When HCCPARAMS register **64-bit Addressing Capability** field is a one, then CTRLDSSEGMENT register must be read/write by system software. **Specification Ref:** 2.3.5 **Test Description:** TD.1.4
2.1.53	When HCCPARAMS register **64-bit Addressing Capability** field is a one, then the host controller must use the value of the CTRLDSSEGMENT register to generate schedule data structure addresses. **Specification Ref:** 2.3.5 **Test Description:** TD.1.2

2.1.8 Config Flag — Section 2.3.8

Config Flag Register

Num	Assertion
2.1.54	CONFIG FLAG Register When the CONFIGFLAG register has a value of zero, the high-speed host controller's PORTSC registers **Port Owner** fields are a one and the high-speed controller ports will not detect port connect events. **Specification Ref: 2.3.8** **Test Description:** TD.1.3
2.1.55	When the CONFIGFLAG register has a value of one, the high-speed host controller's PORTSC registers **Port Owner** fields are a zero and the high-speed controller ports will control the ports. **Specification Ref: 2.3.8** **Test Description:** TD.1.3

2.1.9 PORTSC — Section 2.3.9

Note: all assertions must be applied to all implemented PORTSC registers. No of PORTSC registers must be equal to N_PORTS

Port Status and Control Registers (PORTSC) – Section 2.3.9

Num	Assertion
2.1.56	Number of implemented PORTSC registers must be equal to the value reported in HCSPARAMS register **N_PORTS**. **Specification Ref:** 2.3.9 **Test Description:** TD.1.4

Port Status and Control Registers (PORTSC) – Section 2.3.9

Num **Assertion**

2.1.57 All functionality on a PORTSC register is disabled unless the **Port Power** bit is a one.

Specification Ref: 2.3.9

Test Description: TD.1.6

2.1.58 When software transitions *Port Power* to a one (from a zero), all other PORTSC bits are set to their default value. The notable exception to this is the *Line Status* field, which is defined to be only valid when *Port Connect* is a one and *Port Enabled* is a zero.

Specification Ref: 2.3.9

Test Description: TD.1.4

2.1.59 When an over-current event occurs, the host controller must set the following fields:

Field	Value
PORTSC *Over-current Change*	1
PORTSC *Over-current Active*	1
USBSTS *Port Change Detect*	1

Specification Ref: 2.3.9

Test Description: Not-Tested

2.1.60 Assert PME#, (if enabled) when an over-current event occurs and the PORTSC register *WKOC_E* field is a one.

Specification Ref: 2.3.9, 4.2.5, 4.3.1

Test Description: Not-Tested

2.1.61 Do NOT assert PME# (if enabled) when an over-current event occurs and the host controller device is not in D0 and the PORTSC register *WKOC_E* field is a zero.

Specification Ref: 2.3.9, 4.2.5, 4.3.1

Test Description: Not-Tested

2.1.62 Assert a hardware interrupt when an over-current event occurs and the device is in D0 and the USBINTR register *Port Change Interrupt Enable* field is a one. Set USBSTS Register *Port Change Detect*.

Specification Ref: 2.3.9, 4.2.5, 4.3.1

Test Description: Not-Tested

2.1.63 Do NOT assert a hardware interrupt when an over-current event occurs and the host controller device is NOT in D0.

Specification Ref: 2.3.9, 4.2.5, 4.3.1

Test Description: Not-Tested

2.1.64 When a disconnect event occurs, the host controller must set the following fields:

Field	Value
PORTSC **Current Connect Status**	0
PORTSC **Connect Status Change**	1
PORTSC **Port Enabled/Disabled**	0
USBSTS **Port Change Detect**	1

Specification Ref: 2.3.9, 4.2.2

Test Description: TD.2.2

2.1.65 Assert PME# (and set PME_Status in PMCSR), (if enabled and device not in D0) when a disconnect event occurs and the PORTSC register *WKDSCNNT_E* field is a one.

Specification Ref: 2.3.9, 4.3.1

Test Description: TD.10.3

Port Status and Control Registers (PORTSC) – Section 2.3.9

Num **Assertion**

2.1.66 Do NOT assert PME# (or set PME_Status in PMCSR) (if enabled and device not in D0) when a disconnect event occurs and the host controller device is not in D0 and the PORTSC register *WKDSCNNT_E* field is a zero.

Specification Ref: 2.3.9, 4.3.1

Test Description: TD.10.2

2.1.67 Assert a hardware interrupt when a disconnect event occurs and the host controller device is in D0 and the USBINTR register *Port Change Interrupt Enable* field is a one. Set USBSTS Register bit *Port Change Detect*.

Specification Ref: 2.3.9

Test Description: TD.2.2

2.1.68 Do NOT assert a hardware interrupt when a disconnect event occurs and the host controller device is NOT in D0.

Specification Ref: 2.3.9, 4.3.1

Test Description: Not-Tested

2.1.69 When a connect event occurs, the host controller must set the following fields:

Field	**Value**
PORTSC *Current Connect Status*	1
PORTSC *Connect Status Change*	1
USBSTS *Port Change Detect*	1

Specification Ref: 2.3.9

Test Description: TD.2.2

2.1.70 Assert PME# (and and set PME_Status in PMCSR), (if enabled and device not in D0) when a connect event occurs and the PORTSC register *WKCNNT_E* field is a one.

Specification Ref: 2.3.9, 4.3,1

Test Description: TD.10.5

2.1.71 Do NOT assert PME# (if enabled and device not in D0) when a connect event occurs and the host controller device is not in D0 and the PORTSC register *WKCNNT_E* field is a zero.

Specification Ref: 2.3.9, 4.3.1

Test Description: TD.10.4

2.1.72 Assert a hardware interrupt when a connect event occurs and the host controller device is in D0 and the USBINTR register *Port Change Interrupt Enable* field is a one. Set USBSTS Register *Port Change Detect*.

Specification Ref: 2.3.9, 4.3.1

Test Description: TD.2.2

2.1.73 Do NOT assert a hardware interrupt when a connect event occurs and the host controller device is NOT in D0.

Specification Ref: 2.3.9, 4.3.1

Test Description: Not-Tested

2.1.74 The PORTSC register is not in a test mode (e.g. is usable as a normal port) when the Port Test Control field value is a zero.

Specification Ref: 2.3.9

Test Description: TD.2.2

Port Status and Control Registers (PORTSC) – Section 2.3.9

Num	Assertion
2.1.75	Writing a value of **Test J_State** to a PORTSC *Port Test Control* field causes the port to go into the **Test J_State** test mode. **Specification Ref:** 2.3.9, 4.14 **Test Description:** TD.2.3
2.1.76	Writing a value of **Test K_State** to a PORTSC *Port Test Control* field causes the port to go into the **Test K_State** test mode. **Specification Ref:** 2.3.9, 4.14 **Test Description:** TD.2.3
2.1.77	Writing a value of **Test SE0_NAK** to a PORTSC *Port Test Control* field causes the port to go into the **Test SE0_NAK** test mode. **Specification Ref:** 2.3.9, 4.14 **Test Description:** TD.2.3
2.1.78	Writing a value of *Test Packet* to a PORTSC **Port Test Control** field causes the port to go into the *Test Packet* test mode. **Specification Ref:** 2.3.9, 4.14 **Test Description:** TD.2.3
2.1.79	Writing a value of *Test FORCE_ENABLE* to a PORTSC **Port Test Control** field causes the port to go into the *Test FORCE_ENABLE* test mode. **Specification Ref:** 2.3.9, 4.14 **Test Description:** TD.2.3
2.1.80	If HCSPARAMS *P_INDICATOR* bit is a one, then writing a value of 00b to PORTSC *Port Indicator Control* fields will turn the port indicator off. **Specification Ref:** 2.3.9 **Test Description:** TD.2.1
2.1.81	If HCSPARAMS *P_INDICATOR* bit is a one, then writing a value of 01b to PORTSC *Port Indicator Contro***l** fields will turn the port indicator to Amber. **Specification Ref:** 2.3.9 **Test Description:** TD.2.1
2.1.82	If HCSPARAMS *P_INDICATOR* bit is a one, then writing a value of 10b to PORTSC *Port Indicator Control* fields will turn the port indicator to Green. **Specification Ref:** 2.3.9 **Test Description:** TD.2.1
2.1.83	When the *Port Owner* bit is a zero, the PORTSC will control and report status on the external port. **Specification Ref:** 2.3.9, 4.2.3. **Test Description:** TD.2.2
2.1.84	When the *Port Owner* bit is a one, the PORTSC will neither report status nor allow control of the external port. **Specification Ref:** 2.3.9, 4.2.1 **Test Description:** TD.1.3
2.1.85	When the *Port Enabled/Disabled* bit is a zero, a low-speed attached device will be reported in the *Line Status* field as a 01b. **Specification Ref:** 2.3.9 **Test Description:** TD.2.6

Port Status and Control Registers (PORTSC) – Section 2.3.9

Num **Assertion**

2.1.86 When the *Port Enable/Disabled* bit is a zero, a full-speed attached device will be reported in the *Line Status* field as a 10b.

Specification Ref: 2.3.9

Test Description: TD.2.6

2.1.87 Writing a one to *Port Reset* bit will initiate a high-speed (chirp) port reset if the USBCMD register *Run/Stop* bit is a one and USBSTS register *HCHalted* bit is a zero.

Specification Ref: 2.3.9

Test Description: TD.2.6

2.1.88 The host controller must set *Port Reset* bit to a zero within 2 milliseconds of software writing *Port Reset* to a zero (from a one).

Specification Ref: 2.3.9

Test Description: TD.2.6

2.1.89 If the chirp succeeded during a port reset, when the *Port Reset* bit is transitioned to a zero (signaling end of reset) the *Port Enabled/Disabled* bit will transition to a one (by hardware). An enabled port will have the following bits set to a one:

Field:	**Value**
Current Connect Status	1
Port Enabled/Disabled	1
Port Power	1

Specification Ref: 2.3.9, 4.2.2

Test Description: TD.2.6

2.1.90 If the chirp did not succeed during a port reset, when the *Port Reset* bit is transitioned to a zero (signaling end of reset) the *Port Enabled/Disabled* bit will remain at a zero, and the *Line Status* field must correctly reflect the state of the bus data lines.

Specification Ref: 2.3.9, 4.2.2

Test Description: TD.2.6

2.1.91 The host controller will not propagate transactions down a port if the *Suspend* bit is a one and the *Port Enabled/Disabled* bit is a one and the *Port Connected* bit is a one.

Specification Ref: 2.3.9

Test Description: TD.2.8, TD.2.9

2.1.92 The host controller will ignore a write of a zero to the *Suspend bit*.

Specification Ref: 2.3.9

Test Description: TD.2.8

2.1.93 The host controller will transition *Suspend* bit from a one to a zero when software sets the *Port Reset* bit to a one (from a zero).

Specification Ref: 2.3.9

Test Description: TD.2.10

2.1.94 The host controller will transition *Suspend* bit from a one to a zero when software sets the *Force Port Resume* bit to a zero (from a one).

Specification Ref: 2.3.9, 4.3.1

Test Description: TD.2.8, TD.2.9

Port Status and Control Registers (PORTSC) – Section 2.3.9

Num	Assertion
2.1.95	When software sets the *Force Port Resume* bit to a one on a suspended port (*Suspend* = 1, *Port Enabled* = 1, *Port Connected* = 1), then the host controller will drive a *K_State* on the port. **Specification Ref:** 2.3.1, 4.3.1 **Test Description:** TD.2.8
2.1.96	When software sets the *Force Port Resume* bit to a one (on a suspended port), the host controller must NOT set the USBSTS register *Port Change Detect* bit to a one. **Specification Ref:** 2.3.9, 4.3.1 **Test Description:** TD.2.8
2.1.97	The host controller will set the *Force Port Resume* bit to a one, set the USBSTS register *Port Change Detect* bit to a one, and drive a *K_State* on the port when it detects resume signaling on the port. **Specification Ref:** 2.3.9, 4.3.1 **Test Description:** TD.2.9, TD.10.1
2.1.98	When software transitions the *Force Port Resume* bit to a zero (from a one), the host controller will complete the following within 2 milliseconds: • Stop driving a *K_State* • Put port into high-speed mode (force the port into high-speed idle) • After the port is in high-speed mode, reflect a zero in *Force Port Resume*. **Specification Ref:** 2.3.9, 4.3.1 **Test Description:** TD.2.8, TD.2.9
2.1.99	The host controller will set the *Over-current Change* bit to a one whenever it detects the *Over-current Active* bit change its value (0b to 1b or 1b to 0b). This transition will also set the USBSTS register *Port Change Detect* bit to a one. **Specification Ref:** 2.3.9, 4.2.5 **Test Description:** Not-Tested
2.1.100	The host controller will transition the *Over-current Change* bit from a one to a zero only when software writes a one to this bit position. **Specification Ref:** 2.3.9 **Test Description:** Not-Tested
2.1.101	The host controller will set the *Port Enable/Disable Change* bit to a one when a Frame babble condition is detected. This transition will also set the USBSTS register *Port Change Detect* bit to a one. **Specification Ref:** 2.3.9, 4.14.1.1.1, and Universal Serial Bus Specification, Rev. 2.0 (11.2.5) **Test Description:** TD.3.2
2.1.102	The host controller will transition the *Port Enable/Disable Change* bit from a one to a zero when software writes a one to this bit position. **Specification Ref:** 2.3.9 **Test Description:** TD.2.6
2.1.103	The host controller will not set the *Port Enable/Disable* bit by Software simply writing a one to the bit. **Specification Ref:** 2.3.9 **Test Description:** TD.2.4

Port Status and Control Registers (PORTSC) – Section 2.3.9

Num	Assertion
2.1.104	The host controller will set the *Port Enable/Disable* bit to a zero when software writes a zero to this bit. When this occurs, the host controller will NOT set the *Port Enable/Disable Change* bit to a one. **Specification Ref:** 2.3.9 **Test Description:** TD.2.5
2.1.105	The host controller will transition the *Port Enable/Disable* bit from a one to a zero on any of the following events: • *Connect Status* transitions from a one to a zero, • The host controller detects a Frame Babble on this port, • Software writes a zero to this bit position, • The *Over-current* bit transitions from a zero to a one **Specification Ref:** 2.3.9 **Test Description:** TD.2.6, TD.3.2
2.1.106	The host controller will set the *Connect Status Change* bit to a one when the *Current Connect Status* bit transitions from a one to a zero. This transition will also set the USBSTS register *Port Change Detect* bit to a one. **Specification Ref:** 2.3.2, 2.3.9, 4.2.2 **Test Description:** TD.2.2
2.1.107	The host controller will transition the *Connect Status Change* bit from a one to a zero only when software writes a one to this bit position. **Specification Ref:** 2.3.9 **Test Description:** TD.2.2
2.1.108	The host controller will set the *Current Connect Status* bit to a one when a device is connected to the port (connect is defined in the USB Core Specification). **Specification Ref:** 2.3.9, 4.2.2 **Test Description:** TD.2.2
2.1.109	When system software transitions the *Port Owner* bit from a zero to a one, (e.g. giving up port ownership to a companion controller), then the following events must occur: • *Connect Status* will transition from a one to a zero, and • *Connect Status Change* will transition from a zero to a one, and • USBSTS register *Port Change Detect* will transition from a zero to a one, and • If USBINTR register *Port Change Interrupt Enable* is a one, then the host controller will issue a hardware interrupt. **Specification Ref:** 2.3.9, 4.2.2 **Test Description:** TD.2.6

2.2 Data Structures

All schedule data structures contain at least one schedule link pointer. The Periodic Frame list has more than one. These pointers are used by the host controller to traverse the schedule graph constructed by system software. Before using a schedule link pointer, the host controller must evaluate the low-order three bits to determine what to do next. The following set of assertions apply to all schedule link pointers in this context. These assertions do NOT apply to qTD link pointers in Qheads.

When a schedule link pointer's *T-bit* is a zero, the *Typ* field determines how the host controller will process the next data structure, once it is read by the host controller.

Schedule Link Pointers – Section 3 (all data structures)

Num	Assertion
2.2.1	When processing the Periodic Schedule, the host controller will transition to processing the Asynchronous Schedule (when enabled in the USBCMD register) if it it encounters a *T-bit* value of 1 in a link pointer of the current data structure (siTD, iTD, or Qhead). **Specification Ref:** 4.4 **Test Description:** TD.4.1
2.2.2	When processing the Periodic Schedule, the host controller will stop issuing transactions for the remainder of a micro-frame when it encounters a *T-bit* value of 1 in a link pointer of the current data structure (siTD, iTD, or QH), and the Asynchronous Schedule is not enabled. **Specification Ref:** 4.4 **Test Description:** TD.4.1
2.2.3	If a schedule link pointer's *Typ* field has a value of 00b, then the host controller will process the referenced data structure as an Isochronous Transfer Descriptor (iTD). **Specification Ref:** 3.1 **Test Description:** TD.3.12
2.2.4	If a schedule link pointer's *Typ* field has a value of 01b, then the host controller will process the referenced data structure as a Queue Head (Qhead). **Specification Ref:** 3.1 **Test Description:** TD.3.10
2.2.5	If a schedule link pointer's *Typ* field has a value of 10b, then the host controller will process the reference data structure as a Split Isochronous Transfer Descriptor (siTD). **Specification Ref:** 3.1 **Test Description:** TD.3.14
2.2.6	When processing the Asynchronous Schedule, the host controller must ignore the *T-bit* fields in schedule link pointers of the Qheads in the Asynchronous List. **Specification Ref:** 3.6.1 **Test Description:** TD.3.10

2.2.1 Periodic Frame List — Section 3.1

The base of the Periodic Schedule is the **Periodic List Base** register, which references a 4K-page aligned table. Each table entry is a schedule link pointer, which serves as the base pointer for a graph of schedule data structures which direct the host controller about which transactions to conduct at this *time.* Each table entry represents 1 millisecond worth of periodic schedule.

The Isochronous Scheduling Threshold is part of management of the periodic schedule. The purpose of the Isochronous Scheduling Threshold is to provide system software with some hint as to how much of the periodic schedule the host controller caches. This allows system software to reliably update data structures in the periodic schedule without conflicting with the executing host controller.

Periodic Frame List – Section 3.1

Num	Assertion
2.2.7	If the USBCMD register field *Frame List Size* is a zero (0), then the host controller must access periodic schedule data structures assuming a 1024-element Periodic Frame List. **Specification Ref:** 2.3.1, 3.1 **Test Description:** TD.1.8

Periodic Frame List – Section 3.1

Num	Assertion
2.2.8	If the USBCMD register field *Frame List Size* is a one (1), then the host controller must access periodic schedule data structures assuming a 512-element Periodic Frame List. **Specification Ref:** 2.3.1, 3.1 **Test Description:** TD.1.8
2.2.9	If the USBCMD register field *Frame List Size* is a two (2), then the host controller must access periodic schedule data structures assuming a 256-element Periodic Frame List. **Specification Ref:** 2.3.1, 3.1 **Test Description:** TD.1.8
2.2.10	At the beginning of each micro-frame, if the Periodic Schedule is Enabled, then the host controller must begin processing from the Periodic Schedule. It will continue processing the Periodic List until a schedule pointer with a *T-bit* value of 1 is encountered. **Specification Ref:** 3.1, 4.6 **Test Description:** TD.4.1
2.2.11	At the beginning of each micro-frame, the host controller must traverse the Periodic Schedule, beginning at the appropriate offset in the Periodic Frame List. The appropriate offset is determined by the concatenation of the CTRLDSEGMENT register (if 64-bit capability) with the Periodic List Base register and bits [12:3] of FRINDEX register. This operation yields a DWORD (4 byte) aligned memory address. **Specification Ref:** 3.1, 4.6 **Test Description:** TD.3.10, TD.3.12, TD.3.14
2.2.12	When the HCCPARAMS register *Isochronous Scheduling Threshold* is a zero (no caching selection), system software can reliably update periodic data structures (which should be candidates for execution) two micro-frames ahead of the currently executing host controller. **Specification Ref:** 4.7.2.1 **Test Description:** TD.3.12
2.2.13	When the HCCPARAMS register *Isochronous Scheduling Threshold* is a 80H (frame caching selection), for any current frame *N*, if the current micro-frame is 0 to 6, then software can modify data structures which should be candidates for execution in frame *N*+1. If the current micro-frame is 7, then software can modify data structures that are candidates for execution in frame *N*+2. **Specification Ref: 4.7.2.1** **Test Description:** TD.3.12
2.2.14	The host controller indicates micro-frame caching selection with a non-zero value in 3 least significant bits of HCCPARAMS register *Isochronous Scheduling Threshold*. For any micro-frame *N*, system software can modify data structures that should be candidates for execution *X* micro-frames in front of where the host controller is currently executing. Where *X* is the least-significant three bits of the *Isochronous Scheduling Threshold* value. **Specification Ref: 4.7.2.1** **Test Description:** TD.3.12

2.2.2 Isochronous Transfer Descriptor (iTD) — Section 3.3

The *transfer state* for an iTD is the internal state the host controller keeps on board to keep track of the intermediate state of a transfer. For example, the initial condition of the transfer state is derived from the iTD read from shared memory. The host controller must keep several bits of local state for a high-bandwidth iTD, like the current value of *PG*, a count of the number of transactions executed (MULT greater than 1), etc. The following assertions don't require an implementation of internal state, but do describe the required behavior

on the bus as the host controller transitions through a high-bandwidth transfer. To aid in this discussion, the internal state of the MULT field will be a down-counter called *CurrentXCount*, that is assumed to be loaded from the iTD's MULT field when the iTD is read on chip.

Isochronous Transfer Descriptor (iTD) – Section 3.3

Num	Assertion
2.2.15	The host controller will process from one transaction record of an iTD per micro-frame. **Specification Ref:** 4.7.1 **Test Description:** TD.3.12
2.2.16	The host controller must select a transaction record in an iTD using the current value in FRINDEX[2:0]. **Specification Ref:** 4.7.1 **Test Description:** TD.3.12
2.2.17	The host controller will execute a bus transaction for a selected iTD transaction record if and only if the *Active* bit in the *Status* field of the selected transaction record is a one. **Specification Ref:** 3.3.2, 4.7.1 **Test Description:** TD.3.12, TD.3.13
2.2.18	To execute a bus transaction from an active transaction record, the following fields are used: • The host controller builds the Token packet from the iTD fields: *ENDPT*, *I/O*, and *DEVICEADDRESS* fields. If the *I/O* field indicates an IN (value of 1b), the Token packet PID will be an IN. If the *I/O* field indicates an OUT (value of 0b), then the Token packet PID will be an OUT. • The physical buffer address for where the host controller will transfer data from/to is constructed by using the *PG* field to select a starting Page pointer from the *Buffer Page Pointer List*. The *Transaction Offset* field is concatenated to the page pointer to produce a physical memory address for the data. • Data PIDs for OUTs are selected based on the current progress state of the transfer (which includes number of transactions executed and remaining buffer). Assertions below enumerate the required cases. • Data PIDs for INs are tracked by the host controller, based on the current process state of the transfer (which includes number of transactions executed and remaining buffer). Assertions below enumerate the required cases. **Specification Ref:** 3.3 **Test Description:** TD.3.12, TD.3.19
2.2.19	The host controller must detect a memory page boundary (bits [11:0] of buffer address going from FFFH to 000H). It must increment internal copy of *PG* and form a new physical memory address for the next page, without disturbing the flow of data over the USB. **Specification Ref:** 3.3.2, 4.7.1 **Test Description:** TD.3.12
2.2.20	The host controller must be able to detect a page boundary and transparently move to the next page regardless of the initial value of *Transaction Offset* (.e.g. from any byte boundary). **Specification Ref:** 3.3.2, 4.7.1, 4.7 **Test Description:** TD.3.12
2.2.21	The host controller processing an iTD must be able to correctly handle a data packet (IN or OUT) that starts on any byte boundary. **Specification Ref:** 3.3.2 **Test Description:** TD.3.13

Isochronous Transfer Descriptor (iTD) – Section 3.3

Num	Assertion
2.2.22	The host controller processing an iTD, must be able to correctly handle a data packet (IN or OUT) that ends on any byte boundary. **Specification Ref:** 3.3.2 **Test Description:** TD.3.13
2.2.23	The host controller may move up to the value of *MULT* transactions per transaction record (depending on time left in frame, device PID response (IN-only) and available remaining buffer space). **Specification Ref:** 3.3.3, 4.7.1 **Test Description:** TD.3.13, TD.3.19
2.2.24	When the host controller determines the transfer is done (either all transactions executed, or all data moved, or an error condition detected), it must: • clear the *Active* bit, and • set any appropriate error status bits, and • write the *Status* and optionally the *Transaction Length* fields back to the appropriate iTD transaction record in memory. **Specification Ref:** 3.3.2, 4.7.1 **Test Description:** TD.3.12, TD.3.13, TD.3.19
2.2.25	When the host controller detects that a transfer is done (see previous) and the *IOC* bit is a one, then the host controller will set the USBSTS register *USBINT* to a one. **Specification Ref:** 3.3 **Test Description:** TD.3.8
2.2.26	If the iTD is an OUT and current iTD transfer state is: *MULT* 1 *CurrentXCount* 1 Then the next transaction will have a DATA0 PID and the data payload will be the minimum of *Maximum Packet Size* or *Remaining Buffer* bytes. After the transaction is complete, the host controller will transition *Active* to a zero and write back the results of the transfer. **Specification Ref:** Appendix D **Test Description:** TD.3.19
2.2.27	If the iTD is an OUT and current iTD transfer state is: *MULT* 2 *CurrentXCount* 1 Then the next transaction must have a DATA1 PID and the data payload will be the minimum of *Maximum Packet Size* or *Remaining Buffer* bytes. After the transaction is complete, the host controller will transition *Active* to a zero and write back the results of the transfer. **Specification Ref:** Appendix D **Test Description:** TD.3.19

Isochronous Transfer Descriptor (iTD) – Section 3.3

Num **Assertion**

2.2.28 If the iTD is an OUT and current iTD transfer state is:

MULT	3
CurrentXCount	1

Then the next transaction must have a DATA2 PID and the data payload will be the minimum of *Maximum Packet Size* or *Remaining Buffer* bytes. After the transaction is complete, the host controller will transition *Active* to a zero and write back the results of the transfer.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.29 If the iTD is an OUT and current iTD transfer state is:

MULT	3 or 2
CurrentXCount	> 1
RemainingBuffer	≥ *Maximum Packet Size*

Then the next transaction must have an MDATA PID and the data payload will be *Maximum Packet Size* bytes. After the transaction is complete, the host controller will update the transfer state and prepare for another transaction for this iTD.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.30 If the iTD is an OUT and current iTD transfer state is:

MULT	2
CurrentXCount	2
RemainingBuffer	< *Maximum Packet Size*

Then the next transaction must have a DATA0 PID and the data payload will be *RemainingBuffer* bytes. After the transaction is complete, the host controller will transition *Active* to a zero and write back the results of the transfer.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.31 If the iTD is an OUT and current iTD transfer state is:

MULT	3
CurrentXCount	2
RemainingBuffer	< *Maximum Packet Size*

Then the next transaction must have a DATA1 PID and the data payload will be *RemainingBuffer* bytes. After the transaction is complete, the host controller will transition *Active* to a zero and write back the results of the transfer.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.32 If the iTD is an OUT and current iTD transfer state is:

MULT	3
CurrentXCount	3
RemainingBuffer	< *Maximum Packet Size*

Then the next transaction must have a DATA0 PID and the data payload will be *RemainingBuffer* bytes. After the transaction is complete, the host controller will transition *Active* to a zero and write back the results of the transfer.

Specification Ref: Appendix D

Test Description: TD.3.19

Isochronous Transfer Descriptor (iTD) – Section 3.3

Num **Assertion**

2.2.33 If the iTD is an IN and the current iTD transfer state is:

MULT	3
CurrentXCount	3
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA2 and the size of the data is *Maximum Packet Size*, then the host controller will accept the data packet and will update the transfer state (expecting to execute two more transactions) and prepare the next transaction for this iTD.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.34 If the iTD is an IN and the current iTD transfer state is:

MULT	3
CurrentXCount	3
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA2 and the size of the data is less than *Maximum Packet Size*, then the host controller will:

- set the *Status* field *XactErr* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- optionally update *Transaction Length* with the number of bytes received, and
- write back the required fields to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.35 If the iTD is an IN and the current iTD transfer state is:

MULT	3
CurrentXCount	3
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA1 and the size of the data is *Maximum Packet Size*, then the host controller will accept the data packet and will update the transfer state (expecting to execute one more transaction) and prepare the next transaction for this iTD.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.36 If the iTD is an IN and the current iTD transfer state is:

MULT	3
CurrentXCount	3
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA1 and the size of the data is less than *Maximum Packet Size*, then the host controller will accept the data packet and will:

- set the *Status* field *XactErr* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- write back the status to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

Isochronous Transfer Descriptor (iTD) – Section 3.3

Num **Assertion**

2.2.37 If the iTD is an IN and the current iTD transfer state is:

MULT	3
CurrentXCount	3
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA0 (note that the size of the received data can be ≤ *Maximum Packet Size*), then the host controller will accept the data packet and will:

- set the *Status* field *Active* bit to a zero, and
- update the *Transfer Length* field to reflect the number of bytes received
- write back the required fields to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.38 If the iTD is an IN and the current iTD transfer state is:

MULT	3
CurrentXCount	2
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA2 then the host controller will:

- set the *Status* field *XactErr* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- optionally update *Transaction Length* with the number of bytes received, and
- write back the required fields to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.39 If the iTD is an IN and the current iTD transfer state is:

MULT	3
CurrentXCount	2
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA1 and the size of the data is *Maximum Packet Size*, then the host controller will accept the data packet and will update the transfer state (expecting to execute one more transaction) and prepare the next transaction for this iTD.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.40 If the iTD is an IN and the current iTD transfer state is:

MULT	3
CurrentXCount	2
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA1 and the size of the data is less than *Maximum Packet Size*, then the host controller will accept the data packet and will:

- set the *Status* field *XactErr* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- write back the status to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

Isochronous Transfer Descriptor (iTD) – Section 3.3

Num **Assertion**

2.2.41 If the iTD is an IN and the current iTD transfer state is:

MULT	3
CurrentXCount	2
RemainingBuffer	< *Maximum Packet Size*

And the received data PID is a DATA0 then the host controller will:

- set the *Status* field *XactErr* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- optionally update *Transaction Length* with the number of bytes received, and
- write back the required fields to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.42 If the iTD is an IN and the current iTD transfer state is:

MULT	2
CurrentXCount	2
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA2 then the host controller will:

- set the *Status* field *XactErr* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- optionally update *Transaction Length* with the number of bytes received, and
- write back the required fields to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.43 If the iTD is an IN and the current iTD transfer state is:

MULT	2
CurrentXCount	2
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA1 and the size of the data is *Maximum Packet Size*, then the host controller will accept the data packet and will update the transfer state (expecting to execute one more transaction) and prepare the next transaction for this iTD.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.44 If the iTD is an IN and the current iTD transfer state is:

MULT	2
CurrentXCount	2
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA1 and the size of the data is less than *Maximum Packet Size*, then the host controller will:

- set the *Status* field *XactErr* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- write back the required fields to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

Isochronous Transfer Descriptor (iTD) – Section 3.3

Num **Assertion**

2.2.45 If the iTD is an IN and the current iTD transfer state is:

MULT	2
CurrentXCount	2
RemainingBuffer	≥ *Maximum Packet Size*

And the received data PID is a DATA0 (note that the size of the received data can be ≤ *Maximum Packet Size*), then the host controller will accept the data packet and will:

- set the *Status* field *Active* bit to a zero, and
- update the *Transfer Length* field to reflect the number of bytes received
- write back the required fields to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.46 If the iTD is an IN and the current iTD transfer state is:

MULT	2
CurrentXCount	2
RemainingBuffer	< *Maximum Packet Size*

And the received data PID is a DATA0 and the received data is ≤ than *RemainingBuffer*, then the host controller will accept the data packet and will:

- set the *Status* field *Active* bit to a zero, and
- update the *Transfer Length* field to reflect the number of bytes received
- write back the required fields to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.47 If the iTD is an IN and the current iTD transfer state is:

MULT	3, 2 or 1
CurrentXCount	1

And the received data PID is a DATA0 (note that the amount of data received can be less than or equal to the minimum of *Maximum Packet Size* or *RemainingBuffer*), then the host controller will accept the data packet and will:

- set the *Status* field *Active* bit to a zero, and
- update the *Transfer Length* field to reflect the number of bytes received
- write back the required fields to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

2.2.48 If the iTD is an IN and the current iTD transfer state is:

MULT	3, 2 or 1
CurrentXCount	1

And the received data PID is NOT a DATA0, then the host controller will:

- set the *Status* field *XactErr* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- optionally update *Transaction Length* with the number of bytes received, and
- write back the status to the selected transaction record.

Specification Ref: Appendix D

Test Description: TD.3.19

Isochronous Transfer Descriptor (iTD) – Section 3.3

Num	Assertion
2.2.49	If *MULT* is greater than 1 and the *CurrentXCount* is less than *MULT*, and the end of Micro-frame occurs before all of the transactions have been completed by the host controller, the host controller must update the transfer state to indicate how much data progress was made on the transfer, set the *Active* bit to zero and the required fields written back to the selected transaction record. **Specification Ref:** Appendix D **Test Description:** TD.3.20
2.2.50	If the iTD is an IN, and the host controller receives more than the minimum of *Maximum Packet Size* or *Remaining Buffer* bytes in a bus transaction the host controller must: • set the *Status* field *Babble Detected* bit to a one, and • set the *Status* field *Active* bit to a zero, and • write back the status to the selected transaction record. **Specification Ref:** 3.3.2, 4.7.1 **Test Description:** TD.3.9
2.2.51	If the iTD is an IN and the host controller experiences a data over-run writing incoming data to memory, the host controller must: • set the *Status* field *Data Buffer* bit to a one, and • set the *Status* field *Active* bit to a zero, and • write back the status to the selected transaction record. **Specification Ref:** 3.3.2 **Test Description:** Not-Tested
2.2.52	If the iTD is an OUT and the host controller experiences a data under-run reading outgoing data from memory, the host controller must: • corrupt the current data packet by transmitting an incorrect CRC, and • set the *Status* field *Data Buffer Error* bit to a one, and • set the *Status* field *Active* bit to a zero, and • write back the status to the selected transaction record. **Specification Ref:** 3.3.2 **Test Description:** Not-Tested

2.2.3 Split Isochronous Transfer Descriptor (siTD) — Section 3.4

As with iTDs, the FRINDEX register plays an important role in managing siTDs. Specifically, it is used to both index into the *C-mask* and *S-mask* bit vectors to help determine when it is time to execute transactions and to track progress of the split transaction over the series of micro-frames it must execute.

This document uses the term: *Split Transaction* to denote the aggregate Start-Splits and Complete-Splits required to initiate the low- or full-speed transaction on non-high-speed bus and to extract the complete results of the low- for full-speed transaction from the transaction translator.

Note, that for all siTD OUTs, no handshake is expected from the addressed Hub.

Split Isochronous Transfer Descriptor (siTD) – Section 3.4

Num	Assertion

Split Isochronous Transfer Descriptor (siTD) – Section 3.4

Num **Assertion**

2.2.53 The host controller will execute a bus transaction for an siTD if and only if the following are true:

- *Active* bit in the *Status* byte is a one, and
- The *SplitXState* bit indicates a **StartSplit** (0b), and:
  - the *S-mask* has a one in bit position selected by the current value of FRINDEX[2:0], or
- The *SplitXState* bit indicates a **CompleteSplit** (1b), and:
  - the *C-mask* has a one in bit position selected by the current value of FRINDEX[2:0], and
  - a check with *C-prog-mask* indicates no transactions have been missed

Specification Ref: 3.4.3, 4.12.3.3

Test Description: TD.3.14

2.2.54 To execute a transaction from an active siTD, the following fields are used for the indicated purpose:

- The host controller maps siTD fields into Extended Token packet fields based on:

siTD Field	**Extended Token Field**
HubAddr	Hub Address
Port Number	Port Number
Status.SplitXState	SC (start/complete flag)
siTD	ET (isochronous because siTD)
TP	S/E fields for OUTs for progress of transfer

- The host controller builds the Token packet from siTD fields: *ENDPT*, *DEVICEADDRESS*, and *I/O*. If the *I/O* field indicates an IN (value of 1b), the Token PID will be an IN. If the *I/O* field indicates an OUT (value of 0b), then the Token PID will be an OUT.
- Data PIDs for OUTs are always DATA0.
- The physical buffer address for where the host controller will transfer data from/to is constructed by using the *P* (page select) to select which buffer page pointer to use. The *Current Offset* field is concatenated to the page pointer to produce a physical memory address for the data.

Specification Ref: 3.4.2, 3.4.4

Test Description: TD.3.14

2.2.55 The host controller must detect memory page boundaries (bits [11:0] of a buffer address going from FFFH to 000H). It must adjust the internal value of *P* and form a new physical memory address for the next page from the buffer page pointer selected by the new value of *P*.

Specification Ref: 3.4.3,

Test Description: TD.3.14

2.2.56 While performing a data transfer in the context of an siTD, the host controller must be able to detect a page boundary condition and transparently change the data buffer address to the new page without impacting the data stream, regardless of the initial value of *Current Offset*.

Specification Ref: 3.4.3, 4.12.3.3.1

Test Description: TD.3.14

2.2.57 The host controller processing an siTD must be able to correctly handle a data packet (IN or OUT) that starts on any byte boundary.

Specification Ref: 3.4.4, 4.12.3.3.1, 4.12.3.3.2

Test Description: TD.3.15

Split Isochronous Transfer Descriptor (siTD) – Section 3.4

Num **Assertion**

2.2.58 The host controller processing a siTD, must be able to correctly handle a data packet (IN or OUT) that ends on any byte boundary.

Specification Ref: 3.4.4, 4.12.3.3.1, 4.12.3.3.2

Test Description: TD.3.15

2.2.59 When the host controller completes a bus transaction for an siTD, it must:

- advance the transfer state (adjust *Total Bytes to Transfer* and *Current Offset* by the number of bytes moved) and,
- adjust the *SplitXState* if necessary, and
- adjust the *C-prog-mask* as required, and
- adjust the *TP* and *T-count* fields if *I/O* indicates an OUT, and
- set any appropriate *Status* bits depending on the result of the bus transaction, and
- write the required fields back to the siTD in memory (including the *P* field).

Specification Ref: 4.12.3.3.1, 4.12.3.3.2

Test Description: TD.3.14, TD.3.15, TD.3.23

2.2.60 If *SplitXState* is **Do Start Split** and *I/O* is an OUT and the criteria for doing a bus transaction are met, and the *Transaction Position* state is:

TP	00b (ALL)
T-count	1

Then the host controller will issue a Start-Split OUT transaction with an Extended Token S/E field value equal to *TP* (00b) and a data payload equal to the minimum of 188 or the value of *Bytes to Transfer*. After the bus transaction is complete, the host controller will set *Active* to a zero and write back the results of the transaction to the siTD.

Specification Ref: 4.12.3.3.1

Test Description: TD.3.14, TD.3.15, TD.3.23

2.2.61 If *SplitXState* is **Do Start Split** and *I/O* is an OUT and the criteria for doing a bus transaction are met, and the *Transaction Position* state is:

TP	01b (BEGIN)
T-count	2

Then the host controller will issue a Start-Split OUT transaction with an Extended Token S/E field value equal to *TP* (01b) and the value of *Bytes to Transfer* is greater than 188. After the bus transaction is complete, the host controller will:

- decrement *T-count* to a value of 1, and
- set *TP* to a value of 11b (END), and
- decrement *Total Bytes to Transfer* by the number of data bytes sent in the bus transaction data payload.
- Write back the required fields into the siTD.

Specification Ref: 4.12.3.3.1

Test Description: TD.3.15, TD.3.23

Split Isochronous Transfer Descriptor (siTD) – Section 3.4

Num	Assertion

2.2.62 If *SplitXState* is **Do Start Split** and *I/O* is an OUT and the criteria for doing a bus transaction are met, and the *Transaction Position* state is:

TP	01b (BEGIN)
T-count	> 2

Then the host controller will issue a Start-Split OUT transaction with an Extended Token S/E field value equal to *TP* (01b) and the value of *Bytes to Transfer* is greater than 2x188 (or larger as is appropriate for the selected value of *T-count*). After the bus transaction is complete, the host controller will:

- decrement *T-count*, and
- set *TP* to a value of 10b (MID), and
- decrement *Total Bytes to Transfer* by the number of data bytes sent in the bus transaction data payload (188), and
- write back the required fields into the siTD.

Specification Ref: 4.12.3.3.1

Test Description: TD.3.15, TD.3.23

2.2.63 If *SplitXState* is **Do Start Split** and *I/O* is an OUT and the criteria for doing a bus transaction are met, and the *Transaction Position* state is:

TP	10b (MID)
T-count	> 2

Then the host controller will issue a Start-Split OUT transaction with an Extended Token S/E field value equal to *TP* (10b) and the value of *Bytes to Transfer* is greater than 2x188 (or larger as is appropriate for the selected value of *T-count*). After the bus transaction is complete, the host controller will:

- decrement *T-count*, and
- set *TP* to a value of 10b (MID), and
- decrement *Total Bytes to Transfer* by the number of data bytes sent in the bus transaction data payload (188), and
- write back the required fields into the siTD.

Specification Ref: 4.12.3.3.1

Test Description: TD.3.15, TD.3.23

2.2.64 If *SplitXState* is **Do Start Split** and *I/O* is an OUT and the criteria for doing a bus transaction are met, and the *Transaction Position* state is:

TP	10b (MID)
T-count	2

Then the host controller will issue a Start-Split OUT transaction with an Extended Token S/E field value equal to *TP* (10b) and the value of *Bytes to Transfer* is greater than 188. After the bus transaction is complete, the host controller will:

- decrement *T-count*, and
- set *TP* to a value of 11b (END), and
- decrement *Total Bytes to Transfer* by the number of data bytes sent in the bus transaction data payload (188), and
- write back the required fields into the siTD.

Specification Ref: 4.12.3.3.1

Test Description: TD.3.15, TD.3.23

Split Isochronous Transfer Descriptor (siTD) – Section 3.4

Num	Assertion
2.2.65	If *SplitXState* is **Do Start Split** and *I/O* is an OUT and the criteria for doing a bus transaction are met, and the *Transaction Position* state is: *TP* 11b (END) *T-count* 1 Then the host controller will issue a Start-Split OUT transaction with an Extended Token S/E field value equal to *TP* (11b) and a data payload equal to the minimum of 188 or the value of *Bytes to Transfer*. After the bus transaction is complete, the host controller will: • decrement *Total Bytes to Transfer* by the number of data bytes sent in the bus transaction data payload, and • set *Active* to a zero, and • write back the required fields into the siTD. **Specification Ref:** 4.12.3.3.1 **Test Description:** TD.3.15, TD.3.23
2.2.66	If the *I/O* field indicates an OUT, the host controller will never transition *SplitXState* to **Do Complete Split**. **Specification Ref:** 4.12.3.3.1 **Test Description:** TD.3.14
2.2.67	If the *I/O* field indicates an IN and the host controller receives a NYET handshake from the Transaction Translator (in response to a Complete-split), then the host controller must: • update *C-prog-mask* with the current micro-frame information, and • check to ensure this is not the last scheduled complete-split, and • write back the required fields to the siTD. **Specification Ref:** 4.12.3.3.2 **Test Description:** TD.3.14, TD.3.15, TD.3.23
2.2.68	If the *I/O* field indicates an IN and the host controller receives a NYET handshake from the Transaction Translator (in response to a Complete-split), and it is the last scheduled complete-split, then the host controller must: • update *C-prog-mask* with the current micro-frame information, and • verify this is the last scheduled complete-split. If it is the last scheduled complete-split, then the host controller must set the *XactErr* bit to a one, and • set *Active* to a zero, and • write back the required fields to the siTD. **Specification Ref:** 4.12.3.3.2 **Test Description:** TD.3.14, TD.3.15, TD.3.23
2.2.69	If the *I/O* field indicates an IN and the host controller receives a data packet from the Transaction Translator with a DATA0 (or DATA1) PID (in response to a Complete-split), then the host controller must: • update *C-prog-mask* with the current micro-frame information, and • advance the transfer state to reflect the data received, and • set *Active* to a zero, and • write back the required fields to the siTD. **Specification Ref:** 4.12.3.3.2 **Test Description:** TD.3.14, TD.3.15, TD.3.23

Split Isochronous Transfer Descriptor (siTD) – Section 3.4

Num	Assertion
2.2.70	If the *I/O* field indicates an IN and the host controller receives a data packet from the Transaction Translator with an MDATA PID (in response to a Complete-split), then the host controller must: • update *C-prog-mask* with the current micro-frame information, and • advance the transfer state to reflect the data received (Note that the *Total Bytes to Transfer* field may be set to zero as a result of this data payload. When this occurs and the data PID is MDATA, the host controller MUST NOT set *Active* to a zero), and • check to ensure this is not the last scheduled complete-split. • write back the required fields to the siTD. **Specification Ref:** 4.12.3.3.2 **Test Description:** TD.3.23
2.2.71	If the *I/O* field indicates an IN and the host controller receives a data packet from the Transaction Translator with an MDATA PID (in response to a Complete-split), then the host controller must: • update *C-prog-mask* with the current micro-frame information, and • advance the transfer state to reflect the data received, and • check to if this is the last scheduled complete-split. If it is the last scheduled complete-split, then the host controller must set the *XactErr* bit to a one, and set *Active* to a zero, and • write back the required fields to the siTD. **Specification Ref:** 4.12.3.3.2 **Test Description:** TD.3.24
2.2.72	If the *SplitXState* is **Do Compete Split** and the value of FRINDEX[2:0] is 0h or 1h, then the host controller must save the context of the siTD and use its Back Pointer field to read in the previous frame's siTD. The host controller will then (using the normal rules), determine whether to issue a bus transaction, based on the context of the previous frames' siTD. If a bus transaction is executed for this back pointer context, then the host controller must observe all execution and update rules for siTDs. **Specification Ref:** 4.12.3.3.2.1 **Test Description:** TD.3.23
2.2.73	When the host controller returns from the context of an siTD reached via a Back Pointer, it needs to determine whether (or not) to modify or even execute a bus transaction for the saved siTD context. If and only if the back pointer's siTD's *Active* bit is zero, upon returning to the save siTD state, the host controller must: • transition the siTD state to **Do Start Split**, and • determine whether a Start-Split bus transaction should be executed at this time, if so, the bus transaction is executed using all execution and update rules for siTDS, and • write back the required fields to the siTD. **Specification Ref:** 4.12.3.3.2.1 **Test Description:** TD.3.23

Split Isochronous Transfer Descriptor (siTD) – Section 3.4

Num **Assertion**

2.2.74 If the *SplitXState* is **Do Complete Split** and the host controller does not receive a good packet response from the Transaction Translator (including: Timeout, CRC, Bad PID, Wrong PID, etc.) then the host controller must:

- set *XactErr* to a one, and
- *IMMEDIATELY* retry the bus transaction.

The host controller must not retry the transaction more than two times.

Specification Ref: 4.12.3.3.2

Test Description: TD.3.24

2.2.75 If the host controller cannot issue an immediate retry of a complete-split transaction for an siTD due to reaching the end of the micro-frame, then it must:

- set *XactErr* to a one, and
- set *Active* to a zero, and
- write back the required fields to the siTD.

Specification Ref: 4.12.3.3.2

Test Description: TD.3.24

2.2.76 If the host controller exhausts the immediate retry count,(all bus transactions fail to complete successfully) then the host controller must:

- set *XactErr* to a one, and
- set *Active* to a zero, and
- write back the required fields to the siTD.

Specification Ref: 4.12.3.3.2

Test Description: TD.3.24

2.2.77 If the host controller detects that it has missed issuing a start or complete-split bus transaction, it must:

- set the *Status* field *Missed Micro-Frame* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- write back the required fields to the siTD.

Specification Ref: 4.12.3.3.1, 4.12.3.3.2

Test Description: Not-Tested

2.2.78 If the *I/O* field indicates an IN and the host controller receives more than the minimum of *Total Bytes to Transfer* and 188, then the host controller must:

- set the *Status* field *Babble* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- write back the required fields to the siTD.

Specification Ref: 4.12.3.3.2

Test Description: TD.3.24

2.2.79 If the *I/O* field indicates an IN and the host controller receives an ERR handshake packet from the Transaction Translator then the host controller must:

- set the *Status* field *ERR* bit to a one, and
- set the *Status* field *Active* bit to a zero, and
- write back the required fields to the siTD.

Specification Ref: 3.4.3, 4.12.3.3.2

Test Description: TD.3.24

Split Isochronous Transfer Descriptor (siTD) – Section 3.4

Num	Assertion
2.2.80	If the host controller experiences a data under-run reading outgoing data from memory during an OUT transaction, the host controller must: • corrupt the current data packet by transmitting an incorrect CRC, and • set the *Status* field *Data Buffer Error* bit to a one, and • set the *Status* field *Active* bit to a zero, and • write back the required fields to the siTD. **Specification Ref:** 3.4.3 **Test Description:** Not-Tested

2.2.4 Asynchronous List — Section 3.2

The purpose of the Asynchronous Advance Doorbell feature is to provide system software a handshake form the host controller to ensure that it has flushed all cached asynchronous schedule state.

Asynchronous List – Section 3.2

Num	Assertion
2.2.81	The host controller must ensure that ASYNCLISTADDR register is maintained to ensure that the circular list of queue heads on the Asynchronous Schedule are serviced in strict round-robin order (with one exception noted below). **Specification Ref:** 1.2.1, 3.2, 4.8 **Test Description:** TD.3.4
2.2.82	The End-of-Micro-frame may occur and inhibit the host controller from performing an IMMEDIATE retry of a Low/Full-speed asynchronous complete-split (see Qhead assertions). If this occurs, the host controller must ensure the next time the Asynchronous Schedule gets to run it MUST continue the IMMEDIATE retry for the same complete-split bus transaction. **Specification Ref:** 4.12.1.2 **Test Description:** TD.3.26
2.2.83	When the host controller observes a Qhead with an *H-bit* with a value of one, it will transition USBSTS register **Reclamation** bit to a zero. **Specification Ref:** 4.8.3 **Test Description:** TD.3.3
2.2.84	The host controller must set USBSTS register **Reclamation** bit to a one whenever it executes a bus transaction from the Asynchronous Schedule. **Specification Ref:** 4.8.3 **Test Description:** TD.3.3
2.2.85	The host controller must set USBSTS register **Reclamation** bit to a one when the host controller transitions from the Periodic Schedule traversal (or SOF time) to actively traversing the Asynchronous Schedule. **Specification Ref:** 4.8.3 **Test Description:** TD.3.3
2.2.86	The host controller must set USBSTS register **Reclamation** bit to a one when the host controller resumes traversal of the Asynchronous Schedule AFTER it has an Empty List Detected condition during a micro-frame. **Specification Ref:** 4.8.3 **Test Description:** TD.3.3

Asynchronous List – Section 3.2

Num	Assertion
2.2.87	The host controller will stop executing from the Asynchronous Schedule if it reaches the End-of-Micro-Frame. **Specification Ref:** 4.8 **Test Description:** TD.3.17
2.2.88	The host controller will stop executing from the Asynchronous Schedule if it observes a Qhead *H-bit* with a value of one and a USBSTS Register **Reclamation** bit with a value of zero. This is called Empty List Detected. **Specification Ref:** 4.8.3 **Test Description:** TD.3.3
2.2.89	Marking from the beginning of a Micro-frame (SOF), the host controller will traverse the Asynchronous Schedule only AFTER it has completed traversing the Periodic Schedule for this micro-frame. Completing the Periodic Schedule means either the Periodic Schedule is not active or the host controller encountered a *T-bit* value of one in a schedule link pointer. **Specification Ref:** 4.4 **Test Description:** TD.1.7
2.2.90	If the host controller stops executing from the Asynchronous Schedule due to an Empty List Detected condition, it must implement a *Waking* method which allows the host controller to resume traversal of the Asynchronous Schedule, when there is sufficient time remaining in the micro-frame. **Specification Ref:** 4.8.4 **Test Description:** TD.3.3
2.2.91	The host controller must traverse the Asynchronous schedule after executing a bus transaction for Qhead before executing another bus transaction for the same Qhead. Note the exception to this assertion is an immediate retry for a transaction error during a complete-split bus transaction. **Specification Ref:** 4.8 **Test Description:** TD.3.4
2.2.92	When software sets USBCMD register *Interrupt on Async Advance Doorbell* to a one, then the host controller must set USBSTS register *Interrupt on Async Advance* to a one and set USBCMD register *Interrupt on Async Advance Doorbell* to a zero, when it has traversed beyond all of the reachable asynchronous schedule at the instant the doorbell bit was set to a one. **Specification Ref:** 4.8.2 **Test Description:** TD.3.5

2.2.5 Queue Heads (QH, qTD) — Sections 3.5, 3.6

Qheads and qTDs are used to manage all asynchronous (control and bulk), and interrupt data streams, regardless of endpoint speed. The host controller dif and only iferentiates between an interrupt and asynchronous transfer by looking at the *S-mask*. It is the responsibility of software to ensure that the *S-mask* is set appropriately for each instance of a Qhead. The decision criteria whether to execute a bus transaction in a Qhead context are slightly dif and only iferent between an asynchronous Qhead and an interrupt Qhead.

In the following set of assertions, the *Transfer State* is set of Qhead fields:{*Total Bytes to Transfer*, *C_Page*, *DT*, *Current Offset*}. Advancing the transfer state means that after a bus transaction is complete:

- *Total Bytes to Transfer* is decremented by the number of bytes moved during the bus transaction,
- *C_Page* may have been adjusted during the bus transaction if a page cross was encountered during the data packet,

- *DT* is toggled, and
- *Current Offset* is incremented by the number of bytes moved during the bus transaction).

Note that Transfer State is not *Committed* until it has been written back to the Qhead in memory.

A Halt condition is any event that occurs that causes the host controller to set the *Halted* bit in a Qhead to a one.

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num	Assertion
2.2.93	The host controller will ignore a Qhead if its *Halted* bit is a one and its *Active* bit is a zero. **Specification Ref:** 4.10.2 **Test Description:** TD.3.9
2.2.94	The host controller will attempt to advance a queue if the *Active* bit in the Qhead's overlay area is zero and the *Halted* bit is a zero. The host controller will select a qTD pointer (see next two assertions) and read the qTD referenced from the selected pointer. If the *Active* bit in the qTD is a one, then the host controller moves the value of the pointer used to access the qTD into the field *Current qTD Pointer*, performs the overlay, and attempts to execute a bus transaction. **Specification Ref:** 4.10.2 **Test Description:** TD.3.10, TD.3.11
2.2.95	The host controller will attempt to advance a queue by using the *Alternate qTD Pointer* field when the Qhead's *Total Bytes to Transfer* field is non-zero and the *T-bit* in the *Alternate qTD Pointer* is zero. **Specification Ref:** 4.10.2 **Test Description:** TD.3.21
2.2.96	The host controller will attempt to advance a queue by using the *Next qTD Pointer* field if it determines it cannot use the *Alternate qTD Pointer* and the *T-bit* in the *Next qTD Pointer* is zero. **Specification Ref:** 4.10.2 **Test Description:** TD.3.10, TD.3.21
2.2.97	During an overlay operation, the host controller preserves the existing value of *DT* if the value of *DTC* is a zero. **Specification Ref:** 3.6.3, 4.10.2 **Test Description:** TD.3.10, TD.3.11
2.2.98	During an overlay operation, the host controller replaces the Qhead's *DT* field value with the value of the *DT* bit in the qTD if the value of *DTC* is a one. **Specification Ref:** 3.6.3, 4.10.2 **Test Description:** TD.3.16
2.2.99	During an overlay operation, the host controller preserves the existing value of *Ping State* if the *EPS* field indicates a high-speed device (10b). **Specification Ref:** 3.6.3, 4.10.2 **Test Description:** TD.3.10
2.2.100	During an overlay operation, the host controller always sets to zero the *C-prog-mask* and *FrameTag* fields in the Qhead. **Specification Ref:** 3.6.3, 4.10.2 **Test Description:** TD.3.10

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num	Assertion
2.2.101	During an overlay operation, the *NakCnt* field is loaded from the *RL* field in the Qhead. **Specification Ref:** 3.6.3, 4.10.2 **Test Description:** TD.3.3
2.2.102	The host controller must reload all Qhead *NakCnt* fields (from the *RL* field) only on the *first* pass through the Asynchronous Schedule. The *NakCnt* field must be reloaded *before* the host controller checks to determine whether to issue a bus transaction. The host controller must keep track of when it begins a new activation of the Asynchronous Schedule traversal. This occurs on the transition from the Periodic Schedule, or when resuming after detecting a empty list condition. The host controller reloads all queue heads' *NakCnt* fields until it encounters the Qhead with the *H-bit* set to a one. **Specification Ref:** 4.10.2, 4.9.1 **Test Description:** TD.3.3
2.2.103	The host controller will issue a bus transaction for a Qhead with an *S-mask* of zero if and only if the following conditions in the Qhead are true: • the Empty List Check is false, and • the *NakCnt* field is non-zero, or the *RL* field is zero, and • the *Active* bit in the Qhead is a one and the *Halted* bit is a zero. **Specification Ref:** 4.10.3 **Test Description:** TD.3.3, TD.3.10
2.2.104	The host controller will issue a bus transaction for a Qhead with an *S-mask* of non-zero if and only if the following conditions in the Qhead are true (note that the host controller is not required to check *S-mask* when traversing the Asynchronous Schedule): • FRINDEX[2:0] matches on a bit in *S-mask*, and • the *Mult* field is non-zero, and • the *Active* bit in the Qhead is a one and the *Halted* bit is a zero. Note that these requirements apply to an *EPS* encoding of high-speed (10b) and also to *EPS* encoding of (00b and 01b) and a *SplitXState* of **Do Start Split** (0b). **Specification Ref:** 4.10.3 **Test Description:** TD.3.10
2.2.105	The host controller executes a high-speed bus transaction from a qualifying Qhead when the *EPS* field indicates a high-speed device (10b). The following fields are used to manage the bus transaction: • the host controller builds the Token packet from the Qhead fields: *PID_Code*, *Ping State* (in the *Status* field), *DEVICE ADDRESS* and *ENDPT*. • The starting physical buffer address for where the host controller will transfer data from/to is constructed using the *C_Page* field in the Qhead to select a starting Page pointer from the *Buffer Page Pointer List*. The *Current Offset* field is concatenated to the page pointer to produce a physical memory address for the data stream. • Data PIDs for OUTs/SETUPs are determined from the *DT* field. If *DT* is a zero, then the host controller will generate a DATA0. If *DT* is a one then the host controller will generate a DATA1. • Data PIDs for INs are tracked using the *DT* field. **Specification Ref:** 4.10.3 **Test Description:** TD.3.10

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num	Assertion
2.2.106	The PID for a Token packet must match the intended PID selected by the Qhead's *PID_Code* field. An exception to this rule is when the *PID_Code* indicates an OUT, see next assertion. **Specification Ref:** 3.5.3 **Test Description:** TD.3.10
2.2.107	If the *PID_Code* indicates an OUT and *EPS* indicates a high-speed device (10b), then the host controller will use *Status* field *Ping State* to determine which PID to use in the Token. If *Ping State* is a 0b, then the host controller will use an OUT PID. If the *Ping State* is a 1b, then the host controller will use a PING PID. **Specification Ref:** 3.5.3 **Test Description:** TD.3.6
2.2.108	The host controller will not issue split transactions from a qualifying Qhead when the *EPS* field indicates a high-speed device (10b). **Specification Ref:** 3.6.2 **Test Description:** TD.3.10
2.2.109	The host controller must detect a data buffer memory page boundary during an active transfer (bit [11:0] of a buffer address going from FFFH to 000H). It must increment an internal copy of *C_Page* and form a new physical memory address for the new page, without disturbing the flow of data over the USB. **Specification Ref:** 3.5.3, 3.5.4, 4.10.6 **Test Description:** TD.3.10
2.2.110	The host controller must correctly manage page crossing, regardless of the starting value of *C_Page*. Note that hardware is not required to correctly handle a case where *C_Page* is initialized by software so that it is expected to wrap (from 111b to 000b) before the end of the transfer. **Specification Ref:** 3.5.3, 3.5.4, 4.10.6 **Test Description:** TD.3.10
2.2.111	The host controller processing a Qhead must be able to correctly handle a data packet (IN or OUT) that starts on any byte boundary. **Specification Ref:** 3.5.3, 3.5.4, 4.10.6 **Test Description:** TD.3.10
2.2.112	The host controller processing a Qhead must be able to correctly handle a data packet (IN or OUT) that ends on any byte boundary. **Specification Ref:** 3.5.3, 3.5.4, 4.10.6 **Test Description:** TD.3.10
2.2.113	When a bus transaction (or series of bus transactions as in the case of an Interrupt endpoint with a *Mult* value greater than 1) is (are) finished, the host controller must: • update the transfer state to reflect the number bytes sent or received, and • set any *Status* field bits, and • make any adjustments to *Cerr* field, and • commit the state of the transfer by writing the required fields back to the Qhead. Required fields include: *NakCnt*, *DT*, *Total Bytes to Transfer*, *C_Page*, *Status*, *Cerr*, and *Current Offset*. *C-prog-mask*, *FrameTag*, and *S-bytes* are always additionally written back for an Interrupt Qhead with an *EPS* indicating Low- or Full-speed. **Specification Ref:** 4.10.3 **Test Description:** TD.3.10

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num	Assertion
2.2.114	The host controller will not commit the transfer state if the bus transaction experienced an error condition (transaction error, Babble, etc.). It will commit the *Status* and *Cerr* fields when errors occur. **Specification Ref:** 4.10.3 **Test Description:** TD.3.9
2.2.115	The host controller must set the *Active* bit in the Qhead to a zero whenever the Qhead's field *Total Bytes to Transfer* field decrements to zero as a result of the bus transaction. **Specification Ref:** 4.10.3 **Test Description:** TD.3.8, TD.3.10
2.2.116	This host controller will issue a zero-length bus transaction when the initial value of *Total Bytes to Transfer* is zero (e.g. when first read). This means for an OUT, the data packet will be a zero-length data packet. For an IN, the host will expect a zero-length data packet. **Specification Ref:** 4.10.3 **Test Description:** TD.3.11
2.2.117	The host controller will set the *Active* bit in the Qhead to a zero for *PID_Code* of IN (01b) when the number of bytes returned during the bus transaction returns less than the minimum of *Maximum Packet Length* or *Total Bytes to Transfer*. This rule applies regardless even if the short packet occurs in the midst of a transaction sequence for a high-bandwidth Interrupt transfer. **Specification Ref:** 4.10.3 **Test Description:** TD.3.21
2.2.118	The host controller will set the *XactErr* status bit to a one when the bus transaction determines that it did not received a valid response from the device (timeout, bad CRC, Bad PID, etc.). It will also decrement the *Cerr* field by one for each *XactErr*. Note: a Bad PID includes any PID response other than an Ack for a Setup Token. **Specification Ref:** 3.5.3, 4.10.3 **Test Description:** TD.3.9
2.2.119	The host controller must set the *Active* bit in the Qhead to a zero and the *Halted* bit to a one whenever the result of a bus transaction causes the *Cerr* field to decrement to a value of zero. Note that another way to say this is that the host controller must halt the endpoint when all tries (3) of a bus transaction result in transaction errors. **Specification Ref:** 3.5.3, 4.10.3 **Test Description:** TD.3.9
2.2.120	The host controller must set the *Active* bit in the Qhead to a zero and the *Halted* bit to a one whenever the result of a bus transaction is a STALL handshake. **Specification Ref:** 3.5.3, 4.10.3 **Test Description:** TD.6.2
2.2.121	The host controller must write back the required fields to the source qTD whenever the result of a bus transaction causes a Qhead's *Active* bit to transition from a one to a zero. The host controller always uses Qhead field *Current qTD Pointer* for the address to the source qTD. **Specification Ref:** 4.10.4 **Test Description:** TD.3.10, TD.3.11

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num **Assertion**

2.2.122 If the host receives the expected data PID, it must:

- perform packet babble checking based on the minimum of (*Total Bytes to Transfer* or *Maximum Packet Length*), and
- accept incoming data (qualified by CRC check), and
- if good data (good CRC), respond with ACK handshake and advance transfer state, otherwise
- if not good data (bad CRC), time out transfer, decrement *Cerr* and do not advance transfer state.

Specification Ref: 4.10.3

Test Description: TD.3.9, TD.3.10

2.2.123 If the host controller receives an data PID mismatch, it must:

- ignore packet babble checking (at least based on *Total Bytes to Transfer*), and
- ignore incoming data, and
- respond with an ACK handshake, and
- do not advance transfer state.

Specification Ref: 4.10.3

Test Description: TD.6.1

2.2.124 If the host controller receives a Nyet handshake to a complete-split token or a Nak handshake in to any other token and the Qhead's *NakCnt* field is non-zero, the host controller must decrement the value of *NakCnt* and commit this additional field to the Qhead when the bus transaction results are committed.

Specification Ref: 4.10.3

Test Description: TD.3.3

2.2.125 The host controller must set *Cerr* field to a 11b when a bus transaction completes without error.

Specification Ref: 4.10.3

Test Description: TD.3.9

Execution (HS-Asynchronous) PING

2.2.126 The host controller will not adjust the *Ping State* bit for a high-speed Qhead from **Do Ping** (1b) when the device response to the PING token is a Nak handshake.

Specification Ref: 4.11

Test Description: TD.3.6, TD.3.7

2.2.127 The host controller will not adjust the *Ping State* bit for a high-speed Qhead from **Do Ping** (1b) when the device response to the PING token is a transaction error.

Specification Ref: 4.11

Test Description: TD.3.6, TD.3.7

2.2.128 The host controller will adjust the *Ping State* bit for a high-speed Qhead from **Do Ping** (1b) to **Do OUT** (0b) when the device response to the PING token is an Ack handshake.

Specification Ref: 4.11

Test Description: TD.3.6, TD.3.7

2.2.129 The host controller will adjust the *Ping State* bit for a high-speed Qhead from **Do OUT** (0b) to **Do Ping** (1b) when the device response to the OUT token is a Nak handshake.

Specification Ref: 4.11

Test Description: TD.3.6, TD.3.7

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num **Assertion**

2.2.130 The host controller will adjust the *Ping State* bit for a high-speed Qhead from **Do OUT** (0b) to **Do Ping** (1b) when the device response to the OUT token is a Nyet handshake.

Specification Ref: 4.11

Test Description: TD.3.6, TD.3.7

2.2.131 The host controller will adjust the *Ping State* bit for a high-speed Qhead from **Do OUT** (0b) to **Do Ping** (1b) when the device response to the OUT token is a transaction error.

Specification Ref: 4.11

Test Description: TD.3.6, TD.3.7

2.2.132 The host controller will not adjust the *Ping State* bit for a high-speed Qhead from **Do OUT** (0b) when the device response to the OUT token is an ACK.

Specification Ref: 4.11

Test Description: TD.3.6

Execution High-bandwidth Interrupt

2.2.133 The host controller will execute *Mult* (where *Mult* is greater than 1) bus transactions during a micro-frame if:

- the *S-mask* is non-zero, and
- the initial value of *Total Bytes to Transfer* is greater than or equal to *Mult* * *Maximum Packet Size*, and
- there are no Naks (IN or OUT), or short data packets during the any of the *Mult* transactions.

Note, that bus transaction retries on errors is considered part of the *Mult* opportunities per poll period.

Specification Ref: 4.10.3

Test Description: TD.3.18

Execution (LS/FS-Asynchronous)

2.2.134 To execute a bus transaction when the *EPS* field is a 00b, or 01b (Full- or Low-speed device) the following fields are used to generate the Extended Token fields:

Qhead Field	**Extended Token Field**
Hub Addr	Hub Address
Port Number	Port Number
Status.SplitXState	SC (start/complete flag)
S-mask != 0	ET (interrupt - (11b))
(*C* = 1) & (*S-mask* = 0)	ET (control - (00b))
(C = 0) & (*S-mask* = 0)	ET (bulk - (10b))
EPS (Full-speed (00b))	S - 0b
EPS (Low-speed (01b))	S - 1b

The E/U-field in the Extended token must be zero.

The Token and Data packets are derived as usual.

Specification Ref: 4.10.3, 4.12.1.1

Test Description: TD.3.10

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num	Assertion
2.2.135	The host controller will move the *SplitXstate* from **Do Start Split** (0b) to **Do Complete Split** (1b) when it receives an Ack handshake in response to a start-split bus transaction. Note: the host controller will only expect a handshake packet from the transaction translator if the *S-mask* is zero. **Specification Ref:** 4.12.1.1 **Test Description:** TD.3.10
2.2.136	The host controller will retry an asynchronous Qhead complete-split bus transaction IMMEDIATELY if the bus transaction completes with a transaction error that does not result in a Halt condition (and there is time remaining in the micro-frame). When such an error occurs, the host controller will set *XactErr* to a one and decrement *Cerr*. **Specification Ref:** 4.10.3, 4.12.1.2 **Test Description:** TD.3.26
2.2.137	The host controller will move the *SplitXState* from **Do Complete Split** (1b) to **Do Start Split** (0b) when it receives an Ack, Nak, or Data(0,1) response to a complete-split token. This transition occurs even if the DataX is not the expected X (e.g. data toggle mismatch). **Specification Ref:** 4.12.1.2 **Test Description:** TD.3.10
2.2.138	The host controller will keep a Qhead's *SplitXState* in the **Do Complete Split** state when it receives a Nyet handshake response to a complete-split token. **Specification Ref:** 4.12.1.2 **Test Description:** TD.3.10

Execution (LS/FS-Interrupt)

2.2.139	The host controller will not expect a handshake for the start-split bus transaction for an interrupt endpoint (e.g. *EPS* $\neq$ 10b and *S-mask* $\neq$ 0). After issuing the start-split bus transaction, the host controller will:
	• set *Frame Tag* based on:
	  • if *FRINDEX*[2:0] is equal to 6, use *FRINDEX*[7:3]+1, or
	  • use the value of *FRINDEX*[7:3]
	• set *C-prog-mask* to zero, and
	• set *SplitXState* to **Do Complete Split** (1b), and
	• if the *PID_Code* is an OUT (00b), then set *S-bytes* equal to zero.
	
	Note that it is a software programming error to put a start-split for an interrupt endpoint in micro-frame 7.
	
	Specification Ref: 4.12.2.3.1
	
	Test Description: TD.3.22

2.2.140	The host controller will issue a complete-split transaction for an interrupt queue head if the high-speed interrupt criteria are met, plus the following additional criteria: • *SplitXState* is **Do Complete Split** (1b), and • The value of *FRINDEX*[7:3] equals the value of *FrameTag*, and • The check of *C-prog-mask* succeeds. **Specification Ref:** 4.12.2.3.1 **Test Description:** TD.3.22

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num **Assertion**

2.2.141 The host controller will set *Active* to zero, *Missed Micro-frame* to a one and *Halted* to a one when the host controller detects that it missed a micro-frame for a Low- or Full-speed Interrupt-IN complete-split. A missed micro-frame is detected when any of the following conditions are TRUE:

- *FRINDEX*[7:3] does not equal the value of *FrameTag*, or
- The check of *C-prog-mask* fails, or
- *S-mask* has a one in the bit position selected by *FRINDEX*[2:0] (e.g. a start-split was scheduled for this micro-frame).

Specification Ref: 4.12.2.3.1

Test Description: Not-Tested

2.2.142 The host controller will transition *SplitXState* to **Do Start Split** and set *Missed Micro-frame* to a one when the host controller detects that it missed a micro-frame for a Low- or Full-speed Interrupt-OUT complete-split. for an OUT. A missed micro-frame is detected when any of the following conditions are TRUE:

- *FRINDEX*[7:3] does not equal the value of *FrameTag*, or
- The check of *C-prog-mask* fails.
- *S-mask* has a one in the bit position selected by *FRINDEX*[2:0] (e.g. a start-split was scheduled for this micro-frame).

Note, that if there are additional complete-splits scheduled for this frame, the host controller must not execute them.

Specification Ref: 4.12.2.3.2

Test Description: Not-Tested

2.2.143 The host controller will update the following fields in an Interrupt Qhead with the specified data after a complete-split bus transaction that does not result in an immediate retry, *Active* being set to zero, or *SplitXState* being set to **Do Start Split** (0b):

- *FrameTag* is set to the value of *FRINDEX*[7:3] when *FRINDEX*[2:0] is not equal to 7, otherwise it is set to *FRINDEX*[7:3]+1, and
- The bit position corresponding to *FRINDEX*[2:0] in *C-prog-mask* is set to a one.

Note that these fields are in addition to the transfer state updates, etc. that always occur at the completion of a bus transaction. This data is committed to the Qhead in memory when the transfer state, status byte, etc. are written.

Specification Ref: 4.12.2.3.2, 4.12.2.3.3

Test Description: TD.3.22

2.2.144 If the device returns a Nyet handshake response to a complete-split bus transaction (for an Interrupt Qhead), and this is the last scheduled complete-split, the host controller will:

- set *XactErr* to a one, and
- decrement *Cerr,* and
- set *SplitXState* to **Do Start Split** (0b),
- commit the results to the Qhead.

Specification Ref: 4.12.2.3.2

Test Description: TD.3.25

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num	Assertion
2.2.145	If the device returns a Nyet handshake response to a complete-split bus transaction (for an Interrupt Qhead), and this is not the last scheduled complete-split, the host controller will: • update *C-prog-mask* and *FrameTag* as appropriate, and • commit the results to the Qhead. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.22
2.2.146	The host controller will retry an Interrupt Qhead complete-split bus transaction IMMEDIATELY if the bus transaction completes with a transaction error that does not result in a Halt condition (and there is time remaining in the micro-frame). When such an error occurs, the host controller will set *XactErr* to a one and decrement *Cerr*. Note that if *Cerr* decrements to zero, that is a halt condition. The results are committed to the Qhead. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.25
2.2.147	If the device returns an Ack handshake in response to an OUT complete-split bus transaction for an Interrupt Qhead, the host controller will advance the transfer state and transition *SplitXState* to **Do Start Split**. The results are committed to the Qhead. Note the advancement of the transfer state may cause the completion of the transfer, which will result in the commit of the overlay to the source qTD. Note, that if there are additional complete-splits scheduled for this frame, the host controller must not execute them as the final transaction state has been received. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.22
2.2.148	If the device returns an Mdata PID in response to an IN complete-split bus transaction for an Interrupt Qhead, the host controller will accept the data bytes (assuming CRC is good) and increment *S-bytes* by the number of bytes received. The results are committed to the Qhead. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.22
2.2.149	If the device returns a Data(0,1) PID in response to an IN complete-split bus transaction for an Interrupt Qhead, the host controller will accept the data bytes if the Data PID matches the data toggle and the CRC is good. The state of the transfer is advanced to by the count of received bytes accumulated in the field *S-bytes*. The results are committed to the Qhead. Note, that if there are additional complete-splits scheduled for this frame, the host controller must not execute them as the final transaction state has been received. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.22

Queue Management Data Structures (qTD, Qhead)– Sections 3.5, 3.6

Num	Assertion
2.2.150	If the device returns the opposite data PID, in response to an IN complete-split bus transaction for an Interrupt Qhead, the host controller must transition *SplitXState* to **Do Start Split** (0b) and ensure the transfer state, including *Total Bytes to Transfer*, *DT*, *C_Page* and *Current Offset* are reset as appropriate. The results are committed to the Qhead. Note: need to make sure this works with a sequence with MDATA(payload), DATA(0,1), a with a page cross, where the data PID does not match the expected. Note, that if there are additional complete-splits scheduled for this frame, the host controller must not execute them as the final transaction state has been received. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.25
2.2.151	If the device returns a Nak handshake in response to an IN or OUT complete-split bus transaction for an Interrupt Qhead, the host controller will: • not advance the transfer state, and • transition *SplitXState* to **Do Start Split** (0b), • commit the resultant state to the Qhead. Note, that if there are additional complete-splits scheduled for this frame, the host controller must not execute them as the final transaction state has been received. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.22
2.2.152	If the device returns a ERR handshake in response to an IN or OUT complete-split bus transaction for an Interrupt Qhead, the host controller will: • set the *ERR* status bit to a one, and • decrement *Cerr*, and • transition *SplitXState* to **Do Start Split** (0b), and • commit the resultant state to the Qhead. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.25
2.2.153	If the device returns a STALL handshake in response to an IN or OUT complete-split bus transaction for an Interrupt Qhead, the host controller will: • set the *Active* status bit to a zero, and • set the *Halted* bit to a one, and • commit the resultant state to the Qhead. This halt condition results in committing the Qhead overlay to the source qTD. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.25

2.3 Split Transactions (Detailed Protocol)

These assertions are derived from Chapter 11 of the USB Specification, Revision 2.0.

Note that the retry of bus transactions is covered with assertions in Section 2.2.5. There is a bit of redundancy here, but we can remove later if necessary.

These assertions are all targeted common requirements for all low/full-speed, asynchronous and interrupt endpoint types. So, the first table covers all STALL, ACK, NAK and NYET responses. The Periodic table

covers additional responsibilities specific to interrupt complete-split bus transactions and isochronous split transactions.

Split Transactions Detailed Protocol — Asynchronous/Interrupt
EHCI Section 4.12.1 through 4.12.2
Core Spec. Sections 11.17.1, 11.17.2, 11.20.1 through 11.21.2

Num	Assertion
2.3.1	The host controller will traverse the asynchronous schedule after executing a start-split bus transaction, before executing a retry for the start-split or the first complete-split bus transaction. **Specification Ref:** 4.10 **Test Description:** TD.4.3
2.3.2	The host controller will issue immediate retries of complete-split bus transactions that encounter transaction errors. The number of tries is bounded by the initial value of *Cerr*. **Specification Ref:** 4.12.1.2, 4.12.2.3.2 **Test Description:** TD.3.26
2.3.3	The host controller will issue a complete-split transaction when it detects a good *Ack* handshake in response to a start-split bus transaction for the Qhead. This applies uniformly to OUT/SETUP and IN split transactions. **Specification Ref:** 4.12.1.1, 4.12.2.3.1 **Test Description:** TD.3.10, TD.3.22
2.3.4	The host controller will retry a start-split transaction (not necessarily immediately) when it detects a good *Nak* handshake in response to a start-split bus transaction for a Qhead. This applies uniformly to OUT/SETUP and IN split transactions. **Specification Ref:** 4.12.1.1, 4.12.2.3.1 **Test Description:** TD.3.10, TD.3.22
2.3.5	The host controller will retry a compete-split bus transaction (not necessarily immediately) when it detects a good *Nyet* handshake in response to a complete-split bus transaction for a Qhead. This applies uniformly to OUT/SETUP and IN split transactions. **Specification Ref:** 4.12.1.2, 4.12.2.3.2 **Test Description:** TD.3.10, TD.3.22
2.3.6	The host controller will advance the transfer state when it detects a good Ack handshake in response to a complete-split bus transaction for an OUT/SETUP split-transaction. This means, as a result of the Ack handshake, the next bus transaction to the endpoint will be a start-split bus transaction with a dif and only iferent data PID and data payload as the previous start-split bust transaction. **Specification Ref:** 4.12.1.2, 4.12.2.3.2 **Test Description:** TD.3.10, TD.3.22
2.3.7	The host controller will advance the transfer state when it detects a good data packet (e.g. data pid matches and good CRC) in response to a complete-split bus transaction for an IN split-transaction. This means, as a result of the Data packet response, the next bus-transaction to the endpoint will be a start-split bus transaction with a dif and only iferent data PID and data payload as the previous start-split bust transaction. **Specification Ref:** 4.12.1.2, 4.12.2.3.2 **Test Description:** TD.3.10, TD.3.22

Split Transactions Detailed Protocol — Asynchronous/Interrupt
EHCI Section 4.12.1 through 4.12.2
Core Spec. Sections 11.17.1, 11.17.2, 11.20.1 through 11.21.2

Num	Assertion
2.3.8	The host controller will retry a split transaction (not necessarily immediately) if it detects a good *Nak* handshake in response to a complete-split bus transaction. This means, as a result of the Nak handshake, the next start-split OUT bus transaction will have the same data PID and data payload as the previous start-split bust transaction. **Specification Ref:** 4.12.1.2, 4.12.2.3.2 **Test Description:** TD.3.10, TD.3.22
2.3.9	The host controller will stop executing split transactions to an endpoint (e.g. halt the Qhead) when it detects a good *Stall* handshake in response to a complete-split bus transaction. This applies uniformly to OUT/SETUP and IN split transactions. **Specification Ref:** 4.12.1.2, 4.12.2.3.2 **Test Description:** TD.3.25
2.3.10	The host controller will halt the endpoint (Qhead) if any start-split bus transaction encounters three consecutive transaction errors (assuming *Cerr* is set to 3). **Specification Ref:** 4.12.1.1, 4.12.2.3.1 **Test Description:** TD.3.25, TD.3.26
2.3.11	The host controller will halt the endpoint (Qhead) if any complete-split bus transaction encounters three consecutive transaction errors (assuming *Cerr* is set to 3). **Specification Ref:** 4.12.1.2, 4.12.2.3.2 **Test Description:** TD.3.25, TD.3.26
2.3.12	The host controller will not execute complete-split bus transactions to a low/full-speed endpoint (interrupt or isochronous) during the remainder of a frame, once it has received a good, non-Nyet response to a complete-split bus transaction. **Specification Ref:** 4.12.1.2, 4.12.2.3.2 **Test Description:** TD.3.10, TD.3.22

These assertions apply to both LS/FS interrupt and FS isochronous unless specifically annotated dif and only iferently.

Split Transactions Detailed Protocol — Periodic
EHCI Sections 4.12.2 through 4.12.3
Core Spec. Sections11.20.1 through 11.21.2

Num	Assertion
2.3.13	The host controller will issue one start-split bus transaction for a full/low-speed interrupt endpoint, then proceed to execute complete-split bus transactions as scheduled by software, without receiving a handshake from the device on the start-split. **Specification Ref:** 4.12.2.3.1 **Test Description:** TD.3.22
2.3.14	The host controller will retry a split transaction for a LS/FS, interrupt IN or OUT (at next appropriate poll rate) if it detects a good *ERR* handshake in response to a complete-split bus transaction. This means, as a result of the ERR handshake, the next start-split OUT bus transaction will have the same data PID and data payload as the previous start-split bust transaction. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.25

Split Transactions Detailed Protocol — Periodic
EHCI Sections 4.12.2 through 4.12.3
Core Spec. Sections11.20.1 through 11.21.2

Num	Assertion
2.3.15	The host controller must support an MDATA response, where the data payload completely fills the host buffer. In this situation, the host must proceed with another complete-split bus transaction (at the next appropriate micro-frame) and accept a zero-length DATA0/1. **Specification Ref:** 4.12.2.3.2 **Test Description:** TD.3.22
2.3.16	The host controller must correctly sequence the *payload continuation encoding* in the *S* and *U* fields of the start-split extended token across the start-split bus transactions required to move the full-speed data to the transaction translator (for full-speed isochronous). **Specification Ref:** 4.12.3.3.1 **Test Description:** TD.3.23
2.3.17	The host controller will issue immediate retries of isochronous complete-split bus transactions that encounter transaction errors. The maximum number of tries of an isochronous complete-split bus transaction is three. **Specification Ref:** 4.12.3.3.2 **Test Description:** TD.3.24
2.3.18	The host controller will advance the transfer state when it detects a good data packet (e.g. data pid matches and good CRC) in response to a complete-split bus transaction for an IN split-transaction. **Specification Ref:** 4.12.2.3.2, 4.12.3.3.2 **Test Description:** TD.3.22, TD.3.23

2.4 Singleton Data Streams

All of the assertions in this section are relevant to maintaining singleton data streams.

2.4.1 Interrupts

This sections lists assertions for all types of interrupt sources.

2.4.1.1 Transfer interrupts (Interrupt Threshold)

Each of the following assertions must be tested with each possible setting of the interrupt threshold. A pre-condition for the following assertions is that USBINTR registers *USB Interrupt Enable* and *USB Error Interrupt Enable* bits are a one. A Transfer Interrupt is any event that sets the USBSTS register *USB Interrupt* bit to a one. A Transfer Error Interrupt is any event that sets the USBSTS register *USB Error Interrupt* bit to a one.

Host Controller Transfer-related Interrupts

Num	Assertion
2.4.1	The host controller must generate transfer-related interrupts no more often than the interrupt threshold programmed into the USBCMD register *Interrupt Threshold Control* field. **Specification Ref:** 4.15 **Test Description:** TD.3.8

Host Controller Transfer-related Interrupts

Num	Assertion
2.4.2	The host controller will generate a transfer interrupt when it completes a transfer for a Qhead where the *IOC* bit is a one. **Specification Ref:** 4.15.1.2 **Test Description:** TD.3.8
2.4.3	The host controller will generate a transfer interrupt when it completes a transfer for a siTD where the *IOC* bit is a one. **Specification Ref:** 4.15.1.2 **Test Description:** TD.3.8
2.4.4	The host controller will generate a transfer interrupt when it completes a transfer for an iTD transaction control/status record where the record's *IOC* bit is a one. **Specification Ref:** 4.15.1.2 **Test Description:** TD.3.8
2.4.5	The host controller will generate a transfer interrupt when it encounters a short-packet condition on a Qhead (regardless of whether the *IOC* bit is set in the Qhead instance). **Specification Ref:** 4.15.1.3 **Test Description:** TD.3.21
2.4.6	The host controller will generate a transfer error interrupt when it encounters a halt condition for a Qhead. Halt events include: • a STALL handshake, or • *Cerr* transitioning to zero after a transaction, or • a packet babble is detected during a transfer **Specification Ref:** 4.15.1.1 **Test Description:** TD.3.9

2.4.1.2 Host Controller Event Interrupts

These cover all non-transfer related interrupt sources (i.e. are not subject to deferral to next interrupt threshold).

Host Controller non-Transfer-related Interrupts

Num	Assertion
2.4.7	The host controller must generate an interrupt if it sets the USBSTS register *Port Change Detect* bit to a one and the USBINTR register *Port Change Interrupt Enable* bit is a one. **Specification Ref:** 4.15.2.1 **Test Description:** TD.2.2
2.4.8	The host controller must generate an interrupt if it sets the USBSTS register *Frame List Rollover* bit to a one and the USBINTR register *Frame List Rollover Enable* bit is a one. **Specification Ref:** 4.15.2.1 **Test Description:** TD.1.8
2.4.9	The host controller must generate an interrupt if it sets the USBSTS register *Interrupt on Async Advance* bit to a one and the USBINTR register *Interrupt on Async Advance Enable* bit is a one. **Specification Ref:** 4.15.2.1 **Test Description:** TD.3.5

Host Controller non-Transfer-related Interrupts

Num	Assertion
2.4.10	The host controller must generate an interrupt if it sets the USBSTS register *Host System Error* bit to a one and the USBINTR register *Host System Error Enable* bit is a one. **Specification Ref:** 4.15.2.1 **Test Description:** Not-Tested

2.4.2 Data Integrity

Do for all transfer types except for control at any endpoint speed.

Data Integrity Assertions

Num	Assertion
2.4.11	The host controller must not corrupt data moving from host system memory to any USB device. **Specification Ref:** USB2.0 – Section 4.5 **Test Description:** TD.3.10, TD.3.12, TD.3.14
2.4.12	The host controller must not corrupt data moving from any USB device into host system memory. **Specification Ref:** USB2.0 – Section 4.5 **Test Description:** TD.3.10, TD.3.12, TD.3.14

2.4.3 Short Packets

Short Packet Assertions

Num	Assertion
2.4.13	The host controller will accept all data received and retire a qTD (transfer) when it receives a good, short data packet on any byte boundary. Note that a short packet in the context of a split transaction is not evaluated until after a DATA(0,1) pid has been received. **Specification Ref:** 4.10.3 **Test Description:** TD.3.21

2.4.4 General Host Controller Data Integrity

General Host Controller Interaction Assertions

Num	Assertion
2.4.14	A host controller must not write outside of the data buffer described by the iTD, siTD, or Qhead. This basically means it cannot write before the beginning of the buffer, nor beyond the end of the buffer. **Specification Ref:** 3 **Test Description:** TD.1.1
2.4.15	The host controller must not write outside of the interface data structure during schedule data structure write-backs. This means the host controller is not allowed to write before or after the data structure instance when performing the commit of transaction state to the schedule data structure. **Specification Ref:** 3 **Test Description:** TD.1.1

General Host Controller Interaction Assertions

Num	Assertion
2.4.16	The host controller will not start a bus transaction unless there is time enough in the micro-frame to complete the transaction on the high-speed bus. This assertion applies to all transfer types and speeds. **Specification Ref:** 4.4.1 **Test Description:** TD.3.17

2.5 Multiple Stream Interoperability

General Host Controller Interaction Assertions

Num	Assertion
2.5.1	The host controller must correctly transition from the *end* of the periodic schedule, to the asynchronous schedule regardless of the data structure type marking the end of the periodic schedule. **Specification Ref:** 4.4 **Test Description:** TD.4.1
2.5.2	The host controller must correctly process any valid periodic schedule. This translates that the host controller must properly process all combinations of iTD, siTD and Qhead data structure linkage (including all combinations of device speeds). **Specification Ref:** 4.6 **Test Description:** TD.4.2
2.5.3	The host controller must correctly process any valid asynchronous schedule. This translates that the host control must properly process all combinations of Qheads, with any combination sequence of device speed, PID encoding and transfer size. **Specification Ref:** 4.8 **Test Description:** TD.4.3
2.5.4	The host controller must correctly process all valid combinations of periodic and asynchronous schedules running in unison. **Specification Ref:** 4.4, 4.6, 4.8 **Test Description:** TD.4.4

2.6 Debug Port

These assertions are only valid or host controllers that implement debug ports.

PCI Configuration Registers (Debug Port Capability) – Appendix C

Num	Assertion
2.6.1	A host controller must report the debug port capability (0x0A) in the PCI configuration space capability field. **Specification Ref:** Appendix C **Test Description:** TD.7.1
2.6.2	A host controller must report the debug port capability (0x0A) ONLY ONCE in the PCI configuration space capability field. **Specification Ref:** Appendix C **Test Description:** TD.7.1

PCI Configuration Registers (Debug Port Capability) – Appendix C

Num	Assertion
2.6.3	The debug port capability register in the PCI configuration space must be a read only register. **Specification Ref:** Appendix C **Test Description:** TD.7.1
2.6.4	The debug port offset must be dword aligned. **Specification Ref:** Appendix C **Test Description:** TD.7.1
2.6.5	The debug port BAR# must have a value between 0x01 and 0x06. **Specification Ref:** Appendix C **Test Description:** TD.7.1
2.6.6	The debug port BAR# must reference a memory BAR. **Specification Ref:** Appendix C **Test Description:** TD.7.1

Host Controller Capability Register (Structural Parameters) – Section 2.2.3

2.6.7	The debug port number field cannot be 0x00 and must be less than or equal to the value in the N_PORTS field. **Specification Ref:** Section 2.2.3 **Test Description:** TD.7.2

Host Controller Registers (Debug port Registers) – Appendix C.3

2.6.8	After a host controller reset the debug port Control/Status Register must set to its default value (0x0). **Specification Ref:** Appendix C.3.1 **Test Description:** TD.7.3
2.6.9	A write to the read only fields in the Control/Status register must have no effect on the value of those fields. **Specification Ref:** Appendix C.3.1 **Test Description:** TD.7.3
2.6.10	Software should be able to set the *Enabled* field in the Control/Status register if and only if there is a high speed device connected to the debug port. **Specification Ref:** Appendix C.3.1 **Test Description:** TD.7.4
2.6.11	Software must be able to Set/Clear all Read/Write fields in the Control/Status register. **Specification Ref:** Appendix C.3.1 **Test Description:** TD.7.3
2.6.12	A write to the read only fields in the USB PID register must have no effect on the value of those fields. **Specification Ref:** Appendix C.3.2 **Test Description:** TD.7.3
2.6.13	Software must be able to Set/Clear all Read/Write fields in the USB PID register. **Specification Ref:** Appendix C.3.2 **Test Description:** TD.7.3

PCI Configuration Registers (Debug Port Capability) – Appendix C

Num	Assertion
2.6.14	Software must be able to Set/Clear all Read/Write fields in the Data Buffer registers. **Specification Ref:** Appendix C.3.3 **Test Description:** TD.7.3
2.6.15	After a host controller reset the debug port Device Address Register must set to its default value (0x7f01). **Specification Ref:** Appendix C.3.4 **Test Description:** TD.7.3
2.6.16	A write to the read only fields in the Device Address register must have no effect on the value of those fields. **Specification Ref:** Appendix C.3.4 **Test Description:** TD.7.3
2.6.17	Software must be able to Set/Clear all Read/Write fields in the Device Address register. **Specification Ref:** Appendix C.3.4 **Test Description:** TD.7.3

Debug Port Operation (Data Streaming) – Appendix C.4

2.6.18	A host controller must be able to send and receive data to and from a debug device connected to the debug port when both the *RunStop* field (USBCMD Register) and the *PortEnable* field (Port Status and Control Register) are set. **Specification Ref:** Appendix C.4 **Test Description:** TD.7.5
2.6.19	A host controller must be able to send and receive data to and from a debug device connected to the debug port when the *RunStop* field (USBCMD Register) is clear and the *PortEnable* field (Port Status and Control Register) is set. **Specification Ref:** Appendix C.4 **Test Description:** TD.7.5
2.6.20	A host controller must be able to send and receive data to and from a debug device connected to the debug port when the *RunStop* field (USBCMD Register) is set and the *PortEnable* field (Port Status and Control Register) is clear. **Specification Ref:** Appendix C.4 **Test Description:** TD.7.5
2.6.21	A host controller must be able to send and receive data to and from a debug device when both the *RunStop* field (USBCMD Register) and the *PortEnable* field (Port Status and Control Register) are clear **Specification Ref:** Appendix C.4 **Test Description:** TD.7.5

Debug Port Operation (Error Test) – Appendix C.4

2.6.22	A host controller must set the *Error* field in the Control/Status register when a transaction error occurs. In addition it must set the *Exception* field to 0x01. **Specification Ref:** Appendix C.4 **Test Description:** TD.7.6
2.6.23	A host controller must set the *Error* field in the Control/Status register when a transaction hardware error occurs. In addition it must set the *Exception* field to 0x02. **Specification Ref:** Appendix C.4 **Test Description:** TD.7.6

2.7 IBIT and FSTN Tests

Test Inactive on next transaction (i) bit – Section 4.12.2.5

Num	Assertion
2.7.1	When a host controller encounters an Interrupt Queue head with the *iBIt* set, then it must ONLY reset the *Active* bit if the *Active* bit was set and the *SplitXState* is DoStart. **Specification Ref:** Section 4.12.2.5 **Test Description:** TD.8.1

FSTN (Operational Model) – Section 4.12.2.2

2.7.2	The host controller should follow the *Normal Path Link Pointer* when it encounters an FSTN in micro-frames 2 through 7. **Specification Ref:** Section 4.12.2.2 **Test Description:** TD.9.1, TD.9.2
2.7.3	When the host controller encounters a *Save-Place* FSTN in micro-frames 0 or 1, then it must start executing in *Recovery Path* mode. **Specification Ref:** Section 4.12.2.2 **Test Description:** TD.9.1, TD.9.2
2.7.4	When in the *Recovery Path* the host controller should always follow the *Normal Path Link Pointer* when it encounters an FSTN that is a *Save-Place* indicator. **Specification Ref:** Section 4.12.2.2 **Test Description:** TD.9.1, TD.9.2
2.7.5	When in the *Recovery Path* the host controller should not process any siTDs or iTDs or a Queue head that indicates that is speed is high speed. **Specification Ref:** Section 4.12.2.2 **Test Description:** TD.9.1, TD.9.2
2.7.6	When in the *Recovery Path* the host controller should not process any siTDs or iTDs or a Queue head that indicates that is speed is high speed. **Specification Ref:** Section 4.12.2.2 **Test Description:** TD.9.1, TD.9.2
2.7.7	When in the *Recovery Path* the host controller should only process any Queue Heads that have its *SplitXState* set to DoComplete. **Specification Ref:** Section 4.12.2.2 **Test Description:** TD.9.1, TD.9.2
2.7.8	The host controller should stop traversing the *Recovery Path* when it encounters an FSTN that is a *Restore* indicator. **Specification Ref:** Section 4.12.2.2 **Test Description:** TD.9.1, TD.9.2

3. Test Descriptions

For all tests defined, the host controller is properly initialized by either the standard Microsoft bus driver or by the Intel EHCI compliance driver. This typically means that (minimally) the USBCMD register CONFIG FLAG field is set to a one and the USBCMD register *Run/Stop* bit is set to a one.

Test Interface Integrity

TD.1.1 This is not an independent test, but a component of the testing strategy. The host controller must preserve interface integrity by not writing beyond those areas defined in the specification (Chapter 3) or beyond the end of data buffers provided by the applications.

All data structures allocated by the driver will be bracketed with a known data pattern. The driver will validate the integrity of the interface by checking the boundaries of the data structures (at least) when they are deallocated.

Also, any data buffers allocated by the bus driver or application must also have the upper and lower bounds of the expected transfer space initialized to know data patterns. The buffers bounds are checked after each IN and OUT transfer is completed.

Test fails if the driver or application detects that the host controller has corrupted memory space around the data buffer or data structures.

Required Device Resource: LS, FS and HS compliance device
Using: EHCI Compliance Stack

64-bit vs 32-bit data structures

TD.1.2 This is not an independent test, but a component of the testing strategy. The compliance tests will use 64-bit data structures for its special purpose HC Driver.

The 32-bit host controllers must ignore the 64-bit extensions and the 64-bit host controllers should properly manages the 64-bit data structures.

Test fails if the driver or application detects that the host controller has corrupted memory space beyond the exported interface capabilities.

Required Device Resource: LS, FS and HS compliance device
Using: EHCI Compliance Stack

3.1 Host Controller Parameters (Initialization, etc.)

Test CONFIGFLAG

TD.1.3 This test verifies the CONFIGFLAG feature works as intended.

This test is performed on a machine with a standard Windows OS with the EHCI Compliance Stack loaded on the EHCI component. The test clears the CONFIGFLAG and asks the tester to attach the high speed compliance device to the each of the ports in turn. The tester is asked to verify that the device enumerates as a full speed device.

Test fails if the device is enumerated as a high-speed device or not enumerated at all.

The test then returns ownership of the port to the EHCI component.

Test fails if the Port ownership did not correctly return on transition of CONFIGFLAG.

Required Device Resource: HS compliance device
Using: Standard OHCI/UHCI Microsoft driver and EHCI Compliance Stack

Test Memory-mapped I/O and PCI Configuration Space Registers Default Values

TD.1.4 This test verifies that all memory mapped I/O and configuration space registers default to the correct values after reset.

All devices are disconnected from all externally available ports, then the controller is reset via the *HCReset* bit in the USBCMD register. When the reset is complete, all valid I/O registers are read and checked for proper values. If the host controller support port power control, then the host controller checks the PORTSC registers before and after turning port power on.

Test fails if reserved fields do not have zeros. Test also fails if operational register fields are not at the appropriate default values as specified from the parameter registers.

Required Device Resource: none
Using: EHCI Compliance Stack

Test USBCMD Register

TD.1.5 This test verifies that all of the non-transfer related features of the USBCMD register function properly, specifically the host controller reset and run/stop.

The host controller is reset via the HCReset bit and the i/o registers read and checked to ensure they are at their default values. The run/stop bit is set to a one and zero. Each setting should result in the USBSTS register HCHalted bit transitioning to a zero or one as appropriate.

Test fails if the i/o registers do not report their default values after the HCReset, or the HCHalted bit does not transition appropriately with the run/stop bit transitions.

Note that this test may be used as a method to transition from one compliance test to the next, so could be run many times during testing.

Note that interactions with the companion controller should be minimized, or at least taken into consideration when running this test. For example, if measuring resets on downstream ports during HCRESET, we must ensure the companion controllers are not driving reset at the same time.

Required Device Resource: none
Using: EHCI Compliance Stack

Test Port Routing

TD.1.6 This test verifies the reported port-routing strategy is correctly implemented.

The EHCI host controller is queried for the number of companion host controllers, the number of ports, and the mapping strategy of ports to companion controllers. A full-speed compliance device is attached to port and enumerated. The system is checked to ensure the device is associated with the expected host controller (e.g. is consistent with the reported mapping strategy). The test is repeated on each externally available port.

The test fails if the full-speed device gets associated with the wrong companion controller.

Required Device Resource: FS compliance device
Using: Standard Microsoft Driver

Test Periodic Schedule Runs Before Asynchronous Schedule

TD.1.7 This test explicitly verifies that for each micro-frame, the host controller executes the periodic schedule before executing the asynchronous schedule.

The test uses a high-speed compliance device with isochronous and bulk endpoints and a USB Bus Analyzer. The test enumerates the device, then disables the periodic and asynchronous schedules via the schedule enables bits in the USBCMD register. The test issues transfers to the isochronous and bulk endpoints, then enables the schedules with a single write to the USBCMD register. The bus analyzer is used to capture the start-up bus traffic.

The test fails if the bus analyzer trace illustrates that the asynchronous schedule is running first. Note that due to allowable variances in implementation on when the host controller may actually enable a schedule, there may be one (or more) initial micro-frames that contain only periodic or asynchronous transactions.

Required Device Resource: HS compliance device
Using: Standard Microsoft Driver

Test Frame List Rollover

TD.1.8 This test verifies that the host controller properly traverses the periodic frame list and verifies the frame list rollover interrupt occurs at the correct time.

The host controller is parameterized to enable frame list roll-over interrupts and the PERIODICLISTBASE register initialized to reference a valid frame list. When testing less-than default size of Periodic Frame List, the compliance driver will add valid periodic transfers to the schedule, outside of the programmed size of the list. The periodic schedule is then enabled via the *Periodic List Enable* bit in the USBCMD register.

The host controller must issue an interrupt each time the host controller FRINDEX register wraps the Periodic Frame List. When the test is completed, the periodic list is disabled via the *Periodic List Enable* bit, and the host controller is stopped.

Note: a programmable Frame List size is an optional implementation feature. The HCCPARAMS field *Programmable Frame List Flag* indicates the implementation.

Test fails if:

- the host controller does not issue the frame list rollover interrupt at the proper time,or
- (for any smaller programmed frame list size), if any transfer descriptors are executed by the host controller, or
- if the frame list rollover interrupts occur when the host controller Run/Stop bit is a zero.

Required Device Resource: none
Using: EHCI Compliance Stack

Test FLADJ Register

TD.1.9 This test verifies that the SOF cycle time increases as we increase the value of FLADJ from 0x00 through 0x3F.

Reset the host controller and set the FLADJ register to zero. Measure the time it takes for the frame list roll over bit to be set. Store the result and repeat the test for incrementing values of FLADJ through 0x3F.

Test fails if the time dif and only iference between successive SOF cycle times does NOT increase by 16 high speed bit times.

Required Device Resource: None
Using: EHCI Compliance Stack

Test 32-bit HC Ignores 64-bit Interface Extensions

TD.1.10 This test verifies that a host controller that supports only the 32-bit interface, ignores the fields in the 64-bit versions of the data structures.

Test sets up an IN transfer of each transfer type, utilizing 64-bit versions of the data structures. It then *randomly* sets bits in the upper address bits of the buffer pointers in the each data structure. The respective schedules are enabled and the host controller is expected to execute the data structures and ignore the 64-bit extension fields. If the host controller operates correctly, the test should be able to observe expected data in the appropriate data buffers. Otherwise the system may get a Host Controller Error, or PCI Access Error.

Test fails if host controller uses the 64-bit extensions and attempts to write data to invalid memory.

Required Device Resource: HS, FS, LS Compliance Devices
Using: EHCI Compliance Stack

Test Port Wake Capability Register

TD.1.11 This test verifies that if the optional PORT WAKE CAPABILITIES register is implemented in PCI configuration space, that bits 1-15 (or at least the appropriately defined number of bits, based on N_PORTs) are writable.

Test reads PCI configuration space for the PORT WAKE CAPABILTIIES register. If the least significant bit is a one, then the test verifies that the next N_PORTS bits can be modified by software.

Test fails if the least significant bit of the register is a one and the appropriate bits are not all writable.

Required Device Resource: None
Using: EHCI Compliance Stack

3.2 Port Operation

These test descriptions cover all the aspects of port operation.

Test Port Indicators

TD.2.1 This test verifies that the port indicator controls (if implemented) function as specified. A host controller indicates whether port indicators are implemented with a one in the HCCPARAMS register *Port Indicators*.

In this test, the *Port Indicator Control* field is cycled through every valid encoding. For each encoding, the tester (person) needs to observe the externally available port indicator is correct. The test needs inform the user which indicator value is being exercised and allow the user to indicate whether the correct indicator was observed.

This test is repeated for all externally available ports.

This test fails if the observed indicator does not match the intended indicator (as perceived by the tester).

Required Device Resource: none
Using: EHCI Compliance Stack

Test Port Change Detect

TD.2.2 This test verifies that all the defined events (with the exception of over-current) cause a host controller to correctly set USBSTS register *Port Change Detect*.

The host controller is initialized and turned on (including powering ports if necessary). Each of the stimulus that must cause the port to register a change are stimulated (with the noted exceptions). For each event, the test checks the USBSTS register and the port under test for correct state. The sequence is repeated on each port.

Stimulus events are:

- HS compliance device *attached, *de-attached
- HS compliance device *resumes (after being suspended for a time).

Test fails if an attempted stimulus does not result in the correct port change bit being set and the USBSTS register *Port Change Detect* bit being set.

*Note that the high-speed compliance device can be commanded to disconnect, reconnect, and resume under program control.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

Test Port Test Modes

TD.2.3 This test verifies that each port Test Mode works as specified.

The host controller is initialized as appropriate, and then a port is put into a test mode and the bus observed via an oscilloscope to determine whether the port is in the selected test mode. The host controller is reset and initialized after each test mode is verified.

The following test modes are verified: {**Test J_State**, **Test K_State**, **Test_Packet**, **Test Force_Enable**, **Test SE0_Nak**}. The test sequence is repeated on each accessible port.

Test fails if a port does not support a test mode.

Required Device Resource: Appropriate Test & Measurement equipment, and optionally, HS Compliance Device
Using: EHCI Compliance Stack

Test Invalid Port Enable

TD.2.4 This test verifies that software cannot enable a port by simply writing a one to the *Port Enable/Disable* control bit.

The test instructs the tester to connect a high speed device capable device to a host controller port. The test then attempts to force enable by simply writing a one to the *Port Enable/Disable* bit in the associated portsc register. The test is repeated for each port.

Test fails if the port enable bit transitions from a zero to a one as a result of writing the *Port Enable/Disable* bit to a one.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

Test Port Disable

TD.2.5 This test verifies that software can disable any enabled port by simply writing a zero to the portsc's *Port Enable/Disable* control bit.

A high-speed device capable device is connected to a host controller port. The port is enumerated as usual. Then the test driver will disable the port by writing a zero to the portsc's *Port Enable/Disable* bit.

Test fails if :

- the port enable bit does not transition from a one to a zero as a result of writing the *Port Enable/Disable* bit to a zero and/or
- the host controller sets the portsc's *Port Enable/Disable Change* bit to a one.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

3.2.1 Enumeration

These tests shall be applied to all externally available ports. Port enumeration includes (at least) a port reset and enable sequence and at least one control transfer to the attached device. The Microsoft standard USB driver set will be used for these tests.

Test Port Routing/Correct Enumeration

TD.2.6 This test verifies that the host controller can properly enumerate LS, FS, and HS devices. This test (by side-effect) also tests all of the port power switching, port reset and enable, and port hand-off sequences.

This test is performed on a machine with a standard Windows OS with the EHCI Compliance Stack loaded on the EHCI component. A LS, FS and HS device is plugged into a host controller port. The device must be properly enumerated as a LS or FS device and a HS device must be enumerated as a FS device when the port hand off is performed.

Test fails if any device is not correctly enumerated (e.g. mapped to the correct host controller).

Required Device Resource: LS, FS, and HS compliance devices
Using: Standard Microsoft Driver and EHCI Compliance Stack

Test No SOFs when HC Halted

TD.2.7 This test verifies that a host controller port will not send SOFs down a port when the host controller is halted (e.g. *HCHalted* bit is a one).

The host controller is initialized and started. A high-speed compliance device is attached an enumerated (leaving the port in the enabled state). The host controller is then halted and restarted (via the *Run*/*Stop* bit in the USBCMD register). The tester is instructed to capture traces and verify that the host controller correctly implements the halted bit. A detailed description of what to look for is documented in the test details that pop up when this test is run.

Test fails if the host controller continues sending SOFs when the halted bit is set.

Required Device Resource: HS compliance devices
Using: EHCI Compliance Stack

3.2.2 Suspend/Resume

These tests shall be applied to all externally available ports. The Intel EHCI Compliance Driver will be used for these tests.

Test Port Suspend/Resume Tests (Host Initiated Resume)

TD.2.8 This test verifies that the host controller properly manages a port-suspend followed by a host initiated port resume. It also validates that adjacent ports are not affected by the suspend and resume sequence.

This requires that all available downstream ports have high-speed compliance devices attached and enumerated. Each compliance device is commanded to NOT be a remote wakeup source. The test then iterates over each available port doing a port-suspend and host-initiated port resume. Each port is tested with all other ports suspended and all other ports not suspended.

Test fails if the port status bits do not match what was expected on each of the ports.

Required Device Resource: HS compliance Device
Using: EHCI Compliance Stack

Test Port Suspend/Resume Tests (Remote-wakeup)

TD.2.9 This test verifies that the host controller properly manages a port-suspend followed by a device-initiated port resume. It also validates that adjacent ports are not affected by the suspend and resume sequence.

This requires that all available downstream ports have high-speed compliance devices attached and enumerated. The test then iterates over each available port doing a port-suspend and waiting for a port resume. Each port is tested with all other ports suspended and all other ports not suspended. Only the device on the port under test is enabled for remote wakeup. When the device observes the suspend event, it waits for the pre-programmed time, then initiates a remote wakeup.

Test fails if a device is the port status bits do not match what was expected on each of the ports. The test also fails if the port under test does not successfully detect the remote wakeup or the remote wakeup does not complete normally via compliance manipulations of the PORTSC.

Required Device Resource: HS compliance Device
Using: EHCI Compliance Stack

Test Port Suspend/Resume Test (Reset a Suspended Port)

TD.2.10 This test verifies that the host controller properly resets a suspended port.

A high-speed compliance device is attached and enumerated on a root port. The port is then suspended by writing a one to the PORTSC *Suspend* bit. The port is then reset (after an appropriate idle time) by writing a zero to the *Port Enabled* bit and a one to the *Port Reset* bit. The remainder of the reset sequence is carried out as normal.

This test is repeated on every externally available host controller port.

The test fails if the port does not successfully reset (including clearing of the *Suspend* bit to zero).

Required Device Resource: HS compliance Device
Using: EHCI Compliance Stack

3.3 Transfers

Test FRINDEX

TD.3.1 This test verifies that the host controller manages the relationship between the FRINDEX register and the SOF value in the SOF token correctly.

This test uses a high-speed isochronous endpoint. The host test software ensures that in each micro-frame, the OUT packet data payload contains the value of the FRINDEX register for that micro-frame that can be read (in decimal) in a trace captured using a bus analyzer. The user is instructed to take a trace of the analyzer and examine the same to ensure that the host controller is performing the right operations. A detailed description of what to look for is documented in the test details that pop up when this test is run. The relationship between the value of the FRINDEX and the SOF value when the device received the packet must be consistent with that defined in the EHCI.

This test fails if the relationship is not consistent.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

Test Micro-frame Integrity

TD.3.2 This test verifies that the host controller correctly implements protection on frame babble.

This test uses as many isochronous OUT endpoints as necessary to move an interrupt IN close to the end of the micro-frame. The interrupt IN will always deliver more data than configured that its maximum packet size (i.e. packet babble). The tester will provide input as to the size of the Isochronous endpoint that will be sufficient to cause a frame babble.

Test fails if the host controller does not detect a frame babble.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

Test Qhead Nak Counter/Empty List Detection

TD.3.3 This test verifies that the host controller correctly implements the Qhead nak counter and empty list detection features (asynchronous list issue only).

This test uses a high-speed IN and a full-speed IN. The high speed endpoint will always deliver data when polled (i.e. never Nak). The test first runs with only the full-speed endpoint. For each value of the NAKCount reload field, the tester is instructed to capture traces and verify that the host controller correctly implements the same. The test is run again with transfers being done on the high speed endpoint as well and again the tester verifies the host controller correctly implements the NAK Count field. A detailed description of what to look for is documented in the test details that pop up when this test is run.

Test fails if:

- the host controller does not temporarily stop servicing the full-speed endpoint after the endpoint has been NYETed a number of times equal to the NAK count reload value.
- the full-speed endpoint is serviced in the same microframe while there is high speed traffic and the endpoint has been NYETed a number of times equal to the NAK count reload value.

Required Device Resource: HS and FS Compliance Device
Using: EHCI Compliance Stack

Test Asynchronous Service Order

TD.3.4 This test verifies that the host controller correctly manages the Asynchronous list in a round-robin order.

This test uses a high-speed loop-back endpoint pair and a full speed loop-back endpoint pair. The test will move data concurrently on both the pairs of pipes. The tester is instructed to capture a bus trace and then verify that the host controller performs the transactions in the correct order. A detailed description of what to look for is documented in the test details that pop up when this test is run.

Test fails if the bus trace indicates an out-of-sequence bus transaction.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

Test Asynchronous List Advance Doorbell

TD.3.5 This test verifies that the Interrupt Async Advance Doorbell works as specified.

This test uses a high-speed loop-back endpoint pair. The doorbell interrupt is enabled and the test waits for the doorbell interrupt to be asserted. The test waits expects that the interrupt will be asserted atleast once in 10ms. If the interrupt does not occur in that time period, then a transfer is performed on the endpoints and again the test checks whether the Async Advance Doorbell interrupt occurred.

Test fails if the Async Advance Doorbell interrupt was not asserted.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

Test PING Protocol Management

TD.3.6 This test verifies that the host controller correctly manages the PING protocol.

The test uses high-speed loop-back endpoint pair. An OUT buffer is allocated that is large enough to fill the compliance device's loop-back endpoint buffer, plus one maximum packet size. The buffer is sent to the OUT endpoint. When the device detects that its buffer is full, it will respond with a NYET handshake. The host should then transition to PING state on the OUT endpoint. The test will delay a bit to ensure the endpoint remains in the PING state (e.g. does not move data). The state of the endpoint is queried to determine whether the endpoint is really in the PING state. A single maximum packet IN is then submitted by the host, which should allow the device to take the final packet of the allocated data buffer and both requests should complete.

Note: compliance device must be able to exercise all methods of getting into and out of PING state, as specified in Table 4-11 of the EHCI specification.

Test fails if OUT request completes before the IN request is issued, or the host controller does not transition the state of the Qhead to the PING state.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

Test PING Protocol Handling (2) – more explicit

TD.3.7 This test more explicitly verifies that the host controller correctly manages the PING protocol.

This test uses an OUT endpoint on the high-speed compliance device, a set of vendor-unique commands to adjust the *response mode* of the OUT endpoint and a USB bus analyzer.

The compliance device is instructed to NAK all OUTs. An OUT is issued to the to the device. Capture a bus trace and observe that the addressed endpoint on the device is NaKing PING tokens.

Test fails if the host controller does not start sending PINGs after the first OUT has been NAKed. The host controller must continue to send PINGs while the device NAKs.

The device is then instructed to ACK. Capture a bus trace and analyze the results.

Test fails if the host controller does not send an OUT after the last PING has been ACKed.

The device is instructed to timeout on subsequent OUTs. Issue another OUT to the device (setting *ErrCnt* to zero). Capture a bus trace and analyze the results.

Test fails if the host controller does not start sending PINGs after the first OUT has timed out. The host controller must continue to send PINGs while the device times out.

The device is instructed to ACK. Capture a bus trace and analyze the results.

Test fails if host does not transition to OUT when it receives an ACK to the PING, or the OUT transfer completes with an error.

The device is instructed to NYET on subsequent OUTs. Capture a bus trace and analyze the results.

Test fails if the host controller does not start sending PINGs after the first OUT has been NYETed.

The device is instructed to ACK and the transfer should complete.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

3.3.1 Interrupts

Test Transfer Interrupts

TD.3.8 This test verifies the transfer interrupts are generated correctly. It also verifies the interrupt threshold is functions properly.

The host controller is initialized, including setting the USBCMD register *Interrupt Threshold Control* field to a specific value. Then all transfer events that must generate a transfer interrupt are stimulated. The appropriate schedule is enabled and disabled with each test instance. For each stimulus, the test verifies the interrupt occurs at the appropriate time.

Interrupt Threshold Control exercised for each valid programming value.

All of the following transfer interrupt conditions are stimulated:

- qTD retirement w/IOC bit set (with and without short packet)
- qTD retirement w/o IOC bit set (wit and without short packet)
- qTD retirement on short packet
- iTD record transfer completion w/IOC bit set to a one
- iTD record transfer completion w/o IOC bit set to a one
- siTD completion w/IOC bit set to a one
- siTD completion w/o IOC bit set to a one

Test fails if the host controller does not issue an interrupt for any required event.

Required Device Resource: FS and HS compliance device
Using: EHCI Compliance Stack

Test Transfer Error Interrupts

TD.3.9 This test verifies the transfer error interrupts are generated correctly. It also verifies the interrupt threshold functions properly.

The host controller is initialized, including setting the USBCMD register *Interrupt Threshold Control* field to a specific value. Transfer errors (timeouts and packet babbles) are injected into transfer streams of each transfer type and endpoint speed. The pipes are initialized with error counters set to 1 and the data error is injected on the first bus transaction. For each stimulus, the test verifies the interrupt (USBERRINT) occurs at the appropriate time.

Interrupt Threshold Control is exercised for each valid programming value.

All of the following transfer error interrupt conditions are stimulated:

- HS, FS synchronous (Timeout and packet babble).
- LS Interrupt (Timeout and packet babble).
- HS Interrupt (Packet babble)

Test fails if the host controller does not issue an interrupt for any required event.

Required Device Resource: FS and HS compliance device
Using: EHCI Compliance Stack

3.3.2 Data Structures

Test Qhead Scatter/Gather (qTD)

TD.3.10 This test verifies that the host controller correctly manages the scatter/gather buffer pointers on all buffer byte boundaries for Qheads.

This test moves data through a bulk or interrupt loop-back pair of endpoints on the selected test device. At least one 20K byte buffer (page-aligned), is allocated for the OUT and IN endpoints (it is acceptable to simultaneously stream multiple buffers on each endpoint during this test). The test selects an offset into the buffer and performs a 16K byte transfer to the selected endpoint. During the test, the offset must be adjusted to all byte boundaries in a 4K range [0, 4095]. The data written on the OUT pipe is read on the IN pipe and checked for the correct data pattern. Note, that is assumed the EHCI Driver always maps a 16K transfer request to one qTD.

This test is exercised on the following device speeds/transfer types: {Interrupt(LS, FS, HS), Asynchronous-bulk (FS, HS)}.

The test fails if the data read does not match the data written.

Required Device Resource: LS, FS and HS compliance device
Using: EHCI Compliance Stack

Test Qhead Transfer Size Boundaries (qTD)

TD.3.11 This test verifies that the host controller correctly manages IN and OUT transfers with various transfer sizes, across all byte boundaries within a memory page, using a Qhead. The test also verifies that the set of transfer sizes work across all byte boundaries of a page cross.

This test moves data through a bulk or interrupt loop-back pair of endpoints on the selected test device. The test selects a transfer size (starting with 1) and sends a known data pattern on the OUT pipe. It then makes a similar request on the IN pipe (similar in terms of buffer alignment and transfer size). The data send must match the data received. The IN and OUT pipes are run synchronous with each other (e.g. an IN is started only when the OUT completes).

The following sets of input parameters are exercised:

Transfer Type/ EP Speed	Max Packet	Transfer Size	Buffer Offset
Interrupt/FS	{8, 32, 64}	{0, 1-64}	{0-4095}
Interrupt/HS	{64, 128, 512}	{0, 1-4096}	{0-4095}
Bulk/FS	{64}	{0, 1-64}	{0-4095}
Bulk/HS	{512}	{0, 1-4096}	{0-4095}
Interrupt/LS	{1, 4, 8}	{0, 1-8}	{0-4095}

The test fails if the data read does not match the data written.

Required Device Resource: FS and HS compliance device
Using: EHCI Compliance Stack

Test iTD Scatter/Gather

TD.3.12 This test verifies that the host controller correctly manages the scatter/gather buffer pointers on all buffer byte boundaries for iTDs. It verifies that every byte-boundary page cross in every page pointer is exercised and verified. This test also verifies the isochronous caching strategy.

This test moves data through an isochronous loop-back pair. The maximum packet size for each endpoint is 1024, the multiplier is 1, and period is 1. A 12K byte buffer is allocated for each millisecond on each endpoint. For each millisecond, the test selects an offset, into the buffer for the start of the transfer, for that millisecond and transfers 8K worth of data. All of the subsequent micro-frame offset descriptions are programmed relative to the beginning offset. During the test, the beginning offset (with appropriate adjustments to other micro-frames), must be adjusted to all byte offsets in the range [0, 4095]. The data written on the OUT pipe is read on the IN pipe and checked for the correct data pattern. The test is done with the starting page offset in the Isochronous TD set to 0, 2 and 4.

The test fails if the data read does not match the data written.

Required Device Resource: HS compliance device
Using: EHCI Compliance Stack

Test iTD Transfer Size Boundaries

TD.3.13 This test verifies that the host controller correctly manages IN and OUT transfers with various transfer sizes, across all byte boundaries within a memory page using an iTD. The test also verifies that the set of transfer sizes work across all byte boundaries of a page cross.

This test moves data through a high-speed isochronous loop-back pair of endpoints on the selected test device. The test selects a transfer size (starting with 1) and sends a known data pattern on the OUT pipe. It then makes a similar request on the IN pipe (similar in terms of buffer alignment and transfer size). The test needs to make sure that it starts the test on a page boundary.

The following sets of input parameters are exercised:

- Maximum Packet Size: {8, 512, 1024}
- Transfer Size: {1, 2, 3, 5, 7, 11, 13, 63, 65, 127, 129, 511, 513, 1023, 1025, 4096}
- Buffer offset: {0-4095}

The test fails if the data read does not match the data written.

Required Device Resource: FS and HS compliance device
Using: EHCI Compliance Stack

Test siTD Scatter/Gather

TD.3.14 This test verifies that the host controller correctly manages the scatter/gather buffer pointers on all buffer byte boundaries for siTDs. It verifies that every byte-boundary page cross in every page pointer is exercised and verified.

This test moves data through a full-speed, isochronous loop-back pair. The maximum packet size for each endpoint is 513 and the period is 1. An 8K byte buffer is allocated for each millisecond on each endpoint. The nearest (to the top of the buffer) page boundary is located (which cannot be the top of the buffer). The first buffer offset for the millisecond is selected as the page boundary offset minus one maximum packet (513) minus 1. During the test, the offset must be adjusted to all byte offsets in the range [starting offset, 514]. The data written on the OUT pipe is read on the IN pipe and checked for the correct data pattern.

The test fails if the data read does not match the data written.

Required Device Resource: FS compliance device
Using: EHCI Compliance Stack

Test siTD Transfer Size Boundaries

TD.3.15 This test verifies that the host controller correctly manages IN and OUT transfers with various transfer sizes, across all byte boundaries within a memory page using an siTD. The test also verifies that the set of transfer sizes work across all byte boundaries of a page cross.

This test moves data through a full-speed isochronous loop-back pair of endpoints on the selected test device. The test selects a transfer size (starting with 1) and sends a known data pattern on the OUT pipe. It then makes a similar request on the IN pipe (similar in terms of buffer alignment and transfer size).

The following sets of input parameters are exercised:

- Maximum Packet Size: {32, 512, 1023}
- Transfer Size: {1, 2, 3, 5, 7, 11, 13, 63, 65, 127, 129, 511, 513, 1023}
- Buffer offset: {0-4095}

The test fails if the data read does not match the data written.

Required Device Resource: FS compliance device
Using: EHCI Compliance Stack

Test Qhead Data Toggle Control (DTC)

TD.3.16 This test verifies that the host controller correctly manages the data toggle control mechanism in a Queue Head (e.g. when the *DTC* bit is a one).

This test uses a high-speed compliance device with a bulk loop-back pair. The test uses a vendor-unique command to initialize the endpoint pair's data toggles to zero. The test then writes sends 4 distinct max packet size transfers ((P0, P1, P2 and P3) to the device with the same data toggle. After all of the OUTs have executed, the test issues an IN request for 4X maximum packet size. The device should return 1×maximum packet size worth of data and the data pattern must be only P0.

The test fails if the IN request returns more than 1×maximum packet size or the data pattern is not P0.

Note: *DTC* == 0 is covered by all of the non-control transfer type tests.

Required Device Resource: FS compliance device
Using: EHCI Compliance Stack

Test Frame Boundary Protection (non-Babble)

TD.3.17 This test verifies that the host controller correctly manages starting only those bus transactions that will fit in the current micro-frame.

This test uses a high-speed compliance device with a *schmoo* endpoint(s), a bulk loop-back pair and a USB bus analyzer. The test uses the *schmoo* endpoint(s) to leave only enough room in each micro-frame for one IN on the bulk IN. Then the asynchronous transactions are *schmoo'd* closer to the micro-frame boundary. The *schmoo* process progresses until it completely shuts down (starves) the loop-back data stream. The test is repeated on a high speed interrupt endpoint pair. In this case, the *schmoo* progresses until both the Interrupt IN and OUT are shut down.

The test fails if the host controller starts an OUT when there is no room left in the micro-frame, or starts an IN which creates a frame babble.

Required Device Resource: FS compliance device
Using: EHCI Compliance Stack

3.3.3 High-bandwidth (Interrupt and Isochronous)

Test High-Bandwidth Interrupt

TD.3.18 This test verifies that the host controller correctly manages a high-bandwidth interrupt data stream. Particularly, whether the correct number bus transactions are executed per micro-frame in the face of buffer boundaries, short packets and transaction errors.

A high-speed loop-back pair of interrupt pipes with a period of 1 is used for this test. The host does not stream data on the loop-back pair concurrently during this test. It sends a fixed amount of data, then expects to read the same data back (depending on particular instance of test and what is being evaluated). The host can use the size of the data buffer(s) (e.g. amount of data written) to exercise many of the boundary conditions. The high-speed compliance device will inject bus transaction timeouts in bus transactions when commanded by the host to do so.

Max Packet/Mult combinations: {1024x3, 1024x2}

Test fails if it is determined that the host controller did not execute the proper number of bus transactions in a micro-frame.

Required Device Resource: HS compliance device
Using: EHCI Compliance Stack

Test High-Bandwidth Isochronous

TD.3.19 This test verifies that the host controller correctly manages a high-bandwidth isochronous data stream (both IN and OUT). The test focus is primarily on whether the host controller correctly manages the bus protocol.

A high-speed, loop-back pair of isochronous high-bandwidth pipes is used for this test. All of the boundary conditions for OUT bus transfers are exercised by adjusting the size and alignment of the data buffer for the host controller. Correct operation is evaluated by checking that all of the data written can be read on the loop-back in Pipe. Similarly, by specifically controlling the amount of data written on the OUT loop-back pipe, all of the boundary conditions can be exercised on the IN loop-back endpoint. The high-speed compliance device can also inject bus transaction timeouts in bus transactions when commanded by the host to do so.

Max Packet/Mult (Logical Packet Size) combinations: {1024x3, 1024x2, 1024x1}

Buffer Size combinations:

Logical Packet ∗ N
(Logical Packet ∗ (N-1)) +(Max Packet ∗ {1, 2})

Test fails if actual received data or transfer results do not match the expected.

Required Device Resource: HS compliance device
Using: EHCI Compliance Stack

Test High-Bandwidth Isochronous

TD.3.20 This test verifies that the host controller correctly manages high-bandwidth isochronous data streams (both IN and OUT) when system hold-offs cause the bus transaction sequence to occur later in the micro-frame than budgeted. (E.g. not enough time in the micro-frame to complete all of the scheduled transfers).

A high-speed *schmoo* endpoint is used to push a high-bandwidth isochronous IN (and, separately an OUT) close to the micro-frame boundary, causing the host controller to not execute portions of the high-bandwidth sequence that cannot complete in the current micro-frame. The *schmoo* endpoint is used to force a 1024x3 endpoint (IN and OUT) to drop the second and third transactions in the sequence. The IN and OUT endpoints in this test are not hardware loop-back endpoints.

Test fails if:

- on an OUT, the amount of data sent is not equal to the expected amount., or,
- on an IN, the amount of data received is not equal to the expected amount.

Required Device Resource: HS compliance device
Using: EHCI Compliance Stack

3.3.4 Short Packets

Test Short Packets

TD.3.21 *Test short packets*. This test verifies the host controller's ability to handle short packets and all of the processing boundary conditions associated with residual buffer, maximum packet size and advancing a Queue after a short packet.

A loop-back pair of interrupt or bulk pipes is used for this test. The input pipe will use 2 qTDs per transfer, depending on the feature being evaluated. The host does not stream data concurrently on both pipes during this test. It sends a fixed amount of data out then expects to read the same amount of data back. The host can use the amount of data written and the size of the input buffer to control where a short packet will occur on the input data stream. Each condition below is exercised with the data buffer starting on a dif and only iferent byte boundary within a DWORD.

Condition	**Event**	**Criteria**
Avail. Buffer ≥ Max Packet	Data < Max Packet	HC should use Alt. Pointer to advance Queue (and generate interrupt)
	Data = Max Packet	HC should use Next Pointer to advance Queue (not generate interrupt)
Avail. Buffer < Max Packet	Data < Avail Buffer	HC should use Alt. Pointer to advance Queue (and generate interrupt)
	Data = Avail Buffer	HC should use Next Pointer to advance Queue (not generate interrupt)

This test is repeated on all device speeds (LS/FS/HS). For a LS and FS interrupt endpoint, each boundary condition is exercised along with *schmoo* endpoint that moves the split transaction across a micro-frame boundary.

Test fails if it is determined that the host controller did not handle the short packet correctly (e.g. returned more data or did not advance the queue correctly). This can be determined by how much data is returned per read.

Required Device Resource: LS/FS/HS Compliance Devices w/Bulk Loop-back
Using: EHCI Compliance Stack

3.3.5 Split Transaction

A *schmoo* pipe is an isochronous OUT that is targeted to no valid device. It is always scheduled first in a frame (uframe) and is used to deterministically push other traffic across micro-frame boundaries by adjusting the number of bytes and number of bitstuffs in the data payload. For example, starting out with small packet size, of all zeros (no bitstuff), then adding a data pattern to introduce 1 bitstuff at a time until 7 bitstuffs have been added. Then change data pattern back to zero, add one byte (and repeat process). This will *schmoo* later traffic by one bit time (for each data pattern/data length change).

Test Split-transaction Interrupt Streaming

TD.3.22 This test verifies the host controllers ability to handle split transaction interrupt IN devices where the returned data may be returned either as a single packet or as two packets (MDATA, DATAx). It also verifies that split Interrupt INs can be scheduled to execute at boundary cases near the end of a frame. The test sets up one *schmoo* endpoint and an interrupt loop-back pair with a period of 32 (one test iteration with each). The *schmoo* followed by IN transactions run contiguously in a frame.

The test steps the endpoint's S-masks and C-masks through all valid combinations (earliest start to latest start). For each selection of S-masks and C-masks the interrupt IN is *schmooed* across the nearest micro-frame boundary. The test is repeated for each device speed and defined packet size.

Device Speed	**Packet Sizes for Test**
LS	{7, 8}
FS	{7, 8, 16, 31, 64}

Test fails if data mis-compares occur.

Required Device Resource: LS and FS Compliance Device w/Interrupt Loopback
Using: EHCI Compliance Stack

Test Split-transaction Isochronous Streaming

TD.3.23 This test verifies the host controller's ability to handle split transaction isochronous IN devices where the returned data may be returned either as a single packet or as multiple packets (MDATA, DATAx). It also verifies that split Isochronous INs can be scheduled to execute at boundary cases near the end of a frame. The test sets up one *schmoo* endpoint and a FS isochronous loop-back pair.

The test steps the endpoint's S-masks and C-masks through all valid combinations (earliest start to latest start). For each selection of S-masks and C-masksm, the test first sends the data to be read back from the device on the OUT endpoint and then the read is *schmooed* across the nearest micro-frame boundary.

Packet sizes for test {93, 95, 187, 189, 450, 950}

Test fails if data mis-compares occur.

Required Device Resource: FS Compliance Device w/Isoch Loop-back
Using: EHCI Compliance Stack

Test Split-transaction Isochronous Streaming (error cases)

TD.3.24 This test verifies that errors in the isochronous data stream, or protocol are detected by the host controller and reported correctly.

This test uses a full-speed Isochronous loop-back pair. The test uses various combinations of transfer sizes along with S and C masks to test the following cases:

- First complete split NYET
- Insufficient number of complete splits to complete the transfer
- Cause a packet babble by returning more data than the Maximum Packet Size
- NYET response to all complete splits

The test is repeated for each microframe excluding bus microframe 6.

Test fails if the host controller does not correctly detect and indicate transfers that fail in the status bits for that transfer.

Required Device Resource: FS Compliance Device w/Isoch Loop-back
Using: EHCI Compliance Stack

Test Split-transaction Interrupt Streaming (error cases)

TD.3.25 This test verifies that errors in an interrupt split-transaction are detected by the host controller and reported correctly.

This test uses a *schmoo* endpoint and a full-speed loop-back pair. The *schmoo* endpoint is used to delay the execution of the loop-back pair on the full-speed link. This test exercises the host controller's ability to detect and handle the following:

- The TT responds to all complete-splits with NYET handshakes. This is exercised by setting the interrupt-IN C-mask short (not enough complete-splits) and using the *schmoo* endpoint to delay execution of the full-speed Interrupt-IN until after all of the complete-splits scheduled have been executed.

  Test fails if host controller does not properly transition state to Do-Start and retries the start-split.

- The split-transaction feature of the high-speed compliance device is used to inject transfer errors and a USB analyzer is used to determine whether the host controller properly detects the errors and issues the appropriate number of retries. The *schmoo* endpoint and adjusting of the S-masks and C-masks near the end of the micro-frame will exercise the case where the host controller cannot execute all retries in the current micro-frame.

  Test fails if host does not issue immediate retries, or does not halt the Qhead if the retries cannot completed.

- The split-transaction feature of the high-speed compliance device is used to inject a PID mismatch (data toggle mismatch).

  Test fails if the host controller accepts the data (advances the transfer) before transitioning to the next split transaction.

- The full-speed device injects transaction errors on the full-speed bus and the TT report ERR handshakes to the host in response to a complete-split.

  Test fails if host does not properly transition the Qhead back to the start-split state and retry the transaction.

- The full-speed device responds with a STALL handshake.

  Test fails if host does not properly transition the Qhead to the Halted state.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

Test Split-transaction Asynchronous Streaming (error cases)

TD.3.26 This test verifies that the host controller correctly manages the asynchronous list when it gets an error on an asynchronous complete-split and cannot get the immediate retry completed in the current micro-frame.

This test uses a *schmoo* endpoint, a high-speed bulk OUT endpoint and a non existent full speed OUT endpoint. The test performs OUTS to both the high speed and full speed endpoint. The full speed OUT will time out and an immeadiate retry should be performed by the host controller. The full speed OUT is *schmooed* until there is not enough time to perform the retry in the current microframe. When this occurs, the host controller should ensure that the retry occurs as the first asynchronous transaction in the next micro-frame..

The tester is instructed to capture a bus trace and then verify that the host controller performs the transactions in the correct order. A detailed description of what to look for is documented in the test details that pop up when this test is run.

Test fails if the bus trace indicates that the host controller did not retry the full speed OUT as the first asynchronous transaction in the next microframe when there was not enough time to retry said transfer in the previous microframe.

Required Device Resource: HS Compliance Device
Using: EHCI Compliance Stack

3.4 Multiple Data Streams (Interoperability)

Test Periodic List -End Of List Mark Detection

TD.4.1 This test verifies that the host controller properly detects the end of the periodic list and makes the correct transition to processing from the asynchronous list.

This test uses loop-back endpoints for full- and high-speed isochronous, high-speed interrupt, and high-speed bulk. The bulk loop-back stream is started (with the asynchronous list disabled). The periodic list is enabled and then each periodic loop-back pair is started. Requirement is that each periodic data structure type (iTD, siTD, and Qhead), must be used somewhere in the schedule as the End-of-List Marker. The other phase of the test is exactly the same data streams, except the asynchronous list is enabled.

Test fails if:

- the asynchronous schedule is disabled, and any data is moved on the bulk loop-back pair, or
- the asynchronous schedule is enabled, and no data is moved on the bulk loop-back pair.

Required Device Resource: FS, HS Compliance Devices
Using: EHCI Compliance Stack

Test Periodic Schedule -Inter-operability

TD.4.2 This test verifies that the host controller can successfully manage combinations of high, full, and low-speed endpoints in the periodic schedule.

This test runs the tests described in TD.3.12, TD.3.13, TD.3.14, TD.3.15, TD.3.18, TD.3.19 and the interrupt-subsets of tests described in TD.3.10 and TD.3.11. These tests are run in all different combinations.

Test fails if any endpoints halt for any reason or there are data check errors on loop-back.

Required Device Resource: LS, FS, HS Compliance Devices
Using: EHCI Compliance Stack

Test Asynchronous Schedule -Inter-operability

TD.4.3 This test verifies that the host controller can successfully manage combinations of high, full, and low-speed endpoints in the asynchronous schedule.

This test runs the asynchronous portions tests described in TD.3.10 and TD.3.11. These tests are run in all different combinations.

Test fails if any endpoints halt for any reason or there are data check errors on loop-back data.

Required Device Resource: LS, FS, HS Compliance Devices
Using: EHCI Compliance Stack

Test EHCI -Inter-operability

TD.4.4 This test verifies that the host controller can successfully manage combinations of high, full, and low-speed endpoints in the periodic and asynchronous schedule.

This test runs the tests described in TD.4.1, TD.4.2 and TD.4.3 in parallel. The tests are run in every different combination.

Test fails if any endpoints halt for any reason or there are data check errors on loop-back data.

Required Device Resource: LS, FS, HS Compliance Devices
Using: EHCI Compliance Stack

3.5 Debug Port

Test Debug port PCI Configuration Space Registers

TD.7.1 This test verifies that the debug port exists and that the PCI Configuration space capability registers have the correct values as defined by the EHCI specification for Debug port registers.

Test fails if the PCI capability register is not present or not at the appropriate addresses.

Required Device Resource: none
Using: EHCI Compliance Stack

Test Debug Port Host Controller Capability Register

TD.7.2 This test verifies that the debug port exists and that debug port number in the Host controller capability register is a valid port number.

Test fails if the debug port number is an invalid port number or if the debug port does not exist.

Required Device Resource: none
Using: EHCI Compliance Stack

Test Debug Port Host Controller Registers

TD.7.3 This test verifies that the debug port exists and that the host controller debug port register have the appropriate default values after a host controller reset. It also verifies that all read only fields cannot be written to and that read/write fields can be read/written to.

Test fails if:

- the debug port is not present
- the debug port registers do not have the appropriate default values
- read only fields can be written to
- the test cannot verify Read/Write fields

Required Device Resource: none
Using: EHCI Compliance Stack

Test Debug Port Control/Status Register

TD.7.4 This test verifies that the debug port exists and that the debug port Control/Status register has the defaults values and that the debug port enabled flag can be set when a device is attached and reset to the debug port.

Test fails if:

- the debug port is not present
- the debug port enabled is not set after the device connected to the port has been reset

Required Device Resource: None
Using: EHCI Compliance Stack

Test Debug port Data Streaming

TD.7.5 This test verifies that the debug port exists and that data can be looped back to and from a debug device connected to that port.

This test moves data through a bulk loop-back pair of endpoints on a debug device capable HS compliance device. It performs multiple iterations with all possible combinations of the RunStop bit in the USBCD Register and the PortEnable bit in the PortStatus Register on which the debug device is connected.

The test fails if the data read does not match the data written.

Required Device Resource: HS Compliance Device (Debug Device Capable)
Using: EHCI Compliance Stack

Test Debug port Error Test

TD.7.6 This test verifies that the debug port exists and verifies that the exception bits in the Control/Status register are correctly set when performing transactions to device/endpoint pairs that does not exist. It also checks that the Error/Good# field is set when the error occurs.

This test requires a debug device capable HS compliance device. It initially reads and writes to a valid endpoint on the device. It then performs the following tests:

- an 8 byte OUT to a non existent endpoint on the device
- an 8 byte IN to a non existent endpoint on the device
- an 8 byte OUT to an existent endpoint on the device when the port is suspended
- an 8 byte OUT to an existent endpoint on the device when the port is suspended

The test fails if the proper exception bits are not set when an error occurs.

Required Device Resource: none
Using: EHCI Compliance Stack

3.6 IBIT and FSTN Tests

Test Inactive on next transaction (i) bit

TD.8.1 This test verifies that the host controller properly implements the iBit and correctly transition the state of an interrupt queue head.

This test uses a full speed Interrupt IN endpoint. The test ensures sets the iBit before starting a full speed in transfer of 3 times the max packet size. It checks to ensure that the host controller correctly transitions the queue head from Active to In-Active after the first transaction on that endpoint.

The test fails if the transfer completes.

The second test ensures that the host controller correctly restarts a transfer that was deactivated due to the iBit being set at an earlier time. The host controller must re-activate the queue head and complete the transfer queued on it.

The test fails if the transfer does not complete.

Required Device Resource: FS Compliance Devices
Using: EHCI Compliance Stack

Test Periodic Frame Span Traversal Node (FSTN)

TD.9.1 This test verifies that the host controller properly implements the FSTN node and continues to correctly traverse the linked list.

This test uses an Interrupt IN loop-back pair and an Isochronous schmoo. The test is performed both with and without an Isochronous schmoo endpoint.

In the first test, the interrupt OUT is scheduled late and the interrupt IN is not late into the FSTN test tree. A transfer equal to the max packet size is written and read back. The same test is repeated with the Isochronous schmoo to push the OUT over a frame boundary.

In the second test, the interrupt IN is scheduled late and the interrupt OUT is not late into the FSTN test tree. A transfer equal to the max packet size is written and read back. The same test is repeated with the Isochronous schmoo to push the IN over a frame boundary.

The test is repeated for both full and low speed interrupt endpioints.

The test fails if the data read does not match the data written.

Required Device Resource: FS Compliance Devices
Using: EHCI Compliance Stack

TD.9.2 This test verifies that the host controller properly implements the FSTN node and continues to correctly traverse the linked list. In addition it verifies that the host controller can successfully manage combinations of high, full, and low-speed interrupt endpoints placed on the FSTN test tree.

The test places combinations of low, full and high speed interrupt loop-back endpoints on the FSTN test tree and performs transfers of 30 * max packetsize on each of them.

Test fails if any endpoints halt for any reason or there are data check errors on loop-back data.

Required Device Resource: HS, FS, LS Compliance Devices
Using: EHCI Compliance Stack

3.7 PCI Power Management

Test PCI PM Remote Wake from all supported D-states

TD.10.1 This test verifies that the host controller can successfully respond to remote wake signaling in all implemented device (Dx) states. Note that this test does not take the system into a reduced system power management state.

This test assumes only a high-speed compliance device is attached (to any root port).

There is one test iteration for each device state reported implemented by the device. The device is required to support (at least) D0 and D3. The tests are run with all WakeON* port control bits set to a one and to a zero.

The test algorithm is generally:

For dstate (D0..D3) do

Enable the compliance device for remote wake (with a timeout of 5 or so seconds).
Suspend the device's parent port (a root port) and halt the host controller (also turn off interrupt enables).
Save HC state that is not preserved in dstate
In the PMSCR register, clear PME_Status bit and set Dstate to dstate.
Poll PME_Status in the PMSCR register for a transition from 0 to 1
PME_Status must get set when the host controller detects remote-wake signaling from the compliance device.
Restore Dstate to D0
Restore save HC state
Restart HC
Complete resume signaling on port

End loop

Test fails host controller does not detect a resume event within the remote-wake timeout period (5 seconds or so). Test also fails if the port does not return to the enabled state after the port resume process has completed.

Required Device Resource: HS Compliance Devices
Using: EHCI Compliance Stack

Test PCI PM Ignore Disconnect event from all supported D-states

TD.10.2 This test verifies that the host controller will not report disconnect events on its root ports in all implemented device (Dx) states. Note that this test does not take the system into a reduced system power management state.

This test assumes only a high-speed compliance device is attached (to any root port).

There is one test iteration for each device state reported implemented by the device. The device is required to support (at least) D0 and D3. Each D-state is tested with port not enabled or suspended (testing with enabled is not a valid combination because the HC should not be stopped with any port in the enabled state).

Test algorithm:

Enumerate Bus

Find HS Compliance device and determine its parent port.

Set WakeOnConnect and WakeOnOverCurrent in the parent port to a one and set WakeOnDisconnect in the parent port to a zero.

For dstate (D0..D3) do

Suspend or disable the device's parent port (a root port) and halt the host controller (also turn off interrupt enables).
Save HC state that is not preserved in dstate
In the PMSCR register, clear PME_Status bit and set Dstate to dstate.
Ask user to disconnect the device on the device's parent port
After disconnect by user, check the PME_Status bit in the PMSCR register
Restore Dstate to D0
Restore save HC state
Restart HC
Check that parent port has connect change bit set to a one and connect status bit set to a zero.
Ask user to reattach the device into the same port

End loop

Test fails host controller sets PME_Status to a one when the device is disconnected.

Required Device Resource: HS Compliance Devices
Using: EHCI Compliance Stack

Test PCI PM Signal Disconnect event from all supported D-states

TD.10.3 This test verifies that the host controller will report disconnect events on its root ports in all implemented device (Dx) states. Note that this test does not take the system into a reduced system power management state.

This test assumes only a high-speed compliance device is attached (to any root port).

There is one test iteration for each device state reported implemented by the device. The device is required to support (at least) D0 and D3. Test with port not enabled or suspended. (testing with enabled is not a valid combination because the HC should not be stopped with any port in the enabled state).

Test Algorithm:

Enumerate Bus

Find HS Compliance device and determine its parent port.

Set WakeOnConnect and WakeOnOverCurrent in the parent port to a zero and set WakeOnDisconnect in the parent port to a one.

For dstate (D0..D3) do

Suspend or disable the device's parent port (a root port) and halt the host controller (also turn off interrupt enables).
Save HC state that is not preserved in dstate
In the PMSCR register, clear PME_Status bit and set Dstate to dstate.
Ask user to disconnect the device on the device's parent port
After disconnect by user, Poll PME_Status in the PMSCR register for a transition from 0 to 1. PME_Status must get set when the host controller detects the disconnect
Restore Dstate to D0
Restore save HC state
Restart HC
Check that parent port has connect change bit set to a one and connect status bit set to a zero.
Ask user to reattach the device into the same port

End loop

Test fails host controller does not set PME_Status to a one when the device is disconnected.

Required Device Resource: HS Compliance Devices
Using: EHCI Compliance Stack

Test PCI PM Ignore Connect event from all supported D-states

TD.10.4 This test verifies that the host controller will not report connect events on its root ports in all implemented device (Dx) states. Note that this test does not take the system into a reduced system power management state.

This test assumes only a high-speed compliance device is attached (to any root port).

There is one test iteration for each device state reported implemented by the device. The device is required to support (at least) D0 and D3.

Test Algorithm:

Enumerate Bus

Find HS Compliance device and determine its parent port.

Set WakeOnDisconnect and WakeOnOverCurrent in the parent port to a one and set WakeOnConnect in the parent port to a zero.

For dstate (D0..D3) do

Ask user to disconnect device from port (and verify).
Halt the host controller (also turn off interrupt enables).
Save HC state that is not preserved in dstate
In the PMSCR register, clear PME_Status bit and set Dstate to dstate.
Ask user to Connect the device on the device's parent port
After Connect by user, check the PME_Status bit in the PMSCR register
Restore Dstate to D0
Restore save HC state
Restart HC
Check that parent port has connect status bit set to a one.

End loop

Test fails host controller sets PME_Status to a one when the device is connected.

Required Device Resource: HS Compliance Devices
Using: EHCI Compliance Stack

Test PCI PM Report Connect event from all supported D-states

TD.10.5 This test verifies that the host controller will report connect events on its root ports in all implemented device (Dx) states. Note that this test does not take the system into a reduced system power management state.

This test assumes only a high-speed compliance device is attached (to any root port).

There is one test iteration for each device state reported implemented by the device. The device is required to support (at least) D0 and D3.

Test Algorithm:

Enumerate Bus

Find HS Compliance device and determine its parent port.

Set WakeOnDisconnect and WakeOnOverCurrent in the parent port to a zero and set WakeOnConnect in the parent port to a one.

For dstate (D0..D3) do

Ask user to disconnect device from port (and verify).
Halt the host controller (also turn off interrupt enables).
Save HC state that is not preserved in dstate
In the PMSCR register, clear PME_Status bit and set Dstate to dstate.
Ask user to Connect the device on the device's parent port
After connect by user, Poll PME_Status in the PMSCR register for a transition from 0 to 1. PME_Status must get set when the host controller detects the connect
Restore Dstate to D0
Restore save HC state
Restart HC
Check that parent port has connect status bit set to a one.

End loop

Test fails host controller does not set PME_Status to a one when the device is connected.

Required Device Resource: HS Compliance Devices
Using: EHCI Compliance Stack

3.8 Miscellaneous

Test PID (data toggle) mis-match Events

TD.6.1 This test verifies that the host controller correctly handles data PID (data toggle) mismatch events.

This test uses an *Infinite source* endpoint on the compliance device, and a vendor unique command. The Vendor command has the side-effects:

- resets a selected endpoint's data toggle (to DATA0), and
- sets the endpoint's DWORD data pattern to *X*, and
- sets the endpoint's data pattern modifier to *M*, and
- sets the length for the next IN request to *L*.

On reception of this command, the compliance device initializes the selected IN data toggle, its data to *X*, and prepares it to send L bytes for the next IN. When it receives notification that the IN occurred, the device modifies the first DWORD of the data pattern by adding M, then re-initializes the endpoint to send the next packet of L bytes (data toggle is advanced as normal).

The test sequence is:

1. Use vendor command to initialize source endpoint's data toggle, pattern (X), modifier (M) and length (L1).
2. Issue an IN request for length L1 (assuming DATA0).
3. Use vendor command to initialize source endpoint's data toggle, pattern (Y), modifier (M) and length (L2).
4. Issue an IN request for length L2 (assuming DATA1).

Test fails if data received from the second IN request is not equal to Y' (e.g. modified version of Y). Test is repeated with dif and only iferent transfer size combinations for the first and second requests. The test is also repeated for full-speed asynchronous and interrupt endpoints.

The combinations of buffer sizes are (hs/fs):

L1	L2
<Maximum Packet	<Maximum Packet
Maximum Packet	Maximum Packet
Maximum Packet	<Maximum Packet
<Maximum Packet	Maximum Packet

Required Device Resource: HS and FS Compliance Device
Using: EHCI Compliance Stack

Test Handling STALL Handshake

TD.6.2 This test verifies that the host controller correctly handles a STALL handshake.

The test uses a vendor-specific command that instructs the compliance device to return a STALL handshake in response to the next token. The test then issues a request to the selected endpoint. The host controller must halt the Qhead when it receives a STALL handshake. Test is repeated for HS/FS Bulk (IN/OUT) and HS/FS/LS Interrupt (IN/OUT).

Test fails if host controller fails to halt the Queue head.

Required Device Resource: HS, FS and LS Compliance Device
Using: EHCI Compliance Stack

Appendix A Cross Reference

This appendix provides a summary cross-reference between the test descriptions and the test assertions that reference each test description.

TD.1.1	2.4.14, 2.4.15,
TD.1.10	2.1.12,
TD.1.11	2.1.4,
TD.1.2	2.1.13, 2.1.53,
TD.1.3	2.1.54, 2.1.55, 2.1.84,
TD.1.4	2.1.1, 2.1.2, 2.1.14, 2.1.15, 2.1.22, 2.1.51, 2.1.52, 2.1.56, 2.1.58,
TD.1.5	2.1.21, 2.1.25,
TD.1.6	2.1.7, 2.1.8, 2.1.9, 2.1.57,
TD.1.7	2.2.89,
TD.1.8	2.1.11, 2.1.19, 2.1.20, 2.1.24, 2.1.26, 2.1.27, 2.1.28, 2.1.29, 2.1.45, 2.2.7, 2.2.8, 2.2.9, 2.4.8,
TD.1.9	2.1.3,
TD.2.1	2.1.5, 2.1.80, 2.1.81, 2.1.82,
TD.2.10	2.1.93,
TD.2.2	2.1.30, 2.1.31, 2.1.33, 2.1.46, 2.1.64, 2.1.67, 2.1.69, 2.1.72, 2.1.74, 2.1.83, 2.1.106, 2.1.107, 2.1.108, 2.4.7,
TD.2.3	2.1.75, 2.1.76, 2.1.77, 2.1.78, 2.1.79,
TD.2.4	2.1.103,
TD.2.5	2.1.104,
TD.2.6	2.1.10, 2.1.30, 2.1.85, 2.1.86, 2.1.87, 2.1.88, 2.1.89, 2.1.90, 2.1.102, 2.1.105, 2.1.109,
TD.2.7	2.1.23,
TD.2.8	2.1.91, 2.1.92, 2.1.94, 2.1.95, 2.1.96, 2.1.98,
TD.2.9	2.1.91, 2.1.94, 2.1.97, 2.1.98,
TD.3.1	2.1.49, 2.1.50,
TD.3.10	2.2.4, 2.2.6, 2.2.11, 2.2.94, 2.2.96, 2.2.97, 2.2.99, 2.2.100, 2.2.103, 2.2.104, 2.2.105, 2.2.106, 2.2.108, 2.2.109, 2.2.110, 2.2.111, 2.2.112, 2.2.113, 2.2.115, 2.2.121, 2.2.122, 2.2.134, 2.2.135, 2.2.137, 2.2.138, 2.3.3, 2.3.4, 2.3.5, 2.3.6, 2.3.7, 2.3.8, 2.3.12, 2.4.11, 2.4.12,
TD.3.11	2.2.94, 2.2.97, 2.2.116, 2.2.121,
TD.3.12	2.2.3, 2.2.11, 2.2.12, 2.2.13, 2.2.14, 2.2.15, 2.2.16, 2.2.17, 2.2.18, 2.2.19, 2.2.20, 2.2.24, 2.4.11, 2.4.12,
TD.3.13	2.2.17, 2.2.21, 2.2.22, 2.2.23, 2.2.24,
TD.3.14	2.2.5, 2.2.11, 2.2.53, 2.2.54, 2.2.55, 2.2.56, 2.2.59, 2.2.60, 2.2.66, 2.2.67, 2.2.68, 2.2.69, 2.4.11, 2.4.12,
TD.3.15	2.2.57, 2.2.58, 2.2.59, 2.2.60, 2.2.61, 2.2.62, 2.2.63, 2.2.64, 2.2.65, 2.2.67, 2.2.68, 2.2.69,
TD.3.16	2.2.98,
TD.3.17	2.2.87, 2.4.16,
TD.3.18	2.2.133,
TD.3.19	2.2.18, 2.2.23, 2.2.24, 2.2.26, 2.2.27, 2.2.28, 2.2.29, 2.2.30, 2.2.31, 2.2.32, 2.2.33, 2.2.34, 2.2.35, 2.2.36, 2.2.37, 2.2.38, 2.2.39, 2.2.40, 2.2.41, 2.2.42, 2.2.43, 2.2.44, 2.2.45, 2.2.46, 2.2.47, 2.2.48,
TD.3.2	2.1.101, 2.1.105,
TD.3.20	2.2.49,
TD.3.21	2.2.95, 2.2.96, 2.2.117, 2.4.5, 2.4.13,

TD.3.22	2.2.139, 2.2.140, 2.2.143, 2.2.145, 2.2.147, 2.2.148, 2.2.149, 2.2.151, 2.3.3, 2.3.4, 2.3.5, 2.3.6, 2.3.7, 2.3.8, 2.3.12, 2.3.13, 2.3.15, 2.3.18,
TD.3.23	2.1.50, 2.2.59, 2.2.60, 2.2.61, 2.2.62, 2.2.63, 2.2.64, 2.2.65, 2.2.67, 2.2.68, 2.2.69, 2.2.70, 2.2.72, 2.2.73, 2.3.16, 2.3.18,
TD.3.24	2.2.71, 2.2.74, 2.2.75, 2.2.76, 2.2.78, 2.2.79, 2.3.17,
TD.3.25	2.2.144, 2.2.146, 2.2.150, 2.2.152, 2.2.153, 2.3.9, 2.3.10, 2.3.11, 2.3.14,
TD.3.26	2.2.82, 2.2.136, 2.3.2, 2.3.10, 2.3.11,
TD.3.3	2.2.83, 2.2.84, 2.2.85, 2.2.86, 2.2.88, 2.2.90, 2.2.101, 2.2.102, 2.2.103, 2.2.124,
TD.3.4	2.2.81, 2.2.91,
TD.3.5	2.1.43, 2.2.92, 2.4.9,
TD.3.6	2.2.107, 2.2.126, 2.2.127, 2.2.128, 2.2.129, 2.2.130, 2.2.131, 2.2.132,
TD.3.7	2.2.126, 2.2.127, 2.2.128, 2.2.129, 2.2.130, 2.2.131,
TD.3.8	2.1.16, 2.1.17, 2.1.18, 2.1.39, 2.1.40, 2.1.41, 2.1.42, 2.1.48, 2.2.25, 2.2.115, 2.4.1, 2.4.2, 2.4.3, 2.4.4,
TD.3.9	2.1.34, 2.1.35, 2.1.36, 2.1.37, 2.1.38, 2.1.47, 2.2.50, 2.2.93, 2.2.114, 2.2.118, 2.2.119, 2.2.122, 2.2.125, 2.4.6,
TD.4.1	2.2.1, 2.2.2, 2.2.10, 2.5.1,
TD.4.2	2.5.2,
TD.4.3	2.3.1, 2.5.3,
TD.4.4	2.5.4,
TD.6.1	2.2.123,
TD.6.2	2.2.120,
TD.7.1	2.6.1, 2.6.2, 2.6.3, 2.6.4, 2.6.5, 2.6.6,
TD.7.2	2.6.7,
TD.7.3	2.6.8, 2.6.9, 2.6.11, 2.6.12, 2.6.13, 2.6.14, 2.6.15, 2.6.16, 2.6.17,
TD.7.4	2.6.10,
TD.7.5	2.6.18, 2.6.19, 2.6.20, 2.6.21,
TD.7.6	2.6.22, 2.6.23,
TD.8.1	2.7.1,
TD.9.1	2.7.2, 2.7.3, 2.7.4, 2.7.5, 2.7.6, 2.7.7, 2.7.8,
TD.9.2	2.7.2, 2.7.3, 2.7.4, 2.7.5, 2.7.6, 2.7.7, 2.7.8,
TD.10.1	2.1.2, 2.1.97,
TD.10.2	2.1.2, 2.1.66,
TD.10.3	2.1.2, 2.1.65,
TD.10.4	2.1.2, 2.1.71,
TD.10.5	2.1.2, 2.1.70,