Intel® 82801EB (ICH5), 82801ER (ICH5R), and 82801DB (ICH4) Enhanced Host Controller Interface (EHCI)

Programmer's Reference Manual (PRM)

April 2003

Document Number: 298656-002

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® I/O Controller Hub may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2002–2003, Intel Corporation

Contents

Figures

Tables

Revision History

Revision Number.	Description	Revision Date
-001	• Initial Release	May 2002
-002	• Added ICH5 (Intel® 82801EB chipset) and ICH5R (Intel® 82801ER chipset) support	April 2003

1 Introduction

1.1 Overview

This document was prepared to assist Integrated Hardware Vendors (IHV) in supporting the Intel® 82801EB (ICH5), Intel® 82801ER (ICH5R), and Intel® 82801DB (ICH4) EHCI feature set. This document also describes the general requirements to develop an EHCI (Enhanced Host Controller Interface) system device driver that will make use of the EHCI interface. The primary purpose of this document is to supplement the information provided in the Enhanced Host Controller Interface Specification For Universal Serial Bus, along with the ICH5 and ICH4 I/O controller hub datasheets, for use by IHVs and Intel customers developing their own driver interface.

It is assumed that the reader has a working knowledge of USB1.1/USB2.0 architecture and the ICH4/5 EHCI implementation of the EHCI specification. Also, the reader should have an understanding of USB driver development for the target operating systems.

This document also describes functions that the BIOS or Operating Systems (OS) must perform in order to ensure correct and reliable operation of the platform. This document will be supplemented from time to time with specification updates. The specification updates contain information relating to the latest programming changes. Check with your Intel representative for availability of specification updates.

The recommendations in this Programmers Reference Manual (PRM) apply to the following components:

Figure 1. Intel® ICH4/5 Device IDs

ICH Device	Controller	Device Function Number	PCI Device ID	PCI Vendor ID
82801DB (ICH4) & 82801DBM (ICH4-M)	EHCI	D29:F7	24CD	8086h
	UHCI(s)	D29:F0-F2	24C2, 24C4, 24C7	8086h
82801EB (ICH5) & 82801ER (ICH5R)	EHCI	D29:F7	24DD	8086h
	UHCI(s)	D29:F0-F3	24D2, 24D4, 24D7, 24DE	8086h

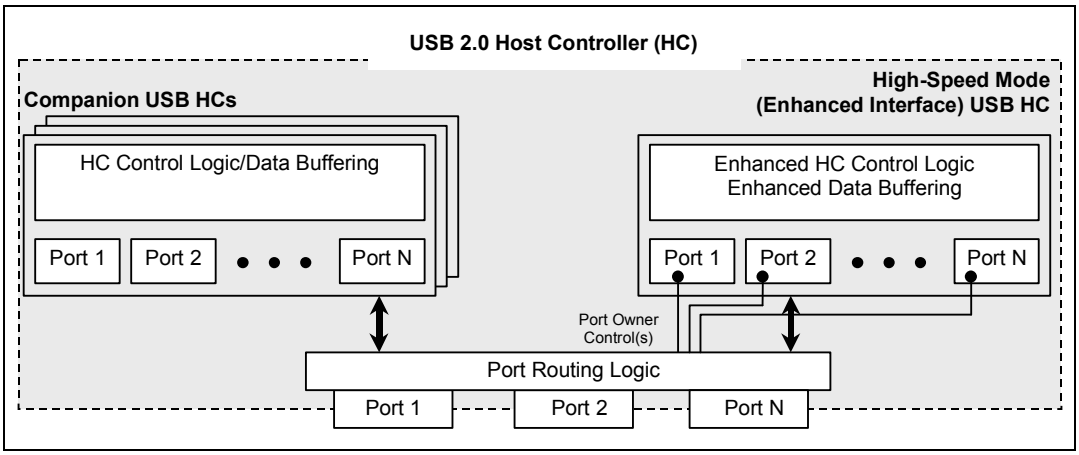
Note: In this document, general reference to the ICH4/5 refers to both the ICH4 and ICH5 devices. Any specific reference to an ICHx device that has a corresponding ICHx-M/R device will include the ICHx-M/R part (i.e., reference to ICH4 includes ICH4-M, reference to ICH5 includes ICH5R).

This page is intentionally left blank.

2 Intel® ICH4/5 EHCI Basic Attributes

In this document, the Intel ICH refers to the Intel I/O Controller Hub. ICH4 is the first generation of the ICH that implements the EHCI interface, followed by ICH5 as the second generation to include this interface. As indicated in Figure 2, the ICH4/5 provides a USB2.0 compliant host controller. Figure 2, below, represents a conceptual view of the USB2.0 Host Controller:

Figure 2. USB 2.0 Host Controller



A USB 2.0 Host Controller includes one high-speed mode host controller and zero or more USB 1.1 compliant host controllers. The high-speed host controller implements an EHCI and is used for all high-speed communications to high-speed-mode devices connected to the root ports of the USB 2.0 host controller. The USB 1.1 compliant host controllers are used for all low/full speed communications. To which controller a device communicates with is determined by system software and the proper programming of the port routing logic. Note that the EHCI cannot *directly* interface with low/full speed devices.

This architecture allows communications to full-speed and low-speed devices connected to the root ports of the USB 2.0 host controller to be provided by companion USB 1.1 host controllers. If an implementation does not include companion host controllers, the host controller must include one or more USB 2.0 Hubs permanently attached to the EHCI ports.

This architecture allows the USB 2.0 host controller to provide USB 1.1 functionality provided that there is at least USB 1.1 system software support in the operating system. Full USB 2.0 functionality is delivered when both USB 1.1 and EHCI software is available in the operating system. Note that if the operating system does not provide support for EHCI, the USB ports are *automatically* routed to the companion USB1.1 controllers, thus insuring that non-USB2 aware BIOS and operating systems can continue to function without modifications.

2.1 Intel® ICH4/5 USB2.0 Implementation

The ICH4/5 contains an Enhanced Host Controller Interface (EHCI) compliant host controller that supports up to eight (ICH5) USB 2.0 spec-compliant root ports (ICH4 supports up to six USB 2.0 root ports). USB2 allows data transfers up to 480 Mbps using the same pins as the USB1.1 (UHCI – Universal Host Controller Interface) ports. The Intel ICH4/5 contains port-routing logic that determines whether a USB port is controlled by one of the UHCI controllers or by the EHCI controller.

A summary of the key architectural differences between the USB1.1 UHCI host controllers and the USB 2.0 EHCI host controller is shown in the table below:

Figure 3. Controller Architectural Differences

	USB1.1 UHCI	USB2.0 EHCI
Accessible by	I/O space	Memory Space
Memory Data Structure	Single linked list	Separate Periodic and Asynchronous lists
Ports per Controller	2	N = 8 for Intel ICH5 = 6 for Intel ICH4

This document is limited to specify the software requirements and driver interface for the ICH4/5 EHCI USB Controller. Where possible, this document has pointers to additional considerations for supporting future proliferation or derivatives of the ICH EHCI. However, considerations for these future devices are subject to change

3 Unsupported Features

The ICH4/5 does not support the following, *optional* features as defined in the *Enhanced Host Controller Interface Specification for Universal Serial Bus:*

- Port Indicators
- Port Routing Rules (including Companion Port Route Description)
- Port Power Control
- Programmable Frame List
- Asynchronous Schedule Park Capability
- Light Host Controller Reset

Specific details regarding these options can be found in the *Enhanced Host Controller Interface Specification for Universal Serial Bus*

This page is intentionally left blank.

4 EHCI Initialization

Initialization of the EHCI hardware is accomplished in three phases:

1. **System Reset** – This occurs during power up when the PCI signal RST# is asserted. This causes the PCI configuration space and the controller registers to be initialized to their hardware defaults.
2. **Pre-Operating System Initialization** – During this phase the BIOS and/or operating system perform initialization of the PCI configuration space (BAR, interrupt line assignment, etc) associated with the EHCI. With the exception of the BAR and the interrupt line assignment, these values will be rarely modified once the system is fully operational.
3. **Post Operating System Initialization** – This phase fully completes the EHCI initialization and places the EHCI in a fully functional state. While operating system agents (i.e., device driver) typically accomplish this phase, it can also be done by the BIOS, if the BIOS is required to support USB2 devices.

4.1 System Reset

EHCI registers are implemented in both the auxiliary well and the core well. Registers in the auxiliary well are reset under different conditions than the registers in the core well. The auxiliary power well registers are initialized to their default values in the following cases:

- Initial power-up of the auxiliary power well
- Setting the ***USB2CMD.HostControllerReset*** bit to 1

The core well registers are initialized to their default values in the following cases:

- Assertion of the system (core-well) hardware reset
- Setting the ***USB2CMD.HostControllerReset*** bit to 1
- Transition from the PCI Power Management D3hot state to the D0 state

Table 1 summarizes the behavior on hardware that HCRESET and D3->D0 incur:

Table 1. Reset Summarization

Reset	Does Reset	Does Not Reset	Comments
HCRESET bit set	Memory space registers except Structural Parameters (which is programmed by BIOS)	Configuration Registers	The HCRESET must only affect registers that the EHCI driver controls. PCI Configuration space and BIOS-programmed parameters cannot be reset.
Software writes the Device Power State from D3hot (11b) to D0 (00b)	Core-well registers (except BIOS-programmed registers)	Suspend-well registers; BIOS-programmed Core-well Registers	The D3-to-D0 transition must not cause wake information (suspend well) to be lost. It also must not clear BIOS-programmed registers because BIOS may not be invoked following the D3-to-D0 transition.

PCI configuration space registers implemented in the auxiliary power well are reset under different conditions that those found in the core power well. The auxiliary well, configuration space registers are initialized to their default value in the following case:

- Initial power-up of the auxiliary power well

Core well PCI configuration space registers are initialized to their default values in the following cases:

- Assertion of the system (core-well) hardware reset
- Transition from the PCI Power Management D3hot state to the D0 state

4.2 Pre-Operating System Initialization

During system startup, the USB2.0 controller will be reset to its default state. In simplest terms, this means that the controller can be conceptually viewed as four distinct USB 1.1 compliant host controllers capable of supporting full-speed (12 Mbs) and low-speed (1.5 Mbs) USB devices. For system software (BIOS and the OS) to take advantage of the capabilities of the EHCI, some special setup of the hardware is necessary. In the event that BIOS *does not* take advantage of the EHCI it must still perform some limited initialization of the EHCI PCI configuration space and registers in the mapped memory space, in order to prepare the hardware for potential use by operating system software

4.3 BIOS Initialization

Assuming that the EHCI functionality is enabled in the system (See Section4.3.1, *Hiding the EHCI and One or More UHCI Companion Controllers*), BIOS must initialize the EHCI as follows:

Note: Unless otherwise noted, the following steps may be completed in any order. Registers denoted with '♦' are described in Section 4.3.2, *PCI Configuration Registers.*

1. Write to the Frame Length Adjustment♦ register as required. This register is maintained in the suspend well; this step is only necessary on the first BIOS initialization sequence following a loss of the suspend well power.
2. Write to the Port Wake Capabilities♦ register as required. This register is maintained in the suspend well; this step is only necessary on the first BIOS initialization sequence following a loss of the suspend well power.
3. Write a 1 to the WRT_RDONLY bit of the Access Control♦ register. This step must occur before any writes to the normally read-only registers in the following steps.
4. Write to the Subsystem Vendor ID♦ and Subsystem ID♦ registers.
5. Write the PCI Power Management Aux_Current field as required.
6. Write 0h to the Next Item Pointer #1♦ if the Debug Port capability needs to be hidden.
7. If fewer than maximum number of supported USB ports (8 for ICH5, 6 for ICH4) are provided in the system, then the number of downstream ports and classic companion controllers must be adjusted as follows:
   - The EHCI memory base register (BAR) ♦ must be written and enabled (set the Command register♦ to 1) for accessing the *Structural Parameters* (**HCSPARAMS**) register.
   - Modify the "Number of Companion Controllers" and "Number of Ports" fields in the *Structural Parameters* register according to Table 2. Port Number Customization, below.
   - Modify the Function Disable Register according to Table 2. See also: Section 4.3.1, *Hiding the EHCI and One or More UHCI Companion Controllers.*
8. Write a 0 to the WRT_RDONLY bit of the Access Control♦ register. This step must occur after any writes to the normally read-only registers in the preceding steps.
9. Depending on the level of support for wake enable implemented in the operating systems EHCI drivers, the BIOS may choose to statically select wake policies (i.e., wake on connect but not on disconnect) in each port's *Port Status and Control* register. This would be a useful mechanism by operating systems drivers could determine the wake policy of a system when no other mechanism exists (i.e., ACPI). If no advanced mechanisms exists for the operating system, then the OS EHCI driver would be expected to maintain the value in each of these bits when performing writes to the *Port Status and Control* registers and after performing a host controller reset.

Table 2. Port Number Customization

Number of Ports	N_CC (EHCI Memory offset 04h, bits 15:12)	N_PORTS (EHCI Memory offset 04H, BITS 3:0)	Function Disable Reg (dev 31, func 0, offset F2h, bits 15:8)
8	4h (ICH5 default)	8h (ICH5 default)	00h (ICH5 default)
7	4h (ICH5 default)	7h	00h (ICH5 default)
6	3h (ICH4 default)	6h (ICH4 default)	08h (ICH4 default)
5	3h (ICH4 default)	5h	08h (ICH4 default)
4	2h	4h	0Ch
3	2h	3h	0Ch
2	1h	2h	0Eh
1	1h	1h	0Eh
0	Don't Care	Don't Care	8Fh

4.3.1 Hiding the EHCI and One or More UHCI Companion Controllers

In some instances, the BIOS may choose to 'hide' the EHCI and one or more companion controllers. When a device is hidden, an OS is incapable of both discovering and using the hidden device. To hide a host controller, the BIOS must program the Function Hide register located in the ICH4/5 Device 31, Function 0, offset F2h, with the following values:

Table 3. Device 31, Function 0: Function Hide Register (Offset F2h)

Bit	Description
15	**D29_F7_DISABLE:** Software sets this to 1 to disable the EHCI host controller. When disabled, the PCI configuration space registers for that function are not decoded by the Intel® ICH4/5. Note software must always disable all functionality within this function before disabling the configuration space.
11	**D29_F3_DISABLE:** Software sets this to 1 to disable the 4th UHCI controller. When disabled, the PCI configuration space registers for that function are not decoded by the ICH4/5. Note software must always disable all functionality within this function before disabling the configuration space. (Not supported in ICH4)
10	**D29_F2_DISABLE:** Software sets this to 1 to disable the 3rd UHCI controller. When disabled, the PCI configuration space registers for that function are not decoded by the ICH4/5. Note software must always disable all functionality within this function before disabling the configuration space.
9	**D29_F1_DISABLE**: Software sets this to 1 to disable the 2nd UHCI controller. When disabled, the PCI configuration space registers for that function are not decoded by the ICH4/5. Note software must always disable all functionality within this function before disabling the configuration space.
8	**D29_F0_DISABLE:** Software sets this to 1 to disable the entire USB Device (Device #29, Functions 0, 1, 2, and 3). When disabled, the ICH4/5 does not decode the PCI configuration space registers for that device. Note software must always disable all functionality within Device 29 before disabling the configuration space.

USB Functions should only be disabled by the BIOS during system initialization. When hiding one or more UHCI controllers, BIOS must insure that UHCI functions are disabled from the highest function number to the lowest. For example, if only 3 UHCI host controllers are required, then the system BIOS must disable Device 29, Function 3 (set bit 11). If only two UHCI host controllers are required, then the system BIOS must disable Device 29, Function 3 and Function #2 (set bits 11and 10). If only 1 UHCI controller is required then disable Device 29, Function 3, Function #2 and Function #1 (set bits 11, 10, and 9). If no USB controllers are required, then BIOS must disable Device 29, Function #7, Function 3, Function #2, Function#1 and Function #0 (set bits 15, 11, 10, 9, and 8).

Note: When disabling UHCI host controllers, the USB2 EHCI Structural Parameters Registers must be updated (by the BIOS) with coherent information in "Number of Companion Controllers" and "N_Ports" fields (See Section *4.3.3.1.1 - HCSPARAMS – Host Controller Structural Parameters*).

It is very important to note that BIOS cannot configure the device to provide EHCI support only. This configuration is prevented by hardware because, as per the PCI Specification, PCI devices that are accessible must always implement function 0. Therefore, EHCI host controller support must always be accompanied by support by at least 1 UHCI host controller (i.e., bits 8 and 15 are not set).

Note: To insure that a disabled USB function cannot initiate transactions for USB transfers or be the target of memory/IO requests, BIOS must insure the following prior to hiding the function:

- Clear the *Memory Space Enable* bit in the *Command Register* (PCI Configuration space, offset 4-5h) – EHCI only.
- Clear the *I/O Space Enable* bit in the *Command Register* – UHCI only.
- Function functionality is disabled:
  - HC is halted (EHCI and UHCI)
  - Interrupt enables (EHCI and UHCI)
  - Asynchronous and periodic schedules are not enabled (EHCI only)
  - Wake capabilities (UHCI and EHCI)

There are no restrictions on how BIOS writes to these bits. For example, BIOS may perform a single write to set/clear bits 15:8 or BIOS may perform individual writes to each bit to enable/disable a device. Regardless of which method is used, it is expected that BIOS will only disable USB functions during system initialization. *System software drivers should never manipulate this register as the location of these bits may change from one ICH version to the next.*

Since this register is in the core power well, BIOS must re-initialize this register when the system transitions from a S3 (or deeper) to S0 state as it is will not be preserved.

4.3.2 PCI Configuration Registers

Before the BIOS and/or system software can use the USB2 host controller portion of the EHCI, the following USB2 Host Controller PCI Configuration registers *must* be initialized by the BIOS:

Note: All other registers not specified are either reserved or not used.

Table 4. Device 29, Function 7: PCI Configuration Registers

Device 29 Function 7 USB2 Host Controller				
Offset	**Register**	**Default**	**Initialize**	**Comments**
04h-05h[1]	Command	0400h	0006h	Bit 2: Bus Master Enable Bit 1: Memory Space Enable. BAR value should be set before writing 1 to this value.
10h-13h [2]	Memory Base Address (BAR)	00000000h	********h	Bits 31:10 – Memory address signals. Aligned on 1kb boundary Bits 9:4 – Reserved. Read-Only 0 Bit 3 – Pre-fetchable. Hardwired to 0 indicating that the memory range is not pre-fetchable. Bits 2:1 – Type. Hardwired '00' indicating that memory range can be mapped anywhere in the 32 bit address space Bit 0 – Resource Type Indictor. Hardwired to 0 indicating that the base address maps to memory space
2Ch-2Dh(2)	USB2 Subsystem Vendor ID	0000h	OEM Specific	Combined with the USB2 Subsystem ID, the operating system can distinguish one subsystem from another.
2Eh-2Fh [3]	USB2 Subsystem Device ID	0000h	OEM Specific	See USB2 Subsystem Vendor ID. Note: Writes to this register should be done using a single 16-bit write cycle.
3Ch	Interrupt Line (INTL)	00h	**h	A hardware interrupt (0-Fh) that follows value assigned to PIRQD#. Has no effect on the Intel® ICH4/5 as it is used to indicate to software the IRQ value assigned to the device.
51h [4]	Next Item Pointer #1	58h	OEM Specific	This value defaults to 58h, which indicates the offset in PCI configuration where the next device capability can be found. If set to 00h, then the BIOS is effectively hiding the existence of the debug port capability.

Device 29 Function 7 USB2 Host Controller				
Offset	**Register**	**Default**	**Initialize**	**Comments**
52h-53h [5]	Power Management Capabilities	C9C2h	****h	Describes the power management capabilities of the device. Bits 15:11 – PME support. The bit assignments are compliant with the PCI Power Management specification. Because the ICH5 EHCI does not support D1 or D2 states, a value of xx11xh is not supported. Bit 10 – D2_Support: Not supported by the ICH4/5 Bit 9 – D1_Support: Not supported by the ICH4/5 Bits 8:6 – Aux_Current. The bit assignments are compliant with the PCI Power Management specification. The ICH4/5 EHCI reports maximum Suspend well current required when in the D3 cold state. This value may be changed by BIOS when a more accurate value is known. Bit 5 – DSI: The ICH4/5 reports 0, indicating that no device-specific initialization is required. Bit 4 – Reserved Bit 3 – PME Clock: The ICH4/5 reports 0, indicating that no PCI clock is required to generate PME#. Bits 2:0 – Version: The ICH4/5 reports 010, indicating that it complies with Revision 1.1 of *the PCI Power Management Specification.*
54h-55h	Power Management Control/Status (PMSCR)	0000h	****h	Bit 15 - PME_Status: This bit is set when the ICH4/5 EHCI would normally assert the PME# signal independent of the state of the PME_En bit. Writing a 1 to this bit will clear it and cause the internal PME to de-assert (if enabled). Writing a 0 has no effect. This bit must be explicitly cleared by the operating system each time the operating system is loaded. Bits 14:13 – Data Scale: The ICH4/5 hardwires these bits to “00” because it does not support the associated Data register. Bits 12:9 – Data Select: The ICH4/5 hardwires these bits to “0000” because it does not support the associated Data register. Bit 8 – PME_En: A 1 enables the ICH4/5 EHCI to generate an internal PME signal when PME_Status is 1. System software (BIOS or OS) should program this bit to 0 when the device is in the D0 state. Bits 7:2 – Reserved. Bit 1:0 – PowerState – Refer to the Section 11 (*Power Management*) for usage details.

Device 29 Function 7 USB2 Host Controller				
Offset	**Register**	**Default**	**Initialize**	**Comments**
61h [6]	Frame Length Adjustment Register (FLADJ)	20h	**h	System specific. This value is used to adjust any offset from the clock source that generates the clock that drives the SOF counter. Default value of 20h indicates a standard 1 ms frame/125us frame length. Note: In the event that the default or BIOS assigned value is incorrect, system software is permitted to reprogram this value.
62h	Port Wake Capability (PORTWAKECAP)	7fh	OEM Specific	Programming of this register does not affect the operation of the EHCI. System software uses this value when enabling devices and ports for remote wakeup. Bits 15:7 – Reserved, read only. Bits 6:1 – Port Wake Capability Mask. These bits correspond to physical port implementations on the ICH4/5 (i.e., set bit 2 if port 2 supports wake), Bit 0 – Set indicates that this register is implemented for OS system software use.
68h-6Bh	USB Legacy Support EHCI Extended Capability Register (USBLEGSUP)	00000001h	OEM Specific	This register is used by system BIOS software and OS software for coordinating the ownership of the EHCI host controller. Bits 31:25 – Reserved, hardwired to 0. Bit 24 - HC OS Owned Semaphore: System software sets this bit to request ownership of the EHCI controller. Ownership is obtained when this bit reads as 1 and the HC BIOS Owned Semaphore bit reads as clear. Bits 23:17 – Reserved, hardwired to 0. Bit 16 - HC BIOS Owned Semaphore: The BIOS sets this bit to establish ownership of the EHCI controller. System BIOS will clear this bit in response to a request for ownership of the EHCI controller by system software. Bits 15:8 - Next EHCI Capability Pointer: A value of 00h indicates that there are no EHCI Extended Capability structures in this device. Bits 7:0 – Capability ID: A value of 01h indicates that this EHCI Extended Capability is the Legacy Support Capability. See Section 5- *EHCI Ownership* for more details.
6C-6Fh	USB Legacy Support Control/Status Register (USBLEGCTLSTS)	00000000h	********h	BIOS uses this register to enable SMIs for EHCI/USB events that require tracking. Bits [21:16] of this register are simply shadow bit of **USB2STS** register [5:0]. See Table 5. USBLEGCTLSTS — USB Legacy Support Control/Status, for field definitions.
80h	Access Control	00h	**h	Bit 0 – WRT_RDONLY. When set this bit allows a select group of registers (documented above) to be written to by software. When clear, the above-mentioned registers are read-only.

NOTES:

1. Regardless of the presence of a PnP OS, BIOS may have to initialize this register with a valid value if the Capability Registers for the EHCI need to be programmed with values other than the defaults. The Capability Registers are located in memory-mapped I/O space.
2. Writes to this register are enabled when the **WRT_RDONLY** bit in the Access Control register is set.
3. Writes to this register are enabled when the **WRT_RDONLY** bit in the Access Control register is set.
4. This register should only be written during system initialization before software has enabled any master-initiated traffic. Only values of 58h and 00h are legal. Writes to this register are enabled when the WRT_RDONLY bit in the Access Control register is set.
5. Normally this register is read-only to report capabilities to the power management software. To allow different power management capabilities (system dependent) to be reported, bits 15:11 and 8:6 are write-able. The value written to this register does not affect the hardware other than changing the value returned during a read.
6. Software must insure that the host controller is not running (i.e., HCHalted bit must be set) prior to modifying the contents of the FLADJ register.

Initialization of the standard PCI registers above (those registers with offsets in **BOLD**) is the responsibility of the PnP (Plug and Play) capable OS. If a PnP OS is not available in the system, it is then the BIOS responsibility to configure all PCI devices including the registers above.

Determination of the presence of a PnP capable OS is usually made via a switch in the BIOS System Setup. However, the final configuration or the existence or not of this switch is implementation dependent.

Table 5. USBLEGCTLSTS — USB Legacy Support Control/Status

Bit	Description
31	**SMI on BAR – R/WC.** 0=Default. This bit is set to 1 whenever the **Base Address Register** (BAR) is written.
30	**SMI on PCI Command – R/WC.** 0=Default. This bit is set to 1 whenever the **PCI Command Register** is written.
29	**SMI on OS Ownership Change– R/WC.** 0=Default. This bit is set to 1 whenever the *HC OS Owned Semaphore* bit in the USBLEGSUP register transitions from 1 to a 0 or 0 to a 1
28:22	**Reserved**. These bits are reserved. Hardwired to 00.
21	**SMI on Async Advance — RO.** 0=Default. Shadow bit of the *Interrupt on Async Advance* bit in the USBSTS register, see the *Enhanced Host Controller Specification for Universal Serial Bus* for definition. To clear this bit system software must write a 1 to the *Interrupt on Async Advance* bit in the USB2STS register.
20	**SMI on Host System Error — RO.** 0=Default. Shadow bit of *Host System Error* bit in the USB2STS register, see *Enhanced Host Controller Specification for Universal Serial Bus* for definition and effects of the events associated with this bit being set to a 1. To clear this bit system software must write a 1 to the *Host System Error* bit in the USB2STS register.
19	**SMI on Frame List Rollover — RO.** 0=Default. Shadow bit of *Frame List Rollover* bit in the USB2STS register, see *Enhanced Host Controller Specification for Universal Serial Bus* for definition. To clear this bit system software must write a 1 to the *Frame List Rollover* bit in the USB2STS register.

Bit	Description
18	**SMI on Port Change Detect — RO.** 0=Default. Shadow bit of *Port Change Detect* bit in the USB2STS register, see *Enhanced Host Controller Specification for Universal Serial Bus* for definition. To clear this bit system software must write a 1 to the *Port Change Detect* bit in the USB2STS register.
17	**SMI on USB Error — RO.** 0=Default. Shadow bit of *USB Error Interrupt* (USBERRINT) bit in the USB2STS register, see *Enhanced Host Controller Specification for Universal Serial Bus* for definition. To clear this bit system software must write a 1 to the *USB Error Interrupt* bit in the USB2STS register.
16	**SMI on USB Complete — RO.** 0=Default. Shadow bit of *USB Interrupt* (USBINT) bit in the USB2STS register, see *Enhanced Host Controller Specification for Universal Serial Bus* for definition. To clear this bit system software must write a 1 to the *USB Interrupt* bit in the USB2STS register.
15	**SMI on BAR Enable — R/W.** 0=Default. When this bit is 1 and **SMI on BAR** is 1, then the host controller will issue an SMI.
14	**SMI on PCI Command Enable — R/W.** 0=Default. When this bit is 1 and **SMI on PCI Command** is 1, then the host controller will issue an SMI.
13	**SMI on OS Ownership Enable — R/W.** 0=Default. When this bit is a 1 AND the **OS Ownership Change** bit is 1, the host controller will issue an SMI.
12:6	**Reserved**. These bits are reserved. Hardwired to 00h.
5	**SMI on Async Advance Enable — R/W.** 0=Default. When this bit is a 1, and the *SMI on Async Advance* bit (above) in this register is a 1, the host controller will issue an SMI immediately.[1]
4	**SMI on Host System Error Enable — R/W.** 0=Default. When this bit is a 1, and the *SMI on Host System Error* bit (above) in this register is a 1, the host controller will issue an SMI immediately.[1]
3	**SMI on Frame List Rollover Enable — R/W.** 0=Default. When this bit is a 1, and the *SMI on Frame List Rollover* bit (above) in this register is a 1, the host controller will issue an SMI immediately.[1]
2	**SMI on Port Change Enable — R/W.** 0=Default. When this bit is a 1, and the *SMI on Port Change Detect* bit (above) in this register is a 1, the host controller will issue an SMI immediately[1.]
1	**SMI on USB Error Enable — R/W.** 0=Default. When this bit is a 1, and the *SMI on USB Error* bit (above) in this register is a 1, the host controller will issue an SMI immediately.[1]
0	**USB SMI Enable — R/W.** 0=Default. When this bit is a 1, and the *SMI on USB Complete* bit (above) in this register is a 1, the host controller will issue an SMI immediately.[1]

NOTES:

1. For all enable register bits, 1= Enabled, 0= Disabled

   SMI – System Management Interrupt
   BAR – Base Address Register
   MSE – Memory Space Enable
2. SMI's are independent of the interrupt threshold value.

4.3.3 Memory Mapped I/O Registers

The USB2 EHCI memory-mapped I/O space is composed of two sets of registers:

1. Capability Registers
2. Operational Registers

4.3.3.1 Capability Registers

The capability registers specify the limits, restrictions and capabilities of the host controller implementation. Table 6 summarizes the Capability registers:

Table 6. Capability Registers

Offset	Register	Default	Type
00h	Capabilities Registers Length	20h	Read Only
01h	Reserved	0	Read Only
02h-03h	Host Controller Interface Version Number	0096h	Read Only
04h-07h [1]	Structural Parameters	00104208h	Read/Write-Special
08h-0Bh	Capability Parameters	00006871h	Read Only
0Ch-1Fh	Reserved	Undefined	Read Only

NOTES:

1. "Read/Write Special" means that the register is normally read-only, but may be written when the **WRT_RDONLY** bit is set. Because these registers are expected to be programmed by BIOS during initialization, their contents must not be modified by HCRESET or D3-to-D0 internal reset.

Note: These registers start at BAR + offset 0 (zero).

4.3.3.1.1 HCSPARAMS – Host Controller Structural Parameters

Default Value: **00104208h**

These registers have no effect on hardware and are set for the benefit of the system software.

Bit	Description
31:24	Reserved. Hardwired to 0.
23:20	**Debug Port Number (DP_N)**. *For Intel® ICH4/5, this bit is hardwire to 1 and is read only.* Debug software can use this bit to determine which port has been designated as the debug port. A value of 1 indicates that the Debug Port is on the lowest numbered port on the ICH4/5.
19:16	Reserved
15:12	**Number of Companion Controllers (N_CC)**. These bits indicate the number of USB1.1 compliant controllers associated with the USB2.0 host controller. If this value is 0, then there are no USB1.1 controllers present and only high-speed devices may be connected to the host root ports. If this value is > 0, then USB1.1 controllers are present and high, full and low speed devices may be connected to the host controller root ports. Default = 4h on ICH5 (default value on ICH4 = 3h).
11:8	**Number of Ports per Companion Controller (N_PCC)**. This field indicates the number of ports supported per companion host controller. It is used to indicate the port routing configuration to system software. The ICH4/5 hardwires this field to 2h.
7:4	Reserved.
3:0	**N_PORTS**. This value specifies the number of *physical* downstream root ports implemented on the USB2 host controller. The value if this field indicates how many port registers are addressable in the Operational Register Space. Valid values are in the range of 1h to Fh. Default = 8h on ICH5 (default value on ICH4 = 6h). Software may write a value less than the default value for some platform configurations. A 0 in this field is undefined.

4.3.3.1.2 HCCPARAMS – Host Controller Capability Parameters

Default Value: **00006871h**

This 32-bit capability register is ***read only*** and specifies to software:

Bit	Description
31:16	Reserved. Hardwired to 0.
15:8	**EHCI Extended Capabilities Pointer (EECP).** For the Intel® ICH4/5, this field is hardwired to 68h, indicating that the EHCI capabilities list exists and begins at offset 68h in the PCI configuration space.
7:4	**Isochronous Scheduling Threshold**. This field indicates, relative to the current position of the executing host controller, where software can reliably update the isochronous schedule. When bit [7] is 0, the value of the least significant 3 bits indicates the number of micro-frames a host controller hold a set of isochronous data structures (one or more) before flushing the state. When bit [7] is a 1, then host software assumes the host controller may cache an isochronous data structure for an entire frame. Refer to the EHCI specification for details on how software uses this information for scheduling isochronous transfers. The ICH4/5 hardwires this field to 7h.
3	Reserved. Hardwired to 0.
2	**Asynchronous Schedule Park Capability.** The ICH4/5 does not support the parking feature. This bit is read-only 0.
1	**Programmable Frame List Flag**. If this bit is set to a 0, then system software must use a frame list length of 1024 elements with this host controller. The frame list must always be aligned on a 4K-page boundary. This requirement ensures that the frame list is always physically contiguous. The ICH4/5 does not support different frame list lengths. This bit is read-only 0.
0	**64-bit Addressing Capability.** This field documents the addressing range capability of this implementation. The value of this field determines whether software should use the 32-bit or 64-bit data structures. The ICH4/5 supports 64-bit addressing only. This bit is read-only 1.

Note: On systems where the BIOS is not required to support high-speed devices connected to the EHCI, no further action by the BIOS is required and the remainder of this PRM may be skipped. If your BIOS is required to support high-speed devices (i.e., mass storage) and/or low/full speed devices via USB2.0 hubs, then the remainder of this document is applicable.

4.3.3.2 Operational Registers

These registers and their usage are discussed in subsequent sections.

The operational registers are located after the capability registers. All registers are 32 bits in length.

Note that these registers start at BAR + <BAR.Capabilities Registers Length>.

Caution: Software must read/write to these registers using DWORD accesses only.

Table 7 summarizes the operational registers:

Table 7. USB2 Host Controller Operational Registers

Offset	Register	Default	Type
0h	USB2 Command (**USB2CMD**)	00080000h	Read/Write
4h	USB2 Status (**USB2STS**)	00001000h	Read/Write
8h	USB2 Interrupt Enable (**USB2INTR**)	00000000h	Read/Write
Ch	USB2 Frame Index (**FRINDEX**)	00000000h	Read/Write
10h	Control Data Structure Segment (**CTRLDSSEGMENT**)	00000000h	Read/Write
14h	Periodic Frame List Base Address (**PERIODICLISTBASE**)	00000000h	Read/Write
18h	Next Asynchronous List Address (**ASYNCLISTADDR**)	00000000h	Read/Write
1Ch-3fh	Reserved	0	Read Only
40h	Configure Flag Register (**CONFIGFLAG**)	00000000h	Read/Write
44h	Port 0 Status and Control (**PORTSC**)	00003000h	Read/Write
48h	Port 1 Status and Control (**PORTSC**)	00003000h	Read/Write
4Ch	Port 2 Status and Control (**PORTSC**)	00003000h	Read/Write
50h	Port 3 Status and Control (**PORTSC**)	00003000h	Read/Write
54h	Port 4 Status and Control (**PORTSC**)	00003000h	Read/Write
58h	Port 5 Status and Control (**PORTSC**)	00003000h	Read/Write
5Ch(1)	Port 6 Status and Control (**PORTSC**)	00003000h	Read/Write
60h(1)	Port 7 Status and Control (**PORT SC**)	00003000h	Read/Write
64-7fh	Reserved	Undefined	Read Only
80h-93h	Debug Port Registers (see Section 13 - *Debug Port*)		Read/Write
94h-3FFh	Reserved	Undefined	Read Only

NOTES:
1. RESERVED in ICH4 device.

4.3.3.3 Post-Operating System Initialization

At this point, the system BIOS and/or the operating system will have completed low level initialization of the EHCI. However, before any useful work can be done by the EHCI, system software *must* complete the controller initialization by performing the following steps:

1. Claim/request ownership of the EHCI. This process is described in detail in Section 5 - *EHCI Ownership*.
2. Program the ***CTRLDSSEGMENT*** register. This value must be programmed since the ICH4/5 only uses 64bit addressing (See Section 4.3.3.1.2-*HCCPARAMS – Host Controller Capability Parameters*). This register **must** be programmed before the periodic and asynchronous schedules are enabled.
3. Determine which events should cause an interrupt. System software programs the ***USB2INTR*** register with the appropriate value. See Section 9 - *Hardware Interrupt Routing* - for additional details.

Note: Always insure that the ***USB2STS*** register is clear and the interrupt associated with the EHCI controller has been 'hooked' (has an *Interrupt Service Routine* - ISR) before setting the Interrupt Enable register.

4. Program the ***USB2CMD.InterruptThresholdControl*** bits to set the desired interrupt threshold and turn the host controller *ON* via setting the ***USB2CMD.Run/Stop*** bit. Setting the ***Run/Stop*** bit with both the periodic and asynchronous schedules disabled will still allow interrupts and enabled port events to be visible to software
5. Program the ***Configure Flag*** to a 1 to route all ports to the EHCI controller. Because setting this flag causes all ports to be unconditionally routed to the EHCI, all USB1.1 devices will cease to function until the bus is properly enumerated (i.e., each port is properly routed to its associated controller type: UHCI or EHCI)

Note: Manipulating the port routing via the ***Configure Flag*** is not intended for use during normal operation. During normal operation, software can route port(s) to the companion controller(s) by programming the ***PORTSC.PortOwner*** bit for the associated port. Individual port ownership is automatically returned to the EHCI controller when the port registers a device disconnect.

Once the above steps have been completed the port registers will begin reporting device connects, disconnects, etc. If a device is connected to a root port, system software can enumerate the bus segment associated with that port *after* it has initiated and completed a port reset (the port will be enabled after the reset if the attached device is high-speed) and has programmed the ***ASYNCLISTADDR*** register (and associated schedule) appropriately. Following the port reset, the port will be in the active state, with SOFs occurring **only** on those ports with high-speed devices connected. See Section 7 - *Port States* - for details regarding the various possible port states and additional register programming.

Optionally at this time, system software may choose to program the following registers:

- ***PERIODICLISTBASE***
- ***ASYNCLISTADDR***

The decision to program these register(s) at this time is implementation dependent. To prevent the host controller from needlessly accessing system memory (and potentially interfering with OS power management) system software should not program the associated enable bits (***USB2CMD.PeriodicScheduleEnable*** and ***USB2CMD.AsynchronousScheduleEnable*** respectively) to 1 until a high-speed device has been connected. See Section 7 - *Port States* -for details.

This page is intentionally left blank.

5 EHCI Ownership

In the event that BIOS has completely initialized the EHCI (See Section 4.3.3.3, *Post-Operating System Initialization*) and operating system specific EHCI device drivers are installed, special care must be taken when performing this 'hand-off' so as to avoid conflicts between the BIOS and the operating system. To perform a smooth transition of EHCI ownership from BIOS to the operating system (and vice-versa), the associated software must perform specific steps that are described later in this section.

To accommodate EHCI ownership transitions, the ICH4/5 implements the USB Legacy Support Extended Capability register. This register, which is located in PCI configuration space, provides the mechanisms required by the BIOS to enable SMI support for different EHCI events as well as providing semaphores required to synchronize ownership changes of the EHCI controller.

5.1 Determining USB Legacy Support

While the ICH4/5 always implements the ***USB Legacy Support Extended Capability*** register at offset 68h in PCI configuration space, it is recommended that system software always read the value of ***HCCPARAMS.EECP*** (See Section *4.3.3.1.2 - HCCPARAMS – Host Controller Capability Parameters*) to insure that the correct PCI configuration offset is used to locate the extended capability register. Furthermore, this will insure software compatibility with future generations of the ICH. Note that for the ICH4/5, this value will never be 0.

Because the ICH4/5 always implements the ***USB Legacy Extended Capability*** register at the PCI offset specified in ***HCCPARAMS.EECP***, it is not required that system software verifies this by examining the ***Capability ID*** bits of the ***Legacy Support*** register (as per the EHCI specification, a value of 01h identifies the register pointed at by ***HCCPARAMS.EECP*** as the USB Legacy Extended Capability register). However, for compatibility with future generations of the ICH, it is recommended that system software always verify that the ***Compatibility ID*** is set to a 1.

5.2 USB Legacy Support Usage Model

In addition to providing the ***Capability ID***, the ***USB Legacy Support Extended Capability*** register also provides the EHCI ownership (2) semaphores. One semaphore is for the operating system (***HC OS Owned Semaphore***) and the other is for the BIOS (***HC BIOS Owned Semaphore)***. These semaphores are readable and writable and are in adjacent bytes, which allow each agent (operating system or BIOS) to update their respective semaphore without overwriting the other ownership semaphore.

5.2.1 USB Legacy Support State Diagram

Following are two state machines that illustrate the proper protocol (e.g., updates to the ownership semaphores) that BIOS and the operating system must adhere to in order to coherently request and/or relinquish ownership of the EHCI controller. The conventions used in these figures are:

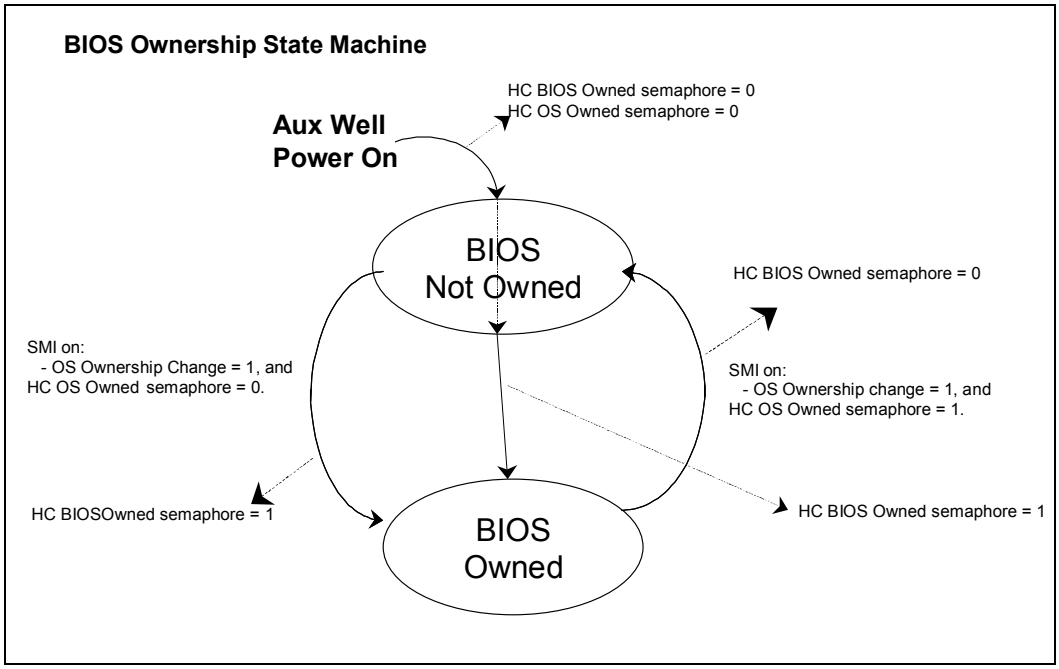
- Solid arcs denote single or multiple events that result in a state change.
- Dotted lines with arrows indicate side effects that take place. When attached to a solid arc, interpretation is that as a result of the event, the side effect occurs.

The figure below illustrates the protocol state machine for the BIOS ownership. The ***Legacy Support*** registers are located in the Auxiliary well, so any system event that removes power from the Aux well will result in these registers being reset to their default values when Auxiliary well power is restored.

5.2.1.1 BIOS State Diagram

When power is applied to the aux power well, the ***HC BIOS Owned*** and ***HC OS Owned*** semaphores in the ***USB Legacy Support Extended Capability*** go to their default values (e.g., 0's). BIOS may take ownership of the EHCI controller by setting the ***HC BIOS Owned Semaphore*** to a 1. BIOS is only allowed to take ownership of the EHCI controller when the ***HC OS Owned Semaphore*** is a 0. BIOS then may configure the SMI events it needs including the ***SMI on OS Ownership Change***. The BIOS now owns the EHCI controller, so it can configure the controller, enumerate the bus and use the devices found as necessary.

Figure 4. BIOS State Diagram



NOTE: The BIOS is allowed to claim control of the EHCI as a result of POST (Power On System Test) or as a result of the OS relinquishing control of the EHCI. The BIOS must never attempt to claim the EHCI once it has relinquished control.

5.2.1.1.1 BIOS Usage Model

BIOS is allowed to take ownership of the EHCI under the following conditions:

- During POST
- Anytime the operating system relinquishes ownership (***HC OS Owned Semaphore is not 1)***

At no time is the BIOS permitted to request ownership of the EHCI from the operating system (if the operating system has ownership), as the resultant behavior is undefined.

The following summarizes the BIOS usage model for the ***HC BIOS Owned Semaphore***:

1. BIOS must set the ***HC BIOS Owned Semaphore*** to a 1 prior to using the EHCI. This semaphore must remain set during the period that the BIOS is using (owns) the EHCI. After taking ownership of the EHCI, BIOS must:

    a. Initialize the EHCI (See Section *4.3.3.3*, *Post-Operating System Initialization*) as the state of the EHCI must be considered unknown.

    b. Set the ***SMI on OS Ownership Change*** bit in the **USB Legacy Support Control/Status** to 0 (if it was set as a result of the operating system relinquishing control of the EHCI) by writing a 1 to this bit.

    c. Set the ***SMI on OS Ownership Change Enable*** bit to a 1 in order to detect future operating system ownership requests.

2. BIOS must set the ***HC BIOS Owned Semaphore*** to 0 when:

    a. It voluntarily relinquishes control of the EHCI.

    b. A request for ownership change has been detected (***HC OS Owned Semaphore*** is set to 1). A request for ownership change can only be initiated by an operating system.

3. When relinquishing control of the EHCI, BIOS must:

    a. Insure that neither the Asynchronous or Periodic schedules are being processed:
    - USB2CMD.Asynchronous Schedule Enable is set to 0.
    - USB2CMD.Periodic Schedule Enable is set to 0.

    b. Insure that the EHCI is not running:
    - ***USB2CMD.Run/Stop*** is set to 0 AND
    - ***USB2STS.HCHalted*** is set to 1.

    c. Set the SMI enable bits 5:0 in the **USB Legacy Support Control/Status** register to 0.

    d. Set the USB2 Status (**USB2STS**) register bits to 0.

    e. Set the ***SMI on OS Ownership Change*** bit in the **USB Legacy Support Control/Status** to 0 by writing it to a 1.

f. Set the ***Configure Flag*** *to 0.*

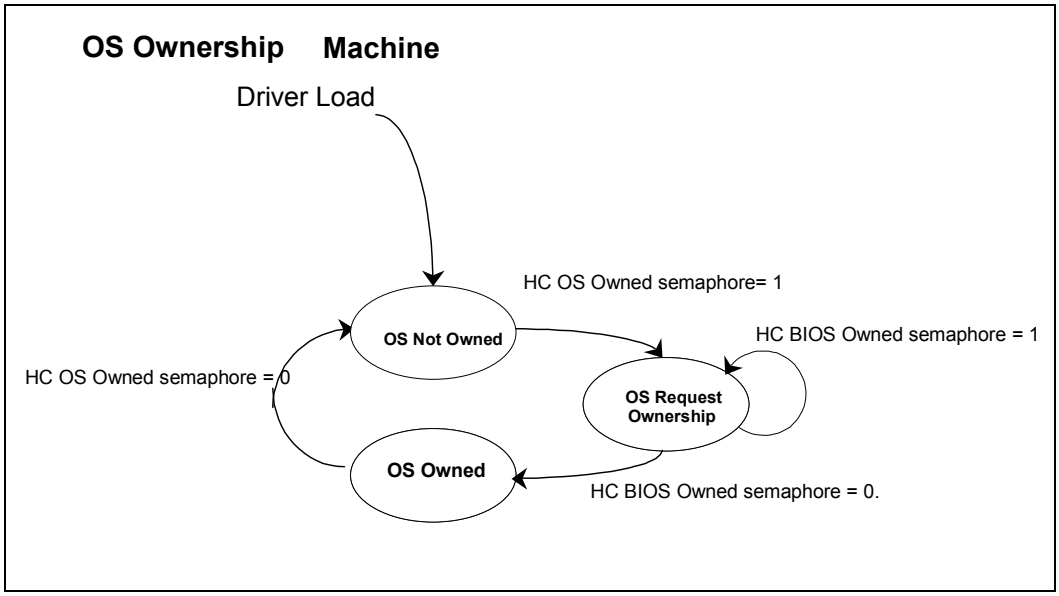
g. Set the ***HC BIOS Owned Semaphore*** to 0.

5.2.1.2 Operating System State Diagram

Eventually, the operating system will load. If the operating system has support for the EHCI controller, it will need exclusive control over the EHCI controller. The operating system must utilize the protocol defined in figure below to request ownership of the host controller before it takes ownership and uses the controller. The operating system initiates an ownership request by setting the ***HC OS Owned Semaphore*** to a 1. The operating system must wait for the ***HC BIOS Owned Semaphore*** to go to a 0 before attempting to use the EHCI controller. The amount of time that the operating system must wait for BIOS to respond to the request for ownership is not bound and is dependent on the BIOS implementation. Note that there is no similar SMI-type event is defined that allows the BIOS to request ownership from the operating system.

If the BIOS sets the ***SMI on OS Ownership ChangeEnable*** bit in the ***USB Legacy Support Control/Status*** register to a 1, it will receive an SMI when the operating system sets the ***HC OS Owned Semaphore*** to a 1.

Figure 5. OS State Diagram



In the event that the operating system driver unloads and/or wants to relinquish ownership of the EHCI controller, it must set the ***HC OS Owned Semaphore*** to 0. Again, if BIOS has set the ***SMI on OS Ownership Enable*** in the ***USB Legacy Support Control/Status*** register to a 1 (See Table 5. USBLEGCTLSTS — USB Legacy Support Control/Status, for field definitions) it will receive an SMI when the operating system sets the ***HC OS Owned Semaphore*** to 0. Once the BIOS has detected that the operating system has relinquished control, it is then free to take control of the EHCI controller as appropriate. Once the operating system has relinquished control of the controller, it must not attempt to use the EHCI until it again requests ownership (as described above).

Note that this mechanism is intended only to ensure that an exchange of ownership of the host controller can be accomplished in a very deterministic and reliable manner.

5.2.1.2.1 OS Usage Model

Unlike the BIOS, the operating system **is** allowed to request ownership of the EHCI at anytime. Because the BIOS is not allowed to request ownership of the EHCI, the operating system does not need to implement any change in ownership detection strategies.

The following summarizes the operating system usage model for the ***HC OS Owned Semaphore***:

1. The operating system requests/gains ownership of the EHCI when it sets the ***HC OS Owned Semaphore*** to a 1. To insure that the BIOS completes the change in ownership (and to prevent a possible race condition), the operating system must not manipulate any of the EHCI registers until the ***HC BIOS Owned Semaphore*** is set to 0. After taking ownership of the EHCI, the operating system must initialize the EHCI (See Section *4.3.3.3*, *Post-Operating System Initialization*)) as the state of the EHCI must be considered unknown.
2. The operating system must release ownership of the EHCI whenever it is no longer being used.

Note: Because the ICH4/5 preserves the contents of the ownership semaphores across the system suspend and hibernate states and because the operating system *normally* will retain ownership of the EHCI throughout these states, it is therefore not required that the operating system re-request ownership of the EHCI (e.g., if the operating system had ownership prior to the suspend/hibernate state, it will have ownership upon return from those states). Consult your operating system specific DDK or appropriate documentation for proper power management behavior.

To release ownership of the EHCI, the operating system must:

a. Insure that neither the Asynchronous or Periodic schedules are being processed:
   - USB2CMD.Asynchronous Schedule Enable is set to 0.
   - USB2CMD.Periodic Schedule Enable is set to 0.

b. Insure that the EHCI is not running:
   - ***USB2CMD.Run/Stop*** is set to 0 AND
   - ***USB2STS.HCHalted*** is set to 1.

c. Set the interrupt enable bits 5:0 in **USB2INTR** to 0.

d. Set the ***USB2STS*** register bits are set to 0.

e. Set the ***Configure Flag*** register to 0.

f. Set the HC OS Owned Semaphore to 0.

This page is intentionally left blank.

6 Host Controller Reset

The ***USB2CMD.HCReset*** bit is used by system software to reset the host controller. A host controller reset can be used to place the host controller in a state that is deterministic (default) and reproducible. The effects of this on the root hub registers are similar to a Chip Hardware Reset (i.e., RSMRST# assertion and PWROK de-assertion on the ICH4/5).

When system software programs the ***USB2CMD.HCReset*** bit to a 1, the host controller resets its internal pipelines, timers, counters, state machines, etc. to their initial value. Any transactions currently in progress on the USB are immediately terminated. The host controller does not drive a port reset on the downstream ports during the host controller reset phase.

Note: A host controller reset does not affect the host controller PCI configuration registers and the host controller Capability Registers. A host controller reset will also cause the debug port registers to be returned to their default state.

All operational registers, including port registers and port state machines are set to their initial values. Port ownership reverts to the companion host controller(s), with the side effects described in the *Enhanced Host Controller Specification for Universal Serial Bus*.

To initiate a host controller reset system software must:

1. Stop the host controller. System software must program the ***USB2CMD.Run/Stop*** bit to 0 to stop the host controller.
2. Wait for the host controller to halt. To determine when the host controller has halted, system software must read the ***USB2STS.HCHalted*** bit; the host controller will set this bit to 1 as soon as the it has successfully transitioned from a running state to a stopped state (halted). Attempting to reset an actively running host controller will result in undefined behavior.
3. Program the ***USB2CMD.HostControllerReset*** bit to a 1. This will cause the host controller to begin the host controller reset.
4. Wait until the host controller has completed its reset. To determine when the reset is complete, system software must read the ***USB2CMD.HostControllerReset*** bit; the host controller will set this bit to 0 upon completion of the reset.

Note: System software ***cannot*** cause early termination of the reset by writing 0 to the ***USB2CMD.HostControllerReset*** bit; this is enforced by the hardware.

After the reset has completed, the system software must reinitialize the host controller so as to return the host controller to an operational state (See Section *4.3.3.3*, *Post-Operating System Initialization*)

System software must re-enumerate the bus after the controller has been reset and reinitialized, as the reset will have had an effect on the port routing logic (e.g., devices not routed to the appropriate host controller).

Resetting the host controller is a very “heavy-handed” process that should only be used by software as a last resort in the event that the host controller appears to be in an undefined state (e.g., schedules not being processed, port malfunction, etc).

This page is intentionally left blank.

7 Port States

Each port supported by the USB controller is required to implement a port status and control register (***PORTSC***). System software uses the ***HCSPARAMS.N_Ports*** information to determine how many ports will need to be serviced during the course of operation. System software must not write to unreported *Port Status and Control Registers*.

The ***PORTSC*** registers are implemented in the suspend power well and are only reset by hardware when the suspend power is initially applied or in response to a host controller reset. The initial conditions of a port are:

- No device connected
- Port disabled

System software uses the ***PORTSC*** registers for placing individual ports into specific states (e.g., suspend) and for detecting when ports have entered specific states (e.g., disconnect).

7.1 Detecting Port Changes

System software can detect when changes occur to each port 'owned' by EHCI system software through the use of the ***USB2STS.Port ChangeDetect*** bit and certain 'change' bits implemented in each individual port's ***PORTSC*** register.

7.1.1 Port Change Detect

The Host Controller sets this bit to a 1 when any port for which the ***PORTSC.PortOwner*** bit is set to 0 has a change bit transition from a 0 to a 1 or a port's ***PORTSC.ForcePortResume*** bit transitions from a 0 to a 1 as a result of a resume signaling detected on a suspended port.

7.1.2 Port Change Bits

The port change bits found in each valid ***PORTSC*** register provide indications to system software about port level events. The following port change bits are implemented for each accessible port:

- ***OverCurrentChange*** – Set to 1 whenever a port's ***OverCurrentActive*** bit experiences a change (0->1 – Over current condition present, 1->0 – Over current condition not present).
- ***PortEnable/DisableChange*** – Set to 1 whenever a port transitions from an enabled state to a disabled state (***PortEnabled/Disabled*** bit: 1->0). Never set to 1 when the port transitions from disabled to being enabled. *This bit is never set to 1 as a result of system software disabling a port.*
- ***ConnectStatusChange*** – Set to 1 whenever a port's ***CurrentConnectStatus*** bit experiences a change (0->1 – device connected, 1->0 – device disconnected).
- ***Force Port Resume*** – Set to 1 whenever resume signaling is detected at the port.

It is important to note that only a causal event will result in a corresponding change bit being set.

Examples of causal events are:

- Device disconnect
- Device Connect
- Over-Current (port disable)
- Port Errors (port disable)
- Force Port Resume (device initiated)

Consider the case where a suspended port experiences a surprise device removal by the user (e.g., device is disconnected). The result of a user disconnecting a device from a port will cause (at a minimum) the port to be *both* disconnected and disabled (***PORTSC.CurrentConnectStatus*** is set to 0 and ***PORTSC.PortEnable/Disable*** is set to 0). Although the port's ***PORTSC*** register implements bits (***PORTSC.ConnectStatusChange*** and ***PORTSC.PortEnable/DisableChange***) to reflect the change in the port's connect status and enabled status, as stated, only the ***PORTSC.ConnectStatusChange*** bit will be set, as the disconnect event is the causal event.

Note: System software must perform only DWORD accesses (i.e., 32bit) to the ***PORTSC*** registers. Also, when clearing a change bit, system software must perform a read/modified write to the **PORTSC** register in order to maintain its coherency. For example, to clear *only* the ***PORTSC.ConnectStatusChange*** bit, system software must perform a DWORD read of the ***PORTSC*** register, mask out the other change bit locations in the read value (set as 0), logically 'OR' a 1 into the bit location associated with the ***PORTSC.ConnectStatusChange*** bit and finally write back the value to the ***PORTSC*** register. The following 'C' code illustrates this example:

```
#define  OC_CHANGE   0x00000020 // Over-current change bit
#define  PED_CHANGE  0x00000008 // Port enable/Disable change bit
#define  CS_CHANGE   0x00000002 // Connect Status Change bit

#define  CHANGEBITMASK  OC_CHANGE | PED_CHANGE | CS_CHANGE

DWORD dwPortsc;      // local storage for the portsc register

dwPortsc = pEHCIBAR->Portsc0;   // read PORTSC for port 0
dwPortsc &= ~CHANGEBITMASK; // preserve all, make change bits 0
dwPortsc |= CS_CHANGE;      // set to clear
pEHCIBAR->Portsc0 = dwPortsc;   // clear connect status change bit
```

Detecting changes in each accessible port's change bits status can be accomplished using two different methods: Polling and Interrupts.

7.1.3 Interrupt Method

When using this method, software takes advantage of hardware's ability to generate an interrupt anytime a port, for which its ***PORTSC.PortOwner*** bit set to 0, has a change bit transition from a 0 to a 1.

To implement the interrupt driven method, software must:

1. Ensure that all interrupt sources are disabled. System software accomplishes this by writing a 0 to the ***USB2INTR*** register.
2. Clear the ***USB2STS*** register. System software accomplishes this by writing a 1 to bit locations 5:0 (i.e., 0x0000001f) of this register.
3. Issue a request to the operating system to 'hook' the interrupt assigned to the EHCI. Using a WDM (*Windows Driver Model*) driver as an example, the device's interrupt assignment information is provided by the operating system when it executes the software's IRP_MN_START_DEVICE handler.

Note: The EHCI uses a single interrupt for multiple event notifications and therefore must only be 'hooked' once by software. Since the associated interrupt can also occur for *non-port change* events (e.g., Frame List Rollover, Resume, Error, etc), the hooked ISR must also be prepared to handle these events if they a have been enabled in the ***USB2INTR*** register (See Section 9 - *Hardware Interrupt Routing*).

4. Program the ***USB2INTR.PortChangeInterruptEnable*** bit so that the EHCI will generate an interrupt whenever any of the port change bits transition from a 0 to a 1. Note that the ***USB2INTR.PortChangeInterruptEnable*** bit if set will cause an interrupt to be generated when *any* of the port's change bits (***PORTSC***) perform a 0 -> 1 transition.
5. Check the ***USB2STS.PortChangeDetect*** bit when software's interrupt service routine (ISR) is called. If this bit is set to 1, then a port has changed state and system software must read and examine the change bits in each valid port's ***PORTSC*** register to determine each port's state and to handle it appropriately. Consult your Operating System documentation for ISR specifics. Before exiting the ISR, system software must clear ***USB2STS.PortChangeDetect*** by writing a 1 to this bit., as this will prevent the hardware from generating another interrupt for the same event.

7.1.4 Polled Method

In the polled method, software does not take advantage of the EHCI's ability to generate an interrupt whenever a port, for which its ***PORTSC.PortOwner*** bit set to 0, has a change bit transition from a 0 to a 1. Instead software chooses to examine the state of the ***PORTSC.PortChangeDetect*** on a regular, timed basis. To implement the polled method, software must:

1. Ensure that all interrupt sources are disabled. System software accomplishes this by writing a 0 to the ***USB2INTR*** register.
2. Clear the ***USB2STS*** register. System software accomplishes this by writing a 1 to bit locations 5:0 (i.e., 0x0000001f) of this register.
3. Implement a mechanism whereby software can read the ***USB2STS.PortChangeDetect*** on a periodic basis. Each time the polling software is executed it must:

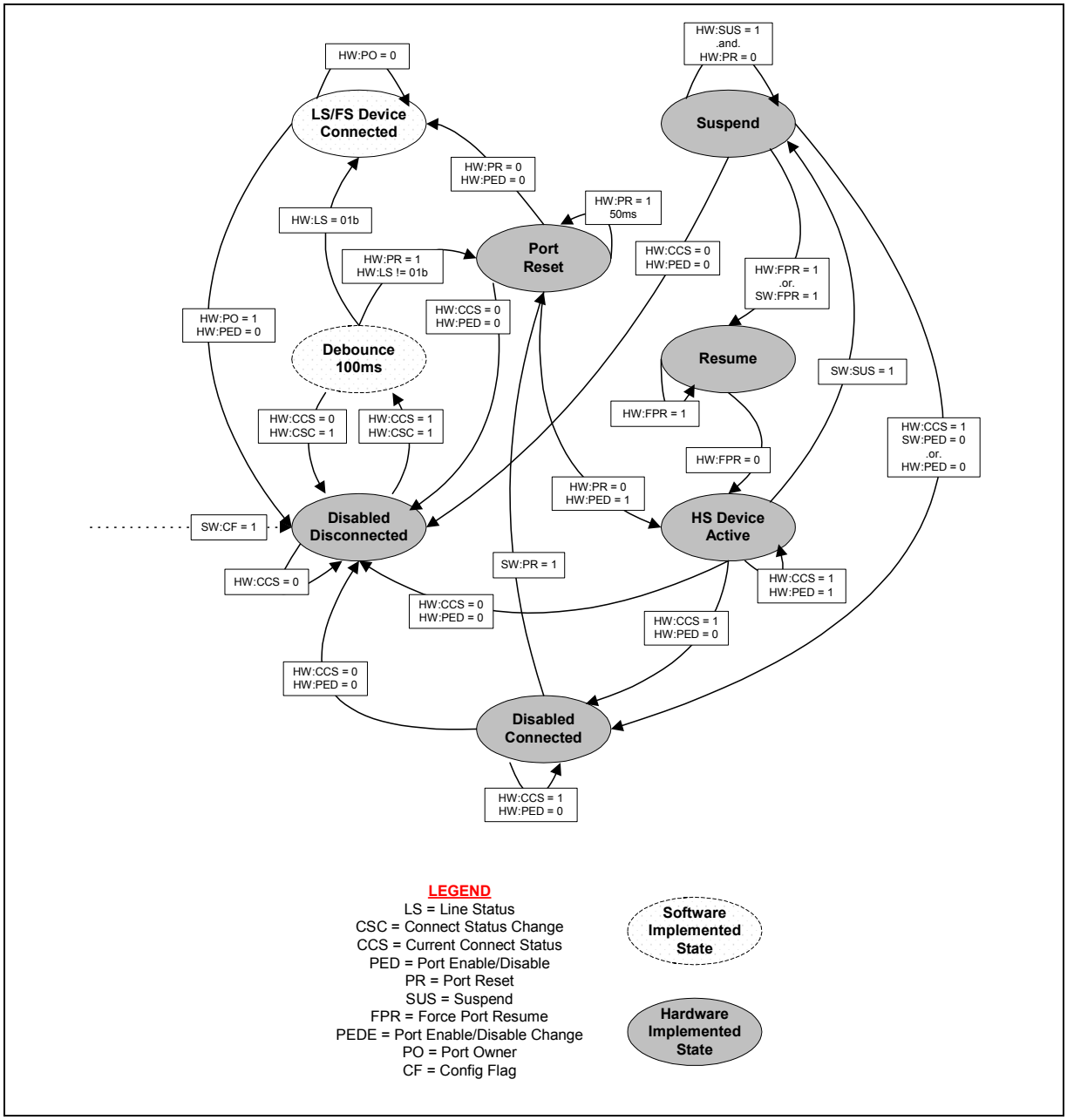
a. Read the ***USB2STS.PortChangeDetect***. If this bit is set (1), then system software must read and examine the change bits in each valid port's ***PORTSC****l* register to determine each port's state and handle appropriately.

b. Clear ***USB2STS.PortChangeDetect*** by writing a 1 to this bit. This will insure that the software polling function will be able to detect future port change events.

7.2 Port States

Figure 6. Port State Diagram illustrates the state diagram for managing each USB port. Note that the USB2 host controller can enter any of these states *after* it has been properly initialized by system software (BIOS or OS) as described in the previous sections.

Figure 6. Port State Diagram



7.3 Disabled Disconnected State

A disabled (***PORTSC.PortEnable/Disabled*** = 0***)*** port is not capable of receiving or transmitting data across the USB. The ***Disabled Disconnected*** state is the default state of all ports following initial system power-on or after a host controller reset has been executed and the ***Configure Flag*** register is programmed to 1. While a device may be connected to a port during host controller reset or initial power on, the port will remain in the disconnected state until system software has had the opportunity to detect changes to each port's ***PORTSC.ConnectStatusChange*** bit.

A port will transition to the ***Disabled Disconnected*** state from the following states:

- ***Configured*** state (See Section *4.3.3.3*, *Post-Operating System Initialization*))
- ***HS Device Active*** state
- ***LS/FS Connected*** state
- ***DeBounce*** state
- ***Port Reset*** state

A port remains in the ***Disabled Disconnect*** state until a connection event occurs (***PORTSC.CurrentConnectStatus*** set to 1).

7.3.1 Initial Transition to the Disabled Disconnect State

Before the EHCI portion of the USB2 controller is usable by EHCI specific system software, it is a requirement that system software program the ***Configure Flag*** to 1. System software must program this bit as the last action in its process of configuring the host controller (Refer to Section *4.2 – Port Routing and Control* in the *Enhanced Host Controller Specification for Universal Serial Bus*).

As stated previously, after programming this bit to 1, all USB2 ports will automatically route to the EHCI and must be considered in the ***Disabled Disconnected*** state. If devices are connected at the time the ***Configure Bit*** is programmed to 1, then the host controller will indicate this by setting the following bits to 1:

- ***USB2STS.PortChangeDetect*** - indicates a change on one or more ports
- ***PORTSC.CurrentConnectStatus*** – indicates a device has been connected to the port
- ***PORTSC.ConnectStatusChange*** – indicates the port transitioned from disconnected to connected

If the above conditions are true, then the port will transition to the ***Debounce*** state.

If the above conditions are not true, then the port(s) will remain in the ***Disabled Disconnected*** state.

7.3.2 Transitioning to the Disabled Disconnect State

A port will transition to the ***Disabled Disconnected*** state from the following states whenever the port detects that a device previously connected to the port has been removed (disconnected). It can be entered from the following previous states:

- HS Device Active state
- Disabled Connected state
- Port Reset state
- Suspend state
- LS/FS Device Connected state

System software can detect when a port experiences this transition based on the setting of the following bits:

- ***USB2STS.PortChangeDetect*** is set to 1 to indicate that the port experienced a change event.
- ***PORTSC.CurrentConnectStatus*** is set to 0 to indicate the absence of a device on the port.
- ***PORTSC.ConnectStatusChange*** is set to 1 to indicate a change to ***PORTSC.CurrentConnectStatus***.

When a port transitions to the ***Disabled Disconnected*** state, system software must:

- Clear the ***PORTSC.ConnectStatusChange*** by writing a 1 to this bit.
- Clear the ***USB2STS.PortChangeDetect*** by writing a 1 to this bit.

To prevent the EHCI from needlessly accessing main memory when ***all*** ports are in the ***Disabled Disconnected*** state, system software must:

- Disable the periodic and asynchronous schedules. Software does this by writing 0 to the ***USB2CMD.PeriodicScheduleEnable*** bit and the ***USB2CMD.AsynchronousScheduleEnable*** bits respectively. For more details, see Section 12 - *Pausing the Transaction Schedules (Entering sub-C0 states)*

7.3.3 Transitioning to the Disabled Disconnected State from the DeBounce State

System software transitions a disabled port out of the ***DeBounce*** state to the ***Disabled Disconnected*** state when, at the end of the debounce interval, system software detects that the ***PORTSC.CurrentConnectStatus*** bit is set to 0. System software effects this transition by clearing the ***PORTSC.ConnectStatusChange*** bit.

Note: System software must not clear the ***PORTSC.ConnectStatusChange*** bit, nor consider the ***PORTSC.CurrentConnectStatus*** bit valid until debounce interval has expired.

To transition a port to the ***Disabled Disconnect*** state from the ***DeBounce*** state, system software must:

- Clear the ***PORTSC.ConnectStatusChange*** by writing a 1 to this bit.
- Clear the ***USB2STS.PortChangeDetect*** by writing a 1 to this bit.

7.3.4 Transitioning to the Disabled Disconnected State from the LS/FS Device Connected State

System software transitions the port from the ***Disabled Disconnected*** state from the ***LS/FS Device Connected*** state by setting the ***PORTSC.PortOwner*** with a 1. Programming the ***PORTSC.PortOwner*** bit to 1 causes port ownership to revert to one of the appropriate UHCI host controllers implemented in the ICH4/5.

Hardware acknowledges the transition by setting the following bits:

- ***USB2STS.PortChangeDetect*** is set to 1 to indicate that the port experienced a change event.
- ***PORTSC.CurrentConnectStatus*** is set to 0 to indicate the absence of a device on the port.
- ***PORTSC.ConnectStatusChange*** is set to 1 to indicate a change occurred to ***PORTSC.CurrentConnectStatus***.
- ***PORTSC.PortEnabled/Disabled*** is set to 0 to indicate that the port is disabled.
- ***PORTSC.PortOwner*** is set to 1 to indicate that a companion UHCI host controller owns the port.

When a port transitions to the ***Disabled Disconnected*** state from the ***LS/FS Device Connected*** state, system software must:

- Clear the ***PORTSC.ConnectStatusChange*** by writing a 1 to this bit.
- Clear the ***USB2STS.PortChangeDetect*** by writing a 1 to this bit.

7.4 DeBounce State

The ***DeBounce*** state ensures that the mechanical and electrical connection (of a newly connected device) is stable before system software causes the port to transition from the ***Disabled Disconnected*** state to either the ***Port Reset*** state or the ***LS/FS Device Connected*** state. This state is not implemented by hardware; rather it is implemented by system software and ensures that power is stable at the device during the ***DeBounce*** interval (this is interval is minimally 100 ms as specified in Section *7.1.7.3* of the *Universal Serial Bus Specification Revision 2.0*). Furthermore, the ***DeBounce*** state:

- Ensures that a detected connect event detected by the port is genuine
- Ensures that a detected connect event is not the result of noise
- Properly detects back- to-back device connect-disconnects
- Ensures that the line status indicators (***PORTSC.LineStatus***) are stable.

A port may only transition to this state from the ***Disabled Disconnected*** state. It *must* successfully complete (i.e., ***PORTSC.CurrentConnectStatus*** remains 1 throughout the interval) before the port is allowed to transition to either the ***Port Reset*** state or the ***LS/FS Device Connected*** state. Failure to complete this state will cause the port to revert back to the ***Disabled Disconnected*** state.

7.4.1 Transitioning to the DeBounce State from the Disabled Disconnected State

System software transitions a *disabled* port to the ***DeBounce*** state from the ***Disabled Disconnected*** state when the port detects a possible device connection event.

System software can detect when a port should transition to the ***DeBounce*** state by the following bits:

- ***USB2STS.PortChangeDetect*** is set to 1 to indicate that the port experienced a change event.
- ***PORTSC.CurrentConnectStatus*** is set to 1 to indicate the possible presence of a device on the port.
- ***PORTSC.ConnectStatusChange*** is be set to 1 to indicate a change occurred to ***PORTSC.CurrentConnectStatus***.

When a port transitions to the ***DeBounce*** state from the ***Disabled Disconnect*** state, system software must:

- Clear the ***PORTSC.ConnectStatusChange*** by writing a 1 to this bit.
- Clear the ***USB2STS.PortChangeDetect*** by writing a 1 to this bit.
- Wait the specified ***Debounce*** interval (100 ms) before re-checking the ***PORTSC.CurrentConnectStatus*** bit and the ***PORTSC.LineStatus*** bits. If the ***PORTSC.CurrentConnectStatus*** bit is not 1, then a device is not connected and the port must transition back to the ***Disabled Disconnected*** state. If the ***PORTSC.CurrentConnectStatus*** bit remained a 1 throughout the ***DeBounce*** state interval, then system software must:
  - Check the PORTSC.LineStatus bits. If the PORTSC.LineStatus bits do not read as ‘01b’ (not low-speed device), then system software must cause the port to advance to the Reset State by programming the PORTSC.PortReset bit to 1. If the PORTSC.LineStatus bits do read as ‘01b’ (low-speed device), then a port reset must not be performed and system software must advance to the LS/S Device Connected state.

7.4.2 Over Current Considerations

After transitioning to this state (from the ***Disabled Disconnected*** state) and prior to entering the ***Port Reset*** state, if an over current condition is active (***PORTSC.OverCurrentActive*** is 1) following the debounce interval, then system software must not attempt to reset the port; the port immediately transitions to the ***Disabled Connected*** state. System software must wait until the over current condition has cleared (***PORTSC.OverCurrentActive*** is 0) before the port is allowed to transition to the Port Reset state.

7.5 LS/FS Device Connected State

This state is not implemented by hardware; it is implemented by system software and is used to route (transfer ownership) a port to one of the companion UHCI controllers implemented in the ICH4/5. This state is only entered when system software determines (in concert with the EHCI) that a newly connected device is not a high-speed USB2 device, but rather, is either a low-speed or full-speed USB1.1 device. Because USB1.1 devices cannot operate at the signaling rates required

of USB2, it is required that system software cause the port that is hosting the device to be routed to a UHCI controller.

To route a port to a companion controller, system software must program the ***PORTSC.PortOwner*** bit to 1. Once the port is routed, the device-connect evaluation circuitry of the companion host controller activates and detects the device the companion controller hardware/software detects the connection and enumerates the port. *When a port is routed to a companion host controller, it remains under the control of the companion host controller until the device is disconnected from the root port.*

Note: When a full or low speed device is connected and ownership is transferred to a companion controller, EHCI system software should be prepared to detect and process a port change indication on the port since this action essentially causes the EHCI to experience a port disconnect event.

When a device is disconnected from a companion controller owned port, ownership of the port automatically reverts back to the EHCI.

The exact behavior of the EHCI after this bit is set to 1 is outlined within the *Enhanced Host Controller Specification for Universal Serial Bus.*

A port can transition to the ***LS/FS Device Connected*** state from the following states:

- Port Reset state
- DeBounce state

7.5.1 Transitioning to the LS/FS Device Connected State from the DeBounce State

System software transitions a *disabled* port to the ***LS/FS Device Connected*** state from the ***DeBounce*** state when system software detects the following bits:

- ***PORTSC.CurrentConnectStatus*** is set to 1 indicating a connected device.
- ***PORTSC.LineStatus*** is set to '01b', indicating that the device connected to the port is a USB1.1 *low speed* device.

When a port transitions to the ***LS/FS Device Connected*** state from the ***DeBounce*** state, system software must initiate the transition to the ***Disabled Connected*** by:

- Transfer ownership to a companion UHCI controller by programming the ***PORTSC.PortOwner*** bit to 1.

Note: Once ownership of a port is transferred to a companion UHCI controller, the device-connect evaluation circuitry of the companion host controller will activate and detect the device, thereby causing the companion controller hardware/system software to detect the connection and enumerate the port. When a port is routed to a companion host controller, it remains under the control of the companion host controller until the device is disconnected from the root port. Refer to Section *4.2.2.Port Routing Control via PortOwner and Disconnect Event* in the *Enhanced Host Controller Specification for Universal Serial Bus* for more detailed information.

7.5.2 Transitioning to the LS/FS Device Connected State from the Port Reset State

A *disabled* port will transition to the ***LS/FS Device Connected*** state from the ***Port Reset*** state when system software completes the reset sequence, and the EHCI fails to detect a high-speed device on the port. The following bits are indicative of this state:

- ***PORTSC.CurrentConnectStatus*** is set to 1 indicating a connected device.
- ***PORTSC.PortEnabled/Disabled*** is set to 0, indicating that following the port reset (***Port Reset*** state), the port *was not* enabled and therefore cannot be a high-speed device.

When a port transitions to the ***LS/FS Device Connected*** state from the ***Port Reset*** state, system software must:

- Transfer port ownership to a companion UHCI controller by programming the ***PORTSC.PortOwner*** bit to 1.

7.6 Port Reset State

This state asserts reset signaling on the bus as defined in Section *7.1.7.5 – Reset Signaling* in the *Universal Serial Bus Specification Revision 2.0*. System software *must* not reset a port that is host to a low speed device. A port must be reset after initially being connected (successful transition from the ***DeBounce*** state) or as a means of waking a connected device from the suspend state. If during the port reset sequence, a device successfully returns the *chirp* sequence, the host controller set the ***PORTSC.PortEnabled/Disabled*** bit to 1, thus indicating that a high-speed device is attached. If the chirp sequence is not successfully returned, then the host controller does not set the ***PORTSC.PortEnabled/Disabled*** to 1. This indicates that the attached device is a classic full-speed device. Refer to Section *7.1.7.5 – Reset Signaling* in the *Universal Serial Bus Specification Revision 2.0* for details.

A port can transition to the ***Port Reset*** state from the following port states:

- DeBounce state
- Disabled Connected state

7.6.1 Transitioning to the Port Reset State from the DeBounce State, or Disabled Connected State

System software transitions the port to the ***Port Reset*** state from the ***DeBounce*** or ***Disabled Connected*** state by setting the ***PORTSC.PortReset*** bit to 1. This begins the port-reset sequence. Note that software is responsible for timing the duration of the reset signal (nominally 50 ms, see Section *7.1.7.5 – Reset Signaling* in the *Universal Serial Bus Specification Revision 2.0*).

After system software has initiated the port reset sequence, system software must:

Note: It is a rule (as per Section *10.5.4.5 – Managing Remote Wakeup Devices* if the *Universal Serial Bus Specification Revision 2.0*) that system software must not reset a *suspended* port that hosts a device that has been enabled for remote wake up. System software is required to first enable the port (force port resume) before beginning the reset sequence. This rule does not apply to devices

that *are not* armed for remote wake; however, system software must ensure that the port is *disabled* before initiating the port reset.

1. Begin timing the reset sequence after successfully reading a value of 1 from the ***PORTSC.PortReset*** bit. Reset signaling on the bus does not occur until the ***PORTSC.PortReset*** bit is set to 1 by host controller.
2. Terminate the reset sequence after the spec-ed amount of time. System software terminates the port-reset sequence by programming the ***PORTSC.PortReset*** bit with a 0. The reset process is actually complete when the host controller sets the ***PORTSC.PortReset*** bit to 0.
3. Determine if the port is enabled and/or connected by examining the ***PORTSC.PortEnabled/Disabled*** bit. If this bit is 1, then the attached device is a USB2 high-speed device and the port has transitioned to the ***HS Device Active*** state. If the ***PORTSC.PortEnabled/Disabled*** bit is 0 and the ***PORTSC.CurrentConnectionStatus*** is 1, then the attached device is not a USB2 high-speed device (USB1.1. full-speed) and the port has transitioned to the ***LS/FS Device Connected*** state. If the ***PORTSC.PortEnabled/Disabled*** bit is 0 and the ***PORTSC.CurrentConnectionStatus*** is 0, then no device is attached (disconnected during the port reset sequence) and the port has transitioned to the ***Disabled Disconnected*** state.

Note: As per Section *7.1.7.5 – Reset Signaling* in the *Universal Serial Bus Specification Revision 2.0,* it is a requirement that after a port is reset and enabled, system software must provide a 'recovery' interval of 10 ms before a device attached to the port is expected to respond to data transfer.

7.6.2 Over Current Considerations

After transitioning to this state (from the ***Debounce*** state or ***Disabled Connected*** state), if an over current event occurs on the port (***PORTSC.OverCurrentChange*** is 1) during the port reset interval or if the over current condition is still active (***PORTSC.OverCurrentActive*** is 1) after the port reset interval, then the port will not be enabled and will immediately transition to the ***Disabled Connected*** state. System software must wait until the over current condition has cleared (***PORTSC.OverCurrentActive*** is 0) before the port is allowed to transition to the ***Port Reset*** state.

7.7 HS Device Active State

While a port is in this state, system software can initiate transactions with a device. Because enumeration involves transmitting asynchronous control packets over the bus, software must remember to program the ***USB2CMD.AsynchronousScheduleEnable*** bit to 1, to enable processing of the asynchronous schedule by the host controller.

Note: Software should enable the asynchronous schedule *only* after a valid pointer to a queue head(s) has been programmed into the ***ASYNCLISTADDR*** register. To prevent the host controller from needlessly accessing memory, software should disable the asynchronous schedule after all scheduled transactions have been retired.

If after enumeration, system software determines that the connected device supports isochronous or interrupt type transactions, then system software must remember to program the ***USB2CMD.PeriodicScheduleEnable*** bit to 1 to enable processing of the periodic frame list (if it has not already done so).

Note: All unused elements in the frame list and the last next pointer in the interrupt tree must have their t-bits set to 1 before the periodic schedule is enabled. To prevent the host controller from needlessly accessing memory, system software should keep the periodic schedule enabled only when valid, active transactions are present.

A port can transition to the ***HS Device Active*** state from the following port states:

- Port Reset state
- Resume state

7.7.1 Transitioning to the HS Device Active State from the Resume State or the Port Reset State

A port transitions to the ***HS Device Active*** state from the ***Port Reset*** state or ***Resume*** state when the following are true:

- ***PORTSC.CurrentConnectStatus*** is set to 1 indicating a connected device.
- ***PORTSC.PortEnabled/Disabled*** is set to 1, indicating that the connected port is enabled.
- ***PORTSC.Suspend*** is set to 0, indicating that the port is active (not suspended).

When a port transitions to the ***HS Device Active*** state system software must:

- Clear the ***PORTSC.PortEnable/DisableChange*** by writing a 1 to this bit.
- Clear the ***USB2STS.PortChangeDetect*** by writing a 1 to this bit.

A port will remain in the ***HS Device Active*** state until:

- The port transitions to the ***Suspend*** state via system software initiation (i.e., system software programs ***PORTSC.Suspend*** to 1).
- The port transitions to the ***Disabled Connected*** state due to an Over-Current condition.
- The port transitions to the ***Disabled Connected*** state due to a frame babble condition.
- The port transitions to the ***Disabled Disconnected*** state due to a device disconnect event.

7.7.2 Over Current Considerations

If the port experiences an over-current condition in this state, then the port immediately transitions to the ***Disabled Connected*** state (***PORTSC.PortEnabledDisabled*** is 0). System software must wait until the over current condition has cleared (***PORTSC.OverCurrentActive*** is 0) before the port is allowed to transition to the ***Port Reset*** state.

7.8 Disabled Connected State

A port in the ***Disabled Connected*** state is not capable of receiving or transmitting data across the USB. An enabled, connected port transitions to this state as a result of the port being disabled by:

- System software programming the ***PORTSC.PortEnabled/Disabled*** bit to 0.
- An over-current condition (***PORTSC.OverCurrentActive*** is 1).
- A port error occurs (Refer to Section *11.8.1-Port Error* in the *Universal Serial Bus Specification Revision 2.0* for exact details).

A port can only transition to this state from the following states:

- Suspend state
- HS Device Active state

A port will remain in the ***Disabled Connected*** state until:

- The over-current condition is remedied AND …
- The port transitions to the ***Port Reset*** state (i.e., system software programs the ***PORTSC.PortReset*** bit to 1).
- The port experiences a device disconnection (e.g., user disconnects babbling device).

7.8.1 Transitioning to the Disabled Connected State from the HS Device Active State

A port will transition to the ***Disabled Connected*** state from the ***HS Device Active*** state whenever the port becomes disabled as a result of some hardware event or fault.

System software can detect when a port experiences this transition based on the setting of the following bits:

- ***USB2STS.PortChangeDetect*** is set to 1 to indicate that the port experienced a change event.
- ***PORTSC.CurrentConnectStatus*** is set to 1 to indicate the presence of a device on the port.
- ***PORTSC.PortEnabled/Disabled*** is set to 0 to indicate that the port is disabled.
- ***PORTSC.PortEnable/DisableChange*** is set to 1 to indicate a change to ***PORTSC.PortEnabled/Disabled***.

When a port transitions to the ***Disabled Connected*** state system software must:

- Clear the ***PORTSC.PortEnable/DisableChange*** by writing a 1 to this bit.
- Clear the ***USB2STS.PortChangeDetect*** by writing a 1 to this bit.

If a port transitions to the ***Disabled Connected*** state as a result of an over-current event, then system software must additionally clear the ***PORTSC.OverCurrentChange*** by writing a 1 to this bit.

7.8.2 Transitioning to the Disabled Connected State from the Suspend State

System software transitions a port to the ***Disabled Connected*** state from the ***Suspend*** state by setting the ***PORTSC.PortEnabled/Disabled*** bit to 0 to indicate the port is disabled.

Note: If system software disables a port with the intent of immediately resetting the port, system software is allowed to accomplish this with a single write to the ***PORTSC.PortEnable/Disable*** and ***PORTSC.PortReset*** bits, as the hardware will guarantee atomicity. For example, programming the ***PORTSC.PortReset*** and ***PORTSC.PortEnable/Disable*** bits with a single write has the same net effect as individually programming the ***PORTSC.PortEnable/Disable*** bit followed by the ***PORTSC.PortReset*** bit. In either case, the host controller guarantees that the port will: Transition *from* the ***Suspend*** state to the ***Disabled Disconnect*** state and then from the ***Disabled Disconnect*** state to the ***Port Reset*** state.

7.8.3 Over Current Considerations

If an over condition is active when a port is in this state, it cannot transition to the Port Reset state. Therefore, system software must not attempt to reset a port when a port is in this state and an over current condition (PORTSC.OverCurrentActive is 0) is active.

7.9 Suspend State

When a port has transitioned to this state, downstream propagation of data is blocked on the port (except for port reset), however the port will still respond to resume signaling initiated by a device. This state may only be entered from the ***HS Device Active*** state and is initiated by system software as part of its power conservation policy.

7.9.1 Transitioning to the Suspend State from the HS Device Active State

System software transitions the port to the ***Suspend*** state from the ***HS Device Active*** state by:

1. 'Arming' all applicable downstream devices for remote wake. Because system software is allowed to arm a device(s) for remote wake in the HS Device Active state; this step may not be required.

Note: Devices capable of remote wake up must have remote wake up specifically enabled before they can wake up a suspended port. Refer to Section *9.4 – Standard Device Requests* in the *Universal Serial Bus Specification Revision 2.0.*

2. Ensuring all active transaction descriptors on the periodic and asynchronous schedules have been removed. Failure to do so will cause the host controller to attempt to process these transactions and could result in errors (e.g., timeout) being reported to system software.
3. Programming the ***PORTSC.Suspend*** bit with a 1.
4. Reading back a 1 in the ***PORTSC.Suspend*** bit. The host controller sets the ***PORTSC.Suspend*** bit when the port has been successfully suspended. System software must not consider the port to be in the ***Suspend*** state until the host controller sets this bit to 1. System software must poll

this bit before continuing (i.e., reads as 1). Note that the host controller ignores a write of 0 to the ***PORTSC.Suspend*** bit.

The port will remain in the ***Suspend*** state until one of the following occurs:

- System software resets the port (programs ***PORTSC.PortReset*** to 1).
- System software forces resume signaling to be driven on the bus (system software programs ***PORTSC.ForcePortResume*** to 1).
- A USB device asserts resume signaling on the bus (host controller sets ***PORTSC.ForcePortResume*** to 1).
- An over-current condition occurs.

7.9.1.1 Over Current Considerations

If the port experiences an over-current event during this state, then the port immediately transitions to the Disabled Connected state (PORTSC.Suspend is 0, PORTSC.PortEnabledDisabled is 0). System software must wait until the over current condition has cleared (PORTSC.OverCurrentActive is 0) before the port is allowed to transition to the Port Reset state.

7.10 Resume State

The ***Resume*** state is a transitive state that a port enters after leaving the ***Suspend*** state but prior to it entering the ***HS Device Active*** state. A port can only enter this state from the ***Suspend*** state.

A port can transition from the ***Suspend*** state to the ***Resume*** state via:

- System software initiation. System software can initiate resume signaling on the bus by programming the ***PORTSC.ForcePortResume*** bit to 1.

Note: A port must remain in the ***Suspend*** state for at least 10 ms before system software may initiate a resume.

- Device initiation. A device connected to a port in the ***Suspend*** state, drives resume signaling on the bus (host controller will set ***PORTSC.ForcePortResume*** to 1).

7.10.1 Resuming

Note: As per the *Universal Serial Bus Specification Revision 2.0,* it is a requirement that after a port is resumed system software is expected to provide a ‘recovery’ interval of 10 ms before a device attached to the port is expected to respond to data transfer.

7.10.1.1 Software Initiation

To place a port in the ***Resume*** state and to initiate resume signaling on the bus, system software must:

1. Program the ***PORTSC.ForcePortResume*** bit to 1.
2. Allow the EHCI to drive resume for the interval specified in Section *7.1.7.7- Resume* in the *Universal Serial Bus Specification Revision 2.0.* System software must insure that resume signaling is driven on the bus for required amount of time (this is stated as at least 20 ms).
3. Instruct the host controller to quit driving resume signaling after the specified resume interval. System software accomplishes this by programming the ***PORTSC.ForcePortResume*** bit to 0.

Note: When system software initiates resume signaling on the bus, the ***USB2STS.PortChangeDetect*** bit does not get set.

After the resume sequence has completed, the port will immediately transition to the ***HS Device Active*** state.

7.10.1.2 Device Initiation

Device initiated port resume occurs when a device on the USB that has been armed for 'remote wake', causes resume signaling to be driven on the bus. Resume signaling alerts the host controller that the associated port should transition from the ***Suspend*** state to the ***Resume*** state and that system software should determine/service the event that caused the suspended device to wake. While any enabled port can be suspended and resumed, not all devices support remote wake and it is the responsibility of system software to make this determination.

System software can detect when a port transitions to the ***Resume*** state based on the setting of the following bits:

- ***USB2STS.PortChangeDetect*** is set to 1 to indicate that the port experienced a change event.
- ***PORTSC.ForcePortResume*** is set to 1 to indicate that a device is driving resume signaling on the bus.

When a port transitions to the ***Resume*** state, system software must:

1. Clear the ***USB2STS.PortChangeDetect*** by writing a 1 to this bit.
2. As specified in Section *7.1.7.7 - Resume* in the *Universal Serial Bus Specification Revision 2.0*, system software must insure that resume signaling is driven on the bus for required amount of time (this is stated as at least 20 ms).
3. System software must instruct the host controller to quit driving resume signaling after the specified amount of time. System software accomplishes this by writing a 0 to the ***PORTSC.ForcePortResume*** bit.

After the resume sequence has completed, the port will immediately transition to the ***HS Device Active*** state.

7.10.2 Over Current Considerations

If the port experiences an over-current condition during this state, then the port immediately transitions to the ***Disabled Connected*** state (***PORTSC.PortEnabledDisabled*** is 0). System

software must wait until the over current condition has cleared (***PORTSC.OverCurrentActive*** is 0) before the port is allowed to transition to the ***Port Reset*** state.

8 64-bit Addressing

The ICH4/5 uses 64-bit addressing to locate all data structures in memory. This is accomplished when software programs the ***CTRLDSSEGMENT*** register with a value that corresponds to the most significant address bits [63:32] used for all EHCI data structures.

To construct a 64-bit address, the hardware takes the value programmed in the ***CTRLDSSEGMENT*** and combines it with the address specified by any of the following addresses:

- ASYCLISTADDR
- PERIODICLISTBASE
- Any pointer to any data structure defined in the Enhanced Host Controller Specification For Universal Serial Bus:
  - 64-bit Isochronous Transaction Descriptor.
  - 64-bit Split Transaction Isochronous Transaction Descriptor.
  - 64-bit Queue Element Transaction Descriptor.
  - 64-bit Queue Head Descriptor.

From a software point of view, the computation of the 64-bit address is equivalent to the following code fragment:

```
LARGE_INTEGER_64      Computed64bitAddr;
//
// This sets the upper 32 bit address (in CTRLDSSEGMENT) into the
// upper 32 bits of Computed64bitAddr and sets the lower 32 bits
of // the specified address (ASYNCLISTADDR in this example) to
form a
// 64bit physical memory address
//
Computed64bitAddr = (CTRLDSSEGMENT << 32) | ASYNCLISTADDR;
```

Thus, if ***CTRLDSSEGMENT*** is programmed by software as 0xDEADBEEF and ***ASYNCLISTADDR*** is programmed by software as 0x00ABACAB, then the computed value for *Computed64bitAddr* is: 0xDEADBEEF00ABACAB. Therefore, once ***CTRLDSSEGMENT*** is programmed, all subsequent data structures allocated by the system software may occupy memory anywhere in the 4GB range as long as the upper 32 bits of the address range are identical.

Note: For operating systems that do not support 64-bit addresses, software must program the ***CTRLDSSEGMENT*** register to zero (0x00000000). Also, all *Extended Buffer Pointer Page n* values in the EHCI 64-bit data structures must also be programmed to zero.

This page is intentionally left blank.

9 Hardware Interrupt Routing

The USB2.0 controller in the ICH4/5 uses PCI IRQ (PIRQ) INTD#. Consistent with the PCI specification, this is a shared, level-triggered interrupt.

Because this interrupt can be shared amongst similar and dissimilar devices, it is reasonable to expect that multiple system software device drivers will 'hook' this interrupt, for purposes of interrupt notification. System software can easily determine if the generated interrupt originated from the EHCI; reading and examining the ***USB2STS*** register accomplish this. If any of the ***USB2STS*** bits 5:0 are 1, then the EHCI is a source of the interrupt. Otherwise, the EHCI is not the interrupt source and system software must indicate this in a manner that is appropriate (See your Operating System device driver kit documentation or equivalent for specific details) for the operating environment.

Before system software can process EHCI related interrupt, it must first program the ***USB2INTR*** register to enable the EHCI to generate an interrupt for events of some interest to the system software.

9.1 USB2INTR- USB2 Interrupt Enable Register

This register enables and disables reporting of the corresponding interrupt to the software. When a bit in this register is set (1) and the corresponding interrupt is active, an interrupt will be generated (and indicated accordingly in the ***USB2STS*** register.

Note: Interrupt sources (events) that are disabled in this register will still appear in the ***USB2STS*** register. This enables system software to use time-based algorithms (polling) instead of/in addition to an interrupt based method.

Each interrupt enable bit description indicates whether it is dependent on the interrupt threshold mechanism (see Section 4 of the *Enhanced Host Controller Interface Specification for Universal Serial* Bus) or not.

Table 8 summarizes the USB2 Interrupt Enable register:

Table 8. USB2 Interrupt Enable Register

Bit	Description
31:6	**Reserved**. These bits are reserved and should be 0.
5	**Interrupt on Async Advance Enable.** When this bit is a 1, and the *Interrupt on Async Advance* bit in the USBSTS register is a 1, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the *Interrupt on Async Advance* bit.
4	**Host System Error Enable.** When this bit is a 1, and the *Host System Error Status* bit in the USB2STS register is a 1, the host controller will issue an interrupt. The interrupt is acknowledged by software clearing the *Host System Error* bit.
3	**Frame List Rollover Enable.** When this bit is a 1, and the *Frame List Rollover* bit in the USB2STS register is a 1, the host controller will issue an interrupt. The interrupt is acknowledged by software clearing the *Frame List Rollover* bit.
2	**Port Change Interrupt Enable.** When this bit is a 1, and the Port Change Detect bit in the USB2STS register is a 1, the host controller will issue an interrupt. The interrupt is acknowledged by software clearing the *Port Change Detect* bit.
1	**USB Error Interrupt Enable.** When this bit is a 1, and the USBERRINT bit in the USB2STS register is a 1, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software by clearing the *USBERRINT* bit in the USB2STS register.
0	**USB Interrupt Enable.** When this bit is a 1, and the USBINT bit in the USB2STS register is a 1, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software by clearing the *USBINT* bit in the USB2STS register.

10 Periodic and Asynchronous Transaction Schedules

The ICH4/5 EHCI uses scatter gather mechanisms to access memory. There are two 2 DMA engines incorporated into the EHCI: one engine for processing the periodic transaction schedule and one for processing the asynchronous transaction schedule.

10.1 Schedule Traversal

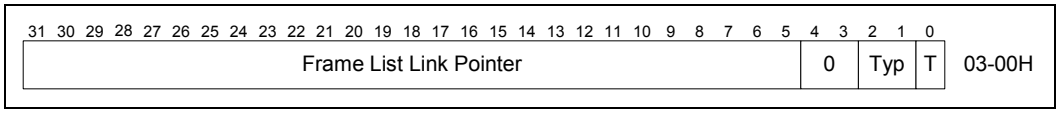
The host controller executes transactions for devices using a simple, shared-memory schedule. The schedule is comprised of a few data structures, organized into two distinct lists. System software maintains two schedules for the host controller: a periodic schedule and an asynchronous schedule. The periodic schedule is used by the EHCI for processing isochronous and interrupt transactions. The asynchronous schedule is used by the EHCI for processing control and bulk transactions.

10.1.1 Periodic Schedule

The root of the periodic schedule is the ***PERIODICLISTBASE*** register and it contains the physical memory base address of the periodic frame list. The periodic frame list is an array of physical memory pointers that reference isochronous transfer descriptors and queue transfer descriptors.

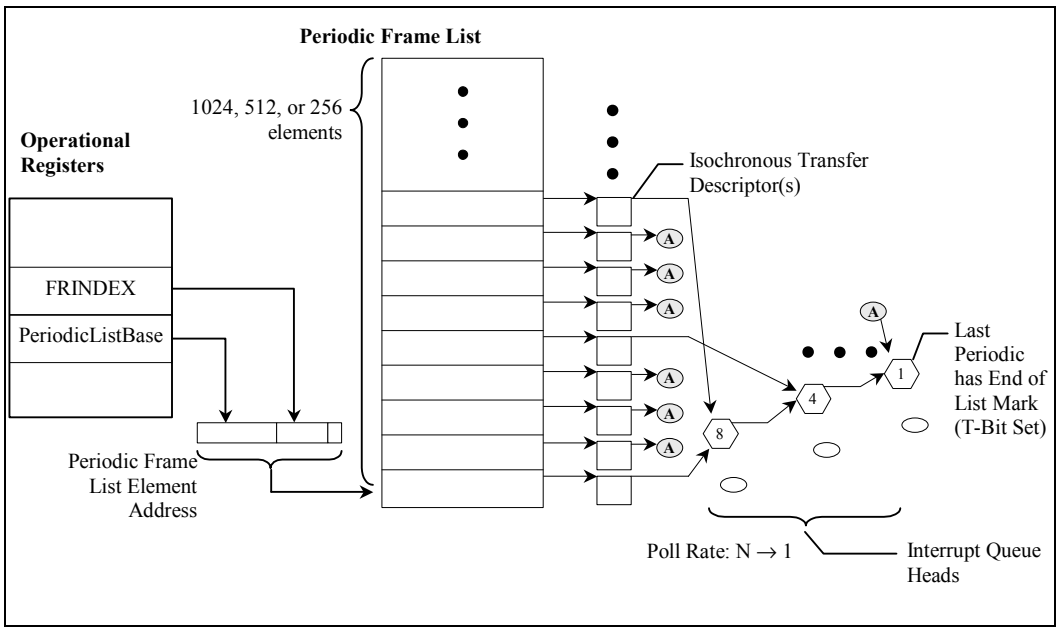
The physical memory pointers that occupy each element in the periodic schedule and have the following format:

Figure 7. Periodic Schedule Frame Entry



The host controller traverses the periodic schedule by constructing an array-offset reference using the ***PERIODICLISTBASE*** and the ***FRINDEX*** registers (refer to Figure 8. Periodic Schedule Organization). It fetches the element and begins traversing the graph of linked schedule data structures. Note that the controller will continue to traverse the linked list until is encounters a pointer with a T-bit set.

Figure 8. Periodic Schedule Organization



The '**T'** bit (refer to Figure 8) in the link pointer indicates to the controller whether or not the pointer is valid. If the periodic schedule is enabled and the controller encounters a link pointer whose 'T' bit is set, then the, the host controller will not process the current periodic schedule entry and will continue processing with the asynchronous schedule (if enabled).

If the periodic schedule is enabled then the host controller must attempt to execute from the periodic schedule before executing from the asynchronous schedule. It will only execute from the asynchronous schedule after it encounters the end of the periodic schedule and the asynchronous schedule is enabled. The end of the periodic schedule (for a micro-frame) is defined as when the Next Link Pointer (Isochronous Transfer Descriptor, Split Transaction Isochronous Transaction Descriptor) or the Queue Head Horizontal Link Pointer (Queue Head) have their 'T' bit set (bit 0 = 1).

Note: The EHCI will not process the periodic schedule if it has been disabled via the ***USB2CMD.PeriodicScheduleEnable*** bit. Because modifications to this bit are not immediate, software must poll (read) the ***USB2STS.PeriodicScheduleStatus*** bit until the requested transition has been made.

For more information regarding various transfer descriptor types that can be placed in the periodic schedule and for the exact operational behavior of the EHCI, refer to the *Enhanced Host Controller Interface Specification for Universal Serial Bus*.

10.1.2 Asynchronous Schedule

The root of the asynchronous schedule is the ***ASYNCLISTADDR*** register and it contains the physical memory base address of the Asynchronous Transfer List. The Asynchronous Transfer List is a circular linked (no T-bits sets) list of queue heads and queue transfer descriptors that provide the host controller with control and/or bulk work related items. The EHCI uses this list only under the following conditions:

- The asynchronous list is enabled

  AND

- The periodic list is enabled but empty

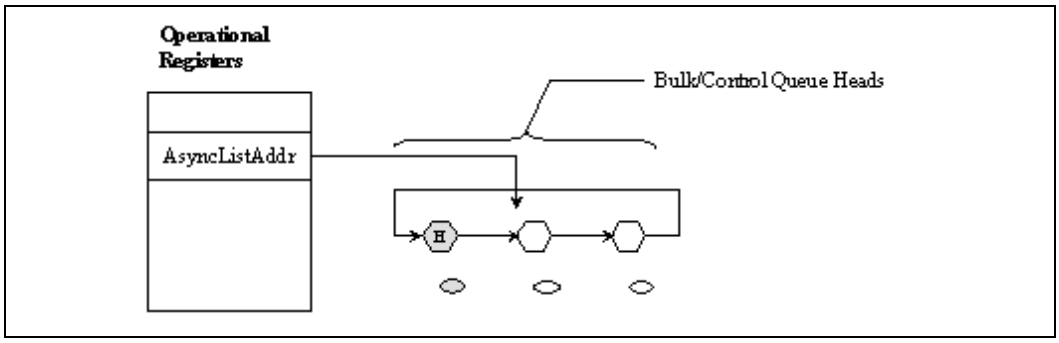
  OR

- The periodic list is disabled

  OR

- The periodic list is enabled but the end of the periodic list was reached

Figure 9. Asynchronous Schedule Organization



Because the Asynchronous transfer list is a circular linked list of queue heads, the ***ASYNCLISTADDR*** register is simply a pointer to the *next* queue head that the EHCI will process when it has completed processing the periodic list.

The ***USB2STS.AsynchronousScheduleStatus*** bit indicates status of the asynchronous schedule. System software enables (or disables) the asynchronous schedule by writing a 1 (or 0) to the ***USB2CMD.AsynchronousScheduleEnable*** bit. Software then can poll the ***USB2STS.AsynchronousScheduleStatus*** bit to determine when the asynchronous schedule has made the desired transition.

Note: Reminder – The EHCI will not process the asynchronous schedule if it has been disabled via the ***USB2STS.AsynchronousScheduleEnable*** bit. Because modifications to this bit are not immediate, software must poll (read) the ***USB2STS.AsynchronousScheduleStatus*** bit until the requested transition has been made.

For more detailed information regarding the operational behavior of the EHCI and the asynchronous schedule, refer to the *Enhanced Host Controller Interface Specification for Universal Serial Bus*.

This page is intentionally left blank.

11 Power Management

Unlike the UHCI (Universal Host Controller Interface), the ICH4/5 EHCI uses power management capabilities that are compliant with the *PCI Bus Power Management Interface Specification Revision 1.1*. Implementing PCI Power Management into the EHCI provides several important software related benefits:

1. Provides a non-operating system specific, industry standard mechanism for identifying devices with PM capabilities.
2. Provides a standard mechanism for controlling (i.e., changing power state) devices with PM capabilities
3. Eliminates the need to provide device specific PME bits in the chipset (i.e., Through PME# and the device's PM Control/Status Register, system software can scan the PCI bus to determine which device(s) generated a PM event.
4. PCI Power Management does not replace the power management capabilities available in an ACPI aware operating system (i.e., not mutually exclusive), but rather compliments it.

11.1 Non-PCI Power Managed Device – UHCI

The UHCI USB controllers integrated into the ICH4/5 are examples of devices whose power management aspects are handled by the ACPI operating system. The following procedure provides a general, high-level description of the interaction between an ACPI operating system and the UHCI hardware when a wake event occurs:

1. An event occurs on the USB (device connection, device disconnection, etc) for which the UHCI is programmed to wake the system. This example assumes that all appropriate enable registers are programmed and the system is in a non-S0 state (S1-S4).

Note: The following registers are defined in I/O space indicated by the ACPI Base Address Register specified in the ICH4/5 Device31:Function 0 Configuration Registers: ***GPE0_STS*** – General Purpose Event 0 Status, ***GPE0_EN*** – General Purpose Event 0 Enables, ***PM1_CNT*** – Power Management 1 Control.

2. The UHCI detects the wake event, and sets the appropriate ***GPE0_STS.USBx_STS*** bit.
3. If the appropriate ***GPE0_STS.USBx_EN*** bit is set, then the UHCI signals a wake event to the system, thus causing the system to transition to the S0 state.
4. Once the system is in the S0 state, an SCI will be generated if the ***PM1_CNT.SCI_EN*** bit is set (For ACPI operating system, ***PM1_CNT.SCI_EN*** should always be set).
5. The ACPI SCI handler will then process the SCI by examining the bits in the ***GPE0_STS*** register. Since the ICH5 contains 4 UHCI devices, 4 separate bits are implemented in the ***GPE0_STS*** register. (For ICH4, 3 separate bits are implemented in the ***GPE0_STS*** register since it contains 4 UHCI devices.)
6. The SCI handler will determine that one or more ***GPE0_STS.USBx_STS*** bits are set and will then execute an ACPI control method specific to the USB controller. This control method in turn will cause the UHCI system software to be made aware of the wake event.

It should be apparent that when using ACPI to power manage a device the following is required:

- Hardware must implement a separate bit (1 for each device) in the ACPI GPE status register
- Hardware must implement a corresponding enable bit (1 for each status) in the ACPI GPE enable register.
- BIOS (via ACPI) must implement separate control methods for each wake capable device defined in the ACPI namespace.

11.2 PCI Power Managed Device – EHCI

As stated previously, the EHCI integrated into the ICH4/5 incorporates power management capabilities that are compliant with the *PCI Bus Power Management Interface Specification Revision 1.1*. The interaction between the EHCI and an ACPI operating system is somewhat simplified and can be described in the following general, high level description of what occurs when the EHCI generates a PME# (PCI Power Management Event signal):

1. An event occurs on the USB (device connection, device disconnection, etc) for which the EHCI is programmed to wake the system. *This example assumes that all appropriate enable registers are programmed and the system is in a non-S0 state (S1-S4).*
2. The EHCI detects the wake event, and sets the ***PME_Status*** bit in the *Power Management Control/Status* (**PMSCR**) register (See Section 4.3.2- *PCI Configuration Registers*). The ***GPE0_STS.PME_B0_STS*** bit will get set only if the ***PMSCR.PME_En*** and the ***GPE0_EN.PME_B0_EN*** bits are also set (by system software).
3. If the ***PMSCR.PME_En*** bit is set, then the EHCI generates the PME# signal internally, which will also cause the ***GPE0_STS.PME_B0_STS*** bit to be set. If the system is in a sub-S0 state, then the assertion of PME# will cause the system to wake and transition to the S0 working state.

Note: The EHCI can assert PME# in any Sx state if the system software has set the ***PME_En*** bit and either a resume event or an event for which system software has enabled any of the ***Wake on xxx*** (See Section 11.2.2.1- D0->D3) bits in each port's ***PORTSC*** register occurs. During normal S0 operation, system software should clear the ***PMSCR.PME_En*** bit, since device connect, disconnect, resume and over-current states can be detected by the system software's normal interrupt service routine (ISR) via the USB2 Status register and associated ***PORTSC*** registers.

4. Once the system is in the S0 state an SCI will be generated if the ***PM1_CNT.SCI_EN*** bit is set (For ACPI operating system, SCI_EN should always be set).
5. The ACPI SCI handler will then process the SCI by examining the bits in the ***GPE0_STS*** register. Because EHCI is compliant with PCI Power Management specification, only 1 bit is required to represent the EHCI status. This bit is the ***GPE0_STS.PME_B0_STS*** bit and it's worth noting that this same bit is shared amongst all PCI Power Management compliant devices found on bus 0.
6. The SCI handler will determine that the ***GPE0_STS.PME_B0_STS*** bit is set and will then execute a control method specific to this status bit. This control method will eventually end up in the PCI bus driver. The PCI bus driver then has the responsibility of scanning PCI configuration space and looking for those devices that are both enabled and have their ***PMSCR.PME_Status*** bit set. In this example, the PCI driver would see that the EHCI has asserted PME# and would then make a request to the EHCI device driver to place itself in the D0 (fully on) state from which the EHCI device driver could service the wake event.

It should be apparent that when using ACPI in concert with a PCI Power Managed device the following is realized:

- Hardware need only implement a single shared bit in the ACPI GPE status register (**PME_B0_STS**); as stated previously, this bit is not device specific but rather is used for all PCI Power Management compliant devices on PCI bus 0.
- Hardware need only implement a single corresponding enable bit in the ACPI GPE enable register (**PME_B0_EN**). Like **PME_B0_STS**, this bit is shared amongst all PCI Power Managed devices on PCI bus 0.
- BIOS (via ACPI) need only implement a single control method for all PCI Power Managed devices on PCI bus 0 in order to handle wake events.

11.2.1 Supported Device Power States

In the ICH4/5, the EHCI supports 2 device power states: D0 (fully on) and D3 (full off). Device states D1 and D2 are optional device power states that **are not** supported by ICH4/5.

11.2.1.1 Device State D0

In this state no power conservation is in effect and the EHCI is fully functional. Support of this state is mandatory for PCI devices. The D0 state has 2 stages: un-initialized and active.

11.2.1.1.1 D0 Un-initialized

The EHCI enters this state from one of two ways:

- As a result of PCI RST# being asserted
- When transitioning from D3hot to the D0 state via system software.

In this state, the EHCI can only be a target of PCI configuration transactions. It is the responsibility of BIOS and/or system software to properly initialize the appropriate registers in the EHCI's PCI configuration space. Once this is completed, the EHCI will then be in the D0 active state.

11.2.1.1.2 D0 Active

Once BIOS and/or system software has configured and enabled the EHCI, it can then be considered in the D0 PM state. EHCI is now fully functional.

11.2.1.1.3 Device State D3

In this state power conservation is maximized and the EHCI has very limited functionality. Support of this state is mandatory for PCI devices. There are two D3 flavors: D3hot and D3cold. The D3hot state is achieved when power (Vcc) is applied and the device is placed into the D3 device state via program control (programming the PowerState field of the ***PMCSR*** register). The D3cold state is achieved when the device is placed into the D3hot state and then Vcc is removed. Any accesses to the EHCI memory mapped I/O range will master abort when in the D3 state.

11.2.2 Power State Transitioning

The following sections describe the process that EHCI software must follow when handling D0->D3 and D3->D0 power state transitions:

11.2.2.1 D0->D3

To transition from the D0 to D3 power state EHCI system software must:

1. Selectively suspend any ports enabled and owned by the EHCI. This is accomplished by setting the ***PORTSC.Suspend*** bit for each port owned by the EHCI.

Note: Important – Note that the port is not truly suspended until software reads back the ***PORTSC.Suspend*** bit as 1 as there may be a delay in suspending the port if there is a transaction in progress on the USB. System software must poll this bit until it reads back as 1 before continuing.

2. Set the ***USB2CMD.Run/Stop*** bit to 0 in order to stop execution of the transaction schedules (Periodic and Asynchronous).

Caution: Important – To insure that the EHCI has stopped executing the transaction schedule, system software must poll the ***USB2STS.HCHalted*** bit until it reads as 1. System software **must** insure that this bit reads 1 before continuing.

3. Because the following registers **will not** be preserved, the EHCI system software must either save them before placing the EHCI into D3 or software must have a mechanism for restoring them after transitioning back into the D0 state:
   - USB2 Command (**USB2CMD**)
   - USB2 Interrupt Enable (**USB2INTR)**
   - Control Data Structure Segment (**CTRLDSSEGMENT**)
   - Period Frame List Base Address (**PERIODICLISTBASE**)
   - Next Asynchronous List Address (**ASYNCLISTADDR**)
4. Because the following registers are maintained in the *suspend power well*, they do not need to be preserved or restored by the EHCI system software:
   - Host Controller Structural Parameters (**HCSPARAMS**)
   - Configure Flag Register (**CONFIGFLAG**)
   - Port 0 through port N Status and Control registers (**PORTSC**)
   - USB Legacy Support Extended Capability register (**USBLEGSUP**)
5. Because the following PCI configuration space registers **will not** be preserved, the EHCI system software prior to entering the D3 state must save them. Those PCI configuration registers not specified require no special attention by the EHCI system software because: 1) they reside in the suspend power well 2) they do not get reset during D3-D0 transition or 3) they are re-initialized by system software (e.g., BIOS) due to system wake (e.g., Vcc re-applied). The following table describes those PCI configuration space registers that will not be preserved:

Table 9. Non-Preserved PCI Configuration Space Registers

Offset	Register Name/Function
04-05h	Command Register
10-13h	Memory Base Address Register
3Ch	Interrupt Line

6. Determine which ports can be used for wake events. System software determines this by examining the Port Wake Capability (See Section 4.3.2 - PCI Configuration Registers) register. Based on the value in the Port Wake Capability register and other information about the device's capabilities and/or system preferences, the following bits in each of the ***PORTSC*** registers must be properly programmed before entering the D3 state:

Table 10. PORTSC Register Wake Enable Bits

Bit	Description
22	**Wake on Over-current Enable (WKOC_E) — RW.** Default = 0b. Writing this bit to a 1 enables the setting of the PME Status bit when the Over-current Active bit is set.
21	**Wake on Disconnect Enable (WKDSCNNT_E) — RW.** Default = 0b. Writing this bit to a 1 enables the setting of the PME Status when the Current Connect Status changes from connected to disconnected.
20	**Wake on Connect Enable (WKCNNT_E) — RW.** Default = 0b. Writing this bit to a 1 enables the setting of the PME Status bit when the Current Connect Status changes from disconnected to connected.

Note: Wake On Resume can be detected by system software (via PME#) if a device connected to a port is configured for remote wake.

Note: System software does not have to disable the EHCI interrupt (i.e., setting ***USB2INTR*** to 0) since the hardware will prevent its assertion while in a non-D0 power state

7. Finally, to place the EHCI in the D3 power state system software must write to the following *Power Management Control/Status* bits:

- ***PMSCR.PME_Status***. Writing a 1 to this bit insures that that PME# is in a de-asserted state.
- ***PMSCR.PME_En***. Writing 1 to this bit insures that the EHCI will assert PME# when the ***PMSCR.PME_Status*** is 1 (e.g., wake event occurred).
- ***PMSCR.PowerState***. Writing '11b' to these bits places the EHCI into the D3hot state.

11.2.2.2 D3->D0

To transition from the D3 to D0 power state EHCI system software must:

1. Place the EHCI in the D0 state by writing to the following *Power Management Control/Status* bits:
   - ***PMSCR.PME_Status***. Writing 1 to this bit insures that that PME# is in a de-asserted state.
   - ***PMSCR.PME_En***. Writing 0 to this bit insures that the EHCI will not assert PME# while in the D0 state.
   - ***PMSCR.PowerState***. Writing '00b' to these bits places the EHCI into the D0 working state.
2. The EHCI system software must restore those PCI configuration registers (*Command, Interrupt Line, Memory Base Address*) that were saved prior to transitioning to the D3 state. This step MUST BE done before the next step.
3. The EHCI system software must reinitialize/restore the following EHCI registers:
   - USB2 Command (**USB2CMD**)
   - USB2 Interrupt Enable (**USB2INTR)**
   - Control Data Structure Segment (**CTRLDSSEGMENT**)
   - Period Frame List Base Address (**PERIODICLISTBASE**)
   - Next Asynchronous List Address (**ASYNCLISTADDR**)
4. Set the ***USB2CMD.Run/Stop*** bit to 1 to restart the controller.
5. For each port owned by the EHCI, the EHCI system software should examine each port's PORTSC register to determine which one is host to the device that was responsible for waking the system. System software should un-suspend only those ports for which some type of status change occurred (i.e., connect, disconnect, over-current or resume). Those ports with no change in status should be left in the suspended state.

11.2.2.3 D3->D0

To transition from the D3 to D0 power state EHCI system software must:

1. Place the EHCI in the D0 state by writing to the following *Power Management Control/Status* bits:
   - ***PMSCR.PME_Status***. Writing 1 to this bit insures that that PME# is in a de-asserted state.
   - ***PMSCR.PME_En***. Writing 0 to this bit insures that the EHCI will not assert PME# while in the D0 state.
   - ***PMSCR.PowerState***. Writing '00b' to these bits places the EHCI into the D0 working state.

2. The EHCI system software must restore those PCI configuration registers (Command, Interrupt Line, Memory Base Address) that were saved prior to transitioning to the D3 state. This step MUST BE done before the next step.
3. The EHCI system software must reinitialize/restore the following EHCI registers:
   - USB2 Command (**USB2CMD**)
   - USB2 Interrupt Enable (**USB2INTR)**
   - Control Data Structure Segment (**CTRLDSSEGMENT**)
   - Period Frame List Base Address (**PERIODICLISTBASE**)
   - Next Asynchronous List Address (**ASYNCLISTADDR**)
4. Set the ***USB2CMD.Run/Stop*** bit to 1 to restart the controller.
5. For each port owned by the EHCI, the EHCI system software should examine each port's PORTSC register to determine which one is host to the device that was responsible for waking the system. System software should un-suspend only those ports for which some type of status change occurred (i.e., connect, disconnect, over-current or resume). Those ports with no change in status should be left in the suspended state.

This page is intentionally left blank.

12 Pausing the Transaction Schedules (Entering Sub-C0 States)

The EHCI system software can easily detect when its internal transaction schedules are under little to no load and as a result can disable them as part of its effort to assist a system in reaching the C3 power state. There are 4 levels of CPU power consumption and they are classified as C0, C1, C2, and C3. A CPU in the C0 power state consumes the most amount of power and a CPU is C3 state consumes the least amount of power.

When the CPU is in the C0 power state, instructions can be executed and no power or thermal savings are realized. However, as the CPU is placed in subsequent power states (i.e., C1, C2 or C3), better power and thermal savings are progressively achieved (for a full explanation of the CPU power states, reference the *Advanced Configuration and Power Management Interface Specification, v1.0b)*.

Because a CPU in the C3 state consumes the least amount of power, mobile computer systems prefer to be in this state the majority of time (if the system is idle) as it helps conserve available battery power. An important requirement for reaching and maintaining the C3 state is that the there can be no active bus mastering occurring in the system prior to entering C3 and during C3.

Since the EHCI incorporates two DMA engines, many bus master cycles can initiated by the EHCI during times of heavy use and even when the system is idle (i.e., no USB traffic). Obviously during periods of frequent USB use, it would not be desirable for operating system software to demote the CPU to a lower power state as this has performance consequences. However, during those periods of infrequent bus use, the operating system power management software (OSPM) should be allowed to place the CPU in a lower power state if it is consistent with the policy of the platform.

While the EHCI is not solely responsible for allowing the CPU to enter a low power state, it does play an integral part. For example, if the EHCI is allowed to continuously fetch inactive transfer descriptors from memory, the resultant bus master cycles will prevent the operating system from placing the CPU in its lowest obtainable power state. To assist the operating system in its power management policy, the EHCI provides system software with the ability to enable and disable processing of the periodic and asynchronous schedules; this is known as pausing the schedules. To pause the periodic schedule, system software should clear the *Periodic Schedule Enable* (**USB2CMD**) bit. To disable the asynchronous schedule, system software should clear the *Asynchronous Schedule Enable* (**USB2CMD**) bit.

Note: To insure that no bus master cycles are initiated by the EHCI, system software must disable both the periodic and the asynchronous schedules. System software should always check the Asynchronous Schedule Status and Periodic Schedule Status bits (USB2STS) after enabling/disable the schedules as these bits report the true status of their respective schedules. NOTE: After setting/clearing a schedule enable bit, system software MUST NEVER manipulate the associated schedule until the associated status bit matches the desired state (enable == status).

Manipulating a schedule before the desired setting has taken place can lead to unpredictable results.

Once the periodic and asynchronous schedules are disabled, they will remain so until system software re-enables them.

Note: System software should always use the pause feature of the EHCI instead of suspending the entire bus (i.e., don't selective suspend each port and clear the run/stop bit). Suspending the entire bus is not recommended if the entire system is not transitioning into a low power state since the latencies incurred when resuming the bus are not desirable.

12.1 Choosing a Model

The EHCI system software can choose to implement one or both of the following models to assist the OSPM software in reaching its power management goals:

1. Before pausing the schedules, EHCI system software waits for both the periodic and asynchronous schedules to become idle. Optionally, EHCI system software may individually pause a schedule as it becomes idle – this is an implementation preference as the net effect is the same.
2. Before pausing the schedules, EHCI system software waits until the asynchronous schedule is idle and only non-high bandwidth interrupt transactions are present in the periodic schedule.

12.1.1 Wait for Idle Periodic and Asynchronous Schedules Model

This is the simplest model for the EHCI system software to implement, as it requires no special timers or computations. To implement this model, EHCI system software need only detect when both schedules are idle and then clear the schedule enable bits. To re-enable the schedule(s), the EHCI system software must *set* the appropriate enable bit(s) in the controller when upper level software performs subsequent transaction requests. For example, if upper level software performs a bulk (In or OUT) request to a particular endpoint, then the EHCI system software must re-enable the *asynchronous schedule* before the transaction can occur. Like wise, if an interrupt request is performed, then the EHCI system software must enable the *periodic schedule* before the transaction can occur. Once the requested transaction(s) have completed and no pending transactions remain in the schedule(s), the EHCI system software can once again pause the schedule(s).

12.1.1.1 Interrupt Endpoints

For interrupt-type endpoints, this model requires that upper level software play more of a role in determining when transactions to the interrupt endpoint should be temporarily held off. For example, when an interrupt transaction is placed on the periodic schedule, it will be processed continuously until one of the following occurs:

1. The transaction is retired due to the associated USB device returning useful data (i.e., does not NAK), or an error occurs.
2. The transaction is retired due to cancellation of the IRP by upper level software.

Because the model described in this section requires that both the periodic and asynchronous schedules are idle before a pause, upper level software must play a larger role in the scheduling of endpoint transactions. For example, upper level software could cancel an interrupt type transaction

on an endpoint after it detects no activity after a certain amount of time. To avoid missing future device interrupts, upper level software must periodically poll its USB device by rescheduling interrupt transaction at some best-computed frequency. Upper level software could keep repeating this process until an interrupt transaction is successfully processed (i.e., device returns some data). Upper level software should implement some type of heuristic algorithm that balances both the power requirements of the system and the expectations of the user.

12.1.2 No Idle Schedules Model

This model is very similar to the model previously described with some exceptions:

- The periodic and asynchronous schedules do not necessarily have to be idle before the EHCI system software pauses them.
- The EHCI system software must periodically re-enable the schedule(s) to insure that scheduled interrupt type transactions are sent to their respective USB devices.

12.1.2.1 Waiting for an Idle Asynchronous Schedule

While it is true that the asynchronous schedule need not be idle for the EHCI system software to invoke its ‘pausing policy’ it is however recommended. This recommendation is based on certain scenarios that could cause the loss of data in the event of a system failure or some other catastrophic event. For example, a user application (Word Processing, database etc.) may cause the asynchronous schedule to contain a number of Bulk out transactions to a mass storage device. To minimize the possibility of data loss due to system failure, loss of power, etc, the EHCI system software should insure that all data is physically transferred to the USB device before the schedule is paused.

12.1.2.1.1 Interrupt Endpoints

As previously described, this model does require that the periodic schedule be *completely* idle; only non-high bandwidth interrupt transactions may be present in the schedule at the time it is paused. High bandwidth interrupts and isochronous transactions **may not** be present in the periodic schedule since their streaming nature would not provide much (if any) opportunity for the system to maintain a state of low power consumption. Note that this model works equally well when he periodic schedule is 100% idle.

Once EHCI system software has determined that only interrupt transactions are present in the periodic schedule (or it is idle), it can begin the process of placing the periodic schedule in a paused state. The process for pausing the periodic schedule is described as follows:

1. Software clears the Periodic Schedule Enable (USB2CMD) bit.
2. If the periodic schedule is idle, then software is done.

Note: If the periodic schedule is idle, system software must provide a mechanism that will allow the schedule to be re-enabled when new items are added to the schedule.

3. System software must compute at which time in the future it will re-enable the periodic schedule. EHCI system software can easily accomplish this by examining the polling interval associated with each interrupt endpoint and use this information to determine a ‘best fit’ value. Specific algorithms for computing this value are beyond the scope of this specification. Once this time is computed, EHCI system software must use whatever operating system timer mechanisms are available to ‘wake’ itself at the desired time.

Note: Depending on the operating system, the resolution of the system timer services can vary somewhat. For example, the minimum resolution of the timer in Windows NT/2000 is ~10 ms. When computing wake times, system software should factor this into the time computations. Because interrupt transactions remain in the periodic schedule until they completed (successfully or in error), the EHCI system software has some flexibility when accounting for the lack of system timer granularity; as long as the periodic schedule gets enabled at some time in the future, the interrupt transaction is guaranteed to be processed regardless as to whether or not the timer under or over sleeps.

4. Upon waking from the system timer, the EHCI system software must enable the periodic schedule. To enable the schedule, the EHCI system software must set the *Periodic Schedule Enable* (**USB2CMD**) bit.
5. When all of the scheduled interrupt transactions have been executed, the EHCI system software should repeat this process.

13 Debug Port

The Debug port allows debugging of software by system software (OS and BIOS). It allows the software to communicate with an external console using a USB2 connection. Because the interface to this link does not go through the normal USB2 system software stack, it allows communication with the external console during instances when the OS is not loaded, the USB2 system software is broken, or when the USB2 system software is being debugged.

- Specific features of this implementation of a debug port are:
- Only works with an external USB2 debug device (console)
- Implemented for a specific port on the host controller
- Operational anytime the port is not suspended AND the host controller is in D0 power state.
- Capability is interrupted when port is driving USB RESET

13.1 Debug Software

The following sections specify how debug port aware system software should discover, enable and interact with the debug port hardware implemented in the EHCI.

- For more details regarding the debug port, refer to Appendix C of the Enhanced Host Controller Interface for Universal Serial Bus Specification.
- For more details regarding the USB Debug device, refer to the USB Debug Port Peripheral Specification.

13.2 Debug Port Control Register

The Debug Port Control register (DebugPortBar + offset 0h) allows software to interact with the USB2 Debug Port. The hardware associated with this register provides no checks to ensure that software programs the interface correctly. How the hardware behaves when programmed illegally is undefined. The definition of each bit or field in the control register is described in the table below.

Table 11. Debug Port Control Register

Offset	80h
Attribute	Read-Write
Size	32-bits
General Behavioral Rules	Debug port system must perform Read-Modify-Write operations to this register to preserve the contents of bits not being modified. This includes Reserved bits. In order to preserve the usage of RESERVED bits in the future, software should always write the same value read from the bit until it is defined. Reserved bits will always return 0 when read.

Bit	Description
31	**Reserved**
30	**OWNER_CNT:** R/W. When software writes a 1 to this bit, the ownership of the debug port is forced to the EHCI controller (i.e., immediately taken away from the companion Classic USB Host Controller). If the port is already owned by the EHCI controller, then setting this bit has no effect. This bit overrides all of the ownership-related bits in the standard EHCI registers. Reset default = 0. Note that the value in this bit may not affect the value reported in the *Port Owner* bit in the associated PORTSC register.
29	**RESERVED**
28	**ENABLED_CNT:** This bit = 1 if the debug port is enabled for operation. Software can clear this by writing a 0 to it. The hardware clears the bit for the same conditions where hardware clears the Port Enable/Disable Change bit (in the PORTSC register). (Note: this bit is not cleared when System Software clears the *Port Enabled/Disabled* bit (in the PORTSC register) Software can directly set this bit if the port is already enabled in the associated Port Status and Control register (this is enforced by the hardware). Reset default = 0.
27:17	**RESERVED**
16	**DONE_STS:** Read/Write-Clear. This bit is set by hardware to indicate that the request is complete. Writing a 1 to this bit will clear it if it is set. Writing a 0 to this bit has no effect. Reset default = 0.
15:11	**RESERVED**
10	**IN_USE_CNT**: Set by software to indicate that the port is in use. Cleared by software to indicate that the port is free and may be used by other software. This bit is cleared after reset. (This bit has no affect on hardware.)
9:7	**EXCEPTION_STS:** Read-Only. This field indicates the exception when the ERROR_GOOD#_STS bit is set. This field should be ignored if the ERROR_GOOD#_STS bit is 0. 000 No Error. Note: this should not be seen, since this field should only be checked if there is an error. 001 Transaction error: indicates the USB2 transaction had an error (CRC, bad PID, timeout, etc.) 010 HW error. Request was attempted (or in progress) when port was suspended or reset. All Others are reserved
6	**ERROR_GOOD#_STS:** Read-Only. The hardware clears this bit to 0 upon the proper completion of a read or write. The hardware sets this bit to indicate that an error has occurred. Details on the nature of the error are provided in the Exception field. Reset default = 0.
5	**GO_CNT:** Write-Only. Software sets this bit to cause the hardware to perform a read or write request. Writing a 0 to this bit has no effect. Reads to this bit will always return 0.
4	**WRITE_READ#_CNT:** R/W. Software sets this bit to indicate that the current request is a write. Software clears this bit to indicate that the current request is a read. Reset default = 0.

Bit	Description
3:0	**DATA_LEN_CNT:** R/W. This field is used to indicate the size of the data to be transferred. Reset default = 0h. For write operations, this field is set by software to indicate to the hardware how many bytes of data in Data Buffer are to be transferred to the console. A value of 0h indicates that a zero-length packet should be sent. A value of 1-8 indicates 1-8 bytes are to be transferred. Values 9-Fh are illegal and how hardware behaves if used is undefined. For read operations, this field is set by hardware to indicate to software how many bytes in Data Buffer are valid in response to a read operation. A value of 0h indicates that a zero-length packet was returned and the state of Data Buffer is not defined. A value of 1-8 indicates 1-8 bytes were received. Hardware is not allowed to return values 9-Fh. The transferring of data always starts with byte 0 in the data area and moves toward byte 7 until the transfer size is reached.

13.2.1 Enabling the Debug Port

There are two conditions that debug software must address as part of its startup processing:

1. The EHCI has been initialized by system software
2. The EHCI has not been initialized by system software

Debug software can determine the current 'initialized' state of the EHCI by examining the ***Configure Flag*** register. If this flag is set to 1, then system software has initialized the EHCI. Otherwise the EHCI should not be considered initialized. Debug software will initialize the debug port registers depending on the state the EHCI. However, before this can be accomplished, debug software must determine which root USB port is designated as the debug port.

13.2.2 Determining the Debug Port

Debug software can easily determine which USB root port has been designated as the debug port by examining the ***HCSPARAMS.DebugPortNumber*** bits (**DP_N**). This 4-bit field represents the numeric value assigned to the debug port (i.e., 0001 == port 1, 0010 == port 2 …… 1111 == port 15). For ICH5 and ICH4, this value is 0001 (port 1).

13.2.3 Debug Software Startup with Non-Initialized EHCI

To enable a non-initialized (***Configure Flag*** = 0) EHCI for debugging via the debug port, debug port system software must perform the following:

1. Make EHCI the owner of the port. Debug port system software must program the ***DebugPort.Control.Owner_Cnt*** bit with a 1.
2. Determine the port connection status. If ***PORTSC.CurrentConnectStatus*** is 1, then a device is present on the port and debug port system software must continue. If a device is not present on the port then debug port system software must terminate or wait until a device is connected to the debug port.
3. Check the Line Status registers. If ***PORTSC.LineStatus*** is not a value of '01b' (not low-speed device) then debug port system software must continue. If the connected device is a low speed device, then debug port system software must terminate or wait until a high-speed device is connected to the debug port.
4. Start the EHCI. Debug port system software must program the ***PORTSC.Run/Stop*** with a 1.

5. Reset the port. To guarantee a successful debug port reset, debug system software must:

   a. Program ***PORTSC.PortReset*** with a 1 to initiate the port reset sequence.

   b. Begin timing the reset sequence after successfully reading a value of 1 from the ***PORTSC.PortReset*** bit. Reset signaling on the bus does not occur until the ***PORTSC.PortReset*** bit is set to 1 by host controller.

Terminate the reset sequence after the spec-ed amount of time (nominally this is 50 ms. See Section *7.1.7.5* of the *Universal Serial Bus Specification Revision 2.0*). System software terminates the port-reset sequence by programming the ***PORTSC.PortReset*** bit with a 0. The reset process is actually complete when the host controller sets the ***PORTSC.PortReset*** bit to 0. Debug port system software must not continue until the host controller sets this bit to 0.

6. Check if the port is enabled. If following the reset a device is connected (***PORTSC.CurrentConnectStatus*** = 1) to the port and the port is enabled (***PORTSC.PortEnabled/Disabled*** = 1), then the attached device is a USB2 high-speed device. If neither of these conditions is true, then debug port system software must terminate or wait until a high-speed device is connected to the debug port. Otherwise, debug port system software must continue.
7. Enable the port for debug traffic. Debug port system software must program the ***DebugPort.Control.Enabled_Cnt*** bit with a 1.
8. Disable the port at the ***PORTSC*** level. Debug port system software must program the ***PORTSC.PortEnabled/Disabled*** bit with a 0.
9. Stop the host controller. Debug port system software must program the ***PORTSC.Run/Stop*** bit with a 0.

Completion of steps 7 and 8 guarantees that when EHCI system software is loaded, it will find the EHCI registers in the proper default idle condition.

13.2.4 Debug Software Startup with Initialized EHCI

To enable an *initialized* (***Configure Flag*** = 1) EHCI for debugging via the debug port, debug port system software must perform the following:

1. Determine the port connection status. If ***PORTSC.CurrentConnectStatus*** is 1, then a device is present on the port and debug port system software must continue. If a device is not present on the port then debug port system software must terminate or wait until a device is connected to the debug port.
2. Check the Line Status registers. If ***PORTSC.LineStatus*** is not a value of '01b' (not low-speed device) then debug port system software must continue. If the connected device is a low speed device, then debug port system software must terminate or wait until a high-speed device is connected to the debug port.
3. Set ownership of the port. To enable debug traffic on the port, debug port system software must program the ***DebugPort.Control.Owner_Cnt*** bit with a 1.
4. Enable the debug port. To enable debug traffic on the port, debug port system software must program the ***DebugPort.Control.Enable_Cnt*** bit with a 1.

Note: Debug port system software may combine steps 3 and 4 with a single write to the ***DebugPort.Control*** register. If debug port system software chooses not to combine these steps, it must insure that the ***Owner_Cnt*** bit is set before the ***Enable*** bit.

Once the above steps have been completed, the debug port will be enabled for debugging.

13.2.4.1 Initializing the Debug Port Registers – Coding Example

The following ‘C’ code illustrates the process that debug port system software should use when enabling a port for debugging. This ‘C’ function covers the topics discussed in sections *13.2.3* and *13.2.4*.

```
/*
IMPORTANT - READ BEFORE COPYING, INSTALLING OR USING.

Do not use or load this source code ("Source Code"), any executable
object-code software derived from Source Code Listed below ("Software")
and any associated materials until you have carefully read the following
terms and conditions. By loading or using the Source Code or Software,
you agree to the terms of the Intel Source License Agreement
(OEM / IHV / ISV Distribution & Single End-User) included
If you do not wish to agree, do not install or use the Source Code or Software.

Copyright (c) 2003, Intel Corporation.

This file, software, or program ("Code") is licensed under the terms of a
license agreement and/or non-disclosure agreement with Intel Corporation
("Intel"), and shall not be used, copied, or disclosed except in accordance
with that agreement. This code is copyrighted or contains trade secret material
of Intel, and must be treated as such. Except as expressly stated in the
agreement You have with Intel, no license or right to the Code is granted
to You directly or by implication, inducement, estoppel or otherwise.

THE CODE IS PROVIDED "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF
ANY KIND INCLUDING WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT,
OR FITNESS FOR A PARTICULAR PURPOSE. INTEL WILL NOT PROVIDE ANY SUPPORT,
ASSISTANCE, INSTALLATION, TRAINING OR OTHER SERVICES.INTEL WILL NOT
PROVIDE ANY UPDATES, ENHANCEMENTS OR EXTENSIONS.

Intel does not warrant or assume responsibility for the accuracy or
completeness of any information, text, graphics, links or other items
contained within the code.

LIMITATION OF LIABILITY.
IN NO EVENT SHALL INTEL OR ITS SUPPLIERS BE
LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION,
LOST PROFITS, BUSINESS INTERRUPTION, OR LOST INFORMATION) ARISING OUT
OF THE USE OF OR INABILITY TO USE THE CODE, EVEN IF INTEL HAS BEEN
ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Other names and brands may be claimed as the property of others.
*/

//
// various constants
//
const   DWORD PORT_RESET_INTERVAL = 50;          // spec-ed as 50ms
//
// Configure Flag register bit sets
//
const   DWORD CONFIGURE_FLAG = 0x00000001;       // HC Configured bit
//
// USB2CMD - Command register, bit sets
//
const   DWORD RUN_STOP = 0x00000001;             // HC running.
//
// PORTSC - Port Status and Control register bit sets
//
const   DWORD PORT_RESET = 0x00000100;           // Port reset
const   DWORD PORT_ENABLED = 0x00000004;         // Port enabled
const   DWORD CONNECTED    = 0x00000001;         // Port connected
```

```
const   DWORD LINE_STATUS_LS = 0x00000400;       // Line Status, LS device
const   DWORD LINE_STATUS_MASK = 0x00000c00;     // Line Status mask
//
// Debug Port Control Register bit sets
//
const   DWORD OWNER = 0x40000000;                // Owner
const   DWORD DEBUG_PORT_ENABLED = 0x10000000;   // Enabled

//
// InitDebugPort - This function initializes the debug port for USB debugging
//
// Entry:
//  dwConfigFlag   ==> Contents of the EHCI Configure Flag register
//  pdwPortsc      ==> pointer to the debug port PORTSC register
//  pdwUsb2Cmd     ==> pointer to the USB2 Command register
//  pdwDbgPort     ==> pointer to the debug port control register
// Exit:
//  Returns 0 if a high-speed device attached to the debug port.
//  Therefore, the following debug port control register bits
//  will be set:
//  - DebugPort.Control
//  - DebugPort.Enabled
//
//  Returns non-0 if no high-speed device attached. The following
//  debug port registers will be cleared:
//  - DebugPort.Control
//  - DebugPort.Enabled
//

int
InitDebugPort( IN dword dwConfigFlag,
            IN dword *pdwPortsc,
            IN dword *pdwUsb2Cmd,
            IN dword *pdwDbgPort)
{
    int     iStatus = ~0;  // assume no HS device attached
    //
    // Check if EHCI controller has been already been
    // initialized by BIOS or the EHCI driver
    //
    if( !(dwConfigFlag & CONFIGURE_FLAG)) {
            //
            // make the EHCI the owner of the debug port
            //
            *pdwDbgPort |= OWNER;
            //
            // check if the port has a connected device
            //(PORTSC.CurrentConnectStatus)
            // If a device is connected then we need to reset the port
            // before we can determine if the connected device
            // is actually a high speed device. However, we don't
            // do a reset if the LineStatus (PORTSC.LineStatus) bits
            // indicate a low-speed device
            //
            if( (*pdwPortsc & CONNECTED)
            && (*pdwPortsc & LINE_STATUS_MASK) != LINE_STATUS_LS) {
                    //
                    // device attached, and not low-speed.
                    // make sure the HC is running (USB2CMD.Run/Stop)
                    //
                    if( !(*pdwUsb2Cmd & RUN_STOP))
                             *pdwUsb2Cmd |= RUN_STOP;
                             //
                             // HC is running. Start the port reset sequence
                             // (PORTSC.PortReset)
                             //
                             *pdwPortsc|= PORT_RESET;
                             //
                             // wait until done.
                             //
                             Sleep( PORT_RESET_INTERVAL); // OS specific, reader exercise
                             //
```

```
                        // terminate reset sequence
                        //
                        *pdwPortsc ^= PORT_RESET;
                        //
                        // not truly done until we read back 0
                        //
                        while( *pdwPortsc & PORT_RESET);
                        //
                        // now check if we have a potential debug device attached
                        // (PORTSC.PortEnabledDisabled)
                        //
                        if( *pdwPortsc & PORT_ENABLED_DISABLED) {
                                //
                                // we have a device connected, let's enable the port
                                        at
                                // the debug port level (CONTROL.Enabled)
                                //
                                *pdwDbgPort |= DEBUG_PORT_ENABLED;
                                //
                                // now we need to reset (clear) the enabled
                                // bit at the HC level
                                //
                                *pdwPortsc ^= PORT_ENABLED_DISABLED;
                                //
                                // and finally, we turn off (stop) the HC
                                //
                                *pdwUsb2Cmd ^= RUN_STOP;
                                //
                                // important that we wait for the Halt bit to be set
                                        by HW
                                //
                                while(
                                iStatus = 0;  // indicate success
                        }
                }
        }
      }
      else {
              //
              // HC is already configured. Check if a device is connected
              //
              if( *pdwPortsc & CONNECTED)
              && ( *dwPortsc & LINE_STATUS_MASK) != LINE_STATUS_LS) {
                      //
                      // must set the Owner bit and Enabled bit
                      //
                      *pdwDbgPort |= (OWNER | DEBUG_PORT_ENABLED);
                      iStatus = 0;  // indicate success
              }
       }
       //
       // if there was an error, make sure the debug port Owner bit and
       // Enabled bit are not set before leaving
       //
       if( iStatus)
               *pdwDbgPort &= ~(OWNER |DEBUG_PORT_ENABLED);
       return( iStatus);
}
```

13.2.5 Determining Debug Peripheral Presence

After enabling the debug port functionality, debug software can determine if a debug peripheral is attached by attempting to send data to the debug peripheral. If all attempts result in an error (i.e., ***DebugPort.Control.Exception*** indicates a Transaction Error), then the attached device is not a debug peripheral. If the debug port peripheral is not present, then debug software may choose to terminate or it may choose to wait until a debug peripheral is connected.