EHCI:

Enhanced Host
Controller Interface
For USB 2.0

A Survey Of Major Features

MINDSHARE, INC.

JAY TRODDEN

SEPTEMBER 10, 2001

Enhanced Host Controller Interface

This Document

This document is a supplement to MindShare's ***Universal Serial Bus System Architecture (2nd Edition)*** book, and surveys the role the Enhanced Host Controller Interface (EHCI) plays in software and hardware management of USB 2.0 operations. EHCI both extends the functionality of, and maintains compatibility with earlier USB 1.x Open Host Controller Interface (OHCI) and Universal Host Controller Interface (UHCI) implementations. Just as the UHCI and OHCI did for USB 1.x, EHCI provides one level of the interface between client software executing on a CPU, and the device it wishes to control on a USB port.

The EHCI Specification was developed jointly by a number of the key USB hardware and software developers, including Intel, Compaq, Microsoft, NEC, and Lucent. The Specification was written with a number of goals, including:

- Full support for USB 2.0 High Speed device protocol (including new features such as *Split Transactions)*.
- A simple method of maintaining backward compatibility for earlier Low Speed and Full Speed USB 1.x devices using **companion** UHC or OHC controllers.
- Improvement over earlier methods in accessing main memory by the USB host controller while processing transactions.
- EHCI implementation of PCI power management features, compliant with the PCI Bus Power Management Interface Specification, revision 1.1.
- Optional 64 bit memory addressing of USB structures.

The sections that follow describe major areas covered in the EHCI 0.95 Specification, and are arranged as follows:

1. **EHCI overview.** A big-picture look at how a USB host controller fits into a system, and how the EHCI layer divides USB management into two parts: set-up of host controller hardware registers, and software control over transaction scheduling and generation of transfer descriptors.
2. **EHCI controller PCI and Memory Mapped I/O Registers**. A description of internal PCI configuration space and MMIO capability and operational registers defined for EHCI controllers.
3. **EHCI register operational summary**. Initialization and use of EHCI PCI and MMIO registers.
4. **EHCI memory data structures**. Use of the four memory structures set up by USB system software and used by the host controller to manage USB transactions:

References:

Enhanced Host Controller Interface Specification for USB, Revision 0.95

MindShare's *Universal Serial Bus System Architecture*, Second Edition

*Universal Serial Bus Specification*, Revision 2.0

I. EHCI Overview

Where A USB Controller Resides

Figure 1 below illustrates two common places a USB host controller might be found in a system: as a dedicated IC on a peripheral bus such as PCI, or integrated into a chipset component on PCI or on one of the new component connections (Intel's hub link, etc.). USB 2.0 includes high-speed transactions (up to 60 Mbtyes/s), which makes the placement of the controller more critical than it is for USB 1.x (about 12 MBytes/s). High speed component connections (Intel's Hub, AMD's HyperTransport, ServerWorks Inter Module Bus, etc.) and faster add-in buses such as PCI-X are expected to alleviate bandwidth problems of USB 2.0 controllers trying to reach memory.

*Figure 1: Dedicated and Integrated USB Host Controller Examples*
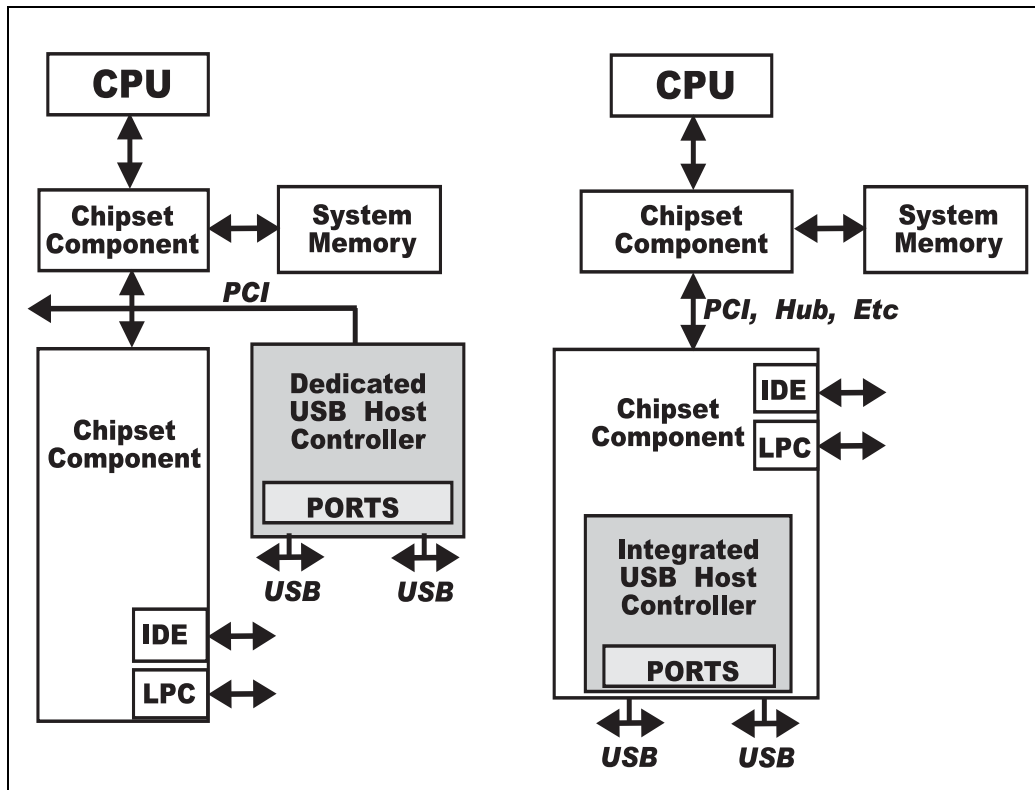
USB 1.x Layered Hardware And Software

In USB, a multi-layered hardware and software scheme is used. Layering helps elimi-nate the need for low-level system "knowledge" on the part of higher-level operating system and client software; this abstraction isolates as many implementation-specific system details as possible--allowing compatible software operation on all sorts of plat-forms. The hardware and software layering typical in USB 1.x systems is illustrated in Figure 2; the UHCI/OHCI interface is highlighted.

Figure 2: UHCI/OHCI Interface Layer In A USB 1.x Host System
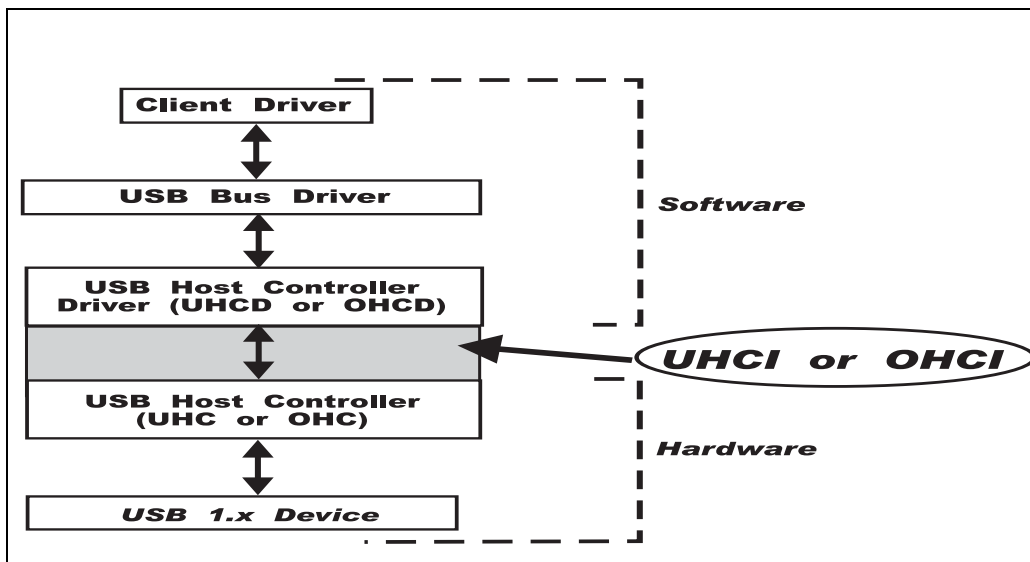
Refer to MindShare's *Universal Serial Bus Architecture* book for a thorough description of the relationships between the various layers of USB hardware/software. A brief summary of the major blocks shown in Figure 2 includes:

- **Client Driver.** This software is used to manage a particular USB device
- **USB Bus Driver (USBD).** This software layer isolates the details of the particular host controller driver (UHCI, OHCI) in use, from the operating system.
- **Host Controller Driver (UHCD, OHCD).** Driver software specific to the host controller hardware registers in use.
- **USB Host Controller.** The specific system hardware managing the USB ports. The host controller complies with OHCI, UHCI specifications.
- **USB 1.x Device.** The hardware component being managed by the client software.

USB 2.0 Layered Hardware And Software

In a full-featured USB 2.0 host controller, the hardware/software layering is extended to include the high-speed enhanced controller and it's EHCI interface. Note that the USB 1.x UHCI or OHCI controller functionality has not migrated to the EHCI controller; Instead, EHCI handles High Speed (HS) USB transactions while the *companion controller(s)* take care of Full-Speed (FS) and Low Speed (LS) devices which may be attached to the USB 2.0 controller's ports. This "companion controller" scheme reduces the complexity of EHCI and allows integrating existing USB 1.x logic and software into newer controllers transparently.

A number of things are done to remove uncertainty about which controller is to be used in the management of a particular port, including:

- At reset, all ports default to being managed be the USB 1.x companion controllers
- When (and if) system software programs the EHCI interface, it may be set up as the default controller, and will handle transactions for all ports unless a low-speed or full-speed device is detected on a port; in that case, port control will be routed to a companion controller. If the LS or FS device is later disconnected, control reverts to EHCI until the next device is attached and it's speed is determined.

Figure 3 illustrates USB 2.0 layered hardware and software. EHCI is highlighted.

Figure 3: EHCI Interface Layer In A USB 2.0 Host System
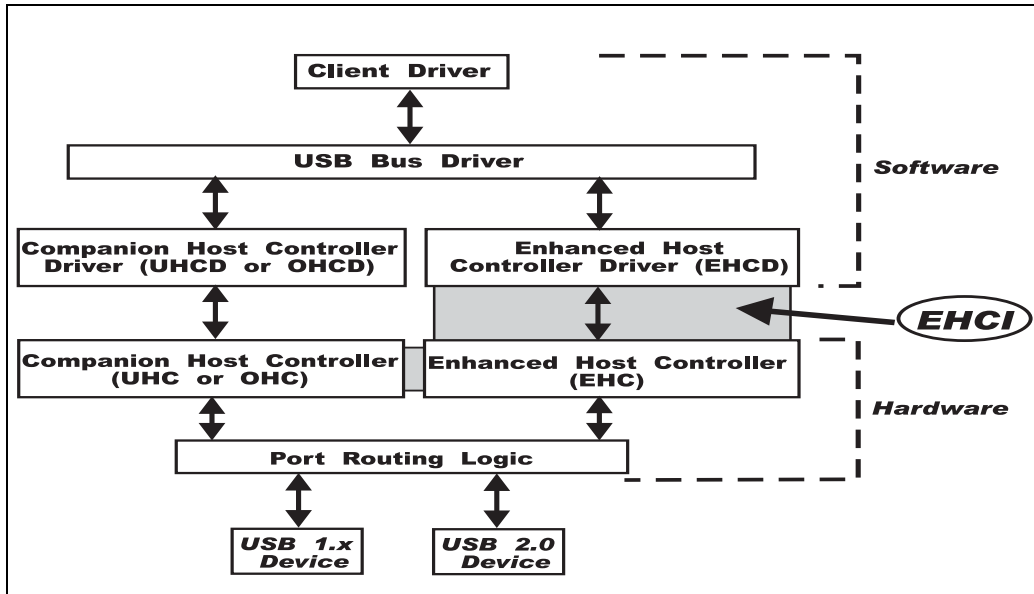
Managing A USB EHCI Controller

In implementation, the EHCI interface layer requires two important low-level components of USB controller management. These are depicted in Figure 4.

The first part of EHCI host controller management consists of hardware-level programming of the host controller itself, some of which is accomplished via PCI configuration and some through reading and writing memory mapped I/O (MMIO) USB capability and operational registers. These registers are used to program controller options as well as retrieve controller capabilities and operation status. See **(1)** in Figure 4.

The other part of USB host controller management is the dynamic programming of transfer descriptors in system memory by USB driver software in response to client software USB requests. See **(2)** in Figure 4. The controller fetches the descriptors, converts them to USB transactions, and returns status to memory. Much of the complexity that would otherwise fall on the USB host controller is avoided by having USB system software manage transfer descriptors in asynchronous and periodic schedules already optimized for the host controller.

Figure 4: EHCI Host Controller Registers And Memory Data Structures
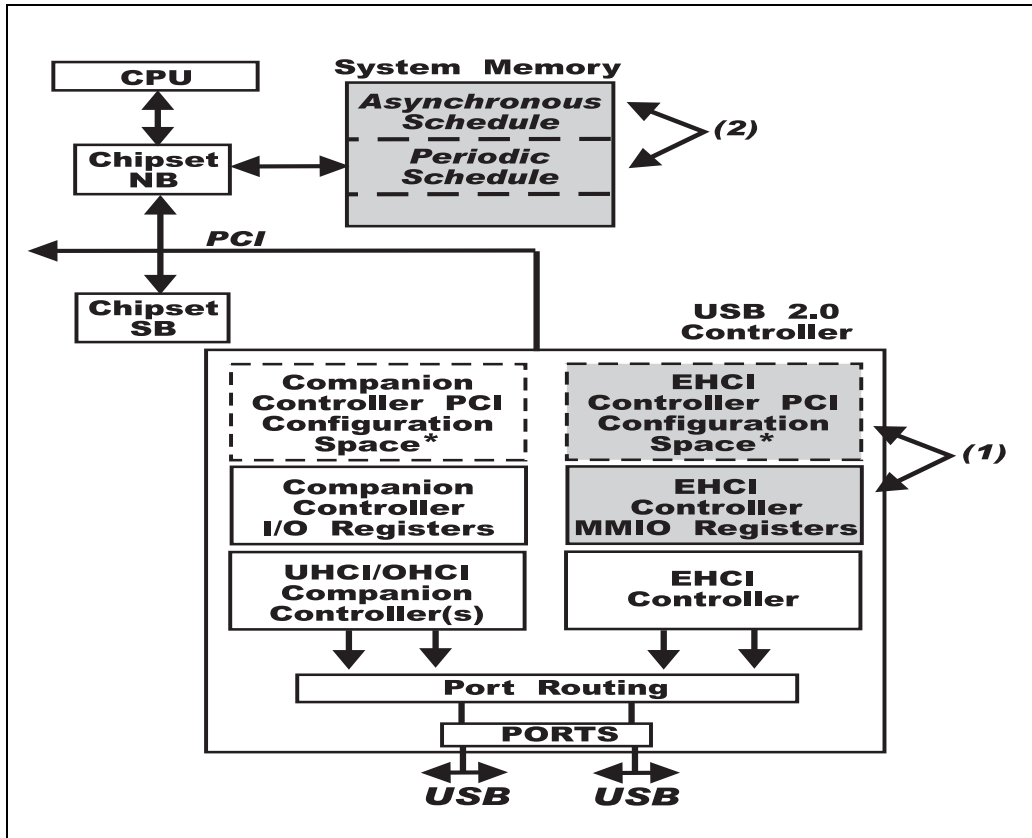
2. EHCI Controller PCI And Memory Mapped I/O Registers

EHCI Host Controller PCI Configuration Registers
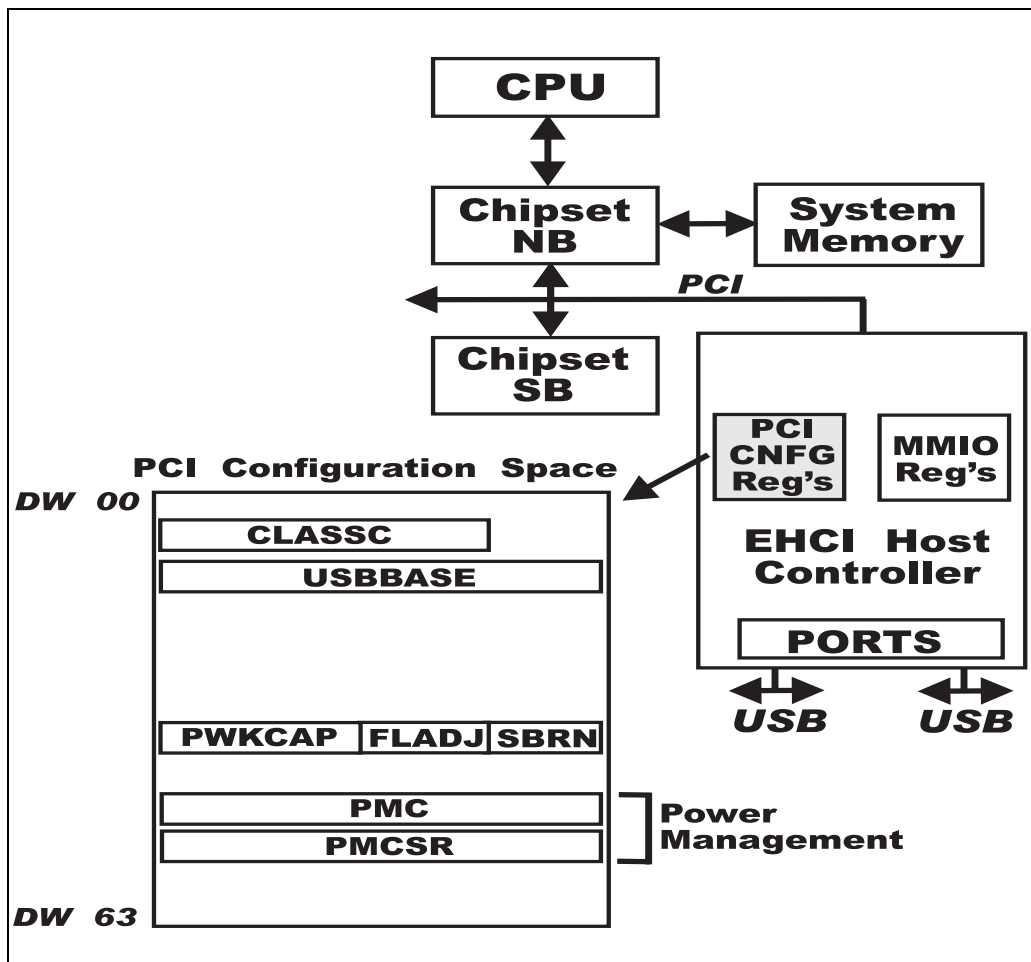
If the EHCI host controller resides on a PCI bus (optional), then it must implement the 256 byte (64 dword) PCI configuration space. The contents of the EHCI PCI configuration space consist mainly of information required for PCI enumeration and bus management--the bulk of USB transaction management is handled through the memory-mapped I/O (MMIO) registers described in the next

section. A PCI-based EHCI controller must also implement the PCI Power Management advanced capability register set.

If companion controllers are also supported (see Figure 5), they will have their own PCI space in a multi-function PCI device. Refer to MindShare's *PCI System Architecture* for a thorough discussion of PCI configuration space. For reference, the important fields in the EHCI PCI configuration space of a USB 2.0 controller are highlighted below.

Figure 5: EHCI Controller PCI Configuration Space

Key EHCI PCI Configuration Register Summary

CLASSC (24 bits) - *PCI Class Code Register.*
PCI Configuration Space Address: Byte Offset 09-0Bh
Default Register Value: implementation dependent
Field: Read-Only (RO)

The designer of the PCI-based EHCI controller hard-codes the PCI class code in this register. An EHCI USB 2.0 controller would return the following 3 bytes of information when this register is read: 0C0320h

- Class = 0Ch (Serial Bus Controller)
- Sub-Class = 03h (USB serial bus host controller)
- Programmers Interface = 20h (Conforms to the USB 2.0 Specification)

USBBASE (32 bits) - *USB Base Address Register* (BAR).
PCI Configuration Space Address: Byte Offset 10h-13h
Default Register Value: implementation dependent
Mixed Fields: Read-(RO, RW)

The first PCI base address register (BAR) is used to request PCI enumeration software to assign a memory mapped I/O (MMIO) address range for the controller to use for it's required USB host controller *capability* and *operational* registers. The designer hard- codes lower bits in this BAR indicating how many registers it has and whether it can decode 64 bit addresses (as a PCI target). Enumeration software later programs upper bits in the BAR to assign a start MMIO address to the controller.

SBRN (8 bits) - *Serial Bus Release Number Register* .
PCI Configuration Space Address: Byte Offset 60h
Default Register Value: see example below
Field: Read-Only (RO)

The designer of the PCI-based controller hard-codes the release of the USB Specification to which the controller complies. USB 2.0 controllers would return the following byte of information when this field is read: 20h (revision 2.0 of the USB Specification)

FLADJ (8 bits) - *Frame Length Adjustment Register.*
PCI Configuration Space Address: Byte Offset 61h
Default Register Value: 20h (decimal value 32 = 60000 ticks)
Field: Read-Write (RW)

This register is implemented so that software may compensate for variations in the USB bit clock. Normally, a High Speed USB micro-frame is derived by counting 60000 ticks of a 480MHz clock (yields a micro-frame time of 125uS). When the bit clock is slightly faster or slower than 480MHz, the six bits in this register allow defining the frame length (SOF cycle time) as 59488 to 60496 ticks--in 16 tick increments, which brings the SOF time back to 125uS. The six bit value has a range of 0-63 (decimal); the default value is mid-range--32 decimal). Once programmed, the value in this register is not generally changed.

PWAKECAP (16 bits) - *Port Wake Capability Register* (option).
PCI Configuration Space Address: Byte Offset 62h
Default Register Value: implementation dependent
Field: Read-Write (RW)

If this register is implemented, it is used to store a set of flag bits for each port the controller supports (up to 16). The state of each bit is programmed to 1 if the port <u>can be</u> enabled as a wake-up device in the event of a connect/disconnect or over current; if not, the bit is programmed to 0. This bit mask has no direct effect on controller hardware, and is instead used to maintain the current state of wake up capabilities.

PMC and PMCSR (32 bits each) - *Power Management Registers.*
PCI Configuration Space Address: device dependent (advanced capability register set)
Default Register Value: implementation dependent
Field: Read-Write (RW)

EHCI integrates advanced power management features compliant with the *PCI Bus Power Management Interface Specification*, revision 1.1. Table 1 below summarizes the EHCI optional and required power management states. Refer to MindShare's **PCI System Architecture** book for a detailed description of the PCI power management registers required by the EHCI specification.

Table 1: EHCI Power Management States

PCI Power Management State	Required/ Optional	Characteristics
D0	Required	Fully functional state, full power mode, enabled interrupts will be seen
D1	Optional	USB in Sleep state, all ports in suspend state, host controller bus mastering disabled, logic in low latency (can return quickly to D0) power savings.
D2	Optional	USB in Deep Sleep state, all ports in suspend state, host controller bus mastering disabled.
D3hot	Required	USB in Deep Sleep state, all ports in suspend state, host controller bus mastering disabled.
D3cold	Required	Fully asleep. Downstream devices either suspended or disconnected (depending on controller's downstream power capabilities)

How EHCI PCI Configuration Information Is Used

As is the case with all other PCI devices, the PCI Configuration space of a USB 2.0 controller is accessed primarily during boot time PCI bus enumeration, for two basic reasons:

Find Out About The Device.

Information hard-coded into the PCI configuration space by the device designer is read by system software (e.g. BIOS) using PCI *Configuration Read* cycles to determine the capabilities and system requirements of the device. As indicated by the PCI registers described above, an EHCI controller will report both basic and USB-specific PCI information such as Vendor ID, Class Code, memory (or MMIO) requirements, whether it uses interrupts (it does), whether it is a 2.0 controller (it is), etc. All of this information is useful to the system as it assigns resources, locates proper drivers, etc.

Program Supported Plug-and-Play Features.

Having read the PCI configuration space of the EHCI controller, system software later uses PCI *Configuration Write* cycles to program it's MMIO Base Address Register,   interrupt line routing, **FLADJ** register to compensate for USB clock deviation, wake-up options in **PORTWAKECAP** register, **PMC** and **PMCSR** power management register set, etc. Finally, system software will enable PCI **COMMAND** register bits enabling the EHCI controller to decode PCI transactions, act as a bus master, etc.

EHCI Host Controller Memory Mapped I/O Registers

While PCI configuration registers are only required if the host controller resides on a PCI bus, the memory mapped I/O (MMIO) USB registers described here are required for all EHCI implementations. Regardless of the host controller used, these registers form the heart of the generic programmer's interface. The USB EHCI MMIO registers fall into two sets, *Capability* registers and *Operational* registers. A single memory mapped I/O address range is used to access both sets of registers; the first group of addresses is for the capability registers, and the operational registers immediately follow. Figure 6 below illustrates the two sets of registers, which are addressed at adjacent MMIO address ranges, starting at the BASE address programmed into the PCI configuration BAR. If the EHCI host controller is not a PCI device, determination of the starting address for the MMIO registers is implementation specific.

Figure 6: EHCI Host Controller MMIO Register Sets
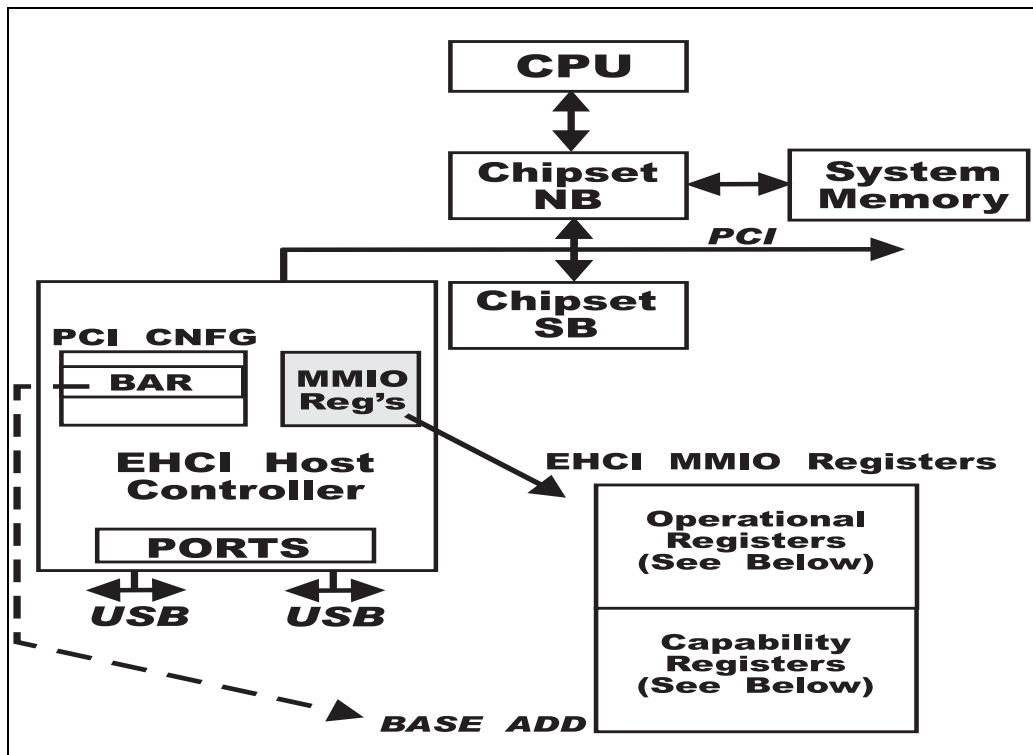
EHCI Host Controller Capability Registers

This set of read-only registers are implemented so that software can determine the capabilities of a particular EHCI host controller. Information returned includes:

- EHCI version supported
- Number of downstream ports and companion (USB 1.x) controllers
- 64 bit addressing capability (when controller acts as a bus master accessing memory data structures)
- Companion port routing information (indicating the routing of each physical port to it's companion controller if a USB 1.x device happens to be attached)

Table 2: EHCI Capability Register Set

Base Offset	Size	Mnemonic	Register Name
00h	1 Byte	**CAPLENGTH**	Capability Length
01h	1 Byte	Reserved	N/A
02h	2 Bytes	**HCIVERSION**	Version Number
04h	4 Bytes	**HCSPARAMS**	Structural Parameters
08h	4 Bytes	**HCCPARAMS**	Capability Parameters
0Ch	60 bits, maximum (15 ports x 4 bits ea.)	**HCSP-PROUTE**	Companion Port Routing

Capability Register Descriptions

CAPLENGTH - *Capability Register Length* .

Address: Capability Base Address + 00h bytes
Default Register Value: implementation dependent (byte offset to operational registers)
All fields: Read-Only (RO)

Table 3: CAPLENGTH Register Fields

Bit	Description
07:00	The value hard-coded in this field indicates the byte offset from the EHCI host controller MMIO BASE address to the first operational register (in other words, the number of bytes used to implement the capability register set). The operational register addresses always start at the end of the capability registers, and software uses the value in this register to offset into the operational registers.

HCIVERSION - *Host Controller Interface Version Number* .

Address: Capability Base Address + 02h bytes
Default Register Value: implementation dependent
All fields: Read-Only (RO)

Table 4: HCIVERSION Register Fields

Bit	Description
15:00	The value hard-coded in this field indicates the USB EHCI revision number to which the host controller conforms (e.g. 0095 = EHCI version 0.95)

Enhanced Host Controller Interface

HCSPARAMS - *Host Controller Structural Parameters.*
Address: Capability Base Address + 04h bytes
Default Register Value: implementation dependent
All fields: Read-Only (RO)

Table 5: HCSPARAMS Register Fields

Bit	Description
31:24	**Reserved.**
23:20	**Debug Port Number.** Optional field indicating the port number (0-15) used as the debug port. High-speed debug transactions can be sent over USB to a debug device. Debug port use is described in Appendix C of the EHCI Specification.
19:17	**Reserved**
16	**Port Indicator.** Read-only bit indicating whether the ports of this controller support port indicator control. • 1 = Indicates that software may write the ***Port Indicator Control*** bits in the **Port Status and Control Register** to light up red, green, amber port indicator lights. • 0 = Indicates that port indicators are not supported.
15:12	**Number Of Companion Controllers (N_CC).** These four bits are coded to indicate how many companion (UHCI or OHCI USB) controllers are resident in this host controller to support LS or FS USB 1.x devices.
11:08	**Number Of Ports Per Companion Controller (N_PCC).** Indicates the number of ports supported by each companion controller (1-15d). This information, along with **N_CC** and **N_PORTS** fields are used to determine companion controller routing.
07	**Port Routing Rules.** This bit indicates which of two conventions has been used in companion controller routing. • 0 = First N_PCC ports are routed to first companion host controller, next N_PCC ports are routed to next one, etc. • 1 = Port routing is explicitly defined in the **HCSP-PORTROUTE** register array.
06:05	**Reserved**

Table 5: HCSPARAMS Register Fields

Bit	Description
04	**Port Power Control.** Used to indicate host controller port power control. • 1 = The host controller includes port power control. • 0 = No port power control
03:00	**Number Of Ports (N_PORTS).** Indicates the number of downstream ports supported by the host controller (1-15d).

HCCPARAMS - *Host Controller Capability Parameters.*
Address: Capability Base Address + 08h bytes
Default Register Value: implementation dependent
All fields: Read-Only (RO)

Table 6: HCCPARAMS Register Fields

Bit	Description
31:08	**Reserved.**
07:04	**Isochronous Scheduling Threshold.** This field is used to inform system software when, relative to where the host controller is currently executing, it can safely update the isochronous schedule. Each isochronous transfer descriptor (iTD) contains 8 micro-frames of transactions. To reduce traffic to memory, the host controller is allowed to fetch and cache more than one (up to eight) micro-frame's transactions from an iTD. The amount of caching is reported in this register because it must be taken into account when USB system software is updating entries in the schedule to stay ahead of the host controller. Two important cases: **Bit 7 hard-coded = 0.** Then lower bits (6:4) represent the number of micro-frames worth of transactions cached by the host controller. **Bit 7 hard-coded = 1.** Indicates that the host controller caches iTD data structures for an entire frame (8 micro-frames)
03:02	**Reserved**

Table 6: HCCPARAMS Register Fields

Bit	Description
01	**Programmable Frame List Flag.** Indicates whether or not the host controller permits a frame list size other than the standard 1024 entries. 1 = Host controller allows programming the frame list size via the **USBCMD** register *Frame List Size* field. Frame list still must start on a 4KB aligned address boundary. 0 = Host controller does not allow a frame list size other than 1024 entries; *Frame List Size* field in **USBCMD** register is not implemented.
00	**64 Bit Addressing Capability.** Indicates whether or not the host controller is capable of using 64-bit address pointers during it's interactions with memory data structures. 1 = Host controller is capable of generating addresses greater than 32-bits when accessing memory as a master. This means it expects to use the 64 bit data structures described in Appendix B of the EHCI Specification. 0 = Host controller is not capable of generating addresses greater than 32-bits when accessing memory as a master. This means it expects to use the 32 bit data structures described in the EHCI Specification.

HCSP-PORTROUTE - *Companion Port Routing.*
Address: Capability Base Address + 0Ch bytes
Default Register Value: implementation dependent
All fields: Read-Only (RO)

There are two conventions used to convey routing of companion controllers in EHCI: one is to use the default routing of ports starting with the lowest numbered companion controller's ports, and proceeding in order through higher numbered companion controllers and their ports. The other convention is to use the bits in this **HCSP-PORTROUTE** register array to explicitly define routing

on a port by port basis. Each of the physical ports (there can be 1-15 of them) can be routed to any of the companion controllers in the event a USB 1.x device is attached. This requires an array of 60 bits (15 x 4 bits/port) to completely define port routing. Note: for PCI USB 2.0 host controllers, the controller is a multi-function device, and the companion controller number is related to it's PCI function number: Companion Controller 0 (CC0) maps to the first companion controller PCI function; CC 1 to the second companion controller function, etc.

Table 7: HCSP-PORTROUTE Bit Assignment

Port Number	Byte, Bits Used
15	Byte 7, bits 03:00
14	Byte 6, bits 07:04
13	Byte 6, bits 03:00
12	Byte 5, bits 07:04
11	Byte 5, bits 03:00
10	Byte 4, bits 07:04
09	Byte 4, bits 03:00
08	Byte 3, bits 07:04
07	Byte 3, bits 03:00
06	Byte 2, bits 07:04
05	Byte 2, bits 03:00
04	Byte 1, bits 07:04
03	Byte 1, bits 03:00
02	Byte 0, bits 07:04
01	Byte 0, bits 03:00

EHCI Host Controller Operational Registers

This set of registers are implemented so that software has a generic way of managing the EHCI host controller functionality. Collectively, these registers allow:

- Programming the host controller with starting addresses in memory for the data structures it must access in processing asynchronous and periodic schedules.
- Programming error, interrupt, and wake-up policies to be used by the controller
- Handling reset and power management events
- Logging and collecting host controller status concerning internal and external events: asynchronous and periodic schedule status, errors encountered, changes in port connection, over-current, etc.

Note that the operational registers are located in USB memory mapped I/O address space immediately after the capability registers.

Table 8: EHCI Operational Register Set

Offset	Size (Bytes	Mnemonic	Register Name
00h	4	**USBCMD**	USB Command
04h	4	**USBSTSS**	USB Status
08h	4	**USBINTR**	USB Interrupt Enable
0Ch	4	**FRINDEX**	USB Frame Index
10h	4	**CTRLDSSEGMENT**	4G Segment Selector
14h	4	**PERIODICLIST-BASE**	Frame List Base Address
18h	4	**ASYNCLISTADDR**	Next Asynch List Address
1C-3Fh	N/A	Reserved	
40h	4	**CONFIGFLAG**	Configured Flag Register
44h	4	**PORTSC**	Port Status Control

Operational Register Descriptions

USBCMD - *USB Command Register.*
Address: Operational Base Address + 00h
Default Register Value: 00080000h
Mixed fields: Read-Only (RO), Write-Only (WO), Read-Write (R/W)

Table 9: USBCMD Register Fields

Bit	Description
31:24	**Reserved. Set = 0.**
23:16	**Interrupt Threshold (R/W).** Value coded in this field sets maximum rate at which host controller may generate interrupts. Valid codes are summarized below: • 00h Reserved • 01h 1 micro-frame interrupt • 02h 2 micro-frame interrupt • 04h 4 micro-frame interrupt • 08h 8 micro-frame interrupt (1ms--This is the default value) • 10h 16 micro-frame interrupt (2ms) • 20h 32 micro-frame interrupt (4 ms) • 40h 64 micro-frame interrupt (8 ms)
15:08	**Reserved**
07	**Light Host Controller Reset (R/W).** Optional; allows the EHCD software to reset only the controller without impacting port setup or routing to companion controllers. If this bit is written = 1, the light host controller reset commences. If software later reads this bit clear (= 0), the reset is complete.
06	**Interrupt on Async Advance Doorbell (R/W).** Setting this bit = 1 requests the host controller to initiate an interrupt the next time it advances the asynchronous schedule (bit 5 in the **USBINTR** register must also be set to enable the interrupt). The host controller clears this bit after initiating the interrupt process.
05	**Asynchronous Schedule Enable (R/W).** Used to enable host controller processing of the asynchronous schedule. • 0 = Don't process the asynchronous schedule (This is the default) • 1 = Process the asynchronous schedule using **ASYNCLISTADDR** register as a pointer to the schedule.

Table 9: USBCMD Register Fields

Bit	Description
04	**Periodic Schedule Enable (R/W).** Used to enable host controller processing of the periodic schedule. • 0 = Don't process the periodic schedule (This is the default) • 1 = Process the periodic schedule using the **PERIODICLIST-BASE** register as a pointer to the beginning of the schedule.
03:02	**Frame List Size (R/W or RO)**. This field is used to set the frame list to one of three sizes: • 00b = 1024 elements in a frame list (This is the default) • 01b = 512 elements in a frame list • 01b = 256 elements in a frame list • 00b = Reserved Note: The value in this register is only R/W if the ***Programmable Frame List Size*** flag in the **HCCPARAMS** register is hard-coded to be = 1. Otherwise, the frame list size is always 1024 elements.
01	**Host Controller Reset (R/W)** This bits allows a software reset of the host controller, and has the following effects: Resets internal host controller pipelines and internal state machines Terminates any USB transactions in progress <u>Does not</u> cause a USB reset or affect any PCI registers • 1 = Commence host controller reset • 0 = If later read as 0, the reset is complete
00	**Run/Stop (R/W).** Used to start and stop the host controller in it's execution of both schedules. • 1 = Start and continue execution of all schedules enabled in bits 4 and 5 of this register. • 0 = Don't execute schedules. If executing, and software writes this bit = 0, current and pipelined operations are completed. The controller then halts and sets ***HCHalted*** bit in the **USBSTS** register. Host controller <u>must</u> halt within 16 micro-frames of detection of Run/Stop bit = 0. Note that this impacts the amount of pipelining allowed. Software verifies the halt state by reading the ***HCHalted*** bit mentioned previously.

USBSTS - *USB Status Register* .

Address: Operational Base Address + 04h
Default Register Value: 00001000h
Mixed fields: Read-Only (RO), Write-Only (WO), Read-Write (R/W), Read/Write to clear (R/WC)

Table 10: USBSTS Register Fields

Bit	Description
31:16	**Reserved. Set = 0.**
15	**Asynchronous Schedule Status (RO).** Used by the host controller to report the actual internal status of asynchronous schedule processing. When software enables or disables asynchronous schedule execution in the **USBCMD** register (bit 5), it may take some time before the change takes place. When complete, host controller hardware sets or clears this bit to reflect the fact. Anytime this bit and the equivalent enable bit in the **USBCMD** register are in the same state, software can confirm the change is complete.
14	**Periodic Schedule Status (RO).** Used by the host controller to report the actual internal status of periodic schedule processing. When software enables or disables periodic schedule execution in the **USBCMD** register (bit 4), it may take some time before the change takes place. When complete, host controller hardware sets or clears this bit to reflect the fact. Anytime this bit and the equivalent enable bit in the **USBCMD** register are in the same state, software can confirm the change is complete.
13	**Reclamation (RO).** Status bit which is used to detect an empty asynchronous schedule. Default is 0.
12	**HCHalted (RO).** Used by the host controller to report the actual internal status of host controller processing. When software clears (writes = 0) the *Run/Stop* bit in the **USBCMD** register (bit 0), it may take some time before the change takes place as the host controller finishes up operations it has pipelined. When complete, host controller hardware sets the **HCHalted** bit to reflect the fact that is really is stopped. Software reads this bit to confirm the halt operation is complete.
11:06	**Reserved. Set = 0.**

Table 10: USBSTS Register Fields

Bit	Description
05	**Interrupt On Asynch Advance (R/WC).** This bit is asserted by host controller hardware to indicate that an interrupt was generated as requested on the last advance of the asynchronous schedule. The request to interrupt occurs when software writes bit 6 of the **USBCMD register**. • 0 = No interrupt was generated by the host controller for this purpose. (This is the default) • 1 = An interrupt was generated as requested; system software writes this bit to clear the interrupt flag.
04	**Host System Error (R/WC).** This bit indicates a general catastrophic error within the host controller. The host controller also clears the Run/Stop bit (bit 0) in the **USBCMD** register to halt all processing.
03	**Frame List Rollover (R/WC).** This bit is set = 1 by host controller hardware each time the frame list "rolls over" from it's maximum value to 0. Normally this occurs after 1024 entries have been completed, but if a programmable frame list size is in use (see **USBCMD** register bits 3:2) then the rollover may happen at 1024, 512, or 256 boundaries. Software may read this bit to verify the rollover, then write the bit to clear it.
02	**Port Change Detect (R/WC)** This bit is set by host controller hardware to indicate a change has occurred on one or more of it's ports, including: • Change in connect status • A port *resume* event Once set, the bit may be read by software, and cleared when the bit is written = 1.
01	**USB Error Interrupt (R/WC).** This bit is set = 1 by host controller hardware when a USB transaction has resulted in an error condition. If the transfer descriptor that had this occur during execution had the IOC bit set, then USBINT is also set (see bit 0).
00	**USB Interrupt (R/WC).** This bit is set = 1 by the host controller at the completion of a USB transaction which resulted in retirement of a transfer descriptor which had its IOC bit set. It is also set in the event of a short packet while receiving data. Software clears this interrupt flag by writing this bit = 1.

USBINTR - *USB Interrupt Enable Register* .
Address: Operational Base Address + 08h
Default Register Value: 00000000h
All fields: Read-Write (R/W)

Table 11: USBINTR Register Fields

Bit	Description
31:06	**Reserved. Set = 0.**
05	**Interrupt On Async Advance Enable (R/W).** This bit is used in conjunction with the *Interrupt On Async Advance* bit (bit 5) in the **USBSTS** register. If this bit is set = 1, then anytime the *Interrupt On Async Advance* bit in **USBTS** is set by the host controller, an interrupt will also be generated. When the interrupt is received, software may then poll **USBSTS** register bit 5 to check the flag. This interrupt flag is cleared when software writes **USBSTS** register bit 5 = 1.
04	**Host System Error Enable (R/W).** This bit is used in conjunction with the *Host System Error Status* bit (bit 4) in the **USBSTS** register. If this bit is set = 1, then anytime the *System Error Status* bit in **USBTS** is set by the host controller, an interrupt will also be generated. When the interrupt is received, software may then poll **USBSTS** register bit 4 to check the flag. This interrupt flag is cleared when software writes USBSTS register bit 4 = 1.
03	**Frame List Rollover Enable (R/W).** This bit is used in conjunction with the *Frame List Rollover* bit (bit 3) in the **USBSTS** register. If this bit is set = 1, then anytime the *Frame List Rollover* bit in **USBTS** is set by the host controller, an interrupt will also be generated. When the interrupt is received, software may then poll **USBSTS** register bit 3 to check the flag. This interrupt flag is cleared when software writes **USBSTS** register bit 3 = 1.
02	**Port Change Interrupt Enable (R/W).** This bit is used in conjunction with the *Port Change Detect* bit (bit 2) in the **USBSTS** register. If this bit is set = 1, then anytime the *Port Change Detect* bit in **USBSTS** is set by the host controller, an interrupt will also be generated. When the interrupt is received, software may then poll **USBSTS** register bit 2 to check the flag. This interrupt flag is cleared when software writes **USBSTS** register bit 2 = 1.

Enhanced Host Controller Interface

Table 11: USBINTR Register Fields

Bit	Description
01	**USB Error Interrupt Enable (R/W).** This bit is used in conjunction with the *USBERRINT* bit (bit 1) in the **USBSTS** register. If this bit is set = 1, then any-time the *USBERRINT* bit in **USBSTS** is set by the host controller, an interrupt will also be generated. When the interrupt is received, software may then poll **USBSTS** register bit 1 to check the flag. This interrupt flag is cleared when software writes **USBSTS** register bit 1 = 1.
00	**USB Interrupt Enable (R/W).** This bit is used in conjunction with the *USBINT* bit (bit 0) in the **USBSTS** register. If this bit is set = 1, then anytime the *USBINT* bit in **USBSTS** is set by the host controller, an interrupt will also be generated. When the interrupt is received, software may then poll **USBSTS** register bit 0 to check the flag. This interrupt flag is cleared when software writes **USBSTS** register bit 0 = 1.

FRINDEX - *Frame Index Register* .

Address: Operational Base Address + 0Ch
Default Register Value: 00000000h
All fields: Read-Write (R/W)

This register is used in processing periodic frame list entries, and is indexed automatically by host controller hardware every micro-frame. Because this register tracks progress through the **frame** list, each frame list entry must be accessed eight times to obtain the eight micro-frames of work. The host controller uses the current value in this register as a pointer to where it is to go next in the periodic schedule.

System software reads and uses the contents of this register to keep track of the current micro-frame number so it can stay ahead in transaction scheduling and so it may return the current micro-frame number to a client driver which requests it (there is a **get micro-frame number** function which system software is required to support).

If system software is required to initialize (write to) the **Frame Index Register**, it may only do so when the host controller is in the halted state (see the **HCHalt** bit in the **USBSTS** register)

Note that this register uses fourteen bits: The upper 10 bits represent the current Frame List entry number (1024 max), and the lower 3 bits are the micro-frame offset in the entry.

Table 12: FRINDEX Register Fields

Bit	Description
31:14	**Reserved.**

13:00	**Frame Index (R/W, auto indexed).** Depending on the frame list size in use (generally 1024, but programmable to 256 or 512 in some implementations), the upper bits in this group indicate the current frame list entry number. The lower three bits reflect the micro-frame offset into the frame list entry. For various frame list sizes, the bits are used as follows:
	Frame list size
	1024: Bits 12:03 = frame list entry; Bits 2:0 = micro-frame number
	  512: Bits 11:03 = frame list entry; Bits 2:0 = micro-frame number
	  256: Bits 10:03 = frame list entry; Bits 2:0 = micro-frame number

CTRLDSSEGMENT - *Control Data Structure Segment Register* .
Address: Operational Base Address + 10h
Default Register Value: 00000000h
All fields: Read-Write (R/W)

This register is only used when 64-bit addressing capability is built into the host controller (indicated in **HCCPARAMS** register, bit 0 set = 1). Software then programs this 32-bit register with the upper half of the 64 bit address to be used by the host controller when it accesses it's data structures in memory. Once this register is programmed, the host controller concatenates this register value with the 32-bit link-pointers in the **PERIODICLISTBASE** register or the **ASYNCLIS-TADDR** register, depending on whether it is accessing the periodic or asynchronous schedules.

One of the results of using this register to provide the upper 32 bits of address for all data structure accesses is that they all then reside in the same 4GB segment of memory.

Table 13: CTRLDSSEGMENT Register Fields

Bit	Description
31:00	**Upper 32 bits of 64 bit data structure address (R/W).**

PERIODICLISTBASE - *Periodic Frame List Base Add Register* .
Address: Operational Base Address + 14h
Default Register Value: Undefined
All fields: Read-Write (R/W)

This register is programmed with the memory starting address for the periodic Frame List. If 64-bit addressing is being used, then the contents of this register are concatenated with the value programed into the **CTRLDSSEGMENT** register (see previous register description).

Using this register's contents as a 4KB aligned base address for the Periodic List, the host controller then steps sequentially through the list using the current contents of the **FRINDEX** register as an offset.

Table 14: PERIODICLISTBASE Register Fields

Bit	Description
31:12	**Periodic List base address (R/W).** * again, if 64 bit addressing is in use, these bits are concatenated with the value in the **CTRLDSSEGMENT** register to form the Periodic List base address.
11:00	**Reserved, must be 0.** Lower 12 bits of Periodic List base address must always be written = 0 to preserve 4KB aligned start address of Periodic List data structure.

ASYNCLISTADDR - *Current Asynchronous List Address Reg* .
Address: Operational Base Address + 18h
Default Register Value: Undefined
All fields: Read-Write (R/W, auto indexed)

This register contains the current address pointer for the next queue head in the asynchronous schedule entry to be processed. If 64-bit addressing is being used, then the contents of this register are concatenated with the value programed into the **CTRLDSSEGMENT** register.

Because each queue head entry address is assumed to be 32 byte (cache line) aligned, the lower 5 bits of this register cannot be modified by software and always read back as 0's.

Table 15: ASYNCLISTADDR Register Fields

Bit	Description
31:05	**Current Asynchronous Schedule Link Pointer (R/W, auto indexed).** * again, if 64 bit addressing is in use, these bits are concatenated with the value in the **CTRLDSSEGMENT** register to form the start address for the next asynchronous schedule queue head entry to be processed.
04:00	**Reserved, assumed to be 0.**

CONFIGFLAG - *Configure Flag Register* .

Address: Operational Base Address + 40h
Default Register Value: 00000000h
All fields: Read-Write (R/W)

Setting this bit is generally the last act of EHCI system software in host control-
ler configuration.

Table 16: CONFIGFLAG Register Fields

Bit	Description
31:01	**Reserved. Set = 0.**
00	**Configure Flag (R/W).** (Default = 0). This global bit determines the default scheme for port routing logic. • 0 = The default routing for each port is to a companion control-ler (OHCI or UHCI). • 1 = The default routing for each port is to the EHCI controller. If the attached device turns out to be USB 1.x, control will be passed to the companion controller specified by port routing set up. In the event no system EHCI driver exists, this bit "coming up 0" assures that compatibility with earlier USB 1.x controllers is maintained.

PORTSC - *Port Status and Control Register (s) .*

Address: Operational Base Address + 44h, 48h, 4Ch, etc.
Default Register Value: 00002000h with **HCSPARAMS** register *PPC* bit = 1
: 00003000h with **HCSPARAMS** register *PPC* bit = 0
Mixed fields: Read-Write (R/W), Read-Only (RO), Read-Write/Clear (R/WC)

A **Port Status and Control Register** is required for each port the host controller
supports, and the format for each one is the same.

A few notes about **PORTSC** registers:

- Software may determine how many ports are supported (and, therefore, how many **PORTSC** registers are in use) by reading various fields in the **HCSPARAMS** register.
- The default (initial) state for each port is: no device connected, port disabled.
- If the port has power control, which is also indicated in the **HCSPARAMS** register (*PPC* bit = 1), power must be applied and stable at the port before software attempts to change the state of the port.

Table 17: PORTSCn Register Fields

Bit	Description
31:23	**Reserved. Read back as 0's.**
22	**Wake on Over-Current Enable (R/W).** (Default = 0). This bit sets the port policy for generating a wake-up event in case of an over-current condition on USB. System software programs the bit as follows: • 0 = Instructs the port to not convert over-current conditions into wake-up events. • 1 = Instructs the port to convert over-current conditions into wake-up events. This field is 0 if port power is turned off
21	**Wake on Disconnect Enable (R/W).** (Default = 0). This bit sets the port policy for generating a wake-up event in case of a device disconnect on USB. System software programs the bit as follows: • 0 = Instructs the port to not convert disconnects into wake-up events. • 1 = Instructs the port to convert disconnects into wake-up events. This field is 0 if port power is turned off

Table 17: PORTSCn Register Fields

Bit	Description
13	**Port Owner (R/W, auto default).** (Default = 1). This bit indicates whether the port is under the management of the EHCI controller or a companion controller: • 0 = Port is owned and will be managed by the EHCI controller. • 1 = Port is owned and will be managed by the companion controller defined in port routing. This bit is automatically set = 1 whenever the ***Configure Flag*** bit (bit 0) in the **CONFIGFLAG** register is 0 (requesting companion controller(s) as default) This bit is automatically cleared (0) whenever the ***Configure Flag*** bit in the **CONFIGFLAG** register transitions from 0 to 1 (requesting EHCI controller as default). Software may also write this bit = 1 to pass ownership to the companion controller when it discovers the attached device is not a high-speed device.
12	**Port Power -PP (R/W or RO)**. This bit is used in conjunction with the ***PPC*** bit (bit 4) in the **HCSPARAMS** register. If ***PPC*** = 0, the host controller does not have power control switches on it's ports--power is hard-wired "on". In this case, this ***PP*** bit is read only, and always = 1. If ***PPC*** = 1, the host controller does have power control switches on each port--and it may be turned on and off using this register bit. In this case, this ***PP*** bit is read-write, and indicates the current state of port power. • ***PP*** = 1; Port power is on • ***PP*** = 0; Port power is off.

Table 17: PORTSCn Register Fields

Bit	Description
11:10	**Line Status (RO).** These bits are used when system software requires a sampling of the current state of the differential USB signal pair to determine if an attached device is low-speed (LS) or not. These bits must be read prior to port *reset and enable sequence.* Bit 11 reflects state of D+; Bit 10 reflects state of D-. Encoding on the bits is as follows: **Bits:** **Activate** 00b SEO state. (not LS device, perform EHCI reset) 10b J_State. (not LS device, perform EHCI reset) 01b K_State. (Is a LS device, release ownership of port*) 11b Undefined (not LS device, perform EHCI reset) * Port ownership is released when software writes **Port Owner** bit (bit 13) = 1. This field is undefined if port power is turned off
09	**Reserved. Read back as 0's.**
08	**Port Reset (RW).** This bit is used by system software to initiate and confirm a reset event on the USB port. The USB 2.0 Specification has a number of parameters which must be observed when performing resets. In general: • The controller should be halted (**HCHalt** = 1) before attempting reset • Software writes this bit = 1 to initiate reset, and must not clear the bit (write it = 0) to terminate a reset sequence before the specified time has elapsed. • After writing this bit = 1 to initiate the reset, software may then poll (read) the bit until it is detected = 0 (indicating completion) • There are also a number of requirements in the USB 2.0 specification concerning how quickly the controller must enable and disable it's port with respect to reset (e.g. the 2ms rule). This field is = 0 if port power is turned off

Enhanced Host Controller Interface

Table 17: PORTSCn Register Fields

Bit	Description

07	**Suspend (RW).** This bit is used in conjunction with the ***Port Enabled*** bit (bit 2) in this register to define port states:
	
	Port Enabled ***Suspend Bit*** **Port State**
	    0 X Disabled
	    1 0 Enabled
	    1 1 Suspend
	Notes:
	    • When a port is in the suspend state, normal downstream traffic is blocked, except for port reset.
	    • While in this state, the port is still capable of resume detection.
	    • If software sets this bit = 1 to suspend the port, a transaction in progress will be completed first, then the suspend will occur. Software may poll this bit until it is read = 0, indicating the suspend operation is complete.
	    • This bit will automatically be cleared (set = 0) in the event system software causes a reset (changes ***Port Reset*** bit from 0 to1) or causes a resume (changes ***Force Port Resume*** bit from 1 to 0)
	This field is = 0 if port power is turned off

06	**Force Port Resume (RW).** While suspending a port is fairly simple (see previous bit), forcing a port to resume signalling can be done through system software or by the host controller itself:
	    • Software writes this bit = 1 to initiate a resume on a suspended port.
	    • The host controller sets this bit = 1 if it detects a J-to-K transition on the USB port while it is suspended. The host controller also sets the ***Port Change Detect*** bit (bit) in the **USBSTS** register in this case. This, in turn, can be programmed to cause an interrupt to be generated
	The EHCI resume sequence is described in detail in the USB 2.0 Specification.
	
	This field is = 0 if port power is turned off

05	**Over-current Change (R/WC).** This bit is set = 1 if there is a change in the ***Overcurrent Active*** bit (see bit 4, below). In effect, this bit "latches" any change in over-current status while bit 4 reports current over-current status. Software may clear this bit by writing it = 1.

Table 17: PORTSCn Register Fields

Bit	Description
04	**Over-current Active (RO).** This bit is set = 1 if there is an over-current event on the USB port. The bit will clear itself if the over-current condition is removed.
03	**Port Enable/Disable Change (R/WC).** This bit is set = 1 if the port enabled/disabled status has changed due to an error. Software may clear this bit by writing it =1 This field is = 0 if port power is turned off
02	**Port Enabled/Disabled (R/W).** (default = 0) This bit is set = 1 by the host controller when the reset sequence is complete and it is verified that the port is attached to a high-speed device. Software may read it to confirm EHC attachment. Once enabled, a port can be disabled either by an fault or by software interaction. This field is = 0 if port power is turned off
01	**Connect Status Change (R/WC).** (default = 0) This bit is set = 1 by the host controller if there is a change if the ports ***Current Connect Status*** (see bit 0). Software may read it to get status, or write it = 1 to clear it. This field is = 0 if port power is turned off
00	**Current Connect Status (RO).** This bit is set = 1 by the host controller if there is a device connected to this port; it is cleared otherwise. This field is = 0 if port power is turned off

3. EHCI Registers: An Operational Summary

Before moving on to a description of the memory data structures used by an EHCI compliant controller, a final look at the initialization and use of the EHCI PCI and MMIO registers just covered is presented.

Initialization

The basic sequence of events involved in initializing an EHCI controller is as follows:

1. If the EHCI controller is a PCI device, then BIOS software will enumerate the EHCI host controller on power up or reset, including:

- Configuration Read cycles to check vendor ID, device ID, Class Code, memory mapped I/O requirements, interrupts, and other generic PCI attributes.
- Assign a base address for the EHCI Capability and Operational registers in BAR 0.
- Program the FLADJ register if supported by the device.

2. Following a power-up USB reset (or software HCReset), the EHCI MMIO operational registers will be in their default state. Table 18 below summarizes the initial (default) state of the EHCI MMIO operational registers.

Table 18: EHCI Operational Register Default Values

EHCI Operational Register	Default Value	Comments
USBCMD	00080000h	Interrupt threshold = 1mS, Frame List = 1024 entries, all execution disabled.
USBSTS	00001000h	Host controller halted, other status bits cleared.
USBINTR	00000000h	Interrupt option mask bits all cleared (no interrupt on error, port change, async advance, etc.)
FRINDEX	00000000h	Points to first entry in Frame List
CTRLDSSEGMENT	00000000h	Upper 32 bits of address = 0
PERIODICLISTBASE	Undefined	Start address of Periodic List is implementation specific.
ASYNCLISTADDR	Undefined	Start address of Asynchronous List is implementation specific.
CONFIGFLAG	00080000h	Default port routing = companion controller(s)

Table 18: EHCI Operational Register Default Values

EHCI Operational Register	Default Value	Comments
PORTSC* **(One PORTSC register per port)**	00003000h	Each port defaults to power-off and to ownership of companion controller(s).

3. Using offsets into the MMIO address space assigned to the controller during PCI enumeration, USB EHCI software (if present) may then:

- Perform a 32-bit MMIO write to the **CTRLDSSEGMENT** register to program the 4GB memory segment to be used for all EHCI memory data structures (Periodic List, Asynchronous List, data buffers, etc.). This register is optional; if implemented, it defaults to 0 on power-up as indicated above.
- Perform a 32-bit MMIO write to the **USBINTR** register to set option mask for USB interrupts.
- Perform a 32-bit MMIO write to the **PERIODICLIST BASE** register to establish the physical memory start address of the Periodic Frame List.
- Perform a 32-bit MMIO write to enable key features in the **USBCMD** register, including interrupt threshold (1mS is the default), frame list size (1024 is the default); the Run/Stop bit may also be enabled.
- Perform a 32-bit MMIO write to the **CONFIGLFLAG** register to change the default routing of all ports from the companion controllers to the EHCI controller.

The EHCI Controller Becomes Active

With the Run/Stop bit set and default port routing established, the EHCI controller is active, and can report attachment of devices. If any devices are already attached and under the control of the companion controllers, they will now appear to disconnect from the companion controllers. While the companion controllers are not aware of the EHCI controller, the disconnection of a device is something that they do understand; this helps keep USB 1.x companion controller software compatible when EHCI is added to the mix.

Once default routing is transferred to the EHCI controller, all port devices remain under the management of the EHCI controller unless they are determined to be USB 1.x devices; at that time, they will be returned back to the control of the companion controllers. As control of a USB 1.x device is passed to the companion controller, it appears as though it has been disconnected from the EHCI controller and connected to the companion controller. The USB 1.x companion controller then proceeds through it's standard protocol as the device is reset and enabled.

Detecting Attachment

Actions taken by an EHCI controller when responding to device attachment:

- The controller detects the attachment, and sets the *Port Change Bit* in the **PORTSC** register for that port. The controller also sets the *Current Connect Status* bit in the same register. If interrupts for changes in connect status are enabled, the host controller will also generate a port-change interrupt to the CPU at the next interrupt threshold.
- Upon receipt of the interrupt, the EHCI driver identifies the port with the change by examining the *Port Change Bit* in it's **PORTSC** register. Because the change bit is detected set, the EHCI driver sends a "change report" indicating the port number to the USB hub driver.
- The USB hub driver uses the information sent to it, and using the GetPortStatus(), request, reads the **PORTSC** register for that port to determine the nature of the change (in this case, the *Connect Status Change* bit will be read = 1, indicating a connect/disconnect event).
- The hub driver then issues a request to the EHCI driver to clear the *Connect Status Change* bit, and to perform a reset and enable of the port.

Discovering The Speed Capability Of The Attached Device

The EHCI portion of a USB 2.0 controller does not support either Low Speed or Full Speed USB ports. When a request to reset and enable a port is received, the first thing that must be done is to determine whether the device attached to the port is High Speed; if not, control will be passed to the companion controllers. Actions taken to map devices to the proper controller include:

- Upon receipt of the request to reset and enable the port which has just had a device attached, the EHCI driver checks the *LineStatus* bits in the **PORTSC** register for the port. If the detected device is HS or FS (D+ asserted) on the USB port, the driver sets the *PortReset* control bit and clears the *PortEnable* bit in the **PORTSC** register for that port. This starts the reset sequence on the port. The duration of the sequence is timed by software; when time is up, software clears the *PortReset* bit. Software verifies completion of reset when the *PortReset* bit is read back as 0.
- It is the responsibility of the EHCI controller hardware to set the *PortEnable* bit in the **PORTSC** register if the results of the reset indicated a high-speed device was attached. (Refer to **MindShare's** *USB System Architecture* book for a complete description of the High Speed detection mechanism). The EHCI driver reads the *PortEnable* bit; if set, the device is high-speed and the EHCI driver sends a "change report" to the hub driver which then continues USB enumeration of the device. If *PortEnable* bit is read back = 0, the device is Full Speed, and control is passed to the companion controller when the EHCI driver writes the *PortOwner* bit in the **PORTSC** register = 1. At that point, port routing logic makes the device visible to the companion controller; the EHCI controller registers a disconnect, and reports it via the *PortChange* bit mechanism described above.
- In the event that the attached device is recognized as Low Speed when the *LineStatus* bits are tested by the EHCI driver (D- asserted), then there is no need for the EHCI controller to perform the reset sequence; instead, control is immediately passed to a companion controller when the EHCI driver writes the associated *PortOwner* bit in the **PORTSC** register = 1.

If A Device Is Disconnected

Actions taken by a USB 2.0 controller when responding to device disconnect:

If the default port routing is EHCI, then any time a Low Speed or High Speed device is disconnected from a port attached to one of the companion controllers, the disconnect event is detected by the companion controller port control and by EHCI port control. The EHCI controller reclaims port ownership by setting the *PortOwner* bit in the **PORTSC** register, and signals a disconnect to the companion controller. This mechanism guarantees that if default routing is EHCI, any new attachment of devices will be checked *first* by the EHCI controller for High Speed operation. This is important because USB 1.x companion controllers don't support (and should never be attached to) High Speed devices and because High Speed devices are not required to be fully functional when running in Full Speed or Low Speed modes.

4. EHCI Memory Data Structures And How They're Used

The second part of managing an EHCI USB controller involves transaction *descriptors* set up in main memory by USB system software, to be fetched and executed by the host controller. In these data structures, control, status, and data are exchanged between high level software and the particular host controller doing the work. While the handling of EHCI data structures is similar to earlier UHCI and OHCI controller models, EHCI is made more complicated by such things as micro-frame scheduling and the addition of split transaction support. *Note:* in this section, the EHCI 32 bit structures are described and used as examples; refer to Appendix B in the EHCI Specification for information on 64 bit data structures.

EHCI employs two transfer schedules: the **periodic schedule** for time-sensitive *isochronous* and *interrupt* USB transfers, and the **asynchronous schedule** for less time-sensitive *control* and *bulk* USB transfers. As client software needs USB transactions performed, USB driver software decomposes each request into one or more entries in the proper schedule; meanwhile, the host controller may be fetching descriptors placed in the schedules previously, parsing them, and performing the required USB transactions. As the descriptors are *retired,* the host controller writes status information back into the data structures to let USB driver software know how things are going. To reduce hardware complexity, the host controller simply fetches and executes descriptors; all of the tricky scheduling and payload decisions are made and enforced by the USB driver software in the way it builds the schedules.

As described above, isochronous and interrupt transactions are processed from the periodic schedule. High-speed device isochronous transfer descriptors (iTD), Full Speed device split-transaction isochronous transfer descriptors (siTD), and queue head structure queue element descriptors (qTD) for interrupts are all accessed through the periodic schedule.

USB bulk and control descriptors in the asynchronous schedule use the queue head structure and queue element transfer descriptors (qTD). This includes split transaction interrupt, bulk, and control transfers.

In this section, processing by the host controller of the periodic and asynchronous schedules is described, as are the formats of four USB data structures found within them:

- isochronous transfer descriptor (iTD)
- queue head structure

- queue element transfer descriptor (qTD)
- split-transaction isochronous transfer descriptor (siTD)

The EHCI Controller And It's Transfer Schedules

Figure 7 below illustrates the periodic and asynchronous transfer schedules and the EHCI host controller operational registers used to generate addresses targeting them.

Figure 7: EHCI Controller And Memory Transfer Schedules
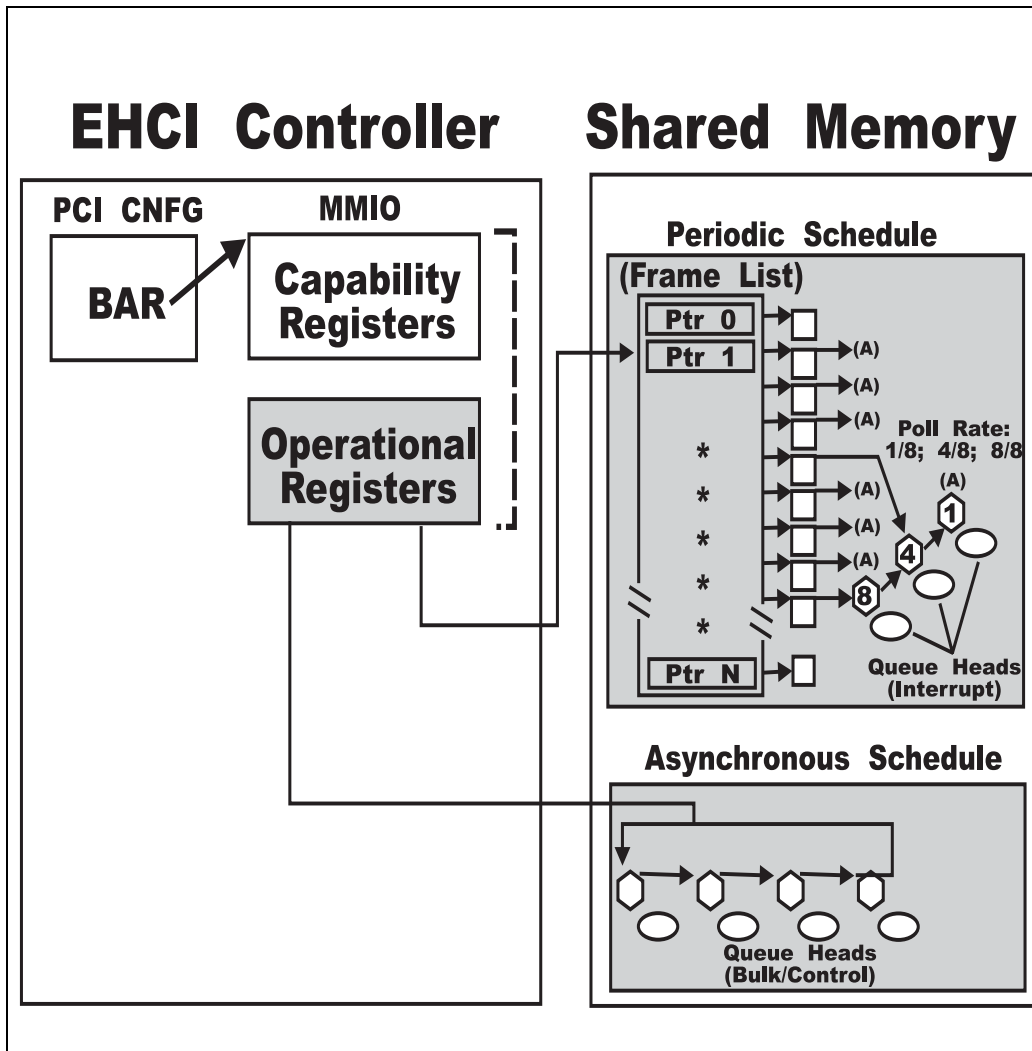
The Periodic Frame List

The periodic schedule is accessed using a set of pointers which reside in the **Periodic Frame List**. The length of the periodic frame list is either 256, 512 or 1024 entries (1024 is the default). Each entry in the frame list is a link pointer to the first task in the current micro-frame (125us time slot), and has the following characteristics:

- The link pointer is 32-byte aligned. This is convenient during CPU caching of descriptor fields.
- The link pointer may point to either an isochronous transfer descriptor (iTD) for HS devices, a split-transaction isochronous transfer descriptor (siTD) for a FS isochronous device, or a queue head for a LS/FS/HS interrupt.
- Information about the nature of the pointer (it's type) and whether or not it is valid is kept in the lower bits of the Frame list entry. See Figure 8 below.

Accessing Frame List Entries

Each 125us micro-frame the host controller indexes the **FRINDEX** offset from the **PeriodicListBase** start address, and fetches the next link pointer. As long as the periodic schedule is enabled, an EHCI compliant host controller must execute from the periodic schedule <u>as long as there are valid entries</u>. It begins execution of the asynchronous schedule only after it encounters the end of the periodic one (when it parses a link pointer and detects a **T** bit set = 1). It will then start fetching from the asynchronous schedule until the end of the current micro-frame, when it returns to the frame list

.

Figure 8: EHCI Frame List Link Pointer Format
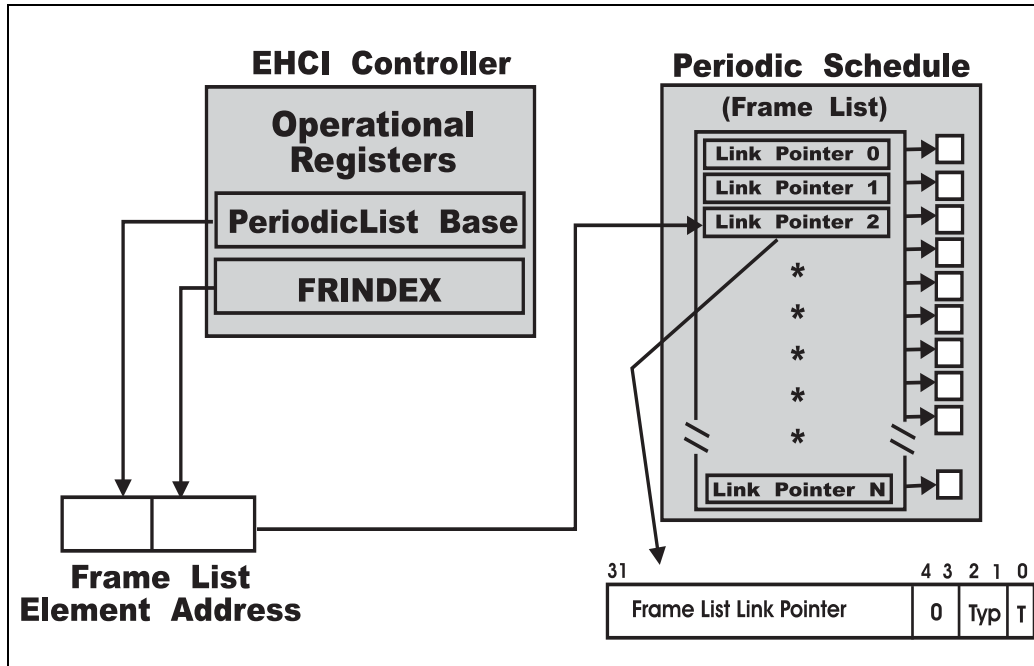
Table 19 below summarizes the bit fields in a frame list link pointer.

Table 19: Frame List Link Pointer Fields

Bit	Description
31:05	Physical address (32 byte aligned) of the first task descriptor in the current micro-frame (125us time slot).
04:03	**Reserved.** Initialize = 0.
02:01	**Type of task being referenced (TYP)** 00 = Isochronous Transfer descriptor (iTD) 01 = Queue Head structure (QH) 10 = Split Transaction Isochronous Transfer descriptor (siTD) 11 = Reserved.

Table 19: Frame List Link Pointer Fields

Bit	Description
00	**Terminate (T)**. This bit indicates whether or not this is the last element in the linked list of tasks. 0 = This is not the last task and the link pointer is valid. 1 = Terminate. Tis is the last task in this linked list.

The Asynchronous Schedule

When the host controller is finished processing tasks in the periodic schedule, it references the operational register **ASYNCLISTADDR**, which contains the pointer to the next queue head structure to be processed. Unlike the frame list, the asynchronous schedule is a simple circular buffer without any form of time base. However, the host controller must always be conscious to micro-frame timing because it is required to stop asynchronous processing and return to the periodic frame list at the beginning of each new micro-frame.

Figure 9 below illustrates the asynchronous schedule and the **ASYNCLIS-TADDR** register which maintains the address of the next queue head to be serviced when the host controller returns to it.

Figure 9: EHCI Asynchronous Schedule And The ASYNCLISTADDR Register
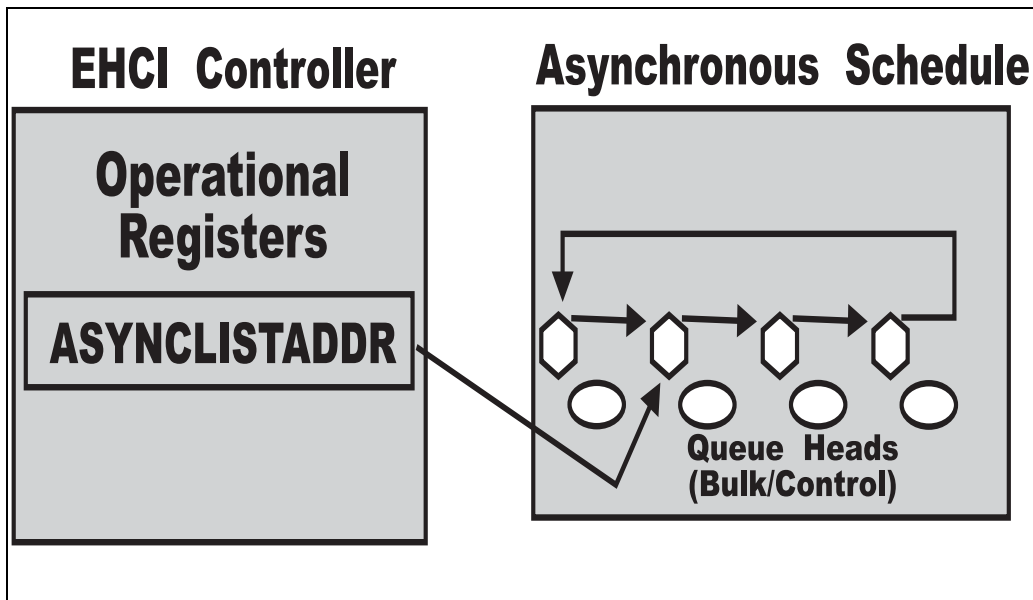
Micro-Frame Integrity

As it processes periodic and asynchronous schedule tasks, the host controller is always required to maintain micro-frame integrity; it is never allowed to start a transaction which will not be completed before the end of the current micro-frame. This is because it must observe USB 2.0 specification requirements for prompt delivery of SOF packets and high-speed EOF1 and EOF2 threshold timing.

Details Of Data Structure Bit Fields

The following tables and diagrams describe bit fields of the four data structure types defined in the EHCI Specification:

- **Isochronous Transfer Descriptors**
- **Split-Transaction Isochronous Transfer Descriptors**
- **Queue Head Structure**
- **Queue Head Element Transfer Descriptors**

Isochronous Transfer Descriptor Format (iTD)

Figure 10 below depicts the 64 byte structure of an isochronous transfer descriptor. Each descriptor defines 8 micro-frames (1 frame) worth of isochronous transactions, and the host controller must actually access each descriptor eight times to get them all.

Figure 10: EHCI Isochronous Transfer Descriptor (iTD)

DW			
01	Next Link Pointer	0 Typ T	03-00h
02	Status	Transaction 0 Length	IOC PG* Transaction 0 Offset*	07-04h
03	Status	Transaction 1 Length	IOC PG* Transaction 1 Offset*	0B-08h
04	Status	Transaction 2 Length	IOC PG* Transaction 2 Offset*	0F-0Ch
05	Status	Transaction 3 Length	IOC PG* Transaction 3 Offset*	13-10h
06	Status	Transaction 4 Length	IOC PG* Transaction 4 Offset*	17-14h
07	Status	Transaction 5 Length	IOC PG* Transaction 5 Offset*	1B-18h
08	Status	Transaction 6 Length	IOC PG* Transaction 6 Offset*	1F-1Ch
09	Status	Transaction 7 Length	IOC PG* Transaction 7 Offset*	23-20h
10	Buffer Pointer, Page 0	ENDPT R Device Address	27-24h
11	Buffer Pointer, Page 1	IO Maximum Packet Size	2B-28h
12	Buffer Pointer, Page 2	Reserved MULT	2F-2Ch
13	Buffer Pointer, Page 3	Reserved	33-30h
14	Buffer Pointer, Page 4	Reserved	37-34h
15	Buffer Pointer, Page 5	Reserved	3B-38h
16	Buffer Pointer, Page 6	Reserved	3F-3Ch

Bits: 31 30 29 28 ... 16 15 ... 12 11 ... 4 3 2 1 0

☐ (shaded) Host Controller R/W   ☐ Host Controller Read-Only   * May be modified by Host Controller if *IO* = OUT

Isochronous Transfer Descriptor Bits (refer to Figure 10)

Next Link Pointer. Offset in descriptor: first DWord.

Table 20: iTD Next Link Pointer Fields

Bit	Description
31:05	Physical memory address bits, 31:5 referencing the next transfer descriptor or queue head.
04:03	**Reserved.** (Initialize = 0).
02:01	**TYP** field. Indicates the type of structure being referenced. Type codes include: 00b = iTD (isochronous transfer descriptor) 01b = QH (queue head structure) 10b = siTD (split transaction transfer description) 11b = reserved
00	**T** bit. Terminate bit indicates whether next link pointer value is valid: 1 = invalid pointer 0 = valid pointer

iTD Transaction Status DWords. Offset in descriptor: DWord 1-8.

This set of eight DWords represent status/control for the 8 micro-frame transactions managed in this descriptor.

Table 21: iTD Transaction Status Field

Bit	Description
31:28	**Status** field. This set of bits tracks the status for the transaction defined in this slot, and executed by the host controller. Bits are encoded as follows: **Bit 31 = Active** bit. Set =1 by host software to enable execution by controller. When transaction is complete, the host controller clears it (0). **Bit 30 = Data Buffer Error** bit. Set =1 by host controller to indicate an overrun (couldn't buffer incoming data fast enough) or underrun (couldn't supply outbound data fast enough) during data transmission. **Bit 29 = Babble Detected** bit. Set =1 by host controller to indicate an babble detected during transmission. **Bit 28 = Transaction Error** bit. Set =1 by host controller only on IN transaction to indicate no valid response was received from device.
15	**IOC (Interrupt On Complete)** bit. If set = 1, when the transaction is complete, the host controller should generate an interrupt at the next interrupt threshold.
14:12	**PG (page Select)** field. These bits are set by software to indicate which of the buffer page pointers the offset field in this slot should be concatenated with to produce the memory buffer start address for this transaction. (Range 0-6)
11:00	**Transaction X Offset** field. These 12 bits provide the 4K byte offset to be concatenated with the buffer page pointer indicated above to produce the 32 bit start address for this transaction.

Enhanced Host Controller Interface

iTD Buffer Page Pointer List. Offset in descriptor: DWord 9-15.

This set of seven DWords represent page pointers (4KB aligned) to the data buffer for each descriptor. Collectively, these page pointers handle physical memory addressing for 7 pages of buffering (28KB) which is helpful because one data structure may call for 8 isochronous transfers of 3 transactions * 1024 bytes each (8 * 3 * 1024 = 24 KB). There are three slightly different formats for these seven DWords, as indicated in the tables below..

Table 22: iTD Buffer Pointer Page 0 Fields

Bit	Description
31:12	**Buffer Pointer** field. This is the 4KB-aligned physical start address to the memory buffer. This is concatenated with an offset provided by **Transaction X** (see previous register)
11:08	**Endpoint Number (EndPt)** field. This 4-bit value is the endpoint in the USB target device
07	**Reserved.** Initialize = 0.
06:00	**Device Address** field. These 7 bits are used for the USB device number (0-127)

Table 23: iTD Buffer Pointer Page 1 Fields

Bit	Description
31:12	**Buffer Pointer** field. This is the 4KB-aligned physical start address to the memory buffer. This is concatenated with an offset provided by **Transaction X** (see previous register)
11	**Direction (I/O)** bit. This bit indicates whether the high-speed transaction is an IN or OUT, and which PID should be used.
10:00	**Maximum Packet Size** field. These 11 bits are used to indicate the maximum packet size associated with an endpoint. For high bandwidth endpoints (which can handle up to three isochronous transfers per microframe), this field is used with **Multi** field (see next table). This field is also used during IN transactions to detect babble.

.

Table 24: iTD Buffer Pointer Page 2 Fields

Bit	Description
31:12	**Buffer Pointer** field. This is the 4KB-aligned physical start address to the memory buffer. This is concatenated with an offset provided by **Transaction X** (see previous register)
11:02	**Reserved.** Initialize = 0.
01:00	**Multi** field. These 2 bits are coded to indicate to the host controller how many transactions should be done per micro-frame: **00b** = Reserved. **01b** = One transaction per micro-frame **10b** = Two transactions per micro-frame **11b** = Three transactions per micro-frame

Table 25: iTD Buffer Pointer Page 3-6 Fields
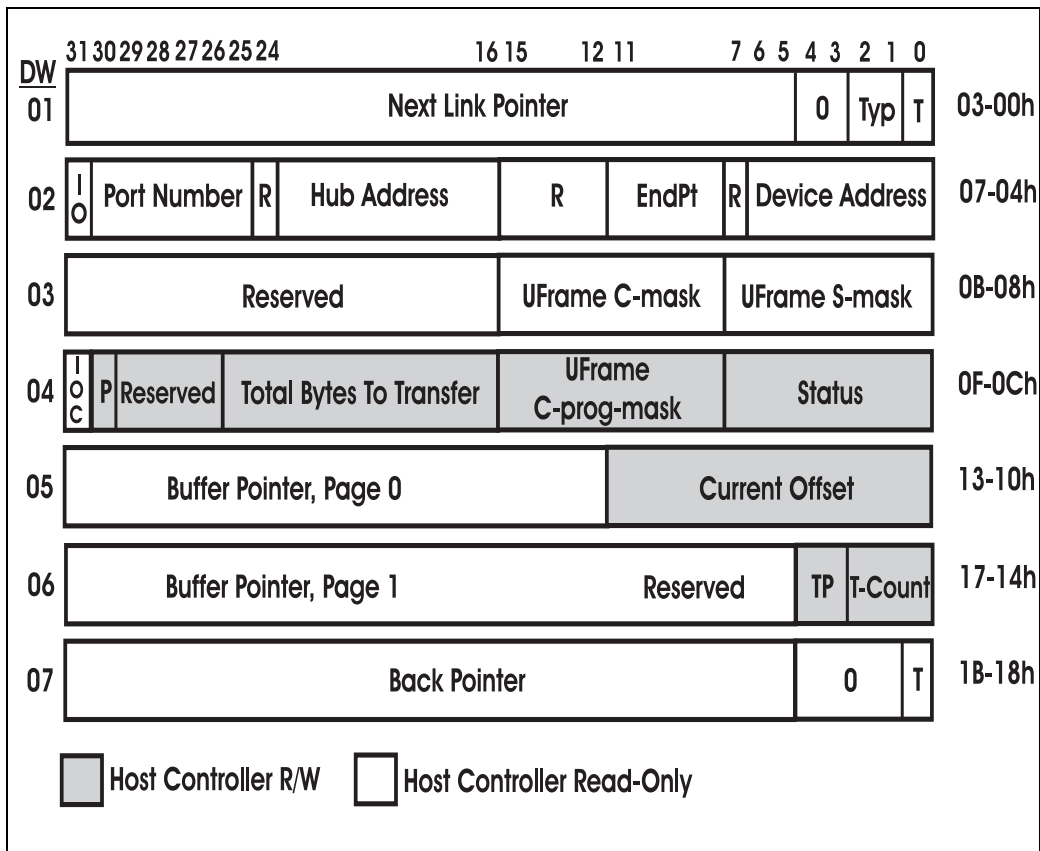
Bit	Description
31:12	**Buffer Pointer** field. This is the 4KB-aligned physical start address to the memory buffer. This is concatenated with an offset provided by **Transaction X** (see previous register)
11:00	**Reserved.** Initialize = 0.

Split Transaction Isochronous Transfer Descriptor (siTD)

Figure 11 below depicts the 28 byte structure of an split transaction isochronous transfer descriptor. These are used when a USB 2.0 controller is required to initiate an isochronous transaction with a Full Speed USB 1.x device.

Figure 11: EHCI Split-Transaction Isochronous Transfer Descriptor (siTD)

Split Transaction Isochronous Transfer Descriptor Fields

(refer to Figure 11)

Next Link Pointer. Offset in descriptor: first DWord.

Table 26: siTD Next Link Pointer Fields

Bit	Description
31:05	Physical memory address bits, 31:5 referencing the next transfer descriptor or queue head.
04:03	**Reserved.** (Initialize = 0).
02:01	**TYP** field. Indicates the type of structure being referenced. Type codes include: 00b = iTD (isochronous transfer descriptor) 01b = QH (queue head structure) 10b = siTD (split transaction transfer description) 11b = reserved
00	**T** bit. Terminate bit indicates whether next link pointer value (bits 31:05) is valid: 1 = invalid pointer 0 = valid pointer

siTD Endpoint Capabilities/Characteristics. Offset in descriptor: DWord 2.

Table 27: siTD Endpoint Capability/Characteristics Fields

Bit	Description
31	**Direction (I/O)**. Indicates whether the FS transaction is IN or OUT 1 = IN 0 = OUT
30:24	**Port Number.** Represents the port number of the companion controller handling this FS transaction.
23	**Reserved.** (Initialize = 0)

Table 27: siTD Endpoint Capability/Characteristics Fields

Bit	Description
22:16	**Hub Address.** Programmed with the device address of the transaction translators hub.
15:12	**Reserved.** (Initialize = 0)
11:08	**Endpoint Number (EndPt)** field. This 4-bit value is the endpoint in the USB target device
07	**Reserved.** Initialize = 0.
06:00	**Device Address** field. These 7 bits are used for the USB device number (0-127)

siTD Micro-frame Schedule Control. Offset in descriptor: DWord 3.

Table 28: siTD Micro-frame Schedule Control Fields

Bit	Description
31:16	**Reserved.** (Initialize = 0)
15:08	**Split Completion Mask (uFrame C-mask).** This field, in conjunction with the *Active* and *Split-X* fields in the **Status** byte are used to indicate the micro-frame in which the host controller should attempt *complete-split* transactions. Each bit represents one siTD; if a bit position = 1, then that siTD complete-split transaction may be executed.The lower three bits of the current FRINDEX register value are used to index into one of the 8 bit positions.
07:00	**Split Start Mask (uFrame S-mask).** This field, in conjunction with the *Active* and *Split-X* fields in the **Status** byte are used to indicate the micro-frame in which the host controller should attempt *start-split* transactions. Each bit represents one siTD; if a bit position = 1, then that siTD start-split transaction may be executed.The lower three bits of the current FRINDEX register value are used to index into one of the 8 bit positions.

siTD Transfer State. Offset in descriptor: DWord 3.

Table 29: siTD Transfer Status And Control Fields

Bit	Description
31	**Interrupt On Complete (IOC)** bit. If set = 1, when the transaction is complete, the host controller should generate an interrupt at the next interrupt threshold.
30	**Page Select (P)** bit. Used to determine which page pointer should be concatenated with the *CurrentOffset* field to form a data buffer pointer. 0 = page 0 pointer; 1 = page 1 pointer.
29:26	**Reserved.** (Initialize = 0)
25:16	**Total Bytes To Transfer.** Set by system software to indicate total number of bytes expected to transfer in this transaction. Maximum = 1023.
15:08	**uFrame Complete-split Progress Mask (C-prog-mask)** field. This 8-bit mask is is referenced by the lower bits of the **FRINDEX** register, and is used by the host controller to log which split-complete transactions have been executed. Split-completes must finish in order, and this mask is checked by the host controller to make sure that earlier ones have not been held off.

Table 29: siTD Transfer Status And Control Fields

Bit	Description

07:00	**Status** field**.** This field tracks the status of the transaction in this slot executed by the host controller. Bit coding is as follows:
	Bit 7 = Active bit. Set =1 by host software to enable execution of an isochronous split transaction by the controller.
	Bit 6 = Error (ERR) bit. Set = 1 by controller when an *ERR* response is received from the USB 1.x transaction translator.
	Bit 5 = Data Buffer Error bit. Set =1 by host controller to indicate an overrun (during an IN) or underrun (during an OUT) while transferring data. For underrun condition, host controller transmits an incorrect ECC to the endpoint, invalidating the data sent.
	Bit 4 = Babble Detected bit. Set =1 by host controller if babble was detected during this transaction.
	Bit 3 = Transaction Error (XactErr) bit. Set =1 (for IN transactions) by host controller if the host did not receive one of the valid responses from the target device.
	Bit 2 = Missed Micro-Frame bit. Set = 1 by controller if it missed a required complete-split transaction because it was "held off" from accessing memory in time.
	Bit 1 = Split Transaction State (SplitXstate) bit. This bit controls when the host controller is allowed to perform a start-split or complete-split transaction:
	    • 0 = do start-split when a match is detected in the proper bit in the S-mask.
	    • 1 = do complete-split when a match is detected in the proper bit in the C-mask.
	Bit 0 = Reserved. Initialize = 0.

siTD Buffer Pointer List (DW 4). Offset in descriptor: DWord 4.

Table 30: siTD Buffer Pointer List (DW4) Fields

Bit	Description
31:12	**Buffer Pointer List** field. These bits provide the upper bits (31:12) of the 4KB aligned page 0 memory (the lower 12 bits are assumed to be 0).
11:00	**Current Offset** field. These 12 bits represent the byte offset for the current page pointer (referenced by the *P* bit).

siTD Buffer Pointer List (DW 5). Offset in descriptor: DWord 5.

Table 31: siTD Buffer Pointer List (DW5) Fields

Bit	Description
31:12	**Buffer Pointer List** field. These bits provide the upper bits (31:12) of the 4KB aligned page 1 memory (the lower 12 bits are assumed to be 0).
11:05	**Reserved.**
04:03	**Transaction Position (TP)** field. This field is used in conjunction with ***T-Count*** field indicate whether the first, middle, or last part of a Full Speed data payload is being sent in this transaction. Initialized by system software, the host controller manages the state of these bits thereafter. Bit coding is as follows: 00 = All. The entire Full Speed data payload is being sent in this transaction. 01 = Begin. The first part of the Full Speed data payload is being sent in this transaction (that is greater than 188 bytes). 02 = Middle. The middle part of the Full Speed data payload is being sent in this transaction (that was greater than 188 bytes). 03 = End. The last part of the Full Speed data payload is being sent in this transaction (that was greater than 188 bytes).
02:00	**Transaction Count (T-Count)** field. Software programs this field with the number of out start-split transactions which will be required to fully satisfy the data payload for this transaction (6 is the maximum).

siTD Back Link Pointer. Offset in descriptor: DWord 6.

Table 32: siTD Back Link Pointer Fields
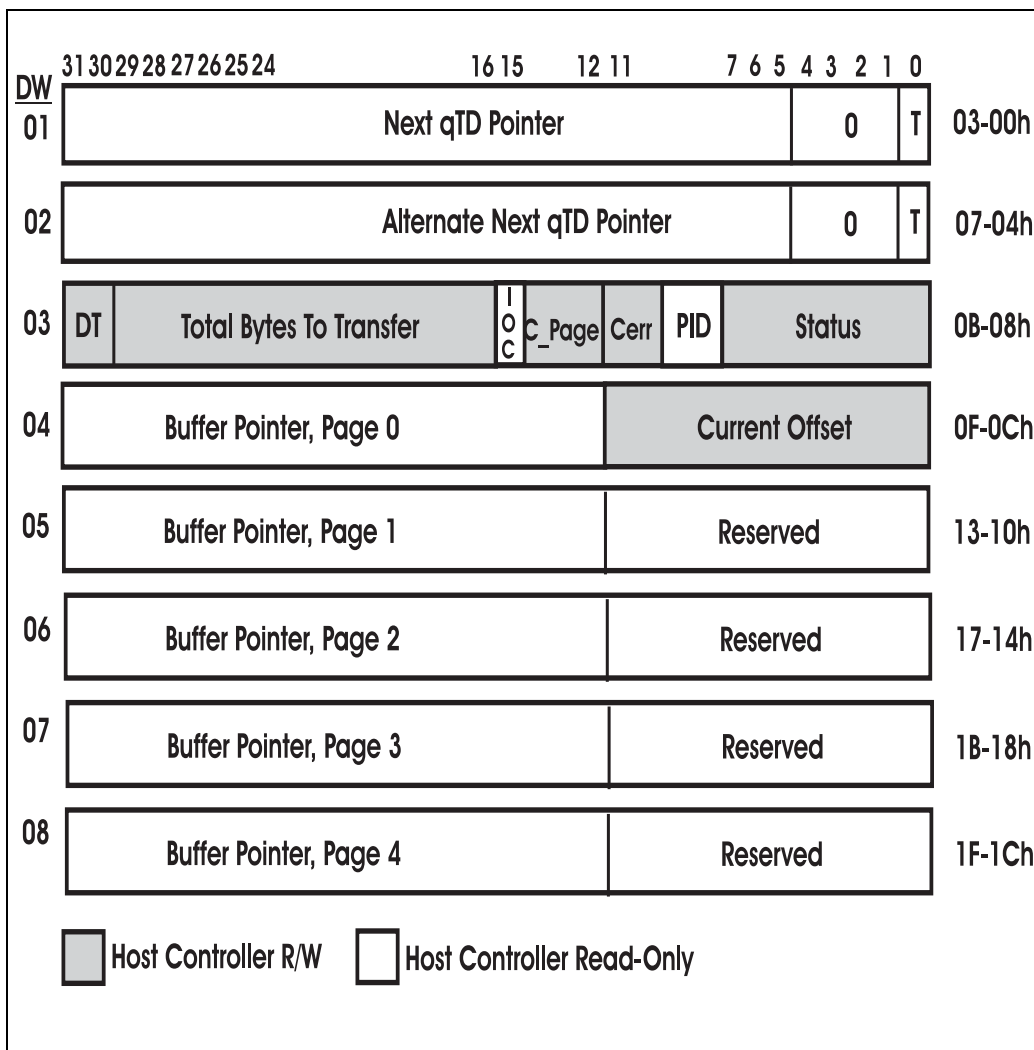
Bit	Description
31:05	**siTD Back Pointer** field. These bits are used as a physical memory pointer (32 byte aligned) to another siTD.
04:01	**Reserved.** (Initialize = 0)
00	**Terminate (T)** bit. Indicates whether **Back Pointer** field is valid or not. 0 = **Back Pointer** field is invalid 1 = **Back Pointer** field is valid

Queue Element Transfer Descriptor (qTD)

Figure 12 below depicts the 32 byte structure of an element used with a queue head. Each qTD structure will result in one or more USB transactions. The total data transfer resulting from processing a qTD may be as large as 20480 bytes, as indicated by the presence of 5 buffer page pointers in the structure (5 * 4KB = 20KB).

Figure 12: EHCI Queue Element Transfer Descriptor (qTD)

Queue Element Transfer Descriptor Fields

(refer to Figure 12)

Next qTD Pointer. Offset in descriptor: first DWord.

Table 33: siTD Next qTD Pointer Fields

Bit	Description
31:05	**Next Transfer Element Pointer** field. Physical memory address bits, 31:05, referencing the next qTD transfer descriptor to be processed
04:01	**Reserved.**
00	**T** bit. Terminate bit indicates whether the **Next Transfer Element Pointer** value above is valid: 1 = invalid pointer 0 = valid pointer This bit is used to indicate to the host controller when there are no more valid entries in the queue.

Alternate Next qTD Pointer. Offset in descriptor: DWord 1.

Table 34: siTD Alternate Next qTD Pointer Fields

Bit	Description
31:05	**Next Transfer Element Pointer** field. Physical memory address bits, 31:05, referencing the next qTD transfer descriptor to be processed if the current qTD encounters a short packet during an IN transaction.
04:01	**Reserved.**
00	**T** bit. Terminate bit indicates whether the **Next Transfer Element Pointer** value above is valid: 1 = invalid pointer 0 = valid pointer This bit is used to indicate to the host controller when there are no more valid entries in the queue.

qTD Token. Offset in descriptor: DWord 3.

Table 35: qTD Token Fields

Bit	Description
31	**Data Toggle** bit. Used in conjunction with the **Data Toggle Control** bit in the queue head structure to indicate the data toggle sequence.
30:16	**Total Bytes To Transfer.** Value indicates the total number of bytes to be moved in the processing of this transfer descriptor. At the successful conclusion of each transaction, this running total is decremented by host controller to indicate the bytes transferred.Maximum initial value is 5 pages * 4KB = 20KB.
15	**Interrupt On Complete (IOC)** bit. If set = 1, at the completion of this qTD, the host controller will generate an interrupt at the next interrupt threshold.
14:12	**Current Page (C_Page)** field. Used as an index into the qTD buffer pointer list. Range is 0-4, representing the 5 pages supported by the qTD structure.
11:10	**Error Counter (CERR)** field. Tracks the number of consecutive errors the host controller encounters in executing this qTD. The value is pre-programmed with a maximum value; if the host controller decrements the count an sees that it = 0, it does a series of things, including generating an interrupt if it is enabled in the **USB Error Interrupt Field** in the **USBINTR** register. Note: if software initializes this field = 0, then the host controller will have no limit on retries of this qTD.
09:08	**PID Code (PID)** field. This field is programmed by software to indicate the type of token which should be used for each transaction when processing this qTD: 00 = OUT token 01 = IN token 10 = SETUP token 11 = Reserved

Table 35: qTD Token Fields

Bit	Description

07:00	**Status** field**.** This field enables the host controller to convey the current state of execution to the Host Controller Driver. Bit coding is as follows:
	Bit 7 = Active bit. Set =1 by host software to enable execution of transactions by the controller. Controller clears this bit when complete.
	Bit 6 = Halted bit. Set = 1 by controller when a serious error has occurred at the device being targeted (babble, STALL, etc.). When this bit is set, the **Active** bit is cleared.
	Bit 5 = Data Buffer Error bit. Set =1 by host controller to indicate an overrun (during an IN) or underrun (during an OUT) while transferring data. For overruns, the host controller forces a time-out on USB.
	Bit 4 = Babble Detected bit. Set =1 by host controller if babble was detected during this transaction. The controller also sets the **Halted** bit = 1, and clears **Active**.
	Bit 3 = Transaction Error (XactErr) bit. Set =1 (for IN transactions) by host controller if the host did not receive one of the valid responses from the target device.
	Bit 2 = Missed Micro-Frame bit. Set = 1 by controller if it missed a required complete-split transaction because it was "held off" from accessing memory in time.
	Bit 1 = Split Transaction State (SplitXstate) bit. This bit controls when the host controller is allowed to perform a start-split or complete-split transaction:
	    • 0 = do start-split transaction to the endpoint
	    • 1 = do complete-split transaction to the endpoint.
	Bit 0 = Ping State (P)/ERR bit. This bit has two uses, depending on the nature of the targeted endpoint. If the endpoint is high-speed AND an OUT endpoint, then this bit is the Ping state bit. If these conditions are true, then:
	0 = Do OUT PID
	1 = Do Ping PID
	If the device is not high-speed, then this bit is an error bit (ERR) which will be set = 1 by the host controller in the event of an ERR handshake.

qTD Buffer Page Pointer List (DW3). Offset in descriptor: DWord 3.

Table 36: qTD Buffer Page Pointer List (DW3) Fields

Bit	Description
31:12	**Buffer Pointer List** field. These bits provide the upper bits (31:12) of a 4KB-aligned memory address (the lower 12 bits are assumed to be 0).
11:00	**Current Offset** field**.** These 12 bits represent the byte offset for the current page pointer (referenced by the *C_Page* bit in the **qTD Token Field**).

qTD Buffer Page Pointer List (DW4-7). Offset in descriptor: DWord 4-7.

Table 37: qTD Buffer Page Pointer List (DW4-7) Fields

Bit	Description
31:12	**Buffer Pointer List** field. These bits provide the upper bits (31:12) of a 4KB-aligned memory address (the lower 12 bits are assumed to be 0).
11:00	**Reserved.** Initialize = 0. field**.**

Queue Head Structure Layout

Figure 13 below depicts the 48 byte structure of an EHCI queue head.

Figure 13: EHCI Queue Head Structure
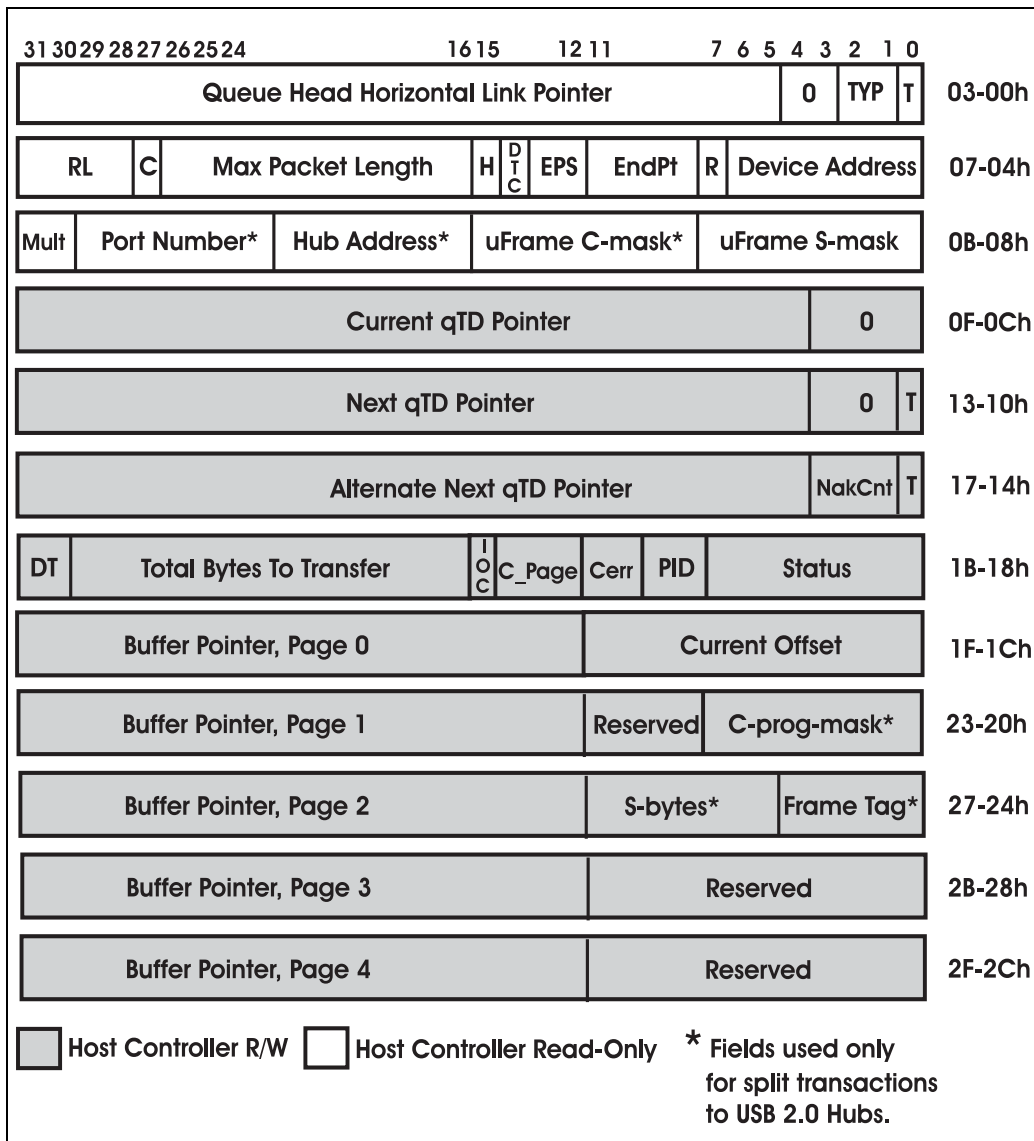
Queue Head Structure Fields

(refer to Figure 13)

Queue Head Horizontal Link Pointer. Offset in descriptor: DWord 0.

Table 38: Queue Head Structure Fields

Bit	Description
31:05	**Queue Head Horizontal Link Pointer (QHLP)** field. Physical memory address bits, 31:05, referencing the next data object to be processed
04:03	**Reserved.** Initialize = 0.
02:01	**QH/(s)iTD Select (TYP)** field. Indicates the class of item being referenced by this link pointer. This indicates the number of bytes in the structure and what type of processing will be done: 00 = iTD (**QHLP** points to an Isochronous Transfer Descriptor) 01 = QH (**QHLP** points to another Queue Head) 10 = siTD (**QHLP** points to a Split Transaction Isochronous Transfer Descriptor) 11 = Reserved.
00	**Terminate (T)** bit. This bit indicates whether the **QHLP** pointer is valid at all: 1 = invalid pointer (this is the last QH) 0 = valid pointer This bit is used to indicate to the host controller when there are no more valid entries in the queue.

Endpoint Capabilities/Characteristics. Offset in the queue head structure: DWord 1.

This DWord is the first of two used to store information about the nature of the endpoint being referenced by this queue head, including such things as address, speed, etc.

Table 39: Endpoint Characteristics (DW1) Fields

Bit	Description
31:28	**Nak Count Reload (RL)** field. The value in this field is the one used by the host controller to reload the **NakCnt** field (see DWord 5)
27	**Control Endpoint Flag** bit. This bit is set = 1 by software in the endpoint is not high-speed AND it is a control endpoint.
26:16	**Maximum Packet Length** field. The number programmed into this field is equal to the maximum packet size information returned by the device (wMaxPacketSize) The maximum value is 1024d.
15	**Head of Reclamation List Flag (H)** bit. When programmed = 1 by system software, this queue head is marked as the head of the reclamation list.
14	**Data Toggle Control (DTC)** bit. Indicates where the host controller should get initial data toggle: 0 = host controller ignores **DT** bit of incoming qTD, and preserves **DT** bit in queue head instead. 1 = Initial data toggle is supplied in qDT **DT** bit, and the host controller replaces **DT** bit in queue head with this one.
13:12	**Endpoint Speed (EPS)** field. This is the speed information for the endpoint associated with this queue head: 00 = Full Speed 01 = Low Speed 10 = High Speed 11 = Reserved.
11:08	**Endpoint Number (EndPt)** field. This is the 4-bit endpoint number within the device associated with this queue head.
07	**Reserved.** Initialize = 0.

Table 39: Endpoint Characteristics (DW1) Fields

Bit	Description
06:00	**Device Address** field. This field is programmed with the 6-bit address assigned to the device containing the endpoint associated with this queue head.

Endpoint Capabilities/Characteristics. Offset in the queue head structure: DWord 2. This DWord is the second of two used to store information about the nature of the endpoint being referenced by this queue head.

Table 40: Endpoint Capabilities (DW2) Fields

Bit	Description
31:30	**High Bandwidth Pipe Multiplier (Mult)** field. The value in this field is used to inform the host controller as to the number of successive packets (in one micro-frame) it may send to one endpoint during the current execution: 00 = Reserved. 01 = Only one transaction to this endpoint per micro-frame is allowed. 10 = Two transactions to this endpoint per micro-frame are allowed 11 = Three transactions to this endpoint per micro-frame are allowed
29:23	**Port Number** field. These bits are relevant only if the **EPS** (speed) field in the previous DWord indicates this is a Full Speed or Low Speed device. In that case, the number in this field (and the one immediately below) are used to store the port number and hub address to which this device is attached. This information is required to send split transactions.
22:16	**Hub Address** field. These bits are relevant only if the **EPS** (speed) field in the previous DWord indicates this is a Full Speed or Low Speed device. In that case, the number in this field (and the one immediately above) are used to store the port number and hub address to which this device is attached. This information is required to send split transactions.
15:08	**Split Completion Mask (uFrame C-Mask)** field. This bit mask is relevant only if the **EPS** (speed) field in the previous DWord indicates this is a Full Speed or Low Speed device. If it is a LS or FS device AND if this queue head is in the Periodic List, then this field is compared against the lower 3 bits of FRINDEX count to determine when this queue head is a candidate for execution.
07:00	**Interrupt Schedule Mask (uFrame S-mask)** bit. A value other than 0 in this register indicates that this is an interrupt endpoint. In that case, the mask contained in these bits is compared with the lower three bits of FRINDEX count by the host controller to determine if this queue head is a candidate for execution. This field should always be set = 0 if this queue head is in the asynchronous schedule.

Current qTD Link Pointer. Offset in the queue head structure: DWord 3.

Table 41: Current qTD Link Pointer Fields

Bit	Description
31:05	**Current Element Transaction Descriptor Link Pointer** field. Physical memory address bits, 31:05, referencing the current transaction being processed in the queue
04:00	**Reserved.**

Transfer Overlay DWords. Offset in the queue head structure: DWord 4-11.

The last nine DWords in the queue head structure are used as a "working space" by the host controller to track incremental progress of an active transfer.When the transaction finally completes, key parts of the results are "written back" to the source qTD being processed. When the next qTD is processed, it is merged into this space. Refer to the EHCI Specification for the use of the overlay area.