Universal Host Controller Interface (UHCI) Design Guide

REVISION 1.1

March 1996, 297650-002

Universal Host Controller Agreement

This is a royalty-free, reciprocal license for Adopters of the USB Specification who wish to implement the Universal Host Controller interface designed by Intel, and described in the Universal Host Controller Interface (UHCI) Design Guide, in their USB-compliant products. By implementing this specification, **you ("User") are agreeing to be bound by the terms of this agreement**. If you do not agree to them, then you have no license to use the specification, and you should destroy these materials or return them to Intel.

Eligible Licensees: The licenses granted in this Agreement shall only extend to a party who has executed and is bound by the agreement executed by Intel and other parties entitled "USB Reciprocal Covenant" relating to the USB Specification.

Agreement: Effective as of User's acceptance of this Agreement, and subject to its terms and conditions, Intel Corporation ("Intel") and User agree as follows:

License: Intel and User each grant to the other and its parents and subsidiaries, under any claim of a patent or patent application otherwise infringed, a non-exclusive, royalty-free, non-transferable, world-wide license, without rights to sublicense, to make or have made such party's products which implement the Interface solely in connection with implementing the USB Specification, and to use, sell, offer to sell, and import such products, where infringement of such claims would not have occurred but for the implementation and incorporation of the Interface in such products, and there is no feasible alternative to such infringement.

"Interface" means interface specification described in the document entitled "Universal Host Controller Interface (UHCI) Design Guide" published by Intel, and any circuitry described therein.

"USB Specification" means a revision of the "*Universal Serial Bus Specification*," numbered 1.0 or greater, published and made available for industry licensing by Intel and the other USB promoters.

No Other Licenses. Except for the rights expressly provided by this Agreement, neither party grants or receives, by implication, or estoppel, or otherwise, any rights under any patents or other intellectual property rights. Licenses to the USB Specification are to be granted by a separate document.

LIMITATION OF LIABILITY: The Interface is provided "AS IS" without warranty of any kind. INTEL OFFERS NO OTHER WARRANTY EITHER EXPRESS OR IMPLIED INCLUDING THOSE OF MERCHANTABILITY, NONINFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY OR FITNESS FOR A PARTICULAR PURPOSE. NEITHER INTEL NOR ITS SUPPLIERS SHALL BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE INTERFACE, EVEN IF INTEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

TERMINATION OF THIS LICENSE: Intel may terminate this license at any time if you are in breach of any of its terms and conditions. Upon termination, you will immediately destroy the Interface or return all copies of the Interface to Intel along with any copies you have made.

U.S. GOVERNMENT RESTRICTED RIGHTS: The Interface is provided with "RESTRICTED RIGHTS." Use, duplication or disclosure by the Government is subject to restrictions set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor. Use of the Interface by the Government constitutes acknowledgment of Intel's rights in them .

APPLICABLE LAWS: Any claim arising under or relating to this Agreement shall be governed by the laws of Delaware. You may not export the Interface in violation of applicable export laws.

Table Of Contents

REVISION HISTORY

Date of Revision	Version	Description
November, 1995	1.0	Original Version
March 21, 1996	1.1	Additional USB 0.99 to 1.0 specification changes, clarifications, and Legacy Keyboard and Mouse support .

Universal Host Controller Interface (UHCI)

This document describes a Universal Host Controller Interface (UHCI) for a device that implements a Universal Serial Bus (USB) Host Controller. The document is intended for hardware vendors. The UHCI description covers the hardware/software interface between the Host Controller Software Driver and the Host Controller hardware (shaded area in Figure 1). It describes the register-level hardware interface to the USB Host Controller. Hardware developers may take advantage of the standard software drivers written to be compatible with this Universal Host Controller Interface by conforming to the register level interface and memory data structures described in this document.

The UHCI consists of two parts—Host Controller Driver (HCD) and Host Controller (HC). The HCD software is responsible for scheduling the traffic on USB by posting and maintaining transactions in system memory. HCD is part of the system software and is typically provided by the operating system vendor. For example, Microsoft Corporation will fully support UHCI.

The Host Controller moves data between system memory and devices on the USB by processing these data structures and generating the transaction on USB. Note that the high transfer rates on USB means that the Host Controller should use a high bandwidth interface to system memory. For the implementation example in this document, the Host Controller is a PCI device. PCI Bus master capability in the Host Controller permits high performance data transfers to system memory.

Some of the key features of UHCI include:

- **Ease Of Use.** The reduced hardware complexity makes the Host Controller easy to implement. In addition, standard UHCI software drivers (HCD) will be supported by companies like Microsoft Corporation. Other operating system vendors can make use of the portability of the HCD software. Note that the run-time portion of the HCD software is operating system independent. The initialization code is easily tailored for a particular operating system.
- **Minimize Cost.** An optimal partition between software and hardware operations was selected to minimize Host Controller gate count, maximize performance, and maintain flexibility.
- **Flexibility.** The UHCI Host Controller can be implemented in a wide range of products from a stand-alone device to integration into a PCI chipset.
- **Reduced Hardware Complexity.** The Host Controller can be implemented with about 10,000 gates. The low gate count and few number of pins needed makes this function easily integrated into a chipset. The UHCI software/hardware interface was designed to minimize Host Controller hardware complexity.
  - Data structures are manipulated by the Host Controller in a very simple fashion. All pointers are set by software and the only operation by the hardware is a copy of the link pointers. No numerical operations are required on data structure pointers.
  - The basic data structure (transfer descriptor) for both isochronous and queued transfers use the same form (fields in the structure determine transfer type behaviors). This simplifies decode logic.
  - The Host Controller transfers the appropriate data over USB by executing a schedule list of transactions set up in system memory by the HCD software. In UHCI, the Host Controller execution of this schedule is inherently tied directly to the real-time nature of isochronous transfers. The frame counter in the Host Controller that is needed to provide a frame count number for the Start Of Frame (SOF) packet is also used to index into the schedule list. No address calculations are necessary. The Host Controller simply walks the schedule list one entry at a time as it generates the next 1 ms frame.
  - The UHCI register set can be implemented with eight hardware registers, including two root hub port registers. To prevent conflicts with other I/O devices, the USB I/O register set can be re-located in the I/O address space by programming one of the Host Controller's PCI configuration registers.
- **Increased System Performance.** UHCI maximizes the use of the USB bus and minimizes the impact on other system buses.

- UHCI minimizes end of frame idle time on USB. Potentially idle time in a 1-ms USB frame can be reclaimed to within a 32- or 64-byte packet.
- Data structures and their manipulation by the Host Controller were designed to minimize bus traffic to system memory. For example, during isochronous transfers, the Host Controller only needs to fetch one data structure (transfer descriptor) from system memory to generate a transaction.

- **Legacy Keyboard and Mouse Support.** While not required by UHCI, Section 5 documents a method to provide support for legacy software which directly accesses an 8042 device-based keyboard and mouse controller.

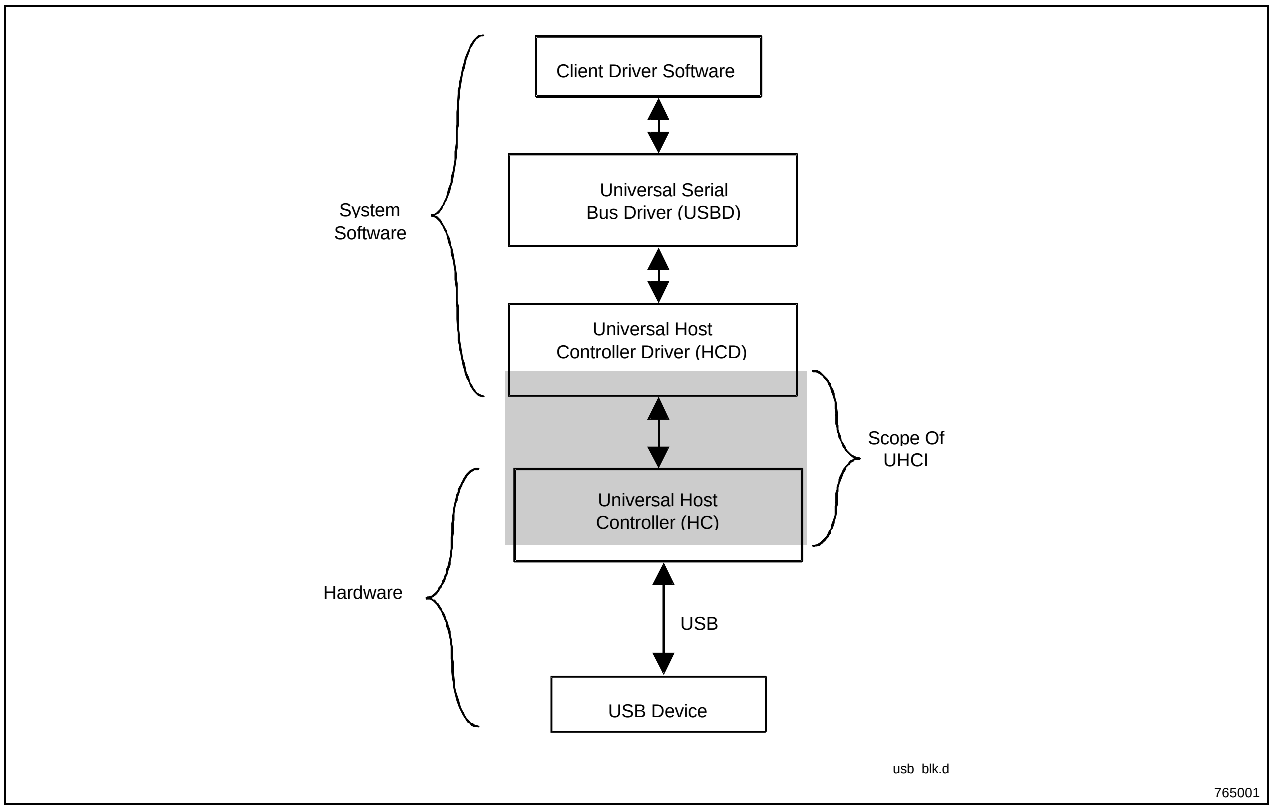

Client Driver Software
Universal Serial Bus Driver (USBD)
System Software
Universal Host Controller Driver (HCD)
Scope Of UHCI
Universal Host Controller (HC)
Hardware
USB
USB Device
usb blk.d
765001

Figure 1. Universal Serial Bus System Block Diagram

1. OVERVIEW

A USB Host System is composed of a number of hardware and software layers. Figure 1 shows a conceptual representation of these layers.

- **USB Driver (USBD).** The USBD system software that supports USB in a particular operating system.
- **Client driver software.** This software executes on the host PC corresponding to a particular USB device. Client software is typically part of the operating system or provided with the USB device.
- **Host Controller Driver (HCD).** HCD provides the software layer between the Host Controller hardware and the USBD. The UHCI's HCD interprets requests from the USBD and builds Frame List, Transfer Descriptor, Queue Head, and data buffer data structures for the Host Controller. The data structures are built in system memory and contain all necessary information to provide end-to-end communication between client software in the host and devices on the USB.
- **Host Controller (HC).** The Host Controller is managed by the HCD software layer. The UHCI's Host Controller executes the schedule lists generated by HCD and reports the status of transactions on the USB to HCD.

Command execution includes generating serial bus token and/or data packets based on the command and initiating transmission on USB. For commands that require the Host Controller to receive data from the USB device, the Host Controller receives the data and then transfers it to the system memory pointed to by the command. The UHCI's HCD provides sufficient commands and data to keep ahead of the Host Controller execution and analyzes the results as the commands are completed.

- **USB Device.** This is a hardware device that performs a useful end-user function. Interactions with USB devices flow from the applications through the software and hardware layers to the USB devices.

1.1 Data Transfer Types

USB transfers data associated with client software on the host to an endpoint on a USB device. A particular device may have multiple endpoints. USB defines four transfer types:

- **Isochronous.** The Isochronous type is characterized by a constant, fixed rate transfer between the USB device and the host. This service is needed for devices that continuously consume or produce data at a fixed rate. Some examples of this class of device would be audio CODECs (microphones or speaker systems) and telephony devices. Isochronous service does not guarantee data delivery. Failed transactions are not retried. The Isochronous data structure elements are always retired after execution, regardless of the outcome of the transaction. See Data Structure section for a definition of Transfer Descriptors (Active Bit).
- **Interrupt.** Small, spontaneous data transfers from a device. The Interrupt transfer type supports devices that require a predictable service interval but do not necessarily produce a predictable flow of data. Interrupt transfers are input only and typically used for devices such as keyboards and pointing devices that may not produce data for long periods of time but require quick response when they do have data to send.
- **Control.** Transfers that convey device control, status, and configuration information. The Control transfer type is used to provide a control channel from the Host to USB devices. Control transfers always consist of a setup phase and zero or more data phases followed by a status phase. It is critical that control transfers to a given endpoint are handled in a FIFO manner. If Control transfers to the same endpoint are interleaved, unpredictable behavior could result.
- **Bulk.** The purpose of Bulk traffic is to provide a guaranteed transmission of data between client and host under relaxed latency requirements. This transfer type is typically used by devices that need to move large amounts of data but are able to tolerate relatively large service latencies (e.g., printers).

To provide a guaranteed delivery mechanism, Interrupt, Control, and Bulk Transfer Descriptors are retried if they do not complete successfully. The number of times that a particular Transfer Descriptor is retried is bounded by the cumulative error counter contained in the Transfer Descriptor. If the permitted error count is exhausted, the Host Controller deactivates the Transfer Descriptor by clearing its ACTIVE bit. Software must provide any error recovery mechanism required to deal with the problem.

On the USB, 1-ms frame times are used to transfer data. The Host Controller begins each frame by generating a Start Of Frame (SOF). In UHCI, if there is isochronous data to be transferred, the Host Controller Driver schedules this data first. The Host Controller Driver ensures that there is enough time to complete all scheduled isochronous and interrupt transfers with some time remaining for control and bulk transfers (see Scheduling Section).

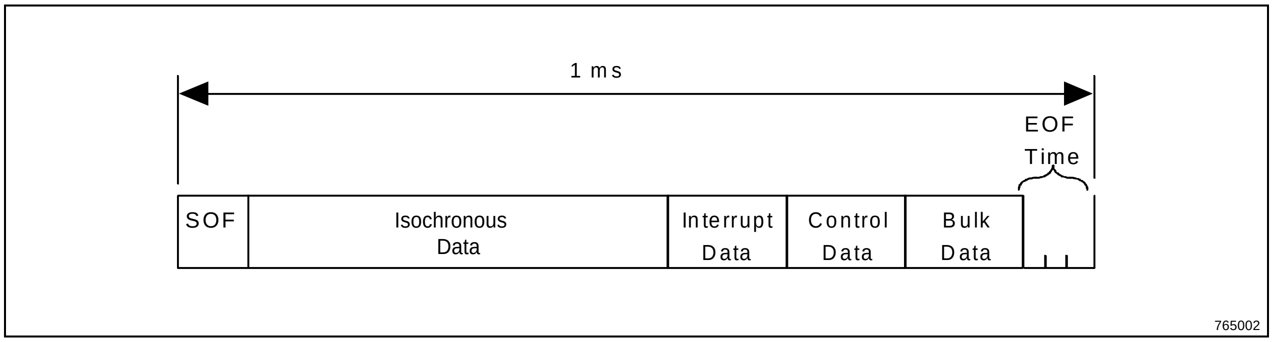


Figure 2. Transfer Type Schedule Order

1.1.1 FRAME TIME FOR DATA TRANSFERS

The Host Controller supports real-time data delivery by generating a Start Of Frame (SOF) packet every 1 ms. The SOF packet is generated when the SOF Counter in the Host Controller (Figure 3) expires. The Host Controller initializes the SOF Counter for a 1-ms frame time. Minor adjustments can be made to this value (and thus, the frame time period) by programming the SOF Modify Register. This feature permits minor changes to be made to the frame time period, if necessary, to maintain real-time synchronization throughout a USB system.

The Host Controller includes a frame number in each SOF packet. This frame number uniquely identifies the frame period in real time. The End of Frame (EOF) condition occurs at the end of the 1-ms time interval at which time the Host Controller begins the next frame time by generating another SOF packet with its corresponding frame number. Inside a frame period, data is transferred as packets of information. The frame time period is strictly enforced by the Host Controller and data packets in the current frame are not allowed to extend beyond the EOF (see Chapter 11 in the Universal Serial Bus Specification).

The Host Controller maintains real-time frame-to-data transfer synchronization by tying the frame number to the execution of a particular entry in the Frame List. The Host Controller's Frame Counter generates the frame number (11-bit value) and includes it in each SOF packet. The counter is programmable via the Frame Number Register and increments every frame period.

The Host Controller uses the lower 10 bits of the frame number as an index into a 1024 entry Frame List that is stored in system memory. Thus, since the frame count drives the Frame List entry selection, the Host Controller processes each Frame List entry in a prescribed frame period. The Host Controller increments to the next Frame List entry for each new frame. This ensures that isochronous transfers are executed in a particular frame.

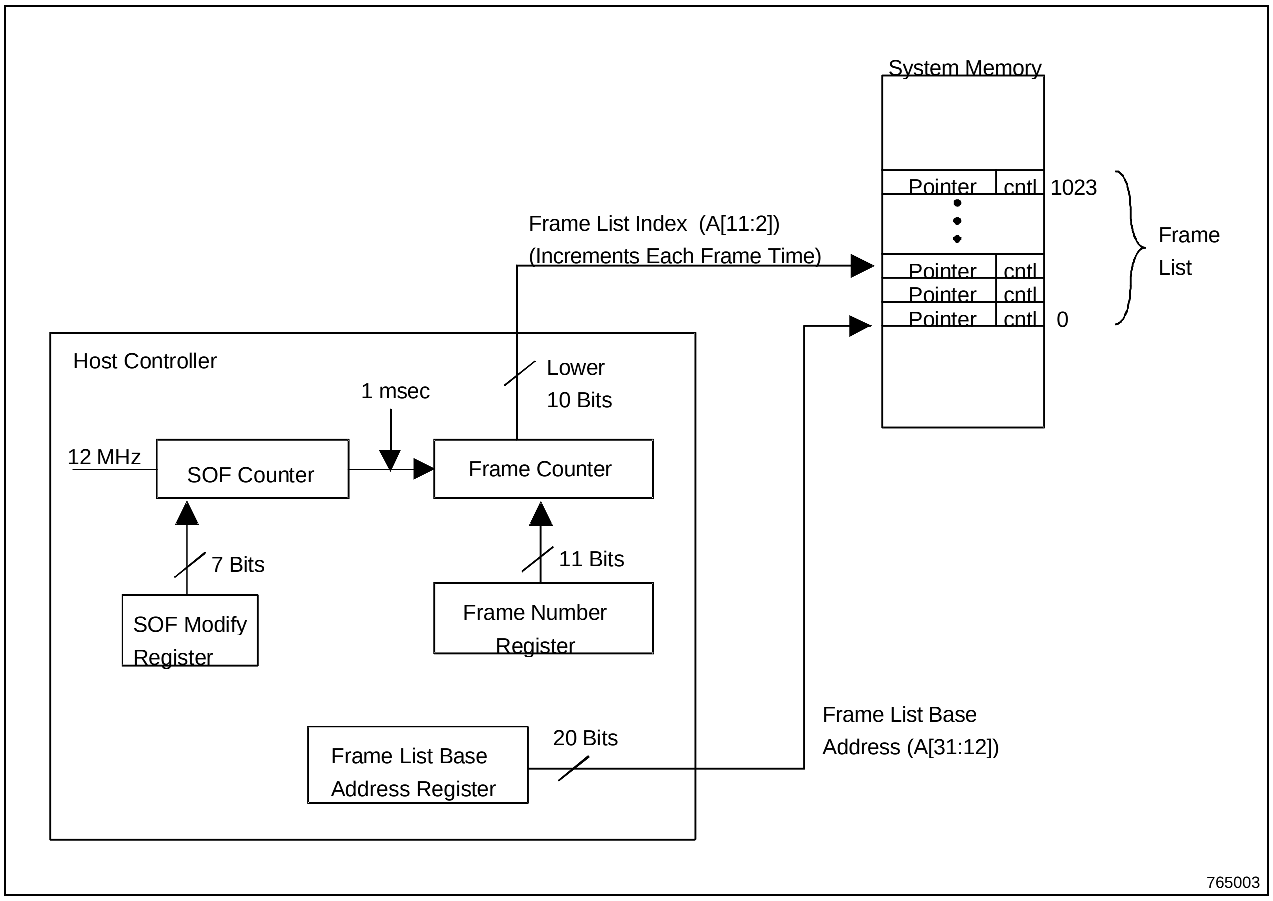


Figure 3. Frame Number Ties Frame List to Real Time

1.2 UHCI Data Structures

The UHCI data structures include a Frame List, Isochronous Transfer Descriptors, Queue Heads, and queued Transfer Descriptors. These data structures are used by HCD software to construct a “schedule” in host memory for the Host Controller to execute (Figure 4). The Host Controller is programmed with the starting address of the Frame List, then released to “execute” the schedule, without any overt synchronization with HCD. Transfer descriptors point to data buffers and include information about the addressing, data, and general behavior characteristics of the transaction.

Flow through the schedule is based on link pointers in the Frame List, Transfer Descriptors, and Queue Heads. Link Pointers are the fundamental component used to connect all the scheduled data “objects” together. The Host Controller uses the link pointer to determine where to find the next transfer descriptor to execute. Addresses in the link pointer fields must be a physical address and not a virtual address.

At the start of a frame, the Host Controller repeatedly follows link pointers, beginning at the current Frame List index, pausing its traversal to perform transactions described in Transfer Descriptors, and stopping when the frame expires (or a terminate bit is set on a horizontal flow execution).

1.2.1 FRAME LIST

The Frame List is an array of up to 1024 entries that represent a window in time. Each entry corresponds to a particular frame (1 ms). An entry serves as a reference to the transactions the Host Controller should conduct during that frame. Each entry contains a pointer to other data structures (Transfer Descriptors or Queue Heads) and control bits. The Host Controller does not update fields in the Frame List. The fields in the Frame List entries are managed by HCD software.

The Host Controller accesses the Frame List using the contents of the Frame List Base Address Register and the Frame Counter (Figure 3). The Frame List Base Address Register provides the base location of the Frame List Table in system memory and the Frame Counter provides the index into the list. The 4-Kbyte Frame List Table is aligned on a 4-Kbyte boundary.

The Host Controller does not execute beyond the 1 ms allocated to the current frame. HCD insures that any manipulation of entries be accomplished in a way that does not cause a coherency problem if the Host Controller needs to access the same entry.

1.2.2 TRANSFER DESCRIPTORS

Transfer Descriptors (TDs) contain a pointer to a data buffer and contain control and status fields for the data and its transmission or reception. Note that a TD could optionally have no data buffer associated with it (i.e., NULL data). There are TDs for isochronous transfers and queued transfers (interrupt TDs, control TDs, and bulk TDs). All TDs have the same structure. During execution, the Host Controller may update fields in the TD, as appropriate. It must maintain the TD in a consistent state (i.e., not allow accesses to partially modified TD).

For isochronous operations, the UHCI HCD software builds separate TDs for each transfer and links them into the schedule in the correct frame as requested by USBD. All temporal ordering is the responsibility of software. The Host Controller fetches the TD and generates the proper transaction on USB. The execution flow is always horizontal as shown in Figure 4. For isochronous transfers, TDs are linked to a specific point in real time. TDs that do not complete successfully on time are not retried. In the example in Figure 4, the TD list is horizontally executed until the end of the TD list is reached. The last TD in the list points to a Queue Head for processing of queued Transfer Descriptors.

For non-isochronous operations see Section 1.2.3—Queue Heads.

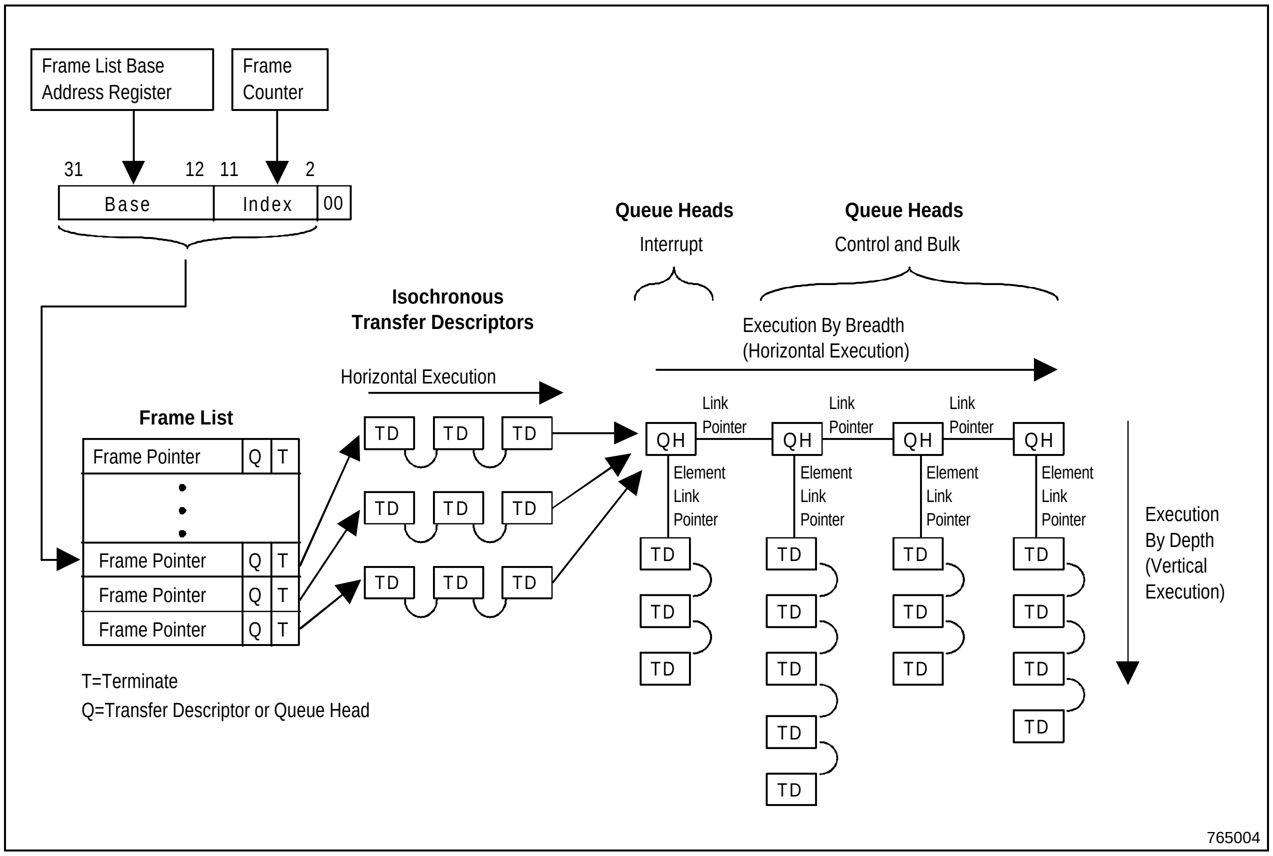

Frame List Base Address Register
Frame Counter
31
12
11
2
Base
Index
00
Isochronous Transfer Descriptors
Horizontal Execution
Queue Heads
Interrupt
Queue Heads
Control and Bulk
Execution By Breadth (Horizontal Execution)
Frame List
Frame Pointer
Q
T
TD
QH
Link Pointer
Element Link Pointer
Execution By Depth (Vertical Execution)
T=Terminate
Q=Transfer Descriptor or Queue Head
765004

Figure 4. Example Schedule

1.2.3 QUEUE HEADS

Queue Heads (QH) are data structures that organize non-isochronous transfer descriptors into queues. A QH and associated TD list form a "*Q Context*". Interrupt, Control, and Bulk, data transfer types can be placed in queues. The UHCI first services Interrupt queues followed by control queues and if there is time remaining, bulk queues. During execution, the Host Controller updates fields in the TDs and QHs, as appropriate.

Queues can be accessed directly from a Frame List entry or from the last TD in an isochronous TD list. Queues can also be accessed from a prior Queue Context. QHs contain two link pointers—a vertical pointer that selects the next TD in the Q Context to be processed and a horizontal pointer that provides a link to the next QH or TD to be processed. Note that the vertical pointer could also point to another QH. See the "Transfer Queuing" section for additional information on Queues. Queued transfers that do not complete successfully can be retried.

When queues are being processed, execution flow can be from one TD to the next within a Q Context (execution by depth) or from one QH to the next QH (execution by breadth) where only one TD in each Q Context is executed at a time. A field in the executing TD (Vf field) determines whether the execution is by depth or breadth.

For execution by depth, the execution flow traverses vertically through the TDs of the same Q Context until the end is reached or TD execution is blocked; in which case, the execution flow moves to the next QH in the link. For execution by breadth, one TD is executed in a Q Context and then the flow is to the next QH where the first TD in that Q Context is executed, and so on. See Section 3.4 for a more complete description.

1.3 Scheduling

The HCD software sets up and manages the data structures to ensure that Isochronous traffic has the highest priority in the Host Controller. The HCD scheduling allows up to 90% of the frame bandwidth to be allocated to isochronous and interrupt traffic, and up to 10% of the frame bandwidth for control. Any remaining bandwidth can be reclaimed for control and bulk transfers.

Scheduling with the UHCI is handled by a Frame List (up to 1024 entries). Each entry is a pointer to the first structure to process in a given frame. Because these pointers are a full 32 bits long, a 1024 entry Frame List occupies 4096 bytes of memory (one page).

Control and bulk transfers are scheduled last to allow bandwidth reclamation on a lightly loaded USB. Bandwidth reclamation allows the hardware to continue executing a schedule until time runs out in the frame, cycling through queue entries as frame time allows. Control is scheduled first to prioritize it over bulk transfers. Also, the software does the scheduling to guarantee that at least 10% of the bandwidth is available for control transfers. UHCI only allows for bandwidth reclamation of full speed control and bulk transfers. The software must schedule low speed control transfers such that they are guaranteed to complete within the current frame. Low speed bulk transfers are not allowed by the USB specification. If full speed control or bulk transfers are in the schedule, the last QH points back to the beginning of the full speed control and bulk queues to allow bandwidth reclamation. As long as time remains in the frame, the full speed control and bulk queues continue to be processed. If bandwidth reclamation is not required, the last QH contains a terminate bit to inform the Host Controller to wait until the beginning of the next frame.

1.3.1 HARDWARE CONTROL FOR FULL SPEED TRANSFER BANDWIDTH RECLAMATION

For full speed control and bulk bandwidth reclamation, the Host Controller hardware uses a preSOF time point (Figure 5) to determine if there is enough time to execute the selected size transaction (32- or 64-byte packet size as programmed in the USB Command Register). The preSOF point permits the Host Controller to maximize bandwidth efficiency by reclaiming potential idle time on the bus to within a 32- or 64-byte full speed packet. Note that any packets which may be subjected to bandwidth reclamation must not be any larger in size than the value (32- or 64- byte) programmed into bit 7 of the USB Command register. Note that for any packets larger than the programmed size (32- or 64-byte), the scheduling software must guarantee that bandwidth is available for its completion within the frame.

The preSOF point also prevents a packet that may not fit in the remaining frame period from being initiated. If the preSOF point has already occurred as the Host Controller is transitioning out of the status update phase of the command, the Host Controller does not fetch the next command in this frame (Figure 5, Case A). If the preSOF point is reached during the fetch of the next command's Transfer Descriptor (TD) or during the initial write data fetch (up to 32 bytes), the command is aborted (Figure 5, Case B). No data is written back to memory. If the Host Controller starts the transaction on the bus before preSOF, the command runs to completion, but is the last TD executed in that frame's list (Figure 5, Case C).

1.4 Root Hub/Ports

The Host Controller is required by the USB Specification to implement the root hub. Hubs provide the electrical interface between USB devices and the host and provide data control for transfers. Hubs can also optionally provide power management capabilities. For the Host Controller implementation described in this design guide, the root hub is integrated into the Host Controller and has two ports (Figure 6). UHCI permits additional ports to be implemented up to the maximum specified by the USB Specification.

The flow of information towards the host is referred to as upstream traffic and the flow of information away is referred to as downstream traffic. For details on hub operations, refer to Chapter 11 of the USB Specification. The UHCI root hub downstream port characteristics and operation conform to the specifications described in chapter 11.

For UHCI implementation described in this design guide, the root hub provides data flow control. Power management control for individual ports is beyond the scope of this UHCI Design Guide.

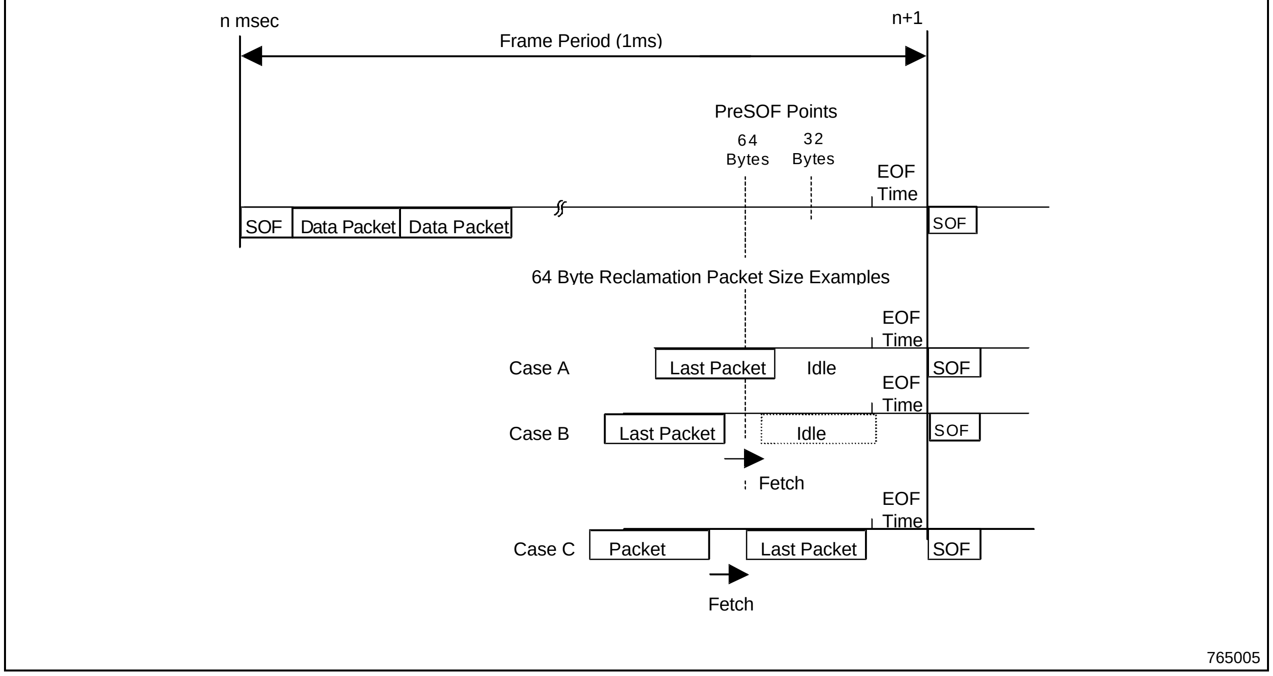


Figure 5. Frame Period

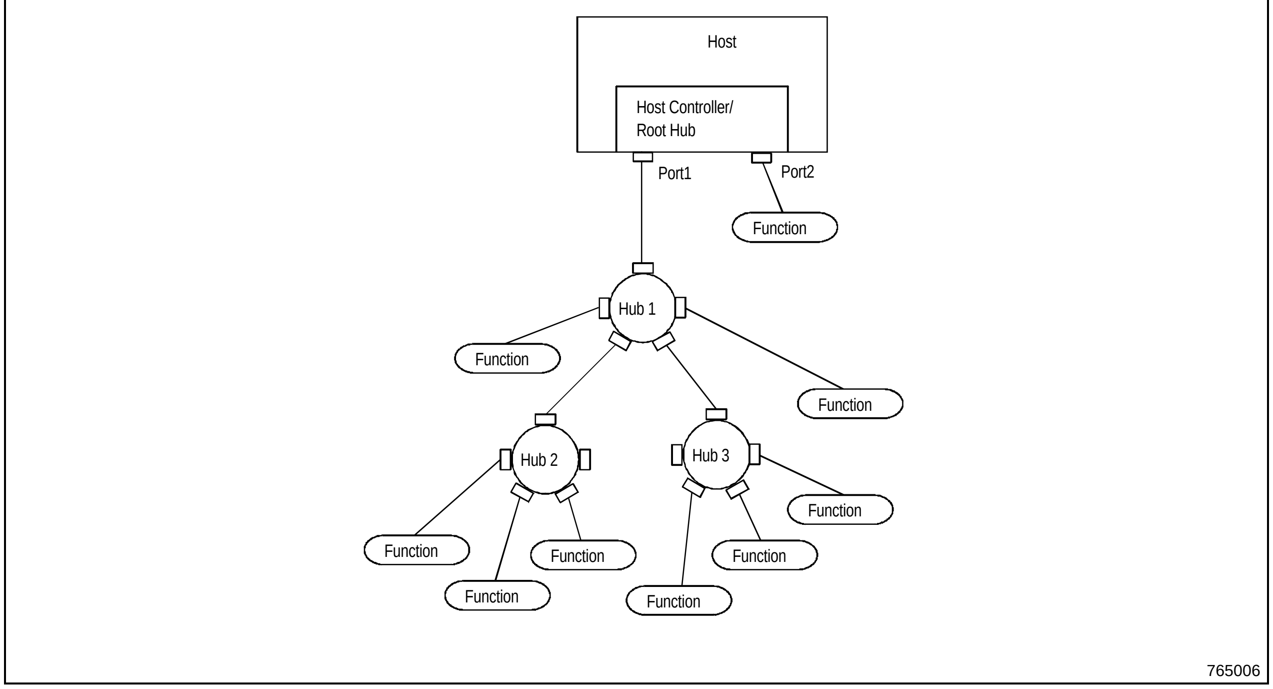


Figure 6. Hub Topology

Chapter 11 of the USB specification describes software commands used for controlling and obtaining status of hubs and ports. The UHCI's HCD abstracts these commands and uses them to set or read the appropriate bits in the Host Controller registers. These commands are shown in Table 1.

Table 1. Host Controller Register bit affected By Port Commands

Command	Register Name	Register Bits	Comment
Hub Descriptors	None	None	Implemented in HCD Software. Does not affect hardware.
ClearHubFeature	None	None	Not part of UHCI Design Guide
ClearPortFeature	Port Status and Control	1, 2, 3, 8, 9, 12	Register Read or Write.
GetBusState	Port Status and Control	4, 5	Register Read
GetHubDescriptor	None	None	Implemented in HCD Software. Does not affect hardware.
GetHubStatus	None	None	Implemented in HCD Software. Does not affect hardware.
GetPortStatus	Port Status And Control	0, 1, 2, 3, 8, 9, 12	Register Read. Reports Current Connect Status and current Connect Status Change; Port Enable/Disable and Enable/Disable Change; Low Speed Device Attached.
SetHubDescriptor	None	None	Implemented in HCD Software. Does not affect hardware.
SetHubFeature	None	None	Not part of UHCI Design Guide
SetPortFeature	Port Status and Control	0, 1, 2, 3, 8, 9, 12	Register Write

2. REGISTER DESCRIPTION

I/O registers are required for specific communications between the CPU and the Host Controller that are not efficiently handled via main system memory. The USB Host Controller module contains two sets of software accessible registers—I/O registers and optional PCI configuration registers. Note that the PCI configuration registers are only needed for PCI devices that implement the Host Controller.

1. **USB Host Controller I/O Registers.** This block of Control and Status registers is I/O mapped into PCI I/O space and controls the various operations of the USB (Table 2). The Base portion of the address location is set via a PCI configuration register.

2. **PCI Configuration Registers (For PCI devices).** In addition to the normal PCI header and device-specific registers, two registers are needed in the PCI Configuration space to support USB (Table 3). The normal PCI header and device specific registers are beyond the scope of this document (The CLASSC register is shown in this document). Note that HCD does not interact with the PCI configuration space. This space is used only by the PCI enumerator to identify the USB Host Controller, and assign the appropriate system resources.

The following notation is used to describe register access attributes:

RO	**Read Only**. If a register is read only, writes have no effect.
WO	**Write Only**. If a register is write only, reads have no effect.
R/W	**Read/Write**. A register with this attribute can be read and written. Note that individual bits in some read/write registers may be read only.
R/WC	**Read/Write Clear**. A register bit with this attribute can be read and written. However, a write of a 1 clears (sets to 0) the corresponding bit and a write of a 0 has no effect.

Table 2. USB Host/Controller I/O Registers

I/O Address	Mnemonic	Register Description	Register Access
Base + (00–01h)	USBCMD	USB Command	R/W
Base + (02–03h)	USBSTS	USB Status	R/WC
Base + (04–05h)	USBINTR	USB Interrupt Enable	R/W
Base + (06–07h)	FRNUM	Frame Number	R/W**
Base + (08–0Bh)	FRBASEADD	Frame List Base Address	R/W
Base + 0Ch	SOFMOD	Start Of Frame Modify	R/W
Base + (10–11h)	PORTSC1	Port 1 Status/Control	R/WC**
Base + (12–13h)	PORTSC2	Port 2 Status/Control	R/WC**

** NOTE: These registers are WORD writeable only. Byte writes to these registers have unpredictable effects.

Table 3. PCI Configuration Registers For USB (PCI Devices Only)

Configuration Offset	Mnemonic	Register	Register Access
00–08h	—	Register implementation as needed for specific PCI device	—
09–0Bh	CLASSC	Class Code	RO
0C–1Fh	—	Register implementation as needed for specific PCI device	—
20–23h	USBBASE	IO Space Base Address	R/W
24–5Fh	—	Register implementation as needed for specific PCI device	—
60h	SBRN	Serial Bus Release Number	RO
61–FFh	—	Register implementation as needed for specific PCI device	—

2.1 USB I/O Registers

This section describes the block of USB registers that are located in normal I/O space. The “base” portion of the I/O address is selected via a PCI Configuration register.

Some of the read/write register bits which deal with changing the state of the USB hub ports function such that on read back they reflect the current state of the port and not necessarily the state of the last write to the register. This allows the software to poll the state of the port and wait until it is in the proper state before proceeding. A Host Controller Reset, Global Reset, or Port Reset will immediately terminate a transfer on the affected ports and disable the port. This affects the USBCMD register, bit [4] and the PORTSC registers, bits [12,6,2]. See individual bit descriptions for more detail.

2.1.1 USBCMD—USB COMMAND REGISTER

I/O Address:	Base+ (00–01h)
Default Value:	0000h
Attribute:	Read/Write
Size:	16 bits

The Command Register indicates the command to be executed by the serial bus host controller. Writing to the register causes a command to be executed. The table following the bit description provides additional information on the operation of the Run/Stop and Debug bits.

Bit	Description
15:8	**Reserved.**
7	**Max Packet (MAXP).** 1=64 bytes. 0=32 bytes. This bit selects the maximum packet size that can be used for full speed bandwidth reclamation at the end of a frame. This value is used by the Host Controller to determine whether it should initiate another transaction based on the time remaining in the SOF counter. Use of reclamation packets larger than the programmed size will cause a Babble error if executed during the critical window at frame end. The Babble error results in the offending endpoint being stalled. Software is responsible for ensuring that any packet which could be executed under bandwidth reclamation be within this size limit.
6	**Configure Flag (CF).** HCD software sets this bit as the last action in its process of configuring the Host Controller. This bit has no effect on the hardware. It is provided only as a semaphore service for software.
5	**Software Debug (SWDBG).** 1=Debug mode. 0=Normal Mode. In SW Debug mode, the Host Controller clears the Run/Stop bit after the completion of each USB transaction. The next transaction is executed when software sets the Run/Stop bit back to 1. The SWDBG bit must only be manipulated when the controller is in the stopped state. This can be determined by checking the HCHalted bit in the USBSTS register.
4	**Force Global Resume (FGR).** 1=Host Controller sends the Global Resume signal on the USB. Software sets this bit to 0 after 20 ms has elapsed to stop sending the Global Resume signal. At that time all USB devices should be ready for bus activity. The Host Controller sets this bit to 1 when a resume event (connect, disconnect, or K-state) is detected while in global suspend mode. Software resets this bit to 0 to end Global Resume signaling. The 1 to 0 transition causes the port to send a low speed EOP signal. This bit will remain a 1 until the EOP has completed.
3	**Enter Global Suspend Mode (EGSM).** 1=Host Controller enters the Global Suspend mode. No USB transactions occurs during this time. The Host Controller is able to receive resume signals from USB and interrupt the system. Software resets this bit to 0 to come out of Global Suspend mode. Software writes this bit to 0 at the same time that Force Global Resume (bit 4) is written to 0 or after writing bit 4 to 0. Software must also ensure that the Run/Stop bit (bit 0) is cleared prior to setting this bit.

Bit	Description
2	**Global Reset (GRESET).** When this bit is set, the Host Controller sends the global reset signal on the USB and then resets all its logic, including the internal hub registers. The hub registers are reset to their power on state. This bit is reset by the software after a minimum of 10 ms has elapsed as specified in Chapter 7 of the USB Specification. Note: Chip Hardware Reset has the same effect as Global Reset (bit 2), except that the Host Controller does not send the Global Reset on USB.
1	**Host Controller Reset (HCRESET).** When this bit is set, the Host Controller module resets its internal timers, counters, state machines, etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. This bit is reset by the Host Controller when the reset process is complete. The HCReset effects on Hub registers are slightly different from Chip Hardware Reset and Global USB Reset. The HCReset affects bits [8,3:0] of the Port Status and Control Register (PORTSC) of each port. HCReset resets the state machines of the Host Controller including the Connect/Disconnect state machine (one for each port). When the Connect/Disconnect state machine is reset, the output that signals connect/disconnect are negated to 0, effectively signaling a disconnect, even if a device is attached to the port. This virtual disconnect causes the port to be disabled. This disconnect and disabling of the port causes bit 1 (connect status change) and bit 3 (port enable/disable change) of the PORTSC to get set. The disconnect also causes bit 8 of PORTSC to reset. About 64 bit times after HCReset goes to 0, the connect and low-speed detect will take place and bits 0 and 8 of the PORTSC will change accordingly.
0	**Run/Stop (RS).** 1=Run. 0=Stop. When set to a 1, the Host Controller proceeds with execution of the schedule. The Host Controller continues execution as long as this bit is set. When this bit is set to 0, the Host Controller completes the current transaction on the USB and then halts. The HC Halted bit in the status register indicates when the Host Controller has finished the transaction and has entered the stopped state. The Host Controller clears this bit when the following fatal errors occur: consistency check failure, PCI Bus errors.

Table 4. Run/Stop, Debug Bit Interaction

SWDBG (Bit 5)	Run/Stop (Bit 0)	Operation
0	0	If executing a command, the Host Controller completes the command and then stops. The 1.0 ms frame counter is reset and command list execution resumes from start of frame using the frame list pointer selected by the current value in the FRNUM register. (While Run/Stop=0, the FRNUM register can be reprogrammed).
0	1	Execution of the command list resumes from Start Of Frame using the frame list pointer selected by the current value in the FRNUM register. The Host Controller remains running until the Run/Stop bit is cleared (by Software or Hardware).
1	0	If executing a command, the Host Controller completes the command and then stops and the 1.0 ms frame counter is frozen at its current value. All status are preserved. The Host Controller begins execution of the command list from where it left off when the Run/Stop bit is set.
1	1	Execution of the command list resumes from where the previous execution stopped. The Run/Stop bit is set to 0 by the Host Controller when a TD is being fetched. This causes the Host Controller to stop again after the execution of the TD (single step). When the Host Controller has completed execution, the HC Halted bit in the Status Register is set.

2.1.1.1 Debug Mode/Single Step

When the USB Host Controller is in Software Debug Mode (USBCMD Register bit 5=1), the single stepping software debug operation is as follows:

To Enter Software Debug Mode:

1. HCD puts Host Controller in Stop state by setting the Run/Stop bit to 0.
2. HCD pus Host Controller in Debug Mode by setting the SWDBG bit to 1.
3. HCD sets up the correct command list and Start Of Frame value for starting point in the Frame List.Single Step Loop:
4. HCD sets Run/Stop bit to 1.
5. Host Controller executes next active TD, sets Run/Stop bit to 0, and stops.
6. HCD reads the USBCMD register to check if the single step execution is completed (HCHalted=1).
7. HCD checks results of TD execution. Go to step 4 to execute next TD or step 8 to end Software Debug mode.
8. HCD ends Software Debug mode by setting SWDBG bit to 0.
9. HCD sets up normal command list and Frame List table.
10. HCD sets Run/Stop bit to 1 to resume normal schedule execution.

In Software Debug mode, when the Run/Stop bit is set, the Host Controller starts. When a valid TD is found, the Run/Stop bit is reset. When the TD is finished, the HCHalted bit in the USBSTS register (bit 5) is set.

The SW Debug mode skips over inactive TDs and only halts after an active TD has been executed. When the last active TD in a frame has been executed, the Host Controller waits until the next SOF is sent and then fetches the first TD of the next frame before halting.

This HCHalted bit can also be used outside of Software Debug mode to indicate when the Host Controller has detected the Run/Stop bit and has completed the current transaction. Outside of the Software Debug mode, setting the Run/Stop bit to 0 always resets the SOF counter so that when the Run/Stop bit is set the Host Controller starts over again from the frame list location pointed to by the Frame List Index (see FRNUM Register description) rather than continuing where it stopped.

2.1.2 USBSTS—USB STATUS REGISTER

I/O Address:	Base + (02–03h)
Default Value:	0000h
Attribute:	Read/Write Clear
size:	16 bits

This register indicates pending interrupts and various states of the Host Controller. The status resulting from a transaction on the serial bus is not indicated in this register. Software sets a bit to 0 in this register by writing a 1 to it. See section 4, Interrupts, for additional information concerning USB interrupt conditions.

Bit	Description
15:6	**Reserved**.
5	**HCHalted.** The Host Controller sets this bit to 1 after it has stopped executing as a result of the Run/Stop bit being set to 0, either by software or by the Host Controller hardware (debug mode or an internal error).
4	**Host Controller Process Error.** The Host Controller sets this bit to 1 when it detects a fatal error and indicates that the Host Controller suffered a consistency check failure while processing a Transfer Descriptor. An example of a consistency check failure would be finding an illegal PID field while processing the packet header portion of the TD. When this error occurs, the Host Controller clears the Run/Stop bit in the Command register to prevent further schedule execution. A hardware interrupt is generated to the system.
3	**Host System Error.** The Host Controller sets this bit to 1 when a serious error occurs during a host system access involving the Host Controller module. In a PCI system, conditions that set this bit to 1 include PCI Parity error, PCI Master Abort, and PCI Target Abort. When this error occurs, the Host Controller clears the Run/Stop bit in the Command register to prevent further execution of the scheduled TDs. A hardware interrupt is generated to the system.

Bit	Description
2	**Resume Detect.** The Host Controller sets this bit to 1 when it receives a "RESUME" signal from a USB device. This is only valid if the Host Controller is in a global suspend state (bit 3 of Command register = 1).
1	**USB Error Interrupt.** The Host Controller sets this bit to 1 when completion of a USB transaction results in an error condition (e.g., error counter underflow). If the TD on which the error interrupt occurred also had its IOC bit set, both this bit and Bit 0 are set.
0	**USB Interrupt (USBINT).** The Host Controller sets this bit to 1 when the cause of an interrupt is a completion of a USB transaction whose Transfer Descriptor had its IOC bit set. The Host Controller also sets this bit to 1 when a short packet is detected (actual length field in TD is less than maximum length field in TD), and short packet detection is enabled in that TD.

2.1.3 USBINTR—USB INTERRUPT ENABLE REGISTER

I/O Address:	Base + (04–05h)
Default Value:	0000h
Attribute:	Read/Write
size:	16 bits

This register enables and disables reporting of the corresponding interrupt to the software. When a bit is set and the corresponding interrupt is active, an interrupt is generated to the host. Fatal errors (Host Controller Processor Error-bit 4, USBSTS Register) cannot be disabled by the host controller. Interrupt sources that are disabled in this register still appear in the Status Register to allow the software to poll for events.

Bit	Description
15:4	**Reserved**.
3	**Short Packet Interrupt Enable.** 1=Enabled. 0=Disabled.
2	**Interrupt On Complete (IOC) Enable.** 1= Enabled. 0=Disabled.
1	**Resume Interrupt Enable.** 1= Enabled. 0=Disabled.
0	**Timeout/CRC Interrupt Enable.** 1= Enabled. 0=Disabled.

2.1.4 FRNUM—FRAME NUMBER REGISTER

I/O Address:	Base + (06–07h)
Default Value:	0000h
Attribute:	Read/Write (Writes must be Word Writes)
Size:	16 bits

Bits [10:0] of this register contain the current frame number which is included in the frame SOF packet. This register reflects the count value of the internal frame number counter. Bits [9:0] are used to select a particular entry in the Frame List during schedule execution. This register is updated at the end of each frame time.

This register must be written as a word. Byte writes are not supported. This register cannot be written unless the Host Controller is in the STOPPED state as indicated by the HCHalted bit (USBSTS register). A write to this register while the Run/Stop bit is set (USBCMD register) is ignored.

Bit	Description
15:11	**Reserved.**
10:0	**Frame List Current Index/Frame Number.** Bits [10:0] provide the frame number in the SOF Frame. The value in this register increments at the end of each time frame (approximately every 1 ms). In addition, bits [9:0] are used for the Frame List current index and correspond to memory address signals [11:2].

2.1.5 FLBASEADD—FRAME LIST BASE ADDRESS REGISTER

I/O Address:	Base + (08–0Bh)
Default Value:	Undefined
Attribute:	Read/Write
Size:	32 bits

This 32-bit register contains the beginning address of the Frame List in the system memory. HCD loads this register prior to starting the schedule execution by the Host Controller. When written, only the upper 20 bits are used. The lower 12 bits are written as zero (4-Kbyte alignment). The contents of this register are combined with the frame number counter to enable the Host Controller to step through the Frame List in sequence. The two least significant bits are always 00. This requires DWord alignment for all list entries. This configuration supports 1024 Frame List entries.

Bit	Description
31:12	**Base Address.** These bits correspond to memory address signals [31:12], respectively.
11:0	**Reserved.** Must be written as 0s.

2.1.6 START OF FRAME (SOF) MODIFY REGISTER

I/O Address:	Base + (0Ch)
Default Value:	40h
Attribute:	Read/Write
Size:	8 bits

This 1-byte register is used to modify the value used in the generation of SOF timing on the USB. Only the 7 least significant bits are used. When a new value is written into the these 7 bits, the SOF timing of the next frame will be adjusted. This feature can be used to adjust out any offset from the clock source that generates the clock that drives the SOF counter. This register can also be used to maintain real time synchronization with the rest of the system so that all devices have the same sense of real time. Using this register, the frame length can be adjusted across the full range required by the USB specification. It's initial programmed value is system dependent based on the accuracy of hardware USB clock and is initialized by system BIOS. It may be reprogrammed by USB system software at any time. Its value will take effect from the beginning of the next frame. This register is reset upon a Host Controller Reset or Global Reset. Software must maintain a copy of its value for reprogramming if necessary.

Bit	Description
7	**Reserved.**
6:0	**SOF Timing Value.** Guidelines for the modification of frame time are contained in Chapter 7 of the USB Specification. The SOF cycle time (number of SOF counter clock periods to generate a SOF frame length) is equal to 11936 + value in this field. The default value is decimal 64 which gives a SOF cycle time of 12000. For a 12 MHz SOF counter clock input, this produces a 1 ms Frame period. The following table indicates what SOF Timing Value to program into this field for a certain frame period.

Frame Length (# 12Mhz Clocks) (decimal)	SOF Reg. Value (decimal)
11936	0
11937	1
.	.
.	.
11999	63
12000	64
12001	65
.	.
.	.
12062	126
12063	127

2.1.7 PORTSC—PORT STATUS AND CONTROL REGISTER

I/O Address:	Base + (10–11h)—Port 1
	Base + (12–13h) —Port 2
Default:	0080h
Access:	Read/Write (WORD writeable only)
Size:	16 bits

After a Power-up reset, Global reset, or Host Controller reset, the initial conditions of a port are: No device connected, Port disabled, and the bus line status is 00 (single-ended zero). Note: If a device is attached, the port state will transition to the attached state and system software will process this as with any status change notification. It make take up to 64 USB bit times for the port transition to occur. If the Host Controller is in global suspend mode, then, if any of bits [6,3,1] gets set, the Host Controller will signal a global resume. Refer to Chapter 11 of the USB Specification for details on hub operation.

Bit	Description
15:13	**Reserved.** Must written as 0s when writing this register.
12	**Suspend—R/W.** 1=Port in suspend state. 0=Port not in suspend state. This bit should not be written to a 1 if global suspend is active (bit 3=1 in the USBCMD register). Bit 2 and bit 12 of this register define the hub states as follows: **Bits [12,2]** — **Hub State** x0 — Disable 01 — Enable 11 — Suspend When in suspend state, downstream propagation of data is blocked on this port, except for single-ended 0 resets (global reset and port reset). The blocking occurs at the end of the current transaction, if a transaction was in progress when this bit was written to 1. In the suspend state, the port is sensitive to resume detection. Note that the bit status does not change until the port is suspended and that there may be a delay in suspending a port if there is a transaction currently in progress on the USB.
11:10	**Reserved.**
9	**Port Reset—R/W**. 1=Port is in Reset. 0=Port is not in Reset. When in the Reset State, the port is disabled and sends the USB Reset signaling. Note that host software must guarantee that the RESET signaling is active for the proper amount of time as specified in the USB Specification.
8	**Low Speed Device Attached—RO.** 1=Low speed device is attached to this port. 0=Full speed device. Writes have no effect.
7	**Reserved—RO.** Always read as 1.
6	**Resume Detect—R/W.** 1= Resume detected/driven on port. 0=No resume (K-state) detected/driven on port. Software sets this bit to a 1 to drive resume signaling. The Host Controller sets this bit to a 1 if a J-to-K transition is detected while the port is in the Suspend state. Note that when this bit is 1, a K-state is driven on the port as long as this bit remains 1 and the port is still in suspend state. Writing a 0 (from 1) causes the port to send a low speed EOP. This bit will remain a 1 until the EOP has completed.
5:4	**Line Status—RO.** These bits reflect the D+ (bit 4) and D- (bit 5) signals lines' logical levels. These bits are used for fault detect and recovery as well as for USB diagnostics. This field is updated at EOF2 time (See Chapter 11 of the USB Specification).
3	**Port Enable/Disable Change—R/WC.** 1=Port enabled/disabled status has changed. 0=No change. For the root hub, this bit gets set only when a port is disabled due to disconnect on the that port or due to the appropriate conditions existing at the EOF2 point (See Chapter 11 of the USB Specification). Software clears this bit by writing a 1 to it.

Bit	Description
2	**Port Enabled/Disabled—R/W.** 1=Enable. 0=Disable. Ports can be enabled by host software only. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by host software. Note that the bit status does not change until the port state actually changes and that there may be a delay in disabling or enabling a port if there is a transaction currently in progress on the USB.
1	**Connect Status Change—R/WC.** 1=Change in Current Connect Status. 0=No change. Indicates a change has occurred in the port's Current Connect Status (see bit 0). The hub device sets this bit for any changes to the port device connect status, even if system software has not cleared a connect status change. If, for example, the insertion status changes twice before system software has cleared the changed condition, hub hardware will be "setting" an already-set bit (i.e., the bit will remain set). However, the hub transfers the change bit only once when the Host Controller requests a data transfer to the Status Change endpoint. System software is responsible for determining state change history in such a case. Software sets this bit to 0 by writing a 1 to it.
0	**Current Connect Status—RO.** 1=Device is present on port. 0=No device is present. This value reflects the current state of the port, and may not correspond directly to the event that caused the Connect Status Change bit (Bit 1) to be set.

2.1.7.1 Behavior Under Global or Selective Suspend Scenarios

Tables 5 and 6 show the behavior of the Host Controller when resume, connect and disconnect signaling is received on ports in various states when the Host Controller is in or not in the global suspend state. (A full explanation of hub behavior is given in Chapter 11 of the USB spec.) Generally speaking, the PORTSC register associated with the port receiving the signaling reflects the status change appropriate for the type of signaling received. Resume signaling (K-State) is recognized in the PORTSC register only if the port is in selective suspend (PORTSC bits 2 and 12 are set). Resume will be recognized in the USBSTS register (bit 2) if resume is received on a suspended or enabled port when the Host Controller is in the global suspend state (USBCMD register bit 3 is set).

The host may also initiate a resume on a suspended port or when the Host Controller has been suspended, by writing the appropriate resume-detect/force-resume bit to a 1. A global resume is started by writing bit 4 in the USBCMD register to a 1. A K-State signal will be sent on all enabled ports. (Any port which needs to send the resume signal and is not enabled must be enabled before the resume is forced.) A resume can be forced on a selectively suspended port by writing bit 6 in the corresponding PORTSC register to a 1.

Resume signaling is ended by clearing the appropriate suspend and resume bits. This is true for either selective or global resumes or resumes initiated by signaling at the port or by the Host Controller. The suspend and resume bits must be written to a 0 together in the same write cycle or the suspend must be written to a 0 after the resume bit is reset in order to get proper single-ended zero termination of the resume signaling. Resume is ended on a suspended Host Controller by writing USBCMD register bits 3 and 4 to 0. Resume is ended on a suspended port by writing PORTSC register bits 6 and 12 to 0. If signaling on a suspended port in a globally suspended Host Controller is the source of the resume, the Host Controller suspend and resume bits should be cleared before the port bits are cleared.

Table 5. Behavior During Resume When Host Not In Global Suspend State

Port Status and Signaling Type	Signaled Port Response	Adjacent Port Response		
		Enabled Port	Disabled Port	Suspended Port
Port disabled, resume K-State received	No Effect	No Effect	No Effect	No Effect
Port suspended, Resume K-State received	Resume reflected downstream on signaled port. Resume Detect status bit in PORTSC register is set.	No Effect	No Effect	No Effect
Port enabled, disabled or suspended and disconnect received	PORTSC Connect and Enable status bits are cleared, and Connect Change and Enable/Disable Change status bits are set.	No Effect	No Effect	No Effect
Port disabled and connect received	PORTSC Connect Status and Connect Status Change bits are set.	No Effect	No Effect	No Effect

Table 6. Behavior During Resume when Host is in Global Suspend State

Port Status and Signaling Type	Signaled Port Response	Adjacent Port Response		
		Enabled Port	Disabled Port	Suspended Port
Port enabled, Resume K-State received	Resume reflected downstream on signaled port. Resume Detect Status bit in USBSTS Reg is set.	Signal resume downstream	No Effect	No Effect
Port disabled, resume K-State received	No Effect	No Effect	No Effect	No Effect
Port suspended, Resume K-State received	Resume signal reflected downstream on signaled port. Resume Detect status bit in PORTSC and USBSTS Regs are set.	Signal resume downstream	No Effect	No Effect
Port enabled, disabled or suspended and disconnect received	Resume Detect status bit in USBSTS Reg is set. PORTSC Connect and Enable status bits are cleared and Connect Change and Enable/Disable Change bits are set.	Signal resume downstream	No Effect	No Effect
Port disabled and connect received	Resume Detect status bit in USBSTS Reg is set. PORTSC Connect status bit and Connect Change status bit are set.	Signal resume downstream	No Effect	No Effect

2.2 PCI Configuration Registers (USB)

2.2.1 CLASSC—CLASS CODE REGISTER

Address Offset:	09–0Bh
Default Value:	010180h
Attribute:	Read Only
Size:	24 bits

This register contains the device programming interface information related to the Sub-Class Code and Base Class Code definition. This register also identifies the Base Class Code and the function sub-class in relation to the Base Class Code.

Bit	Description
23:16	**Base Class Code (BASEC).** 0Ch= Serial Bus controller.
15:8	**Sub-Class Code (SCC)**. 03h=Universal Serial Bus Host Controller.
7:0	**Programming Interface (PI).** 00h=No specific register level programming interface defined.

2.2.2 USBBASE—IO SPACE BASE ADDRESS REGISTER

Address Offset:	20–23h
Default Value:	00000001h
Attribute:	Read/Write
Size:	32 bits

This register contains the base address of the USB I/O Registers.

Bit	Description
31:16	**Reserved.** Hardwired to 0s. Must be written as 0s.
15:5	**Index Register Base Address.** Bits [15:5] correspond to I/O address signals AD [15:5], respectively.
4:1	**Reserved.** Read as 0.
0	**Resource Type Indicator (RTE)—RO.** This bit is hardwired to 1 indicating that the base address field in this register maps to I/O space.

2.2.3 SBRN—SERIAL BUS RELEASE NUMBER REGISTER

Address Offset:	60h
Default Value:	See Description below
Attribute:	Read only
Size:	8 bits

This register contains the release of the Universal Serial Bus Specification with which this Universal Serial Bus Host Controller module is compliant.

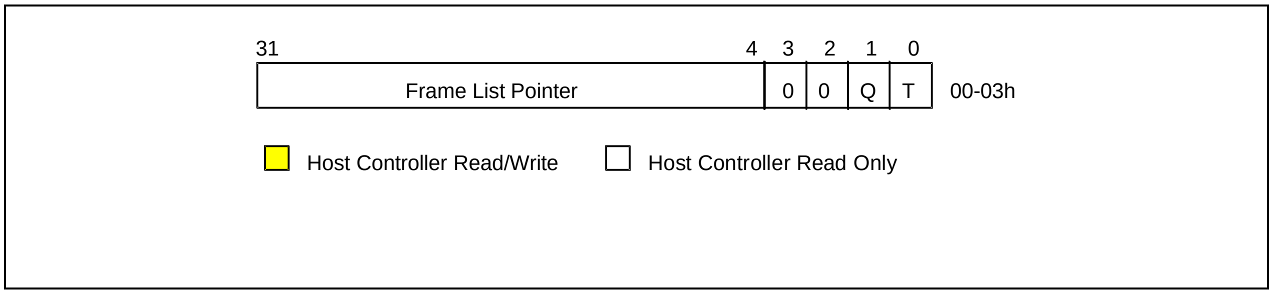
Bit	Description
7:0	**Serial Bus Specification Release Number.** All other combinations are reserved. **Bits[7:0]** **Release Number** 00h Pre-release 1.0 10h Release 1.0

3. DATA STRUCTURES

This section describes the details of the data structures used to communicate control, status, and data between HCD (software) and the Host Controller (hardware)—Frame Lists, Transfer Descriptors, and Queue Heads. Frame Lists are aligned on 4-Kbyte boundaries. Transfer Descriptors and Queue Heads must be aligned on 16-byte boundaries.

3.1 Frame List Pointer

Frame List pointers direct the host controller to the first item in the frame's schedule. The frame pointers are aligned on DWORD boundaries within the Frame List.



3.1.1 FRAME LIST POINTER (DWORD)

The frame list pointer contains a link pointer to the first data object to be processed in the frame, as well as the control bits defined below.

Bit	Description
31:4	**Frame List Pointer (FLP).** This field contains the address of the first data object to be processed in the frame and corresponds to memory address signals [31:4], respectively.
3:2	**Reserved.** These bits must be written as 0s.
1	**QH/TD Select (Q).** 1=QH. 0=TD. This bit indicates to the hardware whether the item referenced by the link pointer is a TD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched.
0	**Terminate (T).** 1=Empty Frame (pointer is invalid). 0=Pointer is valid (points to a QH or TD). This bit indicates to the Host Controller whether the schedule for this frame has valid entries in it.

3.2 Transfer Descriptor (TD)

Transfer Descriptors (TDs) express the characteristics of the transaction requested on USB by a client. TDs are always aligned on 16-byte boundaries. While there are four different transfer types supported by USB, all TDs are formatted identically. The different transfer types are supported by a small number of control bits in the descriptor that the Host Controller interprets during operation. All transfer descriptors have the same basic, 32-byte structure (Figure 7). The last 4 DWords are for software use. During operation, the Host Controller hardware performs consistency checks on some fields of the TD. If a consistency check fails, the Host Controller halts immediately and issues an interrupt to the system. This interrupt is not maskable within the host controller.

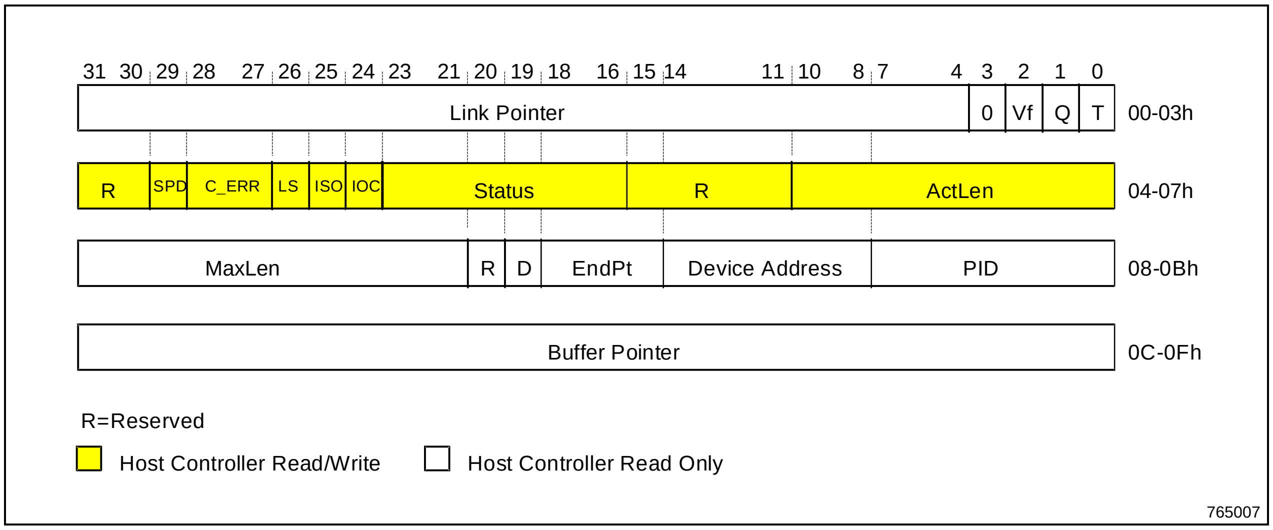


Figure 7. Generic Form of Transfer Descriptor (TD)

3.2.1 TD LINK POINTER (DWORD 0: 00-03h)

The first DWord of a Transfer Descriptor (TD) contains a link pointer to another TD or a Queue Head (QH) as well as control bits.

Bit	Description
31:4	**Link Pointer (LP).** Bits [31:4] Correspond to memory address signals [31:4], respectively. This field points to another TD or QH.
3	**Reserved.** Must be 0 when writing this field.
2	**Depth/Breadth Select (Vf).** 1=Depth first. 0=Breadth first. This bit is only valid for queued TDs and indicates to the hardware whether it should process in a depth first or breadth first fashion. When set to depth first, it informs the Host Controller to process the next transaction in the queue rather than starting a new queue.
1	**QH/TD Select (Q).** 1=QH. 0=TD. This bit informs the Host Controller whether the item referenced by the link pointer is another TD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched
0	**Terminate (T).** 1=Link Pointer field not valid. 0=Link Pointer field is valid. This bit informs the Host Controller that the link pointer in this TD does not point to another valid entry. When encountered in a queue context, this bit indicates to the Host Controller that there are no more valid entries in the queue. A TD encountered outside of a queue context with the T bit set informs the Host Controller that this is the last TD in the frame.

3.2.2 TD CONTROL AND STATUS (DWORD 1: 04-07h)

Bit	Description
31:30	**Reserved (R).**
29	**Short Packet Detect (SPD).** 1=Enable. 0=Disable. When a packet has this bit set to 1 and the packet: 1. is an input packet; 2. is in a queue; and 3 successfully completes with an actual length less than the maximum length; then the TD is marked inactive, the Queue Header is not updated and the USBINT status bit (Status Register) is set at the end of the frame. In addition, if the interrupt is enabled, the interrupt will be sent at the end of the frame. Note that any error (e.g., babble or FIFO error) prevents the short packet from being reported. The behavior is undefined when this bit is set with output packets or packets outside of queues.
28:27	This field is a 2-bit down counter that keeps track of the number of Errors detected while executing this TD. If this field is programmed with a non zero value during setup, the Host Controller decrements the count and writes it back to the TD if the transaction fails. If the counter counts from one to zero, the Host Controller marks the TD inactive, sets the “STALLED” and error status bit for the error that caused the transition to zero in the TD. An interrupt will be generated to HCD if the decrement to zero was caused by Data Buffer error, Bitstuff error, or if enabled, a CRC or Timeout error. If HCD programs this field to zero during setup, the Host Controller will not count errors for this TD and there will be no limit on the retries of this TD. Bits[28:27] — Interrupt After 00 — No Error Limit 01 — 1 error 10 — 2 errors 11 — 3 errors **Error** — **Decrement Counter** — **Error** — **Decrement Counter** CRC Error — Yes — Data Buffer Error — Yes Timeout Error — Yes — Stalled — No* NAK Received — No — Bitstuff Error — Yes Babble Detected — No* * Detection of Babble or Stall automatically deactivates the TD. Thus, count is not decremented.
26	**Low Speed Device (LS).** 1=Low Speed Device. 0=Full Speed Device. This bit indicates that the target device (USB data source or sink) is a low speed device, running at 1.5 Mb/s, instead of at full speed. There are special restrictions on schedule placement for low speed TDs. See section 1.3, Scheduling, for more information on low speed TD schedule placement. If a Host Controller root hub port is connected to a full speed device and this bit is set to a 1 for a low speed transaction, the Host Controller sends out a low speed preamble on that port before sending the PID. No preamble is sent if a Host Controller root hub port is connected to a low speed device.
25	**Isochronous Select (IOS).** 1=Isochronous Transfer Descriptor. 0=Non-isochronous Transfer Descriptor. The field specifies the type of the data structure. If this bit is set to a 1, the TD is an isochronous transfer. Isochronous TDs are always marked inactive by the hardware after execution, regardless of the results of the transaction.
24	**Interrupt on Complete (IOC).** 1=Issue IOC. This bit specifies that the Host Controller should issue an interrupt on completion of the frame in which this Transfer Descriptor is executed. Even if the Active bit in the TD is already cleared when the TD is fetched (no transaction will occur on USB), an IOC interrupt is generated at the end of the frame.

Bit	Description
23:16	**Status.** This field is used by the Host Controller to communicate individual command execution states back to HCD. This field contains the status of the last transaction performed on this TD. For Isochronous TDs, this is always the completion status (no retries). For the other transfer types this field is updated each time the TD is executed. The bit encodings are: **Bit** **Status Field Description** 23 **Active.** Set to 1 by software to enable the execution of a message transaction by the Host Controller. When the transaction associated with this descriptor is completed, the Host Controller sets this bit to 0 indicating that the descriptor should not be executed when it is next encountered in the schedule. The Active bit is also set to 0 if a stall handshake is received from the endpoint. For Host Controller schedule execution operations, see the Script and Data Transfer Primitives section. 22 **Stalled.** Set to a 1 by the Host Controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this TD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during the transaction. Any time that a transaction results in the Stalled bit being set, the Active bit is also cleared (set to 0). If a STALL handshake is received from a SETUP transaction, a Time Out Error will also be reported. 21 **Data Buffer Error.** Set to a 1 by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (underrun). When this occurs, the actual length and Max Length field of the TD will not match. In the case of an underrun, the Host Controller will transmit an incorrect CRC (thus invalidating the data at the endpoint) and leave the TD active (unless error count reached zero). If a overrun condition occurs, the Host Controller will force a timeout condition on the USB, invalidating the transaction at the source. 20 **Babble Detected.** Set to a 1 by the Host Controller during status update when a “babble” is detected during the transaction generated by this descriptor. In addition to setting this bit, the Host Controller also sets the “STALLED” bit (bit 22) to a 1. Since “babble” is considered a fatal error for that transfer, setting the “STALLED” bit to a 1 insures that no more transactions occur as a result of this descriptor. Detection of babble causes immediate termination of the current frame. No further TDs in the frame are executed. Execution resumes with the next frame list index. 19 **NAK Received.** Set to a 1 by the Host Controller during status update when the Host Controller receives a “NAK” packet during the transaction generated by this descriptor. If a NAK handshake is received from a SETUP transaction, a Time Out Error will also be reported. 18 **CRC/Time Out Error.** Set to a 1 by the Host Controller during status update in the case that no response is received from the target device/endpoint within the time specified by the protocol chapter of the USB specification. This bit is also set to a 1 by the Host Controller during status update when a CRC error is detected during the transaction associated with this transfer descriptor. In the transmit case (OUT or SETUP Command), this is in response to the Host Controller detecting a timeout from the target device/endpoint. In the receive case (IN Command), this is in response to the Host Controller’s CRC checker circuitry detecting an error on the data received from the device/endpoint or a NAK or STALL handshake being received in response to a SETUP transaction. 17 **Bitstuff Error.** This bit is set to a 1 by the Host Controller during status update to indicate that the receive data stream contained a sequence of more than 6 ones in a row. 16 **Reserved (R).**
15:11	**Reserved (R).**
10:0	**Actual Length (ActLen).** The Actual Length field is written by the Host Controller at the conclusion of a USB transaction to indicate the actual number of bytes that were transferred. It can be used by the software to maintain data integrity. The value programmed in this register is encoded as n-1 (see Maximum Length field description in the TD Token, Dword 2).

3.2.3 TD TOKEN (DWORD 2: 08-0Bh)

The third DWord of a transfer descriptor is the Packet Header. The Packet Header contains all the information required to fill in a USB Start Token. Please refer to the USB Specification for detailed definitions.

Bit	Description
31:21	**Maximum Length (MaxLen).** The Maximum Length field specifies the maximum number of data bytes allowed for the transfer. The Maximum Length value does not include protocol bytes, such as PID and CRC. The maximum data packet is 1280 bytes. The 1280 packet length is the longest packet theoretically guaranteed to fit into a frame. Actual packet maximum lengths are set by HCD according to the type and speed of the transfer. Note that the maximum length allowed by the USB specification is 1023 bytes. The valid encodings for this field are: **Bits [31:21]** **Length** 0x000 1 byte 0x001 2 bytes ..... 0x3FE 1023 bytes 0x3FF 1024 bytes ..... 0x4FF 1280 bytes 0x7FF 0 bytes (Null Data packet.) Note that values from 500h to 7FEh are illegal and cause a consistency check failure. In the transmit case, the Host Controller uses this value as a terminal count for the number of bytes it fetches from host memory. In most cases, this is the number of bytes it will actually transmit. In rare cases the Host Controller may be unable to access memory (e.g., due to excessive latency) in time to avoid underrunning the transmitter. In this instance the Host Controller would transmit fewer bytes than specified in the Maximum Length field. In the receive case, this field represents the maximum number of bytes that the device should send to the Host Controller. If the device continues to send after the Host Controller has received Max Length bytes, a BABBLE error is generated. See the Interrupt section for further information.
20	**Reserved (R).**
19	**Data Toggle (D).** This bit is used to synchronize data transfers between a USB endpoint and the host. This bit determines which data PID is sent or expected (0=DATA0 and 1=DATA1). The Data Toggle bit provides a 1-bit sequence number to check whether the previous packet completed. This bit must always be 0 for Isochronous TDs. See section 8.6 of the USB specification for a more detailed description of Data Toggle Synchronization.
18:15	**Endpoint (EndPt).** This 4-bit field extends the addressing, internal to a particular device by providing 16 endpoints. This permits more flexible addressing of devices in which more than one sub-channel is required.
14:8	**Device Address.** This field identifies the specific device serving as the data source or sink.
7:0	**Packet Identification (PID).** This field contains the Packet ID to be used for this transaction. Only the IN (69h), OUT (E1h), and SETUP (2Dh) tokens are allowed. Any other value in this field causes a consistency check failure resulting in an immediate halt of the Host Controller. Bits [3:0] are complements of bits [7:4].

3.2.4 TD BUFFER POINTER (DWORD 3: 0C-0Fh)

The fourth DWord of a Transfer Descriptor is the data buffer pointer for this transaction. It points to the beginning of the buffer that will be used during this transaction. This buffer must be at least as long as the value in the Max Length field described above. The data buffer may be byte-aligned.

Bit	Description
31:0	**Buffer Pointer.** Bits [31:0] corresponds to memory address [31:0], respectively.

3.2.5 RESERVED FOR SOFTWARE (DWORDS [7:4])

The last 4 DWords of the Transfer Descriptor are reserved for use by software.

3.3 Queue Head (QH)

Queue heads (Figure 8) are special structures used to support the requirements of Control, Bulk, and Interrupt transfers. Since these TDs are not automatically retired after each use their maintenance requirements can be reduced by putting them into a queue. Queue Heads must be aligned on a 16-byte boundary.

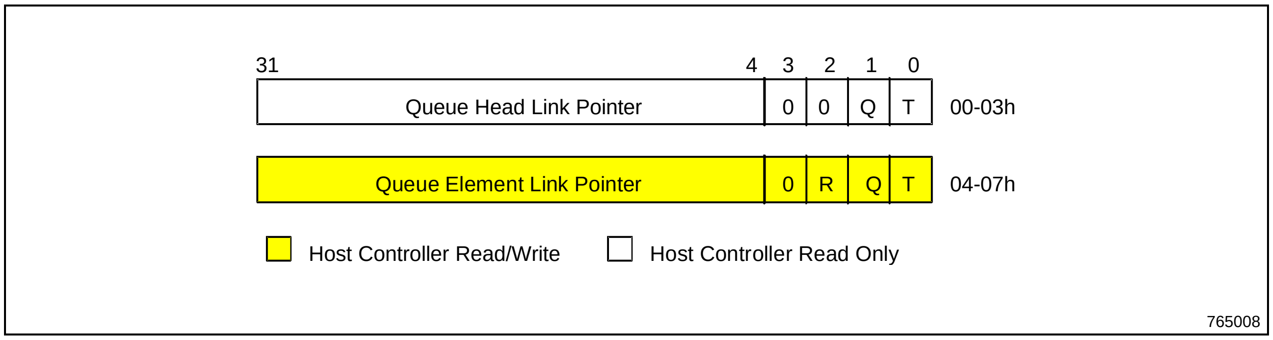

31
4
3
2
1
0
Queue Head Link Pointer
0
0
Q
T
00-03h
Queue Element Link Pointer
0
R
Q
T
04-07h
Host Controller Read/Write
Host Controller Read Only
765008

Figure 8. Generic Form of Queue Head (QH)

3.3.1 QUEUE HEAD LINK POINTER (DWORD 0: 00-03h)

The first DWord of a Queue Head contains a link pointer to the next data object to be processed after any required processing in this queue has been completed, as well as the control bits defined below.

Bit	Description
31:4	**Queue Head Link Pointer (QHLP).** This field contains the address of the next data object to be processed in the horizontal list and corresponds to memory address signals [31:4], respectively.
3:2	**Reserved.** These bits must be written as 0s.
1	**QH/TD Select (Q).** 1=QH. 0=TD. This bit indicates to the hardware whether the item referenced by the link pointer is another TD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched.
0	**Terminate (T).** 1=Last QH (pointer is invalid). 0=Pointer is valid (points to a QH or TD). This bit indicates to the Host Controller that this is the last QH in the schedule. If there are active TDs in this queue, they are the last to be executed in this frame.

3.3.2 QUEUE ELEMENT LINK POINTER (DWORD 1: 04-07h)

The second DWord of a Queue Head contains a link pointer to the first data object in the queue as well as the control bits defined below.

Bit	Description
31:4	**Queue Element Link Pointer (QELP).** This field contains the address of the next TD or QH to be processed in this queue and corresponds to memory address signals [31:4], respectively.
3	**Reserved.** This bit must be 0.
2	**Reserved.** This bit has no impact on operation. It may vary simply as a side effect of the Queue Element pointer update.
1	**QH/TD Select (Q).** 1=QH. 0=TD. This bit indicates to the hardware whether the item referenced by the link pointer is another TD or a QH. This allows the Host Controller to do the proper type of processing on the item after it is fetched. For entries in a queue, this bit is typically set to 0.
0	**Terminate (T).** 1=Terminate (No valid queue entries). This bit indicates to the Host Controller that there are no valid TDs in this queue. When HCD has new queue entries it overwrites this value with a new TD pointer to the queue entry.

3.4 Script and Data Transfer Primitives

The following sub-sections describe the details on how HCD and the Host Controller communicate via the Schedule data structures. The discussion is organized in a top-down manner, beginning with the basics of walking the Frame List, followed by a description of generic processing steps common to all transfer descriptors, and finally a discussion on Transfer Queuing.

3.4.1 EXECUTING THE SCHEDULE

HCD programs the Host Controller with the starting address of the Frame List and the Frame List index, then causes the Host Controller to execute the schedule by setting the Run/Stop bit in the Control register to Run. The Host Controller processes the schedule one entry at a time (this discussion does not preclude prefetching of schedule entries).

Schedule execution proceeds in the following fashion. The Host Controller first fetches an entry from the Frame List (Figure 9). This entry has three fields. Bit 0 indicates whether the address pointer field is valid. Bit 1 indicates whether the address points to a Transfer Descriptor or to a queue head. The third field is the pointer itself.

If isochronous traffic is to be moved in a given frame, the Frame List entry points to a Transfer Descriptor. If no Isochronous data is to be moved in that frame, the entry points to a queue head or the entry is marked invalid and no transfers are initiated in that frame. An entry is marked invalid when the T bit is set to 1. This indicates that the frame pointer field is not a valid pointer value.

If the Frame List entry indicates that it points to a Transfer Descriptor, the Host Controller fetches the entry and begins the operations necessary to initiate a transaction on the USB. Each TD contains a link field that points to the next entry, as well as indicating whether it is a TD or a QH.

If the Frame List entry contains a pointer to a QH, the Host Controller processes the information from the QH to determine the address of the next data object that it should process.

The Host Controller fetches the next entry from the Frame List when the millisecond allotted to the current frame expires. If the Host Controller is not able to process all of the Isochronous transfer descriptors during a given frame, those descriptors are retired by the software without having been executed. In a normal case this should not occur since software sets up isochronous transactions such that they should all complete. If an error occurs, such as a device babble or a consistency check error, then some of the isochronous transactions may not be completed.

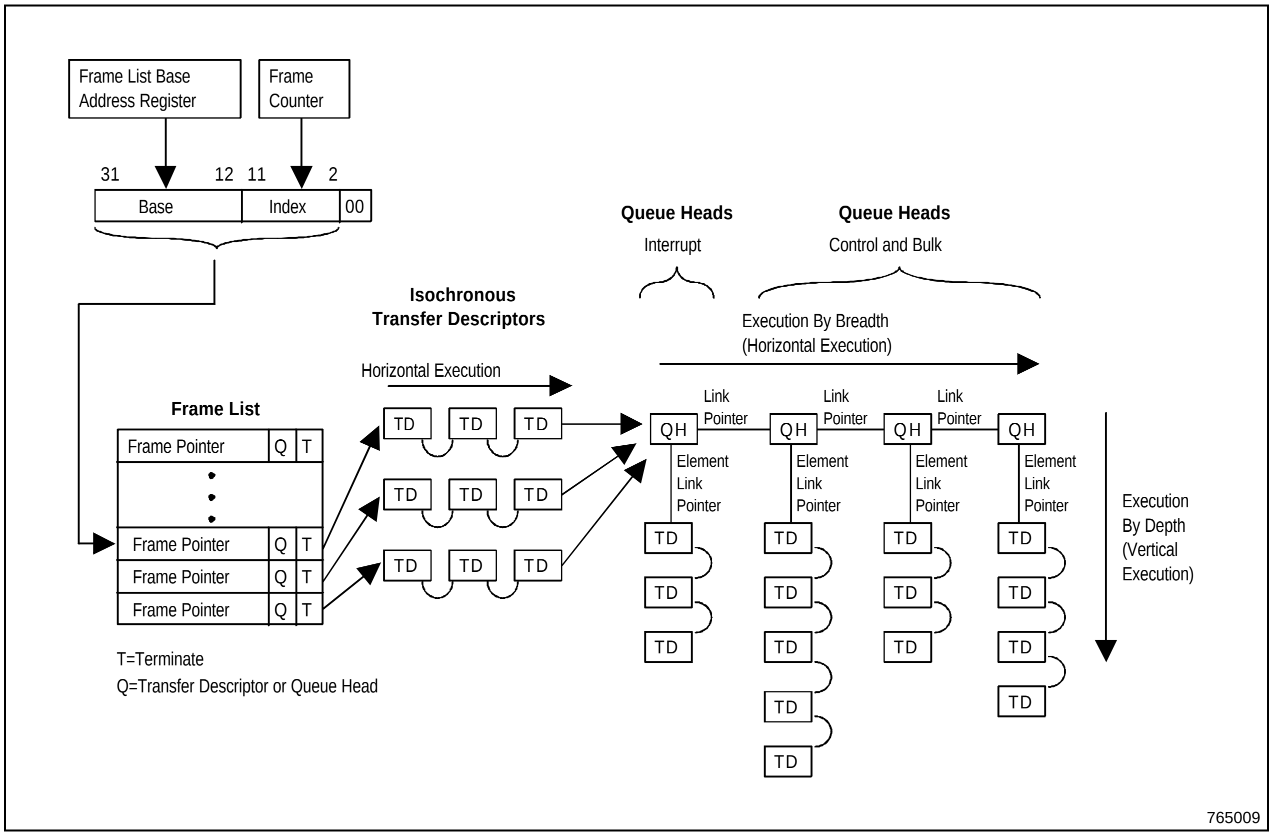

Frame List Base Address Register
Frame Counter
31
12
11
2
Base
Index
00
Queue Heads
Interrupt
Queue Heads
Control and Bulk
Isochronous Transfer Descriptors
Execution By Breadth (Horizontal Execution)
Horizontal Execution
Frame List
Frame Pointer
Q
T
TD
QH
Link Pointer
Element Link Pointer
Execution By Depth (Vertical Execution)
T=Terminate
Q=Transfer Descriptor or Queue Head
765009

Figure 9. Sample Schedule Layout

3.4.1.1 Processing Transfer Descriptors

There are three serial events that occur to complete execution of a Transfer Descriptor (Figure 10). These steps are enumerated in the following list.

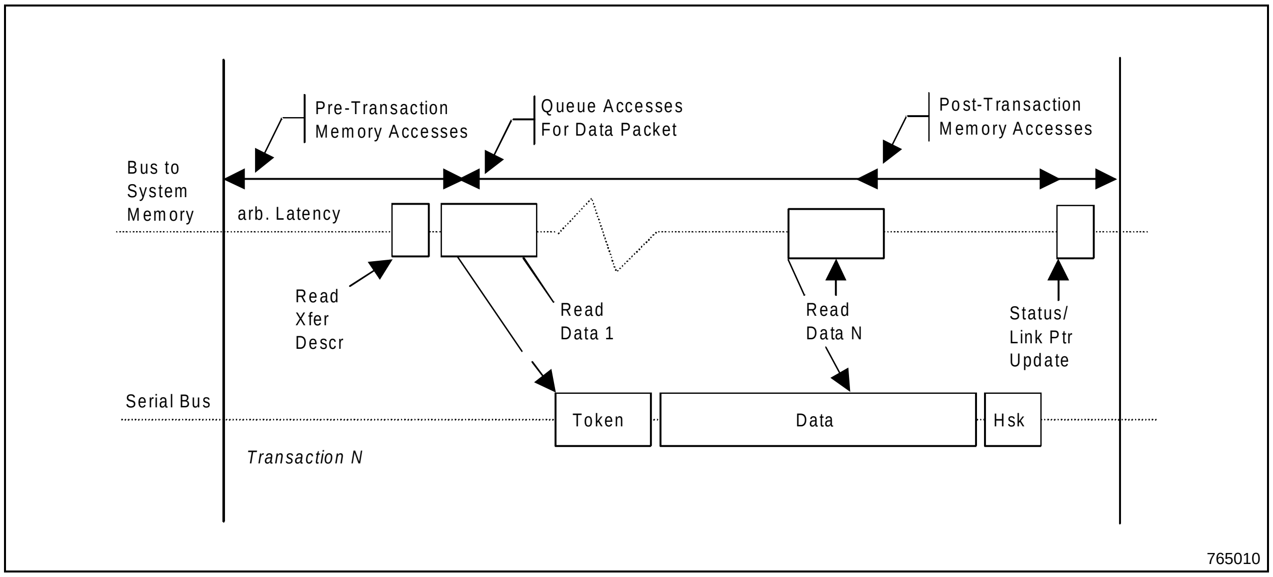


Figure 10. Diagram of an Example Transaction (For a Control Transfer)

1. Host Controller fetches and decodes the Transfer Descriptor. Assuming a Host-to-Function transaction, the Host Controller delays committing to the USB Transaction until the FIFO fills to an appropriate “trigger point”. When this threshold is reached, the Host Controller can then begin issuing the Transaction Token. If the direction is Function-to-Host, the Transaction Token may be issued as soon as ready, after the TD has been received.
2. The second event is the USB Transaction itself. A Transaction takes up to three phases to complete. Its duration depends on the maximum size data transfer, as specified in the TD.
3. The final event involves the time required to update the data structures for Post-Transaction Processing. Typically, this time can be hidden via write posting on the bus to system memory, if the bus supports this capability (e.g., the PCI bus).

The Host Controller executes a TD using the following, generalized algorithm. These basic steps are common across all modes of TDs. Subsequent sections present processing steps unique to each TD mode.

```
1.  Host Controller Fetches TD.
2.  Build Token, actual bits are in TD.Token.
3.  if (Host-to-Function) then
        [PCI Access] issue request for data, (referenced through TD.BufferPointer)
        wait for first chunk data arrival
    end if
4.  [Begin USB Transaction] Issue Token (from token built in 2, above) and begin data transfer.
    if (Host-to-Function) then
        goto 5
    else
        goto 6
    end if
5.  Fetch data from memory (via TD.BufferPointer) and transfer over USB until TD.Max-Length bytes have been
    read and transferred. [Concurrent system memory and USB Accesses].
    goto 8.
6.  Wait for data to arrive (from USB). Write incoming bytes into memory beginning at TD.BufferPointer. Internal
    HC buffer should signal end of data packet. Number of bytes received must be ≤ TD.Max-Length; The length
    of the memory area referenced by TD.BufferPointer must be ≥ TD.Max-Length. [Concurrent system memory
    and USB Accesses].
7.  Issue handshake based on status of data received (Ack or Time-out). Go to 9.
8.  Wait for handshake, if required [End of USB Transaction].
9.  Update Status [PCI Access] (TD.Status and TD.ActualLength).
10. Proceed to next entry.
```

3.4.1.2 Processing Isochronous Transfer Descriptors

Isochronous descriptors are linked in lists on a frame basis. This permits the clients to schedule traffic to occur at specific points in time with a 1 ms granularity. Isochronous service provides a fixed rate data transfer between the host and the endpoint. Isochronous TDs are processed as follows:

1. Host Controller fetches the Isochronous TD.
2. Host Controller decodes the TD fields to determine transaction characteristics.
3. Host Controller issues USB token.
4. When transaction completes, the Host Controller updates status marking TD inactive.
5. Host Controller fetches the TD or QH pointed to by the Link Pointer field of the current TD.
6. Process continues.

3.4.1.3 Processing Bulk, Control and Interrupt Transfer Descriptors

Bulk, Control, and Interrupt TDs derive their guaranteed data delivery transfer characteristic through the use of queues. From a hardware perspective, they operate identically. The various behaviors are produced by the way in which software programs the fields of the TDs. Non-isochronous TDs are processed as follows:

1. Host Controller fetches Queue Head (QH) and checks for active entry.
2. If entry is active, Host Controller fetches the TD or QH pointed to by the QH Element pointer. If TD, go to 3. If QH, go to 1. If entry is inactive, go to 10.
3. Host Controller decodes the TD fields to determine transaction characteristics.
4. Host Controller issues USB token and performs transaction.
5. When transaction completes, Host Controller updates status.
6. If the transaction was successful, the TD is marked inactive. Go to 9.
7. Else if the transaction is unsuccessful, but the error threshold has not been reached, the TD is left active so it can be retried in a subsequent frame. Go to 10.
8. Else if the transaction was unsuccessful and exceeded the error threshold, the TD is marked inactive. Go to 10.
9. Host Controller writes the link pointer from the current TD into the element pointer field of the QH structure. If the Vf bit is set in the link pointer, go to 2. Otherwise, it goes to 10.
10. Host Controller fetches the QH or TD pointed to by the Queue Head Link Pointer field of the current QH. If the Queue Head Link Pointer field has the T bit set to 1, The Host Controller idles until the 1 ms frame timer expires.
11. Process continues.

3.4.1.4 Command Register, Status Register, and TD Status Bit Interaction

Table 7 shows the interaction between bits in the Command register, Status Register, and status fields in the TD.

Table 7. Command Register, Status Register, and TD Status bit Interaction

	Run/ Stop	**USB Int**	**USB Error Int**	**Resume received**	**Host System error**	**HC Process Error**	**HC halted**	**Config flag**	**Active bit in TD status**	**Babble bit in TD status**	**Stall bit in TD status**
Babble			1				0		0	1	1
CRC/Time Out Error			1*				0		0*		1*
Illegal PID, PID Error	0					1	1				
PCI Master Abort	0				1		1				
PCI Target Abort	0				1		1				
Suspend Mode	0**						1				
Resumed Received from USB and Suspend Mode = 1				1							
Run/Stop = 0	0						1				
Config Flag Set								1			
HC Reset	0	0	0	0	0	0	0	0			
Global Reset	0	0	0	0	0	0	0	0			
Max Length (illegal)	0					1	1				
IOC = 1		1									
Stall			1						0		1
Bit Stuff Error			1*						0*		1*
Short Packet Detect		1							0		
Data Buffer Error			1*						0*		1*

* if error counter counted down from 1 to 0

** Suspend mode can be entered only when Run/Stop bit is 0

Note that if a NAK or STALL response is received from a SETUP transaction, a Time Out Error will be reported. This will cause the Error counter to decrement and the CRC/Timeout Error status bit to be set within the TD Control and Status DWORD during write back. If the Error counter changes from 1 to 0, the Active bit will be reset to 0 and Stalled bit to 1 as normal.

3.4.2 TRANSFER QUEUING

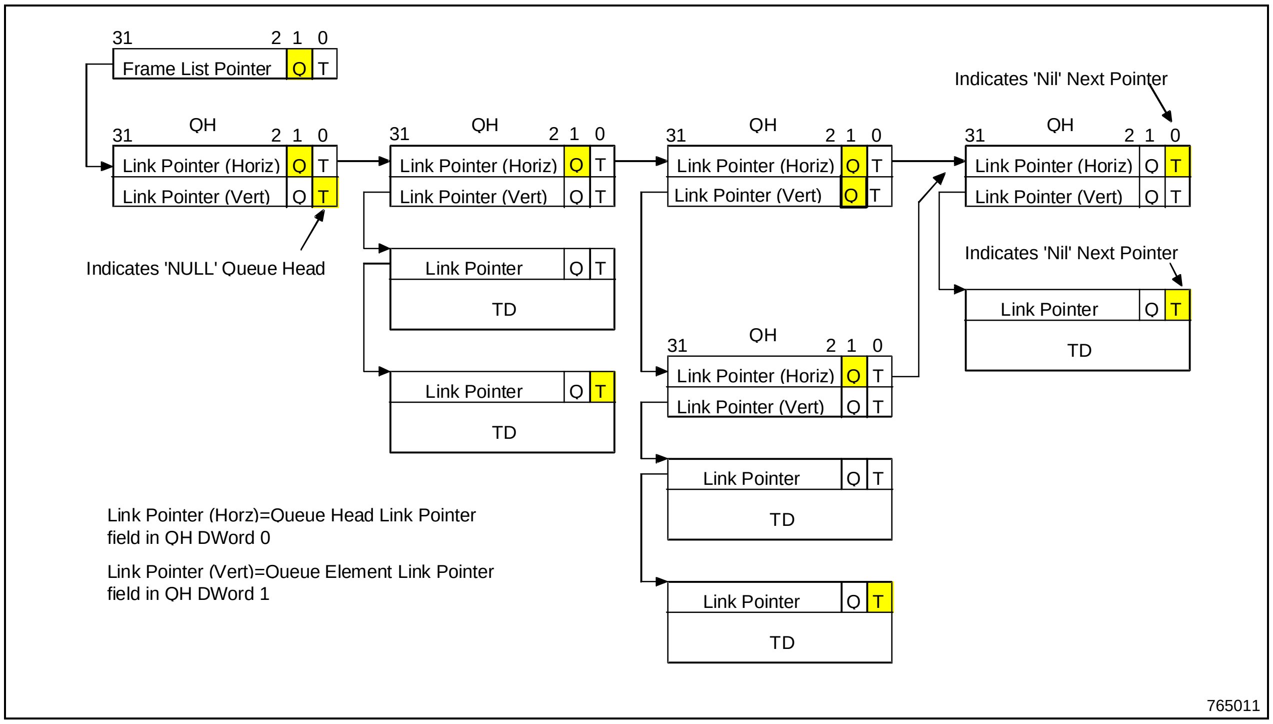
Transfer Queuing is a feature where transfer descriptors are accessed via a queue type data structure. The Host Controller must go through a QH to access the top element in the queue. The operational semantics of transfer queues, described in detail later, roughly dictate that the queue is not advanced until the top element's execution status satisfies some *advance criteria*. Transfer Queues are used to implement a guaranteed data delivery stream to a USB Endpoint. Transfer Queues are composed of two parts—a QH and a linked list. The linked list of TDs and QHs has an indeterminate length (0 to n). Figure 11 illustrates the layouts of several possible transfer queues.

The QH contains two link pointers and is organized as two contiguous DWords. The first DWord is a horizontal pointer (Queue Head Link Pointer), used to link a single transfer queue with either another transfer queue, or a TD (target data structure depends on Q bit). If the T bit is set, this QH represents the last data structure in the current Frame. The T bit informs the Host Controller that no further processing is required until the beginning of the next frame. The second DWord is a vertical pointer (Queue Element Link Pointer) to the first data structure (TD or QH) being managed by this QH. If the T bit is set, the queue is empty. This pointer may reference a TD or another QH.

Figure 11 illustrates four example queue conditions. The first QH (on far left) is an example of an “empty” queue; the termination bit (T Bit), in the vertical link pointer field, is set to 1. The horizontal link pointer references another QH. The next queue is the expected typical configuration. The horizontal link pointer references another QH, and the vertical link pointer references a valid TD.

Typically, the vertical pointer in a QH points to a TD. However, as shown in Figure 11 (third example from left side of figure) the vertical pointer could point to another QH. When this occurs, a new Q Context is entered and the Q Context just exited is NULL (Host Controller will not update the vertical pointer field).

The far right QH is an example of a frame ‘termination’ node. Since its horizontal link pointer has its termination bit set, the Host Controller assumes there is no more work to complete for the current Frame.



31 2 1 0
Frame List Pointer Q T
QH
Link Pointer (Horiz) Q T
Link Pointer (Vert) Q T
Indicates 'NULL' Queue Head
Link Pointer Q T
TD
Indicates 'Nil' Next Pointer
Link Pointer (Horz)=Queue Head Link Pointer field in QH DWord 0
Link Pointer (Vert)=Queue Element Link Pointer field in QH DWord 1
765011

Figure 11. Example Queue Conditions

Transfer Queues are based on the following characteristics:

- A QH's vertical link pointer (Queue Element Link Pointer) references the 'Top' queue member. A QH's horizontal link pointer (Queue Head Link Pointer) references the "next" work element in the Frame.
- Each queue member's link pointer references the next element within the queue.[1]
- In the simplest model, the Host Controller follows vertical link point to a queue element, then executes the element. If the completion status of the TD satisfies the *advance criteria*, the Host Controller *advances* the queue by writing the just-executed TD's link pointer back into the QH's Queue Element link pointer. The next time the queue head is traversed, the *next* queue element will be the Top element.
- The traversal has two options: Breadth first, or Depth first. A flag bit in each TD (Vf - Vertical Traversal Flag) controls whether traversal is Breadth or Depth first. The default mode of traversal is Breadth-First. For Breadth-First, the Host Controller only executes the top element from each queue. The execution path is:

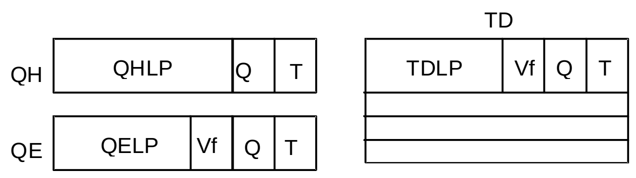
  QH (Queue Element Link Pointer) $\rightarrow$ TD $\rightarrow$ Write-Back to QH (Queue Element Link Pointer) $\rightarrow$ QH (Queue Head Link pointer).

  Breadth-First is also performed for every transaction execution that fails the *advance criteria*. This means that if a queued TD fails, the queue does not advance, and the Host Controller traverses the QH's Queue Head Link Pointer.
- In a Depth-first traversal, the top queue element must complete successfully to satisfy the *advance criteria* for the queue. If the Host Controller is currently processing a queue, and the advance criteria are met, and the Vf bit is set, the Host Controller follows the TD's link pointer to the next schedule work item.
- Note that regardless of traversal model, when the advance criteria are met, the successful TD's link pointer is written back to the QH's Queue Element link pointer.
- When the Host Controller encounters a QH, it caches the QH internally, and sets internal state to indicate it is in a Q-context. It needs this state to update the correct QH (for auto advancement) and also to make the correct decisions on how to traverse the Frame List.
- Restricting the advancement of queues to advancement criteria implements a guaranteed data delivery stream.
- A queue is NEVER advanced on an error completion status (even in the event the error count was exhausted).
- Table 8 lists the general queue advance criteria, which are based on the execution status of the TD at the 'Top' of a currently 'active' queue.

Table 8. Queue Advance Criteria

Function-to-Host (IN)			**Host-to-Function (OUT)**		
Non-NULL	**NULL**	**Error/NAK**	**Non-NULL**	**NULL**	**Error/NAK**
Advance Q	Advance Q	Retry Q Element	Advance Q	Advance Q	Retry Q Element

Table 9 is a decision table illustrating the valid combinations of link pointer bits and the valid actions taken when advancement criteria for a queued transfer descriptor are met. The column headings for the link pointer fields are encoded, based on the following list:



765012

[1] Understand that link pointers reference either Queue Heads *OR* Transfer Descriptors. The Host Controller has all the information it requires to correctly traverse the list, built by Software.

Legends:

QHLP = Queue Head Link Pointer (or Horizontal Link Pointer)	QE.Q = Q bit in QE
QELP = Queue Element Link Pointer (or Vertical Link Pointer)	QE.T = T bit in QE
TDLP = TD Link Pointer	TD. Vf = Vf bit in TD
QH.Q = Q bit in QH	TD.Q = Q bit in TD
QH.T = T bit in QH	TD. Vf = Vf bit in TD

Table 9. USB Schedule List Traversal Decision Table

#	Q Context	QH Q	QH T	QE Q	QE T	TD Vf	TD Q	TD T	Description
1	0	-	-	-	-	x	0	0	Not in Queue Execute TD Use TDLP to get next TD
2	0	-	-	-	-	x	x	1	Not in Queue Execute TD End of Frame
3	0	-	-	-	-	x	1	0	Not in Queue Execute TD Use TDLP to get next (QH+QE) Set Qcontext to 1.
4	1	0	0	0	0	0	x	x	In Queue Use QELP to get TD Execute TD Update QELP with TDLP Use QHLP to get next TD.
5	1	x	x	0	0	1	0	0	In Queue Use QELP to get TD Execute TD Update QELP with TDLP Use TDLP to get next TD
6	1	x	x	0	0	1	1	0	In Queue Use QELP to get TD Execute TD Update QELP with TDLP Use TDLP to get next (QH+QE)
7	1	0	0	x	1	x	x	x	In Queue Empty queue Use QHLP to get next TD
8	1	x	x	1	0	-	-	-	In Queue Use QELP to get (QH+QE)
9	1	x	1	0	0	0	x	x	In Queue Use QELP to get TD Execute TD Update QELP with TDLP End of Frame
10	1	x	1	x	1	x	x	x	In Queue Empty queue End of Frame
11	1	1	0	0	0	0	x	x	In Queue Use QELP to get TD Execute TD Update QELP with TDLP Use QHLP to get next (QH+QE).

#	Q Context	QH Q	QH T	QE Q	QE T	TD Vf	TD Q	TD T	Description
12	1	1	0	x	1	x	x	x	In Queue Empty queue Use QHLP to get next (QH+QE)

Figure 12 illustrates a List Walking State Diagram for a Host Controller. The state diagram is more robust than the above presentation, as it contains transitions for standalone transfer descriptors, as well as queued transfer descriptors (including Depth and Breadth Traversal). In addition, the diagram allows TDs and QHs to be referenced from any link pointer which may be encountered by the Host Controller. The transitions in Figure 12 are derived from the information in Table 9.

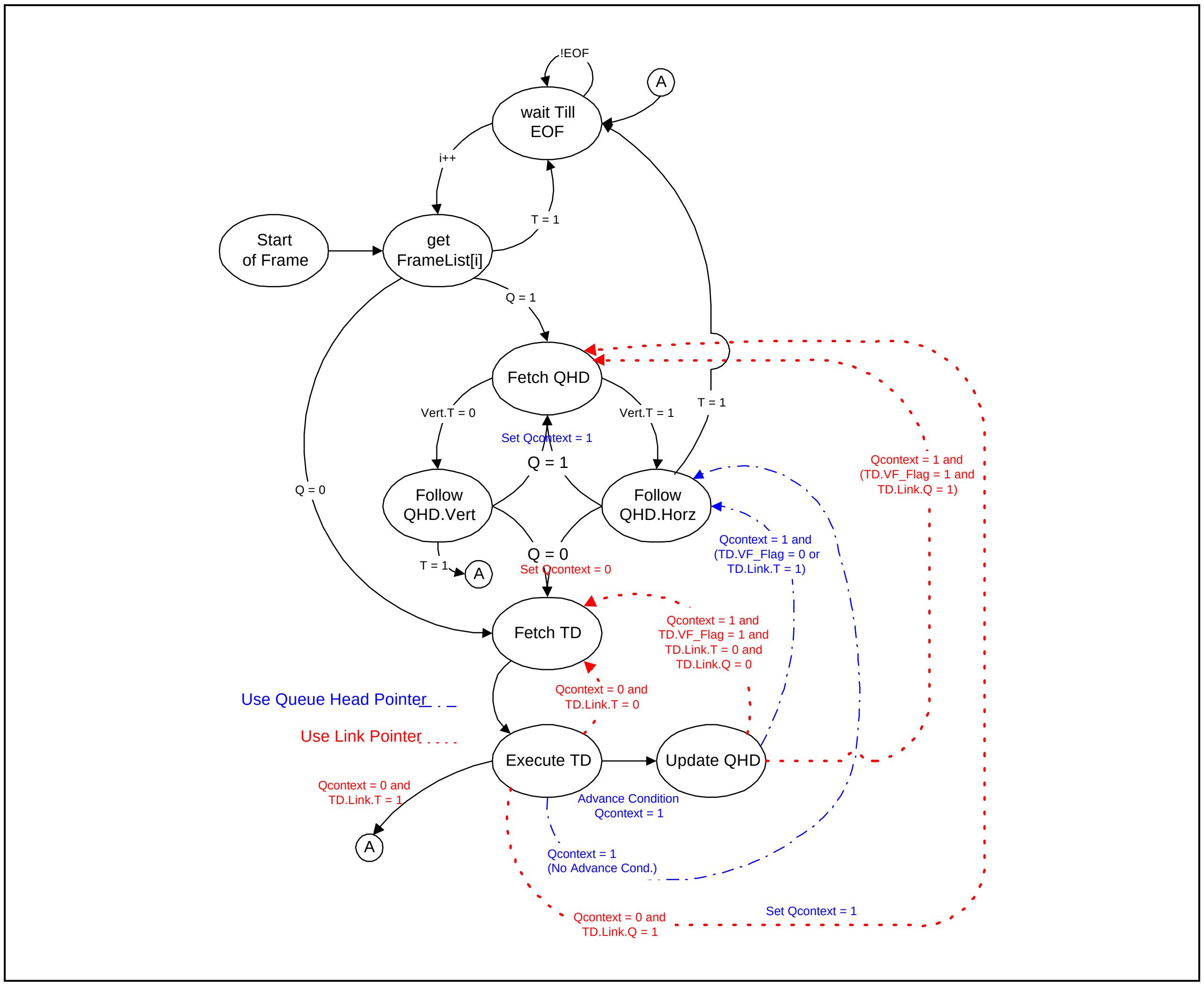

!EOF
A
wait Till EOF
i++
T = 1
Start of Frame
get FrameList[i]
Q = 1
Fetch QHD
Vert.T = 0
Vert.T = 1
T = 1
Set Qcontext = 1
Q = 1
Qcontext = 1 and (TD.VF_Flag = 1 and TD.Link.Q = 1)
Q = 0
Follow QHD.Vert
Follow QHD.Horz
Qcontext = 1 and (TD.VF_Flag = 0 or TD.Link.T = 1)
T = 1
A
Q = 0
Set Qcontext = 0
Fetch TD
Qcontext = 1 and TD.VF_Flag = 1 and TD.Link.T = 0 and TD.Link.Q = 0
Use Queue Head Pointer
Qcontext = 0 and TD.Link.T = 0
Use Link Pointer
Execute TD
Update QHD
Qcontext = 0 and TD.Link.T = 1
Advance Condition Qcontext = 1
A
Qcontext = 1 (No Advance Cond.)
Qcontext = 0 and TD.Link.Q = 1
Set Qcontext = 1

Figure 12. USB Linked List Traversal State Diagram

4. INTERRUPTS

The USB Host Controller hardware (HC) provides interrupt capability based on a number of sources. There are two general groups of interrupt sources, those resulting from execution of transactions in the schedule, and those resulting from a Host Controller operation error. All transaction-based sources are maskable by the Host Controller Driver (HCD) through the Host Controller's Interrupt Enable register. Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt. During normal operation, interrupts are deferred until the last transaction in the frame is complete. This results in the normal runtime interrupts being processed in a batch fashion. If an HC process error or host system error (both considered fatal) occur, the HC halts and immediately issues a hardware interrupt to the system.

If an interrupt has been scheduled to be generated for the frame, the interrupt is not signaled until after the status for the last complete transaction in the frame has been written back to host memory. This may sometimes result in the interrupt not being signaled until after the Start Of Frame (SOF) for the next frame has been sent. This guarantees that the software can safely process through Frame List Current Index -1 when it is servicing an interrupt.

Initial interrupt processing is the same, regardless of the reason for the interrupt. When an interrupt is signaled by the hardware, CPU control is transferred to HCD's USB interrupt handler. The precise mechanism to accomplish the transfer is OS specific. For this discussion it is just assumed that control is received. When the interrupt handler receives control, its first action is to read the status from the Host Controller. At this point the handler can determine whether the interrupt is due to schedule processing or some other event. HCD does the processing required to service the hardware and then, through an OS specific mechanism, schedules a deferred procedure call (DPC) which will execute later. The DPC routine uses a set of internal data structures to access the Queue Heads and Transaction Descriptors of active endpoints to determine what events have occurred. The precise mechanisms used are beyond the scope of this document.

4.1 Transaction Based

4.1.1 CRC ERROR / TIME-OUT

A CRC/Timeout error occurs when a packet transmitted from the host to a USB device or a packet transmitted from a USB device to the host generates a CRC error. The Host Controller is informed of this event by a time out from the USB device or by the Host Controller's CRC checker generating an error on reception of the packet. Additionally, a USB bus time-out occurs when USB devices do not respond to a transaction phase within a period defined in the USB specification. Either of these conditions will cause the C_Err field of the TD to be decremented. This field is initialized to 3 for all data types except Isochronous by the HCD software. When the C_Err field decrements to zero, several things occur; The Active bit in the TD is cleared, the Stalled bit in the TD is set, the CRC/Timeout bit in the TD is set. At the end of the frame, the USB Error Interrupt bit is set in the HC status register. If the CRC/Timeout interrupt is enabled in the Interrupt Enable register, a hardware interrupt will be signaled to the system.

4.1.2 INTERRUPT ON COMPLETION (IOC)

Transfer Descriptors contain a bit that can be set to cause an interrupt on their completion. The completion of the transaction associated with that block causes the USB Interrupt bit in the HC Status Register to be set at the end of the frame in which the transfer completed. When a TD is encountered with the IOC bit set to 1, the IOC bit in the HC Status register is set to 1 at the end of the frame if the active bit in the TD is set to 0 (even if it was set to zero when initially read). If the IOC Enable bit in the Interrupt Enable register is set, a hardware interrupt is signaled to the system. This status bit is set whether the TD completes successfully, or because of errors. If the completion is because of errors, the USB Error bit in the HC status register is also set.

4.1.3 SHORT PACKET DETECT (SPD)

A transfer set is a collection of data which requires more than 1 USB transaction to completely move the data across the USB. An example might be a large print file which requires numerous TDs in multiple frames to completely transfer the data. Reception of a data packet that is less than the endpoint’s Max Packet size during Control, Bulk or Interrupt transfers signals the completion of the transfer set, even if there are active TDs remaining for this transfer set. Setting the SPD bit in a TD indicates to the HC to set the USB Interrupt bit in the HC status register at the end of the frame in which this event occurs. This feature streamlines the processing of input on these transfer types. If the Short Packet Interrupt Enable bit in the Interrupt Enable register is set, a hardware interrupt is signaled to the system at the end of the frame where the event occurred.

4.1.4 SERIAL BUS BABBLE

When a device transmits on the USB for a time greater than its assigned Max Length, it is said to be babbling. Since isochrony can be destroyed by a babbling device, this error results in the Active bit in the TD being cleared to 0 and the Stalled and Babble bits being set to one. The C_Err field is not decremented for a babble. The USB Error Interrupt bit in the HC Status register is set to 1 at the end of the frame. A hardware interrupt is signaled to the system. If an EOF babble was caused by the host controller (due to incorrect schedule for instance), the host controller will force a bit stuff error followed by an EOP and the start of the next frame.

4.1.5 STALLED

This event indicates that a device/endpoint returned a STALL handshake during a transaction or that the transaction ended in an error condition. This indicates that the endpoint has reached a condition where no more activity can occur without intervention from the driver. In addition to the TDs Stalled bit being set, the Active bit will be cleared. HCD will not accept any more transfers on this endpoint until the condition is cleared by driver software. Like the Babble event, reception of a STALL does not decrement the error counter. A hardware interrupt is signaled to the system.

4.1.6 DATA BUFFER ERROR

This event indicates that an overrun of incoming data or a underrun of outgoing data has occurred for this transaction. This would generally be caused by the host controller not being able to access required data buffers in memory within necessary latency requirements. Either of these conditions will cause the C_Err field of the TD to be decremented. This field is initialized to 3 for all data types except Isochronous by the HCD software. When the C_Err field decrements to zero, the Active bit in the TD is cleared to 0, the Stalled bit is set to one, the USB Error Interrupt bit in the HC Status register is set to 1 at the end of the frame and a hardware interrupt is signaled to the system.

4.1.7 BIT STUFF ERROR

A bit stuff error results from the detection of a sequence of more that 6 ones in a row within the incoming data stream. This will cause the C_Err field of the TD to be decremented. This field is initialized to 3 for all data types except Isochronous by the HCD software. When the C_Err field decrements to zero, the Active bit in the TD is cleared to 0, the Stalled bit is set to one, the USB Error Interrupt bit in the HC Status register is set to 1 at the end of the frame and a hardware interrupt is signaled to the system.

4.2 Non-Transaction Based

4.2.1 RESUME RECEIVED

This event indicates that the HC received a RESUME signal from a device on the USB during a global suspend. If this interrupt is enabled in the HC Interrupt Enable register, a hardware interrupt will be signaled to the system allowing the USB to be brought out of the suspend state and returned to normal operation.

4.2.2 HOST CONTROLLER PROCESS ERROR

The HC monitors certain critical fields during operation to ensure that it does not process corrupted data structures. These include checking for a valid PID and verifying that the MaxLength field is less than 1280. If it detects a condition that would indicate that it is processing corrupted data structures, it immediately halts processing, sets the HC Process Error bit in the HC Status register and signals a hardware interrupt to the system. If the transaction has already started on the USB when the condition is recognized, the host controller should force an error condition on the bus (such as invalid PID or inserted bit stuff error) and then halt. This interrupt cannot be disabled through the Interrupt Enable register. This is a catastrophic failure requiring major software intervention at the system level for recovery.

4.2.3 HOST SYSTEM ERROR

The Host Controller sets this bit to 1 when a serious error occurs during a host system access involving the Host Controller module. In a PCI system, conditions that set this bit to 1 include PCI Parity error, PCI Master Abort, and PCI Target Abort. When this error occurs, the Host Controller clears the Run/Stop bit in the Command register to prevent further execution of the scheduled TDs. This interrupt cannot be disabled through the Interrupt Enable register. This level of failure would typically be handled by System Software rather than HCD.

5. KEYBOARD AND MOUSE LEGACY SUPPORT

In today's PC environment, windows-based application software typically does not directly access I/O locations for the keyboard and mouse functions. The system software provides this function. However, in earlier environments some application software directly accessed the keyboard controller's I/O space. In most desktop PC systems the keyboard controller is based on the 8042 controller.

For a USB-based keyboard and mouse to be software compatible with an 8042 controller-based keyboard and mouse, older software must be able to access the same I/O locations with the same values. Furthermore, the time delays associated with the ports must be roughly the same.

The 8042 controller-based keyboard controller (KBC) typically has the following characteristics:

- It provides serial interfaces for both the keyboard and PS/2-style mouse.
- The 8042 controller's registers are mapped to the ISA I/O addresses 60h and 64h.
- The 8042 controller asserts two interrupt lines; IRQ1 for the keyboard and IRQ12 for the mouse.

This section provides a legacy PCI device implementation example that supports the 8042 controller-based keyboard and mouse commands in a USB environment. The legacy software support method described herein requires a combination of hardware and software.

8042 controller emulation is only needed in certain OS environments (e.g., DOS). Windows*, Windows NT*, Windows 95*, and UNIX-based programs do not directly access the hardware. Note however, that Windows 95 supports a drop-through mode in which DOS applications are permitted to directly access the I/O space.

Note that UHCI does not require implementation of legacy keyboard and mouse support. Neither is this implementation the only possible solution for providing this support.

5.1 Operation

Software that is run early in the boot process must be modified to configure the USB controller, enable the keyboard and mouse, and set up the USB controller's scheduler. This code is typically the system BIOS. The major assumptions are:

- A standard 8042 device-like keyboard controller is present in the system at IO locations 60h and 64h. The control logic described here must be able to selectively intercept or pass through IO cycles targeted for the KBC.
- No changes are needed to the 8042 keyboard controller code. However, if changes can be made, then the amount of SMM code required can be reduced.
- USB aware operating systems handle the host controller completely without any legacy hardware or software support. No operating system should take control of the USB host controller while the Legacy Support hardware and software are enabled.

The following two sections describe the general command and data sequences.

5.1.1 COMMANDS GOING TO THE KBC, KEYBOARD, OR MOUSE

Some legacy software will attempt to send commands directly to the keyboard controller (8042 device). In addition, some legacy software may also send commands directly to the keyboard or mouse, although this is rare. This case is typically done by BIOS, which would be already aware of the peripheral being on USB.

The keyboard and mouse commands perform operations like resetting the keyboard, setting the typematic rate, turning on or off LEDs, or changing the scan-code translation method. The software writes the command into the 8042 keyboard controller. The 8042 keyboard controller sends the command out serially to the keyboard or mouse. An interrupt can be generated when the action completes. A USB-based keyboard has similar commands (reset, turn on/off LEDs, etc.) and the commands needs are sent via USB. This leads to the following requirements:

- Since no changes are desired in the 8042 controller code, the combination of software and system logic must determine if the data going to the 8042 controller should be diverted to the keyboard on the USB. The logic must be able to generate an SMI# based on command writes to the keyboard controller. Furthermore, for multi-byte commands, the software must be able to generate an SMI# during a read cycle (which might occur in-between subsequent writes).
- The logic must be able to selectively generate an SMI# based on reads or writes to the keyboard controller. Cycles where an SMI# is generated are not passed to the keyboard controller.
- There must be SMM code to interpret the commands and status results associated with the keyboard controller.
- For one specific case, the keyboard controller is used to control the A20GATE signal. In this case, it is not desired to invoke SMI# due to the great performance degradation. To handle this specific case, the logic must "pass" commands associated with setting and resetting the A20GATE signal. Thus, the fourth required change is that the logic must pass A20GATE sequences to the external 8042 controller. This is called the “A20GATE Pass Through Sequence”.

5.1.2 KEYSTROKE OR MOUSE DATA RECEIVED

In legacy hardware implementations, one of the more common actions performed by the 8042 controller is receiving keystroke or mouse movement data. The keyboard controller deserializes the incoming data streams, generates an interrupt, provides a status value in the status register, and finally provides the data bytes associated with the keystroke or mouse movement. A USB-based keyboard or mouse will be similar; however, the USB controller will store the incoming data in main memory, rather than at an I/O location.

This method requires that SMM code be invoked based on the incoming USB data. The SMM code manages the USB data queues, takes the received data, and writes it into the 8042 device using the "D2" (keyboard) or "D3" (mouse) commands. This instructs the 8042 controller to process the incoming data as if it had come from the standard PS/2 keyboard or mouse. Once the data is placed in its registers, the 8042 controller generates the appropriate interrupt (IRQ1 for the keyboard or IRQ12 for the mouse). The data is then processed by the normal interrupt handlers.

To take advantage of this scheme, the SMM code must be invoked (via SMI#). The USB controller must also be able to generate an SMI# based on the data being stored to main memory. This results in the following requirements:

- There must be the ability to generate an SMI# based on a USB interrupt.
- There must be additional code to move the data to the appropriate registers and manage the USB controller during runtime. It is envisioned that this would be part of the SMM code.

5.2 Register Interface

For implementations in a PCI device, the legacy support register is located at offset C0–C1h, in function 2 PCI configuration space.

5.2.1 LEGSUP—LEGACY SUPPORT REGISTER (PCI Configuration - Function 2)

PCI Address Offset:	C0–C1h
Default:	2000h
Attribute:	Read/Write Clear

This register provides control and status capability for the legacy keyboard and mouse functions.

Bit	Description
15	**End OF A20GATE Pass Through Status (A20PTS)—R/WC.** This bit is set to 1 to indicate that the A20GATE pass-through sequence has ended. Software must use the enable bits to determine the exact cause of an SMI#. Software clears this bit by writing a 1 to it.
14	**Reserved.**

Bit	Description
13	**USB PIRQ Enable (USBPIRQDEN) —R/W.** 1 (default) = USB interrupt is routed to PIRQD. 0 = USB interrupt does not route to PIRQD. This bit prevents the USB controller from generating an interrupt. Note that it will probably be configured to generate an SMI using bit 4 of this register. Default to 1 for compatibility with older USB software.
12	**USB IRQ Status (USBIRQS) —RO.** This bit is set to 1 to indicate that the USB IRQ is active. Software must use the enable bits to determine the exact cause of an SMI#. Writing a 1 to this bit will have no effect. Software must clear the IRQ via the USB controller.
11	**Trap By 64h Write Status (TBY64W)—R/WC.** This bit is set to 1 to indicate that a write to port 64h occurred. Software must use the enable bits to determine the exact cause of an SMI#. Software clears this bit by writing a 1 to it.
10	**Trap By 64h Read Status (TBY64R)—R/WC.** This bit is set to 1 to indicate that a read to port 64h occurred. Software must use the enable bits to determine the exact cause of an SMI#. Software clears this bit by writing a 1 to it.
9	**Trap By 60h Write Status (TBY60W)—R/WC.** This bit is set to 1 to indicate that a write to port 60h occurred. Software must use the enable bits to determine the exact cause of an SMI#. Software clears this bit by writing a 1 to it.
8	**Trap By 60h Read Status (TBY60R)—R/WC.** This bit is set to 1 to indicate that a read to port 60h occurred. Software must use the enable bits to determine the exact cause of an SMI#. Software clears this bit by writing a 1 to it.
7	**SMI At End Of Pass Through Enable (SMIEPTE)—R/W.** 1=Enable the generation of an SMI when the A20GATE pass-through sequence has ended. 0 (default) = Disable. This may be required if an SMI is generated by a USB interrupt in the middle of an A20GATE pass through sequence and needs to be serviced later.
6	**Pass Through Status (PSS)—RO.** 1 =A20GATE pass through sequence is currently in progress. 0 (default) = Not currently executing the A20GATE pass-through sequence. This bit indicates that the host controller is executing the A20GATE pass-through sequence. If software needs to reset this bit, it should set Bit 5 to 0 causing the host controller to immediately end the A20GATE pass through sequence.
5	**A20Gate Pass Through Enable (A20PTEN)—R/W.** 1=Enable A20GATE pass through sequence. 0 (default) = Disable. When enabled, the logic will pass through the following A20GATE command sequence: **Cycle** / **Address** / **Data** Write / 64h / D1h / (1 or more) (Starts the Sequence) Write / 60h / xxh Read / 64h / N/A / (0 or more) Write / 64h / Ffh / (Standard End of A20GATE Pass Through Sequence) Any deviation seen in the above sequence will cause the host controller to immediately exit the sequence and return to standard operation, performing an I/O trap and generating an SMI# if appropriate enable bits are set. When enabled, SMI# will not be generated during the sequence, even if the various enable bits are set. Note that during a Pass-through sequence, the above status bits will not be set for the I/O accesses that are part of the sequence.
4	**Trap/SMI ON IRQ Enable (USBSMIEN)—R/W.** 1 = Enable SMI# generation on USB IRQ. 0 (default) = Disable.
3	**Trap/SMI On 64h Write Enable (64WEN)—R/W.** 1 = Enable I/O Trap and SMI# generation on port 64h write. 0 (default) = Disable.
2	**Trap/SMI On 64h Read Enable (64REN)—R/W.** 1 = Enable I/O Trap and SMI# generation on port 64h read. 0 (default) = Disable.
1	**Trap/SMI On 60h Write Enable (60WEN)—R/W.** 1 = Enable I/O Trap and SMI# generation on port 60h write. 0 (default) = Disable.
0	**Trap/SMI On 60h Read Enable (60REN)—R/W.** 1 = Enable I/O Trap and SMI# generation on port 60h read. 0 (default) = Disable.

5.2.2 KBC ACCESS LOGIC

The logic also needs to block the accesses to the 8042 controller. If there is an external 8042 controller, this is accomplished by not activating the 8042 controller chip select (e.g., logically ANDing the 4 enables for 60R, 60W, 64R, 64W with the 4 types of accesses to determine if 8042CS should go active). It must allow for the "A20GATE Pass Through Sequence.

Where the keyboard controller is not accessed using an explicit chip select (e.g., in some Super I/O components), the ISA cycle has to be modified so that the external device does not recognize the keyboard cycle. Several options are available:

1. Allow the address and data to flow through normally, but block the IOR# or IOW#.
2. Allow the data and IOR# or IOW# to flow through normally, but modify the address to a value that does not cause the keyboard controller to respond.
3. Allow the data and IOR# or IOW# to flow through normally, but drive the AEN signal active so that the peripheral thinks a DMA cycle is occurring, and will not respond.
4. The PCI cycle will be terminated normally, but no ISA cycle will occur.

5.3 Other Considerations

Table 10 lists considerations when implementing the method described in this section.

Table 10. Considerations

Consideration	Solution
The USB controller should only generate an interrupt when an event has occurred on USB. It should not generate an interrupt on every frame. Otherwise, the number of SMIs would be high and performance would suffer. Furthermore, other devices not used by the legacy software may have to be shut down if possible, because a large number of SMIs will occur.	Don't permit other USB devices to be active while in the legacy support mode.
A method must be developed to enable and disable the legacy support mode when entering and exiting a USB aware operating system.	UHCI’s HCD is aware of the legacy support. This is required for any other proposed scheme.
Can’t pulse the mouse CLK and data signals using the F0h command.	None, but should not be an issue.
Password protection can be done in the 8042 device.	SMM and the USB keyboard must handle this case.

Filename:	UHCI11D.DOC
Directory:	N:\PUBLSHED\USB\UHCI_DG\REV_11
Template:	C:\WORD60\TEMPLATE\NORMAL.DOT
Title:	TXC EDS
Subject:	
Author:	Jackie Wensel
Keywords:	
Comments:	
Creation Date:	05/17/96 10:00 AM
Revision Number:	3
Last Saved On:	05/17/96 10:15 AM
Last Saved By:	Jackie Wensel
Total Editing Time:	11 Minutes
Last Printed On:	05/17/96 10:21 AM
As of Last Complete Printing	
Number of Pages:	47
Number of Words:	17,277 (approx.)
Number of Characters:	98,483 (approx.)