USB

Pradeep Patel

Agenda

- Standards-USB1.1,USB2.0,USB3.0,Wireless USB
- Where USB after 2.0/Wireless USB. USB 3.0?
- System Arch- north bridge/south bridge. Where does USB fits in .
- USB System Description
- USB interconnect • USB devices • USB host • USB On the go
- USB Speeds High Speed - 480Mbits/s Full Speed - 12Mbits/s Low Speed - 1.5Mbits/s
- USB HOSTs -UHCI, OHCI ,EHCI
- Data Flow Types (USB Data Flow Model)
- Control Transfers:
  - Bulk Data Transfers:
  - Interrupt Data Transfers:

  Isochronous Data
- Transfers: Frames and Micro frames
- USB Protocol layers Device Descriptor Configuration Descriptors Interface Descriptors Endpoint Descriptors String Descriptors
- USB End points, Interface, Configuration
- Common Packet fields

Motivation

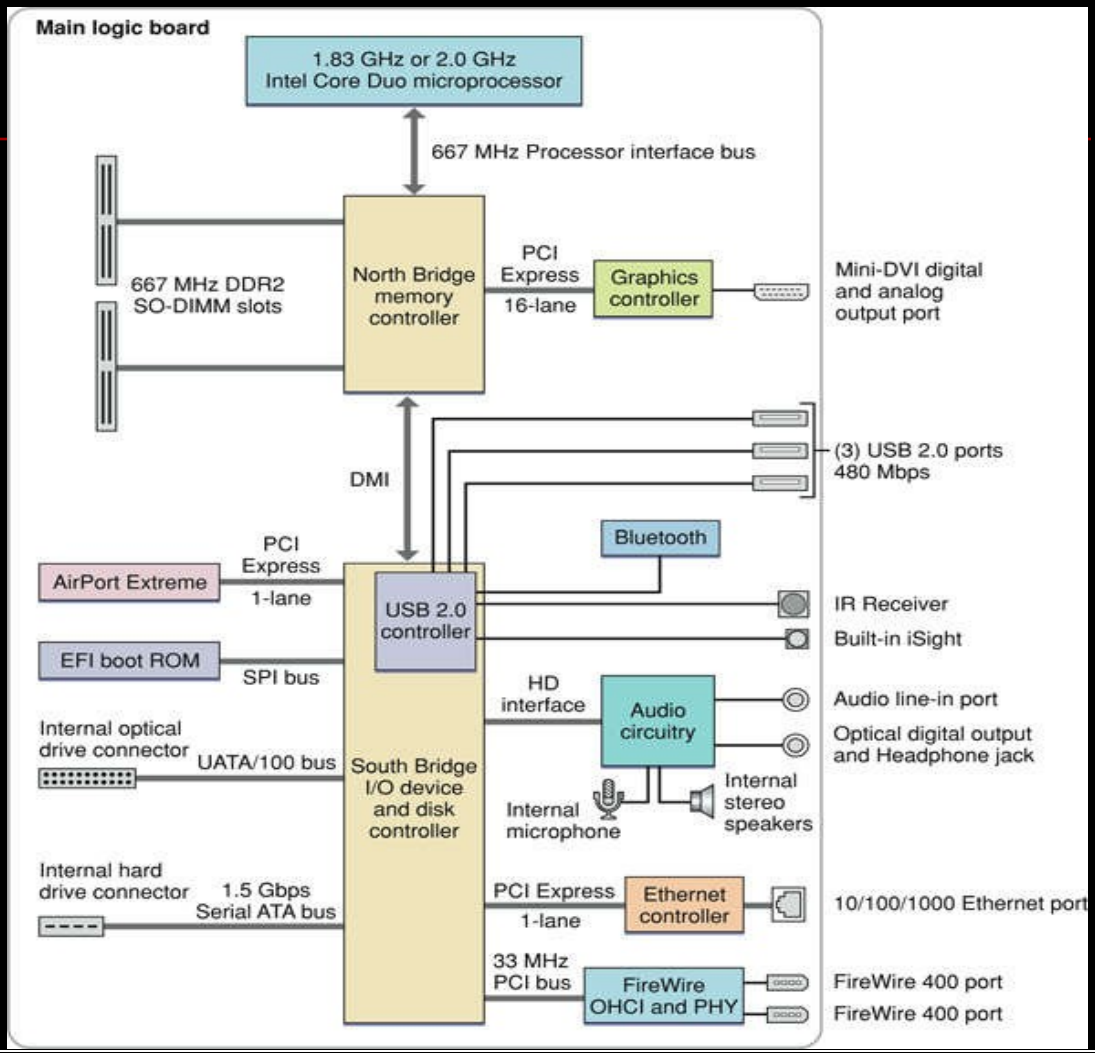
The original motivation for the Universal Serial Bus (USB) came from three interrelated considerations:

- **Inter connection of the Devices.**
- **Ease-of-use**
- **Port expansion**

Why USB

- The consumer market
- Simple and efficient way to communicate with many types of peripherals
- Consumer wish list:
  - Easy to use
  - Fast
  - Reliable
  - Flexible
  - Inexpensive
  - Power-conserving
  - OS support

System Arch- north bridge/south bridge



Various USB devices

- Audio devices
- Communication devices
- Human Interface Device (HID)
- Mass Storage
- Printer
- Imaging

Goals of USB2.0

- Ease-of-use for PC peripheral expansion
- Low-cost solution that supports transfer rates up to 480 Mb/s
- Full support for real-time data for voice, audio, and video
- Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging
- Integration in commodity device technology
- Comprehension of various PC configurations and form factors
- Provision of a standard interface capable of quick diffusion into product
- Enabling new classes of devices that augment the PC's capability
- Full backward compatibility of USB 2.0 for devices built to previous versions of the specification

USB Specification (USB2.0)

Document Organization

- Chapters 1 through 5 provide an overview for all readers, while Chapters 6 through 11 contain detailed
- technical information defining the USB.
- Peripheral implementers should particularly read Chapters 5 through 11.
- USB Host Controller implementers should particularly read Chapters 5 through 8, 10, and 11.
- USB device driver implementers should particularly read Chapters 5, 9, and 10.

The USB Concept (Functional Overview)

- USB is a *Peripheral bus* (PCI,ISA etc.)
  - *Simple* device interface (smart host)
  - Facilities for “register” ops, interrupts, wake, etc.
- Enables easy install & config.
  - PnP descriptors, many well-known device classes
- Data xfer rates / types supported:
  - 1.5, 12 and 480Mbps
  - Bulk and isochronous pipes
- Bus length <= 5m (can ext. to 25m w/ hubs)
  - Addressing for <= 127 devices per host controller

USB-- “Universal Serial Bus”

- Different speeds
  - High Speed – 480 Mb/s (USB 2.0)
  - Full Speed – 12 Mb/s (USB 1.1)
  - Low Speed – 1.5 Mb/s
- Some of the terms that you will come across
  - USB Host – Is the initiator, very intelligent
  - USB Device – Is the responder, not much intelligence

Introduction

- The Universal Serial Bus is host controlled.
- There can only be one host per bus.
- The specification in itself, does not support any form of multimaster arrangement.
- On-The-Go specification(OTG) which is a tack on standard to USB 2.0 has introduced a Host Negotiation Protocol which allows two devices negotiate for the role of host.
- USB OTG is aimed at and limited to single point to point connections such as a mobile phone and personal organiser and not multiple hub, multiple device desktop configuration.

Introduction….

- The USB host is responsible for undertaking all transactions and scheduling bandwidth.
- Data can be sent by various transaction methods using a token-based protocol.
- USB uses a tiered star topology, similar to that of 10BaseT Ethernet.
- Firstly power to each device can be monitored and even switched off if an over current condition occurs without disrupting other USB devices.
- Both high, full and low speed devices can be supported, with the hub filtering out high speed and full speed transactions so lower speed devices do not receive them

Introduction…

- Up to 127 devices can be connected to any one USB bus at any one given time.
- With USB 1.1, there were two Host Controller Interface Specifications, UHCI (Universal Host Controller Interface) developed by Intel which puts more of the burden on software (Microsoft) and allowing for cheaper hardware and the OHCI (Open Host Controller Interface) developed by Compaq, Microsoft and National Semiconductor which places more of the burden on hardware (Intel) and makes for simpler software.

Introduction…

- With the introduction of USB 2.0 a new Host Controller EHCI (Enhanced Host Controller Interface) was born.
- USB uses 4 shielded wires of which two are power (+5v & GND).
- The remaining two are twisted pair differential data signals.
- It uses a NRZI (Non Return to Zero Invert) encoding scheme to send data with a sync field to synchronise the host and receiver clocks.

Introduction

- USB supports plug'n'plug with dynamically loadable and unloadable drivers.
- The user simply plugs the device into the bus. The host will detect this addition, interrogate the newly inserted device and load the appropriate driver all in the time it takes the hourglass to blink on your screen provided a driver is installed for your device.
- Once the user is finished, User can simply lug the cable out, the host will detect its absence and automatically unload the driver.
- The loading of the appropriate driver is done using a PID/VID (Product ID/Vendor ID) combination.

Advantages

- Plug & Play
- The end user needs not worry about IRQs and port addresses, or rebooting the computer.

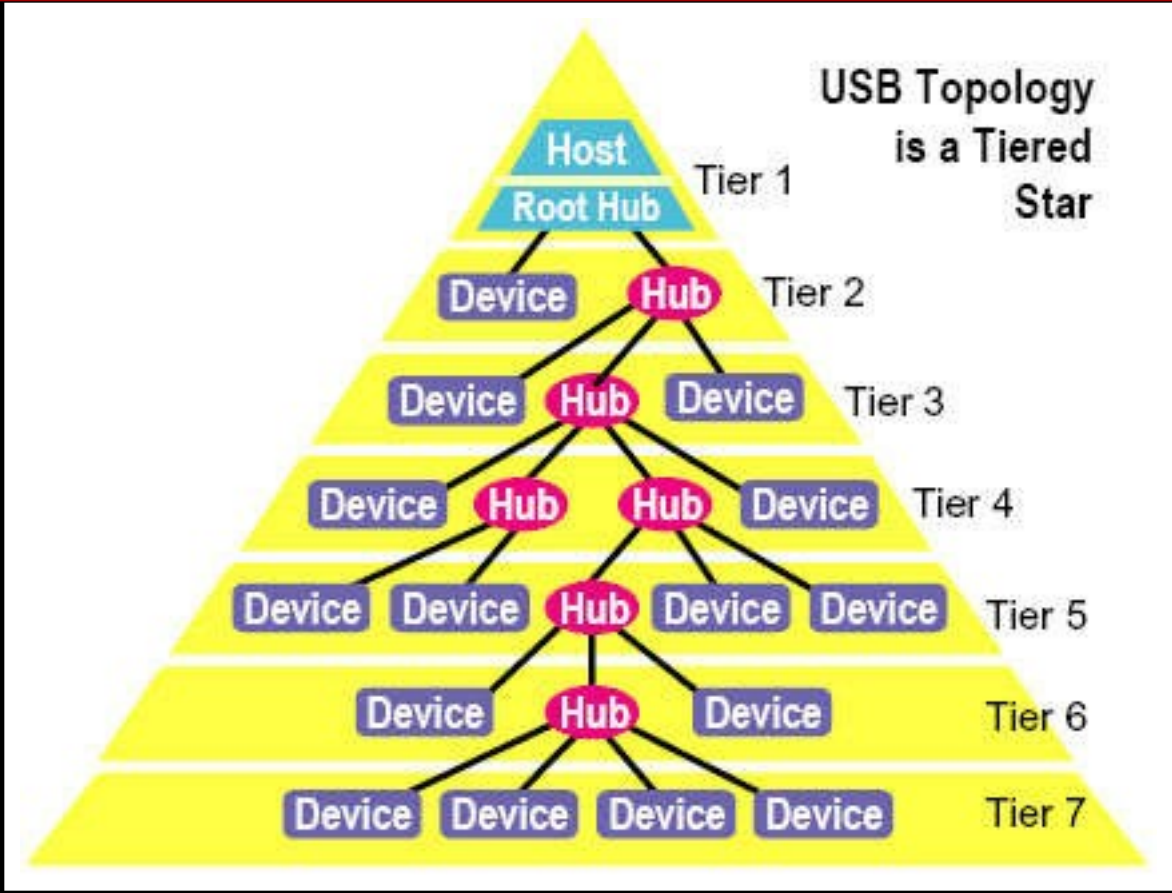
Transfer Types in USB

- Bandwidth of USB is allocated based on the criticality and the nature of the data transferred
- USB establishes a 125 micro-second (micro-frame) time base for High Speed
  - Full speed and Low speed has a millisecond (frame) time base
- Periodic Transfers
  - 80% micro-frame reservation and 90% frame reservation
  - Isochronous Transfer
    - Periodic and time critical communication
  - Interrupt Transfer
    - Periodic, Low-frequency, bounded latency communication

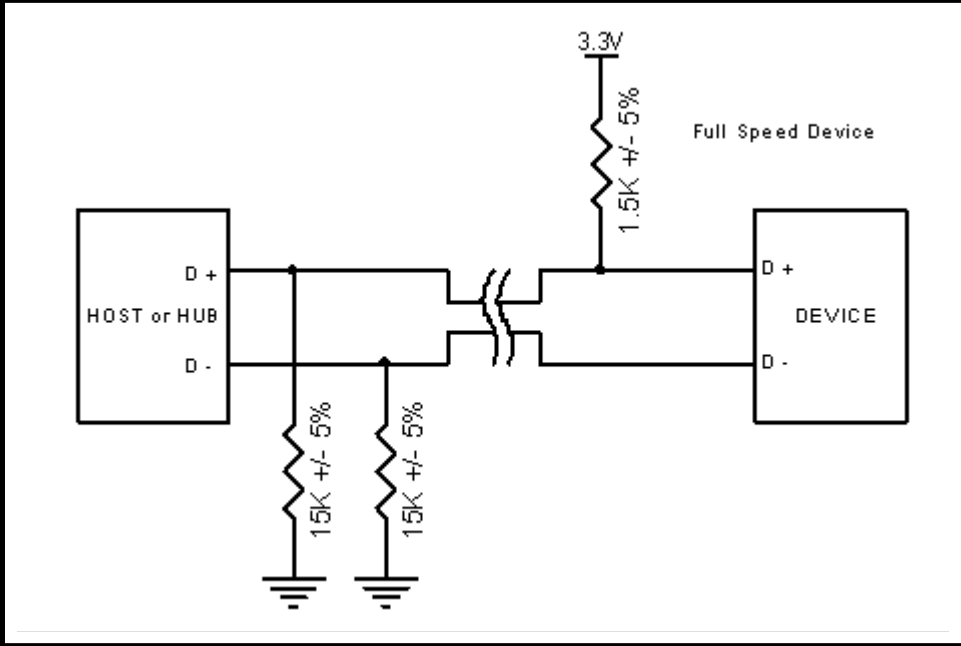
Transfer Types in USB (continued…)

- Non-Periodic Transfers
  - 20% micro-frame reservation and 10% frame reservation
  - Bulk Transfer
    - Non-periodic, large bursty transfer
  - Control Transfer
    - Bursty, non-periodic data used for command/status operations
    - Default Message pipe
      - Message pipes impose a structure on the communication flow

USB topology

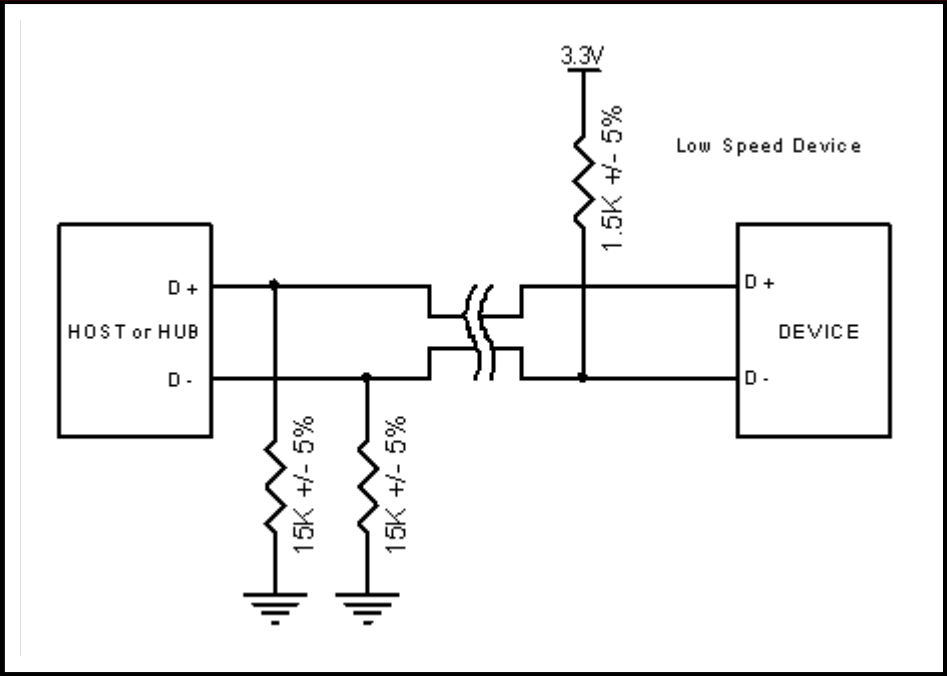


Speed Identification (FS)



Full Speed Device with pull up resistor connected to D+

Speed Identification (LS)



Low Speed Device with pull up resistor connected to D-

Speed Identification (HS)

- High speed devices will start by connecting as a full speed device (1.5k to 3.3V). Once it has been attached, it will do a high speed chirp during reset and establish a high speed connection if the hub supports it. If the device operates in high speed mode, then the pull up resistor is removed to balance the line.
- However a high speed device must not support low speed mode. It should only support full speed mode needed to connect first, then high speed mode if successfully negotiated later. A USB 2.0 compliant downstream facing device (Hub or Host) must support all three modes, high speed, full speed and low speed.

USB Protocol layer

- Packet
  - Token (OUT, IN, SOF, SETUP)
  - Data(DATA0, DATA1, DATA2, MDATA)
  - Handshake(ACK, NACK, STALL, NYET)
  - Special(PRE, ERR, SPLIT, PING)
- Transaction
  - Delivery of service to an endpoint. Each transaction consist of Consists of
    - token packet
    - optional data packet
    - optional handshake packet

Transaction (only token packet)

Packet	Dir	H	SOF	Frame #	CRC5	Pkt Len	Idle	Time Stamp
1	-->	S	0xA5	1250.?	0x16	14	124.767 µs	00000.0234 5127

Transaction (Token, data packet, Handshake)

Transaction	H	SETUP	ADDR	ENDP	T	D	Tp	R	bRequest	wValue	wIndex	wLength	ACK
0	S	0xB4	1	0	0	D->H	S	D	GET_DESCRIPTOR	DEVICE type	0x0000	18	0x4B

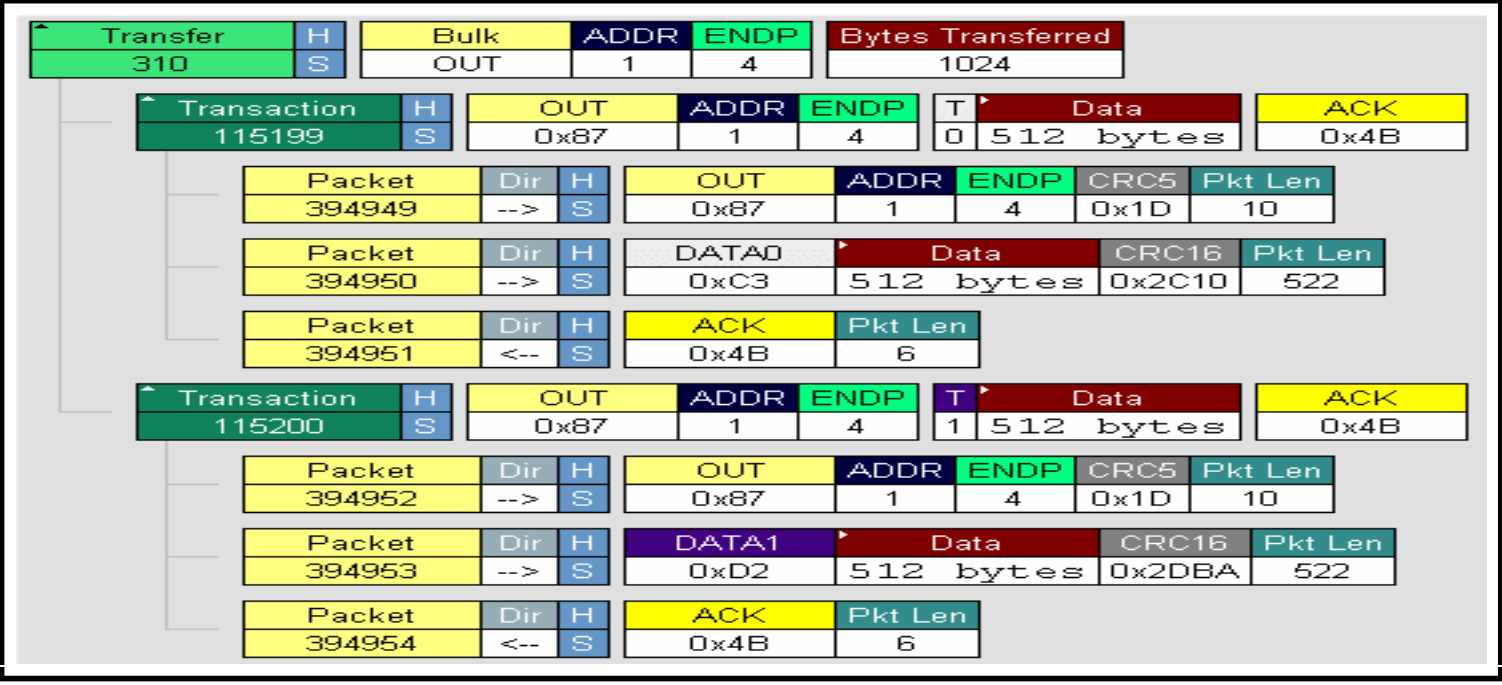
Packet	Dir	H	SETUP	ADDR	ENDP	CRC5	Pkt Len
20670	-->	S	0xB4	1	0	0x17	10

Packet	Dir	H	DATA0	Data	CRC16	Pkt Len
20671	-->	S	0xC3	8 bytes	0x072F	18

Packet	Dir	H	ACK	Pkt Len
20672	<--	S	0x4B	6

USB Protocol layer (continued…)

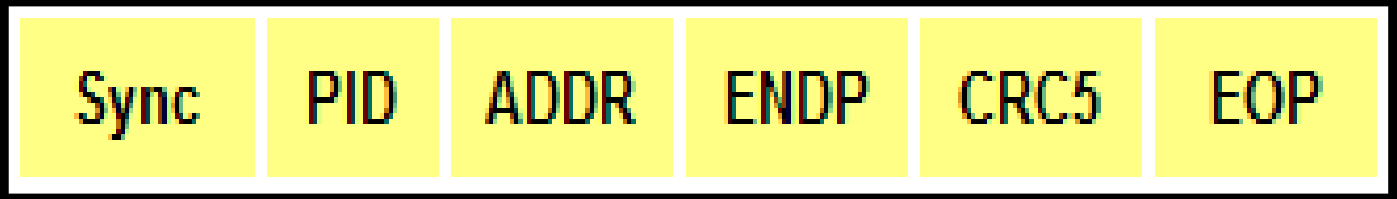
- Transfer
  - One or more transaction to move information between a software client and its function



Common USB Packet Fields

Data on the USBus is transmitted LSBit first. USB packets consist of the following fields-

- Sync-All packets must start with a sync field. The sync field is 8 bits long at low and full speed or 32 bits long for high speed and is used to synchronise the clock of the receiver with that of the transmitter. The last two bits indicate where the PID fields starts.
- **PID** stands for Packet ID. This field is used to identify the type of packet that is being sent. possible values.(Token, Data, Handshake, Special)



Common USB Packet Fields (Cont..)

- **ADDR** The address field specifies which device the packet is designated for. Being 7 bits in length allows for 127 devices to be supported. Address 0 is not valid, as any device which is not yet assigned an address must respond to packets sent to address zero.
- **ENDP** The endpoint field is made up of 4 bits, allowing 16 possible endpoints. Low speed devices, however can only have 2 additional endpoints on top of the default pipe. (4 endpoints max)
- **CRC** Cyclic Redundancy Checks are performed on the data within the packet payload. All token packets have a 5 bit CRC while data packets have a 16 bit CRC.
- **EOP** End of packet. Signalled by a Single Ended Zero (SE0) for approximately 2 bit times followed by a J for 1 bit time.

USB Packet Types

USB has four different packet types.

1) Token packets indicate the type of transaction to follow.
2) Data packets contain the payload.
3) Handshake packets are used for acknowledging data or reporting errors.
4) Special Type like -start of frame packets indicate the start of a new frame.

Token Packets

PID Type	PID Name	PID<3:0>*	Description
Token	OUT	0001B	Address + endpoint number in host-to-function transaction
	IN	1001B	Address + endpoint number in function-to-host transaction
	SOF	0101B	Start-of-Frame marker and frame number
	SETUP	1101B	Address + endpoint number in host-to-function transaction for SETUP to a control pipe

Packet	Dir	H
23113	-->	S

OUT	ADDR	ENDP	CRC5	Pkt Len
0x87	1	8	0x12	10

Packet	Dir	H
23244	-->	S

IN	ADDR	ENDP	CRC5	Pkt Len
0x96	1	9	0x1F	10

SOF	Frame #	CRC5	Pkt Len
0xA5	16.5	0x0F	14

Data Packets

PID Type	PID Name	PID<3:0>*	Description
Data	DATA0	0011B	Data packet PID even
	DATA1	1011B	Data packet PID odd
	DATA2	0111B	Data packet PID high-speed, high bandwidth isochronous transaction in a microframe (see Section 5.9.2 for more information)
	MDATA	1111B	Data packet PID high-speed for split and high bandwidth isochronous transactions (see Sections 5.9.2, 11.20, and 11.21 for more information)

Packet	Dir	H	DATA0	Data	CRC16	Pkt Len
23295	<--	S	0xC3	512 bytes	0xE9EC	520

Handshake Packets

PID Type	PID Name	PID<3:0>*	Description
Handshake	ACK	0010B	Receiver accepts error-free data packet
	NAK	1010B	Receiving device cannot accept data or transmitting device cannot send data
	STALL	1110B	Endpoint is halted or a control pipe request is not supported
	NYET	0110B	No response yet from receiver (see Sections 8.5.1 and 11.17-11.21)

Packet	Dir	H	ACK	Pkt Len
23296	-->	S	0x4B	6

Packet	Dir	H	NAK	Pkt Len
23404	<--	S	0x5A	8

Packet	Dir	H	NYET	Pkt Len
23102	<--	S	0x69	6

Special

Packet	Dir	H
23109	-->	S

PING	ADDR	ENDP	CRC5	Pkt Len
0x2D	1	8	0x12	10

Packet	Dir	H
23110	<--	S

NAK	Pkt Len
0x5A	6

Packet	Dir	H
23111	-->	S

PING	ADDR	ENDP	CRC5	Pkt Len
0x2D	1	8	0x12	10

Address

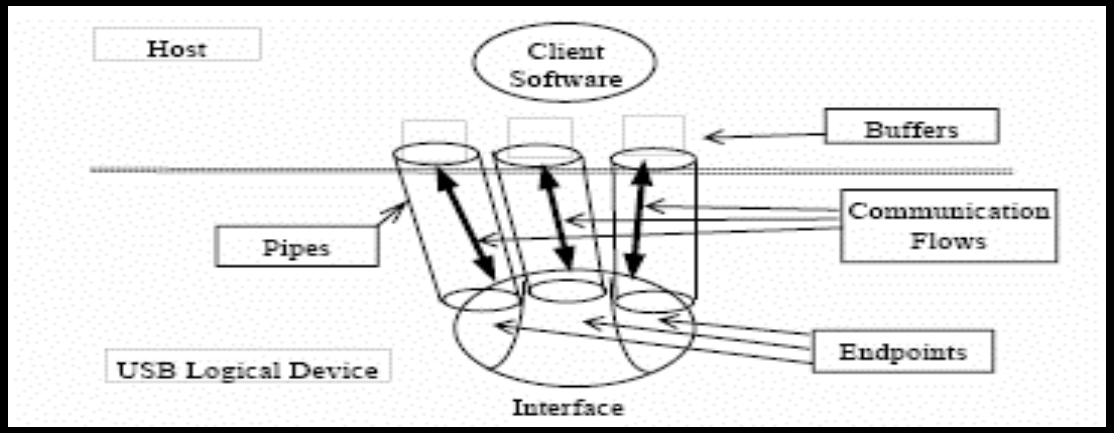
Packet	Dir	H
23112	<--	S

ACK	Pkt Len
0x4B	6

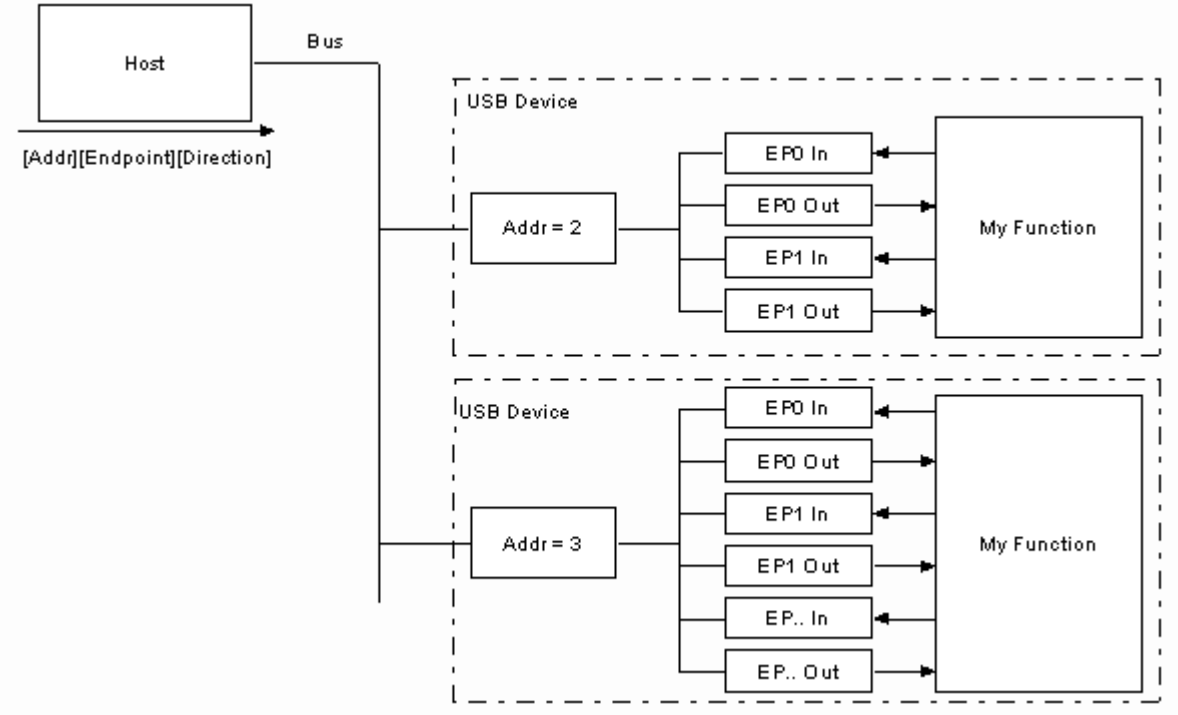
PID Type	PID Name	PID<3:0>*	Description
Special	PRE	1100B	(Token) Host-issued preamble. Enables downstream bus traffic to low-speed devices.
	ERR	1100B	(Handshake) Split Transaction Error Handshake (reuses PRE value)
	SPLIT	1000B	(Token) High-speed Split Transaction Token (see Section 8.4.2)
	PING	0100B	(Token) High-speed flow control probe for a bulk/control endpoint (see Section 8.5.1)
	Reserved	0000B	Reserved PID

Logical organization of USB

- Can be visualized as pipes within pipe
  - Main pipe – USB Cable
  - USB cable can be divided into logical pipes – Interfaces
  - Interfaces again divided into logical pipes – Endpoints
  - Endpoints will be memory buffers on Host and Device



Endpoints


Host
Bus
[Addr][Endpoint][Direction]
USB Device
Addr = 2
EP0 In
EP0 Out
EP1 In
EP1 Out
My Function
USB Device
Addr = 3
EP0 In
EP0 Out
EP1 In
EP1 Out
EP.. In
EP.. Out
My Function

Endpoints (Cont..)

- Endpoints can be described as sources or sinks of data. As the bus is host centric, endpoints occur at the end of the communications channel at the USB function. At the software layer, your device driver may send a packet to your devices EP1 for example. As the data is flowing out from the host, it will end up in the EP1 OUT buffer.
- Your firmware will then at its leisure read this data. If it wants to return data, the function cannot simply write to the bus as the bus is controlled by the host.

- Therefore it writes data to EP1 IN which sits in the buffer until such time when the host sends a IN packet to that endpoint requesting the data. Endpoints can also be seen as the interface between the hardware of the function device and the firmware running on the function device.
- All devices must support endpoint zero. This is the endpoint which receives all of the devices control and status requests during enumeration and throughout the duration while the device is operational on the bus.

- The user plugs a device into a USB port
- The host learns of the new device
- The host learns about the device's capabilities
- The host assigns and loads a device driver
- The application level communication starts

Endpoint Types

The Universal Serial Bus specification defines four transfer/endpoint types,

- **Control Transfers**
- **Interrupt Transfers**
- **Isochronous Transfers**
- **Bulk Transfers**

Control Transfers

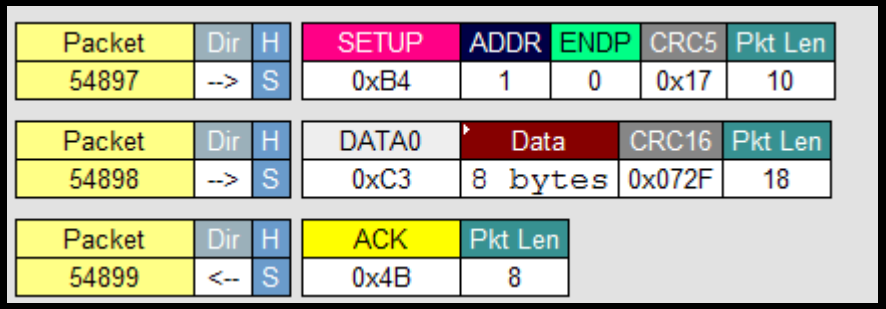
- Control transfers are typically used for command and status operations.
- They are essential to set up a USB device with all enumeration functions being performed using control transfers.
- The packet length of control transfers in low speed devices must be 8 bytes, high speed devices allow a packet size of 8, 16, 32 or 64 bytes and full speed devices must have a packet size of 64 bytes.

A control transfer can have up to three stages.

- **Setup Stage**
- **Data Stage**
- **Status Stage**

Setup Stage

The **Setup Stage** is where the request is sent. This consists of three packets. The setup token is sent first which contains the address and endpoint number. The data packet is sent next and always has a PID type of data0 and includes a setup packet which details the type of request. The last packet is a handshake used for acknowledging successful receipt or to indicate an error. If the function successfully receives the setup data (CRC and PID etc OK) it responds with ACK, otherwise it ignores the data and doesn't send a handshake packet. Functions cannot issue a STALL or NAK packet in response to a setup packet



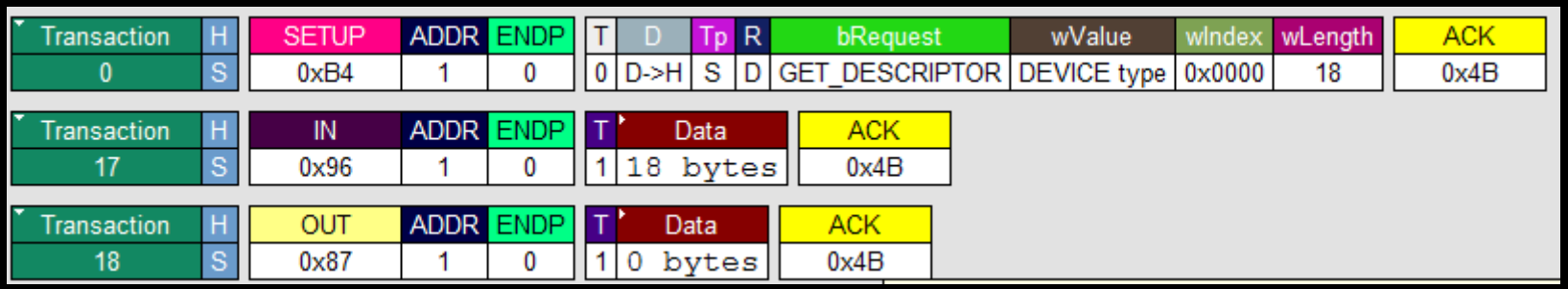
Packet	Dir	H	SETUP	ADDR	ENDP	CRC5	Pkt Len
54897	-->	S	0xB4	1	0	0x17	10

Packet	Dir	H	DATA0	Data	CRC16	Pkt Len
54898	-->	S	0xC3	8 bytes	0x072F	18

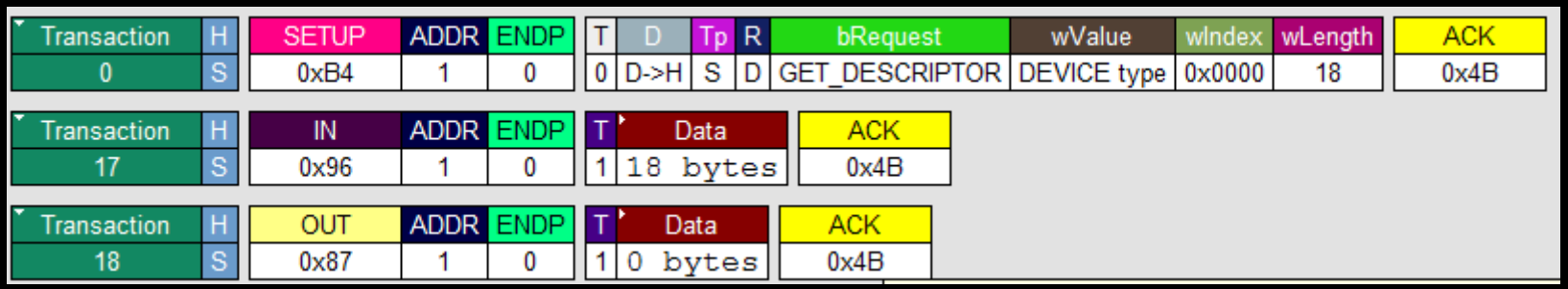
Packet	Dir	H	ACK	Pkt Len
54899	<--	S	0x4B	8

Data Stage

- The optional **Data Stage** consists of one or multiple IN or OUT transfers. The setup request indicates the amount of data to be transmitted in this stage. If it exceeds the maximum packet size, data will be sent in multiple transfers each being the maximum packet length except for the last packet.
- The data stage has two different scenarios depending upon the direction of data transfer- IN or OUT



Status Stage



Transaction	H	SETUP	ADDR	ENDP	T	D	Tp	R	bRequest	wValue	wIndex	wLength	ACK
0	S	0xB4	1	0	0	D->H	S	D	GET_DESCRIPTOR	DEVICE type	0x0000	18	0x4B

Transaction	H	IN	ADDR	ENDP	T	Data	ACK
17	S	0x96	1	0	1	18 bytes	0x4B

Transaction	H	OUT	ADDR	ENDP	T	Data	ACK
18	S	0x87	1	0	1	0 bytes	0x4B

Interrupt Transfers

- USB device is polled (Can't interrupt)
- USB if a device requires the attention of the host, it must wait until the host polls it before it can report that it needs urgent attention!

Interrupt Transfers

- Guaranteed Latency
- Stream Pipe - Unidirectional
- Error detection and next period retry
- The maximum data payload size for low-speed devices is 8 bytes.
- Maximum data payload size for full-speed devices is 64 bytes.
- Maximum data payload size for high-speed devices is 1024 bytes.

Isochronous Transfers

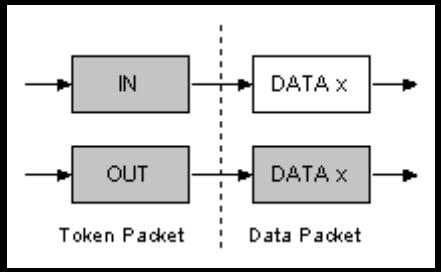
- Isochronous transfers occur continuously and periodically. They typically contain time sensitive information, such as an audio or video stream. If there were a delay or retry of data in an audio stream, then you would expect some erratic audio containing glitches. The beat may no longer be in sync. However if a packet or frame was dropped every now and again, it is less likely to be noticed by the listener.

- **Isochronous Transfers provide**
  - Guaranteed access to USB bandwidth.
  - Bounded latency.
  - Error detection via CRC, but no retry or guarantee of delivery.
  - Full & high speed modes only.

- The maximum size data payload is specified in the endpoint descriptor of an Isochronous Endpoint.

Isochronous Transfers (Cont..)

- This can be up to a maximum of 1023 bytes for a full speed device and 1024 bytes for a high speed device.
- As the maximum data payload size is going to effect the bandwidth requirements of the bus, it is wise to specify a conservative payload size.



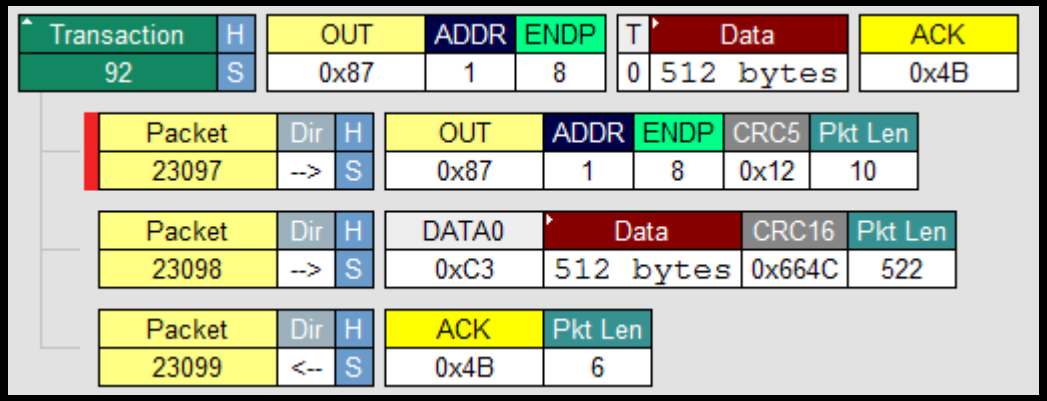
Bulk Transfers

- Bulk transfers can be used for large bursty data. Such examples could include a print-job sent to a printer or an image generated from a scanner. Bulk transfers provide error correction in the form of a CRC16 field on the data payload and error detection/re-transmission mechanisms ensuring data is transmitted and received without error.
- For full speed endpoints, the maximum bulk packet size is either 8, 16, 32 or 64 bytes long.
- For high speed endpoints, the maximum packet size can be up to 512 bytes long.
- If the data payload falls short of the maximum packet size, it doesn't need to be padded with zeros.
- A bulk transfer is considered complete when it has transferred the exact amount of data requested, transferred a packet less than the maximum endpoint size of transferred a zero-length packet.

Bulk Transfers

Bulk Transfers

- Used to transfer large bursty data.
- Error detection via CRC, with guarantee of delivery.
- No guarantee of bandwidth or minimum latency.
- Full & high speed modes only.



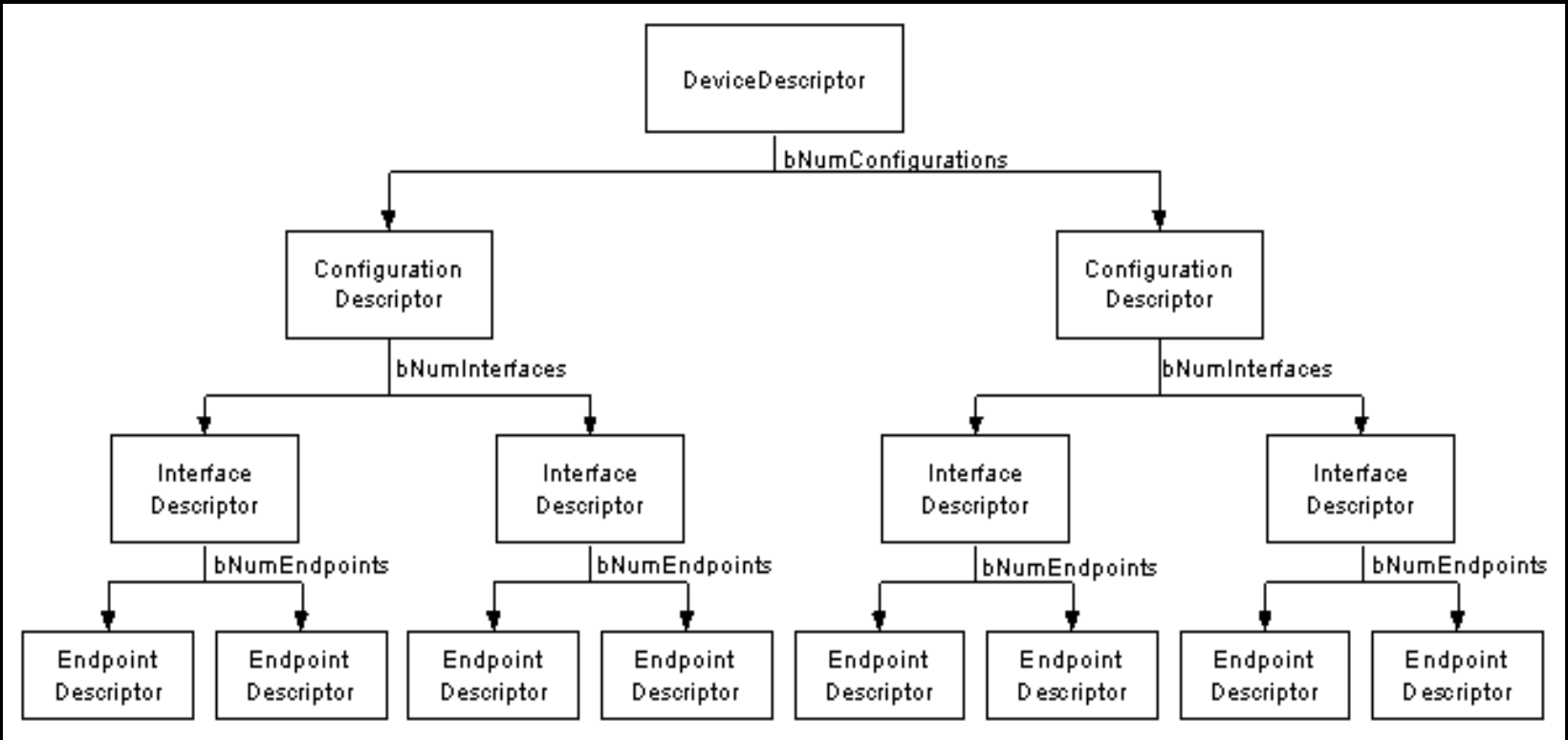
Bandwidth Management

- The host is responsible in managing the bandwidth of the bus. This is done at enumeration when configuring Isochronous and Interrupt Endpoints and throughout the operation of the bus. The specification places limits on the bus, allowing no more than 90% of any frame to be allocated for periodic transfers (Interrupt and Isochronous) on a full speed bus. On high speed buses this limitation gets reduced to no more than 80% of a microframe can be allocated for periodic transfers.
- So you can quite quickly see that if you have a highly saturated bus with periodic transfers, the remaining 10% is left for control transfers and once those have been allocated, bulk transfers will get its slice of what is left

USB Descriptors

- All USB devices have a hierarchy of descriptors which describe to the host information such as what the device is, who makes it, what version of USB it supports, how many ways it can be configured, the number of endpoints and their types etc
- Device (Printer) advertises its capabilities to the Host (PC) through a series of data exchanges. This process is called enumeration
- Several Descriptors are exchanged during enumeration process which includes
  - **Device Descriptors**
  - **Configuration Descriptors**
  - **Interface Descriptors**
  - **Endpoint Descriptors**
  - **String Descriptors**

USB Descriptors (Cont..)



Composition of USB Descriptors

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptionType	1	Constant	DescriptorType
2	...	n		Start of parameters for descriptor

Device Descriptors (Cont..)

- USB devices can only have one device descriptor.
- The number of configurations indicate how many configuration descriptors branches are to follow.
- Has description about the device, like vendor ID, product ID. Based on this information the drivers for the device are loaded
- Has description about the number of configuration supported and the USB spec to which the Device is complaint with.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of the Descriptor in Bytes (18 bytes)
1	bDescriptorType	1	Constant	Device Descriptor (0x01)
2	bcdUSB	2	BCD	USB Specification Number which device complies too.
4	bDeviceClass	1	Class	Class Code (Assigned by USB Org) If equal to Zero, each interface specifies it's own class code If equal to 0xFF, the class code is vendor specified. Otherwise field is valid Class Code.
5	bDeviceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
6	bDeviceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
7	bMaxPacketSize	1	Number	Maximum Packet Size for Zero Endpoint. Valid Sizes are 8, 16, 32, 64

Offset	Field	Size	Value	Description
8	idVendor	2	ID	Vendor ID (Assigned by USB Org)
10	idProduct	2	ID	Product ID (Assigned by Manufacturer)
12	bcdDevice	2	BCD	Device Release Number
14	iManufacturer	1	Index	Index of Manufacturer String Descriptor
15	iProduct	1	Index	Index of Product String Descriptor
16	iSerialNumber	1	Index	Index of Serial Number String Descriptor
17	bNumConfigurations	1	Integer	Number of Possible Configurations

Configuration Descriptors

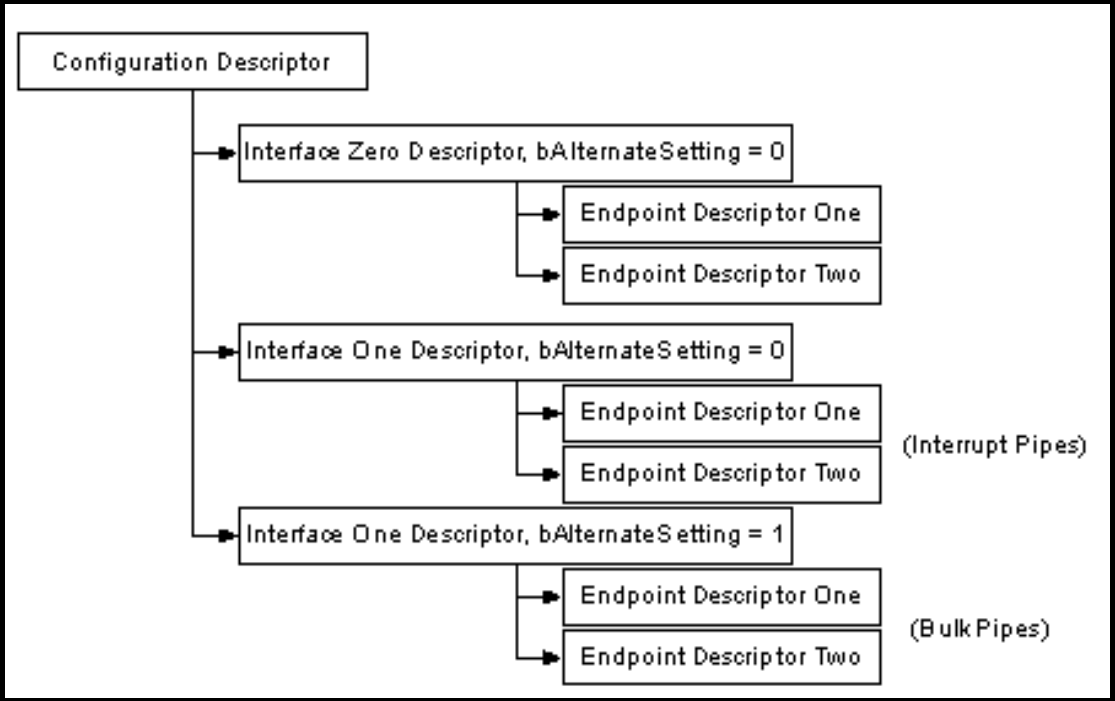
- The configuration descriptor specifies values such as the amount of power this particular configuration uses, if the device is self or bus powered and the number of interfaces it has. When a device is enumerated, the host reads the device descriptors and can make a decision of which configuration to enable. It can only enable one configuration at a time.
- For example, It is possible to have a high power bus powered configuration and a self powered configuration. If the device is plugged into a host with a mains power supply, the device driver may choose to enable the high power bus powered configuration enabling the device to be powered without a connection to the mains, yet if it is connected to a laptop or personal organiser it could enable the 2nd configuration (self powered) requiring the user to plug your device into the power point.
- The configuration settings are not limited to power differences. Each configuration could be powered in the same way and draw the same current, yet have different interface or endpoint combinations. However it should be noted that changing the configuration requires all activity on each endpoint to stop. While USB offers this flexibility, very few devices have more than 1 configuration.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	Configuration Descriptor (0x02)
2	wTotalLength	2	Number	Total length in bytes of data returned
4	bNumInterfaces	1	Number	Number of Interfaces
5	bConfigurationValue	1	Number	Value to use as an argument to select this configuration
6	iConfiguration	1	Index	Index of String Descriptor describing this configuration
7	bmAttributes	1	Bitmap	D7 Reserved, set to 1. (USB 1.0 Bus Powered) D6 Self Powered D5 Remote Wakeup D4..0 Reserved, set to 0.
8	bMaxPower	1	mA	Maximum Power Consumption in 2mA units

Interface Descriptors

- The interface descriptor could be seen as a header or grouping of the endpoints into a functional group performing a single feature of the device.
- For example you could have a multi-function fax/scanner/printer device. Interface descriptor one could describe the endpoints of the fax function, Interface descriptor two the scanner function and Interface descriptor three the printer function.
- Unlike the configuration descriptor, there is no limitation as to having only one interface enabled at a time. A device could have 1 or many interface descriptors enabled at once.

Interface Descriptors



Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (9 Bytes)
1	bDescriptorType	1	Constant	Interface Descriptor (0x04)
2	bInterfaceNumber	1	Number	Number of Interface
3	bAlternateSetting	1	Number	Value used to select alternative setting
4	bNumEndpoints	1	Number	Number of Endpoints used for this interface
5	bInterfaceClass	1	Class	Class Code (Assigned by USB Org)
6	bInterfaceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
7	bInterfaceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
8	iInterface	1	Index	Index of String Descriptor Describing this interface

Endpoint Descriptors

- Each endpoint descriptor is used to specify the type of transfer, direction, polling interval and maximum packet size for each endpoint.
- Endpoint zero, the default control endpoint is always assumed to be control endpoint and as such never has a descriptor.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (7 bytes)
1	bDescriptorType	1	Constant	Endpoint Descriptor (0x05)
2	bEndpointAddress	1	Endpoint	Endpoint Address Bits 0..3b Endpoint Number. Bits 4..6b Reserved. Set to Zero Bits 7 Direction 0 = Out, 1 = In (Ignored for Control Endpoints)
3	bmAttributes	1	Bitmap	Bits 0..1 Transfer Type 00 = Control 01 = Isochronous 10 = Bulk 11 = Interrupt Bits 2..7 are reserved. If Isochronous endpoint, Bits 3..2 = Synchronisation Type (Iso Mode) 00 = No Synchonisation 01 = Asynchronous 10 = Adaptive 11 = Synchronous Bits 5..4 = Usage Type (Iso Mode)

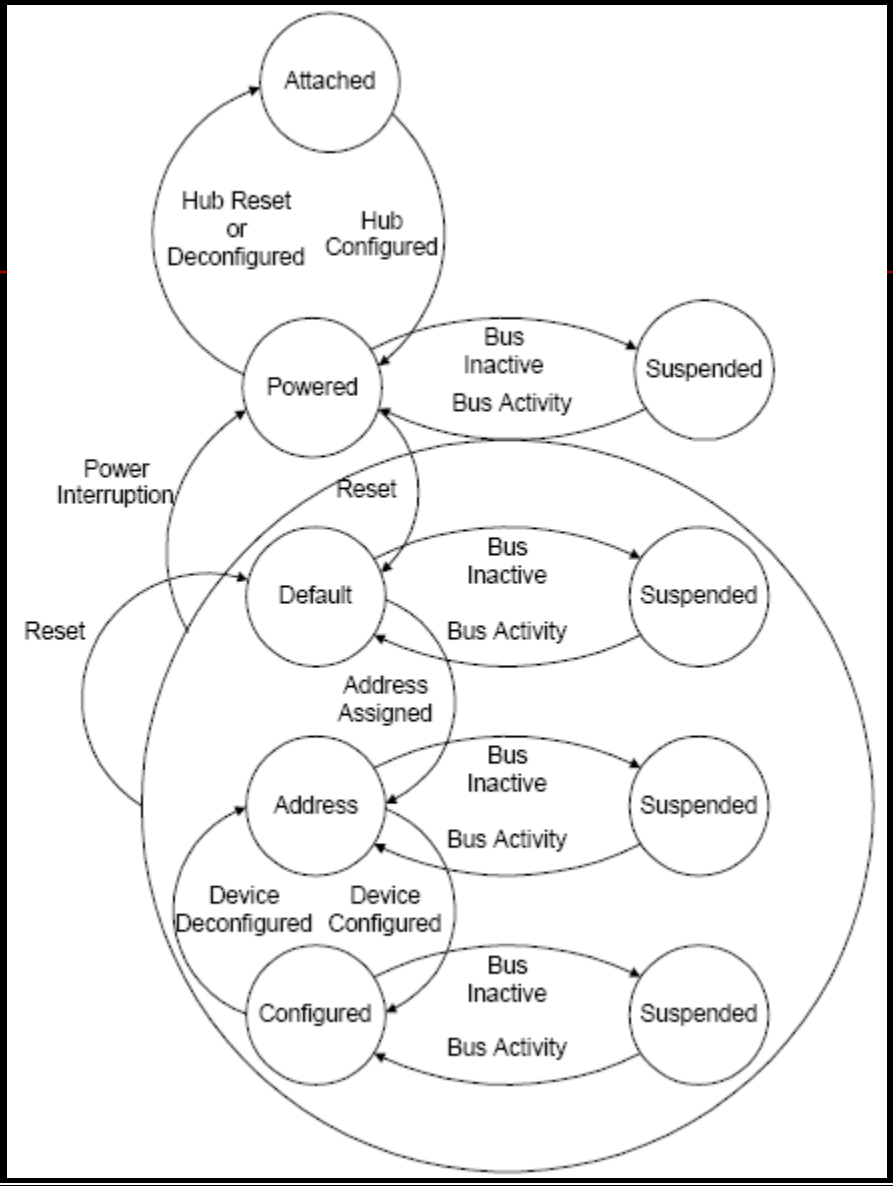
Offset	Field	Size	Value	Description
4	wMaxPacketSize	2	Number	Maximum Packet Size this endpoint is capable of sending or receiving
6	bInterval	1	Number	Interval for polling endpoint data transfers. Value in frame counts. Ignored for Bulk & Control Endpoints. Isochronous must equal 1 and field may range from 1 to 255 for interrupt endpoints.

String Descriptors

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	String Descriptor (0x03)
2	wLANGID[0]	2	number	Supported Language Code Zero (e.g. 0x0409 English - United States)
4	wLANGID[1]	2	number	Supported Language Code One (e.g. 0x0c09 English - Australian)
n	wLANGID[x]	2	number	Supported Language Code x (e.g. 0x0407 German - Standard)

All subsequent strings take on the format below

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	String Descriptor (0x03)
2	bString	n	Unicode	Unicode Encoded String


Attached
Hub Reset or Deconfigured
Hub Configured
Powered
Bus Inactive
Bus Activity
Suspended
Power Interruption
Reset
Default
Bus Inactive
Bus Activity
Suspended
Reset
Address Assigned
Address
Bus Inactive
Bus Activity
Suspended
Device Deconfigured
Device Configured
Configured
Bus Inactive
Bus Activity
Suspended

Attached	Powered	Default	Address	Configured	Suspended	State
No	--	--	--	--	--	Device is not attached to the USB. Other attributes are not significant.
Yes	No	--	--	--	--	Device is attached to the USB, but is not powered. Other attributes are not significant.
Yes	Yes	No	--	--	--	Device is attached to the USB and powered, but has not been reset.
Yes	Yes	Yes	No	--	--	Device is attached to the USB and powered and has been reset, but has not been assigned a unique address. Device responds at the default address.
Yes	Yes	Yes	Yes	No	--	Device is attached to the USB, powered, has been reset, and a unique device address has been assigned. Device is not configured.
Yes	Yes	Yes	Yes	Yes	No	Device is attached to the USB, powered, has been reset, has a unique address, is configured, and is not suspended. The host may now use the function provided by the device.
Yes	Yes	--	--	--	Yes	Device is, at minimum, attached to the USB and is powered and has not seen bus activity for 3 ms. It may also have a unique address and be configured for use. However, because the device is suspended, the host may not use the device's function.

Bus Enumeration

- When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached to a powered port, the following actions are taken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe. At this point, the USB device is in the Powered state and the port to which it is attached is disabled.

2. The host determines the exact nature of the change by querying the hub.

3. Now that the host knows the port to which the new device has been attached, the host then waits for at least 100 ms to allow completion of an insertion process and for power at the device to become stable. The host then issues a port enable and reset command to that port.

4. The hub performs the required reset processing for that port. When the reset signal is released, the port has been enabled. The USB device is now in the Default state and can draw no more than 100 mA from VBUS. All of its registers and state have been reset and it answers to the default address.

Bus Enumeration (Cont..)

5. The host assigns a unique address to the USB device, moving the device to the Address state.

6. Before the USB device receives a unique address, its Default Control Pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.

7. The host reads the configuration information from the device by reading each configuration zero to*n*-1, where *n* is the number of configurations. This process may take several milliseconds to complete.

8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the Configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of VBUS power described in its descriptor for the selected configuration. From the device's point of view, it is now ready for use.

The Setup Packet

-Every USB device must respond to setup packets on the default pipe.

-The setup packets are used –

- Detection and configuration of the device
- Carry out common functions such as setting the USB device's address, requesting a device descriptor or checking the status of a endpoint.

The Setup Packet

Offset	Field	Size	Value	Description
0	bmRequestType	1	Bit-Map	**D7 Data Phase Transfer Direction** 0 = Host to Device 1 = Device to Host **D6..5 Type** 0 = Standard 1 = Class 2 = Vendor 3 = Reserved **D4..0 Recipient** 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4..31 = Reserved
1	bRequest	1	Value	Request
2	wValue	2	Value	Value
4	wIndex	2	Index or Offset	Index
6	wLength	2	Count	Number of bytes to transfer if there is a data phase

Standard Device Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1000 0000b	GET_STATUS (0x00)	Zero	Zero	Two	Device Status
0000 0000b	CLEAR_FEATURE (0x01)	Feature Selector	Zero	Zero	None
0000 0000b	SET_FEATURE (0x03)	Feature Selector	Zero	Zero	None
0000 0000b	SET_ADDRESS (0x05)	Device Address	Zero	Zero	None
1000 0000b	GET_DESCRIPTOR (0x06)	Descriptor Type & Index	Zero or Language ID	Descriptor Length	Descriptor
0000 0000b	SET_DESCRIPTOR (0x07)	Descriptor Type & Index	Zero or Language ID	Descriptor Length	Descriptor
1000 0000b	GET_CONFIGURATION (0x08)	Zero	Zero	1	Configuration Value
0000 0000b	SET_CONFIGURATION (0x09)	Configuration Value	Zero	Zero	None

Standard Interface Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1000 0001b	GET_STATUS (0x00)	Zero	Interface	Two	Interface Status
0000 0001b	CLEAR_FEATURE (0x01)	Feature Selector	Interface	Zero	None
0000 0001b	SET_FEATURE (0x03)	Feature Selector	Interface	Zero	None
1000 0001b	GET_INTERFACE (0x0A)	Zero	Interface	One	Alternate Interface
0000 0001b	SET_INTERFACE (0x11)	Alternative Setting	Interface	Zero	None

Standard Endpoint Requests

bmRequestType	bRequest	wValue	Windex	wLength	Data
1000 0010b	GET_STATUS (0x00)	Zero	Endpoint	Two	Endpoint Status
0000 0010b	CLEAR_FEATURE (0x01)	Feature Selector	Endpoint	Zero	None
0000 0010b	SET_FEATURE (0x03)	Feature Selector	Endpoint	Zero	None
1000 0010b	SYNCH_FRAME (0x12)	Zero	Endpoint	Two	FrameNumber

USB in Thermal Inkjet printers

- USB Device
  - All Thermal Inkjet (TIJ) printers has USB Device support
  - TIJ printer ASIC has a block for USB Device
  - USB Device driver for controlling the USB Device block
- USB Host
  - Some TIJ printers has USB Host support
  - Supported TIJ printer ASIC has a block for USB Host
  - USB Host driver for controlling the USB Host block along the USB Host stack

Links to USB information

- USB 2.0 Spec

- USB org
  - http://www.usb.org/home

Format of Setup Data

Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bitmap	Characteristics of request: D7: Data transfer direction 0 = Host-to-device 1 = Device-to-host D6...5: Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4...0: Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4...31 = Reserved
1	*bRequest*	1	Value	Specific request (refer to Table 9-3)
2	*wValue*	2	Value	Word-sized field that varies according to request
4	*wIndex*	2	Index or Offset	Word-sized field that varies according to request; typically used to pass an index or offset
6	*wLength*	2	Count	Number of bytes to transfer if there is a Data stage

Standard Device Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration Value
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Interface
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None
00000000B	SET_CONFIGURATION	Configuration Value	Zero	Zero	None
00000000B	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
00000000B 00000001B 00000010B	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

USB H/w & S/f

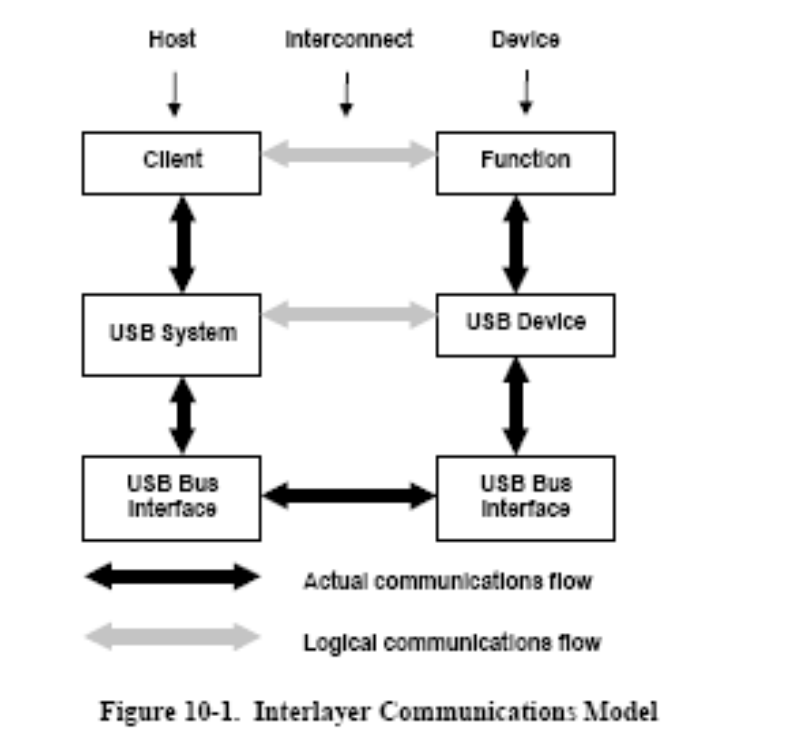
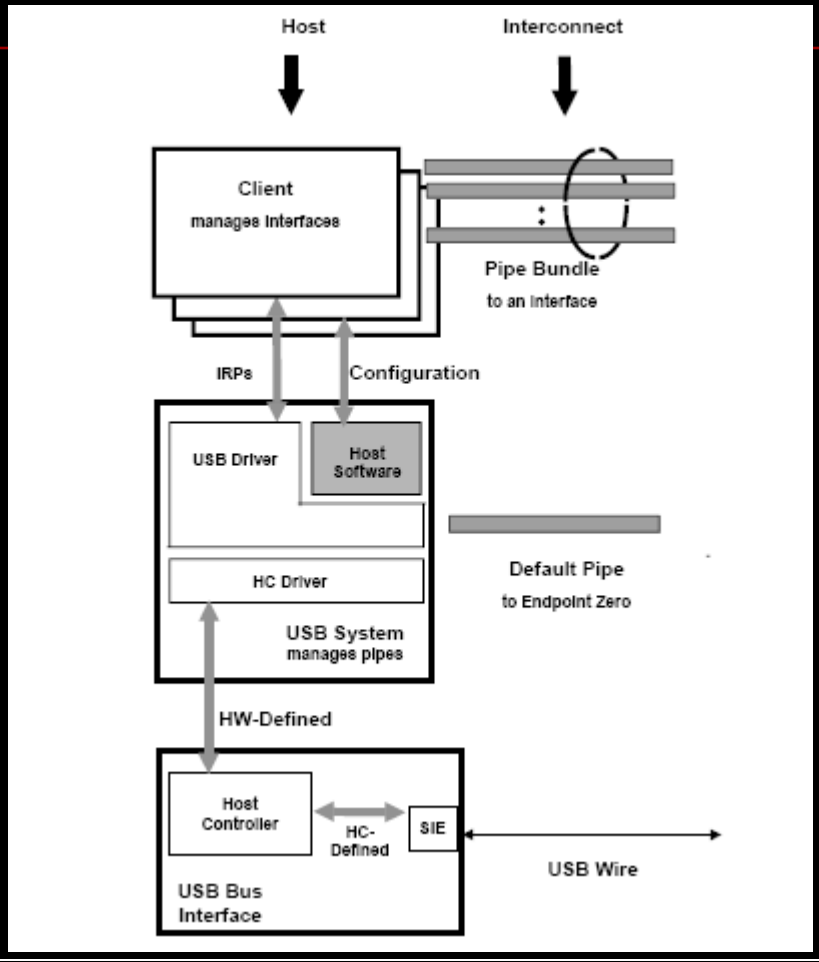


Figure 10-1. Interlayer Communications Model

USB H/w & S/f


Host
Interconnect
Client
manages Interfaces
Pipe Bundle
to an Interface
IRPs
Configuration
USB Driver
Host Software
HC Driver
USB System
manages pipes
Default Pipe
to Endpoint Zero
HW-Defined
Host Controller
HC-Defined
SIE
USB Wire
USB Bus Interface

Host Controller Requirements

- **State Handling -** As a component of the host, the Host Controller reports and manages its states.
- **Serializer/Deserializer -**For data transmitted from the host, the Host Controller converts protocol and data information from its native format to a bit stream transmitted on the USB. For data being received into the host, the reverse operation is performed.
- **(micro) frame Generation -** The Host Controller produces SOF tokens at a period of 1 ms when operating with full-speed devices, and at a period of 125 µs when operating with high-speed devices.
- Controller to one or more USB ports.
- **Host System Interface** Provides a high-speed data path between the Host Controller and host system.

Host Controller Requirements …

- **Data Processing -** The Host Controller processes requests for data transmission to and from the host.
- **Protocol Engine -** The Host Controller supports the protocol specified by the USB.
- **Transmission Error Handling** All Host Controllers exhibit the same behavior when detecting and reacting to the defined error categories.
- **Remote Wakeup** All Host Controllers must have the ability to place the bus into the Suspended state and to respond to bus wakeup events.
- **Root Hub** The root hub provides standard hub function to link the Host

Hub

Supporting many of the attributes that make USB user friendly and hide its complexity from the user. Listed below are the major aspects of USB functionality that hubs must support:

- Connectivity behaviour
- Power management
- Device connect/disconnect detection
- Bus fault detection and recovery
- High, full and low-speed device support

Enumeration

Key	Comment
idVendor & idProduct & bcdDevice	bcdDevice is the device's release number. This option would most likely be a Vendor Specific Class Driver.
idVendor & idProduct	This option would most likely be a Vendor Specific Class Driver
idVendor & bDeviceSubClass & bDeviceProtocol	Only if bDeviceClass is FFH. This option would most likely be a Vendor Specific Class Driver
idVendor & bDeviceSubClass	This option would most likely be a Vendor Specific Class Driver.
bDeviceClass & bDeviceSubClass & bDeviceProtocol	Only if bDeviceClass is not FFH. In this case, the Class Driver is most likely a standard class specification defined by the USB-IF.
bDeviceClass & bDeviceSubClass Only if	Only if bDeviceClass is not FFH. In this case, the Class Driver is most likely a standard class specification defined by the USB-IF.

Enumeration

If no drivers are found from the above search, then system software is expected to choose an appropriate configuration for the USB device and then try to locate/load drivers for each interface in the chosen configuration. Keys for this driver search are based on information from both the 'device' and 'interface' descriptors. The table below shows the search keys in priority order.

Key	Comment
idVendor & idProduct & bcdDevice & bConfigurationValue & bInterfaceNumber	THIS OPTION WOULD MOST LIKELY BE A VENDOR SPECIFIC CLASS DRIVER
idVendor & idProduct & bConfigurationValue & bInterfaceNumber	This option would most likely be a Vendor Specific Class Driver
idVendor & bInterfaceSubClass & bInterfaceProtocol	Only if bInterfaceClass is FFH. THIS OPTION WOULD MOST LIKELY BE A VENDOR SPECIFIC CLASS DRIVER
idVendor & bInterfaceSubClass	Only if bInterfaceClass is FFH. THIS OPTION WOULD MOST LIKELY BE A VENDOR SPECIFIC CLASS DRIVER
bInterfaceClass & bInterfaceSubClass & bInterfaceProtocol	ONLY IF BINTERFACECLASS IS NOT FFH. IN THIS CASE, THE CLASS DRIVER IS MOST LIKELY A STANDARD CLASS SPECIFICATION DEFINED BY THE USB-IF.
bInterfaceClass & bInterfaceSubClass	Only if bInterfaceClass is not FFH. In this case, the Class Driver is most likely a standard class specification defined by the USB-IF.

UHCI Frame list

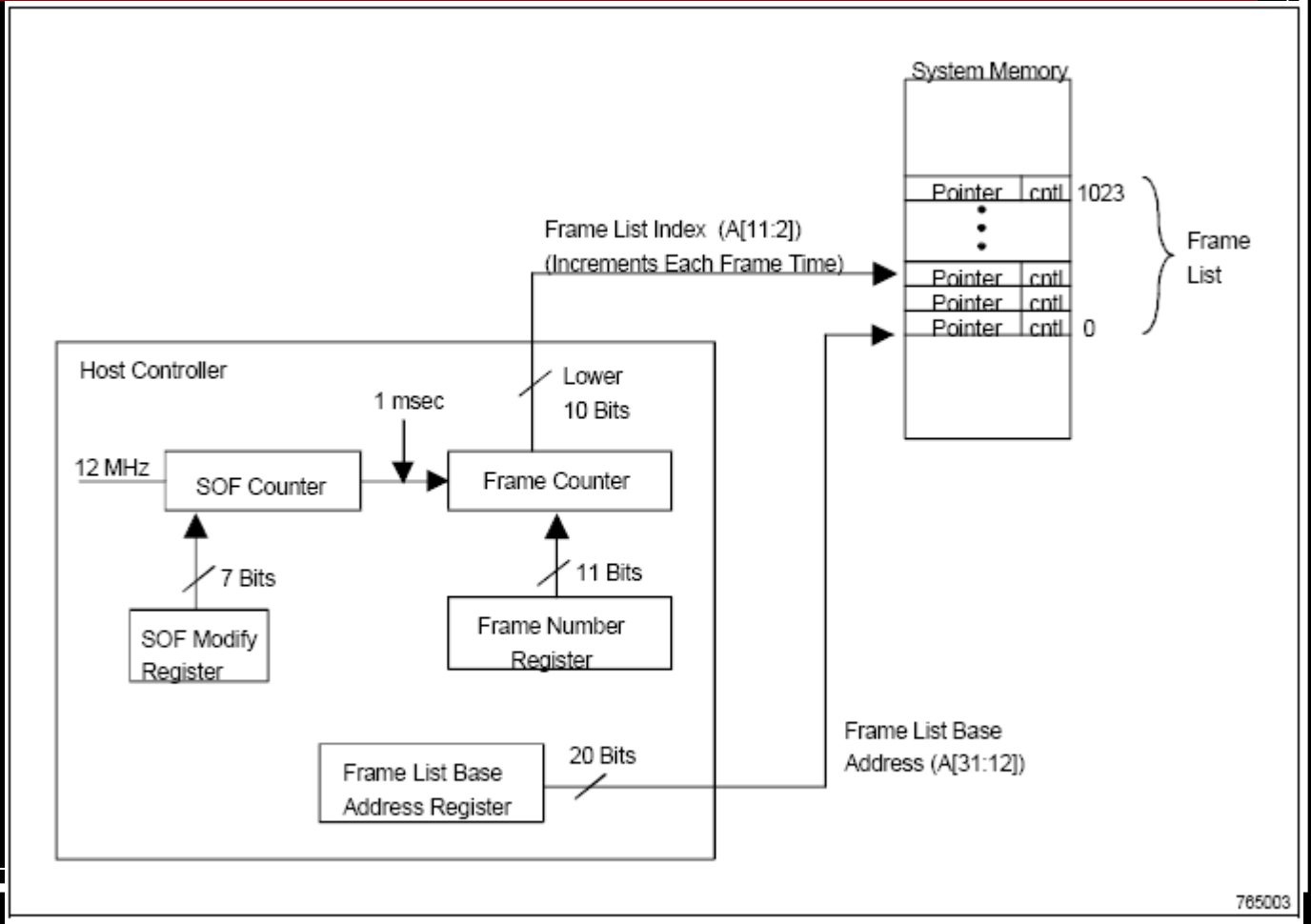


Figure 3. Frame Number Ties Frame List to Real Time

UHCI Frame list

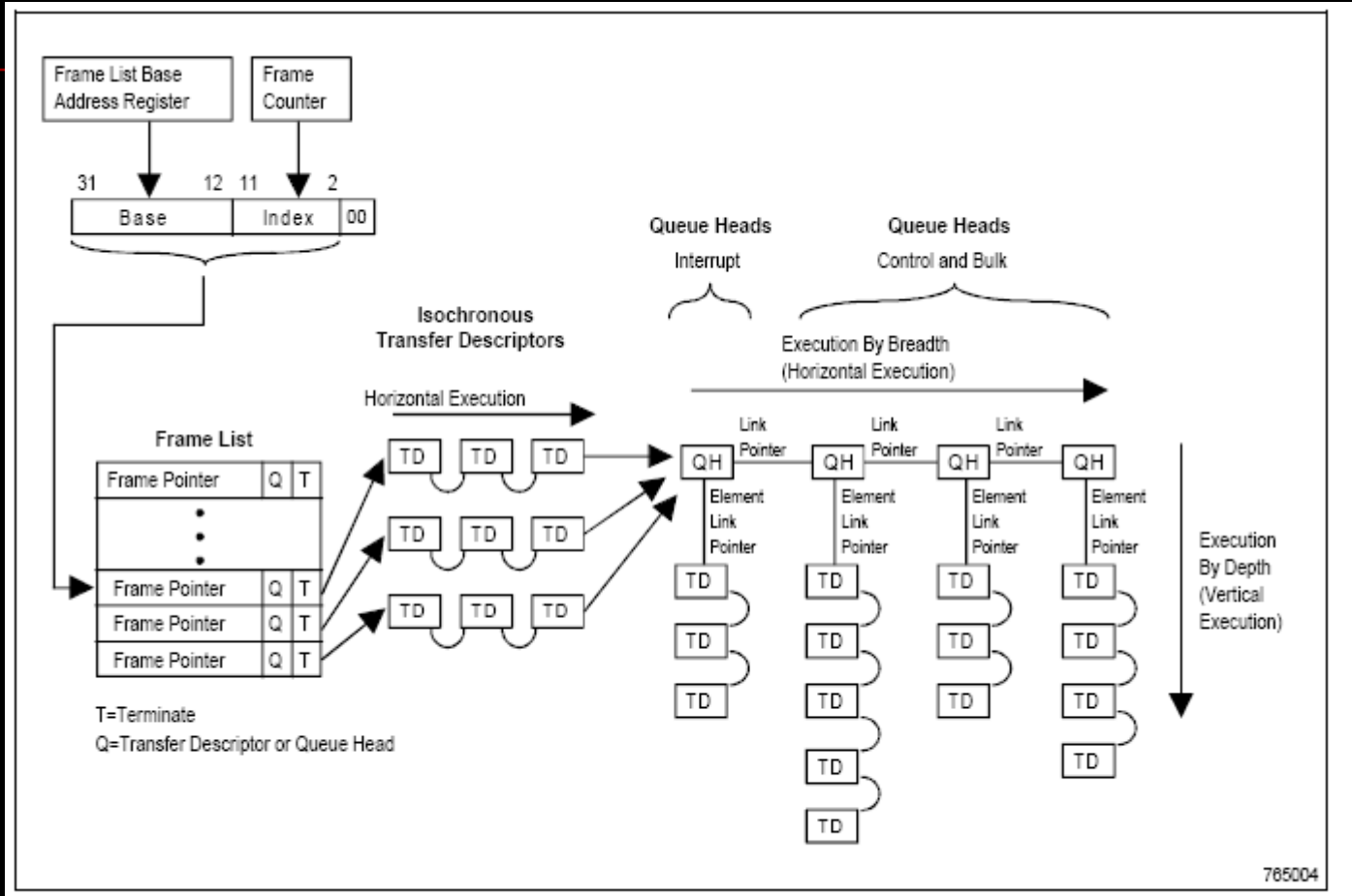


Figure 4. Example Schedule

Thank You