Introduction to SuperSpeed USB 3.0 Protocol

Ankur Tomar & Edmund Lim *– Global Technology Centre*
Volume 1, ***April 2011***

1. What is USB?

As its name suggests, Universal Serial Bus (USB) is an external bus architecture for connecting USB-capable peripheral devices to a host compute. The USB is formed in 1994 by a group of 7 companies namely Compaq, DEC, IBM, Intel, Microsoft, NEC and Nortel. Earlier, the objective of this standardisation is to enable the convenience of connecting external devices to PCs by replacing the multitude of connectors at the back of the PCs. But as technology advanced that leads to faster data transmission rate, USB technology has gradually evolved from just a connection between external devices to the PCs, to a serial single port on the computer that becomes a link for a myriad of devices (up to 127 devices in a USB system). USB 1.0 was released in January 1996 with only a data transfer rate of 1.5 Mbps. Moving on to USB 1.1 with a maximum data transfer rate of 12Mbps was soon introduced in September 1998.

Next, the USB 2.0 specification was launched in April 2000 and was standardized by the USB-IF (Implementers Forum, Inc) at the end of 2001. Hewlett-Packard, Intel, Lucent Technologies (now Alcatel-Lucent), NEC and Philips jointly led the initiative to develop a higher data transfer rate, with the resulting specification achieving 480 Mbps, 40 times faster than the original USB 1.1 specification.

The USB 3.0 specifications were released on November 2008. Here in USB 3.0, it has an increased data transfer rate (up to 5Gbps), decreased power consumption, increased power output and most importantly, USB 3.0 is backwards-compatible with USB 2.0. Also, USB 3.0 consists of a new higher speed bus known as SuperSpeed that is in parallel with the USB 2.0 bus.

2. How SuperSpeed USB 3.0 is Different from USB 2.0?

2.1. Speed: The most obvious change in SuperSpeed USB versus USB 2.0 high-speed is the over 10X speed increase from 480 Mbps to 5 Gbps.

2.2. Physical Layer: Electrical signalling of the physical layer has been changed from the simple two-wire system to a dual-simplex data path. Hence packets arrive and leave at the same time. This is done over a new set of connection along with the existing USB 2.0 two-wire interface, which remains untouched.

2.3. Polling: Polling is eliminated in USB 3.0 and replaced by asynchronous notification. In USB 2.0, the host need to continuously polls all connected peripheral devices to check if the device have data to transfer, to do so all devices must be "on" at all times. In SuperSpeed USB, the host waits until a higher level application tells it that there is a peripheral that has data to transfer. Then the host contacts that specific peripheral to check if data is ready to transfer. When both ends of the link are ready, the data is transferred.

2.4. Broadcast: SuperSpeed USB eliminates the broadcast nature of the USB 2.0 bus protocol and uses directed data transfer from/to the host and device. This again enables power savings by only turning ON the transceiver for the required device for which the data is intended to transfer. The other peripherals on the bus do not need to use power as the data request is not directed towards them.

2.5. Power: The amount of power available to USB 3.0 device is higher than USB 2.0, raised to 5V @900mA from 5V @500mA. The unconfigured current limit is also raised from 100mA to 150mA.

3. What are the USB System cComponents?

Basically, in a USB system configuration, it consists of namely the 3 essential parts which are the USB Host, USB Hub and USB function (peripherals)

3.1. USB Host: This is the area where the USB host controller is installed and where the client software/device driver runs. The *USB Host Controller* is the interface between the host and the USB peripherals. The host is responsible for the detection of the insertion and removal of USB devices and data flow between the host and the devices. The host also provides power to the attached devices.

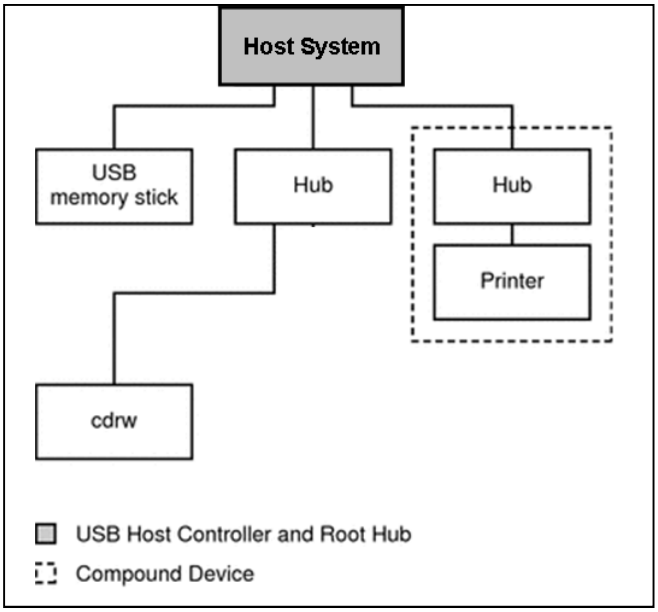

Figure 2: *USB Physical Device Hierarchy*

3.2. USB Hub: A USB device that allows multiple USB devices to attach to a single USB port on a USB host.

3.3. USB Function (Peripheral): A USB device that can transmit or receive data or control information over the bus and that provides a function. A function is typically implemented as a separate peripheral device that plugs into a port on a hub using a cable. This is known as a standalone device. However, it is also possible to create a compound device, which is a physical package that implements multiple functions and an embedded hub with a single USB cable. A compound device appears to the host as a hub with one or more non-removable USB devices, which may have ports to support the connection of external devices.

4. Topology used by USB:

In an USB topology or architecture, it employs tiered *Star topology* physical connections where each hub is the center of a star that can connect to peripherals or additional hubs. At any one time, only a maximum of 5 external hubs can be connected in series and up to a total of 127 peripherals and hubs including the root hub. However it may be impractical to have so many devices communicating with a single host controller. Despite having multiple devices connected at one time, only one device at a time can communicate with a host controller.

Thus, to increase the available bandwidth for USB devices, a PC can have multiple host controllers.

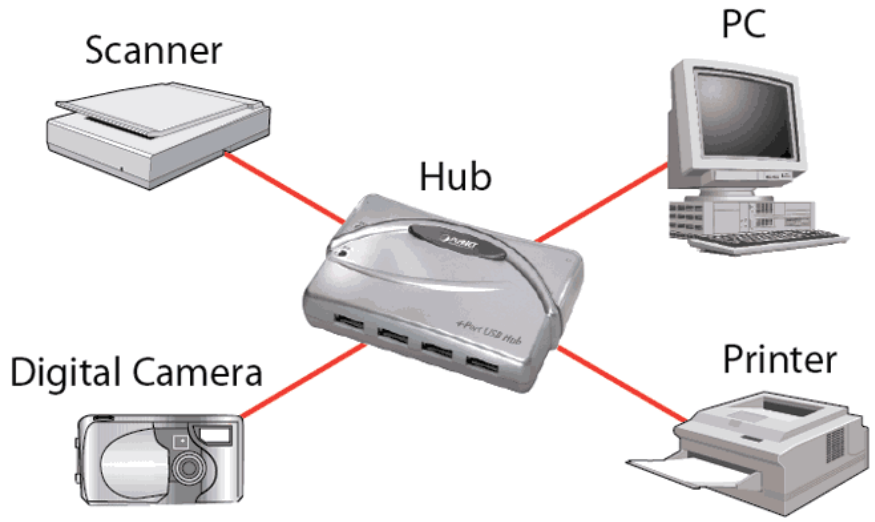

***Figure 3**: USB Star Topology*

5. USB Data Communication:

In USB data communication, the USB encodes the data by using the Non-Return-to-Zero Inverted (NRZI) transmission scheme. In NRZI coding, a "1" bit is represented by a transition of the physical level and "0" bit is represented by no transition of the physical layer. In addition, in order to ensure a minimum density of signal transitions that remains in the bitstream, an additional "0" bit is injected onto the data bus after the occurrence of 6 consecutive "1" bits. Here, in this method, is commonly known as bit stuffing or simply insertion of non-information bits into data. All this is done to ensure the presence of sufficient signal transitions for clock recovery.

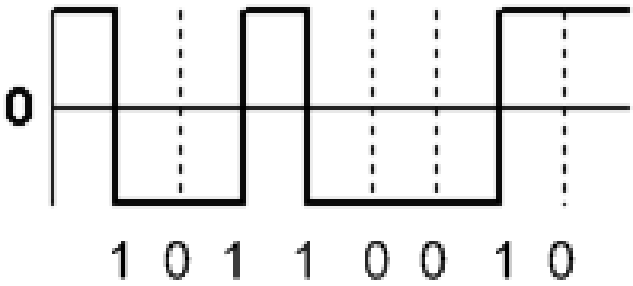

***Figure 4:** Example of NRZI encoding used by USB Data Communication.*

6. USB 3.0 Cables and Connectors:

USB 3.0 cables have two additional shielded differential pairs (SDP) of wires for a total of 8 signal wires. 3.0 cables have to be shielded to prevent electromagnetic interference and maximize signal integrity. This means the cables are thicker, heavier, less flexible and more expensive than 2.0 cables

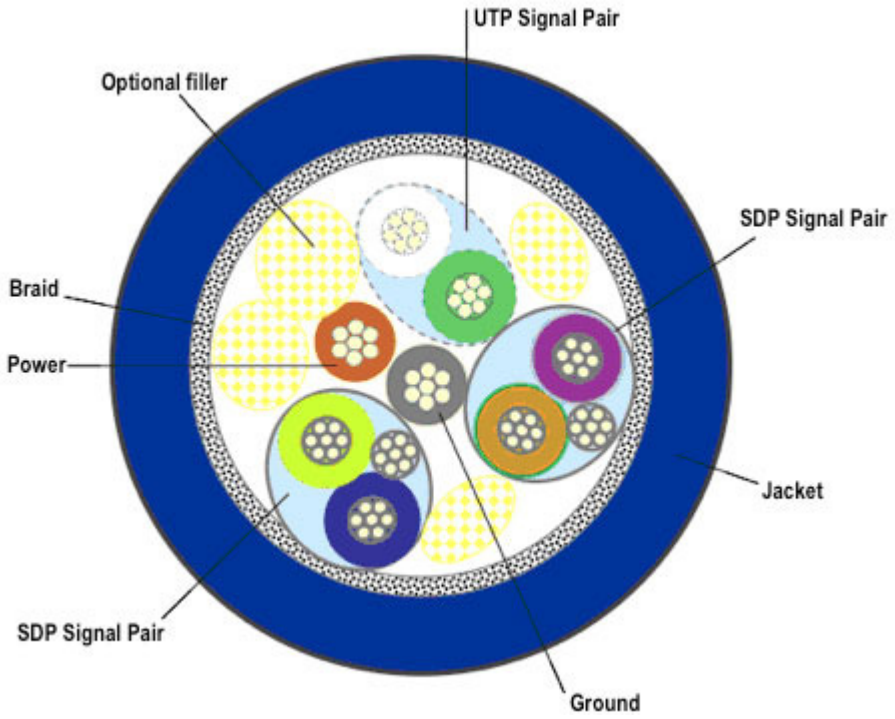

Figure 5: *Cross section view of SuperSpeed USB 3.0 cable*

There are several different USB 3.0 connectors, which are similar to the USB 2.0 connectors.

6.1. USB 3.0 Type-A: USB 3.0 Type-A connector's look very similar to USB 2.0 Type-A connectors aside from subtle differences. USB 3.0 Type-A connectors are slightly extended to accommodate pins for SuperSpeed mode. USB 3.0 Type-A connectors are compatible with USB 2.0 ports, and vice versa.

Figure 6: *USB 3.0 Type A Connector*

6.2. USB 3.0 Type-B: USB 3.0 Type-B connectors are modified USB 2.0 Type-B connectors with SuperSpeed pins added on top. USB 2.0 Type-B cables are compatible with USB 3.0 ports, but not vice versa. Type-B connectors are commonly used on large stationary devices like printers.

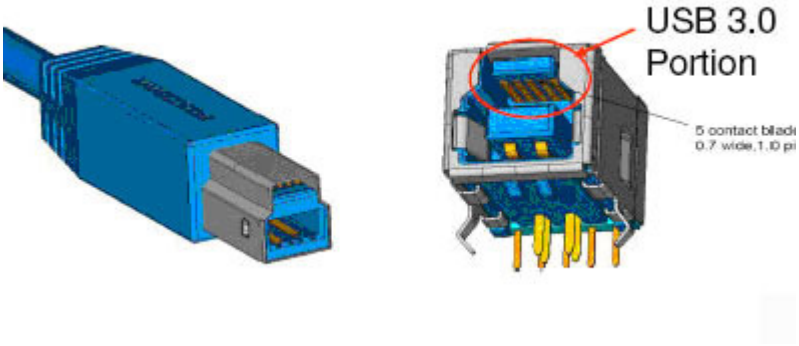

Figure 7: *USB 3.0 Type B Connector*

6.3. USB 3.0 Micro-B: The Micro-B connector is identical to the USB 2.0 connector but with an extended portion for the extra 5 pins. The USB 3.0 cable cannot be plugged into a USB 2.0 port, but the USB 2.0 cable can be used in a 3.0 port.

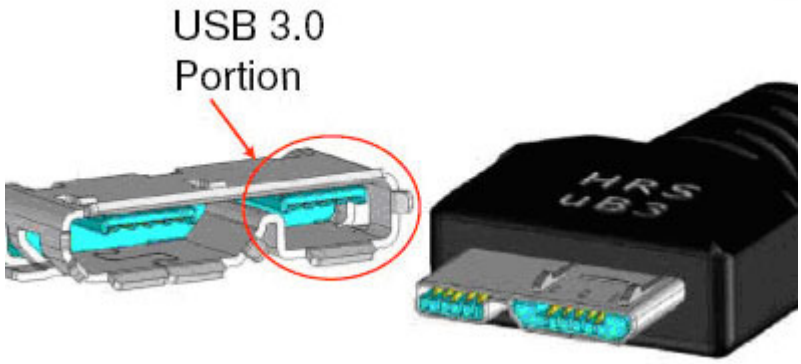

Figure 8: *USB 3.0 Micro B Type Connector*

7. USB 3.0 Enumeration:

USB Enumeration is the process of a host detecting that a USB device has been connected, identify what has been connected and then loading the relevant device drivers. The USB specification defines six device states. During enumeration, a device moves through four of the states: *Powered, Default, Address, and Configured*. (The other states are Attached and Suspend). This involves a mixture of hardware techniques for detecting something is present and software to identify what has been connected. After detecting the presence of a device the host will initiate a transfer with the device to determine what it is. The host does this by asking for device descriptors which define the device class and what drivers need to be loaded.

7.1. Types of USB Descriptors:

Devices are identified with the help of descriptors; each descriptor contains information about the device as a whole or an element in the device. USB descriptors are the data structures that enable the host to learn about a device. Each device (except compound devices) has one and only one *Device Descriptor* that contains information about the device and specifies the number of configurations the device supports. For each configuration all the device has *Configuration Descriptor* which provides information about the device's use of power and the number of interfaces the configuration supports. For each interface, the device has an *Interface Descriptor* that specifies the number of endpoints. Each endpoint has an *Endpoint Descriptor* that contains information needed to communicate with the endpoint.

SuperSpeed devices must provide a *Binary device Object Store (BOS) Descriptor* and at least two subordinate *Device Capability Descriptors*: a SuperSpeed USB descriptor and a USB 2.0 Extension descriptor. Every SuperSpeed endpoint descriptor has a subordinate SuperSpeed endpoint companion descriptor. A string descriptor can store text such as the vendor's or device's name or a serial number. On receiving a request for a configuration descriptor, a device should return the configuration descriptor and all of the configuration's interface, endpoint, and other subordinate descriptors up to the requested number of bytes.

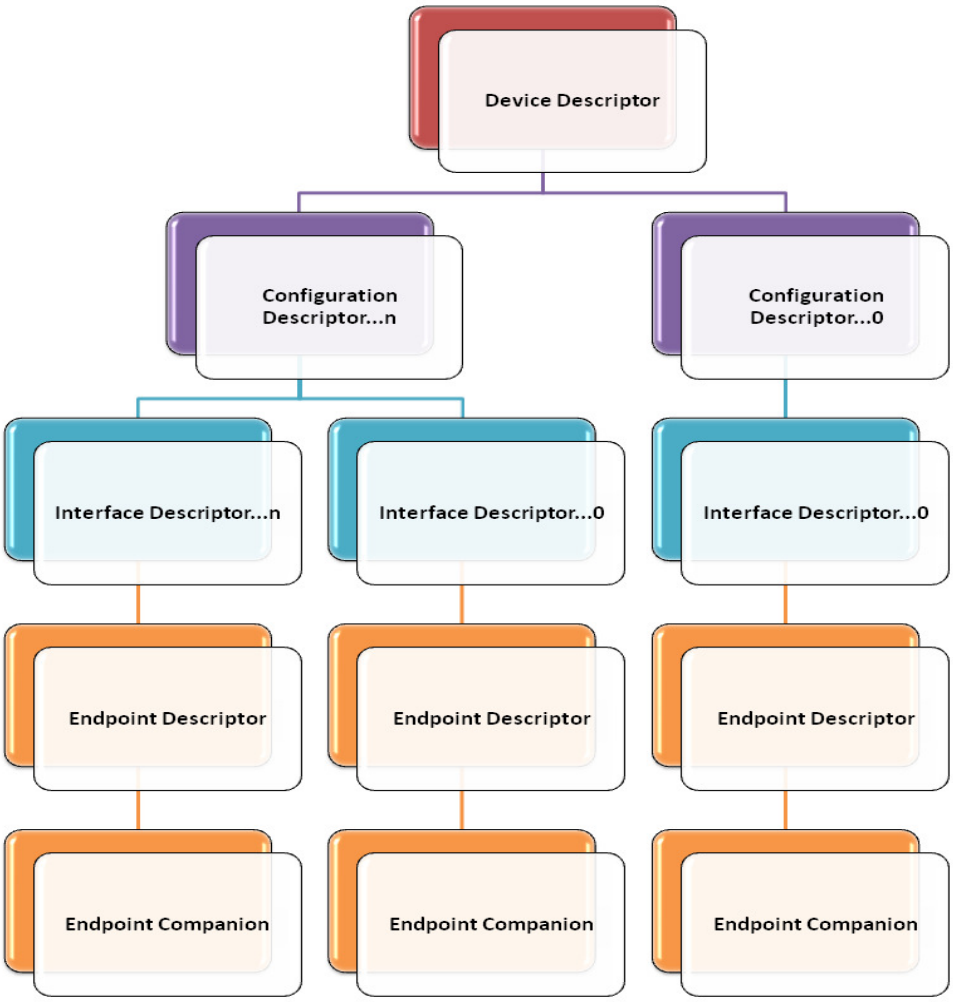

Figure 9: *Device Configuration Descriptor for SuperSpeed USB 3.0*

7.2. Steps involved in USB Enumeration:

SuperSpeed USB 3.0 device Enumeration works almost exactly as it does in USB 2.0. The steps below are a typical sequence of events that occurs during enumeration under Windows OS.

7.2.1 The user attaches a device to a USB port. The port may be on the root hub at the host or a hub that connects downstream from the host. The hub provides power to the port, and the device is in the Powered state. The device can draw up to 150 mA from the bus before configuration and 900 mA after configuration.

7.2.2. The Host detects the device. A USB port with no device connected uses 15kohm pull-down resistors to connect both USB D+ and D- lines to GND. The USB host monitors the voltages level on these signal lines (D+ and D-) of each of its ports. When a device plugs into a port, the device's brings its line high with its pull-up resistors, enabling the host to detect that a device is connected.

A low speed USB device (1.5Mbps) uses a 1k5 pull-up resistor to VCC on the USB DM signal line.

A full speed USB device (12Mbps) uses a 1k5 pull-up resistor to VCC on the USB DP signal line.

A high speed device (480Mbps) will initially appear as a full speed device to the host.

7.2.3. Detecting a Device has been connected. Detecting whether a connected device supports high speed, USB host uses two special signal states known as J and k Chirp. Host sends a series of alternating Chirp K and Chirp J. On detecting the pattern KJKJKJ, the device removes (switch OFF) its full-speed pull-up resistors and performs all further communications at high speed. If this initial communication fails then the USB host assumes that the device is a full speed device.

A J state is defined as a differential signal on USB D+ and USB D- >= +300mV.
A K state is defined as a differential signal on USB D+ and USB D- >= -300mV.

SuperSpeed USB 3.0: *On detecting a downstream SuperSpeed termination at a port, a host initializes and trains the port's link. Enumeration then proceeds at SuperSpeed with no need for further speed detecting.*

7.2.4. Establishing Signal pair: Once the USB host identify device is connected and at what speed it should communicate, then the host will send a reset to the USB device. The device communicates with the host using the default address of 00h. The device is in the Default state and device's USB registers are in their reset states. Now the device is ready

to respond to control transfers at endpoint zero. This reset is visible to the new device only and the other devices on the bus don't see the reset.

SuperSpeed USB3.0: *The host isn't required to reset the port after learning of a new device.*

7.2.5. Identifying What Device is connected. As mentioned above, devices are identified with the help of descriptors. This is basically a question and answer session between host and USB device, the host sends the "Get_Device_Descriptor" command to device default address 00h, endpoint zero. The host enumerates only one device at a time, only one device will respond to the communication addressed to the device address 00h, even if several devices were attach at once.

The host will receive first packet of 8 bytes of device descriptor with the information of maximum packet size supported by endpoint zero. The host begins the Status stage of the transfer. After completion of this stage the device is reset and host assigns a unique address by sending Set Address request. The device is now in the Address state. All communications from

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor size in bytes
1	bDescriptorType	1	The constant DEVICE (01h)
2	bcdUSB	2	USB specification release number (BCD)
4	bDeviceClass	1	Class code
5	bDeviceSubclass	1	Subclass code
6	bDeviceProtocol	1	Protocol Code
7	bMaxPacketSize0	1	Maximum packet size for Endpoint 0
8	idVendor	2	Vendor ID
10	idProduct	2	Product ID
12	bcdDevice	2	Device release number (BCD)
14	iManufacturer	1	Index of string descriptor for the manufacturer
15	iProduct	1	Index of string descriptor for the product
16	iSerialNumber	1	Index of string descriptor containing the serial number
17	bNumConfigurations	1	Number of possible configurations

Table 1: *The Device Descriptor identifies the product and its manufacturer, sets the maximum packet size for endpoint zero, and can specify a device class.*

this point on use the new address. The device completes the Status stage of the request using the default address and then implements the new address. After assigning new address host again sends a "Get Descriptor" request to the new address, this time the host retrieves the entire descriptor. The device descriptor contains the maximum packet size for endpoint zero, the number of configurations the device supports, and other basic information about the device.

7.2.6. The host learns about the device's abilities. The host continues to learn about the device by requesting the one or more configuration descriptors followed by its subordinate descriptors specified in the device descriptor. The device responds by sending the configuration descriptor followed by all of the configuration's subordinate descriptors, including interface descriptor(s), with each interface descriptor followed by any endpoint descriptors for the interface.

The Configuration Descriptor and the Interface Descriptor both are of a fixed length, 9 bytes, and is defined as type 2 and type 4 respectively. The Configuration descriptor provides device specific information such as the number of interfaces supported by the device and maximum power the device is expected to consume. The Interface descriptor provides information such as the number of endpoints to be used.

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor size in bytes
1	bDescriptorType	1	The constant Configuration (02h)
2	wTotalLength	2	The number of bytes in the configuration descriptor and all of its subordinate descriptors
4	bNumInterfaces	1	Number of interfaces in the configuration
5	bConfigurationValue	1	Identifier for Set_Configuration and Get_Configuration requests
6	iConfiguration	1	Index of string descriptor for the configuration
7	bmAttributes	1	Self/bus power and remote wakeup settings
8	bMaxPower	1	Bus power required, expressed as (maximum milliamperes/2)

Table 2: *The Configuration Descriptor specifies the maximum amount of bus current the device will require and gives the total length of the subordinate descriptors.*

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor size in bytes
1	bDescriptorType	1	The constant Interface (04h)
2	bInterfaceNumber	1	Number identifying this interface
3	bAlternateSetting	1	Value used to select an alternate setting
4	bNumEndpoints	1	Number of endpoints supported, not counting Endpoint 0
5	bInterfaceClass	1	Class code
6	bInterfaceSubclass	1	Subclass code
7	bInterfaceProtocol	1	Protocol code
8	iInterface	1	Index of string descriptor for the interface

Table 3: *The Interface Descriptor specifies the number of subordinate endpoints and may specify a USB class.*

7.2.7. SuperSpeed USB 3.0 Additional Descriptors. Some devices use additional descriptors to store information that is specific to a technology or a device function. To provide a standard way to provide this information, the SuperSpeed USB 3.0 specification introduced two new descriptor types: The Binary device Object Store (BOS) descriptor, functions as a base descriptor for one or more related device capability descriptors and a device capability descriptor, provides information about a specific capability or technology.

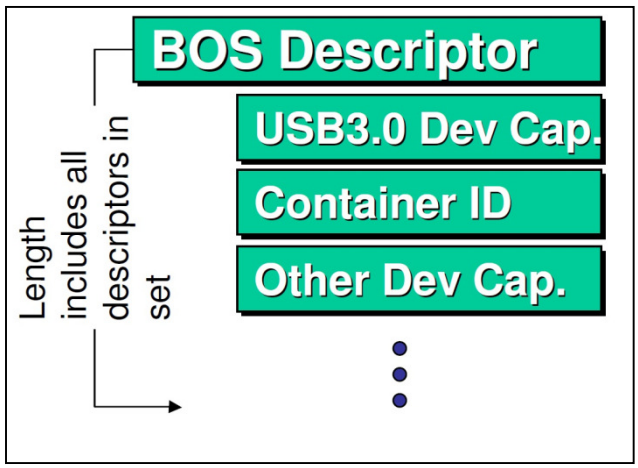

Figure 10: *SuperSpeed USB 3.0 introduced descriptor- BOS Descriptor*

All SuperSpeed USB 3.0 devices must provide this descriptor and must support Link Power Management (LPM) when operating at high speed. USB 2.0 devices that support Link Power Management also provide this descriptor

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor size in bytes (05h).
1	bDescriptorType	1	BOS (0Fh)
2	wTotalLength	2	The number of bytes in this descriptor and all of its subordinate descriptors
4	bNumDeviceCaps	1	The number of device capability descriptors subordinate to this BOS descriptor.

Table 4: *A Binary device Object Store (BOS) Descriptor provides a way to support descriptors that store additional information about a device.*

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor length in bytes (varies).
1	bDescriptorType	1	DEVICE CAPABILITY (10h)
2	bDevCapabilityType	1	01h = Wireless USB 02h = USB 2.0 EXTENSION 03h = SUPERSPEED_USB 04h = CONTAINER ID 00h, 05h–FFH (reserved)
3	Capability-Dependent	varies	Capability-specific data and format.

Table 5: *The Device Capability Descriptor can provide information that is specific to a technology or another aspect of a device or its function.*

7.2.8. Loading the Driver. When the USB device has been fully identified by the USB host using descriptors, the host looks for the best match in a driver to manage communications with the device. This is done with the help of Windows INF file which contains Vendor ID and Product ID. The device driver with the best matched is installed with the desired configuration by sending “Set Configuration” request by the driver. On receiving the request, the device implements the requested configuration. Now device is in Configured state and available for applications to access.

After the initial installation the settings are saved in the PC registry so that on subsequent plug-ins of the USB device, the driver is automatically loaded.

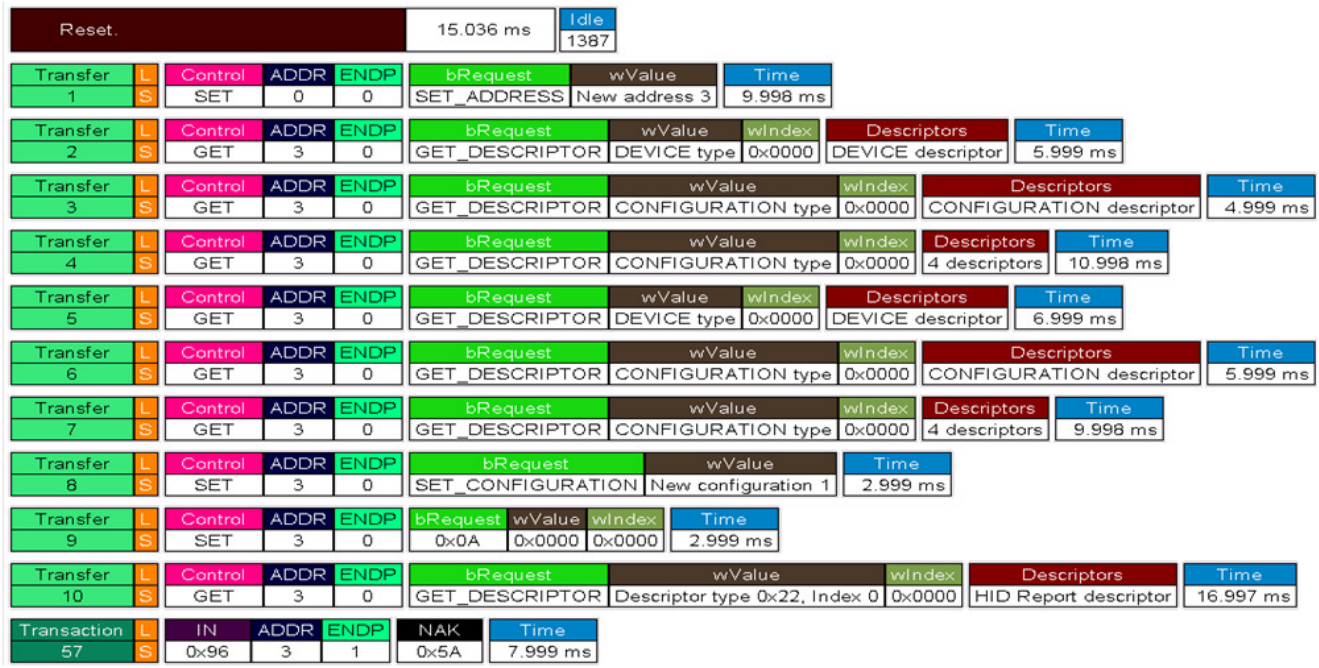

Figure 11: *USB Enumeration Trace*

8. USB Device Class:

As a group of devices or interfaces share many attributes or provide or request similar services, it is vital to define the attributes and services in a class specification.

Therefore, USB defines class code information that is used to identify a device's functionality and to nominally load a device driver based on that functionality. Also, all USB class specifications are based on the Common Class specification, which describes what information a class specification should contain and how to organize a specification document.

Base Class	Descriptor Usage	Description
00h	Device	Use class information in the Interface Descriptors
01h	Interface	Audio
02h	Device or Interface	Communications and CDC Control
03h	Interface	HID (Human Interface Device)
05h	Interface	Physical
06h	Interface	Image
07h	Interface	Printer
08h	Interface	Mass Storage
09h	Device	Hub
0Ah	Interface	CDC-Data
0Bh	Interface	Smart Card
0Dh	Interface	Content Security
0Eh	Interface	Video
0Fh	Interface	Personal Healthcare
DCh	Device or Interface	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Device or Interface	Miscellaneous
FEh	Interface	Application Specific
FFh	Device or Interface	Vendor Specific

***Table 6**: USB device Class codes*

9. USB Data Transfer:

Here, it describes the data flow in the USB devices. Basically, the key structures of the data flow are the endpoint and the pipe. An endpoint is a uniquely identifiable entity on a USB device that is the source or terminus of the data that flows from or to the device. Next, the pipes are the logical channels that link between an endpoint on the USB device and software on the host (host controller).

However, there are 2 types of pipes namely the stream pipe or message pipe depending on the type of data transfer. Stream pipes handle the interrupt, bulk and isochronous transfers while message pipes support the control transfer type. *Figure 8* shows the overview of the USB data flow physical structure that consists of the hosts, pipes and endpoints.

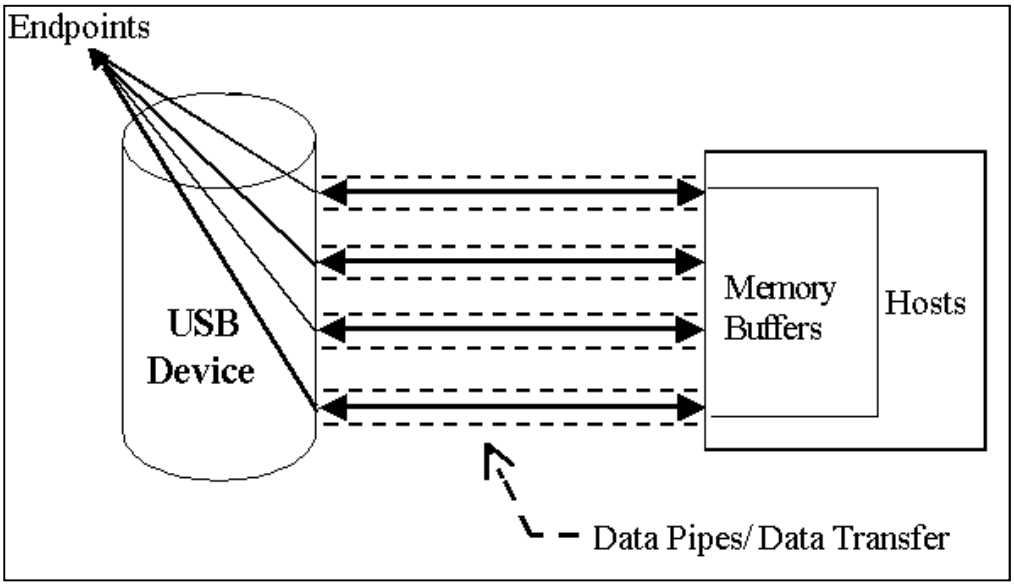

***Figure 12** : USB data flow physical structure*

As mentioned earlier, in USB protocol, basically there are 4 Data Transfer Mechanism to choose from so that to achieve the optimal communication for different peripheral devices. In addition, the methods differ in the amount of data that can be moved in a single transaction in whether any particular periodicity or latency can be guaranteed. The 4 types of transfers are arranged in ascending orders of their priorities:

9.1. Control Transfer:

- Used to send and receive structural information of control nature (used by the Host to send commands or query parameters)
- Can be used in all devices
- Priority 3

9.2. Bulk Transfer:

- Used to send or receive blocks of unstructured data
- Mainly used in Printers, Scanners, Digital Cameras and other peripheral devices that need high reliability transfers
- Priority 3

9.3. Interrupt Transfer:

- Used on devices that need guaranteed quick response (asynchronous time frame)
- Mainly used in pointing devices, mouse, keyboard and joy-stick
- Priority 2

9.4. Isochronous Transfer:

- Used for transfers that need to be performed in real-time (data must arrive at a constant rate, or by specific time)
- Mainly used in speaker, microphone and telephone
- Priority 1

Transfer Type/Speed	High Speed (480Mbps)	Full Speed (12Mbps)	Low Speed (1.5Mbps)	Direction
Control	64 byte/packet	8,16,32 or 64 byte/packet	8 byte/packet	Bi-Dir
Bulk	<512 byte/packet	8,16,32 or 64 byte/packet	Not allowed	Unidir
Interrupt	<1024 byte/packet	< 64 byte/packet	< 8 byte/packet	Unidir
Isochronous	< 3072 byte/packet	< 1023 byte/packet	Not allowed	Unidir

Table 7: *Supported transfer types and speed*

In USB transfer protocol, it uses the concept of frames and microframes which presents 1ms and 125µs respectively for time management. Since both isochronous and interrupt transfers require time management, the amount of Non-Periodic (Bulk or Control) transfers within a particular time unit will vary depending on the amount of Periodic (Interrupt or Isochronous) transfers within the same time unit. This means the bulk and control transfers use whatever space is left.

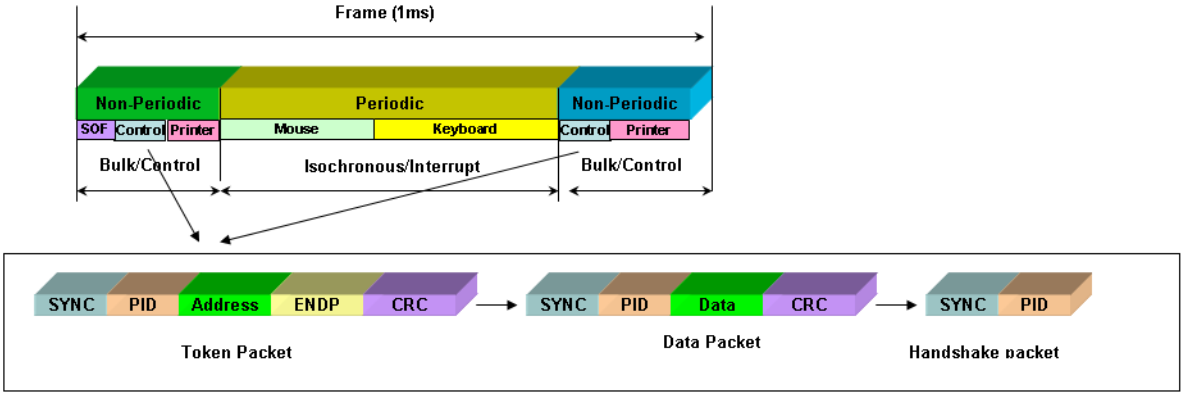

***Figure 13**: Overview of a typical single frame control transfer*

In addition, all USB data is sent serially with least significant bit (LSB) being sent first. USB data transfer is essentially in the form of packets of data which will be sent to and forth between the host and the peripheral devices. In all 4 different transfers of Control, Bulk, Interrupt and Isochronous, these 4 transfers consist of 3 types of packets namely the i) Token, ii) Data and iii) Handshake.

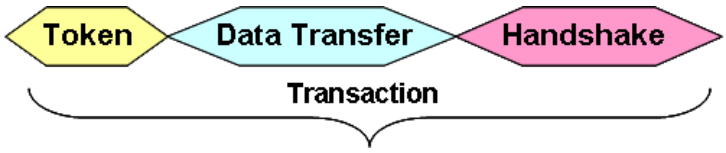

***Figure 14**: Overview of a transaction frame*

The above 3 packets form a **TRANSACTION** where the 4 different data transfers can be made of either one or several transactions.

9.5. Token Packets:

Generally, it consists of the following elements: sync sequence, Package identifier, device address, endpoint number and 5-bit Cyclic Redundancy Check (CRC). Tokens are only sent by the host, not by a device.

***Figure 15**: Frame for Token Packet*

9.6. Data Packets

The data packets are always preceded by an address token. It consists of the following elements: sync sequence, package identifier, data and 16-bit Cyclic Redundancy Check (CRC). Also, the data must be sent in multiples of bytes.

- Maximum data size for low-speed devices is 8 bytes.
- Maximum data size for full-speed devices is 1023 bytes.
- Maximum data size for high-speed devices is 1024 bytes.

Figure 16: *Data Packet*

9.7. Handshake Packets

They are generally sent in response to data packets. It consists of the following elements: sync sequence and Package identifier.

Figure 17*: Handshake Packet*