Universal Serial Bus

Dispense HTML

Salvatore Venticinque

Università
degli Studi
della Campania
Luigi Vanvitelli

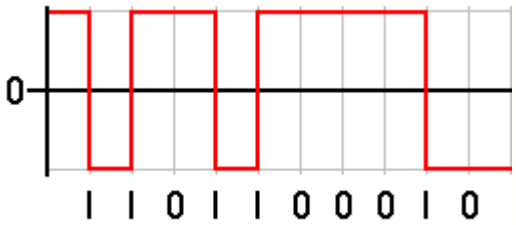
Intro 1/3

- Pheripheral Bus
- USB version 1.1 supported two speeds:
  - full speed mode of 12Mbits/s
  - low speed mode of 1.5Mbits/s.
- USB 2.0
  - High Speed mode of 480Mbits/s replaced Firewire Serial Bus.

Intro 2/3

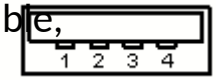
- Universal Serial Bus is ***host controlled***.
  - only one host per bus, not multimaster arrangement.
  - USB 2.0 has introduced a Host Negotiation Protocol which allows two devices negotiate for the role of host (smartphone).
  - The USB host is responsible for undertaking all transactions and scheduling bandwidth.
- USB uses a tiered star topology
  - This imposes the use of a hub somewhere
  - However many devices have USB hubs integrated into them.
- Tiered star topology, rather than simply daisy chaining:
  - power to each device can be monitored and even switched off if an overcurrent 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.
- Up to 127 devices can be connected to any one USB bus at any one given time.
  - Need more devices? - simply add another port/host.

Intro 3/3



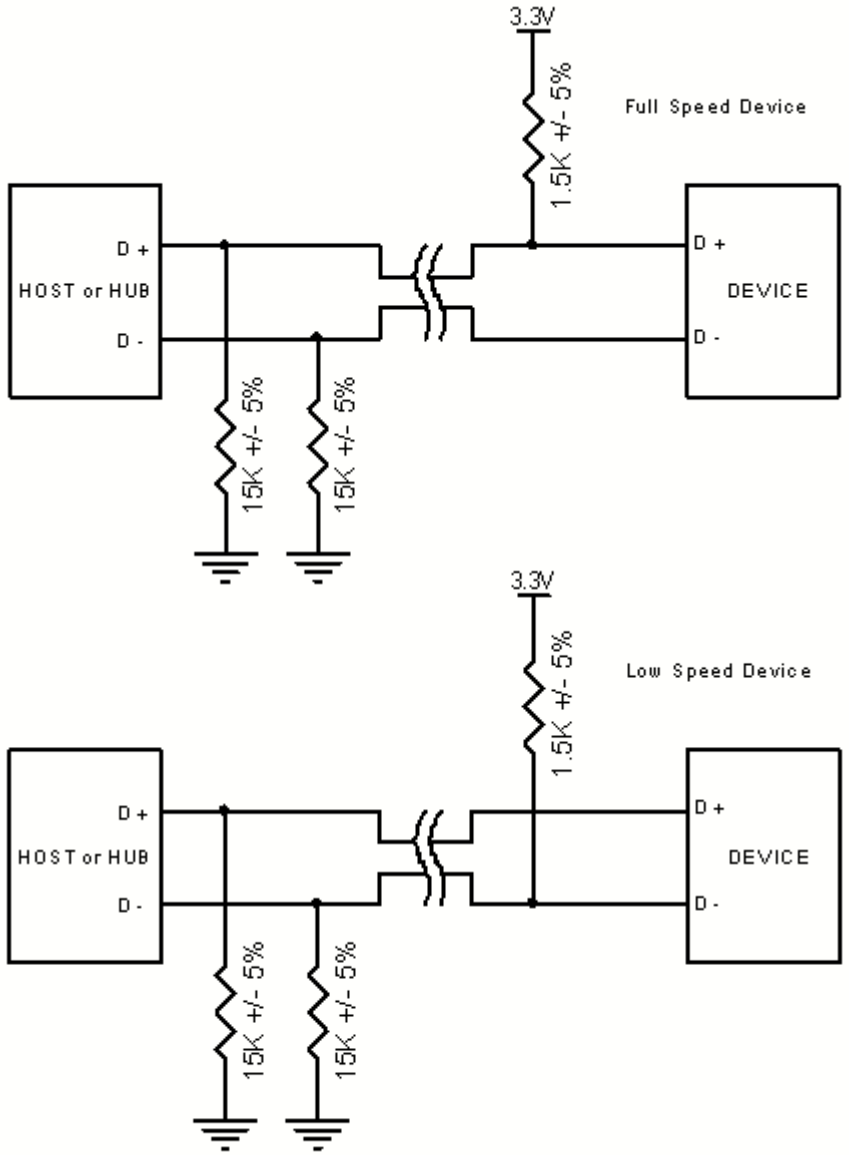
- USB:
  - is a serial bus;
  - uses 4 shielded wires of which two are power (+5v & GND).
  - The remaining two are twisted pair differential data signals.
  - uses a NRZI (Non Return to Zero Invert) encoding scheme to send data with a sync field to synchronise the host and receiver clocks.
- USB supports plug'n'plug with dynamically loadable and unloadable drivers.
  - The user plugs the device into the bus.
  - The host will detect this addition,
  - interrogate the newly inserted device
  - and load the appropriate driver.
- The loading of the appropriate driver is done using a PID/VID (Product ID/Vendor ID) combination.
  - The VID is supplied by the USB Implementor's forum at a cost
  - The latest info on fees can be found on the USB Implementor's Website
- Most chip manufacturers will have a VID/PID combination you can use for your chips which is known not to exist as a commercial device.
- Other chip manufacturers can even sell you a PID to use with their VID for your commercial device.

- Connectors
  - Upstream and downstream connectors are not mechanically interchangeable,
  - Type A sockets will typically find themselves on hosts and hubs.
  - The only type A plug to type A plug devices are bridges which are used to connect two computers together.



- Pin Number Cable Colour Function
  - 1 Red VBUS (5 volts)
  - 2 White D-
  - 3 Green D+
  - 4 Black Ground
- Electrical
  - USB uses a differential transmission pair for data encoded using NRZI.
  - On low and full speed devices, a differential ‘1’ is transmitted by pulling D+ over 2.8V with a 15K ohm resistor pulled to ground and D- under 0.3V with a 1.5K ohm resistor pulled to 3.6V.
  - A differential ‘0’ on the other hand is a D- greater than 2.8V and a D+ less than 0.3V with the same appropriate pull down/up resistors.
  - The receiver defines a differential ‘1’ as D+ 200mV greater than D- and a differential ‘0’ as D+ 200mV less than D-.
  - The polarity of the signal is inverted depending on the speed of the bus.

- Speed Identification
  - High speed devices will start by connecting as a full speed device (1.5k to 3.3V).
  - Once it has been attached, establishes via protocol 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.



- USB is bus-powered devices
  - A USB device specifies its power consumption expressed in 2mA units in the configuration descriptor.
  - A device cannot increase its declared power consumption, even if it looses external power.
- ***Low power bus powered functions***
  - draw all its power from the VBUS (max one unit load: 100mA).
  - must be designed to work down to a VBUS voltage of 4.40V and up to a maximum voltage of 5.25V.
  - For many 3.3V devices, LDO regulators are mandatory.
- ***High power bus powered functions:***
  - will draw all its power from the bus and cannot draw more than one unit load until it has been configured,
  - after it can then drain 5 unit loads (500mA Max)
- ***Self power functions***
  - may draw up to 1 unit load from the bus and the rest from an external source.
- If VBUS is lost, the device has a lengthy 10 seconds to remove power from the D+/D- pull-up resistors used for speed identification.

Suspend Mode

- Suspend mode is mandatory on all devices.
  - The maximum suspend current is proportional to the unit load.
  - For a 1 unit load device (default) the maximum suspend current is 500uA.
- A USB device will enter suspend when there is no activity on the bus for greater than 3.0ms.
- It then has a further 7ms to shutdown and draw no more than the designated suspend current.
- USB has a start of frame packet or keep alive sent periodically on the bus. This prevents an idle bus from entering suspend mode in the absence of data.
  - A high speed bus will have micro-frames sent every 125.0 µs ±62.5 ns.
  - A full speed bus will have a frame sent down each 1.000 ms ±500 ns.
  - A low speed bus will have a keep alive which is a EOP (End of Packet) every 1ms only in the absence of any low speed data.
- The term "Global Suspend" is used when the entire USB bus enters suspend mode collectively.
- However selected devices can be suspended by sending a command to the hub that the device is connected too. This is referred to as a "Selective Suspend."
- The device will resume operation when it receives any non idle signalling.

USB Protocols

- Unlike RS-232 USB is made up of several layers of protocols.
- USB controller will take care of the lower layer, thus making it almost invisible .
- Each USB transaction consists of a
  - Token Packet (Header defining what it expects to follow),
  - Optional Data Packet, (Containing the payload)
  - Status Packet (Used to acknowledge transactions and to provide a means of error correction)
- The host initiates all transactions.
- The first packet, called a token is generated by the host to describe:
  - what is to follow
  - whether the data transaction will be a read or write
  - what the device's address and designated endpoint is.
- The next packet is generally a data packet carrying the payload
- The last packet is an handshaking packet, reporting if the data or token was received successfully, or if the endpoint is stalled or not available to accept data.

USB Packet 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
  - is used to synchronise the clock of the receiver with that of the transmitter.
  - The last two bits indicate where the PID fields starts.

Packet ID (PID)

Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

To insure it is received correctly, the 4 bits are complemented and repeated, making an 8 bit PID in total. The resulting format is shown below.

PID_0	PID_1	PID_2	PID_3	$nPID_0$	$nPID_1$	$nPID_2$	$nPID_3$

Packet Fields

- ADDR
  - 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

Packet Types: Token packets

- In
  - Informs the USB device that the host wishes to read information.
- Out
  - Informs the USB device that the host wishes to send information.
- Setup
  - Used to begin control transfers.
- Token Packets must conform to the following format:

Sync	PID	ADDR	ENDP	CRC5	EOP

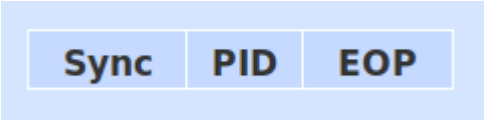
Data Packets

- There are two types of data packets each capable of transmitting up to 1024 bytes of data.
  - Data0
  - Data1
  - High Speed mode defines another two data PIDs, DATA2 and MDATA.
- Data packets have the following format,

- Maximum data payload size for low-speed devices is 8 bytes.
- Maximum data payload size for full-speed devices is 1023 bytes.
- Maximum data payload size for high-speed devices is 1024 bytes.
- Data must be sent in multiples of bytes.

Handshake Packets

- There are three type of handshake packets which consist simply of the PID
  - ACK - Acknowledgment that the packet has been successfully received.
  - NAK - Reports that the device temporary cannot send or received data. Also used during interrupt transactions to inform the host there is no data to send.
  - STALL - The device finds its in a state that it requires intervention from the host.
- Handshake Packets have the following format,



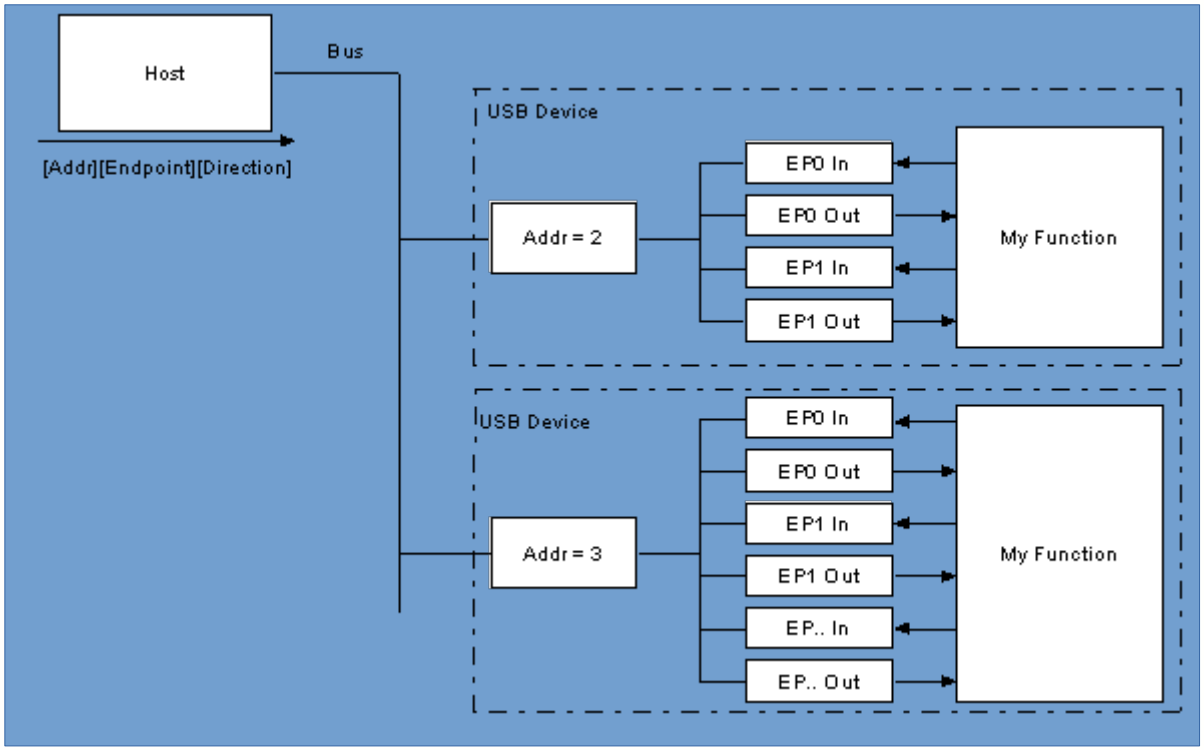
Start of Frame

- The SOF packet consisting of an 11-bit frame number is sent by the host:
  - every 1ms ± 500ns on a full speed bus
  - or every 125 µs ± 0.0625 µs on a high speed bus.

Sync	PID	Frame Number	CRC5	EOP

USB Functions

- USB devices provide a capability or function such as a Printer, Zip Drive, Scanner, Modem or other peripheral.



- Most functions will have a series of buffers, typically 8 bytes long.
- Each buffer will belong to an endpoint - EP0 IN, EP0 OUT etc.
- Say for example, the host sends a device descriptor request.
  1) The function hardware will read the setup packet
  2) determine from the address field whether the packet is for itself,
  3) if so will copy the payload of the following data packet to the appropriate endpoint buffer dictated by the value in the endpoint field of the setup token.
  4) It will then send a handshake packet to acknowledge the reception of the byte
  5) and generate an internal interrupt within the semiconductor/micro-controller for the appropriate endpoint signifying it has received a packet.
- This is typically all done in hardware.
- The software now gets an interrupt, and should read the contents of the endpoint buffer and parse the device descriptor request.

Endpoints and Pipes

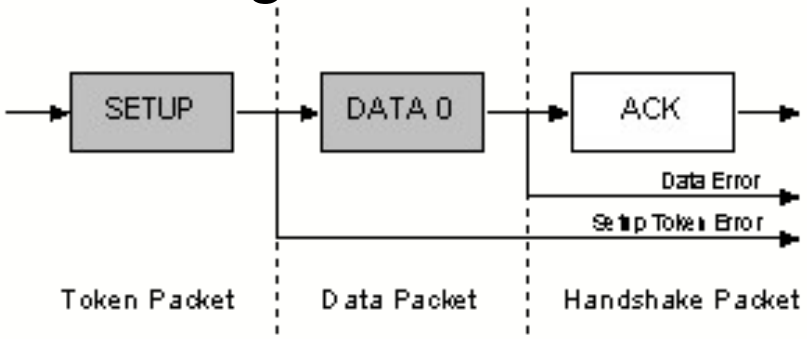
- ***Endpoints***
  - Can be described as sources or sinks of data.
  - Endpoints occur at the end of the communications channel at the USB function.
  - Device driver may send a packet to your devices EP1 → the firmware will then at its leisyre read this data.
  - If it wants to return data, the function 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.***
- ***Pipes***
  - A pipe is a logical connection between the host and endpoint(s).
  - The device sends and receives data on a series of endpoints,
  - The client software transfers data through pipes.
  - Pipes will also have a set of parameters: allocated bandwidth, transfer type, a direction of data flow, maximum packet/buffer sizes.
- ***USB defines two types of pipes***
  - ***Stream Pipes*** have: no defined USB format. Data flows sequentially and has a pre-defined direction, either in or out. Stream pipes can either be controlled by the host or device.
  - ***Message Pipes*** have a defined USB format. They are host controlled, which are initiated by a request sent from the host. Data is then transferred in the desired direction, dictated by the request. Therefore message pipes allow data to flow in both directions but will only support control transfers

Endpoint Types

- The Universal Serial Bus specification defines four transfer/endpoint types,
  - Control Transfers
  - Interrupt Transfers
  - Isochronous Transfers
  - Bulk Transfers

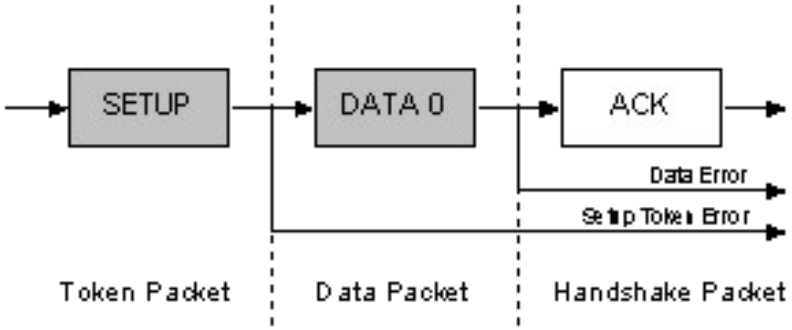
Control Transfers

- Control transfers are typically used for command and status operations.
- They are essential to set up a USB devices
- They are typically bursty, random packets which are initiated by the host and use best effort delivery.
- 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
  - full speed devices must have a packet size of 64 bytes.
- A control transfer can have up to three stages.
  - ***Setup Stage***
  - ***Data Stage***
  - ***Handshake Stage***



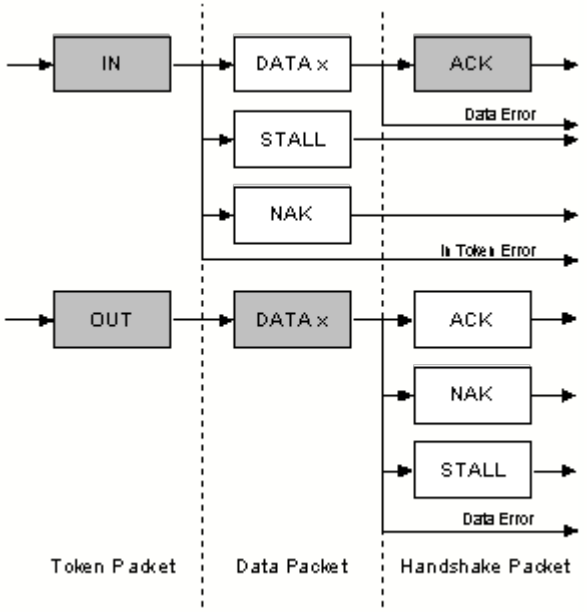
Control Transfers: Setup Stage

- ***The Setup Stage*** consists of three packets.
  - The setup token contains the address and endpoint number.
  - The data packet has a PID type of data0 and includes details the type of request
  - An handshake used for acknowledging successful receipt or to indicate an error.
- If the function successfully receives the setup data it responds with ACK,
- otherwise it ignores the data and doesn't send a handshake packet.



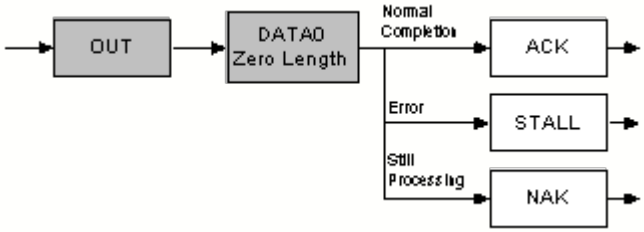
Control Transfer: Data Stage

- The *optional Data Stage* consists of one or multiple IN or OUT transfers.
- IN: the host is ready to receive control data.
  - If the function receives the IN token with an error, it ignores the packet.
  - Otherwise it can either reply:
    - with a DATA packet containing the control data
    - a STALL packet indicating the endpoint has had a error
    - or a NAK packet indicating that temporarily has no data to send.
- OUT the host want to send data to the function:
  - OUT token followed by a data packet with control data as the payload.
  - If the OUT token or data packet is corrupt the function ignores the packet.
  - If the function's endpoint buffer was empty and it stored the data into the endpoint buffer it issues an ACK.
  - If the endpoint buffer is not empty, then the function returns a NAK.
  - However if the endpoint had a error it returns a STALL.
- Data are sent in multiple transfers, each being the maximum packet length except for the last packet

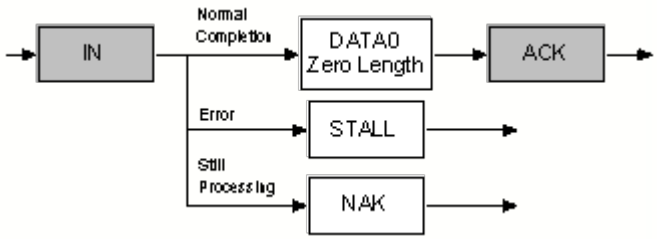


Control Transfer: Status Stage

- Status Stage reports the status of the overall request and this varies due to direction of transfer.
- Status reporting is always performed by the function.
- IN:
  - The host acknowledges the successful recept sending an OUT token followed by a zero length data packet.
  - The function can now report its status:
  - An ACK indicates it is ready to accept another command.
  - If an error occurred then the function will issue a STALL.
  - However if the function is still processing, it returns a NAK.



- OUT:
  - the function will acknowledge the successful receipt of data by sending a zero length packet in response to an IN token.
  - If an error occurred, it should issue a STALL
  - If it is still busy processing data, it should issue a NAK



Control Tranfer: An Example

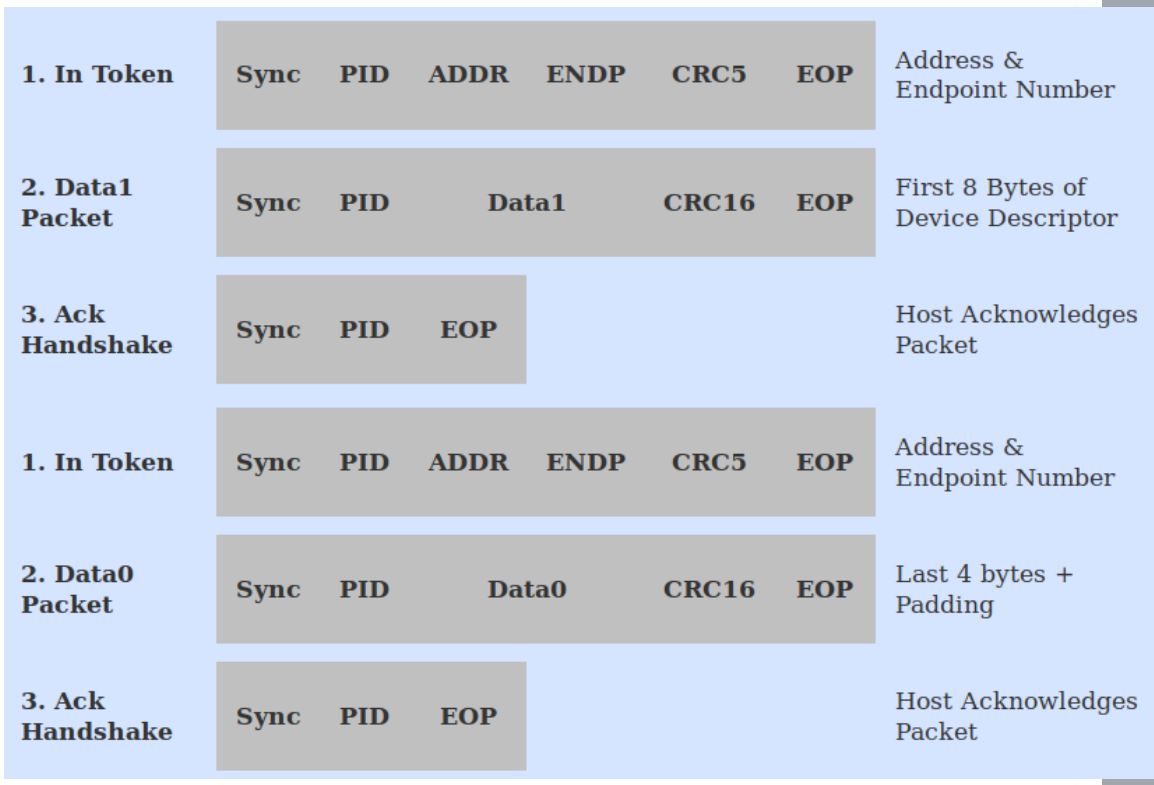
- The first USB transaction to ask the Device Descriptor Request

1. Setup Token	**Sync**	**PID**	**ADDR**	**ENDP**	**CRC5**	**EOP**	Address & Endpoint Number
2. Data0 Packet	**Sync**	**PID**	**Data0**		**CRC16**	**EOP**	Device Descriptor Request
3. Ack Handshake	**Sync**	**PID**	**EOP**				Device Ack. Setup Packet

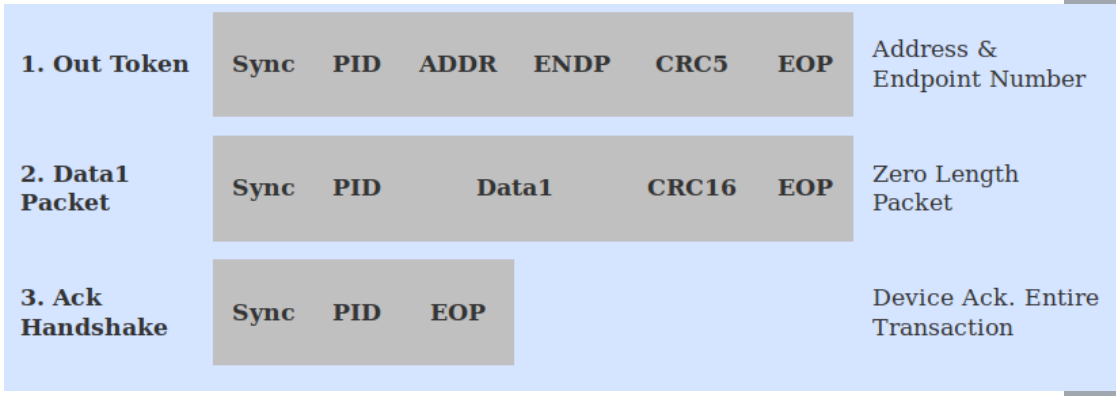
- The USB device decodes the 8 bytes received, and determines it was a device descriptor request.
- The device will then attempt to send the Device Descriptor, which will be the next USB transaction

Returning the Device Description Request

- The IN token tells the device it can send data for th endpoint.
- 12 bytes data are split in 2 chunks of 8 bytes
- The host acknowledges every data packet.



- A status transaction follows.
- If the transactions were successful, the host will send a zero length packet.
- The function then replies indicating its status.

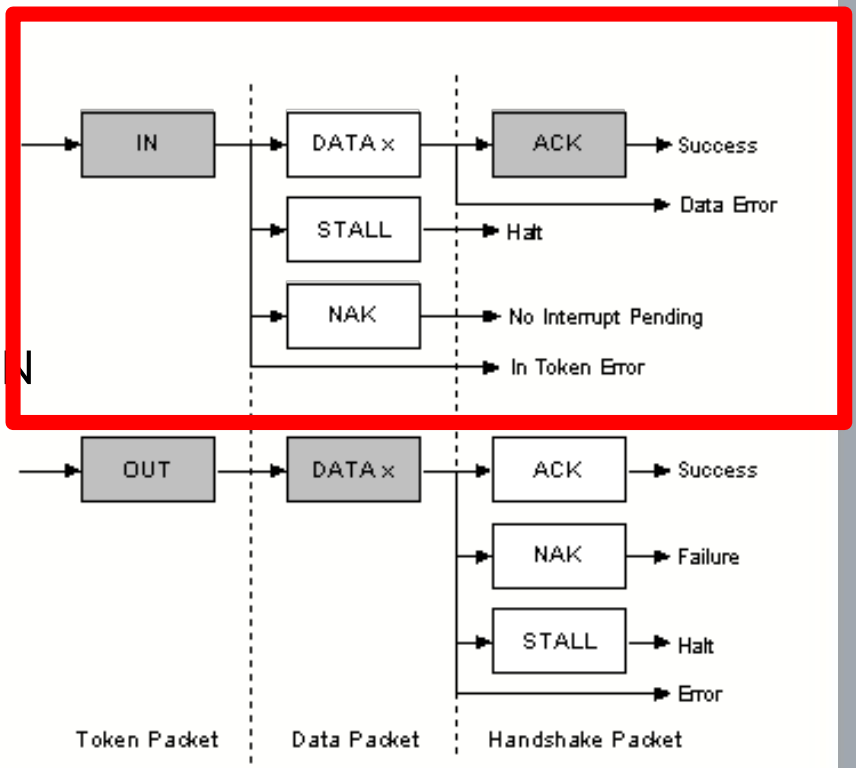


Interrupt Transfer

- Under 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!
- An Interrupt request is queued by small device until the host polls the USB device asking for data.
  - Guaranteed Latency
  - Stream Pipe - Unidirectional
  - Error detection and next period retry.
- Maximum Payload size:
  - For low-speed devices is 8 bytes.
  - For full-speed devices is 64 bytes.
  - For high-speed devices is 1024 bytes.

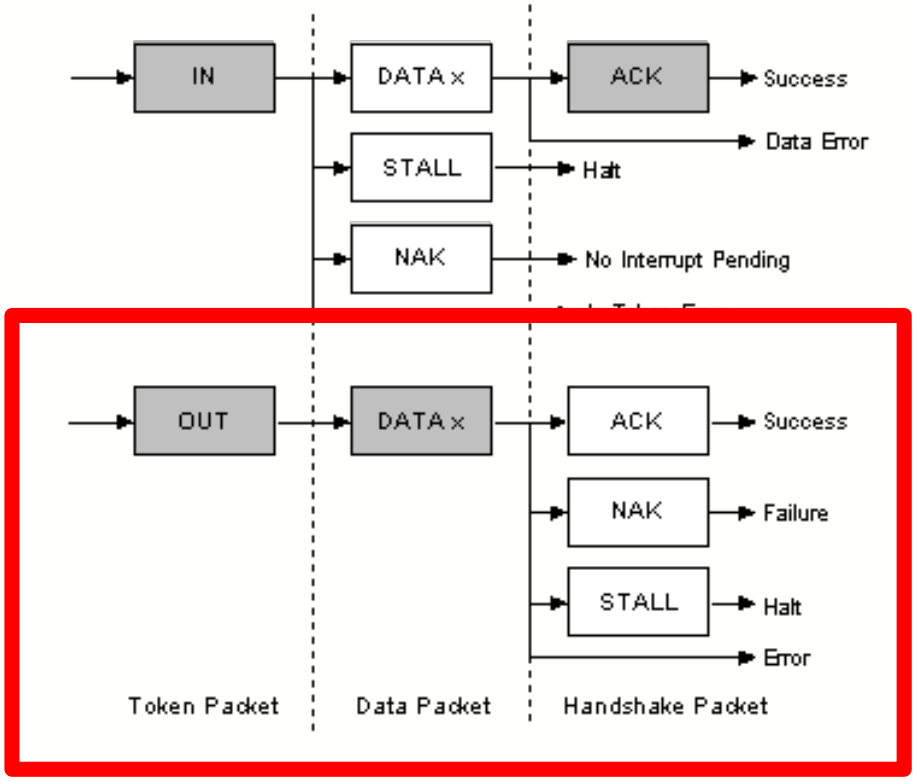
Interrupt Transactions (IN)

- The host will periodically poll the interrupt endpoint by sending an IN Token.
- This rate of polling is specified in the endpoint descriptor.
- The function can:
  - ignores the packet and wait for new tokens if IN is corrupted.
  - Returns a data packet if it needs attention
  - NACK if it does not need attention
  - STALL in case of error.
- The host will return
  - an ACK if everything is ok.
  - no status if data is corrupted.



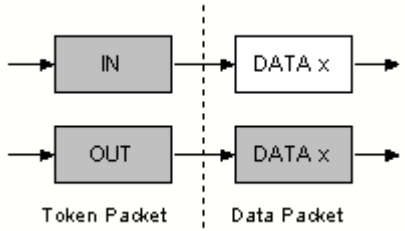
Interrupt OUT Transactions

- The host issues an OUT token followed by a data packet containing the interrupt data.
- The function:
  - ignores the packet in case of error
  - ACK if the endpoint buffer is empty and data is stored
  - NACK if the endpoint buffer is not empty
  - STALL if an error occurred.



Isochronous Transfers

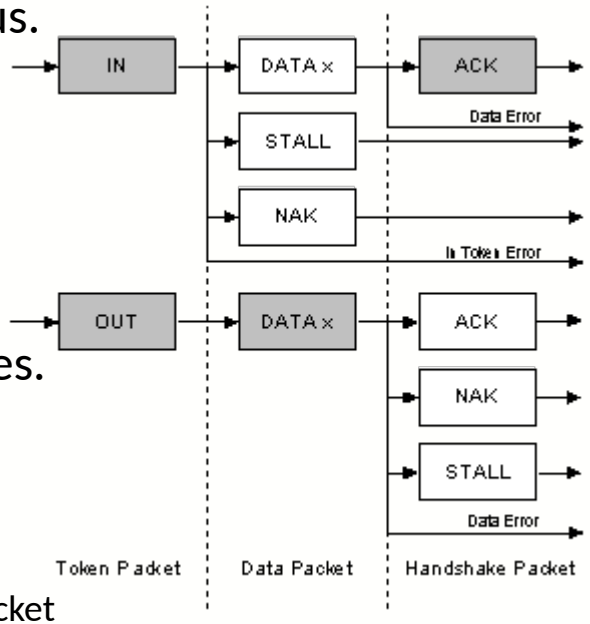
- Isochronous transfers occur continuously and periodically.
- They typically contain time information such as an audio or video stream.
  - A delay or retry of data in an audio stream cause audio glitches. The beat may no longer be in sync.
  - However it is less likely to be noticed by the listener the loss of a packet.
- Isochronous Transfers provide
  - Guaranteed access to USB bandwidth.
  - Bounded latency.
  - Stream Pipe – Unidirectional
  - Error detection via CRC, but no retry or guarantee of delivery.
  - Full & high speed modes only.
  - No data toggling.



- The maximum size data payload is specified in the endpoint descriptor of an Isochronous Endpoint.
  - Maximum of 1023 bytes for a full speed device and 1024 bytes for a high speed device.
  - It is wise to specify a conservative payload size.
  - If you are using a large payload, it may also be to your advantage to specify a series of alternative interfaces with varying isochronous payload sizes.
  - The host cannot enable your preferred isochronous endpoint due to bandwidth restrictions, or to fall back on rather than just failing completely.
- Transactions do not have a handshaking stage.

Bulk Transfers

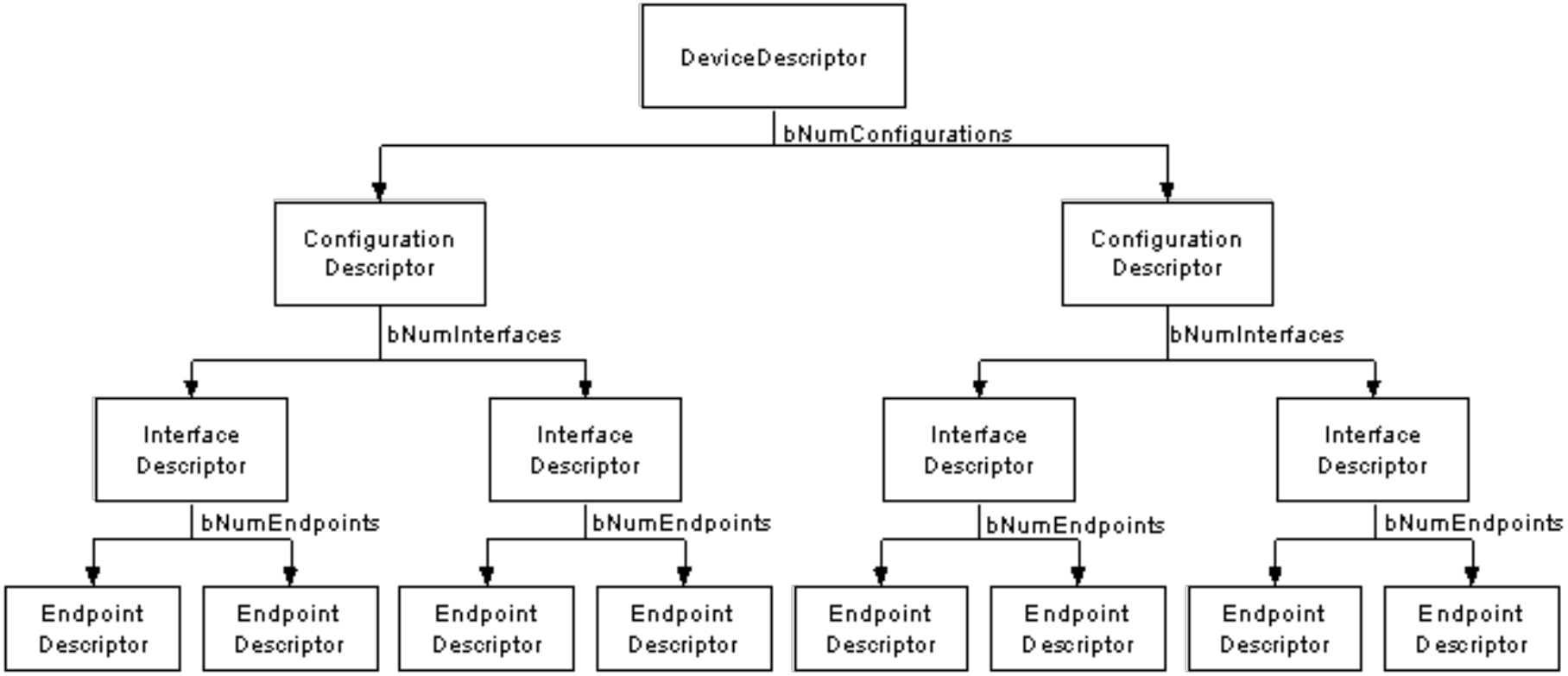
- Can be used for large bursty data (Printers, Scanners, Mass Storages)
- Bulk transfers will use spare un-allocated bandwidth on the bus.
  - Used to transfer large bursty data.
  - Error detection via CRC, with guarantee of delivery.
  - No guarantee of bandwidth or minimum latency.
  - Stream Pipe – Unidirectional
  - Full & high speed modes only.
- Bulk transfers are only supported by full and high speed devices.
- IN:
  - When the host is ready to receive bulk data it issues an IN Token.
  - The function receives the IN token with an error, it ignores the packet
  - If the token was received correctly, the function can either reply with a DATA packet
  - A STALL packet indicating the endpoint has had a error
  - NAK packet indicating to the host that the endpoint is working, but temporary has no data to send.
- OUT:
  - An OUT token followed by a data packet containing the bulk data.
  - If the OUT token or data packet is corrupt then the function ignores the packet.
  - If the function's endpoint buffer was empty and it has clocked the data into the endpoint buffer it issues an ACK
  - If the endpoint buffer is not empty the function returns an NAK.
  - If the endpoint has had an error it returns a STALL.



Bandwidth Management

- The host is responsible for 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:
  - No more than 90% of any frame to be allocated for periodic transfers (Interrupt and Isochronous) on a full speed bus.
  - No more than 80% on high speed buses for periodic transfers.
- The remaining 10% is left for control transfers
- once those have been allocated, bulk transfers will get their slice of what is left.

Device Descriptor

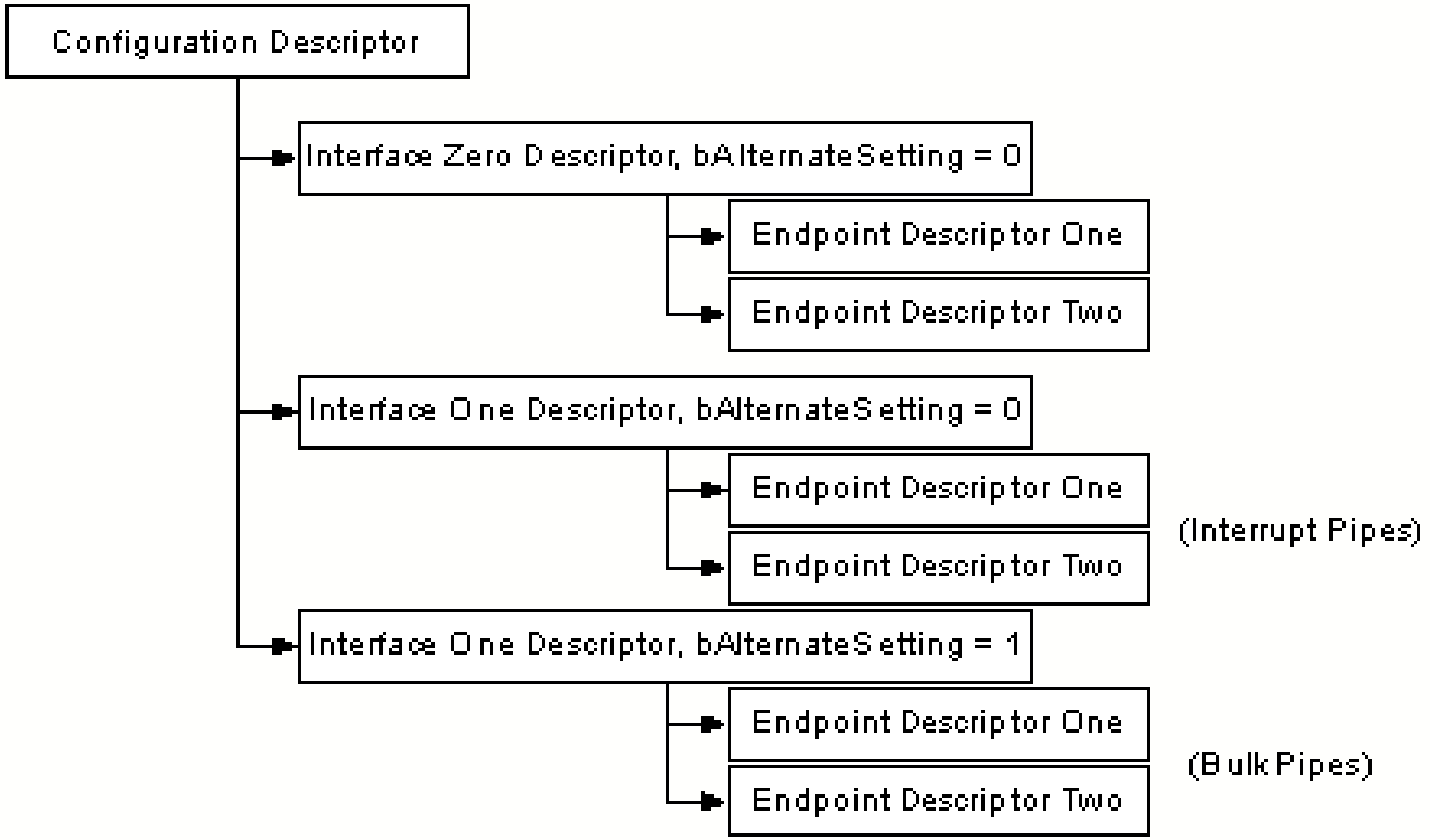


Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of the Descriptor in Bytes (18 bytes)
1	bDescriptorType	1	Constant	Device Descriptor (0x01)
2	bcdUSB	2	BCD	USB Specification Number which device complies too.
4	bDeviceClass	1	Class	Class Code (Assigned by USB Org) If equal to Zero, each interface 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
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 Descriptor

The configuration descriptor specifies:

- how the device is powered,
- what the maximum power consumption is,
- the number of interfaces it has.

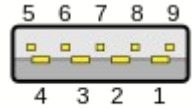


Configuration Descriptor

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

USB 3.0 - 3.1

- USB 3.0
  - il bus "SuperSpeed", a 4,8 Gbit/s, (corrispondenti a 600 MB/s, fino a dieci volte più veloce della versione USB 2.0)



Type A

Pin	Nome segnale	Descrizione
6	SSTX+	Il trasferimento dei dati dall'host al dispositivo
7	SSTX−	SSTX+ ritorno
8	GND	GND
9	SSRX+	Il trasferimento dei dati dal dispositivo all'host
10	SSRX−	SSRX+ ritorno

- USB 3.1
  - 10Gbp/s, rispetto ai 480Mbp/s della tecnologia USB 2.0.
  - supporta vari profili energetici, tra cui uno a 12V, 20V a 5A (60W, 100W)
  - Permette di ricaricare /alimentare praticamente la maggior parte dei dispositivi elettronici, non solo quelli mobili
  - Double faces


Type-C

Endpoint 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) 00 = Data Endpoint 01 = Feedback Endpoint 10 = Explicit Feedback Data Endpoint 11 = Reserved
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.

Alternative configurations