Universal Serial Bus(USB) Interface

Kamalanath Samarakoon

Before USB

- Connecting devices to PC was as issue

- Parallel port
  - Mostly one interface per PC, needed more
  - Used for Zip drives, External CD/HD but slow
- Serial port
  - 2 ports per PC can have more using add on cards
  - Used for modems, printers, plotters, cameras
  - But Slow

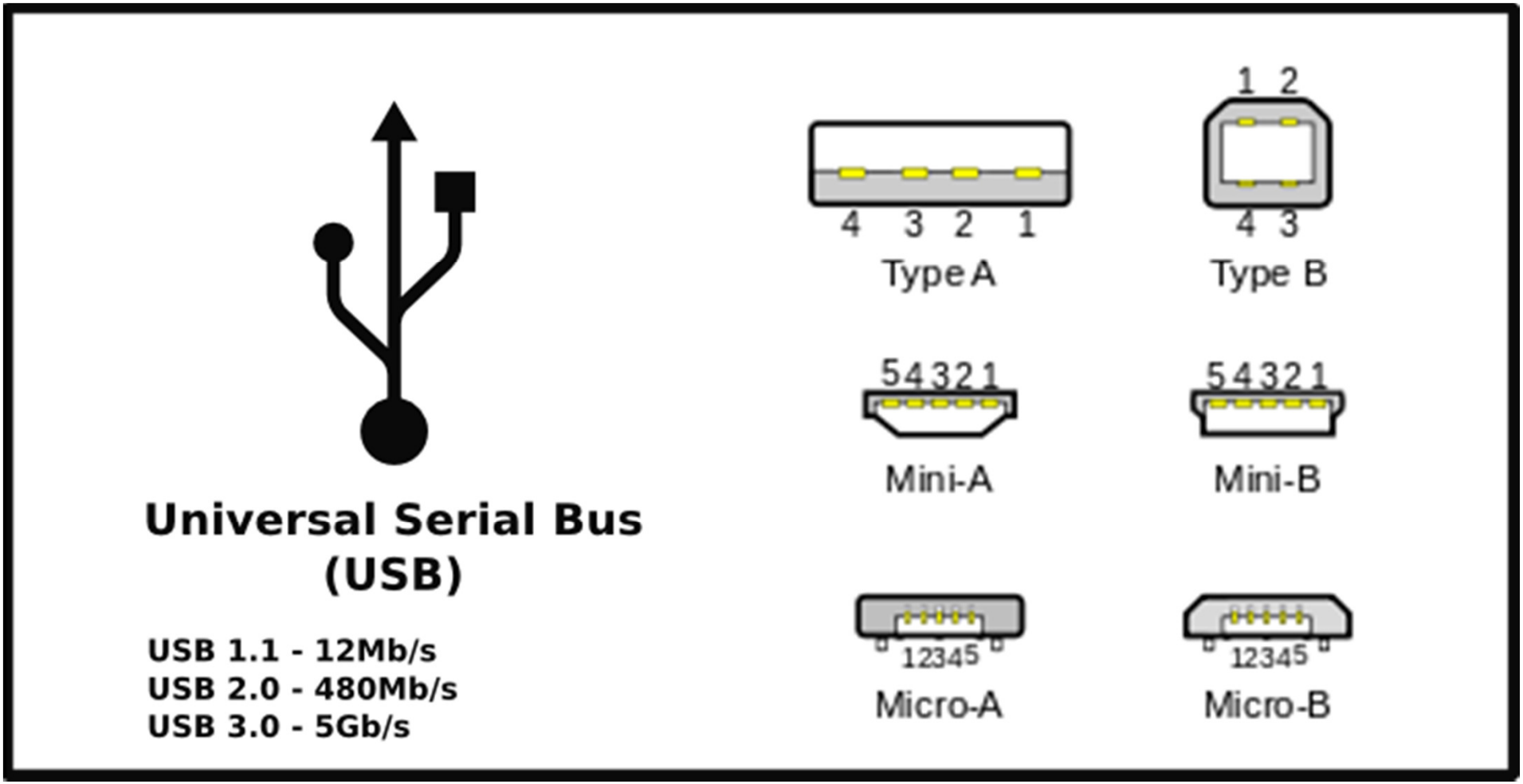
Before USB

- Connecting directly onto PC bus for faster operation
  - Limited number of slots
  - Difficult to configure
  - Need to restart after plug-in
- To solve all above problems, USB was designed

USB

- Single interface
- Standardised
- 127 Devices maximum
  - 650 pages of USB 2.0 document
  - But need not to study all
- Simple connection

Connector
Mechanically not interchangeable



Features

- Host control bus, Mostly PC is the host, one host per bus
- 127 devices
- Individual cable up to 5m
- With a hub Maximum 30m away from the host
- Speeds
  - USB 1.1 1.5 Mb/s (Low speed)
  - USB 1.1 12 Mb/s (Full speed)
  - USB 2.0 480 Mb/s (High speed)

Features (contd..)

- Hot swappable
- Plug and play
  - Dynamically load unload drivers
  - Driver is identified by Product ID(PID)/Vender ID (VID)
  - Supplied by USB implementers forum at a cost
  - Other standard organisations – provide extra VID for non commercial appliations, teaching, research, hobbyist

Topology

- Bus
- Tiered Star

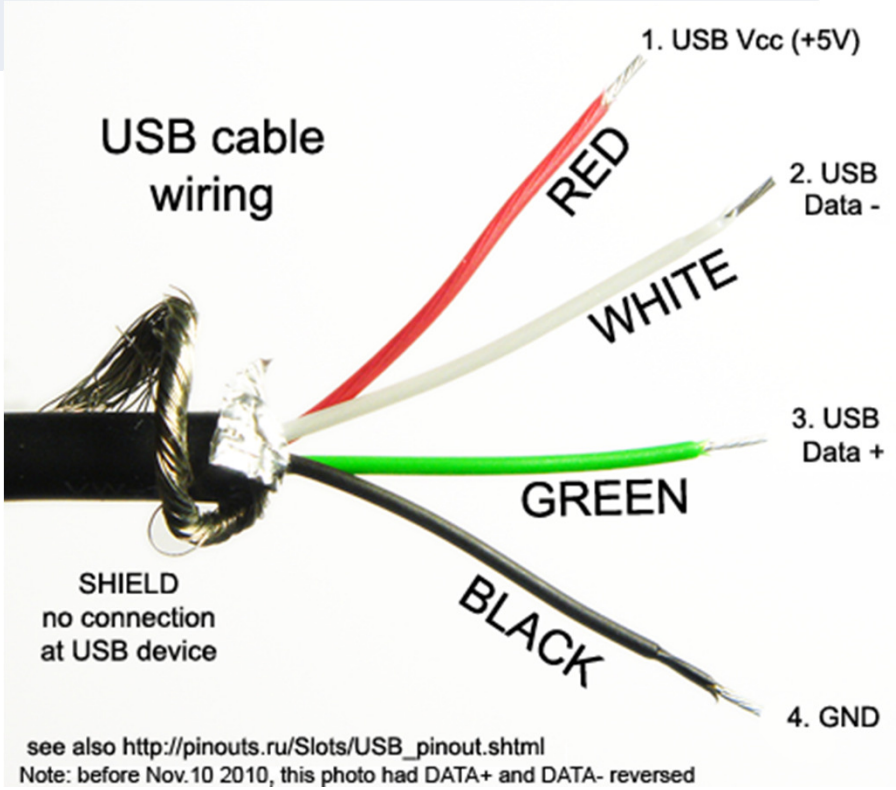
Electrical specification

- Use differential transmission pair for data
- Transmitter
  - Differential 1
    - Pull D+ Over 2.8 V and D- under 0.3 V (with 1.5 k Ohm resister)
  - Differential 0
    - Pull D- Over 2.8 V and D+ under 0.3 V (with 1.5 k Ohm resister)
- Receiver define
  - ‘1’ D+ 200 mv > D-
  - ‘0’ D+ 200 mV< D-

USB 1.1 and 2.0

Pin	Cable colour	Function	Remarks (4 shielded wires0
1	Red	Vbus (5V)	
2	White	D-	Twisted pair
3	Green	D+	
4	Black	Ground	

PC can source 500 mA maximum



USB 3.0 and 3.1
Enhanced Super Speed Bus

- USB 3 5 Gb/s (Super speed) (Colour code blue)
- USB 3.1 10 Gb/s (Super speed+)

With USB 3.1

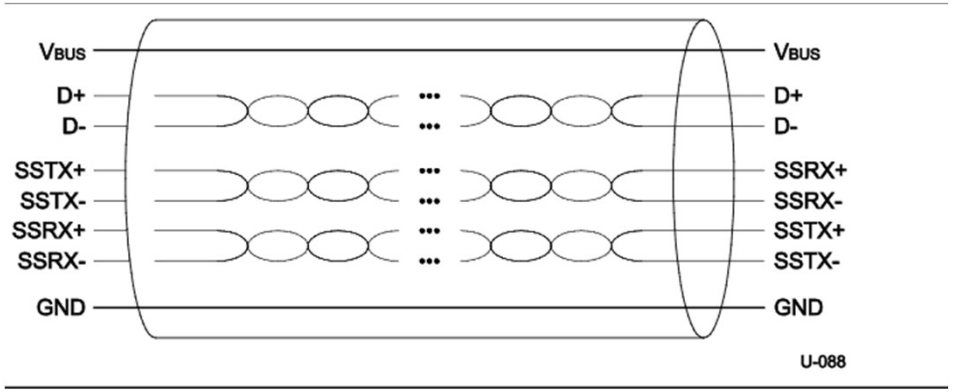
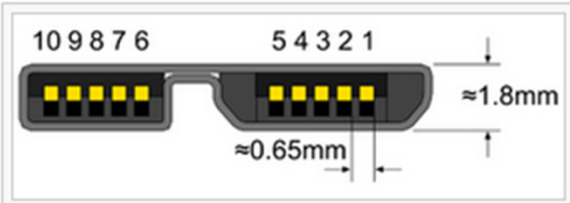


Figure 3-2. USB 3.1 Cable



Micro-B SuperSpeed plug

1. Power (V_{BUS}, 5 V)
2. Data− (D−)
3. Data+ (D+)
4. ID (On-The-Go)
5. GND
6. SuperSpeed transmit− (SSTx−)
7. SuperSpeed transmit+ (SSTx+)
8. GND
9. SuperSpeed receive− (SSRx−)
10. SuperSpeed receive+ (SSRx+)

- USB 2.0 Half duplex two wire differential
- USB 3.0 Added
  - Dual Simplex four wire differential

USB Type C

Type-C

- Connects host and devices
- 24 pin double sided (similar to micr B)
- 4 power
- 4 ground
- 2 differential pairs of non-Super Speed
- 4 differential pairs of high speed bus
- 2 side band use pins
- 2 configuration pins for cable orientation detection

Table 3-1. Comparing Enhanced SuperSpeed Bus to USB 2.0 Bus

Characteristic	Enhanced SuperSpeed USB	USB 2.0
Data Rate	Gen 1 (5.0 Gbps), Gen 2 (10 Gbps)	low-speed (1.5 Mbps), full-speed (12 Mbps), and high-speed (480 Mbps)
Data Interface	Dual-simplex, four-wire differential signaling separate from USB 2.0 signaling Simultaneous bi-directional data flows	Half-duplex two-wire differential signaling Unidirectional data flow with negotiated directional bus transitions
Cable signal count	Six: Four for Enhanced SuperSpeed data path, two for USB 2.0 data path	Two: Two for low-speed/full-speed/high-speed (USB 2.0) data path
Bus transaction protocol	Host directed, asynchronous traffic flow Packet traffic is explicitly routed	Host directed, polled traffic flow Packet traffic is broadcast to all devices.
Power management	Multi-level link power management supporting idle, sleep, and suspend states. Link-, Device-, and Function-level power management.	Port-level suspend with two levels of entry/exit latency Device-level power management
Bus power	Same as for USB 2.0 with a 50% increase for unconfigured power and an 80% increase for configured power	Support for low/high bus-powered devices with lower power limits for un-configured and suspended devices
Port State	Port hardware detects connect events and brings the port into operational state ready for Enhanced SuperSpeed data communication.	Port hardware detects connect events. System software uses port commands to transition the port into an enabled state (i.e., can do USB data communication flows).
Data transfer types	USB 2.0 types with Enhanced SuperSpeed constraints. Bulk has streams capability (refer to Section 3.3)	Four data transfer types: control, bulk, Interrupt, and Isochronous

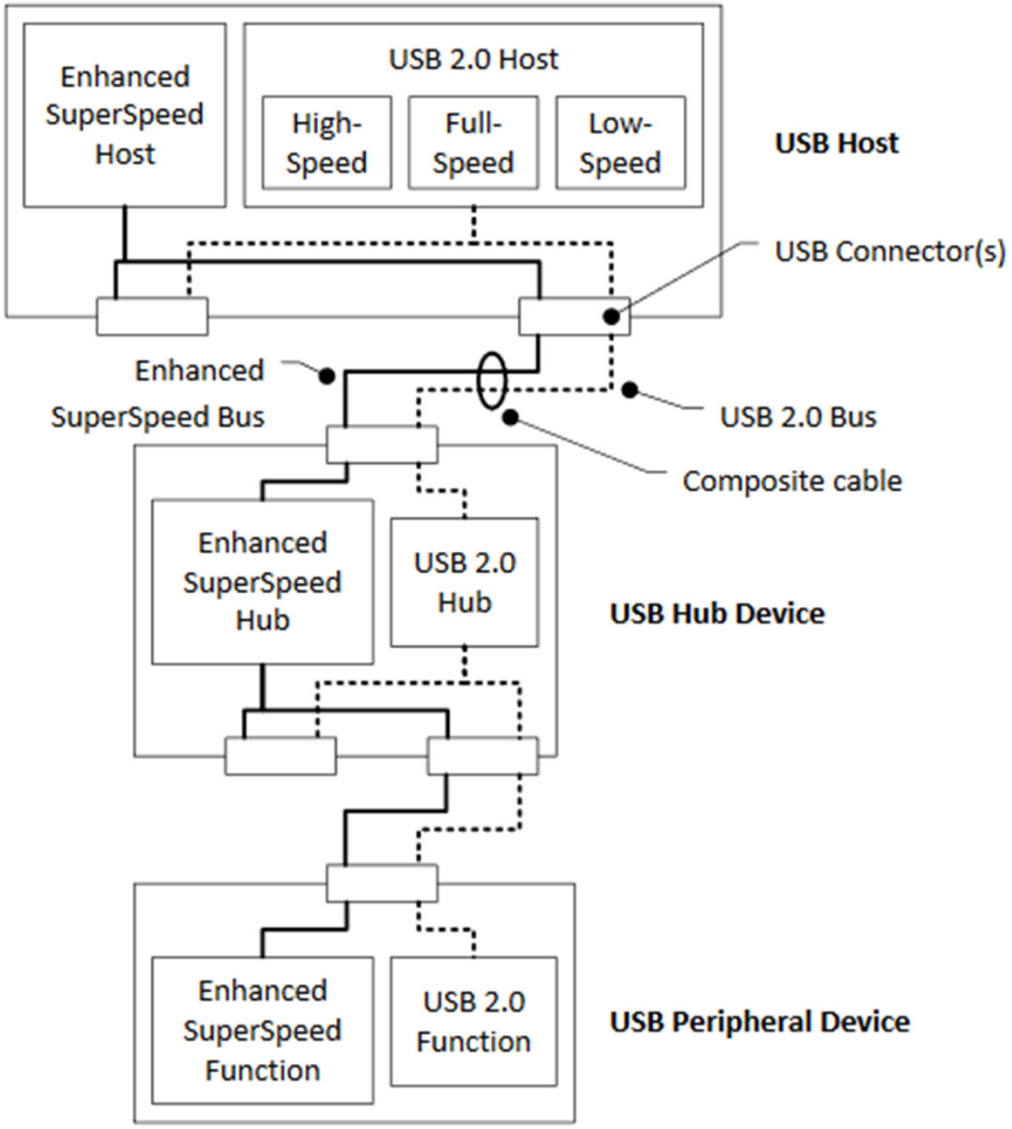


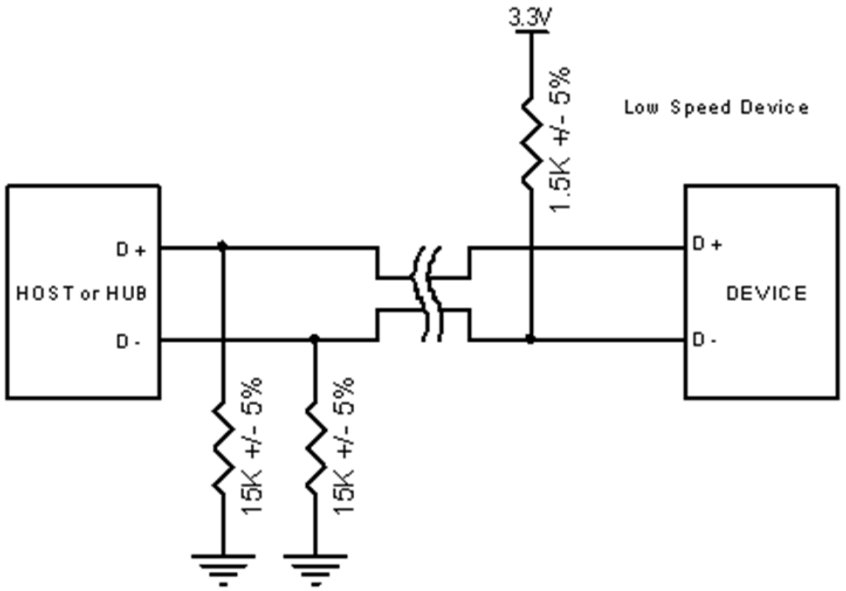
Figure 3-1. USB 3.1 Dual Bus System Architecture

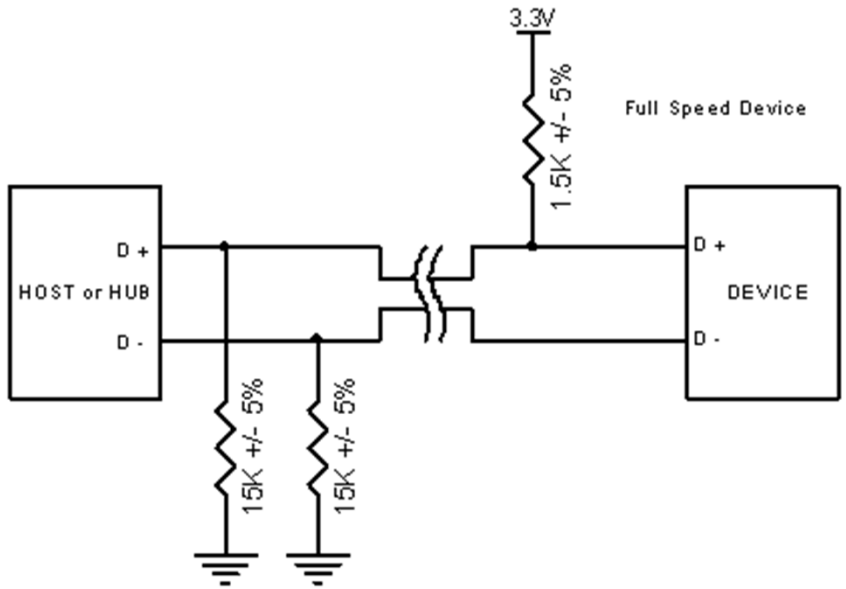
Detecting whether a device is plugged in or not

- The presence of a pull up resistor is used by the host to detect the presence of the devices connected to the port
- If no pull up resistor is connected, host assume nothing is connected

Speed Identification
Ver. 1.1 (Low and Full speed)

- Location of the pull-up resistor is used to identify the speed
  - D- Low speed device
  - D+ Full speed device





Speed Identification
Ver. 2.0 (High speed)

- Connect as a Full speed device and once identified that it can run at high speed remove the pull-up resistor
- (Switchable resistor such as a transistor)

Power specifications

- Three classes of USB functions (one device can have more that one function so use the term function rather than device)
  - Low power bus powered
  - High power bus powered
  - Self powered

Low power

- Draw all its power from the Vbus
- Cannot draw more than one unit load
- 1 unit load= 100 mA

High power

- Draw all its power from the Vbus
- Cannot draw more than one unit load until it has been configured
- After that it can draw maximum 5 unit loads (500 mA)

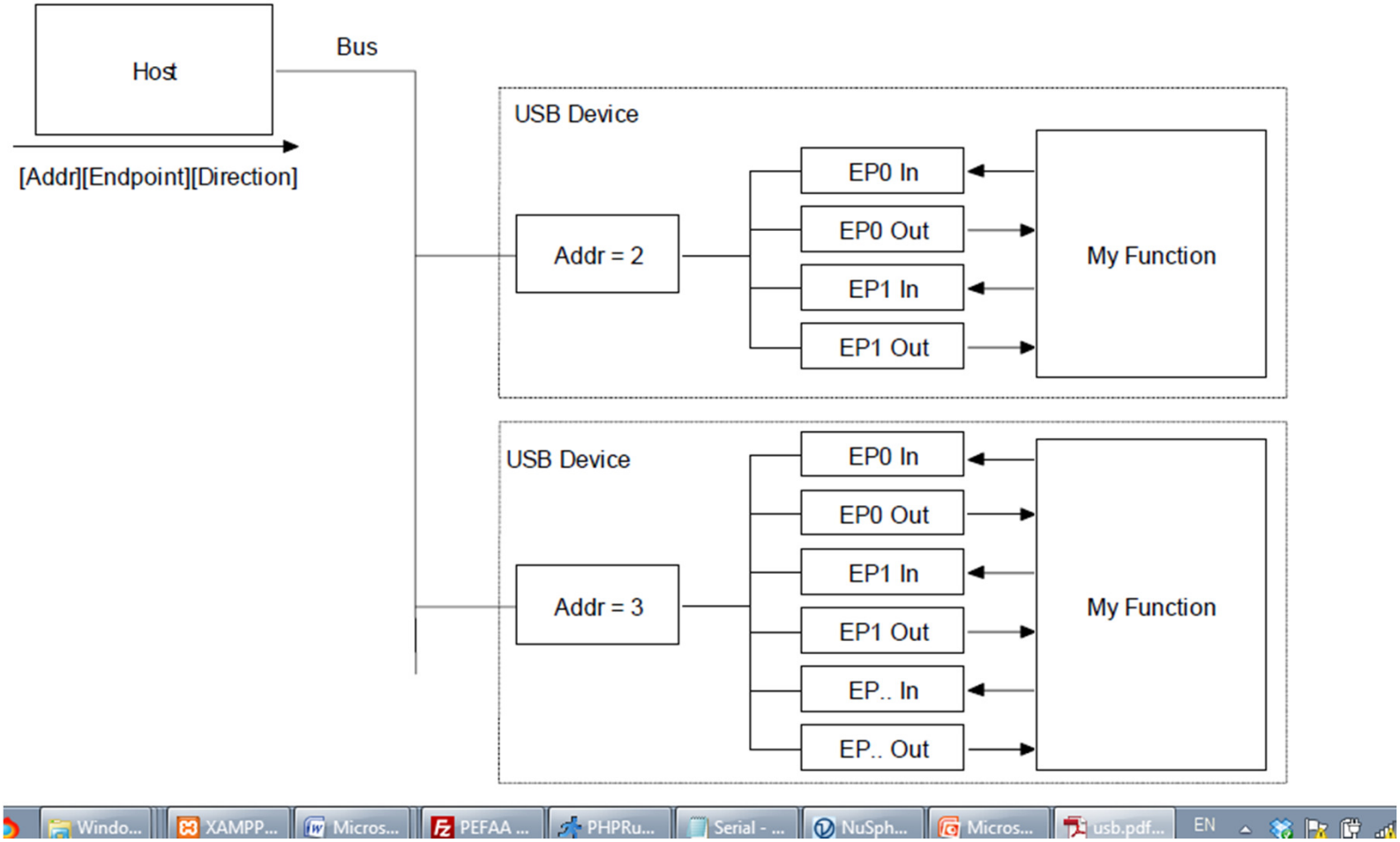
Self powered

- May draw 1 unit load from the bus
- Draw rest of the power from an external source
- Even external power failed, the 1 unit load should not be exceeded

USB 3.0

- 150 mA unit load
- 6 maximum unit loads (900 mA) current draw per USB channel
- CRC-16/CRC-32 For Header/Payload Protection

Functions and End points



Functions and End points

- Most functions have a series of buffers, typically 8 bytes long.
- Each buffer will belong to an endpoint - EP0 IN, EP0 OUT etc.
- E.g: The host sends a device descriptor request. The function hardware will read the setup packet and determine from the address field whether the packet is for itself
- 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.
- It will then send a handshake packet to acknowledge the reception of the byte 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.

End points

- 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.

Protocol

- Unlike other serial protools (RS232, 485 etc.) USB defines layers of protocol (like Ehernet)
- Token based protocol
- Host centric (like master slave)
  - Host initiate all transactions
- Packet based protocol
  - 4 types of packets
- LSB is transmitted first

4 types of packets

- Token packet
  - Generated by the host
  - Defines what to follow
  - Transaction Read/Write
  - Device address and EndPoint
- Data packet (optional)
  - Payload
- Status packet
  - Handshaking method
  - Acknowledgement and to do error correction
- Special packets

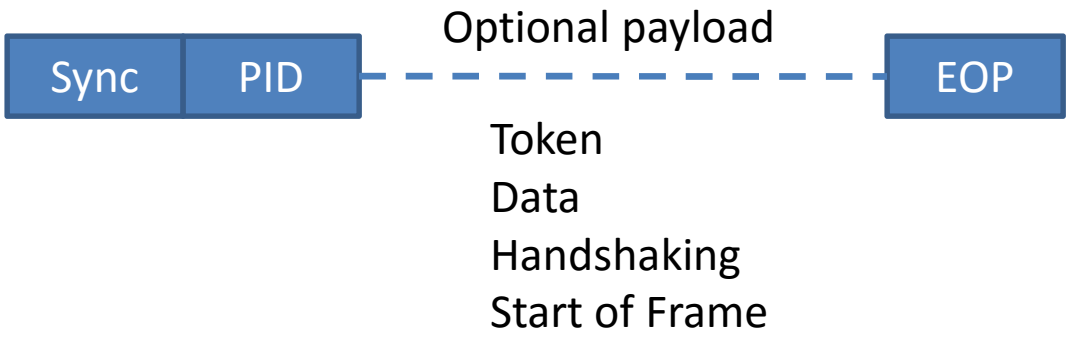
Costruction of packets

- Each packet consists of several fields

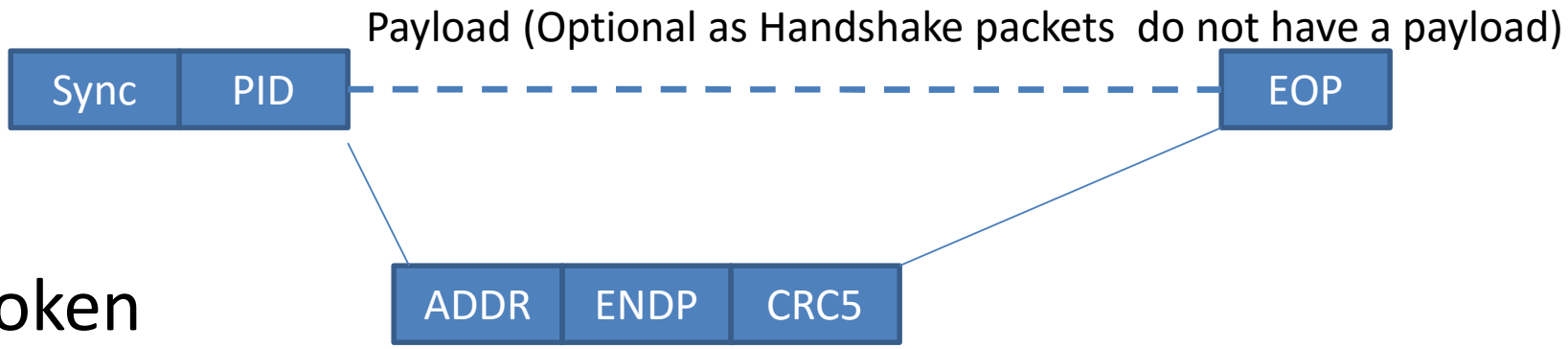
- Field types
  - Sync, PID, ADDR, ENDP,CRC,EOP

Costruction of packets

- Number of fields depend on the packet type
- Start with Sync and PID fields and ends with EOP field



Payload (bits that needs to be send) Sync, PID and EOP are for suporting



- Token
- Data

- Handshake no payload field
- Start of Frame (SOF)

Sync

- All packets must start with a sync field. The sync field is 8 bits long, which is used to synchronise the clock of the receiver with the transmitter. The last two bits indicate where the PID fields starts.

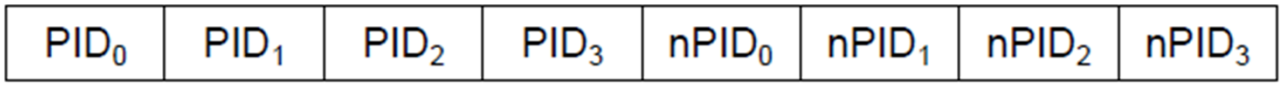
PID: Packet ID

- This field is used to identify the type of packet that is being sent. The following table shows the possible values

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

PID format

- There is 4 bits to the PID, however to insure it is received correctly, the 4 bits are complemented and repeated, making an 8 bit PID in total.



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 endpoint additional addresses on top of the default pipe. (4 Endpoints Max)

CRC

- Cyclic Redundancy Checks are performed on the data within the packet payload for error detecting
- Token packets have a 5 bit CRC
- Data packets have a 16 bit CRC

EOP

- End of packet. Signalled by a Single Ended Zero (SE0) (both lines are zero) for approximately 2 bit times followed by a J for 1 bit time.

- Note: High speed and low speed as opposite polarities for H and L. J state is the state at idel (the line with the pull-up resistor is high, and the other line is low)

Token Packets

- There are three types of 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.**

Data Packets

- There are two types of data packets each capable of transmitting 0 to 1023 bytes of data.
- **Data0**
- **Data1**

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 cannot send nor received data temporary.** Also used during interrupt transaction 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.**

Start of Frame Packets

- The SOF packet consisting of an 11-bit frame number is sent by the host every 1mS ± 500nS.
- The frame is used as a time frame in which to schedule the data transfers which are required. For example, an isochronous endpoint will be assigned one transfer per frame.

Four transfer/endpoint types

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

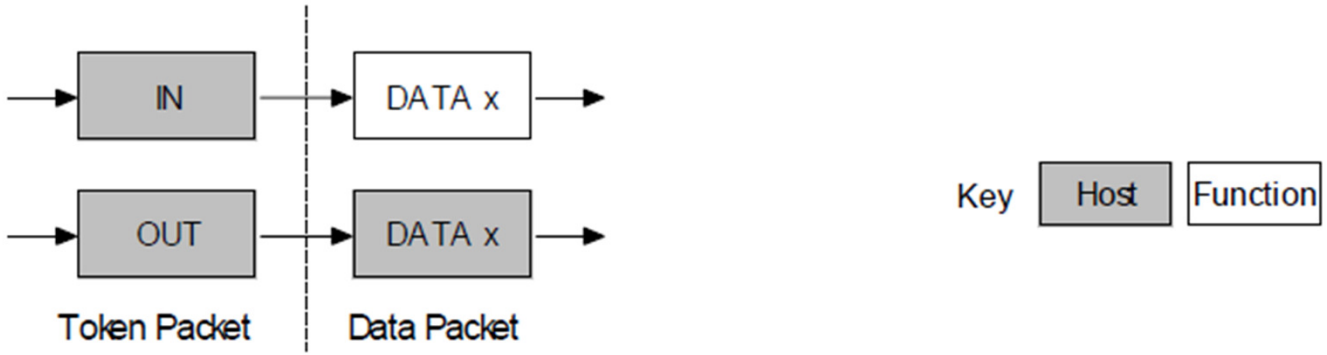
Isochronous transfers

- Occur continuously and periodically.
- 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

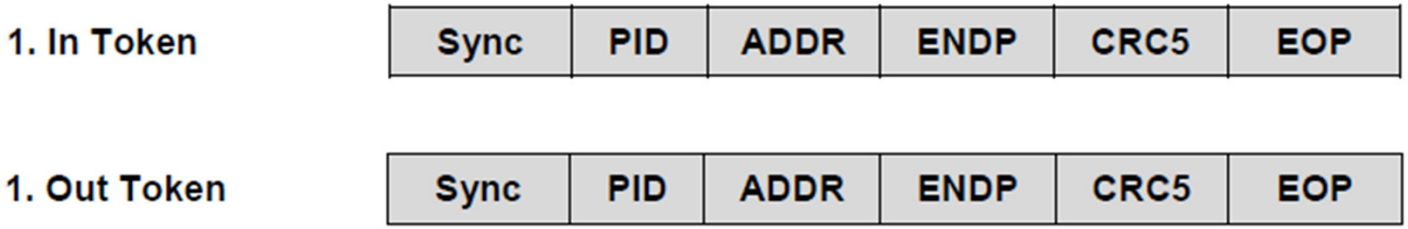
- Guaranteed access to USB bandwidth.
- Bounded latency.
- Stream Pipe – Unidirectional
- Error detection via CRC, but no retry or guarantee of delivery.
- Full & high speed modes only.
- No data toggling.

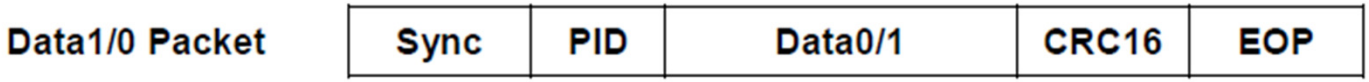
Format of an Isochronous IN and OUT transaction



- The maximum size data payload is specified in the endpoint descriptor of an Isochronous Endpoint.
- This can be up to a maximum of
  - 1023 bytes for a full speed device
  - 1024 bytes for a high speed device
- Isochronous transactions do not have a handshaking stage and cannot report errors or STALL/HALT conditions.

Format of Packets





Bulk transfer

- 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.

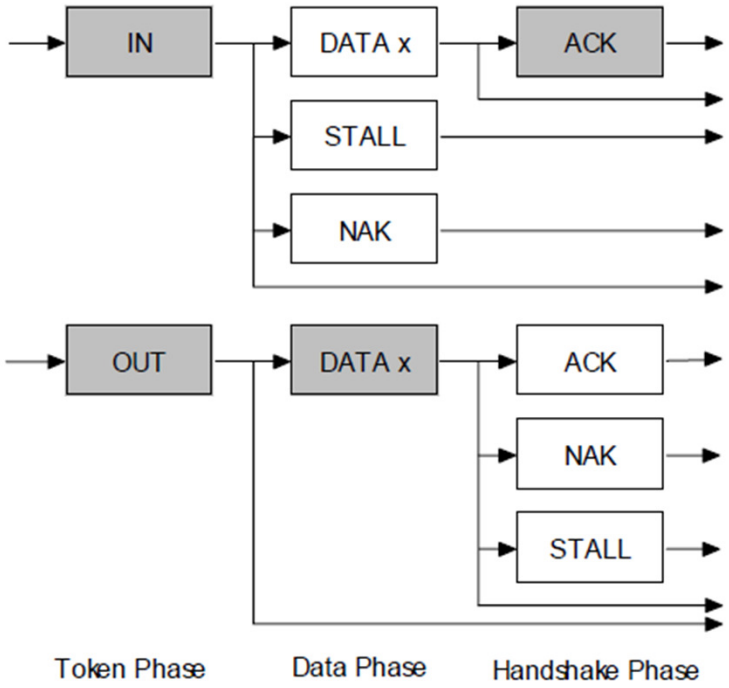
Bulk transfer

- Example: A print-job sent to a printer or an image generated from a scanner.
- 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.
- Use spare un-allocated bandwidth on the bus after all other transactions have been allocated.
- If the bus is busy with isochronous and/or interrupt then bulk data may slowly trickle over the bus. As a result Bulk transfers should only be used for time insensitive communication as there is no guarantee of latency.

Bulk transfer

- Only supported by full and high speed devices.
  - 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.

Format of Bulk Transfer



- IN: When the host is ready to receive bulk data it issues an IN Token. If 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 containing the bulk data to be sent,

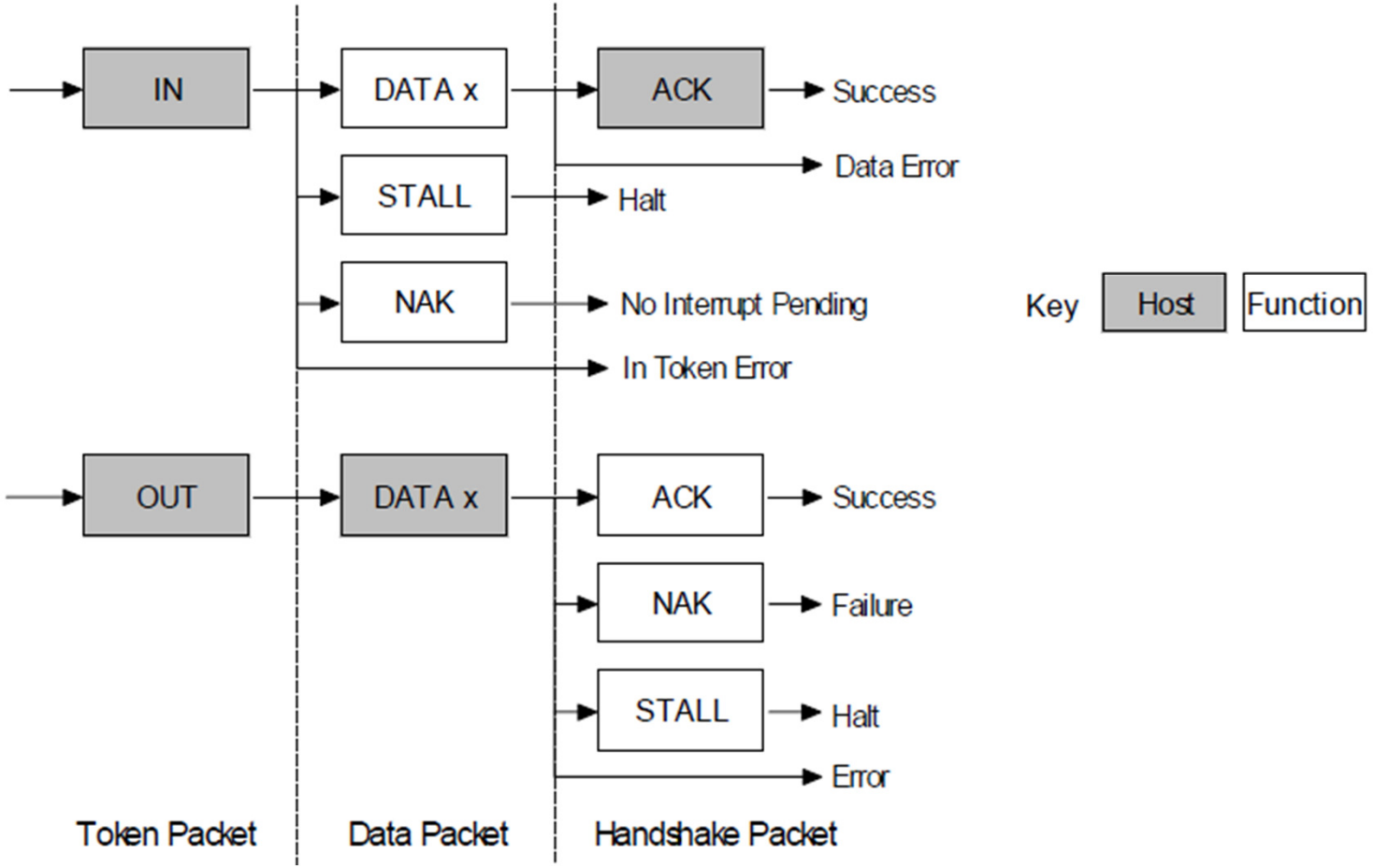
- OUT: When the host wants to send the function a bulk data packet, it issues an OUT token followed by a data packet containing the bulk data. If any part of the OUT token or data packet is corrupt then the function ignores the packet. If the function's endpoint buffer was empty and it has clocked the data into the endpoint buffer it issues an ACK informing the host it has successfullyreceived the data.

Inrerrupt trasfers

- This if for sending an interrupt to host
- However when 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 the device until the host polls the USB device asking for data.

- **Interrupt Transfers**
- Guaranteed Latency
- Stream Pipe – Unidirectional
- Error detection and next period retry.

Format


IN
DATA x
ACK
Success
Data Error
STALL
Halt
NAK
No Interrupt Pending
In Token Error
OUT
DATA x
ACK
Success
NAK
Failure
STALL
Halt
Error
Token Packet
Data Packet
Handshake Packet
Key
Host
Function

IN format

- The host will periodically poll the interrupt endpoint. This rate of polling is specified in the endpoint descriptor.
- Each poll will involve the host sending an IN Token.
- If an interrupt has been queued by the device, the function will send a data packet containing data relevant to the interrupt when it receives the IN Token. Upon successful recept at the host, the host will return an ACK.
- If on the other hand a interrupt condition was not present when the host polled the interrupt endpoint with an IN token, then the function signals this state by sending a NAK.

OUT format

- When the host wants to send the device interrupt data, it issues an OUT token followed by a data packet containing the interrupt data.
- If the function's endpoint buffer was empty and it has clocked the data into the endpoint buffer it issues an ACK informing the host it has successfully received the data.
- If the endpoint buffer is not empty due to processing of a previous packet, then the function returns an NAK.

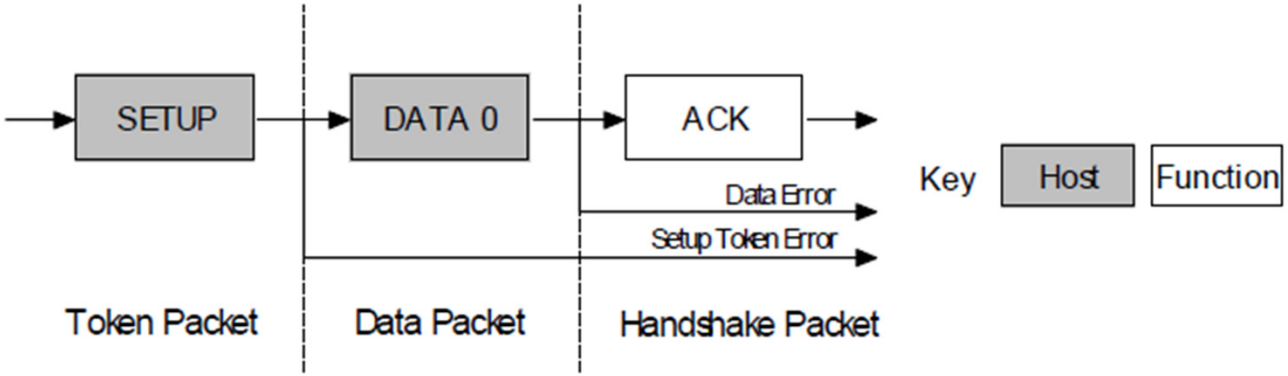
Control Transfers

- Typically used for command and status operations.
- Essential to set up a USB device with all enumeration functions being performed using control transfers.
- 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.

A control transfer can have up to three stages.

- Setup stage
- Data stage
- Status stage

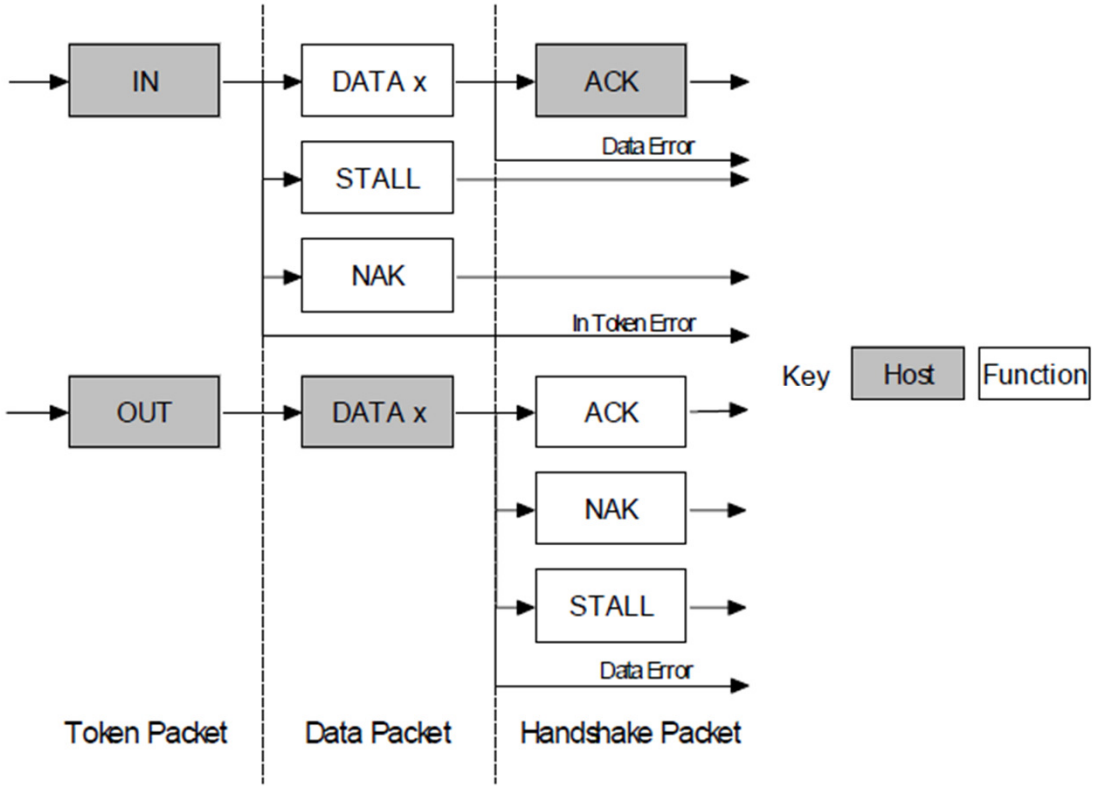
Setup Stage



- Stage where the request is sent.
- Consists of three packets.
- 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.

Data Stage (optional)

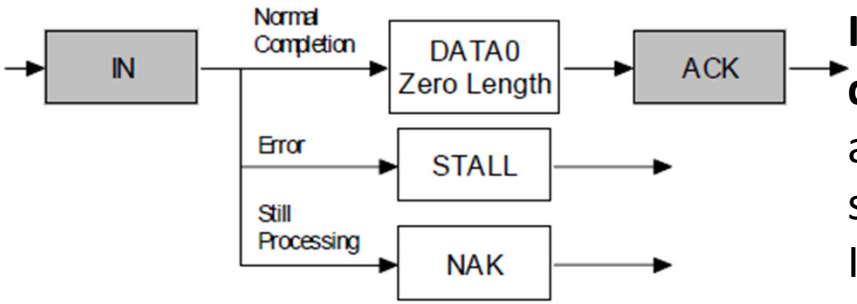
- **IN: When the host is ready to receive control data it issues an IN Token.**
- **OUT: When the host needs to send the device a control data packet, it issues an OUT token**



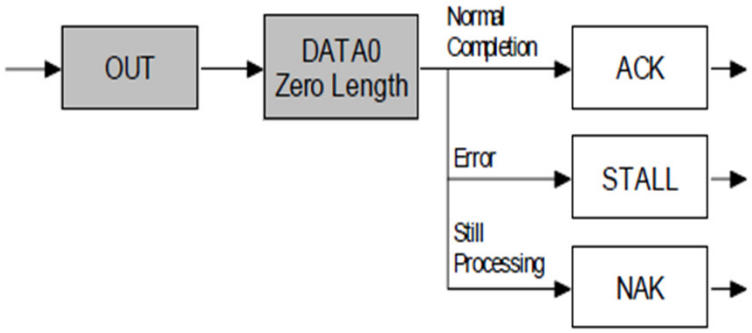
Data Stage (optional)

- 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.

Status Stage

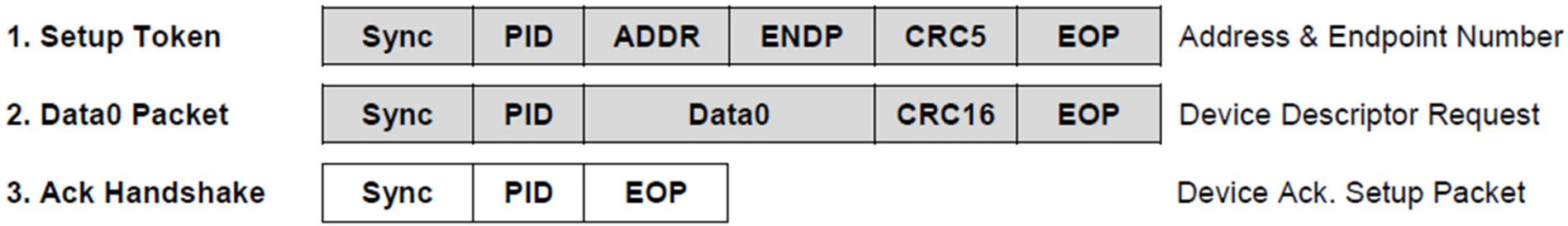


IN : If the host sent OUT token(s) during the data stage to transmit data, the function will acknowledge the successful recept of data by sending a zero length packet in response to an IN token. An ACK indicates the function has completed the command is now ready to accept another command.

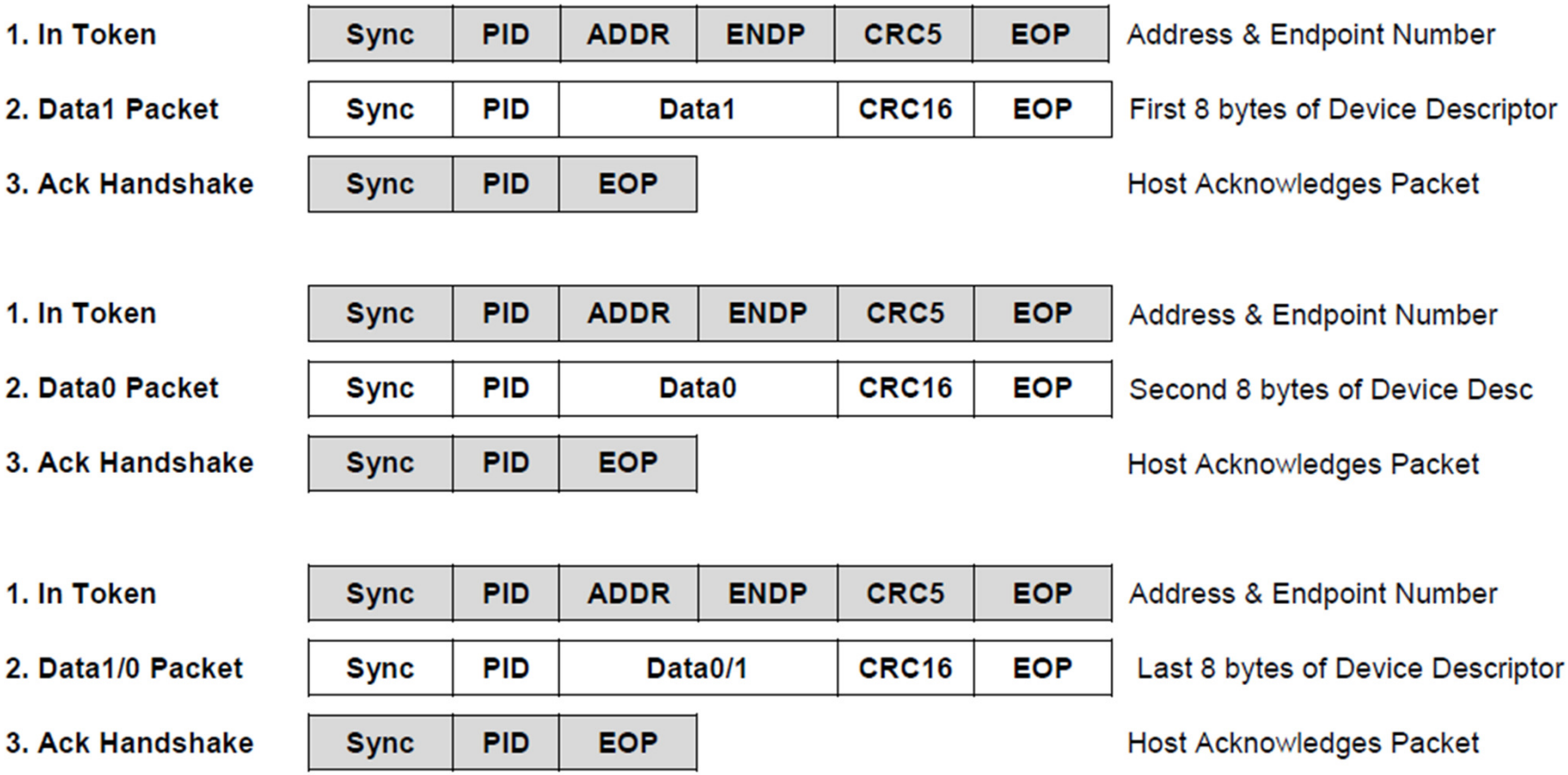


OUT: If the host sent IN token(s) during the data stage to receive data, then the host must acknowledge the successful recept of this data. This is done by the host sending an OUT token followed by a zero length data packet. The function can now report its status in the handshaking stage. An ACK indicates the function has completed the command is now ready to accept another Command..

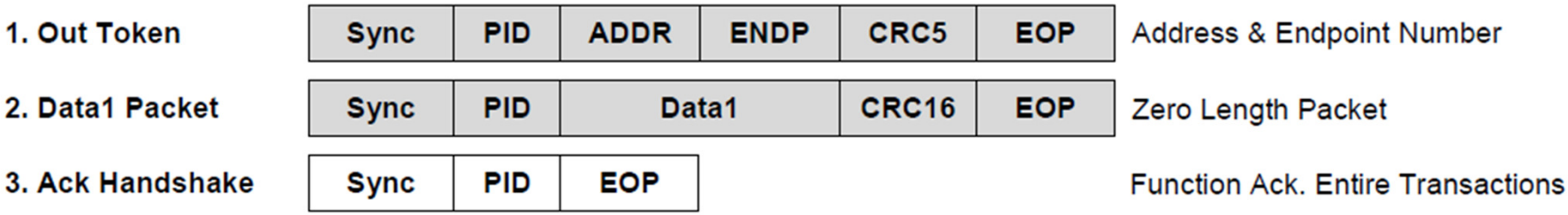
Setup stage


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

Data stage


1. In Token
Sync
PID
ADDR
ENDP
CRC5
EOP
Address & Endpoint Number
2. Data1 Packet
Sync
PID
Data1
CRC16
EOP
First 8 bytes of Device Descriptor
3. Ack Handshake
Sync
PID
EOP
Host Acknowledges Packet
1. In Token
Sync
PID
ADDR
ENDP
CRC5
EOP
Address & Endpoint Number
2. Data0 Packet
Sync
PID
Data0
CRC16
EOP
Second 8 bytes of Device Desc
3. Ack Handshake
Sync
PID
EOP
Host Acknowledges Packet
1. In Token
Sync
PID
ADDR
ENDP
CRC5
EOP
Address & Endpoint Number
2. Data1/0 Packet
Sync
PID
Data0/1
CRC16
EOP
Last 8 bytes of Device Descriptor
3. Ack Handshake
Sync
PID
EOP
Host Acknowledges Packet

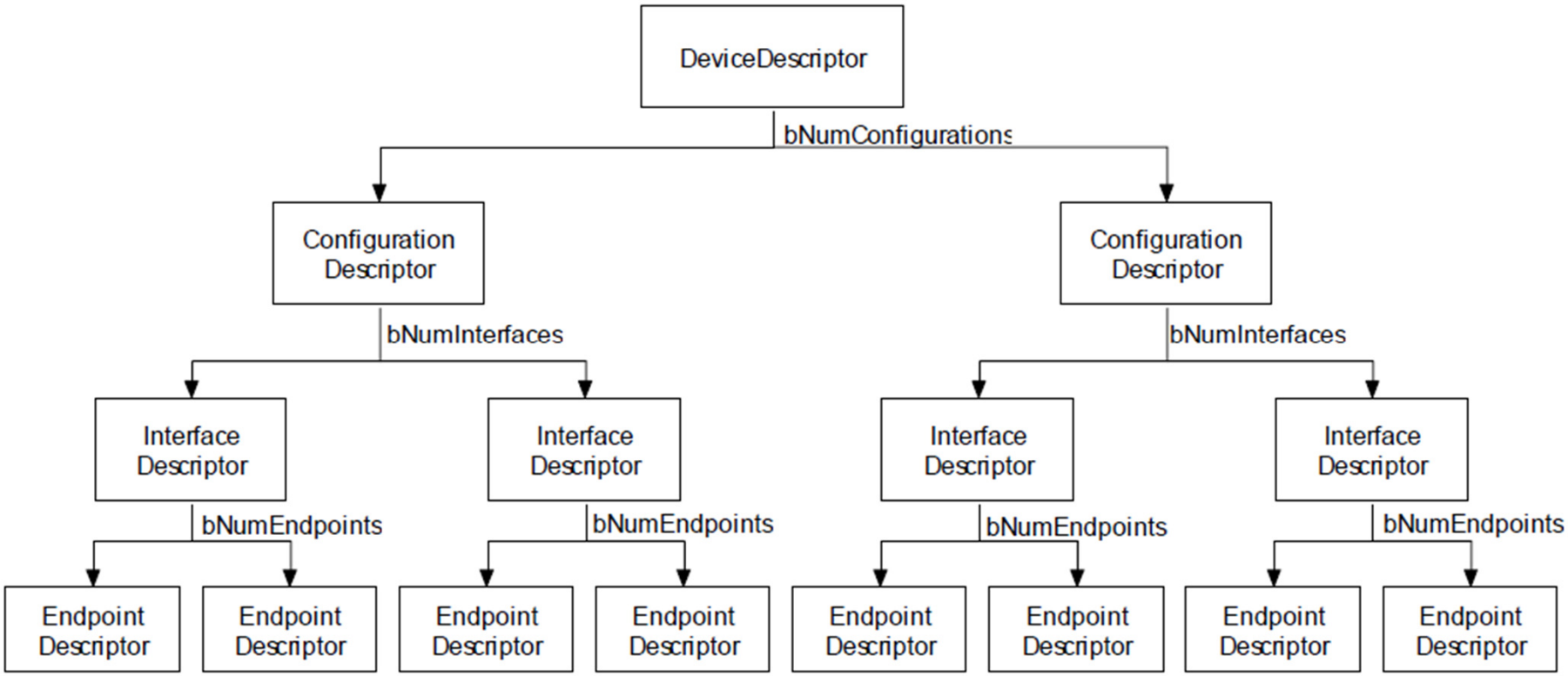
Status stage



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

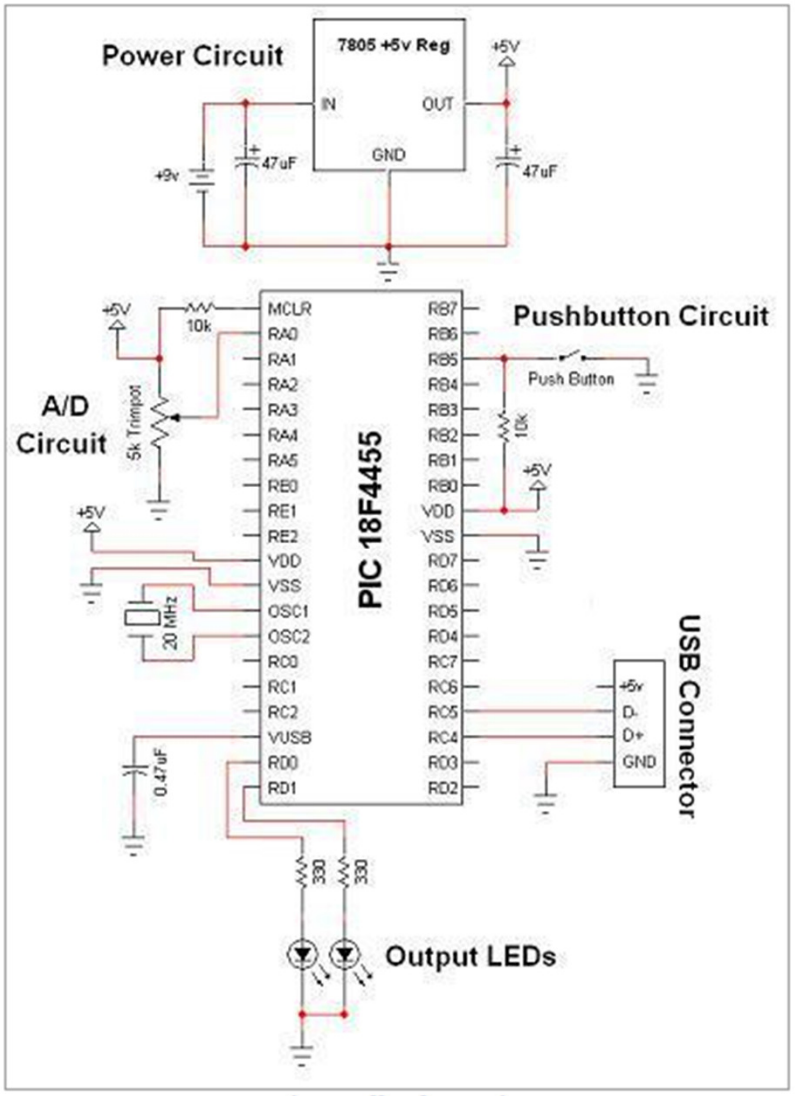
- The more common USB descriptors are
  - Device Descriptors
  - Configuration Descriptors
  - Interface Descriptors
  - Endpoint Descriptors
  - String Descriptors


DeviceDescriptor
bNumConfigurations
Configuration Descriptor
Configuration Descriptor
bNumInterfaces
bNumInterfaces
Interface Descriptor
Interface Descriptor
Interface Descriptor
Interface Descriptor
bNumEndpoints
bNumEndpoints
bNumEndpoints
bNumEndpoints
Endpoint Descriptor
Endpoint Descriptor
Endpoint Descriptor
Endpoint Descriptor
Endpoint Descriptor
Endpoint Descriptor
Endpoint Descriptor
Endpoint Descriptor

USB Descriptor hirearchy

- Has only one device descriptor.
  - Includes information such as what USB revision the device complies to, the Product and Vendor IDs used to load the appropriate drivers and the number of possible configurations the device can have. The number of configurations indicate how many configuration descriptors branches are to follow.

- 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.
- E.g high power bus powered configuration or self powered configuration.



http://www.pyroelectro.com/tutorials/simple_pic_usb_interface/index.html