ARM Cortex core microcontrollers

11th Universal Serial Bus

Balázs Scherer

Budapest University of Technology and Economics
Department of Measurement and Information Systems

Goals

- Cheap standardized interface for PC peripherals (mouse, keyboard …). Originally designed for low data rate communication.
- Designed for low wire count, and power should be transited through the cable
- Ability to connect many devices
- Easy to use from the user's point of view → Plug & Play operation, with dynamic driver loading

<u>Start of the development:</u>

1994 Intel, Microsoft, IBM, Compaq, NEC

Community: USB IF – USB Implementers Forum

Versions

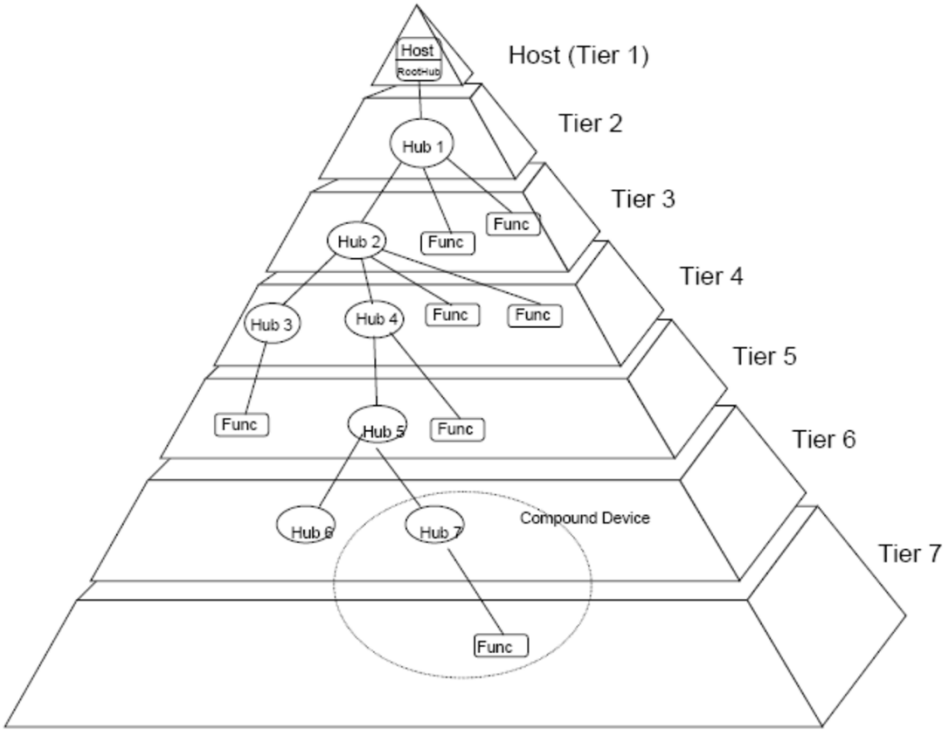
- 1996 USB 1.0 (not getting widespread)
  - There is no hub support
- 1998 USB 1.1
  - Low speed: 1.5 Mbit/sec
  - Full speed: 12 Mbit/sec
- 2001 USB 2.0
  - High speed: 480 Mbit/sec
  - Low-/High-power: 100 mA/500 mA

Versions

- 2008 USB 3.0
  - Super-Speed: 5Gbit/sec (4 Gbit/sec effective bit rate)
  - Low/high power mode: 150 mA/900 mA
  - Battery charger mode (without communication) 1500 mA
  - First devices appears on the market: 2010
  - OS támogatás: Linux, Windows 8
- 2013 USB 3.1
  - Speed increases to 10 Gbit/sec-re

Architecure

- Tired star structure. 3-7 layers (tier): (root + maximum 6 tiers)
- Master – Slave communication: Hub only forwards data
- Messages of the host is visible to every node
- Peripheral answers are routed only towards the Host
- Host-device (master), 1..127 device (slave)
- One HW more address: compound decive
- Different functions on one address: composite device

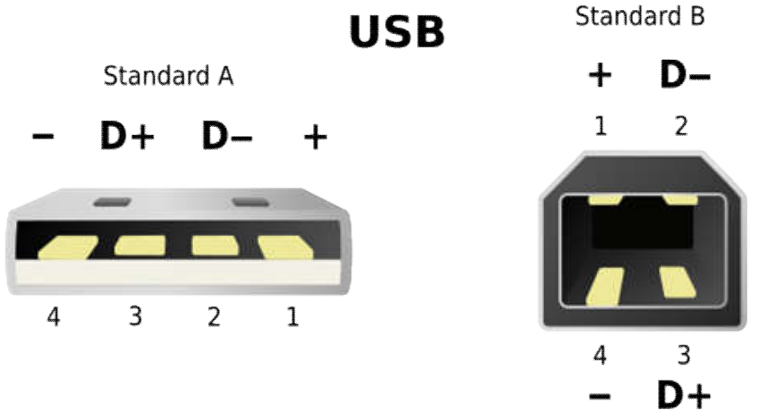


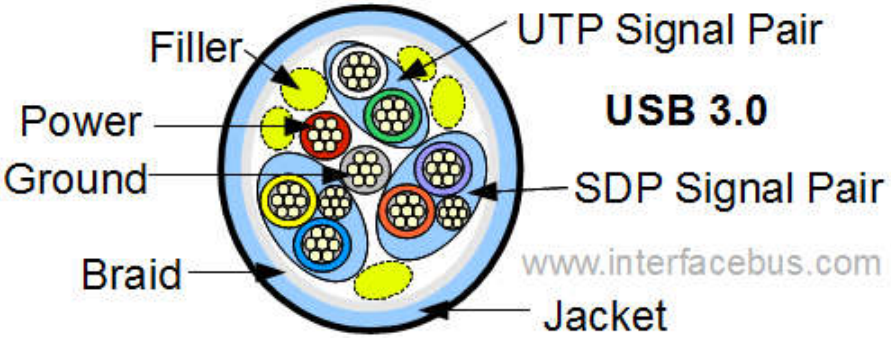
Cables and connectors

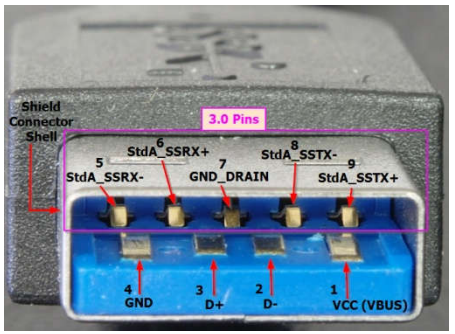
- Connectors:

  A-type: Host side

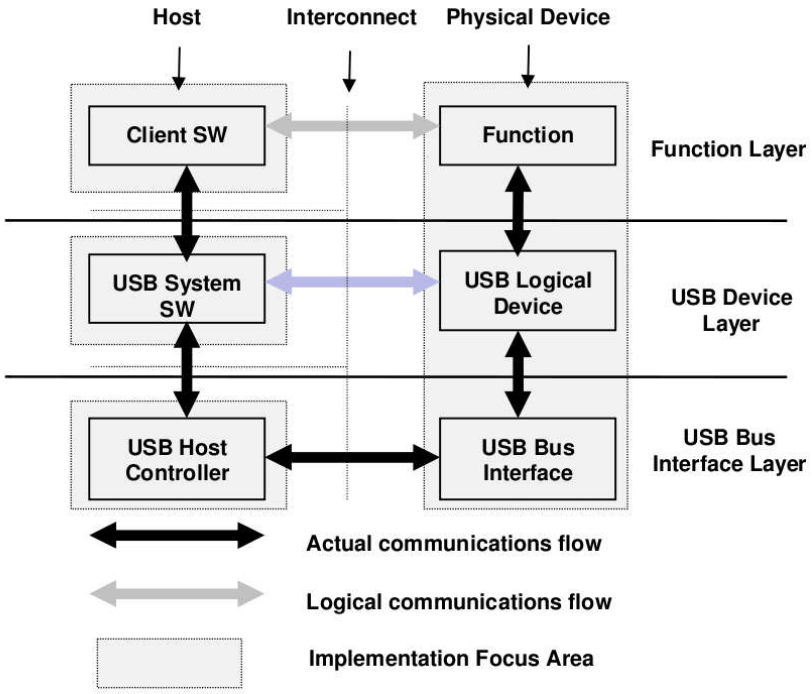
  B- type: Device side
- Max. 5 m-es 2.0 cable lenfgth
- Wires: 5V Power/Ground (Red/Black). Twisted pair data: : D-/D+ (white/green)
- Maximum 3 m cable length for 3.0
  - Additional +2 twisted pair





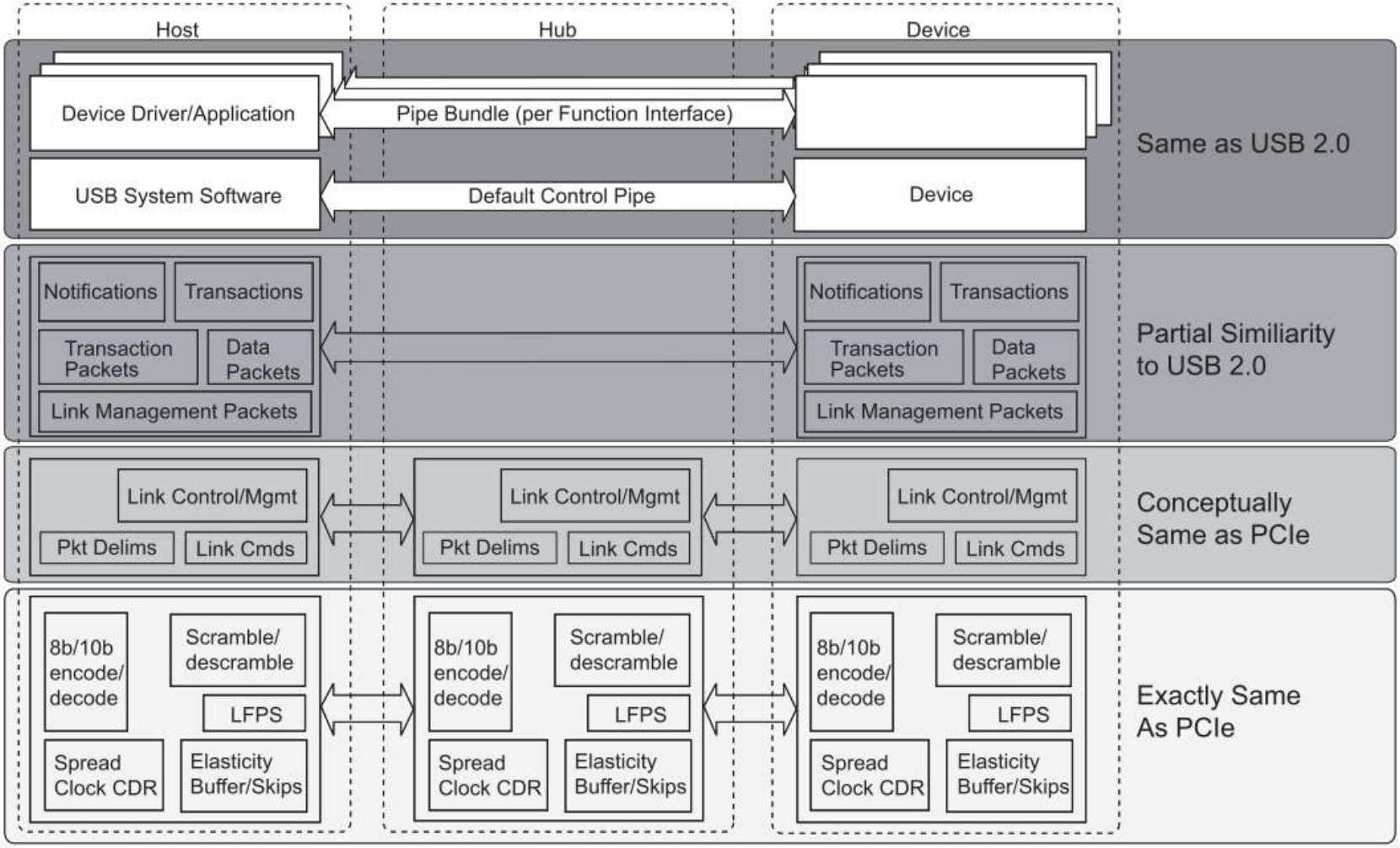


Data Flow USB 2.0 → 3.0

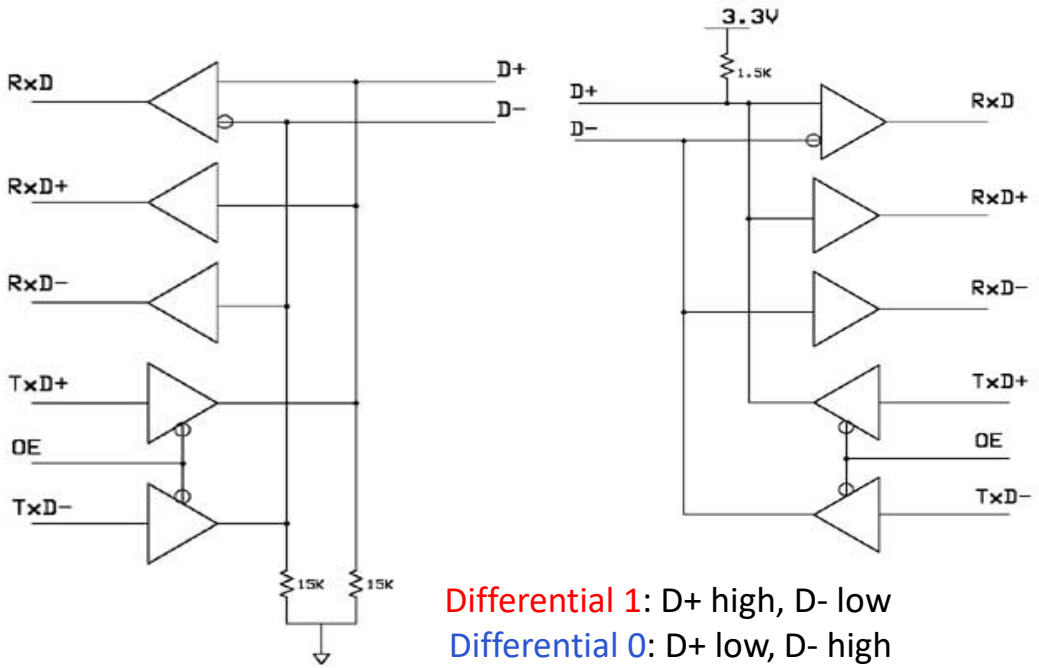


- Physical layers: PCIe based
- Data link layer: based on PCIe
- Protocol layers: modified USB 2.0
- Device/Host level: standardised

Data Flow USB 2.0 → 3.0



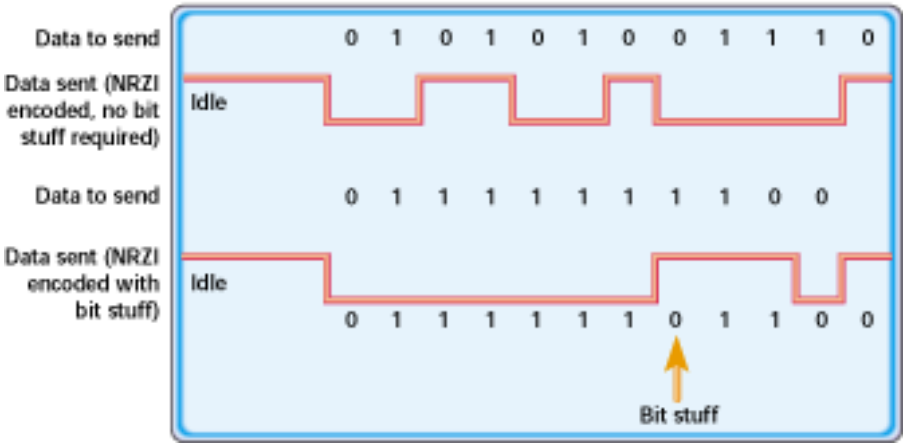
Physical layer – USB 2.0



Differential 1: D+ high, D- low
Differential 0: D+ low, D- high

- Upstream (host side) and downstream (devise side) interface
- Full speed: D+ pull up

- NRZI: singnal change to bit 0
- Bit stuff: after 6 db logical 1 one logical 0



Power supply

Type is descripted in the device configuration

- Low-power bus powered: 4.4-5.25V, maximum 100 mA
- High-power bus powered: In configurated state maximum 500 mA, 4.75-5.25 V
- Self-powered

Shut down state:

- low/high-powered 0.5 mA/2.5 mA

USB 3.0:

- Low/high powered: 150 mA/900 mA
- Buttery charging mode: 1.5 A, there is no communication

Physical layer – USB 2.0

Bus sates:

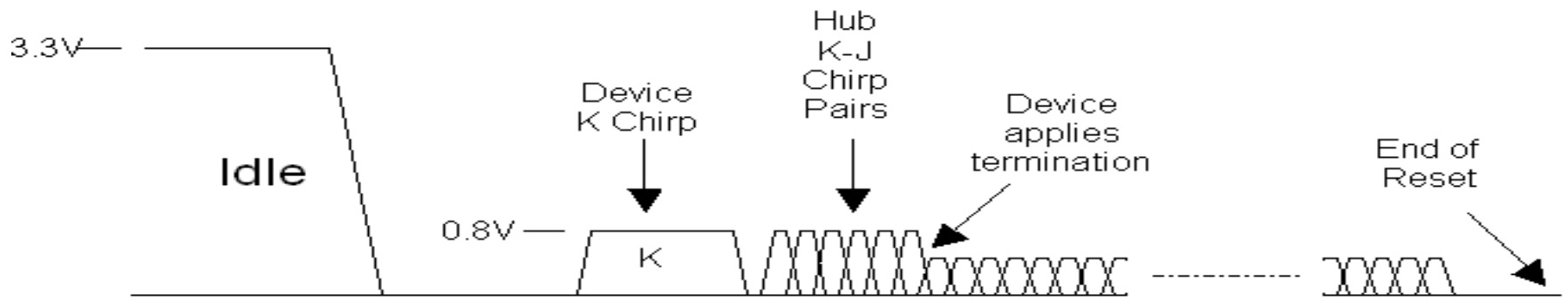
	Full/High		Low		
	D+	D-	D+	D-	LS/FS
SE1	1	1	1	1	Forbidded
SE0	0	0	0	0	liket Detached
J-State	1	0	0	1	like Idle
K-State	0	1	1	0	oposit of J

- K is the driven state
- Reset signal: > 10 ms SE0 (Single Ended 0)
- EOP: End of packet: LS/FS: 2xSE0 + 1xJ, HS: bit stuff error on purpose
- Suspend: >= 3 ms Idle
- Keep Alive: LS: EOP, FS: StartOfFrame packet
- WakeUp: >= 20 ms K (remote wake-up: device wakes up the host)

Physical layer: speed detection

Getting more and more complicated due to compatibility

- Low/Full Speed: based on pull up resistors
- High speed: based on protocol



USB On The Go

The master – slave extension to embedded devices:

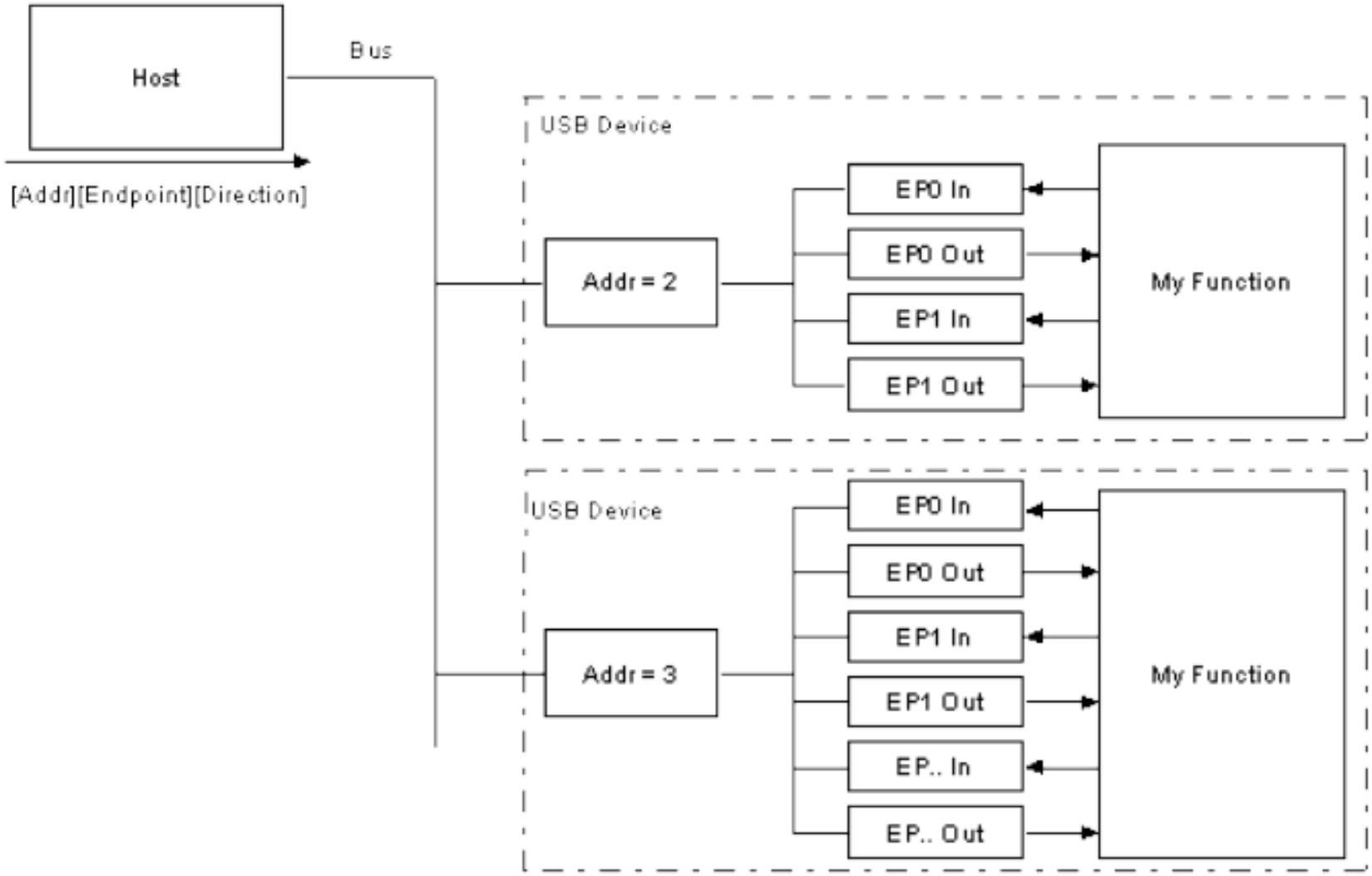
- Detecting the connection without power supplying. Capacitance measurement: *Attach Detection Protocol*
- Embedded device can operate as Device or Host
  (For example. Camera can be a device if connected to a PC or can be a master if a pendrive is connected to it
  Host Negotiation Protocol
- Device can request the host to switch on power
  Session Request Protocol

Data transfer modes

- There are endpoints
- There are endpoints between logical channels: pipe. max. 32 endpoint is allowed pro device
  - Message pipe: bidirectiona, only control transfer – using endpoint 0.
  - Stream pipe: unidirectional
- Stream transfer types

	Many data	Error free	Real-time
Bulk	+	+	-
Isochron	+	-	+
Interrupt	-	+	+

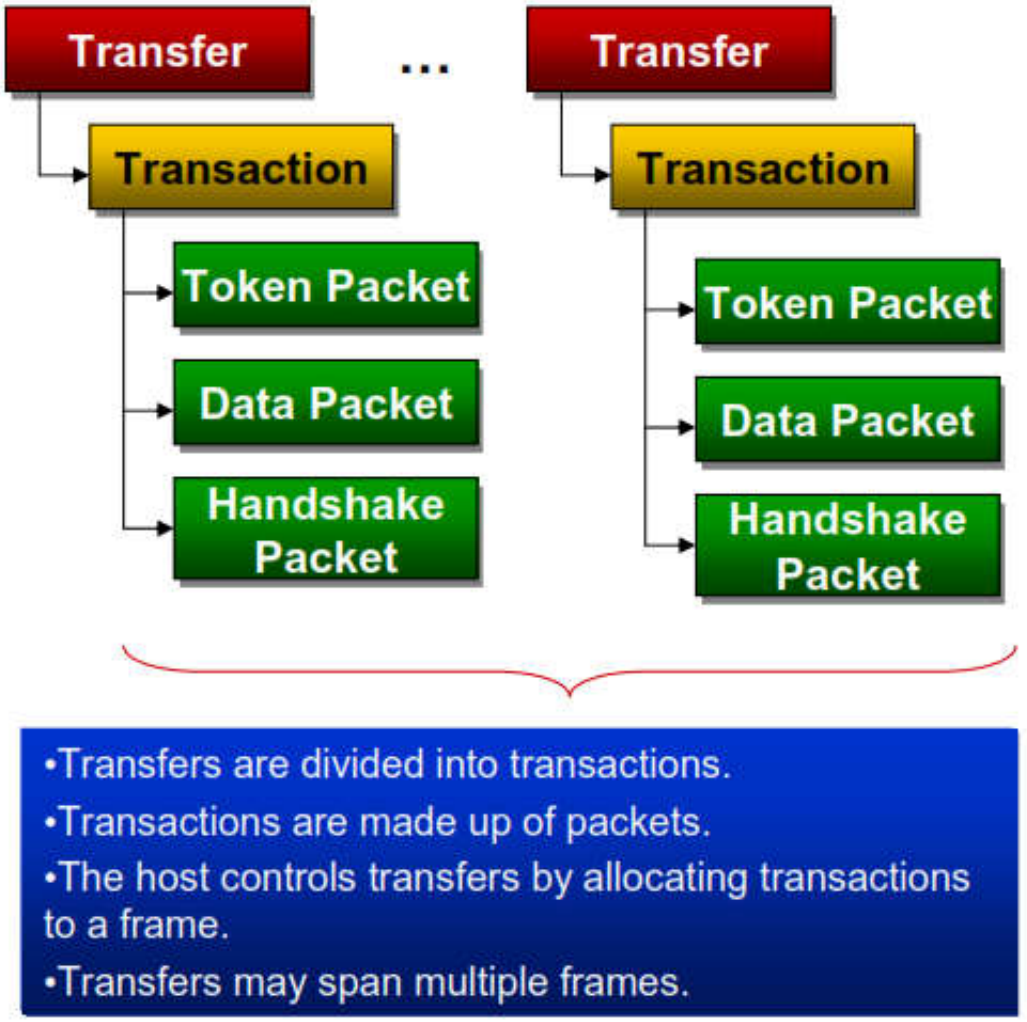
Endpoints and devices



Transfers and transactions

- The USB transfer consists of transactions
- Transactions consist of multiple packets
- Always the Master starts the Transfer

- The Frame is consists of transfers
- One frame is 1ms long
- One frame is 1500 byte in case of FS mode



Data transer

- The transfer build up from transactions
- One Transaction is build up from packets:
  - Token: header packet, determines the type of the transaction
  - Data: optional data
  - Handshake: state information. Isochron transfer do not use it

- USB Packet:

  - SYNC clock syncronisation (8/32 bit LS-FS/HS)
  - PID: Packet ID 4 bit, also transmitted in inverted mode
  - DATA/CRC: PID packet dependent
  - EOP: end of packet

USB packets

PID Type	PID Name	PID<3:0>*
Token	OUT	0001b
	IN	1001b
	SOF	0101b
	SETUP	1101b
Data	DATA0	0011b
	DATA1	1011b
	DATA2	0111b
	MDATA	1111b
Handshake	ACK	0010b
	NAK	1010b
	STALL	1110b
	NYET	0110b
Special	PRE	1100b
	ERR	1100b
	SPLIT	1000b
	PING	0100b
	Reserved	0000b

- Transfer is supported by hardware: fast PID decoding
- LSB first data trasnfer

TOKEN packets

- IN,OUT: direction of the transfer from the Host's point of view
- SETUP: Control transfer

Sync	PID	ADDR	ENDP	CRC5	EOP
	8 bits	7 bits	4 bits	5 bits	

The address 0. is allocated to a new device connection, the End Point 0. is allocated to SETUP packets

- SOF: Start of Frame:

Sync	PID	Frame No.	CRC5	EOP
	8 bits	11 bits	5 bits	

LS: KeepAlive = EOP signal only

FS: 1 ms time limit

HS: 125 us microframe boundary

DATA packets

- DATA0/1: LS/FS transmitted in an alternate style
- DATA2, MDATA: only used to HS isochron transfers

Sync	PID	DATA	CRC16	EOP
	8 bits	(0-1024) x 8 bits	16 bits	

Transfers

Transfer Type	Stages (Transactions)	Phases (Packets)	Comments
Control	Setup	Token	♦Enables host to read configuration information, set addresses and select configurations ♦Only transfer that is required to be supported by peripherals ♦Has both IN and OUT transfers to a single endpoint
		Data	
		Handshake	
	Data (IN or OUT) (optional)	Token	
		Data	
		Handshake	
	Status (IN or OUT)	Token	
		Data	
		Handshake	
Bulk	Data (IN or OUT)	Token	♦Non-critical data transfers ♦Bandwidth allocated to the host ♦Good for file transfer where time critical data is not required
		Data	
		Handshake	
Interrupt	Data (IN or OUT)	Token	♦Periodic transfers on the time base conveyed during enumeration ♦Host guarantees attention before this elapsed time
		Data	
		Handshake	
Isochronous	Data (IN or OUT)	Token	♦Guaranteed delivery time of packets for data streaming ♦No-retransmitting of data allowed
		Data	

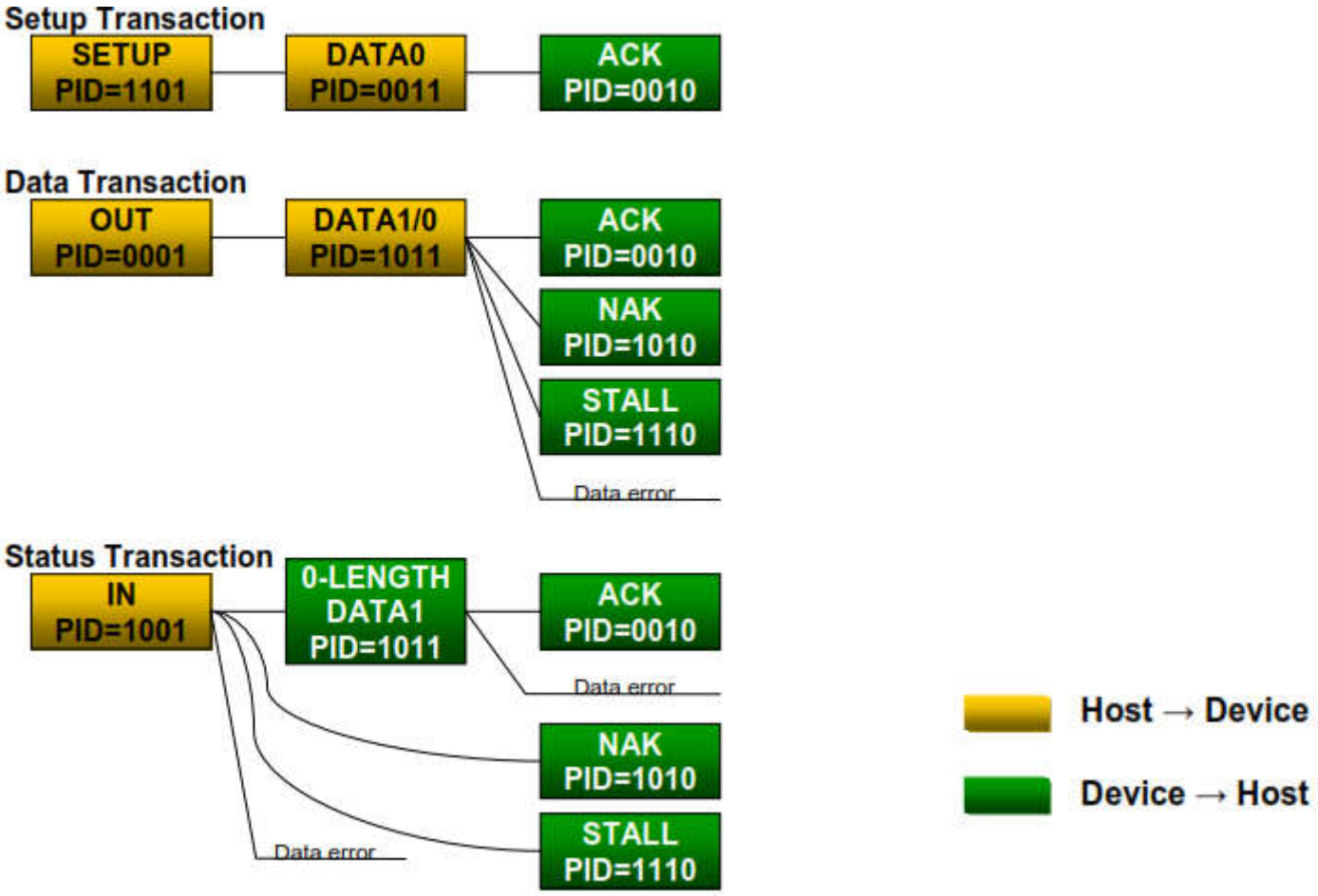
Transfers

Payload size:

	LS	FS	HS
Control	8	64	64
Bulk	--	64	512
Isochron	--	1023	1024
Interrupt	8	64	1024

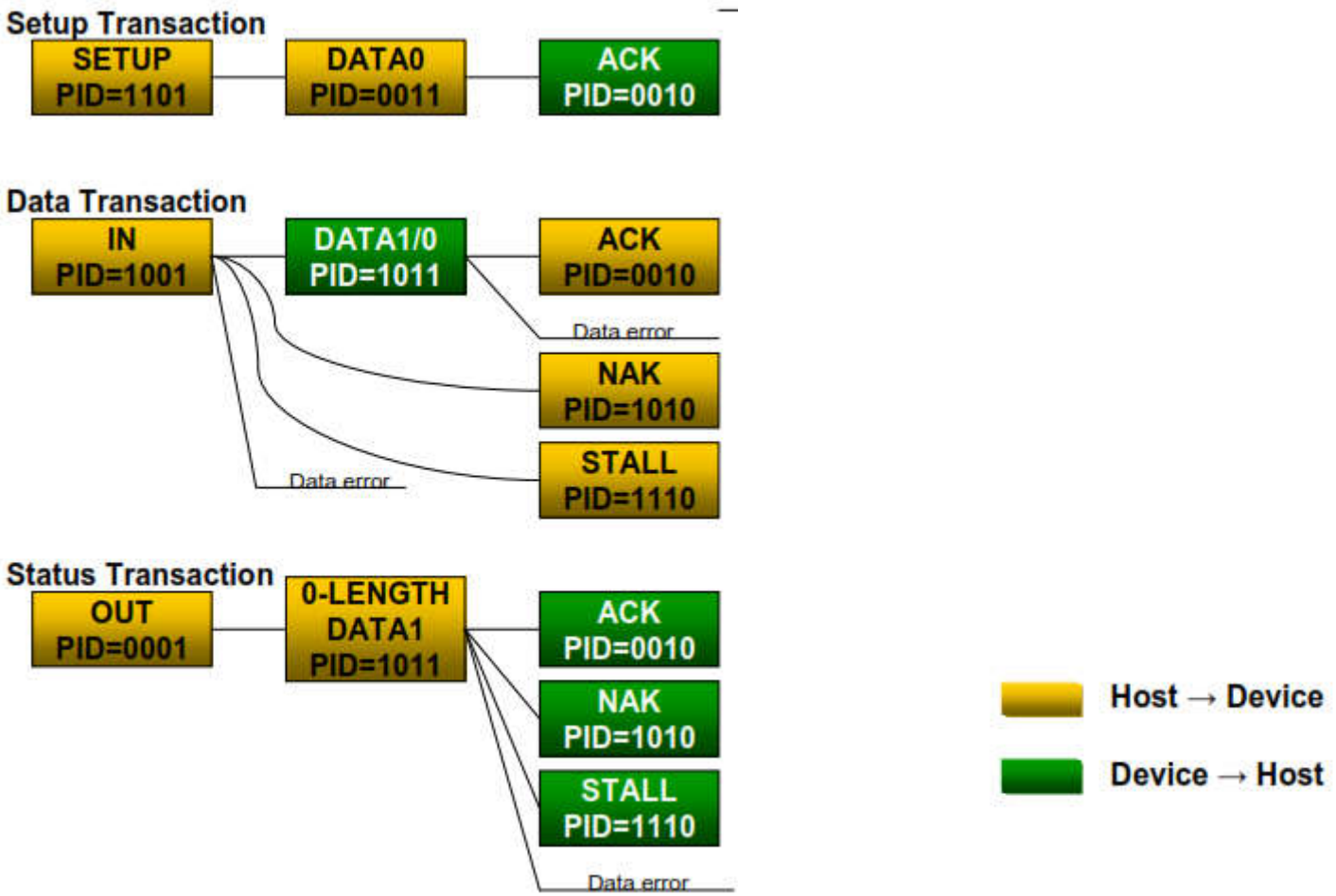
Example transfer

- Countrol OUT Transfer example: Setup, Data, Handshake

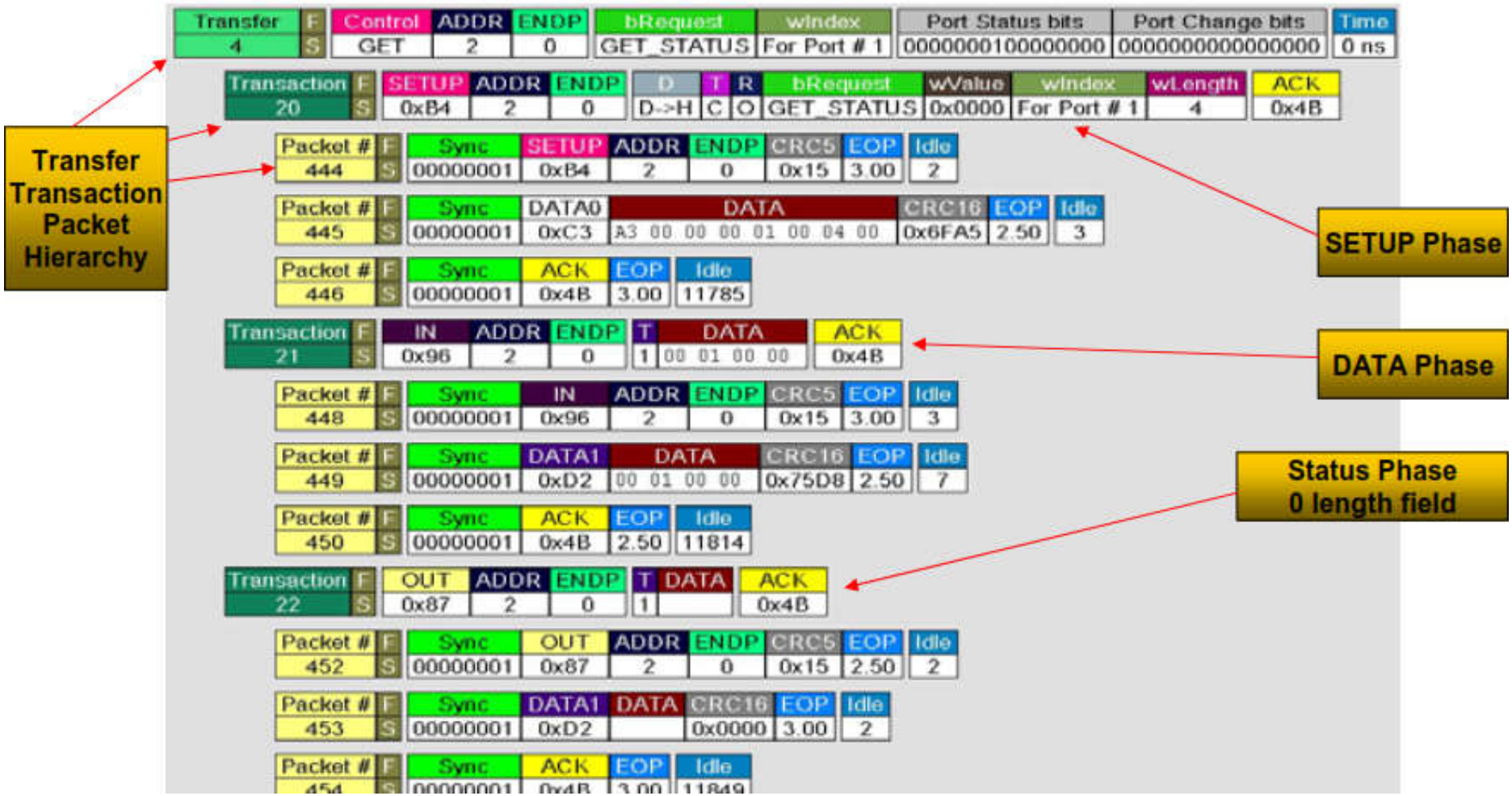


Example transfer

- Countrol IN Transfer: Setup, Data, Handshake

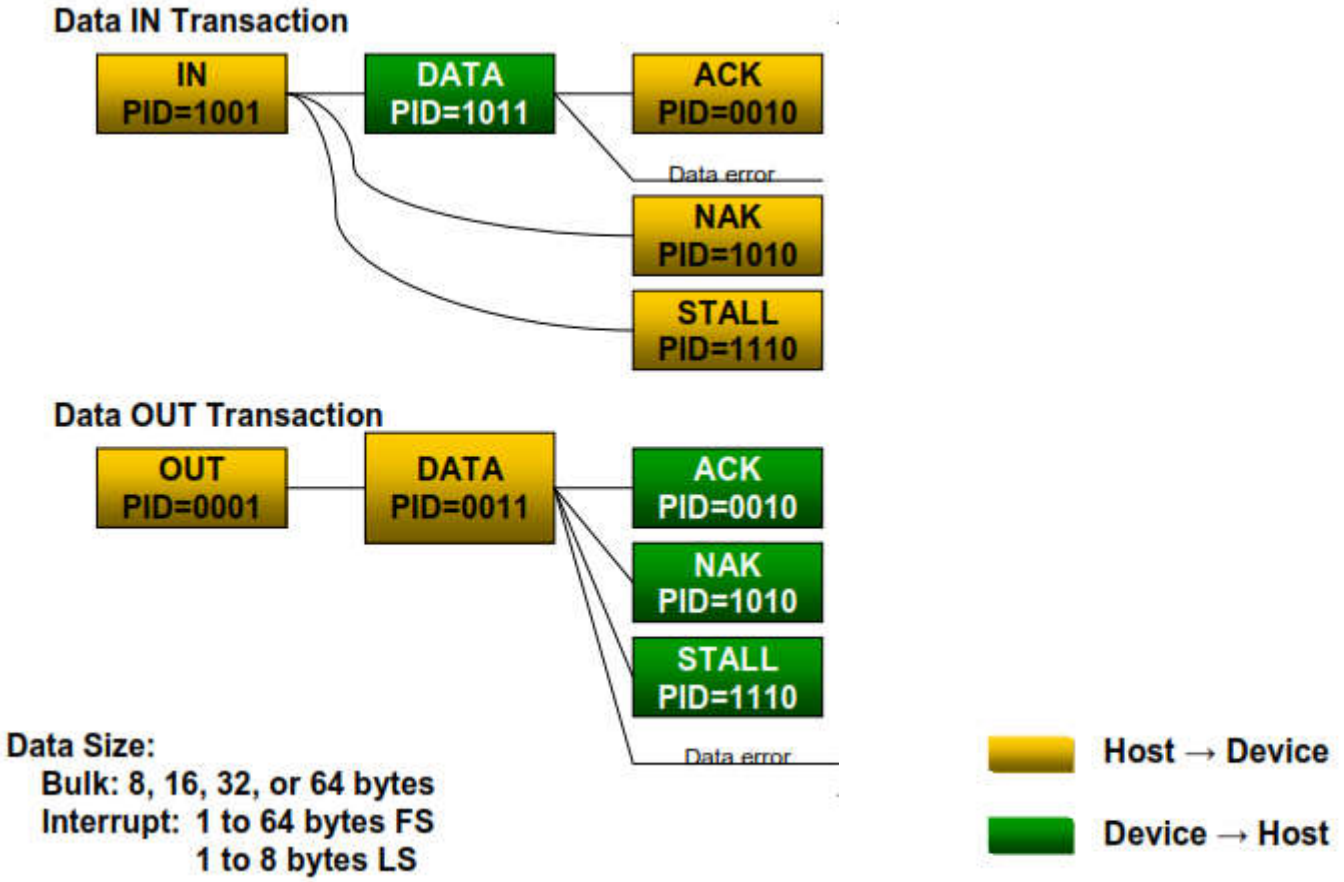


A real Control IN transfer example

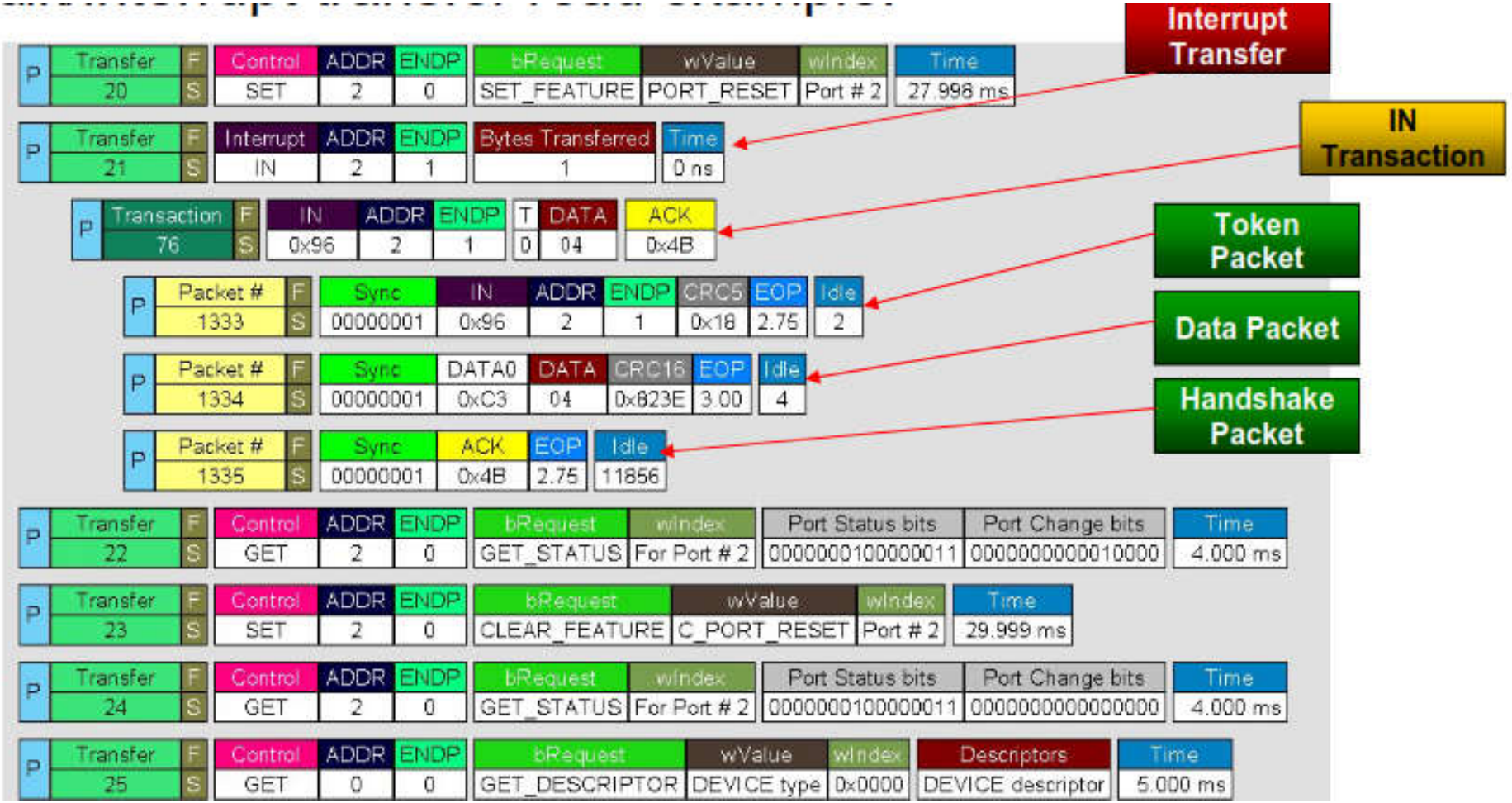


Example transfer

- Bulk and Interupt Transfer: IN/OUT, Data, Handshake

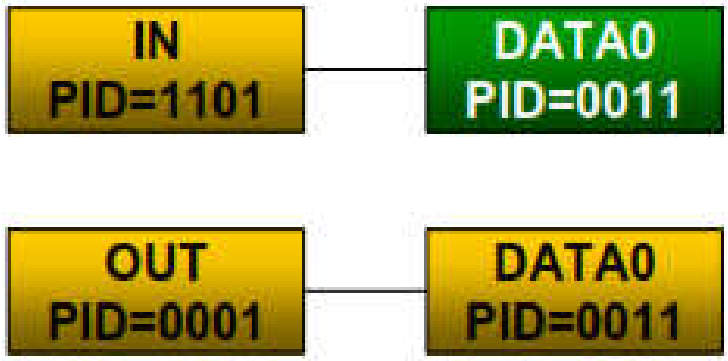


Example transfer: Interupt

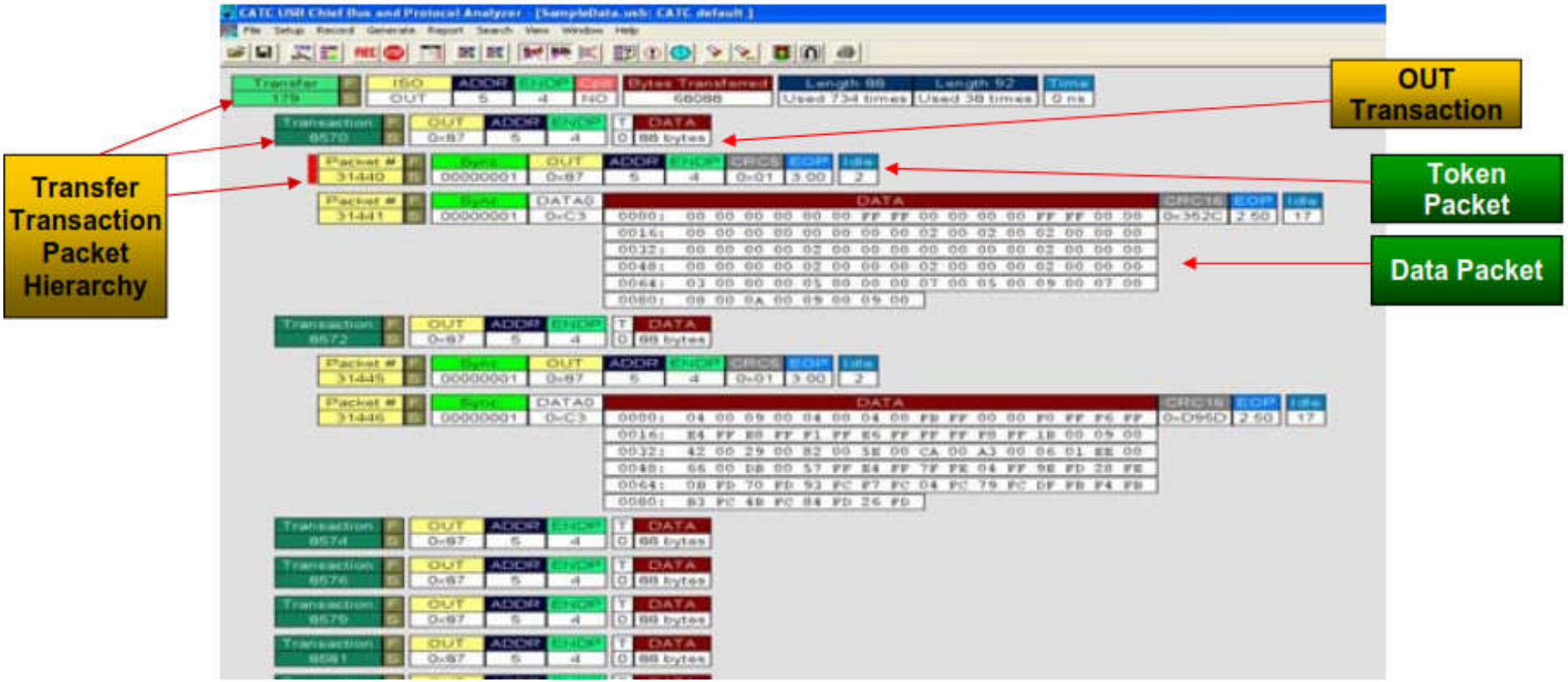


Example transfer

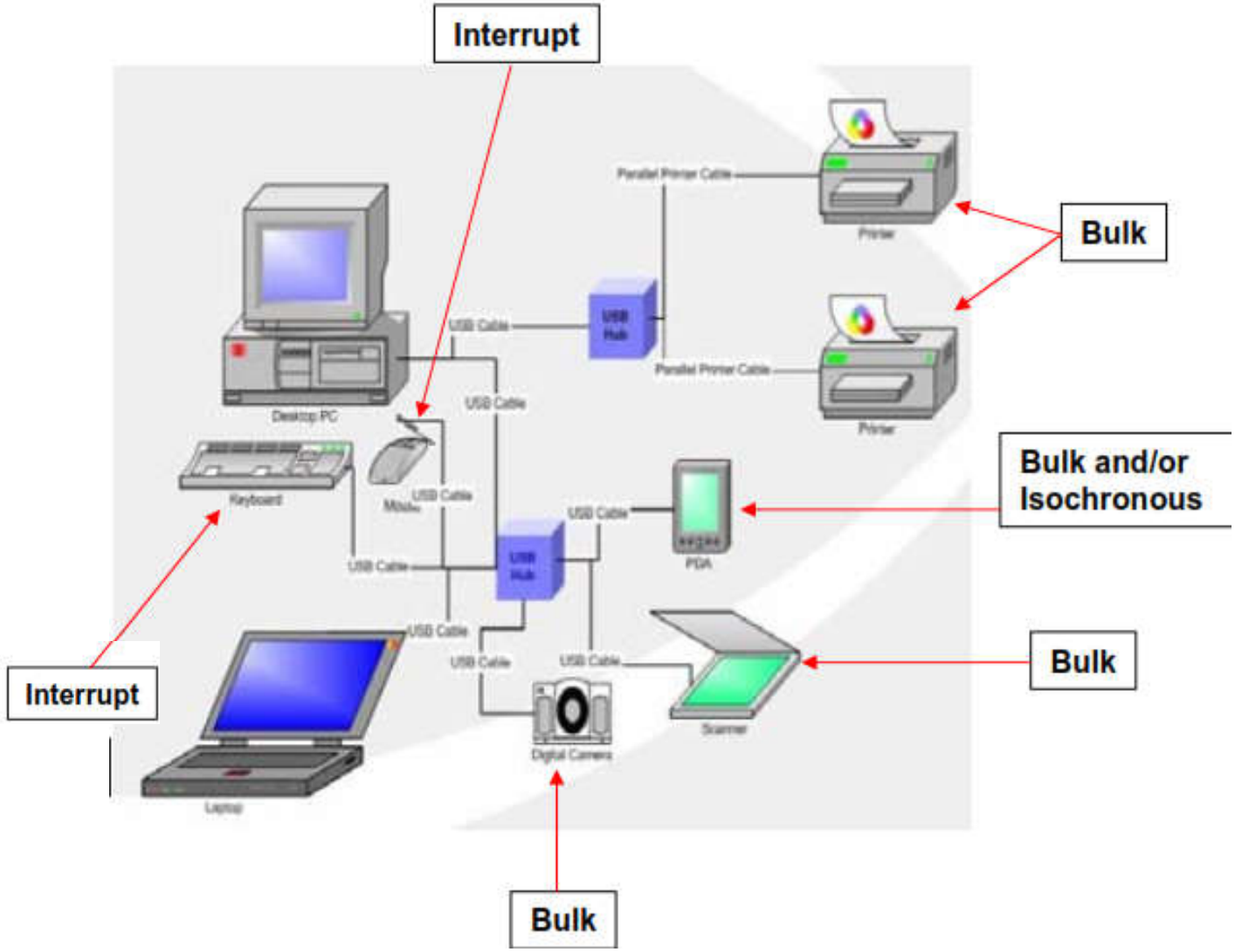
- Isochron transfer: IN/OUT, Data



Example transfer: isochron



Different peripheral types requires different transfer types

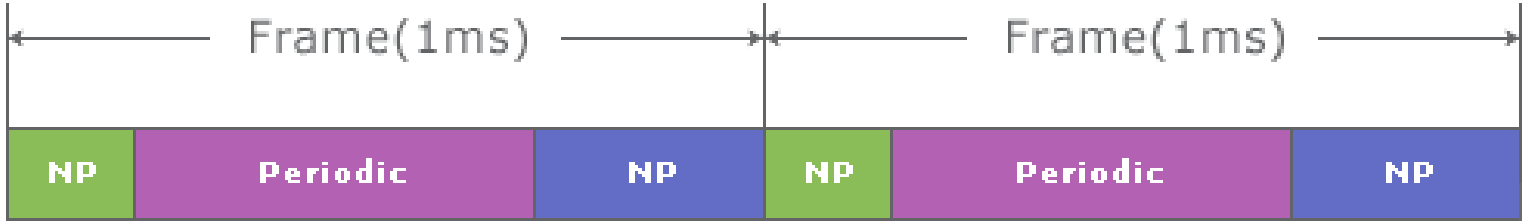


Transfers in Frames

- The isochrone and interrupt transfers has priority
  - Maximum of 90% of the frame can be such transfers
- If a new peripheral request interrupt or isochron transfers resulting a frame utilization more then 90% of such transfers, than the new device is not allowed to connect
- In the remaining 10% the control transfer has priority
- Remaining data bandwidth: Bulk

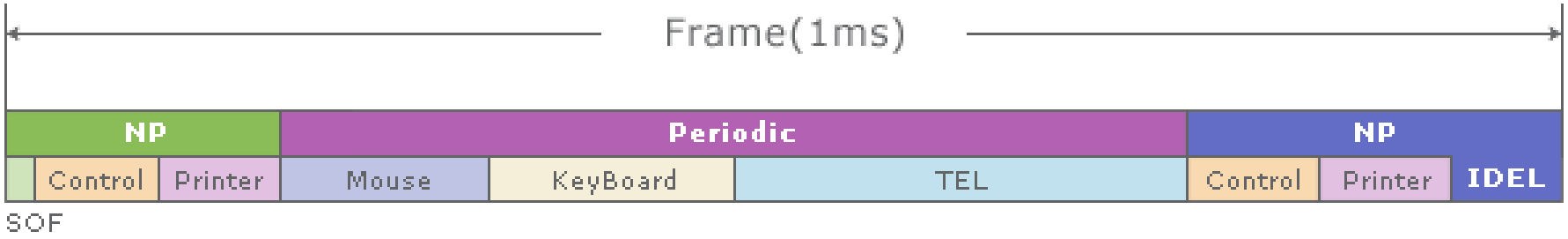
Frames

- Frame example

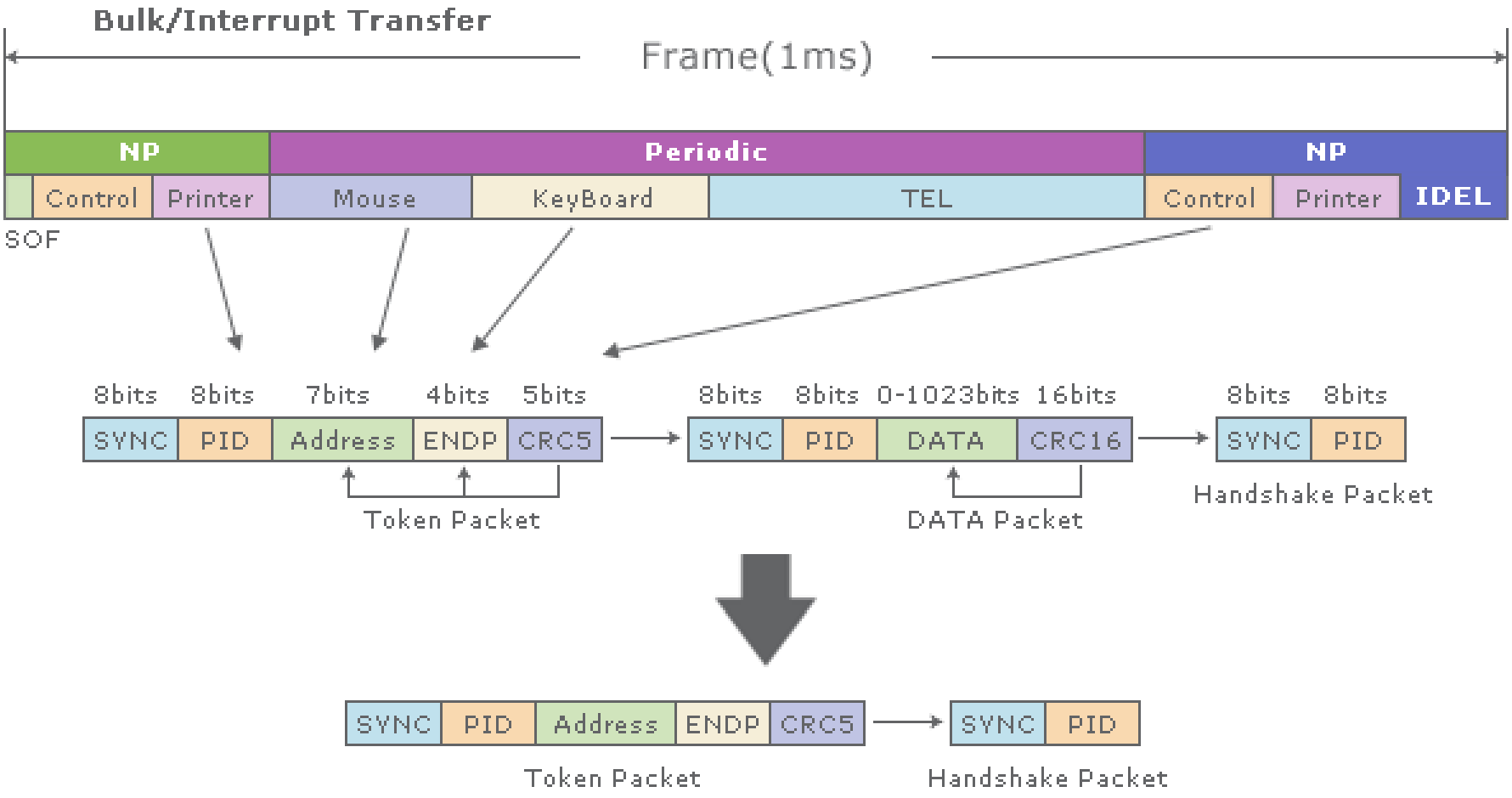


NP :Bulk/Control
Periodic :Isochronous/Interrupt

- Example for a typical frame



More detailed example

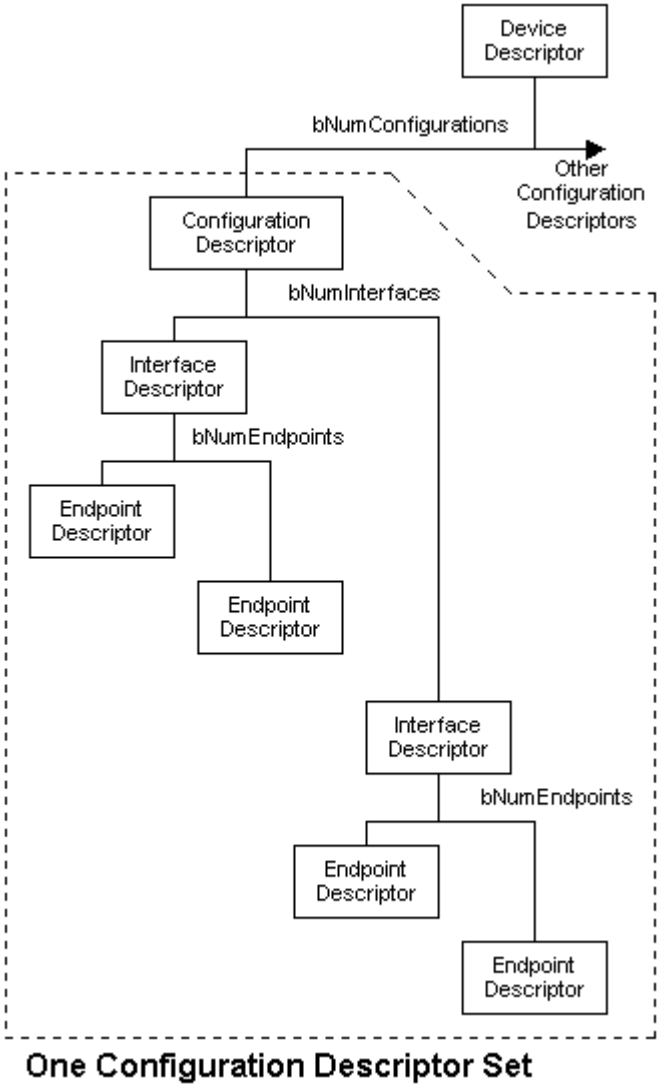


Connecting a new device

- Detecting the speed and protocol: Pull up resistor or K-Chirp
- Reset, the device listen on address 0
- Address 0. End Point 0. GetDeviceDescriptor: Detecting the maximum packet size
- Reset, SetAddress
- Device, Configuration és String descriptor download
- Driver loading
- SetConfiguration

Descriptors

- Device: unique
- Configuration: there is only one active (rare to have more than one)
- Interface: for composit devices: paralelly more than one functions: (VOIP phones)



Device descriptor

- Every device has one
- Main information
  - USB revision number
  - Product ID
  - Vendor ID
  - Number of configuration descriptions

- Loading the driver based on Product ID and Vendor ID

Device descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of the Descriptor in Bytes (18)
1	bDescriptorType	1	Constant	Device Descriptor (0x01)
2	bcdUSB	2	BCD	USB Specification Number which device complies too.
4	bDeviceClass	1	Class	Class Code (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 (by USB Org)
6	bDeviceProtocol	1	Protocol	Protocol Code (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 (by USB Org)
10	idProduct	2	ID	Product ID (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	bNumConfiguratio	1	Integer	Number of Possible Configurations

```
const uint8_t DeviceDescriptor[SIZ_DEVICE_DESC] =
 {
  SIZ_DEVICE_DESC,   /* bLength  */
  0x01,   /* bDescriptorType */
  0x00,   /* bcdUSB, version 2.00 */
  0x02,
  0x00,   /* bDeviceClass : each interface define the device class
  0x00,   /* bDeviceSubClass */
  0x00,   /* bDeviceProtocol */
  0x40,   /* bMaxPacketSize0 0x40 = 64 */
  0x83,   /* idVendor     (0483) */
  0x04,
  0x21,   /* idProduct */
  0x57,
  0x00,   /* bcdDevice 2.00*/
  0x02,
  1,          /* index of string Manufacturer  */
  /**/
  2,          /* index of string descriptor of product*/
  /* */
  3,          /* */
  /* */
  /* */
  0x01   /*bNumConfigurations */
 };
```

Device class codes

Base Class	Descriptor Usage	Description
00h	Device	Use class information in the Interface Descriptors
01h	Interface	Audio
02h	Both	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
10h	Interface	Audio/Video Devices
DCh	Both	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Both	Miscellaneous
FEh	Interface	Application Specific
FFh	Both	Vendor Specific

Configuration descriptor

- What type of interfaces has the device
- What is the power requirement
- Only one configuration can be active

- Host can switch between configurations

Interface descriptor

- The interface organize the endpoints based on function, One interface support one functionality.

- There are multifunctional devices with more interfaces
- For example: multifuncton Fax/scanner/printer
  - Interface 1: fax function
  - Interface 2: Scanner function
  - Interface 3: printer function

- Many interface can be active in the same type

Endpoint descriptor

- Describes the endpoint's
  - type
  - direction
  - Polling intervalum
  - Max packet size

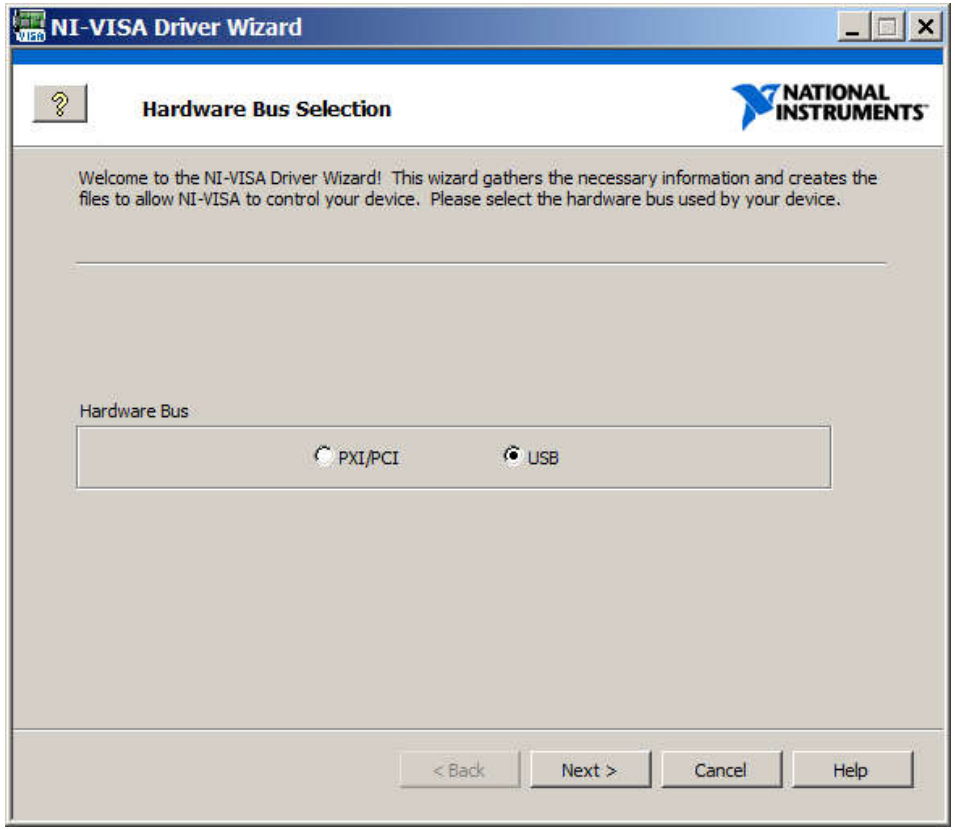
- The endpoint 0 always present and always a control endpoint

USB in embedded systems

- FTDI
  - FT2xxx : Serial-USB converter (VCP) (FS,HS)
  - Vinculum: providing some host functions (FS)
- Native USB
  - Firmware library-t from the microcontroller manufacturer
  - Third party sollutions
- PC side:
  - There is no need for own driver in case of popular classes, like HID devices
  - Vendor specific device:
    - libusb32 driver library
    - **LabVIEW: VISA Driver wizard**

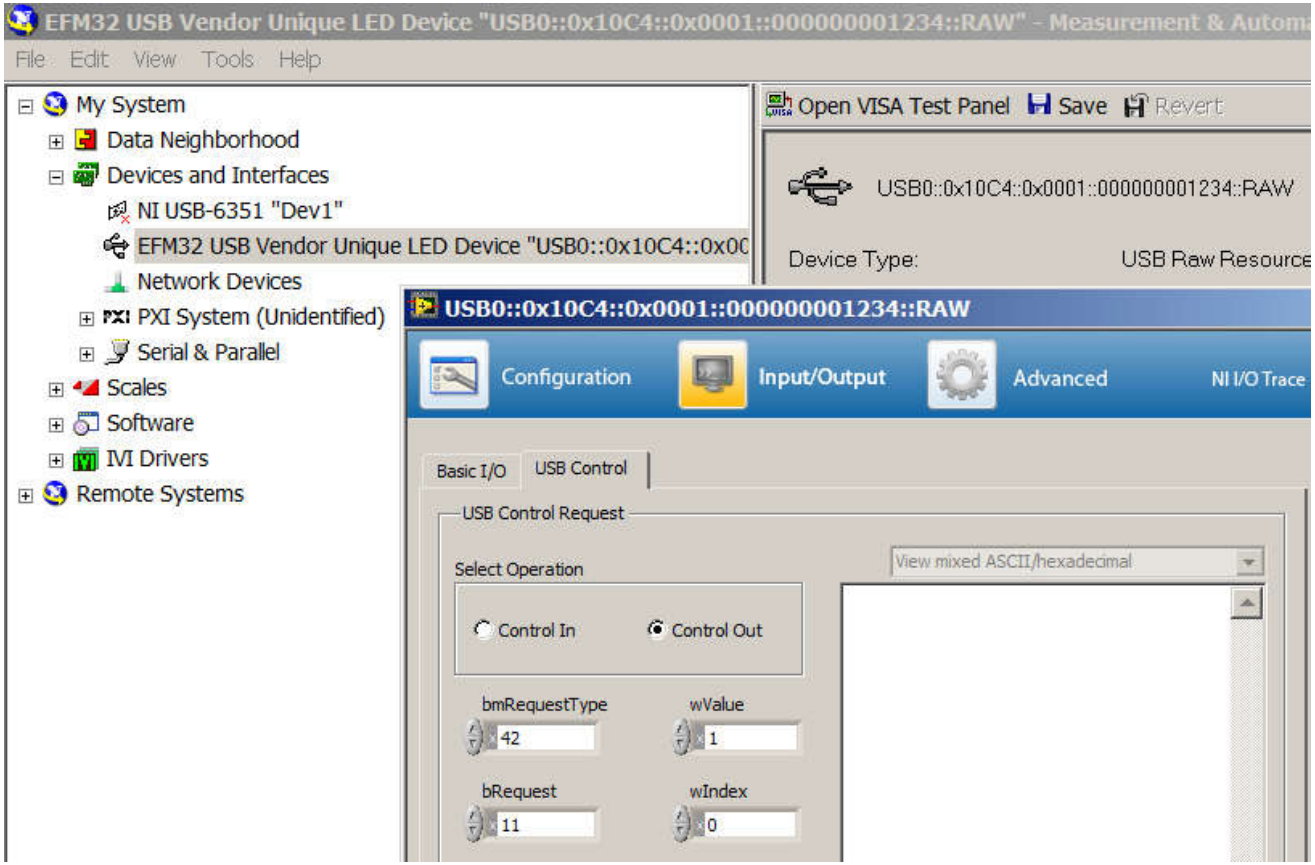
Handling a Vendor specific USB device in LabVIEW

- 1. VISA Driver wizard: Creating a driver
  - http://www.ni.com/tutorial/4478/en/
  - Vendor and Product ID based



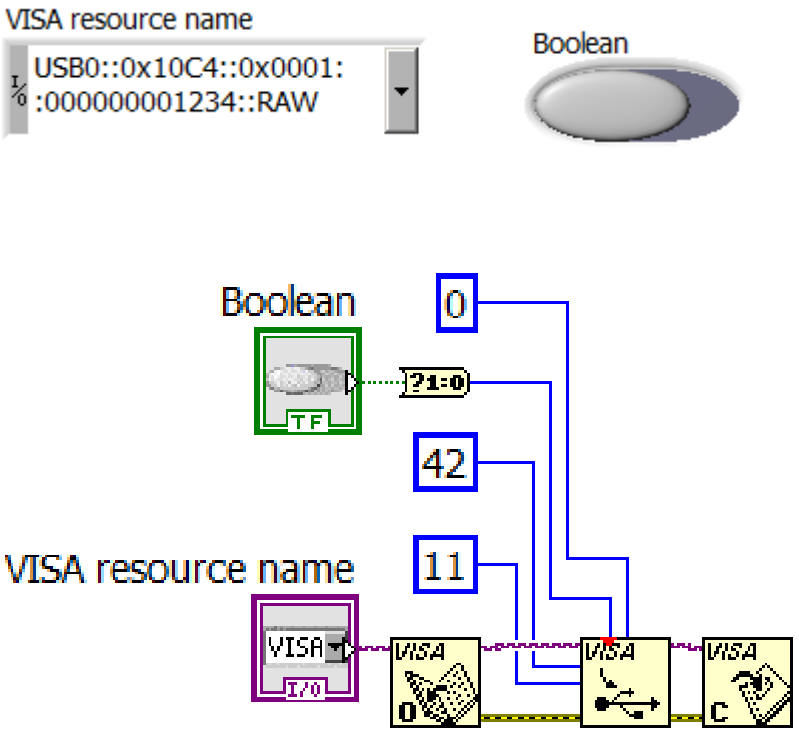
Handling a Vendor specific USB device in LabVIEW

- 2. Using MAX (Measurement Automation Explorer) to try out connection
  - Recognize the device and provides a general purpose user control



Handling a Vendor specific USB device in LabVIEW

- 3. Controller program in LabVIEW
  - VISA communication based examples Control, Iterrupt, Bulk
  - A simple Control example



Usefull links

- http://www.usb.org/developers/
- http://en.wikipedia.org/wiki/Universal_Serial_Bus
- http://www.usbmadesimple.co.uk/
- http://www.beyondlogic.org/usbnutshell/usb1.shtml
- http://www.ftdichip.com
- http://sourceforge.net/apps/trac/libusb-win32/wiki