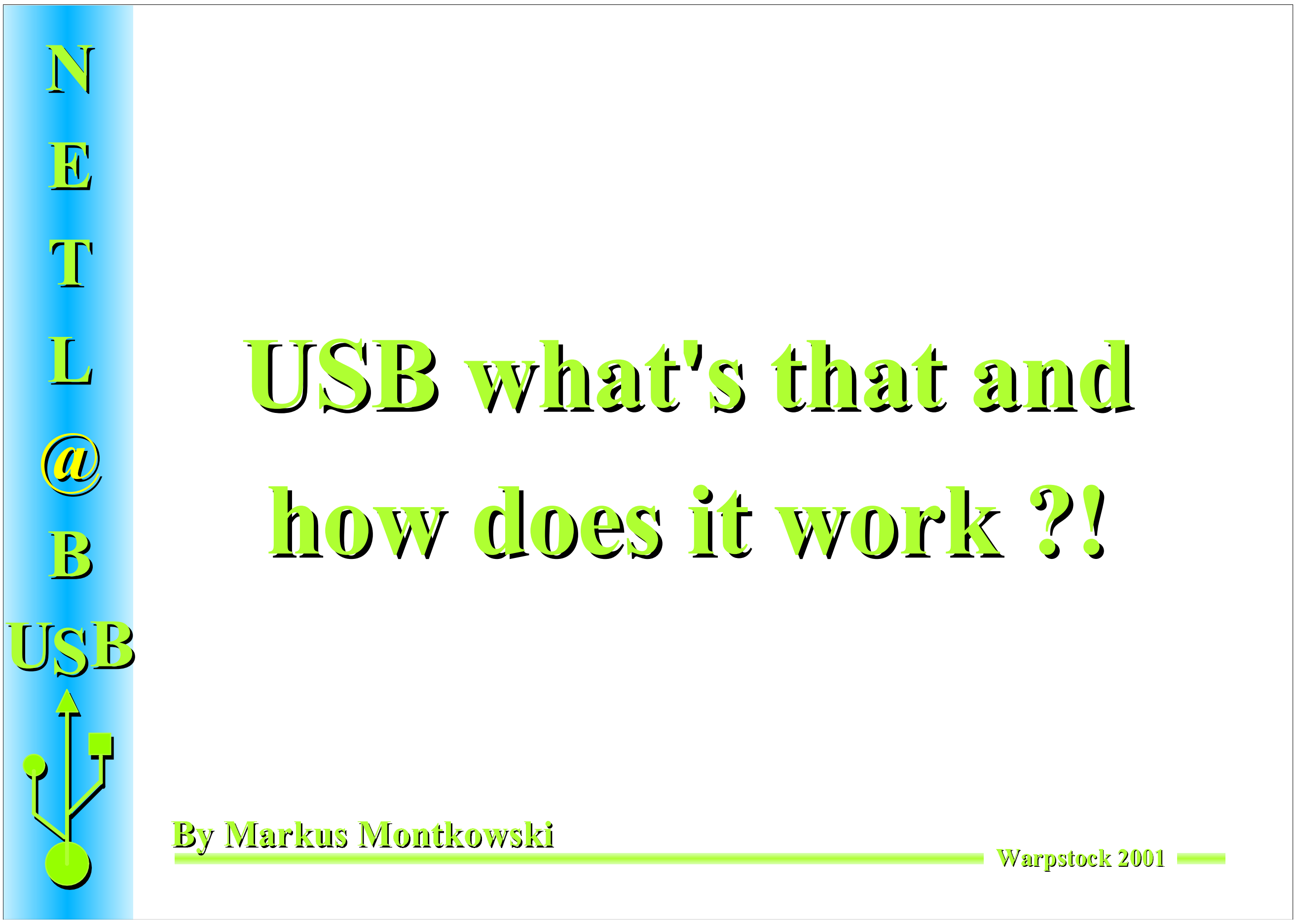


NETL@B USB

USB what's that and how does it work ?!

By Markus Montkowski

Warpstock 2001

NETL@B USB

Agenda

- Background
- Architectural Overview
- Hello or the USB Device Report

Warpstock 2001

Background

- Design Goals
- Feature List

Design Goals

- **Low-cost solution that supports transfer rates up to 12Mb/s**
- **Full support for real-time data for voice, audio, and compressed video**
- **Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging**
- **Integration in commodity device technology**
- **Provision of a standard interface capable of quick diffusion into product**
- **Enablement of new classes of devices that augment the PC s capability.**

Feature List

Easy to use for end user
Wide range of workloads and applications
Isochronous bandwidth
Flexibility
Robustness
Synergy with PC industry
Low-cost implementation
Upgrade path

Easy to use for end user

- Single model for cabling and connectors
- Electrical details isolated from end user (e.g., bus terminations)
- Self-identifying peripherals, automatic mapping of function to driver, and configuration
- Dynamically attachable and reconfigurable peripherals

Wide range of workloads and applications

- Suitable for device bandwidths ranging from a few kb/s to several Mb/s
- Supports isochronous as well as asynchronous transfer types over the same set of wires
- Supports concurrent operation of many devices (multiple connections)
- Supports up to 127 physical devices
- Supports transfer of multiple data and message streams between the host and devices
- Allows compound devices (i.e., peripherals composed of many functions)
- Lower protocol overhead, resulting in high bus utilization

Isochronous bandwidth

- Guaranteed bandwidth and low latencies appropriate for telephony, audio, etc.
- Isochronous workload may use entire bus bandwidth

NETL@B USB

Flexibility

- Supports a wide range of packet sizes, which allows a range of device buffering options
- Allows a wide range of device data rates by accommodating packet buffer size and latencies
- Flow control for buffer handling is built into the protocol

NETL@B USB

Robustness

- Error handling/fault recovery mechanism is built into the protocol
- Dynamic insertion and removal of devices is identified in user-perceived real-time
- Supports identification of faulty devices

Synergy with PC industry

- Protocol is simple to implement and integrate
- Consistent with the PC plug-and-play architecture
- Leverages existing operating system interfaces

Low-cost implementation

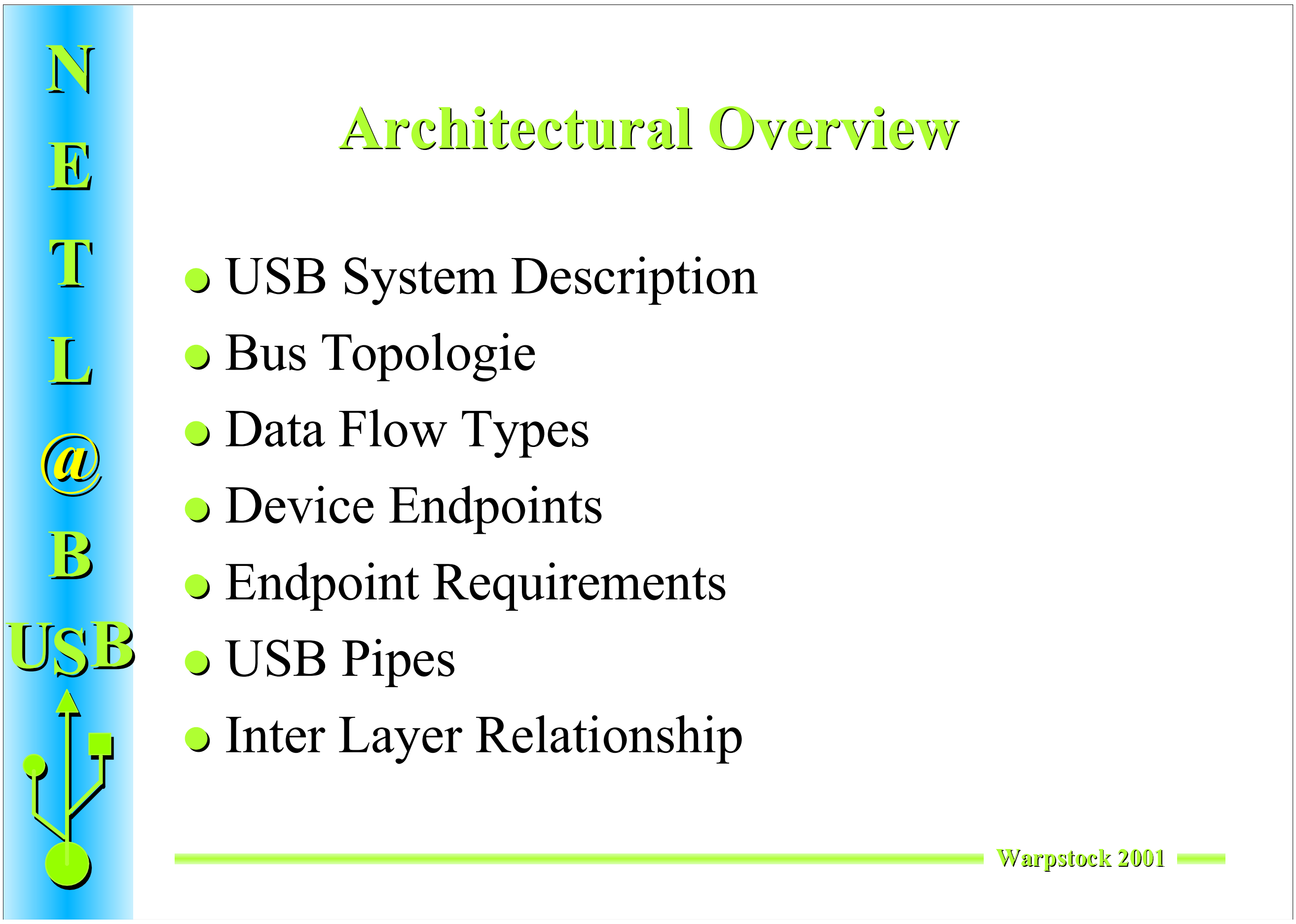
- Low-cost subchannel at 1.5Mb/s
- Optimized for integration in peripheral and host hardware
- Suitable for development of low-cost peripherals
- Low-cost cables and connectors
- Uses commodity technologies

NETL@B USB

Upgrade path

- Architecture upgradeable to support multiple USB Host Controllers in a system

Warpstock 2001



NETL@B USB

Architectural Overview

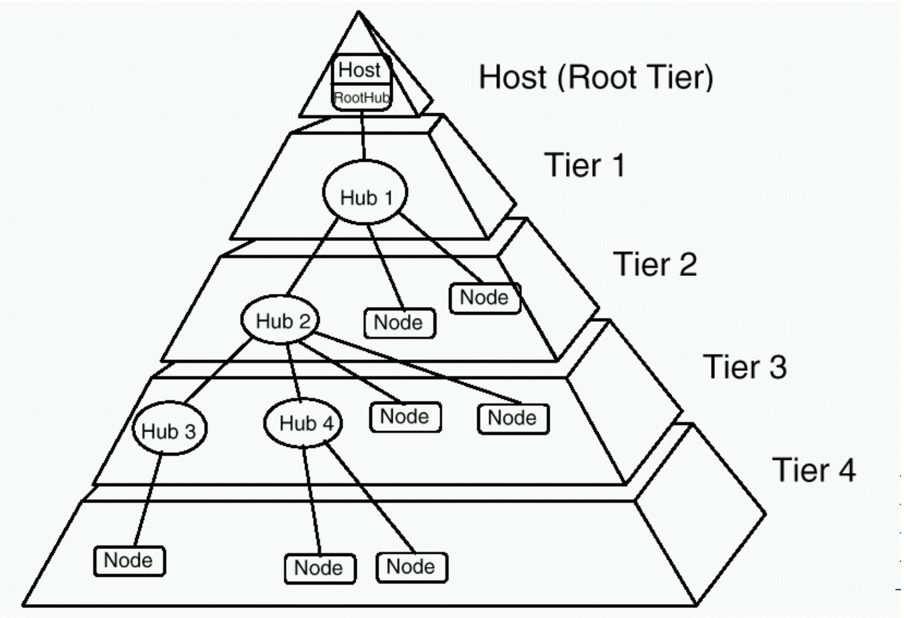
- USB System Description
- Bus Topologie
- Data Flow Types
- Device Endpoints
- Endpoint Requirements
- USB Pipes
- Inter Layer Relationship

Warpstock 2001

USB System Description

- USB interconnect
  - Bus Topology: Connection model between USB devices and the host.
  - Inter-layer Relationships: In terms of a capability stack, the USB tasks that are performed at each layer in the system.
  - Data Flow Models: The manner in which data moves in the system over the USB between producers and consumers.
  - USB Schedule: The USB provides a shared interconnect. Access to the interconnect is scheduled in order to support isochronous data transfers and to eliminate arbitration overhead.
- USB devices
  - Hubs, which provide additional attachment points to the USB
  - Functions, which provide capabilities to the system, such as an ISDN connection, a digital joystick, or speakers.
- USB host

Bus Topology



USB Pipes

- Streamed Pipe
  - Uni-directional
  - No USB defined data structure
  - Bulk, isochronius and interrupt transfers
- Message Pipes
  - Bi-directional
  - USB data/flow structure (request/data/status)
  - Only Control transfers

Device Endpoints

- Their bus access frequency/latency requirements
- Their bandwidth requirements
- Their endpoint number
- The error handling behavior requirements
- Maximum packet size that the endpoint is capable of sending or receiving

Endpoint Requirements

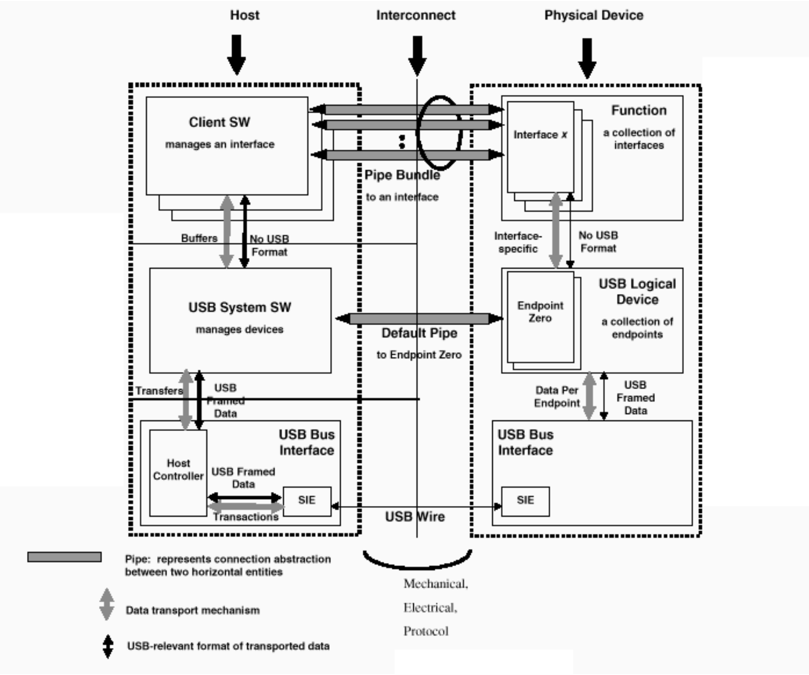
- Endpoint Zero Requirements
  - All Devices
  - Input and Output
  - Allways accessible
- Non-endpoint Zero Requirements
  - Device dependend
    - Lowspeed devices (1,5MBit) max. 2 addional other devices 30 (15 Input & 15 Output)
  - Accessible only after Device configuration

Data Flow Types

- **Control Transfers**
  - **Used to configure a device at attach time and can be used for other device-specific purposes, including control of other pipes on the device.**
- **Bulk Data Transfers**
  - **Generated or consumed in relatively large and bursty quantities and have wide dynamic latitude in transmission constraints.**
- **Interrupt Data Transfers**
  - **Used for characters or coordinates with human-perceptible echo or feedback response characteristics.**
- **Isochronous Data Transfers**
  - **Occupy a prenegotiated amount of USB bandwidth with a prenegotiated delivery latency. (Also called streaming real time transfers).**

NETL@B USB

Inter Layer Relationship



NETL@B USB

Hello or the USB Device Report

- Requested from the Host when a Device is plugged in
- Consists outoff a number a Descriptors
- Contains information about
  - Vendor
  - Product
  - Capabilities
  - Power consumption

USB Device Descriptor

```
Typedef struct _device_descriptor_
{
   UCHAR    bLength;            // (00) Size of descriptor in bytes
   UCHAR    bDescriptorType;    // (01) 0x01 - DEVICE Descriptor type
   USHORT   bcdUSB;             // (02) USB Specification Release Number
   UCHAR    bDeviceClass;       // (04) Class Code
   UCHAR    bDeviceSubClass;    // (05) SubClass Code
   UCHAR    bDeviceProtocol;    // (06) Protocol Code
   UCHAR    bMaxPacketSize0;    // (07) Maximum packet size for endpoint 0
   USHORT   idVendor;           // (08) Vendor ID
   USHORT   idProduct;          // (10) Product ID
   USHORT   bcdDevice;          // (12) Device release number
   UCHAR    iManufacturer;      // (14) Index of string descriptor
                                //      describing manufacturer
   UCHAR    iProduct;           // (15) Index of string descriptor
                                //      describing product
   UCHAR    iSerialNumber;      // (16) Index of string descriptor
                                //      describing device's serial number
   UCHAR    bNumConfigurations; // (17) Number of possible configurations
                                // (18)
}  DeviceDescriptor;
```

USB Configuration Descriptor

```
typedef struct _device_configuration_
{
  UCHAR  bLength;             // (00) Size of descriptor in bytes
  UCHAR  bDescriptorType;     // (01) 0x02 - CONFIGURATION Descriptor type
  USHORT wTotalLength;        // (02) total data length returned in request
  UCHAR  bNumInterfaces;      // (04) number of interfaces in this config
  UCHAR  bConfigurationValue; // (05) value to be used in Set configuration
  UCHAR  iConfiguration;      // (06) String index describing this config
  UCHAR  bmAttributes;        // (07) Configuration characteristics
  UCHAR  MaxPower;            // (08) power consumption in 2 mA units
                              // (09)
}  DeviceConfiguration;
```

USB Interface Descriptor

```
typedef struct _device_interface_
{
  UCHAR bLength;              // (00) Size of descriptor in bytes
  UCHAR bDescriptorType;      // (01) 0x04 - INTERFACE Descriptor type
  UCHAR bInterfaceNumber;     // (02) 0 based index in interface array
  UCHAR bAlternateSetting;    // (03) value to select alternate interface
  UCHAR bNumEndpoints;        // (04) no of endpoints used by current
                              //      interface (excluding endpoint 0)
  UCHAR bInterfaceClass;      // (05) Class code
  UCHAR bInterfaceSubClass;   // (06) Subclass code
  UCHAR bInterfaceProtocol;   // (07) Protocol code
  UCHAR iInterface;           // (08) descriptor string index
                              // (09)
}  DeviceInterface;
```

USB Endpoint Descriptor

```
typedef struct _device_endpoint_
{
  UCHAR   bLength;          // (00) Size of descriptor in bytes
  UCHAR   bDescriptorType;  // (01) 0x05 - ENDPOINT Descriptor type
  UCHAR   bEndpointAddress; // (02) address of endpoint
                            //  #define  DEV_ENDPT_ADDRMASK   0x0f
                            //  #define  DEV_ENDPT_DIRMASK    0x80
                            //  #define  DEV_ENDPT_DIRIN      0x80
                            //  #define  DEV_ENDPT_DIROUT     0x00
  UCHAR   bmAttributes;     // (03) endpoint's attributes
                            //  #define  DEV_ENDPT_ATTRMASK   0x03
                            //  #define  DEV_ENDPT_CTRL       0x00
                            //  #define  DEV_ENDPT_ISOHR      0x01
                            //  #define  DEV_ENDPT_BULK       0x02
                            //  #define  DEV_ENDPT_INTRPT     0x03
  USHORT  wMaxPacketSize;   // (04) maximum packet size for this endpoint
  UCHAR   bInterval;        // (06) interval for polling endpoint for data
                            // (07)
}  DeviceEndpoint;
```

NETL@B USB

Useful information links

- General info docs etc ***www.usb.org***
- USB device information ***www.linux-usb.org***
- Sources for many linux USB drivers ***www.sourceforge.net***
- The OS/2 DDK with sources of USB drivers ***service.boulder.ibm.com/ddk/***
- OS/2 USB Project at ***www.netlabs.org***
  - **CVS CVSROOT=:pserver:guest@www.netlabs.org:e:/netlabs.cvs/usb**
  - **Contact usbguy@netlabs.org**