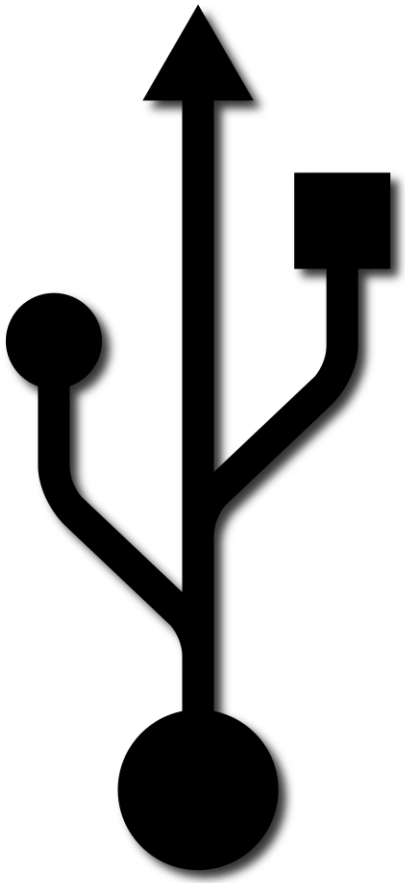


An Introduction to Universal Serial Bus

Neil Scott
June 27, 2008

Overview

- Introduction
- History
- Hierarchy
- Enumeration
- Demonstration Circuit
- Conclusions
- Questions

Introduction

- USB – Universal Serial Bus
- Replaces legacy interfaces – RS232 and Parallel
- Plug-and-play
- Fast, user-friendly (usually)
- Commonly found in printers, scanners, keyboard/mouse, external hard disk, flash drives, digital cameras, webcams ...

History

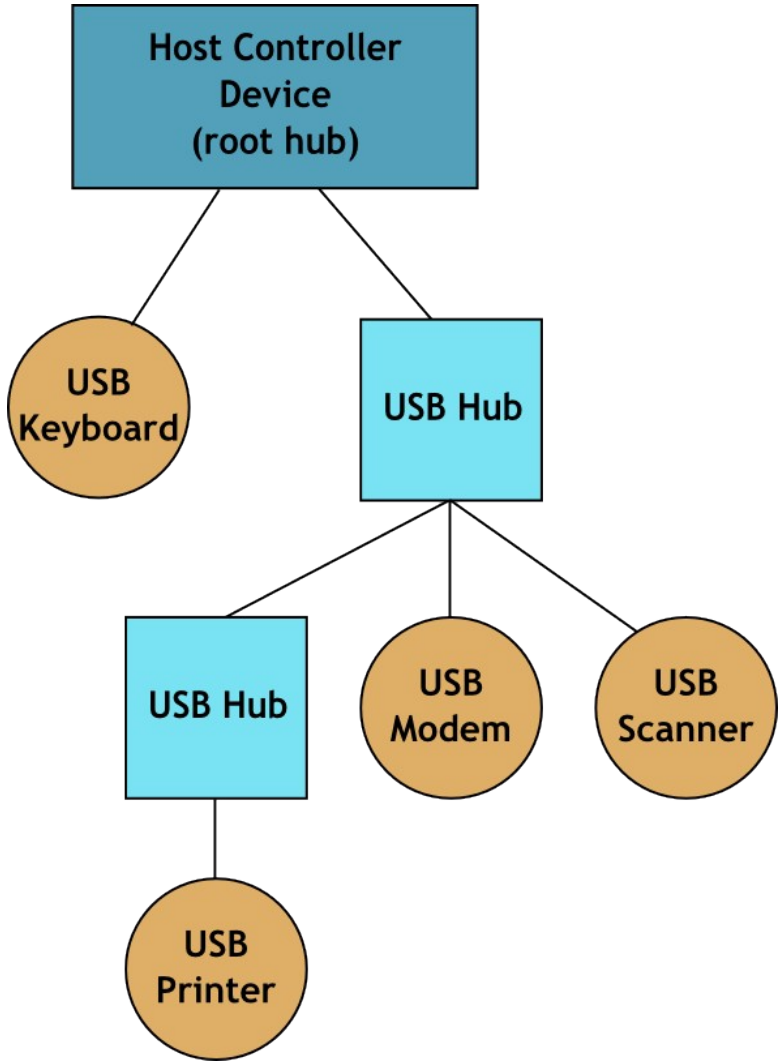
- USB has gone through several revisions
- Initial release USB1.0 in November 1995
- USB1.0 – 1.5Mbps (Low-Speed) and 12Mbps (Full-Speed)
- Re-release USB1.1 in Sept. 1998 – rectify adoption problem
- USB2.0 Released in April 2000
- USB2.0 – 480Mbps(High-Speed) and backward compatible

Why so popular?

- USB is **low cost** in comparison to high-speed serial bus technologies like IEEE1394 (FireWire)
  - USB is tiered-star topology, FireWire is peer-to-peer based requiring “smarter” peripherals
  - USB devices cannot initiate communication, FireWire devices can communicate with any node, at any time.
  - USB network has single host, where in FireWire any node can control the network

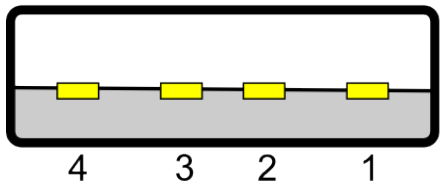
Topology of USB

- Tiered-Star Topology
- Hubs used for branching
- 5 tiers of hubs permitted
- 127 devices maximum
- bandwidth is limited and varies with device count

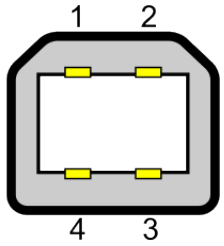


USB Topology

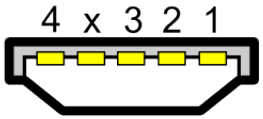
USB Connector & Cabling



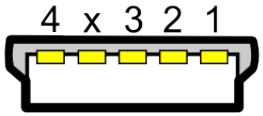
Type A



Type B

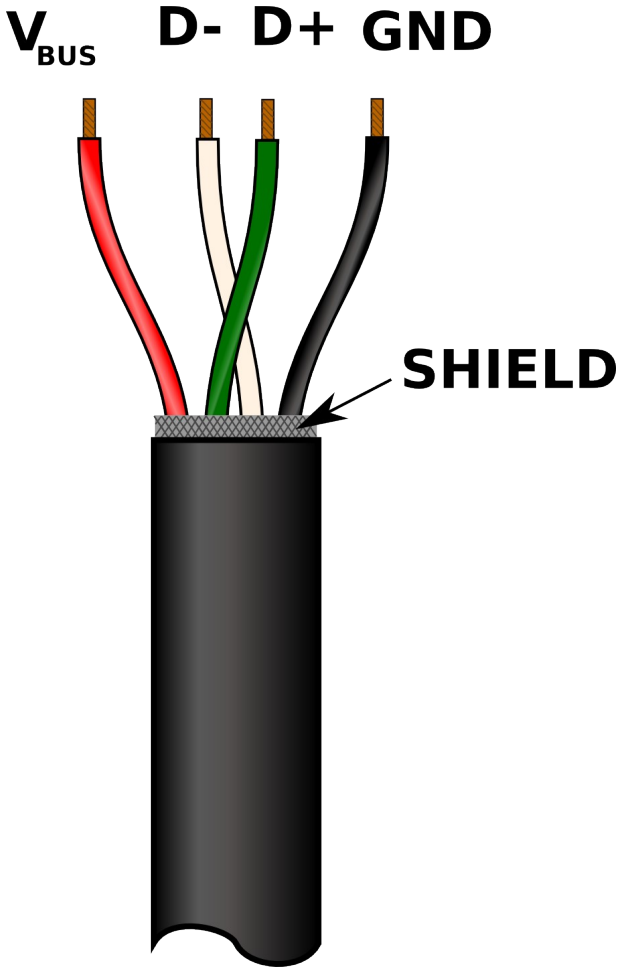


Mini-A



Mini-B

USB Connectors



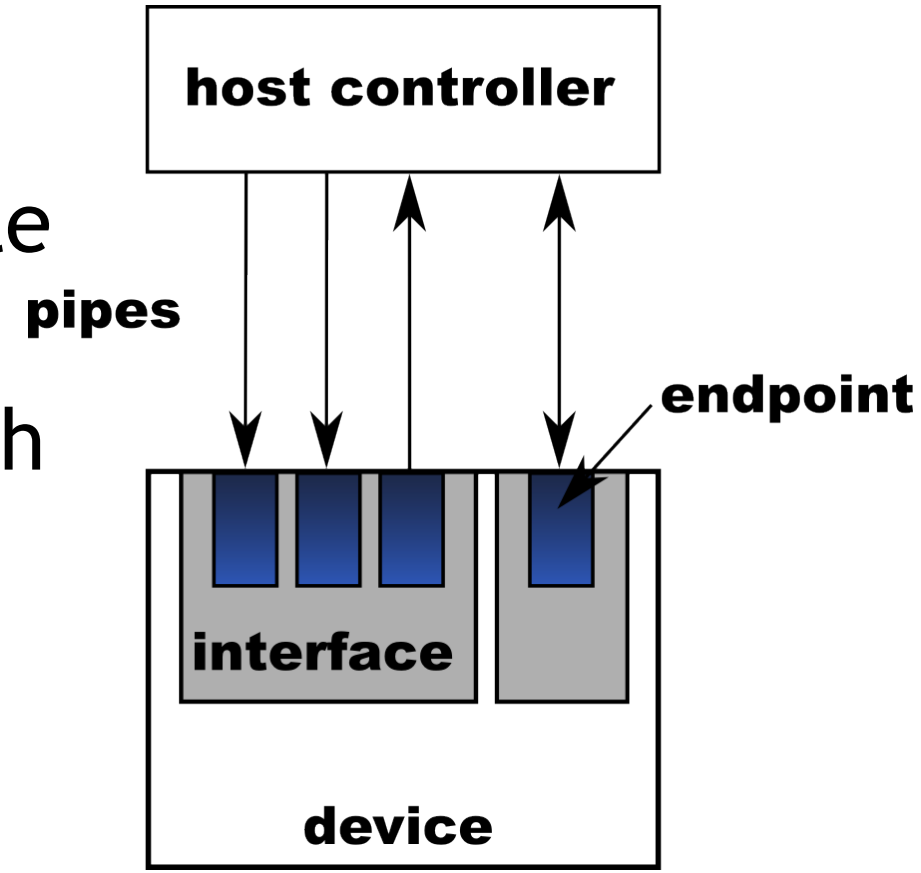
USB Cable

USB Device Classes

- Devices can be fully custom requiring a custom driver, or may belong to a device class
- Device classes define behaviour of device so a single device driver can be used for various devices
- Common Device Classes include: Mass Storage Device, Human Interface Device (HID), USB hub and Printer

USB Device

- Single Physical device may consist of several logical sub-devices
- Interfaces used to separate device functionality
- Drivers exchange data with endpoints over pipes (logical channels)
- Endpoints are uni-directional



USB Device

Endpoint Zero (Control Endpoint)

- Endpoint Zero required by all devices for standard requests (device descriptors)
- During enumeration, device identifies itself with the host OS by set of standard requests over EP0
- Endpoint Zero is only bi-directional endpoint
- Often used for vendor control requests also

Data Flow Types (Endpoints)

- **Control** – for enumeration, control and status where data is non-periodic (device control)
- **Bulk** – large payloads, guaranteed data with built-in error checking and hardware retries (flash drive, external hard drive)
- **Interrupt** – when data is small and infrequent with bounded service period (keyboards/mice)
- **Isochronous** – periodic data where data rate is supersedes integrity (webcam)

Designing a USB Device

- Application speed requirements
  - Full Speed 480Mbps
  - High Speed 12Mbps
  - Low Speed 1.5Mbps
- Find a suitable USB peripheral controller
- Determine the interface(s) and endpoints of the device
- Write firmware, driver and software!

Cypress EZ-USB FX2 Controller

- Integrated high-speed USB microcontroller
  - Single Chip USB2.0 transceiver, serial interface engine and enhanced 8051 core
- Firmware can be loaded from external EEPROM or over USB
- Two USART controllers, an I2C controller
- Integrated 4kB FIFO for endpoints (slave mode or general purpose interface)

Cypress EZ-USB GPIF

- GPIF provides internal master for internal endpoint FIFOs.
- User defined waveform descriptors control GPIF state machine
- Can be used for interface between ASICs, DSPs, or even more complex interfaces such as ATAPI for PATA hard drive interface.

USB Driver Development

- Device driver required to interact with device
- Drivers are OS specific and HW dependent
- **Kernel Drivers**
  - Requires proficient programming or may crash system by overwriting memroy
- **User-Space / User-Mode Drivers**
  - More stable than kernel drivers, cannot crash kernel.
  - Performance overhead when transitioning between user and kernel space

libUSB

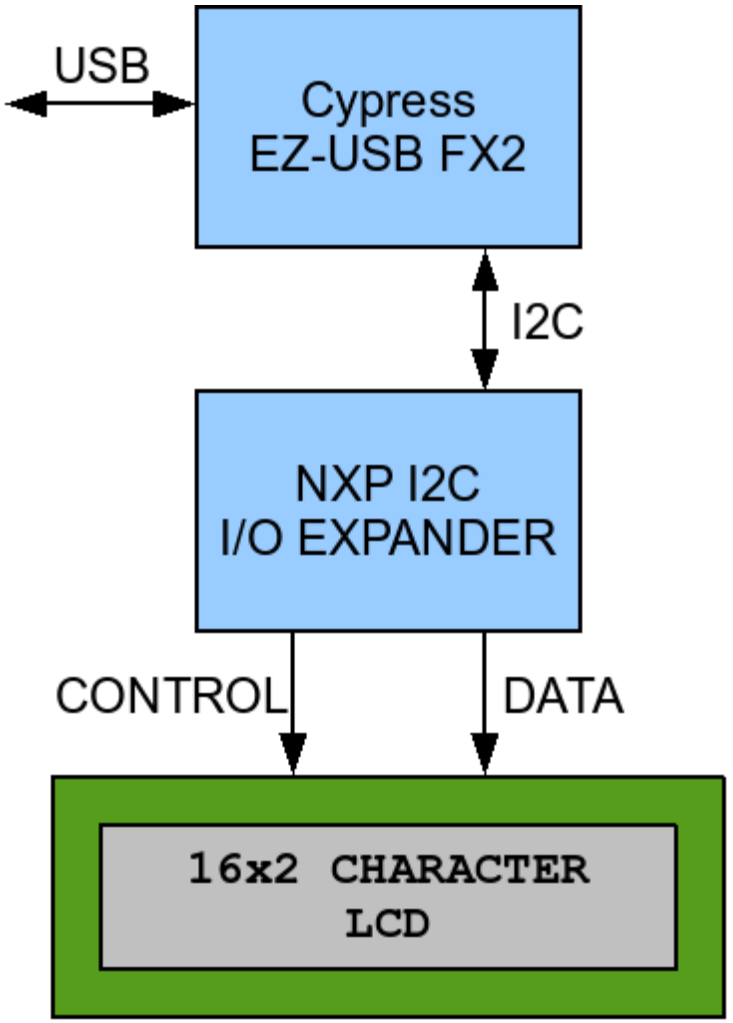
- Collection of User-Space routines for manipulating USB devices without need for kernel driver
- Available for Linux, Windows, Mac OSX and FreeBSD (and others).
- Supporting functions for
  - finding devices on in a USB system
  - selecting and claiming interfaces
  - control, bulk and interrupt transfers
  - no isochronous endpoint support

libUSB

- Allows a designer a relatively easy API to use when interacting with devices
- Not as fast as a kernel-mode driver but much easier to develop
- Wrappers have been created for Python, Perl and Java

Demo Circuit - USBLCD

- Set of Vendor Specific Requests to control character LCD
- Data is sent to default control endpoint. No additional endpoints used.
- I/O Expander board used to control inputs of LCD operating at 5V



USBLCD Block Diagram

Demo Circuit - User-Mode Driver

- Written in Python using pyUSB wrapper for libusb

```
class DeviceDescriptor(object) :
    def __init__(self, vendor_id, product_id, interface_id) :
        self.vendor_id = vendor_id
        self.product_id = product_id
        self.interface_id = interface_id

    def getDevice(self) :
        buses = usb.busses()
        for bus in buses :
            for device in bus.devices :
                if device.idVendor == self.vendor_id :
                    if device.idProduct == self.product_id :
                        return device
        return None
```

Python Code Snippet for finding a USB Device

Demo Circuit - User-Mode Driver

```
class devUSBLCD(object):
    USBLCD_VID = 0xabcd
    USBLCD_PID = 0x0501
    USBLCD_IFID = 0
...
    def __init__(self):
        self.device_descriptor = DeviceDescriptor (self.USBLCD_VID,
                                                   self.USBLCD_PID,
                                                   self.USBLCD_IFID)
        self.device = self.device_descriptor.getDevice()
        self.handle = None

    def open(self):
        self.device = self.device_descriptor.getDevice()
        self.handle = self.device.open()
        self.handle.claimInterface(self.device_descriptor.interface_id)
...
    def setLCDL1(self, msg):
        ret = self.handle.controlMsg(self.VENDOR_REQUEST_OUT,
                                     self.VRQ_SET_LCD_L1,
                                     msg,
                                     value=0,
                                     index=0,
                                     timeout=100)
...
```

Python Code Snippet for finding a USB Device

Questions?