USB in Embedded Systemen

Referat von Peter Voser
Embedded Development GmbH

Winterthur, 27. August 2013
swissT.meeting
Embedded Computing Conference

Embedded Development GmbH
Engineering and Development

- System Engineering
- Hardware/Software Co-Design
- Embedded Software Entwicklung
- Erfahrung aus diversen Branchen

Inhalt

- USB Grundlagen

  - Host und Device
  - Geschwindigkeiten
  - Endpunkte

- Transfer-Typen – der USB Baukasten

  - Wann verwendet man welchen Transfer-Typ?

- Aus Endpunkten werden Geräteklassen

  - Beispiele von Embedded Systemen mit USB aus der Praxis

Host und Device
Die asymmetrische USB Welt

- **Device**

  - Stellt Funktionalität zur Verfügung (Speicher, Drucker, Embedded System, …)
  - Beantwortet Kommunikation (Standard Requests)
  - Verwaltet den Stromverbrauch (Bus-powered, Power States)

- **Host**

  - Konsumiert Funktionalität – zum Beispiel der PC
  - Detektiert Geräte und lädt dazugehörige Treiber
  - Verwaltet die Kommunikation und Bandbreite zu allen Geräten
  - Stellt Versorgungsspannung zur Verfügung
  - Initiiert Kommunikation

Geschwindigkeiten
Keiner zu klein, ein USB Gerät zu sein

- **LowSpeed** 1.5 MBit/s
- **FullSpeed** 12 MBit/s
- **HighSpeed** 480 MBit/s
- **SuperSpeed** 5 GBit/s

Endpunkte
Der USB Baukasten

- **Endpunkt ist Ausgangspunkt oder Senke von Daten im Device**
- Control Endpunkt (Endpunkt 0) ist obligatorisch für alle Devices

  - Host erkennt Device mittels Standard Requests (Deskriptoren)
- Weitere Endpunkte

  - abhängig vom Gerätetyp
  - jeder Endpunkt hat genau eine Richtung (OUT: H → D, IN: D → H)
  - jeder Endpunkt hat genau einen *Transfer-Typ*

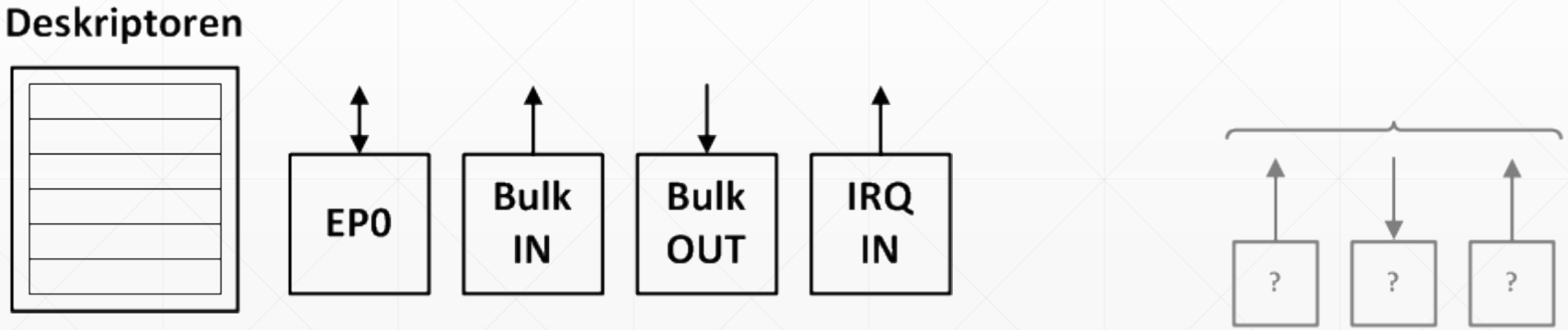
Jeder Endpunkt hat einen Transfer-Typ
Der USB Baukasten

	Control	Bulk	Interrupt	Isochronous
Garantierte Bandbreite	Low/Full: 10% High/Super: 20%	✗	Low/Full: 90% High/Super: 80%	
Max. Paketgrösse	Low: 8 Full: 64 High: 64 Super: 512	Low: × Full: 64 High: 512 Super: 1024	Low: 8 Full: 64 High: 1024 Super: 1024	Low: × Full: 1023 High: 1024 Super: 1024
Garantierte Pakete/Zeit	✗	✗	Low: 1 / 10 ms Full: 1 / ms High: 3 / 125 µs Super: 3 / 125 µs	Low: × Full: 1 / ms High: 3 / 125 µs Sup.: 48 / 125 µs
Fehlerkorrektur	✓	✓	✓	✗

Aus Endpunkten werden Geräteklassen

- **Geräteklassen definieren Protokolle für USB Geräte mit ähnlichen Diensten**
- Spezifikationen sind öffentlich zugänglich (*Approved Device Classes*)
  http://www.usb.org/developers/devclass_docs
- Alle Klassenspezifikationen haben folgende Definitionen gemeinsam:

  - ID’s der Deskriptoren
  - Endpunkte



CDC
Communication Device Class

- Kapitel *ACM - Abstract Control Model*

  - USB-to-serial
  - Virtueller COM Port

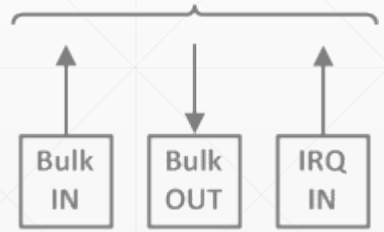
- 2 Bulk Endpunkte für Rx/Tx

- 1 Interrupt Endpunkt für Notifications

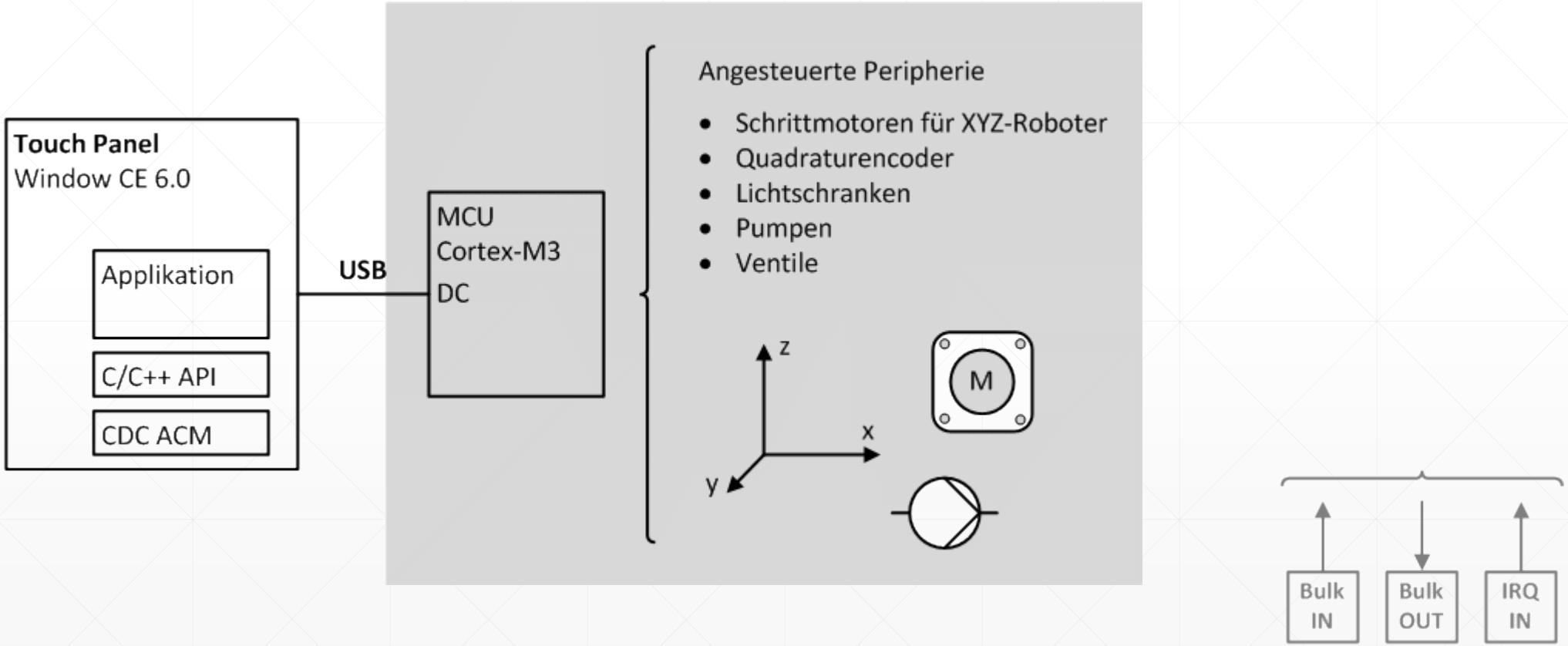
  - Beispiel: Serial State Change (DSR)

- Klassen-spezifische Requests (über EP0)

  - Beispiel: Set Line Coding



CDC ACM Beispiel
Management Schnittstelle für Peripherie



MSC
Mass Storage Class

- Besteht nur aus 2 Bulk Endpunkten

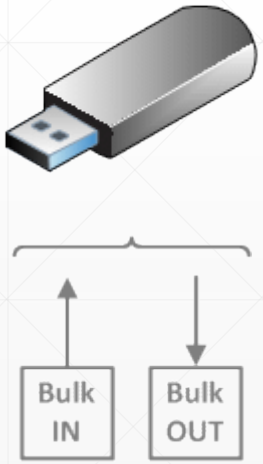
  - BOT
  - CBI (Control/Bulk/Interrupt) wird nicht mehr verwendet

- Zugriff auf Massenspeicher über SCSI Kommandos

  - Framework für Ansteuerung
  - Read/Write Block

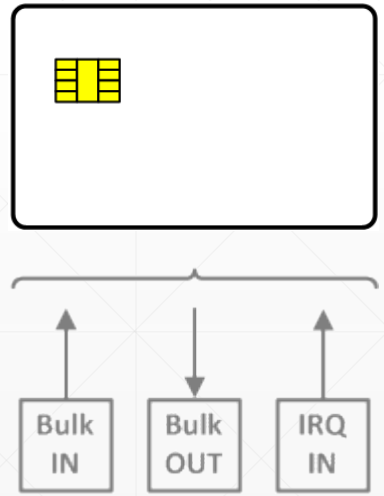
- Unabhängig vom darüberliegenden File System

  - Massenspeicher ist sichtbar als Block Device



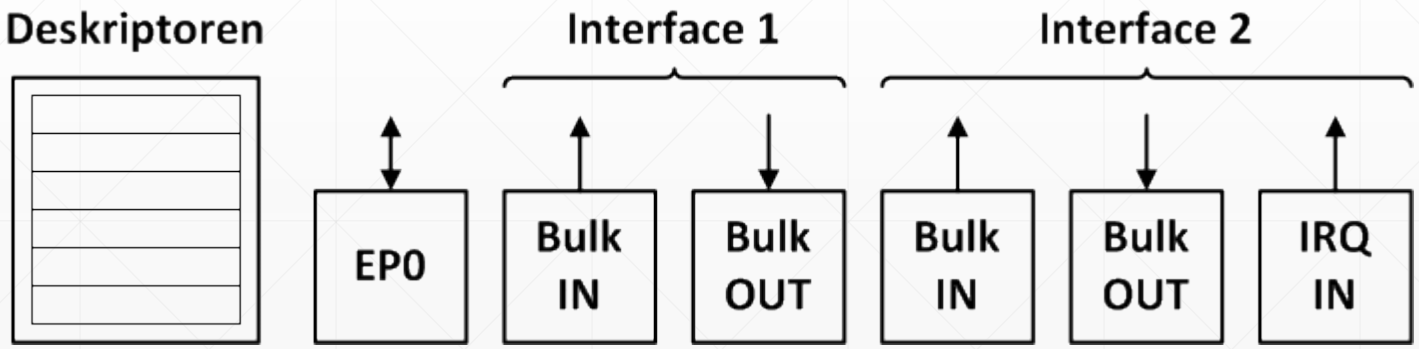
CCID
Smart Card Class

- CCID – *Integrated Circuit Cards Interface Device* – Smart Card Class
- 2 Bulk Endpunkte
  - 1 Command und 1 Response-Pipe
  - Lesegerät Steuerung (Kartenspeisung ein/aus etc.)
  - Kommunikation zur Smart Card (APDU/TPDU)
- 1 Interrupt Endpunkt für Notifications
  - Karte eingesetzt/entfernt
  - Hardwarefehler (Überstrom etc.)

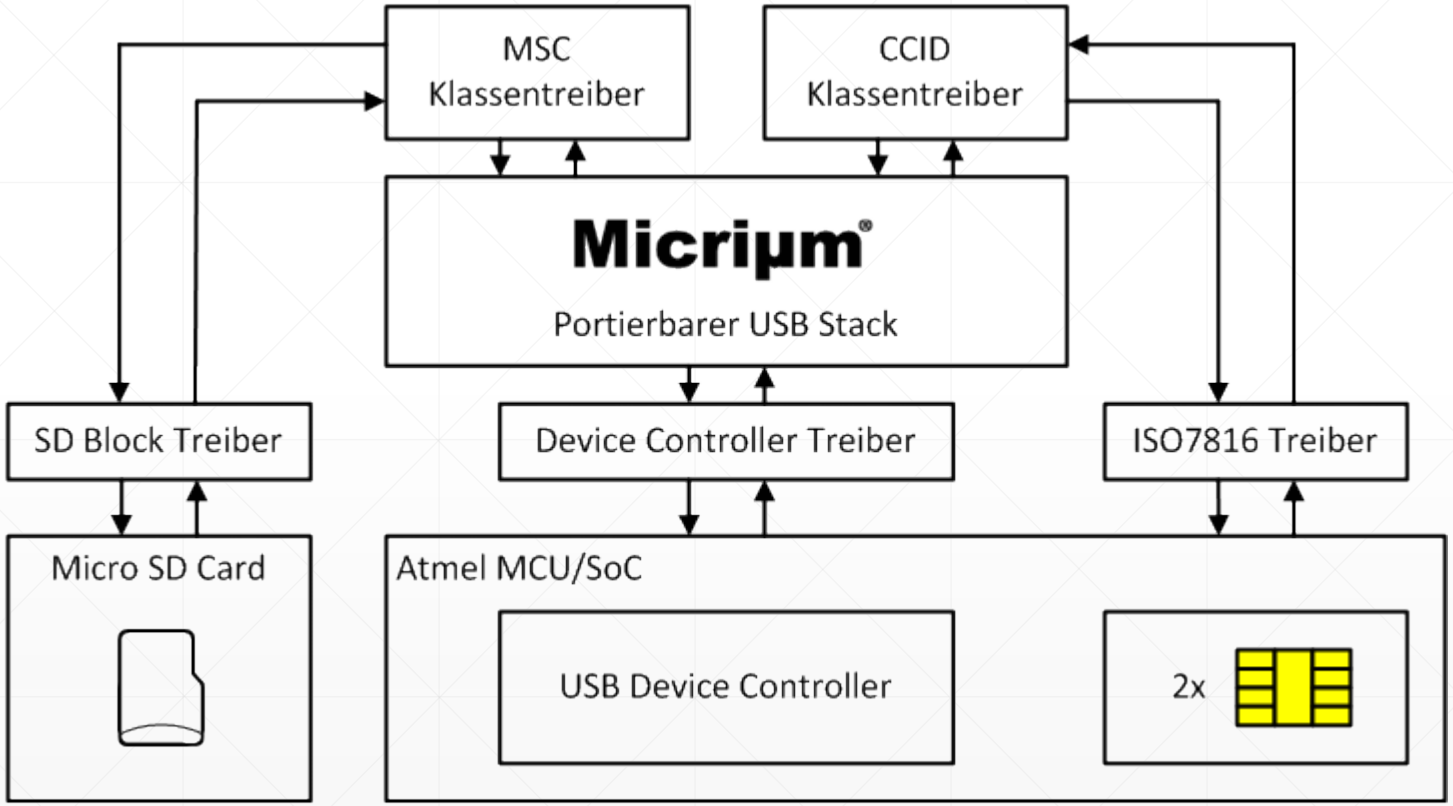


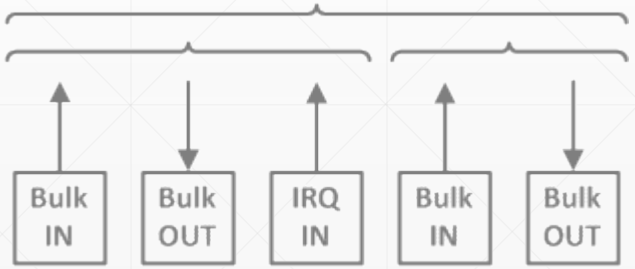
Composite Device
Zwei Klassen im gleichen Gerät – warum nicht?

- IAD – *Interface Association Descriptor* wird benötigt
- Unterschied zu *Compound Device*: kein Hub benötigt
- Windows CE 6.0 unterstützt IAD
  - ActiveSync und RNDIS zusammen



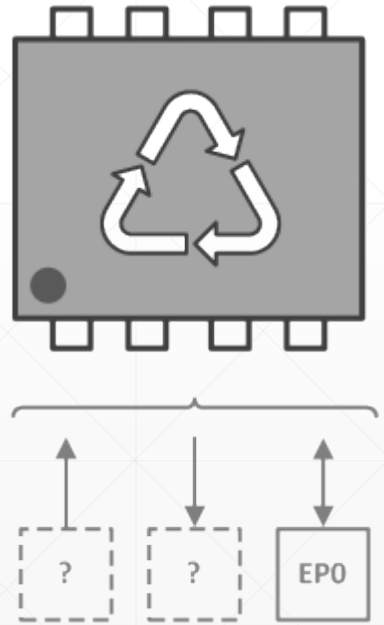
MSC/CCID Beispiel
Smart Card Lesegerät iQey mit Speicher und API





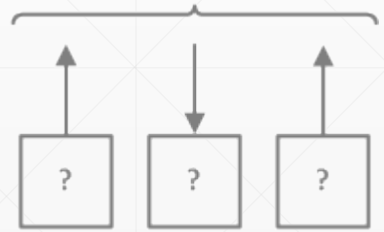
DFU
Device Firmware Update Class

- Ideal für Firmware Updates kleiner Systeme über USB
- Das Protokoll benötigt nur EP0 (Default Control Endpoint)
- Deshalb besonders geeignet für Composite Devices

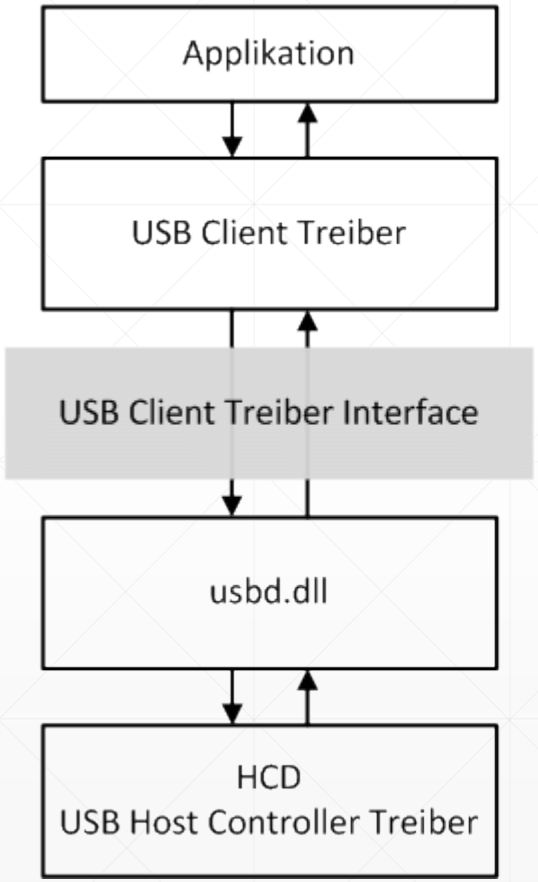


Vendor Class
Selbstdefinierte Geräte

- Host-seitig ist Verwendung von Standard Client-Treibern nicht mehr möglich
- Client Treiber Eigenimplementierungen
- Windows CE
  - Laden über Registry Settings (PID/VID/Rel, Dev./If.-Class/SubClass/Protocol)
  - Programmierschnittstelle im DDK
- Linux
  - Laden über vorgängige Registrierung beim Kernel
  - Programmierschnittstelle im Kernel
- Wiederkehrende Konzepte: libusb, WinUSB, Android USB Host API, …



Windows CE USB Host Stack
Vom Device Attach bis zur Applikation




```
#include "usbclient.h"
```

- Deskriptoren des USB Geräts abfragen
- Steuerung des USB Geräts über EP0
- Transfers

- Zuständigkeit usbd.dll: passenden Client Treiber laden
- **`LoadDeviceDrivers()`** in usbd.cpp
- Ladestrategie:

  1. VID/PID/Release
  2. Klassen ID's
  3. Subklassen ID's

Fragen

Embedded Development GmbH

www.embedded-development.ch