Fachhochschule Augsburg
Fachbereich Informatik

Präsentation der Diplomarbeit

zum Thema

Kommunikation von Linux-Applikationen mit generischer Hardware über das USB-Subsystem, praktisch realisiert am Beispiel einer USB-zu-Mikroprozessor und einer USB-zu-CAN Schnittstelle

13.07.2004 um 10.00 Uhr, Raum J101

Aufgabensteller: Prof. Dr. Hubert Högl

Bearbeiter: Anatolij Gustschin

Überblick

- Aufgabenstellung
- USB Grundlagen (USB 1.1)
- USB-AVR Interface (AVRPIPE), Libavrp
- USB-CAN-Realisierung (Software)
  - CAN-Grundlagen (kurz)
  - USB-CAN-Interface (UCI), USBCAN-Treiber, Libusbcan
- Demonstration
- Literatur, Info-Quellen

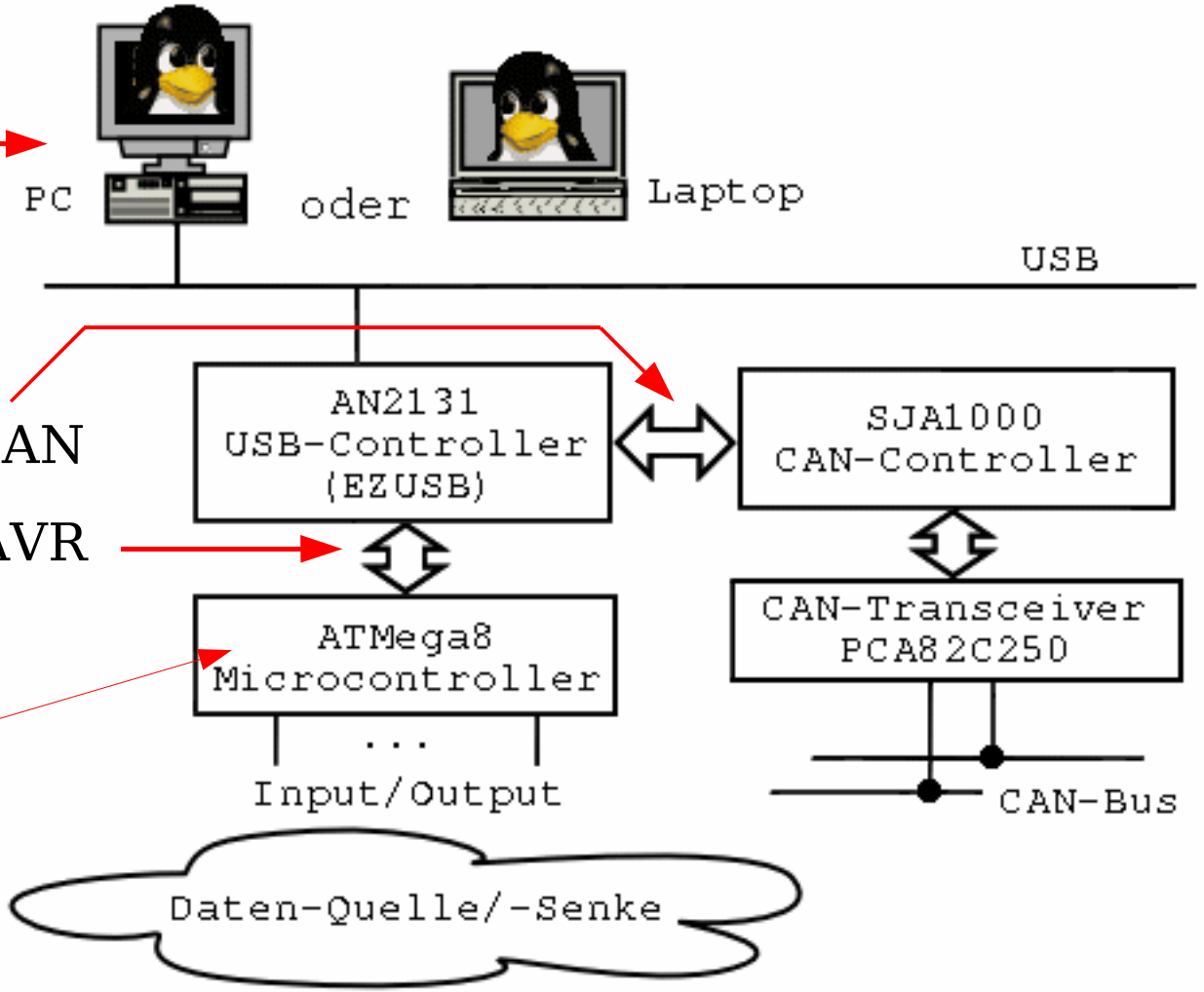
Aufgabenstellung

USB-Treiber
und API

Schnittstellen-
Programme
für Ez-Usb

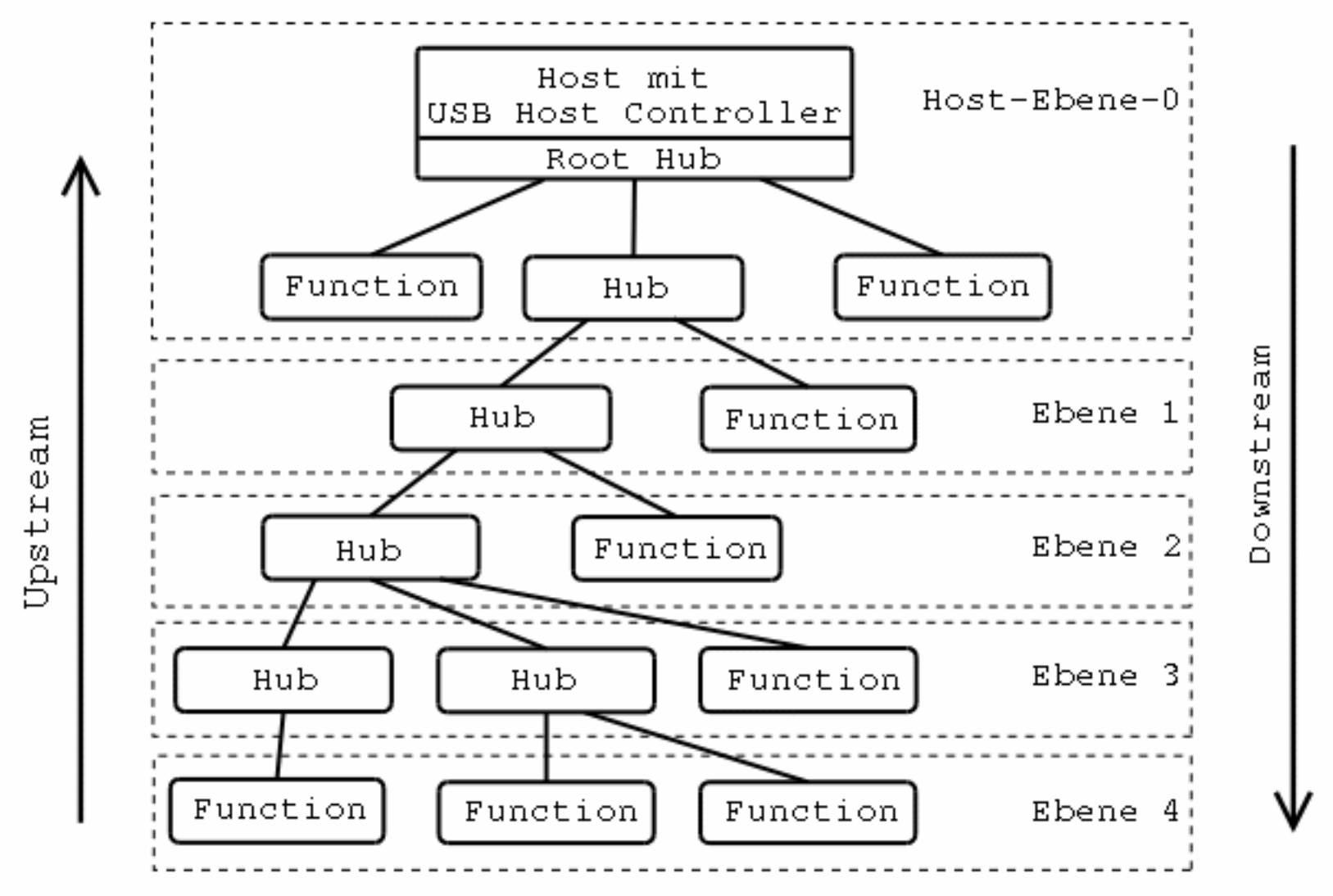
- USB-CAN
- USB-AVR

Schnittstellen-
Programm im AVR



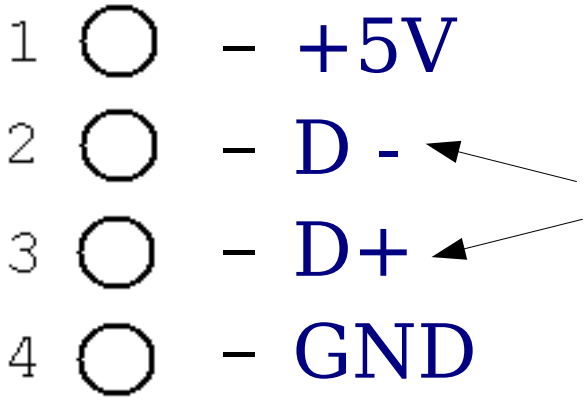
USB-Grundlagen (USB 1.1)

USB-Topologie



USB-Grundlagen (2)

- Single Master Bus (Host ist Master)
- Bis zu 127 USB-Geräte
- 12 MBit/s (ca. 9,7 MBit/s bei einem Gerät)
- Buskabel mit 4 Leitungen (bis zu 5m lang):



Differenz-Signalleitungen

Stromversorgung über den USB-Bus (bis zu 500 mA)

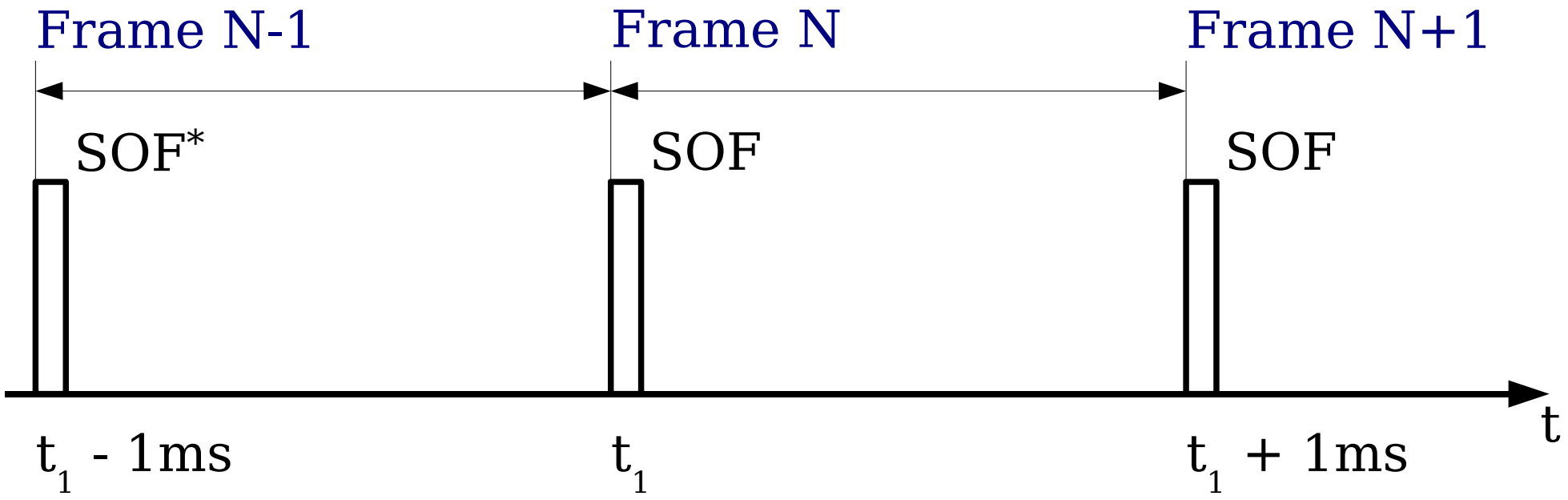
USB-Grundlagen (3)

USB-Transferarten:

- **Control Transfer** (max. 10% der Bandbreite)
  - Konfiguration, Steuerung
- **Interrupt Transfer** (max. 90%)
  - Statusinformationen
- **Bulk Transfer** (keine Bandbreite reserviert)
  - Schnelle Übertragung großer Datenmengen
- **Isochronous Transfer** (max. 90% mit Interrupt)
  - Übertragung mit garantierter Bandbreite, allerdings keine Fehlerbehandlung (z.B. Audiodaten)

USB-Grundlagen (4)

Aufteilung der Bus-Bandbreite

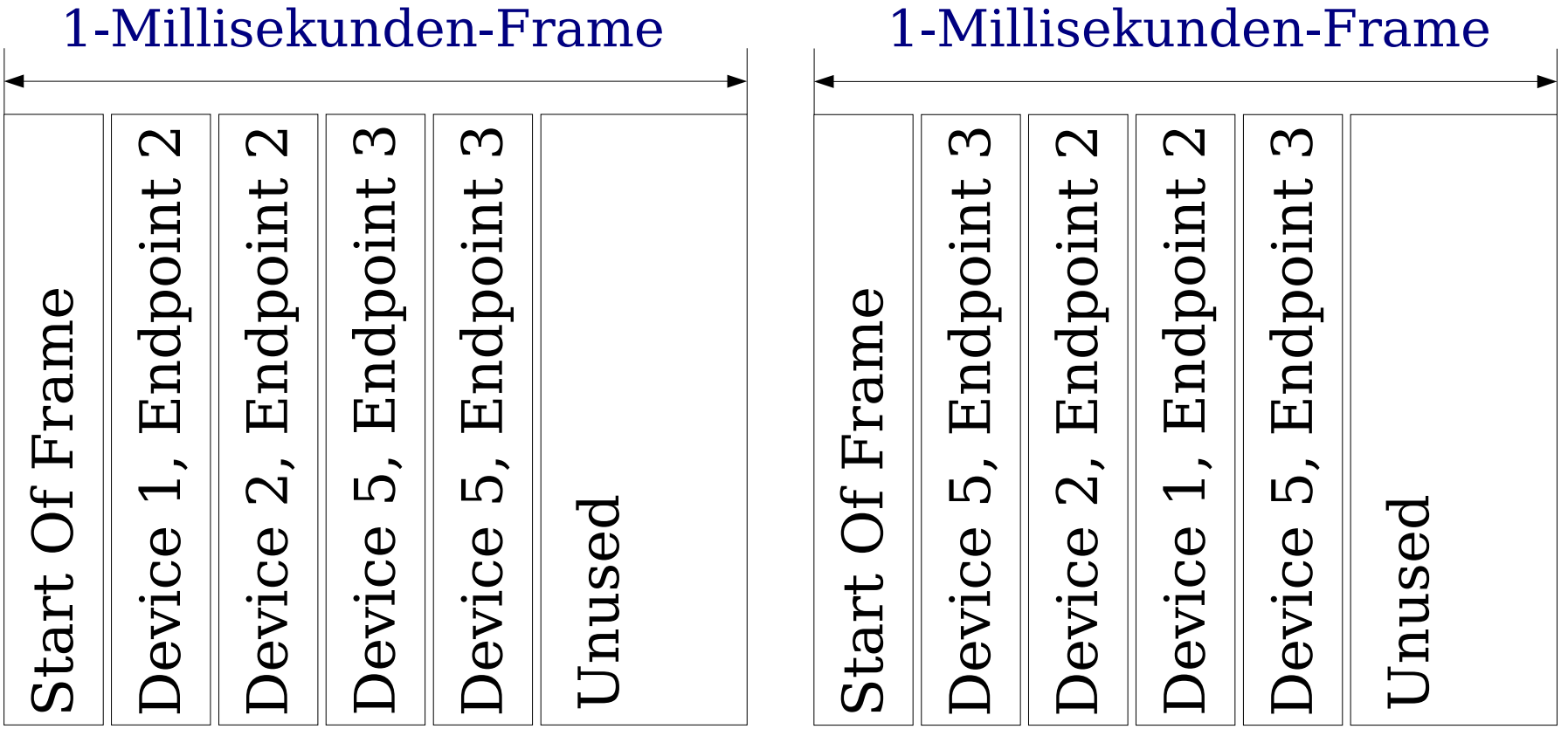


Frame-Sequenz

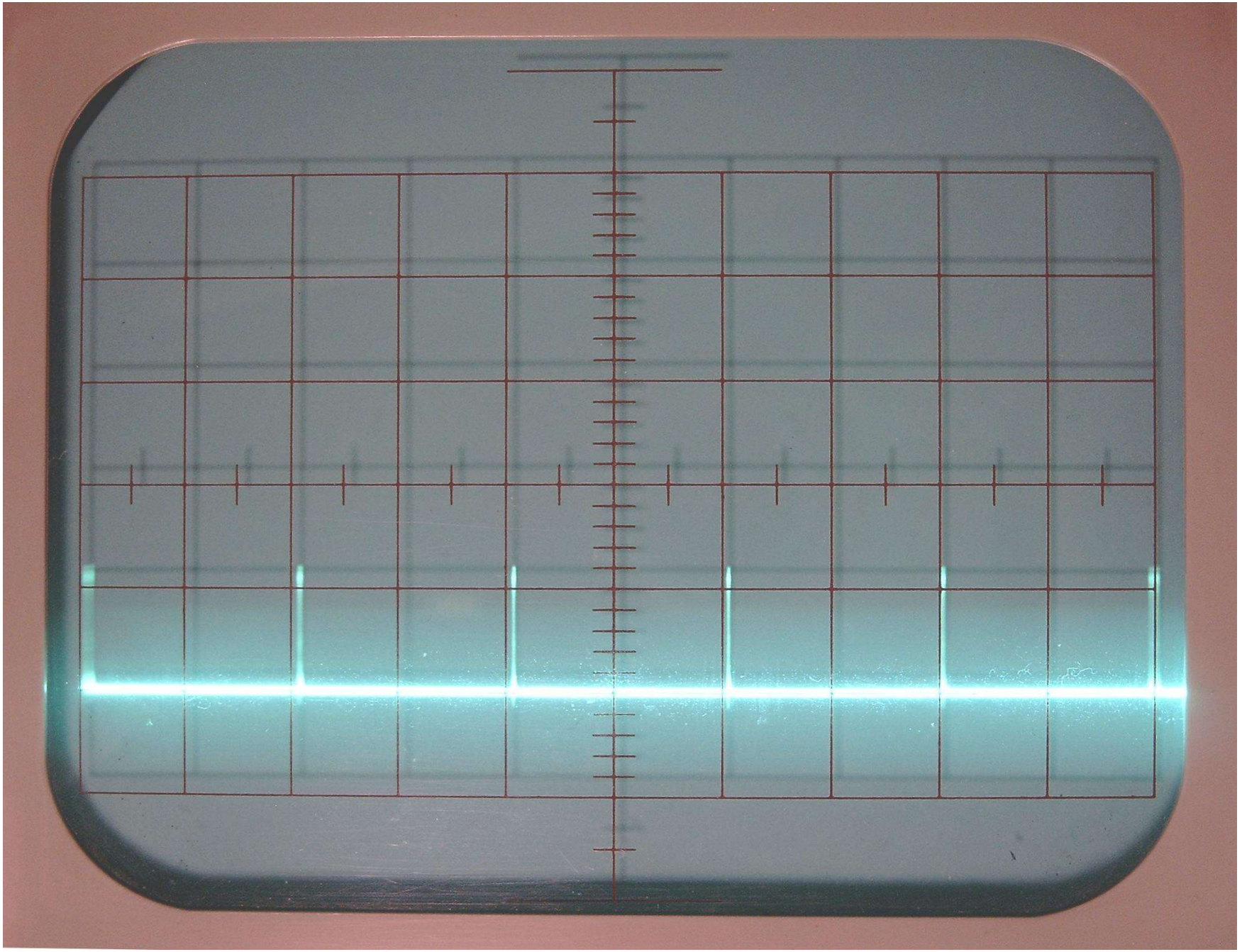
* SOF: Start-Of-Frame Token

USB-Grundlagen (5)

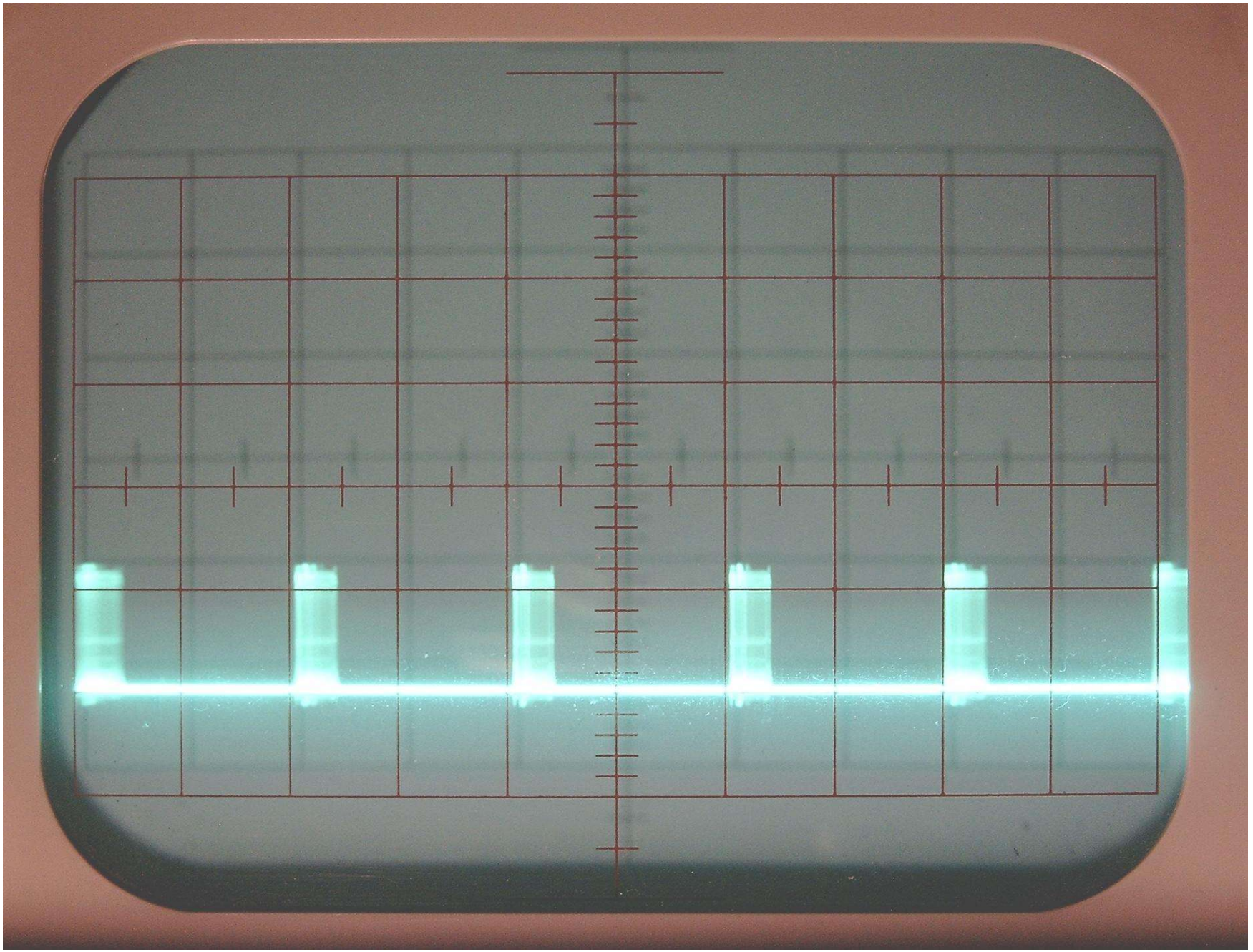
Aufteilung von Transaktionen auf Frames



Frame-Sequenz



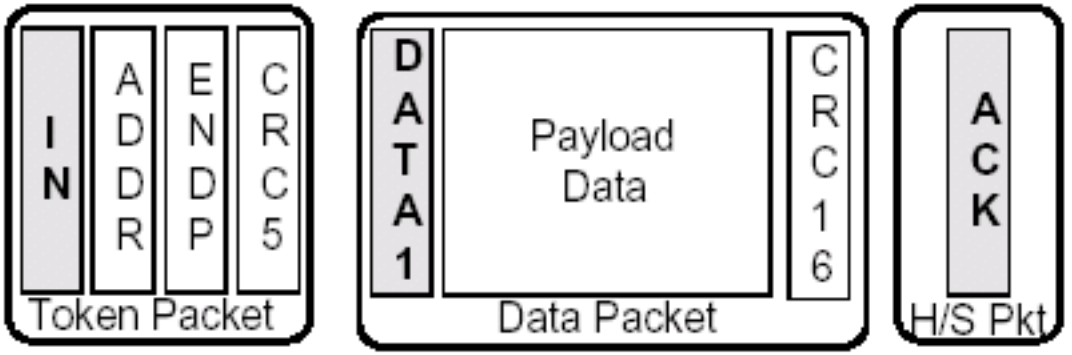
Aufteilung von Transaktionen auf Frames



USB-Grundlagen (6)

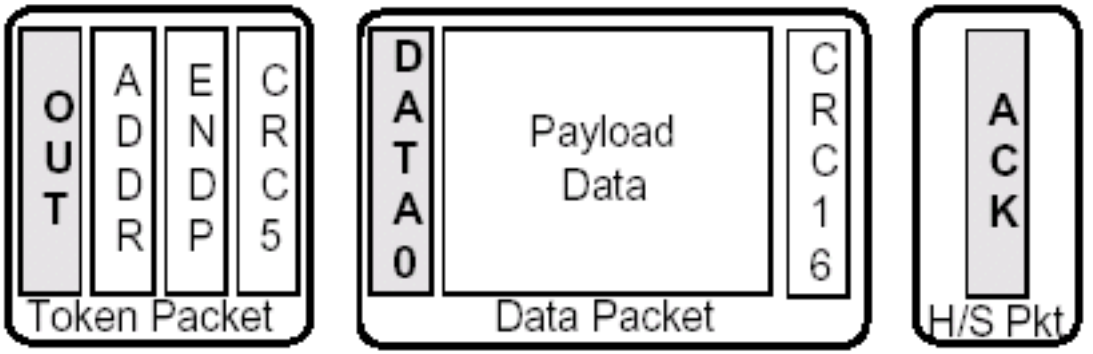
Aufbau von Transaktionen

IN-Transfer



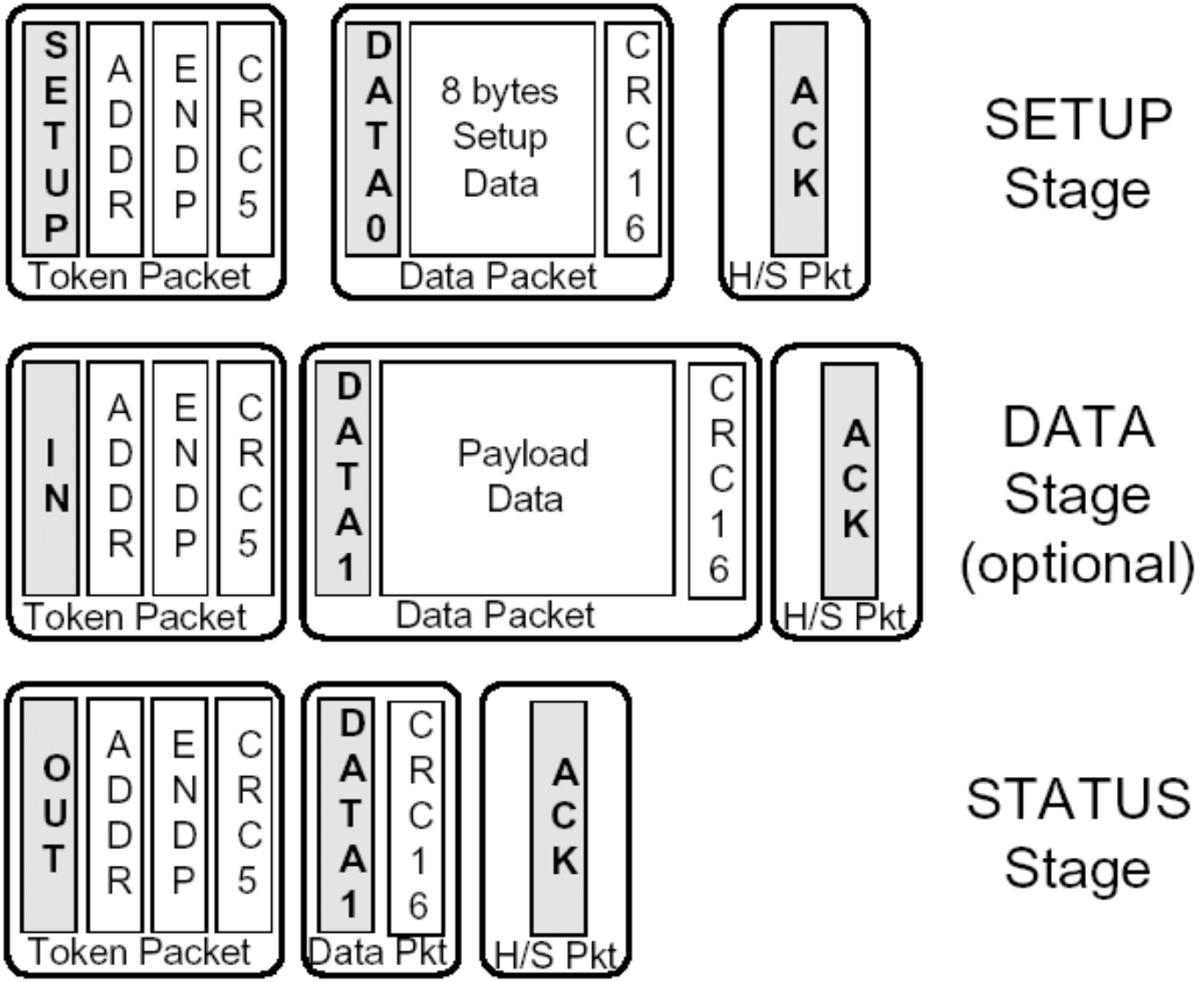
OUT-Transfer

Bildquelle: EZ-USB Technical Reference Manual



USB-Grundlagen (7)

Control Transfer (Bildquelle: EZ-USB Technical Reference Manual)



USB-AVR Interface (AVRPIPE)

- **Auf der Hostseite:** User-Mode Treiber basierend auf **Libusb** überträgt Daten von/zu Ez-Usb
  - Vorteile: + einfache Realisierung
    + verwendbar mit neuerem Kernel (2.6.x)
- **In Ez-Usb:** Schnittstellenprogramm überträgt Pufferdaten von/zu AVR (synchron)
- **In AVR:** Schnittstellenprogramm überträgt Pufferdaten von/zu Ez-Usb auf dessen Anforderung

USB-AVR Interface (AVRPIPE)

Host

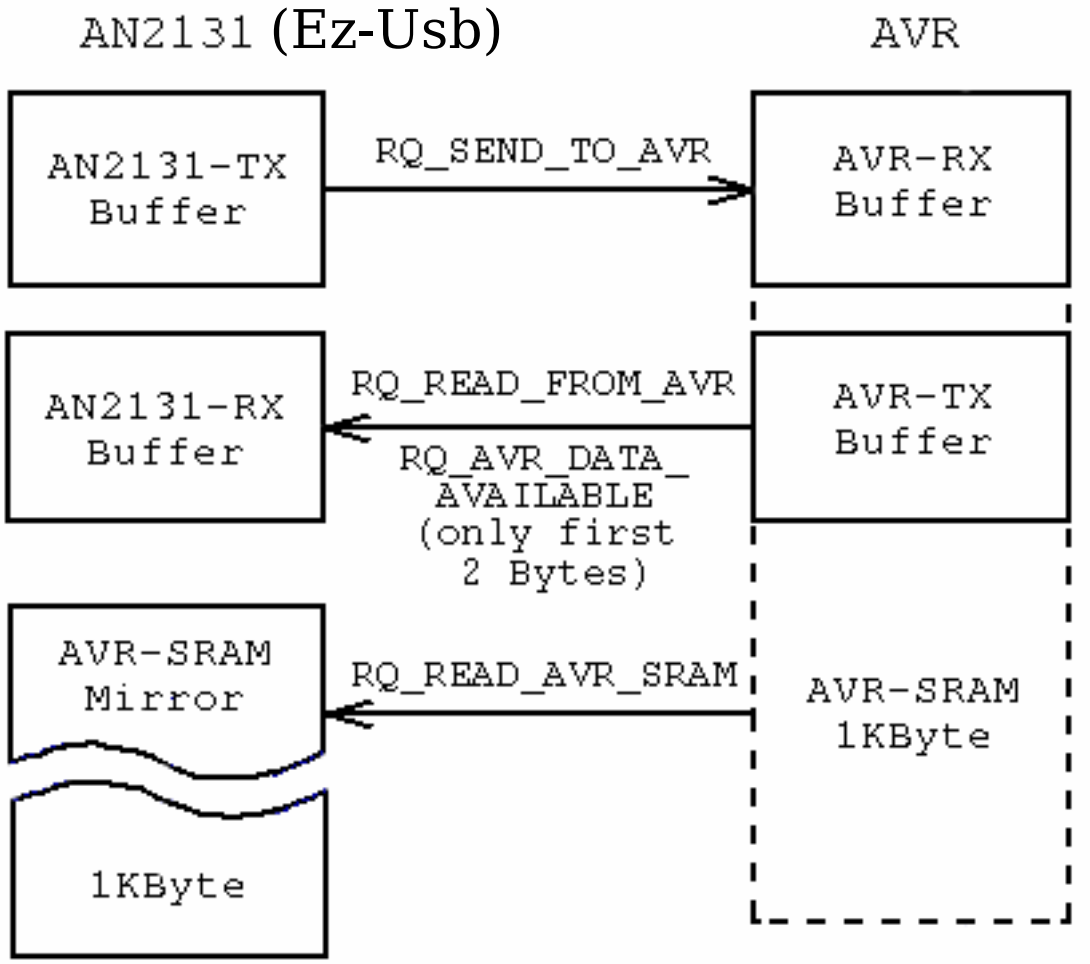
-Überträgt Pufferdaten von/zu Ez-Usb

-Sendet Anforderungen (Requests) zum Starten des Transfers zw. Ez-Usb und AVR:

- RQ_SEND_TO_AVR
- RQ_READ_FROM_AVR
- RQ_AVR_DATA_AVAILABLE
- RQ_READ_AVR_SRAM

-Puffer-Größen:

- 32, 64, 128 und 256 Byte



USB-AVR Interface (AVRPIPE)

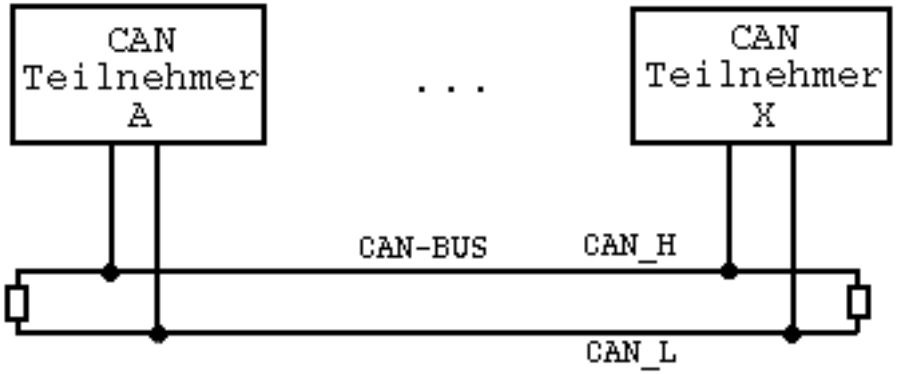
- Ez-Usb Schnittstellenprogramm wird beim Anschließen vom Hot-Plug-System zum Gerät übertragen und gestartet (**ezusb_ldr**, **an2131ctl**)
- AVR Schnittstellenprogramm befindet sich im Flash-Speicher im AVR (Programmierung mit **avrdude** über Parallelport des PC)

USB-AVR Interface (AVRPIPE)

- **Libavrp** – API zur Anwendungs-Entwicklung
  - int **open_avr** (int dev_num);
  - int **close_avr** (int dev_num);
  - int **send_to_avr** (int dev_num, unsigned char * buffer, unsigned short len);
  - int **read_from_avr** (int dev_num, unsigned char * buffer);
  - int **avr_data_available** (int dev_num);
  - int **read_avr_sram** (int dev_num, unsigned char * buffer);

CAN-Grundlagen

- CAN-Bus-Topologie



- CAN-Nachrichten (Telegramme, Messages)
  - 11-Bit bzw. 29-Bit Identifier ($0..2^{29}$)
  - 8-Byte Datenfeld
  - Remote-Nachrichten (ohne Datenfeld)
  - Übertragung mit bis zu 1 MBit/s (10, 20, 50, 100, 125, 250, 500, 1000 Kbit/s)

USB-CAN-Realisierung

- **USB-CAN-Interface (UCI):** Schnittstellen-programm im Ez-Usb Chip
- **USBCAN-Treiber** auf dem Host (Kernel-Mode)
- **Libusbcan:** Bibliothek zur Anwendungsent-wicklung (kapselt Treiberschnittstelle)

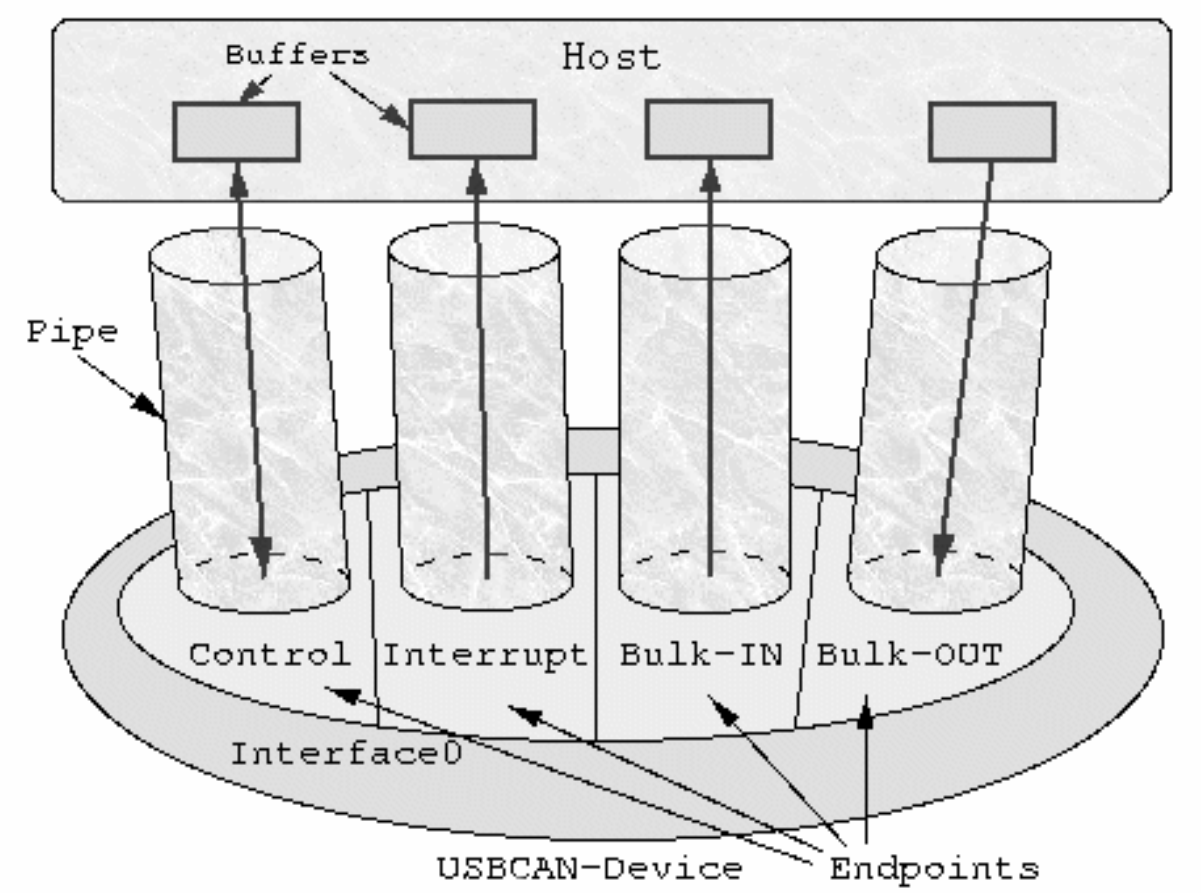
USB-CAN-Realisierung - UCI

UCI:

- wird beim Anschließen in den Ez-Usb geladen (vom Linux-Hot-Plug System)
- simuliert ein „Reconnect“ und beantwortet die Anfragen des Hosts zur Geräte-Identifikation (liefert alle notwendigen Deskriptoren)
- steuert den CAN-Controller (Kommandos vom Treiber)
  - Starten, Anhalten, Bit-Rate und Nachrichten-Filter konfigurieren, etc.
- übernimmt:
  - CAN-Nachrichtentransfer (USB <---> CAN)
  - Behandlung und Mitteilung von Fehlern
- bietet Zugriff auf Register des CAN-Controllers (Debug)

USB-CAN-Realisierung – UCI (2)

UCI Endpoint-Layout



- **Control-Transfers** für Gerätesteuerung
- **Interrupt-Transfer** für Status-Info und Fehlermeldungen
- **Bulk-Transfers** für CAN-Nachrichten

USB-CAN-Realisierung - USBCAN-Treiber

USBCAN-Treiber:

- bisher mit Linux-Kernel 2.4.x und UHCI/OHCI (Kernel 2.6.x geplant)
- wird automatisch geladen
- kann bis zu 16 USBCAN-Geräte bedienen
- unterstützt alle CAN-Bitraten (bis 1 MBit/s)
- verwaltet Puffer für abgehende und ankommende CAN-Nachrichten
- signalisiert Nachrichtenempfang und Fehler an die Anwendung
- bietet Schnittstelle zur Steuerung des USBCAN-Geräts

USB-CAN-Realisierung - Libusbcan

- Libusbcan kapselt Treiberaufrufe
- verwendete CAN-Nachrichten-Struktur:

```
typedef struct {
    unsigned long id;     /* message ID */
    unsigned int flags;   /* std/ext, rtr, dlc */
    unsigned char msgdata[CAN_MSG_LENGTH];
    struct timeval ts;    /* time stamp for received msg */
} canmsg_t;
```

USB-CAN-Realisierung – Libusbcan (2)

Funktionen der Bibliothek:

int **can_init** (void);

int **can_open** (int dev_num, unsigned char flags);

int **can_close** (int dev_num);

int **can_start** (int dev_num);

int **can_stop** (int dev_num);

int **can_set_callback** (int dev_num, unsigned char type,
void(*callback)(unsigned long));

int **can_set_baud** (int dev_num, unsigned char baud);

int **can_set_acc_mask** (int dev_num, unsigned long mask);

int **can_set_acc_code** (int dev_num, unsigned long code);

USB-CAN-Realisierung – Libusbcan (3)

Funktionen der Bibliothek:

int **can_send_msg** (int dev_num, canmsg_t *msg);

int **can_receive_msg** (int dev_num, canmsg_t *msg);

int **can_send_msg_buf** (int dev_num,
canmsg_t *msg_buf, int size);

int **can_receive_msg_buf** (int dev_num,
canmsg_t *msg_buf, int size);

int **can_reset** (int dev_num);

void **can_print_err** (unsigned int err);

void **can_debug_on** (int level);

Literatur

[1] USB 2.0, Hans Joachim Kelm (Hrsg.)
Franzis' Verlag GmbH, 85586 Poing, 2001

[2] CAN; Grundlagen, Protokolle, Bausteine, Anwendungen
Konrad Etschberger München, Wien: Hanser, 1994

[3] CAN Controller Area Network; Grundlagen und Praxis
Wolfhard Lawrenz (Hrsg.) Heidelberg: Hüthig, 1994

[4] Programming Guide for Linux USB Device Drivers
Detlef Fliegl, 2000 http://usb.cs.tum.edu/usbdoc

[5] Linux Device Drivers, 2st Edition
A. Rubini, J. Corbet, O'Reilly & Associates, Inc.

[6] Understanding the Linux Kernel
Daniel P. Bovet & Marco Cesati
O'Reilly & Associates, Inc., 2001

Literatur

Außerdem wurden folgende Datenblätter verwendet:

[7] EZ-USB Technical Reference Manual, Version 1.10,
Cypress Semiconductor

[8] SJA1000 Stand-alone CAN controller,
Philips Semiconductors, 2000

[9] Atmel ATmega8(L) Datasheet,
Atmel Corporation, 2003

[10] AVR Instruction Set, Atmel Corporation, 2002

[11] USB-Tiny-CAN Schaltplan, Version 1.0,
MHS-Elektronik, 2004