PDV
Prozessdatenverarbeitung

Fachhochschule Ulm

FHU

Prof. Dipl.-Ing. Gerhard Schaub

Dieses Skript soll das Mitschreiben bei der Vorlesung erleichtern und konzentrierte Mitarbeit ermöglichen. Es ist kein Daten- und Lehrbuchersatz!

Ziel der Lehrveranstaltung

Übersicht über Methoden des Computereinsatzes für technische Zwecke
Behandlung neuer Techniken
z.B. Realzeit-Betriebssysteme, Java, Steuerung über Internet, Feldbusse

Lehrbuchempfehlung zur Vorlesungsbegleitung

Jacobson:	Einführung in die PDV	Hanser-Verlag
Küveler +..	Informatik für Ingenieure	Vieweg-Verlag
Kelm	USB Universal Serial Bus	Franzis-Verlag
Krüger	GoTo Java 2	Addison-Wesley
Cornell:	Java bis ins Detail	Heise-Verlag
Murray:	Inside Microsoft Windows CE	Microsoft Press
Schnell:	Bussysteme in der Automatisierungst.	Vieweg-Verlag
Lawrenz:	CAN Controller Area Network	Hüthig-Verlag
Furrer:	Ethernet-TCP/IP für die Ind.-Autom.	Hüthig-Verlag

Quellenangaben

Infineon:	Technische Daten
Texas Instruments:	User Manual 320xx
Zartmann, Bernd:	Diplomarbeit ‚CAN-Bus' WS 99/00
Mentor Graphics:	Folien zu Windows CE
3SOFT GmbH, Erlangen:	Firmeninformation über Java/Jini
Ing.-Büro Dr. Kaneff, Nürnberg:	Handbuch zu EUROS
Ing.-Büro Dr. Kaneff, Nürnberg:	Folien zu EUROS

Inhalt

1 Prozesse und ihre Daten

1.1 Begriffe

Begriffe

EDV	**E**lektronische **D**aten **V**erarbeitung. Der Rechner ist Hilfsmittel zur Verarbeitung von Daten.
PDV	**P**rozess **D**aten **V**erarbeitung. Der Rechner ist vorwiegend Hilfsmittel zur Lenkung von Prozessen (Steuerung in der offenen Steuerkette, Regelung im geschlossenen Regelkreis).
Prozeß	Umformung und / oder Transport von Materie, Energie und / oder Information. Hier interessieren vorwiegend technische Prozesse.
Prozessrechner	Rechner, der mittels Prozessperipherie direkt an einen Prozess gekoppelt ist. Bei **Kleingeräten** kann dies ein Microcontroller sein, bei **großen Anlagen** mehrere vernetzte Rechner. Soweit sinnvoll, strebt man Dezentralisierung an. Zur Visualisierung des Prozesszustands ist der PC sinnvoll, zu Regelungszwecken wegen der kürzeren Reaktionszeiten meist Controller oder Signalprozessoren.
IPC	Industrie-PC: mechanisch robust, modular mit Einschubtechnik. Er macht der SPS zunehmend Konkurrenz.

1.2 Prozessmodelle

In einem Prozessmodell werden die **Struktur des Prozesses** und die Zusammenhänge zwischen Eingangs-, Ausgangs- und Zustandsdaten beschrieben. Das Prozessmodell kann gegenständlich (verkleinertes Labormodell) oder abstrakt (Struktogramme oder Diagramme) sein.
Zunehmend wird der Prozess am Computer **simuliert**.

Prozeßbeschreibung:

Der Prozess wird durch die **Zerlegung in Teilprozesse** strukturiert. Man erhält **Elementarprozesse**, die zusammenwirken.

- **Statische Strukturen** werden meist durch Blockschaltbilder dargestellt, in denen die Prozessgrößen definiert und beschrieben werden. Sie zeigen die Grobstruktur der Konstruktion.
- **Stationäre Abläufe** können als Wirkungs- oder Transportwege in Struktogrammen dargestellt werden. Diese beschreiben die Zusammenhänge der Prozessgrößen für den Fall, daß alle Größen stabil sind.
- **Dynamische Strukturen** sind schwierig darzustellen. Mit Methoden der Automatentheorie (nicht behandelt) kann man die zeitlichen Veränderungen der Prozessgrößen beschreiben.

1.3 Daten

1.3.1 Prozessdaten

Sie werden zwischen Prozess und Prozessrechner ausgetauscht und sind dem Rechner zugeordnet.

1.3.2 Prozessgrößen

Sie beschreiben einen technischen Prozess unabhängig von seiner Lenkung durch einen Prozessrechner. Es sind technische, physikalische oder chemische Grössen:

- **Prozesskennwerte oder Anlagendaten** sind Festwerte, z.B. die Phasenzeit einer Verkehrsampel.
- **Prozesszustandswerte und -Daten** sind Variable, die den aktuellen Zustand des Prozesses beschreiben, z.B. der Stauzustand vor einer Ampel.
- **Prozessparameter** sind Grössen, die für einen Prozesslauf fest sind, für den nächsten aber geändert werden können, z.B. das Verhältnis zwischen Grün- und Rotphase einer Verkehrsampel.
- **weitere Betriebsgrössen** sind Eingangsgrössen, Stellgrössen, Störgrössen, Ausgangsgrößen, Messgrössen und Alarme.

2 Prozessrechner Hardware

2.1 Einführung

2.1.1 Anforderungen an den Prozessrechner

Der Prozessrechner ist mittels Prozessperipherie direkt an den Prozess gekoppelt. Jedes Rechnersystem ist geeignet, welches die Anforderungen des Prozesses erfüllt. Dazu gehören:

- **Realzeitfähigkeit**. Der Rechner muß auf Ereignisse (Alarme) im Prozess hinreichend schnell reagieren (Mikrosekunden bis Stunden).
- Möglichkeit zum Setzen von **Prioritäten** beim Ablauf.
- **Speichergröße**, um die Datenmengen verlustlos zu bewältigen.
- geeignete **Schnittstellen**.

2.1.2 Embedded Systeme

Ein **Embedded System** ist ein kleines Computersystem mit definierter Aufgabe.
Es kann unabhängig oder Teil eines grösseren Systems sein.
Seine Funktionalität ist von andern Teilen des Systems abhängig.

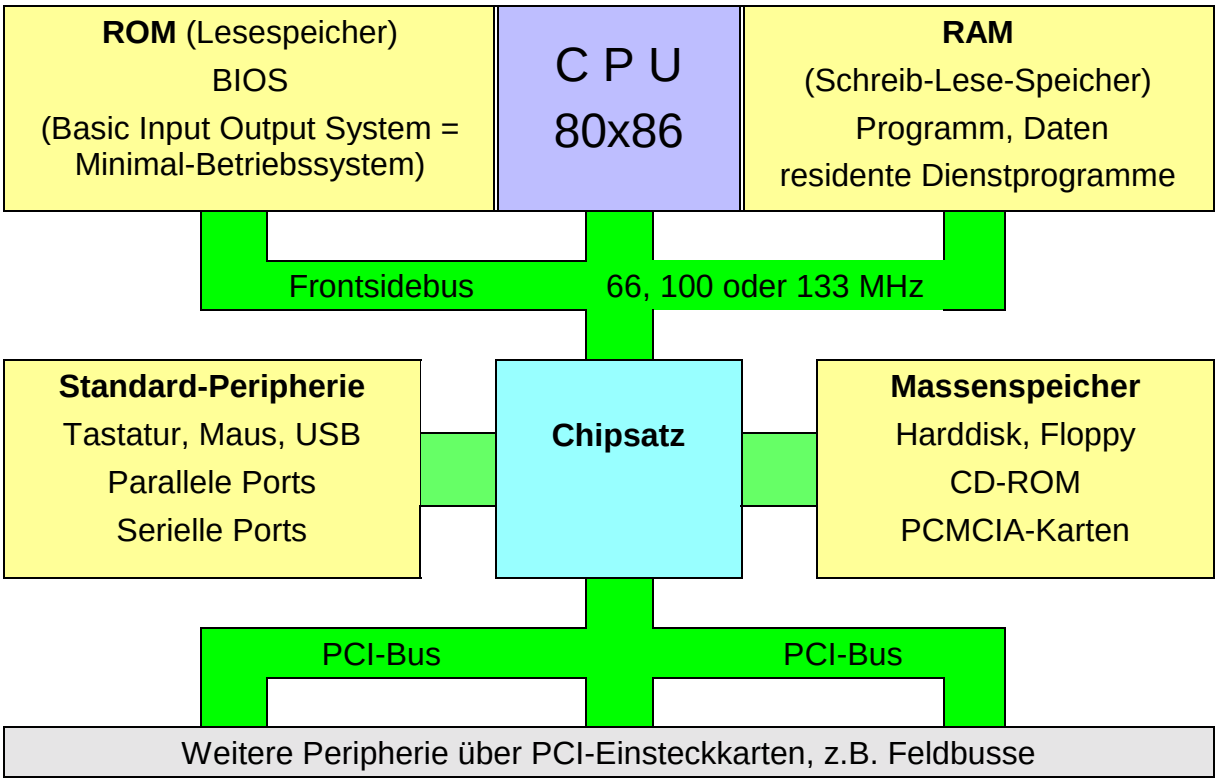
Ein **dezentrales System** kann aus mehreren Embedded Systemen bestehen, die vernetzt sind. Zur Vernetzung werden zunehmend **Feldbusse** verwendet.

Eigenschaften:	4-, 8-, 16-, 32- und 64-bit-Systeme Microcontroller mit unterschiedlicher Peripherie Digitale Signalprozessoren (DSP) Embedded-PC im Scheckkartenformat Realzeit-Multitasking-Betriebssysteme
Beispiel Auto:	Schneller Multitasking-Realzeitrechner für Motor und Bremsen Embedded-PC mit Windows CE für d. Instrumentierung (Zukunft) Vernetzung aller Komponenten über den CAN-Bus

2.2 Personal Computer (Intel 80x86)

Der PC wird im Fach **Informationstechnik I** behandelt und für Steuerungszwecke sowohl in **Assembler** als auch in **Pascal** programmiert. Als Ein-Ausgabe-Peripherie werden die Standard-Schnittstellen oder spezielle I/O-Karten verwendet.

Blockdiagramm:



Der PC ist ein **Offenes System**, d.h. das System ist dokumentiert u. nicht geschützt. Fremdhersteller können **Zusatzkomponenten** für den Busanschluss liefern. Dies ist die Grundlage für die grosse Verbreitung des PC und seine Einsatzmöglichkeit in der PDV.

Harvard-Architektur, d.h. Datenspeicher, Programmspeicher und Register befinden sich in unterschiedlich adressierten Speicherbereichen.
Im Gegensatz dazu steht die **v. Neumann Architektur**, bei der sich Datenspeicher, Programmspeicher, Register und Stack im gleichen linearen Adressraum befinden.

Bauarten:

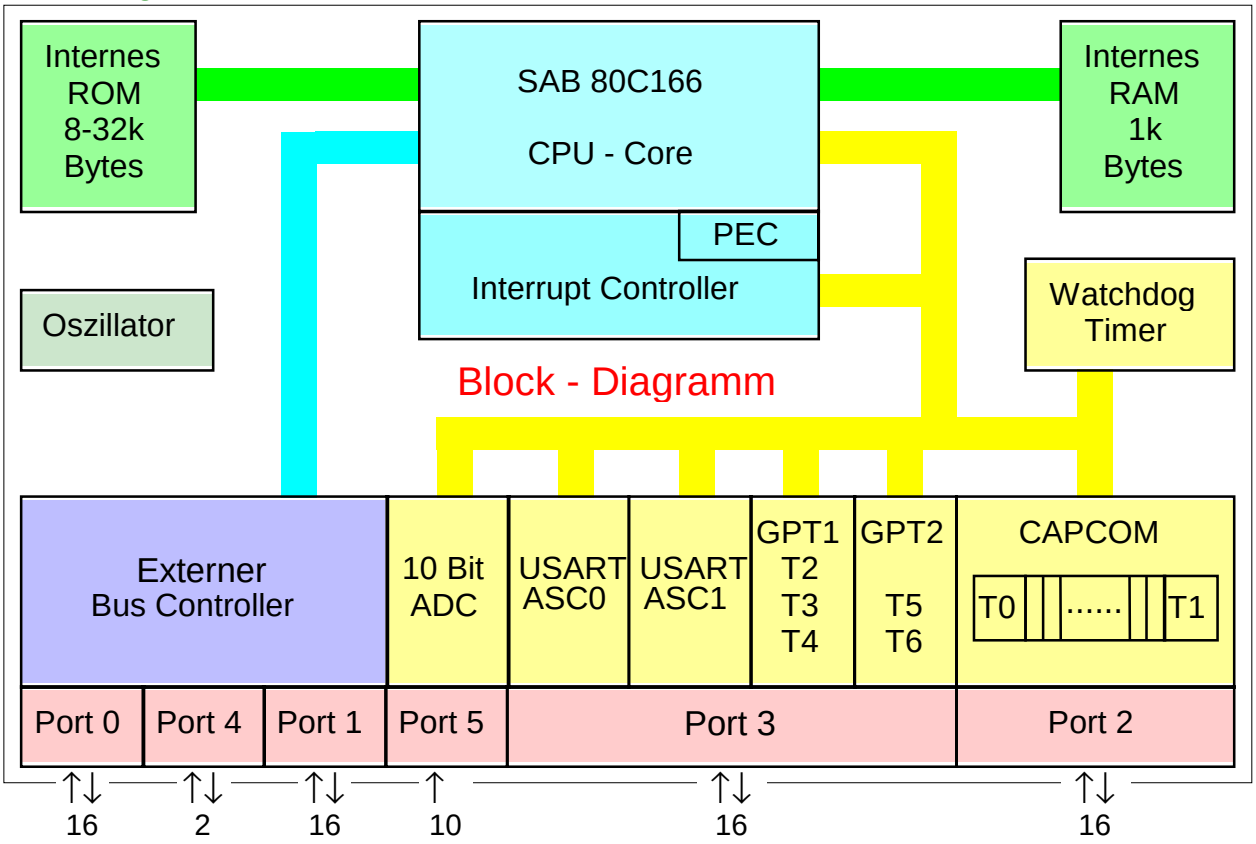
- Normal-PC für Büro- und Laborumgebungen
- IPC (Industrie-PC), meist modular aufgebaut in robuster Einschubtechnik
- CARD-PC von Epson in Scheckkartengrösse für embedded Systeme
- smartPC von DIGITAL-LOGIC in Scheckkartengrösse für embedded Systeme
- STPC Industrial PC-on-Chip von STMicroelectronics für embedded Systeme

Der Vorteil des PC ist, daß praktisch alle Bustechnologien einschließlich Ethernet mit TCP/IP für das Internet standardmäßig verfügbar sind.

2.3 Microcontroller (Infineon 80C16x)

Der 80C166 wird im Fach Mikrocomputertechnik ausführlich behandelt und zur Ansteuerung des Leuchtdiodenfeldes sowohl in Assembler als auch in C programmiert.

Blockdiagramm:



Ports 0, 1 und 4 können zu Adress- und Datenbus umfunktioniert werden.

von Neumann Architektur, d.h. Datenspeicher, Programmspeicher, Register, Special Function Register (SFR) und Stack befinden sich im selben linearen Adressraum.

Belegung von Segment 0:

0,5 kB SFRs (nur intern)	0FFFFh 0FE00h
1 kB RAM (nur intern) einschließlich PEC Pointer, GPR´s und Stack	0FDFFh 0FA00h
30,5 kB externer Speicher	0F9FFh 08000h
31,75 kB internes ROM oder externer Speicher	07FFFh 00100h
256 Byte internes ROM oder externer Speicher, reserviert für 64 Vektor-Sprungbefehle	000FFh 00000h

2.4 Signalprozessoren

2.4.1 Typische Anwendungen für DSPs

Automotive	Consumer	Control
Adaptive ride control Antiskid brakes Digital radios Engine control Global positioning Navigation Vibration analysis Voice commands	Digital radios/TVs Educational toys Music synthesizers Pagers Power tools Radar detectors	Solid-state answering machines Disk drive control Engine control Laser printer control Motor control Robotics control Servo control
General-Purpose	**Graphics / Imaging**	**Industrial**
Adaptive filtering Convolution Correlation Digital filtering Fast Fourier transforms Hilbert transforms Waveform generation Windowing	3-D rotation Animation/digital maps Homomorphic processing Image compression/transmission Image enhancement Pattern recognition Robot vision Workstations	Numeric control Power-line monitoring Robotics Security access
Instrumentation	**Medical**	**Military**
Digital filtering Function generation Pattern matching Phase-locked loops Seismic processing Spectrum analysis Transient analysis	Diagnostic equipment Fetal monitoring Hearing aids Patient monitoring Prosthetics Ultrasound equipment	Image processing Missile guidance Navigation Radar processing Radio frequency modems Secure communications Sonar processing
Telecommunications		**Voice / Speech**
modems Adaptive equalizers ADPCM transcoders Cellular telephones Channel multiplexing Data encryption Digital PBXs Digital speech interpolation DTMF encoding/decoding Echo cancellation	Faxing Line repeaters Pers.comm.systems (PCS) Personal digital ass. (PDA) Speaker phones Spread spectrum communications Video conferencing X.25 packet switching Voice mail	Speaker verification Speech enhancement Speech recognition Speech synthesis Speech vocoding Text-to-speech applications

2.4.2 Die TMS320 Familie und ihre Architektur

Familie	**Die Familienmitglieder TMS320 und ihre Leistungsfähigkeit**

Block-schaltbild	**Arithmetic Unit**	CPU	ALU im Detail	Befehle

Da der Anteil komplexer Operationen bei DSP-Anwendungen hoch ist, werden zur Leistungserhöhung komplexe Befehle durch Hardware implementiert, anstatt, wie sonst üblich, CISC-Architektur durch RISC zu ersetzen.

Eigenschaften:

- Schnelle Rechenoperationen für Realzeitberechnungen (min. 100 mal schneller wie beim PC), bedingt durch leistungsfähige und spezialisierte ALU (schneller Hardware-Multiplizierer).
- Schnelle Fast Fourier Transformation (FFT).
- Mehrere Arithmetikeinheiten können simultan arbeiten, z.B. bei der Summenbildung von Produkten.

Bus-Architektur	**Bus Architektur**

Harvard Architektur mit getrenntem Programmbus und meist 2 Datenbussen. DSPs können damit Befehle und zugehörige Daten (Operanden) simultan aus dem Speicher lesen, wobei Speicheroperanden die Befehlsdauer nicht verlängern. Die Befehle können 2 Speicheroperanden besitzen (statt max. 1 Speicher- und 1 Registeroperand beim normalen Prozessor).

Adress-Arithmetik	**Adressgenerierung**

Hardwarebasierte Datenarray-Verarbeitung mit spezieller Adressgenerierung. Dadurch wird für die Adressberechnung für Speicherzugriffe keine extra Rechenzeit benötigt

Memory-Map	**Memory**

Wegen komlexer (harwarebasierter) Befehle benötigen die Programme wenig Speicher und können deshalb Onchip geladen werden und somit schneller laufen.

Die Speichernutzung ist effizient durch 4 getrennt adressierte Speicherbereiche für Programm, lokale Daten, globale Daten und Input/Output

Das On-Chip Memory beim ’C203 besteht aus:

- DARAM B0 (256 Worte, für Programm- oder Datenmemory)
- DARAM B1 (256 Worte, für Datenmemory)
- DARAM B2 (32 Worte, für Datenmemory)
- I/O (240 Worte, für I/O-Register)

Ext. Memory-Logik	**Zugriff auf den externen Programmspeicher**

Der ’C2xx kann bis zu 64 k Worte externen Programmspeicher adressieren. Für schnellen Speicherzugriff ist es wichtig, externen Speicher mit kleiner Zugriffszeit zu verwenden, um Waitstates zu vermeiden.

Figur 4–1 zeigt ein Anschlussbeispiel für exteren Programmspeicher von 8K×16-bit, wobei zwei 8K×8-bit RAMs verwendet werden.

Multi-Processing	**Multiprocessing**

Beim TMS320C80, der 5 Signalprozessoren enthält, wird der Datenaustausch über einen gemeinsamen RAM-Speicher mit 25 Blöcken organisiert. Die Prozessoren können zwischen diesen Blöcken umgeschaltet werden. Jeder Prozessor arbeitet zu einem bestimmten Zeitpunkt mit seiner eigenen Speicherbank und wird deshalb von andern Prozessoren nicht gestört.

Der DSP TMS320C40 hat 6 lokale Links für Prozessorkopplung. Damit können jeweils zwei der Prozessoren unabhängig von den andern Daten austauschen. Diese lokale Linktechnik wurde aus der Transputertechnik übernommen.

Zusammenfassung der Architektur-Merkmale des DSP verglichen m. Controller

- komplexe Hardwarebefehle
- Hardware-Multiplizierer als Erweiterung der CPU
- hardwarebasierte Komma-Arithmetik
- extra Adressrechenwerk
- 3 Datenbusse statt einem
- 3 Adressbusse statt einem
- 4 getrennt adressierbare Speicherbereiche statt einem
- mehrere Arithmetikeinheiten können simultan arbeiten
- fast keine Peripheriekomponenten - Zugang nur über Bus
- Multiprozessor-Architekturen

2.4.3 Infineon Synthetisierbarer C166 Prozessorkern

Infineon entwickelt mit Synopsis zusammen einen synthetisierbaren C166 16-bit Prozessorkern als lizenzfähiges Intellectual Property Produkt und bietet damit den kunden schnelles **System-on-a-Chip Design** an.

Bestandteile:
VHDL-Beschreibung
Register Transfer Modelle
Prozeßportierbare Synthese-Scripts
kompl. Test- und Verifizierungsumgebung

2.4.4 Infineon TriCore-TC10 32-bit Microcontroller / DSP

Infineon Tricore	**32-bit superskalare Prozessorarchitektur für Realzeit-Multitasking**
Enthält:	32-bit Microcontroller-DSP Core 2-Kanal DMA über Peripheren Control Prozessor (PCP) Interrupt-Responsezeit < 500 ns
Einsatz:	Computer-Perpheriegeräte, KfZ, Industriesteuerung, Kommunikation

2.4.5 ARC anwenderdefinierbare Microcontroller / DSP

Mit anwenderdefinierbaren Mikroprozessoren von ARC ist es möglich, die Funktionen von DSP's und RISC Prozessoren in einem einzigen Core zu vereinigen. Diese Mikroprozessoren bieten gegenüber konventionellen Prozessoren eine **Leistungssteigerung von bis zu 1000%.**

Sie haben eine voll anwenderdefinierbare pipelined 32-bit Harvard Architektur.
Es ist möglich, DSP Erweiterungen aus einer Standardbibliothek und kundenspezifische Erweiterungen zu generieren.

Dies bedeutet, dass der Befehls- und Datenfluss dem Kundenwunsch entsprechend nach einer präzisen anwenderdefinierten Spezifikation gestaltet werden kann. Damit kann der Core jeden **beliebigen Algorithmus** verarbeiten.

Internetadresse: http://www.risccores.com

2.5 Abschließende Übersicht

Für PDV-Lösungen bieten sich somit folgende Möglichkeiten an:

System	Merkmale
PC-Prozessoren 80x86, AMD Kx, PowerPC	32 bit Windows als Betriebssystem Ethernet über Windows Internetzugang über TCP/IP keine Peripherie auf dem Chip
Mikrocontroller z.B. Infineon 80C167CR	4 bis 32 bit stark bei Steuerung schwach in Arithmetik Peripherie auf dem Chip
Signalprozessoren z.B. TI 320x	32 bis 64 bit schwach bei Steuerung stark in Arithmetik keine Peripherie auf dem Chip
Signalcontroller Infineon Tricore	Bei der CPU Vorteile von Microcontrollern und Signalprozessoren vereint keine Peripherie auf dem Chip
Skalierbare Controller C166S-Core und ARC	Auf Kundenwunsch konfigurierbar aufgrund von Standardmodulen und VHDL

3 Prozessrechner Peripherie

3.1 Personal Computer (Intel 80x86)

Der PC bietet einfache Kommunikation mit externen Komponenten über seine parallelen und seriellen Standardschnittstellen.

3.1.1 Parallelschnittstellen LPT 1...2 (Printerports)

Die 8-bit Ports werden durch 8/16-bit-Adressen eines separaten I/O Adressraums identifiziert. Die Adressen werden über die Adressbits 0 ... 15, die Daten über die Datenbits 0 ... 7 übertragen. Zur Unterscheidung von Port- und Speicheradressen dienen die Steuerbusleitungen ¬IOW und ¬IOR. Diese Steuersignale sind mit den Befehlen in und out verknüpft, so dass nur diese Befehle für die Kommunikation mit dem I/O Adressraum zur Verfügung stehen.

Befehl	Datenbits	Port-Adressen	Transfer
in al, port	8 in al	0 ... FFh als Konst.	al ← [port]
in ax, port	16 in ax	0 ... FFh als Konst.	al ← [port], ah ← [port + 1]
in al, dx	8 in al	0 ... FFFFH in dx	al ← [dx]
in ax, dx	16 in ax	0 ... FFFFH in dx	al ← [dx], ah ← [dx + 1]
out port, al	8 in al	0 ... FFh als Konst.	[port] ← al
out port, ax	16 in ax	0 ... FFh als Konst.	[port] ← al, [port + 1] ← ah
out dx, al	8 in al	0 ... FFFFH in dx	[dx] ← al
out dx, ax	16 in ax	0 ... FFFFH in dx	[dx] ← al, [dx + 1] ← ah

Programm-Beispiel:

```
mov   dx, 378h        Port-Adr. 378h
mov   ax, 1122h       ah = 11h,          al = 22h
out   dx, ax          Port 379 ← 11h,    Port 378 ← 22h
```

Standardadressen:

LPT1:	378h ... 37Bh	IRQ7
LPT2:	278h ... 27Bh	

Parallelschnittstellen-Baustein 8255 für parallelen Transfer:

- Anschluss an die unteren 8-bit des Datenbusses des PC
- 3 Parallelports in / out mit je 8 bit werden bereitgestellt

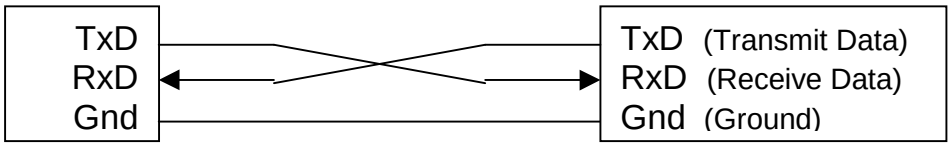
Achtung: Nicht in Systemports des PC schreiben ! !

3.1.2 Serielle Schnittstellen Com 1...4 (RS 232 bzw. V24)

Die RS 232-Schnittstelle (RS = Recommended Standard) ist international genormt und nur für eine Punkt zu Punkt Verbindung geeignet.

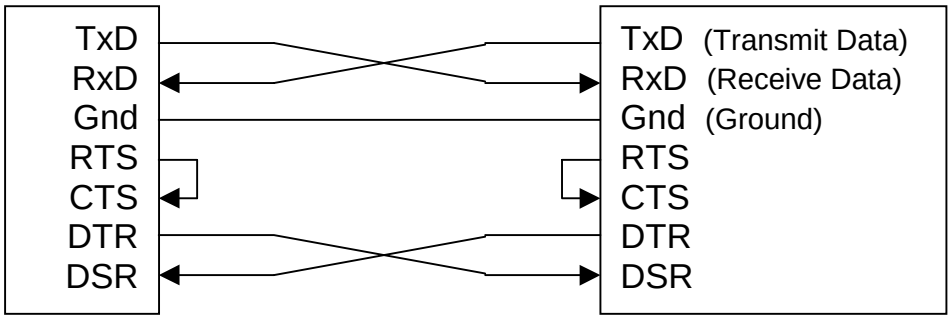
Pegeldefinition gegen Masse:	logisch 1: nicht definiert: logisch 0:	-15V .... -3V -3V .... +3V +3V .... +15 V
Leitungslänge (Richtwert):	900 m bei Übertragungsrate 1200 Baud (Bit/s) 50 m bei Übertragungsrate 19200 Baud	

Die Schnittstelle ist **vollduplexfähig** (kann gleichzeitig senden und empfangen).
Sie kann mit bis zu 7 Verbindungen in verschiedenen Übertragungsverfahren betrieben werden.
Die **Minimalkonfiguration** ist eine Dreidraht-Verbindung:



Hier muss die Software die Sicherheit der Datenübertragung gewährleisten.

Die Schnittstelle kann auch im **Handshaking-Betrieb** arbeiten und hierbei die Sicherheit der Datenübertragung überwachen.
Dazu sind je 2 weitere Melde- und Steuerleitungen nötig:



Standardadressen:

COM1:	3F8h ... 3FFh	IRQ4
COM2:	2F8h ... 2FFh	IRQ3

MUART-Baustein 8256 für parallelen und seriellen Transfer:

- Anschluss an den 8-bit-Datenbus
- 2 Parallelports in / out mit je 8 bit
- 1 Asynchronous Communications Interface
- 1 Baudratengenerator
- 5 programmierbare Timer / Counter
- 1 Interruptcontroller mit 8 Prioritätsebenen
- 1 programmierbare System Clock

Weiter gibt es Standardbausteine in grosser Zahl

3.1.3 Spezielle Peripherie Einsteckkarten

Für Mess- und Steuerungsaufgaben gibt es von vielen namhaften Firmen eine Vielzahl hochwertiger **PC-Einsteckkarten**. Auch der Anschluss an genormte Busse (z.B. USB) wird angeboten. Die zugehörige Software erlaubt eine komfortable Messwertverarbeitung oder Erledigung von Steuerungsaufgaben, wie z.B. den Betrieb von Schrittmotoren bei einfacher Programmierung.

3.1.4 Universal Serial Bus (USB)

Begriffe	Präsentation	Infos	Spezifikation	Entwicklerforum

Der USB entstand aus der Notwendigkeit heraus, den Anschluss von Peripheriegeräten an den PC zu vereinfachen, zu verbilligen, zu verbessern und die veralteten COM- und LPT-Schnittstellen endlich abzulösen. Serienmässig ist USB auf der Hauptplatine und im BIOS von PC's seit 1995 vorhanden. USB wird ab **Windows 98** voll unterstützt. Seither ist das Geräteangebot mit USB gewachsen und der Umschwung in vollem Gang. Ältere PC's mit PCI-Bus sind mit **PCI/USB Karten** nachrüstbar.

Gründungsmitglieder des **USB-Implementer-Forums**:

Compaq, DEC, IBM, Intel, Microsoft, NEC und Northern Telecom

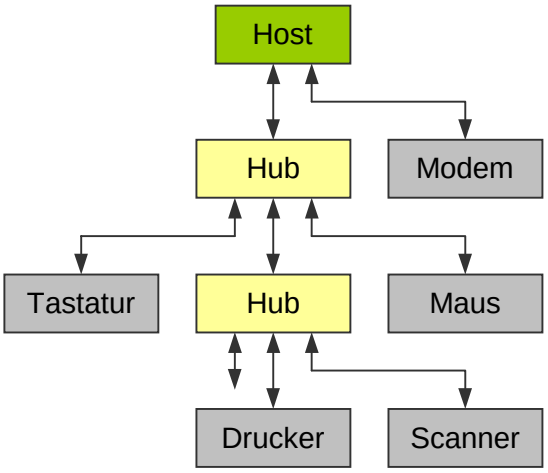
3.1.4.1 Eigenschaften

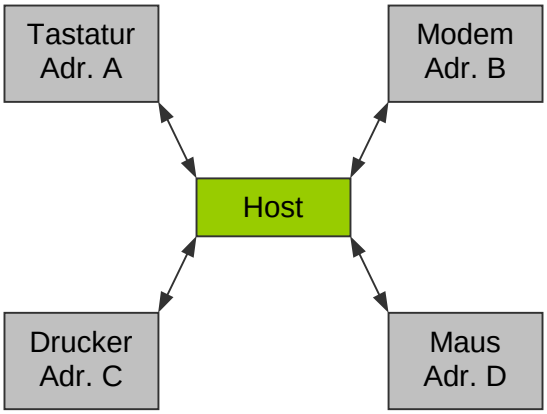
- **127 Geräte** maximal anschließbar (einschließlich Hubs)
- **Übertragungsgeschwindigkeit:**
  Low-Speed: 1,5 Mbit/s für Maus, Tastatur usw. mit einfachem Kabel
  Full-Speed: 12 Mbit/s für komplexe Geräte mit abgeschirmtem Kabel
  High-Speed: bis 200 (oder 500?) Mbit/s in Planung
- **Kabellänge 5 m** maximal
- **4-poliger Stecker** für Geräte wie Tastatur, Maus, Drucker, Fax, Modem mit Vbus (+5V, max. 500 mA), GND, D+ und D-
- **Stromversorgung** der Geräte kann **über den USB** erfolgen (+5V, max. 500 mA)
- USB-Geräte brauchen **keine eigenen Ressourcen** wie Speicher, Interrupt oder I/O. Sie gehen nach 3 ms Inaktivität in den **Suspend-Modus** (max. 2,5 mA Stromverbr.)
- **Hot Plug&Play**-Fähigkeit (Geräte An-und Abschaltung bei laufendem Computer)
- Mechanismen zur **Fehlererkennung** und Transferwiederholung sind vorhanden
- **Transferarten:**
  Control-Transfer für Requests vom Host an ein USB-Gerät bei der Initialisierung
  Interrupt-Transfer, ausgelöst d. Pollingverf. als Interruptersatz (z.B. Tastatur)
  Bulk-Transfer für große Datenmengen ohne Realzeit- und Kontinuitätsbed.
  Isochronous-Transfer für Daten Realzeitbedingung (z.B. Audiodaten)
- USB bei **Controllern** zum Bau von Peripheriegeräten verfügbar.
- Es gibt Module zum Koppeln an andere Bussysteme, z.B. **CAN Bus**
- Es gibt ein Verbindungskit zum Koppeln von **2 PCs** (Advance Peripherals)

3.1.4.2 Hardware-Architektur

USB Hardware Komponenten	
Host Controller	steuert alle Aktivitäten auf dem Bus, serialisiert die Daten und übergibt **Datenpakete** an den Root Hub (u. umgekehrt). Der Host ist der einzige Busmaster.
Root Hub	stellt die Anschlussports für die Geräte bereit und sendet die Datenpakete über den Bus (und umgekehrt). Er besteht aus **Hubcontroller und Repeater**.
Hub	stellt zusätzliche 2-4 Ports bereit. Es werden max. 6 über Hubs kaskadierbare **Kabelsegmente** unterstützt.
Endgeräte (Functions)	Sie enthalten **Deskriptoren** mit Eigenschaften, Attributen und Initialisierung des Geräts.
Kabel	Flacher Stecker A an der Hubseite, quadratischer Stecker an der Geräteseite. Bei Full Speed verdrillte Signaladern und Abschirmung, bei Low Speed nicht (billig u. flexibel).

Physikalische Struktur	Logische Struktur
Baumstruktur. Die Verzweigungsknoten werden durch Hubs (Mehrfachverteiler) gebildet.	Sternstruktur. Die Endgeräte sind sternförmig an den Host angeschlossen.





Mögliche Transferwege:

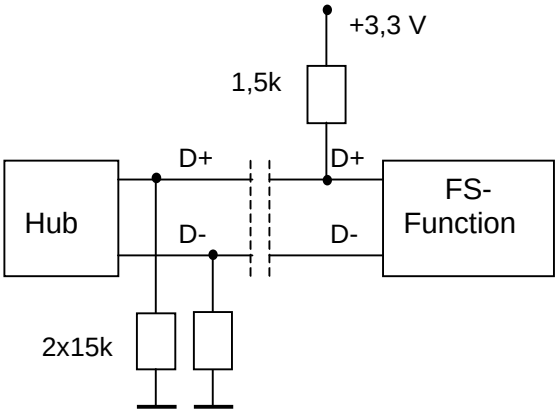
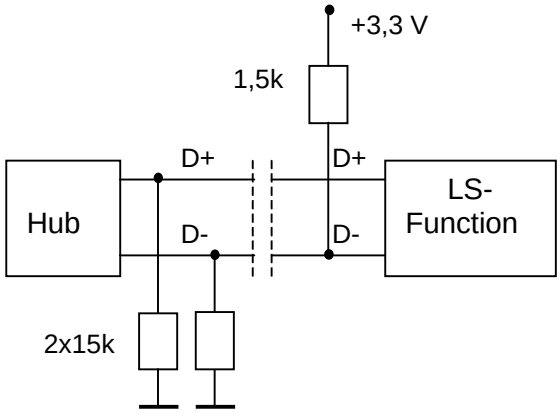
Downstream:	Host	→ Hub	→	Function
Upstream:	Function	→ Hub	→	Host
Nicht möglich:	Function	→ Hub	→	Function

Die Bussignale werden als **Differenzspannungen** auf den D-Leitungen übertragen.

Differentielle Buszustände:

1: (D+) - (D-) > 200 mV

0: (D-) - (D+) > 200 mV

Connect- und Disconnect-Erkennung	
Fullspeed (statisch log. 1)	**Lowspeed** (statisch log. 0)
	

3.1.4.3 Software-Architektur

Der USB muß durch das **Betriebssystem** unterstützt werden.
Dies ist z.B. ab **Windows 98 und 2000 (nicht NT 4!)** der Fall.

USB Software Komponenten	
Gerätetreiber	Erzeugt Anfragen (**Requests**) an den USB-Bustreiber. Dazu versendet dieser I/O Request-**Packets**, die einen Transfer von und zu USB-Geräten initiiert.
Bustreiber	Er kennt die Eigenschaften der USB-Geräte aus der Analyse der Deskriptoren und erzeugt nach Erhalt von Requests passende **Transaktionen**, die in einen **Frame** von 1 ms passen.
Hostcontroller-Treiber	Er organisiert die zeitl. Abfolge der Transaktionen (**Scheduling**) über Transaktionslisten und löst diese über den Root-Hub aus.
Kommunikations-modell	Ein Transfer wird von einem USB-Gerät initiiert, indem es bei der USB-Software einen Transfer anfordert. Der Gerätetreiber stellt einen **Pufferspeicher** bereit, in den d. Übertragungsdaten abgelegt werden. Der Datentransfer erfolgt über **Kommuni-kationspipes**, die bei der Konfiguration aufgebaut werden.

3.1.5 Firewire Bus (IEEE 1394, bei Sony iLink)

1999 sollen 10% der neuen **PC's** diese Schnittstelle erhalten haben.
Sie wird ab **Windows 98** unterstützt.
Von **Apple** für Multimedia konzipiert, hält aber auch schon in der Industrie Einzug.
200 Mbit/s (Low Speed), High Speed bis 800 Mbit/s in Planung.
Kabellängen bis über 100 m konzipiert.
Miniatur-Steckverb. für Geräte wie Festplatten, digitale Kameras u. Camcorder.
Hot Plug&Play-Fähigkeit (Geräte An- und Abschaltung bei laufendem Computer).
1394-Bausteine zum Bau von Peripheriegeräten verfügbar (z.B.v.Texas Instr.).

3.1.6 Bluetooth (Spezifikation 1.0 / Sommer 1999)

Zur Funkvernetzung mobiler Geräte im ISM Band mit 2,4 GHz.
Standard durch Bluetooth Interest Group (Ericsson, IBM, Intel, Nokia, Toshiba)

3.2 Microcontroller

Microcontroller zeichnen sich dadurch aus, dass sie eine mehr oder weniger **umfangreiche Peripherie Onchip** besitzen (siehe 80C166).
Bei der Projektierung eines Embedded Projekts ist es sinnvoll, den Controller so auszuwählen, dass die Onchip Peripherie ausreicht.
Jede externe Peripherie erfordert **8 - 24 Portleitungen** als Daten- und Adressbus und **reduziert die zugehörige Rechenleistung** um ca. den Faktor 2.

Standardperipherie on Chip:

- Bidirektionale Portleitungen
- CAPCOM (Capture Compare)
- Timer / Counter
- Watchdog Timer
- CAN Modul (80C167CR)
- ADC (Analog Digital Converter)
- DAC (Digital Analog Converter)
- Serielle Schnittstellen, auch High-Speed (80C167)
- Es werden auch Typen mit **USB** angeboten, z.B. SAB 80C540U / C541U

3.3 Signalprozessoren

- Signalprozessoren besitzen **fast keine Onchip Peripherie** und benötigen immer externe Peripheriebausteine, die es in grosser Zahl gibt.
- Wegen der komplexen Software werden auch immer **externe Speicher** benötigt, auf die ein schneller Zugriff möglich ist (Prozessor, nicht Controller!).
- Standard sind 1 - 2 serielle Schnittstellen.
- Anschaltbeispiel von **Schnittstellenbausteinen:**

Ext. I/O-Logik **Zugriff auf den externen I/O Adressraum (Schaltungsbeispiel)**
Auf alle I/O Worte (externe I/O Ports und on-chip I/O Register) wird mit den Instruktionen IN und OUT zugegriffen. Zugriffe auf externe parallele I/O Ports werden über die Adress- und Datenbusse für Programm- und Datenspeicher-Zugriffe gemultiplexed. Diese Port-Zugriffe werden von den andern durch das Signal IS = low unterschieden. Der Datenbus ist 16 bit breit. Wird jedoch 8-bit Peripherie verwendet, können entweder die hohen oder niedrigen 8 Leitungen des Datenbusses mit der Anwendung verbunden werden. Figur 4–13 zeigt ein Schaltungs-Beispiel für den Anschluss externer I/O-Plätze (je 8 Ein- und Ausgangsbits).

4 Realzeit - Betriebsarten

4.1 Interruptbetrieb

4.1.1 Prinzip

Im Wahlfach MCT wird die Implementierung von **Interrupts beim 80C166** in Theorie und Labor ausführlich behandelt. Deshalb soll hier nur auf grundsätzliche Aspekte eingegangen werden.

Das Interruptsystem versetzt die CPU in die Lage, auf **zeitkritische Ereignisse (Alarme)** mit geringstmöglicher Zeitverzögerung zu reagieren. Dies ist eine Standardmethode in der Prozessabarbeitung.

Dazu wird das laufende **Programm zur Interrupt Bearbeitung unterbrochen** und danach an der Unterbrechungsstelle mit altem Status fortgesetzt.

Zur Prozessoptimierung gibt es vielfältige Steuermöglichkeiten:

- Die **Maskierung** ermöglicht es, Interupts im Prozess einzeln oder global dynamisch zuzulassen (enable) oder zu verhindern (disable)
- Damit kann die **Verschachtelung** von Interrupts und ihre Tiefe gesteuert werden
- Das **Prioritätssystem** regelt, welche Interrupts welche anderen unterbrechen dürfen

Alarm-Quellen für die Auslösung von Interrupts sind:

- Die **CPU** zur Fehlerbehandlung (Exceptions oder Traps, nicht maskierbar)
- Signale von **externen Einrichtungen** oder Geräten wie Sensoren (maskierbar)
- **Int. periphere Einrichtungen** des Controllers wie Timer und ADC (maskierbar)

4.1.2 Interrupt - Ablauf

Wenn ein **Alarm** eintrifft, setzt dieser das Interrupt Request Bit der betr. Quelle. Dieses bleibt bis zur Bedienung des Interrupts gesetzt und wird bei Eintritt in die Interrupt-Routine zurückgesetzt (Acknowledge). Alarme werden also bis zu ihrer u.U. verzögerten Bearbeitung gespeichert.

Wenn ein Interrupt-Request vorliegt, das Interrupt nicht maskiert und die Priorität hinreichend hoch ist:

- wird der **laufende Befehl** zu Ende bearbeitet,
- werden Rückkehradresse und Status auf den **Stack** gerettet,
- wird das globale **Enable Bit** gelöscht, weitere Interrupts also verhindert,
- wird der Sprung zum zugeordneten **Interrupt-Vektor** ausgeführt,
- wird das **Interruptprogramm abgearbeitet,**
- wird bei Erreichen von **RETI** ins Hauptprogr. mit **altem Status** zurückgekehrt.

Bei Eintritt in die Interrupt Routine übernimmt die CPU die Priorität des ausgeführten Interrupts. Am Ende der Routine restauriert **RETI** den vorigen Status, also die vorige Priorität.

Nur Interrupts mit höherer Priorität als die CPU können diese unterbrechen.
Bei Reset hat die CPU Priorität 0, also werden **Interrupts mit Priorität 0 nicht angenommen**.

4.2 Direct Memory Access (DMA)

Sowohl beim Abruf- als auch beim Anforderungsbetrieb (Polling- und Interruptbetrieb) werden die Daten immer mittels der CPU zwischen einem Gerät und dem Speicher ausgetauscht. Für jedes Byte sind eine Reihe von CPU-Befehlen in einer Schleife nötig, was zu einem **geringen Datendurchsatz** führt und die CPU belastet. Das bedeutet z.B., dass die Wandlungsrate eines AD-Wandlers unter Umständen nicht ausgenutzt werden kann.

Beim direkten Speicherzugriff **(DMA)** werden die Daten **ohne CPU-Beteiligung** direkt zwischen Speicher und Gerät ausgetauscht. Ein DMA-Controller bzw. eine Logikschaltung auf dem Chip muss die Koordinierung des Transfers mit der CPU wegen des Buszugriffs abstimmen. Als einfaches **Beispiel** kann der PEC-Transfer beim 80C166 betrachtet werden, bei dem die PEC-Steuerung mit der Interruptlogik kombiniert ist.

4.2.1 PEC-Transfer beim 80C166

Bei vielen zeitkritischen interruptgesteuerten Vorgängen wie dem Einlesen von Messwerten ist lediglich ein **Datentransfer** in eine Tabelle auszuführen.

Der PEC Transfer, eine Art DMA-Betrieb, erledigt dies **in einem Maschinenzyklus**. Der Programmdurchlauf wird kurz unterbrochen und ein Transferbefehl, dessen Quell- und Zielpointer vorprogrammierbar sind, zum Datentransport über den Bus eingeschoben. Dadurch wird der Zeitverzug beim Abarbeiten einer Interrupt Routine eingespart.

Die Adressen (**Pointer**) von Quelle und Ziel werden in einer fest adressierten Tabelle im RAM gespeichert. Diese wird bei Reset nicht gelöscht.

DSTP7	FDFEH	PECC7	oberes Ende des RAM
SRCP7	FEFCH	PECC7	
insgesamt je 8 PEC Source und Destination Pointer	insgesamt32 Bytes	zugeordnete Steuer-Register	liegt im oberen Teil des GPR Bereichs, bei hohem CP besteht Kollisionsgefahr!
DSTP0	FDE2H	PECC0	
SRCP0	FDE0H	PECC0	Grenze zum Registerbereich

Die **Steuerregister PECC0 ... PECC7** enthalten folgende Steuerparameter:

- Autoinkrementmodus der Pointer (keiner, Ziel- oder Quellpointer-Increment)
- Word- oder Byte-Transfer der Daten
- Länge des übertragenen Datenblocks
- PEC-Transfer ohne Interrupt oder
- Interrupt mit PEC-Transfer oder
- Interrupt ohne PEC-Transfer
- Interrupt am Datenblockende, z.B. zur Datenauswertung

4.2.2 Real Time Data Exchange (RTDX) von Texas Instruments

Die Signalprozessoren von Texas Instruments besitzen als Testanschlüsse den **JTAG** Datenpfad. Dieser ermöglicht dem Entwickler dauernden bidirektionalen Realzeit-Datenaustausch zwischen DSP und externen Testeinrichtungen und Debuggern, ohne das Programm anzuhalten und ohne die CPU-Zeiten nennenswert zu beeinflussen.

Diese universellen Anschlüsse können auch in der Anwendung benutzt werden. Ohne zusätzliche Kosten bieten sie die Möglichkeit, Realzeit-Daten ähnlich wie bei DMA direkt aus dem Speicher auszulesen.

Zusammenfassung der Eigenschaften von RTDX:

- Verwendung des universellen JTAG Datenpfads
- fortlaufender bidirektionaler Datenaustausch ohne Programmunterbrechung
- Realzeitbetrieb mit minimaler Störung der Anwendung
- Daten können mit OLE-fähigen Programmen, z.B. Excel, visualisiert werden
- leicht zu programmieren
- keine Zusatzkosten bei Einsatz von DSP's von Texas Instruments

JTAG	**JTAG- Anschlüsse bei DSPs von Texas Instruments**

4.3 Realzeit-Multitasking

Bei einem **Einprogrammsystem** ist immer nur ein Programm im Speicher geladen und wird von der CPU abgearbeitet. Das Programm hat alle Komponenten des Rechners, seine Betriebsmittel, ständig zur Verfügung.

Bei einem **Mehrprogrammsystem** sind mehrere Programme vorhanden, die miteinander konkurrieren, wenn sie gleichzeitig laufen sollen. Jedes Programm benötigt bestimmte Hard- und Softwarebetriebsmittel, z.B. CPU, Speicher, Dateien, Registerinhalte.

Das Programm bildet zusammen mit seinen notwendigen Betriebsmitteln **(Kontext)** eine **Task.** Da manche Betriebsmittel des Rechners, z.B. die CPU, nicht mehreren Tasks gleichzeitig zugewiesen werden können, werden diese von den Tasks im **Zeitmultiplex (time sharing)** benutzt. Je nach Notwendigkeit sind die Zeitscheiben der Tasks mehr oder weniger groß.

Werden die Programme in **fester Reihenfolge** abgearbeitet, nennt man diese Methode '**Round Robin'**.

Besser ist, die Programme nach ihren **Anforderungen und Prioritäten** so abzuarbeiten, daß die Betriebsmittel gut genutzt werden und wo nötig harte Realzeit erreicht wird.

Im einfachsten Fall werden, durch Timerinterrupts gesteuert, die einzelnen Tasks als **Interruptroutinen** aufgerufen. Dann muss der Programmierer die Verwaltung der Tasks programmieren, was fehlerträchtig ist.

Wegen der komplexen Taskverwaltung und um Sicherheitsstandards einzuhalten, setzen sich zunehmend hochwertige Betriebssysteme durch. Stark im Trend sind **Realzeit-Multitasking-Betriebssysteme** für Controller mit komfortablen Entwicklungs- und Debug-Tools. Einige werden im nächsten Kapitel besprochen.

5 Prozessrechner Software

5.1 Programmiersprachen

5.1.1 Assembler

Die **Assemblersprache** wurde im Fach Informationstechnik I bekanntgemacht und geübt. Ihre Kenntnis gehört zu den technischen Programmiergrundlagen, vor allem beim **Debuggen** von Programmen. Die Hochsprachencompiler für Microcontroller liefern als Zwischenergebnis ein Assemblerlisting, welches in den Maschinencode assembliert wird. Dadurch ist eine **Kontrolle des Compilerergebnisses** möglich.

5.1.2 C(PP)

C und C++ (objektorientiert) sind im technischen Bereich die meistbenutzten Hochsprachen. **Unix und Windows** sind darin geschrieben. Deshalb erlaubt diese Sprache auch direkten Zugang zur **WinAPI** (Windows Application Programming Interfache) von Microsoft.
C wird im **Wahlfach Mikrocomputertechnik** neben Assembler zur Programmierung der Übungsaufgaben mit dem Keil-Compiler benutzt.
C und C++ werden im **C-Blockkurs** und in **anderen Wahlfächern** gelehrt.
Diese Sprachen werden hier nicht weiter vertieft.

5.1.3 JAVA

- Anfang der 90er Jahre zur **Vernetzung** von Consumergeräten von Sun entwickelt.
- Die Java-Syntax entspricht einer bereinigten Version von **C++.**
- Java ist **objektorientiert** und deshalb weitgehend statt C++ einsetzbar.
- Java ist einfacher als C++ und deshalb leichter und fehlerfreier anzuwenden.
- Die Java-Laufzeitbibliothek macht Java-Anwendungen **plattformunabhängig.** Identischer Quelltext kann unter Windows, Solaris, Unix usw. verwendet werden. Deshalb wird Java ab 1994 für die Internetprogrammierung eingesetzt, wo Applets auf unbekannten Clients laufen sollen.

5.1.3.1 Wichtigste Unterschiede zu C++

- Die Möglichkeiten der manuellen **Speicherallokierung** und Deallokierung wurden abgeschafft. Es gibt keinen Ärger mit Speicherreservierungsproblemen mehr!
- Die **Zeigerarithmetik** wurde abgeschafft. Die Gefahr von Speicherüberschreibungen ist deshalb stark reduziert.
- Echte **Array-Objekte** mit eigenen Bearbeitungsmethoden wurden eingeführt.
- Mehrfachvererbungen gibt es nicht mehr. Dadurch wird die Hierarchie vereinfacht (bei Mehrfachvererbung kann eine Klasse mehrere Superklassen haben)
- Es werden **keine Header-Dateien** (*.h) verwendet.
- **Kein Überladen** von Operatoren möglich.
- Da Java plattformunabhängig ist, sind naturgemäß keine Hardwarezugriffe möglich. Diese müssen durch **native Treiber** lokal bereitgestellt werden.
- Der Bedarf an Speicherplatz ist bei Java viel geringer, weil **zur Laufzeit interpretiert** bzw. am Zielort vor Programmstart compiliert wird. Daher sind Java Dateien klein, was wichtig ist beim Laden über das Internet.

5.1.3.2 Variablentypen

Jede **Deklaration einer Variablen** muß deren Typ angeben.
Dieser legt den Wertebereich der Variablen fest und bestimmt die Operationen, die mit ihr ausgeführt werden können.

5.1.3.2.1 Ganzzahlen

Typ	Bereich	Länge	Init.-Wert
byte	-128 ... 127	1 Byte	0
short	-32768 ... 32767	2 Bytes	0
int	-2147483648 ... 2147483647	4 Bytes	0
long	-2^{63} ... 2^{63} -1	8 Bytes	0

Beispiel: int Guthaben = 3865;

5.1.3.2.2 Fliesskommazahlen (IEEE 754)

Typ	Bereich	Länge	Init-Wert
float	+-1.40239846 E-45 ... +-3.40282347 E+38	4 Bytes	0.0
double	+-4.94065645841246544 E-324 ... +-1.79769313486231570 E+308	8 Bytes	0.0

Beispiele: double a = 3865.87 (Festkomma), double b = 3,8E-3 (Gleitkomma)
double c = 3865.87f (Gleitkomma), double d = 2f (Gleitkomma)

5.1.3.2.3 Char

Typ	Bereich	Länge	Init-Wert
char (Unicode character)	\u0000 ... \uFFFF	2 Bytes	\u0000

Beispiel: char Zeichen = '?'; oder Zeichen = \u003F; (3Fh ist der ASCII-Code)

5.1.3.2.4 Boolean

Variable vom Typ **boolean** können die Werte false oder true annehmen.
Sie werden für logische Prüfungen verwendet. Initialisierungswert: false
Beispiel: boolean Gutwetter = true ;

5.1.3.2.5 String

Strings oder Zeichenketten werden in doppelte Anführungszeichen geschrieben.
Beispiel: String Gruesse = "Hallo, Kumpel !";

5.1.3.2.6 Array

Arrays haben eine festgelegte Anzahl von Komponenten eines einzigen Typs.
Sie können ein- oder mehrdimensional sein. Arrays sind in Java Objekte.

Deklarationsbeispiel: Bitmuster für die Spaltenwerte von 2 Zeichen am LED-Feld:

```
byte Feld[]   = {   0xFF,0x88,0x88,0x88,0x88,0x00,      // F
                    0xFF,0x08,0x08,0x08,0xFF        }   // H
```

Zugriffsbeispiel für 3. Spalte von ´F´:

```
byte Spalte = Feld[2];
```

In Java kann die Größe eines Arrays auch noch zur Laufzeit festgelegt werden!

5.1.3.3 Ausdrücke und Operatoren mit Rangfolge

Ausdrücke bestehen aus Operatoren und Operanden.
Binäre Operatoren verknüpfen 2 Operanden (Beispiel: Spalte + 1).
Unitäre Operatoren arbeiten mit einem Operanden (Beispiel: not Wert).

Art	Operatoren	Rang	bei gleichem Rang
	[ ], . , ( ) Funktionsaufruf	0	links nach rechts
unitär	!, ~, ++, --, +, - (Vorz.), (type)	1	rechts nach links
Mult. + Div.	*, /, % (Rest)	2	links nach rechts
Addition + Subtr.	+, -	3	links nach rechts
Shift	<<, >>, >>>	4	links nach rechts
Vergleich	<, >, <=, >=, instanceof	5	links nach rechts
gleich / ungleich	==, !=	6	links nach rechts
bitweise AND	&	7	links nach rechts
bitweise XOR	^	8	links nach rechts
bitweise OR	|	9	links nach rechts
logisch AND	&&	10	links nach rechts
logisch OR	||	11	links nach rechts
bedingt	?:	12	rechts nach links
Zuweisung	=, +=, -=, *=, /=, %=, <<=, >>=, >>>=, ^=, |=, &=	13	rechts nach links

5.1.3.4 Kontrollstrukturen

5.1.3.4.1 while (...) { ... }

Struktogramm:

Wiederholen, solange Ziffer < 10
Ziffer ausgeben an Konsole
Ziffer um 1 erhöhen

Java-Programm:

```
while (Ziffer < 10)
{                                   // Beginn der Verbundanweisung
  System.out.println(Ziffer);       // Ausgabe der Ziffer an Konsole
  Ziffer = Ziffer + 1;              // nächste Ziffer
}                                   // Ende der Verbundanweisung
```

Die Anfangsziffer muß vor Schleifenbeginn gesetzt werden.
Die Ziffernänderung wird innerhalb der Schleife vorgenommen.
Die Bewertung des Kontrollausdrucks erfolgt vor dem Schleifendurchlauf.
Die Schleife wird deshalb mindestens nullmal durchlaufen.

5.1.3.4.2 do { ... } while (...)

Struktogramm:

	Ziffer ausgeben an Konsole
	Ziffer um 1 erhöhen
Wiederholen, solange die Ziffer <= 9 ist	

Java-Programm:

```
do
{                          // Beginn der Verbundanweisung
  System.out.println(Ziffer); // Ausgabe Ziffer
  ++Ziffer;                   // nächste Ziffer
}                          // Ende der Verbundanweisung
while (Ziffer <= 9);
```

Die Anfangsziffer muß vor Schleifenbeginn gesetzt werden.
Die Ziffernänderung wird innerhalb der Schleife vorgenommen.
Die Bewertung des Kontrollausdrucks erfolgt nach dem Schleifendurchlauf.
Die Schleife wird deshalb mindestens einmal durchlaufen.

5.1.3.4.3 for (... ; ... ; ...) { ... }

Struktogramm:

Wiederholen für die Ziffern 0 bis 9	
	Ziffer ausgeben an Konsole

Java-Programm:

```
for (Ziffer = 0; Ziffer <= 9; Ziffer++)
  System.out.println(Ziffer); // Ausgabe Ziffer
```

Die Anfangsziffer wird durch die Schleifenanweisung gesetzt.
Die Ziffernänderung wird nach Durchlauf der Schleife automatisch vorgenommen.
Die Bewertung des Kontrollausdrucks erfolgt vor dem Schleifendurchlauf.
Die Schleife wird deshalb mindestens nullmal durchlaufen.

5.1.3.4.4 Anweisung if (...) { ... } else { ... }

Struktogramm:

Bits = 60H ?	
ja	**nein**
Flags.0 ← 1	Flags.0 ← 0

Java-Programm:

```
if (Bits == 0x60)          // Vergleich
        Flags |= 1;        // 1. Zweig, ´True´
    else Flags &= 0xfe;    // 2. Zweig, ´False´
```

5.1.3.4.5 Anweisung switch (...) { case ... : ... ; ... ; break; usw. }

Struktogramm:

Low Nibble von Flags null setzen			
Bits = ?			
60h	**40h**	**28h**	**Rest**
Flags.0 ← 1	Flags.1 ← 1	Flags.2 ← 1	Flags.3 ← 1

Java-Programm:

```
Flags = Flags & 0xf0;                  // Low Nibble von Flags null setzen
switch (Bits) {                        // Selektorbeginn
    case 0x60 :   Flags |= 1;   break;   // 1. Zweig
    case 0x40 :   Flags |= 2;   break;   // 2. Zweig
    case 0x28 :   Flags |= 4;   break;   // 3. Zweig
    default   :   Flags |= 8;   break;   // 4. Zweig
}                                      // Selektorende
```

5.1.3.5 OOP (objektorientierte Programmierung) mit Java

- Bei der objektorientierten Programmierung wird ein Programm aus Objekten zusammengesetzt (wie ein PKW aus Zulieferteilen), denen jeweils die Erledigung von miteinander **in Beziehung stehender Aufgaben** übertragen wird.
- Ein Objekt kann dazu ggf. auch auf **andere Objekte** zugreifen.
- Ein Objekt soll **nie** die inneren Daten eines andern Objekts verändern können.

5.1.3.5.1 Java Objekte

- Die drei Schlüsseleigenschaften eines Objektes sind:
  das **Verhalten**, definiert durch die Nachrichten, die es akzeptiert,
  der **Zustand**, definiert durch gespeicherte Daten,
  die **Identität**, durch die sich Objekte mit gleichartigen Aufgaben unterscheiden.
- Die Eigenschaften von Objekten werden in **Klassen** zusammengefaßt.
- Wird ein Objekt einer Klasse erzeugt, spricht man vom Bilden einer **Instanz** der Klasse: *MetrSchraube M4 = new MetrSchraube();*
- Das **Schlüsselwort** new reserviert Speicherplatz für die **Instanz M4**. Dieser wird vom Garbage Collector autom. freigegeben, wenn M4 nicht mehr benutzt wird.

5.1.3.5.2 Java Klassen

- Die Eigenschaften von Objekten werden in **Klassen** beschrieben.
- **Klassen sind Datentypen**, die vom Anwender definiert werden oder vorgefertigt aus Bibliotheken stammen.
- Eine Klassendefinition beschreibt eine **Datenform** und deren Benutzung.
- Ein Objekt ist ein Element, das gemäß der Datenformbeschreibung erzeugt wird.
- Eine Klasse steht in Bezug zu einem Objekt wie ein Typ (z.B. int) zu einer Variablen.
- Eine Klasse ist wie eine **Schablone**, mit der das Objekt tatsächlich erzeugt wird. Daten (Instanzvariable und Instanzfelder) und Funktionen (Methoden) sind **Klassenelemente**.

- Java stellt vorgefertigte **Bibliotheksklassen** für häufig wiederkehrende Objekte zur Verfügung, die das einfache Bilden von Objekten ermöglichen. Solche Objekte sind z.B. Buttons, Zeichenflächen (Canvas), Menüs, Labels, Textfelder usw. Weitere Bibliotheken bieten die Compiler an.
- **Abgeleitete Klassen** (Subklassen) erben die Klassenelemente ihrer Superklasse.
- **Hierarchie** der wichtigsten Klassen:
  Object ⇒ Component ⇒ Container ⇒ Window ⇒ Frame

5.1.3.5.3 Beispiel

In einer Klasse "MetrSchraube" werden die grundlegenden Gewindeeigenschaften beschrieben. Diese **vererbt** sie an eine davon **abgeleitete** Klasse "InbusSchraube", welche nur noch die für den Kopf erforderlichen Zusatzeigenschaften enthält.

```
class MetrSchraube                    // Definition der Klasse MetrSchraube
{
        char   Bezeichnung;           // Klassenvariablen (hier privat),
        int    Steigung;              //       keine Veränderung von außen
        ........                      // weitere Klassenvariablen

        public MetrSchraube()         // Klassenmethode),
        {                             // public, kann von außen aufgerufen werden
                .......               // Java Programmzeilen
        }
        ........                      // weitere Klassenmethoden
}
```

```
class Inbusschraube extends MetrSchraube
{
        .........                     // Klassenvariablen und Methoden
}
// Erzeugen einer Instanz Ibus der Klasse MetrSchraube:
Inbusschraube Ibus  =  new Inbusschraube();
Inbusschraube Ibus;                   // Deklaration der Instanz Ibus
```

5.1.3.5.4 Kapselung

- Die **Schlüsselwörter** private, protected und public werden auch **Zugriffsmodifikatoren** genannt. Sie legen fest, wer die Klassenelemente (Daten und Methoden) benutzen kann.
- private deklarierte Klassenelemente können nur von Methoden der eigenen Klasse benutzt werden, sie sind gegen Fremdeinfluss **gekapselt**.
- protected deklarierte Klassenelemente können von Methoden der eigenen Klasse, abgeleiteter Klassen und der Klassen desselben Pakets benutzt werden, sie sind gegen Fremdeinfluss bedingt **gekapselt**.
- public deklarierte Klassenelemente können auch von Methoden anderer Klassen benutzt werden.
- Fehlt der Zugriffsmodifikator bei der Deklaration, gilt protected.
- Daten sollten immer private sein, damit sie von außerhalb der eigenen Klasse nicht manipuliert werden können. Mit public **würde die Kapselung zerstört!**

5.1.3.5.5 Ereignisbehandlung

- Java-Programme sind **ereignisgesteuert**, d.h. die Objekte sind untätig und warten auf Ereignisse (Nachrichten, Botschaften, Events).
- Sender und Empfänger von Nachrichten sind Objekte.
- Quellen von Nachrichten sind die **Event Sources** (z.B. Buttons, Maus, Fenster).
- Empfänger von Nachrichten sind zunächst die **Event Handler**, die den Event an das betreffende Objekt weiterleiten.
- Damit ein Objekt eine bestimmte Nachrichtenart empfangen kann, muss ihm ein spezieller **EventListener** zugeordnet sein (Registrierung) → wie Abhörwanze.
- Jedem Ereignis wird eine **Event-ID** (Identifikations-Bezeichnung) zugewiesen, an der Ihr Ursprung erkannt werden kann.
- Der **Event Handler** prüft die Event-ID und ruft die zugeh. Antwortmethode auf.
- In Java werden die Eventhandler von 11 verschiedenen **Eventklassen** abgeleitet.
- Jede **Eventklasse** hat ihr spezielles **EventListener**-Interface.
- Bei **Anwendungen** befindet sich der Eventhandler in der Frameklasse, in der sich die Funktion 'main' befindet.
- Bei **Applets** befindet sich der Eventhandler in der Appletklasse, in der sich die Funktion 'init' befindet.

Beispiel: Beim Mausklick auf Objekt Button1 wechselt die Farbe von Objekt Panel1

```
// Registrierung von Button1 als Empfänger von Maus-Ereignissen:
SymMouse aSymMouse = new SymMouse();
Button1.addMouseListener(aSymMouse);
```

```
// Eventhandler, der alle Maus-Ereignisse empfängt
class SymMouse extends java.awt.event.MouseAdapter
{
        // und je nach Ereignis die zugehörige Antwort-Methode aufruft
        public void mouseClicked(java.awt.event.MouseEvent event)
        {
                Object object = event.getSource();
                if ( object == Button1 )
                        Button1_MouseClicked(event);
        }
}
```

```
// Antwort-Methode von Button1
void Button1_MouseClicked(java.awt.event.MouseEvent event)
{
        // Umfärben von Panel1
        if ( Panel1.getBackground() == Color.black )
                Panel1.setBackground(Color.red);
        else
                Panel1.setBackground(Color.black);
}
```

5.1.3.6 Java Programmstrukturen

- Java ist eine **Interpretersprache**.
- Zum Ablauf von Java-Programmen benötigt man deshalb einen **plattformabhängigen Interpreter**, welcher auf der Maschine implementiert sein muss, auf welcher das Java-Programm laufen soll.
- Der Java-Compiler erzeugt den plattformunabhängigen Zwischencode, den **Bytecode.** Dieser ist die Maschinensprache für einen **virtuellen Prozessor**, der auf nahezu jedem modernen Mikroprozessor in Software realisiert werden kann.
- Alle compilierten Programme besitzen die Namenserweiterung ***.class**.

5.1.3.6.1 Java Anwendungsprogramme

- Sie verhalten sich wie autonome Windows Anwendungsprogramme. Zum Lauf benötigen sie den **virtuellen Prozessor** „Java **Just in Time (JIT)** Compiler“ Java.exe:

  Java Anwendung [Parameter]

- **Java.exe** muss auf der benutzten Plattform vorhanden sein (Pfad setzen!)
- Die referenzierten Klassenbibliotheken müssen **zur Laufzeit** zur Verfügung stehen.

Anwendungsprogramme haben folgende Minimalstruktur:

```
import        java.awt.*;   // Direktive z. Referenzieren v. Bibliotheksteilen oder DLL´s
              .....

public class Anwendg extends Frame          // Anwendungsfenster unter Windows
{
      // Deklarationen von Variablen und Objekten der Klasse
      public Anwendg()                      // Initialisierung von Objekten
      {
            setLayout(null);
            setSize(250,150);
            ............
      }
      static public void main(String args[])      // Hauptprogramm
      {
            (new Anwendg()).setVisible(true);
      }
}
```

Beispiel eines Anwendungsprogramms:

Anwendg	**Java AWT Application**
	Umfärben eines Buttons durch Anklicken mit der Maus

5.1.3.6.2 Java Applets

- Applets werden innerhalb eines **Webbrowsers** ausgeführt.
- Als **virtueller Prozessor** wird die **JVM (Java Virtual Machine)** des Browsers verwendet.
- Die referenzierten Klassenbibliotheken müssen **zur Laufzeit vom Browser oder vom Server** zur Verfügung gestellt werden.
- Applets haben aus Sicherheitsgründen **keinen Zugriff** auf die Dateien und die Hardware des Client. Der Browser verhindert dies.
- In Applets werden häufig Animationen eingebaut. Als Eventgenerator ist hiefür ein Timer Objekt notwendig, welches in der Compiler-Bibliothek enthalten ist.
- Werden Objekte aus der Compiler-Bibliothek verwendet, müssen die verwendeten Bibliotheksmodule am Ort des Applets abrufbar sein, z.B. die Symantec itools.

Applets haben folgende Minimalstruktur:

```
import      java.awt.*;          // Direktiven zum Referenzieren
import      java.applet.*;       //      von Bibliotheksteilen oder DLL´s
            .....
public class Appl extends Applet
{
      // Deklarationen von Variablen und Objekten der Klasse
      public void init()         // Initialisierung von Objekten
      {
            setLayout(null);
            setSize(250,150);
            .....
      }
}
```

Beispiel eines Applets:

Appl	**Java Applet**
	Umfärben eines Buttons in einem Applet durch Anklicken mit der Maus

5.1.3.7 Java Programmierbeispiele

5.1.3.7.1 Zeichnen von Schrift und Grafik

- Das **Java AWT** (Abstract Window Toolkit) liefert die Klassenbibliothek für die grundlegende GUI-Programmierung (**Graphical User Interface**).
- Windows sind grafische Schnittstellen, in denen Grafikelemente (Schrift und Figuren) gezeichnet werden können.
- In den integrierten **Entwicklungsumgebungen** (IDE) ist es sehr einfach, Fenster und ihre Inhalte zu erzeugen.

Im folgenden Beispiel soll durch Mausklick auf eine farbige **Zeichenfläche** die Anzahl der bisherigen Klicks **als Text** ausgegeben werden. Der für die Textausgabe nötige Programmausschnitt wird gezeigt:

```
public class Test extends Applet          // Fensterklasse
{
        int Count = 1;                    // Zähler für Klicks

        public void init()                // Initialis. von Fenster und Komponenten
        {       .........
                setLayout(null);
                setVisible(false);
                setSize(326,177);
                .........
                canvas1 = new java.awt.Canvas();        // Deklar. u. Initialis. der
                canvas1.setBounds(37,37,250,100);       //       Zeichenfläche
                canvas1.setFont(new Font("Dialog", Font.PLAIN, 40));
                canvas1.setBackground(new Color(16776960));
                add(canvas1);
                .........
        }
        ............

        void canvas1_MouseClicked(java.awt.event.MouseEvent event)
        {
                Graphics g = canvas1.getGraphics();     // Gerätekontext von canvas1
                g.clearRect(0,0,250,100);
                g.drawString(Count+". Klick",40,60);    // Text zeichnen
                ++Count;                                // Zähler erhöhen
                g.dispose();                            // Gerätekontext freigeben
        }
}
```

Soll statt Text eine **Grafikfigur**, z.B. ein **Rechteck** ausgegeben werden, so kann dies wie folgt gemacht werden:

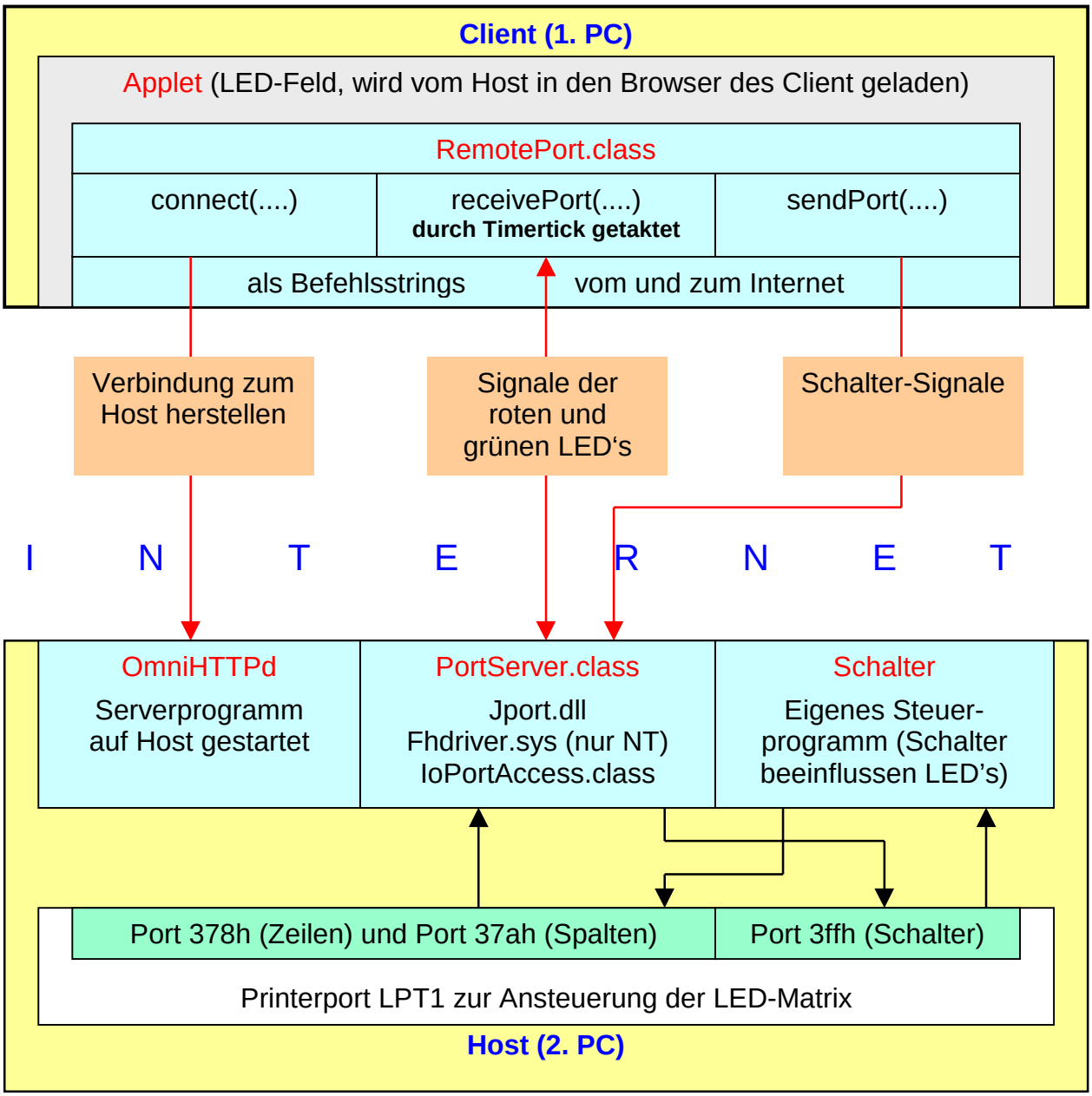
```
        void canvas1_MouseClicked(java.awt.event.MouseEvent event)
        {
                Graphics g = canvas1.getGraphics();     // Gerätekontext von canvas1
                g.drawRect(40,70,100,10);               // Rechteck zeichnen
                g.dispose();                            // Gerätekontext freigeben
        }
```

5.1.3.8 Steuerung über Internet

Das **Internet** bestimmt zunehmend auch die Embedded Systeme.
Die Geräte werden mit einem **HTTP-Server** sowie **HTML-Seiten** und **Applets** ausgestattet, die als **virtuelle Bedienkonsolen** dienen.
Für die Geräteüberwachung und -bedienung wird ein Standard **Web-Browser** verwendet, welcher über HTTP mit dem Server kommuniziert.
Diese Client-Server Kommunikation kann **lokal oder über das Internet** stattfinden.
Es handelt sich dabei immer um eine Interaktion zwischen Benutzer und Gerät.
Die Interaktion zwischen verschiedenen vernetzten Geräten wird später behandelt.

Für das **folgende Beispiel** wird der PC genutzt. Er ist für die Kommunikation über das Internet weitgehend ausgerüstet.

5.1.3.8.1 Struktur der Zugriffe



5.1.3.8.2 Zugriffe auf Ports des Host aus Java-Applikationen auf dem Host

Da Java eine Interpretersprache ist, sind Hardwarezugriffe im Prinzip systemwidrig, da damit die **Portierbarkeit** auf andere Hardware verbaut wird. Da Gerätezugriffe bei Steuerungen unumgänglich sind, müssen Kompromisse gemacht werden, die die Portierbarkeit etwas einschränken. Dazu bieten die Compilerlieferanten **Java Native Interfaces (JNI)** zu anderen Programmiersprachen an, mit denen z.B. native C++ Komponenten mit Hardwarezugang gelinkt werden können.

Hier wird eine Methode vorgestellt, mit welcher der Zugriff auf die Ports im Byte-Format ermöglicht wird, was z.B. den Betrieb der simulierten 8x8-Leuchtdiodenmatrix erlaubt. Sie ist nur bei **Anwendungsprogrammen** anwendbar, da aus Applets der Zugriff auf die Hardware des benutzten Systems sicherheitshalber (Browser!) nicht möglich ist.

Der Zugriff auf die Hardware des Host mit Applets vom Client aus wird im nächsten Kapitel beschrieben.

Im Verzeichnis der Anwendung müssen sich folgende Dateien befinden:

Jport.dll	(Autor: Giebler, FH Ulm)
IOPortAccess.class	(Autor: Giebler, FH Ulm)
Nur bei Windows NT: Fhdriver.sys installieren!!	(Autor: Dubies, FH Ulm)

Damit werden 2 Methoden der Klasse IOPortAccess zur Verfügung gestellt:

```
int   inPort ( int portAddress );
          // liest ein Byte von portAddress (erlaubt sind 0x378 bis 0x3ff)
          // Rückgabewert: bei unerlaubter Adresse -1, sonst eingelesener Wert
```

```
int   outPort ( int portAddress, int value );
          // schreibt ein Byte an portAddress (erlaubt sind 0x378 bis 0x3ff)
          // Rückgabewert: bei unerlaubter Adresse -1, sonst 1
```

Beispiele aus dem 8x8 Simulationsprogramm (Anwendung):

1.
```
class Ports extends IOPortAccess                  // Portzugriff durch Vererbung
{
        static Ports io = new Ports();            // Instanz der Klasse 'Ports'
        public static void refresh()
        {
            int InBits = io.inPort(0x3ff) & 0xe8; // Schalterbits einlesen
            Zeilen     = io.inPort(0x378) & 0xff; // Spaltenwert einlesen
            Spalten  = io.inPort(0x37a) & 7;      // Spaltennummer einlesen
        }
}
```

2.
```
static IOPortAccess io = new IOPortAccess(); // Instanz von IOPortAccess
public static void refresh()
{
        int InBits = io.inPort(0x3ff) & 0xe8;     // Schalterbits einlesen
        Zeilen     = io.inPort(0x378) & 0xff;     // Spaltenwert einlesen
        Spalten  = io.inPort(0x37a) & 7;          // Spaltennummer einlesen
}
```

Die Instanz für den Portzugriff sollte static sein, da sonst die Betriebssystemabfrage ständig wiederholt wird.

5.1.3.8.3 Zugriffe auf Ports des Host aus Java-Applets auf dem Client

Aufgabenstellung (Steuerung über das Internet):

Eine 8x8-Leuchtdiodenmatrix, die als **Applet** in einer HTML-Seite vom Host auf den Client geholt wird, soll die Ports 378h, 37ah und 3ffh des Host abbilden.
Mit den 4 Schaltern soll Port 3ffh des Host beeinflusst werden.
Zum Nachweis der Funktion wird auf dem Host die Simulation der Leuchtdiodenmatrix gestartet und die Hardware-Leuchtdiodenmatrix angeschlossen.

Serverseitige Maßnahmen:

1. Es wird eine handelsübliche Serversoftware auf dem Host installiert und gestartet (z.B. OmniHTTPd der Fa. Omnicron)
2. Im Verzeichnis der Anwendung müssen sich wie oben folgende Dateien befinden:
   Jport.dll (Autor: Giebler, FH Ulm)
   IOPortAccess.class (Autor: Giebler, FH Ulm)
3. Nur bei Windows NT: Fhdriver.sys installieren!! (Autor: Dubies, FH Ulm)
4. Es wird die Anwendung PortServer.class gestartet, die obige Dateien benötigt.
5. Es wird ein Steuerprogramm gestartet, welches die rote Diodenmatrix in Abhängigkeit der 4 Schalter beeinflußt, z. B. Schalter.class

Appletseitige Maßnahmen:

1. Es wird eine HTML-Seite, die eine 8x8-Leuchtdiodenmatrix als Applet enthält, als Homepage des Servers auf dem Host zur Verfügung gestellt.
2. Im Verzeichnis des Applets muss sich die Datei RemotePort.class befinden.
   Damit werden 3 Methoden der Klasse RemotePort zur Verfügung gestellt:

```
void    connect ( String getCodeBase().getHost() );
            // wird bei Applet-Start aufgerufen u. stellt die Verbindung z. Host her
```

```
int     receivePort ( int portAddress );
            // liest ein Byte von portAddress (erlaubt sind 0x378 bis 0x3ff)
```

```
void    sendPort ( int portAddress, int value );
            // schreibt ein Byte an portAddress (erlaubt sind 0x378 bis 0x3ff)
```

Beispiel aus dem 8x8 Simulationsprogramm (Applet):

```
RemotePort io = new RemotePort();           // Instanz von RemotePort
public void start()                         // wird beim Applet-Start ...
{                                           // ... automatisch aufgerufen
        io.connect(getCodeBase().getHost()); // Verbindung zum Host
}
public void refresh()
{
        io.sendPort( 0x3ff, InBits & 0xe8 );  // Schalterbits an Host senden
        Zeilen  =  io.receivePort(0x378) & 0xff; // Spaltenwert v. Host einlesen
        Spalten = io.receivePort(0x37a) & 7;  // Spaltennr. v. Host einlesen
}
```

5.1.3.9 Java Intelligent Network Interface (JINI)

In der Presse ist oft die Rede vom **vernetzten Kühlschrank**, der bei Bedarf automatisch fehlenden Inhalt nachbestellt. Was ist dazu nötig?

Für die Interaktion zwischen verschiedenen vernetzten (embedded) Geräten unterschiedlicher Hersteller und das dynamische Ein- und Ausklinken einzelner Geräte benötigt man **definierte Softwareschnittstellen**. **JINI** bietet diese und stellt darüber hinaus Hilfsmittel für den Betrieb verteilter Systeme bereit.

JINI ist eine auf Java basierende **Komponententechnologie** von Sun Microsystems. Der wichtigste Bestandteil von JINI ist ein **Discovery-Mechanismus**, der das Auffinden von **Lookup-Diensten** ermöglicht. Diese dienen dazu, weitere JINI Dienste anhand von Schnittstellen und Attributen zu finden.

Der Grundgedanke ist, die Geräte durch in Java implementierte Jini-Komponenten zu repräsentieren, die mit der Steuerungssoftware im Gerät über beliebige Schnittstellen kommunizieren, nach außen aber **standardisierte Jini-Schnittstellen** anbieten.

Geräte mit JVM (Java Virtual Machine) verarbeiten ihre lokal gespeicherten Jini-Komponenten selbst mit direkter Anbindung an die Applikation über das **JNI** (Java Native Interface). Die nicht geringen Ressourcen hiefür müssen vorhanden sein.

Geräte ohne JVM, meist kleinere Embedded Systeme, sparen die Ressourcen für die Java Laufzeitumgebung, müssen aber die Klassen der sie repräsentierenden Jini-Komponenten über einen **lokalen HTTP-Server** anderen Rechnern zugänglich machen. Mit Hilfe des **Service Location Protocols (SLP)** kann ein solches Embedded System andern Geräten seine Anwesenheit und die URLs der gespeicherten Jini-Komponenten mitteilen.
Spezielle **Applikationsserver** empfangen die SLP-Nachrichten, laden die Jini-Komponenten vom HTTP-Server des Embedded Systems und führen sie aus. Die Jini-Komponenten kommunizieren über Standardprotokolle oder proprietäre Systeme vom Applikationsserver aus mit ihrem Ursprungssystem. Die **Open Service Gateway Initiative (OSGI)** erarbeitet einen Standard für solche Applikationsserver.

Aufwand für die JINI - Dienste	
Gerät ohne JVM	**Gerät mit JVM** (wird nur eimal im Netzwerk benötigt)
Embedded HTTP-Server SLP Service Agent Speicherplatz für Java-Klassen	Embedded HTTP-Server SLP Service Agent Java Virtual Machine (JVM)

- **Java2 Micro Edition (J2ME)** von Sun bietet die Dienste für ein Gerät mit JVM standardisiert an, benötigt aber mehrere 100 kByte Speicher und ein Betriebssystem, unter dem die JVM läuft.

Die Firma **3SOFT GmbH**, Erlangen, aus deren Firmenschrift ein Teil obiger Informationen stammt, bietet Ingenieurdienstleistungen für JINI an.

Notiz am Rande:
Die Firma **Beck IPC** in Wetzlar bietet eine Kleinsteuerung an, die SMS-Nachrichten versteht und auch versenden kann. Die gesendeten Zeichen werden als Steuerbefehle interpretiert. Damit lassen sich Prozessdaten abfragen und Prozesse starten und beeinflussen. In die Steuerung ist ein GSM-Modem integriert.

5.2 Betriebssysteme

An dieser Stelle interessieren vor allem die Eigenschaften für den Prozess-Einsatz, d.h. das **Realzeitverhalten und die Multitaskingfähigkeit**. Realzeit bedeutet nicht nur schnell, sondern heißt, daß ein System nach Eintreffen eines Interrupts **vorgegebene Zeitlimits** garantieren kann.

5.2.1 Begriffe

Begriffe

Latenzzeit	Zeitdifferenz zwischen dem Eintreffen des Interrupts und dem Beginn seiner Abarbeitung
Jitter	Differenz zwischen maximaler und minimaler Latenzzeit
harte Realzeit	geringer Jitter im Mikrosekundenbereich
weiche Realzeit	größerer Jitter im Millisekundenbereich
Task	'Aufgabe', Anwenderprozess eines Mehrprogrammsystems, welcher mit andern Tasks ‚quasiparallel' abläuft. Die Tasks des Systems werden durch den Dispatcher gewechselt. Die Tasks müssen sinnvollerweise miteinander kommunizieren können.
Thread	'Ausführungsfaden', welcher in multithreaded Programmen im Zeitmultiplex mit anderen Threads läuft. Die Threads eines Programms teilen sich im Gegensatz zu Tasks die Daten. Eine Task kann aus mehreren Threads bestehen.
Semaphor	Signalträger (z.B. 1 Bit). Semaphore werden zur Koordinierung der Zugriffe auf gemeinsam von verschiedenen Tasks benutzte Betriebsmittel eingesetzt. Das xxxIR-Flag beim 80C166 könnte als Semaphor bezeichnet werden.
JTAG	Standard-Schnittstelle für MC-Entwicklungstools, welche unabhängig von der CPU ist

5.2.2 Windows NTE (Embedded NT 4.0)

Unter Windows NT werden ohne spez. Erweiterungen **Latenzzeiten von einigen ms** erreicht, so dass es nicht als 'hartes' Realzeitsystem bezeichnet werden kann.

Windows NT wird zunehmend durch **Zusatz-Hard- und Software** (z.B. Produkte der Fa. DIAdem) realzeitfähig gemacht. Dadurch können z.B. Prozessanbindungskarten unter NT betrieben werden. Die Erweiterungen reichen von der Modifikation der Hardware Abstraction Layer (HAL) bis zur Erweiterung von Windows NT um **Kerneltreiber**, die eine Realzeitschicht bereitstellen.

Den besten Erfolg bietet eine Hardwareerweiterung in Verbindung mit entsprechenden Kerneltreibern. Der **Datenaustausch** zwischen den Tasks kann über den Windows-Standard DDE (Dynamic Data Exchange) erfolgen.

Wegen der **Multitaskingfähigkeit von NT** können Threads mit versch. Realzeitwerten und Prioritäten gebildet werden. Für die Bedienoberfläche reicht weiche Realzeit.

Voraussetzung für den Einsatz von Windows NT ist die Verwendung des PC, wobei natürlich auch die embedded Ausführung (z.B. Card PC) gewählt werden kann.
Nachteilig bei kleineren Systemen ist der **hohe Speicherbedarf von NT**.
Der Einsatz bei Prozessrechnern kam auch wegen Zuverlässigkeitsbedenken bisher nur schleppend voran.

5.2.3 Windows CE

5.2.3.1 Überblick

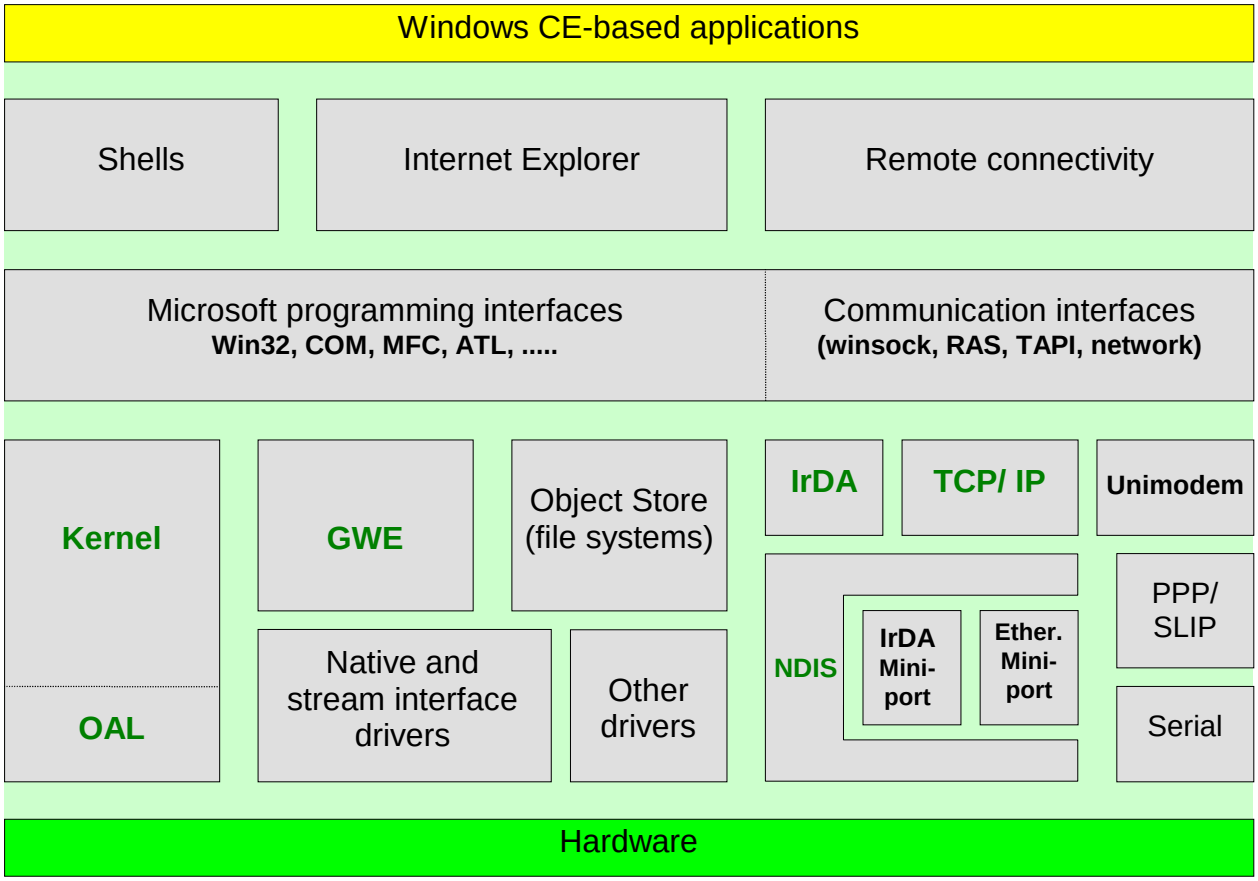
- Windows CE ist eine von Grund auf neu entwickelte, kompakte und tragbare **Betriebssystem-Plattform** für breiten Einsatz in Kommunikations-, Unterhaltungs- und mobilen Computer-Anwendungen.
- Die Windows CE Plattform macht neue Kategorien von kommerziellen und nichtkommerziellen nicht-PC Geräten möglich, die miteinander **kommunizieren, Informationen mit Windows-basierten PCs austauschen and mit dem Internet verbunden** sein können. Die ersten auf Windows CE basierten Handheld PCs wurden 1996 ausgeliefert.
- **Anwendungen:** Handheld PCs, drahtlose Kommunikationsgeräte, Unterhaltungs- und Multimediageräte der nächsten Generation, Internet TV, digitale Set-Top Boxen, Internet "Web phones" u.a.

Windows CE erschließt dem **"embedded systems designer"** die volle Leistung der 32-bit Windows-basierten Development Tools.

5.2.3.2 Eigenschaften von Windows CE

- **32-bit, multitasking, multithreaded** Betriebssystem mit offener Architektur für eine Vielfalt von Einsatzfällen
- **kompakt, skalierbar**, hohe Leistung bei kleinem Speicherbedarf
- freie **Prozessor-Wahl** durch den Anwender
- Integriertes **Power Management**
- Windows CE ist **"embedded"** in Geräten von Microsoft OEM Partnern
- Für den Anwender ist es **nicht direkt verfügbar**, und es kann nicht in einen Desktop Computer oder Laptop geladen werden
- Viele **Windows 95/98-basierte Anwendungen** können mit viel weniger Aufwand auf Windows CE portiert werden als eine Neuentwicklung erfordern würde
- Weil sich Applikationen in ihren Kommunikations-Erfordernissen unterscheiden, unterstützt Windows CE eine breite Kollektion von **Kommunikations-Optionen** mit den zugehörigen APIs
- In Version 2.0 von Windows CE wurde die **Graphik Display Architektur** radikal geändert, um GDI Interfaces mit einem oder mehreren Display Drivern zu ermöglichen. Die GDI-Funktionalität wurde erweitert, um viele der grafischen Eigenschaften anderer Microsoft Win32® "application programming interfaces" (API) zu ermöglichen
- **Ab Mitte 2000 steht Version 3.0 zur Verfügung mit 256 statt 8 Prioritäten für Threads, bessere Prioritätsverwaltung, bessere Zeitscheibenkontrolle, Realzeitleistung 50 bis 100 µs**

5.2.3.3 System Architektur von Windows CE



Begriffserklärungen	
Kernel	**Betriebssystemkern** portierbares multithreaded und mutitasking Win32 System mit dem-selben Prozess- und Datenmodell und Dateisystem wie Windows NT.
GWE	**Graphics, Window Manager, Event Manager** Windows CE Implementierung d. Windows GDI (Graphic Device Int.). Unterstützt viele Displays von keinem bis Color SVGA.
OAL	**OEM Adaption Layer** Diese Hardware Adaption Layer (HAL) wird vom OEM (Original Equipment Manufacturer) implementiert und verbindet Windows CE mit dem Gerät. Es enthält Interrupt Service Routinen und hardware-spezifische Funktionen.
IrDA	**I**nfrar**e**d **D**ata **A**ssociation → Serielle Infrarot Schnittstelle
NDIS	Test Tool
TCP/ IP	**T**ransmission **C**ontrol **P**rotocol / **I**nternet **P**rotocol

Windows CE Folien	**Wie kommt man zu Windows CE ?** **Wie wird Windows CE implementiert ?**

5.2.4 EUROS*plus* Enhanced Universal Realtime Operating System

Anbieter: Ing. Büro Dr. Kaneff, Nürnberg, Neutorgraben 17

5.2.4.1 Eigenschaften und Komponenten

Aufgrund der modularen und hardwareunabhängigen Konzeption ist EUROS geeignet für **Embedded Controller, Boardsysteme, Automatisierungsgeräte, PC**

EUROSplus

- ist **skalierbar** und damit für viele Anwendungsgebiete geeignet
- hat **kurze Reaktionszeiten**
- ist **robust und flexibel**
- kann direkt **on chip** in EPROM / Flash-EPROM eingebrannt werden
- ist eine hardwareunabhängige **Softwareplattform in C**
- hat eine leistungsfähige **Treiberschnittstelle** für eff. Treibererstellung
- bietet eine umfassende Treiberpalette für **Standardschnittstellen**
- ist **zukunftssicher** durch Anp. des Mikrokernels an neue Chipversionen
- ermöglicht **Multitasking** unter Nutzung der jeweiligen Chiparchitektur
- bietet eine leistungsfähige **Cross-Entwicklungsumgebung** für den PC
- ermöglicht den Zugang zum **Internet** mit HTTP-Server
- läßt **Java-Anwendungen** durch die Virtual Machine EUROS*vm* auf dem Zielsystem auch aus ROM / Flash-Speicher ausführen
- bietet **Feldbuslösungen** für Ethernet, CAN, IrDA, DIN-Meßbus, InterBus S, Profibus und andere

Komponenten von EUROS*plus* :

- **Mikrokernel**
  Er enthält eine Sammlung von Unterprogrammen, die von den restlichen EUROS*plus* Komponenten gemeinsam benutzt werden.
- **Prozeßmanager**
  Er umfaßt einen Satz von Systemdiensten für Taskverwaltung, Speicherverwaltung usw. zur Erfüllung der Anforderungen an ein Realzeit-Betriebssystem. Zentrales Objekt ist die Root-Task, welche die andern Tasks erzeugt und verwaltet.
- **I/O System**
  Es beinhaltet die Treiberschnittstelle und ist die Nahtstelle zur Hardware.
- **C-Laufzeitbibliothek**
  Sie entspricht dem ANSI-C Standard und ist reentrant. Sie wird eimal im System installiert und von allen Tasks gemeinsam genutzt.
- **Filesystem Manager**
  Er ist ein Dateiverwaltungssystem im MS-DOS Format.
- **Netzerk Manager**
  Er implementiert die TCP/IP, UDP/IP, SLIP, PPP, Webserver- Protokolle.
- **GUI/WINDOWING Komponente**
  Sie stellt eine grafische Benutzeroberfl. für Anwendungen zur Verfügung.
- **Kompatibilitätsbox**
  Sie ermöglicht die Portierung vorhandener Realzeitanwendungen ohne Leistungsverlust auf **EUROS*plus***.

5.2.4.2 EUROS in der Applikation

Ein **Multitasking Minimalsystem** kommt allein mit dem Mikrokernel und dem Prozeßmanager aus. Die Programmierbeispiele werden auf dieser Basis erstellt.

Die Zuteilung der **CPU-Rechenzeit** an die Tasks bestimmt der **Scheduler**, der Teil des Kernels ist. Er muß bei Taskwechsel auch dafür sorgen, daß der Kontext (Registerinhalte) der alten Task gerettet und der Kontext der neuen Task geladen wird. EUROS bietet 2 grundlegende Scheduling-Verfahren, die auch gemischt verwendet werden können:

- **Zeitscheibenverfahren (Round Robin)**
  Die Tasks mit Zustand „ready“ werden in eine „Task-Ready-Queue“ (Warteschlange) eingeordnet
  in Gruppen gleicher Priorität,
  in diesen Prioritätsgruppen jeweils nach FIFO-Prinzip (first in – first out).
  Nach Ablauf der vom Programmierer eingestellten Zeit wird die aktive Task unterbrochen und ans Ende der Warteschlange gleicher Priorität gestellt.
- **Prioritätssteuerung**
  Die Task mit der momentan höchsten Priorität läuft so lange, bis sie
  1. sich selbst beendet,
  2. durch ein belegtes Betriebsmittel blockiert wird oder
  3. eine Task mit höherer Priorität „ready“ wird.

  Bei Realzeitprozessen ist die Prioritätssteuerung der Normalfall.
  EUROS hat 256 Prioritätsstufen.
  Der Programmierer muß die Prioritätenliste sogfältig planen.
  Vorsicht: Eine Task mit der höchsten Priorität würde mit einer Endlosschleife alle andern Tasks blockieren!

Jedes Realzeitereignis läßt den Scheduler aktiv werden und sorgt für eine neue Bewertung der Taskfolge.

Um sowohl Rechenzeit als auch Speicherplatz optimal einzusetzen, werden die **Module** von EUROS direkt als C-Funktionen in das Anwendungsprogramm eingefügt. Durch die Modularität des Systems läßt es sich optimal an die Erfordernisse der Anwendung anpassen.

In folgenden Stufen werden Beispielprogramme besprochen, vorgeführt und Versuche durchgeführt:

- Zur Einführung wird ein **Minimalbeispiel** in der Datei **`3Tasks\3Tasks`** mit drei Tasks verwendet, zunächst ohne Panic-Dienste, um die Programmstruktur leichter zu demonstrieren.
- Danach wird das gleiche Programm mit **Panic-Diensten** gezeigt, um die Fehlerüberwachung zu demonstrieren. Es handelt sich um die Datei **`3Tasks_P\3Tasks`**.
- Durch Variation der Parameter kann die Funktionsweise der Tasksteuerung getestet werden. Diese Versuchsreihe wird mit dem übersichtlicheren Programm **`3Tasks\3Tasks`** durchgeführt.

Das Tool **EUROScope** ermöglicht Singlestep, Breakpoints und die Anzeige der Variablen, Register, Speicherinhalte und EUROS-Objekte. Auch kann das Histogramm der CPU-Zuteilung an die Tasks sichtbar gemacht werden.

5.2.4.2.1 Initialisierung der Taskstruktur

Die Eigenschaften jeder Task werden mit einer Standardstruktur initialisiert.
Diese sind in **task.h** definiert.

```
static const tTdp Tdp1 =
{
   0,          /* uint16 CpuID              reserviert, zur Zeit immer 0      */
   0,          /* int Arg                   frei interpretierbar              */
   NULL,       /* void *pArg                Pointer frei interpretierbar      */
   Task1,      /* void (*pFunc)(int,void*)  Name der Taskfunktion             */
   500,        /* size_t  StackSize         Stackgröße der Task in Bytes      */
   1,          /* uint16 DefaultPriority    Anfangspriorität der Task         */
   0           /* uint16 Options            Kennung für Taskeigenschaften     */
};
```

5.2.4.2.2 Initialisieren und Starten von EUROS

Ablauf des Programmstarts:

- Laden der Anwendung einschließlich EUROS in die Baugruppe
- Automatischer Start des compilerabhängigen **Startsatzes** durch Reset oder Debugger
- Der Startsatz führt den Sprung zu **main()** aus, wo unser C-Programm beginnt
- Das Initialisieren und Starten von Euros geschieht **mit Systemaufrufen des Mikrokernels und des Prozeß-Managers** in der Funktion **main()**.
- Dabei ist folgende Reihenfolge einzuhalten:
  1. Mikrokernel
  2. I/O-System (im Beispiel nicht verwendet)
  3. Prozeßmanager
  4. File-System (im Beispiel nicht verwendet)
  5. POSIX-Schnittstelle (logische Hardw.-Schnittst., im Bsp. nicht verw.)
- Grundsätzlich ist zu beachten, daß vorgesehene EUROS-Objekte erst nach der Initialisierung der entsprechenden **Komponente** ausgeführt werden.
- In der Root-Task kann grundsätzlich User-Programmcode ausgeführt werden. Übersichtlicher ist aber, dort nur Taskverwaltung anzusiedeln.

C-Programmtext	Vollständiger Programmtext von 3Tasks.c
int HardwareInit (void)	Initialisierung der Hardware des verwendeten Boards → **hw_init.c**
int PanicInit(void)	Initialisiert die Schnittstelle der Panic-Konsole. Danach können die Debug-Dienste in Anspruch genommen werden.
int MicrokernelInit (void)	Der Mikrokernel wird mit eigenem Speicherbereich initialisiert und automatisch die RootTask erzeugt (setzt die Struktur mit festem Namen **RootTaskTdp** voraus) . Auch wird die Funktion TrapsInit → **tr_init.c** aufgerufen, welche die Interrupts initialisiert (kann vom Anwender geändert werden).

`int ProcessManagerInit (void)`	Die Systemkomponente Prozeßmanager wird initialisiert. Dabei werden jeweils Speicherbereiche für Tasks und andere Objekte des PM eingerichtet.
`int CommonSysTimerInit (uint32 Period)`	Hier wird der Common-System-Timer als einziger Timer für System-Watchdogs, Round-Robin und User-Watchdogs verwendet. Es können dafür jeweils auch getrennte Timer genommen werden. Timer vor `EurosStart(..)` initialisieren! Period: Zeit zwischen zwei Interrupts in Nanosek.
`void EurosStart (void)`	Das Betriebssystem wird initialisiert und die Root-Task automatisch gestartet durch Aufruf von `RootTask()`. Diese übernimmt die CPU-Steuerung, also auch die der andern Tasks.

5.2.4.2.3 Initialisieren und Starten der Tasks

In ihr werden die einzelnen Tasks mit Systemdiensten des Prozessmanagers als Systemobjekte erzeugt, konfiguriert und gestartet. Dies vollzieht sich in 3 Stufen:

`int TaskCreate( const char *pPath, uint16 ObjMode, const tTdp *pTdp)`	Eine Task mit dem logischen Namen pPath wird als Euros-Objekt dynamisch erzeugt. Sie ist damit der Taskverwaltung bekannt und hat den Zustand „dormant“. Sie muß explizit gestartet werden.
`int TaskSetRoundRobin( int TaskId, uint16 ExecMode, uint16 Ticks)`	Die Round-Robin-Dauer (Timeout) der benannten Task wird auf den Wert „Ticks“ als Anzahl des in `CommonSysTimerInit` eingestellten Systemtakts gesetzt.
`int TaskStart( int TaskId, uint16 ExecMode, uint32 TimeLimit, uint16 Prio, int Arg, void *pArg)`	Die benannte Task wird gestartet, d.h. in die Ready-Queue eingetragen. In dieser Warteschlange von EUROS warten alle ablaufbereiten Tasks darauf, von EUROS Rechenzeit zu erhalten. Ist die Zieltask zum Zeitpunkt des Taskstarts im Zustand dormant, wird sie in den Zustand ready versetzt.

5.2.4.3 Programm-Experimente am Evaluation-Board

Im Beispielprogramm `3Tasks.c` wird eine Root-Task für die Taskverwaltung und zwei weitere Tasks zum Steuern von LEDs verwendet. Dies ist die Minimalkonfiguration zum Aufzeigen der Taskfunktionen. Durch Veränderung

- des Programms in den Tasks
- des Task-Timeout
- der Prioritäten von Root-Task und der anderen Tasks

kann die Wirkungsweise der Tasksteuerung gezielt getestet werden.

Von Fall zu Fall werden nur die veränderten Einstellungen angegeben.

Im Programmteil `main()` wird der verwendete Port P2 initialisiert:

```
DP2 = 0xffff;          // LED-Ports (Port 2) sind Output
P2  = 0xffff;          // alle 16 LED aus
```

5.2.4.3.1 Tasks 1 und 2 mit Endlosschleife

Fall 1		
Root-Task	Stacksize: Anfangs-Priorität: Options: `kein User-Code`	500 Byte 1 keine
Task 1	Stacksize: Anfangs-Priorität: Options: `while(1) P2=0x00ff;`	500 Byte 1 keine // high 8 LED ein, low 8 aus
Task 2	Stacksize: Anfangs-Priorität: Options: `while(1) P2=0xff00;`	500 Byte 1 keine // high 8 LED aus, low 8 aus

Verhalten:

- Die beiden LED-Gruppen blinken im Sekundentakt
- da alle Prioritäten gleich sind, wechseln die Tasks allein nach Round-Robin
- wegen Endlosschleifen nutzen beide Tasks ihr Timeout

Fall 2		
Root-Task	Anfangs-Priorität:	0
Task 1	Anfangs-Priorität:	1
Task 2	Anfangs-Priorität:	1

Verhalten:

- Programmfunktion wie bei Fall 1: Blinken im Sekundentakt
- Die niedrigere Priorität der Root-Task bezieht sich nur auf den User-Code, legt aber nicht die Taskverwaltung lahm! Sie benötigt hier keine Performance.

Fall 3		
Root-Task	Anfangs-Priorität:	2
Task 1	Anfangs-Priorität:	1
Task 2	Anfangs-Priorität:	1

Verhalten:

- Alle LED ausgeschaltet (Startwert)
- Die Root-Task ist dauernd aktiv, die andern Tasks sind ready, kommen aber nicht dran!

Fall 4		
Root-Task	Anfangs-Priorität:	1
Task 1	Anfangs-Priorität:	2
Task 2	Anfangs-Priorität:	1

Verhalten:

- High 8 LED ein
- Task 1 ist dauernd aktiv wg. Prio., Task 2 ist ready, kommt aber nicht zum Zug
- Die Task mit der höchsten Priorität blockiert Round-Robin wg. Endlosschleife

5.2.4.3.2 Tasks 1 und 2 ohne Endlosschleife

Fall 1	
Root-Task	Anfangs-Priorität: 1
Task 1	Anfangs-Priorität: 1 `P2=0x00ff;` // ohne while(1), high 8 LED ein, low 8 aus
Task 2	Anfangs-Priorität: 1 `P2=0xff00;` // ohne while(1), high 8 LED aus, low 8 ein

Verhalten:

- High 8 LED kurz an, Low 8 LED daurnd an
- Task 1 und 2 laufen je nur einmal durch, weil die Taskprogr. endlich sind
- Taskreihenfolge durch ID bestimmt
- Danach: Tasks 1 und 2 dormant

Fall 2	
Root-Task	Anfangs-Priorität: 1
Task 1	Anfangs-Priorität: 2
Task 2	Anfangs-Priorität: 1

Verhalten:

- High 8 LED kurz an, Low 8 LED daurnd an wie bei Fall 1
- zuerst Task 1 wegen Priorität, dann Task 2
- Danach: Tasks 1 und 2 dormant

Fall 3	
Root-Task	Anfangs-Priorität: 1
Task 1	Anfangs-Priorität: 1
Task 2	Anfangs-Priorität: 2

Verhalten:

- Low 8 LED kurz an, High 8 LED daurnd an (umgekehrt wie bei Fall 1)
- zuerst Task 2 wegen Priorität, dann Task 1
- Danach: Tasks 1 und 2 dormant

5.2.4.3.3 Erkenntnisse

- **Round-Robin ist der Prioritätssteuerung unterlegt**, die Prioritätsentscheidung dominiert zwangsläufig Round-Robin.
- Die Root-Task ist naturgemäß immer **aktiv** wegen der Taskverwaltung.
- Tasks, deren Programm beendet ist, gehen in den Zustand **dormant.**
- Tasks, deren Programm unterbrochen wurde, haben den Zustand **ready** und warten in der **Task-Ready-Queue** auf die CPU-Zuteilung.
- Die Reihenfolge von Tasks gleicher Priorität wird durch die ID-Nr. bestimmt.
- Die **Idle-Task** läuft immer dann, wenn keine Anwendertask ready ist.
- Mit Round-Robin und Prioritätssteuerung allein lassen sich nur einfache, weiche Realzeitprozesse bedienen.
- Es müssen weitere Eingriffsmöglichkeiten hinzukommen wie
  1. Dynamische Veränderung der Taskparameter, z.B. der Priorität
  2. Systemaufrufe aus den Tasks heraus
  3. Kommunikation zwischen den Tasks über Flags und Mailboxen
  4. Interruptsteuerung

5.2.4.4 Interrupt-Dienste des Mikrokernel

Interrupts **unterbrechen den Programmlauf**, um auf zeitkritische Ereignisse zu reagieren. Sie wurden bereits beschrieben.
EUROS behandelt Interrupts als **dynamische Systemobjekte**.

Interrupts können auf zwei Arten verarbeitet werden:

- in **Interrupt-Handlern** (C-Funktionen), die schnelle Ausführung erlauben, aber keine Systemaufrufe absetzen können.
- in **Interrupt-Tasks**, die von den Interrupts direkt gestartet werden. Diese müssen eine höhere Priorität wie die übrigen Tasks haben und können Systemaufrufe absetzen.

Für die Konfiguration und Steuerung von Interrupts stellt der Mikrokernel 9 Systemaufrufe zur Verfügung. Damit können

- Interruptquellen **global oder einzeln** zugelassen und gesperrt werden
- Interrupthandler **angemeldet** und als belegt oder frei markiert werden
- **Interrupt-Prolog-Dienste** eingerichtet werden, die vor die Interruptroutine geschaltet sind (der Int-Vektor zeigt darauf), um die angetroffene Systemumgebung zu retten und dem Int.-Handler eine definierte Systemumgebung anzubieten.

Für Interruptbearbeitung kann das System folgende **Zustände** annehmen:

- **I** (**Interrupt-Zustand**) direkt nach der Auslösung. Hier wird das Betriebssystem umgangen, der Handler muß den Kontext der CPU am Beginn sichern und am Ende restaurieren, sonst ist das System danach undefiniert.
- **N** (**Nested-Interrupt-Zustand**). Hier sind Interrupts mit höherer Priorität freigegeben und können unterbrechen.
- **S** (**System-Zustand**). Hier sind alle Interrupts freigegeben. Der jetzt aktive Handler kann von beliebigen anderen Interrupts unterbrochen werden.
- **X** (**Interrupt-Bearbeitung beendet**). Die Interruptbearbeitung wird als beendet gekennzeichnet, der Handler verlassen und der Systemkontext wiederhergestellt.

Für die **Umschaltung zwischen den Systemzuständen** stellt der Mikrokernel 6 Systemaufrufe zur Verfügung für folgende Transformationen:

I → N	I → S	I → X	N → S	N → X	S → X

5.2.4.5 Prozeßmanager

Vorraussetzung für die Funktion des Prozeßmanagers ist die vorherige Installation des Mikrokernels.

Der Prozeßmanager benötigt **Datenspeicher** für die Datenstrukturen zur Verwaltung seiner Systemobjekte. Die Datenblöcke besitzen einen einheitlichen Aufbau und werden erst bei Bedarf automatisch **objektspezifisch** konfiguriert.

Der Prozeßmanager enthält die meisten **Systemfunktionen**, in Gruppen eingeteilt:

Initialisierungs- und Statusfunktionen	Shared Memory-Dienste
Task-Dienste	Pipeline-Dienste
Eventflag- und Signaldienste	Mailbox-Dienste
Pool-Dienste	Semaphore-Dienste
Datum- und Uhrzeitdienste	Bitprocessing-Dienste

5.2.4.5.1 Task-Dienste

Tasks können folgende 6 **Zustände** einnehmen:

rechnend	Task hat die **CPU-Zuteilung**
bereit	Task wartet in der Ready-Queue auf Zuteilung der CPU
dormant	Task ist existent u.d. Verw. bekannt, sie wird nach Start aktiv
wartend	Task wurde stillgelegt und muß geweckt werden
suspendiert	Task ist ready, aber durch Flag für die Ready-Queue gesperrt
nicht existent	Task wurde aus der Taskverwaltung entfernt

Bei der Erzeugung erhält jede Task **3 private Systemobjekte**: je eine Ereignisflag-, Signal- und System-Signalgruppe, die nur durch sie benutzbar sind.

Der Prozeß-Manager stellt **22 Systemaufrufe** zur Tasksteuerung bereit zum:

- Erzeugen und Löschen
- Starten und Beenden
- Aussetzen und Wiederaufnehmen der Aktivität
- Verdrängen und Zulassen
- Abfragen von Status, ID-Nr. und Priorität
- Setzen von Zeitintervallen für Round-Robin und Pausen

Da sich damit die Tasks gegenseitig beeinflussen können, ist eine Vielzahl von Möglichkeiten für die Steuerung des Sytemverhaltens gegeben.
Ein einfaches Programmbeispiel soll dies zeigen:

Root-Task	Anfangs-Priorität: 1

Task 1	Anfangs-Priorität: 1
	`long i;`
	`P2=0x00ff;` // high LED ein
	`for (i = 0xfffffh; i != 0; i--);` // verzögern
	// Task 2 starten:
	`TaskStart(TaskID2,SINGLE,0,PRIO_TAR_INH,0,NULL);`

Task 2	Anfangs-Priorität: 1
	`long i;`
	`P2=0xff00;` // low LED ein
	`for (i = 0xfffffh; i != 0; i--);` // verzögern
	// Task 1 starten:
	`TaskStart(TaskID1,SINGLE,0,PRIO_TAR_INH,0,NULL);`

Verhalten:

- LED blinken mit einer durch die Verzögerung bestimmten Frequenz
- durch den Systembefehl `TaskStart` werden die Tasks gegenseitig dauernd neu gestartet (sonst würden sie nur einmal durchlaufen)
- wenn die Verzögerungszeit < Timeout von Round Robin ist, kommt Round Robin nicht zum Tragen

5.2.4.5.2 Speicherverwaltung

EUROS stellt **Koordinierungsmechanismen zur dynamischen Speicherverwaltung** in Form von Systemaufrufen bereit, welche eine flexible und sparsame Belegung des Arbeitsspeichers gestattet. Der Speicher wird in **Pools** organisiert, aus denen sich die Tasks durch Systemaufruf bedienen. Pool-Grundtypen:

- **Megapools** mit max. 4 GB, eingeteilt in Blöcke fester Länge.
- **Memorypools** mit Controllerabhängiger Größe, beim C167 sind es 16 kB.
- **Bufferpools** (Fixed-Size-Buffer), eingeteilt in gleich große Blöcke für dyn. Zuweisung, deren Blockgröße bei der Erzeugung vom Anwender definiert wird.

Der **Shared-Memory-Bereich** ist ein einzelner Speicherbereich, auf den mehrere Tasks gleichzeitig Zugriff haben. Der Zugriff muß sorgfältig geregelt werden, meist über ein **Semaphor**, damit beim Taskwechsel keine Kollisionen auftreten.

5.2.4.5.3 Weitere EUROS-Systemobjekte

Eventflags <cef.h>	allgemeine und private **Bitgruppen** zu je 16 Bit als Merker für Koordinierungs- und Synchronisierzwecke, deren Bits durch Sytemaufrufe gesetzt, getestet und gelöscht werden können. Eine Task kann auf das Setzen oder Löschen eines Flags warten. `int EventFlagCreate (const char *pPath, uint16 ObjMode);`
Signale <sig.h>	wie Eventflags, jedoch wird das **Signalbit gelöscht**, wenn eine wartende Task dieses empfängt. Durch Systemaufrufe können Signale an Funktionen angeschlossen und von Tasks gesendet und empfangen werden.
Mailboxen <mailbox.h>	universelles Kommunikationsmittel für Tasks, Port-Treiber und Resource-Manager. Ein Programm kann **Nachrichten** an eine Mailbox senden, die von einem Empfänger abgeholt werden. Der Nachrichtentyp einer Mailbox wird beim Erzeugen definiert. Auf eine Mailbox können **alle Tasks** Zugriff haben. Die Zugriffsteuerung erfolgt über Warteschlangen. Eventflags und Signale könnten durch Mailboxen ersetzt werden. Dadurch würde jedoch die Flexibilität leiden. Erzeugung: `int MailboxCreate (const char *pPath, uint16 ObjMode, int MaxMessages, int MaxTasks, size_t MessageSize);`
Pipes <pipe.h>	zur gepufferten und koordinierten Übertragung von **Byteströmen** zwischen Tasks.
Semaphore <sem.h>	sind bei EUROS Zähler für Zugriffe auf von Tasks gemeinsam benutzte **Betriebsmittel** wie Schnittstellen, Datenfelder oder Ausgabegeräte. Der gesetzte Grenzwert von ≥ 1 kann einer oder mehreren Tasks Zugriff auf das Betriebsmittel gestatten. Semaphore können durch Systemaufrufe belegt, geprüft und freigegeben werden. Erzeugung: `int SemCreate (const char *pPath, uint16 ObjMode, int Count);`
Bit-Processing <bit.h>	erlaubt Ein-Ausgabe an **bitorientierten Schnittstellen**.

5.2.4.6 I/O—System

Die EUROS **I/O-Systemkomponente** ist die Schnittstelle zwischen Tasks und Port-Treiber bzw. Resource-Manager.
Mit ihren Systemaufrufen ist es möglich, Port-Treiber bzw. Resource-Manager zu installieren und Ein/Ausgabe-Anforderungen auszugeben.

Es gibt **2 Gruppen von Systemdiensten**:

- Initialisierungs- und Statusfunktionen
- Allgemeine Ein/Ausgabe-Dienste
  Dies sind Funktionen zur Handhabung von synchronen/asynchronen Lese- und Schreiboperationen. Deren Implementierung erfolgt durch die I/O-Objekte Port-Treiber und Resource-Manager, die vom I/O-System über eine einheitliche Schnittstelle erzeugt, parametriert, geöffnet, geschlossen und gelöscht werden können.

Ein **Port-Treiber** ist ein I/O-Objekt, das einen Hardware-Baustein beschreibt und die zur Bedienung nötigen Dienste bereitstellt.

Ein **Resource-Manager** ist ein I/O-Objekt, das ein bestimmtes Übertragungsprotokoll realisiert und die benötigten Dienste bereitstellt.

Das **EUROS-Treiberkonzept** ermöglicht es, durch Kombination von Port-Treibern und Resource-Managern gleichen Typs Hardware-Schnittstellen mit gewünschen Protokollen zu versehen, z.B. eine serielle Schnittstelle mit dem SLIP-Protokoll. Dies vereinfacht und flexibilisiert die Treiberentwicklung für neue Hardware.

Erzeugung eines I/O-Objekts (<io.h>):

```
int IoCreate (const char pPath,              /* Adr.d.Pfadnamens */
              uint16 ObjMode,                /* Objektparameter */
              const tIoConfig pPoConfig);    /* Konfig.-Struktur */
```

5.2.4.6.1 Erzeugung eines C167CR CAN Port-Treibers

Die Param. des Treibers werden in folgender **Struktur** übergeben, z.B. für CAN167:

```
const tIoConfig Can167PortConfig =
{
  0,                  /* int CpuId                ID der CPU  */
  CAN167_CHANCLASS,/* uint16 Class                def. Objekt-Klasse (CL_..) */
  CAN167_CHANTYPE,    /* uint16 SubType           def. Objekt-Subtyp (T_...) */
  NULL,               /* tIoReqFunc pFuncEntry    Normale Eintrittsfunktion */
  NULL,               /* tIoLinkEntry pLinkEntry  Link-Eintrittsfunktion */
  NULL,               /* tIoPassEntry pPassEntry  Pass-Eintrittsfunktion */
  &PrimaryCanSpec,    /* const void *pSpec        Pointer auf Konfig.-Daten */
  FOREVER,            /* uint32 TimeLimit         Default Timeout für Requ. */
  NULL                /* tIoReqFunc *pFuncTable   Pointer auf Fkt.-Tab. */
};
```

Der **CAN Port-Treiber** benutzt den internen CAN-Controller des C167CR in der Basic-CAN-Bertriebsart. Die Message-Filterung wird vom Treiber vorgenommen, eine durchzulassende Message muß beim Treiber angemeldet werden.

Zur Erzeugung der obersten Ebene des Treibers wird folgende **Struktur** verwendet, welche die Konfigurations-Parameter des CAN167-Treibers enthält:

```
static const tCan167ConfigSpec PrimaryCanSpec =
{
  CAN167ADDR_PRIMARY,  /*uint32 ChannelAddr.  Kanaladresse */
  1000000,             /* uint32 DefaultBaud  Def.-Bitrate bis 1 MBit/s  */
  CAN167VECT_PRIMARY, /* uint16 IntVect       Interrupt-Vektor */
  (3<<2)|2,            /* uint16 IntLevel     ILVL, GLVL */
  10,                  /* uint16 FilterSize   Anz. der angem. Messages */
  5,                   /* uint16 RecvBuffers  Anzahl der Empfangspuffer */
};
```

Der Aufruf für die Erzeugung des Treibers lautet:

```
extern const tIoConfig Can167PortConfig;     // Deklaration d. Struktur
int CanId = IoCreate("Io/can167/Primary",0,&Can167PortConfig);
```

5.2.4.6.2 Programmbeispiel für Kommunikation über den CAN Bus

Die meisten I/O-Systemaufrufe sind für den CAN-Treiber implementiert.
Die Prototypen hierfür stehen in <can167c.h>.

Mit einem kleinen Programm im Verzeichnis **Test_can167\Test_can** kann eine **Kommunikation** mit unserem Labor-CAN-Bus hergestellt werden. Die Implementierung erfolgt in folgenden Schritten:

```
PortId = IoCreate("Io/CAN",0,&Can167PortConfig);
                                  // Can I/O Objekt erzeugen
ChannelId = IoCreate("Io/CAN/1",0,&Can167Channel1Config);
                                  // Can Kanal-Objekt erzeugen
IoOpen(ChannelId, 0)        // Can Kanal-Objekt oeffnen
.....              // hier werden die Kanalparameter, z.B. f. Messagefilter, zugewiesen
IoControl(ChannelId,WAIT,FOREVER,&Control);  // Can Objekt einstellen
SendData.Length = 2;              // Message-Parameter setzen
SendData.MsgID = SEND_ID;
SendData.SendCall = NULL;
SendData.pHandle = NULL;
Write.Write.pBuffer = &SendData;
Write.Write.MaxNumber = sizeof(SendData);
                                  // jetzt können Daten gesendet und empfangen werden
for (i=0UL; i<65000UL; i++)     // fortlaufend Zahlen erzeugen
{
     SendData.Data[0] = (ubyte) i;        // Sendepuffer laden
     SendData.Data[1] = (ubyte) (i>>8); // Senden im Sek.-Takt
     if (IoWrite(ChannelId, WAIT, 1|SEC, &Write) == FAIL)
          PutChar('X');   // bei Fehler Ausgabe von 'X' im Terminalfenster
     else
          PutChar('.');  // bei Gelingen Ausgabe von '.' im Terminalfenster
}
.....              // hier werden die oben erzeugten Objekte wieder gelöscht
```

5.2.5 Weitere Embedded Betriebssysteme

5.2.5.1 OSE

OSE ist ein komplexes Realzeit Multitasking Betriebssystem für Embedded Systeme. Es ist u.a. für den **Controller 80C166** verfügbar.

- Es besitzt eine komfortable Entwicklungsumgebung mit **Debugger und Simulator**.
- Es verwendet ein **Message Passing System** zur schnellen asynchronen Prozess Kommunikation. Die Aufgabe der Anwendung wird in mehrere kommunizierende Prozesse (Tasks) aufgeteilt. Jeder Prozess ist ein separates Objekt mit seinen eigenen geschützten Ressourcen, welche vom Kern gesteuert werden.
- Das System kann in **logische Teile** unterteilt werden, die getrennt entwickelt und getestet werden können.
- Die **Botschaften** bilden ein sauberes Interface zwischen den Prozessen.

Komponenten:

- **Realzeit Betriebssystem Kern**
- Embedded **Filesystem** zum Speichern von Programmodulen und zum Daten-austausch zwischen Geräten
- **Program Handler** zum Laden, Wechseln und Entfernen während Runtime
- Internet Protocol Utilities (unterstützt u.a. **TCP/IP, ftp, telnet**)
- skalierbarer **Webserver** für Embedded Systeme (Speicher ab < 6 kB) zum Koppeln von Embedded Anwendungen und Internet
- **SNMP** (Simple Network Management Protocol)
- Unterstützung zum Laden und Abspielen von **Java-Applets**
- **Board Support** Packages für die Controller Kits führender Hersteller
- **Memory Management** System zum Isolieren und Schützen von Speicherbereichen gegeneinander und gegen den Kernel

5.2.5.2 OSEK

- **OSEK**: **O**ffene **S**ysteme und deren Schnittstellen für die **E**lektronik im **K**fz. Es spezifiziert ein Multitasking Realzeitbetriebssystem für die Automobilindustrie u. andere, geschaffen durch ein internat. Industriekonsortium und die Uni Karlsruhe.
- höchst untypisch: **Kunden definieren den Standard** für Betriebssystemhersteller.
- **3 Layer**: Betriebssystem, Kommunikation und Network Management.
- **OSEKplus** ist skalierbar und braucht nur soviele Ressourcen wie nötig. Durch seine Portabilität kann es an neue CPU Plattformen angepasst werden.
- Die **OSEK Communication Specification** V2.1 stellt die Kommunikation mit dem CANopen Protokoll sicher (beim Auto unverzichtbar!).
- Die Durchsetzung als **Standard** für harte Realzeit ist noch offen.
- Realisierungen: EUROSek Fa. Ing.-Büro Dr.Kaneff, Nürnberg
  ERCOSek Fa. ETAS GmbH (Bosch)

6 Feldbusse

6.1 Einführung

6.1.1 Begriffe

Arbitrierung	Buszugriff, bei Multimastersystemen muß die Prioritätsfrage gelöst werden.
ARP **RARP**	**A**dress **R**esolution **P**rotocol (für Adress-Management) **R**everse **ARP** (Umgekehrte Funktionsrichtung)
Broadcast	Signal geht an alle Busteilnehmer
Multicast	Signal geht an eine Busteilnehmer-Gruppe
Repeater	Signalverstärker zur Erhöhung der Bus-Reichweite
Transceiver	Anpassungsschaltung, welche die Sende- und Empfangs-signale des CAN-Controllers in ein Differenzspannungssignal des Busses und umgehrt wandelt.
Bridge	Verb.-Adapter zw. lokalen Netzen mit gleicher OSI Schicht 2
Gateway	Verb.-Adapter zur Kopplung von Netzen untersch. Architektur
Router	Bestimmt Leitweg v. Datenpaketen, arbeitet auf OSI Schicht 3
Token	Im System existiert genau 1 Token. Der Knoten, der den Token besitzt, hat das Recht, auf das Komm.-Medium zuzugreifen und zu kommunizieren. Nach Beendigung der Kommunikation reicht dieser Knoten den Token an seinen logischen Nachbarn weiter.
Tokenring	Durch das Weiterreichen des Tokens an den logischen Nachbarknoten entsteht aus logischer Sicht die Struktur einer virtuellen Ring –Technologie.
ISO	**I**nternational **S**tandards **O**rganisation (Institution der **UNO**)
OSI	Basic Reference Model for **O**pen **S**ystems **I**nterconnection
CCITT	**C**omité **C**onsultatif **I**nternat. **T**élégraphique et **T**éléphonique
Node	Netzknoten
Remote Frame	Ein Knoten fordert eine Botschaft an, indem er an den Lieferknoten einen Remote Request Frame sendet, welcher den Identifier, jedoch keine Daten enthält.
Hamming-Distanz	Maß für die Störfestigkeit eines Codes (benannt nach R. W. Hamming). Die Hamming-Distanz d ist die Anzahl der sicher erkennbaren Fehler + 1. Bei Verwendung des Paritätsbits ist z.B. ein Fehler sicher erkennbar, also ist die Hammingdistanz d = 2. Übliche Werte von d sind 2 – 6.
Bitfehlerrate p	p = Anzahl der fehlenden Bits / Anzahl der gesendeten Bits, normal ist $p = 10^{-4}$
CRC	**C**yclic **R**edundancy **C**heck Man fasst die Information unabhängig von ihrer Länge als Zahl auf. Diese wird durch eine andere feste Zahl, das sogenannte Generatorpolynom, im Sender dividiert. Der Quotient bleibt unbeachtet, den Rest hängt man beim Senden an die Information an und erhält so den Codevektor, den der Empfänger durch dasselbe Generatorpolynom dividiert. Ist der Rest 0, war die Übertragung fehlerfrei.

Wozu Feldbus?	**Siemens-Folien zu Feldbussen**

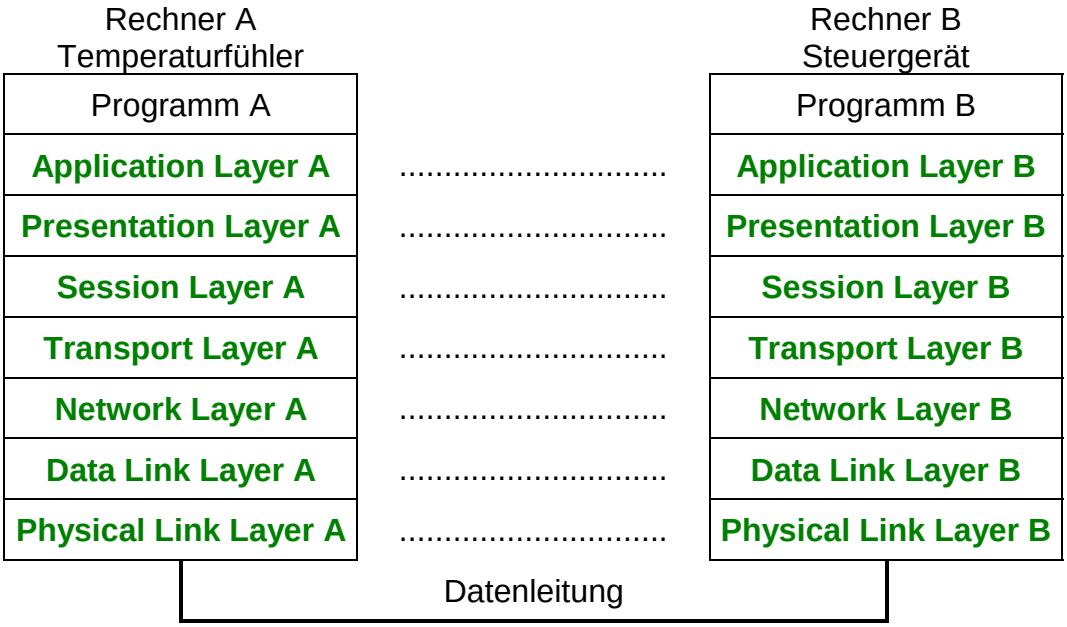
6.1.2 Das ISO/OSI Referenzmodell (Schichtenmodell)

- Das **ISO/OSI Referenzmodell** wurde 1983 zur Vereinbarung der Modalitäten beim Datenaustausch genormt (ISO 7498, CCITT X.200).
- Es teilt die Kommunikation in **7 abstrakte Schichten** mit festgelegter Funktionalität.
- Jeder Kommunikationspartner besitzt alle 7 Schichten.
- Die Schichten kommunizieren untereinander über definierte **Schnittstellen**. An diesen Schnittstellen stellt jede Schicht Dienste bereit, die von den Nachbarschichten benutzt werden können.
- Die **Implementierung** der Schichten ist nicht beschrieben (offenes System).

Nr.	Bezeichnung	Beschreibung
7	**Application Layer** (Anwendung)	Definiert Funktionen, mit denen ein Computer-Programm **auf das Kommunikationssystem zugreifen** kann. Sie stellt dem Anwender z.B. Dienste für das Lesen oder Schreiben von Daten zur Verfügung. Sie regelt jedoch nicht die technischen Details wie Paketierung usw.
6	**Presentation Layer** (Datendarstellung)	Hier wird das Datenformat des Rechners in eine für das Netz verständliche, **rechnerunabhängige Syntax** umgesetzt. Hierzu gehören auch Umcodierung, Ver- und Entschlüsselung und Datenkomprimierung.
5	**Session Layer** (Komm.-Steuerung)	Organisiert die **Synchronisation der Prozesse** (Tasks) auf Sende- und Empfangsebene. Sie umfasst Funktionen wie Verbindungsaufbau, Pufferspeicher-Verwaltung, Prüfung von Zugangsberechtigungen und Überwachung von Verbindungen.
4	**Transport Layer** (Transport)	Trennt die technologieabhängigen unteren Schichten von den oberen. Die Aufgaben sind u.a. Steuerung der **Übertragung des Datensatzes**, Umwandlung von logischer in physikalische Adresse, Einteilung eines Datensatzes in Pakete, Rekonstruktion des Datensatzes aus Paketen, hinzufügen der Paketnummer, ordnen ankommender Pakete nach Nummern, Nachforderung fehlender oder falscher Pakete.
3	**Network Layer** (Vermittlung)	Ist verantwortlich für den ordnungsgemäßen **Transport der Pakete**. Dies ist z.B. wichtig, wenn die Pakete über mehrere unterschiedliche Transportwege laufen.
2	**Data Link Layer** (Verb.-Sicherung)	**Steuerung und Schutz der Kommunikation** auf der Ebene des Botschaftsrahmens (Frame). Jedes Paket wird hierzu mit einem Frame versehen, der Zusatzinformationen enthält, die eine Fehlerprüfung erlauben, z.B. mit CRC oder Paritybit. In jedem Rahmen befindet sich eine maximale Zahl von Datenbytes.
1	**Physical Link Layer** (Bitübertragung)	Definiert die physikalischen (elektrischen und mechanischen) **Eigenschaften der Übertragung**: Art der Codierung, Spannungspegel, Zeitdauer eines Bit, Art der Übertragungsleitung, Stecker mit Pinbelegung.

- Die **Schichten 1-4** sind für die Datenübertragung zw. den Endgeräten zuständig.
- Die **Schichten 5-7** koordinieren das Zusammenwirken mit dem Anwenderprogramm und dem Betriebssystem.
- Bei der Kommunikation von zwei Rechnern kommunizieren jeweils gleiche Schichten miteinander. Die beiden Schichten 1 sind über die Datenleitung verbunden.
- Ein Kommunikationsmodell besitzt alle Schichten, **einzelne Schichten können aber leer sein**. So werden z.B. beim CAN-Bus die Schichten 3-6 nicht benötigt (sind leer).

Ablaufbeispiel einer Kommunikation im OSI-Modell:



Die Verbindung zwischen den Stationen A und B sei bereits aufgebaut.
Das Programm des Steuergeräts mit Rechner B möchte einen neuen Temperatur-Messwert bei Rechner A abfragen.

- **Application Layer B** erhält von Programm B die Anweisung, den Messwert von Rechner A anzufordern. Sie bearbeitet den Auftrag und gibt ihn als Daten an die darunterliegende Schicht weiter.
- **Presentation Layer B** wandelt die Daten in die für die Übertragung vereinbarte Form und gibt diese an die darunterliegende Schicht weiter.
- **Session Layer B** fügt die Identifikationsdaten hinzu und gibt die Daten weiter.
- **Transport Layer B** nimmt die Paketierung vor mit entspr. Informationen.
- **Network Layer B** fügt die Versandinformationen hinzu (Routing).
- **Data Link Layer B** bildet die Frames und fügt die Sicherungsinformationen hinzu.
- **Physical Link Layer B** gibt die gesamte Information über die Leitung an Station A.
- **Physical Link Layer A** empfängt die Information.
- **Data Link Layer A** überprüft die Korrektheit der Übertragung, entfernt die Sicherungsinformation und gibt die restl. Daten an die darüberliegende Schicht.
- **. . . Application Layer A** erkennt den Auftrag und reicht ihn an Programm A zur Bearbeitung weiter.
- **Programm A** gibt den Messwert an **Application Layer A.** Dieser wandert nun umgekehrt durch alle Schichten zum Programm B und wird dort verarbeitet.

6.1.3 Koppelelemente zwischen Netzen

Zur Kommunikation zwischen Netzwerken (auch mit versch. Protokollen) werden als Bindeglieder **Repeater, Bridges, Router und Gateways** eingesetzt. Ihren Einsatz erkennt man am OSI-Schichtenmodell:

Application	**Gateway:** zum Datenaustausch zw. Rechnern mit inkompatiblen Protokollen, z.B. Maschinen u. Netz	Application	7.
Presentation		**Presentation**	**6.**
Session		**Session**	**5.**
Transport		**Transport**	**4.**
Network	**Router:** Vermittlungsrechner, entscheidet über den Weg des Packet (dch. Tabellen oder Algorithmen)	Network	3.
Data Link	**Bridge:** Steuerung der Weitergabe von Packets, Protokollumformung zwischen versch. Netzarten	Data Link	2.
Physical Link	**Repeater:** Nur Verstärkung und/oder Umsetzung auf andere Übertr.-Medien, keine Streamänderung	Physical Link	1.

6.1.4 Vergleichende Übersicht

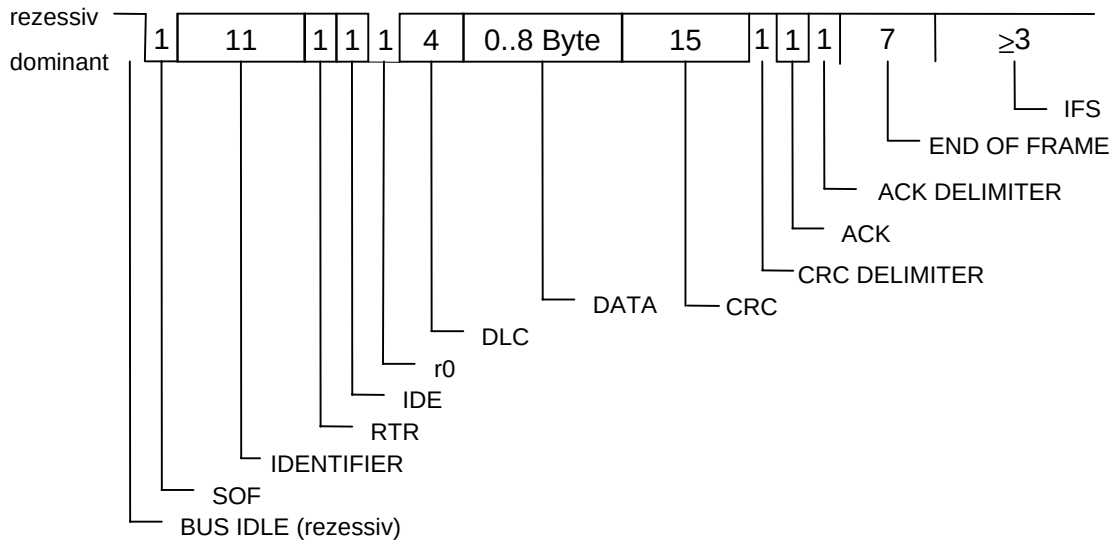
	CAN-Bus	**Ethernet + TCP/IP**	**Profibus**
Hintergrund	für KFZ entwickelt v. Bosch und Intel, zuverlässig, billig	als LAN verbreitet, zunehmend als Feldbus aktuell	für Prozeßautom., BMFT-Proj.v.Bosch, Klöckner, Siemens
Physik. Struktur	Linientopologie, Zweidraht verdrillt	Linie+Baum+Stern, LWL+Koax+verdrillt	Linientopologie, Zweidraht verdrillt
Protokolleigensch.	Multi-Master, flexible Arbitrierung	Multi-Master	Multi-Master, Multi-Slave
Zugriffsverfahren	CSMA / CA (Collision-Avoidance)	CSMA / CD (Collision-Detection)	
Marktsituation	führend bei Auto u. int. Vernetzung von Automat.-Systemen	weltweit verbreitet	zögerliche Verbreitung, teuer
Maximalausdehng.	1 km	500 m / Segment	4,8 km
Übertragungsrate	max. 1 Mbit/s	10/100/1000 Mbit/s	9,6 ... 500 kBit/s
max. Latenzzeit	160 µs	~10 ms (10% Last)	3,2 ms
Prioritäten	2032	keine	2
Datenlänge	0 ... 8 Bytes	46 ... 1500 Bytes	0 ... 246 Bytes
Broad-/Multicast	ja	ja	ja
Standard	ISO 11898 EN50xxx	IEEE 802.3 EN50xxx	DIN 19245 EN50xxx
Vereinigungen	**ISO / OSI** Standard, **CANOpen**Standard, **CiA** (CAN in Automation)	**IAONA** Ind. Autom. Open Netw.Alliance **IDA** Interface for Distributed Autom.	**ISO / OSI** Standard
Ursprung	Bosch, Intel	DEC, Intel, Xerox	Siemens
Einsatzgebiete	Fahrzeuge, Medizintechnik, Flugzeuge, Automatis., Gebäude	Inhouse LANs, Automatisierung (In Zukunft bis auf die Sensorebene)	Automatisierung, Verfahrenstechnik

6.2 CAN Controller Area Network

6.2.1 Grundlagen

CAN-Bus **Siemens-Folien zum CAN-Bus**

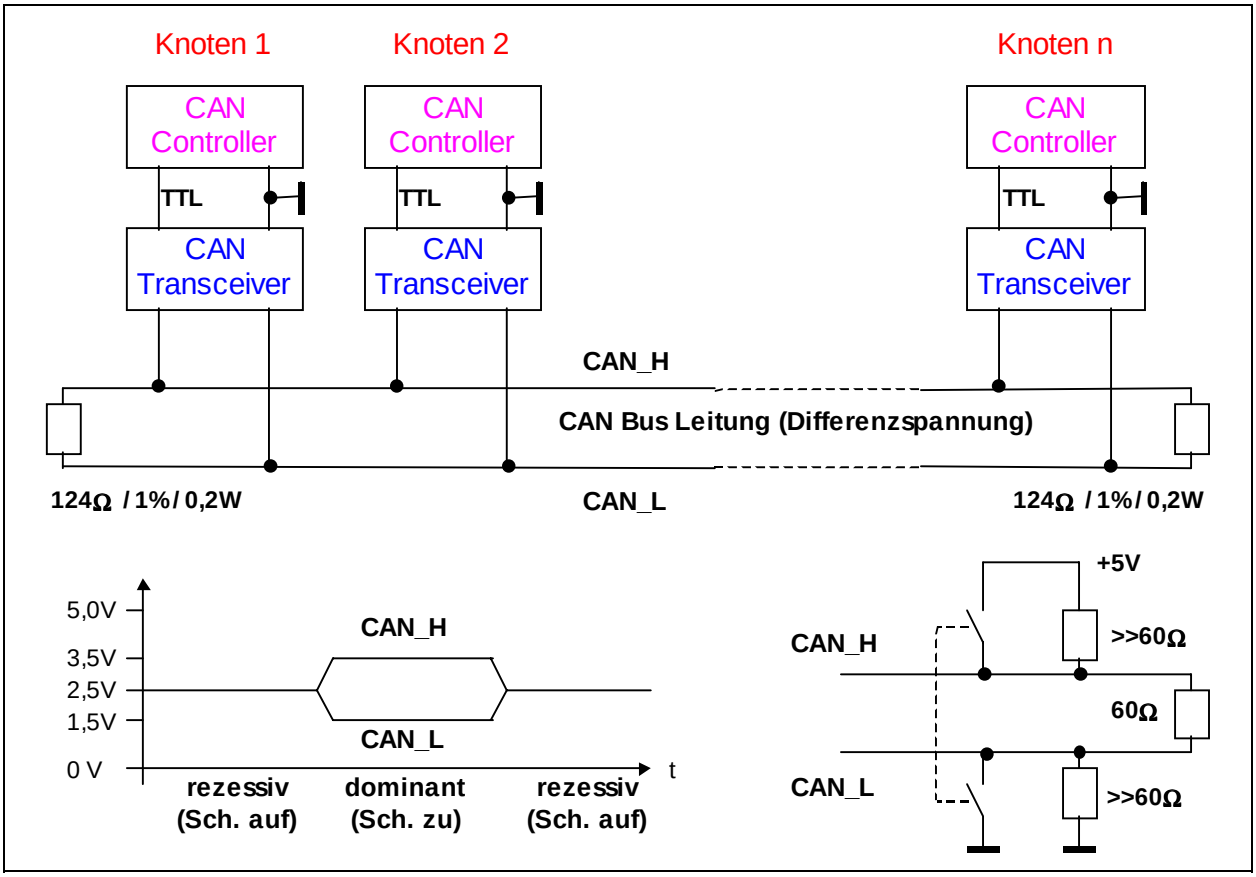
6.2.1.1 Telegrammaufbau (Standard-Format, CAN 2.0A)



Bus Idle	rezessiver Buspegel, CAN-Knoten können Telegramme senden
SOF	**S**tart **O**f **F**rame, einzelnes dominantes Bit, kennzeichnet den Telegrammstart
Identifier	11 Bits, die die Arbitrierung (= Buszugriff) steuern. Der Identifier ist Telegrammname einschliesslich Prioritätsangabe. Kleinster Identifier bedeutet grösste Priorität.
RTR	**R**emote **T**ransmission **R**equest, **dominant in DATA-Frames, rezessiv in REMOTE-Frames**. (Remote Frames enthalten kein Datenfeld, sie werden gesendet, um einen anderen Knoten zur Sendung von Daten zu bewegen)
SRR (2.0B)	**S**ubstitute **R**emote **R**equest Bit, Dummybit für RTR
IDE	**ID**entifier **E**xtended Bit, ist beim Standard-Format dominant, beim Extended-Format rezessiv
r0, r1 (2.0B)	einzelnes dominantes Bit für zukünftige Erweiterungen
DLC	**D**ata **L**ength **C**ode, gibt an, wieviele Datenbytes folgen
DATA	Datenfeld, bis zu acht Byte Daten pro Frame
CRC	**C**yclic **R**edundancy **C**heck, enthält Prüfsumme zur Fehlererkennung (nicht Fehlerkorrektur!)
CRC Delimiter	**C**yclic **R**edundancy **C**heck **D**elimiter, einzelnes rezessives Bit
ACK	**ACK**nowledge Slot, rezessiv, wird vom Empfangsknoten als Quittung dominant überschrieben
ACK Delimiter	**ACK**nowledge **D**elimiter, einzelnes rezessives Bit
EOF	**E**nd **O**f **F**rame, rezessive Bits geben Ende des Frame an
IFS	**I**nter **F**rame **S**pace, Mindestabst. zwischen Telegrammen

CAN 2.0B	zum Vergleich das Extended Protocol CAN 2.0B

6.2.1.2 Physikalische CAN-Schnittstelle nach ISO 11898



Signalfestlegung:

Empänger: soll rezessiv erkennen, wenn $CAN\_H \leq CAN\_L + 0{,}5\ V$

soll dominant erkennen, wenn $CAN\_H \geq CAN\_L + 0{,}9\ V$

Sender: an 60 Ω soll eine Differenzspannung von 1,5 ... 3 V erzeugt werden

6.2.1.3 Eigenschaften

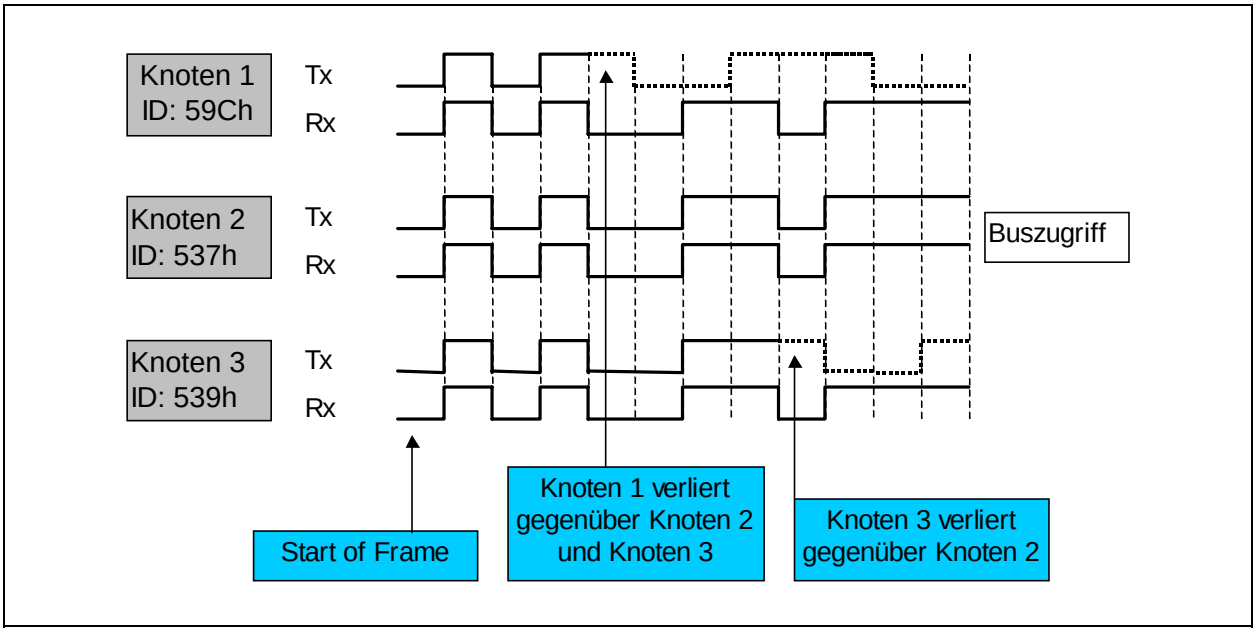
- **Priorisierung** der Botschaften mit dem Identifier
- Garantie von **Latenzzeiten** in der höchsten Prioritätsstufe
- **Multimasterfähigkeit**
- **Fehler**erkennung und –Signalisierung (keine Beseitigung!)
- Automatisch wiederholtes Senden beschädigter Botschaften
- selbständiges **Abkoppeln** einzelner Knoten bei permanenten Fehlern
- Ein Knoten kann mehrere Botschaften empfangen oder auch senden. Damit wird deutlich, daß CAN nicht knotenorientiert arbeitet, sondern nachrichten- bzw. **objektorientiert**. Die Auswahl der Botschaften erfolgt durch '**Akzeptanzfilter**'.
- Die Aufgaben der zweiten Schicht (Data Link Layer) sind in Silizium gegossen und vom Anwender nicht zu implementieren.
- Die physikalischen Eigenschaften der ersten Netzwerkschicht sind vom Anwender zu wählen, ihm werden versch. Zweidrahttechniken bereitgestellt, z.B. von CIA.
- **CIA** = **C**AN **I**n **A**utomation ist eine Vereinigung von Interessengruppen aus Industrie und Forschung. Sie schuf den Standard **CANopen** mit Anwendungsprofilen, die dem Anwender zur Verfügung gestellt werden. Damit wird eine Normierung in CAN-Anwendungen angestrebt, z.B. die Festlegung der Identifier für Gerätegruppen, um den Austausch versch. Fabrikate (Zulieferer) zu ermöglichen.

6.2.1.4 Arbitrierung

Der Buszugriff eines CAN-Knotens vollzieht sich zur Vermeidung von **Datenkollisionen** wegen der Mulitmasterfähigkeit nach dem Prinzip des **CSMA/CA**.

CSMA /CA	**C**arrier **S**ense **M**ultiple **A**ccess with **C**ollision **A**voidance. **Carrier Sense** bedeutet, daß nach dem „Listen-while-talk“-Prinzip jeder Knoten beim Versuch des Sendens auch den Buszustand bitweise „abhört“, also auch prüft, ob sein eigener Pegel durchkommt. **Multiple Access** besagt, dass mehrere Busteilnehmer gleichzeitig den Bus arbitrieren können und so gleichzeitig die aktuelle Botschaft empfangen. Auf Sendung bleibt aber nur der Teilnehmer mit der höchsten Priorität. **Collision Avoidance** ist so zu verstehen, daß kein Datenverlust entsteht, wenn mehrere Teilnehmer um Arbitrierung ringen. Unterbrochene Sendeknoten wiederholen ihr Telegramm, wenn der Bus wieder frei (idle) ist.

In folgender **Abb.** wird d. gleichzeitige (synchr.) Buszugriff mehrerer Knoten gezeigt. **Knoten 1 bis 3** fordern den Bus an und senden dazu auf ihrer Transmit-Leitung Tx gleichzeitig ihren Identifier. Sobald ein dominantes Bit auf ein rezessives stößt, wird das rezessive Bit überschrieben. Der Knoten mit dem rezessiven Bit erkennt dies als Signalfehler auf der Leitung Rx, stoppt seine Übertragung und wiederholt sein Telegramm später (bei **Bus Idle**).



Tx und Rx sind **TTL-Signale**, also massebezogen, im Gegensatz zu den Leitungen CAN_H und CAN_L des CAN-Busses. Die Signaltransformation zwischen beiden Signalarten besorgt der **CAN-Transceiver**.

6.2.1.5 Bit-Stuffing

Stuffbits	Einfügen und Entfernen von Stuffbits

Alle gleichzeitig sendenden Knoten werden aus **BUS IDLE** heraus mit der ersten fallenden Flanke (Startbit) **synchronisiert**. Da während der Übertragung nachsynchronisiert wird, dürfen bis zum Datenfeldende nicht mehr als 5 Bits gleich sein, da sonst die Synchronisierflanken zu lange ausbleiben. Deshalb wird nach 5 gleichen Bits ein inverses **Stuffbit** (Stopfbit) eingefügt. Beim Empfang werden die Stuffbits wieder entfernt. **Ausnahme:** Errorframes erhalten keine Stuffbits.

6.2.1.6 Remoteframe

- Ein Remoteframe wird gesendet, um einen anderen Knoten zur **Sendung eines Datenframe** zu bewegen, er enthält kein Datenfeld
- Er unterscheidet sich vom Datenframe durch das **rezessive RTR-Bit**
- Der Remoteframe hat den **Identifier** des Knotens, von dem er Daten anfordert
- Sendet dieser Knoten gleichzeitig Daten, wenn solche angefordert werden, treffen 2 gleiche Identifier aufeinander. Die Arbitrierung **gewinnt der Datenframe**, weil das RTR-Bit in die Arbitrierung einbezogen ist. **Eigentlich wird mit 12 bit arbitriert**

RTR	RemoteTransmission Request

6.2.1.7 Acknowledge und Akzeptanzfilterung

- Jeder Knoten hört **alle Telegramme** mit (Broadcast)
- Verläuft der CRC-Check positiv, wird die Botschaft dadurch quittiert, dass das rezessiv gesendete **ACK-Bit** dominant überschrieben wird
- Der Sender verzichtet daraufhin auf die **Telegrammwiederholung**
- Die Botschaft wird von dem Knoten quittiert, der ihre Fehlerfreiheit zuerst erkennt
- Der Sendeknoten erkennt an der Quittierung nicht, ob die Botschaft ihr **richtiges Ziel** erreicht hat
- Ein Knoten verwertet nur die Telegramme, deren Identifier in seiner **Akzeptanzliste** gespeichert sind
- Durch Undierung mit einer Akzeptanzmaske kann eine **Gruppe** von Identifiern selektiert werden

6.2.1.8 Übertragungssicherheit und Fehlerbehandlung

CAN hat durch sechs implementierte Mechanismen eine sehr geringe Fehlerwahrscheinlichkeit. Die Mechanismen sind im einzelnen:

Name	**Mechanismus**
Bus Monitoring	Vergleich der Bit-Pegel mit den Bus-Pegeln
Cyclic Redundancy Check	Division durch CRC-Generatorpolynom mit einem mathematischen Algorithmus
Message Frame Check	Prüfung, ob feste Frameelemente vorhanden sind
Acknowledgeprüfung	Sender verlangt von mindestens einem Knoten ein ACK
Prüfung der Bitcodierungsregel	Bit-Stuffing-Regel: nach **fünf gleichen** Bits innerhalb einer Botschaft wird auf der Senderseite automatisch ein Bit mit entgegengesetztem Pegel eingefügt und auf der Empfangsseite wieder entfernt. Bei mehr als 5 gleichen Bits liegt ein Fehler vor
Errorframe und Fehlerzähler (ISO-Standard in Silizium realisiert)	Der Knoten, der einen Fehler erkennt, sendet einen **Error-frame** aus, der bewusst die Kodierungsregeln verletzt, indem er **mehr als 5 dominante Bits** auf den Bus legt. Wegen Verletzung der Stuffregel enden alle Sendeversuche. Im sendenden CAN-Chip zählt ein Zähler die Fehler. Findet ein Überlauf statt, wird der Knoten abgekoppelt (Busoff), um die Blockade des Busses durch Telegrammwiederholungen zu vermeiden. Durch Fehlerbehandlungs-Regeln wird sichergestellt, dass der **Bus nicht blockiert** werden kann

6.2.2 CAN Laborbus

Um die Funktionen von CAN am Beispiel zu demonstrieren, wurde ein einfacher, funktionsfähiger **Laborbus** entwickelt.

Der **MTS-Sensor** erfasst den Weg von 2 Gleitern und sendet die Längeninformation über den CAN-Bus an das **Phytec-Board C167CR**. Dessen Controller rechnet die Daten in eine dezimale cm-Angabe um und sendet diese über den CAN-Bus an den **SLIO-Modul** zur 3-stelligen 7-Segment-Anzeige. Zusätzlich wird der Bus durch einen nicht dargestellten PC mit **CAN-Dongle** beobachtet, der als Busmonitor wirkt. Zunächst werden die Buskomponenten kurz beschrieben.

6.2.2.1 PCAN-Dongle

Der PCAN-Dongle der Firma Phytec ermöglicht den Anschluß eines CAN-Bus an die **parallele Schnittstelle** eines IBM kompatiblen PC. Zusammen mit dem MS-DOS Programm **`CANView.exe`** ist es möglich, den Datenverkehr auf dem CAN-Bus darzustellen und zu überwachen. Durch die mitgelieferten Libraries kann man auch **eigene Applikationen** schreiben, die das Bearbeiten von CAN-Nachrichten ermöglichen. Wichtigste Eigenschaften:

- verwendet den **82C200** als CAN Protokoll Controller
- verwendet den **82C250** als Transceiver-Baustein
- Anschluß an den PC über **parallele Schnittstelle**
- Anschluß an den CAN-Bus über **9-poligen SUB-D** Stecker
- **11-Bit Identifier** (CAN2.0A), verhält sich aber CAN2.0B passiv
- maximale Baudrate **1MBit/s**
- Betrieb von **`CANView.exe`** unter DOS ratsam, nicht im DOS-Fenster!
- mit 2 Dongles kann ein einfacher **Demo**-Busbetrieb aufgebaut werden

6.2.2.2 SLIO (Serial Linked IO) Module

Der **SLIO** stellt **16 Bit I/O** zur Verfügung und besitzt keine eigene Intelligenz.
Die empfangenen Daten werden direkt von der **Empfangs-Mailbox** auf den Port geschrieben. Eingelesene Ports werden direkt in die **Sendemailbox** geschrieben und können dann per **Remote-Request** über den CAN-Bus abgefragt werden. Sie können jedoch auch **ereignisgesteuert** vom SLIO selbständig ausgesendet werden. Solch ein Ereignis kann z.B. ein Flankenwechsel an einem Port sein. Eigenschaften:

- **82C150** I/O device mit CAN Protokoll Controller
- **82C250** als CAN-Treiber
- **11-Bit Identifier** (CAN2.0A), verhält sich aber CAN2.0B passiv
- maximale Baudrate **125KBit/s**
- 16 konfigurierbare I/O Ports
- enthält ein **A/D**- und zwei **D/A-Wandler** mit 10 Bit Auflösung
- Festlegung des **Identifiers** durch 4 Ports (max. 16 SLIOs), Rest intern festgelegt

0	1	P3	1	0	P2	P1	P0	1	0	Dir

Dir = 0: Empfangs-Identifier, Dir = 1: Sende-Identifier
also Empfangs-ID = Sende-ID - 1

- **Baudraten-Kalibrierung** über CAN-Bus durch eine **Calibration Message**, Refresh nach max. jeweils 4000 Bits erforderlich
- Nach erfolgreicher Kalibrierung sendet der SLIO einen Datenframe mit ID + Status

Input/Output-Funktionen des Slio:
Der **82C150** auf dem SLIO-Modul stellt 16 Ports zu Verfügung. Die Konfiguration, das individuelle Lesen und Beschreiben der Ports erfolgt über **neun 16-bit-Register**, welche über **4 Register Marker (RM)** adressiert werden. Diese Register können über Datenframes gelesen oder beschrieben werden und beeinfl. die Ports unterschiedlich. Die **Abbildung** zeigt den Aufbau eines Datenframes, mit dem man die Register lesen und beschreiben kann.

Beispiele der **Registerfunktionen**, wie sie beim Laborbus verwendet werden:

`input data register, RM 0x00, read only`
- Enthält die logischen Zustände der Portpins P15-P0, welche per *Datenframe* abgefragt werden können. Der SLIO sendet dann einen Datenframe mit dem Status der Portpins

`positive edge register, RM 0x01, write only`
- Konfiguriert die Event Capture Eigenschaft bei positiver Flanke am jeweiligen Portpin

`negative edge register, RM 0x02, write only`
- Konfiguriert die Event Capture Eigenschaft bei negativer Flanke am jeweiligen Portpin

`output data register, RM 0x03, write only`
- In dieses Register werden die logischen Zustände, die an die Portpins ausgegeben werden sollen, geschrieben. Zum Ausgeben der Zustände an die Portpins muß noch das Output Enable Register gesetzt werden

`output enable register, RM 0x04, write only`
- Damit werden Werte, die im output data register stehen, auf die Outputpins geschaltet

SlioCalib SlioStart	**Programmbeisp. z. Kalibrierung + Initialisierung des SLIO**
Schaltung	**Schaltbild der Anzeige-Einheit**

6.2.2.3 MTS-Positionssensor (www.mtssensor.com/de)

Bei diesem **Sensor** handelt es sich um ein absolutes **Längenmeßsystem**.
Die Positionserfassung beruht dabei auf einer magnetorestriktiven Laufzeitmessung. Der Sensors erfaßt die **Position eines Dauermagneten** und wandelt diesen Wert in ein wegproportionales und busfähiges Ausgangssignal um, das dann direkt über den Bus gesendet werden kann. Eigenschaften:
- **82C251** als Transceiver Baustein, **82C200** CAN Protokoll Controller
- **11-Bit Identifier** (CAN2.0A), verhält sich aber CAN2.0B passiv
- Meßlänge: 300 mm, maximale **Auflösung 2 µm**
- Meßgrößen: **Position und Geschwindigkeit** max. Baudrate **1MBit/s**
- die **Programmierung** des Sensors erfolgt mit Datenframe (ID = 0x7e5), die **Quittierung** erfolgt durch Antwortframe mit denselben Daten (ID = 0x7e6)
- zur Unterscheidung mehrerer Sensoren erhält jeder zusätzlich zum ID einen frei wählbaren **Node-ID** (1 Byte), bei dessen Programmierung die Serien-Nummer als Adresse verwendet wird
- wenn der Node-ID (im Datenbyte 0) zu beachten ist, lautet der Identifier 0x7ea
- folgende Param. werden über den Bus durch Steuer-Datenframes programmiert:
  - **4 Baudraten** 125 / 250 / 500 / 1000 kBit/s (wird nach Restart wirksam)
  - **Positionsidentifier** für die spätere Übertragung der Positionsdaten
  - **Broadcastidentifier** für Start/Stop des zykl. Sendens der Positionsdaten
  - **Samplingperiode** für das zyklische Senden der Positionsdaten
  - **Protokollformat** 'M' (metrisch) oder 'I' (inkremental)
- Der Positionsdatenframe hat 7 Datenbyte für die Pos. beider Magnete + Status

Init_Mts	**Programmbeispiel zur Initialisierung des MTS-Sensors**

6.2.2.4 Phytec miniModul-167

6.2.2.4.1 Beschreibung des Moduls

Das **Rapid Development Kit** (RAD) enthält den 16-bit Controller 80C167CR von Infineon mit einem Onchip CAN Controller. Der Controller 80C166 mit dem gleichen Prozessorkern wird im Wahlfach Mikrocontrollertechnik ausführlich behandelt. Hier wird lediglich die Erweiterung um den CAN-Modul besprochen.

167CR	**Die Funktionsblöcke des 80C167CR**
Schnittstelle	**Physikalische CAN-Schnittstelle**

Eigenschaften des CAN-Moduls:

- **82C527** als CAN-Modul
- Der **CAN Transceiver** befindet sich nicht auf dem Chip, sondern auf dem Board
- automatisches **Senden und Empfangen** von Datenframes
- unterstützt **11-Bit und 27-Bit Identifier** (CAN2.0A und CAN2.0B)
- **Full-CAN:** 14 Message-Objekte mit Akzeptanzfiltern, unabhängig konfigurierbar
- **Basic-CAN:** Message-Objekt 15 ohne Akzeptanzfilter
- Jedes dieser 15 Objekte hat seinen eindeutigen **Identifier**
- Jedes Botschaftsobjekt besitzt sein eigenes 16-bit **Control/Status-Register**
- Jedes Objekt kann für die Richtung **Senden oder Empfang** konfiguriert werden ausser Objekt 15, das ein Empfanspuffer mit speziellem Maskenregister ist.
- Ein Objekt mit Richtung ‘Senden’ kann für automatische Antwort auf einen **Remote Frame** konfiguriert werden
- Jedes Objekt besitzt getrennte **Sende- und Empfangsinterrupts** und Statusbits, um der CPU volle Flexibilität zum Erkennen eines gesendeten oder empfangenen Remote/Data Frames zu geben
- Die **Botschaftsspeicherung** ist in einem intelligenten Speicher implementiert, welcher vom CAN Controller und vom CPU Interface adressiert werden kann
- Alle Register und Botschaftsobjekte des CAN Controllers liegen im speziellen CAN **Adressbereich von 256 Byte** im Segment 0, Adr. EF00h bis EFFFh (Bereich für externen Speicher)
- Alle **Register** sind 16 Bit lang und können auch byteweise adressiert werden
- die **Interrupts** des CAN-Moduls sind in die Interruptstruktur des C167 integriert und werden wie diese über reservierte Interruptvektoren gesteuert

CAN-Controller	**Der CAN Controller**
Adress-Map	**Organisation von Registern und Message Objekten**
Message-Objekt	**Das Message-Objekt**

Belegung des Controller Control/Status-Registers (Adr. EF00H):

INIT	startet die Initialisierung des CAN-Controllers
IE	Interruptgenerierung des CAN-Modul an die CPU freigeben
SIE	Interruptgenerierung nach erfolgreichem Nachrichtentransfer freigeben
EIE	Fehler Interrupt freigeben
CCE	CPU-Zugriff auf das Bit-Takt-Register freigeben
Bit7	Prüfbit, muß beim Beschreiben des Control-Registers auf 0 gesetzt werden
LEC	letzter Fehler Code
TXOK	Zeigt an, daß eine Nachricht erfolgreich übertragen wurde
RXOK	Zeigt an, daß eine Nachricht erfolgreich empfangen wurde
EWRN	Zeigt an, daß der Fehlerzähler einen Grenzwert erreicht hat
BOFF	Zeigt an, ob sich der CAN-Controller im BusOff Zustand befindet

6.2.2.4.2 Programmierung des Moduls

Vor Benutzung der **Message-Objekte** müssen diese initialisiert und konfig. werden:

Control Register (ef00h)	alle Maskenregister der Akzeptanzfilter
Bit Timing Register (ef04h)	alle Message Control Register
	alle Daten Register

Infineon liefert in der Bibliotheksdatei **CAN16X1.LIB** alle zum Einsatz von CAN nötigen Funktionen. Der Tasking-Compiler setzt diese automatisch ein, wenn im EDE-Menü die Auswahl 'Link CAN libraries' angewählt wird.

initialisieren des CAN Moduls	`init_can_16x(..)`
definieren eines MessageObject	`def_mo_16x(..)`
laden von Datenbytes in ein MessageObject	`ld_modata_16x(..)`
auslesen von Daten aus einem MessageObject	`rd_modata_16x(..)`
auslesen des Inhalts von MessageObject15 (MO15)	`rd_mo15_16x(..)`
senden eines MessageObject	`send_mo_16x(..)`
prüfen ob neue Daten in einem Message-Object vorliegen	`check_mo_16x(..)`
prüfen ob neue Daten o. Remoteframes im MO15 vorliegen	`check_mo15_16x(..)`
prüfen ob ein 'bus off' vorliegt	`check_busoff_16x(..)`

CAN-Funktionen	**Funktionsbeschreibungen der CAN-Bibliothek**
Prog	**Programmbeispiel**

6.2.2.5 Das CAN Laborbus Projekt

Komponenten des Busses:

- **PC-Dongle** zur Beobachtung des Busgeschehens (nicht im Bild)
- **MTS-Weggeber** mit eingebauter CAN-Schnittstelle
- **Anzeige mit SLIO** zur 3-stelligen 7-Segment Weganzeige des MTS-Weggebers
- **Phytec miniModul-167** als Busmaster und zur Umrechnung der Messwerte

Das Projekt ist mit dem **TASKING C-Compiler** bearbeitbar. Damit eignet sich das Projekt zum Experimentieren. Der Source-Code zeigt die Programmierung.

Projektdatei	C-Dateien	Kommentar
X1.pjt	X1main.c	Initialisierung der Busteilnehmer und Datenmanagement
	Sliostrt.c	Sendet Calibration Message und initialisiert SLIO
	Init_mts.c	Initialisiert den Sensor u. startet period. Datenausgabe
	Isr_01.c	Timer-Int. Progr. zur Wdh. der Cal. Mess. des SLIO
	Sende.c	angep. Funktion z. Senden von Datenframes von C167
	Blinken.c	univ. programmierb. Signalausgabe auf C167-Board

6.2.3 Konkurrenz für CAN ?

Markt & Technik Nr.14 / 1999:

TRW und Motorola entwickeln einen neuen Netzschnittstellen-Standard für die Automobilindustrie. Das **Distributed Systems Interface (DSI)** soll die Entwicklung und Installation von neuen Insassenschutz-Systemen mit mehreren Rückhalte-einrichtungen erleichtern und beschleunigen und wird für das Modelljahr 2002 verfügbar sein.
Es handelt sich um einen Zweidrahtbus für Sensoren, Aktoren und Steuerung, der für Kommunikation und Stromversorgung genutzt wird. Es können Komponenten hinzugefügt werden, ohne das gesamte System verändern zu müssen.

6.3 Ethernet und TCP / IP

6.3.1 Begriffe

Client	Ein Programm, das Daten von einem Server empfängt, z.B. ein HTTP-Browser
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol. Methode automatischer Vergabe von festen oder dynamischen IP-Adressen an Clients
DNS	**D**omain **N**ame **S**ystem. Protokoll zur Auflösung von Host-Namen in IP-Adressen. Die Datenbank hierzu wird von einem DNS-Server verwaltet
FTP	**F**ile **T**ransfer **P**rotocol. Ein Client/Server-Protokoll, das zur Übermittlung von Dateien über TCP/IP dient
HTML	**H**yper **T**ext **M**arkup **L**anguage. Formatierungssprache im www
HTTP(S)	**H**yper **T**ext **T**ransfer **P**rotocol (**S**ecure). Client/Server-Protokoll zum Austausch von HTML-Dokumenten im www
IP-Adresse	Numerische Adresse zur Identifizierung von Rechnern in einem TCP/IP Netz. Ziffernblöcke sind durch Punkte getrennt.
NFS	**N**etwork **F**ile **S**ystem
Port	Spezifiziert den Dienst auf dem TCP/IP Zielrechner. *Well known ports:* Port 80 für HTTP und Ports 20 u. 21 für ftp. Die TCP/IP Partner sind durch IP-Adr. und Port-Nr. festgelegt.
PPP	**P**oint-to-**P**oint-**P**rotokoll. Kommunikationsmethode zwischen TCP/IP Partnern (meist über DFÜ, z.B. bei Einwahlknoten von Internet-Providern).
Router	Rechner zur Vermittlung von Datenpaketen zwischen zwei IP-Teilnehmern.
Server	Rechner, der einem Client Daten liefert. Ein Rechner wird zum Server (z.B HTTP- oder ftp-Server) durch Betrieb einer Server-Software.
Shell	Rechner, der ständig mit dem Internet verbunden ist. Über Telnet kann der Benutzer darauf zugreifen.
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
Socket	Mechanismus für virtuelle Verbindungen zwischen Prozessen.
TCP/IP	**T**ransmission **C**ontrol **P**rogram / **I**nternet **P**rotokoll
Telnet	Internet Standard-Protokoll zum Einloggen auf entfernten Rechnern (Remote Login). Benutzt wird TCP/IP mit erweiterten Optionen.
URL	**U**niform **R**esearch **L**ocator. Eindeutige Adresse eines Dokuments oder einer Datei im WWW.
WWW	**W**orld **W**ide **W**eb. Internetdienst zur Bereitstellung von verlinkten Hypertextdokumenten. Entwickelt vom CERN in Genf

6.3.2 Ethernet-Grundlagen

- Ethernet ist ein **logischer Bus**: die sendende Station wird von allen andern gehört.
- Jede Empfangsstation entscheidet aufgrund der **Zieladresse** im Telegramm, ob sie dieses annimmt. Multi- u. Broadcast-Telegramme werden immer angenommen.
- **Fehlerprüfung**: Fehlerhafte Telegramme werden ignoriert.
- **Quittierung** der Telgramm-Annahme findet nicht statt (Aufgabe von TCP/IP)
- Alle Stationen sind vollkommen **unabhängig** voneinander, sie werden von keinem Busmaster oder Token synchronisiert.
- Eine Station sendet nur, wenn der **Bus mindestens 9,6 µs frei** ist.
- Bei **Kollision** von sendenden Stationen (gleichzeitiger Beginn), brechen alle sendenden Stationen die Übertragung ab (keine Prioritätssteuerung wie bei CAN) → Collision Detection, nicht Collision Avoidance!
- Die Station, die die Kollision zuerst erkennt, sendet das Kollisionssignal (**Jamming Burst**) mit 4 bis 6 Bytes FFH, das die anderen Sender stoppt.
- Durch zufällig generierte gestaffelte **Wartezeiten** für die sendewilligen Stationen bleibt es dem Zufall überlassen, wer als nächstes sendet, evtl. eine neue Station.
- Durch Kollisionen entstehen **Ausfallzeiten** in der Busnutzung.

6.3.3 Ethernet Packet

Die auf dem Bus ausgetauschten **Telegramme** haben folgendes Format in Bytes:

Ethernet Packet							
Ethernet Header					Ethernet Data		FCS
Preamble	Start Frame Delimiter	Destin. Address	Source Address	Length	Data Bytes	Padding (Erg.-Zeichen)	Frame Check Sequence
PRE	SFD	DA	SA	LEN	DATA	PAD	FCS
7	1	6	6	2	0 ... 1500	0 ... 46	4

DATA + PAD = 46 bis 1500 Byte
PRE = 10101010 ..... binär dient zur Synchronisation
SFD = 10101011b
FCS = Polynom-Prüfsumme aus DA bis PAD.
Bei Unstimmigkeit von FCS wird das Packet ignoriert.

6.3.4 Ethernet Adresse

Weltweit **eindeutige Adresse** der Teiln.-Station, meist im ROM d.Adapterkarte gesp.
Ihre Länge ist 48 Bit = 6 Bytes (2 Gruppen zu 3 Bytes).

Bit 47	Bit 46	Bit 45 – 24	Bit 23 – 0
Adress-Typ		Hersteller-ID wird vom IEEE vergeben	Adapter-Seriennummer wird vom Hersteller vergeben

Bedeutung der Adressbits in der Destination Adresse:

Bit 47 = 0	individuelle Geräteadresse
= 1	Gruppenadresse
Bit 46 = 0	Adresse wird global verwaltet (IEEE, weltweit eindeutig)
= 1	Adresse wird lokal verwaltet (privates Netz)

Bedeutung der Adressbits in der Source Adresse:

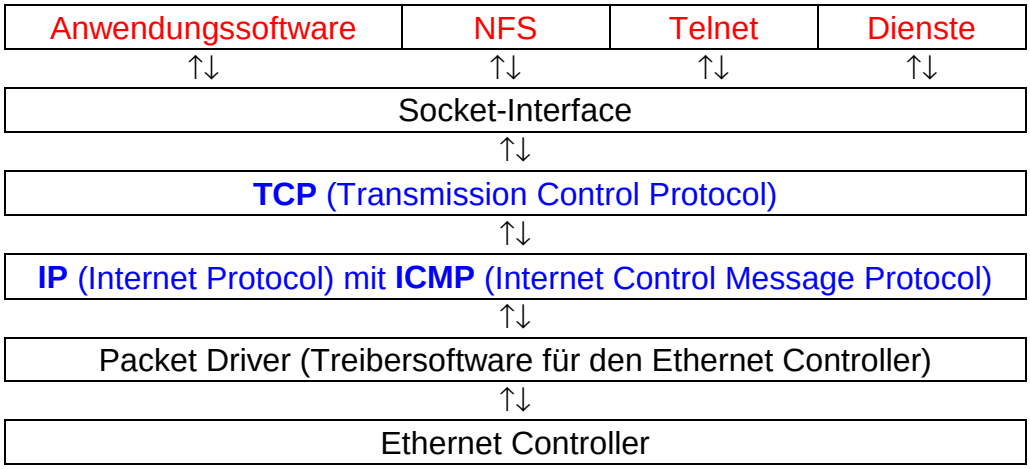
Bit 47	immer 0
Bit 46 = 0	Adresse wird global verwaltet (IEEE, weltweit eindeutig)
= 1	Adresse wird lokal verwaltet (privates Netz)

Broadcastadresse: alle 48 Bit sind auf 1: FF FF FF FF FF FFh

6.3.5 TCP / IP – Kommunikationssoftware

- Ethernet erledigt nur den **Transport von Packets** zwischen Sender und Empfänger, ohne Quittung und ohne Wiederholung von verlorenen Packets.
- Ethernet definiert **keine Anwenderfunktionen** (Dienste), d.h. keine Definition der Struktur und Verwendung der Datenbytes.
- Zur Implementierung zuverlässiger Übertragungseigenschaften und anwender-gerechter Funktionen benötigt man zusätzlich eine **übergeordnete Software**.
- TCP/IP ist eine der Lösungen, die durch ihre Verwendung im Internet besticht und deshalb zum beherrschenden **Standard** werden dürfte.

6.3.5.1 Struktur und Hierarchie von TCP / IP:



Einpassung in das OSI / ISO Schichtenmodell:

TCP / IP				OSI / ISO	
Anwendungs-Programme	Appl. Protocols und Services (Telnet,NFS,FTP..)		Application	7.	Application
				6.	Presentation
				5.	Session
TCP			Transport	4.	Transport
IP ICMP ARP RARP			Internet	3.	Network
Packet-Driver			Network Interface	2.	Data Link
Ethernet Controller			Hardware	1.	Physical Link

6.3.5.2 IP (Internet Protokoll)

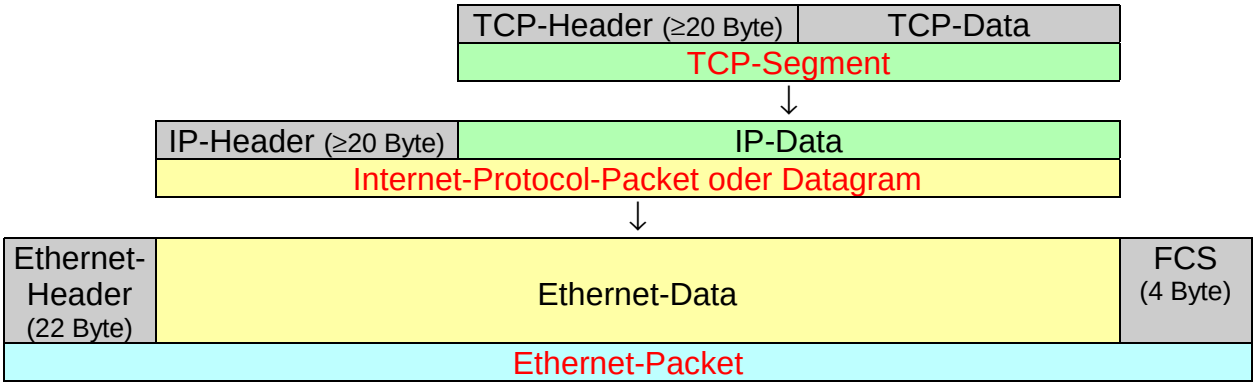
- IP **übermittelt Datagramme** vom Sender z. Empfänger im Netz oder zw. Netzen,
- passt die **Parameter** der Datagramme dem benutzten Netz an,
- ermöglicht den Routern, die **freien Pfade** über die Netze zu wählen,
- **garantiert weder** die Ankunft **noch** die Reihenfolge der Datagramme,
- macht die Netzwerkstruktur **transparent**,
- erledigt das Adressmanagement mit **ARP** (Adress Resolution Protocol),
- **segmentiert** die Datagramme (Aufteilung, Fragmentierung),
- übt **Netzwerkontrollfunktionen** aus.

6.3.5.3 TCP (Transmission Control Protocol)

- garantiert fehlerfreie, sequenzgerechte und vollst. **Übermittlung** v. Datagrammen,
- stellt transparenten **Vollduplex-Datenstrom** ohne Interpretation oder Veränderung des Dateninhalts in beide Richtungen bereit,
- steuert Auf- und Abbau der **Verbindung** zwischen den Kommunikationspartnern (verbindungsorientiertes Protokoll),
- überwacht die Verbindung: Meldung von **Fehlern** beim Transfer an die Anwendungssoftware, bei Abbruch der Verbindung oder Stau im Netz,
- steuert die Zwischenspeicherung der abgehenden Daten im **TCP/IP-Stack** zur Segmentierung und Absendung in unregelm. Abst. (Speicherung bis z. Quittierung),
- steuert die Zwischenspeicherung der ankommenden Daten im **TCP/IP-Stack** und das Zusammenfügen der Segmente,
- stellt **dynamische Ports** bereit als Schnittstelle zwischen TCP und der Anwendungssoftware. Die Portnummer wird beim Verbindungsaufbau zugeteilt.

6.3.5.4 Die TCP/IP-Protokollschichten

Die drei Protokollschichten sind wie folgt geschachtelt:



TCP-Segment:

	← 32 Bit →			
Byte 0	Source Port (Sender)			Destination Port (Empfänger)
Byte 4	Sequence Number (Bytepos. im Sende-Datenstrom)			
Byte 8	Acknowledgement Number (Bytepos. im Empf.-Datenstrom)			
Byte 12	Offset		Flags	Window (Datenpuffer im Empf.)
Byte 16	Checksum			Urgent Pointer (Überspr. v. Bytes)
Byte 20	optional: Options (Reserve)			
	Data			

Internet-Protocol-Packet:

	← 32 Bit →			
Byte 0	Version	Length	Service Type	Length (total)
Byte 4	Identification (Ang. f. Segment.)		Flags	Fragment Offset
Byte 8	Life-Time	Protocol		Header-Checksum
Byte 12	Internet Source Address			
Byte 16	Internet Destination Address			
Byte 20	optional: Options (Reserve)			
	Data			

6.3.5.5 Die IP-Adressierung (Internetadresse)

- Die IP-Adresse hat eine völlig **andere Struktur** wie die 48-Bit-Ethernetadresse.
- Zur Identifizierung der Rechner dient eine **32 Bit** = 4 Byte lange IP-Adresse.
- Sie besteht aus 4 durch Punkte getrennte Zahlen <256, z.B. **141.59.42.00** für den Nameserver der FHU (TCP-Port 42.xxx = Host Name Server, xxx = FHU-Rechner)
- Sie besteht aus einem Netz- und einem Rechneranteil, der in den **Klassen A – C** unterschiedlich aufgeteilt ist.
- Die Bitzahl bestimmt die maximale Anzahl von Netzen und Rechnern.
- Das eigene Netzwerk (kein Routing) hat die Netz-ID = 0.
- In Intranets ist die Adresszuweisung frei, im www dagegen weltweit festgelegt.
- Zur Anpassung an die Benutzerbedürfnisse gibt es drei Adresstypen A, B, C.

Klasse A (0.0.0.0 bis 127.255.255.255)

0	Netz-ID	Host-ID
	7 Bit 128 Netze mit je	**24 Bit 16 Mega Hosts**

Klasse B (128.0.0.0 bis 191.255.255.255)

1	0	Netz-ID	Host-ID
		14 Bit 16k Netze mit je	**16 Bit 64k Hosts**

Klasse C (192.0.0.0 bis 223.255.255.255)

1	1	0	Netz-ID	Host-ID
			21 Bit 2 Mega Netze mit je	**8 Bit 256 Hosts**

Broadcastadresse: alle 32 Bit sind auf 1: 255.255.255.255

- Da die IP-Adressen knapp werden, wird in der Version 6 des IP statt der 32-bit-Adresse ein **128-bit Indikator** eingeführt, der als Folge von 8 durch Doppelpunkte getrennte 16-bit Hex-Zahlen dargestellt wird.
- Die bisherigen IP-Adressen können damit auch dargestellt werden, z.B. **141.59.42.12** wird zu **::8D3B:2A0C**

6.3.6 "Predictable Ethernet"

- Der Zugriffsalgorithmus von Ethernet beruht auf einem **Zufallsalgorithmus** (Collision Detection, nicht -Avoidance). Deshalb ist Realzeitverhalten im Prinzip nicht möglich.
- Um Ethernet als Feldbus einsetzen zu können, muss die **Meldungsverzögerung** wenigstens durch besondere Auflagen eingegrenzt werden:
  1. Überlagerung eines **deterministischen** Anwendungsprotokolls
  2. Beschränkung der **Meldungsrate** (Lastbeschr.) zur Vermeidung von Netzstaus
  3. **Mindestabstand** zwischen Meldungen
  4. Begrenzung der **Meldungsdauer** (z.B. bei niedriger Priorität)
- Optimale Konfigurationen werden durch **Netzsimulation** gewonnen.
- Um Standardisierung solcher Feldbusattribute bemühen sich z.B. die folgenden **Organisationen:**

6.3.6.1 IAONA Industrial Automation Open Networking Alliance

Bisher: Profibus, Interbus und CAN-Bus im Feld,
Ethernet für Kommunikation zwischen intelligenten Einheiten.

Neu: Ethernet im Industrieumfeld bis auf die Sensorebene herab zu etablieren und zu standardisieren (seit 1998).

IAONA ist eine Allianz von international führenden Herstellern der Automatisierungstechnik, die das Ziel verfolgt, durch regelmäßigen Erfahrungsaustausch **ETHERNET** als Standard im gesamten Industrieumfeld zu etablieren.
Die europäische Sektion wurde am 23.11.1999 am Rand der Messe **SPS/IPC/Drives** in Nürnberg gegründet.

Aktivitäten: Informationsaustausch in Foren, Workshops, Tutorials und Seminaren.
Ausdehnung von Ethernet auf alle Automatisierungsebenen
Schaffung von Übergängen zu anderen Feldbustechnologien
Initiierung von Kooperationen
Schaffung eines weltweiten Standards für die Industriekommunikation

6.3.6.2 IDA Interface for Distributed Automation

Die Firmen **Kuka, Jetter, Sick, Phoenix Contact und Leuze** streben gemeinsam die Einführung eines Standards für die Automatisierung an, der als Gateway für den Datenaustausch aller Automatisierungsgeräte entwickelt werden soll. Damit soll die Kommunikation zwischen verschiedenen Systemen erleichtert werden.

Als unterste Schicht soll **Ethernet** verwendet werden. Dazu muss dieses für industrielle Zwecke tauglich gemacht werden. Dazu gehört auch die Erfüllung der Anforderungen eines **Safety-Busses**.