FHD

Vorlesung: Mikroprozessorsysteme II

Dozent: Dr. Raffius (Zug C)

Sprechstunde: Montags 9:00 – 10:00 Uhr
g.raffius@fbi.fh-darmstadt.de, D14 Raum 3.06, Tel: 16 8448

Ressourcen: www2.fbi.fh-darmstadt.de/~raffius

Voraussetzung:

Bestandenes **Praktikum** MPS 1
Prüfungsleistung Programmieren 1 bestanden (ab Studienbeginn WS 04)
Prüfungsleistung Programmieren 2 "angefangen" (ab Studienbeginn WS 04)

Prüfung:

30.1.2005 um 10:15
Anmeldeschluss: 23.1.2005

Leistungsnachweis:

schriftliche Prüfung

Zulassung: Einzelabnahme der Praktika's durch Fachgespräch

Organisation

Praktikumsziel:

- Umgang mit der Peripherie des Prozessors
- Umgang mit Entwicklungswerkzeugen

Praktikumsdurchführung:

- jeweils 16 Personen (8 Gruppen zu je 2 Personen)
- 7 Termine (6 Abnahmen)
- Vorbereitung notwendig
  - vertraut machen mit Aufgabenstellung
  - Lösungsansätze aufzeigen zu Beginn des Praktikums
  - stichprobenhafte Prüfung
  - bei wiederholtem “nicht vorbereitet sein” ist Ausschluss möglich
- Tausch nur am 1. Termin mit Partner möglich
- Anwesenheitspflicht (Ausnahme: Krankheit / Attest)

- 6 Abnahmen, kein “Freischuss”
- **kein** “Überhang” von nicht-testierten Praktika’s in spätere Übungen möglich, da Übungen teilweise aufeinander aufbauen

Vorlesungsinhalt

Gliederung:
- Einleitung
- Power Management
- Parallele I/O
- Timer
- Interrupt Handling
- Serielle Schnittstelle
- Softwareinterrupt
- Weiterführende Themen

Literatur:

[1] Steve Furber, ARM Rechnerarchitekturen für System-on-Chip Design (FH-Bibliothek)

[2] Taschenbuch Mikroprozessortechnik (Beierlein, Hagenbruch)

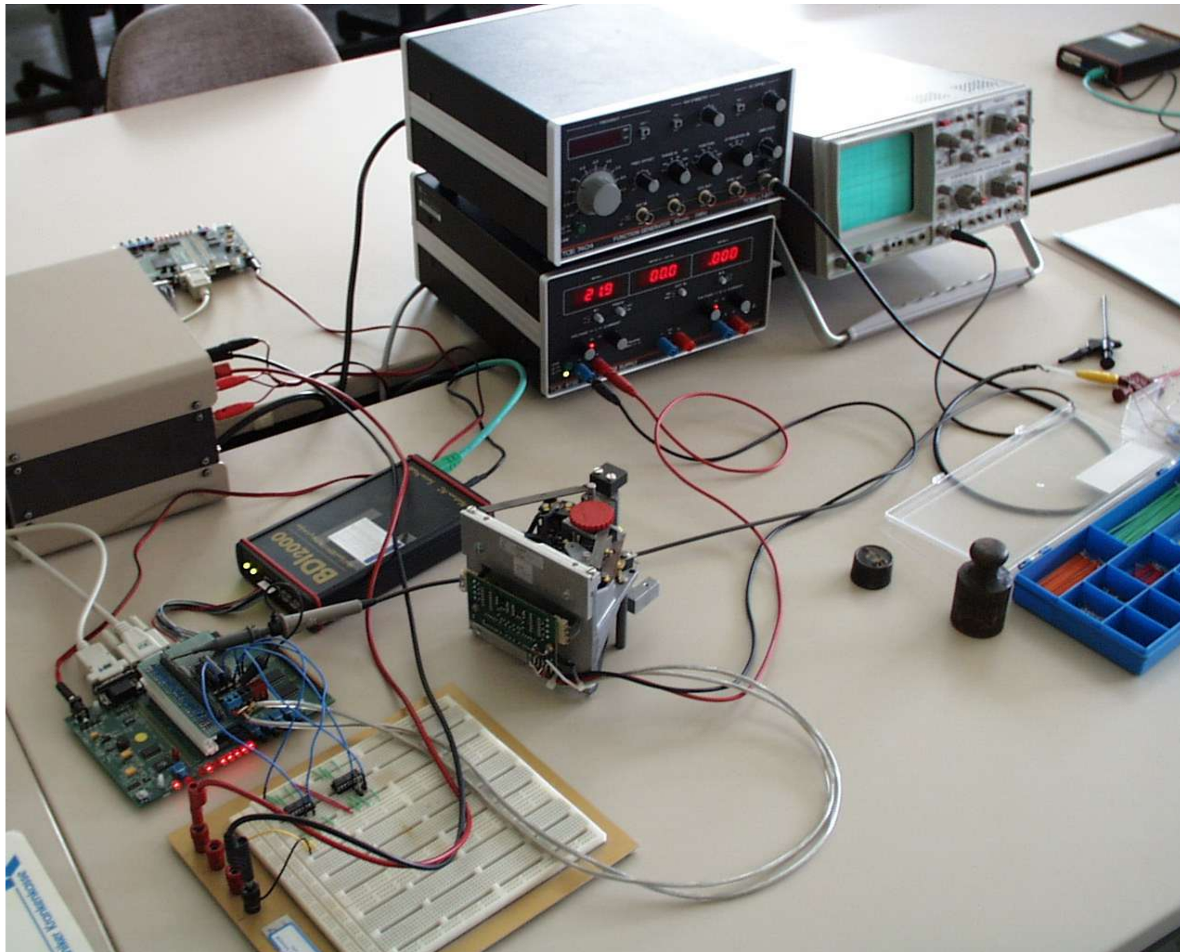
[3] AT91M63200(Complete).pdf (im Netz)

[4] AT91EB63.pdf (im Netz)

Praktikumsinhalt

Ziel: Implementierung einer Ausschankstation mit Waage und Pumpe

Waage (Schwingsaitenprinzip)



Praktikumsinhalt

Ziel: Implementierung einer Ausschankstation mit Waage und Pumpe

- Termin 1: Wiederholung: C-Programmierung, Erzeugen von Assembler Code
- Termin 2: PIO, (Interrupt)
- Termin 3: PIO, Interrupt, Timer (WAVE-Mode)
- Termin 4: PIO, Timer (CAPTURE-Mode)
- Termin 5: USART, SWI
- Termin 6: Zusammenfassung: Realisierung einer Ausschankstation (1)
- Termin 7: Zusammenfassung: Realisierung einer Ausschankstation (2)

Gliederung

- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)

Mikrocontroller

- Mikrocontroller: Mikrorechner auf einem Chip
  - Für spezielle Anwendungsfälle zugeschnitten
  - Meist Steuerungs- oder Kommunikationsaufgaben
  - Anwendung oft einmal programmiert und für die Lebensdauer des Mikrocontrollers auf diesem ausgeführt
  - Anwendungsfelder sind breit gestreut
  - Oft unsichtbar in uns umgebenden Geräten verborgen

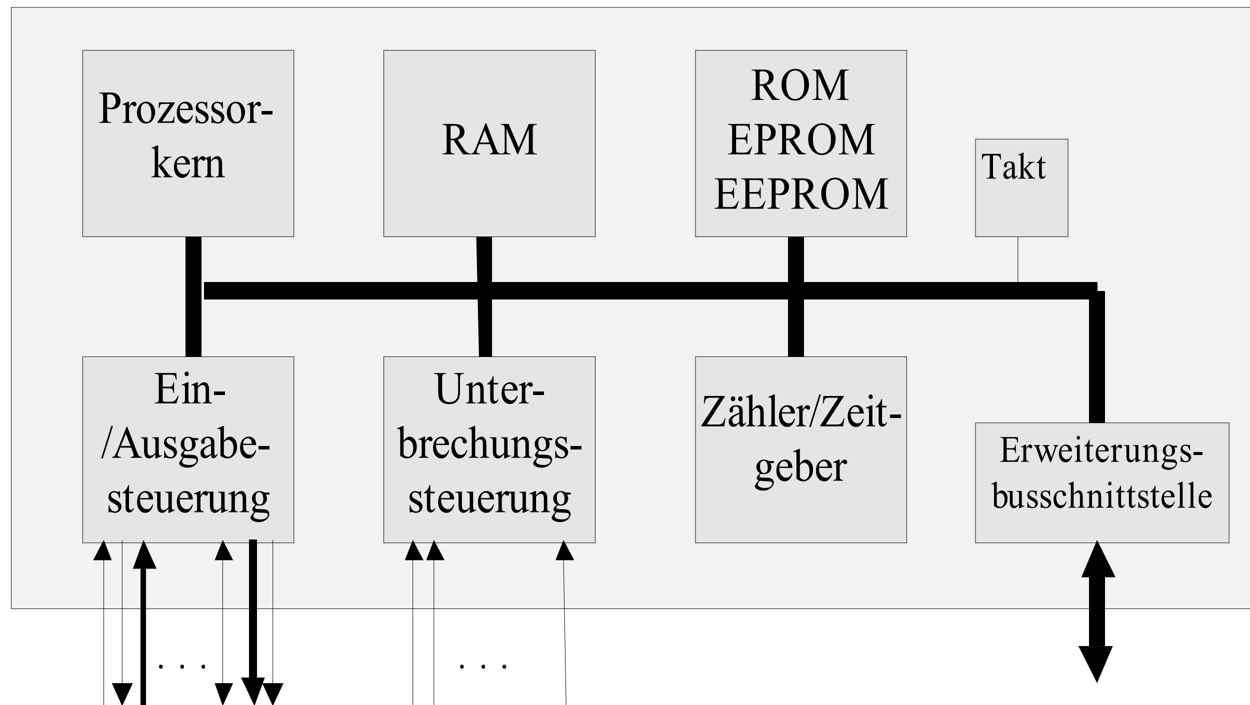
Anwendungsfelder

- im Haushalt
  - die Steuerung der Kaffeemaschine,
  - der Waschmaschine,
  - des Telefons,
  - des Staubsaugers,
  - des Fernsehers, ...
- in der KFZ Technik
  - das Motormanagement,
  - das Antiblockiersystem,
  - das Stabilitätsprogramm,
  - die Traktionskontrolle,
  - diverse Assistenten, z.B. beim Bremsen, ...
- in der Automatisierung
  - das Steuern und Regeln von Prozessen,
  - das Überwachen von Prozessen,
  - das Regeln von Materialflüssen,
  - die Steuerung von Fertigungs- und Produktionsanlagen, ...

Mikrocontroller

Abgrenzung zu Mikroprozessoren

Ein-Chip Mikrorechner mit aufgabenspezifischer Peripherie

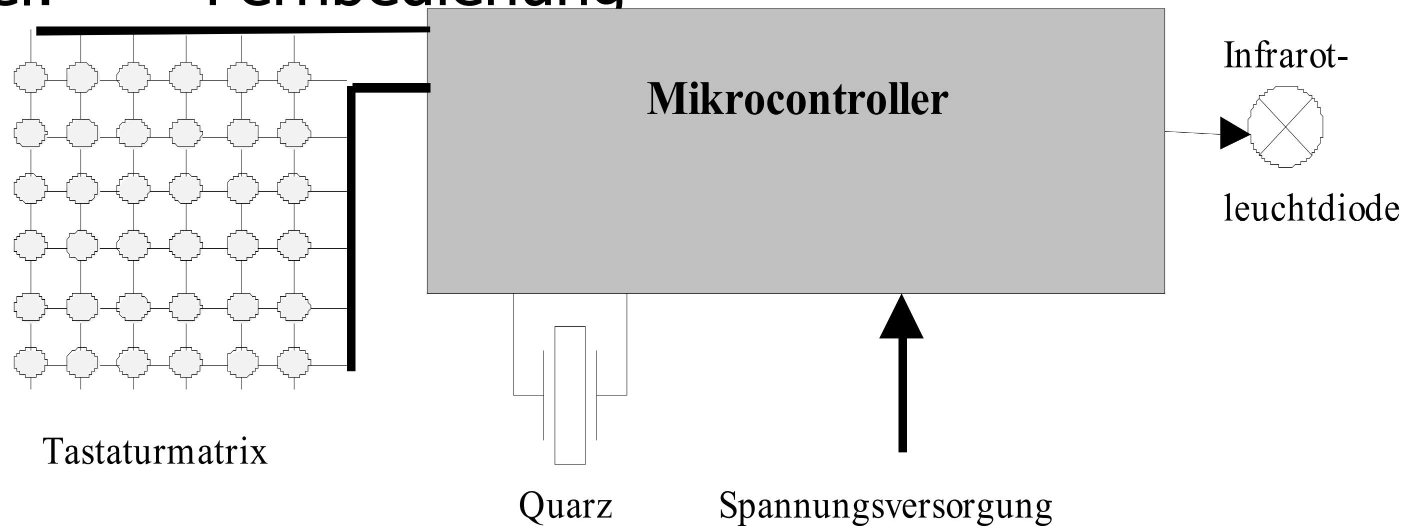


Mikrocontroller

Ziel: Möglichst wenige externe Bausteine für eine Steuerungsaufgabe

Idealfall: Mikrocontroller, Quarz, Stromversorgung sowie ggf. Treiber und ein Bedienfeld

Beispiel: Fernbedienung



Speicher

- integrierter Festwert- und Schreiblesespeicher
- Aufnahme von Daten und Programmen
- Vorteil: Einsparung von Anschlüssen und Decodierlogik bei vollständiger interner Speicherung
- Größe und Typ des Speichers unterscheiden oft verschiedene Untertypen desselben Mikrocontrollers
- z.B. je nach Stückzahl der Anwendung unterschiedlicher Typ des Festwertspeichers (ROM, PROM, EPROM, EEPROM, FLASH)

Mikrocontroller

- Serielle und parallele Ein-/Ausgabekanäle
  - grundlegenden digitalen Schnittstellen eines Mikrocontrollers
  - seriell oder parallel
  - synchron oder asynchron
- AD/DA-Wandler
  - grundlegenden analogen Schnittstellen eines Mikrocontrollers
  - Anschluss analoger Sensoren und Aktoren
  - Auflösung und Wandlungszeit sind die wichtigsten Größen
  - AD-Wandler sind häufiger anzutreffen als DA-Wandler

- Zähler und Zeitgeber (Timer)
  - im Echtzeitbereich ein wichtiges Hilfsmittel
  - für eine Vielzahl unterschiedlich komplexer Anwendungen einsetzbar
  - Bsp:
    - Zählen von Ereignissen, Messen von Zeiten
    - Pulsweitenmodulation, Frequenz- oder Drehzahlmessung

Mikrocontroller

- Watchdog
  - „Wachhund“ zur Überwachung der Programmaktivitäten eines Mikrocontrollers
  - Programm muss in regelmäßigen Abständen Lebenszeichen liefern
  - Bleiben diese aus, so nimmt der Wachhund einen Fehler im Programmablauf an => Reset

Mikrocontroller

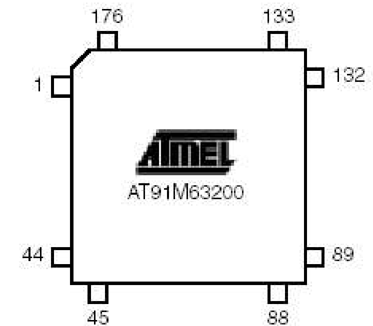
- Unterbrechungen (Interrupts)
  - Unterbrechung des Programmablaufs bei Ereignissen
  - Schnelle, vorhersagbare Reaktion auf Ereignisse
  - Insbesondere wichtig bei Echtzeitanwendungen
  - Behandlung eines Ereignisses durch eine Interrupt-Service-Routine
  - Mikrocontroller kennen meist externe Unterbrechungsquellen (Eingangssignale) und
  - interne Unterbrechungsquellen (Zähler, Zeitgeber, E/A-Kanäle, ...)

DMA (Direct Memory Access)

- Direkter Datentransfer zwischen Peripherie und Speicher ohne Beteiligung des Prozessorkerns
- Höhere Datenraten durch spezielle Transferhardware
- Entlastung des Prozessorkerns
- Prozessorkern muss lediglich die Randbedingungen des Transfers festlegen
- Meist in Mikrocontrollern gehobener Leistungsklasse zu finden

Mikrocontroller (MC)

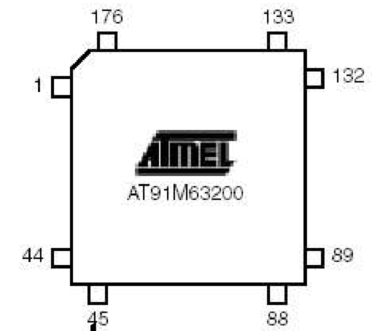
- Allgemeines:
  - vollständiges Mikrocomputersystem auf einem Chip
    - Mikrocontrollerkern (Core),
    - Speicher,
    - Peripherie



- Ziel:
  - maximale funktionelle Integration
  - geringe Systemkosten
  - hohe Verarbeitungsleistung bei niedrigem Energieverbrauch

Mikrocontroller (MC)

- Standardmikrocontroller (AT91M63200)
  - abh. vom Design universell einsetzbar
  - kein bestimmtes Marktsegment
- Kundenorientierte Mikrocontroller
  - in Einschränkung universell einsetzbar
  - zugeschnitten auf bestimmtes Marktsegment
  - hohe Stückzahl für konkrete Projekte



Mikrocontroller (MC)

- Mikrocontrollerkern (Core):
  - On-Chip integrierte CPU
  - beinhaltet komplexes Steuerwerk, ALU, Register, Bussystem
- On-Chip Peripherie:
  - Taktozillator, IO Ports, Timer, (A/D D/A Wandler),
- Interrupt Controller, Watchdog Timer,
- Kommunikationsschnittstellen (USART,...)

ATMEL AT91M63200

JTAGSEL
NTRST
TMS
TDO
TDI
TCK
JTAGSEL
JTAG
Embedded ICE
ARM7TDMI Core
MCKI
PB17/MCKO
PB3
PB4
PB5
PB6
PB7
PB8
PB9
PB10
PB11
PB12
PB13
PB14
PB15
PB16
Clock
Internal RAM 2K Bytes
ASB Controller
ASB
AMBA Bridge
PA9/IRQ0
PA10/IRQ1
PA11/IRQ2
PA12/IRQ3
PA13/FIQ
P I O
AIC: Advanced Interrupt Controller
PA14/SCK0
PA15/TXD0
PA16/RXD0
USART0
2 PDC Channels
PA17/SCK1
PA18/TXD1/NTRI
PA19/RXD1
USART1
2 PDC Channels
APB
PA20/SCK2
PA21/TXD2
PA22/RXD2
USART2
2 PDC Channels
PA23/SPCK
PA24/MISO
PA25/MOSI
PA26/NPCS0/NSS
PA27/NPCS1
PA28/NPCS2
PA29/NPCS3
SPI: Serial Peripheral Interface
2 PDC Channels
PMC: Power Management Controller
Chip ID
NWDOVF
WD: Watchdog Timer
PIOA: Parallel I/O Controller A
PIOB: Parallel I/O Controller B
Reset
NRST
MPI: Multi-processor Interface
MPI_A1-MPI_A9
MPI_D0-MPI_D15
MPI_NCS
MPI_RNW
MPI_BR
MPI_BG
P I O
PB0/MPI_NOE
PB1/MPI_NLB
PB2/MPI_NUB
EBI: External Bus Interface
D0-D15
A1-A19
A0/NLB
NRD/NOE
NWR0/NWE
NWR1/NUB
NWAIT
NCS0-NCS3
A20/CS7
A21/CS6
A22/CS5
A23/CS4
EBI User Interface
PB18/BMS
TC: Timer/ Counter Block 0
TC0
TC1
TC2
PB19/TCLK0
PB22/TCLK1
PB25/TCLK2
PB20/TIOA0
PB21/TIOB0
PB23/TIOA1
PB24/TIOB1
PB26/TIOA2
PB27/TIOB2
P I O
TC: Timer/ Counter Block 1
TC0
TC1
TC2
PA0/TCLK3
PA3/TCLK4
PA6/TCLK5
PA1/TIOA3
PA2/TIOB3
PA4/TIOA4
PA5/TIOB4
PA7/TIOA5
PA8/TIOB5

AT63200 Memory Map

Device	Name	Base
AIC	Advanced Interrupt Controller	0xFFFFF000
Reserved		
WD	Watchdog Timer	0xFFFF8000
PMC	Power Management Controller	0xFFFF4000
PIO	Parallel I/O Controller B	0xFFFF0000
PIO	Parallel I/O Controller A	0xFFFEC000
Reserved		
TC1	Timer/Counter Channels 3, 4, 5	0xFFFD4000
TC0	Timer/Counter Channels 0, 1, 2	0xFFFD0000
Reserved		
USART2	USART 2	0xFFFC8000
USART1	USART 1	0xFFFC4000
USART0	USART 0	0xFFFC0000
SPI	SPI	0xFFFBC000
Reserved		
SF	Special Function	0xFFF00000
Reserved		
EBI	External Bus Interface	0xFFE00000

Gliederung

- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)

Power Management

FHD

- Power Management (Power Saving Modi, Stromsparmodi)
  - Niedrige Stromaufnahmen wichtig für batteriebetriebene Anwendungen
- Möglichkeiten:
  - Reduktion der Taktfrequenz
  - Taktabschaltung von Prozessor und/oder Peripherie
- Takt für CPU wird bei Reset eingeschaltet
- Takt für Peripherie wird bei Reset ausgeschaltet

Initialisierung des Power Managements

Offset	Register	Name	Access	Reset State
0x00	System Clock Enable Register	PMC_SCER	Write only	-
0x04	System Clock Disable Register	PMC_SCDR	Write only	-
0x08	System Clock Status Register	PMC_SCSR	Read only	0x1
0x0C	Reserved			
0x10	Peripheral Clock Enable Register	PMC_PCER	Write only	-
0x14	Peripheral Clock Disable Register	PMC_PCDR	Write only	-
0x18	Peripheral Clock Status Register	PMC_PCSR	Read only	0x0

Initialisierung des Power Managements

Clock für die Peripherie:

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	**PIOB**	**PIOA**	**-**	**TC5**	**TC4**	**TC3**	**TC2**
7	6	5	4	3	2	1	0
TC1	**TC0**	**SPI**	**US2**	**US1**	**US0**	**-**	**-**

Power Management

- Wie spreche ich die Peripherie an ?
  - Ansprechend der Peripherie über Adressen (Memory mapped IO)
  - Schreiben/Lesen des Inhalts von Registern eines E/A Bausteins über die angegebene Adresse
  - Alternative: spezielle E/A Befehle (z.B. 80x86)

```
// Gib an dem Baustein mit der E/A Adresse
// 20H den Wert 1
// aus
MOV AX, 1
OUT 20H            // Spezialbefehl
```

Power Management

FHD

volatile *declarator (Quelle: MSDN Library)*

The volatile keyword is a type qualifier used to declare that an object can be modified in the program by something other than statements, such as the operating system, the hardware, or a concurrently executing thread.

The following example declares a volatile integer nVint whose value can be modified by external processes:

```
int volatile nVint;
```

Objects declared as volatile are not used in optimizations because their value can change at any time. The system always reads the current value of a volatile object at the point it is requested, even if the previous instruction asked for a value from the same object. Also, the value of the object is written immediately on assignment.

One use of the volatile qualifier is to provide access to memory locations used by asynchronous processes such as interrupt handlers.

Gliederung

- Einleitung
- Power Management
- **Parallele I/O (PIO)**
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)

Gliederung

- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)

PIO

- MC ist mit seiner Aussenwelt über Portpins verbunden
  - Zusammenfassung einzelner Portpins zu Ports
  - Parallel I/O (PIO)
  - 32 bit PIO
- digitale Ein- und Ausgabe (I/O)
  - z.B. Taster, LED's, ...
- analoge Werte ebenfalls über Portpins übertragbar
  - erfordert analoge Peripheriekomponenten
  - AD / DA Wandler

PIO

- Ansteuerung der Ports über Portregister
  - Daten -> Port-Daten-Register
  - Richtung des Datenverkehrs (bidirektional) -> Daten- Richtungs-Register
  - Alternativfunktionen für Ports -> Steuerregister

Parallel IO (PIO) - Struktur

PIO_OSR
PAD Output Enable
0
1
Peripheral Output Enable
0
1
PIO_PSR
PIO_ODSR
PIO_MDSR
PAD Output
1
0
PAD
Peripheral Output
Pad Input
Filter
1
0
0
1
Peripheral Input
PIO_IFSR
PIO_PSR
PIO_PDSR
Event Detection
PIO_ISR
PIO_IMR
PIOIRQ

Initialisierung der PIO

Offset	Register	Name	Access	Reset State
0x00	PIO Enable Register	PIO_PER	Write only	-
0x04	PIO Disable Register	PIO_PDR	Write only	-
0x08	PIO Status Register	PIO_PSR	Read only	0x3FFFFFFF (A) 0x0FFFFFFF (B)
0x0C	Reserved	-	-	-
0x10	Output Enable Register	PIO_OER	Write only	-
0x14	Output Disable Register	PIO_ODR	Write only	-
0x18	Output Status Register	PIO_OSR	Read only	0
0x1C	Reserved	-	-	-
0x20	Input Filter Enable Regist	PIO_IFER	Write only	-
0x24	Input Filter Disable Regis	PIO_IFDR	Write only	-
0x28	Input Filter Status Registe	PIO_IFSR	Read only	0
0x2C	Reserved	-	-	-
0x30	Set Output Data Register	PIO_SODR	Write only	-
0x34	Clear Output Data Regist	PIO_CODR	Write only	-
0x38	Output Data Status Regis	PIO_ODSR	Read only	0
0x3C	Pin Data Status Register	PIO_PDSR	Read only	
0x40	Interrupt Enable Register	PIO_IER	Write only	-
0x44	Interrupt Disable Registe	PIO_IDR	Write only	-
0x48	Interrupt Mask Register	PIO_IMR	Read only	0
0x4C	Interrupt Status Register	PIO_ISR	Read only	
0x50	Multi-driver Enable Regis	PIO_MDER	Write only	-
0x54	Multi-driver Disable Regis	PIO_MDDR	Write only	-
0x58	Multi-driver Status Regist	PIO_MDSR	Read only	0
0x5C	Reserved	-	-	-

PIO-A Port Belegung (1)

Bit Number	Port Name	Port Name	Signal Description	Signal Direction	Reset State	Pin Number
0	PA0	TCLK3	Timer 3 Clock Signal	Input	Input	66
1	PA1	TIOA3	Timer 3 Signal A	Bi-directional	Input	67
2	PA2	TIOB3	Timer 3 Signal B	Bi-directional	Input	68
3	PA3	TCLK4	Timer 4 Clock Signal	Input	Input	69
4	PA4	TIOA4	Timer 4 Signal A	Bi-directional	Input	70
5	PA5	TIOB4	Timer 4 Signal B	Bi-directional	Input	71
6	PA6	TCLK5	Timer 5 Clock Signal	Input	Input	72
7	PA7	TIOA5	Timer 5 Signal A	Bi-directional	Input	75
8	PA8	TIOB5	Timer 5 Signal B	Bi-directional	Input	76
9	PA9	IRQ0	External Interrupt 0	Input	Input	77
10	PA10	IRQ1	External Interrupt 1	Input	Input	78
11	PA11	IRQ2	External Interrupt 2	Input	Input	79
12	PA12	IRQ3	External Interrupt 3	Input	Input	80
13	PA13	FIQ	Fast Interrupt	Input	Input	81
14	PA14	SCK0	USART 0 Clock Signal	Bi-directional	Input	82
15	PA15	TXD0	USART 0 Transmit Data Signal	Output	Input	83

PIO-A Port Belegung (2)

FHD

Bit Number	Port Name	Port Name	Signal Description	Signal Direction	Reset State	Pin Number
16	PA16	RXD0	USART 0 Receive Data Signal	Input	Input	84
17	PA17	SCK1	USART 1 Clock Signal	Bi-directional	Input	85
18	PA18	TXD1	USART 1 Transmit Data Signal	Output	Input	86
19	PA19	RXD1	USART 1 Receive Data Signal	Input	Input	91
20	PA20	SCK2	USART 2 Clock Signal	Bi-directional	Input	92
21	PA21	TXD2	USART 2 Transmit Data Signal	Output	Input	93
22	PA22	RXD2	USART 2 Receive Data Signal	Input	Input	94
23	PA23	SPCK	SPI Clock Signal	Bi-directional	Input	95
24	PA24	MISO	SPI Master In Slave Out	Bi-directional	Input	96
25	PA25	MOSI	SPI Master Out Slave In	Bi-directional	Input	97
26	PA26	NPCS0	SPI Peripheral Chip Select 0	Bi-directional	Input	98
27	PA27	NPCS1	SPI Peripheral Chip Select 1	Output	Input	99
28	PA28	NPCS2	SPI Peripheral Chip Select 2	Output	Input	100
29	PA29	NPCS3	SPI Peripheral Chip Select 3	Output	Input	101
30	-	-	-	-	-	-
31	-	-	-	-	-	-

PIO-B Port Belegung (1)

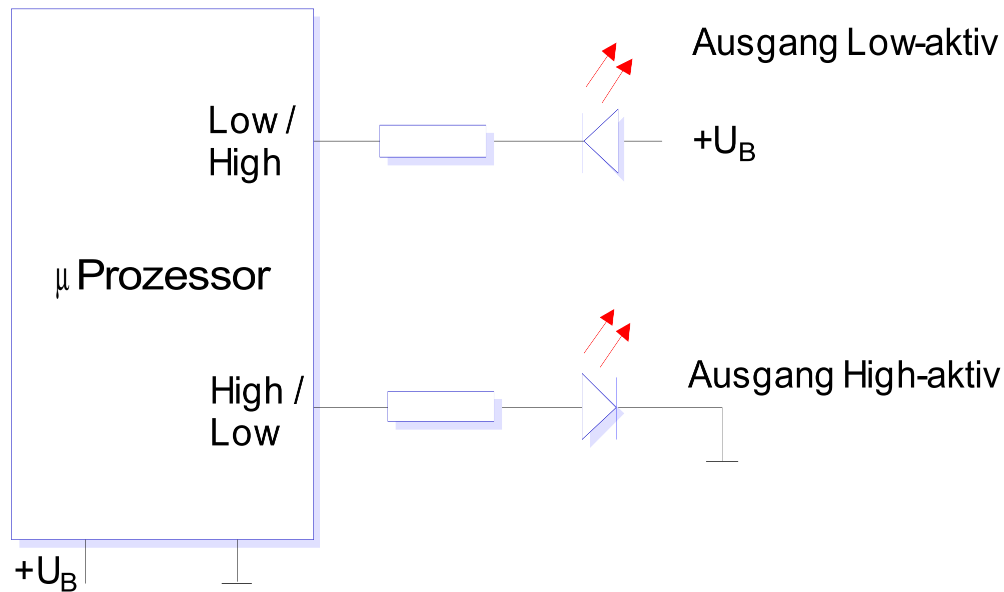
FHD

Bit Number	Port Name	Port Name	Signal Description	Signal Direction	Reset State	Pin Number
0	PB0	MPI_NOE	MPI Output Enable	Input	Input	139
1	PB1	MPI_NLB	MPI Lower Byte Select	Input	Input	140
2	PB2	MPI_NUB	MPI Upper Byte Select	Input	Input	141
3	PB3	-	-	-	Input	142
4	PB4	-	-	-	Input	143
5	PB5	-	-	-	Input	144
6	PB6	-	-	-	Input	145
7	PB7	-	-	-	Input	146
8	PB8	-	-	-	Input	149
9	PB9	-	-	-	Input	150
10	PB10	-	-	-	Input	151
11	PB11	-	-	-	Input	152
12	PB12	-	-	-	Input	153
13	PB13	-	-	-	Input	154
14	PB14	-	-	-	Input	155
15	PB15	-	-	-	Input	156

PIO-B Port Belegung (2)

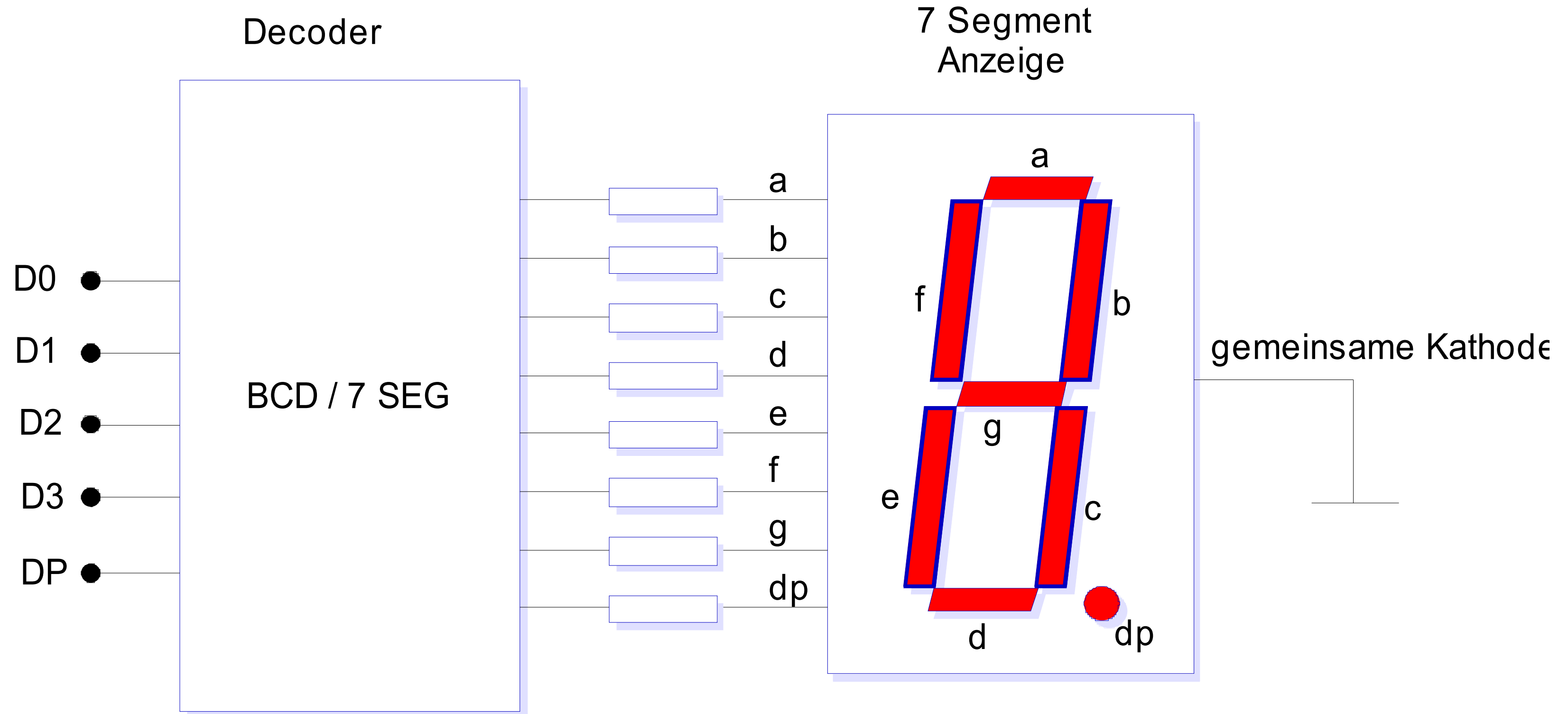
Bit Number	Port Name	Port Name	Signal Description	Signal Direction	Reset State	Pin Number
16	PB16	-	-	-	Input	157
17	PB17	MCKO	Master Clock Output	Output	Input	158
18	PB18	BMS	Boot Mode Select	Input	Input	163
19	PB19	TCLK0	Timer 0 Clock Signal	Input	Input	55
20	PB20	TIOA0	Timer 0 Signal A	Bi-directional	Input	56
21	PB21	TIOB0	Timer 0 Signal B	Bi-directional	Input	57
22	PB22	TCLK1	Timer 1 Clock Signal	Input	Input	58
23	PB23	TIOA1	Timer 1 Signal A	Bi-directional	Input	61
24	PB24	TIOB1	Timer 1 Signal B	Bi-directional	Input	62
25	PB25	TCLK2	Timer 2 Clock Signal	Input	Input	63
26	PB26	TIOA2	Timer 2 Signal A	Bi-directional	Input	64
27	PB27	TIOB2	Timer 2 Signal B	Bi-directional	Input	65
28	-	-	-	-	-	-
29	-	-	-	-	-	-
30	-	-	-	-	-	-
31	-	-	-	-	-	-

LED

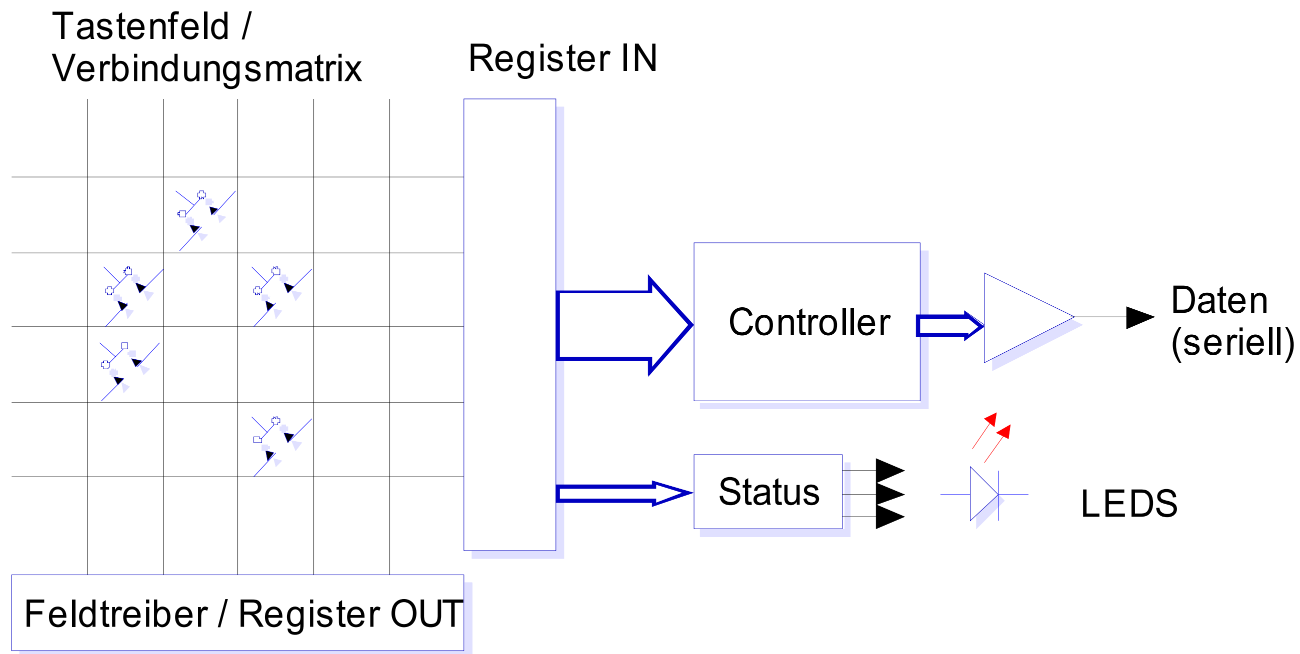


Ausgang Low-aktiv ist häufiger, da dann der Strom für die Leuchtdiode nicht vom Prozessor zur Verfügung gestellt werden muß

Sieben Segment Anzeige



Tastatur



Gliederung

- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)

Ereignisgesteuerte Programme

- Ereignisgesteuerte Programme stellen eine wichtige Klasse von Programmen dar
- Wichtigstes Kennzeichen ist, daß die Steuerung des Programmablaufs durch externe Ereignisse und nicht durch die Daten alleine erfolgt
- Ereignisgesteuerte Programme sind immer dann nötig, wenn der Computer intensiv mit seiner Umwelt in Interaktion tritt
- Beispiele:
  - grafische Oberflächen
  - Kommunikation
  - Prozeßsteuerung

Externe Ereignisse

- Wichtige externe Ereignisquellen sind
  - Tastatur
  - Maus
  - serielle Schnittstellen
    - asynchrone Schnittstellen, Ethernet, USB, Firewire
    - SCSI, SPI, I2C, CAN
  - Prozeßanbindungen
    - parallele Schnittstellen
    - AD und DA Schnittstellen
  - Timer
    - Real Time Clock
    - Watchdog

Mechanismen zur Ereignisverarbeitung

- Busy Waiting
  - Die einfachste Form der Ereignisabfrage
  - Endlosschleife bis Ereignis eintritt
- Polling
  - Zeitgesteuerte Abfrage des Ereignisses. Einfache Form der Parallelverarbeitung, wenn alle Ereignisquellen nacheinander abgefragt werden.
  - Problematisch wenn das Message Arrival Pattern nicht bekannt ist

Mechanismen zur Ereignisverarbeitung

- Interrupt
  - „Echte“ Parallelverarbeitung der Ereignisse
  - Gefahren durch Multithreading, wenn kritische Sections nicht geschützt sind
  - Gefahr bei zu langen Interruptroutinen oder zu vielen Interrupts

Polling versus Interrupt

- Ereignisbehandlung durch Eventloop
  - Programm fragt alle Schnittstellen in einer Schleife ab und reagiert auf auftretende Ereignisse
  - Randbedingung: Reaktionsroutinen dürfen selbst nicht warten, sondern müssen sofort zurückkehren
  - Problem: Die Wahrscheinlichkeit Ereignisse zu verpassen ist hoch, wenn Eventloop zu lange dauert
- Synchrone Variante:
  - Programm wird durch Timer getriggert und fragt nur in definierten Zeitabständen die Ereignisse ab.
  - Vorteil: Die verbleibende Zeit kann für andere Programme genutzt werden
  - Vorteil: Berechenbarkeit

Polling versus Interrupt

- Asynchrone Ereignisbehandlung durch Interrupts
  - laufendes Programm wird unterbrochen und eine Reaktionsroutine für das Ereignis gestartet
  - Vorteil: schnellere Reaktion für den Interrupt höchster Priorität. Trennung verschiedener Ereignisquellen auf Programmebene möglich
  - Problem: Nur höchster Interrupt hat garantierte Reaktionszeit, und auch nur dann, wenn der Interrupt nicht durch die laufenden Programme verboten wird (einige Betriebssysteme tun dies)
  - Problem: Der Datentransport zwischen Interruptroutine und Anwendung ist sehr kritisch
  - Problem: Interruptprogramme dürfen nur sehr eingeschränkt Betriebssystemfunktionalitäten nutzen

Ereignisbehandlung durch DMA

- Für sehr schnelle Rekationen sind Interruptprogramme oft zu langsam.
  - Besonders die schnelle Kommunikation (Gigabit Ethernet, Firewire ...) erfordert eine Hardwareunterstützung um den notwendigen Datenstrom zu gewährleisten.
  - Diese Hardwareunterstützung wird durch Direct Memory Access Controller zur Verfügung gestellt
- DMA Controller können auf Interrupts reagieren und die Daten in den Speicher oder zu IO Device transferieren.

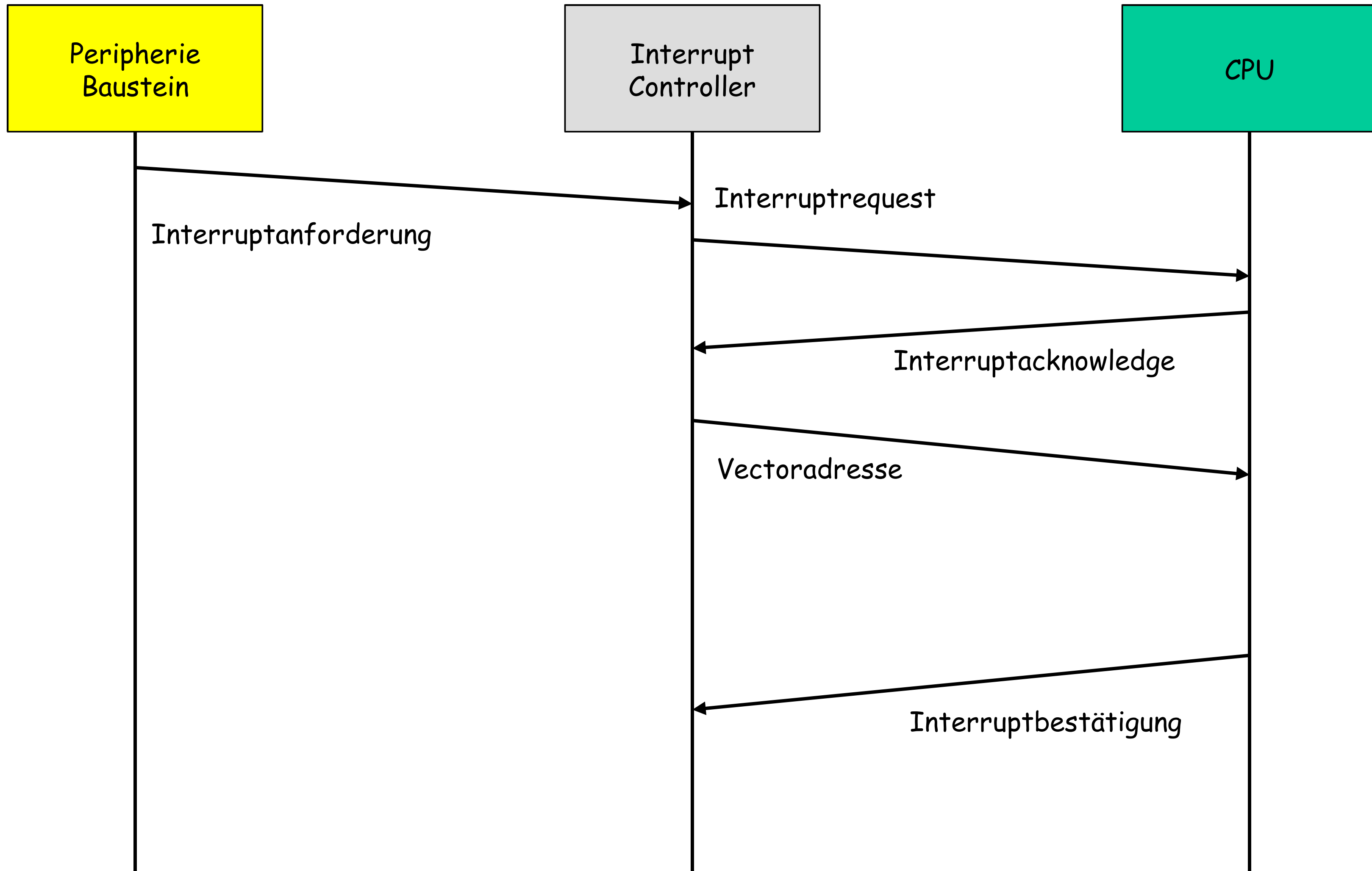
Pollingbeispiel

```
// Polling (round robin)

void main(void)
    {
    for(;;)    // Endlosschleife
        {
        if( ereignis1)
                job1();
        if( ereignis2)
                job2();
        if( ereignis3)
                job3();
        }
    }
```

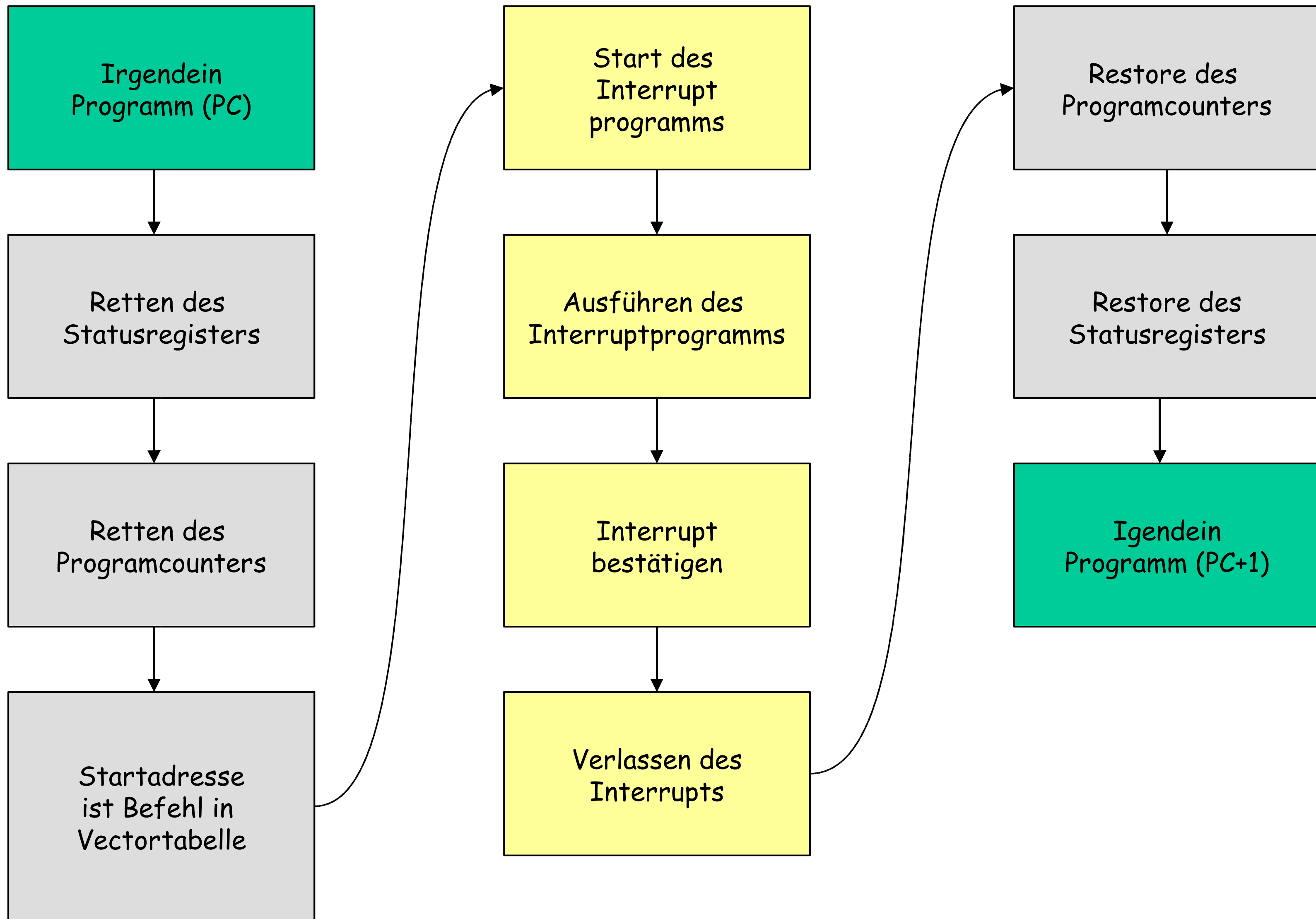
Pollingbeispiel

```
// Polling  (prioritätsgesteuert)

void main(void)
    {
    for(;;) // Endlosschleife
       {
       if( ereignis1)
              { job1(); continue; }
       if( ereignis2)
              { job2(); continue; }
       if( ereignis3)
              { job3(); continue; }
       }
    }
```

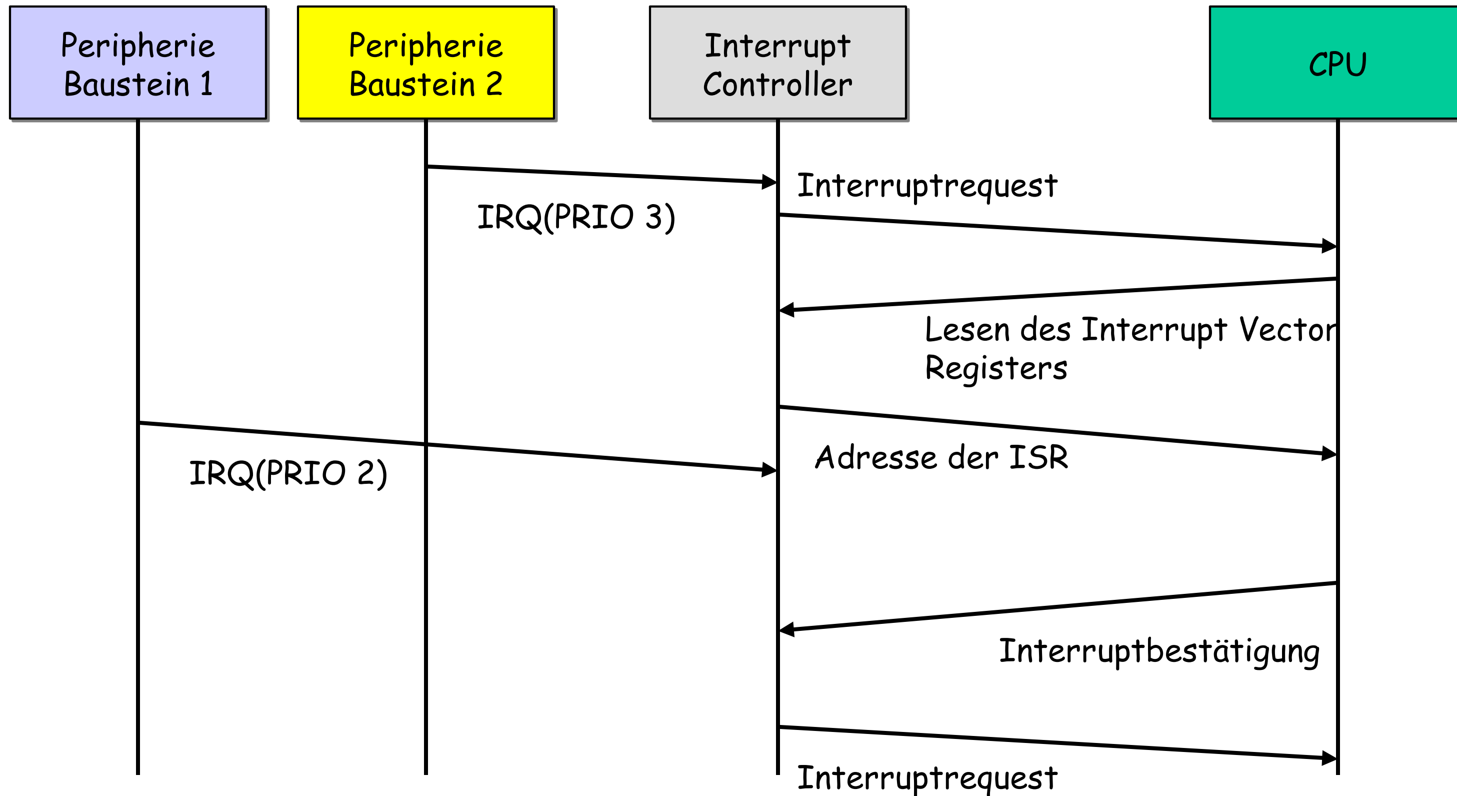
Interrupts



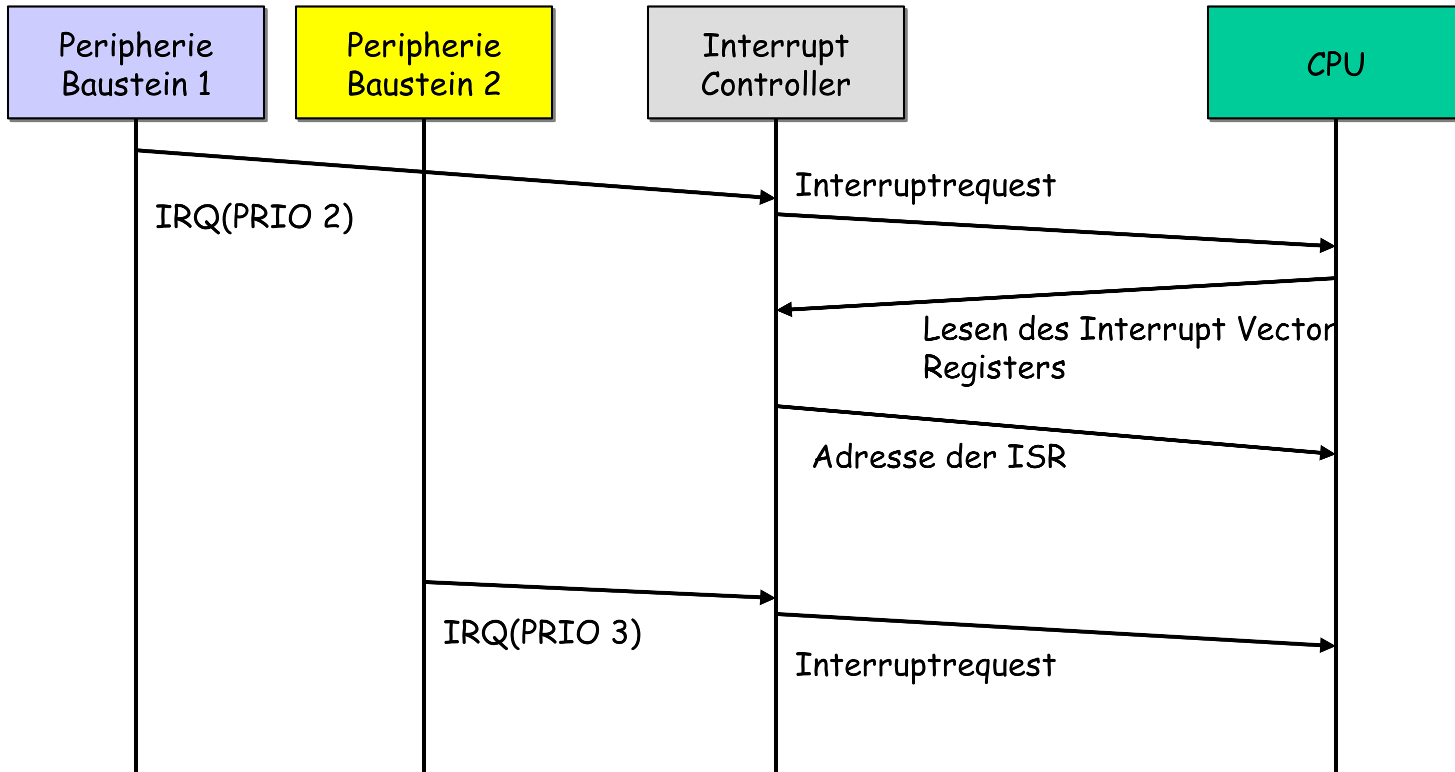
Ablauf einer Interruptverarbeitung



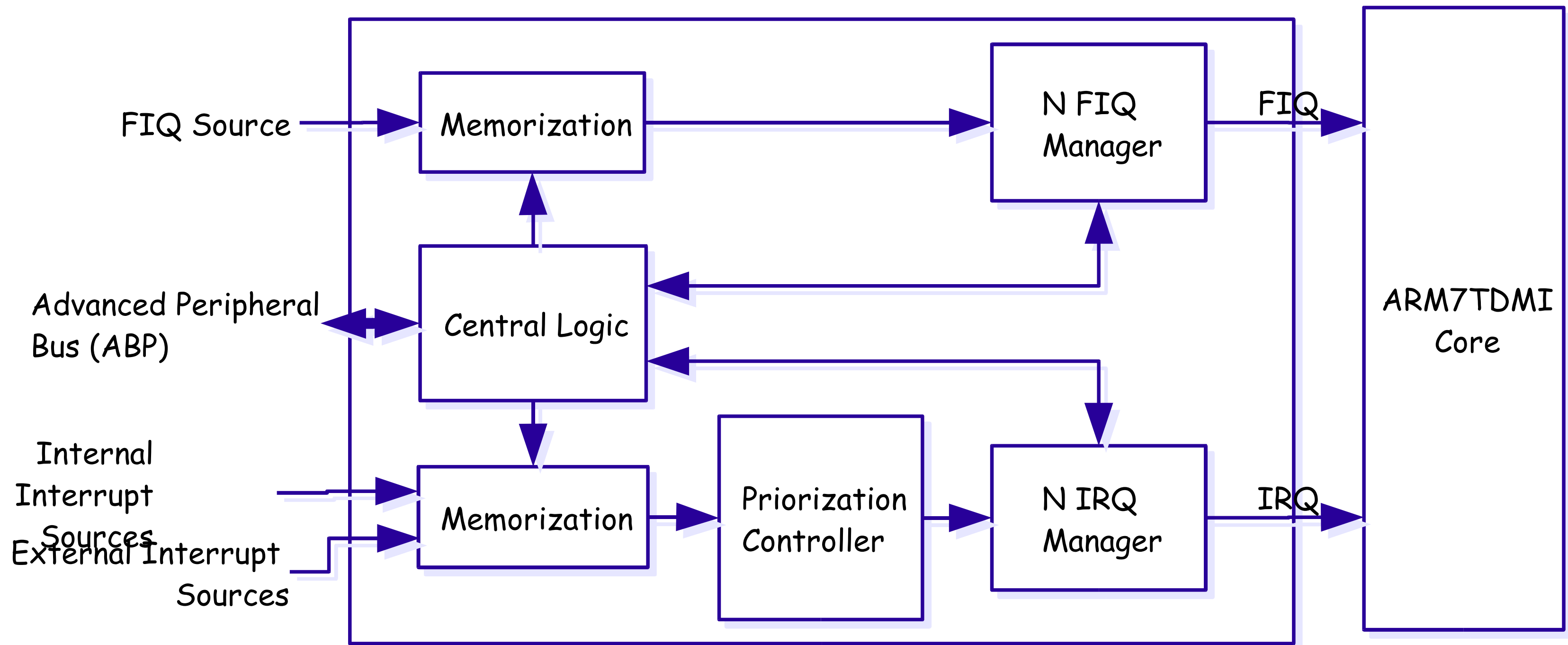
Interrupts (Priorität)



Interrupts (Priorität)



Der Advanced Interrupt Controller (AIC)



Register des AIC

Offset	Register	Name	Access	Reset State
0x000	Source Mode Register 0	AIC_SMR0	Read/Write	0
0x004	Source Mode Register 1	AIC_SMR1	Read/Write	0
-	-	-	Read/Write	0
0x07C	Source Mode Register 31	AIC_SMR31	Read/Write	0
0x080	Source Vector Register 0	AIC_SVR0	Read/Write	0
0x084	Source Vector Register 1	AIC_SVR1	Read/Write	0
-	-	-	Read/Write	0
0x0FC	Source Vector Register 31	AIC_SVR31	Read/Write	0
0x100	IRQ Vector Register	AIC_IVR	Read only	0
0x104	FIQ Vector Register	AIC_FVR	Read only	0
0x108	Interrupt Status Register	AIC_ISR	Read only	0
0x10C	Interrupt Pending Register	AIC_IPR	Read only	
0x110	Interrupt Mask Register	AIC_IMR	Read only	0
0x114	Core Interrupt Status Register	AIC_CISR	Read only	0
0x118	Reserved	-	-	-
0x11C	Reserved	-	-	-
0x120	Interrupt Enable Command Register	AIC_IECR	Write only	-
0x124	Interrupt Disable Command Register	AIC_IDCR	Write only	-
0x128	Interrupt Clear Command Register	AIC_ICCR	Write only	-
0x12C	Interrupt Set Command Register	AIC_ISCR	Write only	-
0x130	End of Interrupt Command Register	AIC_EOICR	Write only	-

Interrupt Quellen

Interrupt Source	Interrupt Name	Interrupt Description
0	FIQ	Fast interrupt
1	SWIRQ	Soft interrupt (generated by the AIC)
2	US0IRQ	USART Channel 0 interrupt
3	US1IRQ	USART Channel 1 interrupt
4	US2IRQ	USART Channel 2 interrupt
5	SPIRQ	SPI interrupt
6	TC0IRQ	Timer Channel 0 interrupt
7	TC1IRQ	Timer Channel 1 interrupt
8	TC2IRQ	Timer Channel 2 interrupt
9	TC3IRQ	Timer Channel 3 interrupt
10	TC4IRQ	Timer Channel 4 interrupt
11	TC5IRQ	Timer Channel 5 interrupt
12	WDIRQ	Watchdog interrupt
13	PIOAIRQ	Parallel I/O Controller A interrupt
14	PIOBIRQ	Parallel I/O Controller B interrupt
15-27	---	Reserved
28	IRQ3	External interrupt 3
29	IRQ2	External interrupt 2
30	IRQ1	External interrupt 1
31	IRQ0	External interrupt 0

Das Enable/Disable/Status Register

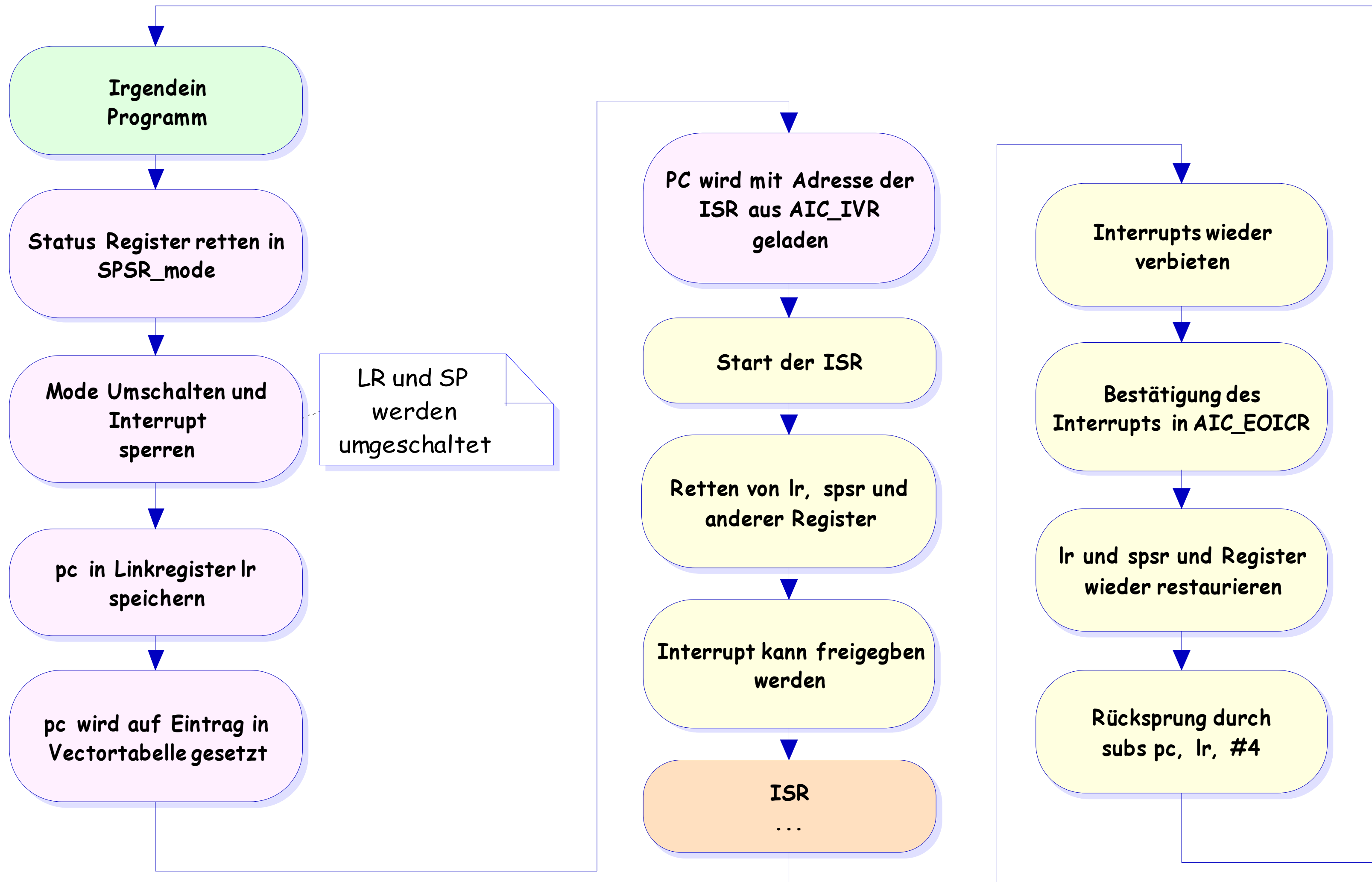
31	30	29	28	27	26	25	24
IRQ0	IRQ1	IRQ2	IRQ3	---	---	---	---
23	22	21	20	19	18	17	16
---	---	---	---	---	---	---	---
15	14	13	12	11	10	9	8
---	PIOBIRQ	PIOAIRQ	WDIRQ	TC5IRQ	TC4IRQ	TC3IRQ	TC2IRQ
7	6	5	4	3	2	1	0
TC1IRQ	TC0IRQ	SPIRQ	US2IRQ	US1IRQ	US0IRQ	SWIRQ	FIQ

Das Source Mode Register

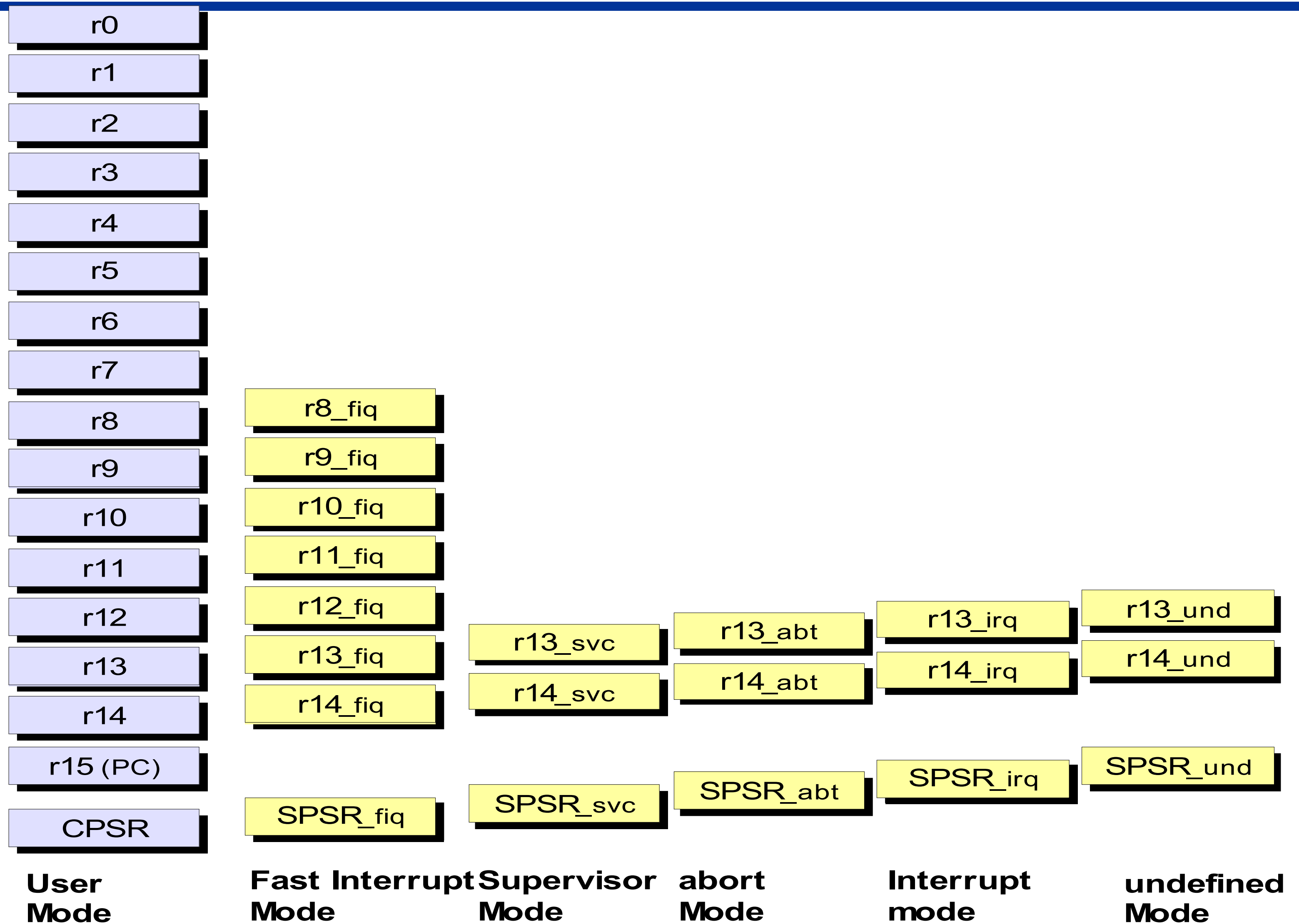
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	SRCTYPE		-	-	PRIOR		

SRCTYPE		Internal Sources	External Sources
0	0	Level Sensitive	Low-Level Sensitive
0	1	Edge Triggered	Negative-Edge Triggered
1	0	Level Sensitive	High-Level Sensitive
1	1	Edge Triggered	Positive-Edge Triggered

Ablauf einer Interruptverarbeitung



ARM Registerstruktur



Exception Handling

0x1c	FIQ
0x18	IRQ
0x14	(Reserviert)
0x10	Data Abort
0x0c	Prefetch Abort
0x08	Software Interrupt
0x04	Undefined Instruction
0x00	Reset

Current Program Status Register (CPSR)

31			28	27			24	23							16	15							8	7	6	5	4				0
N	Z	C	V	Q				undefined								undefined								I	F	T	mode				

- **Condition Code Flags**
  - N = Negatives ALU Ergebnis
  - Z = Alu Ergebnis ist Null
  - C = Alu Ergebnis erzeugte Carry
  - V = Alu erzeugte Overflow
- Interrupt Disable Bits
  - I = 1, IRQ gesperrt
  - F = 1, FIQ gesperrt

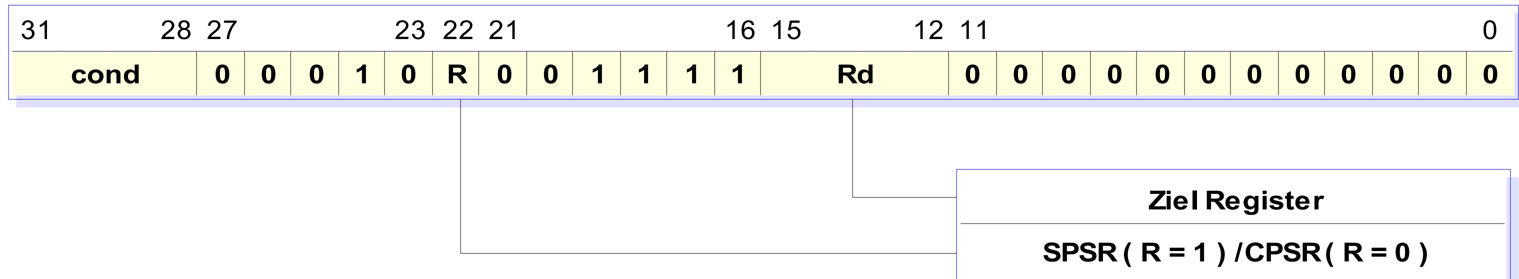
- T Bit
  - T = 0, Prozessor in Arm State
  - T = 1, Prozessor in Thumb State
- Mode Bits
  - Spezifizieren den Prozessor Mode

Mode Bits

31			28	27			24	23							16	15							8	7	6	5	4				0
N	Z	C	V	Q				undefined								undefined								I	F	T	mode				

M[4:0]	Mode	Accessible Registers
10000	User	PC, R14 to R0, CPSR
10001	FIQ	PC, R14_fiq to R8_fiq, R7 to R0, CPSR, SPSR_fiq
10010	IRQ	PC, R14_irq, R13_irq,R12 to R0, CPSR, SPSR_irq
10011	SVC	PC, R14_svc, R13_svc,R12 to R0, CPSR, SPSR_svc
10111	Abort	PC, R14_abt, R13_abt,R12 to R0, CPSR, SPSR_abt
11011	Undef	PC, R14_und, R13_und,R12 to R0, CPSR, SPSR_und
11111	System	PC, R14 to R0, CPSR (Architecture 4 only)

Status Register nach Register Transfer



- Syntax

  mrs{<cond>} Rd, <psr>

  <psr> = CPSR oder SPSR

  z.B. Rd := CPSR

Register nach Status Register Transfer

31		28	27	26	25	24	23	22	21	20	19		16	15			12	11		0
cond			0	0	#	1	0	R	1	0	Feld			1	1	1	1	Operand		

Feld Maske

SPSR(1) / CPSR (0)

1 →

11		8	7		0
#rot			8-Bit Immediate		

0 →

11							4	3		0
0	0	0	0	0	0	0	Rm			

Name	Bedeutung	PSR Bits	Masken Bit
c	Kontroll Feld	[7:0]	16
x	Extension Feld	[15:8]	17
s	Status Feld	[23:16]	18
f	Flag Feld	[31:24]	19

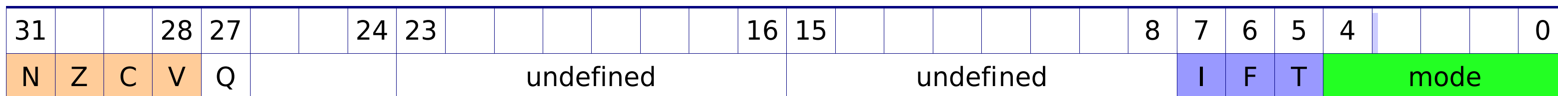
Syntax:

MSR{<cond>} <psr>_<feld>, Rm

MSR {<cond>} <psr>_<feld>, #<immediate>

<psr> = CPSR oder SPSR

<feld> = Kombination von f, s, x, c



Beispiele zu MSR und MRS

- Setzen der Prozessorflags N, C, Z, V

```
msr     CPSR_f, #0xf0000000
```

- Setzen des Carry Flags

```
mrs     r0, CPSR
orr     r0, r0, #0x20000000
msr     cpsr_f, r0
```

- Vom Super Visor Mode in Interrupt Mode schalten

```
mrs     r0, CPSR                    // aktuellen Status holen
bic     r0, r0, #0x1f // unterste 5 Bits löschen
orr     r0, r0, #0x12               // neuen Mode eintragen
msr     cpsr_c, r0    // zurückschreiben
```

Interrupt Entry (Re-entrant)

```
@ lr korrigieren für Rücksprung
    sub       lr, lr, #4
    stmfd     sp!, {lr}
    mrslr, SPSR
    stmfd     sp!, {r0, lr}
@ in System Mode gehen und
@ Interrupt freigeben
    mrslr, CPSR
    bic       lr, lr, #I_BIT
    orr       lr, lr, #ARM_MODE_SYS
    msrcpsr, lr
@ Register und LR_USER auf USER/System Stack retten
    stmfd     sp!, {r1-r3, benutzte Register, r12, r14}
```

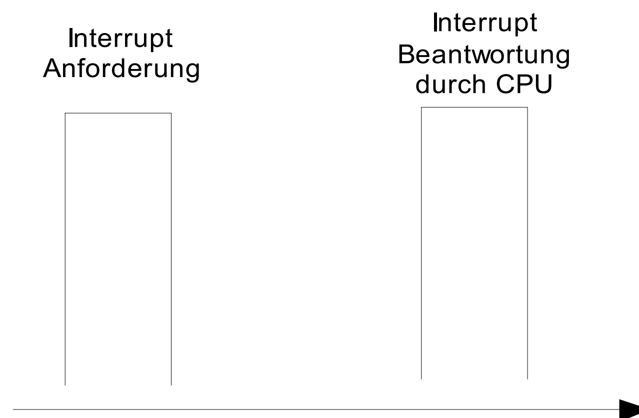
Interrupt Exit

FHD

```
@ Register restaurieren
    ldmfd       sp!, {r1-r3, benutzte Register, r12, r14}
@ CPSR modifizieren um Interrupt zu sperren und um in
@ Interrupt Mode zurückzugelangen
    mrs         r0, CPSR
    bic         r0, r0, #ARM_MODE_SYS
    orr         r0, r0, #(I_BIT | ARM_MODE_IRQ)
    msr         CPSR, r0
@ Ende des Interrupts an AIC
    ldr         r0, =AIC_BASE
    str         r0, [r0, #AIC_EOICR]
@ Restaurieren der Register r0 und SPSR_irq vom IRQ_STACK
    ldmfd       sp!, {r0,lr}
    msr         SPSR, lr
@ Rücksprung über LR_irq mit Rückschreiben des Statusregisters
    ldmfd       sp!, {pc}^
```

Der Spurious Interrupt

- Der Spurious Interrupt tritt auf, wenn ein Device einen Interrupt im Levelsensitiven Mode anfordert und die Anforderung zur Zeit des Eintritts in die Interruptroutine (d. h. beim Lesen des Vector Registers) nicht mehr existiert
- Die Routine für den Spurious Interrupt ist eine normale Interruptroutine, die den Interruptstack durch Beschreiben des EOICR Registers zurücksetzen muß



Der Spurious Interrupt

```
isr_spurious:
    sub     lr, lr, #4
    stmfd   sp!, {r0,lr}
    ldr     r0, =AIC_BASE
    str     r0, [r0, #AIC_EOICR]
    ldmfd   sp!, {r0, pc}^
```

Konstanten

```
ARM_MODE_SYS  = 0x1f
I_BIT         = 0x80
AIC_BASE      = 0xfffff000
AIC_EOICR     = 0x130
```

FAQ zu Interrupts

- Woher weiß ein Mikroprozessor wo die Interruptroutine zu finden ist
- Woher wissen Mikroprozessoren die eine Interruptvektortabelle benutzen, wo diese Tabelle ist
- Kann ein Mikroprozessor mitten in einer Operation unterbrochen werden?
- Welche Interruptroutine wird ausgeführt, wenn zwei Interrupts gleichzeitig auftreten.
- Kann ein Interrupt einen anderen Interrupt unterbrechen?

FAQ zu Interrupts

- Was passiert, wenn ein Interrupt Request auftritt während die Interrupts gesperrt sind?
- Was passiert wenn ich die Interrupts sperre (disable) und vergesse sie freizugeben (enable)
- Was passiert wenn ich einen Interrupt disable wenn er disabled ist oder ihn enable wenn er enabled ist?
- Sind Interrupts enabled oder disabled wenn das System startet
- Kann ich Interruptroutinen in C schreiben?

Kommunikation mit Interruptroutinen

- Eine Kommunikation zwischen Programmen und Interruptroutinen kann nur über den Daten-Speicher erfolgen.
- Das große Problem stellt die Datenkonsistenz dar
- Eine Sicherung über Semaphoren ist nicht möglich, da eine Interruptroutine nicht warten darf
- Der Aufruf von Betriebssystemsfunktionen aus Interruptroutinen ist gefährlich, da bei einem Taskwechsel der Interrupt nicht beendet wird

Das Shared Data Problem

- Beispiel:

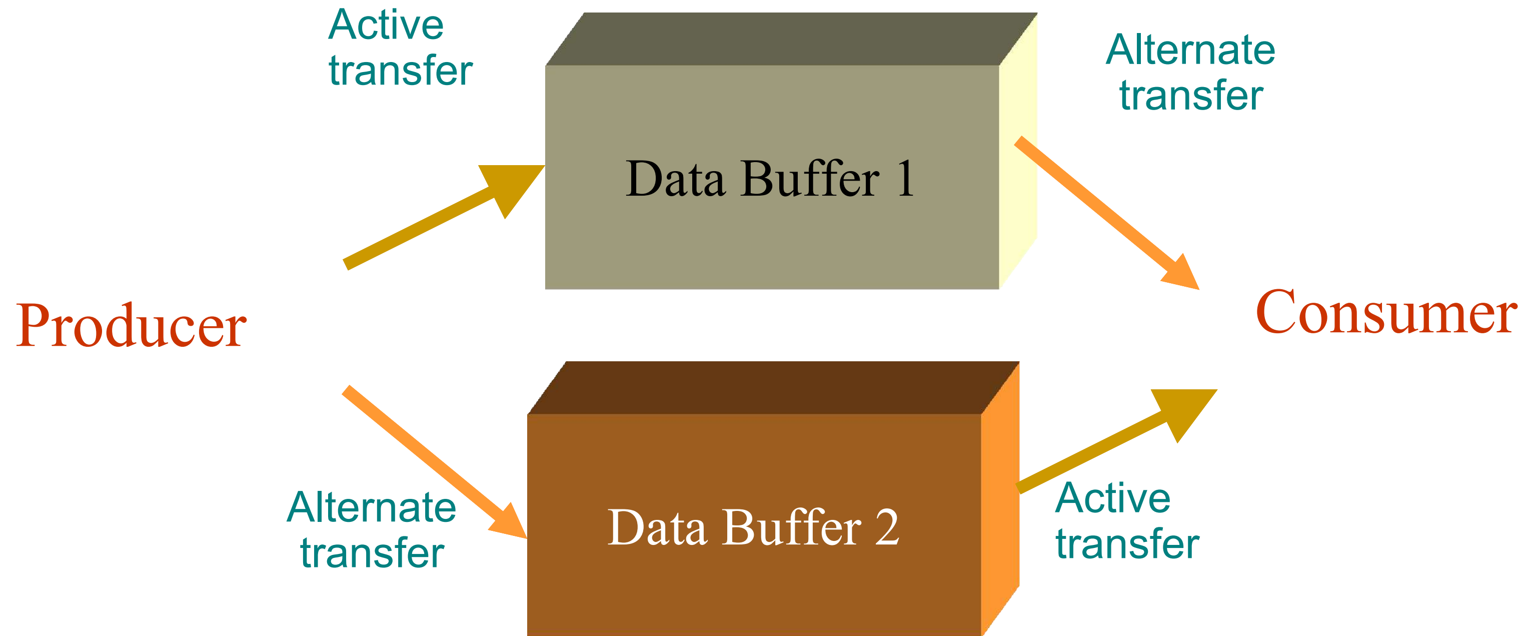
```
static int iTemperature[2];

void interrupt vReadTemperatures (void) {
   iTemperature[0] = !!Wert von AD Wandler 1
   iTemperature[1] = !!Wert von AD Wandler 2
}

void main(void) {
   if (iTemperature[0] != iTemperature[1]) {
     !! Löse Alarm aus
   }
}
```

Lösungen des Shared Data Problems

FHD

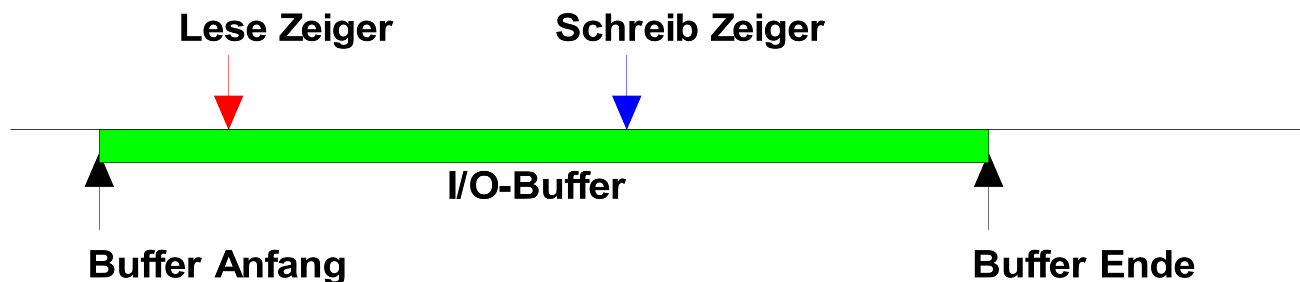
- Schaffen von „Atomaren“ oder „kritischen“ Sections
  - Sperren des Interrupts
    - Funktioniert nur im Systemmode
    - I-Bit im Prozessor Statuswort verhindert Interrupts
    - einfachste Methode um atomare Operationen zu erzeugen
    - Problematisch bei häufiger Anwendung
  - Double Buffering
    - Hauptprogramm gibt an welcher Buffer belegt ist, Interruptroutine benutzt freien Buffer
    - Die Bufferreservierung ist atomare Operation
  - IO-Buffer mit Schreib und Lesezeiger
    - Zeigeroperationen sind atomar.
    - Zeiger wird als letzte Operation bewegt

Double Buffer



I/O Buffer

- Der Schreib Zeiger wird nur vom Hauptprogramm bewegt
- Der Lese Zeiger wird vom Interruptprogramm bewegt
- Wenn Schreibzeiger == Lesezeiger, dann ist Vorgang beendet
- Normaler Buffer: Schreibzeiger und Lesezeiger können auf Bufferanfang gesetzt werden, wenn Interrupt nicht aktiv ist
- Ringbuffer: Wenn Schreib oder Lesezeiger das Bufferende erreichen, werden Sie auf den Bufferanfang gesetzt.



Charakteristiken des Shared Data Problem

- Der Fehler taucht nur selten auf und die Auswirkungen sind jedesmal unterschiedlich.
- Der Fehler taucht beim Kunden auf, der am weitesten entfernt ist
- Der Fehler taucht meist um 5 Uhr Nachmittags auf, besonders an Freitagen. Falls das Gerät dauernd betrieben wird, taucht der Fehler meist nachts auf und an Wochenenden
- Der Fehler taucht immer dann auf, wenn man ihn gerade nicht sucht
- Der Fehler taucht nur dann auf, wenn keine Debugeinrichtung vorhanden ist
- Der Fehler taucht auf, wenn Ihr Produkt bereits auf dem Mars gelandet ist
- Der Fehler taucht bei Kundendemonstrationen auf

Wozu Assembler Programmierung ?

- Es gibt sehr **tiefliegende Routinen in Betriebssystemen**, die man immer noch in Assembler programmieren muß.
  - Beispiel: Das Retten des Kontexts beim Taskwechsel im Scheduler,
  - Startupcode von Systemen
- Das Ansprechen von **Peripheriebausteinen** ist manchmal aus Assembler heraus einfacher wie aus höheren Programmiersprachen
- Die Programmierung von **sehr kleinen und schnellen Interruptroutinen** ist meist effektiver in Assembler zu bewältigen
- In sehr **zeitkritischen Routinen**, kann man versuchen Teile in Assembler nachzuoptimieren

Wozu Assembler Programmierung ?

- Es gibt Prozessoren, die sich nur sehr schlecht in höheren Programmiersprachen programmieren lassen, und die überwiegend in Assembler programmiert werden
  - Beispiel: einige DSP's, einige veraltete 8 und 16 Bit Prozessoren
- Wenn man **Programme debugged** ist es manchmal notwendig sich den Assemblercode anzusehen und direkt im Assembler zu Tracen um den Fehler zu finden
- Wenn man Programme **Laufzeit optimal oder Speicherplatz optimal** schreiben möchte, sollte man verstehen, wie der Compiler den Code in Maschinensprache umsetzt.

Wozu Assembler Programmierung ?

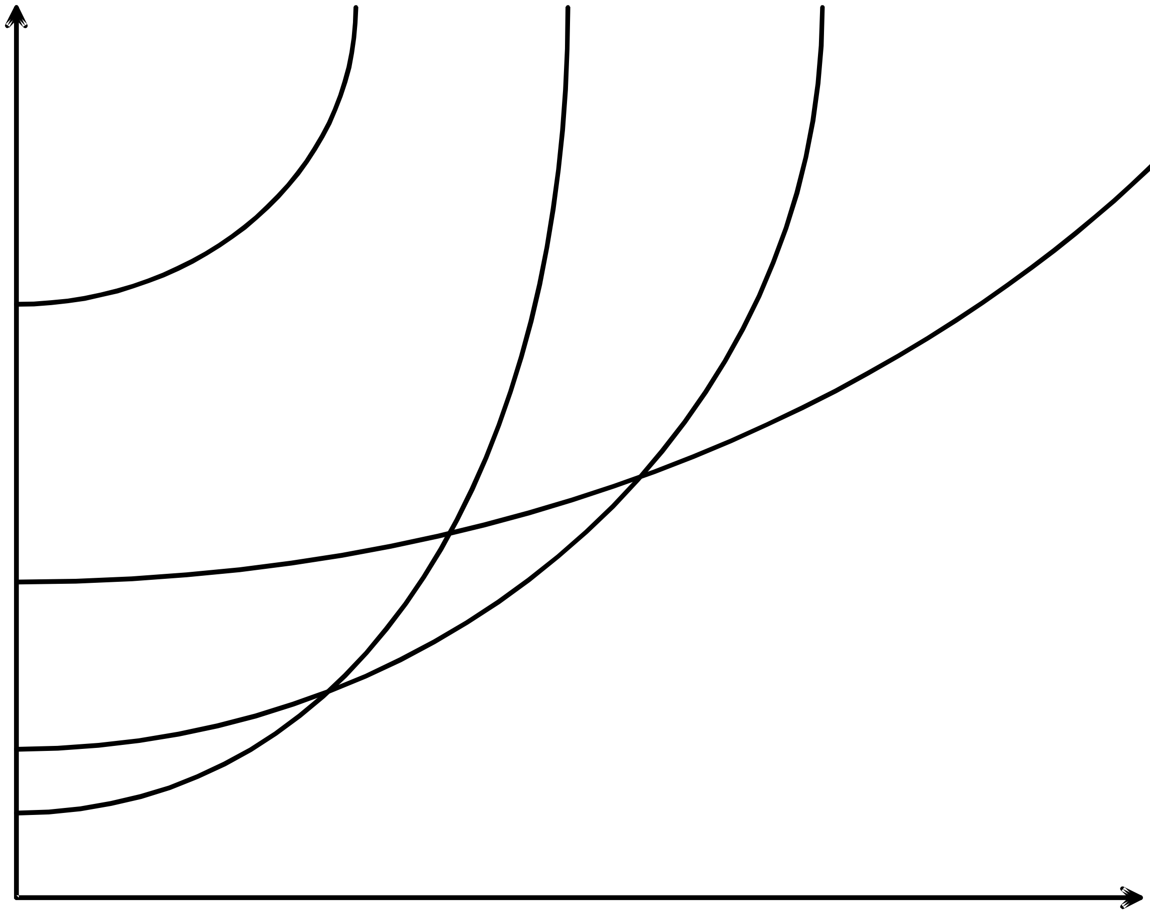
FHD

- Die **Inbetriebnahme von Hardware** erfordert meist das direkte Ansprechen der Hardware aus Assembler heraus.
- Auch **Compiler können Fehler haben**. Diese kann man nur beweisen, wenn man den Assemblercode versteht.
- Die **Entwicklung von Compilern** setzt ein gründliches Verständnis von Mikroprozessorsystem und Assembler voraus.
- Es gibt sehr viel **alte Software** in Assembler, die einen hohen Pflege und Wartungsaufwand erfordert

Gründe gegen Assemblerprogrammierung

- Die **Produktivität** von Assemblerprogrammierern ist um den Faktor 5 bis 10 unter der Produktivität eines Hochsprachen-Programmierers. Assemblerprogrammierung ist daher sehr teuer
- Es gibt eine **Komplexitätsgrenze**, ab der man Programme nicht mehr überblicken kann. Diese liegt in der Assemblerprogrammierung sehr niedrig
- Manche Assemblersprachen sind so **unlesbar**, daß man sie kaum noch programmieren kann (SPARC, PowerPC)
- Assemblersprachen **unterscheiden** sich von Prozessor zu Prozessor, ja sogar für einen einzelnen Prozessor kann es verschiedene **inkompatible** Assembler geben.

Produktivität



Gliederung

- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- **Timer**
  - Compare (WAVE) Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)

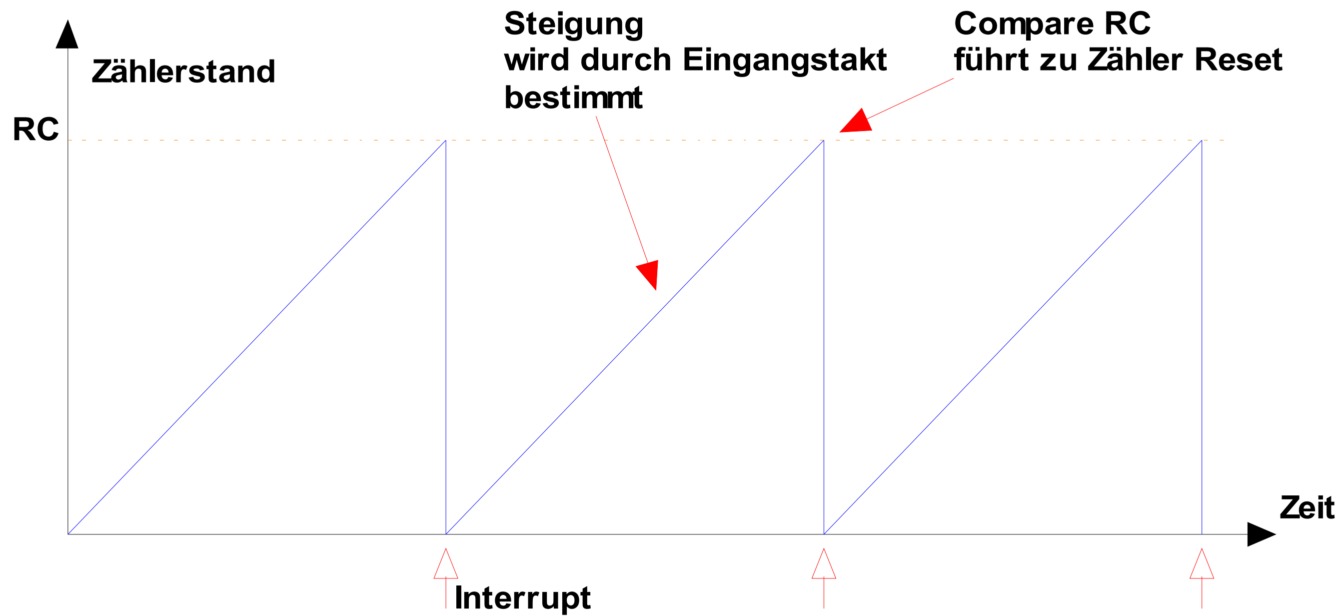
Gliederung

- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)

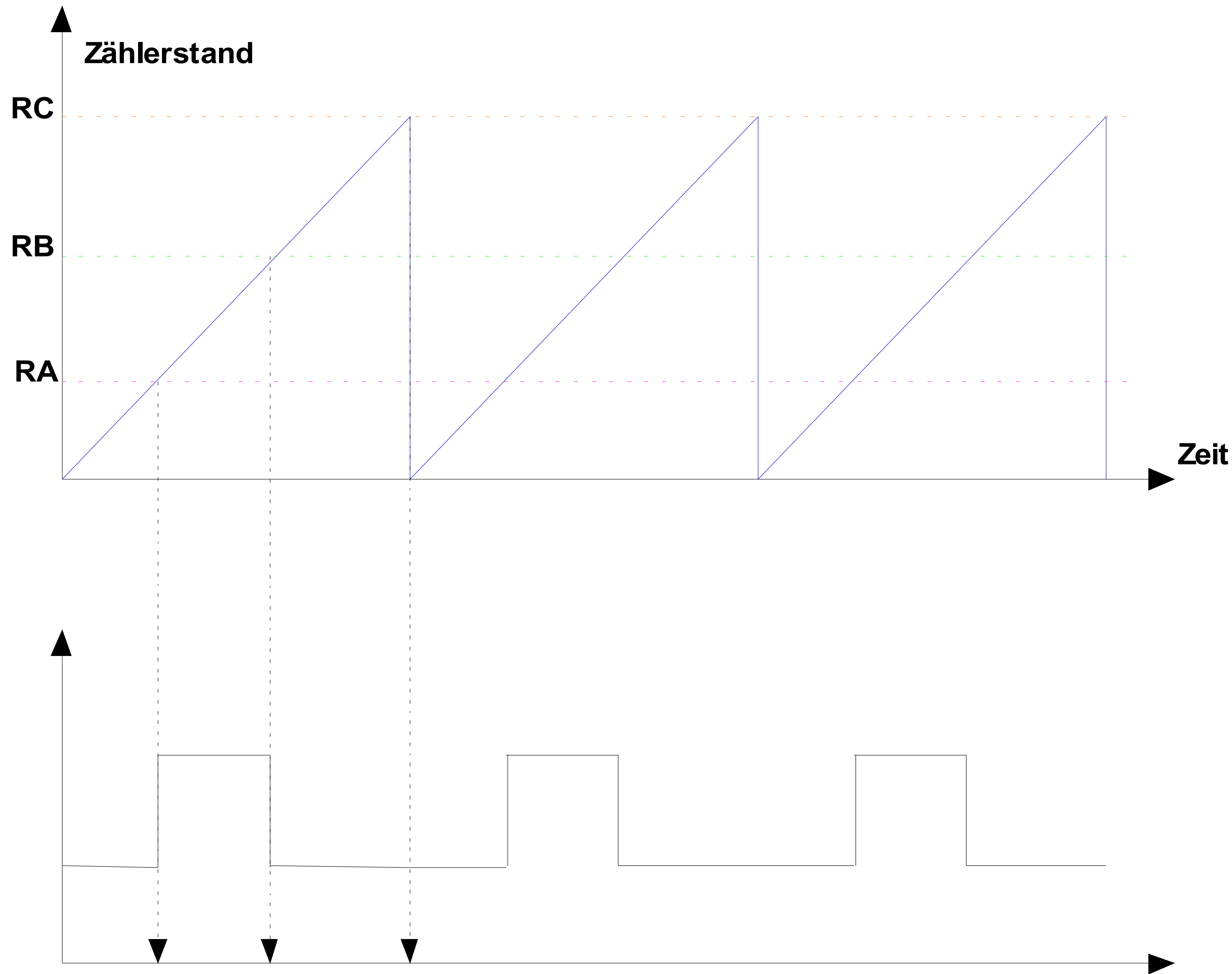
Timeanwendungen

- Erzeugung von periodischen Interrupts
  - Betriebssysteme, Uhr
- Messung von Zeiten
  - Pulsbreitenmessung, Periodendauermessung
  - Anwendung in der Messtechnik
- Messung von Frequenzen
- Zählen von Ereignissen
- Erzeugung von periodischen Signalen
  - Ansteuerung von Motoren, DA Wandlung, Signalerzeugung
- Timeout Behandlung, Watchdog
- Erzeugung von Pulsen

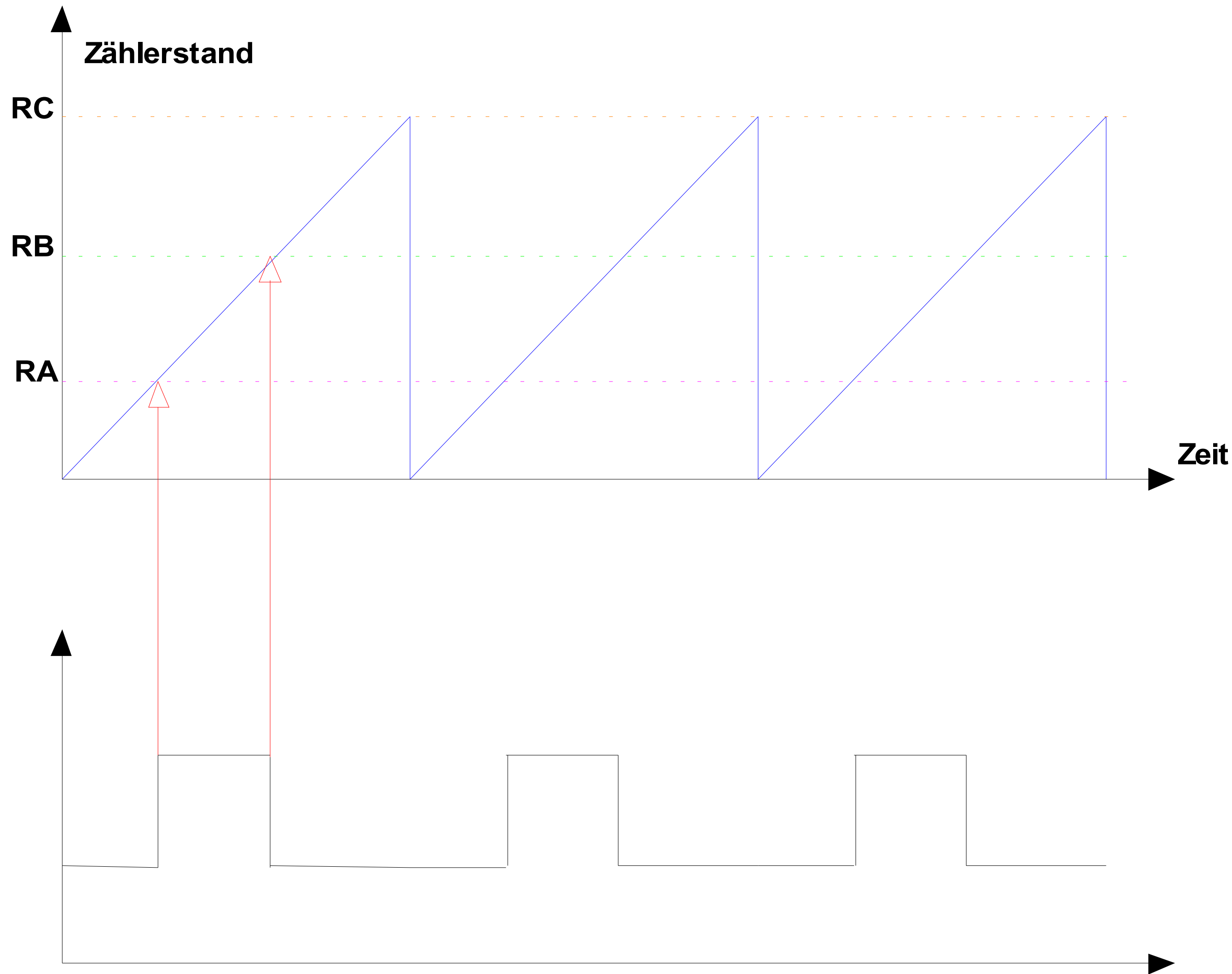
Periodische Interrupts



Erzeugung von Signalen (Compare)



Messung von Zeiten (Capture)



Messung von Zeiten

- Die zu messende Zeit wird mit einem periodischen Signal mit sehr viel schnellerem Takt (Clock) verglichen
- Das Clocksignal wird gezählt
- Das Zeitsignal kann als Torfunktion für das Clocksignal dienen
- Die Flanken des Zeitsignals können zur Speicherung des aktuellen Zählerstands benutzt werden

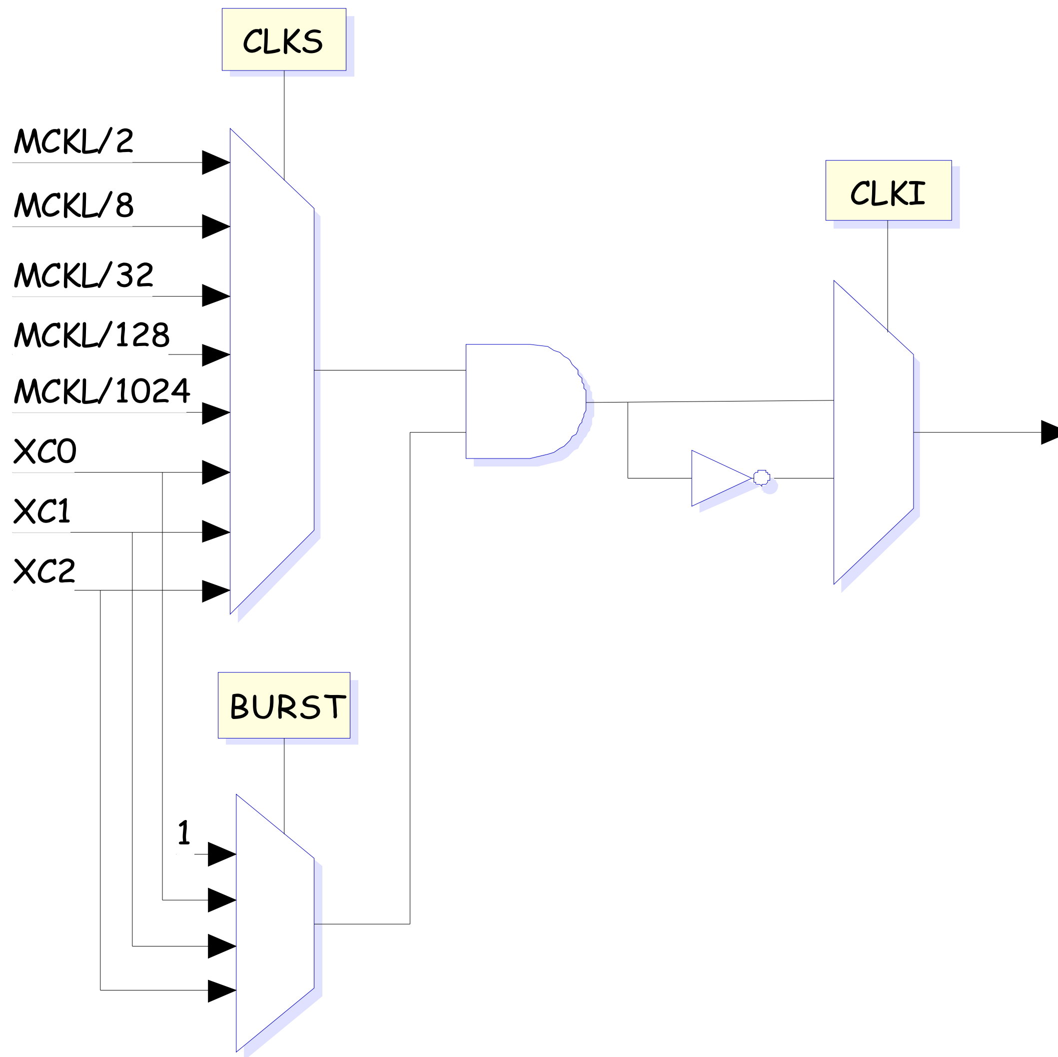
Messung von Frequenzen

- Die Messung von Frequenzen erfolgt indem man die Taktflanken der Frequenz für eine bekannte Zeit zählt
- Die Torzeit muß sehr viel länger sein wie die Periodendauer der Frequenz.

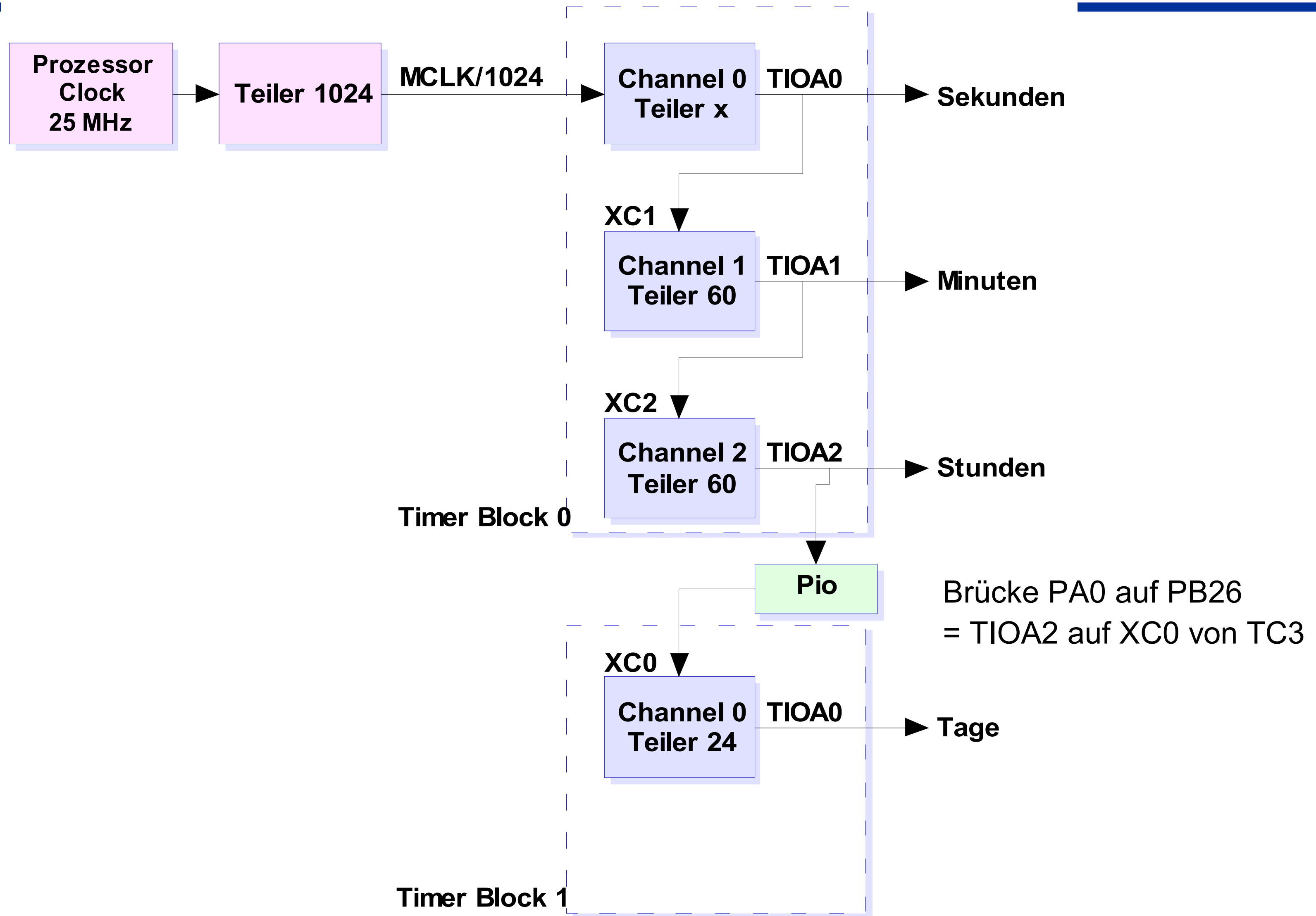
Erzeugung von Analogen Signalen

- Timer können auch zur Erzeugung von analogen Signalen benutzt werden
- Dabei wird das Ausgangssignal pulsweiten moduliert und anschließend durch eine Tiefpassfilterung geglättet
- Beispiel Leuchtstärke LED

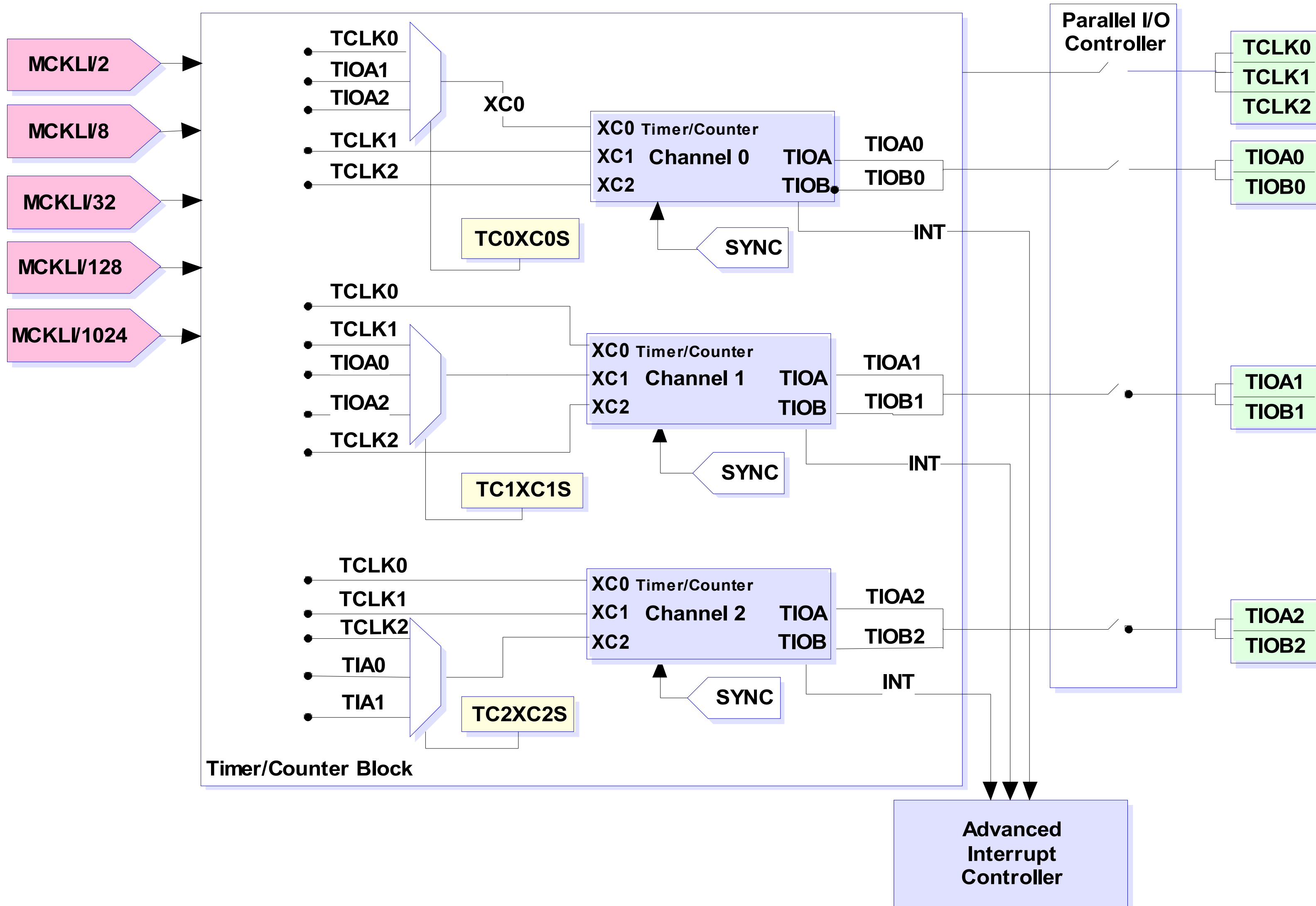
Clock Erzeugung



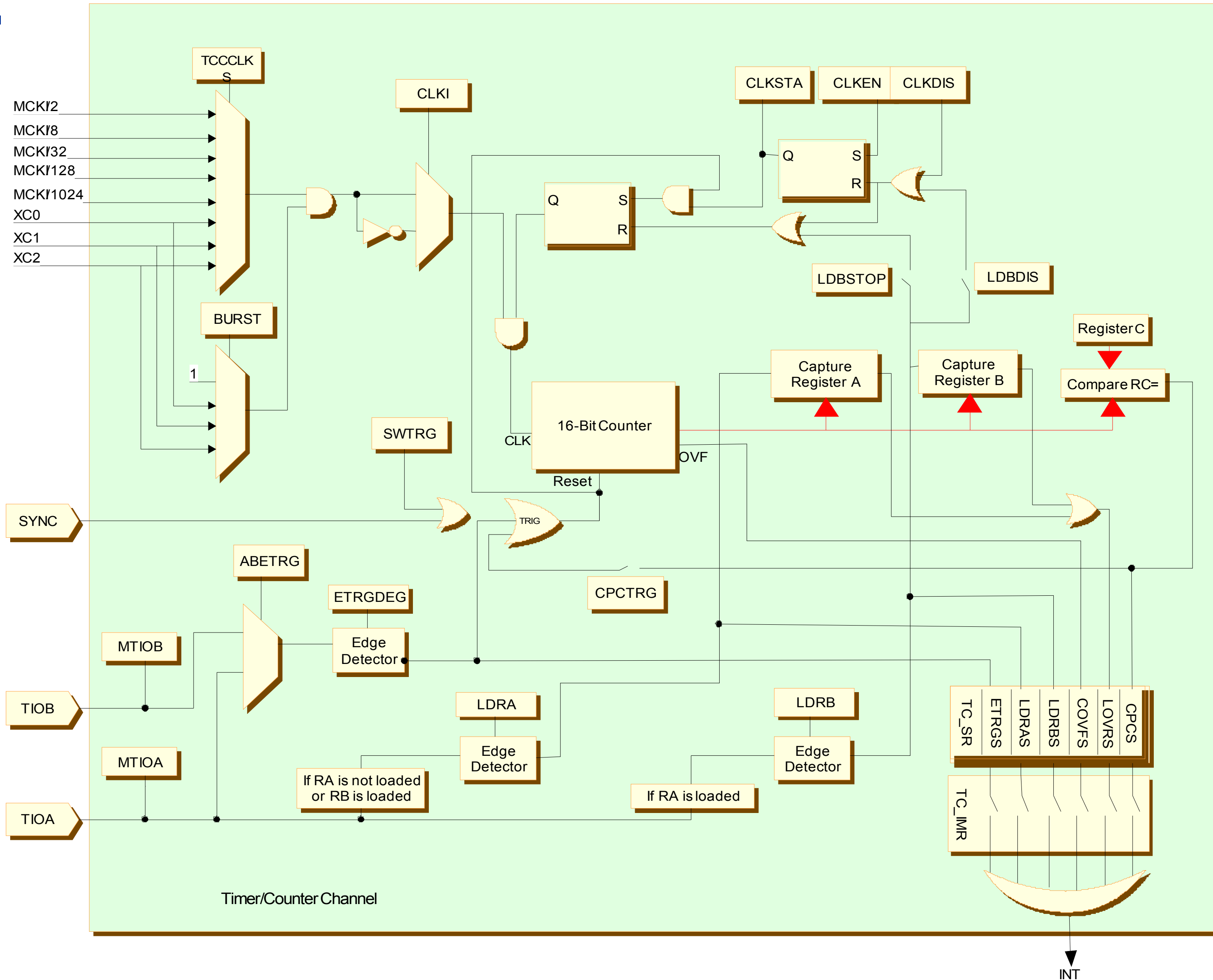
Beispiel: Uhr



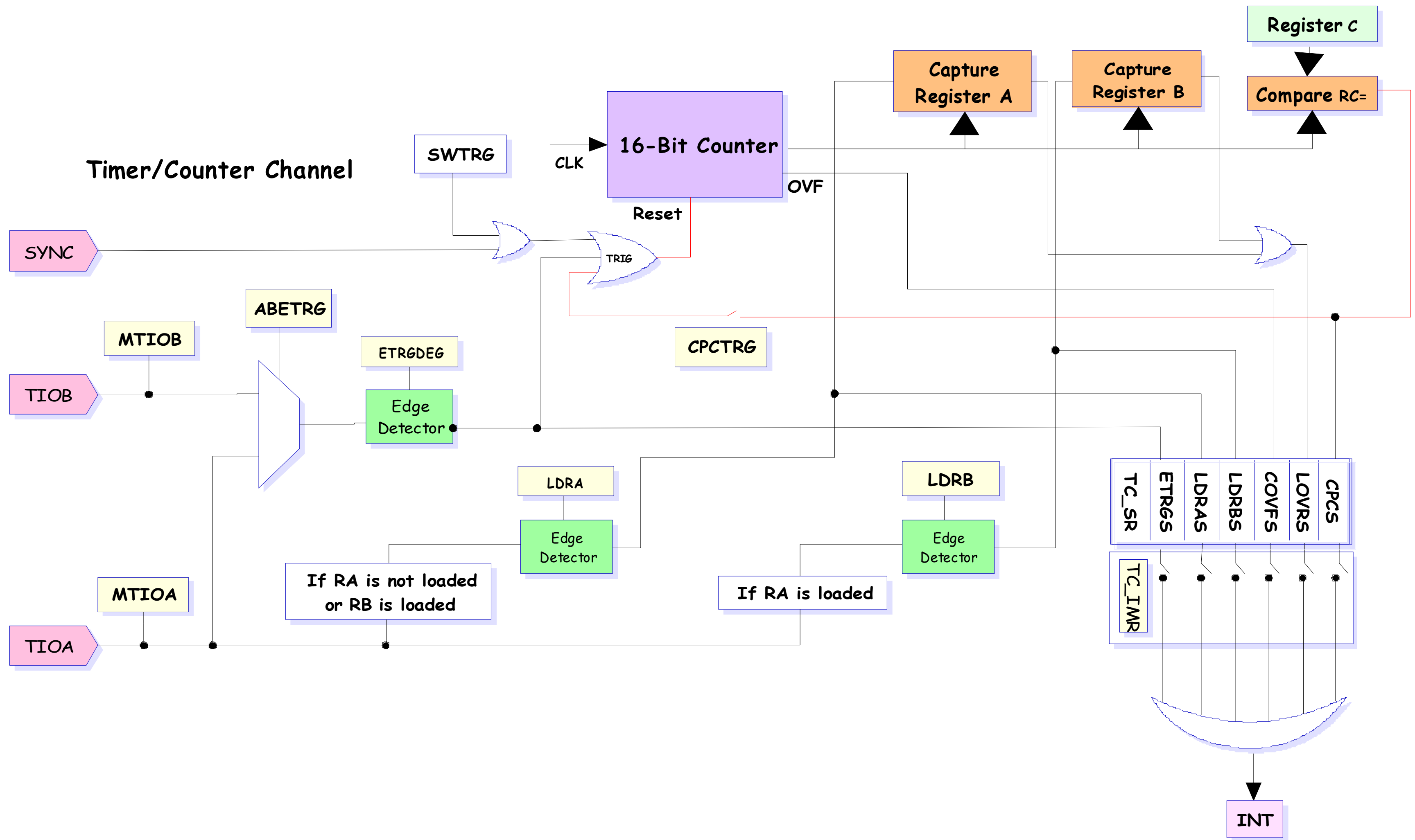
Timer / Counter



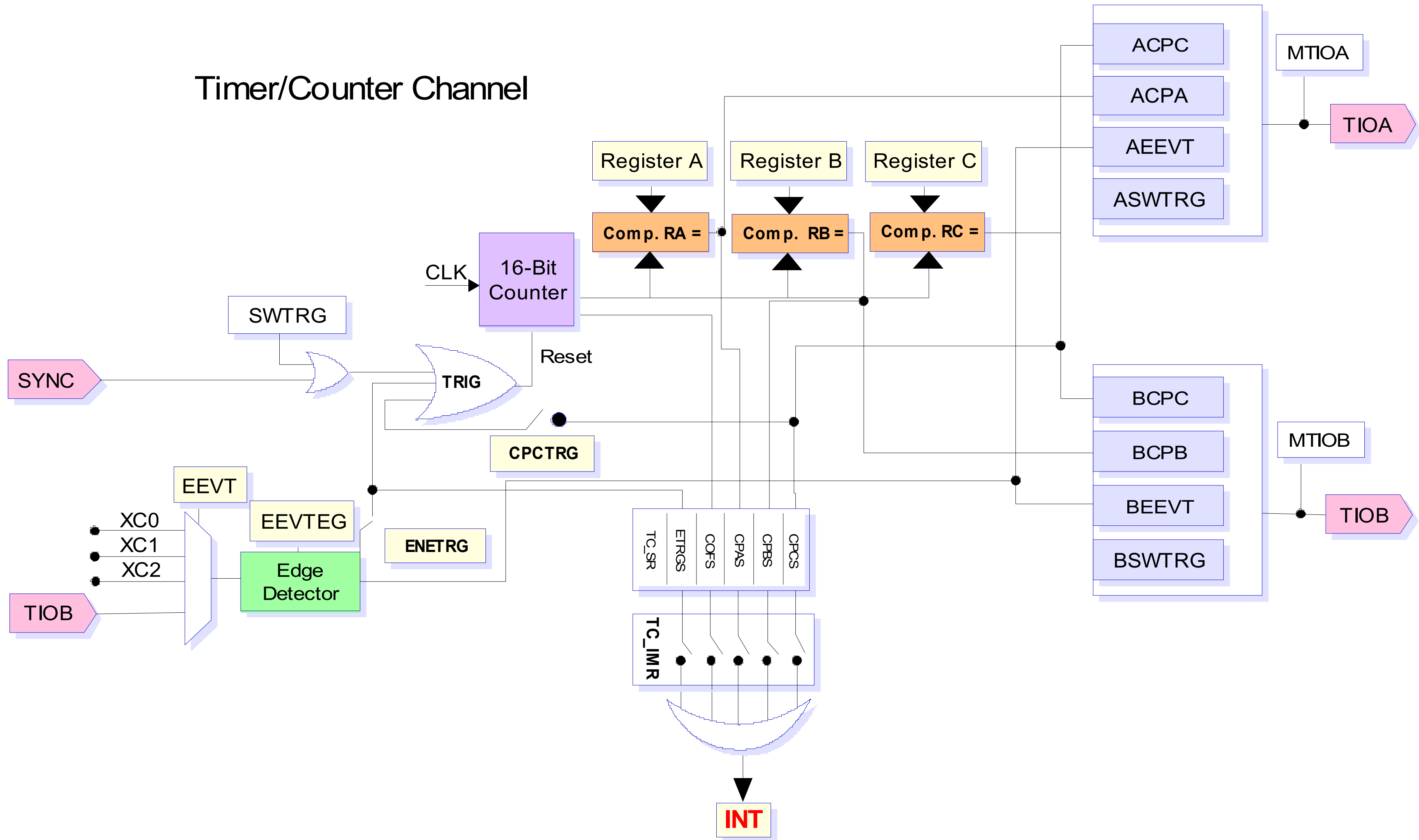
Timer im Capture Mode



Timer im Capture Mode

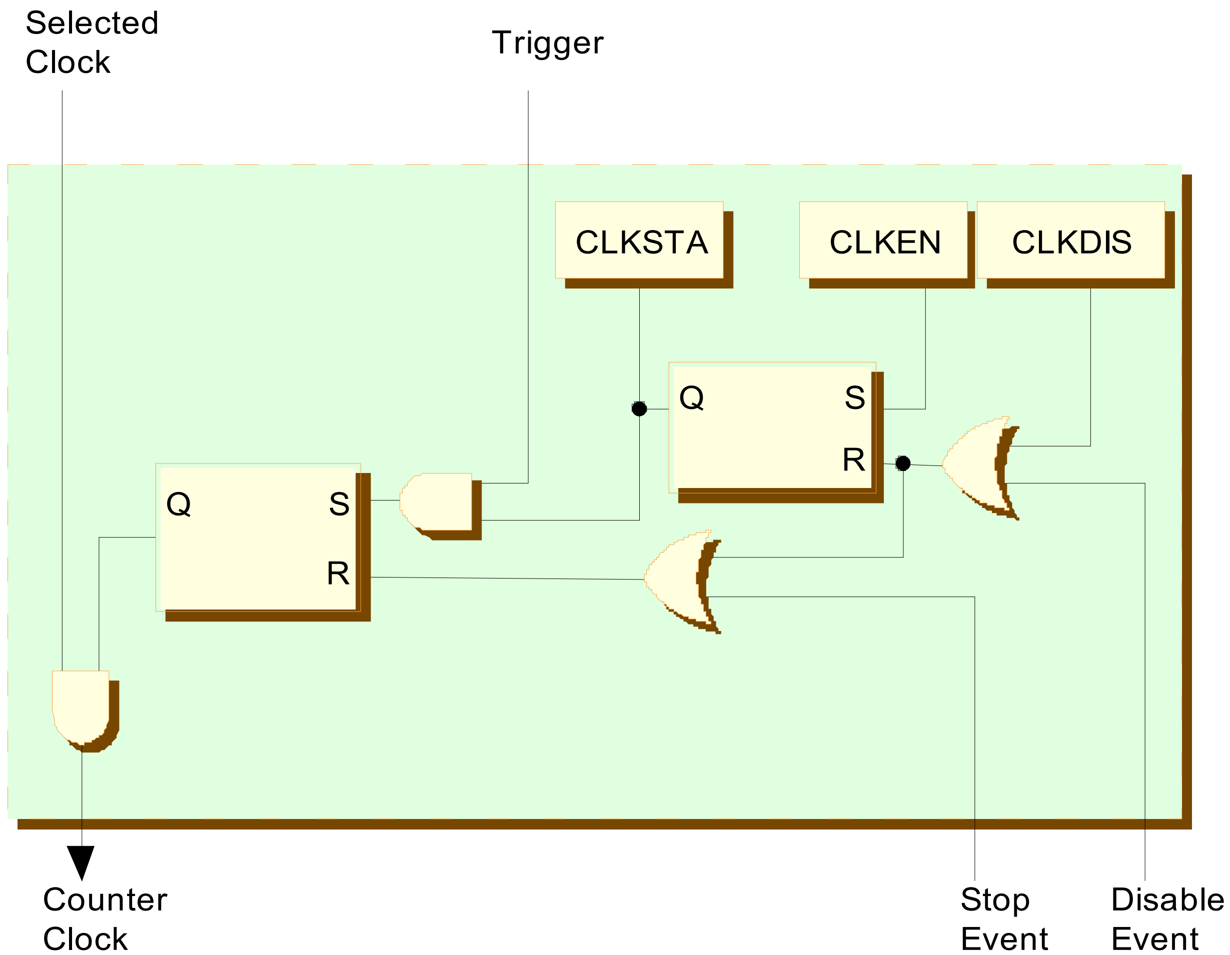


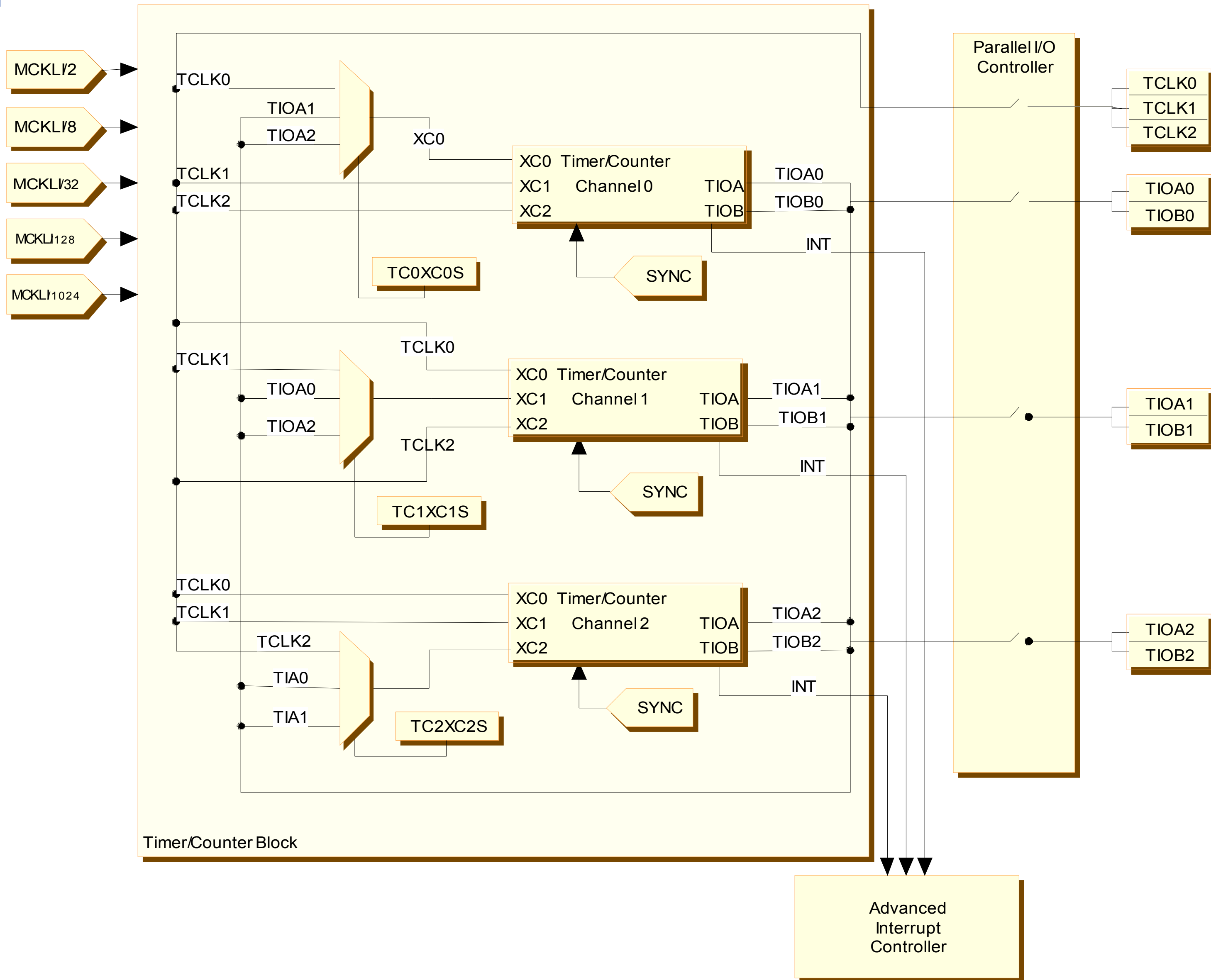
Timer im Compare Mode



Timer im Compare Mode

TCCLKS
MCK/2
MCK/8
MCK/32
MCK/128
MCK/1024
XC0
XC1
XC2
BURST
1
CLKI
CLKSTA
CLKEN
CLKDIS
Q S R
CPCDIS
CPCSTOP
ACPC
ACPA
AEEVT
ASWTRG
MTIOA
TIOA
Register A
Register B
Register C
CompareRA =
CompareRB =
CompareRC =
16-Bit Counter
Reset
Overflow
SWTRG
SYNC
TRIG
CPCTRG
BCPC
BCPB
BEEVT
BSWTRG
MTIOB
TIOB
EEVT
EEVTEG
Edge Detector
ENETRG
TC_SR
ETRGS
COFS
CPAS
CPBS
CPCS
TC_IMR
INT


Selected
Clock
Trigger
CLKSTA
CLKEN
CLKDIS
Q
S
R
Q
S
R
Counter
Clock
Stop
Event
Disable
Event


MCKL/2
MCKL/8
MCKL/32
MCKL/128
MCKL/1024
TCLK0
TIOA1
TIOA2
XC0
TCLK1
TCLK2
XC0 Timer/Counter
XC1 Channel 0
XC2
TIOA
TIOB
TIOA0
TIOB0
INT
TC0XC0S
SYNC
TCLK0
TCLK1
TIOA0
TIOA2
TCLK2
XC0 Timer/Counter
XC1 Channel 1
XC2
TIOA
TIOB
TIOA1
TIOB1
INT
SYNC
TC1XC1S
TCLK0
TCLK1
TCLK2
TIA0
TIA1
XC0 Timer/Counter
XC1 Channel 2
XC2
TIOA
TIOB
TIOA2
TIOB2
INT
SYNC
TC2XC2S
Timer/Counter Block
Parallel I/O Controller
TCLK0
TCLK1
TCLK2
TIOA0
TIOB0
TIOA1
TIOB1
TIOA2
TIOB2
Advanced Interrupt Controller

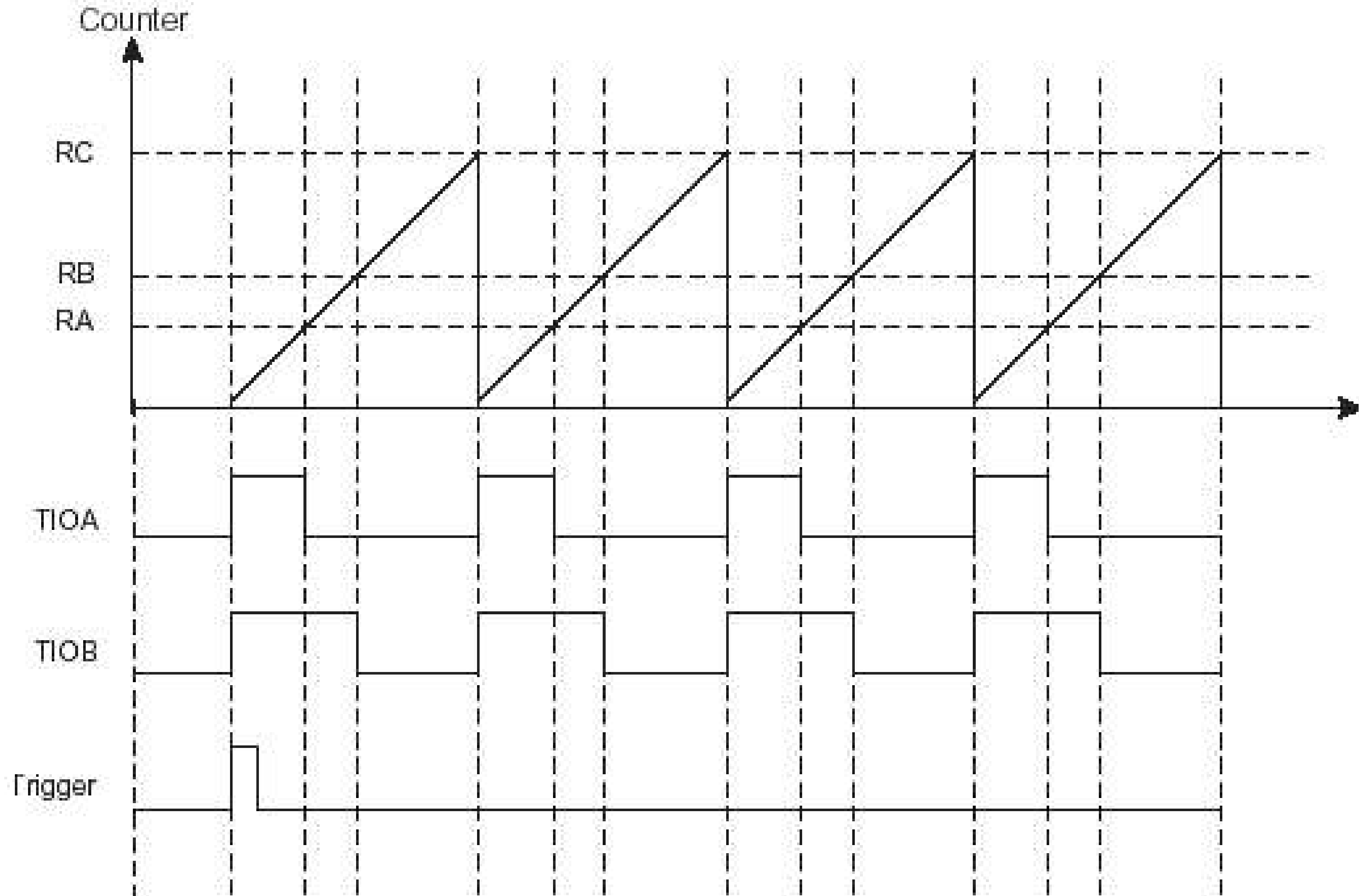
Beispielprogramm Wave (1)

```
int main ( void ) {
/* Enable and clear PIO pin corresponding to TIOA0, TIOB0 and TIOA1 */
    PIO_PER   = (1<<PIOTIOA0) | (1<<PIOTIOA1) | (1<<PIOTIOB0) ;
    PIO_OER   = (1<<PIOTIOA0) | (1<<PIOTIOA1) | (1<<PIOTIOB0) ;
    PIO_CODR = (1<<PIOTIOA0) | (1<<PIOTIOA1) | (1<<PIOTIOB0) ;

    TCB0_BMR = TC_TIOA0XC1 ;                    /* Define XC1 as TIOA0 for channel 0 */
    PIO_PDR = (1<<PIOTIOA1) ;                   /* Define TIOA1 as peripheral */
    TC1_CCR = TC_CLKDIS ;                       /* Disable the timer 1 */
     /* Initialize the mode of the channel 1 */
    TC1_CMR =
        TC_ASWTRG_SET_OUTPUT  |     /* ASWTRG  : software trigger set TIOA */
        TC_ACPC_TOGGLE_OUTPUT |     /* ACPC    : Register C compare toggle TIOA */
        TC_WAVE               |     /* WAVE    : Waveform mode */
        TC_CPCTRG             |     /* CPCTRG  : Register C compare trigger enable */
        TC_BURST_XC1          |     /* BURST   : XC1 is ANDed with the selected clock */
        TC_CLKS_MCK1024 ;           /* TCCLKS  : MCKI / 1024 */
```

Beispielprogramm Wave (2)

```
  TC1_RC = 0x4000 ;          /* Initialize the RC Register value 16.386*/
  TC1_CCR = TC_CLKEN ; /* Enable the clock of the timer */
  TC1_CCR = TC_SWTRG ;              /* Trig the timer */

 PIO_PDR = (1<<PIOTIOA0) ;                        /* Define TIOA0 as peripheral */
 TC0_CCR = TC_CLKDIS ;                            /* Disable the timer 0 */
  /* Initialize the mode of the timer 0 */
 TC0_CMR =
   TC_ASWTRG_SET_OUTPUT   |        /* ASWTRG  : software trigger set TIOA */
   TC_ACPC_TOGGLE_OUTPUT  |        /* ACPC    : Register C compare toggle TIOA */
   TC_WAVE                |        /* WAVE    : Waveform mode */
   TC_CPCTRG              |        /* CPCTRG  : Register C compare trigger enable */
   TC_CLKS_MCK1024 ;               /* TCCLKS  : MCKI / 1024 */

 TC0_RC = 0xA000 ;          /* Initialize the RC Register value  ~ 41.000 */
 TC0_CCR = TC_CLKEN ;   /* Enable the clock of the timer */
 TC0_CCR = TC_SWTRG ;  /* Trigg the timer */
 return(0);
}
```

Pulsweitenmodulation (PWM)



Pulsweitenmodulation (PWM)

maximal counter duration (seconds) = $2^{16} / F_{TC}$ where F_{TC} is in Hz.

counter resolution = $1 / F_{TC}$

Table 1. Maximum Counter Duration for MCK

MCK	5 MHz	10 MHz	20 MHz	33 MHz	66 MHz
MCK/2	26.21ms	13.10ms	6.55ms	3.97ms	1.98ms
MCK/8	104.8ms	52.4ms	26.22ms	14.89ms	7.45ms
MCK/16	419.4ms	209.7ms	104.86ms	63.86ms	31.98ms
MCK/128	1.68s	838.8ms	420.4ms	254.2ms	127.1ms
MCK/1024	13.42s	6.71s	3.36ms	2.03s	1.02s

Gliederung

Einleitung
Power Management
Parallele I/O (PIO)
Interrupt Handling
Timer
- WAVE Mode
- Capture Mode

Softwareinterrupt (SWI)
Serielle Schnittstelle

Software Interrupt (SWI)

Aufgabe des SWI

- Der Software Interrupt ist eine benutzerdefinierte synchrone Programmunterbrechung um aus dem User Mode in den privilegierten Supervisor Mode zu gelangen.
- Im Supervisormode können privilegierte Operationen ausgeführt werden, die im User Mode nicht erlaubt sind
- In vielen Systemen ist der Zugriff auf den Memory Bereich in dem das I/O liegt nur in einem privilegierten Mode (Supervisor Mode oder Interrupt Mode) erlaubt.
- Der Schreibzugriff auf Teile des Prozessor-Statusworts ist geschützt und kann nur in einem privilegierten Mode ausgeführt werden.
  Ein Mode Wechsel oder das Sperren von Interrupts ist daher im User Mode nicht möglich

Aufgabe des Software Interrupts

- Der Speicherbereich, in dem sich die Vektortabelle des Prozessors befindet ist in vielen Systemen geschützt und kann im User Mode nicht beschrieben werden
- Der Zugriff auf die MMU (falls vorhanden) ist nur im privilgierten Mode möglich.

Ausführung eines SWI (1)

Einzelschritte:

- Die CPU kopiert das aktuelle Programm Status Register (CPSR) in das Saved Program Status Register (SPSR_SVC)
  Dadurch werden der augenblickliche Mode, die aktuelle InterruptMaske und die Condition Code Flags gesichert
- Die CPU setzt die Mode Bits im Status Register (CPSR) auf SuperVisor Mode und wechselt damit in diesen Mode.
  Dadurch wird gleichzeitig der Stackpointer und das Linkregister aus diesem Mode in den Registersatz gemapped.

Ausführung eines SWI (2)

Einzelschritte:

- Die CPU setzt das CPSR IRQ disable Bit
  Dadurch wird der normale Interrupt gesperrt und der SWI Handler kann nicht durch einen Interrupt unterbrochen werden. Die Systemroutinen sind selbst dafür verantwortlich den Interrupt freizugeben, wenn sie es wollen.

- Der Wert (PC-4) wird in LR_SVC gespeichert. Dies ist die Programmadresse nach dem SWI Aufruf. (PC zeigt aufgrund des Pipelinings immer auf die ausgeführte Adresse + 8)

- Der Programm Counter wird auf die Adresse 0x08 gesetzt (siehe boot.s)

Das Program Status Register

31			28	27			24	23							16	15							8	7	6	5	4				0
N	Z	C	V	Q				undefined								undefined								I	F	T	mode				

- Condition Code Flags
  - N = Negatives ALU Ergebnis
  - Z = Alu Ergebnis ist Null
  - C = Alu Ergebnis erzeugte Carry
  - V = Alu erzeugte Overflow
- Interrupt Disable Bits
  - I = 1, disables IRQ
  - F = 1, disables FIQ

CPR[4:0]	Mode	Use	Registers
10000	User	Normal User Mode	user
10001	FIQ	Processing fast Interrupts	_fiq
10010	IRQ	Processing standard Interrupts	_IRQ
10011	SVC	Processing Software Interrupts	_SVC
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined Instruction Traps	_und
11111	System	Running privileged operating System tasks	user

Register Banking

User Mode	FIQ	IRQ	SVC	Undef	Abort
r0					
r1					
r2					
r3					
r4					
r5					
r6					
r7					
r8	r8				
r9	r9				
r10	r10				
r11	r11				
r12	r12				
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
r15 (pc)					
cpsr	spsr	spsr	spsr	spsr	spsr

Exception Handling

0x1c	FIQ
0x18	IRQ
0x14	(Reserviert)
0x10	Data Abort
0x0c	Prefetch Abort
0x08	Software Interrupt
0x04	Undefined Instruction
0x00	Reset

Einträge in der Interrupttabelle

- In der Interrupttabelle steht immer ein Befehlscode
- Einfachste Möglichkeit: Sprungbefehl zu Interrupthandler

```
          b      SWI_Handler
```

  - Nachteil: Funktion SWI-Handler muß innerhalb der Reichweite des Sprungbefehls sein, d.h. max 24 Bit Offset (entspricht 16 Mbyte)
- Wenn Offset zu groß:

```
          ldr    pc, _SWI_
_SWI_:           SWI_Handler
```

  - Marke _SWI_ muss sich innerhalb des Offsetbereichs für indirekte Speicherzugriffe (12 Bit: 4096 Bytes) befinden

Rückkehr von einem SW Interrupt (1)

- Das alte Programm Status Register muß wieder hergestellt werden, so daß Bedingungen, Interrupt Status und alter Mode wieder vorhanden sind, d.h. SPSR_SVC muß wieder zu CPSR werden
- Es muß ein Rücksprung in das aufrufende Programm erfolgen, indem LR_SVC nach PC kopiert wird.
- Beide Aktionen müssen in einem (ununterbrechbaren) Befehl erfolgen

Rückkehr von einem SW Interrupt (2)

- Variante 1 für Rücksprung aus SWI, wenn die Rücksprungadresse noch in LR steht
  - movs   pc, lr
  - In Privilegierten Modi, wenn der PC das Zielregister ist, führt ein gesetztes 's' Flag zu einem Rückspeichern des Prozessorstatusregisters

- Variante 2, wenn sich die Rückprungadresse auf dem Stack befindet
  - ldmfd sp!, {...,pc}^

Software Interrupt

31 … 28	27				… 0
cond	1	1	1	1	SWI Nummer / 24 Bit immediate

- Syntax:

  swi{<cond>} <SWI Nummer>

- Der SWI Interrupt Handler kann die SWI Nummer lesen und damit entscheiden, welche Operation ausgeführt werden soll

- Der Software Interrupt setzt LR_SVC auf PC-4, das ist der Befehl nach dem Softwareinterrupt. Durch einen Zugriff auf LR-4 kann man daher den Befehlscode des SWI in ein Register laden und die SWI Nummer extrahieren

Zugriff auf die Nummer des SWI Befehls

- ldr r0, [lr, #-4] @ Befehlscode wird geladen
- bic r0, r0, #0xff000000 @ Obere 8 Bits werden maskiert

Ein einfacher SWI Handler

```
    .global SWIHandler
    .text
SWIHandler:
    stmfd   sp!, {lr}                @ Retten von Ruecksprungadresse und r0
    ldr     ip, [lr, #-4]            @ Laden des Programmcodes
    bic     ip, ip, #0xff000000      @ Ausmaskieren der SWI Nummer
    ldr     lr, =SWIJumpTable
    ldr     ip,[lr, ip, LSL #2]      @ Laden der Funktionsadresse nach ip
    mov     lr, pc                   @ indirekter Unterprogrammaufruf
    mov     pc,ip
    ldmfd   sp!,{pc}^
SWIJumpTable:
    .word   putchar
    .word   getchar
    .word   init_ser
.end
```

Software Interrupt

Verbesserungsmöglichkeiten für SWI Handler

- Reentrant Fähigkeiten, d.h. in einem SW Interrupt kann ein weiterer SW Interrupt aufgerufen werden
- Überprüfen der SWI Nummer auf Gültigkeit
- Übergabe der SWI Nummer an ausführendes Programm
- Freie Aufteilung der SWI Nummern (Bereiche und Lücken)
- Chaining von Interrupthandlern

Gliederung

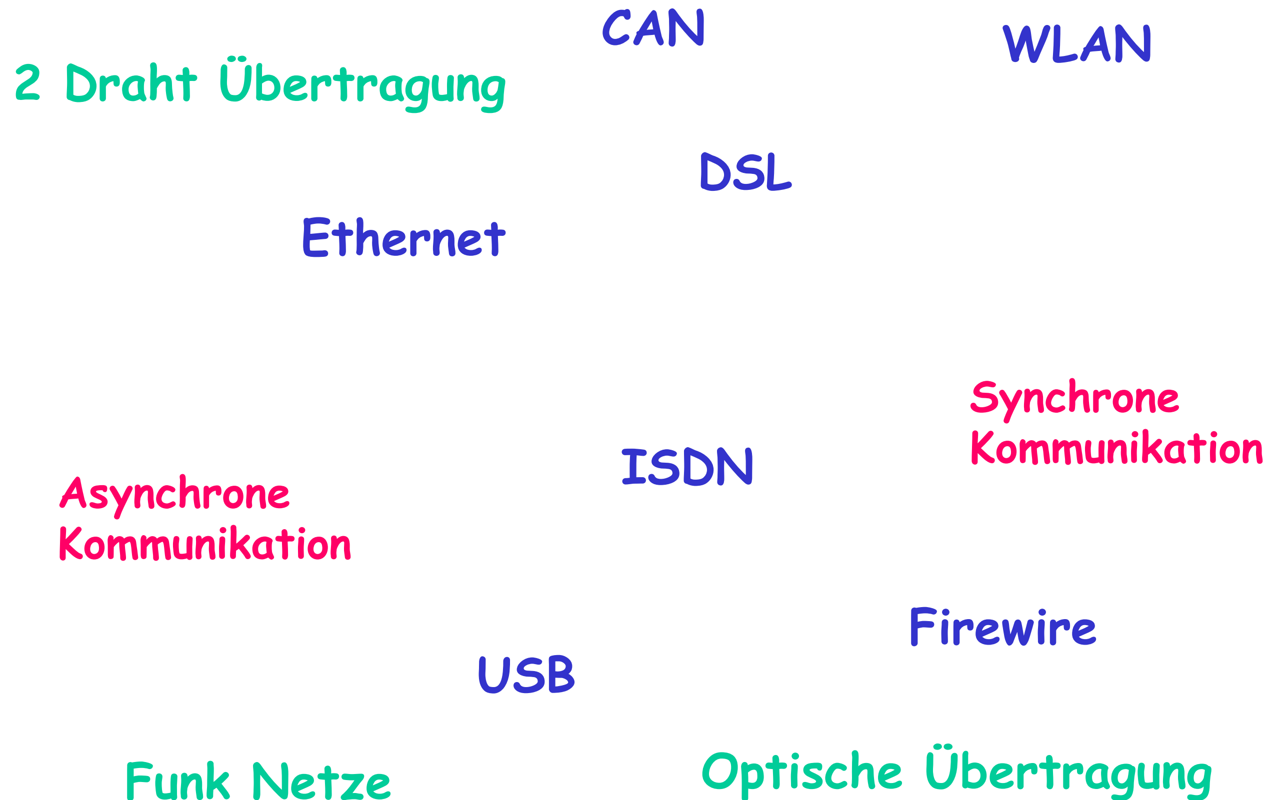
FHD

Einleitung
Power Management
Parallele I/O (PIO)
Interrupt Handling
Timer
    WAVE Mode
    Capture Mode
Softwareinterrupt (SWI)
Serielle Schnittstelle

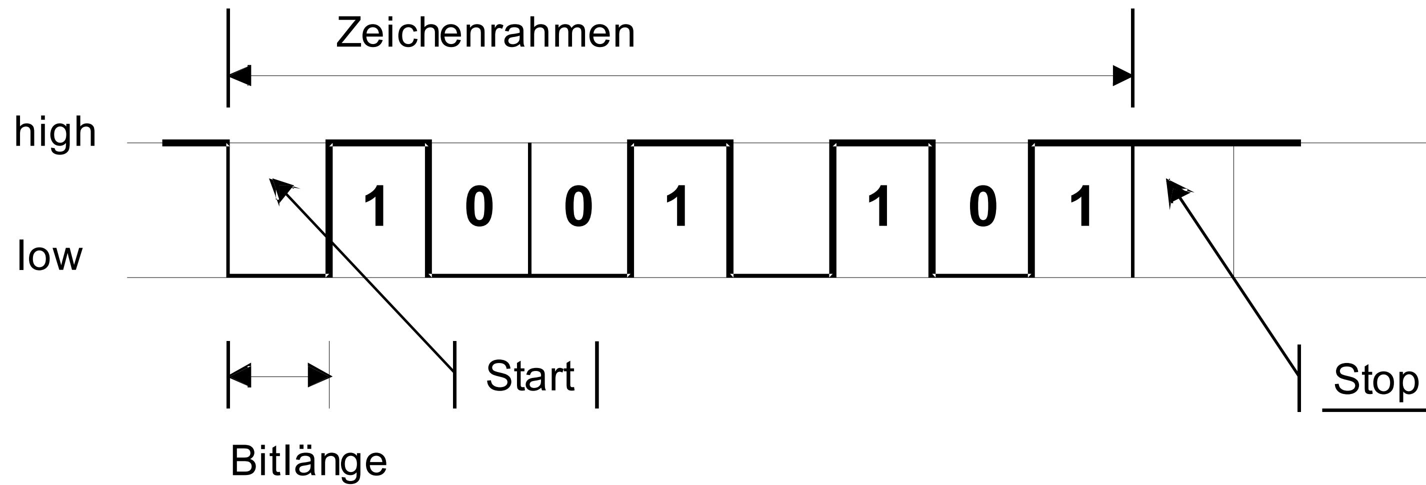
Gliederung

- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)

Serielle Schnittstellen

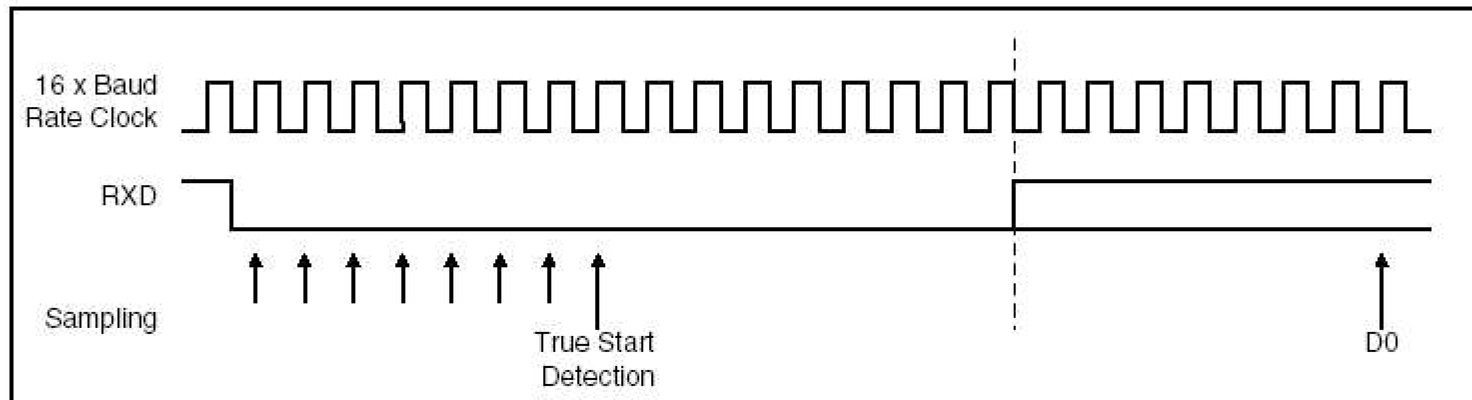


Asynchrone Datenübertragung



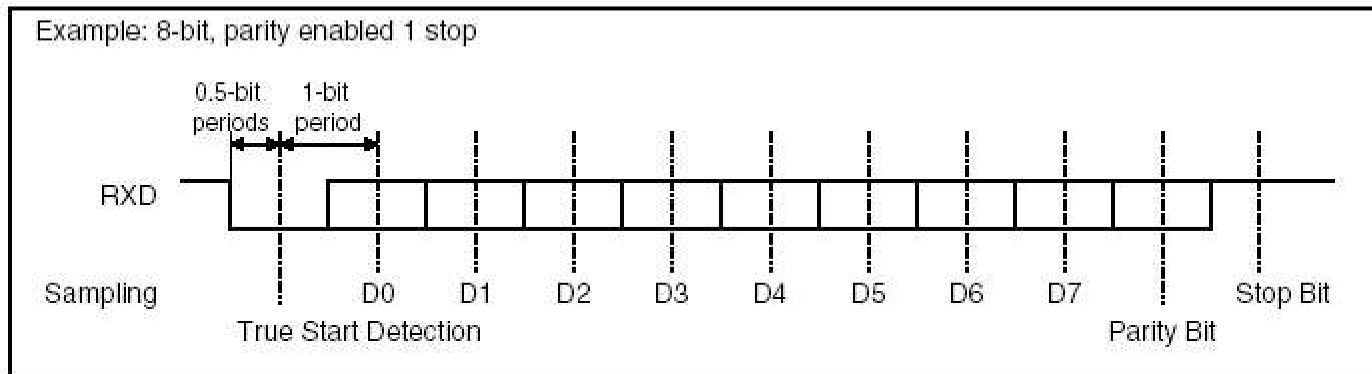
Asynchroner Mode

Start Bit Erkennung

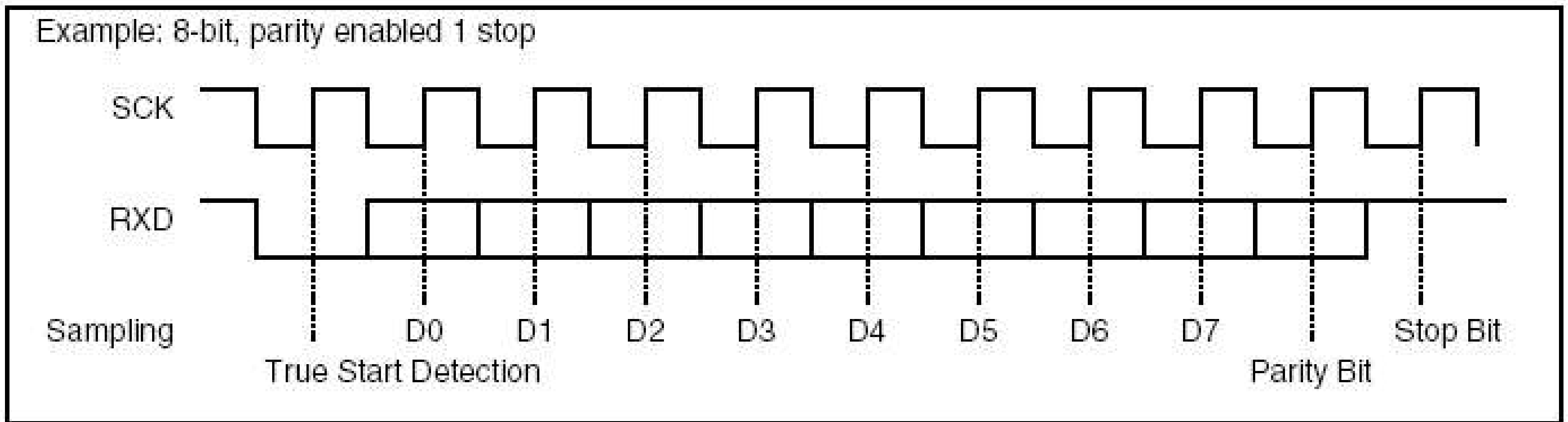


Asynchroner Mode

Zeichenerkennung



Synchroner Mode



Grundaufgabe serieller Kommunikation

- Daten serialisieren und senden
- Daten empfangen und abspeichern

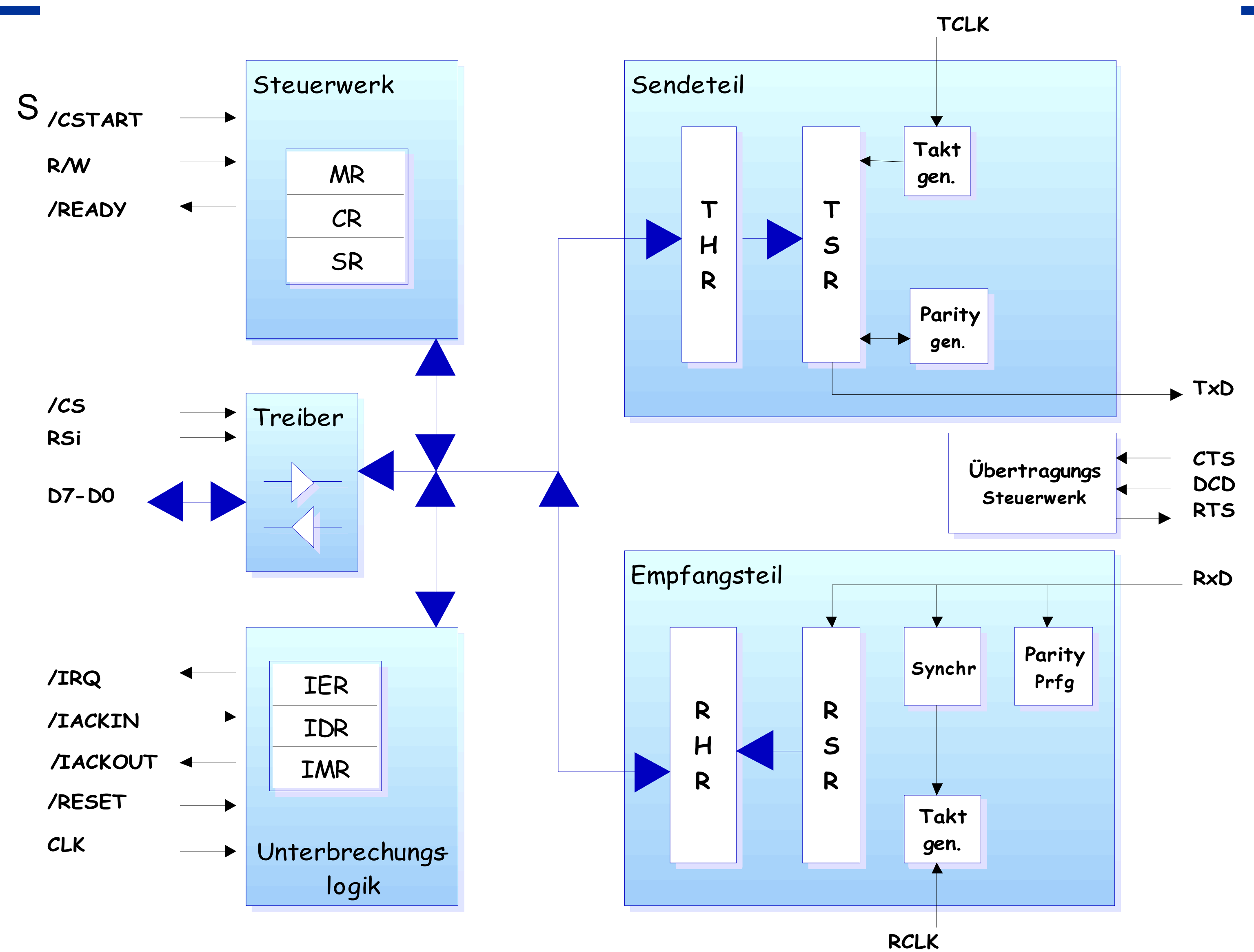
Grundfunktionen serieller Kommunikation

- Schnittstelle initialisisieren
- Nachricht schicken
- Nachricht empfangen

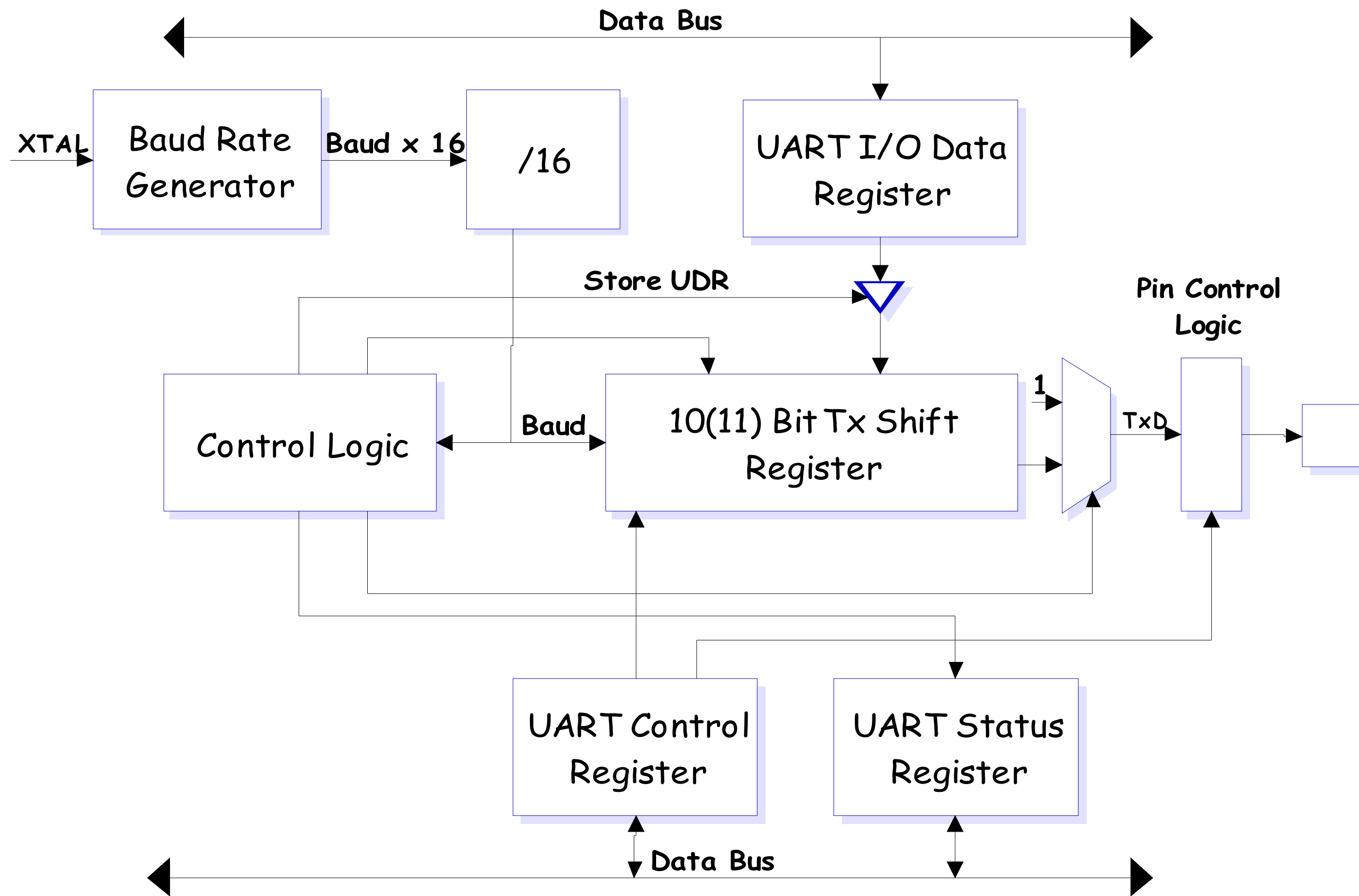
Einfaches API für serielle Schnittstelle

- serial_init()
- void putchar(char c)
- char getchar(void)
- bool write(char* buffer, int len)
- bool read(char* buffer, int len)

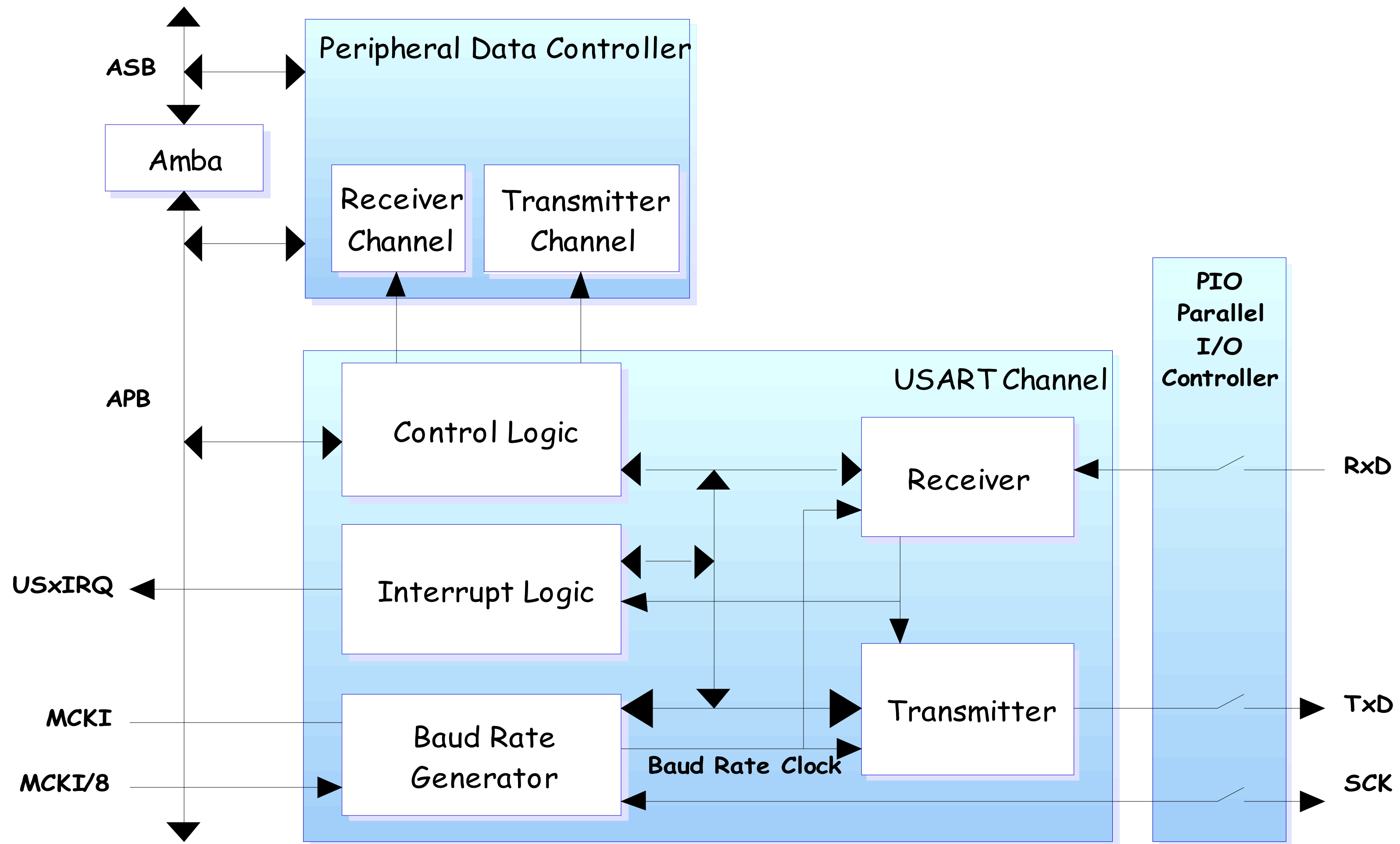
Struktur eines asynchronen seriellen Bausteins



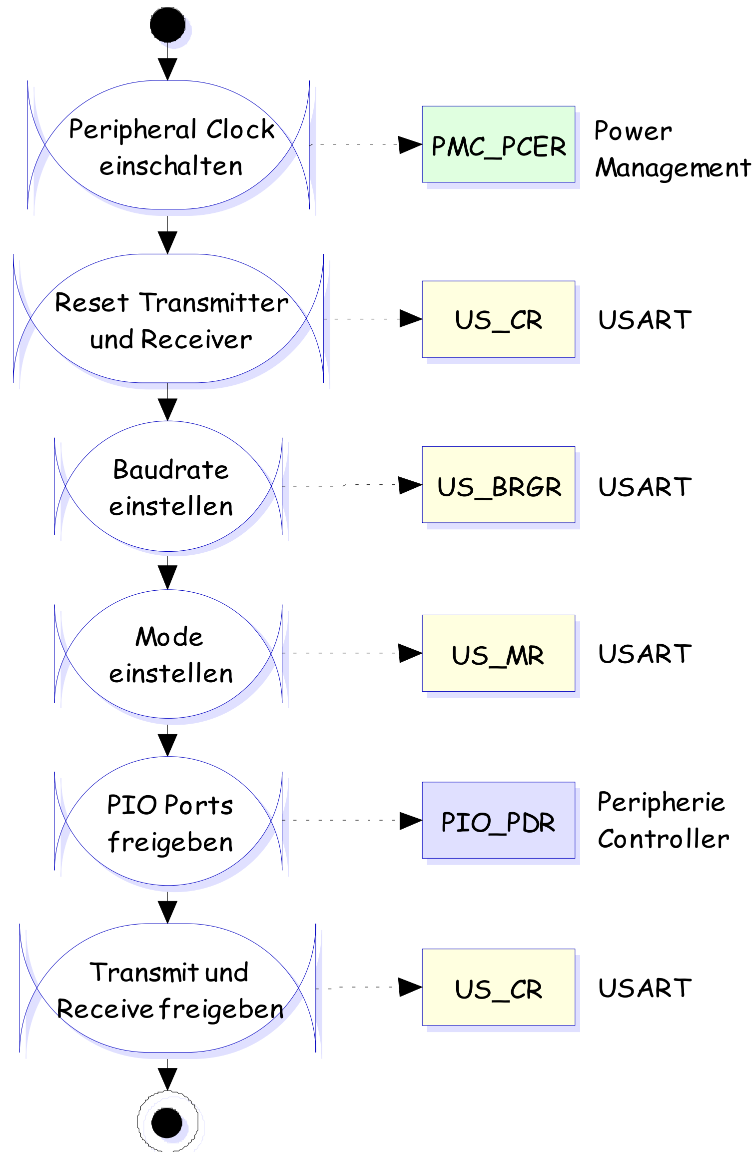
Innerer Aufbau des Sendebausteins



USART Struktur mit DMA Prozessor

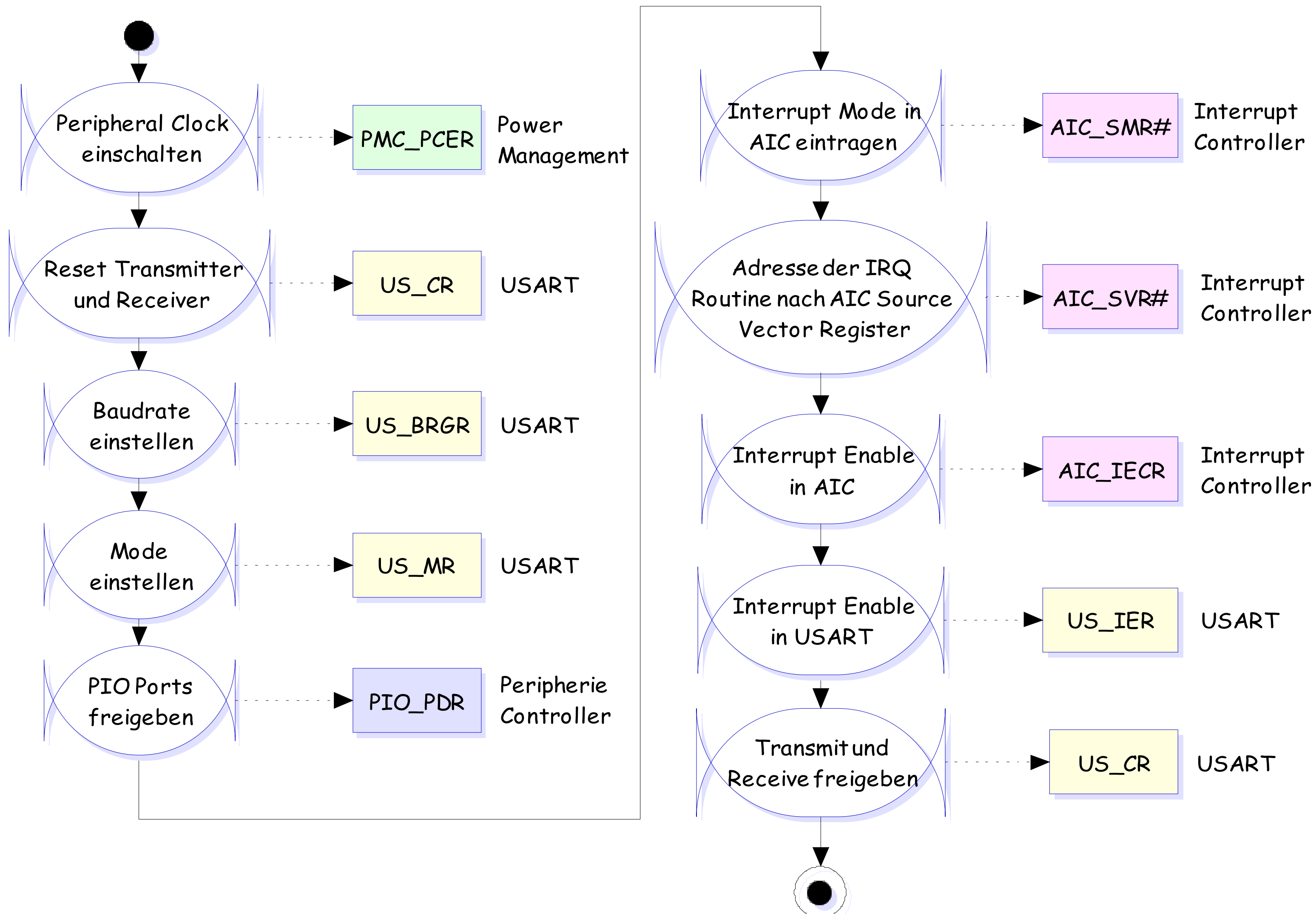


USART Initialisierung



Initialisierung eines Interrupts

Beispiel: serielle Schnittstelle



Initialisierungstabelle (1)

```
L1:
@ Clock für USART und PIOA einschalten
  .word PMC_BASE+PMC_PCER, 0
@ Reset Receiver, Transmitter und StatusBits
  .word   USART0_BASE+US_CR, 0
@ Peripheral Funktion in PIO erlauben
  .word   PIOA_BASE+PIO_PDR, 0
@ Mode 8Bit, No Parity, 1 StopBit
  .word USART0_BASE+US_MR, 0
@ Baudrate einstellen auf 38400 Baud
  .word USART0_BASE+US_BRGR,        0
@ Flankentriggerung als Interruptmode und Piorität
  .word AIC_BASE+AIC_SMR+(US0_ID*4), 0
@ Adresse der Interrupt Service Routine
  .word AIC_BASE+AIC_SVR+(US0_ID*4), 0
```

Initialisierungstabelle (2)

```
@ Adresse des Spurious Interrupt Handlers
    .word AIC_BASE+AIC_SPU, 0
@ Alle anstehenden Interrupts löschen
    .word AIC_BASE+AIC_ICCR, 0
@ Interrupt für US0 freigeben
    .word AIC_BASE+AIC_IECR, 0
@ Transmit Interrupt in USART erlauben
    .word USART0_BASE+US_IER, 0
@ Transmitter und Receiver freigeben
    .word USART0_BASE+US_CR,  0
L1_END:
```

USART Register Anordnung

Offset	Register	Name	Access	Reset State
0x00	Control Register	US_CR	Write only	-
0x04	Mode Register	US_MR	Read/write	0
0x08	Interrupt Enable Register	US_IER	Write only	-
0x0C	Interrupt Disable Register	US_IDR	Write only	-
0x10	Interrupt Mask Register	US_IMR	Read only	0
0x14	Channel Status Register	US_CSR	Read only	0x18
0x18	Receiver Holding Register	US_RHR	Read only	0
0x1C	Transmitter Holding Register	US_THR	Write only	-
0x20	Baud Rate Generator Register	US_BRGR	Read/write	0
0x24	Receiver Time-out Register	US_RTOR	Read/write	0
0x28	Transmitter Time-guard Registe	US_TTGR	Read/write	0
0x2C	Reserved	-	-	-
0x30	Receive Pointer Register	US_RPR	Read/write	0
0x34	Receive Counter Register	US_RCR	Read/write	0
0x38	Transmit Pointer Register	US_TPR	Read/write	0
0x3C	Transmit Counter Register	US_TCR	Read/Write	0

Offset	Register	Name	Access
0x00	Control Regis	US_CR	Write only
0x04	Mode Registe	US_MR	Read/write
0x08	Interrupt Ena	US_IER	Write only
0x0C	Interrupt Disa	US_IDR	Write only

Abbildung in Strukturen (1)

Abbildung der Register in einer C-Struktur

```
struct USART {
    volatile unsigned int  US_CR;
    volatile unsigned int  US_MR;
    volatile unsigned int  US_IER;
    volatile unsigned int  US_IDR;
    volatile unsigned int  US_IMR;
    volatile unsigned int  US_CSR;
    volatile unsigned int  US_RHR;
    volatile unsigned int  US_THR;
    volatile unsigned int  US_BRGR;
    volatile unsigned int  US_RTOR;
    volatile unsigned int  US_TTGR;
    volatile unsigned int  dummy;
    volatile unsigned int  US_RPR;
    volatile unsigned int  US_RCR;
    volatile unsigned int  US_TPR;
    volatile unsigned int  US_TCR;
}
```

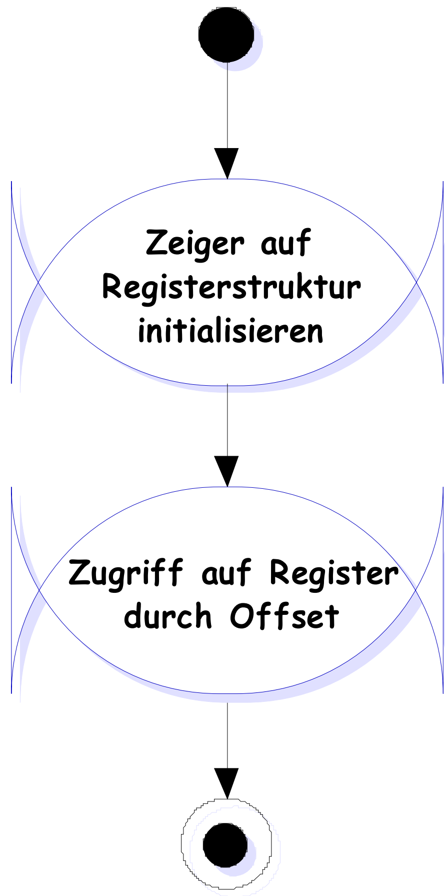
Abbildung in Strukturen (2)

FHD

Definitionen für Assembler

```
#define US_CR  0x00  // Control register
#define US_MR  0x04  // Mode register
#define US_IER 0x08  // Interrupt enable register
#define US_IDR 0x0C  // Interrupt disable register
#define US_IMR 0x10  // Interrupt mask register
#define US_CSR 0x14  // Channel status register
#define US_RHR 0x18  // Receive holding register
#define US_THR 0x1C  // Transmit holding register
#define US_BRG 0x20  // Baud rate generator
#define US_RTO 0x24  // Receive time out
#define US_TTG 0x28  // Transmit timer guard
```

Anwendung der Strukturen




```
ldr     r0, US_BASE_USART
ldr     r1, US_INIT_CR
str     r1, [r0, #US_CR]
...
US_BASE_USART:
    .word 0xfffc0000
US_INI_CR:
    .word ...
```

USART Control Register

15	14	13	12	11	10	9	8
-	-	-	SENDA	STTTO	STPBRK	STTBRK	RSTSTA
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	-	-

- RSTRX: Reset Receiver
- RSTTX: Reset Transmitter
- RXEN: Receiver Enable
- RXDIS: Receiver Disable
- TXEN: Transmitter Enable
- TXDIS: Transmitter Disable

- RSTSTA: Reset Status Bits
- STTBRK: Start Break
- STPBRK: Stop Break
- Start Time-out
- SENDA: Send Address

Weitere Definitionen

```
#define US_CR_RxRESET
    (1<<2)
#define US_CR_TxRESET
    (1<<3)
#define US_CR_RxENAB  (1<<4)
#define US_CR_RxDISAB (1<<5)
#define US_CR_TxENAB  (1<<6)
#define US_CR_TxDISAB (1<<7)
#define US_CR_RSTATUS
    (1<<8)
```

```
#define US_MR_CLOCK     4
#define US_MR_LENGTH    6
#define US_MR_SYNC      8
#define US_MR_PARITY    9
#define US_MR_STOP
        12
#define US_MR_MODE      14
#define US_MR_MODE9     17
#define US_MR_CLKO
        18
```

USART Mode Register

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	CLKO	MODE9	-
15	14	13	12	11	10	9	8
CHMODE		NBSTOP		PAR			SYNC
7	6	5	4	3	2	1	0
CHRL		USCLKS		-	-	-	-

USCLKS		Selected Clock
0	X	MCKI
0	1	MCKI/8
1	X	External (SCK)

CHRL		Character Length
0	0	Five bits
0	1	Six bits
1	0	Seven bits
1	1	Eight bits

PAR			Parity Type
0	0	0	Even parity
0	0	1	Odd parity
0	1	0	Parity forced to 0 (space)
0	1	1	Parity forced to 1 (mark)
1	0	x	No parity
1	1	x	Multi-drop mode

USART Mode Register

NBSTOP		Asynchronous (SYNC = 0)	Synchronous (SYNC = 1)
0	0	1 stop bit	1 stop bit
0	1	1.5 stop bits	Reserved
1	0	2 stop bits	2 stop bits
1	1	Reserved	Reserved

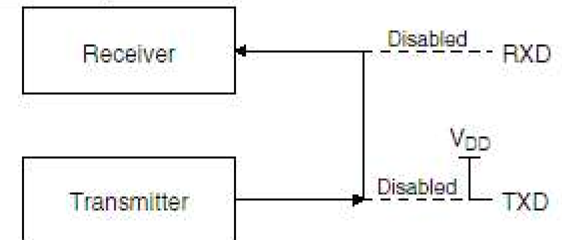
NBSTOP Asynchronous (SYNC = 0) Synchronous (SYNC = 1)
0 0 1 stop bit 1 stop bit
0 1 1.5 stop bits Reserved
1 0 2 stop bits 2 stop bits
CHMODE Mode Description
0 0 Normal Mode
The USART Channel operates as an Rx/Tx USART.
0 1 Automatic Echo
Receiver Data Input is connected to TXD pin.

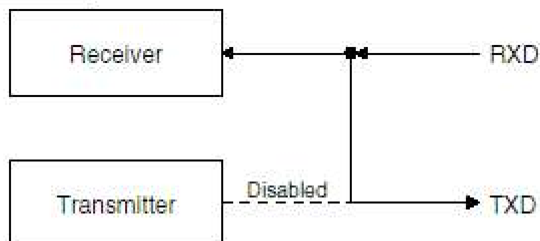
CHMODE		Mode Description
0	0	Normal Mode The USART Channel operates as an Rx/Tx USART.
0	1	Automatic Echo Receiver Data Input is connected to TXD pin.
1	0	Local Loopback Transmitter Output Signal is connected to Receiver Input Signal.
1	1	Remote Loopback RXD pin is internally connected to TXD pin.

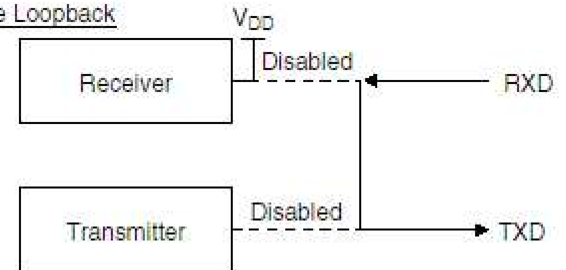
USART Mode Register

CHMODE		Mode Description
0	0	Normal Mode The USART Channel operates as an Rx/Tx USART.
0	1	Automatic Echo Receiver Data Input is connected to TXD pin.
1	0	Local Loopback Transmitter Output Signal is connected to Receiver Input Signal.
1	1	Remote Loopback RXD pin is internally connected to TXD pin.

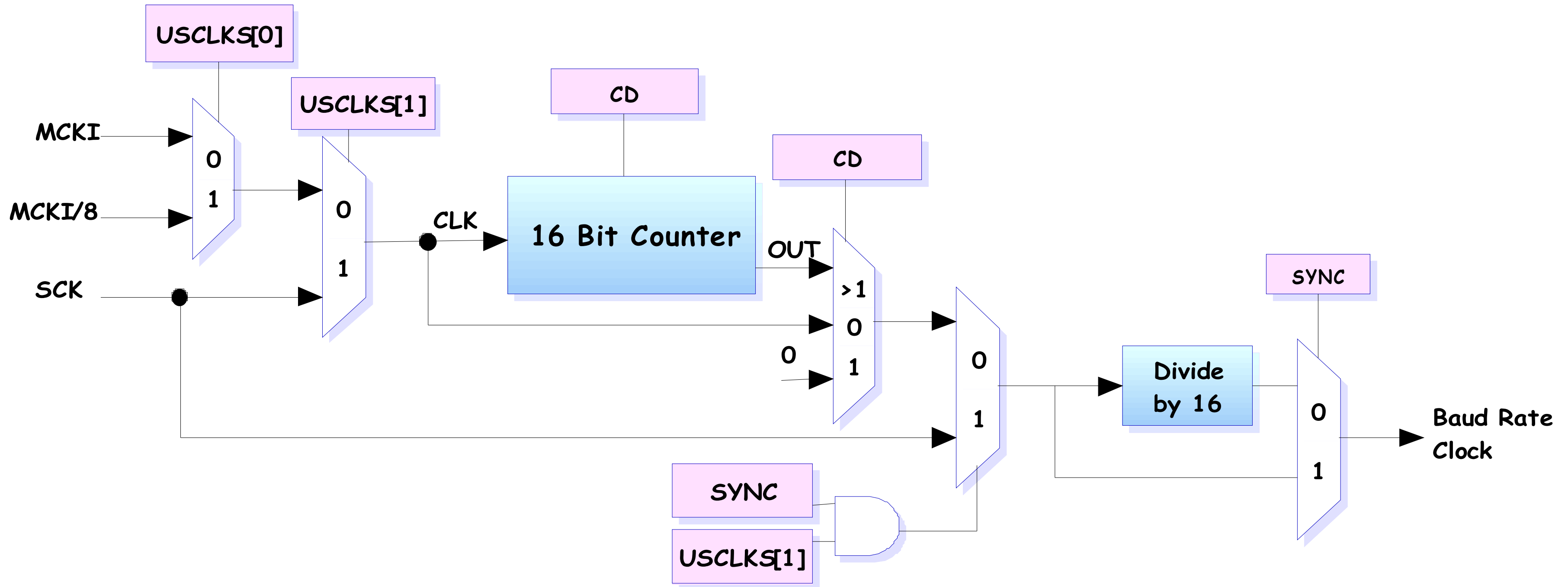
CHMODE Mode Description
0 0 Normal Mode
The USART Channel operates as an Rx/Tx USART.
0 1 Automatic Echo
Receiver Data Input is connected to TXD pin.







Baud Rate Generator



asynchrone Übertragung:

$$\text{Baud Rate} = \frac{\text{Selected Clock}}{16 \times \text{CD}}$$

synchrone Übertragung:

$$\text{Baud Rate} = \frac{\text{Selected Clock}}{\text{CD}}$$

Initialisierung des Baudrate Generator

- Generator wird durch MCKI (Machine Clock) oder MCKI/8 getrieben
- Berechnung des Teilerfaktors für asynchrone Kommunikation
  - BRGR = MCKI / (16 * BAUD)
- Beispiel:
  - MCKI = 25e6
  - BAUD = 38400
  - BRGR = 41 (40.69)

USART Channel Status Register

31	30	29	28	27	26	25	24
COMMRX	COMMTX	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- RXRDY: Receiver Ready
- TXRDY: Transmitter Ready
- RXBRK: Break Received/End of Break
- ENDRX: End of Receive Transfer
- ENDTX: End of Transmit Transfer
- TIMEOUT: Receiver Time-out
- TXEMPTY: Transmitter Empty

- OVRE: Overrun Error
- FRAME: Framing Error
- PARE: Parity Error
- COMMTX: ARM7TDMI ICE Debug Communication Channel Transmit Status
- COMMRX: ARM7TDMI ICE Debug Communication Channel Receive Status

31 30 29 28
COMMRX COMMTX - -
23 22 21 20
- - - -
15 14 13 12

Initialisierung in C und Assembler

```
serial_init:
// Reset device base->US_CR = (US_TXDIS | US_RXDIS);
    ldr r2, .L_USART
    mov     ip, #0xA0
    str     ip, [r2]
// 8-1-no parity: base->US_MR = (US_CLKS_MCK | US_CHRL_8 |
//                               US_PAR_NO | US_NBSTOP_1);
    mov     ip, #0x8C0   // 2240
    str     ip, [r2, #4]
// baud rate setting  base->US_BRGR = 25000000/(16*38400);
    mov     ip, #41      // genau: 40.69
    str     ip, [r2, #0x20]
// Enable RX and TX  base->US_CR = (US_TXEN | US_RXEN);
    mov     ip, #0x50
    str     ip, [r2]
.L_USART:
    .word   0xfffc0000
```

Initialisierung (Alternative)

```
init_ser:
        stmfd       sp!, {r0-r3, lr}        @ Register retten
        adr         r0,L1
        adr         r1,L1_end
init_ser_loop:
        ldmia       r0!, {r2-r3}
        cmp         r0, r1
        str         r3, [r2]
        bne         init_ser_loop
        ldmfd       sp!, {r0-r3, pc}        @ Rücksprung
L1:
        .word   USART0_BASE+US_CR, 0xa0
        .word   USART0_BASE+US_MR, 0x8c0
        .word   USART0_BASE+US_BRGR, 0x29
        .word   USART0_BASE+US_CR, 0x50
L1_end:
```

getch Funktion in Assembler (blockierend)

```
.L_USART:
        .word   0xfffc0000          @ Basisadresse USART
// ….
getch:
        ldr r2, .L_USART
.L8:
        ldr      r0, [r2, #20]      @ Laden des Channel Status Reg.
        tst      r0, #1             @ Receiver Ready Bit gesetzt ?
        ldrne    r1, [r2, #24]      @ Laden des Receiver Holding Reg.
        andne    r0, r1, #255       @ Maskieren
        movne    pc, lr
        b        .L8
.Lfe3:
        .size    getch,.Lfe3-getch
        .ident   "GCC: (GNU) 3.2"
```

Interrupt an der USART0

isr_us0: (Kern einer Interrupt-Serviceroutine)

```
// ….
        ldr     r0, =USART0_BASE
        ldr     r1, [r0, #US_CSR]
        tst     r1, #US_RXRDY
        blne    isr_getch
// …..
        ldr     r0, =USART0_BASE
        ldr     r1, [r0, #US_CSR]
        tst     r1, #US_TXRDY
        blne    isr_putch
// …
isr_getch:
        ldr     r2, =USART0_BASE
        ldr     r1, [r2, #US_RHR] @ Laden des Receiver Holding Reg.
        and     r0, r1, #255         @ Maskieren
        mov     pc, lr
```

putc und getc Funktionen

```
void putc(int channel, char c)  {
    StructUSART* base = ser_channels[channel];
    unsigned int status;
    do {
        status = base->US_CSR;
    } while ((status & US_TXRDY) == 0);
    base->US_THR =  (unsigned int)c;
}

unsigned char getc(int channel) {
    StructUSART* base = ser_channels[channel];
    unsigned char ch;
    while( (base->US_CSR & US_RXRDY) == 0 );
    return  (base->US_RHR & 0xff);
}
```

Serielle Schnittstelle (DMA Funktion)

Serielle Schnittstelle bietet DMA (Direct Memory Access) Funktion

Register:

US_TPR Transmit Pointer Register (Adresse der Zeichenkette)
US_TCR Transmit Counter Register (Länge der Zeichenkette)

US_RPR Receive Pointer Register (Adresse der Zeichenkette)
US_RCR Reveice Counter Register (Länge der Zeichenkette)

Zeichenkettenübertragung:

Counter Register wird dekrementiert, Pointer Register inkrementiert
Trigger durch RXRDY bzw. TXRDY bit
Wenn Zähler 0 ist, wird das Statusbit in US_CSR gesetzt:

Empfang: ENDRX
Senden: ENDTX

Serielle Schnittstelle (DMA Funktion)

Beispielcode:

```
void StringAusgabe(char *StringAdresse) {
int StringLaenge = 0;
unsigned int StringAnfangsAdresse = (unsigned int) StringAdresse;
// Stringlänge bis zur 0-Terminierung ermitteln
while( *StringAdresse++ ) StringLaenge++;
// Transmit Transfer steht zur Verfügung?
while(!(USART0->US_CSR & US_ENDTX));
// Adresse vom String ins TransmitPointerRegister
USART0->US_TPR = StringAnfangsAdresse;
// Länge des String ins TransmitCounterRegister
USART0->US_TCR = StringLaenge;
}
```

Binäre Leitungscodes

NRZ (non return to Zero):

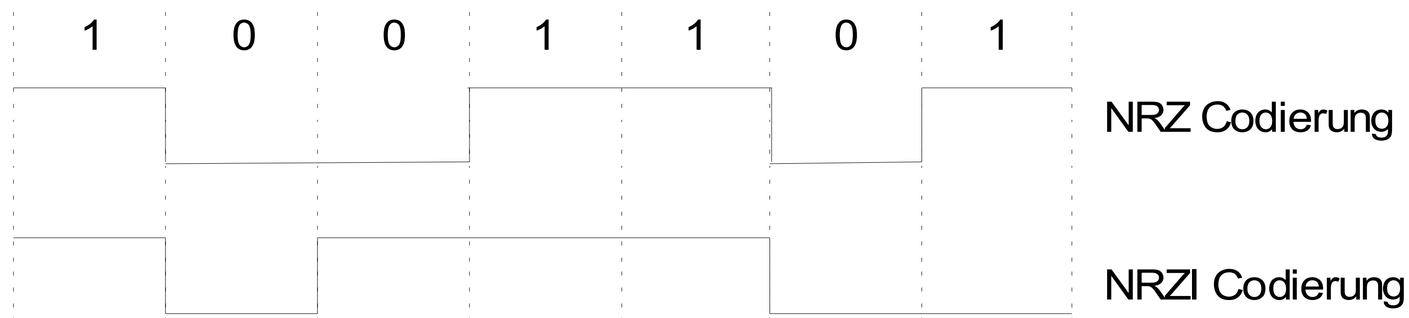
- 0: Lowpegel 1: Highpegel
- Standard bei asynchroner Übertragung

NRZI (NRZ mit Interchange):

- Pegelwechsel bei 0, kein Pegelwechsel bei 1

Manchester Code

- selbsttaktend: Takt- und Dateninformation werden kombiniert



NRZI: Pegelwechsel bei 0, kein Wechsel bei 1

Sicherung der Datenübertragung

Codesicherungsverfahren:

VRC (Vertical Redundancy Checking, Querprität):

Paritätsbit (Querparität): 1 Bit Fehler werden erkannt (aber nicht korrigiert)

LRC (Longitudal Redundancy Checking, Längsparität):

Für die Spalte eines Blocks wird ein entsprechendes Paritätsbit gebildet

Sicherung der Datenübertragung

Beispiel: gerade Quer- und Längsparität

									VRC
	1	0	1	1	0	1	0	1	1
	1	0	1	0	1	1	0	1	1
	0	0	0	1	1	0	1	0	0
	1	1	1	0	1	0	1	0	1
LRC	1	1	1	1	1	0	0	0	1

Kreuzsicherung:

Kombination von Quer- und Längsparität

Einfachfehler werden korrigiert

Doppelfehler und Dreifachfehler werden erkannt

Sicherung der Datenübertragung

Beispiel: Doppelfehler erkennen

									VRC
	1	0	1	1	0	1	0	1	1
	1	0	1	0	1	1	0	1	1
	0	0	0	1	0	0	1	0	0
	1	1	1	0	1	0	1	0	1
LRC	1	1	1	1	1	0	0	0	1

Beispiel: Dreifachfehler erkennen

									VRC
	1	0	1	1	0	1	0	1	1
	1	0	1	0	1	0	0	1	1
	0	0	0	1	0	0	1	0	0
	1	1	1	0	1	0	1	0	1
LRC	1	1	1	1	1	0	0	0	1

Sicherung der Datenübertragung (CRC)

Ziel: Steigerung der Fehlererkennungsrate

Zyklische Blocksicherung (CRC – Cyclic Redundancy Check)

Systematische Berechnung eines CRC Prüfwortes aus der Nutzinformation

Basiert auf modulo 2 Arithmetik

Die N Nutzbits werden als Koeffizienten eines Polynoms B(x) interpretiert

$$B(x) = b_{N-1}x^{N-1} + b_{N-2}x^{N-2} + \ldots + b_1 x + b_0$$

Generatorpolynom:

das erzeugte Codewort (Nutzinformation + Prüfbits) ist ein Vielfaches des Generatorpolynoms (Fehlerkriterium)

CRC Algorithmus

Eingabe: Nachricht der Länge k (entsprechend einem Polynom p vom Grad $< k$), Generatorpolynom g vom Grad m

Ausgabe: Codewort der Länge $n = k + m$ (entsprechend einem Polynom h vom Grad $< n$)

Methode:

1. Multipliziere p mit x^m (m Nullen an die Nachricht anhängen):

$$f = p \cdot x^m.$$

2. Teile das Ergebnis f durch das Generatorpolynom g und bilde den Rest r:

$$f = q \cdot g + r \quad \text{mit grad}(r) < \text{grad}(g) = m.$$

3. „Addiere“ r zu f:

$$h = f + r = q \cdot g + r + r = q \cdot g$$

Das Ergebnis h entspricht dem gesuchten Codewort: h ist durch g teilbar.

Fehlererkennung: empfangene Wort wird durch Generatorpolynom geteilt. Ist der Rest ungleich Null, so ist ein Fehler aufgetreten.

CRC Algorithmus

Beispiel:

Gegeben: Nachricht 100101110011101, Generatorpolynom 100111

<u>1. Schritt:</u> Multipliziere das Polynom *p*, das der Nachricht entspricht, mit x^5

<u>2. Schritt:</u> Teile das Ergebnis *f* durch das Generatorpolynom *g*

```
1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 0 0 0 0
1 0 0 1 1 1
0 0 0 0 1 0 1 1 0 0
        1 0 0 1 1 1
        0 0 1 0 1 1 1 1
            1 0 0 1 1 1
            0 0 1 0 0 0 1 0
                1 0 0 1 1 1
                0 0 0 1 0 1 1 0 0
                      1 0 0 1 1 1
                      0 0 1 0 1 1 0 0
                          1 0 0 1 1 1
                          0 0 1 0 1 1 0
```

$g(x) = x^5 + x^2 + x + 1$

Wort 1 0 1 1 0 entspricht dem Rest *r*

CRC Algorithmus

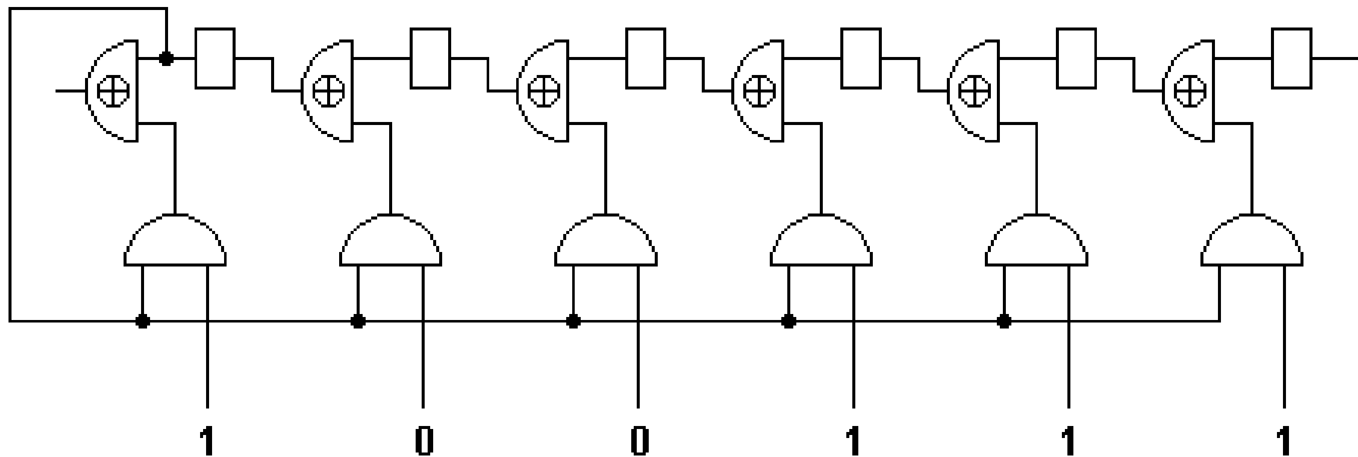
Beispiel:

<u>3. Schritt:</u> Addiere *r* zu *f*. Das Ergebnis entspricht dem gesuchten Codewort

1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 1 0 1 1 0

CRC Implementierung in Hardware

Polynomdivision lässt sich einfach in Hardware realisieren



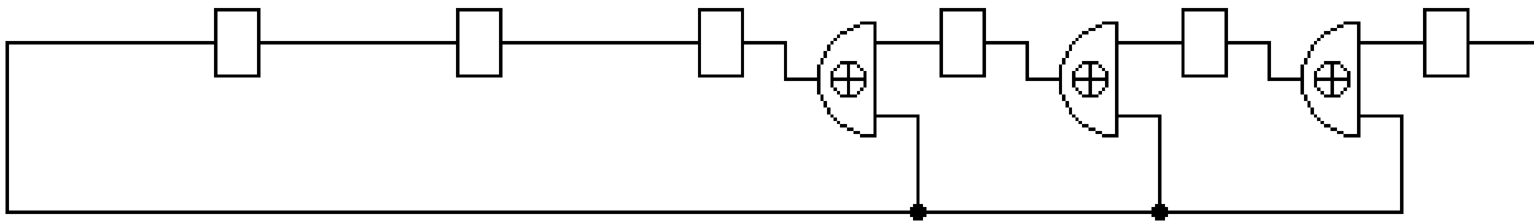
Vorgehensweise im Schieberegister:

- von rechts wird das zu dividierende Polynom h hineingeschoben
- bei einer 1 vorne wird das Generatorpolynom mit dieser 1 multipliziert
- Generatorpolynom wird subtrahiert (EOR Gatter)
- danach eine Position nach links verschoben und nächstfolgende Stelle von h wird nachgeschoben
- bei einer 0 vorne wird nur der Schieberegisterinhalt geschoben

CRC Implementierung in Hardware

Bei festem Generatorpolynom lässt sich Schaltung vereinfachen

Generatorpolynom: 1 0 0 1 1 1



Vereinfachte Schaltung: Linear Feed-Back Shift Register (LFSR)

Typische Generatorpolynome:

CRC-16 (Magnetband): $x^{16} + x^{15} + x^{2} + 1$

CRC-CCITT (Disketten): $x^{16} + x^{12} + x^{5} + 1$

CRC-Ethernet: $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$

Achtung: Operator + ist gleich EOR

CRC Erkennen von Fehlern

Fehler: im gesendeten Wort werden ein oder mehrere Bits invertiert.

Wird das Wort als Polynom aufgefasst, entspricht das Invertieren eines Bits der Addition eines Fehlerpolynoms e, das eine 1 an dieser Position hat.

Beispiel:

Durch Addition des Fehlerpolynoms 1 0 1 0 0 werden zwei Bits verfälscht.

```
  1 0 1 1 0 0 0 1 1 0 0 0 1
+ 0 0 0 0 0 0 0 0 1 0 1 0 0
  -------------------------
  1 0 1 1 0 0 0 1 0 0 1 0 1
```

Erkennung: Empfangene Wort wird durch Generatorpolynom geteilt. Ist der Rest ungleich Null, so ist ein Fehler aufgetreten.

Mit dem CRC-16-Verfahren können

- bis zu 16 aufeinanderfolgende Bitfolge sicher erkannt werden
- längere Bitfehlerfolgen mit der Sicherheit von 99,997% erkannt werden

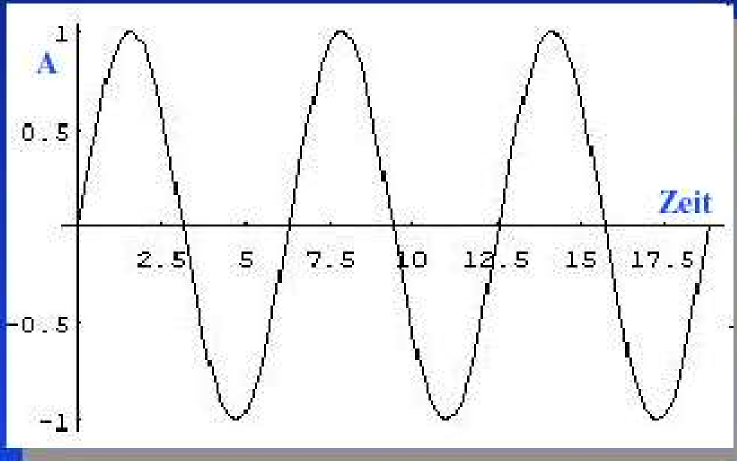
Gliederung

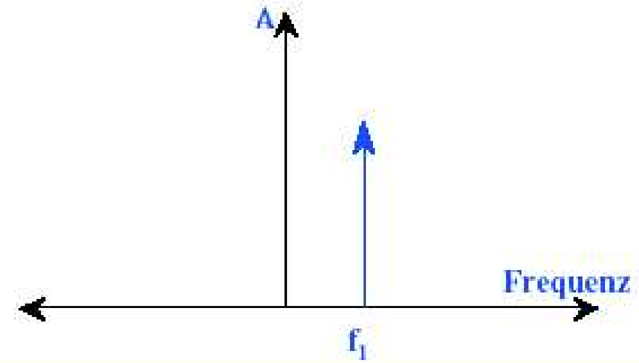
FHD

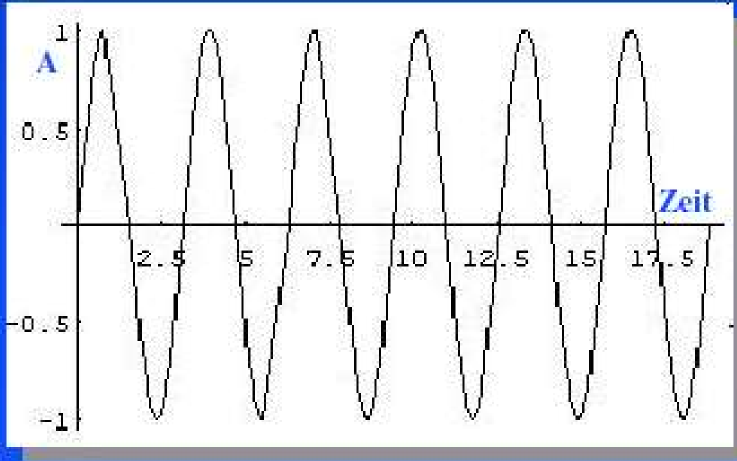
- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)
- Signalaufbereitung
  - **Analog-Digital Wandlung**
  - Digital-Analog-Wandlung
- Weiterführende Themen
  - USB

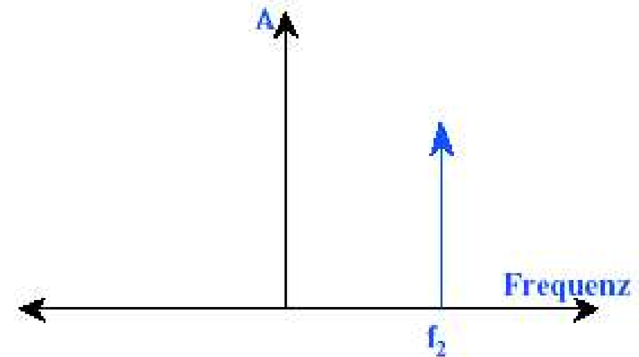
Zeit- und Frequenzbereich

Zeitbereich | Frequenzbereich

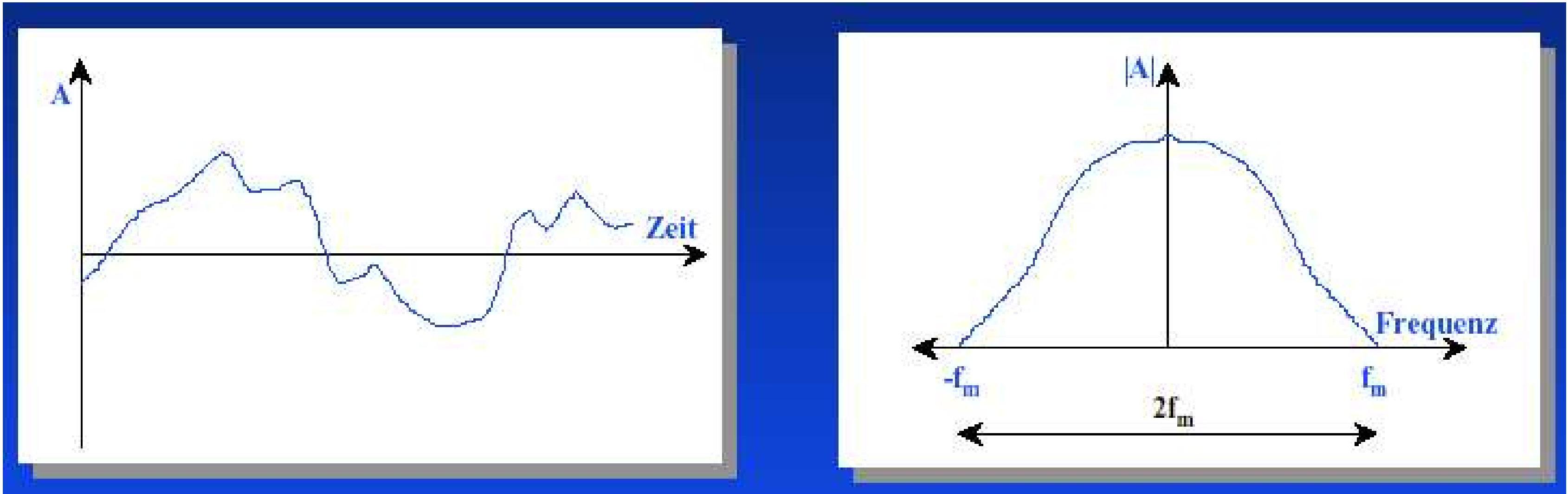








Reale Signale

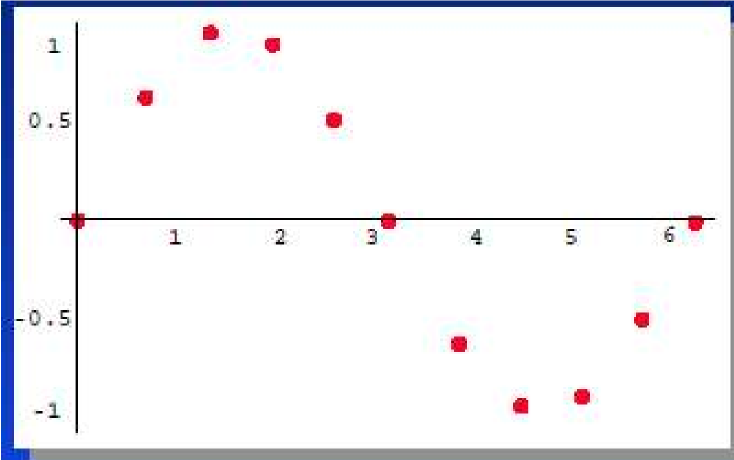


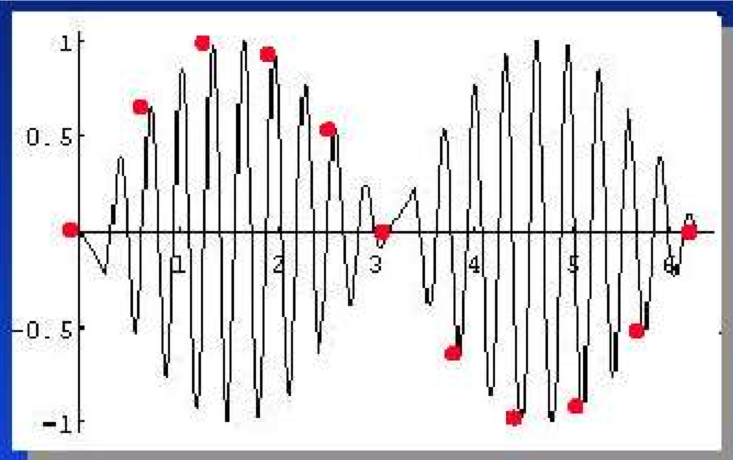
- reale Signale sind Mischungen vieler Frequenzen
- sie besitzen eine Bandbreite 2*fm
- Frequenzspektrum bezeichnet alle enthaltenen Frequenzen

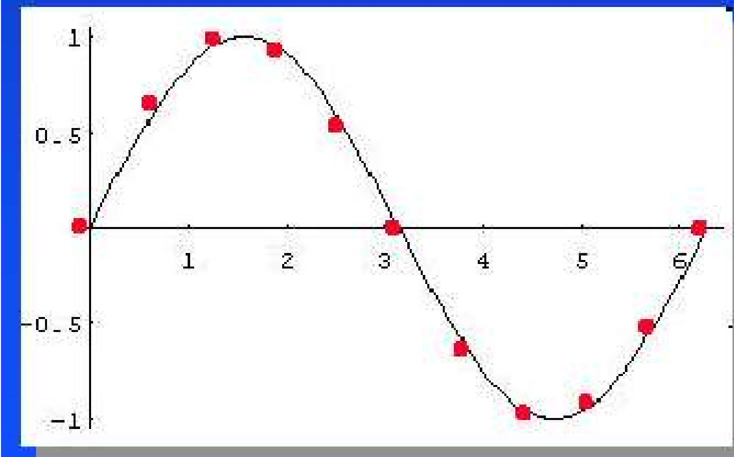
Fehlende Information

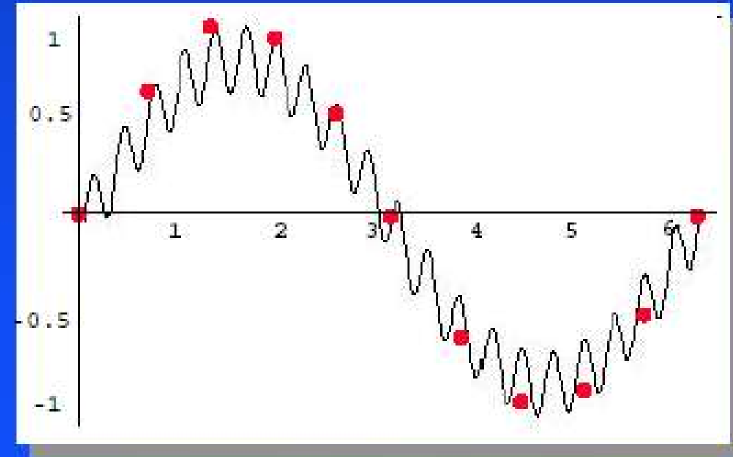
Rekonstruktion des Signals nach Abtastung ?

Wahl der Abtastfrequenz ist wichtig





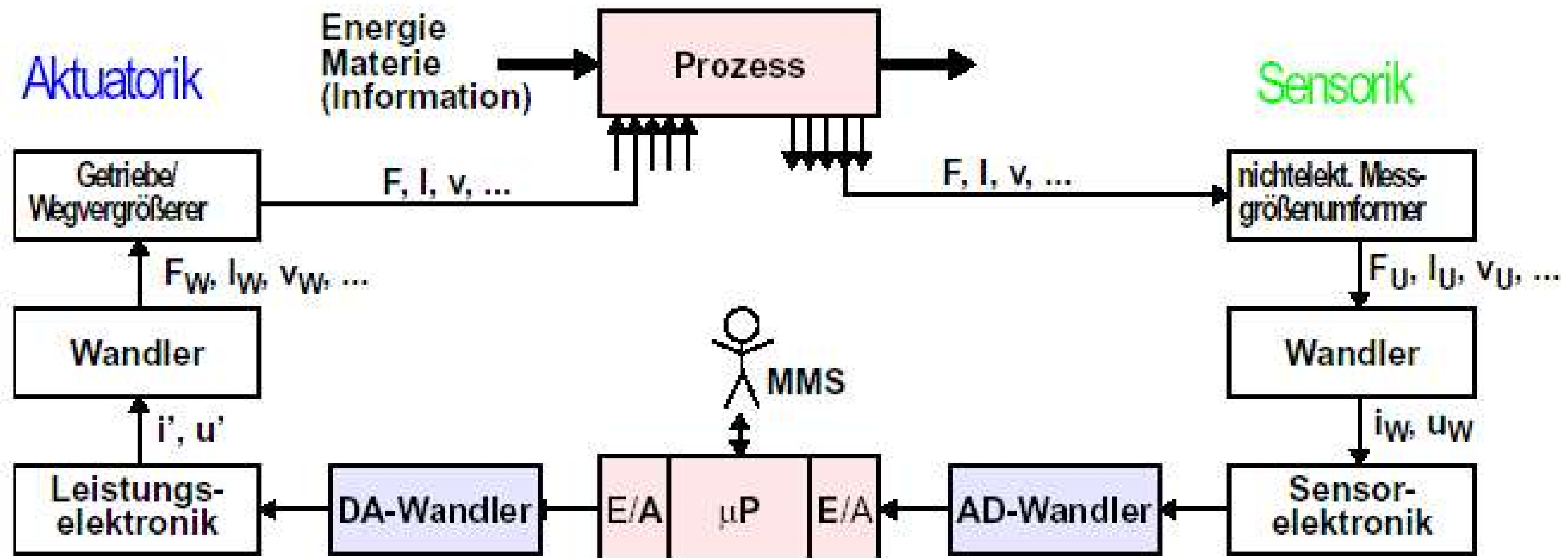




Signalaufbereitung

Mikroprozessor oft eingesetzt in einer

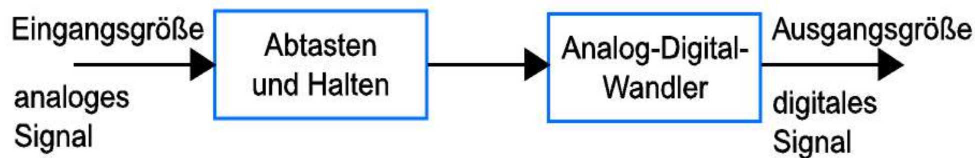
- Regelung
- Prozeßsteuerung

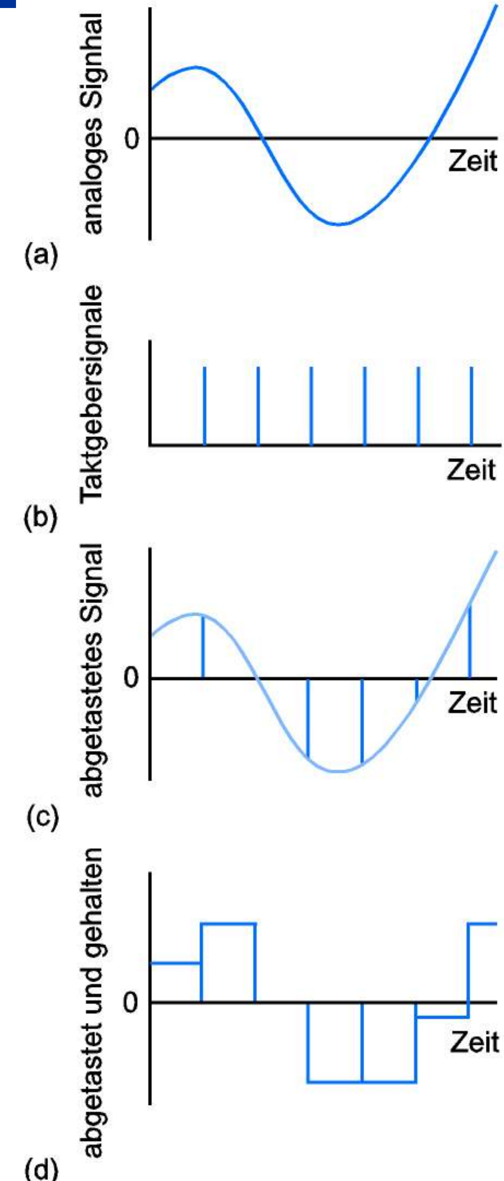


Analog-Digital Wandlung (ADC)

Analog-Digital-Wandlung

- umfasst die Umwandlung von analogen Signalen in binäre Wörter
- Takt sorgt für Abtastung
- Abtast- und Halteeinheit hält den abgefragten Wert bis zum nächsten Taktimpuls

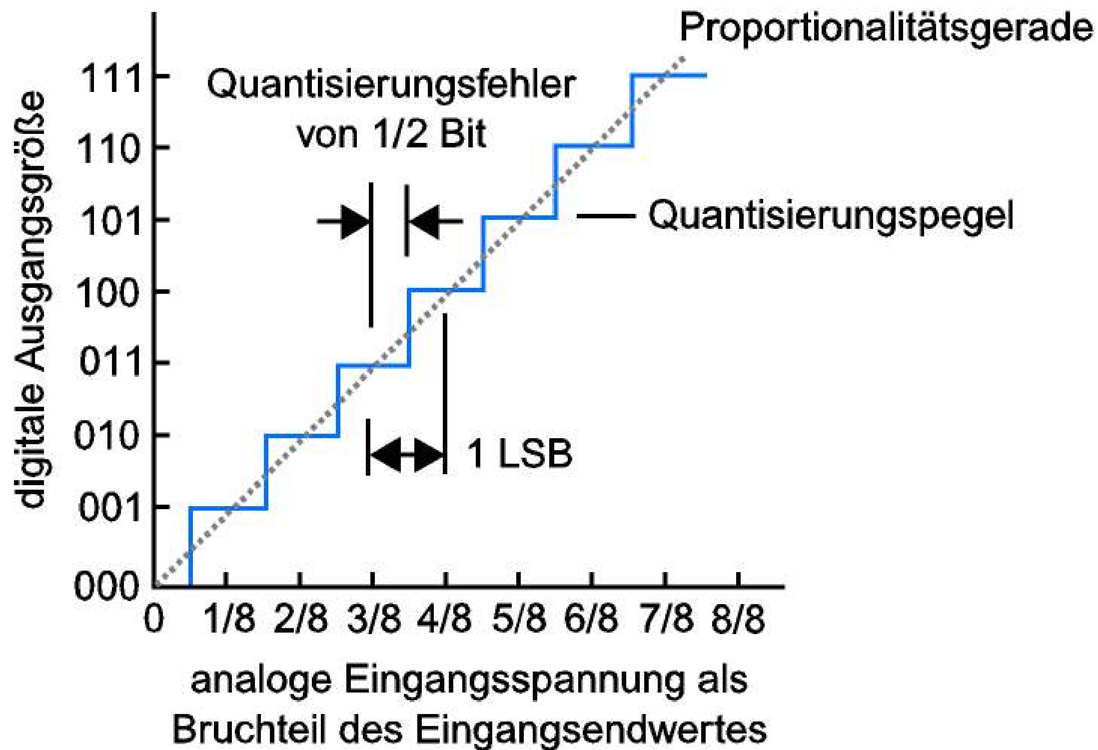




Analog-Digital Wandlung

Beziehung zwischen der abgetasteten (und gehaltenen) Eingangsgrösse und der Ausgangsgrösse

Bsp.: 3 Bits Auflösung



Analog-Digital Wandlung

Allgemein: Wortlänge n Bits

Minimale Änderung der Eingangsgrösse, die erfasst wird, beträgt: $U/2^n$ (U Größenbereich des Analogsignals)

Beispiel:

Wortlänge n: 10 Bits

Größenbereich des Analogsignals U: 10 V

-> Auflösung: $10\ V / 2^{10} = 10\ V/1024 = 9.8\ mV$

Hörsaalübung:

Thermoelement mit Ausgangsgrösse 0.5 mV/Grad Celsius

Welche Wortlänge n ist erforderlich bei einem Temperaturbereich von 0 bis 200 Grad

Geforderte Auflösung: 0.5 Grad

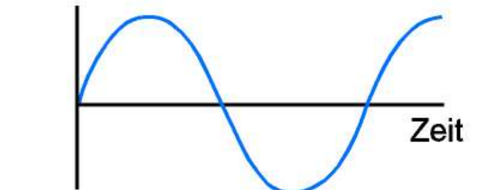
Abtasttheorem

Abtasttheorem von Shannon (Nyquist Kriterium):

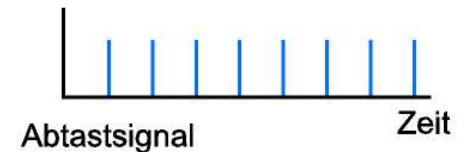
Rekonstruktion des Signals in der ursprünglichen
Form nur möglich, wenn Abtastfrequenz mindestens doppelt so gross ist, wie die grösste Frequenz im abgetasteten Signal

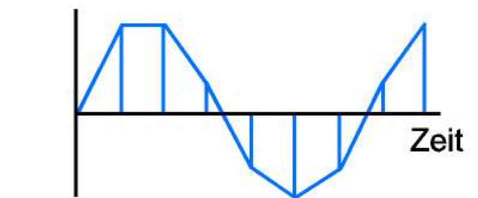
Falls Abtastfrequenz niedriger, kann die Rekonstruktion ein anderes analoges Signal darstellen

-> Aliasing (dt. Pseudonymisierung)

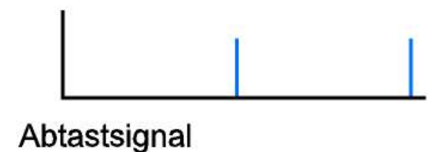


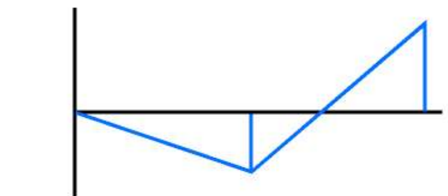
(a)





(b)



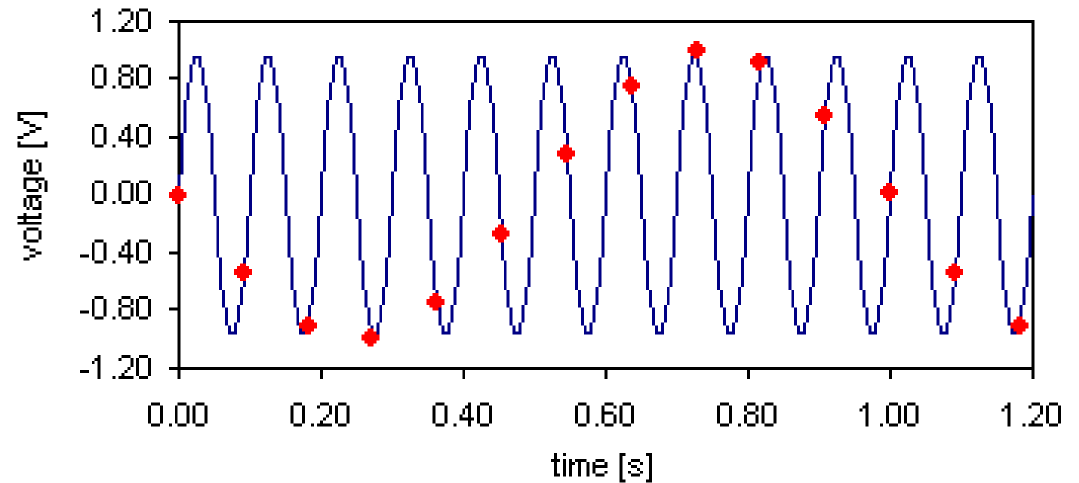


(c)

Problem Aliasing

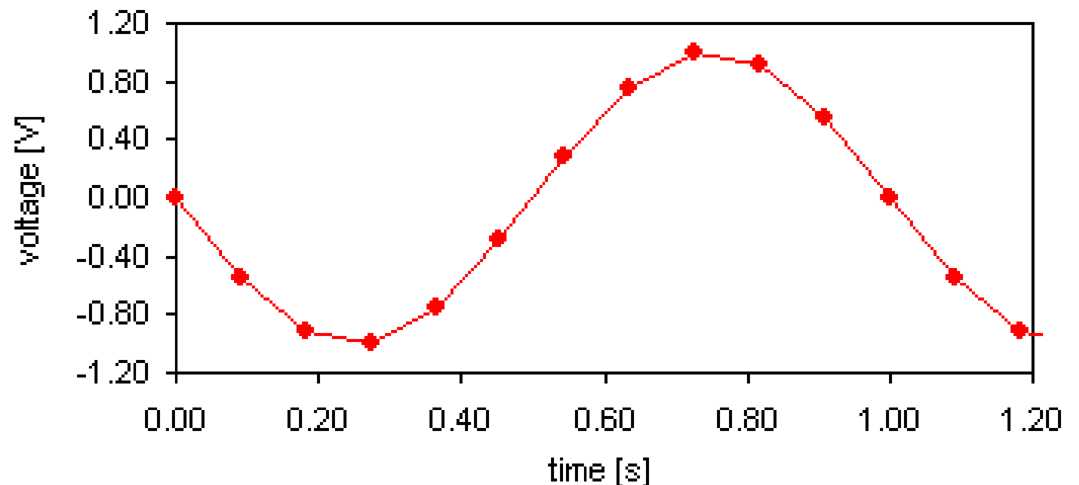
10 Hz Signal

Abtastung: 11 Hz



Ergebnis:

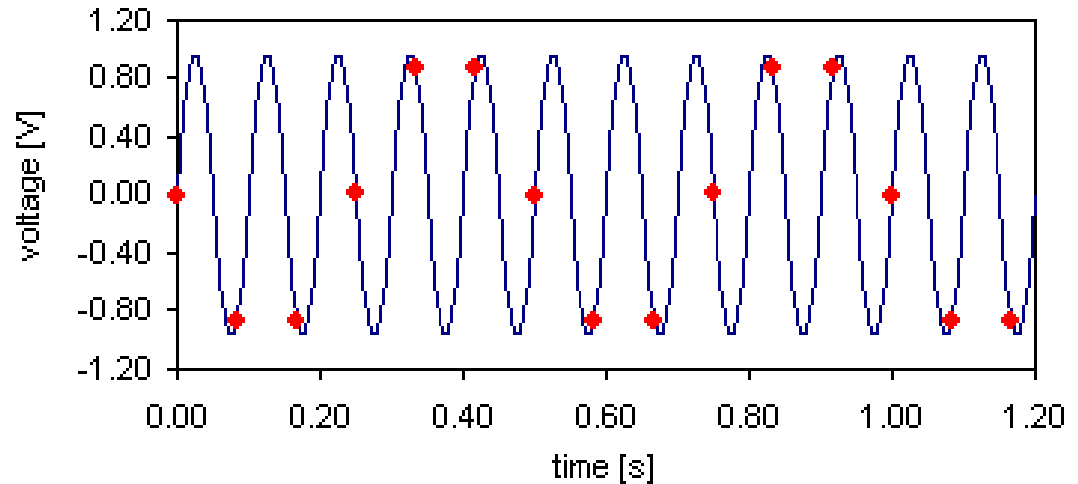
1 Hz (?!)



Problem Aliasing

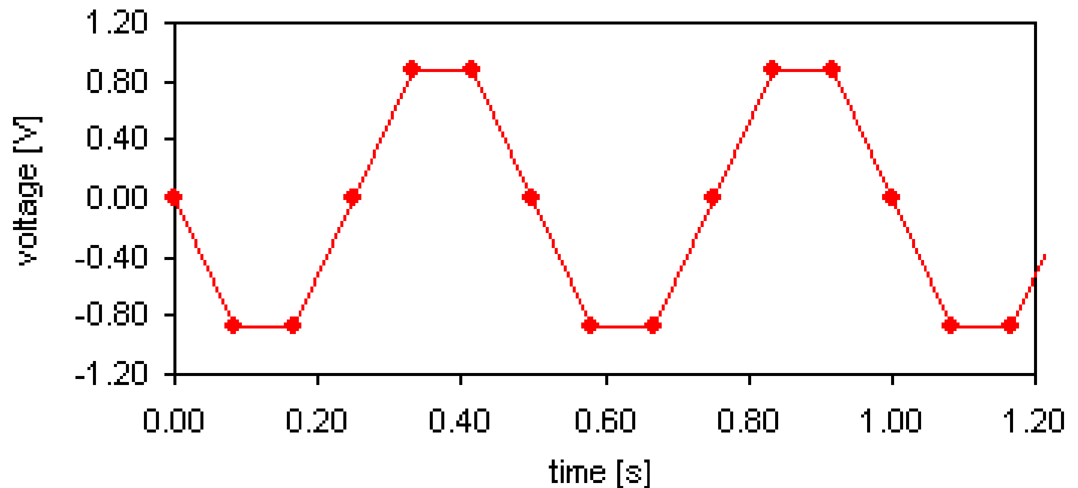
10 Hz Signal

Abtastung: 12 Hz

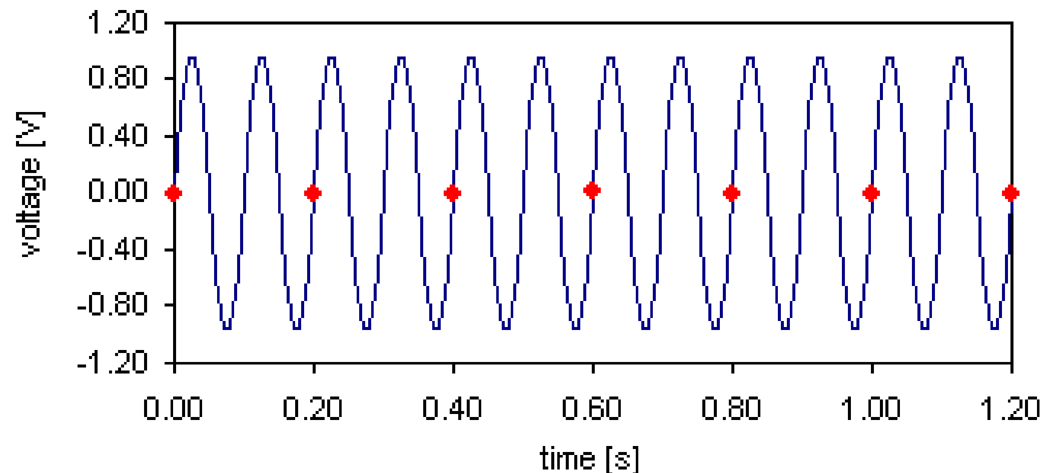


Ergebnis:

2 Hz (?!)



Problem Aliasing

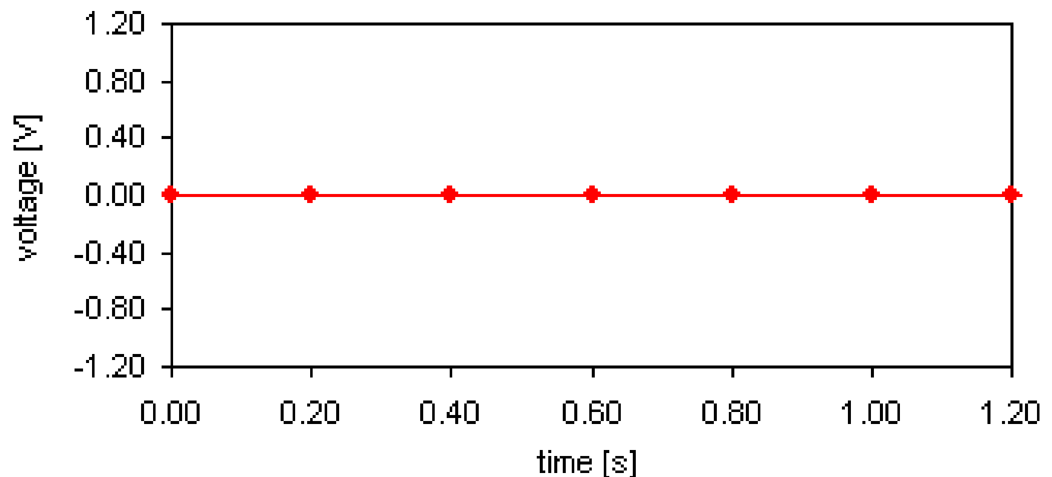


10 Hz Signal

Abtastung: 5 Hz

Achtung:

$f_{sample} = f_{signal} / N$



Ergebnis:

0 Hz (?!)

Analoge Signale

Datenvolumen Beispiel: 1 min Audiosignal

- Hifi-Qualität

D= 60 s · 44100 Abtastungen/Sek. · 16 bit/Sample = 5292 kByte

- Telefon(ISDN)-Qualität

D= 60 s · 8000 Abtastungen/Sek. · 8 bit/Sample = 480 kByte

Telefonie in Hifi-Qualität verursacht 11-fach höhere Übertragungskosten.

Ursache: Abtastrate und Wertdiskretisierung

Analog-Digital Wandlung

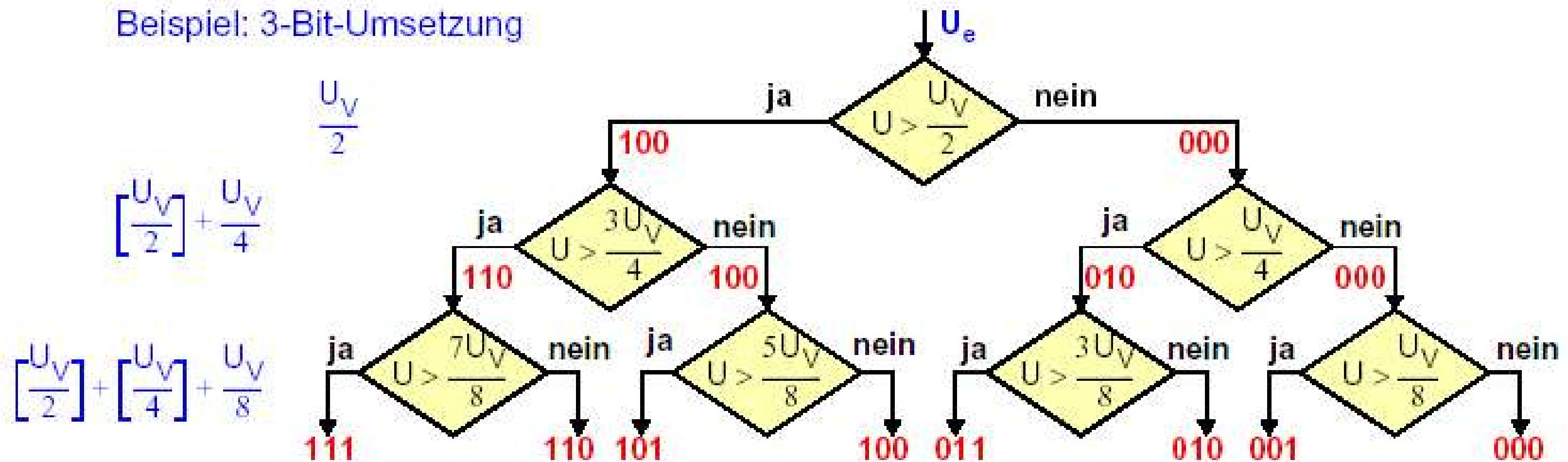
z.B. durch schrittweise Annäherung (Wägeverfahren)

- direktes Umsetzverfahren

  Vergleich der Eingangsgröße mit einer Referenzgröße (ohne Zwischenumsetzung)

- bitserieller Vergleich, beginnend mit MSB

  analog zum Abwiegen eines unbekannten Gewichts mittels Balkenwaage

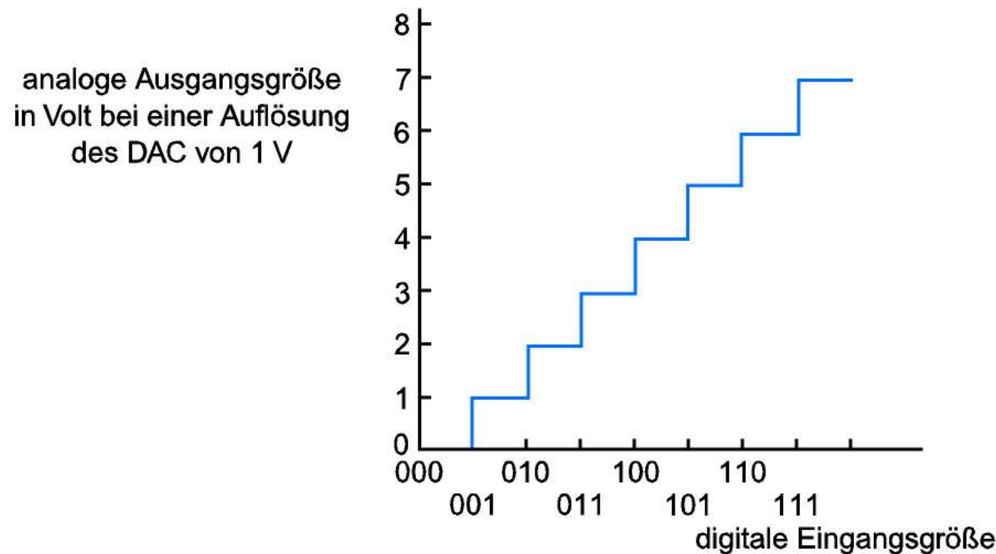


Gliederung

- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)
- Signalaufbereitung
  - Analog-Digital Wandlung
  - **Digital-Analog-Wandlung**
- Weiterführende Themen
  - USB

Digital-Analog Wandlung (DAC)

- Eingangsgrösse ist ein binäres Wort
- Ausgangsgrösse analoges Signal der gewichteten Summe der (Nicht-Null) Bits



Beispiel: Regelventil (vollständige Öffnung 6 V)

8 Bit Digital-Analog Wandlung

Frage: Änderung der Ausgangsgrösse zum Ventil bei einer Änderung von 1 Bit ?

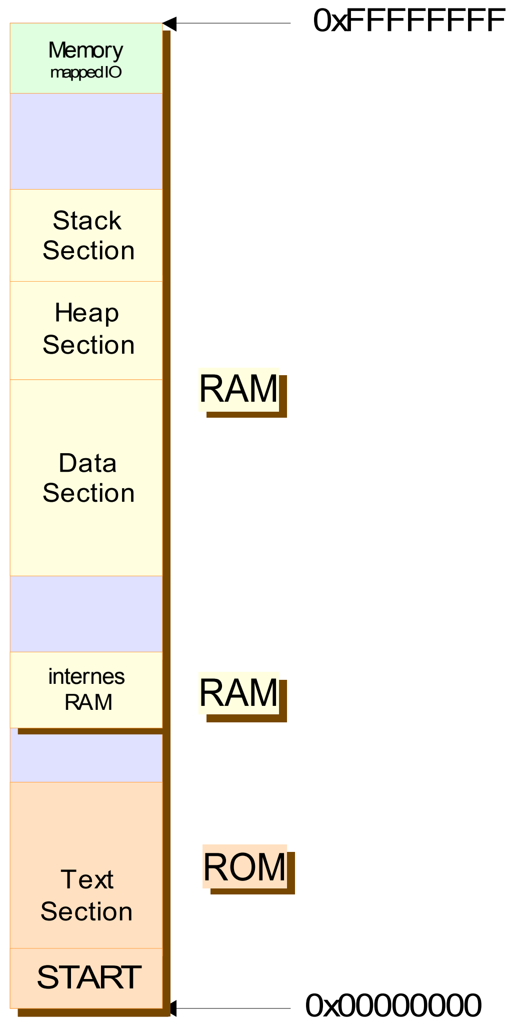
Antwort: $6.0\ V / 2^8 = 0.023\ V$

Systemstart

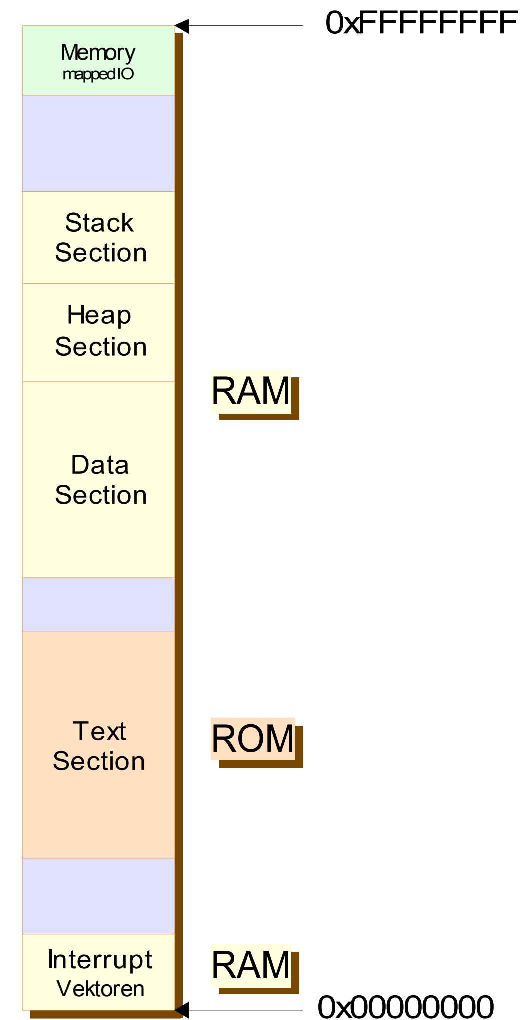
- Prozessor startet bei Power UP an der Adresse 0
- Problem:
  - bei Systemstart sollte an dieser Stelle ROM (Flash) Speicher sein
  - später sollte die Vektortabelle im RAM liegen um flexible Programmierung zu ermöglichen
- Lösung:
  - Memory Controller erlaubt ein Remapping der Speicherbereiche zur Laufzeit
  - ARM7: External Bus Interface (EBI)

Systemstart

Speicher bei Power UP



Speicher nach Remapping



Ablauf des Bootvorgangs

Initialisierung des Advanced Interrupt Controller
Kopieren der Interrupt Vektortabelle in internen RAM
Remapping der Speicherbereiche
Definition der Stackgrössen
Initialisieren der Stackpointer
Initialisieren der BSS Section (uninitialisierte Variablen)
Sprung in main Routine

siehe Datei boot.s

0x1c	FIQ
0x18	IRQ
0x14	(Reserviert)
0x10	Data Abort
0x0c	Prefetch Abort
0x08	Software Interrupt
0x04	Undefined Instruction
0x00	Reset

Gliederung

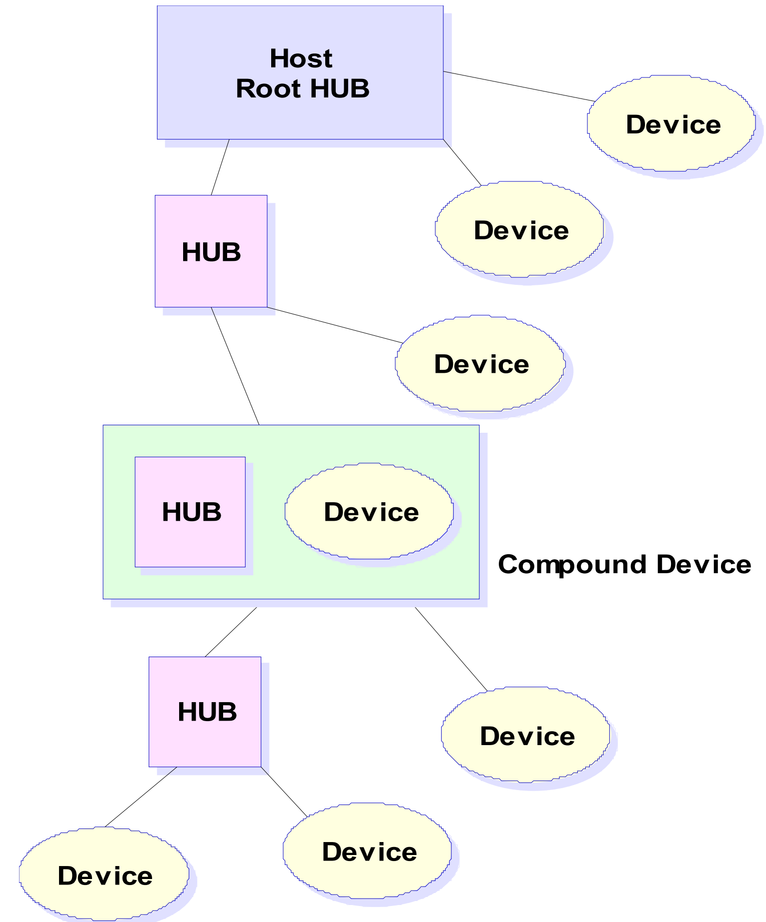
- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode
  - Capture Mode
- Serielle Schnittstelle
- Softwareinterrupt (SWI)
- Signalaufbereitung
  - Analog-Digital Wandlung
  - Digital-Analog-Wandlung
- Weiterführende Themen
  - **USB**

Eigenschaften des Universal Serial Bus

- Geschwindigkeiten
  - High Speed - 480 Mbit/s   USB Version 2.0
  - Full Speed - 12 Mbit/s   USB Version 1.1
  - Low Speed - 1.5 Mbit/s   USB Version 1.1
- USB ist Master basiert. Es kann nur einen Master (Host) pro Bus geben. Der Master ist verantwortlich für die Durchführung aller Transaktionen und für die Verteilung der Bandbreite.
- Bis zu 127 Devices können an den USB angeschlossen werden
- USB besitzt eine geschichtete Stern Architektur, ähnlich wie Ethernet 10BaseT. Die Erweiterung des Buses erfolgt über Hubs. Viele Peripherie Devices haben zusätzlich Hubs integriert.

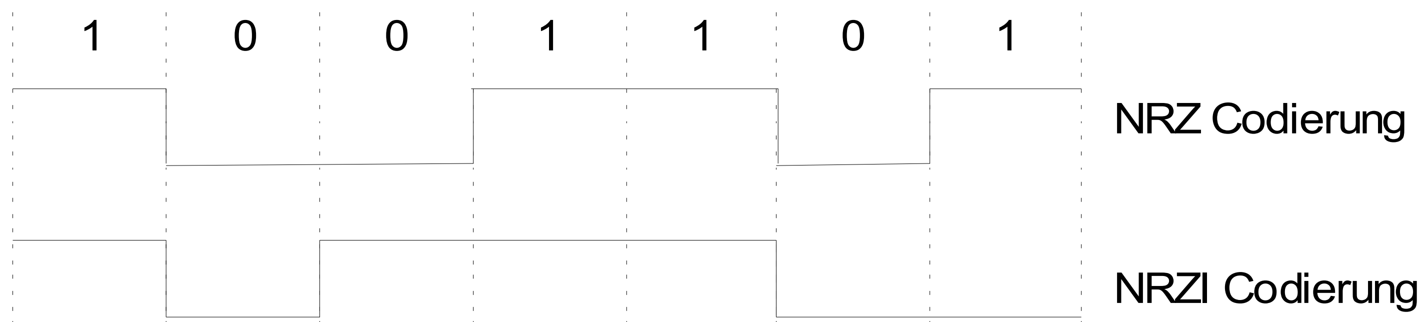
Eigenschaften des Universal Serial Bus

- Vorteile der Architektur mit HUB:
  - Hot Plug IN
  - Vollständige Überwachung der Energieversorgung der Devices und Abschaltung bei Fehlern
  - Es können gleichzeitig Devices unterschiedlicher Geschwindigkeit am Bus sein, da der HUB die höheren Geschwindigkeiten für langsame Devices herausfiltert



Eigenschaften des Universal Serial Bus

- USB unterstützt Plug and Play mit mit dynamsich ladbaren Treibern
- USB ist ein serieller Bus.
  - Er benutzt 4 geschirmte Kabel, von denen 2 zur Energieversorgung dienen (+5V und Grnd).
  - Die anderen zwei Leitungen sind ein „Twisted Pair Kabel“ und übertragen ein differentielles Daten Signal.
  - Die Codierung ist NRZI mit Bitstuffing und einem Sync Feld am Anfang des Telegramms zur Synchronisation der Clock‘s von Sender und Empfänger



NRZI: Pegelwechsel bei 0, kein Wechsel bei 1

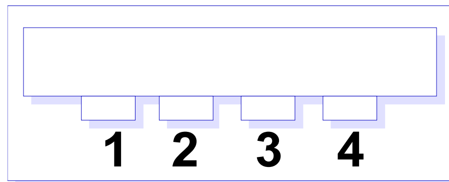
Hot Plug In

- Wird ein neues Device an den Bus angeschlossen erkennt der zuständige HUB diese Änderung und meldet sie dem Host
- Der Host fragt das Device nach seiner PID (Product ID) und seiner VID (Vendor ID) und lädt die entsprechenden Treiber
- Ebenso wird die Entfernung eines Devices vom Bus gemeldet und der Treiber wieder entfernt
- Die Kombination PID/VID wird vom USB Implementers Forum vergeben und ist kostenpflichtig

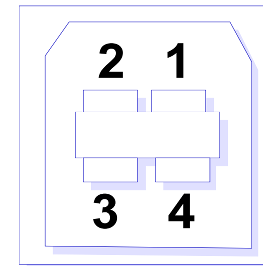
USB Stecker

- Der USB Bus kennt zwei verschiedene Steckerformen, einen für die Hostseite und für HUB‘s und einen für Devices

Typ A



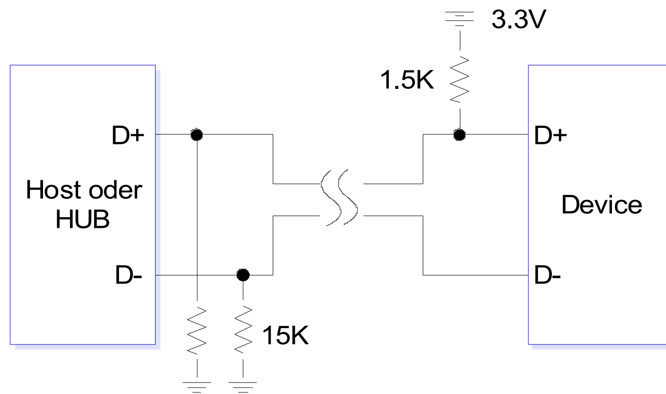
Typ B



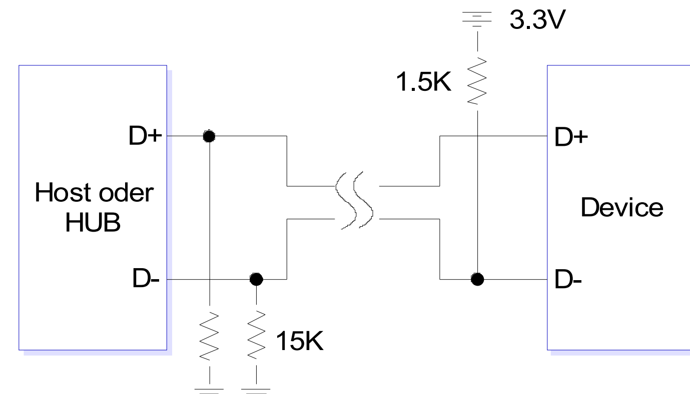
PIN Nummer	Kabel Farbe	Funktion
1	rot	V_{BUS}(5 Volt)
2	weiss	D-
3	grün	D+
4	schwarz	Ground

Bussignale

- Logisch 1: D+ > D- + 200mV
- Logisch 0: D+ < D- - 200mV
- SE0: D+ und D- für mehr als 10ms < 0.3 V



Full Speed Device mit Pull Up an D+



LOW Speed Device mit Pull Up an D-

Keep alive Signale

- High Speed Bus micro-frames alle 125 µs +- 62.5 ns
- Full Speed Bus frame alle 1.000 ms +- 500ns
- Low Speed Bus EOP Signal alle 1ms wenn kein Datenverkehr

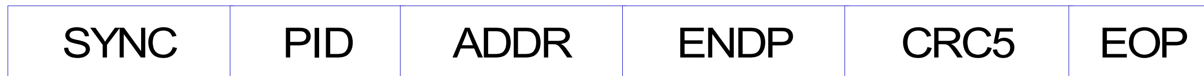
- Wenn auf dem Bus länger als 3 ms keine Aktivität ist, müssen alle Devices innerhalb der nächsten 7 ms in den Suspend Mode gehen, bei stark reduzierter Leistungsaufnahme

USB Protokoll

- Jede USB Transaktion besteht aus einem
  - Token Paket
  - Daten Paket (optional)
  - Status Paket
- Alle Aktionen werden vom Host angestossen
- Das erste Paket (Token) beschreibt welche Aktion folgt und ob im folgenden Datenpaketen Daten gelesen (vom Device zum Host) oder Daten geschrieben werden sollen (vom Host zum Device)

USB Paket Typen

- Token Paket
  - IN informiert das Device, dass der Host Daten lesen möchte
  - OUT informiert das Device, dass der USB Daten schreiben möchte
  - SETUP Start eines Steuer Transfers



USB Paket Typen

- Data Paket: es gibt zwei Typen von Datenpaketen, jedes ist in der Lage 0 bis 1023 Bytes zu tranportieren
  - DATA0
  - DATA1

USB Paket Typen

- Handshake Pakete - die drei Typen von Handshake Paketen bestehen nur aus dem PID
  - ACK: Bestätigung, dass Daten erfolgreich empfangen wurden
  - NAK: Signalisiert, dass das Device im Moment nicht erfolgreich Daten senden oder empangen konnte.
  - STALL: Das Device befindet sich in einem Zustand, in dem es Unterstützung vom Host benötigt

USB Paket Typen

- Start of Frame Pakete
  - Start of Frame Pakete bestehen aus einer 11 Bit Frame Nummer die jede ms (Full Speed USB) gesendet wird

Das Packet ID Feld

- Im Packet ID Feld (PID) ist codiert um welchen Typ von Paket es sich handelt

Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

Group PID Value Packet Identifier
Token 1 OUT Token
Data 1001 IN Token
Handshake 101 SOF Token
Special 1101 SETUP Token

Endpoints

- Endpoints stellen die Enpunkte von logischen Datenkommunikationskanälen auf dem USB dar
- Jedes Device muss den Endpoint 0 unterstützen, der für die Steuer und Statusinformationen des Devices zuständig ist.
- Ein Full Speed Device kann bis zu 16 Endpoints unterstützen, ein Low Speed Device neben den Default Endpoints nur 2 weitere.
- Endpoints stellen normalerweise Sende und Empfangsbuffer zur Verfügung und stellen so eine Schnittstelle zwischen logischer Struktur und Hardware dar

Endpoint und Transfer Typen

- **Control-Transfers**
  - für die Konfiguration von Devices. Paketgröße ist 8, 16, 32 oder
  - 64 Byte. Bidirektionale Transfervariante.
- **Interrupt-Transfers**
  - Garantierte Latenzzeit
  - für kleine Datenpakete mit geringer Häufigkeit (Beispiel: Maus).
  - Unidirektionale Transfervariante.
  - Im Fehlerfall werden die Daten noch einmal gesendet.
  - Paketgröße ist maximal 8 Bytes für Low-Speed und 64 Byte für Full-Speed Devices.

Endpoint und Transfer Typen

FHD

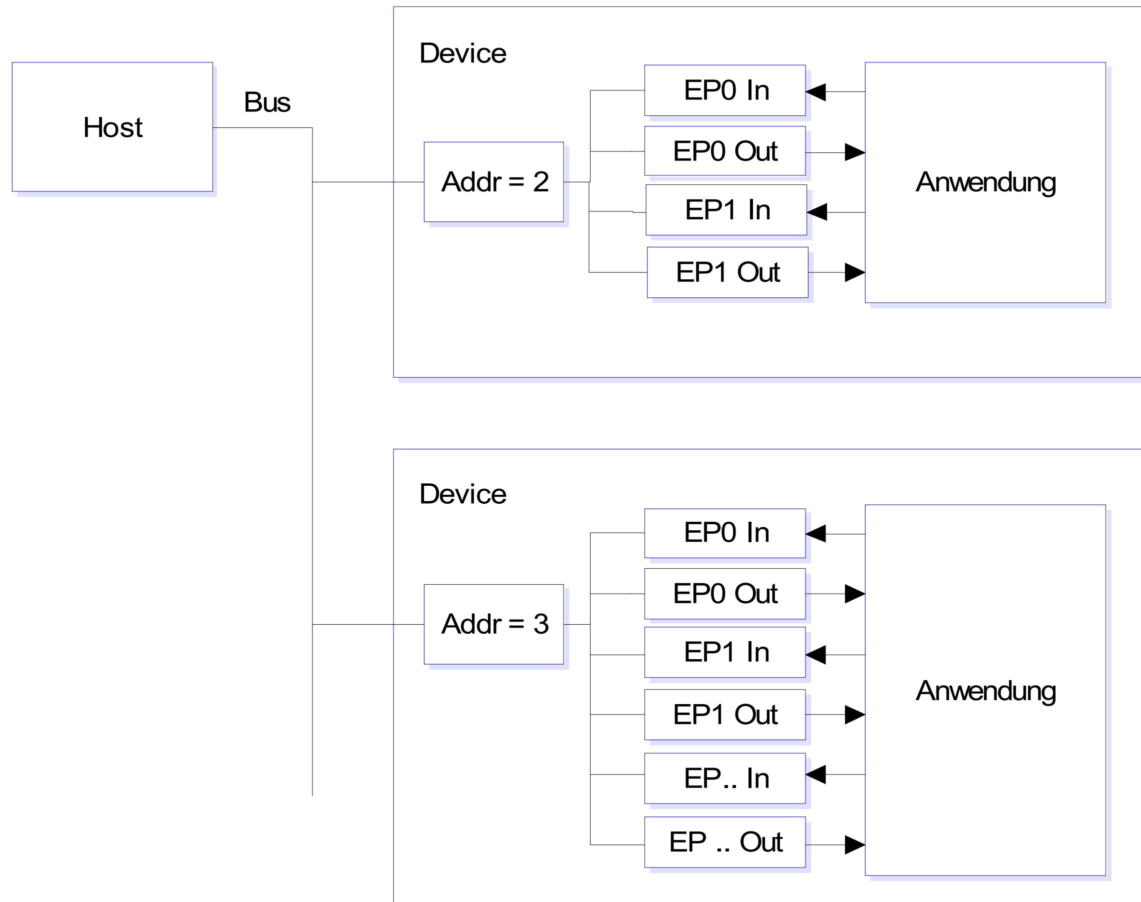
Bulk-Transfers

- für große Datenmengen und nicht-periodische Übertragung.
- Dieser Transfer wird für Anwendungen verwendet die keine bestimmte Bandbreite benötigen und deren Daten auch verzögert gesendet werden können (Beispiel:Drucker, Scanner).
- Paketgröße ist 8,16,32 oder 64 Byte.
- Nur im Full und High Speed Mode

Isochrone-Transfers

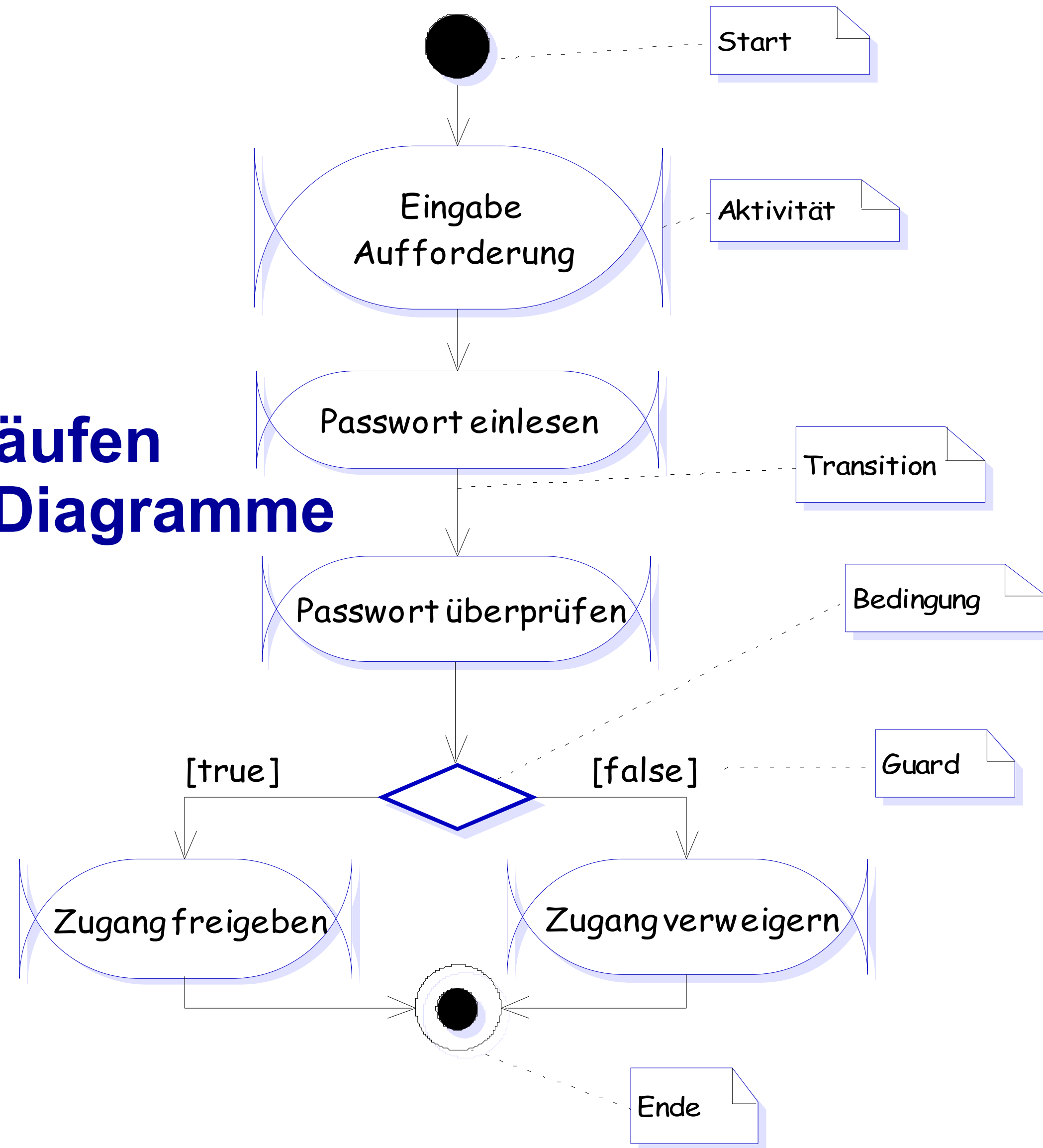
- für periodische und kontinuierliche Übertragung von zeitkritischen Daten wie Audio und Video.
- Uni-direktionale Transfervariante mit geringer Verzögerung.
- Paketgröße ist maximal 1023 Bytes.
- Nur im Full und High Speed Mode

USB Adressierung

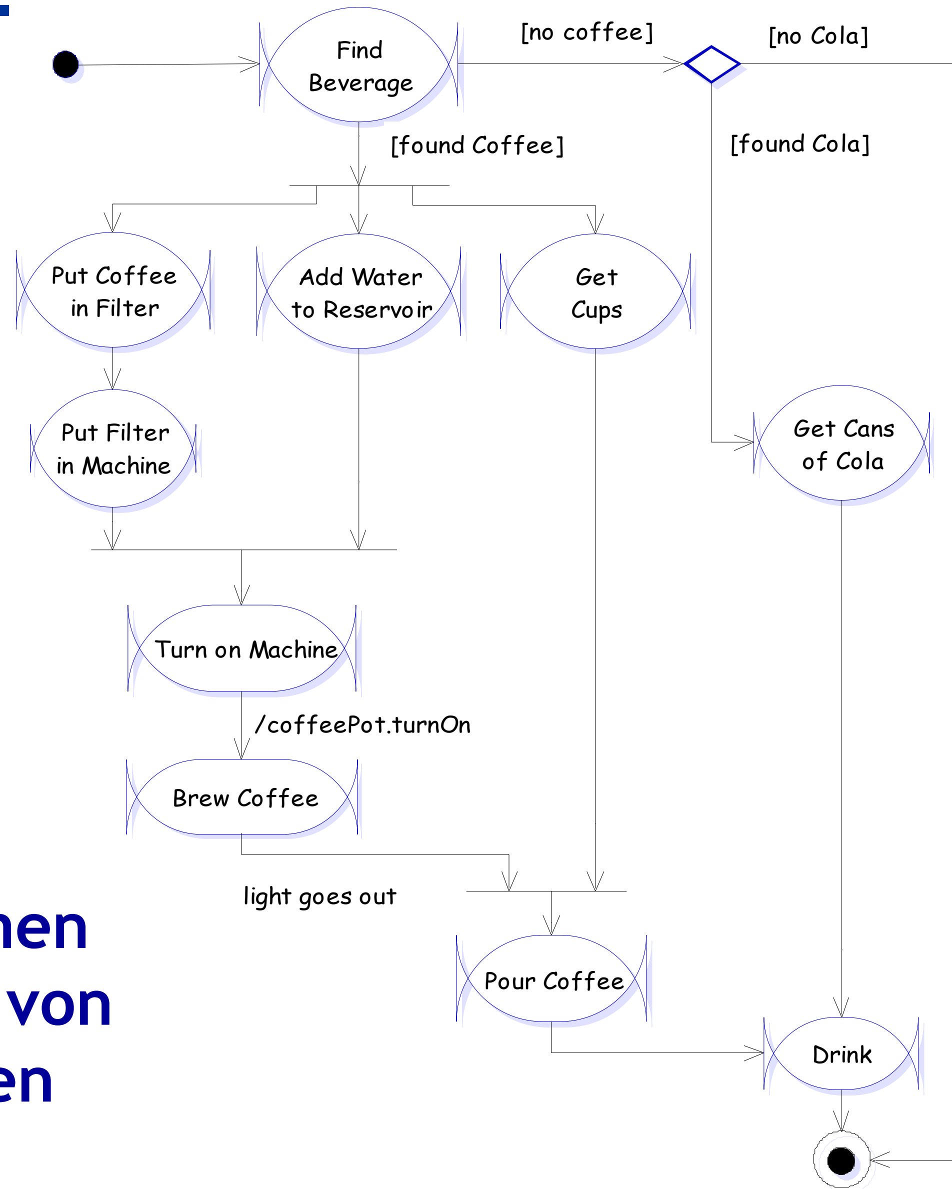


UML Activity Diagramme

Darstellung von Abläufen durch UML Activity Diagramme



UML Activity Diagramme



Die Nutzung von Activity Diagrammen zur Beschreibung von parallelen Abläufen