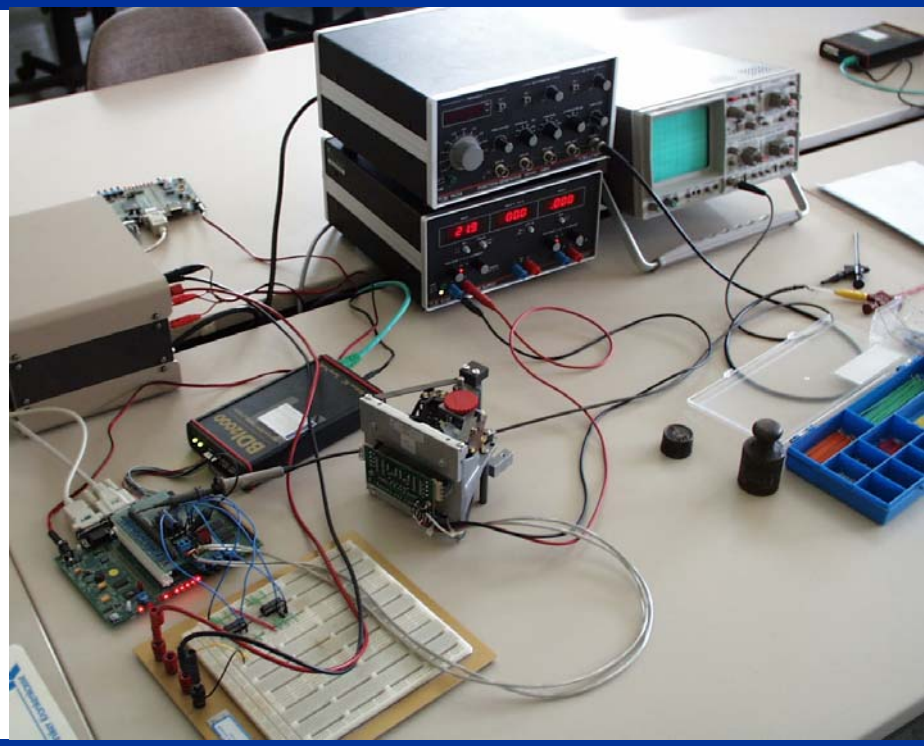
Mikroprozessorsysteme

Prof. Dr.Ing. Jens-Peter Akelbein



Prof. Dr.-Ing. Jens-Peter Akelbein

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Professur	Technische Informatik, Softwaretechnik und Grundlagen der Informatik
Interessen	Embedded Systeme, Internet of Things (IoT), Speichernetze, Projektmanagement PMP®
Forschung	Projekte BASE MoVE, sDDS4SmartHome (Smart Home, Ambient Assisted Living)
Aufgaben	KESS Koordination , Leitung EST Lab, Alumnibeauftrager, Direktorium aIDa
Telefon	+49 (6151) 16-38481
E-Mail	jens-peter.akelbein[at]h-da.de
Büro	Gebäude D 19, Raum 3.07
Sprechzeit	per Email Termin vereinbaren

Mikroprozessorsysteme

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Umfang:

3 SWS Vorlesung, Termine nach OBS
X: Doppelblöcke (10:15 – 13:30) mit kürzeren Pausen
1 SWS Praktikum, Termine nach OBS
Begleitende Tutorien

Voraussetzung:

Modulprüfung PG1 bestanden, PG2 angetreten (SPO 2014)

Materialien:

Kurs unter Informatik -> Informatik (B. Sc.) ->
WS 2018/2019 -> MPS Akelbein
Schlüssel AK2018MPS (alle Buchstaben groß)

Leistungsnachweis:

schriftliche Prüfung

Prüfungsvorleistung: Einzelabnahme der Praktika durch Fachgespräch

Prüfung:

schriftlich am Ende des Semesters
Prüfungstermin 15.2.2018

Vorlesungsinhalt

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Gliederung:

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

Originalskript: Mikroprozessorsysteme Th. Horsch (in Kooperation mit G. Raffius) V1.3
Quellen: Prof. Dr. U. Brinkschulte, Uni Karlsruhe
Prof. Dr. Th. Ungerer, Uni Augsburg

Literatur:

[1] Steve Furber, ARM Rechnerarchitekturen für System- on-Chip Design (h_da Bibliothek)
[2] Taschenbuch Mikroprozessortechnik (Beierlein, Hagenbruch)
[3] AT91M63200(Complete).pdf (im Netz)
[4] AT91EB63.pdf (im Netz)

Lernziele (siehe Modulbeschreibung)

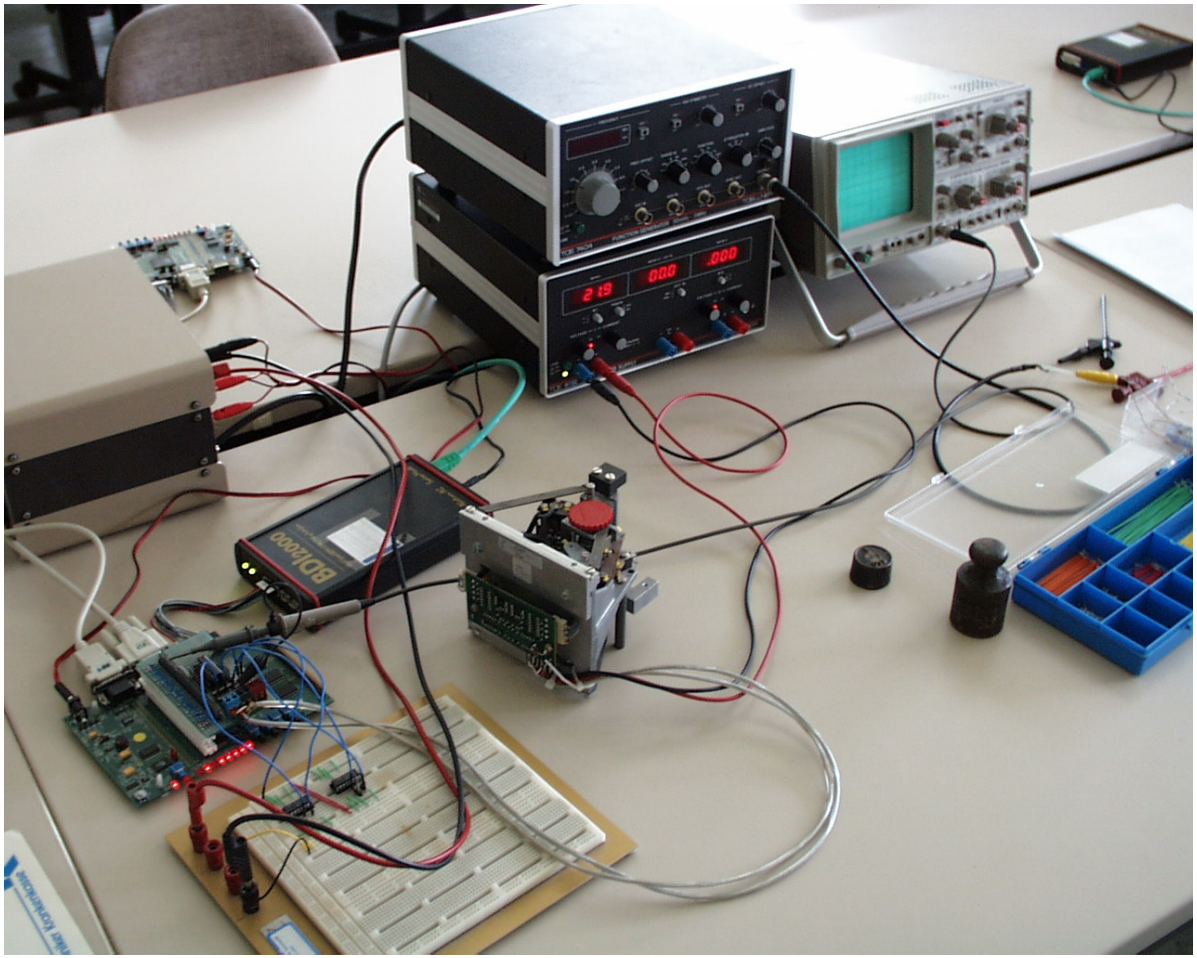
- Sie verstehen die HW- und SW-Konzepte für die Wechselwirkung eines Rechners mit seiner Umgebung mittels Peripheriekomponenten.
- Sie haben praktische Kenntnisse von Prozessoren und Peripherie in modernen Mikrocontrollern.
- Sie können mit einfachen eingebetteten Systemen umgehen, kennen deren Aufbau und können Software hierfür in einer Entwicklungsumgebung entwickeln.
- Sie haben tieferes Verständnis für systemnahe Programmierung und wissen, wie prozedurale Hochsprachenkonstrukte am Beispiel C in maschinennahe Implementierungen abgebildet werden.
- Sie verstehen, wie hardwarenahe Eigenschaften von Rechnern durch einen HAL (Hardware Abstraction Layer) auf höheren Abstraktionsebenen verborgen werden.

Mit Ihrem erworbenen Verständnis legen Sie die Grundlagen weiterführende Module in höheren Semestern (Betriebssysteme, Module im Bereich technischer Systeme, Sprachverarbeitung, Software für Embedded Systeme, Hardwarenahe Projekte, etc.).

Vielleicht entwickeln Sie dabei auch Begeisterung an der Technischen Informatik und deren Gestaltungskraft.

Praktikumsinhalt

Ihr Ziel: Implementierung einer Ausschankstation mit Waage und Pumpe
Waage (Schwingsaitenprinzip)



Organisation Praktikum

Praktikumsziel:

- Umgang mit der Peripherie des Prozessors
- Umgang mit Entwicklungswerkzeugen
- 5 Termine zur Entwicklung der Bestandteile einer **Münzwaage**
- 6. Termin Integration aller Bestandteile als lauffähige Software

Praktikumsdurchführung:

- Jeweils 16 Personen (8 Gruppen zu je 2 Personen)
- Vorbereitung notwendig: Nutzen Sie die angebotenen Tutorien von Beginn an (!), um sich mit den Aufgabenstellungen vertraut zu machen

- zu Beginn des Termins Lösungsansätze aufzeigen können
- in den 90 Minuten gemeinsame Diskussion einzelner Themenbereiche
- zum Ende des Termins alle Aufgaben abschließend bearbeitet

- “gelbe Karte” als Signal bei einer schlechten Vorbereitung
- bei unzureichender Vorbereitung ist der Ausschluss möglich

Organisation Praktikum Forts.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Praktikumsdurchführung Forts.:

- Protokollerstellung
  - Ihre Praktikumsdokumentation können Sie jederzeit vorzeigen.
  - Hieraus entsteht ihre Projektdokumentation für den 6. Termin.

- 6. Termin:
  - Lauffähige Software und gute, vollständige Dokumentation
  - Dokumentieren Sie so, wie Sie dies selbst mit den Kenntnissen vor dem ersten Praktikumstermin erwarten.
  - Erstellung eines Benutzerhandbuchs
  - Gesamtabnahme entscheidet über Bestehen (Ihr Kunde ist ihr Dozent :-)

Rahmenbedingungen

- Tausch nur am 1. Termin mit Partner möglich
- Anwesenheitspflicht (Ausnahme: Krankheit / Attest)
- Es gibt keinen Freischuss! Nicht fertig gestellte Teile eines Termins müssen spätestens zum nächsten Termin nachgeliefert werden.
- Nach dem 6. Termin gibt es keine weiteren Termine!

Praktikumsinhalt

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Ziel: Implementierung einer Ausschankstation mit Waage und Pumpe

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

- Entwicklungsumgebung:
  - Linux Kubuntu, GNU-Arm Tools, ARM AT91EB63 Board, HW-Debugger (offene Tutorien für die Nutzung der Hardware)
- Simulationen (liefern Ihnen keine Hardwareeigenschaften)
  - TechnoWiki des FBI: https://wiki.h-da.de/fbi/technische-systeme/
  - gnuarm.org (mittlerweile gehostet auf http://www2.amontec.com/gnuarm/) mit cygwin und Source Navigator
  - ARM-Tool-Chains z.B. devkitARM, emIDE

Gliederung

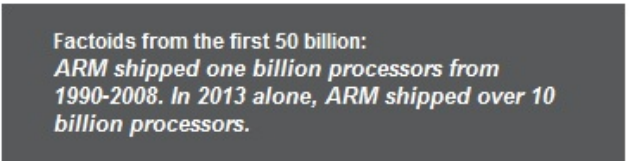
- **Einleitung**
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode / Capture Mode
- Softwareinterrupt (SWI)
- Serielle Schnittstelle
- Signalverarbeitung
  - Analog-Digital Wandlung / Digital-Analog-Wandlung
- Weiterführende Themen
  - Thumb Befehlssatz / Systemstart

Mikroprozessormarkt

Im Praktikum wird der ARM7TDMI verwendet.

Bedeutung von ARM (Advanced Risc Machines)

- Mehr als 90% aller mobilen Geräte (Smartphones, Tablets, …) beinhalten einen ARM Prozessor
- Datenbank benutzter ARM Prozessoren z. B. auf www.jbenchmark.com/ace
- 12.2.2007: 5 Milliarden ARM Prozessoren in mobilen Geräten
- Okt. 2010: 19 Milliarden ARM Prozessoren
- 2013: 50 Milliarden ARM Prozessoren http://www.50billionchips.com/



ARM-Architekturen und -familien

Architektur	Familie(n)	Erscheinungsjahr	Takt
ARMv1	ARM1	1985	4 MHz
ARMv2	ARM2, ARM3	1986, 1989	8–25 MHz
ARMv3	ARM6, ARM7	1991, 1993	12–40 MHz
ARMv4	ARM7TDMI, ARM8,	1995	16,8–75 MHz,
	StrongARM	1997	203–206 MHz
	ARM9TDMI		180 MHz
ARMv5	ARM7EJ, ARM9E, ARM10E,	2002	104–369 MHz
	XScale		133–1250 MHz
ARMv6	ARM11,	2002	400–772 MHz
	ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1		bis 200 MHz
ARMv7	ARM Cortex-M3, ARM Cortex-M4	2004	
	ARM Cortex-A (A8, A9, A5, A15, A7 und A12),	2005	bis 2 GHz
	ARM Cortex-R		
ARMv8	ARM Cortex-A53, ARM Cortex-A57	2013	3 GHz

Quelle: Wikipedia.de (ARM Architektur)

Die Familien sind nach unterschiedlichen Kriterien entworfen:

z.B. minimaler Stromverbrauch, maximale Rechenleistung,
ab ARMv7 auch Multi-Core-Designs

Mikroprozessor / Mikrocontroller

- **Mikroprozessor: Mikrorechner auf einem Chip**
  - Bausteine des Prozessors auf einem Mikrochip
  - Integrierter Schaltkreis (IC) vereinigt
- **Mikrocontroller (MC): zusätzlich Peripherie integriert**
  - 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
- **Oft Verzicht auf sonst übliche Prozessorbausteine bei MCs**
  - Speicherverwaltungseinheit (Memory Management Unit MMU)
  - Virtualisierungsfunktionen (wie Intel VT, AMD Virtualization)
  - Spezialbefehle für Vektorverarbeitung, Hyper-Threading, …

Anwendungsfelder

Klassische „Embedded" Anwendungen

- 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 Fahrassistenten, ...
- in der Automatisierung
  - das Steuern und Regeln von Prozessen,
  - das Überwachen von Prozessen,
  - das Regeln von Materialflüssen,
  - die Steuerung von Fertigungs- und Produktionsanlagen, ...

Begriffe - Embedded Systems

Ein **eingebettetes System** benötigt **Sensoren** und **Aktoren**, um mit seiner Umgebung wechselwirken zu können.

Weitere Anwendungsfelder

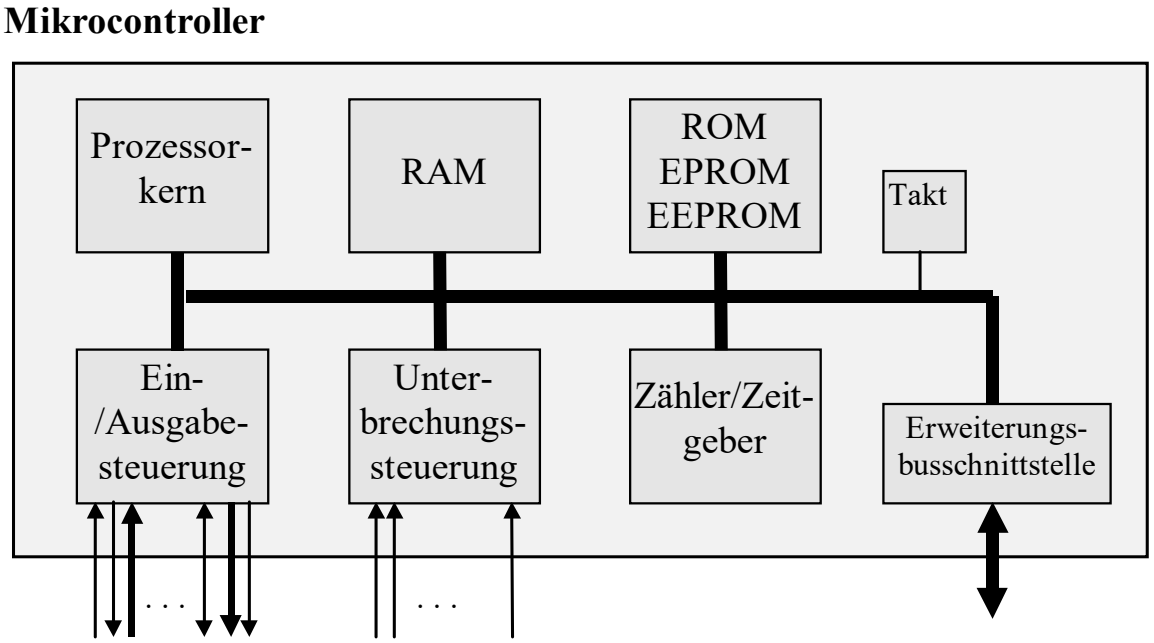
„Embedded"-Anforderungen in anderen Bereichen

- Mobile Devices
  - Embedded-Anforderungen an Rechner mit User-Front-End
    - Multitouch Displays
    - GPS-Sensoren,
    - Funkprotokolle,
    - Lage und Beschleunigung,
    - Fingerabdruck, …

- IT-Infrastruktur
  - Embedded-Anforderungen an IT-Komponenten
    - Festplatten-Controller
    - RAID-Controller
    - Netzwerk-Router
    - Netzwerkkomponenten zur Verschlüsselung/Kompression
    - Bandroboter, …

Abgrenzung zu Mikroprozessoren

Ein **Mikrocontroller** ist ein Ein-Chip-Mikrorechner **mit** aufgabenspezifischer Peripherie.



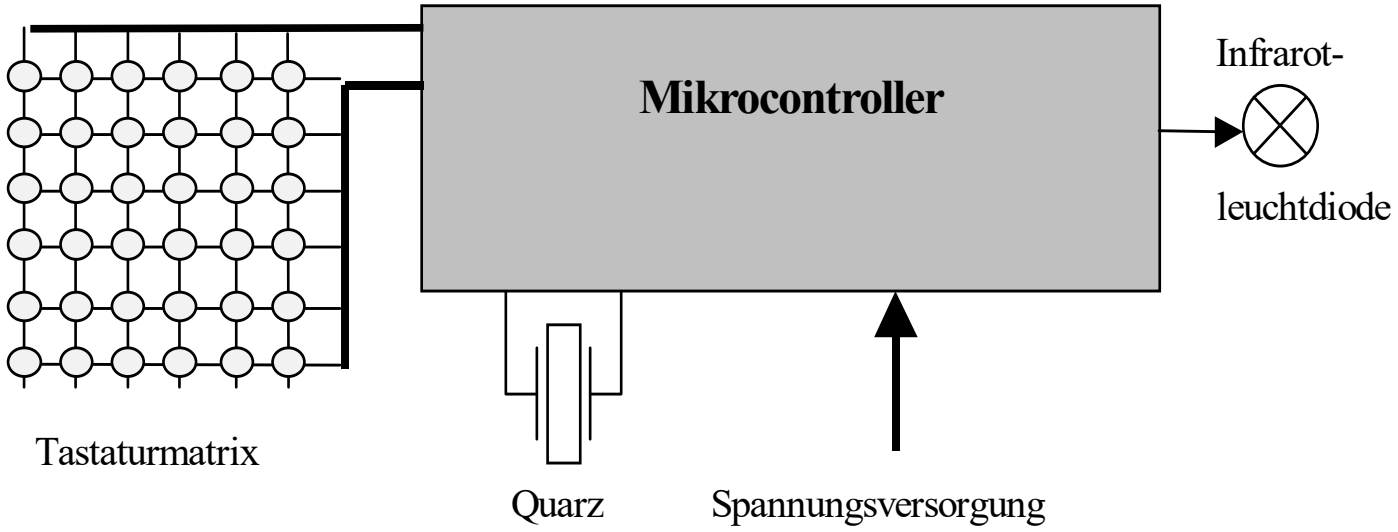
Für komplexe Chips wird oft der Begriff **System-on-Chip** (Ein-Chip-System) verwendet. Sie integrieren alle oder fast alle Bestandteile eines Systems in sich.

Mikrocontroller - Designziele

Ziel: Möglichst wenige externe Bausteine für eine Steuerungsaufgabe

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

Beispiel: Fernbedienung



Mikrocontroller - Speicher

- integrierter Festwert- und Schreiblesespeicher
- Aufnahme von Programmen und Daten
- Vorteil: Einsparung von Anschlüssen und Dekodierlogik 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 FLASH (ROM, PROM, EPROM, EEPROM)

Serielle und parallele Ein-/Ausgabekanäle

- grundlegenden digitalen Schnittstellen eines Mikrocontrollers
- seriell oder parallel
- synchron oder asynchron

AD/DA-Wandler

- grundlegende analoge Schnittstellen eines Mikrocontrollers
- Anschluss analoger Sensoren und Aktoren
- Auflösung und Wandlungszeit sind die wichtigsten Größen
- AD-Wandler sind häufiger anzutreffen als DA-Wandler

Mikrocontroller – Zähler und Zeitgeber

- Zähler (Counter) und Zeitgeber (Timer) sind im Echtzeitbereich ein wichtiges Hilfsmittel.
- **Echtzeiteigenschaften** entstehen, wenn ein System auf Ereignisse in einem vorgegebenen Zeitrahmen reagieren muss (**Reaktionszeit**). Bei der Zuverlässigkeit der Einhaltung von Echtzeitbedingungen unterscheidet man
  - **harte** Echtzeitbedingungen (hard real-time): garantierte Einhaltung einer Reaktionszeit
  - **weiche** Echtzeitbedingungen (soft real-time): statistische Garantie der Einhaltung einer Reaktionszeit; typischerweise wird die Zeit eingehalten. Dies ist aber nicht für alle Fälle garantiert.
- Zähler und Zeitgeber sind für eine Vielzahl unterschiedlich komplexer Anwendungen einsetzbar wie

  Zählen von Ereignissen, Messen von Zeiten, Pulsweitenmodulation, Frequenz- oder Drehzahlmessung, …

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

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**

Vergleiche: Lokführer müssen alle 30 Sekunden einen Knopf im Führerstand betätigen.

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

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, ...)

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

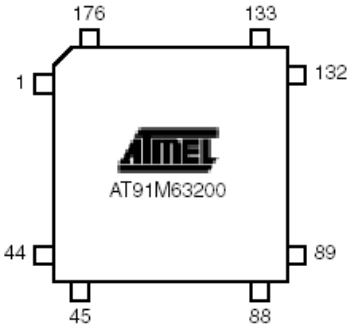
Direkter Speicherzugriff - Direct Memory Access (DMA)

- 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
- Erfordert eine Logik, die gleichzeitige Zugriffe durch den Prozessorkern und die Peripheriebausteine verhindert
- Meist in Mikrocontrollern gehobener Leistungsklasse zu finden

Mikrocontroller - Anforderungen

Allgemeines:

ein möglichst vollständiges Mikrocomputersystem auf einem Chip also Mikrocontrollerkern (Core), Speicher, Peripherie



Nichtfunktionale Eigenschaften:

Kosten: maximale funktionelle Integration bei möglichst geringen Systemkosten

Performance: möglichst hohe Verarbeitungsleistung

Energieverbrauch: möglichst geringer Strombedarf insbesondere in Bereichen ohne stationäre Stromversorgung

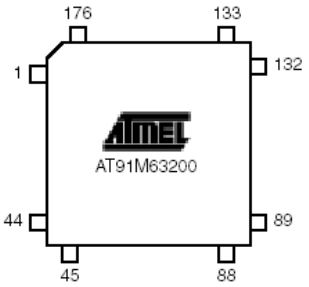
Zuverlässigkeit: Betrieb über lange Laufzeiten in kritischen Umgebungen (Temperatur, Feuchtigkeit, elektromagnetische Verträglichkeit, Schwankungen in der Spannungsversorgung, Strahlungsresistenz, …)

Wartbarkeit: dauerhafter Betrieb häufig ohne Möglichkeit der Aktualisierung

Mikrocontroller – Designaspekte

Standardmikrocontroller (AT91M63200)

abhängig vom Design universell einsetzbar
ohne spezifisches Marktsegment



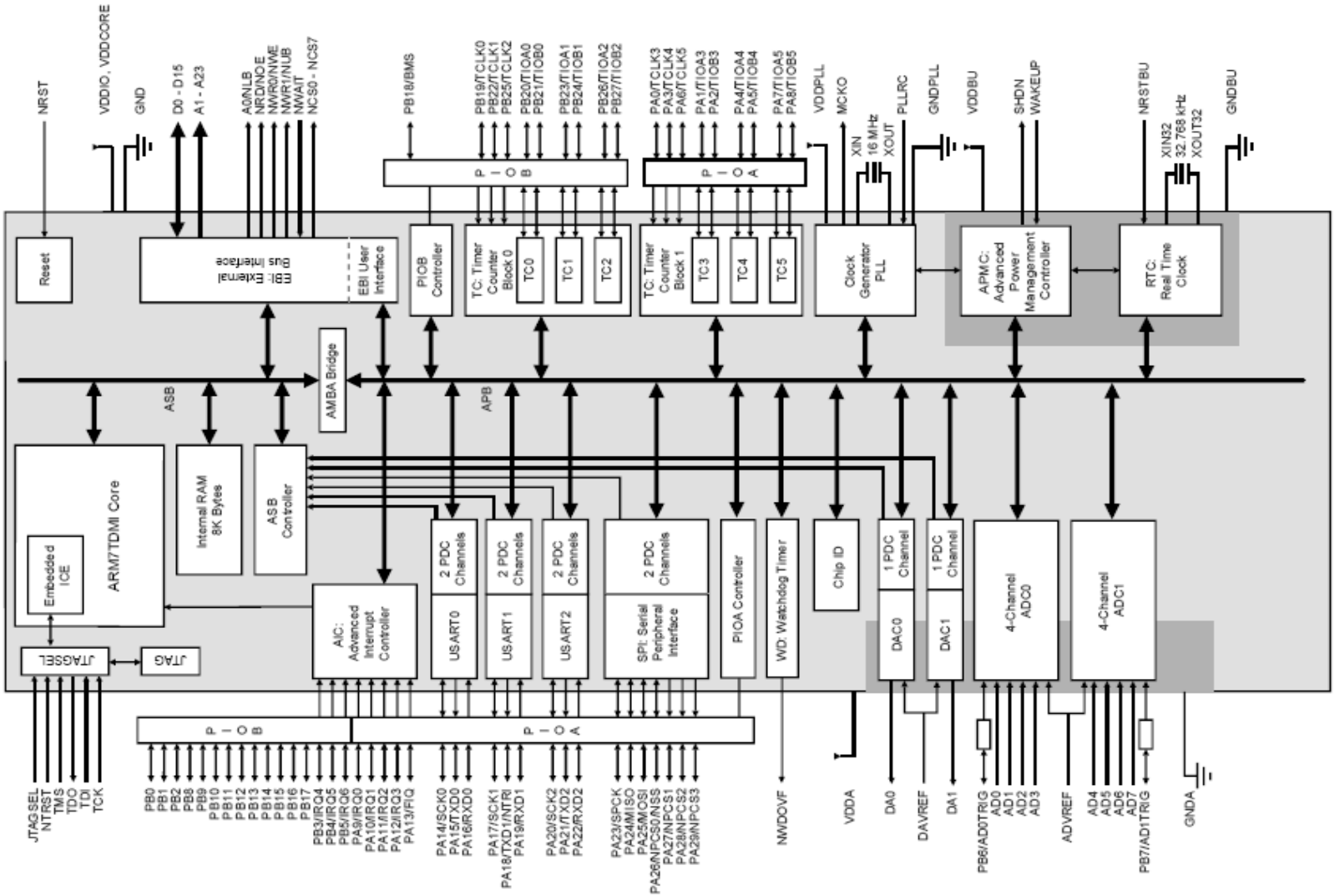
Kundenorientierte Mikrocontroller

in Einschränkung universell einsetzbar; zugeschnitten auf bestimmtes Marktsegment; hohe Stückzahl für konkrete Projekte

Ein MC besteht aus

dem Mikrocontrollerkern (Core): On-Chip integrierte CPU beinhaltet komplexes Steuerwerk, ALU, Register, Bussystem **und der benötigten On-Chip Peripherie:**
Taktoszillator, **IO Ports, Timer**, A/D D/A Wandler, **Interrupt Controller**, Watchdog Timer, Schnittstellen (**USART**,...)

Beispiel - ATMEL AT91M55800



Programmierung der Peripherie

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Der Prozessorkern muss seine Peripheriebausteine ansprechen und steuern können. Wie spreche ich die Peripherie an ?

Memory Mapped I/O
(z.B. ARM-Architektur)
Die Peripherie wird in den Adressraum des Hauptspeichers eingeblendet und direkt über das Lesen und Schreiben an die Adressen gesteuert. Die Inhalte An den Adressen sind die Inhalte der Register der E/A Bausteine.

Die Übersicht über alle eingeblendeten (mapped) Bausteine gibt die **Memory Map** des Mikrocontrollers.

Spezielle E/A Maschinenbefehle
(z.B. Intel x86 Architektur)
Die Peripherie wird über E/A-Ports angesprochen, die wiederum über Spezialbefehle (IN/OUT) ausgewählt und beschrieben oder gelesen werden.

Programmierbeispiel:

```
// Gib an dem Baustein mit der
// E/A Adresse 20H den Wert 1 aus
MOV AX, 1   // Lade den Wert 1 in
            // Register AX
OUT 20H, AX // Spezialbefehl, lade den
            // Inhalt des Registers AX
            // an den E/A Port 20H
```

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
Reserved		

Speicheraufteilung

- .text
  - legt einen Textbereich an
- .data
  - legt einen Datenbereich an
- .comm symbol, size
  - legt ein Symbol in die globale bss Section für uninitialisierte Daten
- .global
  - nimmt ein symbol in die globale Symboltabelle auf

- .word Ausdruck
  - legt einen intitialisierten Speicher an
- align #Bits
  - sorgt dafür, daß die nachfolgende Anweisung auf einer Speicherstelle steht, deren unterste #Bits 0 sind
- label:
  - legt ein Programmlabel (Marke) an
- .end
  - das Ende des Programms

Sichtbarkeit von HW-Eigenschaften in C

- Zusammengesetzte Datenstrukturen
  - Alignment der Elemente auf Zugriffsgrenzen
- Deklaration von Variablen als **`volatile`**
  - engl. flüchtig, beweglich
  - Speicherinhalte, die sich ohne Einfluss des Prozessors verändern
- Deklaration von Variablen als **`static`**
  - Zugreifbarkeit wie bei einer globalen Variable aber mit Reduktion der Sichtbarkeit auf Modulgrenzen (bei C++ Einschränkung der Sichtbarkeit auf Klasse)
- Schiebeoperationen
  - Links-Shift: **`1 << 14`** schiebt eine „1“ auf das 14. Bit in einem Register
  - Rechts-Shift **`0x0F000000 >> 24`** wird zu 0x0000000F

Sichtbarkeit von HW-Eigenschaften in C

- Arithmetische Operatoren für UND / ODER
  - Die Operatoren **&|** verknüpfen Operanden bitweise
  - Der Operator ~ invertiert Bitmuster bitweise
    (anders als die logischen Operatoren &&, || und !)
  - Der Operator ^ implementiert das bitweise XOR
- Parameterübergabe an Funktionen
  - C kennt „Call by value“ und „Call by reference“
    -> value = Wert
    -> reference = Zeiger = Adresse
  - ARM Procedure Call Standard (APCS) definiert, wie Daten zwischen aufrufender und aufgerufener Funktion ausgetauscht werden
  - Möglichkeit der Übergabe per Register und Stack
  - Rückgabewerte mittels **`return`**

Hardwarenahe Deklarationen in C

- Hardwarebausteine, die in einen Speicherbereich „gemapped“ sind, werden als zusammengesetzter Datentyp (Record) deklariert.
- Adressen werden als Konstanten bekannt gemacht.

```
// Typische Inhalte von Deklarationsdateien (Header-Files)
// Vergleichen Sie mit den Inhalten des Verzeichnisses h
// im Praktikum

// Variablentyp fuer Register mit 32 Bit Breite bei einer
// 32-Bit-Prozessorarchitektur
typedef volatile unsigned int register32;

// Deklaration einer Struktur, die einer Register-Map entspricht
typedef struct {
        register32 REGISTER0;
        register32 REGISTER1;
} structBaustein;
```

```
// Praeprozessordirektive fuer eine Konstante, die die
// absolute Basisadresse eines Bausteins darstellt
#DEFINE baustein_basisadresse ((structBaustein*) 0x00FF0000)
```

Sichtbarkeit von HW-Eigenschaften in C

- Zeiger und Adressen
  - Der Begriff des Zeigers (engl. Pointer) in C entspricht einer Speicheradresse
- Beispiele für die Verwendung von Adressen

```
// Direkte Zuweisung benötigt Cast-Operator
volatile unsigned int *reg = (???*) 0xFF000000;

// mit Deklarationsdateien
structBaustein *bausteinA = baustein_basisadresse;
bausteinA->REGISTER0 = 0;
```

- Einzelne Bits setzen

```
// Funktion einzelner Bits mittels Präprozessordirektiven festlegen
#DEFINE BAUSTEIN_LED1 (1 << 12)
#DEFINE BAUSTEIN_LED2 (1 << 13)

// das Bit 12 setzen ohne andere Bits zu verändern
bausteinA->REGISTER0 = bausteinA->REGISTER0|BAUSTEIN_LED1;
// oder in Kurzform auch so
bausteinA->REGISTER0 |= BAUSTEIN_LED1;
```

Registerfelder in C

- Häufig werden mehrere Register als Vektor von Registern zusammengefasst.

```
// Deklaration einer Struktur, die einen Registervektor enthaelt
typedef struct {
    register16 REGISTER0;    // Offset 0
    // Alignment auf 32-Bit-Grenze
    register16 reserved0;    // Offset 2
   // Deklaration eines Vektors aus 32-Bit-Registern
    register32 VECTOR[4];    // Offset [4, 8, 12, 16]
} structBaustein; // Groesse im Speicher insgesamt 20 Byte

// Funktionsbeschreibung eines Registers in einem Vektor
#DEFINE VEC_TIMER0 0
#DEFINE VEC_TIMER1 1
#DEFINE VEC_INT0 2
#DEFINE VEC_INT1 3
…
// Zeiger auf den Baustein
structBaustein *baustein1 = baustein_basisadresse;
// Zuweisung zu dem Element VEC_TIMER1 des Vektors
baustein1->VECTOR[VEC_TIMER1] = 0;
```

Optimierungsstufen bei Compilern

- Unterschiedliche Optimierungen können auf Basis des gleichen Quelltextes zu sehr unterschiedlichem Maschinencode führen.
  Optimierungsanalysen stellen fest, ob Ergebnisse weiterverwendet werden und Abhängigkeiten hierzu.

- Allerdings sind Analysen rein auf Quelltextbasis d.h. es wird nur der Ablauf analysiert, den der Prozessorkern durchführt. Andere Einflüsse werden nicht erkannt.

- Optimierungsstufen sollten bei hardwarenaher Programmierung vorsichtig eingesetzt werden.

Gliederung

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

Power Management

Power Management (Power Saving Modi, Stromsparmodi)

Eine niedrige Stromaufnahme ist wichtig für batteriebetriebene Anwendungen.

Möglichkeiten:
- Reduktion der Taktfrequenz
- Taktabschaltung von Prozessor und/oder Peripherie

Beispiel ARM-Board:
Es gibt zwei Takte auf dem Board, den System-Takt (CPU) und den Peripherie-Takt.

Grundeinstellung:

Takt für CPU	ist beim Reset eingeschaltet
Takt für Peripherie	ist beim Reset ausgeschaltet

Power Management – Memory Map

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

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	-	-

Layout des Peripherie Clock Registers:

Power Management – Code (1)

Ansprechen des Peripheral Clock Enable Register

```
#define PMC_PCER  ((volatile unsigned int *) 0xFFFF4010)

int main(void) {
*PMC_PCER = 0x4000; // Peripheral Clock für PIOB
                    // einschalten
// …
}
```

Power Management

volatile Typqualifikator *(Quelle: Wikipedia)*

Volatile ist ein Zusatz bei der Deklaration von Variablen in C und C++.

Dadurch wird festgelegt, dass sich der Wert der Variablen außerhalb des aktuellen Programmkontextes ändern kann, beispielsweise durch externe Hardware.

Bei der Generierung des Maschinen-Codes aus einem in C oder C++ geschriebenen Programm verhindert die Kennzeichnung einer Variablen als volatile eine in diesem Fall die Funktionalität beeinträchtigende Optimierung, so dass das Programm immer auf den tatsächlich in der Hardware vorhandenen Wert zugreift.

Beispiel in C

Ohne den Zusatz volatile könnte der Compiler die Schleife im folgenden Programmausschnitt durch eine einfache Endlosschleife ersetzen und die Variable ***status*** wegoptimieren:

```
static volatile int status = 0;
void poll_status( void ) { while ( status == 0 ) ; }
```

Power Management

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**.

Power Management

C und Assembler bei der Optimierungsstufe –O0

01-volatile\01-volatile.c | boot\boot_ice.S

```
1   // Beispiel zur Wirkung des Keywords volatile
2
3   // Definition von globalen Variablen
4   int globvar1;
5   volatile int globvar2;
6
7
8   // Einsprung
9   int main(void) {
10      int locvar1;
11
12      globvar1 = 0xAAFFAAFF;
13      globvar2 = 0xFFAAFFAA;
14
15      locvar1 = globvar1;
16      locvar1 = globvar2;
17
18      globvar1 = locvar1;
19      globvar2 = locvar1;
20
21      return 0;
22  }
```

Disassembly

Function:

Frame start: 02040000

```
0x020001D0      push    {r11}           ; (str r11, [sp, #-4]!)
0x020001D4      add     r11, sp, #0
0x020001D8      sub     sp, sp, #12
0x020001DC      ldr     r3, [pc, #84]   ; 0x2000238 <main+104>
0x020001E0      ldr     r2, [pc, #84]   ; 0x200023c <main+108>
0x020001E4      str     r2, [r3]
0x020001E8      ldr     r3, [pc, #80]   ; 0x2000240 <main+112>
0x020001EC      ldr     r2, [pc, #80]   ; 0x2000244 <main+116>
0x020001F0      str     r2, [r3]
0x020001F4      ldr     r3, [pc, #60]   ; 0x2000238 <main+104>
0x020001F8      ldr     r3, [r3]
0x020001FC      str     r3, [r11, #-8]
0x02000200      ldr     r3, [pc, #56]   ; 0x2000240 <main+112>
0x02000204      ldr     r3, [r3]
0x02000208      str     r3, [r11, #-8]
0x0200020C      ldr     r3, [pc, #36]   ; 0x2000238 <main+104>
0x02000210      ldr     r2, [r11, #-8]
0x02000214      str     r2, [r3]
0x02000218      ldr     r3, [pc, #32]   ; 0x2000240 <main+112>
0x0200021C      ldr     r2, [r11, #-8]
0x02000220      str     r2, [r3]
0x02000224      mov     r3, #0
0x02000228      mov     r0, r3
0x0200022C      add     sp, r11, #0
0x02000230      pop     {r11}
0x02000234      bx      lr
```

Power Management

C und Assembler bei der Optimierungsstufe –O3

01-volatile\01-volatile.c | 01-volatile\makefile | boot\boot_ice.S

```
1  // Beispiel zur Wirkung des Keywords volatile
2
3  // Definition von globalen Variablen
4  int globvar1;
5  volatile int globvar2;
6
7
8  // Einsprung
9  int main(void) {
10     int locvar1;
11
12     globvar1 = 0xAAFFAAFF;
13     globvar2 = 0xFFAAFFAA;
14
15     locvar1 = globvar1;
16     locvar1 = globvar2;
17
18     globvar1 = locvar1;
19     globvar2 = locvar1;
20
21     return 0;
22 }
23
```

Disassembly

Function:

Frame start: 02040000

```
0x02000000      ldr     r3, [pc, #28]   ; 0x2000024 <main+36>
0x02000004      ldr     r2, [pc, #28]   ; 0x2000028 <main+40>
0x02000008      str     r2, [r3]
0x0200000C      ldr     r1, [pc, #24]   ; 0x200002c <main+44>
0x02000010      ldr     r2, [r3]
0x02000014      mov     r0, #0
0x02000018      str     r2, [r1]
0x0200001C      str     r2, [r3]
0x02000020      bx      lr
```

Interpretation des Assemblercode –O3

```
0x02000000      ldr     r3, [pc, #28]   ; 0x2000024 <main+36>
0x02000004      ldr     r2, [pc, #28]   ; 0x2000028 <main+40>
0x02000008      str     r2, [r3]
0x0200000C      ldr     r1, [pc, #24]   ; 0x200002c <main+44>
0x02000010      ldr     r2, [r3]
0x02000014      mov     r0, #0
0x02000018      str     r2, [r1]
0x0200001C      str     r2, [r3]
0x02000020      bx      lr
```

Memory Dump ab 0x02000020

```
0x02000020:  e12fff1e 02008200 ffaaffaa 02008204
```

Interpretation des Speicherinhalts

```
0x2000024 => Adresse &globvar2
0x2000028 => Konstante 0xFFAAFFAA
0x200002C => Adresse &globvar1
```

Als „Pseudo“-C Notation

```
C: globvar2 = 0xFFAAFFAA;
        // &globvar2
        [R3] = *[0x2000024]
        // 0xFFAAFFAA
        [R2] = *[0x2000028]
        Speichern der Konstanten
        *[R3] = [R2]

// Laden der Adresse von globvar1 in R1
[R1] = &globvar1;

// => Erneutes Laden des Inhalts von
// globvar2, falls dieser sich geaendert
// haben sollte
[R2] = *[R3]

C: globvar1 = locvar1;
*[R1] = [R2]
C: globvar2 = locvar1;
*[R3] = [R2]
```

Power Management – Code (2)


```
typedef volatile unsigned int at91_reg ;
typedef struct {
at91_reg PMC_SCER ;  // System Clock Enable  Register
at91_reg PMC_SCDR ;  // System Clock Disable Register
at91_reg PMC_SCSR ;  // System Clock Status  Register
at91_reg Reserved0 ;
at91_reg PMC_PCER ;  // Peripheral Clock Enable Reg.
at91_reg PMC_PCDR ;  // Peripheral Clock Disable Reg.
at91_reg PMC_PCSR ;  // Peripheral Clock Status Reg.
} StructPMC ;
#define PMC_BASE (( StructPMC *) 0xFFFF4000)
int main(void) {
      StructPMC* pmcbase = PMC_BASE; // Basisadresse PMC
      pmcbase->PMC_PCER = 0x4200;   // Peripheral Clocks
                          // einschalten für PIOB, _____
      // …..
}
```

Gliederung

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

Parallele IO - Einführung

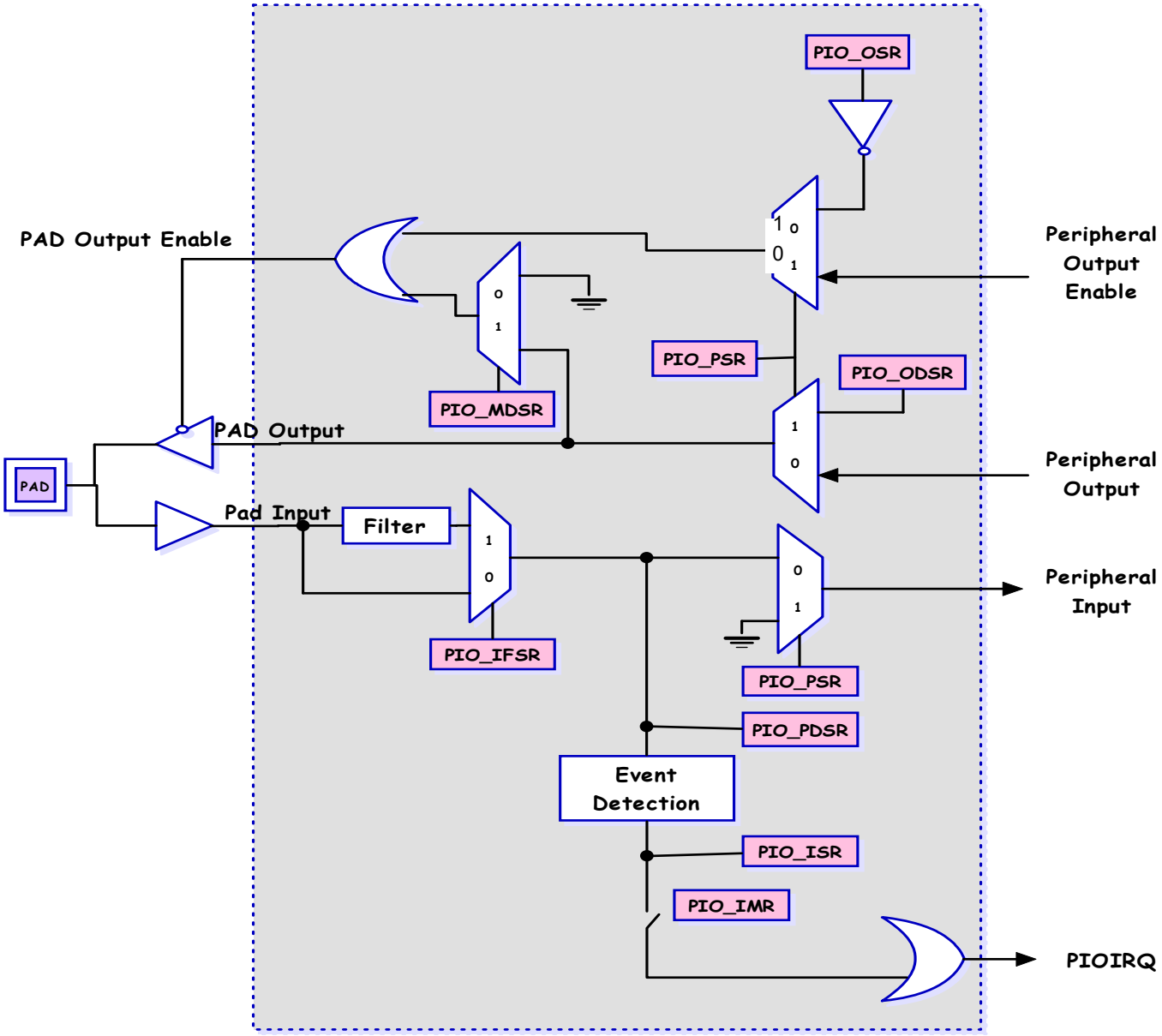
- MC ist mit seiner Außenwelt über Portpins verbunden
- Zusammenfassung einzelner Portpins zu Ports
- Parallele I/O (PIO)
- 32 Bit PIO
- digitale Ein- und Ausgabe (I/O)
  z.B. Taster, LED's, ...
- analoge Werte ebenfalls über Portpins übertragbar
  erfordert analoge Peripheriekomponenten
  AD / DA Wandler

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

Parallele IO – Register eines Ports

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 Register	PIO_IFER	Write only	-
0x24	Input Filter Disable Register	PIO_IFDR	Write only	-
0x28	Input Filter Status Register	PIO_IFSR	Read only	0
0x2C	Reserved	-	-	-
0x30	Set Output Data Register	PIO_SODR	Write only	-
0x34	Clear Output Data Register	PIO_CODR	Write only	-
0x38	Output Data Status Register	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 Register	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 Register	PIO_MDER	Write only	-
0x54	Multi-driver Disable Register	PIO_MDDR	Write only	-
0x58	Multi-driver Status Register	PIO_MDSR	Read only	0
0x5C	Reserved	-	-	-

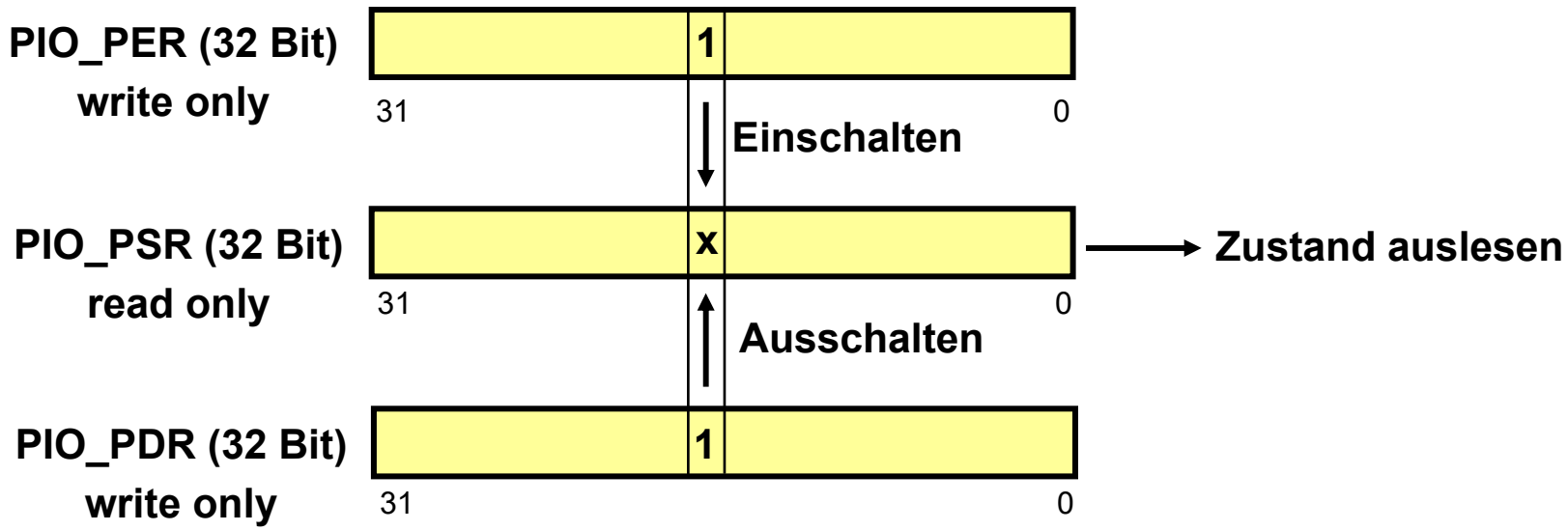
Parallele IO – Struktur eines Ports



Parallele IO – Struktur eines Ports (Status)

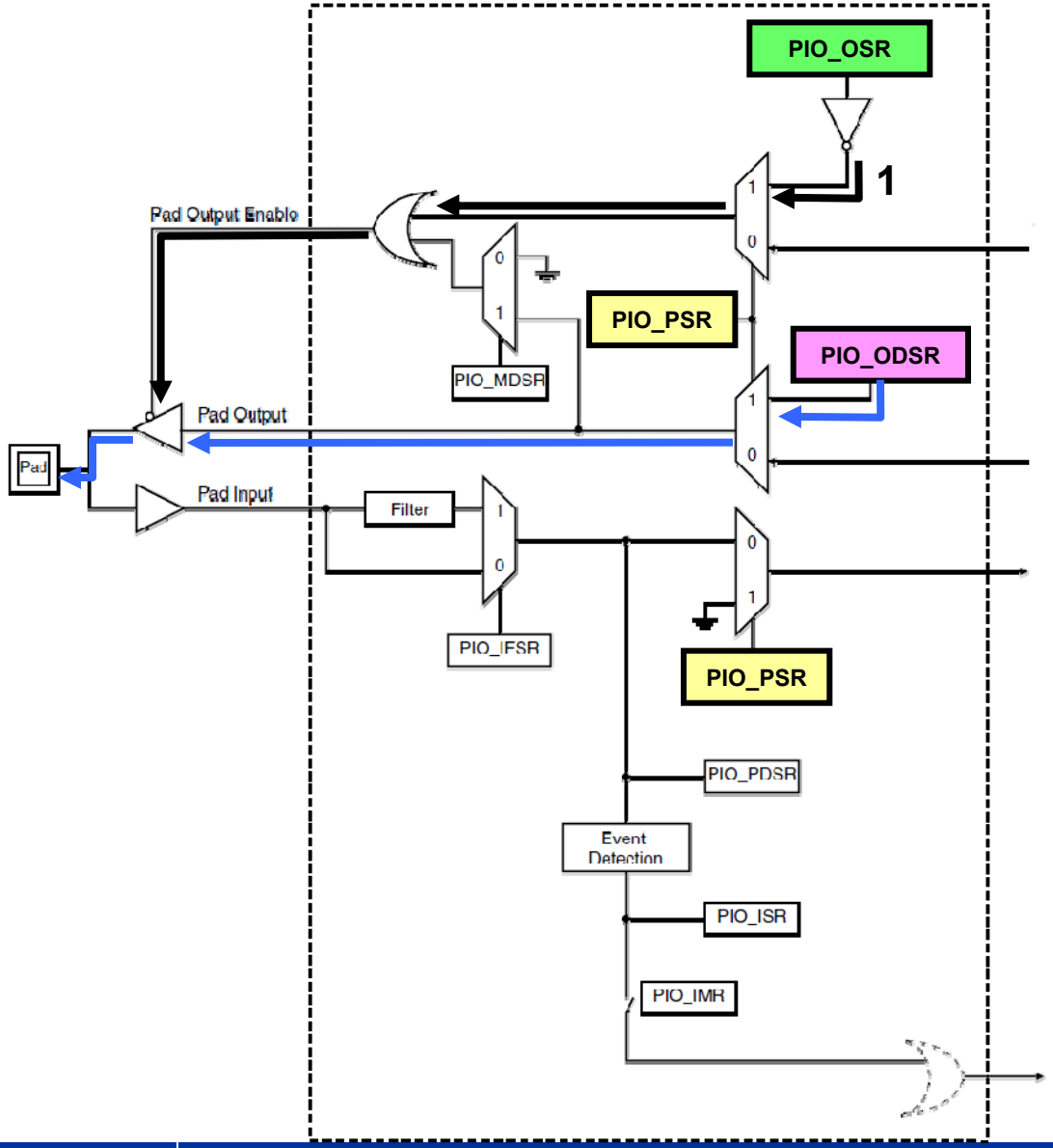
Zum Modifizieren eines Bits in einem Statusregister werden Enable-(Einschalt)- und Disable-(Auschalt-)register verwendet.

PIO_PER	PIO Enable Register
PIO_PDR	PIO Disable Register
PIO_PSR	PIO Status Register



Parallele IO – Struktur eines Ports (Output)

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK



PIO_PER	PIO Enable Reg.
PIO_PDR	PIO Disable Reg.
PIO_PSR	PIO Status Reg.
PIO_OER	Output Enable Reg.
PIO_ODR	Output Disable Reg.
PIO_OSR	Output Status Reg.
PIO_SODR	Set Output Data Reg.
PIO_CODR	Clear Output Data Reg.
PIO_ODSR	Output Data Status Reg.

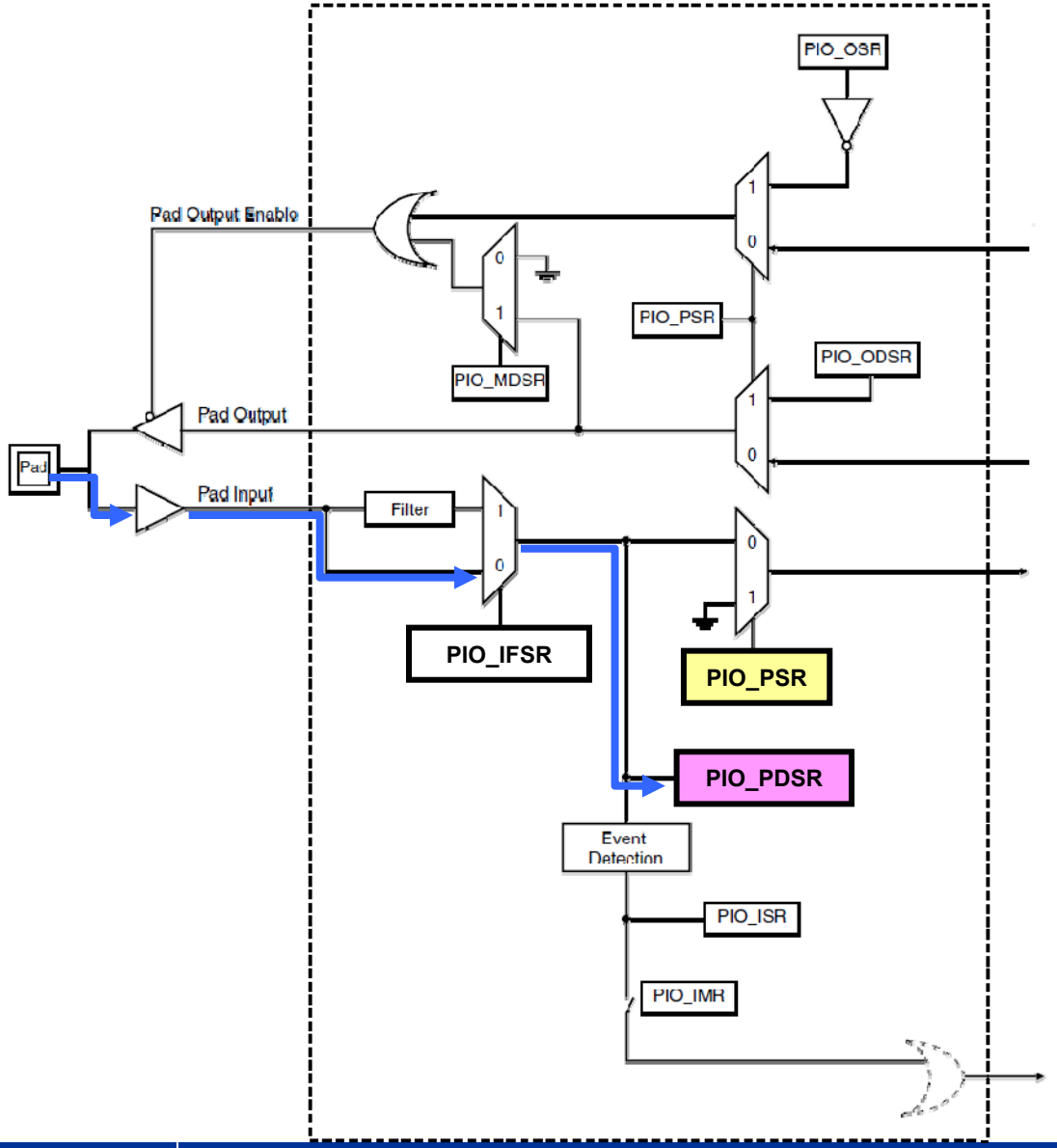
Daten werden auf Pad geschaltet

Steuerung des Datenflusses

Vergl. AT91M63200 Referenz, Seite 56

Parallele IO – Struktur eines Ports (Input)

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

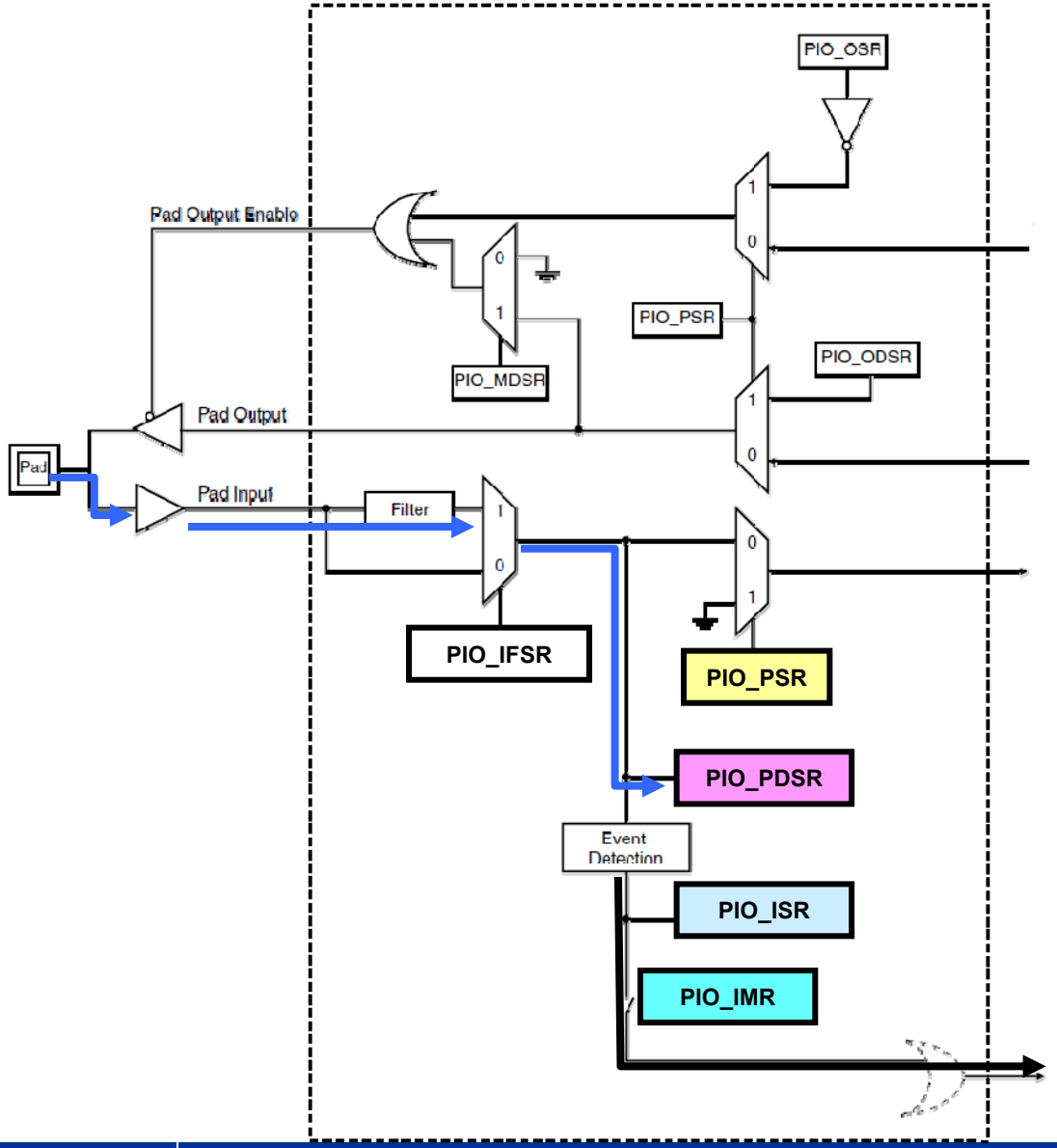


PIO_PER	PIO Enable Reg.
PIO_PDR	PIO Disable Reg.
PIO_PSR	PIO Status Reg.
PIO_IFER	Input Filter Enable Reg.
PIO_IFDR	Input Filter Disable Reg.
PIO_IFSR	Input Filter Status Reg.
PIO_PDSR	Pin Data Status Reg.

Daten werden in PDSR gespeichert (hier ohne Filterung von Glitches)

Parallele IO – Struktur eines Ports (Input)

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK



PIO_PER	PIO Enable Reg.
PIO_PDR	PIO Disable Reg.
PIO_PSR	PIO Status Reg.
PIO_IFER	Input Filter Enable Reg.
PIO_IFDR	Input Filter Disable Reg.
PIO_IFSR	Input Filter Status Reg.
PIO_IER	Interrupt Enable Reg.
PIO_IDR	Interrupt Disable Reg.
PIO_IMR	Interrupt Mask Reg.
PIO_ISR	Interrupt Status Reg.
PIO_PDSR	Pin Data Status Reg.

Daten werden in PDSR gespeichert (hier mit Filterung von Glitches)

Interrupt von PIO-Port wird signalisiert

PIO_ISR

IRQ-Quelle (Bit des Pad wird gesetzt)

Parallele IO – Port A 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

Parallele IO – Port A Belegung (2)

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	-	-	-	-	-	-

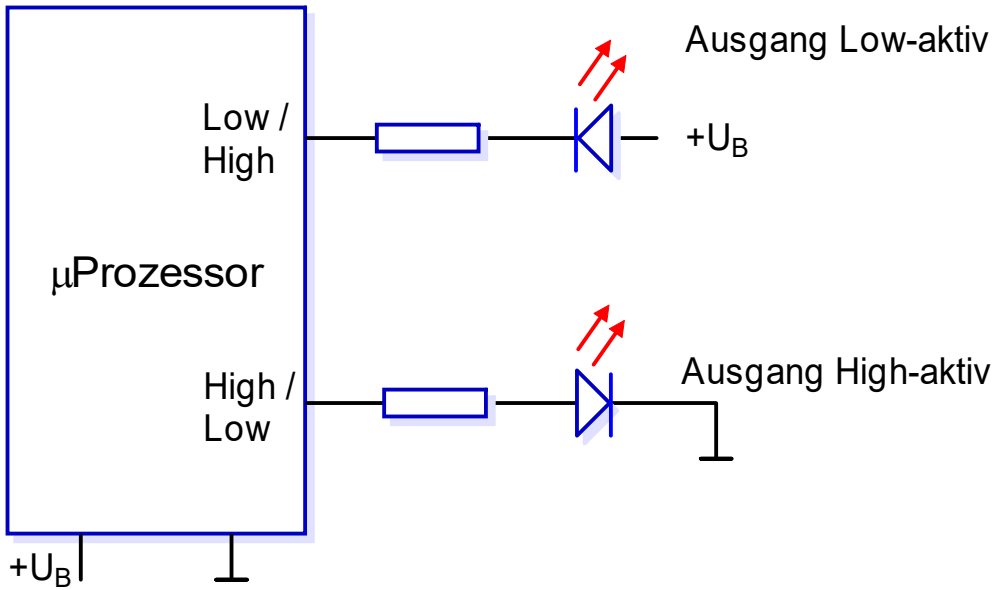
Parallele IO – Port B Belegung (1)

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	-	SW1 -	-	Input	142
4	PB4	-	SW2 -	-	Input	143
5	PB5	-	SW3 -	-	Input	144
6	PB6	-	-	-	Input	145
7	PB7	-	-	-	Input	146
8	PB8	-	LED 1 -	-	Input	149
9	PB9	-	LED 2 -	-	Input	150
10	PB10	-	LED 3 -	-	Input	151
11	PB11	-	LED 4 -	-	Input	152
12	PB12	-	LED 5 -	-	Input	153
13	PB13	-	LED 6 -	-	Input	154
14	PB14	-	LED 7 -	-	Input	155
15	PB15	-	LED 8 -	-	Input	156

Parallele IO – Port B Belegung (2)

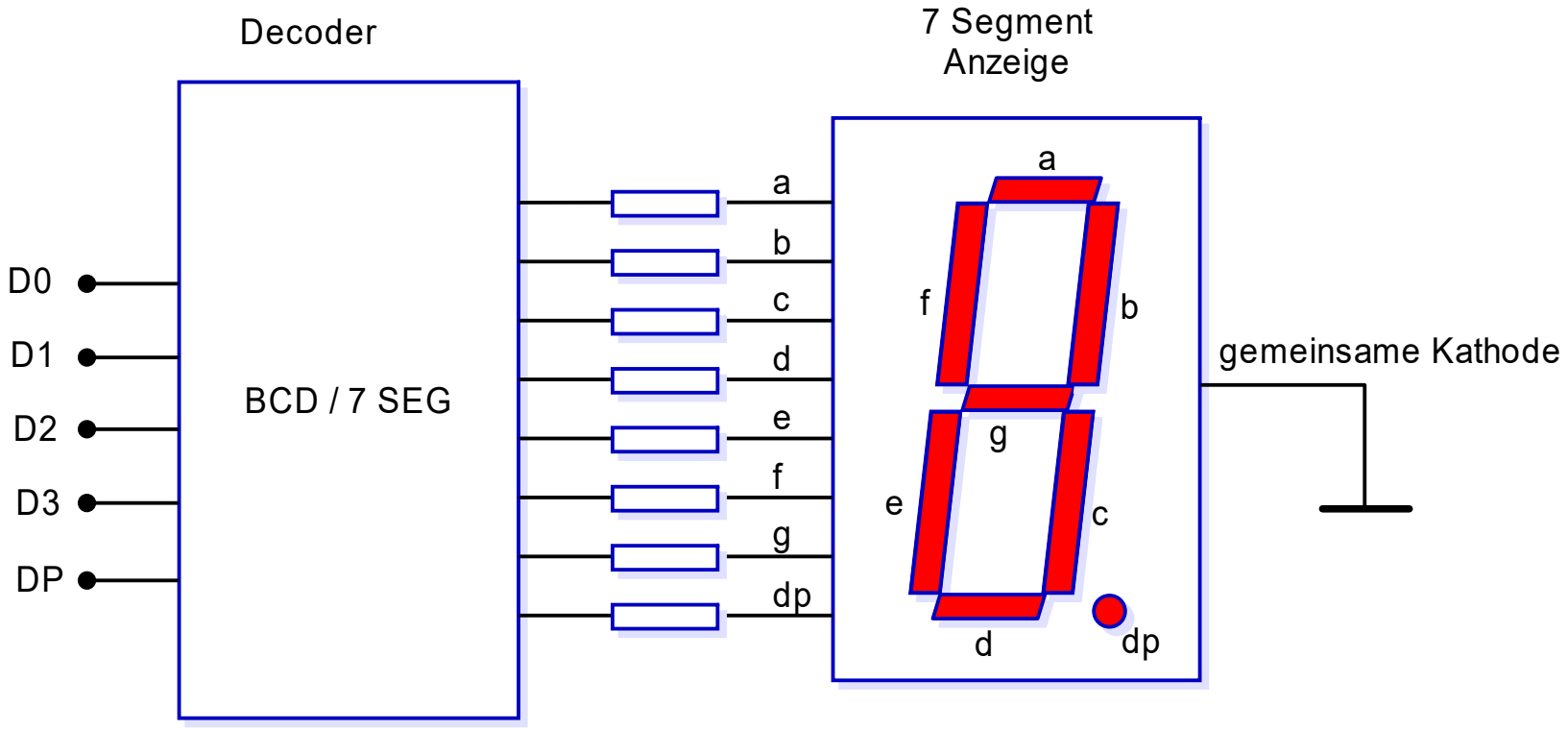
Bit Number	Port Name	Port Name	Signal Description	Signal Direction	Reset State	Pin Number
16	PB16	-	-	-	Input	157
17	PB17	MCKO	Master Clock Output	Output	Input	158
18	PB18	BMS	Boot Mode Select	Input	Input	163
19	PB19	TCLK0	Timer 0 Clock Signal	Input	Input	55
20	PB20	TIOA0	Timer 0 Signal A	Bi-directional	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	-	-	-	-	-	-

Parallele IO – LED

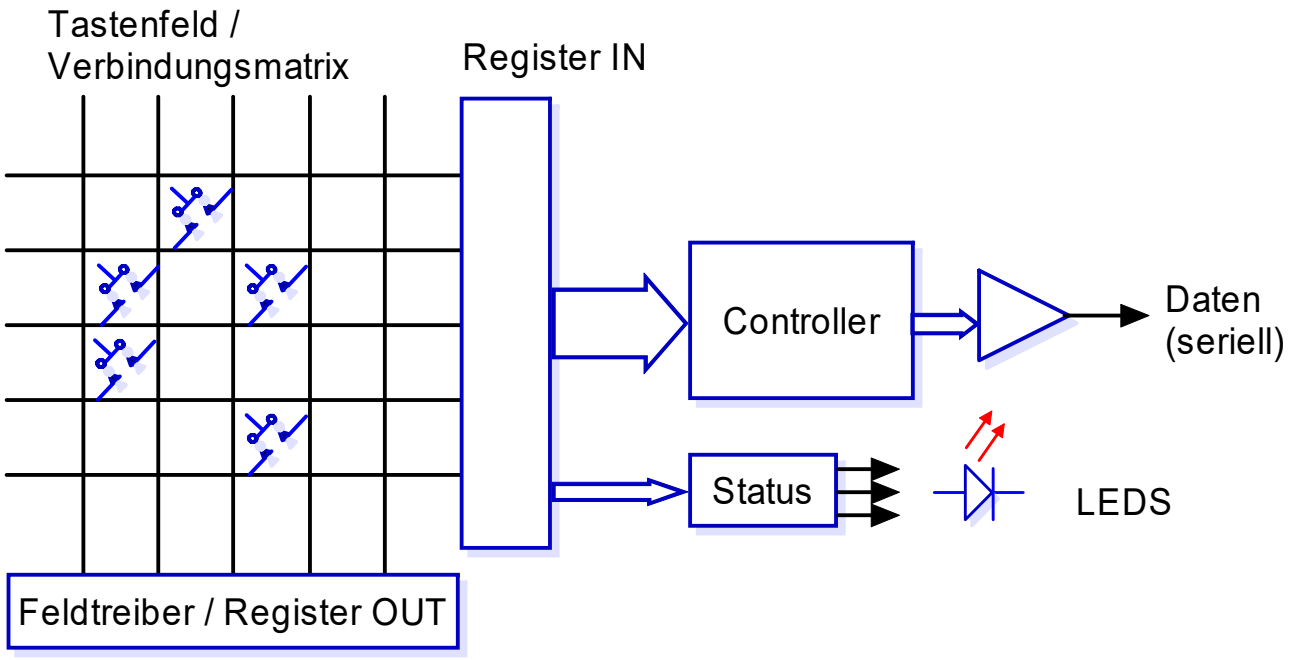


Ausgang Low-aktiv
ist häufiger,
da dann der Strom
für die Leuchtdiode
nicht vom Prozessor
zur Verfügung
gestellt werden muß

Parallele IO – 7 Segment Anzeige



Parallele IO – Tastatur



Gliederung

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

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Ereignisgesteuerte Programmierung

- Ereignisgesteuerte Programmierung liefert eine eigene wichtige Klasse von Programmen
- Wichtigstes Kennzeichen ist, dass die Steuerung des Programm-ablaufs durch externe Ereignisse und nicht nur durch die Daten des Programms selbst erfolgt
- Ereignisgesteuerte Programme sind immer dann nötig, wenn der Computer intensiv mit seiner Umwelt in Interaktion tritt
- Beispiele:
  - grafische Oberflächen
  - Kommunikation
  - Prozesssteuerung

Interrupt Handling - Ereignisse

Externe Ereignisse

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

Interrupt Handling - Mechanismen

Ereignisverarbeitung implementiert als

- 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
- Interrupt
  - „Echte" Parallelverarbeitung (Nebenläufigkeit) der Ereignisse
  - Gefahren durch Multithreading, wenn kritische Bereiche nicht geschützt sind
  - Gefahr bei zu langen Interruptroutinen oder zu vielen Interrupts

Interrupt Handling - Polling

Polling

- Synchrone Ereignisbehandlung durch **Event Loop**
  - Programme 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 sehr hoch
- 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

Interrupt Handling - Interrupt

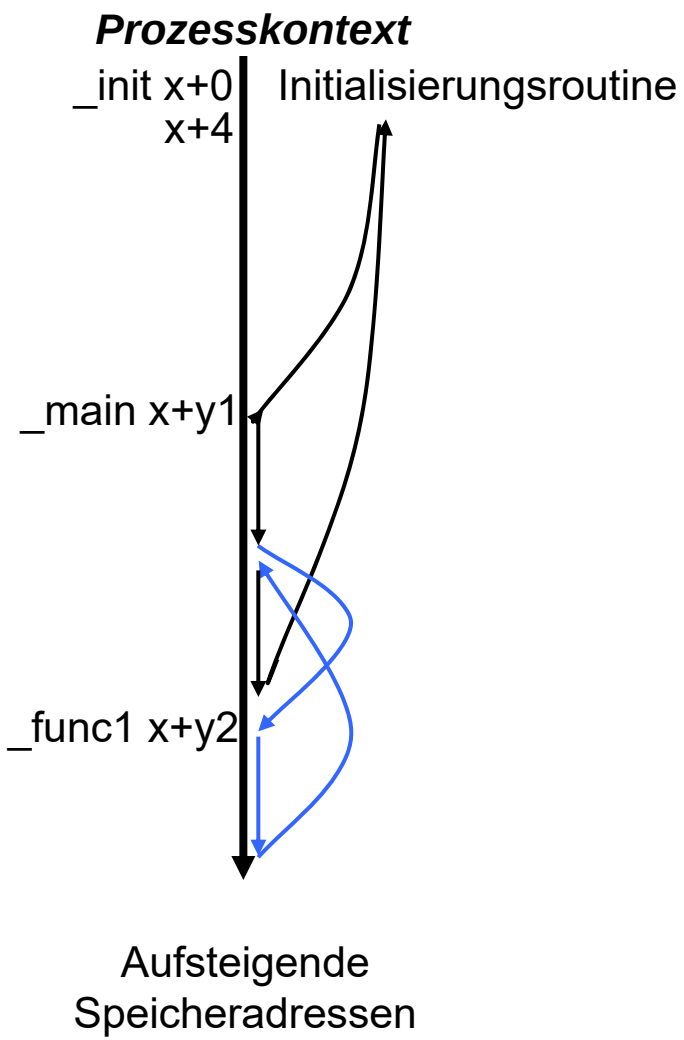
Interrupt

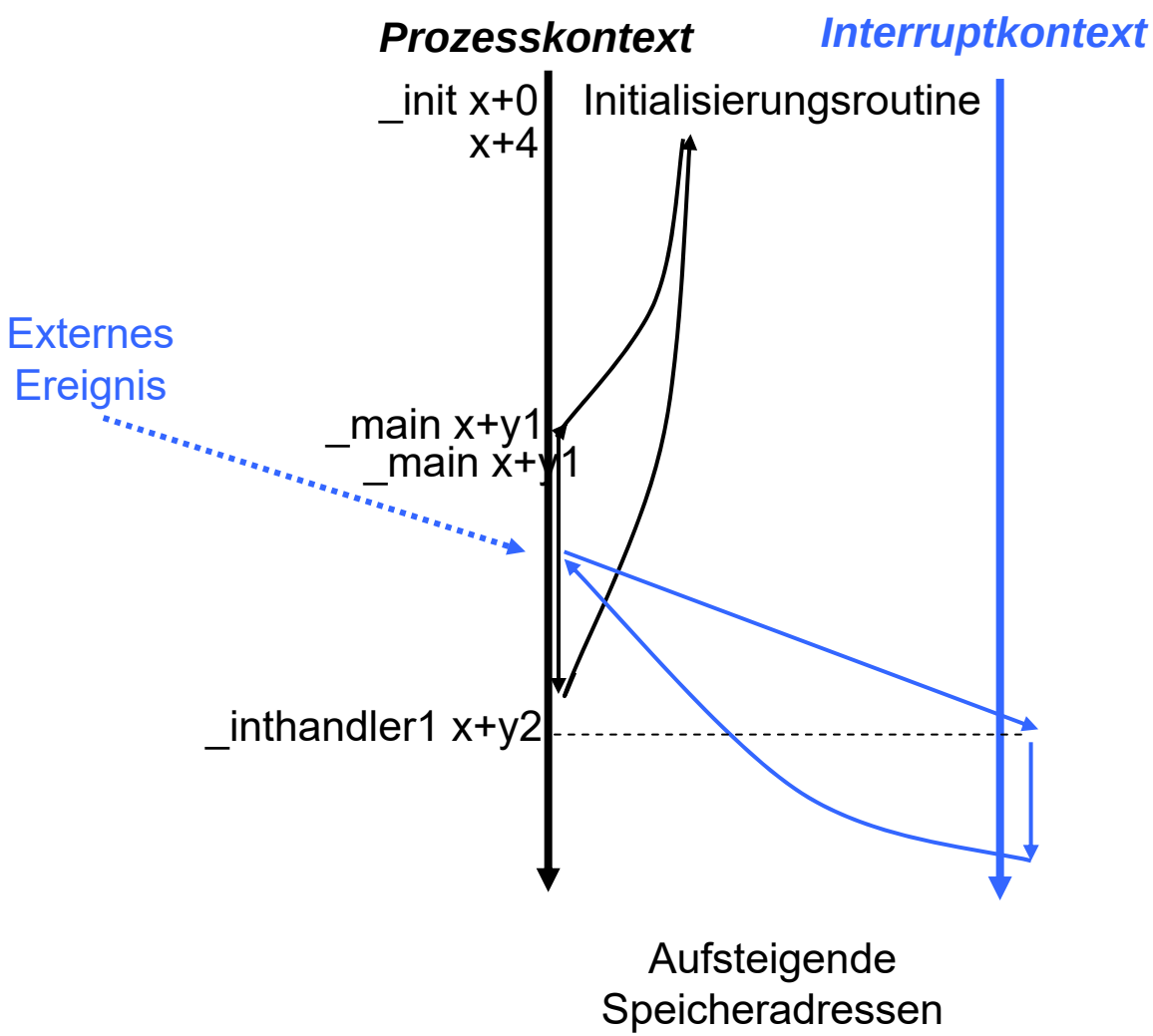
- Asynchrone Ereignisbehandlung durch Interrupts
  - laufendes Programm wird unterbrochen und eine Reaktionsroutine (Interrupt Service Routine ISR) für das Ereignis gestartet
  - Der **Kontext** der laufenden Programms (Inhalt von Registern) muss gerettet werden (vergleiche APCS Funktionsaufruf)
  - Der Interrupthandler benötigt einen eigenen Kontext, der unabhängig von dem ursprünglich laufenden Programm ist, um eine **Nebenläufigkeit** zu erreichen also keine Beeinflussung des Zustands außerhalb der Handlers
  - Interrupts besitzen eine **Priorität**, um bei Unterbrechungen durch mehrere Quellen für diejenigen mit hoher Priorität Reaktionszeiten garantieren zu können

Interrupt Handling - Kontextwechsel

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Funktionsaufrufe durch Sprünge als Ablaufsteuerung





Interrupt Handling - Folgen

Interrupt

- 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 (Shared Data Problem)

Interrupt Handling – Bewertung I

Vorteile des Pollings gegenüber Interrupts (Quelle: Wikipedia)

- Einfacher zu implementieren, da Abfrage des Geräts im Hauptprogramm erfolgen kann.
- Meist schnellere Reaktion auf das externe Ereignis. Beim Polling kann nach Erkennung des Ereignisses sofort reagiert werden. Bei Interrupts wird in eine ISR (Interrupt Service Routine) verzweigt: Kontextwechsel mit Sicherung der benutzten Register auf dem Stack.
- Geringerer Hardwareaufwand bei den Geräten notwendig, da diese nicht in der Lage sein müssen, auf das Auftreten eines Ereignisses mit einer Interrupt-Anfrage zu reagieren.
- Beispiel: ein Temperatursensor für Polling braucht im einfachsten Fall nur einen A/D-Wandler. Bei der Verwendung von Interrupts muss er jedoch neben der Möglichkeit, die Temperatur zu messen, zusätzlich auf die Änderung der Temperatur mit einer Interruptanfrage reagieren können.

Interrupt Handling – Bewertung II

Vorteile von Interrupts gegenüber Polling (Quelle: Wikipedia)

- Hauptprogramm wird deutlich einfacher und besser verständlich, weil es sich auf das Wesentliche konzentriert, ohne ständig oder von unterschiedlichen Codestellen aus Ereignisse abzufragen zu müssen.
- Das Auftreten eines externen Ereignisses wird immer überwacht. Beim Polling geschieht dies nur zu den Zeiten, zu denen das Hauptprogramm danach fragt.
- Das Hauptprogramm kann andere Aufgaben übernehmen als das externe Gerät zu überwachen. Beim Polling wird häufig sämtliche Rechenleistung für die Abfrage der Geräte aufgewendet.
- Die Kommunikationsleitungen, Datenbusse und externen Geräte werden entlastet. Im Gegensatz zum Polling wird nur mit dem Gerät kommuniziert, wenn tatsächlich ein externes Ereigniss stattfindet.
- Es ist oft stromsparender: Die CPU kann bis zum nächsten Auftreten eines Interrupts in einen stromsparenden Sleep-Mode versetzt werden (beim nächsten Interrupt automatisch wieder aktiviert).

Interrupt Handling - DMA

Ereignisbehandlung durch Direct Memory Access (DMA)

- Für sehr schnelle Reaktionen 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 (DMA) Controller zur Verfügung gestellt.

- DMA Controller können auf Interrupts reagieren und die Daten in den Speicher oder zu IO Device transferieren.

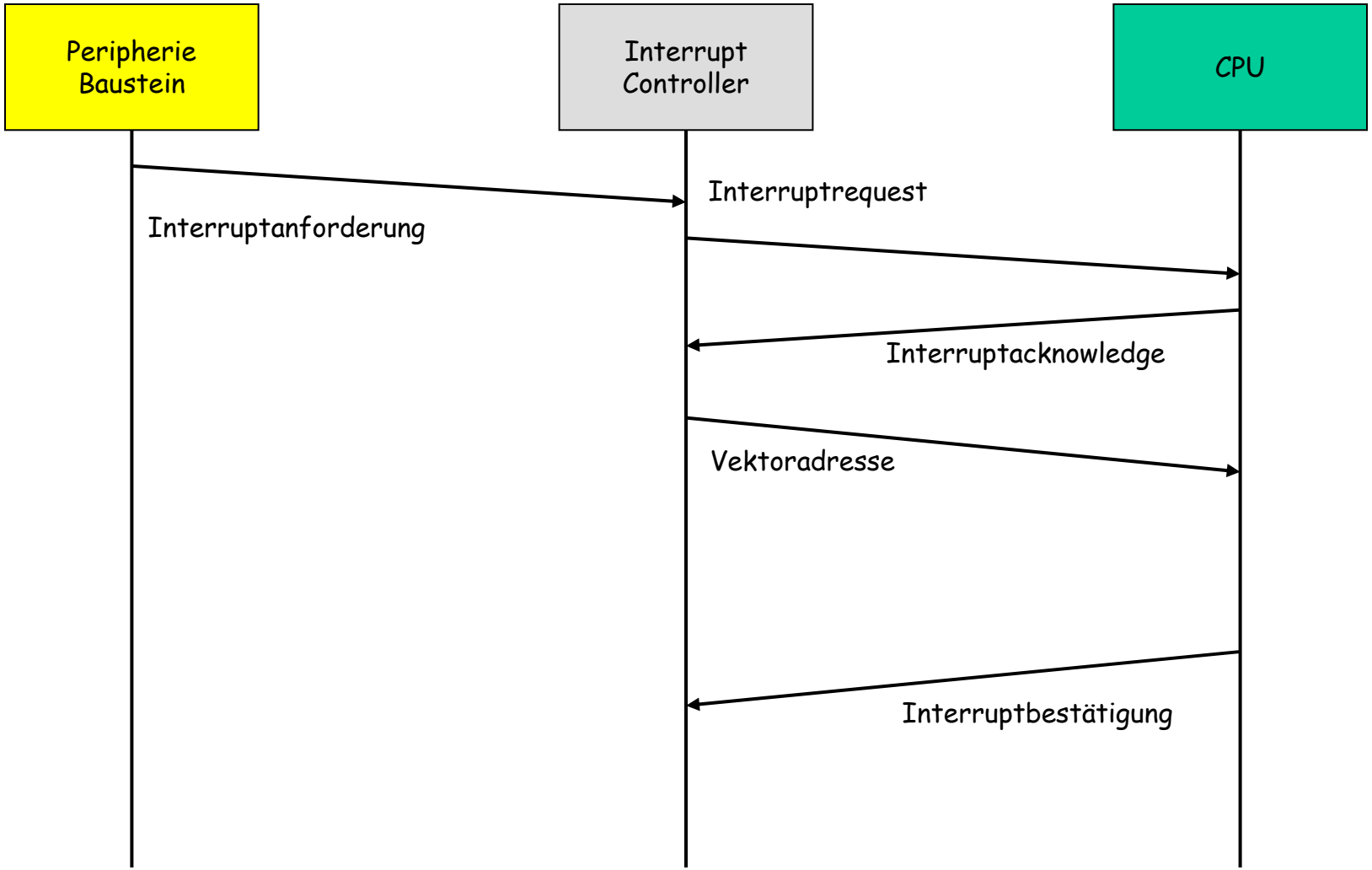
Interrupt Handling – Beispiel Polling


```
// Pollingbeispiel (round robin)

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

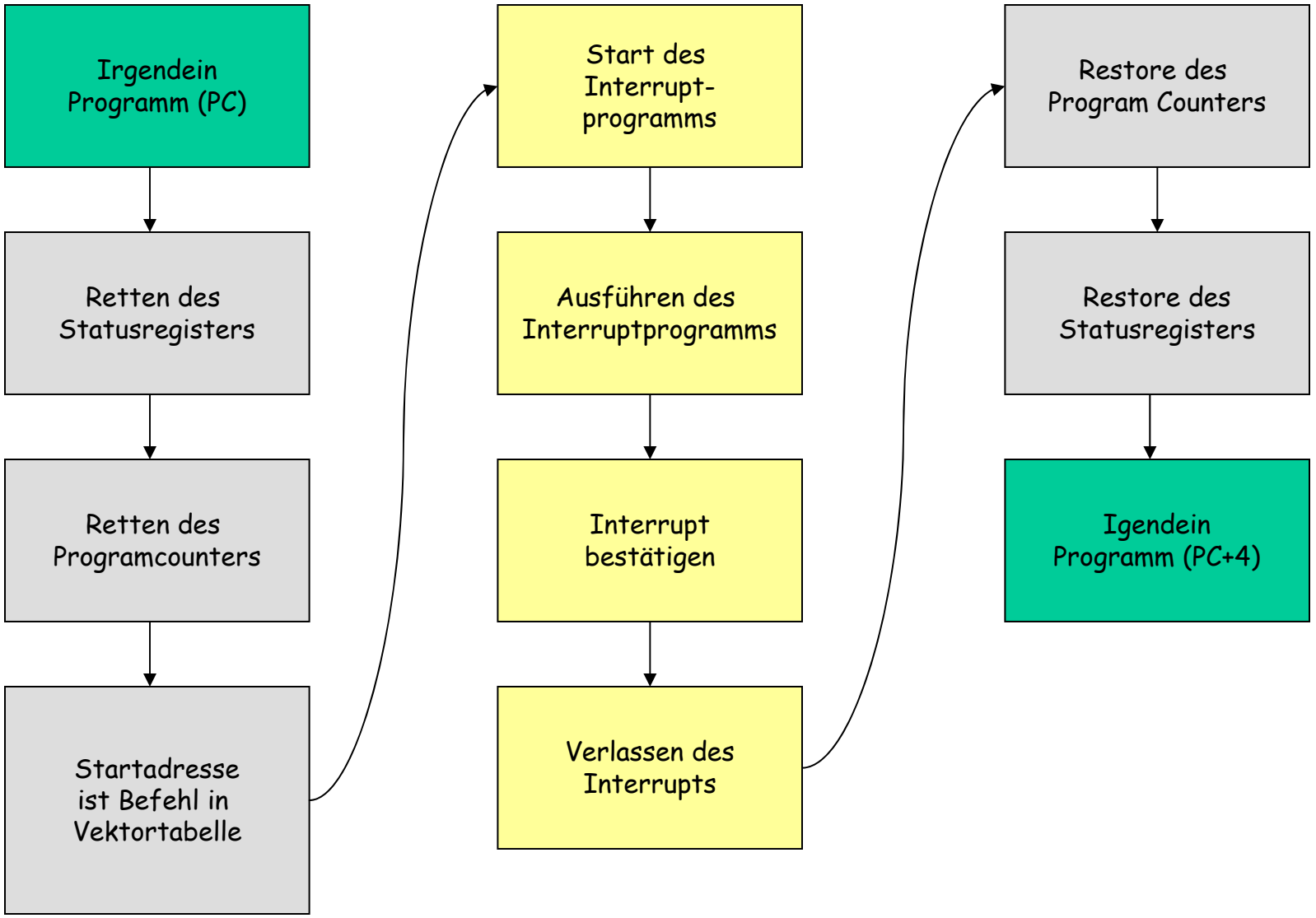
Interrupt Handling – Polling mit Priorität


```
// Pollingbeispiel (prioritätsgesteuert)

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

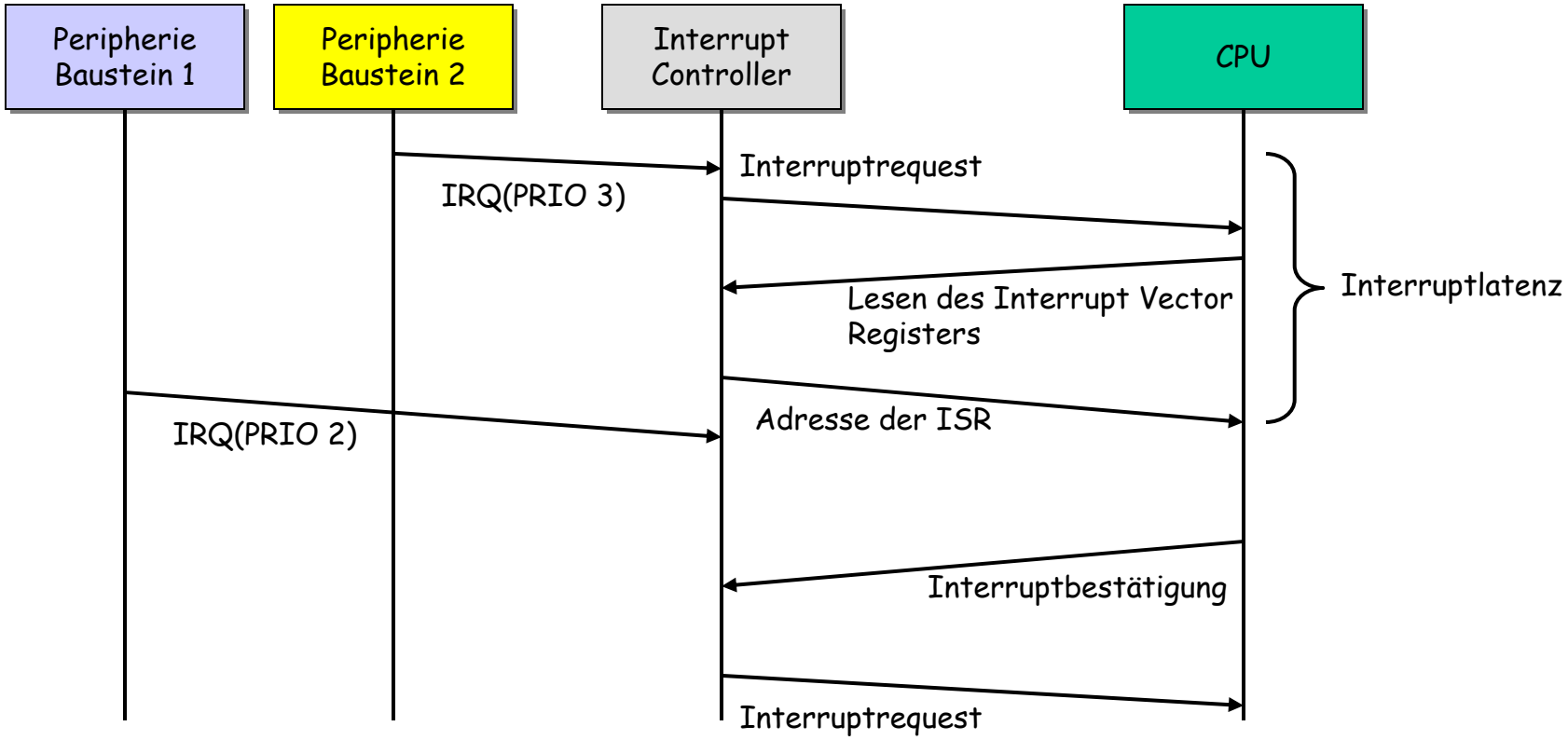
Interrupt Handling – Interruptanforderung



Interrupt Handling - Ablauf

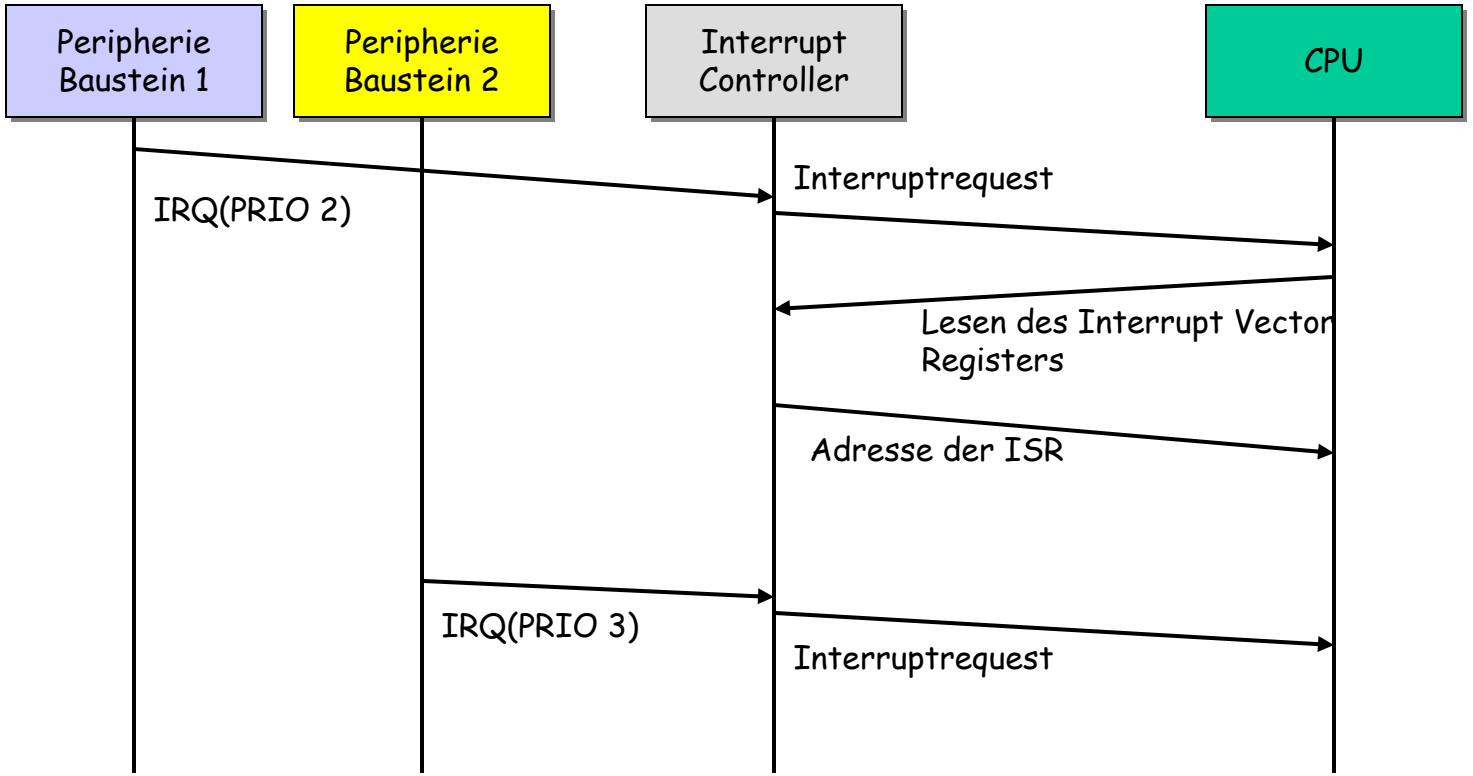


Interrupt Handling – Priorität I



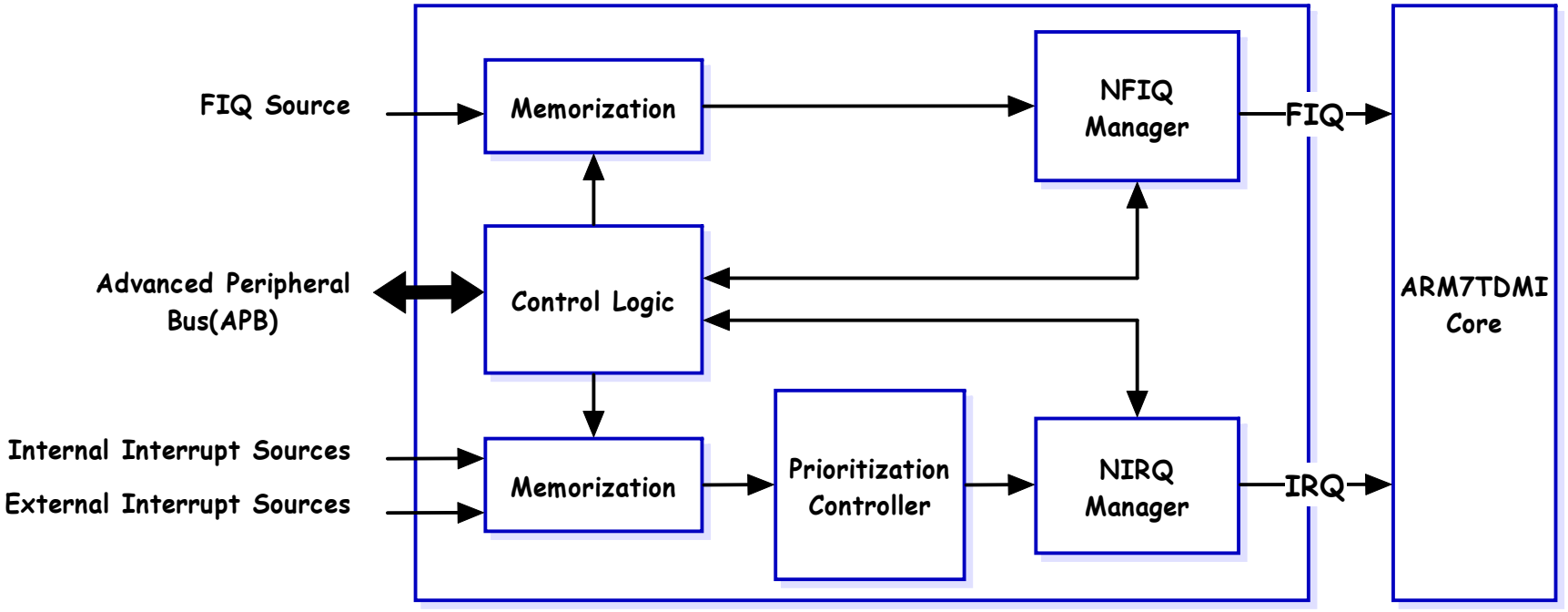
Interruptacknowledge = Lesen des Interrupt Vector Registers
Interruptbestätigung = Schreiben des End Of Interrupt Command Registers

Interrupt Handling – Priorität II



Interrupt Handling - AIC

Advanced Interrupt Controller (AIC)



Interrupt Handling - 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	-
0x134	Spurious Vector Register	AIC_SPU	Read/Write	0

Basisadresse:
0xFFFFF000

Interrupt Handling - 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

Interrupt Handling – Quellen an/aus

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

Interrupt Handling – Triggerarten

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

Interrupt Handling – Code (1)

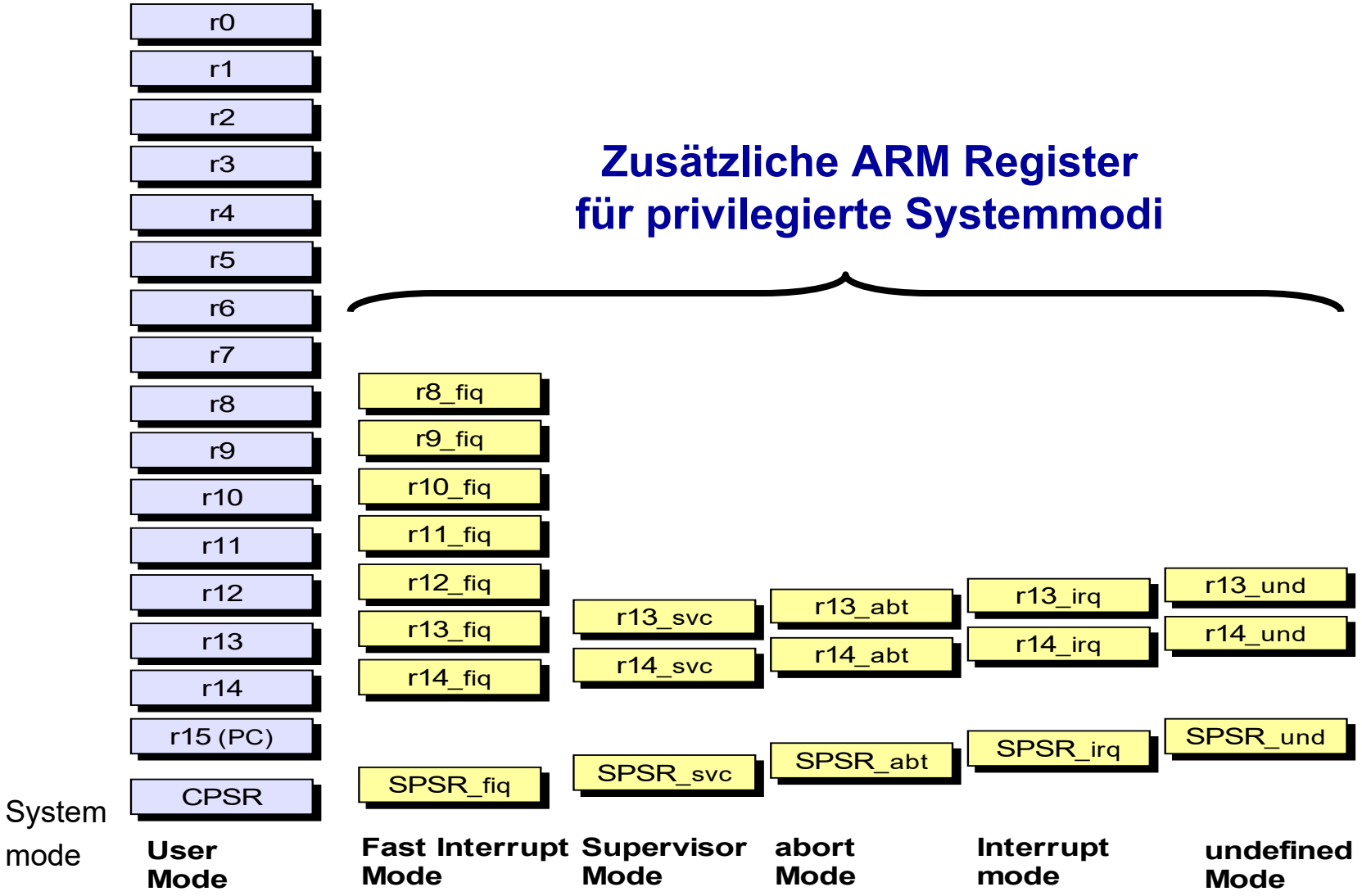

```
int main(void) {
    StructAIC* aicbase = AIC_BASE; // Basisadresse AIC
    aicbase->AIC_IDCR = 1 << 14;  // Interrupt PIOB ausschalten
    aicbase->AIC_ICCR = 1 << 14;  // Interrupt PIOB löschen
    aicbase->AIC_SVR[PIOB_ID] = (unsigned int)irq_handler;
    aicbase->AIC_SMR[PIOB_ID] = 0x7; // Priorität
    aicbase->AIC_IECR = 1 << 14;//  Interrupt PIOB einschalten
    piobaseB->PIO_IER = ...;     //  Interrupt innerhalb PIOB
                                     erlauben
    while( 1 );                  // Endlosschleife

    aicbase->AIC_IDCR = 1 << 14; // Interrupt PIOB ausschalten
    aicbase->AIC_ICCR = 1 << 14; // Interrupt PIOB löschen
    return (0); }
```

Interrupt Handling – Code (2)


```
// Compiler mitteilen, dass es sich um eine Interrupt-
// Service Routine handelt
void irq_handler (void) __attribute__ ((interrupt));

void irq_handler ( void )
{
    StructPIO* piobaseB = PIOB_BASE; // Basisadr. PIO B
    StructAIC* aicbase = AIC_BASE;   // Basisadresse AIC

    //   …              // Inhalt der Interrupt-Service Routine

  // InterruptStatusRegister lesen und in
   // End of Interrupt Command Register (EOICR) schreiben
    aicbase->AIC_EOICR = piobaseB->PIO_ISR;
    }
```

Interrupt Handling – System-Mode

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK



Interrupt Handling - Sprungtabelle

Sprungtabelle für Ausnahmebehandlung an Adresse 0

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

Speicher signalisiert Abbruch beim Holen eines Datenwortes

Speicher signalisiert Abbruch beim Holen einer Instruktion

z.B. 0x0EEEEEEE

Interrupt Handling – IRQ Vectoring

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Laden des PC mit der Adresse des IRQ-Handlers

(negativer Offset -0xF20 führt in die Memory Map des AIC)

```
AIC_Base:
0xFFFF F000      AIC_SMR0    @ AIC Source Mode Reg. 0
…
0xFFFF F080      AIC_SVR0    @ AIC Source Vector Reg. 0
…
0xFFFF F0FC      AIC_SVR31   @ AIC Source Vector Reg. 31
…
0xFFFF F100      AIC_IVR     @ AIC Interrupt Vector Reg.
…
0xFFFF FFFF      @ Ende des Adressraums
VectorTable:
0x0000 0000      ldr         pc, [pc, #0x18]          @ SoftReset
…
0x0000 0018      ldr         pc, [pc,#-0xF20]         @ IRQ : read the AIC
…
0x0000 0020      .word         _SoftReset @ Adresse des Reset-Handlers
```

1. Schritt
AIC kopiert
AIC_SVRxx in
AIC_IVR

2. Schritt
CPU lädt PC
mit Inhalt von
AIC_IVR

Interrupt Handling - Exceptions

Arten von Ausnahmezuständen (Exceptions):

- Ereignisse, die während der Ausführung eines Programms auftreten können
- Externe Ereignisse stehen in keinem Zusammenhang zum Befehlsfluss, Beispiele: IRQ und FIQ als Unterbrechungen (Interrupts) und Reset
- Exceptions, die direkt aus der Ausführung eines Befehls resultieren Beispiele: Software-Interrupts, undefinierte Befehle und „Prefetch Aborts" (Befehlsabbrüche wegen Speicherfehlern)
- Exceptions, die als Nebeneffekt eines Befehls erzeugt werden Beispiele: „Data Aborts" (Abbrüche wegen Speicherfehlern beim Lesen und Schreiben von Daten)

Interrupt Handling – Eintritt

Eintritt in eine Exception (exception entry):

- Wenn eine Exception eintritt, versucht der ARM-Prozessor, den aktuellen Befehl möglichst abzuschließen.
- Reset-Exceptions beenden die Bearbeitung sofort.
- Der Prozessor wechselt in den Systemmodus der jeweiligen Exception.
- Die Adresse des nächsten auszuführenden Befehls wird in r14 des neuen Modus gespeichert.
- Der alte Wert des CPSR wird im SPSR des neuen Modus gespeichert.
- IRQs werden durch Setzen von Bit 7 des CPSR deaktiviert. Wenn es sich um ein FIQ handelt, werden diese durch Setzen von Bit 6 des CPSR deaktiviert.
- Der Prozessor setzt die Ausführung an der für die Exception relevanten und hinterlegten Adresse fort. Die IRQ-Vektoradresse in der Sprungtabelle zeigt auf den Anfang des Vektors, der die für die jeweiligen Quellen hinterlegten Adressen der IRQ-Handler enthält.

Interrupt Handling – Austritt

Austritt aus einer Exception (exception return):

- Alle geänderten Anwenderregister müssen vom Stack des Handlers wiederhergestellt werden.
- Das CPSR muss vom entsprechenden SPSR wiederhergestellt werden.
- Der PC muss so geändert werden, dass er wieder auf die relevante Befehlsadresse im Befehlsstrom des Anwenders verweist.

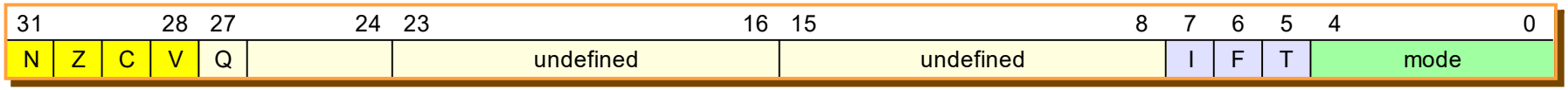
- Die Restauration von CPSR und PC kann nicht unabhängig voneinander erfolgen (Restauration von CPSR -> Zugriff auf r14 geht verloren; Restauration von PC -> Zugriff auf Befehlsablauf geht verloren)

- „Atomarer" Schritt bei Rücksprungadresse in r14 für IRQ und FIQ

```
; Kehre zu dem Befehl zurueck, der vom Exception-Einsprung
; beansprucht wurde d.h. einen Befehl frueher
SUBS pc, r14, #4
```

Interrupt Handling - CPSR

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

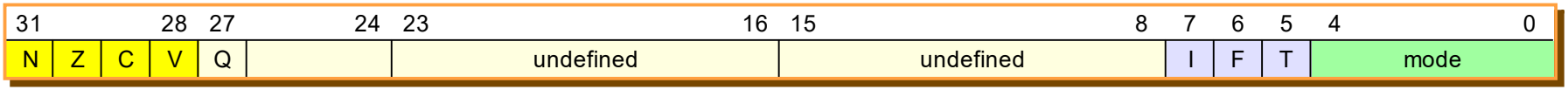


- **Condition Code Flags**
  - N = Negatives ALU Ergebnis
  - Z = Alu Ergebnis ist Null
  - C = Alu Ergebnis erzeugte Carry
  - V = Alu erzeugte Overflow
- Interrupt Disable Bits
  - I = 1, IRQ gesperrt
  - F = 1, FIQ gesperrt

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

Das CPSR ist Teil des Kontextes des laufenden Programms.

Interrupt Handling – CPSR Mode Bits



Mode Bits

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)

Interrupt Handling – Statusregister

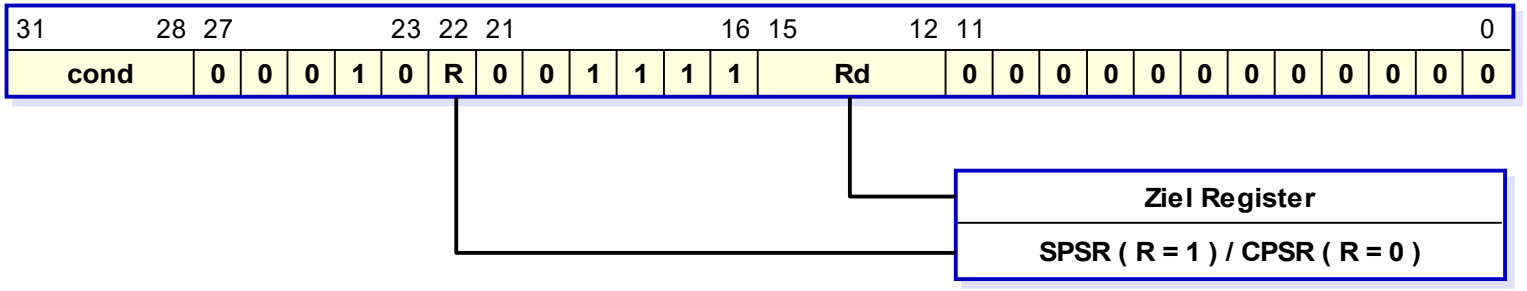
- Das CPSR oder SPSR im jeweiligen Systemmodus kann nicht direkt modifiziert werden.
- Statt dessen muss der Inhalt erst in ein Allzweckregister übertragen werden, bevor ausgewählte Bits dort modifiziert werden können.
- Der modifizierte Wert wird dann in das Statusregister zurück übertragen.
- Hierzu gibt es eigenständige Transfer-Befehle

MSR Move <Register|Immediate> to <CPSR|SPSR>

MRS Move <CPSR|SPSR> to <Register>

Interrupt Handling - MRS

Status Register nach Register Transfer



Syntax:

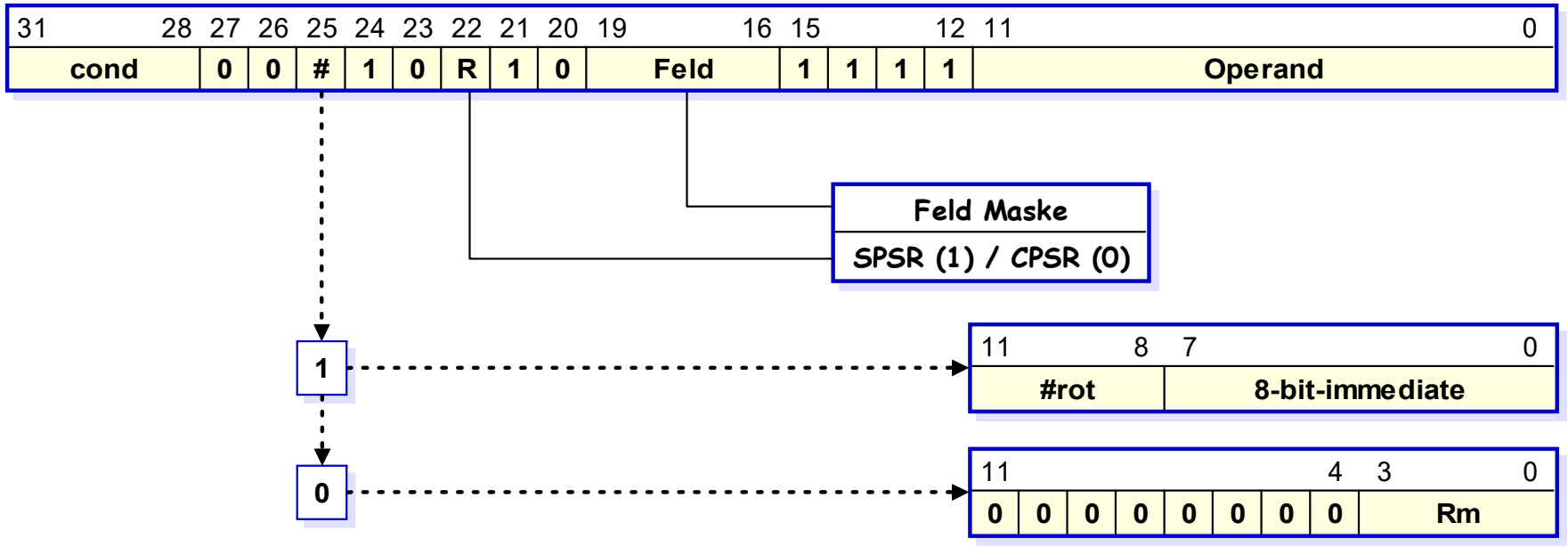
mrs{<cond>} Rd, <psr>

<psr> = CPSR oder SPSR

z.B. Rd := CPSR

Interrupt Handling - MSR

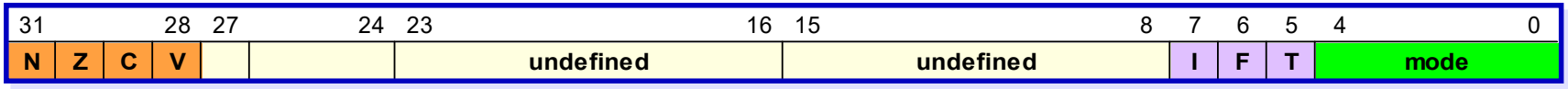
Register nach Status Register Transfer



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

- Beispiele zur MSR und MRS

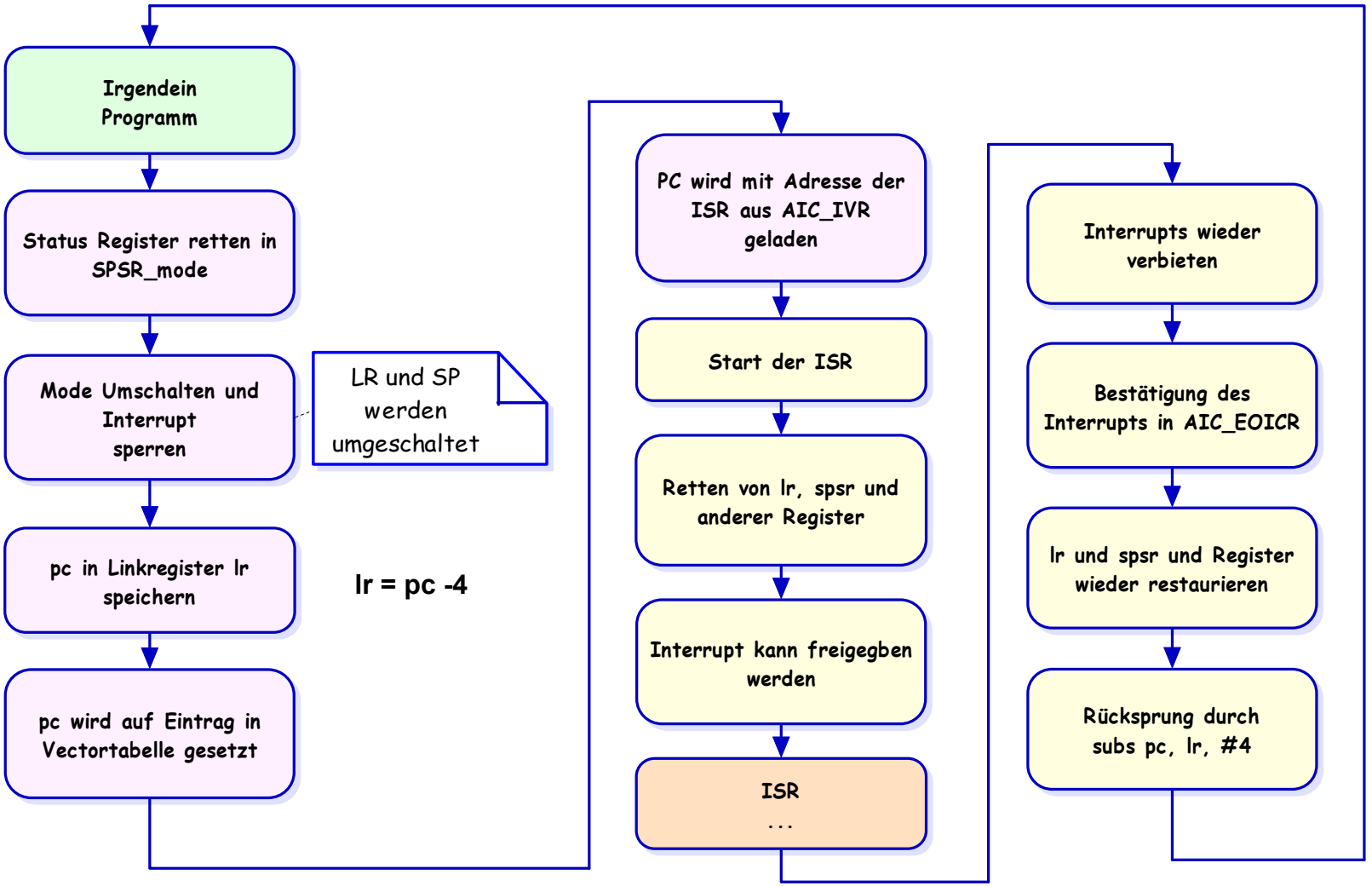
- Setzen der Prozessorflags N, Z, C, V
  - msr CPSR_f, #0xf0000000

- Setzen des Carry Flags
  - mrs r0, CPSR
  - orr r0, r0, #0x20000000
  - msr CPSR_f, r0

- Prozessor in den Interrupt Mode schalten
  - mrs r0, CPSR // aktuellen Status in r0 holen
  - bic r0, r0, #0x1f // unterste 5 Bits löschen
  - orr r0, r0, #0x12 // neuen Mode eintragen
  - msr CPSR_c, r0 // in das CPSR zurückschreiben

- Anwenderprogramme können nur CPSR[31:24] ändern.

Interrupt Handling – Ablauf I



Interrupt Handling – Ablauf II

Wenn eine Ausnahmebehandlung auftritt, wird folgendes ausgeführt:

LR_<exception_mode> = Rücksprungadresse (PC – 4)

SPSR_<exception mode> = CPSR

CPSR[4:0] = exception mode Nummer

Falls <exception_mode> = RESET oder FIQ

    CPSR[6] = 1 ; Disable FIQ Bit setzen

Sonst

    CPSR[7] = 1 ; Disable IRQ Bit setzen

PC = Eintrag in Vektortabelle (in Abh. der Ausnahmebehandlung, z.B. 0x18 für Interrupt)

Interrupt Handling - reentrant

- reentrant = wiedereintrittsfähig

- Eigenschaft einer Funktion, dass sie unterbrechbar ist und wiedereintrittsfähig

- In der Anwendungsprogrammierung müssen z.B. Bibliotheken (Libraries) reentrant sein, wenn Multi-Threading verwendet wird.

- In der ereignisgesteuerten Programmierung bezeichnet man ISRs als reentrant-fähig, wenn diese durch weitere Interrupts unterbrechbar sind.

Interrupt Handling – Reentrant Entry

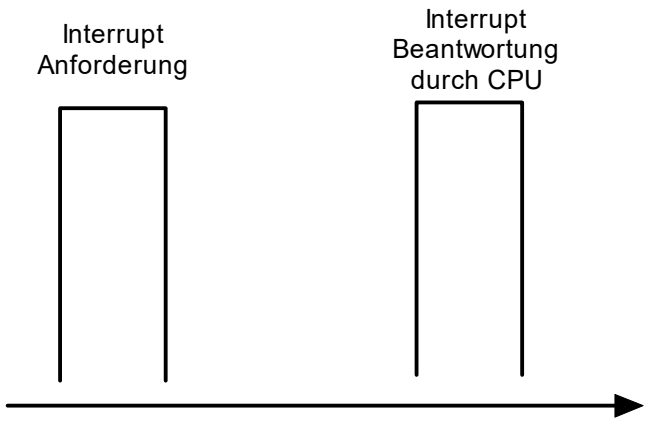

```
@ re-entrant fähiger Interrupt
@ lr korrigieren für Rücksprung
    sub     lr, lr, #4
    stmfd   sp!, {lr}
    mrs     lr, SPSR
    stmfd   sp!, {lr}
@ in System Mode gehen (um später I_BIT setzen zu können) und
@ Interrupt freigeben
    mrs     lr, CPSR
    bic     lr, lr, #I_BIT
    orr     lr, lr, #ARM_MODE_SYS
    msr     CPSR, lr                (lr = 0x1F)
@ Register auf USER/System Stack retten
    stmfd   sp!, {r0-r3, benutzte Register, r12, lr}
```

Interrupt Handling – Reentrant Exit


```
@ Register restaurieren
    ldmfd       sp!, {r0-r3, benutzte Register, r12, lr}
@ CPSR modifizieren um Interrupt zu sperren und um in
@ Interrupt Mode zurückzugelangen
    mrs         lr, CPSR
    bic         lr, lr, #ARM_MODE_SYS
    orr         lr, lr, #(I_BIT | ARM_MODE_IRQ)
    msr         CPSR, lr                    (lr = 0x92)
@ Ende des Interrupts an AIC
    ldr         lr, =AIC_BASE
    str         lr, [lr, #AIC_EOICR]
@ Restaurieren der Register LR und SPSR_irq vom IRQ_STACK
    ldmfd       sp!, {lr}
    msr         SPSR, lr
@ Rücksprung über LR_irq mit Rückschreiben des Statusregisters
    ldmfd       sp!, {pc}^
```

Spurious Interrupt

- spurious engl., störend, unecht
- Der Spurious Interrupt tritt auf, wenn eine Komponente einen Interrupt im pegelsensitiven Mode anfordert und die Anforderung zur Zeit des Lesens des Vector Registers nicht mehr existiert
- Der Spurious Interrupt tritt auf, wenn ein Interrupt ausgelöst wird und sofort danach per Software (AIC_ICCR) gelöscht wird. Dies kann aufgrund des Pipelining geschehen.
- Die Routine für den Spurious Interrupt ist eine normale Interruptroutine, die den Interruptstack durch Beschreiben des EOICR Registers zurücksetzen muß.



Spurious Interrupt

Auslösung per Software (Optimierungsstufe 0 - kein Spurious Interrupt)

```
                                    if(piobaseB->PIO_PDSR & Taste3)
-   0x2000338   <main+164>:     ldr     r3, [r11, #-20]
-   0x200033c   <main+168>:     ldr     r3, [r3, #60]
-   0x2000340   <main+172>:     and     r3, r3, #524288     ; 0x80000
-   0x2000344   <main+176>:     cmp     r3, #0      ; 0x0
-   0x2000348   <main+180>:     beq     0x2000338 <main+164>
    85                          {
    86                                  aicbase->AIC_ISCR = 0x4000;
-   0x200034c   <main+184>:     ldr     r2, [r11, #-16]
-   0x2000350   <main+188>:     mov     r3, #16384          ; 0x4000
-   0x2000354   <main+192>:     str     r3, [r2, #300]
    87
    88                                  aicbase->AIC_ICCR = 0x4000;
-   0x2000358   <main+196>:     ldr     r2, [r11, #-16]
-   0x200035c   <main+200>:     mov     r3, #16384          ; 0x4000
-   0x2000360   <main+204>:     str     r3, [r2, #296]
```

Spurious Interrupt

Auslösung per Software (Optimierungsstufe 1 - Spurious Interrupt !!!)

```
                    if(piobaseB->PIO_PDSR & Taste3)
-   0x2000284   <main+72>:      mvn     r3, #65280      ; 0xff00
-   0x2000288   <main+76>:      ldr     r3, [r3, #-195]
-   0x200028c   <main+80>:      tst     r3, #524288     ; 0x80000
    85                      {
    86                          aicbase->AIC_ISCR = 0x4000;
-   0x2000290   <main+84>:      mvnne   r2, #3840; 0xf00
-   0x2000294   <main+88>:      movne   r3, #16384      ; 0x4000
-   0x2000298   <main+92>:      strne   r3, [r2, #45]
    87
    88                          aicbase->AIC_ICCR = 0x4000;
-   0x200029c   <main+96>:      strne   r3, [r2, #41]
-   0x20002a0   <main+100>:     b       0x2000284 <main+72>
```

Spurious Interrupt

Spurious Interrupt Service Routine:

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

Dabei kennzeichnet ^ am Ende der Parameter, dass der Befehl ldmfd auch das CPSR restauriert. Dies ist nur in priviligierten Modi möglich.

Konstanten:

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

Rücksprung aus Ausnahmebehandlungen (Exceptions)

Vektor	Exception	Mode	Rücksprung	Priorität
0x1C	FIQ	FIQ 10001	subs pc, lr, #4	3
0x18	IRQ	IRQ 10010	subs pc, lr, #4	4
0x14	(Reserviert)			
0x10	Data Abort	ABORT 10111	subs pc, lr, #8	2
0x0C	Prefetch Abort	ABORT 10111	subs pc, lr, #4	5
0x08	Software Interrupt	SVC 10011	movs pc, lr	6
0x04	Undefined Instruction	UND 11011	movs pc, lr	6
0x00	Reset	SVC 10011		1
		SYSTEM 11111		

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 sperre, wenn er gesperrt ist oder ihn freigebe wenn er freigegeben ist?
- Sind Interrupts freigegeben oder gesperrt, wenn das System startet?
- Kann ich Interruptroutinen in C schreiben?

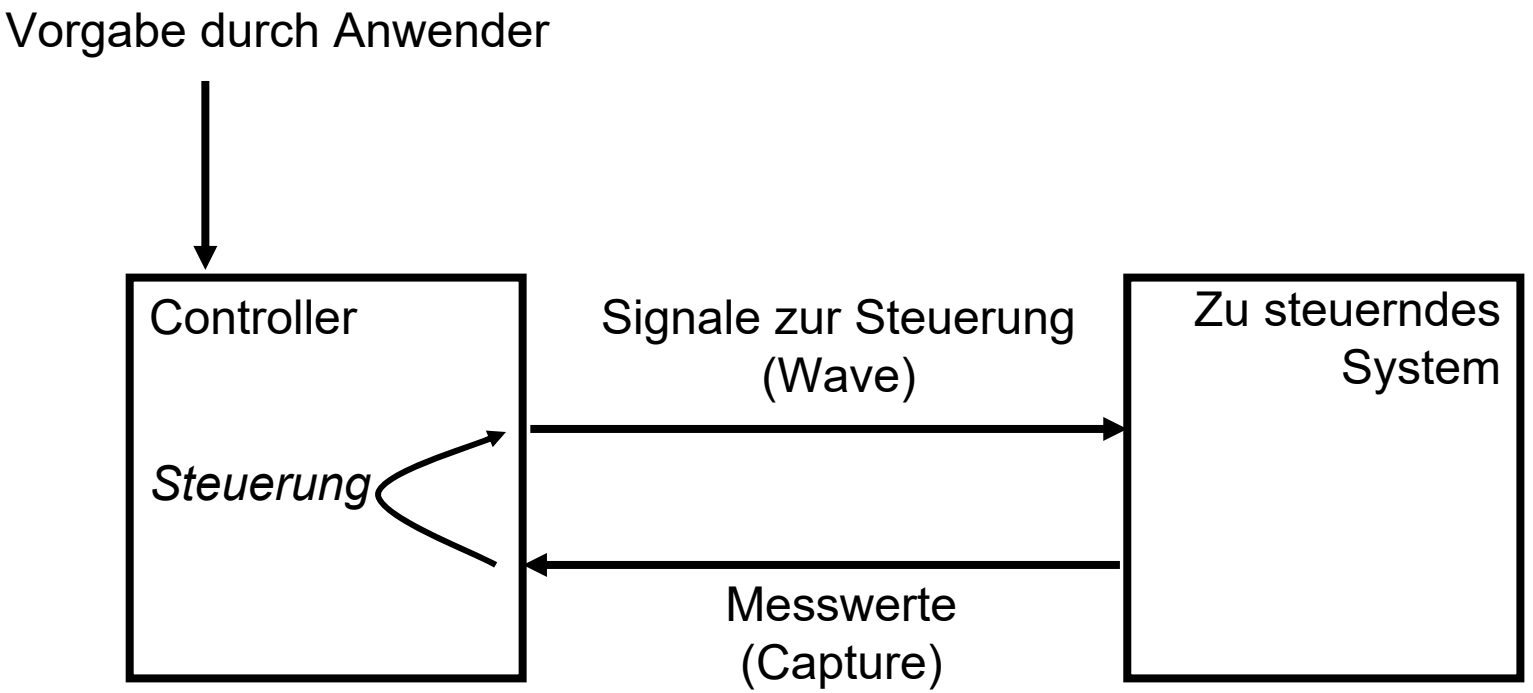
Gliederung

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

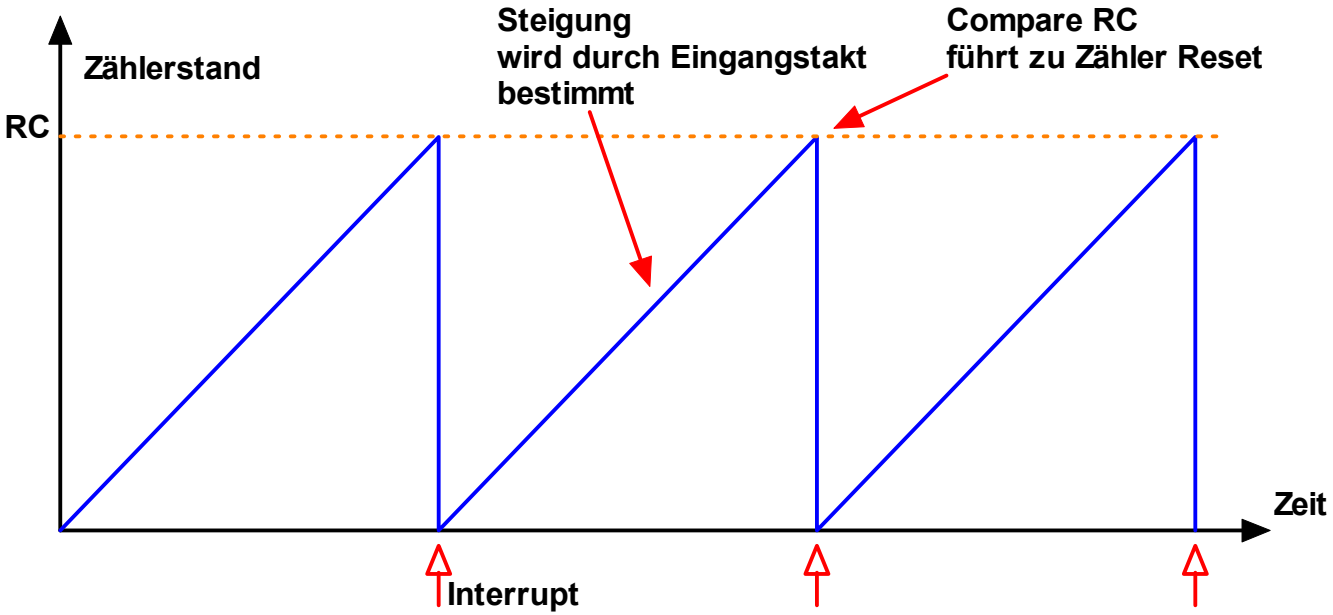
Timer - Anwendungen

- 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

Timer – ein einfacher Regelkreis

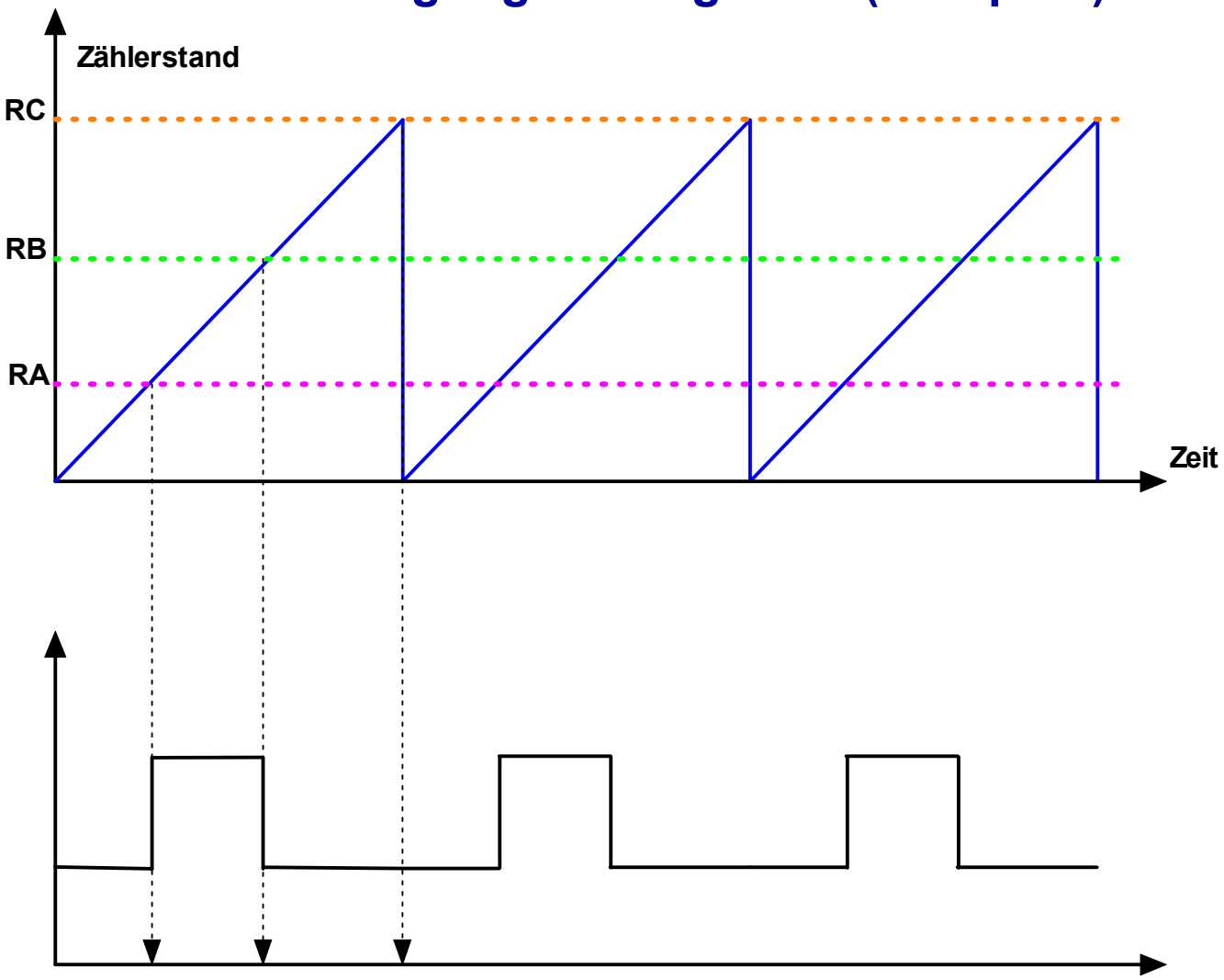


Timer – periodische Interrupts



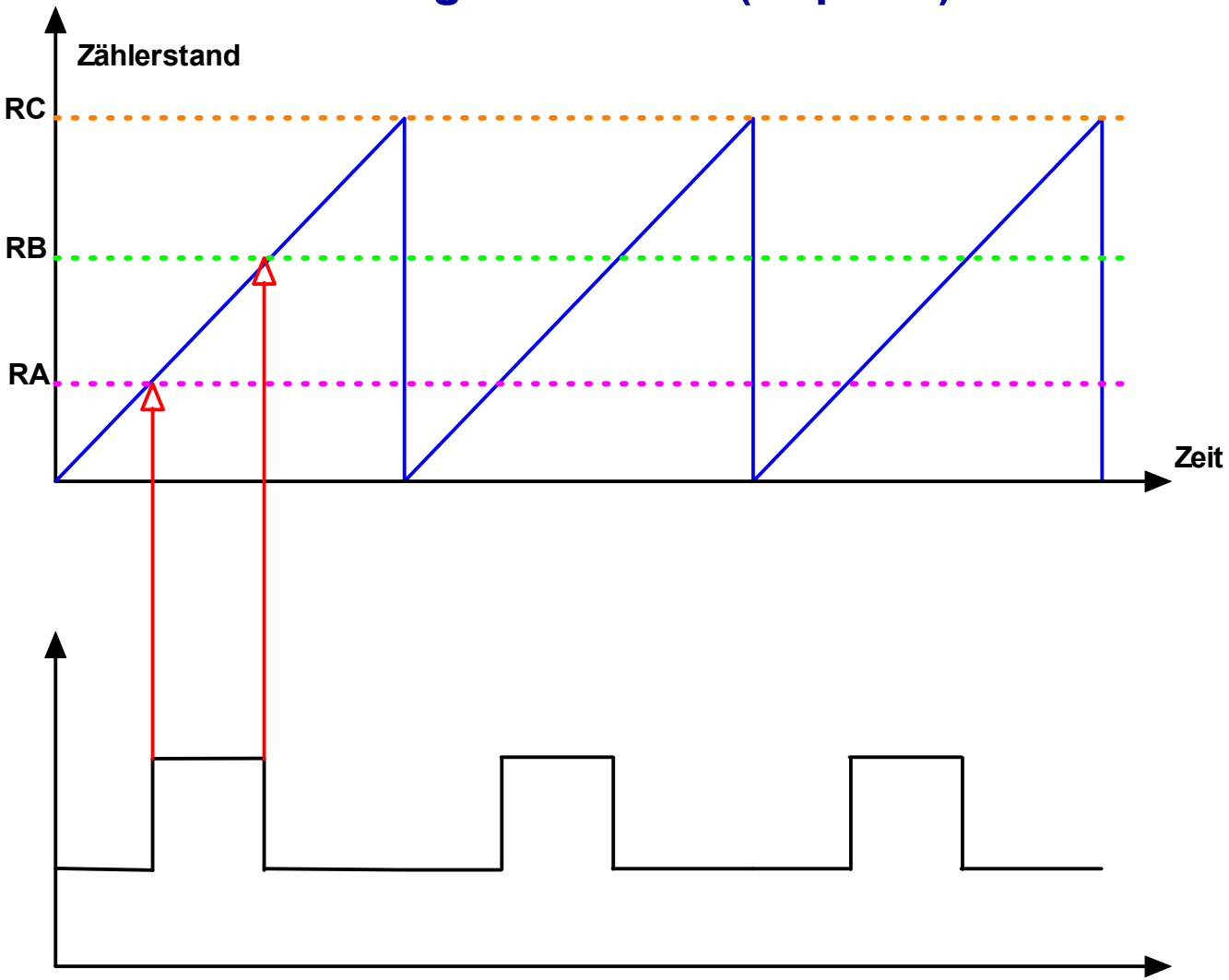
Timer - Waveform

Erzeugung von Signalen (Compare)



Timer - Capture

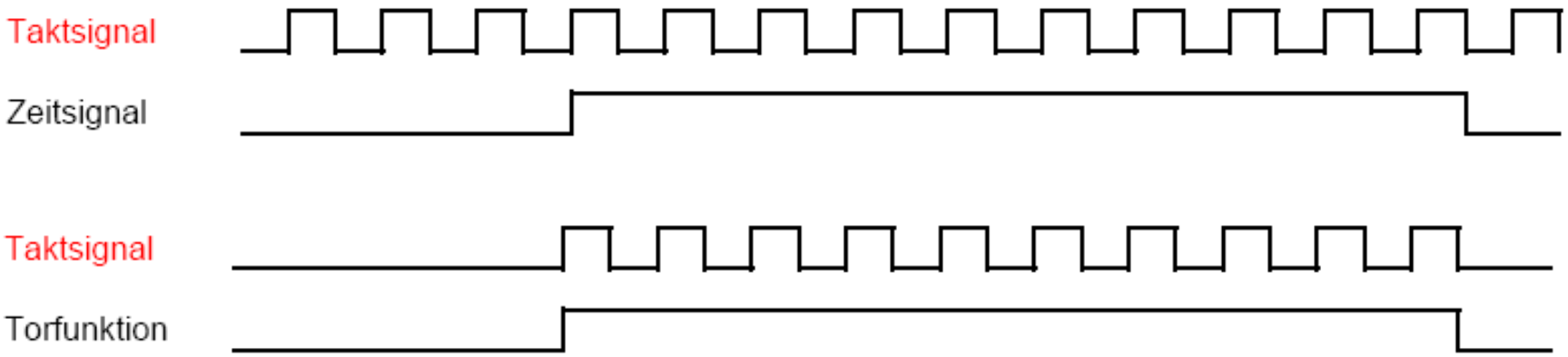
Messung von Zeiten (Capture)



Timer - Messungen

Messung von Zeiten:

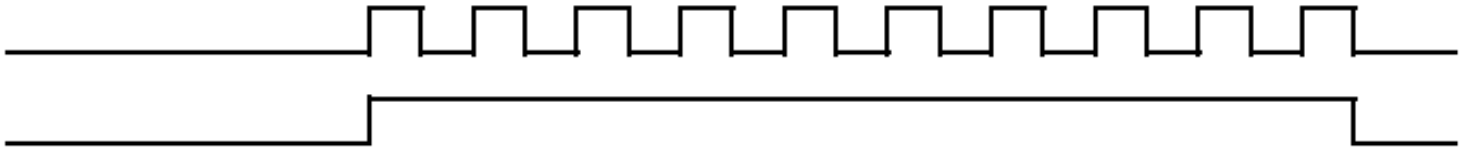
- Die zu messende Zeit wird mit einem periodischen Signal mit sehr viel schnellerem Takt (Clock) verglichen
- Das Clocksignal wird gezählt
- Das Zeitsignal kann als „Torfunktion" für das Taktsignal dienen
- Die Flanken des Zeitsignals können zur Speicherung des aktuellen Zählerstands benutzt werden



Timer – Frequenz und Periodendauer

Messung von Frequenzen

- Frequenz ist invers zur Periodendauer
- Die Messung von Frequenzen erfolgt indem man die Taktflanken des Zählerbausteins (bekannte Periodendauer) für die unbekannte Periodendauer zählt
- Die Periodendauer des Signals muss sehr viel länger sein als die Taktdauer des Zählerbausteins.



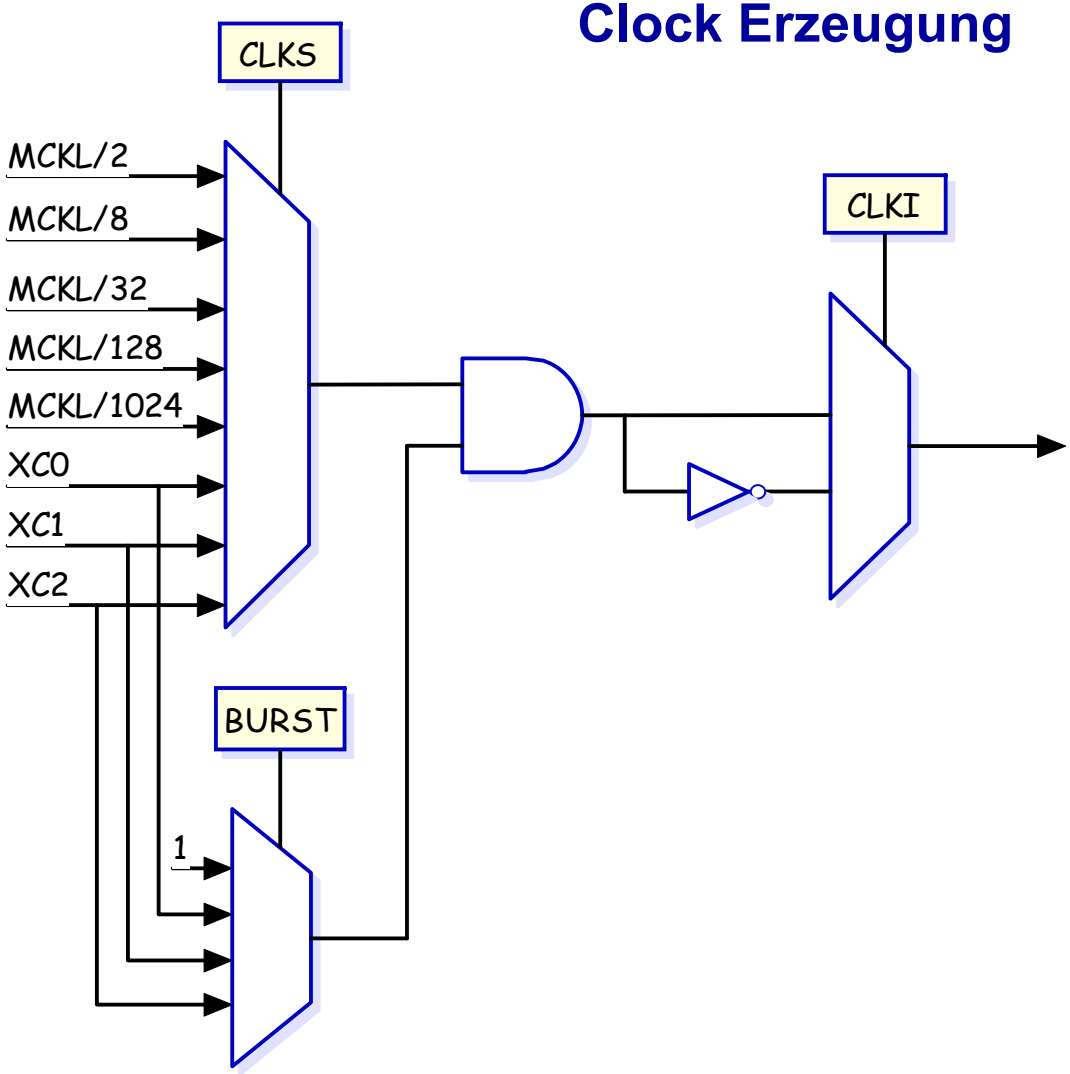
Timer - Signalerzeugung

Erzeugung von analogen Signalen

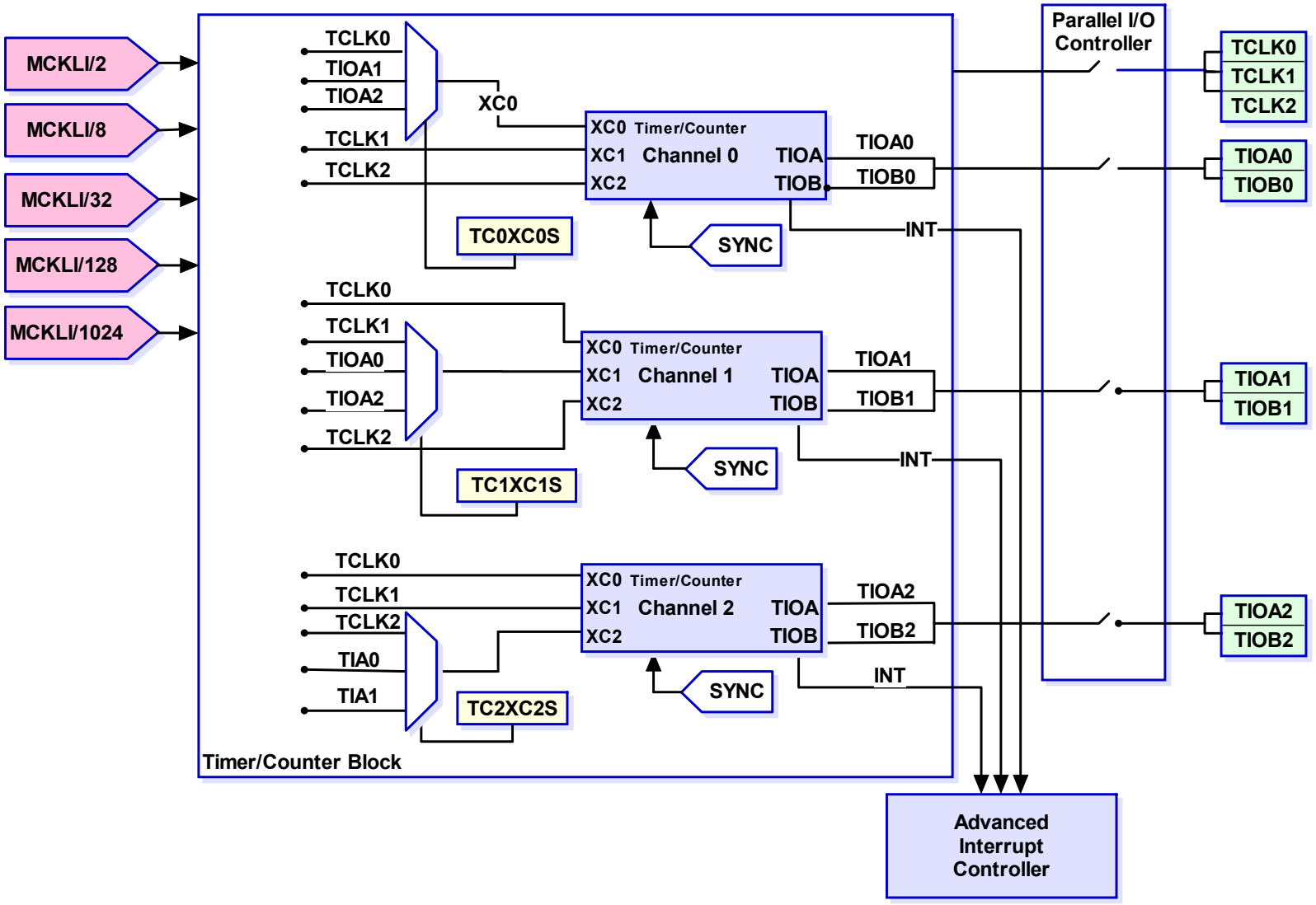
- Timer können auch zur Erzeugung von analogen Signalen benutzt werden
- Dabei wird das Ausgangssignal pulsweitenmoduliert und anschließend durch eine Tiefpassfilterung geglättet
- Beispiel Leuchtstärke LED

Timer - Diagramm

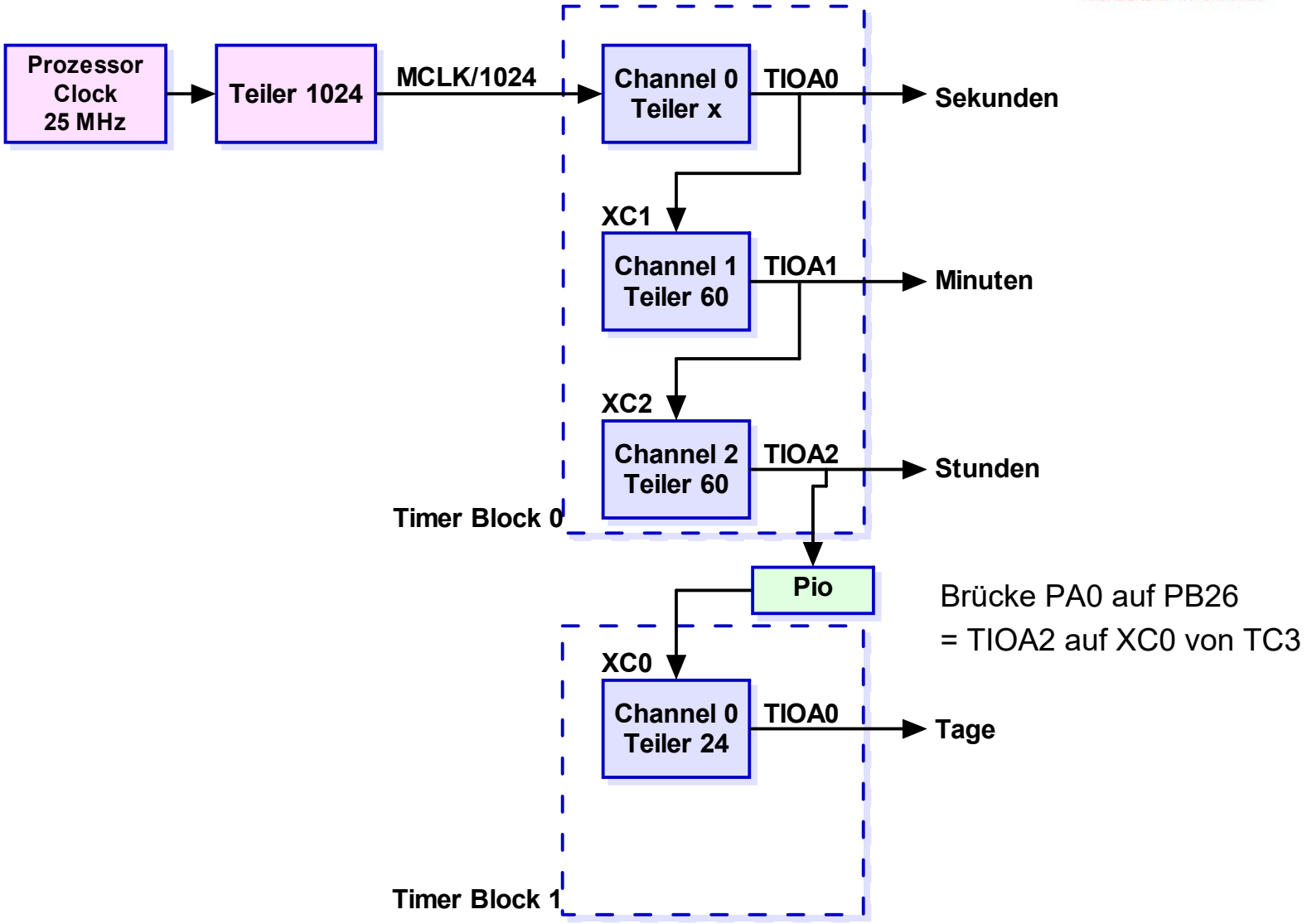
Clock Erzeugung



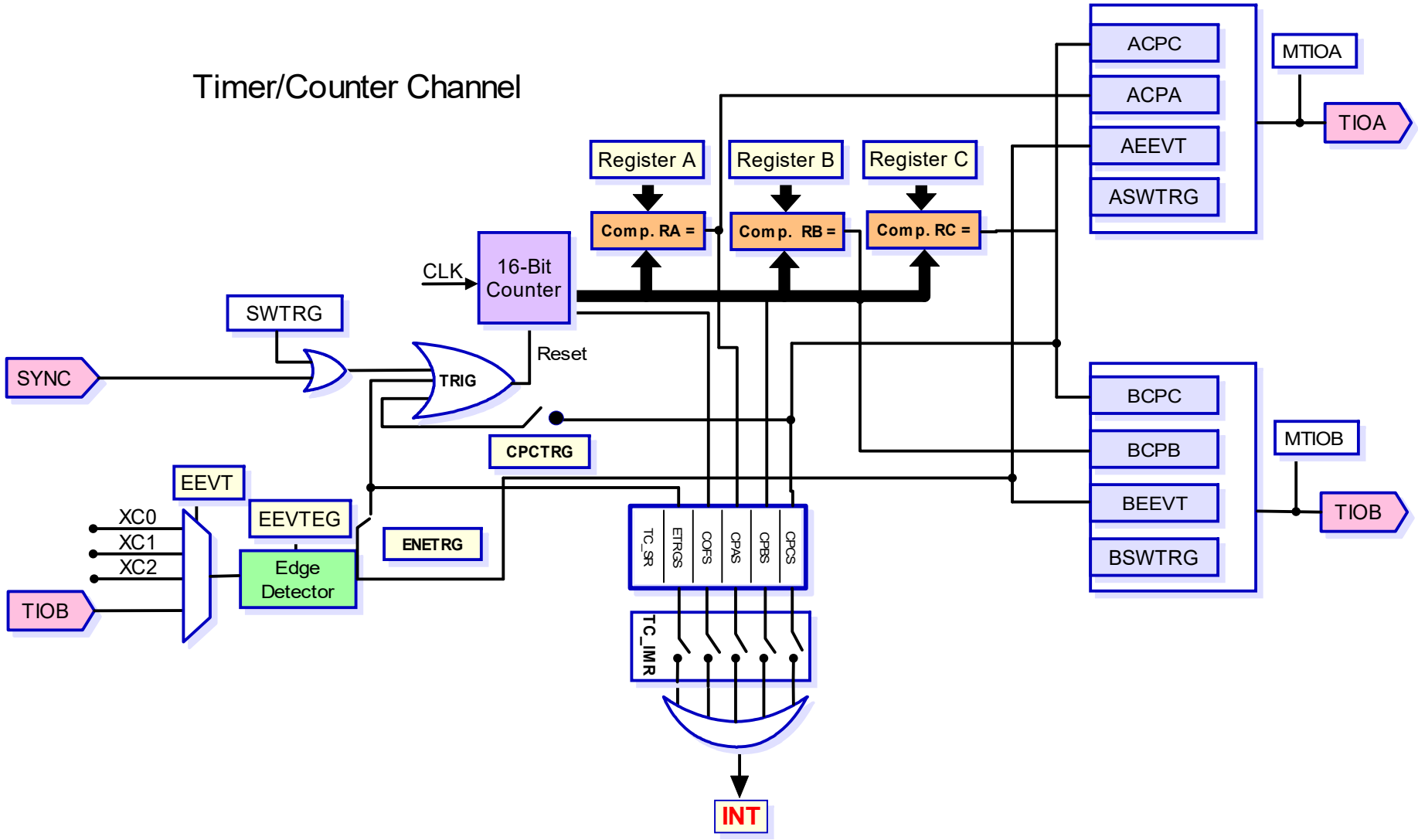
Timer - Counter



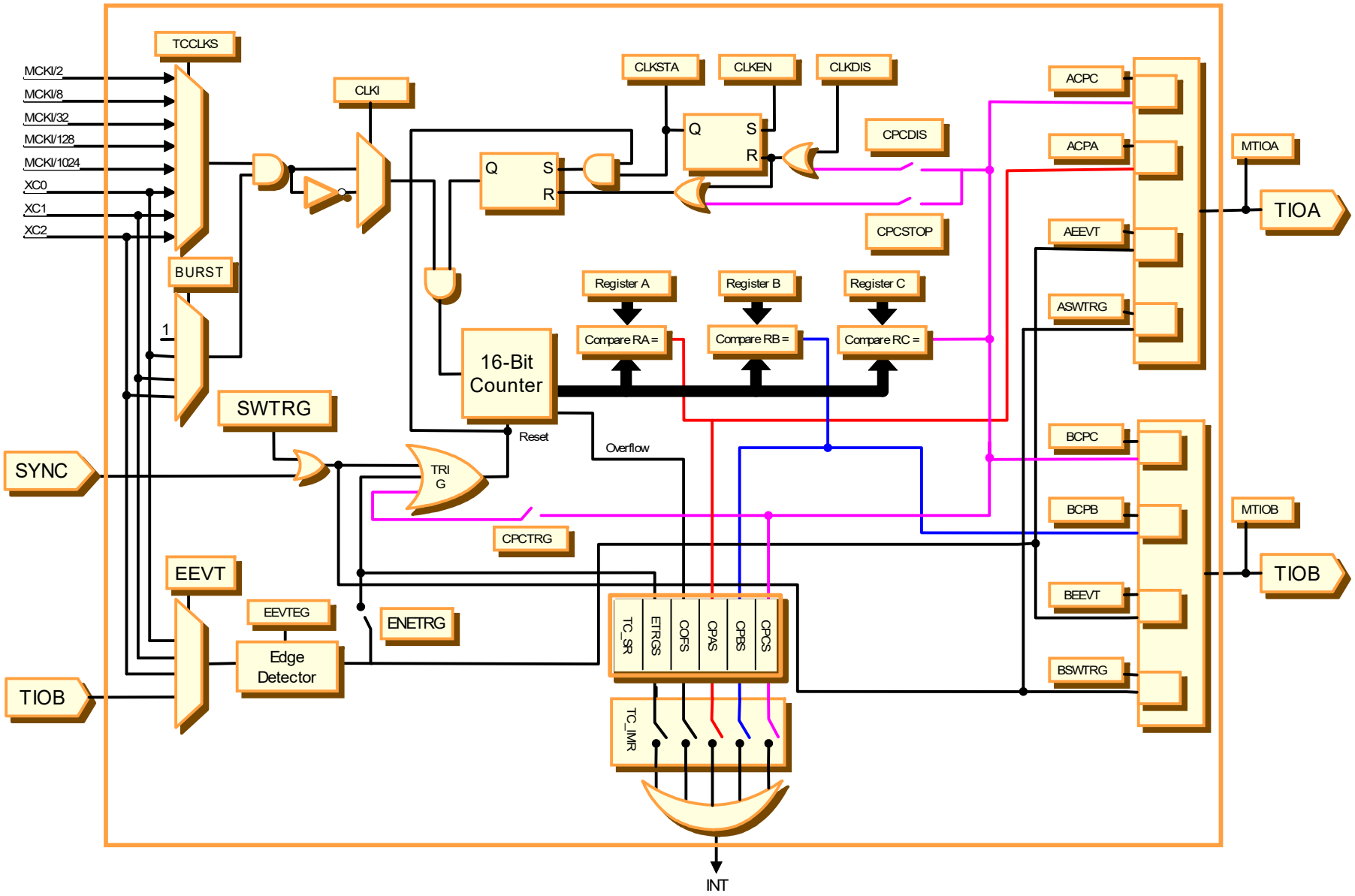
Timer – Beispiel Uhr



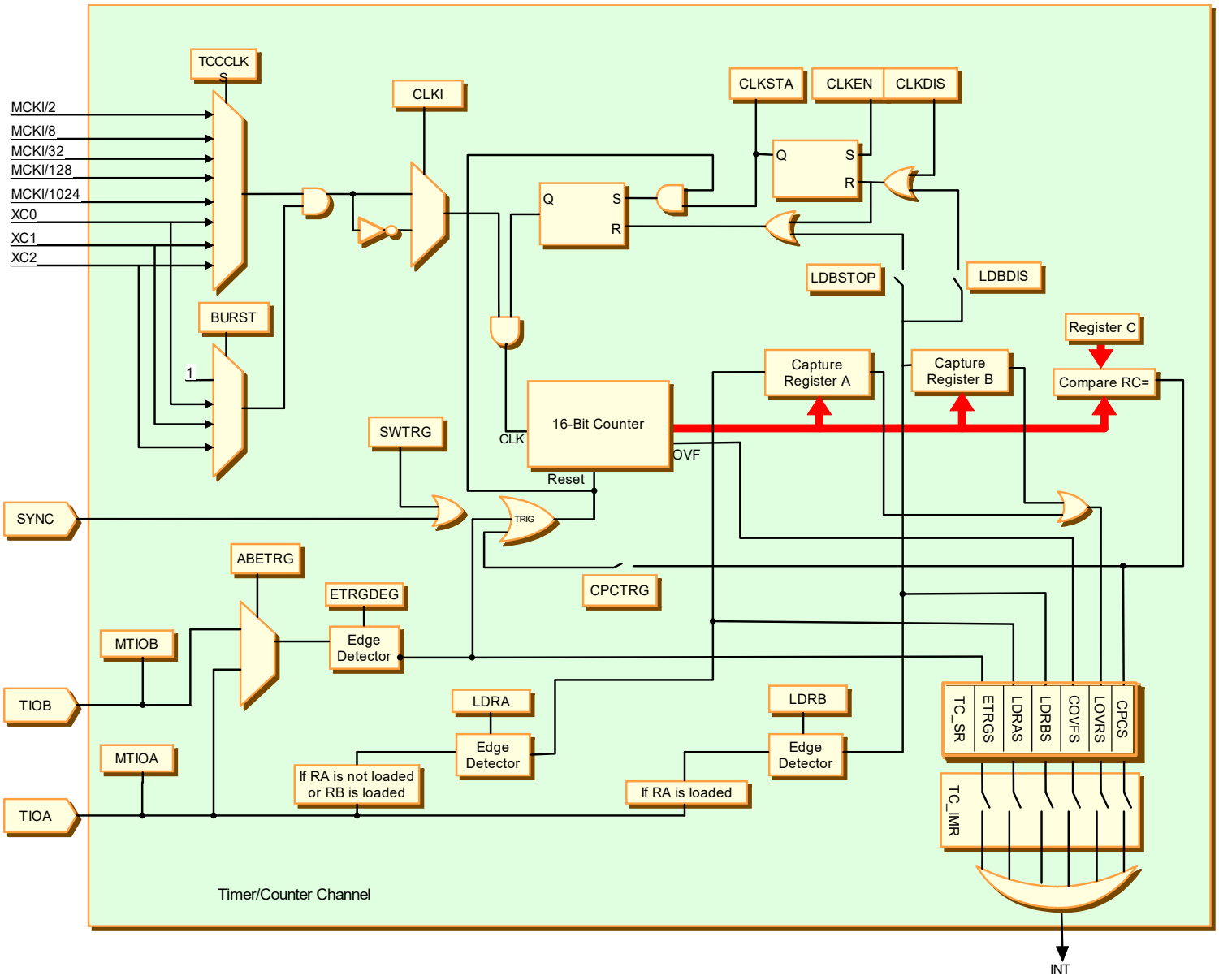
Timer - Compare (Waveform)



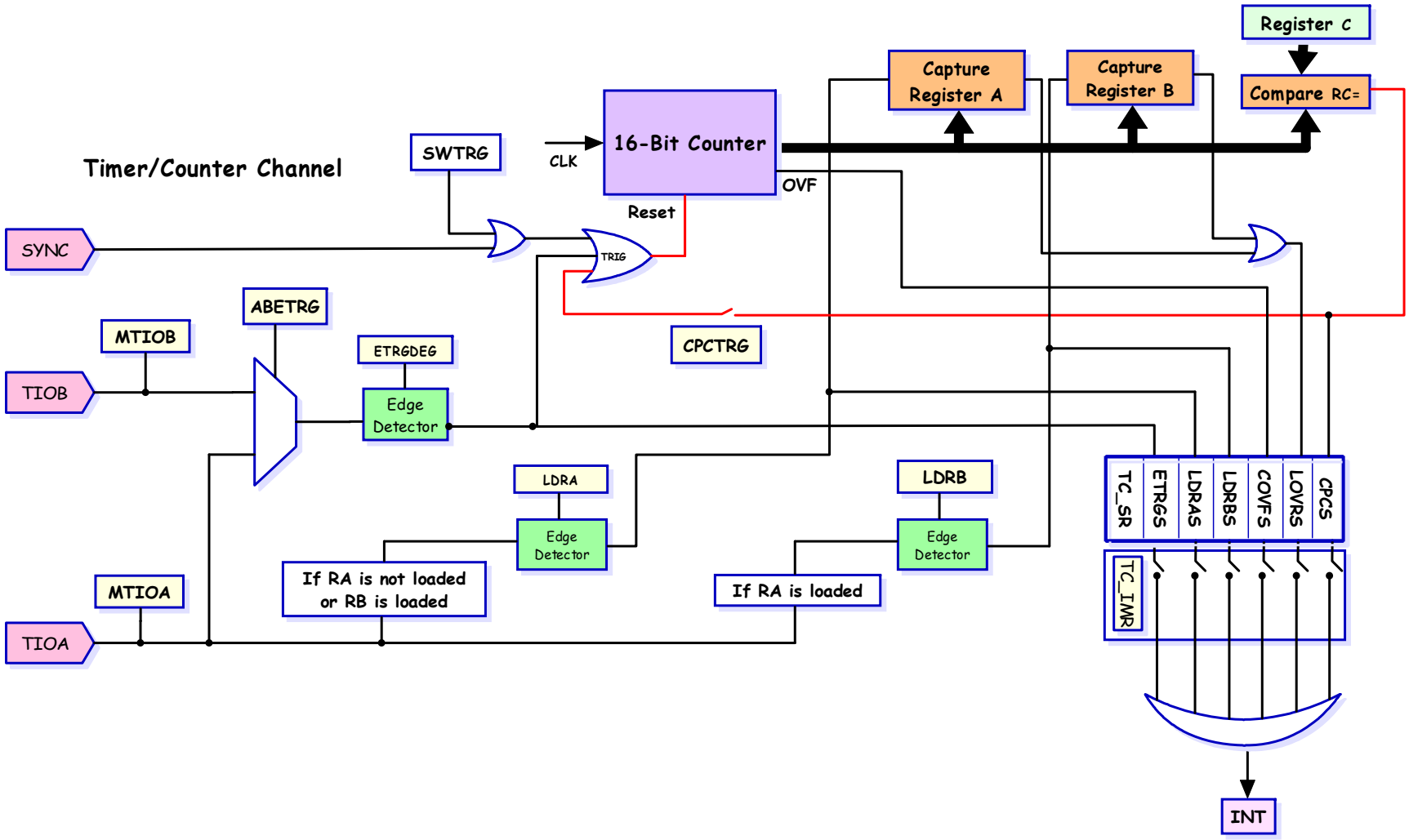
Timer - Compare (Waveform)



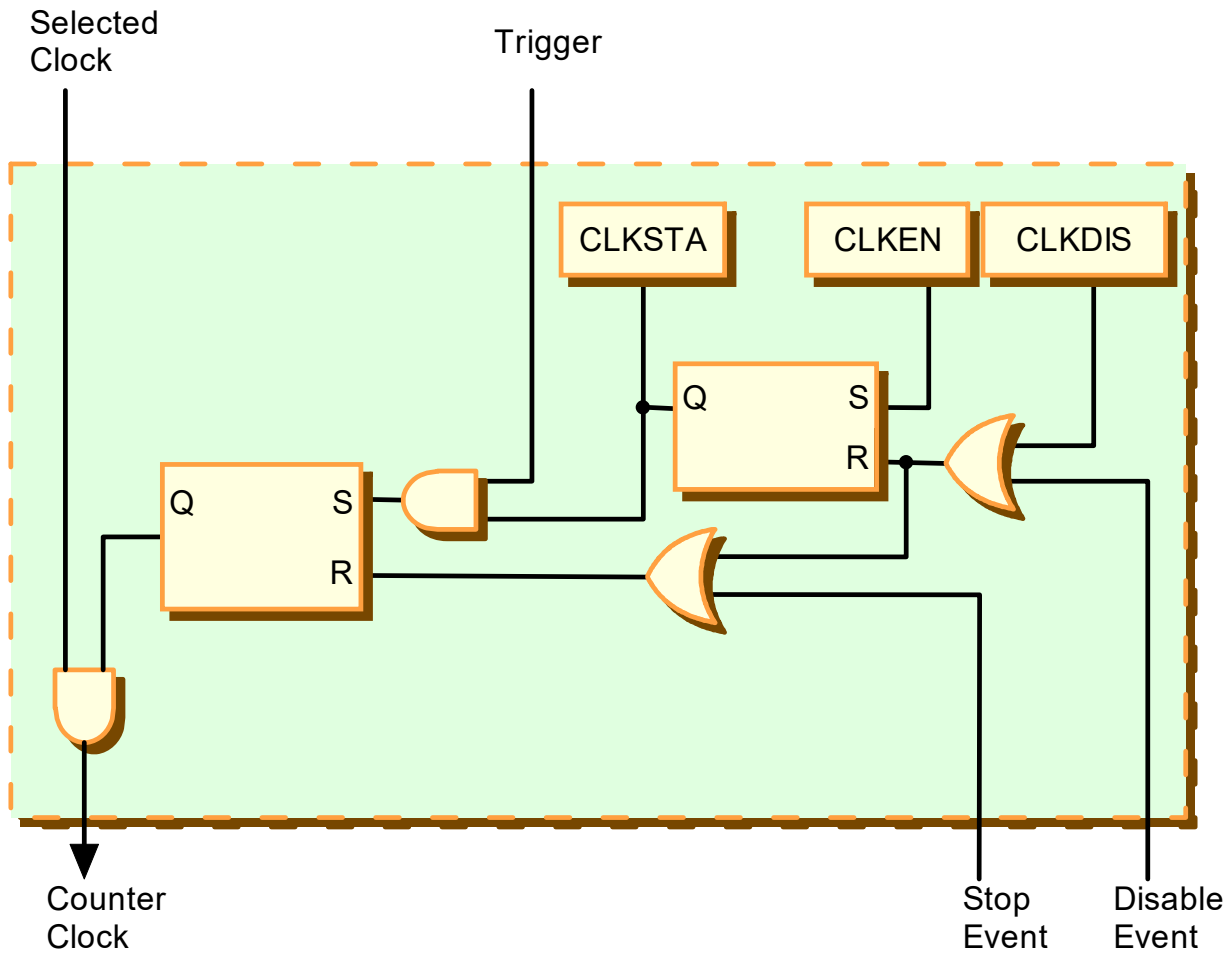
Timer - Capture Mode



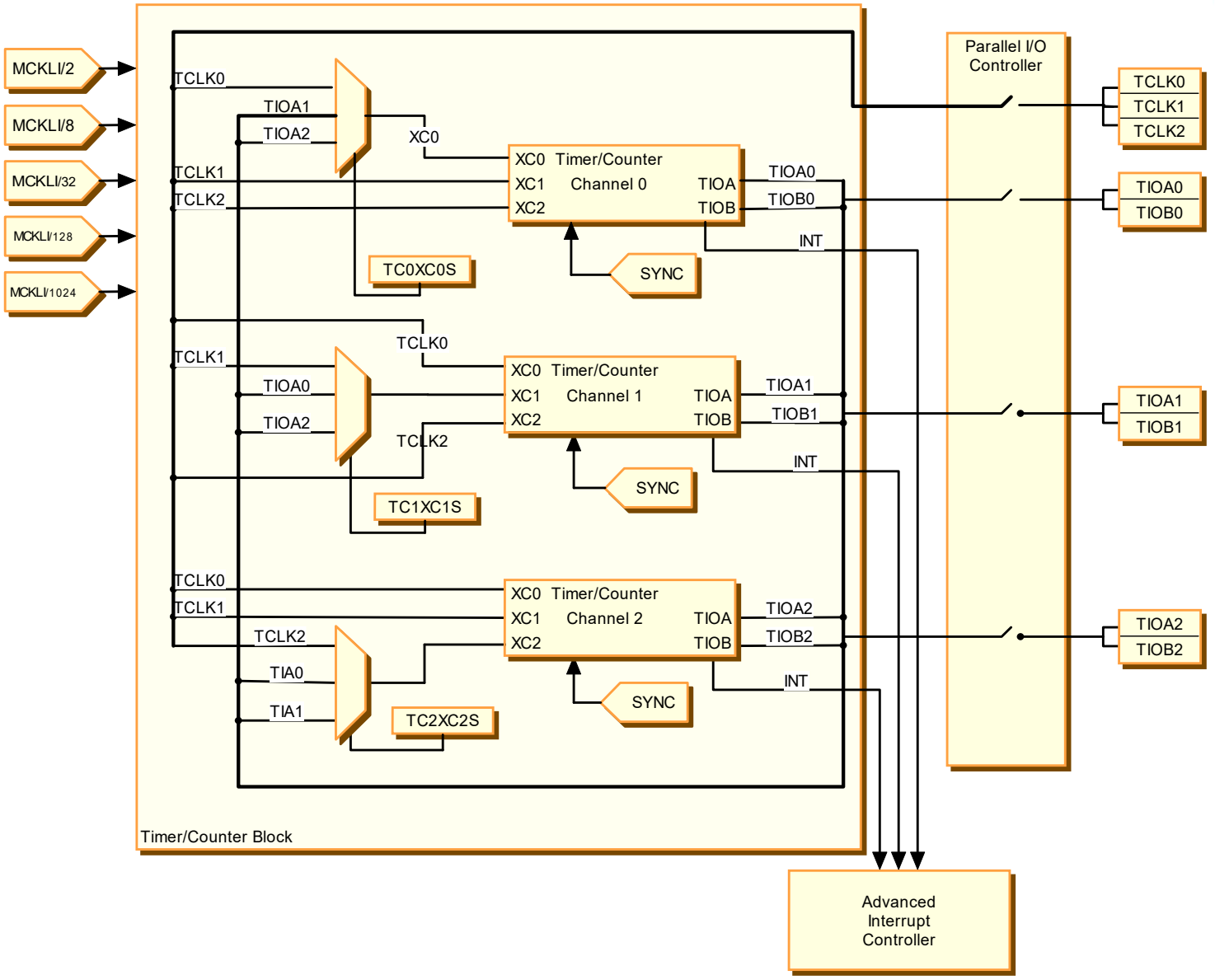
Timer - Capture Mode



Timer


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

Timer – 3 Kanäle pro Block



Timer - Beispielprogramm (1)

Applikationsbeispiel:

Timerbaustein soll zwei pulsweitenmodulierte Signale erzeugen

Weitere Kenngrössen:

Frequenz:	100 Hz
PWM Signal an TIOA	30% High-Pegel
PWM Signal an TIOB	50% High-Pegel
Taktrate Masterclock	25 MHz

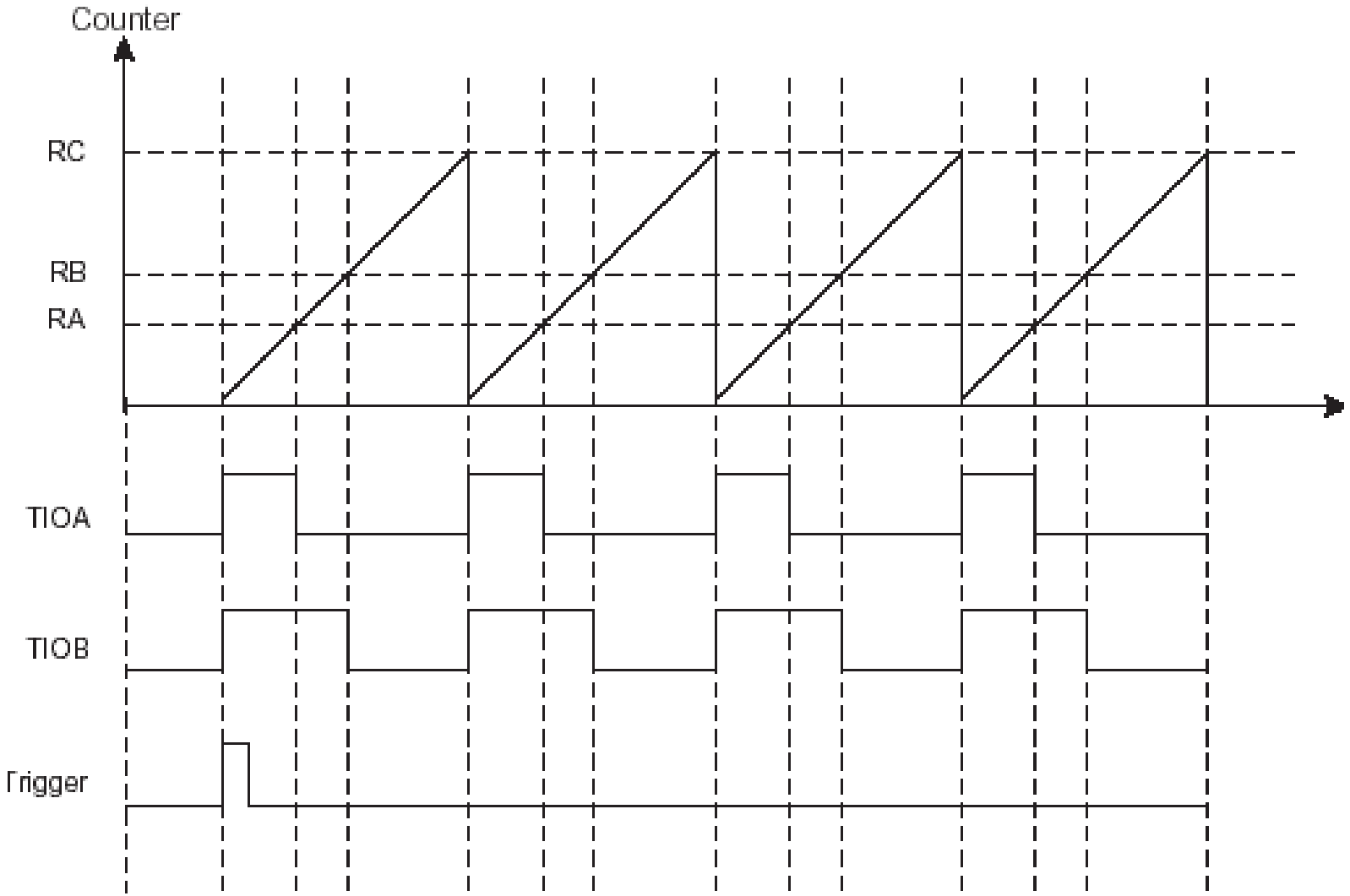
Timer - Beispielprogramm (2)


```
// Function Name : main
// Object: Timer Counter 1 configuration to generate Dual PWM generation
int main ( void )  {
// ... Variablendeklarationen
timerbase->TC_CCR = TC_CLKDIS ; // Disable the Clock Counter
// Define TIOA1 and TIOB1 as peripheral
piobase->PIO_PDR = (1<<PIOTIOA1) | (1<<PIOTIOB1) ;
// Timer/Counter 1 mode configuration
timerbase->TC_CMR =
TC_BSWTRG_SET_OUTPUT | // BSWTRG : software trigger set TIOB
TC_BCPC_TOGGLE_OUTPUT | // BCPC : Register C compare toggle TIOB
TC_BCPB_TOGGLE_OUTPUT | // BCPB : Register B compare toggle TIOB
TC_ASWTRG_SET_OUTPUT | // ASWTRG : software trigger set TIOA
TC_ACPC_TOGGLE_OUTPUT | // ACPC : Register C compare toggle TIOA
TC_ACPA_TOGGLE_OUTPUT | // ACPA : Register A compare toggle TIOA
TC_WAVE | TC_CPCTRG | // CPCTRG : Register C compare trigger enable
TC_EEVT_XC0 | // EEVT : XC0 as external event (TIOB=output)
TC_CLKS_MCK8; // TCCLKS : MCK / 8
```

Timer - Beispielprogramm (3)


```
// Compare registers initialization
timerbase->TC_RC = 31250 ;          // 100 Hz PWM generation
timerbase->TC_RA = 9375 ;           // 30% duty cycle on TIOA1
timerbase->TC_RB = 15625 ;          // 50% duty cycle on TIOB1
timerbase->TC_CCR = TC_CLKEN ;      // Enable the Clock counter
timerbase->TC_CCR = TC_SWTRG ;      // Trig the timer

for (;;) {}                         // Endlosschleife

return(0) ;                         // Rücksprung aus main()
}
```

Timer - Pulsweitenmodulation (PWM)



Timer - Pulsweitenmodulation (PWM)

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

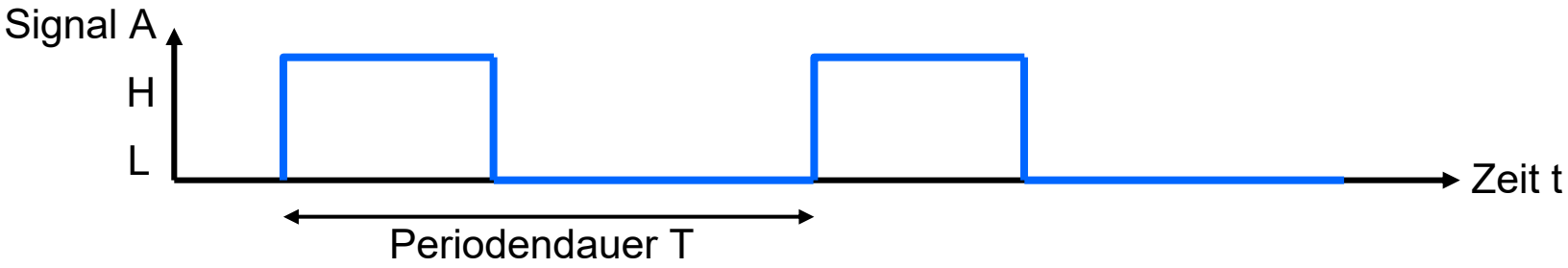
counter resolution = $1/F_{TC}$

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/32	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

Periodendauer / Frequenz

Allgemein: Frequenz $f = \frac{1}{T}$ mit der Periodendauer T

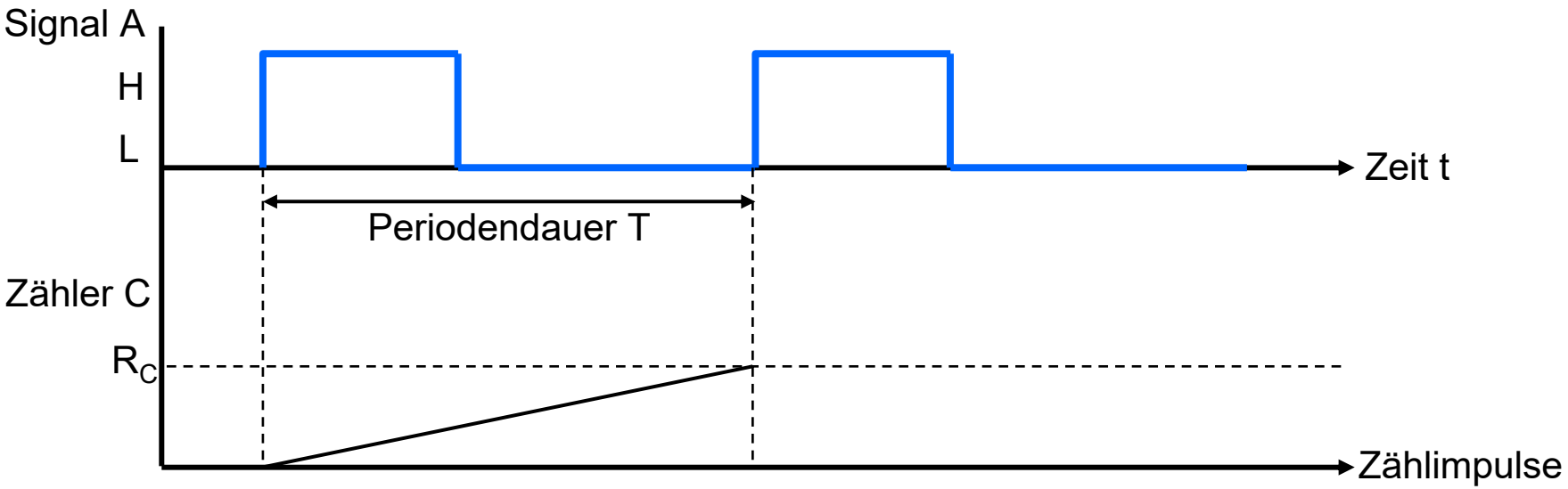


Signal A: $f_A = 100\ Hz$; $T_A = \frac{1}{100\ Hz} = 10 * 10^{-3} s = 10\ ms$

Systemtakt CLK: $f_{CLK} = 25\ MHz = 25*10^6\ Hz$

$$T_{CLK} = \frac{1}{25*10^6\ Hz} = \frac{1}{25}\ 10^{-6}\ s = 40*10^{-9}\ s = 40\ ns$$

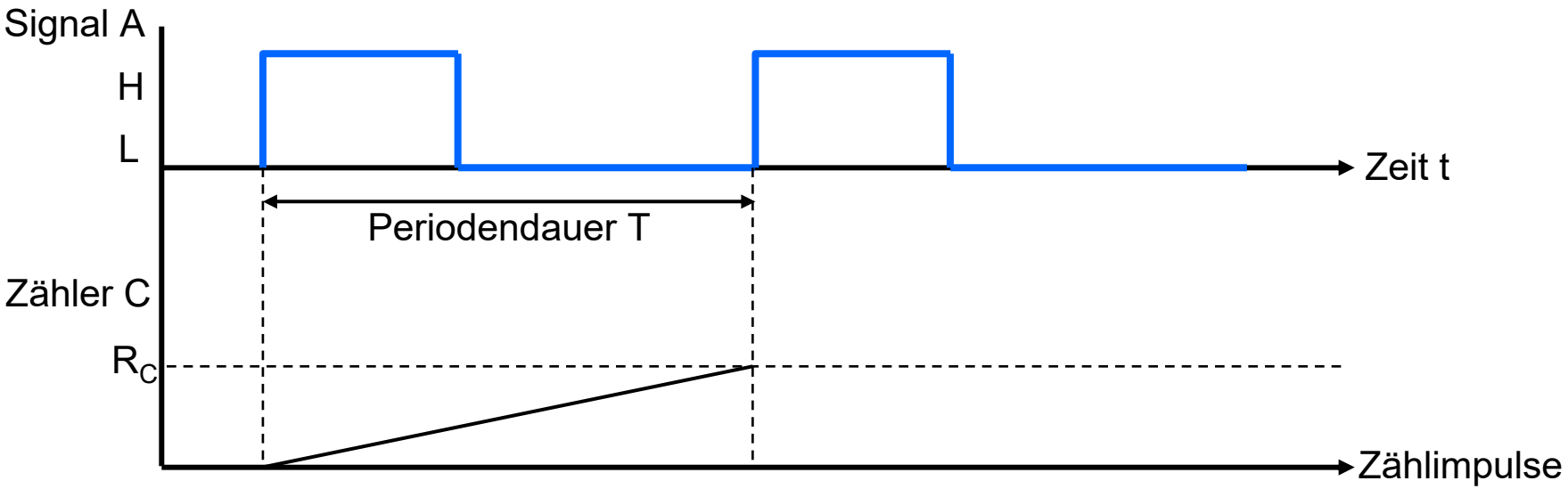
Periodendauer / Zählimpulse



Bedingung für die Anzahl Zählimpulse R_C:

nach Periodendauer T_A = 10 ms ist $R_C < 2^{16}$ Zählimpulse

Frequenzteiler (Prescaler)



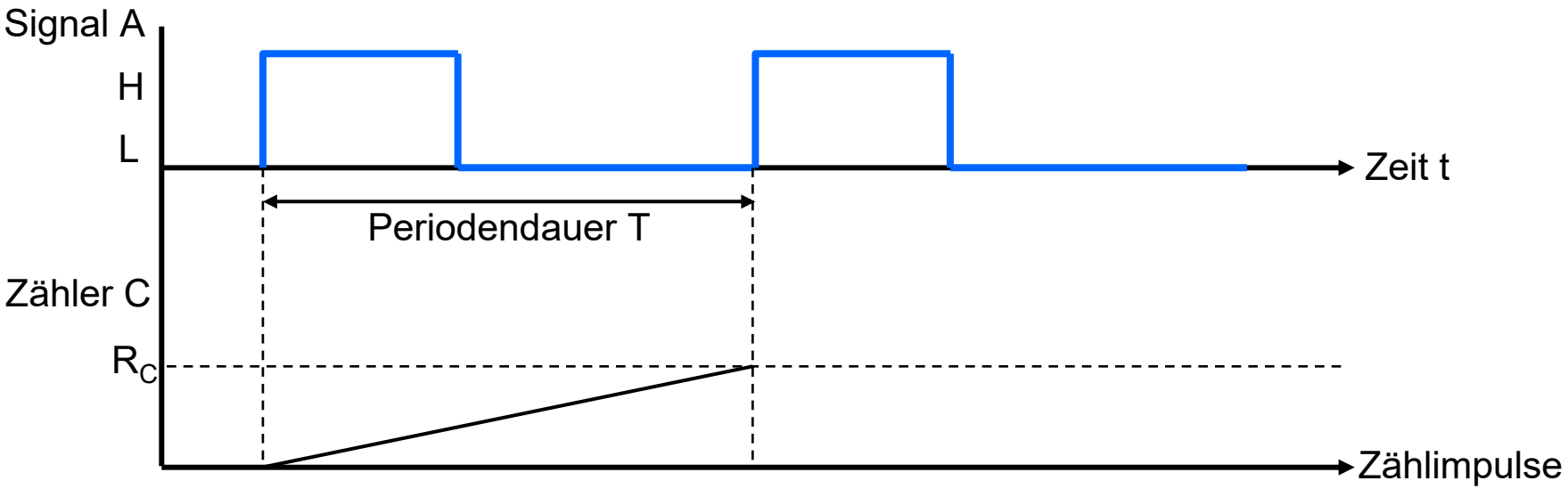
Frequenzteiler, auch Vorteiler (Prescaler)

für den Timerbaustein gibt es Vorteiler von 2, 8, 32, 128 und 1024

$$f_A = \frac{f_{CLK}}{Prescaler \cdot R_C}$$

$$Prescaler = \frac{f_{CLK}}{f_A \cdot R_C}$$

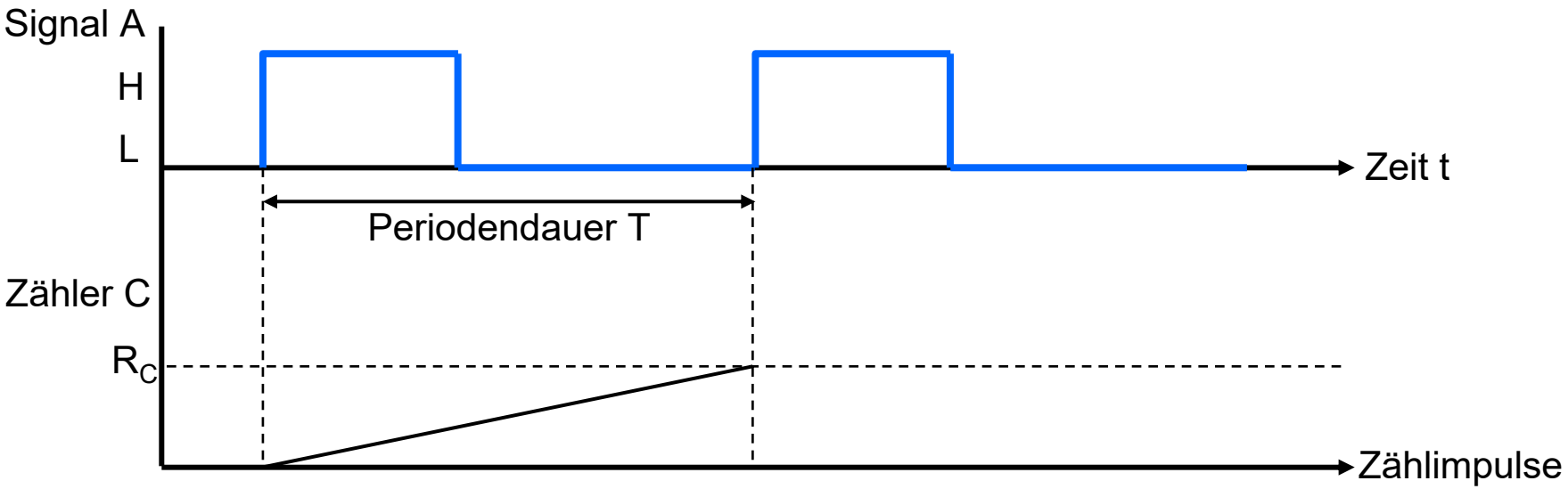
Frequenzteiler - ideal



f_{CLK} = 25MHz maximaler Wert für R_C = 65535

$$Prescaler = \frac{25\ MHz}{100Hz \cdot 65535} = \frac{250000}{65535} \approx 3{,}81$$

Frequenzteiler – wählbare Werte



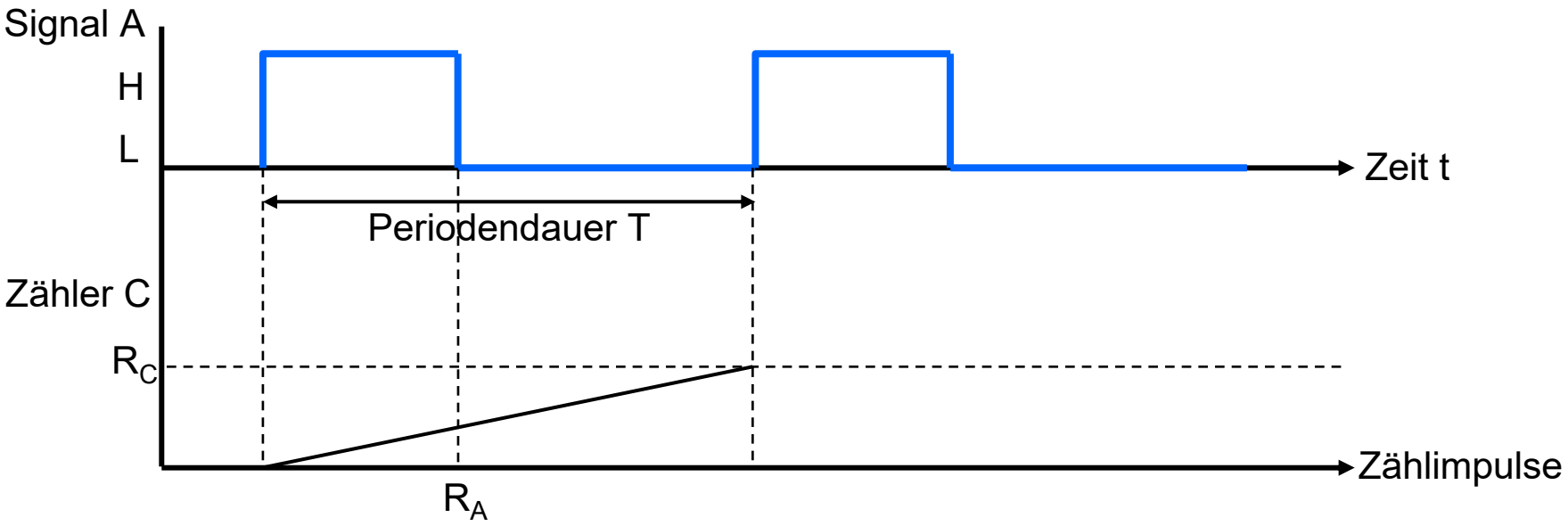
f_{CLK} = 25MHz maximaler Wert für R_C = 65535

Benötiger Frequenzteiler 3,81

Verfügbare Frequenzteiler: 2, **8**, 32, 128, 1024

Der nächstgrößere Frequenzteiler liefert die bestmögliche Auflösung (*Prescaler* = 8).

Signalerzeugung / Wahl der R-Werte



Wie berechnet sich R_C bei einem Prescaler 8?

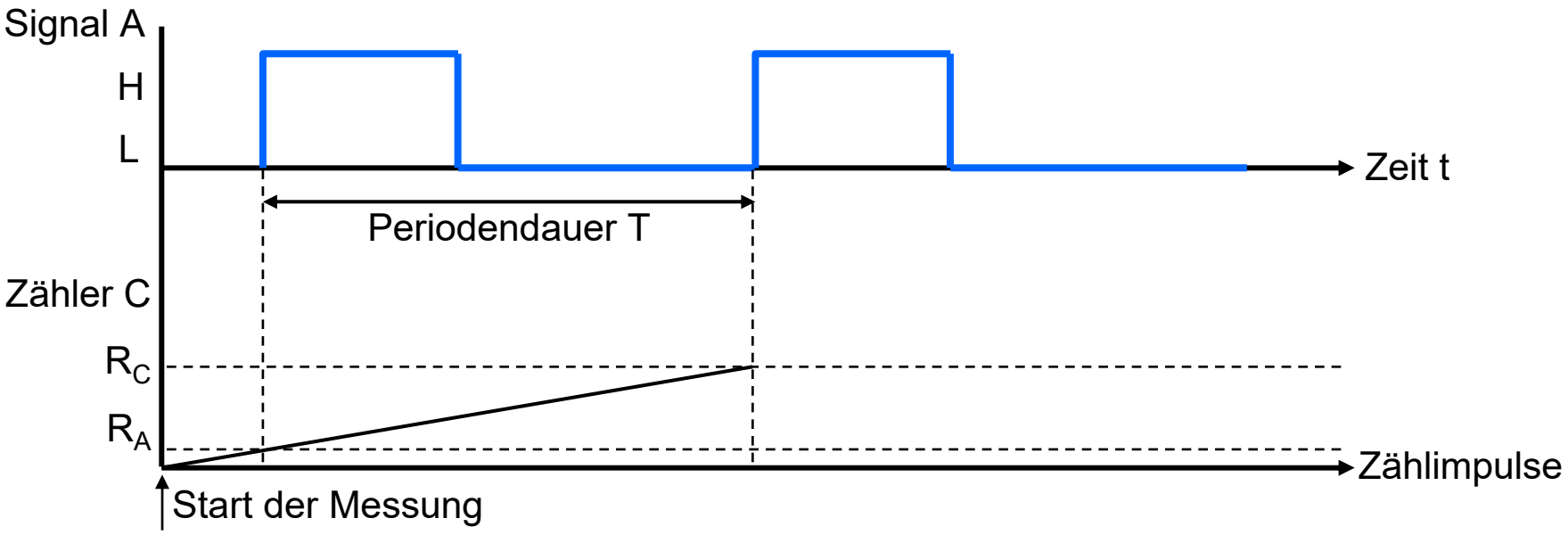
Zeitdauer eines Zählimpulses $T_{prescaler} = 8 \cdot 40ns = 320ns$

Pulsweite $T_A = 10\ ms = 10 \cdot 10^6\ ns$

$R_C = T_A / T_{prescaler} = 10 \cdot 10^6 / 320 = 1/32 \cdot 10^6 = 31250$

Wie berechnet man R_A (30%) bzw. R_B (50%)? …

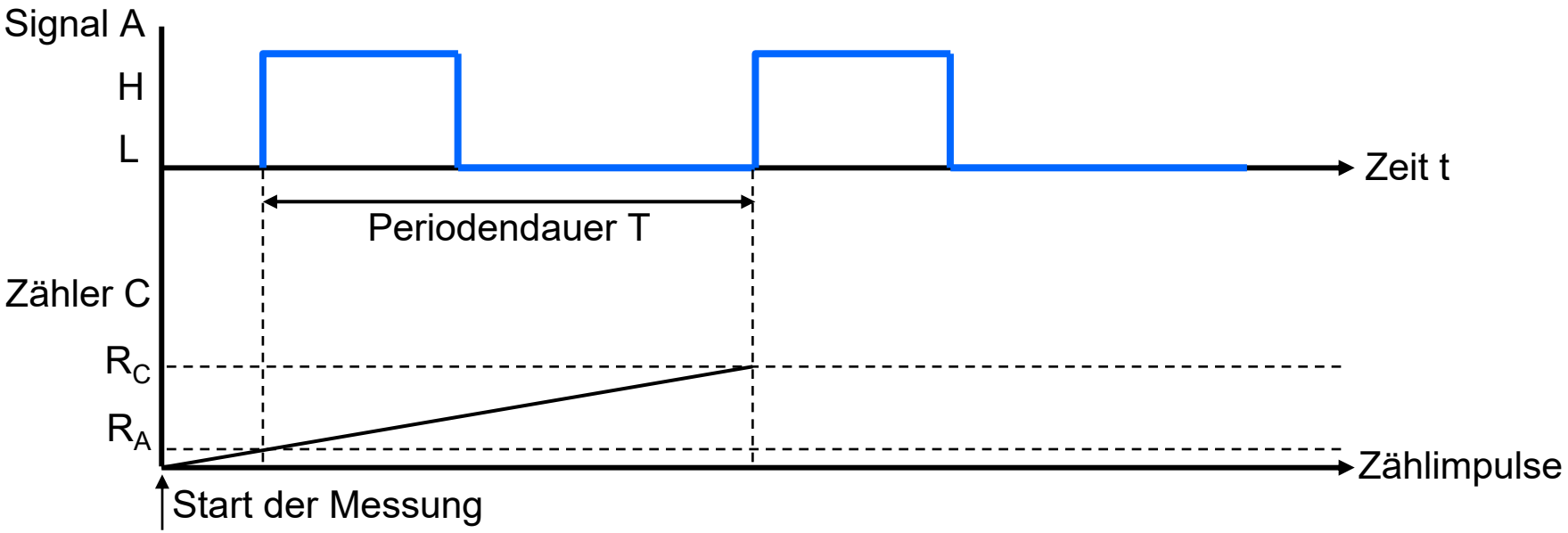
Messung einer Periode T



Messung der Periodendauer T bei den positiven Flanken

$R_C - R_A$:
Anzahl der Zählimpulse während einer Periodendauer

Messperiode / Frequenz



Beispiel:

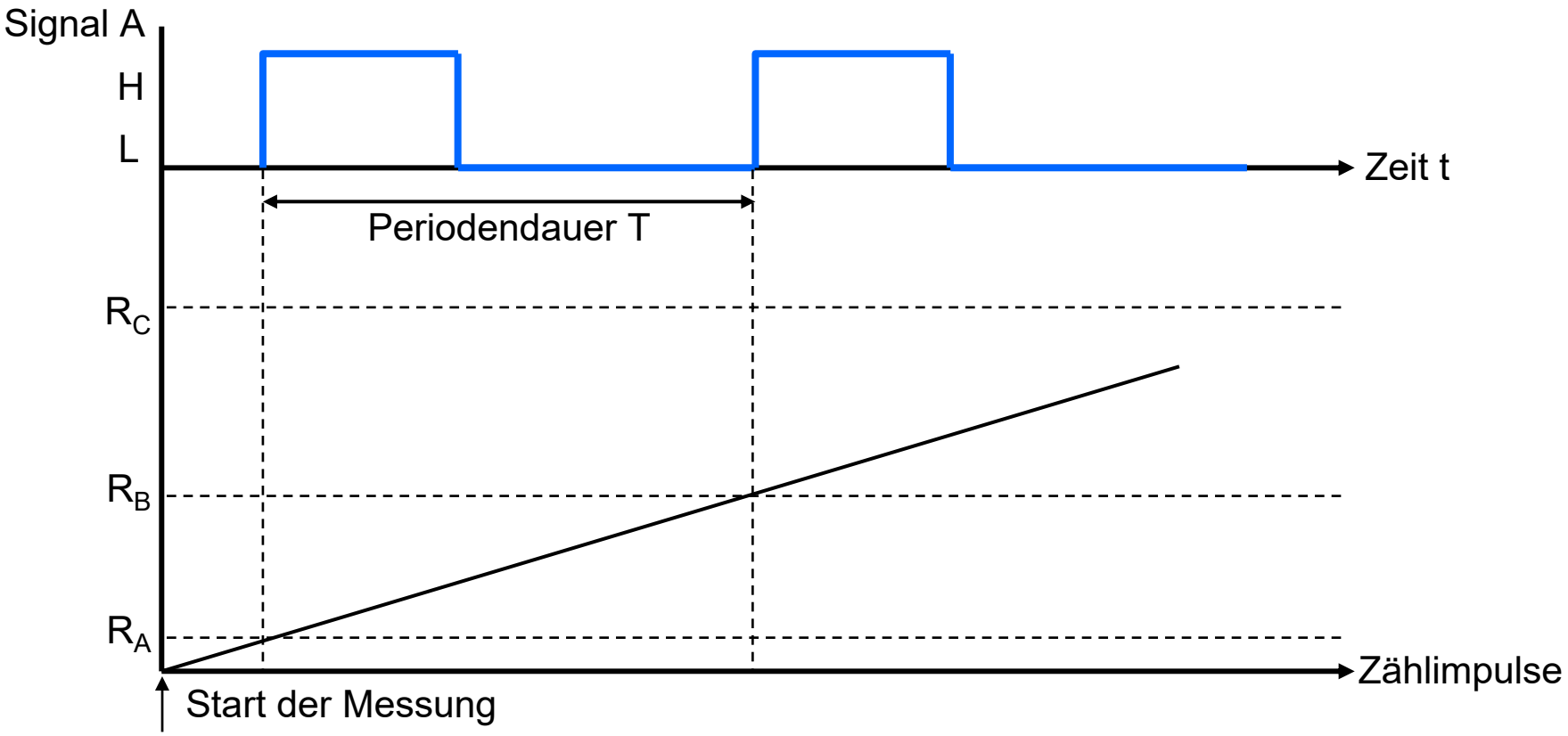
Frequenzteiler 8; $R_C = 3250$; $R_A = 125$

$R_C - R_A = 3125$

T = 320ns * 3125 = 1000000 ns = 1 ms

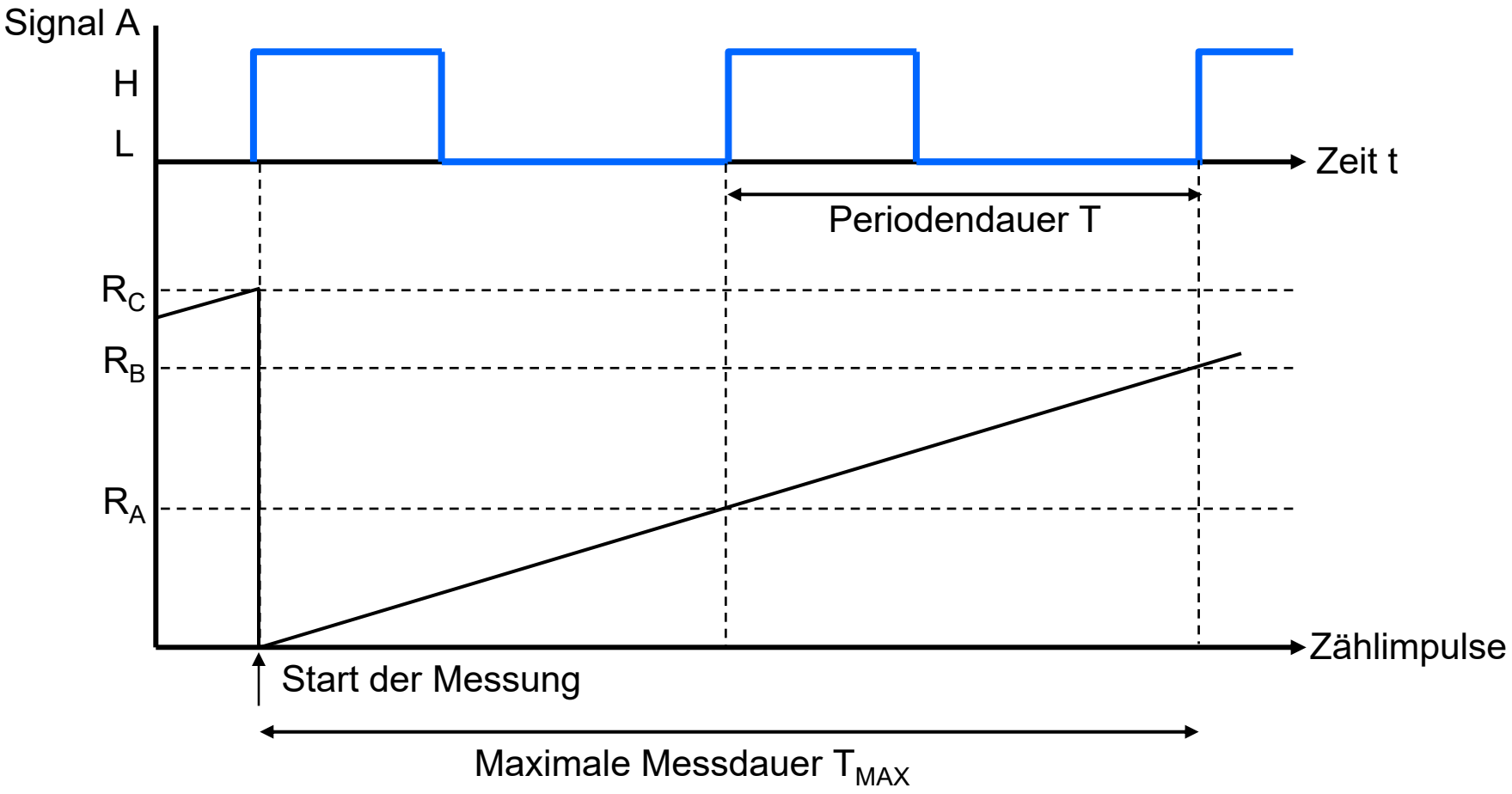
$f_{Signal} = 1000$ Hz

Benötigte Messdauer



Welche maximale Zeitdauer einer Messung muss berücksichtigt werden, wenn diese bei Triggerung durch positive Taktflanken einem beliebigen Startzeitpunkt hat?

Messung einer Periodendauer T



Wenn die Messung zu einem beliebigen Zeitpunkt startet, können fast zwei Periodendauern in ein Messintervall fallen.

Anwendung in Betriebssystemen

Wie stoppen Sie einen CD-Laufwerkmotor nach einer bestimmten Zeitdauer, ohne eine ständige Abfrage machen zu müssen?

Wo sind Quellen für Atmel-Timer-Bausteine im Linux-Kernel zu finden?

Suchen Sie auf https://www.kernel.org/

Ansichten: cgit -> tree

Verzeichnis: include/linux

Welchen Inhalt hat folgende Datei?

atmel_tc.h

Beispiel: Kernel-Timer in Linux

Was sind Timer in einem Betriebssystem-Kernel?

Suchen Sie auf

www.oreilly.de/german/freebooks/linuxdrive2ger/book1.html

Kapitel 6 – Der Lauf der Zeit

-> Kernel-Timer

Vergleichen Sie dies mit folgender Datei auf kernel.org

Include/linux/timer.h

Gliederung

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

SWI - Software Interrupt – Aufgaben I

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

SWI –Software Interrupt – Aufgaben II

- 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 privilegierten Mode möglich.

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 eingeblendet.

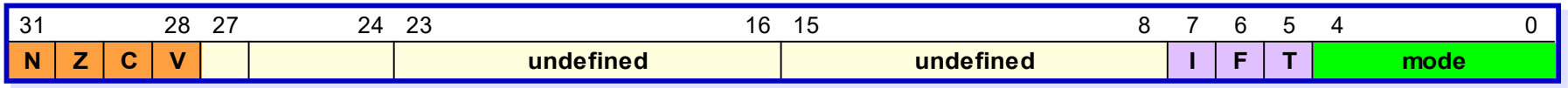
Einzelschritte:

- 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)

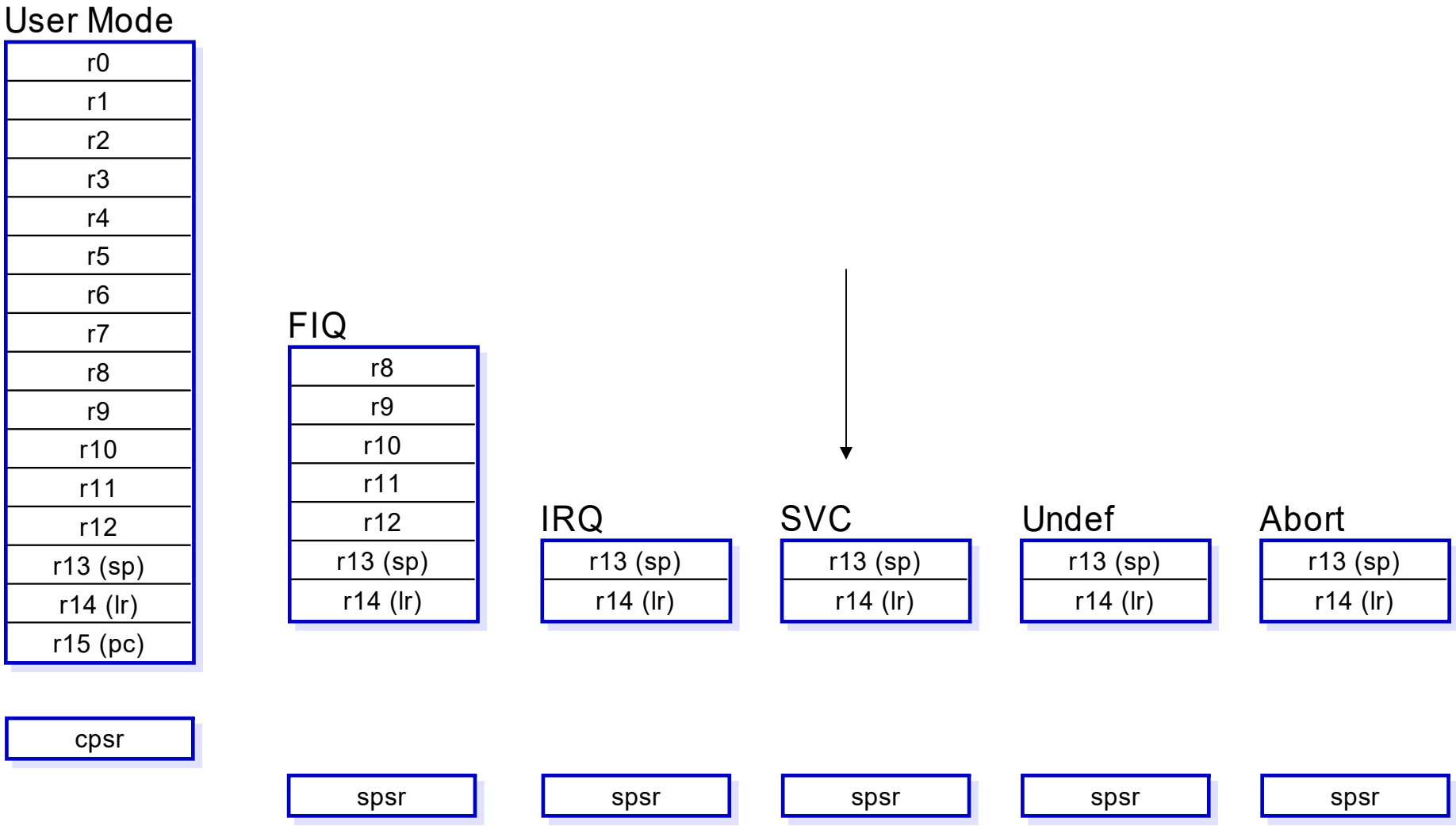
SWI - Program Status Register



- 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

SWI - Register ARM



SWI - Interrupt Vektor

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

SWI – Aufruf des Interrupthandlers

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

SWI - Rückkehr aus Interrupt (1)

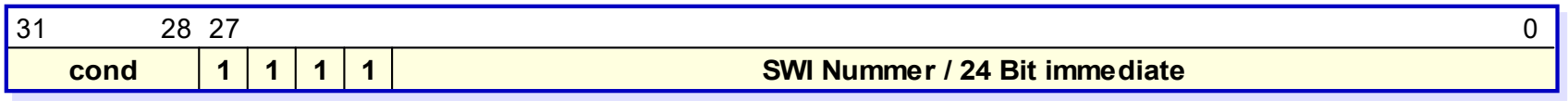
- Das alte Programm Status Register muß wieder hergestellt werden, so daß Bedingungen, Interrupt Status und alter Mode wieder vorhanden sind, d.h. SPSR_SVC muß wieder zu CPSR werden
- Es muß ein Rücksprung in das aufrufende Programm erfolgen, indem LR_SVC nach PC kopiert wird.
- Beide Aktionen müssen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

SWI - Rückkehr aus 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}^

SWI - Syntax



- 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

SWI - Zugriff

Zugriff auf die Nummer des SWI Befehls

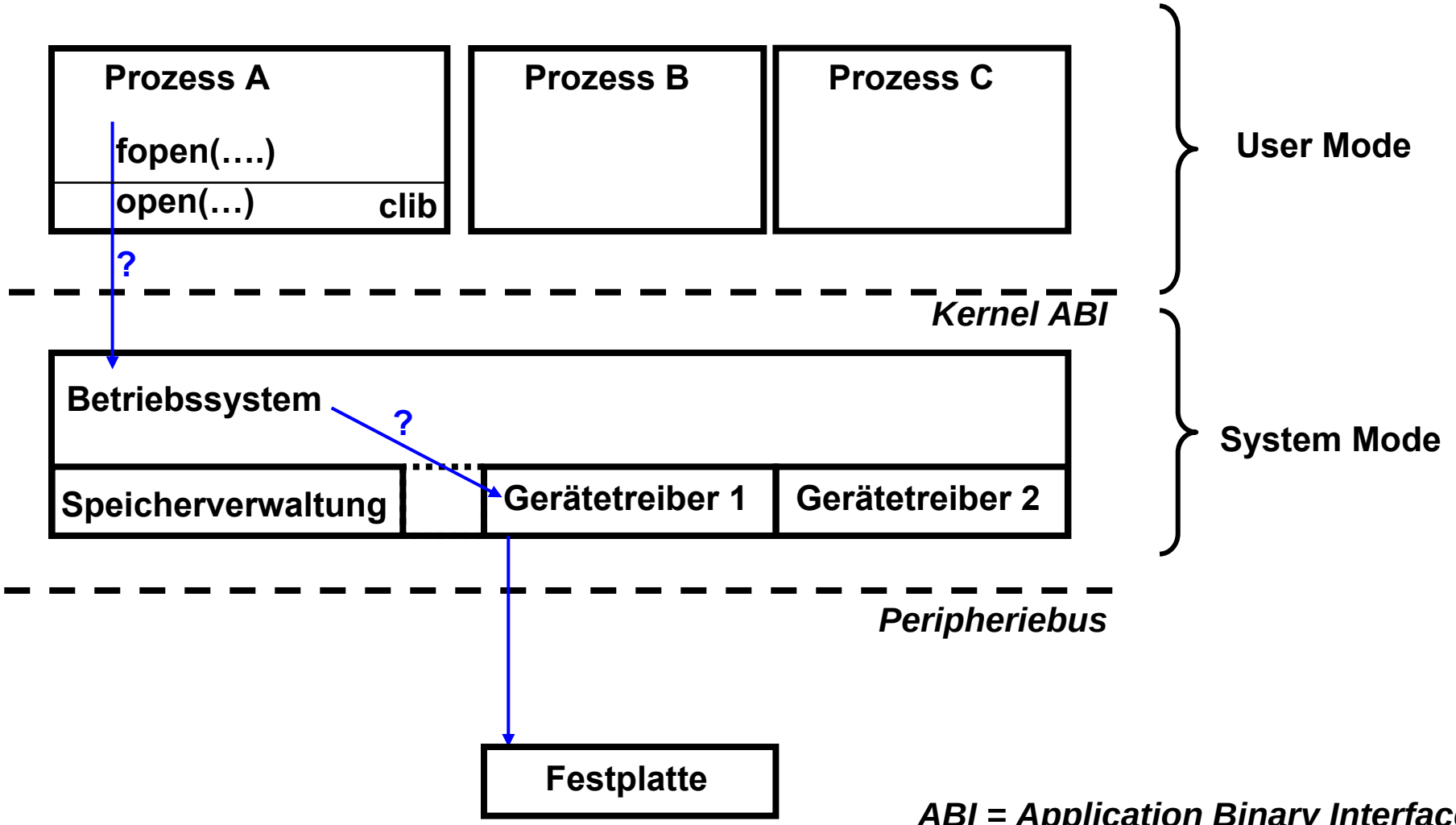
- ldr r0, [lr, #-4] @ Befehlscode wird geladen
- bic r0, r0, #0xff000000 @ Obere 8 Bits werden maskiert

SWI - ein einfacher SWI Handler


```
    .global SWIHandler
    .text
SWIHandler:
    stmfd   sp!, {lr}           @ Retten von Ruecksprungadresse
    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
```

SWI – Anwendung in Betriebssystemen

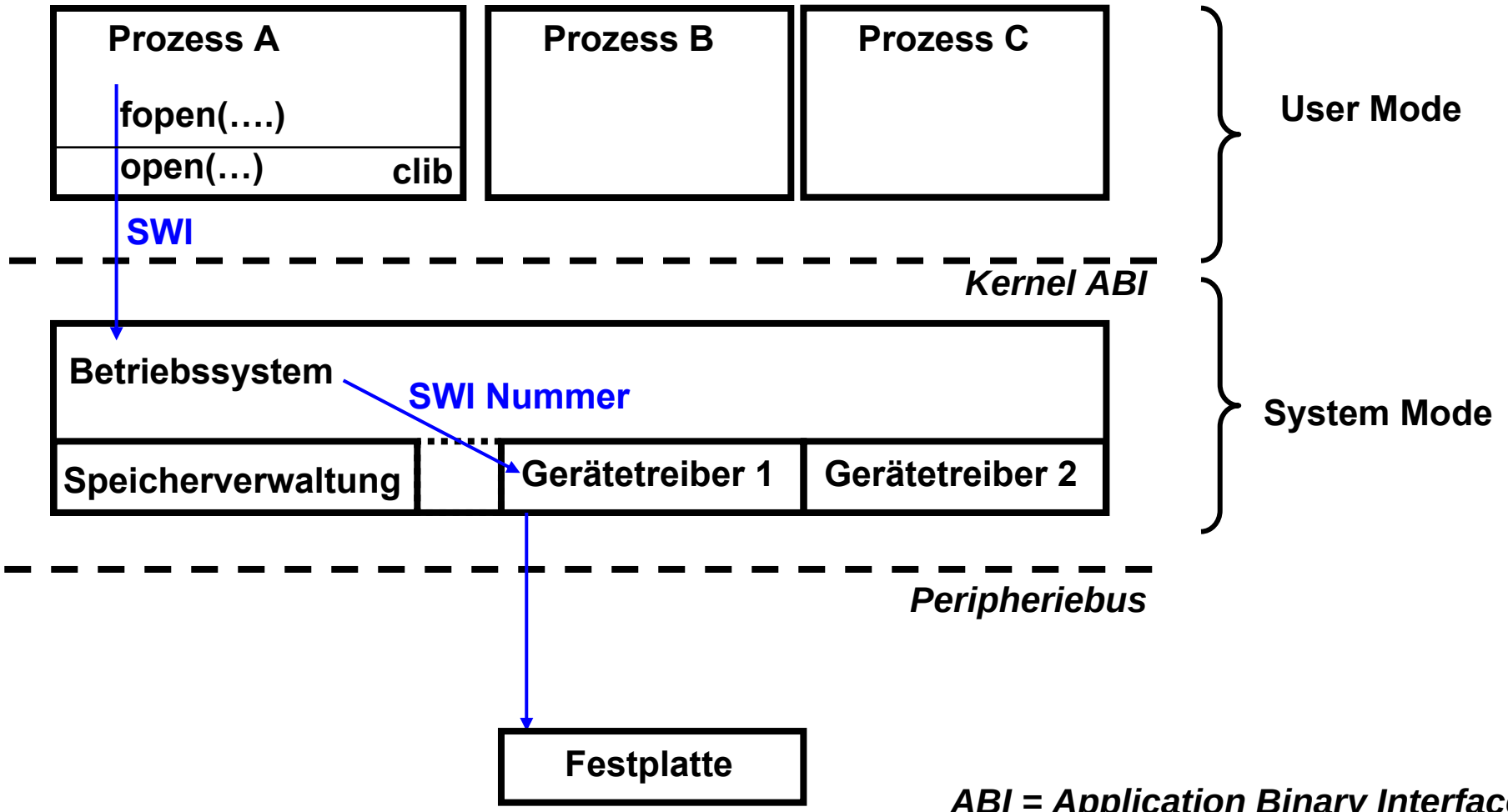
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK



ABI = Application Binary Interface

SWI – Systemaufrufe mittels SWI

Systemaufrufe (System Calls) werden mittels SWI realisiert. Dem jeweils benötigten Systemaufruf ist eine SWI-Nummer assoziiert.



ABI = Application Binary Interface

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

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)

SWI – Beispiel: Linux Kernel-Entry

Suchen Sie auf https://www.kernel.org/

Ansichten: Release (z.B. stable) browse -> tree

Verzeichnis: arch/arm/kernel

Suchen Sie in folgender Datei den SWI-Handler.

Entry-common.S

Was muss ein Betriebssystem wie Linux in einem SWI-Handler berücksichtigen?

Gliederung

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

Serielle Schnittstelle - Überblick

CAN

WLAN

2 Draht Übertragung

DSL

Ethernet

Synchrone Kommunikation

ISDN

Asynchrone Kommunikation

Firewire

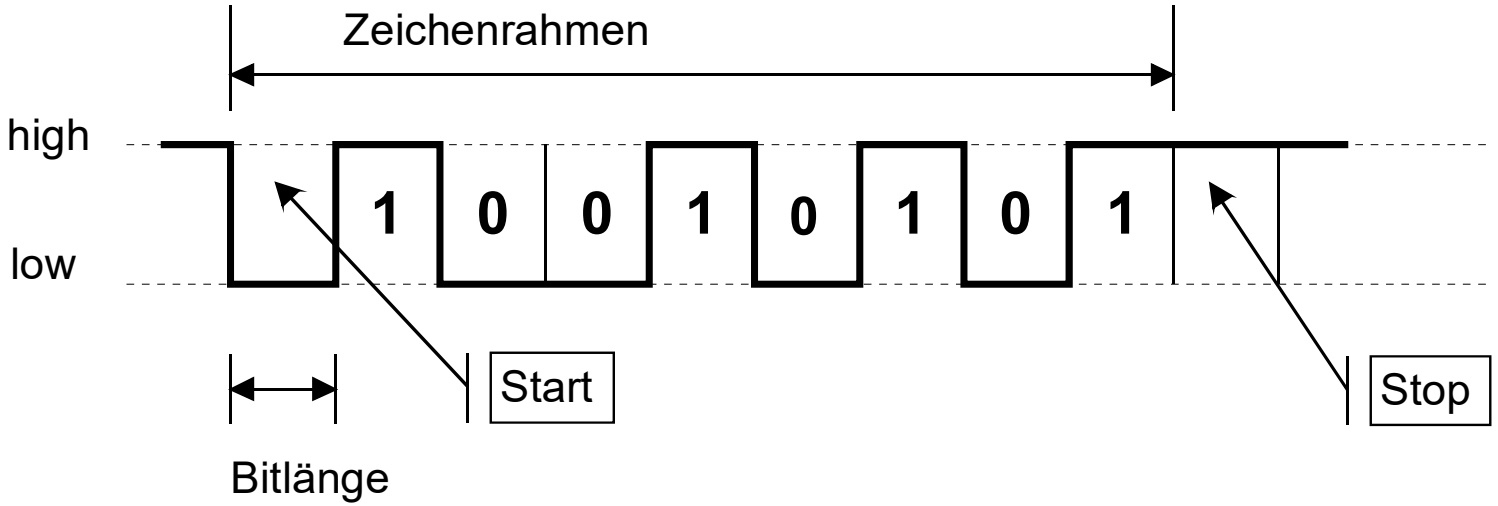
USB

Funk Netze

Optische Übertragung

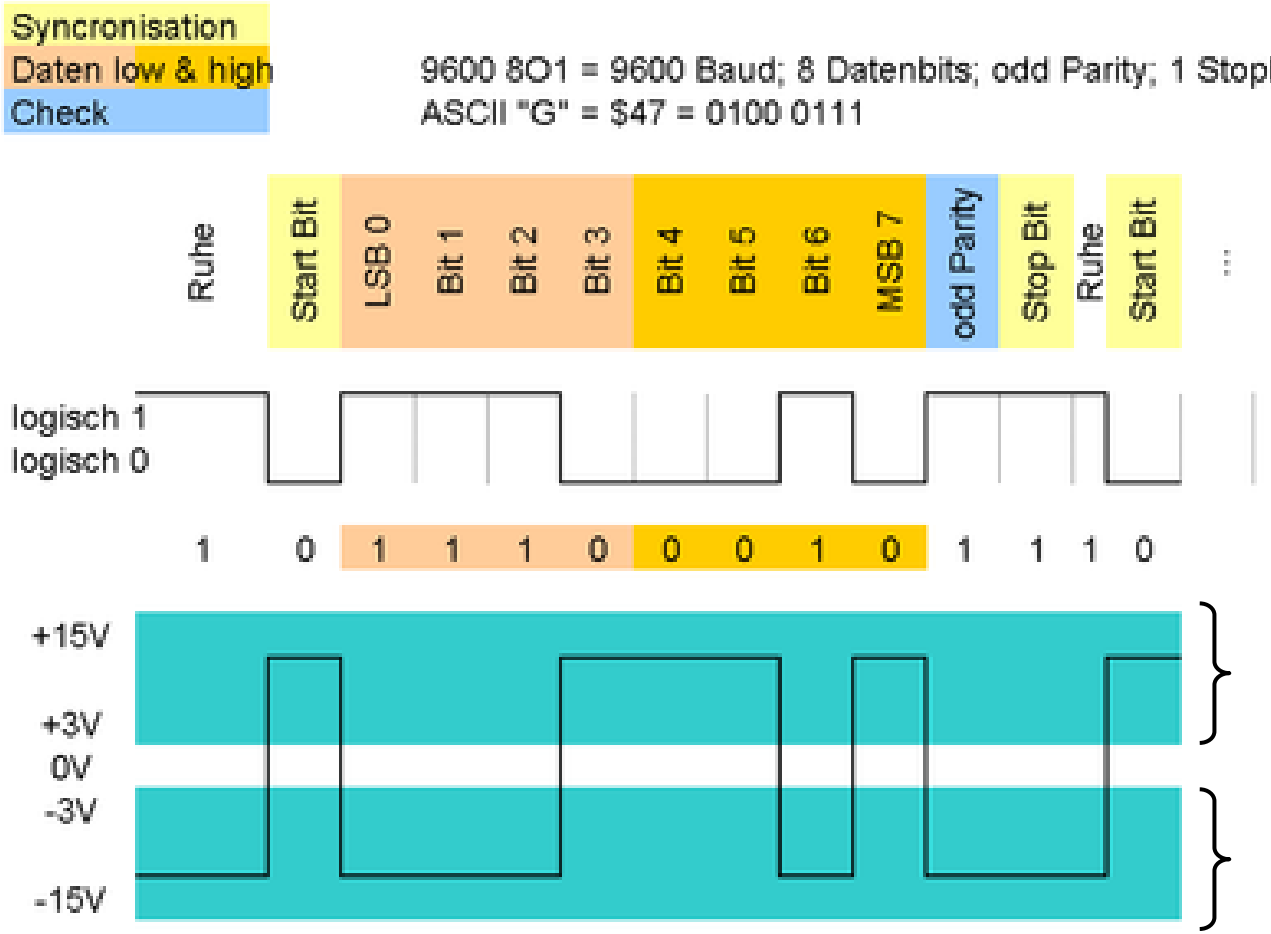
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Asynchrone Datenübertragung



Serielle Schnittstelle

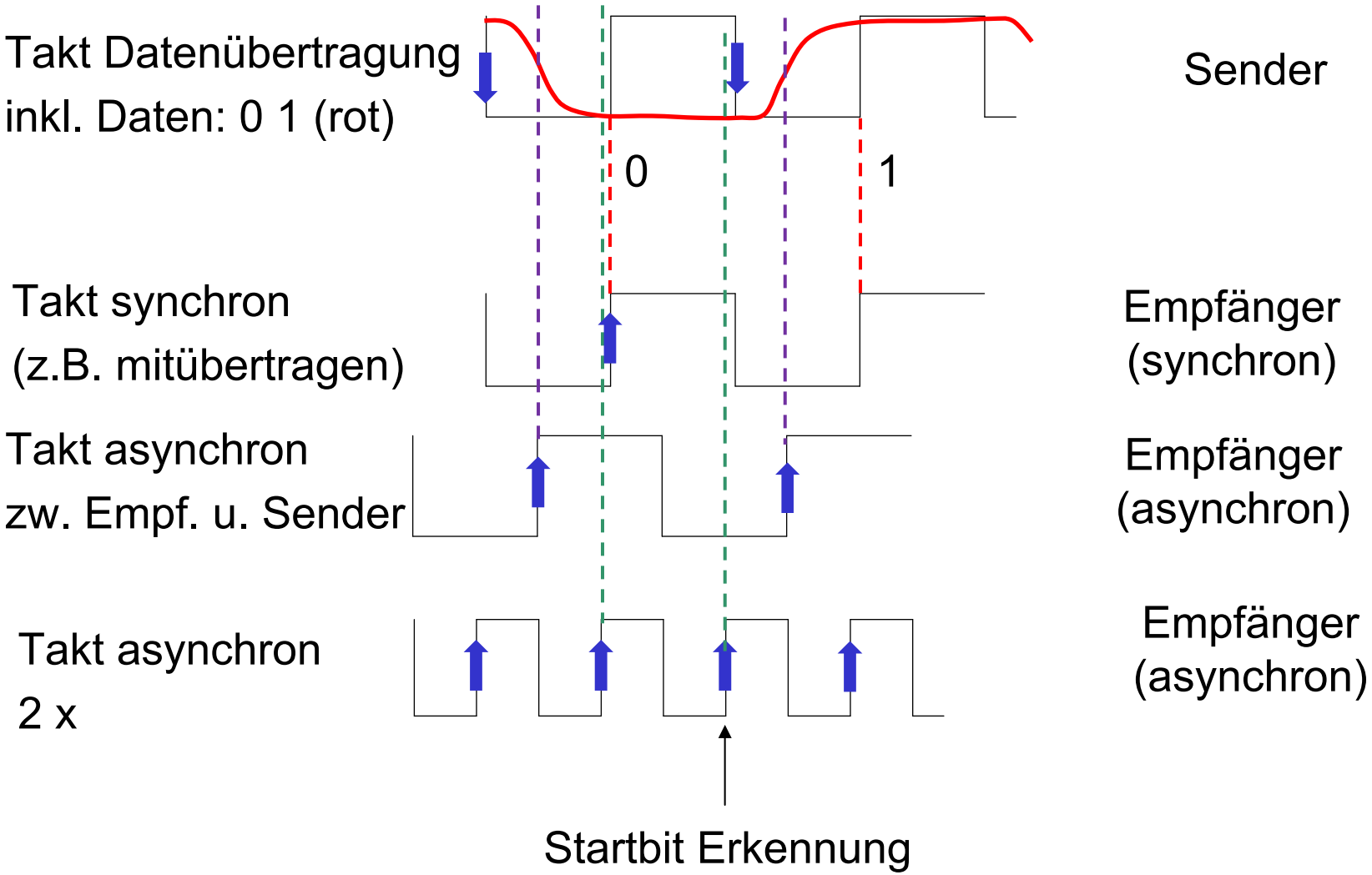
Asynchrone Datenübertragung mit 9600 Baud (Bit/Sekunde): dies entspricht einer Bitlänge von 0,104 ms



Erlaubt symmetrische Signalüber-tragung

Quelle: Wikipedia

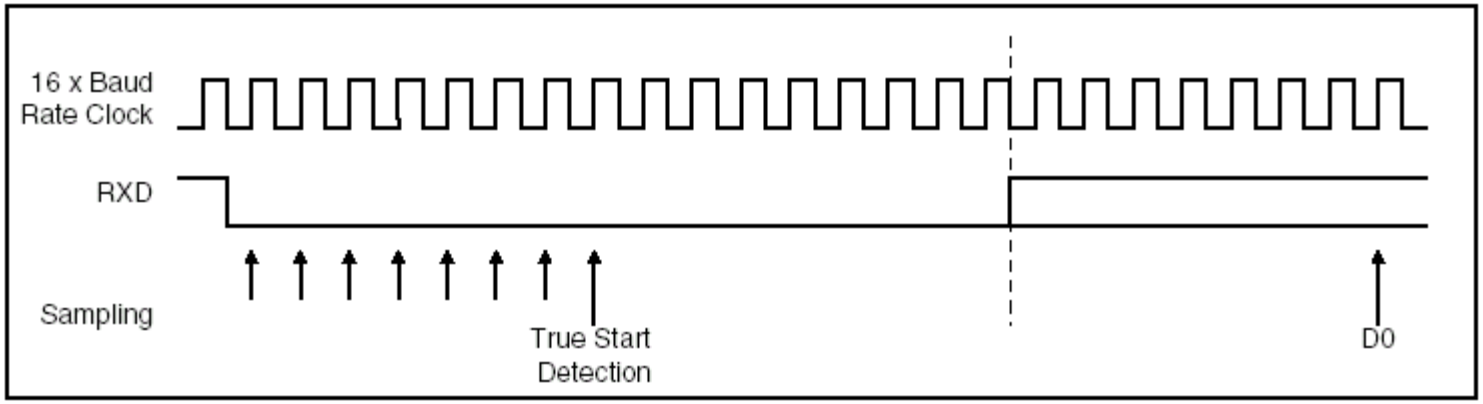
Serielle Schnittstelle - asynchron



h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

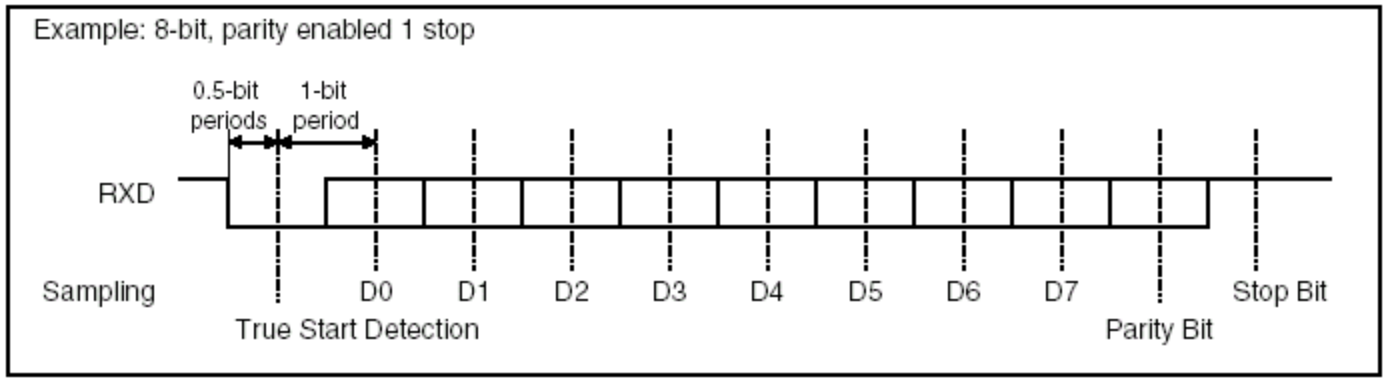
Asynchroner Modus

Start Bit Erkennung



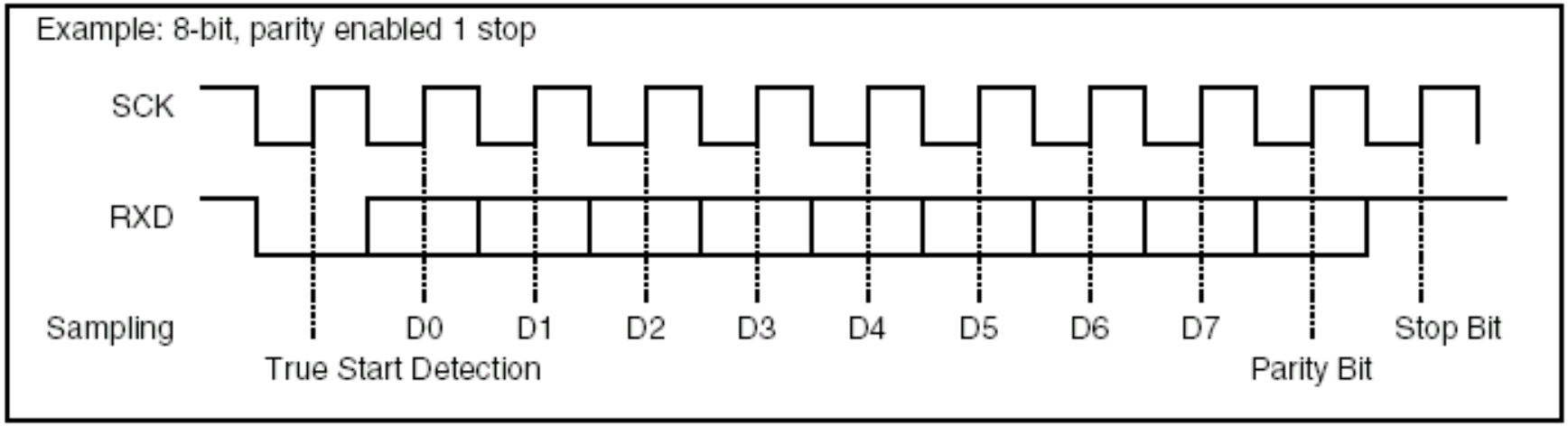
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Asynchroner Modus
Zeichenerkennung



Serielle Schnittstelle - synchron

Synchroner Modus



Serielle Schnittstelle - Aufgaben

- Daten serialisieren und senden
- Daten empfangen und abspeichern

Grundfunktionen serieller Kommunikation

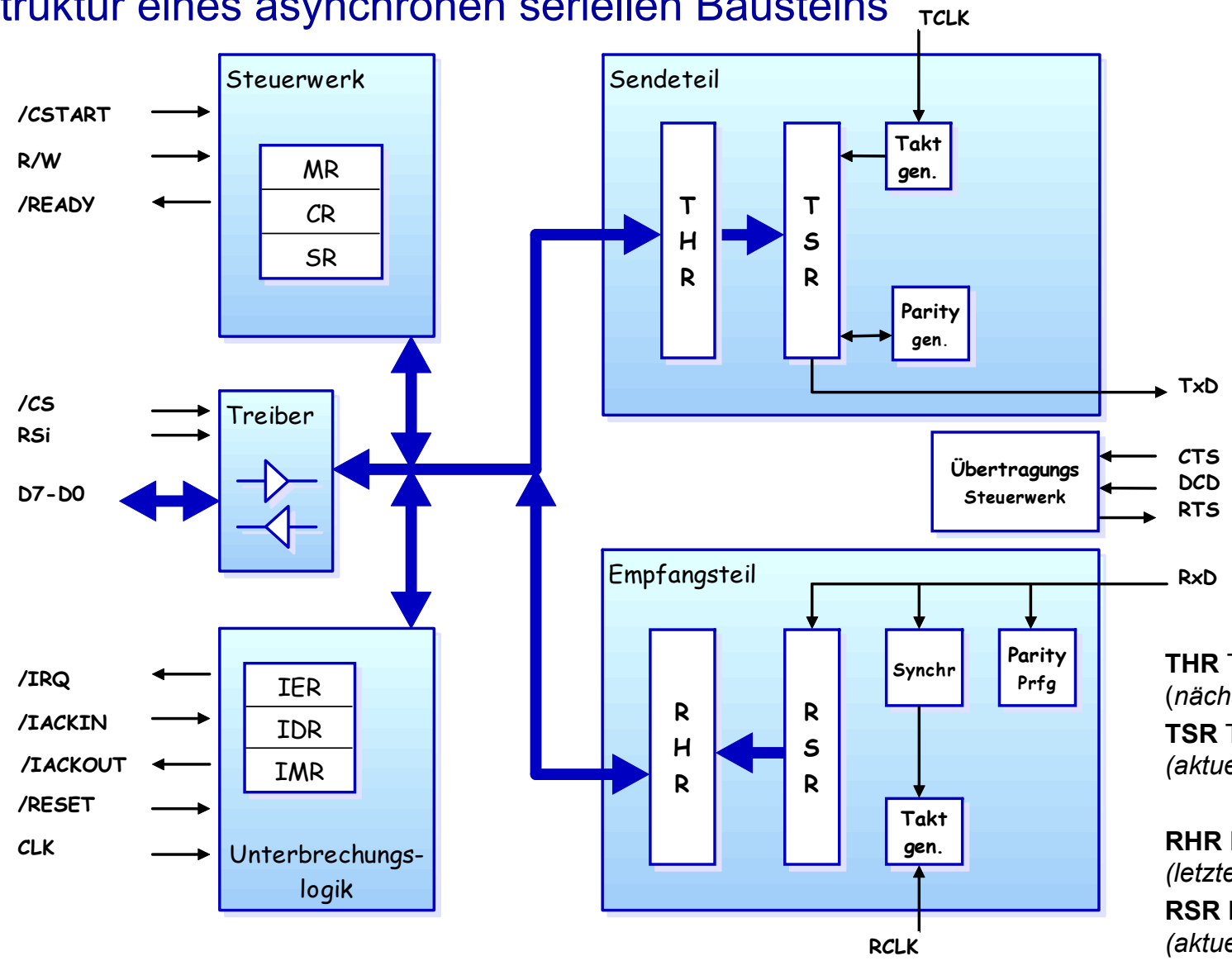
- Schnittstelle initialisieren
- Nachricht schicken
- Nachricht empfangen

Serielle Schnittstelle - einfache API

- serial_init()
  - Initialisierung, Einstellen der Übertragungsrate
- void putchar(char c)
  - Senden eines Zeichens
- char getchar(void)
  - Empfang eines Zeichens
- bool write(char* buffer, int len)
  - Senden einer Zeichenkette
- bool read(char* buffer, int len)
  - Empfang einer Zeichenkette

Serielle Schnittstelle - Struktur

Struktur eines asynchronen seriellen Bausteins



Transmitted Data

Clear To Send
Data Carrier Detect
Request To Send

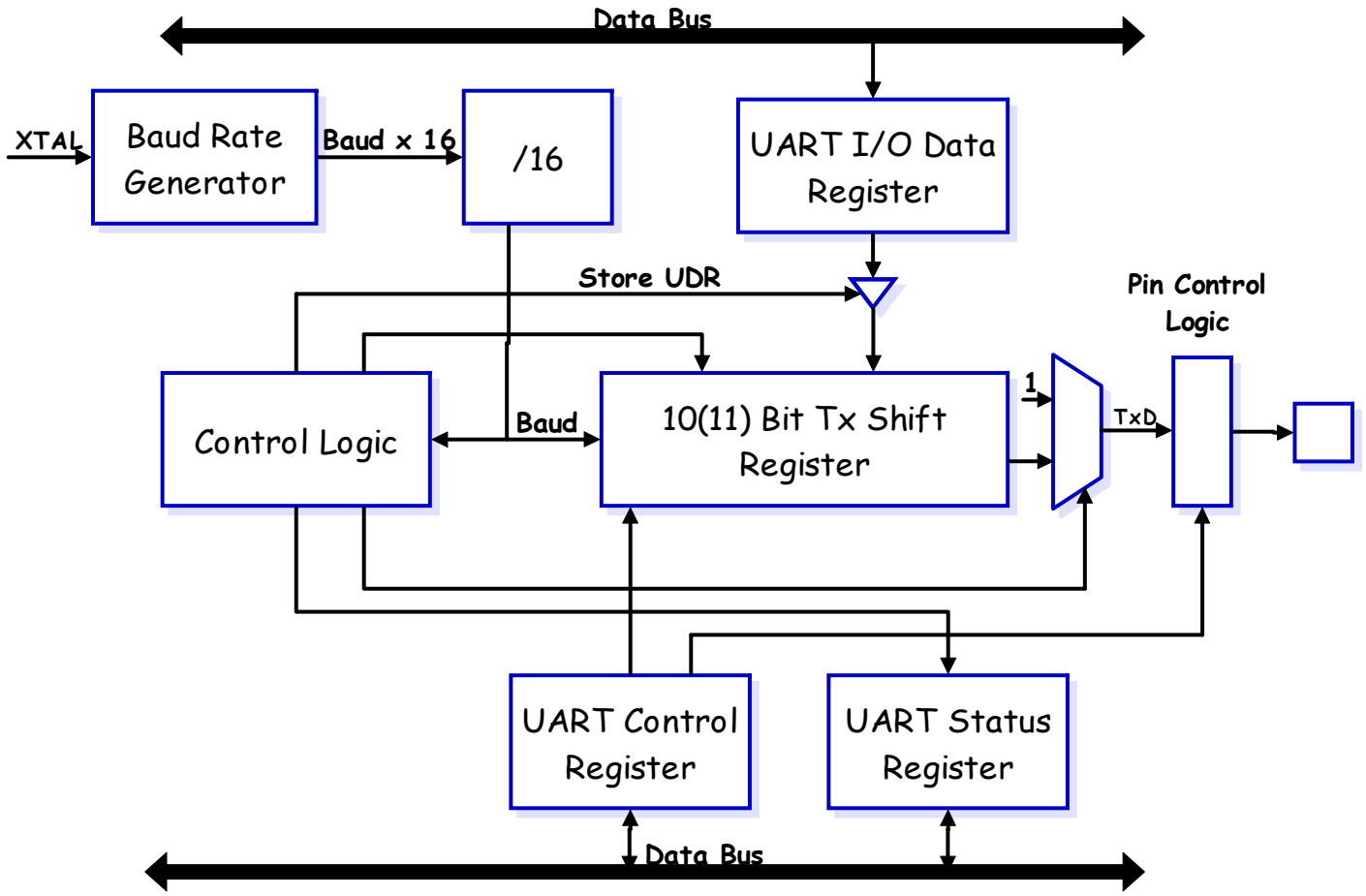
Received Data

THR Transmit Holding Register
(nächstes zu sendendes Wort)
TSR Transmit Shift Register
(aktuell gesendetes Wort)

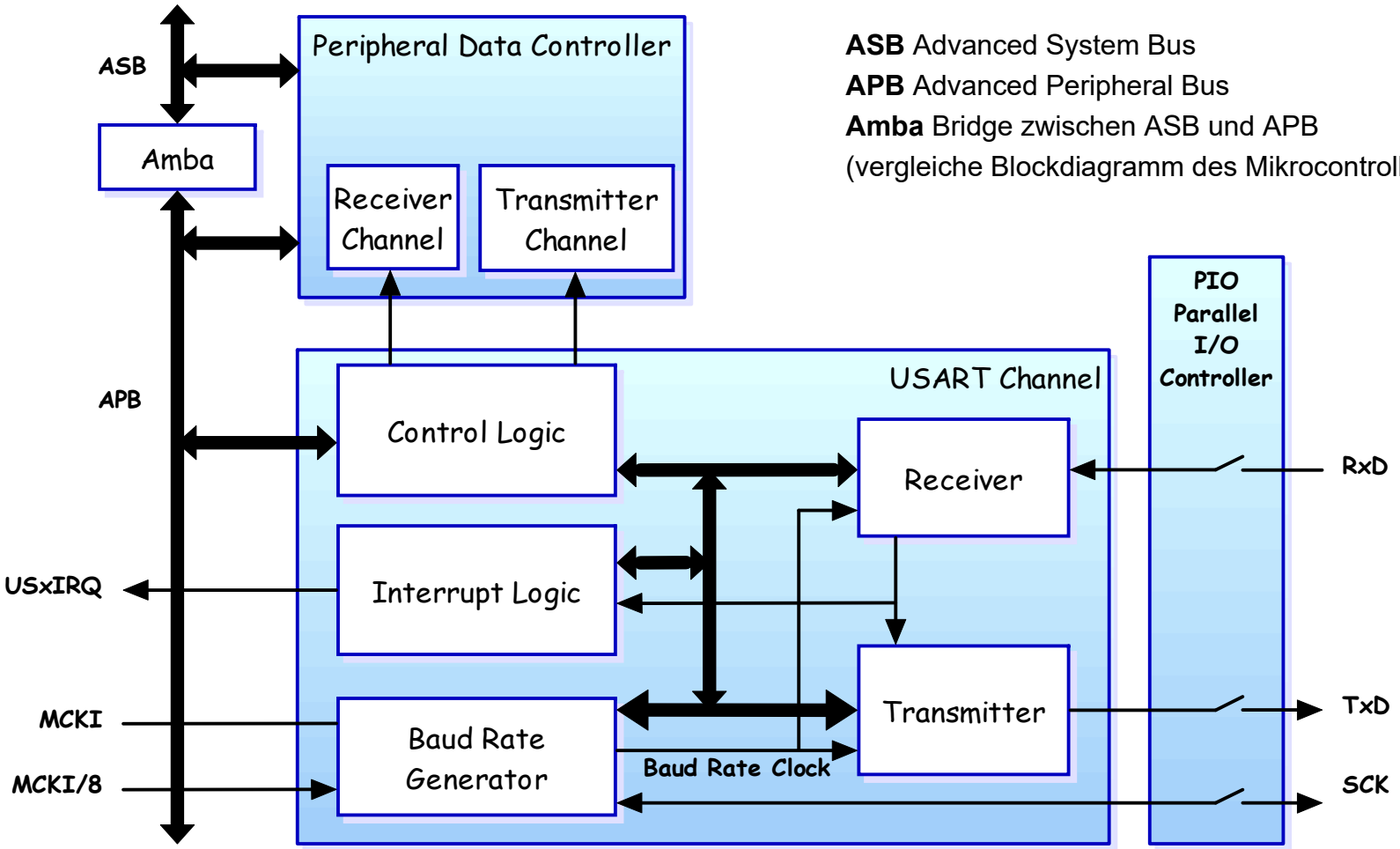
RHR Receive Holding Register
(letztes empfangenes Wort)
RSR Receive Shift Register
(aktuell empfangenes Wort)

Serielle Schnittstelle - Aufbau

innerer Aufbau des Sendebausteins



USART Struktur mit DMA Prozessor

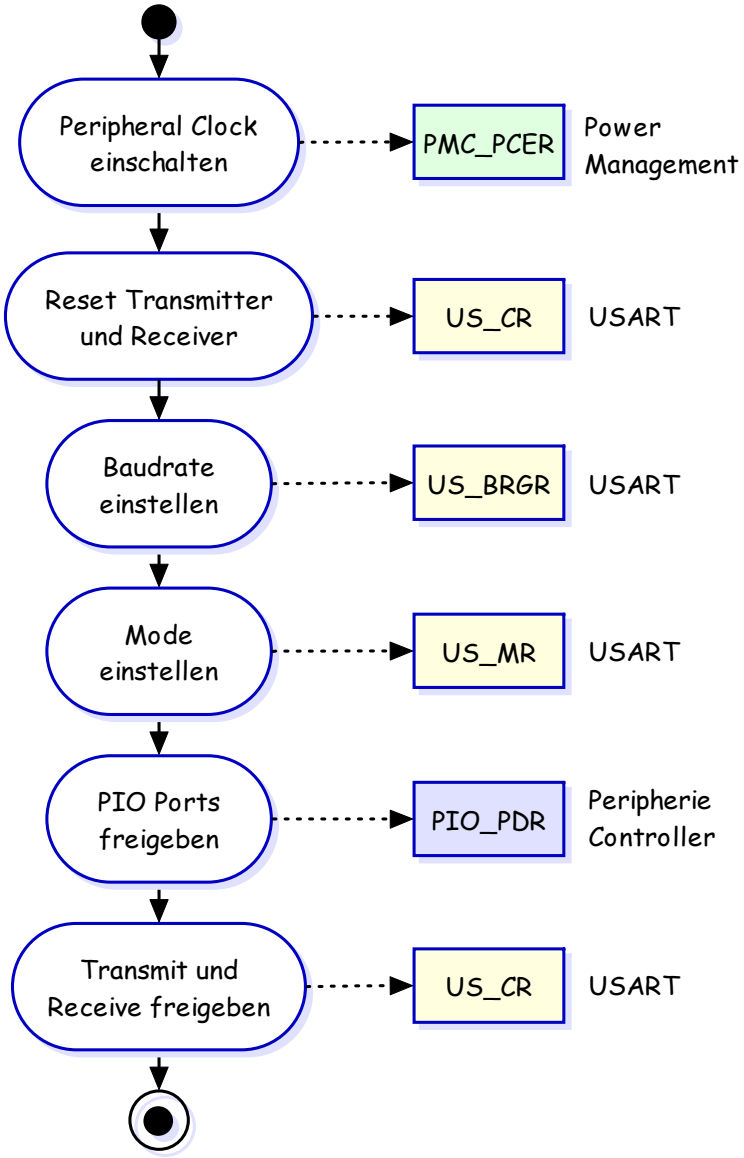


ASB Advanced System Bus
APB Advanced Peripheral Bus
Amba Bridge zwischen ASB und APB
(vergleiche Blockdiagramm des Mikrocontrollers

USART Register Anordnung

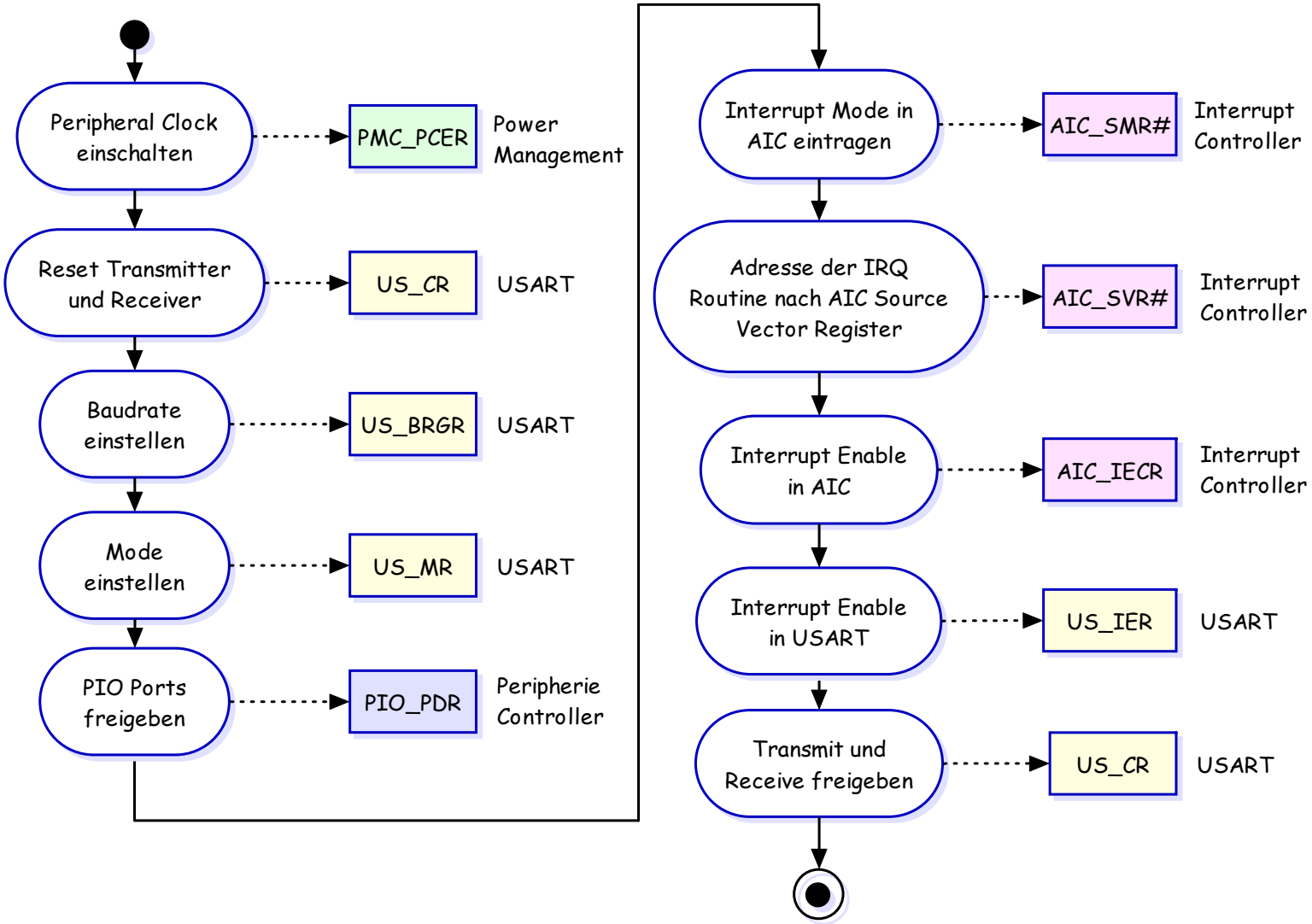
Offset	Register	Name	Access	Reset State
0x00	Control Register	US_CR	Write only	-
0x04	Mode Register	US_MR	Read/write	0
0x08	Interrupt Enable Register	US_IER	Write only	-
0x0C	Interrupt Disable Register	US_IDR	Write only	-
0x10	Interrupt Mask Register	US_IMR	Read only	0
0x14	Channel Status Register	US_CSR	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 Register	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

Serielle Schnittstelle - Initialisierung



Serielle Schnittstelle - Interrupt

Initialisierung eines USART Bausteins mit Interrupt



Serielle Schnittstelle – Beispiel (1)

Beispiel einer Initialisierung über Tabelle

```
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
```

Serielle Schnittstelle – Beispiel (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:
```

Abbildung der Register in einer C-Struktur

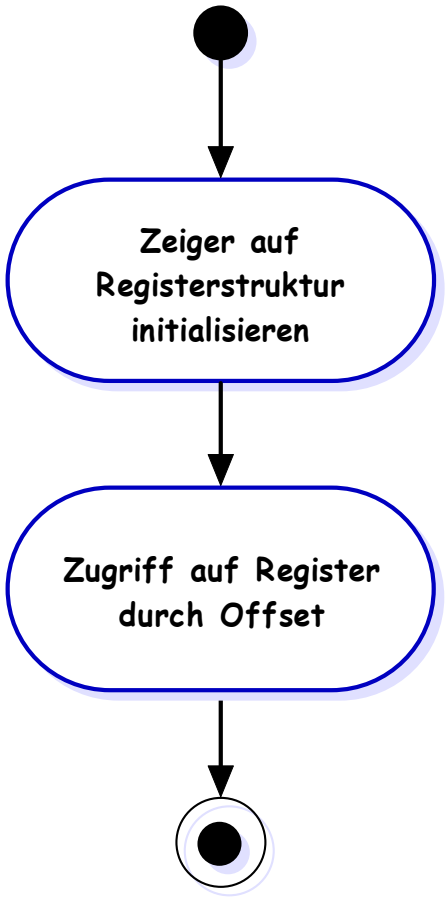
```
struct USART {
    volatile unsigned int  US_CR;
    volatile unsigned int  US_MR;
    volatile unsigned int  US_IER;
    volatile unsigned int  US_IDR;
    volatile unsigned int  US_IMR;
    volatile unsigned int  US_CSR;
    volatile unsigned int  US_RHR;
    volatile unsigned int  US_THR;
    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;
}
```

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

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: // Inhalt Channel Reg.
    .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

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	0	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

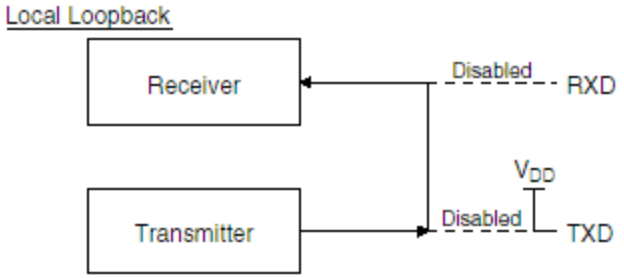
USART Mode Register

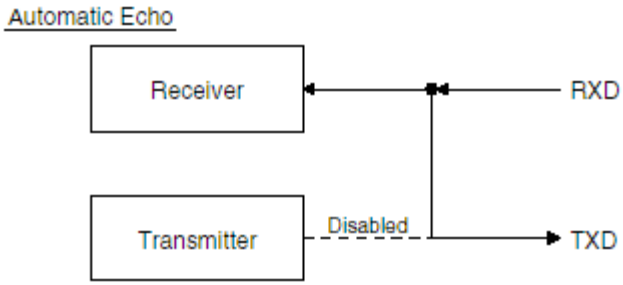
NBSTOP		Asynchronous (SYNC = 0)	Synchronous (SYNC = 1)
0	0	1 stop bit	1 stop bit
0	1	1.5 stop bits	Reserved
1	0	2 stop bits	2 stop bits
1	1	Reserved	Reserved

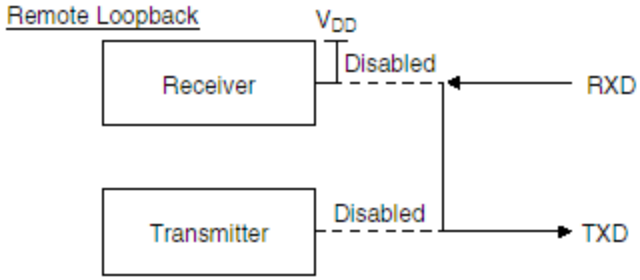
CHMODE		Mode Description
0	0	Normal Mode The USART Channel operates as an Rx/Tx USART.
0	1	Automatic Echo Receiver Data Input is connected to TXD pin.
1	0	Local Loopback Transmitter Output Signal is connected to Receiver Input Signal.
1	1	Remote Loopback RXD pin is internally connected to TXD pin.

USART Mode Register: Channel Modes zum Testen der Schnittstelle

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.

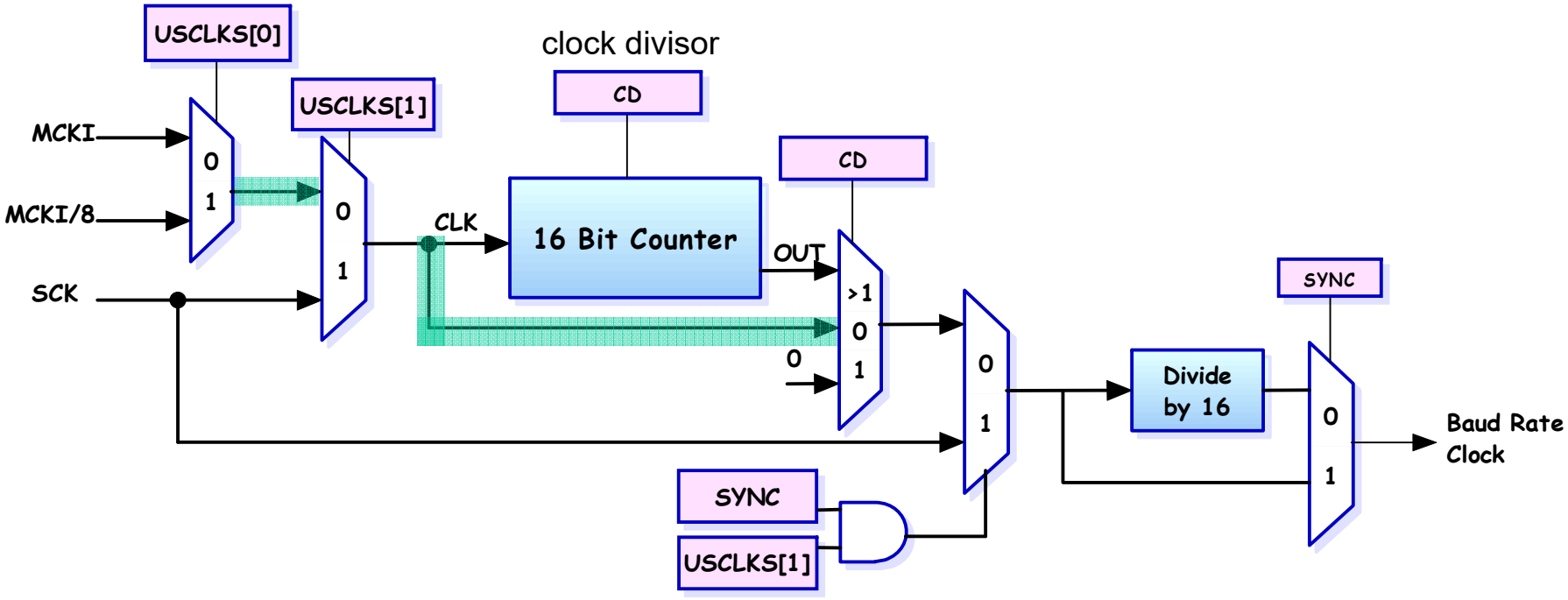






h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Baudraten-Generator



asynchrone Übertragung:

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

synchrone Übertragung:

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

Kombination nicht erlaubt

Initialisierung des Baudraten-Generators

- Generator wird durch MCKI (Machine Clock) oder MCKI/8 getrieben
- Berechnung des Teilerfaktors für asynchrone Kommunikation
  - BRGR = MCKI / (16 * BAUD)
- Beispiel:
  - MCKI = $25*10^6$
  - 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

Serielle Schnittstelle – Code (1)


```
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    // dezimal 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
```

Serielle Schnittstelle – Code (2)


```
serial_init:                // Alternative
        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:
```

Serielle Schnittstelle - Interrupt

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 Receive 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

Abbildung im OSI-Schichtenmodell – Schicht 1

7. Anwendungsschicht (application layer)
6. Darstellungsschicht (presentation layer)
5. Sitzungsschicht (session layer)
4. Transportschicht (transport layer)
3. Vermittlungsschicht (network layer)
2. Sicherungsschicht (data link layer)

1. Bitübertragungsschicht (physical layer), die Spezifikation
   - der Verbindung, dazu gehören Stecker, Leitungen, Repeater, Hubs und Antennen
   - der elektrischen, optisch, elektromagnetischen Signale auf dem Übertragungsmedium
   - der Codierung von Bitfolgen in zu übertragende Signale
   - der Bedeutung der Bits in einer Bitfolge als Schnittstelle zu Schicht 2

Binäre Leitungscodes - Pegelwechsel

Codierung High-active bzw. Low-active:

High-active

- 0V: Lowpegel 5V: Highpegel (andere Spannungspegel je nach Schaltung)

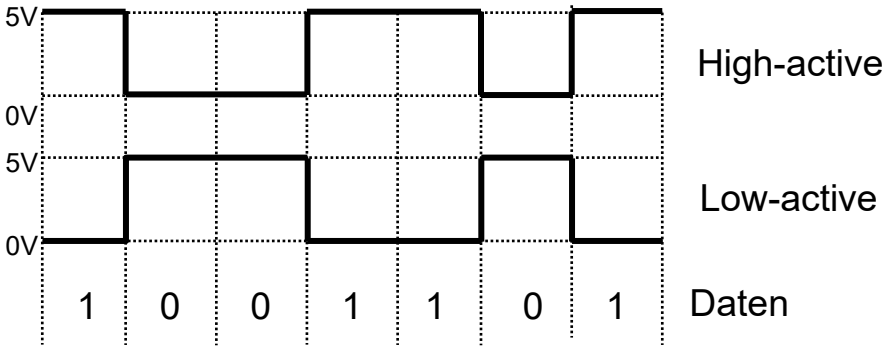
Low-active

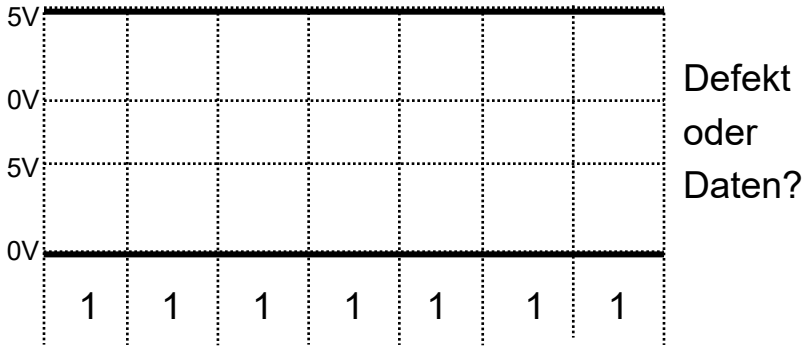
- 0V: Highpegel 5V: Lowpegel (andere Spannungspegel je nach Schaltung)

Probleme bei der Übertragung:

Leitungsfehler wie statische Pegel (0V, 5V) werden nicht erkannt.

Lange Folgen von 0‘en oder 1‘en führen zu keinem Pegelwechsel.





Binäre Leitungscodes - Abbildungen

NRZ (Non Return to Zero):

- 0: Lowpegel 1: Highpegel
- Standard bei asynchroner Übertragung (kein Wechsel bei langen 0/1-Folgen)

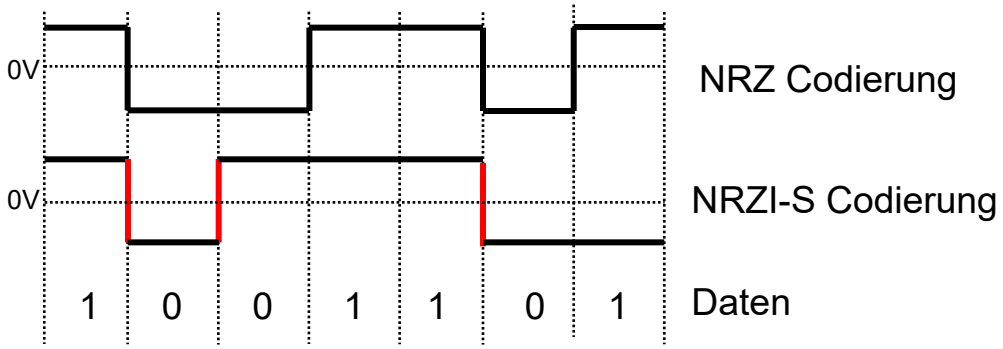
NRZI (NRZ Invert) mit den Varianten NRZI-S und NRZI-M

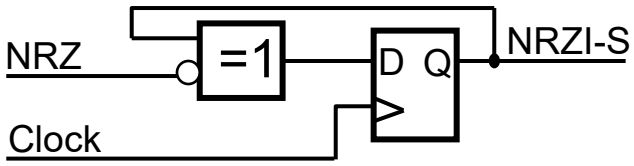
NRZI-S (NRZ Invert - Space):

- Pegelwechsel bei 0, kein Pegelwechsel bei 1
- bessere Synchronisation des Empfängers auf den Bittakt möglich, aber Synchronisationsproblem nicht vollständig behebbar (kein Wechsel bei langen 1-Folgen)
- wird z.B. bei USB verwendet, wenn nur NRZI angegeben ist, ist NRZI-S gemeint

NRZI-M (NRZ Invert - Mark):

- Pegelwechsel bei 1, kein Pegelwechsel bei 0 (kein Wechsel bei langen 0-Folgen)





Umwandlung von NRZ in die NRZI-S Codierung

Binäre Leitungscodes - Beispiele

NRZI-S (NRZ Invert - Space):

- Pegelwechsel (Invertierung) bei 0, kein Pegelwechsel bei 1

Das auf der physikalischen Leitung übertragene Muster ist von dem Ausgangsstatus der Leitung abhängig:

Beispiel 1: Datenbits (logisch):1 1 1 1 1 1 1 1
phys. Leitung bei Ausgangszustand "1":1 1 1 1 1 1 1 1
phys. Leitung bei Ausgangszustand "0":0 0 0 0 0 0 0 0

Beispiel 2: Datenbits (logisch):0 0 0 0 0 0 0 0
phys. Leitung bei Ausgangszustand "1":0 1 0 1 0 1 0 1
phys. Leitung bei Ausgangszustand "0":1 0 1 0 1 0 1 0

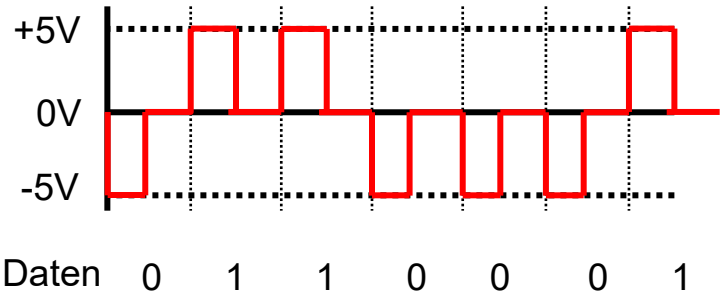
Beispiel 3: Datenbits (logisch):1 1 1 1 1 0 1 0 1 0 1 1 0 0 0 1
phys. Leitung bei Ausgangszustand "1":1 1 1 1 1 0 0 1 1 0 0 0 1 0 1 1
phys. Leitung bei Ausgangszustand "0":0 0 0 0 0 1 1 0 0 1 1 1 0 1 0 0

Quelle: Wikipedia

Ternärer Leitungscode

RZ Code (Return to Zero)

- statt zwei Pegeln (0, 1) gibt es drei zulässige Pegel (-1, 0, 1) auf einer Leitung (negative, positive Spannung und 0V-Pegel)
- Die Daten werden als negativer Pegel (logisch 0) und positiver Pegel (logisch 1) dargestellt
- nach der Hälfte jeder Taktdauer fällt der Pegel auf 0V



Vorteil:
selbsttaktend, der Takt lässt sich aus dem Signal wieder generieren

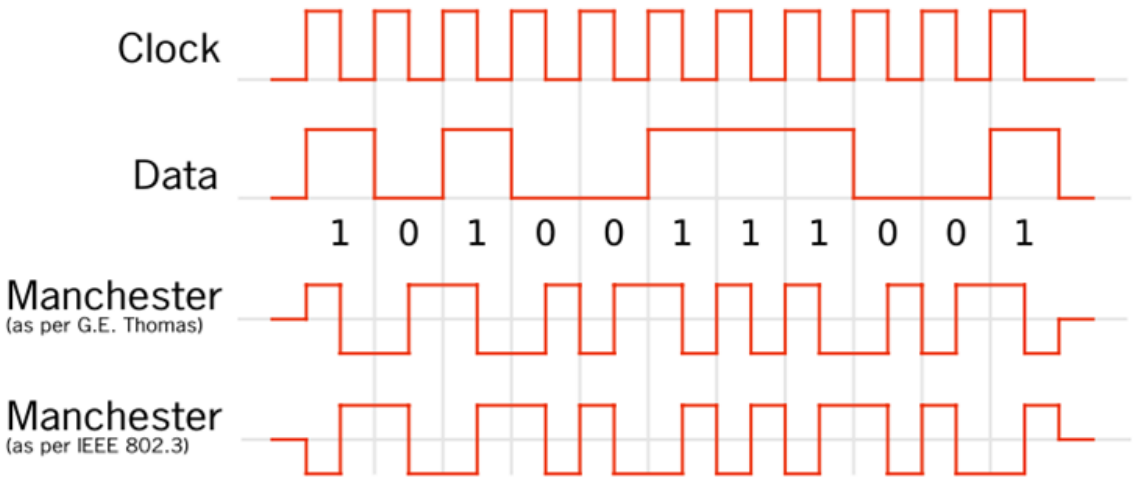
Nachteile:
Baudrate = 2 * Bitrate (doppelte Bandbreite benötigt)
Das Signal hat einen Gleichspannungsanteil bei langen Folgen von 0‘en und 1‘en (keine galvanische Trennung von elektrischen Leitungen möglich, nicht übertragbar als elektromagnetisches Signal)

Binäre Leitungscodes

Manchester Code

- selbsttaktend: Takt- und Dateninformation werden kombiniert
- fallende Flanke logische Null, steigende Flanke logische Eins (IEEE 802.3) (Signal enthält kein Gleichspannungsanteil mehr)
- Baudrate = 2 * Bitrate
- Bitrate (übertragene Daten je Zeiteinheit in Bit pro Sekunde)
- Baudrate (Anzahl der *Zustandsänderungen* des Signals *pro Zeiteinheit)*
- Kodierung und Dekodierung durch eine EXOR – Verknüpfung
- Beginn der Datenübertragung: „Header“ zur Synchronisation
- z.B. 10 MBit-Ethernet

Wenn nur Manchester-Code angegeben ist, ist die Variante nach IEEE 802.3 gemeint.



Quelle: Wikipedia

Sicherung der Datenübertragung

Codesicherungsverfahren:

VRC (Vertical Redundancy Checking, Querparität):

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

Kombinierter Übertragungsfehler wird nicht erkannt (Datenbit und Prüfbit

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

Bitfehler

Fehler-erkennung

Kreuzsicherung:

Kombination von Quer- und Längsparität

Einfachfehler werden erkannt **und korrigiert**

Mehr als ein Bitfehler

- ❑ Korrektur nicht möglich
- ❑ Erkennung sehr wahrscheinlich

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

Beispiel: Dreifachfehler wird nicht erkannt (selten)

								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	1
1	1	1	0	1	0	1	0	1
LRC	1	1	1	0	1	0	0	0	1

Beispiel: Vierfachfehler wird nicht erkannt (selten)

								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	1	0	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 eines gegebenen Generatorpolynoms,

d.h. Generatorpolynom teilt das erzeugte Codewort ohne Rest (Fehlerkriterium auf der Gegenseite)

CRC Algorithmus

Eingabe: Nachricht der Länge k (entsprechend einem Polynom p vom Grad $k-1$),
Generatorpolynom g vom Grad m

Ausgabe: Codewort der Länge $n = k + m$ (entsprechend einem Polynom h vom Grad $n-1$)

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:

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

3. „Addiere“ r zu f:

$$h = f + r = q \cdot g + r + r = q \cdot g$$ Achtung: Operator + ist gleich XOR

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 I

Beispiel:

Gegeben: Nachricht 100101110011101, Generatorpolynom 100111

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

2. Schritt: 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
```

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

Achtung: Operator + ist gleich XOR

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

CRC Algorithmus – Beispiel II

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

2. Schritt: Teile das Ergebnis *f* durch das Generatorpolynom *g*
(Darstellung inklusive aller rechten Ziffern)

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

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

Achtung: Operator + ist gleich XOR

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

3. Schritt: Addiere *r* zu *f*. Das Ergebnis entspricht dem gesuchten Codewort

Ergebnis: 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 1 0 1 1 0

CRC Beispiel - Senden und Überprüfen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Generieren beim Senden

Daten	10011
Polynom	110101
f	1001100000
h	1001110011

Restbildung

```
1001100000
110101
0100110000
 110101
 010011000
  110101
  01001100
   110101
   0100110
    110101
    010011 Rest
```

Überprüfen beim Empfangen

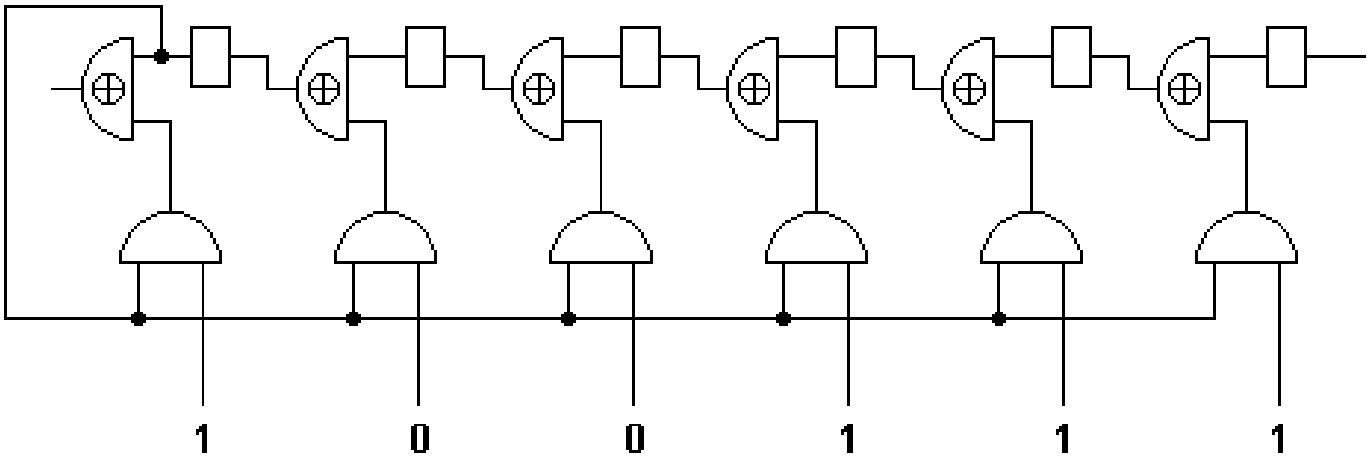
h	1001110011
Polynom	110101
Rest	0 (korrekt)

Restbildung

```
1001110011
110101
0100100011
 110101
 010001011
  110101
  01011111
   110101
   0110101
    110101
    000000
```

CRC Implementierung in Hardware

Polynomdivision lässt sich einfach in Hardware realisieren

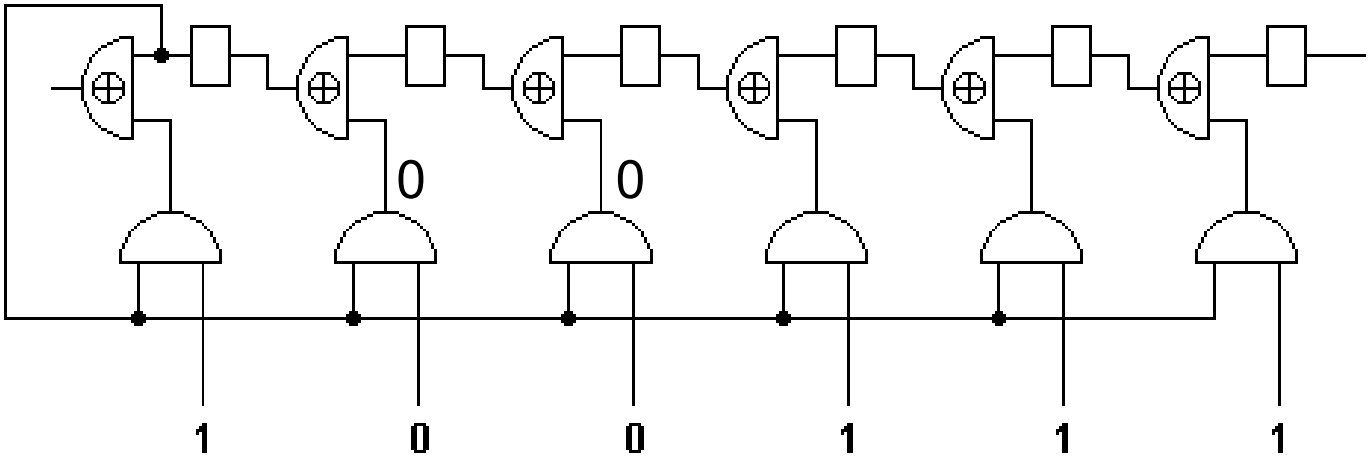


Vorgehensweise im Schieberegister:

- von rechts wird das zu dividierende Polynom h hineingeschoben
- bei einer 1 vorne wird das Generatorpolynom mit dieser 1 multipliziert
- Generatorpolynom wird subtrahiert (XOR 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

Polynomdivision lässt sich einfach in Hardware realisieren

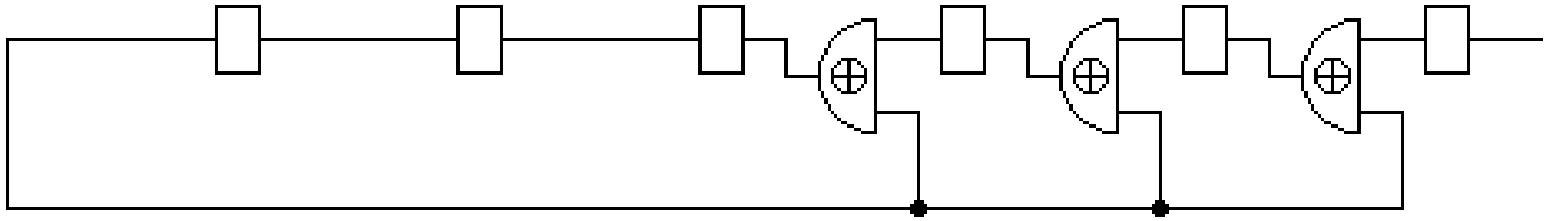


Wie lassen sich die Stellen vereinfachen, bei denen das Generatorpolynom 0 ist?

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): $x^{16} + x^{15} + x^2 + 1 = (x+1)(x^{15} + x + 1)$

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 XOR

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

CRC Erkennen von Fehlern

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Generieren beim Senden

Daten	10011
Polynom	110101
f	1001100000
h	1001110011

Restbildung

1001100000
110101
0100110000
110101
010011000
110101
01001100
110101
0100110
110101
010011 Rest

Überprüfen beim Empfangen

h	1001100011
Polynom	110101
Rest	010000 (Fehler)

Restbildung

1001100011
110101
0100110011
110101
010011011
110101
01001111
110101
0100101
110101
010000

CRC Nichterkennung von Fehlern

Nicht erkennbare Fehler

Daten	10011
Polynom g (m=5)	110101
f	1001100000
f*	10011000000
r	10011
h	1001110011
h*	10011010011

Beispiel: h* enthält eine zusätzliche (eingeschobene) 0 als Fehler

Auffällig: die Daten entsprechen dem Rest r um 5 oder 6 Stellen verschoben in h bzw. h* ($f = rx^5$ bzw. $f^* = rx^6$)

h*	10011010011
Polynom	110101
Rest	0 (fehlerhaft korrekt)

Restbildung

```
10011010011
110101
01001110011
 110101
 0100100011
  110101
  010001011
   110101
   01011111
    110101
    0110101
     110101
     000000
```

CRC Nichterkennung von Fehlern

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Daten	10011
Polynom g (m=5)	110101
f	1001100000
f*	1001100000
r	10011
h	1001110011
h*	1001101011

Polynomdivision g/r

```
110101 : 10011 = 11
10011
010011
 10011
 00000
```

g = r (x+1) = $(x^4+x+1)(x+1)$

$h = rx^5 + r = r(x^5+1)$

Polynomdivision $(x^5+1)/(x+1)$

```
100001 : 11 = 11111
11
010001
 11
 01001
  11
  0101
   11
   011
    11
    00
```

h = r (x+1) $(x^4+x^3+x^2+x^1+1)$

$h^* = rx^6 + r = r(x^6+1)$

Polynomdivision $(x^6+1)/(x+1)$

```
1000001 : 11 = 111111
11
0100001
 11
 010001
  11
  01001
   11
   0101
    11
    011
     11
     00
```

h*= r (x+1) $(x^5+x^4+x^3+x^2+x^1+1)$

Fehler sind nicht erkennbar, wenn der Fehler ein Vielfaches des Generatorpolynoms ist.

CRC Eigenschaften

- CRC erkennt nicht alle möglichen Fehler.
- Ein eingeschobenes Bit tritt bei Übertragungen nicht auf, die einen festen Rahmen mit vorgegebener Bitanzahl für Datenpakete verwenden. Nicht erkennbare Fehler sollten für das jeweilige Übertragungsmedium und die darauf aufbauende Sicherungsschicht möglichst selten sein.
- Wenn weitere Eigenschaften des Übertragungsmediums insbesondere häufige Fehlerklassen bekannt sind, lässt sich dies bei der Wahl der Implementierung (des Generatorpolynoms) berücksichtigen.
- Die Generatorpolynome werden nach günstigen Eigenschaften in Bezug auf oft auftretende Fehler ausgesucht.

CRC Beispiele

Bezeichnung	Polynom	Anwendung (Auswahl)
CRC-5	x^5+x^2+1	USB Token
CRC-16	$x^{16}+x^{15}+x^2+1$	USB-Daten
CRC16-CCITT	$x^{16}+x^{12}+x^5+1$	X.25 (OSI) Bluetooth
CRC16-DECT	$x^9+x^8+x^7+x^3+1$	Mobilteile für Telefon
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$	Internet (IEEE 802.3), USB3-Daten

CRC Eigenschaften

Mit dem CRC-16-Verfahren können

- bis zu 16 aufeinanderfolgende Bitfolgen sicher erkannt werden
- längere Bitfehlerfolgen mit der Sicherheit von 99,997% erkannt werden
- durch den enthaltenen Faktor (x+1) werden alle Fehler mit einer ungeraden Anzahl von Fehlerstellen erkennbar (Paritätsprüfung).

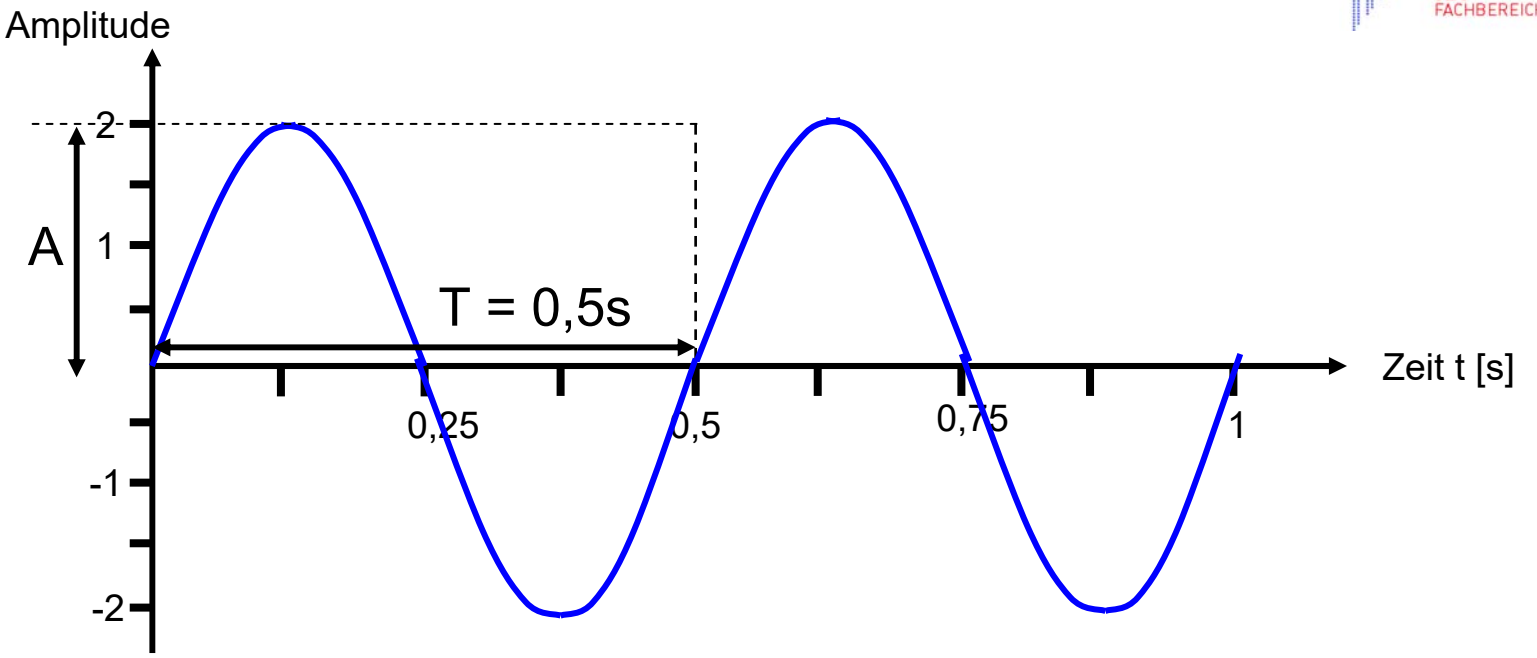
Diskutieren Sie folgende Fragestellung:

Vergleichen Sie einen Ethernet-Netzwerktreiber, der die Sicherung der Datenübertragung in Software implementiert mit einem Treiber für eine Netzwerkkarte, in der die Sicherung bereits in Hardware implementiert vorliegt.

Gliederung

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

Kreisfunktion im Zeitbereich



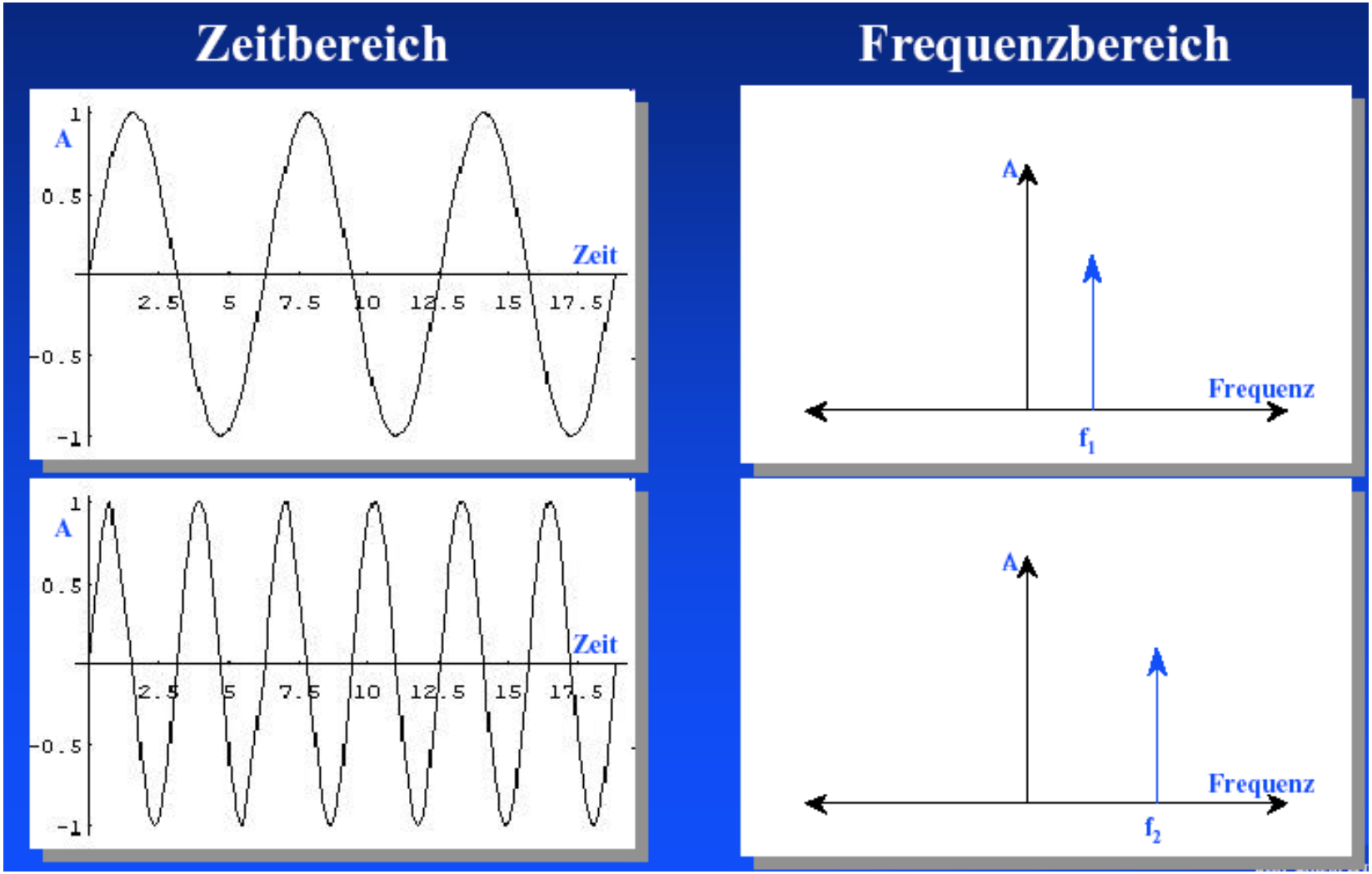
Dargestellte Kreisfunktion: Periodendauer T = 0,5s

$$y(t) = 2\sin\left(2\pi \cdot \frac{t}{T}\right) = 2\sin(2\pi \cdot t \cdot \underbrace{2s^{-1}}_{\text{Frequenz } f = \frac{1}{T}\,[Hz]}) = 2\sin(4\pi \cdot t\, s^{-1}) = 2\sin(4\pi \cdot t\, Hz)$$

Allgemein: $y(t) = A\sin(2\pi f t)$

Mit der Kreisfrequenz $\omega = 2\pi f$ auch oft geschrieben als $y(t) = A\sin(\omega t)$

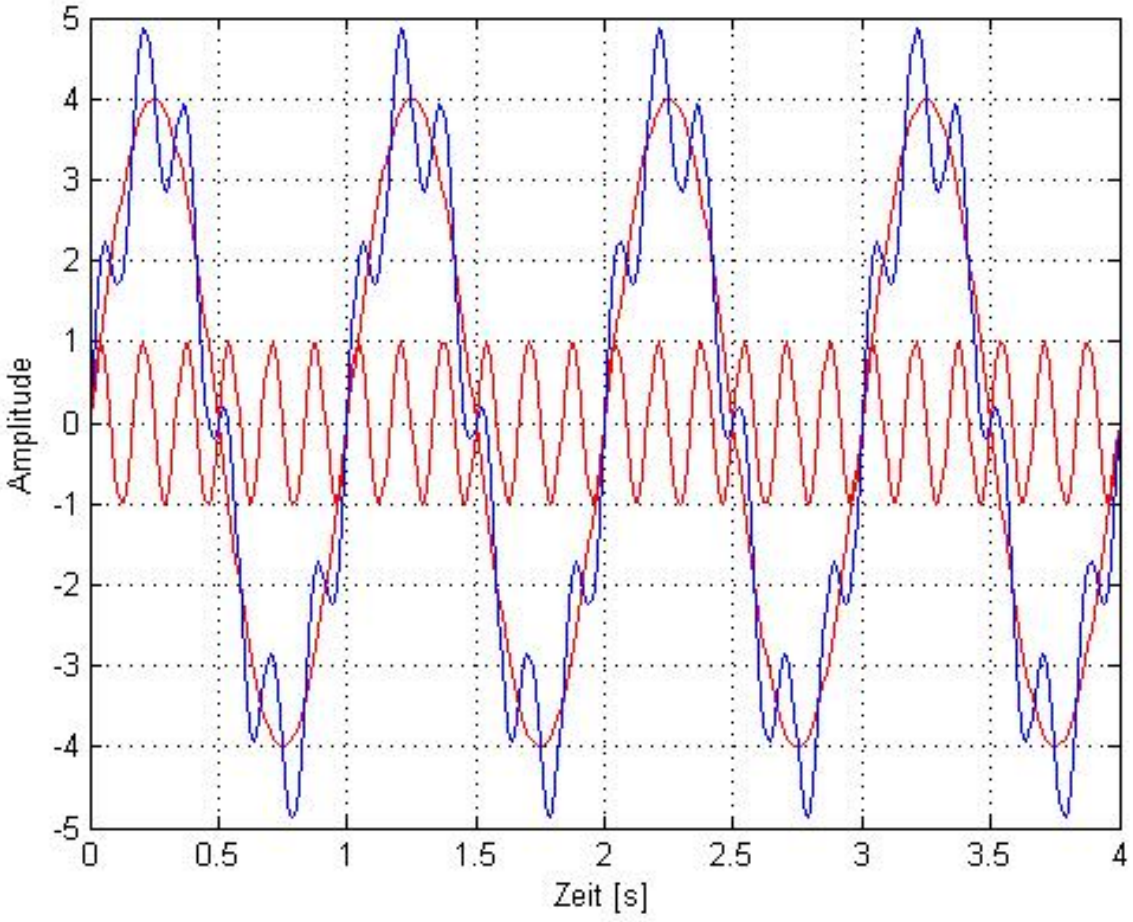
Zeit- und Frequenzbereich



Zeit- und Frequenzbereich

Beispiel einer überlagerten Funktion:

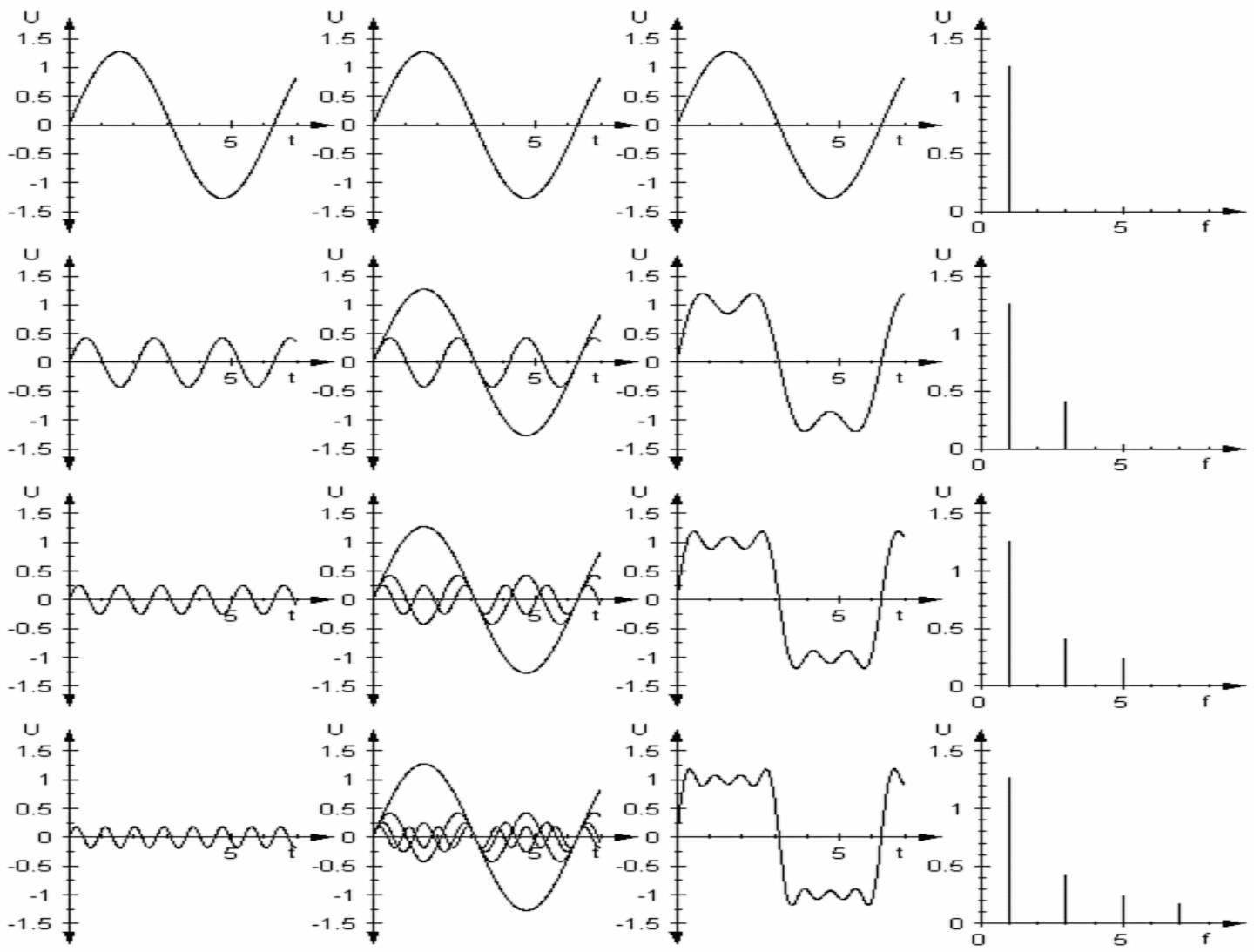
$y(t) = 4\sin(2\pi \cdot t\ s^{-1}) + \sin(12\pi \cdot t\ s^{-1})$



Vergleiche Beispiele auf
http://www.falstad.com/fourier/e-index.html

Zeit- und Frequenzbereich

Beispiel einer Rechteckfunktion: (Quelle Wikipedia)



Zeit- und Frequenzbereich

Eine periodische (oder periodisch fortsetzbare) Funktion lässt sich durch eine Reihe von Sinus- und Kosinusfunktionen darstellen, deren Frequenzen ganzzahlige Vielfache der Grundfrequenz sind:

(Quelle Wikipedia)

$$f_n(t) = \frac{a_0}{2} + \sum_{k=1}^{n} (a_k \cdot \cos(k\omega t) + b_k \cdot \sin(k\omega t)).$$

$$\omega = 2\pi \cdot f$$

Die **Kreisfrequenz** ω ist definiert als das 2π-fache der Frequenz f eines periodischen Vorgangs.

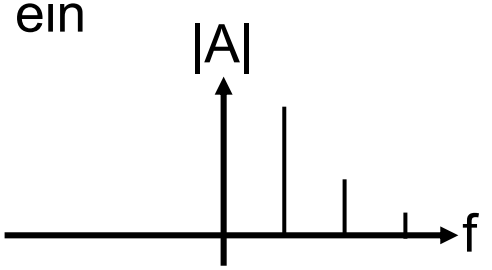
a_k = 0 für alle k gilt, falls ungerade ist,

b_k = 0 für alle k gilt, falls gerade ist

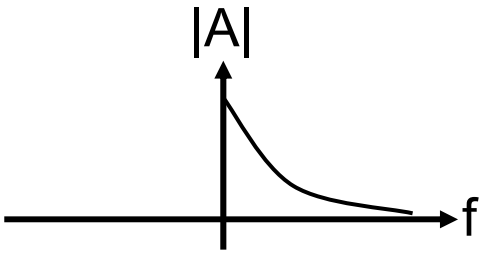
Zeit- und Frequenzbereich

Die Abbildung des Zeit- in den Frequenzbereich erfolgt durch die **Fourier-Transformation**. Man spricht dabei auch von der **Fourier-Analyse**.

Ein **periodisches Signal** bestehend aus einer Grundfrequenz und deren ganzzahlige Vielfache besitzt ein diskretes Spektrum. Im Frequenzbereich bilden Einzelfrequenzen eine **Fourier-Reihe** (diskrete Fourier-Transformation mittels Zahlenreihen und in Form von nummerischen Algorithmen).



Ein **nicht-periodisches Signal** besteht aus einem kontinuierlichem Spektrum an Frequenzen (kontinuierliche Fourier-Transformation mittels Infinitesimalrechnung).



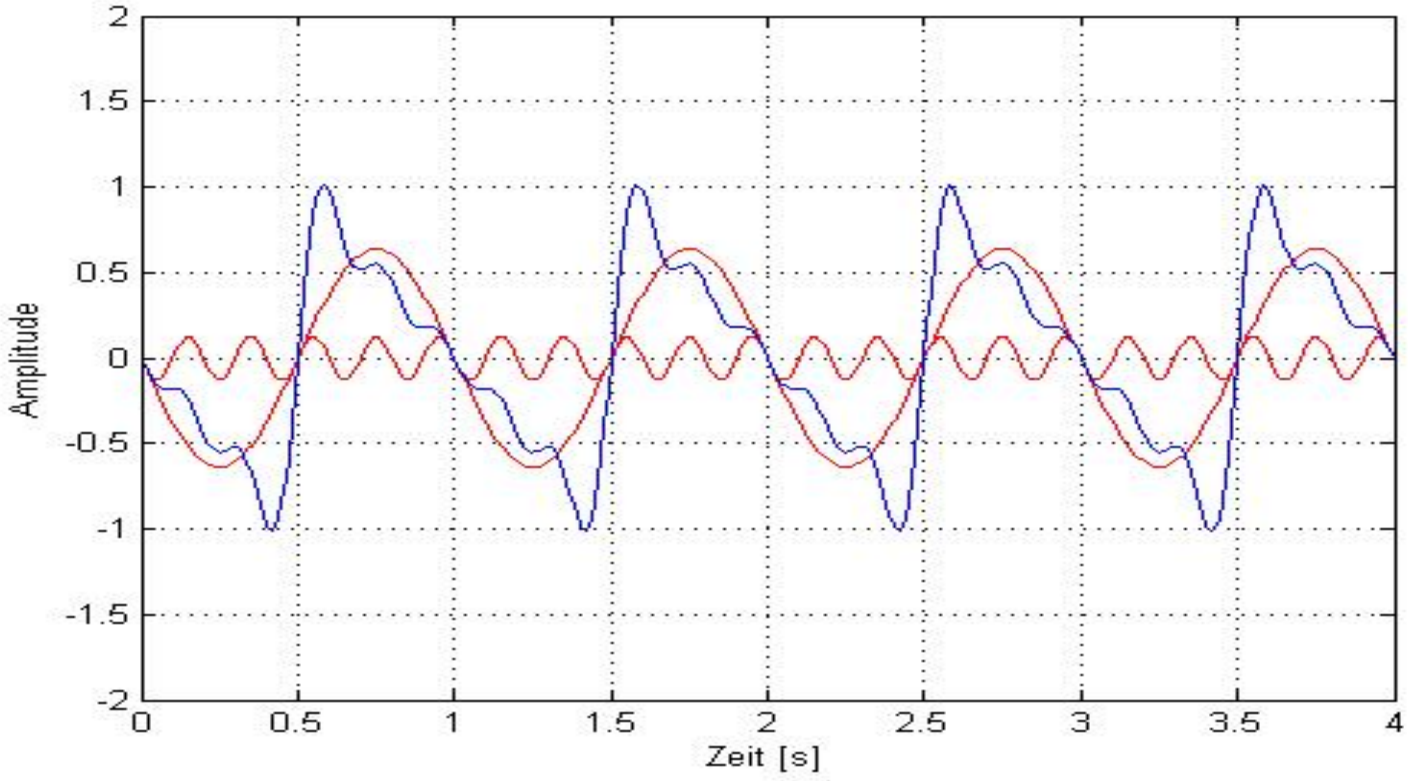
Umkehrfunktionen der Fourier-Transformation erlauben die **Fourier-Synthese** von Signalen aus ihrem Spektrum.

Zeit- und Frequenzbereich

Beispiel einer Sägezahnfunktion: (Quelle Wikipedia)

$$f(t) = -\frac{2h}{\pi}\left[\sin\omega t - \frac{1}{2}\sin 2\omega t + \frac{1}{3}\sin 3\omega t \mp \cdots\right]$$

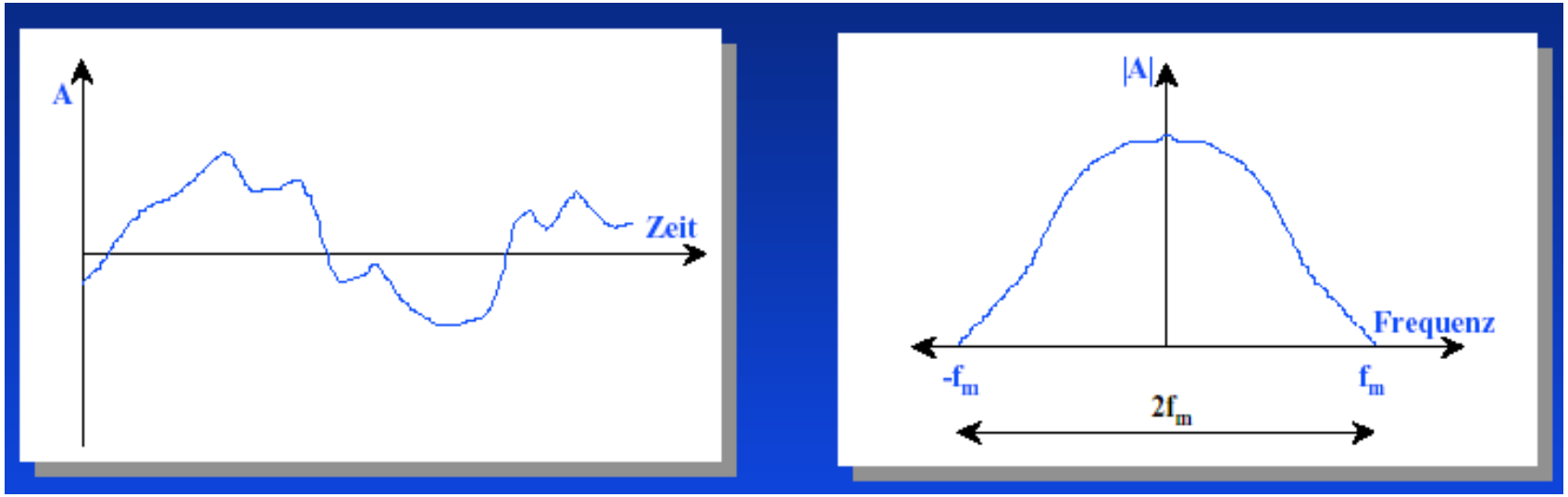
$$= -\frac{2h}{\pi}\sum_{k=1}^{\infty}(-1)^{k-1}\frac{\sin k\omega t}{k}$$



h = 1,

k = 1,…,5

Reale Signale

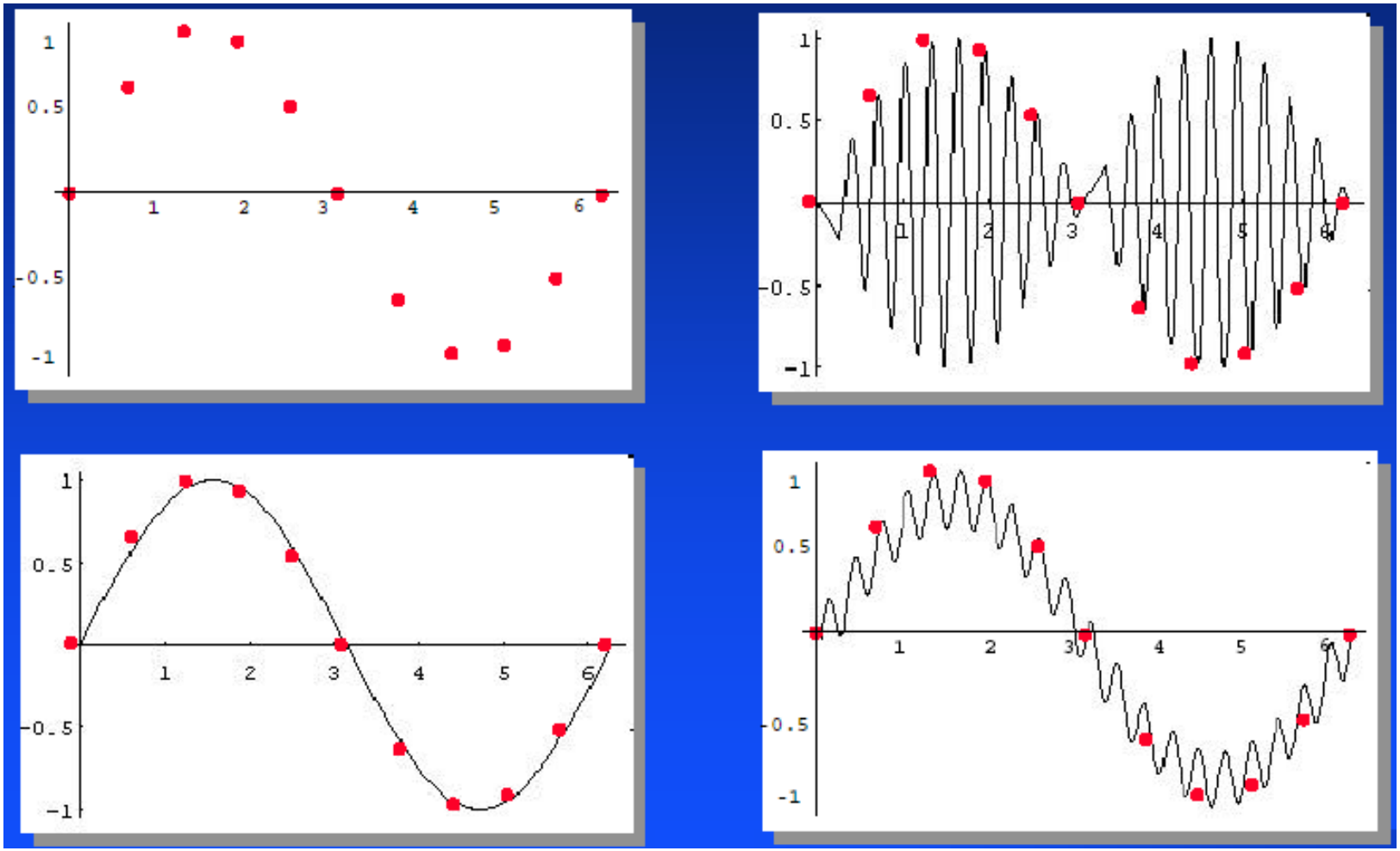


- reale analoge Signale sind kontinuierliche Signale
- reale Signale sind Mischungen vieler Frequenzen
- sie besitzen eine Bandbreite $2*f_m$
- Frequenzspektrum bezeichnet alle enthaltenen Frequenzen

Fehlende Information

Rekonstruktion des Signals nach Abtastung ?

Wahl der Abtastfrequenz ist wichtig



Abtasttheorem

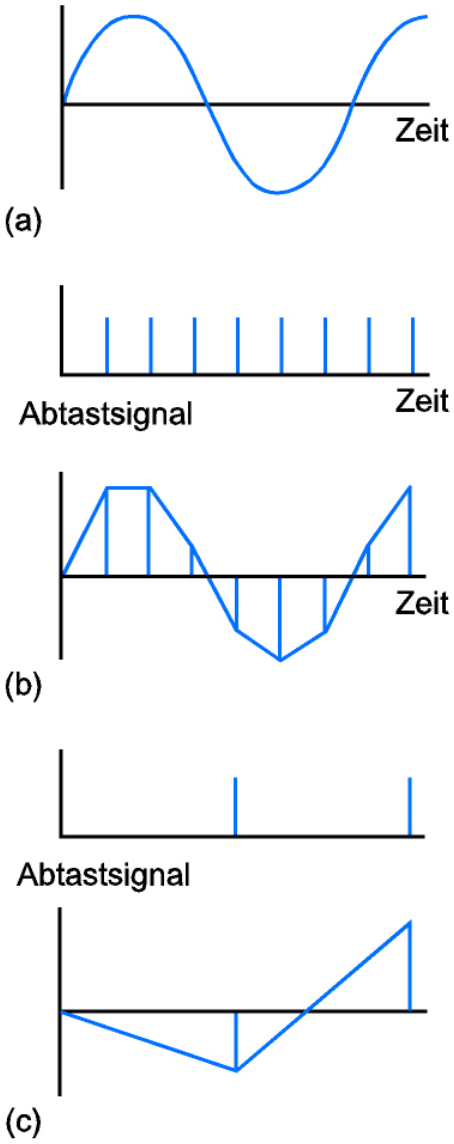
Abtasttheorem von Shannon (Nyquist Kriterium):

Rekonstruktion des Signals in der ursprünglichen Form nur möglich, wenn Abtastfrequenz f_s mehr als doppelt so gross ist, wie die grösste Frequenz f_{max} im abgetasteten Signal

$f_s > 2 * f_{max}$

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

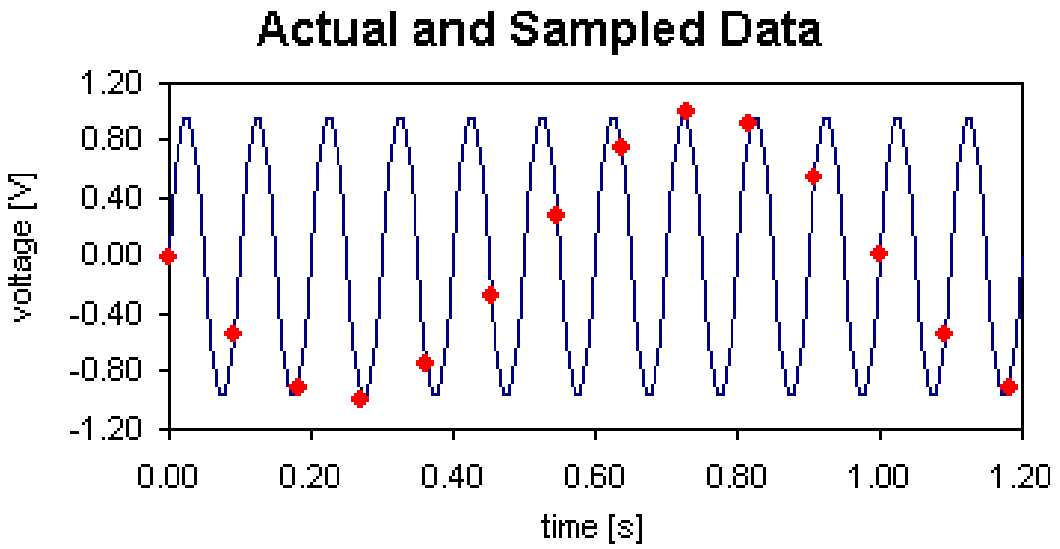
-> Aliasing (dt. Pseudonymisierung)



Problem Aliasing

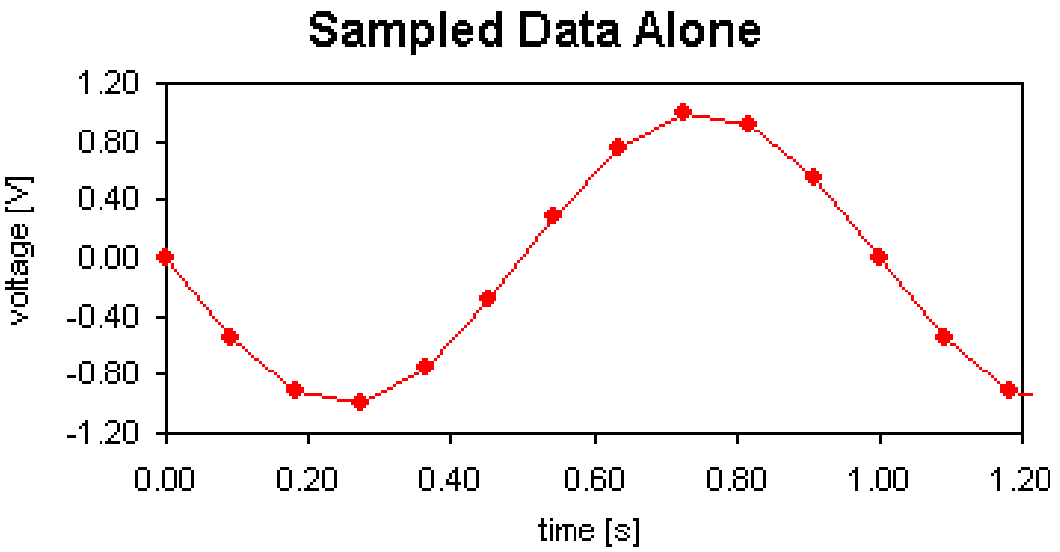
10 Hz Signal

Abtastung: 11 Hz



Ergebnis:

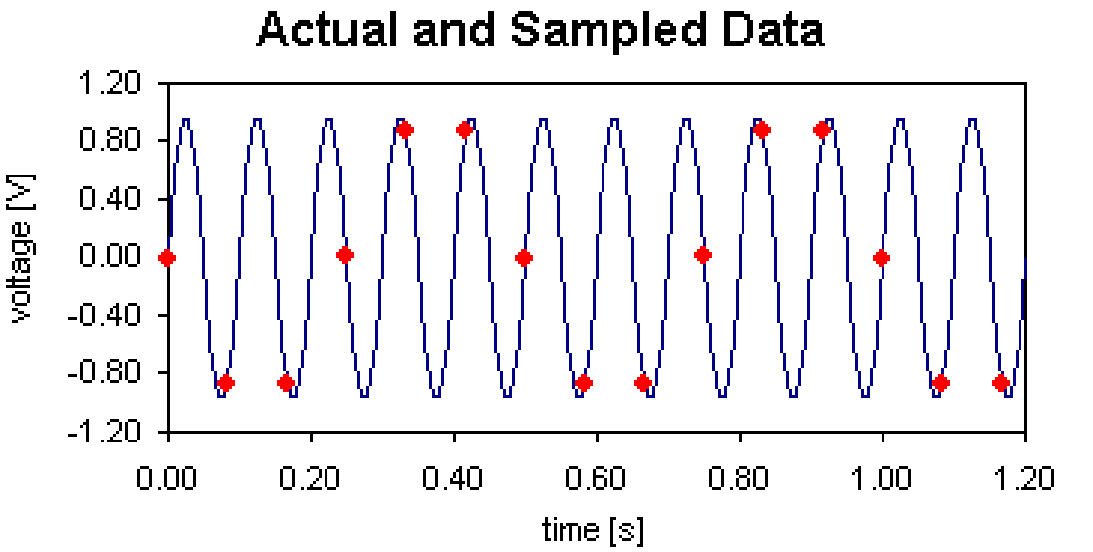
1 Hz (?!)



Problem Aliasing

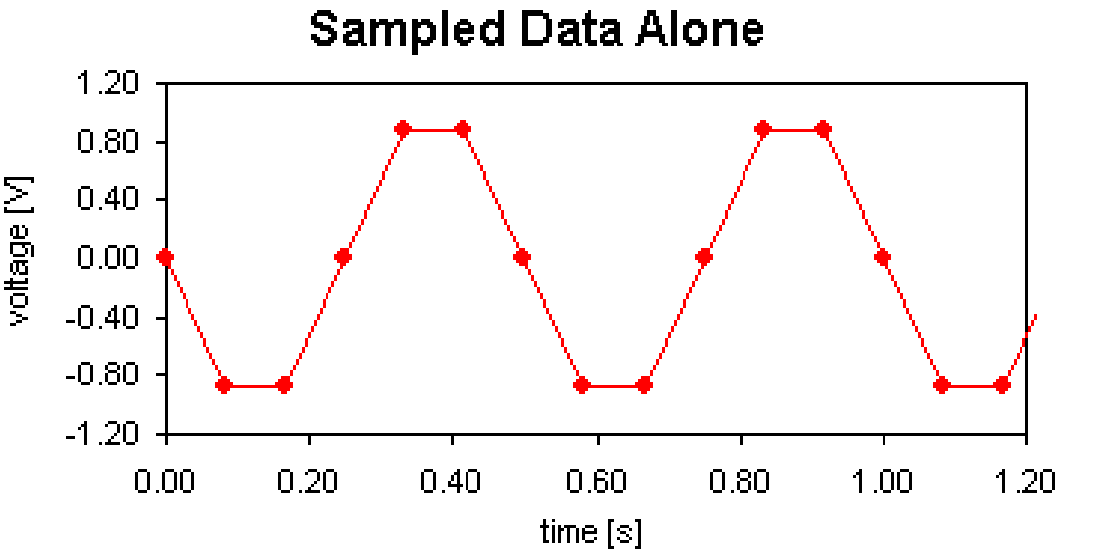
10 Hz Signal

Abtastung: 12 Hz



Ergebnis:

2 Hz (?!)



Problem Aliasing

10 Hz Signal

Abtastung: 5 Hz

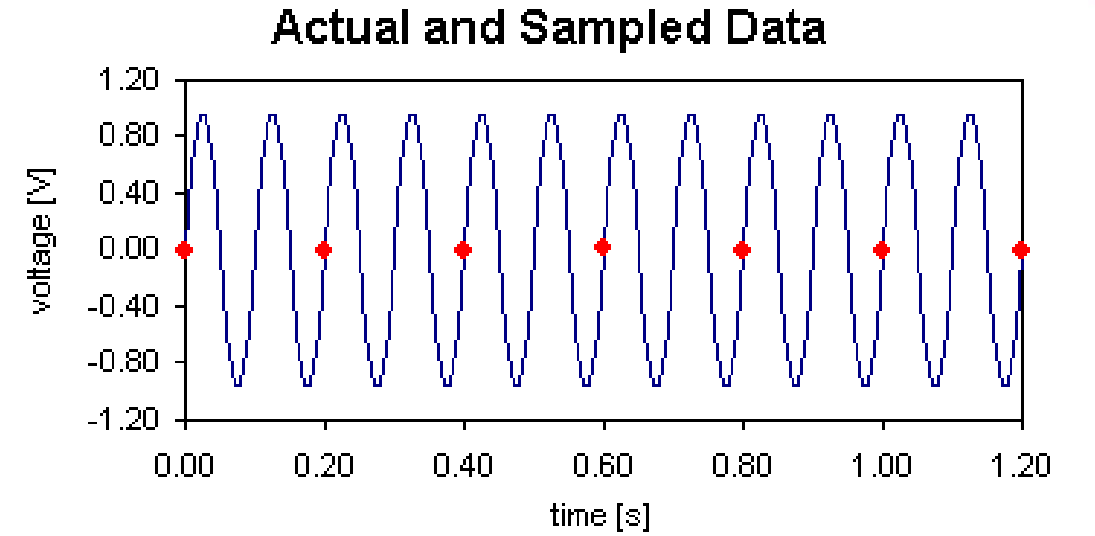
Achtung:

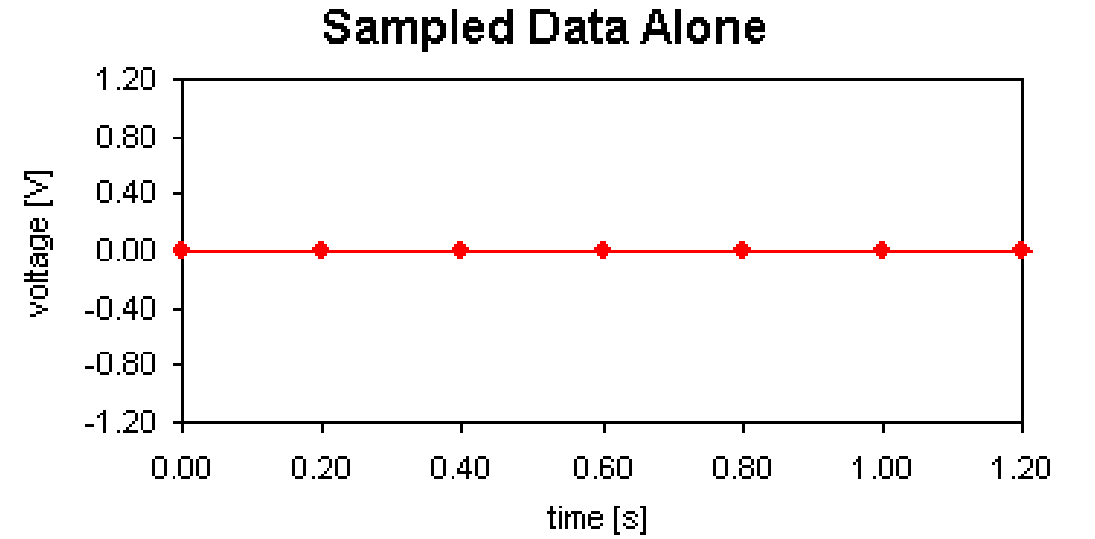
$f_{sample} = f_{signal} / N$

hier N = 2

Ergebnis:

0 Hz (?!)

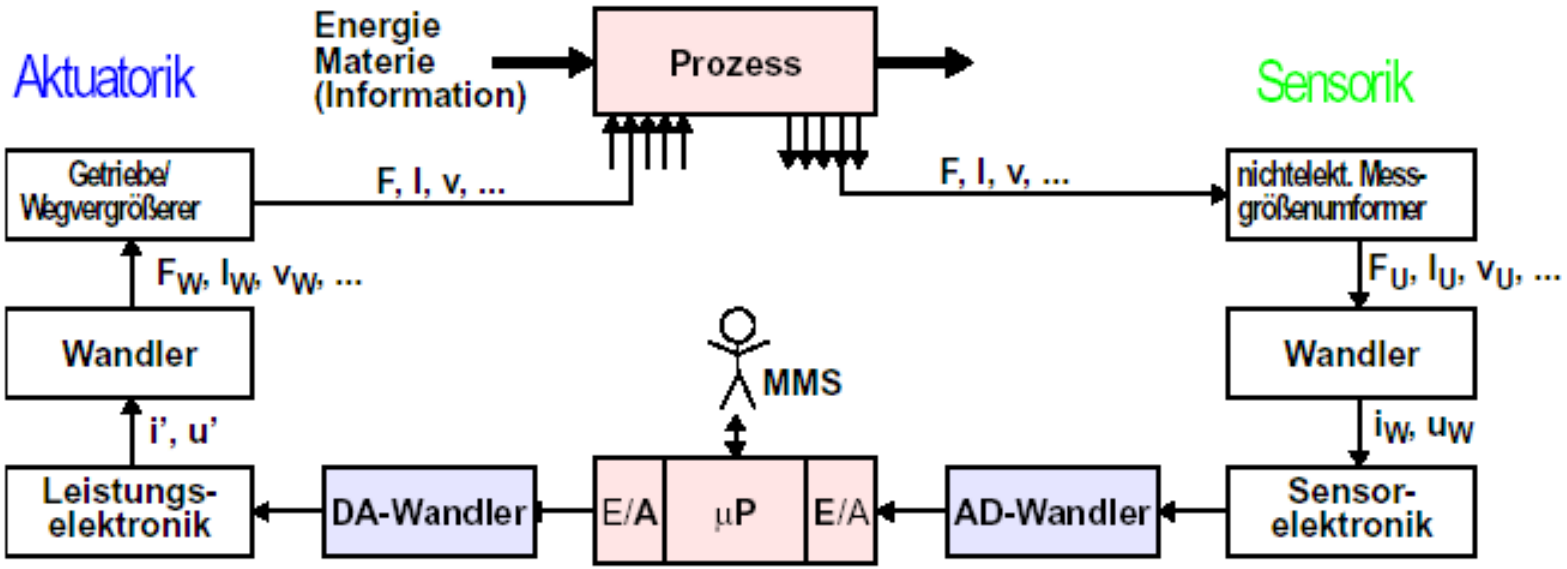




Signalverarbeitung

Mikroprozessor wird oft eingesetzt in einer

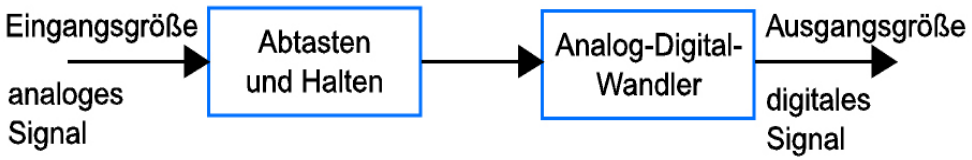
- Regelung
- Prozesssteuerung

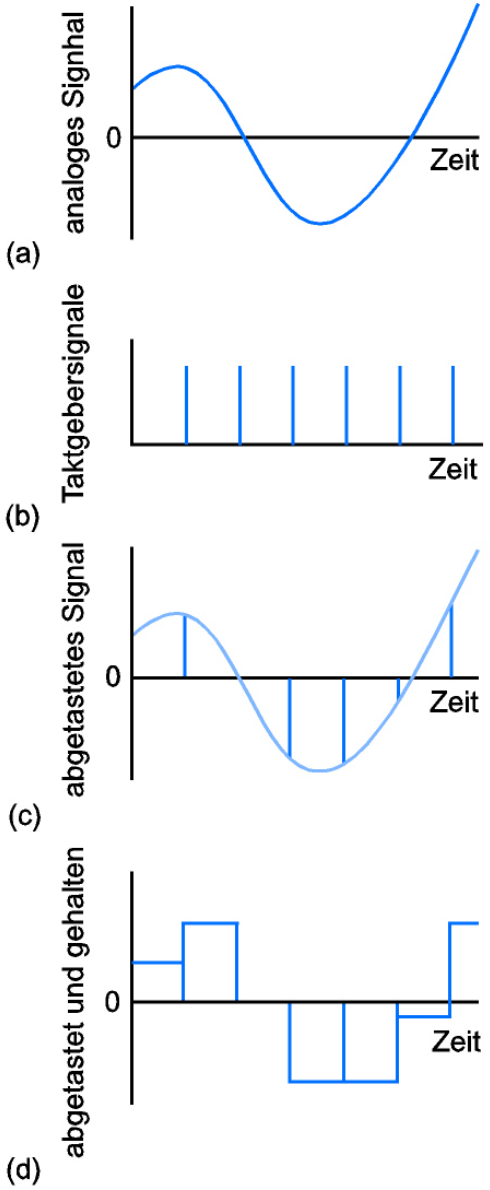


Analog-Digital Wandlung (ADC)

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

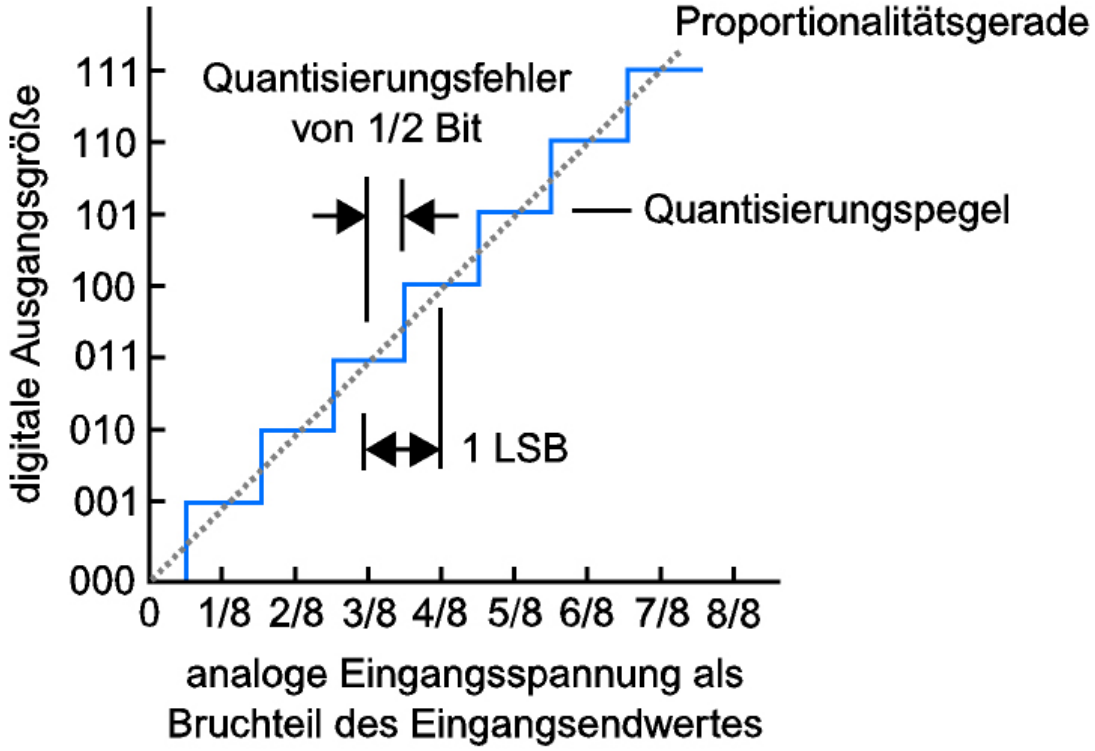




Analog-Digital Wandlung

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

Beispiel: 3 Bit Auflösung



Analog-Digital Wandlung

Allgemein: Wortlänge n Bits

Die 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

Analoge Signale

Datenvolumen Beispiel: 1 min Audiosignal

- Hifi-Qualität

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

(hörbarer Bereich bis ca. 20 kHz)

- 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

Zählverfahren

- der kleinste gewünschte Schritt (LSB) aufeinander addiert und an einen Komparator geliefert, bis der Wert gleich oder größer der angelegten analogen Referenzgröße ist.
- Schritte werden mit einem Zähler erzeugt
- geringer Schaltungsaufwand
- Umsetzungszeit abhängig von der Eingangsgröße

Wägeverfahren

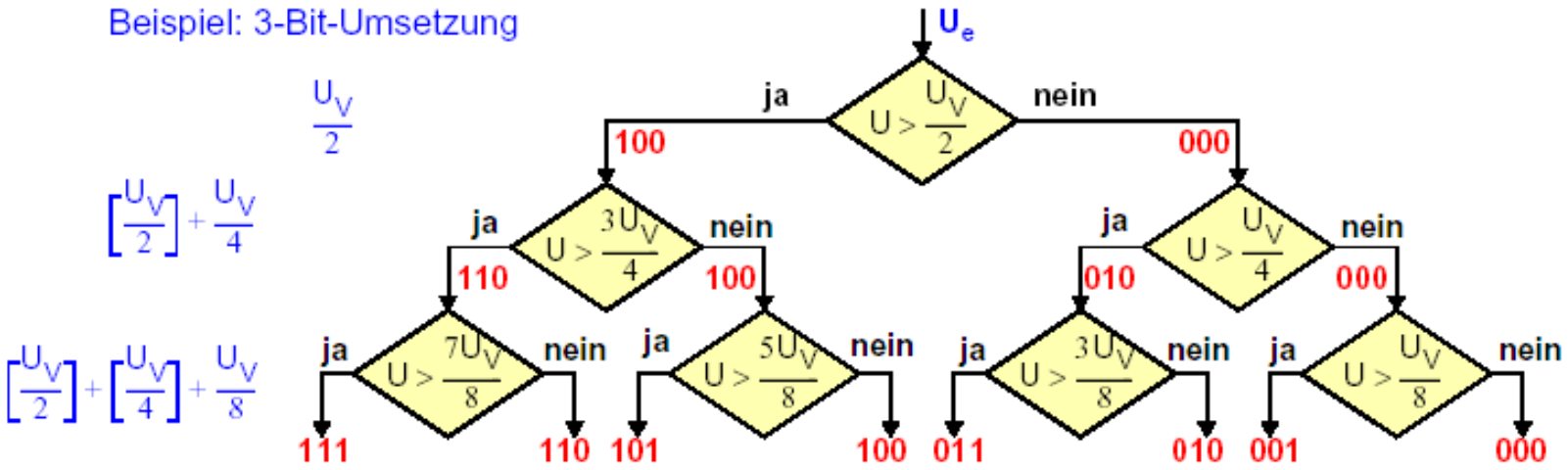
- bitserieller Vergleich, beginnend mit MSB
- analog zum Abwiegen eines unbekannten Gewichts mittels Balkenwaage

Analog-Digital Wandlung

Wägeverfahren

- bitserieller Vergleich, beginnend mit MSB

  analog zum Abwiegen eines unbekannten Gewichts mittels Balkenwaage

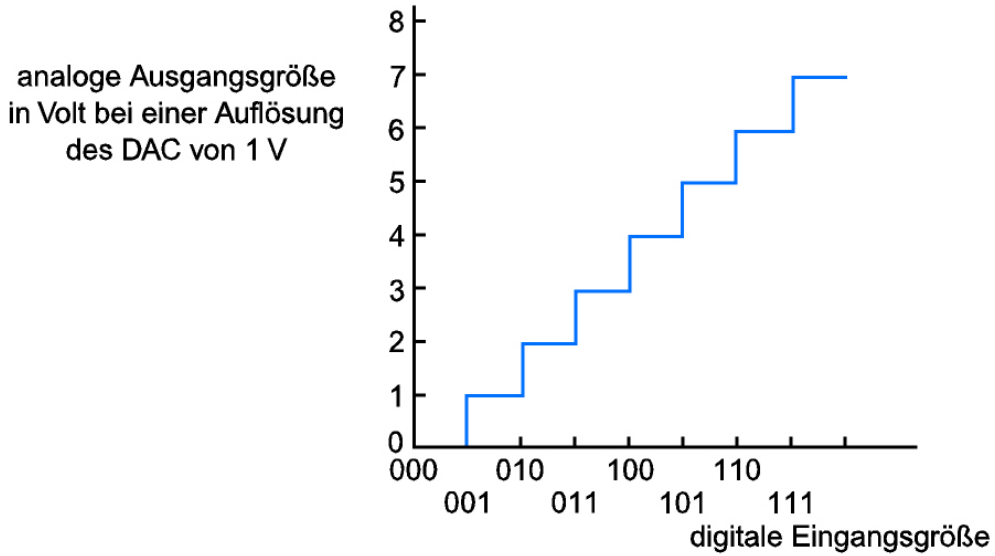


Gliederung

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

Digital-Analog Wandlung (DAC)

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



Beispiel: Regelventil (vollständige Öffnung 6 V)

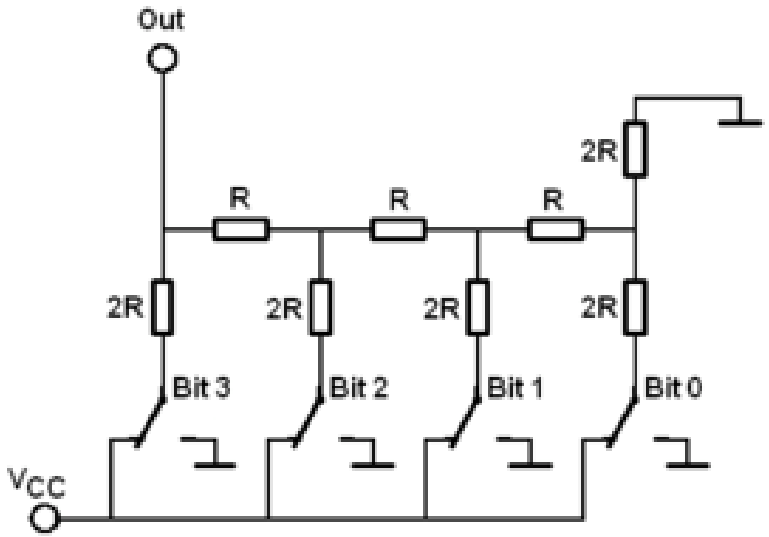
8 Bit Digital-Analog Wandlung

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

Digital-Analog Wandlung (DAC)

- Direktes Verfahren
  - Alle möglichen Ausgangswerte bereitgestellt (Spannungsteiler)
  - Direkt mit dem digitalen Wert über einen 1-aus-*n* Schalter (Multiplexer) ausgewählt.
  - Schnellste und aufwändigste Verfahren
  - Einsatz nur in Umsetzern mit sehr wenigen Bits

- Paralleles Verfahren
  - Analoges Ausgangssignal durch je einen parallel geschalteten Widerstand pro Bit erzeugt, der je nach Bit gewichtet ist.
  - Verwendung eines R2R-Netzwerk
  - Man benötigt so viele Schalter, wie Bits zur Darstellung der digitalen Werte verwendet werden.

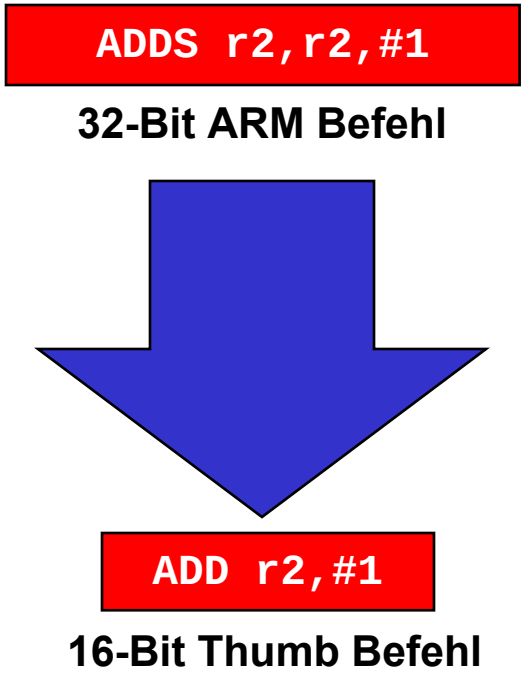


Gliederung

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

Thumb Befehlssatz

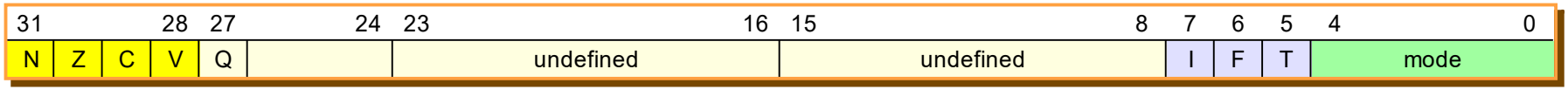
- Thumb Modus ist ein 16-Bit Befehlssatz
  - Optimiert auf Codedichte (~65% der ARM Codegrösse)
  - Teilmenge der Funktionalität des ARM Befehlssatzes
- Prozessorkern verfügt über zusätzlichen Ausführungsmodus - Thumb
  - Umschalten zwischen ARM und Thumb durch **BX** Befehl
  - geringere Ausführungsgeschwindigkeit (häufiges Entleeren der PL)



Eigenschaften von Compiler generiertem Code:

- keine bedingte Befehlsausführung (4)
- Quell- und Zielregister sind identisch (4)
- I.d.R. werden niedrige Register benutzt (1)
- Konstanten haben geringere Grösse
- Inline Barrel Shifter wird nicht benutzt (7)

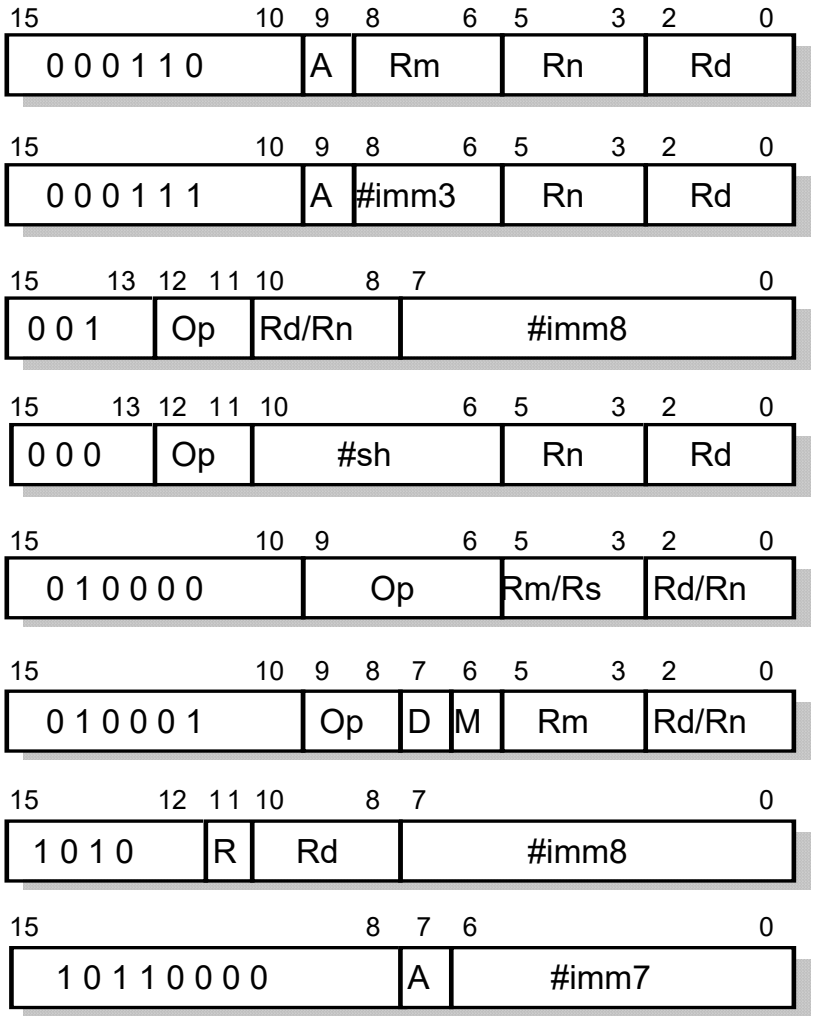
Thumb Befehlssatz



- T Bit (Bit 5 in CPSR)
  - T = 0, Prozessor in Arm State
  - T = 1, Prozessor in Thumb State
- Wenn T = 1, interpretiert der Prozessor den Befehlsstrom als 16 Bit-Thumb-Befehle
- Bei 16-Speicherbausteinen kann jeder Befehl in einem Zyklus geholt werden, während sonst ein 32-Bitbefehl zwei Holzyklen benötigt.

Thumb Befehlssatz

- Datenverarbeitende Befehle (Binärkodierung)



(1) `ADD|SUB Rd,Rn,Rm`

(2) `ADD|SUB Rd,Rn,#imm3`

(3) `<Op> R d/Rn ,#imm8`

(4) `LSL|LSR|ASR Rd,Rn,#shift`

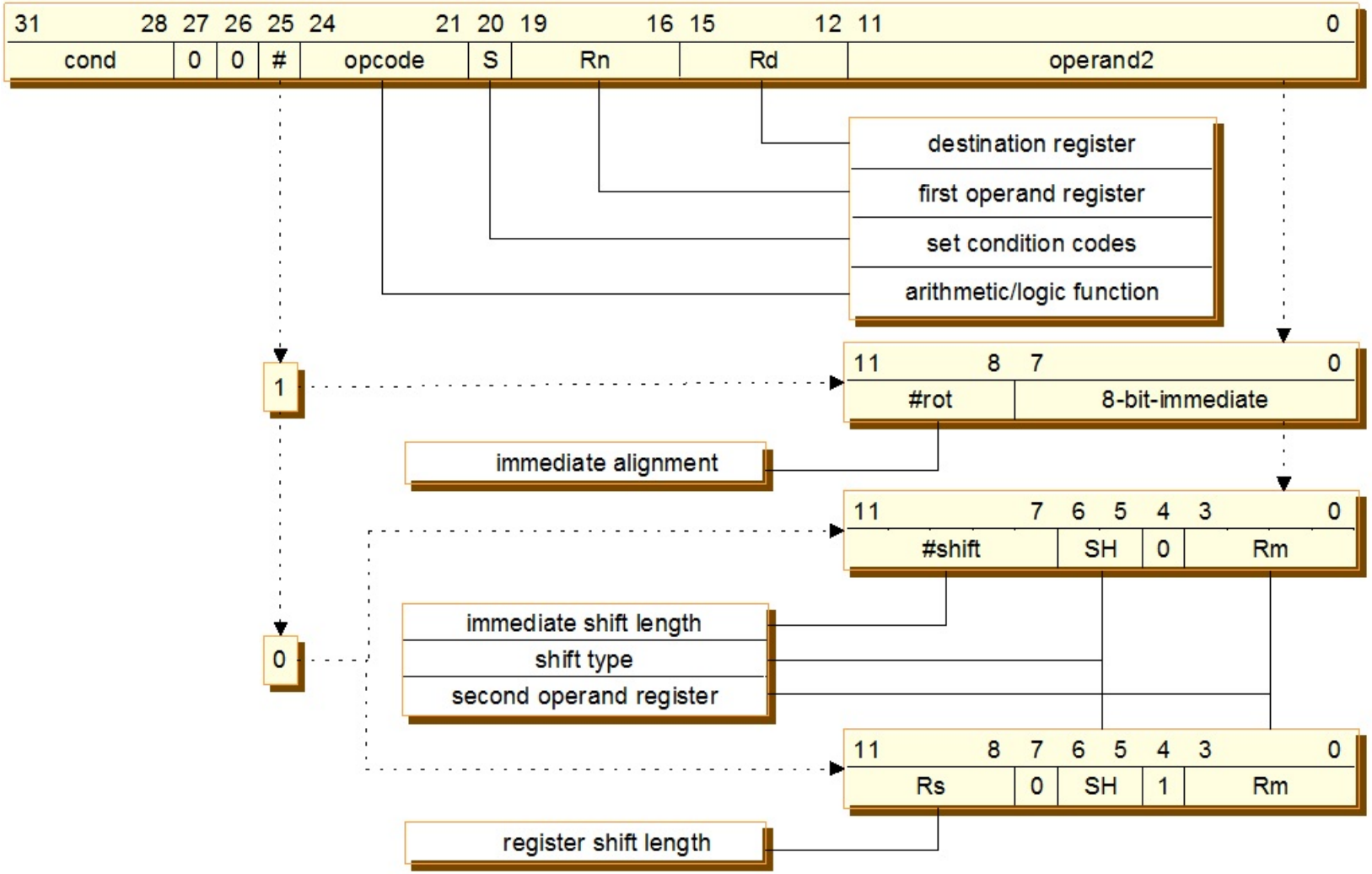
(5) `<Op> Rd/Rn,Rm/Rs`

(6) `ADD|CMP|MOV Rd/Rn,Rm`

(7) `ADD Rd,SP|PC,#imm8`

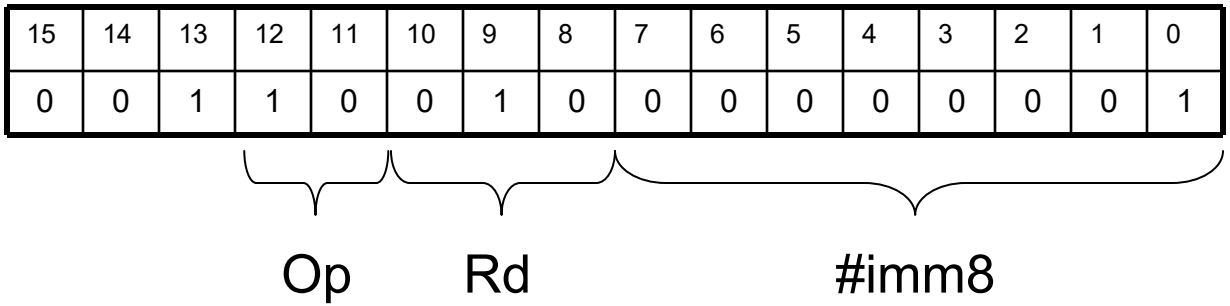
(8) `ADD|SUB SP,SP,#imm7`

Vergleich zum ARM-Befehlsformat



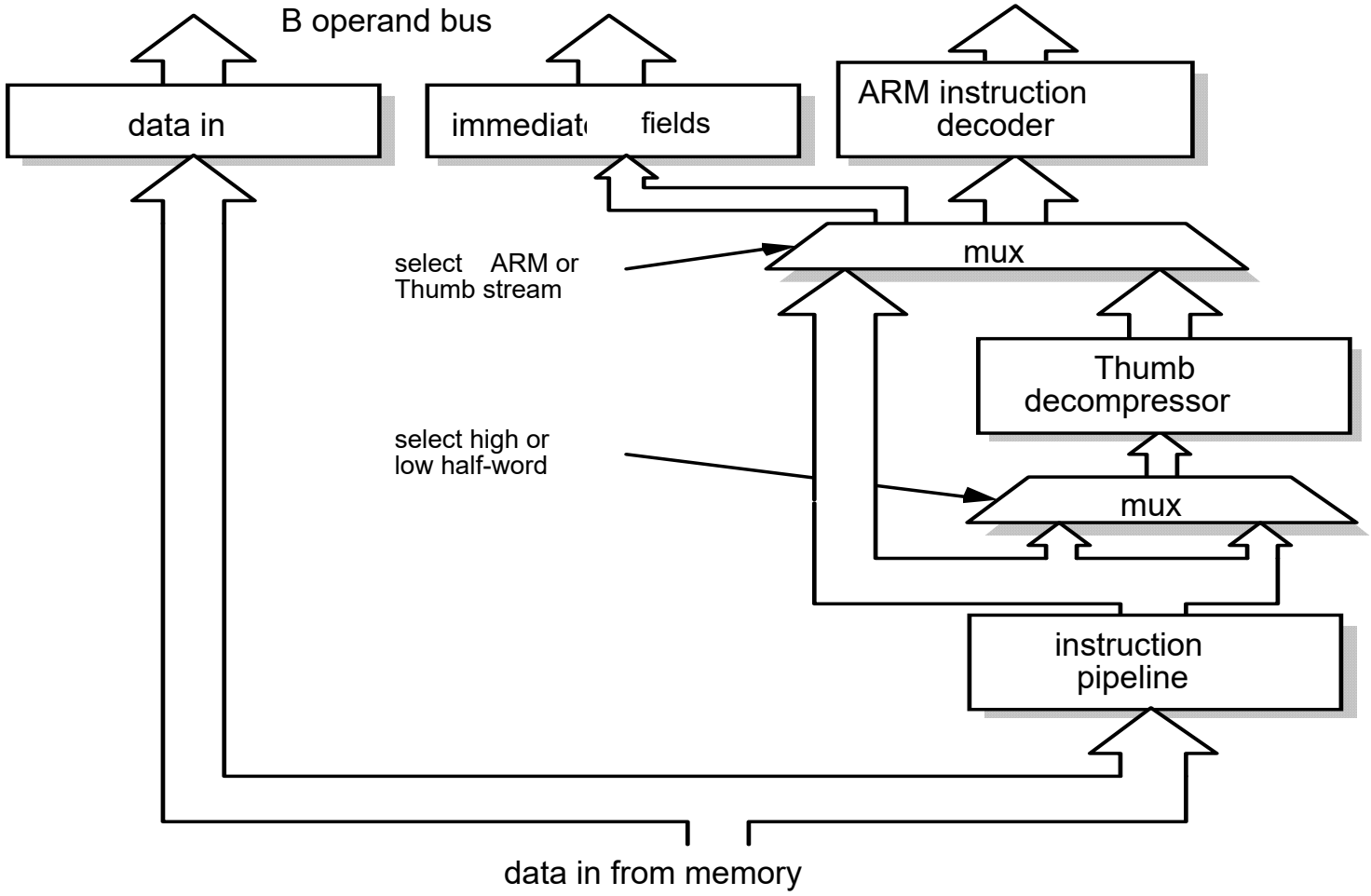
Thumb Befehlssatz

- Beispiel (3) Addition ADD R2, #1



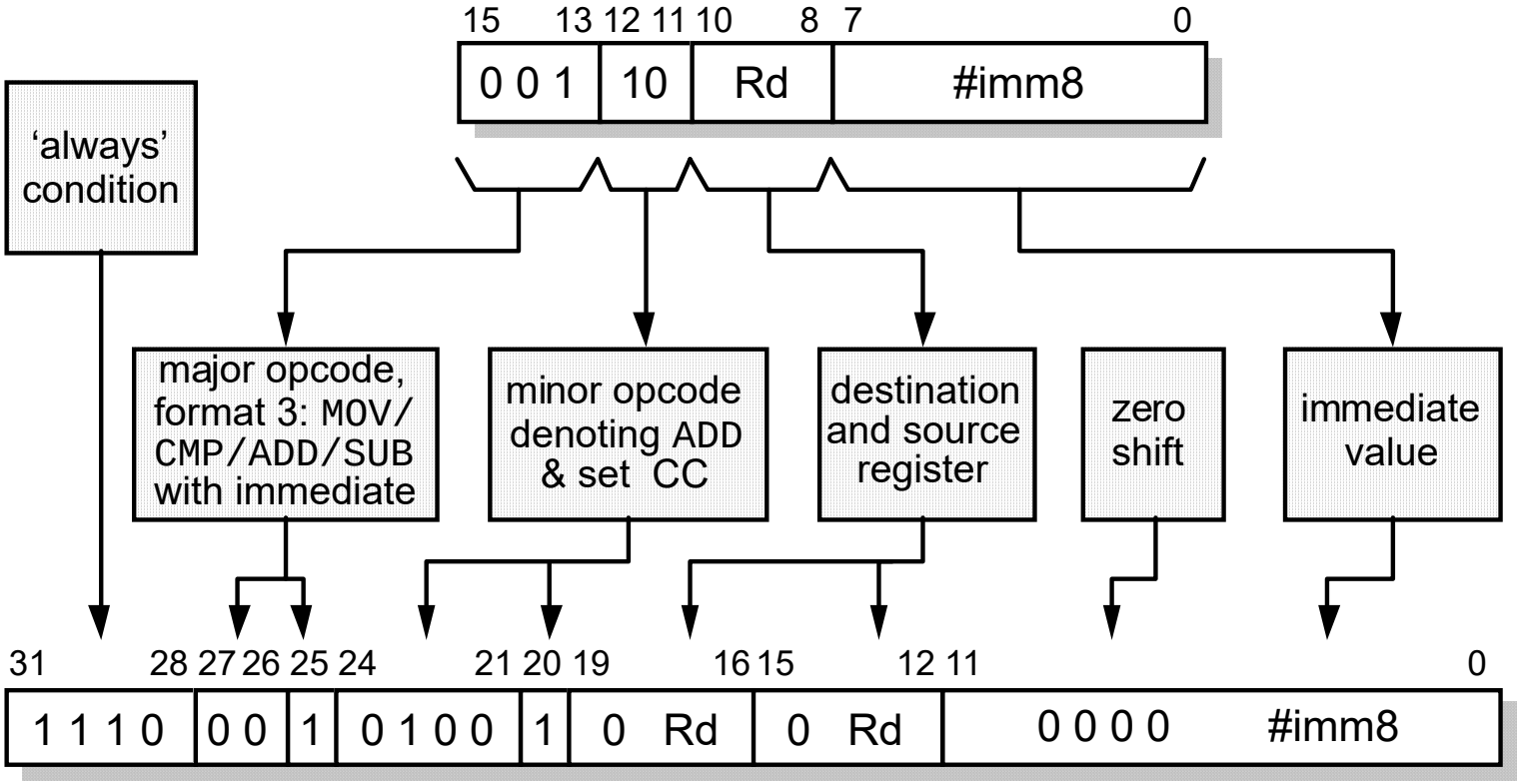
Thumb Befehlssatz

- Dekodierung von Thumb-Befehlen zu ARM Befehlen



Thumb Befehlssatz

- Decodierung eines 16-Bit Thumb-Befehls zu einem 32-Bit ARM-Befehl

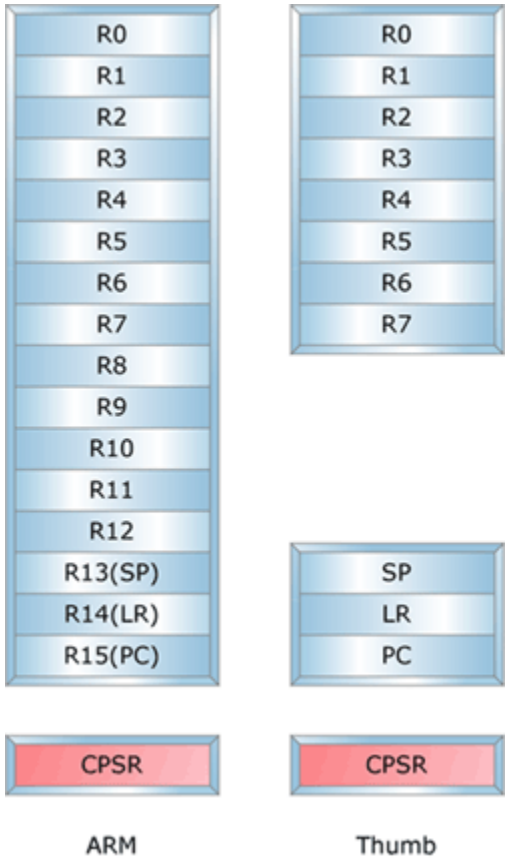


Thumb Befehlssatz

- Thumb-Entry:
  - Nach Systemstart im ARM Modus
  - Sprung in Thumb-Modus durch BX Rm Befehl, falls unterstes Bit in Rm gesetzt
- Thumb-Exit:
  - Implizite Rückkehr durch eine Exception
  - Explizite Rückkehr zu ARM durch BX Rm Befehl, falls unterstes Bit in Rm nicht gesetzt

Thumb Befehlssatz

- Eingeschränkter Zugriff auf Register R8 – R12



Thumb Befehlssatz

makefile

```
$(TC)gcc -c -g –O0 -mthumb -mthumb-interwork thumb.c -I ../h
```

bzw.

```
$(TC)gcc -c -g –O0 thumb.c -I ../h
```

Quelltext thumb.c

```
…
void led_on(int led) {
        StructPIO* piobaseB = PIOB_BASE;   //Basisadresse PIO B
        piobaseB->PIO_SODR = led;          //LED einschalten
}

void flash_function(void) {
        unsigned long i,delay;
        for (i = 0; i <= 8; i++)  {  //LED Flasher
                //einfache Warteschleife
                for (delay = 0; delay<0x10000; delay++);
                //Setzen der naechsten LED
                led_on((1<<i+8));}
}
…
```

Thumb Befehlssatz

Vergleich ARM-Assembler

```
led_on:
        str     fp, [sp, #-4]!
        add     fp, sp, #0
        sub     sp, sp, #20
        str     r0, [fp, #-16]
        ldr     r3, .L2
        str     r3, [fp, #-8]
        ldr     r2, [fp, #-16]
        ldr     r3, [fp, #-8]
        str     r2, [r3, #48]
        add     sp, fp, #0
        ldmfd   sp!, {fp}
        bx      lr


flash_fct:
        stmfd   sp!, {fp, lr}
        add     fp, sp, #4
        sub     sp, sp, #8
        mov     r3, #0
        str     r3, [fp, #-8]
        b       .L5
.L8:
        mov     r3, #0
        str     r3, [fp, #-12]
        b       .L6
.L7:
        ldr     r3, [fp, #-12]
        add     r3, r3, #1
        str     r3, [fp, #-12]
.L6:
        ldr     r2, [fp, #-12]
        ldr     r3, .L9
        cmp     r2, r3
        bls     .L7
        ldr     r3, [fp, #-8]
        add     r3, r3, #8
        mov     r2, #1
        mov     r3, r2, asl r3
        mov     r0, r3
        bl      led_on
        ldr     r3, [fp, #-8]
        add     r3, r3, #1
        str     r3, [fp, #-8]
.L5:
        ldr     r3, [fp, #-8]
        cmp     r3, #8
        bls     .L8
        sub     sp, fp, #4
        ldmfd   sp!, {fp, lr}
        bx      lr
```

Thumb-Assembler

```
led_on:
        push    {r7, lr}
        sub     sp, sp, #16
        add     r7, sp, #0
        str     r0, [r7, #4]
        ldr     r3, .L2
        str     r3, [r7, #12]
        ldr     r2, [r7, #4]
        ldr     r3, [r7, #12]
        str     r2, [r3, #48]
        mov     sp, r7
        add     sp, sp, #16
        pop     {r7}
        pop     {r0}
        bx      r0

flash_fct:
        push    {r7, lr}
        sub     sp, sp, #8
        add     r7, sp, #0
        mov     r3, #0
        str     r3, [r7, #4]
        b       .L5
.L8:
        mov     r3, #0
        str     r3, [r7]
        b       .L6
.L7:
        ldr     r3, [r7]
        add     r3, r3, #1
        str     r3, [r7]
.L6:
        ldr     r2, [r7]
        ldr     r3, .L9
        cmp     r2, r3
        bls     .L7
        ldr     r3, [r7, #4]
        add     r3, r3, #8
        mov     r2, #1
        mov     r1, r2
        lsl     r1, r1, r3
        mov     r3, r1
        mov     r0, r3
        bl      led_on
        ldr     r3, [r7, #4]
        add     r3, r3, #1
        str     r3, [r7, #4]
.L5:
        ldr     r3, [r7, #4]
        cmp     r3, #8
        bls     .L8
        mov     sp, r7
        add     sp, sp, #8
        pop     {r7}
        pop     {r0}
        bx      r0
```

Thumb Befehlssatz

Beispielprogramm Led-Flasher
(Codedichte Thumb 58%)

	ARM	Thumb
led_on()	12	14
flasher_fct()	31	36
Bytes	43*4=172	50*2=100

Analyse nach Furber

- Der Thumb-Code benötigt nur 70% so viel Platz wie der ARM-Code.
- Der Thumb-Code verwendet 40% mehr Befehle als der ARM-Code.
- Bei 32-Bit-Speicher ist der ARM-Code 40% schneller als der Thumb-Code.
- Bei 16-Bit-Speicher ist der Thumb-Code 45% schneller als der ARM-Code.
- Thumb-Code benötigt für externen Speicher 30% weniger Energie als ARM-Code.

Quelle: Steve Furber, ARM-Rechnerarchitekturen für System-on-Chip-Design

Gliederung

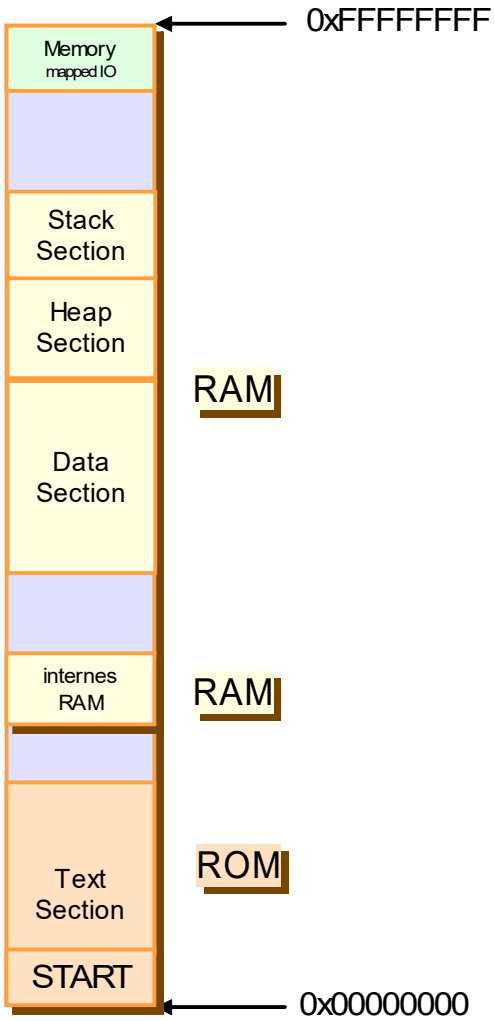
- Einleitung
- Power Management
- Parallele I/O (PIO)
- Interrupt Handling
- Timer
  - WAVE Mode / Capture Mode
- Softwareinterrupt (SWI)
- Serielle Schnittstelle
- Signalaufbereitung
  - Analog-Digital Wandlung / Digital-Analog-Wandlung
- Weiterführende Themen
  - Thumb Befehlssatz / **Systemstart**

Systemstart

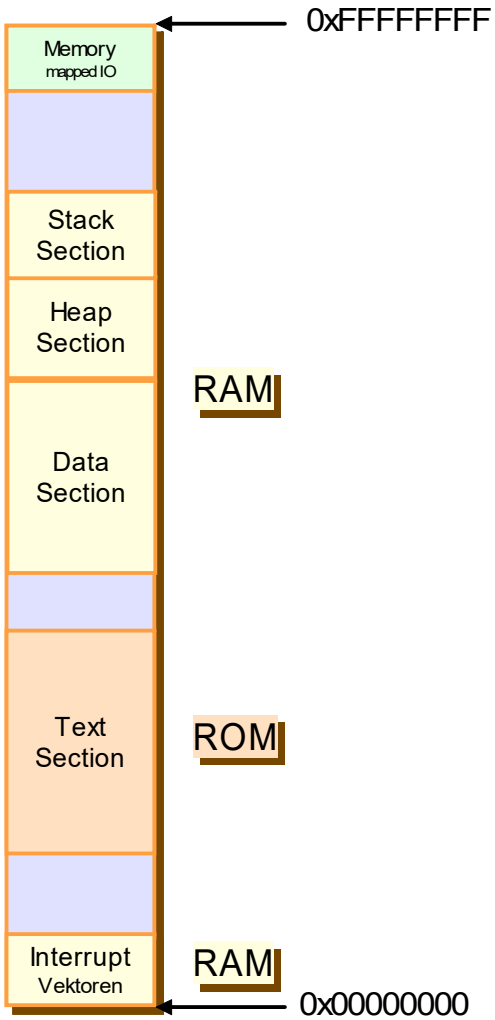
- Prozessor startet bei Power UP an der Adresse 0
- Problem:
  - bei Systemstart sollte an dieser Stelle ROM (Flash) Speicher sein
  - später sollte die Vektortabelle im RAM liegen, um flexible Programmierung zu ermöglichen
- Lösung:
  - Memory Controller erlaubt ein Remapping der Speicherbereiche zur 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 internes RAM
Remapping der Speicherbereiche
Definition der Stackgrössen
Initialisieren der Stackpointer
Initialisieren der BSS Section (uninitialisierte Variablen)
Sprung in main Routine

siehe Datei boot_flash.s

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

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 in Vergleich zu 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 ?

- 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

Wozu Assembler Programmierung ?

Linux Kernel LOC by language

Quelle: 5/2014 ohloh.net

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C	15,972,569	3,374,925	17.4%	3,061,307	22,408,801	94.5%
Assembly	477,900	99,789	17.3%	78,229	655,918	2.8%
C++	256,809	115,807	31.1%	53,242	425,858	1.8%
XML	86,028	497	0.6%	7,133	93,658	0.4%
Make	41,319	12,081	22.6%	11,152	64,552	0.3%
Perl	23,049	3,661	13.7%	4,536	31,246	0.1%
shell script	9,362	3,823	29.0%	1,600	14,785	0.1%
Python	4,460	701	13.6%	814	5,975	0.0%
TeX/LaTeX	1,822	6	0.3%	216	2,044	0.0%
HTML	1,128	0	0.0%	116	1,244	0.0%
AWK	844	103	10.9%	107	1,054	0.0%
Scheme	436	0	0.0%	126	562	0.0%
Objective-C	378	0	0.0%	110	488	0.0%
Autoconf	91	5	5.2%	15	111	0.0%
XSL Transformation	70	27	27.8%	13	110	0.0%
Vim Script	27	12	30.8%	3	42	0.0%
Automake	20	0	0.0%	5	25	0.0%
Totals	16,876,312	3,611,437		3,218,724	23,706,473	

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.

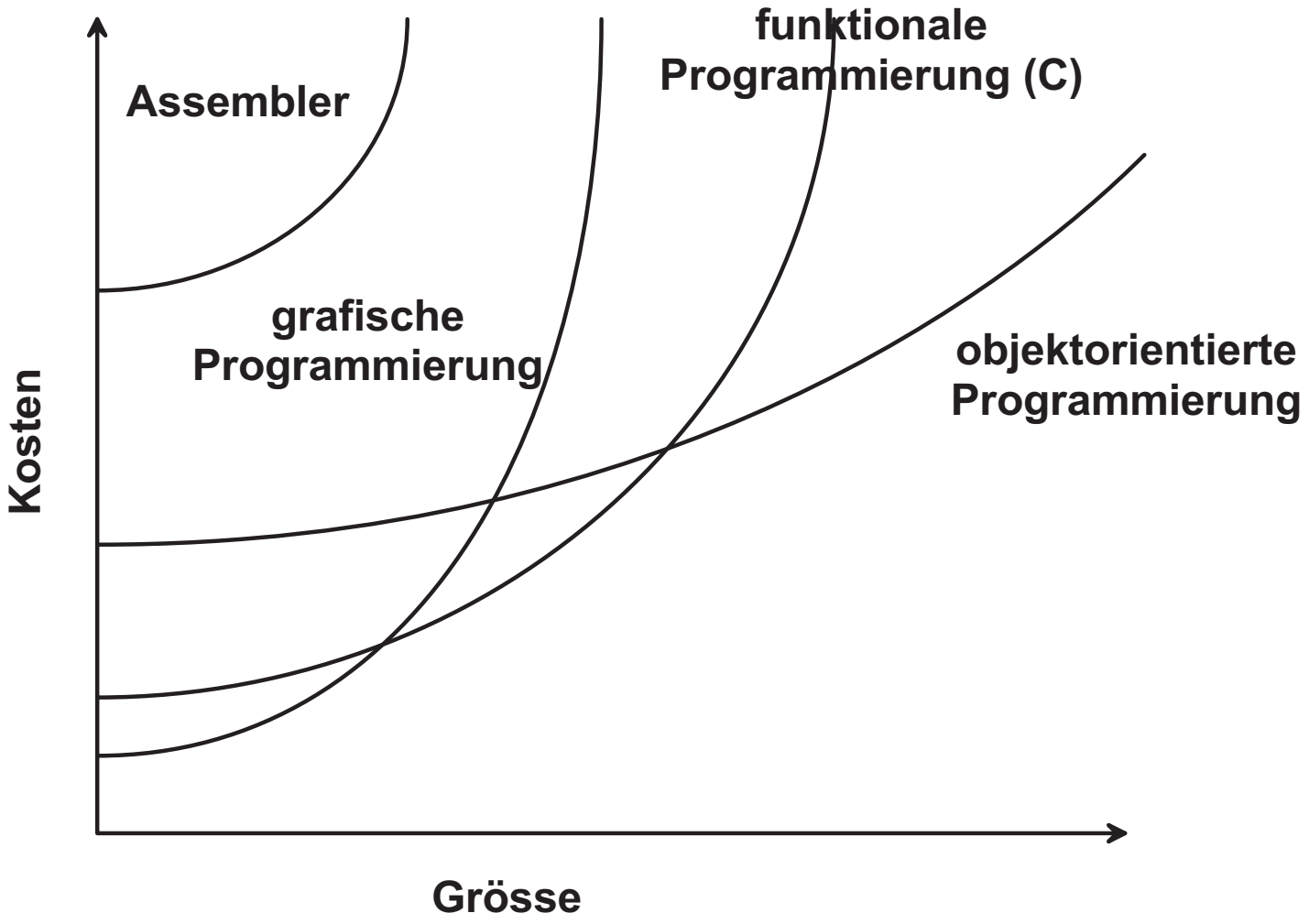
Wozu Assembler Programmierung ?

Open Office LOC by language

Quelle: 5/2014 ohloh.net

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C++	4,808,277	1,090,805	18.5%	1,004,504	6,903,586	45.8%
HTML	4,031,781	45,158	1.1%	848,096	4,925,035	32.7%
XML	1,021,736	133,427	11.6%	71,847	1,227,010	8.1%
Java	615,279	270,814	30.6%	133,641	1,019,734	6.8%
XSL Transformation	179,689	15,616	8.0%	13,929	209,234	1.4%
Make	114,540	55,298	32.6%	34,385	204,223	1.4%
CSS	110,438	29,153	20.9%	28,825	168,416	1.1%
C	78,629	29,811	27.5%	18,695	127,135	0.8%
Perl	75,770	19,045	20.1%	19,842	114,657	0.8%
JavaScript	46,418	10,571	18.5%	5,505	62,494	0.4%
Visual Basic	14,973	2,096	12.3%	2,680	19,749	0.1%
Modula-2	10,800	0	0.0%	1,736	12,536	0.1%
Objective-C	9,871	2,162	18.0%	1,793	13,826	0.1%
shell script	9,652	3,624	27.3%	1,366	14,642	0.1%
Python	8,637	2,470	22.2%	2,105	13,212	0.1%
C#	6,632	1,463	18.1%	1,030	9,125	0.1%
Autoconf	6,534	635	8.9%	562	7,731	0.1%
TeX/LaTeX	5,494	0	0.0%	1,011	6,505	0.0%
Assembly	2,602	387	12.9%	246	3,235	0.0%
XML Schema	2,062	121	5.5%	46	2,229	0.0%
Scilab	1,111	0	0.0%	3	1,114	0.0%
AWK	771	389	33.5%	77	1,237	0.0%
Totals	11,164,083	1,713,526		2,192,354	15,069,963	

Produktivität von Programmiersprachen



Ausblick

- Die Programmierung von Mikroprozessoren erfordert besondere Fähigkeiten
  - Erfordert Bewusstsein für den schonenden Umgang mit Ressourcen
  - Speicher, Laufzeit, Energieverbrauch
- Mikroprozessoren „bewegen“
  - http://www.mikrokopter.de
  - http://www.heise.de/ct/projekte/machmit/ctbot/wiki
  - Balancing two-wheel scooter (ATMEL contest)
  - Rotierende Uhr
  - ….