

# Computer Engineering

## WS 2012

### Einleitung

HTM – SHF - SWR



## Themen:

- ▶ **Ein- und Ausgabe, Peripherie**
  - ▶ **Memory Mapped, IO Mapped**
  - ▶ **Einfache Peripherie, General Purpose IO (GPIO)**
  - ▶ **Timer**
  - ▶ **Interruptverarbeitung**
  - ▶ **Serielle Übertragung**
  - ▶ **Speicher: RAM, Flash, EEPROM**
- ▶ **Softwareerstellung für Embedded Systems**
  - ▶ **Erstellung, Download, Debuggen**
  - ▶ **Startup-Code**



CE WS12

## Literatur:

- ▶ **LPC24XX User Manual**  
[http://www.nxp.com/acrobat\\_download/usermanuals/UM10237.pdf](http://www.nxp.com/acrobat_download/usermanuals/UM10237.pdf)
- ▶ **LPC2468 Data Sheet**  
[http://www.nxp.com/acrobat\\_download/datasheets/LPC2468.pdf](http://www.nxp.com/acrobat_download/datasheets/LPC2468.pdf)
- ▶ **The Insider's Guide To The NXP LPC2300/2400 Based Microcontrollers, Hitex**  
<http://www.hitex.com/index.php?id=download-insiders-guides>
- ▶ **ARMv5 Architecture Reference Manual, gilt u. a. für ARM7**  
<http://www.arm.com/misPDFs/14128.pdf>
- ▶ **Sloss, Symes, Wright : ARM System Developer's Guide, Morgan Kaufmann Publishers, 2004.**
- ▶ **William Hohl: ARM Assembly Language, CRC Press, 2009.**
- ▶ **Helmut Bähring: Mikrorechner-Technik, Springer, 2002.**
- ▶ **Wayne Wolf: Computer as Components, Morgan Kaufmann, 2008.**



CE WS12

## Übersicht



- ▶ Einleitung
- ▶ Softwareerstellung

## Was kennzeichnet ein eingebettetes System?

- ▶ Gerät, das ein programmierbares Prozessorsystem enthält
- ▶ Kein Allzweckrechner
  - ▶ PC selbst ist kein eingebettetes System
  - ▶ PC kann aber zum Aufbau eines eingebetteten Systems verwendet werden

## Anwendungsbereiche

- ▶ Automatisierung
  - ▶ Steuern und Regeln von Prozessen, Fertigungs- und Produktionsanlagen, Überwachung
- ▶ Medizintechnik
  - ▶ Beatmungsgeräte, Narkosesysteme
  - ▶ Computer-Tomograph, Ultraschallgeräte
- ▶ Netzwerk
  - ▶ Router, Switches, WLAN, VoIP
- ▶ Haushalt
  - ▶ Waschmaschine, Mikrowelle, ...
- ▶ Unterhaltung
  - ▶ Handys, Digitalkameras
  - ▶ Digitales Fernsehen, Video-on-demand, Hi-Fi, DVD, Spiele, ...



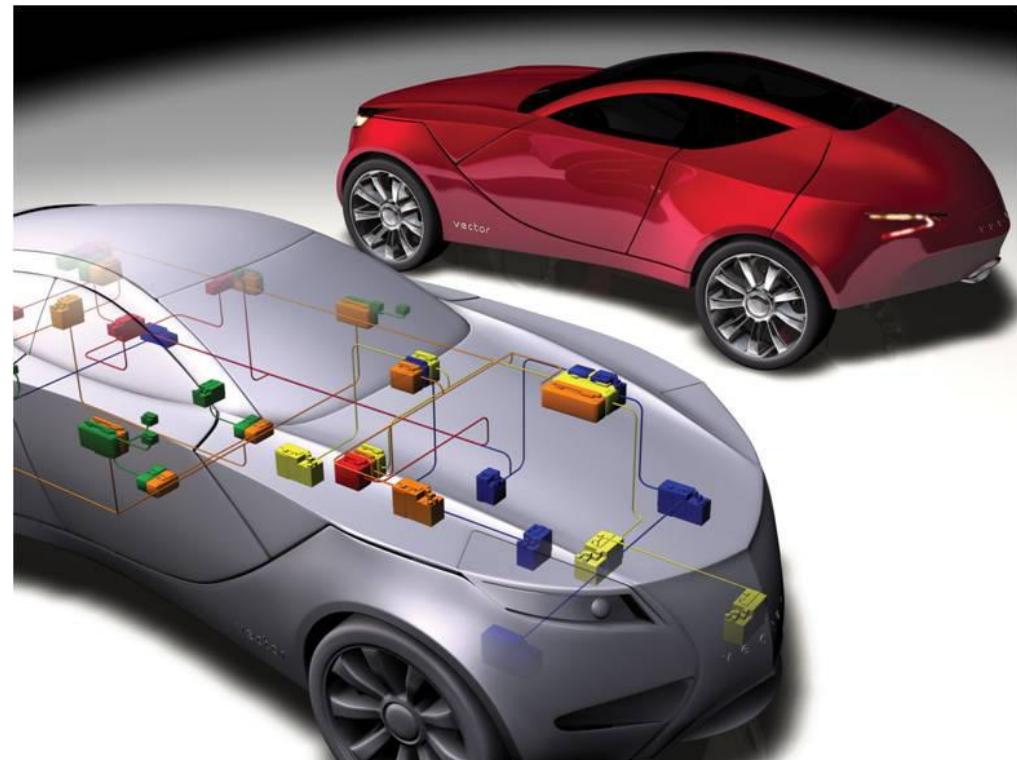


CE WS12

## Anwendungsbereiche

### ► Automobiltechnik

- ▶ Oberklasse:  
**Anzahl Steuergeräte >80**
- ▶ Diverse Bussysteme:
  - CAN
  - LIN
  - Flexray
  - MOST
- ▶ Anwendungen:
  - Motorsteuerung
  - x-by-wire:  
**Bremsen,  
Lenkung**
  - Komfort
  - Unterhaltung



CE WS12

## Typische Anforderungen an eingebettete Systeme

- ▶ Hohe Ausfallsicherheit
- ▶ kurze Reaktionszeiten
- ▶ vorgegebene technische Randbedingungen und Schnittstellen
- ▶ Geringer Leistungsverbrauch
- ▶ Niedrige Entwicklungs-/Produktionskosten

## Eingebettetes Prozessorsystem: Typische Komponenten

- ▶ CPU
- ▶ Speicher
  - ▶ **Programmspeicher (Flash), Datenspeicher (RAM), Parameterspeicher (EEPROM)**
- ▶ Zeitgeber
  - ▶ Steuerung von periodischen Abläufen
  - ▶ PWM (Puls-Weiten-Modulation)
- ▶ Ein- und Ausgabe
  - ▶ Digitale Schnittstellen
  - ▶ parallel, seriell (RS232, USB, I2C, CAN, ...)
  - ▶ Analoge Schnittstellen
    - Analoge Ein- und Ausgabe
- ▶ Systemmanagement
  - ▶ Speicherverwaltung
  - ▶ Interruptsteuerung
  - ▶ DMA-Steuerung
  - ▶ Powermanagement, Ausfallsicherheit (Watchdog)

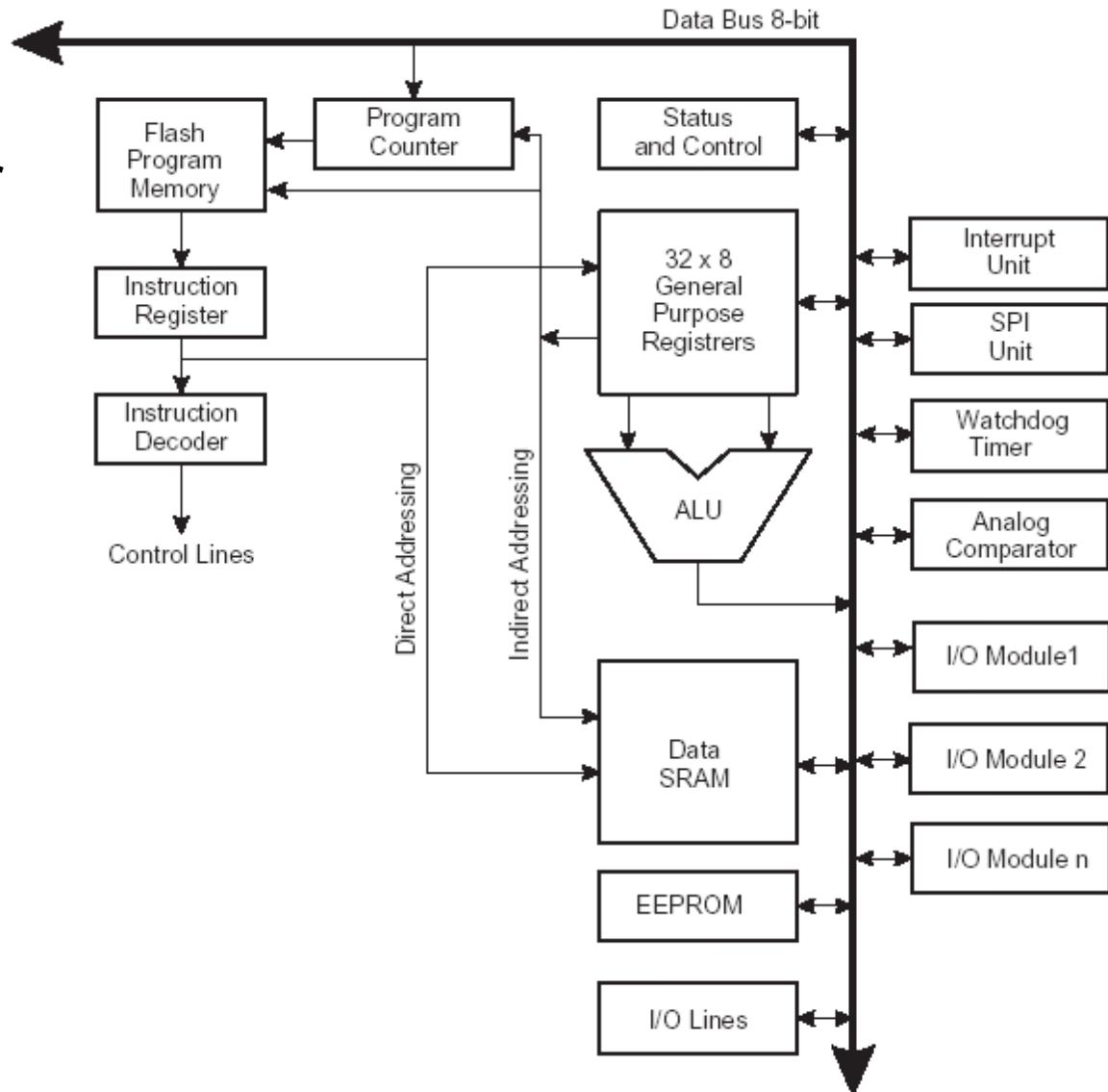
## Mikrocontroller

- ▶ Enthält viele Komponenten auf einem Chip, z. B.:
  - ▶ CPU
  - ▶ Speicher
  - ▶ RAM
  - ▶ Flash
  - ▶ Zeitgeber
  - ▶ I/O
- ▶ Unterscheiden sich in
  - ▶ Geschwindigkeit (Taktfrequenz)
  - ▶ Breite Datenbus ( z. B.: 4, 8, 16, 32 Bit )
  - ▶ Speichergrößen
  - ▶ I/O Komponenten

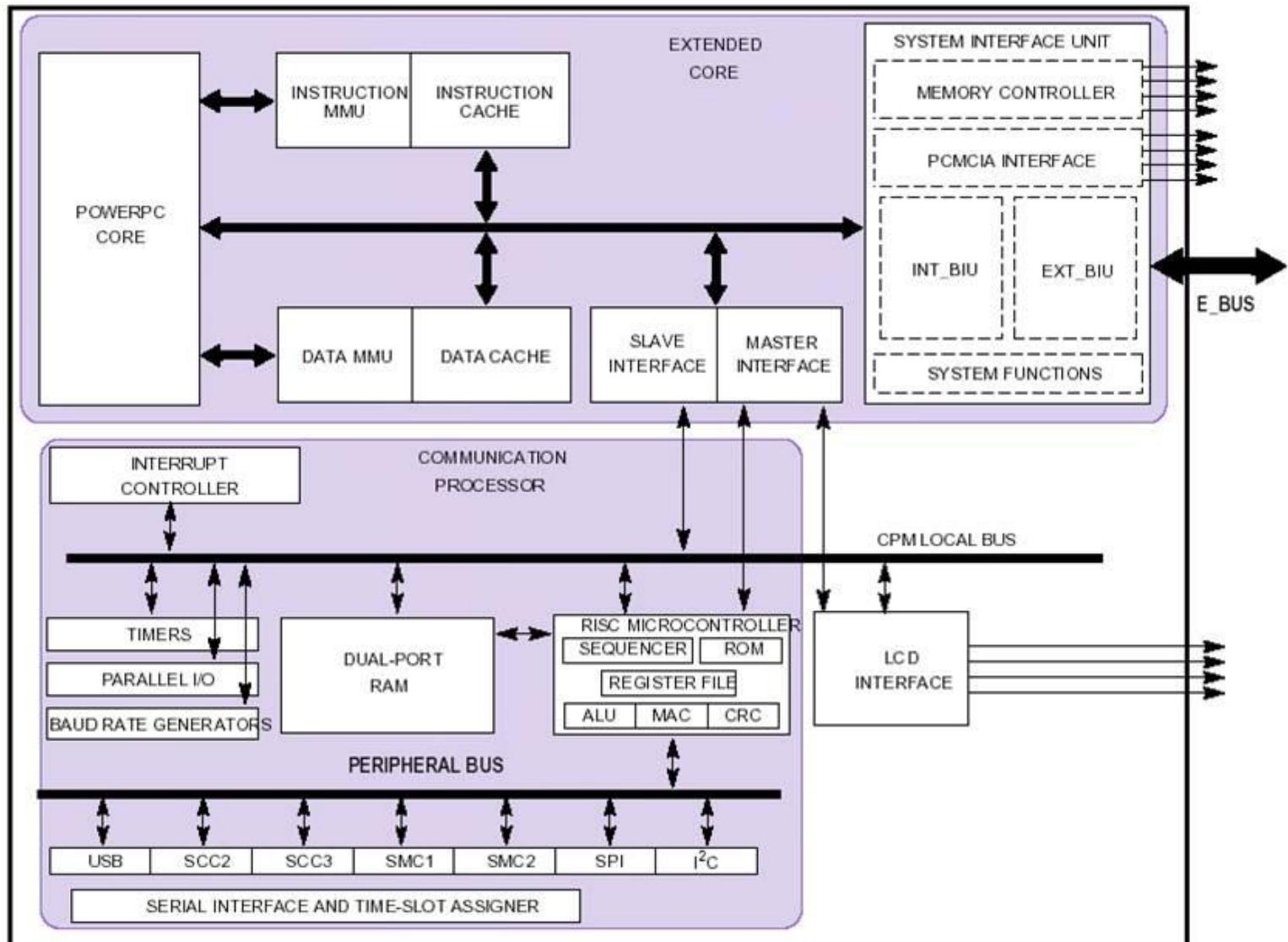


## Mikrocontroller ATMEL, AVR

- ▶ Harvard-Architektur
- ▶ 8-Bit Datenbus
- ▶ I/O Module:
  - 53 I/O Lines
  - 4 Timer
  - A/D Converter
  - 2 RS232
  - 1 SPI
  - CAN Feldbus



CE WS12 **Mikrocontroller Freescale, MPC823e (Power PC)**

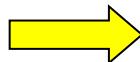




CE WS12

## Übersicht

- ▶ Einleitung
- ▶ Softwareerstellung



## Besonderheiten Softwareentwicklung für Eingebettete Systeme

- ▶ Aufteilung der Implementierung in
  - ▶ **Hardware-basiert**
  - ▶ **Software-basiert**
- ▶ Betriebssysteme
  - ▶ häufig kein Betriebssystem
  - ▶ wenn Betriebssystem, dann meist
    - kein Speicherschutz
    - echtzeitfähig
    - erfordern meist BSPs (Board Support Package)
- ▶ Massenspeicher
  - ▶ häufig kein Massenspeicher
    - Programme werden im ROM und
    - nichtflüchtige Daten im EEPROM gespeichert
- ▶ Softwareerstellung
  - ▶ Verwendung von Crosscompilern
    - erfordert „Download des erzeugten Codes“



# Embedded Systems

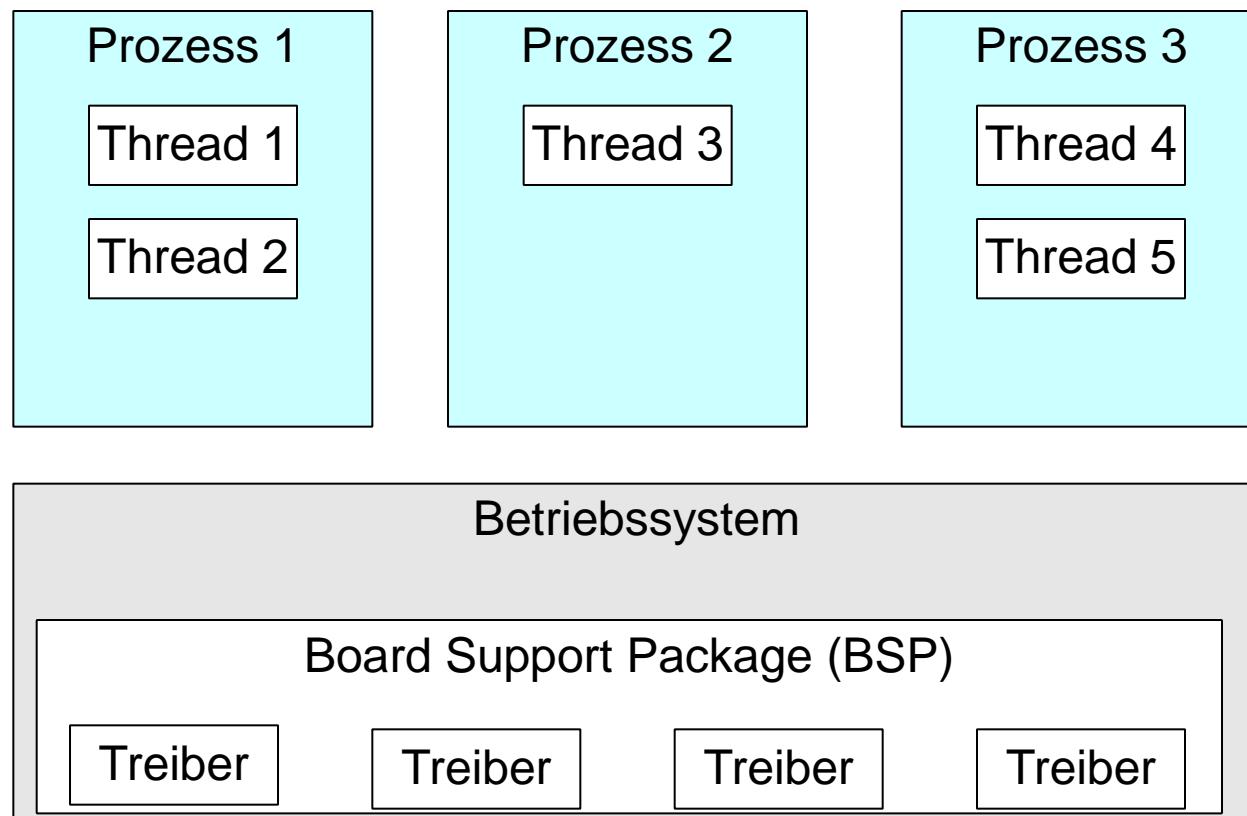
CE WS12

## Besonderheiten Softwareentwicklung für Eingebettete Systeme

- ▶ Debugging, Fehlersuche
  - ▶ erfordert zusätzliche Werkzeuge
    - Simulator
    - Crossdebugger, Anbindung über
      - serielle Schnittstelle
      - Netzwerk
      - JTAG-Interface
    - Logikanalysator
      - Fehlersuche ohne Beeinflussung des Zeitverhaltens
    - Emulator
  - ▶ Testen
    - ▶ erfordert
      - Einbindung in die reale Umwelt oder
      - Verwendung von „Hardware in the Loop-Technik“
  - ▶ Softwarepflege
    - ▶ erfordert Möglichkeit zum Firmware-Update

CE WS12

## Embedded System mit Betriebssystem



# Embedded Systems

## Typischer Aufbau einer Thread

```
int variable1;

double variable2 = 3.14;

void *PrintHello(void *threadarg) {

    int variable3;

    initialisierung();

    while( 1 ) {

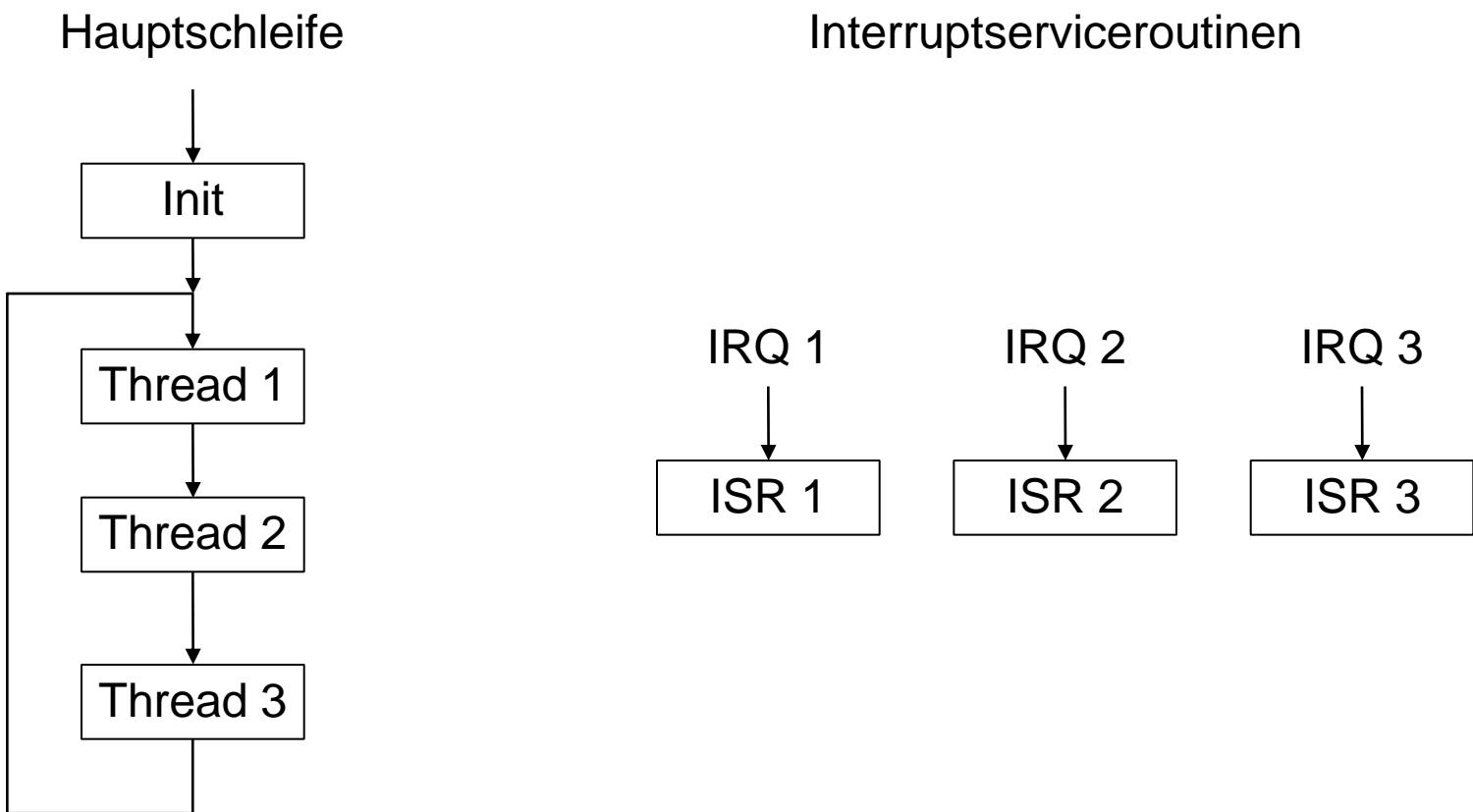
        warte_auf_Ereignisse();

        bearbeitung();

        ausgabe();
    }
}
```

CE WS12

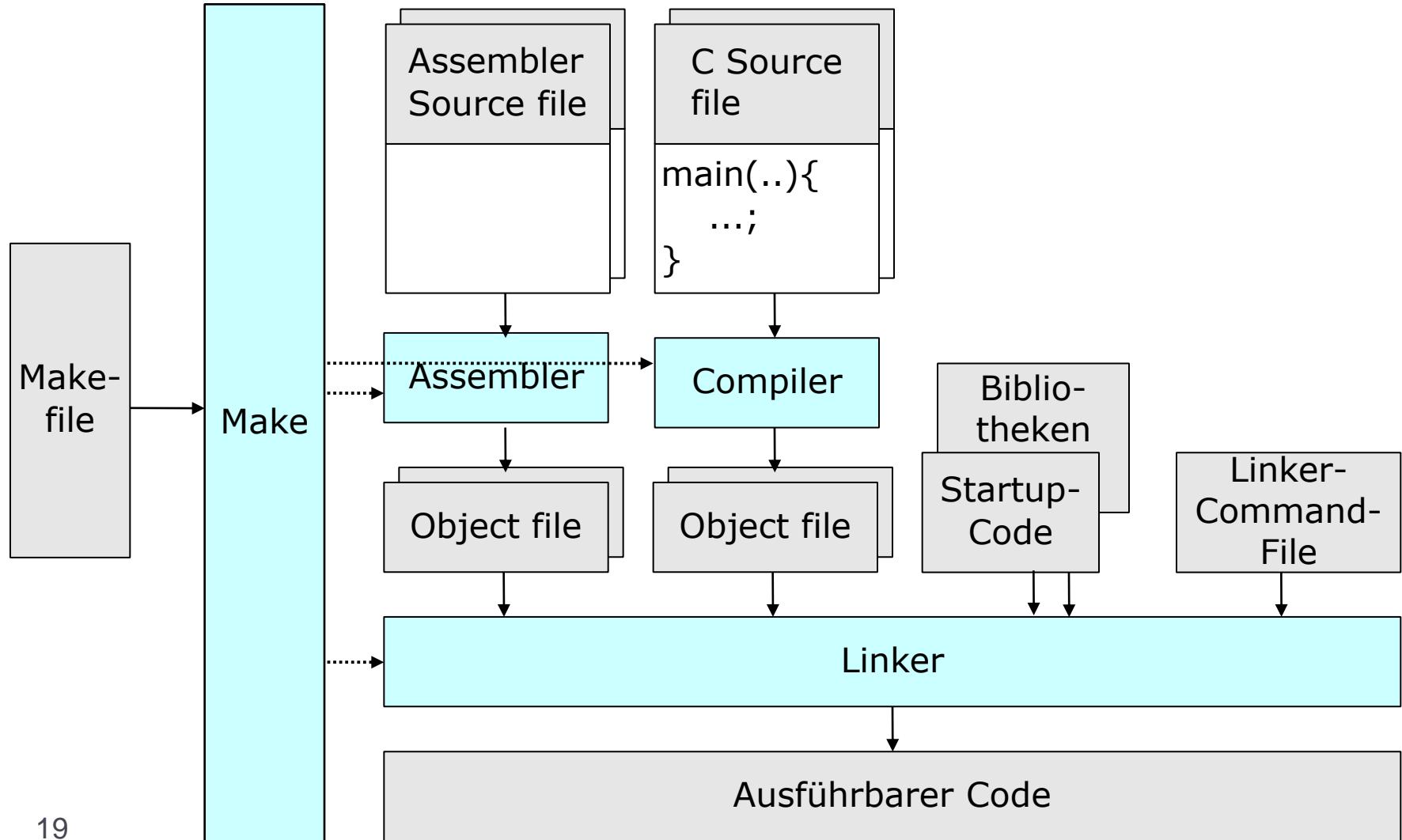
## Embedded System ohne Betriebssystem





CE WS12

## Softwareentwicklung



CE WS12

## Speicherarten

ROM  
(Flash)

- Persistente Speicherung
- Löschbar (blockweise)
- N-mal wiederholbar (typ. 1000)
- Langsam
- Preiswert

- Programmspeicher

RAM

- Flüchtige Speicherung
- Schnell
- Teuer

- Arbeitsspeicher

EEPROM

- Persistente Speicherung
- Byteweise löschbar
- N-mal wiederholbar (typ.  $10^6$ )
- Langsam
- Teuer

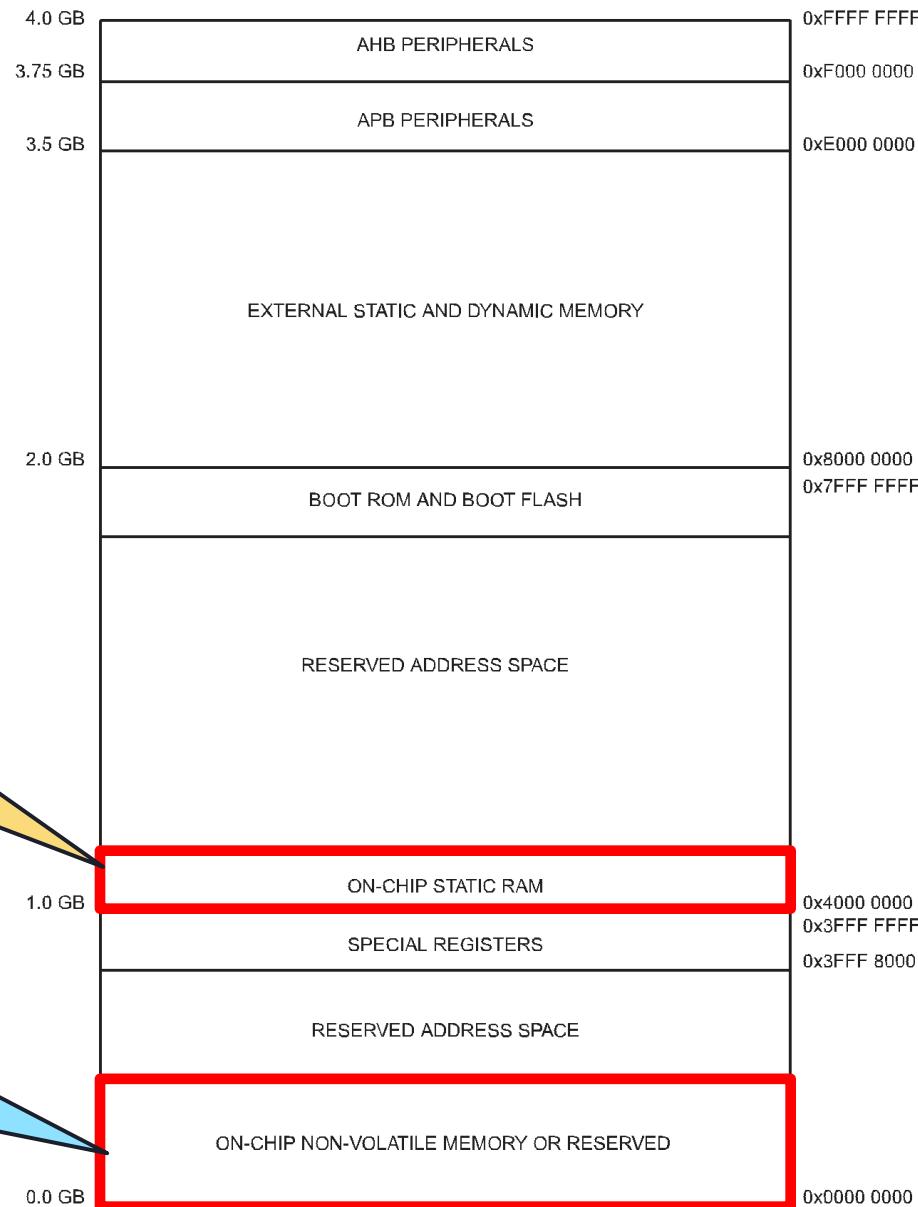
- Speicherung von Konfigurationsdaten



# Embedded Systems

CE WS12

## Speicherbelegung LPC2468





CE WS12

## Sektionen

- ▶ Ausführbarer Code ist in Sektionen aufgeteilt
- ▶ Ermöglicht gezielte Platzierung des erzeugten Codes im Speicher
- ▶ Skript für Linker (Loader) gibt die Adressen der Sektionen vor

## Wichtige Sektionen:

Name	Inhalt	Speicherort	Bemerkung
.text	Programmcode	FLASH	
.data	Initialisierte globale Variable	RAM	Bei Programmstart wird Inhalt der Variablen vom FLASH in das RAM kopiert
.bss	Nicht initialisierte globale Variable	RAM	Bei Programmstart wird Inhalt der Variablen gelöscht

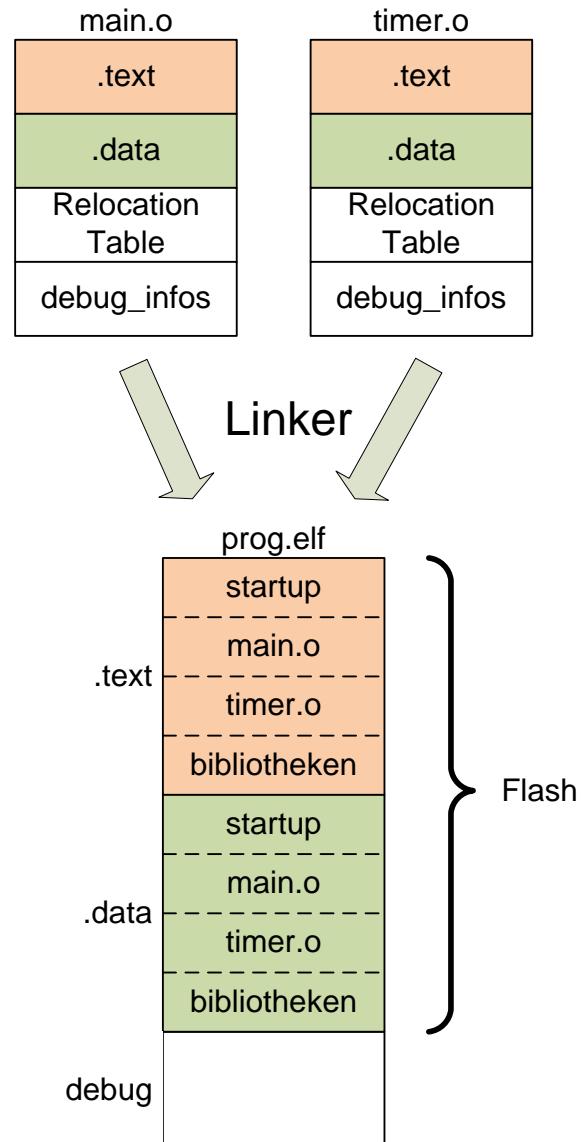


# Embedded Systems

CE WS12

## Linker (loader, ld)

- ▶ Zusammenfügen der Sektionen
- ▶ Zuweisung der Adressen
  - ▶ **Jede Sektion hat zwei Adressen:**
    - **Virtual Memory Address (VMA):** Adresse der Sektion, wenn das Programm ausgeführt wird.
    - **Load Memory Address (LMA):** Adresse, unter der die Daten abgelegt werden.
- ▶ Anwendung der Relocation Table
  - ▶ Platzhalter für Adressen werden durch die tatsächlichen Adressen ersetzt.



## Programmiersprache C: Speicheraufteilung

- ▶ Verwendung des Speichers:
    - ▶ Konstante (Verwendung des Schlüsselworts **const**):  
→ Spezielle Sektion **.rodata**.
    - ▶ Globale und statische Variable:  
→ **.data** oder **.bss** (Abhängig von Initialisierung)
    - ▶ Lokale Variable:  
→ **Stack**
    - ▶ Dynamische Variable:  
→ **Heap**
- Dynamische Verwendung  
Speicherbedarf schwer vorhersagbar.  
Kritisch: Rekursive Funktionsaufrufe.

# Embedded Systems

## Programmiersprache C: Speicheraufteilung

- ▶ Häufige Anordnung:

- ▶ **Heap:**

- Beginnt am Ende der **.bss-Sektion**.
    - Wächst von kleinen nach großen Adressen.

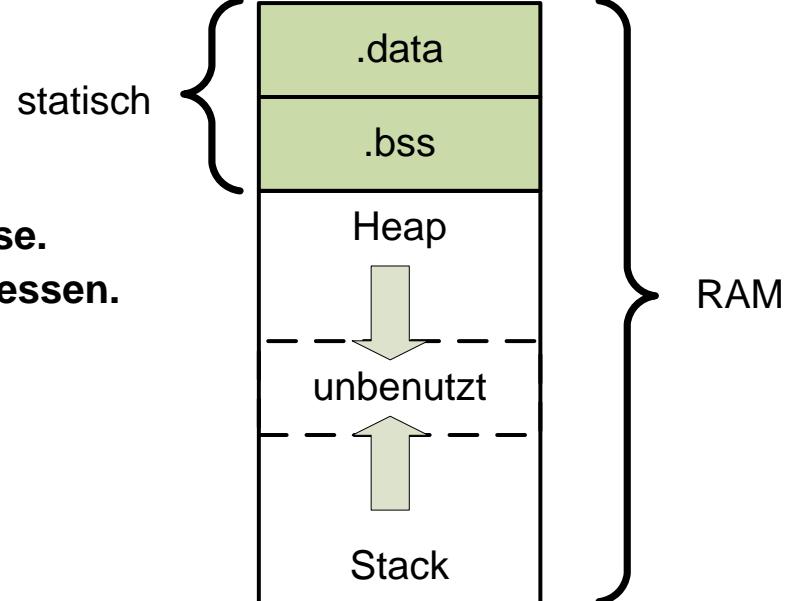
- ▶ **Stack:**

- Beginnt an der höchsten RAM-Adresse.
    - Wächst von großen nach kleinen Adressen.

- ▶ **Vorteil:** Sehr flexibel.

- ▶ **Nachteil:** Kein Schutz vor Stack- oder Heapüberlauf.

- ▶ In Embedded Systems muss der Heap- und Stackbedarf vorab abgeschätzt werden.  
Heap und Stack sollten statisch eingerichtet werden.



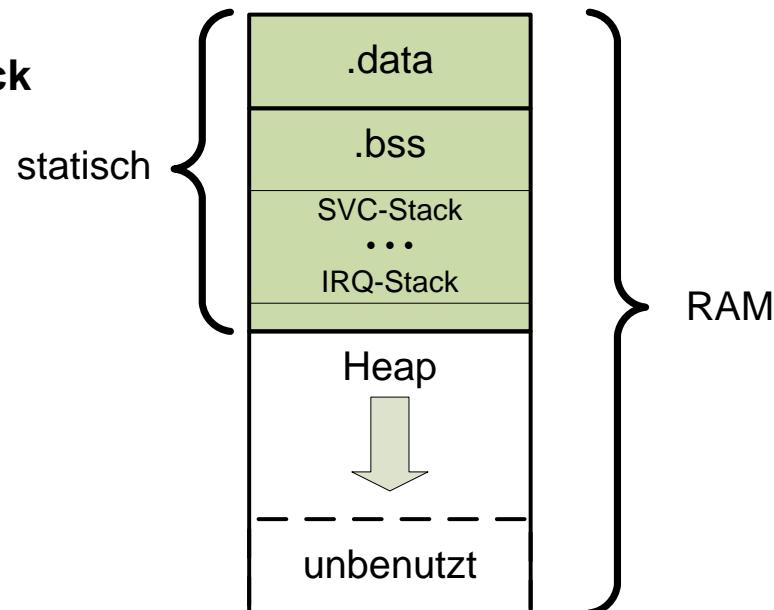


# Embedded Systems

## Programmiersprache C: Speicheraufteilung HiTOP

### ▶ ARM 7 TDMI:

- ▶ **6 Stacks für 7 Betriebsarten**
  - USR/SYS haben gemeinsamen Stack
- ▶ **Größen der Stacks werden statisch in der .bss-Sektion festgelegt**
  - Stacks werden in *start.s* definiert
- ▶ **Heap nutzt den gesamten restlichen Speicher ab Ende der .bss-Sektion**
  - Definiert durch die Laufzeitbibliothek



CE WS12

## Programmiersprache C: Laufzeitbibliothek

- ▶ Meist nicht Bestandteil des Compilers.
- ▶ Üblich:  
Verbindung der Anwendung mit dem Betriebssystem.
  - ▶ Betriebssystem wird über Software-Interrupt aufgerufen:
    - Umschaltung in System-Betriebsmode.
    - Höhere Privilegien.
  - ▶ Wesentliche Aufgaben:
    - Ein- und Ausgabe, Dateien und Geräte.
    - Prozessverwaltung, Starten und Stoppen von Prozessen.
    - Speicherverwaltung.

## Programmiersprache C: Laufzeitbibliothek HiTOP

- ▶ Verwendung der **newlib**:
  - ▶ Open-source Projekt
  - ▶ Standard C Bibliothek für Embedded Systems
  - ▶ Wird gepflegt von Red Hat Entwicklern
  
- ▶ Kann einfach angepasst werden:
  - Unterstützung einer Vielzahl von CPU-Typen
  - Verwendung von insgesamt 17 Stub-Funktionen (System Calls).
  - Optimierung des Speicherbedarfs (printf versus iprintf).
  - Kann reentrant-fähig übersetzt werden.

# Embedded Systems

## Programmiersprache C: Laufzeitbibliothek HiTOP

- ▶ newlib: System Calls, Beispiel Speicherallokierung

```
extern unsigned char end[];
extern unsigned char _end_of_heap_[];
static unsigned char *heap_ptr = NULL;

void* __sbrk_r( struct _reent *_s_r, ptrdiff_t nbytes )  {

    unsigned char *prev_heap_end;

    if ( heap_ptr == NULL)
        heap_ptr = end;

    prev_heap_end = heap_ptr;

    if ((heap_ptr + nbytes) > _end_of_heap_){
        return NULL;
    }

    heap_ptr += nbytes;

    return (void*)prev_heap_end;
}
```



# Embedded Systems

## Programmiersprache C: Laufzeitbibliothek HiTOP

### ▶ newlib: System Calls, Beispiel Ausgabe

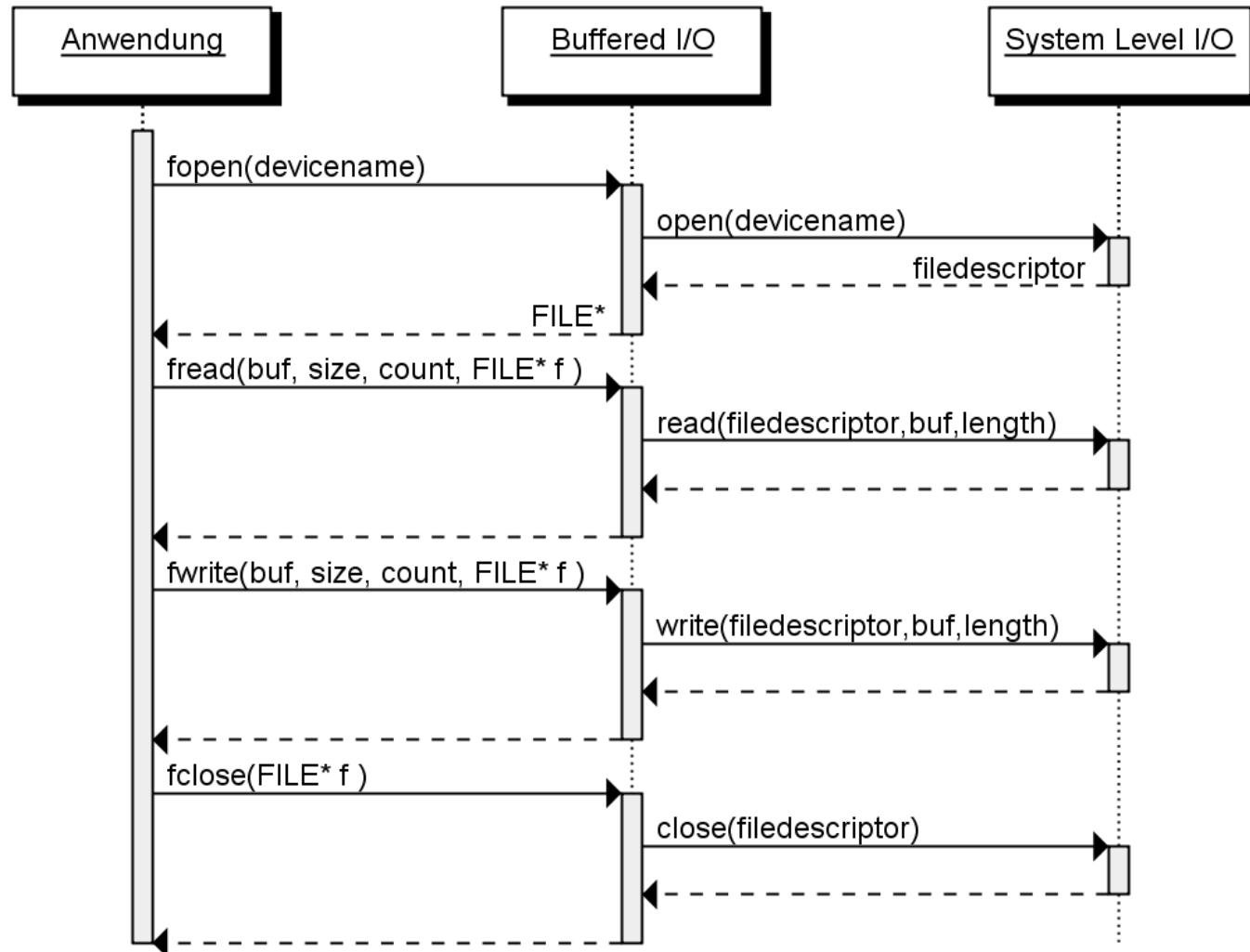
```
_ssize_t _write_r ( struct _reent *r, int file, const void *ptr,     size_t len) {  
    int i;  
    const unsigned char *p;  
    int retval;  
  
    p = (const unsigned char*) ptr;  
  
    for (i = 0; i < len; i++) {  
        if (*p == '\n' ){  
            do{  
                retval = uart0Putch('\r');  
            }while(retval == -1);  
        }  
  
        do{  
            retval = uart0Putch(*p);  
        }while(retval == -1);  
        p++;  
    }  
  
    return len;  
}
```



# Embedded Systems

## Programmiersprache C: Laufzeitbibliothek HiTOP

- ▶ newlib: System Calls, Beispiel Ausgabe



# Computer Engineering

## WS 2012

### Interruptverarbeitung

HTM – SHF - SWR

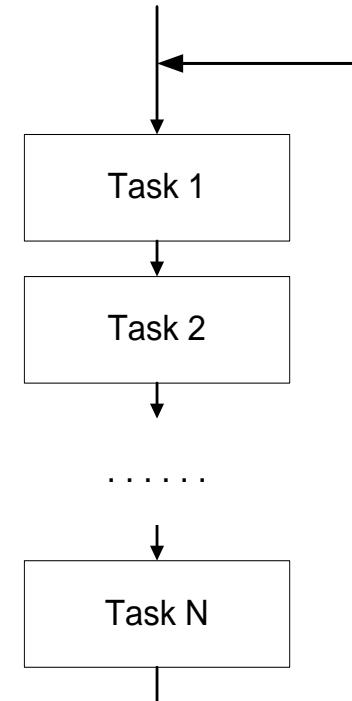


# Interrupt-verarbeitung

## Programmaufbau ohne Betriebssystem

- ▶ **Typisch:**  
Die einzelnen Aufgaben werden der Reihe nach in einer Hauptschleife abgearbeitet.
- ▶ **Maximale Reaktionszeit:**

**Summe der maximalen Reaktionszeiten aller Tasks**

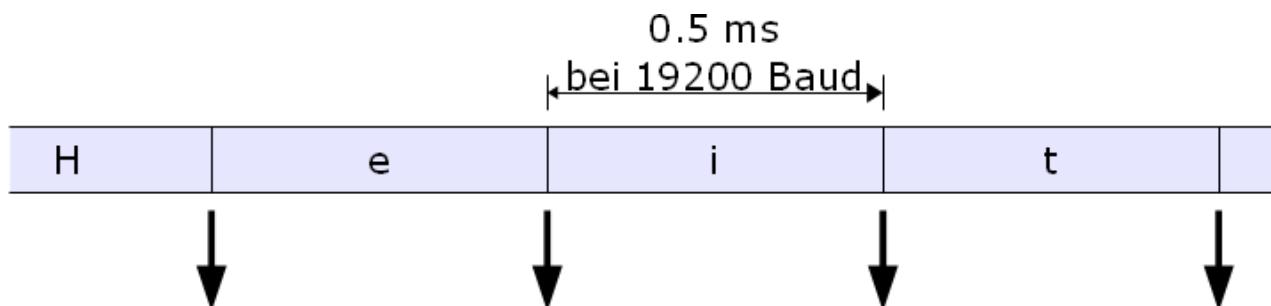
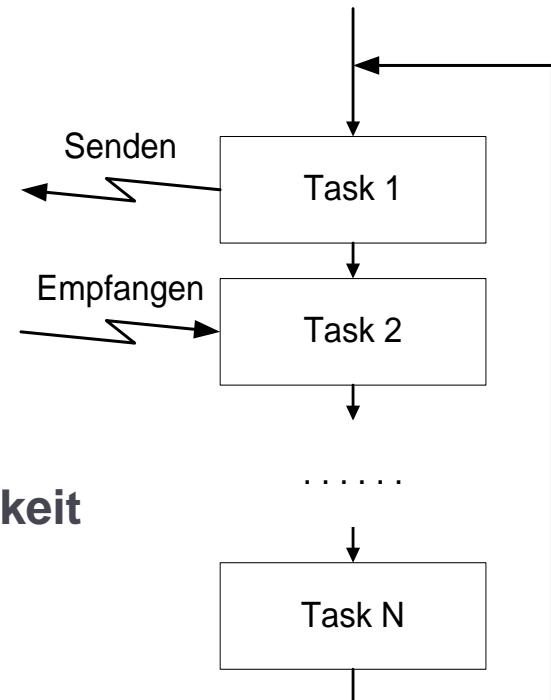




# Interrupt-verarbeitung

## Reaktion auf externes Ereignis

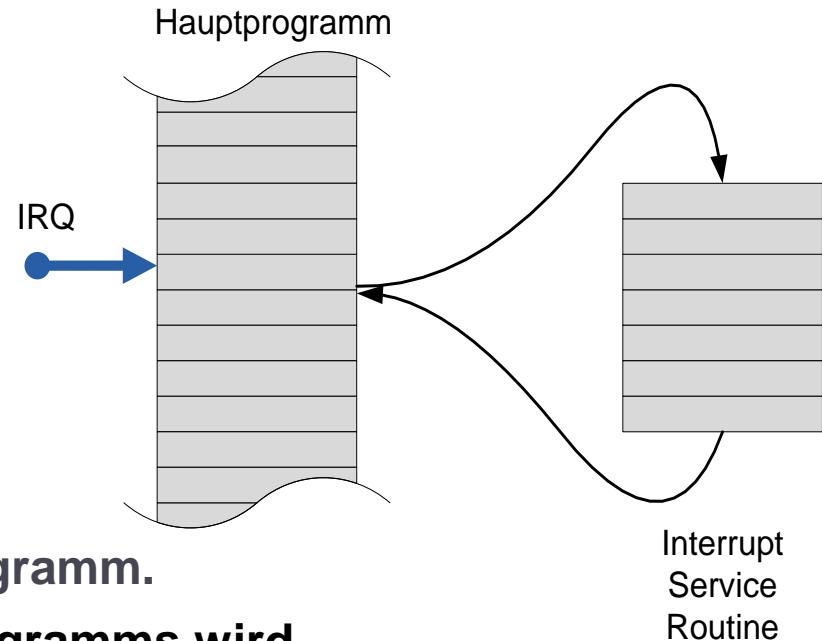
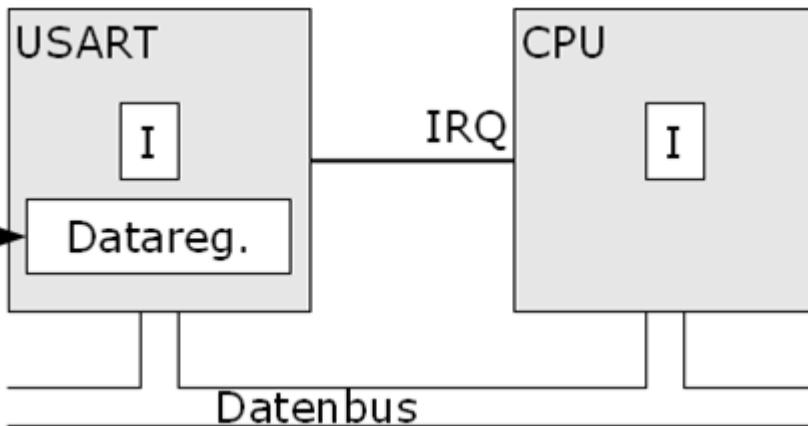
- ▶ Regelmäßige Abfrage des Statusregisters z.B.: „Receiver Data Ready“ oder „Transmitter Empty“
- ▶ Falls nichts zu tun ist: Task wird beendet
- ▶ Andernfalls werden die Daten empfangen oder gesendet und verarbeitet.
- ▶ Typisch:  
Empfangsdaten kommen mit fester Geschwindigkeit
  - ▶ Erfordert ausreichend kleine Reaktionszeit





# Interruptverarbeitung

## Interruptverarbeitung

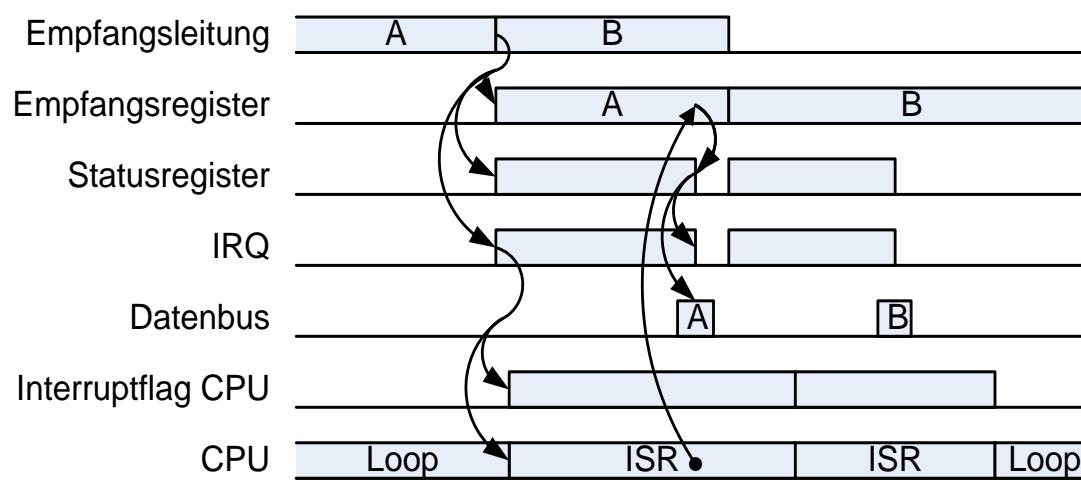
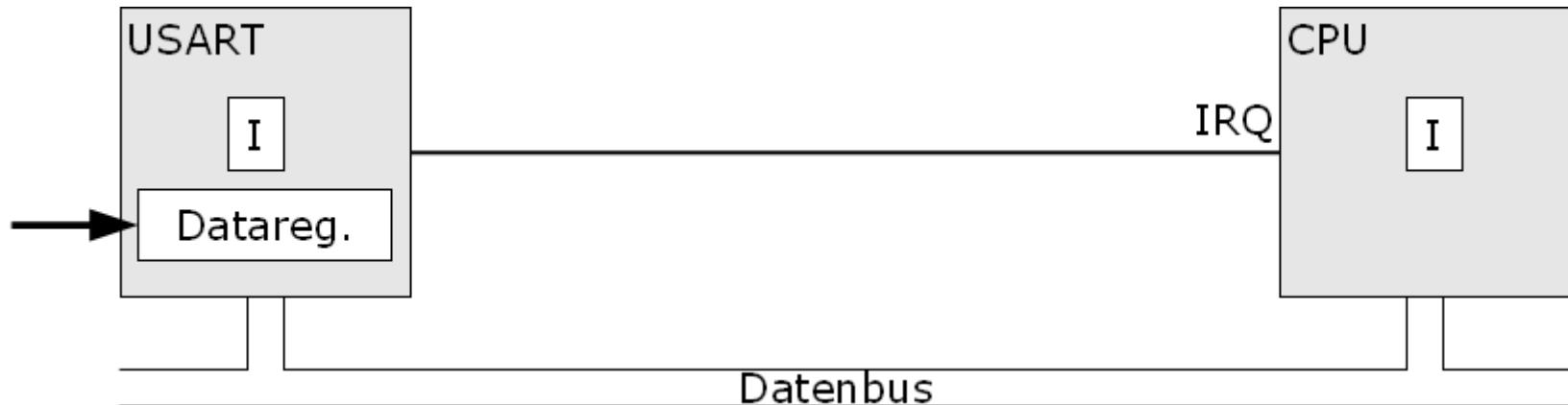


- ▶ **IRQ unterbricht laufendes Hauptprogramm.**
  - ▶ Aktuelle Instruktion des Hauptprogramms wird vollständig abgearbeitet.
  - ▶ Zustand der CPU (Kontext) wird gerettet.
  - ▶ Abarbeitung der Interrupt Service Routine (ISR).
  - ▶ Alter Zustand der CPU wird wieder hergestellt.
  - ▶ Fortsetzung des Hauptprogramms an unterbrochener Stelle.



# Interruptverarbeitung

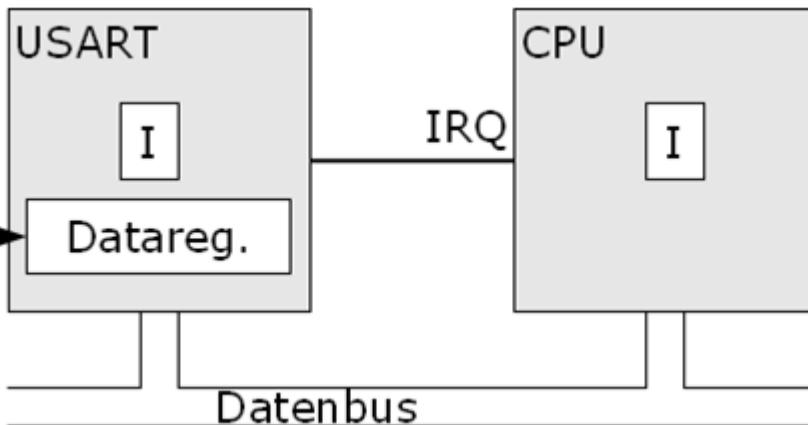
## Einzelne Interruptquelle



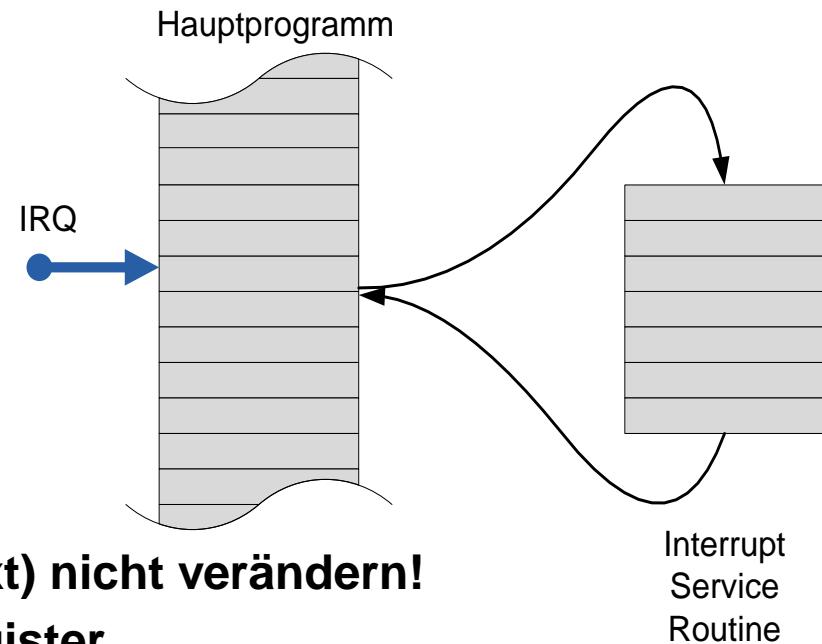


# Interruptverarbeitung

## Interruptverarbeitung



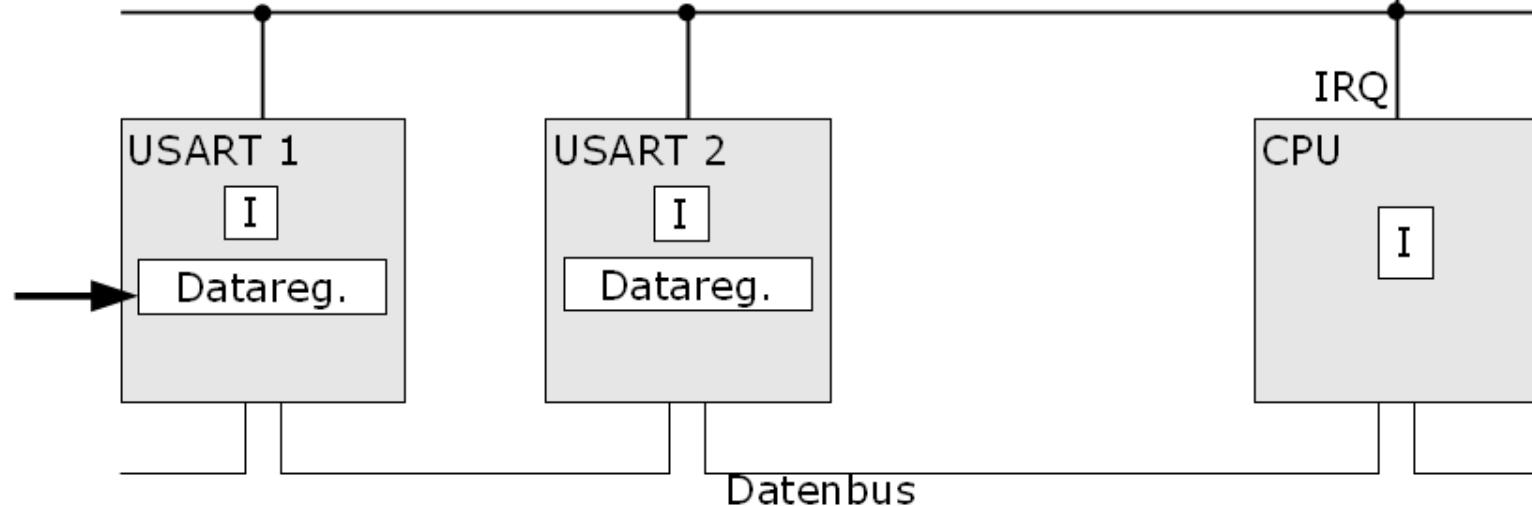
- ▶ **Wichtig:**
  - ▶ **ISR darf Zustand der CPU (Kontext) nicht verändern!**
  - ▶ **Alle von der ISR verwendeten Register**
    - **Statusregister, Datenregister müssen zuvor gerettet und hinterher wieder hergestellt werden.**
  - ▶ **Retten kann z.B. erfolgen:**
    - **auf dem Stack**
    - **in zusätzlichen Registersätzen.**





# Interruptverarbeitung

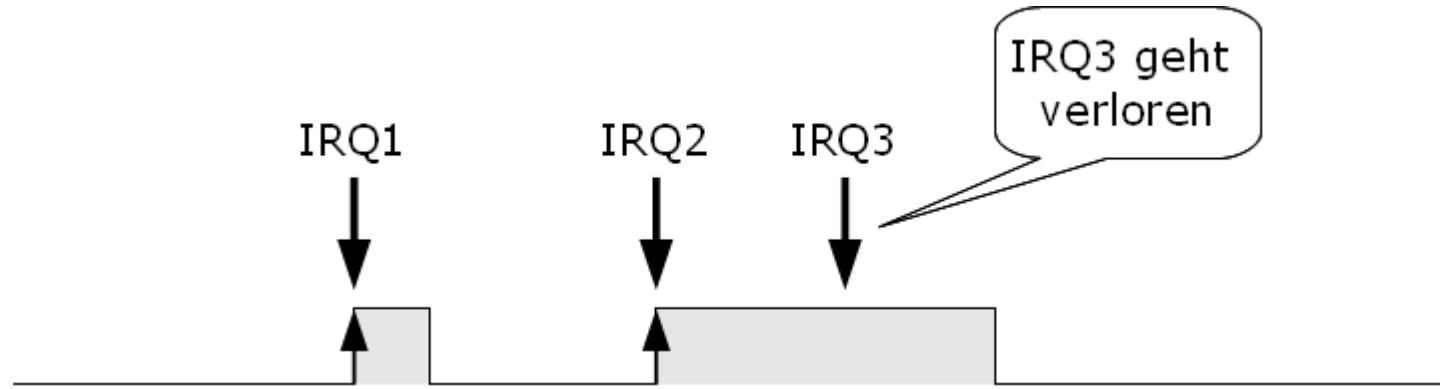
## Mehrere Interruptquellen



- ▶ Signalisierung über gemeinsame IRQ-Leitung (Open-Kollektor-Prinzip)
- ▶ Probleme:
  - ▶ Was passiert, wenn beide Bausteine gleichzeitig Interrupt auslösen?
  - ▶ Wie findet die CPU die aktive Interruptquelle?

# Interruptverarbeitung

## Flankensensitiver Interrupt

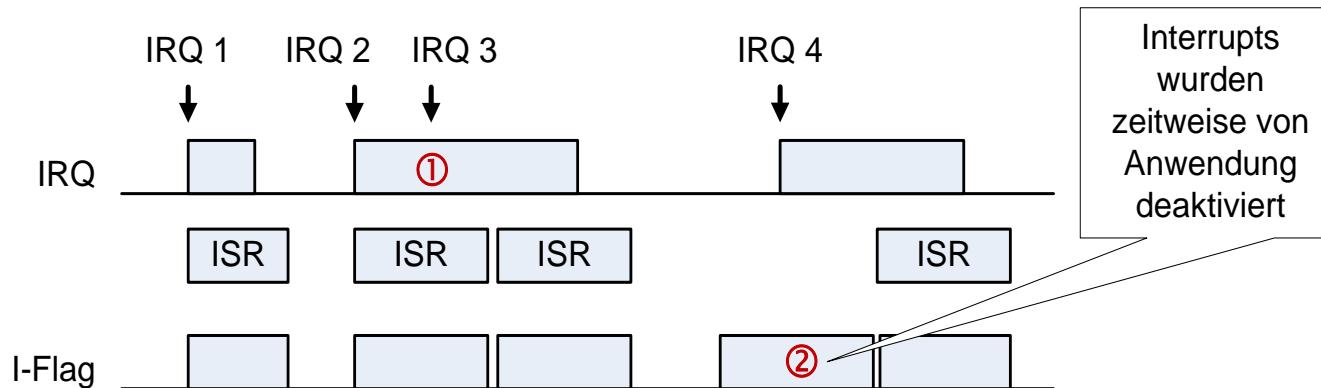


- ▶ CPU reagiert auf eine Flanke des Interruptsignals.
- ▶ Gut geeignet z.B. zum Zählen von Impulsen.
- ▶ Problematisch bei mehreren Interruptquellen, Interrupts können verlorengehen.
- ▶ Wird verwendet bei „Nicht maskierbaren Interrupts“ (NMI)



# Interruptverarbeitung

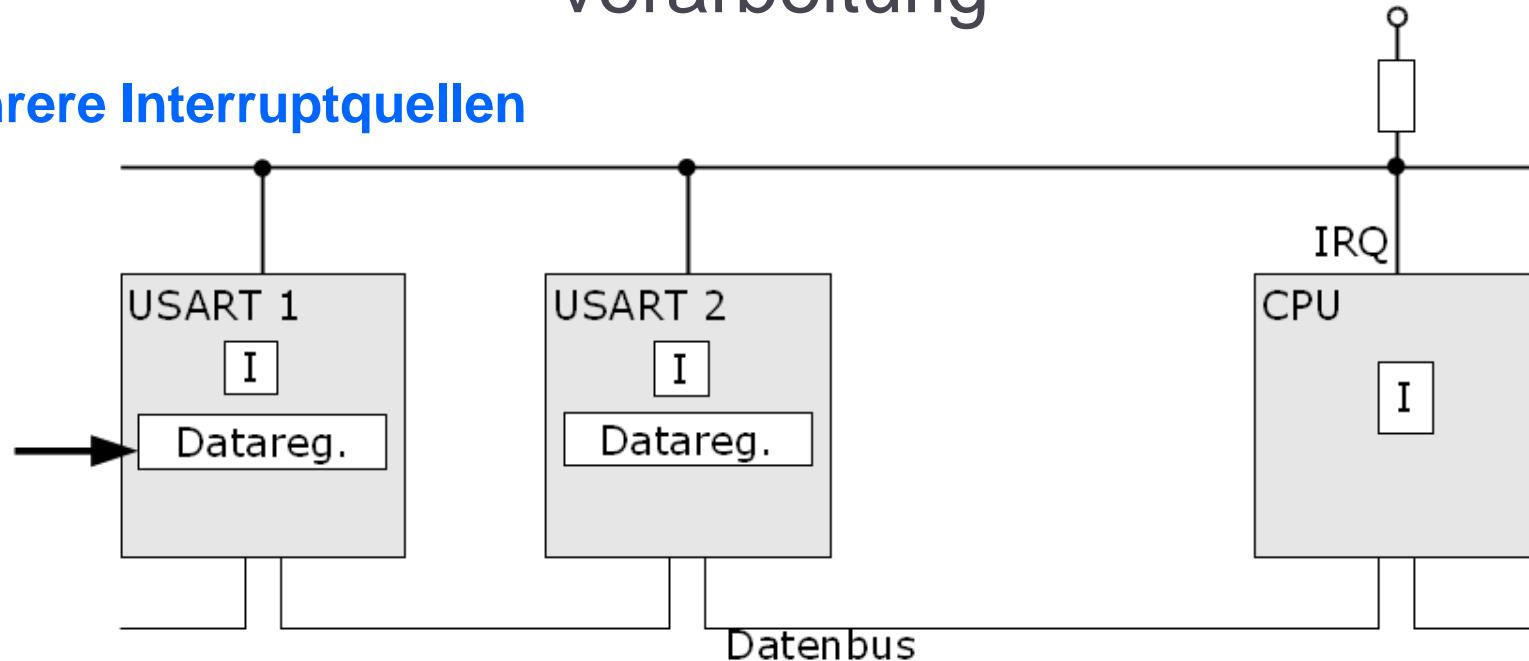
## Pegelsensitiver Interrupt



- ▶ CPU reagiert auf den **Pegel** des Interruptsignals.
- ▶ Erfordert **Interruptflag** zum Deaktivieren der IRQ-Logik.
  - ▶ Ohne I-Flag würde nach gestarteter ISR sofort die nächste Unterbrechung ausgelöst!
- ▶ Mehrere gleichzeitige Interruptanfragen gehen nicht mehr verloren. ①
- ▶ I-Flag gesetzt:
  - 9 Interrupts bleiben erhalten, bis Interruptlogik wieder aktiv wird. ②

# Interruptverarbeitung

## Mehrere Interruptquellen

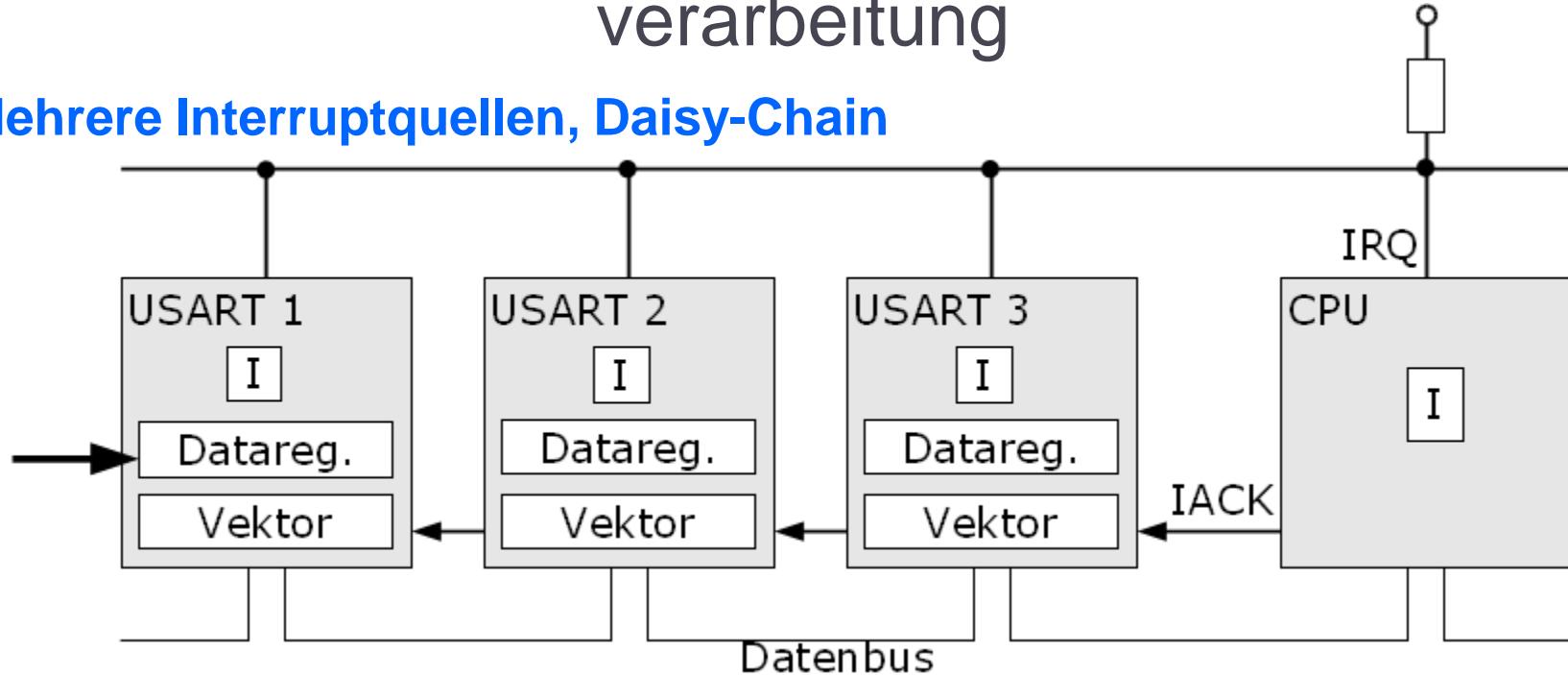


- ▶ Auswahl der ISR per Software:
  - ▶ Polling der Statusregister in den Peripheriebausteinen.
  - ▶ Reihenfolge (Priorität) kann per Software festgelegt werden.
  - ▶ Problematisch: Zeitaufwand zum Auffinden der Interruptquelle.



# Interruptverarbeitung

## Mehrere Interruptquellen, Daisy-Chain



- ▶ Bausteine sind in Daisy-Chain angeordnet
  - ▶ CPU-Hardware aktiviert IACK
  - ▶ Erster Baustein in der Kette mit aktiviertem IRQ:
    - IACK wird nicht weitergereicht
    - IRQ-Vektor wird auf Datenbus gelegt
  - ▶ Priorität wird durch Daisy-Chain festgelegt.



# Interruptverarbeitung

## Interruptvektor

- ▶ Häufig: Integerzahl, z.B. 8-Bit.
- ▶ Wird während der Initialisierung der Bausteine in das entsprechende Register geschrieben.
- ▶ Verweist auf einen Eintrag der **Vektortabelle**
- ▶ Vektortabelle kann enthalten:
  - ▶ Startadresse der ISR oder
  - ▶ ersten Befehl der ISR (meist Sprungbefehl zur eigentlichen Routine).
- ▶ Durchführung des IACK-Zyklusses und Auswertung der Vektortabelle meist durch Hardware der CPU



# Interruptverarbeitung

## Beispiel: Vektortabelle PC

- ▶ **Vektortabelle** enthält Startadressen der ISR (32-Bit).
- ▶ **Hardwareinterrupts** sind IRQ0 bis IRQ15
- ▶ **Zusätzlich sind sogenannte Ausnahmen enthalten, z.B.:**
  - ▶ **Teilen durch 0**
  - ▶ **Unbekannter Befehl**

Nr.	Adresse	Belegung
0	000-003	CPU: Division durch 0
1	004-007	CPU: Einzelschritt
2	008-00B	CPU: NMI (Fehler in RAM-Baustein)
3	00C-00F	CPU: Breakpoint erreicht
4	010-013	CPU: numerischer Überlauf
5	014-017	Hardcopy
6	018-01B	unbekannter Befehl (nur 80286)
7	01D-01F	reserviert
8	020-023	IRQ0: Timer (Aufruf 18,2 mal/s)
9	024-027	IRQ1: Tastatur
0A	028-02B	IRQ2: zweiter IR-Baustein 8259 (nur AT)
0B	02C-02F	IRQ3: serielle Schnittstelle 2
0C	030-033	IRQ4: serielle Schnittstelle 1
0D	034-037	IRQ5: Festplatte
0E	038-03B	IRQ6: Diskette
0F	03C-03F	IRQ7: Drucker
....	....	.....
68-6F	1A0-1BF	frei für Anwendungsprogramme
70	1C0-1C3	IRQ08: Echtzeituhr (nur AT)



# Interrupt-verarbeitung

## Beispiel: Vektortabelle AVR-Familie (8-Bit CPU)

- ▶ **Vektortabelle ersten Befehl der ISR (32-Bit).**
  - ▶ **meist Sprungbefehl zur eigentlichen ISR**

Vector No.	Program Address <sup>(2)</sup>	Source	Interrupt Definition
1	0x0000 <sup>(1)</sup>	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	INT2	External Interrupt Request 2
5	0x0008	INT3	External Interrupt Request 3
6	0x000A	INT4	External Interrupt Request 4
7	0x000C	INT5	External Interrupt Request 5
8	0x000E	INT6	External Interrupt Request 6
9	0x0010	INT7	External Interrupt Request 7
10	0x0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	0x0014	TIMER2 OVF	Timer/Counter2 Overflow
12	0x0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	0x0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	0x001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	0x001C	TIMER1 COMPC	Timer/Counter1 Compare Match C
16	0x001E	TIMER1 OVF	Timer/Counter1 Overflow
17	0x0020	TIMER0 COMP	Timer/Counter0 Compare Match
18	0x0022	TIMER0 OVF	Timer/Counter0 Overflow
19	0x0024	CANIT	CAN Transfer Complete or Error
20	0x0026	OVRIT	CAN Timer Overrun
21	0x0028	SPI, STC	SPI Serial Transfer Complete

# Interruptverarbeitung



## Mehrere Interruptquellen

nIRQ6



nIRQ4



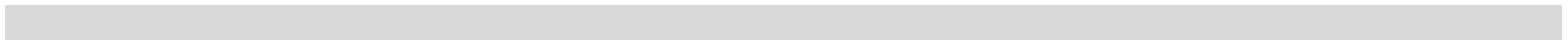
nIRQ2



nIRQ



nMaske



ISR 6



ISR 4



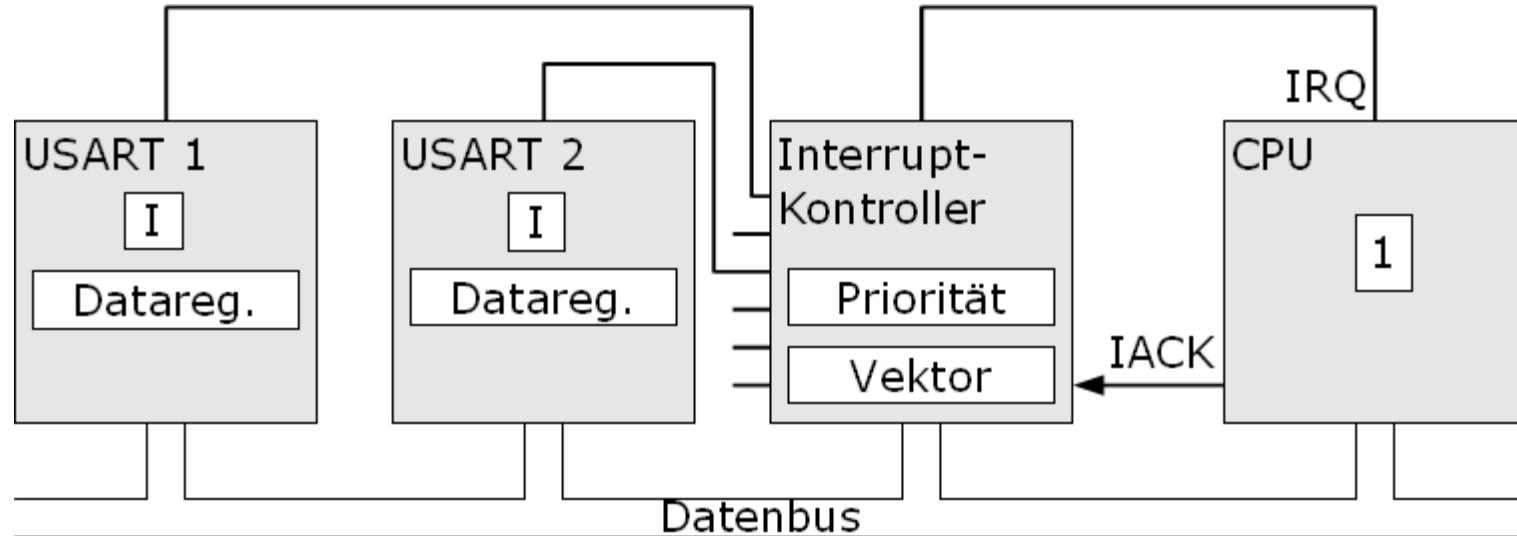
ISR 2





# Interruptverarbeitung

## Mehrere Interruptquellen

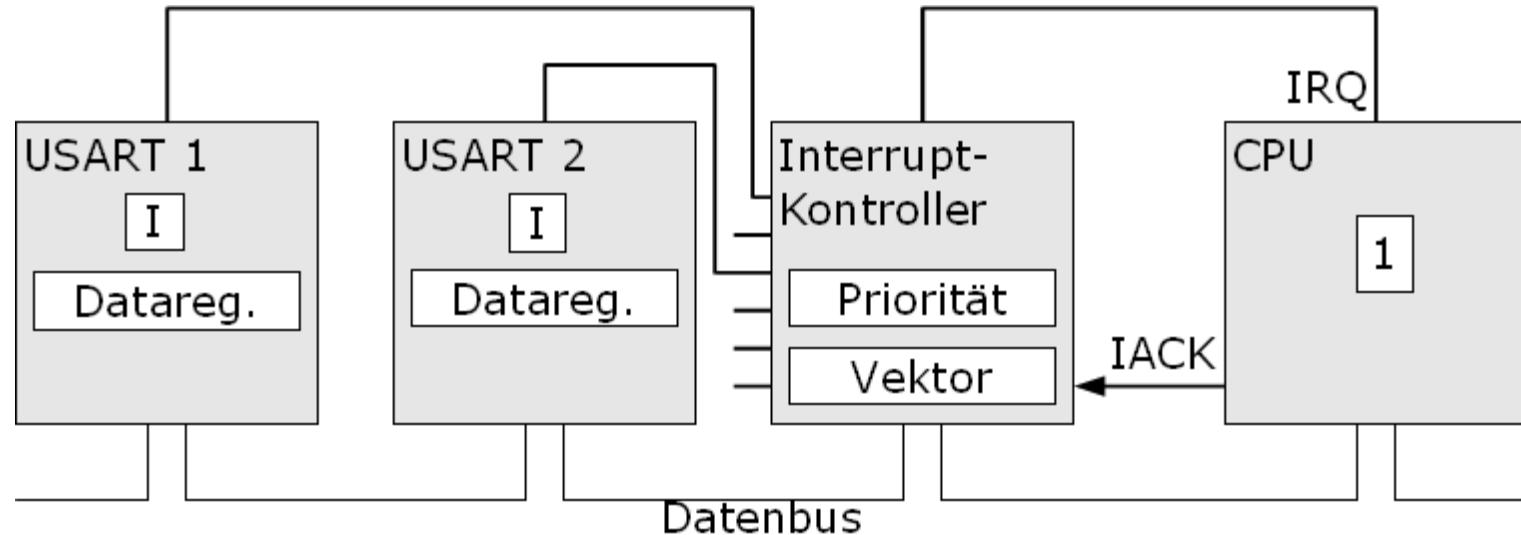


- ▶ **Auswahl mit Interrupt-Controller:**
  - ▶ Ermöglicht Verwendung von Standard-Bausteinen
  - ▶ Priorität wird vom Controller festgelegt
    - Unterschiedliche Strategien: Feste Prioritäten, Round Robin
- ▶ **Controller ermöglicht preemptive Interruptbehandlung:**
  - IRQ mit höherer Priorität können laufende ISR unterbrechen



# Interruptverarbeitung

## Mehrere Interruptquellen



- ▶ Preemptive Interruptbehandlung
  - ▶ Steuerung mit einem zusätzlichem Prioritätsregister
    - Register enthält Priorität des Interrupts, der gerade bearbeitet wird.
  - ▶ Controller blockiert alle Interrupts mit kleinerer oder gleicher Priorität.
  - ▶ Interrupts mit höherer Priorität werden weitergereicht.
  - ▶ Zum Zurücksetzen der Prio muss Controller Ende der ISR erkennen:
    - Meist mittels zusätzlichem Ausgabebefehls von CPU

# Interruptverarbeitung



## Mehrere Interruptquellen

nIRQ6



nIRQ4



PrioReg



nIRQ



nMaske



ISR 6



ISR 4



# Interrupt-verarbeitung

## Volatile und globale Variable

- ▶ mit volatile werden Variable gekennzeichnet, deren Wert sich außerhalb des aktuellen Programmpfades ändern kann, z.B.:
  - ▶ in einer ISR.
  - ▶ Hardware-Register, z.B. Timer.
- ▶ Kennzeichnung verhindert eine übermäßige Optimierung, z.B.:
  - ▶ Entfernen einer solchen Variable aus einer Schleife.

### Interrupt-Service-Routine:

```
volatile int counter;

void __attribute__ ((interrupt("IRQ"))) little_isr(void) {
    ....
    counter++;
    ....
}
```

### Hauptschleife:

```
.....
while( 1 ){
    printf( "Zaehler ist: %d\n", counter );
}
.....
```



# Interrupt-verarbeitung

## Wiedereintrittsfeste Bibliotheksfunktionen

### Nicht wiedereintrittsfeste Bibliotheksfunktion:

```
char buf[100];
char* textUmkehr( char* str){
    int i;
    for( i=0,j=strlen(str)-1; str[i]!='\0'; i++,j-- ){
        buf[j] = str[i];
    }
    return buf;
}
```

### Interrupt-Service-Routine:

```
volatile char* rev;
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {
    .....
    rev = textUmkehr("Heitmann");
    .....
}
```

### Hauptschleife:

```
.....
while( 1 ){
    printf( "Reverse Text ist: %s\n", textUmkehr( "HAW" ) );
}
.....
```



# Interrupt-verarbeitung

## Wiedereintrittsfeste Bibliotheksfunktionen

### Wiedereintrittsfeste Bibliotheksfunktion:

```
void textUmkehr( char* str, char* ergebnis){  
    int i;  
    for( i=0, j=strlen(str)-1; str[i]!='\0'; i++,j-- ){  
        ergebnis[j] = str[i];  
    }  
}
```

### Interrupt-Service-Routine:

```
volatile char rev[80];      //warum soll und darf dies keine lokale Variable sein?  
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {  
    ....  
    textUmkehr("Heitmann", rev);  
    ....  
}
```

### Hauptschleife:

```
....  
while( 1 ){  
    char rev[80];  
    textUmkehr( "HAW", rev );  
    printf( "Reverse Text ist: %s\n", rev );  
}  
....
```

# Interruptverarbeitung



## Wiedereintrittsfeste Bibliotheksfunktionen

- ▶ Bibliotheksfunktionen, die von der Hauptschleife und von der ISR aufgerufen werden, müssen wiedereintrittsfähig (reentrant) sein.
  - ▶ **printf und malloc sind es nicht.**
- ▶ Kritisch:
  - ▶ **Schreibzugriffe auf statische oder globale Variable.**
  - ▶ **Rückgabe von Adressen auf statische oder globale Variable.**
  - ▶ **Aufrufe von non-reentrant Funktionen.**
- ▶ Bibliotheksfunktionen sollten mit
  - ▶ **lokalen Variablen oder**
  - ▶ **Daten, die vom Aufrufer zur Verfügung gestellt werden**arbeiten.



# Interrupt-verarbeitung

## Synchronisation zwischen ISR und Hauptschleife

- ▶ Z.B. zu beachten bei „gleichzeitigen“ Schreibzugriffen von
  - ▶ ISR und
  - ▶ Hauptschleifeauf eine globale Variable:

### Interrupt-Service-Routine:

```
volatile int counter;
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {
    .....
    counter++;
    .....
}
```

### Hauptschleife:

```
.....
while( 1 ){
    if( counter >= COUNTERMAX ){
        <mach etwas>
        counter = 0;
    }
}
.....
```



# Interrupt-verarbeitung

## Synchronisation zwischen ISR und Hauptschleife

- ▶ Z.B. zu beachten bei „gleichzeitigen“ Schreibzugriffen von
  - ▶ ISR und
  - ▶ Hauptschleifeauf eine globale Variable.
  
- ▶ Unkritisch, wenn der Zugriff „atomar“ erfolgt  
(er besteht aus einer einzigen Maschineninstruktion).

```
//atomar:  
counter = 55;           //ok, wenn int und 32-Bit CPUs  
localvalue = counter;  
  
//nicht atomar:  
counter++;             //Bei RISC_CPUs: Read-Modify-Write Zyklus  
counter |= (1<<bitnr);
```

# Interruptverarbeitung

## Synchronisation zwischen ISR und Hauptschleife

- ▶ Wenn Zugriff nicht atomar, Synchronisationsmechanismen verwenden:
  - ▶ Bei Unterstützung durch ein Betriebssystem:
    - Semaphore
    - Mutex
    - Monitor
  - ▶ Ohne Betriebssystem:
    - Für die Dauer des Zugriffs Interruptverarbeitung deaktivieren:
      - Nur notwendig in Hauptschleife.
      - ISR ist in der Regel nicht unterbrechbar.

### Hauptschleife:

```
#include <armVIC.h>
.....
while( 1 ){
    disableIRQ();
    if( counter >= COUNTERMAX ){
        <mach etwas>
        counter = 0;
    }
    enableIRQ();
    .....
}
```

# Interruptverarbeitung

## Synchronisation zwischen ISR und Hauptschleife

- ▶ Ohne Betriebssystem:
  - ▶ Für die Dauer des Zugriffs Interruptverarbeitung deaktivieren.
  - ▶ Deaktivierung bedeutet Verlängerung der Latenzzeit, daher
    - Ausschaltdauer möglichst kurz halten!

### Hauptschleife:

```
.....
while( 1 ){
    int merker = 0;
    disableIRQ();
    if( counter >= COUNTERMAX ){
        merker = 1;
        counter = 0;
    }
    enableIRQ();
    if( merker ){
        <mach etwas>
    }
    .....
}
```

# Interruptverarbeitung



## Synchronisation zwischen ISR und Hauptschleife

- ▶ Ohne Betriebssystem:
  - ▶ Für die Dauer des Zugriffs Interruptverarbeitung deaktivieren.
  - ▶ Bei Verwendung in Bibliotheksfunktionen aufpassen, dass Interrupt nicht versehentlich eingeschaltet wird.

### Hauptschleife:

```
....  
while( 1 ){  
    int merker = 0;  
    int oldirq = disableIRQ();  
    if( counter >= COUNTERMAX ) {  
        merker = 1;  
        counter = 0;  
    }  
    restoreIRQ( oldirq );  
    if( merker ){  
        <mach etwas>  
    }  
    ....  
}
```

# Interrupt-verarbeitung



## Ein- und Ausschalten von Interrupts in der Hitex Laufzeitumgebung

- ▶ Funktionen in armVIC.h:

**unsigned disableIRQ(void);**

**unsigned enableIRQ(void);**

**unsigned restoreIRQ(unsigned oldCPSR);**

**unsigned disableFIQ(void);**

**unsigned enableFIQ(void);**

**unsigned restoreFIQ(unsigned oldCPSR);**

# Interruptverarbeitung

## System mit kurzen Reaktionszeiten

- ▶ **Latenzzeit:**  
Zeit, die zwischen dem Auftreten eines Ereignisses und bis zur Bearbeitung des Ereignisses vergeht.
- ▶ Oft gefordert: Latenzzeit soll vorgegebene Grenze nicht überschreiten:
  - ▶ **Beispiel: Serielle Schnittstelle.**
  - ▶ Bei nicht unterbrechbaren ISRs ergibt sich die Latenzzeit des Systems aus der Bearbeitungsdauer der langsamsten ISR.
  - ▶ Interruptroutinen müssen daher möglichst schnell abgearbeitet werden.
    - ▶ In ISRs sollte nicht gewartet werden, z.B.:
      - Warten auf Timer.
      - Warten auf Fertig-Bit einer Hardware-Komponente.
      - Eingabeaufforderung an Benutzer.
    - ▶ Besonders wichtig bei vielen Interruptquellen.



# Computer Engineering WS 2012

Interruptverarbeitung  
im LPC 2468

HTM – SHF - SWR

# Ausnahme- und Interruptverarbeitung im LPC 2468

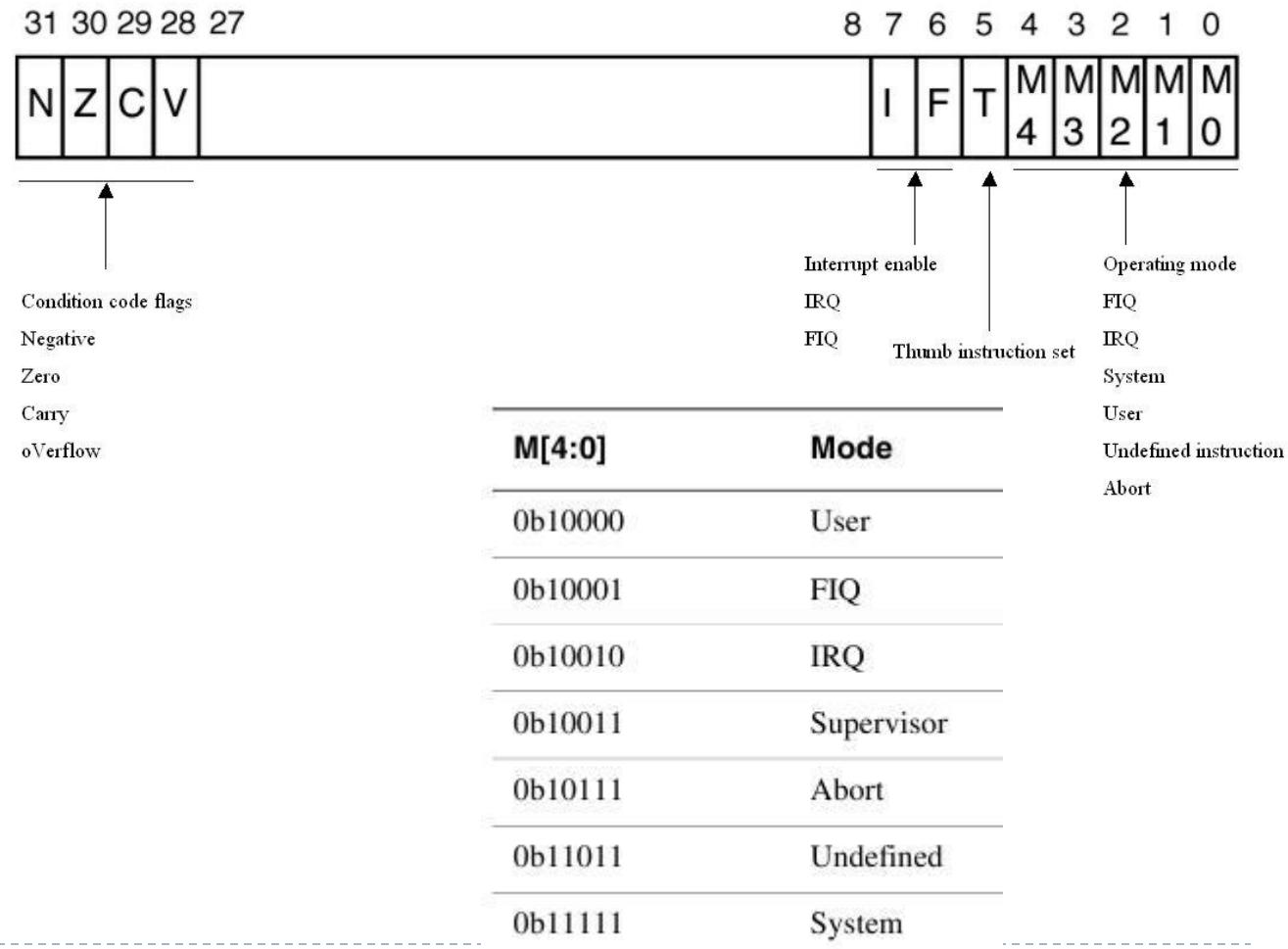
## Betriebsarten, Statuswort

- ▶ ARM 7 TDMI unterscheidet 7 Betriebsarten.
- ▶ **User Mode** ist die normale Betriebsart.
- ▶ Wechsel in andere Betriebsarten durch **Exceptions (Ausnahmen)**.
- ▶ Momentane Betriebsart steht im „**Current Program Status Register**“ (**CPSR**)
- ▶ Betriebsarten haben:
  - ▶ **eigene Stacks**
  - ▶ **eigene Register** (z.B. **LR** und **SP**) (siehe nächste Folie)
  - ▶ „**Saved Program Status Register**“ (**SPSR**)  
(beim Wechsel der Betriebsart wird hier das **CPSR** gespeichert)
  - ▶ **unterschiedliche Rechte bei Zugriff auf Systemregister**  
(z.B. im User Mode kann nicht der **SP** oder das **SPSR** verändert werden)

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Betriebsarten

## Statuswort





# Ausnahme- und Interruptverarbeitung im LPC 2468

## Betriebsarten Statuswort

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

### HITOP:

Register werden angezeigt mit

View→

SFR Window →  
ARM Processor Register

CPSR	CPSR SPSR_fiq	CPSR SPSR_svc	CPSR SPSR_abt	CPSR SPSR_irq	CPSR SPSR_und

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Betriebsarten, Statuswort

- ▶ Wie kann man auf R8 der Betriebsart FIQ zugreifen?

1. Momentane Betriebsart in R1 merken.
2. Auf Betriebsart FIQ umschalten
3. Registerinhalt von R8 nach R0 schreiben
4. Zurückschalten auf ursprüngliche Betriebsart

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Ausnahmen

- ▶ Wenn eine Ausnahme auftritt:
  - ▶ CPU rettet CPSR nach SPSR und ändert Betriebsart
  - ▶ PC wird auf Eintrag in Vektortabelle gesetzt:

Exception	Mode	Address
Reset	Supervisor	0x00000000
Undefined Instruction	Undefined	0x00000004
Software Interrupt (SWI)	Supervisor	0x00000008
Prefetch Abort (instruction fetch memory abort)	Abort	0x0000000C
Data Abort (data access memory abort)	Abort	0x00000010
IRQ (interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Ausnahmen

- ▶ Rückkehr von Ausnahme:
  - ▶ CPU hat dafür (im Gegensatz zu vielen anderen) keinen speziellen Befehl.
  - ▶ Daher müssen PC und CPSR mit regulären ARM-Assembler Befehlen geändert werden.

Exception	Mode	Return Statement
Reset	Supervisor	-
Undefined Instruction	Undefined	SUBS PC,LR,#4
Software Interrupt (SWI)	Supervisor	MOVS PC,LR
Prefetch Abort (instruction fetch memory abort)	Abort	SUBS PC,LR,#4
Data Abort (data access memory abort)	Abort	SUBS PC,LR,#8
IRQ (interrupt)	IRQ	SUBS PC,LR,#4
FIQ (fast interrupt)	FIQ	SUBS PC,LR,#4



# Ausnahme- und Interruptverarbeitung im LPC 2468

## Ausnahmeroutinen

### ► HITOP Entwicklungsumgebung:

#### startup.s

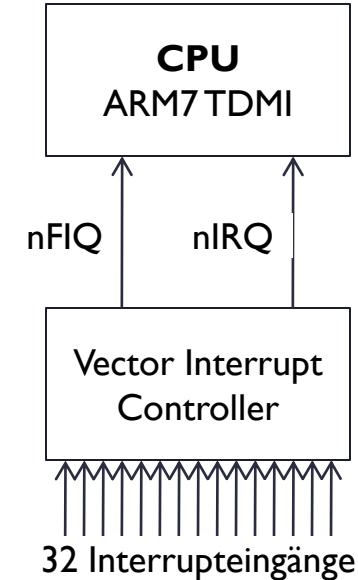
- definiert Standard-Ausnahmeroutinen, trägt die Aufrufe in die Vektortabelle ein und
- richtet die Stacks für die Betriebsarten ein.

Exception	Mode	Behandlung
Reset	Supervisor	Startup-Code, Aufruf von main
Undefined Instruction	Undefined	Endlosschleife
Software Interrupt (SWI)	Supervisor	Rahmen für eigene Routinen
Prefetch Abort (instruction fetch memory abort)	Abort	Endlosschleife
Data Abort (data access memory abort)	Abort	Endlosschleife
IRQ (interrupt)	IRQ	spezielle Behandlung
FIQ (fast interrupt)	FIQ	Endlosschleife

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Interruptverarbeitung

- ▶ ARM-CPU:
  - ▶ 2 Interruptrequest-Eingänge: Fast Interrupt (FIQ) und Interrupt (IRQ)
- ▶ LPC2468:
  - ▶ Zusätzlich „Vector Interrupt Controller“ (VIC)
  - ▶ Jede Interruptquelle des Chips ist fest mit einem bestimmten Eingang des VIC verbunden.
  - ▶ Per Software kann jede Interruptquelle auf FIQ oder vectored IRQ eingestellt werden.
  - ▶ Idealerweise sollte nur eine Interruptquelle auf FIQ gestellt werden.
  - ▶ Die restlichen Quellen benutzen vectored IRQ. Hierfür kann die Reihenfolge der Abarbeitung mittels 16 Prioritätsstufen festgelegt werden.



# Ausnahme- und Interruptverarbeitung im LPC 2468

## Interruptquellen des LPC 2468

Bit	31	30	29	28	27	26	25	24
Symbol	I2S	I2C2	UART3	UART2	TIMER3	TIMER2	GPDMA	SD/MMC
Bit	23	22	21	20	19	18	17	16
Symbol	CAN1&2	USB	Ethernet	BOD	I2C1	AD0	EINT3	EINT2/ LCD <sup>(1)</sup>
Bit	15	14	13	12	11	10	9	8
Symbol	EINT1	EINT0	RTC	PLL	SSP1	SPI/SSP0	I2C0	PWM0&1
Bit	7	6	5	4	3	2	1	0
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT

# Ausnahme- und Interruptverarbeitung im LPC 2468

## FIQ

- ▶ Aktionen zum Eintreten in die ISR
  1. Speichern der Adresse der nächsten Instruktion (**PC**) in **LR**
  2. Speichern von **CPSR** in **SPSR**
  3. Setzen der **Betriebsart FIQ** in **CPSR**, Registersatz wird umgeschaltet
  4. Setzen des F-Bits in **CPSR (FIQ aus)**
  5. Setzen des **PC** auf **FIQ-Adresse der Vektortabelle**
- ▶ Maximale Dauer bis Start ISR (Interruptlatenzzeit) setzt sich zusammen:

1. Synchronisation des externen Signals	max 4 Takte
2. Abarbeiten der aktuellen Instruktion: Längste Instruktion ist LDM mit allen Registern	max 20 Takte
3. Möglicher Weise findet gerade eine Data Abort Exception statt (hat höhere Priorität):	3 Takte
4. FIQ-Aktivierung	2 Takte

Insgesamt: 29 Takte, bei 48 MHz also ca. 0.6 µsec

# Ausnahme- und Interruptverarbeitung im LPC 2468

## FIQ

- ▶ Aktionen zum Verlassen der ISR
  - 1. Speichern des **LR**, verkleinert um 4, in den **PC**.
  - 2. Speichern von **SPSR** in das **CPSR**.  
Damit automatisch:
    - Umschalten der Betriebsart und
    - Reaktivierung des FIQs.
- ▶ Beide Aktionen lassen sich wie folgt durchführen:

```
SUBS    PC, LR, #4      @ ISR verlassen
```

Basiert auf Sonderfunktion:

Wenn S bei Datenmanipulationsbefehlen gesetzt und  
das Zielregister PC ist,  
dann wird SPSR nach CPSR kopiert!

# Ausnahme- und Interruptverarbeitung im LPC 2468

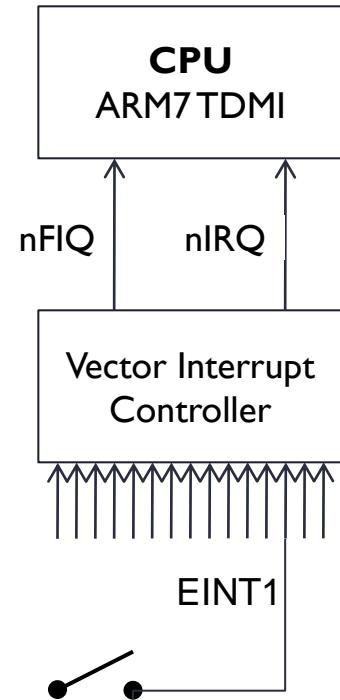
## FIQ-ISR: Eigenschaften

- ▶ CPU: schaltet Register R8 bis R14 um,  
können also von ISR beliebig verändert werden.  
  
rettet Statusregister und Rücksprungadresse.
- ▶ Änderungen an R0 bis R7 müssen am Ende der ISR  
durch die ISR rückgängig gemacht werden!
- ▶ Vor Aufruf eines Unterprogramms muss LR gerettet werden!
- ▶ Unterprogramme, die vom Hauptprogramm und von der ISR aufgerufen  
werden, müssen wiedereintrittsfest (**reentrant**) sein:  
  
Eine Funktion ist reentrant, wenn sie mehrmals gleichzeitig aktiv sein kann,  
ohne dass sich diese Aufrufe gegenseitig beeinflussen.
  - ▶ Mehrfachaufrufe passieren häufig bei Bibliotheksfunktionen (`strcpy`).
  - ▶ `printf` und `malloc` sind meist nicht reentrant!

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Beispiel: Auslösung eines FIQ durch eine Taste

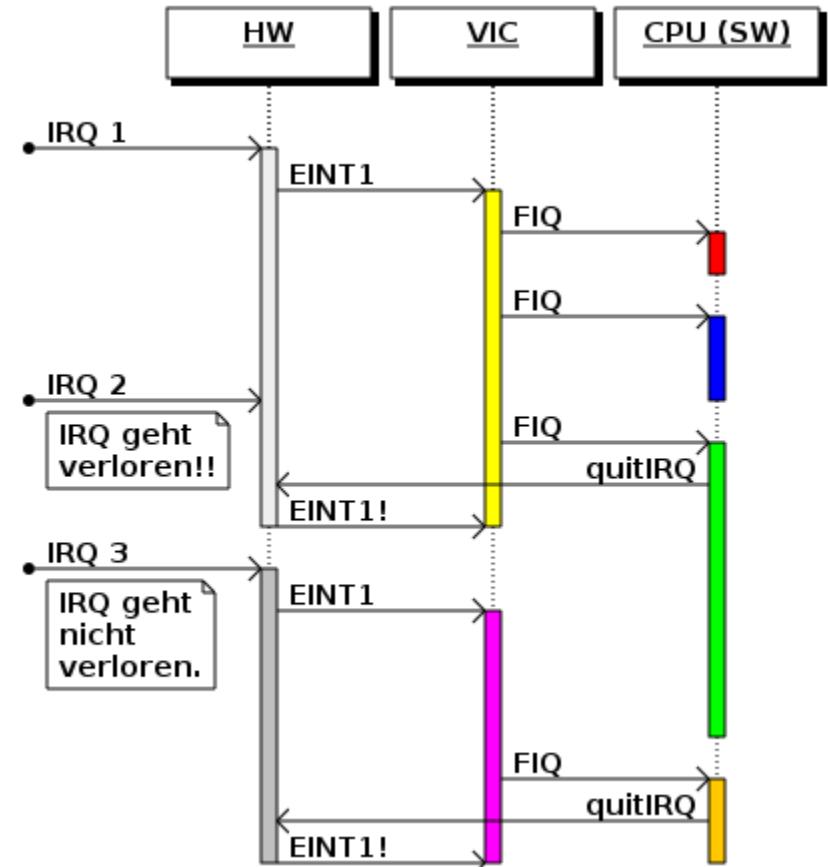
- ▶ FIQ-Serviceroutine : Zählen der Tastenbetätigungen.
- ▶ Verwendung des „Externen Interrupt 1“ (EINT1)
  - ▶ Auslösung von EINT1 durch externes digitales Signal.
  - ▶ EINT1 kann eingestellt werden auf:
    - Pegelsensitiv, Auslösung bei High-Pegel
    - Pegelsensitiv, Auslösung bei Low-Pegel
    - Flankensensitiv, Auslösung bei steigender Flanke
    - Flankensensitiv, Auslösung bei fallender Flanke



# Ausnahme- und Interruptverarbeitung im LPC 2468

## Beispiel „Zählen von externen Ereignissen“: Ablauf

- ▶ Interrupts müssen in der Regel quittiert werden.
- ▶ Ohne Quittierung:  
Nach Beendigung der ISR wird sie sofort erneut aktiviert!  
→ System ist dauerhaft blockiert
- ▶ Interrupts können verlorengehen
- ▶ daher:  
Quittierung des Interrupts möglichst früh durchführen!



# Ausnahme- und Interruptverarbeitung im LPC 2468

## Beispiel „Zählen von externen Ereignissen“: ISR

1. Interrupt muss quittiert werden
  - ▶ R10 enthält die Adresse des Ports **EXTINT**
  - ▶ R9 die notwendige Maske

Beides wurde während der Initialisierung in die FIQ-Register geladen

2. Zähler erhöhen, R8 enthält aktuellen Zählerwert.
3. Interruptserviceroutine beenden.

```
fiq:    STR      R10,[R9]        @ Interrupt quittieren
          ADD      R8,R8,#1       @ Zaehler erhoehen
          SUBS    PC,LR,#4       @ ISR verlassen
```

# Ausnahme- und Interruptverarbeitung im LPC 2468

Beispiel „Zählen von externen Ereignissen“: **Initialisierung der ISR**

```
.section .text
.global fiq,fiqinitasm

.equ EXTINT,0xE01FC140

fiqinitasm:                                @ Funktion kann nur in Betriebsart
                                                @ Supervisor aufgerufen werden!

    MRS      R0, CPSR          @ aktuelle Betriebsart retten
    MSR      CPSR_c,#0xD1     @ auf FIQ-Betriebsart umschalten
                            @ FIQ und IRQ aus

    MOV      R8,#0             @ Zahler loeschen
    LDR      R9,=EXTINT       @ Adr von EXTINT laden
    MOV      R10,#1<<1        @ Maske fuer EINT1 laden

    BIC      R0,#1<<6         @ FIQ aktivieren: FIQ-Bit im CPSR loeschen
    MSR      CPSR_c,R0         @ auf urspruengliche Betriebsart schalten

    MOV      PC,LR             @ Unterprogramm verlassen
```

# Ausnahme- und Interruptverarbeitung im LPC 2468

## C-Programm: Auslösung eines FIQ durch eine Taste

```
extern void fiqinitasm(void);
```

```
void fiqinit(void) {
```

```
PINSEL4 |= 1<<22; //P2.11 is EINT1
```

```
EXTMODE = 1<<1; //edge sensitivity  
EXTPOLAR = 0; //Falling edge
```

Externer  
Interrupt

```
VICIntSelect |= 1<<15; //EINT1 is FIQ  
VICIntEnable |= 1<<15; //Enable EINT1
```

Vector Interrupt  
Controller

```
fiqinitasm(); //Restliche Initialisierung  
//mit Assemblerprogramm
```

# Ausnahme- und Interruptverarbeitung im LPC 2468

## ISR in C programmiert

- ▶ Nicht im C-Standard vorgesehen!
- ▶ Implementierung abhängig von
  - ▶ CPU-Hersteller
  - ▶ Compiler-Hersteller
- ▶ Nachfolgend: Implementierung von ISR mittels GNU C-Compiler (gcc) für ARM-CPUs.
- ▶ Im gcc können Funktionen zusätzliche Attribute erhalten:

```
void f () __attribute__ ((interrupt ("FIQ")));
void f () __attribute__ ((interrupt ("IRQ")));
```

- ▶ Compiler rettet alle notwendigen Register und restauriert sie am Ende der ISR.
- ▶ Return am Ende der Funktion wird ersetzt durch „Return from Interrupt“



# Ausnahme- und Interruptverarbeitung im LPC 2468

Beispiel „Zählen von externen Ereignissen“: **C-Version der ISR**

- Übersetzt mit -O2: Hohe Optimierung. Im Praktikum: -O0

C-Code

```
void __attribute__ ((interrupt("FIQ"))) cfiq(void) {  
  
    counter++;  
  
    EXTINT = 1<<1;  
  
}
```

Erzeugter Assemblercode

```
    stmfd    sp!, {r1, r2, r3}  
  
    ldr r2, .L3  
    ldr r3, [r2, #0]  
    add r3, r3, #1  
    str r3, [r2, #0]  
  
    mov r3, #-536870912  
    add r3, r3, #2080768  
    mov r1, #2  
    add r3, r3, #256  
    strb r1, [r3, #64]  
  
    ldmfd    sp!, {r1, r2, r3}  
    subs    pc, lr, #4
```



# Ausnahme- und Interruptverarbeitung im LPC 2468

Beispiel „Zählen von externen Ereignissen“: Eintrag in Vektortabelle

- ▶ Hitop: Vektortabelle ist in start.s definiert

```
_startup:  
# -----  
# Interrupt vector table at address 0  
# -----  
  
Reset_Vec:    LDR      PC, _ResetEntry  
               LDR      PC, _Undef_Addr  
               LDR      PC, _SWI_Addr  
               LDR      PC, _PAbt_Addr  
               LDR      PC, _DAbt_Addr  
               .word   CheckSum          /* Reserved Vector */  
               LDR      PC, [PC,#-0x120]  
               LDR      PC, _FIQ_Addr       /* Calling the FIQ handler */  
# -----  
  
_ResetEntry:   .word   ResetEntry  
_Undef_Addr:   .word   Undef_Handler  
_SWI_Addr:    .word   SWI_Handler  
_PAbt_Addr:   .word   PAbt_Handler  
_DAbt_Addr:   .word   DAbt_Handler  
_FIQ_Addr:    .word   fiq
```

Hier Startadresse  
der ISR eintragen

# Ausnahme- und Interruptverarbeitung im LPC 2468

Beispiel „Zählen von externen Ereignissen“: **Stack definieren**

- ▶ Hitop: Stack ist in start.s definiert
- ▶ Standardeinstellung für FIQ-Stacklänge: nur 8 Worte!

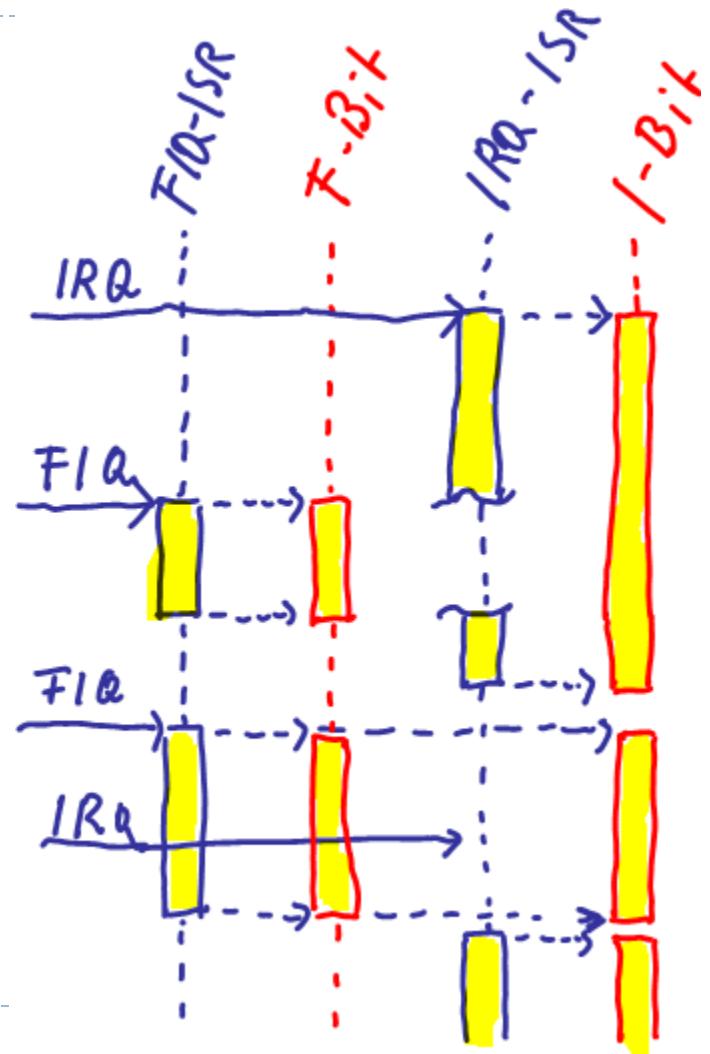
```
# Stack definitions
# size in words!
    .equ    UND_Stack_Size , 8
    .equ    SVC_Stack_Size , 1024
    .equ    ABT_Stack_Size , 8
    .equ    FIQ_Stack_Size , 8
    .equ    IRQ_Stack_Size , 512
```

Hier ausreichende  
Stacklänge eintragen

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ

- ▶ IRQ verhält sich innerhalb der CPU wie FIQ, hat aber eine geringere Priorität.
- ▶ Priorität entscheidet, welcher Interrupt als nächstes bearbeitet wird, wenn beide Interrupts gleichzeitig auftreten.
- ▶ Unterbrechbarkeit wird gesteuert durch Setzen und Löschen des I- und des F-Bits im Statusregister (siehe Ablaufdiagramm).
  - ▶ Aktivieren von IRQ setzt nur I-Bit.
  - ▶ Aktivieren von FIQ setzt I- und F-Bit.
- ▶ Damit kann FIQ eine laufende IRQ-ISR unterbrechen.
- ▶ Aber IRQ kann eine FIQ-ISR nicht unterbrechen!



# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ: Zuordnung der ISR

- ▶ VIC verwaltet für jeden Interrupteingang:
  - ▶ Startadresse der ISR
  - ▶ Priorität des Interrupts
- ▶ Aktivierung eines Interrupteingangs:
  - ▶ VIC kopiert zugeordnete Startadresse in das Register **VICVectAddr**.
  - ▶ VIC setzt internes Prioritätsregister auf Priorität des Interrupts.
  - ▶ VIC aktiviert IRQ Eingang der CPU.
- ▶ Ausnahmeverarbeitung der CPU liest **VICVectAddr** (Adr 0xFFFF FF00) und verzweigt direkt in die ISR.

Vektortabelle

0000	LDR	PC, _ResetEntry
0004	LDR	PC, _Undef_Adr
0008	LDR	PC, _SWI_Adr
000C	LDR	PC, _PAbt_Adr
0010	LDR	PC, _DAbt_Adr
0014	.word	CheckSum
0018	LDR	PC, [PC,#-0x120]
001C	LDR	PC, _FIQ_Adr

Lesen von  
VicVectAddr und  
Sprung in die ISR

/\* Reserved Vector \*/  
/\* Calling the IRQ handler \*/

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ: Prioritäten

- ▶ Nach Auslösen des IRQs:
  - ▶ Internes Register des VIC enthält **Priorität** des aktuellen Interrupts.
- ▶ Nachfolgende Interrupts mit **kleinerer oder gleicher** Priorität:
  - ▶ Werden von VIC nicht bearbeitet.
- ▶ Nachfolgende Interrupts mit **höherer** Priorität:
  - ▶ VIC aktualisiert sofort **VICVectAddr** und
  - ▶ aktiviert IRQ-Eingang der CPU
- ▶ Ermöglicht die Implementierung von **unterbrechbaren (preemptiven)** ISRs!
- ▶ Erfordert allerdings zusätzlichen Programmieraufwand:
  - ▶ **Frühzeitige Freigabe** des IRQs.
  - ▶ **Zusätzliche Verwaltungsmaßnahmen.**



# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ: Nicht unterbrechbare ISR

- ▶ Erfordert zwei wesentliche Aktivitäten:
  1. Quittierung des Interrupts beim Interrupt auslösenden Gerät:
    - Gerät deaktiviert IRQ.
  2. Quittierung des Interrupts beim VIC:
    - VIC setzt internes Prioritätenregister zurück.  
Sollte erst am Ende der ISR erfolgen.
- ▶ IRQ ist während der gesamten Abarbeitung gesperrt (**I-Bit gesetzt**):
  - ▶ ISR kann nicht unterbrochen werden (bzw. nur durch FIQ).

```
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {  
  
    EXTINT = 1<<1;                                /* Interrupt bestaetigen: */  
                                                /* Device deaktiviert IRQ-Leitung */  
  
    counter++;  
  
    VICVectAddr = 0;                                /* VIC mitteilen: ISR ist fertig */  
                                                /* VIC setzt internes Prio Register */  
                                                /* zurueck */  
}  
}
```

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ: Initialisierung von nicht unterbrechbaren ISRs

```
void irqinit(void){  
    PINSEL4          |= 1<<24;           // P2.12 is EINT2  
    EXTMODE         = 1<<2;            // edge sensivity  
    EXTPOLAR        = 0;                // Falling edge  
  
    VICIntSelect    &= ~(1<<16);       // EINT2 is IRQ  
    VICVectAddr16   = (int)little_isr; // EINT2 fires little_isr  
    VICVectPriority16 = 5;             // EINT2 gets Prio 5  
    VICIntEnable    |= 1<<16;          // Enable EINT2  
}
```

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ: Nichtunterbrechbare ISR, Latenzzeiten

- ▶ Es wurden 3 ISR implementiert:
  - ▶ ISR1 hat die höchste Priorität und dauert 200 µs,
  - ▶ ISR2 hat eine mittlere Priorität und dauert 400 µs und
  - ▶ ISR3 hat die niedrigste Priorität und dauert 300 µs.
- ▶ Jede Interruptquelle feuert maximal einmal pro 1 msec.
- ▶ Welche Interrupt-Latenzzeiten sind im schlimmsten Fall auf Grund der obigen Angaben für die ISRs mindestens zu erwarten?

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ: Preemptive ISR

- ▶ Erfordert zwei zusätzliche Aktivitäten:

1. Nach Quittierung des Interrupts:

- Löschen des I-Bits (Freigabe des IRQs).
- Retten von LR und SPSR (nicht trivial)

2. Vor Quittierung des Interrupts beim VIC:

- Sperren des IRQs
- Zurückspeichern von LR und SPSR

```
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {

    EXTINT = 1<<1;                                /* Interrupt bestaetigen:          */
                                                       /* Device deaktiviert IRQ-Leitung */

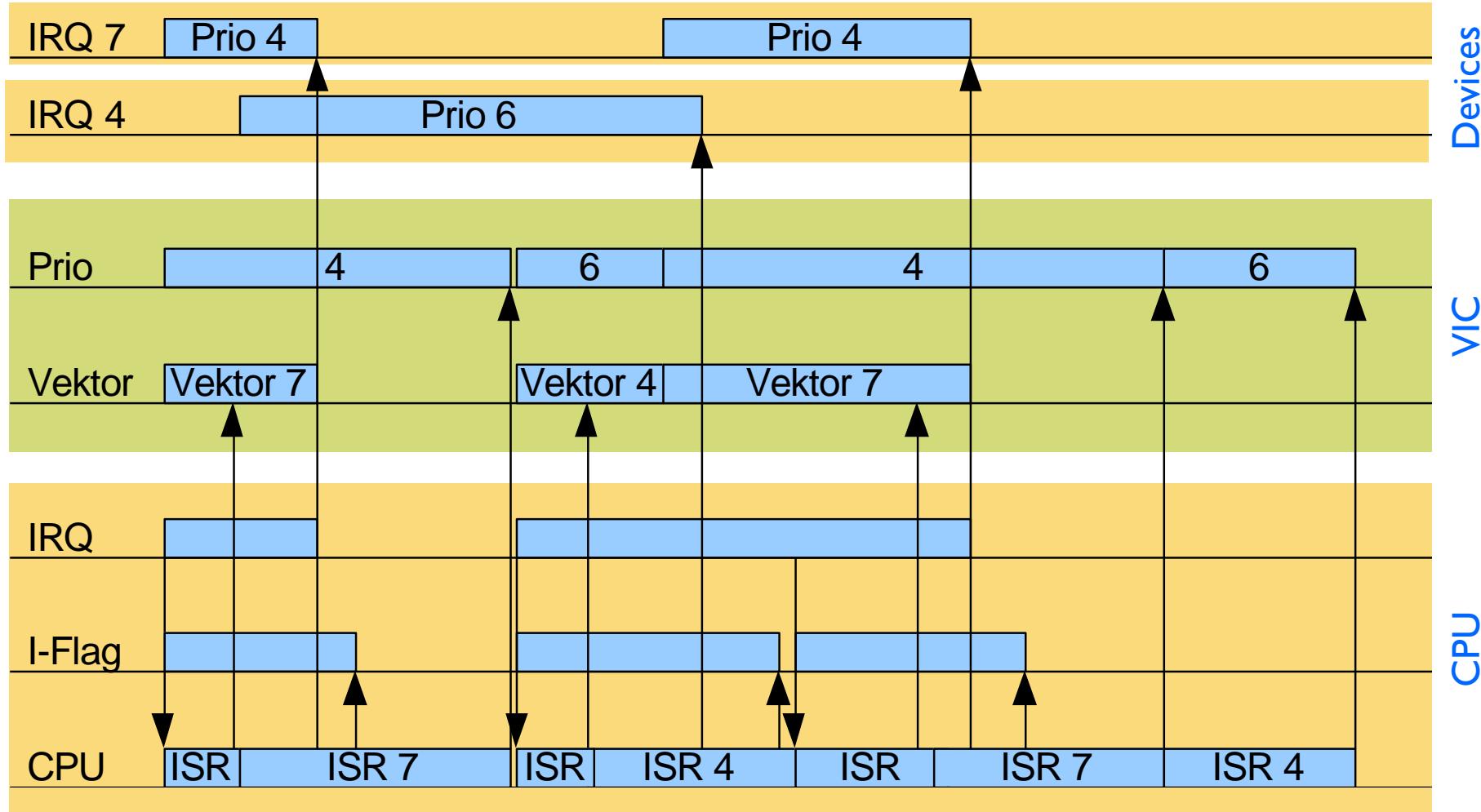
    ENABLE_NESTED_ISR;
    ....
    ....
    ....
    DISABLE_NESTED_ISR;

    VICVectAddr = 0;                                 /* VIC mitteilen: ISR ist fertig */
                                                       */
}
```



# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ: Preemptive ISR



# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ: Unterbrechbare ISR

- ▶ Retten von LR funktioniert nur mit Umschaltung in eine andere Betriebsart!
- ▶ Ohne Umschaltung geht aktuelles LR beim erneuten Interrupt verloren.
- ▶ Im Beispiel: Rumpf der ISR wird in Supervisor-Mode abgearbeitet.
- ▶ Achtung: Verwendung des Supervisorstacks!

```
#define ENABLE_NESTED_ISR
asm volatile(" mrs    r1, spsr\n"
            " stmfd  sp!,{r1,lr}\n"           /*SPSR_irq und LR_irq retten*/ \
            " msr    cpsr_c, #0x53\n"        /*Umschalten nach SVC-Mode */ \
            " stmfd  sp!,{lr}\n" : : "r1" ); /*LR_svc retten*/ \
```

```
#define DISABLE_NESTED_ISR
asm volatile(" ldmfd  sp!,{lr}\n"           /*LR_svc restaurieren*/ \
            " msr    cpsr_c, #0xd2\n"       /*Umschalten auf IRQ-Mode*/ \
            " ldmfd  sp!,{r1,lr}\n"         /*SPSR und LR wieder herstellen*/ \
            " msr    spsr_fc,r1\n" : : "r1" ); \
```

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Vectored IRQ: Unterbrechbare ISR, Latenzzeiten

- ▶ Es wurden 3 ISR implementiert:
  - ▶ ISR1 hat die höchste Priorität und dauert 200 µs,
  - ▶ ISR2 hat eine mittlere Priorität und dauert 400 µs und
  - ▶ ISR3 hat die niedrigste Priorität und dauert 300 µs.
- ▶ Jede Interruptquelle feuert maximal einmal pro 1 msec.
- ▶ Welche Interrupt-Latenzzeiten sind im schlimmsten Fall auf Grund der obigen Angaben für die ISRs mindestens zu erwarten?

# Ausnahme- und Interruptverarbeitung im LPC 2468

## Ein- und Ausschalten von Interrupts in der Hitex Laufzeitumgebung

- ▶ Funktionen in armVIC.h:

- ▶ **unsigned disableIRQ(void);**
- ▶ **unsigned enableIRQ(void);**
- ▶ **unsigned restoreIRQ(unsigned oldCPSR);**
  
- ▶ **unsigned disableFIQ(void);**
- ▶ **unsigned enableFIQ(void);**
- ▶ **unsigned restoreFIQ(unsigned oldCPSR);**

# Computer Engineering

## WS 2012

CPU  
Ein- und Ausgabe

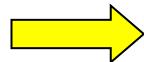
HTM – SHF - SWR



# CPU

## Ein- und Ausgabe

### Übersicht



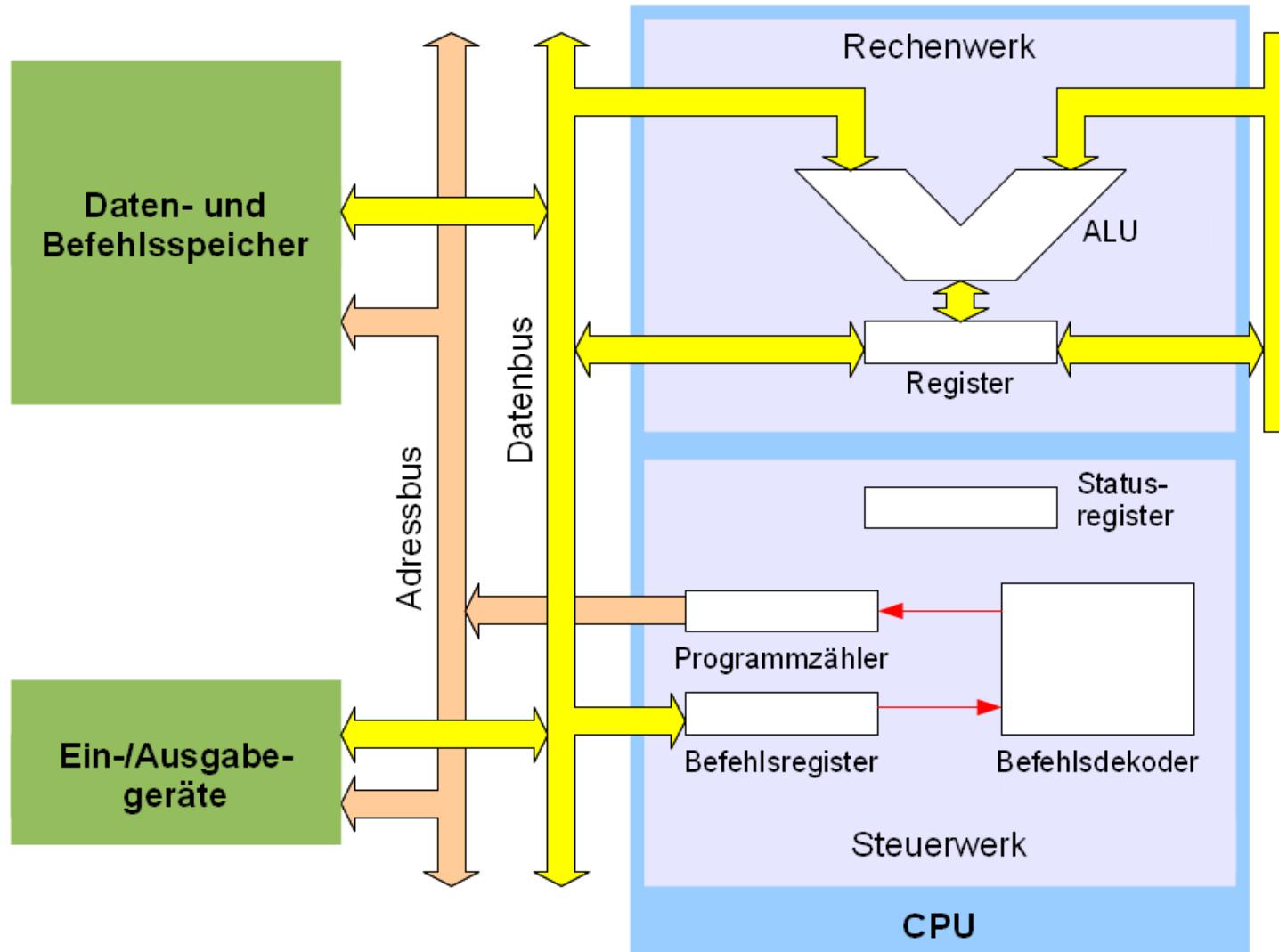
- ▶ Ein- und Ausgabe
  - ▶ Memory-Mapped I/O
  - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.  
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.



# Ein- und Ausgabe

CE WS12

## Prinzipieller Aufbau eines Computers



## Wie greift man auf Register eines externen Gerätes zu?

- ▶ Alternative 1: Memory Mapped
  - ▶ Die Register sind auf Hauptspeicheradressen abgebildet (mapped).
  - ▶ Ein lesender / schreibender Zugriff auf diese Hauptspeicheradressen greift nicht auf den Speicher zu, sondern auf die entsprechenden Register des Devices.
- ▶ Alternative 2: I/O Mapped (muss die CPU unterstützen, Intel tut dies)
  - ▶ Es gibt einen weiteren Adressraum, so genannte I/O Adressen. Diese Adressen stehen in keiner Relation zu den Hauptspeicheradressen.
  - ▶ Über spezielle Befehle (in, out Assembler Befehle) wird über I/O Adressen auf die Register eines Devices zugegriffen.

CE WS12

## Memory-Mapped

Hauptspeicher-  
adressen  
 $0x00000000$

RAM

$0xAFFFFFFF$

$0xFF000000$

$0xFF00000F$

$0xFF000100$

$0xFF0001FF$

$0xFFFFFFFF$

*Adressraum:  
von der CPU adressierbarer  
Bereich  
 $0x00000000-0xFFFFFFFF$*

RAM

*intelligente HW*

*nichts*

*intelligente HW*



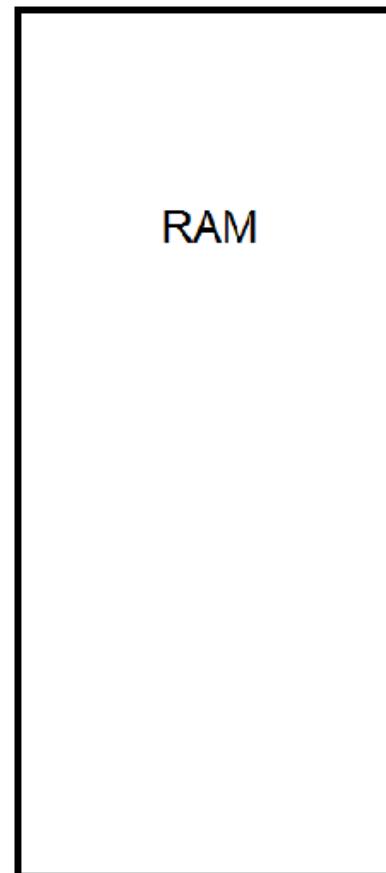
CE WS12

# Ein- und Ausgabe

## IO-Mapped

0x00000000

RAM:  
Adressbereich  
vollständig  
verfügbar



I/O Adressraum

Device 1

Device 2

intelligente HW

nichts

intelligente HW

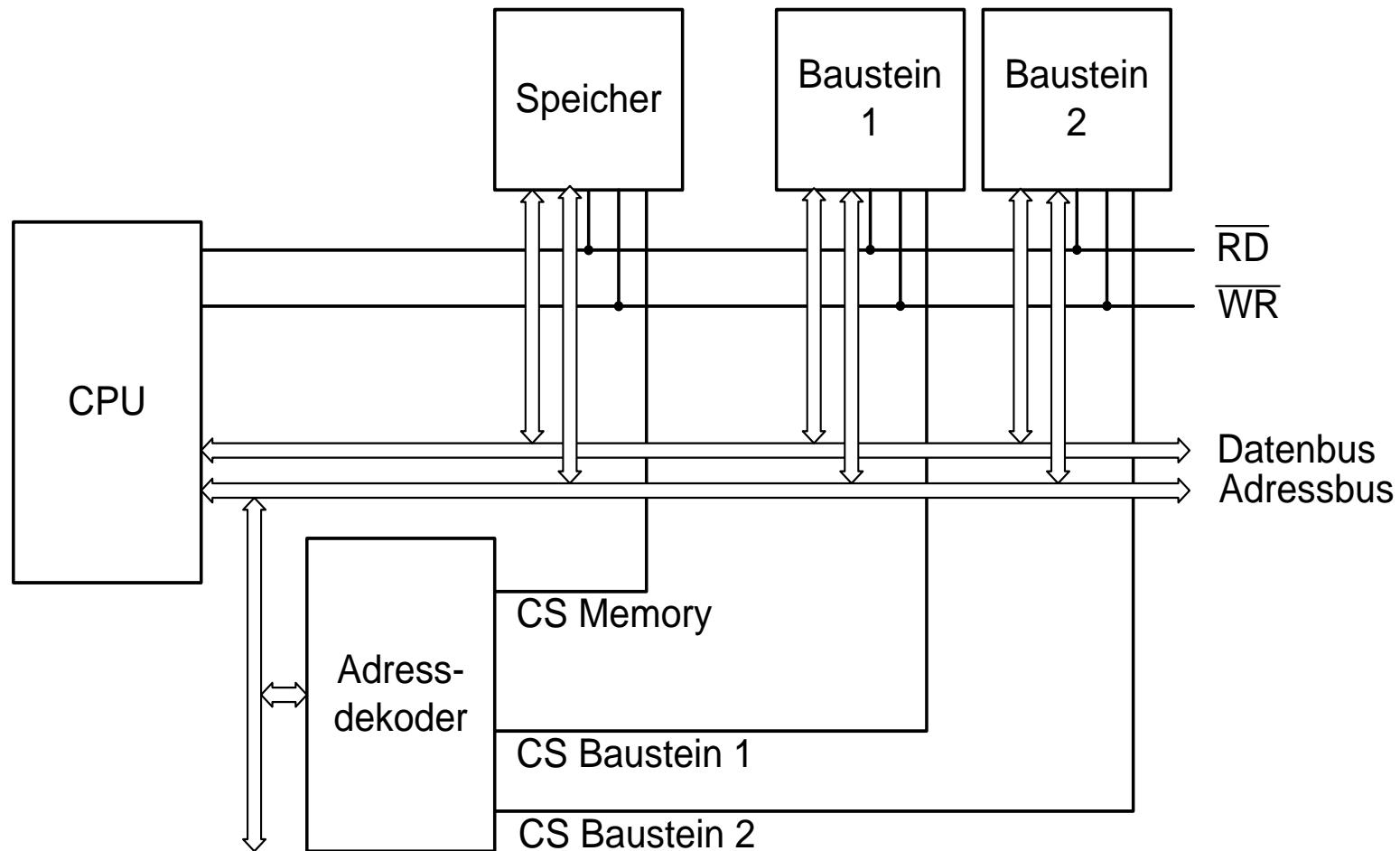
Zweiter Adressraum  
parallel zum Adressraum  
des Hauptspeichers

0xFFFFFFFF

# Ein- und Ausgabe

CE WS12

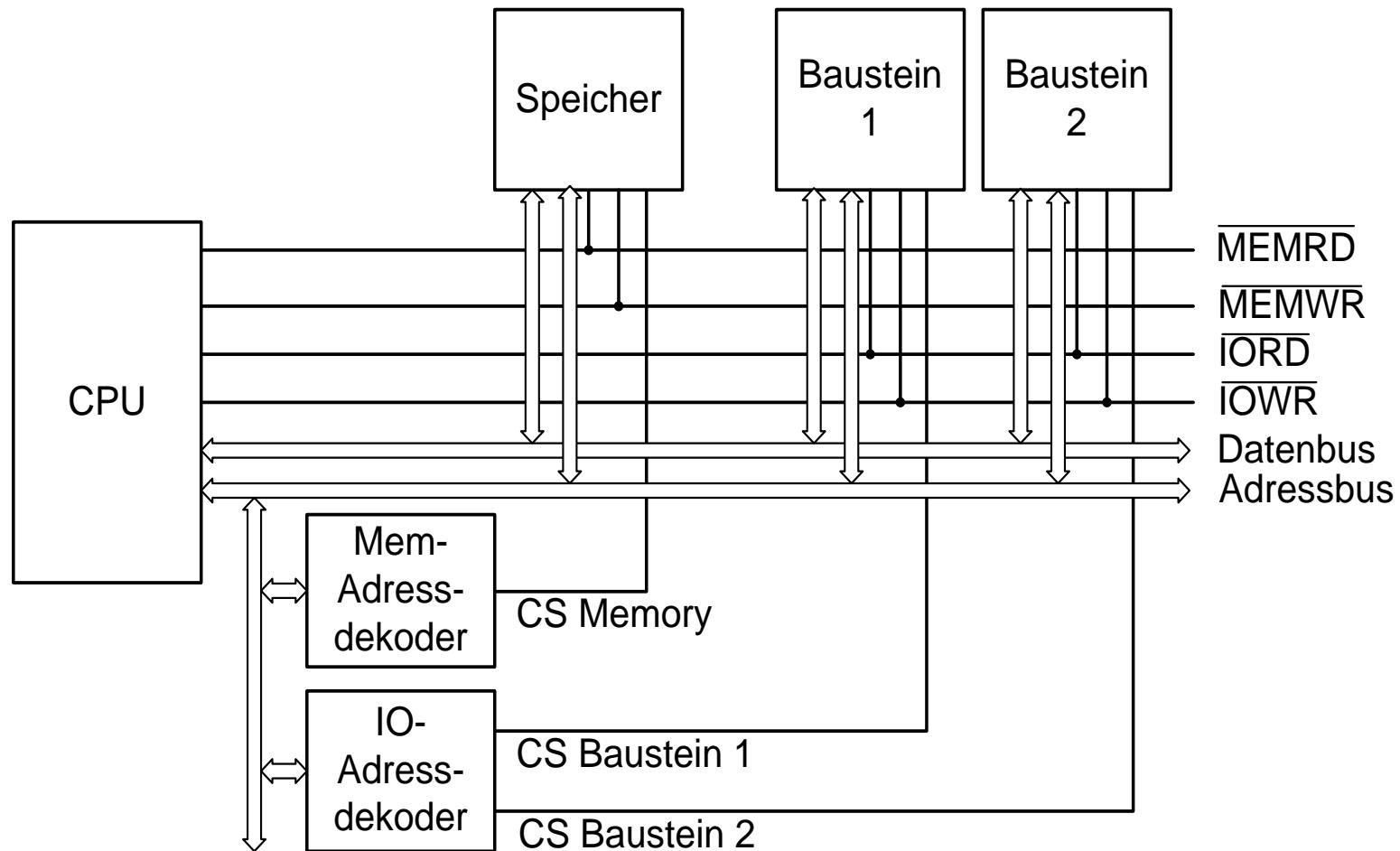
## Einfaches Bussystem, Memory-Mapped



# Ein- und Ausgabe

CE WS12

## Einfaches Bussystem, IO-Mapped



# Ein- und Ausgabe

## Merkmale Bussysteme: Maximale Übertragungsrate, Einheit?

- ▶ hängt ab von:
  - ▶ Anzahl der gleichzeitig übertragbaren Bytes
    - gegeben durch Datenbusbreite
  - ▶ Bustaktfrequenz
    - maximale Frequenz, mit der die Signalleitungen eines Busses betrieben werden können
  - ▶ Anzahl der für die Übertragung notwendigen Bustakte



# Ein- und Ausgabe

## Merkmale Bussysteme: Buszyklenarten

- ▶ Einzelzyklusbus (single cycle)
  - ▶ Führt nur eine Datenübertragung durch
- ▶ Blockzyklus (burst cycle)
  - ▶ mehrfache aufeinander folgende Zugriffe
  - ▶ Adresse wird nur einmal übertragen

# Ein- und Ausgabe

## Merkmale Bussysteme: Bustypen

- ▶ **Split-Bus**
  - ▶ **Getrennte Adress- und Datenleitungen**
  - ▶ **Gleichzeitige Übertragung von Adressen und Daten möglich.**
- ▶ **Gemultiplexer Bus**
  - ▶ **Adressen und Daten werden über die gleichen Leitungen übertragen.**
  - ▶ **Während eines Speicherzugriffs werden zuerst die Adressen, dann die Daten übertragen.**
  - ▶ **Nachteil: Langsamer.**
  - ▶ **Vorteil: Geringerer Verdrahtungsaufwand, geringere Pin-Zahl.**

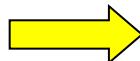


# CPU

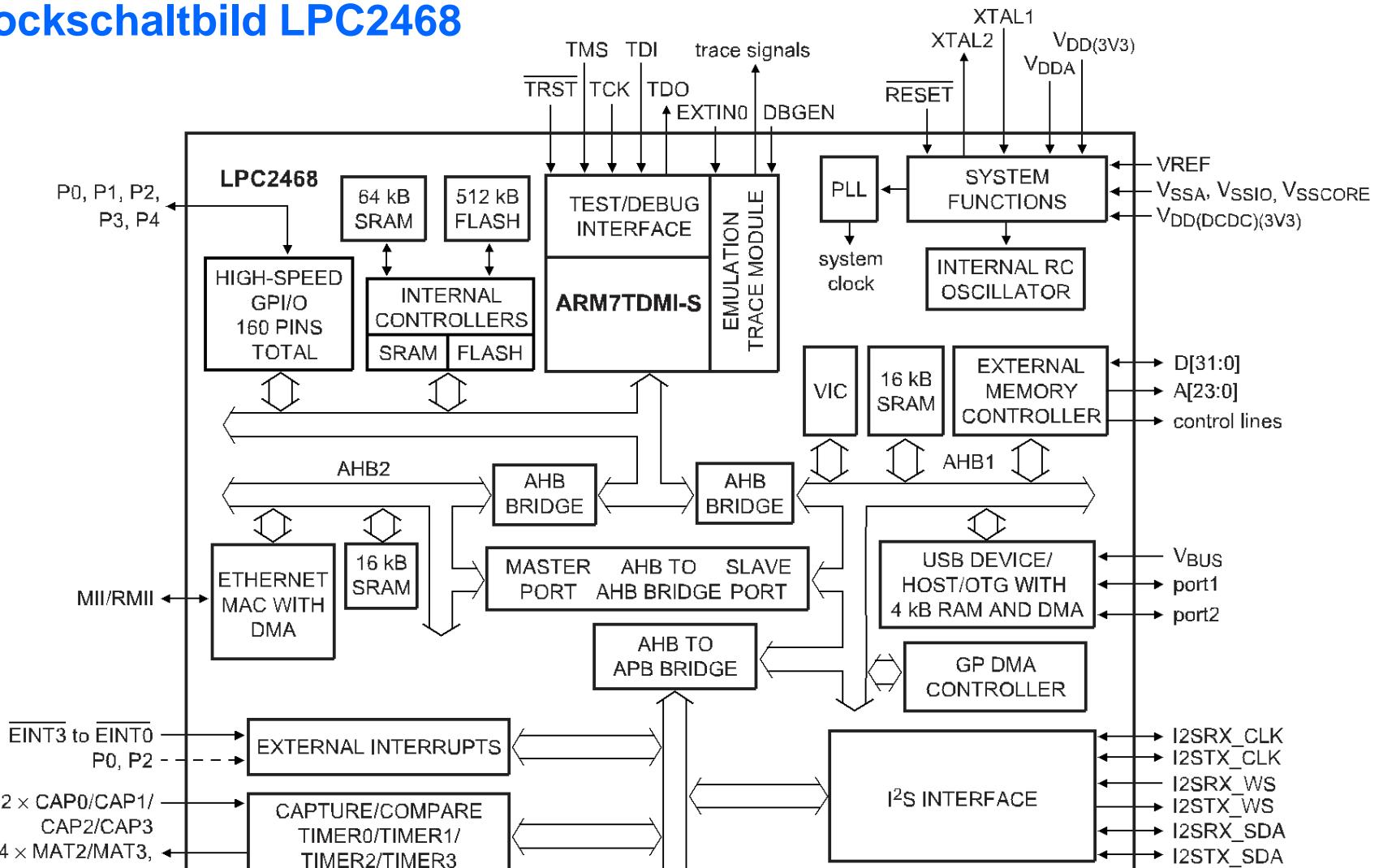
## Ein- und Ausgabe

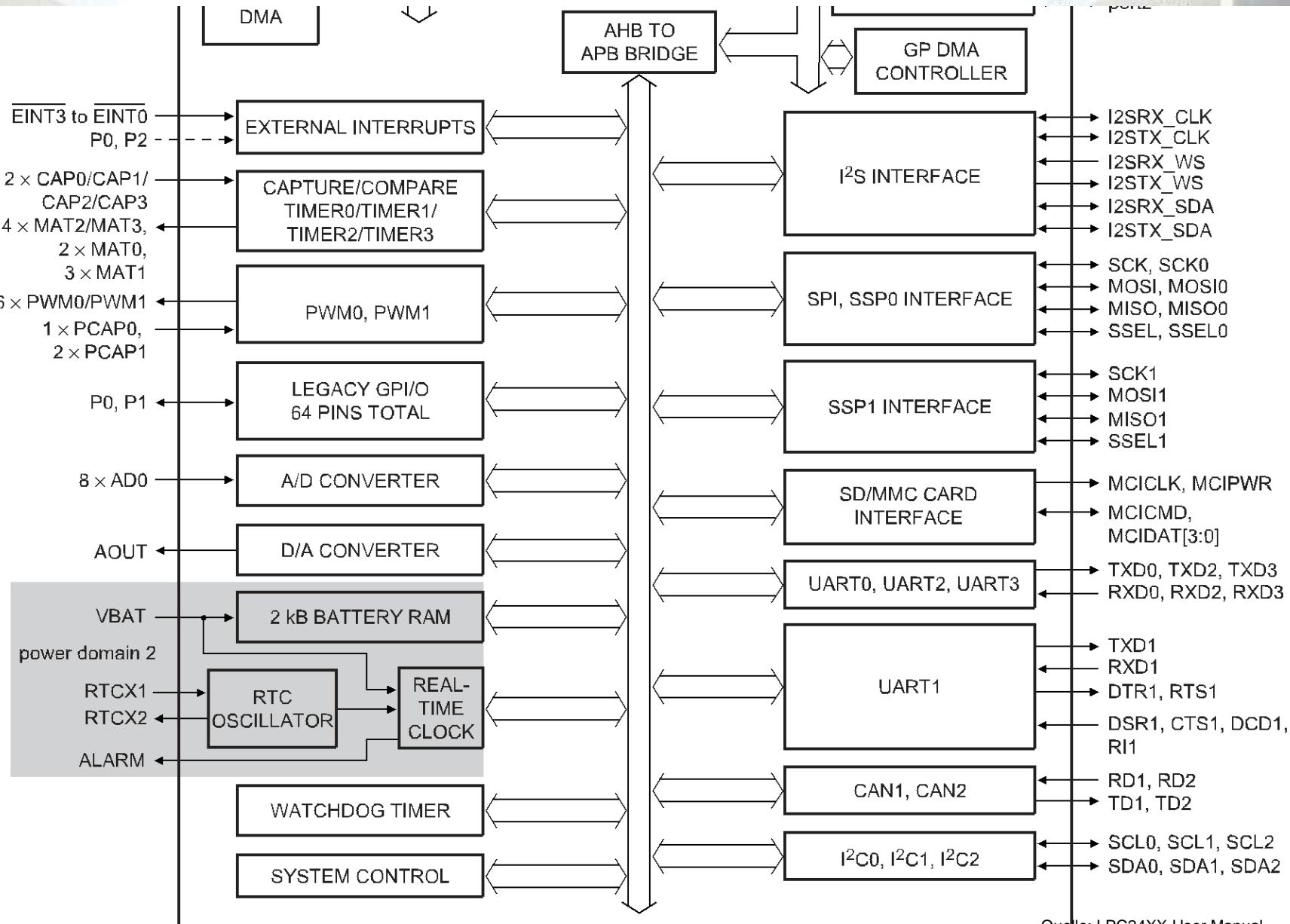
### Übersicht

- ▶ Ein- und Ausgabe
  - ▶ Memory-Mapped I/O
  - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.  
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.



## Blockschaltbild LPC2468







CE WS12

## Bushierarchie

- ▶ LPC2468 enthält 3 Hierarchieebenen:

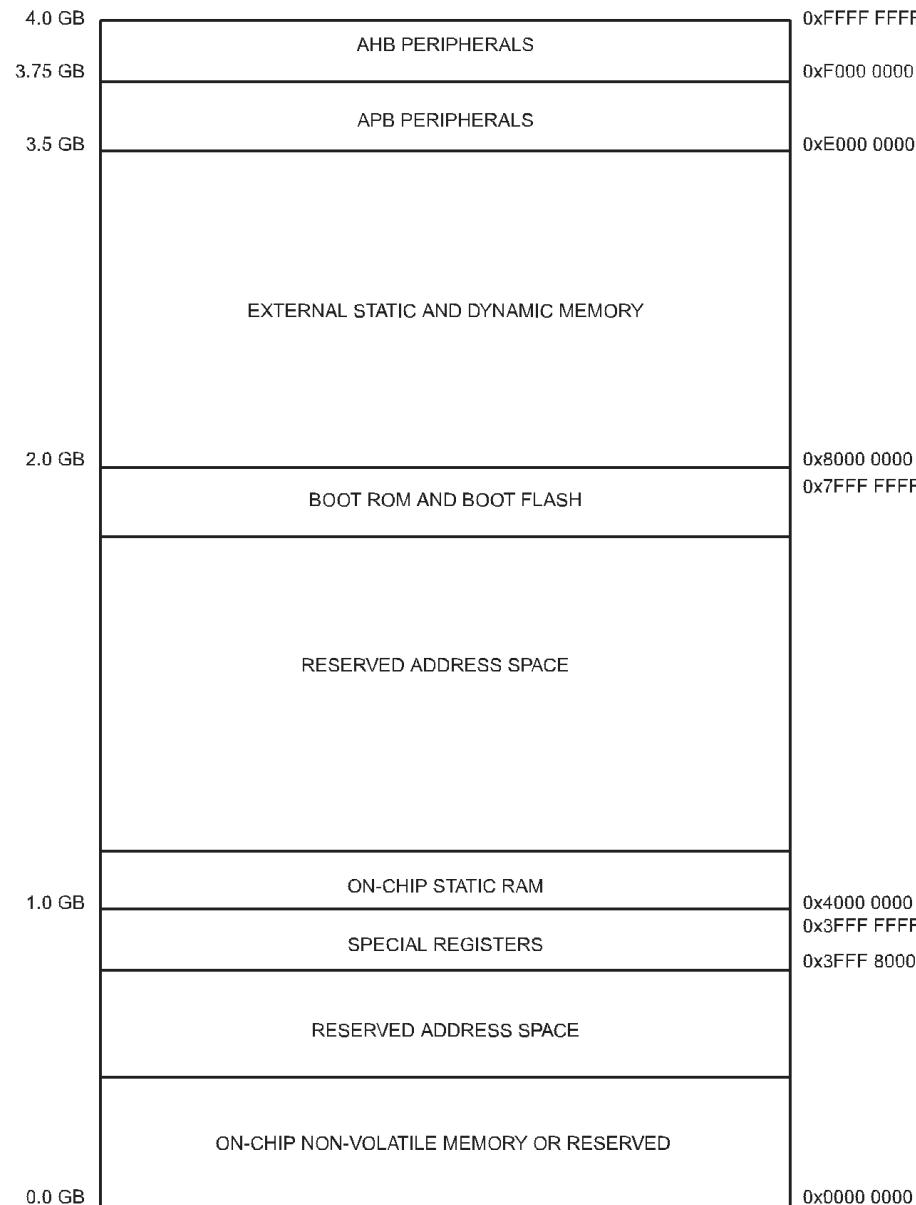
1. **ARM 7 local bus**
  - **High Speed**
  - **Anschluss von On-chip-Speicher (Flash, RAM)**
  - **Single Master: CPU**
2. **„Advanced High-performance BUS“ (AHB)**
  - **Multimasterfähig (z.B.: Bridge, USB, Ethernet)**
  - **burst transfers**
  - **single clock operation**
3. **„Advanced Peripheral Bus“ (APB)**
  - **Low Speed**
  - **Single Master: Bridge**
  - **Sehr einfacher Steuerbus**



# LPC2468

CE WS12

## Speicherbelegung





## Adressaufteilung

### APB

### (Ausschnitt)

- ▶ Jeder Baustein belegt 16 KBytes

APB Peripheral	Base Address	Peripheral Name
0	0xE000 0000	Watchdog Timer
1	0xE000 4000	Timer 0
2	0xE000 8000	Timer 1
3	0xE000 C000	UART0
4	0xE001 0000	UART1
5	0xE001 4000	PWM0
6	0xE001 8000	PWM1
7	0xE001 C000	I <sup>2</sup> C0
8	0xE002 0000	SPI
9	0xE002 4000	RTC
10	0xE002 8000	GPIO
11	0xE002 C000	Pin Connect Block
12	0xE003 0000	SSP1
13	0xE003 4000	ADC
14	0xE003 8000	CAN Acceptance Filter RAM
15	0xE003 C000	CAN Acceptance Filter Registers
16	0xE004 0000	CAN Common Registers
17	0xE004 4000	CAN Controller 1
18	0xE004 8000	CAN Controller 2
19 to 22	0xE004 C000 to 0xE005 8000	Not used
23	0xE005 C000	I <sup>2</sup> C1

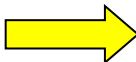


# CPU

## Ein- und Ausgabe

### Übersicht

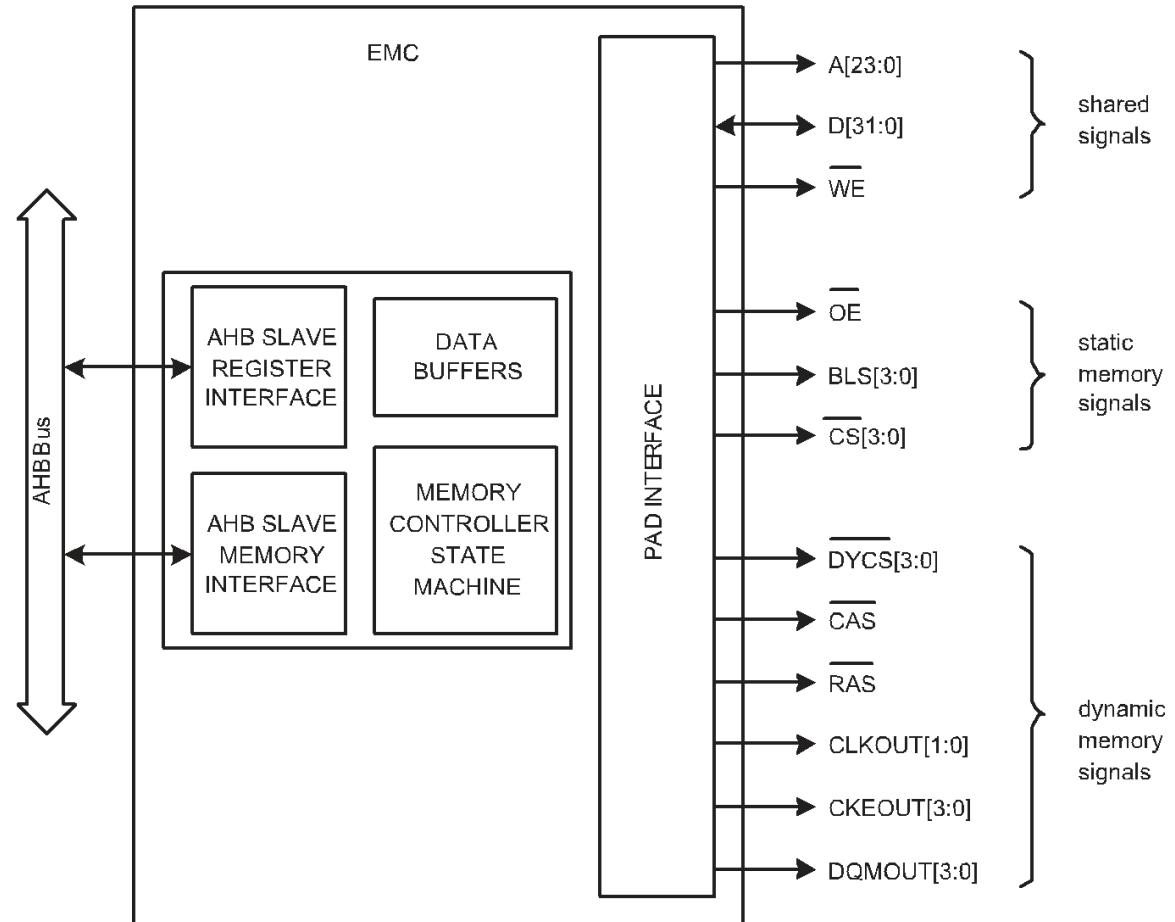
- ▶ Ein- und Ausgabe
  - ▶ Memory-Mapped I/O
  - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.  
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.





## Externe Speicherschnittstelle

- ▶ Zugriff über AHB1
- ▶ Schnittstelle für
  - ▶ **Statisches RAM**
  - ▶ **Dynamisches RAM**
- ▶ Unterschiedliche Busbreiten:
  - ▶ **8 bit**
  - ▶ **16 bit**
  - ▶ **32 bit**

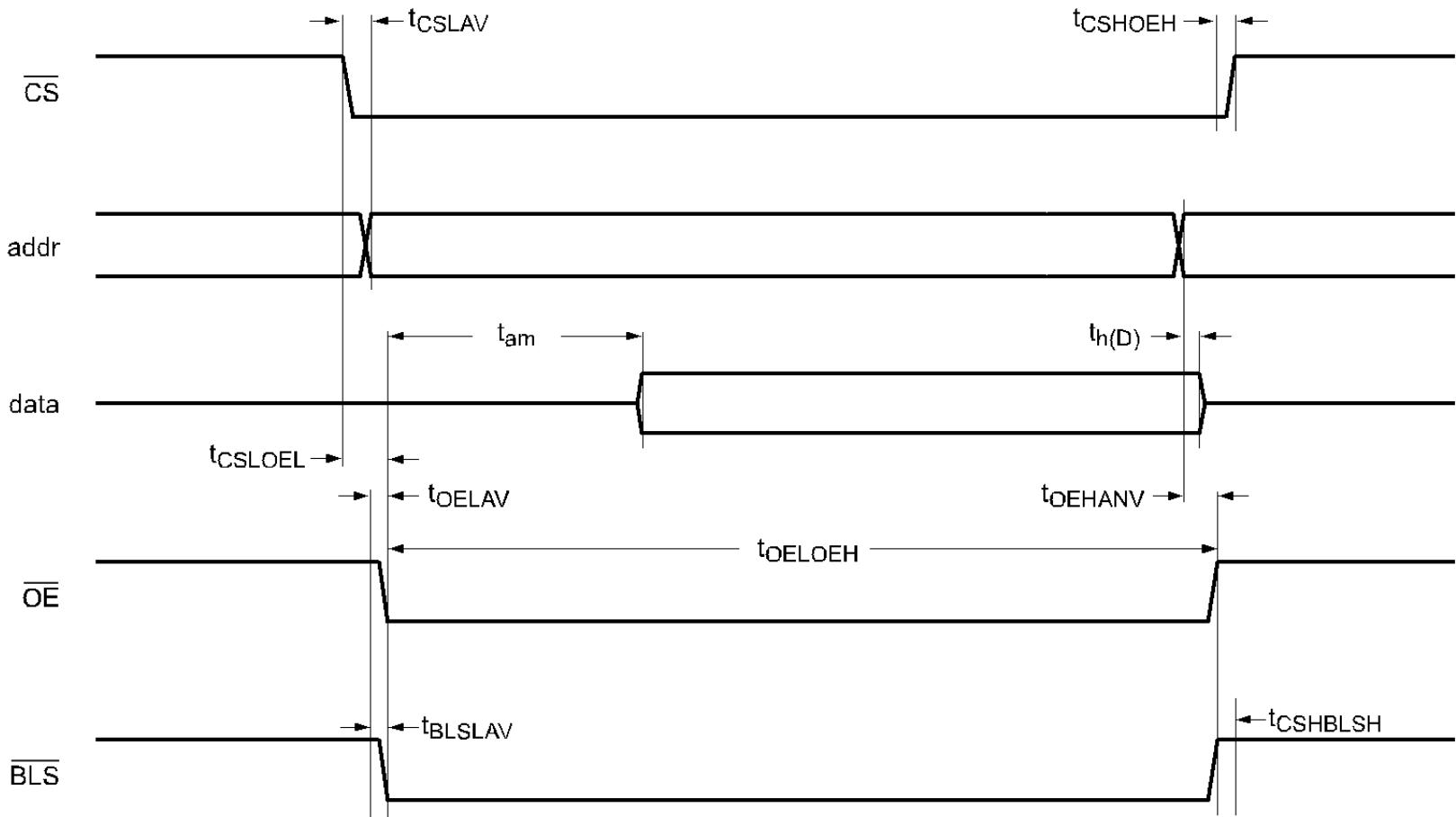




## Speicherbereiche für externe Speicherschnittstelle 8 bit

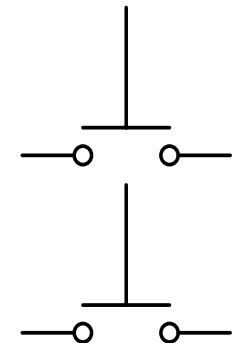
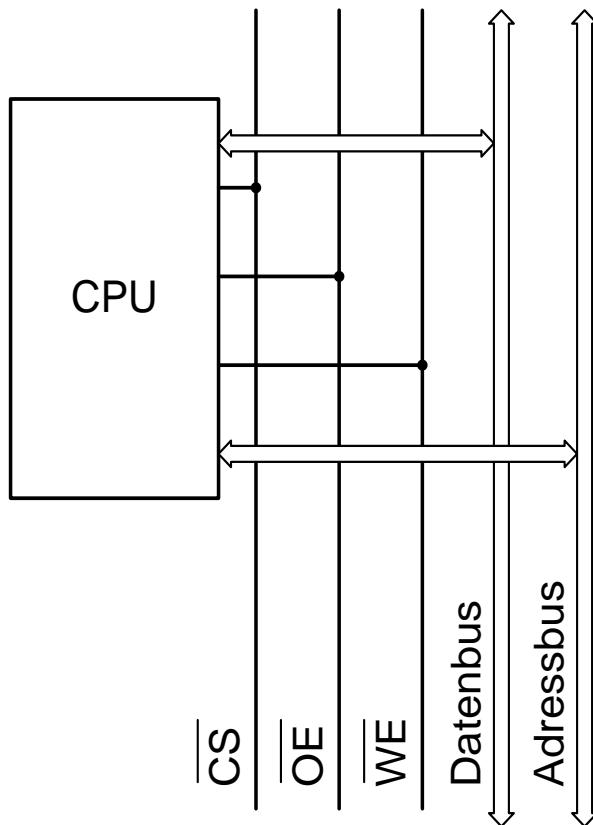
Chip select pin	Address range	Memory type	Size of range
CS0	0x8000 0000 - 0x80FF FFFF	Static	16 MB
CS1	0x8100 0000 - 0x81FF FFFF	Static	16 MB
CS2	0x8200 0000 - 0x82FF FFFF	Static	16 MB
CS3	0x8300 0000 - 0x83FF FFFF	Static	16 MB
DYCS0	0xA000 0000 - 0xAF00 FFFF	Dynamic	256 MB
DYCS1	0xB000 0000 - 0xBF00 FFFF	Dynamic	256 MB
DYCS2	0xC000 0000 - 0xCF00 FFFF	Dynamic	256 MB
DYCS3	0xD000 0000 - 0xDF00 FFFF	Dynamic	256 MB

## Externe Speicherschnittstelle, Lesezugriff auf statischen Speicher





## Lesezugriff, Anschließen von Tasten





CE WS12

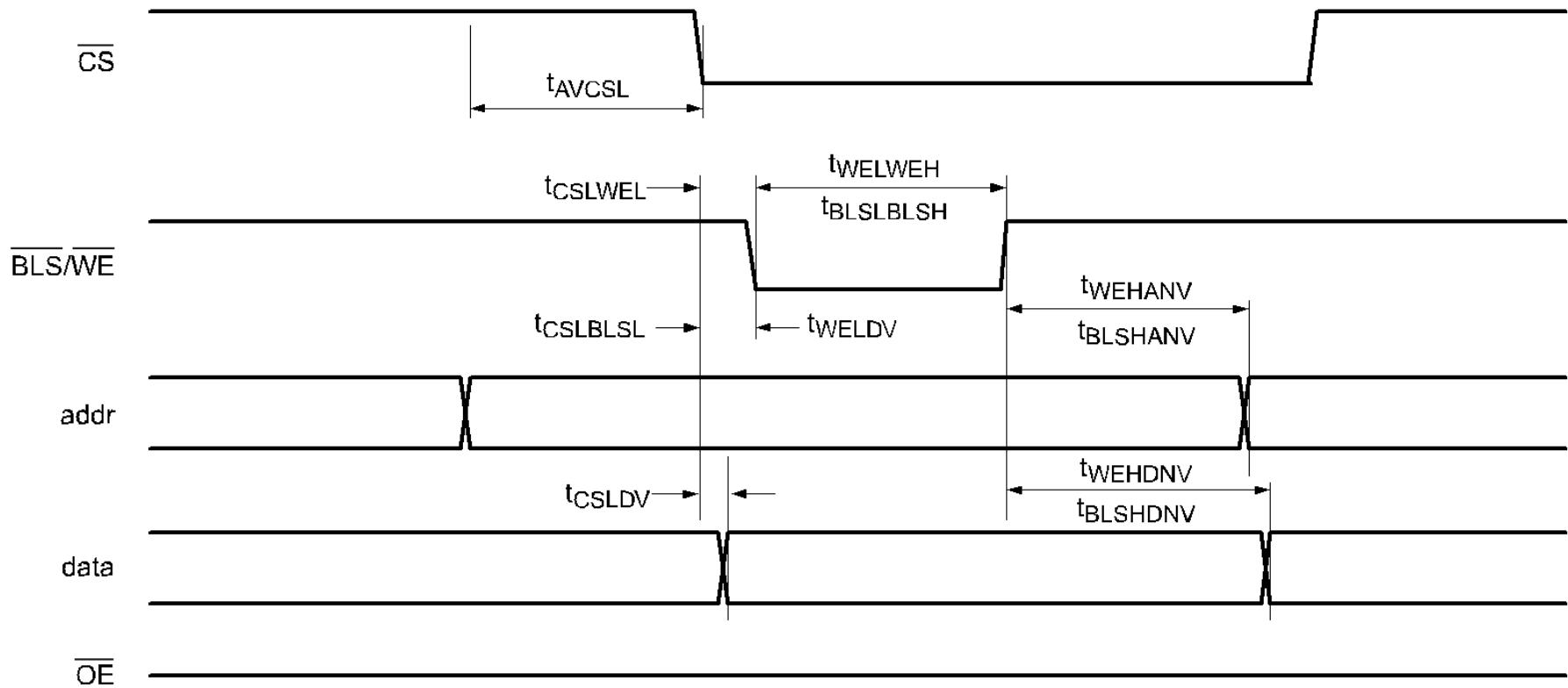
## Lesezugriff

- ▶ Ausgewählte Speicheradresse: CS0 + 0x9 = 0x8000 0009
- ▶ Assembler
  
- ▶ C



CE WS12

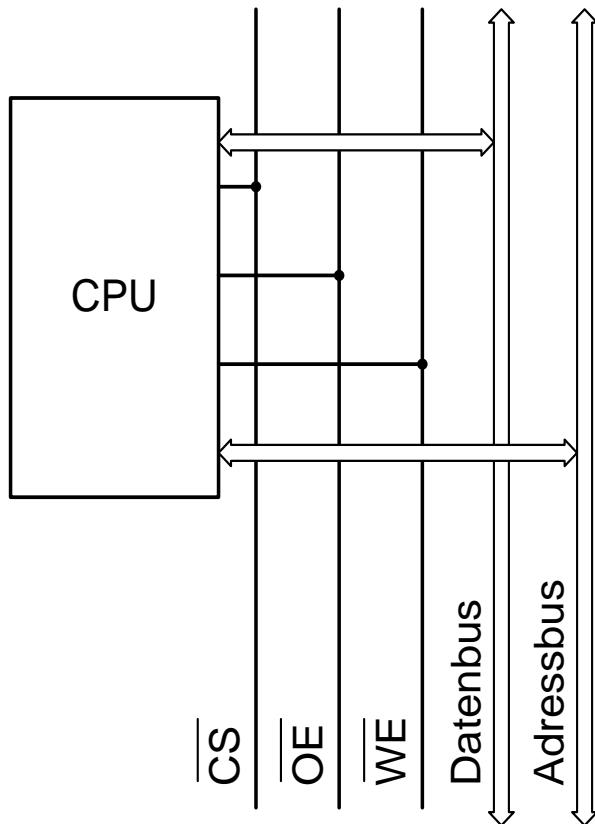
## Externe Speicherschnittstelle, Schreibzugriff auf statischen Speicher





CE WS12

## Schreibzugriff, Anschließen von Lampen





CE WS12

## Schreibzugriff

- ▶ Ausgewählte Speicheradresse: CS0 + 0x9 = 0x8000 0009
- ▶ Assembler
  
- ▶ C

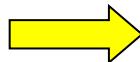


# CPU

## Ein- und Ausgabe

### Übersicht

- ▶ Ein- und Ausgabe
  - ▶ Memory-Mapped I/O
  - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.  
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.





CE WS12

## Steuerung des Stromverbrauchs

- ▶ Betriebszustände zum Reduzieren des Stromverbrauchs:
  - ▶ **Idle mode,**
  - ▶ **Sleep mode,**
  - ▶ **Power-down mode,**
  - ▶ **Deep power-down mode.**
- ▶ Stromverbrauch wird wesentlich durch die **Taktfrequenz** bestimmt
  - ▶ CPU-Taktfrequenz kann per Software verändert werden
- ▶ Steuerung des Leistungsverbrauchs von **Peripherie-Bausteinen**:
  - ▶ Nicht benötigte Bausteine können ausgeschaltet werden (siehe PCONP-Register).
  - ▶ Taktfrequenzen der einzelnen Komponenten lassen sich individuell einstellen (siehe PCLKSEL0 and PCLKSEL1).



CE WS12

## Ein- und Ausschalten von Peripherie-Komponenten

- ▶ Einstellung über PCONP:
  - ▶ Für jede Komponente steht 1 Bit zur Verfügung:
    - 0: Komponente ist deaktiviert.
    - 1: Komponente wird mit Takt versorgt.

Ausschnitt von  
PCONP:

Bit	Symbol	Description	Reset value
0	-	Unused, always 0	0
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	PCPWM0	PWM0 power/clock control bit.	1
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I <sup>2</sup> C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1



CE WS12

## Takteinstellung Peripherie-Komponenten

- ▶ Einstellung über PCLKSEL0 und PCLKSEL1:
  - ▶ In PCLKSELx stehen für jede Komponente 2 Bits zur Verfügung.
  - ▶ Damit kann jede Komponente auf 4 verschiedene Taktfrequenzen eingestellt werden (siehe Tabelle).
  - ▶ Standard ist CCLK/4, CCLK ist die CPU-Frequenz, z.B.: 48 MHz

### Ausschnitt von PCLKSEL0:

Bit	Symbol	Description
1:0	PCLK_WDT	Peripheral clock selection for WDT.
3:2	PCLK_TIMER0	Peripheral clock selection for TIMER0.
5:4	PCLK_TIMER1	Peripheral clock selection for TIMER1.
7:6	PCLK_UART0	Peripheral clock selection for UART0.
9:8	PCLK_UART1	Peripheral clock selection for UART1.

### Bedeutung der Bits:

00	PCLK_xyz = CCLK/4
01	PCLK_xyz = CCLK
10	PCLK_xyz = CCLK/2
11	PCLK_xyz = CCLK/8 PCLK_xyz = CCLK/6 CAN1, CAN2, CAN filtering

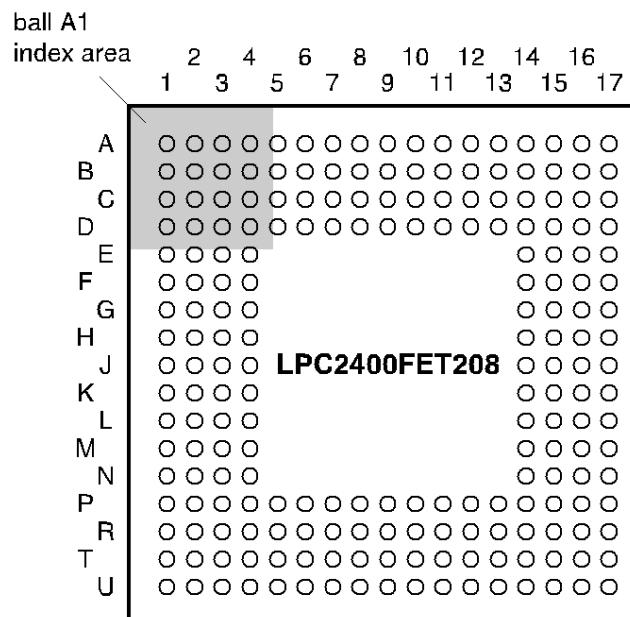


CE WS12

## Anschlüsse (Pins)

- ▶ Baustein hat 208 Pins
- ▶ Möglich wird dies durch ein Ball Grid Array (BGA)
  - ▶ Anschlüsse über kleine Lotkügelchen (Balls).
  - ▶ Dadurch Anordnung in mehreren Reihen und Spalten möglich.

**LPC2400 pinning  
TFBGA208 package**





CE WS12

## Anschlüsse (Pins)

- Die meisten Pins können zwischen 4 verschiedene Funktionen per Software umgeschaltet werden.
- Hierzu dienen die 10 Register PINSEL0 bis PINSEL9.
- Standardmäßig ist die „General Purpose IO“ (GPIO)-Funktion aktiviert.

PINSEL4	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P2[0]	GPIO Port 2.0	PWM1[1]	TXD1	TRACECLK	00
3:2	P2[1]	GPIO Port 2.1	PWM1[2]	RXD1	PIPESTAT0	00
5:4	P2[2]	GPIO Port 2.2	PWM1[3]	CTS1	PIPESTAT1	00
7:6	P2[3]	GPIO Port 2.3	PWM1[4]	DCD1	PIPESTAT2	00
9:8	P2[4]	GPIO Port 2.4	PWM1[5]	DSR1	TRACESYNC	00
11:10	P2[5]	GPIO Port 2.5	PWM1[6]	DTR1	TRACEPKT0	00
13:12	P2[6]	GPIO Port 2.6	PCAP1[0]	RI1	TRACEPKT1	00
15:14	P2[7]	GPIO Port 2.7	RD2	RTS1	TRACEPKT2	00



CE WS12

## Anschlüsse (Pins)

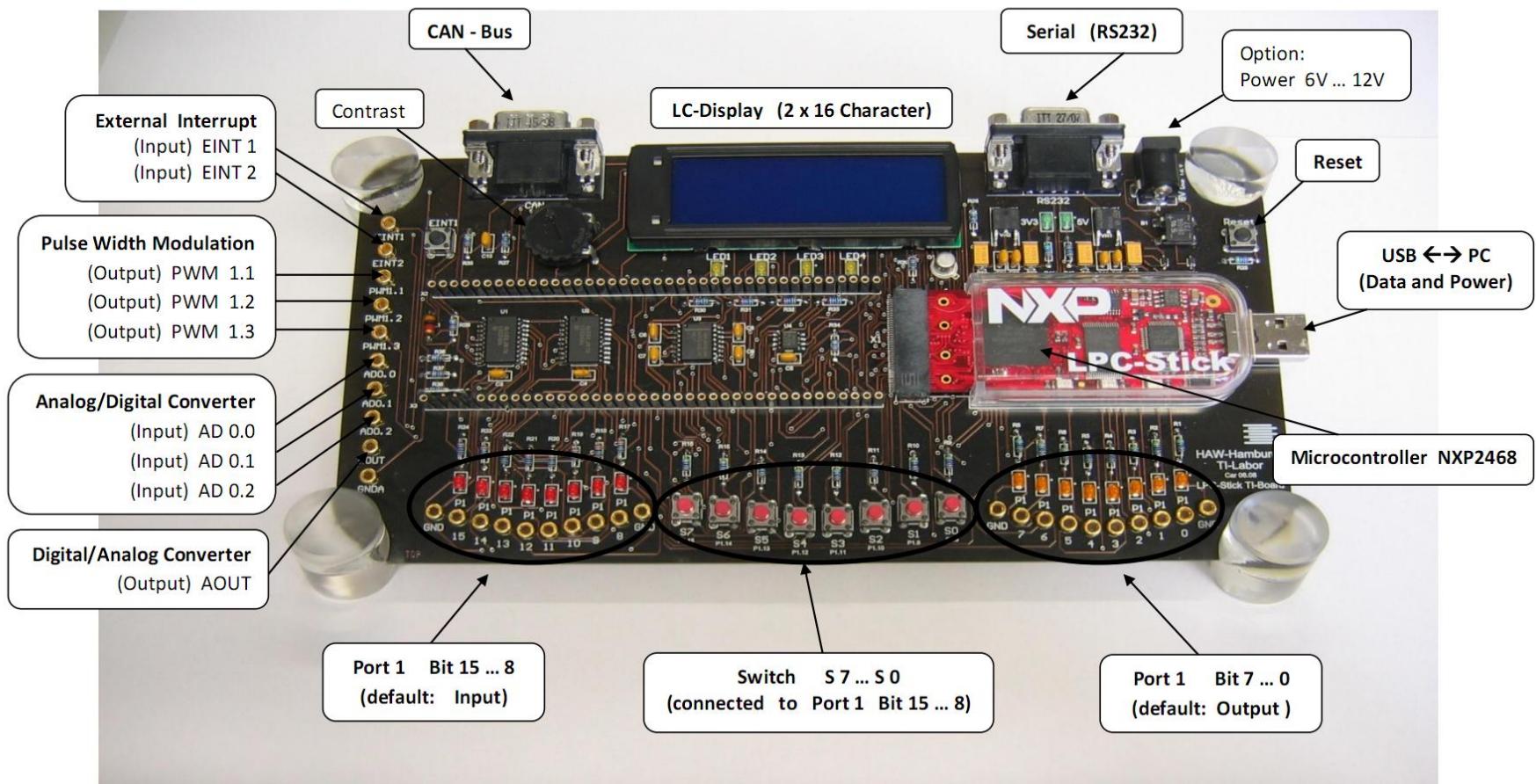
- ▶ Viele Pins können mit einem chipinternen Pull-up oder Pull-down Widerstand beschaltet werden.
- ▶ Hierzu dienen die 10 Register PINMODE0 bis PINMODE9.
- ▶ Standardmäßig ist der Pull-up Widerstand aktiviert.

PINMODE0 to PINMODE9 Values	Function
00	Pin has an on-chip pull-up resistor enabled.
01	Reserved. This value should not be used.
10	Pin has neither pull-up nor pull-down resistor enabled.
11	Pin has an on-chip pull-down resistor enabled.

PINMODE4	Symbol	Description	Reset value
1:0	P2.00MODE	PORT2 pin 0 on-chip pull-up/down resistor control.	00
...			
31:30	P2.15MODE	PORT2 pin 15 on-chip pull-up/down resistor control.	00



## LPC-Stick TI-Board





# CPU

## Ein- und Ausgabe

### Übersicht

- ▶ Ein- und Ausgabe
  - ▶ Memory-Mapped I/O
  - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.  
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.





CE WS12

## General Purpose Input/Output

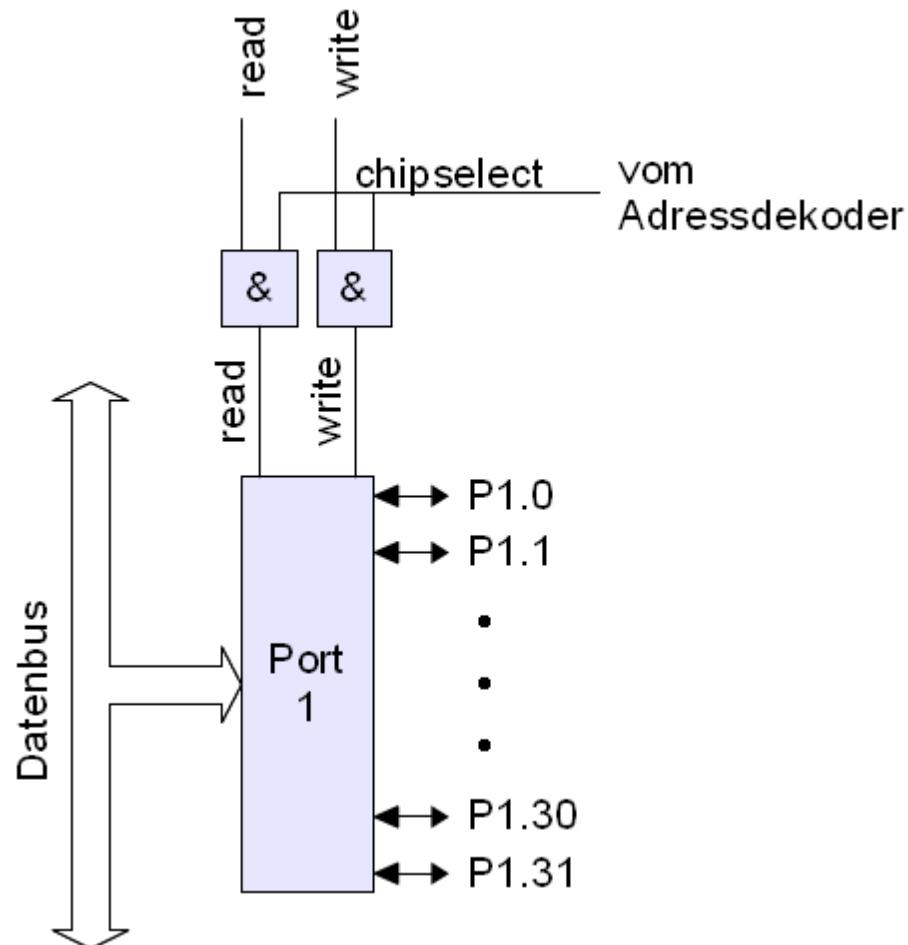
- ▶ Chipselect wird aktiviert beim Zugriff auf die Variable `GPIO1_IOPIN`.
- ▶ `GPIO1_IOPIN` ist in der Headerdatei `lpc24xx.h` definiert.
- ▶ **Eingabe:** Alle 32 Bit lesen:

```
int x;
x = GPIO1_IOPIN;
printf( "%08x",  x );
```

- ▶ **Ausgabe:** Alle 32 Bit verändern:

```
GPIO1_IOPIN = 0x88FF0011;
```

- ▶ **Fazit:**
  - ▶ Mit `GPIO1_IOPIN` können nur alle 32 Ein-/Ausgabeleitungen gleichzeitig bearbeitet werden.
  - ▶ Wie können einzelne Bits behandelt werden?





CE WS12

## LPC2468: General Purpose Input/Output

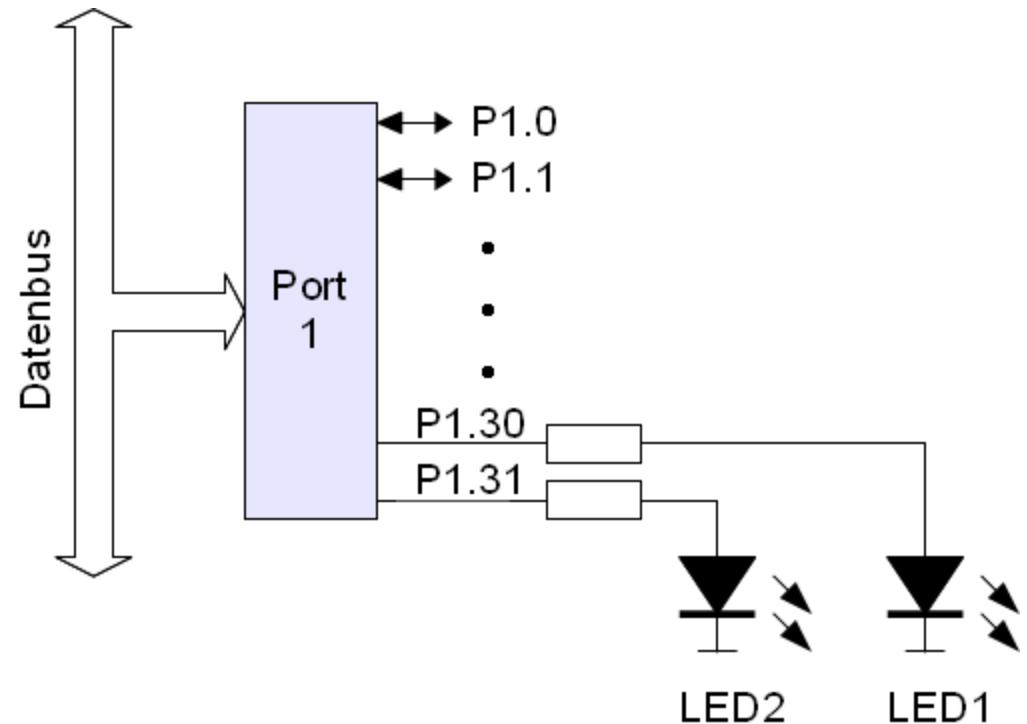
- ▶ **GPIOx\_IODIR**  
steuert individuell die Ein- und Ausgaberichtung jedes einzelnen Pins.
- ▶ **GPIOx\_IOMASK**  
steuert den Zugriff über die Ports  
**GPIOx\_IOPIN**, **GPIOx\_IOSET** und **GPIOx\_IOCLR**.  
Ermöglicht Auswahl einzelner Bits.  
Bit n == 0: Bit n kann gelesen oder beschrieben werden  
Bit n == 1: Beim Lesen ist dieses Bit immer 0  
                  Beim Schreiben wird dieses Bit nicht verändert
- ▶ **GPIOx\_IOPIN**  
Lesen des momentanen Zustandes der mit **GPIOx\_IOMASK** ausgewählten Bits.  
Schreiben der mit **GPIOx\_IOMASK** ausgewählten Bits.
- ▶ **GPIOx\_IOSET**  
Setzen der mit **GPIOx\_IOMASK** ausgewählten Bits.  
Bit n == "1": Ausgang n wird High (logisch 1).  
Bit n == "0": Ausgang n wird nicht verändert.
- ▶ **GPIOx\_IOCLR**  
Löschen der mit **GPIOx\_IOMASK** ausgewählten Bits.  
Bit n == "1": Ausgang n wird Low (logisch 0).  
Bit n == "0": Ausgang n wird nicht verändert.



CE WS12

## Ansteuerung einzelner Bits

- ▶ LED1 einschalten:



- ▶ LED1 ausschalten:

## ÜBUNG: Ein- und Ausgabe, Bitmanipulation

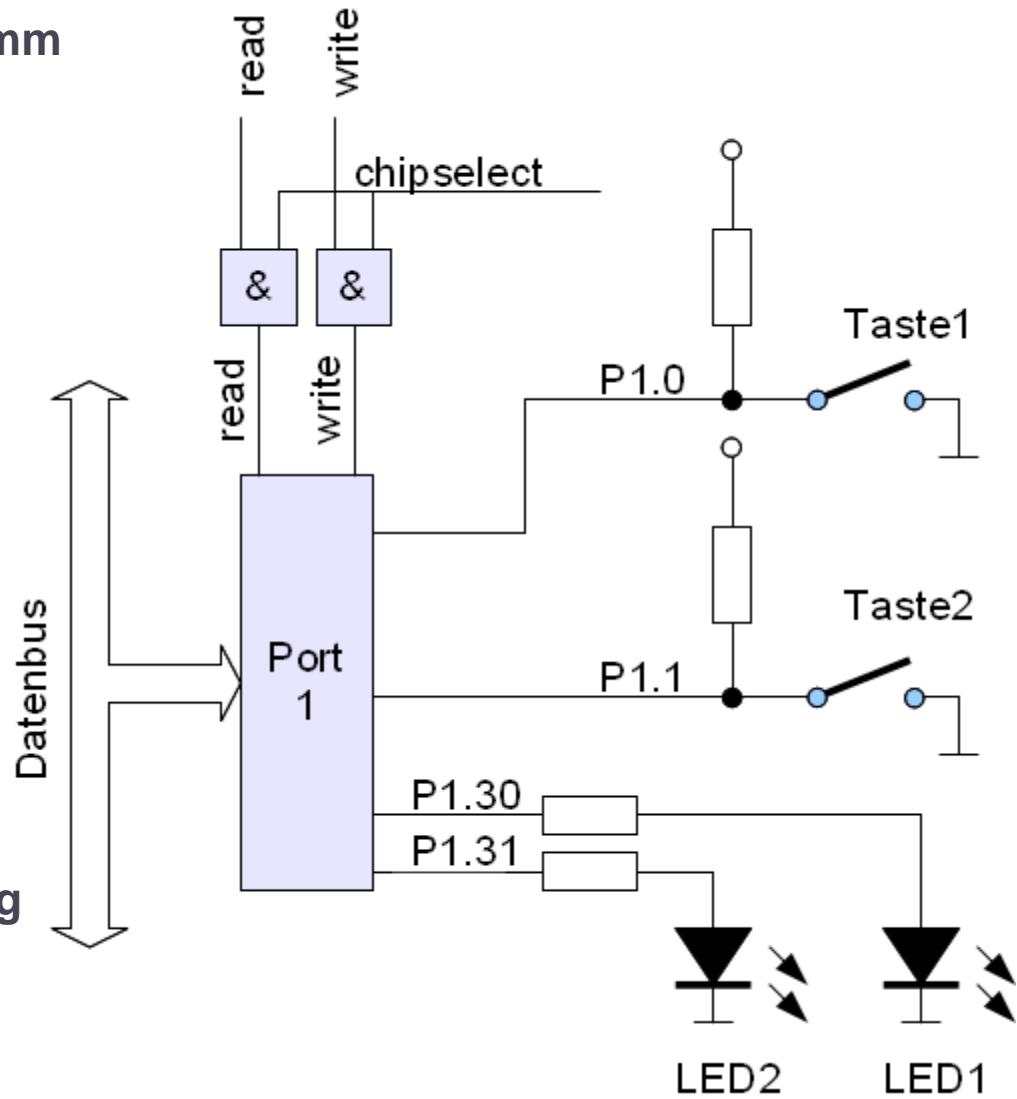
- Erstellen Sie ein Unterprogramm zum Ansteuern der LEDs:

```
void setLed( int leds );
```

mit folgendem Verhalten:

leds	LED1	LED2
0	aus	aus
1	ein	aus
2	aus	ein
3	ein	ein

- a) ohne Ausnutzung der LPC2xxx Hardware
- b) mit Hardwareunterstützung



# Computer Engineering

## WS 2012

### Timer/Counter

HTM – SHF - SWR



CE WS12

## Übersicht



- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468



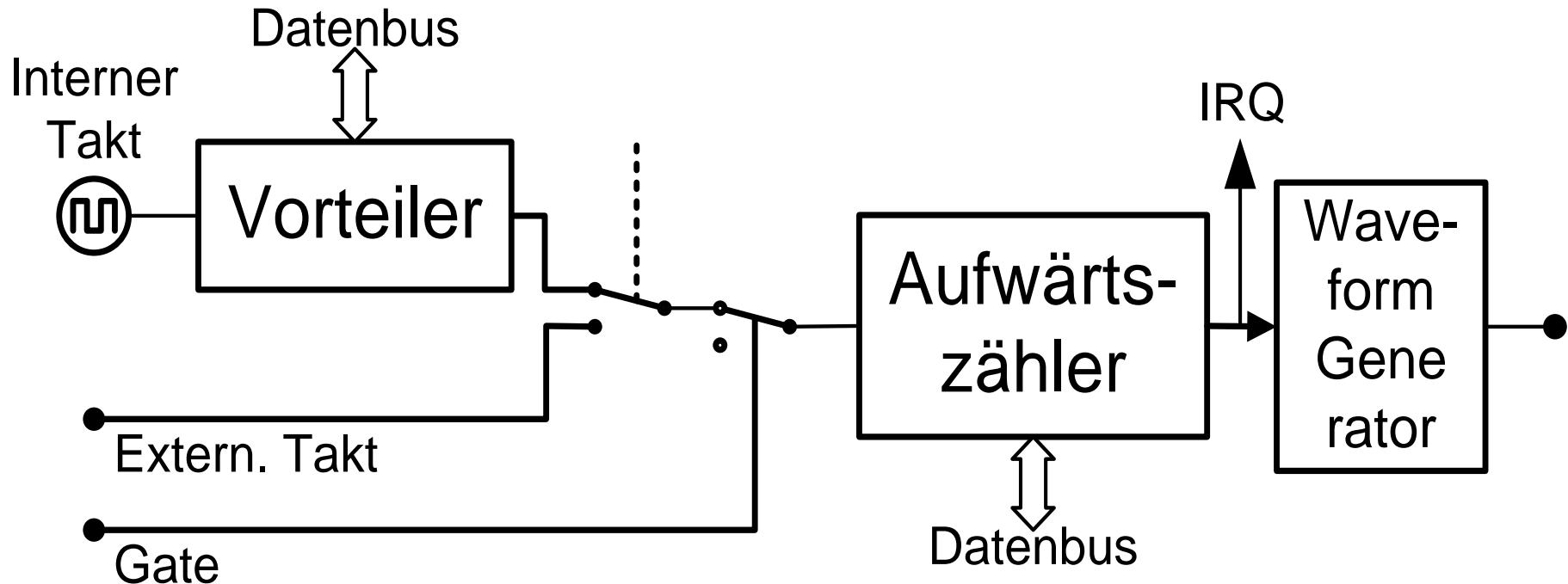
## Aufgaben

- ▶ **Erzeugen von Zeitfunktionen**
  - ▶ **Programmteile, die zu genau vorgegebenen Zeitpunkten ausgeführt werden sollen.**
  - ▶ **Externe Signale zur Steuerung von Peripheriegeräten.**
- ▶ **Messung von Zeitfunktionen**
  - ▶ **Dauer von externen Signalen.**
  - ▶ **Frequenz von externen Signalen.**
- ▶ **Zählen von Ereignissen.**

# Timer/Counter

CE WS12

## Prinzipieller Aufbau





## Anwendungen:

### Zeitgeber

- ▶ Periodisch oder Single Shot
- ▶ Interner Takt
- ▶ Beispielanwendungen
  - ▶ Systemtakt für Betriebssysteme
  - ▶ Schrittmotorsteuerung

### Zeitmessung

- ▶ Interner Takt
- ▶ Start und/oder Stop durch externes Ereignis
- ▶ Beispielanwendungen
  - ▶ Bestimmung der Baudrate

### Frequenzmessung

- ▶ Externer Takt
- ▶ Start und Stop des Zählers per Software

### Zähler

- ▶ Externe Ereignisse werden als Takt verwendet
- ▶ Beispielanwendungen
  - ▶ Auswertung von Drehimpulsgebern



## Überwachung von und Reaktion auf Timer-Ereignisse

- ▶ **Polling**
  - ▶ **Kontinuierliche Abfrage**
    - des Zählerstandes oder
    - des Statusregisters:

```
/* warte, bis Timer-Ereignis auftritt */

while(( TIMER0_IR & MR0) == 0 ) {  
}
```

- ▶ **Interrupt**
- ▶ **Automatische Änderung von Signalen an Anschlusspins**
  - ▶ **Mögliche Aktionen an Ausgängen bei Timer-Ereignissen:**
    - Setzen
    - Löschen
    - Toggeln



# Timer/Counter

CE WS12

## Zeitliche Auflösung

- ▶ bestimmt durch verwendete
  - ▶ Taktfrequenz

$$\Delta T = \frac{N_{Vorteiler}}{f_{Takt}}$$

## Intervall

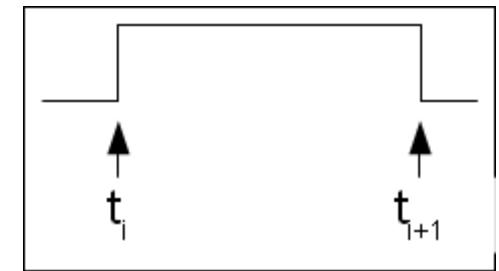
- ▶ bestimmt durch verwendete
  - ▶ Taktfrequenz
  - ▶ Anzahl Bits des Zählers

$$T_{Max} = 2^N \cdot \frac{N_{Vorteiler}}{f_{Takt}}$$

# Timer/Counter

## Übung: Dimensionierung eines Timers

- ▶ Mit Hilfe eines 16-Bit Timers soll eine Impulsdauer zwischen 0.5 und 2.0 Sekunden gemessen werden. Die Systemfrequenz beträgt 8 MHz. Als Verteiler stehen die Werte 1, 8, 64, 256 und 1024 zur Verfügung.



1. Wie muss der Verteiler eingestellt werden?
2. Wie genau kann die Zeit erfasst werden?  
Verwenden Sie bitte handhabbare Einheiten.



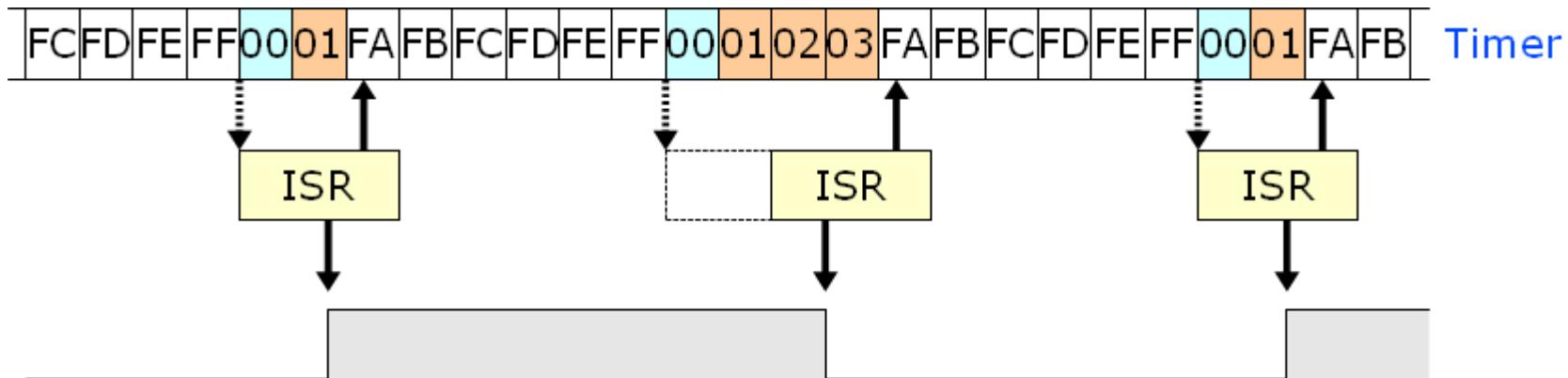
CE WS12

## Übersicht

- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468



## Erzeugung eines periodischen Ausgangssignals

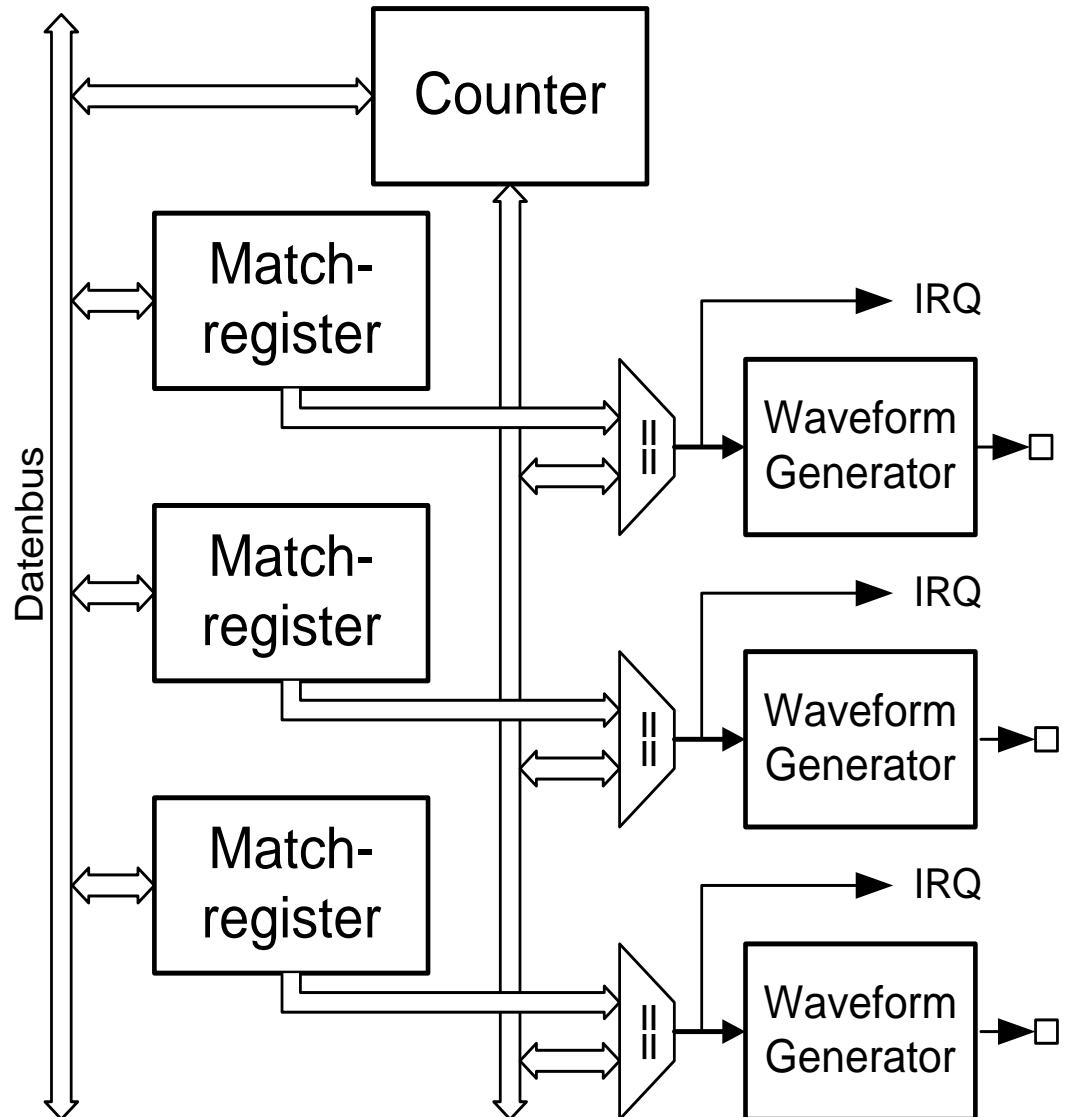


- ▶ **Ablauf:**
  - ▶ Beim Überlauf des Zählers wird Overflow-Interrupt ausgelöst.
  - ▶ In der ISR wird der Zähler auf den Ausgangswert zurückgesetzt.
- ▶ **Probleme:**
  - ▶ Latenz- und Bearbeitungszeit des Interrupts bewirken:
    - Falsche Frequenz
    - Jitter.
  - ▶ Timer kann nur ein Signal zur Zeit erzeugen.

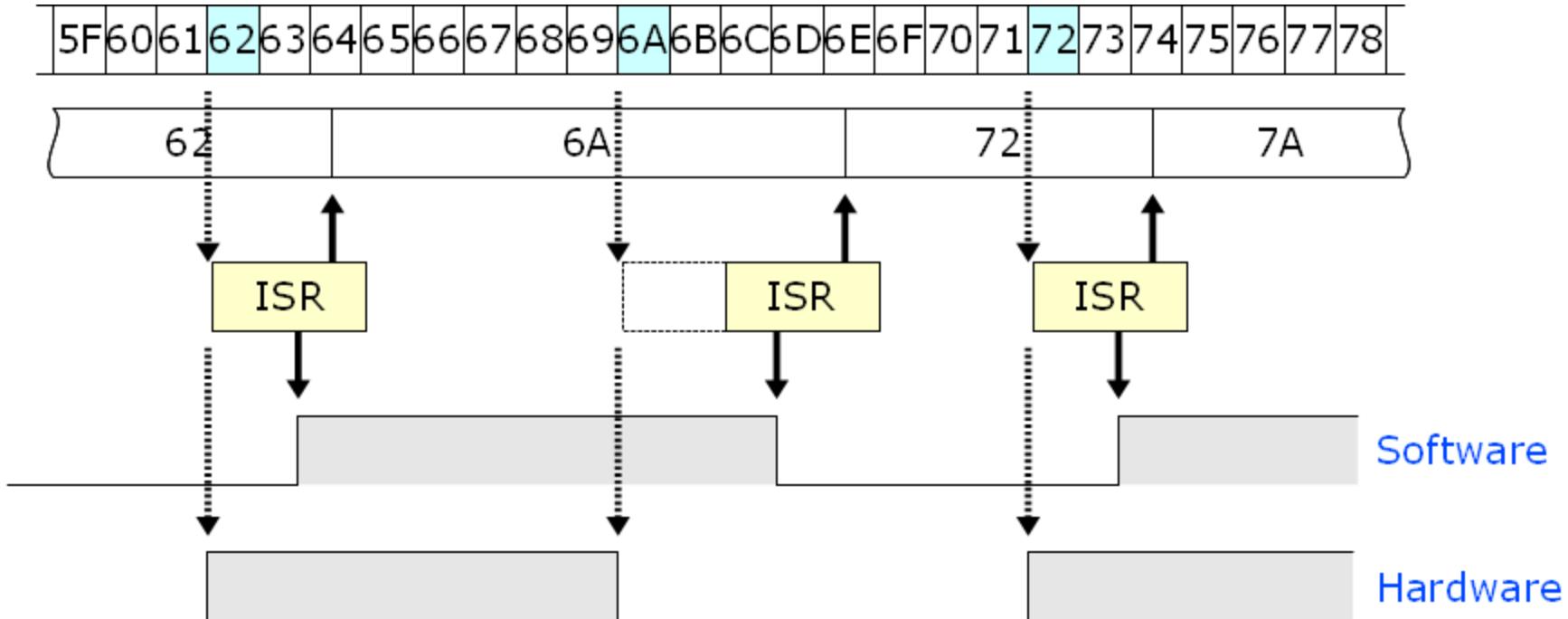


## Timer mit Output-Compare-Funktion

- ▶ Ermöglicht die Erzeugung von mehreren unabhängigen Signalen



## Output-Compare-Funktion: Periodisches Ausgangssignal



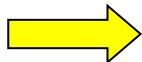
- ▶ **Hardwaregesteuerter Ausgang:**
  - ▶ **taktgenau**
- ▶ **Softwaregesteuerter Ausgang:**
  - ▶ **richtige Frequenz**
  - ▶ **aber Jitter**



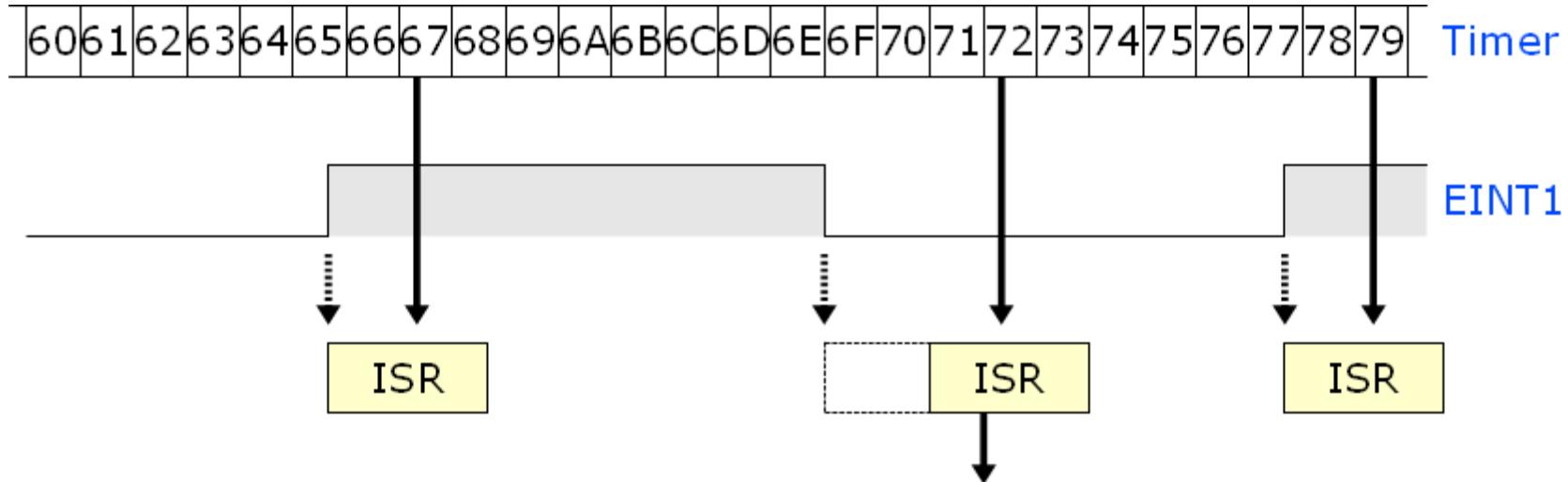
CE WS12

## Übersicht

- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468



## Zeitmessung



- ▶ Idee:
  - ▶ Externes Signal löst Interrupt aus.
  - ▶ In der ISR werden die Zeitpunkte mit Hilfe des intern getakteten Timers bestimmt.
- ▶ Aber Latenz- und Bearbeitungszeiten des Interrupts bewirken:
  - ▶ Ungenaue Messung
  - ▶ Jitter in Messergebnisse

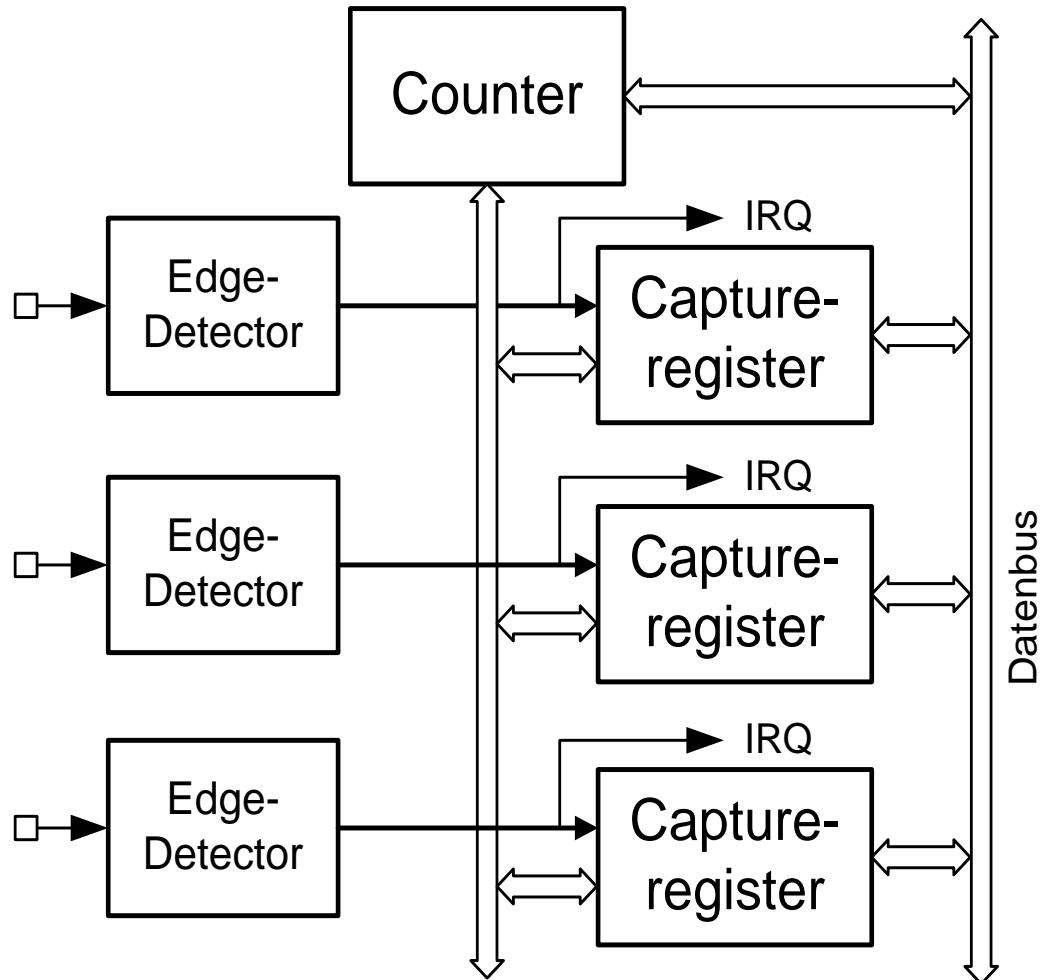


# Timer

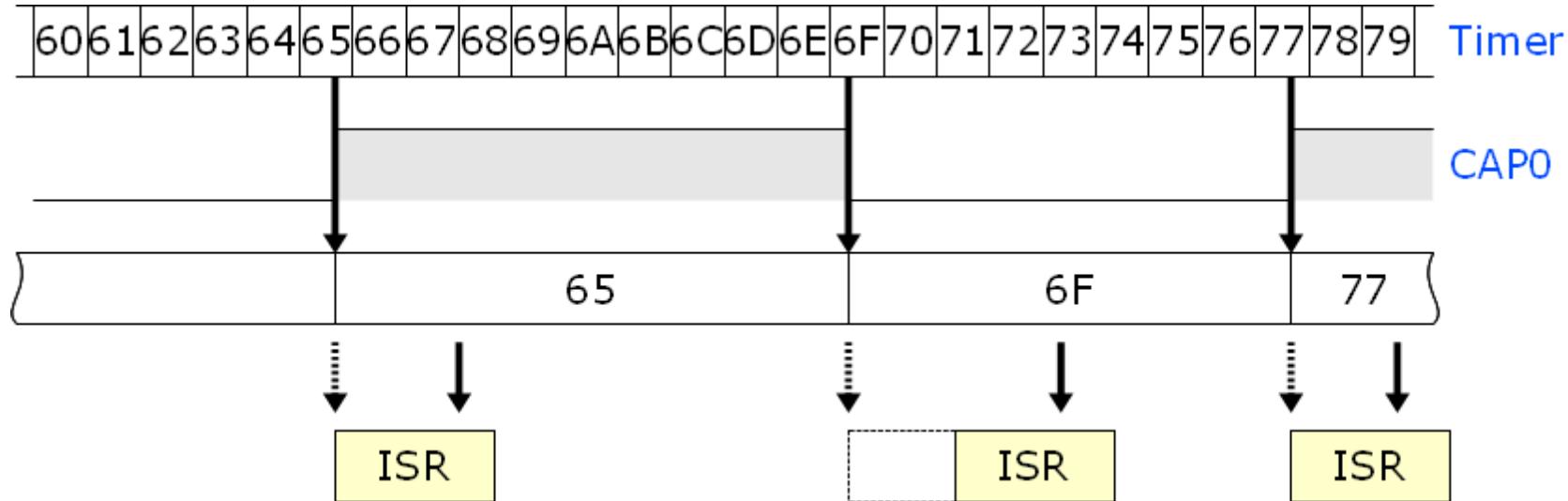
CE WS12

## Timer mit Input-Capture-Funktion

- ▶ Ermöglicht das taktgenaue Messen von externen Ereignissen.



## Input-Capture-Funktion: Zeitmessung



- ▶ Latenz- und Bearbeitungszeit des Interrupts haben keinen Einfluss auf die Messung:
  - ▶ taktgenaue Bestimmung möglich.
- ▶ Aber Ergebnis muss bis zum Eintreffen des nächsten Ereignisses gelesen werden.



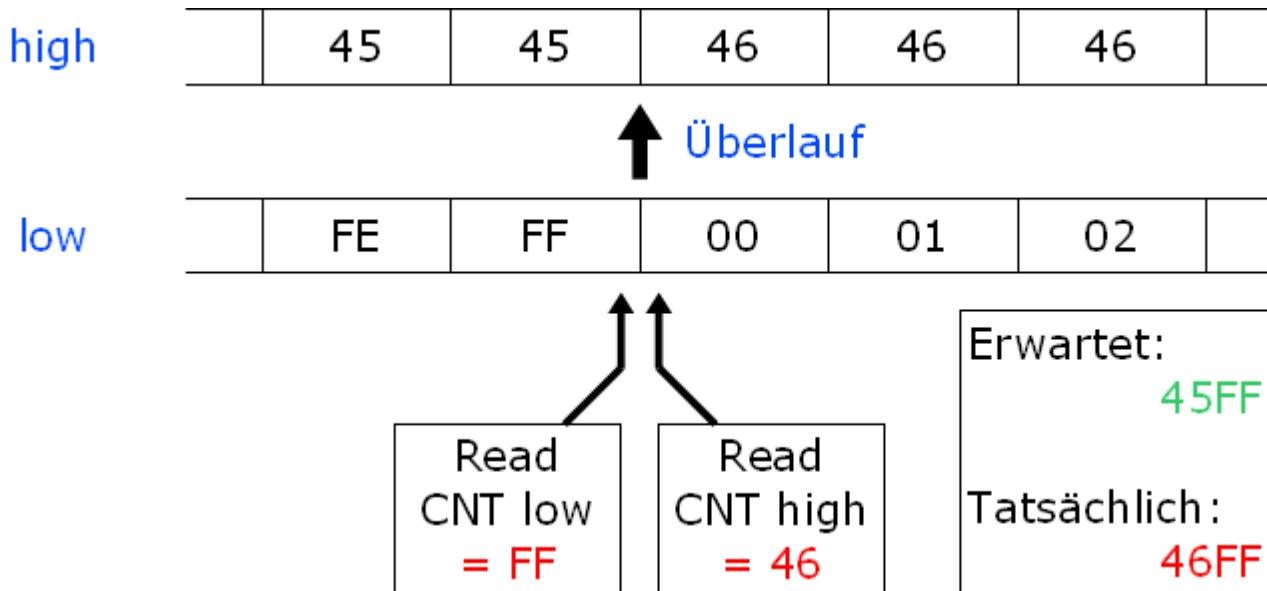
CE WS12

## Übersicht

- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468



## Zugriff auf 16-Bit Zähler mittels 8-Bit Bus

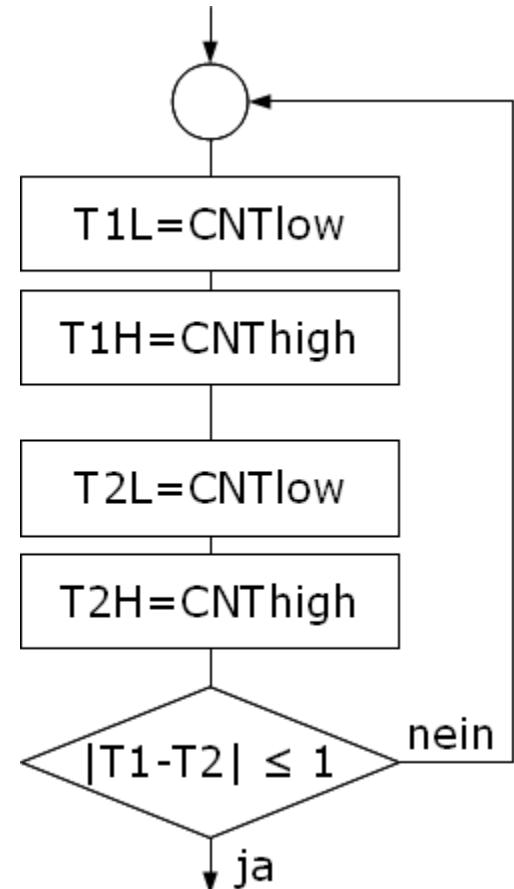


- ▶ Lesen des 16-Bit Registers erfordert zwei Zugriffe.
- ▶ Normalerweise Ungenauigkeit von 1
  - ▶ Bei gleichzeitiger Änderung des High-Registers:
    - Abweichung von 255

CE WS12

## Zugriff auf 16-Bit Zähler mittels 8-Bit Bus: Sicherer Lesezugriff per Software

- ▶ Problematisch:
  - ▶ mindestens 4 Leseoperationen auf Register
  - ▶ Zeitliche Ungenauigkeit des Zugriffs
  - ▶ Achtung:
    - Unterbrechung möglich zwischen Lesezugriffe auf low und high durch IRQ!
- ▶ Probleme bei Schreibzugriffe?



LPC2468: Lese- und Schreibzugriffe unproblematisch,  
da 32-Bit Zugriffe und 32-Bit Register (atomarer Zugriff möglich)



CE WS12

## Übersicht

- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468



## Timer-Bausteine im LPC2468

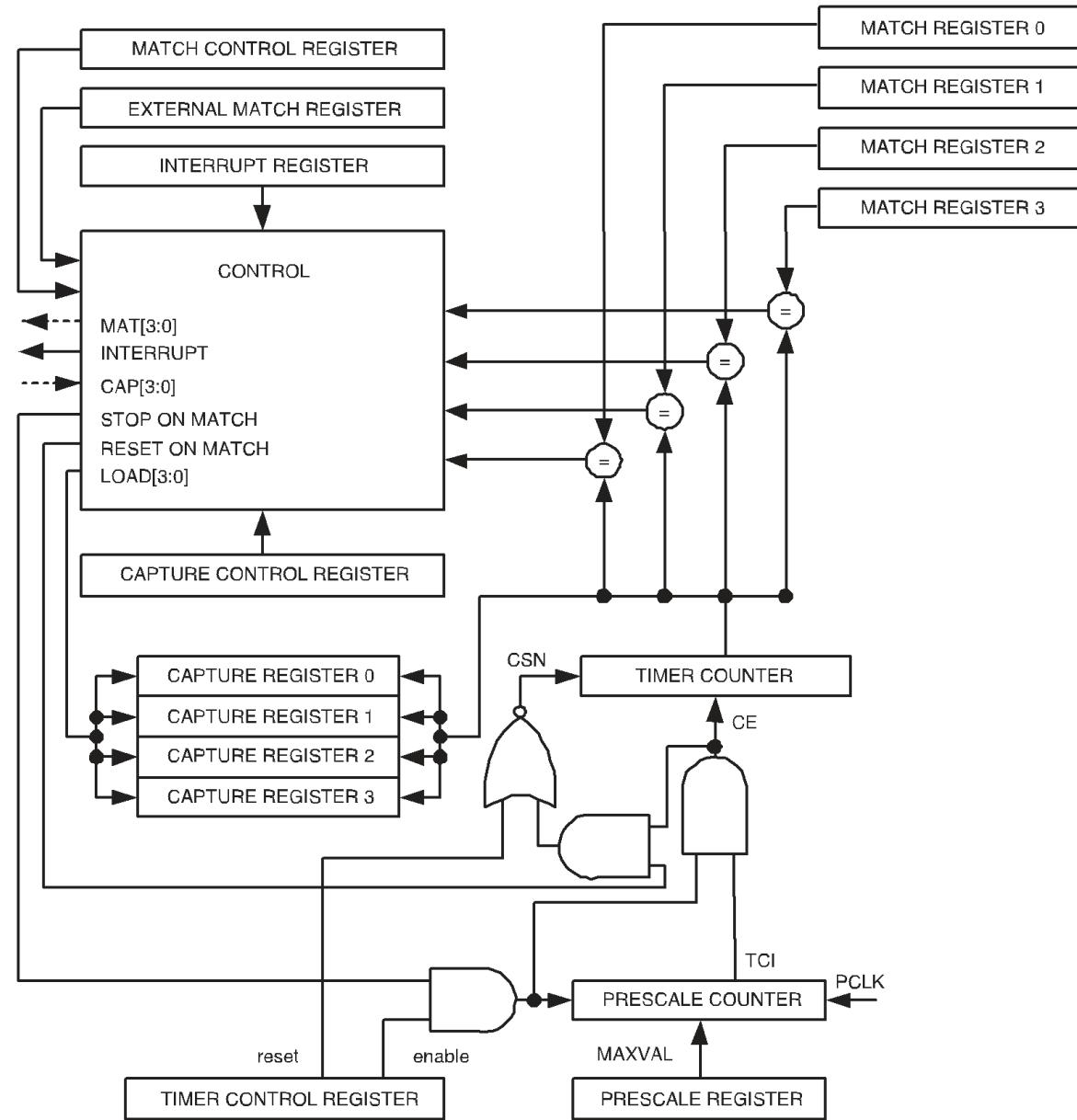
- ▶ **LPC2468 hat 4 Timer-Bausteine: Timer0 bis Timer3.**
- ▶ **Zusätzlich 2 „Pulse Width Modulators“ (PWM).**
- ▶ **Hohe zeitliche Auflösung:**
  - ▶ **Alle Register sind haben 32-Bit.**
  - ▶ **Maximale Taktfrequenz: Systemtakt (typisch 48 MHz).**
- ▶ **4 Capture-Register zum Erfassen externer Signale.**
- ▶ **4 Match-Register mit jeweils 3 Betriebsarten:**
  - ▶ **Kontinuierlicher Betrieb.**
  - ▶ **Stop beim Erreichen des Match-Wertes (Single Shot).**
  - ▶ **Zurücksetzen des Zählers beim Erreichen des Match-Wertes.**
- ▶ **4 Ausgänge / 4 Eingänge pro Timer möglich.**



CE WS12

# Timer/Counter

## Timer-Bausteine im LPC2468: Blockdiagramm





# Timer/Counter

## Timer-Bausteine im LPC2468: Datenregister

- ▶ Timer Counter Register **TIMERn\_TC**
- ▶ Aktueller Zählerstand
- ▶ Prescale Register **TIMERn\_PR**
- ▶ Maximaler Zählerstand des Vorteilerzählers.
- ▶ Prescale Counter Register **TIMERn\_PC**
- ▶ Aktueller Zählerstand des Vorteilerzählers.
- ▶ Match Register **TIMERn\_MR[0/1/2/3]**
- ▶ Wert des Match Registers
- ▶ Capture Register **TIMERn\_CR[0/1/2/3]**
- ▶ Wert des Capture Registers.



## Timer-Bausteine im LPC2468: Steuerregister

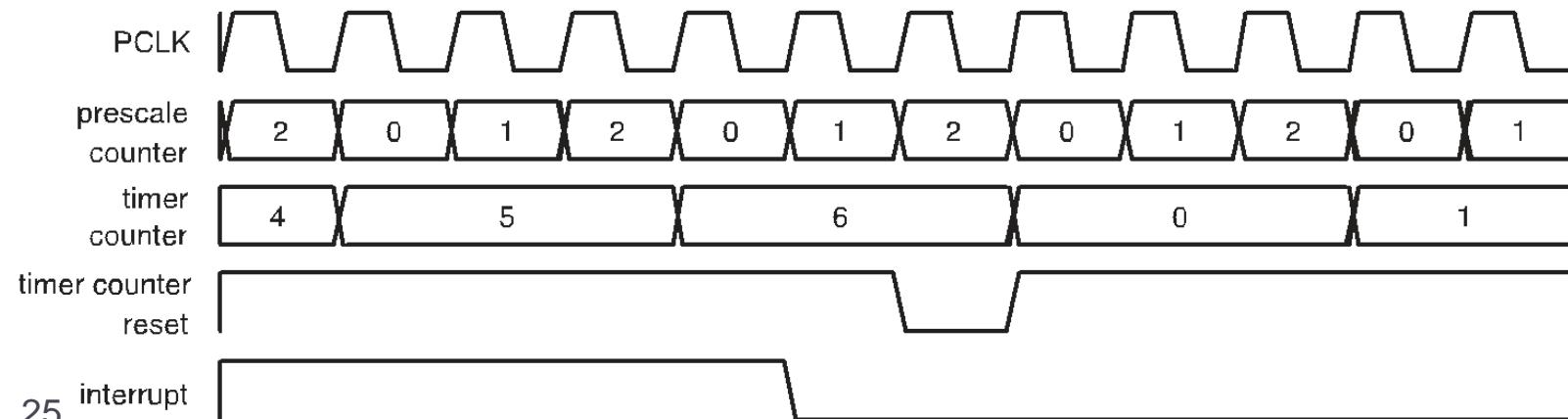
- ▶ Interrupt Register **TIMERn\_IR**
  - ▶ Zeigt an, ob ein Match- oder Capture-Event aufgetreten ist.  
Muss per Software zurückgesetzt werden.
- ▶ Timer Control Register **TIMERn\_TCR**
  - ▶ Reset und Freigabe des Zählers.
- ▶ Count Control Register **TIMERn\_CTCR**
  - ▶ Einstellung Betriebsmode: Zeitgeber oder Zähler.  
Auswahl des Zähleingangs.
- ▶ Match Control Register **TIMERn\_MCR**
  - ▶ Festlegung, was bei einem Match-Event passieren soll:  
Interrupt, Reset des Zählerstandes, Stop des Zählers.
- ▶ Capture Control Register **TIMERn\_CCR**
  - ▶ Festlegung, welche Flanke erfasst werden soll.  
Freigabe des Interrupts.
- ▶ External Match Register **TIMERn\_EMR**
  - ▶ Festlegung, was bei einem Match-Event mit dem externen Match-Ausgang (MATn.m) passieren soll: High, Low, Toggle

## Timer-Bausteine im LPC2468: Anwendungsbeispiel 1

### ► Kontinuierlicher Zähler

```

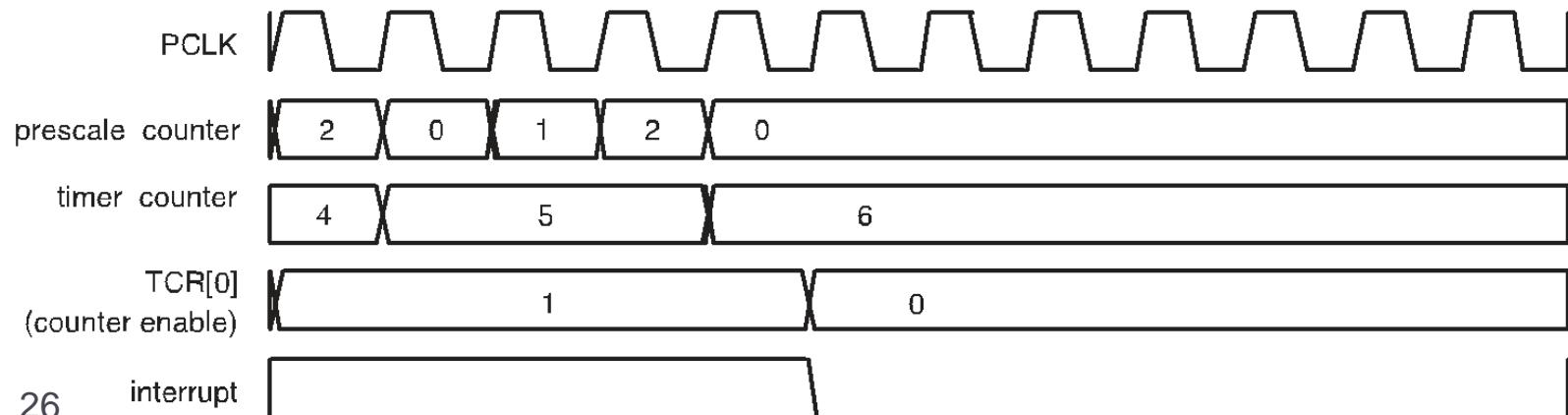
PCOMP      |= (1<<22); // enable timer
PCLKSEL1   |= (1<<12); // set clock to cclk (48 MHz)
TIMER2_TCR = 0x02;      // reset timer
TIMER2_MR0 = 6;          // set match register 0
TIMER2_PR  = 2;          // set prescaler
TIMER2_IR  = 0xff;       // reset all interrupts
TIMER2_MCR = 3<<0;      // interrupt on MR0, reset on MR0,
                         // do not stop timer on match
TIMER2_TCR = 0x01;       // start timer
    
```



## Timer-Bausteine im LPC2468: Anwendungsbeispiel 2

### ▶ Single Shot

```
PCOMP      |= (1<<22); // enable timer
PCLKSEL1   |= (1<<12); // set clock to cclk (48 MHz)
TIMER2_TCR = 0x02;      // reset timer
TIMER2_MR0 = 6;          // set match register 0
TIMER2_PR  = 2;          // set prescaler
TIMER2_IR  = 0xff;       // reset all interrupts
TIMER2_MCR = 5<<0;     // interrupt on MR0, no reset on MR0,
                        // stop timer on match
TIMER2_TCR = 0x01;       // start timer
```





# Timer/Counter

CE WS12

## Timer-Bausteine im LPC2468: Anwendungsbeispiel 3

- ▶ LED2 auf dem TI-Board blinkt mit 1 Sekunde
  - ▶ Verwendung des externen Match Signals MAT0.0

```
PCONP      |= (1<<1);      // enable timer
PCLKSEL0   |= (1<<2);      // set clock to cclk
PINSEL3    |= (3<<24);     // P1.28 is MAT0.0
TIMER0_TCR = 0x02;          // reset timer
TIMER0_PR  = 0;              // set prescaler
TIMER0_IR  = 0xff;           // reset all interrrupts
TIMER0_MCR = 3<<0;          // interrupt on MR0, reset on MR0,
                             // do not stop timer on match
TIMER0_EMR = 3<<4;          // toggle external match bit
TIMER0_TCR = 0x01;           // start timer
TIMER0_MR0 = 48000000 /*Hz*/ / 2 /*Hz*/;
```

# Computer Engineering

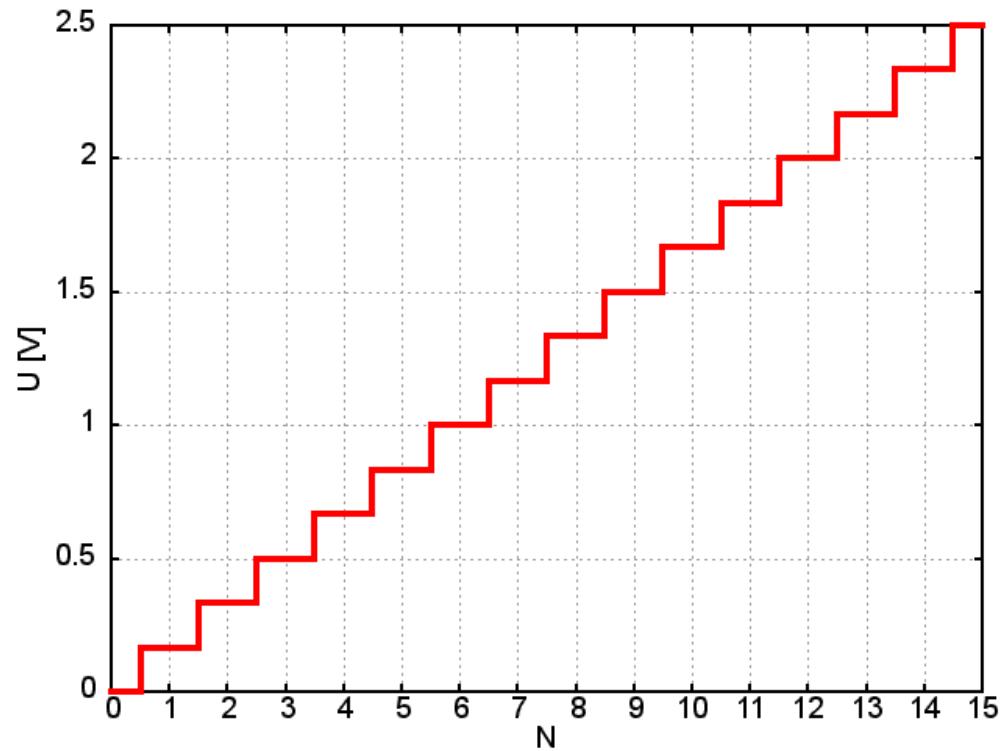
## WS 2012

**PWM**

HTM – SHF - SWR

## Prinzip

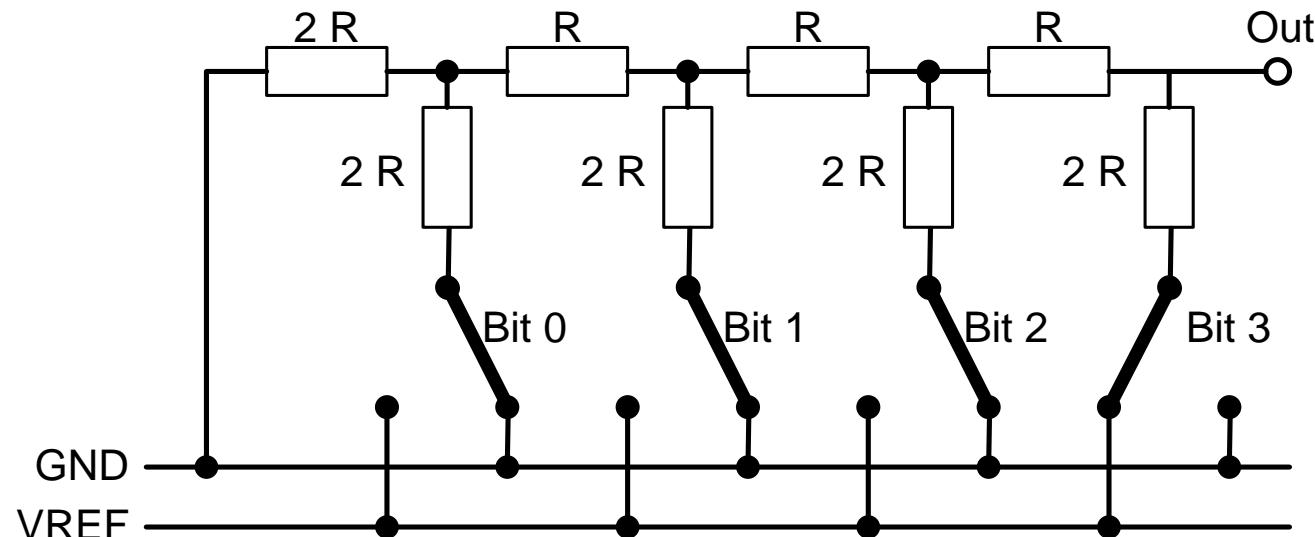
- ▶ Erzeugung einer analogen Spannung
- ▶ Eingang:
  - ▶ Integerzahl  $N$  im Bereich  $N_{\min}$  bis  $N_{\max}$ ,  
z.B. M-Bit Wandler: 0 bis  $2^M - 1$
- ▶ Ausgang:
  - ▶ Analoge Spannung  $U$   
im Bereich  $U_{\min}$  bis  $U_{\max}$
  - ▶ Linear abhängig von  $N$



CE WS12

## Prinzipieller Aufbau

- ▶ R/2R-Netzwerke





## LPC2468: DAC

- ▶ Eigenschaften:
  - ▶ 10-Bit Digital-Analog Wandler
  - ▶ Separate Anschlüsse für Spannungsversorgung:  $V_{DDA}$ ,  $V_{SSA}$
  - ▶ Separater Anschluss für Referenzspannung:  $V_{REF}$ 
    - Im Praktikum:  $V_{REF} = 3,3$  Volt
  - ▶  $A_{OUT} = \text{VALUE} / 1024 * V_{REF}$
- ▶ Initialisierung:
  - ▶ Takt einstellen: PCLKSEL0
  - ▶ Ausgangspin AOUT aktivieren und konfigurieren:  
PINSEL1 und PINMODE1.
- ▶ Ausgabe:
  - ▶ Wert in das Register DACR schreiben.  
Bitanordnung beachten.



CE WS12

## DAC: Nachteile

- ▶ Aufwändige analoge Schaltung
  - ▶ Kostenintensiv
  - ▶ Störanfällig

## Pulsweitenmodulation (PWM)

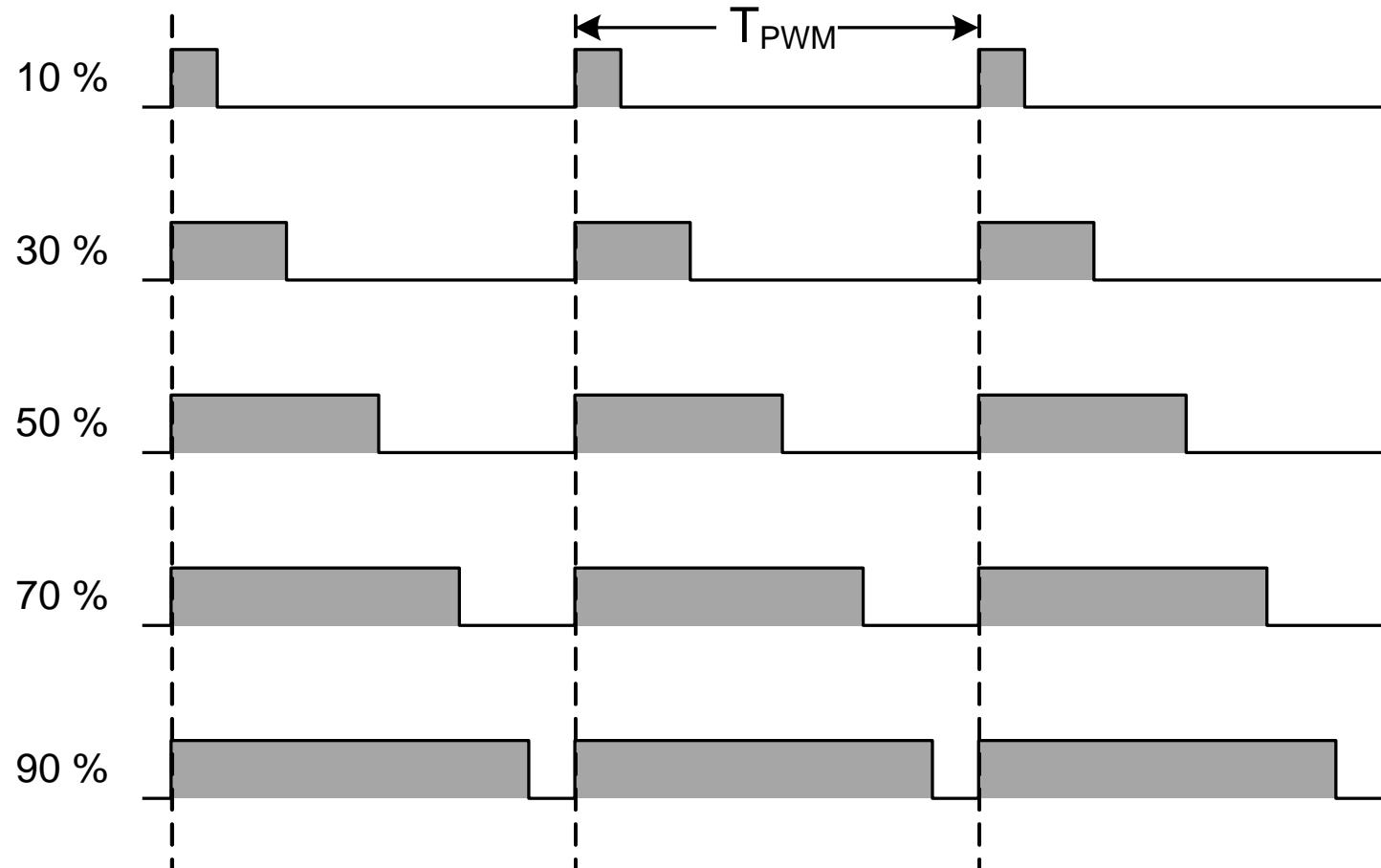
- ▶ Kann mit reinen digitalen Schaltungen realisiert werden
- ▶ Kostengünstig in Mikrocontroller integrierbar.
- ▶ Vielfältige Einsatzmöglichkeit, z.B.:
  - ▶ Erzeugung analoger Signale,
  - ▶ Ansteuerung von Gleichstrommotoren,
  - ▶ Ansteuerung von Schritt- und bürstenlosen Motoren,
  - ▶ Signalübertragung (z.B.: Funkfernsteuerungen).



CE WS12

## Grundprinzip

- ▶ Signal mit vorgegebener Frequenz und
- ▶ variabler Pulspausenzeit.



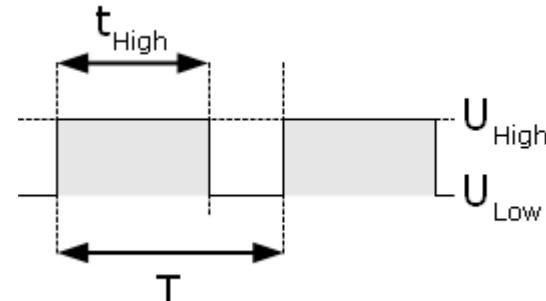


CE WS12

## Tastverhältnis (Duty Cycle)

- Verhältnis der Dauer des High-Zustandes zur Periodendauer

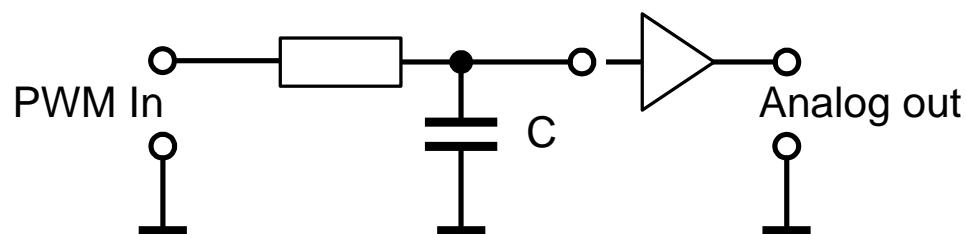
$$D = \frac{t_{High}}{T}$$



## Spannungsmittelwert

$$U_m = U_{Low} + D \cdot (U_{High} - U_{Low})$$

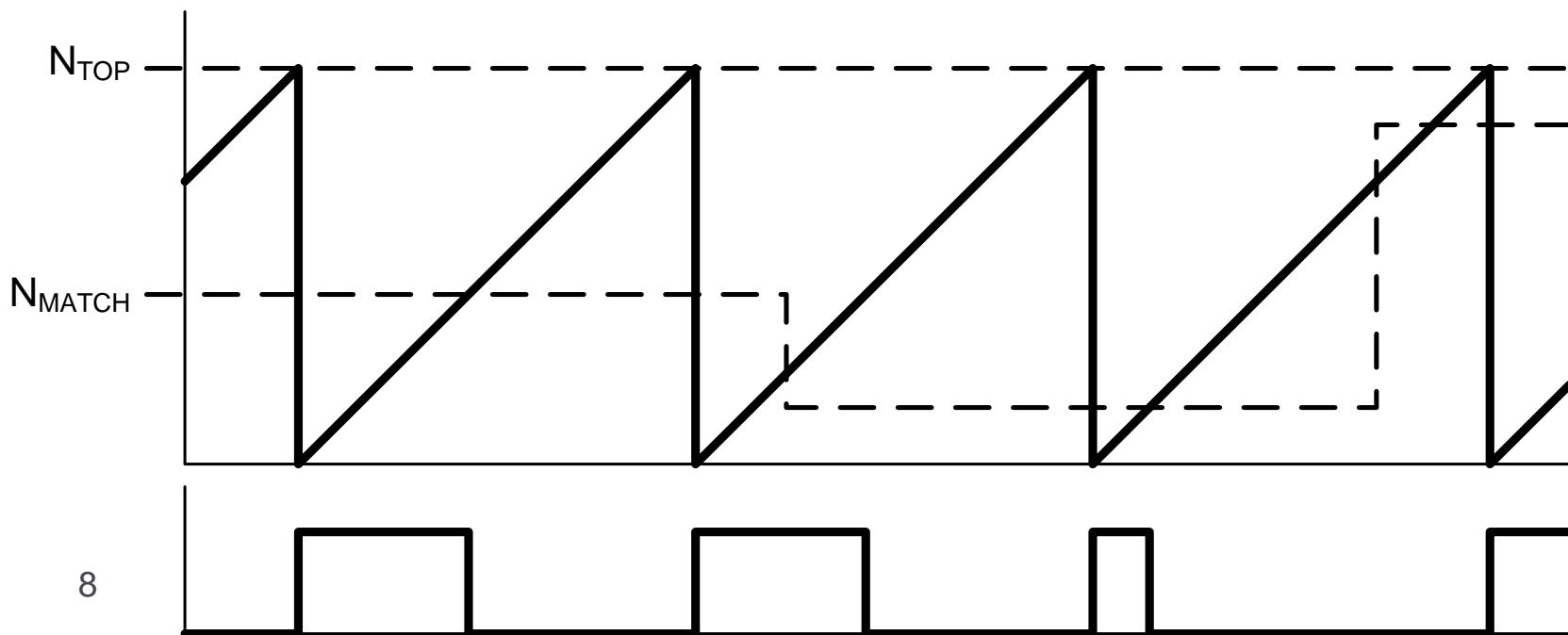
- Lineare Beziehung zwischen  $U_m$  und  $D$ !
- Mittelwertberechnung mit Hilfe eines Tiefpassfilters



CE WS12

## Tastverhältnis (Duty Cycle)

- ▶ Basiert auf Zähler mit M Bit.
- ▶ Zähler wird mit festem Takt  $f_{Takt}$  betrieben.
- ▶ Zähler wird beim Erreichen von  $N_{TOP}$  zurückgesetzt.
- ▶ Steuerung des PWM-Ausgangs:
  - ▶ z.B.: High beim Zurücksetzen des Zählers und
  - ▶ Low beim Erreichen des Match-Registers.





CE WS12

## Abbildung auf physikalische Größen

- ▶ Meist linearer Zusammenhang.
- ▶ Beispiel Temperatursollwert (z.B. für Heizung):
  - ▶ Gefordert:      Bereich von -40 °C bis 100 °C  
                         Auflösung 0.2 °C
  - ▶ Realisierung:    PWM-Zähler      Temperatur
|  |  |
| --- | --- |
| 0 | -40 °C |
| 200 | 0 °C |
| 700 | 100 °C |
  - ▶ Mindestens erforderlich: 10-Bit Zähler
  - ▶ Annahme Taktfrequenz  $f_{Takt} = 8 \text{ MHz}$
  - ▶  $f_{PWM} = f_{Takt} / N_{TOP} = 11.4 \text{ kHz}$



CE WS12

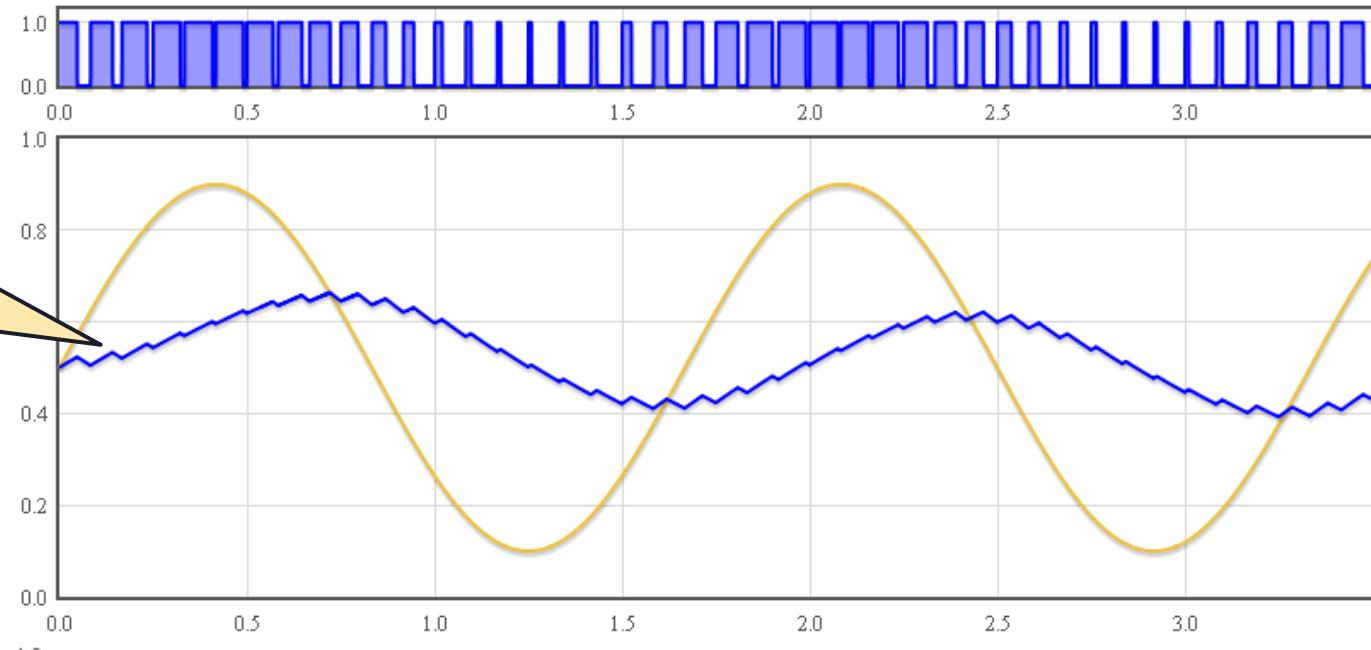
## Kenngrößen

- ▶  $f_{\text{PWM}}$  Frequenz der PWM-Ausgangsimpulse.
- ▶  $f_{\text{Takt}}$  Taktfrequenz des PWM-Zählers.
- ▶ Auflösung (Resolution):
- ▶ Bereich (Range):
- ▶ Einschwingzeit (settling time):

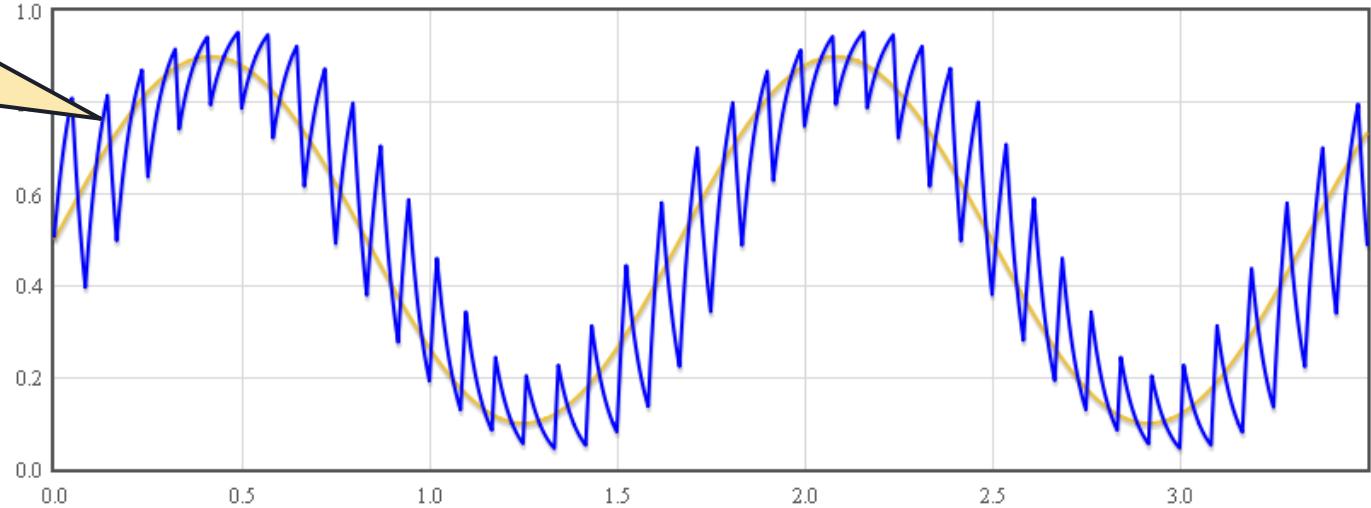
CE WS12

## Optimierung Tiefpassfilter

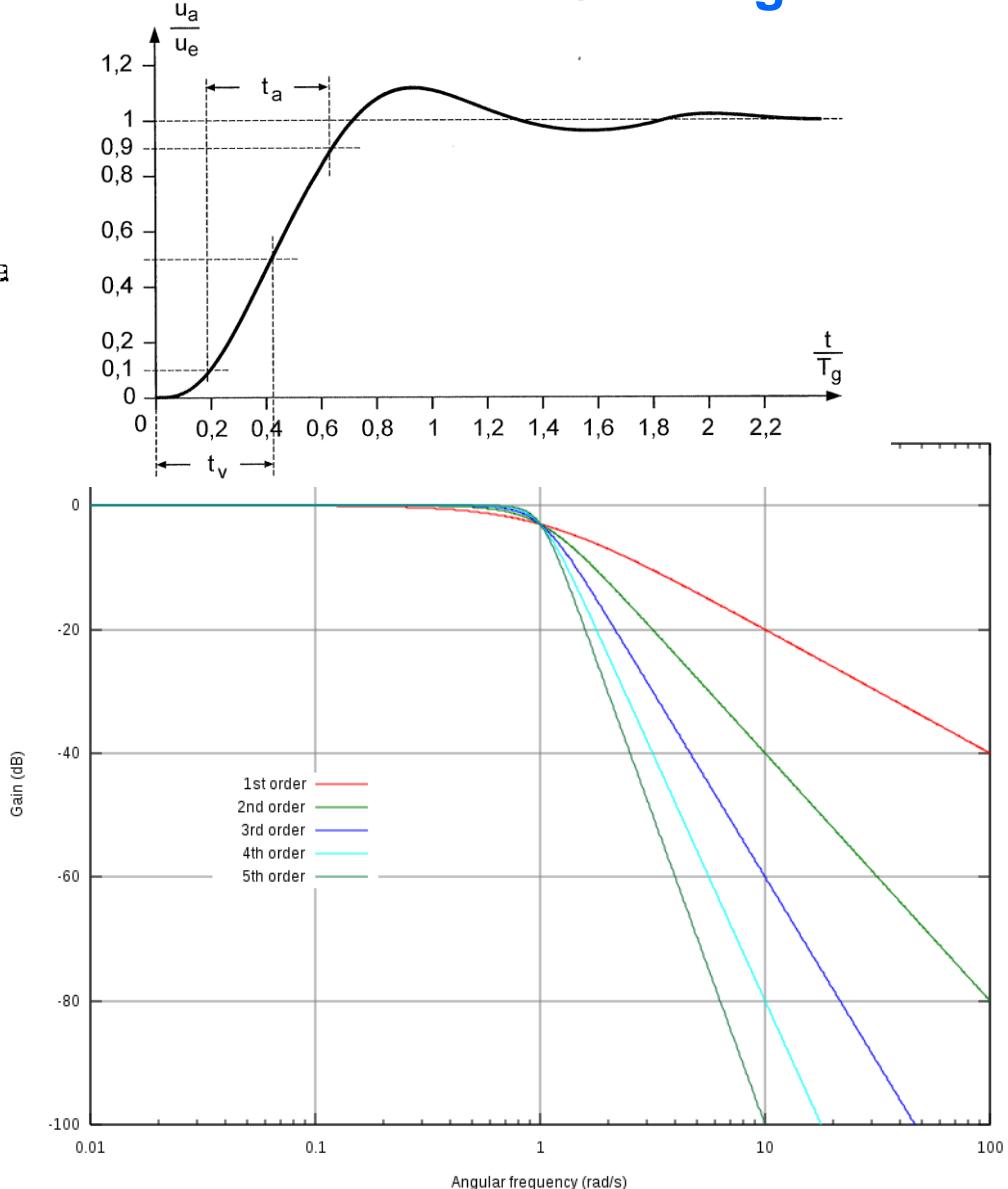
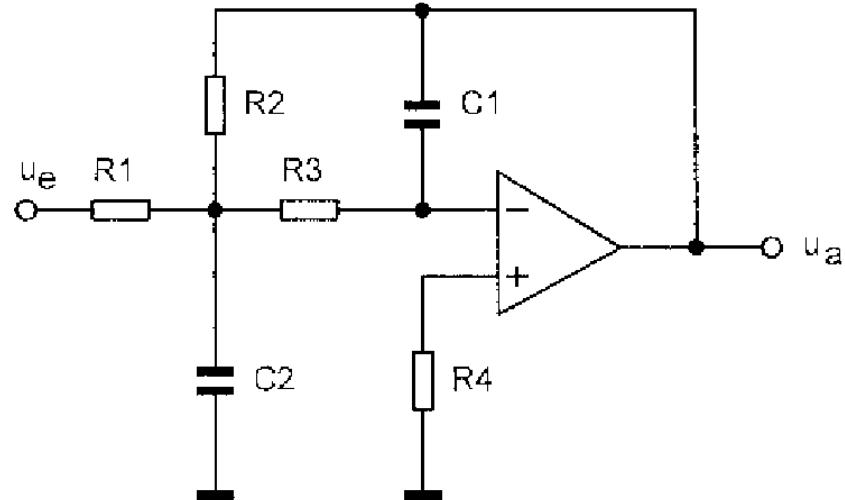
Grenzfrequenz  
zu niedrig



Grenzfrequenz  
zu hoch



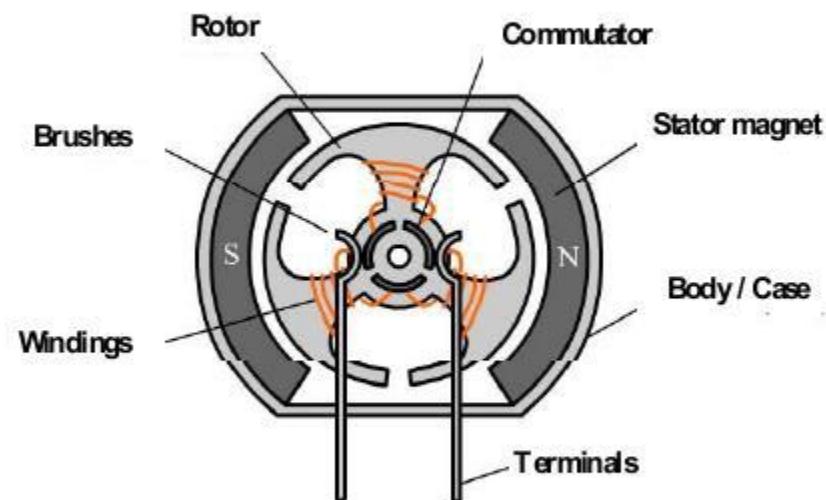
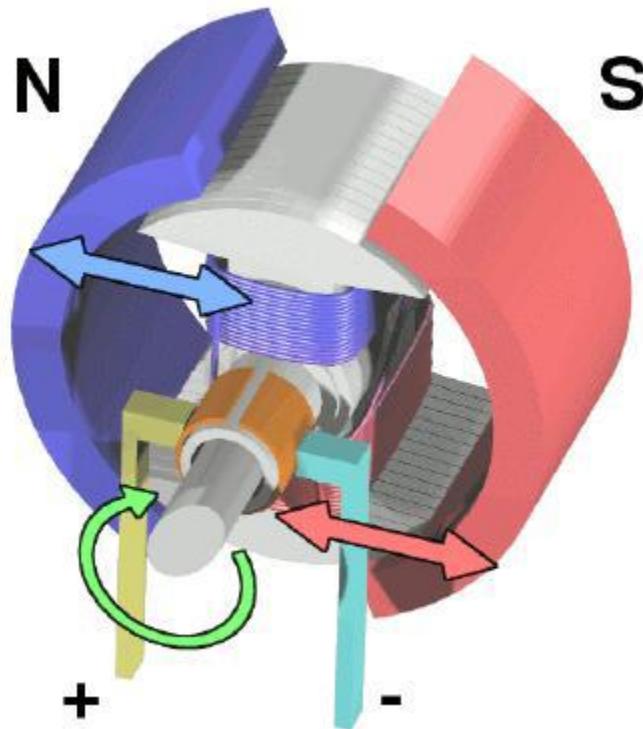
## Optimierung Tiefpassfilter, Butterworthfilter zweiter Ordnung



Aus: Trampert,  
„Messen, Steuern und Regeln  
mit AVR Mikrocontrollern“  
und Wikipedia



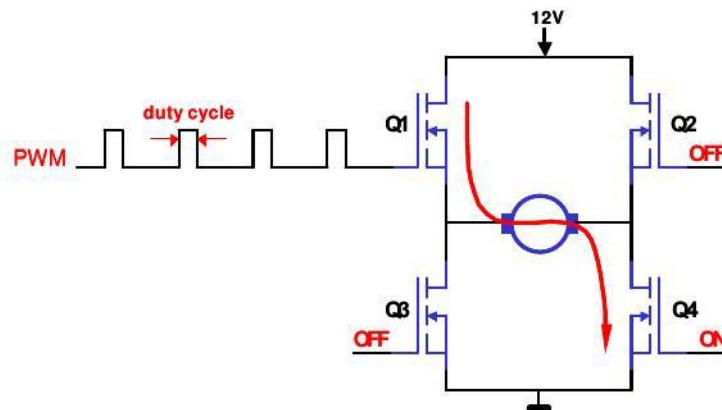
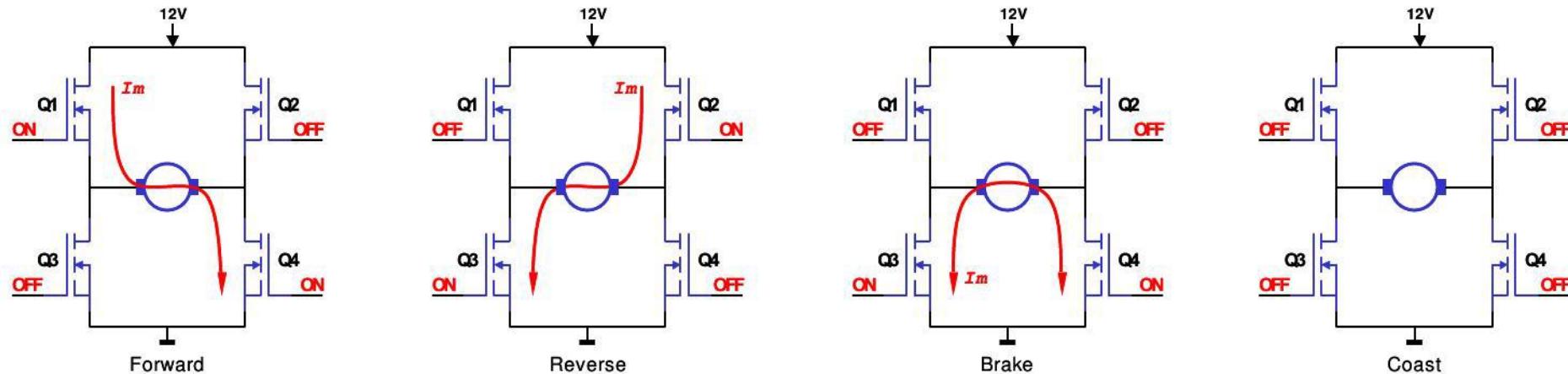
## Anwendung PWM: Ansteuerung von Gleichstrommotoren





CE WS12

## Anwendung PWM: Ansteuerung von Gleichstrommotoren

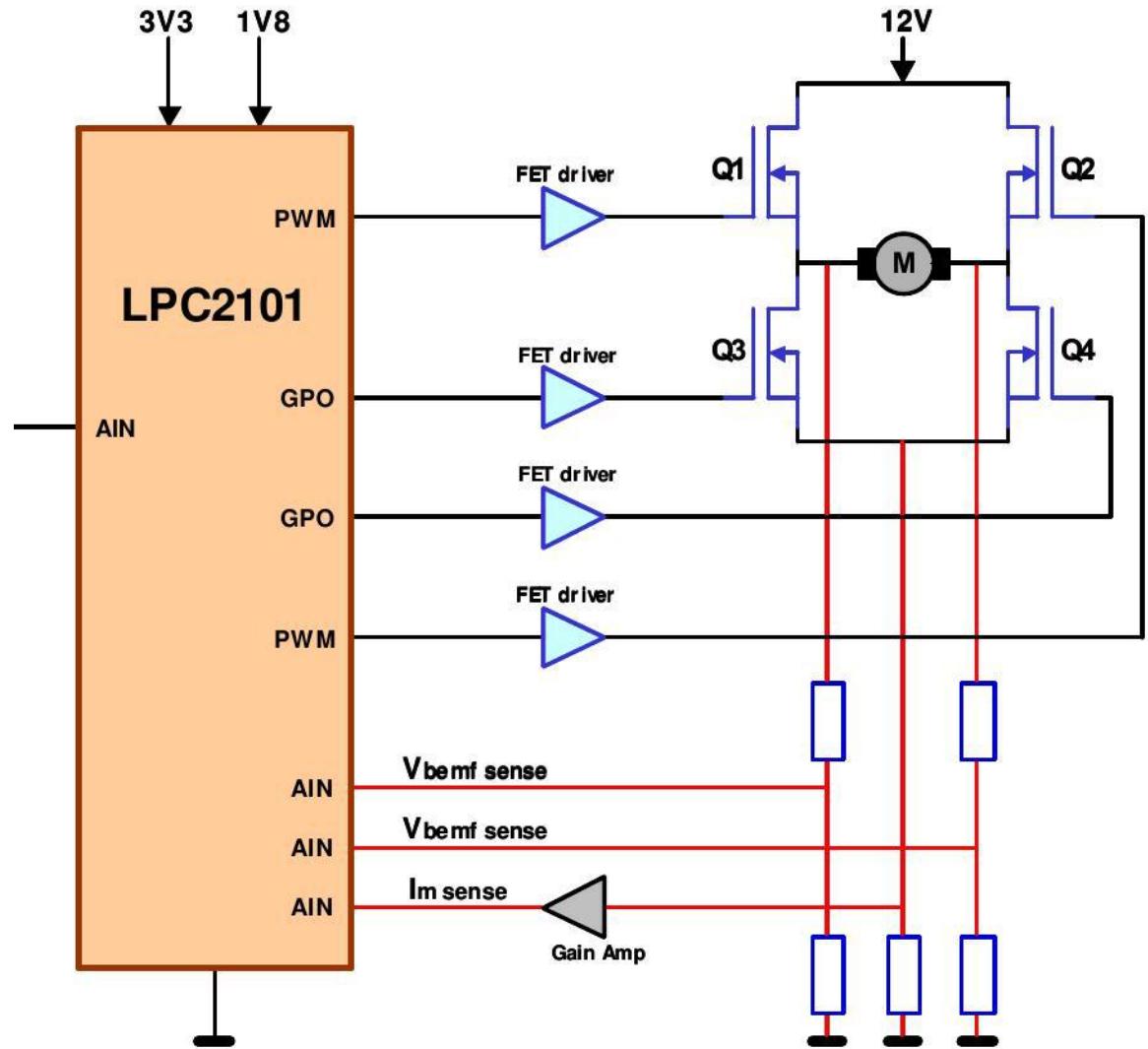




# PWM

CE WS12

## Anwendung PWM: Ansteuerung von Gleichstrommotoren

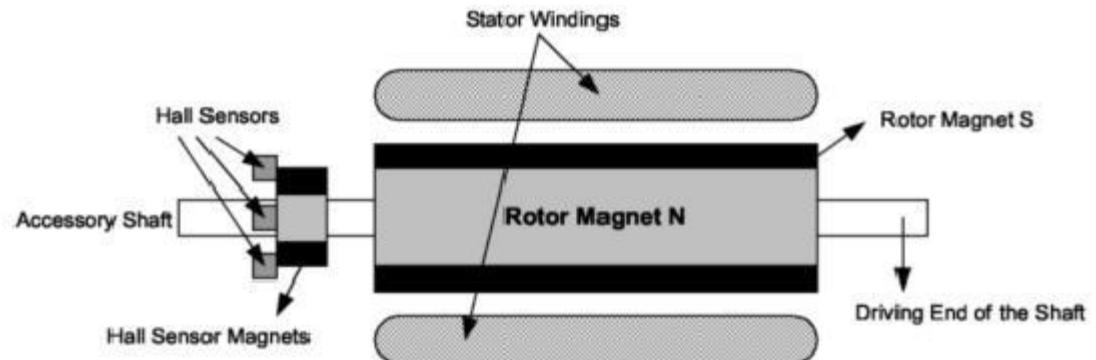
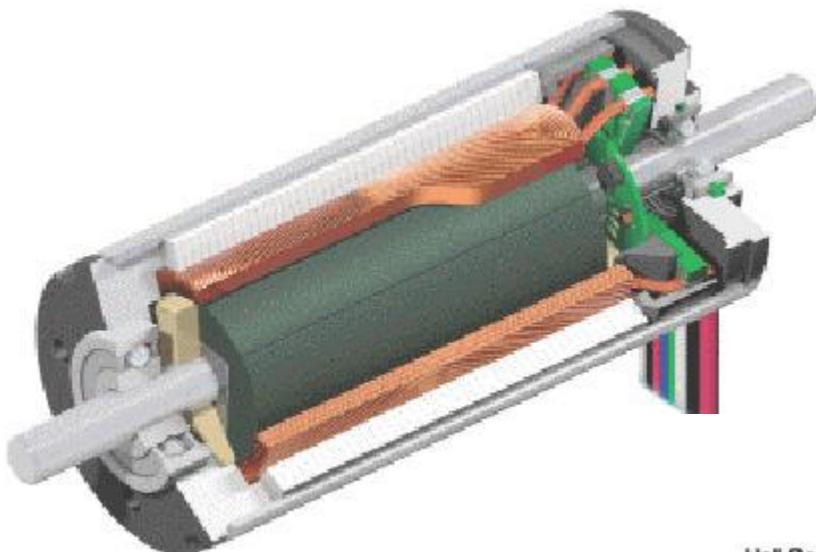




# PWM

CE WS12

## Anwendung PWM: Ansteuerung bürstenloser Motoren

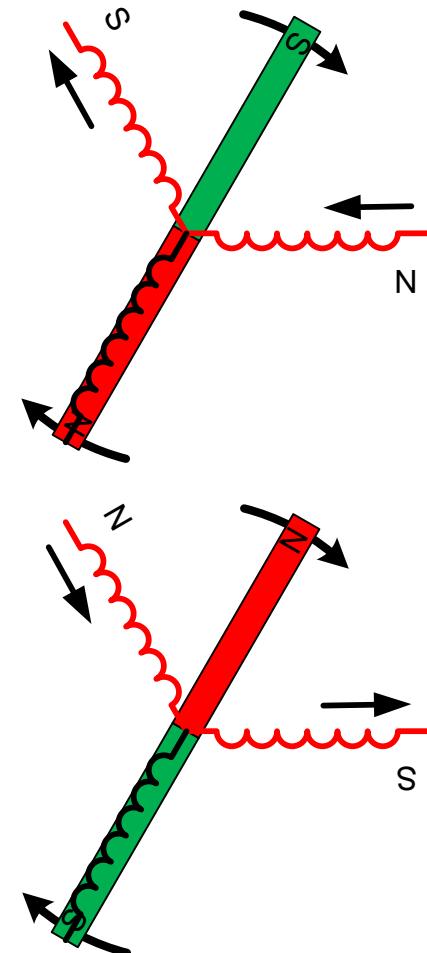
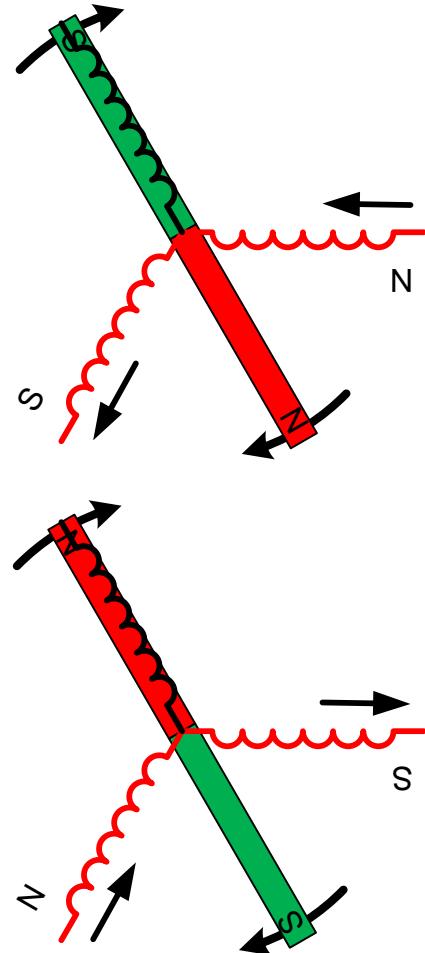
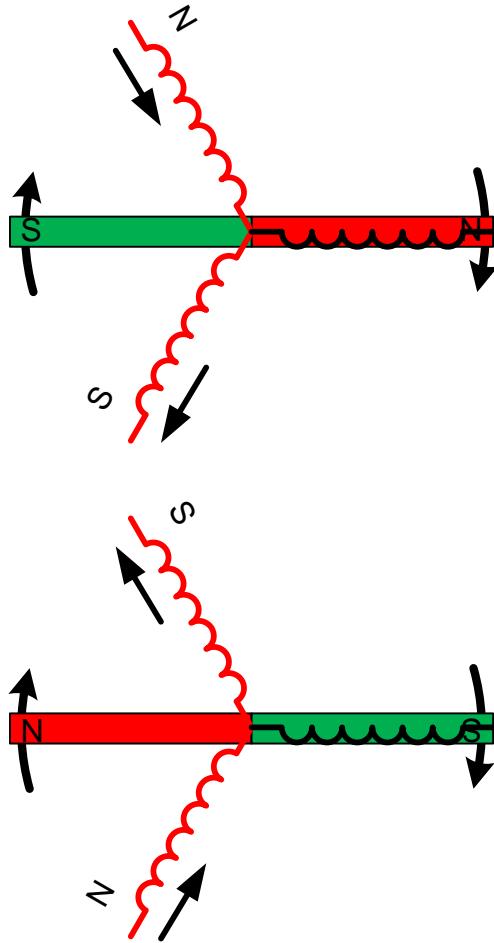




# PWM

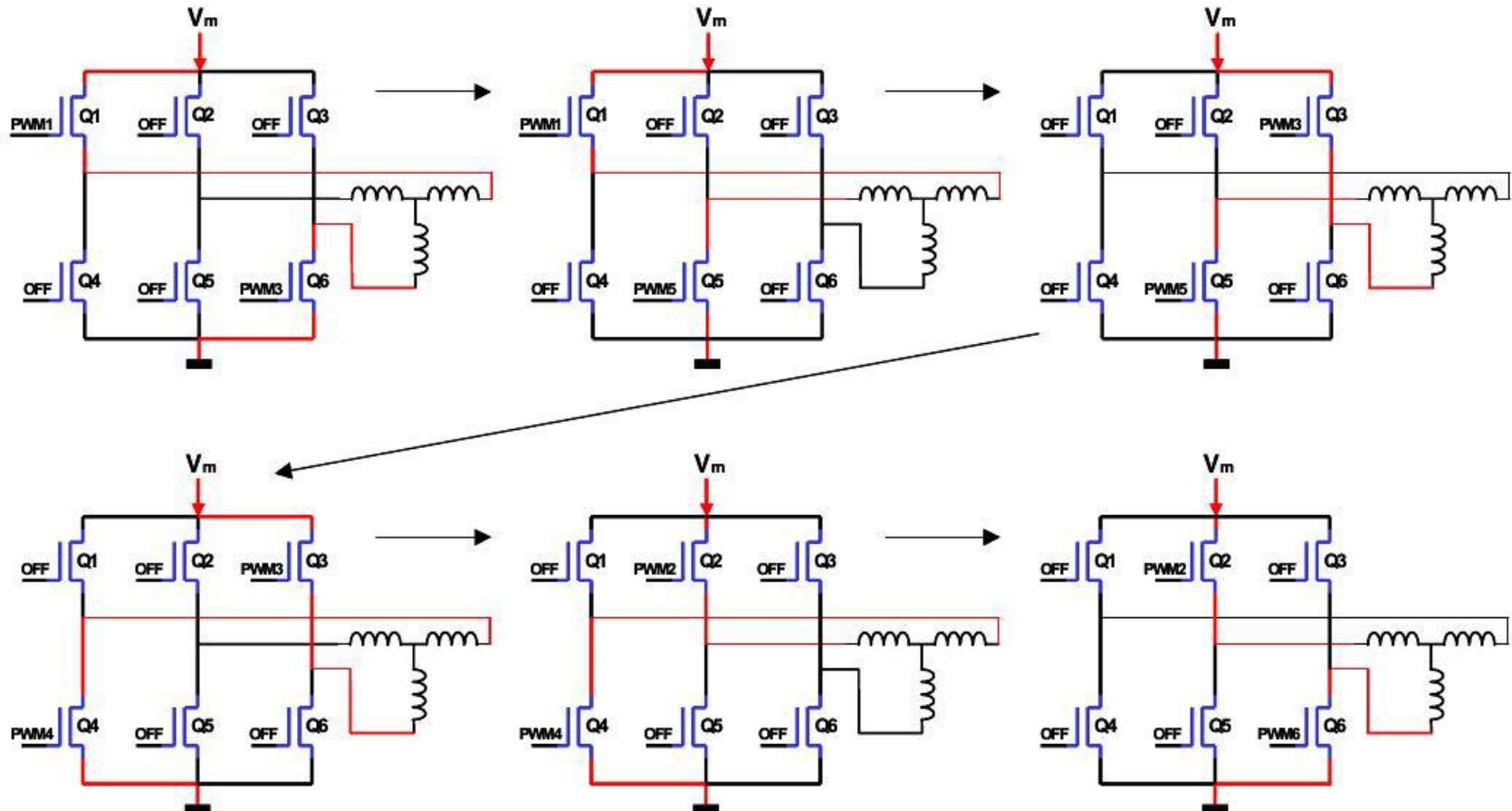
CE WS12

## Anwendung PWM: Ansteuerung bürstenloser Motoren





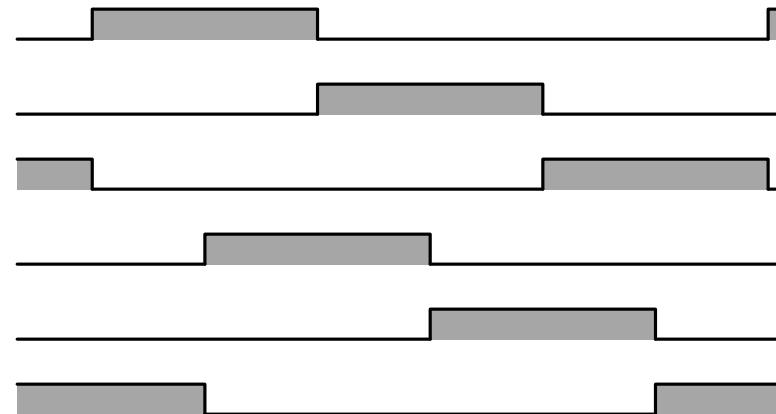
## Anwendung PWM: Ansteuerung bürstenloser Motoren



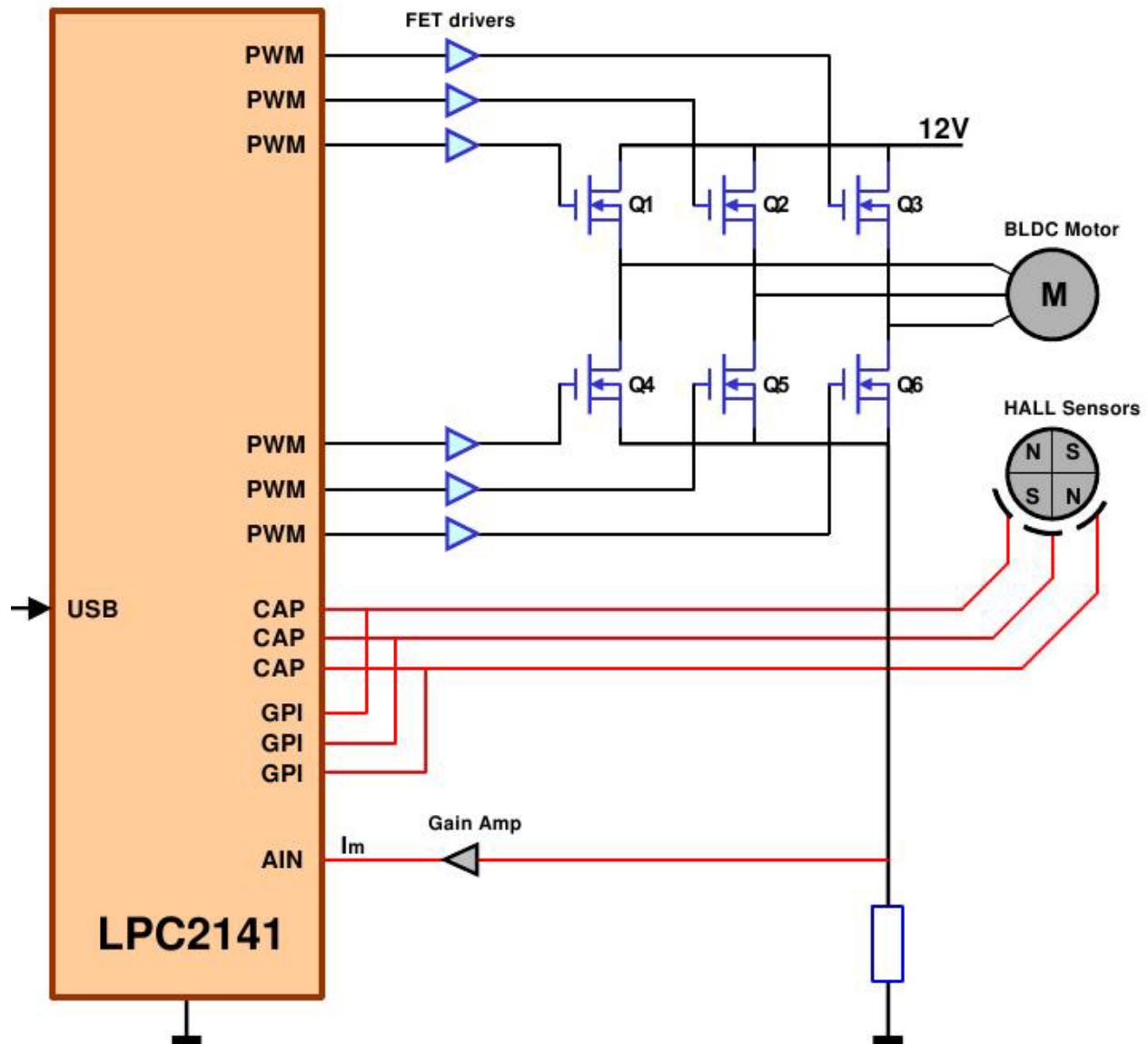


CE WS12

## Anwendung PWM: Ansteuerung bürstenloser Motoren



## Anwendung PWM: Ansteuerung bürstenloser Motoren





CE WS12

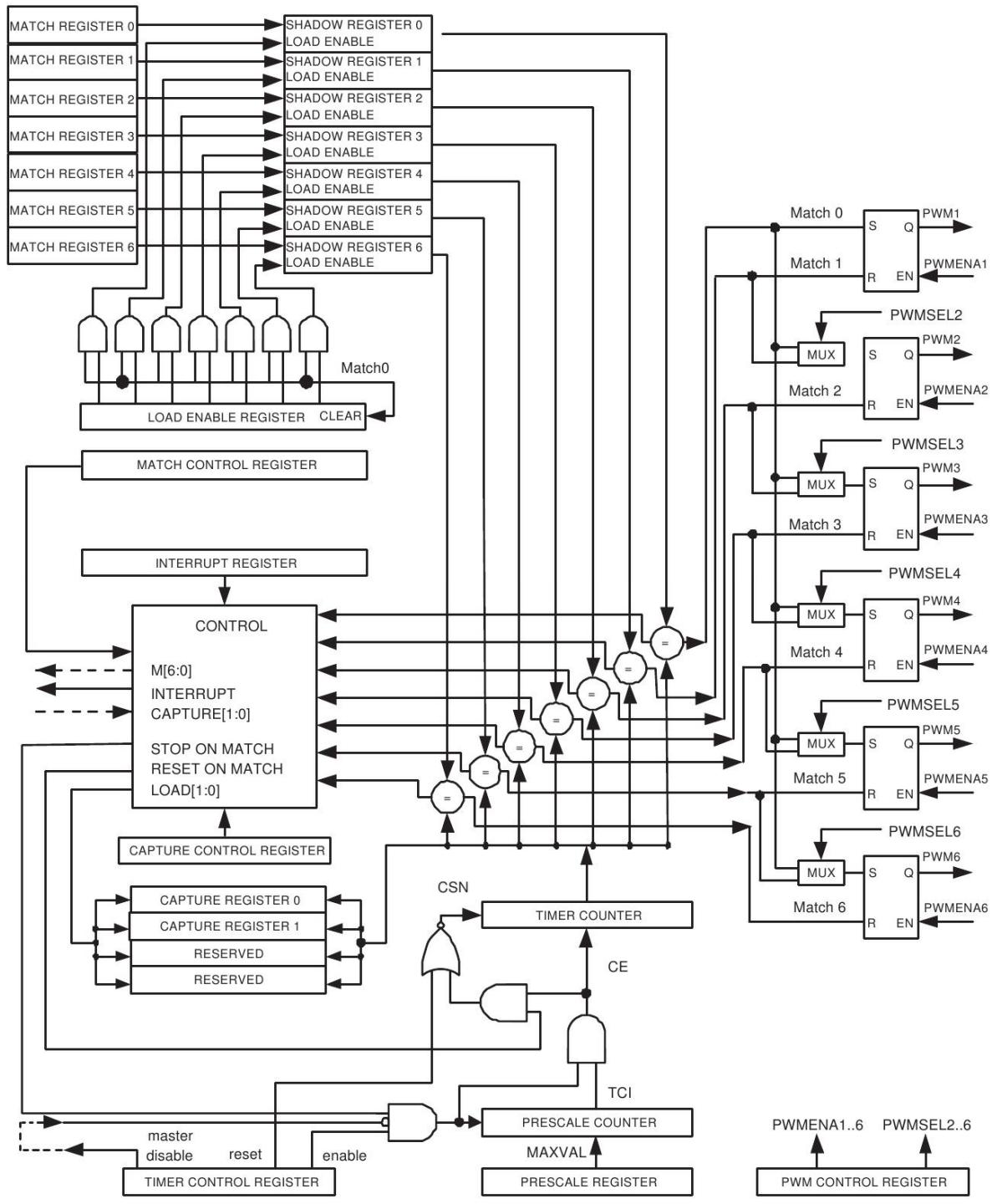
## LPC2468 PWM: Eigenschaften

- ▶ 2 identische PWM-Komponenten
- ▶ Erweiterung der Timer/Counter-Komponenten
- ▶ Zwei Betriebsarten der Ausgänge:
  - ▶ Single edge controlled (bis zu 6 Ausgänge)
  - ▶ Double edge controlled (bis zu 3 Ausgänge)

## LPC2468 PWM:

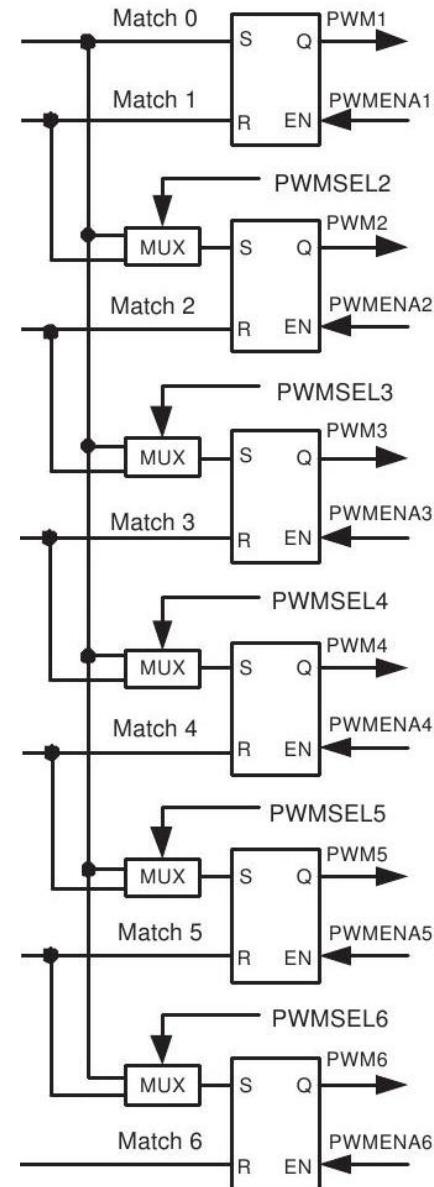
### Erweiterung gegenüber Timer/Counter

- ▶ Reset Timer/Counter statt auf 0 auf 1.
- ▶ Zusätzliche Steuerung für PWM-Ausgänge.
- ▶ 7 Match Register.
- ▶ Jedes Match-Register hat ein Shadow-Register.
- ▶ Zusätzliches Register zum Freigeben neuer Match-Werte.
- ▶ Freigegebene Werte werden erst mit nächstem Match 0 – Ereignis übernommen.



## LPC2468 PWM: PWM-Ausgänge

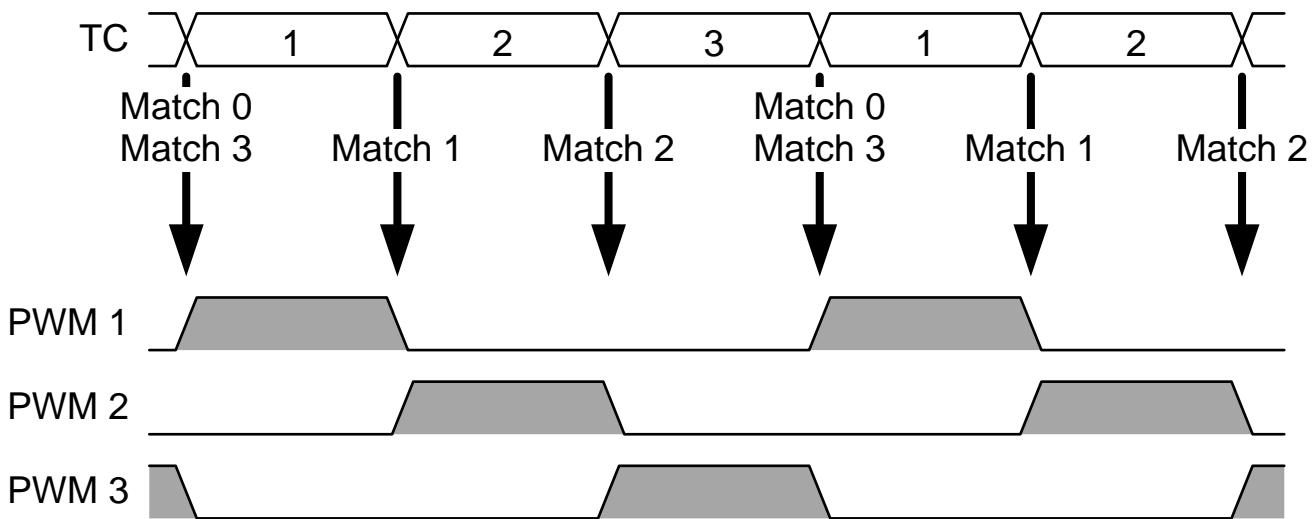
- ▶ Jeder Ausgang wird durch RS-Flipflop gesteuert
- ▶ **Single edge controlled:**
  - ▶ Ausgang x wird aktiviert durch Match 0
  - ▶ Ausgang x wird zurückgesetzt durch Match x
- ▶ **Double edge controlled:**
  - ▶ Ausgang x wird aktiviert durch Match x-1
  - ▶ Ausgang x wird zurückgesetzt durch Match x
- ▶ **Steuerbits:**
  - ▶ **PWMENAx:**
    - 0: Ausgang gesperrt
    - 1: Ausgang freigegeben
  - ▶ **PWMSELx:**
    - 0: Single edge controlled
    - 1: Double edge controlled



# PWM

CE WS12

## Beispiel: Lauflicht



```

PCONP    |= (1<<6);                      // Enable PWM1
PCLKSEL0 |= (1<<12);                     // Clock == CCLK (48 MHz)
PINSEL4  |= (1<<0) | (1<<2) | (1<<4);   // P2.0 ist PWM1.1, P2.1 ist PWM1.2, P2.2 ist PWM1.3
PWM1_PR  = 48000000-1;                      // Prescaler, PWM-Takt = 1 Hz
PWM1_MCR = (1<<0) | (1<<1);              // Interrupt on PWMMR0, Reset on PWMMR0
PWM1_PCR = (1<<2) | (1<<3) |             // PWM1.2 double edge, PWM1.3 double edge,
      (1<<9) | (1<<10) | (1<<11);        // PWM1.1 PWM1.2 PWM1.3 enabled
PWM1_MR0 = 3;                                // PWM-Period
PWM1_MR1 = 1;                                // PWM-Match 1
PWM1_MR2 = 2;                                // PWM-Match 2
PWM1_MR3 = 3;                                // PWM-Match 3
PWM1_LER = (1<<0) | (1<<1) |             // Latch Enable
      (1<<2) | (1<<3);

PWM1_TCR = (1<<0) | (1<<3);               // Counter Enable, PWM Enable

```