Computer Engineering
WS 2012

Interruptverarbeitung

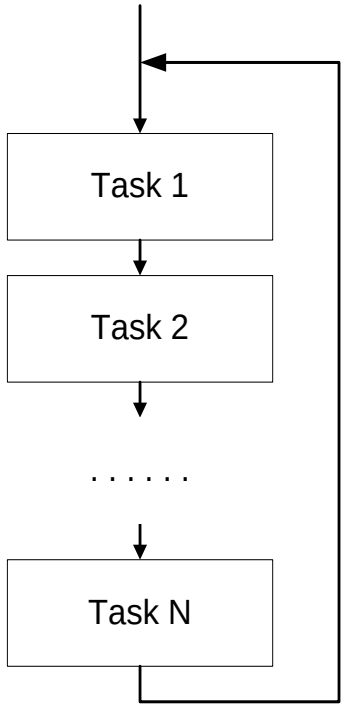
HTM – SHF - SWR

Interrupt-verarbeitung

Programmaufbau ohne Betriebssystem

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

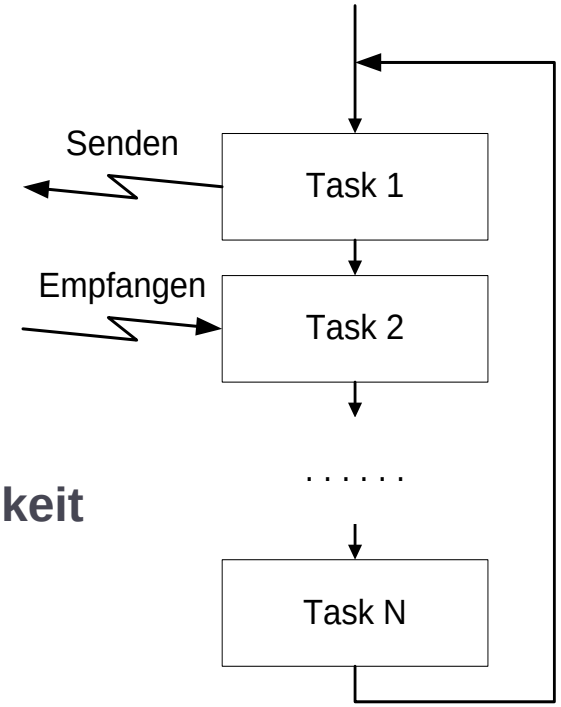
  Summe der maximalen Reaktionszeiten aller Tasks

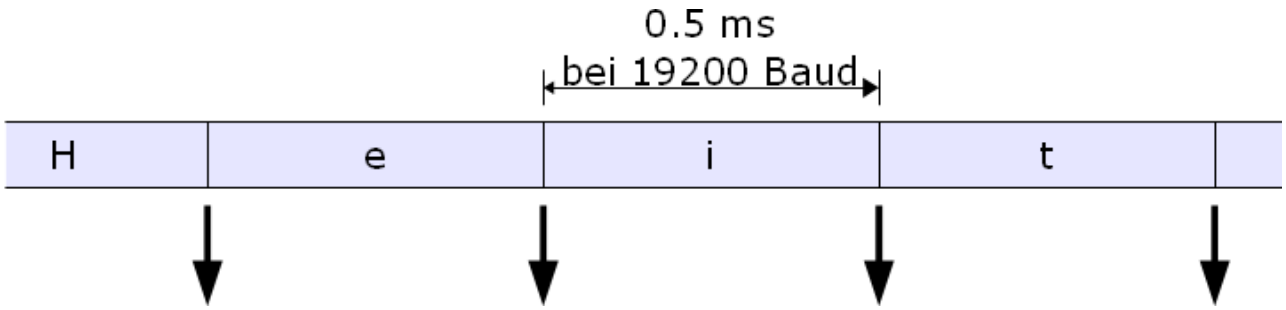


Interrupt-verarbeitung

Reaktion auf externes Ereignis

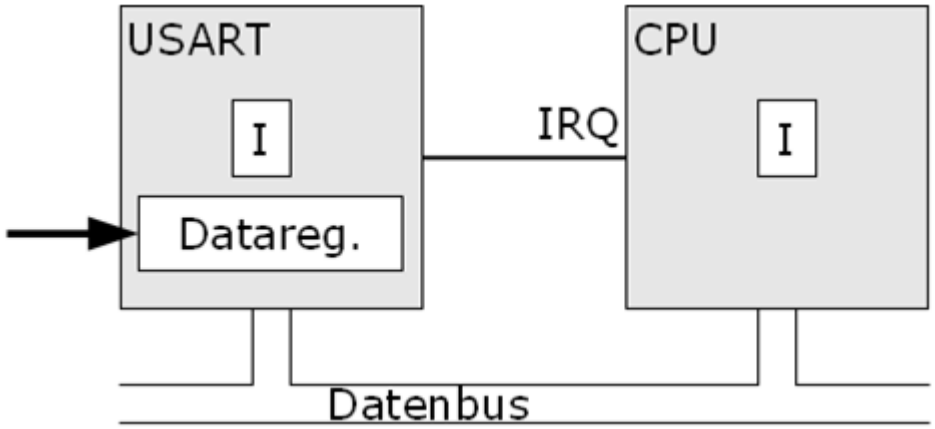
- **Regelmäßige Abfrage des Statusregisters z.B.: „Receiver Data Ready“ oder „Transmitter Empty“**
- **Falls nichts zu tun ist: Task wird beendet**
- **Andernfalls werden die Daten empfangen oder gesendet und verarbeitet.**
- **Typisch: Empfangsdaten kommen mit fester Geschwindigkeit**
  - **Erfordert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

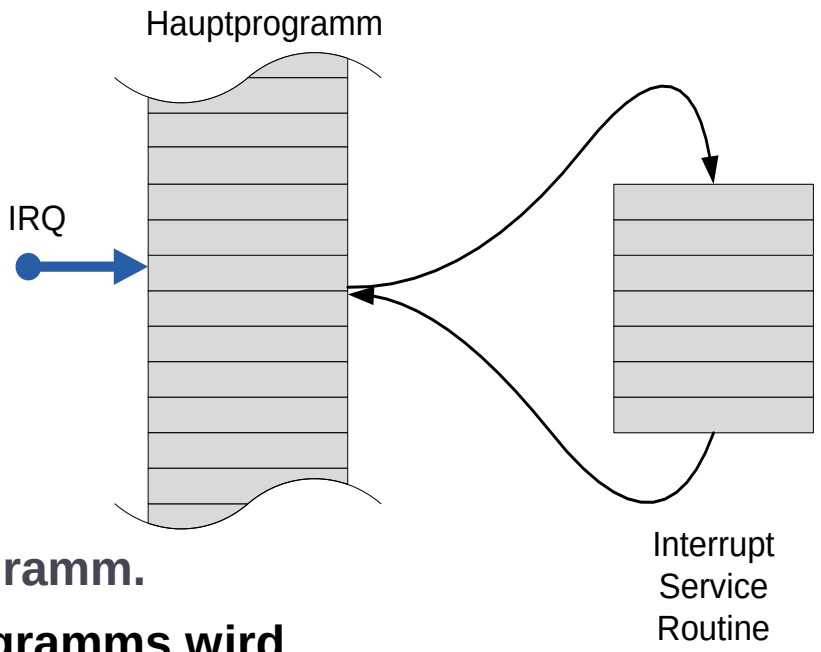




Interrupt-verarbeitung

Interruptverarbeitung

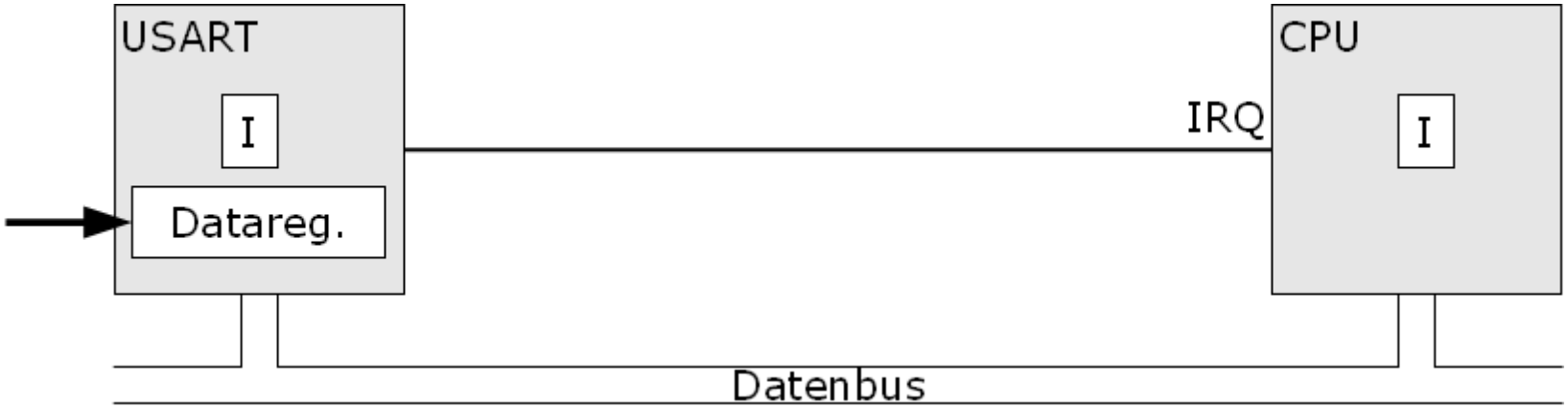


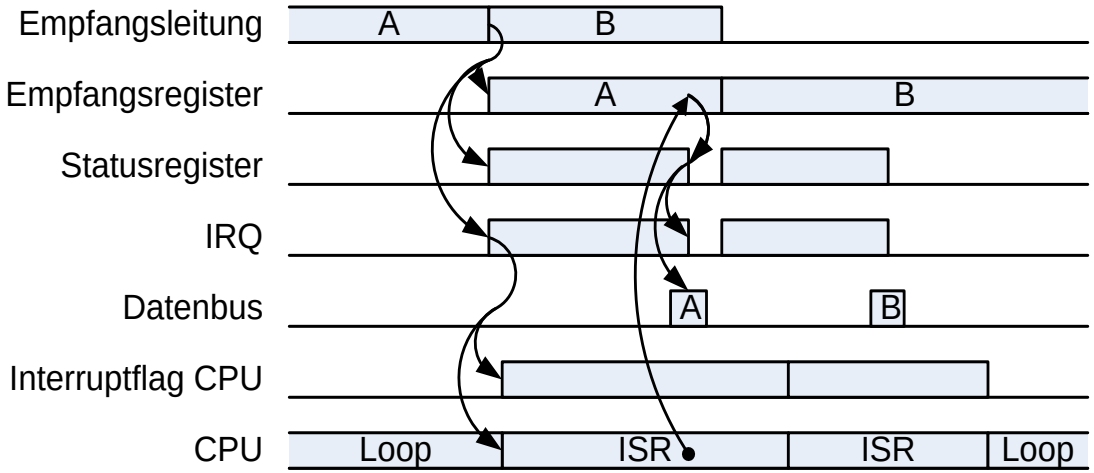


- **IRQ unterbricht laufendes Hauptprogramm.**
  - **Aktuelle Instruktion des Hauptprogramms wird vollständig abgearbeitet.**
  - **Zustand der CPU (Kontext) wird gerettet.**
  - **Abarbeitung der Interrupt Service Routine (ISR).**
  - **Alter Zustand der CPU wird wieder hergestellt.**
  - **Fortsetzung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

Interrupt-verarbeitung

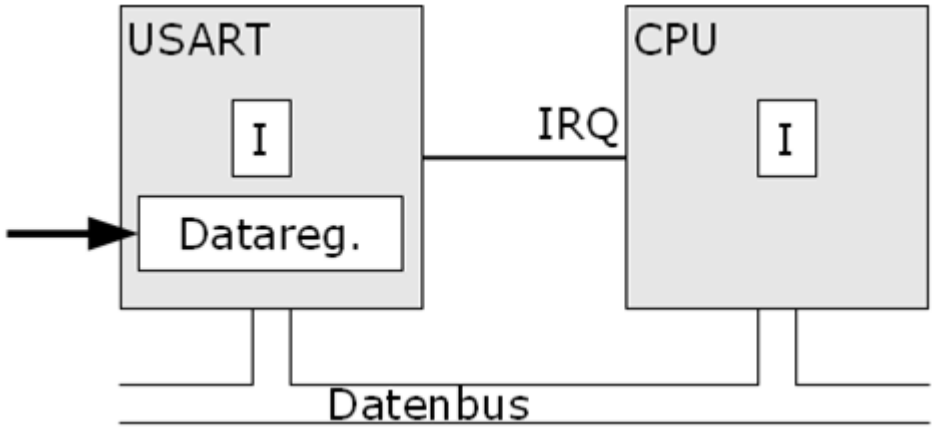
Einzelne Interruptquelle

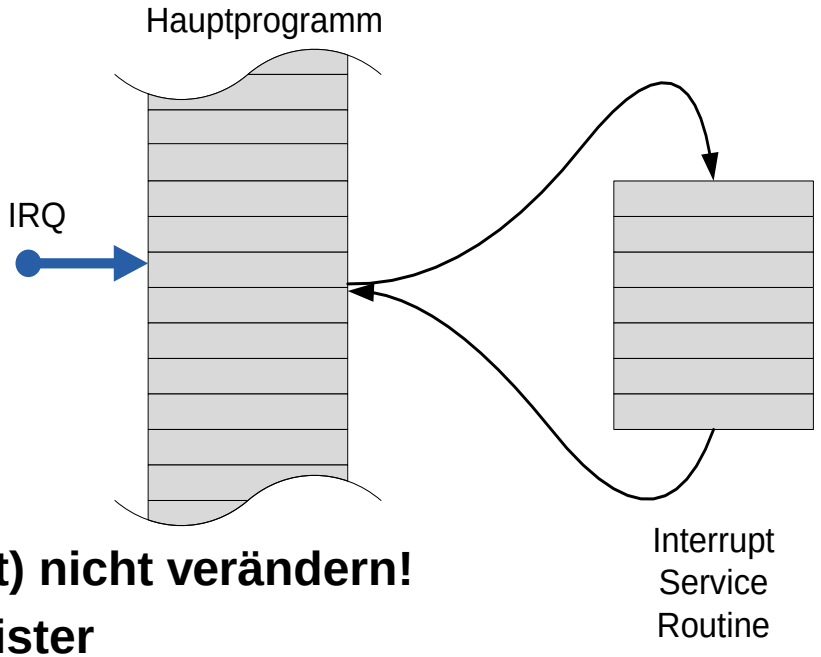




Interrupt-verarbeitung

Interruptverarbeitung



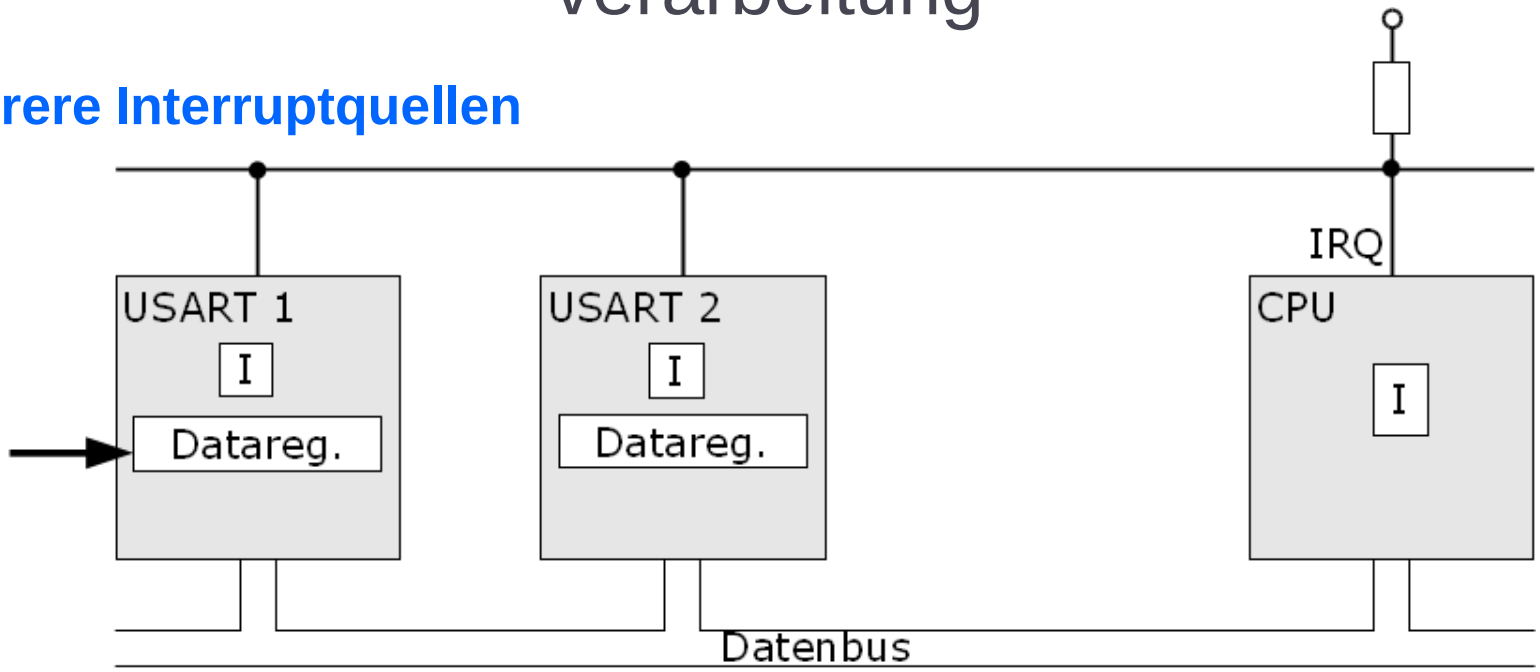


- **Wichtig:**
  - **ISR darf Zustand der CPU (Kontext) nicht verändern!**
  - **Alle von der ISR verwendeten Register**
    - **Statusregister, Datenregister**

    müssen zuvor gerettet und hinterher wieder hergestellt werden.
  - **Retten kann z.B. erfolgen:**
    - **auf dem Stack**
    - **in zusätzlichen Registersätzen.**

Interrupt-verarbeitung

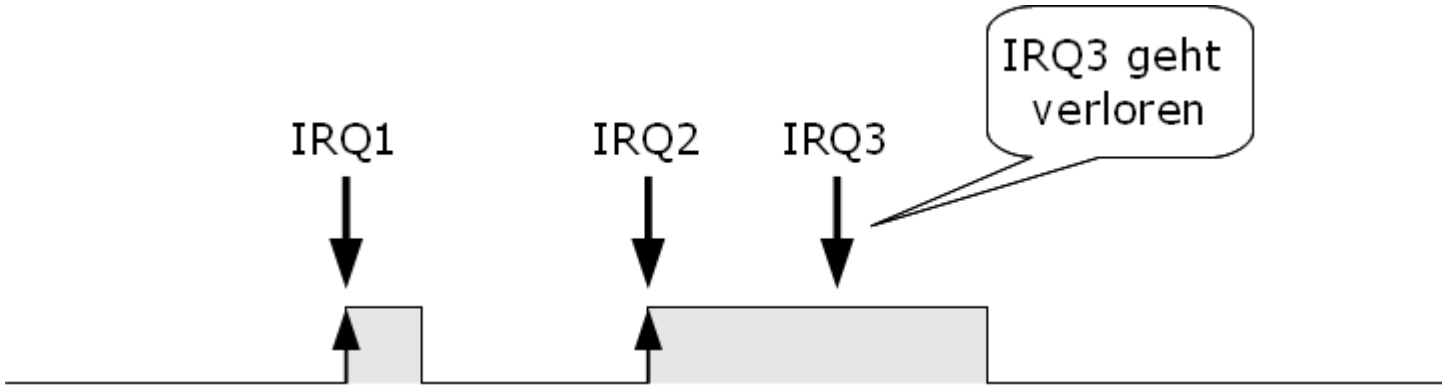
Mehrere Interruptquellen



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

Interrupt-verarbeitung

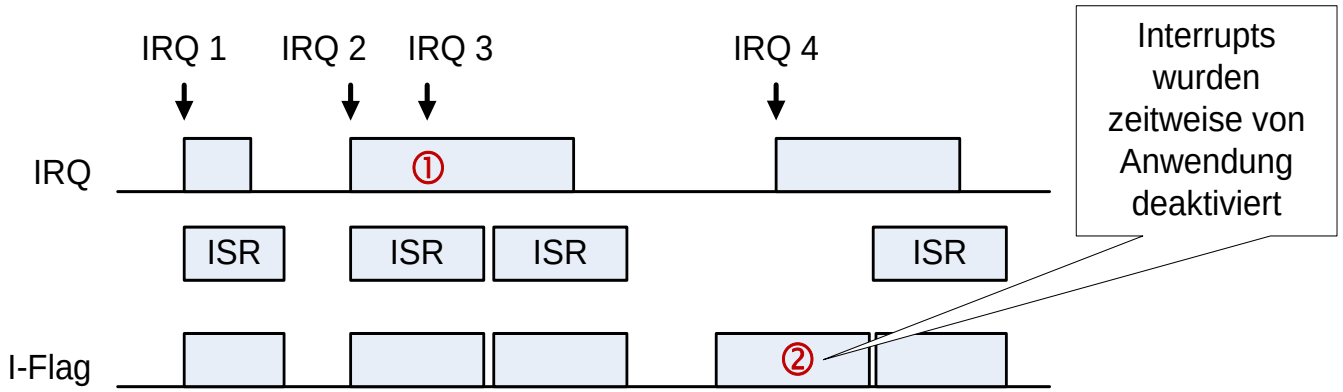
Flankensensitiver Interrupt



- **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)**

Interrupt-verarbeitung

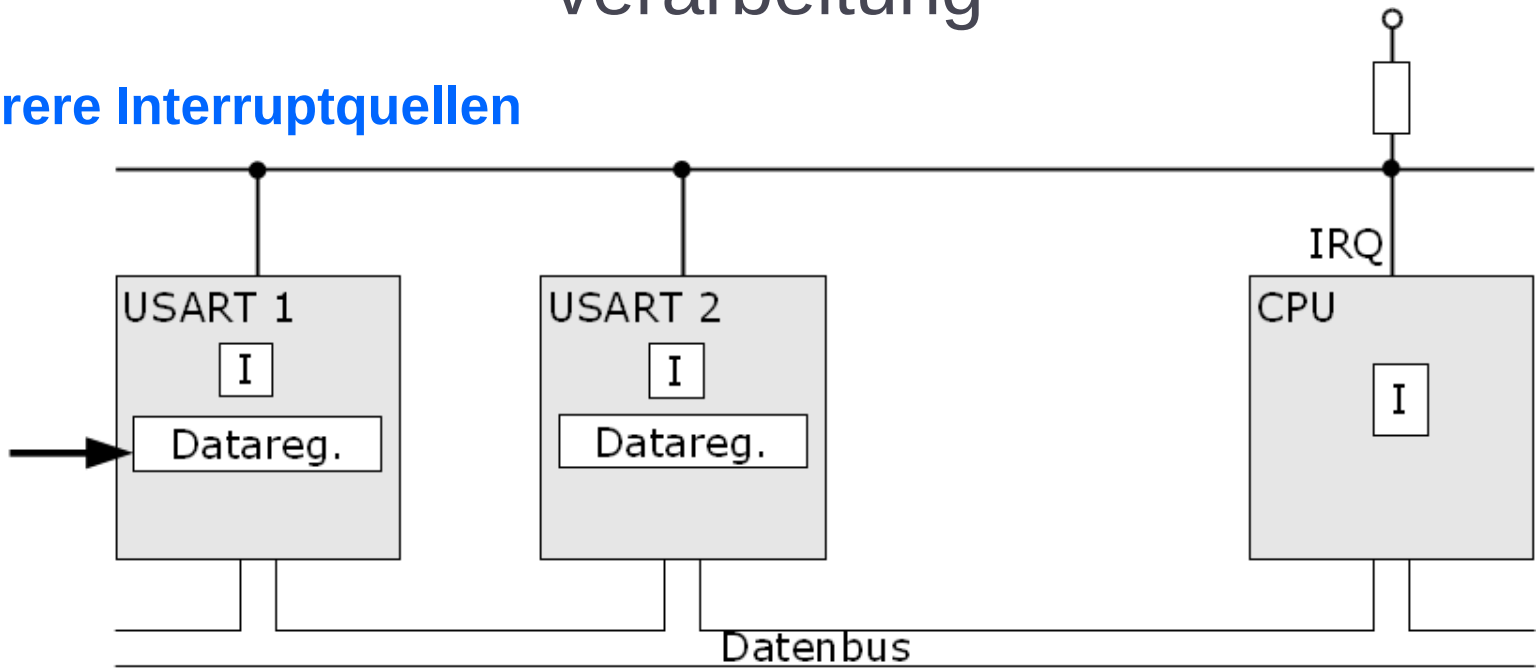
Pegelsensitiver Interrupt



- **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:**
  Interrupts bleiben erhalten, bis Interruptlogik wieder aktiv wird. ②

Interrupt-verarbeitung

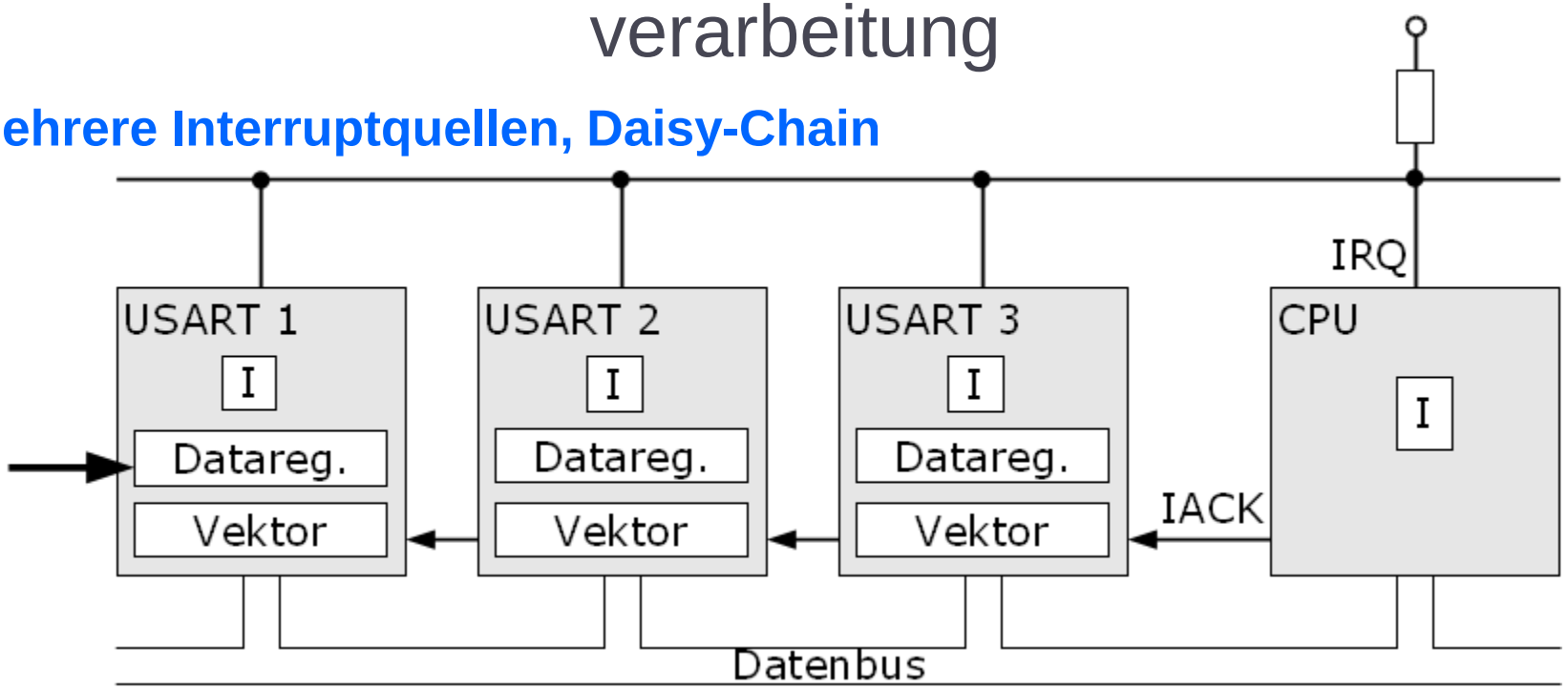
Mehrere Interruptquellen



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

Interrupt-verarbeitung

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.**

Interrupt-verarbeitung

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

Interrupt-verarbeitung

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 Anwendungspogramme
70	1C0-1C3	IRQ08: Echtzeituhr (nur AT)

Interrupt-verarbeitung

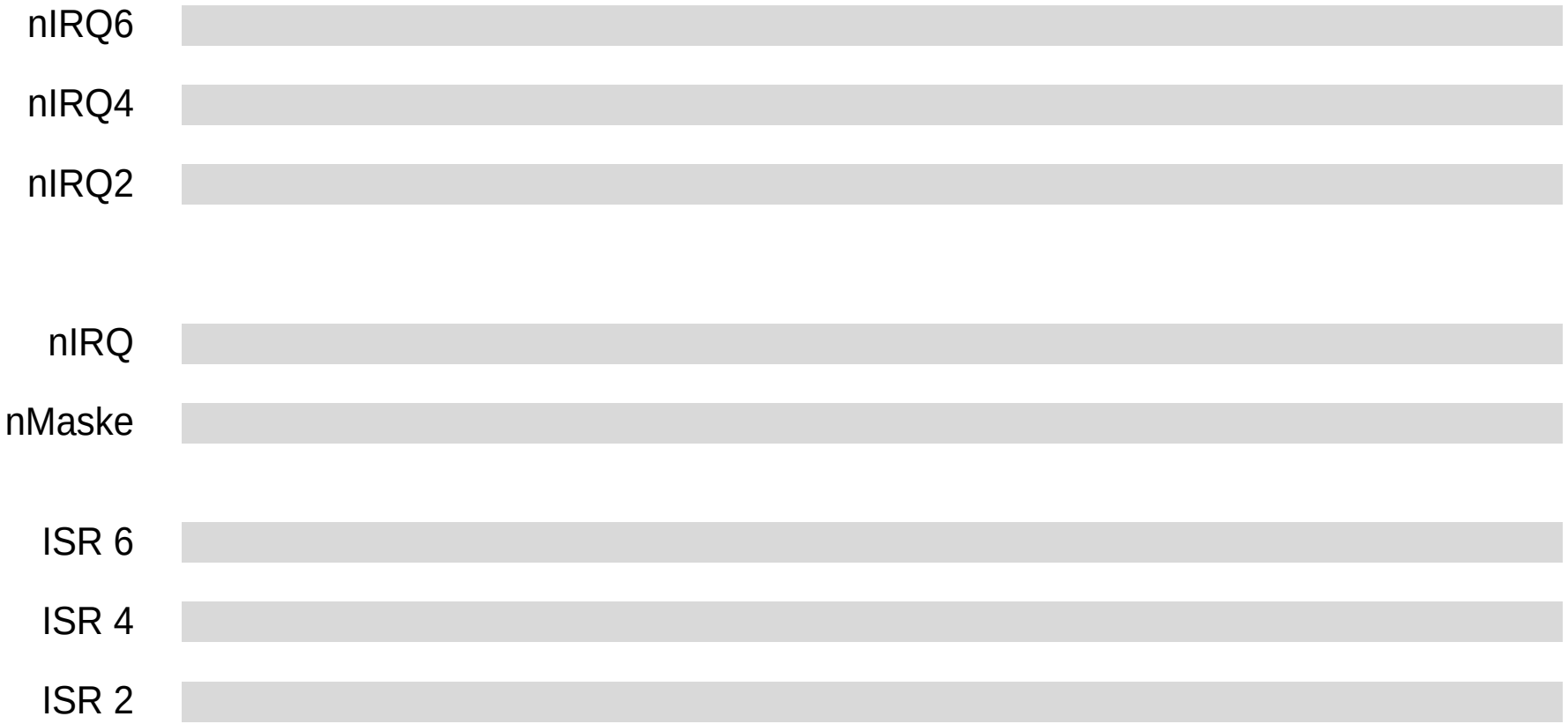
Beispiel: Vektortabelle AVR-Familie (8-Bit CPU)

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

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	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

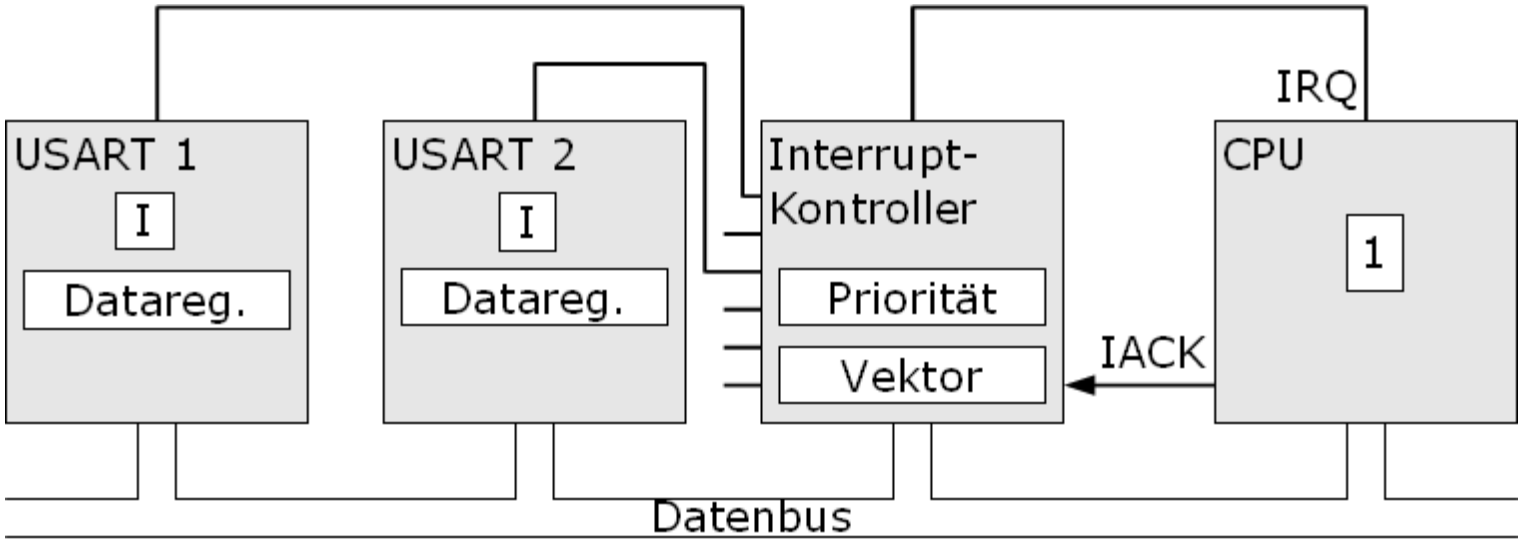
Interrupt-verarbeitung

Mehrere Interruptquellen



Interrupt-verarbeitung

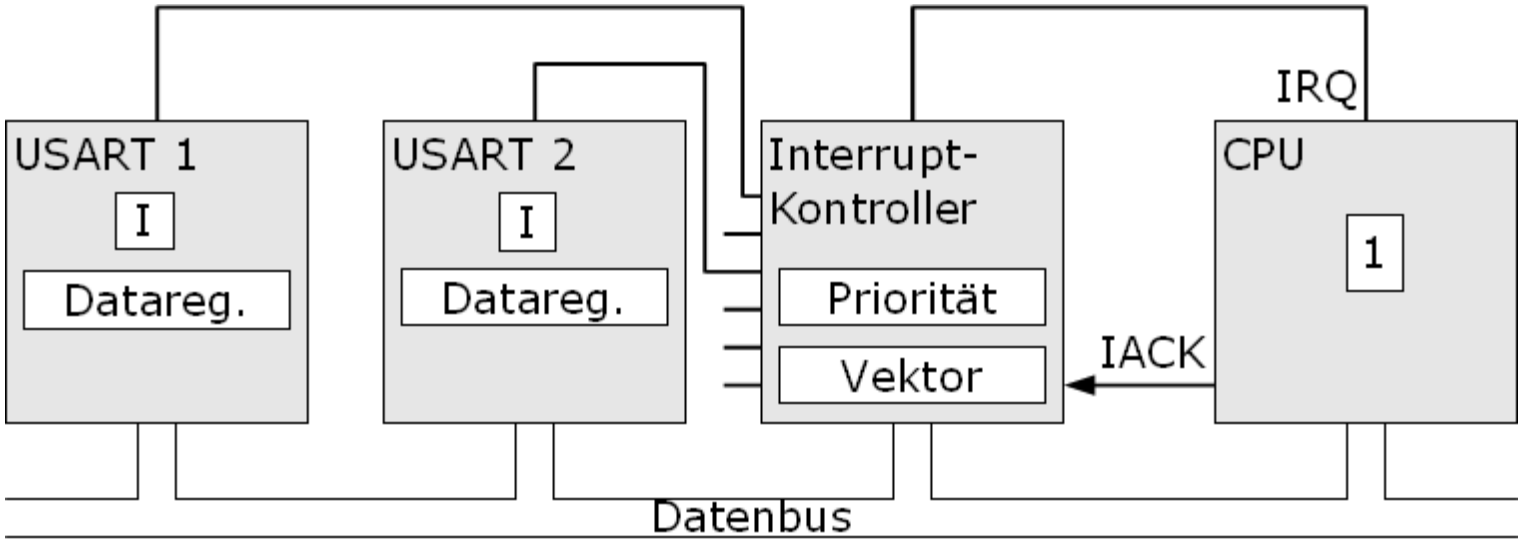
Mehrere Interruptquellen



- **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**

Interrupt-verarbeitung

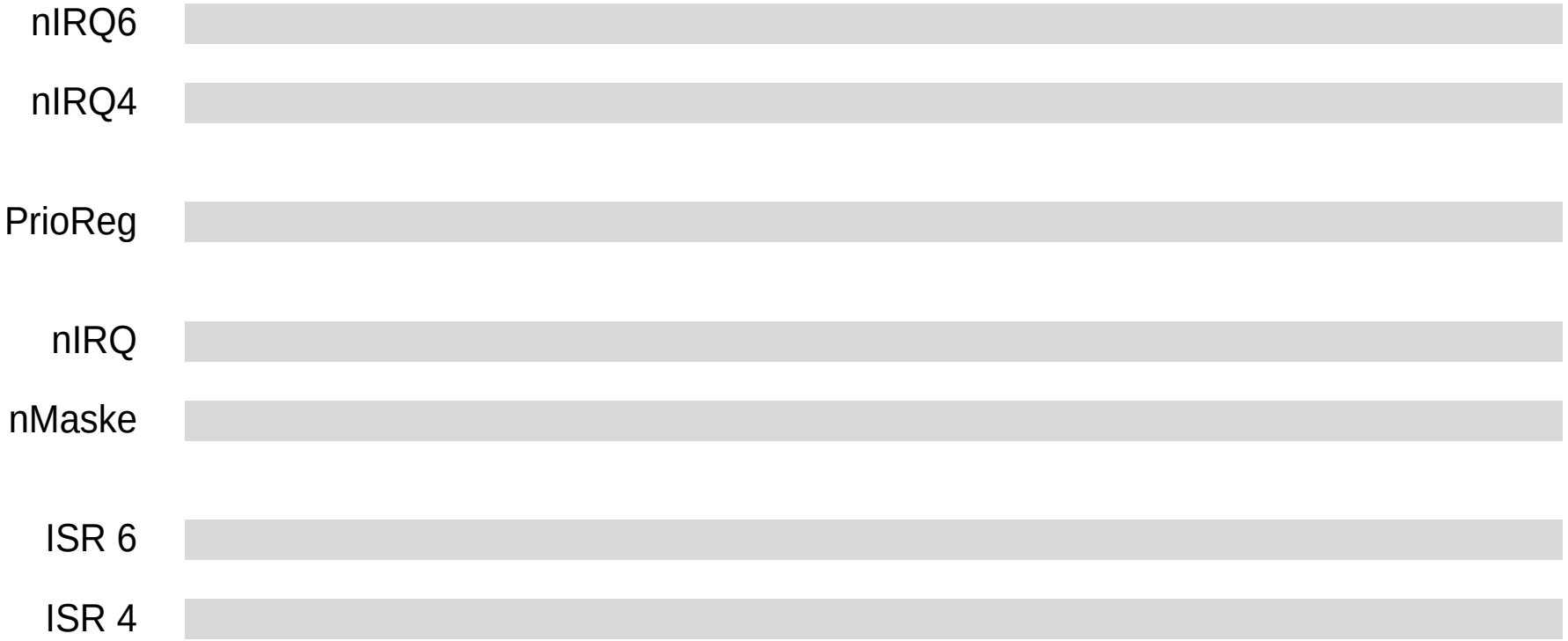
Mehrere Interruptquellen



- **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**

Interrupt-verarbeitung

Mehrere Interruptquellen



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 );
    }
    .....
```

Interrupt-verarbeitung

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
  - Hauptschleife

  auf 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**
  - **Hauptschleife**

  auf 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);
```

Interrupt-verarbeitung

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 inHauptschleife.**
      - **ISR ist in der Regel nicht unterbrechbar.**

Hauptschleife:

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

Interrupt-verarbeitung

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>
    }
    .....
}
```

Interrupt-verarbeitung

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);
```

Interrupt-verarbeitung

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.**