

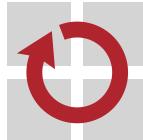
# Übungen zu Grundlagen der systemnahen Programmierung in C (GSPIC) im Sommersemester 2018

---

2018-05-29

Bernhard Heinloth

Lehrstuhl für Informatik 4  
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme  
und Betriebssysteme



FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
TECHNISCHE FAKULTÄT

## Interrupts

---

- Ablauf eines Interrupts (vgl. 15-7)
  - 0. Hardware setzt entsprechendes Flag
  - 1. Sind die Interrupts aktiviert und der Interrupt nicht maskiert, unterbricht der Interruptcontroller die aktuelle Ausführung
  - 2. weitere Interrupts werden deaktiviert
  - 3. aktuelle Position im Programm wird gesichert
  - 4. Adresse des Handlers wird aus Interrupt-Vektor gelesen und angesprungen
  - 5. Ausführung des Interrupt-Handlers
  - 6. am Ende des Handlers bewirkt ein Befehl "Return from Interrupt" die Fortsetzung des Anwendungsprogramms und die Reaktivierung der Interrupts

## Implementierung von Interruptbehandlungen

- Je Interrupt steht ein Bit zum Zwischenspeichern zur Verfügung
- Ursachen für den Verlust von weiteren Interrupts
  - Während einer Interruptbehandlung
  - Interruptsperren (zur Synchronisation von kritischen Abschnitten)
- Das Problem ist generell nicht zu verhindern
  - ↪ Risikominimierung: Interruptbehandlungen sollten möglichst kurz sein
    - Schleifen und Funktionsaufrufe vermeiden
    - Auf blockierende Funktionen verzichten (ADC/serielle Schnittstelle!)

- Timer
- Serielle Schnittstelle
- ADC (Analog-Digital-Umsetzer)
- Externe Interrupts durch Pegel(änderung) an bestimmten I/O-Pins
  - ⇒ ATmega328PB: 2 Quellen an den Pins PD2 (INT0) und PD3 (INT1)
    - Wahlweise pegel- oder flankengesteuert
    - Abhängig von der jeweiligen Interruptquelle
- Dokumentation im ATmega328PB-Datenblatt
  - Interruptbehandlung allgemein: S. 77-80
  - Externe Interrupts: S. 81-91

## (De-)Aktivieren von Interrupts

- Interrupts können durch die spezielle Maschinenbefehle aktiviert bzw. deaktiviert werden.
- Die Bibliothek avr-libc bietet hierfür Makros an:  
`#include <avr/interrupt.h>`
  - `sei()` (Set Interrupt Flag) - lässt ab dem nächsten Takt Interrupts zu
  - `cli()` (Clear Interrupt Flag) - blockiert (sofort) alle Interrupts
- Beim Betreten eines Interrupt-Handlers werden automatisch alle Interrupts blockiert, beim Verlassen werden sie wieder deblockiert
- `sei()` sollte niemals in einer Interruptbehandlung ausgeführt werden
  - potentiell endlos geschachtelte Interruptbehandlung
  - Stackoverflow möglich (Vorlesung, voraussichtlich Kapitel 17)
- Beim Start des µC sind die Interrupts abgeschaltet

# Konfigurieren von Interrupts

- Interrupt Sense Control (ISC) Bits befinden sich beim ATmega328PB im External Interrupt Control Register A (EICRA)
- Position der ISC-Bits im Register durch Makros definiert

Interrupt 0		Interrupt bei	Interrupt 1	
ISC01	ISC00		ISC11	ISC10
0	0	low Pegel	0	0
0	1	beliebiger Flanke	0	1
1	0	fallender Flanke	1	0
1	1	steigender Flanke	1	1

- Beispiel: INT1 bei ATmega328PB für fallende Flanke konfigurieren

```
01  /* die ISC-Bits befinden sich im EICRA */
02  EICRA &= ~(1<<ISC10); /* ISC10 löschen */
03  EICRA |= (1<<ISC11); /* ISC11 setzen */
```

5

# (De-)Maskieren von Interrupts

- Einzelne Interrupts können separat aktiviert (=demaskiert) werden
  - ATmega328PB: External Interrupt Mask Register (EIMSK)
- Die Bitpositionen in diesem Register sind durch Makros INTn definiert
- Ein gesetztes Bit aktiviert den jeweiligen Interrupt
- Beispiel: Interrupt 1 aktivieren

```
01  EIMSK |= (1<<INT1); /* demaskiere Interrupt 1 */
```

6

- Installieren eines Interrupt-Handlers wird durch C-Bibliothek unterstützt
- Makro ISR (Interrupt Service Routine) zur Definition einer Handler-Funktion (`#include <avr/interrupt.h>`)
- Parameter: gewünschten Vektor; z. B. INT1\_vect für externen Interrupt 1
  - verfügbare Vektoren: siehe avr-libc-Doku zu `avr/interrupt.h`  
⇒ verlinkt im Doku-Bereich auf der SPiC-Webseite
- Beispiel: Handler für Interrupt 1 implementieren

```
01 #include <avr/interrupt.h>
02 static uint16_t zaehler = 0;
03
04 ISR(INT1_vect){
05     zaehler++;
06 }
```

## Synchronisation

---

# Schlüsselwort volatile

- Bei einem Interrupt wird event = 1 gesetzt
- Aktive Warteschleife wartet, bis event != 0
- Der Compiler erkennt, dass event innerhalb der Warteschleife nicht verändert wird
  - ⇒ der Wert von event wird nur einmal vor der Warteschleife aus dem Speicher in ein Prozessorregister geladen
  - ⇒ Endlosschleife

```
01 static uint8_t event = 0;
02 ISR(INT0_vect) { event = 1; }
03
04 void main(void) {
05     while(1) {
06         while(event == 0) { /* warte auf Event */ }
07         /* bearbeite Event */
```

8

# Schlüsselwort volatile

- Bei einem Interrupt wird event = 1 gesetzt
- Aktive Warteschleife wartet, bis event != 0
- Der Compiler erkennt, dass event innerhalb der Warteschleife nicht verändert wird
  - ⇒ der Wert von event wird nur einmal vor der Warteschleife aus dem Speicher in ein Prozessorregister geladen
  - ⇒ Endlosschleife
- volatile erzwingt das Laden bei jedem Lesezugriff

```
01 static volatile uint8_t event = 0;
02 ISR(INT0_vect) { event = 1; }
03
04 void main(void) {
05     while(1) {
06         while(event == 0) { /* warte auf Event */ }
07         /* bearbeite Event */
```

8

- Fehlendes `volatile` kann zu unerwartetem Programmablauf führen
- Unnötige Verwendung von `volatile` unterbindet Optimierungen des Compilers
- Korrekte Verwendung von `volatile` ist Aufgabe des Programmierers!
  - ⇒ Verwendung von `volatile` so selten wie möglich, aber so oft wie nötig

## Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm

```
01 volatile uint8_t zaehler;
02
03 ; C-Anweisung: zaehler--;
04 lds r24, zaehler
05 dec r24
06 sts zaehler, r24
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++
08 lds r25, zaehler
09 inc r25
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		

# Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm

```
01 volatile uint8_t zaehler;  
02  
03 ; C-Anweisung: zaehler--;  
04 lds r24, zaehler  
05 dec r24  
06 sts zaehler, r24
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++  
08 lds r25, zaehler  
09 inc r25  
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
4	5	5	-

10

# Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm

```
01 volatile uint8_t zaehler;  
02  
03 ; C-Anweisung: zaehler--;  
04 lds r24, zaehler  
05 dec r24  
06 sts zaehler, r24
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++  
08 lds r25, zaehler  
09 inc r25  
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
4	5	5	-
5	5	4	-

10

# Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm

```
01 volatile uint8_t zaehler;
02
03 ; C-Anweisung: zaehler--;
04 lds r24, zaehler
05 dec r24
06 sts zaehler, r24
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++
08 lds r25, zaehler
09 inc r25
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
4	5	5	-
5	5	4	-
8	5	4	5

10

# Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm

```
01 volatile uint8_t zaehler;
02
03 ; C-Anweisung: zaehler--;
04 lds r24, zaehler
05 dec r24
06 sts zaehler, r24
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++
08 lds r25, zaehler
09 inc r25
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
4	5	5	-
5	5	4	-
8	5	4	5
9	5	4	6

10

# Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm

```
01 volatile uint8_t zaehler;
02
03 ; C-Anweisung: zaehler--;
04 lds r24, zaehler
05 dec r24
06 sts zaehler, r24
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++
08 lds r25, zaehler
09 inc r25
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
4	5	5	-
5	5	4	-
8	5	4	5
9	5	4	6
10	6	4	6

10

# Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm

```
01 volatile uint8_t zaehler;
02
03 ; C-Anweisung: zaehler--;
04 lds r24, zaehler
05 dec r24
06 sts zaehler, r24
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++
08 lds r25, zaehler
09 inc r25
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
4	5	5	-
5	5	4	-
8	5	4	5
9	5	4	6
10	6	4	6
6	4	4	-

10

# 16-Bit-Zugriffe (Read-Write)

## ■ Nebenläufige Nutzung von 16-Bit-Werten (Read-Write)

Hauptprogramm

```
01 volatile uint16_t zaehler;  
02  
03 ; C-Anweisung: z=zaehler;  
04 lds r22, zaehler  
05 lds r23, zaehler+1  
06 ; Verwendung von z
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++;  
08 lds r24, zaehler  
09 lds r25, zaehler+1  
10 adiw r24,1  
11 sts zaehler+1, r25  
12 sts zaehler, r24
```

Zeile	zaehler	zaehler (in r22 & r23)
-	0x00ff	

11

# 16-Bit-Zugriffe (Read-Write)

## ■ Nebenläufige Nutzung von 16-Bit-Werten (Read-Write)

Hauptprogramm

```
01 volatile uint16_t zaehler;  
02  
03 ; C-Anweisung: z=zaehler;  
04 lds r22, zaehler  
05 lds r23, zaehler+1  
06 ; Verwendung von z
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++;  
08 lds r24, zaehler  
09 lds r25, zaehler+1  
10 adiw r24,1  
11 sts zaehler+1, r25  
12 sts zaehler, r24
```

Zeile	zaehler	zaehler (in r22 & r23)
-	0x00ff	
4	0x00ff	0x??ff

11

# 16-Bit-Zugriffe (Read-Write)

## ■ Nebenläufige Nutzung von 16-Bit-Werten (Read-Write)

Hauptprogramm

```
01 volatile uint16_t zaehler;  
02  
03 ; C-Anweisung: z=zaehler;  
04 lds r22, zaehler  
05 lds r23, zaehler+1  
06 ; Verwendung von z
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++;  
08 lds r24, zaehler  
09 lds r25, zaehler+1  
10 adiw r24,1  
11 sts zaehler+1, r25  
12 sts zaehler, r24
```

Zeile	zaehler	zaehler (in r22 & r23)
-	0x00ff	
4	0x00ff	0x??ff
8 - 12	0x0100	0x??ff

11

# 16-Bit-Zugriffe (Read-Write)

## ■ Nebenläufige Nutzung von 16-Bit-Werten (Read-Write)

Hauptprogramm

```
01 volatile uint16_t zaehler;  
02  
03 ; C-Anweisung: z=zaehler;  
04 lds r22, zaehler  
05 lds r23, zaehler+1  
06 ; Verwendung von z
```

Interruptbehandlung

```
07 ; C-Anweisung: zaehler++;  
08 lds r24, zaehler  
09 lds r25, zaehler+1  
10 adiw r24,1  
11 sts zaehler+1, r25  
12 sts zaehler, r24
```

Zeile	zaehler	zaehler (in r22 & r23)
-	0x00ff	
4	0x00ff	0x??ff
8 - 12	0x0100	0x??ff
5 - 6	0x0100	0x01ff

⇒ Abweichung um 255 !

11

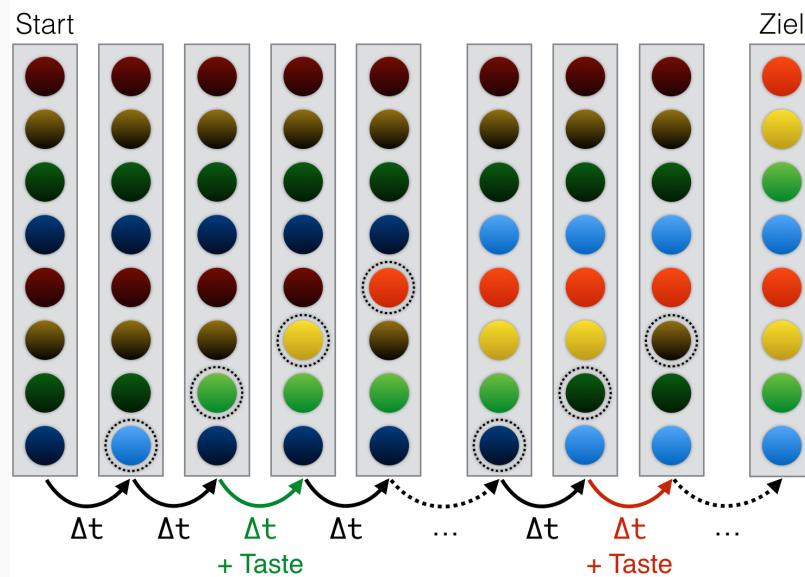
- Viele weitere Nebenläufigkeitsprobleme möglich
  - Nicht-atomare Modifikation von gemeinsamen Daten kann zu Inkonsistenzen führen
  - Problemanalyse durch den Anwendungsprogrammierer
  - Auswahl geeigneter Synchronisationsprimitive
- Lösung hier: Einseitiger Ausschluss durch Sperren der Interrupts
  - Sperrung aller Interrupts (`cli()`, `sei()`)
  - Maskieren einzelner Interrupts (EIMSK-Register)
- Problem: Interrupts während der Sperrung gehen evtl. verloren
  - Kritische Abschnitte sollten so kurz wie möglich gehalten werden

## Aufgabe: Geschicklichkeitsspiel

---

## Aufgabe: Geschicklichkeitsspiel (1)

- Spielcursor wandert dabei über LED-Reihe hin und her und invertiert (engl. toggle) den LED-Zustand
- LED-Zustand bleibt durch Drücken des Tasters erhalten
- Ziel: alle LEDs zum Leuchten bringen



13

## Aufgabe: Geschicklichkeitsspiel (2)

- Nach einem Level wird eine Siegessequenz auf den LEDs dargestellt

```
01 static void play(uint8_t level)
02 static void show_win(void);
03
04 void main(void) {
05     // Initialisierung
06     sei();
07     uint8_t level = 1;
08     while(1){
09         play(level);
10         show_win();
11         // Level aktualisieren
12     }
13 }
```

14

- Schwierigkeit (Geschwindigkeit) steigt mit jedem Level
- Schwierigkeit nähert sich einer maximalen Geschwindigkeit
- Asymptotische Annäherung:  $f(x) = \frac{c}{x}$  ( $c$  ist eine Konstante)

