

Übungen zu Systemnahe Programmierung in C (SPiC) – Wintersemester 2018/19

Übung 4

Rebecca Felsheim
Benedict Herzog

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Synchronisation



- 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 */
```



- 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 */
```



- 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



- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrucke
 - 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		



- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrucke
 - 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	-



- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrucke
 - 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	-



- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrucke
 - 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



- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrucke
 - 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



- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrucke
 - 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



- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrucke
 - 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	-



■ 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	



■ 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



■ 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



■ 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!



- 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

Stromsparmodi



- AVR-basierte Geräte oft batteriebetrieben (z.B. Fernbedienung)
- Energiesparen kann die Lebensdauer drastisch erhöhen
- AVR-Prozessoren unterstützen unterschiedliche Powersave-Modi
 - Deaktivierung funktionaler Einheiten
 - Unterschiede in der "Tiefe" des Schlafes
 - Nur aktive funktionale Einheiten können die CPU aufwecken
- Standard-Modus: Idle
 - CPU-Takt wird angehalten
 - Keine Zugriffe auf den Speicher
 - Hardware (Timer, externe Interrupts, ADC, etc.) sind weiter aktiv
- Dokumentation im ATmega328PB-Datenblatt, S. 58-66



- Unterstützung aus der avr-libc: (`#include <avr/sleep.h>`)
 - `sleep_enable()` - aktiviert den Sleep-Modus
 - `sleep_cpu()` - setzt das Gerät in den Sleep-Modus
 - `sleep_disable()` - deaktiviert den Sleep-Modus
 - `set_sleep_mode(uint8_t mode)` - stellt den zu verwendenden Modus ein
- Dokumentation von `avr/sleep.h` in avr-libc-Dokumentation
 - verlinkt im Doku-Bereich auf der SPiC-Webseite
- Beispiel

```
01 #include <avr/sleep.h>
02 set_sleep_mode(SLEEP_MODE_IDLE); /* Idle-Modus verwenden */
03 sleep_enable(); /* Sleep-Modus aktivieren */
04 sleep_cpu(); /* Sleep-Modus betreten */
05 sleep_disable(); /* Empfohlen: Sleep-Modus danach deaktivieren
   ↪ */
```



■ Dornröschenschlaf

⇒ **Problem:** Es kommt genau ein Interrupt

Hauptprogramm

```
01 sleep_enable();
02 event = 0;
03
04 while( !event ) {
05
06     sleep_cpu();
07
08 }
09
10 sleep_disable();
```

Interruptbehandlung

```
01 ISR(TIMER1_COMPA_vect) {
02     event = 1;
03 }
```



■ Dornröschenschlaf

⇒ **Problem:** Es kommt genau ein Interrupt

Hauptprogramm

```
01 sleep_enable();
02 event = 0;
03
04 while( !event ) {
05     ⚡ Interrupt ⚡
06     sleep_cpu();
07
08 }
09
10 sleep_disable();
```

Interruptbehandlung

```
01 ISR(TIMER1_COMPA_vect) {
02     event = 1;
03 }
```



■ Dornröschenschlaf

⇒ **Problem:** Es kommt genau ein Interrupt

⇒ **Lösung:** Interrupts während des kritischen Abschnitts sperren

Hauptprogramm

```
01 sleep_enable();
02 event = 0;
03 cli();
04 while( !event ) {
05     sei();
06     sleep_cpu();
07     cli();
08 }
09 sei();
10 sleep_disable();
```

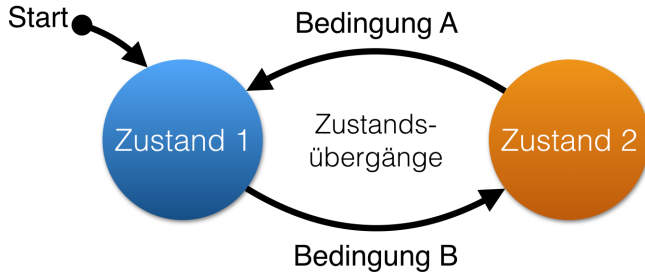
Interruptbehandlung

```
01 ISR(TIMER1_COMPA_vect) {
02     event = 1;
03 }
```

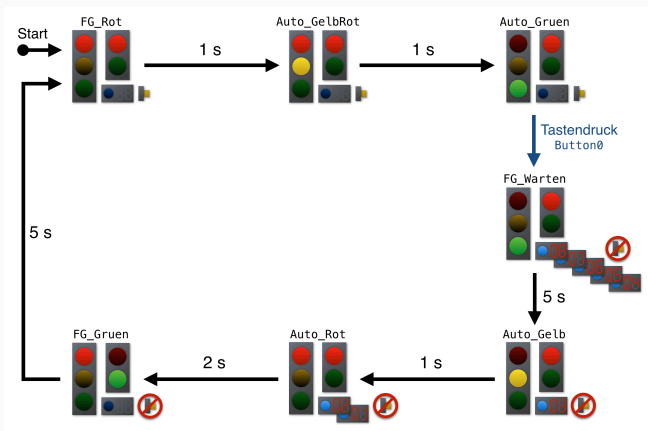
Aufgabe: Ampel



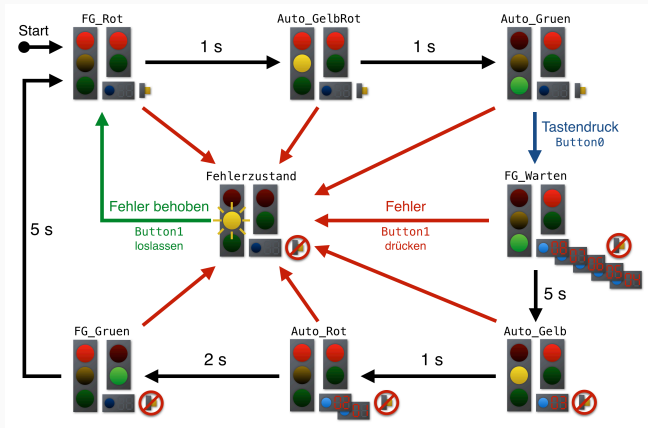
- Implementierung einer (Fußgänger-)Ampel mit Wartezeitanzeige
- Ablauf (exakt) nach Aufgabenbeschreibung (Referenzimplementierung verfügbar)
- Hinweise
 - Tastendrucke und Alarme als Events (kein aktives Warten)
 - In Sleep-Modus wechseln, wenn keine Events zu bearbeiten sind
 - nur eine Stelle zum Warten auf Events (sleep-Loop)
 - Deaktivieren des Tasters durch Ignorieren des Events (Änderung der Interrupt-Konfiguration ist nicht notwendig)
 - Abbildung auf Zustandsmaschine sinnvoll
 - Verwendung von `volatile` begründen



- **Zustände** mit bestimmten Eigenschaften; definierter Initialzustand
- **Zustandswechsel** in Abhängigkeit von definierten Bedingungen



Ampel als Zustandsmaschine





- Festlegung durch Zahlen ist fehleranfällig
 - Schwer zu merken
 - Wertebereich nur bedingt einschränkbar
- Besser enum:

```
01 enum state { STATE_RED, STATE_YELLOW, STATE_GREEN };  
02  
03 enum state my_state = STATE_RED;
```

- Mit typedef noch lesbarer:

```
01 typedef enum { STATE_RED, STATE_YELLOW, STATE_GREEN } state;  
02  
03 state my_state = STATE_RED;
```



```
01 switch ( my_state ) {
02 case STATE_RED:
03     ...
04     break;
05 case STATE_YELLOW:
06     ...
07     break;
08 case STATE_GREEN:
09     ...
10     break;
11 default:
12     // maybe invalid state
13     ...
14 }
```

- Vermeidung von if-else-Kaskaden
 - switch-Ausdruck muss eine Zahl sein (besser ein enum-Typ)
 - break-Anweisung nicht vergessen!
 - Ideal für die Abarbeitung von Systemen mit verschiedenen Zuständen
- ⇒ Implementierung von Zustandsmaschinen



- Nachbilden der Funktionalität aus dem `timer` Modul der `libpicboard`
- Regelmäßiger Alarm, um Zustandsübergänge zu realisieren
- Der ATmega328PB bietet 5 verschiedene Timer
- Für diese Aufgabe: Verwendung von `TIMER0` (8-bit Timer)
 - `TIMER0` kann Werte zwischen 0 und 255 annehmen
 - Wird automatisch, abhängig von der CPU Geschwindigkeit, erhöht
 - Kann benutzt werden, um Interrupts beim Erreichen bestimmter Werte oder bei Überläufen auszulösen
 - Für diese Aufgabe: Verwendung des Überlaufinterrupt (`OVF`)



- Geschwindigkeit durch `prescaler` einstellbar (Anzahl der CPU-Takte, bis der Zähler inkrementiert wird)
 - `prescaler` $\in \{1, 8, 64, 256, 1024\}$
 - Zum Beispiel:
 - 8-bit Timer mit Überlaufinterrupt
 - CPU Frequenz: 1 MHz
 - `prescaler`: 64
- alle $\frac{64}{1\text{MHz}} = 64 \mu\text{s}$ wird der Zähler erhöht
- alle $\frac{64 \cdot 256}{1\text{MHz}} = 16.4 \text{ ms}$ wird der Überlaufinterrupt ausgelöst
- CPU Geschwindigkeit des ATmega328PB: 16 MHz
 - Wie viele Interrupts müssen auftreten, bis 1 s vergangen ist?
 - Welcher `prescaler` ist am ressourcenschonendsten?



- Clock Select (CS) Bits befinden sich beim ATmega328PB im TC0 Control Register B (TCCR0B)
- (De-)aktivieren den TIMER0 und stellen die Geschwindigkeit ein

CS02	CS01	CS00	Beschreibung
0	0	0	Timer aus
0	0	1	prescaler 1
0	1	0	prescaler 8
0	1	1	prescaler 64
1	0	0	prescaler 256
1	0	1	prescaler 1024
1	1	0	Ext. Takt (fallende Flanke)
1	1	1	Ext. Takt (steigende Flanke)



- TIMER0 Overflow Interrupt Enable (TOIE0) Bit befindet sich beim ATmega328PB im TC0 Interrupt Mask Register (TIMSK0)
- (De-)aktivieren des Überlaufinterrupts

```
01 ISR(TIMER0_OVF_vect) {
02     // [...]
03 }
04
05 void init(void) {
06     // Timer mit prescaler 64 aktivieren
07     TCCR0B |= (1 << CS01) | (1 << CS00);
08     TCCR0B &= ~(1 << CS02);
09
10     // Überlaufunterbrechung aktivieren
11     TIMSK0 |= (1 << TOIE0);
12
13     // [...]
14 }
```

Hands-on: Laufschrift



- Funktionsweise:
Schrittweises Anzeigen eines Textes auf der 7-Segment-Anzeige
- Lernziele:
 - Alarme & Schlafenlegen
 - Zeiger & Zeigerarithmetik
 - Zeichenfolgen in C
- Vorgehen:
 - Aktivieren eines wiederkehrenden Alarms
(`sb_timer_setAlarm()`)
 - Zusammensetzen des aktuellen Teilstrings
 - Ausgabe über 7-Segment-Anzeige
 - In Wartephasen Mikrocontroller in den Energiesparmodus versetzen



"hallo" \equiv

- Repräsentation von Strings
 - Zeiger auf erstes Zeichen der Zeichenfolge
 - null-terminiert: Null-Zeichen kennzeichnet Ende
- ⇒ `strlen(s)` != Speicherbedarf



■ const uint8_t*

- ein Pointer auf einen uint8_t-Wert, der konstant ist
- Wert nicht über den Pointer veränderbar

■ uint8_t* const

- ein **konstanter Pointer** auf einen (beliebigen) uint8_t-Wert
- Pointer darf nicht mehr auf eine andere Speicheradresse zeigen

