

Übungen zu Systemnahe Programmierung in C (SPiC)

Sebastian Maier
(Lehrstuhl Informatik 4)

Übung 4



Wintersemester 2016/2017



Inhalt

Wiederholung: Synchronisation

Wiederholung: Stromsparmodi

Aufgabe 4: Ampel

Hands-on: Zeiger, Felder, Strukturen & Zeichenfolgen



Inhalt

Wiederholung: Synchronisation

Schlüsselwort volatile

Lost Update

16-Bit-Zugriffe (Read-Write)

Sperren von Interrupts

Wiederholung: Stromsparmodi

Aufgabe 4: Ampel

Hands-on: Zeiger, Felder, Strukturen & Zeichenfolgen



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

```
1 static uint8_t event = 0;
2 ISR (INT0_vect) { event = 1; }
3
4 void main(void) {
5     while(1) {
6         while(event == 0) { /* warte auf Event */
7             /* bearbeite Event */
```



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

```
1 volatile static uint8_t event = 0;
2 ISR (INT0_vect) { event = 1; }
3
4 void main(void) {
5     while(1) {
6         while(event == 0) { /* warte auf Event */ }
7         /* bearbeite Event */
8     }
9 }
```



Verwendung von volatile

- 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 Tastendrucke
 - Inkrementierung in der Unterbrechungsbehandlung
 - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

Hauptprogramm

```
1 ; volatile uint8_t zaehler;
2 ; C-Anweisung: zaehler--;
3 lds r24, zaehler
4 dec r24
5 sts zaehler, r24
```

Interruptbehandlung

```
7 ; C-Anweisung: zaehler++
8 lds r25, zaehler
9 inc r25
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		



Lost Update

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

Hauptprogramm

```
1 ; volatile uint8_t zaehler;
2 ; C-Anweisung: zaehler--;
3 lds r24, zaehler
4 dec r24
5 sts zaehler, r24
```

Interruptbehandlung

```
7 ; C-Anweisung: zaehler++
8 lds r25, zaehler
9 inc r25
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3	5	5	-



Lost Update

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

Hauptprogramm

```
1 ; volatile uint8_t zaehler;  
2 ; C-Anweisung: zaehler--;  
3 lds r24, zaehler  
4 dec r24  
5 sts zaehler, r24
```

Interruptbehandlung

```
7 ; C-Anweisung: zaehler++  
8 lds r25, zaehler  
9 inc r25  
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3	5	5	-
4	5	4	-



Lost Update

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

Hauptprogramm

```
1 ; volatile uint8_t zaehler;  
2 ; C-Anweisung: zaehler--;  
3 lds r24, zaehler  
4 dec r24  
5 sts zaehler, r24
```

Interruptbehandlung

```
7 ; C-Anweisung: zaehler++  
8 lds r25, zaehler  
9 inc r25  
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3	5	5	-
4	5	4	-
8	5	4	5



Lost Update

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

Hauptprogramm

```
1 ; volatile uint8_t zaehler;  
2 ; C-Anweisung: zaehler--;  
3 lds r24, zaehler  
4 dec r24  
5 sts zaehler, r24
```

Interruptbehandlung

```
7 ; C-Anweisung: zaehler++  
8 lds r25, zaehler  
9 inc r25  
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3	5	5	-
4	5	4	-
8	5	4	5
9	5	4	6



Lost Update

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

Hauptprogramm

```
1 ; volatile uint8_t zaehler;  
2 ; C-Anweisung: zaehler--;  
3 lds r24, zaehler  
4 dec r24  
5 sts zaehler, r24
```

Interruptbehandlung

```
7 ; C-Anweisung: zaehler++  
8 lds r25, zaehler  
9 inc r25  
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3	5	5	-
4	5	4	-
8	5	4	5
9	5	4	6
10	6	4	6



Lost Update

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

Hauptprogramm

```
1 ; volatile uint8_t zaehler;  
2 ; C-Anweisung: zaehler--;  
3 lds r24, zaehler  
4 dec r24  
5 sts zaehler, r24
```

Interruptbehandlung

```
7 ; C-Anweisung: zaehler++  
8 lds r25, zaehler  
9 inc r25  
10 sts zaehler, r25
```

Zeile	zaehler	r24	r25
-	5		
3	5	5	-
4	5	4	-
8	5	4	5
9	5	4	6
10	6	4	6
5	4	4	-

16-Bit-Zugriffe (Read-Write)

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

Hauptprogramm

```
1 volatile uint16_t zaehler;  
2  
3 ; C-Anweisung: z=zaehler;  
4 lds r22, zaehler  
5 lds r23, zaehler+1  
6 ; Verwendung von z
```

Interruptbehandlung

```
8 ; C-Anweisung: zaehler++  
9 lds r24, zaehler  
10 lds r25, zaehler+1  
11 adiw r24,1  
12 sts zaehler+1, r25  
13 sts zaehler, r24
```

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

16-Bit-Zugriffe (Read-Write)

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

Hauptprogramm

```
1 volatile uint16_t zaehler;  
2  
3 ; C-Anweisung: z=zaehler;  
4 lds r22, zaehler  
5 lds r23, zaehler+1  
6 ; Verwendung von z
```

Interruptbehandlung

```
8 ; C-Anweisung: zaehler++  
9 lds r24, zaehler  
10 lds r25, zaehler+1  
11 adiw r24,1  
12 sts zaehler+1, r25  
13 sts zaehler, r24
```

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

16-Bit-Zugriffe (Read-Write)

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

Hauptprogramm

```
1 volatile uint16_t zaehler;  
2  
3 ; C-Anweisung: z=zaehler;  
4 lds r22, zaehler  
5 lds r23, zaehler+1  
6 ; Verwendung von z
```

Interruptbehandlung

```
8 ; C-Anweisung: zaehler++  
9 lds r24, zaehler  
10 lds r25, zaehler+1  
11 adiw r24,1  
12 sts zaehler+1, r25  
13 sts zaehler, r24
```

Zeile	zaehler	zaehler (in r22 & r23)
-	0x00ff	
4	0x00ff	0x??ff
9 - 13	0x0100	0x??ff

16-Bit-Zugriffe (Read-Write)

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

Hauptprogramm

```
1 volatile uint16_t zaehler;  
2  
3 ; C-Anweisung: z=zaehler;  
4 lds r22, zaehler  
5 lds r23, zaehler+1  
6 ; Verwendung von z
```

Interruptbehandlung

```
8 ; C-Anweisung: zaehler++  
9 lds r24, zaehler  
10 lds r25, zaehler+1  
11 adiw r24,1  
12 sts zaehler+1, r25  
13 sts zaehler, r24
```

Zeile	zaehler	zaehler (in r22 & r23)
-	0x00ff	
4	0x00ff	0x??ff
9 - 13	0x0100	0x??ff
5 - 6	0x0100	0x01ff

⇒ Abweichung um 255!



Sperren der Unterbrechungsbehandlung beim AVR

- 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 (GICR-Register)
- Problem: Interrupts während der Sperrung gehen evtl. verloren
 - Kritische Abschnitte sollten so kurz wie möglich gehalten werden



Inhalt

Wiederholung: Synchronisation

Wiederholung: Stromsparmodi
Nutzung der Sleep-Modi
Lost Wakeup

Aufgabe 4: Ampel

Hands-on: Zeiger, Felder, Strukturen & Zeichenfolgen



Stromsparmodi von AVR-Prozessoren

- 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 ATmega32-Datenblatt, S. 33-37



Nutzung der Sleep-Modi

- 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

```
1 #include <avr/sleep.h>
2 set_sleep_mode(SLEEP_MODE_IDLE); /* Idle-Modus verwenden */
3 sleep_enable(); /* Sleep-Modus aktivieren */
4 sleep_cpu(); /* Sleep-Modus betreten */
5 sleep_disable(); /* Empfohlen: Sleep-Modus danach deaktivieren */
```



Lost Wakeup

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

Hauptprogramm

```
1 sleep_enable();
2 event = 0;
3
4 while( !event ) {
5
6     sleep_cpu();
7
8 }
9
10 sleep_disable();
```

Interruptbehandlung

```
11 ISR(TIMER1_COMPA_vect) {
12     event = 1;
13 }
```



Lost Wakeup

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

Hauptprogramm

```
1 sleep_enable();
2 event = 0;
3
4 while( !event ) {
5     ⚡ Interrupt ⚡
6     sleep_cpu();
7
8 }
9
10 sleep_disable();
```

Interruptbehandlung

```
11 ISR(TIMER1_COMPA_vect) {
12     event = 1;
13 }
```



Lost Wakeup

- Dornröschenschlaf
 - ⇒ **Problem:** Es kommt genau ein Interrupt
 - ⇒ **Lösung:** Interrupts während des kritischen Abschnitts sperren

Hauptprogramm

```
1 sleep_enable();
2 event = 0;
3 cli();
4 while( !event ) {
5     sei();
6     sleep_cpu();
7     cli();
8 }
9 sei();
10 sleep_disable();
```

Interruptbehandlung

```
11 ISR(TIMER1_COMPA_vect) {
12     event = 1;
13 }
```



Wiederholung: Synchronisation

Wiederholung: Stromsparmodi

Aufgabe 4: Ampel

Aufgabenbeschreibung

Ampel als Zustandsmaschine

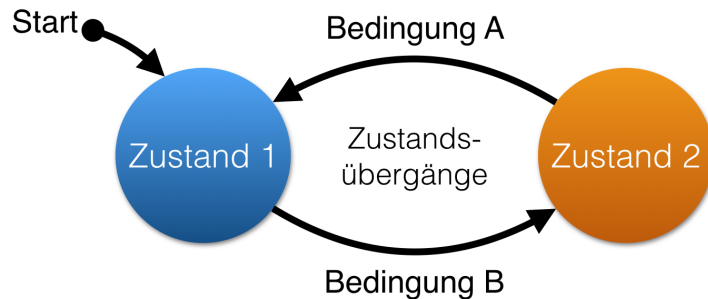
Hands-on: Zeiger, Felder, Strukturen & Zeichenfolgen



- Implementierung einer (Fußgänger-)Ampel mit Wartezeitanzeige
- Ablauf (exakt) nach Aufgabenbeschreibung (Musterlösung verfügbar)
- Hinweise
 - Tastendrucke und Alarmer 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



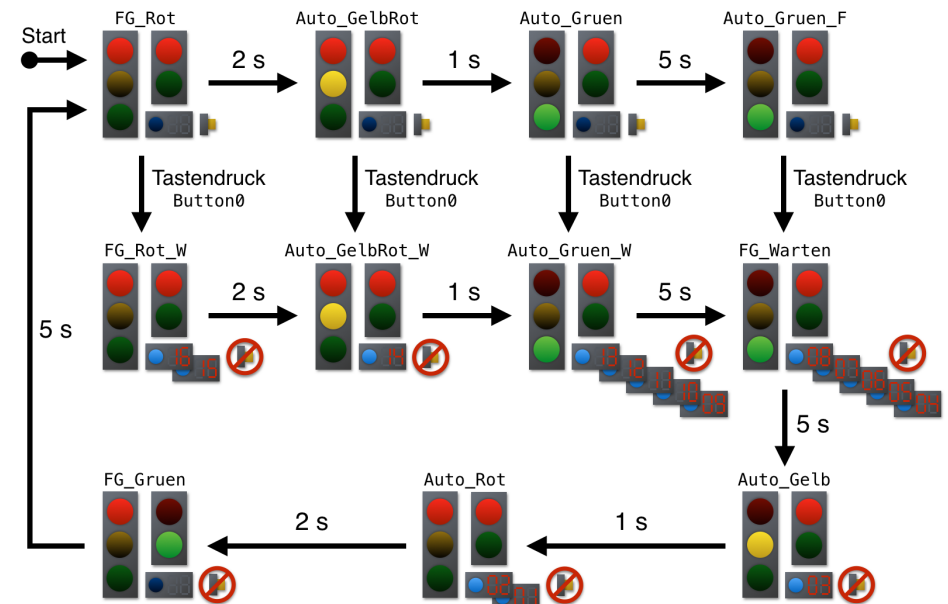
Zustandsmaschinen



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



Ampel als Zustandsmaschine



Festlegen von Zuständen: enum-Typen

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

- Besser enum:

```
1 enum state { STATE_RED, STATE_YELLOW, STATE_GREEN };
2
3 enum state my_state = STATE_RED;
```

- Mit typedef noch lesbarer:

```
1 typedef enum { STATE_RED, STATE_YELLOW, STATE_GREEN } state;
2
3 state my_state = STATE_RED;
```



Zustandsabfragen: switch-case-Anweisung

```
1 switch ( my_state ) {
2 case STATE_RED:
3     ...
4     break;
5 case STATE_YELLOW:
6     ...
7     break;
8 case STATE_GREEN:
9     ...
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



Inhalt

Wiederholung: Synchronisation

Wiederholung: Stromsparmodi

Aufgabe 4: Ampel

Hands-on: Zeiger, Felder, Strukturen & Zeichenfolgen

Hands-on: Laufschrift

Hands-on: Strukturen, Felder & Zeiger



Hands-on: Laufschrift

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

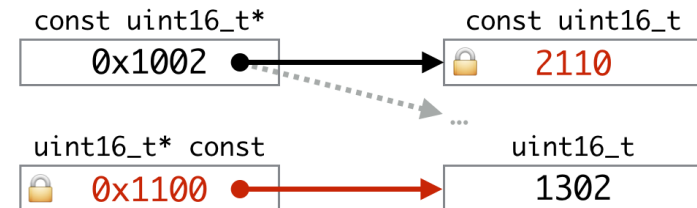


"hallo" $\equiv$ 

- 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



Hands-on: Strukturen, Felder & Zeiger

- `struct` für GPS-Koordinaten
- Feld von GPS-Koordinaten
- Call-by-Value vs. Call-by-Reference
- Zeigerarithmetik
- Funktionszeiger

