

# Übungen zu Systemnahe Programmierung in C (SPiC) – Sommersemester 2022

---

## Übung 6

Tim Rheinfels  
Phillip Raffeck  
Maximilian Ott

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



Lehrstuhl für Verteilte Systeme  
und Betriebssysteme



FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

## **Vorstellung Aufgabe 3**

---

# AVR Timer

---



- Häufige Aufgaben in der Mikrocontrollerprogrammierung:
  - Regelmäßige Aktualisierung der Ausgabe (z.B. Bildwiederholrate)
  - Regelmäßiges Auslesen eines Wertes (z.B. serielle Konsole)
  - Pulseweitenmodulation (PWM)
  - Passives Warten
  - ...

⇒ Timer helfen bei der Umsetzung



- Ein Timer modifiziert pro Timertakt seinen Zähler
    - Inkrement (default)
    - Dekrement
  - Bei vorher konfigurierten Ereignissen wird ein Interrupt ausgelöst
    - Zähler erreicht einen bestimmten Wert
    - Zähler läuft über
    - (externes Ereignis tritt auf)
  - Der ATmega328PB bietet 5 verschiedene Timer:
    - `TIMER{0, 2}`: 8-bit Zähler
    - `TIMER{1, 3, 4}`: 16-bit Zähler
- ⇒ Für die Übungsaufgaben: `TIMER0`
- ⇒ In der libspicboard: `TIMER{1, 2, 4}`



## ■ Wie schnell läuft der Timer:

- TCCR0B: TC0 Control Register B
- CSxx: Clock Select Bits
- Prescaler: Anzahl der CPU-Takte bis Zähler inkrementiert wird
- Was passiert, wenn die CPU in den Schlafmodus geht?

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)



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)

```
01 static void init(void) {  
02     // Timer mit prescaler 64 aktivieren  
03     TCCR0B |= (1 << CS01) | (1 << CS00);  
04     TCCR0B &= ~(1 << CS02);  
05  
06     // [...]  
07 }
```



- **Wann löst der Timer einen Interrupt aus:**
  - **Overflow:** Wenn der Zähler überläuft
  - **Match:** Wenn der Zähler einen bestimmten Wert erreicht
    - ⇒ Register OCR0A (TIMER0 Output Compare Register A)
    - ⇒ Register OCR0B (TIMER0 Output Compare Register B)
  - Interrupts einzeln demaskierbar
  - TIMSK0: TIMER0 Interrupt Mask Register

Bit	ISR	Beschreibung
TOIE0	TIMER0_OVF_vect	TIMER0 Overflow (Interrupt Enable)
OCIE0A	TIMER0_COMPA_vect	TIMER0 Output Compare A (...)
OCIE0B	TIMER0_COMPB_vect	TIMER0 Output Compare B (...)





- **Wann löst der Timer einen Interrupt aus:**
  - **Overflow:** Wenn der Zähler überläuft
  - **Match:** Wenn der Zähler einen bestimmten Wert erreicht
    - ⇒ Register OCR0A (TIMER0 Output Compare Register A)
    - ⇒ Register OCR0B (TIMER0 Output Compare Register B)
  - Interrupts einzeln demaskierbar
  - TIMSK0: TIMER0 Interrupt Mask Register

Bit	ISR	Beschreibung
TOIE0	TIMER0_OVF_vect	TIMER0 Overflow (Interrupt Enable)
OCIE0A	TIMER0_COMPA_vect	TIMER0 Output Compare A (...)
OCIE0B	TIMER0_COMPB_vect	TIMER0 Output Compare B (...)



Bit	ISR	Beschreibung
TOIE0	TIMER0_OVF_vect	TIMER0 Overflow (Interrupt Enable)
OCIE0A	TIMER0_COMPA_vect	TIMER0 Output Compare A (...)
OCIE0B	TIMER0_COMPB_vect	TIMER0 Output Compare B (...)

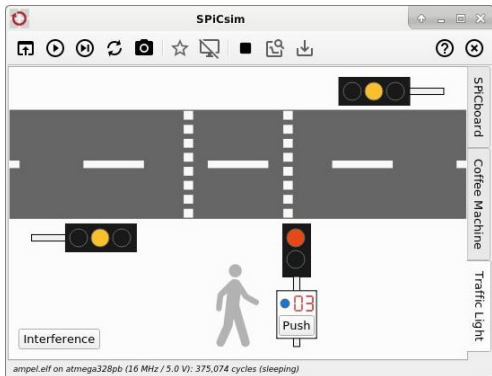
```
01 ISR(TIMER0_OVF_vect) {
02     // [...]
03 }
04
05 static void init(void) {
06     // Überlaufunterbrechung aktivieren
07     TIMSK0 |= (1 << TOIE0);
08
09     // [...]
10 }
```



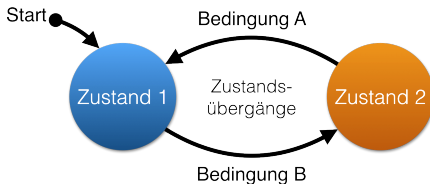
- Zur Erinnerung:  $\text{prescaler} \in \{1, 8, 64, 256, 1024\}$
  
  - **Beispiel:**
    - 8-bit Timer mit Überlaufinterrupt
    - CPU Frequenz: 16 MHz (ATmega328PB)
    - Ziel: Mit Periode 1 s zählen
- ⇒ Welcher  $\text{prescaler}$  ist am ressourcenschonendsten?
- ⇒ Wie viele Überlaufinterrupts bis 1 s vergangen ist?
- ⇒ Welcher Fehler entsteht?

## Aufgabe: Ampel

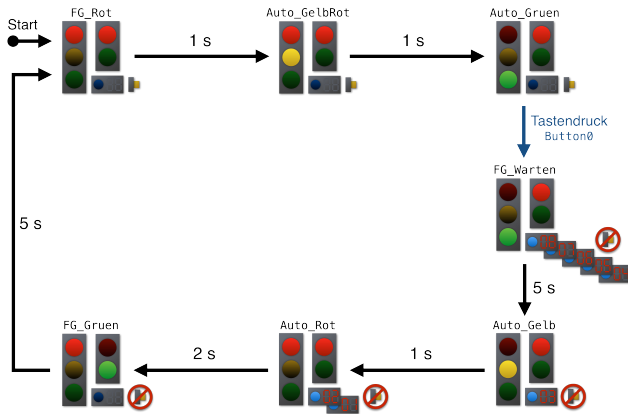
---



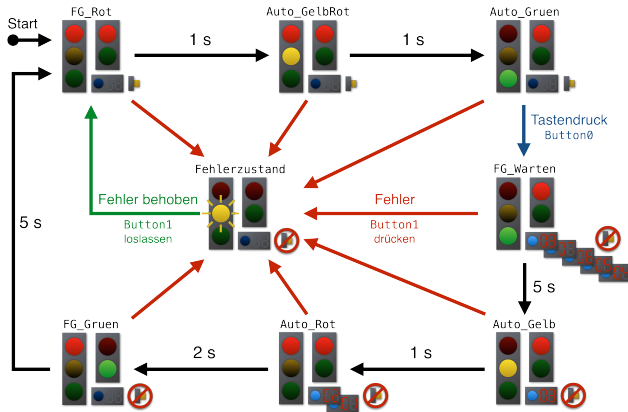
- Implementierung einer (Fußgänger-)Ampel mit Wartezeitanzeige



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



# Ampel als Zustandsautomat







- 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 Abarbeitung von Systemen mit versch. Zuständen  
⇒ Implementierung von Zustandsautomaten



- Alle Transitionen werden durch Interrupts ausgelöst
  - BUTTON0 und BUTTON1 für Interrupts konfigurieren
    - ⇒ Welche Flanke soll Interrupts auslösen?
  - TIMER0 konfigurieren (Einheit: 1 Sekunde)
- Keine Verwendung des Timer Moduls der libspicboard für die Abgabe
  - ⇒ Zum debuggen aber u.U. praktisch

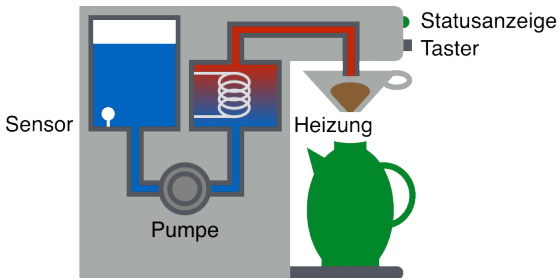


## ■ Hinweise

- Ablauf (exakt) nach Aufgabenbeschreibung (Referenzimplementierung verfügbar)
- Tastendrucke und Alarme als Ereignisse
- Passiv Warten auf die jeweiligen Interrupts
- Deaktivieren des Tasters durch Ignorieren des entsprechenden Interrupts (Änderung der Interrupt-Konfiguration ist nicht notwendig)
- Abbildung auf Zustandsautomaten sinnvoll
- Verwendung von `volatile` begründen

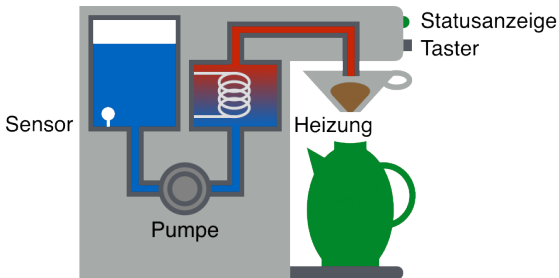
# Hands-on: Kaffeemaschine

Screencast: <https://www.video.uni-erlangen.de/clip/id/17647>



## ■ Lernziele:

- Zustandsautomaten
- Timer bzw. Alarm
- Interrupts & Schlafenlegen



## ■ Beschaltung:

- Pumpe & Heizung: Port D, Pin 5 (active-low)
- Taster: INT0 an Port D, Pin 2 (active-low)
- Sensor: INT1 an Port D, Pin 3 (Wasser: high; kein Wasser: low)
- Statusanzeige:
  - BLUE0: **STANDBY**
  - GREEN0: **ACTIVE**
  - RED0: **NO\_WATER**



## STANDBY

- Kaffeemaschine ist aus
- Pumpe und Heizung sind aus
- Benutzer kann Kaffeezubereitung durch Tastendruck starten
- Anfangszustand

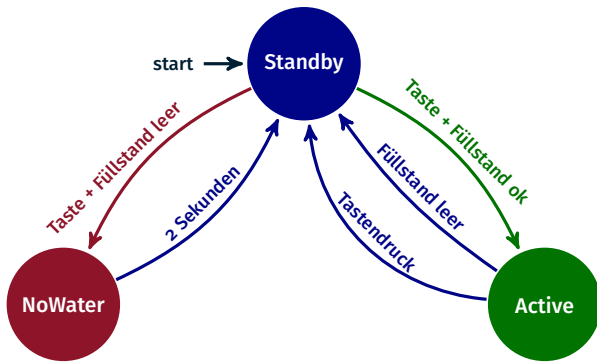
## ACTIVE

- Kaffeemaschine ist an
- Pumpe und Heizung sind an
- Wassertank ist nicht leer
- Benutzer kann Kaffeezubereitung durch Tastendruck beenden

## NO\_WATER

- Kaffeemaschine zeigt an, dass sie nicht genügend Wasser hat
- Pumpe und Heizung sind aus
- Zeitdauer: 2 Sekunden





## ■ Hinweise:

- Tastendruck & Füllstandsänderung durch Interrupts
- Statusanzeige: `void setLEDState(state_t state)`
- Wartephasen ggf. über SingleShot-Alarm realisieren
- In Wartephasen Mikrocontroller in den Energiesparmodus



**DDR<sub>x</sub>** hier konfiguriert man Pin  $i$  von Port  $x$  als Ein- oder Ausgang

- Bit  $i = 1 \rightarrow$  Pin  $i$  als Ausgang verwenden
- Bit  $i = 0 \rightarrow$  Pin  $i$  als Eingang verwenden

**PORT<sub>x</sub>** Auswirkung **abhängig von DDR<sub>x</sub>**:

- ist Pin  $i$  **als Ausgang konfiguriert**, so steuert Bit  $i$  im PORT<sub>x</sub> Register ob am Pin  $i$  ein high- oder ein low-Pegel erzeugt werden soll
  - Bit  $i = 1 \rightarrow$  high-Pegel an Pin  $i$
  - Bit  $i = 0 \rightarrow$  low-Pegel an Pin  $i$
- ist Pin  $i$  **als Eingang konfiguriert**, so kann man einen internen pull-up-Widerstand aktivieren
  - Bit  $i = 1 \rightarrow$  pull-up-Widerstand an Pin  $i$  (Pegel wird auf high gezogen)
  - Bit  $i = 0 \rightarrow$  Pin  $i$  als tri-state konfiguriert

**PIN<sub>x</sub>** Bit  $i$  gibt aktuellen Wert des Pin  $i$  von Port  $x$  an (nur lesbar)



- 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

- ATmega328PB: External Interrupt Mask Register (EIMSK)
- Die Bitpositionen in diesem Register sind durch Makros INTn definiert