

# Übungen zu Systemnahe Programmierung in C (SPiC) – Sommersemester 2025

---

## Übung 6

Maxim Ritter von Onciul

Eva Dengler

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



Lehrstuhl für Informatik 4  
Systemsoftware



Friedrich-Alexander-Universität  
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)
  - Pulswweitenmodulation (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 << CS02);  
04     TCCR0B |= (1 << CS01) | (1 << CS00);  
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 (...)      |

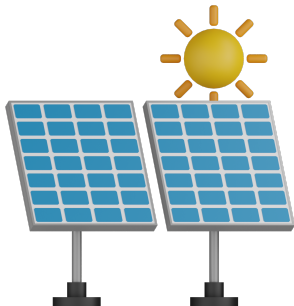
```
01 ISR(TIMER0_OVF_vect) {  
02     // [...]  
03 }  
04  
05 static void init(void) {  
06     // Überlaufunterbrechung aktivieren  
07     TIMSK0 |= (1 << TOIE0);  
08  
09     // [...]  
10 }
```

| 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_COMPA_vect) { /* [...] */ }
02
03 static void init(void) {
04     // Vergleichswert setzen
05     OCR0A = 125 - 1; // Die Maschine fängt bei Null an zu zählen.
06
07     // Damit der Zähler bei 124 wieder zurückgesetzt wird.
08     TCCR0A = (1<<WGM01);
09
10     // Vergleichswertunterbrechung aktivieren
11     TIMSK0 |= (1 << OCIE0A);
12
13     /* [...] */ }
```

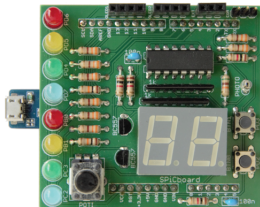


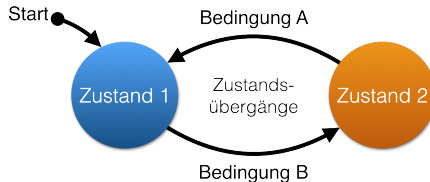
- Zur Erinnerung:  $\text{prescaler} \in \{1, 8, 64, 256, 1024\}$
  
- **Beispiel:**
  - 8-bit Timer mit Überlaufinterrupt
  - CPU Frequenz: 16 MHz (ATmega328PB)
  - Ziel: Mit Periode 1 s zählen
- ⇒ Welcher  $\text{prescaler}$  ist am ressourcenschonendsten?
- ⇒ Wie viele Überlaufinterrupts bis 1 s vergangen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



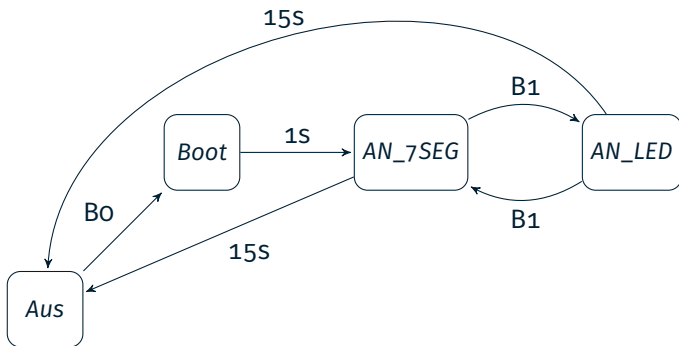
## Anzeige für ein Solarpaneel

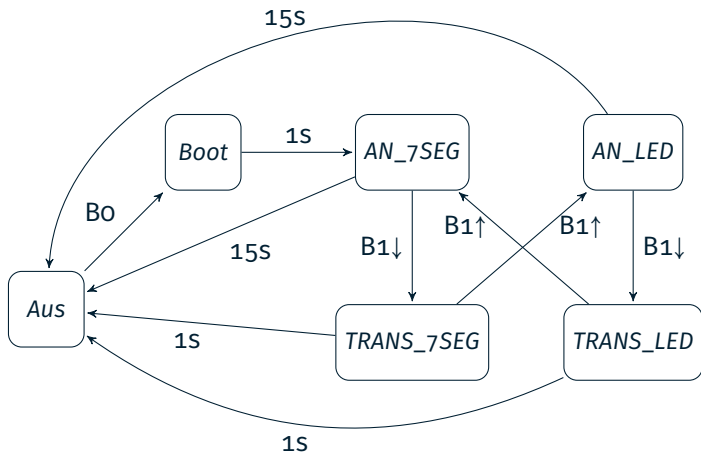
- PHOTO ist das Paneel
- Button0 schaltet das Paneel ein
- Button1 schaltet zwischen
  1. 7seg zeigt aktuelle Leistung
  2. LEDs zeigen aktuelle Leistung
- Button1 gedrückt halten
  1. Anzeige ausschalten





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







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

```
01 enum state { STATE_OFF, STATE_BOOT, STATE_ON };  
02  
03 enum state my_state = STATE_OFF;
```

- Mit typedef noch lesbarer:

```
01 typedef enum { STATE_OFF, STATE_BOOT, STATE_ON } state;  
02  
03 state my_state = STATE_OFF;
```



```
01 switch ( my_state ) {  
02 case STATE_OFF:  
03     ...  
04     break;  
05 case STATE_BOOT:  
06     ...  
07     break;  
08 case STATE_ON:  
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



- Wie kann ein Button zwischen kurz und lang unterscheiden?

⇒ Zwischenzustände und Timer notwendig.

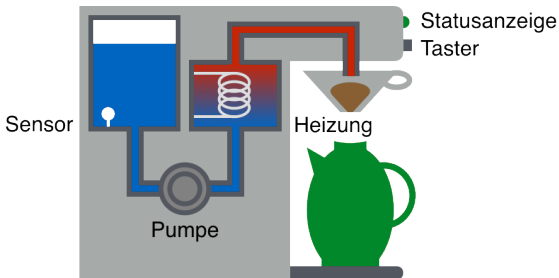
- Bei einer fallenden Flanke treten wir in den Übergangszustand
- Dieser kann durch eine steigende Flanke oder einen Timerinterrupt verlassen werden

## ■ Hinweise

- Ablauf (exakt) nach Aufgabenbeschreibung  
(Referenzimplementierung verfügbar)
- Tastendrucke 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

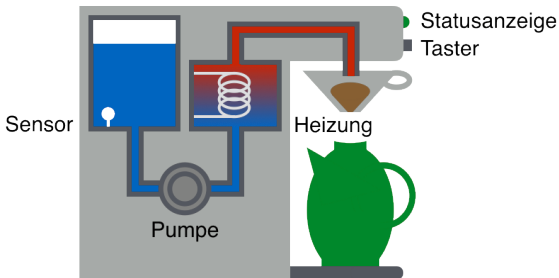
# Hands-on: Kaffeemaschine

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

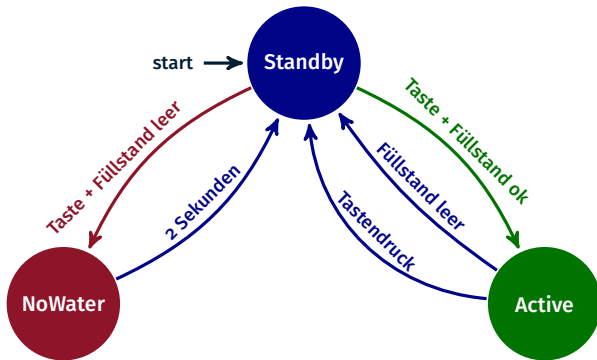
- Kaffeemaschine ist aus
- Pumpe und Heizung sind aus
- Benutzer kann Kaffeezubereitung durch Tastendruck starten
- Anfangszustand

### 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



**DDRx** 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

**PORTx** Auswirkung **abhängig von DDRx**:

- ist Pin  $i$  **als Ausgang konfiguriert**, so steuert Bit  $i$  im PORTx 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

**PINx** 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