

Embedded Systems

Kapitel 10: Automaten, Peripherie

Prof. Dr. Wolfgang Mühlbauer

Fakultät für Informatik

`wolfgang.muehlbauer@th-rosenheim.de`

Sommersemester 2020

Beispiel: Anforderungen an einfachen Getränkeautomat

- ❑ Annahme
 - Automat liefert nur 2 Getränkesorten.
- ❑ Einfacher Getränkeautomat erwartet
 - 1. Schritt: Einwurf von 1 EUR
 - 2. Schritt: Auswahl von 2 möglichen Getränken
- ❑ Bei anderem Verhalten: Keine Reaktion
- ❑ Drücken des Rückgabeknopfes
 - Eingeworfenes Geld (für nicht gelieferte Ware) wird zurückgegeben.



gemeinfrei

Schaltnetze, Automaten

□ Schaltnetze

- == Boolesche Funktion, die Eingabe auf Ausgabe abgebildet.
- Implementierung: Gatter, Wahrheitstabellen
- *Kein Gedächtnis*: Schaltnetze vergessen vorherige Eingaben.

□ Automat

- Endliches Eingabealphabet: $X = \{x_1, x_2, x_3, \dots, x_m, \}$
- Endliches Ausgabealphabet: $Y = \{y_1, y_2, y_3, \dots, y_p, \}$
- **Zeitliche Folge** von Elementen des **Eingabealphabets** wird in zeitliche Folge von Elementen eines **Ausgabealphabets** abgebildet.
 - *Gedächtnis*: Bisherige Eingabe hat Einfluss auf die folgende Ausgabe.
 - Fester Eingabetakt oder Warten auf **asynchrone** Ereignisse.



Eingabe und Ausgabe im Beispiel

□ **Eingabealphabet: X**

- m : Euro einwerfen,
- $w1$: Sorte auswählen
- $w2$: Sorte auswählen
- r : Geldrückgabe anfordern

□ **Ausgabealphabet: Y**

- G : Geld zurück
- k : keine Reaktion
- $a1$: Sorte 1 ausgeben
- $a2$: Sorte 2 ausgeben



❑ Gleichwertige Begriffe

- Endlicher Automat
- Zustandsmaschine
- Zustandsautomat

❑ Endlicher Automat

- Endlich viele Zustände. Selbst wenn Eingabesequenz unendlich!
- Zustand kodiert Aspekte der Vergangenheit.

❑ 2 Grundtypen

- **Akzeptoren:** Akzeptieren und erkennen eine Eingabe
 - Anwendung: Wort- und Spracherkennung
- **Transduktoren:** Generieren Ausgaben
 - Anwendung: Steuerungsaufgaben

Automaten: Definition

- ❑ Formale Beschreibung eines Automaten $A(X, Y, Z, \delta, \lambda)$:
 - $X = \{x_1, x_2, x_3, \dots, x_m\}$ die endliche Menge der **Eingabewerte**
 - $Y = \{y_1, y_2, y_3, \dots, y_p\}$ die endliche Menge der **Ausgabewerte**
 - $Z = \{z_1, z_2, z_3, \dots, z_n\}$ die endliche **Zustandsmenge**,
 - δ die **Zustandsüberföhrungsfunktion** und
 - λ : **Ausgabefunktion** beschreibt.
 - Details, siehe nächste Folie.

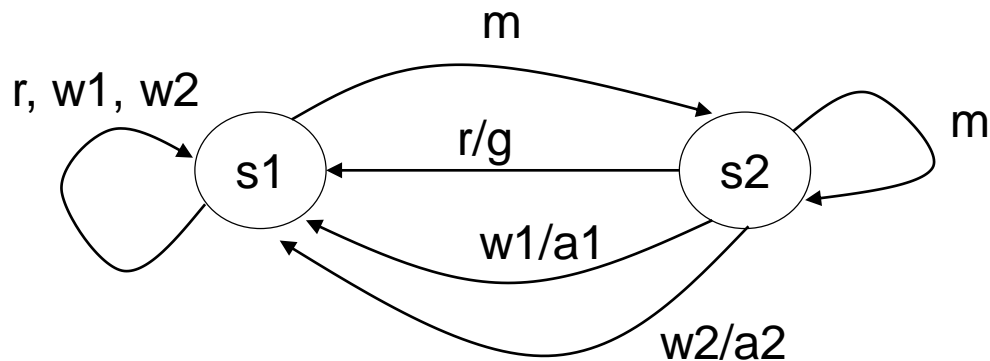
- ❑ Darstellung der Funktionen δ und λ häufig in
 - **Tabellenform** oder als
 - **Zustandsdiagramm**

Beispiel: Getränkeautomat

□ Darstellung als Tabelle

δ / λ	m	r	w ₁	w ₂
s ₁	s ₂ / k	s ₁ / k	s ₁ / k	s ₁ / k
s ₂	s ₂ / k	s ₁ / g	s ₁ / a ₁	s ₁ / a ₂

□ Darstellung als Zustandsdiagramm



- **Eingabealphabet: X**
 - m: Euro einwerfen,
 - w1: Sorte auswählen
 - w2: Sorte auswählen
 - r: Geldrückgabe anfordern
- **Ausgabealphabet: Y**
 - g: Geld zurück
 - k: keine Reaktion
 - a1: Sorte 1 ausgeben
 - a2: Sorte 2 ausgeben

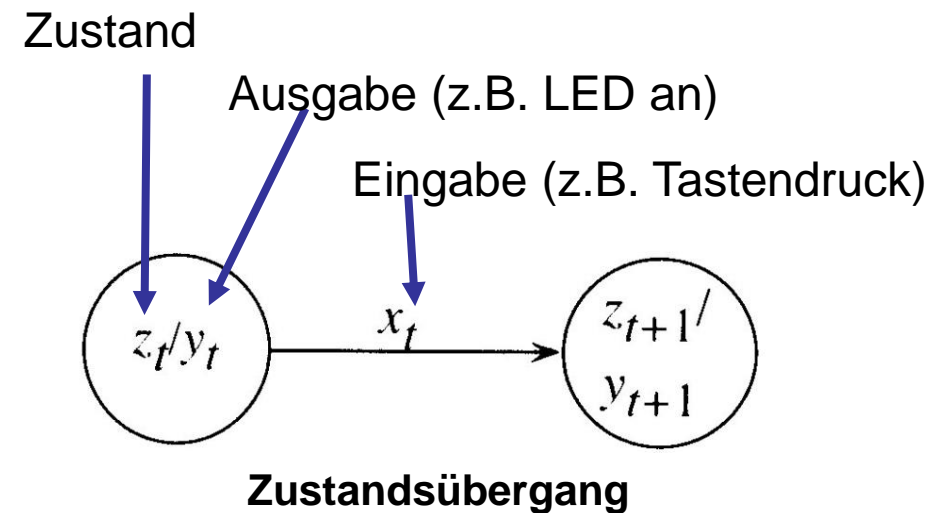
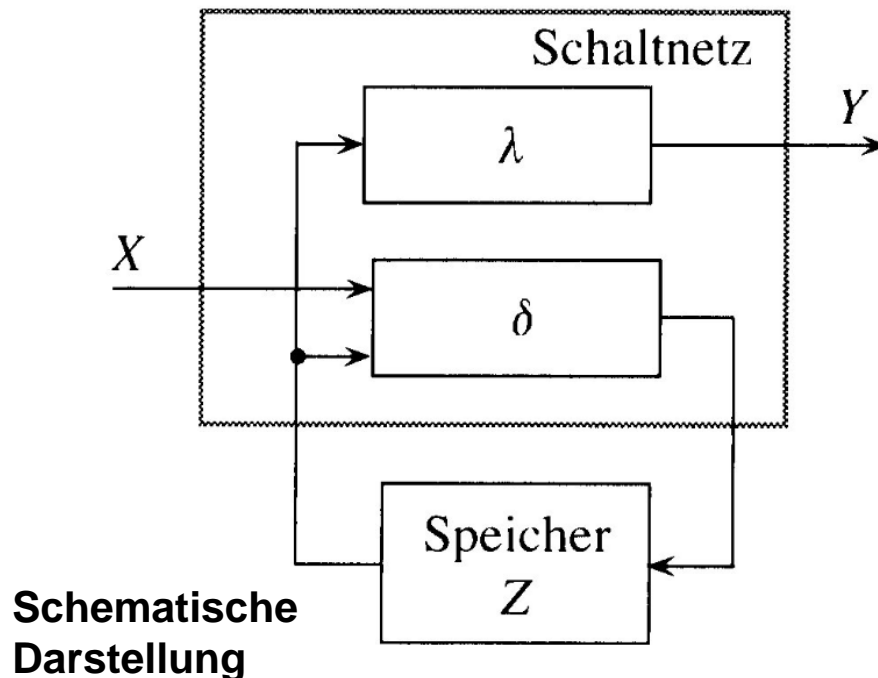
Moore Automat

- Ausgabe hängt **nur vom aktuellen Zustand** ab.

- *Ausgabefunktion: $\lambda: Z \rightarrow Y$*
- *Zustandsüberföhrungsfunktion: $\delta: Z \times X \rightarrow Z$*

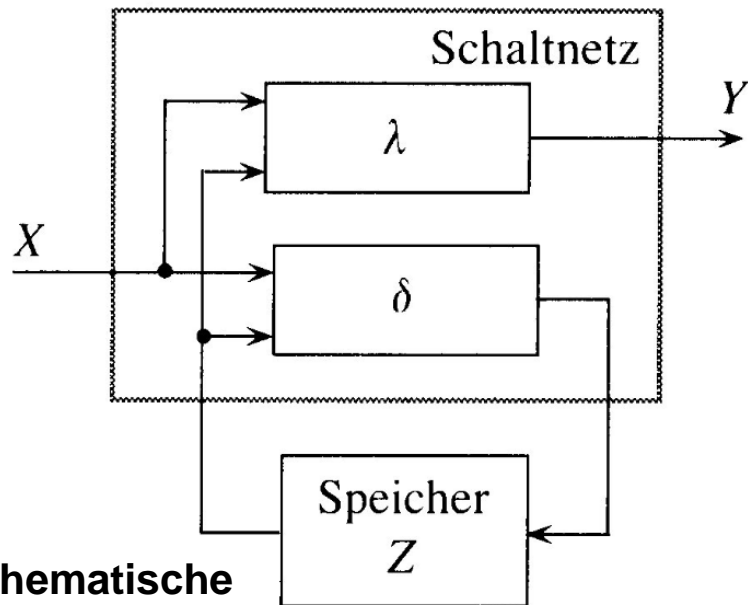
- Typische Beispiele

- Synchrone Zähler und Schieberegister

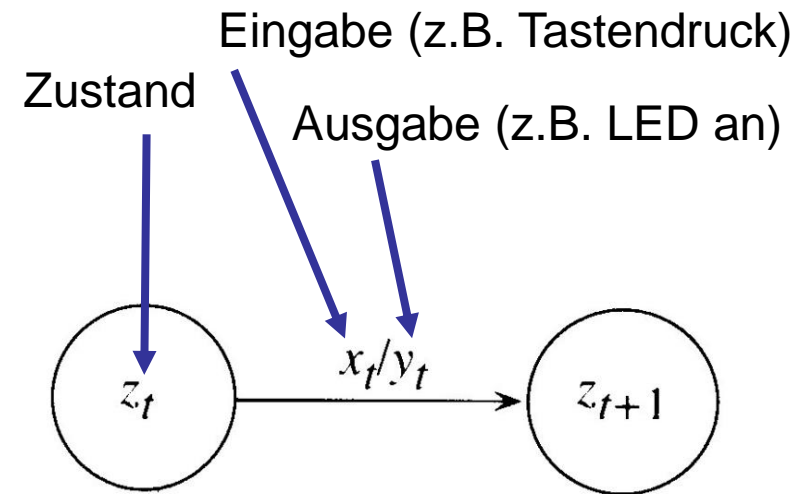


Mealy Automat

- Die Ausgabe hängt von Eingabe **UND** aktuellem Zustand ab.
 - Zustandsüberföhrungsfunktion: $\delta: Z \times X \rightarrow Z$
 - Ausgabefunktion: $\lambda: Z \times X \rightarrow Y$, die Ausgabe ist an ein Ereignis gebunden!
- Zu jedem Mealy Automat lässt sich ein gleichwertiger Moore Automat konstruieren.



Schematische
Darstellung



Zustandsübergang

UML Zustandsdiagramme (1)

❑ **Zustände:** Verhalten

- bei Eintritt in Zustand (**entry**)
- bei Austritt aus Zustand (**exit**)
- während des Zustands (**do**)

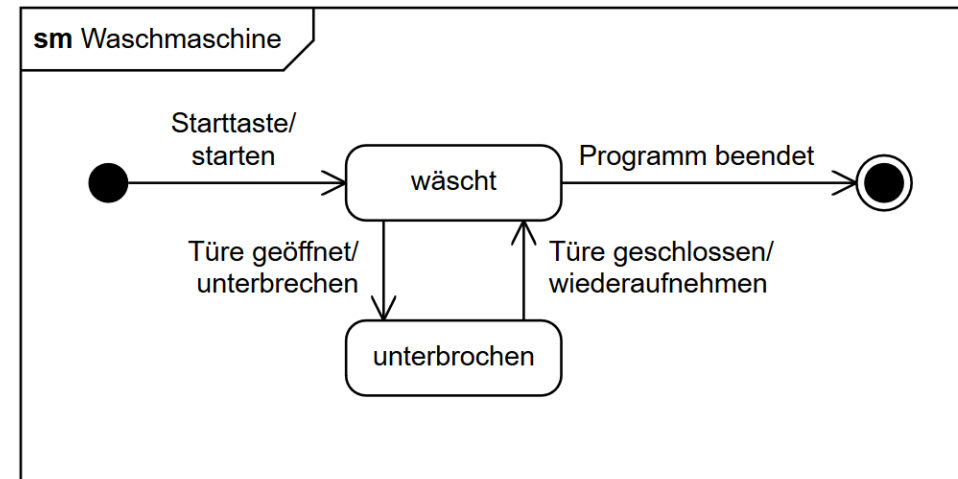
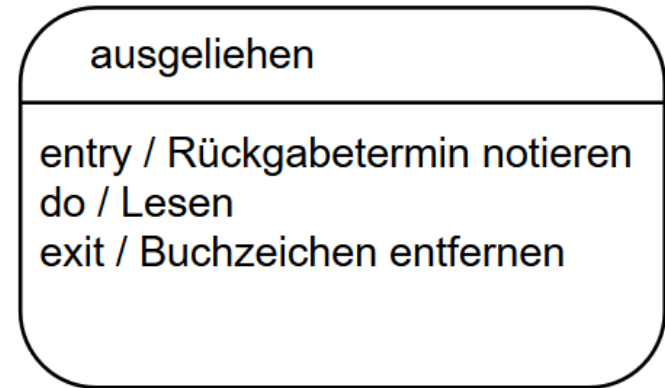
❑ Spezielle **Start- und Endzustände**

❑ **Transitionen**

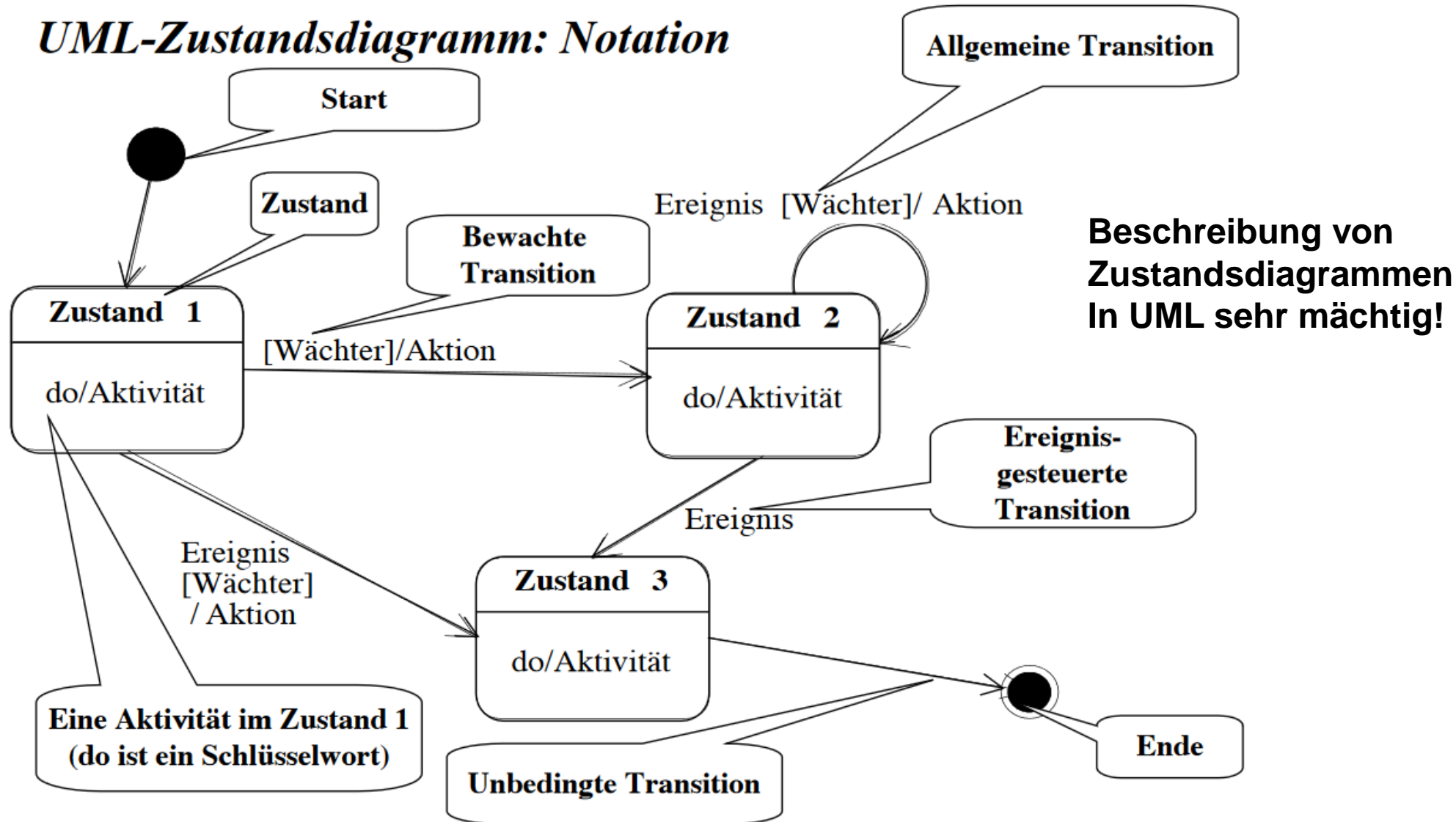
- "Zustandsüberföhrungsfunktion"
- Transition ("unterbrechen") wird nur ausgeführt, falls Ereignis "Türe geöffnet" eintritt.

❑ Zahlreiche weitere "Features"

- Z.B. Hierarchie von Zuständen, Unterzustände

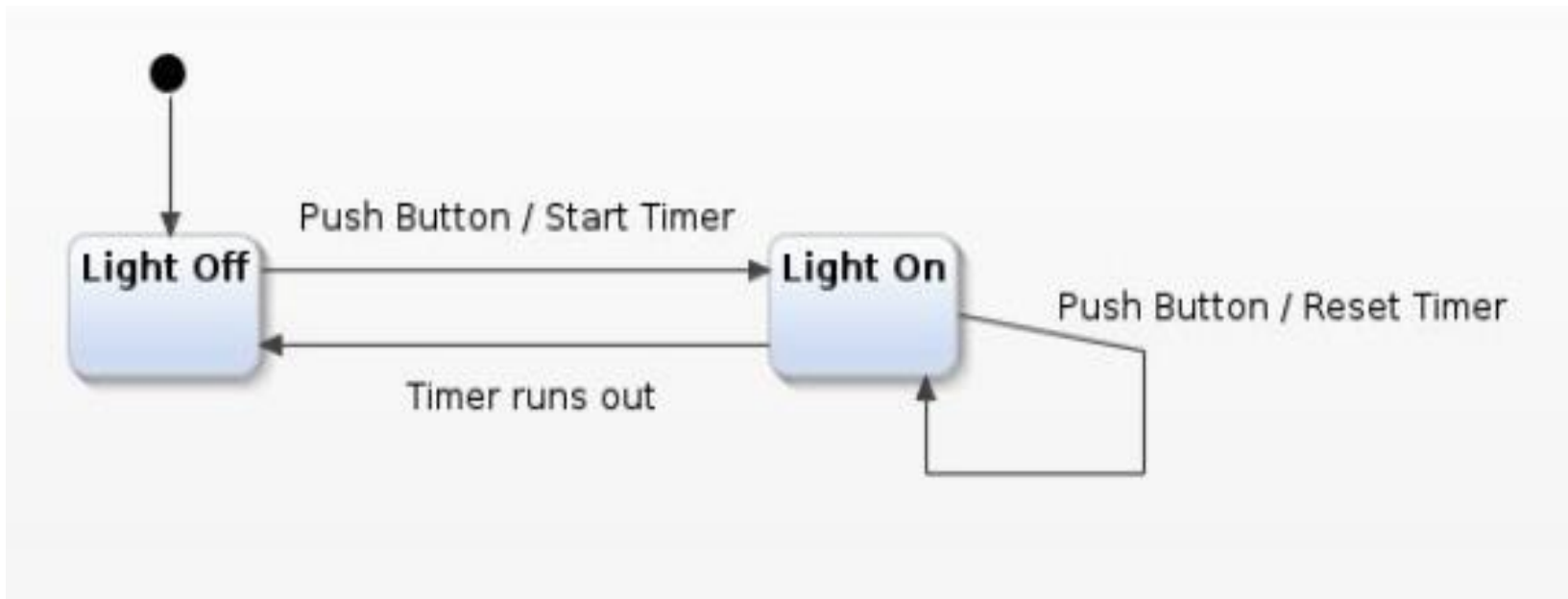


UML Zustandsdiagramme (2)



Fallbeispiel: Zeitgesteuerter Lichtschalter

- ❑ Licht normalerweise aus.
- ❑ Drücken des Tasters schaltet Licht an.
- ❑ Licht schaltet sich nach festem Zeitintervall automatisch ab.



Variante 1: Implementierung über Case Struktur

```
#define BUTTON 22    // Digital Pin 22
#define LED 21       // Digital Pin 21
```

```
typedef enum {
    LIGHT_OFF,
    LIGHT_ON
} state_t;    // enum for all states
```

```
state_t state; // keep current state
```

```
// start time of timer
unsigned long timer = 0;
```

```
void setup() {
    state = LIGHT_OFF; // set initial state
    pinMode(LED, OUTPUT);
    digitalWrite(LED, LOW);
    digitalWrite(BUTTON, HIGH); //Pull-up
}
```

```
void loop() {
    state_machine();
}
```

```
void state_machine() {
    switch (state) {
        case LIGHT_OFF:
            if (digitalRead(BUTTON) == LOW) {
                digitalWrite(LED, HIGH);
                timer = millis(); // start timer
                state = LIGHT_ON;
            }
            break;
        case LIGHT_ON:
            if (millis() - timer > 5000) {
                digitalWrite(LED, LOW);
                state = LIGHT_OFF;
            }
            else if (digitalRead(BUTTON) == LOW) {
                timer = millis(); // start timer
            }
            break;
    }
}
```


Kein
delay

Exkurs: Funktionszeiger in C

□ "Normaler Zeiger"


- Variable, die Adresse eines Wertes speichert.
- Deklaration: `int *a`

```
int v = 5;  
int *a = &v;
```



□ Funktionszeiger

- Variable, die Adresse einer Funktion speichert.
- Deklaration:
 - `void (*func) (int)`
 - Zeiger auf Funktion, die `int` als Parameter erwartet und keinen Rückgabewert hat.



```
void print_func(int x) {  
    printf("Der Integer ist: %d", x);  
}  
  
int main(void) {  
    // declare function variable  
    void (*funcp) (int);  
    // assign actual function  
    funcp = &print_func;  
    // call actual function  
    funcp(5);  
}
```

Variante 2: Implementierung über Tabelle (1)

- ❑ **Idee:** Zustandsübergangstabelle als 2-dimensionales Array
 - *Zeilen:* Zustände z
 - *Spalten:* Mögliche Ereignisse e
 - *Eintrag:* Funktion, die bei Eintreten von Ereignis e in Zustand z ausgeführt werden soll.

- ❑ **Hinweis:**
 - Funktionszeiger bzw. Funktion muss separat implementiert werden.

```
// state/event lookup table (stores function pointers)
void (*state_table[2][3]) (void) = {
    NO_EVENT      TIMER_EXPIRED      BUTTON_PRESSED
    /* LIGHT_OFF*/ { idle,             keep_state,      startTimer },
    /* LIGHT_ON*/  { idle,             timerExpires,    resetTimer }
};
```

Variante 2: Implementierung über Tabelle (2)

```
#define BUTTON 22 // Digital Pin 22
#define LED 21 // Digital Pin 21
typedef enum {
    LIGHT_OFF,
    LIGHT_ON
} state_t; // enum datatype for states
typedef enum {
    NONE,
    TIMER_EXPIRED,
    BUTTON_PRESSED
} event_t; // enum datatype for events
```

```
state_t state; // keep current state
unsigned long timer = 0; // timer start time
```

```
void startTimer (void) {
    digitalWrite(LED, HIGH);
    state = LIGHT_ON; // set new state
    timer = millis(); // start timer
}
void resetTimer (void) {
    timer = millis();
}
void timerExpires (void) {
    digitalWrite(LED, LOW);
    state = LIGHT_OFF; // set new state
}
void idle(void) {
}
```

```
void setup() {
    pinMode(LED, OUTPUT);
    digitalWrite(LED, LOW);
    digitalWrite(BUTTON, HIGH); // Pull-UP
    state = LIGHT_OFF; // initial state
}
```

```
// lookup table with function pointers
void (*state_table[2][3]) (void) = {
    // NO_EVENT TIMER_EXPIRED BUTTON_PRESSED
    /* LIGHT_OFF*/ {idle, idle, startTimer},
    /* LIGHT_ON*/ {idle, timerExpires, resetTimer }
};
```

```
void loop() {
    state_machine();
}
```

```
void state_machine() {
    // check for new events
    event_t event = NONE; // default: no event
    if (timer && millis() - timer > 5000) {
        event = TIMER_EXPIRED;
        timer = 0; ← Timer deaktivieren!
    } else if (digitalRead(BUTTON) == LOW) {
        event = BUTTON_PRESSED;
    }
    // go to next state
    state_table[state][event](); ← Funktionsaufruf!
}
```


Implementierungsaspekte

❑ **Aufruf des Zustandsautomaten**

- Meist periodisch, z.B prüfe jede Millisekunde auf Ereignis bzw. Zustandsübergang
- Bei Betriebssystem: Zustandsautomat als Task implementieren

❑ **Erkennen von Ereignissen**

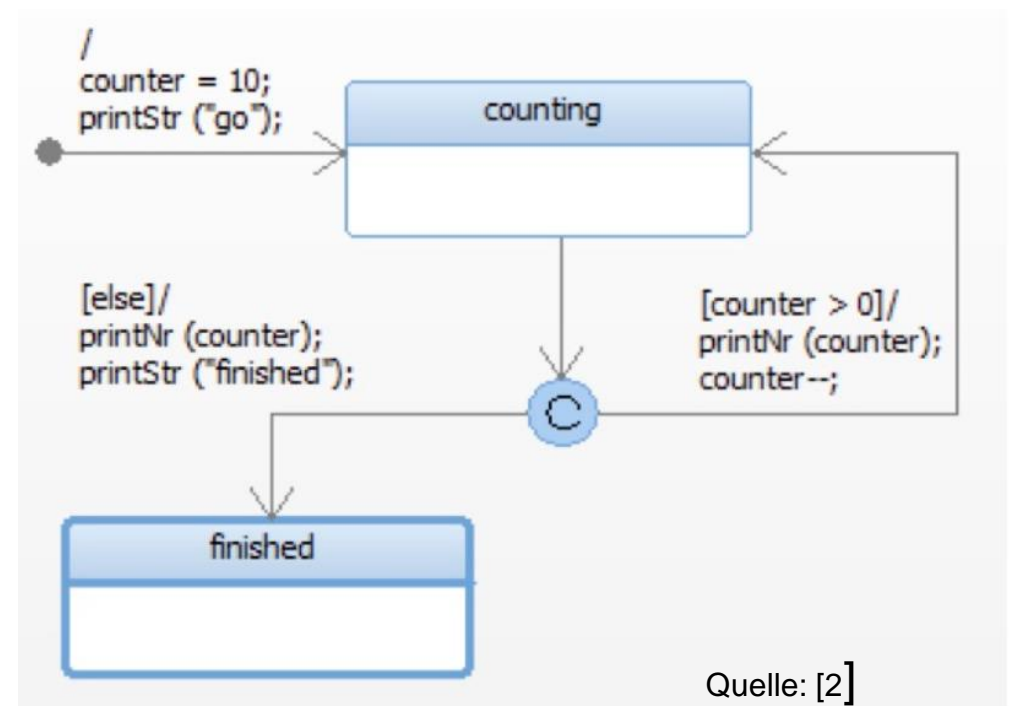
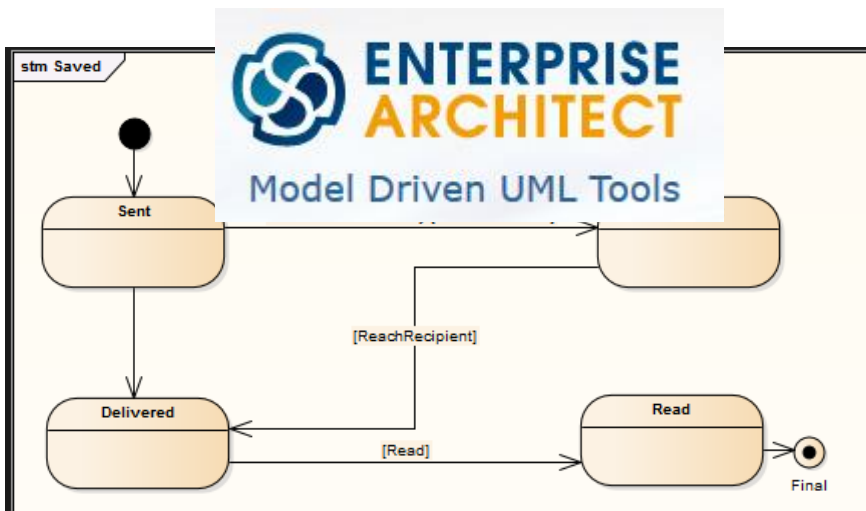
- Timer-Ereignisse: Kein blockierendes `delay(.)`, sondern `millis()` oder Interrupt-Flags
- Implizite Priorisierung von Ereignissen
 - Switch/Case: Ereignis A vor Ereignis B in case → A bevorzugt!
 - Tabelle: Zuerst geprüfte Ereignisse überdecken später geprüfte Ereignisse!

❑ **Welcher Implementierungsansatz?**

- Case-Anweisung
 - Gut, falls Übergangsmatrix relativ leer
 - Evtl. etwas performanter
- Tabelle, Funktionszeiger
 - Hohe Lesbarkeit
 - Gute Erweiterbarkeit
- [State Pattern]
 - Nur bei Objektorientierung, z.B. C++

Automatische Codegenerierung

- ❑ Zeichne Zustandsautomat, generiere Code
- ❑ Angenehm: Man muss sich nicht um Timer, Ereignisbehandlung, etc. kümmern.
- ❑ **Tools**
 - IBM Rhapsody
 - Yakindu State Chart Tool
 - Enterprise Architect



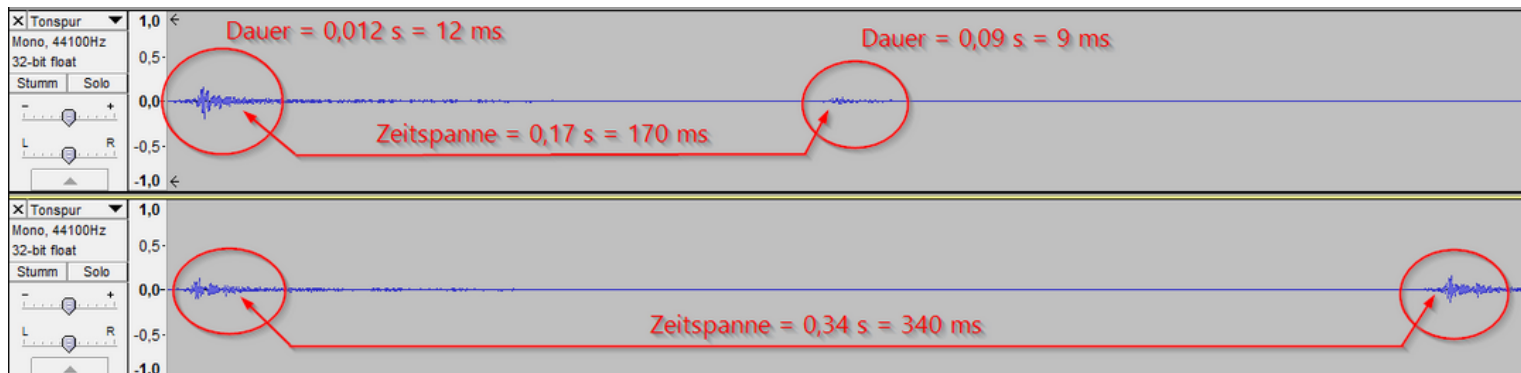
Übung: "Doppelklatschen"

□ Anwendung

- Zustandsautomat, der doppeltes Klatschen erkennt.
- Doppeltes Klatschen soll die LED „toggeln“.
- == Fernsteuerung für einen Lichtschalter.

□ Erfahrung aus der Praxis

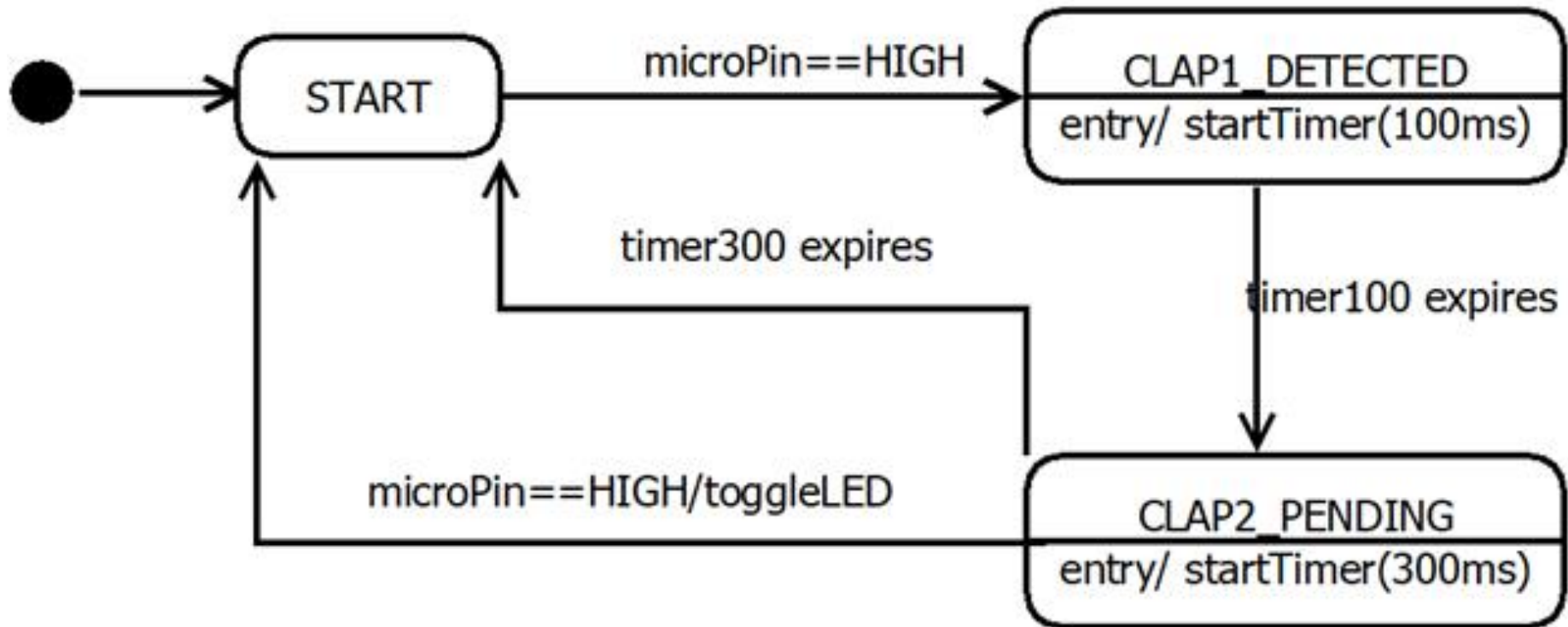
- Nach 1. Klatschen mindestens 100 ms warten, um das 1. Klatschen sauber vom 2. Klatschen zu unterscheiden.
- Spätestens 300 ms nach 1. Klatschen muss das 2. Klatschen erfolgen.
- Wie sieht der Zustandsautomat aus?



Audacity: **Wie schnell klatscht man typischerweise hintereinander?**

Zustandsautomat

- Wie erkennt man “doppeltes Klatschen”?



Mikrofonsensor SE019

□ **Analog A0**

- Analoge Spannung proportional zur gemessenen Lautstärke
- AD Wandler in Arduino notwendig

□ **Digital D0**

- Logisch HIGH falls Lautstärke über Schwellwert
- Schwellwert durch Potentiometer einstellbar.



http://www.produktinfo.conrad.com/datenblaetter/1400000-1499999/001485300-da-01-en-IDUINO_SE019_MICROFON_SOUND_SENSOR_MODUL.pdf

Pin	Description
A0	Analog signal output pin
G	Ground
+	Power(5V/3.3V)
D0	Digital signal output pin

Quellenverzeichnis

- [1] G. Gridling und B. Weiss. *Introduction to Microcontrollers*, Version 1.4, 26. Februar 2007, verfügbar online: <https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf> (abgerufen am 08.03.2017)
- [2] <http://www.willert.de/assets/Schulung-Handout/TraM-Trng-Modeling-UML-Startup-Basics-in-Rhapsody.pdf> (abgerufen am 19.06.2017)
- [3] <https://www.sparxsystems.de/start/startseite/> (abgerufen am 19.06.2017)
- [4] <https://public.hochschule-trier.de/~rudolph/gdv/cg/node48.html> (abgerufen am 19.06.2017)