

# Embedded Systems / Eingebettete Systeme

Studiengang Informatik  
Campus Minden

Matthias König

**FH Bielefeld**  
University of  
Applied Sciences



# Beispiel einer bekannten Anwendung: Prozessorchipkarte

- Verwendung meistens zur Identifikation, Authentifizierung, Datenspeicherung
- Mikroprozessor, RAM, ROM (Betriebssystem), EEPROM, Input/Output
- Kommunikation über Pinout
- Standardisiert: ISO/IEC 7810

Pinout Chipkarte



[Quelle: <http://commons.wikimedia.org/wiki/File:SmartCardPinout.svg>]

WIEDERHOLUNG

# Arduino - Blink in AVR-Assembler

```
.section .text
.global _main

_main:
    cli          ; no interrupts
    sbi 0x04, 5  ; set bit 5 of i/o reg 0x04 (portB5 output)
```

34 Byte

```
loop:
    sbi 0x05, 5  ; set bit 5 of 0x05 (portB5=led on)
    rcall wait   ; wait
    cbi 0x05, 5  ; clear bit 5 of 0x05 (portB5=led off)
    rcall wait   ; wait
    rjmp loop    ; jump back, endless loop

wait:
    ; just waiting
    clr r26      ; clear r24, r25, r26
waitCount:
    ; inner wait loop
    clr r24
    clr r25

waitLoop:
    adiw r24, 1  ; inc word r24,25
    brvs waitNext ; branch to next if overflow
    rjmp waitLoop ; otherwise to waitLoop

waitNext:
    inc r26      ; inc r26
    cpi r26, 0x40 ; compare with 0x40
    brlo waitCount ; branch to waitCount if lower (inner loop)

waitEnd:
    ret          ; return from sub function

.end
```

# Arduino - Blink in C

```
// Arduino Uno PIN 13 is connected to PortB5 of ATmega328P
typedef unsigned char uint8_t; // set type for unsigned int 8

void simpleDelay(unsigned int f_iterations) {
    // loop with nop for delay
    for (unsigned long i = 0; i < f_iterations; ++i) {
        for (volatile unsigned long j = 0; j < 1000; ++j) ;
    }
}

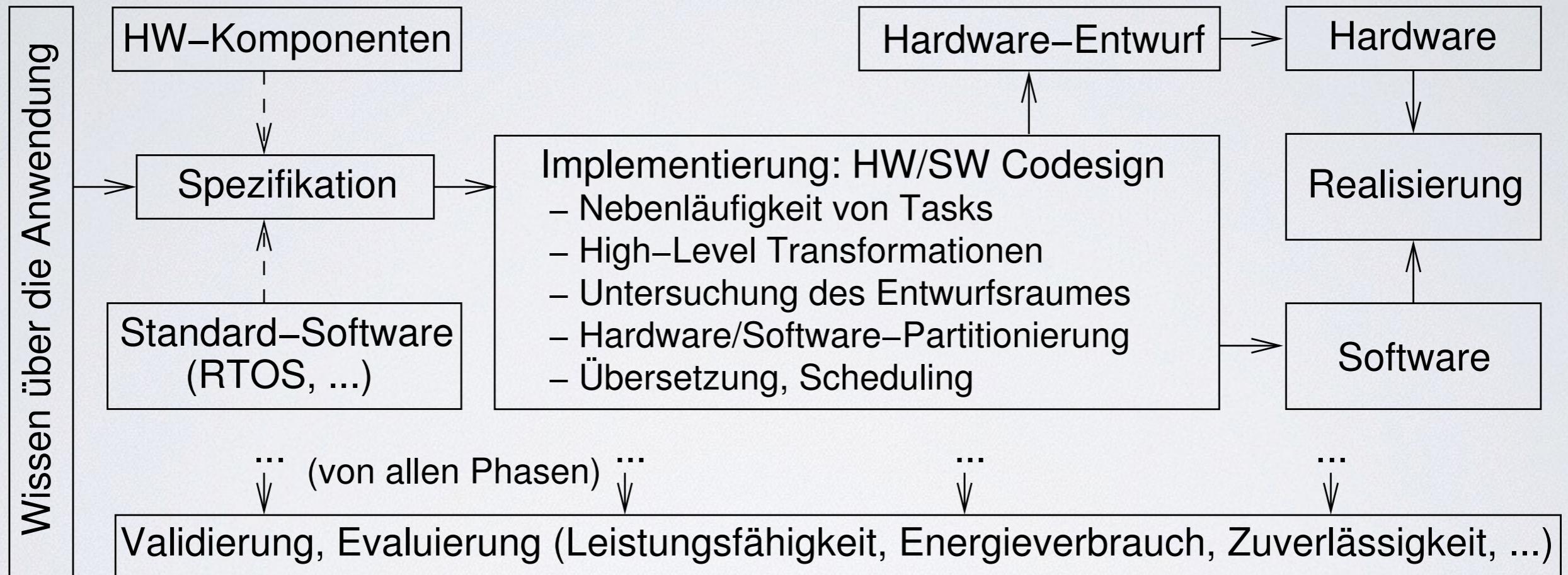
int main() {
    // set direction of DDRB5
    uint8_t *portAdd = (uint8_t*) 0x24; // DDRB address
    *portAdd |= 0x20; // set bit 5

    for (;;) { // infinite loop
        uint8_t *portB = (uint8_t*) 0x25; // PortB address
        *portB |= 0x20; // set bit 5
        simpleDelay(1000);
        *portB &= 0xDF; // clear bit 5
        simpleDelay(1000);
    }
}
```

368 Byte  
gcc mit Option -O1

```
ldi r28, 0x25
ldi r29, 0x00
ld r24, Y
ori r24, 0x20
st Y, r24
```

# Hardware/Software Codedesign



Entwurf Eingebetteter Systeme (nach Marwedel)

[Quelle: Marwedel, Eingebettete Systeme]

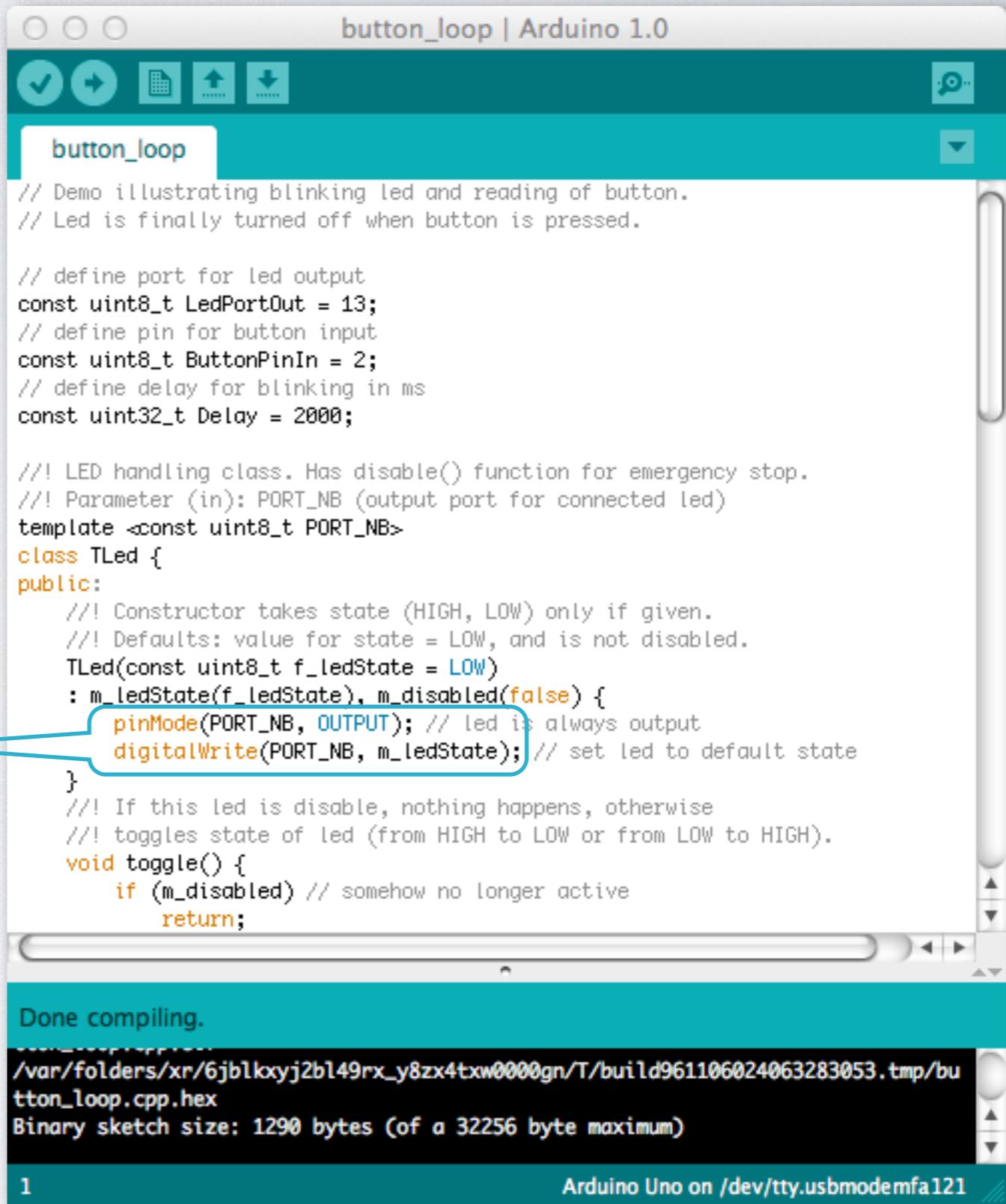
# Entwicklung Eingebetteter Systeme

# Entwicklungswerkzeuge

Beispiel Arduino:

Entwicklungsumgebung  
und Bibliotheken basierend  
auf C++

Bibliotheksfunktionen



```
button_loop | Arduino 1.0

button_loop

// Demo illustrating blinking led and reading of button.
// Led is finally turned off when button is pressed.

// define port for led output
const uint8_t LedPortOut = 13;
// define pin for button input
const uint8_t ButtonPinIn = 2;
// define delay for blinking in ms
const uint32_t Delay = 2000;

/// LED handling class. Has disable() function for emergency stop.
/// Parameter (in): PORT_NB (output port for connected led)
template <const uint8_t PORT_NB>
class TLed {
public:
    /// Constructor takes state (HIGH, LOW) only if given.
    /// Defaults: value for state = LOW, and is not disabled.
    TLed(const uint8_t f_ledState = LOW)
        : m_ledState(f_ledState), m_disabled(false) {
        pinMode(PORT_NB, OUTPUT); // led is always output
        digitalWrite(PORT_NB, m_ledState); // set led to default state
    }
    /// If this led is disable, nothing happens, otherwise
    /// toggles state of led (from HIGH to LOW or from LOW to HIGH).
    void toggle() {
        if (m_disabled) // somehow no longer active
            return;
    }

Done compiling.

/var/folders/xr/6jblkxyj2bl49rx_y8zx4txw0000gn/T/build961106024063283053.tmp/bu
tton_loop.cpp.hex
Binary sketch size: 1290 bytes (of a 32256 byte maximum)

1
```

Arduino Uno on /dev/tty.usbmodemfa121

# Softwareentwicklungsprozess

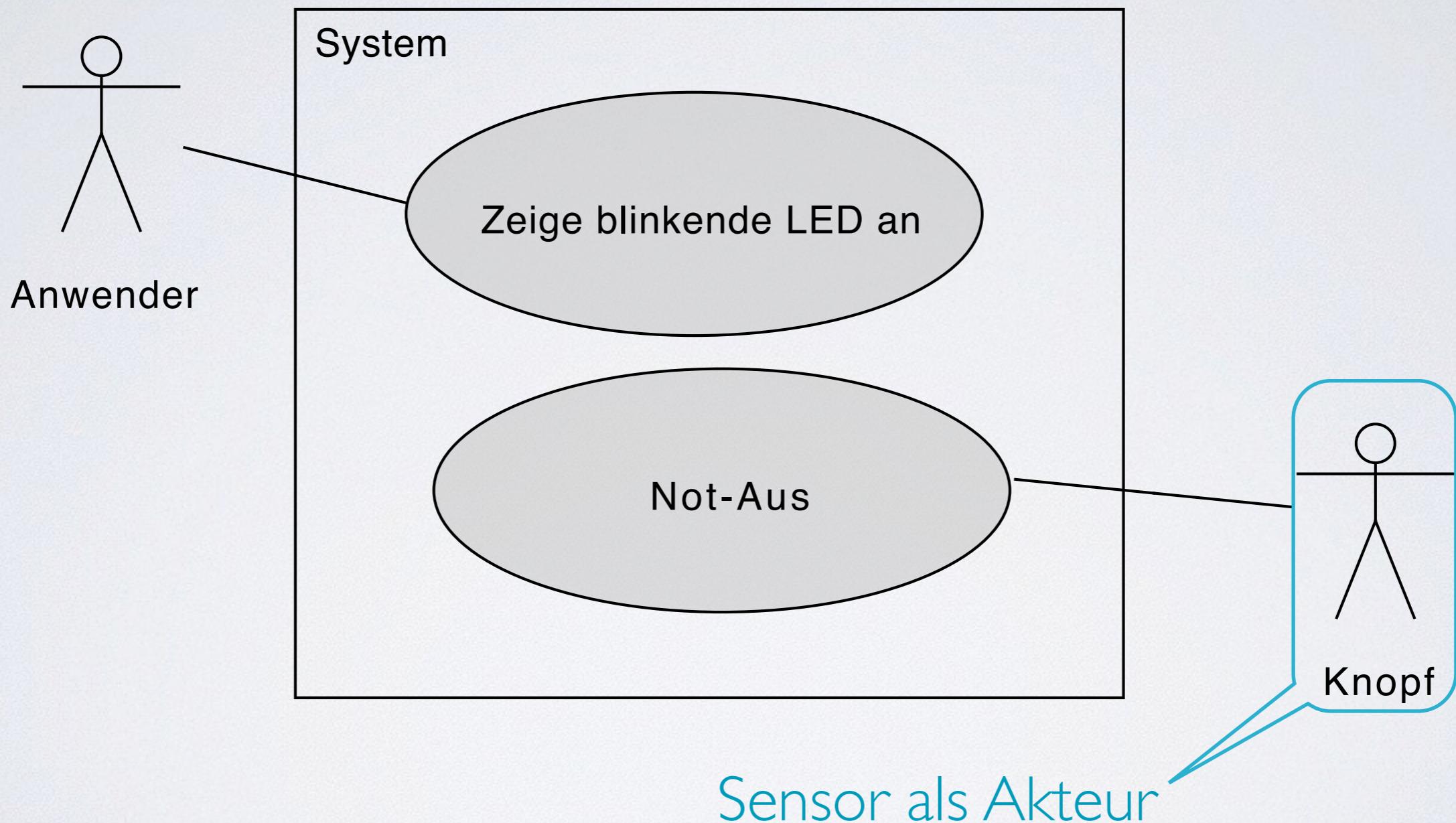
beinhaltet generell:

- Anforderungen / Requirements
- Entwurf / Design
- Implementierung / Construction
- Test / Testing
- Wartung / Maintenance

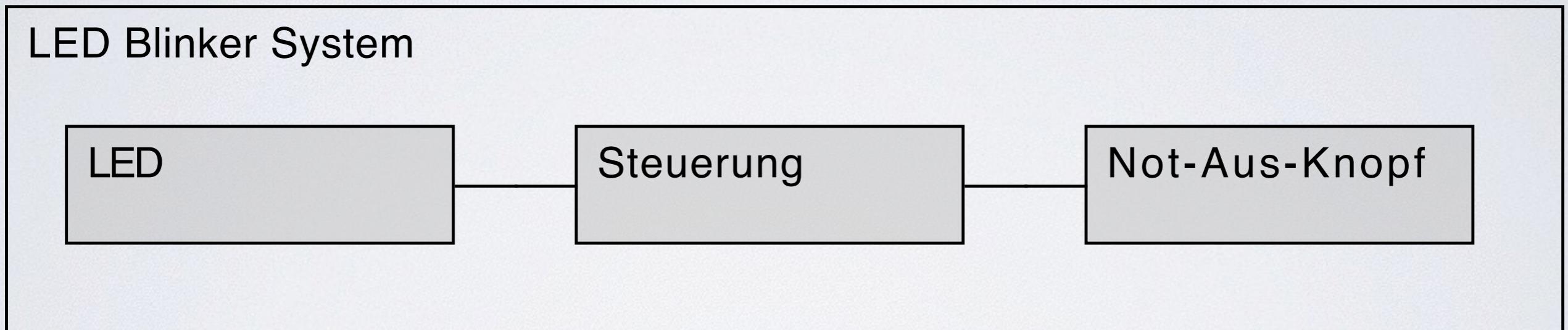
# Anforderungen / Requirements

- Feststellung von Anforderungen (Requirements engineering) entsprechend Vorgehen des Software Engineering (vgl. entsprechende Veranstaltung), beispielsweise mit
  - Anwendungsfällen
  - Kompositionssstrukturdiagramm

# Beispiel: Anwendungsfall



# Beispiel: Kompositionssstrukturdiagramm



# Entwurf: Spezifikation & Modellierung

- Spezifikation / Modellierung für systematisches Vorgehen bei nachfolgenden Schritten bezüglich
  - Prüfung auf Widerspruchsfreiheit, Vollständigkeit
  - Herleitung der Implementierung
- Maschinenlesbarkeit vorteilhaft

[Quelle: Marwedel, Eingebettete Systeme]

# Anforderungen an Spezifikationssprachen

- Abbildung von Hierarchie (Abstraktion zum Herunterbrechen von Komplexität, besser lesbar für Menschen)
  - Verhaltenshierarchie,  
z.B. hierarchische Zustände, Funktionen
  - Strukturelle Hierarchie,  
Zusammensetzung aus physikalische Komponenten

[Quelle: Marwedel, Eingebettete Systeme]

# Anforderungen an Spezifikationssprachen

- Unterstützung von reaktiven Systemen hinsichtlich:
  - Zuständen
  - Ereignissen (externe/interne)
  - Ausnahmen

[Quelle: Marwedel, Eingebettete Systeme]

# Anforderungen an Spezifikationssprachen

- Darstellung von Zeitverhalten
- Nebenläufigkeit (auch bei verteilten Systemen)
- Synchronization und Kommunikation
- Plattformunabhängig
- Ausführbar, prüfbar
- Unterstützung bei Entwurf komplexer Systeme

[Quelle: Marwedel, Eingebettete Systeme]

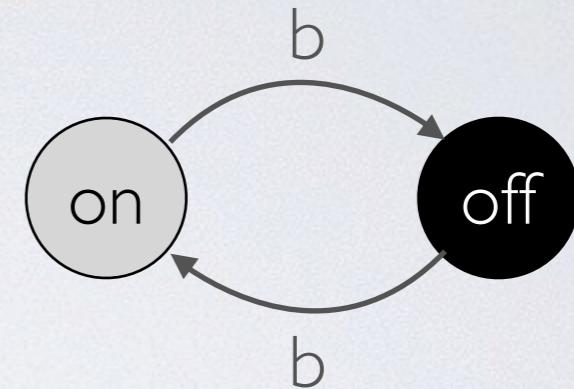
# Spezifikationssprachen

- stellen aufgrund der vielen Anforderungen immer einen **Kompromiss** dar,
- unterscheiden sich bei Modellen für
  - Ablauf von Berechnungen (sequentiell, Zustand, Zeitstempel)
  - Kommunikation (shared memory, message passing)

# Endliche Automaten / Finite State Machines

Systembeschreibung durch

- Zustände / State
- Ereignisse / Event
- Übergänge / Transition
  - wenn Ereignis b im Zustand on → gehe in Zustand off

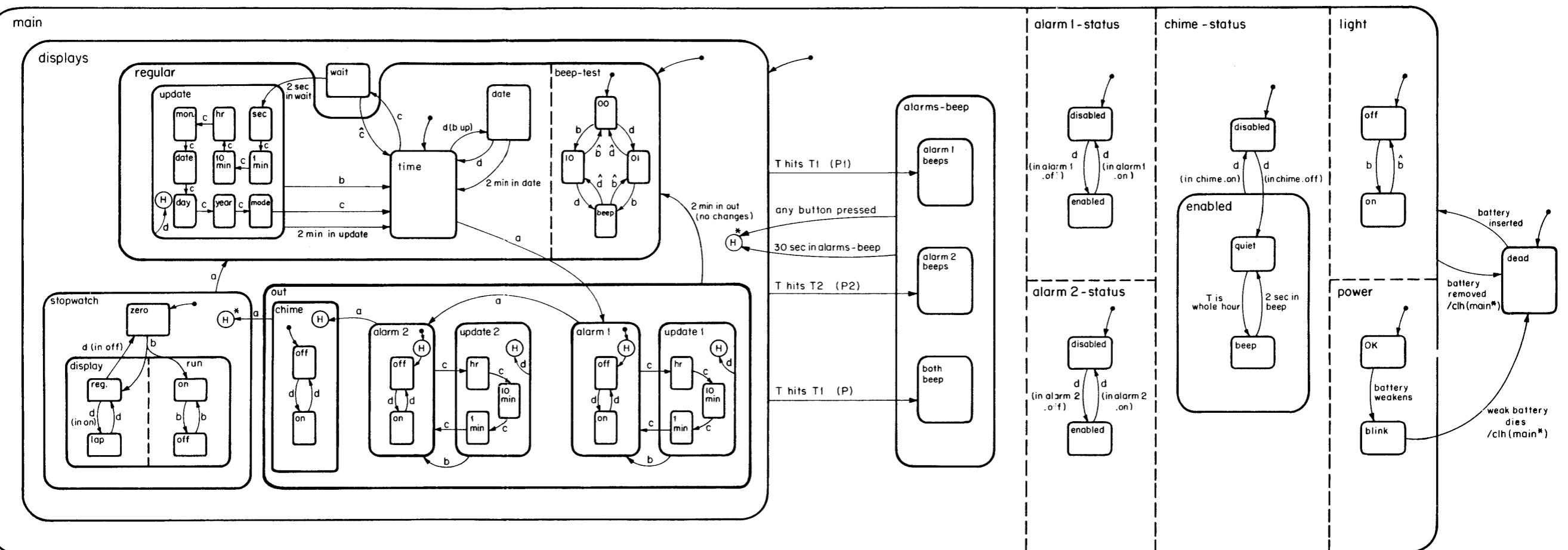


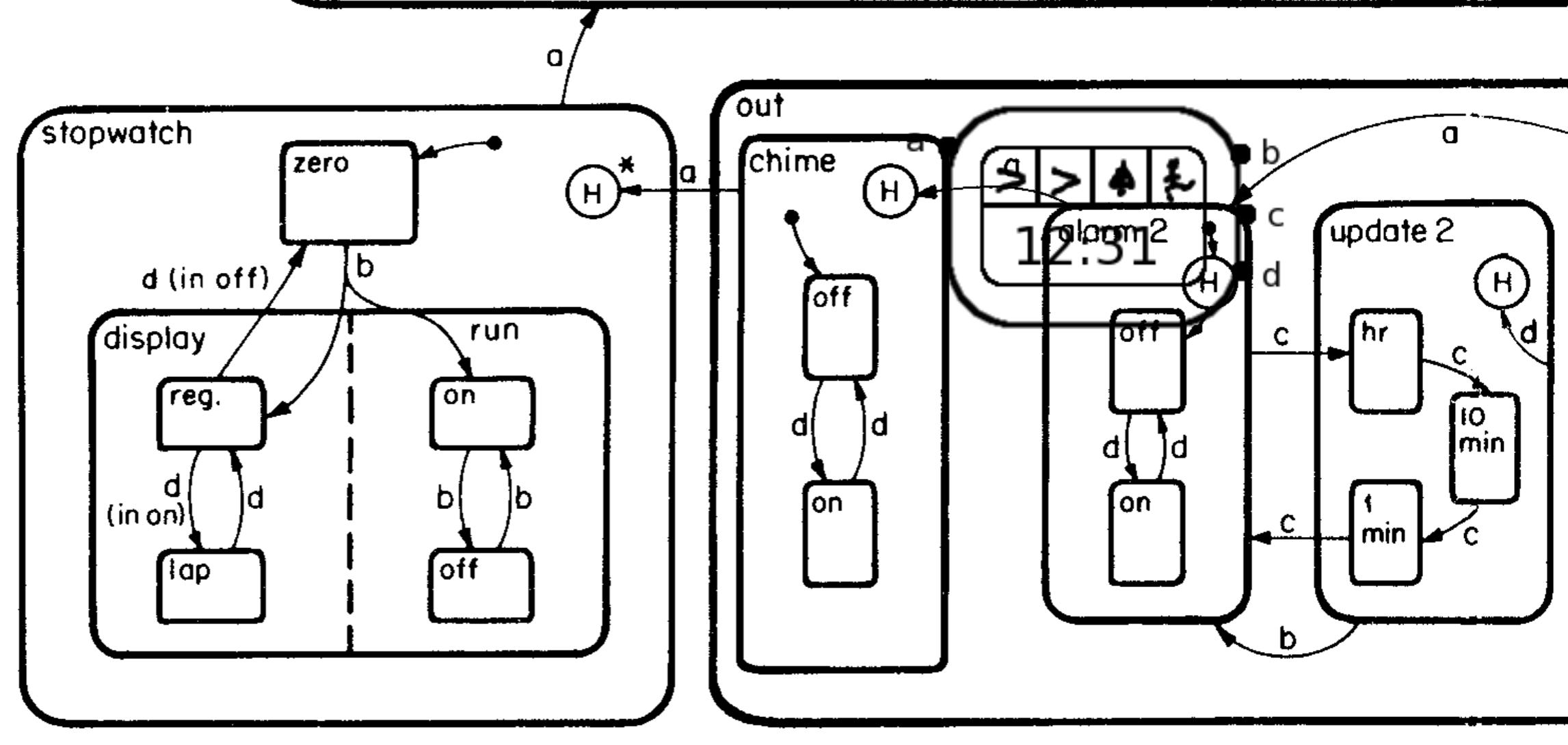
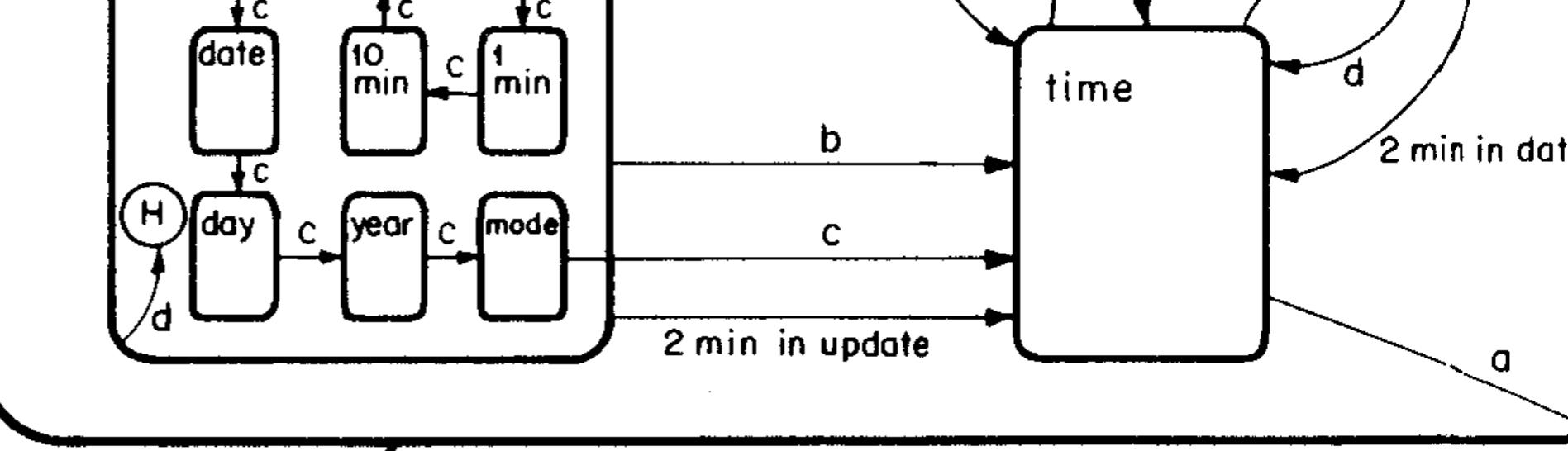
# Statecharts

[Harel, 1987]: “*Statecharts constitute a visual formalism for describing states and transitions in a modular fashion, enabling clustering, orthogonality (i.e. concurrency) and refinement...*”

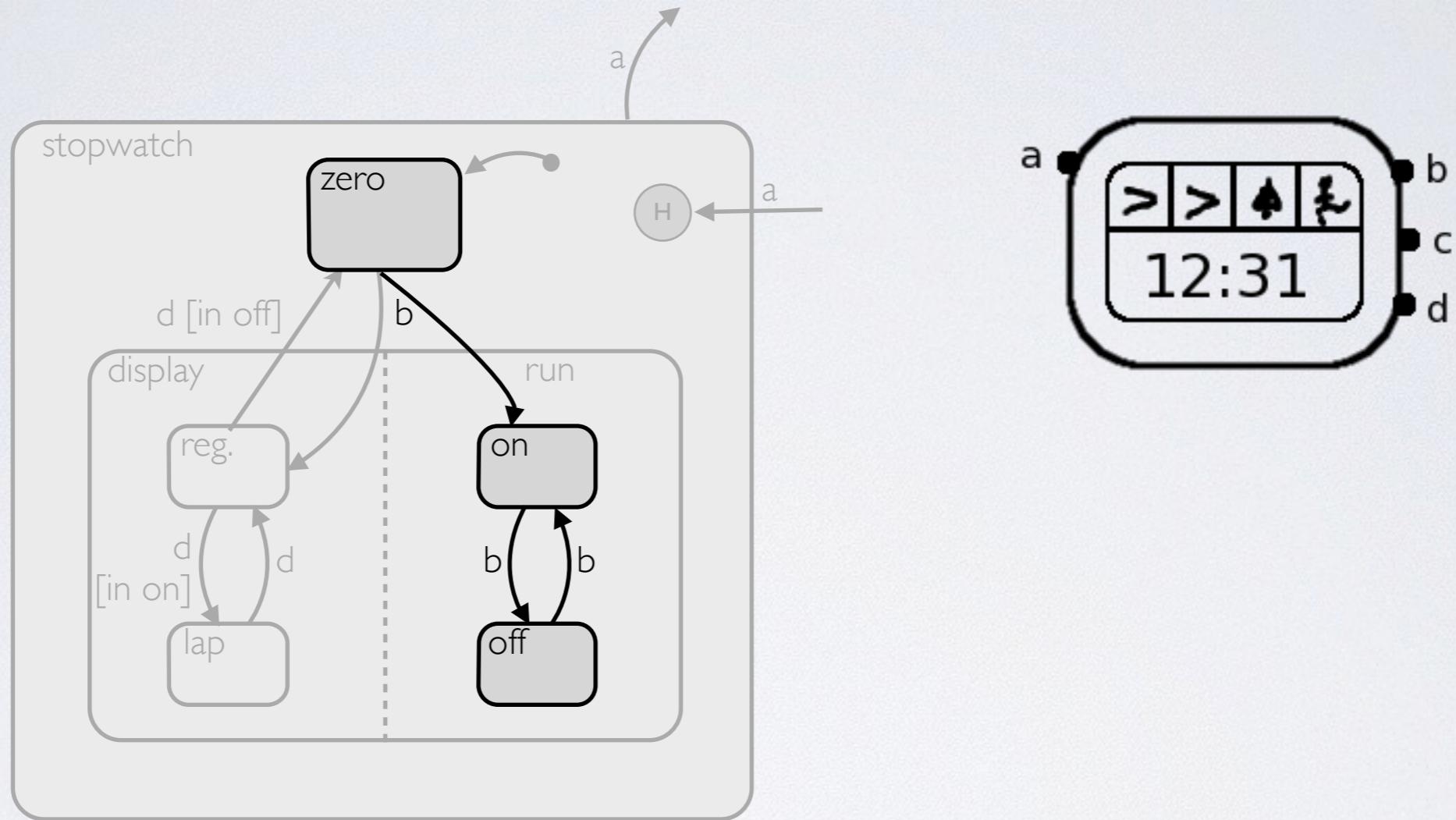


Citizen quartz multi-alarm

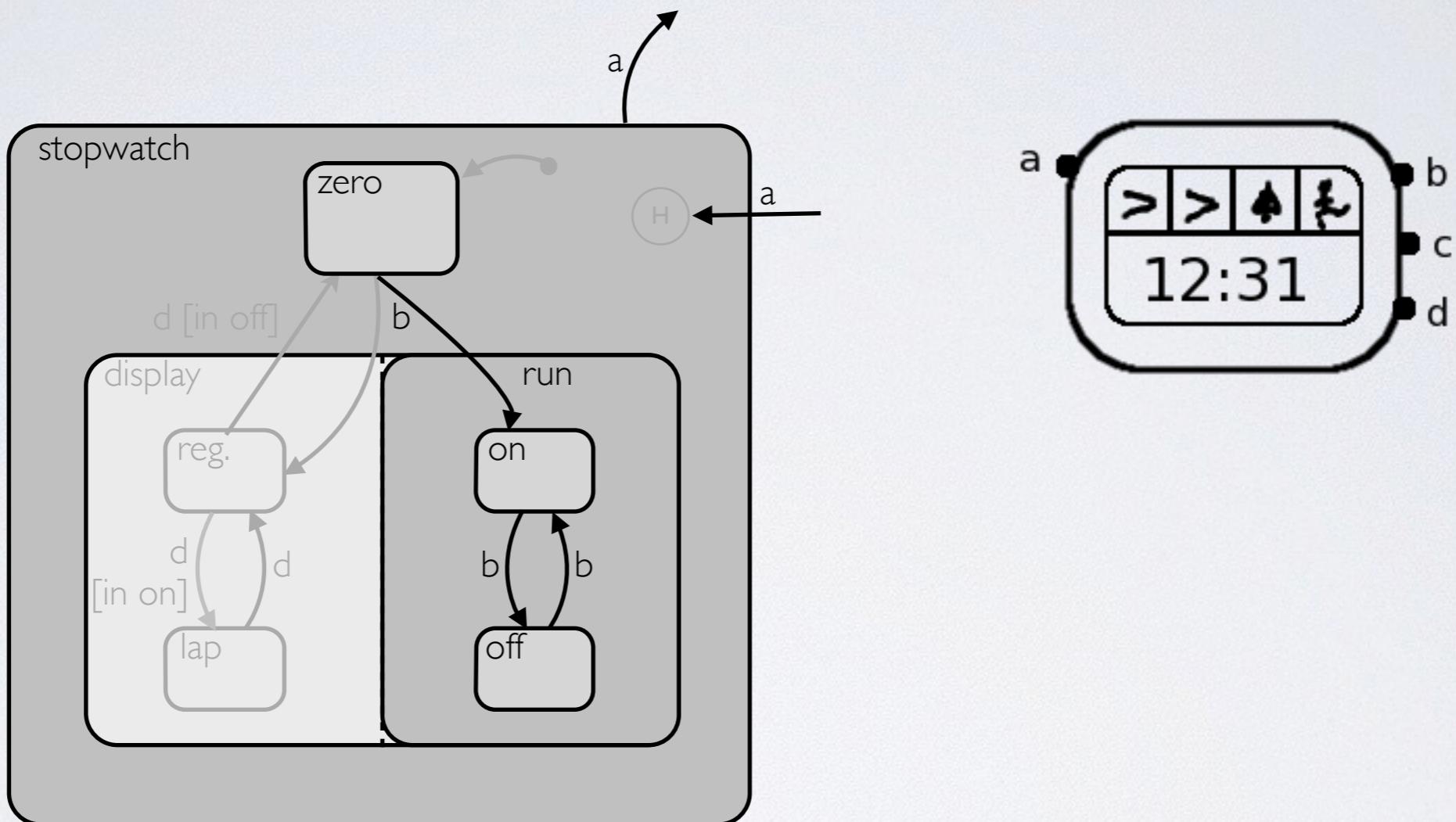




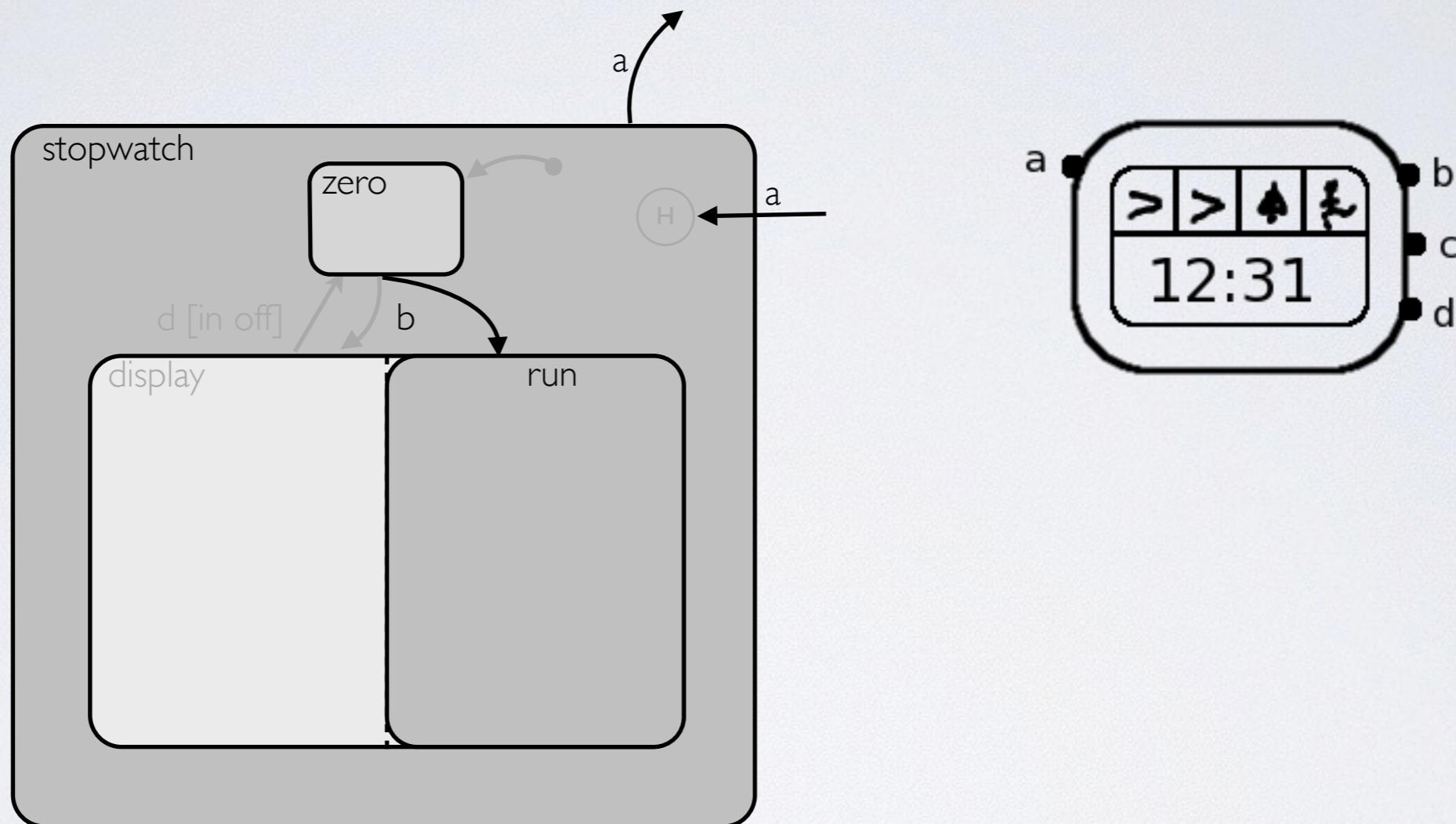
# Endliche Automaten / Finite State Machines



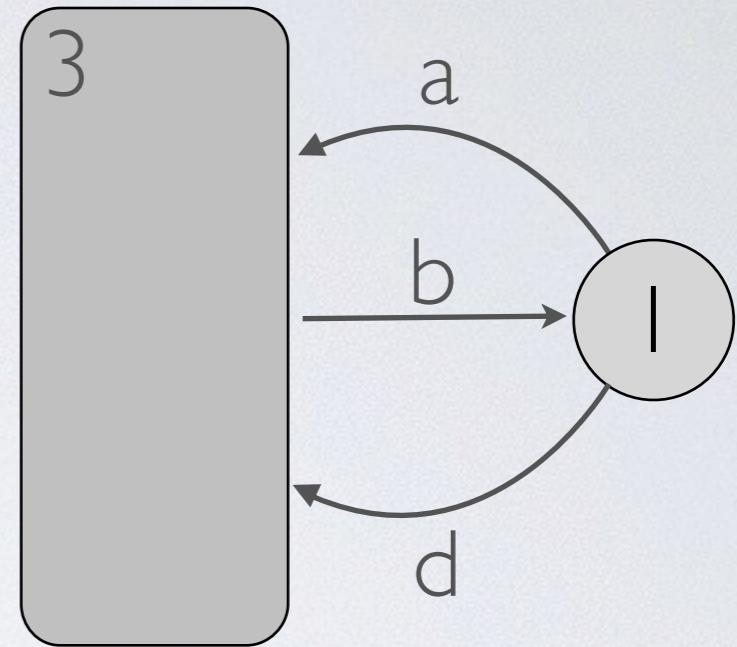
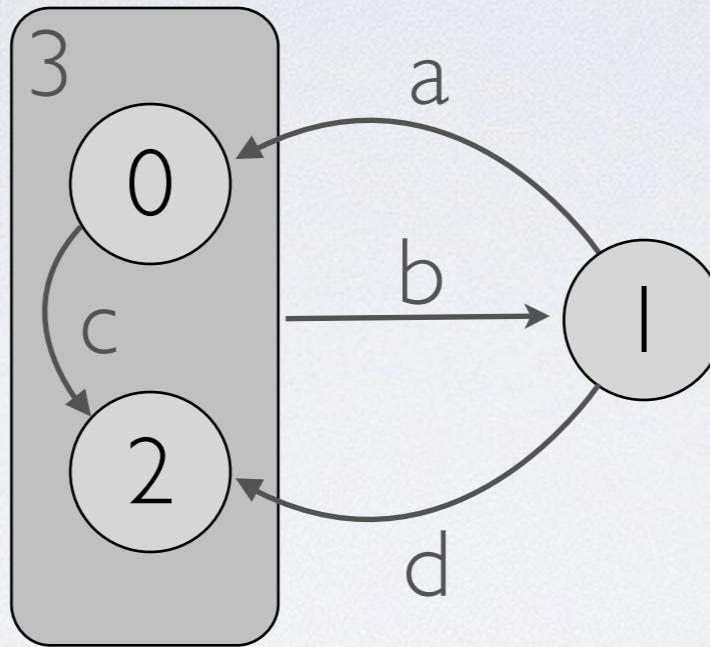
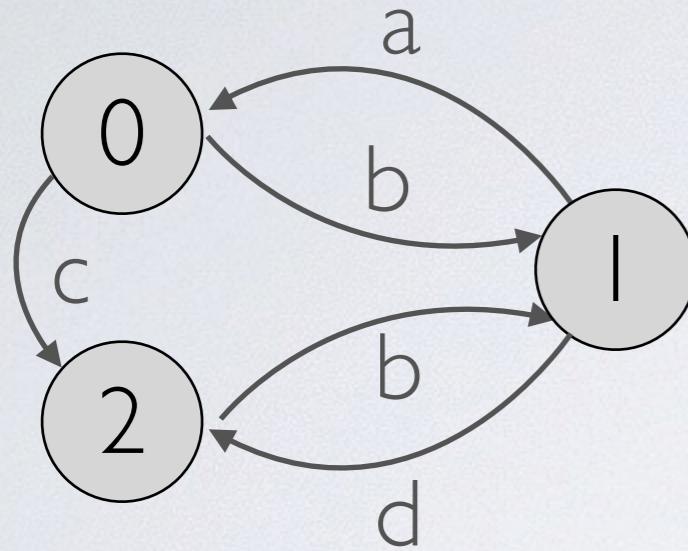
# Statecharts: Hierarchie



# Statecharts: Modularisierung

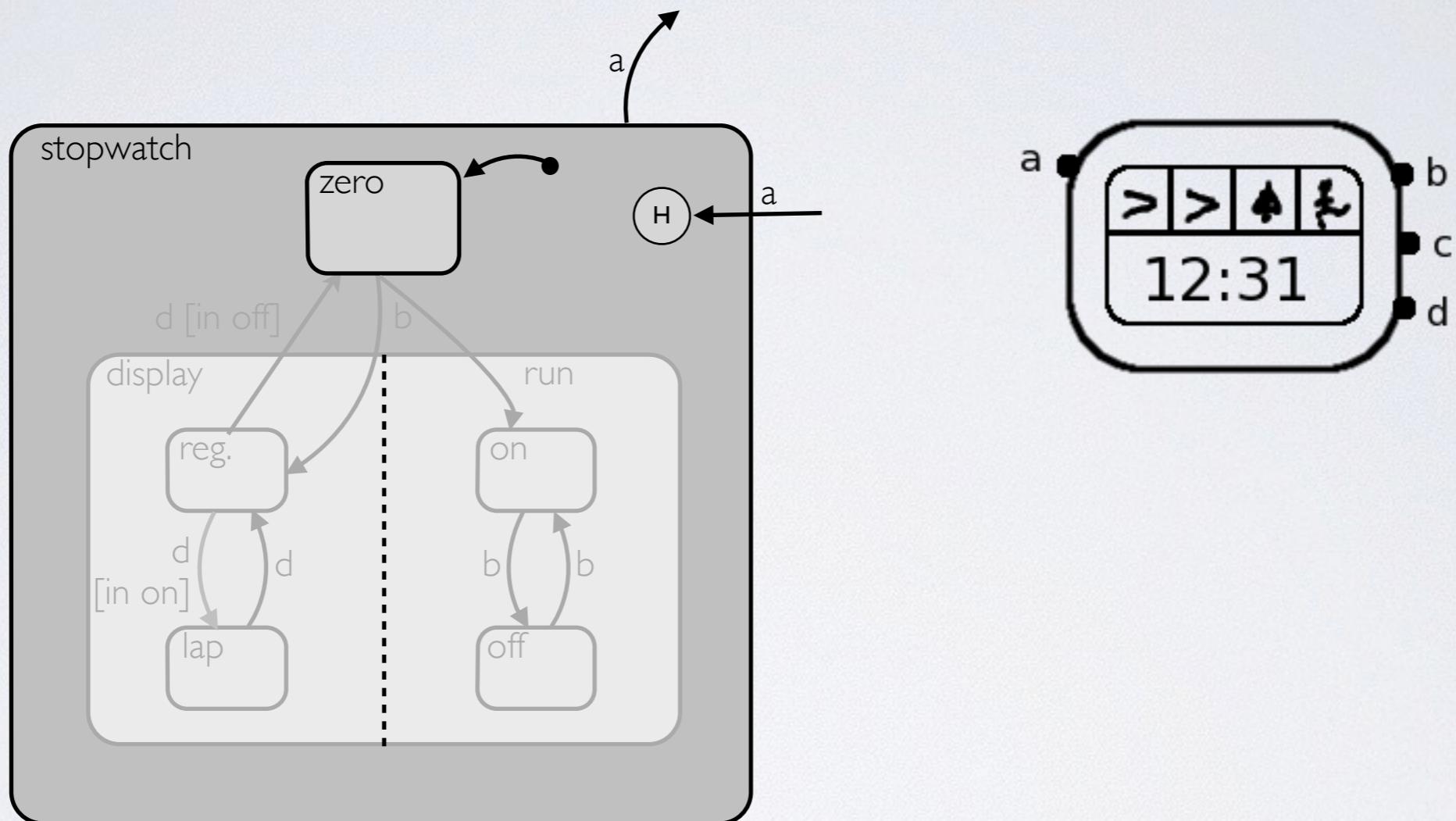


# Statecharts: Hierarchie / Modularisierung

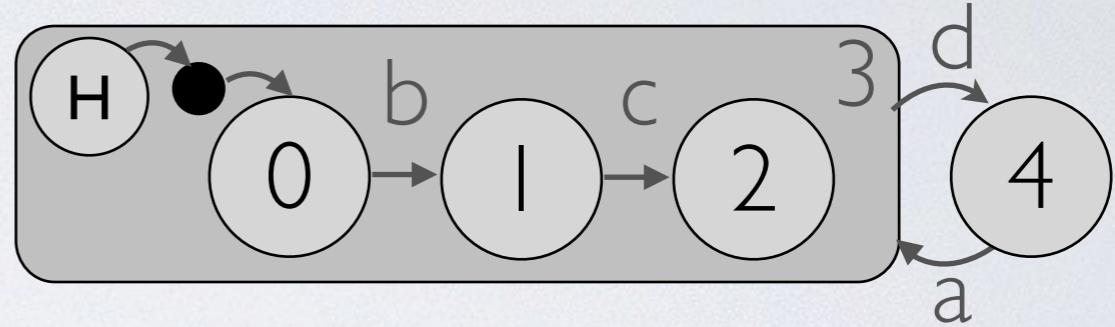
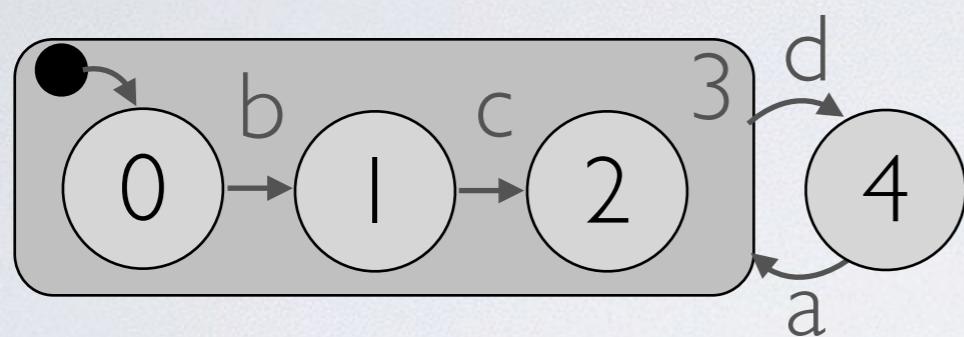


- Superzustände umfassen Unterzustände.
- Ein Oder-Superzustand hat nur einen aktiven Unterzustand.

# Statecharts: Defaults und History

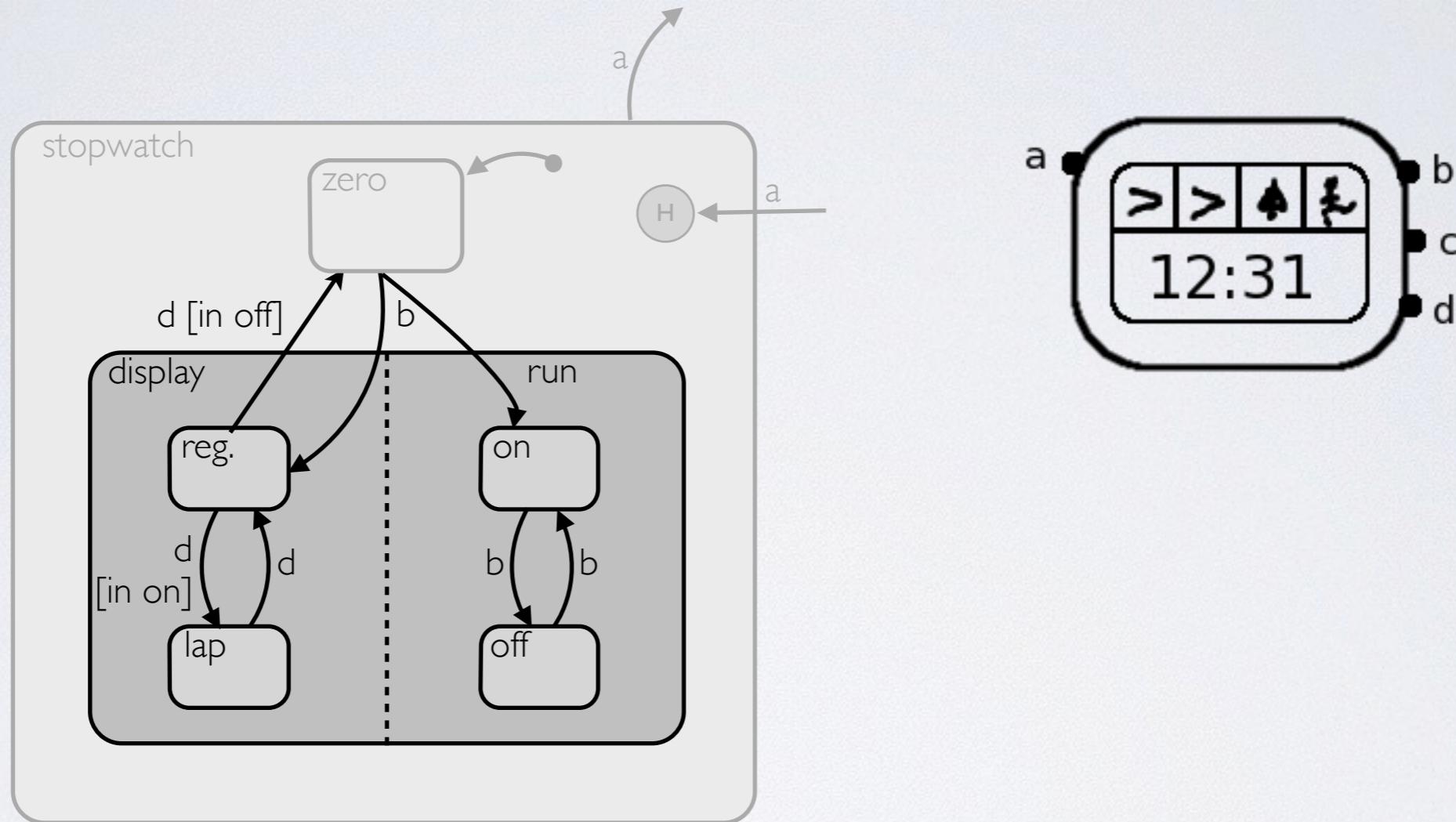


# Statecharts: Defaults und History

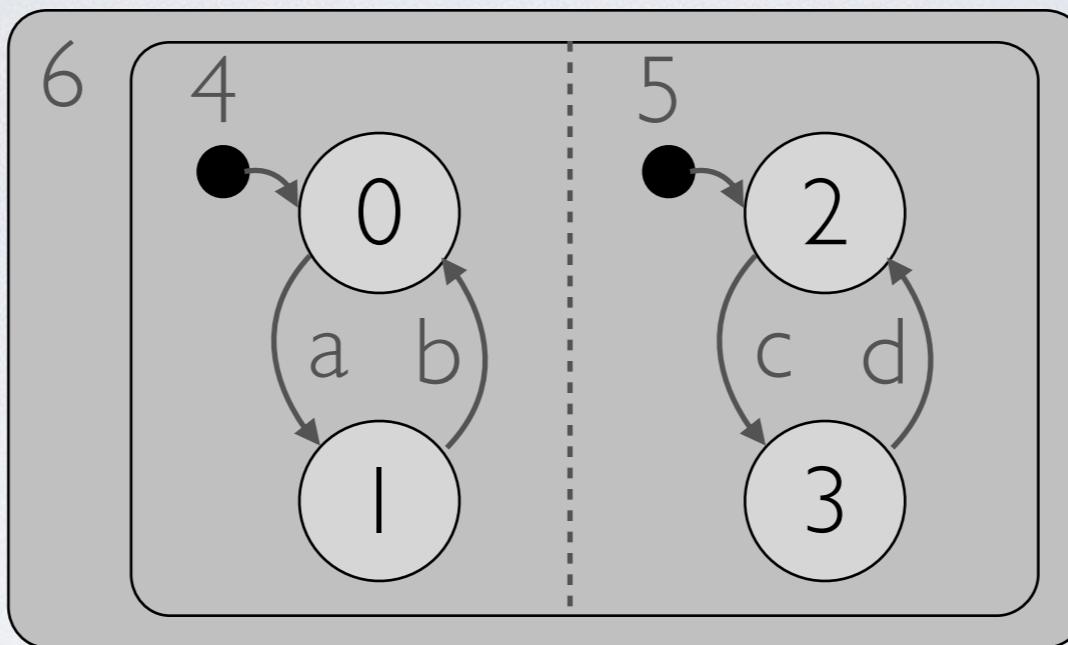


- Default-Zustand ● wird aktiv, wenn Superzustand aktiv wird.
- History-Merkmal (H) ermöglicht Rückkehr zum letzten aktiven Unterzustand, wenn der Superzustand aktiv wird.

# Statecharts: Nebenläufigkeit & Kanten

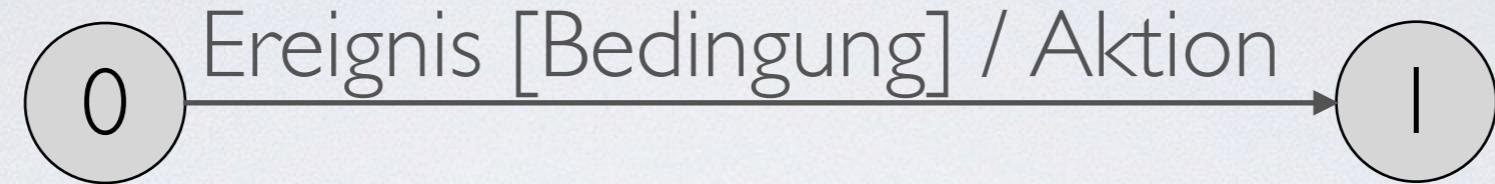


# Statecharts: Nebenläufigkeit



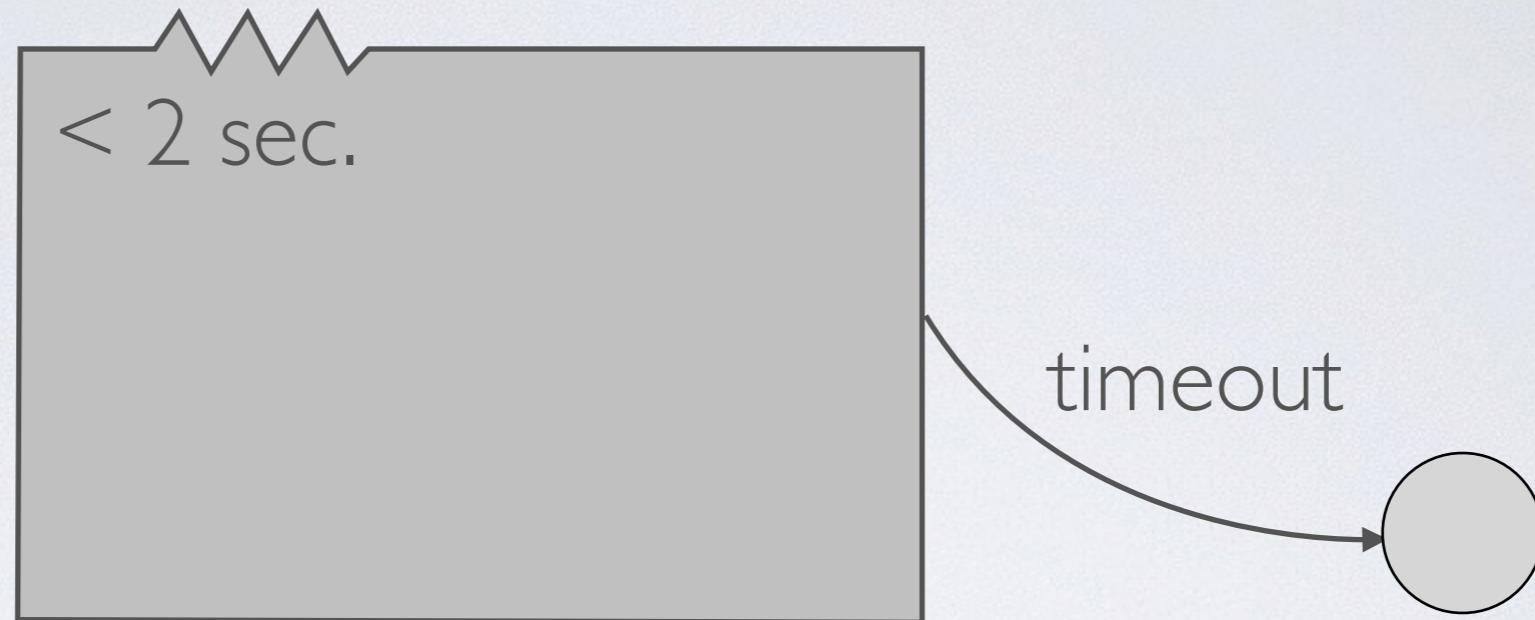
- Ein Und-Superzustand beinhaltet zwei oder mehr unabhängige Komponenten mit aktiven Unterzuständen.
- Und-Superzustände werden mit gestrichelter Line getrennt.

# Statecharts: Kantenbeschriftungen



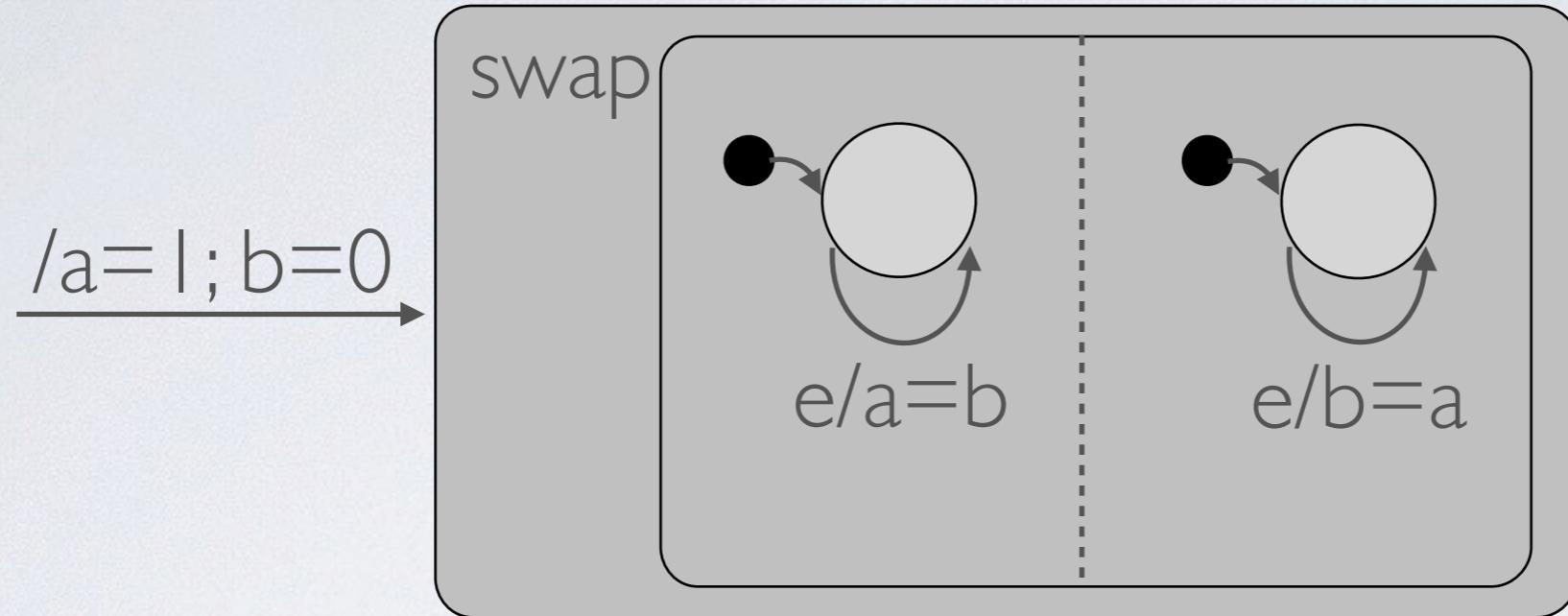
- Ereignisse aktivieren/deaktivieren Zustände.
- Bedingungen werden geprüft und ausgewertet.
- Aktionen weisen Variablen zu oder generieren Ereignisse.

# Statecharts: Timer



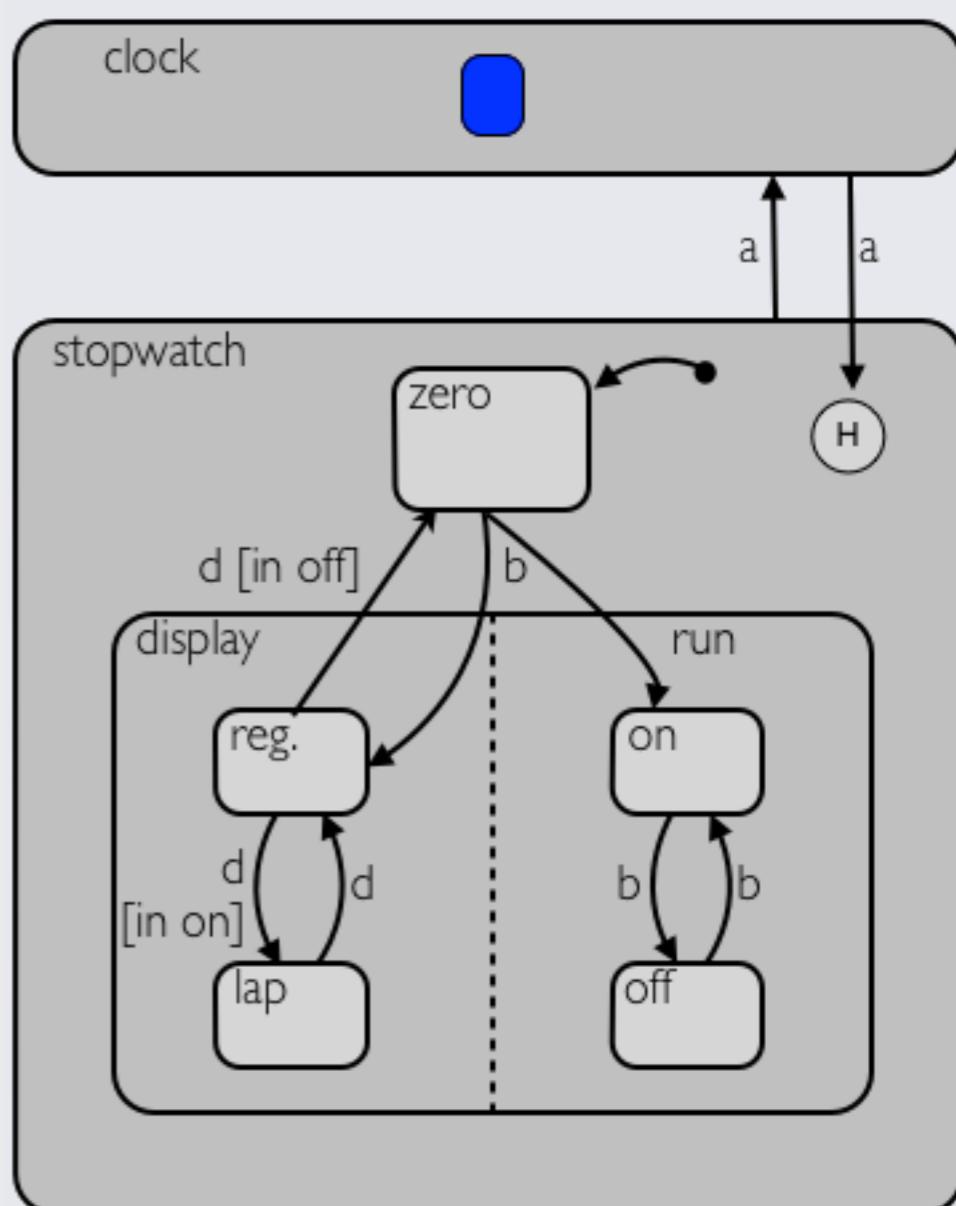
- Wenn kein Ereignis eintritt, geht der Zustand nach Ablauf einer vorgegebenen Zeit in spezifizierten Zustand über.

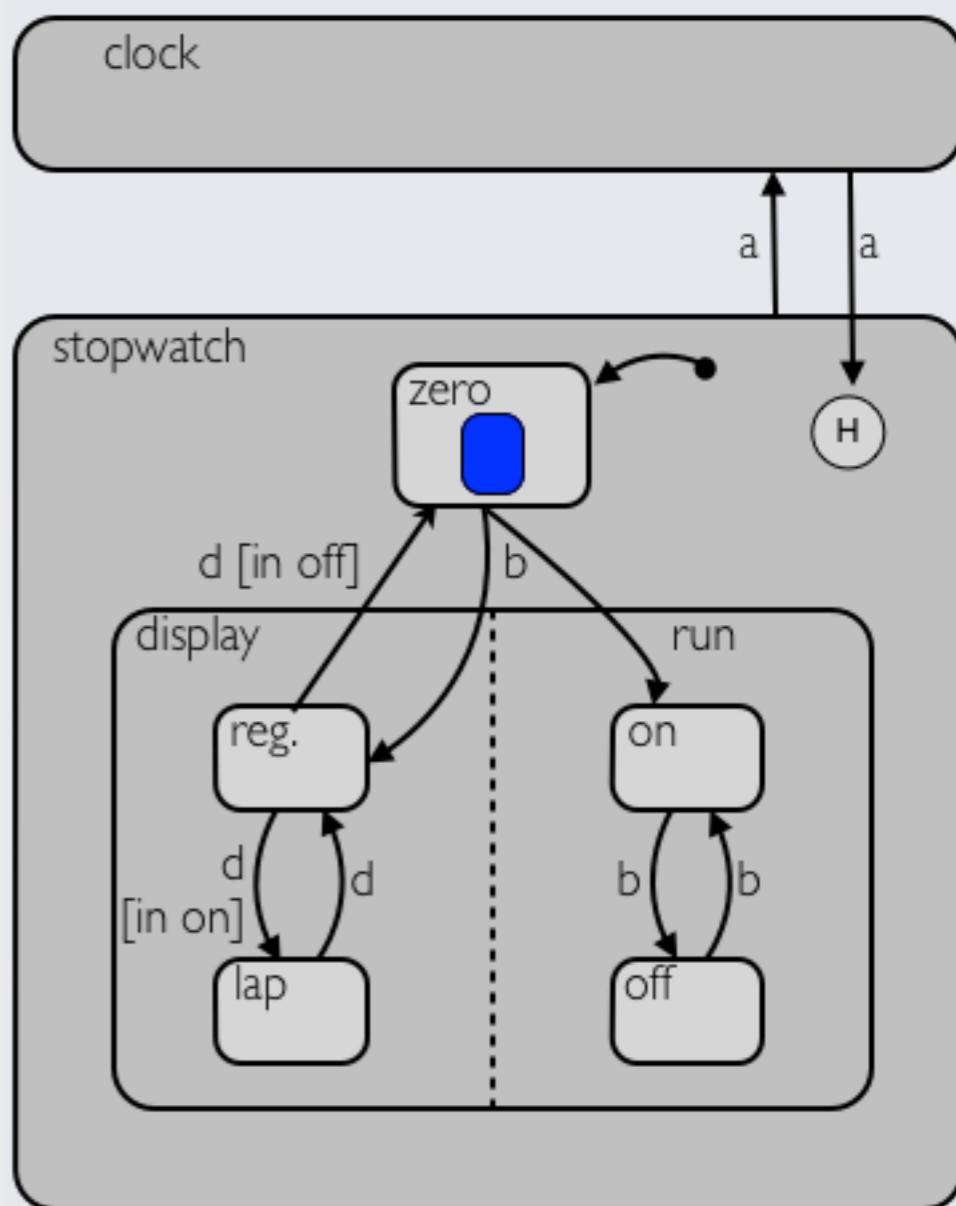
# Statecharts: StateMate Semantics

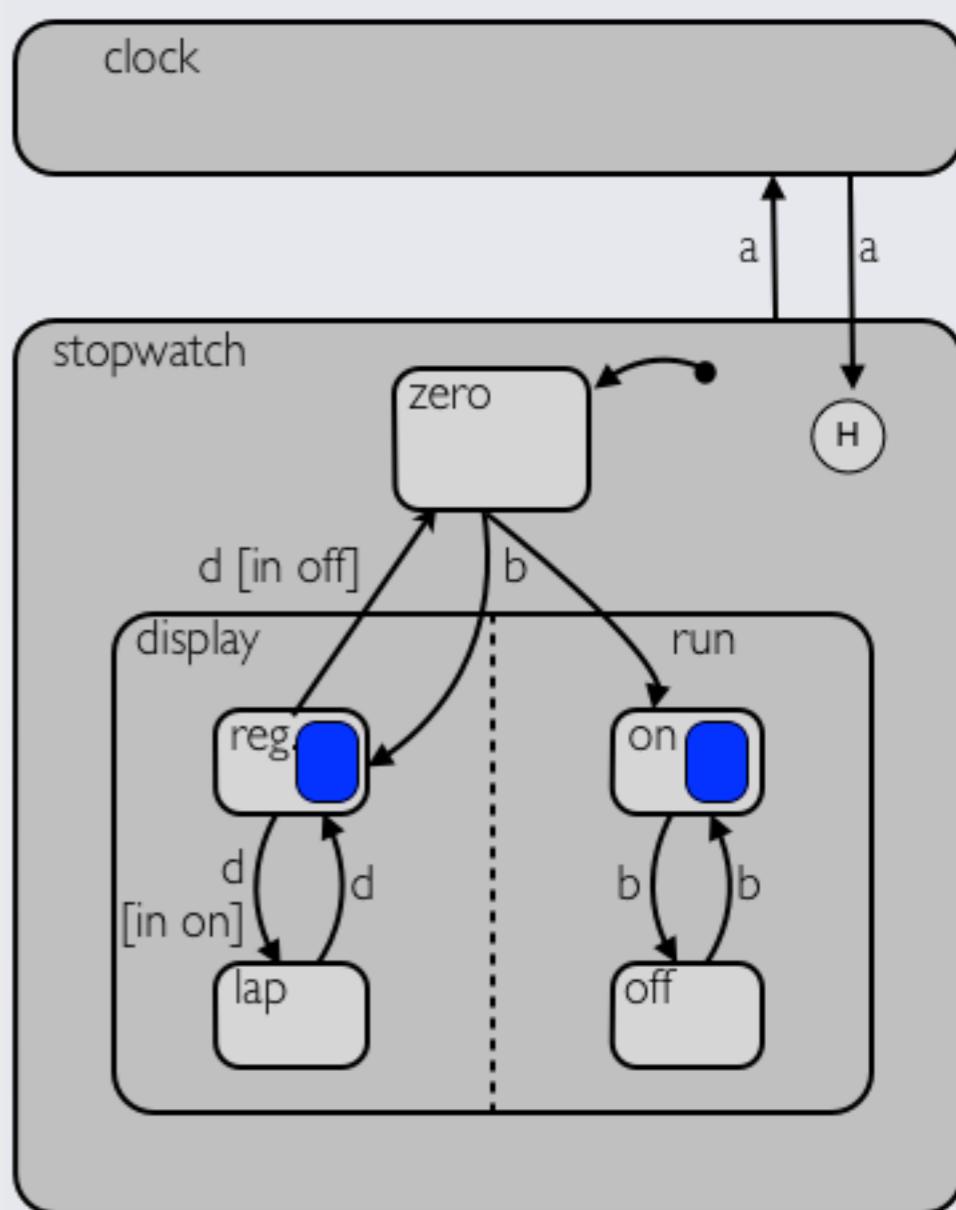


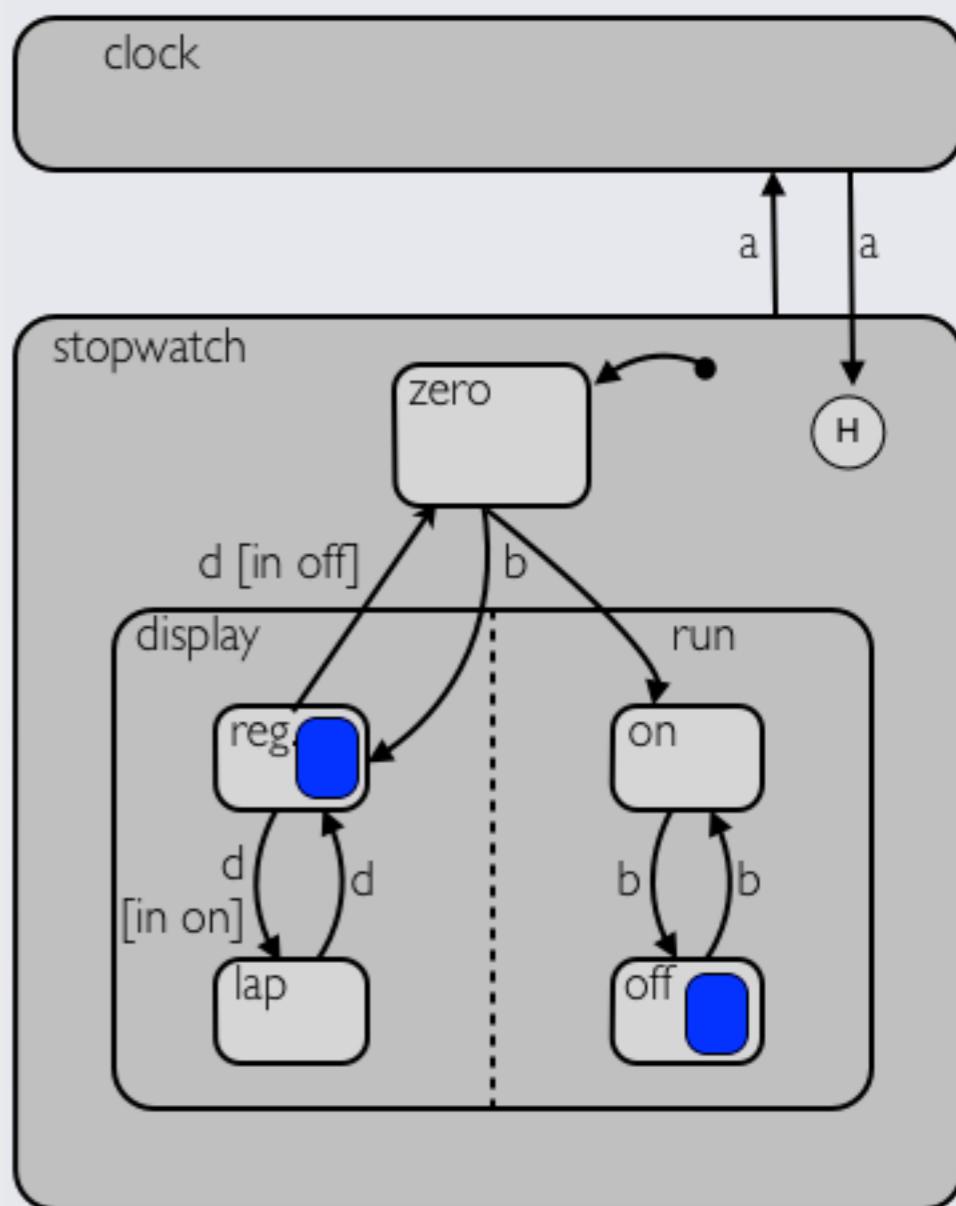
1. Auswertung von Ereignisse und Bedingungen.
  2. Berechnung von Zuweisungen (temporär Speicherung).
  3. Übergang in neue Zustände und Zuweisung zu Variablen.
- Erreicht Broadcast-Mechanismus.

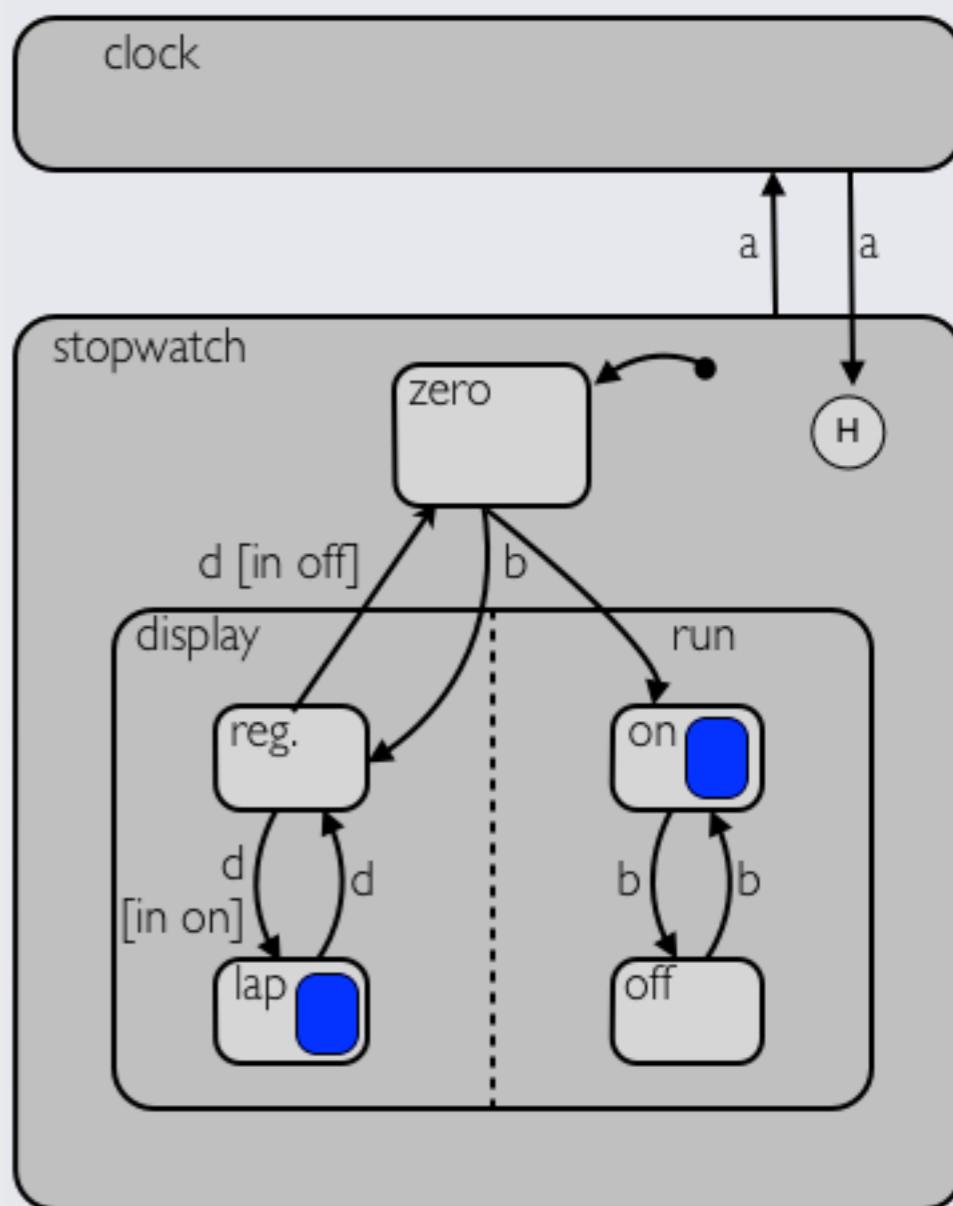
Demo

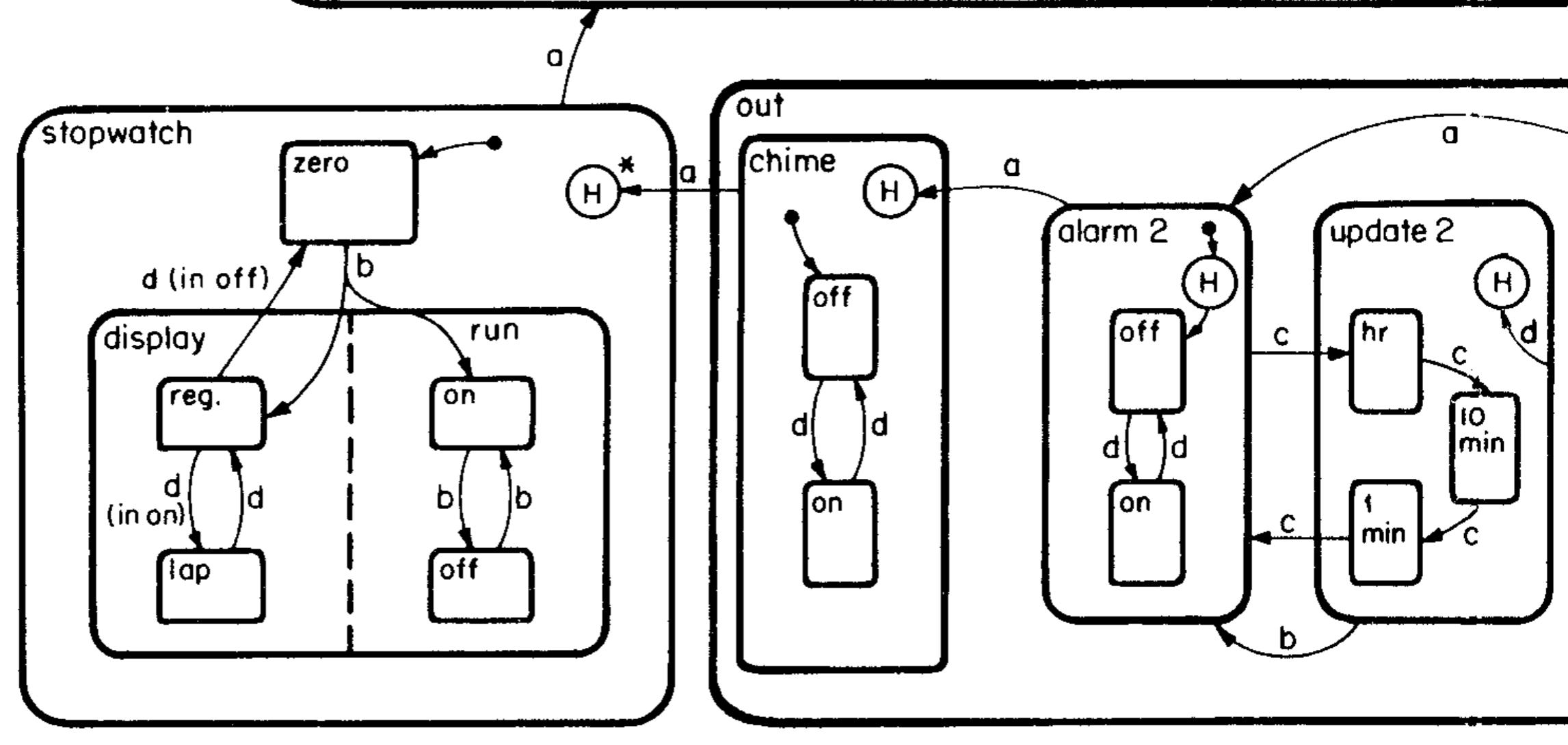
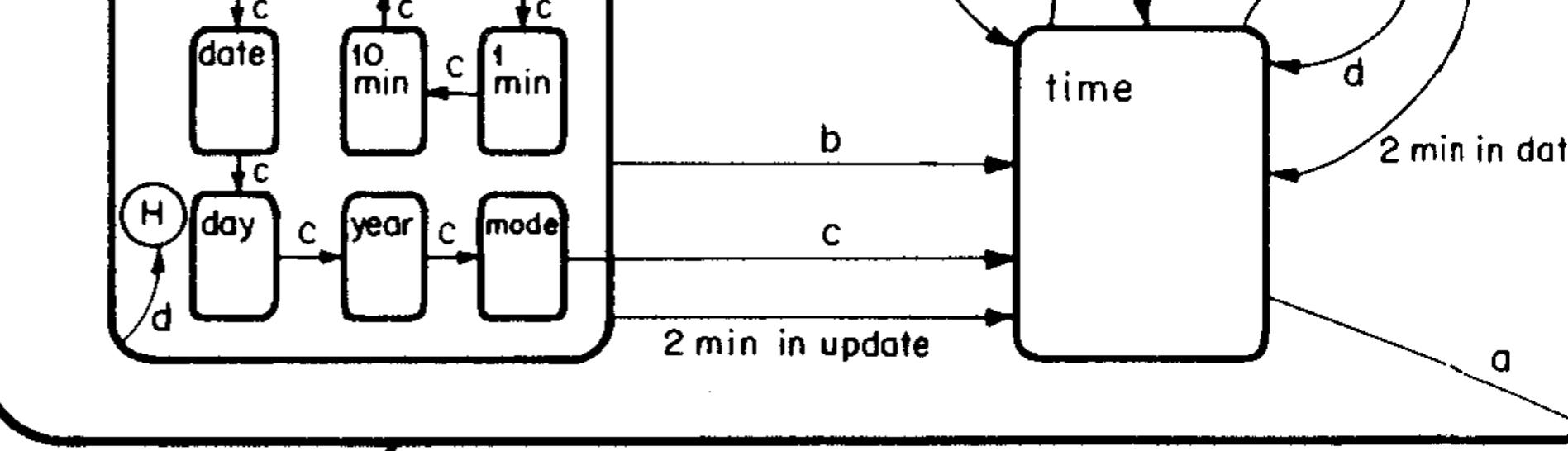












main

displays

regular

update

2 sec  
in wait

wait

c

time

b

c

2 min in update

date

d(b up)

d

2 min in date

beep-test

2 min in c

(no char)

stopwatch

out

a

chime

alarm 2

a

a

c

c

b

alarm 1

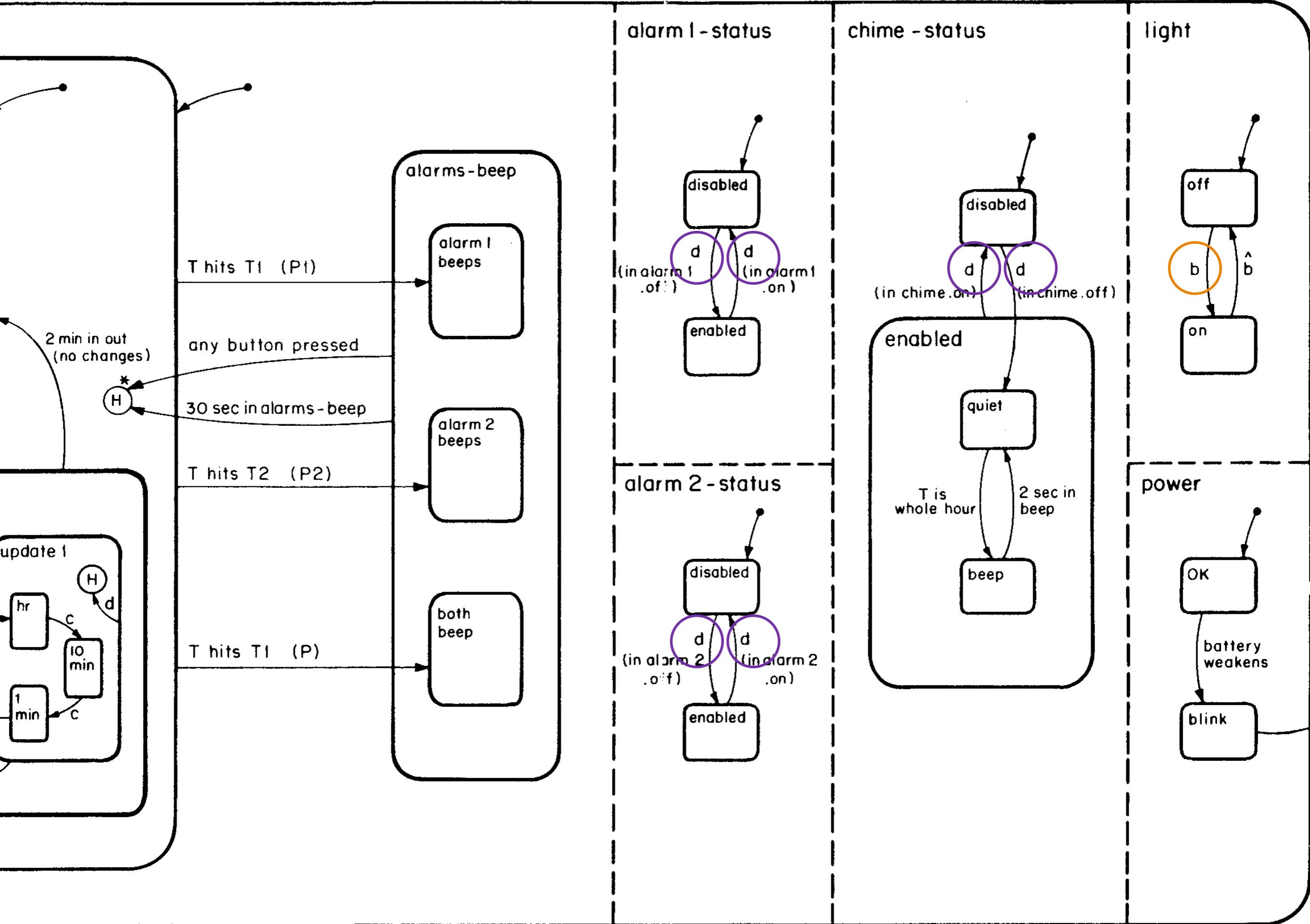
a

update 1

c

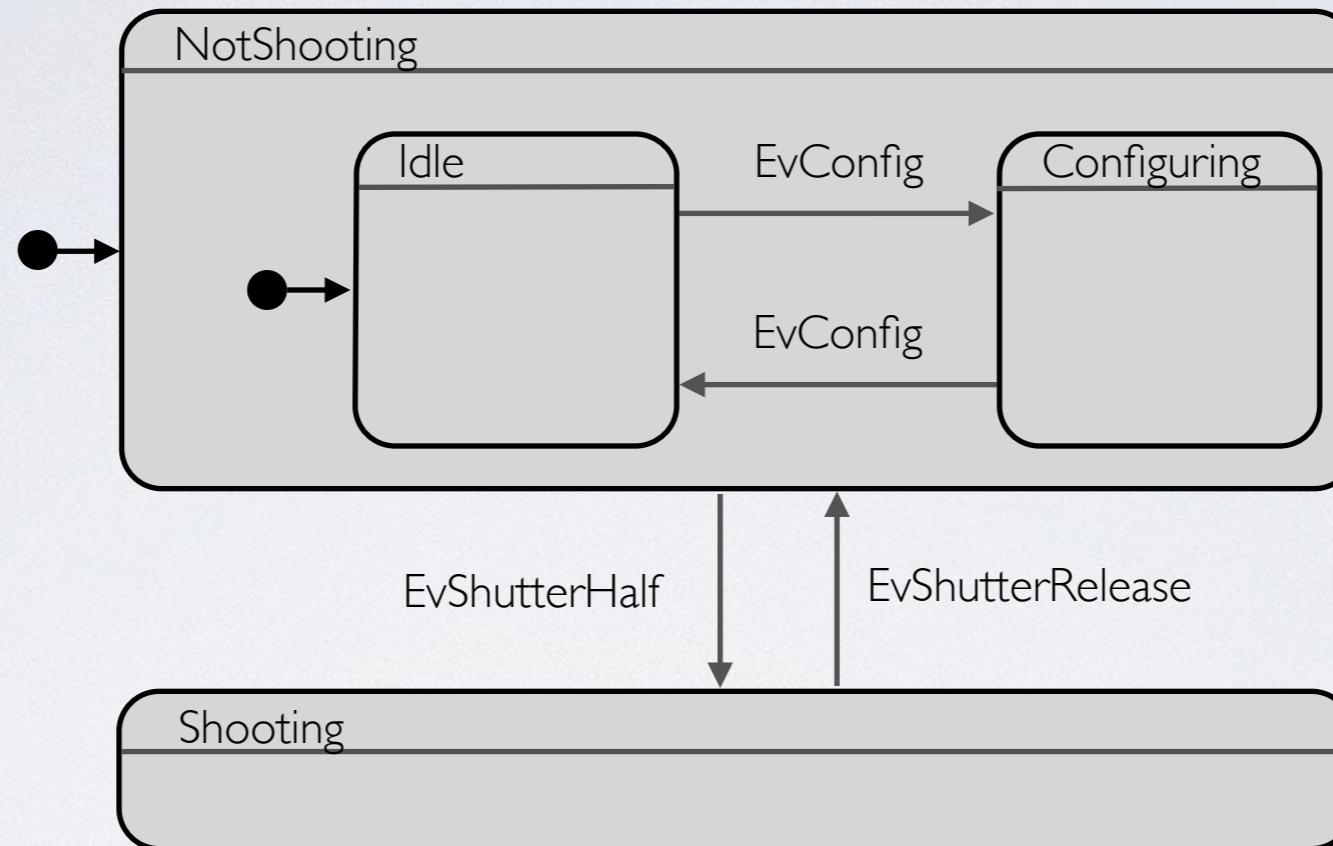
c

b

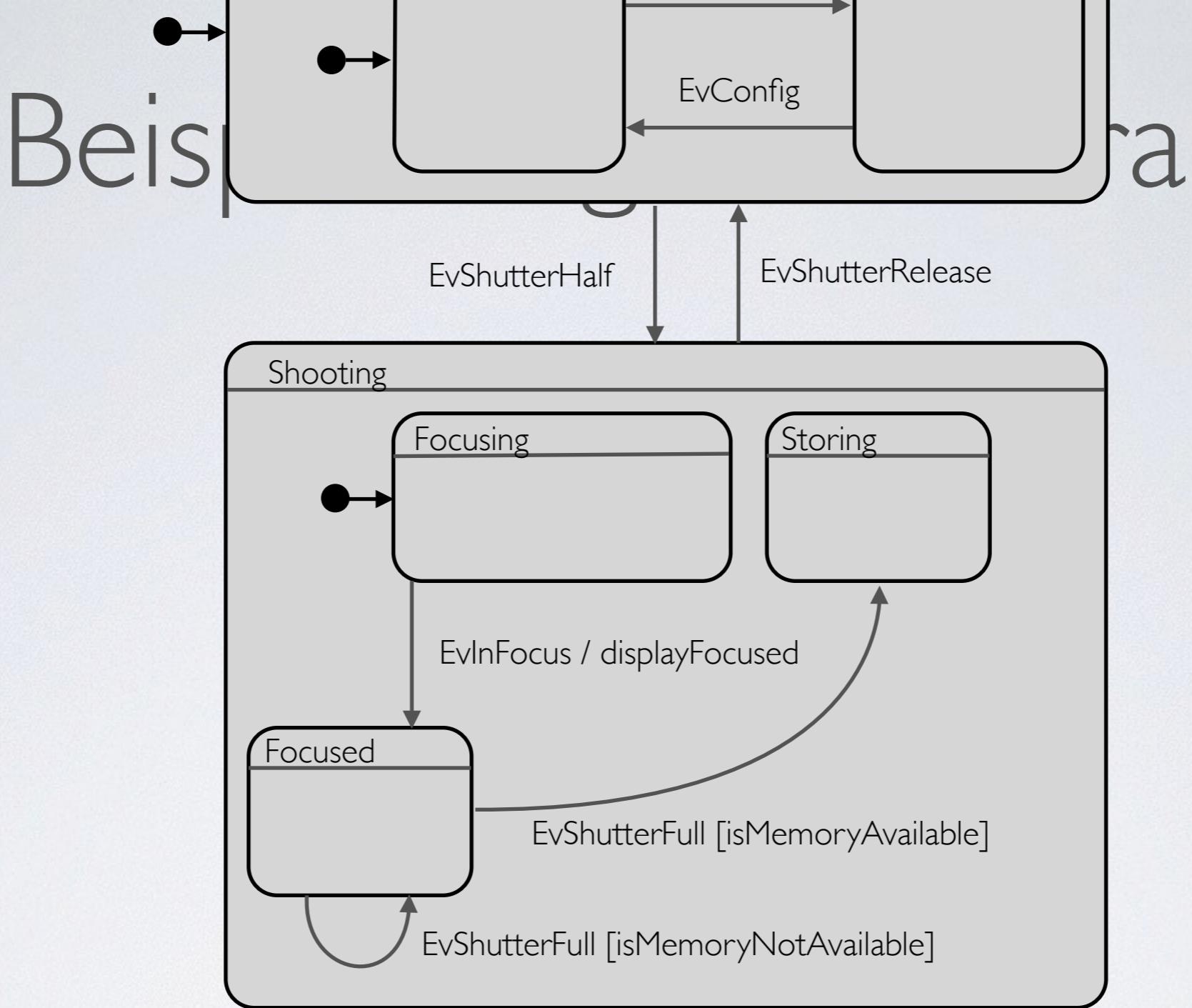


Beispiel - Modell einer Kamera

# Beispiel: Digitalkamera

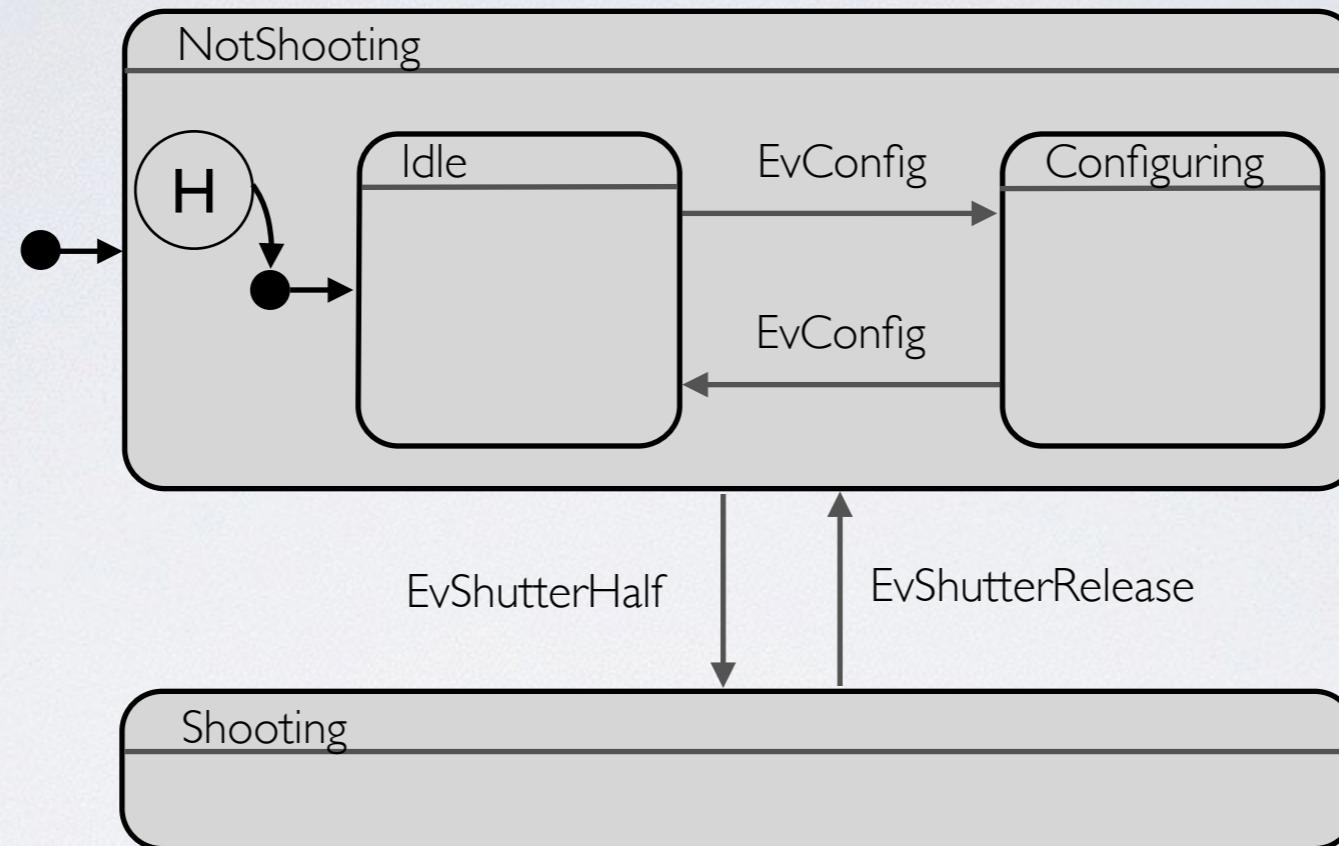


- Shutter-Knopf: halb- oder voll gedrückt
  - halbgedrückt aktiviert Aufnahmezustand (*Shooting*)
- Config-Knopf



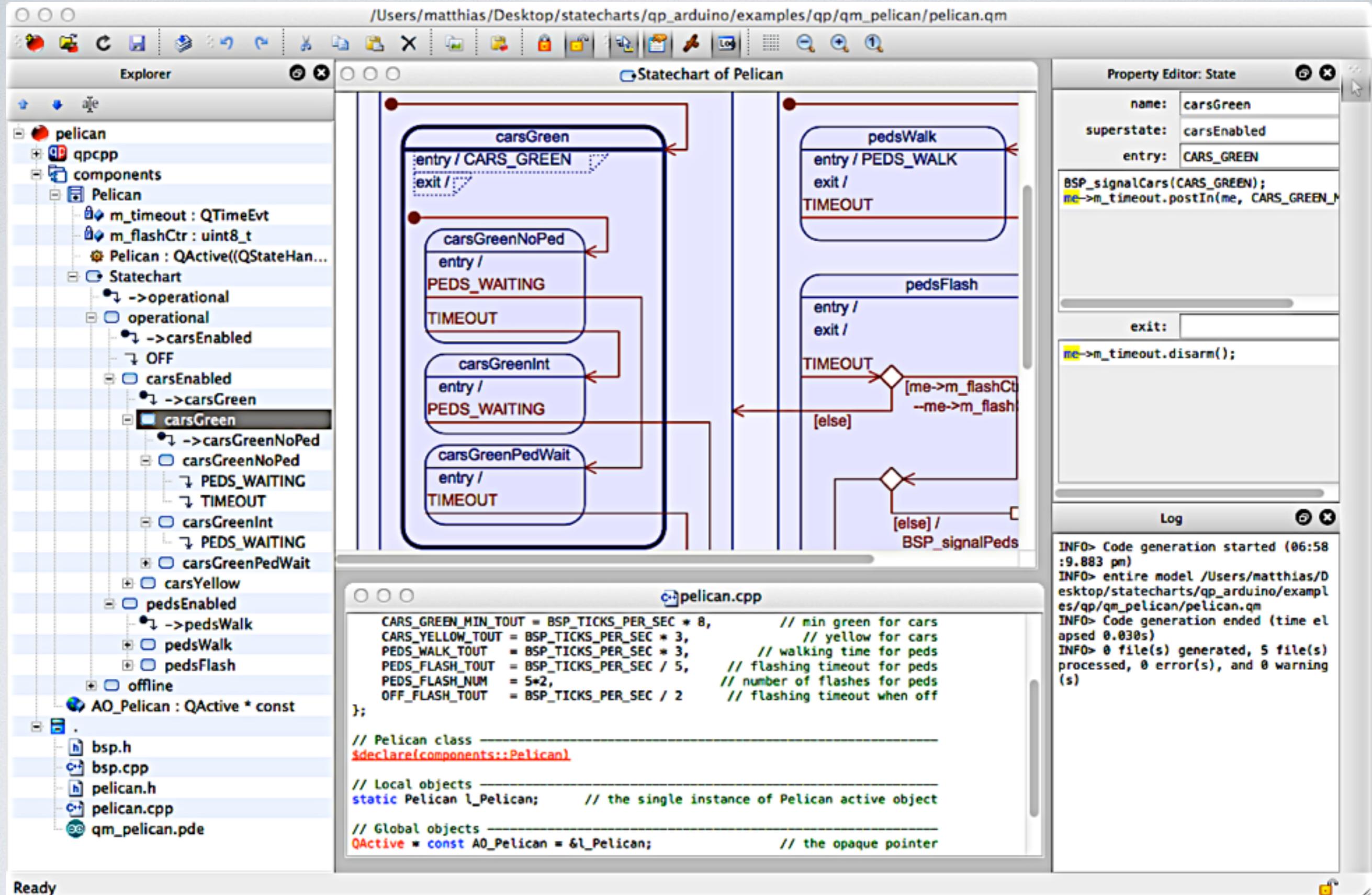
- *Focusing*
  - Fokussieren eines Ziels
- *Focused*
  - Aufnahme eines Bildes bei genug Speicher

# Beispiel: Digitalkamera



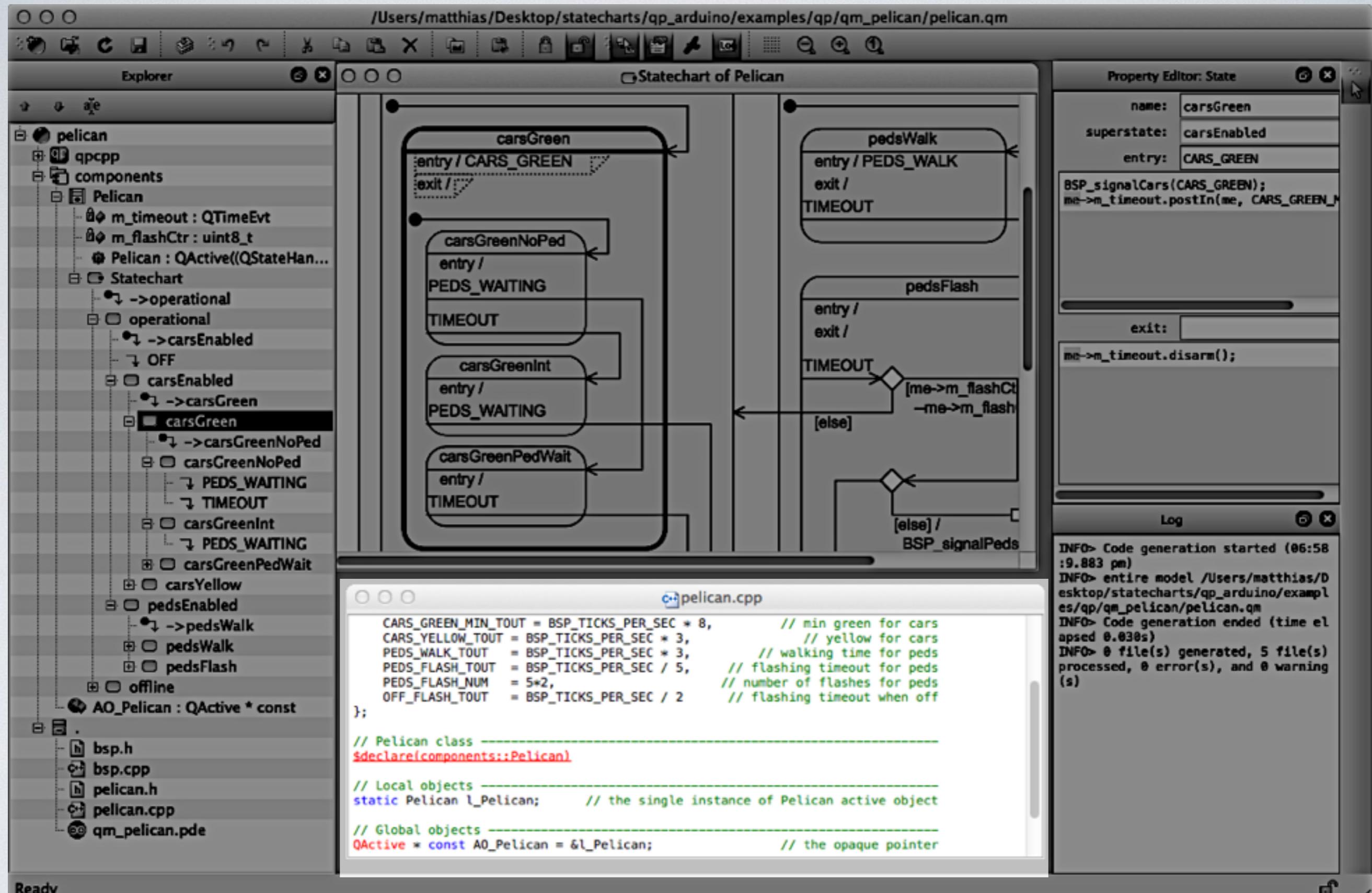
- *EvShutterHalf* in *Configure* bzw. *Idle* Zustand ausgelöst?
- Dann Rückkehr in diesen Zustand
- History-Mechanismus

# Beispiel - Tool für Statecharts



Ready

# Beispiel: Modellierung und Codegenerierung mit QM



# Beispiel: Modellierung und Codegenerierung mit QM

pelican.cpp

```

CARS_GREEN_MIN_TOUT = BSP_TICKS_PER_SEC * 8,           // min green for cars
CARS_YELLOW_TOUT = BSP_TICKS_PER_SEC * 3,             // yellow for cars
PEDS_WALK_TOUT = BSP_TICKS_PER_SEC * 3,               // walking time for peds
PEDS_FLASH_TOUT = BSP_TICKS_PER_SEC / 5,              // flashing timeout for peds
PEDS_FLASH_NUM = 5*2,                                // number of flashes for peds
OFF_FLASH_TOUT = BSP_TICKS_PER_SEC / 2                // flashing timeout when off
};

// Pelican class --
// $(components::Pelican) .....
/// PEdestrian LIght CONtrolled (PELICAN) crossing
class Pelican : public QActive {
private:
    QTimeEvt m_timeout;
    uint8_t m_flashCtr;

public:
    /// constructor
    Pelican() : QActive((QStateHandler)&Pelican::initial), m_timeout(TIMEOUT_SIG)
    {
    }

protected:
    static QState initial(Pelican *me, QEvent const *e);
    static QState operational(Pelican *me, QEvent const *e);
    static QState carsEnabled(Pelican *me, QEvent const *e);
    static QState carsGreen(Pelican *me, QEvent const *e);
    static QState carsGreenNoPed(Pelican *me, QEvent const *e);
    static QState carsGreenInt(Pelican *me, QEvent const *e);
    static QState carsGreenPedWait(Pelican *me, QEvent const *e);
    static QState carsYellow(Pelican *me, QEvent const *e);
    static QState pedsEnabled(Pelican *me, QEvent const *e);
    static QState pedsWalk(Pelican *me, QEvent const *e);
    static QState pedsFlash(Pelican *me, QEvent const *e);
    static QState offline(Pelican *me, QEvent const *e);
};

// Local objects --
static Pelican l_Pelican;      // the single instance of Pelican active object

// Global objects --
QActive * const A0_Pelican = &l_Pelican;                  // the opaque pointer
...

```

pelican.cpp

```

CARS_GREEN_MIN_TOUT = BSP_TICKS_PER_SEC * 8,           // min green for cars
CARS_YELLOW_TOUT = BSP_TICKS_PER_SEC * 3,             // yellow for cars
PEDS_WALK_TOUT = BSP_TICKS_PER_SEC * 3,               // walking time for peds
PEDS_FLASH_TOUT = BSP_TICKS_PER_SEC / 5,              // flashing timeout for peds
PEDS_FLASH_NUM = 5*2,                                // number of flashes for peds
OFF_FLASH_TOUT = BSP_TICKS_PER_SEC / 2                // flashing timeout when off
};

// Pelican class --
// declare[components::Pelican]

// Local objects --
static Pelican l_Pelican;      // the single instance of Pelican active object

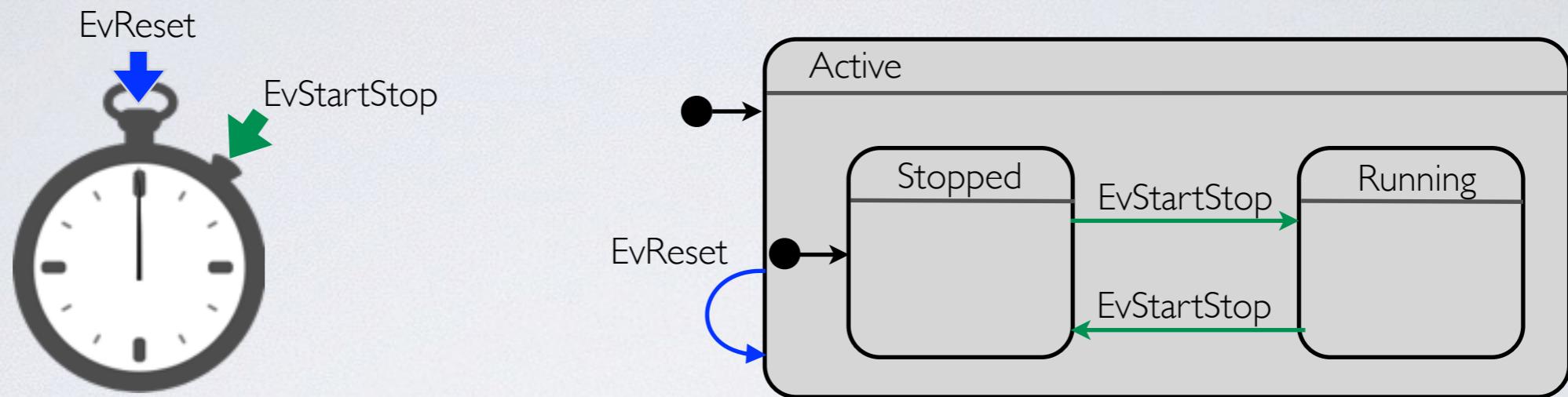
// Global objects --
QActive * const A0_Pelican = &l_Pelican;                  // the opaque pointer

```

# Beispiel: Modellierung und Codegenerierung mit QM

# Beispiel - Programmierbibliothek

# Implementierungsbispiel: Stopp Uhr



Stopp Uhr modelliert durch die Zustände:

- *Stopped*: Anhalten der Zeit.
  - Reset: Uhr zurück in 0 Position. Bleibt im *Stopped* Zustand.
  - Start/Stopp: Übergang in *Running* Zustand.
- *Running*: Zeit läuft und wird gezeigt.
  - Reset: Uhr zurück in 0 Position. Übergang in *Stopped* Zustand.
  - Start/Stopp: Übergang in *Stopped* Zustand.

# Beispiel: Stopp Uhr mit C++ & Boost Bibliothek

```
// setup
#include <boost/statechart/event.hpp>
#include <boost/statechart/state_machine.hpp>
#include <boost/statechart/simple_state.hpp>

using namespace boost::statechart;

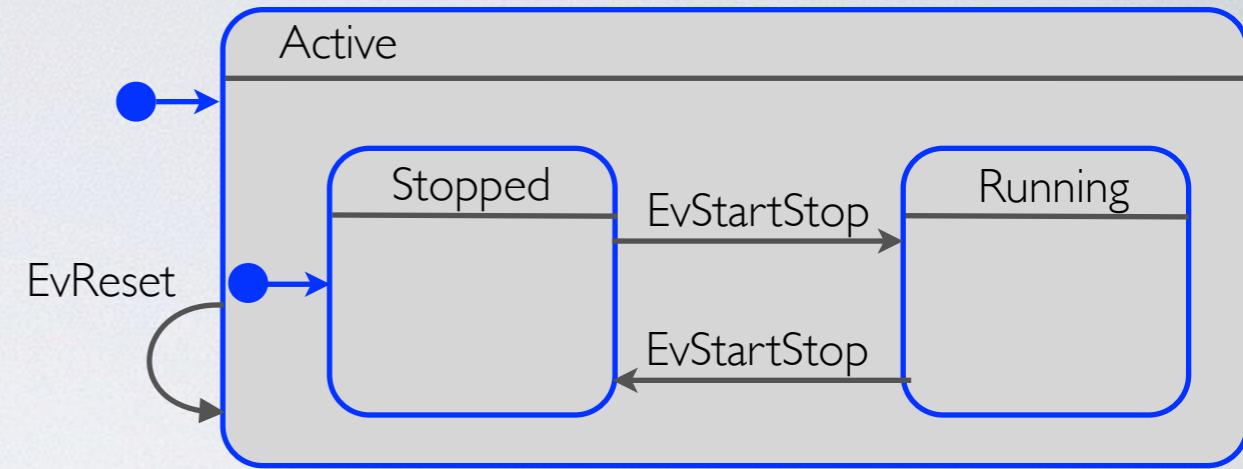
// event<most_derived_class>
struct EvStartStop : event<EvStartStop> {};
struct EvReset : event<EvReset> {};

// forward declaration
struct Active;
struct Stopped;

// state_machine<most_derived_class, initial_state>
struct Stopwatch : state_machine<Stopwatch, Active> {};

// States
// simple_state<most_derived_class, context, initial_state>
struct Active : simple_state<Active, Stopwatch, Stopped> {};
struct Running : simple_state<Running, Active> {};
struct Stopped : simple_state<Stopped, Active> {};

int main() {
    Stopwatch myWatch;
    myWatch.initiate();
    return 0;
}
```



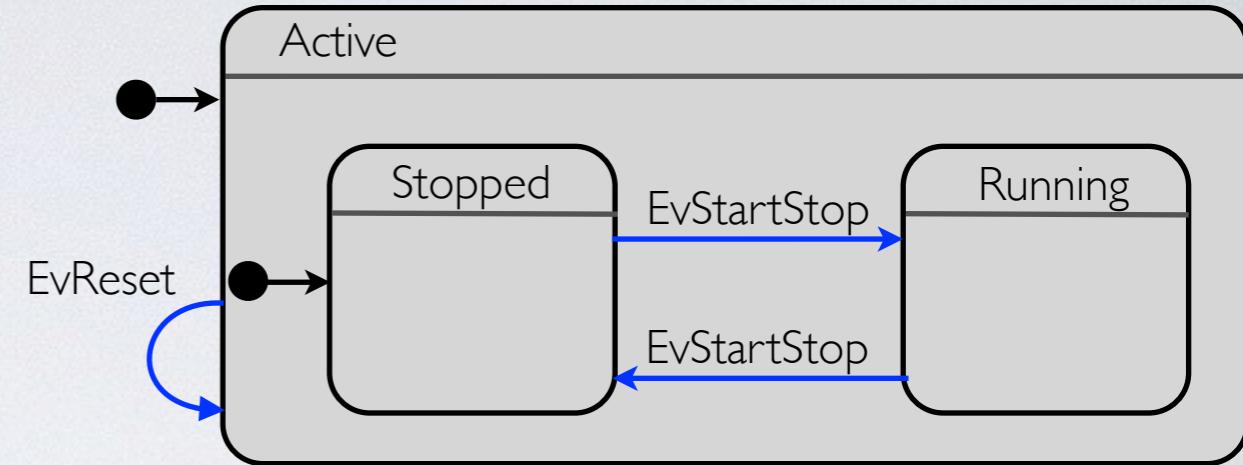
# Beispiel: Stopp Uhr mit C++ & Boost Bibliothek

```
#include <boost/statechart/transition.hpp>
...
// State active
struct Active : simple_state< Active, StopWatch, Stopped > {
    typedef transition< EvReset, Active > reactions;
};

// State running
struct Running : simple_state< Running, Active > {
    typedef transition< EvStartStop, Stopped > reactions;
};

// State stopped
struct Stopped : simple_state< Stopped, Active > {
    typedef transition< EvStartStop, Running > reactions;
};

int main() {
    StopWatch myWatch;
    myWatch.initiate();
    myWatch.process_event( EvStartStop() );
    myWatch.process_event( EvStartStop() );
    myWatch.process_event( EvReset() );
    return 0;
}
```



# Beispiel: Stopp Uhr mit C++ & Boost Bibliothek

```
...
```

```
// State active
struct Active : simple_state< Active, StopWatch, Stopped > {
    typedef transition< EvReset, Active > reactions;

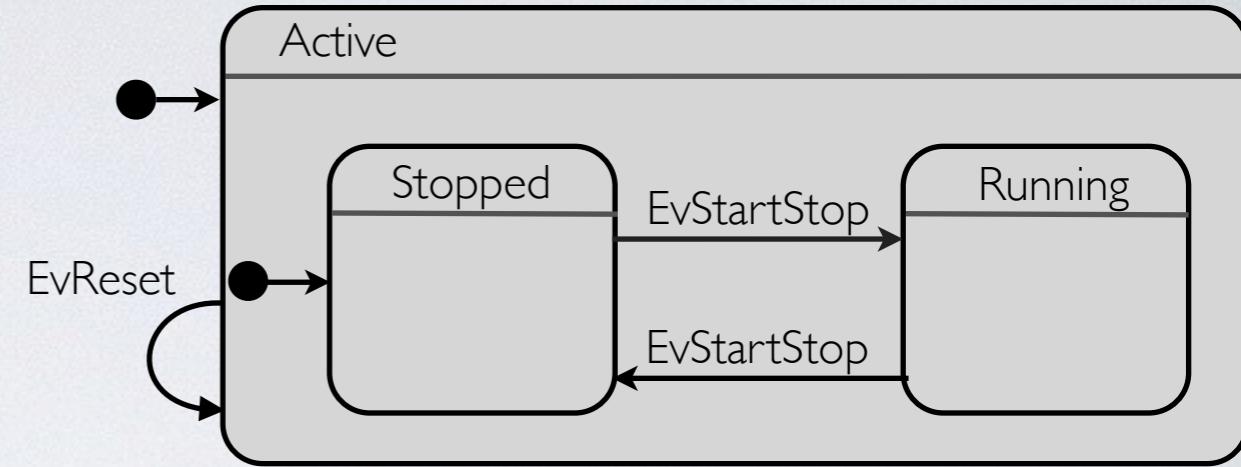
    Active() : elapsedTime_( 0.0 ) {}
    double & elapsedTime() { return elapsedTime_; }
private:
    double elapsedTime_;
};

// State running
struct Running : simple_state< Running, Active > {
    typedef transition< EvStartStop, Stopped > reactions;

    Running() : startTime_( std::time( 0 ) ) {}
    ~Running() { context< Active >().elapsedTime() += std::difftime( std::time( 0 ), startTime_ ); }
private:
    std::time_t startTime_;
};

// State stopped
struct Stopped : simple_state< Stopped, Active > {
    typedef transition< EvStartStop, Running > reactions;
};

int main() {
    StopWatch myWatch;
    myWatch.initiate();
    myWatch.process_event( EvStartStop() );
    myWatch.process_event( EvStartStop() );
    myWatch.process_event( EvReset() );
    return 0;
}
```



# Beispiel: Stopp Uhr mit C++ & Boost Bibliothek

```
struct IElapsedTime {
    virtual double elapsedTime() const = 0;
};

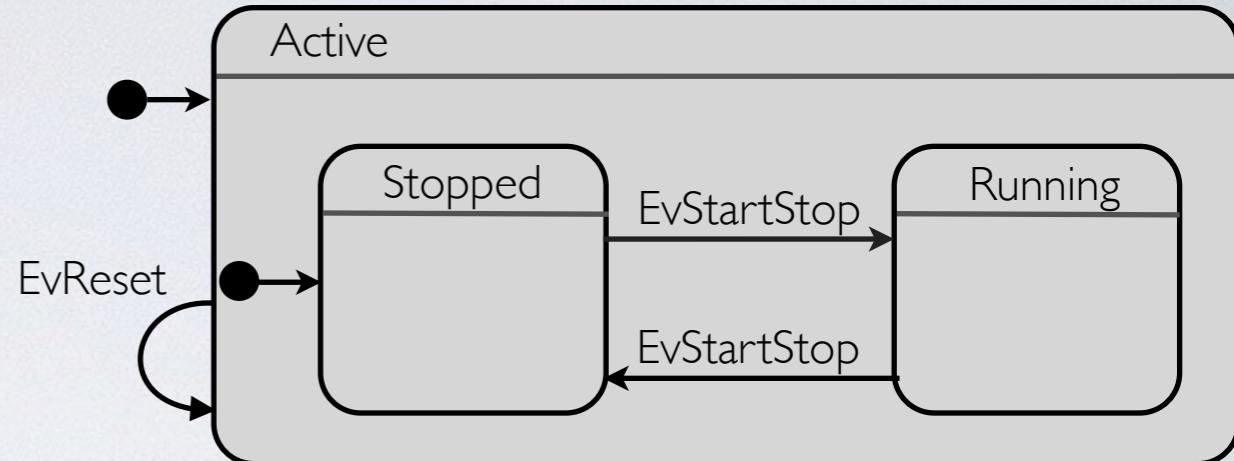
// State machine
struct StopWatch : state_machine< StopWatch, Active > {
    double elapsedTime() const {
        return state_cast< const IElapsedTime & >().elapsedTime(); }
};

// State active
struct Active : simple_state< Active, StopWatch, Stopped > {
    typedef transition< EvReset, Active > reactions;
    Active() : elapsedTime_( 0.0 ) {}
    double elapsedTime() const { return elapsedTime_; }
    double & elapsedTime() { return elapsedTime_; }
private:
    double elapsedTime_;
};

// State running
struct Running : IElapsedTime, simple_state< Running, Active > {
    typedef transition< EvStartStop, Stopped > reactions;
    Running() : startTime_( std::time( 0 ) ) {}
    ~Running() { context< Active >().elapsedTime() = elapsedTime(); }
    virtual double elapsedTime() const { return context< Active >().elapsedTime()
                                         + std::difftime( std::time( 0 ), startTime_ ); }
private:
    std::time_t startTime_;
};

// State stopped
struct Stopped : IElapsedTime, simple_state< Stopped, Active > {
    typedef transition< EvStartStop, Running > reactions;
    virtual double elapsedTime() const { return context< Active >().elapsedTime(); }
};

int main() {
    StopWatch watch;
    watch.initiate();
    watch.process_event(EvStartStop());
    sleep(1);
    watch.process_event(EvStartStop()); //watch.elapsedTime() = 1
    watch.process_event(EvStartStop()); //watch.elapsedTime() = 2
    sleep(3); //watch.elapsedTime() = 4
    watch.process_event(EvReset()); //watch.elapsedTime() = 0
    return 0;
}
```



# Statecharts

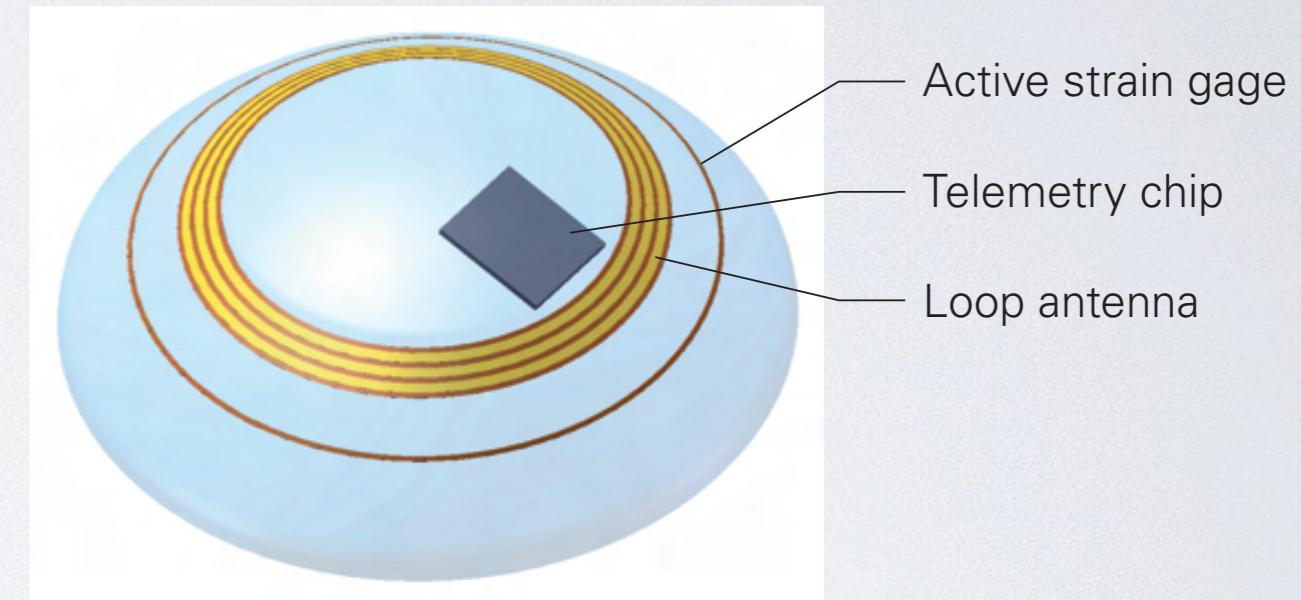
- Erweiterung von endlichen Automaten durch
  - Hierarchie / Modularität
  - Nebenläufigkeit
- Ähnliche/verwandte Ansätze vorhanden
  - UML Statechart Diagramme
  - Rhapsody
- Begrenzt einsetzbar für
  - komplexe Berechnungsverfahren
  - verteilte Systeme

# Statecharts: Vor- und Nachteile

- + Hierarchische Beschreibung endlicher Automaten
- + Nebenläufigkeit
- + Werkzeugunterstützung
- Unübersichtlich für komplexe Modelle
- Grenzen bei der Wiederverwendung
- Nicht ausgelegt für verteilte Systeme

# Beispiel einer wenig bekannten Anwendung: Kontaktlinse

- 2010
  - Kleines Embedded System
  - Messung von Augeninnendruck
  - Mikroprozessor,  
Dehnungssensor, Antenne
  - Energie über Radiowellen



- 2014
  - Google schützt Kontaktlinse mit eingebauter Kamera

[Quelle: [Sensimed AG, SENSIMED Triggerfish® brochure, 2010](#)]

# Literatur / Quellen

- Arduino, URL: <http://www.arduino.cc>, 2012
- Boost Statechart Library, URL, [http://www.boost.org/doc/libs/1\\_35\\_0/libs/statechart/doc/tutorial.htm](http://www.boost.org/doc/libs/1_35_0/libs/statechart/doc/tutorial.htm), 2012
- Harel, Statecharts: A Visual Formalism For Complex Systems, Science of Computer Programming 8, 1987
- Marwedel, Eingebettete Systeme, Springer-Verlag, 2008
- Sensimed AG, SENSIMED Triggerfish® brochure, URL: [http://www.sensimed.ch/images/pdf/sensimed\\_brochure\\_for\\_office\\_use.pdf](http://www.sensimed.ch/images/pdf/sensimed_brochure_for_office_use.pdf), 2010
- Quantum Leaps, QP™ Modeler, URL: <http://www.state-machine.com/qm/index.php>, 2012
- **Stand aller Internetquellen: 10.04.2012**