Exercises in System Level Programming (SLP) – Summer Term 2024

Exercise 7

Maximilian Ott

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

Chair in Distributed Systems and Operating Systems

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Presentation Assignment 3

AVR Timer

Timer: Motivation

- Common task for μController programming:
  - Regularly updating an output (e. g. frame rate)
  - Regularly reading of a value (e. g. serial console)
  - Pulse width modulation (PWM)
  - Passive waiting
  - ...

⇒ Implementation using a *timer*

Timer: Functionality

- A timer modifies a counter in every cycle
  - Increment (default)
  - Decrement
- When a previously configured event occurs, an interrupt is generated
  - Counter reaches a specific value
  - Counter overflows
  - (external event occurs)
- The ATmega328PB provides 5 different timers:
  - `TIMER{0,2}`: 8-bit counter
  - `TIMER{1,3,4}`: 16-bit counter

⇒ For all exercise tasks: `TIMER0`

⇒ Used by the libspicboard: `TIMER{1,2,4}`

Timer: Configuration (Timer clock speed)

- **How fast does the timer run:**
  - `TCCR0B`: TC0 control register B
  - `CSxx`: Clock select bits
  - Prescaler: Amount of CPU cycles until the counter is incremented
  - What happens when the CPU enters a sleeping state?

CS02	CS01	CS00	Description
0	0	0	Timer off
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. clock (falling edge)
1	1	1	Ext. clock (rising edge)

Timer: Configuration (Timer clock speed)

CS02	CS01	CS00	Description
0	0	0	Timer off
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. clock (falling edge)
1	1	1	Ext. clock (rising edge)

```c
static void init(void) {
  // Activate timer with prescaler 64
  TCCR0B &= ~(1 << CS02);
  TCCR0B |= (1 << CS01) | (1 << CS00);

  // [...]
}
```


Timer: Configuration (Trigger Event)

- **When does the timer trigger an interrupt:**
  - **Overflow**: When the counter flows over
  - **Match**: When the counter reaches a specific value
    - ⇒ Register `OCR0A` (`TIMER0` Output Compare Register A)
    - ⇒ Register `OCR0B` (`TIMER0` Output Compare Register B)
  - Interrupts can be unmasked individually
  - `TIMSK0`: `TIMER0` Interrupt Mask Register

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

Timer: Configuration (Trigger Event)

- **When does the timer trigger an interrupt:**
  - **Overflow**: When the counter flows over
  - **Match**: When the counter reaches a specific value
    - ⇒ Register OCR0A (TIMER0 Output Compare Register A)
    - ⇒ Register OCR0B (TIMER0 Output Compare Register B)
  - Interrupts can be unmasked individually
  - TIMSK0: TIMER0 Interrupt Mask Register

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

Timer: Configuration (Trigger Event)

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


```c
ISR(TIMER0_OVF_vect) {
  // [...]
}

static void init(void) {
  // Activate overflow interrupt
  TIMSK0 |= (1 << TOIE0);

  // [...]
}
```

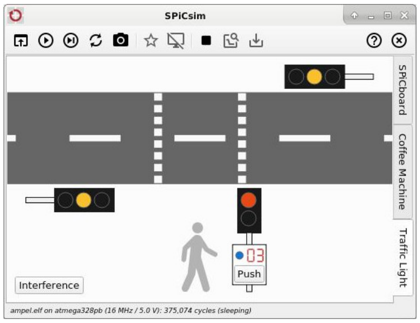

Timer: Example

- Reminder: `prescaler` $\in \{1, 8, 64, 256, 1024\}$

- **Example**:
  - 8-bit timer with overflow interrupt
  - CPU frequency: $16\,\mathrm{MHz}$ (ATmega328PB)
  - Goal: Count with a cycle of length $1\,\mathrm{s}$
  - ⇒ Which `prescaler` is the most resource efficient?
  - ⇒ How many overflow interrupts are required until $1\,\mathrm{s}$ has passed?
  - ⇒ How big is the error that we have to accept?

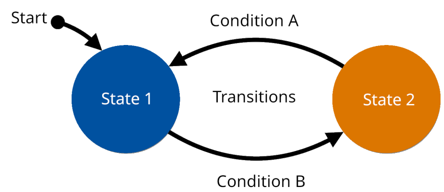
Task: Traffic Light

Assignment: Traffic Light



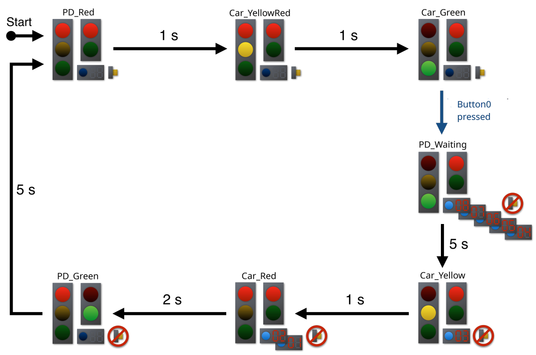
- Implementation of a (pedestrian) traffic light with waiting-time display

Finite State Machines

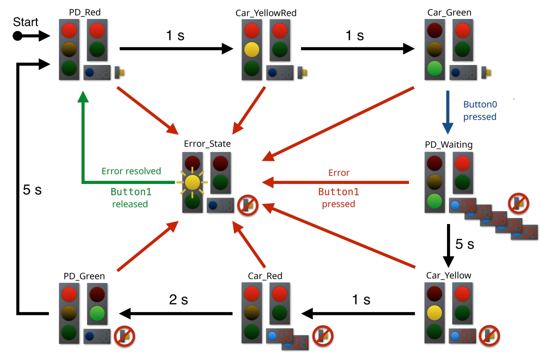


- **States** with specific attributes; well-defined initial state
- **Transition** depends on certain conditions

Traffic Light as a Finite State Machine



Traffic Light as a Finite State Machine



Choosing States: enum-Types

- Using states with hardcoded integer values is prone to errors
  - Hard to memorize
  - Range of value cannot easily be restricted
- Better enum:

```
enum state { STATE_RED, STATE_YELLOW, STATE_GREEN };

enum state my_state = STATE_RED;
```

- With typedef even more readable:

```
typedef enum { STATE_RED, STATE_YELLOW, STATE_GREEN } state;

state my_state = STATE_RED;
```


Choosing States: switch-case Instruction

```
switch ( my_state ) {
case STATE_RED:
  ...
  break;
case STATE_YELLOW:
  ...
  break;
case STATE_GREEN:
  ...
  break;
default:
  // maybe invalid state
  ...
}
```

- Avoid any `if-else`-cascades
- `switch`-expression has to be an integer (or even better: `enum`)
- Do not forget the `break`-instruction!
- Ideal for handling systems with different states
  ⇒ Implementation of finite state machines

Switching States

- Each transition is triggered by an interrupt
  - Configure `BUTTON0` and `BUTTON1` as interrupt inputs
    ⇒ Which edge should trigger the interrupt?
  - Configure `TIMER0` (interval: 1 second)
- Do not use the timer module of the libspicboard when submitting
  ⇒ However, its use can be helpful for debugging

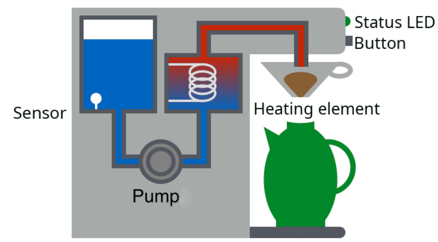
Hints

- Hints:
  - Implement each function exactly as specified in the description (reference implementation available)
  - Model presses of the buttons and alarms as events
  - Wait passively for all interrupts
  - “Deactivate” the button by simply ignoring its interrupt (It is not necessary to modify the interrupt configuration)
  - Mapping to a finite state machine can be useful
  - Usage of `volatile` always needs a reason

Hands-on: Coffee Machine

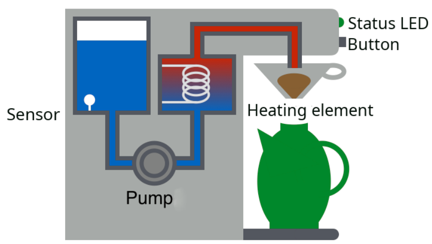
Screencast: `https://www.video.uni-erlangen.de/clip/id/17647`

Hands-on: Coffee Machine (1)



- Learning goals:
  - Finite state machines
  - Timers and alarms
  - Interrupts & sleep modes

Hands-on: Coffee Machine (1)



- Wiring:
  - Pump & heating: Port D, Pin 5 (active-low)
  - Button: `INT0` an Port D, Pin 2 (active-low)
  - Sensor: `INT1` an Port D, Pin 3 (water: high; no water: low)
  - State LED:
    - `BLUE0`: **STANDBY**
    - `GREEN0`: **ACTIVE**
    - `RED0`: **NO_WATER**

Hands-on: Coffee Machine (2)

STANDBY

- Machine is switched off
- Pump and heating are off
- User can start making coffee by pressing the button
- Initial state

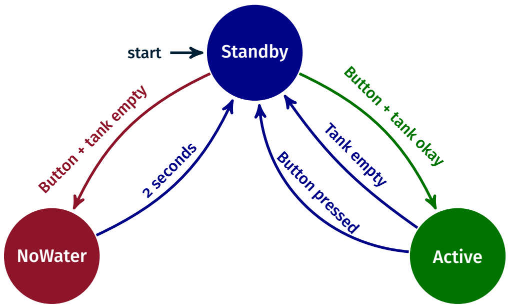
ACTIVE

- Machine is switched on
- Pump and heating are on
- Water tank is not empty
- User can stop the machine by pressing the button

NO_WATER

- Coffee machine shows that not enough water is in the tank
- Pump and heating are off
- Time period: 2 seconds

Hands-on: Coffee Machine (2)



- Hints:
  - Pressed button & change of water level by interrupts
  - State LED: `void setLEDState(state_t state)`
  - Waiting phases can be implemented using the single-shot alarms
  - During waiting phases always enter a power saving mode

Hands-on: Coffee Machine (3)

DDRx Configuration of pin i of port x as in-/output

- `Bit i = 1` → Pin i as output
- `Bit i = 0` → Pin i as input

PORTx Mode of operation **depends on DDRx**:

- If pin i is **configured as output**, then bit i in the PORTx register controls whether a high level or a low level has to be generated at pin i
  - `Bit i = 1` → high level at pin i
  - `Bit i = 0` → low level a pin i
- If pin i is **configured as input**, then the internal pull-up resistor can be activated
  - `Bit i = 1` → pull-up resistor at pin i (level is pulled high)
  - `Bit i = 0` → pin i configured as tri-state

PINx Bit i returns the current level of pin i at port x (read only)

Hands-on: Coffee Machine (4)

- Interrupt sense control (ISC) bits of the ATmega328PB are located at the external interrupt control register A (EICRA)
- Position of the ISC-bits inside the register defined by macros

Interrupt INT0		Interrupt on	Interrupt INT1	
ISC01	ISC00		ISC11	ISC10
0	0	low level	0	0
0	1	either edge	0	1
1	0	falling edge	1	0
1	1	rising edge	1	1

- ATmega328PB: External interrupt mask register (EIMSK)
- The position of the bits in this register is also defined by macros INTn