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

Exercise 8

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

Presentation Assignment 4

Hands-on: Coffee Machine

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

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

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

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 at 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)
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:

<table>
<thead>
<tr>
<th>Interrupt INTO</th>
<th>Interrupt on</th>
<th>Interrupt INTO1</th>
<th>Interrupt on1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISC01</td>
<td>ISC00</td>
<td>ISC11</td>
<td>ISC10</td>
</tr>
<tr>
<td>0 0</td>
<td>low level</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>either edge</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>falling edge</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>rising edge</td>
<td>1 1</td>
<td></td>
</tr>
</tbody>
</table>

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

In Depth: Strings

- char: Single character (e.g. 'a')
- String: Array of chars (e.g. "Hello")
- C: Last char of a string: '\0'

⇒ Memory requirement: strlen(s) + 1

```c
01 char s[] = "World\n";
02 char c = s[0];
03 c = s[4];
04 char *s2 = s + 2;
05 c = s2[1];
```

In Depth: Strings

- char: Single character (e.g. 'a')
- String: Array of chars (e.g. "Hello")
- C: Last char of a string: '\0'

⇒ Memory requirement: strlen(s) + 1

```c
01 char s[] = "World\n";
02 char c = s[0];
03 c = s[4];
04 char *s2 = s + 2;
05 c = s2[1];
```

Hands-on: Coffee Machine (4)

Hands-on: Ticker
In Depth: Strings

- **char**: Single character (e.g. 'a')
- **String**: Array of chars (e.g. "Hello")
- **C**: Last char of a string: '\0'

⇒ Memory requirement: strlen(s) + 1

```c
char s[] = "World\n";
char c = s[0];
c = s[4];
char *s2 = s + 2;
c = s2[1];
```

```c
01 char s[] = "World\n";
02 char c = s[0];
03 c = s[4];
04 char *s2 = s + 2;
05 c = s2[1];
```
Hands-on: Ticker

- **Functionality:**
  Displaying a text step-by-step on the 7-segment display

- **Learning goals:**
  - Strings in C
  - Pointers & pointer arithmetic
  - Alarms & sleep modes

- **Procedure:**
  - Recurring alarms with TIMER0
  - Combining the current substring
  - Output via the 7-segment display
  - During waiting phases, the microcontroller has to enter a sleep mode (passive waiting)

```c
const char *string = "HELLO SPIC";
const char *current = string;

// current[0] == 'H' && current[1] == 'E'
++current;

// current[0] == 'E' && current[1] == 'L'

// current[0] == '\0', current[1] == ?
current = string;
```

---

Hands-on: Ticker – Determine Substrings

```c
const char *string = "HELLO SPIC";
const char *current = string;

// ... == 'E' && current[1] == 'L'

// current[0] == '\0', current[1] == ??
current = string;
```
```
const char *string = "HELLO SPIC";
const char *current = string;

// current[0] == 'H' && current[1] == 'E'
++current;

// current[0] == 'E' && current[1] == 'L'
....

// current[0] == '\0', current[1] == ??
current = string;
```

```
const char *string = "HELLO SPIC";
const char *current = string;

// current[0] == 'H' && current[1] == 'E'
++current;

// current[0] == 'E' && current[1] == 'L'
....

// current[0] == '\0', current[1] == ??
current = string;
```
Hands-on: Ticker – Determine Substrings

```c
const char *string = "HELLO SPIC";
const char *current = string;
// current[0] == 'H' && current[1] == 'E'
++current;
// current[0] == 'E' && current[1] == 'L'
// [...] // current[0] == '\0', current[1] == ??
current = string;
```

Hands-on: Ticker – Determine Substrings

```
const char *string = "HELLO SPIC";
const char *current = string;
// current[0] == 'H' && current[1] == 'E'
++current;
// current[0] == 'E' && current[1] == 'L'
// [...] // current[0] == '\0', current[1] == ??
current = string;
```