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
Hands-on: Coffee Machine (1)

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
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
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
**DDRx** Configuration of pin $i$ of port $x$ as in-/output
- $\text{Bit } i = 1 \rightarrow \text{Pin } i \text{ as output}$
- $\text{Bit } i = 0 \rightarrow \text{Pin } i \text{ as input}$

**PORTx** Mode of operation depends on DDRx:
- If pin $i$ is **configured as output**, then bit $i$ in the PORT$x$ register controls whether a high level or a low level has to be generated at pin $i$
  - $\text{Bit } i = 1 \rightarrow \text{high level at pin } i$
  - $\text{Bit } i = 0 \rightarrow \text{low level at pin } i$
- If pin $i$ is **configured as input**, then the internal pull-up resistor can be activated
  - $\text{Bit } i = 1 \rightarrow \text{pull-up resistor at pin } i \text{ (level is pulled high)}$
  - $\text{Bit } i = 0 \rightarrow \text{pin } i \text{ configured as tri-state}$

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

- ATmega328PB: External interrupt mask register (EIMSK)
- The position of the bits in this register is also defined by macros INTn
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'
  
  \[ \Rightarrow \text{Memory requirement: strlen(s) + 1} \]

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td><code>char s[] = &quot;World\n&quot;;</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td>02</td>
<td><code>char c = s[0];</code></td>
<td><code>0x0911</code></td>
</tr>
<tr>
<td>03</td>
<td><code>c = s[4];</code></td>
<td><code>0x0910</code></td>
</tr>
<tr>
<td>04</td>
<td><code>char *s2 = s + 2;</code></td>
<td><code>0x090f</code></td>
</tr>
<tr>
<td>05</td>
<td><code>c = s2[1];</code></td>
<td><code>0x090e</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>0x090d</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>0x090c</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>0x090b</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>0x090a</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>0x0909</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>0x0908</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>0x0907</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>0x0906</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>...</code></td>
</tr>
</tbody>
</table>
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];
```
In Depth: Strings

- `char`: Single character (e.g. 'a')
- `String`: Array of `char`s (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];
```
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];
```

<table>
<thead>
<tr>
<th>Stack ↓</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x0911</td>
</tr>
<tr>
<td>s[6]</td>
<td>'\0'</td>
</tr>
<tr>
<td>s[5]</td>
<td>'\n'</td>
</tr>
<tr>
<td>s[4]</td>
<td>'d'</td>
</tr>
<tr>
<td>s[3]</td>
<td>'l'</td>
</tr>
<tr>
<td>s[2]</td>
<td>'r'</td>
</tr>
<tr>
<td>s[1]</td>
<td>'o'</td>
</tr>
<tr>
<td>s[0]</td>
<td>'W'</td>
</tr>
<tr>
<td>C</td>
<td>'d'</td>
</tr>
</tbody>
</table>

0x0910 0x090f 0x090e 0x090d 0x090c 0x090b 0x090a 0x0909 0x0908 0x0907 0x0906
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
char s[] = "World\n";
char c = s[0];
c = s[4];
char *s2 = s + 2;
c = s2[1];
```

```
<table>
<thead>
<tr>
<th></th>
<th>Stack ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>0x0911</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>s[6]</td>
<td>'\0'</td>
</tr>
<tr>
<td>s[5]</td>
<td>'\n'</td>
</tr>
<tr>
<td>s[4]</td>
<td>'d'</td>
</tr>
<tr>
<td>s[3]</td>
<td>'l'</td>
</tr>
<tr>
<td>s[2]</td>
<td>'r'</td>
</tr>
<tr>
<td>s[1]</td>
<td>'o'</td>
</tr>
<tr>
<td>s[0]</td>
<td>'W'</td>
</tr>
<tr>
<td>c</td>
<td>'l'</td>
</tr>
<tr>
<td>s2</td>
<td>0x090c</td>
</tr>
<tr>
<td>s2</td>
<td>0x090c</td>
</tr>
<tr>
<td></td>
<td>0x0906</td>
</tr>
</tbody>
</table>
```
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)
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

```c
01 const char *string = "HELLO SPIC";
02 const char *current = string;
03 // current[0] == 'H' && current[1] == 'E'
04 ++current;
05 // current[0] == 'E' && current[1] == 'L'
06 // [...]
07 // current[0] == '\0', current[1] == ??
08 current = string;
```

string → H E L L O S P I C \0
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;
```
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

```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;
```

string → HELLO L O S P I C \0

...
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

```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;
```

<table>
<thead>
<tr>
<th>String</th>
<th>H</th>
<th>E</th>
<th>L</th>
<th>L</th>
<th>O</th>
<th>S</th>
<th>P</th>
<th>I</th>
<th>C</th>
<th>\0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The `string` is initially set to "HELLO SPIC", and `current` is set to a memory address pointing to the same location. The code checks conditions for substrings, modifying `current` accordingly.
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;
```

string → HELLO SPIC \0

current → \0 ??
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;
```

---

string → HELLOLOSPIC\0 → current

---