Exercises in System Level Programming (SLP) – Sommersemester 2024

Exercise 1

Maximilian Ott

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

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

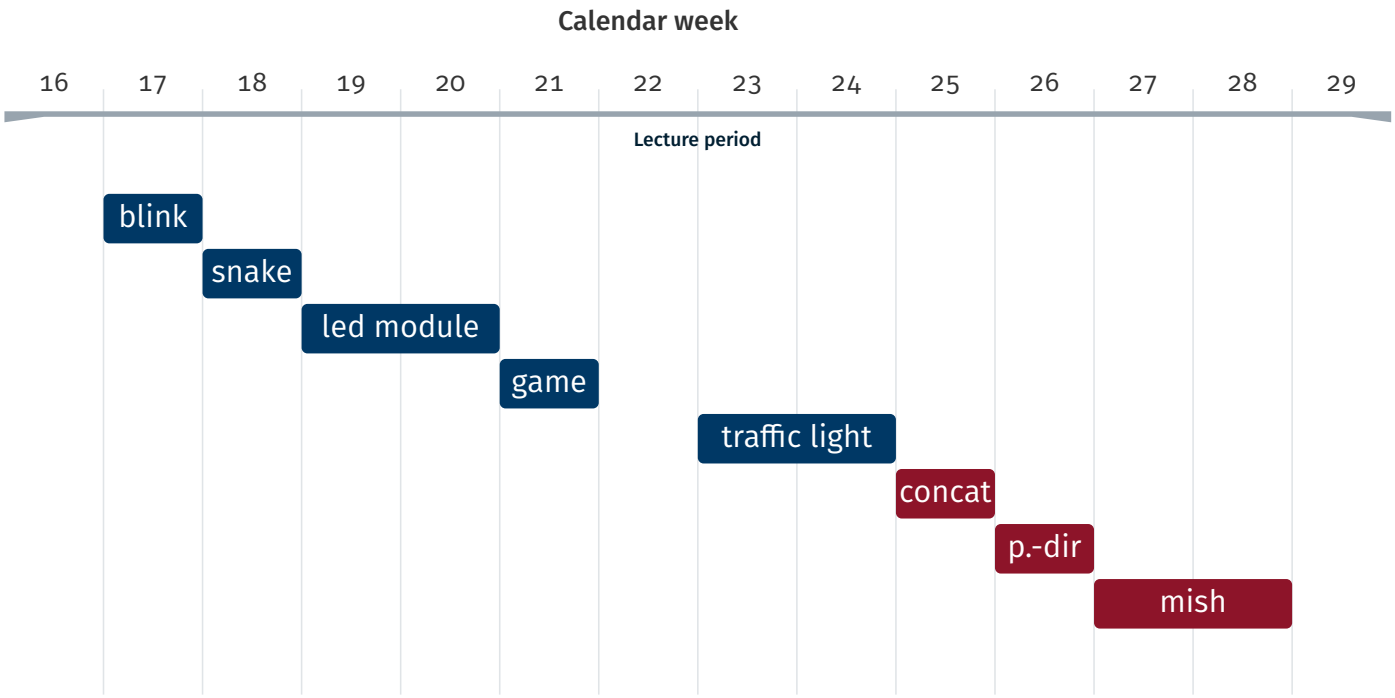
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Organizational Matters

Tutorial

- Concept of Tutorial:
  1. Correct the last programming assignment
  2. Deepen lecture materials
  3. Introduction to the new programming assignments
  4. Possibly development of a solution sketch
  5. Hands-on: joined programming
- Slides are not necessarily made to be studied on their own
  → attendance required, write along
- Overview for the term and SLP appointments:
  `https://sys.cs.fau.de/lehre/SS24/spic/`

Assignments



Solutions

- Assignments are submitted via Linux
- Automatic check for plagiarism
  - Comparison to all other solutions (including old ones)
  - Plagiarism yields 0 points

  $\Rightarrow$ If in doubt talk to your tutor
- Deduction of points
  - -1 point for each compiler warning
  - -50% of possible points if the code does not compile
- (Helpful) comments in the code can help you and your tutor

Bonus Points

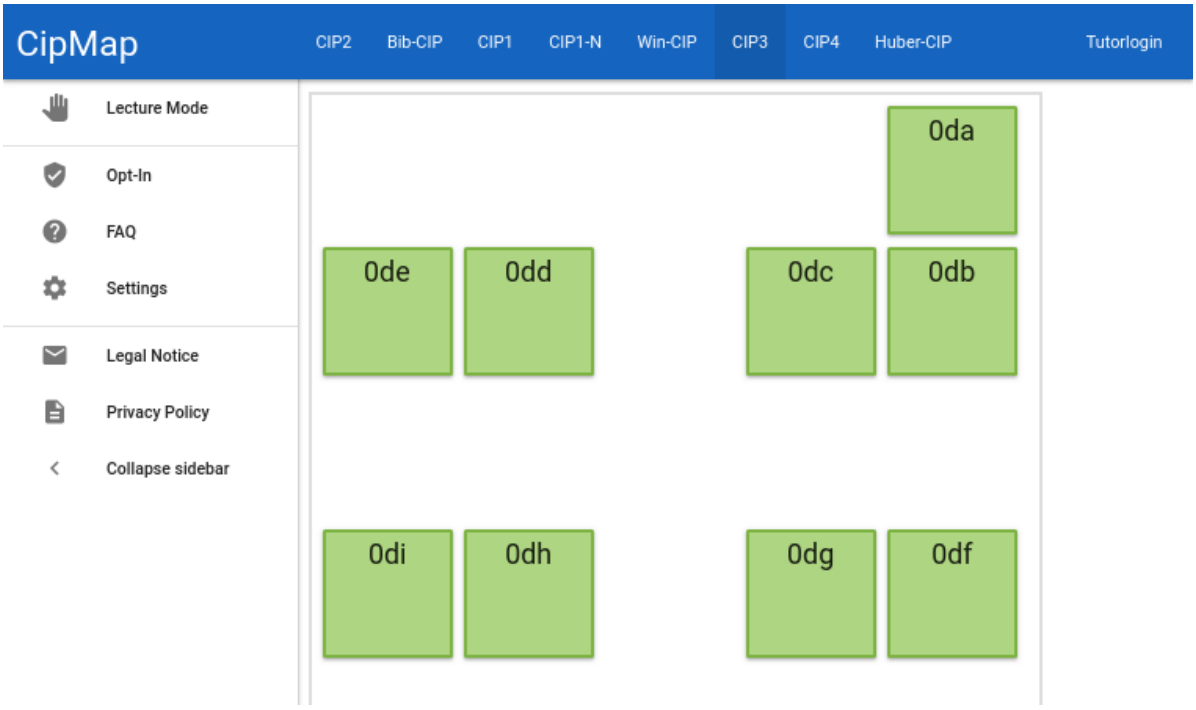
- Submitted assignments get graded with bonus points
- If you reach 20% or more of all bonus points, there is a bonus for the exam
- For 80% or more you get rewarded with full bonus points for the exam
- Conversion of points from the assignments into bonus points for the exam (up to 10% of points)

  $\longrightarrow$ Example: 80% of points from the assignments yield 9 bonus points if the exam has 90 points total
- However, you *cannot pass* the exam by the help of bonus points
- Bonus points cannot be transferred to the next semester

Computer Exercise

- Room for the Computer exercise: 01.153-113 (WinCIP)
- Help from the tutor during your work with the assignment
  „First come, first served“-principle
- If after 30 minutes after the beginning of the Computer exercise no student is present, the exercise is cancelled

CipMap



Make Requests via CipMap

1. Visit the site cipmap.cs.fau.de
2. Choose the room where the Computer exercise takes place (e. g. 01.153-113)
3. Click on *Lecture Mode*.
   - **colored PC:** request sent
   - **grey PC:** no request
4. By clicking *Request Tutor*, a request will be queued
5. After your question is answered: click on the button again to mark the request as finished

Please note:

- You can only make requests during the time of Computer exercises
- When logging off, all open requests get deleted

If a problems occur

- Consult the slides
- Write an e-mail

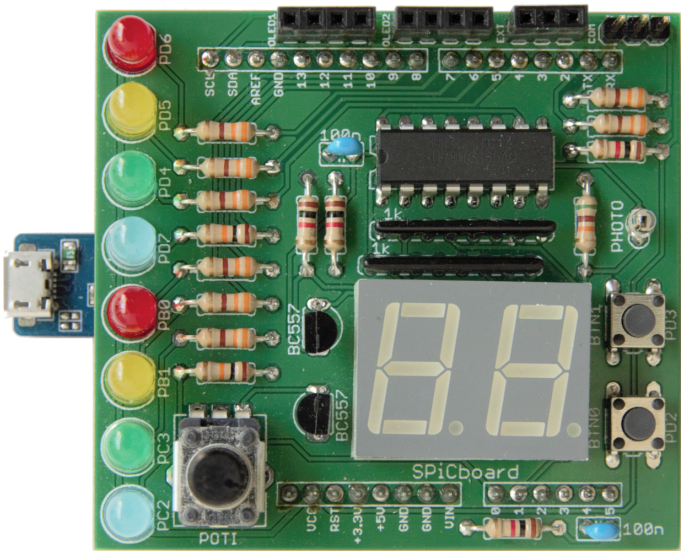
  Questions on lecture contents (tutors):
  i4slp@i4.cs.fau.de

  Organizational questions (all staff):
  i4slp-orga@i4.cs.fau.de

Development Environment

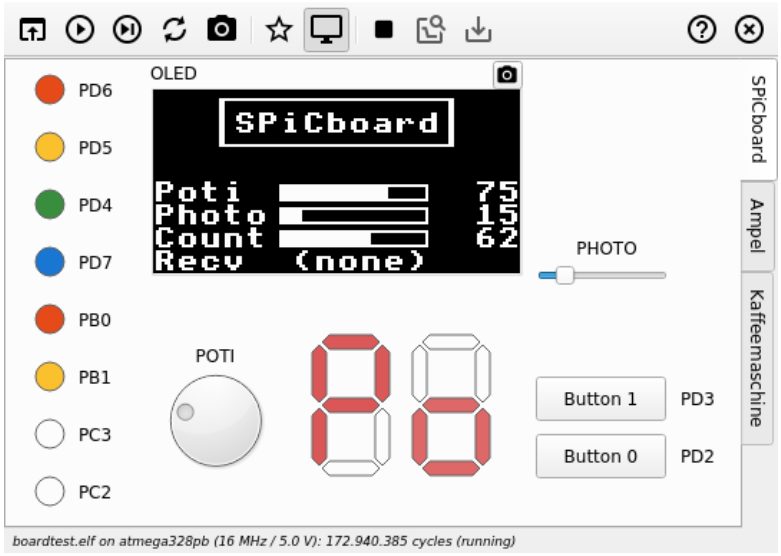
Hardware: SPiCboard

- **ATmega328PB Xplained Mini:**
  Micro-controller board with integrated programmer/debugger
- Custom-made extension PCB for SPiC/SLP



Simulator: SPiCsim

- **SPiCsim:**
  Simulates ATmega328PB and SPiCBoard
- Makes recording and visualizing of signals possible



Handling the Assignments

- Supervised programming for the assignments during Computer exercises
  - ⇒ Hardware is made available during the exercises
- Independent working style (partially) required
  - Using own SPiCboard: can be soldered at the soldering night
  - SPiCboard Simulator: `SPiCsim`

Function Libraries

- `libspicboard`: function library for addressing the hardware

  Example: `sb_led_on(GREEN0);` switches on the first green LED
- Direct configuration of the hardware by the application developer is not needed
- Usage mainly for the first assignments, later the functions of the `libspicboard` have to be implemented by yourself
- Documentation online:

  `https://sys.cs.fau.de/lehre/SS24/spic/uebung/spicboard/libapi`

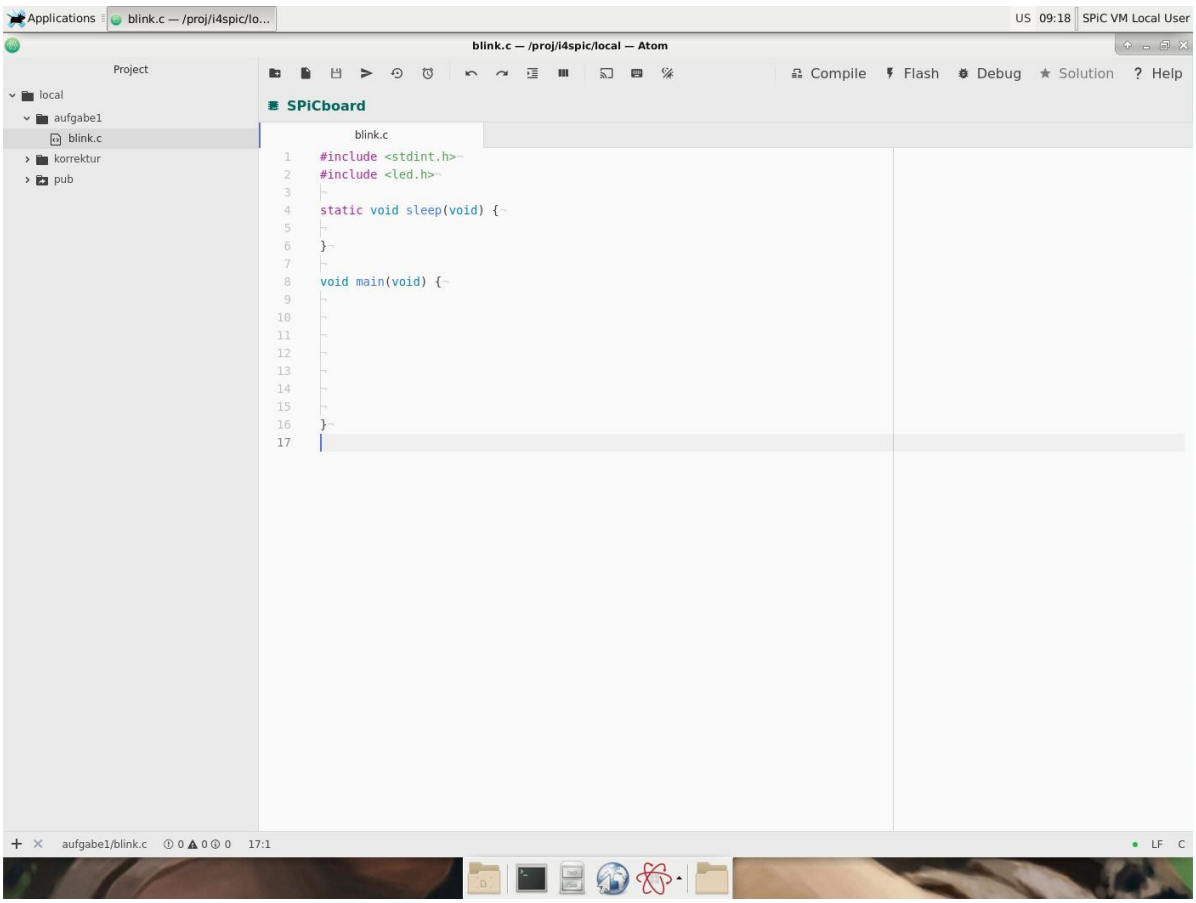
Important Directories

- Public directory `/proj/i4spic/<login>/pub/`
  - Auxiliary material for each assignment can be found in `aufgabeX/`
  - `libspicboard` with documentation and minimal working examples
  - All lecture slides in `lecture/`
  - All exercise slides in `exercise/`
  - Assistance for dealing with the language *C*

Important Directories

- Public directory `/proj/i4spic/<login>/pub/`
  - Auxiliary material for each assignment can be found in `aufgabeX/`
  - `libspicboard` with documentation and minimal working examples
  - All lecture slides in `lecture/`
  - All exercise slides in `exercise/`
  - Assistance for dealing with the language *C*
- Project directory
  - `/proj/i4spic/<login>/`
  - Solutions have to be saved in subdirectories `aufgabeX`
    - ⇒ The program for submitting searches only there
  - Others cannot read this directory
  - Directory is created automatically
  - Contains symbolic links to the public directory

The Editor



- Can be found in the start menu in *FAU Courses* as *SPiC-IDE*
- Designed in particular for SPiC, based on Atom
- Combines editor, compiler and debugger into a signle environment
- Cross-compiler for creating programs for different architecture
  - Host system: Intel-PC
  - Target system: AVR-Mikrocontroller

Manuals

CIP-Login

- To use the CIP infrastructure (and therefore the tools for assignment submission) a login for the CIP is required

  When running into problems, please contact the CIP Admins
- Criteria for a secure password
  - At least 8 characters, 10 is better
  - At least 3 different types of characters, 4 are better (capitalized letters, small letters, digits, special characters)
  - **Do not** use any dictionary words, names, login, etc.

Submitting Assignments (1)

- At the latest after testing the program, you should submit your solution for grading
- When working with a partner, only ONE of you is allowed to submit the assignment!
  - Your partner has to take part in the same Tutorial
  - When submitting, you can specify your partner
- Submission in the SPiC IDE with the click of a button or
- Open a terminal window and execute the following command (`aufgabeX` has to be replaced):

  ```
  /proj/i4spic/bin/submit aufgabeX
  ```
  - Important: **green text** indicates that the submission was successful,
    red text indicates an error!

Submitting Assignments (2)

- Causes for an error
  - Necessary files are not present in the right directory
  - `aufgabeX` has to be written without capitalization
  - `.c`-file has been wrongly named
  - Deadline was missed
- Useful tools
  - Show the source code of the submitted assignment: `/proj/i4spic/bin/show-submission aufgabeX`
  - Differences between submitted version and current version in the project directory `/proj/i4spic/<login>`: `/proj/i4spic/bin/show-submission aufgabeX -d`
  - Show deadline: `/proj/i4spic/bin/get-deadline aufgabeX`

Your Todos

1. Registration for the exercises via Waffel: `https://waffel.cs.fau.de`
   - For submission and correction of assignments

   ⇒ from **Thursday, 18.04.2024, 6:00 PM**
2. Registration for the CIP: `https://account.cip.cs.fau.de`
   - For working on the assignments, submitting them and receiving feedback

Your Todos

1. Registration for the exercises via Waffel: `https://waffel.cs.fau.de`
   - For submission and correction of assignments

   ⇒ from **Thursday, 18.04.2024, 6:00 PM**
2. Registration for the CIP: `https://account.cip.cs.fau.de`
   - For working on the assignments, submitting them and receiving feedback

Since the registration for the CIP can take up to 24 hours until you can log in with your new account, please make sure to **register asap**. Without an account you cannot take part in working on the assignments!

Compiler Optimizations

Compiler Optimizations: Background

- AVR micro-controller, as well as nearly all CPUs cannot execute operations directly on memory
- Procedure of operations:
  1. **Load** the operands from the memory into processor registers
  2. **Execute** the operations using the registers
  3. **Store** the result into memory

  ⇒ More detailed description in the lecture
- The compiler is allowed to arbitrarily change the code as long as the "global" state after exiting a function stays the same
- Optimizations can lead to drastically faster code

Compiler Optimizations: Examples

- Typical optimizations:
  - When entering a function the variable is loaded into a register and only written back to memory when leaving the function
  - Redundant and "dead" code is removed
  - Some instructions get reordered
  - For automatic variables no memory is reserved; they are placed in processor registers instead
  - If possible, the compiler does some calculations (constant folding):
    `a = 3 + 5;` is replaced `a = 8;`
  - The range of values of automatic variables gets adapted:
    Instead of `0` to `10`, one can count from `246` to `256` (`= 0` for `uint8_t`) and then check if an overflow occurred

Compiler Optimizations: Example (1)

```
01 void wait(void) {
02   uint8_t u8 = 0;
03   while(u8 < 16) {
04     u8++;
05   }
06 }
```

- Incrementing the variable u8 up to a value of 16
- Used for e. g. active waiting

Compiler Optimizations: Example (2)

- Assembler without optimizations

```
01 ; void wait(void){
02 ; uint8_t u8;
03 ; [Prologue (store registers, initialize Y, etc.)]
04 rjmp while      ; jump to while
05 ; u8++;
06 addone:
07 ldd r24, Y+1    ; load data from Y+1 into register 24
08 subi r24, 0xFF  ; subtract 255 (add 1)
09 std Y+1, r24    ; write data from register 24 into Y+1
10 ; while(u8 < 16)
11 while:
12 ldd r24, Y+1    ; load data from Y+1 into register 24
13 cpi r24, 0x10   ; compare register 24 with 16
14 brcs addone     ; if smaller, jump to addone
15 ;[Epilogue (restore registers)]
16 ret             ; return from the function
17 ;}
```

Compiler Optimizations: Example (3)

- Assembler with optimizations

```
01 ; void wait(void){
02 ret             ; Return from the function
03 ; }
```


Compiler Optimizations: Example (3)

- Assembler with optimizations

```
01 ; void wait(void){
02 ret             ; Return from the function
03 ; }
```

- C does not know the semantics of a waiting loop
- The loop does not have any effect on the (global) state

$\rightsquigarrow$ The compiler optimises the loop by removing it

Keyword volatile

- Variables can be declared as `volatile`

⇝ The compiler is not allowed to optimise the variable:

  - **Memory has to be reserved** for the variable
  - The **life span** cannot be shortened
  - Prior to each operation, the variable has to be **loaded from memory** and afterwards it has to be written back to memory
  - The **range of value** of the variable cannot be adapted

- Possible uses of `volatile`:
  - Active waiting loops: prevents optimization of the loop
  - Concurrent execution (later in the lecture)
    - Variable is used in the interrupt handler and in the main loop
    - Changes of the variable have to be “made observable”
  - Access to hardware (e. g. pins) ⇝ important for the LED module
  - (Debugging: the value cannot be removed due to optimizations)

Task: blink

Task Description: blink

- Learning objective:
  - Make first experiences with the programming environment and the submission system
  - Active waiting
- Flashing LEDs `YELLOW0` and `YELLOW1`
  - Switching on and off alternately (warning light)
  - Frequency of approx. 2 times per second
  - Use of the library functions for addressing the LEDs
  - Implementation by active waiting (loop with counter)
- Documentation of the library:

  `https://sys.cs.fau.de/lehre/SS24/spic/uebung/spicboard/libapi`
- File to be submitted: `blink.c`

Hands-on: Light

Screencast: `https://www.video.uni-erlangen.de/clip/id/13444`

Hands-on: Light

- Inside the SPiC-IDE:
  - Create new folder (e. g. hands-on/licht)
  - Create new source file (e. g. `licht.c`)
- Create the program:
  - Switch on one LED (e. g. `GREEN0`)
  - Wait inside an endless loop
- Inside the SPiC-IDE:
  - Compile the program
  - Test and execute the program in the simulator or on an actual SPiCboard