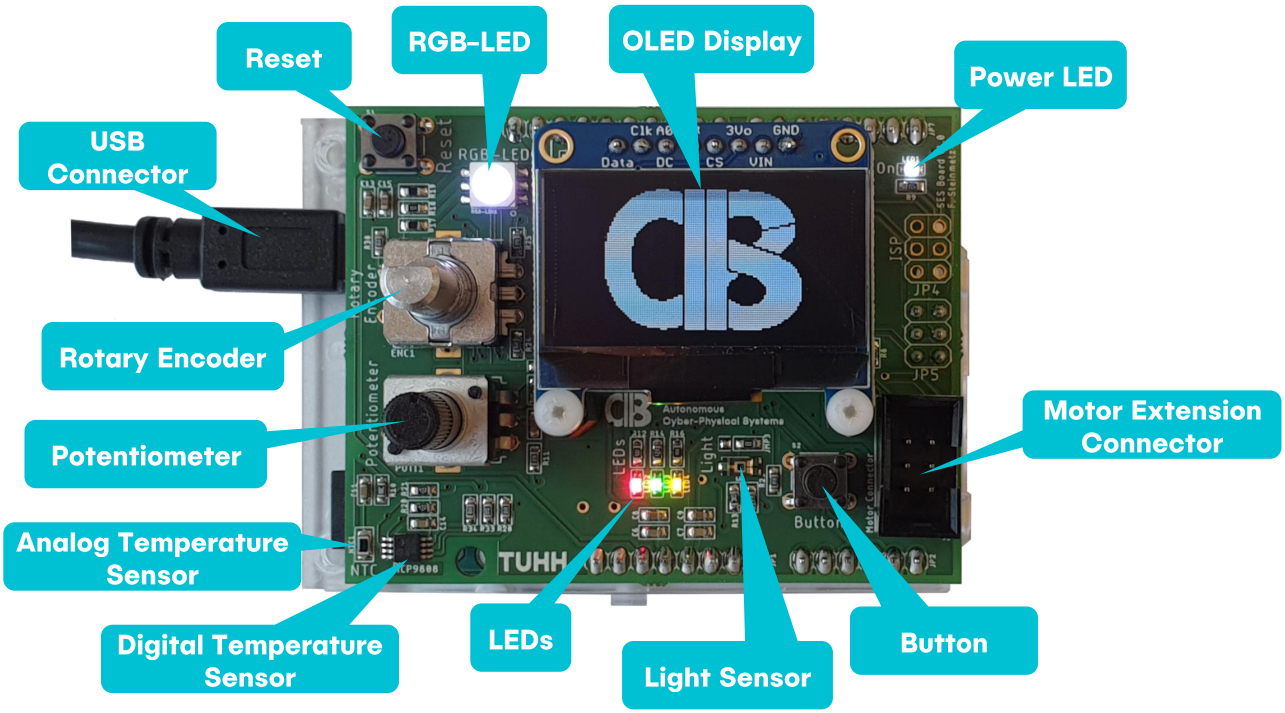
Exercise Sheet

Installation and Bit Arithmetic

April 15th, 2025

Successful participation in this exercise is required to continue the course. As proof of having set up the toolchain correctly, both team members must show that they can flash a program to the hardware with their notebooks during the first interview.

In the following labs, we will put several components of the SES board into operation. In this lab, however, we'll have an easy start with the LEDs.



Preparation

Read the *Usage of Git Version Control* (Stud.IP) and install the software needed for AVR programming. Execute the example given in the tutorial.

Read the *Git Usage Guide* (Stud.IP) and make sure you have your project folder connected to your groups' Git repository. Commit and push your solutions to the repository regularly. We expect every group to have a working project pushed to their git repository by next week's exercise session.

TUHH Hamburg University of Technology

Prof. Dr.-Ing. B. Renner
Institute for Autonomous Cyber-Physical Systems

Task 1.1: Get the Party Started

Open a new VSCode window and click on *File* → *Open Workspace*. In the root of your Git repository, there is a file called *ses-workspace.code-workspace*. Select this file.

Your workspace already contains a folder for every task on this sheet. You will learn in the next lab how to extend your workspace with new projects/tasks. Have a look into the folder *task_1-1/src*. It contains a simple LED-blinking program.

```
#include <avr/io.h>
#include <util/delay.h>

/** toggles the red LED of the SES-board */
int main(void) {

    DDRF |= (1 << DDF5);

    while (1) {
        _delay_ms(1000);

        PORTF ^= (1 << PORTF5);
    }
    return 0;
}
```

Here, three Macros `PORTF`, `DDRF`, and `PF5` are used. Explain their function!

Use the provided datasheet *ATmega32U4.pdf* and the *SES Board Circuit Diagram* (Stud.IP) to find out how to toggle the other LEDs on the board

Make sure you can build and upload this program. You have to show this during the first interview.

Task 1.2: Looping Louie

For this exercise, switch to the folder *task_1-2*. Always use a new folder for every exercise.

On some occasions—e.g., the initialization of a display—delays have to be inserted in the program before performing the next operation. A straightforward way of doing this is busy waiting—i.e., running a loop to burn processor cycles. You have already seen the function `_delay_ms()`. In this exercise, you implement a waiting function on your own. The function

```
void wait(uint16_t millis)
```

takes a 16-bit unsigned integer `millis` as input parameter, corresponding to the requested delay in milliseconds. You can test your function with the LED-blinking program.

For this task, assume a processor clock frequency of 16 MHz (16 million clock cycles per second). Furthermore, an arbitrary busy waiting function is shown in the code snippet below. The challenge is to determine the clock cycles that the C program uses for each iteration of the for-loop. This can be achieved by inspecting compiler-generated assembly code.

TUHH Hamburg University of Technology
Prof. Dr.-Ing. B. Renner
Institute for Autonomous Cyber-Physical Systems

Displaying the assembly code is a bit tricky with PlatformIO. In the menu bar at the top of the VSCode window, select *Terminal* → *New Terminal*. In the appearing terminal, you have to enter a command that takes the compiled **.elf* file and generates human-readable assembly code from it. The **.elf* file is placed by the compiler in the same directory as the **.hex* file, e.g. *.../task_1-1/.pio/build/ses_avr*.

The program needed for translation is part of the toolchainthat is installed by PlatformIO into your user directory. The toolchain's executables are by default placed in

```
~/.platformio/packages/toolchain-atmelavr/avr/bin
```

In Linux, MAC, and modern Windows systems—where PowerShell is available—the command is

```
<toolchain-path>/objdump -h -S <elf-path>
```

After calling this program, you see the assembly code for each function in the terminal window. A short description of the mnemonics and the required clock cycles can be found in the document *AVR Instruction Set Manual*, which can be retrieved from Stud.IP.

```c
#include <stdint.h>
#include <avr/io.h>

void shortDelay (void)
{
    uint16_t i;
    for (i = 0x0100; i > 0; i--) {
        // prevent code optimization by using inline assembly
        asm volatile ( "nop" ); // one cycle with no operation
    }
}
```

Task 1.3 : Around the Clock

To test the correctness of your delay loop, change the LED program to use your delay function instead of the `_delay_ms(int ms)` function. After exactly 1 second, toggle the LED. Check your program by measuring, e.g., with a watch, the time it takes to blink exactly 60 times.

Task 1.4 : Practice C

During the semester, you will often have to use pointers and to manipulate single bits in registers. In this exercise, you practice that. Ensure you have installed the VSCode extension *C/C++ Compile Run* as described in the *SES Toolchain Tutorial*. Switch to the folder *task_1-4*, open the file *main.c*, and run it by pressing F6.

Try to understand the program and play around with it a little.

TUHH Hamburg University of Technology

Prof. Dr.-Ing. B. Renner
Institute for Autonomous Cyber-Physical Systems

Task 1.5 : A Bit of Masking and Shifting

Bit operations are performed to access various internal components and peripherals of the microcontroller. Solve the following tasks by writing to and reading from the variable `uint8_t var` with *a single* C statement. Try this by hand first; afterwards, you can check your solution by writing a small test program in C.

- Set bit 3
- Set bits 4 and 6 (with / without bit shifting)
- Clear bit 2
- Clear bits 2 and 7
- Toggle (invert) bit 3
- Set bit 2 and clear bits 5 and 7 at the same time
- Swap bits 3–5 and bits 0–2


For each operation, ensure that the remaining bits of `var` are not changed!


Note that the least significant bit (LSB) is bit 0 and the most significant bit (MSB) is bit 7. Use the following bit operators:

`&`	bitwise AND	`|`	bitwise OR	`^`	bitwise XOR
`«`	left shift	`»`	right shift	`~`	one's complement

Task 1.6 : More Fun with Bits

Write a function that reverses the bit order of an 8 bit (use `uint8_t`) input: Given the input's bit representation $(b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0)$, the output becomes $(b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7)$.

Task 1.7 : Take a look at Atmel's ATmega32U4

In general, you should be aware of the resources and capabilities of your embedded system. For this purpose, please try to find out more about the ATmega32U4 by reading the datasheet! Find the numbers of the following parameters:

- size of programmable flash
- size of internal EEPROM
- size of internal SRAM
- available ports
- available counters/timers
- peripherals (e.g., UART, ADC)