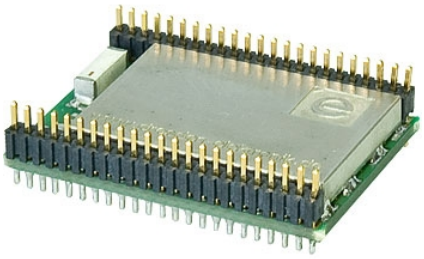
AVR Toolchain Tutorial

During the SES lab, you will program an Atmel ATMega128RFA1 microcontroller. This tutorial describes the installation of the required tools for compiling and flashing a program on the microcontroller. Since the exercise can not take place in person due to the coronavirus, you will use a web interface to upload your programs to the microcontrollers. The usage of this web interface is also described in this tutorial.



The following steps are required to participate in the exercise.

- Install Python 3.
- Install Visual Studio Code with PlatformIO extension.
- Compile a first program.
- Install Git Version Control.
- Checkout your group repository.
- Familiarize yourself with the remote flashing application.

Install Python 3

(You can skip this section, if you have already installed Python 3.)

Windows

Download the latest release from

https://www.python.org/downloads/windows/

and install it following the instructions from the installer. When you are asked during the installation if you want to add python to the system *Path*, do so. Therefore, please note to which directory you are installing Python.

Ubuntu/Debian

use the terminal to install python. Type

```
sudo apt install python3 python3-dev
```

Mac OS X

Download and install the latest release from

https://www.python.org/downloads/mac-osx/

Visual Studio Code and PlatformIO

The Integrated Development Environment (IDE) for this lab is **Visual Studio Code (VSCode)**, which will be used for coding and compilation. The plugin **PlatformIO (PIO)** contains the compiler needed for our AVR microcontroller. We highly recommend using this IDE. If you prefer a different one, we will not offer any support with installing the toolchain or using the IDE.

Download and install VSCode for your operating system from the official page

https://code.visualstudio.com/Download

Once you have installed VSCode, run it and go to the extensions page. The latter is available by clicking on the small extensions logo in the toolbar on the left (see Region A in Figure 1). In the search bar, search for the `PlatformIO IDE` extension and install it. This extension already comes with the complete toolchain to build and flash the programs on the microcontroller.

During the installation, the extension might ask you where to find the python executables. If it does, please provide the path pointing to your python installation. At the end, you will be asked to reload VSCode.

Create a new Project

To create a new project go to the **PlatformIO home screen** and click on *New Project* (Figure NewProject.png). In the popup, you are asked to provide a project name and to choose a Platform. Select the `ATMega128/A (Microchip)` option and click on *finish*.

Now, the project files are shown on the left in the *Explorer*. Next, you have to make the configuration for our board. In the *Explorer*, click on the `platformio.ini` file. Replace the content of the file with the content shown to the right.

```
[env:ses_avr]
platform = atmelavr
board_mcu = atmega128rfa1
board_f_cpu = 16000000L
```

Visual Studio Code

Your VSCode should now look like in Figure 1

- **A**: The left toolbar allows you to switch between the Project Explorer, Version Control, Extensions and other views.
- **B**: Dependent on the selection in the left toolbar (**A**). Here, it shows the Project Explorer — an overview of all the files in your project. You can also create new files.
- **C**: The code editor — Here, you can modify the source files. The syntax is highlighted for most programming languages and you can use autocompletion.
- **D**: The output view — Here you can see outputs from different actions, such as compilation of your program or flashing it to the microcontroller.
- **E**: The `PlatformIO` toolbar. This is a convenient way to start actions such as building and uploading.

Writing code

Now you can start writing code. In the *Project Explorer*, you see several different folders. Your source code folder is called *src* and already contains a file *main.cpp*. Delete this file and create a new file in the

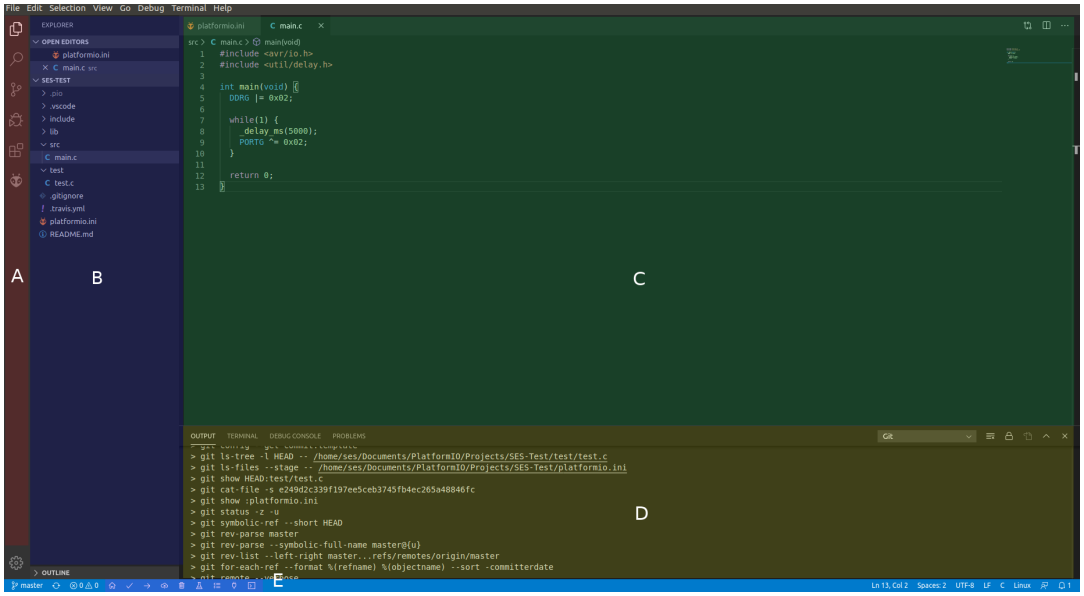


Figure 1: Visual Studio Code - Perspectives

src folder called *main.c*, which will be the entry point of your program. Open this file by clicking on it and enter the following code.

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

/**Toggles the red LED of the SES-board*/
int main(void)
{
    DDRG |= 0x02;

    while (1) {
        _delay_ms(1000);
        PORTG ^= 0x02;
    }
    return 0;
}
```

Compiling

Save the file by using the shortcut (Ctrl + s) and build it by clicking on the build button in the PlatformIO toolbar at the bottom of the window. In the output, you should see a *SUCCESS* message.

The compiled program is stored in your project directory under

`./.pio/build/<project_name>/firmware.hex`.

You will need to select this file for upload in the web interface later.

Git Version Control

You will be using the Git version control system during the exercise to hand in your solutions. To install Git, follow the tutorial found at

https://www.atlassian.com/de/git/tutorials/install-git .

On Windows, make sure that during the installation you select to use Git from the Windows command prompt.

C Compiler

Since we will compile and run C code not only on the microcontroller but also on your local computer, you should also install a C compiler toolchain.

C Compiler - Linux only

If you are using Linux, you probably have a GNU C Compiler (gcc) installed already. Open a terminal window and type `gcc -version`. If that command prints the compiler version, you do not need to do anything more. If not, install it with `sudo apt install build-essentials`

C Compiler - Windows only

First, you have to install a C toolchain (including compiler and libraries) to be able to compile your code to machine language. In this exercise, we use MinGW. MinGW, a contraction of "Minimalist GNU for Windows", is a minimalist development environment for native Microsoft Windows applications.

Open https://osdn.net/projects/mingw/releases/ and click on *mingw-get-setup.exe*. Download and execute the file. Follow the installation, use default settings, **don't use an installation path including whitespaces!** Once the Installation is complete, you are asked to select packages to install. Select *mingw32-gcc-g++-bin*. Then click on *Installation* → *Apply Changes* in the Menu bar. Finish the installation.

It is necessary to add the folder of MinGW's bin directory to the *Path* system variable of Windows. Add *MINGW_ROOT\bin* (e.g. *C:\MinGW\bin*) to the Path system variable. The tutorial at

https://www.computerhope.com/issues/ch000549.htm

explains the process in detail for different windows versions.

Visual Studio Code - Compile and Run C programs

Open Visual Studio Code, click on *Extensions* in section **A** (see Figure 1). Search for the *C/C++ Compile Run* extension and install it. Now, navigate back to the Project Explorer in VS Code and create a new file called *HelloWorld.c*. Paste the following simple C program into the file and save it. Then press F6 to compile and run the file. In the terminal output in the bottom of the window you should see the *Hello World*.

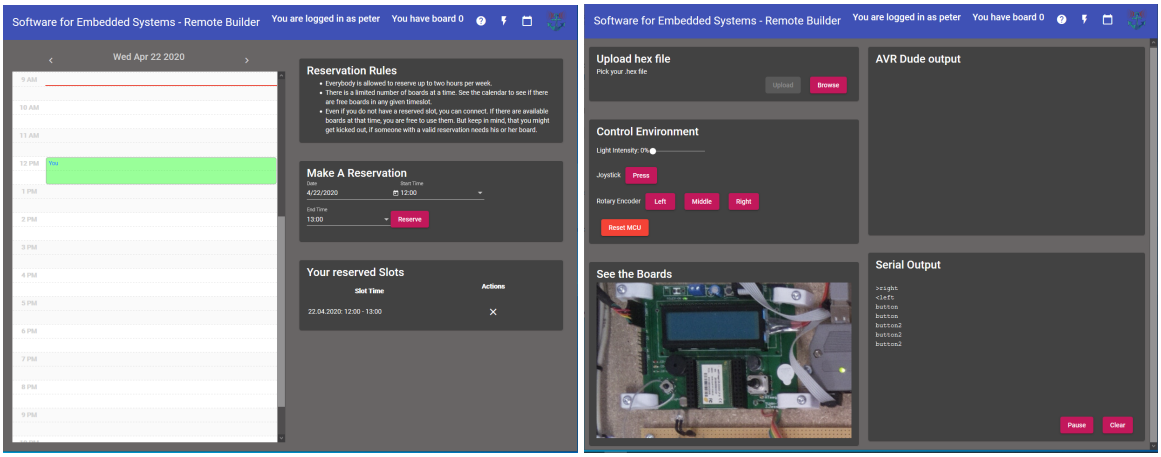


Figure 2: The scheduling view (left) and the Remote Lab Enviroment (right)

```c
#include <stdio.h>

int main(int argc, char** argv)
{
    printf("Hello World");
    return 0;
}
```

Upload Programs to the Microcontrollers

There are six microcontrollers available in our lab. Once you are signed up for the exercise in Stud.IP, we will send you credentials for the **remote flashing web application**. You can only access the web app from within the university network. Hence, you will have to use a VPN client. If you have not yet done that, configure the VPN from the TUHH https://www.tuhh.de/rzt/netze/vpn/anleitungen.html.

Once the VPN is running, use a browser—preferably Firefox or Chrome—to navigate to

http://seslab.smp.tuhh.de .

The browser will ask you for your credentials. **Do not use your usual TUHH credentials**, you will receive a dedicated username and password from us. After login you will see the scheduling view, as shown in Figure 2 (left).

Three white icons are shown on the top right corner of the screen. The first one, the question mark, leads directly to our forum in Stud.IP. If you have any questions or experience problems during the exercise, this is your first place to go. We will try to answer all questions and solve technical issues as soon as possible. Before asking a question in the forum, please quickly browse the existing topics, if someone has reported the same problem already.

By clicking on the flash, you will get to the remote lab environment, where you can upload (flash) your compiled programs to the microcontroller. The last icon, the calendar, will always bring you back to the scheduling view.

The Scheduling View

Since we do have only six boards, not everybody can work in parallel. You can use the application at any time, and if there is a board free at the moment, you will always be able to work with that. However, when there are more than six people connected at the same time, some will not be able to work with a board. When you are not using the board for a while, please be fair and close the application to give others a chance to work on it.

We have introduced a reservation system so that everybody has a minimum guaranteed work time each week. In this view, you see a calendar on the left, where you can see, at which times there are already blocked slots. If there are no blocked slots, you can reserve slots for yourself. During those booked times, you are guaranteed to have a board for yourself. The reservations are, however, limited to two hours a week per person.

You won't be able to book slots during the exercise times (Tuesday 13:00 to 18:00), we will do that for you.

Remote Lab Environment

In the remote lab environment (see Figure 2 (right)), you see multiple tiles for different functionality. The top left tile allows you to select a `.hex` file that you want to upload to the microcontroller. Once the upload is done, you should see the output of the uploading program in the *AVR Dude output* tile.

Below, there is the *Control Environment* tile. Some tasks in the exercise require you to read out sensors or react on button presses. In this tile, you can trigger button presses or change the brightness. Note, however, that due to the complexity involved in remotely trigger these events, we can only press the joystick middle button and not move it around. You can turn the rotary encoder to the left and right as well as press it down. Note that for both the joystick and the rotary encoder button presses, you can also keep it pressed by holding the mouse button down. If neccessary you can trigger a reset to the microcontroller by clicking *Reset MCU*.

In the left lower tile, you see a webcam stream of your board. Here, you can inspect the output on the LCD screen or observe the LEDs blink. Be aware that there is a short delay of one to two seconds in the stream.

Right next to it is the *Serial Output* tile. During the exercise you will use the serial interface of the microcontroller for debugging. In this view, you can inspect the serial output. Note that you have to start capturing the serial output by clicking on the *Play* button.