
	<p>Professorship Computer Engineering  <b>Automotive Software Engineering</b>          Prof. Dr. Dr. h. c. Wolfram Hardt          Dr. René Bergelt</p>	
	CE Virtual ECU - Manual	10-2024

## Contents

1. Introduction .....	1
2. Prerequisites .....	1
2.1. Setting up on Windows .....	2
2.1.1. Installing .NET 8.0 .....	2
2.1.2. Installing C compiler .....	2
2.1.3. Installing Visual Studio Code .....	3
2.2. Setting up on Linux .....	4
2.2.1. Installing .NET 8.0 .....	4
2.2.2. Installing C compiler .....	4
2.2.3. Installing Visual Studio Code .....	4
2.3. Setting up on MacOS .....	5
3. Developing for the CE Virtual ECU .....	6
Bibliography .....	8

## 1. Introduction

This manual will guide you in setting up the CE Virtual ECU, which is a software-based simulator of the ECU platform (based on the STMicroelectronics SPC560P microcontroller) used in Unit 1 & 2 in the Automotive Software Engineering practical. By providing you with this simulator, the Professorship Computer Engineering allows you to practice embedded C programming with regards to the specific ECU used in the practical as well as to prepare for the respective practical sessions at home.

## 2. Prerequisites

Before you can program for the Virtual ECU you need to setup a corresponding development environment on your computer. In general, the following things need to be installed:

- .NET 8.0 Runtime (needed to run the emulator itself)
- C compiler
- C IDE (we will use Visual Studio Code)

Based on the target computer system (i.e. Windows, Linux or MacOS) different steps for installing the prerequisites might be required. Please refer to the section which describes the procedure for the operating system you have.



Currently, the Virtual ECU is only supported on **64-bit systems**. So make sure that your computer has a 64-bit CPU and is running a 64-bit operating system. Usually, this should almost always be the case for most modern systems.

## 2.1. Setting up on Windows

The Virtual ECU can be run on Windows 10 and later. Previous versions of Windows might work but are neither officially supported nor tested.

### 2.1.1. Installing .NET 8.0

Make sure that the .NET 8.0 Runtime or SDK for **x64** is installed on your PC. The installer can be downloaded from <https://dotnet.microsoft.com/en-us/download/dotnet/8.0>. At the time of writing, the latest version is [8.0.10](#).

**Alternatively**, if you are using a package manager, usually there should be a package for .NET 8.0 you can install by invoking the corresponding command as follows:

**winget:**

```
winget install dotnet-runtime-8
```

**Chocolatey<sup>1</sup>:**

```
choco install dotnet-8.0-runtime
```

### 2.1.2. Installing C compiler

We will use GCC<sup>2</sup> as C compiler to compile the code for the Virtual ECU. As GCC is not available by default on Windows we need to install a corresponding development environment which contains it. While there are other options, for this guide we will use MSYS2 as it is very easy to setup and to be kept up-to-date.



If you already have a recent version of GCC & GDB installed (maybe from other sources), you can skip this step. However, make sure to update the paths in the project template's tasks.json and launch.json accordingly.

Basically, it is sufficient to follow the basic setup guide available on the homepage of MSYS2<sup>3</sup> as this includes the installation of GCC. However, one additional package for GDB needs to be installed as well. For clarity's sake, the relevant steps from the website are replicated here. If you need more details, please refer to MSYS2's complete official documentation.

1. Download the latest installer from [msys2.org](https://msys2.org)  
At the time of writing this is [msys2-x86\\_64-20240727.exe](#)
2. Run the installer. Installing MSYS2 requires 64 bit Windows 10 or newer.
3. Leave the installation folder as is; see Figure 1 for reference
4. When the installation is done, click Finish.
5. Now MSYS2 is ready for you and a terminal for the UCRT64 environment will launch as shown in Figure 2.
6. In order to install GCC & GDB please run the following command in this terminal and confirm the download and installation:

```
pacman -S mingw-w64-ucrt-x86_64-gcc mingw-w64-ucrt-x86_64-gdb
```

7. Once the installation has succeeded you can close the terminal

---

<sup>1</sup><https://chocolatey.org>

<sup>2</sup><https://gcc.gnu.org>

<sup>3</sup><https://www.msys2.org>

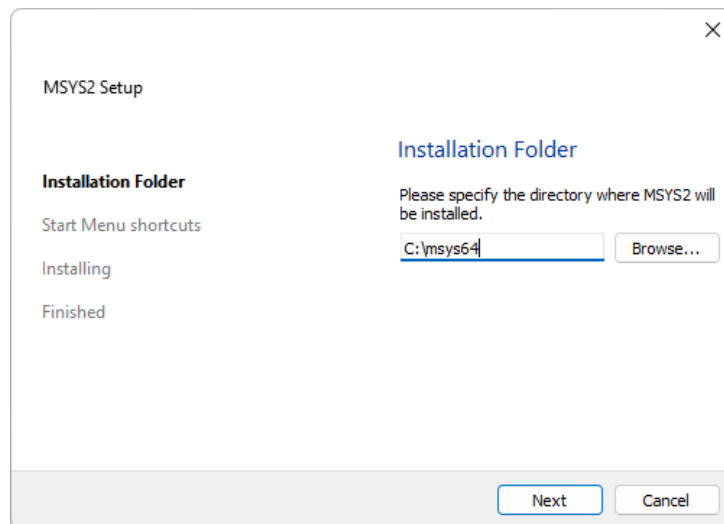


Figure 1: Specifying the installation path for MSYS2 [1]

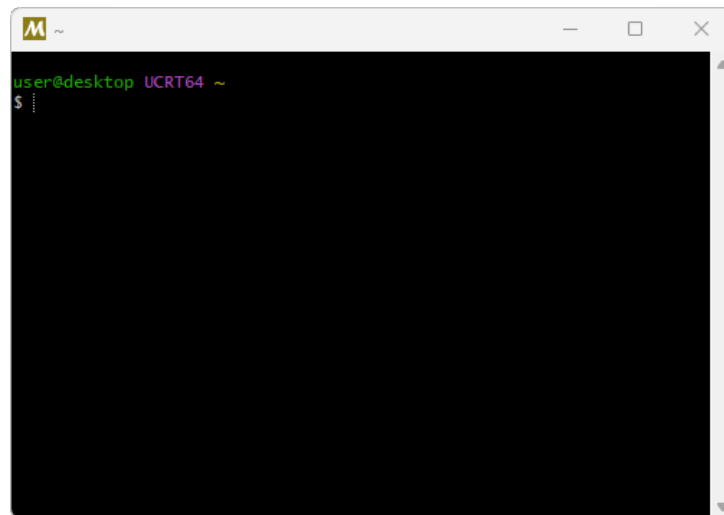


Figure 2: The MSYS2 terminal

### 2.1.3. Installing Visual Studio Code

As the provided template project for the Virtual ECU is a Visual Studio Code workspace (and since you will use Visual Studio Code in later units of the practical as well), we will now set it up to build programs for the Virtual ECU. Obviously, if you already have an installation of VS Code you can skip this step, just make sure that you are using one of the latest versions.

For this, download the latest installer from <https://code.visualstudio.com> by clicking on “Download for Windows” and execute it.

**Alternatively**, if you have a package manager available, run the corresponding command to install VS Code. For example:

**winget:**

```
winget install -e --id Microsoft.VisualStudioCode
```

**Chocolatey:**

```
choco install vscode
```

Congratulations, you have successfully setup the development environment for the Virtual ECU and you can continue with Section 3 to see how you can implement programs for the Virtual ECU for preparing for the ASE practical units 1 and 2.

## 2.2. Setting up on Linux

The CE Virtual ECU should work on most linux distributions based on Debian 9+ (Stretch and later), Ubuntu 16.04+ as well as Fedora 30+.



Based on the actual Linux distribution you use, the provided steps or commands might differ.

### 2.2.1. Installing .NET 8.0

Browse to <https://dotnet.microsoft.com/en-us/download/dotnet/8.0> and follow the setup instructions for your type of linux distribution. Only the “.NET Runtime” is required to run the Virtual ECU. For many distributions .NET can be installed over the official package manager. For instance, if your distribution is using **apt** you can run the following to set everything up:

```
apt install dotnet-runtime-8.0
```

### 2.2.2. Installing C compiler

Usually, most Linux distributions already come with gcc and gdb. You can check this by running:

```
gcc --version
```

If you get a similar output as follows, you are already finished with this step:

```
gcc (Ubuntu 13.2.0-23ubuntu4) 13.2.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

If the command is not recognized, please refer to the documentation of your distribution to install the gcc compiler toolchain as well as gdb.

### 2.2.3. Installing Visual Studio Code

Please follow the official guide on installing Visual Studio Code on Linux: <https://code.visualstudio.com/docs/setup/linux> .



While you can technically use any IDE or editor to program the Virtual ECU, we currently only provide a project template for Visual Studio Code since it is available cross-platform. If you want to use your own toolchain, please refer to the tasks.json and launch.json files in the Visual Studio Code project template to extract the corresponding compiler and gdb commands (see Section 3)

### **2.3. Setting up on MacOS**

While MacOS is technically supported, we cannot offer any extended support or guidance due to lack of Apple hardware for testing. The Virtual ECU has not been tested on the ARM-based products of Apple (Apple M1 etc.). If you have such a system, we would be glad if you could offer us feedback regarding the experience in setting it up.

Currently, we are still working on the setup guideline for Mac OS. As an alternative, there is always the possibility to use a virtualized Linux system on your Mac in order to execute the Virtual ECU, e.g. by using VirtualBox<sup>4</sup>. You can then follow the steps in Section 2.2 to set up the virtualized Linux accordingly.

---

<sup>4</sup><https://www.virtualbox.org>

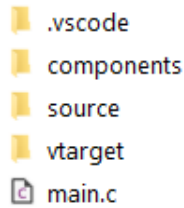


Figure 3: Folder structure of the template project

### 3. Developing for the CE Virtual ECU

Once you have installed all pre-requisites you are ready to use Visual Studio Code to implement and build programs for the CE Virtual ECU. For this, please download the zipped template project which is provided to you where you downloaded this manual. Once extracted, you should have the same folder structure and files as shown in Figure 3. This is basically the same project structure of the SPC5 Studio IDE used in the practical with the actual hardware.

After launching Visual Studio Code, click on File >> Open Folder... and navigate to the extracted folder of the template project and confirm. When the workspace has been opened, you should see the main window similar to what is shown in Figure 4a. On the left side, the “Explorer” window shows you the structure of the project as shown in Figure 4b. If it has not been opened automatically, click on the “main.c” file in the project explorer which contains the entry point of the ECU’s program. If you are asked to install the “C/C++ Extension Pack” when opening the “main.c” file as shown in Figure 4c, please do so. Alternatively, make sure that the “C/C++ Extension Pack” is installed and up-to-date through VS Code’s extension manager (see Figure 4d).

#### Additional steps on Windows:

If you changed the installation path of MSYS2 or want to use a different source for GCC, please set the correct path in the file `.vscode/settings.json` in the variable **build\_tools\_directory**.

#### Additional steps for Linux / MacOS:

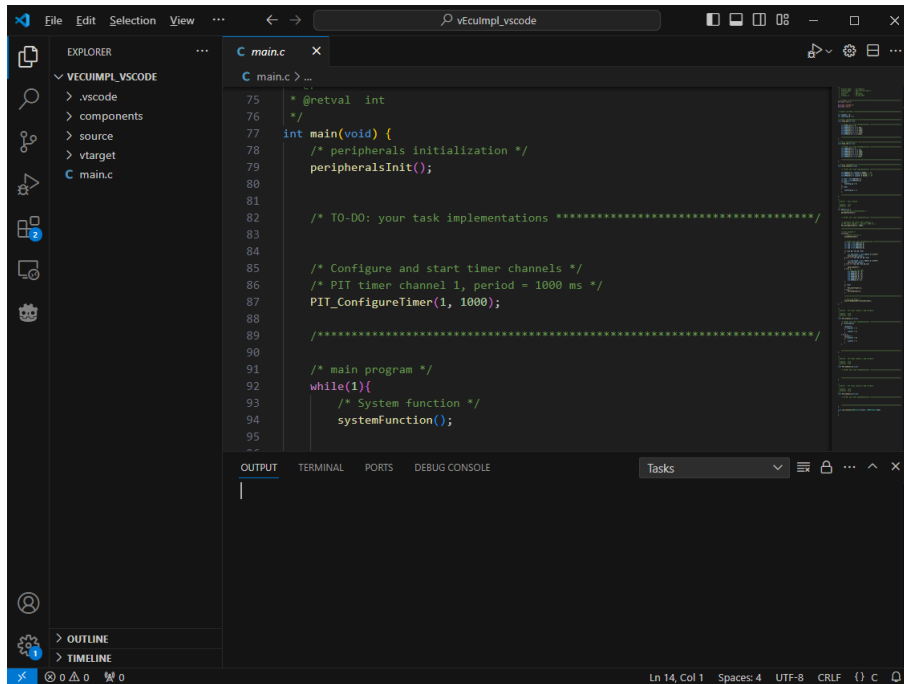
Please set the correct path in the file `.vscode/settings.json` in the variable **build\_tools\_directory** which points to the location of gcc and gdb. In most cases, you can just set it to an empty value (i.e. “”) since the compiler tools are globally accessible:

```
{
  "vecu":
  {
    "build_tools_directory": ""
  }
}
```

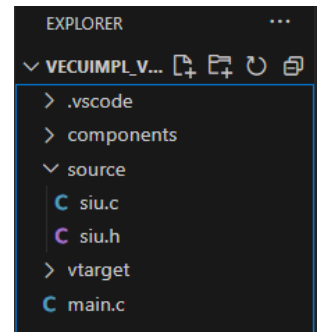
In addition, in order to be able to run the Virtual ECU interface for debugging you need to mark it as executable. To do this, run the following command with the appropriate permissions in the project’s root folder:

```
chmod +x vtarget/linux/vECU.UI
```

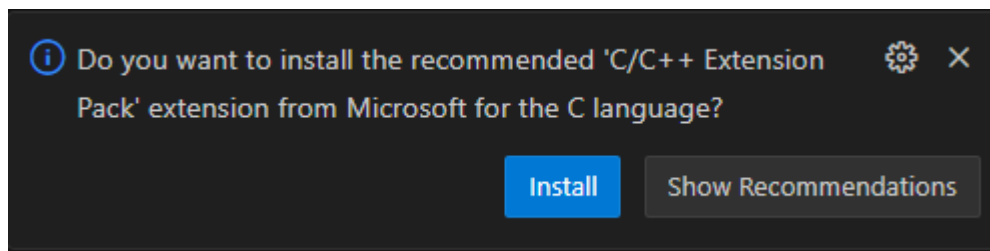
The project is now correctly set-up for compiling and running the code on the CE Virtual ECU which is provided as part of the template project.



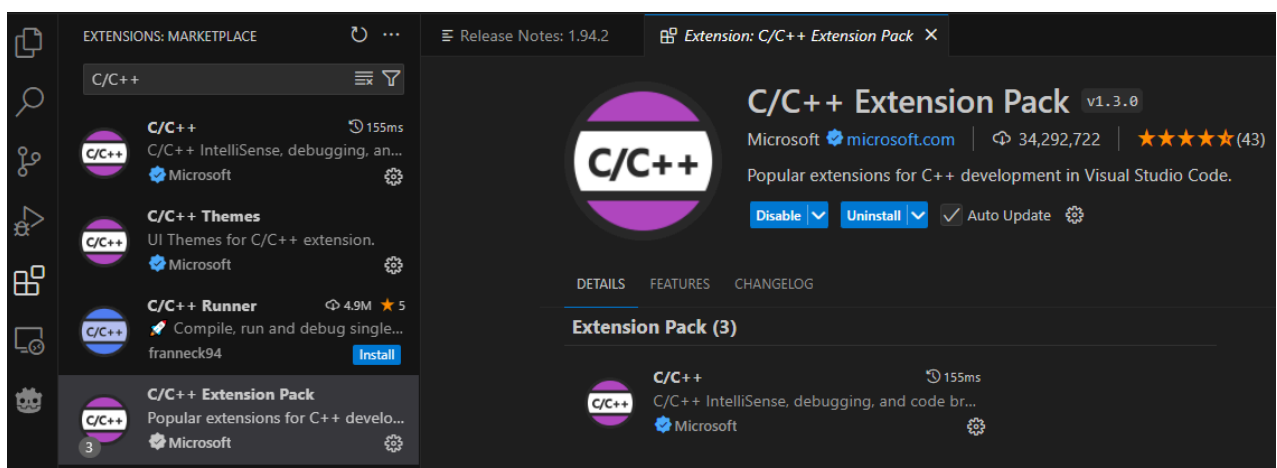
a): Main window of the IDE



b): Project explorer



c): Language Extension prompt



d): C/C++ Extension Pack

Figure 4: Views of Visual Studio Code

In order to **build and run** the program, either press “F5” or switch to the “Run and Debug View” (see Figure 5) and click on the green triangle next to “Launch with Virtual ECU”. If this is successful your program will be automatically *flashed* to the CE Virtual ECU which automat-

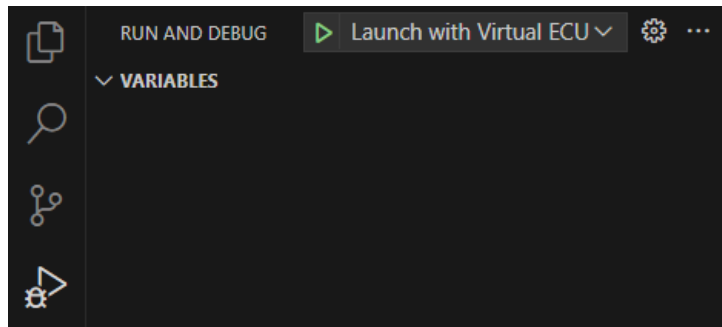


Figure 5: VS Code's Run and Debug View

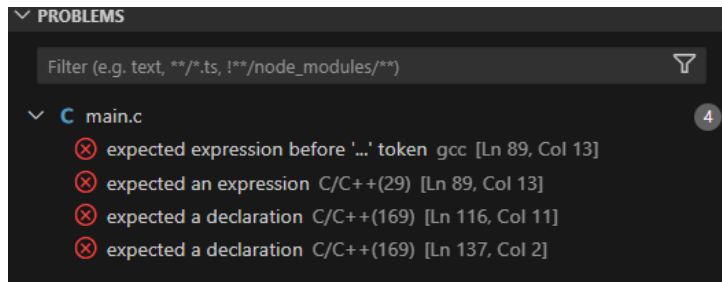


Figure 6: VS Code's Problems View

ically opens in a new window. Please notice that it resembles the look of the front panel of the hardware ECU which is used in the ASE practical, see Figure 7 for reference. As with the real ECU you need to turn on the Virtual ECU by switching the power switch at the bottom to on (i.e. from 0 to —). By doing so after building the template project, the *Onboard LED* in the top right corner should start blinking to indicate that the code is indeed running.

If there are any errors in your code they will be shown in the problems view similar to Figure 6. In addition, the errors will be highlighted in the code itself. Fix the shown errors before trying to rebuild the program.

Close the Virtual ECU, by either clicking on the window's X button or by turning the Power Switch back to off.

You can now start to implement programs which access the buttons, LEDs and additional sensors. You can follow the normal task description of Unit 1 as the CE Virtual ECU is programmed in exactly the same way as the hardware ECU used in the practical (this also applies to the need to correctly configure the pins as inputs or outputs). In general, you should only make changes in **main.c** and the files which are in the *source* directory (e.g. **siu.h**, **siu.c**).

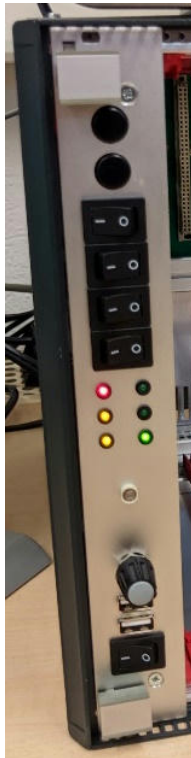


If you prepared source code using the Virtual ECU **at home** you can bring these files (**main.c**, **siu.h**, **siu.c**) to the practical and copy the contents into the actual hardware ECU project of SPC 5 Studio.

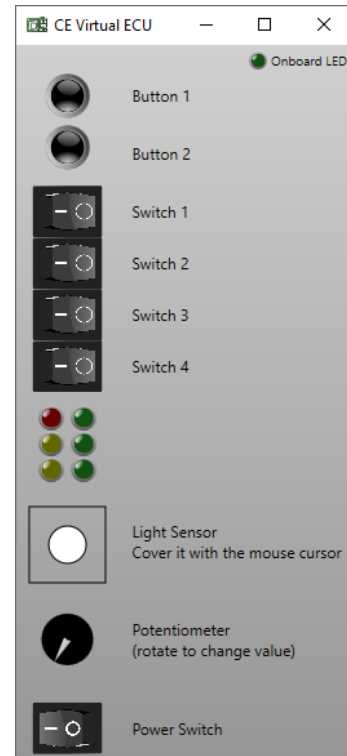
## Bibliography

[1] MSYS2, "MSYS2 - Getting Started." [Online]. Available: <https://www.msys2.org/>





a): The Hardware ECU's front panel



b): Window of the Virtual ECU

Figure 7: Comparison of actual hardware used in the practical and Virtual ECU