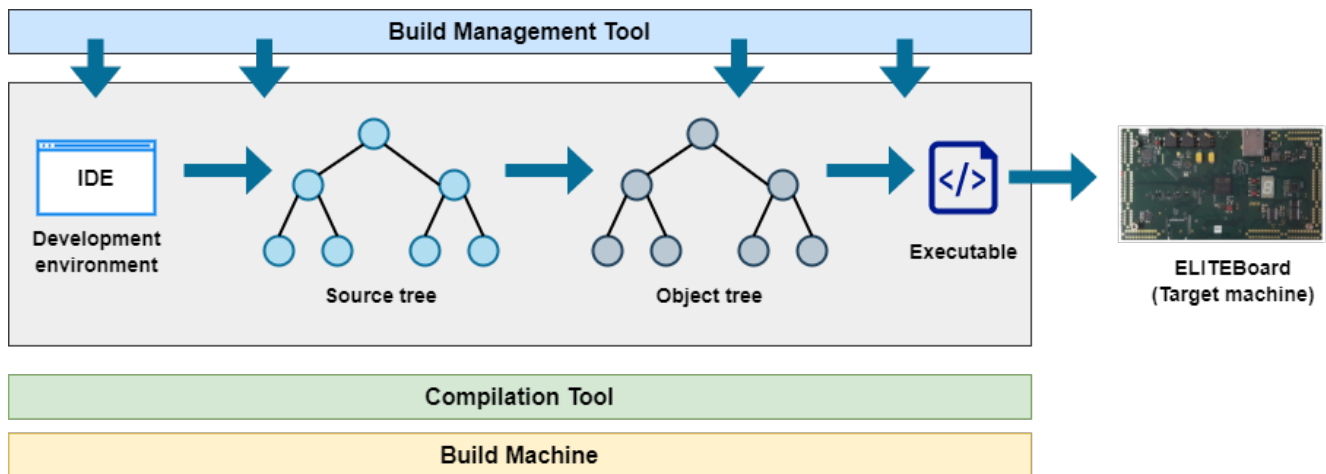


# Lab 0: ELITEBoard Set-up

## 1. Description of the Build System



In the above diagram, the functionality of a build system is shown. It will give you some basic idea about the tools that we are going to use in our lab. The build process works as follows:

1. We use an Integrated Development Environment (IDE) to write our program code. For this practicum, we will use Visual Studio Code (VS CODE) IDE and the C programming language.
2. We will use a build tool (GNU make) that will compile the source code (written in C) into object code and create an executable program.
3. The build machine is the computer you will use to build our code, i.e., your personal computer (Windows/Linux/Mac).
4. The target machine is the ELITEBoard, where we deploy and run our executable program.
5. We also need a cross-compiler (arm-gcc) because the system architecture of our build machine (CPU of your PC) is different from that of the ELITEBoard (ARM architecture).

**For all installations, use a path without spaces or special characters! In rare cases, this can cause issues that are difficult to detect.**

Example: create a folder for your installations on `C:\data\nes` with subfolders for `vscode`, `mingw`, `cubemx`, `arm`.

**Do not** install any tools on paths that contain spaces. This includes `C:\Program Files` (in german-language Windows visualized as `C:\Programme` in the frontend) and especially any shared folders, such as Dropbox, OneDrive, etc.

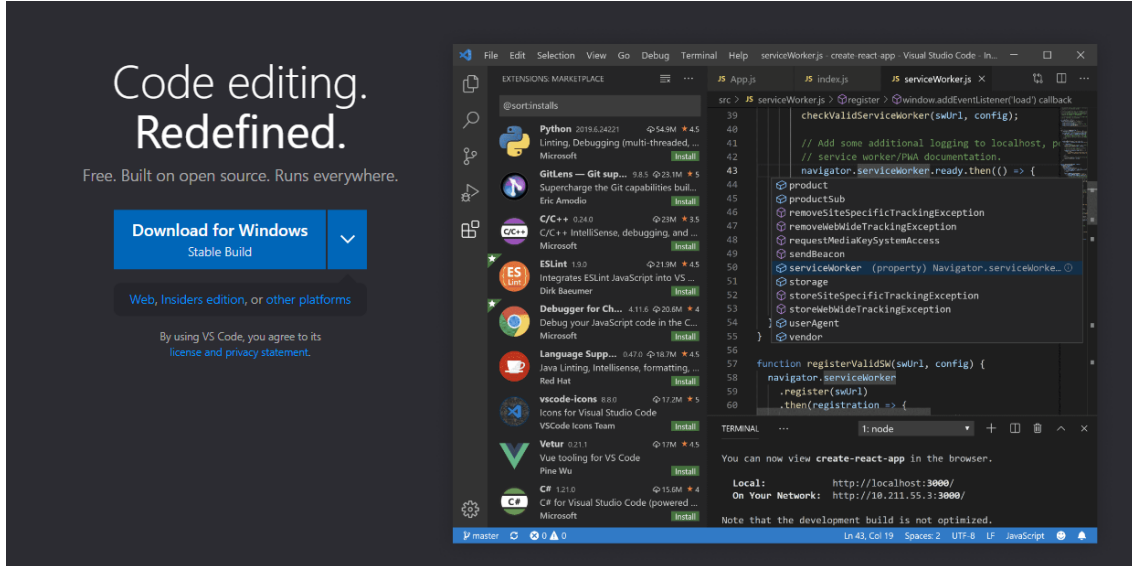
**Having a common setup helps to spot errors quickly** and the instructions reflect our experience of avoiding common issues as we repeated the practicum over the past years.

## 2. Setting up the Development Environment

Goal: We will provide a VS Code project. This IDE is also used for communication with the ELITEboard.

### 2.1. Download & Install Visual Studio Code (VS Code)

Use the link to download the VS Code IDE -> <https://code.visualstudio.com/download>. Download the necessary package based on your operating system. Run the downloaded file and install the IDE, e.g., on the path C:\nes\vscode. Use default settings for the installation.

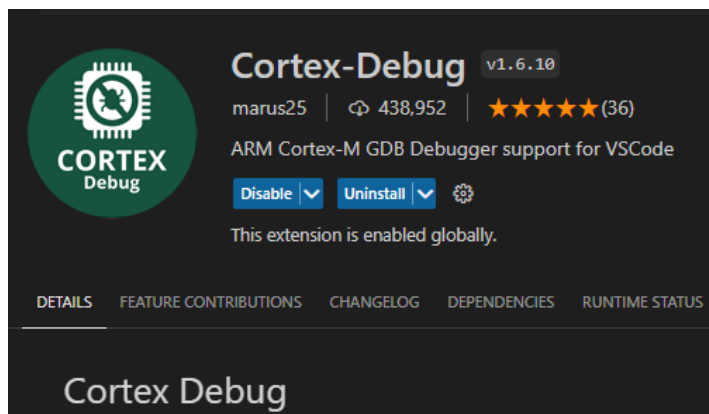
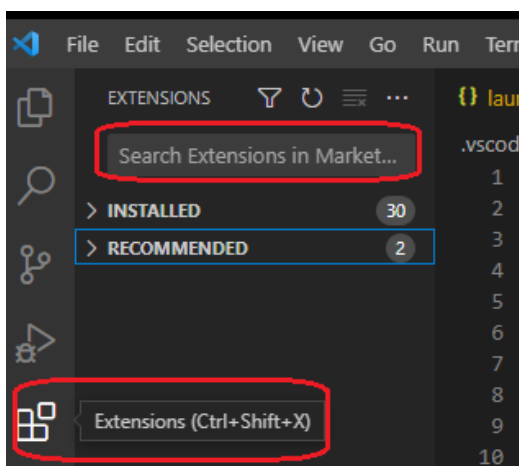


### 2.2. Download & Install VS Code Extensions

#### 2.2.1. Install the Cortex Debug extension for VS Code (current version 1.12.1)

The **Cortex-Debug** extension is required to communicate with the ELITEboard via the VS Code and debug the program code in run-time.

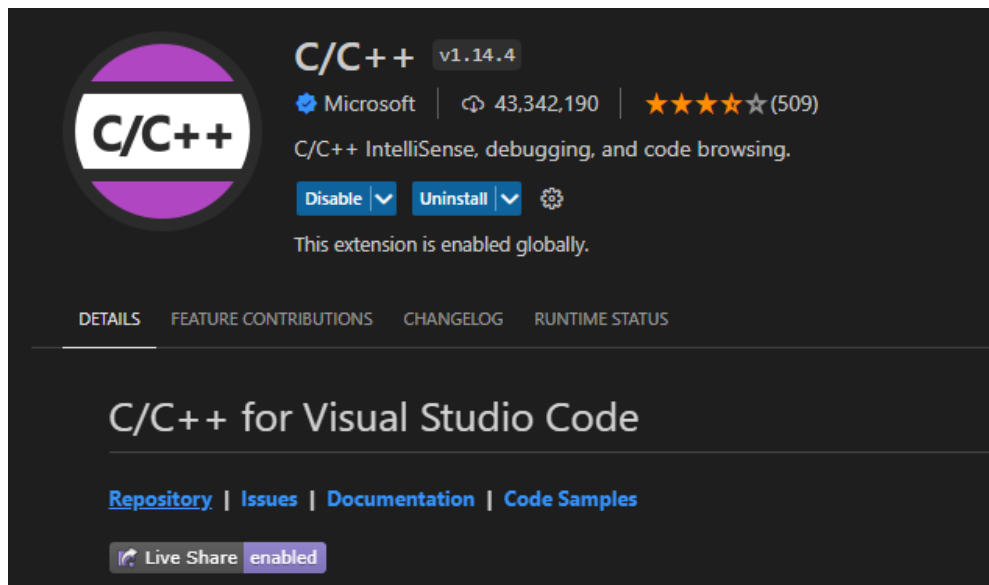
1. Start the IDE (you have already installed it)
2. Click on the extensions tab on the left or press CTRL+SHIFT+X
3. Search for **Cortex Debug** extension
4. Install the extension



## 2.2.2. Install the “C/C++” extension in VSC from the marketplace (current version 1.17.5)

The **C/C++ extension** is required for language support in the VS Code IDE.

1. Search for C/C++ extension
2. Install the extension (you may need to restart the IDE after installation)



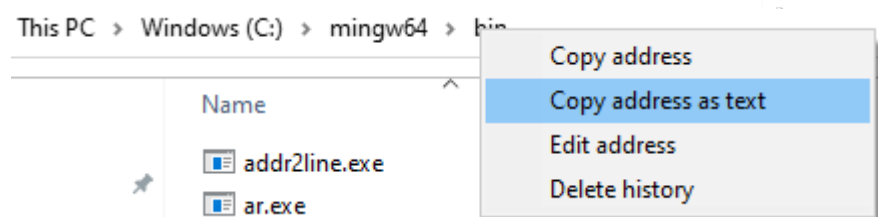
## 2.3. Download & Install the Build System and Cross Compiler

### 2.3.1. Install MinGW-W64

Goal: a `make` command has to be available on your system.

For Windows:

1. Go to the link -> <https://github.com/nixman/mingw-builds-binaries>
2. Use the MinGW-W64 online installer that you can download directly from this link
3. An executable called **mingwInstaller** is downloaded
4. Run the installation with default settings but **install to a path such as C:\nes\mingw**. If you know that you have a 64-bit architecture, you can select it in the respective step.
5. Locate the installation folder in your explorer. Within the subfolder **bin**, you can find a list of .exe files. It should look like the folder in the screenshot on the right.
6. Find the “**mingw32-make.exe**” file (selected in the screenshot), make a copy of the file, rename the copied file to “**make.exe**”
7. Copy the address of the bin folder, it should look something like -> C:\nes\mingw\bin. We need this in the next step to set the



PATH variable.

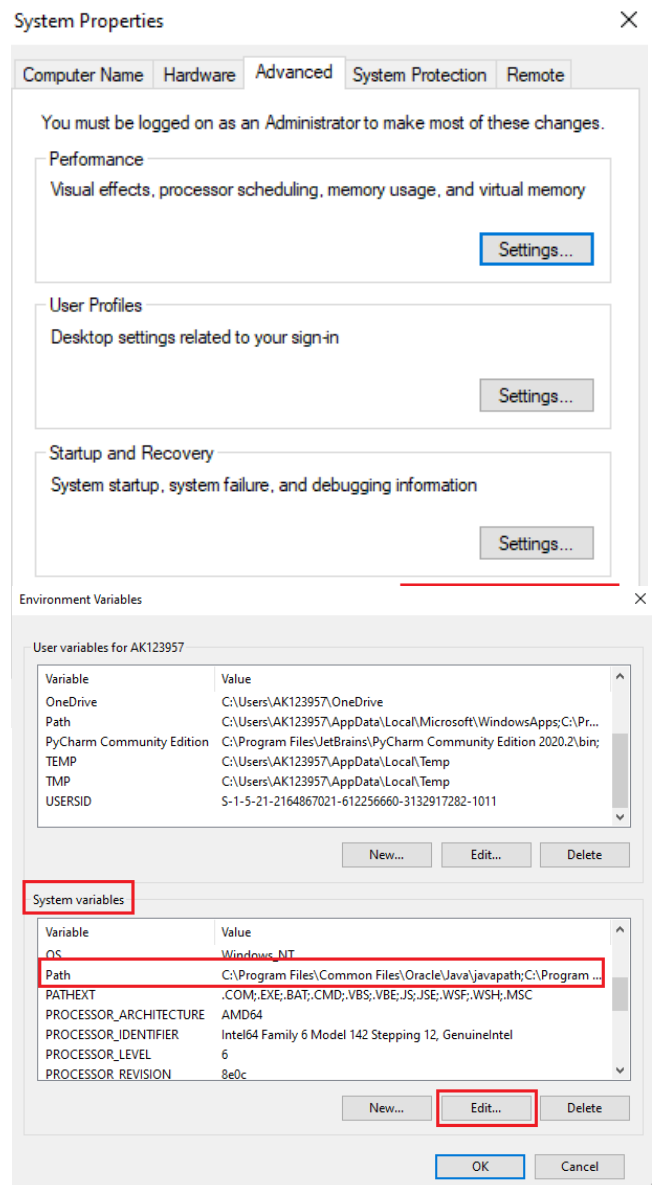
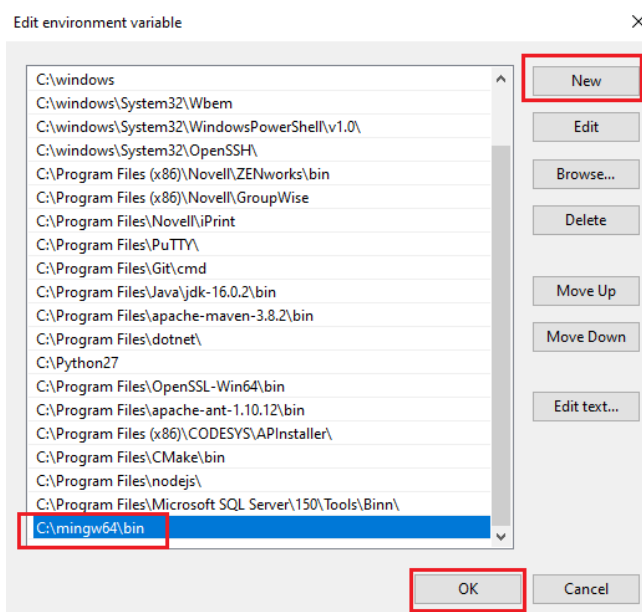
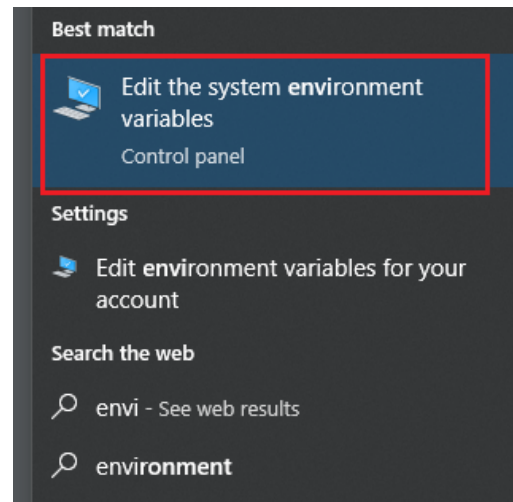
**For MacOS:** Install Xcode Command Line Tools (CLT) using `xcode-select --install`

**For Linux:** Use the following terminal command: `sudo apt-get install build-essential`

### 2.3.2. Update the PATH variable

#### For Windows:

1. Click on the windows start button and search for “**Environment variables**” (in German: Umgebungsvariablen)
2. You should see an option called “**Edit the system environment variables**”
3. Click on it and a small window will open, click on the “**Environment Variables**” button
4. Next, you should see all the environment variables e.g. system and user variables
5. Under **System variables**, find the **Path** variable
6. Select the **Path** variable and click on the **Edit** Button (alternatively double-click on the Path variable). It will open another window with a list of path variables.
7. Click on the **New** button and it will create an empty line
8. Paste the address of the bin folder from the previous step (**C:\mingw64\bin**) This is the folder that contained your copy of make.exe
9. Click **OK** and close the window



### 2.3.3. Validate that the installation was successful

- Open the command prompt (either: windows button + R, opens the Run window, type cmd and enter or: type **terminal** into the search field)
- Type **make -v**, you will see the following output

```
C:\Users\AK123957>make -v
GNU Make 4.2.1
Built for x86_64-w64-mingw32
Copyright (C) 1988-2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

An error could occur if the PATH variable is not set correctly.

- (1) restart the terminal, the PATH variable is only updated when launching an application
- (2) check whether the address is correct
- (3) check whether you have downloaded the correct MinGW version

More on setting up environment variables in Windows:

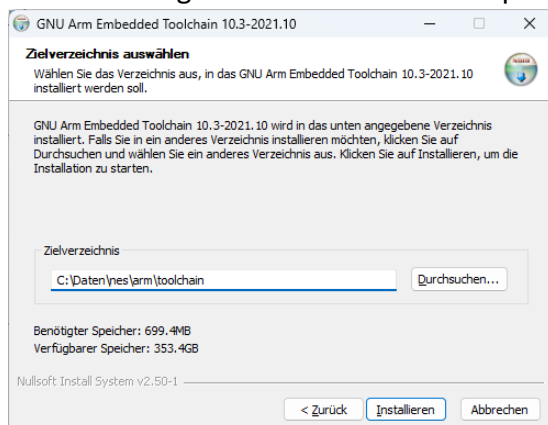
<https://geekflare.com/system-environment-variables-in-windows/>

For Linux: <https://www.baeldung.com/linux/path-variable>

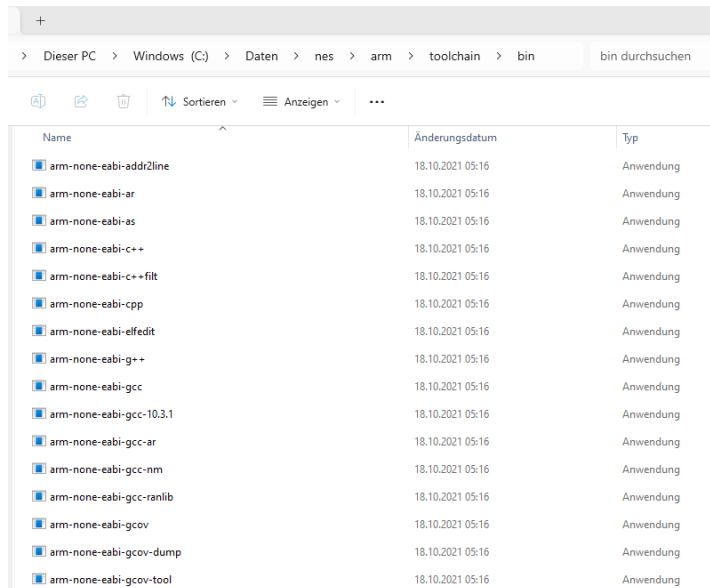
For Mac: <https://techpp.com/2021/09/08/set-path-variable-in-macos-guide/>

### 2.3.4. Install GCC cross compiler (gcc-arm-none-eabi-)

1. Use the link -> <https://developer.arm.com/downloads/-/gnu-rm> Although the page is marked as deprecated it contains a cross-compiler that is compatible with our setup.
2. Scroll down to the Downloads section
3. You can see a list of downloaded files for different operating systems
4. For Windows, download the “**gcc-arm-none-eabi-10.3-2021.10-win32.exe**”  
For Linux/Mac, download the appropriate installation package
5. The **version should be 10.3-2021.10**, this is important, newer versions don’t work well with the Cortex-debug extension.
6. Run the downloaded executable file and install the program. **Choose an installation path** such as `C:\nes\arm` because spaces, -, or . in the path often cause problems. Note that the default name of the installation folder is “10 2021.10”, so you need to **edit the installation path**. Use default settings for other installation options. Press Finish.



- After finishing the installation process, go to the **installation folder** and **copy the address** of the bin folder. If you accidentally have a path that contains elements such as 10 2021 .10, you can edit the names of the folders. The contents should look as follows:



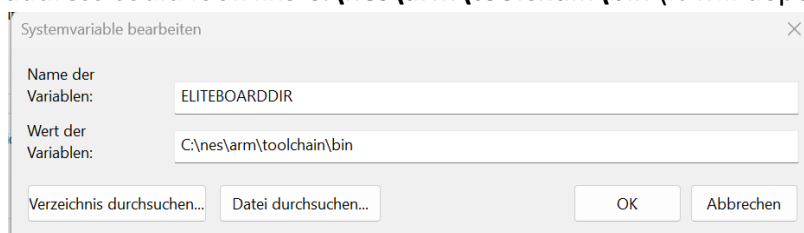
Copy the path again for the next step.

### Create an environment variable that locates the installation of the cross-compiler.

Goal: The provided sample projects use this variable to locate your installation path.

#### For Windows:

- Follow the steps described before and open the “**Environment variables**” panel
- Under System variables, click on the New button, it will open a small window. In this step, we are creating a new variable rather than editing the path variable!
- In the **Variable name** field, write -> **ELITEBOARDDIR (the exact name is important)**
- In the **Variable value** field, copy and paste the address of the bin folder from the previous step. The address could look like **C:\nes\arm\toolchain\bin** (it will depend on your installation location)



- Additionally, **add the same path** to the PATH variable.
- Click **OK** and close the window

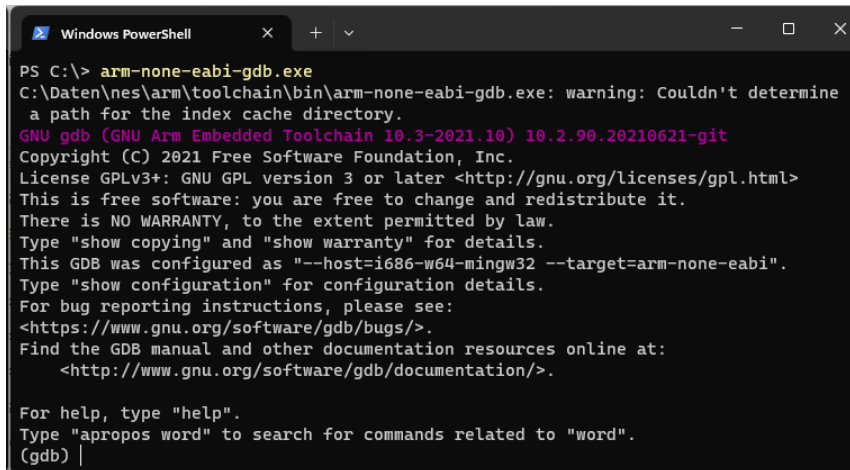
**For Linux:** The path of **ELITEBOARDDIR** should point to the folder where your “gcc” folder is located.

You can find instructions on setting the path variable on Linux (and verifying that it worked correctly) here: <https://www.serverlab.ca/tutorials/linux/administration-linux/how-to-set-environment-variables-in-linux/>

Use **sudo** to set the variables. Otherwise, they might be removed once you close the terminal!

### 2.3.5. Validate that the installation was successful

1. Restart the terminal
2. Type arm-none-eabi-gdb.exe (or equivalent for your operation system)
3. Check the output, it should look like this:



```
PS C:\> arm-none-eabi-gdb.exe
C:\Daten\nes\arm\toolchain\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine
a path for the index cache directory.
GNU gdb (GNU Arm Embedded Toolchain 10.3-2021.10) 10.2.90.20210621-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) |
```

## 3. Install the STM32Cube Project Generator

Goal: We use this tool to generate C code based on a project configuration. This allows us to configure the elements of our microcontroller in a graphical tool rather than by writing C code manually.

1. Go to the link-> <https://www.st.com/en/development-tools/stm32cubemx>
2. Press the “**Get Software**” button and select your operating system.
3. You need to create an account to download the tool. Make an account by providing your name and a valid email address.
4. Click download and the download link will be sent to the email address.
5. Unpack the file and run the installation.
6. Accept the license agreement and confirm that you have read the privacy policy.
7. Start the CubeMX software. On the Home screen, run “**Check for updates**”. If any potential updates are suggested, please install them.



# Build Empty Project from Template Project

## 1. Download and Unpack the Template Project

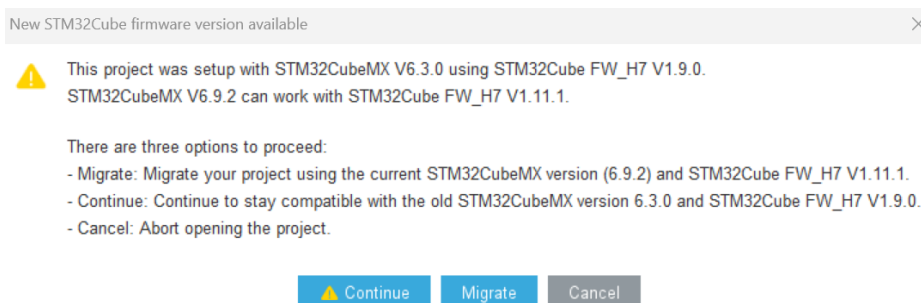
1. Download the Template Project from Moodle
  - Unpack the folder, e.g., to `C:\nes\projects`. Inside you will see a **Project folder** and the **STM32H743x.svd** file
    - a. The **STM32H743x.svd** file is a configuration file required for the debugger (cortex gdb). It enables the communication between the board and your PC
2. Inside the **Project folder**, you will see a **.vscode** folder and a **template.ioc** file
  - a. The **.vscode** folder contains all the JSON files for VS Code-related configurations, environment variable configurations etc.
  - b. **template.ioc** file is an empty CubeMX project file that will be used for code generation
3. Rename the Project folder to Startup (or something of your choice) and rename the **template.ioc** file to **startup.ioc** (**same name as project folder**)
4. Alternatively, you can create a new folder and copy and paste the **Project folder** and the **STM32H743x.svd** file. In this way, you will have a copy of the template project and don't have to download it every time)

The svd file is expected in the parent folder of your project/ioc file. A single svd file is sufficient for multiple projects.

## 2. Download and Unpack the Template Project

Goal: After the downloading and initial set-up is done, we will use the CubeMX code generator to generate some basic skeleton code for our project from the empty template.

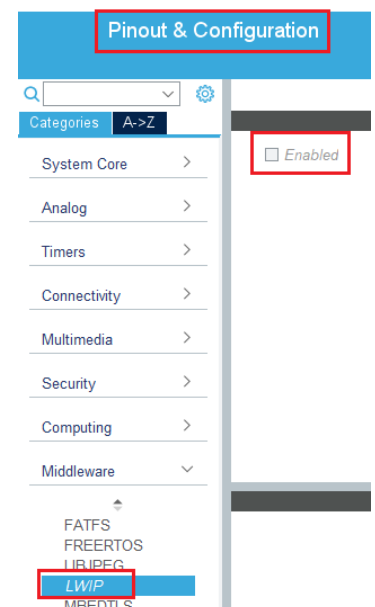
1. Open STM32CubeMX
2. Go to File -> Load Project -> select your <project name>.ioc file from step 4 -> Open
3. You might see the following notification window. Just click **continue**.



Note that it may ask you to register an account and to download a package.

4. In the tab **Pinout & Configuration**, select the part Middleware and look for the entry LWIP. You can also search for it in the searchbox on the top. Deselect the checkbox if LWIP is "Enabled".
5. Go to the tab Project Manager -> Code Generator (tab on the left) -> Check "Copy all used libraries into the project folder". Generate the Initialization code by clicking **GENERATE CODE** (see screenshot below). If there are warnings, please ignore them and press "yes".

**GENERATE CODE**

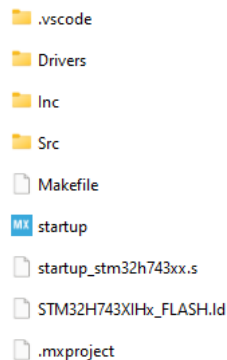


6. If the tool asks to download project related/board-related libraries, confirm the download (typically ~1 GB). You will need these libraries to program the board.



### 3. Compiling and Deploying an Empty Project

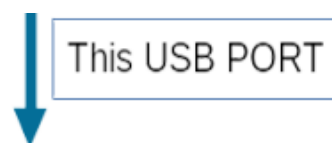
The code generation process might take a while to finish. When the code generation process is finished, you can close the CubeMX application and open the project folder. You will see some new files and folders are generated by the CubeMX software. The project folder now contains a pre-configured project for programming on your ELITEboard. The content of the folder should look like this:



- **Drivers (folder):** It contains the source code and header files for HAL and other drivers
- **Src (folder):** It contains all the source code files for protocols, input/output access etc.
- **Inc (folder):** It contains all the header files
- **Makefile:** This file contains the build description

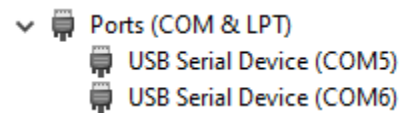
The next step is to compile the generated source code and create an executable file.

1. Open your project folder in VS Code by right-clicking on the folder and clicking “**Open with Code**”. Alternatively, you can open the VS Code-> Click Files -> Open Folder -> Select the project folder
2. If asked, choose that you trust the authors.
3. Locate **launch.json** in the folder .vscode in Explorer on the left sidebar.
4. Open launch.json by clicking on it.
5. Locate the “**executable**” key in launch.json and rename the templatnew.elf file to <your project name>.elf (e.g., startup.elf). The name of the executable file should be the same as the project name.
6. Connect your board to the laptop. The USB-C port is located in the centre of the board (next to the LAN port).




7. Find the COM port and update in the JSON file:

- In windows**, go to device manager, In **Ports (COM & LPT)**, you should see two COM ports
- Use the first COM port and update the value in the JSON file for the key **BMPGDBSerialPort** under the windows section



```
"windows": {  
  "armToolchainPath": "${env:ELITEB...  
  "BMPGDBSerialPort": "\\\\.\\COM8"  
  "preLaunchCommands": ["mon connec
```

- For Linux:** run `sudo dmesg` and you will see the COM ports. Update the value of the key under the Linux section

8. Click the **Run & Debug**  button on the activity bar (on the left side of the window)

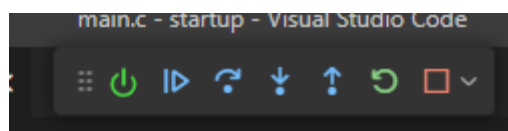
9. Locate the run button: Compile and deploy your project onto the board by pressing the play button. Alternatively, you can compile and deploy from the menu bar by going to Run -> Run Without Debugging.



10. If no major issues occurred (**see FAQ for resolving them**, check both Terminal and Debug Console for errors), you will see the program execution started and stopped after the main function

```
76  */  
77  int main(void)  
78  {  
79      /* USER CODE BEGIN 1 */  
80  
81      /* USER CODE END 1 */  
82
```

11. You can continue the execution by clicking the continue button. You can also click the stop button to stop the execution.



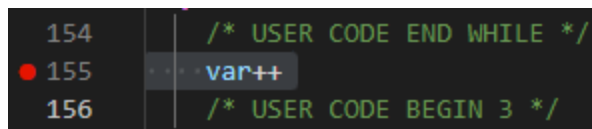
## 4. Write some Code

1. Open the **main.c** file (it should be inside the Src folder)
2. Go to the **while (1)** loop inside the main function and add your code in between the comments indicating the space for USER CODE.

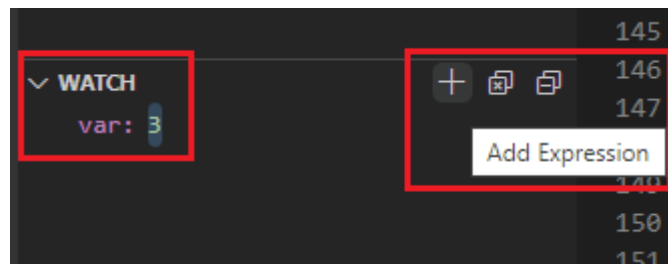
```
/* USER CODE BEGIN 2 */
int var = 0;
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
    var++;
    /* USER CODE BEGIN 3 */
}
```

3. Add a breakpoint in the line `var++`.



4. In the Watch window (left), watch the value of `var` to ensure that the execution is correct (it should increase after every cycle). This is to check that the compilation went correctly.



5. You have to manually add the variable by clicking on the plus sign (Add expression)
6. If you can't see the value or if it says optimized out, go to the **makefile** -> find the **OPT** option and change the value from **-Og** to **-O0**

## 5. Organizing your projects for Practicum

For each exercise create an individual project folder. Inside the project folder, create separate projects for an individual task. For example:

- NES\_Projects (parent directory for all projects)
  - Lab\_0\_Startup
  - Lab\_1\_GPIO (all tasks of lab 1)
    - Task\_1 (individual project folder)
    - Task\_2 (individual project folder)
  - STM32H743x.svd (this file should be at this level)

**If you have space in your project folder path, you might see some errors. Better to create project folders without any space**

**For submission of your tasks, create a copy of your project folder (e.g. Lab\_1\_GPIO), remove the build and the Drives folder from individual task folders and create a zip file.**