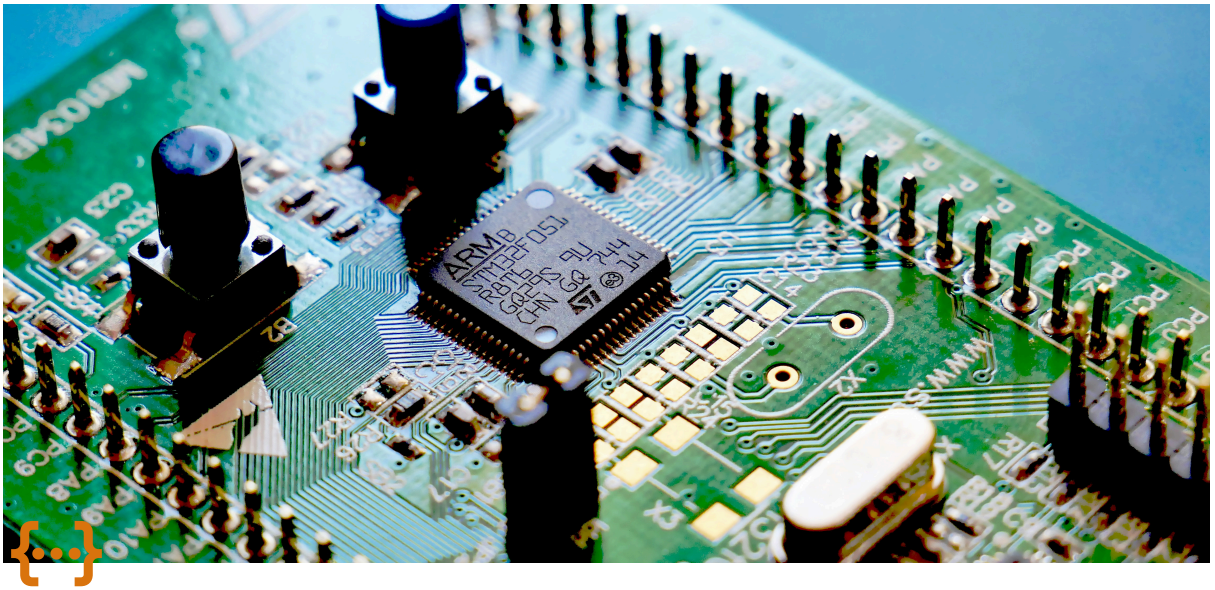


DESN2000 (Computer Engineering) Getting Started With STM32cubeIDE

Author: William Chan and Riley Haydon *(updated from Riley Haydon's guide which was adapted from Dr. Hammond Pearce's blog post https://01001000.xyz/2020-05-11-Tutorial-STM32CubeIDE-Getting-started/)*

Created: T2 2026

STM32CubeIDE is STMicroelectronics's integrated development environment (IDE) for their microcontrollers. The development you have got by now has a NUCLEO-F303RE board equipped with a 32-bit STM microcontroller. The STM32CubeIDE includes the necessary drivers, compilers, and a debugger, all in a single package.

Install STM32CubeIDE

You can download and install the latest version of CubeIDE from the link below:

https://www.st.com/en/development-tools/stm32cubeide.html

It's free, although you do have to sign up for an account / give them your email address. The good thing is it is supported on Windows, Linux and MacOS. Download the package for your operating system and install the package as you would install any other software on your computer.

When launching the app for the first time, **you may** be prompted to create/login to an account. If not you can access the login by going ***mySt*** → ***login*** in the menu bar. I know creating an account to use an IDE is a pain and unnecessary, but **some features will not work** if you do not log in.

Install STM32CubeMX

As of STM32CubeIDE Version 2.0, STM32CubeIDE and STM32CubeMX need to be installed **separately**, where previously they were integrated together. Whilst STM32CubeIDE is necessary for building and deployment, STM32CubeMX is optional (but highly recommended). STM32MX provides a graphical interface for configuring microcontroller hardware which includes initialising pins. Most students find this graphical tool much easier to use and thus we recommend you install it.

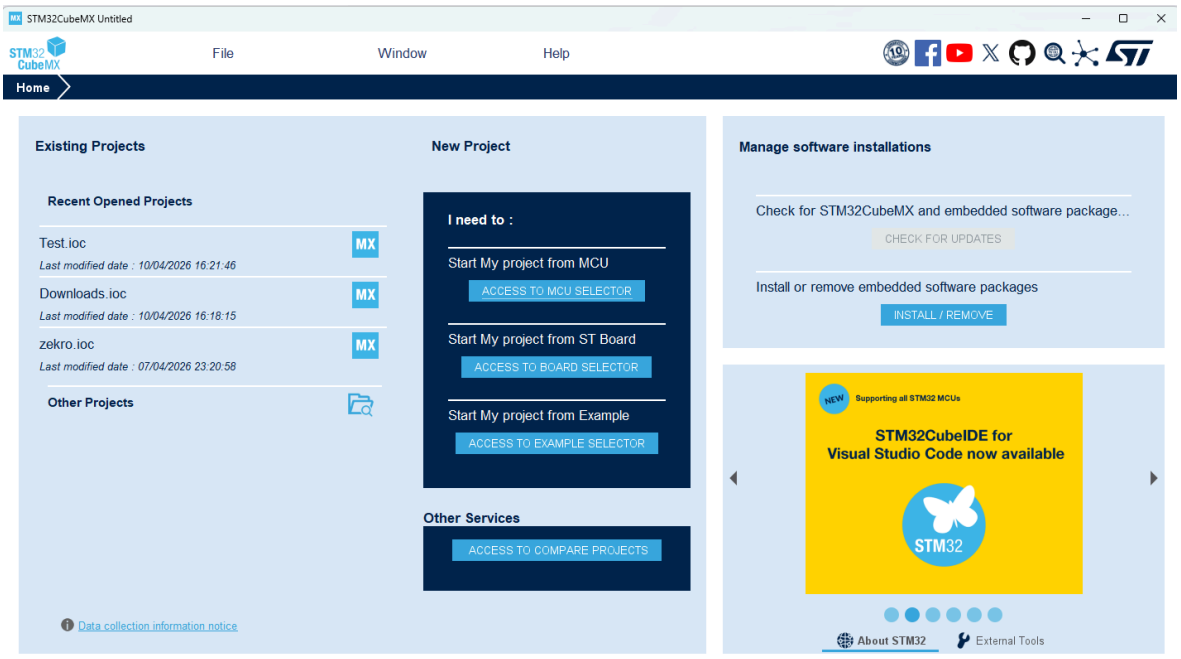
You can download and install the latest version CubeMX from the link below:

https://www.st.com/en/development-tools/stm32cubemx.html

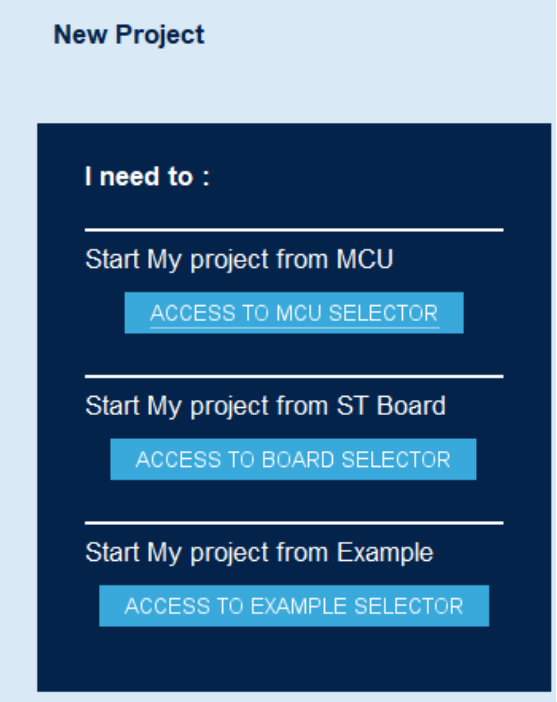
Creating a Project

For this part of the guide I will assume you will be using STM32CubeMX as it is the far easier workflow.

Start with the CubeMX application, it should look something like this when opened (but without the recently opened projects.

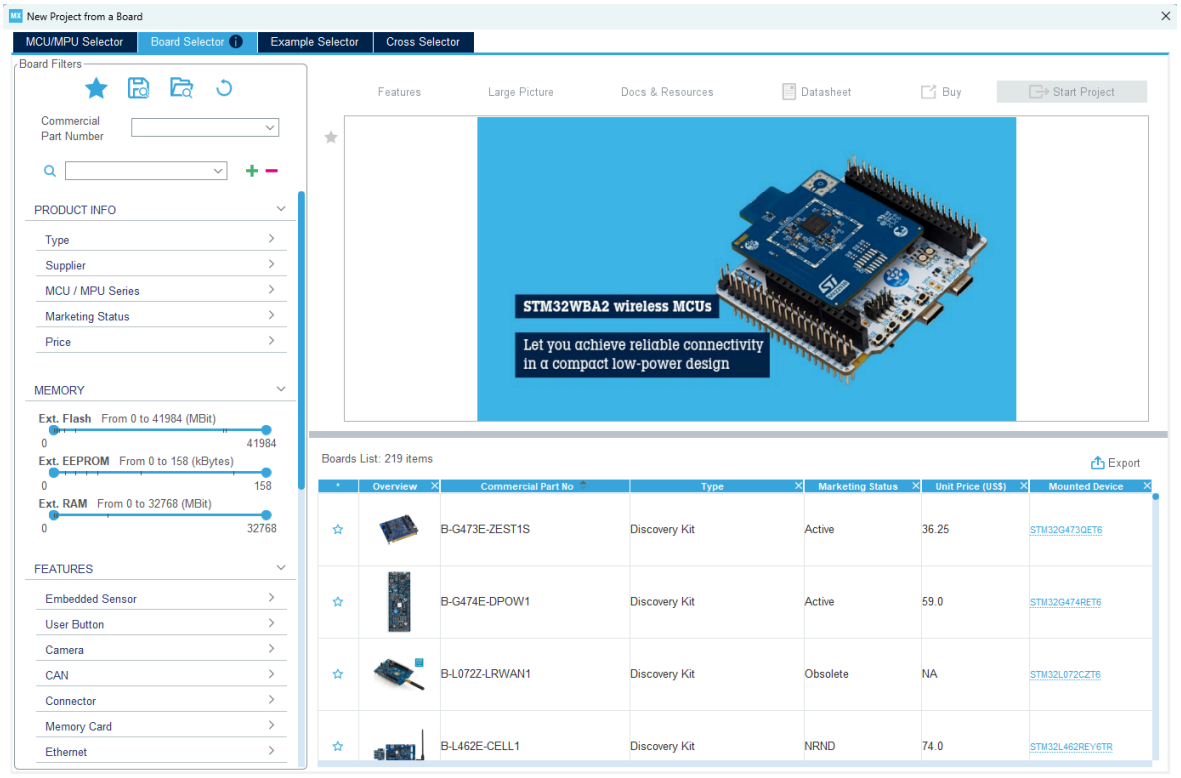


Click on `ACCESS TO BOARD SELECTOR`



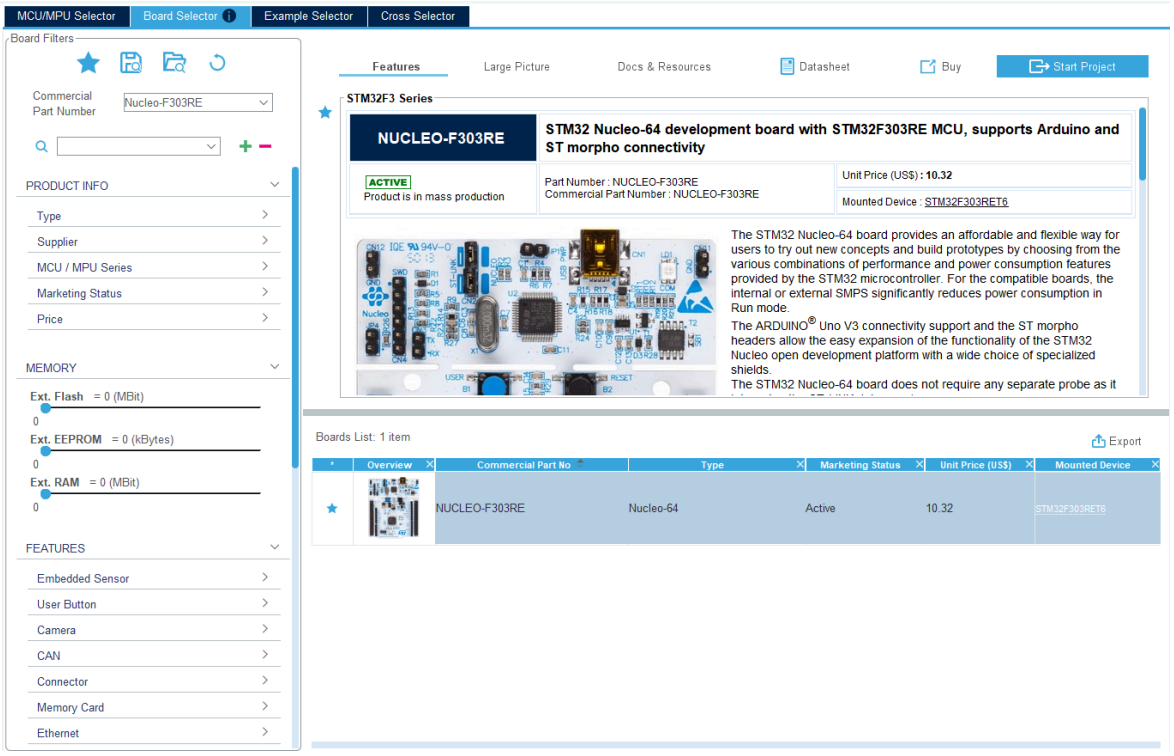
⚠️ If it downloads a few files don't be alarmed - It will download some stuff after you create the new project and initialise the GUI for the board selector it can be a bit laggy, ***so be patient***

Next you should be brought to this screen

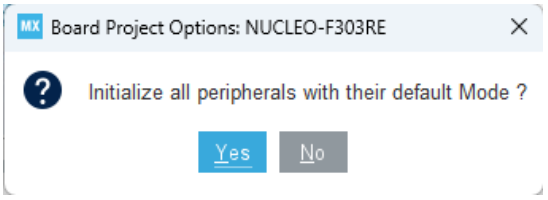


Next to `Commercial Part Number` type **Nucleo-F303RE** (the board we will be using for this course)

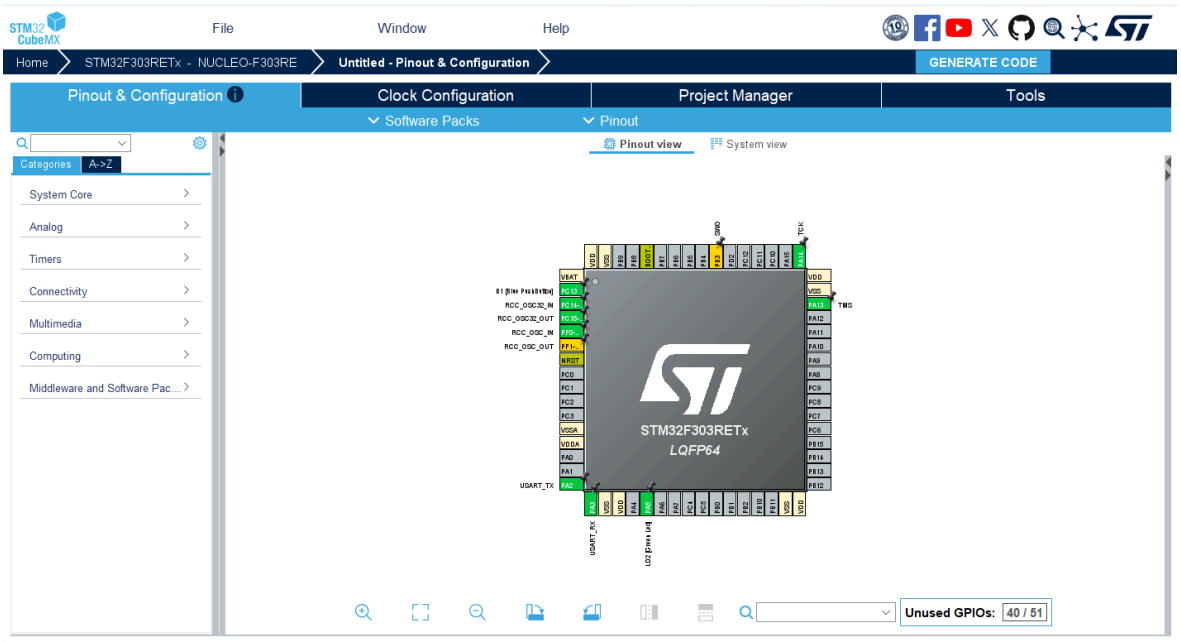
You should now be presented with a singular board option, select it by clicking it and press the `Start Project` button in the top right corner



Click `Yes` to initialising all peripherals to their default Mode.



You should now see a nice GUI of your microcontroller's pins as shown.



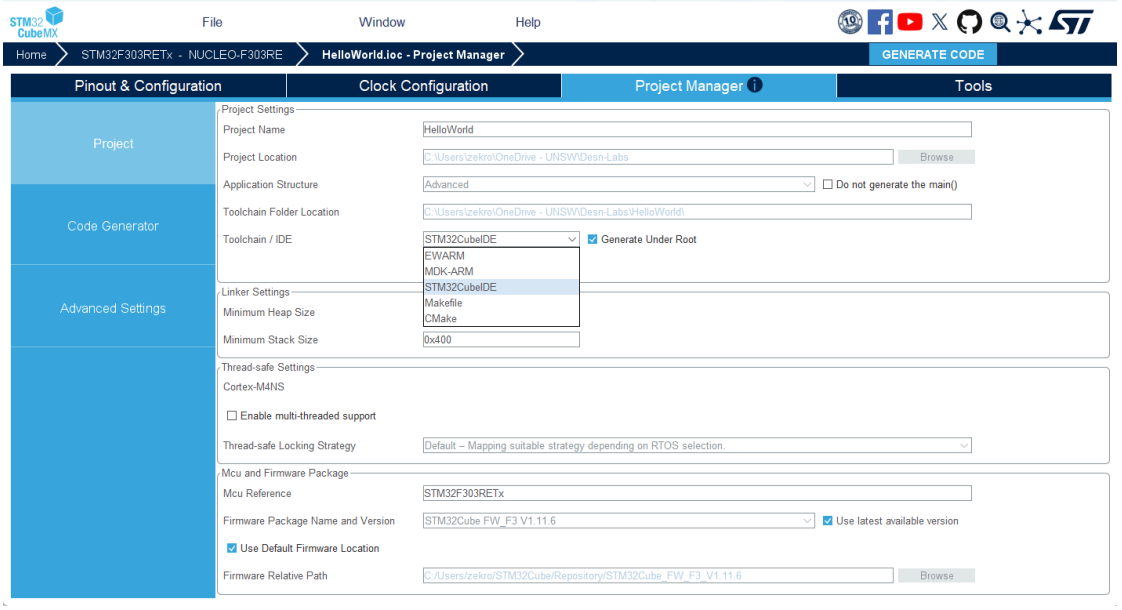
You will eventually become very familiar with this GUI as you will need to use it everytime you change your pin configuration for the each of the labs as well as the project.

Check that

- **PA5** is LD2, the NUCLEO board's Green LED
- **PC13** is B1, the NUCLEO board's Blue Push Button

Keep the defaults for now, we will learn all these shortly, so don't worry.

Next, lets give our project a suitable name by clicking on `Project Manager` and inputting our chosen name (in my case I have chosen "HelloWorld") next to `Project Name`. Also, we must change our targeted `Toolchain / IDE` to `STM32CubeIDE` using the dropdown.

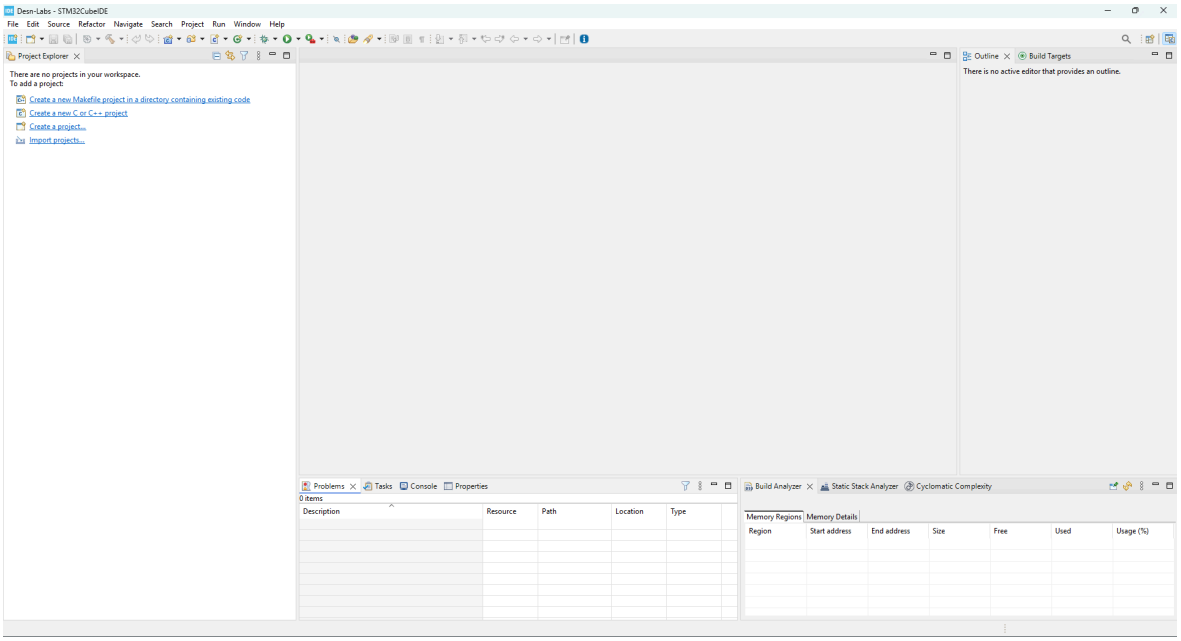


After we confirm that our pin configuration is to our liking and we have given our project a suitable name we can then generate this configuration in the form of code by clicking `Generate Code` in the top right corner

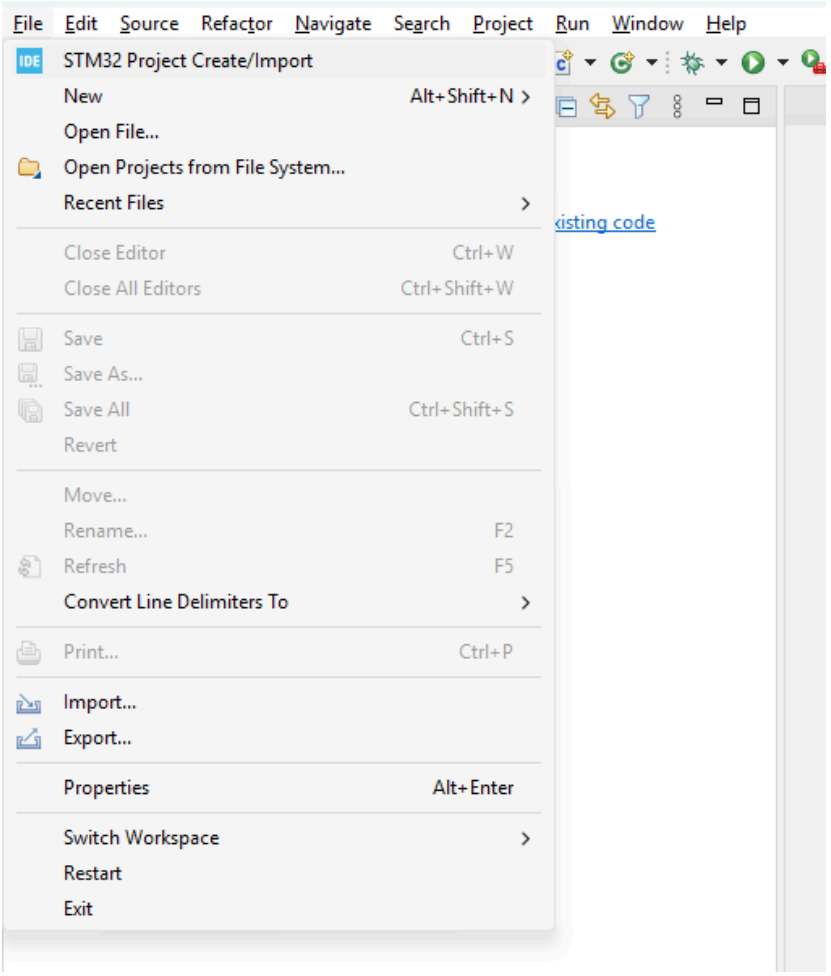
As mentioned before you could write this code corresponding to our pin initialisations manually, however that would waste both time and effort.

After you generate code, click the `Open Folder` option and take note of where your files are stored.

Next start the CubeIDE application, it should look something like this → i.e an empty workspace with no projects. You might have to create a workspace if this is your first time.

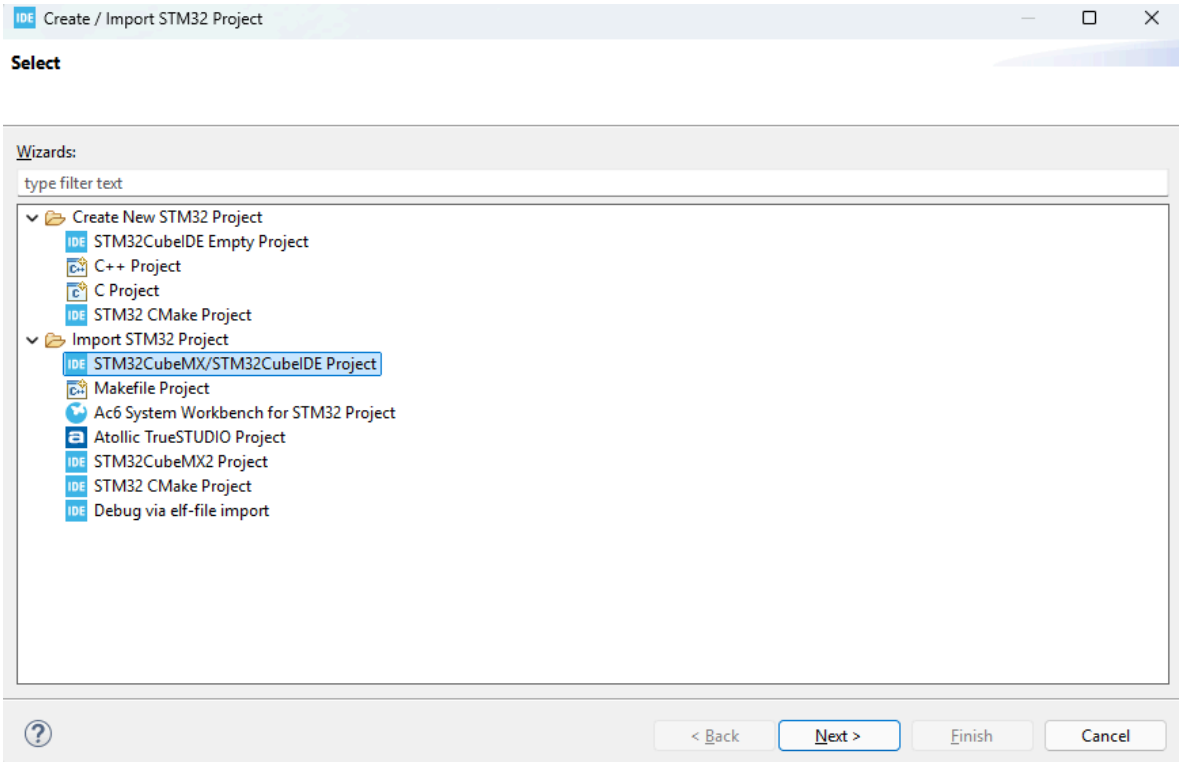


Under the menu bar at the top: `File` click `STM32 Project Create/Import`



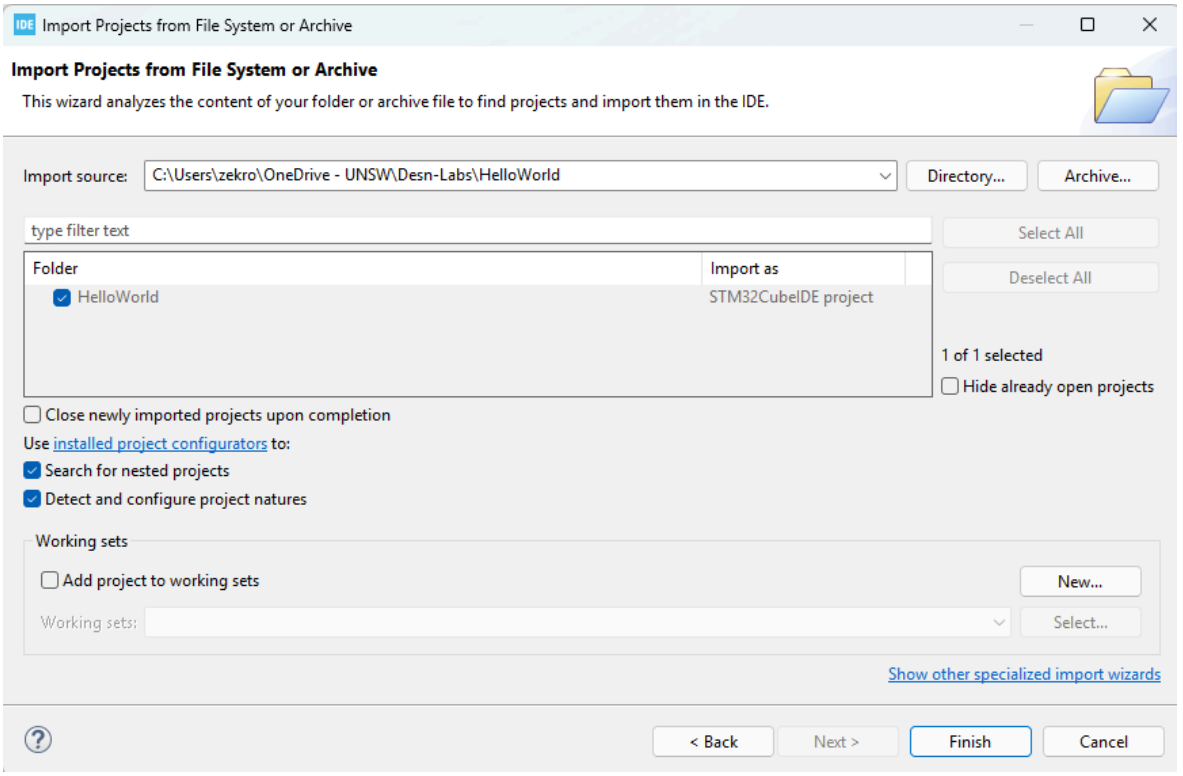
Since we have already generated the code using CubeMX we do not need to create a new project, however if you did not use CubeMX you would have to create a whole new project.

Under the folder `Import STM32 Project` select `STM32CubeMX/STM32CubeIDE Project` and click `Next`

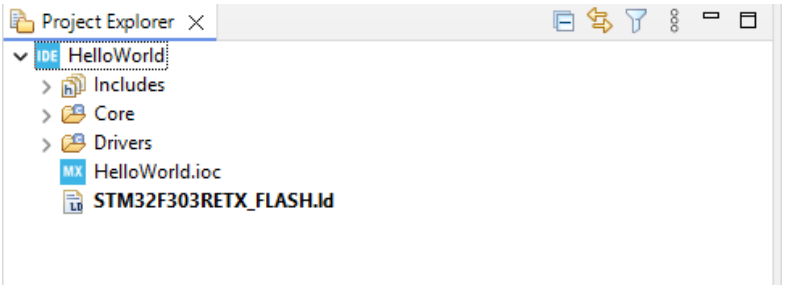


Next, click `Directory` and find the folder where you generated your CubeMX Project

It should automatically detect the project file as indicated by the ☑️ next to your folder name. Click `Finish`

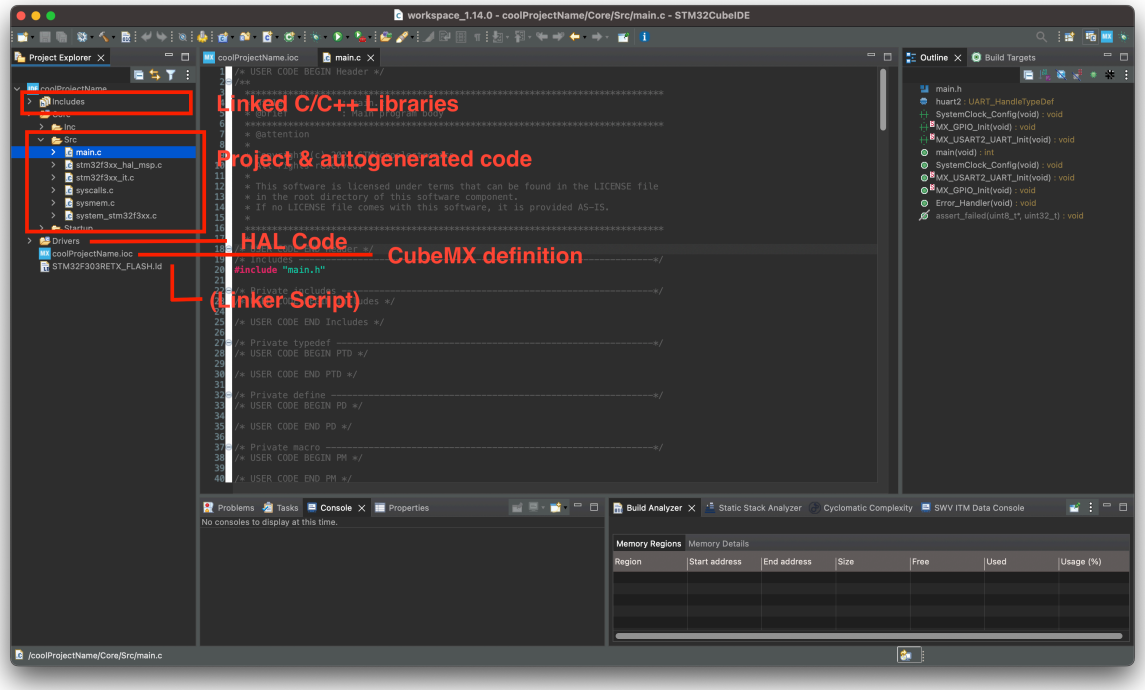


You should now see your created project visible in the `Project Explorer`



💡 If you want to reconfigure your pins then you can do so by double clicking the `.ioc` file and then configuring the pins using the GUI and finally generating the code again.

💾 Writing to the Board



👀 Looking at `main.c`

In this tutorial, we'll focus on the project and autogenerated code section, starting with `main.c`.

You'll see that main.c is already quite large, containing a fair amount of autogenerated code.

⚠️ A key piece of information to remember here is that `main.c` can be edited by the code generator, so it's important to only write code in the ***USER sections***

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------
-----------------------------
----------------*/

  /* Reset of all peripherals,
Initializes the Flash interface
and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock
*/
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured
peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
```

Observe in this block of code that there are several `USER CODE BEGIN` and `USER CODE END` sections marked out by comments

Any code written **inside** these blocks is **safe** and will not be deleted by the code autogenerator.

Any code written **outside** those blocks is **unsafe** and will be *deleted* by the code autogenerator any time you edit the device configuration settings that we looked at earlier.

Files that you yourself add to the project (e.g. `MySuperCFile.c`) are also totally safe. It's just this set of autogenerated `.c` (and `.h`) files that you must be careful with when adding code.

Now to look at the function names used in the autogenerated code.

- Any function call beginning with `HAL_` is from the STM32 HAL, and is provided in the library files. There's HAL functions to do all sorts of things, including using the UART, writing a Pin, etc.
- Any function call beginning with `MX_` is autogenerated by CubeIDE. These functions tend to be used to initialise functions.
- There are exceptions to these rules, including, annoyingly, `SystemClock_Config()`, which is also an autogenerated function.

```
  /* USER CODE END 3 */
}
```

👨‍💻 Writing Our Own Code!

Let's add a smidge of C code of our own now! After the `Infinite Loop` area, we're going to add code to toggle the LED under section 3. To get the autosuggest to show up you press Ctrl+Space:

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    //Here's my new code that
I've added to toggle the Green
LED (LD2)

    HAL_GPIO_TogglePin(LD2_GPIO
_Port, LD2_Pin);

    HAL_Delay(1000);
}
/* USER CODE END 3 */
```

Note the following:

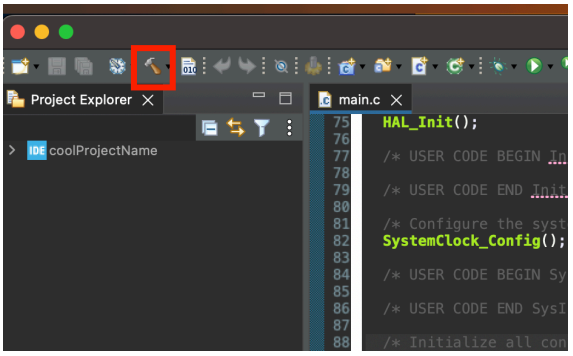
- We have put my code only inside section 3
- Under the device configuration tool, the Led GPIO pin was named LD2 - observe how a name has automatically been generated for both the Pin and the Port
- We have used two HAL functions, one to toggle a GPIO pin, and one to cause a delay of 1000 milliseconds.

This tiny project is all we needed to blink that on-board LED at around once per second.

Compiling and Flashing

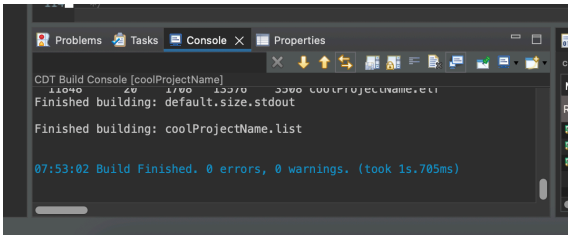
STM32CubeIDE actually makes it pretty easy to compile our work and get it onto the STM32 chip. The first step is to produce the first version of the compiled `.elf` (a binary version of our code). We need this `.elf` so that we can point the download tool to it.

To generate the `.elf`, we need to do a build. Press the *build button on the toolbar*:



Pressing the build button

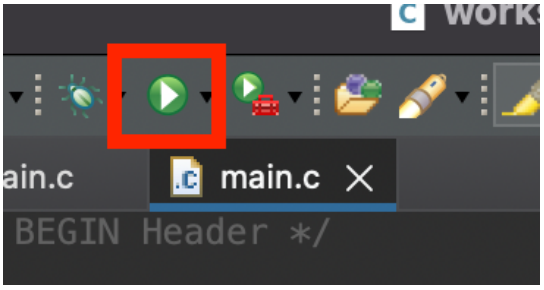
Now, build information is presented in the console at the bottom of the screen:



Result of the build

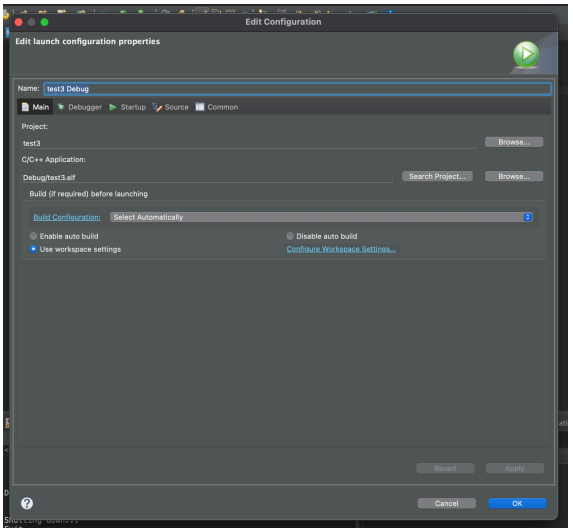
Now we want to send this compiled binary onto the STM32 microcontroller. Let's plug in the development board. The Red power LED (below the blue switch) is lit, as is the larger communication LED (by the USB cable).

Inside STM32CubeIDE, select the **run** button.



This will open the Run dialog (as it's the first time we've run it). The settings we choose now will be saved as a run configuration which we can re-use or edit later. Press **OK** and the download will proceed.

The LED next to the usb connector should flash red and green several times indicating a download is in progress.



Interestingly, we do not get the option to choose a board or USB port or anything during this process, it all just happened automagically. The NUCLEO board's communication LED should light up during this time, and after that, it seems that the board should be running the programme.

It is as easy as that - you've got everything set up now. For any new code from here, you just need to hit the run button - it will compile it for you automatically.

This tutorial was adapted from the blog post by Dr Hammond Pearce. If you prefer a detailed version, you may follow that blog post.

Tutorial: Getting started with an ST development board using STM32CubeIDE

Project initialisation, intro to debugging, and the virtual COM port

https://01001000.xyz/2020-05-11-Tutorial-STM32CubeIDE-Getting-started/