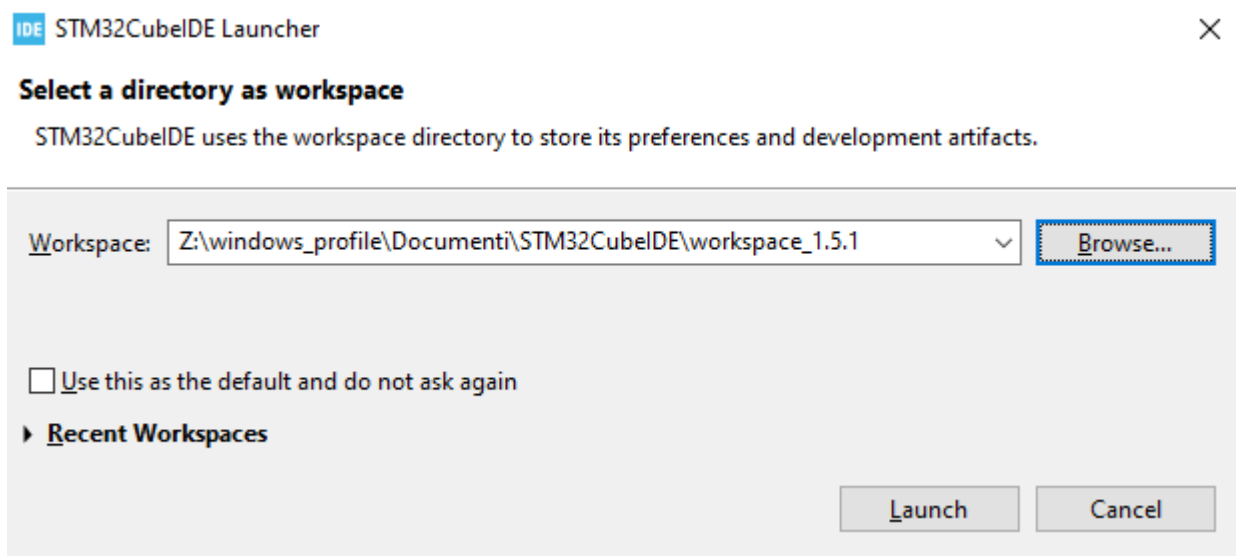


ERTC Laboratory

Preliminary Configurations

Select a location for the workspace

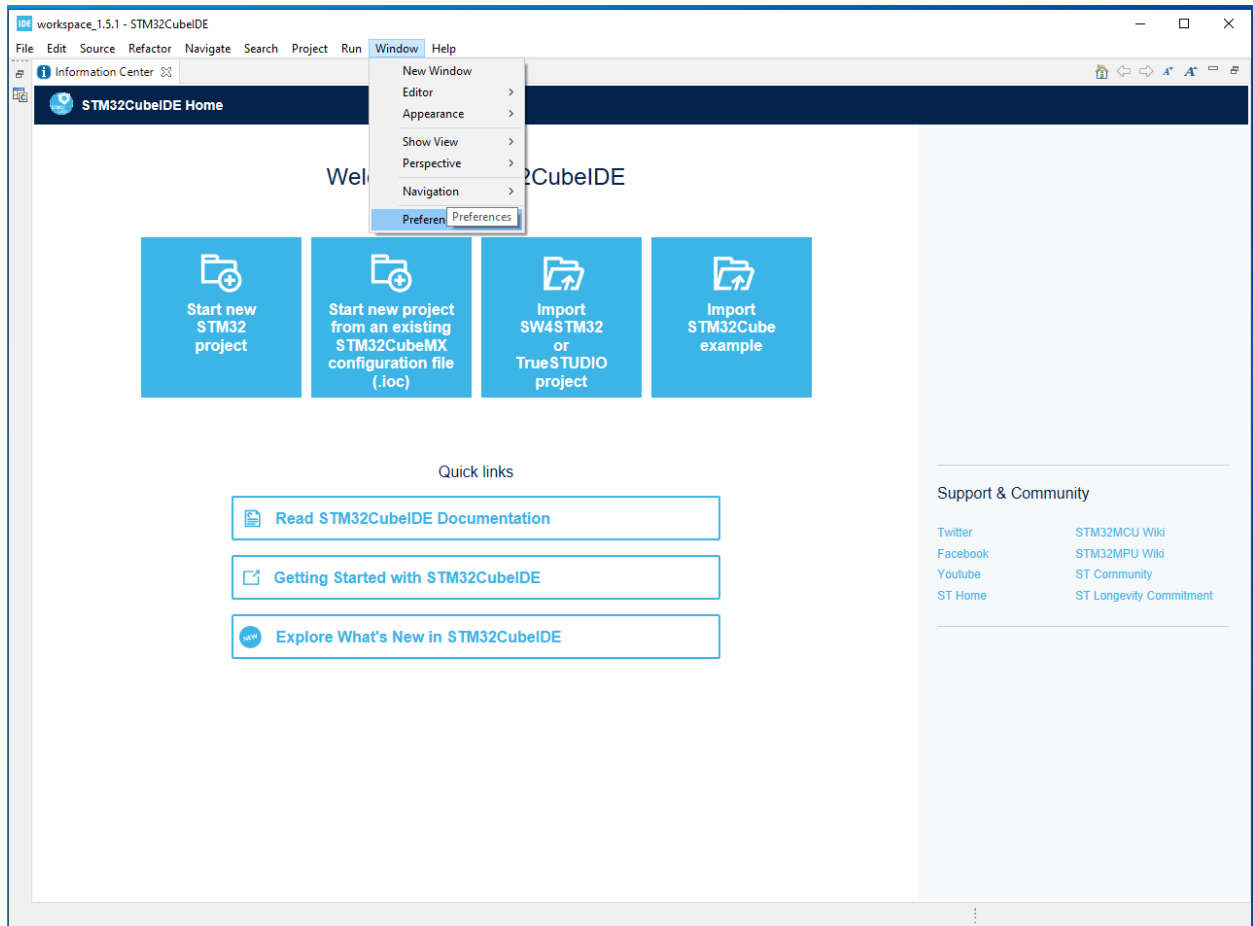
The first time you start STM32CubeIDE, you are going to be asked to select a directory as a workspace. The workspace is the location where your code is placed. Select the location as showed on the image below (should be part of your remote user directory):



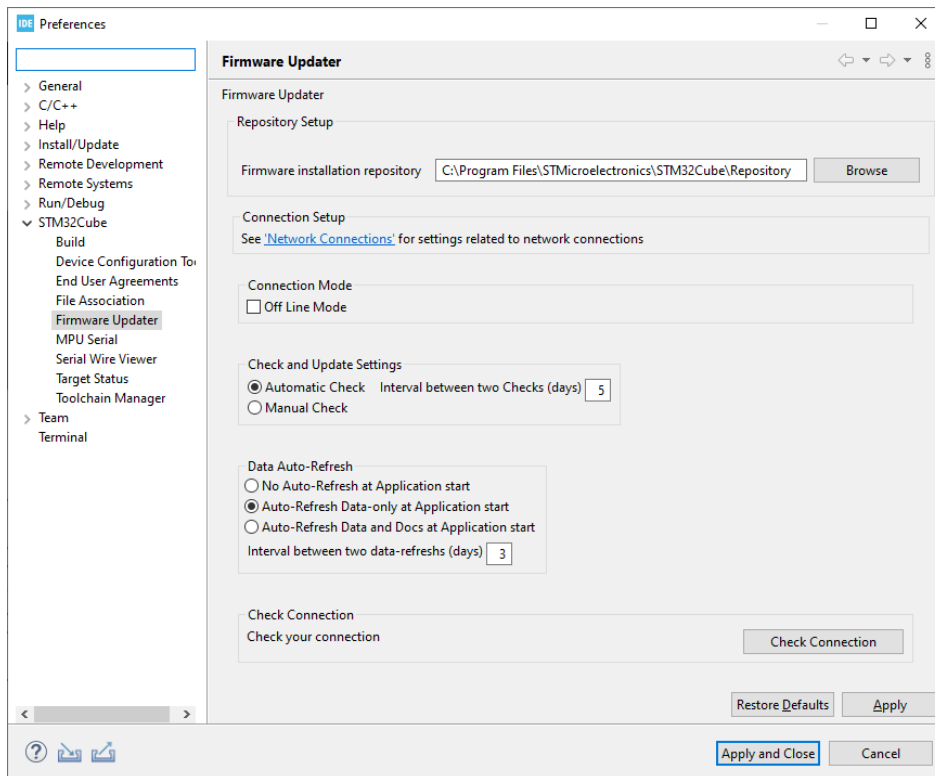
Select the correct repository

Before you create a project, it is appropriate to select the proper repository; in this context, a repository is the location where libraries and other tools are installed. The images below describes the step needed to select the correct repository.

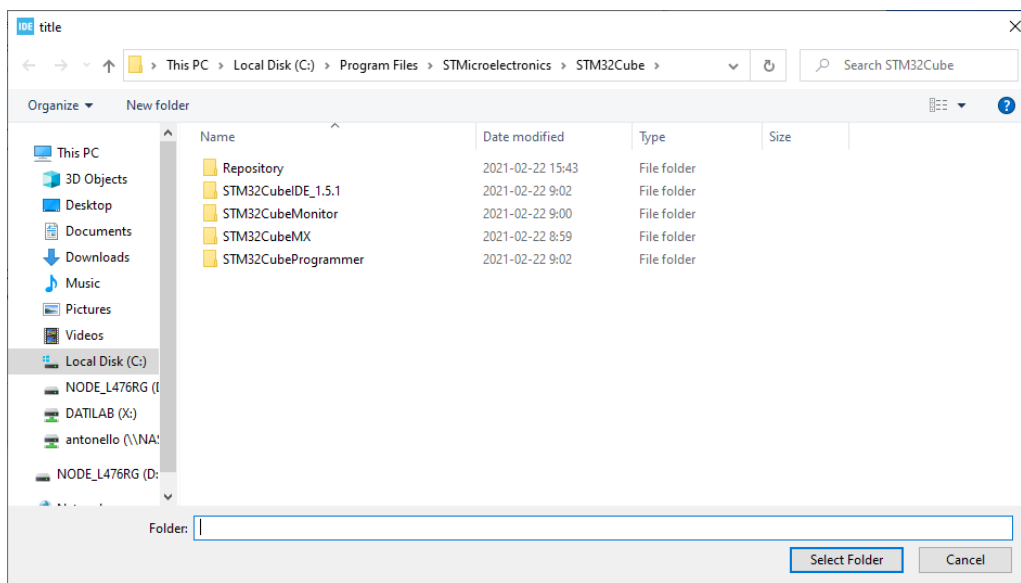
1. Open STM32CubeIDE and select **Window->Preferences** from the Menu bar;



2. Go to the **STM32Cube > Firmware Updater** section and make sure that **Firmware installation repository** is set to the path that you can see from the picture below;

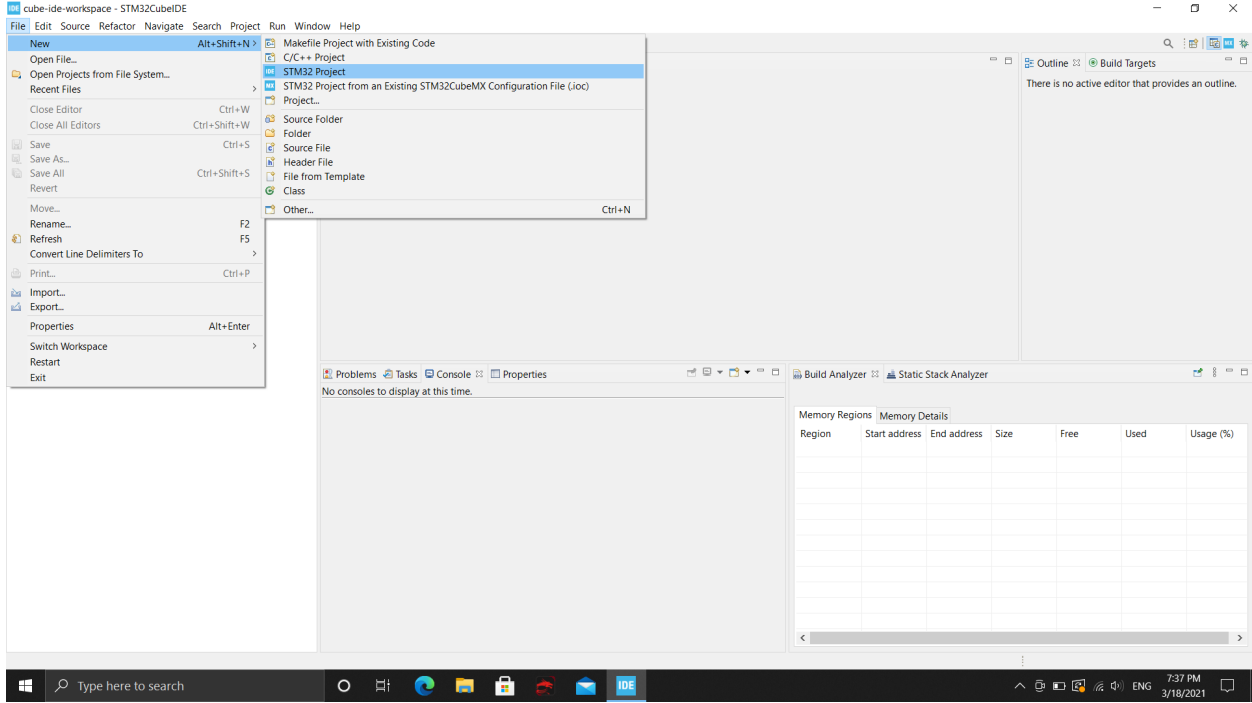


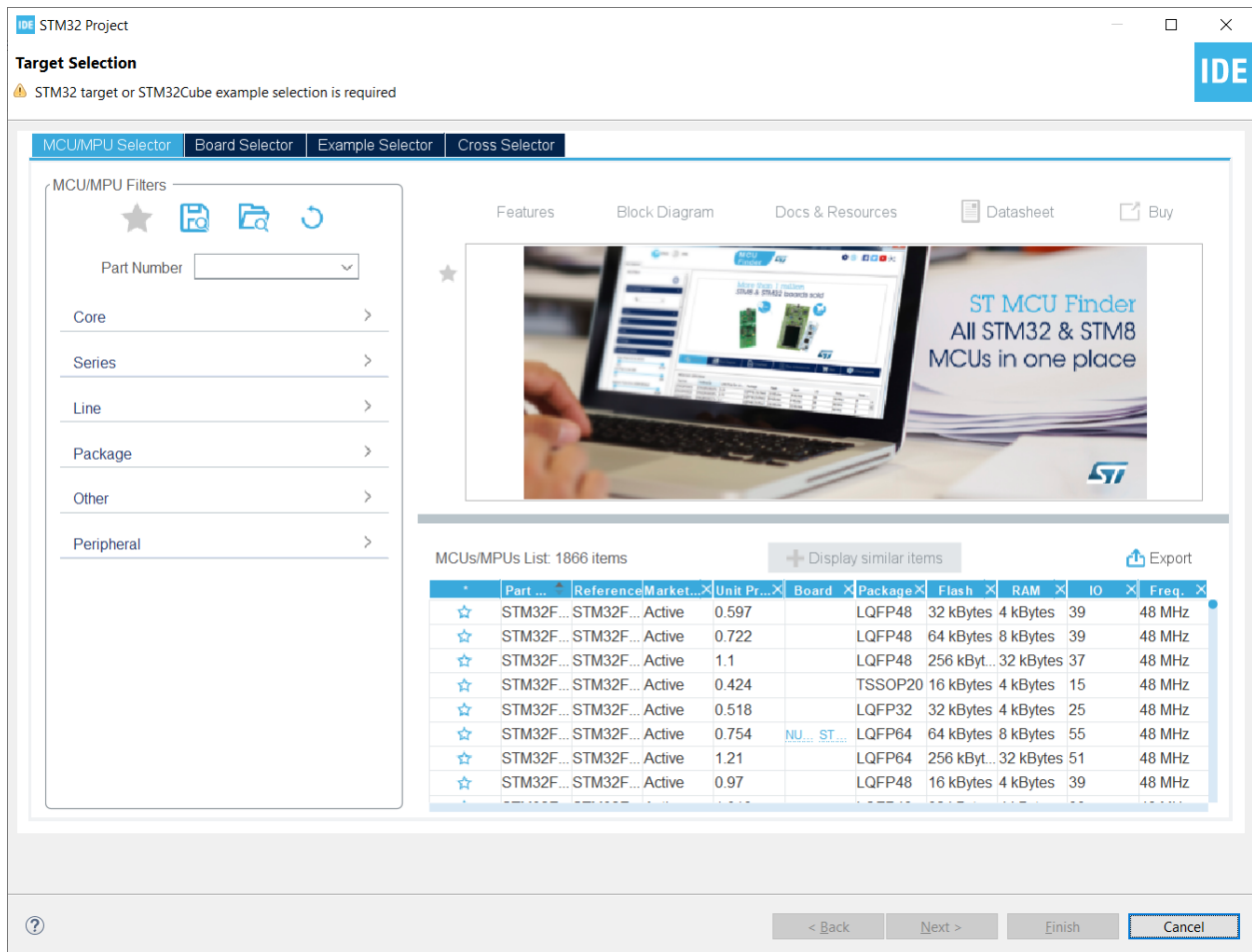
3. If the configuration described in the previous point is not properly setup please fix the configuration, selecting the correct repository location



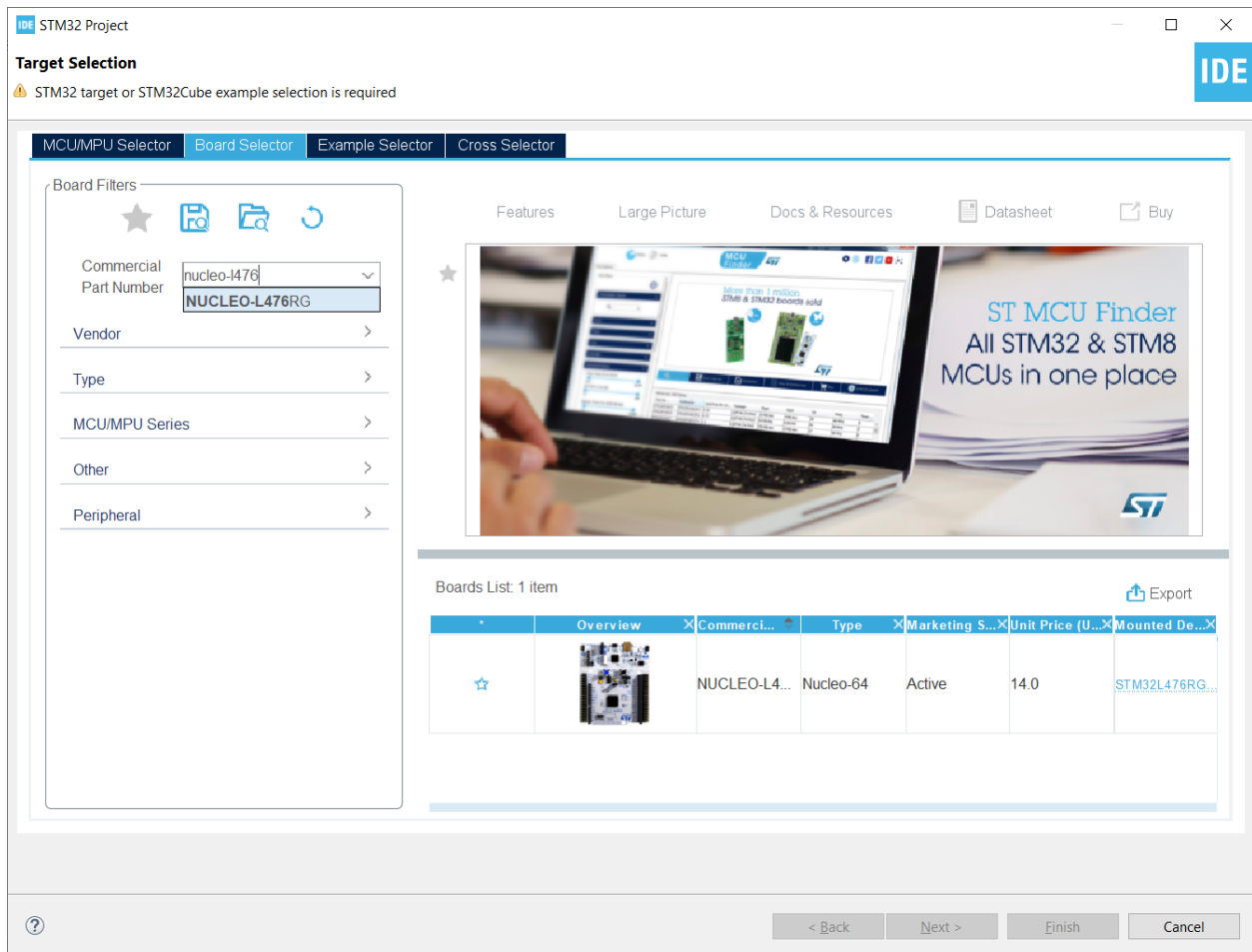
Create a new

1. After having selected **File -> New -> STM32 Project** from the Menu bar, the STM32 Project interface pops.

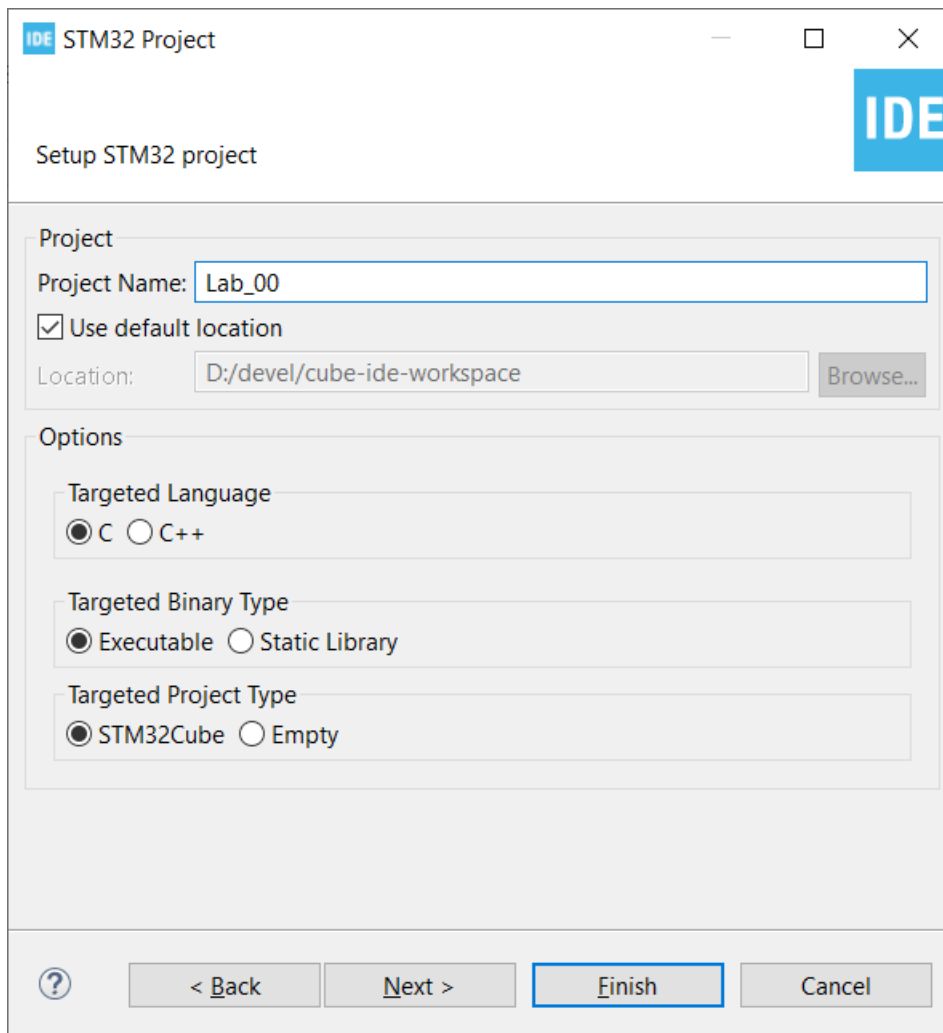




2. Go to the Board Selector tab and select the board, in our case Nucleo-1476rg; after this, click Next.



3. You are prompted to specify a Project Name: you can assign the name that you prefer here.



The image shows the 'STM32 Project' setup window in an IDE. The title bar says 'IDE STM32 Project'. The main title is 'Setup STM32 project'. Under the 'Project' section, 'Project Name' is 'Lab_00'. The 'Use default location' checkbox is checked. The 'Location' is 'D:/devel/cube-ide-workspace' with a 'Browse...' button. Under the 'Options' section, 'Targeted Language' has 'C' selected. 'Targeted Binary Type' has 'Executable' selected. 'Targeted Project Type' has 'STM32Cube' selected. At the bottom are buttons for '?', '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

IDE STM32 Project

Setup STM32 project

Project

Project Name: Lab_00

☒ Use default location

Location: D:/devel/cube-ide-workspace Browse...

Options

Targeted Language

☒ C ☐ C++

Targeted Binary Type

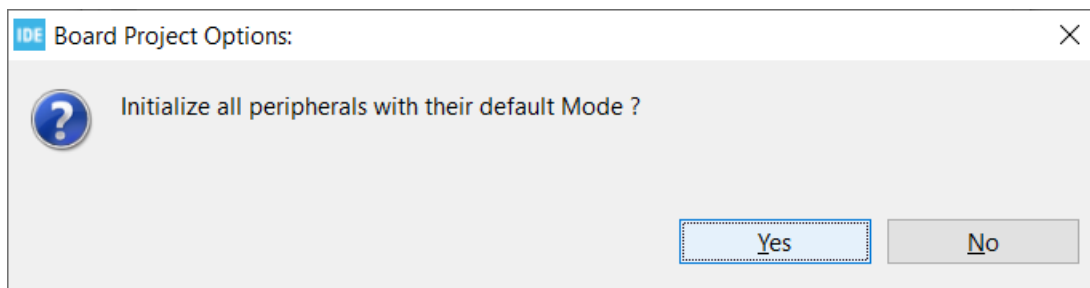
☒ Executable ☐ Static Library

Targeted Project Type

☒ STM32Cube ☐ Empty

? < Back Next > Finish Cancel

4. Initialize all the peripherals with their default mode for the selected board



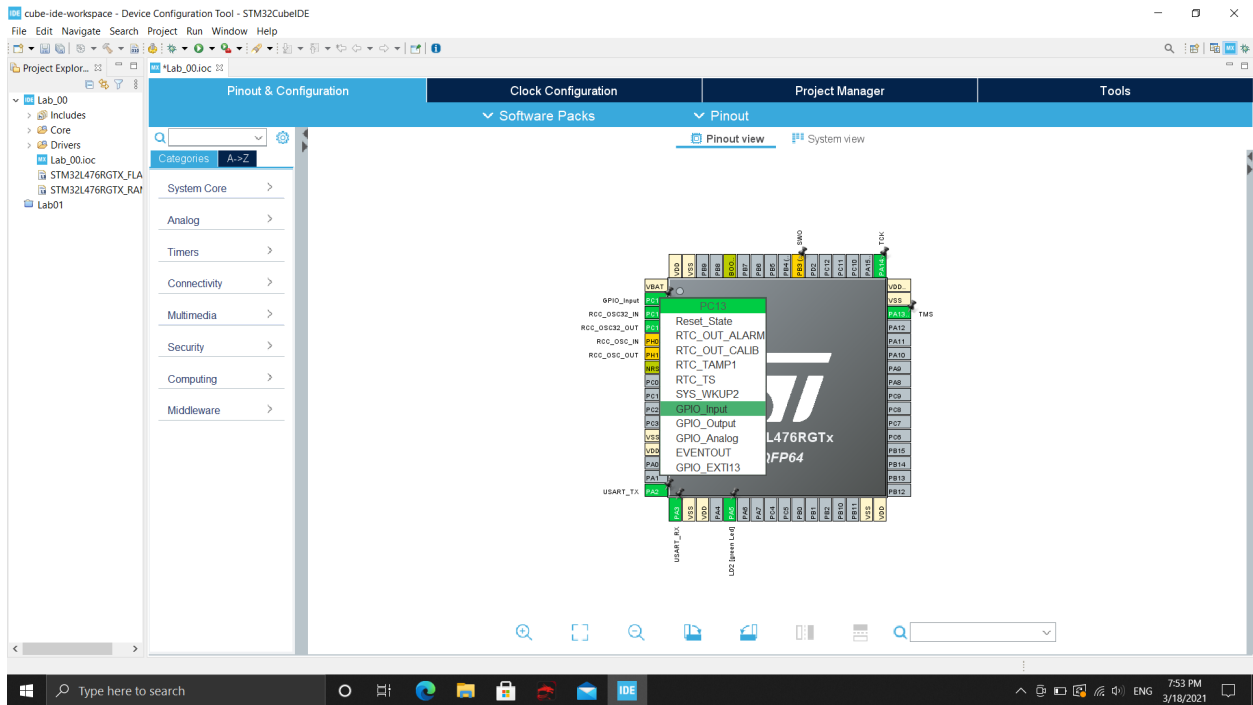
The image shows the 'Board Project Options' dialog box. It has a title bar 'IDE Board Project Options:'. Inside, there is a question mark icon and the text 'Initialize all peripherals with their default Mode ?'. At the bottom are 'Yes' and 'No' buttons, with 'Yes' highlighted by a blue dotted border.

IDE Board Project Options:

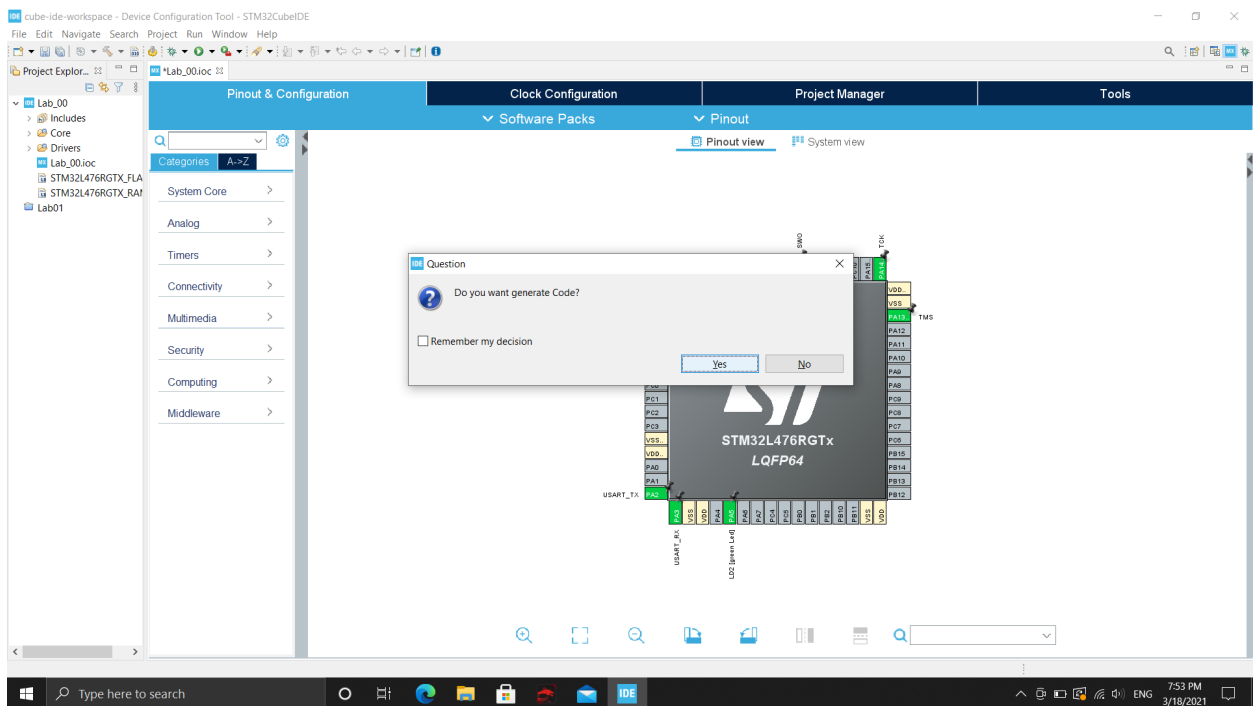
? Initialize all peripherals with their default Mode ?

Yes No

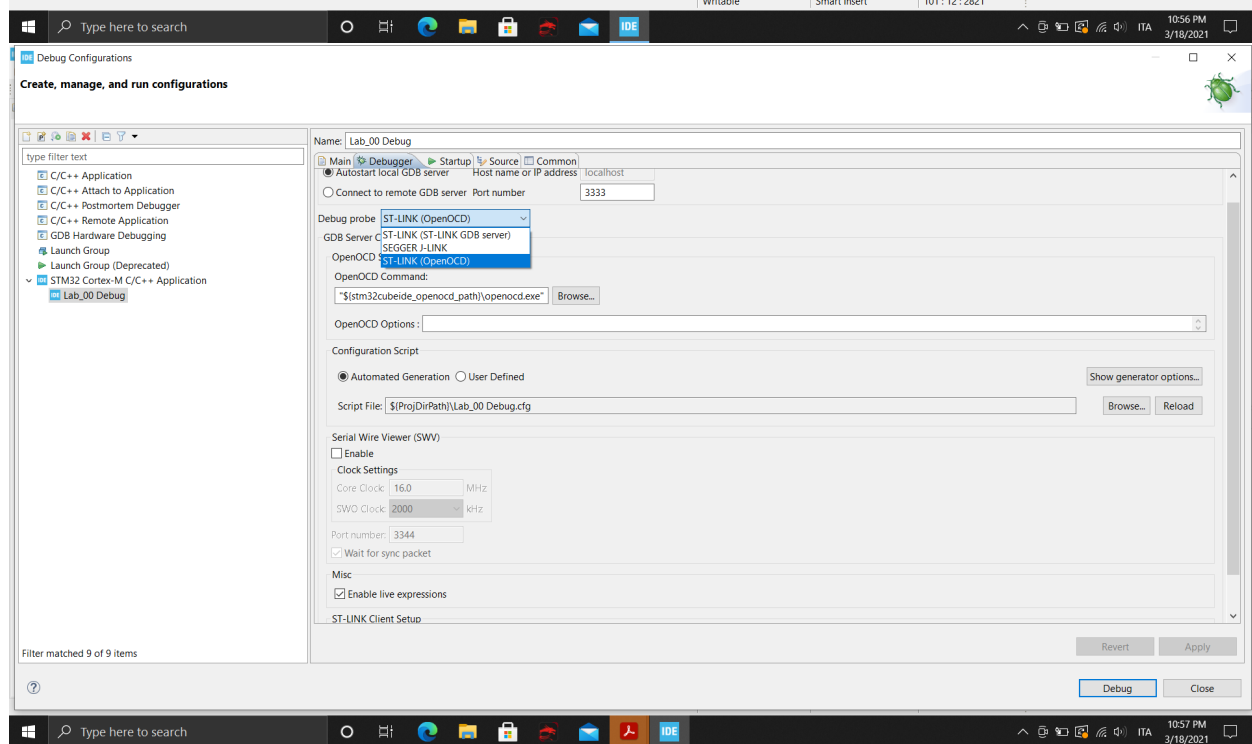
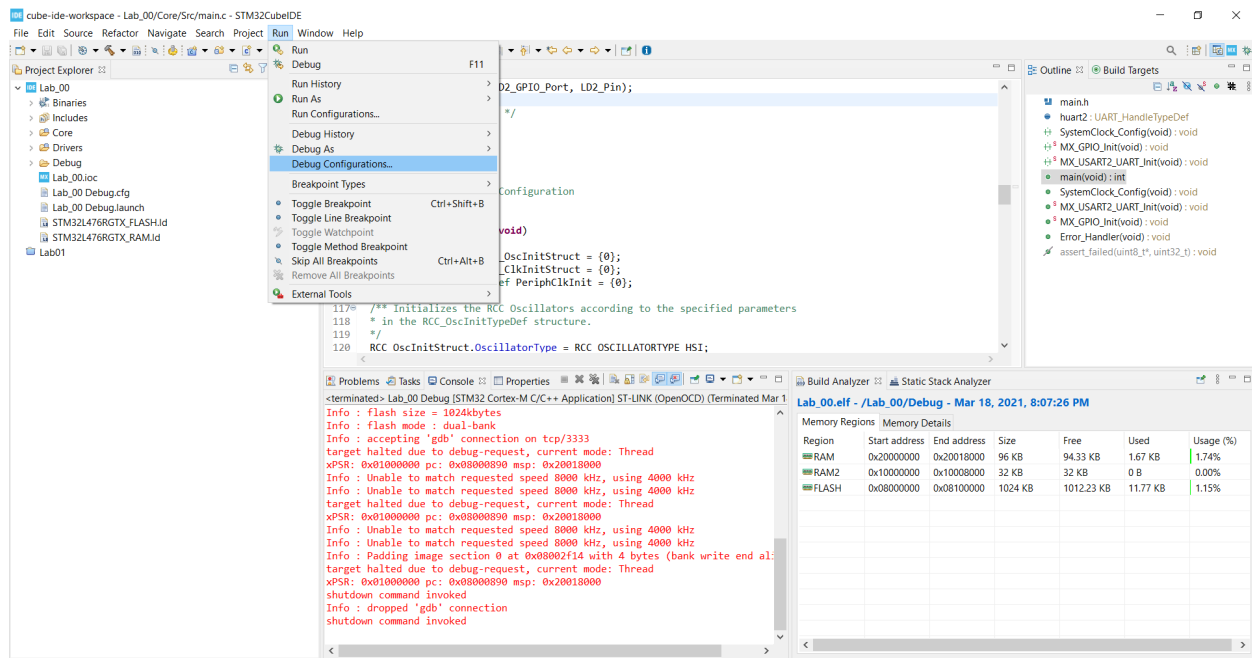
5. Assign the function "GPIO_Input" to the "PC13" pin



6. Saving the project (CTRL + S) trigger the code generation



Configure the debugger



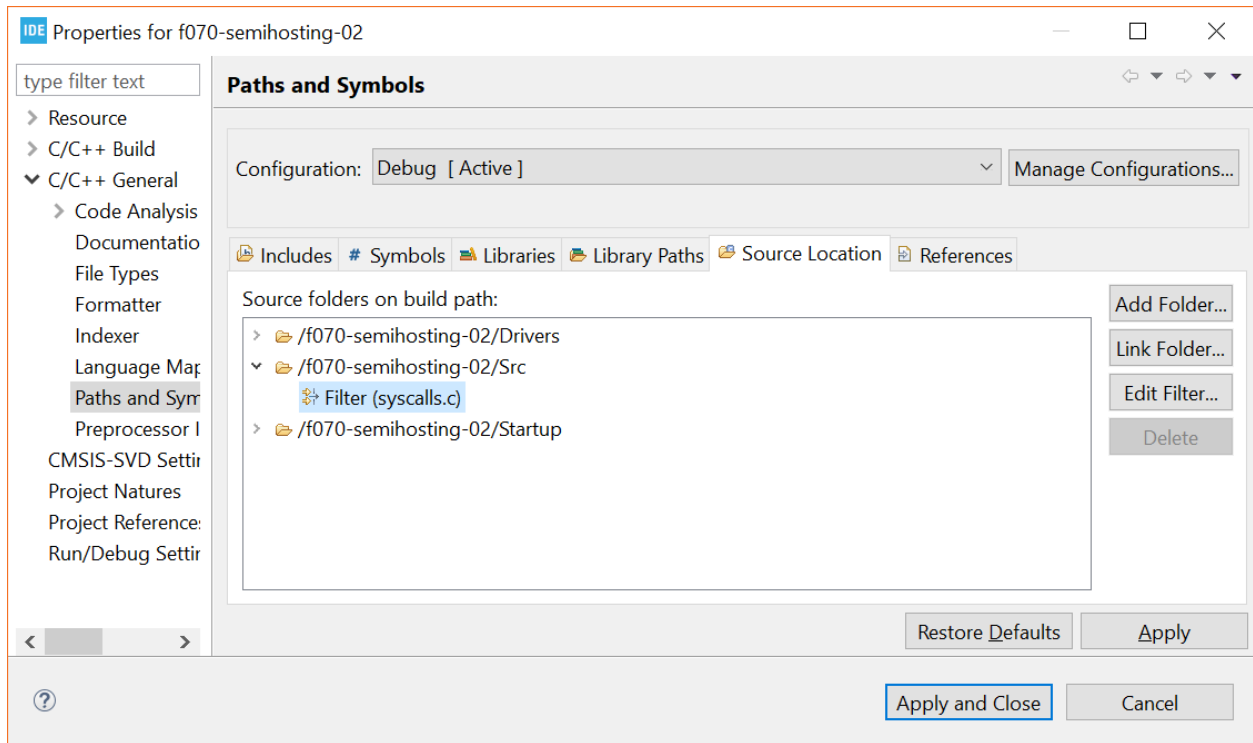
Use debug print

Debugger and libraries configuration

This guide is taken from <https://shawnhymel.com/1840/how-to-use-semihosting-with-stm32/>

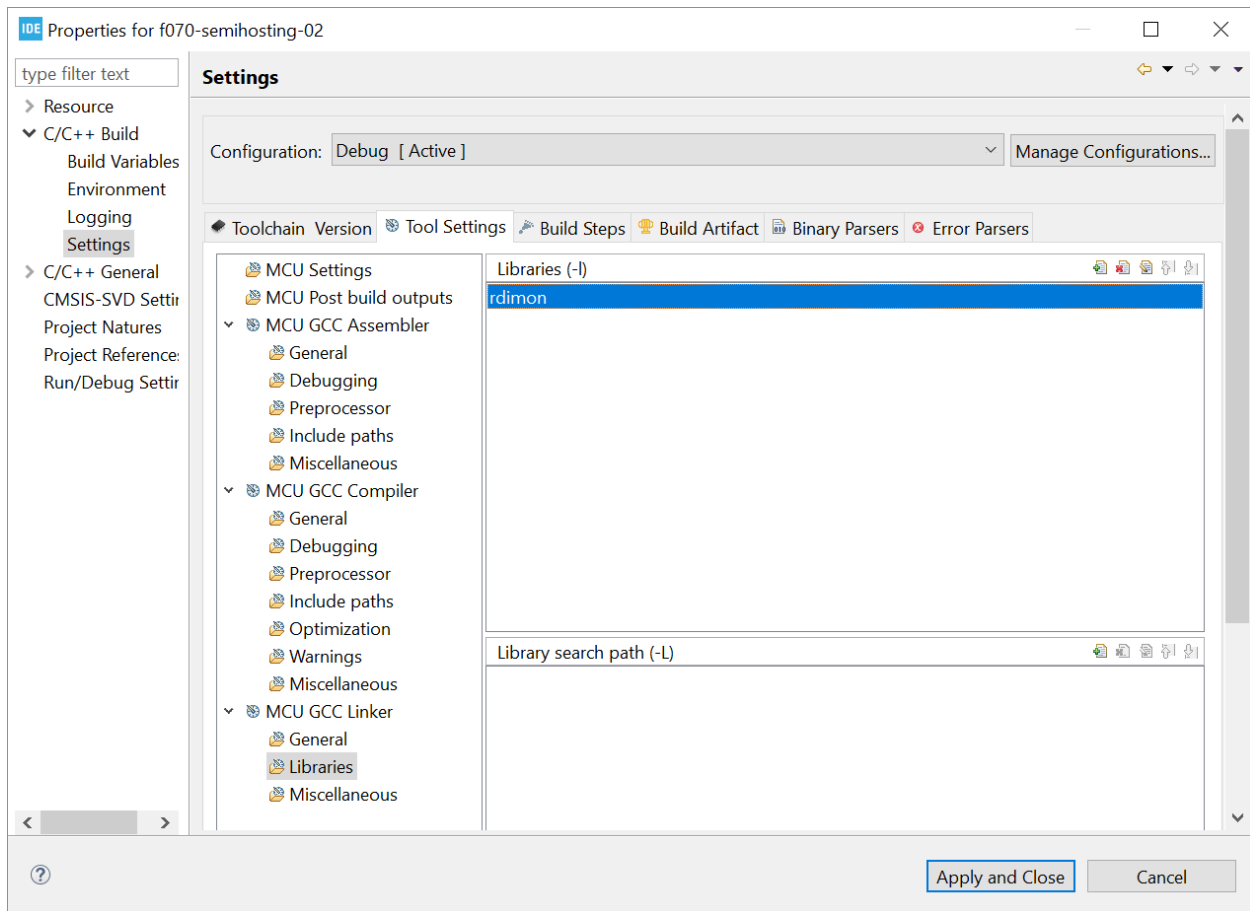
To start, go to Project > Properties.

Go in to C/C++ General > Paths and Symbols. Click on the Source Location tab. Click on the arrow next to //Src to view the filter. Select Filter (empty) and click the Edit Filter... button. Add syscalls.c to the Exclusion patterns list and click OK.



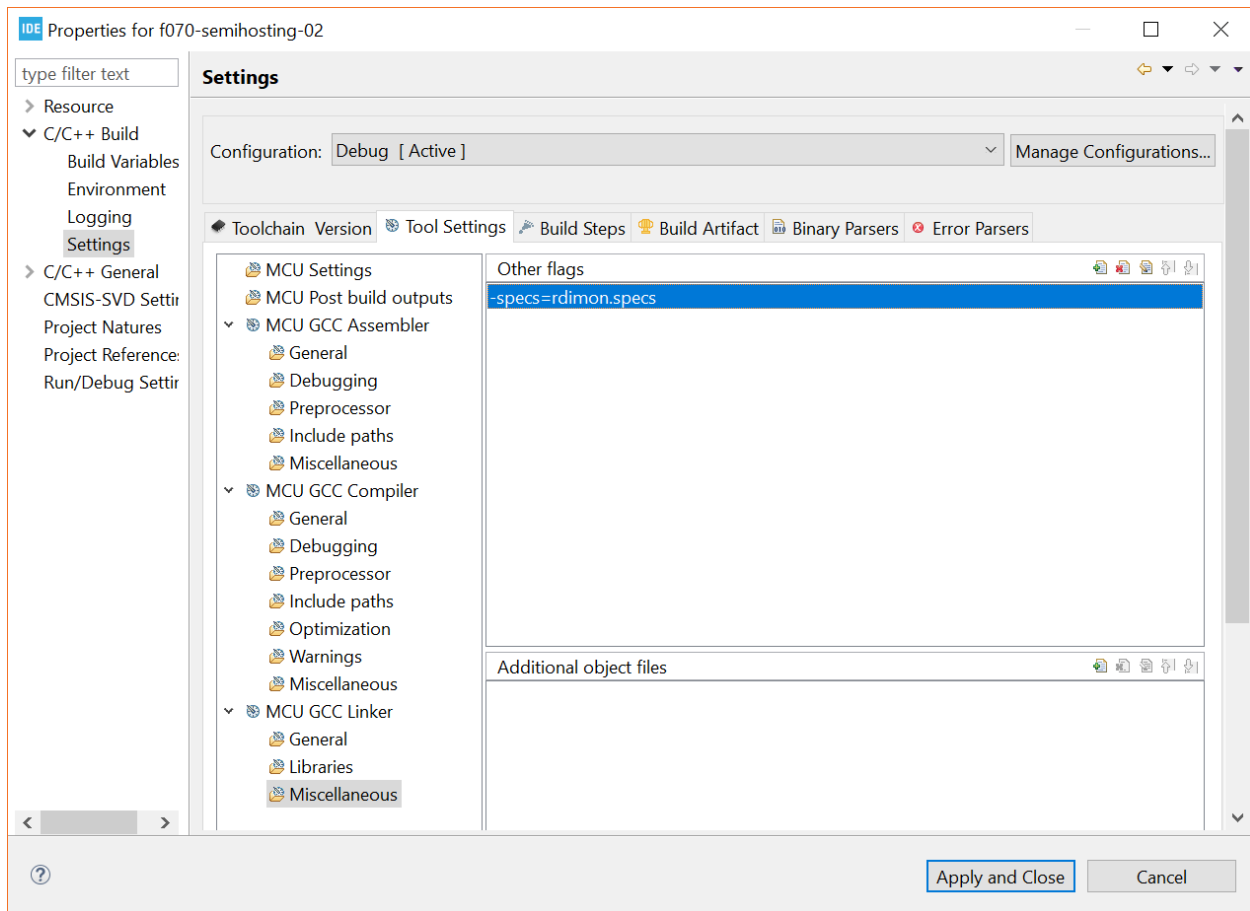
Click Apply.

On the left-side pane, go into C/C++ Build and select the Tool Settings tab. Then, select MCU GCC Linker > Libraries. In the libraries pane, click the Add... button and enter rdimon. This enables librdimon for us to make system calls with semihosting.



Next, click on MCU GCC Linker > Miscellaneous while still in the Tool Settings tab. Click the Add... button and enter `-specs=rdimon.specs` into the dialog box.

Open `Src/main.c`. Above `int main(void)`, add the following line (I put mine in the USER CODE 0 section):



Printf Code

```
extern void initialise_monitor_handles(void);
```

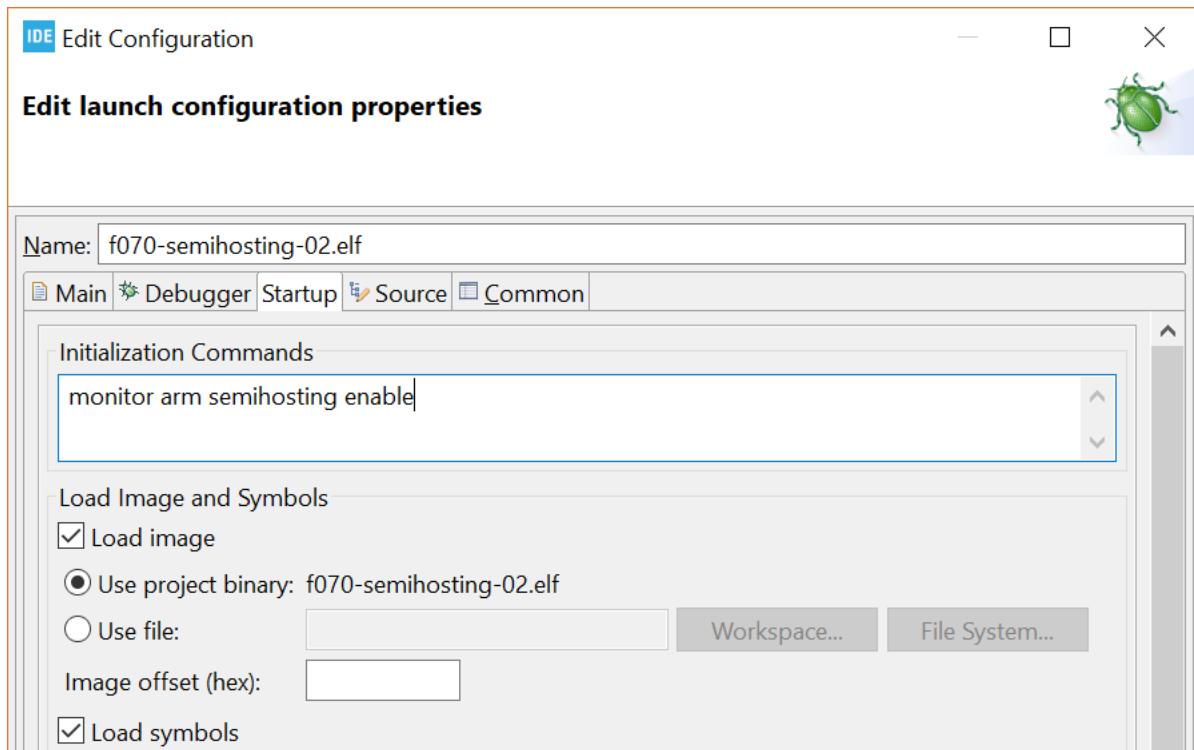
In `int main(void)` (before the `while(1)` loop), add the following line (I put mine in the USER CODE 1 section):

```
initialise_monitor_handles();
```

Finally, inside the `while(1)` loop, add the following:

```
printf("Hello, World!\n");
HAL_Delay(1000);
```

Set Debug Configuration



Installing and running your code

Flashing is the act of installing your code into the flash of the microcontroller;

- Flash the board and start a debug session by pressing the button on the IDE;

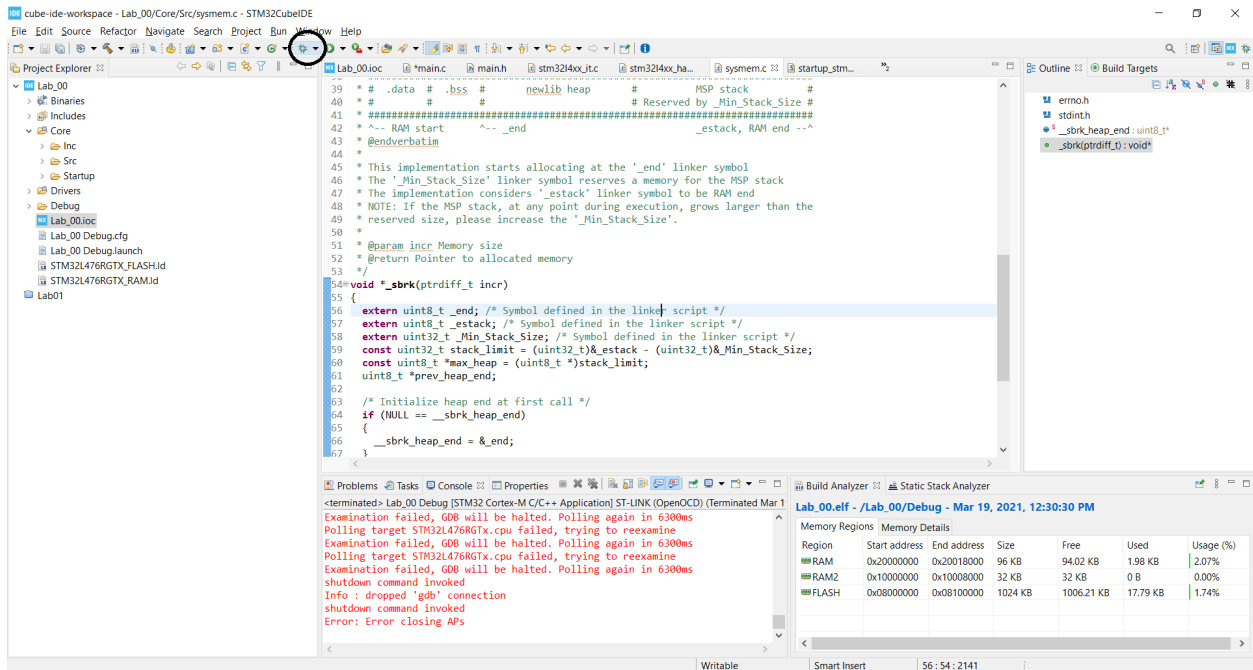
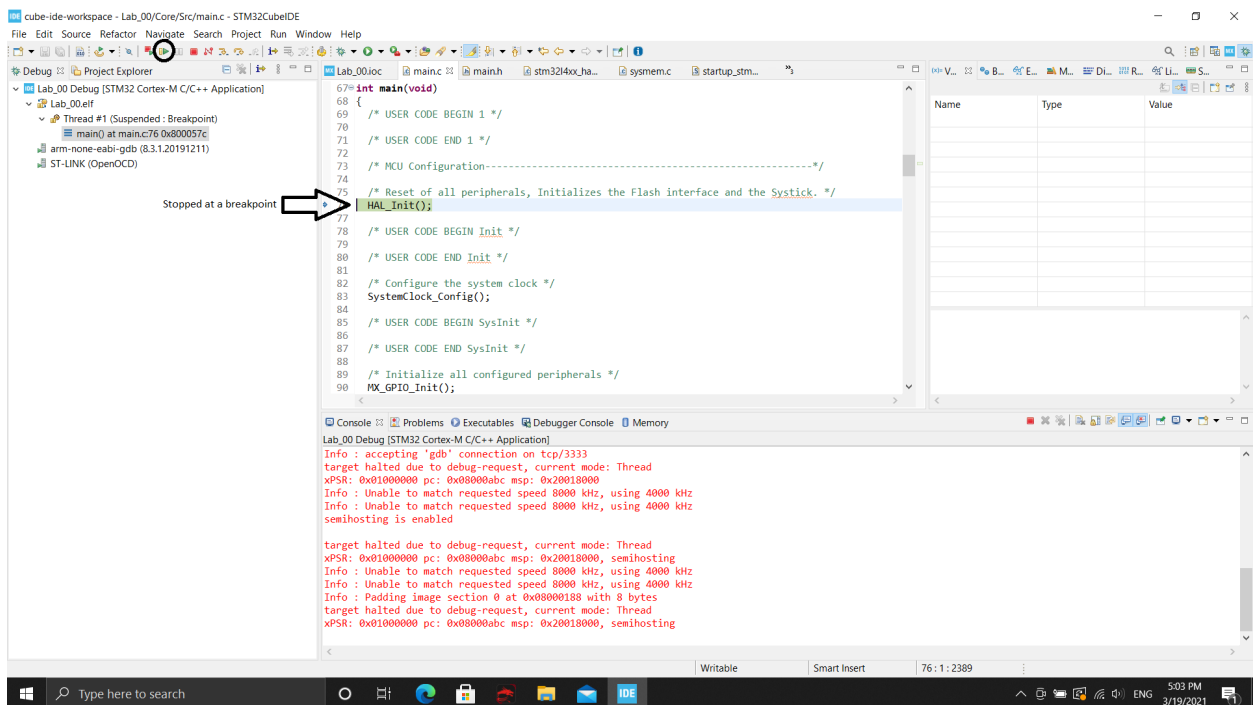


Figure 1: Debug button

- The code blocks on a breakpoint at startup, press the button to resume the execution;



* In debug mode you can pause the execution or put breakpoints to stop the program if the execution meet the breakpoint: this can be very useful if you need to investigate a problem in your code;

Access to your code

- If you have a DEI account, you should be able to login into your home directory by connecting, via ssh, to `login.dei.unipd`;
- Alternatively, you can copy the code into a pendrive;
- If you know how to do that, you are free to use versioning tools like, for example, **git** or **SVN**;

Suggestions:

- Auto completion: you can auto complete the code by using the “CTL + Spacebar” key combination;
- Keep the API reference open;
- You can use multiple files;
- Use the schematics to understand how the button and the led should work;
- Write code inside the blocks like:

```
/* USER CODE BEGIN ... */  
  
/* USER CODE END ... */
```

this way the code will be preserved in case you change something in the project