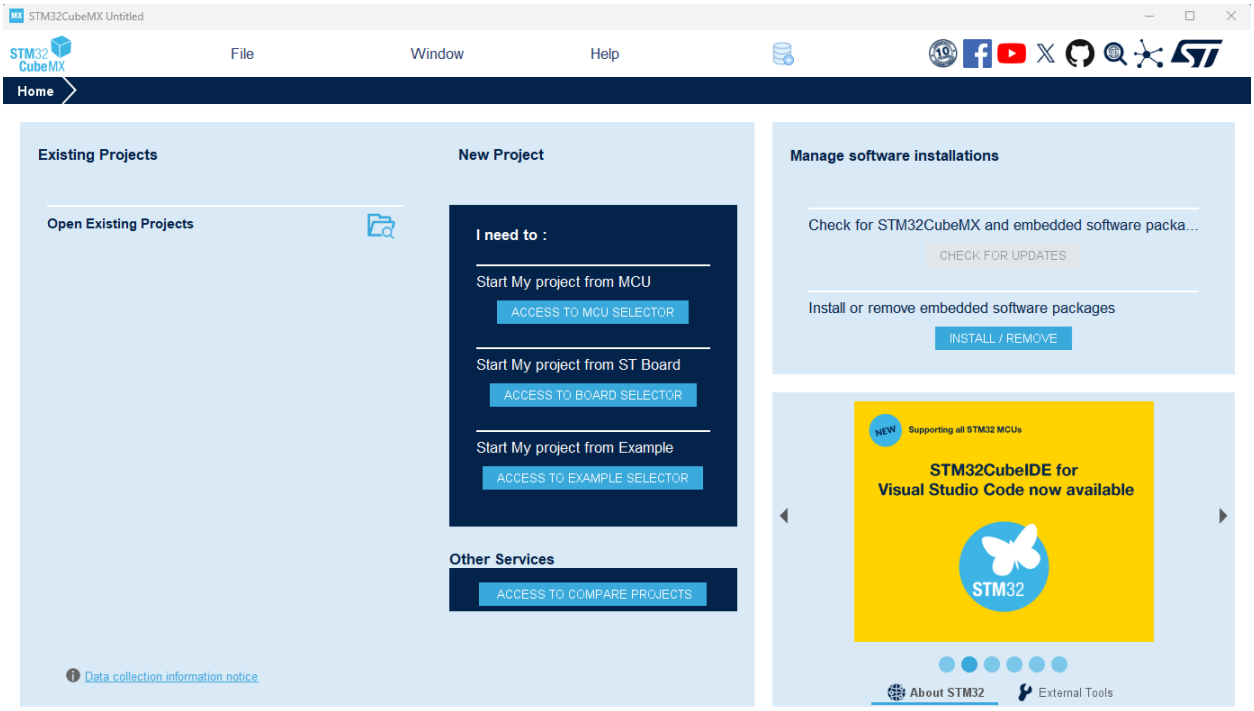
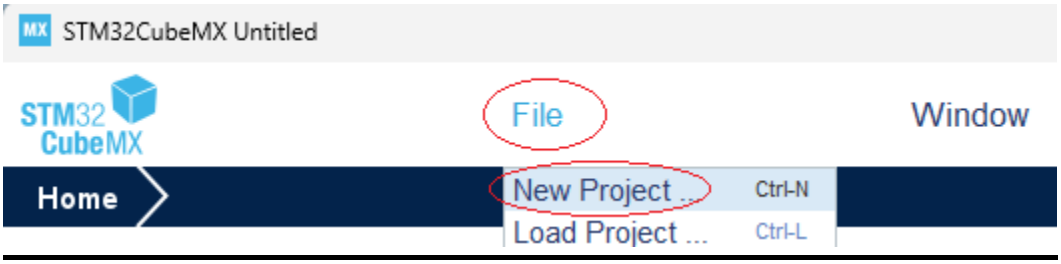
ECE 212 Lab 0 – Tutorial

Getting Started with STM32CubeMX

Double click the STM32CubeMX shortcut icon on the desktop or use the Start menu or the search feature to find the STM32CubeMX program and open it.



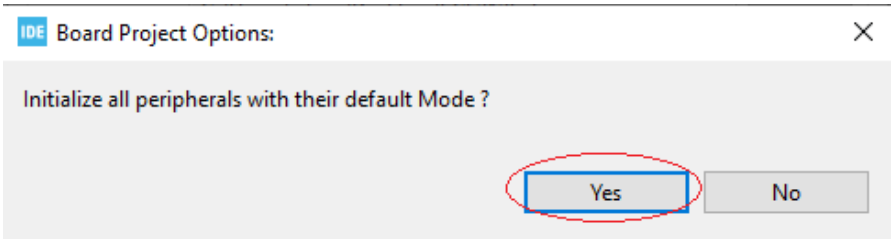
Launch a new project by selecting **'File' -> 'New Project'**



Select "Board Selector" and type in the name of your board (NUCLEO-L432KC) in the Commercial Part Number search window. Once the

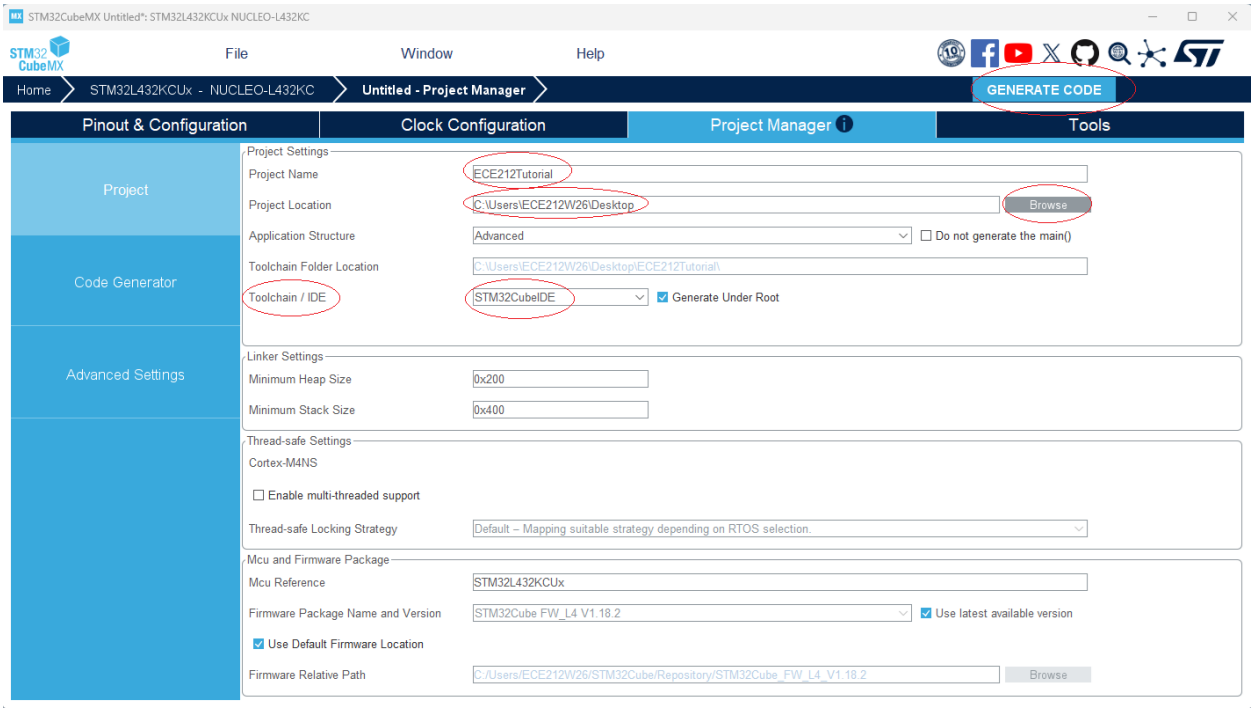
board is populated in the Boards List, go ahead and select it before clicking on ‘Start Project’

Select “Yes” to initialize all peripherals and

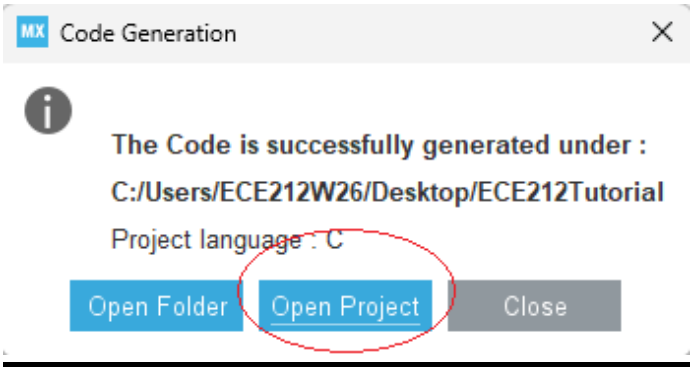


A configuration tool editor will be presented. This is where you will customize certain options for your board such as pinouts, clock, etc. For our tutorial, we only need to change the Toolchain/IDE. In lab4, we need to configure some GPIOs for the hardware interface.

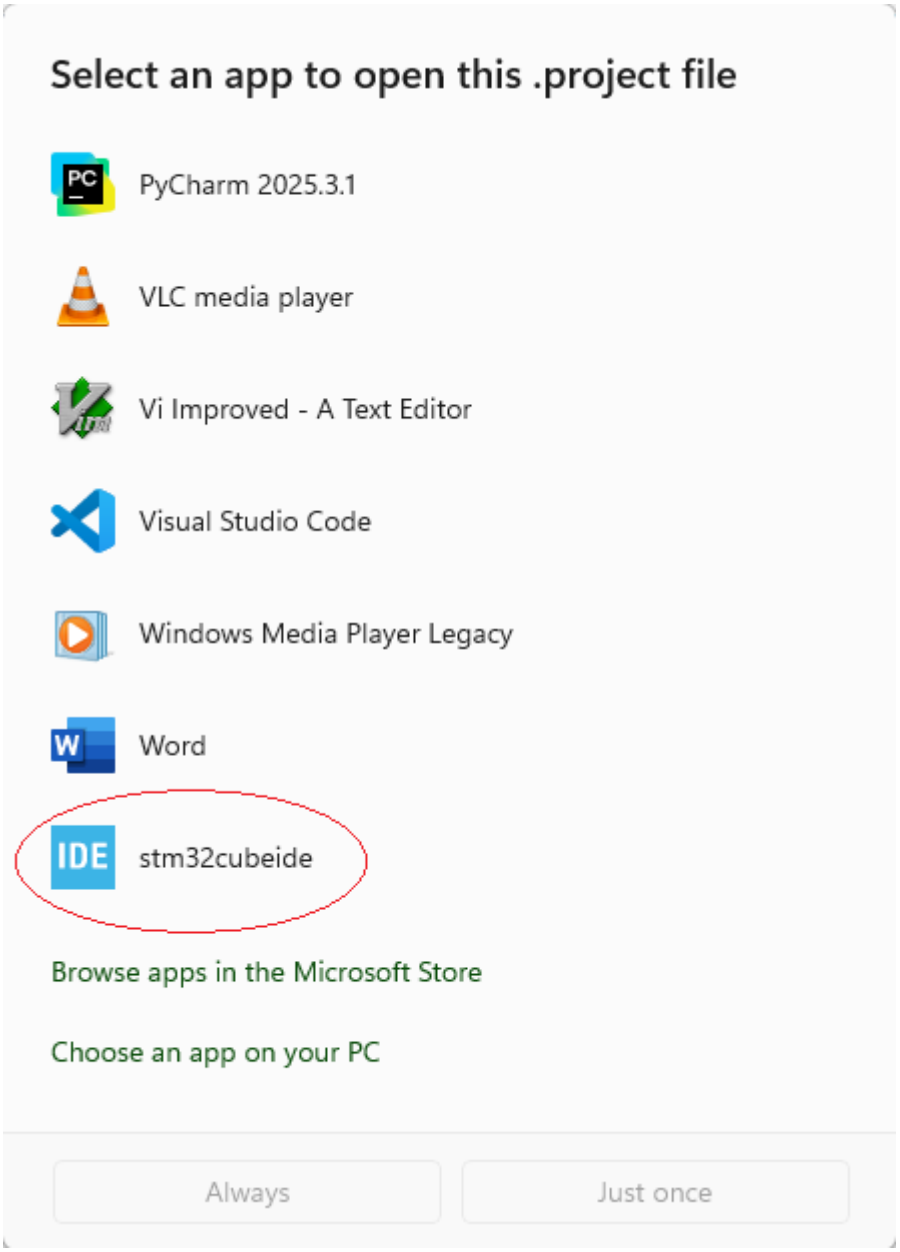
Click on the ‘Project Manager’ tab and do the following. In the Project name, call it ‘ECE212Tutorial’. In the Project location, click on Browse and select the Desktop. Change the Toolchain/IDE dropdown option to ‘STM32CubeIDE’. Once this change is made, click on “GENERATE CODE”.



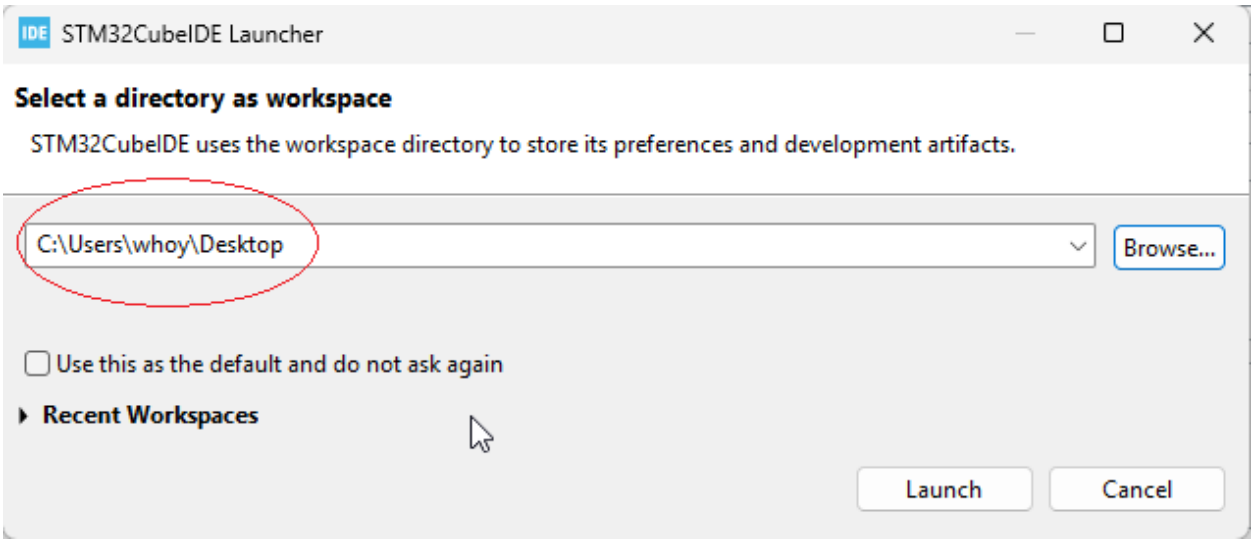
Click on ‘Open Project’ to launch the application in STM32CubeIDE.

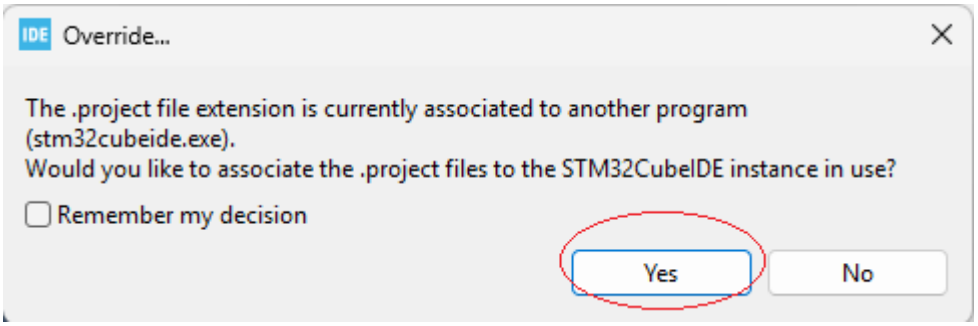


During the first time, you will need to select which application to open the project, please select STM32CubeIDE program from the list and click ‘Always’

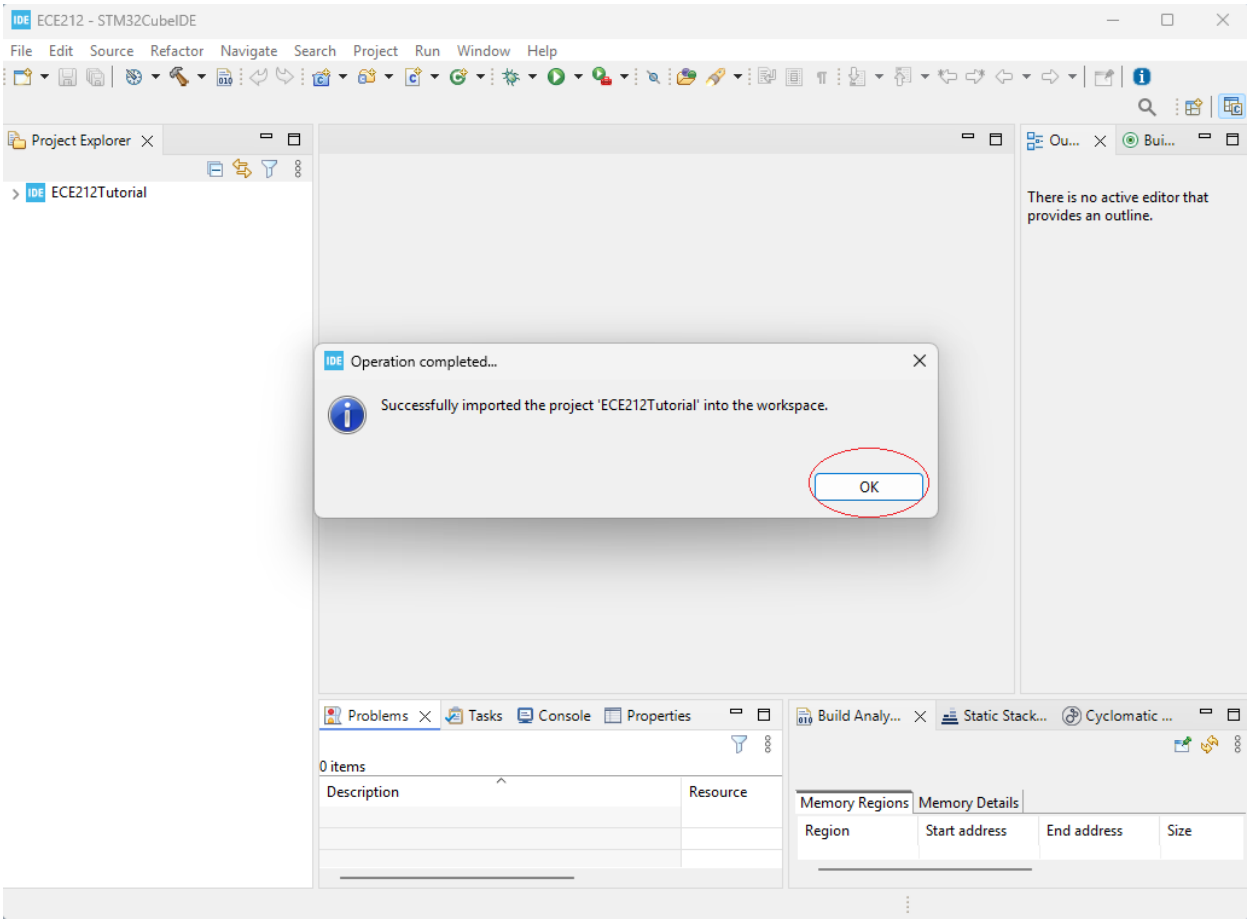

Select an app to open this .project file
PyCharm 2025.3.1
VLC media player
Vi Improved - A Text Editor
Visual Studio Code
Windows Media Player Legacy
Word
IDE
stm32cubeide
Browse apps in the Microsoft Store
Choose an app on your PC
Always
Just once

Click on ‘Browse’ and select the ‘Desktop’ as the directory for the workspace. Finally, click on ‘Launch’ to move on.



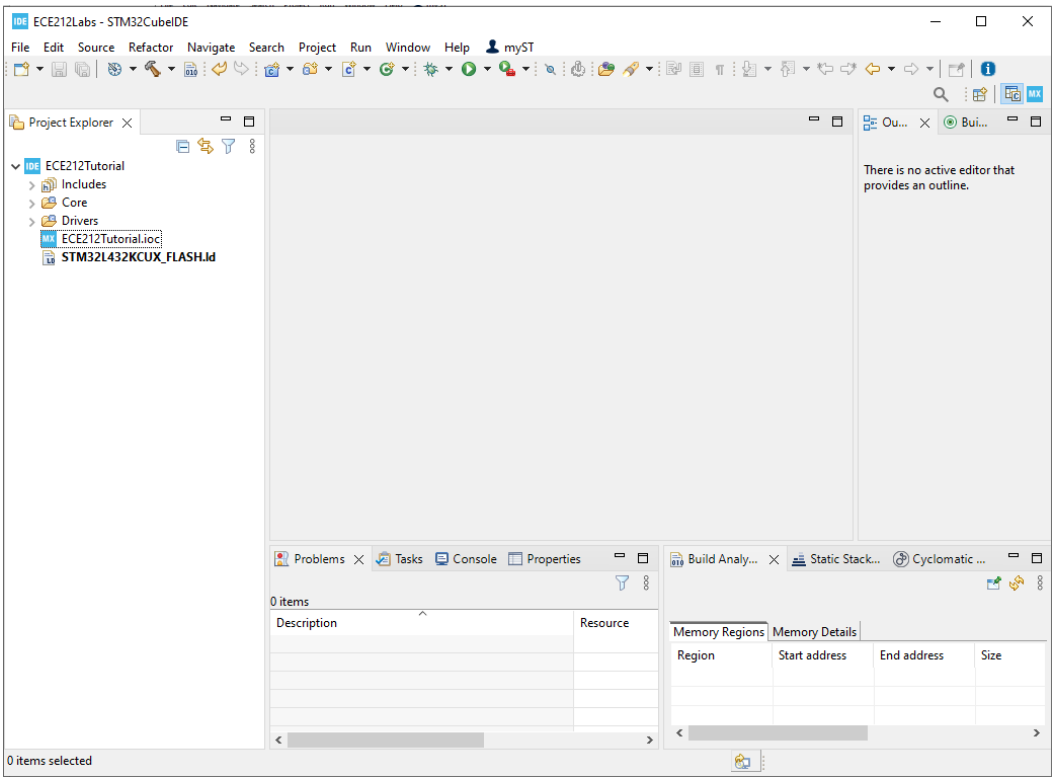


You should see the following screenshot. Click ‘OK’ to continue.


ECE212 - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
ECE212Tutorial
There is no active editor that provides an outline.
Operation completed...
Successfully imported the project 'ECE212Tutorial' into the workspace.
OK
Problems Tasks Console Properties
0 items
Description
Resource
Build Analy... Static Stack... Cyclomatic ...
Memory Regions Memory Details
Region
Start address
End address
Size

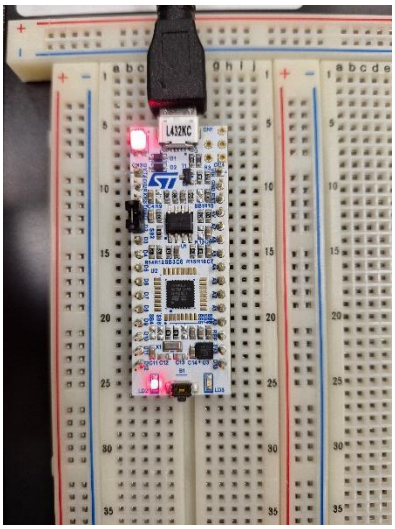
Using STM32CubeIDE

Confirm that you a screenshot similar to the one below

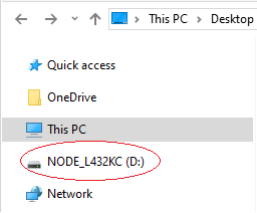


Compile an application

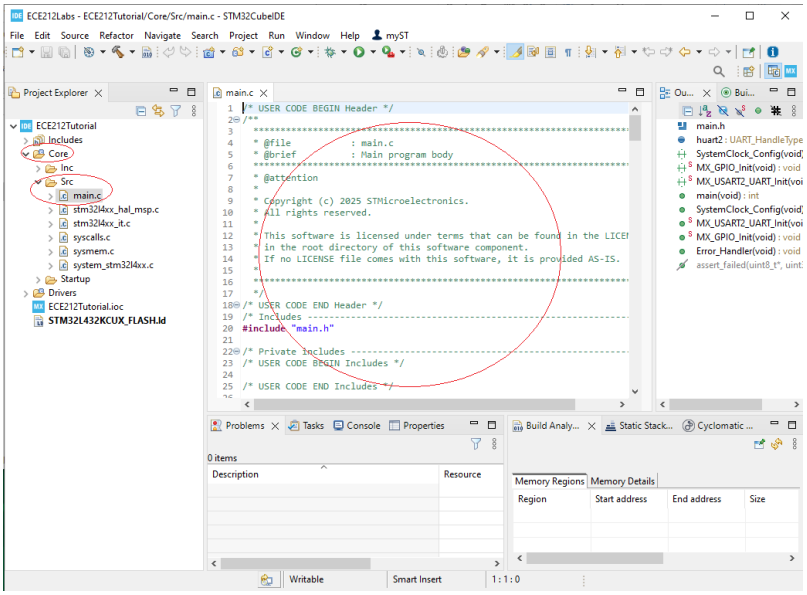
Plug in one end of the USB cable provided into the board and the other end into the computer USB port.



Confirm there is an established connection with the 2 red Leds on. Also check to see that your computer detected the device. We will need to access this drive for lab 4.

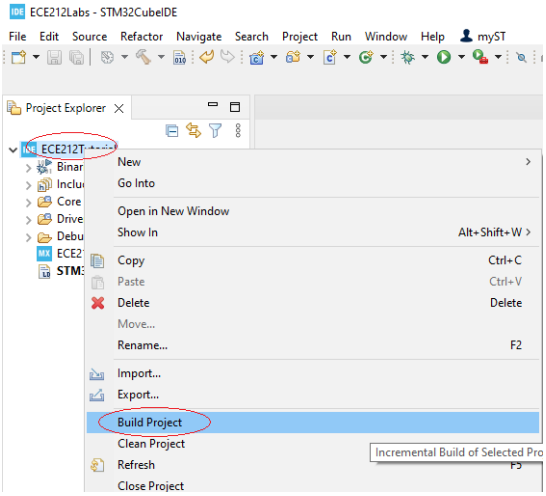


If both are present, you can move on with compiling the blank application. Locate the “main.c” file in the “Core/Src” folder and double click on it to open it up

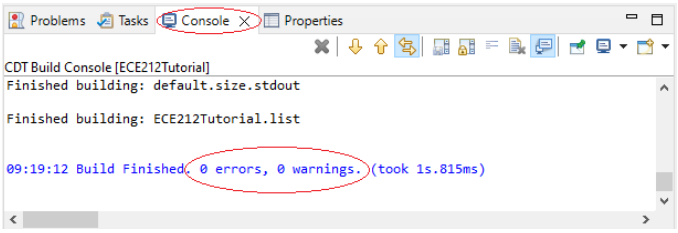


For the workflow, this is where you will be writing your program. However, for our labs, we will always provide the “main.c” file and you will be asked to write your code somewhere else (provided template .s file). For now, you can explore it but you don’t need to know too much about it for this course.

To compile the project, right click on the ECE212Tutorial project folder and select “Build Project”

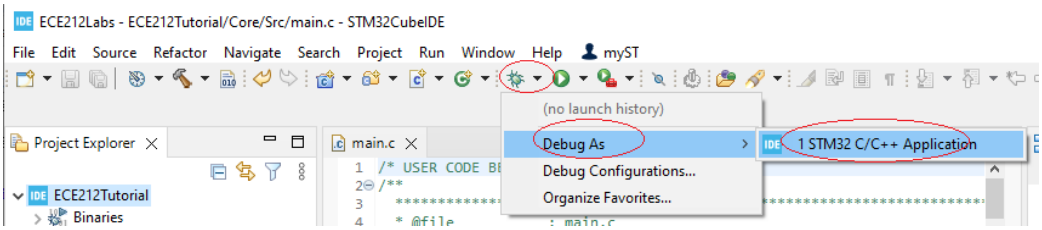


After the project is compiled, you should see in the console window that the project finished building and that there were 0 errors.

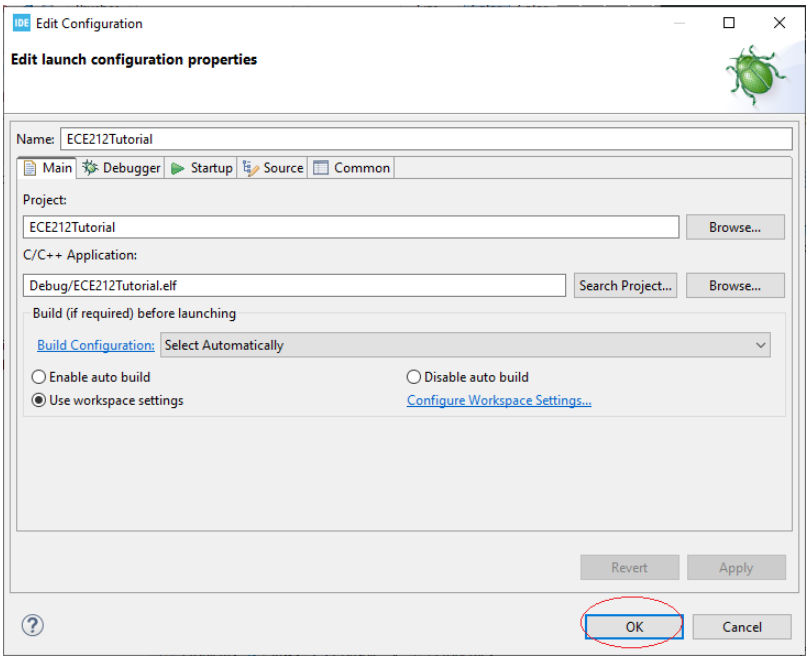


Download the program

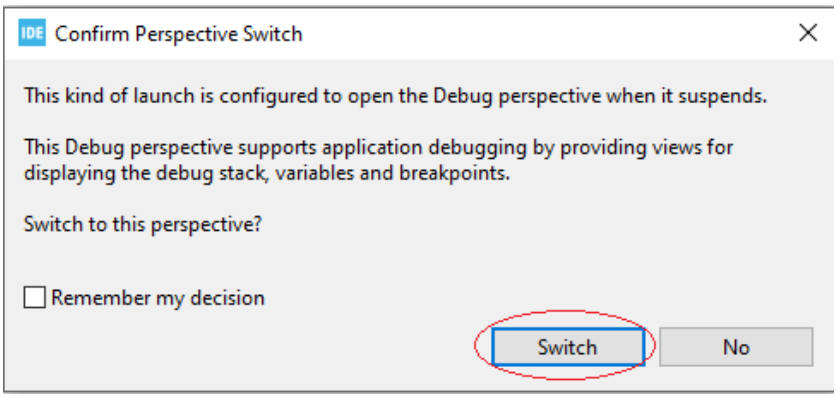
In order to run the program, we can either use the run option or the debug option. We will focus on the debug option in this tutorial because it will provide us with the debugging tools. To launch a debug configuration. Select the **Debug drop down arrow -> “Debug As” ->”1 STM32 Cortex-M C/C++ Application”**



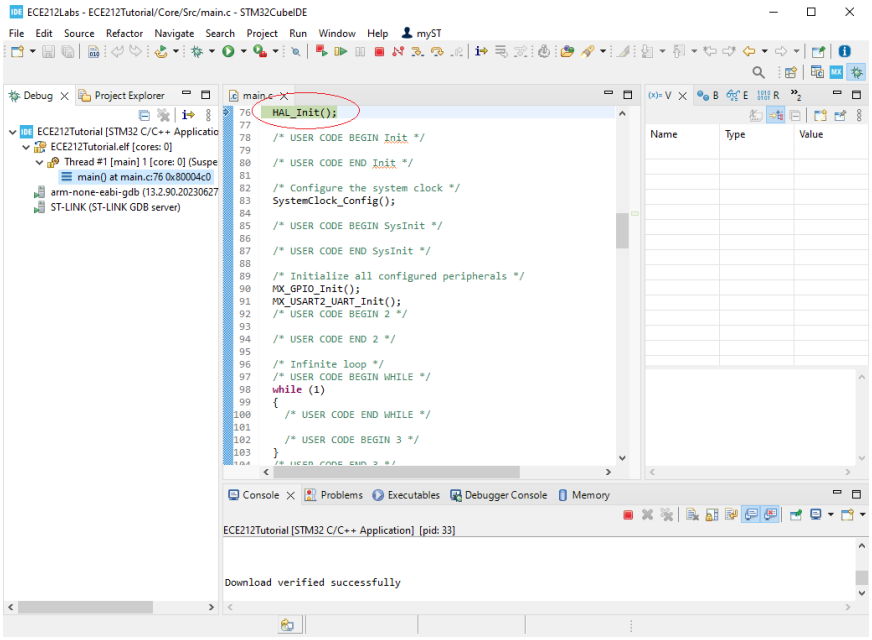
Leave the default settings in the Edit configuration and click “OK”



Open up the Debug perspective by clicking on “Switch” in the confirm perspective pop up window.



You should see a similar screenshot to the one below with the “HAL_Init();” highlighted in green. If you see this, this confirms that the application was successfully downloaded on the NUCLEO Board.

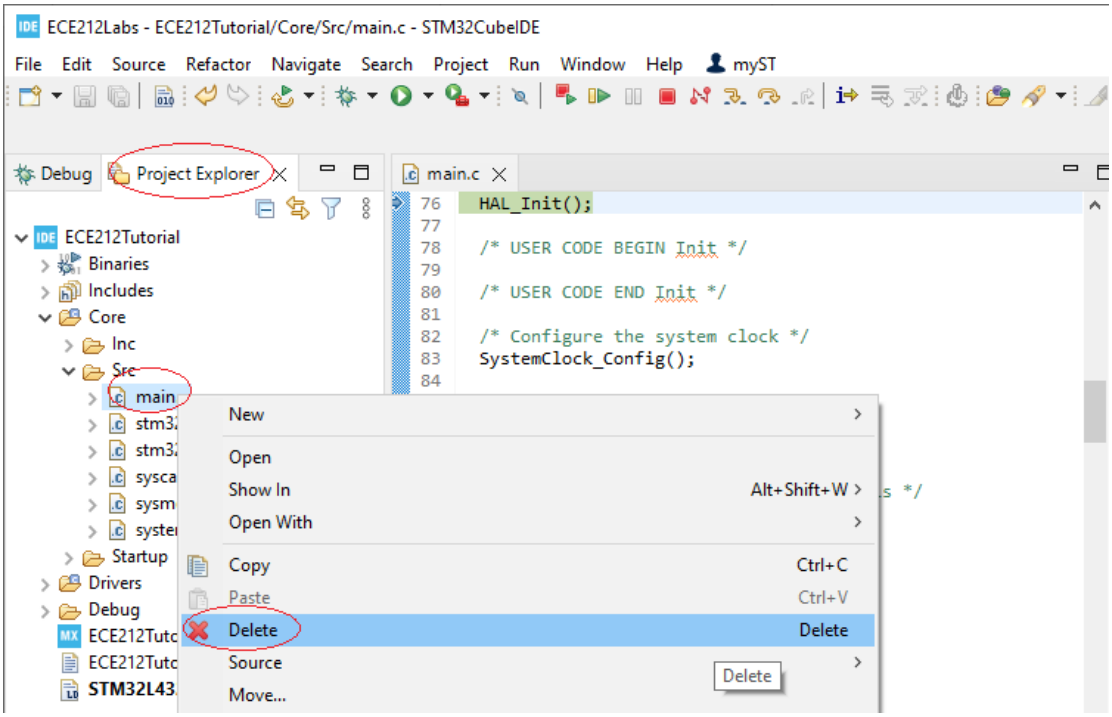


At this point, if you cannot get the above screenshot, please ask the LI/TA for assistance. We need to confirm that everything works before moving onto the next example

Import files into project

In order to communicate with a serial terminal program, we need to import some files to our project. For our example, we will be using the files created from the link https://shawnhymel.com/1873/how-to-use-printf-onstm32/ Full credit is given to the Author from the link above.

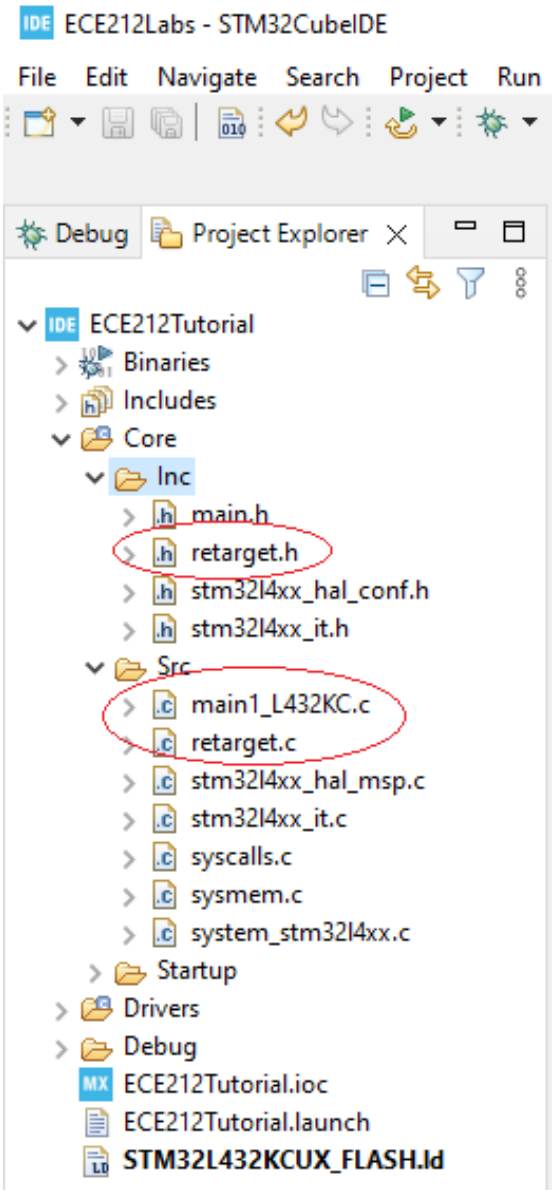
Go back to the “Project Explorer” tab and delete the “main.c” file from the “Core/Src” folder



Click “OK” to confirm the deletion. Download the following files from Canvas and place them in the specified folders.

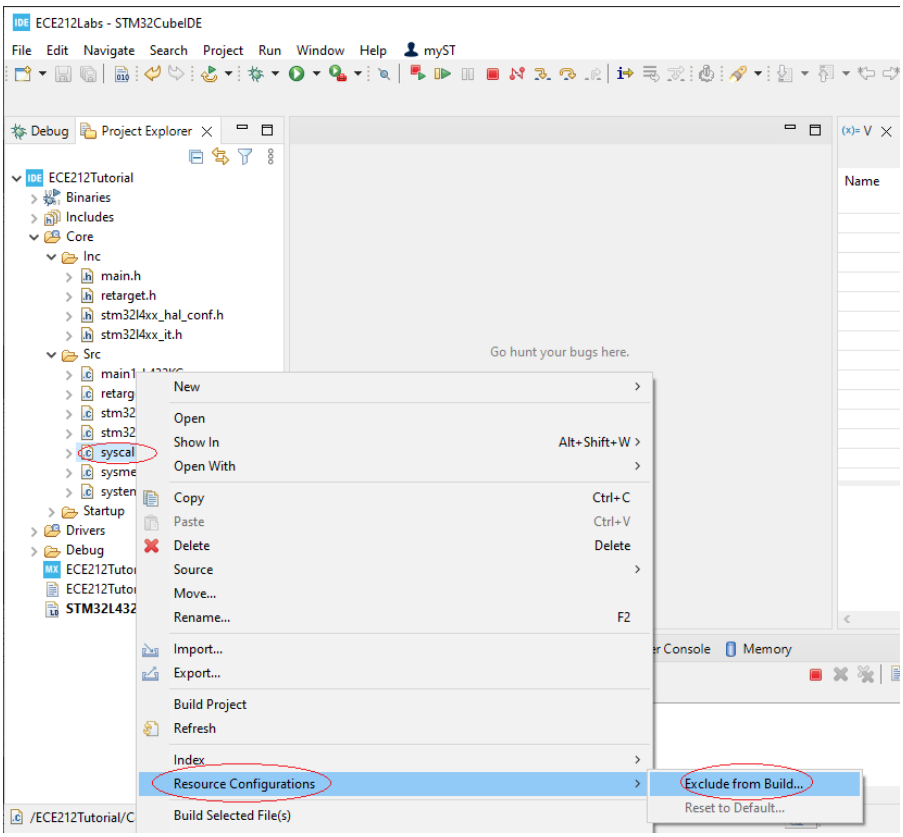
File	Move to location
main1_L432KC.c	Core/Src
retarget.c	
retarget.h	Core/Inc

You can also drag the files directly into the folders within the STM32CUBE IDE program.

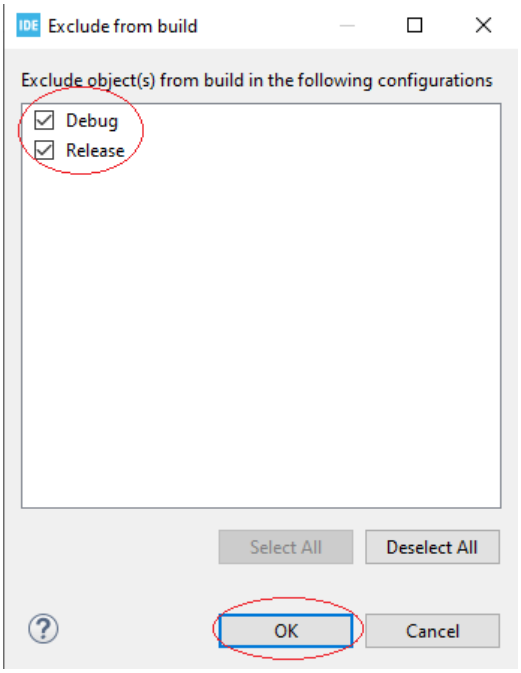


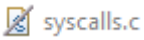
If you don’t see the files in the folder after you place them there, right click within the “Project Explorer” window and select “refresh”

Disable the “syscalls.c” file in the “Core/Src” folder. Select the file, right click with the mouse and select
“Resource Configurations” -> “Exclude from Build”



Exclude the file from both configurations and click ‘OK’

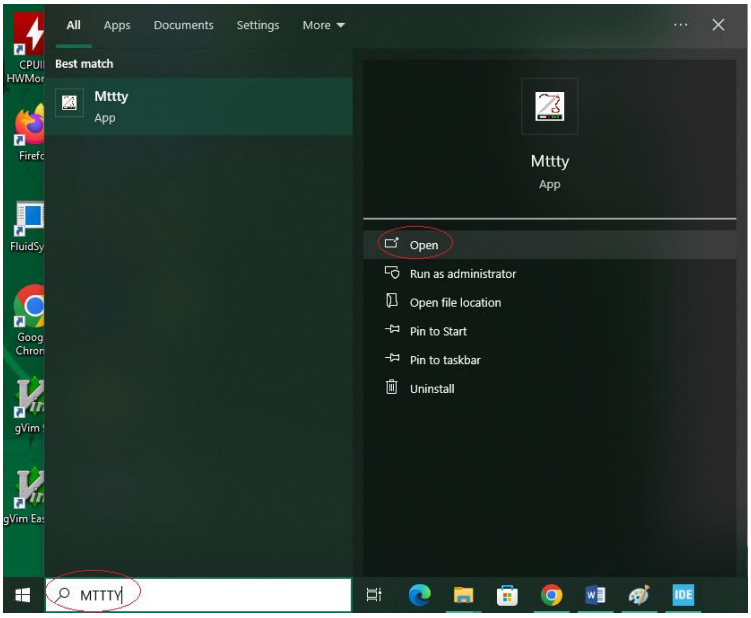


Verify that the “syscalls.c” file has been excluded by checking that the icon is crossed and the name of the file changes color 

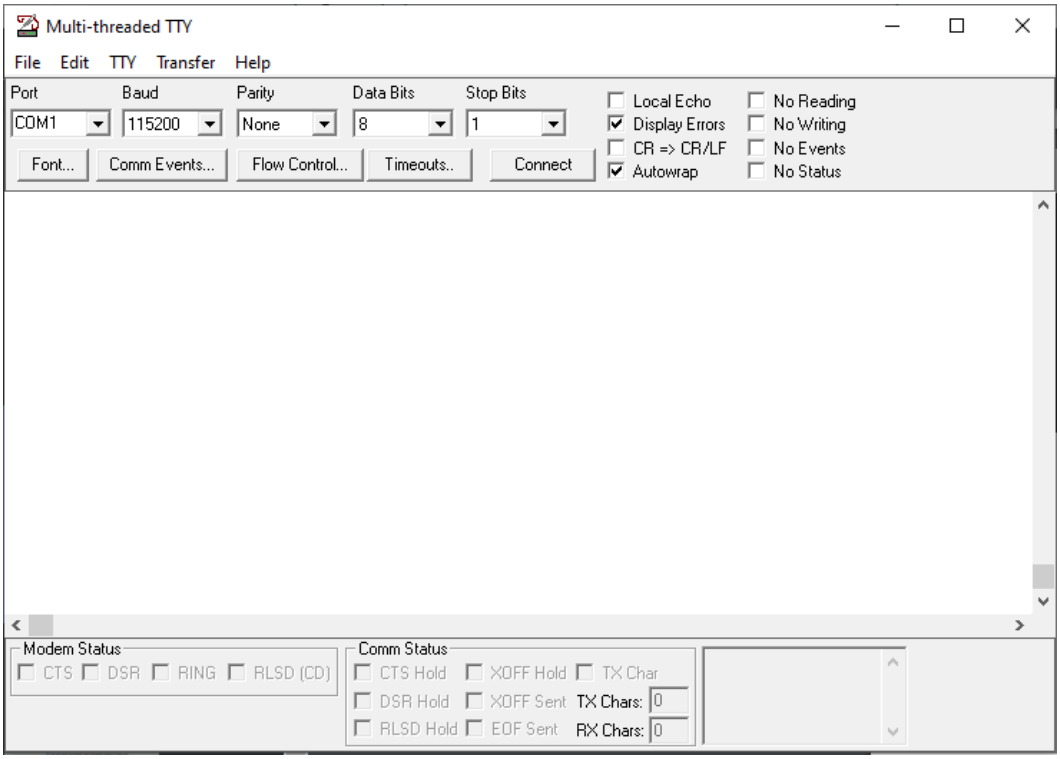
Compile the project with the “Build Project” option covered earlier. Before running the application in Debug mode, launch a serial communication program.

Serial Communication Program

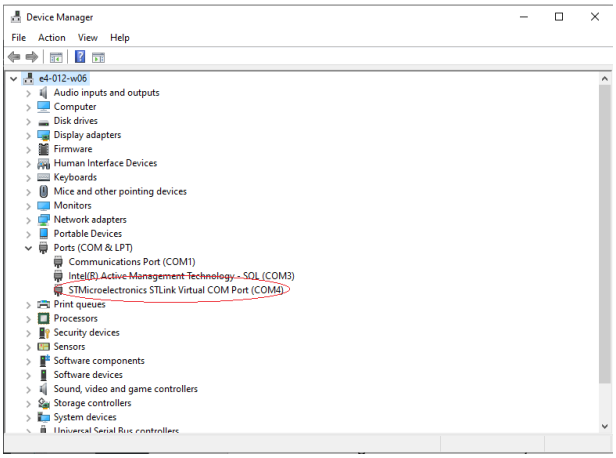
There are many programs available (MTTTY, Putty, HyperTerminal, Minicom,etc) and they all act in similar fashion but only the MTTTY will be discussed here. Find the MTTTY program by performing a search for it and launch the program by clicking on ‘Open’.



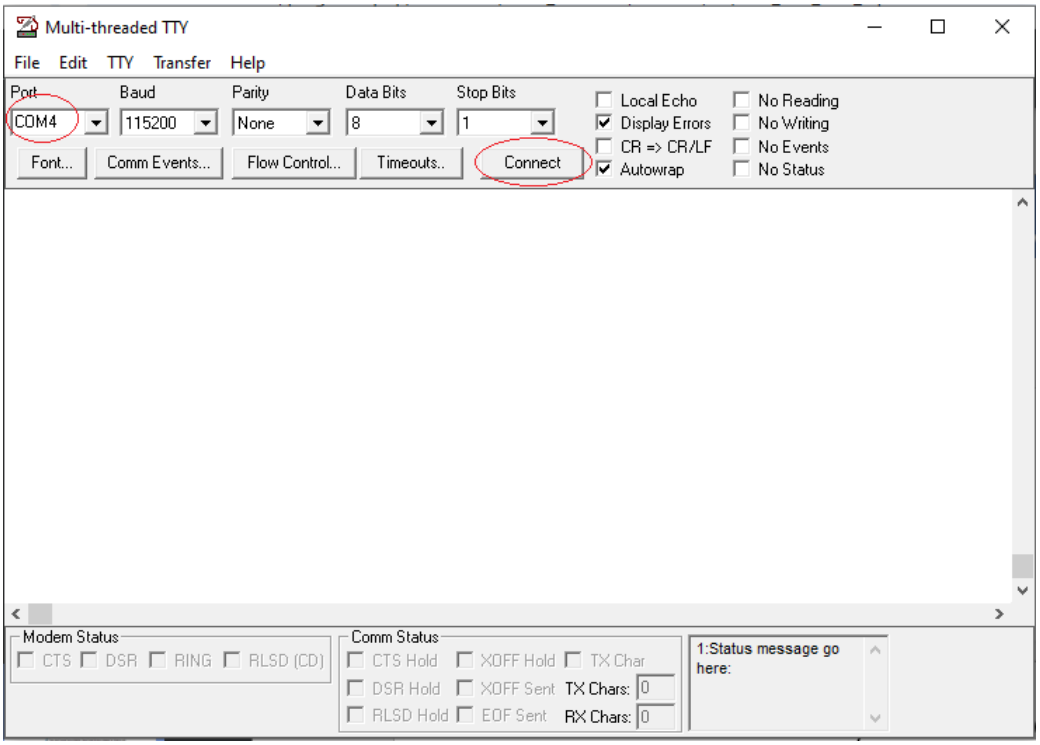
You should see a something similar to the screenshot below



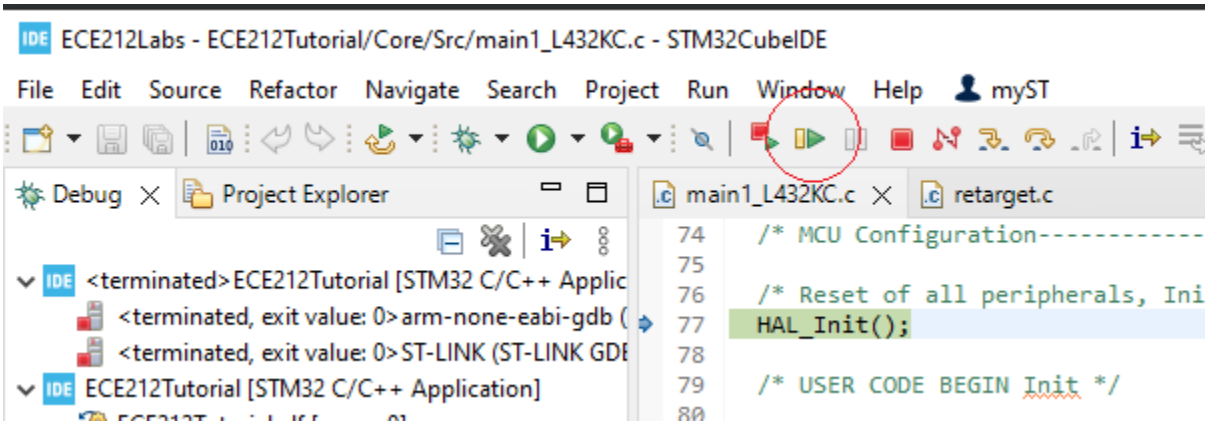
The only thing we need to set is the “Port” channel. If the software doesn’t detect it automatically, you might have to look within your device manager (this will be different for every Operating System, you must know how to access this on your own) to locate the Com channel. Our example is set to ‘COM4’. Yours should be different. Leave all other values as default.



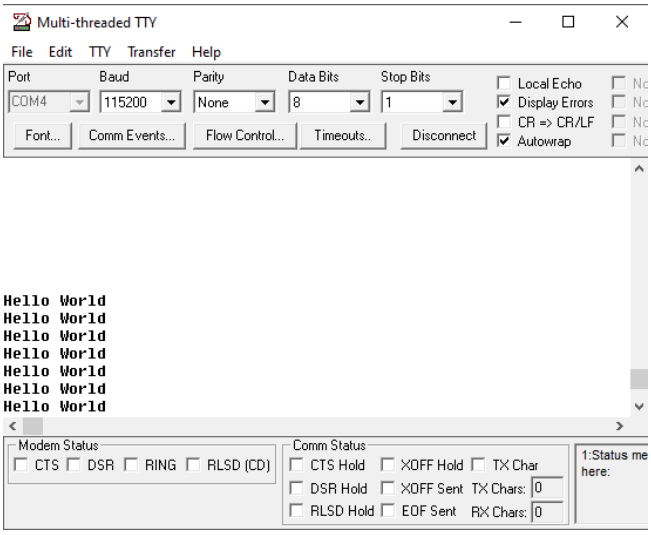
Once the ‘Port’ is set, click on the “Connect” button to establish a connection with the Nucleo board.



Go back to your STM32CubeIDE and launch a debug session using the debug icon. Switch over to the debug perspective by clicking ‘Switch’ if it prompts you to. The program will halt at the instruction “HAL_Init” and wait for your command. Resume the program by clicking on the green triangle.



Once you click on the run button, go back to MTTTY and verify that you are receiving the 'Hello World' message. (*If you don't see anything, you might relaunch the debugger session again)

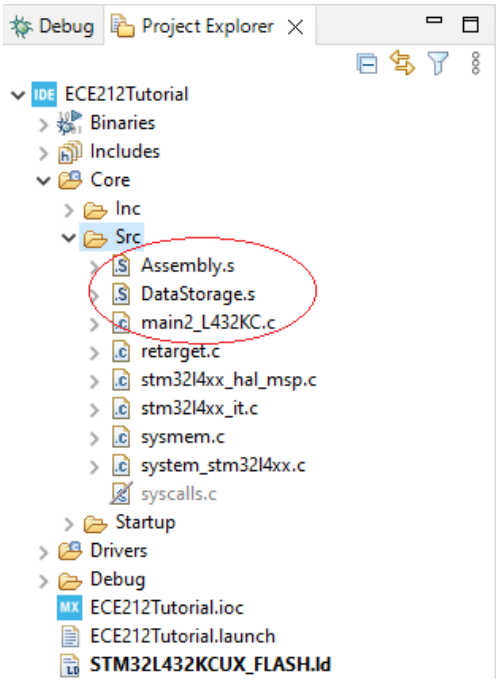


Congratulations, you have successfully compiled a program on the Nucleo board and interfaced it with a Serial Communication program.

Debugging:

Go back to the "Project Explorer" tab and delete the **'main1_L432KC.c'** file from the **'Core/Src'** folder. Download the following files from Eclass and place them in the specified folders.

File	Move to location
main2_L432KC.c	Core/Src
Assembly.s	
DataStorage.s	



Compile the project and launch a debug session. The program will halt at the 'HAL_Init();' instruction.

Breakpoints

Breakpoints are set by double-clicking in the far-left blue shaded area next to the instruction in the edit window. The breakpoint will be marked by a blue dot next to the instruction. Place a breakpoint next to the line "TestAsmCall();" by double clicking on the blue area next to it. Double clicking on the breakpoint again will get rid of it. If you right click on the breakpoint with the mouse, it will bring up additional

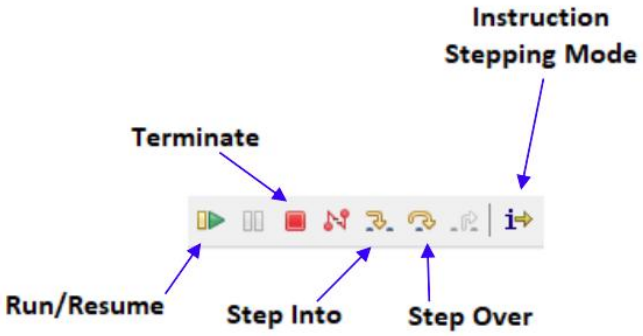
options.

```
113   while (1)
114   {
115 /*---------Added by Lab Tech-------------------
116 Intialization();
117 TestAsmCall();
118 HAL_Delay(1000);
119 /*----------------------------------------*/
120     /* USER CODE END WHILE */
```

Once a breakpoint is placed, the program will halt at that line of code when it reaches it. Click on the resume button  and confirm that the application halted at the “TestAsmCall();” (notice that the breakpoint symbol now has an arrow on top).

```
115 /*---------Added by Lab Tech-------------------
116 Intialization();
117 TestAsmCall();
118 HAL_Delay(1000);
119 /*----------------------------------------*/
120     /* USER CODE END WHILE */
```

Other useful options that are available are indicated below



The first active icon is the green “Run/Resume” button in the “debugger” window. Clicking on it will resume the application until it hits the next breakpoint. The red “Terminate” button ends the debug session. The yellow arrows from left to right are “Step Into”, “Step Over”, and “Step Return”. These enable you to perform a single step to the next line (Step Over), step into a function/subroutine (Step Into), or

return out of a function/subroutine (Step Return).

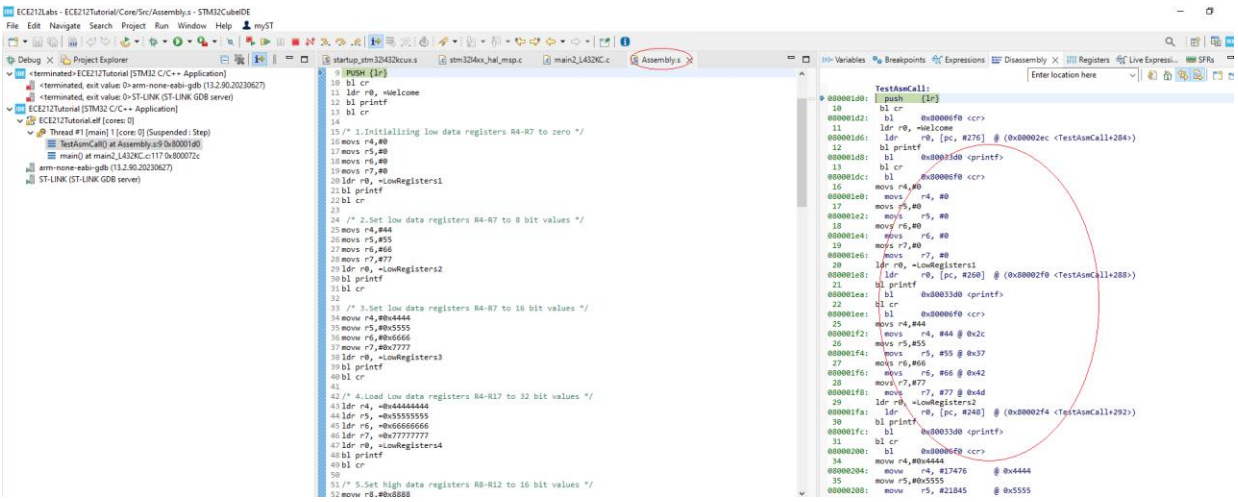
Click on the “Step Over” button to see what happens

```
115  /*---------Added by Lab Tech-------------------*/
116  Intialization();
117  TestAsmCall();
118  HAL_Delay(1000);
119  /*----------------------------------------*/
```

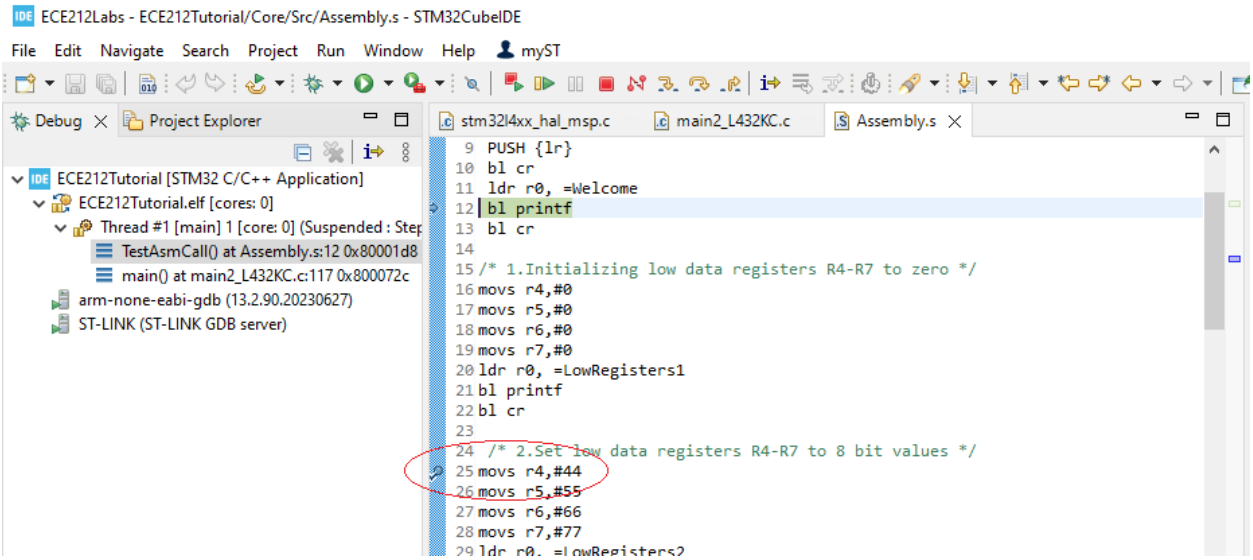
Notice that our program is now halted at “HAL_Delay(1000);” which is the line after the “TestAsmCall();”. You can tell where you are in the program by locating the arrow in the blue shaded area or the line of code that is highlighted in green. Resume our program by clicking the “Resume” button

The program ran once and it halted back at the breakpoint set at the instruction “TestAsmCall();”. Now, instead of stepping over this line of code, step into it by clicking on the “Instruction Stepping Mode”  and then the “Step Into”  button. This will allow the program to jump into the “TestAsmCall();” subroutine which is located in the “Assembly.s” file

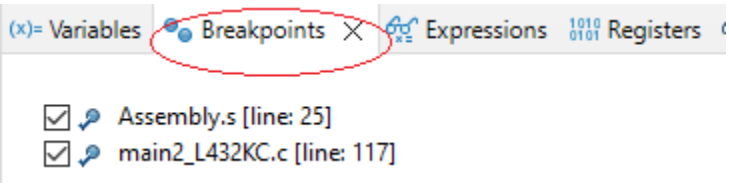
In addition, you will also be given the “Disassembly” window which offers additional information on the instructions(Opcode, memory location, etc).



We can step through our assembly language code one line at a time by clicking on the “Step Over” button. Try it a few times to get comfortable with it. You can also place breakpoints in the “Assembly.s” file (preferred method). Place a breakpoint at the line “movs r4,#44”



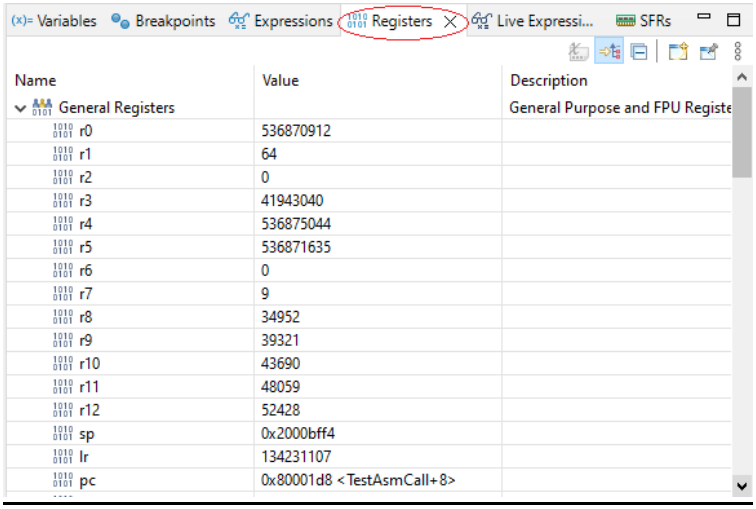
As an aside, to confirm that we have two breakpoints, you can go to the left pane and click on the “Breakpoints” tab for viewing. There should be two breakpoints, one in “main2.c” and one in “Assembly.s”. (*NOTE – Maximum of 7 breakpoints is allowed at one time)



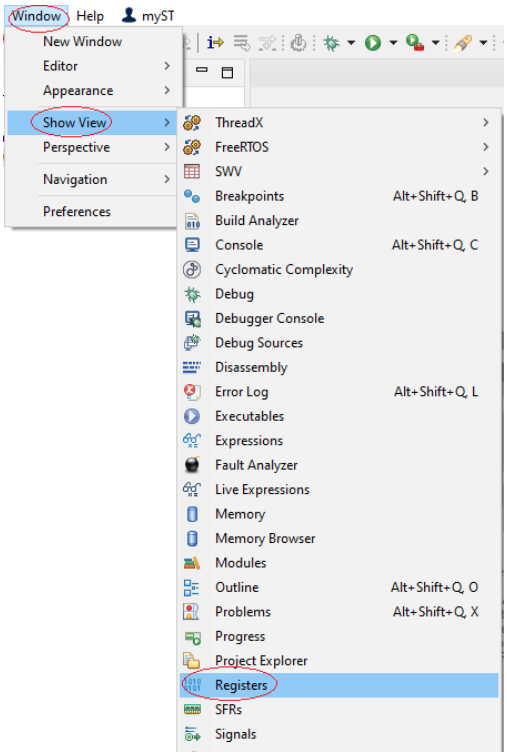
Click on the “Resume” button a few times to observe what happens. The program should halt at each breakpoint. This is how we debug the program by placing breakpoints in the code at certain spots and then using the Step Into/Step Over to execute the next line of code

Registers

Now that we know how to halt our program at specific points in our code, there are tools available to for monitoring the Registers and Memory Content. To see the Registers, click on the register tab.

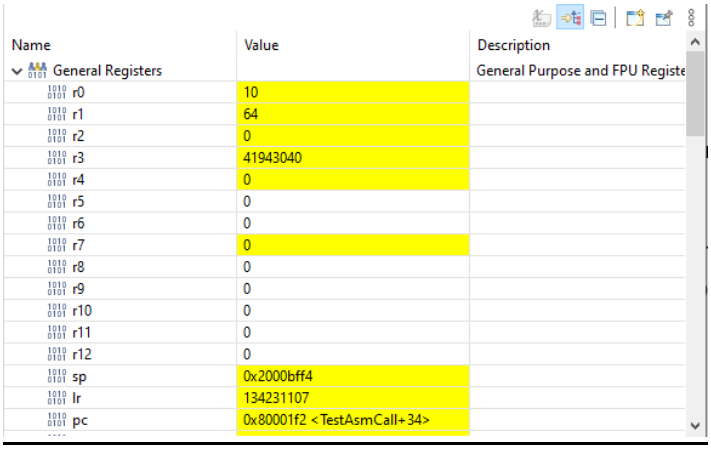


Alternative, you can also access it by Windows->Show View -> Registers



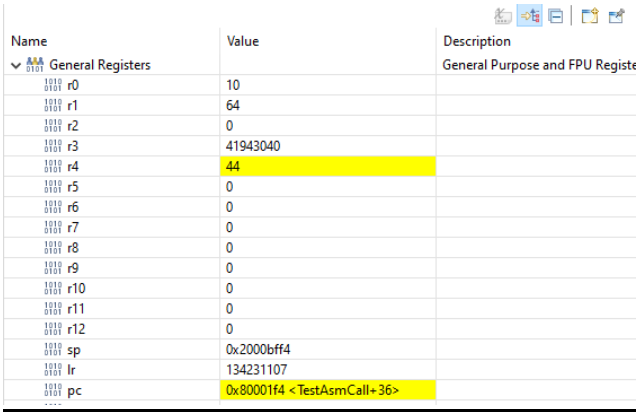
When you are stepping through the code, the contents of the registers will get updated accordingly. Step over each line and watch the register values change. We had previously placed a breakpoint at “movs r4,#44”. Run the program until you reach this breakpoint.

Now bring up the Register Window. Registers r4-r7 should be cleared to values of zero. The values highlighted in yellow indicate that they were modified/updated from previous values. Do not worry about the contents in r0-r3/r8- r12.



Name	Value	Description
General Registers		General Purpose and FPU Registe
r0	10	
r1	64	
r2	0	
r3	41943040	
r4	0	
r5	0	
r6	0	
r7	0	
r8	0	
r9	0	
r10	0	
r11	0	
r12	0	
sp	0x2000bff4	
lr	134231107	
pc	0x80001f2 <TestAsmCall+34>	

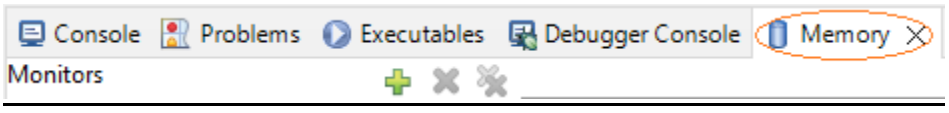
Now try stepping over one instruction at a time. Step over the instruction “movs r4,#44” and monitor the Register r4.



Name	Value	Description
General Registers		General Purpose and FPU Registe
r0	10	
r1	64	
r2	0	
r3	41943040	
r4	44	
r5	0	
r6	0	
r7	0	
r8	0	
r9	0	
r10	0	
r11	0	
r12	0	
sp	0x2000bff4	
lr	134231107	
pc	0x80001f4 <TestAsmCall+36>	

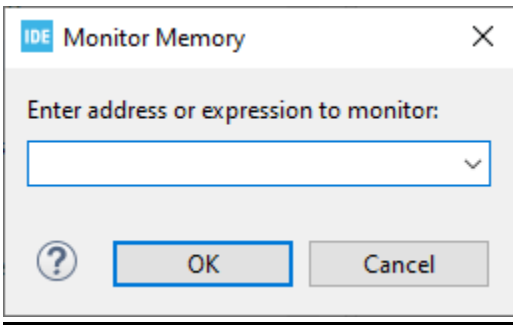
Memory

To view the internal memory, bring up the memory monitor by clicking on the Memory icon



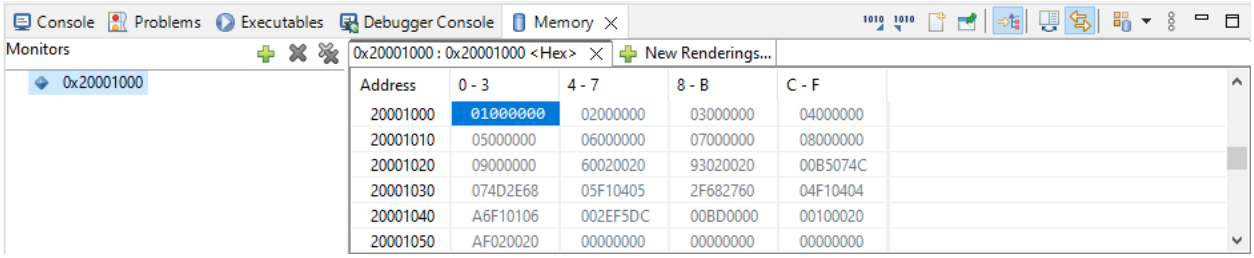
If you don’t see it, you can also bring it up by clicking on Windows -> Show View -> Memory

Click on the green “+” sign to add the memory address in which you want to monitor



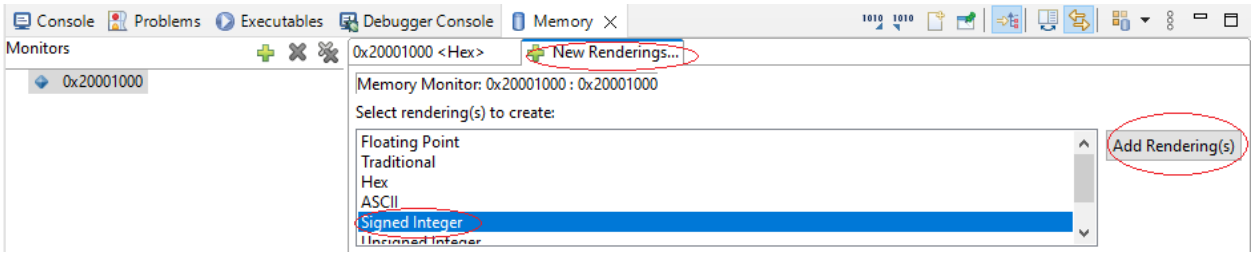
Enter the value 0x20001000 in the memory monitor and click “OK”

The memory address should now be displayed in the memory monitor along with the data

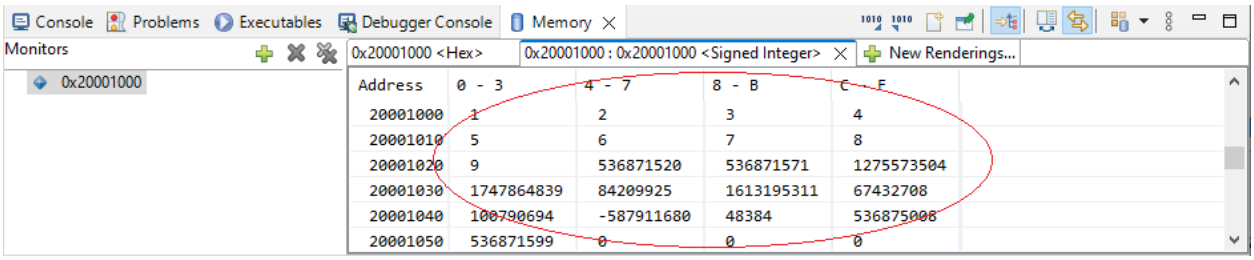


Change the view of the data from Hex to Signed Integer for more clarity by clicking on the “New Rendering” tab.

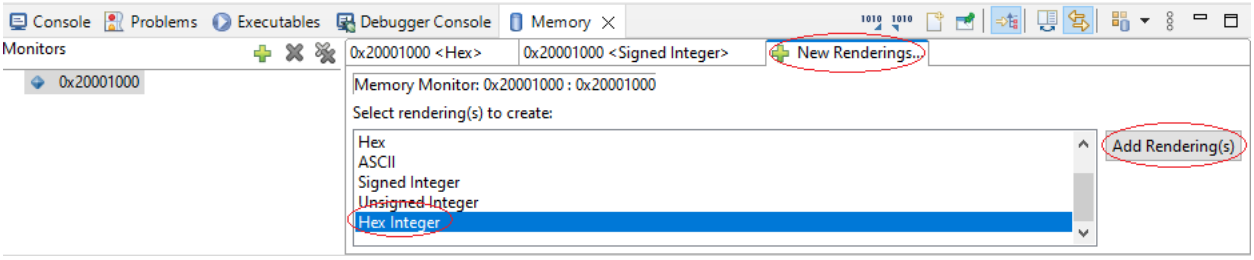
Select the sign integer option and click on “Add Rendering”



You should now see the integer numbers from 1-9 located from 0x20001000 to 0x20001020. Each number occupies 32 bits(4bytes) of memory



In most cases, the default “Hex” view should not be used. If you are working with Hex values, it’s better to change the data view to “Hex Integer”



To view data as strings (ASCII Text) data, place a breakpoint at “ldr r0, =Welcome” and run the program until it halts at that breakpoint

Step over the “ldr r0, =Welcome” instruction and copy the value in register r0 to the memory monitor and open up the memory monitor at that location to view the data

In our example, the memory address 536870912(0x20000000) contains the data 0x57. Although this may not seem apparent, the data 0x57 represents the ASCII character “W”.

536870912 : 0x20000000 <Hex> | New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
20000000	57656C63	6F6D6520	746F2074	68652074
20000010	75746F72	69616C00	312E4D6F	6E69746F
20000020	72207468	65204C6F	77205265	67697374
20000030	65727328	496E6974	69616C69	7A652920
20000040	72342D72	3700322E	4D6F6E69	746F7220
20000050	74686520	4C6F7720	52656769	73746572

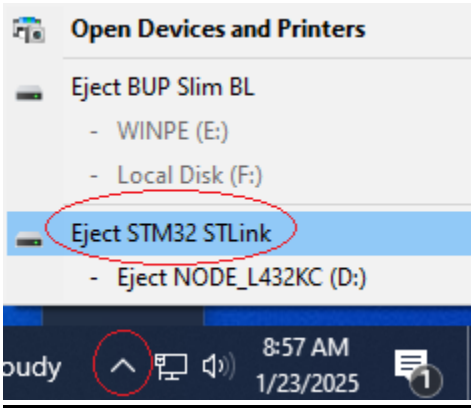
To see the data more clearly, add a New Rendering and select ASCII.

536870912 <Hex> | 536870912 : 0x20000000 <ASCII> | New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
20000000	Welc	ome	to t	he t
20000010	utor	ial	1.Mo	nito
20000020	r th	e Lo	w Re	gist
20000030	ers(	Init	iali	ze)
20000040	r4-r	7 2.	Moni	tor
20000050	the	Low	Regi	ster

This concludes the tutorial.

Please proceed to complete the Lab0 Quiz. After you are done, unmount the board by the screenshot below.



Clean up your desk before leaving.