

# Tutorial 4: Debugging with STM32CubeIDE

**Instructor:**

Dr. Carl Latino

[carl.latino@okstate.edu](mailto:carl.latino@okstate.edu)

**Graduate Teaching Assistants:**

Francisco E. Fernandes Jr.

[feferna@okstate.edu](mailto:feferna@okstate.edu)

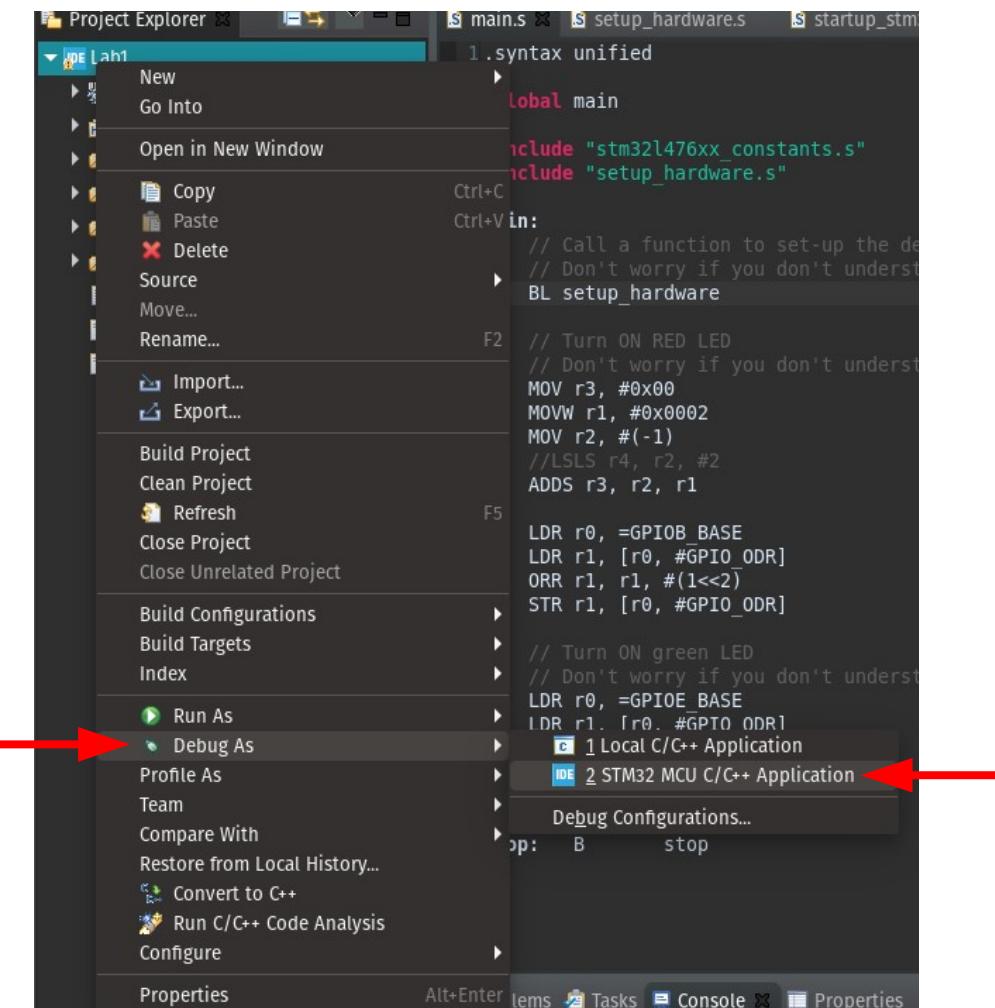
**School of Electrical and Computer Engineering  
Oklahoma State University  
Fall 2019**



- Debugging is hard enough – programs are wrong and they do not work many times. This is especially true the first time you try to compile a program. In fact, I do not ever remember an instance where a program worked the first time I compiled it.
- To help with this issue, companies have inserted debugging software tools into their compiler. These debugging tools are helpful in that enable you to find out what's going wrong quickly. They enable you to scan memory as well as see what is stored in a particular variable. This is extremely helpful especially when dealing with external or global variables.

# Entering the debugging environment

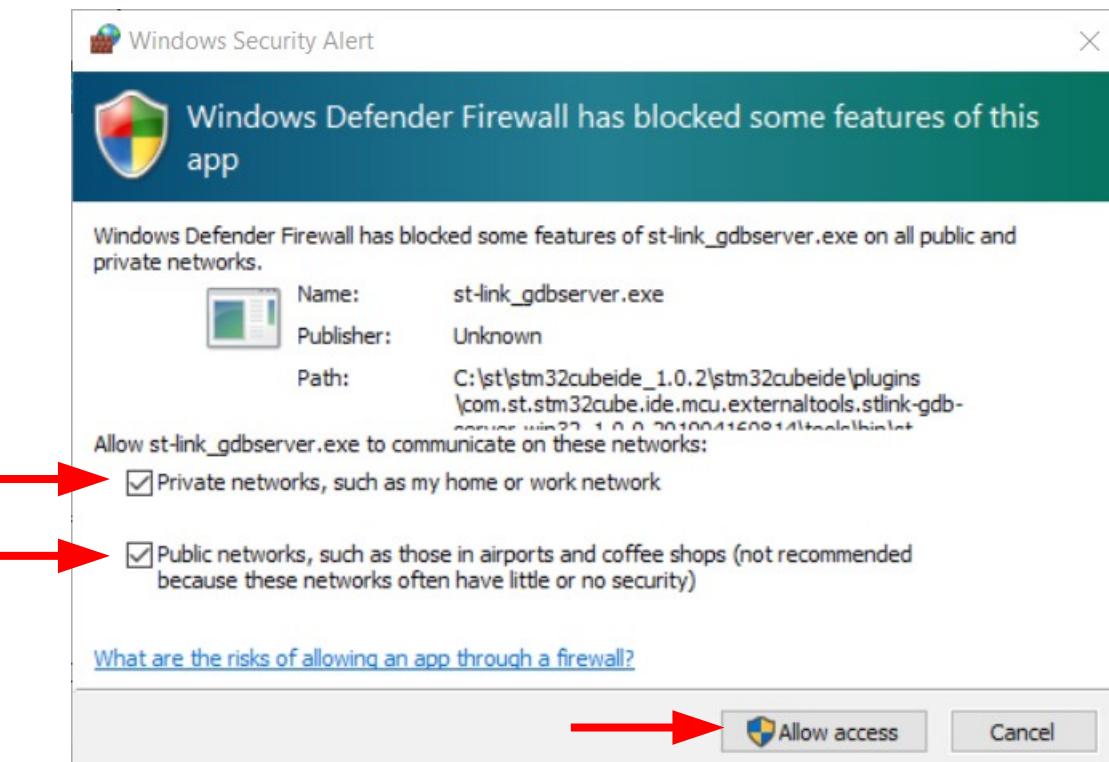
- After you have built your project, you can debug it.
- Right-click on your project name, and select **Debug as → STM32 MCU C/C++ Application**.



Make sure your development board is connected to your PC!

# Entering the debugging environment

- If this is your first time debugging your code, the **Windows Firewall** will ask your permission to connect to the network.
- Make sure to **allow access** in the new screen that will show up!

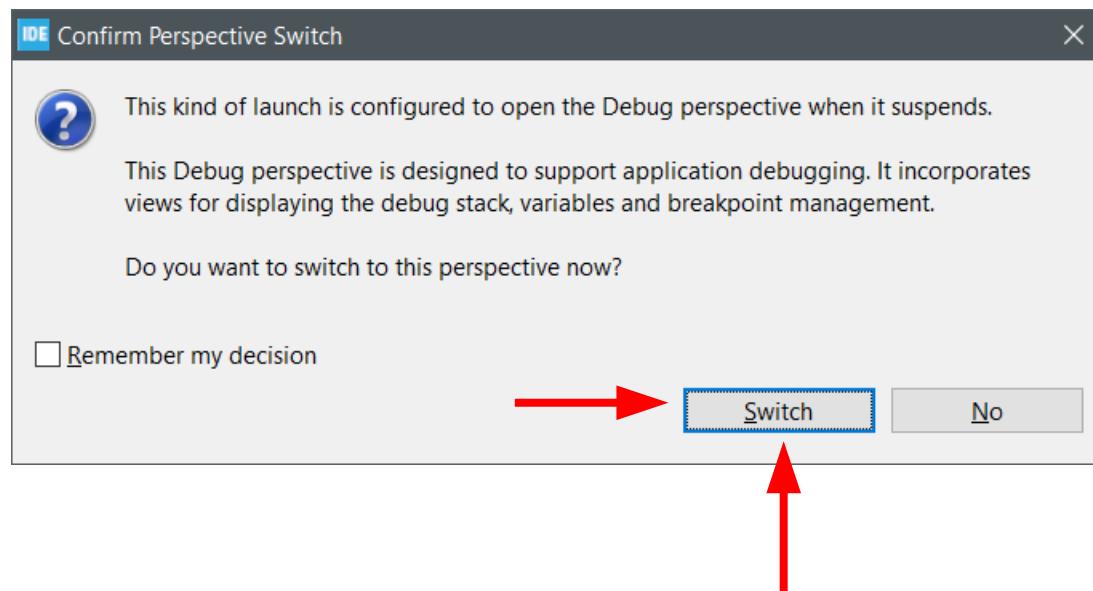


**Note:** If you are using a lab's computers, the T.A. will need to enable the access to the network.

# Entering the debugging environment



- A window will pop-up asking if you want to switch to the **Debug** view. Click on **Switch**.
- Don't panic, but your IDE will look different now! See the next pages.



# Entering the debugging environment



- This is the **Debug** view of the IDE. Your code is on the **development board**, but it is **not running yet**.

workspace\_1.0.2 - Lab1/Src/main.c - STM32CubeIDE

File Edit Navigate Search Project Run Window Help

Debug Project Explorer

Lab1.elf [STM32 MCU Debugging]

Lab1.elf [cores: 0]

Thread #1 [main] 1 Core: 0 (Suspended : B)

main() at main.c:11 0x80001f0

/opt/st/stm32cubeide\_1.0.2/plugins/com.st

ST-LINK (ST-LINK GDB server)

main:

```
2 .global main
3
4 .include "stm32l476xx_constants.s"
5 .include "setup_hardware.s"
6
7
8 main:
9 // Call a function to set-up the development board's hardware.
10 // Don't worry if you don't understand the code yet.
11 BL setup_hardware
12
13 // Turn ON RED LED
14 // Don't worry if you don't understand the code yet.
15 MOV r3, #0x00
16 MOVW r1, #0x0002
17 MOV r2, #(-1)
18 //LSLS r4, r2, #2
19 ADDS r3, r2, r1
20
21 LDR r0, =GPIOB_BASE
22 LDR r1, [r0, #GPIO_ODR]
23 ORR r1, r1, #1<<2
24 STR r1, [r0, #GPIO_ODR]
25
26 // Turn ON green LED
27 // Don't worry if you don't understand the code yet.
28 LDR r0, =GPIOE_BASE
29 LDR r1, [r0, #GPIO_ODR]
30 ORR r1, r1, #1<<8
31 STR r1, [r0, #GPIO_ODR]
32
33 // dead loop & program hangs here
34 stop: B stop
35
```

Console Registers Progress Problems Executables Debugger Console Memory

Lab1.elf [STM32 MCU Debugging] ST-LINK (ST-LINK GDB server)

Erasing memory corresponding to segment .

Erasing internal memory sector 0

Download in Progress:

File download complete  
Time elapsed during download operation: 00:00:00.220

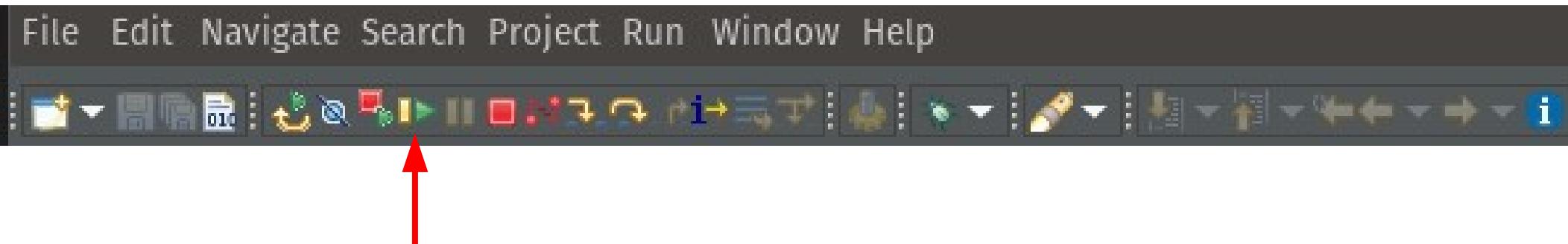
Verifying ...

Download verified successfully

# Understanding the debugging environment



- You should familiarize yourself with the buttons indicated in the picture below. They are the most important ones to using when debugging your code.

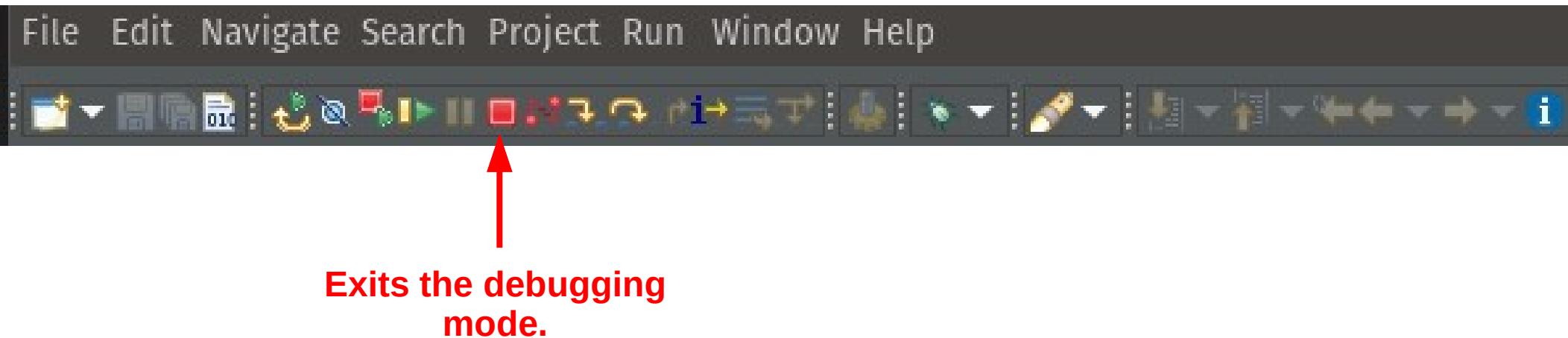


**Runs your code in its entirety. It will only stop running manually or in a breakpoint.**

# Understanding the debugging environment



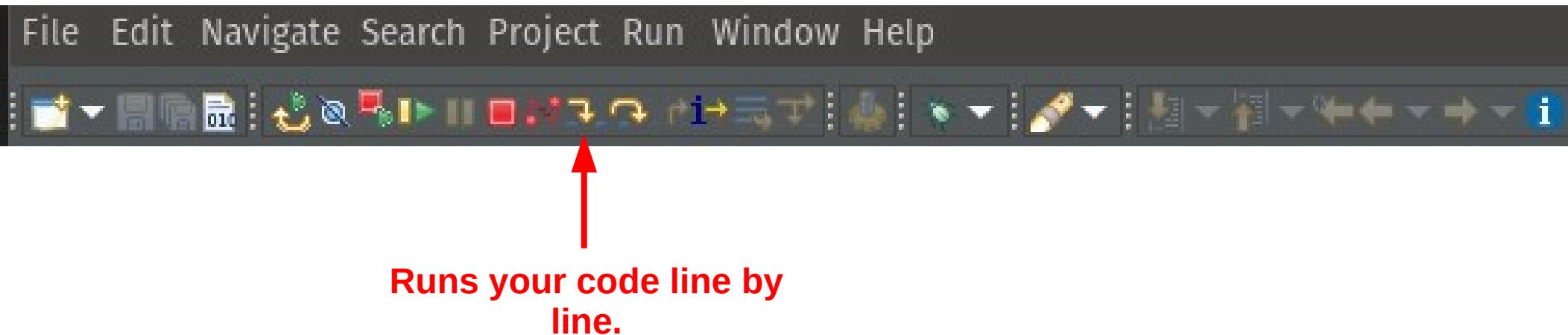
- You should familiarize yourself with the buttons indicated in the picture below. They are the most important ones to using when debugging your code.



# Understanding the debugging environment



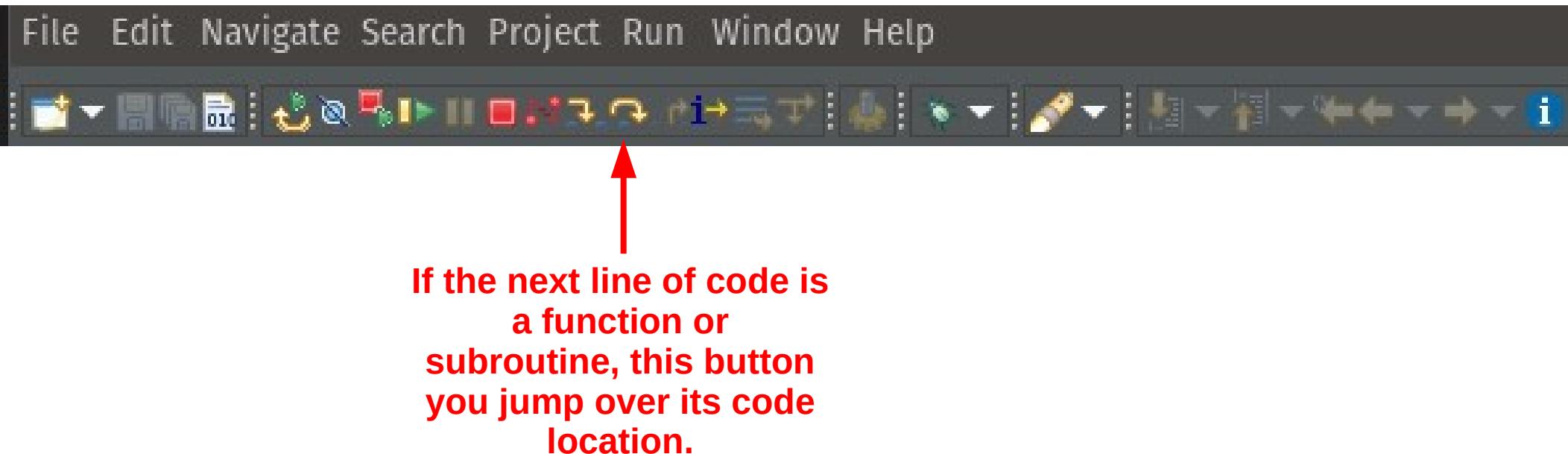
- You should familiarize yourself with the buttons indicated in the picture below. They are the most important ones to using when debugging your code.



# Understanding the debugging environment



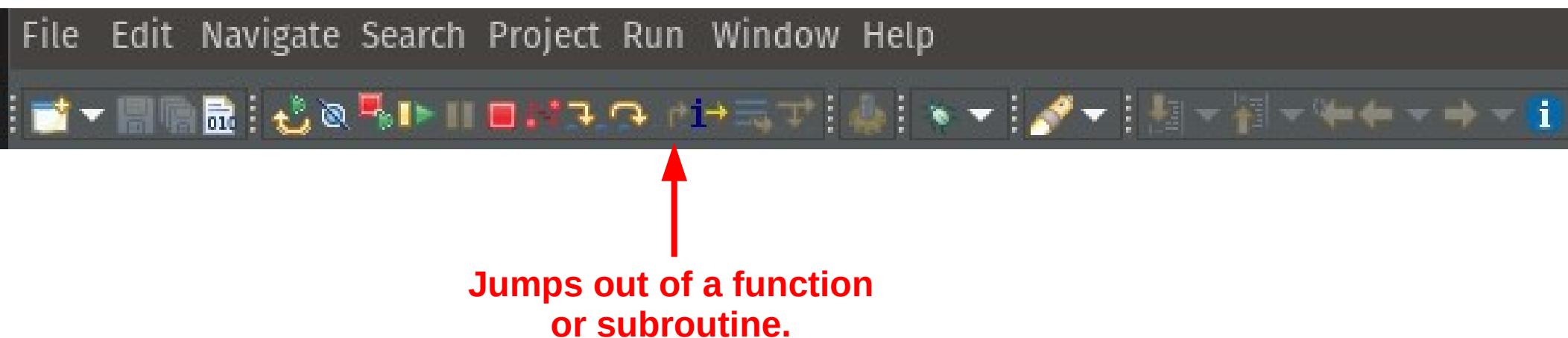
- You should familiarize yourself with the buttons indicated in the picture below. They are the most important ones to using when debugging your code.



# Understanding the debugging environment



- You should familiarize yourself with the buttons indicated in the picture below. They are the most important ones to using when debugging your code.

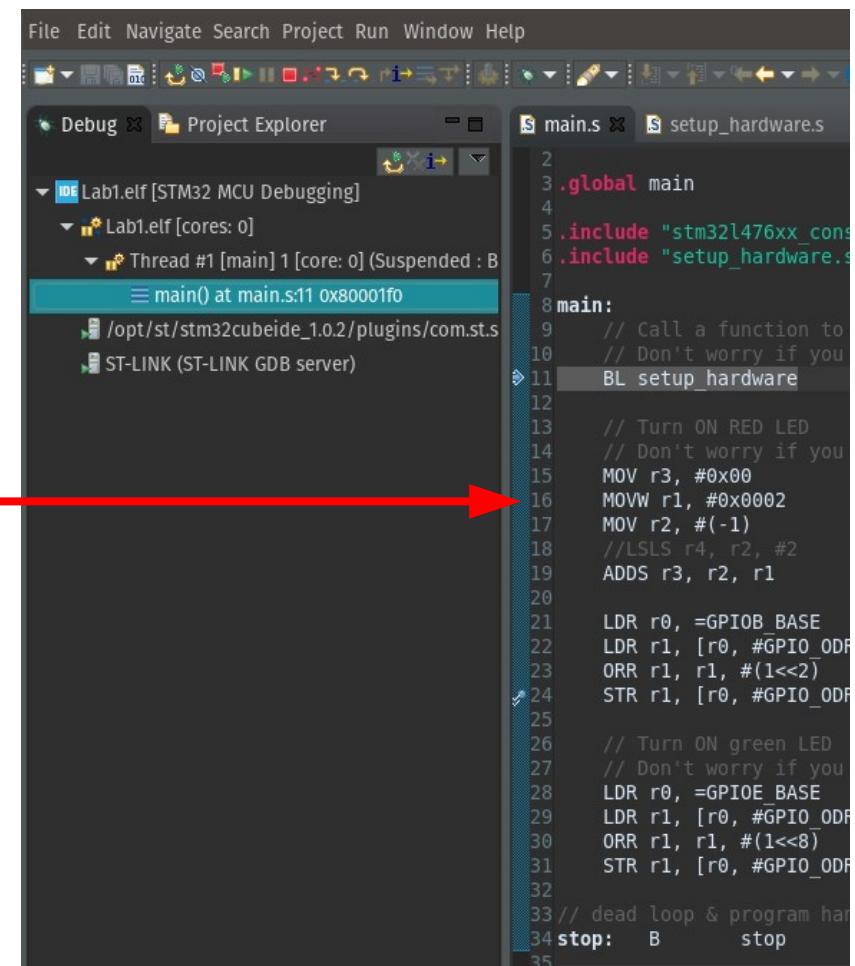


# Inserting Breakpoints in your code



- Breakpoints can help you debug your code by stopping execution when a desired line of code is reached.
- To set a breakpoint, right-click in the space to the left of the number line and select **Toggle Breakpoint**.

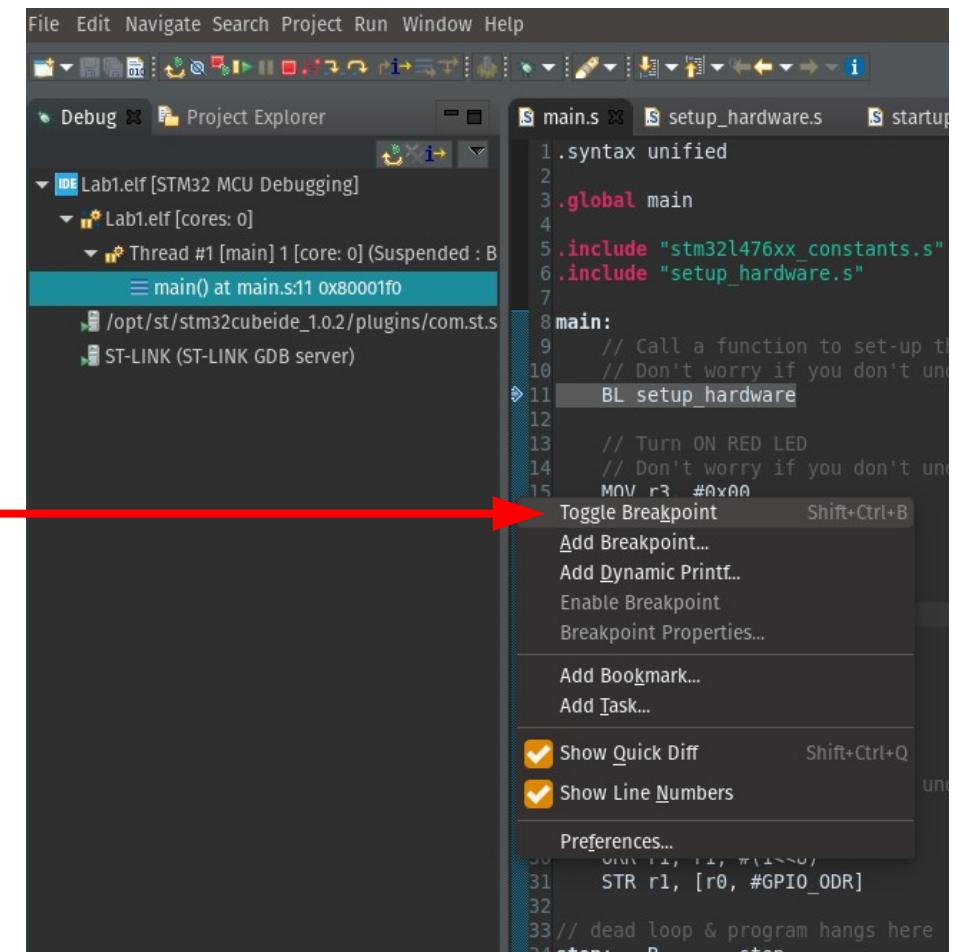
Right-click in this blue strip.



# Inserting Breakpoints in your code



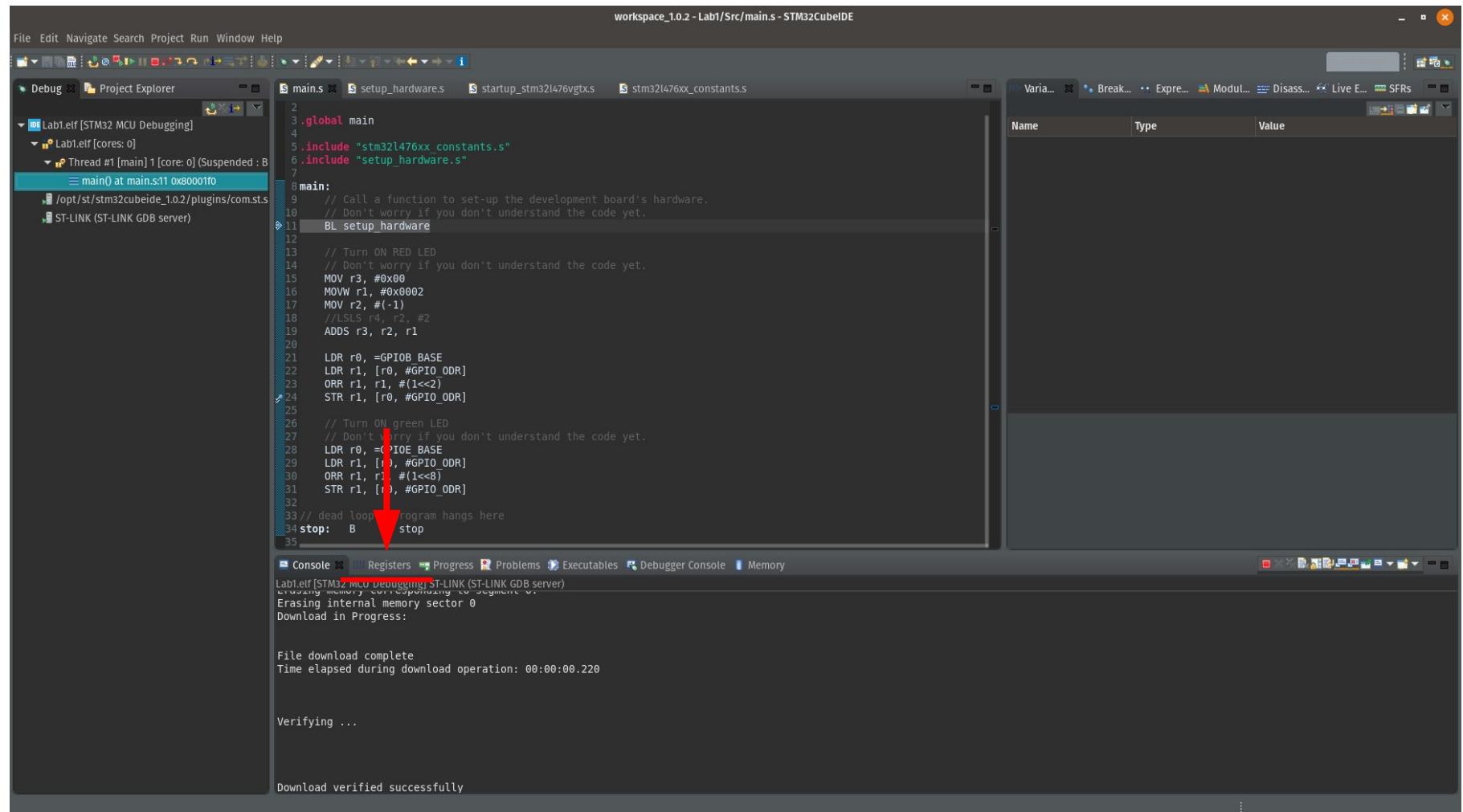
- Breakpoints can help you debug your code by stopping execution when a desired line of code is reached.
- To set a breakpoint, right-click in the space to the left of the number line and select **Toggle Breakpoint**.



# Reading the General Purpose Registers



- To read the values stored in the General Purpose Registers (**R0** to **R15**), open the **Registers** tab as indicated below.



# Reading the General Purpose Registers



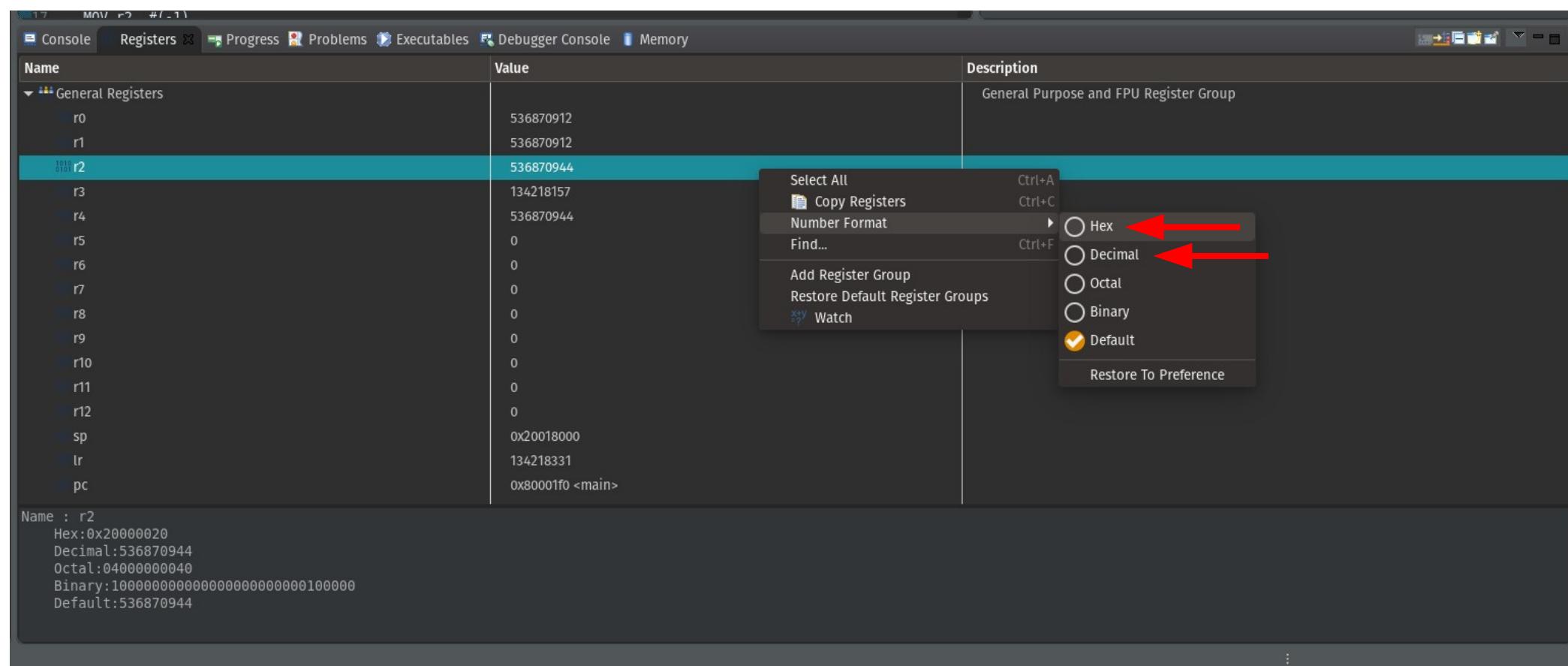
- The image below shows the values of the general purpose registers, which can be used to help you debug your code.
- By **default**, the values are in **decimal format**. You can manually change it to hexadecimal or binary.

Name	Value	Description
General Registers		General Purpose and FPU Register Group
r0	536870912	
r1	536870912	
r2	536870944	
r3	134218157	
r4	536870944	
r5	0	
r6	0	
r7	0	
r8	0	
r9	0	
r10	0	
r11	0	
r12	0	
sp	0x20018000	
lr	134218331	
pc	0x80001f0 <main>	

# Reading the General Purpose Registers



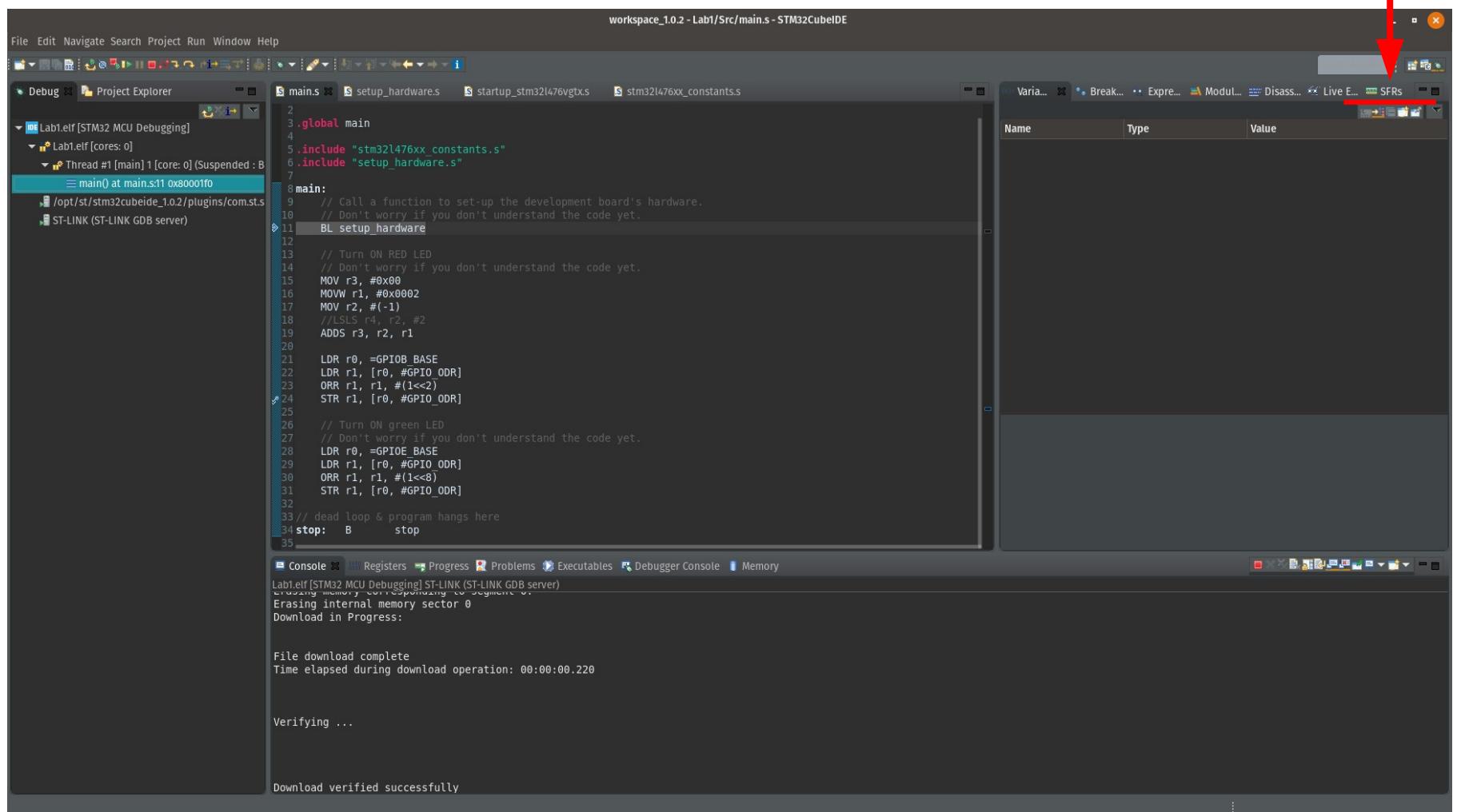
- You can change the format by **right-clicking** on the values and select **Number Format** → **Hex** or **Number Format** → **Binary**.



# Reading the Special Function Registers



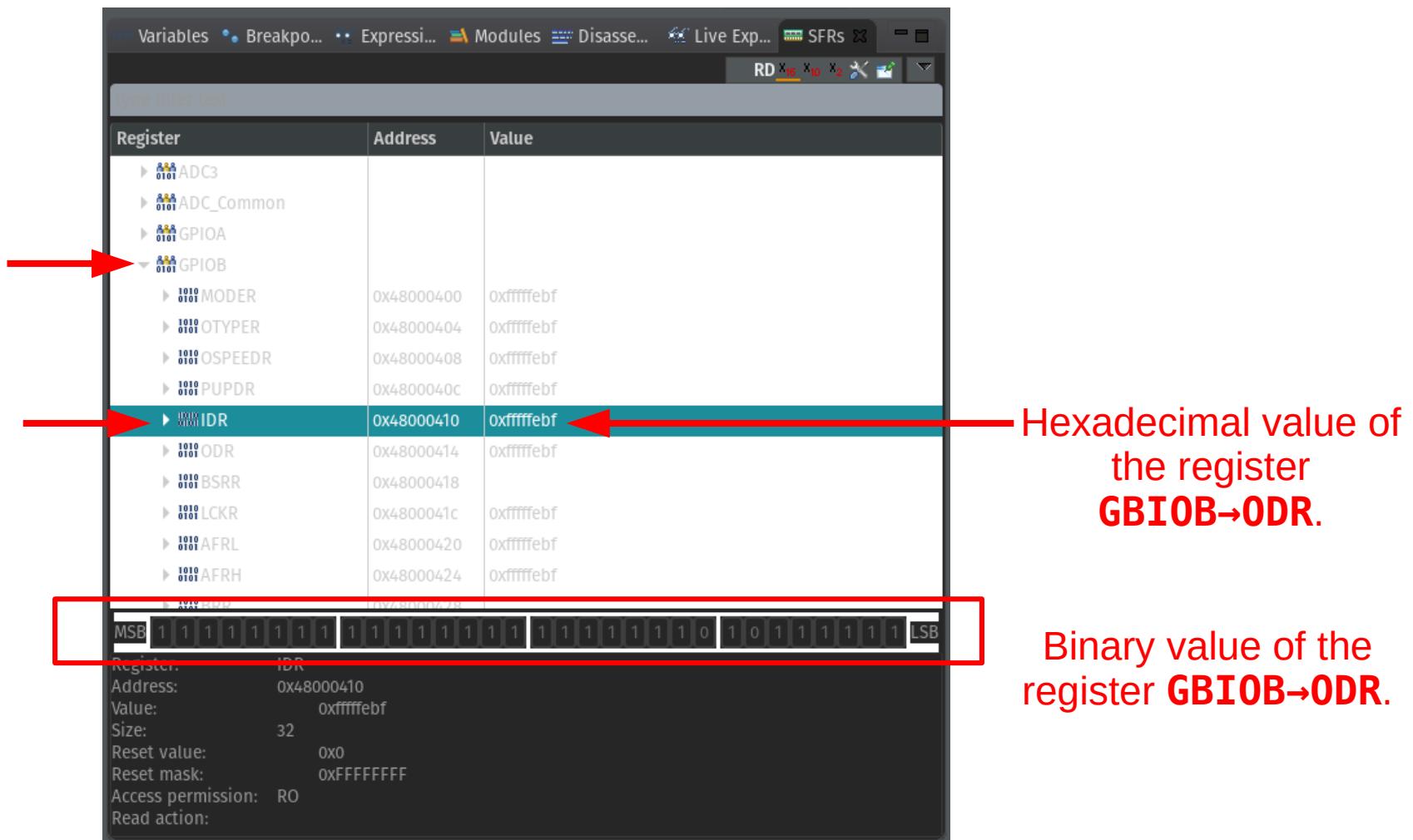
- To read the values stored in the Special Function Registers, such as **GPIOs registers**, open the **SFRs** tab as indicated below.



# Reading the Special Function Registers



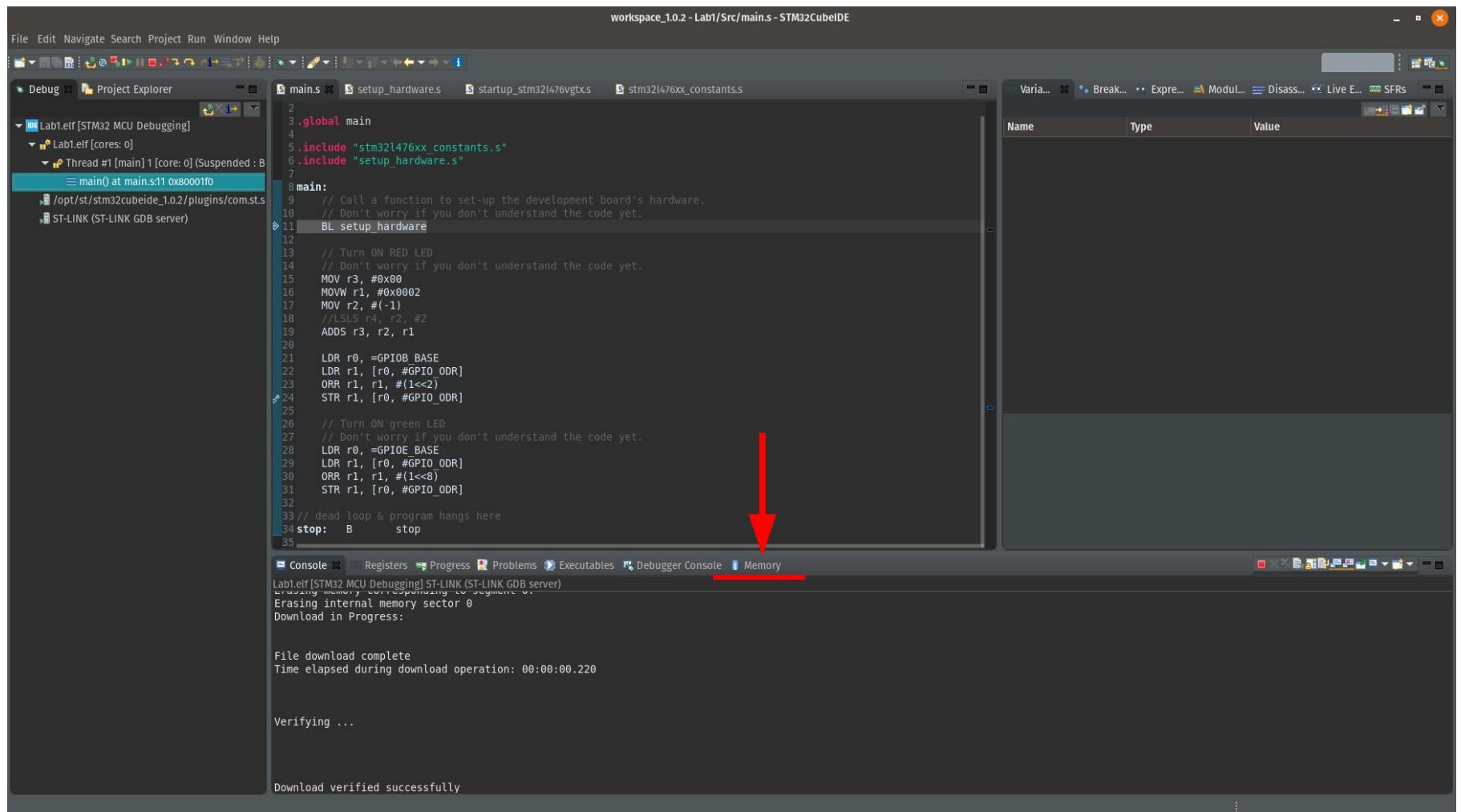
- For example, **GPIOB→IDR** is responsible for reading the inputs connected in GPIO port B.
- It can be read as shown in the picture below.



# Reading the Data Memory

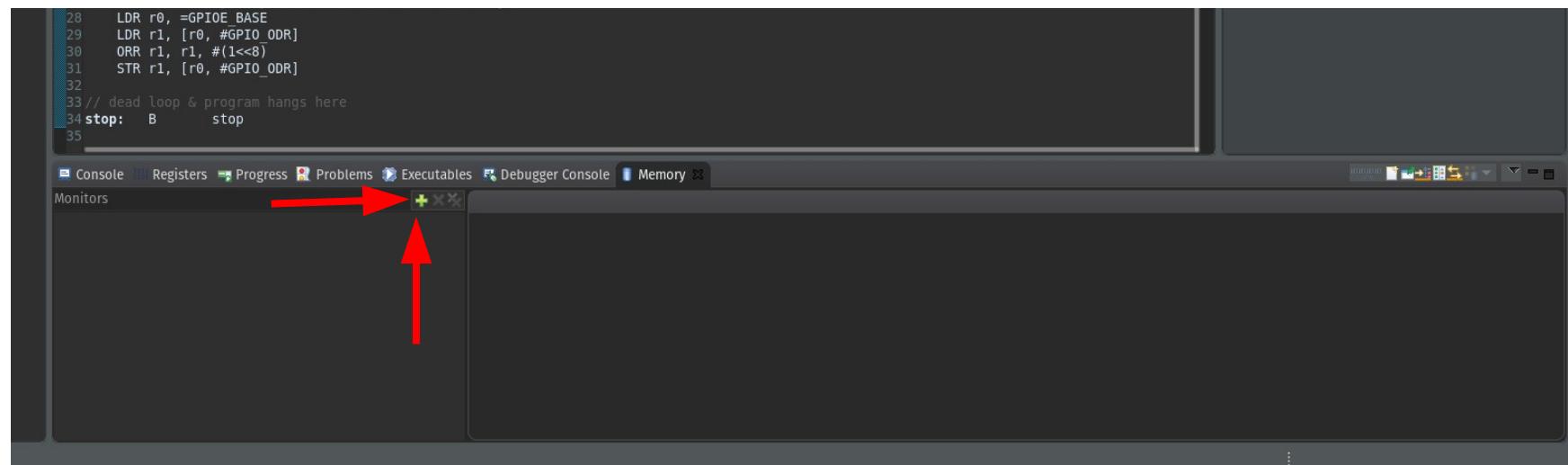


- Data memory can be used to save strings and variables without using the general purpose registers.
- It can be read by clicking in the **Memory** tab.



# Reading the Data Memory

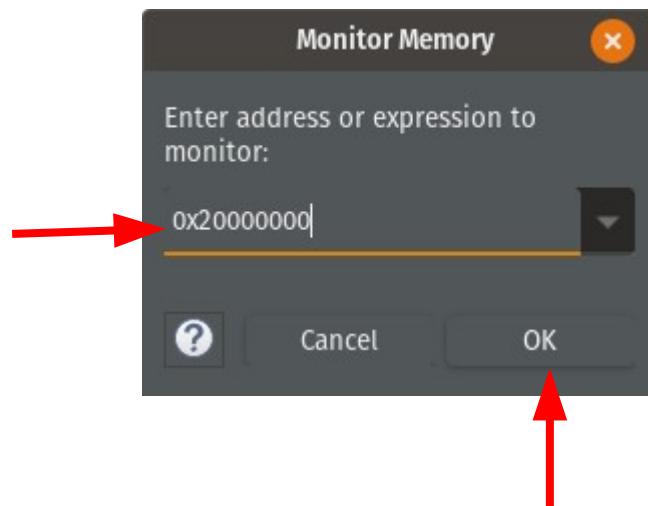
- Data memory starts in the address **0x20000000**.
- You have to **Add a Memory Monitor** to this address.



# Reading the Data Memory



- Data memory starts in the address **0x20000000**.
- You have to **Add a Memory Monitor** to this address.



# Reading the Data Memory



- Data memory starts in the address **0x20000000**.
- You have to **Add a Memory Monitor** to this address.

Contents of the data memory in Hexadecimal.

Address	0 - 3	4 - 7	8 - B	C - F
20000000	00000000	00000000	00000000	00000000
20000010	00000000	00000000	00000000	00000000
20000020	4FF40040	00F061FC	DFF86803	00684004
20000030	03D54FF4	804000F0	58FCDF8	58030068
20000040	800702D5	022000F0	50FCDF8	48030068
20000050	000702D5	082000F0	48FCDF8	38030068
20000060	C00602D5	102000F0	40FCDF8	28030068
20000070	800602D5	202000F0	38FCDF8	18030068