./

Learning Report –

Embedded C

Course Code: <CODE>



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **Approved By** | **Remarks/Revision Details** |
| 1 | 30th Sept, 2020 | Milind Mohapatra |  | To be approved by Dr. Vivek Kaundal, Bhargav N |  |
| 2 | 5th Oct, 2020 | Milind Mohapatra |  | To be approved by Dr. Vivek Kaundal, Bhargav N |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Document History**

CONTENTS

[TABLE OF FIGURES 4](#_Toc52839032)

[Activity 1: Linker Script 5](#_Toc52839033)

[Activity 2: Semi Hosting 6](#_Toc52839034)

[Activity 3: Changing bit state of SPI1 SPE 7](#_Toc52839035)

[Activity 4: Debugging techniques 8](#_Toc52839036)

[Serial wire viewer and data tracing 8](#_Toc52839037)

[Single stepping, stepping over and stepping out 8](#_Toc52839038)

[Breakpoints 9](#_Toc52839039)

[Call stack (Static stack analyzer) 9](#_Toc52839040)

[Expression and Variable window 10](#_Toc52839041)

[Memory browser 11](#_Toc52839042)

[Data watch points 11](#_Toc52839043)

[Disassembly 12](#_Toc52839044)

[Activity 5: MCU specific header file 13](#_Toc52839045)

[Link to repository: Link 13](#_Toc52839046)

[Activity 6- Code Quality MISRA C Standards 13](#_Toc52839047)

[6.1. Reliability 13](#_Toc52839048)

[6.2 Maintainability 14](#_Toc52839049)

[6.3 Efficiency 15](#_Toc52839050)

[6.4 Portability 15](#_Toc52839051)

# TABLE OF FIGURES

[Figure 1: Linker section 5](#_Toc52839018)

[Figure 2: Code Snippet for semi hosting 6](#_Toc52839019)

[Figure 3: Semi hosting enabled 6](#_Toc52839020)

[Figure 4: Bit modification of SPI1 SSI 7](#_Toc52839021)

[Figure 5: Bit modification of SPI1 SPE 7](#_Toc52839022)

[Figure 6: Enable serial wire viewer 8](#_Toc52839023)

[Figure 7: Breakpoint stepping 8](#_Toc52839024)

[Figure 8: Multiple breakpoints 9](#_Toc52839025)

[Figure 9: Static stack analyzer 9](#_Toc52839026)

[Figure 10: Variable window 10](#_Toc52839027)

[Figure 11: Expression window 10](#_Toc52839028)

[Figure 12: Memory browser 11](#_Toc52839029)

[Figure 13: Assigning watch-point 11](#_Toc52839030)

[Figure 14: Disassembly window 12](#_Toc52839031)

# Activity 1: Linker Script

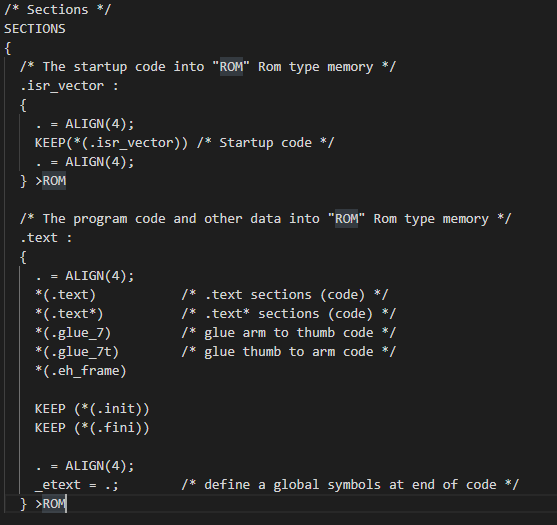


Figure 1: Linker section

# Activity 2: Semi Hosting

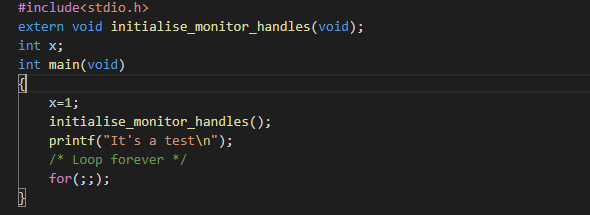


Figure 2: Code Snippet for semi hosting

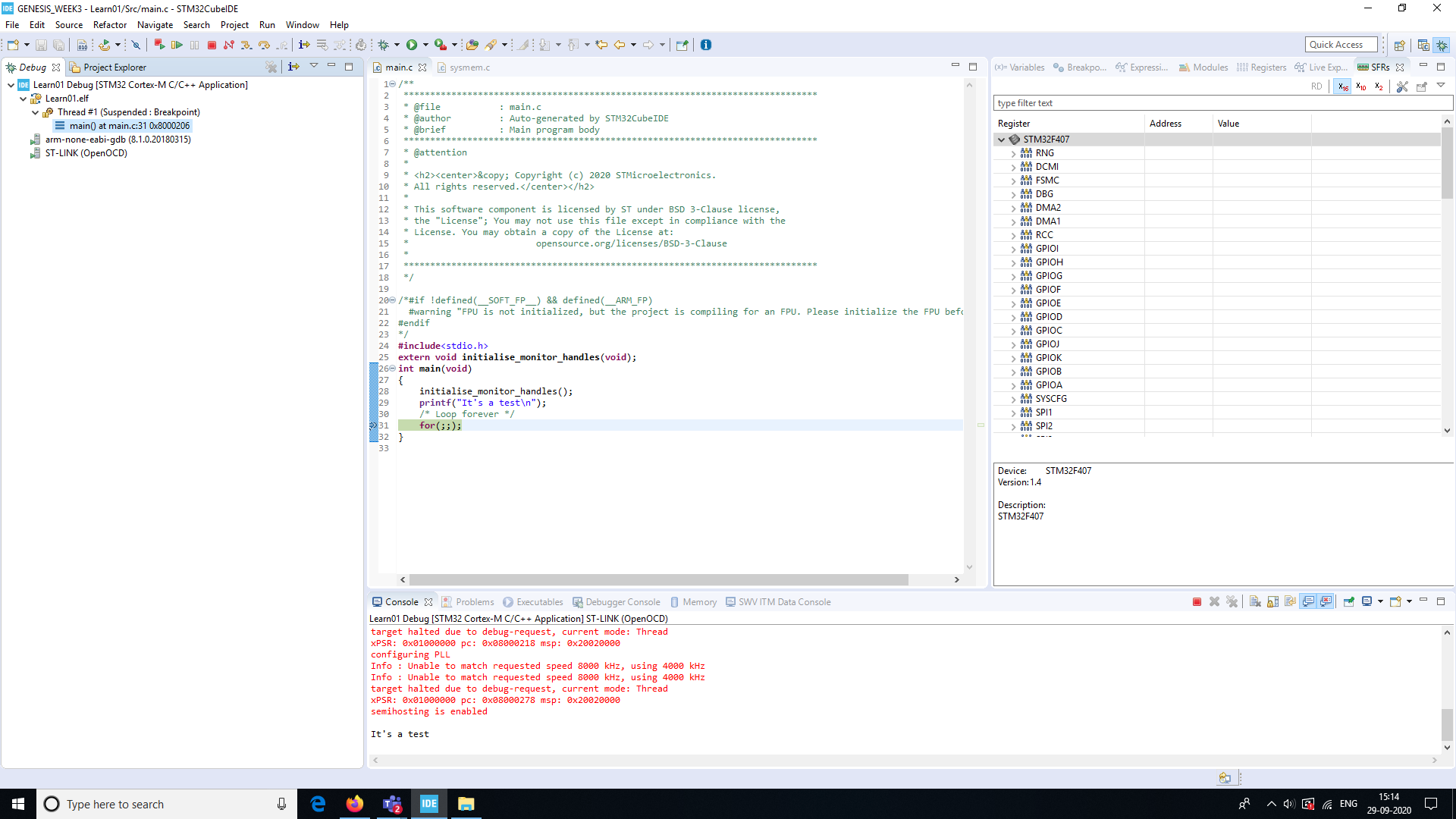


Figure 3: Semi hosting enabled

# Activity 3: Changing bit state of SPI1 SPE

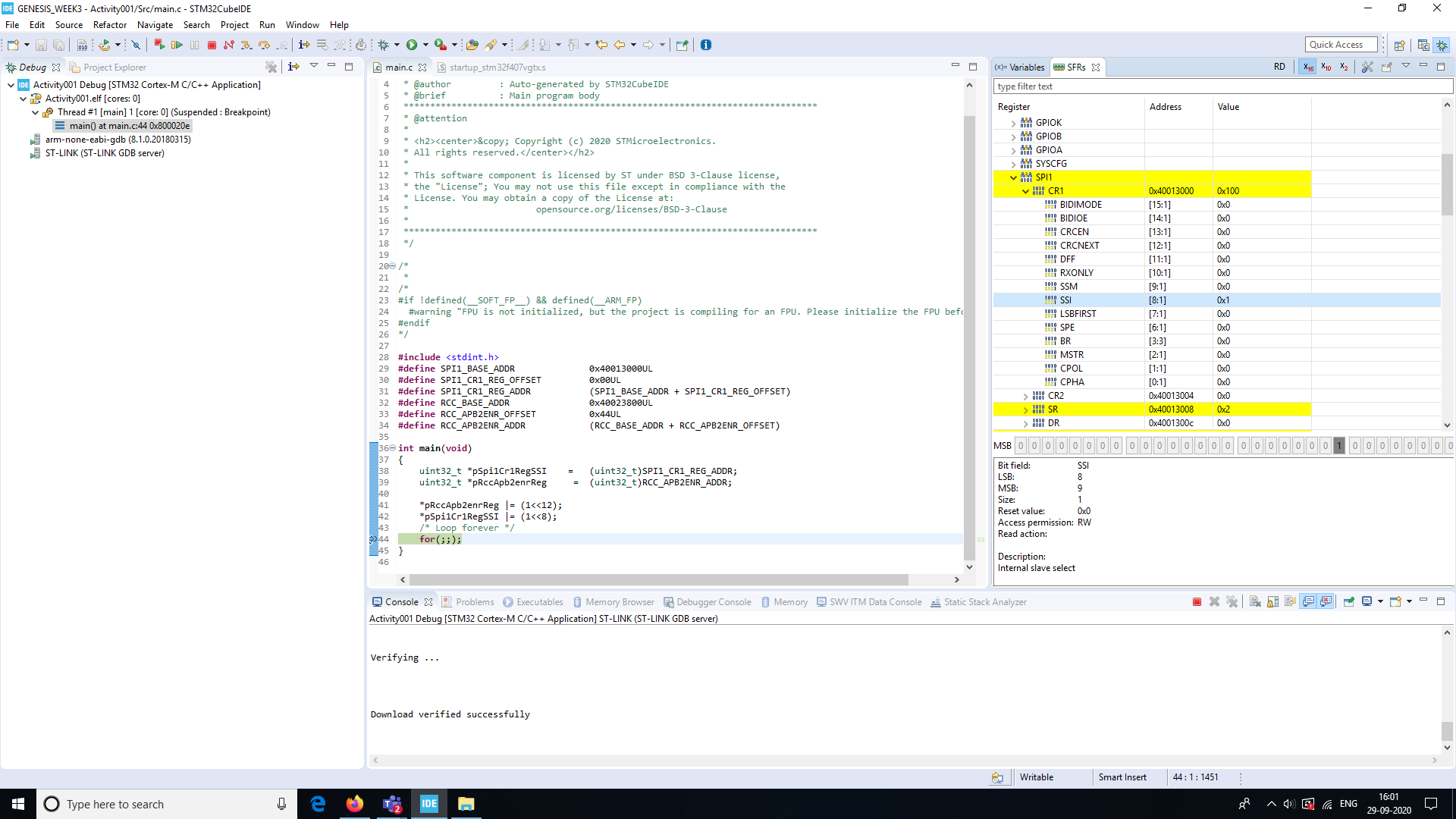


Figure 4: Bit modification of SPI1 SSI

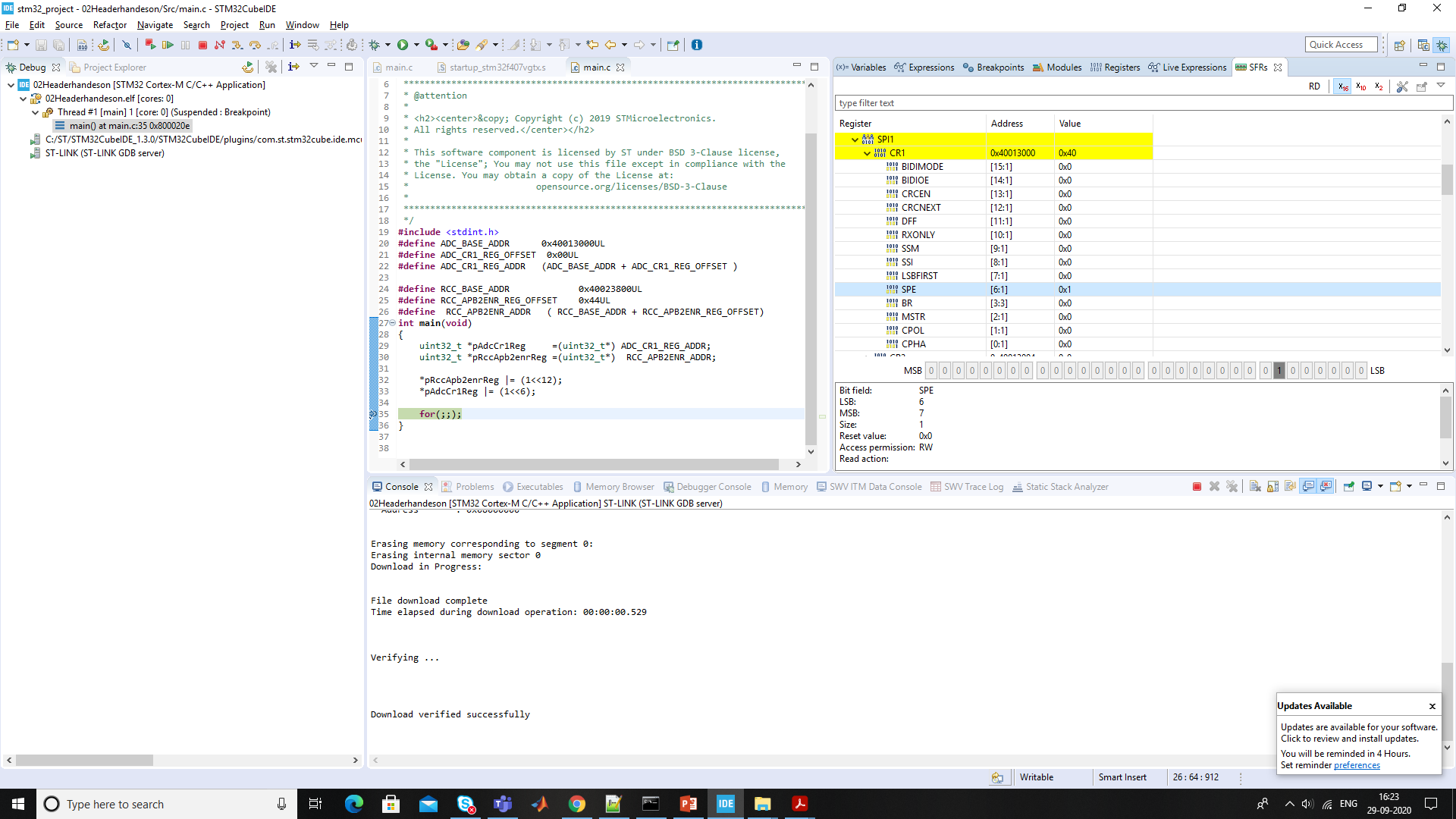


Figure 5: Bit modification of SPI1 SPE

# Activity 4: Debugging techniques

## Serial wire viewer and data tracing

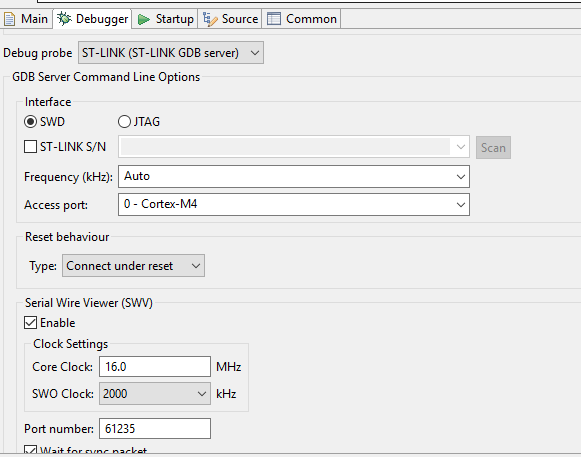


Figure 6: Enable serial wire viewer

## Single stepping, stepping over and stepping out

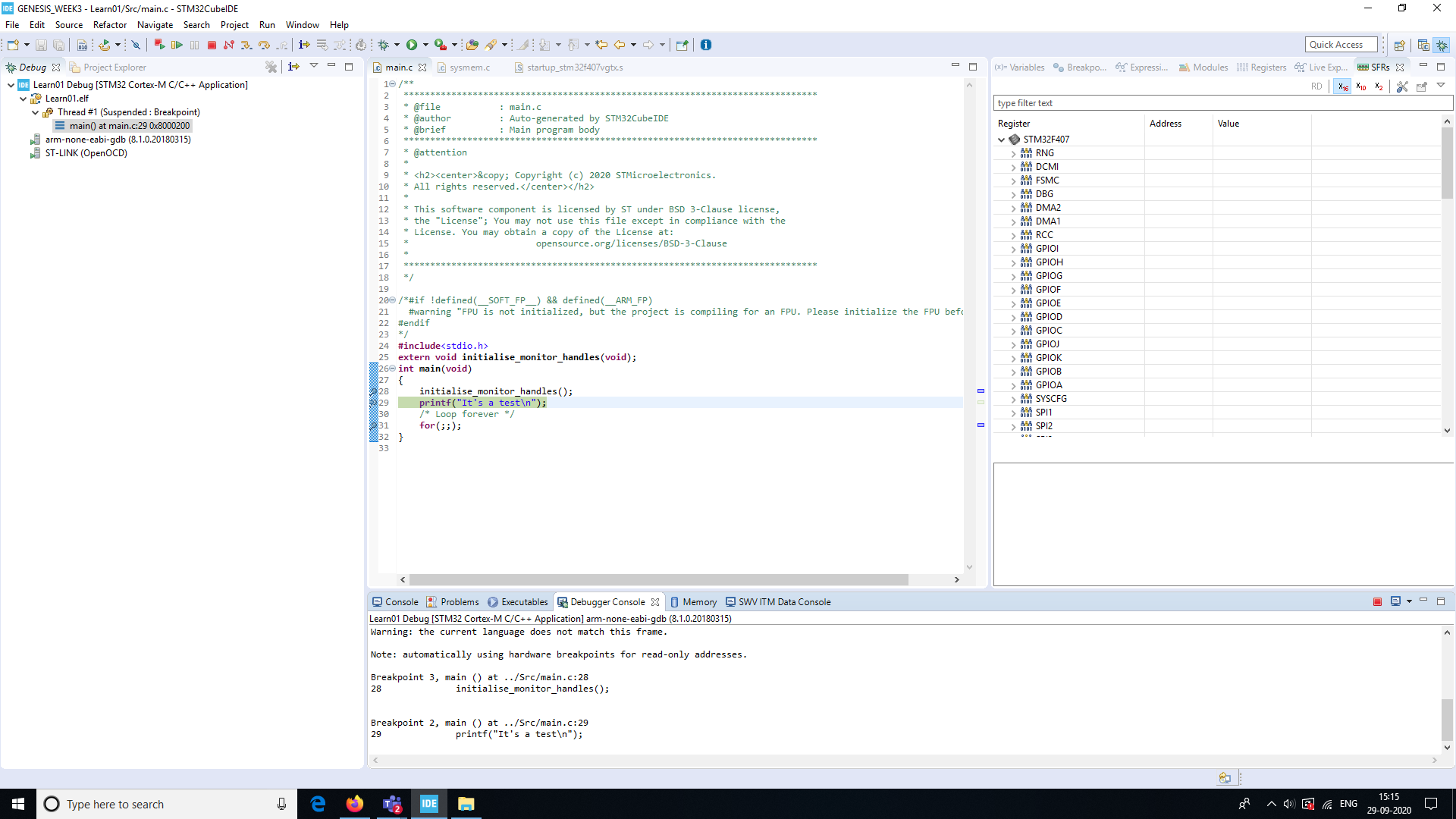


Figure 7: Breakpoint stepping

## Breakpoints

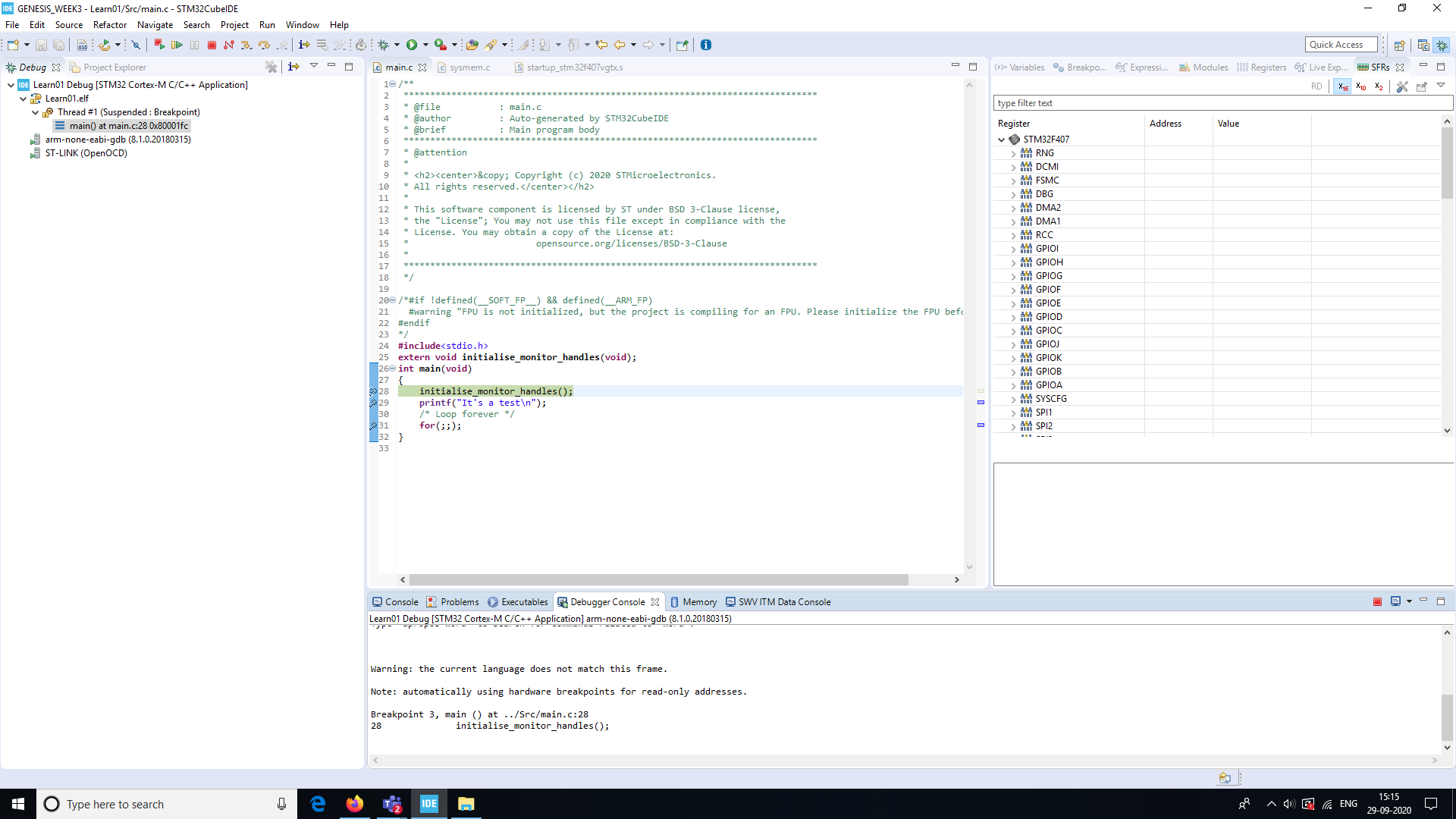


Figure 8: Multiple breakpoints

## Call stack (Static stack analyzer)

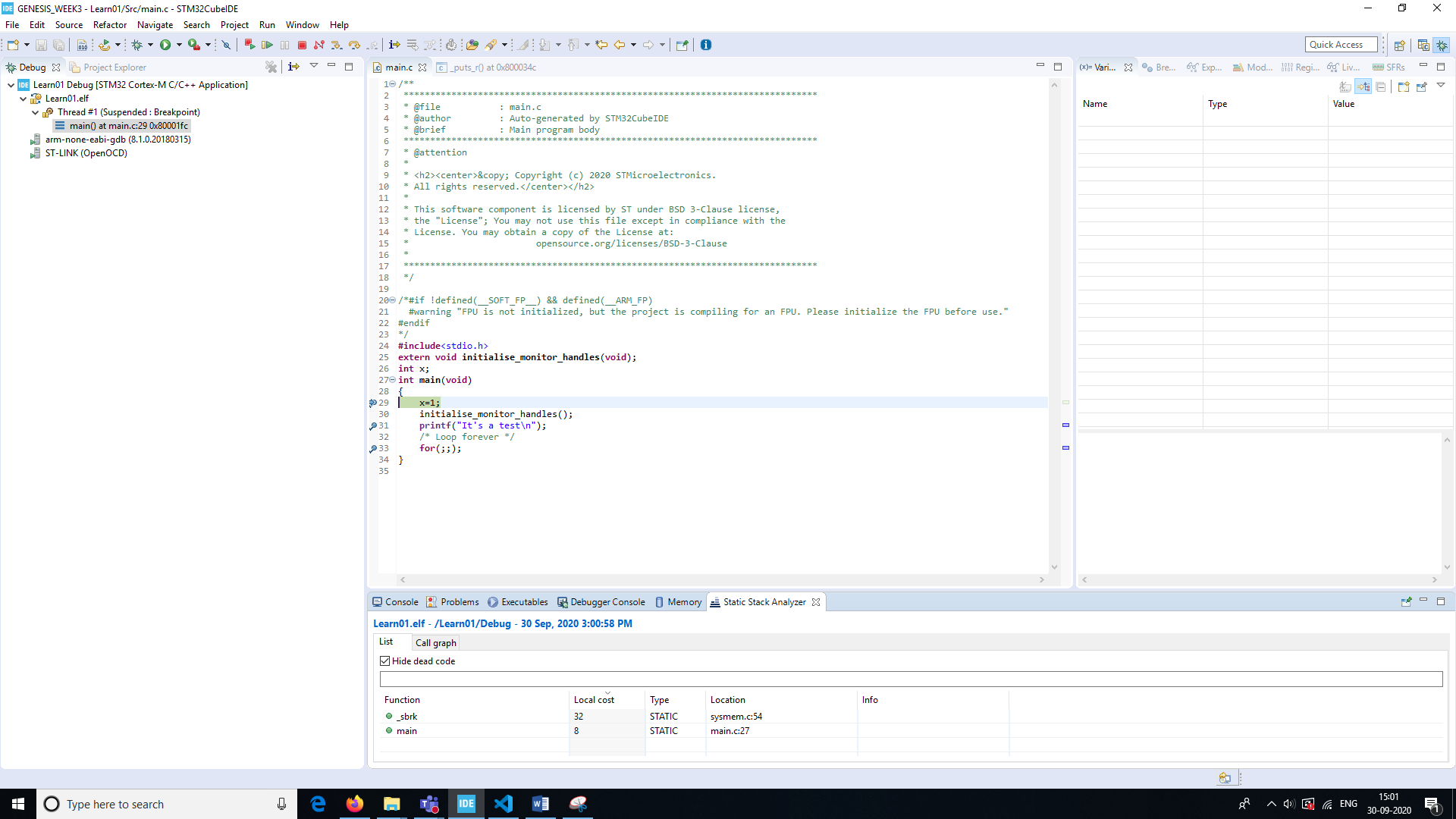


Figure 9: Static stack analyzer

## Expression and Variable window

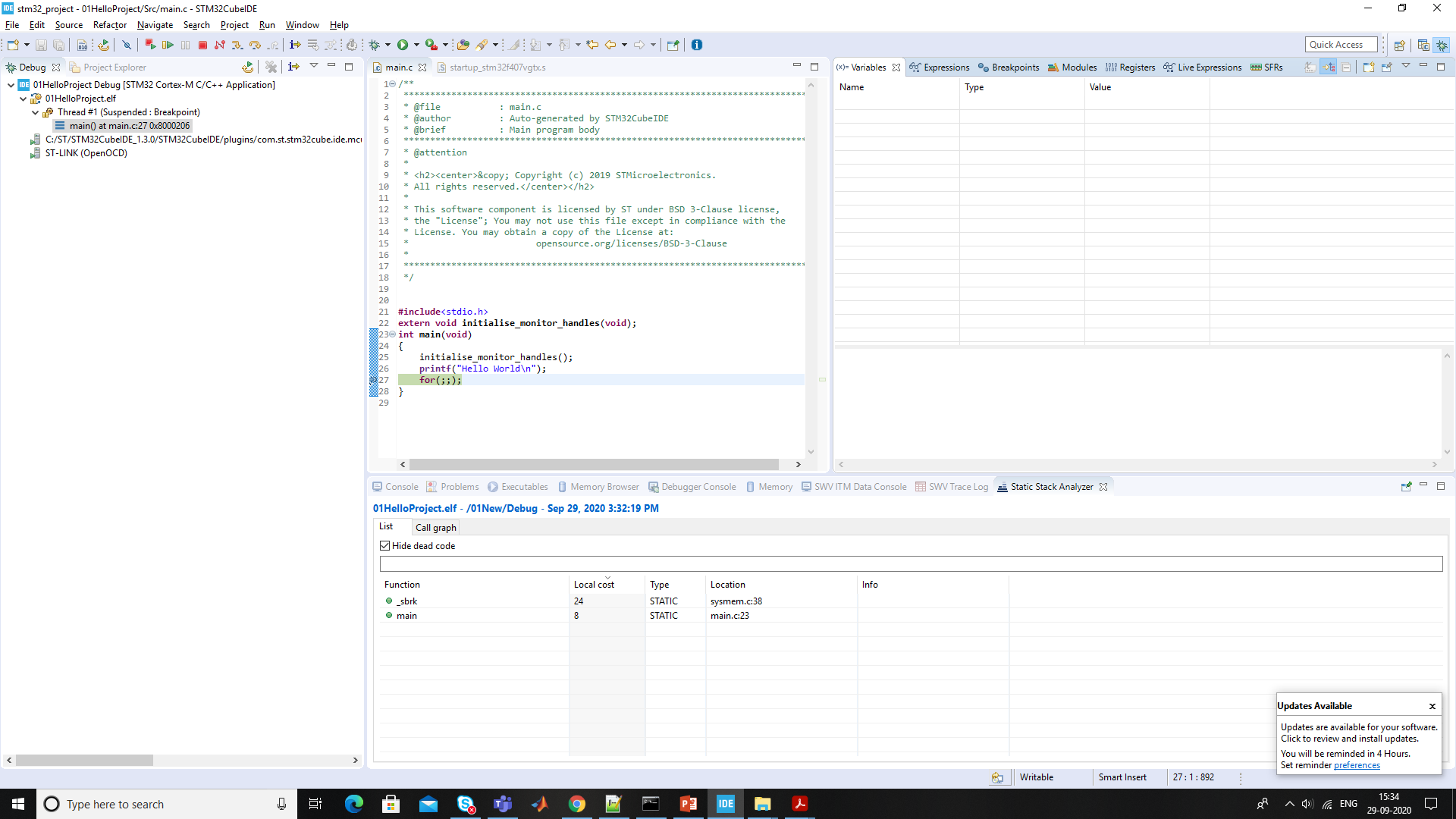


Figure 10: Variable window

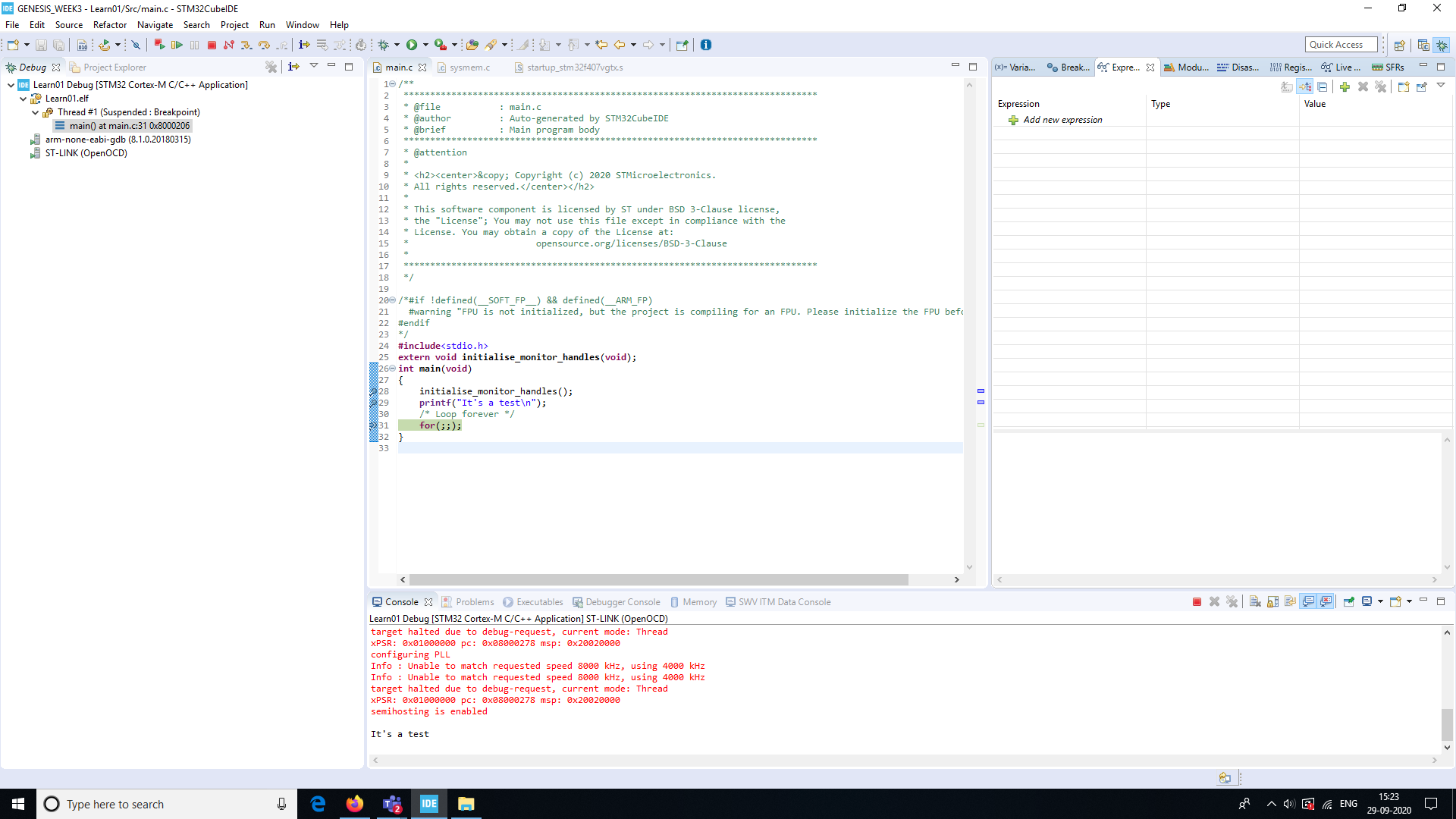


Figure 11: Expression window

## Memory browser

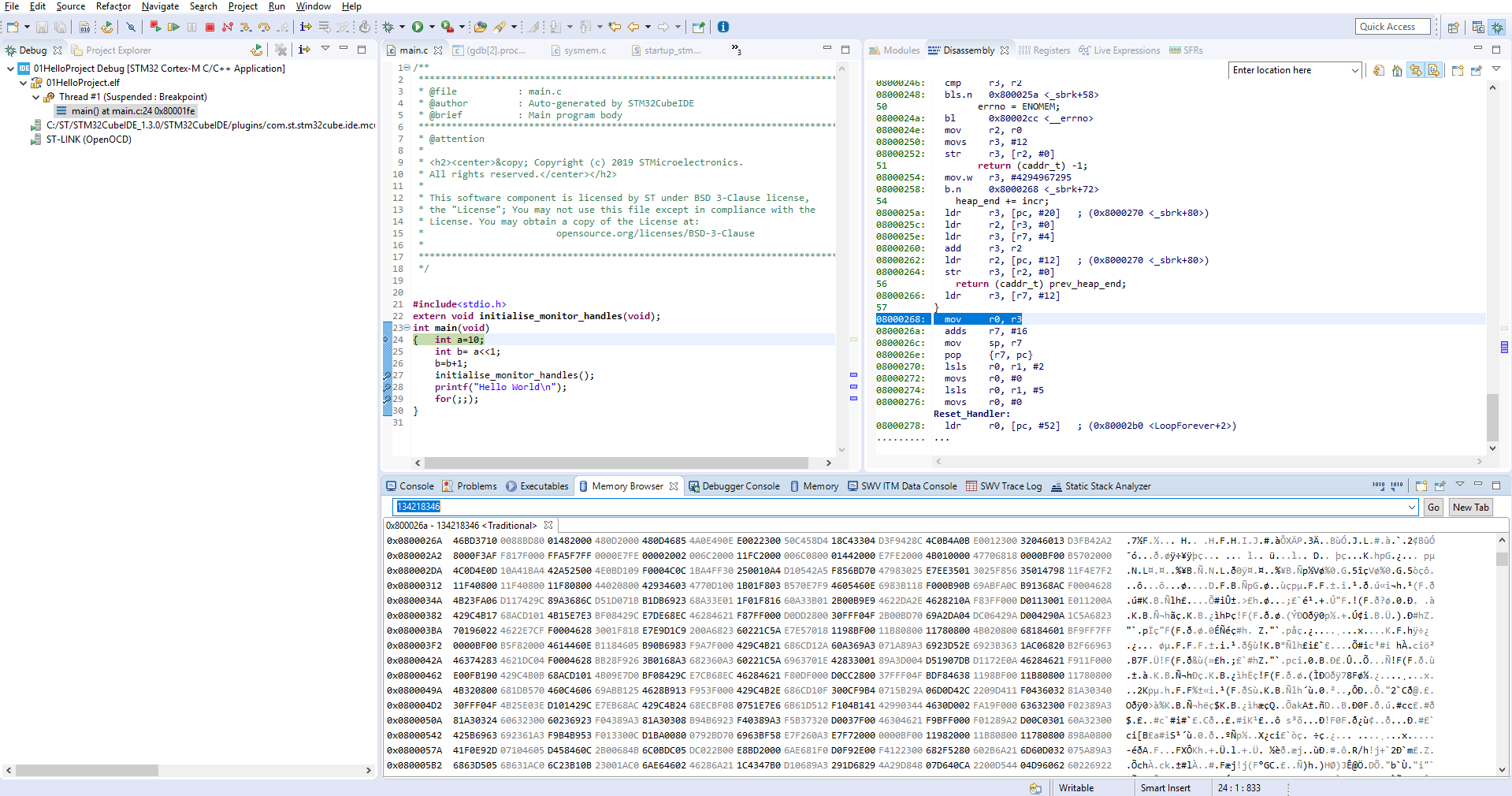


Figure 12: Memory browser

## Data watch points

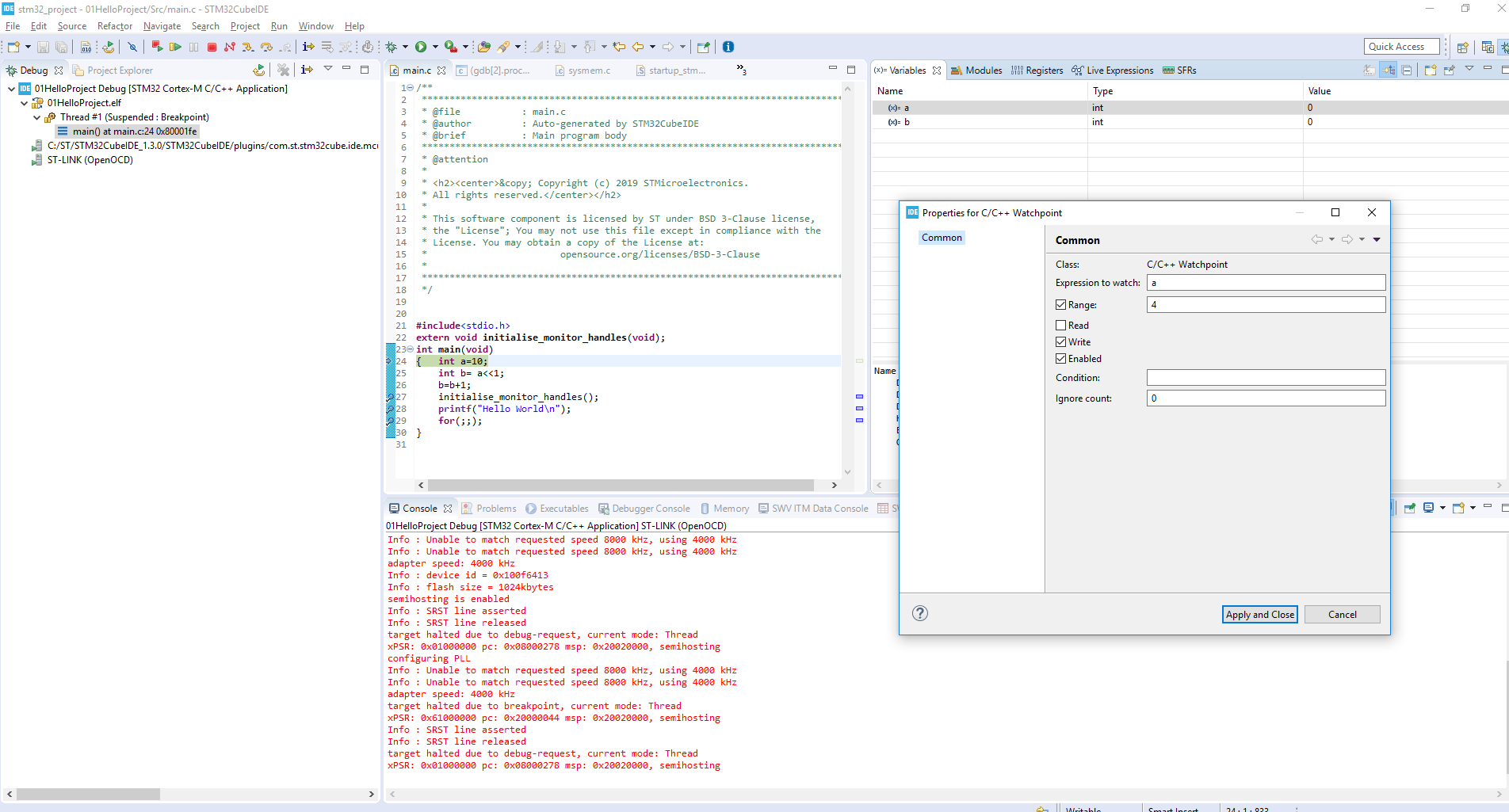


Figure 13: Assigning watch-point

## Disassembly

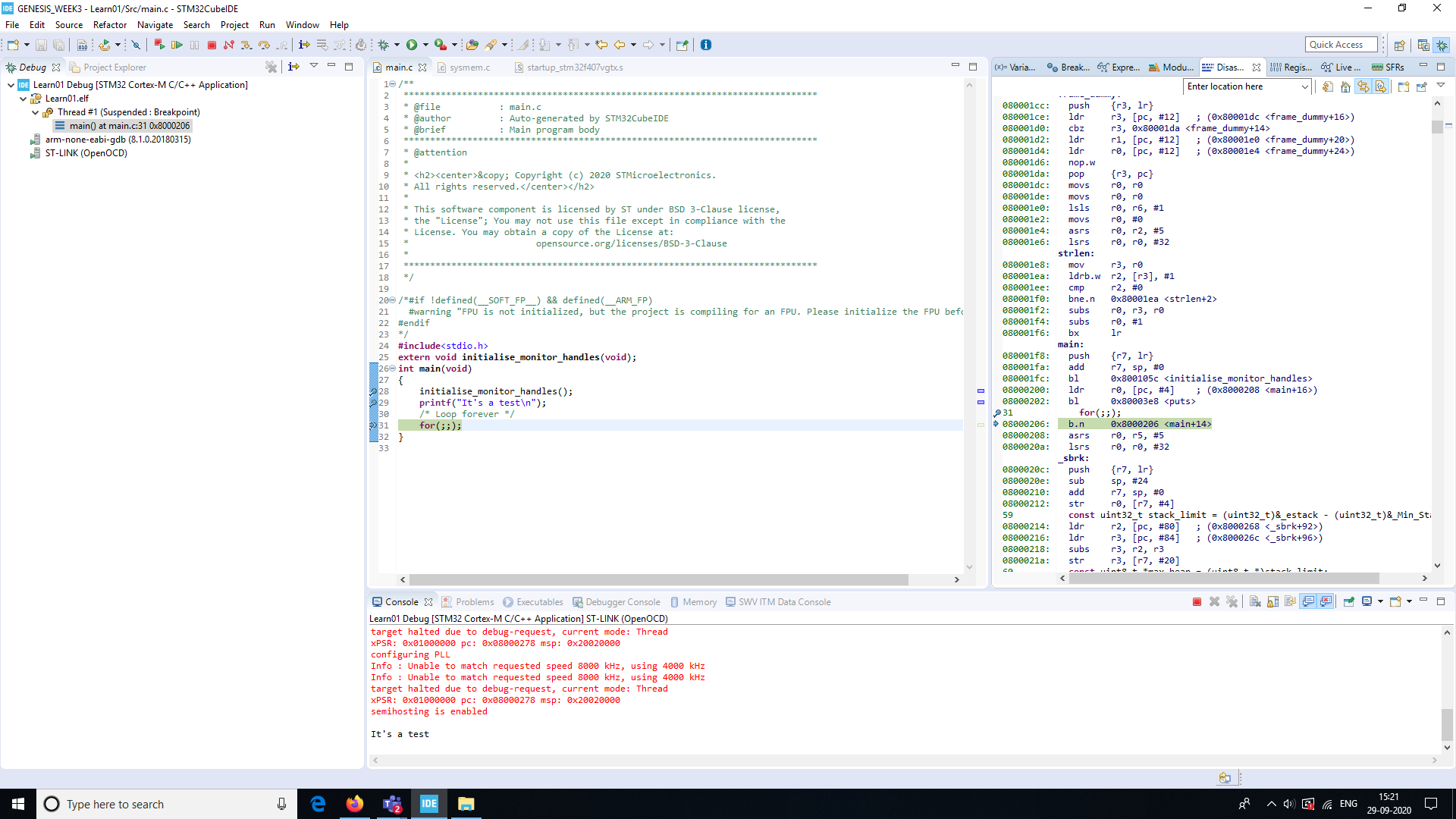


Figure 14: Disassembly window

# Activity 5: MCU specific header file

## Link to repository: [Link](https://github.com/99002670/week3-embedded/tree/master/Driver)

# Activity 6- Code Quality MISRA C Standards

## 6.1. Reliability

* Initialize areas and use them by taking their sizes into consideration.
  + - Use areas after initializing them.
    - Describe initializations without excess or deficiency
    - Pay attention to the range of the area pointed by a pointer.
* Use data by taking their ranges, sizes and internal representations into consideration.
  + - Make comparisons that do not depend on internal representations.
    - When values such as logical values are defined as a range, do not make a judgment by finding whether a value is equivalent to any value (representative value that is implemented) within this range
    - Use the same data type to perform operations or comparisons.
    - Describe code by taking operation precision into consideration.
    - Do not use operations that have the risk of information loss.
    - Use types that can represent the target data.
    - Pay attention to pointer types.
    - Write in a way that will enable the compiler to check that there are no conflicting declarations, usages and definitions.
* Write in a way that ensures intended behavior
  + - Prevent operations that may cause runtime error from falling into error cases.
    - Check the interface restrictions when a function is called.
    - Do not perform recursive calls.
    - Pay attention to branch conditions and describe how to handle cases that do not follow the predefined conditions when they occur.
    - Pay attention to the order of evaluation.
    - Be careful with how to access the shared data in programs that use threads or signals.

## 6.2 Maintainability

* Keep in mind that others will read the program.
  + - Do not leave unused descriptions.
    - Do not writing confusingly.
    - Do not write in an unconventional style.
    - Write in a style that clearly specifies the operator precedence.
    - Explicitly describe the operations that are likely to cause misunderstanding when they are omitted.
    - Use one area for one purpose.
    - Do not reuse names.
    - Do not use language specifications that are likely to cause misunderstanding.
    - When writing in an unconventional style, explicitly state its intention.
    - Do not embed magic numbers.
    - Explicitly state the area attributes.
    - Correctly describe the statements even if they are not compiled.
* Write in a style that can prevent modification errors.
  + - Clarify the grouping of structured data and blocks.
    - Localize access ranges and related data.
* Write programs simply.
  + - Do structured programming.
    - Limit the number of side effects per statement to one.
    - Write expressions that differ in purpose separately.
    - Do not use complicated pointer operations.
* Write in a unified style.
  + - Unify the coding styles.
    - Unify the style of writing comments.
    - Unify the naming conventions.
    - Unify the contents to be described in a file and the order of describing them.
    - Unify the style of writing declarations.
    - Unify the style of writing null pointers.
    - Unify the style of writing pre-processor directives.
* Write in a style that makes testing easy.
  + - Write in a style that makes it easy to investigate the causes of problems when they occur.
    - Be careful when using dynamic memory allocations.

## 6.3 Efficiency

* + Write in a style that takes account of resource and time efficiencies.

## 6.4 Portability

* Write in a style that is not dependent on the compiler
  + - Do not use functionalities that are advanced features or implementation-defined
    - Use only the characters and escape sequences defined in the language standard.
    - Confirm and document data type representations behavioral specifications of advanced functionalities and implementation- dependent parts.
    - For source file inclusion confirm the implementation dependent parts and write in a style that is not implementation- dependent.
    - Write in a style that does not depend on the environment used for compiling.
* Localize the code that has a problem with portability.