tinyMicon MatisseCORE™

matiseye™-studio User's Guide

How to build and operate a C development environment for Matisse using Visual Studio Code

© 2024 ROHM Co., Ltd.

No. 67UG023E Rev.007
May 2026

Revision History

Date	Version	Description
2024/02/22	Rev.001	Initial publication
2024/05/10	Rev.002	Adding Stack Static Analysis (Stack View)
2024/07/24	Rev.003	Modified VS Code installation description
2025/03/24	Rev.004	3.2. Changed the description of the debug clock speed in the status bar 4.2. Added several configuration items 6.8. Added the method to start Memory View when not debugging 6.9. Added the method to start Peripheral View when not debugging 6.10. Added a new startup method for Performance View
2025/08/28	Rev.005	1.2. Updated the file list to separate the consumer and automotive development environments 2.1. Added a description of the updater tool for automotive development environment
2025/11/11	Rev.006	7.7. Updated to enable copying Memory View values in table format. 7.9. Added a feature to DISASSEMBLY that displays the execution results for the entire executable file. 7.7 7.8 7.9. Added Note.
2026/05/15	Rev.007	1.3 Change the description of ".NET Framework" in the operating environment 5 Add tool launcher (matiseye-studio Sidebar View) 6.4 Add build result analysis (Build Analysis View)

© 2024 ROHM Co., Ltd.

Table of Contents

1. Introduction 5
1.1. Overview 5
1.2. File List 5
1.3. Operating Environment 5
1.4. Compiler Restrictions 5
2. Installation Procedure 6
2.1. Installing C Programming Tools for Matisse 6
2.2. Installing Visual Studio Code (VS Code) 7
2.3. Installing cpptools on VS Code 8
2.4. Installing matiseye™-studio on VS Code 9
3. How to use Visual Studio Code (VS Code) 11
3.1. VS Code Window Description 11
3.2. Status Bar Description 12
3.3. Operating VS Code 13
4. Project Configuration and Setting Items 14
4.1. Configuration of the Samples Project 14
4.2. Compiler/Debugger Setting Items（settings.json） 15
5. Tool Launcher (matiseye-studio Sidebar View) 17
6. How to Build 18
6.1. Running Build Task and Generated Files 18
6.2. MAP File 19
6.3. Stack Static Analysis 20
6.4. Build Analysis (Build Analysis View) 21
6.4.1. Summary Display 21
6.4.2. Detailed Analysis View 22
7. Debug 25
7.1. Debug Window Descriptions 25
7.2. Debug Menu 27
7.3. Breakpoints 28
7.4. Debug Toolbar 29
7.5. Data Inspection 30
7.6. Call stack 31
7.7. MEMORY 32
7.8. PERIPHERALS 33
7.9. DISASSEMBLY 35
7.10. PERFORMANCE 36
7.11. Debug Related Keyboard Shortcuts 37
7.12. Memory Window 38
7.13. Peripheral Window 39
7.14. Function call history on CPU resetting 40
8. Command Execution 41
8.1. Displaying the Command Palette and Command Input 41
9. Frequently asked questions 42
10. Shortcut Key List 44

© 2024 ROHM Co., Ltd.

10.1. General 44
10.2. File Management....45
10.3. Editor Management....46
10.4. Editing 47
10.5. Search and Replace....48
10.6. Rich Language Editing 49
10.7. Display 50
10.8. Debug....51
11. Open-source software licenses....52
12. Trademark notices 52

© 2024 ROHM Co., Ltd.

1. Introduction

1.1. Overview

matiseye™-studio is a C development environment for Matisse. This tool is based on Visual Studio Code.

NOTES

Visual Studio Code (VS Code) is a powerful and lightweight OSS code editor developed by Microsoft.

1.2. File List

Table 1. matiseye™-studio Consumer Software Development Environment-File List

Name	Description
matisse-dev-env-installer-3.*.*.exe	Installer for C language development environment for Matisse for consumer development.
matiseye-studio-*.*.*.vsix	Extension files that enable VS Code C programming features to be used for Matisse.
sample_project.zip	A sample C language development project.

Table 2. matiseye™-studio Automotive Software Development Environment-File List

Name	Description
MatisseCCompiler-2.*.*.exe	Installer for C language development environment for Matisse for automotive development.
matisse-automotive-dev-env-updater_ *.*.*.exe	Updater for C language development environment for Matisse for automotive development. It is intended to update peripheral tools, excluding the compiler.
matiseye-studio-*.*.*.vsix	Extension files that enable VS Code C programming features to be used for Matisse.
sample_project.zip	A sample C language development project.

1.3. Operating Environment

Table 3. matiseye™-studio Operating Environment

Environment	Description
OS	Windows 11
CPU	Comparable performance to Intel Core line
Memory	2GByte or more
HDD	1GByte or more free space
.Net Framework	Version 4.7.2(preinstalled on Windows 10 version 1803 and later)

1.4. Compiler Restrictions

The current version has the following limitations:

[Matisse Settings]

- MUL instruction: selectable for hardware (default without MUL for both hardware and IDE)
- Number of general-purpose registers: fixed to 16

[Others]

- 64-bit integer type (long long) and floating point number types (float and double) are not supported.
- Variable-length arguments and variable-length arrays are not supported.
- Dynamic memory allocation is not supported.

© 2024 ROHM Co., Ltd.

2. Installation Procedure

Notes

If you are using an environment earlier than Windows 10 (version 1803), and .NET Framework 4.7.2 is not installed, please install it in advance.

2.1. Installing C Programming Tools for Matisse

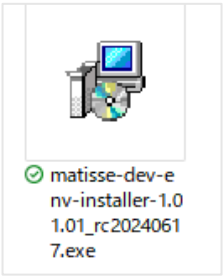


Figure 1. Run matisse-dev-env-installer

1 When using the consumer software development environment, double-click on "matisse-dev-env-installer- 3.*.*.exe" to run the installer.
When using the automotive software development environment, double-click on "MatisseCCompiler- 2.*.*.exe" to run the installer.

NOTES

If User Account Control appears, click Yes.

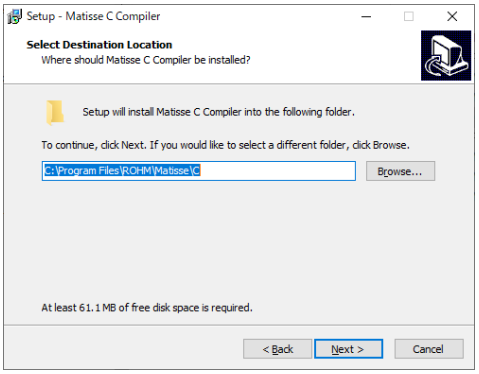


Figure 2. Matisse Develop Environment Installation Window

2 Follow the on-screen instructions to install the software.

NOTES

If you changed the installation location in "Select Destination Location" from the default setting, you need to specify the path to the c programming tools in the setting file.

See "Project Configuration and Setting Items".

3 During the installation of the C language development environment, the following installers will also be executed automatically.
- Microsoft Visual Studio Redistributable package
- MtProxy (Proxy Server for Debug Board Communication)
- MtChecker (Development Environment Configuration Checker)
- mtloader (Program Downloader)

NOTES

If User Account Control appears, click Yes.

4 When using the automotive software development environment, run "matisse-automotive-dev-env-updater_*.*.*.exe".

5 Restart your PC

© 2024 ROHM Co., Ltd.

2.2. Installing Visual Studio Code (VS Code)

NOTES

Please download the latest installer of VS Code from the website (https://code.visualstudio.com).

If the version of VS Code is outdated, the features of matiseye™-studio may not function properly, so please install the latest version even if you already have it installed.

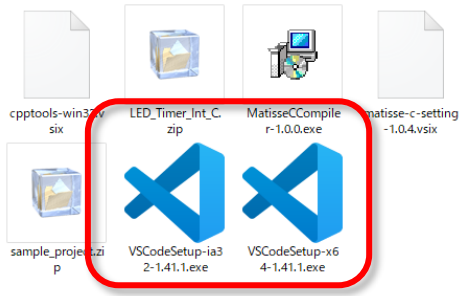


Figure 3. Run VS Code Installer

1 Double-click VS Code installer to launch VS Code installer.

NOTES

The installer is available for 32-bit and 64-bit.
For 32-bit Windows: VSCodeSetup-ia32-*.*.*.exe
For 64-bit Windows: VSCodeSetup-x64-*.*.*.exe

NOTES

If User Account Control appears, click "Yes".

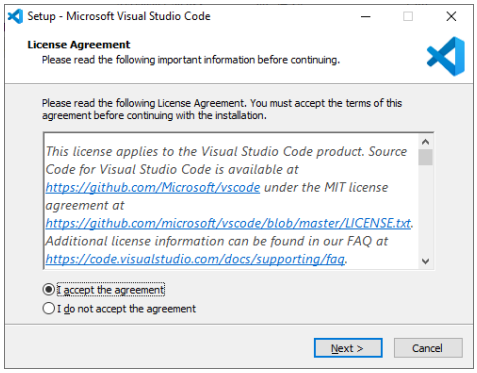


Figure 4. VS Code Installation

2 Follow the on-screen instructions to install the software.

2.3. Installing cpptools on VS Code

NOTES

・ Cpptools is an extension file that adds the ability to develop C programs to VS Code.

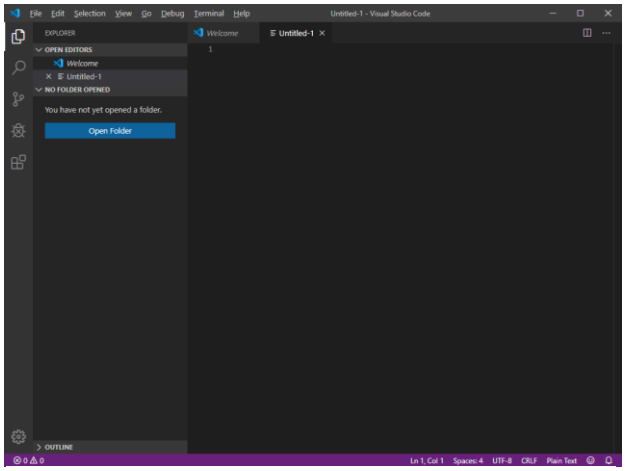


Figure 5. Start VS Code

1 Start VS Code.

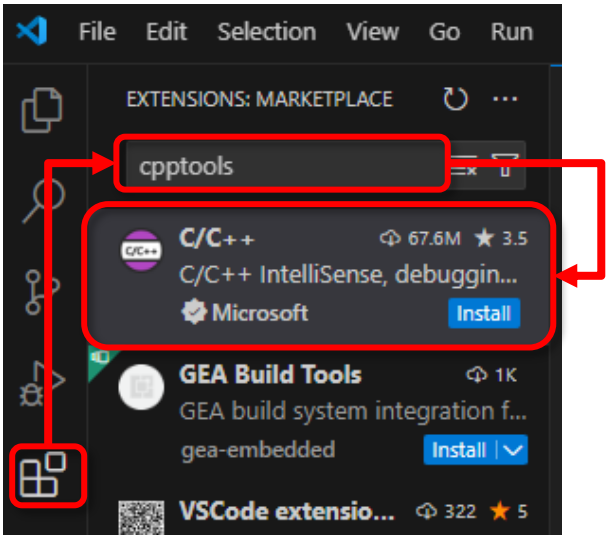


Figure 6. Search cpptools

2 Click [ ] (Extensions) > Enter "cpptools" in the search window and select "C/C++".

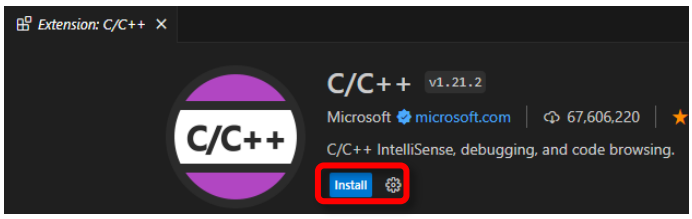


Figure 7. Install cpptools

3 Click "Install" to complete the installation.

4 Restart VS Code.

4. VS Code を再起動します

2.4. Installing matiseye™-studio on VS Code

NOTES

Installing the matiseye™-studio extension file adds to VS Code the features required to develop C language programs for Matisse.
If you have a previous version of the extension file (matise-c-setting, mt-studio) installed, please uninstall it before performing the following steps.

Notes

If other VS Code extensions for C/C++ development (CMake, PlatformIO, etc.) are installed, functionality may conflict. Please disable them when using matiseye™-studio.

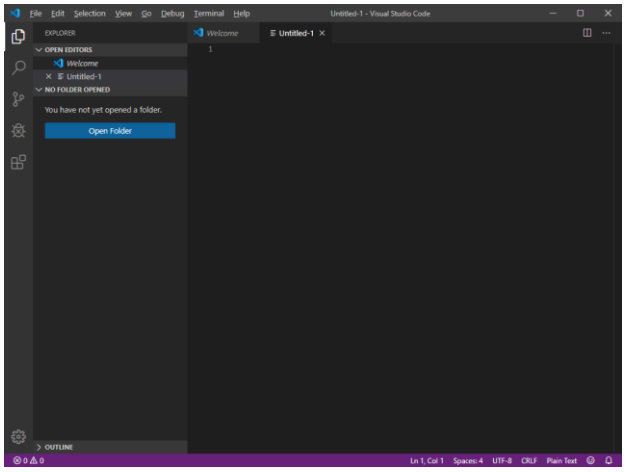


Figure 8. Start VS Code

1 Start VS Code.

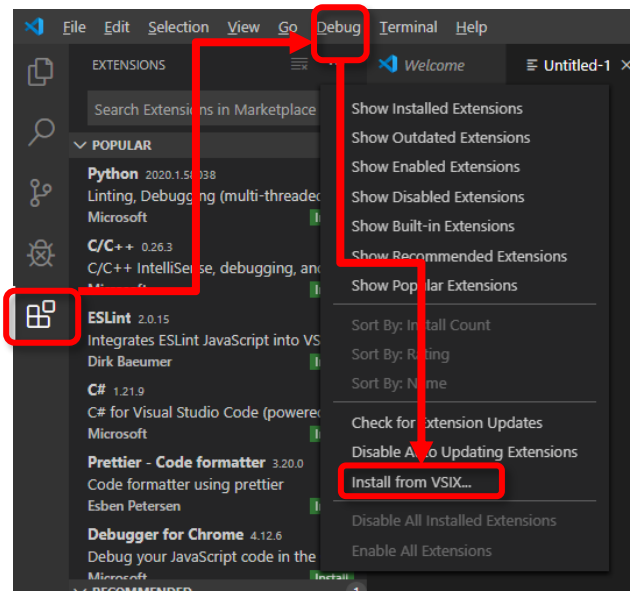


Figure 9. Click Install from VSIX

2 Click [ ] (Extensions) > [...] (More Actions) Click > [Install from VSIX].

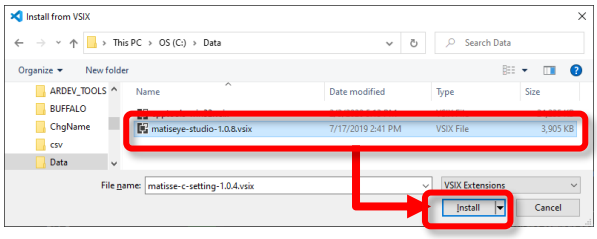


Figure 10. Install matiseye-studio_*.*.*.vsix

3 Select "matiseye-studio-*.*.*.vsix" and press [Install].

4 Restart VS Code.

© 2024 ROHM Co., Ltd.



Figure 11. matiseye-studio Extension Detail View

5 After restarting, the matiseye-studio will be added to the list of installed extensions. Clicking on it will display the Extension Detail View. If it says "Disable", the extension is enabled. If it says "Enable", clicking on it will enable the extension.

© 2024 ROHM Co., Ltd.

3. How to use Visual Studio Code (VS Code)

3.1. VS Code Window Description

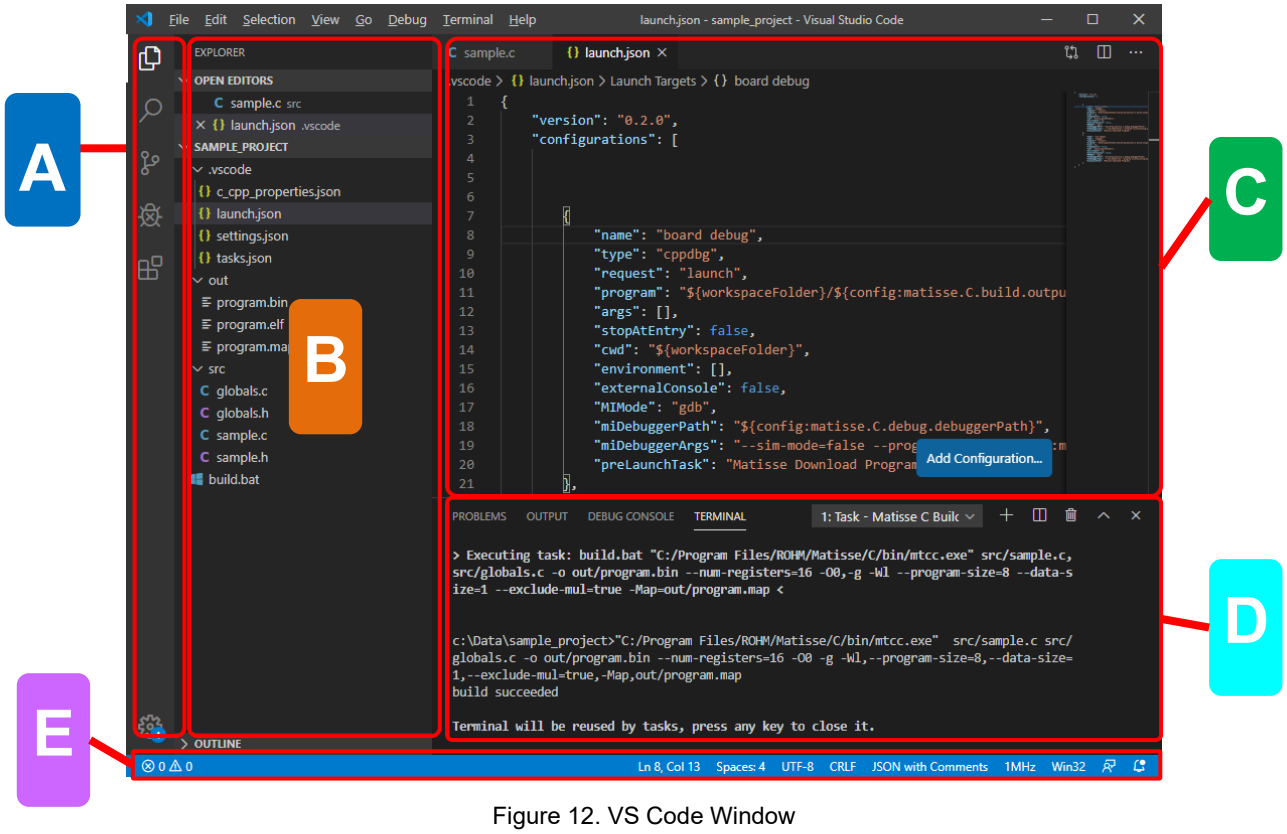


Figure 12. VS Code Window

Table 4. VS Code Window Description

	Name	Description
A	Explorer	Lists open files.
	Search	Searches for and replaces files with the specified keyword.
	Source Control	It works with Git.
	Debug and Run	Debug the program.
	Extensions	Search for an extension, etc.
B	Sidebar	When "Explorer" is selected, folders and files are displayed. When "Search" is selected search forms and results are displayed. Displayed items vary depending on the selected function.
C	Editor	Displays the contents of the open file. Split view of the editor is also supported.
D	Panel	Displays debugging information and command prompts.
E	Status Bar	Displays information about the status of the file, such as character codes and line feed codes. There are more details on the next page.

© 2024 ROHM Co., Ltd.

3.2. Status Bar Description

The status bar at the bottom of the screen allows you to check and change the current settings of VS Code.

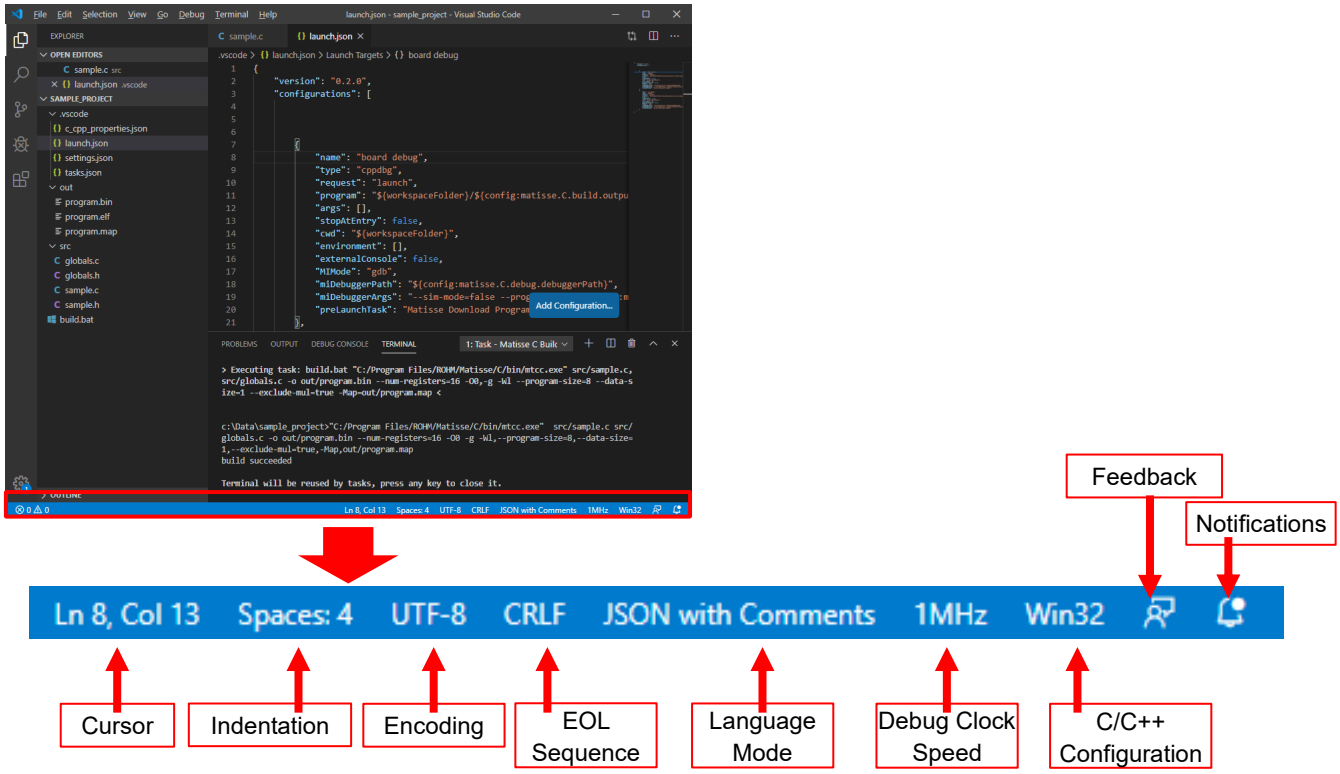


Figure 13. VS Code Status Bar

Table 5. VS Code Status Bar Description

Name	Description
Cursor	Displays the cursor position. Click to change it.
Indentation	Displays the indentation. Click to change it.
Encoding	Displays the character encoding. Click to change it.
EOL Sequence	Displays the EOL sequence or newline code. Click to change it.
Language Mode	Displays which language mode the current file is opened in. Click to change it.
Debug Clock Speed	Displays the debug interface operating frequency when debugging on the Matisse development board. Click to change the operating frequency. It is locked and cannot be changed during debugging.
C/C++ Configuration	Displays the language settings for C/C++. No need to change it. Do not click.
Feedback	Click to send feedback comments to Microsoft about VS Code. This is not a ROHM support contact.
Notifications	Clicking it will display notifications from VS Code.

Notes

You cannot contact ROHM support desk by using the feedback function.
Please contact the official support desk for feedback and troubleshooting of ROHM products.

© 2024 ROHM Co., Ltd.

3.3. Operating VS Code

Use sample_project to learn VS Code.

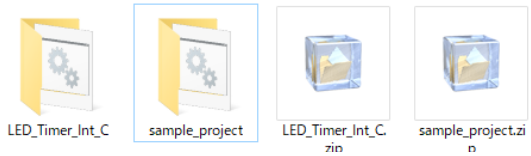


Figure 14. VS Code sample_project

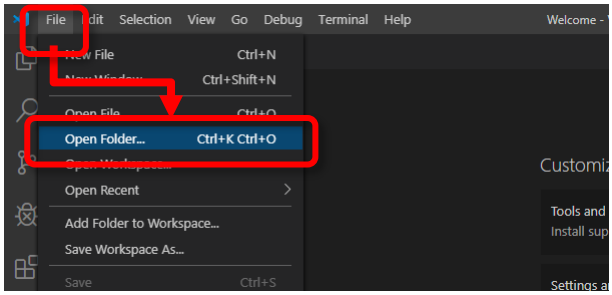


Figure 15. Click Open Folder

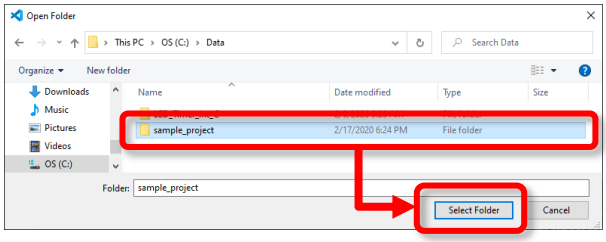


Figure 16. Select sample_project Folder

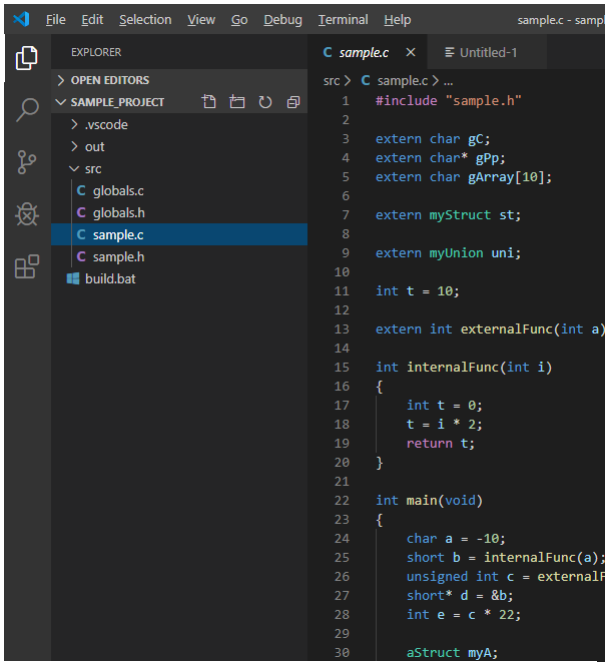


Figure 17. Edit C source file

Open the sample_project.

1 Unzip sample_project.zip and save it to a suitable folder.

Notes

The path of the folder should not contain multi-byte characters or spaces.

2 Launch VS Code and select "Open Folder" from File menu in the toolbar.

Notes

The workspace is not yet supported by matiseye™-studio. Please open the project folder directly.

3 Select the extracted "sample_project" folder and click [Select Folder].

Edit C source file

3 Select sample_project > src > sample.c in EXPLORER. The contents of the C source file is displayed. You can edit the file as is.

NOTES

If you write C syntactically incorrect, it will also be pointed out before compilation.

Building

Build the data and generate an executable file.

See "".

Debug

Check the operation of the program.

See "".

© 2024 ROHM Co., Ltd.

4. Project Configuration and Setting Items

4.1. Configuration of the Samples Project

This section explains the data structure and settings based on the sample project "sample_project".

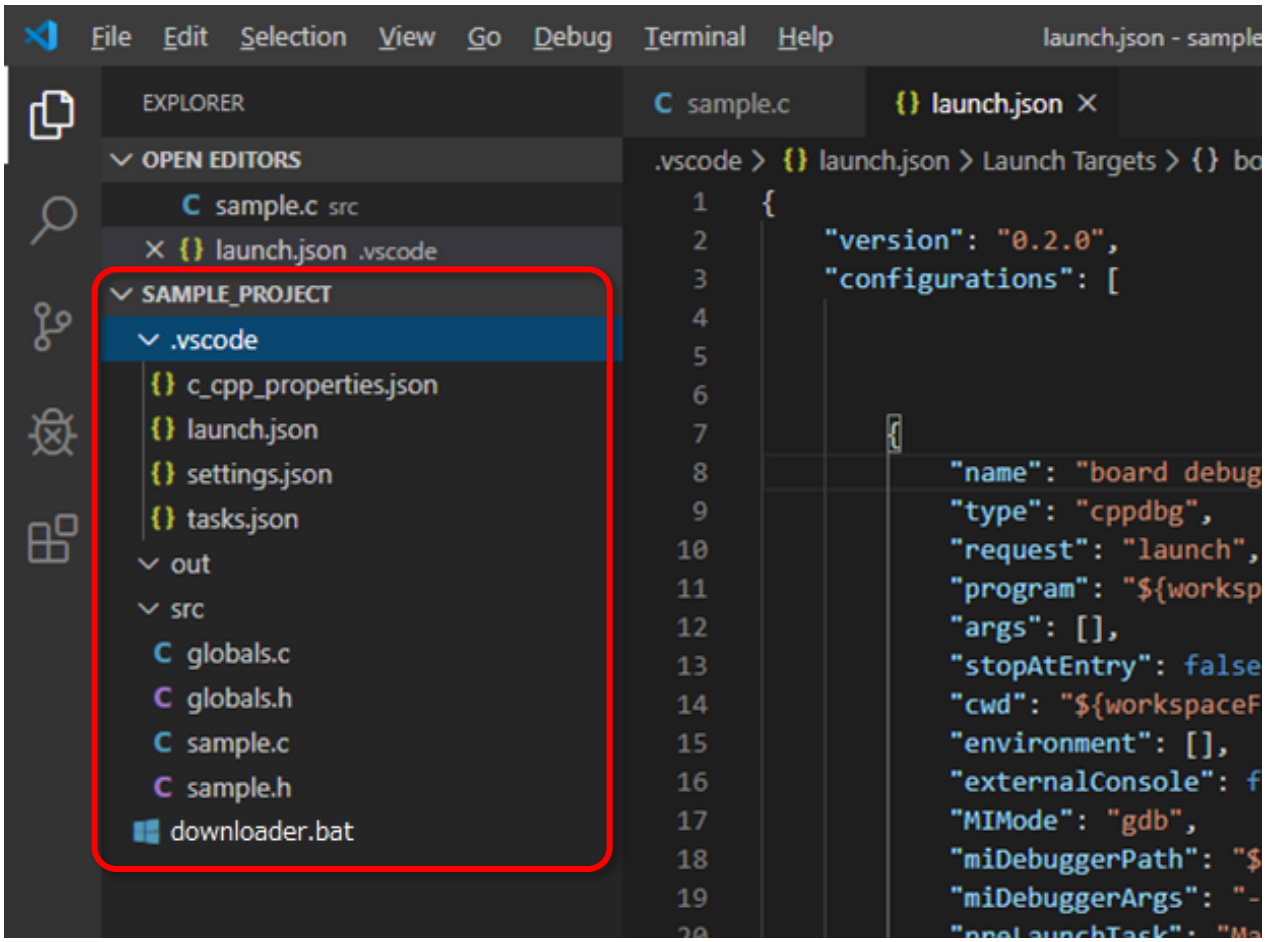


Figure 18. Samples Project Configuration

Table 6. Samples Project Configuration

Folder/file name	Description
.vscode/settings.json	Build and debug settings
src/	C language source code directory
out/	Build result output directory
downloader.bat	When using a development board with non-volatile memory, please run this batch file before debugging. Then the program data will be written to the non-volatile memory. See the program downloader manual for details.

© 2024 ROHM Co., Ltd.

4.2. Compiler/Debugger Setting Items（settings.json）

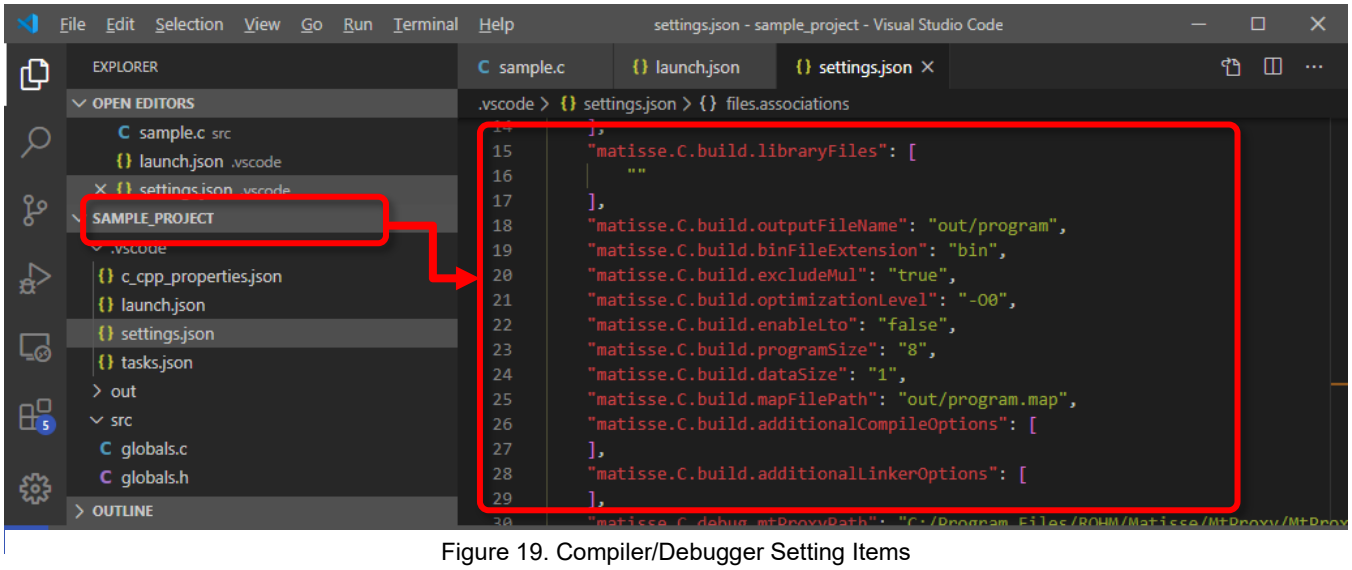


Figure 19. Compiler/Debugger Setting Items

Table 7. Compiler/Debugger Setting Items Description

Setting Name	Description
matisse.C.build.compilerPath (*1)	Path to the compiler(mtcc). Specify an absolute path.
matisse.C.debug.debuggerPath (*1)	Path to the debugger(mtcsim). Specify an absolute path.
matisse.C.debug.mtProxyPath (*1)	Path to the debug communication proxy server (MtProxy). Specify an absolute path.
matisse.C.others.downloaderPath (*1)	Path to the program downloader (mtloader). Specify an absolute path.
matisse.C.others.mtCheckerPath (*1)	Path to the development environment configuration checker (MtChecker). Specify an absolute path.
Matisse.C.build.srcFiles	A list of source files.
matisse.C.build.includePath	A list of directories from which load include files.
matisse.C.build.libraryPath	A list of directories from which load library files.
matisse.C.build.libraryFiles	A list of library file names.
matisse.C.build.preprocessorDefinitions	A list of preprocessor definitions (#define).
matisse.C.build.excludeMul	true: The target board doesn't include multiplier. false: The target board includes multiplier.
matises.C.build.optimizationLevel	Optimization level option. -O0(no optimization) / -O1(optimization level 1) / -O2(max optimization level) / -Os(Reduced ROM size)
matisse.C.others.downloaderOptions	A list of program downloader options. Separated with a comma.
(continued next page)	

NOTES

(*1) If you have changed the installation location from the default setting of the installer, you need to specify the absolute path to these tools in the configuration file. If not, you do not need to specify these fields.

Notes

The setting items should not contain multi-byte characters.

© 2024 ROHM Co., Ltd.

Setting Name	Description
matisse.C.build.programSize	Specify the size of the program area (8-14). 8: 32 kbyte / 9: 36 kbyte / 10: 40 kbyte / 11: 44 kbyte 12: 48 kbyte / 13: 52 kbyte / 14: 56 kbyte
matisse.C.build.dataSize	Specify the size of the data area (1-8). 1: 4 kbyte / 2: 8 kbyte / 3: 12 kbyte / 4: 16 kbyte 5: 20 kbyte / 6: 24 kbyte / 7: 28 kbyte / 8: 32 kbyte
matisse.C.build.exProgramSize	Specify the size of the extended program area (0-32). 0: 0byte / 1: 2kbyte / 2: 4kbyte ... / 32: 64kbyte
matisse.C.build.enableC99CompliantDiagnostics	true: The source code diagnostic function strictly complies with the C99 standard. false: The source code diagnostic function remains at the default settings.
matisse.C.debug.debugClockSpeed	Specify the clock speed of the debug interface. When you edit and save the value, the debug clock is immediately applied. - The matiseye-adapter pro can be set in the range of 10kHz - 24MHz. - The matiseye-adapter can be set in the range of 10kHz - 1.5MHz. If a value exceeding 1.5MHz is specified, 1.5MHz will be automatically applied. This item can be also configured from the status bar.
matisse.C.debug.showMemoryWindowOnStart	true: Show Memory Window at the start of debugging. false: Don't show Memory Window at the start of debugging. See "Memory Window".
matisse.C.debug.showPeripheralWindowOnStart	true: Show Peripheral Window at the start of debugging. false: Don't show Peripheral Window at the start of debugging. See "Peripheral Window".
matisse.C.debug.showBacktraceOnReset	true: Detects a CPU resetting during debugging, displays a function call history and stop debugging. false: Detect a CPU resetting and stop debugging. See "Function call history on CPU resetting".
matisse.C.debug.peripheralViewSvdPath	Path the peripheral register information file that PERIPHERALS reads at startup. Specify the absolute/relative path. The file is in CMSIS-SVD format. See "PERIPHERALS".
matisse.C.debug.enablePeripheralView	true: Show PERIPHERALS at the start of debugging. false: Don't show PERIPHERALS at the start of debugging.
"matisse.C.debug.showPerformanceViewOnStart"	true: Show Performance View at the start of debugging. false: Don't show Performance View at the start of debugging. See "PERFORMANCE".
"matisse.C.debug.performanceViewSampleRate"	Sampling rate settings when PERFORMANCE starts up.
matisse.C.others.extensionDebugLevel	This is a setting related to log output for the extension. Since it's intended for developers of matiseye-studio, it should normally be set to "None".
matisse.C.others.openExtensionDetailsOnProjectsStartup	Setting for whether to display the extension detail view when the project starts true: Displays the detail view when the project starts. false: Does not display the detail view when the project starts.
(Other settings don't need to be changed.)	

© 2024 ROHM Co., Ltd.

5. Tool Launcher (matiseye-studio Sidebar View)

The matiseye-studio Sidebar View (hereinafter referred to as mt Sidebar View) is a tool launcher that consolidates execution buttons for build processing and analysis tools in the sidebar, allowing each function to be executed with a single click or double-click.

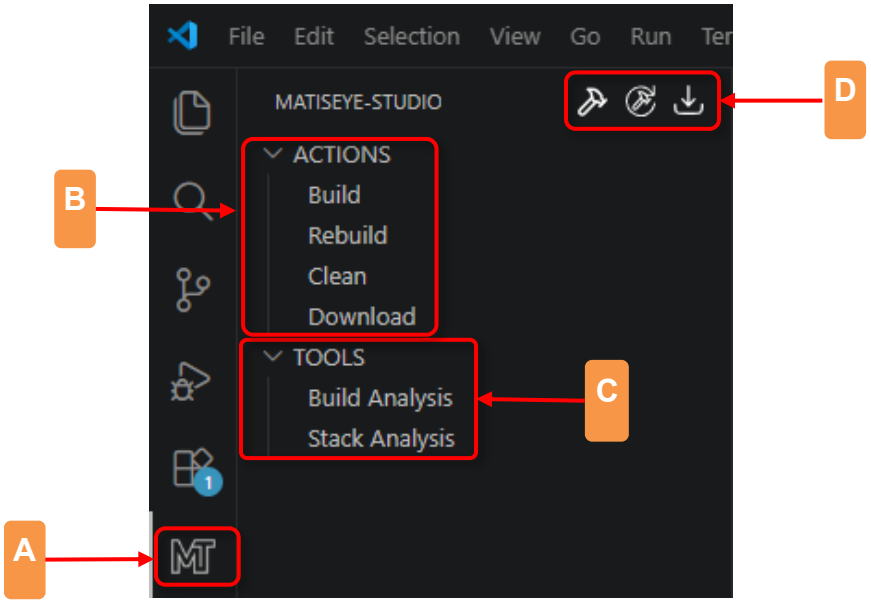


Figure 20. mt Sidebar View

Table 8. mt Sidebar View Description

	Name	Description
A	MT Icon	Clicking displays the "mt Sidebar View" View in the sidebar.
B	ACTIONS Tree	Double-click to execute build-related tasks defined in tasks.json. ・Build: Executes the "Build" task(same operation as Ctrl + Shift +B). ・Clean: Executes the "Clean" task to delete files generated by Build. ・Rebuild: Executes "Rebuild," which performs Clean followed by Build. ・Download: Executes the "Matisse Download Program" task and writes data to target. See "Running Build Task and Generated Files".
C	TOOLS Tree	Executes various analysis functions. ・Build Analysis: Analyzes build results. See "Build Analysis (Build Analysis View)". ・Stack Analysis: Analyzes stack usage. See "Stack Static Analysis".
D	ACTIONS Icons	Some ACTIONS operations can be executed by clicking the icons. From left to right, the buttons execute the Build, Rebuild, and Download tasks.

Note:
・If a Matisse project is not open, the MT icon will not be displayed.
・Task execution depends on tasks.json, so do not change the task names in the file.

© 2024 ROHM Co., Ltd.

6. How to Build

6.1. Running Build Task and Generated Files

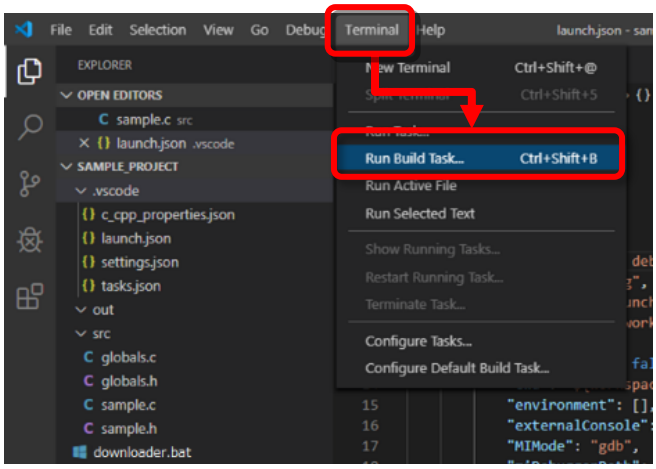


Figure 21. Execution Run Build Task

1 Choose Terminal > Run Build Task from the toolbar, or press Ctrl + Shift + B to run the build task.

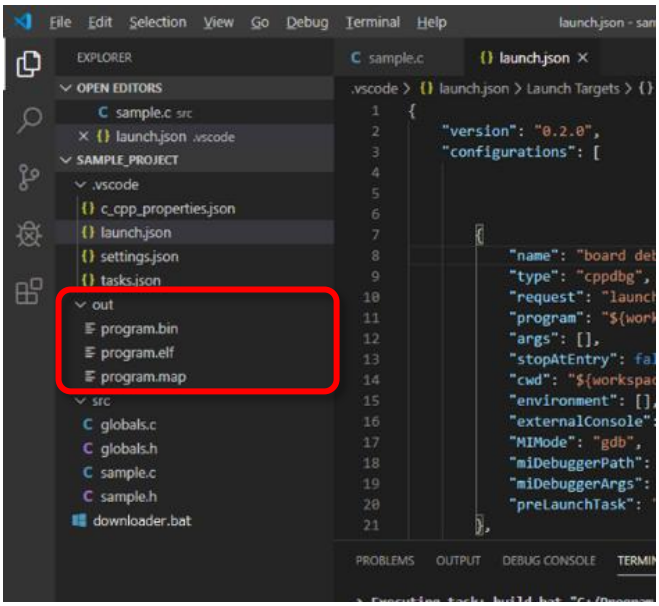


Figure 22. File Generation Run Build Task

2 After the build task is performed, the files are generated in the "out" folder.

Table 9. Run Build Task Description

File name	Description
out/program.map*1	MAP file. It can be used to check memory usage, address of functions, and address of variables.
out/program.elf*2	Program's debugging binaries.
out/program.bin*2	Program binaries.

© 2024 ROHM Co., Ltd.

6.2. MAP File

This section describes the format of MAP file.

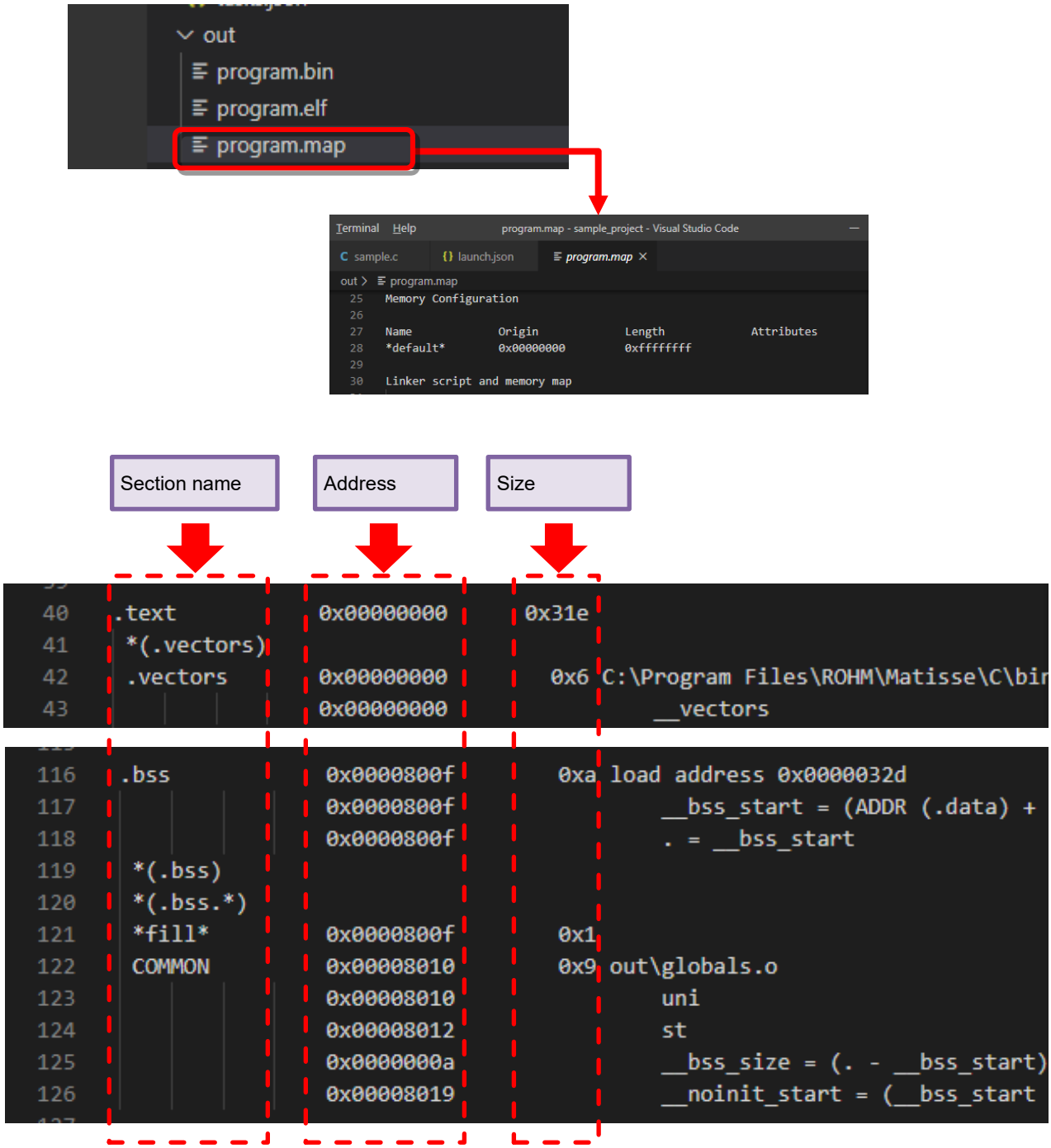


Figure 23. MAP File Configuration

NOTES

Basically, text is ROM (functions, constants table, etc.), and the.bss/.data is RAM.

© 2024 ROHM Co., Ltd.

6.3. Stack Static Analysis

The Stack View performs static analysis of stack usage based on the built program. It displays the maximum stack usage per function and the maximum usage for the entire program, helping to prevent stack overflow in advance. Additionally, it allows you to check the call relationships between functions.

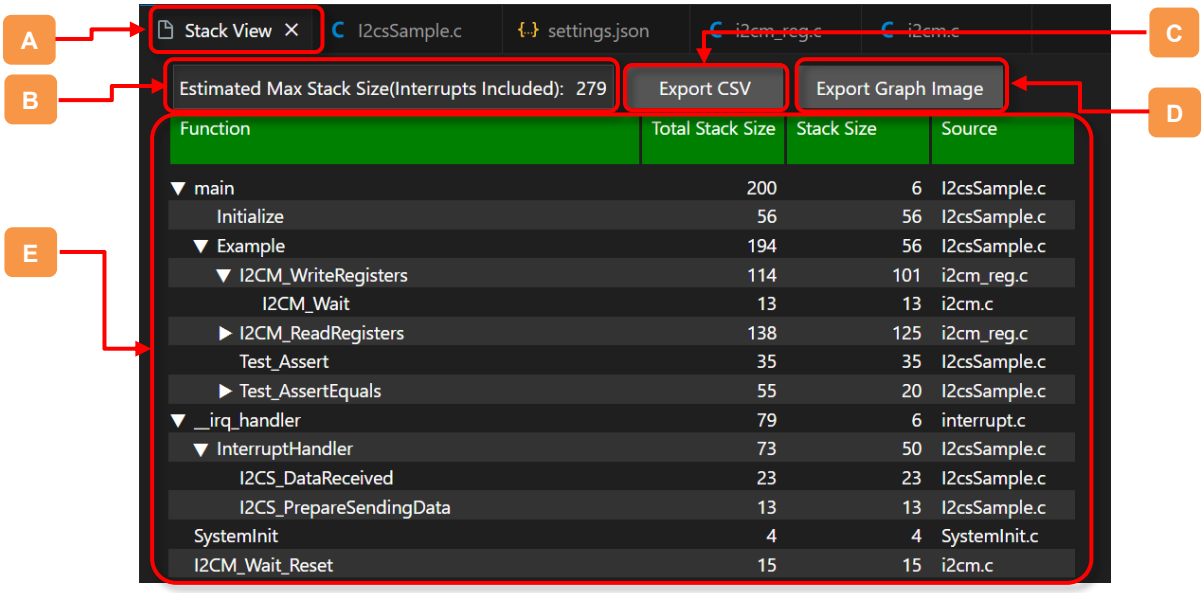


Figure 24. Stack View

■Start Procedure: To start the stack analysis process, select "Matisse Analysis: Start Stack Analysis" from the command palette. Once the process is complete, the Stack View will appear on the left side of the editor screen.

See "Displaying the Command Palette and Command Input".

Table 10. Stack View Description

	Name	Description	
A	Stack View Tab	Displayed on the right side of the editor. If you press the close button while there is unsaved data, a file save confirmation window will appear. To save data, use the "Export CSV".	
B	Maximum Stack Usage	Displays the maximum stack usage for the entire program. This value is the sum of the maximum stack usage of the main function and the interrupt handlers (e.g., *irg_handler, nmi_handler*).	
C	Export CSV	Outputs the "Detailed Stack Usage Table" as a CSV file.	
D	Export Graph Image	Outputs the "Function" call graph as a PNG image file.	
E	Detailed Stack Usage Table	Column Name	Description
		Function	Displays the call relationships in a tree format. **NOTES** Functions that are not called by eny other function are displayed as root nodes
		Total Stack Size	In addition to the "Stack Size" of the function itself, the displayed value is the sum of the "Total Stack Size" of the function with the largest stack usage among all the functions called by that function.
		Stack Size	Displays the maximum stack usage of the function itself.
		Source	Displays the file name where the function is defined.

Notes

When compiler optimization is enabled, function inlining may cause the call graph to differ from the actual program.

© 2024 ROHM Co., Ltd.

6.4. Build Analysis (Build Analysis View)

The Build Analysis View analyzes the .elf and .map files generated upon build completion and visualizes ROM and RAM memory usage. By using this feature, you can understand the program's memory usage and utilize it for optimization considerations.

The display consists of two screens: "Summary Display" and "Detailed Analysis View."

```
================================================================================================================
Build Analysis View
================================================================================================================
Generated at: 2026-05-19 12:56:06 (UTC+09:00)
Sources:
  Map File : c:\Users\191019\OneDrive - ROHM Co., Ltd\matiseye-studio_develop\src\TestProject\BD15034_LE
  ELF File : c:\Users\191019\OneDrive - ROHM Co., Ltd\matiseye-studio_develop\src\TestProject\BD15034_LE
Compiler Options: "-I../Driver/" "-I../middleware/" "-I../middleware/iot_protocol/" "-Isrc/" "-ffunction
Linker Options: "-Wl,--program-size=12,--data-size=2,--ex-program-size=0,--exclude-mul=True,-Map=out/pro
================================================================================================================


[1. Memory Usage Summary]
----------------------------------------------------------------------------------------------------------------
Region   | Address Range             | Used / Total            | Usage   | Included Sections
----------------------------------------------------------------------------------------------------------------
ROM      | 0x00000000 - 0x0000c000   | 47,491 / 49,152 bytes   | 96.62% | .text, .data (LMA), .bss (LMA)
RAM      | 0x0000c000 - 0x0000e000   | 2,779 / 8,192 bytes     | 33.92% | .data, .bss
----------------------------------------------------------------------------------------------------------------


[2. Stack Usage Analysis]
----------------------------------------------------------------------------------------------------------------
Stack Capacity (RAM Free Area) : 5,413 bytes (Range: 0x0000cadb - 0x0000e000)
Max Stack Usage (Static Calc)  : 556 bytes
----------------------------------------------------------------------------------------------------------------
Margin                         : + 4,857 bytes
Evaluation                     : Safe
----------------------------------------------------------------------------------------------------------------
* "Max Stack Usage" includes interrupts context.
```

Figure 25. Build Analysis View (Detailed Analysis View)

6.4.1. Summary Display

The Summary Display has the following features:

- Displays ROM/RAM usage and usage rates.
- If the target .elf or .map file does not exist, the Summary Display will not be shown.
- When file is updated by a Build, the displayed content is automatically updated.
- Clicking the Summary Display opens the Detailed Analysis View.

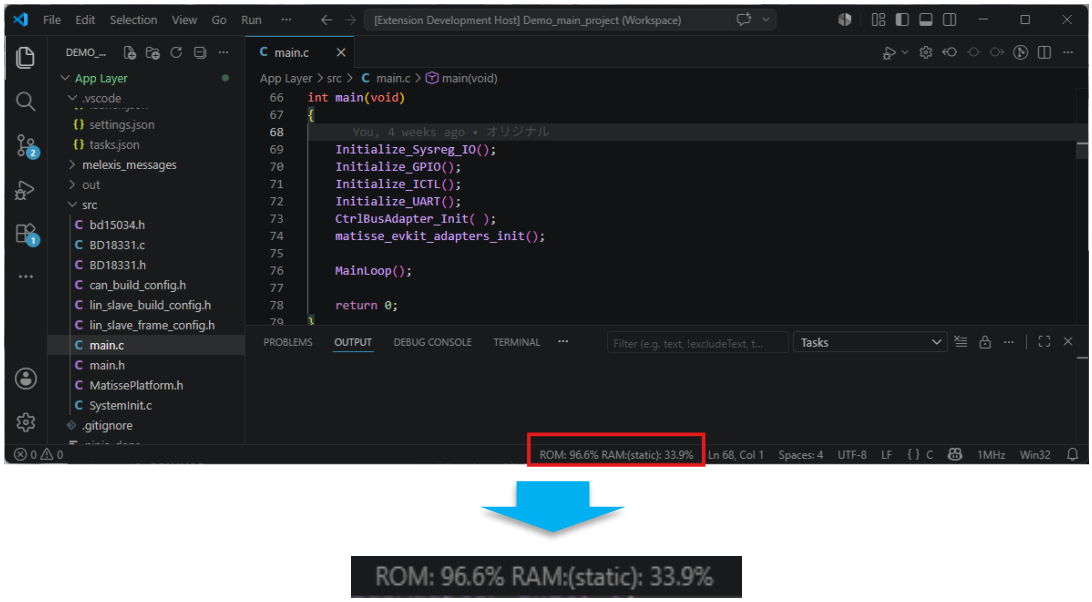


Figure 26. Build Analysis View (Summary Display)

© 2024 ROHM Co., Ltd.

- When you hover the mouse cursor over the Summary Display, you can check detailed information for each memory area in a tooltip.

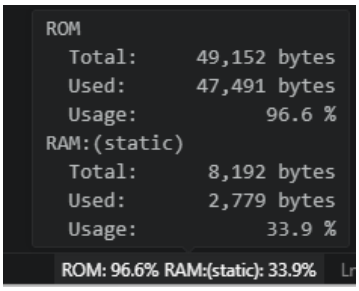


Figure 27. Summary Display (Tooltip)

Table 11. Summary Display (Tooltip) Description

Name	Description
Total	Total capacity of each memory area.
Used	Used amount of each memory area.
Usage	Usage rate of each memory area.

6.4.2. Detailed Analysis View

The Detailed Analysis View has the following features:

- It is displayed after the build is completed when you click the Summary Display or [Build Analysis] in the Sidebar View.
- Displays ROM/RAM usage, usage rates, and address ranges in a list.
- Displays the analysis results of stack usage (maximum usage, margin, etc.).
- Displays a list of each symbol's address, size, section at compile time, and source file.

```
[1. Memory Usage Summary]
-------------------------------------------------------------------------------------------------------------
Region   | Address Range            | Used / Total           | Usage  | Included Sections
-------------------------------------------------------------------------------------------------------------
ROM      | 0x00000000 - 0x0000c000  | 47,491 / 49,152 bytes  | 96.62% | .text, .data (LMA), .bss (LMA)
RAM      | 0x0000c000 - 0x0000e000  | 2,779 / 8,192 bytes    | 33.92% | .data, .bss
-------------------------------------------------------------------------------------------------------------
```

Figure 28. Detailed Analysis View (Memory Usage Summary)

Table 12. Memory Usage Summary Description

Name	Description
Region	Type of memory area (ROM/RAM).
Address Range	Address range of the corresponding memory area, defined in the linker script. **Notes**: The RAM end address is obtained from the "__data_end" keyword in the map file. This value can be changed using the "matisse.C.build.dataSize" setting in "settings.json". See "Compiler/Debugger Setting Items（settings.json）".
Used/Total	Used amount and total capacity.
Usage	Usage rate.
Included Sections	A list of output sections placed in each memory area (ROM/RAM) within the executable by the linker. An output section refers to the final section created by merging input sections from multiple object files according to the linker script.

© 2024 ROHM Co., Ltd.

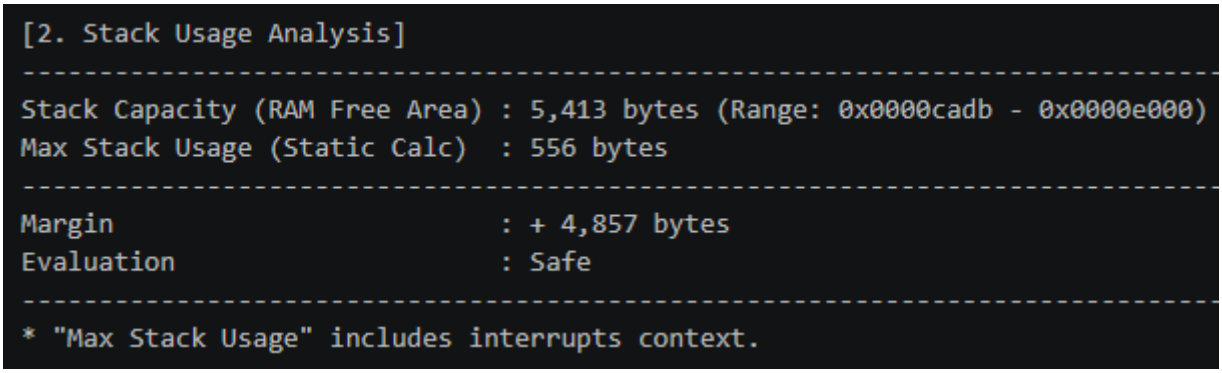


Figure 29. Detailed Analysis View (Stack Usage Analysis)

Table 13. Stack Usage Analysis Description

Name	Description
Stack Capacity	Total stack capacity and address range. The start address is calculated as "RAM start address + RAM usage after build."
Max Stack Usage	Maximum stack usage based on static analysis (obtained from Stack View).
Margin	The theoretical margin size in the RAM area reserved to prevent stack overflow at runtime. Multi-level interrupt cases are not considered. Calculated as Stack Capacity - Max Stack Usage.
Evaluation	Evaluation result of the margin. If the margin is 0 or greater, "Safe" is displayed; if negative, "Capacity Exceeded" is displayed.

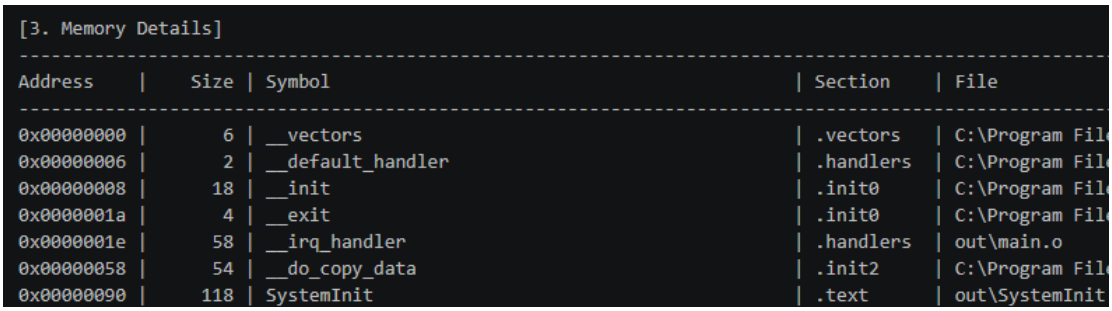


Figure 30. Detailed Analysis View (Memory Details)

Table 14. Memory Details Description

Name	Description
Address	Address where the symbol is placed.
Size	Size of the symbol.
Symbol	Symbol name (function name or variable name). Static symbols that are not shown in the map file are also displayed.
Section	Input section to which each symbol belongs. An input section refers to a section defined within each object file (.o).
File	Object file from which it was generated.

Notes

To save the output of the Detailed Analysis View, click anywhere in the display and press Ctrl + S.

© 2024 ROHM Co., Ltd.

Notes

- If the compiler optimization option is enabled, symbols that have been inlined due to optimization will not be listed.
- The memory area information displayed by this feature is based on analysis results at build time. The memory layout at runtime may vary depending on the actual hardware environment and settings.
- The method of placing program memory areas is defined in the linker script. If you need to change the location or size of memory areas, modify the linker script. For details, refer to the **mtld User's Guide**.

© 2024 ROHM Co., Ltd.

7. Debug

7.1. Debug Window Descriptions

To view the debugging Window, select (Run and Debug).

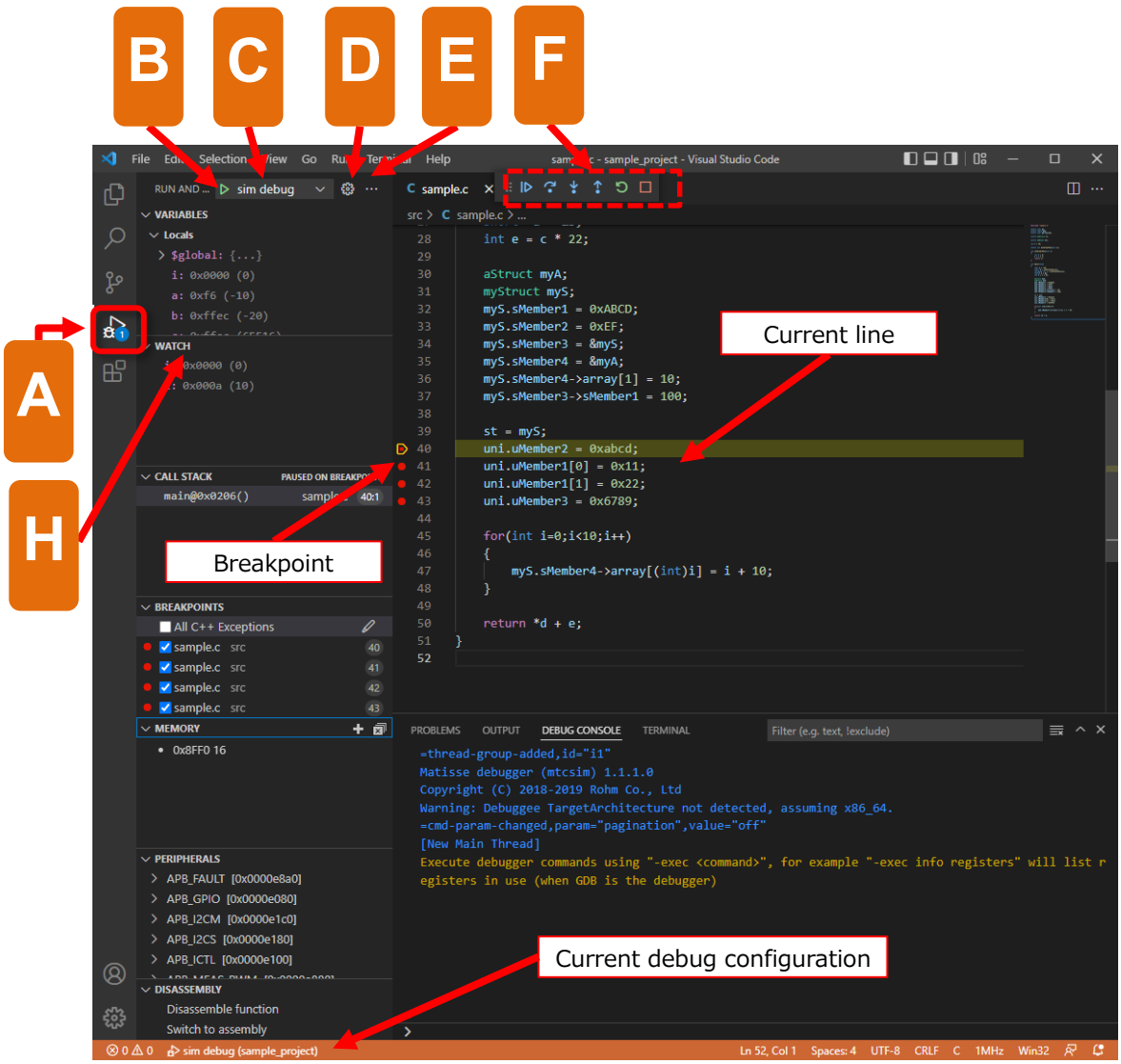


Figure 31. Debug Window when "Debug and Run" selected

Table 15. Debug and Run Description

	Name	Description
A	Debug and Run	Display the debugging screen.
B	Start Debugging	Start debugging.
C	Choosing the debug mode	Select board debug (when hardware is connected) or sim debug (when hardware is not connected) to start debugging in the mode shown.
D	Open launch.json	Create/display launch.json files and allows you to change the debugging configuration.
E	View Settings	The debug console appears.
F	Debug toolbar	See "Debug Toolbar".
	(continued on next page)	

© 2024 ROHM Co., Ltd.

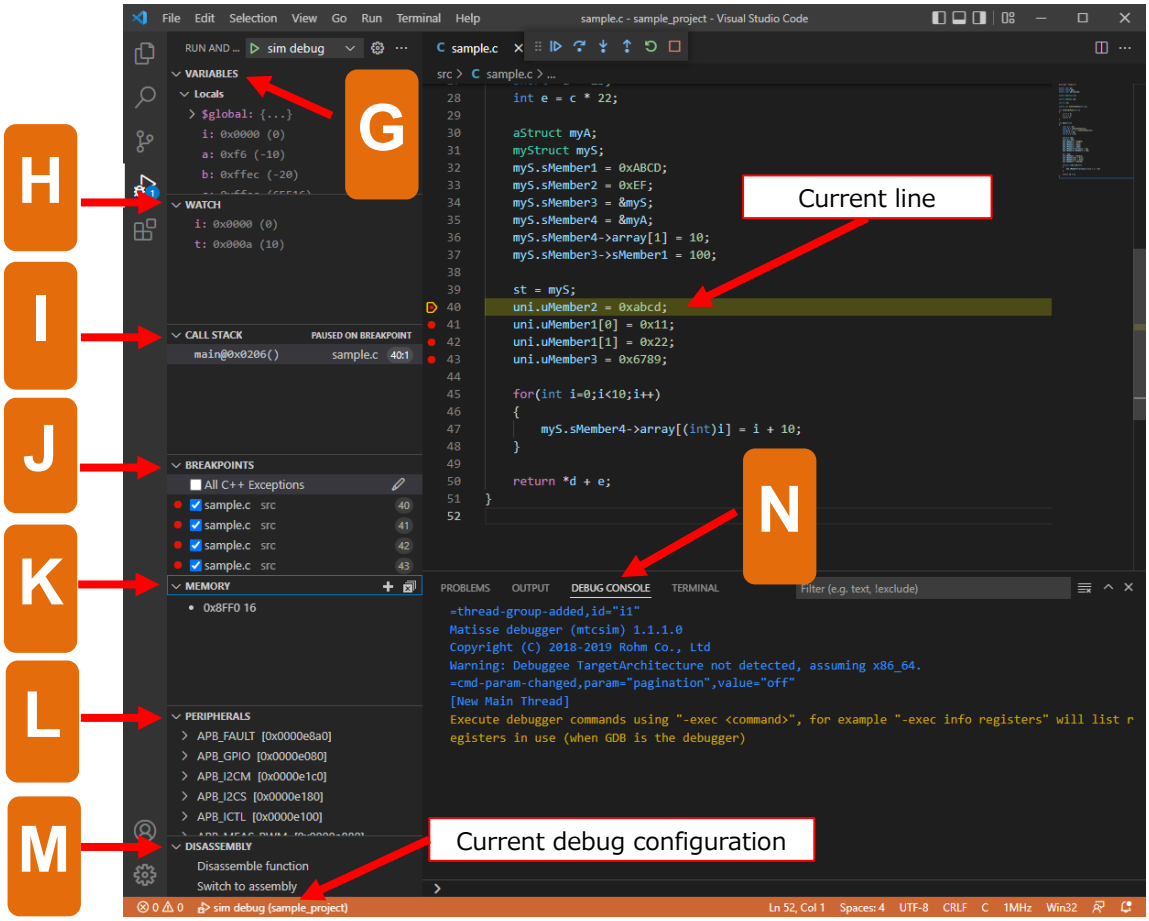


Figure 32. Debug Window when "Debug and Run" selected

Table 16. Debug and Run Description

	Name	Description
G	VARIABLES	Displays the variable names and values.
H	WATCH	See "Data Inspection".
I	CALL STACK	See "Call stack".
J	BREAKPOINTS	See "Breakpoints".
K	MEMORY	See "MEMORY".
L	PELIPHERALS	See "PERIPHERALS".
M	DISASSEMBLY	See "DISASSEMBLY".
N	Debug Console	Displays debug logs.

© 2024 ROHM Co., Ltd.

7.2. Debug Menu

You can also work debugging items from Debug on the toolbar.

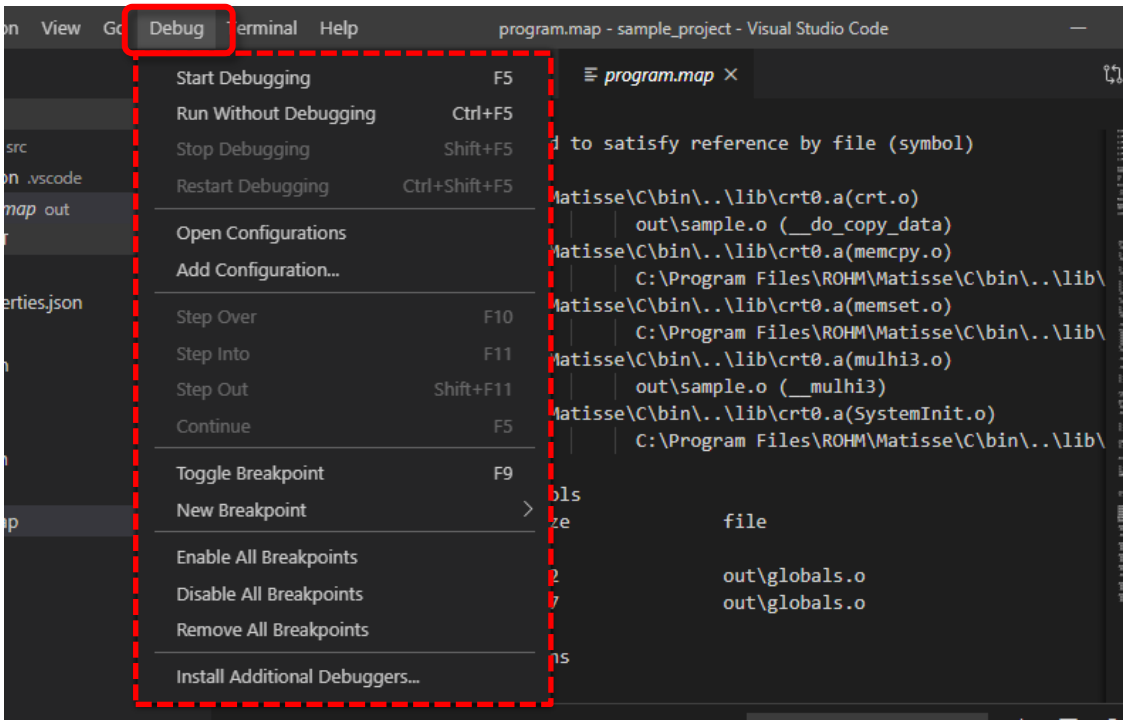


Figure 33. Debug Menu

© 2024 ROHM Co., Ltd.

7.3. Breakpoints

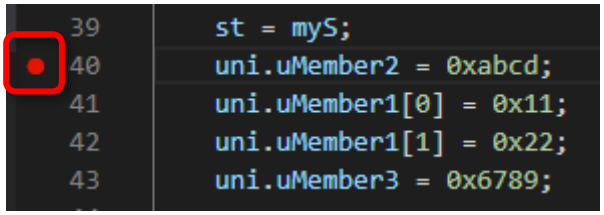


Figure 34. BREAKPOINTS

You can add or delete breakpoints by clicking to the space to the left of the code line number.
When debug session starts, the program breaks just before executing the line of code where you added the breakpoint.

NOTES

- Breakpoints could be disabled and displayed in gray if you change source code during debugging. In that case re-build the program and re-start debugging.
- Breakpoints on lines that do not involve arithmetic processing, such as variable declarations and blank lines are disabled and displayed in gray.

BREAKPOINTS window

You can turn on, off or delete the breakpoints both individually and collectively.
You can specify function name on "BREAKPOINTS window", if you want to break on the start of the function.

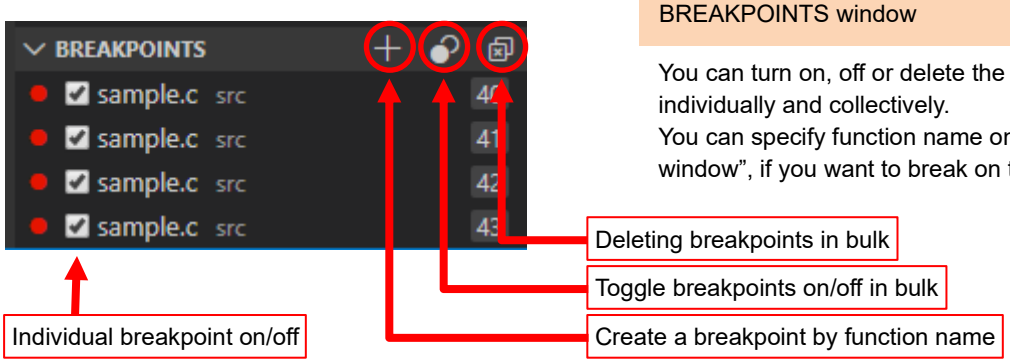


Figure 35. BREAKPOINTS Window

© 2024 ROHM Co., Ltd.

7.4. Debug Toolbar

Once a debug session starts, the Debug toolbar will appear on the top of the editor.

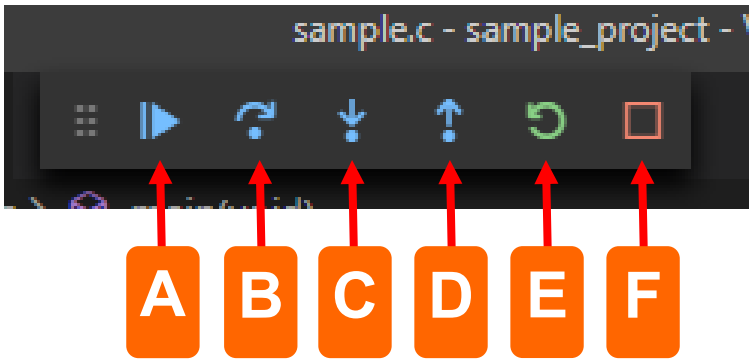


Figure 36. Debug Toolbar

■The left button has the Step Execution function as follows.

Table 17. Debug Toolbar Left Button Description

	Name	Keyboard Shortcuts	Description
A	Continue	F5	Go to the next breakpoint.
	Pause	F6	Breaks a running program.
B	Step Over	F10	Executes the current line and proceeds to the next line.
C	Step Into	F11	If the execution of the current line is a function, proceed into it.
D	Step Out	Shift+F11	Execute until exiting the currently executing function.

■The button on the right has functions related to the entire debug execution.

Table 18. Debug Toolbar Right Button Description

	Name	Keyboard Shortcuts	Description
E	Restart	Ctrl+Shift+F5	Restart debugging.
F	Stop	Shift+F5	Terminate debugging.

© 2024 ROHM Co., Ltd.

7.5. Data Inspection

Checking the value of variables while the program breaks is called data inspection.
You can use this feature in the "VARIABLE window" and "WATCH window".

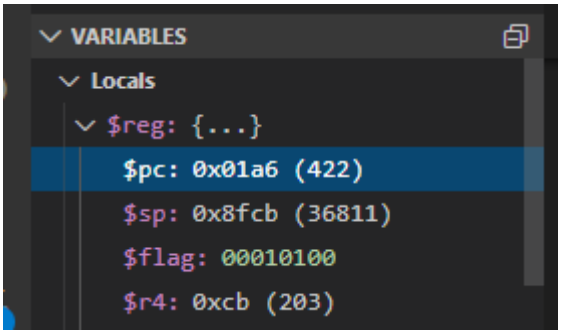


Figure 37. VARIABLES Window

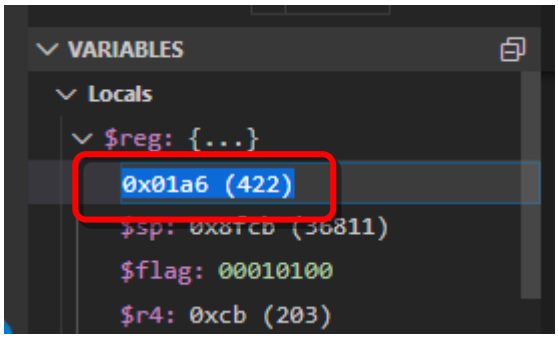


Figure 38. Editing Values

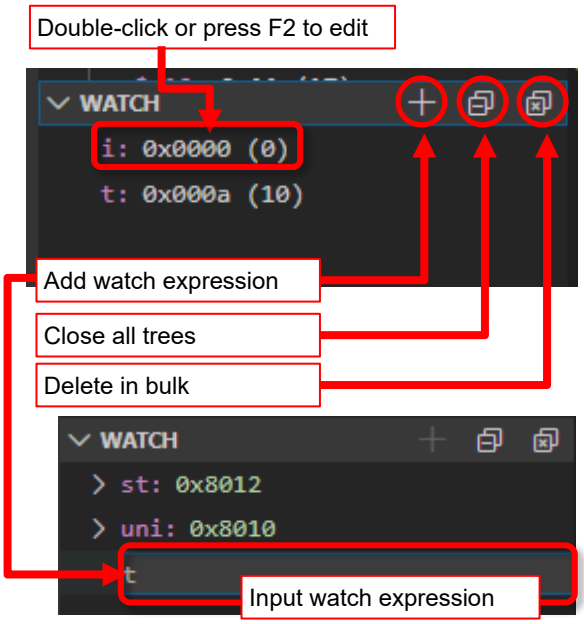


Figure 39. WATCH Window

VARIABLES window

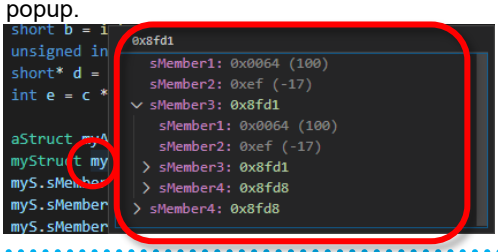
You can check and edit the value of variables.
If the variable type is a structure, pointer, or array, details of each element are displayed in a tree format.
The variables displayed in the "VARIABLE window" are local variables on the call stack.

See "Call stack".

Use "WATCH window" for displaying global variables

NOTES

When you hover the mouse cursor over a variable in the source code, the contents of the variable are displayed in a popup.



WATCH window

Displaying the result of evaluating an expression for a specific variable is called a watch expression.
You can specify variable names in "WATCH window". You can add or delete variables you want to monitor changes.

NOTES

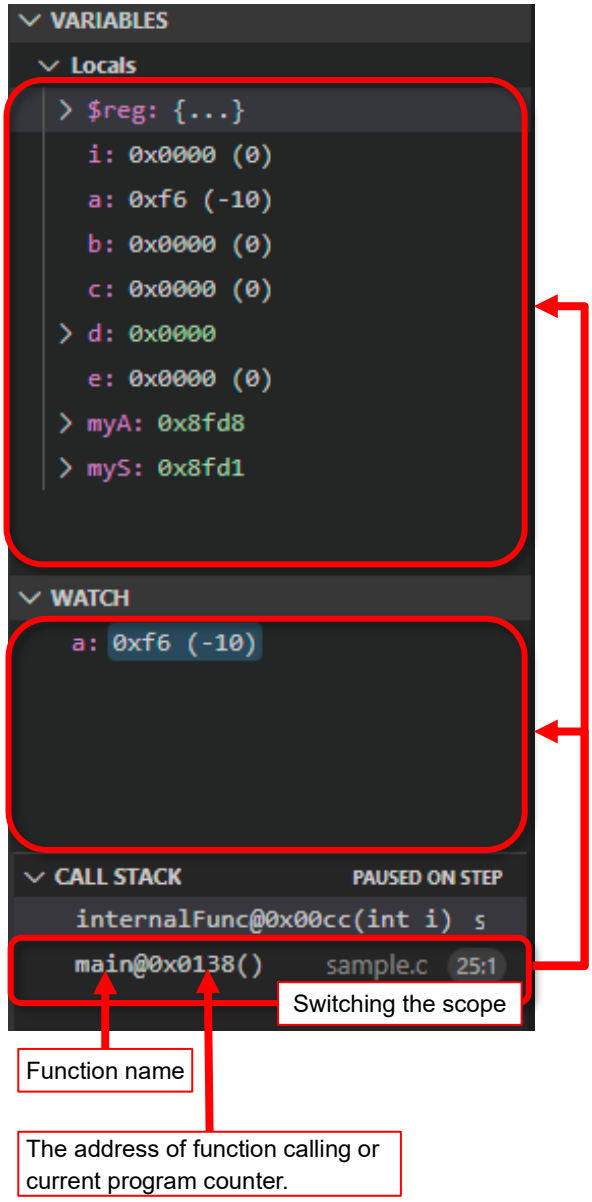
Use "VARIABLE window" to change the value of a variables. You cannot change the value in the "WATCH window".
You can also add a watch expression by right-clicking the variable in the "VARIABLES window" and clicking "Add to Watch".



© 2024 ROHM Co., Ltd.

7.6. Call stack

The "CALL STACK window" displays the history of function calling in reverse order.



If you select a function name displayed in the "CALL STACK window", the contents of the "VARIABLES window" and "WATCH window" are switched to the contents according to the scope of the function.

Figure 40. CALL STACK Window

7.7. MEMORY

MEMORY area displays the memory value of the specified range in the screen of VS Code. The display screen shows the memory range and the corresponding values. This feature can be launched regardless of whether debugging is in progress.

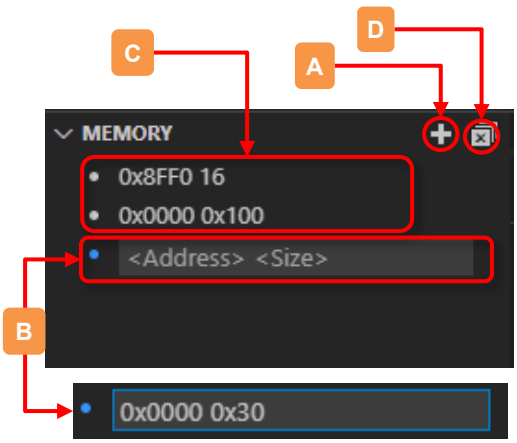


Figure 41. MEMORY

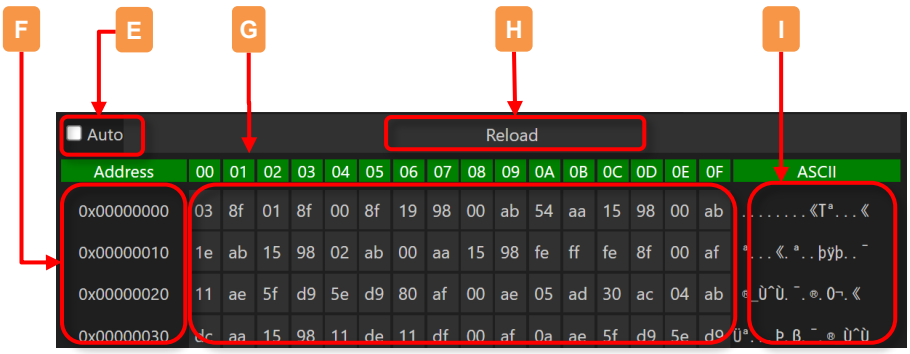


Figure 42. MEMORY display Editor Area

Table 19. MEMORY Description

	Name	Description
A	Add Memory Display	Clicking this opens the "Add Memory Display Form," where you can specify the range of memory addresses you want to display.
B	Add Memory Display Form	Enter the starting address and size of the memory range you want to display, then press Enter. You can input values in both hexadecimal and decimal formats. Memory ranges that exceed the installed memory cannot be displayed.
C	Display History	A list of previously displayed memory ranges is shown. Clicking on an entry re-displays the corresponding memory range in the editor area.
D	Delete Display History	Clicking this deletes all previously displayed memory range histories.
E	Auto-Reload Toggle	Checking this starts auto-reloading. Unchecking it stops auto-reloading.
F	Memory Address	The starting address of each line of memory is displayed.
G	Memory Address offset	Click the list item to write values.
H	Reload Button	Clicking this reloads the memory values and updates them to the latest values.
I	Memory Value	The memory values corresponding to each address are displayed. Double-clicking allows you to edit the values. When reloading, values that have changed from the previous values are displayed in red. If there are no changed values, the memory values that were previously displayed in red will continue to be displayed in red. **NOTES**: You can copy a desired range to the clipboard by selecting it with a mouse drag and pressing Ctrl+C.
J	ASCII Display of Memory Values	The memory values are displayed in ASCII format.

NOTES: It can be launched when the board is connected, regardless of whether debugging is in progress.

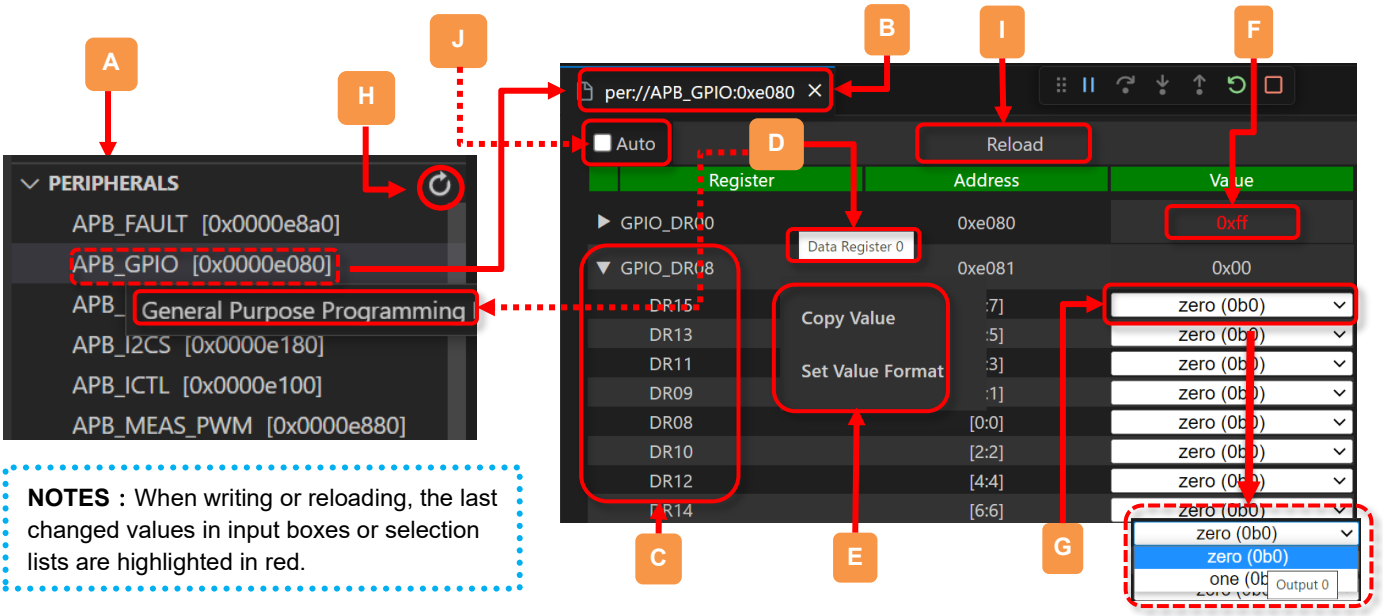
NOTES

Limit the concurrent use of auto-reload for MEMORY and PERIPHERALS to a maximum of two tabs. Running more than two simultaneously increases communication load and may cause communication errors.

© 2024 ROHM Co., Ltd.

7.8. PERIPHERALS

PERIPHERALS displays device info on screen for reading/writing values. Data is from CMSIS-SVD file (<XXX>) specified in File Path Setting. This feature can be launched regardless of whether debugging is in progress.



NOTES : When writing or reloading, the last changed values in input boxes or selection lists are highlighted in red.

Figure 43. PERIPHERALS View

Table 20. PERIPHERALS Description

	Name	Description
A	Peripheral Lists	Peripheral name double-click creates corresponding tab in editor.
B	Peripheral Tabs	Each tab shows a hierarchical list of registers for a peripheral. • The register layer displays name, offset, and value. • The field layer displays bit field name, width, and value or list name.
C	Tree Expand/Collapse	Double-click on a register/field name or click on the "▸" to expand/collapse it.
D	Description Tooltip	Mouse over the register/field name to see the contents of the <description>.
E	Popup Menu	Right-clicking on a peripheral/register/field name opens a menu for performing the following functions. • "Copy Value": Copy the register/field value to the clipboard (If "<access>" is write-only , "Copy Value" is not displayed.). • "Set Value Format": Switch display format of register/field to 16, 10, or binary **NOTES:** If register is "<readAction>", "Force Read" is displayed.
F	Value Input Box	Enter a value (in hexadecimal/decimal/binary format) and press Enter to write.
G	Register Value List	Click the list item to write values.
H	Reload All Button	Click to reload and update the display of all tabbed peripheral register values.
I	Reload Single Button	Click to reload and update the display of the register value for the current tab.
J	Auto Reload Check	When checked, the "Register Single Button" is automatically executed every 1s.

File path setting

SVD files are written in XML, a markup language that uses tags ("<**>").
The file path can be set in settings.json (compiler/debugger settings).

See "Project Configuration and Setting Items".

© 2024 ROHM Co., Ltd.

NOTES :

- By default, multi-byte registers are written in MSB to LSB order, but can be changed to LSB to MSB using the specified keyword. Additionally, the byte order is little-endian.
- It can be launched when the board is connected, regardless of whether debugging is in progress.

NOTES

Limit the concurrent use of auto-reload for MEMORY and PERIPHERALS to a maximum of two tabs. Running more than two simultaneously increases communication load and may cause communication errors.

© 2024 ROHM Co., Ltd.

7.9. DISASSEMBLY

The DISASSEMBLY feature displays the disassembly results (assembly instructions) for individual functions or for the entire executable file. Particularly when displaying the entire executable (C), you can view the original source code and the assembly instructions side-by-side. This feature is available only in debug mode.

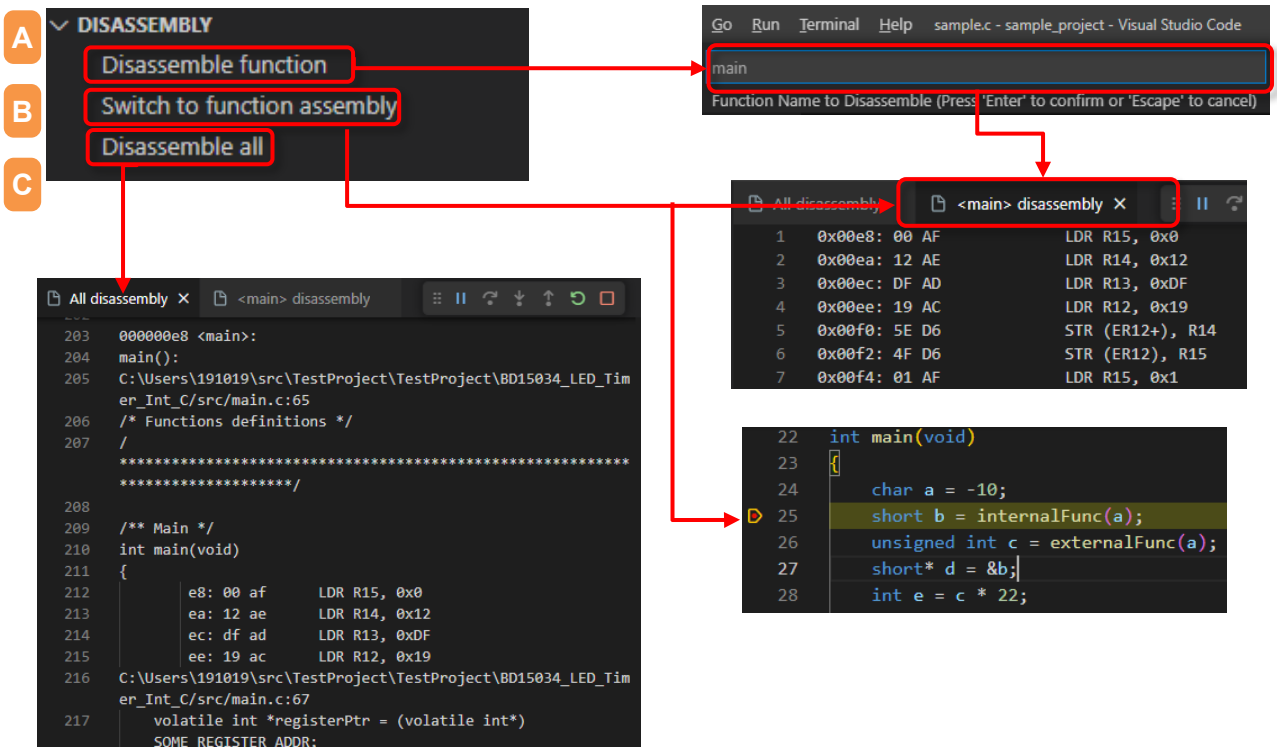


Figure 44. DISASSEMBLE View

Table 21. DISASSWMBLE View Description

	Name	Description
A	Disassemble function button	A button to display the disassembly result for a specific function. Clicking it opens a command palette to enter a function name. Enter the name of the function you want to disassemble and press Enter. The disassembly result for that function will be newly displayed in a "<Function Name> disassembly" tab. **NOTES:** If the result does not appear even after entering the function name, the function may have been inline due to compiler optimization.
B	Switch to function assembly / Switch to function source button	A toggle button that switches between "assembly view" and "source code view" for the currently open function. Each time you press it, the button name also changes between "Switch to function source" and "Switch to function assembly". **NOTES:** If the currently executing function has been inline, the disassembly result of its caller function will be displayed.
C	Disassemble all button	A button to display the disassembly result for the entire executable file. Clicking it newly displays the disassembly result corresponding to the entire address range of the executable file in the "All disassembly" tab. In this view, assembly instructions and their corresponding source code (with file name and line number) are displayed interleaved.

NOTES: This feature is available only in debug mode.

© 2024 ROHM Co., Ltd.

7.10. PERFORMANCE

PERFORMANCE displays the execution time ratio of each function in a program. During debugging, the program counter (PC) is periodically sampled, increasing the count for the current function. This helps verify the execution ratio of each function and identify high-load functions, enabling efficient system performance enhancement.

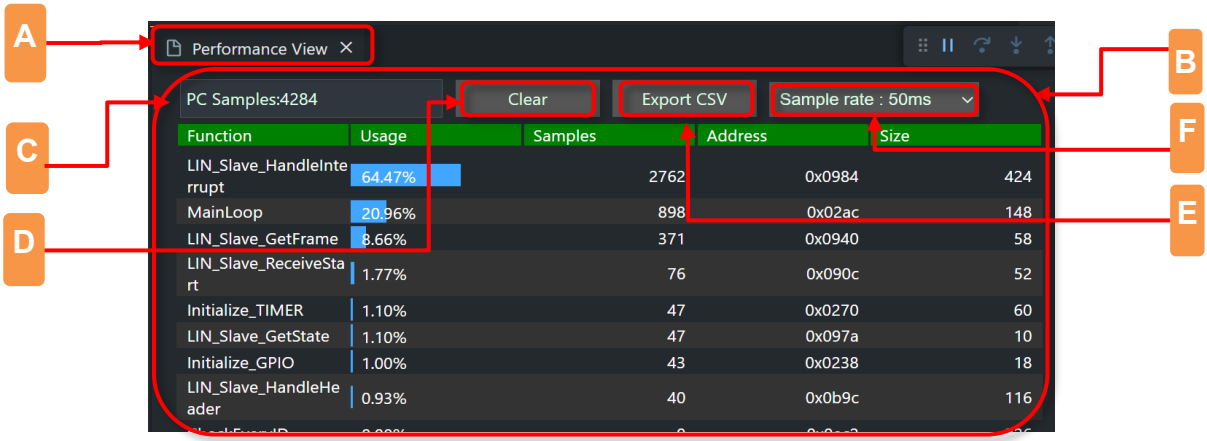


Figure 45. PERFORMANCE View

■Startup Procedure: You can enable startup in the settings file (settings.json), or start it during debugging by selecting "Matisse Debug: Start Performance View" from the command palette. Once launched, the above screen will appear, and sampling will begin. You can also specify the sampling rate at startup in the settings file.

See "Project Configuration and Setting Items".

■Pause Procedure: Stopping method: Sampling is interrupted during break, pause, and step execution.

Table 22. PERFORMANCE View Description

	Name	Description	
A	Performance View Tab	Displayed on the right side of the editor. If unsaved data is present when you press the close button, a file save confirmation pop-up will appear. Use the "Save File" button to save data.	
B	PERFORMANCE Table	Updates every 1s. Details are as follows:	
		Name	Description
		Function	Lists function names in the program, ordered by the number of samples, and updates automatically.
		Usage	Displays the ratio and graph of the number of samples for each function relative to the total.
		Samples	Counts the number of samples executed during sampling.
		Address	Displays the start address of each function.
		Size	Displays the size of each function.
C	Total Samples	Displays the total number of sampled PC. Sampling occurs only during debugging (excluding pause and stop).	
D	Clear Button	Clears the display of Total Samples, Usage, and Samples (resets to 0).	
E	Save File Button	Saves the Total Samples and table information as a CSV file.	
F	Sampling Rate Selection List	Allows selection of the sampling rate from the dropdown list. Can be changed at any time. Rates: 30ms/50ms/70ms/100ms/200ms/500ms/1s.	

Note: Results are statistical estimates based on sampling. We recommend measuring for at least 30 seconds to a few minutes to ensure higher accuracy.
Please avoid using the MEMORY or PERIPHERALS auto-reload function at the same time, as this may compromise measurement accuracy.

© 2024 ROHM Co., Ltd.

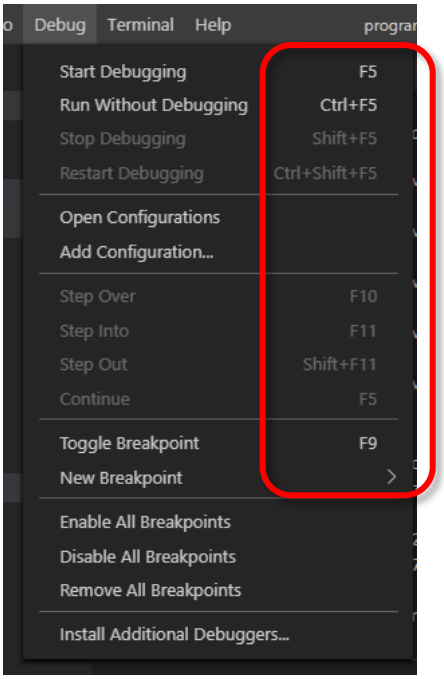
7.11. Debug Related Keyboard Shortcuts

Table 23. Debug Related Keyboard Shortcuts

Key	Action
F5	Start debugging
Shift+F5	Stop debugging
Ctrl+Shift+F5	Restart debugging
Ctrl+Shift+D	Open debugging
Ctrl+Shift+Y	Open the debug console
F9	Toggle breakpoints
Shift+F9	Continue
F5	Pause
F6	Step Over
F10	Step Into
F11	Step Out
Shift+F11	Start debugging

NOTES

You can see shortcuts on the right side of the each menu.



© 2024 ROHM Co., Ltd.

7.12. Memory Window

You can monitor memory values and peripheral registers, and you can change their values.

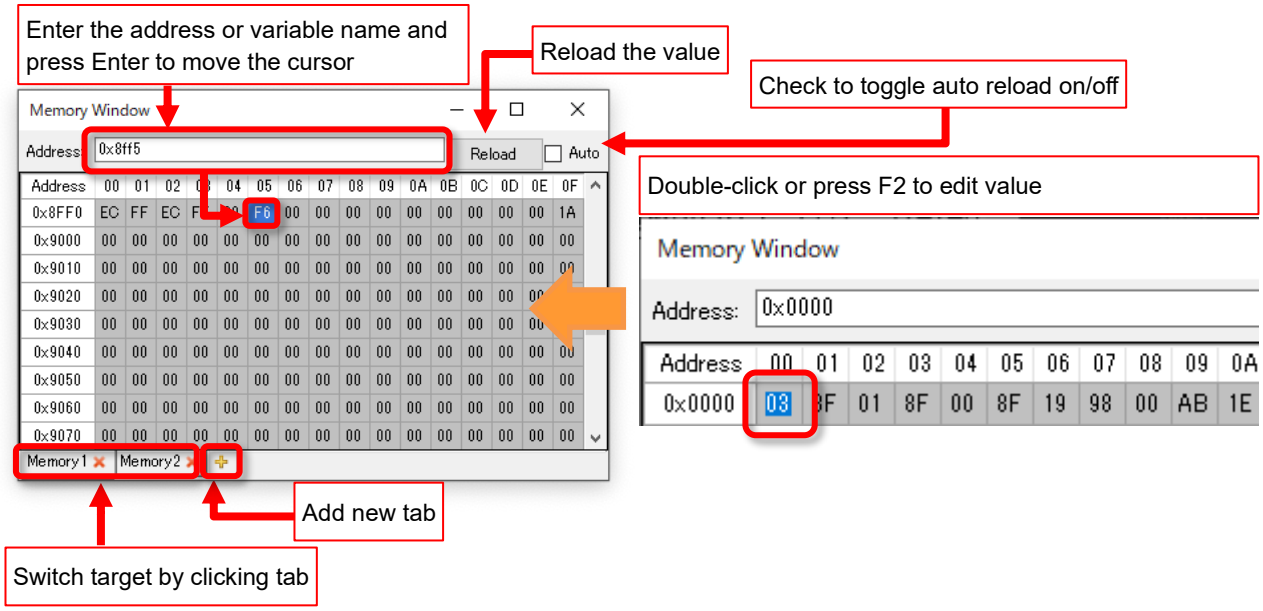
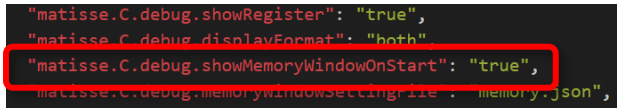
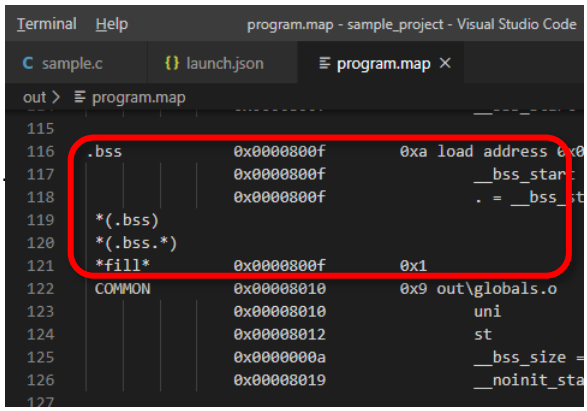


Figure 46. Memory Window





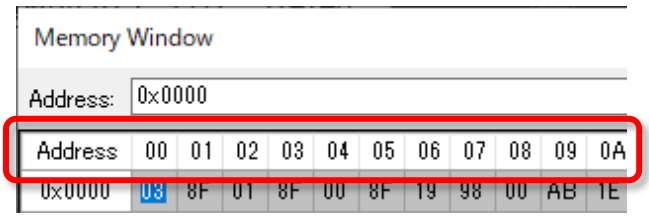


Figure 47. Memory Window Display Settings

Memory Window Display Settings

If "matisse.C.debug.showMemoryWindowOnStart" in settings.json is "true", the "Memory Window" will be shown when debugging starts.

See "Project Configuration and Setting Items".

NOTES

You can check the addresses of the global variables in the MAP file. The address of the global variables are written in .data area or .bss area. The static variables are not displayed in the MAP file.

Notes

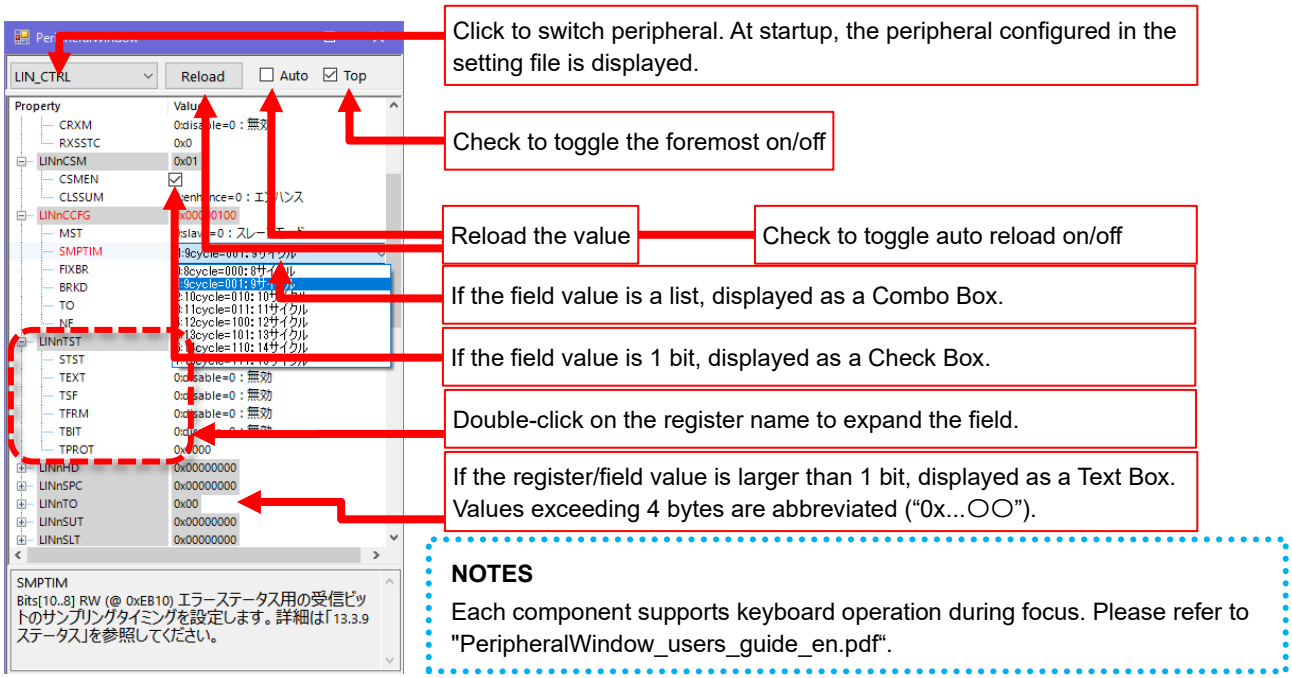
Do not click on column header (the area where displayed as "Address 00 01 .. 0F"). The memory display may be corrupted.

7.13. Peripheral Window

The Peripheral window displays information about the peripheral registers. You can monitor or change the property values in three different ways. The window display is based on the XML file set in the settings file (settings.ini)

NOTES

Setting File：A .ini file that set the peripheral name selected on startup and XML file path.
XML File：A .xml file that describes the peripheral information output by RapidMaker™.



NOTES

Each component supports keyboard operation during focus. Please refer to "PeripheralWindow_users_guide_en.pdf".

Figure 48. Peripheral Window

settings.json

```
"matisse.C.build.additionalLinkerOptions": [
],
"matisse.C.debug.showMemoryWindowOnStart": "true",
"matisse.C.debug.showPeripheralWindowOnStart": "true",
"files.associations": {
    "*.h": "c",
```

Figure 49. Peripheral Window Display Settings

Peripheral Window Display Settings

If "matisse.C.debug.showPeripheralWindowOnStart" in settings.json is "true", the "Peripheral Window" will be shown when debugging starts.

See "Project Configuration and Setting Items".

.ini File contents

Edit .ini File in the following formats.

.ini File

```
[Peripheral]
SelectedPeripheral=APB_GPIO
[FilePath]
XMLFilePath=".¥Matisse_svd.xml"
```

Table 24. ini File Format Description

Setting Item	Description
[Peripheral] SelectedPeripheral=	Describe the peripheral name to be displayed when Peripheral Window is started. If nothing is selected, the first peripheral will be displayed. The peripheral name displayed when Peripheral Window was closed will be filled in.
[FilePath] XMLFilePath=	Set the XML file 's path in which the peripheral information is written in CMSIS-SVD format. ⚠**Notes** If this path is incorrect, Peripheral Window will not be started.

© 2024 ROHM Co., Ltd.

7.14. Function call history on CPU resetting

When the CPU is reset during debugging due to WDT or hardware failure etc, this feature allows you to check which function was executed when the reset occurred.
If you set "matisse.C.debug.showBacktraceOnReset" in settings.json to "true" and run the build task, the function call history will be displayed at CPU resetting during debugging.
The function call history will be displayed in the "DEBUG CONSOLE". After the history is displayed, debugging will be automatically terminated.

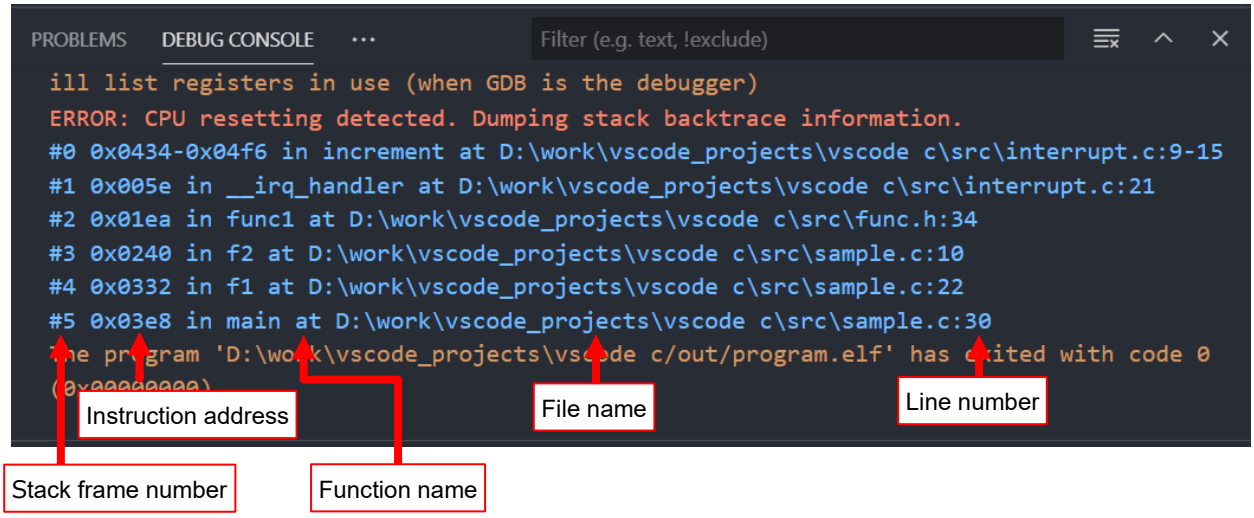


Figure 50. Display Function Call Stack at reset

Table 25. Call Stack Format Description

Field	Description
Stack frame number	0 is the function that was being executed at CPU resetting. Followed by the caller functions.
Instruction address	The address of the instruction being executed. **Notes** Since the program counter is cleared at CPU resetting, the instruction address is not fixed for functions with a stack frame number 0. Therefore, the address range is displayed.
Function name	The function name being executed.
File name	The file name being executed.
Line number	The line number of the file being executed. **Notes** Since the program counter is cleared at CPU resetting, the line number is not fixed for functions with a stack frame number of 0. Therefore, the line number range is displayed.

NOTES

- When this functionality is enabled, the instructions for displaying the function call history will be automatically added to the program. The ROM size becomes larger and the program becomes slower.
- This functionality restores the function call history using the data left on the stack. If the stack data is corrupted, it will not work properly.
- After detecting a reset by this function, clear the RAM (stack data); if a reset is detected twice in a row without clearing the RAM, the function call history may not be displayed correctly.
- Inline functions may not appear in the call history. If compiler optimization is enabled, even functions without the inline keyword will be subject to inlining.

© 2024 ROHM Co., Ltd.

8. Command Execution

8.1. Displaying the Command Palette and Command Input

Type "Ctrl + Shift + P" or F1 from the keyboard to display the Command Palette at the top of the screen.
From here, you have access to various functionality of VS Code.

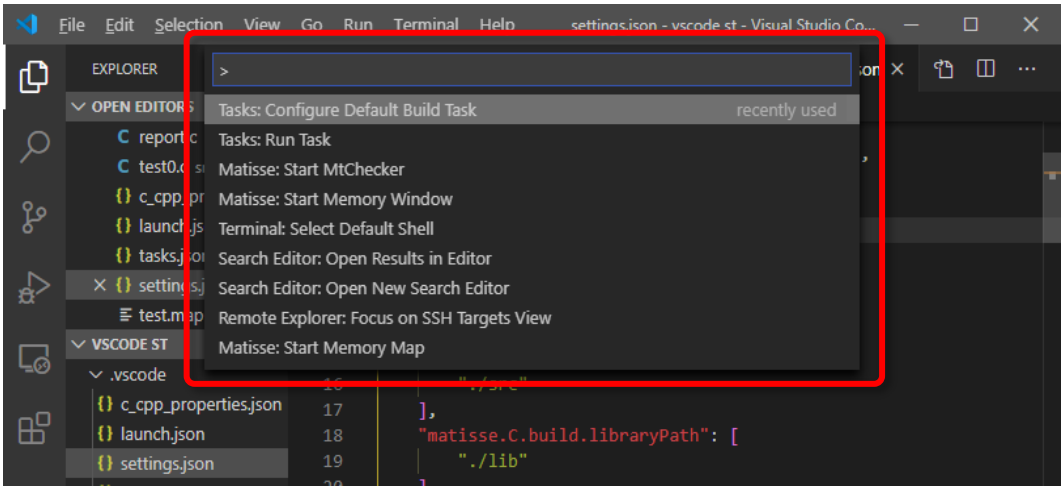


Figure 51. Command Palette Display and Command Input

■The matiseye™-studio adds the following commands to VS Code

Table 26. Command Palette Description

Command	Description
Matisse Debug: Start Memory Window	Startup the Memory Window. See "Memory Window". **Notes** This command is only available during debugging.
Matisse Debug: Start Peripheral Window	Startup the Peripheral Window. See "Peripheral Window". **Notes** This command is only available during debugging.
Matisse Debug: Start MtChecker	Startup the MtChecker (Development Environment Configuration Checker).
Tasks: Run Task	The following tasks which are defined in tasks.json can be executed. • Build: Perform build. Same as Ctrl + Shift + B. • Clean: Delete all files generated by the build task. • Rebuild: Run the Clean task and Build task.
Matisse Analysis: Start Stack Analysis	Startup the Stack View. See "Stack Static Analysis".
Matisse Debug: Start Performance View	Startup the Performance View. Usable only during debugging.

© 2024 ROHM Co., Ltd.

9. Frequently asked questions

The functions ei() and di() defined in matisse/interrupt. h display error squiggly lines.
Is there a way to erase it?

Answer

In "matisse.C.build.compilerPath" of the configuration file (settings.json), set the full path of mtcc (default setting is C:/Program Files/ROHM/Matisse/C/bin/mtcc.exe).

I do not know how to set the library path where the library file is located.

Answer

For example, if you want to add a folder called "AAA" to the library path, add the setting "AAA" to "matisse.C.build.libraryPath".

I do not know how to link the library files.

Answer

For example, if you want to link a library file called "libAAA.a", place "libAAA.a" in the folder where the library path is set and add the setting "libAAA.a" to "matisse.C.build.libraryFiles".

Notes

mtcc supports only static links. You cannot link dynamic library files (*.so) .

I wrote macro definitions in "matisse.C.build.preprocessorDefinitions" of the configuration file (settings.json), but they are not reflected in the source code.

Answer

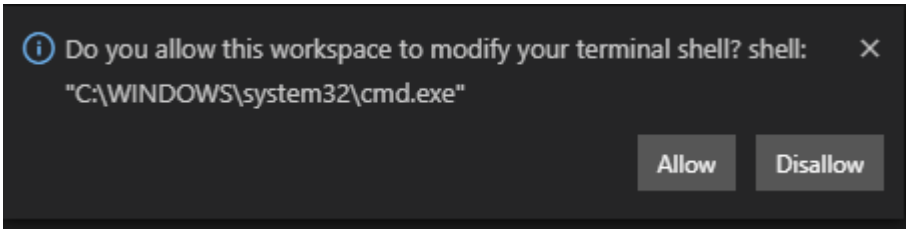
Please run the build task (Ctrl + Shift + b) after editing the configuration file (settings.json). In the current development environment, the contents of the settings.json will not be reflected in the source code until you run the build task.

Notes

If you want to define function macros, please describe them in your header files, not in the configuration file (settings.json)

© 2024 ROHM Co., Ltd.

During the build, a pop-up like a following image is displayed in the lower right corner of the screen. What do I need to do?



Answer

Press "Allow". If you accidentally clicked a "Disallow",
Follow these steps:

Press the F1 key and enter "Select Default Shell".
Press the Enter key.
Select "Command Prompt" from the displayed options and press the Enter key.

Is there a way to delete unused functions?

Answer

Add "-ffunction-sections" to "matisse.C.build.additionalCompileOptions" and "--gc-sections" to "matisse.C.build.additionalLinkerOptions" to "matisse.C.build.additionalLinkerOptions"in the settings file (settings.json). This will prevent unused functions from being included in the build output file.

⚠ Notes

This setting disables debug functionality. Please do not perform this setting when debugging is required, but only when creating output files for release.

© 2024 ROHM Co., Ltd.

10. Shortcut Key List

10.1. General

Table 27. Shortcut Key List (General)

Shortcut Key	Action
Ctrl + P	Quick Open
Ctrl + Shift + N	New window
Ctrl + W	Close Tab
Ctrl + Shift + W	Close Window

© 2024 ROHM Co., Ltd.

10.2. File Management

Table 28. Shortcut Key List (File Management)

Shortcut Key	Action
Ctrl + N	New File
Ctrl + O	Open File
Ctrl + S	Save
Ctrl + Shift + S	Save As...
Ctrl + K ⇨ S	Save All
Ctrl + K ⇨ P	Copy Path of Active File
Ctrl + K ⇨ R	Reveal Active File in Explorer
Ctrl + K ⇨ O	Show Active File in New Window

© 2024 ROHM Co., Ltd.

10.3. Editor Management

Table 29. Shortcut Key List (Editor Management)

Shortcut Key	Action
Ctrl + \	Split Editor
Ctrl + F4	Close Editor
Ctrl + Shift + T	Reopen Closed Editor
Ctrl + PgUp	Open Left Editor
Ctrl + PgDn	Open Right Editor
Ctrl + 1 / 2 / 3	Focus into 1st, 2nd, 3rd Editor Group
Ctrl + Tab	Open Next in Current Editor Group
Ctrl + Shift + Tab	Open Previous in Current Editor Group

© 2024 ROHM Co., Ltd.

10.4. Editing

Table 30. Shortcut Key List (Editing)

Shortcut Key	Action
Ctrl + C	Copy Line
Ctrl + X	Cut Line
Ctrl + Shift + \	Jump to Matching Bracket
Ctrl +] OR Tab	Indent Line
Ctrl + [OR Shift + Tab	Outdent Line
Ctrl + Home	Go to Beginning of Line
Ctrl + End	Go to End of Line
Ctrl + /	Toggle Line Comment
Ctrl + Shift + [	Fold Region
Ctrl + Shift +]	Unfold Region
Ctrl + Alt + Mouse	Box Selection

© 2024 ROHM Co., Ltd.

10.5. Search and Replace

Table 31. Shortcut Key List (Search and Replace)

Shortcut Key	Action
Ctrl + F	Find
Ctrl + H	Replace
F3	Find Next
Ctrl + F3	Find Previous

© 2024 ROHM Co., Ltd.

10.6. Rich Language Editing

Table 32. Shortcut Key List (Rich Language Editing)

Shortcut Key	Action
Ctrl + Space	Trigger Suggestion
Ctrl + Ait + F	Format Document
Ctrl + K ⇨ Ctrl + F	Format Selection
F12	Go to Definition
Alt + ←	Go Back
Alt + →	Go Next
Alt + F12	Peek Definition
Ctrl + K ⇨ F12	Open Definition to the Side
Shift + F12	Show References
Shift + Ait + F12	Find All References
F2	Rename Symbol
Ctrl + G	Go to Line
Ctrl + P	Go to File

© 2024 ROHM Co., Ltd.

10.7. Display

Table 33. Shortcut Key List (Display)

Shortcut Key	Action
F11	Toggle Full Screen
Ctrl + +	Zoom in
Ctrl + -	Zoom out
Ctrl + B	Toggle Sidebar Visibility
Ctrl + Shift + E	Show Explorer
Ctrl + Shift + F	Show Search
Ctrl + Shift + D	Show Debug

© 2024 ROHM Co., Ltd.

10.8. Debug

Table 34. Shortcut Key List (Debug)

Shortcut Key	Action
F5	Start Debugging
Shift + F5	Stop Debugging
Ctrl + Shift + F5	Restart Debugging
F11	Step in
Shift + F11	Step out
F10	Step over
F9	Toggle Breakpoint
F6	Pause Debugging

© 2024 ROHM Co., Ltd.

11. Open-source software licenses

This software includes open-source software (hereinafter referred to as "open-source software program") provided under the following license conditions, in addition to software for which ROHM owns or is licensed.
Open-source software programs are subject to their respective license terms, so in the event of a conflict between the license terms of an open-source software program and this material, the license terms of the open-source software program shall prevail.

Included open-source software and their license terms.
・glob(ISC)
・gulp(The MIT License)
・CSV Writer(The MIT License)
・@aduh95/viz.js (The MIT License)
・jsonc-parser(The MIT License)
・find-process(The MIT License)
・loadash(The MIT License)
・node-xml2js(The MIT License)
・async-mutex(The MIT License)
・date-fns(The MIT License)
・sharp(Apache-2.0)

12. Trademark notices

"Windows" and "VS Code" are trademarks of Microsoft Group companies.
"Intel" is a trademark of Intel Corporation or its subsidiaries.
"Core™" is a trademark or registered trademark of Intel Corporation or its subsidiaries.
“tinyMicon MatisseCORE™” and “matiseye™” are a trademark or registered trademark of ROHM Corporation.

© 2024 ROHM Co., Ltd.

Caution

1. The information written in this material regarding the software and system (hereinafter collectively “Software”) and the contents written in this material are the ones at the time of this material’ s issuance, and may be changed without prior notice.

2. In case the customer considers using the Product or the Software for the equipment or device (such as the medical equipment, transportation equipment, traffic equipment, aerospace equipment, nuclear power control equipment, the vehicle equipment including the fuel control system and/or car accessories, and/or the various kinds of safety devices etc.) which require extremely high reliability, and whose breakdown or malfunction relate to the risk or damage to the person’ s life or body, or any other serious damage (such usage is hereinafter called “Special Usage”), please consult with the ROHM’ s sales counter. In relation to the damage etc. caused by using for Special Usage without ROHM’ s prior written approval, incurred by the customer or a third party, ROHM shall not owe any responsibility.

3. Although ROHM always strives to improve the quality and reliability, the semiconductor products may break or malfunction caused by various factors. Should the Software and ROHM products using the Software (such ROHM products are hereinafter called “Product”) break or malfunction, please secure the safety, including but not limited to derating, reductant design, fire spread prevention, backup, and/or fail safe etc. in order to prevent the accident resulting in injury or death and/or fire damage etc. as the influence of such break or malfunction. In case of the use beyond the rated value, or the non-compliance with precaution for use, ROHM shall not owe any responsibility.

4. ROHM shall not owe any responsibility for any direct and/or indirect damage to the customers (including the damage caused by loss of intangible asset such as information, data, or program etc., loss and/or interruption of profit) which is caused by the use or impossibility to use of the Software.

5. Since the Software, this material, and/or the Product contain the ROHM’ s confidential information, technical information, and/or trade secret,
   ROHM definitely declines the following acts in whole or part without ROHM’ s prior written approval:
   (i) disclosing to a third party;
   (ii) disassemble, reverse engineering, and/or any other analysis;
   (iii) reprinting, copy, and/or reproduction; or
   (iv) removing the copyright notice included in the Software.

6. When exporting or providing overseas the software, the technology written in this material, please copy with the export related lows and relations such as “Foreign Exchange and Foreign Tarade Act” and/or “Export Administration Regulations (EAR)” , and please follow the necessary procedures according to these rules.

7. ROHM shall not guarantee that, regarding the Software and/or the information written in this material, there is no infringement of the third parties’ intellectual property rights and any other rights. Accordingly, ROHM shall not owe any responsibility for such infringement of rights.

8. ROHM shall not, regarding the information written in this material, license to the customers by this material expressly nor impliedly the implementation or utilization of the intellectual property rights or any other rights owned or managed by ROHM or a third party.

9. The customers shall guarantee to exempt ROHM and ROHM’ s officers and/or employees from responsibility, and hold them harmless, and protect them from any damage, from the loss, penalty, or cost caused by the claim of damage compensation (including but not limited to the attorney fees) resulting from, or incurred relating to the following customer’ s acts:
   (1) the infringement of the third party’ s rights or the law offence caused by reading, download, encryption,summarization, copy, or transfer etc.; or
   (2) violation of this material.

10. Although ROHM has made this material aiming at accuracy and carefully, ROHM shall not guarantee that there is no error. Should the error of the contents written in this material cause the damages of the customers or a third party, ROHM shall not owe any responsibility.

Thank you for using ROHM products.
For inquiries about our products, please contact us.

ROHM Customer Support System

http://www.rohm.co.jp/contact/

© 2024 ROHM Co., Ltd. All rights reserved.