tinyMicon MatisseCORE™

matiseye™-studio User's Guide

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

No. 67UG023E Rev.003
June 2024

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

Table of Contents

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.

● File List

Table 1. matiseye™-studio File List

MatisseCCompiler-*.*.*.exe	Installer for C programming tools for Matisse.
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.
LED_Timer_Int_C.zip	A sample C project for timer interrupt and LED blinking.

● Prerequisite

OS: Windows 7 32-bit / Windows 7 64-bit / Windows 10 32-bit / Windows 10 64-bit
CPU: Comparable performance to Intel Core line
Memory: 2GByte or more
HDD: 1GByte or more free space
.Net Framework: Version 4.7.2

● 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.

2 Installation Procedure

Notes

If .Net Framework is not installed on your computer, install Net Framework 4.7.2 first.

2.1 Installing C Programming Tools for Matisse

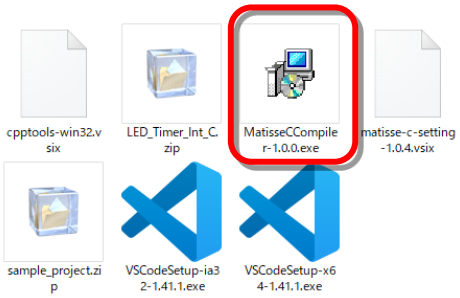


Figure 1. Run MatisseCCompiler Installer

1 Double-click "MatisseCCompiler-*.*.*.exe" to run the installer.

NOTES

If User Account Control appears, click Yes.

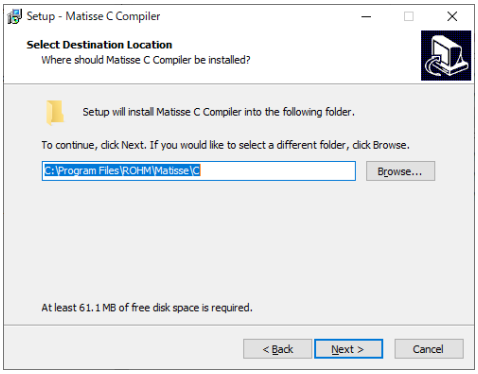


Figure 2. MatisseCCompiler Installation Window

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 The following installers are also run automatically during the installation of the C language compiler.

- 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 Restart your PC

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.

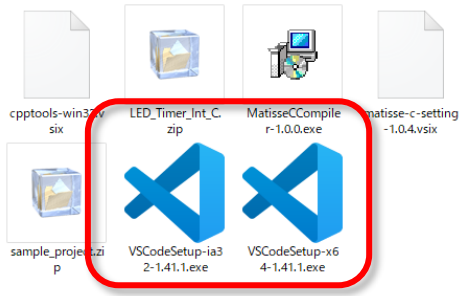


Figure 3. Run VS Code Installer

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".

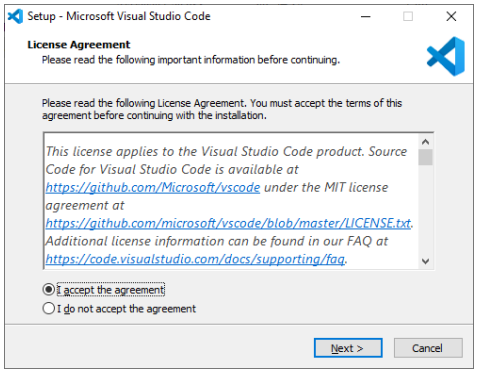


Figure 4. VS Code Installation

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.

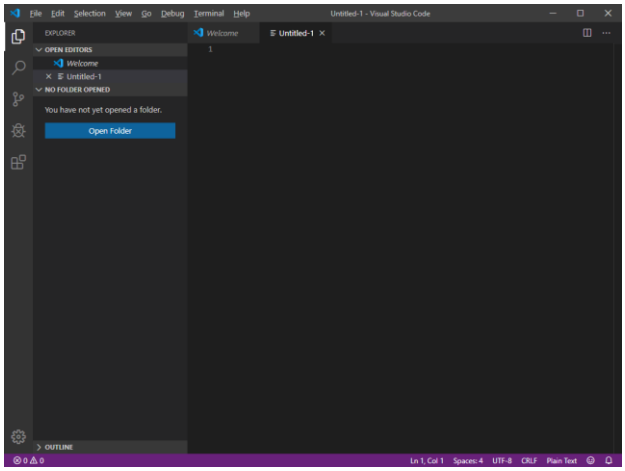


Figure 6. Start VS Code

1 Start VS Code.

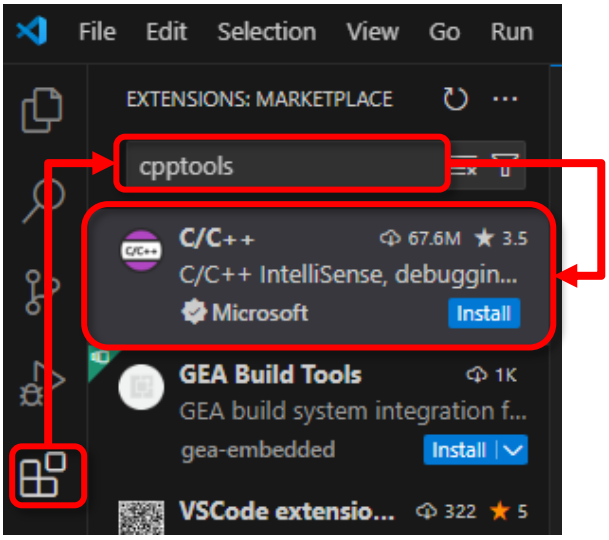


Figure 5. Search cpptools

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

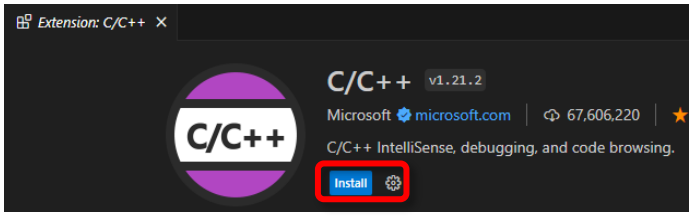


Figure 7. Install cpptools

3 Click "Install" to complete the installation.

4 Restart VS Code.

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.

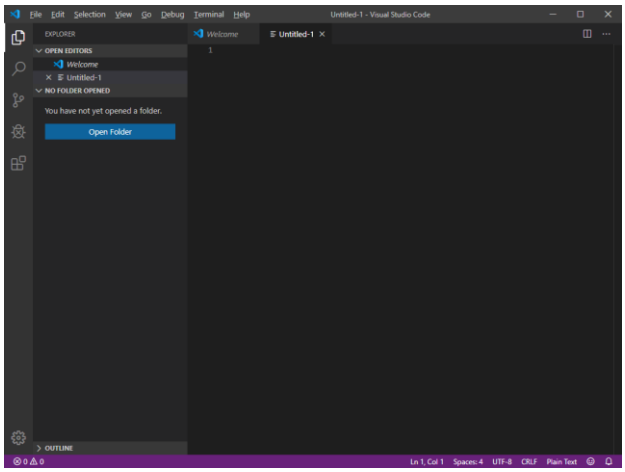


Figure 8. Start VS Code

1 Start VS Code.

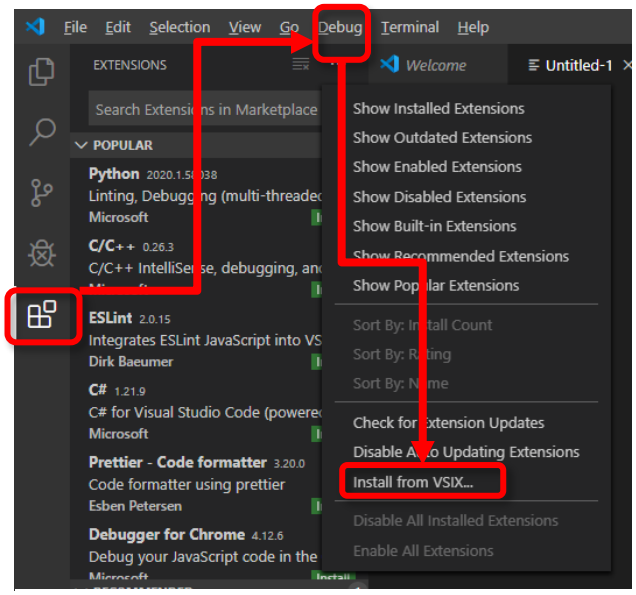


Figure 9. Click Install from VSIX

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

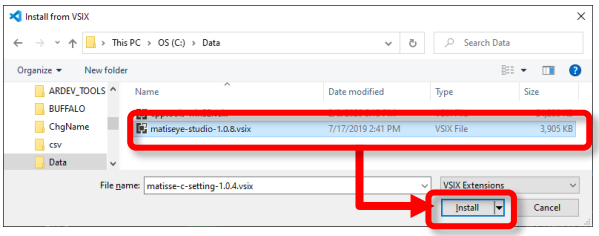


Figure 10. Install matiseye-studio_*.*.*.vsix

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

4 Restart VS Code.

3 How to use Visual Studio Code (VS Code)

3.1 VS Code Window Description

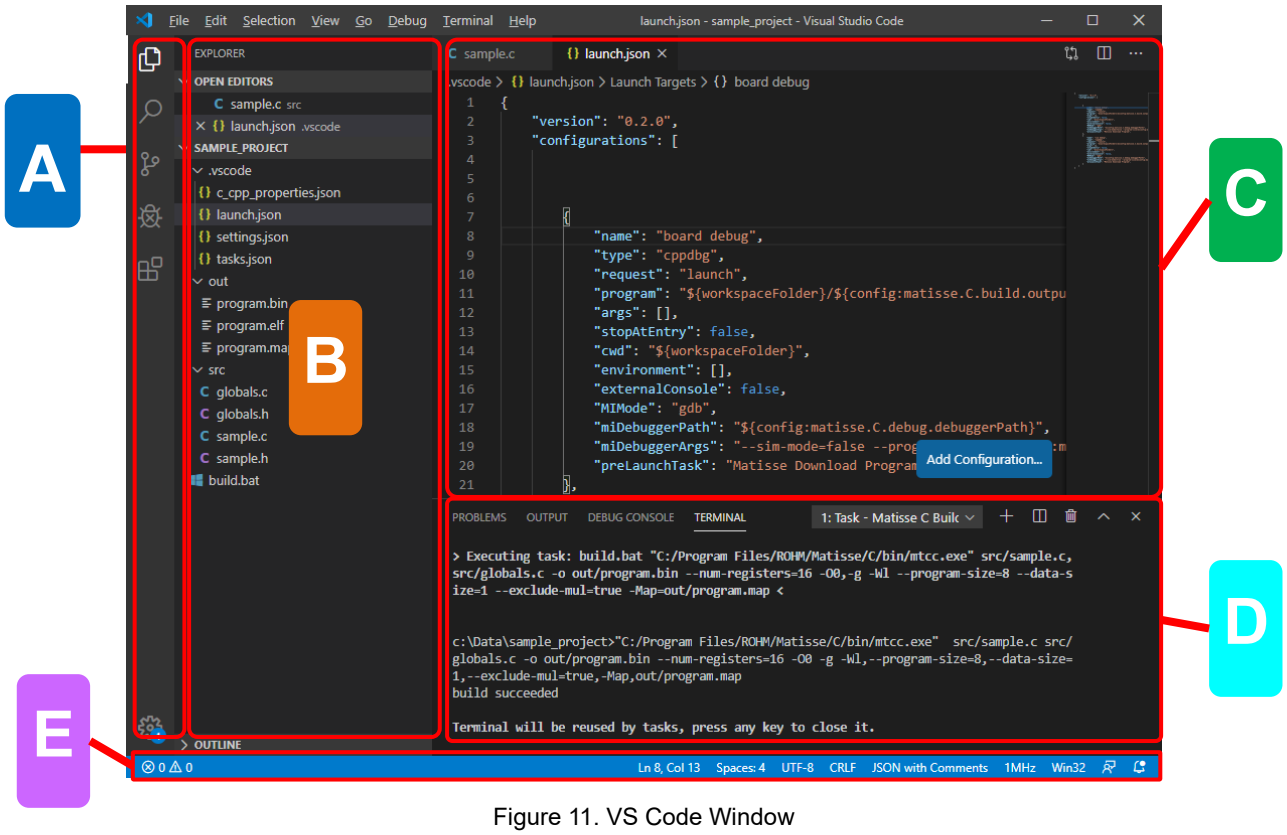


Figure 11. VS Code Window

Table 2. 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.

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.

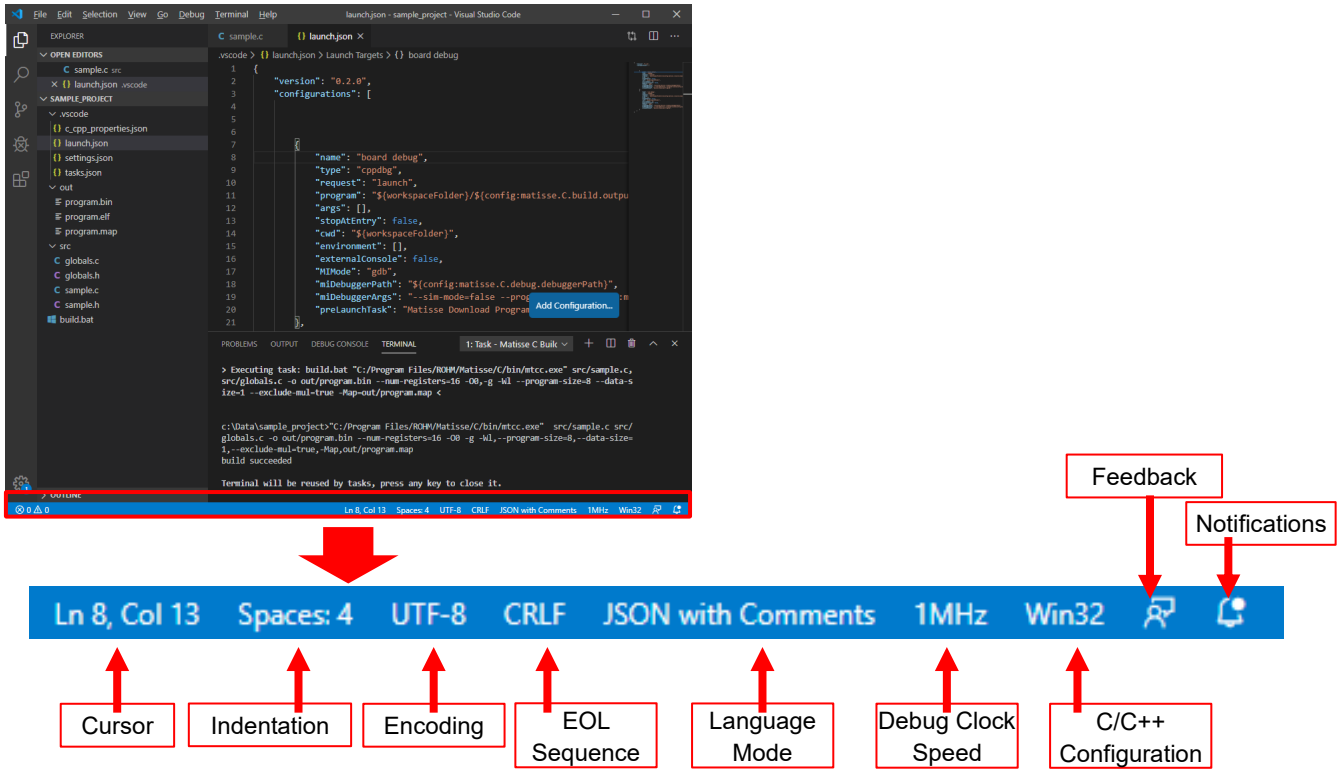


Figure 12. VS Code Status Bar

Table 3. 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 clock speed of the debug interface when debugging with the Matisse development board. Click to change it. If you change it during debugging, the change will not be reflected immediately, but will be reflected from the next debug session.
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.

3.3 Operating VS Code

Use sample_project to learn VS Code.

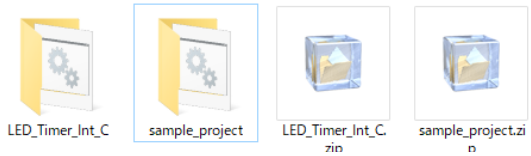


Figure 13. VS Code sample_project

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.

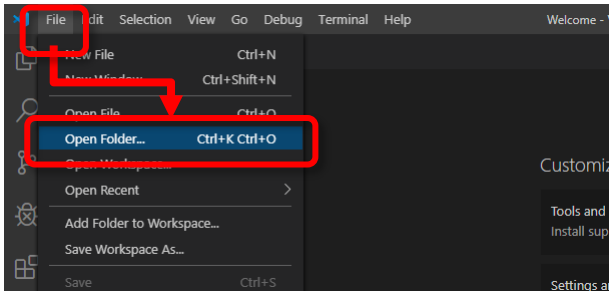


Figure 14. Click Open Folder

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.

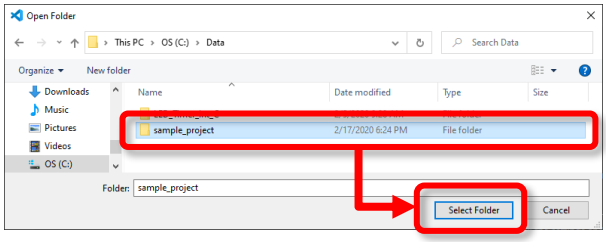


Figure 15. Select sample_project Folder

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.

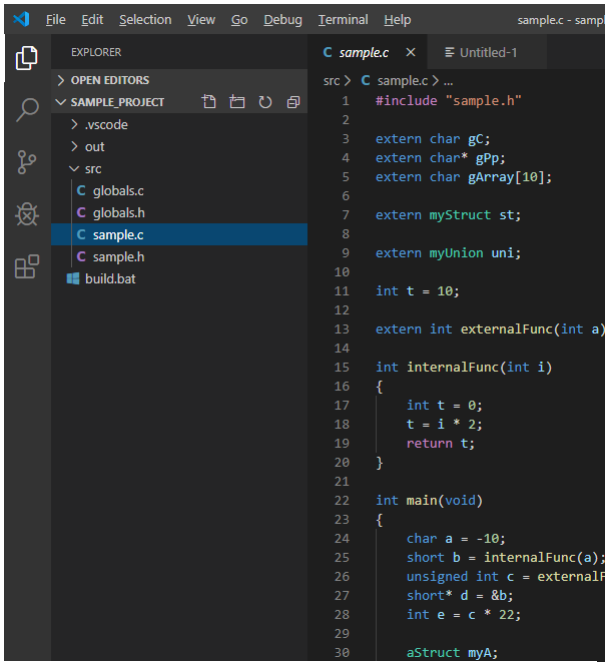


Figure 16. Edit C source file

Building

Build the data and generate an executable file.

See "How to Build".

Debug

Check the operation of the program.

See "Debug".

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".

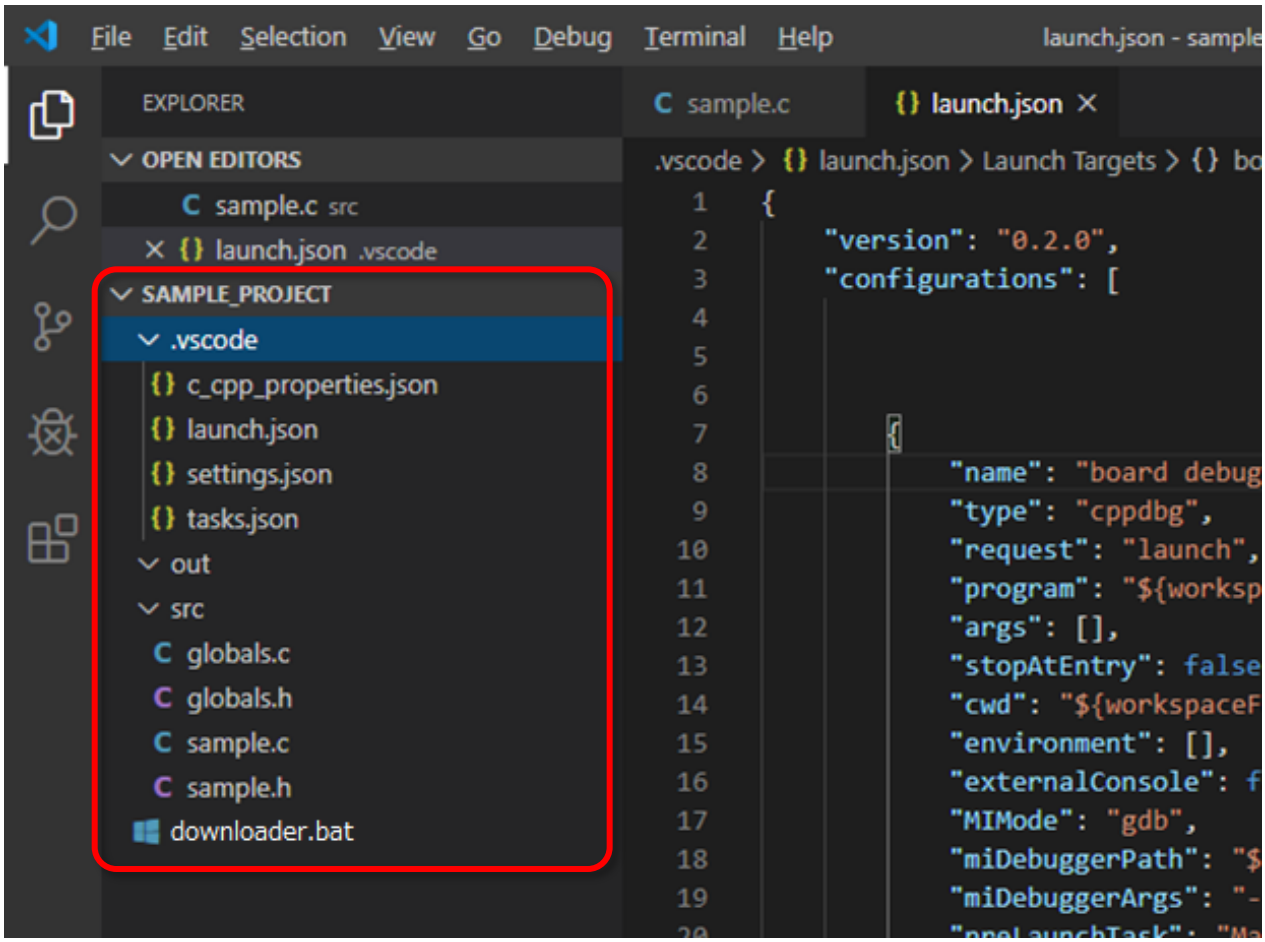


Figure 17. Samples Project Configuration

Table 4. 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.

4.2 Compiler/Debugger Setting Items（settings.json）

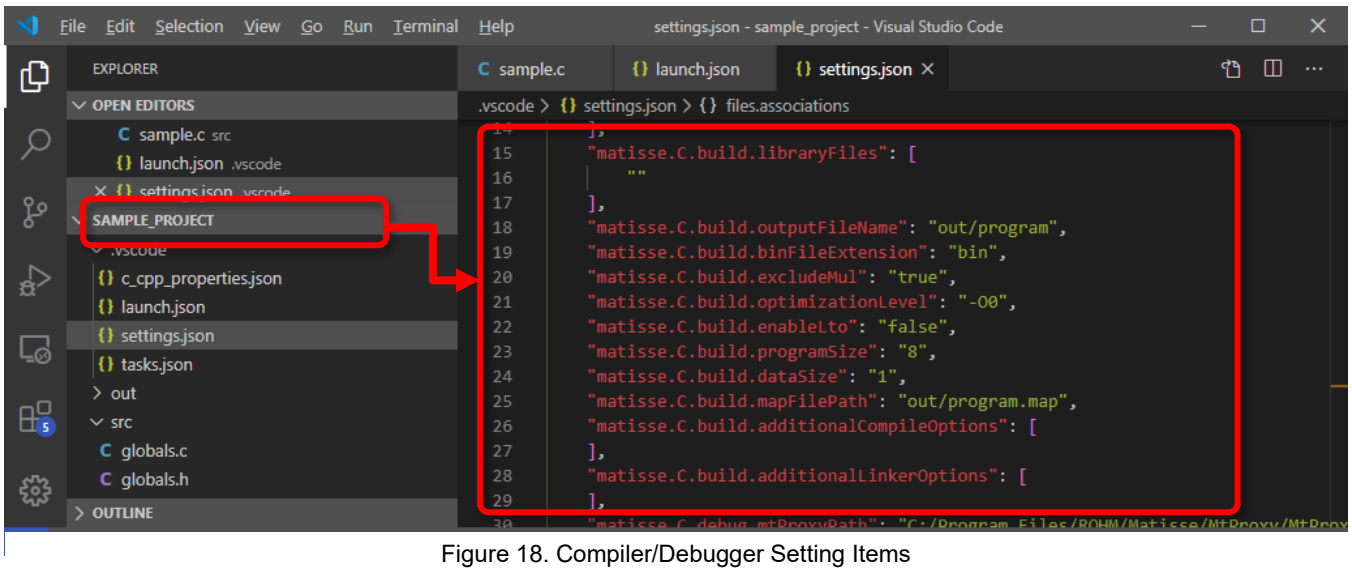


Figure 18. Compiler/Debugger Setting Items

Table 5. 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)
(continued on 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.

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. The matiseye-adapter Pro supports from 10kHz to 24MHz. The matiseye-adpter supports from 10kHz to 1.5MHz. If faster than 1.5MHz is specified, 1.5MHz will be 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.downloaderOptions	A list of program downloader options. Separated with a comma.
(Other settings don't need to be changed.)	

5 How to Build

5.1 Running Build Task and Generated Files

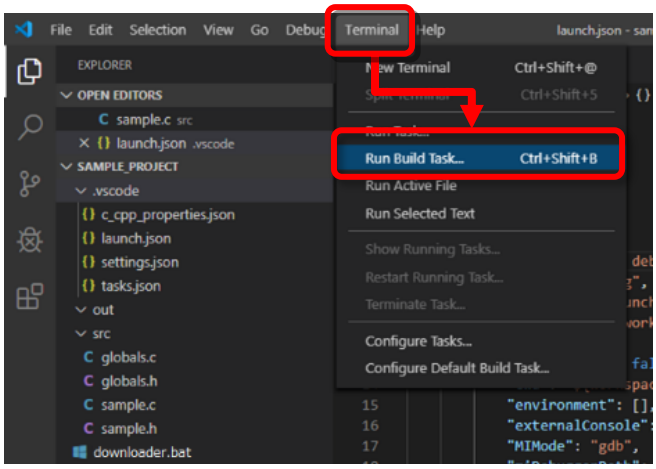


Figure 19. Execution Run Build Task

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

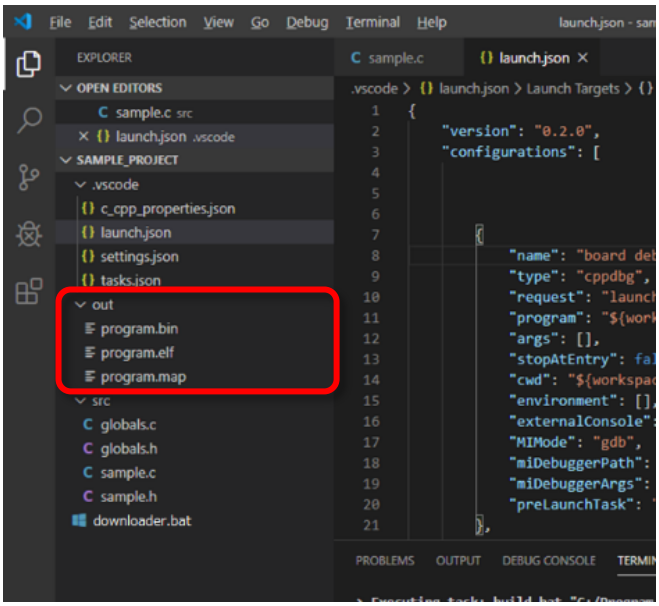


Figure 20. File Generation Run Build Task

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

Table 6. 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.

5.2 MAP File

This section describes the format of MAP file.

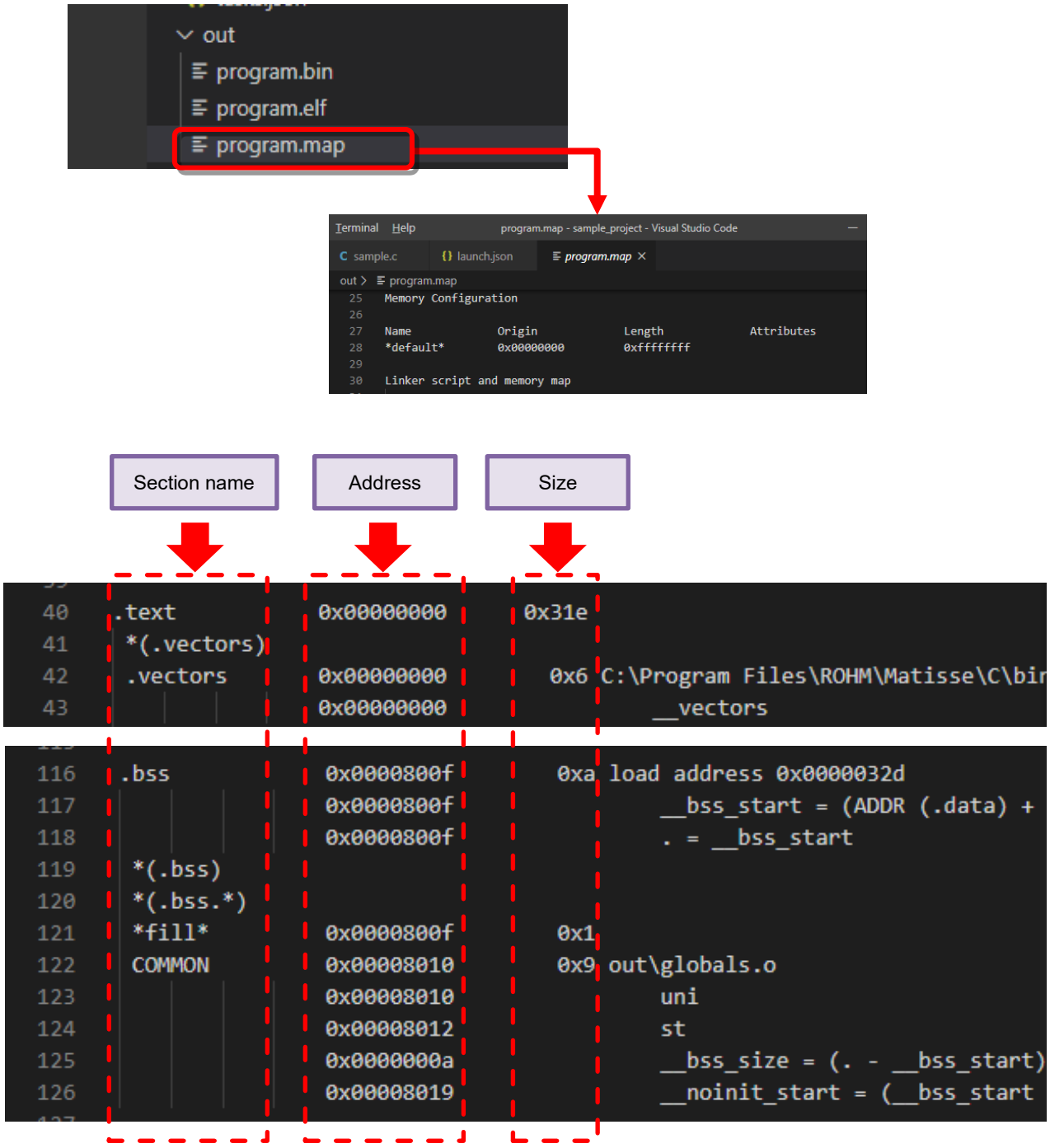


Figure 21. MAP File Configuration

NOTES

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

5.3 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.

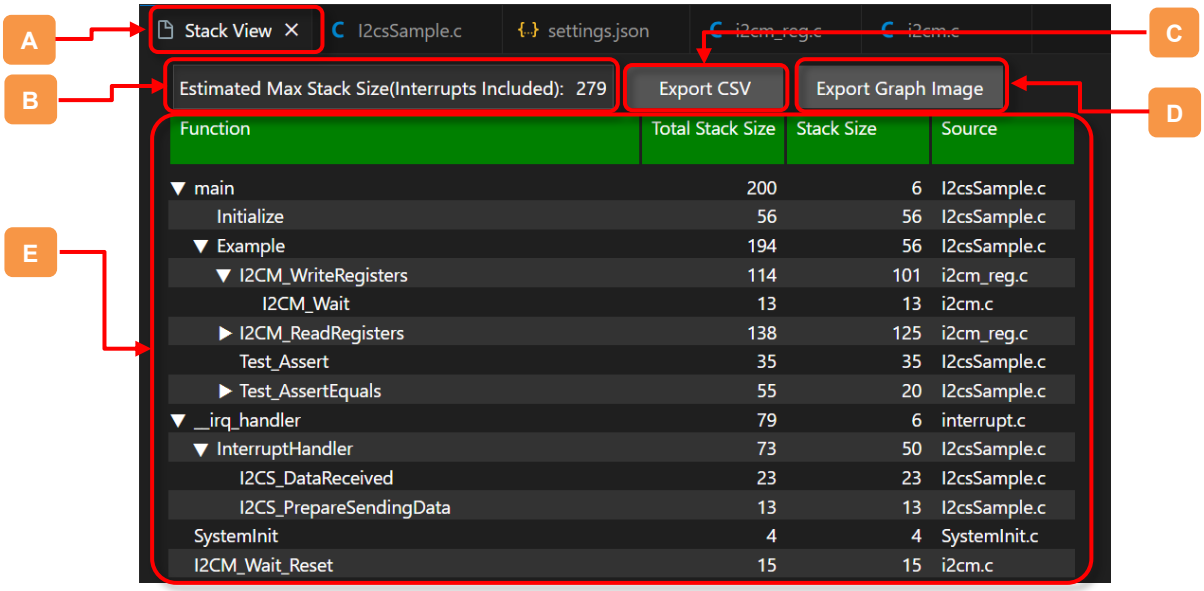


Figure 22. Stack View Window

■Start Procedure: To start the analysis, execute "Matisse Analysis: Start Stack Analysis" from the command palette. The build process will begin, and upon completion, the aforementioned tab will be displayed on the left side of the editor screen.

See "Displaying the Command Palette and Command Input".

Table 7. Stack View Window 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 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.

6 Debug

6.1 Debug Window Descriptions

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

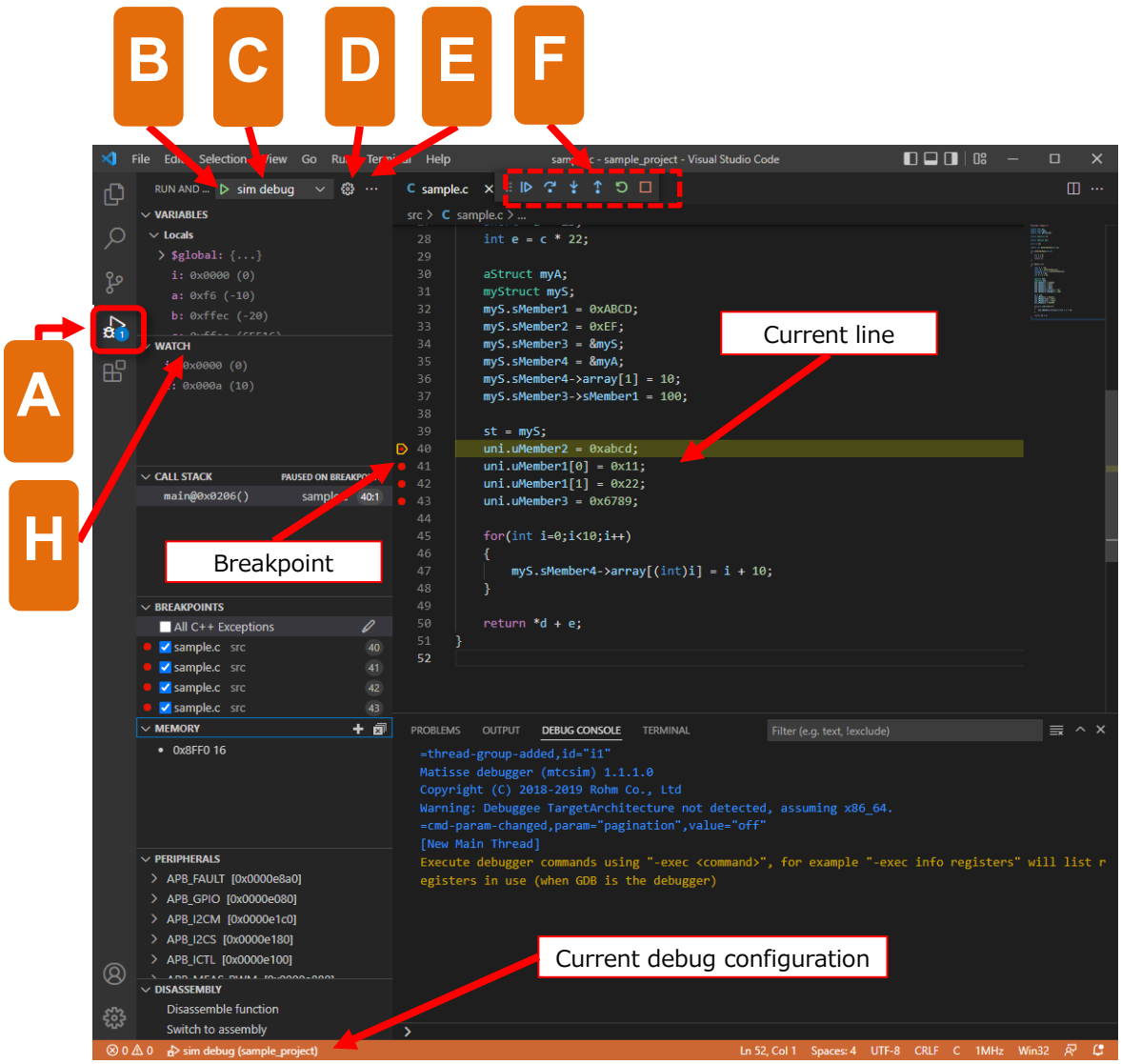


Figure 23. Debug Window when "Debug and Run" selected

Table 8. 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)	

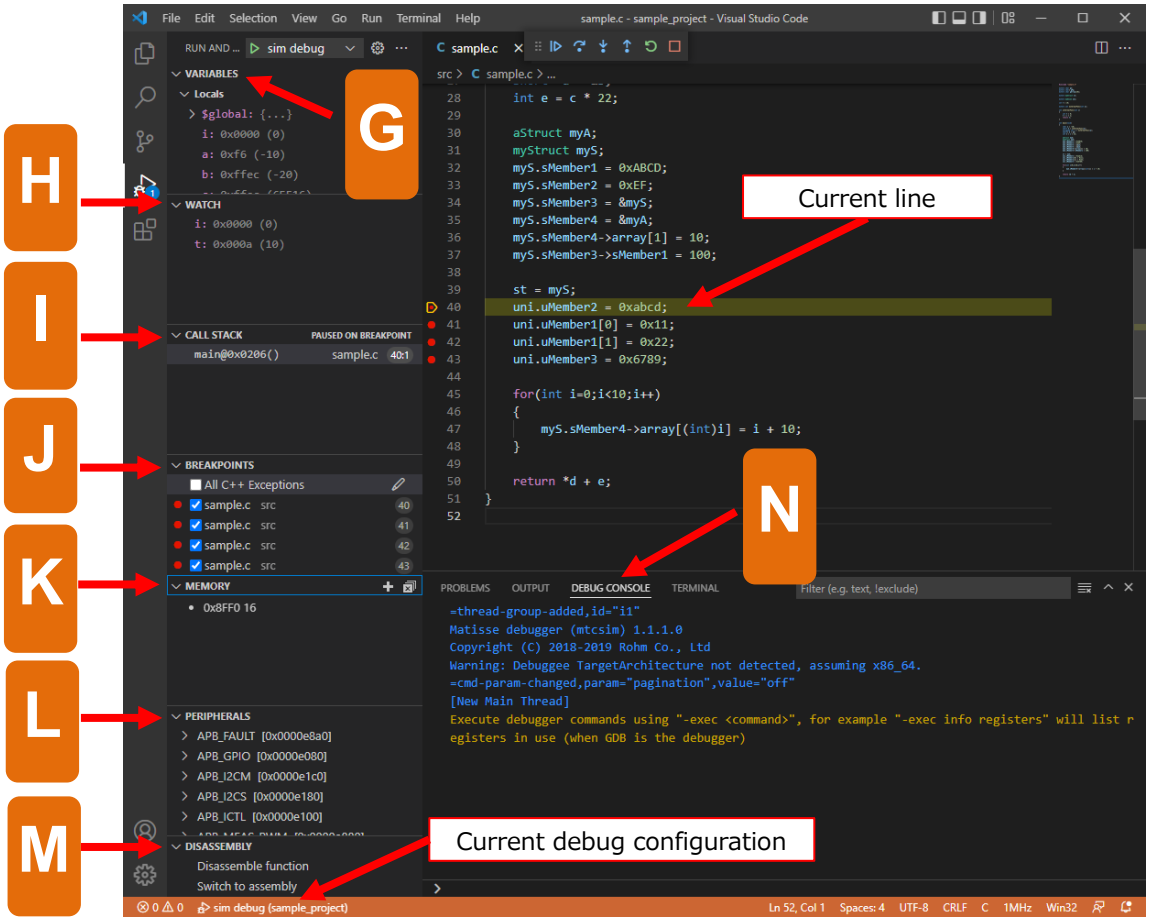


Figure 24. Debug Window when "Debug and Run" selected

Table 9. 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.

6.2 Debug Menu

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

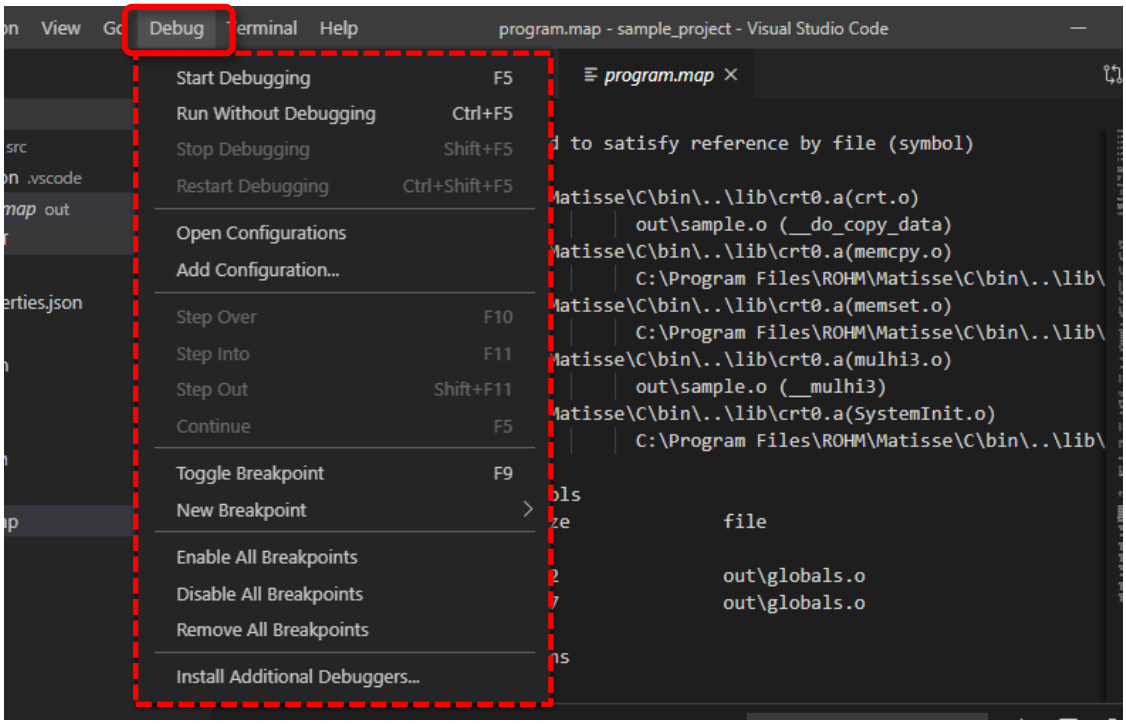


Figure 25. Debug Menu

6.3 Breakpoints

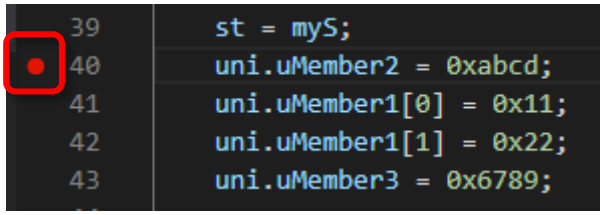


Figure 26. BREAKPOINTS

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.

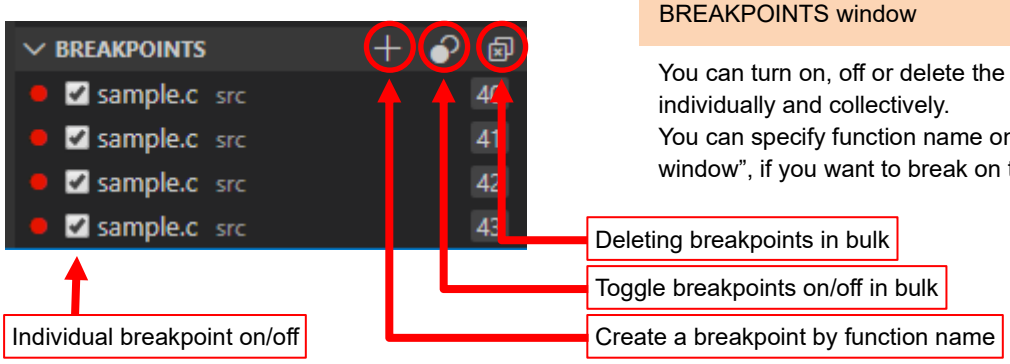


Figure 27. BREAKPOINTS Window

6.4 Debug Toolbar

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

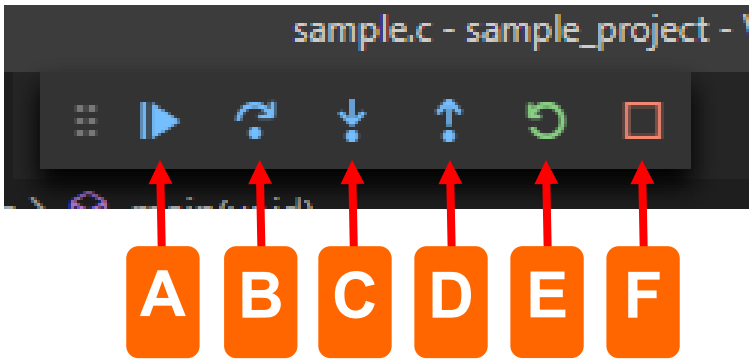


Figure 28. Debug Toolbar

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

Table 10. 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 11. Debug Toolbar Right Button Description

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

6.5 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".

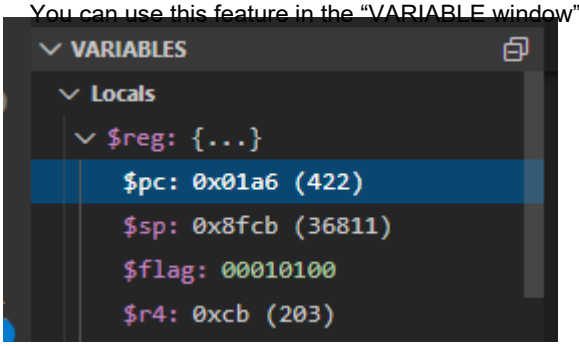


Figure 29. VARIABLES Window

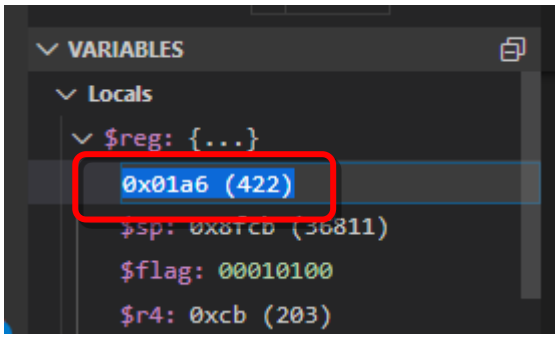


Figure 30. Editing Values

VARIABLES window

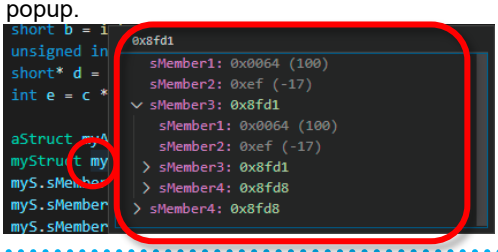
You can check and edit the value of variables.
If the variable type is a structure, pointer, or array, details of each element are displayed in a tree format.
The variables displayed in the "VARIABLE window" are local variables on the call stack.

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.

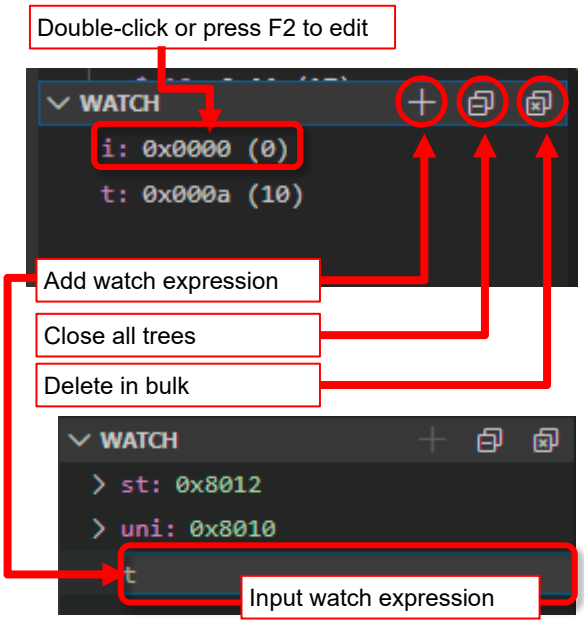


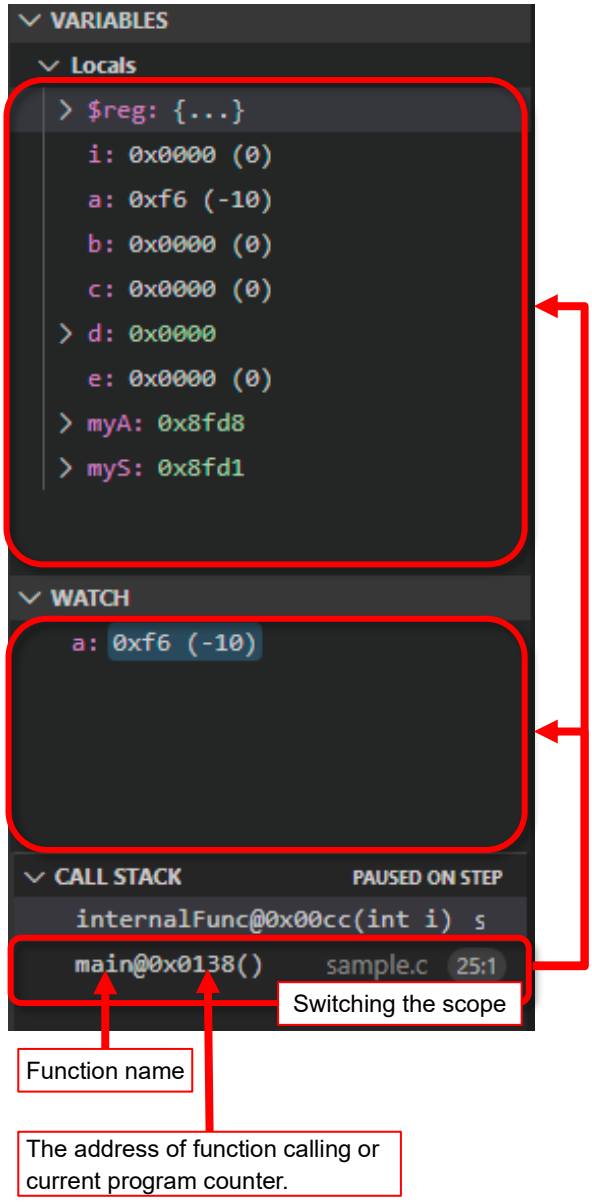
Figure 31. WATCH Window

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".

6.6 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 32. CALL STACK Window

6.7 MEMORY

MEMORY area displays the memory value of the specified range in the screen of VS Code.

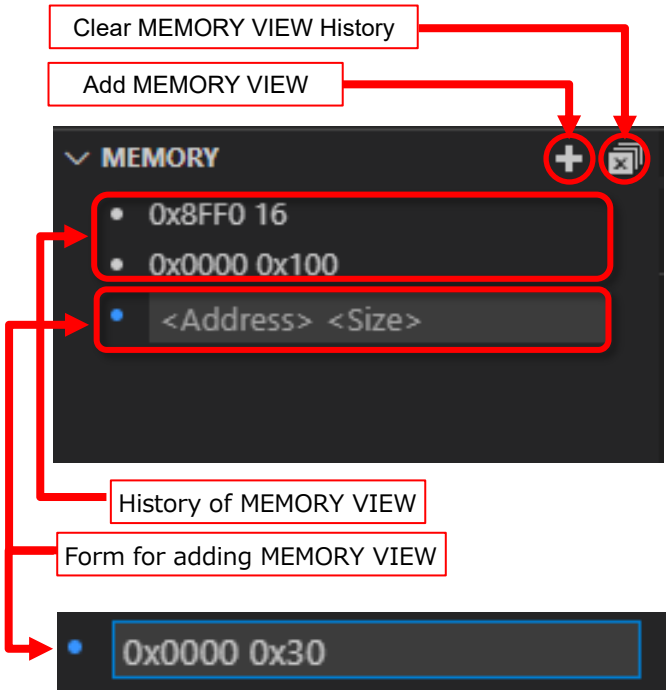


Figure 33. MEMORY VIEW

Click "Add MEMORY VIEW" to display the form for adding MEMORY VIEW.

Enter the start address and size of the memory range you want to display in the form and hit enter. The MEMORY VIEW will be newly displayed in the editor area.
You can enter both hexadecimal and decimal values in the Add New form.
It is not possible to display a range that exceeds the on-board memory area.

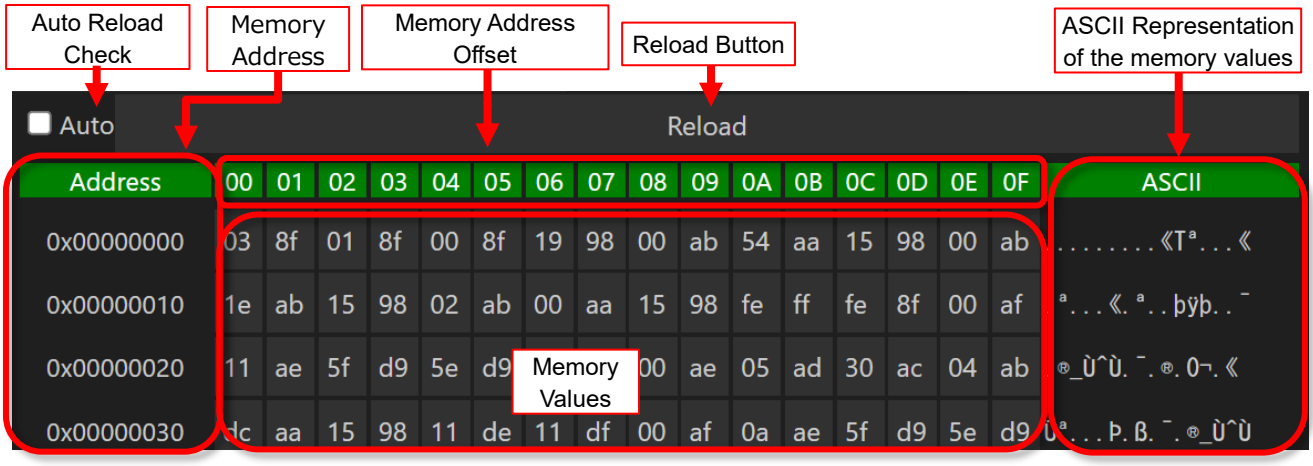
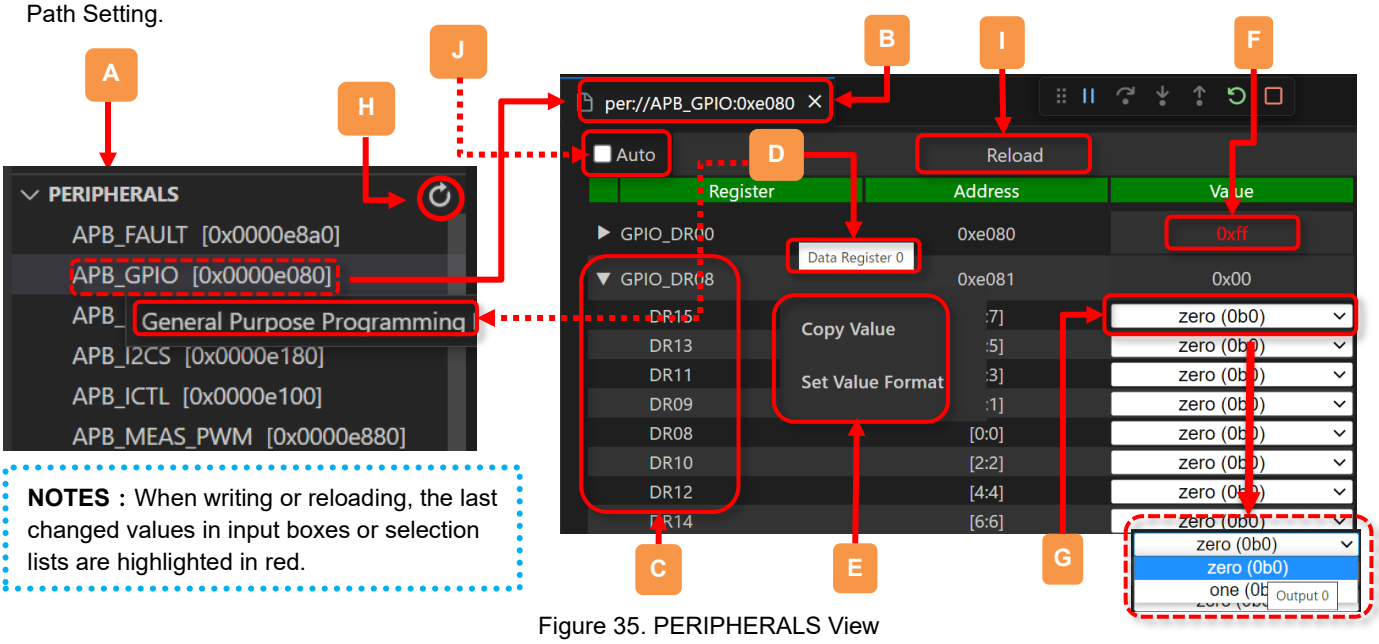


Figure 34. MEMORY VIEW Editor Area

The MEMORY VIEW displays the memory range and value as shown above.
Double-click on a memory value to edit the value.
Clicking the Reload button will reload the memory value.
When "Auto" is checked, auto auto-reload starts.
The memory value that has been changed from the previous state will be displayed in red.

6.8 PERIPHERALS

PERIPHERALS displays device info on screen for reading/writing values. Data is from CMSIS-SVD file (<XXX>) specified in File Path Setting.



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

Figure 35. PERIPHERALS View

Table 12. 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

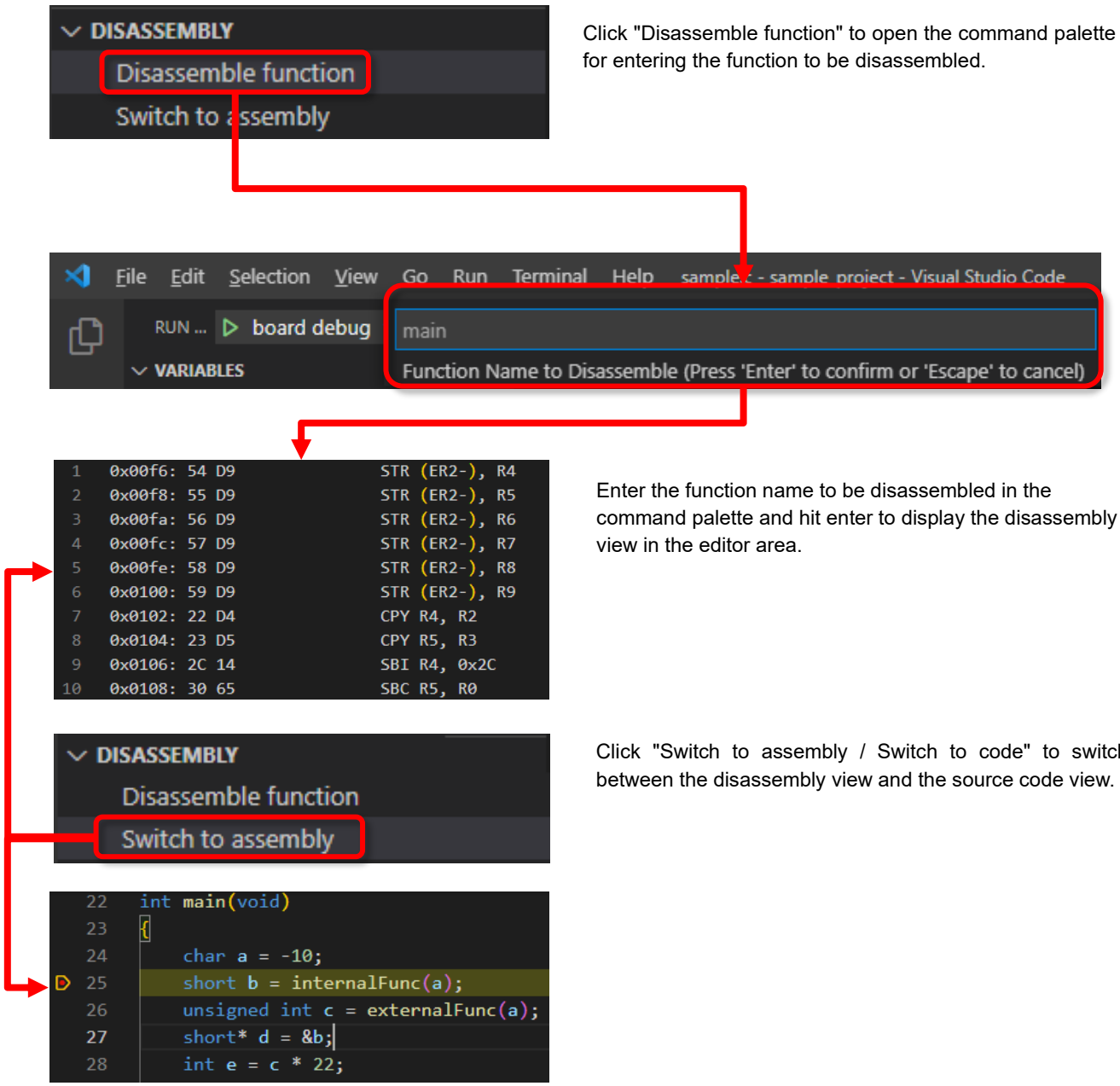
SVD files are written in XML, a markup language that uses tags ("<**>").
The file path can be set in settings.json (compiler/debugger settings).

See "Project Configuration and Setting Items".

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.

6.9 DISASSEMBLY

DISASSEMBLY area displays the disassembly result of the specified function.



Click "Disassemble function" to open the command palette for entering the function to be disassembled.

Enter the function name to be disassembled in the command palette and hit enter to display the disassembly view in the editor area.

Click "Switch to assembly / Switch to code" to switch between the disassembly view and the source code view.

Figure 36. DISASSEMBLY Operation Window

6.10 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.

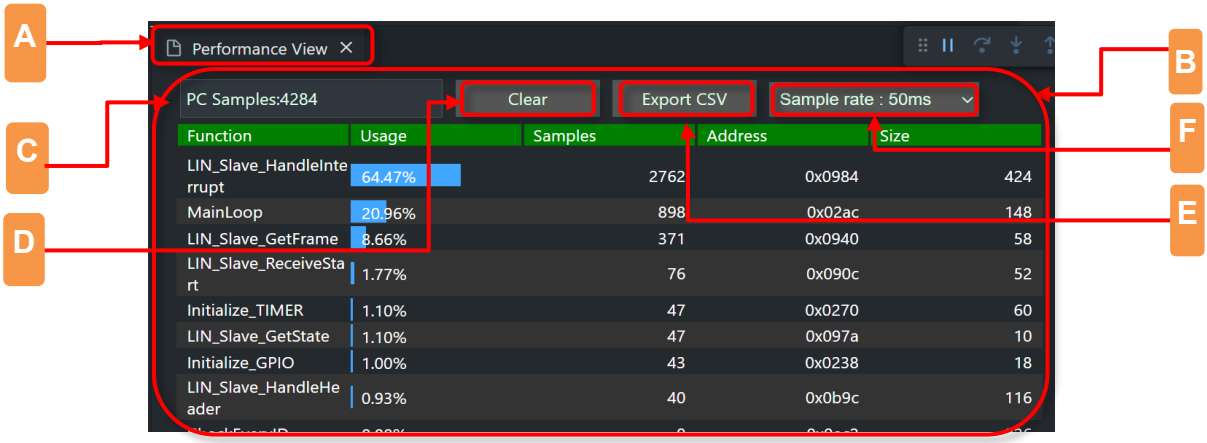


Figure 37. PERFORMANCE View

■Startup procedure: Set to enable startup in the settings file (settings.json), and when debugging is executed, a window appears and sampling begins. You can 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 13. 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.	

Notes

Since a sampling method, the measurement results are probabilistic. For accuracy, measurements of tens of seconds to several minutes are required.

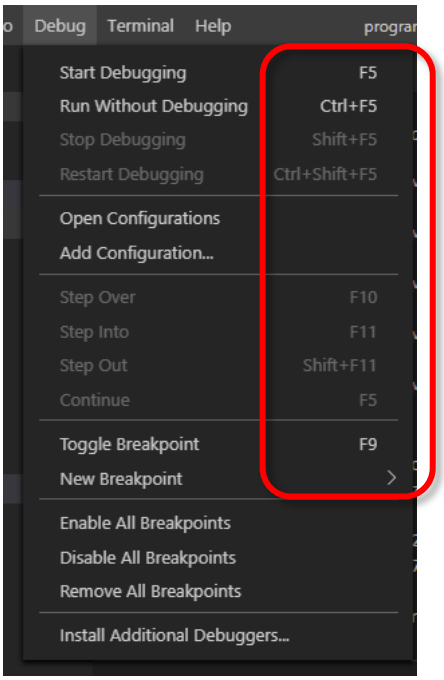
6.11 Debug Related Keyboard Shortcuts

Table 14. 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.



6.12 Memory Window

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

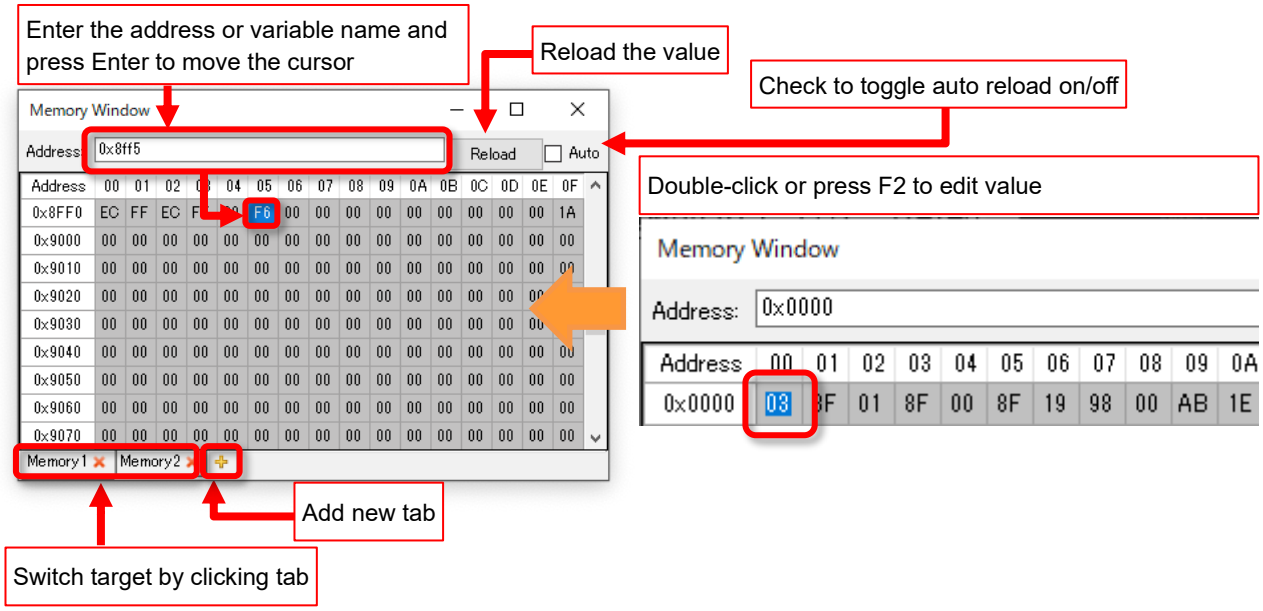
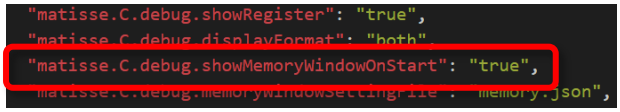
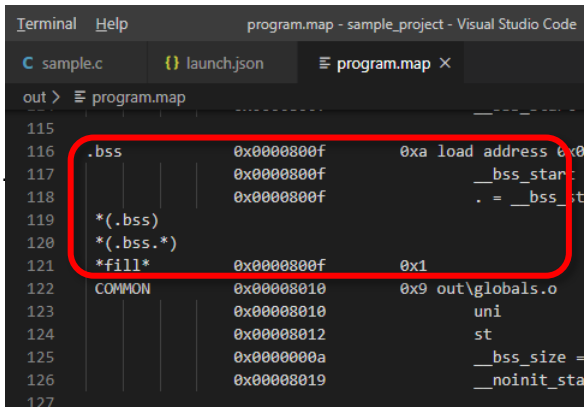


Figure 38. Memory Window





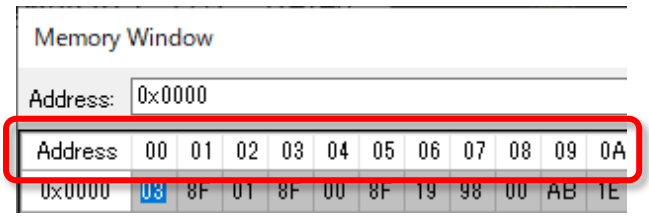


Figure 39. Memory Window Display Settings

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

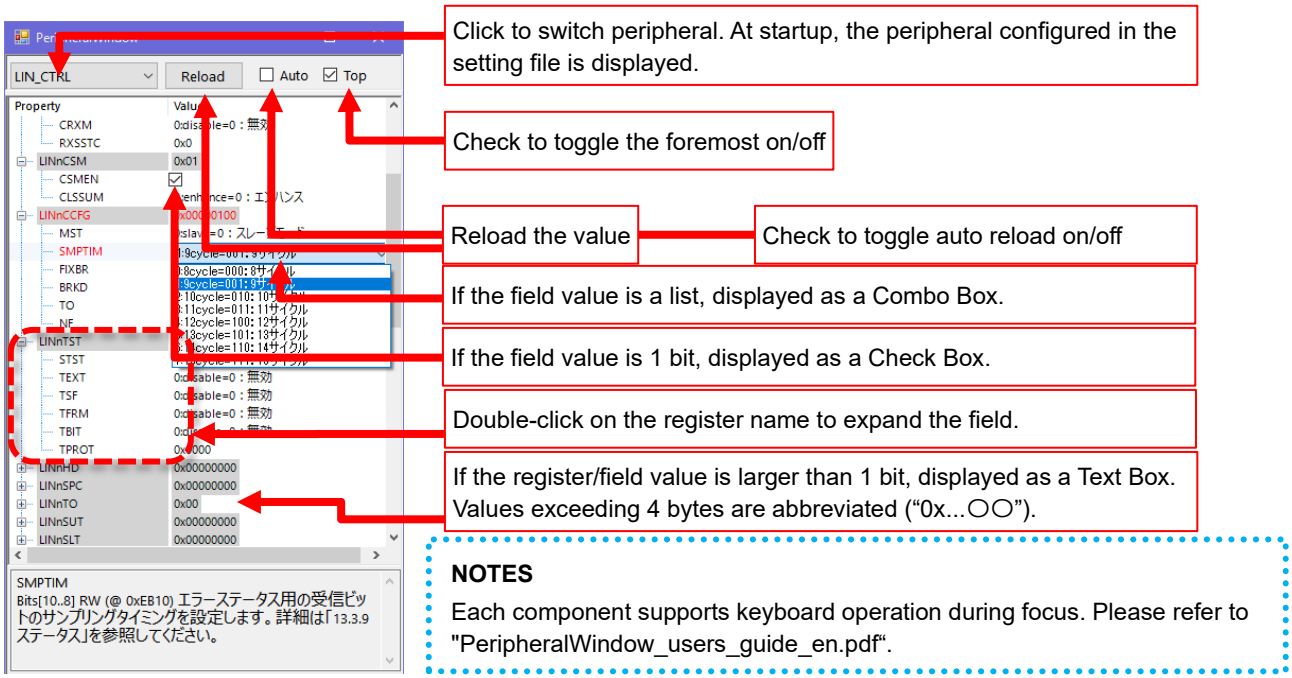
Do not click on column header (the area where displayed as "Address 00 01 .. 0F"). The memory display may be corrupted.

6.13 Peripheral Window

The Peripheral window displays information about the peripheral registers. You can monitor or change the property values by 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 40. Peripheral Window

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".

settings.json

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

Figure 41. Peripheral Window Display Settings

.ini File contents

Edit .ini File in the following formats.

.ini File

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

Table 15. ini File Format Description

Setting Item	Description
[Peripheral] SelectedPeripheral=	Describes 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.

6.14 Function call history on CPU resetting

When the CPU is reset during debugging due to WDT or hardware failure or 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.

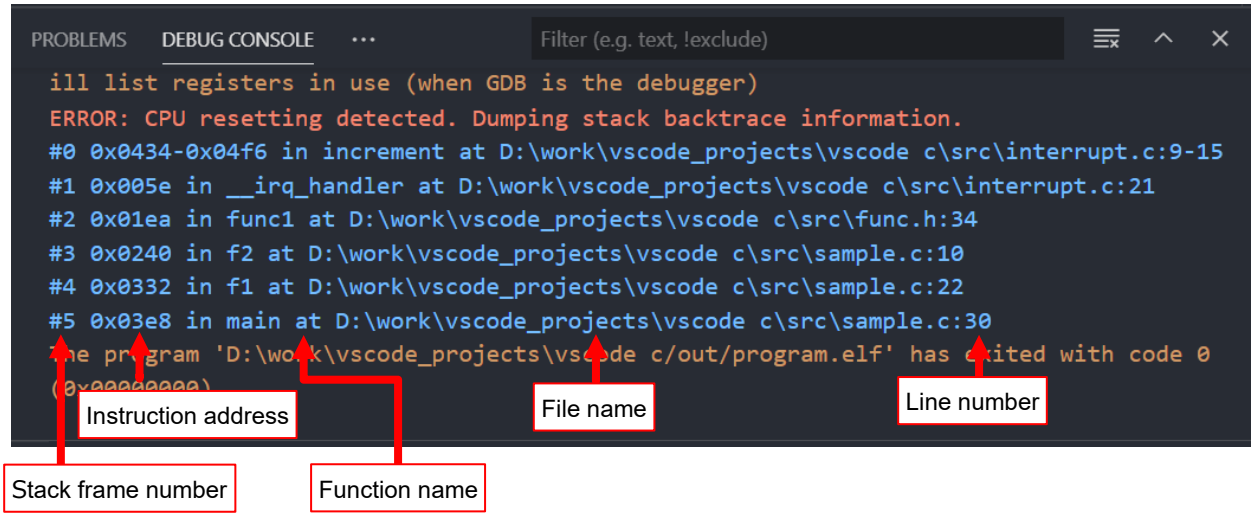


Figure 42. Display Function Call Stack at reset

Table 16. 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.

7 Command Execution

7.1 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.

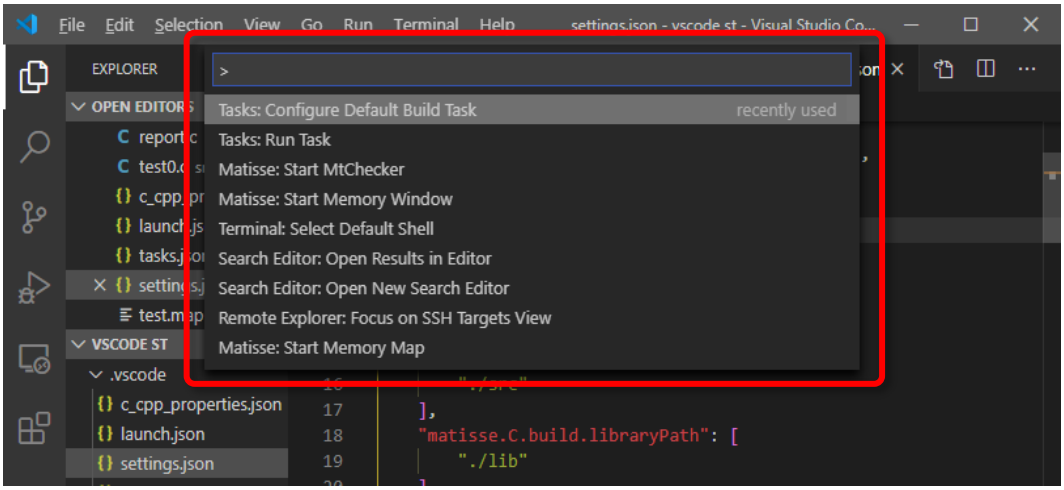


Figure 43. Command Palette Display and Command Input

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

Table 17. 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".

8 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.

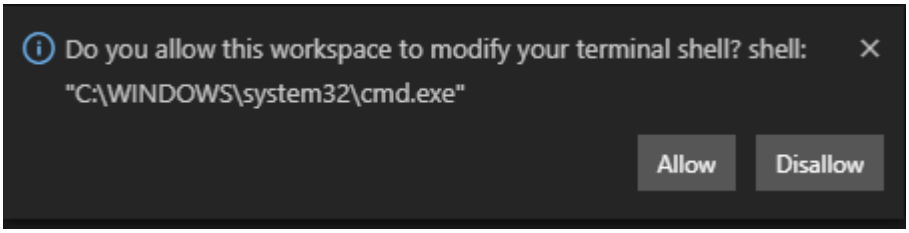
Answer

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)

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.

9 Shortcut Key List

9.1 General

Table 18. 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

9.2 File Management

Table 19. 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

9.3 Editor Management

Table 20. 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

9.4 Editing

Table 21. 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

9.5 Search and Replace

Table 22. Shortcut Key List (Search and Replace)

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

9.6 Rich Language Editing

Table 23. 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

9.7 Display

Table 24. 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

9.8 Debug

Table 25. 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

10 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)
・svgexport(The MIT License)

11 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.

Caution

1. The information written in these materials regarding the software and system (hereinafter collectively “Software”) and the contents of the materials are current as of the date of the material’s issuance, and may be changed by ROHM, at any time and for any reason, without prior notice.

2. If you plan to use the Software in connection with any equipment or device (such as the medical equipment, transportation equipment, traffic equipment, aerospace equipment, nuclear power control equipment, vehicle equipment including the fuel control system and/or car accessories, and/or various kinds of safety devices etc.) which require extremely high reliability, and whose breakdown or malfunction relate to the risk of personal injury or death, or any other serious damage (such usage is hereinafter called “Special Usage”), you must first consult with the ROHM’s sales representative. ROHM is not responsible for any loss, injury, or damage etc. incurred by you or any other third party caused by any Special Usage without ROHM’s prior written approval.

3. Semiconductor products may break or malfunction due to various factors. You are responsible for designing, testing, and implementing safety measures in connection with your use of any ROHM products using the Software (such ROHM products are hereinafter called “Product”) Such safety measures include, but are 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.. ROHM is not responsible and hereby disclaims liability for any damage in relation to your use beyond the rated value, or the non-compliance with any precaution for use.

4. ROHM is not responsible for any direct and/or indirect damage to you, or any third parties, (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, these materials, and/or the Product contain confidential information of ROHM’, including technical information, and/or trade secrets, you are prohibited from engaging in any of the following acts in whole or part, without ROHM’s prior written approval:
   (i) disclosing any ROHM confidential information to a third party;
   (ii) disassembling, 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 the Software, or the technology and/or confidential information written in these materials, you are required to follow the applicable export control laws and regulations such as “Foreign Exchange and Foreign Tarade Act” and/or “Export Administration Regulations (EAR).”.

7. ROHM disclaims all warranties, statutory or otherwise, and ROHM hereby disclaims any warranty for non-infringement for the Software and/or the information written in these materials. Accordingly, ROHM is not liable to you for any direct or third-party claims of infringement of rights.

8. No license, whether expressly or implied, is granted hereby under any intellectual property rights or other rights of ROHM or any third parties with respect to the Software or Products or the information contained in these materials.

9. You agree to indemnify, defend and hold harmless ROHM and ROHM’s officers and/or employees from responsibility, and hold them harmless, and defend them from any damage, loss, penalty, or cost caused by any claim of liability (including but not limited to the attorney fees) resulting from, or incurred relating to the following acts:
   (1) any alleged infringement of a third party’s rights or the violation of laws caused by reading, download, encryption, summarization, copy, or transfer etc.; or
   (2) violation of these materials.

10. ROHM does not guarantee that these materials or the Software is error free. ROHM shall not be in any way responsible or liable for any damages, expenses, or losses incurred by you or third parties resulting from errors contained in these materials.

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

ROHM Customer Support System

https://www.rohm.co.jp/contactus