tinyMicon MatisseCORE™

matiseye™-studio ユーザーズガイド

Visual Studio Code を使った Matisse 向け C 言語開発環境の構築と操作方法

© 2024 ROHM Co., Ltd.

No. 67UG022J Rev.008
2026.10

更新履歴

日付	バージョン	内容
2024/02/22	Rev.001	初版発行
2024/05/10	Rev.002	スタック静的解析（Stack View）の追加
2024/07/24	Rev.003	VS Code インストールの説明文の修正
2025/03/24	Rev.004	3.2. ステータスバーのデバッグクロック速度に関する説明を変更 4.2. いくつかの設定項目を追加 6.8. Memory View にデバッグ中でない場合の起動方法を追加 6.9. Peripheral View にデバッグ中でない場合の起動方法を追加、ファイル出力機能を追加 6.10. Performance View に新たな起動方法を追加
2025/08/28	Rev.005	1.2. 必要なファイルを民生開発環境用、車載開発環境用に分割して表記へ変更 2.1. 車載開発環境のアップデータの説明を追加
2025/11/11	Rev.006	7.7 Memory View のメモリ値をテーブルフォーマットでコピーできるように改善 7.8. DISASSEMBLY に実行ファイル全体の逆アセンブリ結果を表示する機能を追加 7.7 0 7.8. 注意事項を追加
2026/05/15	Rev.007	1.3 動作環境の「.NET Framework」の説明文を変更 5 ツールランチャー（matiseye-studio Sidebar View）の追加 6.4 ビルド結果解析（Build Analysis View）の追加
2026/10/22	Rev.008	5 ツールランチャー（matiseye-studio Sidebar View）に、CPU の制御およびドキュメント起動ボタンを追加 6.4.2.4 Object File Memory Details: Build Analysis View に、オブジェクトファイル別のメモリ使用量を表示する機能を追加 7.7.1 メモリパターンフィル: Memory View に、指定範囲への一括メモリ書き込み機能を追加 7.7.2 シンボルオーバーレイ: Memory View に、カーソルを合わせたアドレスのグローバル変数の情報を表示する機能を追加

© 2024 ROHM Co., Ltd.

目次

1. はじめに 5
1.1. 概要 5
1.2. ファイル構成 5
1.3. 動作環境 5
1.4. コンパイラ制限事項 5
2. インストール手順 7
2.1. Matisse 向け C 言語開発環境のインストール 7
2.2. Visual Studio Code(VS Code)のインストール 8
2.3. VS Code に cpptools をインストールする 9
2.4. VS Code に matiseye™-studio をインストールする 10
3. Visual Studio Code(VS Code)の使い方 12
3.1. VS Code 画面の説明 12
3.2. ステータスバーの説明 13
3.3. VS Code の操作 14
4. プロジェクト構成と設定項目 15
4.1. サンプルプロジェクトの構成 15
4.2. コンパイラ・デバッガ設定項目（settings.json） 16
5. ツールランチャー（matiseye-studio Sidebar View） 19
6. ビルド 21
6.1. ビルドの実行と生成されるファイル 21
6.2. MAP ファイル 22
6.3. スタック静的解析（Stack View） 23
6.4. ビルド結果解析（Build Analysis View） 24
6.4.1. サマリ表示 24
6.4.2. 詳細解析ビュー 25
7. デバッグ 28
7.1. デバッグ画面の説明 28
7.2. デバッグメニュー 30
7.3. ブレークポイント（BREAKPOINTS） 31
7.4. デバッグツールバー 32
7.5. データインスペクション 33
7.6. コールスタック（CALL STACK） 34
7.7. MEMORY 35
7.8. PERIPHERALS 39
7.9. DISASSEMBLY 41
7.10. PERFORMANCE 42
7.11. デバッグ関連のキーボードショートカット 43
7.12. メモリウィンドウ（Memory Window） 44
7.13. ペリフェラルウィンドウ（Peripheral Window） 45
7.14. リセット時の関数呼び出し履歴表示 46
8. コマンド実行 47

© 2024 ROHM Co., Ltd.

8.1. コマンドパレットの表示とコマンド入力 47
9. よくある質問 48
10. ショートカットキー 一覧 50
10.1. 基本機能 50
10.2. ファイル操作 51
10.3. エディタ操作 52
10.4. 編集 53
10.5. 検索と置換 54
10.6. プログラミング 55
10.7. 画面表示 56
10.8. デバッグ 57
11. オープンソースソフトウェアについて 58
12. 商標について 58

© 2024 ROHM Co., Ltd.

1. はじめに

1.1. 概要

matiseye™-studio は Visual Studio Code をベースにした Matisse 向け C 言語開発ツールです。

> 補足
> Visual Studio Code(以下 VS Code)は Microsoft 社が開発した多機能かつ軽量なオープンソースのコードエディタです。

1.2. ファイル構成

Table 1. matiseye™-studio 民生ソフト開発環境 ファイル構成

ファイル名	説明
matisse-dev-env-installer-3.*.*.exe	民生開発用 Matisse 向け C 言語開発環境のインストーラです。
matiseye-studio-*.*.*. vsix	VS Code の C 言語開発機能を Matisse 向けに使用可能にする拡張ファイルです。
sample_project.zip	C 言語開発プロジェクトのサンプルです。

Table 2. matiseye™-studio 車載ソフト開発環境 ファイル構成

ファイル名	説明
MatisseCCompiler-2.*.*.exe	車載開発用 Matisse 向け C 言語開発環境のインストーラです。
matisse-automotive-dev-env-updater_ *.*.*.exe	車載開発用 Matisse 向け C 言語開発環境のアップデータです。コンパイラ以外の周辺ツールをアップデートするためのものです。
matiseye-studio-*.*.*. vsix	VS Code の C 言語開発機能を Matisse 向けに使用可能にする拡張ファイルです。
sample_project.zip	C 言語開発プロジェクトのサンプルです。

1.3. 動作環境

Table 3. matiseye™-studio 動作環境

環境	説明
OS	Windows 11
CPU	Intel Core シリーズ、もしくは同等の性能の CPU。
メモリ	2GByte 以上搭載。
HDD/SSD	1GByte 以上の空き容量。
.NET Framework	バージョン 4.7.2(Windows 10 バージョン 1803 以降ではプレインストール済み)

1.4. コンパイラ制限事項

現バージョンでは以下の制限事項があります。

© 2024 ROHM Co., Ltd.

［Matisse の設定］

・ MUL 命令: ハードウェアに合わせて選択可能(ハードウェア/IDE とも MUL 無しがデフォルト)

・ 汎用レジスタ数：16 に固定

［その他］

・ 64bit 整数(long long)未対応、浮動小数点(float, double)未対応

・ 可変長引数と可変長配列未対応

・ 動的メモリ確保未対応

© 2024 ROHM Co., Ltd.

2. インストール手順

> ⚠ 注意
> Windows10(バージョン 1803)以前の環境をご使用の場合、.NET Framework 4.7.2 がインストールされていない場合は、事前にインストールしてください。

2.1. Matisse 向け C 言語開発環境のインストール

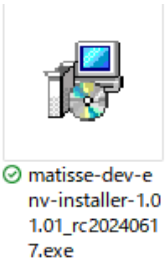


Figure 1. matisse-dev-env-installer 実行

1 民生ソフト開発環境を使用する場合は、「matisse-dev-env-installer-3.*.*.exe」、車載ソフト開発環境を使用する場合は、「MatisseCCompiler-2.*.*.exe」をダブルクリックして、インストーラを実行します。

> 補足
> ユーザーアカウント制御が表示された場合は、「はい」をクリックします。

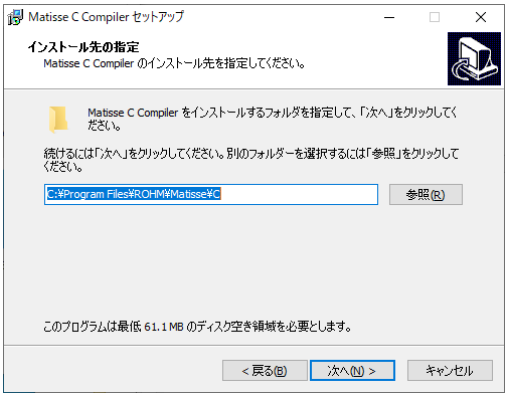


Figure 2. Matisse Develop Environment インストール画面

2 表示される画面の指示に従って、インストールを行います。

> 補足
> 「インストール先の指定」でデフォルトのインストール場所を変更した場合は、設定ファイルに C 言語開発ツールのパスを記入する必要がありますのでご注意ください。

「プロジェクト構成と設定項目」を参照。

3 C 言語開発環境のインストール中に以下のインストーラも自動的に実行します。

・マイクロソフト Visual Studio 再頒布可能パッケージ
・MtProxy(デバッグボード通信プロキシサーバ)
・MtChecker(開発環境設定確認ツール)
・mtloader(プログラムダウンローダ)

> 補足
> ユーザーアカウント制御が表示された場合は、「はい」をクリックします。

4 車載ソフト開発環境を使用する場合は、「matisse-automotive-dev-env-updater_ *.*.*.exe」を実行します。

5 パソコンを再起動してください。

© 2024 ROHM Co., Ltd.

2.2. Visual Studio Code(VS Code)のインストール

補足
VS Code のインストーラは WEB サイト(https://code.visualstudio.com)から最新版をダウンロードしてください。VS Code のバージョンが古い場合、matiseye™-studio の機能が正常に動作しない可能性があります。インストール済みの場合でも、最新のバージョンをインストールしてください。

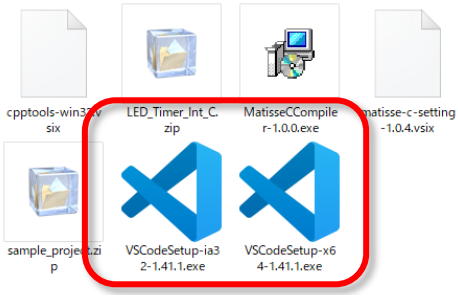


Figure 3. VS Code インストーラ実行

1 VS Code のインストーラをダブルクリックして、インストーラを起動します。

補足
インストーラには、32-bit Windows 用と 64-bit Windows 用があります。
32-bit 用：VSCodeSetup-ia32-*.*.*.exe
64-bit 用：VSCodeSetup-x64-*.*.*.exe

補足
ユーザーアカウント制御が表示された場合は、「はい」をクリックします。

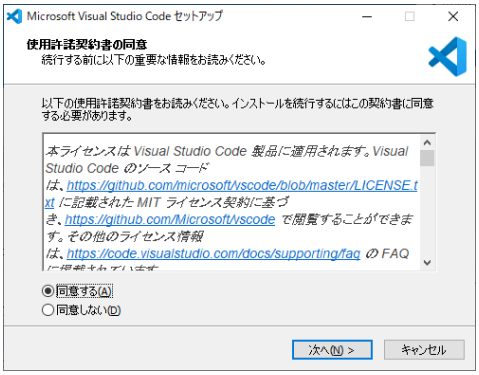


Figure 4. VS Code インストール

2 表示される画面の指示に従って、インストールを行います。

© 2024 ROHM Co., Ltd.

2.3. VS Code に cpptools をインストールする

補足
・cpptools は C 言語プログラムを開発する機能を VS Code に追加する拡張ファイルです。

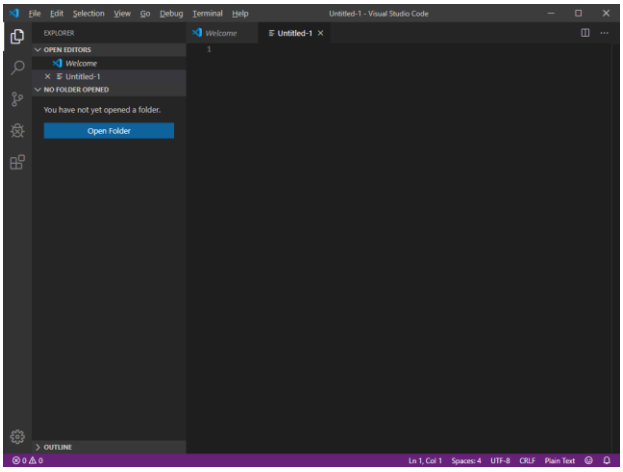


Figure 5. VS Code 起動

1 VS Code を起動します。

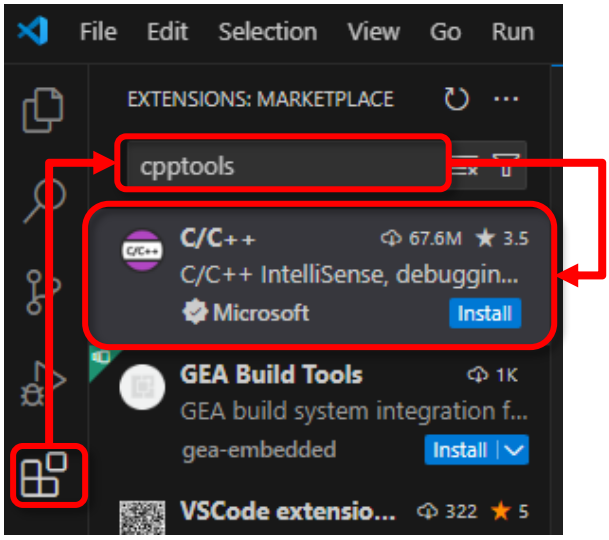


Figure 6.cpptools の検索

2 [ ] （Extensions）→検索窓に「cpptools」を入力し、「C/C++」を選択します。

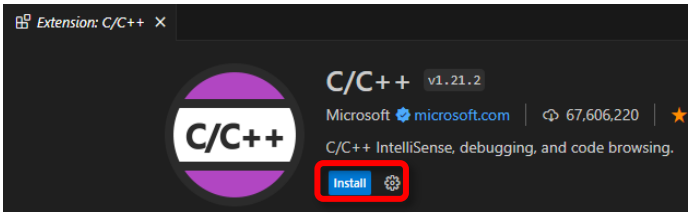


Figure 7. cpptools インストール

4 「Install」をクリックし、インストールを実行します。

5 VS Code を再起動します。

© 2024 ROHM Co., Ltd.

2.4. VS Code に matiseye™-studio をインストールする

matiseye™-studio の拡張ファイルをインストールする事で VS Code に Matisse 向け C 言語開発に必要な機能が追加されます。
以前のバージョンの拡張ファイル(matisse-c-setting や mt-studio)をインストールしている場合はアンインストールして 1 から以下の手順を実行してください。

⚠ 注意
他の C/C++開発用の VS Code 拡張(CMake, PlatformIO など)がインストールされていると機能が衝突する場合があります。
matiseye™-studio を使用する場合はそれらの拡張を無効化してください。

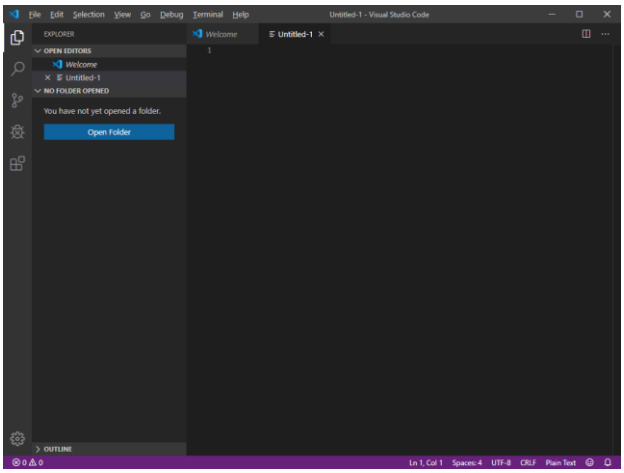


Figure 8. VS Code 起動

1 VS Code を起動します。

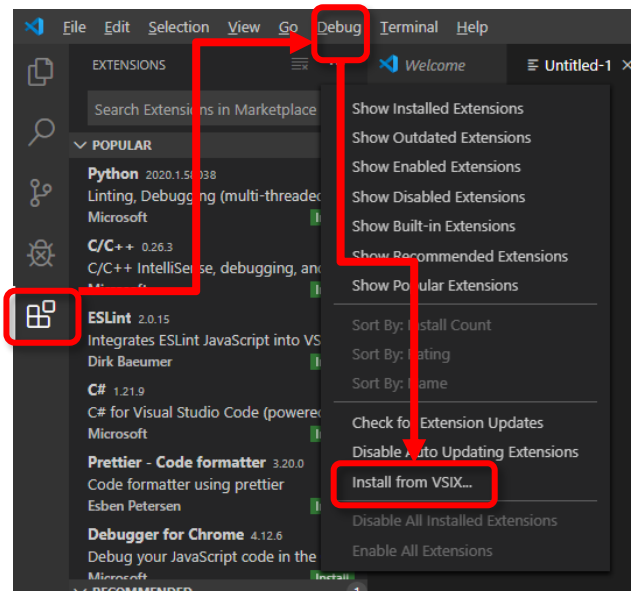


Figure 9. Install from VSIX クリック

2 [ ] (Extensions) → [・・・] (More Actions) → [Install from VSIX] を順にクリックします。

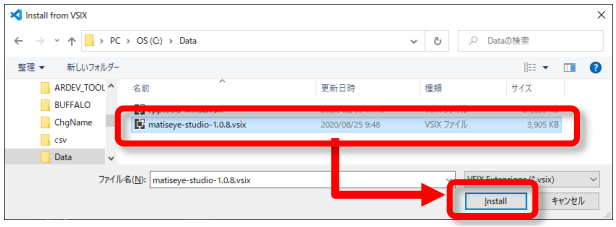


Figure 10. matiseye-studio_*.*.*. vsix インストール

3「matiseye-studio_*.*.*. vsix」を選択して [Install] をクリックします。

4 VS Code を再起動します。

© 2024 ROHM Co., Ltd.



Figure 11. matiseye-studio 拡張機能詳細ビュー

5 再起動後、インストール済みの拡張機能リストに matiseye-studio が追加されます。クリックすると、拡張機能の詳細ビューが表示されます。" Disable" であれば、拡張機能は有効になっています。" Enable" と表示されている場合は、クリックすることで有効化されます。

© 2024 ROHM Co., Ltd.

3. Visual Studio Code(VS Code)の使い方

3.1. VS Code 画面の説明

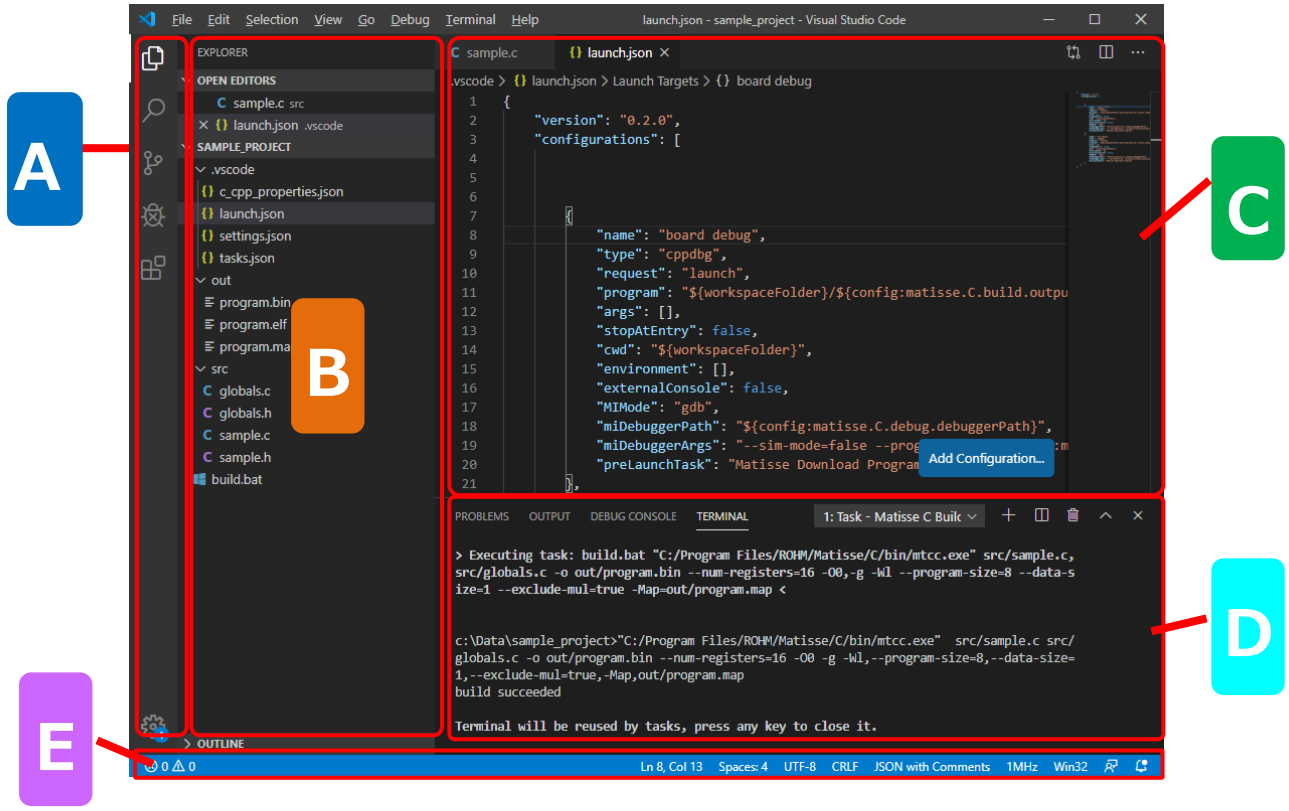


Figure 12. VS Code 画面

Table 4. VS Code 画面の説明

	名称	説明
A	Explorer	開いているファイルを一覧表示します。
	Search	ファイルから指定したキーワードを含むファイルを検索、置換します。
	Source Control	Git と連携します。
	Debug and Run	プログラムのデバッグを行います。
	Extensions	拡張機能の検索などを行います。
B	サイドバー	「Explorer」が選択されている場合はフォルダやファイル一覧、「Search」が選択されている場合は検索ワードの入力や結果表示など、選択した機能により異なる内容が表示されます。
C	エディタ	開いているファイルの内容を表示する。エディタの分割表示にも対応しています。
D	パネル	デバッグ情報やコマンドプロンプトなどが表示されます。
E	ステータスバー	文字コードや改行コードなど、ファイルのステータスに関する情報を表示します。 次ページに詳細があります。

© 2024 ROHM Co., Ltd.

3.2. ステータスバーの説明

画面最下部のステータスバーでは VS Code の現在の設定内容の確認と変更ができます。

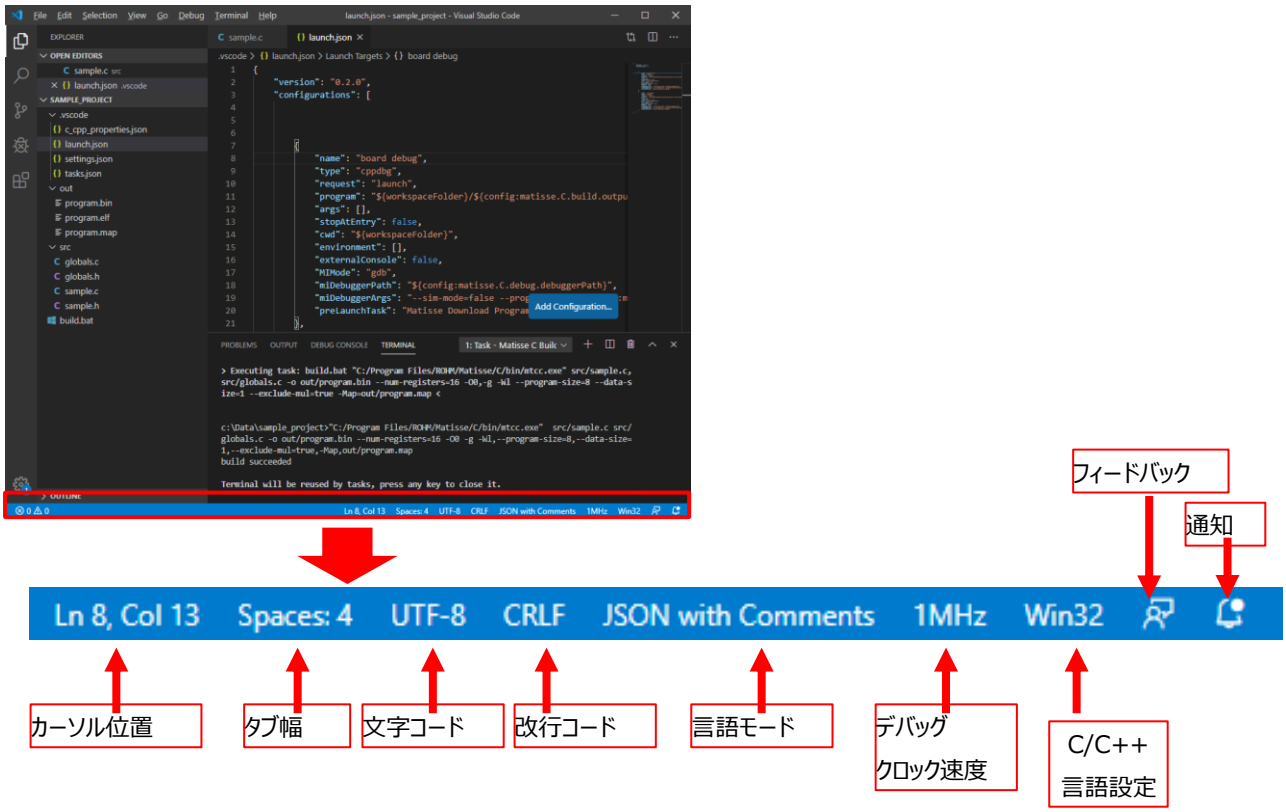


Figure 13. VS Code ステータスバー

Table 5. VS Code ステータスバーの説明

名称	説明
カーソル位置	カーソル位置を表示します。クリックすると変更できます。
インデント(字下げ幅)	インデントを表示します。クリックすると変更できます。
文字コード	文字コードを表示します。クリックすると変更できます。
改行コード	改行コードを表示します。クリックすると変更できます。
言語モード	表示中のファイルをどの言語モードで開いているか表示します。クリックすると変更できます。
デバッグクロック速度	Matisse の開発ボードでデバッグする時のデバッグインターフェイスの動作周波数を表示します。 クリックすると動作周波数が変更できます。デバッグ中はロックがかかり変更できないようになっています。
C/C++言語設定	C/C++の言語設定が表示されます。変更は不要です。クリックしないでください。
フィードバック	クリックするとマイクロソフト社に VS Code に関するフィードバックコメントを送る事ができます。 ロームのサポート窓口ではありません。
通知	クリックすると VS Code からの通知が表示されます。

⚠ 注意

フィードバック機能を使用してもロームのサポート担当には連絡できません。

ローム製品へのご意見やトラブル対応は正式なサポート窓口に連絡してください。

© 2024 ROHM Co., Ltd.

3.3. VS Code の操作

sample_project を使って VS Code の操作をしてみましょう。

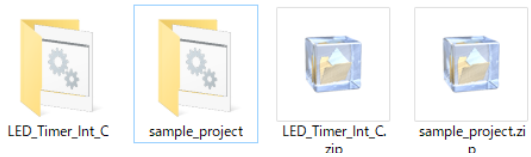


Figure 14. VS Code sample_project

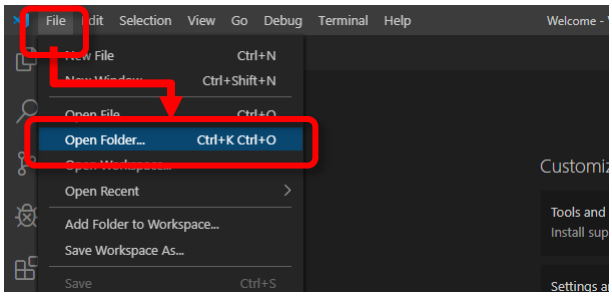


Figure 15. Open Folder クリック

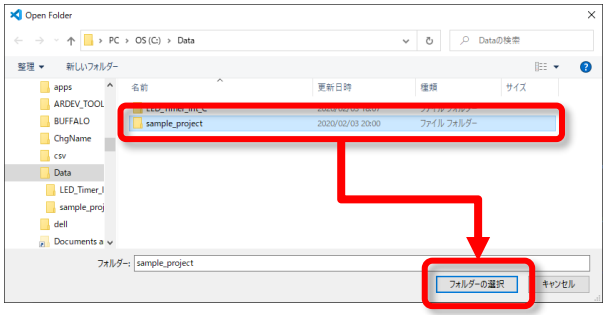


Figure 16. sample_project フォルダ選択

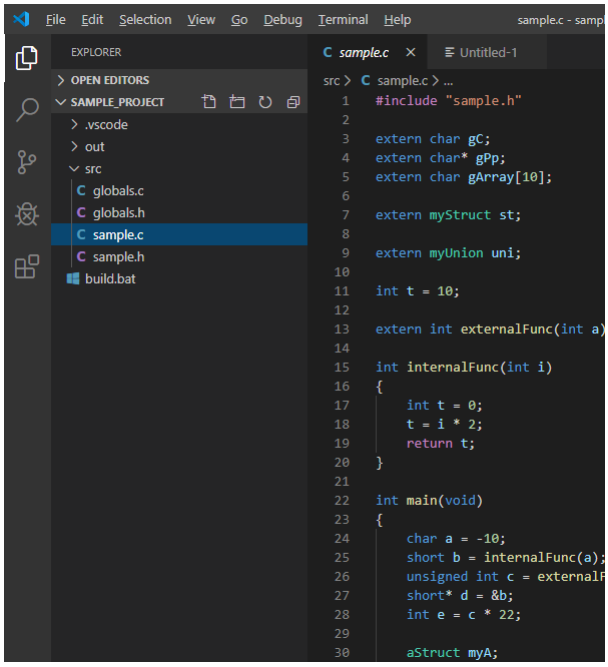


Figure 17. C ソースファイル編集

sample_project を開く

1 sample_project.zip を解凍して適当な場所に保存してください。

> ⚠ 注意
> プロジェクトディレクトリは、日本語や空白を含むパスには置かないでください。

2 VS Code を起動してツールバーの「File」から「Open Folder」を選択します。

> ⚠ 注意
> ワークスペース機能は未対応です。プロジェクトフォルダを直接開いてください。

3 解凍した「sample_project」フォルダを選択し、［フォルダーの選択］をクリックします。

C ソースを書く

4 EXPLORER で sample_project > src > sample.c を選択します。C ソースが表示されます。そのままソースファイルを編集することができます。

> 補足
> C の文法的に間違った内容を書くとコンパイル前でも指摘が出ます。

ビルド

5 作成したデータをビルドして実行可能ファイルに変換します。

「ビルド」を参照。

デバッグ

6 プログラムの動作を確認します。

「デバッグ」を参照。

© 2024 ROHM Co., Ltd.

4. プロジェクト構成と設定項目

4.1. サンプルプロジェクトの構成

サンプルプロジェクト「sample_project」を元にデータの構成と設定を説明します。

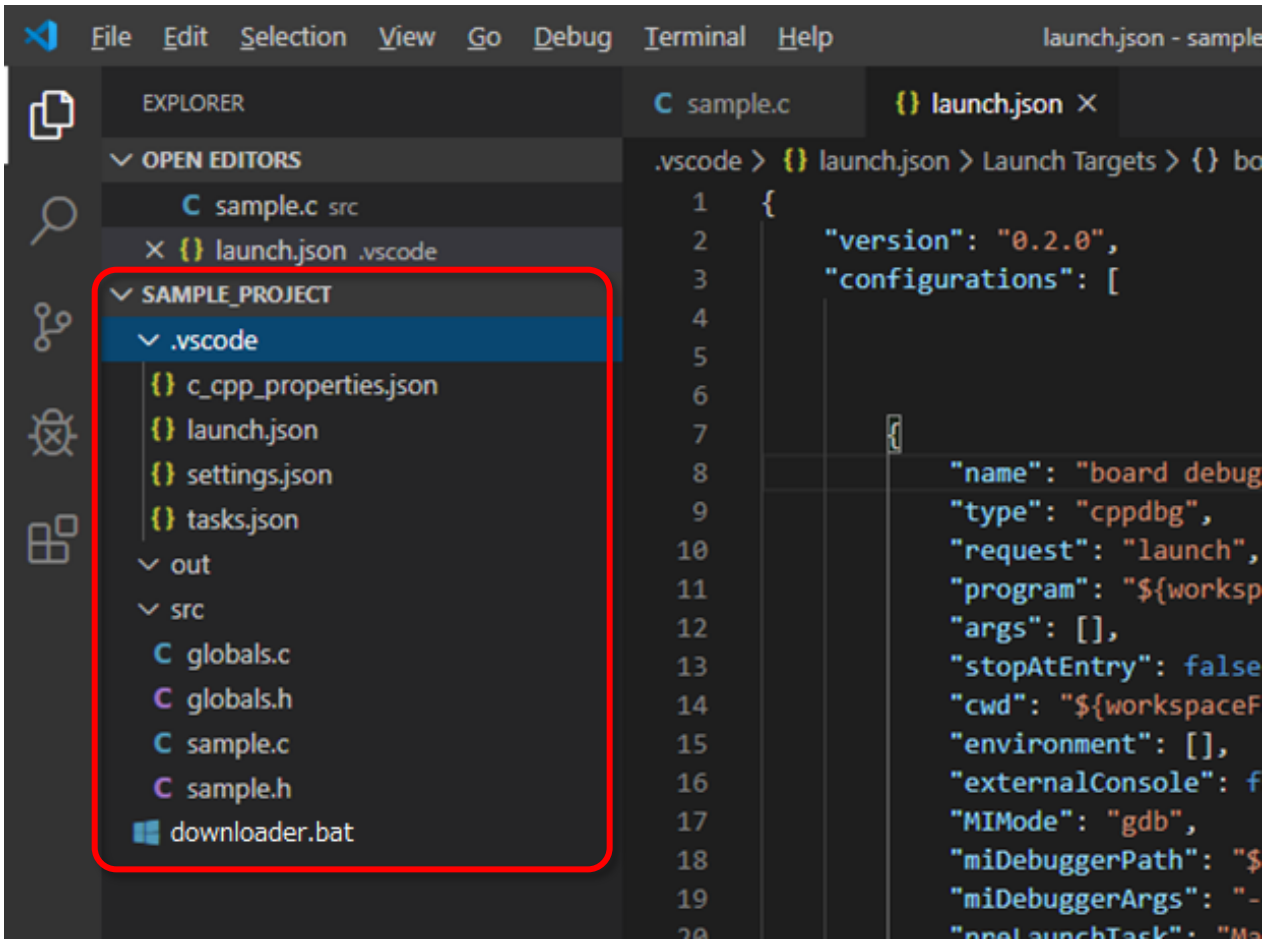


Figure 18. サンプルプロジェクト構成

Table 6. サンプルプロジェクト構成

フォルダ・ファイル名	説明
.vscode/settings.json	ビルド・デバッグ設定
src/	C 言語ソースコード用フォルダ
out/	ビルド結果出力フォルダ
downloader.bat	不揮発メモリを搭載した開発ボードを使用する場合はデバッグ実行の前にこのバッチファイルを実行してプログラムデータを不揮発メモリに書き込んでください。 詳細はプログラムダウンローダ(mtloader)のマニュアルを参照してください。

© 2024 ROHM Co., Ltd.

4.2. コンパイラ･デバッガ設定項目（settings.json）

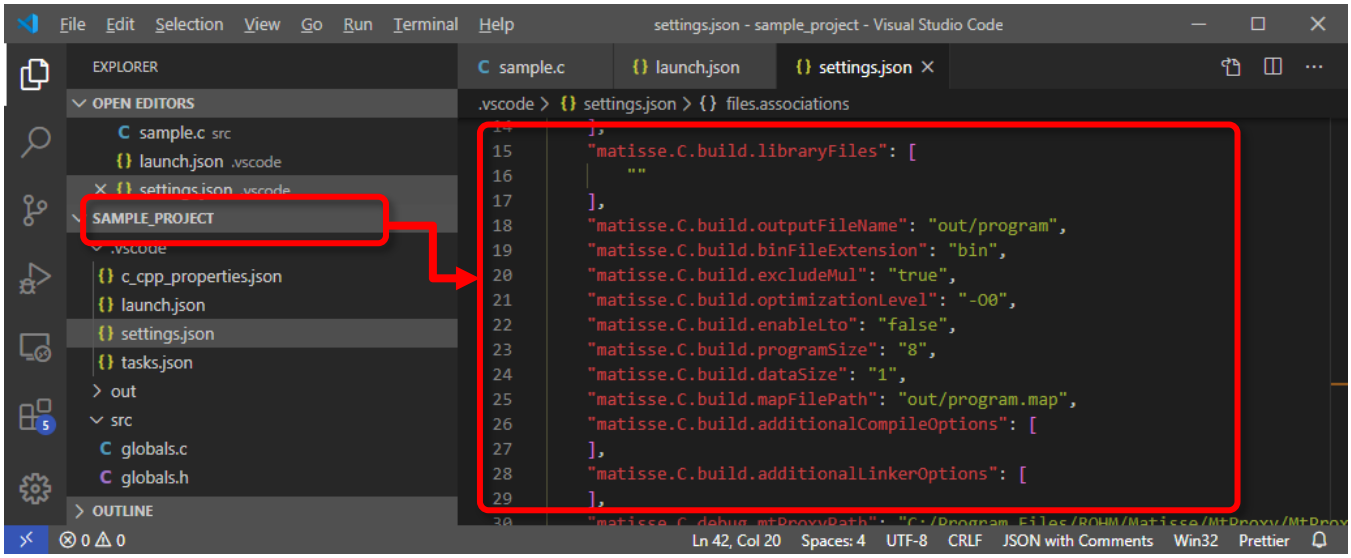


Figure 19. コンパイラ･デバッガ設定項目

Table 7. コンパイラ･デバッガ設定項目の説明

設定名	説明
matisse.C.build.compilerPath (*1)	コンパイラ(mtcc)のパス。絶対パスを指定します。
matisse.C.debug.debuggerPath (*1)	デバッガ(mtcsim)のパス。絶対パスを指定します。
matisse.C.debug.mtProxyPath (*1)	デバッグボード通信プロキシサーバ(MtProxy)のパス。絶対パスを指定します。
matisse.C.others.downloaderPath (*1)	プログラムダウンローダ(mtloader)のパス。絶対パスを指定します。
matisse.C.others.mtCheckerPath (*1)	開発環境設定確認ツール(MtChecker)のパス。絶対パスを指定します。
matisse.C.build.srcFiles	コンパイル対象のソースファイルのリスト。カンマで区切ります。
matisse.C.build.includePath	インクルードファイルを読み込むフォルダのリスト、カンマで区切ります。
matisse.C.build.libraryPath	ライブラリファイルを読み込むフォルダのリスト、カンマで区切ります。
matisse.C.build.libraryFiles	リンクするライブラリファイルのリスト、カンマで区切ります。
matisse.C.build.preprocessorDefinitions	プリプロセッサの定義(#define)のリスト。カンマで区切ります。
matisse.C.build.excludeMul	true:乗算器を含まない false:乗算器を含む
matisse.C.build.optimizationLevel	最適化レベル選択 -O0(最適化なし)/-O1(最適化レベル 1)/-O2(最適化レベル最大)/-Os(省 ROM サイズ)
matisse.C.others.downloaderOptions	ダウンローダのオプションリスト、カンマで区切ります。
次ページに続く	

(*1)C 言語開発ツールのインストール時にインストール場所を変更した場合は、設定ファイルにこれらのツールの絶対パスを記入する必要があります。変更しなかった場合はこれらの項目の記入は不要です。

⚠注意
設定項目には、日本語を入力しないでください。

© 2024 ROHM Co., Ltd.

設定名	説明
matisse.C.build.programSize	プログラム領域サイズを指定(8-14) 8: 32kbyte / 9:36kbyte / 10: 40kbyte / 11:44kbyte 12: 48kbyte / 13: 52kbyte / 14: 56kbyte
matisse.C.build.dataSize	データ領域サイズを指定(1-8) 1: 4kbyte / 2: 8kbyte / 3: 12kbyte / 4: 16kbyte 5: 20kbyte / 6: 24kbyte / 7: 28kbyte / 8: 32kbyte
matisse.C.build.exProgramSize	拡張プログラム領域サイズを指定(0-32) 0: 0byte / 1: 2kbyte / 2: 4kbyte / ... / 32: 64kbyte
matisse.C.build.enableC99CompliantDiagnostics	true: ソースコード診断機能が C99 規格に厳密に準拠する。 false: ソースコード診断機能はデフォルト設定のまま。
matisse.C.debug.debugClockSpeed	デバッグインターフェイスの動作周波数を指定する設定 値を編集して保存すると、デバッグクロックに即座に適用されます。 ・matiseye-adapter pro は 10kHz - 24MHz の範囲で設定可能。 ・matiseye-adapter は 10kHz - 1.5MHz の範囲で設定可能で、 1.5MHz を超える値を指定した場合、自動的に 1.5MHz が適用されます。 また、ステータスバーからも設定の変更が可能です。
matisse.C.debug.showMemoryWindowOnStart	true: デバッグ開始時にメモリウィンドウを表示する。 false: デバッグ開始時にメモリウィンドウを表示しない。 「メモリウィンドウ（Memory Window）」を参照。
matisse.C.debug.showPeripheralWindowOnStart	true: デバッグ開始時にペリフェラルウィンドウを表示する。 false: デバッグ開始時にペリフェラルウィンドウを表示しない。 「ペリフェラルウィンドウ（Peripheral Window）」を参照。
matisse.C.debug.showBacktraceOnReset	true: デバッグ中に CPU リセットを検出し、関数呼び出し履歴を表示してデバッグを終了する。 false: デバッグ中に CPU リセットを検出したらデバッグを終了する。 「リセット時の関数呼び出し履歴表示」を参照。
matisse.C.debug.peripheralViewSvdPath	PERIPHERALS が起動時に読み込むペリフェラルレジスタ情報ファイルのパス。 絶対/相対パスを指定します。ファイルは CMSIS-SVD 形式となります。 「PERIPHERALS」を参照。
matisse.C.debug.enablePeripheralView	true：デバッグ開始時に PERIPHERALS を起動する。 false：デバッグ開始時に PERIPHERALS を起動しない。
"matisse.C.debug.showPerformanceViewOnStart"	true：デバッグ開始時に PERFORMANCE を起動する。 false：デバッグ開始時に PERFORMANCE を起動しない。 「PERFORMANCE」を参照。
"matisse.C.debug.performanceViewSampleRate"	PERFORMANCE 起動時のサンプリングレートの設定 30ms/50ms(初期値)/70ms/100ms/200ms/500ms/1s
matisse.C.others.extensionDebugLevel	拡張機能のログ出力に関する設定です。これは matiseye-studio の開発者向けの設定のため、通常は"None"に設定してください。

© 2024 ROHM Co., Ltd.

matisse.C.others.openExtensionDetailsOnProjects Startup	プロジェクト起動時に拡張機能の詳細ビューを表示するかどうかの設定 true：プロジェクト起動時に詳細ビューを表示する。 false：プロジェクト起動時に詳細ビューを表示しない。
他の項目は変更不要です	

© 2024 ROHM Co., Ltd.

5. ツールランチャー（matiseye-studio Sidebar View）

matiseye-studio Sidebar View（以下、mt Sidebar View）は、ビルド処理や解析系ツールの実行ボタンを サイドバーに集約し、ワンクリックまたはダブルクリックで各機能を実行できるツールランチャーです。

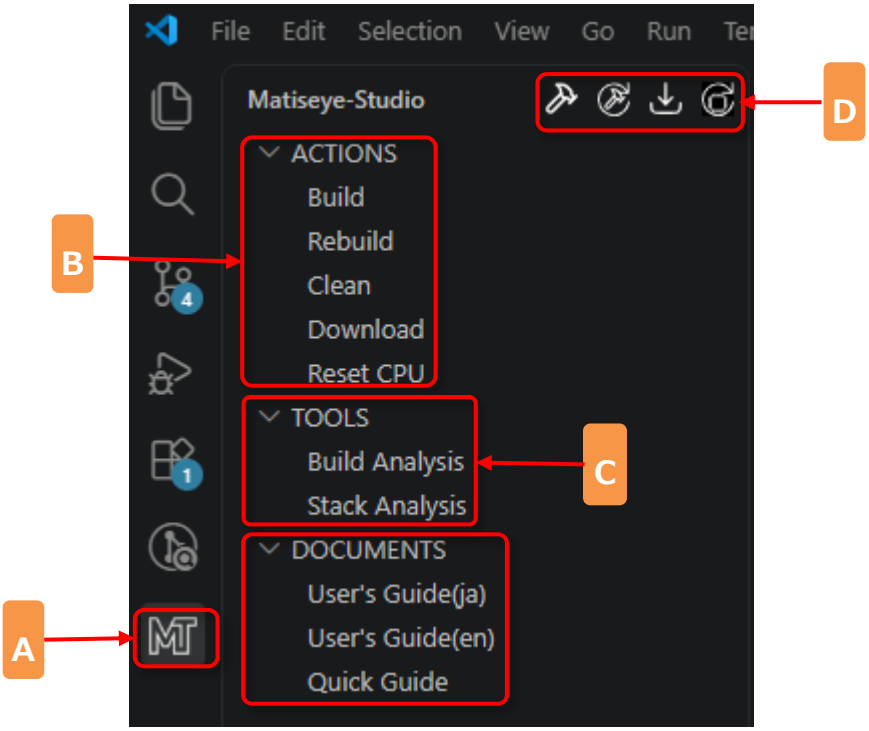


Figure 20. mt Sidebar View

Table 8.mt Sidebar View の画面説明

	名称	説明
A	MT アイコン	クリックするとサイドバーに「mt Sidebar View」の画面が表示されます。
B	ACTIONS ツリー	ダブルクリックすると、tasks.json で定義された以下のビルド関連タスクを実行できます。 ・Build: "Build"タスクを実行します（Ctrl + Shift + B と同じ操作）。 ・Clean: "Clean"タスクを実行し、Build で生成されたすべてのファイルを削除します。 ・Rebuild: "Rebuild"タスクを実行し、Clean 後に Build を実行します。 ・Download: "Matisse Download Program"タスクを実行し、ターゲットにプログラムを書き込みます。 「ビルドの実行と生成されるファイル」を参照。 ダブルクリックすると、以下の CPU 制御を実行できます。 ・Reset CPU: CPU とデバッグインターフェイスをリセットします。
C	TOOLS ツリー	各種解析機能を実行できます。 ・Build Analysis: ビルド結果を解析します。 「ビルド結果解析」を参照。 ・Stack Analysis：スタック使用量を解析します。 「スタック静的解析」を参照。
D	ACTIONS アイコン	ACTIONS の一部の機能は、対応するアイコンをクリックして実行できます。 Build Rebuild Download Reset CPU

© 2024 ROHM Co., Ltd.

E	DOCUMENTS ツリー	ユーザーズガイド(日/英)およびクイックガイドを開くことができます。 ・User's Guide (ja): 日本語版ユーザーズガイド（PDF）を開きます。 ・User's Guide (en): 英語版ユーザーズガイド（PDF）を開きます。 ・Quick Guide: クイックガイドを開きます。各種機能の使い方が動画付きで確認できます。

⚠ **注意：**

・Matisse 用プロジェクトを開いていない場合、MT アイコンは表示されません。

・タスクの実行は tasks.json に依存するため、ファイルのタスク名は変更しないでください。

© 2024 ROHM Co., Ltd.

6. ビルド

6.1. ビルドの実行と生成されるファイル

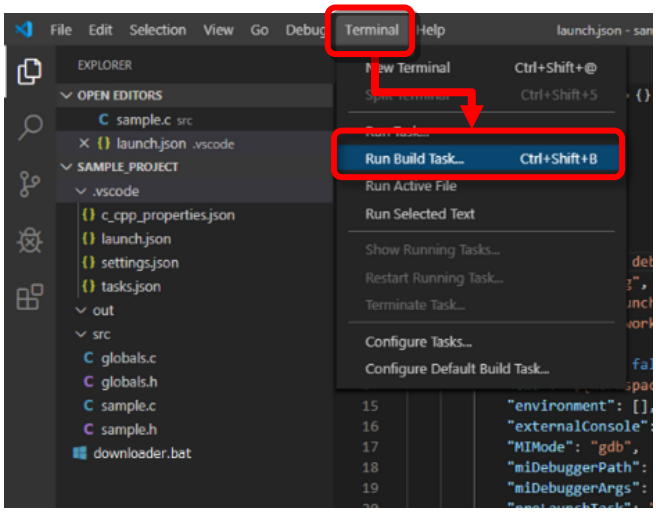


Figure 21. Run Build Task 実行

1 ツールバーの「Terminal」から「Run Build Task」を選択するか、Ctrl + Shift + B を押すとビルド処理が実行されます。

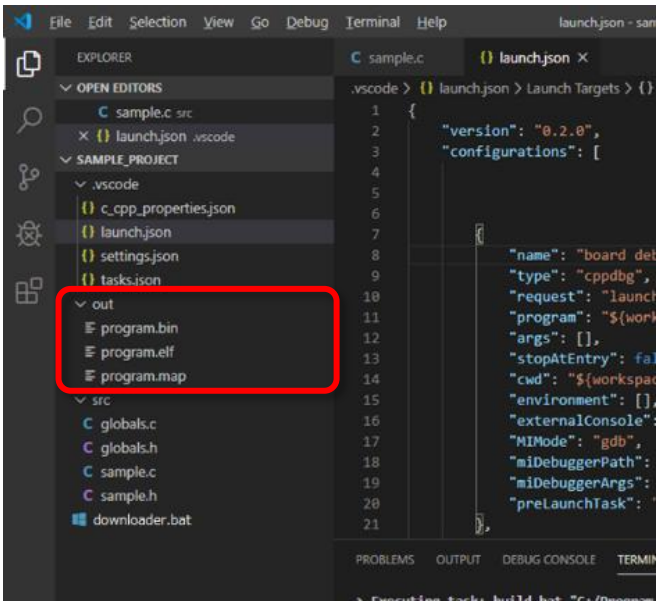


Figure 22. Run Build Task ファイル生成

2 ビルド実行後、「out」フォルダにファイルが生成されます。

Table 9.Run Build Task の説明

ファイル名	説明
out/program.map	MAP ファイル。メモリ使用量や関数、変数のアドレス確認に利用できる
out/program.elf	プログラムのデバッグ用バイナリ
out/program.bin	プログラムのバイナリ

© 2024 ROHM Co., Ltd.

6.2. MAP ファイル

サンプルプロジェクト「sample_project」の MAP ファイル

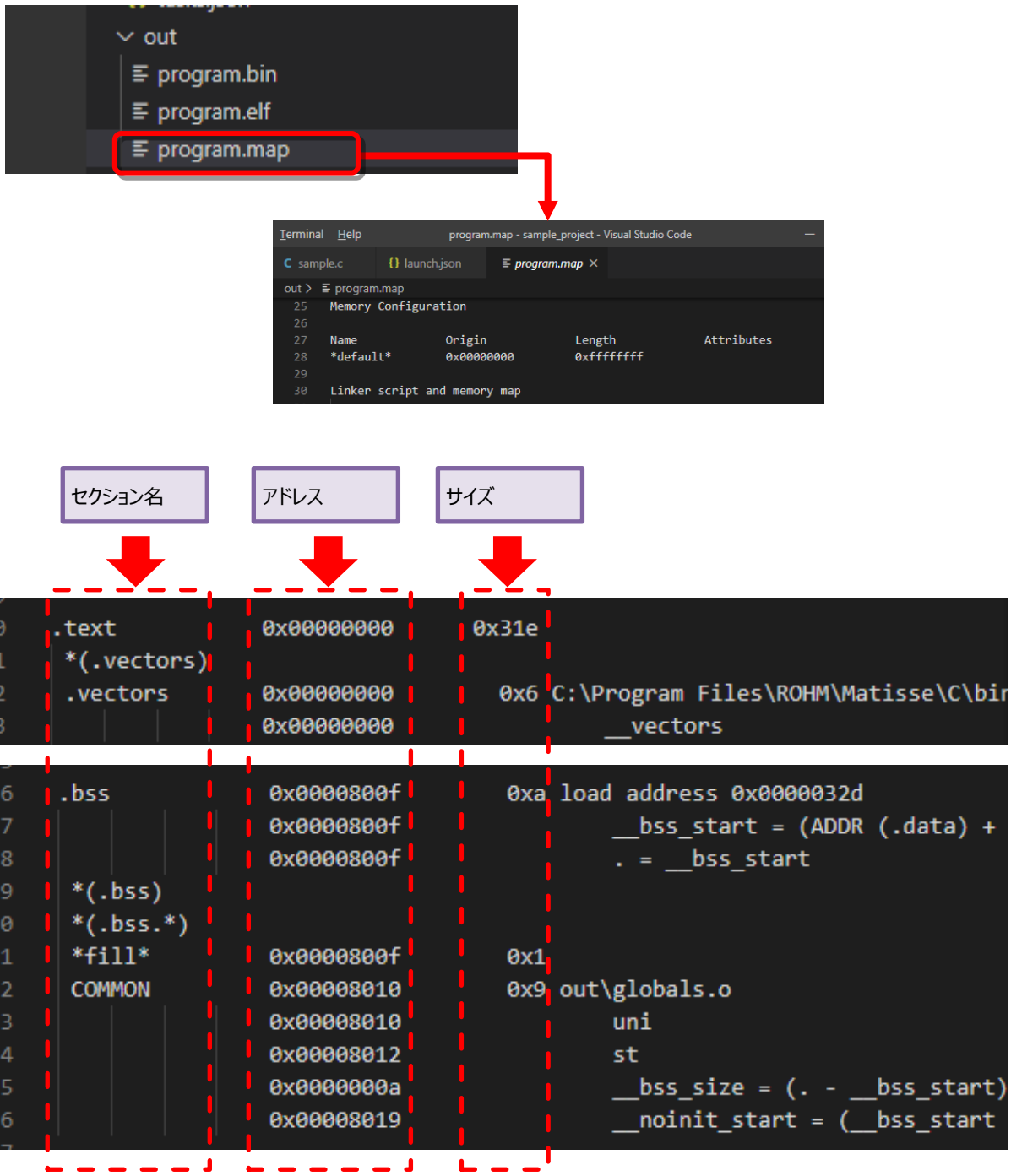


Figure 23. MAP ファイルの構成

補足
基本的に、.text が ROM（関数、定数テーブルなど）、.bss/.data が RAM のシンボル

© 2024 ROHM Co., Ltd.

6.3. スタック静的解析（Stack View）

Stack View は、ビルドされたプログラムに基づいてスタックの使用量を静的解析します。関数ごとのスタック使用量の最大値と、プログラム全体の使用量の最大値が表示されるため、スタックオーバーフローの事前防止に役立ちます。また、関数間の呼び出し関係も確認することができます。

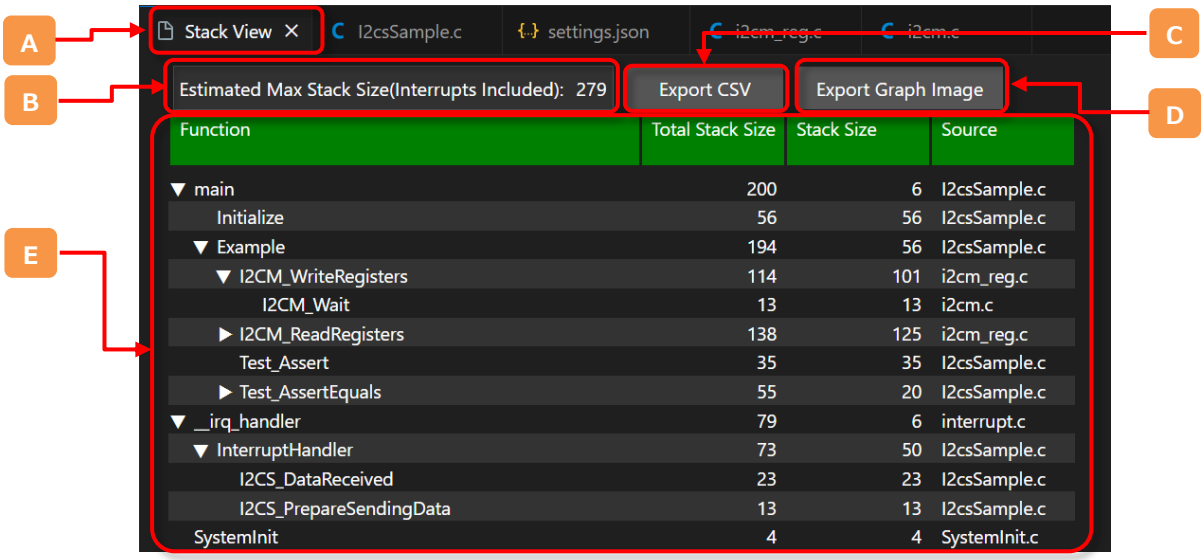


Figure 24. Stack View 画面

■起動方法：コマンドパレットから「Matisse Analysis: Start Stack Analysis」を選択すると、スタックの解析処理が開始します。処理が完了すると、エディタ画面左側に Stack View が表示されます。

「コマンドパレットの表示とコマンド入力」を参照。

Table 10. Stack View 画面の説明

	名称	説明	
A	Stack View タブ	エディタ画面の右側に表示されます。未保存のデータがある状態でクローズボタンを押すと、 ファイル保存確認のポップアップウィンドウが表示されます。データの保存は「Export CSV」を利用してください。	
B	最大スタック使用量	プログラム全体でのスタック使用量の最大値が表示されます。この値は、main 関数と割り込みハンドラ(_irq_handler、_nmi_handler)のスタック使用量の最大値の合計です。	
C	Export CSV	E「スタック使用量の詳細テーブル」が CSV 形式のファイルで出力されます。	
D	Export Graph Image	E 中の「Function」がコールグラフとして、PNG 形式の画像ファイルで出力されます。	
E	スタック使用量の詳細テーブル	列名	説明
		Function	関数の呼び出し関係がツリー形式で表示されます。 **補足：**他のいずれの関数からも呼び出されない関数は、ルートノードとして表示されることになります。
		Total Stack Size	その関数自身の「Stack Size」に加えて、その関数が呼び出すすべての関数の中で最もスタック使用量が大きい関数の「Total Stack Size」を合計した値が表示されます。
		Stack Size	関数自身のスタック使用量の最大値が表示されます。
		Source	関数が定義されているファイル名が表示されます。

⚠ **注意：**コンパイラの最適化が有効になっている場合、関数のインライン化によりコールグラフが実際のプログラムと異なることがあります。

© 2024 ROHM Co., Ltd.

6.4. ビルド結果解析（Build Analysis View）

Build Analysis View はビルド完了時に生成される.elf ファイルおよび.map ファイルを解析し、ROM および RAM のメモリ使用量を可視化する機能です。本機能を利用することで、プログラムのメモリ使用状況を把握し、最適化の検討に役立てることができます。
表示には、「サマリ表示」と「詳細解析ビュー」の 2 つの画面があります。

```
================================================================================================================
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(詳細解析ビュー)

6.4.1. サマリ表示

サマリ表示には以下の機能があります。

・ROM / RAM の使用量および使用率を表示します。
・対象となる .elf ファイルまたは .map ファイルが存在しない場合、サマリ表示は表示されません。
・ビルドによってファイル情報が更新されると、表示内容が自動的に更新されます。
・サマリ表示をクリックすると、詳細解析ビューが表示されます。

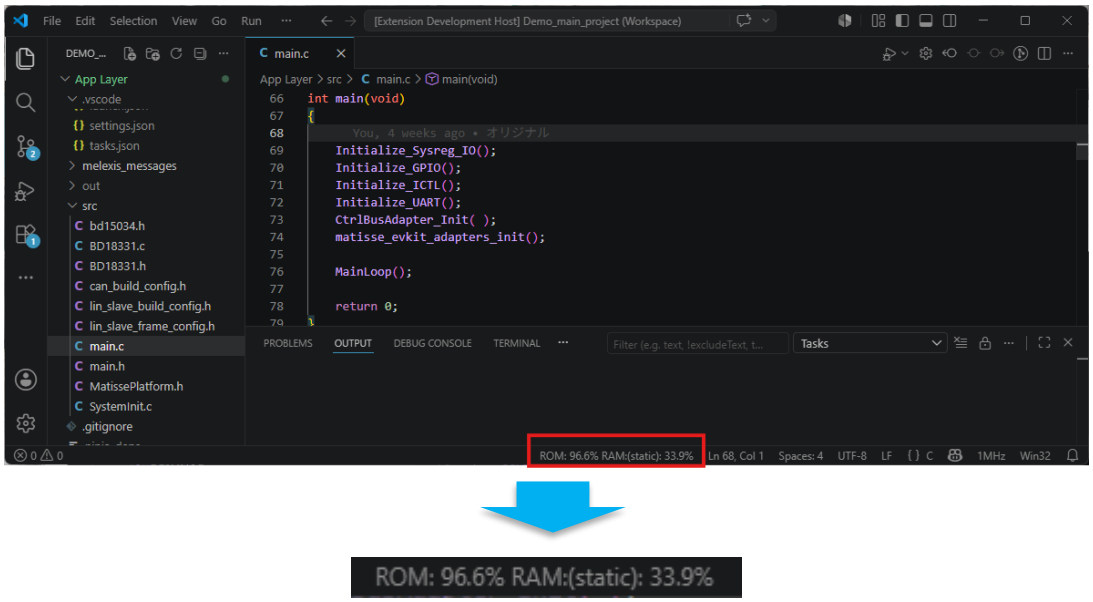


Figure 26. サマリ表示

・サマリ表示にマウスカーソルを重ねると、ツールチップで各メモリ領域の詳細情報を確認できます。

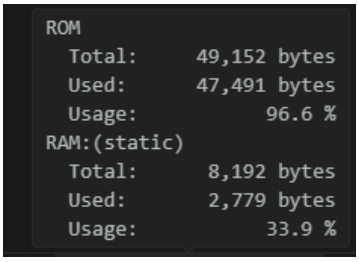


Figure 27. サマリ表示(ツールチップ)

Table 11. サマリ表示(ツールチップ)の項目説明

項目名	説明
Total	各メモリ領域の総容量。
Used	各メモリ領域の使用量。
Usage	各メモリ領域の使用率。

6.4.2. 詳細解析ビュー

詳細解析ビューは、「サマリ表示」または Sidebar View の「Build Analysis」をクリックすると、ビルド完了後に表示されます。

詳細解析ビューは、次の 4 つのセクションで構成されます。

1. Memory Usage Summary: ROM / RAM の使用量、使用率、アドレス範囲を一覧表示します。
2. Stack Usage Analysis: 最大使用量やマージンなど、スタック使用量の解析結果を表示します。
3. Memory Details: 各シンボルのアドレス、サイズ、コンパイル時のセクション、生成元ファイルを一覧表示します。
4. Object File Memory Details: オブジェクトファイルごとの ROM/RAM 使用量を一覧表示します。

6.4.2.1. Memory Usage Summary

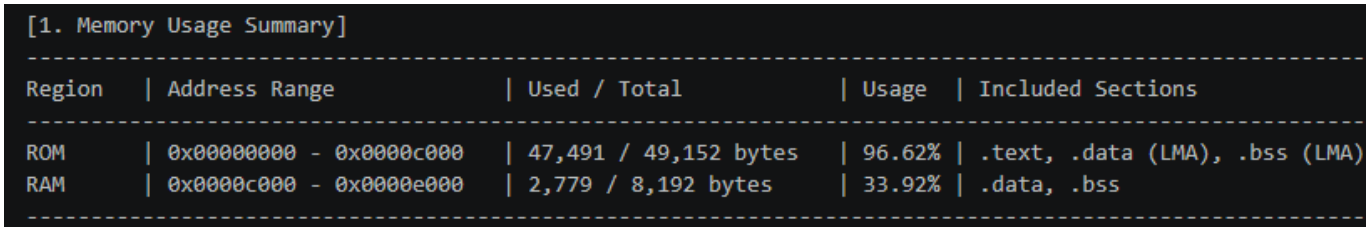


Figure 28. 詳細解析ビュー(Memory Usage Summary)

Table 12. Memory Usage Summary の項目説明

項目名	説明
Region	メモリ領域の種別（ROM/RAM）。
Address Range	該当メモリ領域のアドレス範囲。リンカスクリプトで定義されます。 **補足：**RAM の終了アドレスは、map ファイルの"__data_end"キーワードから取得されます。この値は、「settings.json」の"matisse.C.build.dataSize"設定項目で変更できます。 「コンパイラ・デバッガ設定項目（settings.json）」を参照。
Used/Total	使用量および総容量。

© 2024 ROHM Co., Ltd.

Usage	使用率。
Included Sections	リンカにより、実行ファイル内の各メモリ領域（ROM / RAM）へ配置された出力セクションの一覧。 出力セクションとは、複数のオブジェクトファイル内の入力セクションを、リンカスクリプトに従って統合した最終的なセクションを指します。

6.4.2.2. Stack Usage Analysis

```
[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 29. 詳細解析ビュー(Stack Usage Analysis)

Table 13. Stack Usage Analysis の項目説明

項目名	説明
Stack Capacity	スタック領域の総容量およびアドレス範囲。開始アドレスは「RAM 開始アドレス + ビルド後の RAM 使用量」。
Max Stack Usage	静的解析によるスタック最大使用量（Stack View から取得）。
Margin	実行時のスタックオーバーフローを防ぐために確保されている、RAM 領域の理論上のマージンサイズ。多段割込みケースは非考慮。Stack Capacity - Max Stack Usage により算出。
Evaluation	Margin の評価結果。Margin が 0 以上の場合は“Safe”、負の値の場合は“Capacity Exceeded”と表示。

6.4.2.3. Memory Details

```
[3. Memory Details]
--------------------------------------------------------------------------------------------------------
Address    |    Size | Symbol                                          | Section    | File
--------------------------------------------------------------------------------------------------------
0x00000000 |       6 | __vectors                                       | .vectors   | C:\Program Fil
0x00000006 |       2 | __default_handler                               | .handlers  | C:\Program Fil
0x00000008 |      18 | __init                                          | .init0     | C:\Program Fil
0x0000001a |       4 | __exit                                          | .init0     | C:\Program Fil
0x0000001e |      58 | __irq_handler                                   | .handlers  | out\main.o
0x00000058 |      54 | __do_copy_data                                  | .init2     | C:\Program Fil
0x00000090 |     118 | SystemInit                                      | .text      | out\SystemInit
```

Figure 30. 詳細解析ビュー(Memory Details)

Table 14. Memory Details の項目説明

項目名	説明
Address	シンボルの配置アドレス。
Size	シンボルのサイズ。
Symbol	シンボル名（関数名や変数名）。map ファイル上に表示されない Static シンボルも表示されます。
Section	各シンボルが属する入力セクション。入力セクションとは、各オブジェクトファイル（.o）内に定義されるセクションを指します。

© 2024 ROHM Co., Ltd.

File	生成元のオブジェクトファイル。

6.4.2.4. Object File Memory Details

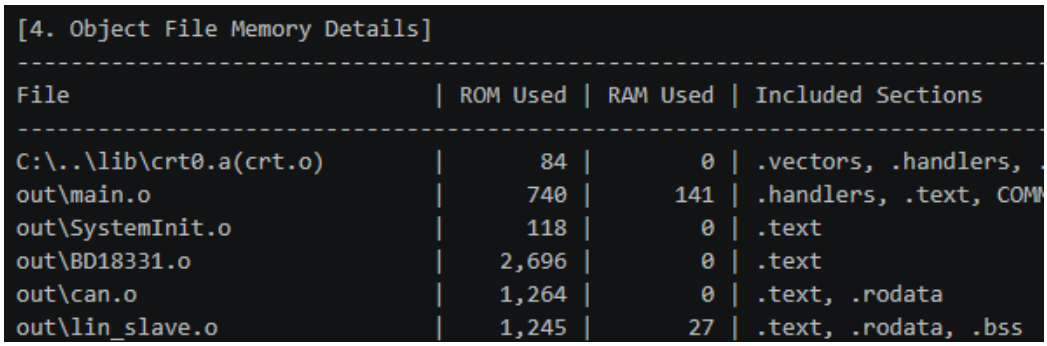


Figure 31. 詳細解析ビュー(Object File Memory Details)

Table 15. Memory Details の項目説明

項目名	説明
File	対象のオブジェクトファイル名
ROM Used	対象のオブジェクトファイルが使用する ROM 容量
RAM Used	対象のオブジェクトファイルが使用する RAM 容量
Included Sections	対象のオブジェクトファイルに含まれる入力セクション。入力セクションとは、各オブジェクトファイル（.o）内に定義されたセクションです。

補足：詳細解析ビューの出力を保存する場合は、表示画面をクリックし、Ctrl＋S を押してください。

⚠注意

・コンパイラの最適化オプションが有効な場合、最適化によりインライン化されたシンボルは一覧に表示されません。

・本機能で表示されるメモリ領域情報は、ビルド時点の解析結果です。実行時のメモリ配置は実機環境や設定によって異なる場合があります。

・プログラムのメモリ領域の配置方法は、リンカスクリプトで定義されます。メモリ領域の位置やサイズを変更する必要がある場合は、リンカスクリプトを変更してください。詳細は **mtld ユーザーズガイド**を参照してください。

© 2024 ROHM Co., Ltd.

7. デバッグ

7.1. デバッグ画面の説明

デバッグ画面を表示する場合は、（Debug and Run）を選択します。

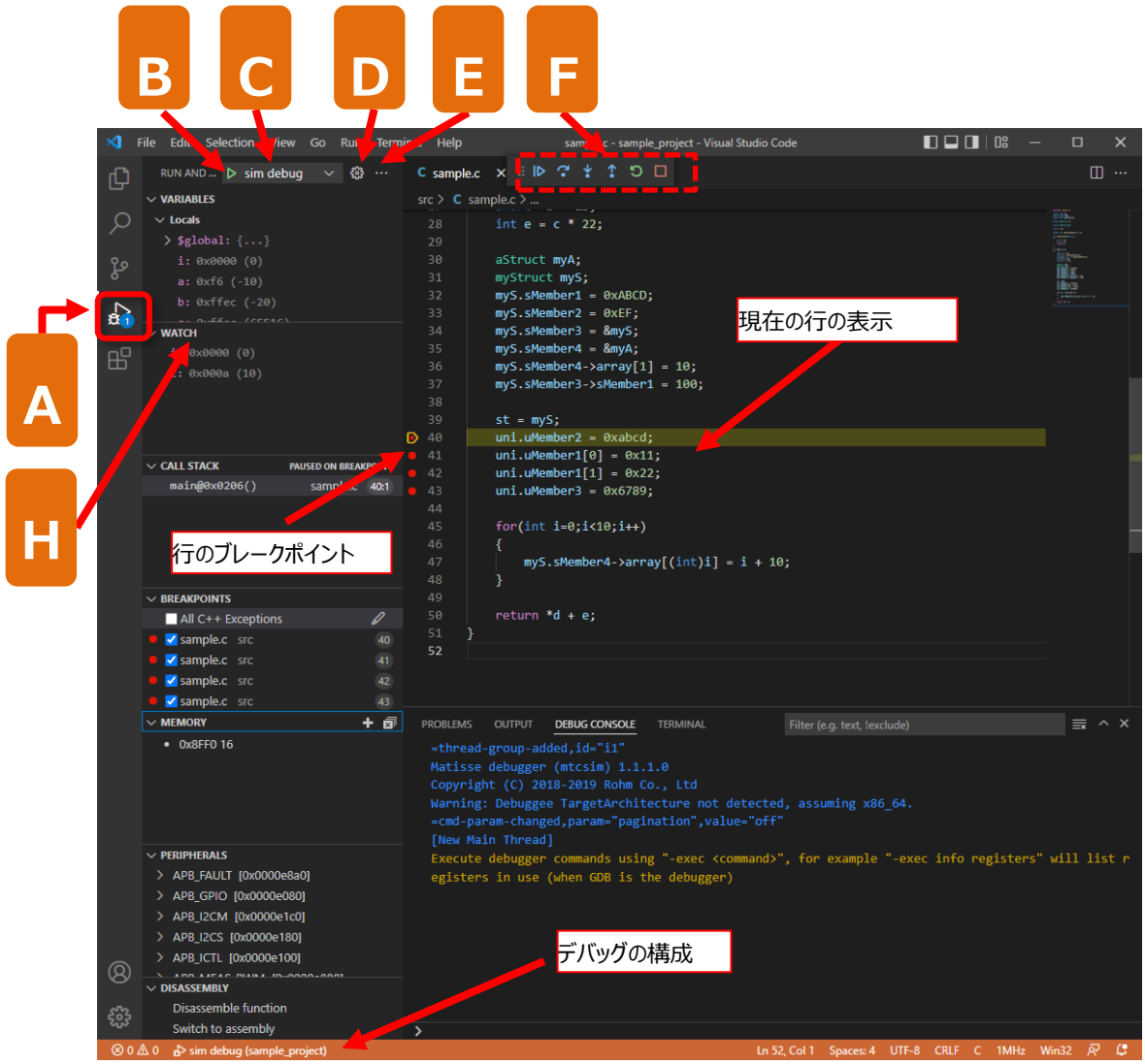


Figure 32. Debug and Run 選択時の画面

Table 16. Debug and Run の説明

	名称	説明
A	Debug and Run	デバッグ画面を表示します。
B	Start Debugging	デバッグを開始します。
C	デバッグモードの選択	「board debug」（ハードウェア接続時）または「sim debug」（ハードウェア未接続時）を選択して、表示されているモードでデバッグが開始されます。
D	Open launch.json	launch.json ファイルが作成／表示され、デバッグの構成を変更できます。
E	View Settings	サイドバーの表示設定、デバッグ コンソールの表示オンオフ設定ができます。
F	デバッグツールバー	「デバッグツールバー」を参照。

© 2024 ROHM Co., Ltd.

次ページに続く

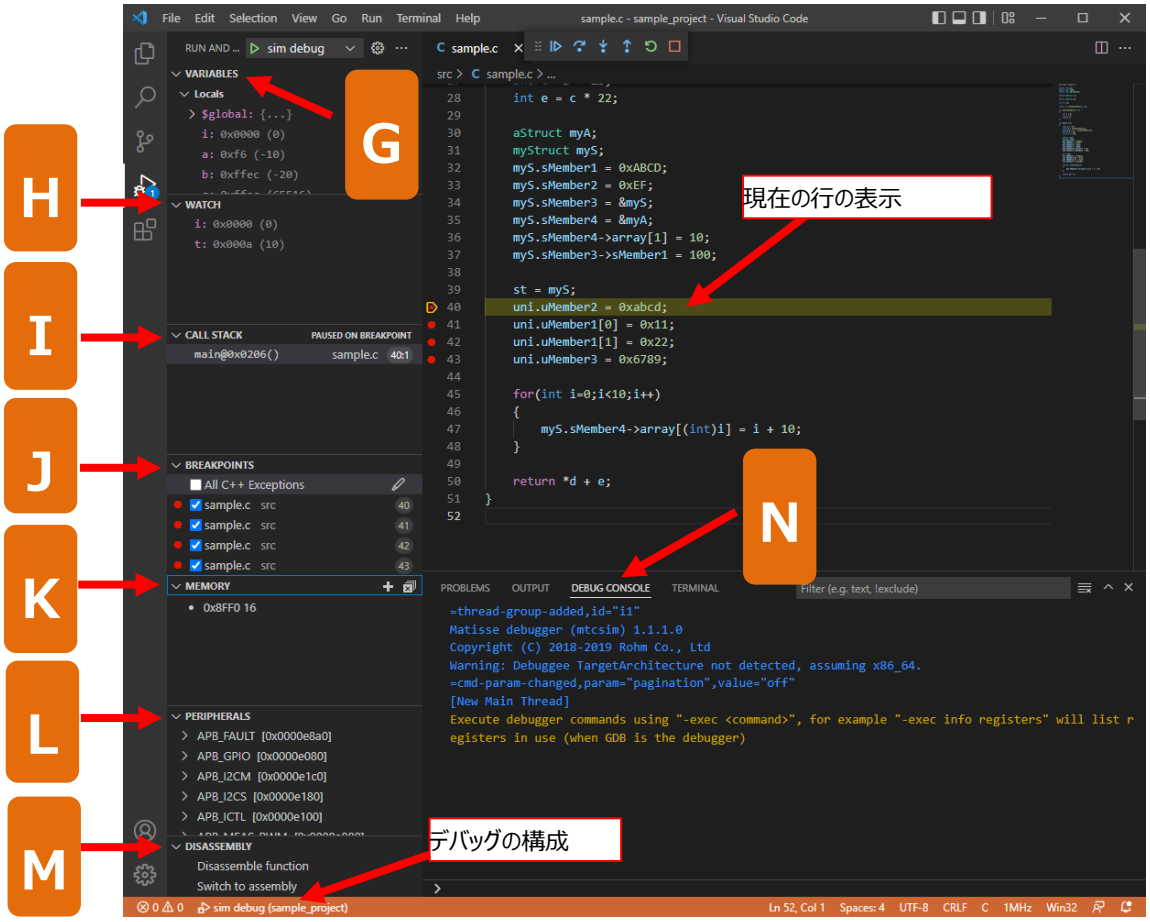


Figure 33. Debug and Run 選択時の画面

Table 17. Debug and Run の説明

	名称	説明
G	VARIABLES	変数名と値を表示します。
H	WATCH	「データインスペクション」を参照。
I	CALL STACK	「コールスタック（CALL STACK）」を参照。
J	BREAKPOINTS	「ブレークポイント（BREAKPOINTS）」を参照。
K	MEMORY	「MEMORY」を参照。
L	PERIPHERALS	「PERIPHERALS」を参照。
M	DISASSEMBLY	「DISASSEMBLY」を参照。
N	デバッグコンソール	デバッグのログを表示します。

© 2024 ROHM Co., Ltd.

7.2. デバッグメニュー

ツールバーの「Debug」からデバッグの項目を操作することもできます。

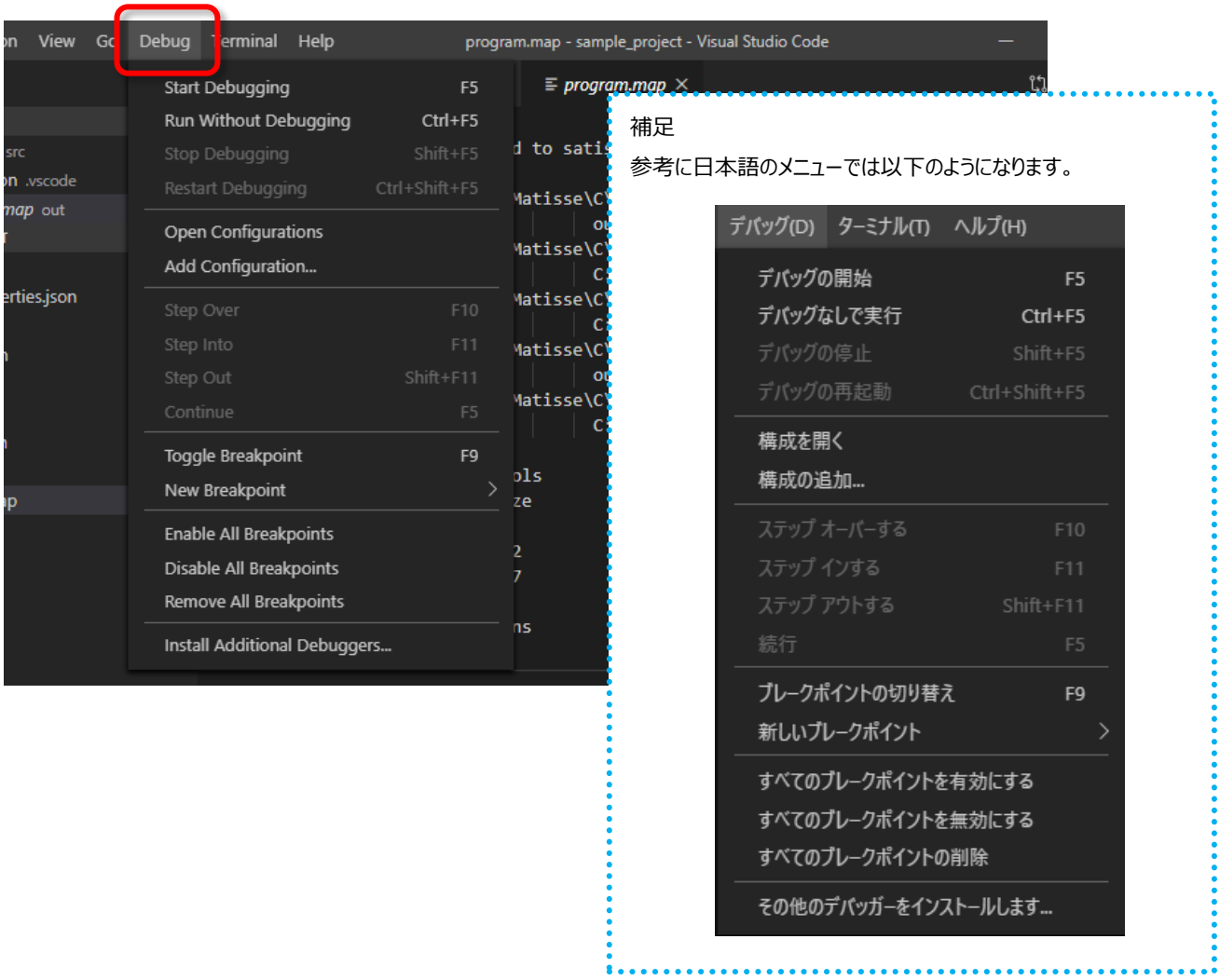


Figure 34. Debug メニュー

© 2024 ROHM Co., Ltd.

7.3. ブレークポイント（BREAKPOINTS）

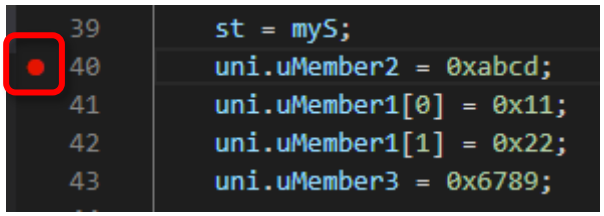


Figure 35. ブレークポイント

コードの左をクリックすることでブレークポイントを設定することができます。
ブレークポイントのコードが実行される前にブレイク（プログラムが停止）します。

補足
そのソースコードに相当するバイナリが読まれていないなど、ブレークポイントの設定が不可能な時には、マークが灰色で表示されます。

ブレークポイント（BREAKPOINTS）のウィンドウ

ブレークポイント（BREAKPOINTS）のウィンドウでは、個別のブレークポイントのオンオフや、ブレークポイントの一括のオンオフもしくは削除の設定ができます。

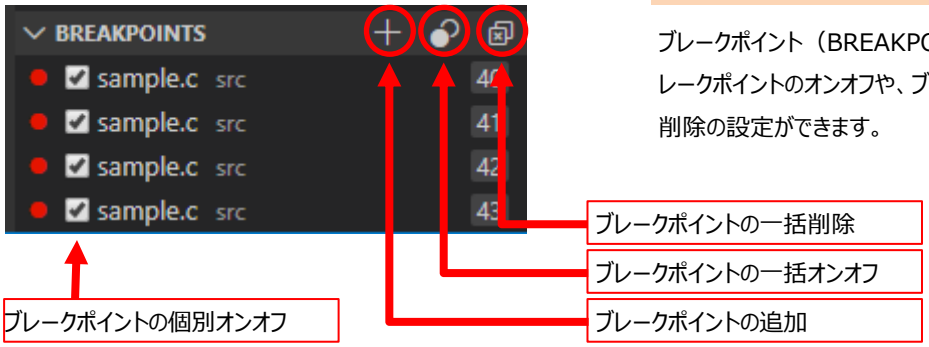


Figure 36. BREAKPOINTS ウィンドウ

7.4. デバッグツールバー

デバッグを開始するとエディタの上部にデバッグツールバーが表示されます。

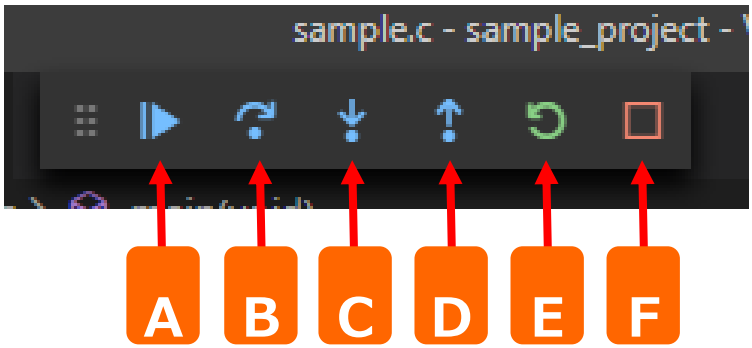


Figure 37. デバッグツールバー

■左のボタンは次のようにステップ実行の機能を持ちます。

Table 18. デバッグツールバー 左ボタンの説明

	名称	キーボードショートカット	説明
A	Continue（続行）	F5	次のブレークポイントまで進みます。
	Pause（中断）	F6	ステップ実行のため、停止させます。 言語によって有効な場合、表示されます。
B	Step Over（ステップオーバー）	F10	現在の行を実行し、次の行に進みます。
C	Step Into（ステップイン）	F11	現在の行の実行が関数の場合、その中に進みます。
D	Step Out（ステップアウト）	Shift+F11	現在実行中の関数を抜けるまで実行します。

■右のボタンはデバッグ実行全体に関する機能を持ちます。

Table 19. デバッグツールバー 右ボタンの説明

	名称	キーボードショートカット	説明
E	Restart（再起動）	Ctrl+Shift+F5	デバッグを再起動します。
F	Stop（停止）	Shift+F5	デバッグを終了します。

© 2024 ROHM Co., Ltd.

7.5. データインスペクション

ステップ実行でプログラムが停止している時、各変数の値を確認したり更新したりすることを、データインスペクションといいます。
VARIABLES（変数）のウィンドウと、 WATCH （ウォッチ式）のウィンドウ、またはコード中の変数にマウスオーバーした時のポップアップがこの機能を持ちます。

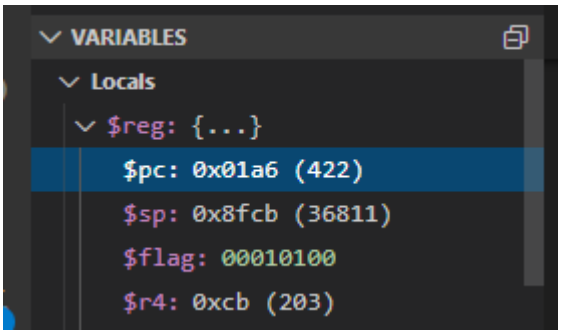


Figure 38. VARIABLES（変数）ウィンドウ

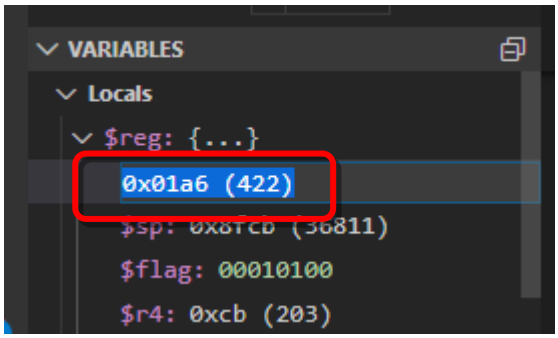


Figure 39. 値の編集

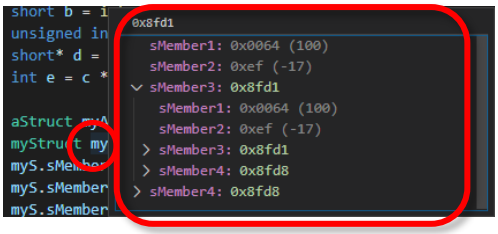
VARIABLES（変数）ウィンドウ

VARIABLES（変数）のウィンドウは、現在選択されているタスク、コールスタックにおける、変数の値の中身を確認することができます。
対象の変数がクラス、構造体等の場合、ツリー形式で各プロパティの詳細を見ることができます。

「コールスタック（CALL STACK）」を参照。

グローバル変数の表示には「WATCH（ウォッチ式）ウィンドウ」を使用してください。

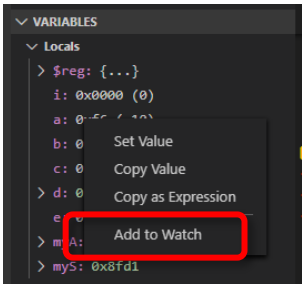
補足
ソースコード中の変数にマウスオーバーした時に、その変数の中身がポップアップ表示されます。



WATCH（ウォッチ式）ウィンドウ

WATCH （ウォッチ式）ウィンドウでは、プログラムのデバッグ中に特定の変数の値を監視する事ができます。
WATCH（ウォッチ式）ウィンドウでは監視したい変数の名前の追加と削除ができます。

補足
変数の値を変更するには VARIABLES（変数）ウィンドウを使用してください。WATCH（ウォッチ式）ウィンドウでは変数の値を変更することはできません。
VARIABLES（変数）ウィンドウの変数を右クリックして、「Add to Watch」を選択すると、ウォッチ式を追加する事もできます。



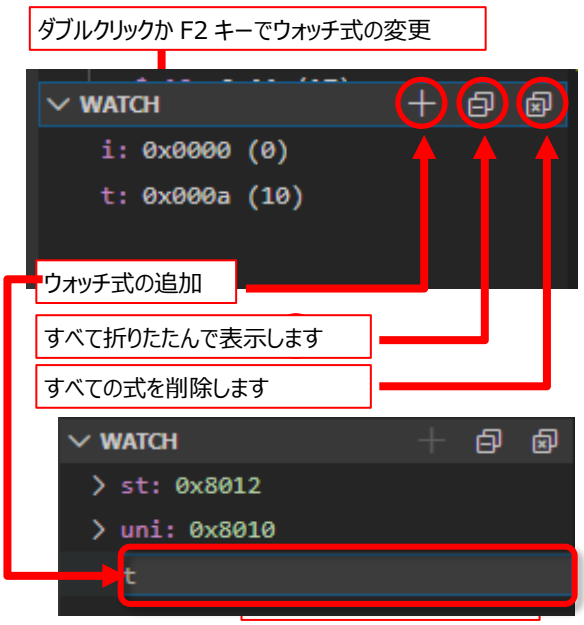
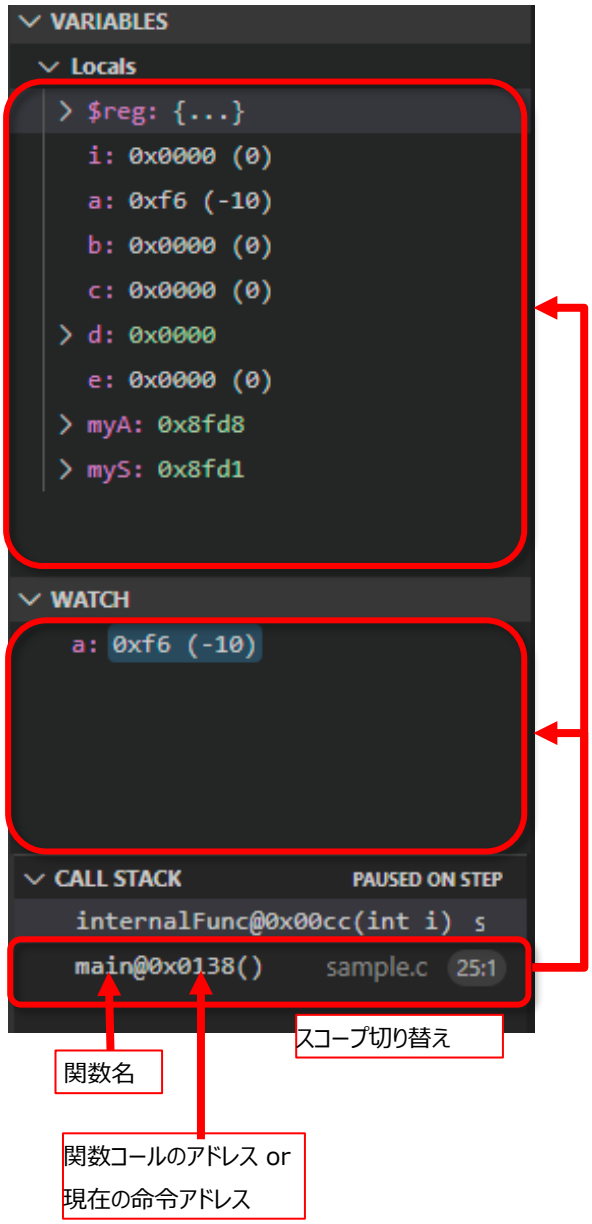


Figure 40. WATCH（ウォッチ式）ウィンドウ

© 2024 ROHM Co., Ltd.

7.6. コールスタック（CALL STACK）

コールスタック(CALL STACK)ウィンドウは関数の呼び出し履歴を逆順に表示します。



コールスタック(CALL STACK)ウィンドウに表示されている関数名をクリックすると、VARIABLE ウィンドウと WATCH ウィンドウの内容が選択された関数のスコープに切り替わります。

Figure 41. コールスタック(CALL STACK)ウィンドウ

© 2024 ROHM Co., Ltd.

7.7. MEMORY

MEMORY は、指定したメモリ範囲の値を VS Code の画面内に表示します。表示画面には、メモリの範囲と対応する値が表示されます。この機能は、デバッグ中かどうかに関係なく起動可能です。

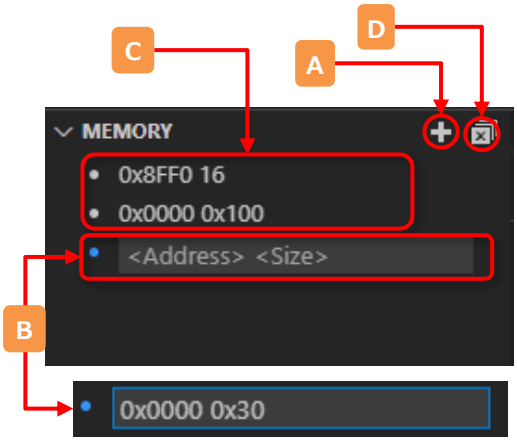


Figure 42. MEMORY

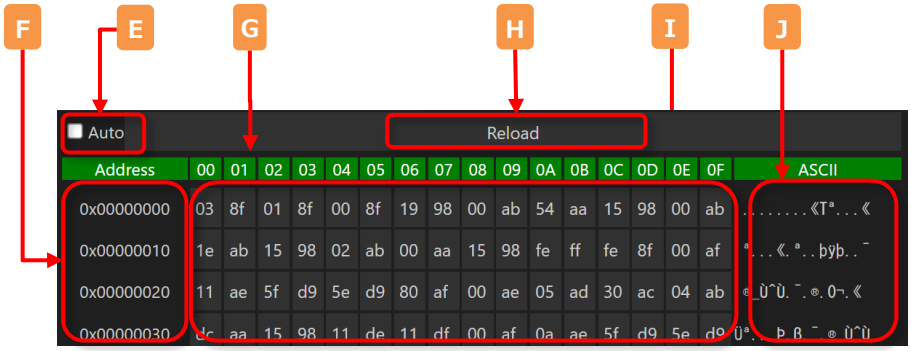


Figure 43. MEMORY 表示画面 (エディタ画面領域)

Table 20. MEMORY の説明

	名称	説明
A	メモリ表示の追加	クリックすると、「メモリ表示の追加フォーム」が開き、表示したいメモリアドレス範囲を指定できます。
B	メモリ表示の追加フォーム	表示したいメモリ範囲の先頭アドレスとサイズを入力し、Enter キーを押すと、その範囲がエディタ領域に新たに表示されます。フォームには 16 進数および 10 進数の値を入力できます。搭載しているメモリ領域を超える範囲は表示できません。
C	表示履歴	過去に表示したメモリ範囲の履歴が一覧で表示されます。クリックすると、対応するメモリ範囲がエディタ領域に再表示されます。
D	表示履歴の削除	クリックすると、過去に表示したメモリ範囲の履歴を一括で削除します。
E	オートリロード切替	チェックするとオートリロードが開始されます。チェックアウトするとオートリロードが停止されます。
F	メモリアドレス	各行のメモリの開始アドレスが表示されています。
G	メモリアドレスオフセット	メモリアドレスからのオフセットです。
H	リロードボタン	クリックすると、メモリ値をリロードし最新の値が反映されます。
I	メモリ値	各アドレスに対応するメモリ値が表示されます。ダブルクリックすると、値を編集できます。リロードを行うと、前回の値から変化した値が赤字で表示されます。なお、変化した値がない場合、前回赤字表示だったメモリはそのまま赤字で表示され続けます。 **補足：**コピーしたい範囲をマウスドラッグで選択し、Ctrl＋C キーでクリップボードにコピーできます。
J	メモリ値の ASCII 表示	メモリ値が ASCII 形式で表示されます。

補足：デバッグ中かどうかに関係なくボードを接続すると起動可能です。

⚠注意：MEMORY および PERIPHERALS のオートリロードでの同時実行は、画面に表示されているタブの範囲で最大 2 つまでに制限してください。複数の同時実行を行うと、通信負荷が増大し、通信エラーが発生する恐れがあります。

© 2024 ROHM Co., Ltd.

7.7.1. メモリパターンフィル

指定したメモリ範囲に、任意のパターンを一括で書き込むことができる機能です。

1．メモリ値を右クリックし、メニューから「Fill Memory」を選択します。

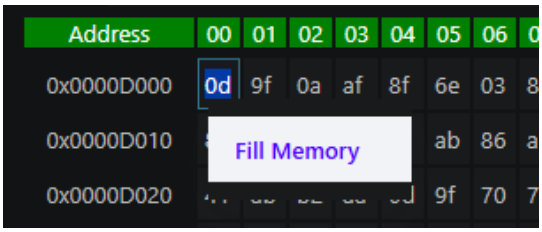


Figure 44. Memory View の右クリックメニュー

2．表示されたコマンドパレットの入力フォームに、 <開始アドレス> <終了アドレス> <パターン(1～4 バイト)> を入力し、Enter キーを押します（例: 0x0000d000 0xd030 0xff）。開始アドレスには、右クリックしたセルに対応するアドレスが自動的に入力されます。自動入力されたアドレスは変更できます。

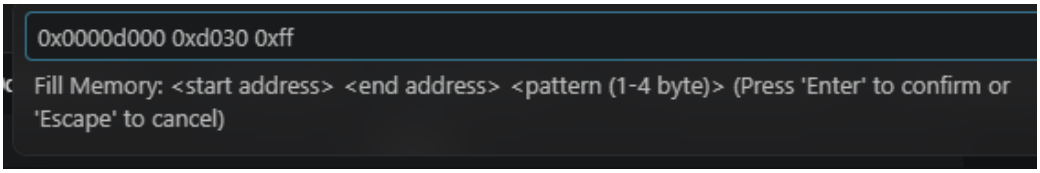


Figure 45. メモリフィル範囲入力フォーム

3．書き込み方向(低位アドレス→高位アドレス / 高位アドレス→低位アドレス)を選択します。

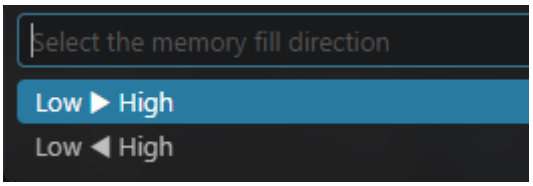


Figure 46. 書き込み方向選択リスト

4．書き込みが終わると、値を読み戻して検証し、結果が表示値へ反映されます。

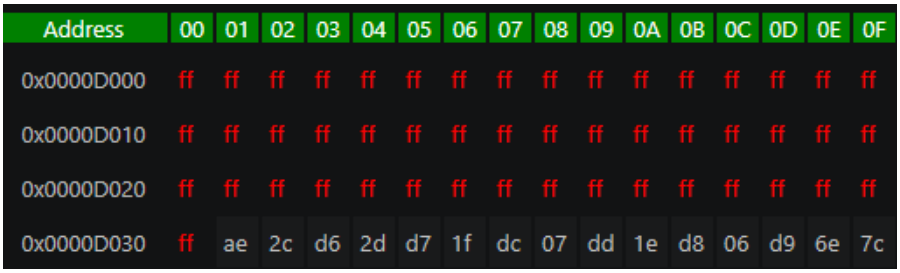


Figure 47. メモリフィル実行結果

なお、不一致があった場合は、通知メッセージの「Show Details"」をクリックすると、不一致のアドレスを確認できます。

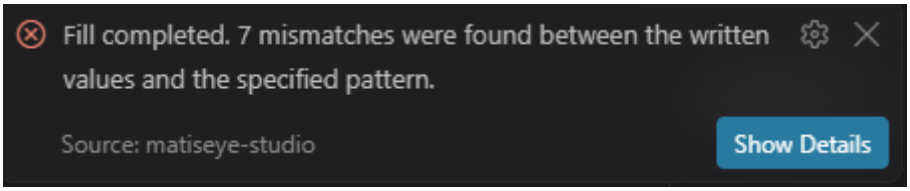


Figure 48. 書き込み結果不一致時のメッセージ

© 2024 ROHM Co., Ltd.

7.7.2. シンボルオーバーレイ

メモリ値にマウスカーソルを合わせると、そのアドレスにあるグローバル変数の範囲がハイライト表示されます。あわせて、ツールチップに変数の情報が表示されます。構造体の場合は、カーソルのアドレスを含む最下層のメンバの範囲と情報が表示されます。



Figure 49. シンボルオーバーレイ



Figure 50. 右クリックメニュー

Table 21. シンボルオーバーレイの説明

	名称	説明
A	ホバーハイライト	カーソルを合わせたアドレスを含む変数（構造体の場合は最下層のメンバ）の範囲が、ハイライトされます。カーソルを移動すると解除されます。
B	固定ハイライト	固定されたハイライトです。下線付きで表示され、カーソルを移動しても残ります。複数設定できます。固定は F、解除は G、H で行います。
C	変数名	ツールチップの 1 行目に、変数名が表示されます。構造体のメンバの場合は、「変数名.メンバ名」の形式で表示されます。
D	Type / Address / Size	型、アドレス範囲、サイズです。
E	Members	共用体の場合のみ、すべてのメンバが「メンバ名(型名)」の形式で表示されます。
F	Keep Highlight	ホバーハイライトの範囲を右クリックすると表示されます。選択すると、その範囲を固定ハイライトにします。
G	Clear Highlight	固定ハイライトの範囲を右クリックすると表示されます。選択すると、その範囲の固定を解除します。
H	Clear All Highlights	固定ハイライトが 1 つ以上あると表示されます。選択すると、すべての固定を解除します。

© 2024 ROHM Co., Ltd.

補足：

・配列は要素ごとではなく配列全体が 1 つの範囲として表示されます。

・型名やメンバ名が未定義の場合は、<anonymous>と表示されます。

・ELF ファイルの解析に失敗した場合や、グローバル変数が存在しない場合は、この機能は使用できません。

© 2024 ROHM Co., Ltd.

PERIPHERALS

PERIPHERALS は、ペリフェラル情報を画面に表示し、その値を読み書きすることができます。ペリフェラル情報は、設定ファイルで指定された CMSIS-SVD 方式（<○○>と表記）の SVD ファイルに基づいて表示されます。この機能は、デバッグ中かどうかに関係なく起動可能です。

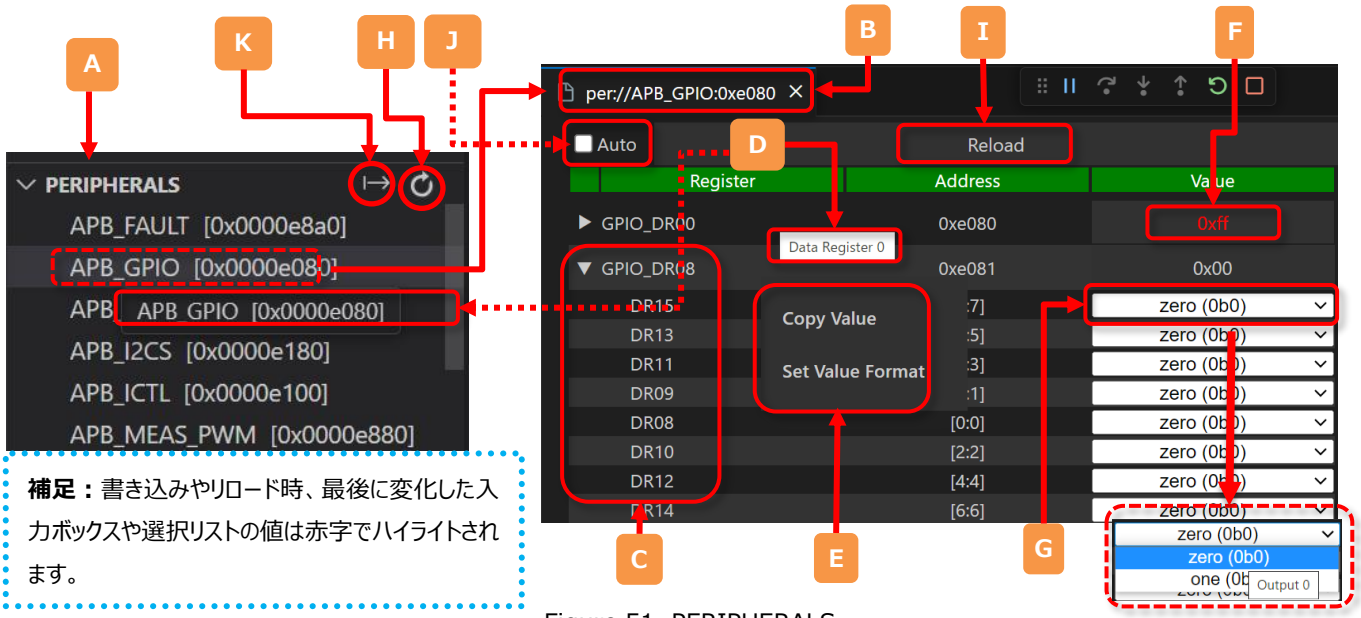


Figure 51. PERIPHERALS

Table 22. PERIPHERALS の説明

	名称	説明
A	ペリフェラル選択リスト	ペリフェラル名をダブルクリックすると、対応するペリフェラル名のタブがエディタに追加されます。
B	ペリフェラルタブ	各タブには、ペリフェラルのレジスタリストが表示されます。レジスタは階層構造を持っています。 ・レジスタ層：レジスタ名・アドレスオフセット・値を表示 ・フィールド層：ビットフィールド名・ビット幅・値(リスト型の場合リスト名)を表示
C	ツリーの展開・折り畳み	レジスタ/フィールド名のダブルクリックまたは"▶"のマークのクリックで、展開/折り畳みできます。
D	説明ツールチップ	レジスタ/フィールド名をマウスオーバーすると、<description>の内容が表示されます。
E	ポップアップメニュー	ペリフェラル/レジスタ/フィールド名を右クリックで、以下の機能の実行メニューが表示されます。 ・"Copy Value"：レジスタ/フィールド値をクリップボードにコピー（<access>"が"write-only"の場合表示されません。） ・"Set Value Format"：レジスタ/フィールドの表示形式を 16/10/2 進数に切替 **補足** "<readAction>"レジスタの場合、"Force Read"が表示されます。
F	値入力ボックス	16/10/2 進数の値を入力してエンターキーを押すと書き込みができます。
G	値選択リスト	リストのアイテムをクリックすると値の書き込みができます。
H	レジスタ値全体更新ボタン	クリックすると、タブ全てのペリフェラルレジスタの値がリロードされ、表示が更新されます。
I	レジスタ値単体更新ボタン	クリックすると、タブ単体のペリフェラルレジスタの値がリロードされ、表示が更新されます。
J	オートリロード切替	チェックすると、I「レジスタ値単体更新ボタン」 が 1S 間隔で自動的に実行されます。
K	ファイル出力	クリックすると、アドレス範囲の入力フォームが表示され、指定範囲のペリフェラルレジスタの名前と値をファイルに出力できます。

SVD ファイルパス設定

SVD ファイルは、マークアップ言語である XML を用いて、タグ（<○○>）を使用して記述されます。ファイルパスは「settings.json」（コンパイラおよびデバッガの設定用）で設定可能です。

「プロジェクト構成と設定項目」を参照。

© 2024 ROHM Co., Ltd.

補足：

- 多バイトレジスタのデフォルトの書き込み順序は MSB から LSB です。ただし、タグ内のキーワードを使用して、この順序を LSB から MSB に変更することが可能です。また、バイトオーダーはリトルエンディアンです。
- デバッグ中かどうかに関係なくボードを接続すると起動可能です。

⚠注意：MEMORY および PERIPHERALS のオートリロードでの同時実行は、画面に表示されているタブの範囲で最大 2 つまでに制限してください。複数の同時実行を行うと、通信負荷が増大し、通信エラーが発生する恐れがあります。

© 2024 ROHM Co., Ltd.

7.8. DISASSEMBLY

DISASSEMBLY は関数単位、または実行ファイル全体を対象に、逆アセンブル結果（アセンブリ命令）を表示する機能です。特に実行ファイル全体の表示(C)では、元のソースコードとアセンブリ命令を並べて確認できます。この機能は、デバッグ中のみ使用可能です。

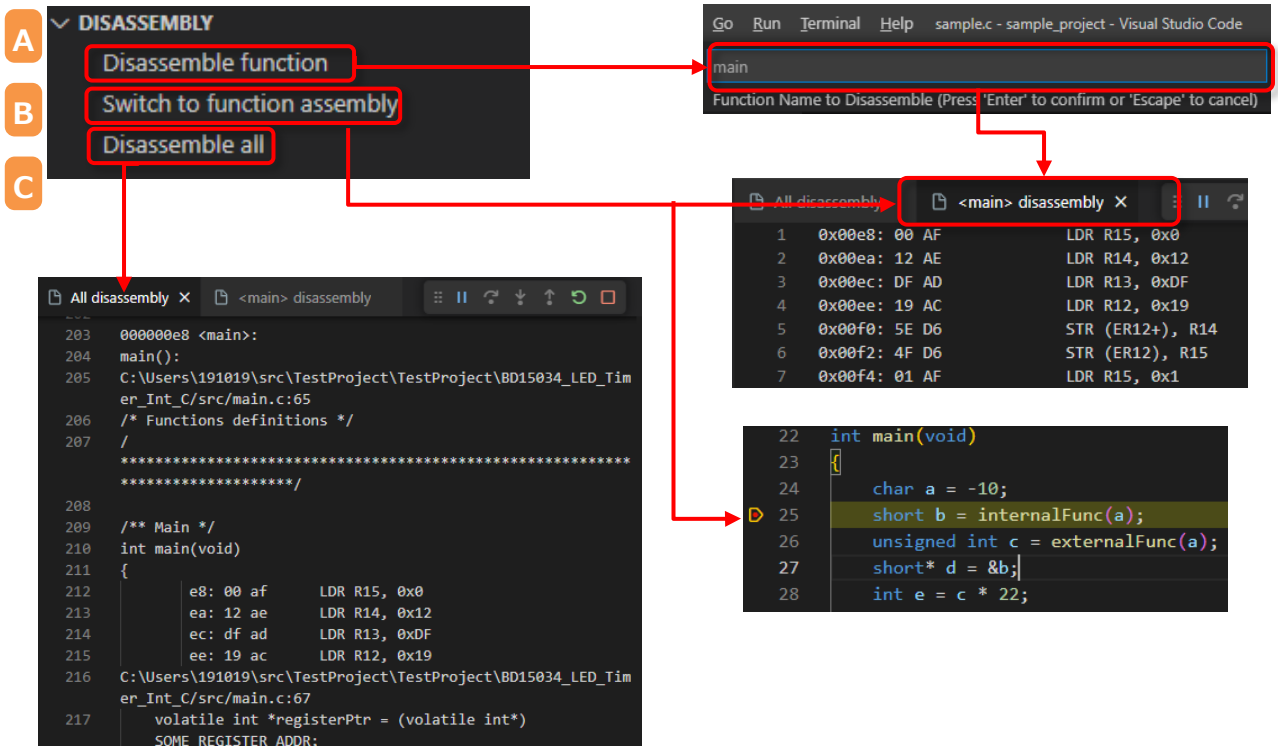


Figure 52. DISASSEMBLE の画面

Table 23. DISASSEMBLE 画面の説明

	名称	説明
A	Disassemble function ボタン	関数を指定して逆アセンブル結果を表示するボタンです。 クリックすると、関数名を入力するためのコマンドパレットが開きます。逆アセンブルしたい関数名を入力して Enter キーを押すと、その関数の逆アセンブル結果が「<関数名> disassembly」タブで新しく表示されます。 **補足：**関数名を入力しても結果が表示されない場合、コンパイラ最適化により関数がインライン展開されている可能性があります。
B	Switch to function assembly / Switch to function source ボタン	現在開いている関数の「アセンブリ表示」と「ソースコード表示」を切り替えるトグルボタンです。 押すたびに、ボタンの名称も「Switch to function source」と「Switch to function assembly」に変化します。 **補足：**実行中の関数がインライン化されている場合、その呼び出し元の関数の逆アセンブル結果が表示されます。
C	Disassemble all ボタン	実行ファイル全体の逆アセンブル結果を表示するボタンです。 クリックすると、実行ファイル全体のアドレス範囲に対応した逆アセンブル結果が「All disassembly」タブで新しく表示されます。この表示では、アセンブリ命令と、それに対応するソースコード(ファイル名、行番号付き)が混在して表示されます。

補足：この機能はデバッグ中のみ起動できます。

© 2024 ROHM Co., Ltd.

7.9. PERFORMANCE

PERFORMANCE はプログラム内の各関数の実行時間の割合を表示する機能です。デバッグ中に、定期的にプログラムカウンタ(PC)の位置をサンプリングし、その時点で実行中の関数のサンプル数が加算されます。これにより、各関数の実行割合の確認や処理負荷の高い関数を特定することができ、システム全体の性能向上を効率的に進めることができます。

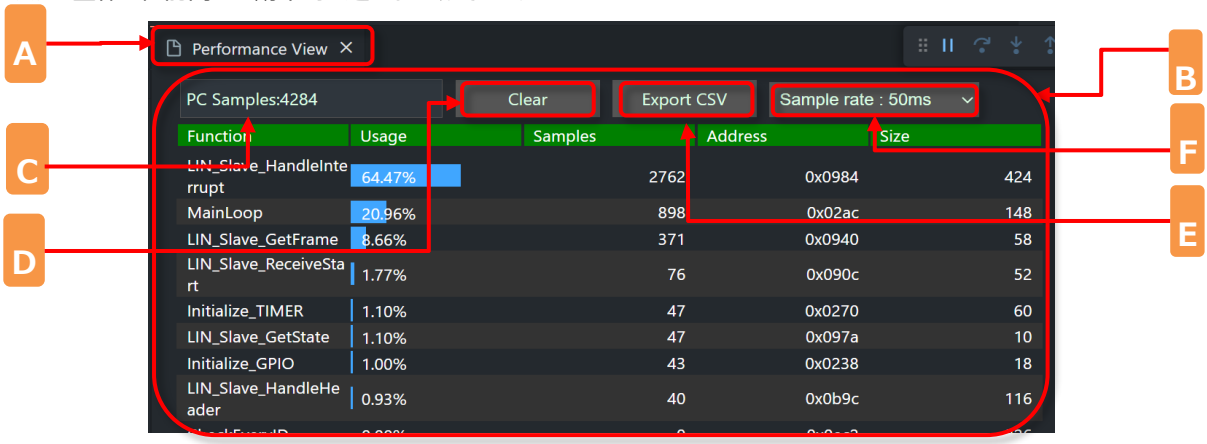


Figure 54. PERFORMANCE の画面

■起動方法：設定ファイル(settings.json)で起動を有効にするか、またはデバッグ中にコマンドパレットから「Matisse Debug: Start Performance View」を選択すると、起動できます。起動すると上記画面が表示され、サンプリングが開始されます。また、設定ファイルで起動時のサンプリングレートも指定できます。

「プロジェクト構成と設定項目」を参照。

■停止方法：ブレーク・一時停止・ステップ実行時は、サンプリングが中断されます。

Table 24. PERFORMANCE 画面の説明

	名称	説明	
A	Performance View タブ	エディタ画面の右側に表示されます。未保存のデータがある状態でクローズボタンを押すと、ファイル保存確認のポップアップウィンドウが表示されます。データの保存は「ファイル保存ボタン」を利用してください。	
B	PERFORMANCE テーブル	1s 間隔で画面表示が更新されます。テーブルの情報を以下に示します。	
		列名	説明
		Function	プログラム内の関数名が一覧で表示されます。表示順はサンプル数の多い順になっており、サンプル数の変化に応じて自動更新されます。
		Usage	サンプル総数に対する各関数のサンプル数の比率とそのグラフが表示されます。
		Samples	サンプリング時に実行中の関数に加算されていきます。
		Address	各関数のスタートアドレスが表示されます。
		Size	各関数のサイズが表示されます。
C	サンプル総数	サンプリングされた PC の総数が表示されます。サンプリングはデバッグ中のみ(ポーズ、停止中除く)されます。	
D	クリアボタン	クリックすると、サンプル総数、使用率、サンプル数の表示が 0 にクリアされます。	
E	ファイル保存ボタン	サンプル総数とテーブル情報を CSV 形式でファイル保存できます。	
F	サンプリングレート選択リスト	プルダウンリストから以下のサンプリングレートを選択できます。変更はいつでも可能です。 サンプリングレート：30ms/50ms/70ms/100ms/200ms/500ms/1s	

⚠ **注意：** サンプリング法により、計測結果は確率論的なものとなります。精度を上げるためには、数十秒から数分間の計測が必要になります。
MEMORY や PERIPHELALS のオートリロード機能と本機能の同時使用は、正確な計測が行われない場合がございます。

© 2024 ROHM Co., Ltd.

7.10. デバッグ関連のキーボードショートカット

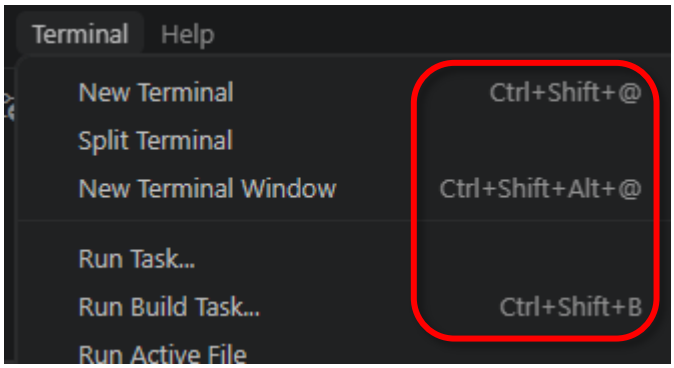
Table 25. デバッグ関連のキーボードショートカット

キー	アクション
F5	デバッグの開始
Shift+F5	デバッグの停止
Ctrl+Shift+F5	デバッグを再起動する
Ctrl+Shift+D	デバッグを開く
Ctrl+Shift+Y	デバッグコンソールを開く
F9	ブレークポイントの切り替え
Shift+F9	列 Column ブレークポイントの設定
F5	Continue（続行）
F6	Pause（中断）
F10	Step Over（ステップオーバー）
F11	Step Into（ステップイン）
Shift+F11	Step Out（ステップアウト）

補足

メニューの右側にショートカットが表示されていますので、

参考にしてください。



© 2024 ROHM Co., Ltd.

7.11. メモリウィンドウ（Memory Window）

メモリとペリフェラルのレジスタの読み書きができます。

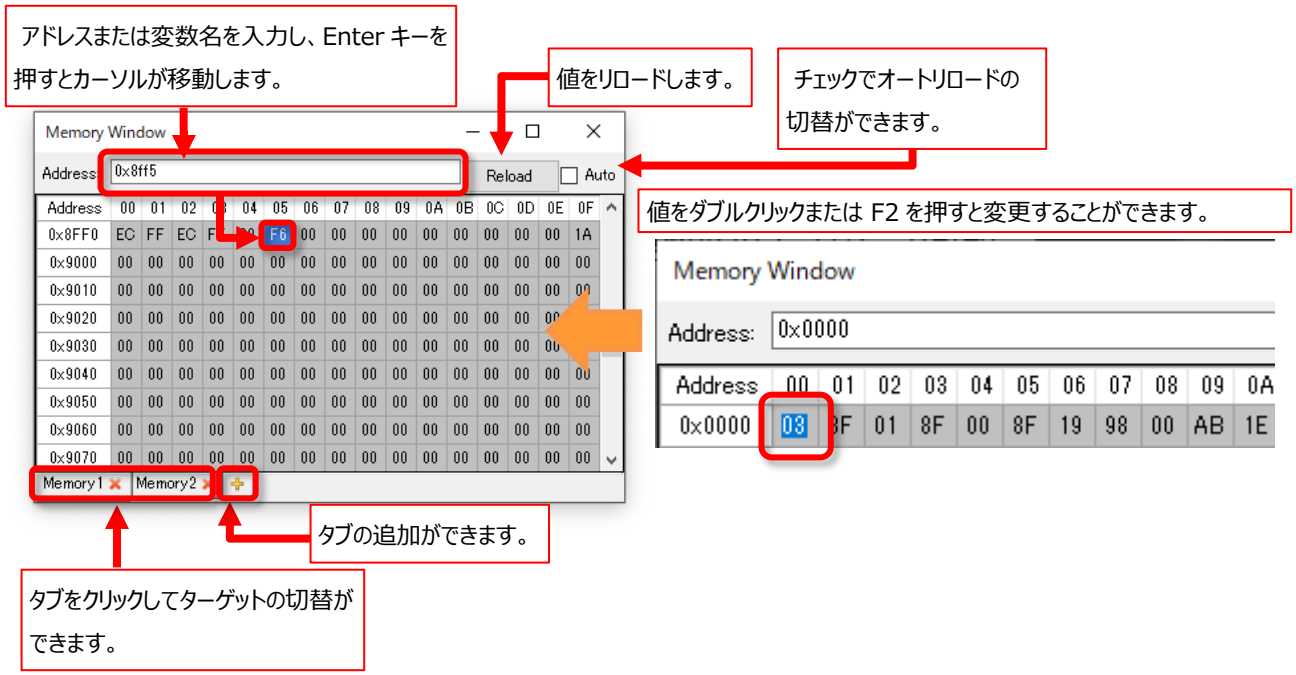
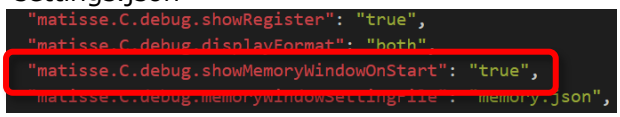
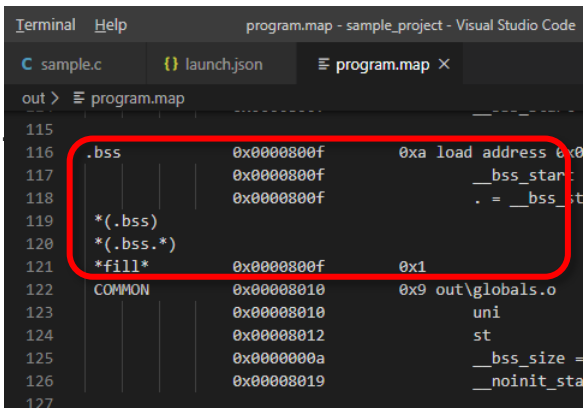


Figure 55. Memory Window 画面

settings.json





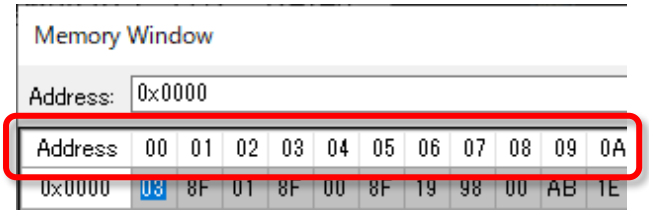


Figure 56. Memory Window の表示設定

Memory Window 表示設定

settings.json（コンパイラ・デバッガ設定）の「 matisse.C.debug.showMemoryWindowOnStart」を「true」に設定している場合、デバッグ開始時にメモリウィンドウを開きます。

「プロジェクト構成と設定項目」を参照。

補足

.map ファイルでグローバル変数のアドレスが確認できます。

.data 領域か.bss 領域の欄にグローバル変数が表示されます。

（static 変数は MAP ファイルに表示されません）

注意

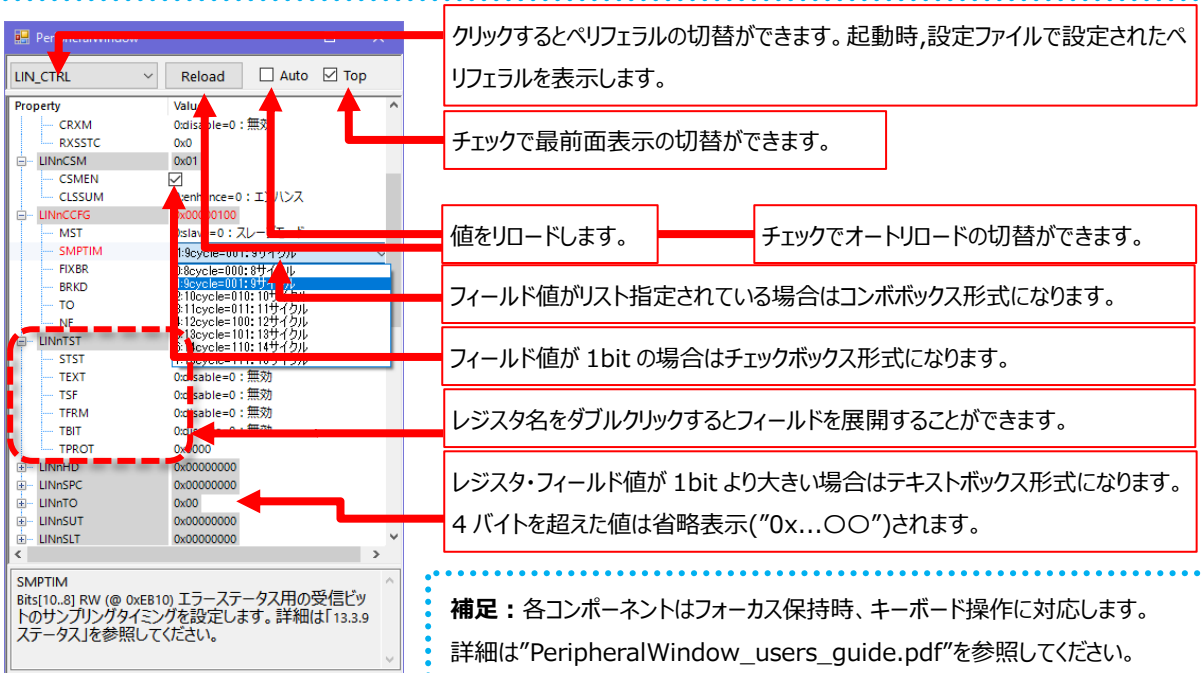
ヘッダー行(Address 00 01 .. 0F と表示されている箇所)はクリックしないでください。メモリの表示が崩れる場合があります。

© 2024 ROHM Co., Ltd.

7.12. ペリフェラルウィンドウ（Peripheral Window）

ペリフェラルウィンドウはペリフェラルレジスタに関する情報を表示します。プロパティ値は 3 パターンの読み書き方法があります。
画面表示は設定ファイル(settings.ini)に設定された XML ファイルに基づき表示されます。

補足
設定ファイル：起動時に表示するペリフェラル名、XML ファイルパスが設定された.ini 形式のファイル
XML ファイル：RapidMaker により出力されたペリフェラル情報が記述された.xml 形式のファイル



補足：各コンポーネントはフォーカス保持時、キーボード操作に対応します。
詳細は”PeripheralWindow_users_guide.pdf”を参照してください。

Figure 57. Peripheral Window 画面

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

Figure 58. Peripheral Window の表示設定

Peripheral Window 表示設定

settings.json（コンパイラ・デバッガ設定）の
“matisse.C.debug.showPeripheralWindowOnStart”を”true”に設定している場合、デバッグ開始時にペリフェラルウィンドウを開きます。

「プロジェクト構成と設定項目」を参照。

.ini ファイルの内容

.ini ファイルは以下のフォーマットで編集できます。

.ini ファイル

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

Table 26. ini ファイルのフォーマットの説明

設定項目	説明
[Peripheral] SelectedPeripheral=	ペリフェラルウィンドウの起動時に表示されるペリフェラル名が記述されています。 何も記述されていない場合は 1 番目のペリフェラルが表示されます。ペリフェラルウィンドウの終了時に表示されていたペリフェラル名が記述されます。
[FilePath] XMLFilePath=	ペリフェラル情報が CMSIS-SVD 形式で記述された XML ファイルのパスを設定してください。 **注意** このパスが誤っている場合、ペリフェラルウィンドウは起動できません。

© 2024 ROHM Co., Ltd.

7.13. リセット時の関数呼び出し履歴表示

WDT やハードウェアの不具合などでデバッグ中に CPU がリセットされた場合に、どの関数の実行中にリセットされたかを確認できる機能があります。
settings.json（コンパイラ・デバッガ設定）の「matisse.C.debug.showBacktraceOnReset」を「true」に設定してビルド実行すると、リセット時の関数呼び出し履歴表示機能が有効になります。
呼び出し履歴は DEBUG CONSOLE に表示されます。呼び出し履歴表示の後にデバッグは自動的に終了します。

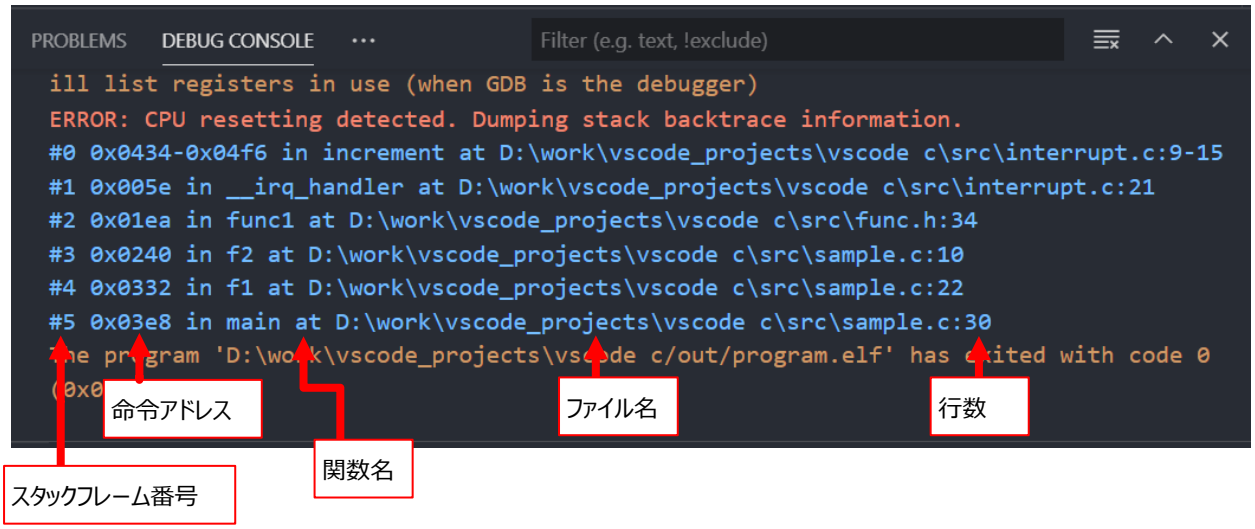


Figure 59. リセット時の関数呼び出し履歴表示

Table 27. 呼び出し履歴のフォーマットの説明

フィールド	説明
スタックフレーム番号	0 が CPU リセット時に実行中だった関数です。その後に呼び出し元の関数が続きます。
命令アドレス	実行していた命令アドレスです。 ⚠ **注意** リセット時にプログラムカウントがクリアされるため、スタックフレーム番号が 0 の関数は命令アドレスが確定しません。 そのためアドレスの範囲が表示されます。
関数名	実行していた関数名です。
ファイル名	実行していたファイル名です。
行数	実行していた箇所の行数です。 ⚠ **注意** リセット時にプログラムカウントがクリアされるため、スタックフレーム番号が 0 の関数は行数が確定しません。 そのため行数の範囲が表示されます。

- 本機能を有効にすると関数呼び出し履歴表示用の処理がプログラムに自動的に追加されます。ROM サイズが増加し、実行速度も低下します。
- 本機能はスタックに残された情報を元に関数呼び出し履歴を復元しています。スタック情報が破壊されていると正常に動作しません。
- 本機能でリセットを検出した後は RAM(スタックデータ)をクリアしてください。
  RAM をクリアせずに 2 回連続でリセットを検出した場合は、関数呼び出し履歴が正常に表示されない場合があります。
- インライン関数は呼び出し履歴に表示されない場合があります。
  コンパイラの最適化が有効になっていると、inline キーワードを付けていない関数でもインライン化の対象になります。

© 2024 ROHM Co., Ltd.

8. コマンド実行

8.1. コマンドパレットの表示とコマンド入力

キーボードから「Ctrl + Shift + P」もしくは F1 キーを入力すると、画面上部にコマンド入力欄(コマンドパレット)が表示されます。ここにコマンドを入力する事で VS Code の様々な機能が実行できます。

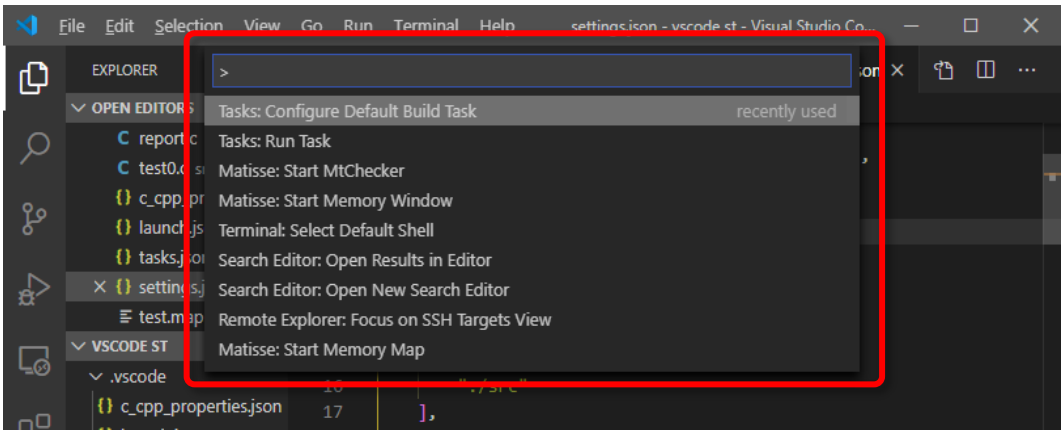


Figure 60. コマンドパレット表示とコマンド入力

■matiseye™-studio は次のようなコマンドを VS Code に追加します。

Table 28. コマンドパレットの説明

コマンド	説明
Matisse Debug: Start Memory Window	メモリウィンドウを起動します。 「メモリウィンドウ（Memory Window）」を参照。 **⚠注意** このコマンドはデバッグ中のみ使用できます。
Matisse Debug: Start Peripheral Window	ペリフェラルウィンドウを起動します。 「ペリフェラルウィンドウ（Peripheral Window）」を参照。 **⚠注意** このコマンドはデバッグ中のみ使用できます。
Matisse Debug: Start MtChecker	MtChecker(開発環境設定確認ツール)を起動します。
Tasks: Run Task	tasks.json で定義している以下のタスクを起動できます。 • Build: ビルド実行(Ctrl + Shift + B と同じ) • Clean: ビルドタスクで生成されたファイルを全て削除 • Rebuild: Clean タスク実行後に Build タスク実行
Matisse Analysis: Start Stack Analysis	Stack View を起動します。 「スタック静的解析（Stack View）」を参照。
Matisse Debug: Start Performance View	Performance View を起動します。デバッグ中のみ使用できます。 「PERFORMANCE」を参照。

© 2024 ROHM Co., Ltd.

9. よくある質問



matisse/interrupt.h で定義されている関数 ei()や di()を使うとエラーの波線が表示されます。
消す方法はありませんか。



回答
設定ファイル(settings.json)の"matisse.C.build.compilerPath"に mtcc のフルパス(デフォルト設定だと C:/Program Files/ROHM/Matisse/C/bin/mtcc.exe)を設定してください。



ライブラリファイルを置くライブラリパスの設定方法がわかりません。



回答
例えば AAA というフォルダをライブラリパスに追加したい場合は、設定ファイル(settings.json)の"matisse.C.build.libraryPath"に"AAA"という設定を追加してください。



ライブラリファイルをリンクする設定方法がわかりません。



回答
例えば libAAA.a というライブラリファイルをリンクしたい場合は、libAAA.a をライブラリパスが設定されているフォルダ内に置き、設定ファイル(settings.json)の"matisse.C.build.libraryFiles"に"libAAA.a"という設定を追加してください。

注意

mtcc では静的リンクのみサポートしています。動的リンク用のライブラリファイル(*.so)はリンクできません。



マクロ定義を設定ファイル(setitngs.json)の"matisse.C.build.preprocessorDefinitions"に書いてもソースコードに反映されません。



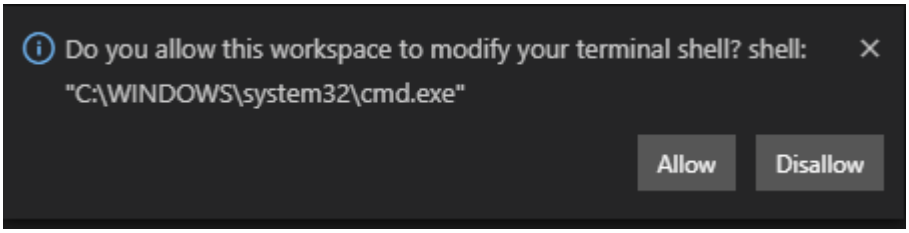
回答
設定ファイル(settings.json)を編集した後にビルド(Ctrl + Shift + b)を実行してください。現状の開発環境ではビルドをしないと設定ファイル(settings.json)の"matisse.C.build.preprocessorDefinitions"がソースコードに反映されません。また、AAA というプリプロセッサマクロを定義したい場合は"matisse.C.build.preprocessorDefinitions"に"AAA"という設定を追加してください。

注意

関数マクロを定義する場合は設定ファイル(settings.json)ではなくヘッダファイルに記述してください。

© 2024 ROHM Co., Ltd.

ビルド時に画面右下に画像のようなポップアップが表示されます。どうすればいいですか？



回答

Allow ボタンをクリックしてください。間違えて Disallow をクリックしてしまった場合は、
以下の手順を実行してください。
F1 キーを押して"Select Default Shell"と入力する。
エンターキーを押す。
表示される選択肢から"Command Prompt"を選択してエンターキーを押す。

使用していない関数を削除する方法はありますか？

回答

設定ファイル(settings.json)の”matisse.C.build.additionalCompileOptions”に”-ffunction-sections”を追加し、”matisse.C.build.additionalLinkerOptions”に”--gc-sections”を追加してください。
こうすることで使用していない関数がビルドの出力ファイルに含まれなくなります。

⚠ 注意

この設定を行うとデバッグができなくなります。デバッグが必要な時は実施せず、リリース用の出力ファイルを作成する時のみ実施してください。

© 2024 ROHM Co., Ltd.

10. ショートカットキー 一覧

10.1. 基本機能

Table 29. ショートカットキー(基本機能)

ショートカットキー	機能（アクション）
Ctrl + P	クイックオープンの表示
Ctrl + Shift + N	新規ウィンドウを開く
Ctrl + W	タブを閉じる
Ctrl + Shift + W	ウィンドウを閉じる

© 2024 ROHM Co., Ltd.

10.2. ファイル操作

Table 30. ショートカットキー(ファイル操作)

ショートカットキー	機能（アクション）
Ctrl + N	新規ファイルを開く
Ctrl + O	ファイルを開く
Ctrl + S	ファイルを保存
Ctrl + Shift + S	名前を付けてファイルを保存
Ctrl + K ⇨ S	開いている全てのファイルを保存
Ctrl + K ⇨ P	ファイルの絶対パスをコピー
Ctrl + K ⇨ R	ファイルをエクスプローラで表示
Ctrl + K ⇨ O	ファイルを新規ウィンドウで開く

© 2024 ROHM Co., Ltd.

10.3. エディタ操作

Table 31. ショートカットキー(エディタ操作)

ショートカットキー	機能（アクション）
Ctrl + \	エディタの分割
Ctrl + F4	エディタを閉じる
Ctrl + Shift + T	閉じたエディタを再度開く
Ctrl + PgUp	左のタブに移動
Ctrl + PgDn	右のタブに移動
Ctrl + 1 / 2 / 3	エディタグループの移動
Ctrl + Tab	エディタグループ内の次のエディタに移動
Ctrl + Shift + Tab	エディタグループ内の前のエディタに移動

© 2024 ROHM Co., Ltd.

10.4. 編集

Table 32. ショートカットキー(編集)

ショートカットキー	機能（アクション）
Ctrl + C	カーソルがある行をコピー
Ctrl + X	カーソルがある行を切り取り
Ctrl + Shift + \	対応する括弧に移動
Ctrl +] OR Tab	行のインデント
Ctrl + [OR Shift + Tab	行のアウトデント
Ctrl + Home	ファイルの先頭に移動
Ctrl + End	ファイルの末尾に移動
Ctrl + /	行コメントの追加/削除
Ctrl + Shift + [	選択領域を折りたたむ
Ctrl + Shift +]	選択領域を展開する
Ctrl + Alt + マウスドラッグ	矩形選択

© 2024 ROHM Co., Ltd.

10.5. 検索と置換

Table 33. ショートカットキー(検索と置換)

ショートカットキー	機能（アクション）
Ctrl + F	検索
Ctrl + H	置換
F3	次の検索結果に移動
Ctrl + F3	前の検索結果に移動

© 2024 ROHM Co., Ltd.

10.6. プログラミング

Table 34.ショートカットキー(プログラミング)

ショートカットキー	機能（アクション）
Ctrl + Space	補完候補の表示
Ctrl + Ait + F	ファイル全体を整形
Ctrl + K ⇨ Ctrl + F	選択箇所を整形
F12	定義へ移動
Alt + ←	前に戻る
Alt + →	次に進む
Alt + F12	定義を表示
Ctrl + K ⇨ F12	エディタを分割して定義を開く
Shift + F12	参照を表示
Shift + Ait + F12	プロジェクトフォルダ内の全ての参照を表示
F2	シンボルの名前を変更
Ctrl + G	指定行に移動
Ctrl + P	クイックオープン、ファイル移動

© 2024 ROHM Co., Ltd.

10.7. 画面表示

Table 35.ショートカットキー(画面表示)

ショートカットキー	機能（アクション）
F11	フルスクリーン/フルスクリーン解除
Ctrl + +	ズームイン
Ctrl + -	ズームアウト
Ctrl + B	サイドバー表示/サイドバー非表示
Ctrl + Shift + E	エクスプローラバーの表示
Ctrl + Shift + F	検索ビューの表示
Ctrl + Shift + D	デバッグビューの表示

© 2024 ROHM Co., Ltd.

10.8. デバッグ

Table 36.ショートカットキー(デバッグ)

ショートカットキー	機能（アクション）
F5	デバッグ開始
Shift + F5	デバッグの停止
Ctrl + Shift + F5	デバッグの再起動
F11	ステップイン
Shift + F11	ステップアウト
F10	ステップオーバー
F9	ブレークポイントの作成/削除
F6	デバッグの一時停止

© 2024 ROHM Co., Ltd.

11. オープンソースソフトウェアについて

本ソフトウェアは、ロームが権利を保有もしくは権利許諾を受けているソフトウェアに加えて、下記のライセンス条件の下で提供されているオープンソースソフトウェア（以下「オープンソースソフトウェアプログラム」といいます）を含んでいます。
オープンソフトウェアプログラムに関しては、それぞれの使用許諾条件が適用されますので、万一、オープンソースソフトウェアプログラムのライセンス条件と本資料の間で矛盾が生じた場合は、オープンソースソフトウェアプログラムのライセンス条件が優先されます。

含まれているソフトウェアとそのライセンス条件
・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. 商標について

「Windows」「VS Code」は、マイクロソフト グループの企業の商標です。
「Intel」は、Intel Corporation またはその子会社の商標です。
「Core™」は、Intel Corporation またはその子会社の商標または登録商標です。
「tinyMicon MatisseCORE™」「matiseye™」は、ローム株式会社の商標または登録商標です。

© 2024 ROHM Co., Ltd.

ご 注 意

1) 本資料に記載されている内容は、ロームグループ(以下「ローム」という)製品のご紹介を目的としています。ローム製品のご使用にあたりましては、別途最新のデータシートもしくは仕様書を必ずご確認ください。

2) ローム製品は、一般的な電子機器(AV機器、OA機器、通信機器、家電製品、アミューズメント機器等)もしくはデータシートに明示した用途への使用を意図して設計・製造されています。したがいまして、極めて高度な信頼性が要求され、その故障や誤動作が人の生命、身体への危険もしくは損害、またはその他の重大な損害の発生に関わるような機器または装置(医療機器、輸送機器、交通機器、航空宇宙機器、原子力制御装置、燃料制御、カーアクセサリーを含む車載機器、各種安全装置等)(以下「特定用途」という)にローム製品のご使用を検討される際は事前にローム営業窓口までご相談くださいますようお願いいたします。ロームの文書による事前の承諾を得ることなく、特 定用途にローム製品を使用したことによりお客様または第三者に生じた損害等に関し、ロームは一切その責任を負いません。

3) 半導体を含む電子部品は、一定の確率で誤動作や故障が生じる場合があります。万が一、誤動作や故障が生じた場合であっても、人の生命、身体、財産への危険または損害が生じないように、お客様の責任においてフェールセーフ設計など安全対策をお願いいたします。

4) 本資料に記載された応用回路例やその定数などの情報は、ローム製品の標準的な動作や使い方を説明するためのもので、実際に使用する機器での動作を明示的にも黙示的にも保証するものではありません。したがいまして、お客様の機器の設計において、回路やその定数及びこれらに関連する情報を使用する場合には、外部諸条件を考慮し、お客様の判断と責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関 し、ロームは一切その責任を負いません。

5) ローム製品及び本資料に記載の技術を輸出または国外へ提供する際には、「外国為替及び外国貿易法」、「米国輸出管理規則」など適用される輸出関連法令を遵守し、それらの定めにしたがって必要な手続きを行ってください。

6) 本資料に記載された応用回路例などの技術情報及び諸データは、あくまでも一例を示すものであり、これらに関する第三者の知的財産権及びその他の権利について権利侵害がないことを保証するものではありません。また、ロームは、本資料に記載された情報について、ロームもしくは第三者が所有または管理している知的財産権その他の権利の実施、使用または利用を、明示的にも黙示的にも、お客様に許諾するものではありません。

7) 本資料の全部または一部をロームの文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。

8) 本資料に記載の内容は、本資料発行時点のものであり、予告なく変更することがあります。ローム製品のご購入及びご使用に際しては、事前にローム営業窓口で最新の情報をご確認ください。

9) ロームは本資料に記載されている情報に誤りがないことを保証するものではありません。万が一、本資料に記載された情報の誤りによりお客様または第三者に損害が生じた場合においても、ロームは一切その責任を負いません。

ローム製品のご検討ありがとうございます。
より詳しい資料やカタログなどご用意しておりますので、お問い合わせください。

ROHM Customer Support System

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

© 2023 ROHM Co., Ltd. All rights reserved.