# IAR Embedded Workbench®

## C-SPY® デバッガガイド

Advanced RISC Machines Ltd

**ARM® コア**

UCSARM-11-J

IAR SYSTEMS

**改版情報**
第 11 版 : 2015 年 2 月

部品番号 : UCSARM-11-J

本ガイドは、ARM 用 IAR Embedded Workbench® のバージョン 7.40.x に適用する。

内部参照 : M18、Hom7.2、IMAE。

# 目次（章）

# 目次

# 表

# はじめに

*C-SPY® デバッガガイド*をご利用いただきありがとうございます。本書の目的は、IAR C-SPY® デバッガの機能を十分に理解して、ARM コアに基づいたアプリケーションのデバッグに役立てることです。

## 本ガイドの対象者

本ガイドは、IAR Embedded Workbench を使用してアプリケーションを開発し、C-SPY で利用可能なすべての機能を活用する場合に利用してください。

### 必要な知識

IAR Embedded Workbench のツールを使用するには、以下の実践的な知識が必要です。

- ARM コアのアーキテクチャ、命令セット（チップメーカのドキュメントを参照）
- C/C++ プログラミング言語
- 組込みシステム用アプリケーションの開発
- ホストコンピュータのオペレーティングシステム

IDE に統合されている他の開発ツールの詳細は、それぞれのドキュメントをご覧ください（29 ページの*その他のドキュメント*を参照）。

## 本ガイドの使用方法

IAR Embedded Workbench を初めて使用する場合は、まずガイド『IAR Embedded Workbench の使用開始の手順』で、IDE で利用可能なツールおよび機能の概要を確認することをお勧めします。

IAR Embedded Workbench の使用経験者で、IAR システムズの開発ツールの使用方法を再確認する必要がある場合は、IAR インフォメーションセンタのチュートリアルから始めることをお勧めします。プロジェクト管理、ビルド、編集の手順については*『ARM 用 IDE プロジェクト管理およびビルドガイド』*で説明しています。C-SPY を使用したデバッグの方法については、本書で説明しています。

本書では多くの*トピック*をカバーしており、各トピックのセクションにはそのトピックに関する概念を説明した概要が含まれます。これに目を通せば、C-SPY の機能が良く理解できます。さらに、トピックのセクションではス

テップごとの手順が説明されており、機能を使用する上で役に立ちます。最後に、各トピックのセクションには適切なリファレンス情報がすべて記載されています。

また、IAR システムズのユーザガイドおよびリファレンスガイドで不明な用語がある場合は、*『ARM 用 IDE プロジェクト管理およびビルドガイド』* の用語集も推奨します。

# このガイドの概要

本ガイドの構成および各章の概要を以下に示します。

**注 :** 本書のスクリーンショットの一部は、同種の製品からのものであり、ARM 用 IAR Embedded Workbench のものではありません。

### パート 1. 基本デバッグ

- 「*IAR C-SPY デバッガ*」は、C-SPY デバッガおよび一般的なデバッグと C-SPY に関する概念について説明します。また、さまざまな C-SPY ドライバについても解説します。本章では、さまざまな C-SPY ドライバ機能の違いについて簡単に説明します。
- 「*C-SPY を使用するにあたって*」は、C-SPY の使用を開始するための準備（設定や起動、ターゲットハードウェアに合わせた C-SPY の調整など）について説明します。
- 「*アプリケーションの実行*」では、ソースモードと逆アセンブリモードのデバッグの違い、アプリケーションの実行機能、ターミナルの I/O の操作方法について説明します。
- 「*変数と式*」では、C-SPY で使用する式や変数の構文、および変数の制限について説明します。また、変数や式のモニタ方法についても説明します。
- 「*ブレークポイント*」では、ブレークポイントシステムとブレークポイントの設定方法について説明します。
- 「*メモリとレジスタ*」では、メモリやレジスタの内容を調べる方法について説明します。

### パート 2. アプリケーションの解析

- 「*トレース*」は、トレースデータを使用して特定の状態までプログラムのフローを点検する方法について説明します。
- 「*プロファイラの使用*」では、実行中に最も多くの時間を費やす、プロファイラを使用したアプリケーションソースコードでの関数の検索方法について説明します。

- 「*コードカバレッジ*」では、コードカバレッジ機能を使用して、コードのすべての部分が実行されたか検証し、それによって実行されていない部分を特定する方法について説明します。
- 「*Power デバッグ*」では、Power デバッグのテクニックと、C-SPY を使用して予想外の電力消費につながるソースコード構造体を探す方法について説明します。
- 「*C-RUN ランタイムエラー解析*」では、ランタイムエラーチェックに C-RUN を使用する方法について説明します。

### パート 3. 高度なデバッグ

- 「*マルチコアデバッグ*」では、複数のコアを持つターゲットのデバッグ方法を説明します。
- 「*割込み*」では、C-SPY の割込みシミュレーションシステムの詳細と、ターゲットハードウェアの割込みに応じて割込みシミュレーションを設定する方法について説明します。
- 「*C-SPY マクロの使用*」では、C-SPY のマクロシステムとその機能、マクロの用途、マクロの使用方法について説明します。
- 「*C-SPY コマンドラインユーティリティ — cspybat*」では、バッチモードでの C-SPY の使用方法について説明します。
- 「*フラッシュローダ*」では、フラッシュローダの機能と利用方法について説明します。

### パート 4. 追加リファレンス情報

- 「*[デバッガ] オプション*」では、C-SPY デバッガを起動する前に設定しなければならないオプションについて説明します。
- 「*C-SPY ドライバについての追加情報*」では、他のトピックに記載されていない、C-SPY ドライバのメニューおよび機能について説明します。

## その他のドキュメント

ユーザドキュメンテーションは、ハイパーテキスト PDF 形式、およびコンテキスト依存のオンラインヘルプシステム（HTML フォーマット）があります。ドキュメンテーションには、インフォメーションセンタあるいは IAR Embedded Workbench IDE の［**ヘルプ**］メニューからアクセスできます。オンラインヘルプシステムは、F1 キーを押しても使用できます。

## ユーザガイドおよびリファレンスガイド

IAR システムズの各開発ツールについては、一連のガイドで説明しています。知りたい情報に対応するドキュメントを以下に示します。

- IAR システムズの製品のインストールおよび登録の要件と詳細は、同梱されているクイックレファンレンスのブックレットおよび *『インストールおよびライセンス ガイド』* にあります。
- IAR Embedded Workbench および同梱のツールを使用するにあたっては、*『IAR Embedded Workbench の使用開始の手順』* を参照してください。
- プロジェクト管理とビルドでの IDE の使用については、*『ARM 用 IDE プロジェクト管理およびビルドガイド』* を参照してください。
- IAR C-SPY デバッガの使用については、*『ARM 用 C-SPY® デバッガガイド』* を参照してください。
- ARM 用 IAR C/C++ コンパイラのプログラミングおよび IAR ILINK リンカを使用したリンクについては、*『ARM 用 IAR C/C++ 開発ガイド』* を参照してください。
- ARM 用 IAR アセンブラを使用したプログラミングについては、*『ARM 用 IAR アセンブラリファレンスガイド』* を参照してください。
- IAR DLIB ライブラリの使用については、オンラインヘルプで利用できる *DLIB ライブラリリファレンス情報* を参照してください。
- C-STAT と必要なチェックを使用した静的解析の実行については、*C-STAT® Static Analysis Guide* を参照してください。
- MISRA-C ガイドラインを使用して、安全性を最重要視したアプリケーションを開発する方法については、*『IAR Embedded Workbench® MISRA-:2004 Reference Guide』* または *『IAR Embedded Workbench® MISRA-C:1998 Reference Guide』* を参照してください。
- I-jet の使用法については、*『I-jet®、I-jet Trace、I-scope 用 IAR デバッグプローブガイド』* を参照してください。
- JTAGjet-Trace の使用については、*『ARM 用 JTAGjet-Trace ユーザガイド』* を参照してください。
- IAR J-Link と IAR J-Trace については、*『ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド』* を参照してください。
- ARM 用 IAR Embedded Workbench の旧バージョンで開発したアプリケーションコードやプロジェクトの移植については、*『IAR Embedded Workbench® 移行ガイド』* を参照してください。

**注 :** 製品のインストール内容によっては、他のドキュメントも提供される場合があります。

### オンラインヘルプシステムを参照

コンテキスト依存のオンラインヘルプの内容は以下のとおりです。

- IDE でのプロジェクト管理と編集、ビルドに関する情報
- IAR C-SPY® デバッガを使用したデバッグについての情報
- IDE のメニューやウィンドウ、ダイアログボックスに関するリファレンス情報
- コンパイラのリファレンス情報
- DLIB ライブラリ関数のキーワードリファレンス情報 関数のリファレンス情報を確認するには、エディタウィンドウで関数名を選択し、F1 キーを押します。

### WEB サイト

推奨 Web サイト :

- Advanced RISC Machines Ltd の Web サイト (**www.arm.com**) には、ARM コアに関する情報とニュースが記載されています。
- IAR システムズの Web サイト (**www.iar.com/jp**) では、アプリケーションノートおよびその他の製品情報を公開しています。
- C 標準化作業グループの Web サイト、**www.open-std.org/jtc1/sc22/wg14**。
- C++ Standards Committee の Web サイト、**www.open-std.org/jtc1/sc22/wg21**。
- Embedded C++ Technical Committee の Web サイト (**www.caravan.net/ec2plus**) には、Embedded C++ 規格についての情報が公開されています。

## 表記規則

IAR システムズのドキュメントでプログラミング言語 C と記述されている場合、特に記述がない限り C++ も含まれます。

製品インストール内のディレクトリについて言及する場合（`arm¥doc` など）、その場所のフルパスを前提とします。この場合、`c:¥Program Files¥IAR Systems¥Embedded Workbench 7.`*n*`¥arm¥doc` を意味します。

## 表記規則

IAR システムズのドキュメント セットでは、次の表記規則を使用します :

| スタイル | 用途 |
|---|---|
| `computer` | • ソースコードの例、ファイルパス。<br>• コマンドライン上のテキスト。<br>• 2 進数、16 進数、8 進数。 |
| *parameter* | パラメータとして使用される実際の値を表すプレースホルダ。たとえば、*filename*.h の場合、*filename* はファイルの名前を表します。 |
| **[**option**]** | ディレクティブのオプション部分、[と] は実際のディレクティブの一部ではなく、すべての [、]、{、} はディレクティブの構文の一部です。 |
| **{**option**}** | ディレクティブの必須部分、{ と } は実際のディレクティブの一部ではなく、すべての [、]、{、} はディレクティブ構文の一部です。 |
| [option] | コマンドのオプション部分。 |
| [a\|b\|c] | 代替の選択肢を持つコマンドのオプション部分。 |
| {a\|b\|c} | コマンドの必須部分に選択肢があることを示します。 |
| **太字** | 画面で表示されるメニュー、メニューコマンド、ボタン、ダイアログボックス の名前を示します。 |
| *斜体* | • 本ガイドや他のガイドへのクロスリファレンスを示します。<br>• 強調。 |
| … | 3 点リーダは、その前の項目を任意の回数繰り返せることを示します。 |
|  | IAR Embedded Workbench® IDE 固有の内容を示します。 |
|  | コマンドライン インタフェース固有の内容を示します。 |
|  | 開発やプログラミングについてのヒントを示します。 |
|  | ワーニングを示します。 |

*表 1: 本ガイドで使用されている表記規則*

## 命名規約

以下の命名規約は、このドキュメントに記述されている IAR システムズの製品およびツールで使用されています。

| ブランド名 | 一般名称 |
|---|---|
| ARM 用 IAR Embedded Workbench® | IAR Embedded Workbench® |
| ARM 用 IAR Embedded Workbench® IDE | IDE |
| ARM 用 IAR C-SPY® デバッガ | C-SPY、デバッガ |
| IAR C-SPY® シミュレータ | シミュレータ |
| ARM 用 IAR C/C++ コンパイラ | コンパイラ |
| ARM 用 IAR アセンブラ | アセンブラ |
| IAR ILINK リンカ | ILINK、リンカ |
| IAR DLIB ライブラリ | DLIB ライブラリ |

*表 2: このガイドで使用されている命名規約*

# パート I. 基本デバッグ

*ARM 用 C-SPY® デバッガガイド*のこのパートは、以下の章で構成されています。

- IAR C-SPY デバッガ
- C-SPY を使用するにあたって
- アプリケーションの実行
- 変数と式
- ブレークポイント
- メモリとレジスタ

# IAR C-SPY デバッガ

- C-SPY の概要
- デバッガの概念
- C-SPY の概要
- IAR C-SPY シミュレータ
- C-SPY ハードウェアデバッガドライバ

## C-SPY の概要

以下のトピックについて説明します :

- 統合環境
- C-SPY デバッガの特長
- RTOS 認識

### 統合環境

C-SPY は、組込みアプリケーション用の高級言語デバッガです。IAR システムズのコンパイラおよびアセンブラとともに使用することを目的としており、IDE に完全に統合されているため、同一アプリケーション内での開発とデバッグが可能になります。このため、以下のような操作が可能です。

- デバッグ中の編集。デバッグセッション中に、デバッグの制御に使用するものと同じソースコードウィンドウで直接修正を行うことができます。変更内容は、次にプロジェクトをリビルドするときに有効になります。
- 開発サイクル中の任意の位置へのブレークポイントの設定。デバッガを実行していないときもブレークポイント定義を確認および修正することができ、ブレークポイント定義フローを編集中のテキストで確認および修正できます。ウォッチプロパティ、ウィンドウレイアウト、レジスタグループなどのデバッグ設定は、デバッグセッションが終わるまで保持されます。

Embedded Workbench のワークスペースで開いているウィンドウはすべて、C-SPY デバッガを起動してもそのまま開いています。それ以外に、C-SPY 固有の複数のウィンドウが開かれます。

## C-SPY デバッガの特長

IAR システムズはツールチェーン全体を提供しているため、コンパイラやリンカの出力にデバッガ用の詳細なデバッグ情報を含めることができ、デバッグ時に非常に役立ちます。

C-SPY は以下のような一般的特長があります。

- ソースおよび逆アセンブリレベルのデバッグ

  C-SPY では、C/C++ とアセンブラの両方のソースコードで、ソースと逆アセンブリのデバッグを必要に応じて切り替えることができます。

- 関数呼出しのステップ実行

  ソースレベルのステップ実行の最小単位が行単位である従来のデバッガとは異なり、C-SPY ではすべての文および関数呼出しがステップポイントとして認識されるため、より詳細な制御が可能です。つまり、式に含まれる呼出しと、他の関数への呼出しのパラメータとして記述されている呼出しの両方をステップ実行できます。パラメータ中の呼出しのステップ実行は、オブジェクトのコンストラクタなどのように、多数の追加関数呼出しが行われる C++ コードで特に便利です。

- コード / データブレークポイント

  C-SPY のブレークポイントシステムでは、デバッグ対象のアプリケーションでさまざまなブレークポイントを設定し、目的箇所で実行を停止することができます。たとえば、ブレークポイントを設定して、プログラムロジックが正しいかどうかや、データアクセスに設定してデータが変更されたタイミングと方法を調べたりできます。

- 変数および式のモニタ

  変数および式の場合、さまざまな機能から選択できます。指定した変数および式セットの値を、連続またはオンデマンドで簡単にモニタすることができます。また、ローカル変数や静的変数だけをモニタするよう選択も可能です。

- コンテナ認識

  C-SPY でアプリケーションを実行すると、STL のリストやベクタなどのライブラリデータ型のエレメントを表示することができます。これにより、C++ STL コンテナの使用時に内容を簡単に把握してデバッグすることができます。

- コールスタック情報

  コンパイラは、詳細なコールスタック情報を生成します。これにより、実行に影響することなく、プログラムカウンタがどこを指していても、関数呼出しの完全なコールスタックをデバッガで表示できます。コールスタックで任意の関数を選択し、その関数についてローカル変数やレジスタの情報を表示することができます。

- 強力なマクロシステム

  C-SPY は強力な内蔵マクロシステムを装備しており、複雑なアクションを定義して実行することができます。C-SPY マクロは、単独で、あるいはブレークポイントや割込みシミュレーションシステム（シミュレータ使用時）と組み合せて使用し、さまざまなタスクを実行できます。

## C-SPY デバッガのその他の特長

他にも、以下のような特長があります。

- スレッド実行により、ターゲットアプリケーションの実行中も IDE の応答性を維持
- 自動ステップ実行
- ソースブラウザで、関数、型、変数を簡単に表示可能
- さまざまな変数の型を認識
- 設定可能なレジスタ（CPU、周辺）、メモリウィンドウ
- オーバフローの検出機能を持った、グラフィック表示のスタックビュー
- コードカバレッジ、関数レベルのプロファイルをサポート
- ターゲットアプリケーションは、ファイル I/O を使用してホスト PC のファイルにアクセス可能
- オプションのターミナル I/O エミュレーション

## RTOS 認識

C-SPY は RTOS 認識デバッグをサポートしています。

以下のオペレーティングシステムが現在サポートされています。

- AVIX-RT
- CMX-RTX
- CMX-Tiny+
- eForce uC3/Compact
- eSysTech X realtime kernel
- Express Logic ThreadX
- FreeRTOS、OpenRTOS、SafeRTOS
- Freescale MQX
- Micrium uC/OS-II および Micrium uC/OS-III
- Micro Digital SMX
- MISPO NORTi
- OSEK (ORTI)

- RTXC Quadros
- Segger embOS
- unicoi Fusion

RTOS プラグインモジュールは、IAR システムズかサードパーティより提供されます。サポートされている RTOS モジュールについては、ソフトウェア販売代理店か IAR システムズの担当者までお問い合せください。また、IAR システムズの Web サイトでも情報を提供しています。

C-SPY RTOS 認識プラグインモジュールを使用すると、RTOS 上で構築されたアプリケーションを高度に制御し、モニタできます。モニタできるのは、タスクリスト、キュー、セマフォ、メールボックス、さまざまな RTOS システム変数などの RTOS 固有の項目です。タスク固有のブレークポイントとタスク固有のステップ実行により、タスクを簡単にデバッグできます。

デバッグセッションが開始されると、ロードされたプラグイン独自のメニューとウィンドウ、ボタンが追加されます（RTOS がアプリケーションとリンクされている場合）。他の RTOS 認識プラグインモジュールについては、メーカのプラグインモジュールを参照してください。RTOS 文書へのリンクについては、［**ヘルプ**］メニューから入手可能なリリースノートを参照してください。

# デバッガの概念

以下のトピックについて説明します：

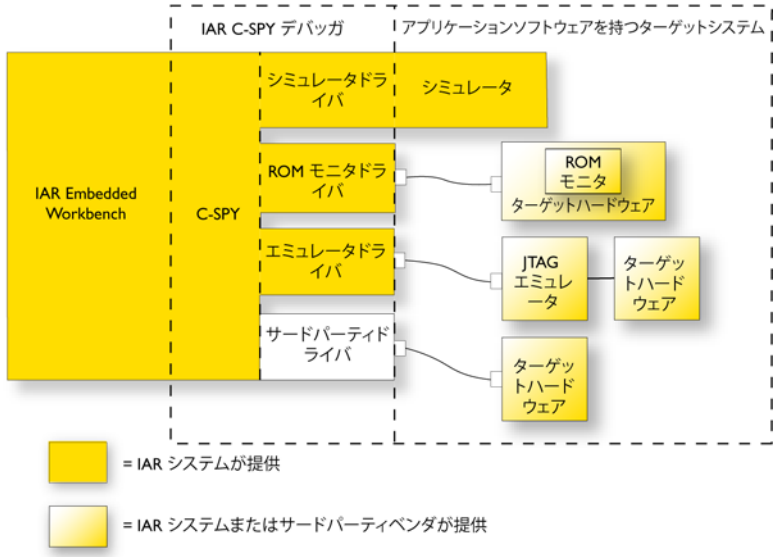
- C-SPY とターゲットシステム
- デバッガ
- ターゲットシステム
- アプリケーション
- C-SPY デバッガシステム
- ROM モニタプログラム
- サードパーティ製デバッガ
- C-SPY プラグインモジュール

このセクションでは、デバッグの一般的な概念と用語、特に C-SPY に関連する内容について説明します。このセクションは、C-SPY の機能に関する具体的な情報を含んでいません。代わりに、これらの情報はこのドキュメントの他の章にあります。IAR システムズのユーザドキュメントでは、デバッガの概念を説明するときに、このセクションで説明されている用語を使用します。

## C-SPY とターゲットシステム

C-SPY は、ソフトウェアターゲットシステムとハードウェアターゲットシステムの両方のデバッグに使用できます。

以下の図は、C-SPY およびターゲットシステムの概要を示します。




## デバッガ

C-SPY のようなデバッガは、ターゲットシステム上でアプリケーションをデバッグするためのプログラムです。

## ターゲットシステム

ターゲットシステムは、アプリケーションをデバッグするときに、それを実行するシステムです。ターゲットシステムは、評価ボードか独自に設計したハードウェアから構成されます。また、その全体または一部をソフトウェアでシミュレーションすることもできます。ターゲットシステムは、その種類ごとに専用の C-SPY ドライバを必要とします。

## アプリケーション

ユーザアプリケーションは、ユーザが開発し、C-SPY を使用してデバッグを行うソフトウェアです。

## C-SPY デバッガシステム

C-SPY は、デバッガの基本機能セットを提供する部分とターゲット固有のバックエンドで構成されています。バックエンドは、各マイクロコントローラに 1 つずつあってマイクロコントローラのプロパティを定義するプロセッサモジュールと、*C-SPY ドライバ*の 1 つから構成されています。C-SPY ドライバは、ターゲットシステムとの通信およびターゲットシステムの管理を提供します。また、特殊ブレークポイントなど、ターゲットシステムが提供する機能へのユーザインタフェースとして、メニュー、ウィンドウ、ダイアログボックスを提供します。C-SPY ドライバには通常、大きく分けて以下の 3 つの種類があります。

- シミュレータドライバ
- ROM モニタドライバ
- エミュレータドライバ

C-SPY にはシミュレータドライバのほか、製品パッケージによっては、ハードウェアデバッガシステム用のオプションのドライバもインストールされます。C-SPY ドライバおよび各ドライバの機能については、43 ページの *C-SPY の概要*を参照してください。

## ROM モニタプログラム

ROM モニタプログラムは、ターゲットハードウェアの不揮発性メモリにロードされるファームウェアで、アプリケーションと並行して動作します。ROM モニタは、デバッガと通信して、ステップ実行やブレークポイントなど、アプリケーションのデバッグに必要なサービスを提供します。

## サードパーティ製デバッガ

サードパーティ製デバッガも IAR システムズのツールチェーンに組み込むことができます。その場合、サードパーティ製デバッガは、ELF/DWARF、Intel-extended、または Motorola の読み込みが可能であることが必要です。サードパーティ製デバッガで使用するフォーマットについては、そのデバッガに付属のユーザドキュメントを参照してください。

## C-SPY プラグインモジュール

C-SPY は、プラグインモジュールとしてデバッガの追加機能を実装するときに使用できる、オープン SDK で構築したモジュール化構造アーキテクチャとして設計されています。これらのモジュールは、IDE にシームレスに統合できます。

IAR システムズかサードパーティ製のプラグインモジュールを使用できます。このようなモジュールの例を以下に示します。

- IDE に統合されているコードカバレッジ。
- 特定のデバッグシステムを使用するための様々な C-SPY ドライバ。
- リアルタイム OS 対応のデバッグをサポートする RTOS プラグインモジュール。
- C-SPY が周辺ユニットをシミュレートするための周辺デバイスシミュレーションモジュール。このようなプラグインモジュールは、IAR システムズでは提供されていませんが、サードパーティサプライヤが開発および提供できます。
- IAR visualSTATE と IAR Embedded Workbench間のインタフェースとして機能する C-SPYLink。これは、標準 C レベルシンボリックデバッグのほか、真の意味での高水準ステートマシンデバッグを C-SPY で直接可能にします。詳細については、IAR visualSTATE の付属ドキュメントを参照してください。

C-SPY SDK の詳細については、IAR システムズまでお問い合わせください。

## C-SPY の概要

本書の執筆時点では、ARM コアの IAR C-SPY デバッガでは、以下に示すターゲットシステムのドライバと評価ボードを使用できます。

- シミュレータ
- I-jet / I-jet Trace / JTAGjet / JTAGjet-Trace、JTAGjet-Trace-CM のデバッグプローブ
- J-Link / J-Trace JTAG/SWD プローブ
- RDI (Remote Debug Interface)
- Macraigor JTAG プローブ
- GDB サーバ
- ST-LINK JTAG/SWD プローブ（ST Cortex-M デバイスの場合のみ）
- FTDI または ICDI を使用した TI Stellaris JTAG/SWD インタフェース（Stellaris Cortex デバイスの場合のみ）
- TI XDS JTAG インタフェース
- P&E Microcomputer Systems。このドライバの詳細は、`arm¥doc` ディレクトリのドキュメント、*「Configuring IAR Embedded Workbench for ARM to use a P&E Microcomputer Systems Interface」* を参照してください。
- Angel デバッグモニタ

- Analog Devices ADuC7xxx ボード用 IAR ROM モニタ、Philips LPC210x 用 IAR Kickstart Card

**注 :** IAR Embedded Workbench で提供するドライバに加えて、サードパーティベンダ製のデバッガドライバもロードできます（570 ページの *サードパーティ製ドライバのオプション*を参照してください）。

## C-SPY ドライバ間の差異

次の表は、Simulator、I-jet/JTAGjet、J-Link/J-Trace、ST-LINK、CMSIS-DAP 間の主な違いをまとめたものです。

| 機能 | シミュレータ | I-jet/JTAGjet | J-Link/J-Trace | ST-LINK | CMSIS-DAP |
|---|---|---|---|---|---|
| コードブレークポイント | x | x | x | x | x |
| データブレークポイント | x | x | x | x | x |
| 割込みロギング [1] | x | x | x | x | -- |
| データロギング [1] | x | x | x | x | -- |
| コールスタックトレース [1] | x | x | x | -- | x |
| イベントロギング [1] | -- | x | x | -- | -- |
| ライブウォッチ [1] | -- | x | x | x | x |
| サイクルカウンタ [1] | x | x | x | x | x |
| コードカバレッジ [1] | x | x | x | x | x |
| データカバレッジ | x | -- | -- | -- | -- |
| 関数 / 命令プロファイラ [1] | x | x | x | x | x |
| トレース [1] | x | x | x | x | x |
| マルチコアデバッグ [1] | x | x | -- | -- | x[2] |
| Power デバッグ | -- | x | x | -- | -- |

*表 3: ドライバの違い、I-jet/JTAGjet、J-Link/J-Trace および ST-LINK*

1 特定の要件や制限については、このガイドの該当する章を参照してください。

2 限定サポート。

次の表は、シミュレータと他のサポートされているハードウェアデバッガドライバの違いをまとめたものです。

| 機能 | シミュレータ | RDI | Mac-raigor | GDB サーバ | TI Stellaris | TI XDS | Angel |
|---|---|---|---|---|---|---|---|
| コードブレークポイント | x | x | x | x | x | x | x |
| データブレークポイント | x | x | x | x | x | -- | -- |
| 割込みログ | x | -- | -- | -- | -- | -- | -- |
| サイクルカウンタ | x | -- | -- | -- | -- | -- | -- |
| コードカバレッジ | x | -- | -- | -- | -- | -- | -- |
| データカバレッジ | x | -- | -- | -- | -- | -- | -- |
| 関数 / 命令プロファイラ | x | -- | -- | -- | -- | -- | -- |
| トレース[1] | x | x | -- | -- | -- | -- | -- |

*表 4: ドライバの違い、他のデバイス*

1 特定の要件や制限については、このガイドの該当する章を参照してください。

# IAR C-SPY シミュレータ

C-SPY シミュレータは、ターゲットプロセッサの機能をソフトウェアで完全にシミュレーションするため、ハードウェアがすべて揃っていなくてもプログラムロジックをデバッグできます。ハードウェアが不要であるため、多くのアプリケーションにとって最も費用効果の高いソリューションでもあります。

C-SPY シミュレータは以下をサポートしています。

- 命令レベルのシミュレーション
- メモリの構成、検証
- 割込みシミュレーション
- イミディエイトブレークポイントと C-SPY マクロシステムを使用した周辺シミュレーション

ハードウェアデバッグシステムを使用せずにハードウェアをシミュレーションすると、一部の制限が適用されず、他の制限が適用されることを意味します。次に例を示します。

- シミュレータでは無制限にブレークポイントを設定できます
- アプリケーションの実行を停止すると、シミュレータでは時間が実際に停止します。ハードウェアデバッグシステムではアプリケーションの実行を停止すると、システムでアクティビティが残っていることがあります。た

とえば、周辺ユニットがまだアクティブで、SFR ポートからの読取りや書込みが行われていることがあります

- ハードウェアデバッグシステムを使用する場合に比べて、シミュレータではアプリケーションの実行がずっと遅くなります。ただし、デバッグセッション中にはこのことが問題とならない可能性もあります
- シミュレータはサイクルアキュレートではありません
- 周辺シミュレーションは C-SPY シミュレータでは制限されるため、シミュレータは主に周辺ユニットとのやりとりが多くないコードのデバッグに適しています

# C-SPY ハードウェアデバッガドライバ

C-SPY は、C-SPY ハードウェアデバッガドライバをインタフェースとして使用することにより、ハードウェアデバッガに接続できます。

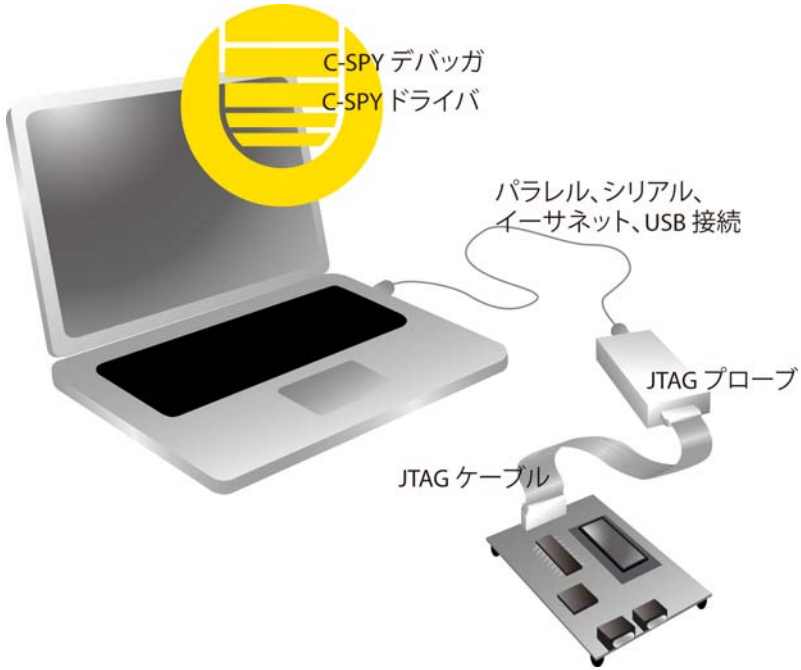
デバッグセッションが開始されると、アプリケーションは自動的にダウンロードされてターゲットメモリにプログラムされます。この機能は必要があれば無効にできます。

## 通信の概要

ターゲットシステムのタイプに応じて、2 つの主な通信設定があります。ほとんどの arm コアには内蔵のオンチップデバッグサポートがあります。ハードウェアデバッガのロジックがコアに組み込まれているため、デバッグが機能するためにデバッグプローブのほかに、通常の ROM モニタプログラムや特定の追加ハードウェアは必要ありません。オンチップのデバッグサポートが組み込まれていない一部のデバイスでは、代わりに使用可能な ROM モニタデバッガソリューションがあります。

### デバッグプローブまたはエミュレータを持つターゲットシステムの概要

ほとんどのターゲットシステムにはエミュレータやデバッグプローブ、デバッグアダプタがあって、ホストコンピュータと評価ボード間に接続されています。




USB 接続を使用する場合、USB ポート経由でプローブを使用する前に、特定の USB ドライバをインストールする必要があります。ドライバは IAR Embedded Workbench for ARM のインストールメディアにあります。

### ROM モニタを使用したターゲットシステムの概要

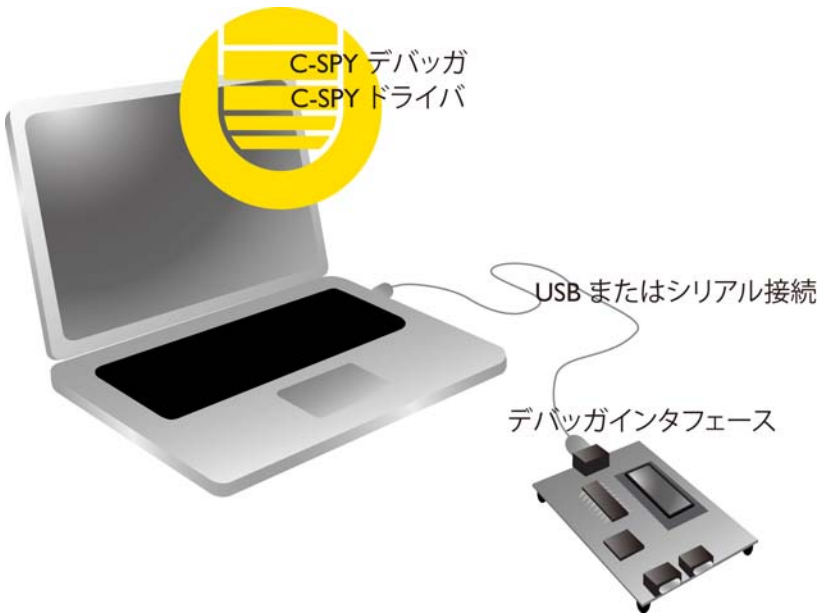
IAR Embedded Workbench には、既成の ROM モニタが 2 つ用意されています。

- IAR Angel デバッグモニタドライバを使用すると、Angel デバッグモニタプロトコルに準拠するすべてのデバイスと通信できます。ほとんどの場合、これらは評価ボードです。
- IAR ROM モニタドライバを使用すると、C-SPY では Analog Devices ADuC7xxx ボード、Philips LPC210x 用 IAR Kickstart Card に接続できます。ほとんどの ROM モニタでは、デバッグするコードを RAM に置く必要が

あります。これは、ブレークポイントの設定やアプリケーションコードのステップインを行う唯一の方法が、コードを RAM にダウンロードすることであるためです。ROM モニタによっては（Analog Devices ADuC7xxx の場合など）、デバッグするコードはフラッシュメモリにあってもかまいません。デバッグ機能を維持するために、ROM モニタではいくつかの命令をシミュレートする場合（シングルステップを行うときなど）があります。

このボードには、アプリケーションソフトウェアと同時に実行するファームウェア（ROM モニタ自体）が搭載されています。このファームウェアでは、シリアルポート接続経由で IAR C-SPY デバッガからコマンドを受け取り、アプリケーションの実行を制御します。

C-SPY ROM モニタドライバを使用すると、C-SPY はフラッシュメモリに ROM モニタを備えたターゲットシステムに接続できます。




このプロトコルは、必要なものはシリアルケーブルだけであるため、ターゲットのデバッグに費用のかからない方法です。デバッグするソースコードはすべて RAM に置く必要があります。ブレークポイントの設定やアプリケーションのステップインを行う唯一の方法が、コードを RAM にダウンロードすることです。

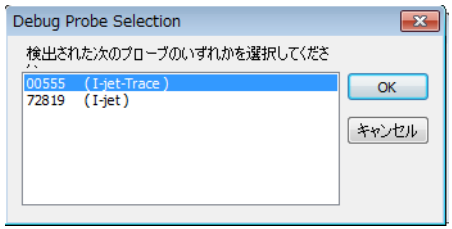
詳細については、次を参照してください。

- angel_quickstart.html ファイル（arm¥doc¥infocenter ディレクトリ内）、またはメーカのドキュメントを参照してください。
- iar_rom_quickstart.html ファイル（arm¥doc¥infocenter ディレクトリ内）、またはメーカのドキュメントを参照してください。

### ハードウェアインストール

ハードウェアインストールについて詳しくは、ターゲットシステムに付属のメーカのマニュアルを参照してください。ターゲットボードとエミュレータ、デバッグプローブ、C-SPY 間の適切な通信を確実に行うため、以下の起動シーケンスを推奨します。

1. プローブをターゲットボードに接続します。
2. USB ケーブルをデバッグプローブに接続します。
3. USB から電源が供給されていない場合は、デバッグプローブを起動します。
4. デバッグプローブから電源が供給されていない場合は、ターゲットボードを起動します。
5. C-SPY デバッグセッションを開始します。
6. 複数のデバッグプローブがコンピュータに接続されている場合、**[デバッグプローブ選択]** ダイアログボックスが表示されます。ダイアログボックスで、使用するプローブを選択して **[OK]** をクリックします。詳細については、484 ページの *--drv_communication* を参照してください。




### USB ドライバのインストール

USB ドライバも必要です。場合によっては、このドライバが自動的にインストールされることもありますが、一部のプローブについては手動でインストールする必要があります。

## I-jet および JTAGjet USB ドライバのインストール

USB ポート経由で I-jet や JTAGjet インタフェースを使用するには、まず適切な USB ドライバをインストールする必要があります。USB ケーブルを使用してコンピュータと I-jet、JTAGjet、または JTAGjet-Trace プローブを接続します。

### *Windows 7 およびそれ以降*

1 Windows のデバイスマネージャを起動します。

2 ［他のデバイス］を選択し、**[JTAGjet]** を右クリックして［**ドライバ ソフトウェアの更新**］を選択します。

3 ［**コンピュータを参照してドライバ ソフトウェアを検索します**］をクリックして、`arm¥drivers¥jet¥USB` に移動します。

4 ［**次へ**］に続いて［**インストール**］をクリックします。

### *Windows 7 以前*

I-jet または JTAGjet インタフェースとコンピュータが初めて接続されるとき、Windows でダイアログボックスが開いて、USB ドライバを探すよう求められます。ドライバは、`arm¥drivers¥jet¥USB` の製品インストールにあります。

初期設定が終了すると、再度ドライバをインストールする必要はありません。

## J-Link USB ドライバのインストール

USB ポート経由で J-Link JTAG プローブを使用するには、まず Segger J-Link USB ドライバをインストールする必要があります。

1 ARM 用 IAR Embedded Workbench をインストールします。

2 USB ケーブルを使用してコンピュータと J-Link を接続します。まだターゲットボードには J-Link を接続してはいけません。Windows が USB ドライバを検索している間、J-Link の前面パネルにある緑の LED が数秒間点滅します。

`InstDrivers.exe` アプリケーションを実行します。これは arm¥drivers¥JLink ディレクトリの製品インストールにあります。

初期設定が終了すると、再度ドライバをインストールする必要はありません。

USB ドライバが J-Link プローブとの接続を確立するまで、J-Link では点滅が継続することに注意してください。接続が確立されると、J-Link では接続されていることを示すために、連続して点灯し始めます。

### ST-LINK バージョン 2 の ST-LINK USB ドライバのインストール

USB ポート経由で ST-LINK バージョン 2 の JTAG プローブを使用するには、まず ST-LINK USB ドライバをインストールする必要があります。

1 ARM 用 IAR Embedded Workbench をインストールします。

2 USB ケーブルを使用してコンピュータと ST-LINK を接続します。まだターゲットボードには ST-LINK を接続してはいけません。

ST-LINK とコンピュータの接続は初めてであるため、Windows ではダイアログボックスを開き、USB ドライバがある場所を尋ねます。USB ドライバは、arm¥drivers¥ST-Link ディレクトリ内の製品インストールにあります。`ST-Link_V2_USBdriver.exe`.

初期設定が終了すると、再度ドライバをインストールする必要はありません。

### TI Stellaris USB ドライバのインストール

USB ポート経由で FTDI または ICDI を使用して TI Stellaris JTAG インタフェースを使用する前に、Stellaris USB ドライバをインストールする必要があります。

1 ARM 用 IAR Embedded Workbench をインストールします。

2 USB ケーブルを使用してコンピュータと TI ボードを接続します。

Stellaris JTAG インタフェースとコンピュータの接続は初めてであるため、Windows ではダイアログボックスを開き、USB ドライバがある場所を尋ねます。FTDI と ICDI には異なる USB ドライバがあります。ドライバは、`arm¥drivers¥StellarisFTDI` および `arm¥drivers¥StellarisICDI` のディレクトリにある製品インストールにあります。

初期設定が終了すると、再度ドライバをインストールする必要はありません。

### TI XDS USB ドライバのインストール

TI XDS JTAG インタフェースを USB ポート経由で使用する前に、TI XDS パッケージをインストールする必要があります。

1 ARM 用 IAR Embedded Workbench をインストールします。

2 TI XDS パッケージをインストールします。これは `arm¥drivers¥ti-xds` ディレクトリにあります。提示されたインストールディレクトリを選択することを推奨します。569 ページの *TI XDS の設定オプション*も参照してください。

3 USB ケーブルを使用してコンピュータと TI ボードを接続します。

### OpenOCD サーバの設定

詳細については、arm¥doc¥infocenter ディレクトリの gdbserv_quickstart.html ファイルを参照するか、メーカの資料を参照してください。

# C-SPY を使用するにあたって

- C-SPY の設定
- C-SPY の起動
- ターゲットハードウェアへの適合
- デバッガ起動の概要
- サンプルプロジェクトの実行
- C-SPY の起動についてのリファレンス情報

## C-SPY の設定

以下のタスクについて説明します :

- デバッグの設定
- リセットからの実行
- セットアップマクロファイルの使用
- デバイス記述ファイルの選択
- プラグインモジュールのロード

### デバッグの設定

1. C-SPY ドライバで必要な場合、USB ドライバまたはその他の通信ドライバをインストールしてください。

   詳細については、以下を参照してください。

   - 50 ページの *I-jet および JTAGjet USB ドライバのインストール*
   - 50 ページの *J-Link USB ドライバのインストール*
   - 51 ページの *ST-LINK バージョン 2 の ST-LINK USB ドライバのインストール*
   - 51 ページの *TI Stellaris USB ドライバのインストール*
   - 51 ページの *TI XDS USB ドライバのインストール*
   - 52 ページの *OpenOCD サーバの設定*

2 C-SPY を起動する前に、［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**設定**］を選択し、シミュレータやハードウェアデバッガシステムの中から使用するデバッガシステムに合った C-SPY ドライバを選んでください。

3 ［**カテゴリ**］リストで、適切な C-SPY ドライバを選択して設定を行います。

これらのオプションについては、529 ページの*［デバッガ］オプション*を参照してください。

4 **[OK]** をクリックします。

5 ［**ツール**］>［**オプション**］を選択して、［**IDE オプション**］ダイアログボックスを開きます。

- ［**デバッガ**］を選択して、デバッガの動作を設定します。
- ［**スタック**］を選択して、デバッガによるスタック使用量のトラッキングを設定します。

これらのオプションについては、*『ARM 用 IDE プロジェクト管理およびビルドガイド』*を参照してください。

59 ページの*ターゲットハードウェアへの適合*も参照してください。

### リセットからの実行

［**指定位置まで実行**］オプション（［**デバッガ**］>［**設定**］ページ）では、デバッグセッションの起動時やリセット後の C-SPY の実行先の位置を指定します。C-SPY は指定された位置に一時的なブレークポイントを配置して、そこまでのすべてのコードを実行してから、その位置で停止します。

デフォルトでは、`main` 関数まで実行します。他の位置まで実行する場合は、その位置の名前を入力します。アセンブララベルかそれに相当するもの（関数名など）を指定できます。

チェックボックスを選択していない場合は、リセットごとにプログラムカウンタに通常のハードウェアリセットアドレスが格納されます。C-SPY によりリセットアドレスが設定されます。

C-SPY の起動時にブレークポイントが設定されていない場合は、ステップ実行をする必要があることと、それには非常に時間がかかることを通知する警告メッセージが表示されます。そこでステップ実行を継続するか、最初の命令で停止するかを選択できます。最初の命令で停止することを選択した場合、デバッガは［**指定位置まで実行**］ボックスで入力した位置ではなく、`PC`（プログラムカウンタ）に格納されているデフォルトリセット位置から実行を開始します。

**注 :** C-SPY シミュレータではブレークポイントに制限がないため、このメッセージは表示されません。

### セットアップマクロファイルの使用

セットアップマクロファイルは、C-SPY の起動時に自動的にロードするように指定するマクロファイルです。セットアップマクロ関数とシステムマクロを使用することにより、ニーズに合せたアクションを実行するセットアップマクロファイルを定義できます。したがって、セットアップマクロファイルをロードすることによって、アクションを自動的に実行するように C-SPY を初期化できます。

セットアップマクロファイルと関数の詳細については、393 ページの *C-SPY マクロの概要*を参照してください。セットアップマクロファイルの使用例については、59 ページの *C-SPY の起動前にターゲットハードウェアを初期化する*を参照してください。

**セットアップマクロファイルを登録するには、以下の手順に従います。**

1 C-SPY を起動する前に、**［プロジェクト］**>**［オプション］**>**［デバッガ］**>**［設定］**を選択します。

2 **［マクロファイルの使用］**を選択して、`Setup.mac` というようにセットアップマクロファイルのパスと名前を入力します。ファイル名拡張子の入力を省略した場合、拡張子は `mac` とみなされます。

### デバイス記述ファイルの選択

C-SPY はデバイス記述ファイルを使用して、デバイス固有の情報を処理します。

デフォルトのデバイス記述ファイル（IAR 固有の `ddf` ファイルまたは CMSIS システムビュー記述ファイル）が、プロジェクトの設定に基づいて自動的に選択されます。デフォルトのファイルをオーバライドする場合、デバイス記述ファイルを選択する必要があります。IAR システムズのデバイス記述ファイルは、`arm¥config` ディレクトリにあり、ファイル名拡張子は `ddf` です。

デバイス記述ファイルの詳細については、59 ページの *ターゲットハードウェアへの適合*を参照してください。

**デフォルトのデバイス記述ファイルをオーバライドするには、次の手順に従います。**

1 C-SPY を起動する前に、**［プロジェクト］**>**［オプション］**>**［デバッガ］**>**［設定］**を選択します。

2 デバイス記述ファイルの使用を有効にし、**［デバイス記述ファイル］**参照ボタンを使用してファイルを選択します。

**注 :** プロジェクトに使用されているデバイス記述ファイルを簡単に参照することができます。［**プロジェクト**］>［**デバイス記述ファイルを開く**］を選び、参照するファイルを選択します。

### プラグインモジュールのロード

［**プラグイン**］ページでは、デバッグセッションでロードして使用する C-SPY プラグインモジュールを指定します。IAR システムズやサードパーティ製のプラグインモジュールを使用できます。使用可能なモジュールについては、ソフトウェア販売代理店または IAR システムズの担当者までお問い合せください。また、IAR システムズの Web サイトでも情報を提供しています。

詳細については、534 ページの *プラグイン* を参照してください。

## C-SPY の起動

デバッガの設定が完了したら、デバッガセッションの開始準備が整いました。

以下のタスクについて説明します :

- デバッグセッションを開始する
- IDE 外部でビルドされた実行可能ファイルのロード
- ソースファイルがない状態でデバッグセッションを開始する
- 複数イメージのロード

### デバッグセッションを開始する

デバッグセッションは、現在の実行可能ファイルをロードしてもしなくても開始できます。

C-SPY を起動し、現在の実行可能ファイルをダウンロードするには、［**ダウンロードしてデバッグ**］ボタンをクリックしてください。または、［**プロジェクト**］>［**ダウンロードしてデバッグ**］を選択してください。

現在の実行可能ファイルをダウンロードせずに C-SPY を起動するには、［**ダウンロードせずにデバッグ**］ボタンをクリックしてください。または、［**プロジェクト**］>［**ダウンロードせずにデバッグ**］を選択してください。

### IDE 外部でビルドされた実行可能ファイルのロード

IDE 外でビルドされたアプリケーション、たとえばコマンドラインでビルドされたアプリケーションとともに C-SPY をロードすることもできます。外部でビルドされた実行可能ファイルをロードしてビルドのオプションを設定するには、最初にワークスペース内にそのファイル用のプロジェクトを作成する必要があります。

**外部でビルドされたファイルにプロジェクトを作成するには、次の手順に従います。**

1 ［プロジェクト］>［新規プロジェクトの作成］を選択して、プロジェクト名を指定します。

2 プロジェクトに実行可能ファイルを追加するには、［プロジェクト］>［ファイルの追加］を選択して、［ファイルの種類］ドロップダウンリストで［すべてのファイル］を選択します。実行可能ファイルを指定します。

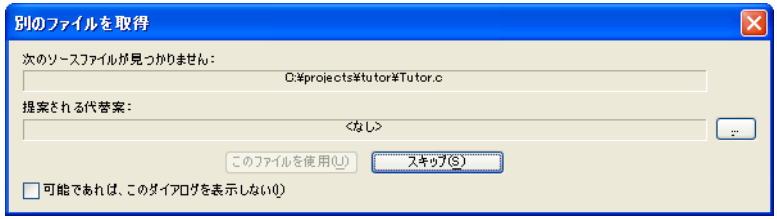
3 実行可能ファイルを起動するには、［ダウンロードしてデバッグ］ボタンをクリックします。プロジェクトは、実行可能ファイルをリビルドするたびに再利用できます。

このようなプロジェクトで設定する意味があるプロジェクトオプションは、［一般オプション］カテゴリと［デバッガ］カテゴリにだけあります。プロジェクトの一般オプションは、実行可能ファイルをビルドしたときと同様に設定するようにしてください。

## ソースファイルがない状態でデバッグセッションを開始する

通常は、IAR Embedded Workbench IDE を使用してソースファイルを編集したり、プロジェクトのビルド、デバッグセッションを開始する場合、必要なすべてのファイルが入手できてプロセスは想定したとおりに機能します。

ただし、C-SPY は自動的にソースファイルを見つけることはできません。たとえば、アプリケーションが別のコンピュータでビルドされた場合、［別のファイルを取得］ダイアログボックスが表示されます。




通常は以下のような場合にこのダイアログボックスを使用できます。

- ソースファイルが使用できない場合。［可能であれば、このダイアログを表示しない］に続いて［スキップ］をクリックします。C-SPY は、単に入手可能なソースファイルがないと見なします。ダイアログボックスは再び表示されず、デバッグセッションではソースコードは表示されません。
- 代替のソースファイルが別の場所にある場合。代替のソースコードを指定して、［可能であれば、このダイアログを表示しない］をクリックし、［このファイルを使用］をクリックします。C-SPY は代替ファイルを使用する

ものと想定します。代替ファイルが指定されておらず、自動的に検索されないファイルが必要な場合を除いて、ダイアログボックスは再び表示されません。

IAR Embedded Workbench IDE を再起動すると、**［可能であれば、このダイアログを表示しない］**をクリックした場合でも、**［代替ファイルを指定］**ダイアログボックスが再び表示されます。これによって、以前の設定を変更する機会が提供されます。

詳細については、73 ページの*［代替ファイルを指定］ダイアログボックス*を参照してください。

### 複数イメージのロード

通常、デバッグ可能なアプリケーションは、デバッグ対象の 1 ファイルのみで構成されます。ただし、追加のデバッグファイル（イメージ）をロードすることもできます。すなわち、プログラム全体は複数のイメージで構成されることになります。

この機能は、プラットフォーム提供の機能に関する追加ライブラリが含まれるビルド済 ROM イメージと一緒にアプリケーションをデバッグする場合に便利です。ROM イメージとアプリケーションは、別々のプロジェクトを使用して IAR Embedded Workbench IDE でビルドされ、別々の出力ファイルを生成します。

複数のイメージがロードされている場合、ロードされたすべてのイメージを合わせたデバッグ情報が表示されます。［イメージ］ウィンドウでは、1 つのイメージまたは全部のイメージのデバッグ情報を表示するかを選択できます。

**C-SPY の起動時に追加のイメージをロードするには、以下の手順に従います。**

1 **［プロジェクト］>［オプション］>［デバッガ］>［イメージ］**を選択して、ロードするイメージを最高 3 つまで指定します。詳細については、533 ページの*イメージ*を参照してください。

2 デバッグセッションを開始します。

特定の瞬間で追加のイメージをロードするには、以下の手順に従います。

`__loadImage` システムマクロを使用し、396 ページの *C-SPY マクロの使用*にある方法のいずれかによって実行します。

ロードされたイメージのリストを表示するには、次の手順に従います。

**［表示］**メニューから**［イメージ］**を選択します。イメージのウィンドウが表示されます（72 ページの*［イメージ］ウィンドウ*を参照）。

# ターゲットハードウェアへの適合

以下のトピックについて説明します:

- デバイス記述ファイルの修正
- C-SPY の起動前にターゲットハードウェアを初期化する
- メモリの再配置
- デバイスのサポートを目的とした事前定義済みの C-SPY マクロの使用

177 ページの*使用するデバイスのメモリに合わせた C-SPY の設定*も参照してください。

## デバイス記述ファイルの修正

C-SPY は、製品に付属のデバイス記述ファイルを使用して、さまざまなターゲット固有の調整を行います（55 ページの*デバイス記述ファイルの選択*を参照）。これらには以下のようなデバイス固有の情報が含まれています。

- 周辺ユニットおよびそれらのグループにおけるレジスタの定義。
- 割込み定義（Cortex-M デバイス専用）（369 ページの*割込み*を参照）。

通常はデバイス記述ファイルを修正する必要はありません。ただし、なんらかの理由によって事前定義が十分でない場合、ファイルを編集できます。ただし、これらの記述構文の形式は、製品の今後のアップグレードで更新される可能性がある点に注意してくだい。

ニーズに最適なデバイス記述ファイルのコピーを作成し、ファイルの記述に応じて修正します。変更を有効にするには、プロジェクトを再度読み込んでください。

I-jet/JTAGjet または I-jet Trace のデバッグプローブを使用し、変更したデバイス記述ファイルに修正済みのメモリ範囲が含まれる場合、**[メモリ構成]** ダイアログボックスで **[出荷時の設定を使用]** オプションを必ず選択してください。

デバイス記述ファイルの構文については、`arm¥doc` ディレクトリの *『ARM 用 IAR Embedded Workbench デバイス記述ファイルのフォーマット』ガイド* (`EWARM_DDFFormat.pdf`) に説明があります。

デバイス記述ファイルのロード方法に関する情報については、55 ページの*デバイス記述ファイルの選択*を参照してください。

## C-SPY の起動前にターゲットハードウェアを初期化する

C-SPY マクロを使用して、C-SPY が起動する前にターゲットハードウェアを初期化できます。たとえば、コードをダウンロードする前に有効にしなければならない外部メモリをハードウェアで使用する場合、アプリケーションを

ダウンロードする前に、C-SPY はこのアクションを実行するマクロを必要とします。

1 新しいテキストファイルを作成して、マクロ関数を定義します。

組込みセットアップマクロ関数の execUserPreload を使用することにより、ターゲットシステムとの通信が確立されてから C-SPY がアプリケーションをダウンロードするまでにマクロ関数が実行されます。

たとえば、外部の SDRAM を有効にするマクロは以下のようになります :

```
/* 使用するマクロ関数 */
enableExternalSDRAM()
{
  __message "Enabling external SDRAM¥n";
  __writeMemory32(...);
}

/* セットアップマクロが実行タイミングを決定 */
execUserPreload()
{
  enableExternalSDRAM();
}
```

2 ファイル名拡張子を mac としてこのファイルを保存します。

3 C-SPY を起動する前に、**［プロジェクト］> ［オプション］> ［デバッガ］**を選択して、**［設定］**タブをクリックします。

4 オプション **［セットアップファイルの使用］**を選択して、作成したマクロファイルを選択します。

これで、セットアップマクロが C-SPY の起動シーケンス中にロードされるようになります。

### メモリの再配置

ARM を基本とする多くのプロセッサに共通する機能は、メモリの再配置機能です。メモリコントローラでは、リセット後に、フラッシュのような不揮発性メモリにアドレスのゼロをマッピングするのが一般的です。メモリコントローラを構成することで、RAM をアドレスマップのゼロに配置し、不揮発性メモリをアドレスマップの上位に配置するように、システムメモリを再配置することができます。再配置することで、例外テーブルは RAM に置かれ、ターゲットハードウェアにコードをダウンロードしたときに簡単に修正できます。

メモリコントローラを設定してからアプリケーションコードをダウンロードする必要があります。これを実行する最も良い方法は、コードのダウンロードを行う前に C-SPY マクロ関数の execUserPreload() を実行することです。

マクロ関数 __writeMemory32() では、メモリコントローラに必要な初期化を実行します。

以下の例では、Atmel AT91SAM7S256 チップでメモリを再配置するために使用するマクロについて示します。同様なしくみは他の ARM ベンダのプロセッサに存在します。

```
execUserPreload()
{
  // REMAP コマンド
  // 1 を MC_RCR（MC 再配置制御レジスタ）に書き込む
  // 再配置ビットの切替え
  __writeMemory32(0x00000001, 0xFFFFFF00, "Memory");
}
```

設定マクロ execUserReset() は、C-SPY リセット後にマッピングするメモリの再初期化と同じように定義する必要があることに注意してください。これは、ハードウェアデバッガシステムを設定して C-SPY リセットでハードウェアリセットを行う場合に（たとえば __hwReset() を execUserReset() マクロに追加）、必要となることがあります。

C-SPY にマクロファイルをインストールする方法についての詳細は、397 ページの *セットアップマクロとセットアップファイルによる登録と実行*を参照してください。使用するマクロ関数の詳細については、410 ページの *C-SPY システムマクロについてのリファレンス情報*を参照してください。

### デバイスのサポートを目的とした事前定義済みの C-SPY マクロの使用

一部の ARM デバイスでは、特定のデバイスサポート付きの事前定義済み C-SPY マクロが用意されており、通常これらはチップメーカが提供します。これらのマクロは、デバイスに固有な特定のタスクを実行する際に便利です。

これらのマクロは、[マクロクイック起動] ウィンドウを使用して簡単にアクセス、起動することができます。

## デバッガ起動の概要

起動フローの理解と実施を簡単に行うため、以下の図で、C-SPY およびターゲットハードウェアが実行する処理フローと、事前定義された C-SPY 設定マクロの実行について説明します。1 つめの図がフラッシュ内のデバッグコードであり、もう 1 つの図が RAM 内のデバッグコードです。

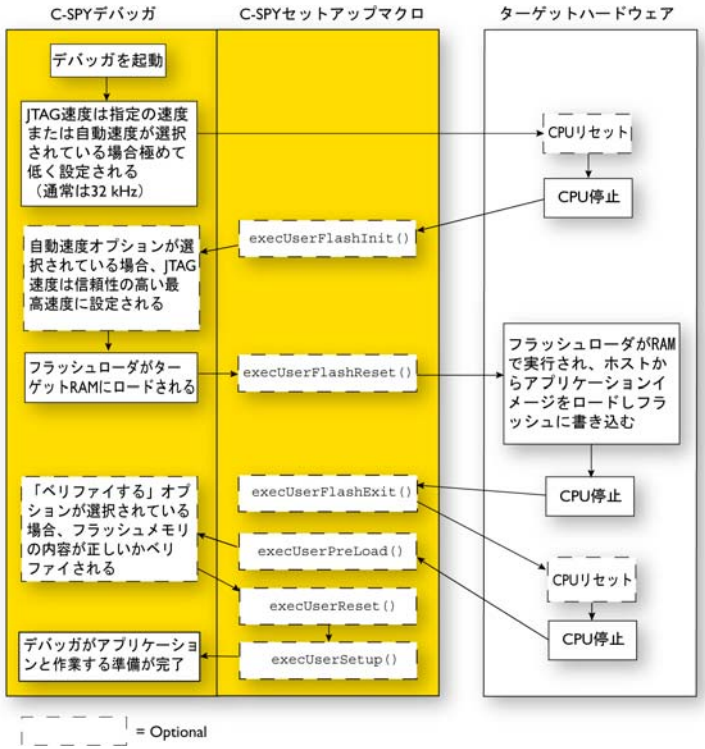
以下のタスクについて説明します:

- フラッシュのコードのデバッグ
- RAM のコードのデバッグ

C-SPY システムマクロの詳細については、本ガイドの *C-SPY マクロ*の章を参照してください。

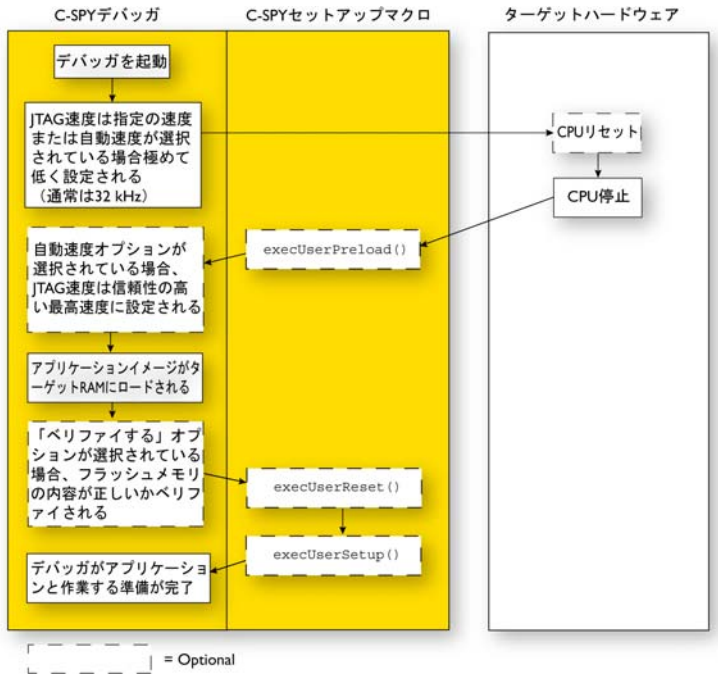
## フラッシュのコードのデバッグ

次の図は、フラッシュメモリのコードをデバッグする場合のデバッガの起動を示します。



### RAM のコードのデバッグ

次の図は、RAM のコードをデバッグする場合のデバッガの起動を示します。




## サンプルプロジェクトの実行

リファレンス情報：

- サンプルプロジェクトの実行

### サンプルプロジェクトの実行

アプリケーションのサンプルは IAR Embedded Workbench に同梱されています。これらのサンプルを使用して、IAR システムズの開発ツールを使用する準備を行えます。また、これらのサンプルを基にして、アプリケーションプロジェクトを開始することもできます。

サンプルはすぐ使用できるようになっています。既製のワークスペースファイルが、ソースコードファイルおよび関連する他のすべてのファイルとともに提供されます。

**サンプルプロジェクトを実行するには、次の手順に従います。**

1. ［ヘルプ］>［インフォメーションセンタ］を選択して、［サンプルプロジェクト］をクリックします。
2. デバイスに合ったチップメーカのダウンロードボタンをクリックします。
3. 表示されるダイアログボックスで、サンプルの取得元を選択します。以下から選択します。
   - IAR システムズからダウンロード
   - インストール用 DVD からコピー。この場合、［参照］ボタンを使用して必要なサンプルの自己解凍アーカイブを探します。アーカイブは DVD の `¥examples-archive` ディレクトリにあります。

   選択されたデバイスベンダのサンプルがコンピュータに抽出されます（使用する Windows オペレーティングシステムに応じて、`Program Data` ディレクトリまたは対応するディレクトリ）。
4. ダウンロードしたサンプルのリストから、チップメーカをクリックして使用する特定の評価ボードまたはスターターキットを探します。


5. ［プロジェクトを開く］ボタンをクリックします。
6. 表示されたダイアログボックスで、プロジェクトの配置先フォルダを選択します。
7. ［ワークスペース］ウィンドウに、使用可能なサンプルプロジェクトが表示されます。プロジェクトを 1 つ選択します。アクティブなプロジェクト（太字で強調表示）でない場合は、そのプロジェクトを右クリッして、コンテキストメニューから［アクティブに設定］を選択します。

8 プロジェクト設定を表示するには、プロジェクトを選択して、コンテキストメニューから［**オプション**］を選択します。［**一般オプション**］>［**ターゲット**］>［**プロセッサ選択**］および［**デバッガ**］>［**設定**］>［**ドライバ**］の設定を確認します。プロジェクトのその他の設定については、選択したターゲットシステムに合わせて設定されます。

C-SPY オプションの詳細と、ターゲットボード操作用に C-SPY を設定する方法については、529 ページの*［デバッガ］オプション*を参照してください。

**[OK]** をクリックして、プロジェクトの［**オプション**］ダイアログボックスを閉じます。

9 アプリケーションをコンパイルしてリンクするには、［**プロジェクト**］>［**メイク**］を選択するか、［**メイク**］ボタンをクリックします。

10 C-SPY を起動するには、［**プロジェクト**］>［**デバッグ**］を選択するか、［**ダウンロードしてデバッグ**］ボタンをクリックします。C-SPY がターゲットシステムとの接続を確立できない場合は、587 ページの*問題の解決*を参照してください。

11 ［**デバッグ**］>［**移動**］を選択するか、［**移動**］ボタンをクリックしてアプリケーションを起動します。

実行を停止するには、［**停止**］ボタンをクリックします。

# C-SPY の起動についてのリファレンス情報

リファレンス情報：

- 65 ページの *C-SPY デバッガメインウィンドウ*
- 72 ページの*［イメージ］ウィンドウ*
- 73 ページの*［代替ファイルを指定］ダイアログボックス*
- 74 ページの*［サンプルプロジェクトの取得］ダイアログボックス*

関連項目：

- *『ARM 用 IDE プロジェクト管理およびビルドガイド』*のデバッガのツールオプション。

## C-SPY デバッガメインウィンドウ

デバッグセッションを開始すると、IAR Embedded Workbench IDE のメインウィンドウに、以下のデバッガ専用項目が表示されます。

- アプリケーションの実行、デバッグ用コマンドが含まれる**デバッグ**専用メニュー

- 使用する C-SPY ドライバに応じた、ドライバ固有のメニュー。本書では通常 *ドライバメニュー*と表記します。通常は、このメニューには、ドライバ専用のウィンドウやダイアログボックスを表示するためのメニューコマンドが表示されます。
- デバッグ用ツールバー
- C-SPY 専用のウィンドウ、ダイアログボックス

C-SPY メインウィンドウは、製品インストールのどのコンポーネントを使用するかによって外観が異なる場合があります。

### メニューバー

これらのコマンドは、デバッグセッション中に使用できます。

**デバッグ**

ソースアプリケーションを実行およびデバッグするコマンドを提供します。ほとんどのコマンドは、デバッグツールバーのアイコンボタンからも実行できます。

***C-SPY ドライバメニュー***

C-SPY ドライバに固有のコマンドを提供します。このドライバ固有のメニューは、ドライバの使用時にのみ利用可能です。ドライバ固有メニューコマンドについては、571 ページの *C-SPY ドライバメニューのリファレンス情報*を参照してください。

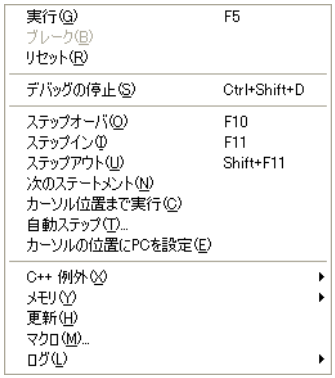
**逆アセンブリ**

ソースアプリケーションを実行およびデバッグするコマンドを提供します。

### [デバッグ] メニュー

[**デバッグ**] メニューは、デバッグセッション中に使用できます。[**デバッグ**] メニューには、ソースアプリケーションの実行 / デバッグ用コマンドが表示

されます。ほとんどのコマンドは、デバッグツールバーのアイコンボタンからも実行できます。




以下のコマンドがあります。

**［実行］(F5)**

現在の文 / 命令から、ブレークポイントかプログラム終了までコードを実行します。

**［ブレーク］**

アプリケーション実行を停止します。

**［リセット］**

ターゲットプロセッサをリセットします。ドロップダウンボタンをクリックして、追加コマンドを持つメニューにアクセスします。

**［*'label'* まで実行］**を有効にします。*label* は通常 `main` です。デバッグセッションを終了せずに、プロジェクトオプション**［指定位置まで実行］**を有効または無効にします。このメニューコマンドは、**［オプション］**ダイアログボックスで**［指定位置まで実行］**を有効にしたときのみ利用可能です。

*［リセット方式］*には、使用する C-SPY ドライバでサポートされているリセット方式のリストが含まれます。つまり、デバッグセッションを終了しなくても、初期設定で使用さているものとは異なるリセット方式を選択できます。［リセット方式］は、使用する C-SPY ドライバが別のリセット方式をサポートする場合のみ利用可能です。

**［デバッグ停止］(Ctrl+Shift+D)**

デバッグセッションを停止し、プロジェクトマネージャに戻ります。

**［ステップオーバ］(F10)**

C/C++ 関数やアセンブラサブルーチンに入らずに、次の文 / 関数呼出し / 命令を実行します。

**［ステップイン］(F11)**

C/C++ 関数やアセンブラサブルーチンに入って、次の文 / 命令 / 関数呼出しを実行します。

**［ステップアウト］(Shift+F11)**

現在の文から、現在の関数の呼出し後の文までを実行します。

**次の実行文**

各関数呼び出しを停止せずに次の文を直接実行します。

**カーソルまで実行**

現在の文 / 命令から、選択した文 / 命令までコードを実行します。

**自動ステップ**

自動ステップをカスタマイズしたり実行できるダイアログボックスを表示します（97 ページの *［自動ステップの設定］ダイアログボックス* を参照）。

**次の実行文の設定**

プログラムカウンタをカーソル位置に直接移動します。ソースコードは実行しません。ただし、プログラムフローに異常が生じ、予期しない効果が発生することがあります。

**C++ 例外 >**
**throw 時にブレーク**

ターゲットアプリケーションが `throw` 文を実行したときに、実行が停止するように指定します。

この機能を使用するには、オプションの**［低レベルインタフェースのライブラリ実装］**を選択し、**C++** 規格の言語オプションを [C++] にしてアプリケーションをビルドする必要があります。

**C++ 例外 >**
**例外が catch されなかったときにブレーク**

`catch` 文を照合して取得されない例外をターゲットアプリケーションが throw したときに、実行が停止するように指定します。

この機能を使用するには、オプションの**［低レベルインタフェースのライブラリ実装］**を選択し、**C++** 規格の言語オプションを [C++] にしてアプリケーションをビルドする必要があります。

**メモリ > セーブ**

特定のメモリエリアの内容をファイルに保存できるダイアログボックスを表示します（184 ページの *［メモリセーブ］ダイアログボックス*を参照）。

**メモリ > リストア**

特定のメモリゾーンにファイルの内容を Intel-extended または Motorola s-record などのフォーマットでロードできるダイアログボックスを表示します（185 ページの *［メモリリストア］ダイアログボックス*を参照）。

**更新**

すべてのデバッガウィンドウの内容を更新します。ウィンドウの更新は自動的に行われるため、C-SPY が検出できない方法でターゲットメモリが修正された場合など、通常の状態でない場合にのみ更新が必要です。［逆アセンブリ］ウィンドウに表示されたコードが変更された場合にも有効です。

**マクロ**

マクロファイルと関数を一覧表示、登録、編集するためのダイアログボックスを表示します（396 ページの *C-SPY マクロの使用*を参照）。

**ログ > ログファイルの設定**

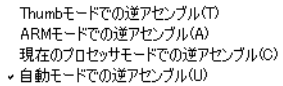
［デバッグログ］ウィンドウの内容をファイルに記録することもできるダイアログボックスを表示します。ログファイルの種類と場所を選択できます。以下の項目から選択して記録できます。エラー、ワーニング、システム情報、ユーザメッセージのいずれか、またはすべて。95 ページの *［ログファイル］ダイアログボックス*を参照してください。

**ログ > ターミナル I/O ログファイルの設定**

シミュレーションされたターゲットアクセス通信をファイルに記録することもできるダイアログボックスを表示します。ログファイルの場所を選択できます。93 ページの *［ターミナル I/O ログファイル］ダイアログボックス*を参照してください。

## ［逆アセンブリ］メニュー

**［逆アセンブリ］**メニューは、C-SPY の実行中のみ使用できます。このメニューには、ソースアプリケーションの実行 / デバッグ用コマンドが表示されます。ほとんどのコマンドは、デバッグツールバーのアイコンボタンからも実行できます。




メニューのコマンドを使用して、使用する逆アセンブリモードを選択します。

**注 :** 逆アセンブリモードを変更した後は、**［デバッグ］**メニューの**［更新］**コマンドを使用して［逆アセンブリ］ウィンドウの表示内容を更新してください。

以下のコマンドがあります。

| | |
|---|---|
| **Thumb モードでの逆アセンブル** | アプリケーションを Thumb モードで逆アセンブルします。 |
| **ARM モードでの逆アセンブル** | アプリケーションを ARM モードで逆アセンブルします。 |
| **現在のプロセッサモードでの逆アセンブル** | アプリケーションを現在のプロセッサモードで逆アセンブルします。 |
| **自動モードでの逆アセンブル** | アプリケーションを自動モードで逆アセンブルします。デフォルトのオプションです。 |

85 ページの*逆アセンブリウィンドウ*も参照してください。

### C-SPY のウィンドウ

使用する C-SPY ドライバに応じて、デバッグセッション中に C-SPY に固有の以下のウィンドウが使用できます。

- C-SPY デバッガメインウィンドウ
- 逆アセンブリウィンドウ
- ［メモリ］ウィンドウ
- ［シンボルメモリ］ウィンドウ
- ［レジスタ］ウィンドウ
- ［ウォッチ］ウィンドウ
- ［ローカル］ウィンドウ
- ［自動］ウィンドウ
- ［ライブウォッチ］ウィンドウ
- ［クイックウォッチ］ウィンドウ
- ［静的変数］ウィンドウ
- ［コールスタック］ウィンドウ

- ［トレース］ウィンドウ
- ［関数トレース］ウィンドウ
- ［タイムライン］ウィンドウ
- ［ターミナル I/O］ウィンドウ
- ［コードカバレッジ］ウィンドウ
- ［関数プロファイラ］ウィンドウ
- ［イメージ］ウィンドウ
- ［スタック］ウィンドウ
- ［シンボル］ウィンドウ

使用する C-SPY ドライバに応じて、他のウィンドウも使用可能です。

### C-SPY ウィンドウでの編集

［メモリ］、［シンボルメモリ］、［レジスタ］、［自動］、［ウォッチ］、［ロケール］、［静的］、［ライブウォッチ］、［クイックウォッチ］の各ウィンドウの内容を編集できます。

これらのウィンドウ内容の編集には、以下のキー操作を使用します。

| | |
|---|---|
| **Enter** | 項目を編集可能にします。また、新しい値を保存します。 |
| **Esc** | 新しい値を取り消します。 |

**［式］**フィールドが編集可能なウィンドウおよび［クイックウォッチ］ウィンドウでは、セミコロンに続いて整数を追加すると表示されるエレメントの数を指定できます。たとえば、`myArray` という配列の最初の 3 つのエレメントのみ、またはポインタによって示されるエレメントから続いて 3 つのエレメントを表示するには、次のように記述します。

```
myArray;3
```

`myPtr`、`myPtr+1`、`myPtr+2` によってポイントされる 3 つのエレメントを表示するには、以下のように記述します。

```
myPtr;3
```

または、最初のエレメントを指定するコンマと整数を 1 つずつ追加します。たとえば、10 から 14 のエレメントを表示するには、次のように記述します。

```
myArray;5,10
```

`myPtr+10`、`myPtr+11`、`myPtr+12`、`myPtr+13`、`myPtr+14` を表示するには、以下のように記述します。

```
myPtr;5,10
```

**注 :** ポインタの場合、表示するエレメント数に組み込まれた制限はなく、ポインタ値の検証も行われません。

## ［イメージ］ウィンドウ

［イメージ］ウィンドウは［**表示**］メニューから利用できます。


［イメージ］ウィンドウには、現在ロードされたすべてのイメージ（デバッグファイル）が一覧表示されます。

通常、ソースアプリケーションは、デバッグ対象の 1 つのイメージのみで構成されます。ただし、追加のイメージをロードすることもできます。すなわち、デバッグ対象のユニット全体は複数のイメージで構成されることになります。

### 要件

ありません。このウィンドウは常に使用できます。

### 表示エリア

C-SPY はロードされているすべてのイメージからのデバッグ情報を同時に使用するか、または 1 回ごとに 1 つのイメージから使用できます。そのイメージだけの情報を表示するには、行をダブルクリックします。現在の選択内容が強調表示されます。

このエリアには、以下の列にロードされたイメージが一覧表示されます。

**名前**

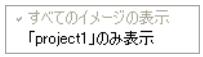
ロードされているイメージ名。

**パス**

ロードされているイメージのパス。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**すべてのイメージの表示**

ロードされたすべてのデバッグイメージのデバッグ情報を表示します。

***イメージのみを表示***

選択されたデバッグイメージのデバッグ情報を表示します。

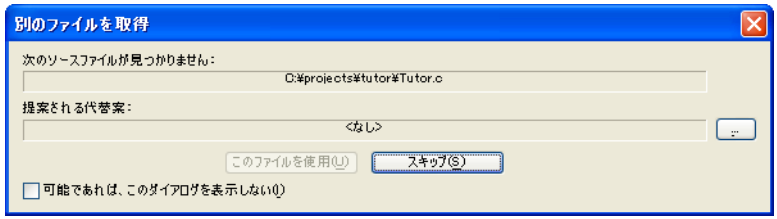
### 関連情報

関連情報については、以下を参照してください。

- 58 ページの *複数イメージのロード*
- 533 ページの *イメージ*
- 432 ページの *__loadImage*

## ［代替ファイルを指定］ダイアログボックス

C-SPY がロードするソースファイルを自動的に見つけられない場合（アプリケーションが別のコンピュータでビルドされた場合など）、**［別のファイルを取得］**ダイアログボックスが表示されます。




### 次のソースファイルが見つかりません

見つからないソースファイル。

### 提案される代替案

代わりのファイルを指定します。

### このファイルを使用

代替ファイルを指定した後に、**［このファイルを使用］**によって、そのファイルを要求したファイルのエイリアスとして確定します。このアクションを選

択した後は、これらのファイルが最初に選択された代替ファイルと似たようなディレクトリ構造にある場合、C-SPY は自動的に他のソースファイルを検索します。

次にデバッグセッションを開始するときに、選択した代替ファイルが自動的にあらかじめロードされます。

**スキップ**

このデバッグセッションについて、C-SPY はソースファイルが入手できないと想定します。

**可能であれば、このダイアログを表示しない**

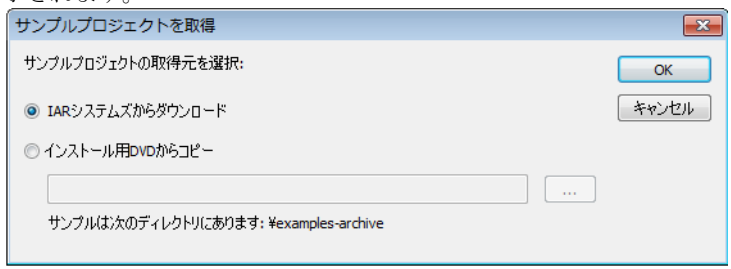
不足しているソースファイルについてダイアログボックスを表示する代わりに、C-SPY は以前の応答を使用します。

**関連情報**

関連情報については、57 ページの *ソースファイルがない状態でデバッグセッションを開始する*を参照してください。

## [サンプルプロジェクトの取得] ダイアログボックス

[**サンプルプロジェクトの取得**] ダイアログボックスは、IAR インフォメーションセンタでチップメーカのダウンロードボタンをクリックしたときに表示されます。




63 ページの *サンプルプロジェクトの実行*も参照してください。

### IAR システムズからダウンロード

IAR システムズからアプリケーションをダウンロードします。

### インストール用 DVD からコピー

インストール用 DVD からアプリケーションをコピーします。この場合、［参照］ボタンを使用して必要なサンプルの自己解凍アーカイブを探します。アーカイブは DVD の `¥examples-archive` ディレクトリにあります。

選択されたデバイスベンダのサンプルがコンピュータに抽出されます（使用する Windows オペレーティングシステムに応じて、`Program Data` ディレクトリまたは対応するディレクトリ）。

# アプリケーションの実行

- アプリケーション実行の概要
- アプリケーション実行についてのリファレンス情報

## アプリケーション実行の概要

以下のトピックについて説明します:

- アプリケーション実行の概要について
- ソースモードと逆アセンブリモードのデバッグ
- ステップ実行
- ステップインの速度
- アプリケーションの実行
- 強調表示
- コールスタック情報
- ターミナル I/O
- デバッグログ

### アプリケーション実行の概要について

C-SPY では、アプリケーションの実行をモニタおよび制御することができます。シングルステップでアプリケーションを実行し、ブレークポイントを設定することにより、変数やレジスタの値など、アプリケーション実行の詳細を調べることができます。また、コールスタックを使用して、関数のコールチェーン内でステップの前後を調べることもできます。

ターミナル I/O およびデバッグログ機能を使用すると、アプリケーションを操作できます。

そのようなコマンドは、**[デバッグ]** メニューとツールバーにあります。

### ソースモードと逆アセンブリモードのデバッグ

C-SPY では、必要に応じてソースモードと逆アセンブリモードを切り替えてデバッグできます。

ソースモードのデバッグは、アプリケーションを短期間で簡単に開発するための手段であり、コンパイラやアセンブラがどのようにコードを実装したか

を知る必要はありません。エディタウィンドウでは、一度に 1 文ずつアプリケーションを実行しながら、変数やデータ構造体の値をモニタできます。

逆アセンブリモードのデバッグでは、アプリケーションの重要なセクションに焦点を当てて、アプリケーションコードを高い精度で制御できます。逆アセンブリウィンドウを開くと、アプリケーションが、ソースコードではなく実際のメモリの内容に基づいて、ニーモニックアセンブリリストとして表示され、一度にひとつのマシン命令をアプリケーションで実行できます。

どちらのモードでデバッグしている場合でも、レジスタとメモリを表示したり、その内容を書き換えたりすることができます。

## ステップ実行

C-SPY は文単位でステップを実行できるため、行単位でステップを実行する他の多くのデバッガよりも高い精度でステップ実行できます。コンパイラは、文や関数呼出しごとに*ステップポイント*という形式で詳細なステップ実行情報を生成します。すなわち、コマンドでステップインするか、ステップオーバするかを選択する可能性のあるソースコード上の位置に、ステップポイントを生成します。ステップポイントは文だけでなく、関数呼出しの位置にも生成されるので、単純に文をステップ実行するよりも詳細にステップ実行できます。

ステップ実行が低速になる要素はいくつかあります。速度が遅すぎると感じる場合は、588 ページの*ステップ速度が遅い場合*のヒントを参照してください。

## ステップコマンド

ステップコマンドには、以下の 4 つのコマンドがあります。

- **ステップイン**
- **ステップオーバ**
- **次の実行文**
- **ステップアウト**

**［自動ステップの設定］**ダイアログボックスを使用して、シングルステップの実行を自動化できます。詳細については、97 ページの*［自動ステップの設定］ダイアログボックス*を参照してください。

コード外部で検出される例外（通常はステップの一部として実行されます）がアプリケーションに含まれる場合、C-SPY は `catch` 文の位置でステップを終了します。

以下に示す例で、1 つ前のステップ実行の結果、現在 `f(i)` 関数呼出し（強調表示）の位置にいるものとします。

```
extern int g(int);
int f(int n)
{
 value = g(n-1) + g(n-2) + g(n-3);
 return value;
}
int main()
{
  ...
  f(i);
  value ++;
}
```

## ステップイン

ステップ実行中は、通常は関数にステップインして関数かサブルーチンの内部でステップ実行を継続することを考えます。[**ステップイン**] コマンドを実行すると、サブルーチン `g(n-1)` 内部の最初のステップポイントに移動します。

```
extern int g(int);
int f(int n)
{
 value = g(n-1) + g(n-2) + g(n-3);
 return value;
}
```

[**ステップイン**] コマンドは、通常の制御の流れに従い、次のステップポイントが同じであるか別の関数であるかどうかに関わらず、次のステップポイントまで実行します。

## ステップオーバ

[**ステップオーバ**] コマンドは、同じ関数の次のステップポイントまで実行します。呼び出された関数の内部にステップインすることはありません。上の例でステップオーバを実行すると、`g(n-2)` 関数呼出しまで実行します。この関数呼出しは独立した文ではなく、`g(n-1)` と同じ文にあります。このように

して、文の一部に興味のない関数呼出しがある場合にそこをスキップして、重要な部分だけをデバッグすることができます。

```
extern int g(int);
int f(int n)
{
 value = g(n-1) + g(n-2) + g(n-3);
 return value;
}
```

### 次の実行文が設定されました

[次のステートメント] コマンドは、この場合の次の文である `return value` まで一気に実行します。これによって、高速にステップを進めることができます。

```
extern int g(int);
int f(int n)
{
 value = g(n-1) + g(n-2) + g(n-3);
 return value;
}
```

### ステップアウト

関数内部では、必要に応じて、[ステップアウト] コマンドを実行すると、関数の最後に到達する前に、ステップアウトすることができます。これによって、関数呼出しの直後の文まで一気に実行します。

```
extern int g(int);
int f(int n)
{
 value = g(n-1) + g(n-2) g(n-3);
 return value;
}
int main()
{
  ...
  f(i);
  value ++;
}
```

複雑な文の一部である個別の関数にステップインできる機能は、複数回ネストしている関数呼出しを含む C ソースコードを使用している場合に、特に役に立ちます。また、コンストラクタ、デストラクタ、代入演算子、その他のユーザ定義演算子など、多くの間接的な関数呼出しを使用する傾向がある C++ でも、ステップイン機能が非常に役に立ちます。

細かいステップ実行は、状況によっては役立つ場合もありますが、不必要に処理を遅くすることもあります。そのため、文単位でステップ実行する機能があり、これによって高速なステップ実行が可能になります。

### ステップインの速度

C-SPY でのステップ実行は通常、ブレークポイントを使用して行われます。ステップコマンドを実行する際は、ブレークポイントが次の文に設定され、このブレークポイントに到達するまでプログラムが実行されます。ハードウェアデバッガシステムを使用してデバッグする場合、ハードウェアブレークポイントの数は限られています。通常これらは、フラッシュ /ROM メモリに配置されたコード内にステップブレークポイントを設定するために使用されます。たとえば、C の `switch` 文にステップインする場合、ブレークポイントはそれぞれの分岐に設定されるため、いくつかのハードウェアブレークポイントを消費することがあります。使用可能なハードウェアブレークポイントを超過すると、C-SPY はアセンブリレベルでシングルステップ に切り替わり、速度が非常に遅くなります。

このため、いくつハードウェアブレークポイントを使用しているかを把握して、そのうちいくつかがステップイン用に残っていることを確認すると役に立ちます。詳細については、137 ページの*ブレークポイントの設定元*を参照してください。

ハードウェアブレークポイントが限られていることに加えて、以下の問題もステップインの速度に影響することがあります。

- トレースまたは関数プロファイリングが有効かどうか。収集されたトレースデータが各ステップの後に処理されるため、ステップが低速になることがあります。対応するウィンドウを閉じてトレースデータの収集を無効にしても、十分ではない点に注意してください。代わりに、［トレース］および［関数プロファイリング］ウィンドウの両方で**［有効化 / 無効化］**ボタンを無効にする必要があります。
- ［レジスタ］ウィンドウが開いていて、SFR レジスタが表示されているかどうか。この場合、選択したレジスタグループにあるすべてのレジスタを各ステップの後にハードウェアから読み込む必要があるため、ステップ実行が低速になることがあります。これを解決するには、限られた SFR レジスタのみを表示するように次のどちらかを選択することができます。［ウォッチ］ウィンドウに `#SFR_name`（`#SFR_name` はモニタする SFR 名を示します）を入力するか、［レジスタ］ウィンドウで限られたグループの SFR を表示する独自のフィルタを作成します。178 ページの*アプリケーション固有のレジスタグループの定義*を参照してください。
- ［メモリ］または［シンボルメモリ］ウィンドウのどちらかが開いているかどうか。これによって、各ステップの後に認識できるメモリを読み取る必要があるため、ステップの実行が遅くなることがあります。

- ［ウォッチ］、［ライブウォッチ］、［ローカル］、［静的］など、式に関連するウィンドウが開いているかどうか。これらのウィンドウはすべて各ステップの後にメモリを読み込むため、ステップの速度が遅くなることがあります。
- ［スタック］ウィンドウが開いているかどうか、特にオプションの**［グラフィカルスタック表示とスタック使用トラッキングを有効にする］**が選択されているかどうか。このオプションを無効にするには、**［ツール］>［オプション］>［スタック］**を選択して無効にします。
- C-SPY とターゲットボード/エミュレータ間で設定された通信速度が遅すぎる場合、可能であれば速度を上げることを検討するべきです。

## アプリケーションの実行

### 実行

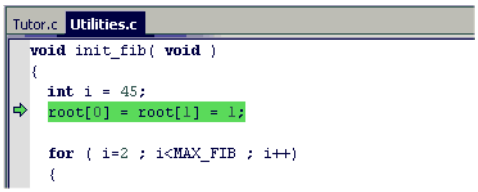
**［実行］**コマンドは、現在の位置からブレークポイントまたはプログラムの最後に到達するまで、続けて実行します。

### カーソルまで実行

**［カーソルまで実行］**コマンドは、ソースコードの現在カーソルのある位置まで実行します。**［カーソルまで実行］**コマンドは、［逆アセンブリ］ウィンドウと［コールスタック］ウィンドウでも使用できます。

### 強調表示

エディタウィンドウと［逆アセンブリ］ウィンドウの両方で実行を停止するたびに、C-SPY は対応する C/C++ ソースまたは命令を緑色で強調表示します。さらに緑色の矢印が、C/C++ ソースレベルでステップする場合はエディタウィンドウに、逆アセンブリレベルでステップする場合は［逆アセンブリ］ウィンドウに表示されます。これは、どのウィンドウがアクティブであるかによって決まります。アクティブなウィンドウがない場合、最後にどのウィンドウがアクティブだったかによって決まります。




関数呼出しのない単純な文の場合、通常は文全体が強調表示されます。複数の関数呼出しを含む文で停止した場合、**［ステップイン］**と**［ステップオーバ］**の違いを明確に表すために、最初の関数呼出しが強調表示されます。

ときどき、ソースウィンドウの文が通常の強調表示よりも薄い色で強調表示される場合があります。これは、プログラムカウンタはソース文を構成するアセンブラ命令を指していても、そこが正確なステップポイントではないことを意味します。これは、［逆アセンブリ］ウィンドウでステップ実行しているときによく発生します。プログラムカウンタがソース文の最初の命令を指している場合は、通常の強調表示色が使用されます。

### コールスタック情報

コンパイラは、大量のバックトレース情報を生成します。これにより、C-SPY は、実行に影響を及ぼすことなく、いつでも関数コールチェーン全体を表示できます。

この機能は通常、以下の 2 つの目的で使用します。

- 現在の関数の呼出し元のコンテキストを決定する場合
- 不正な変数やパラメータの値を検出したときに、その発生元をトレースして、コールチェーン内で問題が発生した関数を特定する場合

［コールスタック］ウィンドウには関数呼出しのリストが表示され、現在の関数が一番上になります。コールチェーンの関数を調べるときに、影響を受けたすべてのウィンドウの内容が更新され、特定のコールフレームの状態が表示されます。影響を受けるウィンドウには、エディタ、［ローカル］、［レジスタ］、［ウォッチ］、［逆アセンブリ］の各ウィンドウがあります。通常、関数がすべてのレジスタを使用することはないため、一部のレジスタは状態が未定義であり、そのときはダッシュ記号 (---) で表示されます。

エディタウィンドウと［逆アセンブリ］ウィンドウでは、最上位または現在のコールフレームは緑色で強調表示され、他のフレームを検証しているときは黄色で強調表示されます。

コールスタックで関数を選択し、**［カーソル位置まで実行］**コマンドをクリックしてその関数を実行すれば便利です。

アセンブラソースコードには、自動的にバックトレース情報が挿入されることはありません。適切な `CFI` アセンブラディレクティブをアセンブラソースコードに追加すると、アセンブラモジュールのコールチェーンも表示できるようになります。詳細については、*『ARM 用 IAR アセンブラリファレンスガイド』*を参照してください。

### ターミナル I/O

場合によっては、`stdin` や `stdout` を使用するアプリケーションの構文を、実際のハードウェアデバイスを入出力として使用しないでデバッグする必要があります。［ターミナル I/O］ウィンドウでは、アプリケーションへ入力したり、アプリケーションからの出力を表示したりできます。また、**［ターミナル**

**I/O ログファイル**］ダイアログボックスを使用して、ターミナル I/O をファイルに出力することもできます。

この機能は、以下の 2 つのコンテキストで使用します。

- アプリケーションが `stdin` と `stdout` を使用している場合
- デバッグトレース出力を生成する場合

詳細については、91 ページの*［ターミナル I/O］ウィンドウ*、93 ページの*［ターミナル I/O ログファイル］ダイアログボックス*を参照してください。

### デバッグログ

［デバッグログ］ウィンドウには、診断メッセージやマクロにより生成された出力、イベントログメッセージ、トレースについての情報など、デバッガの出力が表示されます。

これらの情報は、簡単に検証できるファイルにログとして記録しておくと便利です。デバッガの出力をファイルに記録することで、主に 2 つのメリットがあります。

- ファイルをエディタなどの別のツールで開くことができるので、ファイルの中で特に興味のある部分だけ調べることが可能
- ファイルには、どこでブレークポイントがトリガされたかなど、プログラム実行がどのように制御されたかの履歴が残る

## アプリケーション実行についてのリファレンス情報

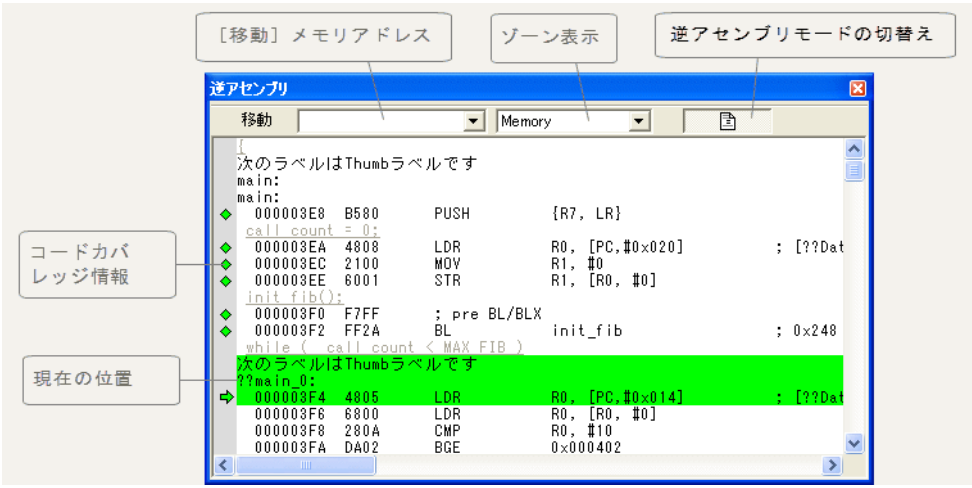
リファレンス情報：

- 85 ページの*逆アセンブリウィンドウ*
- 89 ページの*［コールスタック］ウィンドウ*
- 91 ページの*［ターミナル I/O］ウィンドウ*
- 93 ページの*［ターミナル I/O ログファイル］ダイアログボックス*
- 93 ページの*［デバッグログ］ウィンドウ*
- 95 ページの*［ログファイル］ダイアログボックス*
- 96 ページの*［アサートの報告］ダイアログボックス*
- 97 ページの*［自動ステップの設定］ダイアログボックス*

*『ARM 用 IDE プロジェクト管理およびビルドガイド』*のターミナル I/O オプションも参照してください。

## 逆アセンブリウィンドウ

［C-SPY 逆アセンブリ］ウィンドウは［表示］メニューから利用できます。




このウィンドウには、デバッグ中のアプリケーションが、逆アセンブリされたアプリケーションコードとして表示されます。

**［逆アセンブリ］ウィンドウでソースコードのデフォルト色を変更するには、次の手順に従います。**

1 ［ツール］>［オプション］>［デバッガ］を選択します。

2 デフォルトの色を設定するには、**［［逆アセンブリ］ウィンドウのソースコード色］**オプションを使用します。

関数に対応するアセンブラコードを表示するには、エディタウィンドウでその関数を選択し、［逆アセンブリ］ウィンドウにドラッグします。

### 要件

ありません。このウィンドウは常に使用できます。

### ツールバー

ツールバーの内容は以下のとおりです。

**移動**

表示するメモリアドレス（ロケーション）またはシンボルを指定できます。

**ゾーン**

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

**混在モードのトグル**

逆アセンブリしたコードだけを表示するか、対応するソースコードも併せて表示するかを選択します。ソースコードを表示するには、デバッグ情報付きでソースファイルをコンパイルしておく必要があります。

### 表示エリア

表示エリアには、逆アセンブリしたアプリケーションコードが表示されます。

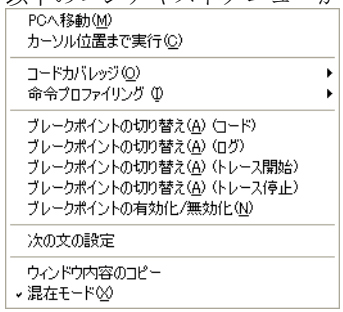
このエリアには以下のグラフィック要素が含まれます。

| | |
|---|---|
| 緑の強調表示 | 現在の位置は、次に実行されるアセンブラ命令を示します。［逆アセンブリ］ウィンドウの任意の行をクリックすると、その行にカーソルが移動します。また、移動キーを使用してカーソルを移動することもできます。 |
| 黄色の強調表示 | ［コールスタック］ウィンドウのフレーム間を移動したり、［トレース］ウィンドウでアイテム間を移動するときなど、現在の位置以外の位置を示します。 |
| 赤の点 | ブレークポイントを示します。ブレークポイントを設定するには、ウィンドウの左側の灰色で表示された余白部分をダブルクリックします。詳細については、133 ページの *ブレークポイント*を参照してください。 |
| 緑色のひし形 | 実行済みのコードを示します。すなわち、コードカバレッジです。 |

命令プロファイリングが有効化（コンテキストメニューで設定）されている場合、それぞれの命令が実行された回数に関する情報を表示する列が左側の余白部に追加されます。

## コンテキストメニュー

以下のコンテキストメニューがあります。




**注 :** このメニューの内容は一定ではありません。つまり、メニュー上のコマンドは、ご使用の製品パッケージに応じて異なる可能性があります。

以下のコマンドがあります。

**PC へ移動**

現在のプログラムカウンタ位置のコードを表示します。

**カーソルまで実行**

現在の位置からカーソルを含む行まで、アプリケーションを実行します。

**コードカバレッジ**

コードカバレッジ制御用コマンドのサブメニューを表示します。このコマンドは、使用しているドライバがサポートする場合のみ使用できます。

| | |
|---|---|
| **有効化** | コードカバレッジの有効 / 無効を切り替えます。 |
| **表示** | コードカバレッジの有効 / 無効の表示を切り替えます。実行されるコードは、緑色のひし形で示されます。 |
| **クリア** | すべてのコードカバレッジ情報を消去します。 |

**命令プロファイリング**

命令プロファイリング制御用コマンドのサブメニューを開きます。このコマンドは、使用しているドライバがサポートする場合のみ使用できます。

| | |
|---|---|
| **有効化** | 命令プロファイリングの有効 / 無効を切り替えます。 |
| **表示** | 命令プロファイリングの有効 / 無効の表示を切り替えます。それぞれの命令ごとに、命令の実行回数が左側の余白部に表示されます。 |
| **クリア** | すべての命令プロファイリング情報を消去します。 |

**ブレークポイントの切替え（コード）**

コードブレークポイントを設定 / 解除します。コードブレークポイントが設定されたアセンブラ命令および対応するすべてのラベルは、赤色で強調表示されます。詳細については、150 ページの *［コード］ブレークポイントダイアログボックス*を参照してください。

**ブレークポイントの切替え（ログ）**

ログでトレースを出力するためのブレークポイントを設定 / 解除します。ログブレークポイントが設定されたアセンブラ命令は赤色で強調表示されます。詳細については、155 ページの *［ログ］ブレークポイントダイアログボックス*を参照してください。

**ブレークポイントの切替え（トレース開始）**

トレース開始ブレークポイントを切替えます。ブレークポイントがトリガされると、トレースデータの収集が始まります。このメニューコマンドは、使用している C-SPY ドライバでトレースがサポートされている場合にのみ使用できます。詳細については、256 ページの *［トレース開始］ブレークポイントダイアログボックス*を参照してください。

**ブレークポイントの切替え（トレース停止）**

トレース停止ブレークポイントを切替えます。ブレークポイントがトリガされると、トレースデータの収集が終了します。このメニューコマンドは、使用している C-SPY ドライバでトレースがサポートされている場合にのみ使用できます。詳細については、258 ページの *［トレース停止］ブレークポイントダイアログボックス*を参照してください。

**ブレークポイントの有効化 / 無効化**

ブレークポイントを有効 / 無効にします。特定の行に複数のブレークポイントがある場合、［**有効 / 無効**］コマンドによってすべてのブレークポイントが影響を受けます。

**ブレークポイントの編集**

［ブレークポイントの編集］ダイアログボックスを表示します。このダイアログボックスを使用して、現在選択しているブレークポイントの編集ができます。選択した行に複数のブレークポイントがある場合、使用可能なすべてのブレークポイントの一覧を示すサブメニューがその行に表示されます。

**次の実行文の設定**

プログラムカウンタを挿入ポイントの命令のアドレスに設定します。

**ウィンドウ内容のコピー**

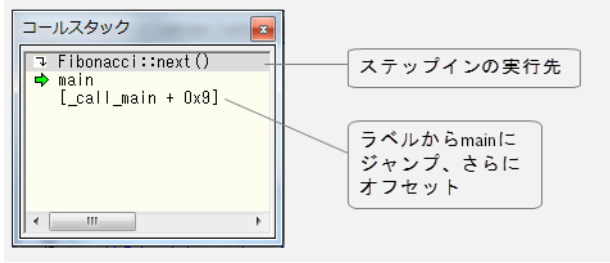
［逆アセンブリ］ウィンドウで選択した内容をクリップボードにコピーします。

**混在モード**

逆アセンブリしたコードだけを表示するか、対応するソースコードも併せて表示するかを選択します。ソースコードを表示するには、デバッグ情報付きでソースファイルをコンパイルしておく必要があります。

## ［コールスタック］ウィンドウ

［コールスタック］ウィンドウは**［表示］**メニューから利用できます。




C 関数コールスタックが、現在の関数とともに上部に表示されます。関数呼出しを調べるには、ダブルクリックします。C-SPY でそのコールフレームが代わりに表示されます。

次の**［ステップイン］**コマンドが関数呼出し内部に移動してステップ実行する場合は、ウィンドウ上部の灰色部分に関数名が表示されます。これは、C++ のコンストラクタ、デストラクタ、演算子のような暗黙的な関数呼出しの場合に特に便利です。

### 要件

ありません。このウィンドウは常に使用できます。

### 表示エリア

コマンド［**引数の表示**］が有効な場合、表示エリアにあるそれぞれのエントリは次のフォーマットになります。

*function*(*values*)***

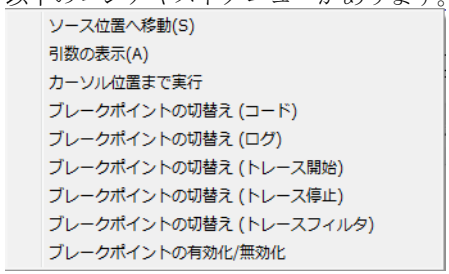
説明

*(values)* は現在のパラメータ値のリストか、関数がパラメータを受け入れない場合は空白です。

*** が表示されている場合、コンパイラによって関数がインライン化されていることを示します。関数のインライン化については、*『ARM 用 IAR C/C++ 開発ガイド』* を参照してください。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**ソースへ移動**

選択した関数を［逆アセンブリ］ウィンドウかエディタウィンドウで表示します。

**引数の表示**

関数の引数を表示します。

**カーソルまで実行**

コールスタックで選択した関数に戻るまで実行します。

**ブレークポイントの切替え（コード）**

コードブレークポイントを設定 / 解除します。

**ブレークポイントの切替え（ログ）**

ログブレークポイントを設定 / 解除します。

**ブレークポイントの切替え（トレース開始）**

トレース開始ブレークポイントを切替えます。ブレークポイントがトリガされると、トレースデータの収集が始まります。このメニューコマンドは、使用している C-SPY ドライバでトレースがサポートされている場合にのみ使用できます。

**ブレークポイントの切替え（トレース停止）**

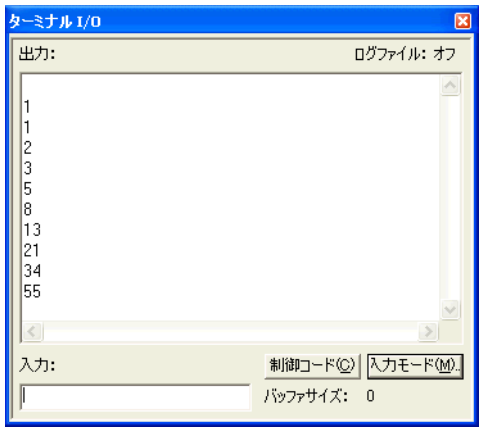
トレース停止ブレークポイントを切替えます。ブレークポイントがトリガされると、トレースデータの収集が停止します。このメニューコマンドは、使用している C-SPY ドライバでトレースがサポートされている場合にのみ使用できます。

**ブレークポイントの有効化 / 無効化**

選択したブレークポイントを有効 / 無効にします。

## ［ターミナル I/O］ウィンドウ

［**表示**］メニューを使用して、［ターミナル I/O］ウィンドウを開きます。




このウィンドウを使用してアプリケーションにデータを入力し、その出力を表示します。

**このウィンドウを使用するには、次のことを行う必要があります。**

- オプション［**セミホスティング**］または［**IAR ブレークポイント**］オプションを使用してアプリケーションをリンクします。

  それによって、C-SPY は `stdin`、`stdout`、`stderr` をこのウィンドウに転送します。［ターミナル I/O］ウィンドウが表示されていない場合は、入力が必要な場合に自動的に表示されます。出力の場合は自動表示されません。

以下の場合にリアルタイムのターミナル I/O を使用します。

| デバイス | 説明 |
|---|---|
| Cortex-M | アプリケーションの stdout が SWO を経由している。「233 ページの *[SWO 設定] ダイアログボックス*」の **ITM 事象ポート**オプションを参照してください。 |
| ARM7/ARM9（ARMxxx-S を含む）および C-SPY J-Link/J-Trace ドライバを使用する場合 | ファイル arm¥src¥debugger¥dcc¥DCC_Write.c をプロジェクトに追加することによって、DCC を I/O 出力に使用できます。DCC_write.c は、ライブラリ関数 write をオーバライドします。printf などの関数を使用して、テキストを［ターミナル I/O］ウィンドウに出力できます。<br>この場合、セミホスティングを無効にして、ブレークポイントを他の用途に解放できます。セミホスティングを無効にするには、**［一般オプション］>［ライブラリ設定］>［低レベルインタフェースのライブラリ実装］>［なし］**を選択します。 |

*表 5: リアルタイムのターミナル I/O*

### 要件

ありません。このウィンドウは常に使用できます。

### 入力

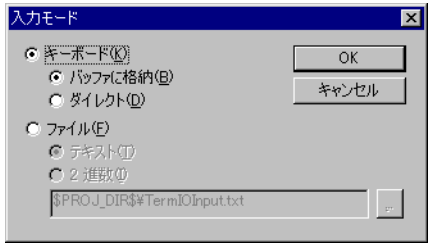
アプリケーションに入力するテキストを入力します。

### 制御コード

EOF（ファイル終端）や NUL などの特殊文字の入力メニューを開きます。


### 入力モード

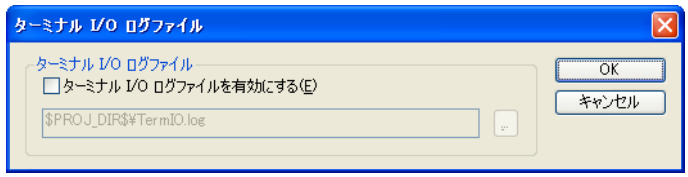
［入力モード］ダイアログボックスを開きます。ここではキーボードとファイルのどちらからデータを入力するかを選択します。



このダイアログボックスで使用可能なオプションのリファレンス情報については、*『ARM 用 IDE プロジェクト管理およびビルドガイド』*の「ターミナル I/O オプション」の項を参照してください。

## ［ターミナル I/O ログファイル］ダイアログボックス

**［ターミナル I/O ログファイル］**ダイアログボックスは、**［デバッグ］** > **［ログ］** > **［ターミナル I/O ログファイルの設定］**を選択して使用します。




このダイアログボックスを使用して、C-SPY からターミナル I/O の記録先ログファイルを選択します。

### 要件

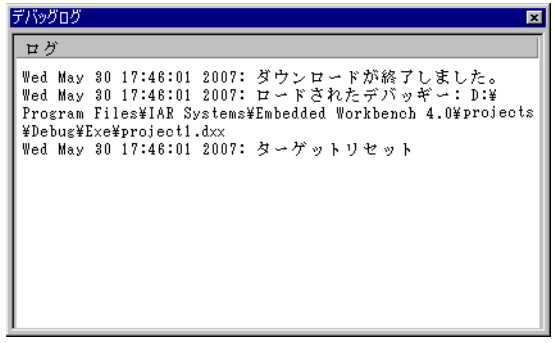
ありません。このダイアログボックスは常に使用できます。

### ターミナル IO ログファイル

ターミナル I/O のログを制御します。ターミナル I/O のファイルへのログを有効にするには、**［ターミナル IO ログファイルの有効化］**を選択してファイル名を指定します。デフォルトのファイル名拡張子は `log` です。参照ボタンを使用して選択することもできます。

## ［デバッグログ］ウィンドウ

［デバッグログ］ウィンドウは、**［表示］** > **［メッセージ］**を選択すれば使用できます。



このウィンドウには、診断メッセージやマクロにより生成された出力、イベントログメッセージ、トレースについての情報など、デバッガの出力が表示されます。この出力は、デバッグセッション中しか使用できません。デフォルトでは、このウィンドウは他のメッセージウィンドウとグループ化されて表示されます（*ARM 用 IDE プロジェクト管理およびビルドガイド*を参照）。

以下のフォーマットのいずれかの行をダブルクリックすると、対応するソースコードが［エディタ］ウィンドウに表示されます。

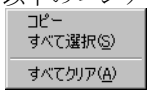
```
<path> (<row>):<message>
<path> (<row>,<column>):<message>
```

### 要件

ありません。このウィンドウは常に使用できます。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**コピー**

ウィンドウの内容をコピーします。

**すべて選択**

ウィンドウの内容を選択します。

**すべてをクリア**

ウィンドウの内容をクリアします。

# ［ログファイル］ダイアログボックス

［ログファイル］ダイアログボックスは、［デバッグ］>［ログ］>［ログファイルの設定］を選択すると使用できます。


このダイアログボックスを使用して、C-SPY からの出力をファイルに記録します。

### 要件

ありません。このダイアログボックスは常に使用できます。

### ログの有効化

ファイルへのログを有効 / 無効にします。

### 含める内容

ファイルに出力される情報は、デフォルトでは［ログ］ウィンドウに表示される内容と同一です。参照ボタンを使用して、デフォルトファイルとログファイルの場所をオーバライドします（デフォルトのファイル名拡張子は `log`）です。ログに記録する情報を変更するには、以下から選択します。

**エラー**

C-SPY が処理の実行に失敗したことを示します。

**ワーニング**

問題となるエラーや漏れ。

**情報**

C-SPY が実行したアクションの進行状況を示します。

**ユーザ**

C-SPY マクロからのメッセージ。つまり、`__message` 文を使用した自分のメッセージ。

## ［アサートの報告］ダイアログボックス

［アサートの報告］ダイアログボックスは、アプリケーションのソースコードに assert 関数の呼出しがあり、アサート条件が偽の場合に表示されます。このダイアログボックスで先の処理を選択できます。


**アサートメッセージをテキストとして出力するには、次の手順に従います。**

1 以下の関数をアプリケーションソースコードに追加します。

```
void __aeabi_assert(char const * msg, char const *file, int line)
{
   printf( "%s:%d %s -- assertion failed¥n", file, line, msg );
   abort();
}
```

2 アサートメッセージが表示されます。

### 中止

アプリケーションは実行を停止し、ターゲットシステム上のアプリケーションの一部であるランタイムライブラリ関数 abort が呼び出されます。つまり、アプリケーション自体が実行を終了します。

### デバッグ

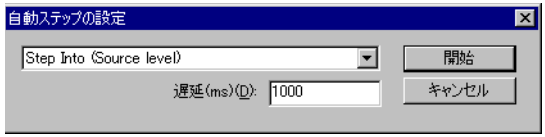
C-SPY がアプリケーションの実行を停止し、ユーザに制御の権限が戻ります。

### 無視

アサートが無視され、アプリケーションは実行を継続します。

## ［自動ステップの設定］ダイアログボックス

**［自動ステップの設定］**ダイアログボックスは、**［デバッグ］**メニューから表示します。




このダイアログボックスを使用して、自動ステップをカスタマイズします。

ドロップダウンメニューに、使用可能なステップコマンドが表示されます。

### 要件

ありません。このダイアログボックスは常に使用できます。

### 遅延

各ステップ間の遅延をミリ秒単位で指定します。

# 変数と式

- 変数と式の扱いの概要
- 変数と式の扱い
- 変数と式の扱いについてのリファレンス情報

## 変数と式の扱いの概要

以下のトピックを解説します。

- 変数と式の扱いの概要について
- C-SPY 式
- 変数情報の制限

### 変数と式の扱いの概要について

変数の値を参照、計算する方法はいろいろあります。

- ツールチップウォッチは、エディタウィンドウで表示され、変数や、より複雑な式の値を表示するための最も簡単な方法です。マウスポインタで変数を指すだけで、変数の横にその値が表示されます。
- ［自動］ウィンドウには、現在の文やその近くにある文の変数や式が自動的に表示されます。実行が停止すると、ウィンドウは自動的に更新されます。
- ［ローカル］ウィンドウには、ローカル変数、つまりアクティブな関数の自動変数と関数パラメータが表示されます。実行が停止すると、ウィンドウは自動的に更新されます。
- ［ウォッチ］ウィンドウを使用すると、C-SPY 式および変数の値をモニタできます。実行が停止すると、ウィンドウは自動的に更新されます。
- ［ライブウォッチ］ウィンドウは繰り返しサンプリングを行い、アプリケーションの実行中に式の値を表示します。式の変数は、グローバル変数のように、静的に特定できる必要があります。
- ［静的変数］ウィンドウには、静的記憶寿命を持つ変数の値が表示されます。実行が停止すると、ウィンドウは自動的に更新されます。
- ［マクロクイック起動］ウィンドウと［クイックウォッチ］ウィンドウでは、式を評価するタイミングを細かく制御できます。

- ［シンボル］ウィンドウには、ランタイムライブラリのシンボルを含め、静的な位置を持つすべてのシンボル（すなわち、C/C++ 関数、アセンブララベル、静的記憶寿命変数）が表示されます。
- ［データログ］ウィンドウと［データログ一覧］ウィンドウには、最大 4 つの異なるメモリ位置、またはデータログブレークポイントを設定して選択するエリアへのアクセスのログが表示されます。データログを使用すると、頻繁にアクセスするデータを容易に検索できます。データを特定したら、より効率的なメモリに配置するかどうかを検討できます。
- ［イベントログ］ウィンドウと［イベントログサマリ］ウィンドウには、実行がアプリケーションコードの特定位置を通過する際に生成される*イベントログ*が表示されます。［タイムライン］ウィンドウは、共通の時間軸に相関するこれらのイベントログをグラフィックを使用して表示します。イベントのログを使用すると、プログラムのフローを解析したり、アプリケーションコードの特定の場所に関係するデータを点検するときに役立ちます。

  Cortex ITM 通信チャンネルは、実行中のアプリケーションから C-SPY イベントログシステムにイベントを渡すときに使用されます。アプリケーションのソースコードで使用できる定義済のプリプロセッサマクロがあります。イベントログは、プログラムの実行時にこうしたマクロが渡されるたびに生成されます。各イベントとともに値を渡すことができます。この値は通常、変数またはレジスタの識別子か内容のどちらかです（スタックポインタなど）。値は 8 ビット、16 ビット、32 ビットのフォーマットで書き込むことができます。小さいサイズを使用すると、SWO ワイヤで必要な帯域幅が少なくなります。イベントは関連の PC（プログラムカウンタ）の値があるかどうかに関わらず生成できます。PC の値によって、デバッガでイベントを実行されたコードに関連付けることができます。
- トレース関連のウィンドウでは、プログラムの流れを特定の状態まで調べることができます。詳細については、217 ページの *トレース*を参照してください。

## C-SPY 式

C-SPY 式には、関数呼出しを除く任意の種類の C 言語の式を使用できます。また、以下に示すシンボルも使用できます。

- C/C++ シンボル
- アセンブラシンボル（レジスタ名、アセンブララベル）
- C-SPY マクロ関数
- C-SPY マクロ変数

これらのシンボルから構成された式を C-SPY 式と呼び、さまざまな方法で C-SPY 式をモニタできます。有効な C-SPY 式の例を以下に示します。

```
i + j
i = 42
myVar = cVar
cVar = myVar + 2
#asm_label
#R2
#PC
my_macro_func(19)
```

いくつかの異なる関数で同じ名前で宣言された静的変数がある場合、*function*::*variable* という記述を使用して、どの変数をモニタするかを指定します。

### C/C++ シンボル

C シンボルは、アプリケーションの C ソースコードで定義したシンボルです。たとえば、変数や定数、関数（関数はシンボルとして使用できますが、実行はできません）。C 言語のシンボルは、その名前で参照できます。C++ シンボルに、C-SPY シンボルや式で有効でない関数呼出しが暗黙的に含まれる場合があります。

**注 :** C/C++ で使用可能な一部の属性（たとえば `volatile`）は、C-SPY で完全にサポートされていません。たとえば、次の行は C-SPY で認められません。

```
sizeof(unsigned char volatile __mematтr *)
```

しかし、次の行は認められます。

```
sizeof(unsigned char __mematтr *)
```

### アセンブラシンボル

アセンブラシンボルは、アセンブララベルやレジスタ（たとえばプログラムカウンタやスタックポインタ、その他の CPU レジスタ）です。デバイス記述ファイルを使用する場合は、I/O ポートなど、メモリにマッピングされたすべての周辺ユニットも、CPU レジスタと同様にアセンブラシンボルとして使用できます。59 ページの*デバイス記述ファイルの修正*を参照してください。

アセンブラシンボルの頭に `#` が付いている場合は C-SPY 式で使用できます。

| 例 | 実行される内容 |
|---|---|
| `#PC++` | プログラムカウンタ値をインクリメント |
| `myVar = #SP` | スタックポインタレジスタの現在の値を C-SPY 変数に割り当てます。 |

*表 6: C-SPY アセンブラシンボル式*

| 例 | 実行される内容 |
|---|---|
| `myVar = #label` | `myVar` を `label` のアドレスにある整数値に設定します。 |
| `myptr = &#label7` | `myptr` を `label7` をポイントする `int *` ポインタに設定します。 |

*表 6: C-SPY アセンブラシンボル式*

ハードウェアレジスタとアセンブララベルの名前が衝突する場合、ハードウェアレジスタが優先的に選択されます。そのような場合にアセンブララベルを参照するには、ラベルをバッククォート`（ASCII 文字 0x60）で囲む必要があります。次に例を示します。

| 例 | 実行される内容 |
|---|---|
| `#PC` | プログラムカウンタを参照 |
| `#`PC`` | アセンブララベル `PC` を参照 |

*表 7: ハードウェアレジスタとアセンブララベルの名前が衝突する場合の処理*

［レジスタ］ウィンドウには、デフォルトで使用できるプロセッサ固有のシンボルが、`CPU` レジスタレジスタグループに基づいて表示されます。193 ページの *［レジスタ］ウィンドウ*を参照してください。

## C-SPY マクロ関数

マクロ関数は、C-SPY マクロ変数定義と、マクロが呼び出されたときに実行されるマクロ文から構成されます。

C-SPY マクロ関数とその使用方法の詳細については、395 ページの *マクロ言語の概要*を参照してください。

## C-SPY マクロ変数

マクロ変数の定義と配置はアプリケーションの外部で行われ、C-SPY 式で使用できます。C 言語のシンボルと C-SPY マクロ変数の名前が衝突する場合、C-SPY マクロ変数が C 言語の変数よりも優先的に選択されます。マクロ変数に代入すると、その値と型の両方に代入が行われます。

C-SPY マクロ変数とその使用方法の詳細については、401 ページの *マクロ言語についてのリファレンス情報*を参照してください。

## sizeof の使用

C 規格に準じて、2 つの `sizeof` の構文形式をとります。

```
sizeof（タイプ）
sizeof expr
```

前者がタイプ用で、後者が式用です。

**注 :** C-SPY では、`sizeof` 演算子の使用時に式を括弧で囲いません。たとえば、`sizeof (x+2)` ではなく `sizeof x+2` を使用します。

### 変数情報の制限

C 言語の変数の値は、ステップポイント、すなわち文の先頭の命令と関数呼出しの位置でのみ有効です。その場合、エディタウィンドウで淡い緑色で強調表示されます。実際には、変数の値はそれ以外の位置でもアクセスでき、正しい値を示すことがほとんどです。

プログラムカウンタが文の内部でステップポイント以外の位置を指している場合は、その文または文の一部は通常の強調表示色よりも薄い色で強調表示されます。

### 最適化の影響

コンパイラは、予想される動作が実行できる範囲内であれば、可能な限り自由にアプリケーションソフトウェアを最適化します。最適化によって、コードが影響を受けます。生成されたコードがソースコードにどう関連するかが明確でなくなり、デバッグがより困難になるためです。通常は、高いレベルで最適化を行うと、コードに変更が生じ、予想と違って、変数の値を参照できなくなります。

1 つ例を示します。

```
myFunction()
{
 int i = 42;
 ...
 x = computer(i); /* ここで、i の値は C-SPY に認識されています */
 ...
}
```

変数 `i` が宣言されてから実際に使用されるまで、コンパイラはその変数用のスペースをスタックやレジスタに確保する必要はありません。コンパイラはこのコードを最適化できますが、C-SPY はその変数が実際に使用されるまでその値を表示できなくなります。一時的に使用できない状態にある変数の値を表示しようとすると、C-SPY は以下のメッセージを表示します。

```
使用不可能
```

デバッグセッションで変数の値を常に把握する必要がある場合は、コンパイル時に一番低い最適化レベル、**[なし]** を指定する必要があります。

# 変数と式の扱い

以下のタスクについて解説します。

- 変数と式に関連するウィンドウの使用
- アセンブラ変数の表示
- データログを開始するには
- イベントログを開始するには

## 変数と式に関連するウィンドウの使用

該当する場合、式の追加や修正、削除を行ったり、変数や式に関連するウィンドウの表示フォーマットを変更できます。

値を追加するには、点線の長方形をクリックして、調べる式を入力します。式の値を変更するには、**[値]** フィールドをクリックして、その内容を変更します。式を削除するには、式を選択して、Delete キーを押します。

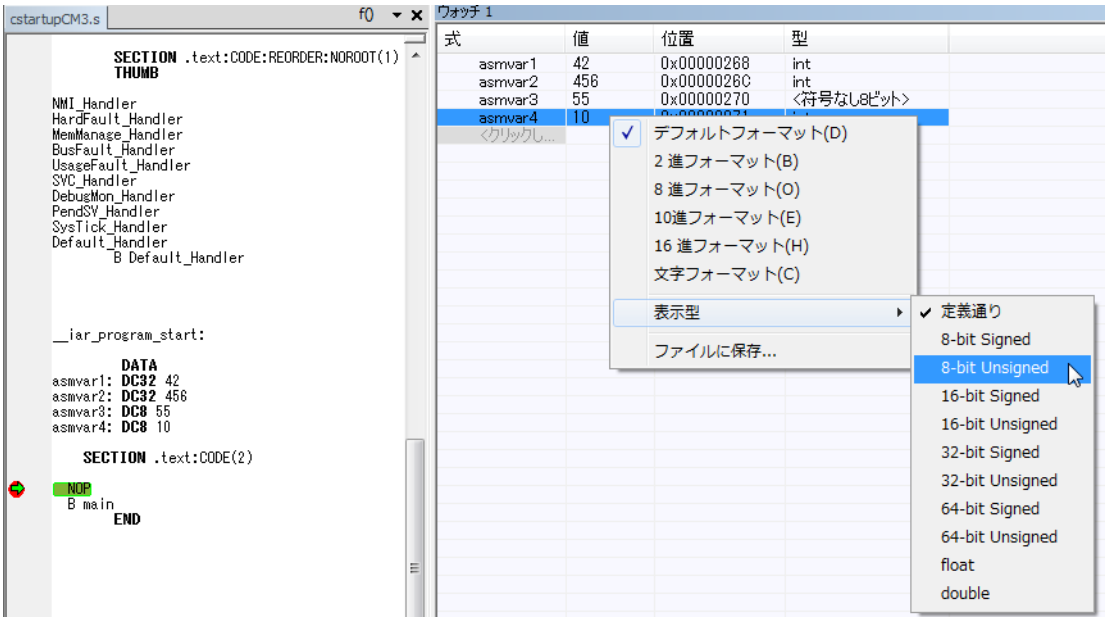
列の長さ以上の長さのテキストは、[トレース] ウィンドウ以外のこれらのすべてのウィンドウで切り詰められており、該当するテキストにマウスポインタを置くと、ツールチップ情報が表示されます。

ウィンドウのいずれかで右クリックして、追加のコマンドを含むコンテキストメニューにアクセスします。[ローカル] ウィンドウとデータログのウィンドウ、[クイックウォッチ] ウィンドウ以外のウィンドウでは、ウィンドウ間のドラッグアンドドロップもサポートされています。

## アセンブラ変数の表示

アセンブララベルは、型に関する情報を何も持たないため、C-SPY はそれ以外の情報が与えられなければ、そのラベルの位置にあるデータを簡単に表示することはできません。データを簡単に表示する方法として、デフォルトでは、C-SPY はアセンブララベルの位置にあるすべてのデータを `int` 型の変数として処理します。ただし、[ウォッチ]、[ライブウォッチ]、[クイックウォッチ] の各ウィンドウでは、変数宣言に合せて別の解釈を選択することができます。

以下に示す図では、4 つの変数が［ウォッチ］ウィンドウに表示され、対応する宣言が左側のアセンブラソースファイルで示されています。




asmvar4 は、元のアセンブラ宣言では 1 バイトデータとして意図していたのに対し、［ウォッチ］ウィンドウでは int 型として表示されていることに注意してください。コンテキストメニューを使用すると、たとえば 8 ビットの符号なし変数として表示するように指定できます。asmvar3 変数は、すでにそのように指定されています。

## データログを開始するには

1 データログを設定するには、*［C-SPY ドライバ］*>［**SWO 設定**］を選択します。ダイアログボックスで、トレースデータの SWO 通信チャンネルを設定します。特に［**CPU クロック**］オプションに注意してください。CPU クロックのデフォルト値は、［**プロジェクト**］>［**オプション**］>*［C-SPY ドライバ］*ページで設定することができます。［**SWO 設定**］ダイアログボックスでは、デフォルト値をオーバライドできます。

C-SPY シミュレータを使用する場合は、この手順は無視してかまいません。

2 ［ブレークポイント］または［メモリ］ウィンドウで右クリックし、**［新規ブレークポイント］> ［データログ］**を選択して、［ブレークポイント］ダイアログボックスを開きます。ログ情報を収集するデータにデータログブレークポイントを設定します。最大 4 つのデータログブレークポイントを設定できます。

3 *［C-SPY ドライバ］*> **［データログ］**を選択して、［データログ］ウィンドウを開きます。または、以下のように選択することもできます。

- *［C-SPY ドライバ］*> **［データログ一覧］**を選択して、［データログ一覧］ウィンドウを開く。
- *［C-SPY ドライバ］*> **［タイムライン］**を選択して［タイムライン］ウィンドウを開き、データロググラフを表示する。

4 ［データログ］ウィンドウのコンテキストメニューから、**［有効化］**を選択してロギングを有効にします。

5 **［SWO 設定］**ダイアログボックスの［データログイベント］エリアで、データログが有効になっていることが確認できます。必要なロギングのレベルを以下から選択します。

- PC のみ
- PC + データ値 + ベースアドレス
- データ値 + 正確なアドレス

C-SPY シミュレータを使用する場合は、この手順は無視してかまいません。

6 アプリケーションプログラムの実行を開始して、ログ情報を収集します。

7 データログ情報を表示するには、［データログ］、［データログ一覧］、［タイムライン］ウィンドウのデータグラフを参照します。

8 ログまたは概要をファイルに保存する場合は、対象のウィンドウのコンテキストメニューから**［ログファイルを保存］**を選択します。

9 データおよび割込みログを無効にするには、有効になっている対象のウィンドウのコンテキストメニューで**［無効］**を選択します。

### イベントログを開始するには

1 イベントを生成するアプリケーションのソースコードの位置を指定するには、arm_itm.h（arm¥inc¥c 内）にある定義済のプリプロセッサマクロを使用します。アプリケーションのソースコードで、たとえば次のように記述します。

```
#include <arm_itm.h>
void func(void)
{
  ITM_EVENT8_WITH_PC(1,25);
  ITM_EVENT32_WITH_PC(2, __get_PSP());
}
```

最初の行は値が 25 のイベントをチャネル 1 に送信し、2 行目はスタックポインタの現在の値を持つイベントをチャネル 2 に送信します。つまり、C-SPY はスタックポインタを指定のコード位置で表示することができます。プログラム実行時にこれらのソース行が渡されると、C-SPY によりイベントが生成されて視覚化され、それらを詳細に解析できるようになります

2 イベント情報を表示するには、次のいずれかの方法を選択します。

- *[C-SPY ドライバ]* >［タイムライン］を選択して［タイムライン］ウィンドウを開き、コンテキストメニューから［有効化］を選択します。各チャネルのイベントをグラフとして表示できます（イベントグラフ）。
- *[C-SPY ドライバ]* >［イベントログ］を選択して［イベントログ］ウィンドウを開き、コンテキストメニューから［有効化］を選択します。各チャネルのイベントを数値として表示できるようになります。
- *[C-SPY ドライバ]* >［イベントログサマリ］を選択して［イベントログサマリ］ウィンドウを開き、コンテキストメニューから［有効化］を選択します。すべてのイベントのサマリが入手できます。

**注 :** イベントグラフや［イベントログ］ウィンドウが有効になっている場合は、いつでも［イベントログサマリ］ウィンドウを有効にしてサマリを入手することができます。ただし、［イベントログサマリ］ウィンドウを有効にして［イベントログ］ウィンドウまたは［タイムライン］ウィンドウでイベントグラフを有効にしない場合、サマリは入手できますが、イベントの詳細な情報は得られません。

3 表示型を変更する場合（値を 16 進数または 10 進数で表示できます）、型を変更するイベントグラフを［タイムライン］ウィンドウで選択します。右クリックして、コンテキストメニューから表示フォーマットを選択します。この設定は［イベントログ］ウィンドウと［イベントログサマリ］ウィンドウにも影響する点に注意してください。

4 アプリケーションプログラムの実行を開始して、ログ情報を収集します。

5 イベント情報を見るには、イベントログかイベントログサマリ、［タイムライン］ウィンドウの特定のチャネルのイベントグラフを参照してください。

6 ログまたは概要をファイルに保存する場合は、ウィンドウのコンテキストメニューから**［ログファイルを保存］**を選択します。

7 イベントログを無効にするには、有効になっている対象のウィンドウのコンテキストメニューから**［無効］**を選択します。

# 変数と式の扱いについてのリファレンス情報

リファレンス情報 :

- 109 ページの*［自動］ウィンドウ*
- 110 ページの*［ローカル］ウィンドウ*
- 112 ページの*［ウォッチ］ウィンドウ*
- 114 ページの*［ライブウォッチ］ウィンドウ*
- 117 ページの*［静的変数］ウィンドウ*
- 120 ページの*［クイックウォッチ］ウィンドウ*
- 122 ページの*［シンボル］ウィンドウ*
- 124 ページの*［シンボルの曖昧さの解決］ダイアログボックス*
- 125 ページの*［データログ］ウィンドウ*
- 127 ページの*［データログ一覧］ウィンドウ*
- 129 ページの*［イベントログ］ウィンドウ*
- 131 ページの*［イベントログサマリ］ウィンドウ*

参照 :

- 225 ページの*トレースについてのリファレンス情報*（トレース関連のリファレンス情報）
- 466 ページの*［マクロクイック起動ウィンドウ］*

# ［自動］ウィンドウ

［自動］ウィンドウは［**表示**］メニューから利用できます。


このウィンドウには、現在の文やその近くにある文の変数や式が自動的に表示されます。C-SPY で実行が停止するたびに、［自動］ウィンドウの値が再計算されます。前回停止した後に変更になった値は赤色で強調表示されます。

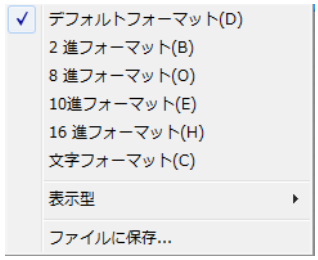
C-SPY のウィンドウの編集について詳しくは、65 ページの *C-SPY デバッガメインウィンドウ*を参照してください。

## 要件

ありません。このウィンドウは常に使用できます。

## コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**デフォルトフォーマット**
**2 進フォーマット**
**8 進フォーマット**
**10 進フォーマット**
**16 進フォーマット**
**文字フォーマット**

式の表示フォーマットを変更します。表示フォーマット設定は、式の種類によって適用対象が異なります。表示フォーマットの選択は、デバッグセッションの終了後も保持されます。ウィンドウで選択された行に変数が含まれる場合、これらのコマンドが使用できます。

表示フォーマット設定は、式の種類によって以下のように適用対象が異なります。

| | |
|---|---|
| 変数 | 表示設定は、選択した変数だけに適用されます。他の変数には適用されません。 |
| 配列エレメント | 表示設定は配列全体に適用されます。つまり、配列の各エレメントに同一の表示フォーマットが使用されます。 |
| 構造体のフィールド | 定義が同一のエレメント（フィールド名、C の宣言型）に表示設定が適用されます。 |

**表示フォーマット**

変数のデフォルトの型解釈を変更するコマンドをサブメニューで表示します。このサブメニューのコマンドは、デフォルトで整数として表示されるアセンブラの変数（アセンブララベルでのデータ）に主に使用します。詳細については、104 ページの *アセンブラ変数の表示*を参照してください。

**オプション**

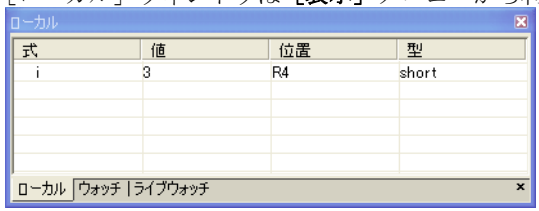
**［IDE オプション］**ダイアログボックスを表示します。ここでは**［更新間隔］**オプションを設定できます。このオプションのデフォルト値は 1000 ミリ秒です。つまり、プログラム実行中に、**［ライブウォッチ］**ウィンドウが 1 秒に 1 回更新されます。このコマンドは、**［ライブウォッチ］**ウィンドウのこのコンテキストメニューでのみ使用できる点に注意してください。

**ファイルに保存**

内容をタブ区切り形式でファイルに保存します。

## ［ローカル］ウィンドウ

［ローカル］ウィンドウは**［表示］**メニューから利用できます。




このウィンドウには、現在の関数のローカル変数およびパラメータが表示されます。C-SPY で実行が停止するたびに、［ローカル］ウィンドウの値が再計算されます。前回停止した後に変更になった値は赤色で強調表示されます。

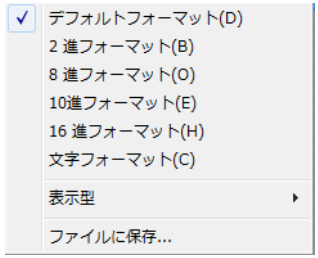
C-SPY のウィンドウの編集について詳しくは、65 ページの *C-SPY デバッガメインウィンドウ*を参照してください。

### 要件

ありません。このウィンドウは常に使用できます。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**デフォルトフォーマット**
**2 進フォーマット**
**8 進フォーマット**
**10 進フォーマット**
**16 進フォーマット**
**文字フォーマット**

式の表示フォーマットを変更します。表示フォーマット設定は、式の種類によって適用対象が異なります。表示フォーマットの選択は、デバッグセッションの終了後も保持されます。ウィンドウで選択された行に変数が含まれる場合、これらのコマンドが使用できます。

表示フォーマット設定は、式の種類によって以下のように適用対象が異なります。

| | |
|---|---|
| 変数 | 表示設定は、選択した変数だけに適用されます。他の変数には適用されません。 |
| 配列エレメント | 表示設定は配列全体に適用されます。つまり、配列の各エレメントに同一の表示フォーマットが使用されます。 |
| 構造体のフィールド | 定義が同一のエレメント（フィールド名、C の宣言型）に表示設定が適用されます。 |

**表示フォーマット**

変数のデフォルトの型解釈を変更するコマンドをサブメニューで表示します。このサブメニューのコマンドは、デフォルトで整数として表示されるアセンブラの変数（アセンブララベルでのデータ）に主に使用します。詳細については、104 ページの *アセンブラ変数の表示*を参照してください。

**オプション**

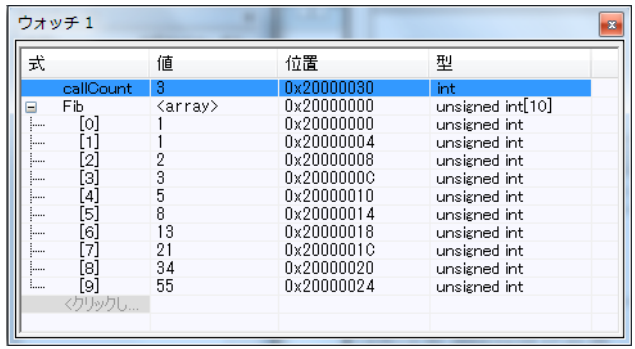
**［IDE オプション］**ダイアログボックスを表示します。ここでは**［更新間隔］**オプションを設定できます。このオプションのデフォルト値は 1000 ミリ秒です。つまり、プログラム実行中に、**［ライブウォッチ］**ウィンドウが 1 秒に 1 回更新されます。このコマンドは、**［ライブウォッチ］**ウィンドウのこのコンテキストメニューでのみ使用できる点に注意してください。

**ファイルに保存**

内容をタブ区切り形式でファイルに保存します。

## ［ウォッチ］ウィンドウ

［ウォッチ］ウィンドウは**［表示］**メニューから利用できます。




このウィンドウを使用して、C-SPY の式や変数の値をモニタします。このウィンドウは最大 4 つまで開くことができ、式の表示や追加、変更、削除を行えます。配列、構造体、共用体は展開可能です。つまり、各エレメントの値をモニタすることができます。

C-SPY で実行が停止するたびに、［ウォッチ］ウィンドウの値が再計算されます。前回停止した後に変更になった値は赤色で強調表示されます。

非常に大きな配列を展開すると、メモリ不足でクラッシュすることがあるため注意してください。これを防止するために、拡張は自動的に 5000 エレメントのステップで実行されます。

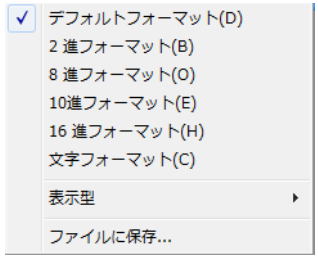
C-SPY のウィンドウの編集について詳しくは、65 ページの *C-SPY デバッガメインウィンドウ*を参照してください。

### 要件

ありません。このウィンドウは常に使用できます。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**デフォルトフォーマット**
**2 進フォーマット**
**8 進フォーマット**
**10 進フォーマット**
**16 進フォーマット**
**文字フォーマット**

式の表示フォーマットを変更します。表示フォーマット設定は、式の種類によって適用対象が異なります。表示フォーマットの選択は、デバッグセッションの終了後も保持されます。ウィンドウで選択された行に変数が含まれる場合、これらのコマンドが使用できます。

表示フォーマット設定は、式の種類によって以下のように適用対象が異なります。

| | |
|---|---|
| 変数 | 表示設定は、選択した変数だけに適用されます。他の変数には適用されません。 |
| 配列エレメント | 表示設定は配列全体に適用されます。つまり、配列の各エレメントに同一の表示フォーマットが使用されます。 |
| 構造体のフィールド | 定義が同一のエレメント（フィールド名、C の宣言型）に表示設定が適用されます。 |

**表示フォーマット**

変数のデフォルトの型解釈を変更するコマンドをサブメニューで表示します。このサブメニューのコマンドは、デフォルトで整数として表示されるアセンブラの変数（アセンブララベルでのデータ）に主に使用します。詳細については、104 ページの *アセンブラ変数の表示*を参照してください。

**オプション**

**［IDE オプション］**ダイアログボックスを表示します。ここでは**［更新間隔］**オプションを設定できます。このオプションのデフォルト値は 1000 ミリ秒です。つまり、プログラム実行中に、**［ライブウォッチ］**ウィンドウが 1 秒に 1 回更新されます。このコマンドは、**［ライブウォッチ］**ウィンドウのこのコンテキストメニューでのみ使用できる点に注意してください。

**ファイルに保存**

内容をタブ区切り形式でファイルに保存します。

## ［ライブウォッチ］ウィンドウ

［ライブウォッチ］ウィンドウは**［表示］**メニューから利用できます。


このウィンドウは繰り返しサンプリングを行い、アプリケーションの実行中に式の値を表示します。式の変数は、グローバル変数のように、静的に特定できる必要があります。

以下の場合にライブウォッチを使用します。

| デバイス | |
|---|---|
| Cortex-M | メモリへのアクセスやブレークポイントは、実行中常に設定可能です。 |
| ARMxxx-S | ハードウェアブレークポイントは、実行中常に設定可能です。 |

*表 8: 異なるデバイスのライブウォッチ*

| デバイス | |
|---|---|
| ARM7/ARM9（ARMxxx-S を含む）および C-SPY J-Link/J-Trace ドライバを使用する場合 | アプリケーションからメモリへのアクセスが発生します。DCC ユニット経由でデバッガと通信する小さなプログラム（DCC ハンドラ）をアプリケーションに追加すると、実行中にメモリが読み取り / 書き込み可能になります。ソフトウェアブレークポイントも、DCC ハンドラによって設定できます。<br>arm¥src¥debugger¥dcc にあるファイル JLINKDCC_Process.c と JLINKDCC_HandleDataAbort.s をプロジェクトに追加して、JLINKDCC_Process 関数をミリ秒ごとなど定期的に呼び出します。<br>cstartup ファイルのローカルコピーで、データ異常終了によって JLINKDCC_HandleDataAbort ハンドラが呼び出されるように割込みベクタテーブルを変更します。506 ページの *–jlink_dcc_timeout* を参照してください。 |

*表 8: 異なるデバイスのライブウォッチ*

C-SPY のウィンドウの編集について詳しくは、65 ページの *C-SPY デバッガメインウィンドウ*を参照してください。

## 要件

サポートされているすべてのハードウェアデバッガシステム。

## 表示エリア

このエリアには以下の列が含まれます。

### 式

変数名。変数のベース名に続いて、モジュール、クラス、または関数スコープを含むフルネームが表示されます。この列は編集できません。

### 値

変数の値。変更された値は赤色で強調表示されます。

テキストや変数を別のウィンドウからドラッグして、[値] の列にドロップすると、新しい値がその行の変数に割り当てられます。

この列は編集できます。

### 位置

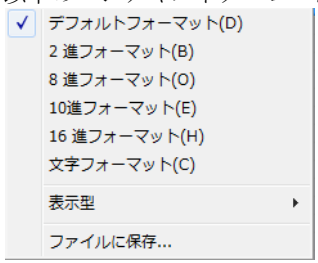
この変数が格納されているメモリの場所。

### タイプ

変数のデータ型。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**デフォルトフォーマット**
**2 進フォーマット**
**8 進フォーマット**
**10 進フォーマット**
**16 進フォーマット**
**文字フォーマット**

式の表示フォーマットを変更します。表示フォーマット設定は、式の種類によって適用対象が異なります。表示フォーマットの選択は、デバッグセッションの終了後も保持されます。ウィンドウで選択された行に変数が含まれる場合、これらのコマンドが使用できます。

表示フォーマット設定は、式の種類によって以下のように適用対象が異なります。

| | |
|---|---|
| 変数 | 表示設定は、選択した変数だけに適用されます。他の変数には適用されません。 |
| 配列エレメント | 表示設定は配列全体に適用されます。つまり、配列の各エレメントに同一の表示フォーマットが使用されます。 |
| 構造体のフィールド | 定義が同一のエレメント（フィールド名、C の宣言型）に表示設定が適用されます。 |

**表示フォーマット**

変数のデフォルトの型解釈を変更するコマンドをサブメニューで表示します。このサブメニューのコマンドは、デフォルトで整数として表示されるアセンブラの変数（アセンブララベルでのデータ）に主に使用します。詳細については、104 ページの *アセンブラ変数の表示*を参照してください。

**オプション**

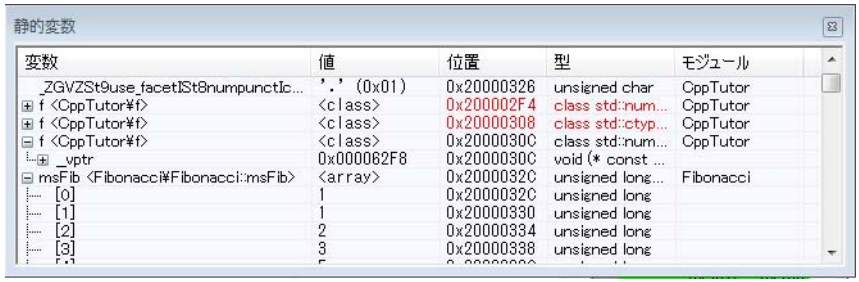
［**IDE オプション**］ダイアログボックスを表示します。ここでは［**更新間隔**］オプションを設定できます。このオプションのデフォルト値は 1000 ミリ秒です。つまり、プログラム実行中に、［**ライブウォッチ**］ウィンドウが 1 秒に 1 回更新されます。このコマンドは、［**ライブウォッチ**］ウィンドウのこのコンテキストメニューでのみ使用できる点に注意してください。

**ファイルに保存**

内容をタブ区切り形式でファイルに保存します。

## ［静的変数］ウィンドウ

［静的変数］ウィンドウは［**表示**］メニューから利用できます。




このウィンドウには、選択した変数の値および静的記憶寿命が表示されます。通常はファイルスコープを持つ変数ですが、関数やクラス内の静的変数ということもあります。`volatile` として宣言された静的記憶寿命変数は表示されないことに注意してください。

C-SPY で実行が停止するたびに、［静的変数］ウィンドウの値が再計算されます。前回停止した後に変更になった値は赤色で強調表示されます。

列をソートするには、その列の見出しをクリックします（［**値**］を除く）。

C-SPY のウィンドウの編集について詳しくは、65 ページの *C-SPY デバッガメインウィンドウ*を参照してください。

**モニタする変数を選択するには、以下の手順に従います。**

1 ウィンドウで右クリックし、コンテキストメニューで［**静的変数を選択**］を選択します。このウィンドウには、静的記憶寿命を持つすべての変数が一覧表示されるようになります。

2 表示する変数を個別に選択するか、コンテキストメニューから［**選択**］コマンドのいずれかを選びます。

3 選択が終わったら、コンテキストメニューから［**静的変数を選択**］を選択して通常の表示モードに切り替えます。

### 要件

ありません。このウィンドウは常に使用できます。

### 表示エリア

このエリアには以下の列が含まれます。

**式**

変数名。変数のベース名に続いて、モジュール、クラス、または関数スコープを含むフルネームが表示されます。この列は編集できません。

**値**

変数の値。変更された値は赤色で強調表示されます。

テキストや変数を別のウィンドウからドラッグして、［**値**］の列にドロップすると、新しい値がその行の変数に割り当てられます。

この列は編集できます。

**位置**

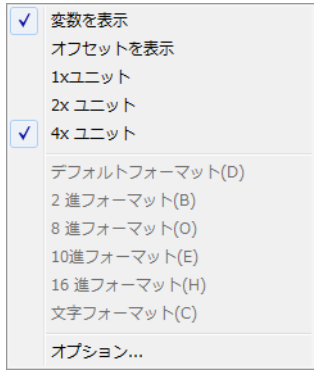
この変数が格納されているメモリの場所。

**タイプ**

変数のデータ型。

### コンテキストメニュー

以下のコンテキストメニューがあります。



以下のコマンドがあります。

**デフォルトフォーマット**
**2 進フォーマット**
**8 進フォーマット**
**10 進フォーマット**
**16 進フォーマット**
**文字フォーマット**

式の表示フォーマットを変更します。表示フォーマット設定は、式の種類によって適用対象が異なります。表示フォーマットの選択は、デバッグセッションの終了後も保持されます。ウィンドウで選択された行に変数が含まれる場合、これらのコマンドが使用できます。

表示フォーマット設定は、式の種類によって以下のように適用対象が異なります。

| | |
|---|---|
| 変数 | 表示設定は、選択した変数だけに適用されます。他の変数には適用されません。 |
| 配列エレメント | 表示設定は配列全体に適用されます。つまり、配列の各エレメントに同一の表示フォーマットが使用されます。 |
| 構造体のフィールド | 定義が同一のエレメント（フィールド名、C の宣言型）に表示設定が適用されます。 |

**ファイルに保存**

［**静的**］ウィンドウの内容をログファイルに保存します。

**静的変数を選択**

静的記憶寿命変数をすべて選択 : このコマンドは以下の［**選択**］コマンドをすべて有効にします。モニタする変数を選択します。選択が終わったら、このメニューコマンドをもう一度選択して通常の表示モードに戻ります。

**すべて選択**

すべての変数を選択します。

**何も選択しない**

すべての変数を選択解除します。

***モジュール*内のすべてを選択**

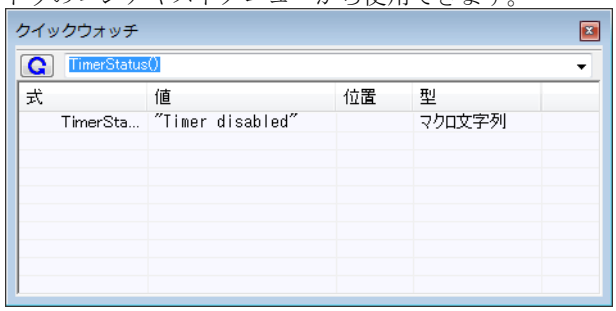
選択したモジュール内のすべての変数を選択します。

***モジュール*内で何も選択しない**

選択したモジュール内のすべての変数を選択解除します。

## ［クイックウォッチ］ウィンドウ

［クイックウォッチ］ウィンドウは、**［表示］**メニューおよびエディタウィンドウのコンテキストメニューから使用できます。




このウィンドウを使用して、変数や式の値を監視するほか、特定の時点で式を評価します。

［ウォッチ］ウィンドウと違って、［クイックウォッチ］ウィンドウでは式を評価するタイミングを細かく制御できます。代入や C-SPY マクロ関数などアクションのある式の場合は、条件を制御しながら評価することができます（単一変数では必要ない場合が多い）。

C-SPY のウィンドウの編集について詳しくは、65 ページの *C-SPY デバッガメインウィンドウ*を参照してください。

**式を評価するには、以下の手順に従います。**

1 エディタウィンドウで評価式を右クリックして、表示されるコンテキストメニューで［クイックウォッチ］を選択します。

2 ［クイックウォッチ］ウィンドウに式が自動的に表示されます。

別の方法は、

3 ［クイックウォッチ］ウィンドウで、確認するファイル名を**［式］**テキストボックスに入力します。


4 **［再計算］**ボタンをクリックすると、式の値が計算されます。

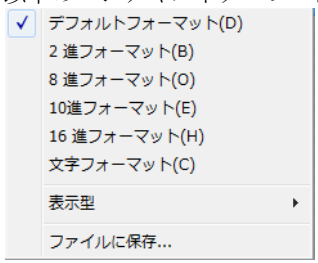
例については、396 ページの *C-SPY マクロの使用*を参照してください。

### 要件

ありません。このウィンドウは常に使用できます。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**デフォルトフォーマット**
**2 進フォーマット**
**8 進フォーマット**
**10 進フォーマット**
**16 進フォーマット**
**文字フォーマット**

式の表示フォーマットを変更します。表示フォーマット設定は、式の種類によって適用対象が異なります。表示フォーマットの選択は、デバッグセッションの終了後も保持されます。ウィンドウで選択された行に変数が含まれる場合、これらのコマンドが使用できます。

表示フォーマット設定は、式の種類によって以下のように適用対象が異なります。

| | |
|---|---|
| 変数 | 表示設定は、選択した変数だけに適用されます。他の変数には適用されません。 |
| 配列エレメント | 表示設定は配列全体に適用されます。つまり、配列の各エレメントに同一の表示フォーマットが使用されます。 |
| 構造体のフィールド | 定義が同一のエレメント（フィールド名、C の宣言型）に表示設定が適用されます。 |

**表示フォーマット**

変数のデフォルトの型解釈を変更するコマンドをサブメニューで表示します。このサブメニューのコマンドは、デフォルトで整数として表示されるアセンブラの変数（アセンブララベルでのデータ）に主に使用します。詳細については、104 ページの *アセンブラ変数の表示*を参照してください。

**オプション**

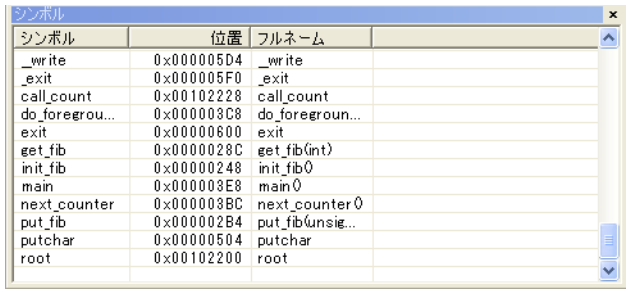
［**IDE オプション**］ダイアログボックスを表示します。ここでは［**更新間隔**］オプションを設定できます。このオプションのデフォルト値は 1000 ミリ秒です。つまり、プログラム実行中に、［**ライブウォッチ**］ウィンドウが 1 秒に 1 回更新されます。このコマンドは、［**ライブウォッチ**］ウィンドウのこのコンテキストメニューでのみ使用できる点に注意してください。

**ファイルに保存**

内容をタブ区切り形式でファイルに保存します。

## ［シンボル］ウィンドウ

［シンボル］ウィンドウは、シンボルプラグインモジュールを有効にした後、［**表示**］メニューから使用できます。




このウィンドウには、ランタイムライブラリのシンボルを含め、静的な位置を持つすべてのシンボル（すなわち、C/C++ 関数、アセンブララベル、静的記憶寿命変数）が表示されます。

シンボルプラグインモジュールを有効にするには、［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**ロードするプラグインの選択**］>［**シンボル**］を選択します。

### 要件

ありません。このウィンドウは常に使用できます。

### 表示エリア

このエリアには以下の列が含まれます。

**シンボル**

シンボル名。

**位置**

メモリアドレス。

**フルネーム**

シンボル名。通常は［シンボル］列の内容と同じですが、C++ メンバ関数などでは異なります。

列の見出しをクリックすると、リストがシンボル名、位置、またはフルネームによってソートされます。

## コンテキストメニュー

以下のコンテキストメニューがあります。


以下のコマンドがあります。

**関数**

リスト内の関数シンボルの表示を切り替えます。

**変数**

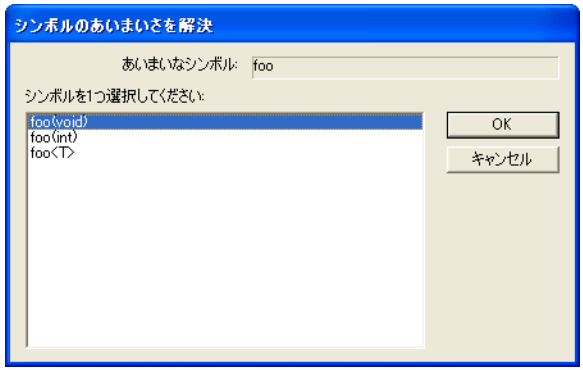
リスト内の変数の表示を切り替えます。

**ラベル**

リスト内のラベルの表示を切り替えます。

## ［シンボルの曖昧さの解決］ダイアログボックス

［シンボルの曖昧さの解決］ダイアログボックスは、たとえば［逆アセンブリ］ウィンドウで移動先のシンボルを指定して、テンプレートや関数のオーバロードのために同じシンボルが複数ある場合などに表示されます。




### 要件

ありません。このウィンドウは常に使用できます。

### 曖昧なシンボル

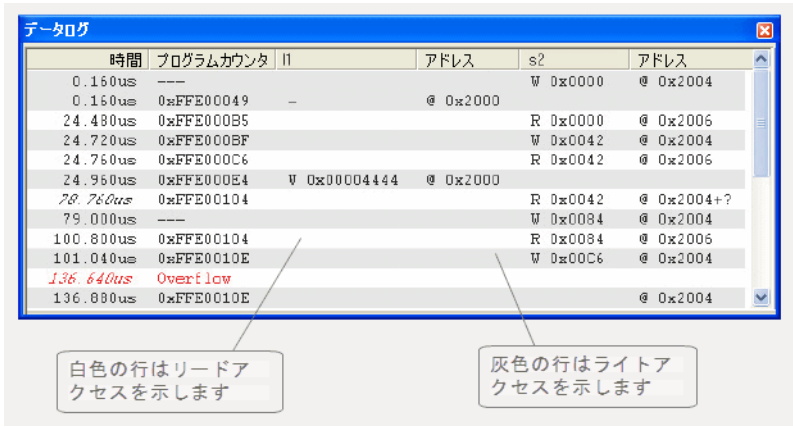
曖昧なシンボルを指定します。

### シンボルを 1 つ選択してください

曖昧なシンボルに一致する項目の一覧。使用するものを 1 つ選択します。

# ［データログ］ウィンドウ

［データログ］ウィンドウは、C-SPY ドライバメニューから使用できます。




このウィンドウを使用して、最大 4 つの異なるメモリ位置またはエリアへのアクセスを記録します。

105 ページの*データログを開始するには*も参照してください。

## 要件

Cortex-M デバイスと以下の以下のいずれか 1 つが必要です。

- C-SPY シミュレータ
- C-SPY I-jet/JTAGjet ドライバ、およびデバッグプローブとターゲットシステム間の SWD インタフェースを持つ I-jet または I-jet Trace インサーキットデバッグプローブ
- C-SPY J-Link/J-Trace ドライバ、およびデバッグプローブとターゲットシステム間の SWD インタフェースを持つ J-Link または J-Trace デバッグプローブ

  J-Trace の場合、このウィンドウは ETM トレースが無効なときに使用可能です。デバッグの際に ETM が有効な場合、このウィンドウには限られた量の収集されたデータのみが表示されます。実行が停止すると、トレースデータ全体が表示されます。
- デバッグプローブとターゲットシステム間の SWD インタフェースを持つ C-SPY ST-LINK ドライバと ST-LINK デバッグプローブ

### 表示エリア

表示エリアの各行には、時刻、プログラムカウンタが表示されます。また、追跡されるデータオブジェクトごとに、その値とアドレスがこれらの列に表示されます。

**時間**

I-jet インサーキットデバッグプローブについては、データアクセスの時間は専用の 48MHz クロックに基づきます。

クロック周波数に基づいた、C-SPY J-Link ドライバ、C-SPY ST-LINK ドライバ、シミュレータのデータアクセス時間。C-SPY J-Link ドライバと C-SPY ST-LINK ドライバの場合、これは［**SWO 設定**］ダイアログボックスで指定します。

ターゲットシステムが正確な時間を収集できなかった場合は、おおよその時刻が斜体で表示されます。

この列は、コンテキストメニューから［**時間表示**］を選択した場合に有効になります。

**サイクル**

実行の開始からイベントまでのサイクルの数。この情報は、リセットでクリアされます。

ターゲットシステムが正確な数を収集できなかった場合は、おおよその値が斜体で表示されます。

この列は、コンテキストメニューから［**サイクル表示**］を選択した場合に有効になります。

**プログラムカウンタ ***

以下のいずれかが表示されます。

`PC` の内容です。メモリアクセスを実行した命令のアドレスです。

`---`、ターゲットシステムがデバッガに情報を提供できなかったことを示します。

赤色で `Overflow` と表示されている場合、通信チャンネルがすべてのデータをターゲットシステムから送信できなかったことを示します。

***値***

アクセスタイプと、アクセスの記録対象の位置またはエリアの値（アクセスサイズを使用）を表示します。たとえば、バイトアクセスでゼロがリードされると、`0x00` と表示され、ロングアクセスの場合は `0x00000000` と表示されます。

アクセスの記録対象のデータを指定するには、**［データログ］**ブレークポイントダイアログボックスを使用します。135 ページの*データログブレークポイント*を参照してください。

**アドレス**

アクセスされた実際のメモリアドレス。たとえば、1 つのワードの 1 バイトだけアクセスされた場合は、そのバイトのアドレスだけが表示されます。アドレスは、ベースアドレス + オフセットによって算出されます。このベースアドレスは、**［データログ］**ブレークポイントダイアログボックスから取得され、オフセットはログから取得されます。ターゲットシステムからのログによってデバッガにオフセットが提供されなかった場合、オフセットには + ? が含まれます。オフセットを表示するには（C-SPY I-jet/JTAGjet ドライバ、C-SPY J-Link ドライバ、C-SPY ST-LINK ドライバの場合）、**［SWO 設定］**ダイアログボックスで**［値 + 正確なアドレス］**オプションを選択します。

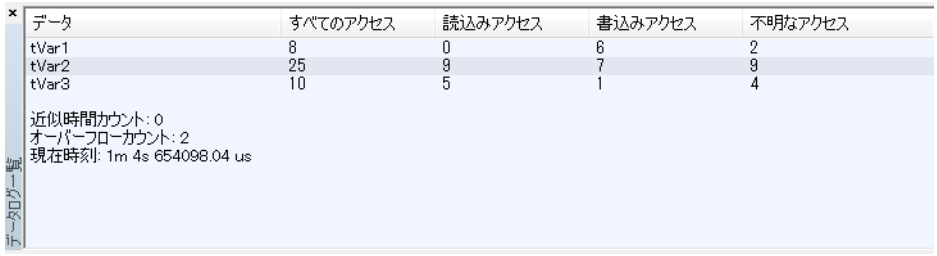
* 表示エリアの行をダブルクリックできます。そのラインの PC の値がソースコードで使用可能な場合、エディタウィンドウに、対応するソースコードが表示されます（ライブラリソースコードは除く）。

**コンテキストメニュー**

［割込みログ］ウィンドウのコンテキストメニューと同じです（385 ページの*［割込みログ］ウィンドウ*を参照）。

## ［データログ一覧］ウィンドウ

［データログ一覧］ウィンドウは、C-SPY ドライバメニューから使用できます。




このウィンドウには、特定のメモリ位置またはメモリエリアへのデータアクセスの概要が表示されます。

105 ページの*データログを開始するには*も参照してください。

### 要件

Cortex-M デバイスと以下の以下のいずれか 1 つが必要です。

- C-SPY シミュレータ
- C-SPY I-jet/JTAGjet ドライバ、およびデバッグプローブとターゲットシステム間の SWD インタフェースを持つ I-jet または I-jet Trace インサーキットデバッグプローブ
- C-SPY J-Link/J-Trace ドライバ、およびデバッグプローブとターゲットシステム間の SWD インタフェースを持つ J-Link または J-Trace デバッグプローブ

  J-Trace の場合、このウィンドウは ETM トレースが無効なときに使用可能です。デバッグの際に ETM が有効な場合、このウィンドウには限られた量の収集されたデータのみが表示されます。実行が停止すると、トレースデータ全体が表示されます。
- C-SPY ST-LINK ドライバと ST-LINK デバッグプローブ、デバッグプローブとターゲットシステム間の SWD インタフェース

### 表示エリア

このエリアの各行の以下の列には、各メモリ位置またはメモリエリアへのアクセスのタイプと回数が表示されます。

**データ**

アクセスの記録対象のデータオブジェクトの名前。アクセスの記録対象のデータオブジェクトを指定するには、**[データログ]** ブレークポイントダイアログボックスを使用します。135 ページの *データログブレークポイント*を参照してください。

現在の時間やサイクルが表示されます（実行開始からの実行時間またはサイクル数)。オーバフローの数が表示されます。

**合計アクセス数**

合計アクセス数。

リードアクセスとライトアクセスの合計が、[合計アクセス] の値以下の場合、何らかの理由によりターゲットシステムが有効なアクセスタイプ情報を提供しなかったアクセスログが存在します。

**リードアクセス数**

合計リードアクセス数。

**ライトアクセス数**

合計ライトアクセス数。

**不明なアクセス**

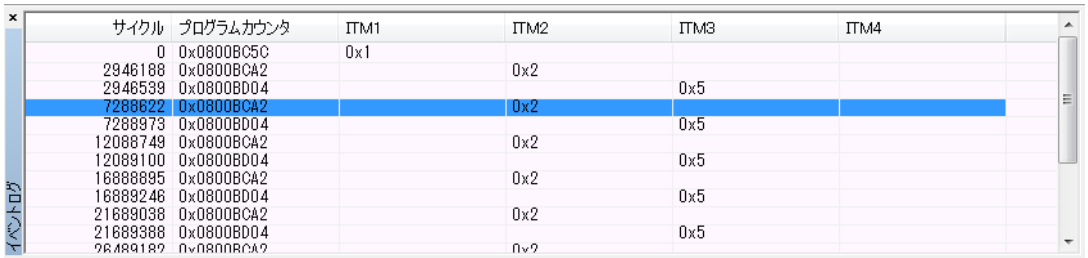
不明なアクセス数。つまり、アクセスタイプが分からないアクセス。

**コンテキストメニュー**

［割込みログ］ウィンドウのコンテキストメニューと同じです（385 ページの *［割込みログ］ウィンドウ*を参照）。

## ［イベントログ］ウィンドウ

［イベントログ］ウィンドウは、C-SPY ドライバメニューから使用できます。




| サイクル | プログラムカウンタ | ITM1 | ITM2 | ITM3 | ITM4 |
|---|---|---|---|---|---|
| 0 | 0x0800BC5C | 0x1 | | | |
| 2946188 | 0x0800BCA2 | | 0x2 | | |
| 2946539 | 0x0800BD04 | | | 0x5 | |
| 7288622 | 0x0800BCA2 | | 0x2 | | |
| 7288973 | 0x0800BD04 | | | 0x5 | |
| 12088749 | 0x0800BCA2 | | 0x2 | | |
| 12089100 | 0x0800BD04 | | | 0x5 | |
| 16888895 | 0x0800BCA2 | | 0x2 | | |
| 16889246 | 0x0800BD04 | | | 0x5 | |
| 21689038 | 0x0800BCA2 | | 0x2 | | |
| 21689388 | 0x0800BD04 | | | 0x5 | |
| 26489182 | 0x0800BCA2 | | 0x2 | | |

このウィンドウには、アプリケーションコード内で実行が特定の位置を通過したときに生成されたイベントが表示されます。Cortex ITM 通信チャンネルは、実行中のアプリケーションから C-SPY イベントシステムにイベントを渡すときに使用されます。

107 ページの*イベントログを開始するには*も参照してください。

**要件**

Cortex デバイスと以下のいずれかが必要です。

- C-SPY I-jet/JTAGjet ドライバ、およびデバッグプローブとターゲットシステム間の SWD インタフェースを持つ I-jet または I-jet Trace インサーキットデバッグプローブ
- C-SPY J-Link/J-Trace ドライバ、およびデバッグプローブとターゲットシステム間の SWD インタフェースを持つ J-Link または J-Trace デバッグプローブ

### 表示エリア

表示エリアの各行では、イベントが以下の列に表示されます。

**サイクル**

実行の開始からイベントまでのサイクルの数。この情報は、リセットでクリアされます。

ターゲットシステムが正確な数を収集できなかった場合は、おおよその値が斜体で表示されます。

この列は、コンテキストメニューから［**サイクル表示**］を選択した場合に有効になります。

**プログラムカウンタ**

PC の内容です。メモリアクセスを実行した命令のアドレスです。

---、ターゲットシステムがデバッガに情報を提供できなかったことを示します。

赤色で Overflow と表示されている場合、通信チャンネルがすべてのデータをターゲットシステムから送信できなかったことを示します。

**ITM1**
**ITM2**
**ITM3**
**ITM4**

イベントが記録される Cortex ITM の通信チャンネル。各イベントについて、イベントの値が表示されます。

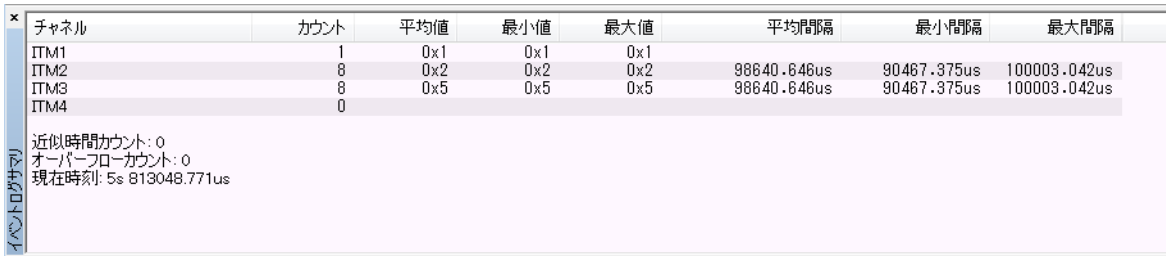
イベントを生成するアプリケーションのソースコードに、プリプロセッサマクロを追加します。107 ページの*イベントログを開始するには*を参照してください。

### コンテキストメニュー

［割込みログ］ウィンドウのコンテキストメニューと同じです（385 ページの*［割込みログ］ウィンドウ*を参照）。

# ［イベントログサマリ］ウィンドウ

［イベントログサマリ］ウィンドウは、C-SPY ドライバメニューから使用できます。




このウィンドウには、アプリケーションコード内で実行が特定の位置を通過したときに生成されたイベントのサマリが表示されます。Cortex ITM 通信チャンネルは、実行中のアプリケーションから C-SPY イベントシステムにイベントを渡すときに使用されます。

## 要件

Cortex デバイスと以下のいずれかが必要です。

- C-SPY I-jet/JTAGjet ドライバ、およびデバッグプローブとターゲットシステム間の SWD インタフェースを持つ I-jet または I-jet Trace インサーキットデバッグプローブ
- C-SPY J-Link/J-Trace ドライバ、およびデバッグプローブとターゲットシステム間の SWD インタフェースを持つ J-Link または J-Trace デバッグプローブ

## 表示エリア

**チャネル**

イベントが生成される通信チャネル名。

列の下には、現在の時間やサイクル（実行開始からの実行時間やサイクル数）が表示されます。オーバフローの数が表示されます。

**カウント**

ログに記録されたイベントの数。

**平均値**

受信したすべてのイベントの値の平均値。

**最小値**

受信したすべてのイベントの値の最小値。

**最大値**

受信したすべてのイベントの値の最大値。

**平均間隔**

イベント間の平均時間（サイクル数）。

**最短間隔**

2 つのイベント間の最短時間（サイクル数）。

**最長間隔**

2 つのイベント間の最長時間（サイクル数）。

### コンテキストメニュー

［割込みログ］ウィンドウのコンテキストメニューと同じです（385 ページの *［割込みログ］ウィンドウ*を参照）。

# ブレークポイント

- ブレークポイントの設定と使用の概要
- ブレークポイントの設定
- ブレークポイントについてのリファレンス情報

## ブレークポイントの設定と使用の概要

以下のトピックについて説明します:

- ブレークポイントを使用する理由
- ブレークポイントの設定の概要
- ブレークポイントの種類
- ブレークポイントアイコン
- C-SPY シミュレータのブレークポイント
- C-SPY ハードウェアデバッガドライバのブレークポイント
- ブレークポイントの設定元
- 138 ページの *［ブレークポイント］オプション*

### ブレークポイントを使用する理由

C-SPY® を使用すると、デバッグ中のアプリケーションでさまざまな種類のブレークポイントを設定して、必要な位置で実行を停止することができます。ブレークポイントを*コード*部分に設定すると、プログラムロジックが正しいかどうかを調べたり、トレースを出力したりできます。コードブレークポイントの他に、使用している C-SPY ドライバによっては、別の種類のブレークポイントを使用できる場合があります。たとえば、*データ*ブレークポイントを設定すると、データがいつどのように変更されるかを調べることができます。

ユーザが指定した特定の*条件*が成立したときに実行を停止させることができます。また、非表示で実行を停止してから再開することにより、C-SPY マクロ関数の実行などの *2 次アクション*をブレークポイントからトリガすることもできます。マクロ関数を定義すると、ハードウェアの動作のシミュレーションなど、さまざまなアクションを実行できます。

このようにさまざまな使い方ができるため、アプリケーションのステータスを検証するための柔軟なツールとして使用できます。

## ブレークポイントの設定の概要

ブレークポイントは異なるレベルの相互作用や精度、タイミング、自動化に合わせて、さまざまな数多くの方法で設定できます。定義したすべてのブレークポイントが［ブレークポイント］ウィンドウに表示されます。このウィンドウでは、すべてのブレークポイントの表示、ブレークポイントの有効化 / 無効化、新しいブレークポイントを定義するためのダイアログボックスの表示を実行できます。［ブレークポイントの使用］ウィンドウには、内部的に使用されるすべてのブレークポイントもリストされます（137 ページの *ブレークポイントの設定元*を参照）。

ブレークポイントは、ステップ動作と同じメカニズムを使用して、行単位よりも細かい精度で設定されます。精度に関する詳細については、78 ページの *ステップ実行*を参照してください。

デバッグセッションがアクティブでなくても、コードを編集しながらブレークポイントを設定できます。設定したブレークポイントは、デバッグセッションを開始するときに検証されます。ブレークポイントはデバッグセッション終了後も保持されます。

**注 :** ほとんどのハードウェアデバッグシステムでは、アプリケーションが実行中でないときにだけブレークポイントを設定できます。

## ブレークポイントの種類

使用している C-SPY ドライバによっては、C-SPY で別の種類のブレークポイントを使用できる場合があります。

## コードブレークポイント

コードブレークポイントは、プログラムロジックが正しいかどうかや、トレースの出力を取得するためにコードの位置を探すときに使用します。コードブレークポイントは、指定位置から命令をフェッチした時にトリガされます。特定のマシン命令にブレークポイントを設定した場合は、命令の実行前にブレークポイントがトリガされ、実行が停止します。

## ログブレークポイント

ログブレークポイントは、アプリケーションのソースコードにコードを追加することなく、トレース出力を追加する便利な方法です。ログブレークポイントは、指定位置から命令をフェッチ時にトリガされます。特定のマシン命令にブレークポイントを設定した場合は、命令の実行前にブレークポイントがトリガされ、実行が一時停止し、指定したメッセージが［C-SPY デバッグログ］ウィンドウに出力されます。

### ［トレース開始］および［トレース停止］ブレークポイント

トレース開始および停止ブレークポイントは、トレースデータの収集を開始および停止します。これは、2 つの実行ポイント間で命令を解析する便利な方法です。

### データブレークポイント

主にメモリ上の固定アドレスに割り当てられた変数に使用します。アクセス可能なローカル変数にブレークポイントを設定した場合、実際には対応するメモリアドレス（ロケーション）に設定されます。この位置の妥当性が保証されるのは、コードの一部だけです。データブレークポイントは、指定された位置のデータがアクセスされたときにトリガされます。通常は、データにアクセスする命令が実行された直後に、実行が停止します。

### データログブレークポイント

データログブレークポイントは、指定された位置のデータがアクセスされたときにトリガされます。特定のアドレスまたは範囲にブレークポイントを設定した場合は、その位置へのアクセスが発生するたびに、ログメッセージが［SWO トレース］ウィンドウ（シミュレータの［トレース］ウィンドウ）に表示されます。ログメッセージは［データログ］ウィンドウにも表示されます。データログは［タイムライン］ウィンドウのデータロググラフにも表示することができます（そのウィンドウが有効な場合）。ただし、これらのログメッセージを使用するには、**［SWO 設定］**ダイアログボックスでトレースデータを設定しておく必要があります（233 ページの *［SWO 設定］ダイアログボックス*を参照）。

### イミディエイトブレークポイント

C-SPY シミュレータでは、*イミディエイト*ブレークポイントを設定できます。これによって、命令の実行が一時的に停止します。このブレークポイントを使用すると、シミュレーションされたプロセッサがある位置からデータを読み込む直前か、ある位置にデータを書き込んだ直後に、C-SPY マクロ関数を呼び出すことができます。アクションが終了すると、命令の実行が再開されます。

イミディエイトブレークポイントは、メモリにマッピングされたさまざまな種類のデバイス（シリアルポートやタイマなど）をシミュレーションする場合に便利です。デバイスがメモリマッピングされた位置からシミュレーションされたプロセッサが読み込むと、C-SPY マクロ関数が実行されて適切なデータを供給します。逆に、デバイスがメモリマッピングされた位置にシミュレーションされたプロセッサが書き込むと、C-SPY マクロ関数が実行されて、書き込まれた値に応じた適切な動作を実行します。

## JTAG ウォッチポイント

C-SPY J-Link/J-Trace ドライバおよび C-SPY Macraigor ドライバでは、ARM7/9 コアの JTAG ウォッチポイント機構を活用できます。

ウォッチポイントは、ARM EmbeddedICE のマクロセルが提供する機能を使用して実装されます。このマクロセルは、JTAG インタフェースをサポートするすべての ARM コアの一部です。EmbeddedICE ウォッチポイントコンパレータでは、アドレスバス、データバス、CPU 制御信号、および外部入力信号と、定義されたウォッチポイントをリアルタイムで比較します。定義された条件がすべて真の場合、プログラムが中断します。

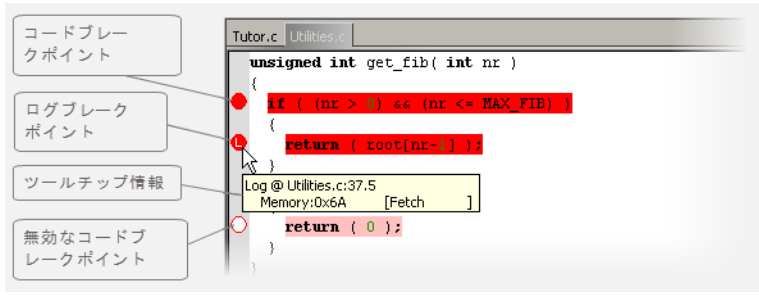
ウォッチポイントは、C-SPY で暗黙的に使用して、アプリケーションのコードブレークポイントやデータブレークポイントを設定します。リード / ライトメモリにブレークポイントを設定する場合、デバッガではウォッチポイントが 1 つのみ必要です。リードオンリーメモリにブレークポイントを設定する場合、各ブレークポイントには 1 つのウォッチポイントが必要です。マクロセルでは 2 つのハードウェアウォッチポイントしか実装しないため、リードオンリーメモリのブレークポイントの最大値は 2 つです。

ARM JTAG ウォッチポイント機構の詳細については、Advanced RISC Machines Ltd の以下の資料を参照してください。

- 『ARM7TDMI (rev 3) Technical Reference Manual』の第 5 章「Debug Interface」および付録 B の「Debug in Depth」
- アプリケーションノート 28 の「The ARM7TDMI Debug Architecture」

## ブレークポイントアイコン

ブレークポイントはエディタウィンドウの左余白にあるアイコンでマークを付け、コードブレークポイント用とログブレークポイント用で違うアイコンを使用します。




ブレークポイントアイコンが表示されない場合は、**[ブックマークの表示]** オプションが選択されていることを確認します（*『ARM 用 IDE プロジェクト管理およびビルドガイド』* のエディタオプションを参照）。

マウスポインタをブレークポイントアイコンに置くだけで、同じ位置に設定したすべてのブレークポイントに関する詳細なツールチップ情報を取得できます。最初の行がユーザブレークポイント情報を、後続する行が、ユーザブレークポイントの実装に使用する物理ブレークポイントを説明します。後者の情報は、［ブレークポイントの使用］ウィンドウでも表示されます。

**注 :** ブレークポイントアイコンは、使用している C-SPY ドライバによって異なった外観になります。

### C-SPY シミュレータのブレークポイント

C-SPY シミュレータは全種類のブレークポイントをサポートしており、ブレークポイントを無制限に設定できます。

### C-SPY ハードウェアデバッガドライバのブレークポイント

ハードウェアデバッガシステムステムに C-SPY ドライバを使用して、さまざまな種類のブレークポイントを設定できます。設定可能なブレークポイントの数は、ターゲットシステム上で使用できる*ハードウェアブレークポイント*の数や、*ソフトウェアブレークポイント*が有効かどうかによって変わります（有効な場合は、設定できるブレークポイントの数は限られません）。

ソフトウェアブレークポイントが有効な場合、デバッガはソフトウェアブレークポイントより先に、まず利用可能なハードウェアブレークポイントを使用します。ソフトウェアブレークポイントが有効でない場合に、使用可能なハードウェアブレークポイントの数を超えると、デバッガがシングルステップで実行するようになります。この場合、大幅に実行速度が低下します。このため、異なるブレークポイントの設定元に注意する必要があります。

異なるターゲットシステムのブレークポイントの特徴については、メーカのドキュメントを参照してください。

### ブレークポイントの設定元

デバッガシステムには複数のブレークポイントの設定元が存在します。

### ユーザブレークポイント

［ブレークポイント］ダイアログボックスで定義したり、［エディタ］ウィンドウでブレークポイントを切り替えると、通常は物理的なブレークポイントが 1 つ使用されますが、これは状況に応じて大きく異なります。一部のユーザブレークポイントは複数の物理的ブレークポイントを使用します。逆に複数のユーザブレークポイントが 1 つの物理的ブレークポイントを共有することもできます。ユーザブレークポイントは、たとえば「`Data @[R] callCount`」のように、［ブレークポイントの使用］ウィンドウと［ブレークポイント］ウィンドウに同じように表示されます。

### C-SPY 自身

C-SPY 自身もブレークポイントを使用します。C-SPY は以下の場合にブレークポイントを設定します。

- デバッガオプション［**指定位置まで実行**］が選択され、いずれかのステップコマンドが使用されている場合。これらはデバッグセッション中にのみ設定される一時的なブレークポイントです。したがって、［ブレークポイント］ウィンドウにはこれらのブレークポイントは表示されません。
- リンカオプション［**セミホスティング**］または［**IAR ブレークポイント**］が選択されています。

  DLIB ランタイム環境では、C-SPY は `__DebugBreak` ラベル上にシステムブレークポイントを設定します。

これらのブレークポイントの設定元は、［ブレークポイントの使用］ウィンドウに、たとえば「`C-SPY Terminal I/O & libsupport module`」のように表示されます。

### C-SPY プラグインモジュール

たとえば、リアルタイムオペレーティングシステム用のモジュールは、追加のブレークポイントを使用します。特にデフォルトでは、［スタック］ウィンドウで 1 つの物理的ブレークポイントを使用します。

#### ［スタック］ウィンドウで使用するブレークポイントを無効にするには、以下の操作を行います。

1 ［**ツール**］>［**オプション**］>［**スタック**］を選択します。

2 ［**プログラム開始までスタックポインタを無効にする**］の *［ラベル］* オプションの選択を解除します。

［スタック］ウィンドウを完全に無効にするには、［**ツール**］>［**オプション**］>［**スタック**］を選択して、すべてのオプションの選択が解除されているか確認してください。

### ［ブレークポイント］オプション

以下の C-SPY ドライバでは、C-SPY を起動する前にドライバ固有のブレークポイントオプションをいくつか設定できます。

- GDB サーバ
- I-jet/JTAGjet
- J-Link/J-Trace
- CMSIS-DAP
- Macraigor

詳細については、164 ページの *［ブレークポイント］ダイアログボックス*を参照してください。

# ブレークポイントの設定

以下のタスクについて説明します :

- ブレークポイントのさまざまな設定方法
- シンプルなコードブレークポイントトグル
- ダイアログボックスによるブレークポイントの設定
- ［メモリ］ウィンドウでのデータブレークポイントの設定
- システムマクロによるブレークポイントの設定
- 例外ベクタ上へのブレークポイントの設定
- __ramfunc 宣言関数のブレークポイントの設定
- ブレークポイントのヒント

## ブレークポイントのさまざまな設定方法

ブレークポイントは、さまざまな方法で設定できます。

- シンプルなコードブレークポイントトグル。
- エディタウィンドウ、［ブレークポイント］ウィンドウ、［逆アセンブリ］ウィンドウのコンテキストメニューから**［新規ブレークポイント］**ダイアログボックスおよび**［ブレークポイントの編集］**ダイアログボックスを使用する方法。これらのダイアログボックスでは、すべてのブレークポイントオプションにアクセスできます。
- ［メモリ］ウィンドウでメモリエリアに直接データブレークポイントを設定する方法。
- 定義済システムマクロを使用してブレークポイントを設定する方法。自動化が可能になります。

方法によって簡単さ、複雑さ、自動化のレベルが異なります。

## シンプルなコードブレークポイントトグル

コードブレークポイントのトグルは、簡単にブレークポイントを設定するための方法です。エディタウィンドウと［逆アセンブリ］ウィンドウの両方で、次の方法を使用できます。

- ウィンドウ左側の灰色で表示された余白部分をクリック

- ブレークポイントを設定する C 言語のソース文、アセンブラ命令に挿入ポイントを配置して、ツールバーの［**ブレークポイントの切替え**］ボタンをクリック
- ［**編集**］>［**ブレークポイントの切替え**］を選択
- 右クリックして、表示されるコンテキストメニューで［**ブレークポイントの切替え**］を選択

### ダイアログボックスによるブレークポイントの設定

ブレークポイントダイアログボックスを使用する利点は、グラフィカルインタフェースで対話的にブレークポイントの特性を微調整できるということです。この方法では、オプションを設定した後、すぐにブレークポイントが意図したとおりに動作するかどうかをテストできます。

ブレークポイントダイアログボックスで定義したブレークポイントはすべて、デバッグセッションが終了後も保持されます。

ブレークポイントダイアログボックスは、エディタウィンドウ、［ブレークポイント］ウィンドウ、［逆アセンブリ］ウィンドウのコンテキストメニューから開くことができます。

#### 新しいブレークポイントを設定するには：

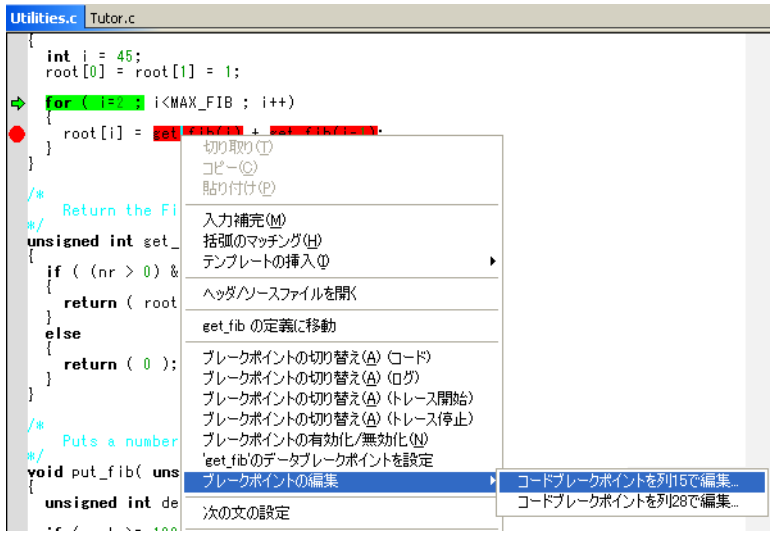
1. ［**表示**］>［**ブレークポイント**］を選択して、［ブレークポイント］ウィンドウを開きます。
2. ［ブレークポイント］ウィンドウで右クリックし、コンテキストメニューで［**新規ブレークポイント**］を選択します。
3. サブメニューで、設定するブレークポイントの種類を選択します。

   使用している C-SPY ドライバによっては、別の種類のブレークポイントを使用できる場合があります。
4. 表示される［ブレークポイント］ダイアログボックスで、ブレークポイントの設定をして **[OK]** をクリックします。

   ブレークポイントが［ブレークポイント］ウィンドウに表示されます。

### 既存のブレークポイントを変更するには：

1 ［ブレークポイント］ウィンドウ、エディタウィンドウ、または［逆アセンブリ］ウィンドウで変更するブレークポイントを選択し、右クリックしてコンテキストメニューを開きます。




同じソースコードの行に複数のブレークポイントが設定されている場合、それらはサブメニューに一覧表示されます。

2 コンテキストメニューで、目的のコマンドを選択します。

3 表示される［ブレークポイント］ダイアログボックスで、ブレークポイントの設定をして **[OK]** をクリックします。

ブレークポイントが［ブレークポイント］ウィンドウに表示されます。

## ［メモリ］ウィンドウでのデータブレークポイントの設定

［メモリ］ウィンドウでメモリロケーションにブレークポイントを直接設定することができます。ウィンドウを右クリックして、表示されるコンテキストメニューからブレークポイントコマンドを選択します。範囲にブレークポイントを設定するには、メモリの該当領域を選択します。

ブレークポイントは［メモリ］ウィンドウでは強調表示されません。代わりに、**［表示］**メニューの［ブレークポイント］ウィンドウを使用して、確認や編集、削除ができます。［メモリ］ウィンドウで設定したブレークポイントは、リードとライトの両方のアクセスでトリガされます。このウィンドウで

定義したブレークポイントはすべて、デバッグセッションが終了後も保持されます。

**注 :** ［メモリ］ウィンドウで直接ブレークポイントを設定するには、使用するドライバでそれがサポートされている必要があります。

## システムマクロによるブレークポイントの設定

ブレークポイントの設定は、［ブレークポイント］ダイアログボックス以外に、C-SPY の組込みシステムマクロでも行えます。システムマクロを使用してブレークポイントを設定すると、ブレークポイントの特性はマクロパラメータとして指定されます。

マクロによる定義は、要求どおりのブレークポイント設定ができない場合に便利です。組込みのシステムマクロを使用してブレークポイントをマクロファイルに定義し、C-SPY の起動時にマクロファイルを実行することができます。これにより、ブレークポイントは、C-SPY を起動するたびに自動的に設定されます。他にも、デバッグセッションがドキュメント化される、開発プロジェクトに携わる複数のエンジニア間でマクロファイルを共有できるといった長所があります。

**注 :** システムマクロを使用して設定されたブレークポイントも、［ブレークポイント］ウィンドウで表示や変更を行えます。ダイアログボックスを使用して定義されたブレークポイントと異なり、システムマクロを使用して定義されたブレークポイントはデバッグセッションを終了するとすべて削除されます。

以下のブレークポイントマクロが使用できます。

| ブレークポイント用の C-SPY マクロ | シミュレータ | I-jet/JTAGjet | J-Link/J-Trace | CMSIS-DAP |
|---|---|---|---|---|
| `__setCodeBreak` | あり | あり | あり | あり |
| `__setDataBreak` | あり | あり | なし | あり |
| `__setLogBreak` | あり | あり | あり | あり |
| `__setDataLogBreak` | あり | あり | なし | なし |
| `__setSimBreak` | あり | なし | なし | なし |
| `__setTraceStartBreak` | あり | あり | なし | なし |
| `__setTraceStopBreak` | あり | あり | なし | なし |
| `__clearBreak` | あり | あり | あり | あり |

*表 9: ブレークポイント用の C-SPY マクロ*

| ブレークポイント用の C-SPY マクロ | GDB Server / Macraigor / RDI | ST-LINK | PE micro | TI Stellaris / TI XDS | Angel / IAR ROM モニタ |
|---|---|---|---|---|---|
| __setCodeBreak | あり | あり | あり | あり | あり |
| __setDataBreak | なし | なし | なし | なし | なし |
| __setLogBreak | あり | あり | あり | あり | あり |
| __setDataLogBreak | なし | なし | なし | なし | なし |
| __setSimBreak | なし | なし | なし | なし | なし |
| __setTraceStartBreak | なし | なし | なし | なし | なし |
| __setTraceStopBreak | なし | なし | なし | なし | なし |
| __clearBreak | あり | あり | あり | あり | あり |

*表 10: ブレークポイント用の C-SPY マクロ*

各ブレークポイントマクロの詳細については、410 ページの *C-SPY システムマクロについてのリファレンス情報*を参照してください。

## セットアップマクロファイルを使用して C-SPY 起動時にブレークポイントを設定

セットアップマクロファイルを使用して C-SPY の起動時にブレークポイントを定義できます。手順の詳細については、396 ページの *C-SPY マクロの使用*を参照してください。

## 例外ベクタ上へのブレークポイントの設定

ARM9、Cortex-R4、Cortex-M3 の各デバイスで例外ベクタにブレークポイントを設定できます。**［ベクタキャッチ］**ダイアログボックスを使用すると、ハードウェアのブレークポイントを使用せずに、割込みベクタテーブルのベクタにブレークポイントを直接設定できます。詳細については、167 ページの *［ベクタキャッチ］ダイアログボックス*を参照してください。

C-SPY I-jet/JTAGjet ドライバ、C-SPY J-Link/J-Trace ドライバ、C-SPY RDI ドライバの場合、オプションのダイアログボックスにすでにあるベクタにブレークポイントを直接設定することも可能です（556 ページの *J-Link/J-Trace の設定オプション*および 565 ページの *RDI* を参照）。

この手順は C-SPY I-jet/JTAGjet ドライバ、C-SPY J-Link/J-Trace ドライバ、C-SPY Macraigor ドライバに該当します。

例外ベクタ上にブレークポイントを設定するには：

1. 正しいデバイスを選択します。C-SPY を起動する前に、［**プロジェクト**］>［**オプション**］を選択して、［**一般オプション**］カテゴリを選択します。［**ターゲット**］ページで使用可能な［**プロセッサ選択**］ドロップダウンリストから、該当するコアまたはデバイスを選択します。
2. C-SPY を起動します。
3. *［C-SPY ドライバ］*>［**ベクタキャッチ**］を選択します。デフォルトでは、ベクタはブレークポイントオプションページの設定に基づいて選択されます（164 ページの *［ブレークポイント］ダイアログボックス*を参照）。
4. ［**ベクタキャッチ**］ダイアログボックスで、ブレークポイントを設定するベクタを選択し、**[OK]** をクリックします。ブレークポイントは、例外の開始時にのみトリガされます。

### __RAMFUNC 宣言関数のブレークポイントの設定

`__ramfunc` 宣言関数にブレークポイントを設定するには、プログラムの実行が `main` 関数に達する必要があります。システム起動コードでは、すべての `__ramfunc` 宣言関数を、コードが格納されている場所（一般的にはフラッシュメモリ）から RAM まで移動します。つまり、`__ramfunc` 宣言関数は適切な場所にはないため、`main` 関数を実行するまでブレークポイントを設定できません。この問題を解決するには［**ソフトウェアブレークポイント復元位置**］オプションを使用します。164 ページの *［ブレークポイント］ダイアログボックス*の［**ソフトウェアブレークポイント復元位置**］オプションの項を参照してください。

また、エディタから追加された `__ramfunc` 宣言関数のブレークポイントは、C-SPY の起動前およびデバッグセッションの終了前に無効にする必要があります。

`__ramfunc` キーワードについては、*ARM 用 IAR C/C++ 開発ガイド*を参照してください。

### ブレークポイントのヒント

以下は、ブレークポイントの設定に関連して役に立つヒントです。

#### 不正な関数引数のトレース

ポインタ引数を持つ関数が時々 `NULL` 引数によって誤って呼び出される場合、その動作をデバッグした方がよいときがあります。以下の方法が役に立ちます。

- 関数の最初の行にブレークポイントを設定して、パラメータが `0` のときにだけ条件が真となるようにします。このブレークポイントは、問題となる

状況が実際に発生するまでトリガされません。この方法の利点は、余分なソースコードが必要ないことです。欠点は、実行速度が極端に低下する可能性があることです。

- 問題のある関数で assert マクロを使用できます。たとえば、次のようになります。

```
int MyFunction(int * MyPtr)
{
  assert(MyPtr != 0); /* アサートマクロがソースコードに
                         追加されます。*/
  /* 関数の残りがここに入ります */
}
```

条件が真のときは必ず実行が中断します。利点は、実行速度がわずかにしか影響を受けないことですが、欠点はソースコードに小さいフットプリントが追加されることです。また、実行の停止を除去する唯一の方法は、マクロを削除してソースコードをリビルドすることです。

- assert マクロを使用する代わりに、次のように関数を修正できます。

```
int MyFunction(int * MyPtr)
{
  if(MyPtr == 0)
     MyDummyStatement; /* ブレークポイントを設定するダミーの文 */
  /* 関数の残りがここに入ります */
}
```

また、条件が真のときに常に実行が中断するように、追加のダミー文にブレークポイントを設定する必要があります。利点は、実行速度がわずかにしか影響を受けないことですが、欠点はソースコードに小さいフットプリントが追加されることです。ただし、この方法ではブレークポイントを削除するだけで、実行の停止を除去することができます。

### タスクを処理して実行を継続する

ブレークポイントがトリガされたらタスクを処理して、自動的に実行を継続できます。

[アクション] テキストボックスを使用すると、C-SPY マクロ関数などのアクションをブレークポイントに関連付けることができます。ブレークポイントがトリガされ、アプリケーションの実行が停止すると、マクロ関数が実行されます。この場合は、実行は自動的に継続されません。

代わりに、0（偽）を返す条件を設定できます。ブレークポイントがトリガされると、条件（タスクを実行する C-SPY マクロの呼出しなど）が評価され、真ではないために実行が継続します。

C-SPY マクロ関数が単純なタスクを実行する例を考えます。

```
__var my_counter;

count()
{
  my_counter += 1;
  return 0;
}
```

この関数をブレークポイントの条件として使用するには、**[条件]** の **[式]** テキストボックスに「`count()`」と入力します。これにより、ブレークポイントがトリガされると、タスクが実行されます。マクロ関数 `count` は常に `0` を返すため、条件は偽であり、プログラムは停止することなく自動的に再開されます。

## ブレークポイントについてのリファレンス情報

リファレンス情報 :

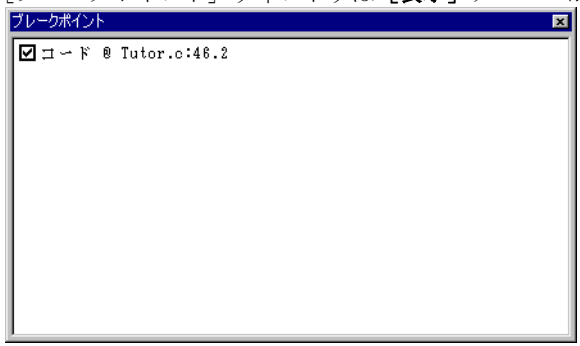
- 147 ページの *[ブレークポイント] ウィンドウ*
- 149 ページの *[ブレークポイントの使用] ウィンドウ*
- 150 ページの *[コード] ブレークポイントダイアログボックス*
- 152 ページの *[JTAG ウォッチポイント] ダイアログボックス*
- 155 ページの *[ログ] ブレークポイントダイアログボックス*
- 157 ページの *[データブレークポイント] ダイアログボックス*
- 162 ページの *[データログ] ブレークポイントダイアログボックス (C-SPY ハードウェアドライバ)*
- 164 ページの *[ブレークポイント] ダイアログボックス*
- 166 ページの *[イミディエイト] ブレークポイントダイアログボックス*
- 167 ページの *[ベクタキャッチ] ダイアログボックス*
- 168 ページの *[位置入力] ダイアログボックス*
- 170 ページの *[ソースの曖昧さの解決] ダイアログボックス*

関連項目 :

- 410 ページの *C-SPY システムマクロについてのリファレンス情報*
- 225 ページの *トレースについてのリファレンス情報*

# ［ブレークポイント］ウィンドウ

［ブレークポイント］ウィンドウは［**表示**］メニューから利用できます。




［ブレークポイント］ウィンドウには、定義するすべてのブレークポイントが一覧表示されます。

このウィンドウでは、ブレークポイントのモニタや有効 / 無効の切替えを簡単に行うことができます。また、新しいブレークポイントの定義や、既存のブレークポイントの修正も行うことができます。

### 要件

ありません。このウィンドウは常に使用できます。

### 表示エリア

このエリアには、定義するすべてのブレークポイントが一覧表示されます。それぞれのブレークポイントについて、ブレークポイントの種類、ソースファイル、ソース行、ソース列の情報が表示されます。

### コンテキストメニュー

以下のコンテキストメニューがあります。


以下のコマンドがあります。

**ソースへ移動**

ブレークポイントに対応する位置がソースにある場合に、挿入ポイントをブレークポイント位置に移動します。［ブレークポイント］ウィンドウでブレークポイントをダブルクリックした場合も、同一の操作が実行されます。

**編集**

選択したブレークポイントに対して［ブレークポイント］ダイアログボックスを開きます。

**削除**

ブレークポイントを削除します。Delete キーを押した場合も、同一の操作が実行されます。

**有効化**

ブレークポイントを有効にします。行の最初にあるチェックボックスが選択されます。チェックボックスを手動で選択しても、同一の結果になります。このコマンドは、ブレークポイントが無効になっている場合にだけ使用できます。

**無効**

ブレークポイントを無効にします。行の最初にあるチェックボックスが選択解除されます。チェックボックスを手動で選択解除しても、このコマンドを実行できます。このコマンドは、ブレークポイントが有効になっている場合にだけ使用できます。

**すべて有効**

定義されたすべてのブレークポイントを有効にします。

**すべて無効**

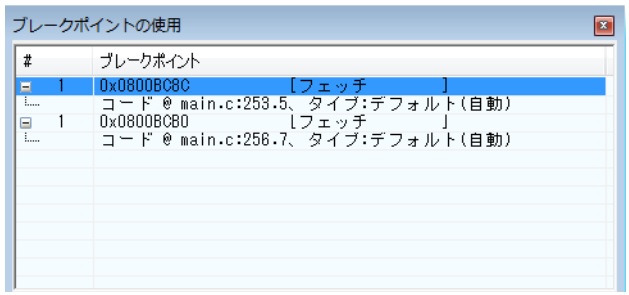
定義されたすべてのブレークポイントを無効にします。

**新規ブレークポイント**

使用可能な種類のブレークポイントについて、［ブレークポイント］ダイアログボックスを開くためのサブメニューを表示します。このダイアログボックスで定義したブレークポイントはすべて、デバッグセッションが終了後も保持されます。

## ［ブレークポイントの使用］ウィンドウ

**［ブレークポイントの使用］**ウィンドウは、使用する C-SPY ドライバに固有のメニューから利用できます。




［ブレークポイントの使用］ウィンドウには、ターゲットシステムで現在設定されているすべてのブレークポイントのリストが表示されます。これらのブレークポイントには、ユーザ定義によるブレークポイントと C-SPY が内部的に使用しているブレークポイントが含まれます。このダイアログボックスの項目のフォーマットは、使用している C-SPY ドライバによって異なります。

このウィンドウでは、すべてのブレークポイントの下位レベルの情報が表示されます。これらの項目は、［ブレークポイント］ウィンドウで表示されるブレークポイントのリストと関連はありますが、同一ではありません。

C-SPY はステップの実行時にブレークポイントを使用します。［ブレークポイントの使用］ウィンドウは以下の目的で使用します。

- すべてのブレークポイント設定元の特定
- ターゲットシステムでサポートされているアクティブなブレークポイントの数をチェック
- 可能であれば、使用できるブレークポイントを効率よく利用できるようにデバッガを設定

### 要件

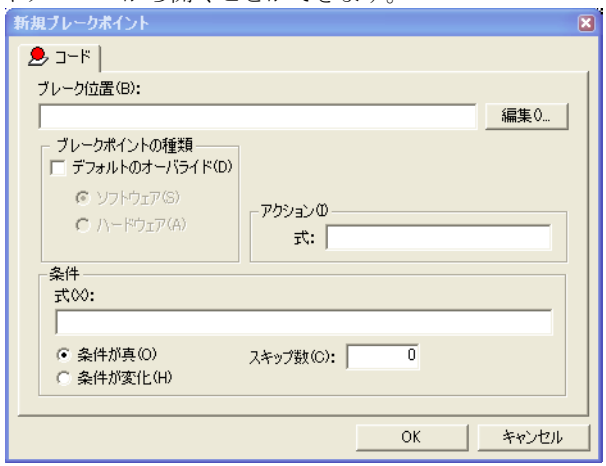
ありません。このウィンドウは常に使用できます。

### 表示エリア

リスト内の各ブレークポイントについて、アドレスとアクセスタイプが表示されます。また、リストの各ブレークポイントを拡張すると、その発生元が表示されます。

## [コード] ブレークポイントダイアログボックス

[コード] ブレークポイントダイアログボックスは、エディタウィンドウ、[ブレークポイント] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。




この図は C-SPY シミュレータを示します。

[コード] ブレークポイントダイアログボックスを使用して、コードブレークポイントを設定します。

### 要件

ありません。このダイアログボックスは常に使用できます。

### ブレーク位置

テキストボックスでブレークポイントのコードの位置を指定します。または、[編集] ボタンをクリックして [位置入力] ダイアログボックスを表示します（168 ページの *[位置入力] ダイアログボックス*を参照）。

### ブレークポイントの種類

デフォルトのブレークポイントの種類をオーバライドします。［デフォルトのオーバライド］チェックボックスを選択し、［ソフトウェア］と［ハードウェア］オプションから選択します。

以下の C-SPY ドライバについて、ブレークポイントタイプを指定できます。

- C-SPY I-jet/JTAGjet ドライバ
- C-SPY CMSIS-DAP ドライバ
- C-SPY GDB サーバドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY Macraigor JTAG ドライバ

### サイズ

ブレークポイントがトリガされる位置にサイズ（特に範囲）があるべきかを指定します。指定したメモリ範囲に対してフェッチアクセスが発生するごとに、ブレークポイントがトリガされます。サイズの指定方法を選択します。

**自動**

サイズが自動設定される（通常は 1）。

**手動**

テキストボックスでブレークポイント範囲のサイズを指定します。

### アクション

有効な C-SPY 式を指定します。この式は、ブレークポイントのトリガ時に条件が真であるときに評価されます。詳細については、144 ページの*ブレークポイントのヒント*を参照してください。

### 条件

単純または複雑な条件を指定します。

**式**

有効な C-SPY 式を指定します（100 ページの *C-SPY 式*を参照）。

**条件式の値が真**

式の値が真の場合に、ブレークポイントがトリガされます。

**条件式の値が変更**

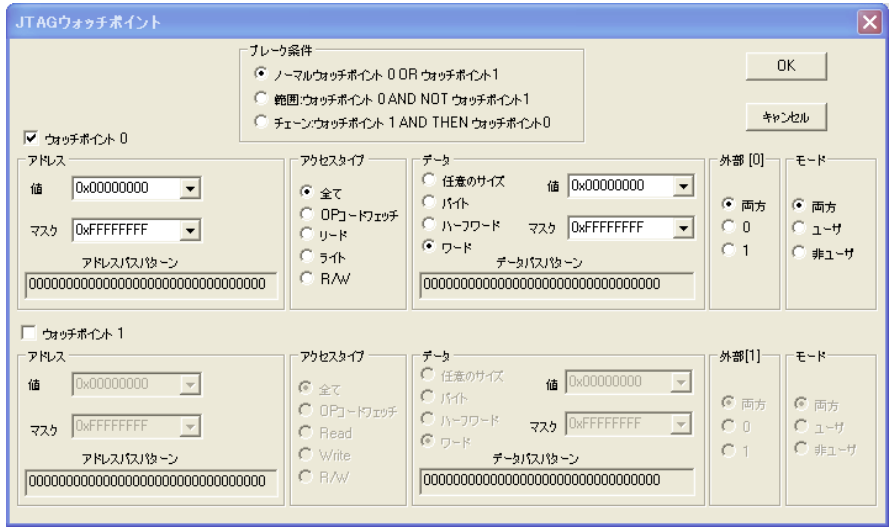
最後の評価時以降に式の値が変化した場合に、ブレークポイントがトリガされます。

**スキップ数**

ブレークポイントがトリガを開始するまでにブレークポイント条件が真となる回数を指定します。この回数に達すると、条件が満たされるたびにブレークポイントがトリガされます。

## ［JTAG ウォッチポイント］ダイアログボックス

［JTAG ウォッチポイント］ダイアログボックスは、ドライバ個別のメニューから開きます。




このダイアログボックスを使用して、2 つのハードウェアウォッチポイント装置を直接制御します。必要なウォッチポイント数（ブレークポイントシステムが使用する暗黙的なウォッチポイントを含む）が 2 を超えると、**[OK]** ボタンをクリックしたときにエラーメッセージが表示されます。このチェックは C-SPY の［**実行**］ボタンをクリックした場合も行われます。

**0x20-0xFF の範囲でアクセスをトリガするには、以下の手順を行います。**

1 ［**ブレーク条件**］を［**範囲**］に設定します。

2 ウォッチポイント 0 のアドレス値を 0 に設定し、`0xFF` にマスクします。

3 ウォッチポイント 1 のアドレス値を 0 に設定し、`0x1F` にマスクします。

### 要件

以下のいずれかが必要です。

- J-Link/J-Trace ドライバ
- Macraigor ドライバ

### アドレス

モニタするアドレスを指定します。

**値** アドレスまたは評価の結果がアドレスになる C-SPY 式を指定します。また、以前に監視したアドレスをドロップダウンリストから選択することができます。C-SPY 式の詳細については、100 ページの *C-SPY 式*を参照してください。

**マスク** 値の各ビットを制限します。マスクのビットがゼロである場合、値に対応するビットが比較のときに無視されます。どのアドレスとも一致させるには、0 を入力します。なお、マスクの値は ARM ハードウェアマニュアルで使用する表記法に基づいて変換されます。

**アドレスバスパターン** アドレスコンパレータで使用するビットパターンが表示されます。マスクで指定したように無視されたビットは x と表示されます。

### アクセスタイプ

モニタするデータのアクセスタイプを選択します。

**全て** どのアクセスタイプにも一致。

**OP フェッチ** 演算コード（命令）フェッチに一致。

**リード** 指定された位置から読み取ります。

**ライト** 指定の位置に書き込みます。

**R/W** 指定された位置から読み取り / 書き込みを行います。

### データ

モニタするデータを指定します。サイズについては、以下から選択します。

**任意のサイズ** あらゆるサイズのデータアクセスに一致。

| | |
|---|---|
| **バイト** | バイトサイズのアクセスに一致。 |
| **ハーフワード** | ハーフワードのサイズのアクセスに一致。 |
| **ワード** | ワードサイズのアクセスに一致。 |

モニタする値を指定します。以下から選択します。

| | |
|---|---|
| **値** | 値または C-SPY 式を指定します。また、以前に監視した値をドロップダウンリストから選択することができます。C-SPY 式の詳細については、100 ページの *C-SPY 式*を参照してください。 |
| **マスク** | 値の各ビットを制限します。マスクのビットがゼロである場合、値に対応するビットが比較のときに無視されます。どのアドレスとも一致させるには、0 を入力します。なお、マスクの値は ARM ハードウェアマニュアルで使用する表記法に基づいて変換されます。 |
| **データバスパターン** | アドレスコンパレータで使用するビットパターンが表示されます。マスクで指定したように無視されたビットは `x` と表示されます。 |

**外部**

外部入力の状態を定義します。以下から選択します。

| | |
|---|---|
| **全て** | 状態を無視します。 |
| **0** | 状態を「低」として定義します。 |
| **1** | 状態を「高」として定義します。 |

**モード**

一致するためにアクティブでなければならない CPU モードを選択します。以下から選択します。

| | |
|---|---|
| **ユーザ** | CPU モードの USER を選択します。 |
| **非ユーザ** | CPU モードの SYSTEM SVC、UND、ABORT、IRQ または FIQ を選択します。 |
| **全て** | CPU モードを無視します。 |

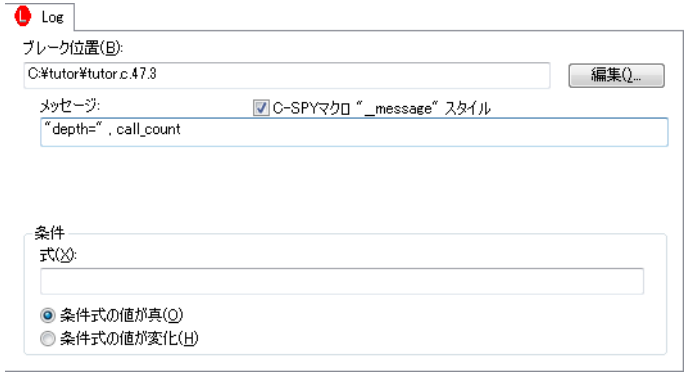
### ブレーク条件

定義されたウォッチポイントをどう使用するかを選択します。以下から選択します。

| | |
|---|---|
| **通常** | 2 つのウォッチポイントを個別に使用します (OR)。 |
| **範囲** | 両方のウォッチポイントを組み合わせて対象の範囲とします。ウォッチポイント 0 で定義する開始点と、ウォッチポイント 1 で定義する終了点の範囲です。選択可能な範囲は 2 の累乗で制限されます。 |
| **チェーン** | ウォッチポイント 1 のトリガをウォッチポイント 0 にします。次にウォッチポイント 0 がトリガされると、プログラムが中止されます。 |

## [ログ] ブレークポイントダイアログボックス

[**ログ**] ブレークポイントダイアログボックスは、エディタウィンドウ、[ブレークポイント] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。




この図は C-SPY シミュレータを示します。

[**ログ**] ブレークポイントダイアログボックスを使用して、ログブレークポイントを設定します。

### 要件

ありません。このダイアログボックスは常に使用できます。

**トリガ位置**

ブレークポイントのコード位置を指定します。または、**[編集]**ボタンをクリックして**[位置入力]**ダイアログボックスを表示します（168 ページの*[位置入力]ダイアログボックス*を参照）。

**メッセージ**

[C-SPY デバッグログ]ウィンドウで表示するメッセージを指定します。通常のテキストか、コンマ区切りの引数リスト（**[C-SPY マクロ "__message" スタイル]**オプションも選択している場合）を入力します。

**C-SPY マクロ "__message" スタイル**

[メッセージ]テキストボックスで指定したコンマ区切りの引数リストを C-SPY マクロ言語文 `__message` の引数として使用する場合は、このオプションを選択します（405 ページの*フォーマットした出力*を参照）。

**条件**

単純または複雑な条件を指定します。

**式**

有効な C-SPY 式を指定します（100 ページの *C-SPY 式*を参照）。

**条件式の値が真**

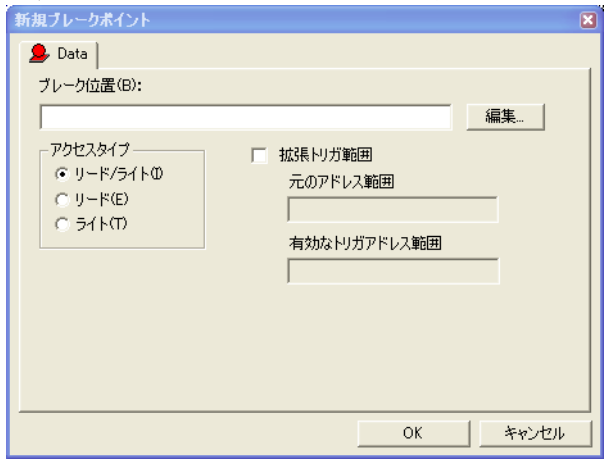
式の値が真の場合に、ブレークポイントがトリガされます。

**条件式の値が変更**

最後の評価時以降に式の値が変化した場合に、ブレークポイントがトリガされます。

# [データブレークポイント] ダイアログボックス

[データ] ブレークポイントダイアログボックスは、エディタウィンドウ、[ブレークポイント] ウィンドウ、[メモリ] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。




この図は C-SPY シミュレータを示します。

[データ] ブレークポイントダイアログボックスを使用して、データブレークポイントを設定します。データブレークポイントによって単一命令内で実行が停止することはありません。ブレークポイントは命令の実行後に記録、レポートされます。

## 要件

以下のいずれかが必要です。

- C-SPY シミュレータ
- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY ST-LINK ドライバ
- C-SPY RDI ドライバ
- C-SPY CMSIS-DAP ドライバ
- C-SPY Macraigor ドライバ
- C-SPY GDB サーバドライバ
- C-SPY TI Stellaris ドライバ

### ブレーク位置

テキストボックスでブレークポイントのデータ位置を指定します。または、**［編集］**ボタンをクリックして**［位置入力］**ダイアログボックスを表示します（168 ページの*［位置入力］ダイアログボックス*を参照）。

### アクセスタイプ

ブレークポイントをトリガするメモリアクセスの種類を選択します。

**リード / ライト**
指定された位置から読み取り / 書き込みを行います。

**リード**
指定された位置から読み取ります。

**ライト**
指定の位置に書き込みます。

### サイズ

ブレークポイントがトリガされる位置にサイズ（特に範囲）があるべきかを指定します。指定したメモリ範囲に対してフェッチアクセスが発生するごとに、ブレークポイントがトリガされます。サイズの指定方法を選択します。

**自動**
ブレークポイントが設定されている式のタイプに基づいて、サイズが自動的に決まります。たとえば、ブレークポイントを 12 バイトの構造体に設定すると、そのブレークポイントのサイズは 12 バイトになります。

**手動**
テキストボックスでブレークポイント範囲のサイズを指定します。

配列、構造体、共用体などのデータ構造へのアクセスによってデータブレークポイントをトリガする必要がある場合に、この機能は便利です。

### アクション

有効な C-SPY 式を指定します。この式は、ブレークポイントのトリガ時に条件が真であるときに評価されます。詳細については、144 ページの*ブレークポイントのヒント*を参照してください。

### 条件

単純または複雑な条件を指定します。

**式**

有効な C-SPY 式を指定します（100 ページの *C-SPY 式*を参照）。

**条件式の値が真**

式の値が真の場合に、ブレークポイントがトリガされます。

**条件式の値が変更**

最後の評価時以降に式の値が変化した場合に、ブレークポイントがトリガされます。

**スキップ数**

ブレークポイントがトリガを開始するまでにブレークポイント条件が真となる回数を指定します。この回数に達すると、条件が満たされるたびにブレークポイントがトリガされます。

### トリガ範囲

要求された範囲とトレースでカバーする有効範囲が表示されます。推奨される範囲は、**［ブレーク位置］**と**［サイズ］**オプションによって指定された領域とまったく同じか、その内側です。

**拡張して要求された範囲をカバー**

データ構造体がカバーされるようにブレークポイントを拡張します。ハードウェアブレークポイント装置で提供できるブレークポイント範囲のサイズと合わないデータ構造（たとえば 3 バイト）の場合、ブレークポイントの範囲はデータ構造全体を対象としません。ブレークポイントの範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

**［トリガ範囲］**オプションは、データブレークポイントをサポートするすべての C-SPY ハードウェアドライバで使用可能です。

### データ照合

アクセスされるデータの照合を有効にします。**［データ照合］**オプションとデータのアクセスタイプを組み合わせて使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

**値**

データ値を指定します。

**マスク**

値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。

Cortex-M の場合、データマスクは以下のいずれかの値に限定されます。

`0xFFFFFFFF` は、語句が完全一致しなければならないことを意味します。

`0xFFFF` は、一致項目がワードまたはハーフワードの上位、下位 16 ビット部分のどれでもいいことを示します。

`0xFF` は、一致項目がワード、ハーフワード、またはバイトの上位、中央、下位 8 ビット部分のどれでもいいことを示します。たとえば、データ `0xVV` の場合、`xxxxxxVV`、`xxxxVVxx`、`xxVVxxxx`、`VVxxxxxx` パターンに一致する 32 ビットアクセス、および `xxVV` または `VVxx` に一致する 16 ビットアクセス、正確に一致する 8 ビットアクセスによってブレークポイントがトリガされます。

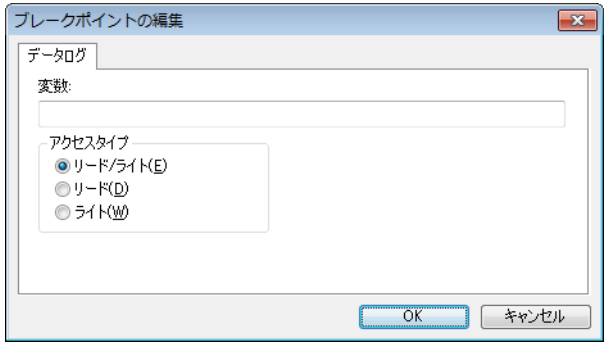
**［データ照合］**オプションは、I-jet、I-jet Trace、JTAGjet、J-Link/J-Trace と ST-LINK でのみ使用可能です（ARM7/9 または Cortex-M デバイスを使用中のみ）。

**注 :** Cortex-M デバイスについては、1 つのブレークポイントにのみ［データ照合］を設定できます。このようなブレークポイントでは、2 つのハードウェアブレークポイントを使用します。

**注 :［データ照合］**オプションは、Cortex-M0、Cortex-M1、Cortex-M0+ では使用できません。

# [データログ] ブレークポイントダイアログボックス

[データ] ブレークポイントダイアログボックスは、エディタウィンドウ、[ブレークポイント] ウィンドウ、[メモリ] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。




この図は C-SPY シミュレータを示します。

[データログ] ブレークポイントダイアログボックスを使用して、静的記憶寿命を持つ整数型の変数に最大 4 つのデータログブレークポイントを設定します。マイクロコントローラは、1 つの命令によるメモリアクセスで変数にアクセスできる必要があります。つまり、データログブレークポイントを設定できるのは、8 ビット、16 ビット、32 ビットの変数のみです。

135 ページの *データログブレークポイント*および 105 ページの *データログを開始するには*を参照してください。

## 要件

C-SPY シミュレータ。

## 変数

アクセスを記録する変数を指定します。

## アクセスタイプ

ログエントリを生成せる変数へのアクセスのタイプを選択します。

**リード / ライト**

変数のアドレスからのリード / ライトアクセス、または変数のアドレスに対するリード / ライトアクセス。

**リード**

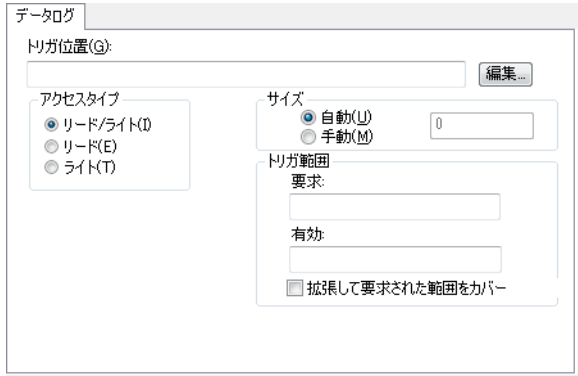
変数のアドレスからのリードアクセス。

**ライト**

変数のアドレスへのライトアクセス。

## [データログ] ブレークポイントダイアログボックス（C-SPY ハードウェアドライバ）

**[データログ]** ブレークポイントダイアログボックスは、エディタウィンドウ、[ブレークポイント] ウィンドウ、[メモリ] ウィンドウのコンテキストメニューから、および [逆アセンブル] ウィンドウで使用できます。




**[データログ]** ブレークポイントダイアログボックスを使用して、最高 4 つのデータログブレークポイントを設定できます。

データログブレークポイントは、8 ビット、16 ビット、32 ビットの変数に設定できます。

135 ページの *データログブレークポイント*、105 ページの *データログを開始するには* も参照してください。

### 要件

以下のいずれかが必要です。

- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link//J-Trace ドライバ
- C-SPY ST-LINK ドライバ

### トリガ位置

ブレークポイントのデータ位置を指定します。または、**[編集]** ボタンをクリックして **[位置入力]** ダイアログボックスを表示します（168 ページの *[位置入力] ダイアログボックス* を参照）。

### アクセスタイプ

ブレークポイントをトリガするメモリアクセスの種類を選択します。

**リード / ライト**
: 指定された位置から読み取り / 書き込みを行います。

**リード**
: 指定位置からのリード。ただし Cortex-M3 の場合は、レビジョン 1 以外のデバイスで機能します。

**ライト**
: 指定位置でのライト。ただし Cortex-M3 の場合は、レビジョン 1 以外のデバイスで機能します。

### サイズ

ブレークポイントがトリガされる位置にサイズ（特に範囲）があるべきかを指定します。指定したメモリ範囲に対してフェッチアクセスが発生するごとに、ブレークポイントがトリガされます。サイズの指定方法を選択します。

**自動**
: ブレークポイントが設定されている式のタイプに基づいて、サイズが自動的に決まります。たとえば、ブレークポイントを 12 バイトの構造体に設定すると、そのブレークポイントのサイズは 12 バイトになります。

**手動**
: テキストボックスでブレークポイント範囲のサイズを指定します。

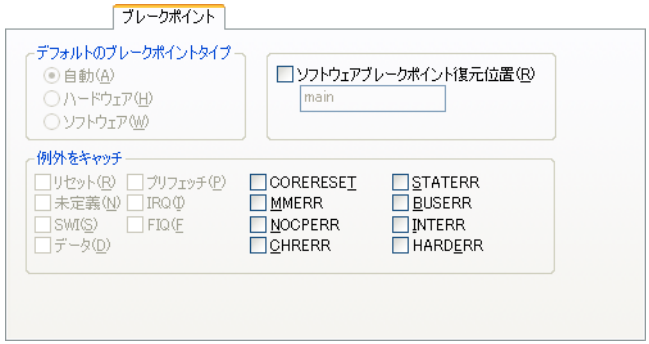
### トリガ範囲

要求された範囲とトレースでカバーする有効範囲が表示されます。推奨される範囲は、［**トリガ位置**］と［**サイズ**］オプションによって指定された領域とまったく同じか、その内側です。

**拡張して要求された範囲をカバー**
: データ構造体がカバーされるようにブレークポイントを拡張します。ハードウェアブレークポイント装置で提供できるブレークポイント範囲のサイズと合わないデータ構造（たとえば 3 バイト）の場合、ブレークポイントの範囲はデータ構造全体を対象としません。ブレークポイントの範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

# ［ブレークポイント］ダイアログボックス

［**ブレークポイント**］オプションページは、［**オプション**］ダイアログボックスから使用できます。［**プロジェクト**］>［**オプション**］を選択して、使用するデバッガシステムに固有のカテゴリを選び、［**ブレークポイント**］タブをクリックします。




このダイアログボックスを使用して、ドライバ固有のブレークポイントオプションを設定します。

## 要件

以下のいずれかが必要です。

- C-SPY CMSIS-DAP ドライバ
- C-SPY GDB サーバドライバ
- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY Macraigor ドライバ

## デフォルトのブレークポイントタイプ

ブレークポイントの設定時に使用するブレークポイントリソースの種類を選択します。以下から選択します。

| | |
|---|---|
| **自動** | ソフトウェアブレークポイントを使用します。不可能な場合は、ハードウェアブレークポイントが使用されます。RAM のテストにはリード / ライトシーケンスを使用します。この場合は、ソフトウェアブレークポイントが使用されます。［自動］オプションは多くのアプリケーションに有効です。ただし、実行されたリード / ライトシーケンスによってフラッシュメモリが誤動作をする場合があります。この場合、［**ハードウェア**］オプションを使用します。 |
| **ハードウェア** | ハードウェアブレークポイントを使用します。不可能な場合は、ブレークポイントは設定されません。 |
| **ソフトウェア** | ソフトウェアブレークポイントを使用します。不可能な場合は、ブレークポイントは設定されません。 |

### ソフトウェアブレークポイント復元位置

システム起動中に破壊されたブレークポイントを自動的に復元します。

起動中に RAM にコピーしてから RAM で実行しているアプリケーションの場合に有効です。たとえば、リンカ構成ファイルのコードに `initialize by copy` リンカディレクティブを使用する場合、あるいはアプリケーションに `__ramfunc` 宣言関数がある場合に有効となることがあります。

この場合、C-SPY デバッガが起動すると、すべてのブレークポイントは RAM のコピー中に破棄されます。C-SPY では、［**ソフトウェアブレークポイント復元位置**］オプションを使用して、破棄されたブレークポイントを復元します。

このテキストフィールドでは、C-SPY がブレークポイントを復元する地点の、アプリケーションの位置を指定します。デフォルトの位置は、ラベル `_call_main` です。

### 例外をキャッチ

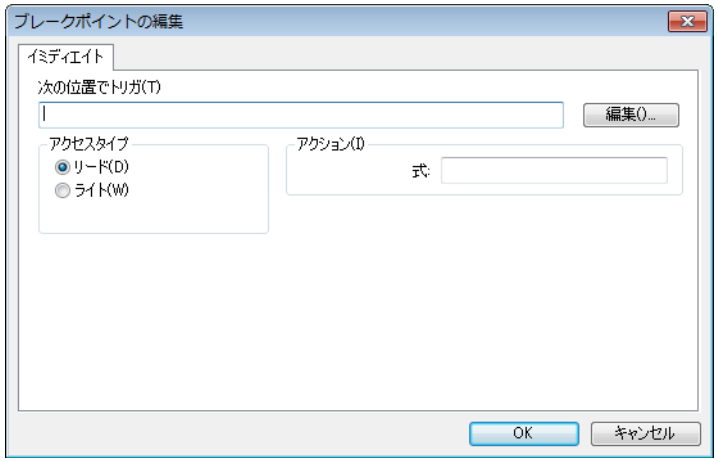
ハードウェアのブレークポイントを使用せずに、割込みベクタテーブルのベクタにブレークポイントを直接設定します。このオプションは、ARM9、Cortex-R4、Cortex-M3 のデバイスで使用可能です。この設定は、プロジェクトのデフォルト設定として機能します。ただし、これらのデフォルト設定は、デバッグセッション中に［**ベクタキャッチ**］ダイアログボックスを使用してオーバライドできます（143 ページの *例外ベクタ上へのブレークポイントの設定*を参照）。

これらの設定は、デバッグセッション中は保持されます。

このオプションは、C-SPY I-jet/JTAGjet ドライバおよび C-SPY J-Link/J-Trace ドライバでサポートされています。

## [イミディエイト] ブレークポイントダイアログボックス

[イミディエイト] ブレークポイントダイアログボックスは、エディタウィンドウ、[ブレークポイント] ウィンドウ、[メモリ] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。




C-SPY シミュレータで、[イミディエイト] ブレークポイントダイアログボックスを使用してイミディエイトブレークポイントを設定します。イミディエイトブレークポイントは一時的に命令の実行を停止するだけで、すぐに実行を再開します。

### 要件

C-SPY シミュレータ。

### トリガ位置

ブレークポイントのデータ位置を指定します。または、[編集] ボタンをクリックして [位置入力] ダイアログボックスを表示します（168 ページの *[位置入力] ダイアログボックス*を参照）。

### アクセスタイプ

ブレークポイントをトリガするメモリアクセスの種類を選択します。

**リード**

指定された位置から読み取ります。

**ライト**

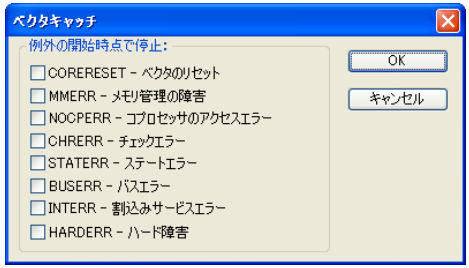
指定の位置に書き込みます。

### アクション

有効な C-SPY 式を指定します。この式は、ブレークポイントのトリガ時に条件が真であるときに評価されます。詳細については、144 ページの*ブレークポイントのヒント*を参照してください。

## [ベクタキャッチ] ダイアログボックス

**[ベクタキャッチ]** ダイアログボックスは、C-SPY ドライバのメニューから使用できます。




このダイアログボックスを使用すると、ハードウェアのブレークポイントを使用せずに、割込みベクタテーブルのベクタにブレークポイントを直接設定できます。ARM9、Cortex-R4、Cortex-M3 の各デバイスでベクタにブレークポイントを設定できます。ここでの設定はデバッグセッションが終了した後は保持されません。

次の図は Cortex-M デバイスを示します。別のデバイスを使用する場合は、このダイアログボックスの内容が異なることがあります。

**注 :** C-SPY I-jet/JTAGjet ドライバ、C-SPY J-Link/J-Trace ドライバ、C-SPY RDI ドライバの場合、オプションのダイアログボックスにすでにあるベクタにブレークポイントを直接設定することも可能です（556 ページの *J-Link/J-Trace の設定オプション*および 565 ページの *RDI* を参照）。

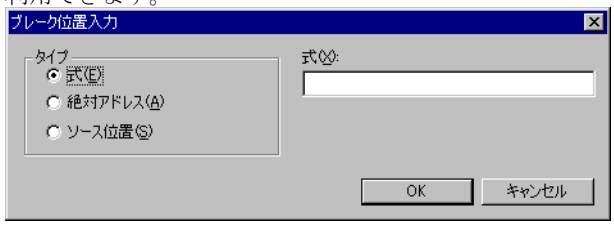
### 要件

以下のいずれかが必要です。

- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY CMSIS-DAP ドライバ
- C-SPY Macraigor ドライバ

## [位置入力] ダイアログボックス

[位置入力] ダイアログボックスは、新しいブレークポイントを設定するかブレークポイントを編集するときにブレークポイントダイアログボックスから利用できます。




[位置入力] ダイアログボックスを使用して、ブレークポイントの位置を指定します。

**注 :** このダイアログボックスは、選択する [タイプ] に応じて外観が変わります。

### タイプ

ブレークポイントで使用する位置のタイプを以下から選択します。

**式**

C-SPY 式。値は有効なコードやデータ位置に評価されます。

コード位置。たとえば関数 `main` は通常、コードブレークポイントに使用されます。

データ位置は変数名で、通常はデータブレークポイントに使用されます。たとえば、`my_var` は変数 `my_var` の位置を、`arr[3]` は配列 `arr` の 4 番目のエレメントの位置をそれぞれ参照します。いくつかの関数で同じ名前で宣言された静的変数については、特定の変数を参照するために構文 `my_func::my_static_variable` を使用してください。

C-SPY 式の詳細については、100 ページの *C-SPY 式*を参照してください。

**絶対アドレス**

フォーム *[ゾーン:hexaddress]* 上、または単に *hexaddress* (`Memory:0x42` など) 上の絶対アドレス。*ゾーン*は C-SPY のメモリゾーンを参照し、アドレスがどのメモリに属するかを指定します (174 ページの *C-SPY メモリゾーン*を参照)。

### ソース位置

次の構文を使用して示した C ソースコード内の位置。
{*filename*}.*row*.*column*.

*filename* には、ファイル名およびフルパスを指定します。

*row* には、ブレークポイントを設定する行を指定します。

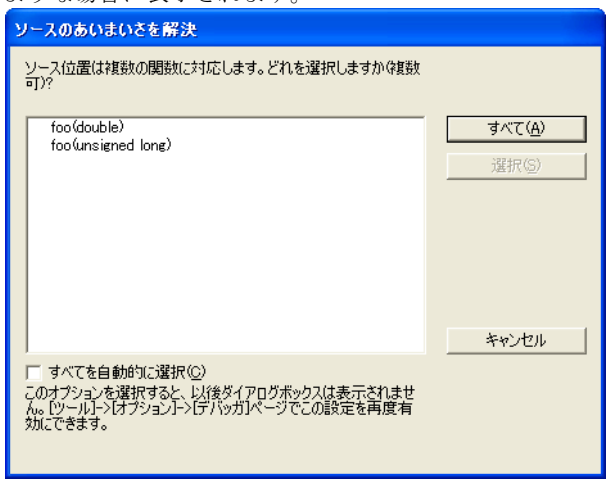
*column* には、ブレークポイントを設定する列を指定します。

たとえば、{C:¥*src*¥prog.c}.22.3
は、ソースファイル prog.c の 22 行目の 3 文字目の位置にブレークポイントを設定します。C-SPY マクロなどの引用符が付いた形式では、代わりに {C:¥¥*src*¥¥prog.c}.22.3 と記述する必要があります。

［ソース位置］のタイプは通常、コードブレークポイントのコード位置だけに使用します。使用する C-SPY ドライバによっては、データブレークポイントおよびイミディエイトブレークポイントで**［ソース位置］**が使用できないことがあります。

## ［ソースの曖昧さの解決］ダイアログボックス

［**ソースの曖昧さの解決**］ダイアログボックスは、たとえばテンプレート上にブレークポイントを設定しようとして、ソース位置が複数の関数に対応するような場合に表示されます。




ソースの曖昧さを解決するには、次のアクションのいずれかを実行します。

- テキストボックスで、リストされた位置を選択して（複数可）［**選択**］をクリックします。
- ［**すべて**］をクリックします。

**すべて**

ブレークポイントは、リストされたすべての位置に設定されます。

**選択**

ブレークポイントは、テキストボックスで選択したソース位置に設定されます。

**キャンセル**

どの場所も使用されません。

**自動的にすべて選択**

指定されたソース位置が複数の関数と一致する場合、すべての場所が使用されます。

このオプションは、［**IDE オプション**］ダイアログボックスでも使用できます（*『ARM 用 IDE プロジェクト管理およびビルドガイド』*のデバッガオプションを参照）。

# メモリとレジスタ

- メモリとレジスタのモニタの概要
- メモリとレジスタのモニタ
- メモリとレジスタについてのリファレンス情報

## メモリとレジスタのモニタの概要

以下のトピックについて説明します:

- メモリとレジスタのモニタの概要について
- C-SPY メモリゾーン
- スタック表示
- メモリアクセスチェック

### メモリとレジスタのモニタの概要について

C-SPY には、メモリとレジスタをモニタするウィンドウが多数あり、それらは個々に［**表示**］メニューから表示できます。

- ［メモリ］ウィンドウ

  指定メモリエリアであるメモリゾーンの最新状態を表示して、編集できます。様々な色を使用して、アプリケーションの実行に伴うデータカバレッジを示します。指定エリアに特定の値を設定して、メモリ位置と範囲に直接ブレークポイントを設定できます。このウィンドウで数個のインスタンスを開き、様々なメモリエリアをモニタできます。ウィドウの内容は、アプリケーションの実行中に定期的に更新されます。

- ［シンボルメモリ］ウィンドウ

  静的記憶寿命変数がメモリ内でどのように配置されるかを表示します。これにより、メモリの使用が理解し易くなり、バッファオーバランなど上書きされた変数に起因して発生した問題の調査に役立ちます。

- ［スタック］ウィンドウ

  メモリ内でのスタック変数の配置を含むスタック内容を表示します。また、スタックの整合性チェックを実行し、スタックオーバフローを検出してワーニングすることもできます。たとえば、［スタック］ウィンドウを使用して、スタックの最適サイズを特定できます。このウィンドウのインスタンスを最高 2 つ開いて、それぞれに異なるスタックを表示したり、同じスタックを異なる表示モードで表示したりできます。

- ［レジスタ］ウィンドウ

  プロセッサレジスタと SFR の内容の最新状態を表示し、それを編集できます。固定グループの CPU レジスタを除いて、追加のレジスタはデバイス記述ファイルで定義します。これらのレジスタとてしては、ARM デバイスの周辺ユニットに対する、メモリにマッピングされたデバイス固有の制御レジスタとステータスレジスタがあります。レジスタは、メモリにマッピングされた周辺ユニットレジスタや CPU レジスタなどが多数存在するため、［レジスタ］ウィンドウに同時にすべてのレジスタを表示するのは不便です。その場合は、レジスタを*レジスタグループ*に分割する方法があります。定義済みのレジスタグループをロードするか、アプリケーションに固有な独自のグループを定義することもできます。このウィンドウのインスタンスを複数開いて、それぞれに異なるレジスタグループを表示することができます。

- ［SFR 設定］ウィンドウ

  C-SPY が情報を持っている、現在定義された SFR を表示します。必要があれば、このウィンドウを使用して SFR の詳細をカスタマイズすることができます。

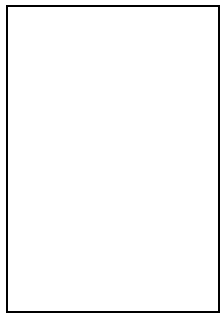
特定の変数の内容を表示するには、単純にその変数を [Memory]（メモリ）ウィンドウまたは [Symbolic memory]（シンボルメモリ）ウィンドウにドラッグします。変数が配置されているメモリエリアが表示されます。

一部のレジスタの値を読み取ると、アプリケーションの実行時の動作に影響することがあります。たとえば、UART ステータスレジスタの値を読み取ると、保持ビットがリセットされて、受け取ったバイトを処理するはずの割込みがない状態になることがあります。こうならないようにするには、実行中のアプリケーションをデバッグする際には、このようなレジスタを含む［レジスタ］ウィンドウを必ず閉じてください。

## C-SPY メモリゾーン

C-SPY では、*ゾーン*は名前付きメモリエリアを表します。メモリアドレス、または*メモリアドレス（ロケーション）*は、ゾーンとそのゾーン内のオフ

セット値の組合せです。ARM アーキテクチャには、ARM メモリ範囲全体をカバーするメモリというゾーンが 1 つだけあります。




デフォルトゾーンメモリ

メモリゾーンはさまざまなコンテキストで使用されますが、［メモリ］ウィンドウと［逆アセンブリ］ウィンドウ、C-SPY マクロで最も重要な役割を果たします。これらのウィンドウにある［**ゾーン**］ボックスを使用して、表示するメモリゾーンを選択します。

一般的なメモリの場合、デフォルトゾーンメモリを使用できます。しかし、正しい結果を得るには、いくつかの I/O レジスタには 8、16、32 または 64 ビットとしてのアクセスが必要な場合があります。さまざまなメモリゾーンを使用することで、［メモリ］ウィンドウなどの読込み / 書込みに使用するアクセス幅を制御することができます。ゾーン `Memory` を使用する場合、デバッガは最も適したアクセス幅を自動的に選択します。

**注 :** C-SPY I-jet/JTAGjet ドライバの場合、アクセス幅の自動選択を［**メモリ範囲の編集**］ダイアログボックスで指定できます（207 ページの *［メモリ範囲の編集］ダイアログボックス、C-SPY ハードウェアデバッガドライバを参照*）。

## スタック表示

［スタック］ウィンドウにはグラフィカルスタックバーがあり、スタックの内容とオーバフローのワーニングが表示されます。これらは多くの場面で役に立ちます。以下に例を示します。

- C モジュールからアセンブラモジュールを呼び出すか、その逆のときに、スタック使用量を調べる場合
- 適切なエレメントがスタック上に配置されているかどうかを調べる場合
- スタックが正しくリストアされているかどうかを調べる場合
- 最適なスタックサイズの判定
- スタックオーバフローの検出

複数のスタックを持つコアの場合は、表示するスタックを選択できます。

### スタックの使用量

アプリケーションを最初にロードするときや、リセットごとに、スタックエリアのメモリに 0xCD というバイト値が設定され、その後でアプリケーションの実行が開始されます。実行が停止すると、スタックの最後から、値が 0xCD でないバイトの位置（スタック中で使用された最も上の位置）までの範囲のスタックメモリが検索されます。これはスタック使用率のトレース方法としては信頼性の高い方法ですが、スタックオーバフローが検出されるという保証はありません。たとえば、スタックが範囲を超えて誤って拡張され、スタック上限近辺のバイトは変更されることなく、スタックエリア外のメモリが変更される*可能性があります*。同様に、アプリケーションがスタックエリア内のメモリを誤って修正する可能性もあります。

［スタック］ウィンドウで検出できるのはスタックオーバフローの痕跡だけで、スタックオーバフローを発生時点で検出することはできません。ただし、グラフィカルスタックバーを有効にすると、スタックオーバフローの検出とワーニングに必要な機能も有効になります。

**注：**スタックのサイズと場所は、リンカ設定ファイルで作成されるスタックを保持するセクションの定義から読み込まれます。何らかの理由で、システム起動コード (cstartup) によるスタック初期化を変更する場合は、その変更に応じて、リンカ設定ファイルのセクション定義も変更する必要があります。これを行わないと、［スタック］ウィンドウでスタック使用率を追跡できません。これらの詳*『ARM 用 IAR C/C++ 開発ガイド』*を参照してください。

### メモリアクセスチェック

C-SPY シミュレータは、ターゲットハードウェアのさまざまなメモリアクセスタイプをシミュレーションして、ライト禁止のメモリへのリードアクセスなど、無効なアクセスを検出できます。特定のメモリエリアに対して指定されたアクセスタイプに従わないメモリアクセスが発生した場合、C-SPY はそれを不正なアクセスと認識します。また、定義されていないメモリへのメモリアクセスは、不正なアクセスと見なされます。メモリアクセスチェック機能によって、ユーザはメモリアクセス違反を特定しやすくなります。

メモリエリアは、デバイス記述ファイルで定義済のゾーンか、デバッグファイルのセクション情報に基づいています。これら以外に、ユーザが独自のメモリエリアを定義できます。アクセスタイプには、読み取り、書き込み、読み取のみ、書き込みのみがあります。2 つの異なるアクセスタイプを同一のメモリエリアに割り当てることはできません。アクセスタイプの違反および未指定の範囲へのアクセスをチェックすることができます。違反が検出された場合は、［デバッグログ］ウィンドウにロギングされます。実行を停止するかどうかを選択することもできます。

# メモリとレジスタのモニタ

以下のタスクについて説明します:

- 177 ページの*使用するデバイスのメモリに合わせた C-SPY の設定*
- 178 ページの*アプリケーション固有のレジスタグループの定義*

## 使用するデバイスのメモリに合わせた C-SPY の設定

通常、プロジェクトを設定する場合は、特定デバイスのデバイス記述ファイルが自動または手動で選択されます。そのファイルでデバイスのメモリ範囲情報が完全に指定されている場合は、この点において C-SPY を設定する必要はありません。

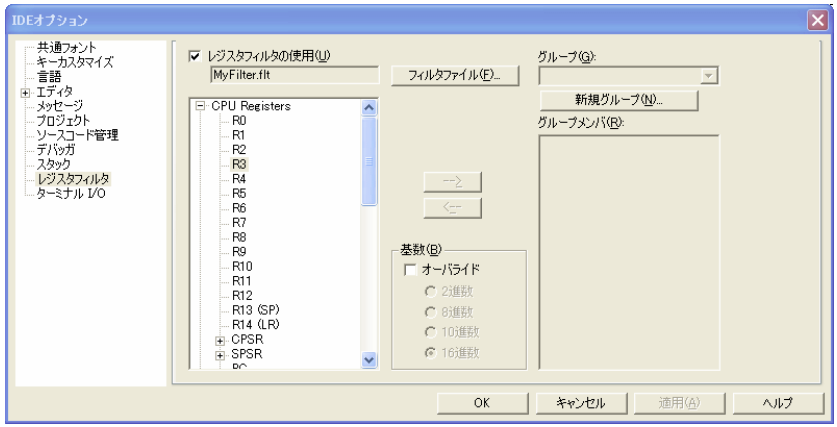
ただし、そのファイルで特定のデバイスに対してメモリ範囲が指定されておらず、デバイスのグループについて指定されている場合は（異なるサイズのオンチップ RAM を持つものなど）、デバイスに合わせてエリアを調整する必要があります。

使用するデバイスに合わせてメモリエリアを微調整するには、200 ページの*［メモリ構成］ダイアログボックス、C-SPY シミュレータ*および 204 ページの*［メモリ構成］ダイアログボックス、C-SPY ハードウェアデバッガドライバ*を参照してください。

### アプリケーション固有のレジスタグループの定義

アプリケーションに固有のレジスタグループを定義すると、［レジスタ］ウィンドウに表示されるレジスタの量を最小限に抑えて、デバッグを高速化できます。

1 デバッグセッション中に［ツール］>［オプション］>［レジスタフィルタ］を選択します。




レジスタフィルタのオプションについては、*『ARM 用 IDE プロジェクト管理およびビルドガイド』*を参照してください。

2 ［レジスタフィルタの使用］を使用して、表示されるダイアログボックスの新しいグループに、フィルタファイルのファイル名と出力先を指定します。

3 ［新規グループ］をクリックして、「My Timer Group」というようにグループ名を指定します。


4 ［レジスタフィルタ］ページのレジスタツリービューで、レジスタを選択して矢印ボタンをクリックし、グループにレジスタを追加します。グループに追加するすべてのレジスタについて、このプロセスを繰り返します。

5 オプションで、整数基数を変更するレジスタを選び、適切な基数を選択します。

6 終わったら、**[OK]** をクリックします。新しいグループが［レジスタ］ウィンドウで使用できるようになりました。

フィルタファイルにグループをさらに追加するには、追加する各グループに対してこの手順を繰り返してください。

**注 :** レジスタのリストに表示されるレジスタは、現在使用している ddf ファイルから読み込まれます。必要な特定の SFR が表示されない場合は、自分の SFR を登録できます。詳細については、195 ページの *［SFR 設定］ウィンドウ*を参照してください。

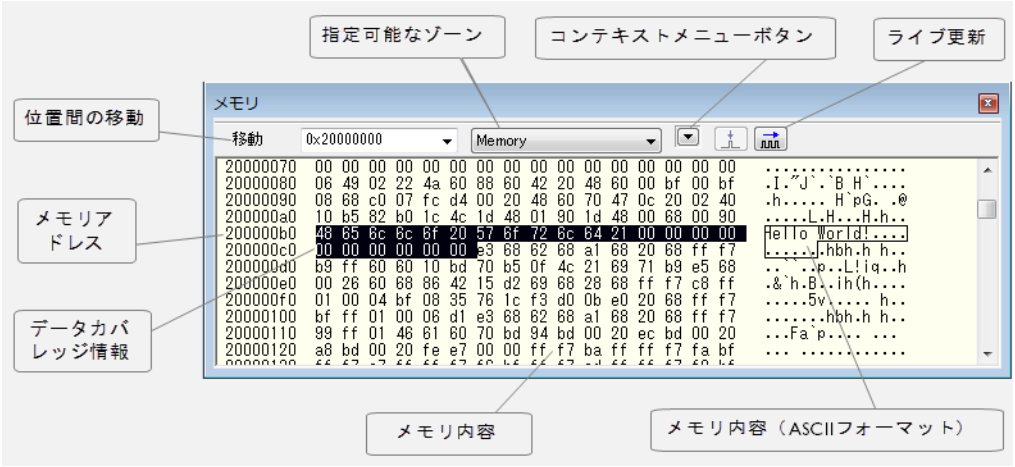
## メモリとレジスタについてのリファレンス情報

リファレンス情報 :


- 180 ページの *［メモリ］ウィンドウ*
- 184 ページの *［メモリセーブ］ダイアログボックス*
- 185 ページの *［メモリリストア］ダイアログボックス*
- 186 ページの *［フィル］ダイアログボックス*
- 187 ページの *［シンボルメモリ］ウィンドウ*
- 190 ページの *［スタック］ウィンドウ*
- 193 ページの *［レジスタ］ウィンドウ*
- 195 ページの *［SFR 設定］ウィンドウ*
- 199 ページの *［SFR の編集］ダイアログボックス*
- 200 ページの *［メモリ構成］ダイアログボックス、C-SPY シミュレータ*
- 203 ページの *［メモリ範囲の編集］ダイアログボックス、C-SPY シミュレータ*
- 204 ページの *［メモリ構成］ダイアログボックス、C-SPY ハードウェアデバッガドライバ*
- 207 ページの *［メモリ範囲の編集］ダイアログボックス、C-SPY ハードウェアデバッガドライバ*
- 210 ページの *［メモリアクセス設定］ダイアログボックス*
- 212 ページの *［メモリアクセスの編集］ダイアログボックス*

# ［メモリ］ウィンドウ

［メモリ］ウィンドウは［**表示**］メニューから利用できます。




このウィンドウでは、指定メモリエリアであるメモリゾーンの最新状態を表示して、編集できます。このウィンドウは複数表示でき、メモリやレジスタの複数のゾーンをトレースする場合や、メモリのさまざまな部分をモニタする場合に非常に便利です。

変数に対応するメモリを表示するには、エディタウィンドウでその変数を選択し、［メモリ］ウィンドウにドラッグします。

C-SPY のウィンドウにおける編集について詳しくは、65 ページの *C-SPY デバッガメインウィンドウ*を参照してください。

## 要件

ありません。このウィンドウは常に使用できます。

## ツールバー

ツールバーの内容は以下のとおりです。

**移動**

表示するメモリアドレス（ロケーション）またはシンボルを指定できます。

**ゾーン**

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

**コンテキストメニューボタン**

コンテキストメニューを表示します。

**今すぐ更新**

アプリケーションの実行中に［メモリ］ウィンドウの内容を更新します。このボタンは、使用している C-SPY ドライバがアプリケーションの実行中にターゲットのシステムメモリにアクセス可能な場合のみ有効化されます。

**ライブ更新**

アプリケーションの実行中に［メモリ］ウィンドウの内容を定期的に更新します。このボタンは、使用している C-SPY ドライバがアプリケーションの実行中にターゲットのシステムメモリにアクセス可能な場合のみ有効化されます。更新頻度を設定するには、**［IDE オプション］>［デバッガ］**ダイアログボックスに適切な頻度を指定します。

## 表示エリア

表示エリアには現在表示しているアドレスとメモリの内容が選択したフォーマットで表示されるほか、表示モードが **1x ユニット**に設定されていれば、メモリの内容が ASCII フォーマットで表示されます。表示エリアの内容は、16 進表示と ASCII 表示のどちらの部分でも編集できます。

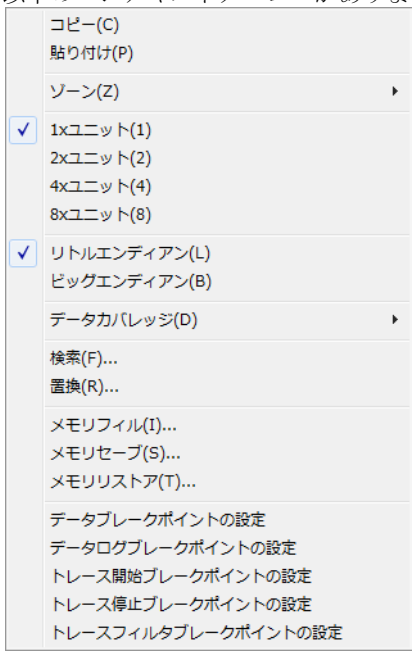
データカバレッジは、以下の色で表示されます。

| | |
|---|---|
| 黄色 | データのリードが実行されたことを示します。 |
| 青 | データのライトが実行されたことを示します。 |
| 緑 | データのリードとライトの両方が実行されたことを示します。 |

**注 :** 一部の C-SPY ドライバは、データカバレッジをサポートしていません。C-SPY シミュレータは、データカバレッジをサポートしています。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**コピー、貼り付け**

標準の編集コマンド

**ゾーン**

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

**1x ユニット**

メモリの内容を単一のバイトで表示します。

**2x ユニット**

メモリの内容を 2 バイトのグループで表示します。

**4x ユニット**

メモリの内容を 4 バイトのグループで表示します。

**8x ユニット**

メモリの内容を 8 バイトのグループで表示します。

**リトルエンディアン**

リトルエンディアンのバイトオーダで内容を表示します。

**ビッグエンディアン**

ビッグエンディアンのバイトオーダで内容を表示します。

**データカバレッジ**

以下から選択します。

**［有効化］**は、データカバレッジの有効 / 無効を切り替えます。

**［表示］**は、データカバレッジの表示 / 非表示を切り替えます。

**［クリア］**は、すべてのデータカバレッジ情報を消去します。

これらのコマンドは、C-SPY ドライバがデータカバレッジをサポートする場合のみ使用できます。

**検索**

［メモリ］ウィンドウ内でテキストを検索するダイアログボックスを表示します。**［検索］**ダイアログボックスについて詳しくは、*『ARM 用 IDE プロジェクト管理およびビルドガイド』*を参照してください。

**置換**

指定した文字列を検索して、該当する項目を別の文字列に置換するダイアログボックスを表示します。**［置換］**ダイアログボックスについては、*『ARM 用 IDE プロジェクト管理およびビルドガイド』*を参照してください。

**メモリフィル**

指定エリアに値を設定できるダイアログボックスを開きます（186 ページの*［フィル］ダイアログボックス*を参照）。

**メモリセーブ**

特定のメモリエリアの内容をファイルに保存できるダイアログボックスを表示します（184 ページの*［メモリセーブ］ダイアログボックス*を参照）。

**メモリリストア**

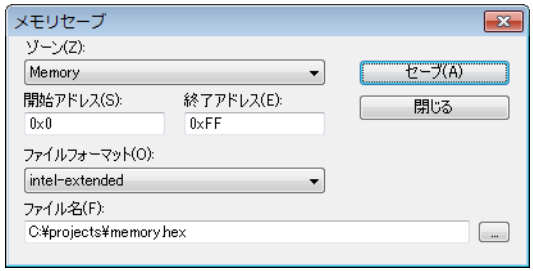
Intel-hex や Motorola s-record フォーマットでファイルの内容を指定したメモリゾーンにロードできるダイアログボックスを表示します（185 ページの*［メモリリストア］ダイアログボックス*を参照）。

### データブレークポイントの設定

［メモリ］ウィンドウでブレークポイントを直接設定します。ブレークポイントが強調表示されていない場合は、［ブレークポイント］ダイアログボックスでブレークポイントを表示、編集、削除することができます。このウィンドウで設定したブレークポイントは、リードとライトの両方のアクセスでトリガされます。詳細については、141 ページの*［メモリ］ウィンドウでのデータブレークポイントの設定*を参照してください。

## ［メモリセーブ］ダイアログボックス

**［メモリセーブ］**ダイアログボックスは、**［デバッグ］**>**［メモリ］**>**［保存］**を選択するか、［メモリ］ウィンドウのコンテキストメニューから使用できます。




このダイアログボックスを使用して、指定したメモリエリアの内容をファイルに保存します。

### 要件

ありません。このダイアログボックスは常に使用できます。

### ゾーン

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

### 開始アドレス

保存するメモリ範囲の開始アドレスを指定します。

### 終了アドレス

保存するメモリ範囲の終了アドレスを指定します。

**ファイルフォーマット**

使用するファイルフォーマットを選択します。デフォルトでは Intel-extended です。

**ファイル名**

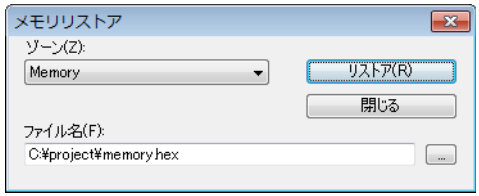
使用する対象ファイルを指定します。参照ボタンを使用して選択すると便利です。

**セーブ**

選択したメモリゾーン範囲を指定ファイルに保存します。

## ［メモリリストア］ダイアログボックス

**［メモリリストア］**ダイアログボックスは、**［デバッグ］** > **［メモリ］** > **［元に戻す］**を選択するか、［メモリ］ウィンドウのコンテキストメニューから使用できます。




このダイアログボックスを使用して、特定のメモリゾーンにファイルの内容を Intel-extended または Motorola s-record フォーマットでロードします。

**要件**

ありません。このダイアログボックスは常に使用できます。

**ゾーン**

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

**ファイル名**

リード対象ファイルを指定します。参照ボタンを使用して選択すると便利です。

**元に戻す**

指定ファイルの内容を選択したメモリゾーンにロードします。

## ［フィル］ダイアログボックス

［フィル］ダイアログボックスは、［メモリ］ウィンドウのコンテキストメニューから使用できます。


このダイアログボックスを使用して、指定したメモリエリアに値を設定できます。

### 要件

ありません。このダイアログボックスは常に使用できます。

### 開始アドレス

2 進数、8 進数、10 進数、16 進数のいずれかの表記法で開始アドレスを入力します。

### 長さ

2 進数、8 進数、10 進数、16 進数のいずれかの表記法でデータ長を入力します。

### ゾーン

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

### 値

各メモリアドレス（ロケーション）に設定する 8 ビット値を入力します。

### 操作

以下のメモリ操作を使用できます。

**COPY**

［値］に入力した値が指定したメモリエリアにコピーされます。

**AND**

［AND］を指定すると、［値］の値とメモリの既存値の論理積がメモリに書き込まれます。

**XOR**

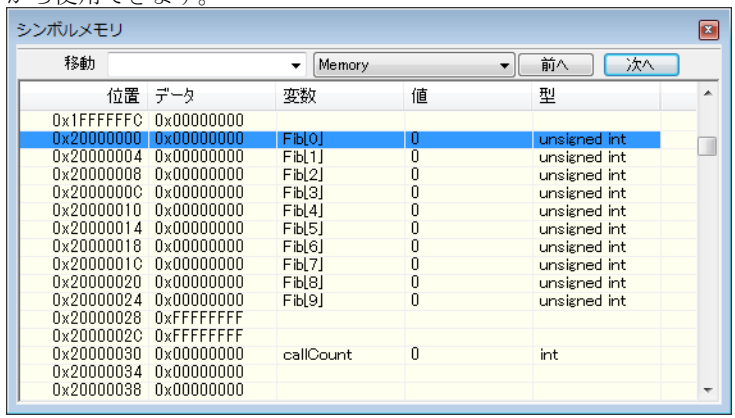
[XOR] を指定すると、［値］の値とメモリの既存値の論理積がメモリに書き込まれます。

**OR**

[OR] を指定すると、［値］の値とメモリの既存値の論理積がメモリに書き込まれます。

## ［シンボルメモリ］ウィンドウ

［シンボルメモリ］ウィンドウは、デバッグセッション中に［**表示**］メニューから使用できます。




このウィンドウには、静的記憶寿命を持つ変数や、通常はファイルスコープのほかに静的変数を関数およびクラスに持つ変数が、メモリ内でどのように配置されるかが表示されます。これにより、メモリの使用が理解し易くなり、バッファオーバランなど上書きされた変数に起因して発生した問題の調査に役立ちます。アラインメントの穴の検出や上書きしたバッファに起因する問題の理解にも役立ちます。

変数に対応するメモリを表示するには、エディタウィンドウでその変数を選択し、［シンボルメモリ］ウィンドウにドラッグします。

C-SPY のウィンドウにおける編集について詳しくは、65 ページの *C-SPY デバッグメインウィンドウ*を参照してください。

### 要件

ありません。このウィンドウは常に使用できます。

### ツールバー

ツールバーの内容は以下のとおりです。

**移動**

表示するメモリアドレス（ロケーション）またはシンボルを指定できます。

**ゾーン**

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

**前へ**

表示エリアで前のシンボルを強調表示します。

**次へ**

表示エリアで次のシンボルを強調表示します。

### 表示エリア

このエリアには以下の列が含まれます。

**位置**

メモリアドレス。

**データ**

16 進フォーマットのメモリ内容。データはシンボルサイズに従ってグループ化されます。この列は編集できます。

**変数**

変数名。変数には固定されたメモリ位置が必要です。ローカル変数は表示されません。

**値**

変数の値。この列は編集できます。

**タイプ**

変数の型。

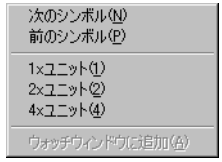
いくつかの方法でメモリ空間内を移動できます。

- ウィンドウでドロップされるテキストがシンボルと解釈されます
- ウィンドウ右側のスクロールバー
- ツールバーボタン［**次へ**］と［**前のエラー**］
- ツールバーリストボックス［**移動**］は、特定の場所やシンボルの検出に使用できます

**注：**対応する値を変更すると行に赤色のマークが付けられます。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**次のシンボル**

表示エリアで次のシンボルを強調表示します。

**前のシンボル**

表示エリアで前のシンボルを強調表示します。

**1x ユニット**

メモリの内容を単一のバイトで表示します。これは変数を含まない行にのみ適用します。

**2x ユニット**

メモリの内容を 2 バイトのグループで表示します。

**4x ユニット**

メモリの内容を 4 バイトのグループで表示します。

**［ウォッチ］ウィンドウに追加**

選択したシンボルを［ウォッチ］ウィンドウに追加します。

## ［スタック］ウィンドウ

［スタック］ウィンドウは［**表示**］メニューから利用できます。


このウィンドウには、スタックの内容が表示されます。また、スタックの整合性チェックを実行し、スタックオーバフローを検出してワーニングすることもできます。たとえば、［スタック］ウィンドウを使用して、スタックの最適サイズを特定できます。

このウィンドウでは、リンカ設定ファイルに実施されたスタックを保持するセクションの定義からスタックのサイズと場所に関する情報を取得します。このセクションは『*ARM 用 IAR C/C++ 開発ガイド*』に記述されています。

スタックを他の方法で設定するアプリケーションに対しては、デフォルトの方法をオーバライドできます。C-SPY コマンドラインの派生オプションのいずれかを使用します（515 ページの *--proc_stack_stack* を参照）。

**グラフィカルスタックバーを表示するには、以下の手順に従います。**

1 ［**ツール**］>［**オプション**］>［**スタック**］を選択します。

2 オプション［**グラフィカルスタック表示とスタック使用トラッキングを有効にする**］を選択します。

［スタック］ウィンドウを 2 つまで表示し、それぞれ異なるスタックを表示する（スタックが複数ある場合）か、同じスタックを異なる表示設定値で表示することができます。

**注 :** デフォルトでは、このウィンドウは物理的ブレークポイントを 1 つ使用します。詳細については、137 ページの *ブレークポイントの設定元*を参照してください。

［スタック］ウィンドウに固有の情報については、*『ARM 用 IDE プロジェクト管理およびビルドガイド』*を参照してください。

### 要件

ありません。このウィンドウは常に使用できます。

### ツールバー

ツールバーの内容は以下のとおりです。

**スタック**

表示するスタックを選択します。これは複数のスタックを持つコアに適用されます。

### グラフィカルスタックバー

スタックの状態をグラフィックを使用して表示します。

スタックバーの左端はスタックの底、つまりスタックが空白のときのスタックポインタの位置を示します。右端は、スタック用に予約されているメモリエリアの最後を示します。スタック使用率が指定のしきい値を超えると、スタックバーは赤色に変化します。

スタックバーを有効にすると、スタックオーバフローの検出とワーニングに必要な機能も有効になります。

マウスポインタをスタックバーの上に移動すると、スタック使用量に関するツールチップ情報が表示されます。

### 表示エリア

このエリアには以下の列が含まれます。

**位置**

メモリでの位置を示します。アドレスは小さい方から順に表示されます。スタックポインタが参照するアドレス、つまりスタック最上部は、緑色で強調表示されます。

**データ**

その位置のメモリユニットの内容を表示します。［スタック］ウィンドウのコンテキストメニューで、データの表示方法（1 バイト、2 バイト、4 バイトのいずれかの単位）を選択できます。

**変数**

その位置にローカル変数がある場合に、その変数名を表示します。変数は、関数内でローカルに宣言されていて、レジスタではなくスタックにある場合にだけ表示されます。

**値**

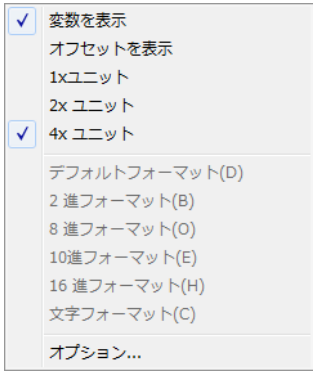
［**変数**］列に表示されている変数の値を示します。

**フレーム**

呼出しフレームが対応する関数の名前を示します。

## コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**変数を表示**

**変数**、**値**、**フレーム**という別々の列を［スタック］ウィンドウに表示します。［スタック］ウィンドウで表示されるメモリ位置にある変数が、これらの列に表示されます。

**オフセットを表示**

［**位置**］列にスタックポインタからのオフセットとして場所を表示します。選択を解除すると、位置は絶対アドレスとして表示されます。

**1x ユニット**

メモリの内容を単一のバイトで表示します。

**2x ユニット**

メモリの内容を 2 バイトのグループで表示します。

**4x ユニット**

メモリの内容を 4 バイトのグループで表示します。

**デフォルトフォーマット**
**2 進フォーマット**
**8 進フォーマット**
**10 進フォーマット**
**16 進フォーマット**
**文字フォーマット**

式の表示フォーマットを変更します。表示フォーマット設定は、式の種類によって適用対象が異なります。表示フォーマットの選択は、デバッグセッションの終了後も保持されます。ウィンドウで選択された行に変数が含まれる場合、これらのコマンドが使用できます。

表示フォーマット設定は、式の種類によって以下のように適用対象が異なります。

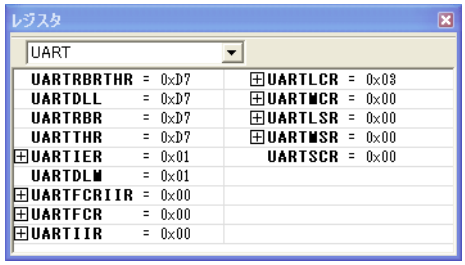
| | |
|---|---|
| 変数 | 表示設定は、選択した変数だけに適用されます。他の変数には適用されません。 |
| 配列エレメント | 表示設定は配列全体に適用されます。つまり、配列の各エレメントに同一の表示フォーマットが使用されます。 |
| 構造体のフィールド | 定義が同一のエレメント（フィールド名、C の宣言型）に表示設定が適用されます。 |

**オプション**

**［IDE オプション］**ダイアログボックスを表示します。このダイアログボックスで、［スタック］ウィンドウ専用オプションを設定できます（*『ARM 用 IDE プロジェクト管理およびビルドガイド』*を参照）。

## ［レジスタ］ウィンドウ

［レジスタ］ウィンドウは**［表示］**メニューから利用できます。




このウィンドウには、プロセッサレジスタおよび特別な関数レジスタの最新の内容が表示され、一部のレジスタの内容を編集することができます。オプ

ションで、定義済みのレジスタグループをロードするか、アプリケーションに固有な独自のグループを定義することもできます。

このウィンドウは複数表示でき、複数のレジスタグループをトレースするのに非常に便利です。

C-SPY のウィンドウにおける編集について詳しくは、65 ページの *C-SPY デバッガメインウィンドウ*を参照してください。

**定義済レジスタグループを有効にするには、次の手順に従います。**

1 デバイスに適合するデバイス記述ファイルを選択します（55 ページの*デバイス記述ファイルの選択*を参照）。

2 デバイス記述ファイルで定義されていれば、レジスタグループは［レジスタ］ウィンドウに表示されます。使用可能なレジスタグループは、**［レジスタフィルタ］**ページにも一覧表示されます。

アプリケーション固有のレジスタグループを定義するには、次の手順に従います。

178 ページの*アプリケーション固有のレジスタグループの定義*を参照してください。

### 要件

ありません。このウィンドウは常に使用できます。

### ツールバー

ツールバーの内容は以下のとおりです。

**CPU レジスタ**

表示するレジスタグループを選択します。デフォルトは CPU レジスタです。デフォルトでは、デバッガには次の 2 つのレジスタグループがあります。SFR の一部が足りない場合、カスタムグループに独自の SFR を登録することができます（195 ページの*［SFR 設定］ウィンドウ*を参照）。

**［現在の CPU レジスタ］**には、現在のプロセッサモードで使用可能なレジスタが含まれます。

**［CPU レジスタ］**には、現在のレジスタと、他のプロセッサモードで使用可能なバンクレジスタの両方が含まれます。

追加のレジスタグループは、デバイス記述ファイル（`arm¥config` ディレクトリ）に定義されます。これらによって、すべての SFR レジスタが［レジスタ］ウィンドウで使用可能になります。デバイス記述ファイルには、特殊機能レジスタとそのグループを定義するセクションがあります。

### 表示エリア

レジスタとその値を表示します。C-SPY が停止するたびに、前回停止したときから変更された値が強調表示されます。レジスタにはリード専用のものと、ライト専用のもの（W で示されます）、編集可能なものがあります。編集可能なレジスタの内容を編集するには、クリックして値を変更します。新しい値をキャンセルするには、[Esc] を押してください。

一部のレジスタは展開可能です。つまり、関連するビットやビットのサブグループが含まれています。

表示フォーマットを変更するには、**［レジスタフィルタ］**ページで **［ベース］**設定を変更します。このページにアクセスするには、**［ツール］** > **［オプション］**を選択します。

C-SPY シミュレータおよび C-SPY ハードウェアデバッガドライバの場合、これらの追加サポートレジスタは CPU レジスタグループにあります。

| | |
|---|---|
| **CYCLECOUNTER** | アプリケーションの起動またはリセット時にクリアされ、実行中に使用済みのサイクル数だけ増分します。 |
| **CCSTEP** | 前回実行された C/C++ ソースまたはアセンブラステップ中に使用されたサイクル数を表示します。 |
| **CCTIMER1** および **CCTIMER2** | いつでも手動でクリアすることができる 2 つの *トリップカウント*。実行中に使用済みのサイクル数だけ増分されます。 |

## ［SFR 設定］ウィンドウ

［SFR 設定］ウィンドウは **［プロジェクト］**メニューから利用できます。


このウィンドウには、C-SPY が情報を持っている現在定義された SFR が表示されます。出荷時の SFR とカスタム定義された SFR の両方、またはどちらか一方を表示するよう選択できます。必要があれば、このウィンドウを使用して SFR の詳細をカスタマイズすることができます。出荷時の SFR（現在使用

中の ddf ファイルから取得したもの）については、アクセスタイプをカスタマイズすることしかできません。

カスタム定義された SFR は、「カスタム」という専用のレジスタグループに追加されます。このグループは［レジスタ］ウィンドウに表示することができます。カスタム定義された SFR は、*project*CustomSFR.sfr に保存されます。

C-SPY デバッガが実行中でないときは、SFR の追加または変更以外はできません。

### 要件

ありません。このウィンドウは常に使用できます。

### 表示エリア

このエリアには以下の列が含まれます。

**ステータス**

SFR のステータスを示す文字で、以下のいずれかを使用できます。

空白 : 出荷時の SFR

C: 変更された出荷時の SFR

+: カスタム定義された SFR

?: 何らかの理由で無視される SFR。出荷時の SFR が変更されていると SFR は無視できますが、SFR は使用できなくなるか、どこか別の場所にあるか、あるいはサイズが異なります。通常、デバイスの変更時にそうなるときがあります。

**名前**

SFR の一意の名前。

**アドレス**

SFR のメモリアドレス。

**ゾーン**

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

**サイズ**

レジスタのサイズ（8、16、32、64 のいずれか）。

**アクセス**

レジスタのアクセスタイプ。リード / ライト、リードオンリー、ライトオンリー、なしのいずれかです。

値を変更するには、名前かアドレスをクリックします。アドレスの 16 進数 0x のプレフィックスは省略可能です。省略しても、入力した値は 16 進数として解釈されます。たとえば、4567 と入力すると、0x4567 となります。

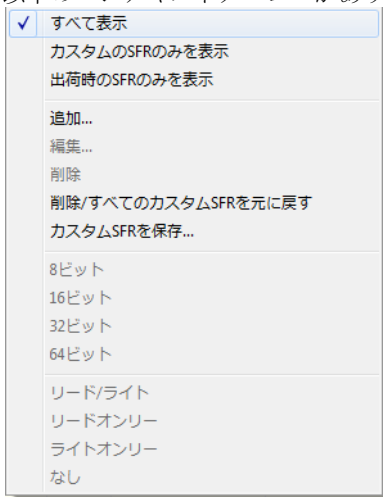
列のプロパティに従って SFR をソートするには、列のヘッダをクリックします。

表示エリアで使用される色分け :

- 緑は対応する値が変わったことを示します。
- 赤は SFR が無視されたことを示します。

## コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**すべて表示**

すべての SFR を表示します。

**カスタムの SFR のみを表示**

カスタム定義されたすべての SFR を表示します。

**出荷時の SFR のみを表示**

ddf ファイルから取得された出荷時の SFR をすべて表示します。

**追加**

［**SFR の編集**］ダイアログボックスが表示され、ここで新しい SFR を追加できます（199 ページの *［SFR の編集］ダイアログボックス*を参照）。

**編集**

［**SFR の編集**］ダイアログボックスが表示され、ここで SFR を編集できます（199 ページの *［SFR の編集］ダイアログボックス*を参照）。

**削除**

SFR を削除します。このコマンドは、カスタム定義された SFR についてのみ機能します。

**削除 / すべてのカスタム SFR を元に戻す**

カスタム定義されたすべての SFR を削除して、変更された出荷時の SFR を元の設定に戻します。

**カスタム SFR を保存**

標準の保存ダイアログボックスを開いて、カスタム定義されたすべての SFR を保存します。

**8|16|32|64 ビット**

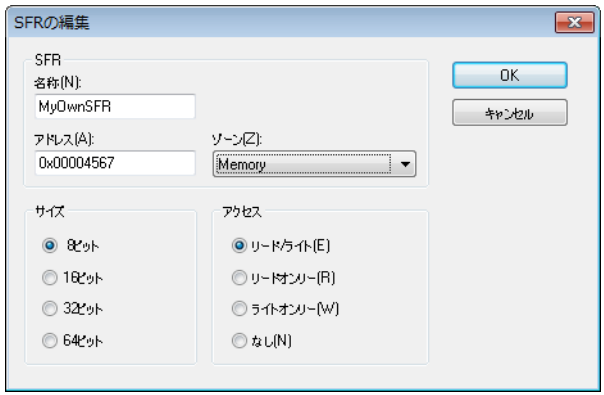
選択した SFR の表示フォーマットを 8 ビット、16 ビット、32 ビット、64 ビットから選択します。表示フォーマットを変更できるのは、カスタム定義された SFR のみです。

**リード / ライト | リードオンリー | ライトオンリー | なし**

選択した SFR のアクセスタイプを選択します。**リード / ライト**、**リードオンリー**、**ライトオンリー**または**なし**のいずれかを使用できます。出荷時の SFR の場合、デフォルトのアクセスタイプが指定されます。

## ［SFR の編集］ダイアログボックス

［**SFR の編集**］ダイアログボックスは、［SFR の設定］ウィンドウから使用できます。




このダイアログボックスを使用して、SFR を定義します。

### 要件

ありません。このダイアログボックスは常に使用できます。

### 名前

追加または編集する SFR 名を指定します。

### アドレス

追加または編集する SFR のアドレスを指定します。アドレスの 16 進数 `0x` のプレフィックスは省略可能です。省略しても、入力した値は 16 進数として解釈されます。たとえば、`4567` と入力すると、`0x4567` となります。

### ゾーン

追加または編集する SFR のゾーンを選択します。ゾーンのリストは、現在使用している `ddf` ファイルから読み込まれます。

### サイズ

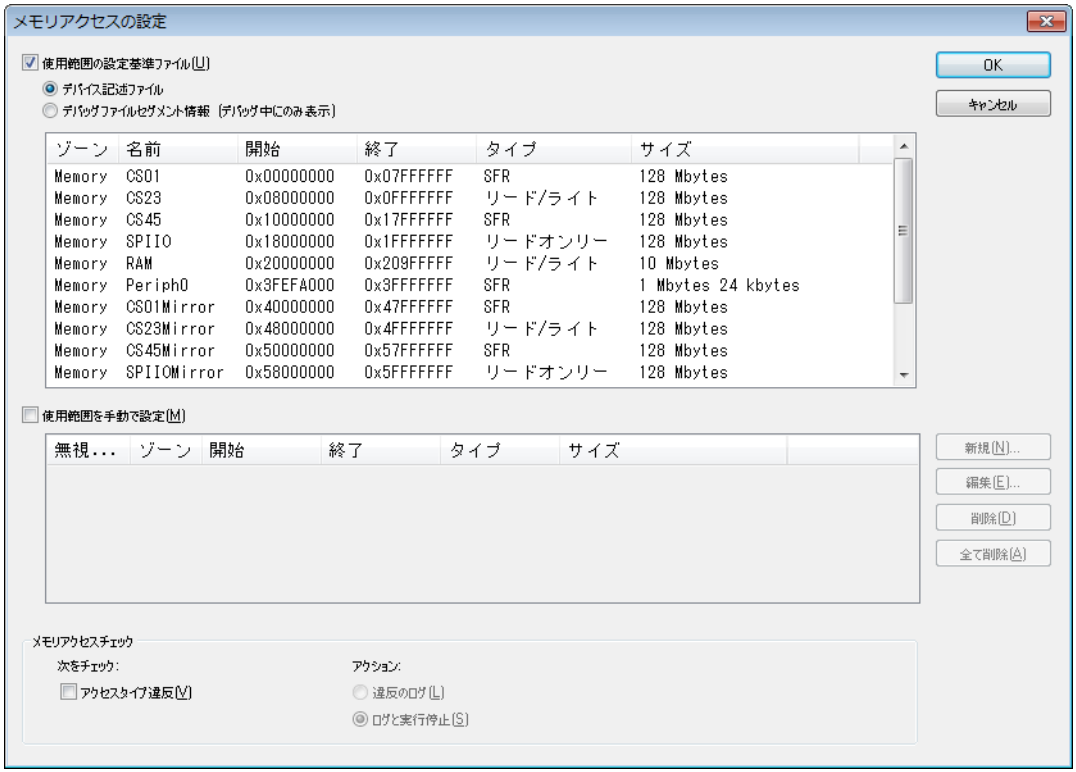
SFR のサイズを選択します。8 ビット、16 ビット、32 ビット、64 ビットから選択してください。表示フォーマットを変更できるのは、カスタム定義された SFR のみです。

**アクセス**

SFR のアクセスタイプを選択します。**リード / ライト**、**リードオンリー**、**ライトオンリー**、**なし**から選択します。出荷時の SFR の場合、デフォルトのアクセスタイプが指定されます。

## ［メモリ構成］ダイアログボックス、C-SPY シミュレータ

**［メモリ構成］**ダイアログボックスは、C-SPY ドライバメニューから使用できます。




デバッグ中にメモリを最大限効率的に処理するため、C-SPY はメモリ構成についての情報を必要とします。デフォルトでは、C-SPY は選択したデバイス記述ファイルから取得した情報に基づいて初期構成を使用します。

使用する特定のデバイスに対してそのファイルでメモリ範囲が指定されておらず、デバイスのグループ（異なるサイズのオンチップ RAM を持つものな

ど）について指定されている場合は、このダイアログボックスを使用してデバイスで使用可能なメモリに合わせて、メモリエリアを追加してください。

デフォルトでは、シミュレータは定義されたメモリエリア外のアクセスを許可しません。また、このダイアログボックスを使用して、アクセスタイプが正しいかどうかを C-SPY でチェックすることもできます。この場合、リードオンリーのメモリにライトアクセスがあれば、C-SPY はワーニングを表示します。

### 要件

C-SPY シミュレータ。

### 使用範囲の設定基準ファイル

メモリ構成を事前に定義済みの構成から読み込むかどうかを指定します。以下から選択します。

**デバイス記述ファイル**

指定したデバイス記述ファイルからメモリ構成を読み込みます。55 ページの*デバイス記述ファイルの選択*を参照してください。

このオプションはデフォルトで使用されます。

**デバッグファイルセグメント情報**

デバッグファイル（リンカ設定ファイルから読み込まれたもの）からメモリ構成ファイルを読み込みます。この情報はデバッグセッション中にのみ使用可能です。このオプションの利点は、シミュレータがリンクされたアプリケーション外のメモリアクセスを把握できることです。

メモリ情報は以下の列に表示されます。

**ゾーン**

メモリゾーン（174 ページの *C-SPY メモリゾーン*を参照）。

**名前**

メモリエリア名。

**開始**

メモリエリアの開始アドレスを 16 進表記で指定します。

**終了**

メモリエリアの終了アドレスを 16 進表記で指定します。

**タイプ**

メモリエリアのアクセスタイプ。

**サイズ**

メモリエリアのサイズ。

**使用範囲を手動で設定**

**[メモリ範囲の編集]** ダイアログボックスで独自の範囲を手動で指定します。このダイアログボックスを開くには、**[新規]** をクリックして新しいメモリ範囲を指定するか、メモリ範囲を選択して **[編集]** をクリックし、範囲を変更します。詳細については、203 ページの *[メモリ範囲の編集] ダイアログボックス、C-SPY シミュレータ*を参照してください。

手動で定義した範囲は、デバッグセッション終了後も保持されます。

**[無視]** の列に x があると、手動で指定した範囲が無効であることを示します (別の範囲と重複している場合など)。このような範囲は使用されません。

**メモリアクセスチェック**

**[チェック]** により、チェックの対象が決まります。

- アクセスタイプの違反。

**[アクション]** では、アクセス違反が発生したときに実行するアクションを以下から選択します。

- 違反のログ
- ログと実行停止

違反が検出された場合は、[デバッグログ] ウィンドウにロギングされます。

**ボタン**

これらのボタンは手動で指定した範囲で使用できます。

**新規**

**[メモリ範囲の編集]** ダイアログボックスが開き、ここで新しいメモリ範囲を指定してそれにアクセスタイプを関連付けることができます (203 ページの *[メモリ範囲の編集] ダイアログボックス、C-SPY シミュレータ*を参照 )。

**編集**

**[メモリ範囲の編集]** ダイアログボックスを開きます。選択されたメモリエリアを編集できます。203 ページの *[メモリ範囲の編集] ダイアログボックス、C-SPY シミュレータ*を参照してください。

**削除**

選択されたメモリエリア定義を削除します。

**全て削除**

定義されているすべてのメモリエリア定義を削除します。

## ［メモリ範囲の編集］ダイアログボックス、C-SPY シミュレータ

**［メモリ範囲の編集］**ダイアログボックスは、**［メモリ構成］**ダイアログボックスから使用できます。


このダイアログボックスを使用して、メモリエリアとそのアクセスタイプを指定します。

関連項目 200 ページの *［メモリ構成］ダイアログボックス、C-SPY シミュレータ*。

### 要件

C-SPY シミュレータ。

### メモリ範囲

デバイスに固有のメモリエリアを定義します。

**ゾーン**

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

**開始アドレス**

メモリエリアの開始アドレスを 16 進表記で指定します。

**終了アドレス**

メモリエリアの終了アドレスを 16 進表記で指定します。

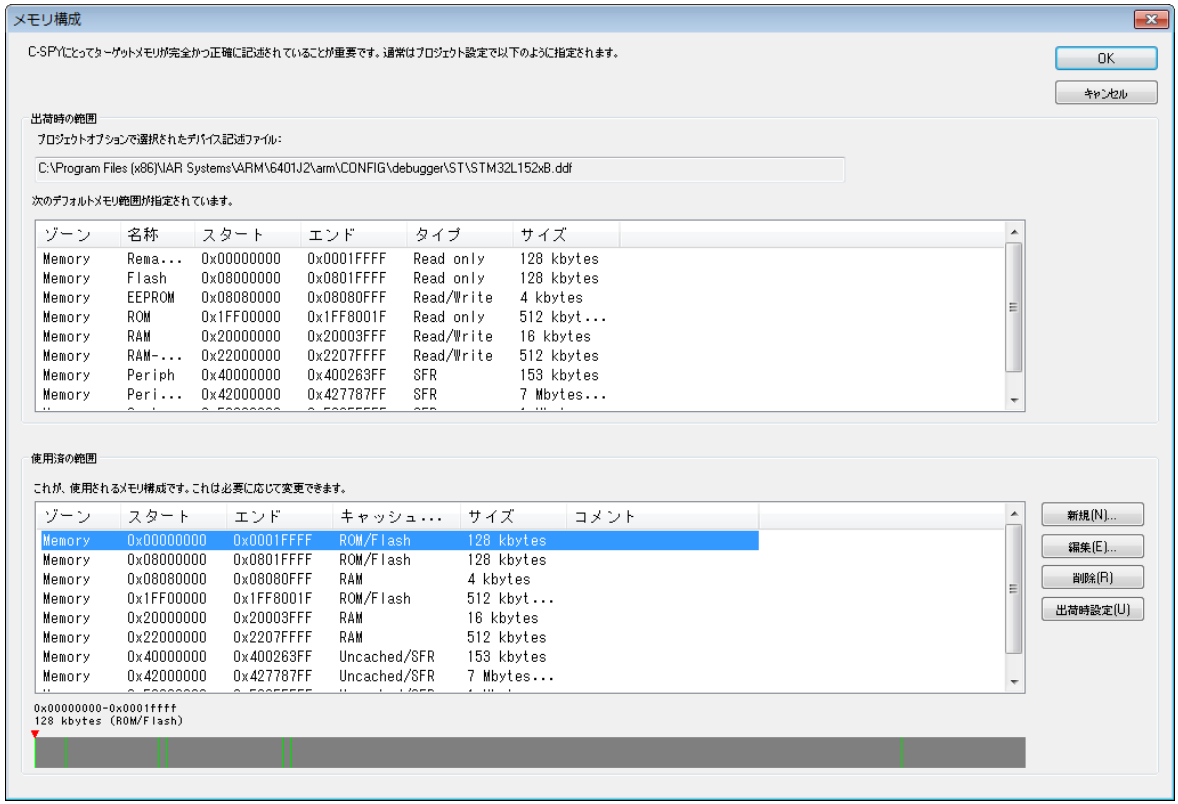
**アクセスタイプ**

メモリ範囲のアクセスタイプを以下から選択します。

- **[RAM]**（リード / ライトメモリ）
- **[ROM/Flash]**（リードオンリーメモリ）
- **[SFR]**（SFR リード / ライトメモリ）

## ［メモリ構成］ダイアログボックス、C-SPY ハードウェアデバッガドライバ

［メモリ構成］ダイアログボックスは、C-SPY ドライバのメニューから使用できます。




C-SPY は、選択したデバイス記述ファイルから読み込まれた情報に基づいてデフォルトのメモリ構成を使用します。メモリ構成がデバイス記述ファイル

にない場合は、使用可能な出荷時のデフォルトを提供しようとします。
55 ページの*デバイス記述ファイルの選択*を参照してください。

このダイアログボックスを使用して、メモリエリアを検証し、必要があればデバイスで使用可能なメモリに合わせて変更します。C-SPY にターゲットシステムのメモリレイアウトに関する情報を提供すると、パフォーマンスと機能の面で役立ちます。

- メモリのリード（およびライト）が高速になりますが（デバッグプローブが USB ポートなどを介して接続されている場合）、C-SPY で多くのデバッガウィンドウを更新しなければならないときには、足かせとなりがちです。メモリをキャッシュするとパフォーマンスは高速になりますが、そうすると C-SPY はターゲットメモリについての情報を必要とします。
- デバッグセッション中に特定のメモリエリアの内容が変更されることが C-SPY に伝えられると、C-SPY は、通常はターゲットが読取りを許可しない場合でも（実行時など）、そのメモリのコピーを読取り可能に維持することができます。
- C-SPY は、まったくメモリのないエリアへのアクセスを防止できます。これは、特定のハードウェアにとって重要です。

あるプロジェクトで C-SPY ドライバを初めて起動したときに、**[メモリ構成]** ダイアログボックスが自動的に表示されます。ただし、デバイス記述ファイルにすでに適切で完全なメモリの記述が含まれている場合を除きます。別のデバイス記述ファイルを選択した場合など、メモリ構成の変更につながるプロジェクトの変更を行わない限り、その後の起動時にこのダイアログボックスは表示されません。

メモリ構成を変更できるのは、C-SPY を実行していないときのみです。

### 要件

以下のいずれかが必要です。

- C-SPY CMSIS-DAP ドライバ
- C-SPY I-jet/JTAGjet ドライバ

### 出荷時の範囲

現在選択されているデバイス記述ファイルを識別し、以下の列から取得されたデフォルトのメモリエリアの一覧を表示します。

**ゾーン**

メモリゾーン（174 ページの *C-SPY メモリゾーン*を参照）。

**名前**

メモリエリア名。

**開始**

メモリエリアの開始アドレスを 16 進表記で指定します。

**終了**

メモリエリアの終了アドレスを 16 進表記で指定します。

**タイプ**

メモリエリアのアクセスタイプ。

**サイズ**

メモリエリアのサイズ。

### 使用済の範囲

以下の列には、手動で指定したメモリエリアの一覧が表示されます。

**ゾーン**

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

**開始**

メモリエリアの開始アドレスを 16 進表記で指定します。

**終了**

メモリエリアの終了アドレスを 16 進表記で指定します。

**キャッシュタイプ**

メモリエリアのキャッシュタイプ。

**サイズ**

メモリエリアのサイズ。

**コメント**

メモリエリアの情報。

これらのボタンを使用して、デバイス記述ファイルから取得したデフォルトのメモリエリアをオーバライドします。

### グラフィカルバー

デフォルトバスの理論的なメモリ空間全体を視覚化するグラフィカルバー。定義されたエリアは緑で強調表示されます。

### ボタン

以下のボタンを選択できます。

**新規作成**

［**メモリ範囲の編集**］ダイアログボックスを開きます。新しいメモリエリアを指定したり、キャッシュタイプを割り当てたりすることができます（207 ページの *［メモリ範囲の編集］ダイアログボックス、C-SPY ハードウェアデバッガドライバ*を参照）。

**編集**

［**メモリ範囲の編集**］ダイアログボックスを開きます。選択されたメモリエリアを編集できます。207 ページの *［メモリ範囲の編集］ダイアログボックス、C-SPY ハードウェアデバッガドライバ*を参照してください。

**削除**

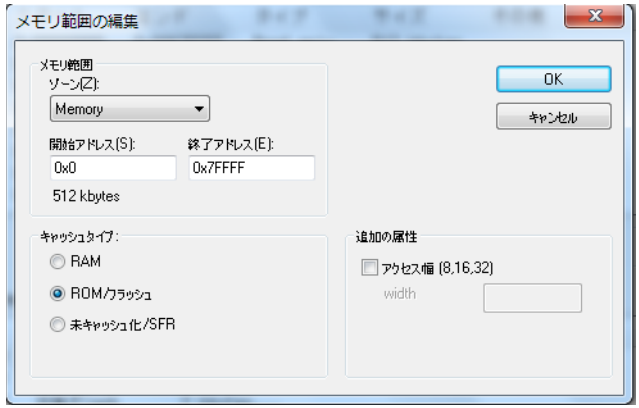
選択されたメモリエリア定義を削除します。

**出荷時の設定を使用**

選択したデバイス記述ファイルで指定されたメモリエリアを取得するか、デバイス記述ファイルにメモリの情報がない場合は、使用可能な出荷時のデフォルトを提供しようとします。

## ［メモリ範囲の編集］ダイアログボックス、C-SPY ハードウェアデバッガドライバ

［**メモリ範囲の編集**］ダイアログボックスは、［**メモリの構成**］ダイアログボックスから使用できます。




このダイアログボックスを使用してメモリエリアを指定し、それぞれのメモリ範囲にキャッシュタイプを割り当てます。

204 ページの*［メモリ構成］ダイアログボックス、C-SPY ハードウェアデバッガドライバ*も参照してください。

### 要件

以下のいずれかが必要です。

- C-SPY CMSIS-DAP ドライバ
- C-SPY I-jet/JTAGjet ドライバ

### メモリ範囲

デバイスに固有のメモリエリアを定義します。

**ゾーン**

メモリゾーンを選択します（174 ページの*C-SPY メモリゾーン*を参照）。

**開始アドレス**

メモリエリアの開始アドレスを 16 進表記で指定します。

**終了アドレス**

メモリエリアの終了アドレスを 16 進表記で指定します。

### キャッシュタイプ

メモリエリアのキャッシュタイプを以下から選択します。

**RAM**

ターゲット CPU が実行中でないときは、メモリからのすべてのリードアクセスはキャッシュにロードされます。たとえば、2 つの［メモリ］ウィンドウにメモリの同じ部分が表示される場合、実際のメモリは両方のウィンドウを更新するためにハードウェアから 1 度だけ読み込まれます。C-SPY のウィンドウからのメモリを変更すると、データはキャッシュのみに書き込まれます。ターゲットの実行前に、マシン命令を 1 つでもステップ実行する前に、変更されたすべてのバイトがハードウェア上のメモリに書き込まれるように、RAM キャッシュがフラッシュされます。

**ROM/ フラッシュ**

このメモリは、デバッグセッション中に変わらないことを想定しています。デバッグセッションの開始時にダウンロードされるこうした範囲にあるコード（あるい正確に言うならば、デバッグするアプリケーションの一部であるコード）は、キャッシュに格納されてそこで保持

されます。こうした範囲の他の部分は、必要に応じてメモリからキャッシュにロードされますが、デバッグセッション中は保持されます。また、C-SPY では C-SPY のウィンドウからのこうしたメモリを変更できません。

通常フラッシュメモリは固定されたリードオンリーのメモリとして使用されますが、実行時に記憶領域を変更するためにフラッシュメモリの一部を使用するアプリケーションがあります。たとえば、フラッシュメモリの一部をファイルシステムで使用したり、単に不揮発性の情報を格納するために使用することがあります。これを C-SPY で反映するには、フラッシュメモリのこれらの部分を 1 つまたは複数の RAM の範囲として指定する必要があります。こうすることで、C-SPY はこれらの部分が実行中にいつでも変更される可能性があると想定します。

**SFR/ 未キャッシュ化**

このタイプの範囲は完全に未キャッシュ化されます。C-SPY ウィンドウからのリードまたはライトコマンドはすべて、ハードウェアにアクセスします。通常このタイプは、リードアクセスのたびに値が異なり、以前に書き込まれたときと同じ値を持たないために、書き込まれる際に他のレジスタに影響を及ぼすことになるあらゆる種類の稀な動作を持つ特殊な関数レジスタに役立ちます。

デバイスについて正しい情報を持っていない場合は、メモリ全体を **SFR/ 未キャッシュ化**として指定できます。このは間違ってはいませんが、ウィンドウを更新するときに C-SPY の動作が遅くなることがあります。実際、メモリ範囲の情報が何もない場合に、この方法がデフォルトで推奨されることがあります。

### 追加の属性

追加の属性を提供します。

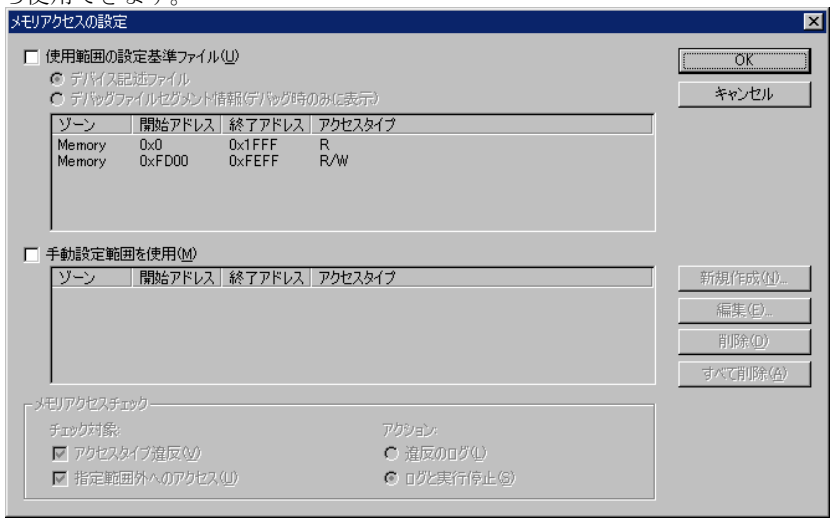
**アクセス幅 [8,16,32]**

8 ビット、16 ビット、32 ビットの範囲のメモリにアクセスする際、C-SPY で 8 ビット、16 ビット、32 ビットを強制的に使用します。テキストボックスで 8、16、32 を指定してください。

このオプションは、使用する C-SPY ドライバで利用できない場合があります。

## ［メモリアクセス設定］ダイアログボックス

**［メモリアクセス設定］**ダイアログボックスは、C-SPY ドライバのメニューから使用できます。




このダイアログボックスには定義済みのすべてのメモリエリアが一覧表示され、リストのそれぞれの列ではエリアのプロパティが指定されます。つまり、このダイアログボックスには、シミュレーション中に使用されるメモリアクセス設定が表示されます。

**注 :［使用範囲の設定基準ファイル］**オプションと**［使用範囲を手動で設定］**オプションを両方とも有効にすると、定義されているすべての範囲へのメモリアクセスがチェックされます。

表示される列とプロパティについては、212 ページの*［メモリアクセスの編集］ダイアログボックス*を参照してください。

### 要件

C-SPY シミュレータ。

### 使用範囲の設定基準ファイル

定義済メモリアクセス設定を選択します。以下から選択します。

**デバイス記述ファイル**

デバイス記述ファイルからプロパティをロードします。

**デバッグファイルセグメント情報**

プロパティは、デバッグファイルで使用できるセクション情報に基づいています。この情報は、デバッグ中のみ使用できます。このオプションの利点は、シミュレータが、リンクされているアプリケーション以外からのメモリアクセスをキャプチャできることです。

### 使用範囲を手動で設定

［**メモリアクセスの編集**］ダイアログボックスで独自の範囲を手動で指定します。このダイアログボックスを開くには、［**新規作成**］を選択して新しいメモリ範囲を指定するか、メモリゾーンを選択してから［**編集**］を選択してそのメモリゾーンを変更します。詳細については、212 ページの *［メモリアクセスの編集］ダイアログボックス*を参照してください。

手動で定義した範囲は、デバッグセッション終了後も保持されます。

### メモリアクセスチェック

［**チェック対象**］では、チェックの対象を指定します。

- アクセスタイプ違反
- 指定範囲外へのアクセス

［**アクション**］では、アクセス違反が発生したときに実行するアクションを以下から選択します。

- 違反のログ
- ログと実行停止

違反が検出された場合は、［デバッグログ］ウィンドウにロギングされます。

### ボタン

以下のボタンを選択できます。

**新規作成**

［**メモリアクセスの編集**］ダイアログボックスを開きます。新しいメモリ範囲を指定したり、アクセスタイプを割り当てたりすることができます（212 ページの *［メモリアクセスの編集］ダイアログボックス*を参照）。

**編集**

［**メモリアクセスの編集**］ダイアログボックスを開きます。選択されたメモリエリアを編集できます。212 ページの *［メモリアクセスの編集］ダイアログボックス*を参照してください。

**削除**

選択されたメモリエリア定義を削除します。

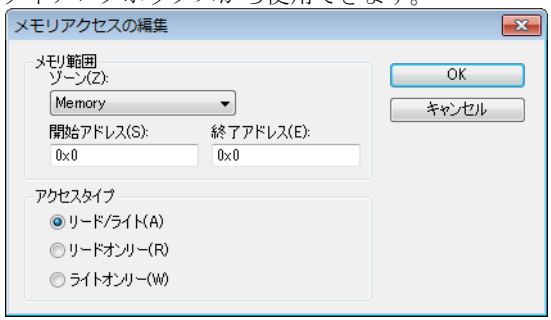
**全て削除**

定義されているすべてのメモリエリア定義を削除します。

[OK] ボタンと［キャンセル］ボタン以外のボタンは、**［使用範囲を手動で設定］**オプションが選択されているときだけ使用できます。

## ［メモリアクセスの編集］ダイアログボックス

**［メモリアクセスの編集］**ダイアログボックスは、**［メモリアクセスの設定］**ダイアログボックスから使用できます。




このダイアログボックスを使用してメモリ範囲を指定し、各メモリ範囲にアクセスタイプを割り当てます、これに対して、シミュレーション中に不正なアクセスを検出します。

### 要件

C-SPY シミュレータ。

### メモリ範囲

デバイスに固有のメモリエリアを定義します。

**ゾーン**

メモリゾーンを選択します（174 ページの *C-SPY メモリゾーン*を参照）。

**開始アドレス**

メモリエリアの開始アドレスを 16 進表記で指定します。

**終了アドレス**

メモリエリアの終了アドレスを 16 進表記で指定します。

**アクセスタイプ**

メモリ範囲へのアクセスタイプを以下から選択します。

- **リード / ライト**
- **リードオンリー**
- **ライト専用**

# パート 2. アプリケーションの解析

*ARM 用 C-SPY® デバッガガイド*のこのパートは、以下の章で構成されています。

- トレース
- プロファイリング
- コードカバレッジ
- Power デバッグ
- C-RUN ランタイムエラー解析

# トレース

- トレースの使用の概要
- トレースデータの収集と使用
- トレースについてのリファレンス情報

## トレースの使用の概要

以下のトピックを解説します。

- トレースを使用する理由
- トレースの概要
- トレースを使用するための条件

関連項目：

- 105 ページの*データログを開始するには*
- 107 ページの*イベントログを開始するには*
- 299 ページの *Power デバッグ*
- 377 ページの*割込みログを使用するにあたって*
- 279 ページの*プロファイリング*

### トレースを使用する理由

トレースを使用することで、特定の状態（アプリケーションのクラッシュなど）になるまでのプログラムの流れを調べたり、トレースデータを使用して問題の原因を特定したりすることができます。トレースデータは、不規則な症状が散発的に発生するようなプログラミングエラーを特定する際に役立ちます。

### トレーストリガとトレースフィルタを使用する理由

トレーストリガとトレースフィルタ条件を使用することで、ソースコードの関心を持つ部分を選択し、トレースプローブのトレースバッファをより効率的に使用できます。トレーストリガ（トレース開始ブレークポイントとトレース停止ブレークポイント）は、たとえばトレースデータを収集する対象のコードセクションを指定します。トレースフィルタは、実行中に満たされたときにトレースデータの収集を有効にする条件を指定します。

ARM7/9 デバイスの場合、合計で最大 16 のトレーストリガおよびトレースフィルタを指定でき、うち 8 つはトレースフィルタとすることができます。

Cortex-M デバイスの場合、合計で最大 4 つのトレーストリガおよびトレースフィルタを指定できます。

## トレースの概要

ターゲットシステムは、トレースデータを生成できる必要があります。データが生成されると、C-SPY でそれを収集し、さまざまなウィンドウやダイアログボックスでそのデータを視覚化および解析することができます。

C-SPY は、以下のターゲットシステムからのトレースデータの収集をサポートしています。

- ETM (Embedded Trace Macrocell) をサポートするデバイス — ETM トレース
- SWO (Serial Wire Output) 通信チャンネルを使用してSWD (Serial Wire Debug) インタフェースをサポートするデバイス — SWO トレース
- C-SPY シミュレータ

使用しているターゲットシステムに応じて、異なるタイプのトレースデータが生成できます。

## ETM トレース

ETM トレース（別名フルトレース）は、選択された実行の一部について実行されたすべての命令を連続して収集したシーケンスです。バッファが保持できるだけのデータしか収集できません。

デバッグプローブには、トレースデータをリアルタイムで収集するトレースバッファが含まれていますが、データは実行が停止するまで C-SPY のウィンドウに表示されません。

## ETB トレース

ETB トレース (Embedded Trace Buffer) は ETM トレースに似ていますが、主な違いはトレースデータがオンチップのトレースバッファに収集される点です。トレースバッファには指定のメモリエリアがあり、そのサイズは事前に定義されています。

## MTB トレース

MTB トレース (Micro Trace Buffer) は ETM トレースの簡易版で、オンチップのトレースバッファを使用します。MTB トレースの場合、トレースバッファはアプリケーションコードと RAM メモリを共有します。

MTB トレースは、Cortex-M0+ コアに基づいてデバイスの命令トレースへのアクセス権を付与します。

## SWO トレース

SWO トレースは、オンチップのデバッグハードウェアによって生成されるさまざまな種類のイベントのシーケンスです。イベントは SWO 通信チャンネルを介して、ターゲットシステムからリアルタイムで送信されます。つまり、C-SPY のウィンドウはターゲットシステムの実行中に連続して更新されます。最も重要なイベントは以下のとおりです。

- PC サンプリング

  ハードウェアは、プログラムカウンタの値を一定の間隔でサンプリングおよび送信することができます。これは（ETM トレースのような）実行済み命令の連続シーケンスではなく、PC の散発的かつ定期的なサンプリングです。現在の ARM CPU は通常、毎秒数百万の命令を実行しますが、PC サンプリングレートは一般的に毎秒数千の単位でカウントされます。

- 割込みログ

  ハードウェアは割込みの実行に関連するデータを生成して送信できます。割込みハンドラルーチンの投入および終了時にイベントが生成されます。

- データログ

  データログブレークポイントを使用して、ある変数または単にアドレス範囲が CPU によってアクセスされたときにイベントを生成して送信するようハードウェアを設定できます。

SWO チャンネルの処理量には制限があります。よって、PC サンプリング、割込み、指定した変数へのアクセスの頻度のいずれかが高い場合、通常は上記のすべての機能を同時に使用することはできません。

トレースプローブで SWO 通信チャンネルを使用する場合、データはトレースバッファで収集され、実行が停止した後に表示されます。

## C-SPY のトレース機能

C-SPY では、トレース関連のウィンドウとして［トレース］、［関数トレース］、［タイムライン］、［トレースを検索］が使用できます。C-SPY シミュレータでは、［トレース式］ウィンドウも使用できます。使用する C-SPY ドライバに応じて、さまざまなタイプのトレースブレークポイントやトリガを設定してトレースデータの収集を制御することができます。

C-SPY I-jet/JTAGjet ドライバ C-SPY J-Link/J-Trace ドライバまたは ST-LINK ドライバを使用する場合、［割込みログ］、［ログサマリの割込み］、［データログ］、［データログ一覧］などのウィンドウを使用できます。

デバッグ時には、**ETM** および **SWO** という 2 つのボタンが IDE のメインウィンドウに表示されます。これらのボタンのどれかが緑の場合、対応するトレースハードウェアがトレースデータを生成中であることを意味します。C-SPY のどの機能がトレースデータの生成を要求しているのかについて詳細なツールチップ情報を得るには、マウスのポインタをボタンにあわせてください。この情報は、多くの C-SPY 機能が現在トレースデータを使用しているために SWO 通信チャンネルがよくオーバフローする場合などに便利です。ボタンをクリックすると、対応する設定ダイアログボックスが開きます。

また、プロファイラやコードカバレッジ、命令プロファイリングなど、C-SPY の他のいくつかの機能でもトレースデータを使用します。

### トレースを使用するための条件

C-SPY のシミュレータはトレース関連の機能をサポートしており、特定の要件はありません。

**注 :** 使用するデバッグコンポーネントの特定のセット（ハードウェア、デバッグプローブ、C-SPY ドライバ）によって、サポートされる C-SPY のトレース機能が決まります。

### ETM トレースを使用するための条件

ETM トレースは一部の ARM デバイスで使用可能です。

ETM トレースを使用するには、以下の組合せのいずれかが必要です。

- I-jet、I-jet Trace または JTAGjet インサーキットデバッグプローブ、および ETB によって ETM をサポートするデバイス。JTAGjet デバッグプローブは、ETB バッファから ETM データを読み込みます。必ず C-SPY I-jet/JTAGjet ドライバを使用してください。
- JTAGjet-Trace インサーキットデバッグプローブおよび ETM をサポートするデバイス。必ず C-SPY I-jet/JTAGjet ドライバを使用してください。
- J-Trace デバッグプローブおよび ETM をサポートするデバイス。必ず C-SPY J-Link/J-Trace ドライバを使用してください。
- J-Link デバッグプローブおよび ETB (Embedded Trace Buffer) によって ETM をサポートするデバイス。J-Link プローブは、ETB バッファから ETM データを読み込みます。必ず C-SPY J-Link/J-Trace ドライバを使用してください。

詳しくは、*『I-jet、I-jet Trace、I-scope 用 IAR デバッグプローブガイド』*、*『ARM 用 JTAGjet-Trace ユーザガイド』*、*『IAR J-Link および IAR-J-Trace ユーザガイド』* をそれぞれ参照してください。

### MTB(Micro Trace Buffer) トレースを使用するための要件

MTB トレースを使用するには、以下の選択肢のいずれかが必要です。

- I-jet または JTAGjet インサーキットデバッグプローブ、および MTB を持つデバイス
- C-SPY CMSIS-DAP ドライバ、および MTB を持つ CMSIS-DAP 対応のデバイス

### SWO トレースを使用するための条件

SWO トレースを使用するには、I-jet または I-jet Trace インサーキットデバッグプローブ、J-Link, J-Trace、または SWO 通信チャンネルをサポートする ST-LINK デバッグプローブ、および SWD/SWO インタフェースをサポートするデバイスが必要です。

### トレーストリガとトレースフィルタを使用する要件

トレースのトリガおよびトレースフィルタ機能は、ETM トレースが使用可能な場合のみ利用できます。

## トレースデータの収集と使用

以下のタスクについて解説します。

- ETM トレースを開始するには
- SWO トレースを開始するには
- ETM および SWO の並列使用の設定
- ブレークポイントを使用したトレースデータの収集
- トレースデータの検索
- トレースデータの参照

### ETM トレースを開始するには

1 C-SPY を起動する前に :

- デバイスにトレースポートを設定する必要があります。一部のデバイスでは、これはトレースロジックを有効にすると自動的に行われます。ただし、ARM 7 または ARM 9 に基づいた Atmel および ST デバイスなど一部のデバイスでは、トレースポートを明示的に設定する必要があります。これは、C-SPY マクロファイルを介して行います。こうしたファイル (`ETM_init*.mac`) の例は、サンプルプロジェクトにあります。マクロファイルを使用するには、**[プロジェクト]** > **[オプション]** > **[デバッガ]** >

［**設定**］>［**マクロファイルの使用**］を選択します。マクロファイルを指定します。参照ボタンを使用すると便利です。

ハードウェアでトレースシグナルに使用されたピンは、アプリケーションでは使用できません。

2 C-SPY を起動して、C-SPY ドライバのメニューから［**ETM トレース設定**］を選択します。表示される［**ETM トレース設定**］ダイアログボックスで、デフォルト設定を変更する必要があるか確認します。

3 ［トレース］ウィンドウ（ドライバ固有のメニューからアクセス）を開き、［**有効化**］ボタンをクリックしてトレースデータの収集を有効にします。

4 ［**設定の編集**］ボタンをクリックして、［**ETM トレース設定**］ダイアログボックスを開きます。ETM レジスタとピンが正しく初期化されて、デバッグプローブがトレースクロック (TCLK) を受信していることを確認します。ダイアログボックスにトレースクロック周波数（デバッグプローブにより受信）が表示されます。［**キャンセル**］をクリックして、ダイアログボックスを閉じます。

5 実行を開始します。ブレークポイントがトリガされたなどの理由で実行が停止したときは、［トレース］ウィンドウにトレースデータが表示されます。ウィンドウの情報については、237 ページの *［トレース］ウィンドウ*を参照してください。

### SWO トレースを開始するには

**SWO トレースを開始するには :**

1 C-SPY を起動する前に、［**プロジェクト**］>［**オプション**］> *［C-SPY ドライバ］* を選択します。

**[JTAG/SWD]** タブまたは［**接続**］タブをクリックして、［**インタフェース**］>［**SWD**］を選択します。または I-jet の場合、**[JTAG]** に続いてオプション［**SWO**］>［**TraceD0 ピンの SWO**］を選択します。

2 C-SPY を起動したら、*［C-SPY ドライバ］* メニューから［**SWO トレースウィンドウ設定**］を選択します。表示されるダイアログボックスで、［トレース］ウィンドウの出力を制御する設定を行います。

統計的なトレースデータを見るには、オプション［**強制**］>［**PC サンプル**］を選択します（231 ページの *［SWO トレースウィンドウ設定］ダイアログボックス*を参照）。

3 ハードウェアによるトレースデータの生成を設定するには、［**SWO 設定**］ダイアログボックスの［**SWO 設定**］ボタンをクリックします。詳細については、233 ページの *［SWO 設定］ダイアログボックス*を参照してください。

特に以下の設定に注意してください。

- **［CPU クロック］**オプションの値は、アプリケーションを実行する CPU クロックのスピードを反映する必要があります。また、設定はデバッグセッション間で保持されます。
- 通信チャンネルの送信量を減らすには、**［タイムスタンプ］**オプションを無効にします。または、［PC サンプリング］のレートを低くするか、SWO クロック周波数を高く設定します。

4 *［C-SPY ドライバ］*メニューから［SWO トレース］ウィンドウを開き、**［有効化］**ボタンをクリックしてトレースデータの収集を有効にします。

5 実行を開始します。［トレース］ウィンドウは、トレースデータによって継続的に更新されます。このウィンドウの情報については、「237 ページの *［トレース］ウィンドウ*」を参照してください。

### ETM および SWO の並列使用の設定

Cortex-M3 用の JTAGjet-Trace または J-Trace デバッグプローブがある場合、ETM トレースと SWO トレースを同時に使用できます。

この場合、ETM トレースと SWO トレースを有効にすると、SWO トレースは SWO チャンネルを介してストリーム送信される代わりに、ETM トレースバッファでも収集されます。つまり、SWO トレースデータは［SWO トレース］ウィンドウで連続してリアルタイムで更新されるのでなく、実行が停止するまで表示されません。

### ブレークポイントを使用したトレースデータの収集

2 つの実行ポイント間でトレースデータを収集する簡単な方法は、専用のブレークポイントを使用してデータ収集を開始および停止することです。以下から選択します。

- エディタか［逆アセンブリ］ウィンドウで、挿入ポイントを配置して右クリックし、コンテキストメニューから**［トレース開始］**または**［トレース停止］**ブレークポイントを切り替えます。
- ［ブレークポイント］ウィンドウで、**［トレース開始］**、**［トレース停止］**、**［トレースフィルタ］**を選択します。
- C-SPY システムマクロ `__setTraceStartBreak` と `__setTraceStopBreak` も使用できます。

これらのブレークポイントについて詳しくは、「256 ページの *［トレース開始］ブレークポイントダイアログボックス*」と「258 ページの *［トレース停止］ブレークポイントダイアログボックス*」をそれぞれ参照してください。

### トレーストリガとトレースフィルタの使用 :

1 ［**トレース開始**］ダイアログボックスを使用して、トレースデータの収集を開始するための開始条件（開始トリガ）を設定します。

2 ［**トレース停止**］ダイアログボックスでは、トレースデータの収集を停止するための停止条件（停止トリガ）を設定します。

3 オプションで、トレースデータの収集を続行するための追加条件を設定します。さらに、［**トレースフィルタ**］ダイアログボックスを使用して、1 つまたは複数のトレースフィルタを設定します。

4 必要ならば、追加のトレース条件またはトレース停止条件を設定してください。

5 ［トレース］ウィンドウを有効にして、実行を開始します。

6 実行を停止します。

7 トレースデータは、［トレース］ウィンドウおよび［逆アセンブリ］ウィンドウのブラウズモードで参照できます。ここでは、トレーストリガおよびトレースフィルタのトレース跡も見ることができます。

8 トレースフィルタを設定した場合、条件が真で何らかの命令がある場合にトレースデータの収集が実行されます。トレースデータを参照して特定のデータアクセスを探すときには、アクセスが命令 1 つ前に発生していることに注意してください。

## トレースデータの検索

トレースデータの収集が完了したら、収集したデータ内で検索を実行して、特定の割込みや特定の変数のアクセスなど、コードの部分や興味のあるデータを検索することができます。

［**トレースを検索**］ダイアログボックスで検索基準を指定すると、［トレースを検索］ウィンドウに結果が表示されます。

**注 :** ［**トレースを検索**］ダイアログは、使用するハードウェアデバッガシステムによって異なります。

［トレースを検索］ウィンドウは［トレース］ウィンドウと非常によく似ており、表示される列とデータは同じですが、表示される行は、指定された検索基準に一致した行*だけ*です。［トレースを検索］ウィンドウで項目をダブルクリックすると、［トレース］ウィンドウに同じ項目が表示されます。

### トレースデータ内を検索するには :

1 ［トレース］ウィンドウのツールバーで、［**検索**］ボタンをクリックします。

2 ［**トレースを検索**］ダイアログボックスで、検索基準を指定します。

通常は以下を対象に検索基準を選択します。

- 文字の一部。検索基準を追加して適用することができます
- アドレスの範囲
- 特定のアドレス範囲における特定の文字の一部というように、上記を組み合わせたもの

さまざまなオプションの詳細については、274 ページの *［トレースを検索］ダイアログボックス*を参照してください。

3 検索基準を指定したら、**［検索］**をクリックします。［トレースを検索］ウィンドウが表示され、トレースデータを解析できるようになります。詳細については、276 ページの *［トレースを検索］ウィンドウ*を参照してください。

### トレースデータの参照

［トレース］ウィンドウを表示してスクロールするだけで、実行履歴をたどることができます。別の方法として、*ブラウズモード*に入ることができます。

ブラウズモードに入るには、［トレース］ウィンドウで項目をダブルクリックするか、ツールバーの**［ブラウズ］**ボタンをクリックします。

選択された項目が黄色で表示され、ソースウィンドウと逆アセンブリウィンドウの対応する位置が強調表示されます。ここで、上向き矢印キーと下向き矢印キーを使用するか、スクロールしてクリックすることで、トレースデータ内を移動できます。ソースウィンドウと逆アセンブリウィンドウも、対応する位置が表示されるように更新されます。これは、実行履歴を前後に移動するのと似ています。

もう一度ダブルクリックすると、ブラウズモードが終了します。

## トレースについてのリファレンス情報

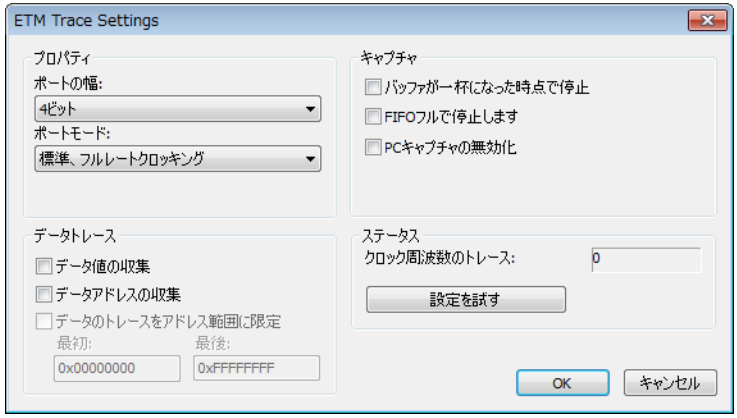
リファレンス情報 :

- 226 ページの *［ETM トレース設定］ダイアログボックス*
- 229 ページの *［ETM トレース設定］ダイアログボックス (J-Link/J-Trace)*
- 231 ページの *［SWO トレースウィンドウ設定］ダイアログボックス*
- 233 ページの *［SWO 設定］ダイアログボックス*
- 237 ページの *［トレース］ウィンドウ*
- 243 ページの *［関数トレース］ウィンドウ*
- 244 ページの *［タイムライン］ウィンドウ*
- 255 ページの *［表示範囲］ダイアログボックス*

- 256 ページの *［トレース開始］ブレークポイントダイアログボックス（シミュレータ）*
- 258 ページの *［トレース停止］ブレークポイントダイアログボックス（シミュレータ）*
- 259 ページの *［トレース開始］ブレークポイントダイアログボックス（I-jet/JTAGjet および CMSIS-DAP）*
- 261 ページの *［トレース停止］ブレークポイントダイアログボックス（I-jet/JTAGjet および CMSIS-DAP）*
- 264 ページの *［トレースフィルタ］ブレークポイントダイアログボックス (I-jet/JTAGjet)*
- 265 ページの *［トレース開始］ブレークポイントダイアログボックス (J-Link/J-Trace)*
- 268 ページの *［トレース停止］ブレークポイントダイアログボックス (J-Link/J-Trace)*
- 271 ページの *［トレースフィルタ］ブレークポイントダイアログボックス (J-Link/J-Trace)*
- 273 ページの *［トレース式］ウィンドウ*
- 274 ページの *［トレースを検索］ダイアログボックス*
- 276 ページの *［トレースを検索］ウィンドウ*
- 277 ページの *［トレースの保存］ダイアログボックス*

## ［ETM トレース設定］ダイアログボックス

**［ETM トレース設定］**ダイアログボックスは、C-SPY ドライバのメニューから使用できます。



このダイアログボックスを使用して、ETM トレースの生成と収集を設定します。

関連項目：

- 220 ページの *ETM トレースを使用するための条件*
- 221 ページの *ETM トレースを開始するには*

### 要件

以下のいずれかが必要です。

- C-SPY CMSIS-DAP ドライバ
- C-SPY I-jet/JTAGjet ドライバ

### ポートの幅

トレースのバス幅に 1、2、4、8、16 ビットを指定します。この値はハードウェアおよびデバッグプローブがサポートするバス幅に合わせる必要があります。

［**FIFO フルで停止**］オプションを使用していない限り、値が小さいと FIFO バッファオーバフローのリスクが高くなります。

### ポートモード

使用されたトレースのクロックレートを指定します。

- 標準、フルレートクロッキング
- 標準、ハーフレートクロッキング
- 多重化
- 逆多重化
- 逆多重化、ハーフレートクロッキング

### データトレース

C-SPY で収集するトレースデータのタイプを選択します。以下から選択します。

**データ値の収集**
データ値を収集します。

**データアドレスの収集**
データアドレスを収集します。

**データのトレースをアドレス範囲に限定**

［**最初**］と［**最後**］テキストボックスで指定したアドレス範囲にある指定したタイプのデータを収集します。

### キャプチャ

トレース収集は通常、実行の開始や停止時、または［トレース開始］あるいは［トレース停止］ブレークポイントがトリガされたときに開始もしくは停止します。これを変更するには、以下から選択します。

**バッファが一杯になった時点で停止**

プローブバッファがいっぱいになればトレースデータの収集を停止します。

**FIFO フルで停止**

FIFO のバッファが満杯になったときにプロセッサを停止します。CPU 上のトレース FIFO バッファは、条件によっては満杯になる（FIFO バッファオーバフロー）可能性があるため、トレースデータが失われる場合があります。これは、CPU が時間的に互いに近いいくつかの分岐を実行している場合（タイトなループなど）に起こります。

**PC キャプチャの無効化**

PC トレースを無効にします。使用するハードウェアに応じて、データトレースが使用できる場合があります。

### ステータス

ETM のステータスが表示されます。

**トレースクロックの周波数**

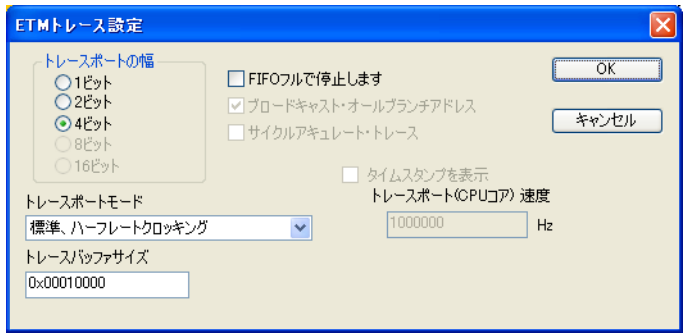
トレースハードウェアが正しく設定されているかを判断しやすくなるように、トレースクロックの周波数を表示します。通常これは、［ポート幅］および［ポートモード］の設定によって異なります。

**設定を適用**

このダイアログボックスで行った設定を適用します。トレースのクロック周波数が更新されます。

# ［ETM トレース設定］ダイアログボックス (J-Link/J-Trace)

［**ETM トレース設定**］ダイアログボックスは、C-SPY ドライバのメニューから使用できます。




このダイアログボックスを使用して、ETM トレースの生成と収集を設定します。

関連項目：

- 220 ページの *ETM トレースを使用するための条件*
- 221 ページの *ETM トレースを開始するには*

## 要件

C-SPY J-Link/J-Trace ドライバ。

## トレースポートの幅

トレースのバス幅に 1、2、4、8、16 ビットを指定します。この値はハードウェアおよびデバッグプローブがサポートするバス幅に合わせる必要があります。Cortex-M3 の場合、J-Trace デバッグプローブで 1、2、4 ビットがサポートされています。ARM7/9 の場合、J-Trace デバッグプローブで 4 ビットのみがサポートされています。

［**FIFO フルで停止**］オプションを使用していない限り、値が小さいと FIFO バッファオーバフローのリスクが高くなります。

## トレースポートモード

使用されたトレースのクロックレートを指定します。

- 標準、フルレートクロッキング
- 標準、ハーフレートクロッキング
- 多重化

- 逆多重化
- 逆多重化、ハーフレートクロッキング

**注 :** J-Trace ドライバの場合、使用可能な選択肢は使用するデバイスによって異なります。

### トレースバッファサイズ

トレースバッファのサイズを指定します。デフォルトでは、トレースフレームの数は `0xFFFF` です。ARM7/9 では最大数は `0xFFFFF` で、Cortex-M3 では `0x3FFFFF` です。

ARM7/9 の場合、1 つのトレースフレームは J-Trace の物理的バッファサイズの 2 バイトに相当します。Cortex-M3 では、1 つのトレースフレームはバッファサイズの約 1 バイトになります。

**注 :** [**トレースバッファサイズ**] オプションは J-Trace ドライバでのみ使用できます。

### サイクルアキュレート・トレース

トレースデータが使用できない場合でも、プロセッサのクロックに同期するトレースフレームを出力します。このため、リアルタイムのタイミング計算にトレースデータを利用できます。ただしこのオプションを選択すると、FIFO バッファオーバフローの危険性が増大します。

**注 :** このオプションは、ARM7/9 デバイスでのみ使用できます。

### ブロードキャスト・オールブランチ

プロセッサからより詳細なアドレストレース情報が送信されます。ただしこのオプションを選択すると、FIFO バッファオーバフローの危険性が増大します。

**注 :** このオプションは、ARM7/9 デバイスでのみ使用できます。Cortex では、このオプションは常に有効になっています。

### FIFO フルで停止

FIFO のバッファが満杯になったときにプロセッサを停止します。CPU 上のトレース FIFO バッファは、条件によっては満杯になる（FIFO バッファオーバフロー）可能性があるため、トレースデータが失われる場合があります。これは、CPU が時間的に互いに近いいくつかの分岐を実行している場合（タイトなループなど）に起こります。

### タイムスタンプを表示

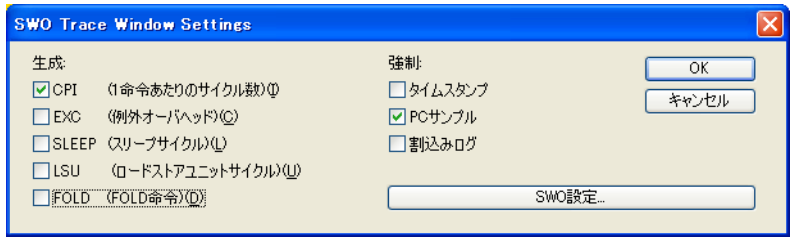
［トレース］ウィンドウの［**インデックス**］列にサイクルではなく、秒を表示します。このオプションを使用するには、［**トレースポート（CPU コア）速**

度］テキストボックスに、使用する CPU に適した速度を設定することも必要です。

**注 :** このオプションは、J-Trace ドライバを ARM7/9 デバイスで使用する場合のみ利用できます。

## ［SWO トレースウィンドウ設定］ダイアログボックス

**［SWO トレースウィンドウ設定］**ダイアログボックスは、**[I-jet/JTAGjet]** メニュー、**[J-Link]** メニューまたは **[ST-LINK]** メニュー、または［SWO トレース］ウィンドウのツールバーから使用できます。




このダイアログボックスを使用して、［SWO トレース］ウィンドウに表示する内容を指定します。

トレースデータの生成も設定する必要があります。**［SWO 設定］**をクリックしてください。詳細については、233 ページの *［SWO 設定］ダイアログボックス*を参照してください。

### 要件

以下のいずれかが必要です。

- I-jet または I-jet Trace インサーキットデバッグプローブ
- J-Link/J-Trace JTAG/SWD プローブ
- ST-LINK JTAG/SWD プローブ

### 強制

SWO トレースデータを使用して他の機能により有効になっていない場合、データ生成を有効にします。［トレース］ウィンドウには、生成されたすべての SWO データが表示されます。プロファイリングなど C-SPY の他の機能では、SWO トレースデータの生成を有効にすることもできます。他の機能によって生成が有効化されていない場合、**［強制］**オプションを使用して SWO トレースデータを生成します。

生成されたデータは、［トレース］ウィンドウに表示されます。以下から選択します。

**タイムスタンプ**

SWO 通信チャンネルを介して送信される、さまざまな SWO トレースパケットについてタイムスタンプを有効にします。タイムスタンプ値の間隔をドロップダウンリストから選択します。たとえば、1 サイクルごとにカウントする場合は 1 を、16 サイクルごとにカウントする場合は 16 を選択します。最小値は、各イベントパケットの間隔が十分に長い場合にのみ有益であることに注意してください。16 は低い SWO クロック周波数を使用している場合に便利です。

このオプションは I-jet には適用されません。

**PC サンプル**

プログラムカウンタレジスタ (PC) の定期的なサンプリングを有効にします。サンプリングレートを選択するには、233 ページの *［SWO 設定］ダイアログボックス*のオプション **［PC サンプリング］**を参照してください。

**割込みログ**

［SWO トレース］ウィンドウに対して割込みログの生成を強制します。割込みにトレースデータを使用する他の C-SPY 機能については、369 ページの*割込み*を参照してください。

**ITM ログ**

［SWO トレース］ウィンドウに対して ITM ログの生成を強制します。

このオプションは I-jet のみに適用されます。

### 生成

以下のイベントでトレースデータの生成を有効にします。生成されたデータは、［トレース］ウィンドウに表示されます。カウンタの値は、［SWO トレース］ウィンドウの **［コメント］**列に表示されます。以下から選択します。

**CPI**

CPI カウンタについてトレースデータの生成を有効にします。

**EXC**

EXC カウンタについてトレースデータの生成を有効にします。

**SLEEP**

SLEEP カウンタについてトレースデータの生成を有効にします。

**LSU**

LSU カウンタについてトレースデータの生成を有効にします。

**FOLD**

FOLD カウンタについてトレースデータの生成を有効にします。

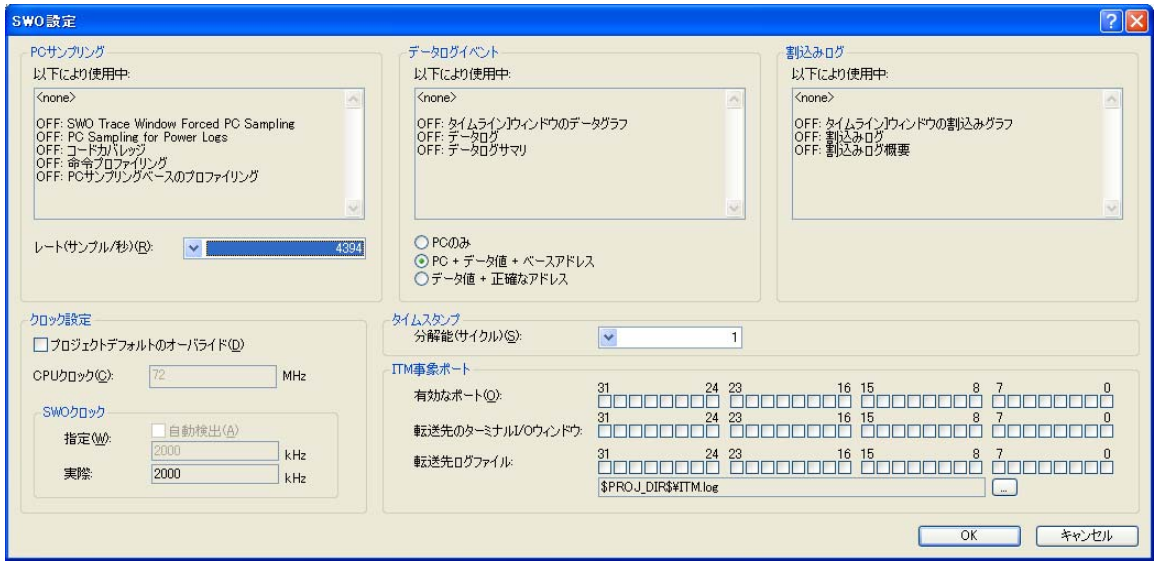
**SWO 設定**

［**SWO 設定**］ダイアログボックスを表示します。ここでは、ハードウェアのトレースデータの生成を設定できます。233 ページの *［SWO 設定］ダイアログボックス*を参照してください。

このボタンは、I-jet の使用中には利用できません。

## ［SWO 設定］ダイアログボックス

［**SWO 設定**］ダイアログボックスは、C-SPY ドライバのメニュー、または［**SWO トレースウィンドウの設定**］ダイアログボックスから使用できます。




このダイアログボックスは、SWO 通信チャンネルおよびハードウェアによるトレースデータの生成を設定するときに使用します。

222 ページの *SWO トレースを開始するには*も参照してください。

### 要件

以下のいずれかが必要です。

- I-jet または I-jet Trace インサーキットデバッグプローブ
- J-Link/J-Trace JTAG/SWD プローブ
- ST-LINK JTAG/SWD プローブ

### PC サンプリング

プログラムカウンタのサンプリング動作を制御します。以下を指定できます。

**以下により使用中**

PC サンプリングでトレースデータを使用可能な C-SPY の機能を一覧表示します。ON は、現在トレースデータを使用中の機能を示します。OFF は、現在トレースデータを使用中でない機能を示します。

**レート**

サンプリングレート（1 秒あたりのサンプル数）をドロップダウンリストから選択します。最大可能サンプリングレートは、SWO クロック値と、SWO 通信チャンネル経由で送信される他のデータ量により決まります。SWO 通信チャンネルが大量のデータを処理できるほど高速でない場合、リストで最大値を指定すると適切に機能しません。

このオプションは I-jet には適用されません。

**分周**

CPU クロックのスピードに適用されて PC サンプルのレートを決定する除算を選択します。最大可能サンプリングレートは、SWO クロック値と、SWO 通信チャンネル経由で送信される他のデータ量により決まります。SWO 通信チャンネルが大量のデータを処理できるほど高速でない場合、リストで小さい値を指定すると適切に機能しません。

このオプションは I-jet のみに適用されます。

### データログイベント

データログブレークポイントがトリガされたときにログに記録する対象を指定します。以下の項目を選択できます。

**以下により使用中**

データログイベントでトレースデータを使用可能な C-SPY の機能を一覧表示します。ON は、現在トレースデータを使用中の機能を示します。OFF は、現在トレースデータを使用中でない機能を示します。

**PC のみ**

プログラムカウンタの値をロギングします。

**PC + データ値 + ベースアドレス**

プログラムカウンタの値、データオブジェクトの値、およびそのベースアドレスをロギングします。

**データ値 + 正確なアドレス**

データオブジェクトの値、およびアクセスされたデータオブジェクトの正確なアドレスを記録します。

### 割込みログ

割込みログでトレースデータを使用可能な C-SPY の機能を一覧表示します。ON は、現在トレースデータを使用中の機能を示します。OFF は、現在トレースデータを使用中でない機能を示します。

割込みログの詳細については、369 ページの*割込み*を参照してください。

### プロジェクトデフォルトのオーバライド

［**プロジェクト**］>［**オプション**］>［**J-Link/J-Trace**］>［**設定**］ページ（J-Link/J-Trace の場合）または［**プロジェクト**］>［**オプション**］>［**ST-Link**］>［**設定**］ページ（ST-LINK の場合）の［**CPU クロック**］と［**SWO クロック**］のデフォルト値をオーバライドします。

このオプションは I-jet には適用されません。

### プロジェクト設定のオーバライド

［**プロジェクト**］>［**オプション**］>［**I-jet**］>［**設定**］ページの［**CPU クロック**］と［**SWO プリスケーラ**］のデフォルト値をオーバライドします。

このオプションは I-jet のみに適用されます。

### CPU クロック

内部プロセッサクロック HCLK の正確なクロック周波数を指定します (MHz)。10 進数で指定できます。

この値は、SWO の通信速度を設定するときに使用します。

J-Link と ST-LINK の場合、この値はタイムスタンプの計算にも使用されます。

### SWO クロック

SWO 通信チャンネルのクロック周波数を指定します (kHz)。以下から選択します。

**自動検出**

J-Link デバッグプローブで使用できる最大可能周波数を自動的に使用します。選択された場合、［指定］（最大）テキストボックスにその周波数が表示されます。

**指定**

［自動検出］が選択されていない場合に、使用する周波数を手動で選択します。10 進数で指定できます。このオプションは、送信中にデータパケットが失われる場合に使用します。

**実際**

実際に使用される周波数を表示します。最大周波数とは微妙に異なる場合があります。

このオプションは I-jet には適用されません。

### SWO プリスケーラ

SWO 通信チャンネルのクロックプリスケーラを指定します。プリスケーラによって、SWO クロック周波数が決まります。通信中にデータパケットが失われる場合、より大きいプリスケーラの値を使用してみてください。以下から選択します。

**自動**

I-jet デバッグプローブで使用できる最大可能周波数を自動的に使用します。

**1, 2, 5, 10, 20, 50, 100**

プリスケーラの値。

このオプションは I-jet のみに適用されます。

### タイムスタンプ

タイムスタンプ値の分解能を選択します。たとえば、1 サイクルごとにカウントする場合は 1 を、16 サイクルごとにカウントする場合は 16 を選択します。最小値は、各イベントパケットの間隔が十分に長い場合にのみ有益であることに注意してください。

このオプションは I-jet には適用されません。

### ITM 事象ポート

リダイレクトするポートと転送先を選択します。ITM 事象ポートは、アプリケーションからデバッガホストに、プログラムの実行を停止せずにデータを送信するときに使用されます。32 個のポートがあります。以下から選択します。

**ポートの有効化**

使用するポートを有効にします。有効にしたポートのみが、SWO 通信チャンネル経由でデバッガにデータを送信します。

ポート 0 はターミナル I/O のライブラリ関数で使用されます。

ポート 1 から 4 は、［イベントログ］の ITM マクロによって使用されます。

ポート 5 は、ITM マクロに追加されるオプションの PC 値に使用されます。

**転送先の［ターミナル I/O］ウィンドウ**

［ターミナル I/O］ウィンドウへのデータルーティングに使用するポートを指定します。

**転送先ログファイル**

ログファイルへのデータルーティングに使用するポートを指定します。デフォルトのログファイル以外を使用する場合は、参照ボタンを使用して指定します。

アプリケーションの `stdout` と `stderr` は、セミホスティングではなく、SWO 経由で C-SPY の［ターミナル I/O］ウィンドウにルートすることができます。これには、**［プロジェクト］>［オプション］>［一般オプション］>［ライブラリ構成］>［低レベルインタフェースのライブラリ実装］>［stdout/stderr］>［SWO 経由］**を選択します。こうすることで、セミホスティングを使用した場合に比べて、`stdout/stderr` のパフォーマンスが格段に向上します。

これは、**［ポートの有効化］**および**［転送先ターミナル I/O］**オプションでポート設定の選択を解除すると無効になります。

## ［トレース］ウィンドウ

［トレース］ウィンドウは、C-SPY ドライバメニューから使用できます。

このウィンドウには収集されたトレースデータが表示されます。

［トレース］ウィンドウの表示内容は、使用する C-SPY ドライバおよびデバッグプローブのトレースサポートによって異なります。

**注 :** C-SPY シミュレータには［ETM トレース］、［SWO トレース］、［トレース］の 3 つのウィンドウがあります。これらのウィンドウは少しずつ異なります。

### 要件

以下のいずれかが必要です。

- C-SPY シミュレータ
- I-jet または I-jet Trace インサーキットデバッグプローブ
- JTAGjet デバッグプローブ
- J-Link/J-Trace JTAG/SWD プローブ
- ST-LINK JTAG/SWD プローブ

### ［トレース］ツールバー

［トレース］ウィンドウと［関数トレース］ウィンドウのツールバーには以下が含まれます。

**有効 / 無効**

このウィンドウにおけるトレースデータの収集および表示を有効および無効にします。［関数トレース］ウィンドウではこのボタンは使用できません。

**トレースデータのクリア**

トレースバッファをクリアします。［トレース］ウィンドウと［関数トレース］ウィンドウが両方ともクリアされます。

**ソースの切替え**

［トレース］列で、逆アセンブリだけを表示するか、逆アセンブリおよび対応するソースコードの両方を表示するか切り替えます。

**ブラウズ**

［トレース］ウィンドウで選択された項目のブラウズモードのオンとオフを切り替えます（225 ページの *トレースデータの参照*を参照）。

**検索**

検索を実行するダイアログボックスを表示します（274 ページの *［トレースを検索］ダイアログボックス*を参照）。

**保存**

［ETM トレース］および［SWO トレース］の各ウィンドウでは、このボタンによって **［トレースの保存］**ダイアログボックスが表示されます（277 ページの *［トレースの保存］ダイアログボックス*を参照）。

C-SPY I-jet/JTAG-jet ドライバと C-SPY シミュレータでは、このボタンによって標準の［**名前を付けて保存**］ダイアログボックスが表示され、ここで収集したトレースデータをタブで列ごとに区切られたテキストファイルに保存できます。

**設定の編集**

C-SPY シミュレータでは、このボタンは使用できません。

［ETM トレース］ウィンドウでは、このボタンによって［**トレース設定**］ダイアログボックスが表示されます（229 ページの *［ETM トレース設定］ダイアログボックス (J-Link/J-Trace)* と 226 ページの *［ETM トレース設定］ダイアログボックス*を参照）。

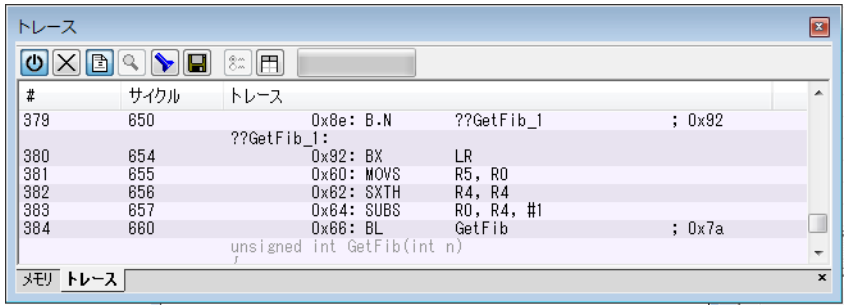
［SWO トレース］ウィンドウでは、このボタンによって［**SWO トレースウィンドウ設定**］ダイアログボックスが表示されます（231 ページの *［SWO トレースウィンドウ設定］ダイアログボックス*を参照）。

**式の編集（C-SPY シミュレータのみ）**

［トレース式］ウィンドウを開きます（273 ページの *［トレース式］ウィンドウ*を参照）。

## 表示エリア（C-SPY シミュレータ内）

このエリアには、収集された実行済みのマシン命令のシーケンスが表示されます。また、式のトレースデータも表示できます。




このエリアには、C-SPY シミュレータの以下の列が含まれます。

**#**

トレースバッファの各行のシリアル番号。バッファ内の移動を簡略化します。

**サイクル**

この時点までのサイクル数。

**トレース**

収集された実行済みのマシン命令シーケンス。オプションで、対応するソースコードも表示できます。

***式***

表示するように定義した式はそれぞれ別の列に表示されます。式列の各エントリには、同じ行の命令が実行された*後*に値が表示されます。トレースデータを収集する式は、[トレース式] ウィンドウで指定します（273 ページの *[トレース式] ウィンドウ*を参照）。

赤の行は前の行を示し、赤い行は連続しません。つまり、トレースデータがオーバフローによって失われたときなど、収集されたトレースデータに差があることを示します。

### 表示エリア（ETM トレース）

このエリアには以下の列が含まれます。

**インデックス、#**

それぞれのパケットに対応する番号。パケットの例は、命令や同期ポイント、例外マーカなどです。

**フレーム | 時間**

サイクルアキュレートモード（ARM7/9 が必要です）でトレースデータを収集する場合（**[ETM トレース設定]** ダイアログボックスで **[サイクルアキュレート・トレース]** を有効化）、値は実行開始後に経過したサイクル数に一致します。この列は C-SPY J-Link/J-Trace ドライバでのみ使用できます。

サイクルアキュレート以外のモードでトレースデータを収集する場合、値はサイクルのおよその数となります。Cortex-M デバイスの場合、値は実際のサイクル数によって繰り返し調整されます。

**[ETM トレース設定]** ダイアログボックスで **[タイムスタンプを表示]** オプションが選択されている場合、値としてサイクルではなく時間が表示されます。値を時間として表示するには、サイクルアキュレートモードでデータを収集する必要があります（226 ページの *[ETM トレース設定] ダイアログボックス*と 229 ページの *[ETM トレース設定] ダイアログボックス (J-Link/J-Trace)* の **[サイクルアキュレート・トレース]** オプション、および J-Link/J-Trace ドライバを参照）。

**サイクル**

JTAGjet-Trace 内部のタイムスタンプに基づいたサイクル数。

この列は JTAGjet-Trace でのみ使用できます。

**アドレス**
実行した命令のアドレス。

**OP コード**
実行した命令の動作コード。

この列は J-Link/J-Trace でのみ使用できます。

**トレース**
収集された実行済みのマシン命令シーケンス。オプションで、対応するソースコードも表示できます。

**Exec**
実行モード — ARM、Thumb、NoExec。

この列は JTAGjet-Trace でのみ使用できます。

**except**
例外のタイプ（発生した場合）。

この列は JTAGjet-Trace でのみ使用できます。

**アクセス**
データトレースのアクセスタイプ。

この列は JTAGjet-Trace でのみ使用できます。

**データアドレス**
データトレースのアドレス。

この列は JTAGjet-Trace でのみ使用できます。

**データ値**
データトレースの値。

この列は JTAGjet-Trace でのみ使用できます。

**コメント**
追加情報。

赤の行は前の行を示し、赤い行は連続しません。つまり、トレースデータがオーバフローによって失われたときなど、収集されたトレースデータに差があることを示します。

### 表示エリア（SWO トレース）

このエリアには、SWO トレースの以下の列が含まれます。

**インデックス**

トレースバッファの各行のインデックス番号。バッファ内の移動を簡略化します。

この列は JTAGjet-Trace でのみ使用できます。

**SWO パケット**

取得した SWO パケットの内容が、16 進数の値の値で表示されます。

**サイクル**

実行の開始からイベントまでのサイクルの概数。

J-Link の場合、この値は CPU により報告されます。

I-jet の場合、この値は I-jet/JTAGjet-Trace 内部のタイムスタンプと同じになります。

**イベント**

取得した SWO パケットのイベントのタイプ。列にオーバフローと表示される場合、多くの SWO 機能で同時に SWO チャンネルが使用されているため、データバケットは送信できませんでした。通信チャンネルの送信量を減らすには、SWO ボタン（IDE のメインウィンドウのツールバー上にあります）にマウスのポインタを合わせて、どの C-SPY 機能がトレースデータの生成を要求しているかについて詳細なツールチップ情報を入手します。一部の機能を無効にしてください。

**値**

イベントの値（該当する場合）。

**トレース**

イベントがサンプルの PC 値の場合、逆アセンブルされた命令もこの列に表示されます。オプションで、対応するソースコードも表示できます。

**コメント**

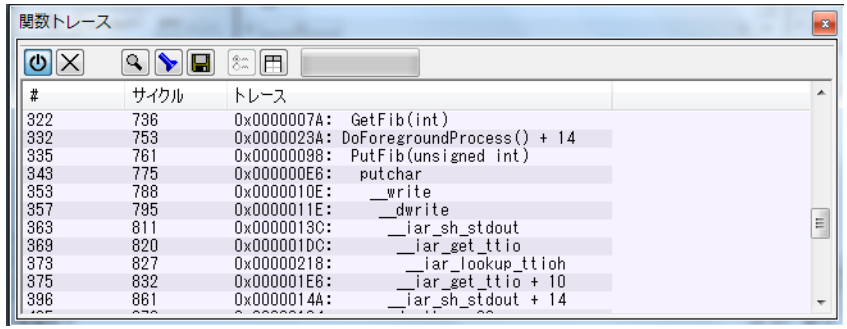
追加情報。選択したトレースイベントカウンタの値、データログブレークポイントに使用されたコンパレータ（ハードウェアブレークポイント）の数が表示されます。

赤の行は前の行を示し、赤い行は連続しません。つまり、トレースデータがオーバフローによって失われたときなど、収集されたトレースデータに差があることを示します。

表示エリアに不明な文字が表示される場合、**［SWO 設定］**ダイアログボックスの**［CPU クロック］**に正しい値を指定したか確認してください。

## ［関数トレース］ウィンドウ

［関数トレース］ウィンドウは、デバッグセッション中に［C-SPY ドライバ］メニューから使用できます。




このウィンドウには、［トレース］ウィンドウに表示されたトレースデータのサブセットが表示されます。［関数トレース］ウィンドウにはすべての行は表示されません。関数呼出しと関数からの復帰に対応するトレースデータだけが表示されます。

### 要件

以下のいずれかが必要です。

- C-SPY シミュレータ
- I-jet または I-jet Trace インサーキットデバッグプローブ
- JTAGjet デバッグプローブ
- J-Link/J-Trace JTAG/SWD プローブ
- ST-LINK JTAG/SWD プローブ

### ツールバー

ツールバーについては、237 ページの*［トレース］ウィンドウ*を参照してください。

### 表示エリア

表示エリアの列については、237 ページの*［トレース］ウィンドウ*を参照してください。

## ［タイムライン］ウィンドウ

［タイムライン］ウィンドウは、デバッグセッション中に［C-SPY ドライバ］メニューから使用できます。

使用するハードウェア、デバッグプローブ、C-SPY ドライバの機能に応じて、このウィンドウにはトレースデータがさまざまな共通の時間軸に比例したグラフで表示されます。

- コールスタックグラフ
- データロググラフ
- イベントグラフ
- 割込みロググラフ
- Power ロググラフ

### グラフを表示するには：

1. *［C-SPY ドライバ］*>［**SWO 設定**］を選択して、［**SWO 設定**］ダイアログボックスを開きます。［**CPU クロック**］オプションが、アプリケーションの CPU クロックの値と同じに設定されているか確認してください。SWO クロックを設定して、デバッグプローブへ正しいデータ転送を行うには、こうする必要があります。

   C-SPY シミュレータを使用する場合は、この手順は無視してかまいません。

2. C-SPY ドライバのメニューから［**タイムライン**］を選択して、［タイムライン］ウィンドウを開きます。

3. ［タイムライン］ウィンドウのグラフエリア右クリックして、コンテキストメニューから［**有効化**］を選択し、特定のグラフを有効にします。

4. データロググラフの場合は、［タイムライン］ウィンドウでグラフィック表示を行う各変数にデータログブレークポイントを設定する必要があります。162 ページの *［データログ］ブレークポイントダイアログボックス（C-SPY ハードウェアドライバ）*を参照してください。

5. イベントグラフの場合は、イベントを生成するアプリケーションのソースコードにプリプロセッサマクロを追加する必要があります。107 ページの*イベントログを開始するには*を参照してください。

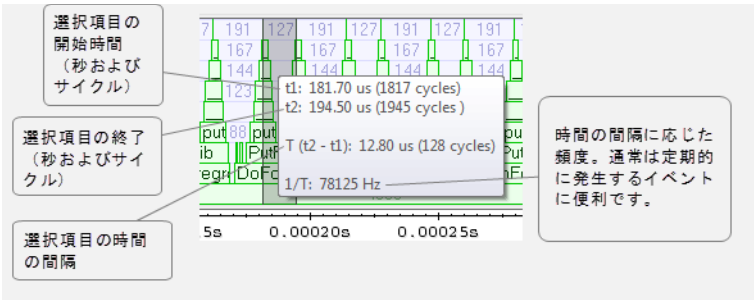
6. ツールバーで［**実行**］をクリックして、アプリケーションの実行を開始します。グラフが表示されます。

   グラフ内を移動するには、以下の選択肢のいずれかを使用します。

   - 右クリックしてコンテキストメニューから［**ズームイン**］または［**ズームアウト**］を選択します。その他に、[+] や [-] のキーを使用することもでき

ます。使用したコマンドに応じて、グラフがズームインまたはズームアウトします。

- グラフを右クリックして、コンテキストメニューから［**移動**］およびグラフ上で前後に移動するための適切なコマンドを選択します。または、矢印キーや [Home]、[End]、[Ctrl]+[End] などのショートカットキーをどれでも使用できます。
- 関心のあるサンプルの値をダブルクリックすると、対応するソースコードがエディタウィンドウと［逆アセンブリ］ウィンドウで強調表示されます。
- グラフをクリックし、ドラッグして間隔を選択します。[Enter] または右クリックして、コンテキストメニューから［**ズーム**］>［**選択範囲をズーム**］を選択します。選択内容にズームインします。グラフで選択した部分について詳細なツールチップ情報を入手するには、マウスポインタで選択項目をポイントします。




マウスポインタをグラフにあわせると、その場所の詳細なツールチップ情報が表示されます。

### 要件

使用するハードウェア、デバッグプローブ、C-SPY ドライバの機能によっては、表示エリアに異なるグラフが読み込まれることがあります。

| ターゲットシステム | コールスタックグラフ | データロググラフ | イベントグラフ | 割込みロググラフ | Power ロググラフ |
|---|---|---|---|---|---|
| C-SPY シミュレータ | X | X | -- | X | -- |
| CMSIS-DAP | X[2] | -- | -- | -- | -- |
| I-jet | X[2] | X | X | X | X |
| I-jet Trace | X | X | X | X | X |
| JTAGjet | X[2] | -- | -- | -- | -- |
| JTAGjet-Trace | X | -- | -- | -- | -- |
| J-Link | X[2] | X | X | X | X |
| J-Trace | X | X[1] | X[1] | X[1] | -- |
| ST-LINK | -- | X | -- | X | -- |

*表 11: ［タイムライン］ウィンドウでサポートされているグラフ*

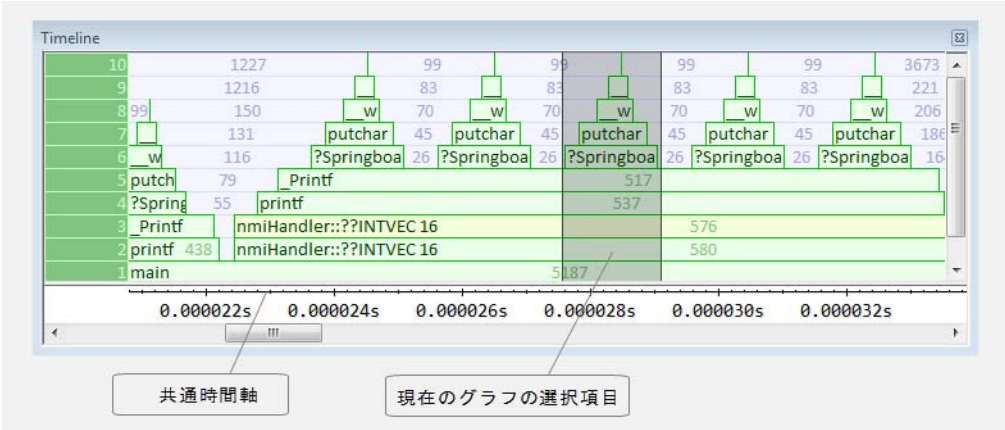
1 ETM トレースが有効な場合は非常に制限される。

2 ETB/MTB が必要。

トレースデータの要件については、220 ページの *トレースを使用するための条件*を参照してください。

### コールスタックグラフの表示エリア

コールスタックグラフには、ETM トレースによって収集された呼出しおよびリターンのシーケンスが表示されます。




グラフの下部分には通常、main があり、その上には main という関数があります。横方向の棒は関数の呼出しを示し、4 つの異なる色を使用します。

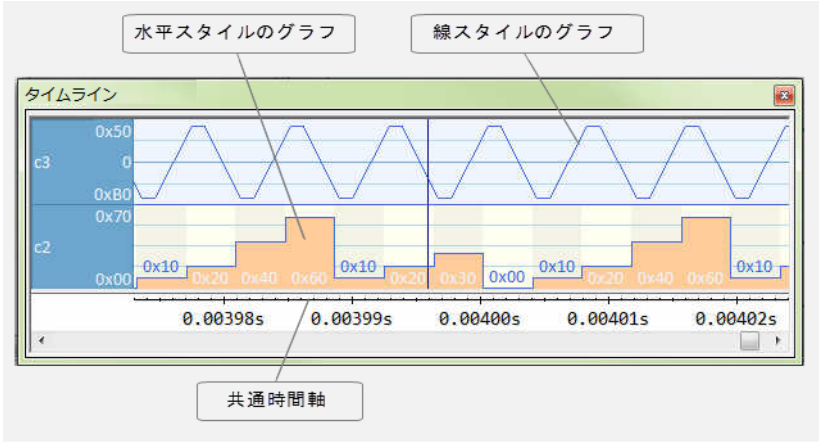
- 緑は、デバッグ情報を持つ通常の C 関数です。
- ライトグリーンは、デバッガがアセンブララベルを通してのみ認識する関数です。
- 黄色または薄い黄色は、割込みハンドラで緑の場合と同じように区別されます。

数字は関数の呼出し時、または呼出し間のサイクル数を表します。

ウィンドウの下部分に、秒を時間単位として使用する共通の時間軸があります。

### データロググラフの表示エリア

データロググラフには、SWO トレースまたは C-SPY シミュレータによって生成されるデータログが、データログブレークポイントとして指定された最高 4 つの変数またはアドレス範囲について表示されます。



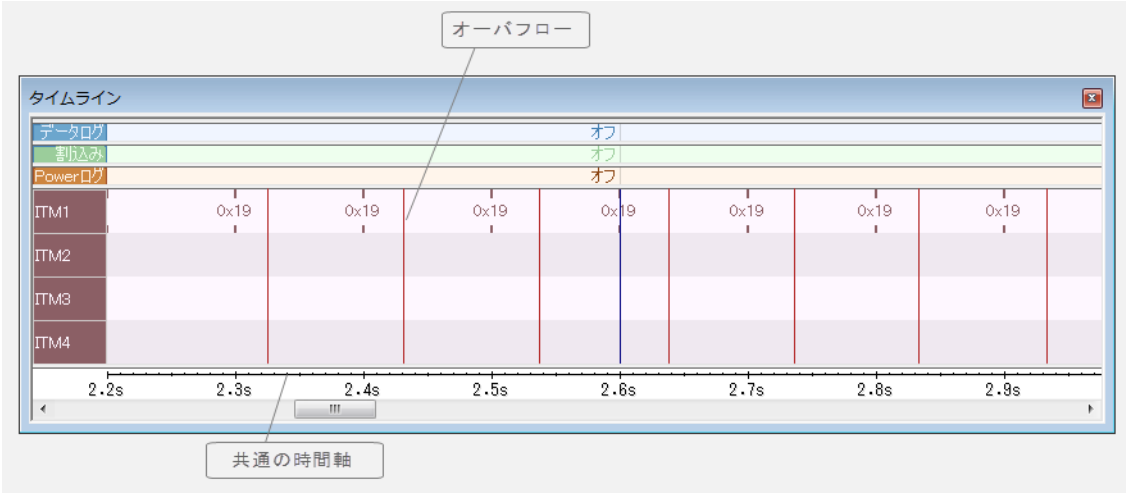

場所 :

- グラフ左端のレベルエリアには、データログブレークポイントを指定した変数名またはアドレスが表示されます。
- グラフ自体には、変数の値が時間とともにどう変化するかが示されます。ラベルエリアには、変数に対して Y 軸の制限や範囲も表示されます。コンテキストメニューを使用して、これらの制限を変更できます。このグラフは、［データログ］ウィンドウの情報をグラフィック表示したものです（125 ページの *［データログ］ウィンドウ*を参照）。
- グラフは連続したログ間の細い線として、または各ログを示す長方形（オプションでカラー）として表示することができます。
- 赤色の垂直線はオーバフローを示します。これは、通信チャンネルがすべてのデータログをターゲットシステムから送信できなかったことを示します。赤い疑問符は値のないログを示します。

ウィンドウの下部分に、秒を時間単位として使用する共通の時間軸があります。

### イベントグラフの表示エリア

イベントグラフには、実行がアプリケーションコードの特定の位置を通過したときに生成されたイベントが表示されます。



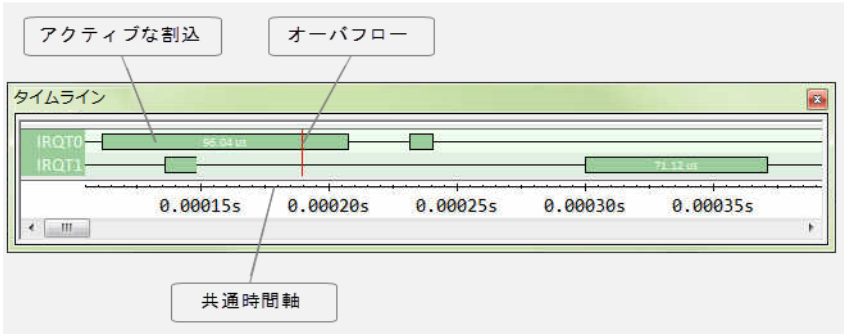

場所：

- グラフ左端のラベルエリアには、チャンネル名が表示されます。
- 各チャネルについて、イベントがいつ発生したかを示す縦の線が入ります。オプションで、イベントとともに渡されたイベントの値を表示することができます。
- グラフは連続したログ間の細い線として、または各ログを示す長方形（オプションでカラー）、あるいは縦の棒グラフとして表示することができます。
- 赤色の垂直線はオーバフローを示します。これは、通信チャンネルがすべてのデータログをターゲットシステムから送信できなかったことを示します。

ウィンドウの下部分に、秒を時間単位として使用する共通の時間軸があります。

### 割込みロググラフの表示エリア

割込みロググラフには、SWO トレースまたは C-SPY シミュレータによって報告される割込みが表示されます。つまり、このグラフには、アプリケーションプログラム実行中の割込みイベントがグラフィック表示されます。



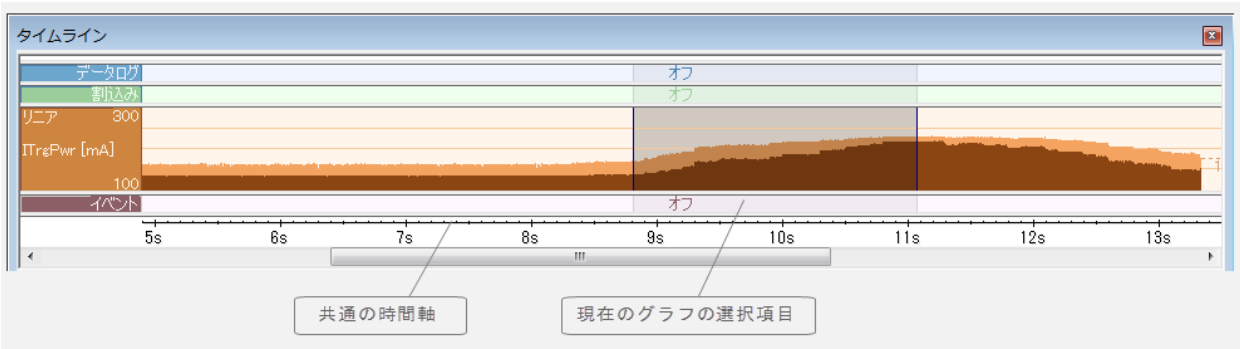

場所 :

- グラフ左端のラベルエリアには、割込み名が表示されます。
- グラフ自体には、アクティブな割込みが濃い緑の横棒として示され、白の数値は割込みで費やされた時間を示します。このグラフは、[割込みログ] ウィンドウの情報をグラフィック表示したものです（385 ページの *[割込みログ] ウィンドウ*を参照）。
- 横方向の境界なしにバーが表示される場合は、次の 2 つの原因が考えられます。
  - 割込みがリエントラントで、自身に対する割込みです。いちばん内側の割込みのみに境界が表示されます。
  - おそらくログが欠損しているために、割込みの開始と終了のシーケンスが不規則です。
- 縦の境界がないバーが表示される場合、境界がない部分はログの時間が正確でないことを示します。
- 赤色の垂直線は、オーバフローを示します。これは、通信チャンネルがすべての割込みログをターゲットシステムから送信できなかったことを示します。

ウィンドウの下部分に、秒を時間単位として使用する共通の時間軸があります。

### Power ロググラフの表示エリア

Power ロググラフには、電力測定サンプルのグラフビューが表示されます。




場所：

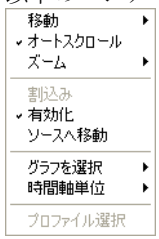
- グラフ左端のラベルエリアには、測定チャンネル名が表示されます。
- グラフ自体には、デバッグプローブや関連のハードウェアによって生成された電力測定サンプルが表示されます。
- グラフは連続したログ間の細い線として、または各ログを示す長方形（オプションでカラー）、あるいは列として表示することができます。
- グラフの解像度は変更することができます。
- 赤色の垂直線は、オーバフローを示します。これは、通信チャンネルがすべての割込みログをターゲットシステムから送信できなかったことを示します。

### 選択およびナビゲーション

選択するには、クリックしてドラッグします。選択内容はすべてのグラフについて縦方向に伸びますが、選択したグラフに対して濃い色で強調表示されます。左右の矢印キーを使用して、選択したグラフ内を前後に移動することができます。Home キーおよび End キーを使用して、先頭または最後の地点にそれぞれ移動します。選択内容を拡げるには、ナビゲーションキーを Shift キーと組み合わせて使用します。

### コンテキストメニュー

以下のコンテキストメニューがあります。




**注 :** コンテキストメニューには、すべてのグラフに共通なコマンドと、各グラフに固有のコマンドが含まれます。この図はコールスタックグラフのコンテキストメニューを表します。つまり、メニューの外観が他のグラフとは少し違います。

以下のコマンドがあります。

**移動（すべてのグラフ）**

グラフ上を移動するコマンド。以下から選択します。

**［次へ］**は、グラフ内の次の適切な地点に選択内容を動かします。ショートカットキー：→。

**［前へ］**は、グラフ内の次の適切な地点に選択内容を戻します。ショートカットキー：←。

**［最初］**は、グラフ内の最初のデータ項目に選択内容を動かします。ショートカットキー：Home。

**［最後］**は、グラフ内の最後のデータ項目に選択内容を動かします。ショートカットキー：End。

**終了**は、表示されたすべてのグラフの最後のデータに選択内容を動かします。つまり、時間軸の最後です。ショートカットキー：Ctrl+End。

**オートスクロール（すべてのグラフ）**

スクロールのオンとオフを切り替えます。オンの場合、コマンド**［移動］>［終了］**を実行すると最近収集したデータが自動的に表示されます。

**ズーム（すべてのグラフ）**

ウィンドウをズームするためのコマンド。つまり、タイムスケールを変更します。以下から選択します。

**選択範囲をズーム**は、現在の選択内容をウィンドウに合わせます。ショートカットキー：リターン。

**ズームイン**は、タイムスケールを拡大します。ショートカットキー：+。

**ズームアウト**は、タイムスケールを縮小します。ショートカットキー：-。

**10ns** や **100ns**、**1us** などは、それぞれ間隔 10 ナノ秒、100 ナノ秒、1 マイクロ秒をウィンドウに合わせます。

**1ms**、**10ms** などは、間隔をそれぞれ 1 ミリ秒または 10 ミリ秒にしてウィンドウに合わせます。

**10m**、**1h** などは、間隔をそれぞれ 10 分または 1 時間にしてウィンドウに合わせます。

**データログ（データロググラフ）**

以下のデータログ固有のコマンドが使用可能なことを示す見出し。

**イベント（イベントグラフ）**

以下のイベント固有のコマンドが使用可能なことを示す見出し。

**Power ログ（Power ロググラフ）**

以下の Power ログ固有のコマンドが使用可能なことを示す見出し。

**コールスタック（コールスタックグラフ）**

以下のコールスタック固有のコマンドが使用可能なことを示す見出し。

**割込み（割込みロググラフ）**

以下の割込みログ固有のコマンドが使用可能なことを示す見出し。

**有効化（すべてのグラフ）**

グラフの表示のオンとオフを切り替えます。グラフを無効にすると、そのグラフは［タイムライン］ウィンドウで OFF として示されます。グラフについて収集されたトレースデータがない場合、グラフではなくデータなしと表示されます。

***変数（データロググラフ）***

以下のデータログ固有のコマンドが適用される変数名。このメニューコマンドはコンテキストに依存します。つまり、［タイムライン］ウィンドウで選択したデータロググラフが反映されます（最高 4 つまで）。

***変数（イベントグラフ）***

以下のイベント固有のコマンドが適用されるチャンネル名。このメニューコマンドはコンテキストに依存します。つまり、［タイムライン］ウィンドウで選択したイベントグラフのチャンネルが反映されます（最高 4 つまで）。

**線グラフ（データロググラフ）**

細い線ではなく、カラーの実線グラフとしてグラフを表示します。

**表示範囲（データ、イベント、Power ロググラフ）**

ダイアログボックスが開きます（255 ページの*［表示範囲］ダイアログボックス*を参照）。

**サイズ（データ、イベント、Power ロググラフ）**

グラフの縦のサイズを決定します。**小**、**中**、**大**から選択してください。

**［スタイル（データ、イベントおよび Power ロググラフ）］**

グラフのスタイルを選択します。以下から選択します。

**［棒グラフ］**：各ログが縦のバーとして表示されます。

**［列］**：各ログを列で表示します。

**［レベル］**：各ログが長方形のグラフで表示されます（オプションでカラー）。

**［線形］**：連続したログ間の細い線でグラフが表示されます。

すべてのグラフであらゆるスタイルが使用可能なわけではありません

**数値を表示（データ、イベント、および Power ロググラフ）**

グラフに加えて、変数の数値を表示します。

**数値を表示（イベントグラフ）**

イベントの値を表示します。

**16 進数（イベントグラフ）**

値の表示モードを決定します。16 進数か 10 進数を選択します。この設定は、［イベントログ］ウィンドウと［イベントログサマリ］ウィンドウの同じチャンネルの表示モードも制御する点に注意してください。

**ソースへ移動（共通）**

エディタウィンドウの対応するソースコードを表示します（該当する場合）。

**［設定ウィンドウ］を開く（Power ロググラフ）**

［Power ログの設定］ウィンドウを開きます。

**ソート基準（割込みグラフ）**

ID や名前を基準にエントリをソートします。新しい割込みが現れると、選択した順序がグラフ内で使用されます。

**グラフを選択（共通）**

［タイムライン］ウィンドウで表示するグラフを選択します。

**時間軸単位（共通）**

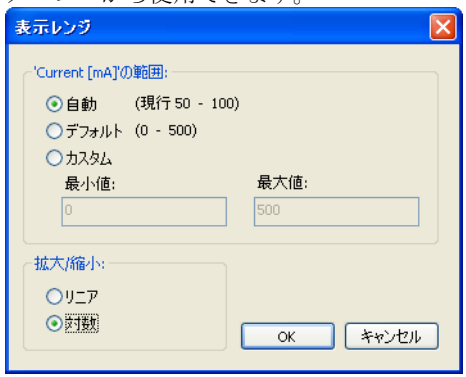
時間軸で使用する単位として、**秒**と**サイクル**のどちらかを選択します。

**プロファイル選択**

［関数プロファイラ］ウィンドウでプロファイリングの間隔を有効にします。このコマンドは、C-SPY ドライバが PC サンプリングをサポートしている場合にのみ使用できます。

## ［表示範囲］ダイアログボックス

**［表示範囲］**ダイアログボックスは、［タイムライン］ウィンドウの Power ロググラフまたはデータロググラフを右クリックして表示されるコンテキストメニューから使用できます。




このダイアログボックスを使用して、値の範囲を指定します。つまり、グラフの Y 軸の範囲です。

### 要件

以下のいずれかが必要です。

- C-SPY シミュレータ
- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY ST-LINK ドライバ

**範囲 ...**

表示された値の表示範囲を選択します。

**自動**

最小値や最大値の継続的に管理しつつ、実際に収集された値の範囲に基づいた範囲を使用します。現在算出されている範囲があれば、それが括弧内に表示されます。範囲は適度に*均等な*の限界値に丸められます。

**デフォルト**

データロググラフの場合 : 変数の範囲の値に基づいて範囲を使用します。たとえば、符号なしの 16 ビットの整数の場合、0–65535 です。

Power ロググラフの場合 : 測定用ハードウェアのプロパティに基づいた範囲を使用します。

**カスタム**

テキストボックスを使用して、明示的な範囲を指定します。

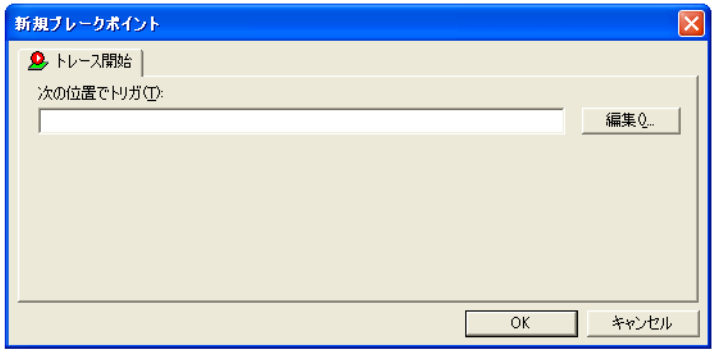
**スケール**

Y 軸のスケールタイプを選択します。

- **リニア**
- **対数**

## [トレース開始] ブレークポイントダイアログボックス

**[トレース開始]** ダイアログボックスは、[ブレークポイント] ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。




このダイアログボックスを使用して、トレースデータの収集を開始するトレース開始ブレークポイントを設定します。特定の範囲についてのみトレー

スデータを収集するには、データの収集を停止するトレース停止ブレークポイントも設定する必要があります。

258 ページの *［トレース停止］ブレークポイントダイアログボックス*も参照してください。

**トレース開始ブレークポイントを設定するには、以下の手順に従います。**

1. エディタまたは［逆アセンブリ］ウィンドウで、右クリックしてコンテキストメニューから**［トレース開始］**を選択します。

   または、**［表示］> ［ブレークポイント］**を選択して、［ブレークポイント］ウィンドウを開きます。

2. ［ブレークポイント］ウィンドウで、右クリックして**［新規ブレークポイント］> ［トレース開始］**を選択します。

   既存のブレークポイントを変更するには、［ブレークポイント］ウィンドウでブレークポイントを選択し、コンテキストメニューから**［編集］**を選択します。

3. **［トリガ位置］**テキストボックスで、式や絶対アドレス、ソース位置を指定します。**[OK]** をクリックします。

4. ブレークポイントがトリガされると、トレースデータの収集が始まります。

### 要件

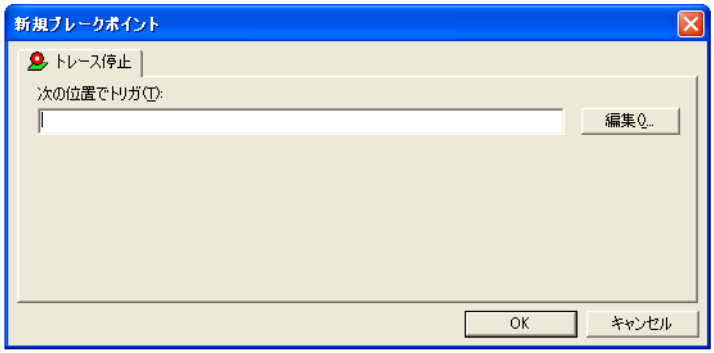
C-SPY シミュレータ。

### トリガ位置

ブレークポイントのコード位置を指定します。または、**［編集］**ボタンをクリックして**［位置入力］**ダイアログボックスを表示します（168 ページの *［位置入力］ダイアログボックス*を参照）。

## ［トレース停止］ブレークポイントダイアログボックス

［**トレース停止**］ダイアログボックスは、［ブレークポイント］ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。




このダイアログボックスを使用して、トレースデータの収集を停止するトレース停止ブレークポイントを設定します。特定範囲についてのみトレースデータを収集するには、データの収集を開始するトレース開始ブレークポイントを設定しなければならない場合があります。

256 ページの *［トレース開始］ブレークポイントダイアログボックス*も参照してください。

### トレース停止ブレークポイントを設定するには：

1 エディタまたは［逆アセンブリ］ウィンドウで、右クリックしてコンテキストメニューから［**トレース停止**］を選択します。

または、［**表示**］>［**ブレークポイント**］を選択して、［ブレークポイント］ウィンドウを開きます。

2 ［ブレークポイント］ウィンドウで、右クリックして［**新規ブレークポイント**］>［**トレース停止**］を選択します。

既存のブレークポイントを変更するには、［ブレークポイント］ウィンドウでブレークポイントを選択し、コンテキストメニューから［**編集**］を選択します。

3 ［**トリガ位置**］テキストボックスで、式や絶対アドレス、ソース位置を指定します。**[OK]** をクリックします。

4 ブレークポイントがトリガされると、トレースデータの収集が終了します。

### 要件

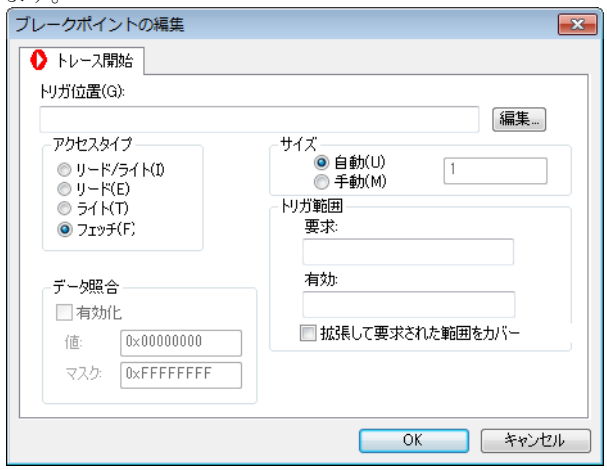
C-SPY シミュレータ。

### トリガ位置

ブレークポイントのコード位置を指定します。または、**[編集]** ボタンをクリックして **[位置入力]** ダイアログボックスを表示します（168 ページの *[位置入力] ダイアログボックス*を参照）。

## [トレース開始] ブレークポイントダイアログボックス（I-jet/JTAGjet および CMSIS-DAP）

**[トレース開始]** ダイアログボックスは、[ブレークポイント] ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。または、[エディタ] ウィンドウまたは [逆アセンブリ] ウィンドウで右クリックして、**[ブレークポイントの切替え（トレース開始）]** を選択することもできます。




このダイアログボックスを使用して、トレースデータの収集を開始するタイミングを決定する条件を設定します。トレース条件がトリガされると、トレースデータの収集が始まります。

### 要件

以下のいずれかが必要です。

- C-SPY CMSIS-DAP ドライバ
- C-SPY I-jet/JTAGjet ドライバ

### トリガ位置

トレースデータを収集するコードセクションの開始点を指定します。変数名やアドレス、サイクルカウンタの値を指定できます。

### アクセスタイプ

ブレークポイントをトリガするメモリアクセスの種類を選択します。

**リード / ライト**
指定された位置から読み取り / 書き込みを行います。

**リード**
指定された位置から読み取ります。

**ライト**
指定の位置に書き込みます。

**フェッチ**
実行位置のアドレスにおけるアクセス。

指定された種類のあらゆるアクセスによって、トレースデータの収集が有効になります。

### データ照合

アクセスされるデータの照合を有効にします。以下から選択します。

| | |
|---|---|
| **値** | データ値を指定します。 |
| **マスク** | 値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。 |

**［データ照合］**オプションを、リード / ライト、リードまたはライトのデータアクセスタイプと組み合わせて使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

**注 :［データ照合］**オプションは、Cortex-M デバイス使用時のみ使用可能です。Cortex-M デバイスについては、1 つのブレークポイントにのみ**［データ照合］**を設定できます。このようなブレークポイントでは、2 つのブレークポイントリソースを使用します。

**サイズ**

アドレス範囲のサイズを制御します。このサイズに達すると、トレースデータ収集の開始がトリガされます。以下から選択します。

**自動**

サイズを自動的に設定します。これは、［**トリガ位置**］に変数が含まれる場合に便利です。

**手動**

ブレークポイント範囲のサイズを手動で指定します。

**トリガ範囲**

要求された範囲とトレースデータの収集でカバーする有効範囲が表示されます。推奨される範囲は、［**トリガ位置**］と［**サイズ**］オプションによって指定された領域とまったく同じか、その内側です。

**拡張して要求された範囲をカバー**

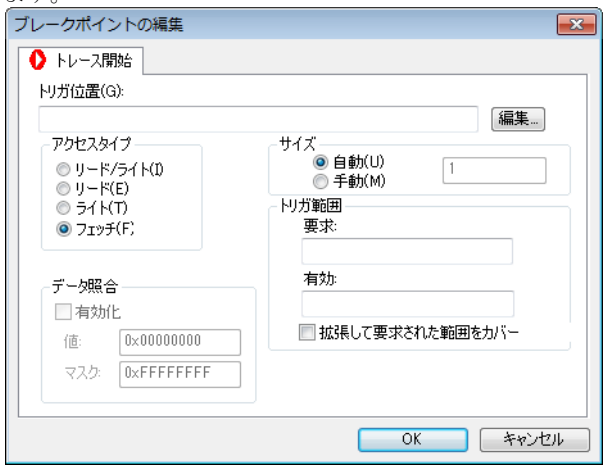
データ構造体がカバーされるように範囲を拡張します。ハードウェアブレークポイント装置で提供可能な範囲のサイズと合わないデータ構造（3 バイトなど）の場合、範囲はデータ構造全体を対象としません。範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

このオプションは、ARM7/9 デバイスでは有効になっていません。理由は、こうしたデバイスの範囲は常にデータ構造全体をカバーするためです。

## ［トレース停止］ブレークポイントダイアログボックス（I-jet/JTAGjet および CMSIS-DAP）

［**トレース停止**］ダイアログボックスは、［ブレークポイント］ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。または、［エディタ］ウィンドウまたは［逆アセンブリ］ウィンドウで右クリック

して、[ブレークポイントの切替え（トレース停止）]を選択することもできます。




このダイアログボックスを使用して、トレースデータの収集を停止するタイミングを決定する条件を設定します。トレース条件がトリガされると、レースデータの収集が停止します。

### 要件

以下のいずれかが必要です。

- C-SPY CMSIS-DAP ドライバ
- C-SPY I-jet/JTAGjet ドライバ

### トリガ位置

トレースデータを収集するコードセクションの終了点を指定します。変数名やアドレス、サイクルカウンタの値を指定できます。

### アクセスタイプ

ブレークポイントをトリガするメモリアクセスの種類を選択します。

**リード / ライト**
指定された位置から読み取り / 書き込みを行います。

**リード**
指定された位置から読み取ります。

**ライト**
指定の位置に書き込みます。

**フェッチ**

実行位置のアドレスにおけるアクセス。

指定された種類のあらゆるアクセスによって、トレースデータの収集が有効になります。

### データ照合

アクセスされるデータの照合を有効にします。以下から選択します。

| | |
|---|---|
| **値** | データ値を指定します。 |
| **マスク** | 値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。 |

**［データ照合］**オプションを、リード / ライト、リードまたはライトのデータアクセスタイプと組み合わせて使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

**注 :［データ照合］**オプションは、Cortex-M デバイス使用時のみ使用可能です。Cortex-M デバイスについては、1 つのブレークポイントにのみ**［データ照合］**を設定できます。このようなブレークポイントでは、2 つのブレークポイントリソースを使用します。

### サイズ

アドレス範囲のサイズを制御します。このサイズに達すると、トレースデータ収集の開始がトリガされます。以下から選択します。

**自動**

サイズを自動的に設定します。これは、**［トリガ位置］**に変数が含まれる場合に便利です。

**手動**

ブレークポイント範囲のサイズを手動で指定します。

### トリガ範囲

要求された範囲とトレースデータの収集でカバーする有効範囲が表示されます。推奨される範囲は、**［トリガ位置］**と**［サイズ］**オプションによって指定された領域とまったく同じか、その内側です。

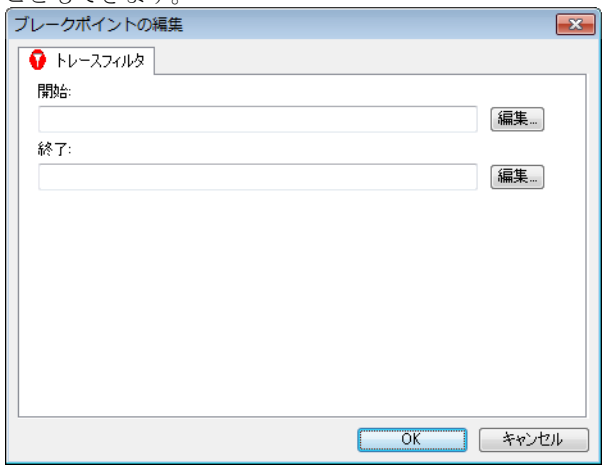
**拡張して要求された範囲をカバー**

データ構造体がカバーされるように範囲を拡張します。ハードウェアブレークポイント装置で提供可能な範囲のサイズと合わないデータ構造（3 バイトなど）の場合、範囲はデータ構造全体を対象としません。範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

このオプションは、ARM7/9 デバイスでは有効になっていません。理由は、こうしたデバイスの範囲は常にデータ構造全体をカバーするためです。

## ［トレースフィルタ］ブレークポイントダイアログボックス (I-jet/JTAGjet)

**［トレースフィルタ］**ダイアログボックスは、［ブレークポイント］ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。または、［エディタ］ウィンドウまたは［逆アセンブリ］ウィンドウで右クリックして、**［ブレークポイントの切替え（トレースフィルタ）］**を選択することもできます。




このダイアログボックスを使用して、トレースデータの収集を開始するタイミングを決定する条件を設定します。トレース条件がトリガされると、トレースデータの収集が始まります。

**要件**

以下のいずれかが必要です。

- C-SPY CMSIS-DAP ドライバ
- C-SPY I-jet/JTAGjet ドライバ

**開始**

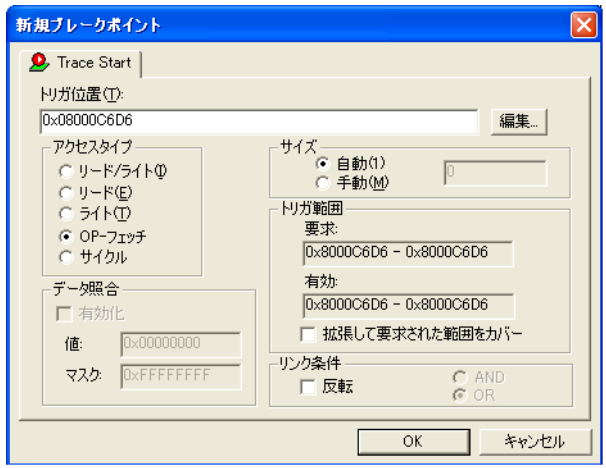
トレースデータを収集する対象のコードセクションの開始位置を指定します。または、**[編集]** ボタンをクリックして **[位置入力]** ダイアログボックスを表示します（168 ページの *[位置入力] ダイアログボックス*を参照）。

**終了**

トレースデータを収集するコードセクションの終了点を指定します。または、**[編集]** ボタンをクリックして **[位置入力]** ダイアログボックスを表示します（168 ページの *[位置入力] ダイアログボックス*を参照）。

## [トレース開始] ブレークポイントダイアログボックス (J-Link/J-Trace)

**[トレース開始]** ダイアログボックスは、[ブレークポイント] ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。または、[エディタ] ウィンドウまたは [逆アセンブリ] ウィンドウで右クリックして、**[ブレークポイントの切替え（トレース開始）]** を選択することもできます。



このダイアログボックスを使用して、トレースデータの収集を開始するタイミングを決定する条件を設定します。トレース条件がトリガされると、トレースデータの収集が始まります。

### 要件

C-SPY J-Link/J-Trace ドライバ。

### トリガ位置

トレースデータを収集するコードセクションの開始点を指定します。変数名やアドレス、サイクルカウンタの値を指定できます。

### サイズ

アドレス範囲のサイズを制御します。このサイズに達すると、トレースデータ収集の開始がトリガされます。以下から選択します。

**自動**

サイズを自動的に設定します。これは、［**トリガ位置**］に変数が含まれる場合に便利です。

**手動**

ブレークポイント範囲のサイズを手動で指定します。

### トリガ範囲

要求された範囲とトレースデータの収集でカバーする有効範囲が表示されます。推奨される範囲は、［**トリガ位置**］と［**サイズ**］オプションによって指定された領域とまったく同じか、その内側です。

**拡張して要求された範囲をカバー**

データ構造体がカバーされるように範囲を拡張します。ハードウェアブレークポイント装置で提供可能な範囲のサイズと合わないデータ構造（3 バイトなど）の場合、範囲はデータ構造全体を対象としません。範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

このオプションは、ARM7/9 デバイスでは有効になっていません。理由は、こうしたデバイスの範囲は常にデータ構造全体をカバーするためです。

### アクセスタイプ

ブレークポイントをトリガするメモリアクセスの種類を選択します。

**リード / ライト**

指定された位置から読み取り / 書き込みを行います。

**リード**
指定された位置から読み取ります。

**ライト**
指定の位置に書き込みます。

**OP フェッチ**
実行位置のアドレスにおけるアクセス。

**サイクル**
実行開始点から数えた特定の時点でのカウンタサイクル数。このオプションは、Cortex-M デバイスでのみ使用できます。

指定された種類のあらゆるアクセスによって、トレースデータの収集が有効になります。

### データ照合

アクセスされるデータの照合を有効にします。［**データ照合**］オプションを、リード / ライト、リードまたはライトのデータアクセスタイプと組み合わせて使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

| | |
|---|---|
| **値** | データ値を指定します。 |
| **マスク** | 値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。 |

［**データ照合**］オプションは、J-Link/J-Trace でのみ使用可能です（Cortex-M デバイス使用時のみ）。

**注 :** Cortex-M デバイスについては、1 つのブレークポイントにのみ［**データ照合**］を設定できます。このようなブレークポイントでは、2 つのブレークポイントリソースを使用します。

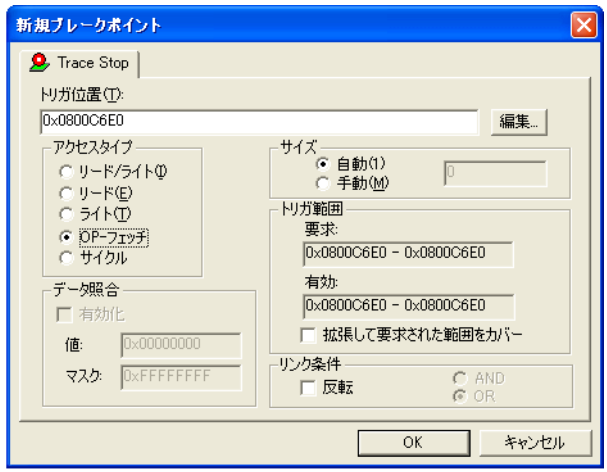
### リンク条件

**AND** と **OR** を使用して、トレース条件の組合せ方法を指定します。リンク条件 **AND** を持つ条件を、リンク条件 **OR** を持つ条件と組み合わせる場合、**AND** が優先します。オプション［**反転**］はトレース条件を反転させ、各トレースフィルタ条件に対して個別に作用します。あるトレースの開始条件または停止条件が反転されると、他のすべてもそうなります。反転されたトレース開始条件または停止条件は、アプリケーションコードのこのセクションを除いて、トレースデータの収集がすべての場所で実行されることを意味します。

ARM7/9 デバイスの場合、トレースフィルタは OR アルゴリズムを使用して結合されます。トレースフィルタを反転するには、[反転] オプションを使用します。すべてのフィルタが対象になります。トレースフィルタは、AND アルゴリズムを使用して開始トリガおよび停止トリガと結合されます（存在する場合）。

## [トレース停止] ブレークポイントダイアログボックス (J-Link/J-Trace)

[トレース停止] ダイアログボックスは、[ブレークポイント] ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。または、[エディタ] ウィンドウまたは [逆アセンブリ] ウィンドウで右クリックして、[ブレークポイントの切替え（トレース停止）] を選択することもできます。




トレース条件がトリガされると、何らかのさらなる指示についてトレースデータの収集が実行され、続いて収集が停止します。

### 要件

C-SPY J-Link/J-Trace ドライバ。

### トリガ位置

トレースデータを収集するコードセクションの停止点を指定します。変数名やアドレス、サイクルカウンタの値を指定できます。

### サイズ

アドレス範囲のサイズを制御します。このサイズに達すると、トレースデータ収集の停止がトリガされます。以下から選択します。

**自動**

サイズを自動的に設定します。これは、［**トリガ位置**］に変数が含まれる場合に便利です。

**手動**

ブレークポイント範囲のサイズを手動で指定します。

### トリガ範囲

要求された範囲とトレースデータの収集でカバーする有効範囲が表示されます。推奨される範囲は、［**トリガ位置**］と［**サイズ**］オプションによって指定された領域とまったく同じか、その内側です。

**拡張して要求された範囲をカバー**

データ構造体がカバーされるように範囲を拡張します。ハードウェアブレークポイント装置で提供可能な範囲のサイズと合わないデータ構造（3 バイトなど）の場合、範囲はデータ構造全体を対象としません。範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

このオプションは、ARM7/9 デバイスでは有効になっていません。理由は、こうしたデバイスの範囲は常にデータ構造全体をカバーするためです。

### アクセスタイプ

ブレークポイントをトリガするメモリアクセスの種類を選択します。

**リード / ライト**

指定された位置から読み取り / 書き込みを行います。

**リード**

指定された位置から読み取ります。

**ライト**

指定の位置に書き込みます。

**OP フェッチ**

実行位置のアドレスにおけるアクセス。

**サイクル**

実行開始点から数えた特定の時点でのカウンタサイクル数。このオプションは、Cortex-M デバイスでのみ使用できます。

指定した種類のアクセスがあると、トレースデータの収集が停止します。

### データ照合

アクセスされるデータの照合を有効にします。［**データ照合**］オプションを、リード / ライト、リードまたはライトのデータアクセスタイプと組み合わせて使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

| | |
|---|---|
| **値** | データ値を指定します。 |
| **マスク** | 値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。 |

［**データ照合**］オプションは、J-Link/J-Trace でのみ使用可能です（Cortex-M デバイス使用時のみ）。

**注 :** Cortex-M デバイスについては、1 つのブレークポイントにのみ［**データ照合**］を設定できます。このようなブレークポイントでは、2 つのブレークポイントリソースを使用します。

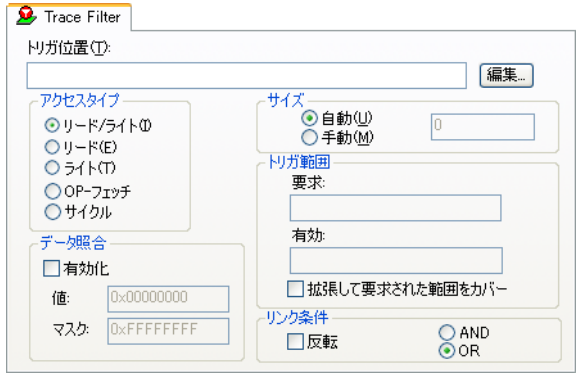
### リンク条件

**AND** と **OR** を使用して、トレース条件の組合せ方法を指定します。リンク条件 **AND** を持つ条件を、リンク条件 **OR** を持つ条件と組み合わせる場合、**AND** が優先します。オプション［**反転**］はトレース条件を反転させ、各トレースフィルタ条件に対して個別に作用します。あるトレースの開始条件または停止条件が反転されると、他のすべてもそうなります。反転されたトレース開始条件または停止条件は、アプリケーションコードのこのセクションを除いて、トレースデータの収集がすべての場所で実行されることを意味します。

ARM7/9 デバイスの場合、トレースフィルタは OR アルゴリズムを使用して結合されます。トレースフィルタを反転するには、［**反転**］オプションを使用します。すべてのフィルタが対象になります。トレースフィルタは、AND アルゴリズムを使用して開始トリガおよび停止トリガと結合されます（存在する場合）。

# ［トレースフィルタ］ブレークポイントダイアログボックス (J-Link/J-Trace)

［**トレースフィルタ**］ダイアログボックスは、［ブレークポイント］ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。または、［エディタ］ウィンドウまたは［逆アセンブリ］ウィンドウで右クリックして、［**ブレークポイントの切替え（トレースフィルタ）**］を選択することもできます。




トレース条件がトリガされると、何らかのさらなる指示についてトレースデータの収集が実行され、続いて収集が停止します。

### 要件

C-SPY J-Link/J-Trace ドライバ。

### トリガ位置

ブレークポイントのコード位置を指定します。または、［**編集**］ボタンをクリックして［**位置入力**］ダイアログボックスを表示します（168 ページの *［位置入力］ダイアログボックス*を参照）。

### サイズ

フィルタトレースを有効にするアドレス範囲のサイズを制御します。以下から選択します。

**自動**

サイズを自動的に設定します。これは、［**トリガ位置**］に変数が含まれる場合に便利です。

**手動**

ブレークポイント範囲のサイズを手動で指定します。

### トリガ範囲

要求された範囲とトレースデータの収集でカバーする有効範囲が表示されます。推奨される範囲は、［**トリガ位置**］と［**サイズ**］オプションによって指定された領域とまったく同じか、その内側です。

**拡張して要求された範囲をカバー**

データ構造体がカバーされるように範囲を拡張します。ハードウェアブレークポイント装置で提供可能な範囲のサイズと合わないデータ構造（3 バイトなど）の場合、範囲はデータ構造全体を対象としません。範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

このオプションは、ARM7/9 デバイスでは有効になっていません。理由は、こうしたデバイスの範囲は常にデータ構造全体をカバーするためです。

### アクセスタイプ

ブレークポイントをトリガするメモリアクセスの種類を選択します。

**リード / ライト**

指定された位置から読み取り / 書き込みを行います。

**リード**

指定された位置から読み取ります。

**ライト**

指定の位置に書き込みます。

**OP フェッチ**

実行位置のアドレスにおけるアクセス。

**サイクル**

実行開始点から数えた特定の時点でのカウンタサイクル数。このオプションは、Cortex-M デバイスでのみ使用できます。

### データ照合

アクセスされるデータの照合を有効にします。［**データ照合**］オプションを、リード / ライト、リードまたはライトのデータアクセスタイプと組み合わせて使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

**値**　　データ値を指定します。

**マスク**　　値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。

［**データ照合**］オプションは、J-Link/J-Trace でのみ使用可能です（Cortex-M デバイス使用時のみ）。

**注 :** Cortex-M デバイスについては、1 つのブレークポイントにのみ［**データ照合**］を設定できます。このようなブレークポイントでは、2 つのブレークポイントリソースを使用します。

**リンク条件**

**AND** と **OR** を使用して、トレース条件の組合せ方法を指定します。リンク条件 **AND** を持つ条件を、リンク条件 **OR** を持つ条件と組み合わせる場合、**AND** が優先します。オプション［**反転**］はトレース条件を反転させ、各トレースフィルタ条件に対して個別に作用します。あるトレースの開始条件または停止条件が反転されると、他のすべてもそうなります。反転されたトレース開始条件または停止条件は、アプリケーションコードのこのセクションを除いて、トレースデータの収集がすべての場所で実行されることを意味します。

ARM7/9 デバイスの場合、トレースフィルタは OR アルゴリズムを使用して結合されます。トレースフィルタを反転するには、［**反転**］オプションを使用します。すべてのフィルタが対象になります。トレースフィルタは、AND アルゴリズムを使用して開始トリガおよび停止トリガと結合されます（存在する場合）。

## ［トレース式］ウィンドウ

［トレース式］ウィンドウは、［トレース］ウィンドウツールバーから使用できます。


このウィンドウを使用して、トレースデータを収集する特定の変数（または式）などを指定します。

**要件**

C-SPY シミュレータ。

**ツールバー**

ツールバーのボタンによって、式の表示順序を変更します。

**上向きの矢印**
選択された行を上に移動します。

**下向きの矢印**
選択された行を下に移動します。

**表示エリア**

表示エリアを使用して、トレースデータを収集する式を指定します。

**式**
データを収集する元の任意の式を指定します。変数やレジスタなど、評価可能な式を指定できます。

**フォーマット**
各式で使用される表示フォーマットが表示されます。表示形式はコンテキストメニューから変更できます。

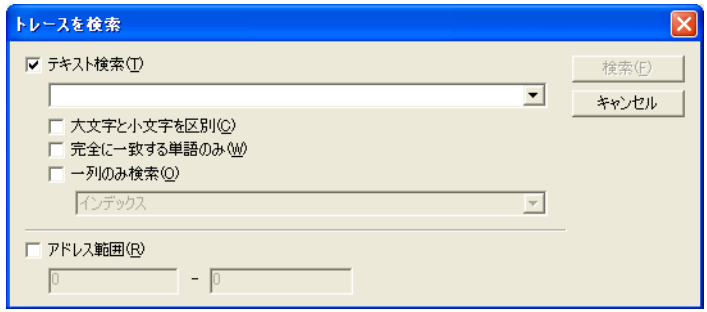
このエリアの各行は、［トレース］ウィンドウに追加列として表示されます。

## ［トレースを検索］ダイアログボックス

**［トレースを検索］**ダイアログボックスは、［トレース］ウィンドウで**［検索］**ボタンをクリックするか、**［編集］**>**［検索と置換］**>**［検索］**を選択すると使用できます。

**［編集］**>**［検索と置換］**>**［検索］**コマンドはコンテキスト依存型であることに注意してください。このコマンドの操作時の現在のウィンドウが、［トレース］ウィンドウであれば**［トレースを検索］**ダイアログボックスが表示され、

エディタウィンドウであれば［**検索**］ダイアログボックスがそれぞれ表示されます。




このダイアログボックスを使用して、トレースデータ内の高度な検索の検索基準を指定します。

検索結果は［トレースを検索］ウィンドウ（［**表示**］>［**メッセージ**］コマンドを選択）に表示されます（276 ページの *［トレースを検索］ウィンドウ*を参照）。

224 ページの *トレースデータの検索*も参照してください。

## 要件

以下のいずれかが必要です。

- C-SPY シミュレータ
- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY CMSIS-DAP ドライバ
- C-SPY ST-LINK ドライバ

## テキスト検索

検索する文字列を指定します。検索基準を以下から選択します。

**大文字 / 小文字の区別**

指定されたテキストの大文字と小文字が完全に一致するものだけを検索します。このオプションを指定しない場合は、`int` を検索すると、`INT`、`Int` なども検索されます。

**完全に一致する単語のみ**

単語として一致する箇所だけを検索します。このオプションを指定しない場合は、`int` を検索すると、`print`、`sprintf` なども検索されます。

**一列のみ検索**

ドロップダウンリストから選択した列だけを検索します。

**アドレス範囲**

表示または検索するアドレス範囲を指定します。アドレス範囲内のトレースデータが表示されます。**［テキスト検索］**フィールドにテキスト文字列も指定すると、アドレス範囲内でテキスト文字列を検索します。

## ［トレースを検索］ウィンドウ

［トレースを検索］ウィンドウは、**［表示］>［メッセージ］**メニューから利用できます。または、**［トレースを検索］**ダイアログボックスを使用して検索を実行するか、またはエディタウィンドウのコンテキストメニューから**［トレースを検索］**コマンドを使用して検索を実行すると、自動的に表示されます。


このウィンドウには、トレースデータの検索結果が表示されます。［トレースを検索］ウィンドウで項目をダブルクリックすると、［トレース］ウィンドウに同じ項目が表示されます。

トレースデータを表示するには、**［トレースを検索］**ダイアログボックスで検索基準を指定する必要があります（274 ページの *［トレースを検索］ダイアログボックス*を参照）。

詳細については、224 ページの *トレースデータの検索*を参照してください。

**要件**

以下のいずれかが必要です。

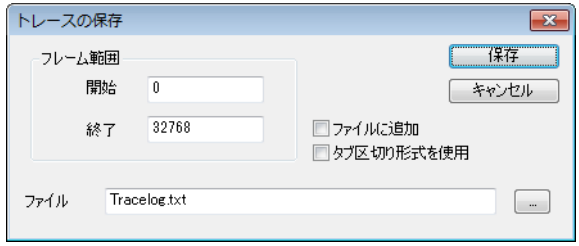
- C-SPY シミュレータ
- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY CMSIS-DAP ドライバ
- C-SPY ST-LINK ドライバ

### 表示エリア

［トレースを検索］ウィンドウは［トレース］ウィンドウと非常によく似ており、表示される列とデータは同じですが、表示される行は指定された検索基準に一致した行*だけ*です。

## ［トレースの保存］ダイアログボックス

**［トレースの保存］**ダイアログボックスは、ドライバ固有のメニューと、［トレース］ウィンドウおよび［SWO トレース］ウィンドウから使用できます。




### 要件

以下のいずれかが必要です。

- C-SPY J-Link/J-Trace ドライバ
- C-SPY ST-LINK ドライバ

### インデックス範囲

フレームの範囲をファイルに保存します。開始インデックスと終了インデックスを指定します（［トレース］ウィンドウのインデックス列の値）。

### ファイルに追加

トレースデータを既存ファイルに追加します。

### タブ区切りフォーマットを使用

列の内容を、スペースではなくタブ区切りで保存します。

### ファイル

トレースデータのファイルを指定します。

# プロファイリング

- プロファイラの概要
- プロファイラの使用
- プロファイラについてのリファレンス情報

## プロファイラの概要

以下のトピックについて説明します:

- プロファイラの用途
- プロファイラの概要について
- プロファイラの使用に関する要件

### プロファイラの用途

*関数プロファイリング*を行うと、実行中に最も多くの時間を費やす、ソースコードでの関数の検索が簡単になります。コードを最適化する際にはこれらの関数に注目します。簡単に関数を最適化するには、実行速度最適化を指定してコンパイルします。別の方法として、関数で使用するデータをより効率のよいメモリ内に移動することもできます。効率的なメモリの使用方法の詳細については、*『ARM 用 IAR C/C++ 開発ガイド』*を参照してください。

または、*［フィルタリング］*を使用すると、たとえば個々の関数をプロファイリングされないように除外できます。コードの特定部分のみをプロファイルする場合は、［タイムライン］ウィンドウを使用して *［間隔］*を選択すると、それに対して C-SPY でプロファイリング情報が生成されます。

*命令プロファイリング*は、特にアセンブラのソースコードを非常に詳細なレベルで微調整するときに役立ちます。命令プロファイリングは、C/C++ ソースコードのコンパイルで時間がかかった部分を把握するのに便利です。パフォーマンス向上のために、どのように書き直したらよいかのヒントになります。

### プロファイラの概要について

*関数プロファイリング*情報は［関数プロファイラ］ウィンドウに表示されます。これはアプリケーションでの関数のタイミング情報です。プロファイリングは、ウィンドウのツールバーにあるボタンを使用して明示的に有効にす

る必要があります。有効にした後は、無効にするまではその状態に保持されます。

*命令プロファイリング*情報は［逆アセンブル］ウィンドウに表示されます。これは各命令の実行回数です。

## ソースのプロファイリング

プロファイラは、さまざまなメカニズムや*ソース*を使用してプロファイリング情報を収集できます。使用可能なトレースソースの特長によっては、1 つまたは複数のソースをプロファイリングで使用できます。

- トレース（呼出し）

  すべての関数の呼出しとリターンを判定するために、命令のトレース全体（ETM トレース）が解析されます。収集された命令シーケンスが不完全だったり不連続の場合（ETM トレースの使用時に起こるときがあります）、プロファイリング情報は正確ではなくなります。

- トレース（フラット）/ サンプリング

  命令フルトレース（ETM トレース）または各 PC サンプル（SWO トレースから）の各命令は、関数呼出しやリターンに関係なく、対応する関数またはコードフラグメントに割り当てられます。RTOS を使用していたり、完全なデバッグ情報を持たないコードをプロファイリングする場合など、アプリケーションが通常の呼出し / リターンシーケンスの動作を見せないときにこれは非常に便利です。

- ブレークポイント

  プロファイラは、関数のエントリポイントごとにブレークポイントを設定します。実行中に、プロファイラは各ブレークポイントに当たるたびに関数呼出しおよびリターンについての情報を収集します。これは、ハードウェアで大量のブレークポイントがサポートされていることが前提で、実行のパフォーマンスに多大な影響を与えます。

## Power サンプリング

一部のデバッグプローブでは、開発ボードまたはボード上のコンポーネントの電力消費のサンプリングがサポートされています。各サンプルは PC サンプルに関連付けられ、サンプル時に先立つわずかな間隔の電力消費（実際のところは電流）を表します。プロファイルが *Power サンプリング*を使用するように設定されている場合、［プロファイラ］ウィンドウに追加の列が表示されます。各 Power サンプリングは、通常の PC サンプリングの場合と同じように関数またはコードフラグメントに関連付けられます。ただし、サンプルに対応するすべてのエネルギーが、その関数やコードフラグメントに関するものということではありません。Power サンプリングと命令実行のタイムスケールは大きく異なります。電力測定を 1 回行う間に、通常 CPU は命令を数千回実行しています。Power サンプリングは、統計ツールです。

## プロファイラの使用に関する要件

C-SPY シミュレータはプロファイラをサポートしており、使用にあたって特定の要件はありません。

ハードウェアデバッガシステムでプロファイラを使用するには、以下のいずれかが必要です。

- I-jet または I-jet Trace インサーキットデバッグプローブ、JTAGjet、J-Link、J-Trace、ST-LINK デバッグプローブ、プローブとターゲットシステム間の SWD/SWO インタフェース（Cortex-M デバイスに基づくもの）。
- JTAGjet-Trace インサーキットデバッグプローブおよびETM トレース機能を持つ ARM デバイス。
- J-Trace デバッグプローブと ARM7/9 またはETM トレースを備えた Cortex-M デバイス。

次の表は、C-SPY ドライバのプロファイリングのサポート一覧です。

| ターゲットシステム | トレース（呼出し） | トレース（フラット） | サンプリング | 電源 |
|---|---|---|---|---|
| C-SPY シミュレータ | X | X | -- | -- |
| CMSIS-DAP | X | X | -- | -- |
| I-jet | X | X | X[1] | X |
| I-jet Trace | X | X | X[1] | X |
| JTAGjet/JTAGjet-Trace | X | X | -- | -- |
| J-Link | X | X | X[1] | -- |
| J-Link Ultra | X | X | X[1] | X[2] |
| J-Trace | X | X | X[1] | -- |
| RDI | -- | -- | -- | -- |
| Macraigor | -- | -- | -- | -- |
| GDB サーバ | -- | -- | -- | -- |
| ST-LINK | -- | -- | X[1] | -- |
| TI Stellaris | -- | -- | -- | -- |
| TI XDS | -- | -- | -- | -- |
| Angel | -- | -- | -- | -- |
| IAR ROM モニタ | -- | -- | -- | -- |

*表 12: C-SPY ドライバのプロファイリングのサポート*

1 SWO をサポートする Cortex-M デバイスのみ。

2 SWO トレースが必要。

# プロファイラの使用

以下のタスクについて説明します:

- 関数レベルでプロファイラを使用するにあたって
- プロファイリングデータの解析
- 命令レベルでプロファイラを使用するにあたって
- プロファイリング情報の間隔を選択する

## 関数レベルでプロファイラを使用するにあたって

### 関数プロファイリング情報を［関数プロファイラ］ウィンドウに表示するには:

1 以下のオプションを使用してアプリケーションをビルドします:

| カテゴリ | 設定 |
|---|---|
| C/C++ コンパイラ | ［出力］>［デバッグ情報の生成］ |
| リンカ | ［出力］>［出力ファイルにデバッグ情報を含める］ |

*表 13: プロファイラを有効にするためのプロジェクトオプション*

2 関数プロファイリングのプロファイラを設定するには、以下の手順を実行します。

- ETM トレースを使用する場合、［**トレース設定**］ダイアログボックスで［**サイクルアキュレート・トレース**］オプションが選択されていることを確認します。
- SWD/SWO インタフェースを使用する場合、設定は特に必要ありません。

3 アプリケーションをビルドしてC-SPY を起動するときは、*［C-SPY ドライバ］*>［**関数プロファイラ**］を選択して［関数プロファイラ］ウィンドウを開き、［**有効化**］ボタンをクリックしてプロファイラを有効にします。または、［関数プロファイラ］ウィンドウを右クリックして表示されるコンテキストメニューで［**有効化**］を選択します。

4 アプリケーションの実行を開始して、プロファイリング情報を収集します。

5 プロファイリング情報が［関数プロファイラ］ウィンドウに表示されます。ソートするには、対象の列見出しをクリックします。

6 新しいサンプリングを開始する場合は、［**クリア**］ボタンをクリックして（またはコンテキストメニューを使用して）、データをクリアします。

## プロファイリングデータの解析

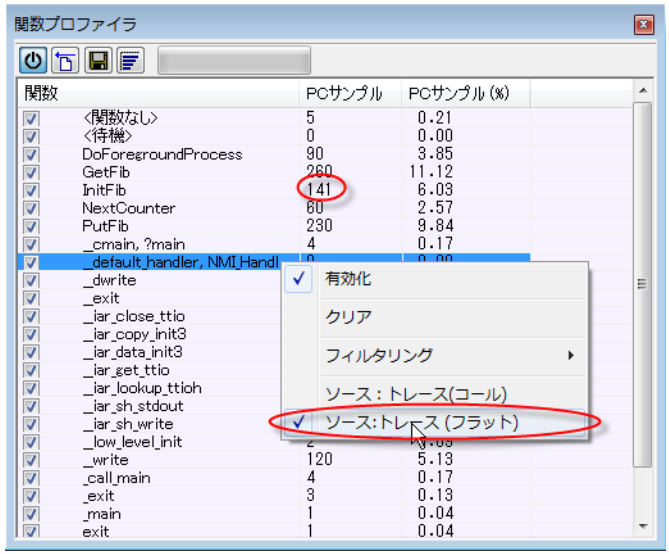
ここでは、データの解析方法の例を紹介します。

最初の図は、[ソース：トレース（呼出し)] を使用したプロファイリングの結果を示します。プロファイラはプログラムの流れに沿って、関数の入口と出口を検出します。

- **InitFib** 関数の場合、**フラット時間** 231 は関数自体の内部で消費された時間です。
- **InitFib** 関数の場合、**累積時間** 487 は、InitFib が呼出すすべての関数も含めて、関数自体の内部で消費された時間です。
- **InitFib/GetFib** 関数の場合、**累積時間** 256 は、**GetFib** 内で消費された時間です（ただし、**InitFib** から呼出されたときのみ)。これには **GetFib** が呼出すすべての関数が含まれます。
- さらにデータを見ていくと、**GetFib** 関数が別に示され、そのサブ関数がすべて表示されています（この場合は該当なし)。


2 番目の図は、[ソース：トレース（フラット)] を使用したプロファイリングの結果を示します。この場合、プロファイラはプログラムの流れに従わず、PC のアドレスが関数のスコープ内にあるかどうかだけが検出されます。トレースデータが不十分な場合、データに軽度のエラーが含まれることがあります。

**InitFib** 関数の場合、**フラット時間** 231 は関数自体の内部で消費された回数（ヒット数）を示します。

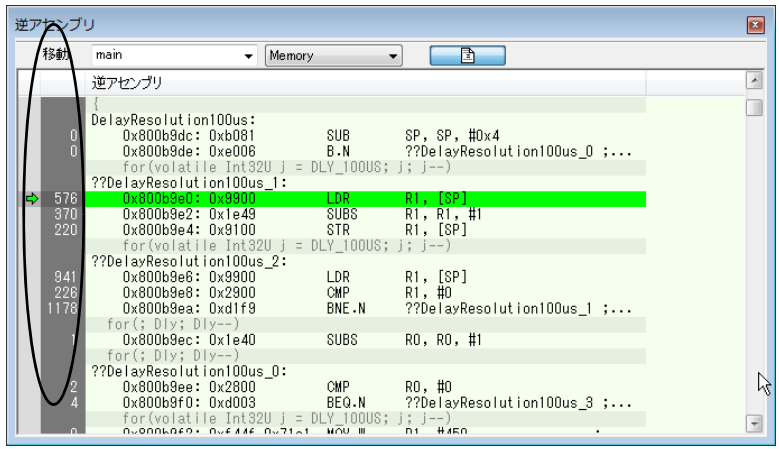



デバッグプローブ中に有効なデータを確保するには、最大のトレースバッファサイズを使用し、コード内にブレークポイントを設定して、バッファがフルになる前に実行を停止するようにしてください。

### 命令レベルでプロファイラを使用するにあたって

#### 命令プロファイリング情報を［逆アセンブリ］ウィンドウに表示するには：

1. アプリケーションをビルドして C-SPY を起動したら、**［表示］>［逆アセンブリ］**を選択して［逆アセンブリ］ウィンドウを開き、［逆アセンブリ］ウィンドウの左端を右クリックすると表示されるコンテキストメニューで**［命令プロファイリング］>［有効化］**を選択します。
2. プロファイリング情報を表示するために、コンテキストメニューで**［表示］**コマンドが選択されていることを確認します。
3. アプリケーションの実行を開始して、プロファイリング情報を収集します。
4. プログラムの終了に到達した、ブレークポイントがトリガされたなどの理由で実行が停止したときは、ウィンドウの左側の余白で命令レベルのプロファイリング情報を確認できます。




命令ごとに、実行された回数が表示されます。

命令プロファイリングでは、関数プロファイラと同じソースを使用するよう試みます。関数プロファイラがオンでない場合、命令プロファイラは最初のトレースを使用してから、PC サンプリングをソースとして使用するよう試みます。プロファイラのウィンドウで使用できるコンテキストメニューから、使用するソースを変更できます。

## プロファイリング情報の間隔を選択する

通常プロファイラは、受け取るすべての PC サンプルから情報を算出し、プロファイリング情報を明示的に消去するまで情報を蓄積します。ただし、プロファイラが PC サンプルを算出する間隔を選択できます。この関数は、I-jet および I-jet Trace インサーキットデバッグプローブ、JTAGjet デバッグプローブ、J-Link プローブ、J-Trace プローブ、ST-LINK プローブでサポートされています。

### 時間間隔を選択するには：

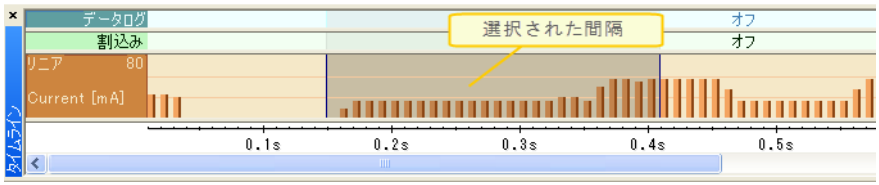
1 ［C-SPY ドライバ］メニューで**［関数プロファイラ］**を選択します。

2 ［関数プロファイラ］ウィンドウで右クリックして、コンテキストメニューから**［ソース：サンプリング］**を選択します。

3 アプリケーションを実行してサンプルを収集します。

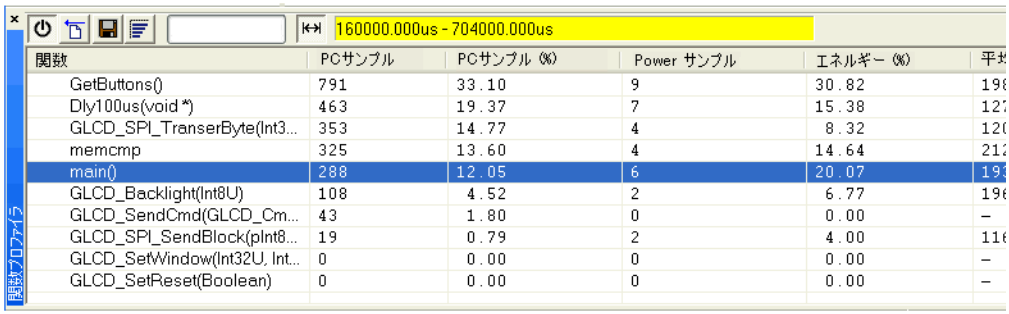
4 *［C-SPY ドライバ］*>［タイムライン］を選択します。

5 ［タイムライン］ウィンドウでクリックし、ドラッグして間隔を選択します。




6 選択された間隔で、右クリックしてコンテキストメニューから**［プロファイラの選択］**を選択します。

［関数プロファイラ］ウィンドウに選択した間隔のプロファイリング情報が表示されます。




| 関数 | PCサンプル | PCサンプル (%) | Power サンプル | エネルギー (%) | 平均 |
|---|---|---|---|---|---|
| GetButtons() | 791 | 33.10 | 9 | 30.82 | 198 |
| Dly100us(void *) | 463 | 19.37 | 7 | 15.38 | 127 |
| GLCD_SPI_TranserByte(Int3... | 353 | 14.77 | 4 | 8.32 | 120 |
| memcmp | 325 | 13.60 | 4 | 14.64 | 212 |
| main() | 288 | 12.05 | 6 | 20.07 | 191 |
| GLCD_Backlight(Int8U) | 108 | 4.52 | 2 | 6.77 | 196 |
| GLCD_SendCmd(GLCD_Cm... | 43 | 1.80 | 0 | 0.00 | – |
| GLCD_SPI_SendBlock(pInt8... | 19 | 0.79 | 2 | 4.00 | 116 |
| GLCD_SetWindow(Int32U, Int... | 0 | 0.00 | 0 | 0.00 | – |
| GLCD_SetReset(Boolean) | 0 | 0.00 | 0 | 0.00 | – |

7 **［フル / 間隔プロファイリング］**ボタンをクリックして、フルプロファイリング表示を切り替えます。

## プロファイラについてのリファレンス情報

リファレンス情報：

- 287 ページの*［関数プロファイラ］ウィンドウ*

関連項目：

- 85 ページの*逆アセンブリウィンドウ*
- 229 ページの*［ETM トレース設定］ダイアログボックス (J-Link/J-Trace)*
- 226 ページの*［ETM トレース設定］ダイアログボックス*
- 231 ページの*［SWO トレースウィンドウ設定］ダイアログボックス*
- 233 ページの*［SWO 設定］ダイアログボックス*

# ［関数プロファイラ］ウィンドウ

［関数プロファイラ］ウィンドウは、C-SPY ドライバメニューから使用できます。


このウィンドウには関数プロファイラの情報が表示されます。

トレース（フラット）が選択されている場合、ウィンドウの左端の各行にチェックボックスが表示されます。これらのチェックボックスを使用して、プロファイリングに行を含めたり、除外します。除外された行は灰色表示されますが、削除はされません。

## 要件

以下のいずれかが必要です。

- C-SPY シミュレータ
- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY CMSIS-DAP ドライバ
- C-SPY ST-LINK ドライバ

## ツールバー

ツールバーの内容は以下のとおりです。

**有効 / 無効**

プロファイラを有効 / 無効にします。

**クリア**

すべてのプロファイリングデータをクリアします。

**保存**

標準の［**名前を付けて保存**］ダイアログボックスを開きます。ウィンドウの内容をタブ区切りでファイルに保存できます。展開されていない行のみがリストファイルに含まれます。

**グラフィック表示**

パーセント値を表す列の値をグラフィカルバーにオーバレイします。

***進行度バー***

処理中のプロファイリングデータのバックログを表示します。受信データのレートが、データを処理するプロファイラのレートより高い場合は、バックログが蓄積されます。進行度バーは、プロファイラがデータ処理を実行中であることを示します。また、プロファイラがどのぐらいの速度で処理中か、おおよその速度も示します。プロファイラは特定のレートでデータを処理し、ターゲットシステムは違うレートでデータを提供するため、処理待ちのデータ量は増減する可能性があることに注意してください。進行度バーは、それに従って伸縮します。

**間隔モード**

選択した間隔のプロファイリングと完全なプロファイリングを切り替えます。このツールバーのボタンは、PC サンプリングがデバッグプローブでサポートされている場合にのみ使用できます。

使用する C-SPY ドライバでどのビューがサポートされているかについては、281 ページの*プロファイラの使用に関する要件*を参照してください。

***ステータスフィールド***

選択した間隔の範囲が表示されます。すなわち、プロファイルされた選択範囲です。間隔プロファイリングモードが有効な場合、このフィールドは黄色です。このフィールドは、PC サンプリングがデバッグプローブ（SWO トレース）でサポートされている場合にのみ使用できます。

使用する C-SPY ドライバでどのビューがサポートされているかについては、281 ページの*プロファイラの使用に関する要件*を参照してください。

## 表示エリア

表示エリアの内容は、プロファイリング情報に使用されるソースによって決まります。

- ブレークポイントとトレース（呼出し）ソースの場合は、各行に、有効なデバッグ情報でコンパイルされた各関数が表示されます。複数のプロファ

イリング情報が収集された場合は、別の関数を呼び出した関数の行を展開できます。特定の関数の子要素には、その親関数によって呼び出されたすべての関数と、関連する統計がリストされます。

- サンプリングおよびトレース（フラット）ソースの場合は、各行にアプリケーションの C 関数がそれぞれ表示されます。ランタイムライブラリからのコードまたはデバッグ情報なしの別コードからのセクションが、対応するアセンブラベルによって示されたもののみ、表示されます。トレースデータからの実行済 PC アドレスは、個別のサンプルとして扱われ、［プロファイリング］ウィンドウで対応する行に関連付けられます。各行には、これらのサンプル数が含まれます。

使用する C-SPY ドライバでどのビューがサポートされているかについては、281 ページの*プロファイラの使用に関する要件*を参照してください。

表示エリアには以下の情報が表示されます。

***関数（すべてのソース）***

プロファイルされた C 関数の名前。

サンプリングソースの場合は、ランタイムライブラリからのコードまたはデバッグ情報なしの別コードからのセクションが、対応するアセンブラベルによって示されたもののみ、表示されます。

**呼出し（ブレークポイントとトレース（呼出し））**

関数の呼出し回数。

**フラット時間（ブレークポイントとトレース（呼出し））**

関数内で消費された時間（サイクル数）。

**フラット時間 (%)（ブレークポイントとトレース（呼出し））**

合計時間の割合で表されたフラット時間。

**累積時間（ブレークポイントとトレース（呼出し））**

関数内で消費された時間（サイクル数）。

**累積時間 (%)（ブレークポイントとトレース（呼出し））**

合計時間の割合で表された累計時間。

**PC サンプル（トレース（フラット）とサンプリング）**

関数に関連する PC サンプルの数。

**PC サンプル (%)（トレース（フラット）とサンプリング）**

関数に関連付けられた PC サンプルの数（サンプル総数に対するパーセント値）。

**Power サンプリング（Power サンプリング）**

関数に関連する Power サンプリングの数。

**エネルギー (%)（Power サンプリング）**

関数に関連するすべての測定値の累計。全測定値に対するパーセントで表します。

**平均電流 [mA]（Power サンプリング）**

関数に関連するすべてのサンプルの平均測定値。

**最小電流 [mA]（Power サンプリング）**

関数に関連するすべてのサンプルの最小測定値。

**最大電流 [mA]（Power サンプリング）**

関数に関連するすべてのサンプルの最大測定値。

## コンテキストメニュー

以下のコンテキストメニューがあります。


このメニューの内容は、使用している C-SPY ドライバによって異なります。

以下のコマンドがあります。

**有効化**

プロファイラを有効にします。ウィンドウを閉じるときにも情報が記録されます。

**クリア**

すべてのプロファイリングデータをクリアします。

**フィルタリング**

プロファイルするコードの行を 以下から選択します。

**すべてをチェック** — すべての行をプロファイリングから除外します。

**すべてをチェック解除** — すべての行をプロファイリングに含めます。

**ロード** — 保存したファイルから除外された行をすべて読み込みます。

**保存** — 除外された行をすべてファイルに保存します。これは通常、エンジニアのグループが除外したセットを共有する場合に便利です。

これらのコマンドは、トレース（フラット）またはサンプリングのどちらかを使用する場合にのみ利用できます。

**ソース ***

プロファイリング情報に使用するソースを選択します。以下から選択します。

**サンプリング** — 命令プロファイリングの命令カウントは、各命令のサンプル数を示します。

**トレース（呼出し）**— 命令プロファイリングの命令カウントは、トレースデータ収集時の完了した数のみを示します。

**トレース（フラット）**— 命令プロファイリングの命令カウントは、トレースデータ収集時の完了した数のみを示します。

**Power サンプリング**

Power サンプリング情報の有効 / 無効を切り替えます。このコマンドは、I-jet および I-jet Trace インサーキットデバッグプローブ、JTAGjet、J-Link、J-Link Ultra デバッグプローブでサポートされています。

**ログファイルを保存**

すべてのプロファイリングデータをファイルに保存します。

* 使用する C-SPY ドライバによって、使用可能なソースが決まります。

# コードカバレッジ

- コードカバレッジの概要
- コードカバレッジについてのリファレンス情報

## コードカバレッジの概要

以下のトピックについて説明します :

- コードカバレッジを使用する理由
- コードカバレッジの概要
- コードカバレッジを使用するための要件と制限

### コードカバレッジを使用する理由

コードカバレッジ機能は、コードのあらゆる部分が実行されたことを確認するテスト手順を設計する場合に便利です。また、コードに到達不可能な部分が存在するかどうかを調べる場合にも使用できます。

### コードカバレッジの概要

［コードカバレッジ］ウィンドウでは、現在のコードカバレッジ解析のステータスが表示されます。それぞれのプログラム、モジュール、関数について、コードカバレッジがオンになってからアプリケーションが停止するまでに実行されたコードの割合がパーセントで解析に表示されます。また、実行されていないすべての文の一覧も表示されます。解析は無効にするまで続行されます。

### コードカバレッジを使用するための要件と制限

コードカバレッジは C-SPY シミュレータでサポートされているため、特定の要件や制限はありません。

ハードウェアデバッガシステムでコードカバレッジを使用するには、以下の要件および制限を考慮してください。

- SWO トレースを使用する場合 : コードカバレッジ情報はトレースサンプルのみに基づきます。つまり、100% のコードカバレッジに達するまでに関数を数回実行する必要があります。また、シングルステップの実行時はコードカバレッジ情報は収集されません。
- ETM トレースを使用する際、唯一の制限はトレースバッファのサイズです。トレースバッファを効率的に使用するには、トレース開始およびト

レース停止ブレークポイントを使用してトレースデータの収集を制限します。

# コードカバレッジについてのリファレンス情報

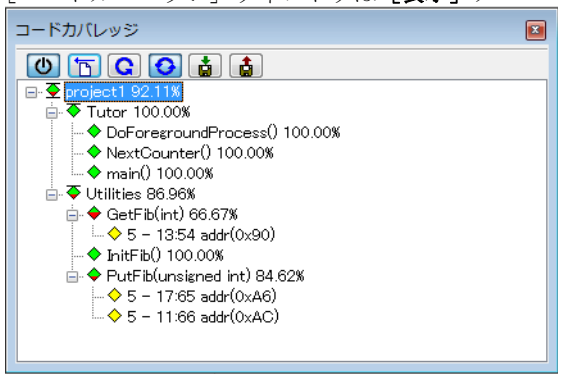
リファレンス情報：

- 294 ページの *［コードカバレッジ］ウィンドウ*。

78 ページの *ステップ実行*も参照してください。

## ［コードカバレッジ］ウィンドウ

［コードカバレッジ］ウィンドウは**［表示］**メニューから利用できます。




このウィンドウには、現在のコードカバレッジ解析のステータスが表示されます。それぞれのプログラム、モジュール、関数について、コードカバレッジがオンになってからアプリケーションが停止するまでに実行されたコードの割合がパーセントで解析に表示されます。また、実行されていないすべての文の一覧も表示されます。解析は無効にするまで続行されます。

タイトルバーにアスタリスク (*) が表示されている場合は、C-SPY が実行を継続していること、および［コードカバレッジ］ウィンドウに表示されている情報が最新ではないため、それを最新の情報に更新する必要があることを示します。最新の情報に更新するには、**［更新］**コマンドを使用します。

### コードカバレッジを使用するには、以下の手順に従います。

1 コードカバレッジ機能を使用するには、アプリケーションをビルドする際に以下のオプションを使用する必要があります。

| カテゴリ | 設定 |
|---|---|
| C/C++ コンパイラ | ［出力］>［デバッグ情報の生成］ |
| リンカ | ［出力］>［出力ファイルにデバッグ情報を含める］ |
| デバッガ | ［プラグイン］>［コードカバレッジ］ |

*表 14: コードカバレッジを有効にするためのプロジェクトオプション*

2 アプリケーションをビルドして C-SPY を起動した後、**［表示］>［コードカバレッジ］**を選択して［コードカバレッジ］ウィンドウを開きます。

3 **［有効化］**ボタンをクリックするか、コンテキストメニューから**［有効化］**を選択してコードカバレッジを有効にします。

4 実行を開始します。プログラムの終了に到達した、ブレークポイントがトリガされたなどの理由で実行が停止したときは、**［更新］**ボタンをクリックして、コードカバレッジ情報を確認します。

## 要件

以下のいずれかが必要です。

- C-SPY シミュレータ
- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY CMSIS-DAP ドライバ
- C-SPY ST-LINK ドライバ

## 表示エリア

コードカバレッジ情報には、ツリー構造でプログラム、モジュール、関数、文のレベルが表示されます。ウィンドウに表示されるのは、デバッグ情報付きでコンパイルされたソースコードだけです。したがって、起動コード、終了コード、ライブラリコードはウィンドウには表示されません。また、インライン化された関数内の文のカバレッジ情報は表示されません。インライン化された関数呼出しを含む文だけが実行済みとしてマークされます。プラス記号とマイナス記号をクリックすると、構造を展開したり折りたたんだりできます。

すべてのレベルの現在の状態は、以下のアイコンで示されます。

| | |
|---|---|
| 赤色のひし形 | モジュールや関数の 0% が実行されたことを示します。 |
| 緑色のひし形 | モジュールや関数の 100% が実行されたことを示します。 |
| 赤と緑のひし形 | モジュールや関数の一部が実行されたことを示します。 |
| 黄色のひし形 | 文が 1 つ実行されていないことを示します。 |

プログラム、モジュール、関数の各行の末尾に表示されるパーセント値は、それまでにカバーされた文の量、すなわち実行済みの文の数を文の総数で割った値を表します。

実行されていない文（黄色のひし形）の場合、表示される情報はソースウィンドウの列番号の範囲と行番号、続いてステップポイントのアドレスです。

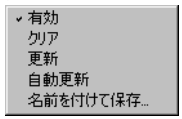
*<column_start>-<column_end>:row address.*

文は、その命令が 1 つでも実行されると、ステップポイントが実行されたとみなされます。文が実行されるとその文はウィンドウから削除され、それに対応してパーセント値が増加します。

[コードカバレッジ] ウィンドウで文か関数をダブルクリックすると、ソースウィンドウがアクティブウィンドウになり、ダブルクリックした文や関数がソースウィンドウでの現在の位置になります。プログラムレベルでモジュールをダブルクリックすると、ツリー構造を展開したり、折りたたんだりできます。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**有効化**

実行時のコードカバレッジの有効 / 無効を切り替えます。

**クリア**

すべてのコードカバレッジ情報を消去します。すべてのステップポイントが未実行として表示されます。

**更新**

コードカバレッジ情報を更新し、ウィンドウを再描画します。最後の更新以降に実行されたすべてのステップポイントは、ツリーから削除されます。

**自動更新**

コードカバレッジ情報の自動再ロードの有効 / 無効を切り替えます。有効にした場合は、ブレークポイント、ステップポイント、プログラム終了で C-SPY が停止したときに、コードカバレッジ情報が自動的に再ロードされます。

**名前を付けて保存**

現在のコードカバレッジ結果をテキストファイルに保存します。

# Power デバッグ

- Power デバッグの概要
- 電力消費のソースコードの最適化
- Power ドメインのデバッグ
- Power デバッグのリファレンス情報

## Power デバッグの概要

以下のトピックについて説明します:

- Power デバッグを使用する理由
- Power デバッグの概要
- Power デバッグの要件および制限

### POWER デバッグを使用する理由

バッテリー寿命の長さは、医療や家電、ホームオートメーションなど、ほとんどすべての市場区分において、多くの組込みシステムで非常に重要な要素です。これらのシステムの消費電力は、ハードウェアの設計だけでなく、ハードウェアの使用方法によっても異なります。システムソフトウェアは、使用方法を制御します。

Power デバッグが役に立つ例については、301 ページの *電力消費のソースコードの最適化* をご覧ください。

### POWER デバッグの概要

Power デバッグは、消費電力（より正確に言うと、CPU と周辺ユニットによって消費される電力）をサンプリングし、それぞれのサンプルをアプリケーションの命令シーケンスと関連付けて、それからプログラム実行におけるソースコードやさまざまなイベントと関連付けます。

従来からソフトウェア設計の主なゴールは、使用するメモリをなるべく少なくすることです。しかし、アプリケーションの消費電力をソースコードに関連付けることで、ソフトウェアが消費電力にどのように影響するのか理解して、電力の消費を最小限にする方法を考えることができます。

### 消費電力の測定

デバッグプローブは、デバイスへの供給電力に直列の小さい抵抗（シャント抵抗）に対する電圧の低下を測定します。電圧の低下は差動増幅器で測定され、続いて AD コンバータによってサンプリングされます。

### C-SPY を使用した Power デバッグ

C-SPY は Power デバッグを設定するインタフェースとなるほか、電力の値を参照するウィンドウのセットを提供します。

- ［Power 設定］ウィンドウでは、しきい値およびしきい値に達したときに実行されるアクションを指定できます。つまり、電力測定を有効または無効にしたり、アプリケーションの実行を停止して、予期しない電力値の原因を特定できます。
- ［Power ログ］ウィンドウには、記録された電力の値がすべて表示されます。このウィンドウは Power ログのピークを探すときに使用できます。値は実行されたコードに関連付けられているため、［Power ログ］ウィンドウの値をダブルクリックすれば、対応するコードを取得できます。精度はサンプルの周波数によって異なりますが、かなりの確率でピークの原因となったソースコードのシーケンスを見つけられます。
- ［タイムライン］ウィンドウの Power グラフには、時系列で電力の値が表示されます。［タイムライン］ウィンドウには、時系列で電力の値が表示されます。これは、ウィンドウに表示される他の情報と比較しながら消費電力を参照する便利な方法です。［タイムライン］ウィンドウは［Power ログ］ウィンドウと［ソースコード］ウィンドウ、［逆アセンブリ］ウィンドウに関連付けられており、タイムライン上の値に対応するソースコードがダブルクリックするだけで見つかります。
- ［関数プロファイラ］ウィンドウは、関数プロファイリングと Power ログを組み合わせて、関数ごとの電力消費、つまり電力プロファイリングを表示します。関数別の値のリストのほか、最大値と最小値とともに平均値も得られます。こうすることで、電力消費を最適化する際に集中すべきアプリケーションの領域を発見します。

### POWER デバッグの要件および制限

C-SPY でこの機能を使用して Power デバッグを実行するには、以下のいずれかが必要です。

- J-Link デバッグプローブおよび SWO を持つ Cortex-M デバイス。J-Link プローブの精度は非常に限られており、分解能も低い点に注意してください。
- I-jet または I-jet Trace インサーキットデバッグプローブ、もしくは J-Link Ultra デバッグプローブ。I-jet Trace で ETM トレースを取得時は Power デバッグを実行できない点に注意してください。

# 電力消費のソースコードの最適化

以下のトピックについて説明します :

- デバイスのステータスの待機
- ソフトウェア遅延
- DMA とポールされた I/O の比較
- 低電力モードの診断
- CPU 周波数
- 誤って放置されている周辺ユニットの検出
- イベント駆動型システムでの周辺ユニット
- 衝突するハードウェア設定の検出
- アナログ干渉

ここでは、Power デバッグが役に立つ例をいくつか紹介して、低消費電力のために最適化できるソースコード構造を特定しやすくするのが狙いです。

## デバイスのステータスの待機

不要な電力消費の原因となりうる一般的な構造は、たとえば周辺デバイスなどのステータス変更を待つためにポーリングループを使用することです。次の例にある構造は、ステータスの値が予想される状態になるまで中断なしに実行されます。

```
while (USBD_GetState() < USBD_STATE_CONFIGURED);
while ((BASE_PMC->PMC_SR & MC_MCKRDY) != PMC_MCKRDY);
```

電力消費を最小限にするには、デバイスのステータス変更のポーリングを記述し直して、ポールしていないときに CPU がスリープになれるように、割り込み、または可能であればタイマ割込みを使用することです。

## ソフトウェア遅延

ソフトウェア遅延は、たとえば次のように `for` または `while` ループとして実装できます。

```
i = 10000;  /* ソフトウェア遅延 */
do i--;
while (i != 0);
```

このようなソフトウェア遅延は、時間を消費する以外に目的のない命令の実行で CPU を稼動状態のままにします。時間の遅延は、ハードウェアタイマを使用して実装した方がずっと効率的です。タイマ割込みを設定した後は、CPU は割込みによって起こされるまで低電力モードになります。

### DMA とポールされた I/O の比較

これまで DMA は、転送速度を速くするために使用されてきました。MCU の場合、柔軟性や速度を高めたり、消費電力を抑える DMA のテクニックはたくさんあります。時には、DMA 転送中に CPU をスリープモードにすることもできます。Power デバッグを使用すると、従来の CPU 主体のポールソリューションに対して、これらの DMA テクニックが消費電力に与える影響を直接デバッガで実験して確認できます。

### 低電力モードの診断

多くの組込みアプリケーションでは、ほとんどの時間を何かが起こるまで待機して過ごします。シリアルポートでのデータ受信や、I/O ピンの状態の変更を観察したり、時間の遅延が期限切れとなるまで待機するなどです。プロセッサが待機中にまだフルスピードで実行中であれば、ほぼ何も処理されていないにも関わらずバッテリーが消費されます。そのため、多くのアプリケーションでは、コアは非常に短い時間だけアクティブになり、待機時間は低電力モードにすることで、バッテリー寿命を格段に延ばすことができます。

タスク指向の設計を行って RTOS を使用するのが、賢明なアプローチです。タスク指向の設計では、タスクは最も優先度を低く定義できます。実行するタスクが他にないときにだけ実行されます。この待機タスクは、電力管理を導入する完璧な場所です。実際には、待機タスクがアクティブになるたびに、コアが低電力モードに設定されます。多くのマイクロプロセッサおよびシリコンデバイスには、たくさんの低電力モードがあり、不要なときにコアの異なる部分をオフにできます。たとえば、発振器はオフにするか、低い周波数に切り替えることができます。また、個々の周辺ユニットやタイマ、CPU は停止可能です。異なる低電力モードでは、どの周辺ユニットがオンのままになっているかにより電力消費が異なります。Power デバッグツールは、異なる低電力モードで実験を行うときに非常に便利です。

C-SPY の関数プロファイラを使用して、異なる低電力モードが使用されたときにシステムを低電力モードにするタスクや関数の電力測定を比較できます。比較では平均値と合計消費電力のパーセント値がどちらも役に立ちます。

### CPU 周波数

CMOS MCU の電力消費は、理論的には次の公式により算出されます。

$P = f * U2 * k$

$f$ はクロック周波数、$U$ は供給電圧、$k$ は定数です。

Power デバッグを使用すると、クロック周波数の係数として電力消費を検証できます。50 MHz でほとんどスリープモードの時間がないシステムは、100 MHz で実行した場合に 50% の時間がスリープモードになることが予想されます。C-SPY で収集された電力データを使用して予想される動作を検証し、

クロック周波数にリニアでない依存性がある場合は、最も消費電力の少ない動作周波数を選択するようにしてください。

### 誤って放置されている周辺ユニットの検出

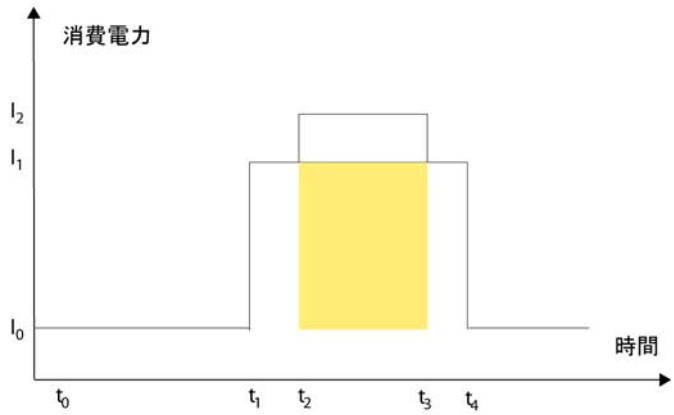
周辺ユニットは、使用されていない場合でも大量の電力を消費することがあります。低電力を考えて設計する場合、使用していないときは周辺ユニットを無効にして、放置しないことが重要です。ただし、さまざまな理由で周辺ユニットの電源供給をオンのままにすることがあります。慎重で正しい設計上の決定のこともあれば、不十分な設計か単なるミスの可能性もあります。前者の場合でなければ、予想を上回る電力がアプリケーションで消費されることになります。このことは、［タイムライン］ウィンドウの Power グラフで簡単に分かります。［タイムライン］ウィンドウで電力消費が予想外に高い部分をダブルクリックすると、対応するソースコードと逆アセンブリコードに移動します。ほとんどの場合、アクティブでないときに周辺ユニットを無効にするだけで十分です。たとえば、クロックをオフにすれば、たいていの場合は電力の消費が完全に停止します。

ただし、クロックのゲートだけでは不十分な場合もいくつかあります。コンバータやコンパレータなどアナログの周辺ユニットは、クロックがオフの場合でもかなりの電力を消費します。［タイムライン］ウィンドウでは、クロックをオフにするだけでは不十分で、周辺ユニットを完全にオフにする必要があることが示されます。

### イベント駆動型システムでの周辺ユニット

実行中にあるタスクがアナログコンパレータを使用し、そのタスクがより優先度の高いタスクによって停止される場合のシステムを考えてください。理想的には、タスクが停止されたときにコンパレータがオフになり、タスクが再開したときに再びオンになるべきです。こうすれば、優先度の高いタスクの実行中に、消費される電力を最小限に抑えられます。

これはイベント駆動型を想定したシステムを電力消費の回路図で、$t_0$ 時点でシステムは非アクティブモードにあり、電流は $_0$ です。




$t_1$ の時点ではシステムがアクティブになり、電流は少なくとも 1 つの周辺デバイスがオンになっているアクティブ時のシステムの消費電力である $I_1$ に上がって、電流は $I_1$ に上がります。$t_2$ では、優先度の高い割込みによって実行が停止します。すでにアクティブだった周辺デバイスは、優先度の高いタスクで使用されないにも関わらず、オフになっていません。その代わりに、新しいタスクによってさらに周辺デバイスがアクティブになり、制御が優先度の低いタスクに戻る $t_2$ から $t_3$ の間に電流が $I_2$ に上がります。

システムの機能は申し分なく、速度とコードサイズの面では最適化が可能です。しかし、Power ドメインでもさらなる最適化が行えます。重なっているエリアは、$t_2$ と $t_3$ の間で使用されない周辺デバイスをオフにしたり、2 つのタスクの優先度が変わった場合に、節約できたエネルギーを表します。

［タイムライン］ウィンドウを使用すると、綿密な調査を行って、使用されていない周辺デバイスがアクティブになって、不要に長い時間にわたり電力を消費していたことを明らかにできます。当然ながら、例のような状況で、追加のクロックサイクルを使用して周辺デバイスをオンやオフにする価値があるかどうかを考えなければなりません。

## 衝突するハードウェア設定の検出

フローティング入力を回避するため、使用されていない MCU I/O ピンを接地するのが一般的な設計上の慣習です。誤ってソースコードで接地された I/O ピンのいずれかを論理的 1 出力として設定した場合、そのピンで高電流が失われる可能性があります。この予想外の高電流は、［タイムライン］ウィンドウの Power グラフから電流の値を読み取れば、簡単に観測できます。対応す

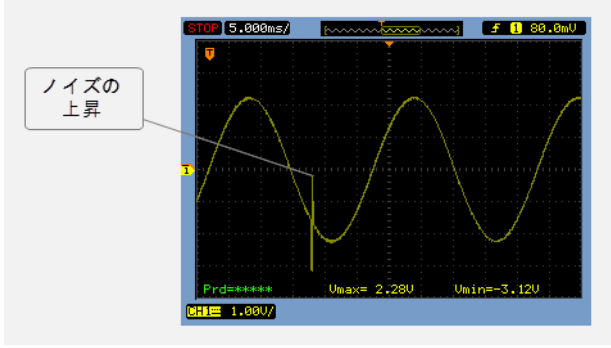
る間違った初期化コードも、アプリケーション起動時の Power グラフを見れば発見できます。

ある I/O ピンが入力として設計されて外部の回路によって駆動するにも関わらず、コードで誤って入力ピンを出力として設定した場合、同じような状況が発生します。

## アナログ干渉

同じボード上でアナログとデジタルの回路を混在させる場合、ボードのレイアウトとルーティングがアナログのノイズレベルに影響することがあります。低レベルのアナログ信号の正確なサンプリングを確実に行うため、ノイズレベルを低く保つことが重要です。効率的に混在した信号設計を実現するには、ハードウェアを慎重に考慮する必要があります。また、ソフトウェア設計がアナログ測定の質に影響することもあります。

アナログ信号のサンプリングと同時に I/O アクティビティを大量に実行すると、多くのデジタル線で状態が同時に切り替わり、AD コンバータにさらなるノイズが追加されることがあります。




Power デバッグは、アナログ部品へのデジタルおよび電力供給線からの干渉を調べる上で役に立ちます。AD 変換の付近での電力の上昇はノイズの発生源かもしれず、調査する必要があります。［タイムライン］ウィンドウにあるすべてのデータは、実行されたコードに関連付けられています。疑問を感じる電力の値をダブルクリックするだけで、対応する C のソースコードが表示されます。

# Power ドメインのデバッグ

以下のタスクについて説明します:

- 電力消費プロファイルの表示と結果の解析
- アプリケーション実行中の予想外の電力消費の検出
- グラフの解像度の変更

関連項目:

- 244 ページの *［タイムライン］ウィンドウ*
- 285 ページの *プロファイリング情報の間隔を選択する*

## 電力消費プロファイルの表示と結果の解析

**電源プロファイルを表示するには:**

1. ［**C-SPY ドライバ**］>［**SWO 設定**］を選択して、［**SWO 設定**］ダイアログボックスを開きます。［CPU クロック］オプションが、アプリケーションの CPU クロックの値と同じに設定されているか確認してください。SWO クロックを設定して、デバッグプローブへ正しいデータ転送を行うには、こうする必要があります。

   C-SPY シミュレータを使用する場合は、この手順は無視してかまいません。

   このステップには Cortex-M3/M4 デバイスが必要です。

2. デバッガを起動します。
3. **C-SPY ドライバ >Power ログの設定**を選択します。**[ID]** 列で、Power ロギングを有効化する項目を必ず選択してください。
4. ［**C-SPY ドライバ**］>［**タイムライン**］を選択して、［タイムライン］ウィンドウを表示します。
5. グラフエリアで右クリックして、コンテキストメニューから［**有効化**］を選択し、表示する Power グラフを有効にします。
6. ［**C-SPY ドライバ**］>［**Power ログ**］を選択して、［Power ログ］ウィンドウを開きます。
7. オプションで、電力の値を特定の割込みや変数に関連付ける場合は、割込みまたはデータロググラフのエリアをそれぞれ右クリックして、コンテキストメニューから［**有効化**］を選択します。

   変数の場合は、［タイムライン］ウィンドウでグラフィック表示を行う各変数にデータログブレークポイントを設定する必要もあります。162 ページの *［データログ］ブレークポイントダイアログボックス（C-SPY ハードウェアドライバ）* を参照してください。

このステップには Cortex-M3/M4 デバイスが必要です。

8 オプションで、アプリケーションの実行を開始する前に、Power グラフの Y 軸の表示範囲を設定できます。255 ページの*［表示範囲］ダイアログボックス*を参照してください。

9 ツールバーで［**実行**］をクリックして、アプリケーションの実行を開始します。［Power ログ］ウィンドウに、すべての電力の値が表示されます。［タイムライン］ウィンドウでは電力の値がグラフィック表示され、これらのグラフを有効にした場合はデータおよび割込みのログ（Cortex-M3/M4 を使用する場合）も表示されます。グラフでのナビゲートの方法について詳しくは、244 ページの*［タイムライン］ウィンドウ*を参照してください。

10 電力消費を解析するには、以下の手順に従います（Cortex-M3/M4 デバイスが必要です）：

- 関心のある電力の値をダブルクリックすると、対応するソースコードがエディタウィンドウと［逆アセンブリ］ウィンドウで強調表示されます。対応するログは、［Power ログ］ウィンドウで強調表示されます。これが役に立つ例については、301 ページの*電力消費のソースコードの最適化*をご覧ください。
- 使用していないときに無効にする周辺ユニットを特定することができます。これは、Power グラフと［タイムライン］ウィンドウの他のグラフを組み合わせて解析すれば検出できます。303 ページの*誤って放置されている周辺ユニットの検出*も参照してください。
- 特定の割込みについては、割込みの終了後に電力消費が予想外に変化したかどうかを確認できます。たとえば、割込みによって電力消費の大きいユニットが有効になり、終了する前にオフにしない場合などです。
- 関数プロファイリングについては、285 ページの*プロファイリング情報の間隔を選択する*を参照してください。

### アプリケーション実行中の予想外の電力消費の検出

**予想外の電力消費を検出するには：**

1 ［**C-SPY ドライバ**］>［**SWO 設定**］を選択して、［**SWO 設定**］ダイアログボックスを開きます。以下の設定が使用されていることを確認します。

- ［**CPU クロック**］が、アプリケーションの CPU クロックの値と同じに設定されているか確認してください。SWO クロックを設定して、デバッグプローブへ正しいデータ転送を行うには、こうする必要があります。

このステップには Cortex-M3/M4 デバイスが必要です。

2 ［**C-SPY ドライバ**］>［**Power ログの設定**］を選択して、［Power 設定］ウィンドウを開きます。

3 ［Power 設定］ウィンドウで、しきい値と適切なアクションを指定します。たとえば、**［すべてをログしてしきい値以上で CPU を停止］**などです。

4 **［C-SPY ドライバ］>［Power ログ］**を選択して、［Power ログ］ウィンドウを開きます。電力の値を継続的にファイルに保存する場合は、コンテキストメニューから**［ライブログファイルの選択］**を選択します。この場合、**［指定先へのライブログを有効にする］**も選択する必要があります。

5 実行を開始します。

消費電力がしきい値を超えたとき、実行が停止して指定したアクションが行われます。

記録した電力の値をファイルに保存した場合、外部ツールでそのファイルを開いて、さらに解析することが可能です。

### グラフの解像度の変更

**［タイムライン］ウィンドウの Power グラフの解像度を変更するには、以下の手順に従います。**

1 ［タイムライン］ウィンドウで［Power グラフ］を選択し、右クリックして**［設定ウィンドウを開く］**を選択して［Power ログ設定］ウィンドウを開きます。

2 ［Power ログ設定］ウィンドウのコンテキストメニューから、適切な測定の単位を選択します。

3 ［タイムライン］ウィンドウで［Power グラフ］を選択し、コンテキストメニューから**［表示範囲］**を選択します。

4 **［表示範囲］**ダイアログボックスで、**［カスタム］**を選択し、**［最小値］**と**［最大値］**のテキストボックスで値の範囲を指定します。**［OK］**をクリックします。

5 設定に従ってグラフが自動的に更新されます。

## Power デバッグのリファレンス情報

リファレンス情報 :

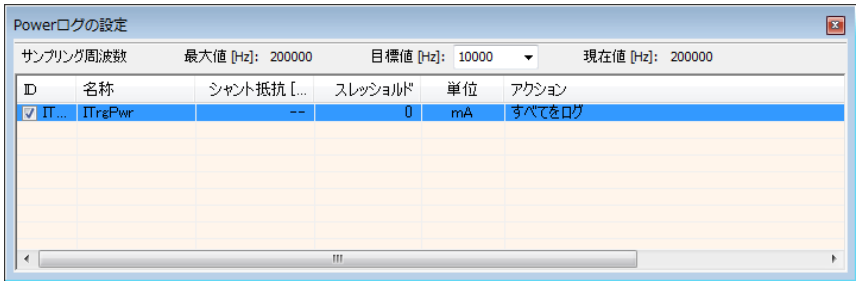
- 309 ページの *［Power ログ設定］ウィンドウ*
- 311 ページの *［Power ログ］ウィンドウ*
- 315 ページの *［タイムライン］ウィンドウの Power グラフ*

関連項目：

- 237 ページの *［トレース］ウィンドウ*
- 244 ページの *［タイムライン］ウィンドウ*
- 255 ページの *［表示範囲］ダイアログボックス*
- 287 ページの *［関数プロファイラ］ウィンドウ*

## ［Power ログ設定］ウィンドウ

［Power ログ設定］ウィンドウは、デバッグセッション中に C-SPY ドライバのメニューから使用できます。




このウィンドウを使用して、電力の測定を設定します。

**注 :** Power ログを有効にするには、［Power ログ］ウィンドウのコンテキストメニューまたは［タイムライン］ウィンドウの Power グラフのコンテキストメニューから **［有効化］**を選択します。

### 要件

以下のいずれかが必要です。

- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ

### 表示エリア

このエリアには以下の列が含まれます。

**ID**

プローブの測定チャンネルを識別する一意の文字列。このチェックボックスを選択して、チャンネルをアクティブにします。このチェックボックスの選択を解除すると、そのチャンネルのログは生成されません。

**名前**

ユーザ定義名を指定します。

**シャント抵抗 [Ω]**

I-scope を除くすべてのデバッグプローブについて、この列には常に --（ダッシュ 2 つ）が含まれます。

I-scope の場合、シャント抵抗を指定してください。

**しきい値**

選択した単位でしきい値を指定します。しきい値に達したときに、指定したアクションが実行されます。

**単位**

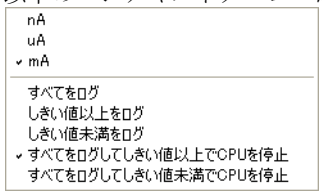
電力（電流）の表示単位を選択します。以下から選択します。**nA**、**uA**、**mA**。

**アクション**

測定チャンネルについて選択した処理内容が表示されます。以下から選択します。**すべてをログ**、**しきい値以上をログ**、**しきい値未満をログ**、**すべてをログしてしきい値以上で CPU を停止**、**すべてをログしてしきい値未満で CPU を停止**。

## コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**nA、uA、mA**

電力（電流）の表示単位を選択します。これらは、電力を測定するチャンネルで使用できます。

**すべてをログ**

すべての値を記録します。

**しきい値以上をログ**

しきい値を超えたすべての値を記録します。

**しきい値未満をログ**

しきい値未満の値をすべて記録します。

**すべてをログしてしきい値以上で CPU を停止**

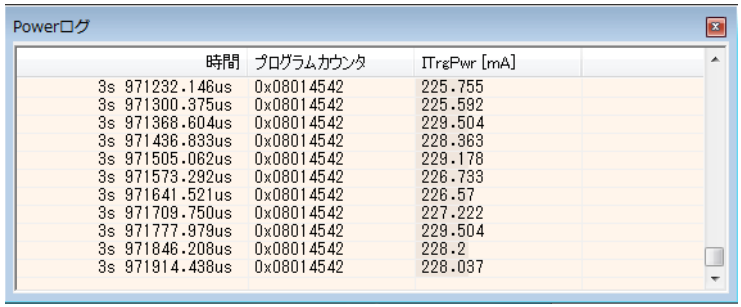
すべての値を記録します。記録された値がしきい値を超えると、実行が停止します。

**すべてをログしてしきい値未満で CPU を停止**

すべての値を記録します。記録された値がしきい値未満の場合、実行が停止します。

## ［Power ログ］ウィンドウ

［Power ログ］ウィンドウは、デバッグセッション中に C-SPY ドライバのメニューから使用できます。




このウィンドウには、収集された電力の値が表示されます。

時間 / サイクルとプログラムカウンタのみ灰色で表示された行は、［Power ログの設定］ウィンドウで実際のデータ収集時にアクティブで現在は無効になっているチャンネルについて記録された電力の値を示します。

**注 :** 記録される電力の値の数が制限されます。この制限を超過すると、バッファの最初のエントリが消去されます。

### 要件

以下のいずれかが必要です。

- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ

### 表示エリア

このエリアには以下の列が含まれます。

**時間**

［**SWO 設定**］ダイアログボックスで指定したクロック周波数に基づく、アプリケーションのリセットからイベントまでの時間。

ターゲットシステムが正確な時間を収集できなかった場合は、おおよその時刻が斜体で表示されます。

この列は、コンテキストメニューから［**時間表示**］を選択した場合に有効になります。

**サイクル**

アプリケーションのリセットからイベントまでのサイクル数。この情報は、リセットでクリアされます。

ターゲットシステムが正確な時間を収集できなかった場合は、おおよそのサイクルが斜体で表示されます。

この列は、コンテキストメニューから［**サイクル表示**］を選択した場合に有効になります。

**プログラムカウンタ**

以下のいずれかが表示されます。

PC の内容であるアドレス。つまり、電力の値が収集されたポイントに近い命令のアドレスです。

---、ターゲットシステムがデバッガに情報を提供できなかったことを示します。

赤色で Overflow と表示されている場合、通信チャンネルがすべてのデータをターゲットシステムから送信できなかったことを示します。

待機、電力の値は待機モード中に記録されます。

***名称*［*単位*］**

［Power 設定］ウィンドウで指定した単位で表される電力の測定値。

### コンテキストメニュー

以下のコンテキストメニューがあります。


以下のコマンドがあります。

**有効化**

ロギングシステムを有効にします。つまり、電力の値は IDE で内部的に保存されます。値は［タイムライン］ウィンドウの［Power ログ］ウィンドウに表示されます（有効になっている場合）。ロギングシステムでは、ウィンドウを閉じるときにも情報が記録されます。

**クリア**

IDE に内部的に保存された電力の値を消去します。デバッガをリセットしたり、**［SWO の設定］**ダイアログボックスで実行周波数を変更した場合も、値は消去されます。

**ログファイルを保存**

記録された電力の値の保存先ファイルを選択する、標準のファイル選択用ダイアログボックスを表示します。このコマンドにより、内部ログバッファの最新の内容が保存されます。

**ライブログファイルの選択**

記録された電力の値の保存先ファイルを選択する、標準のファイル選択用ダイアログボックスを表示します。電力の値は実行時に連続してこのファイルに保存されます。ライブログファイルの内容は自動的にクリアされることはなく、記録された値は単にファイルの末尾に追加されます。

**指定先へのライブログを有効にする**

ライブロギングのオンとオフを切り替えます。ログは指定したファイルに保存されます。

***ログファイルのクリア***

ライブログファイルの内容を消去します。

**時間表示**

［Power ログ］ウィンドウに［**時間**］列を表示します。この選択は、ログファイルにも反映されます。

**サイクル表示**

［Power ログ］ウィンドウに［**サイクル**］列を表示します。この選択は、ログファイルにも反映されます。

**［設定］ウィンドウを開く**

［Power ログの設定］ウィンドウを開きます。

### ログファイルのフォーマット

ログファイルは、タブで区切られたフォーマットです。ログファイルのエントリは、タブおよびラインフィードで区切ります。記録された電力の値は、以下の列に表示されます。

**時間 / サイクル**

アプリケーションのリセットから電力の値が記録されるまでの時間。

**概算値**

この列の `x` は、電力の値が時間 / サイクルの概算値であることを示します。

**PC**

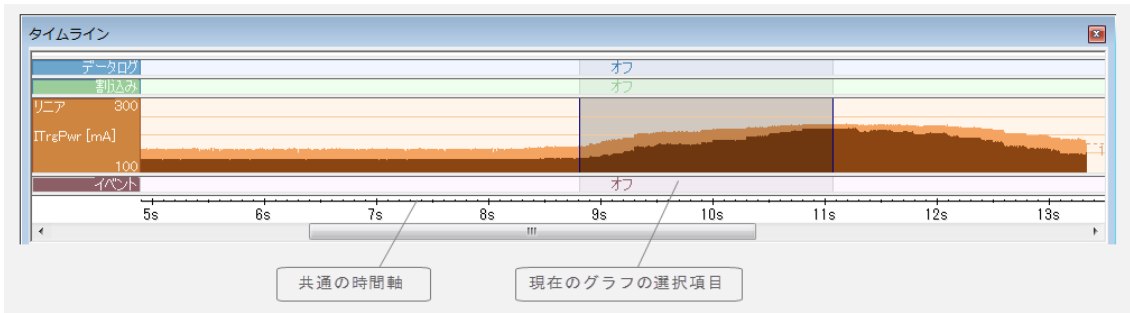
電力の値が記録されたポイントに近いプログラムカウンタの値。

***名称*［*単位*］**

［Power ログ］ウィンドウからの対応する値。*名称*と*単位*は、［Power ログの設定］ウィンドウの設定に従います。

# ［タイムライン］ウィンドウの Power グラフ

［タイムライン］ウィンドウの Power グラフは、デバッグセッション中に［C-SPY ドライバ］メニューから使用できます。




Power グラフには、デバッグプローブまたは共通の時間軸に比例する関連のハードウェアによって生成された電力測定サンプルがグラフで表示されます。

［タイムライン］ウィンドウやグラフの表示方法、サポートされている他のグラフについて詳しくは、244 ページの*［タイムライン］ウィンドウ*を参照してください。

300 ページの *Power デバッグの要件および制限*を参照してください。

## 要件

以下のいずれかが必要です。

- C-SPY I-jet/JTAGjet ドライバ
- C-SPY J-Link/J-Trace ドライバ

## 表示エリア

場所：

- グラフ左端のラベルエリアには、測定チャンネル名が表示されます。
- グラフ自体には、デバッグプローブや関連のハードウェアによって生成された電力測定サンプルが表示されます。
- グラフは連続したログ間の細い線として、または各ログを示す長方形（オプションでカラー）、あるいは列として表示することができます。
- グラフの解像度は変更することができます。
- 赤色の垂直線は、オーバフローを示します。これは、通信チャンネルがすべての割込みログをターゲットシステムから送信できなかったことを示します。

ウィンドウの下部分に、秒を時間単位として使用する共通の時間軸があります。

# C-RUN ランタイムエラー解析

- ランタイムエラー解析の概要
- C-RUN の使用
- さまざまなランタイムエラーの検出
- ランタイムエラー解析のリファレンス情報
- C-RUN のためのコンパイラおよびリンカのリファレンス
- C-RUN の cspybat オプション

この章に記載されている機能は、IAR Embedded Workbench アドオン製品の C-RUN を必要とします。

## ランタイムエラー解析の概要

以下のトピックを解説します。

- ランタイムエラー解析
- C-RUN を使用したランタイムエラー解析
- ライブラリにより提供されるチェック済みヒープ
- IAR Embedded Workbench IDE での C-RUN の使用
- 非対話型モードでの C-RUN の使用 non-interactive mode
- ランタイムエラー解析の要件

### ランタイムエラー解析

ランタイムエラー解析は、アプリケーションの実行中に不正なコードの構造を検出する方法です。これは、アプリケーション内にコードを実装するか、C/C++ ライブラリの機能をランタイムエラー解析をサポートする専用のライブラリに置き換えることにより実行します。

ランタイムエラー解析では、アプリケーションのタイプと実行する環境に応じて、解析の実行にさまざまな方法を使用出来ます。

解析を実行するコードを実装することで、コードサイズが大きくなり、処理速度が低下します。また、解析をサポートするライブラリ関数の派生型も、チェックをサポートしない関数に比べて、一般的にコードサイズが大きくなり速度が低下します。

## C-RUN を使用したランタイムエラー解析

C-RUN は、以下の 3 種類のランタイムエラー解析をサポートしています。

- *算術解析*には、整数のオーバフローとアンダーフロー、誤ったシフト、ゼロによる除算、値が変化する変換、switch 文における未処理のケースの解析が含まれます。通常は算術解析のオーバヘッドは特に高くはありません。複雑な問題もなく、モジュールごとに有効または無効にすることができます。
- *境界チェック*は、ポインタ経由のアクセスがポイント先のオブジェクトの境界内にあるかどうかをチェックします。境界チェックにはポインタの境界を追跡するためのコードの実装が伴い、コードサイズと速度の両面でコストが比較的高くなります。間接的にアクセスされるポインタのグローバル境界テーブルも必要です。モジュールや関数ごとに追跡やチェックだけを無効にすることができますが、ポインタの境界がすべてのコードで追跡されない設定の場合は通常、何らかのコード調整が必要となります。
- *チェック済みのヒープを使用したヒープチェック*では、ヒープメモリの使用においてエラーがないかチェックします。ヒープチェックでは、ヒープメモリへの間違ったライトアクセス、ダブル・フリー、一致しない割当てと割当て解除、さらに明示的な呼出しをすることでリークのあるヒープブロックを検出できます。チェック済みヒープを使用すると、各ヒープブロックのメモリサイズが大きくなり、ヒープサイズを増やさなければならないことがあります。また、通常のヒープに比べてヒープの処理に多くの時間がかかる可能性があります。さらに、ヒープ関数が呼び出されたときにだけチェックが行われるため、すべてのヒープライトエラーが検出されるわけではありません。

C-RUN が実行可能なすべてのチェックは、C と C++ のソースコードのどちらについても使用できます。

数種類の C-RUN チェックを同時に有効化することができます。有効化するチェックの種類が増えるごとに、実行時間が長く、コードサイズが大きくなります（増え方が非常に少ないこともあります）。

コンパイラが複数のチェックを 1 つにまとめたり、ループからチェックを移動することもあります。この場合、実際のアクセスよりずっと前に問題が検出される可能性があります。このような場合、C-RUN のメッセージに問題のソース位置が現在の位置とは別に表示されます。

C-RUN のランタイムチェックを実行する前に、問題を見つけるためにコンパイラのあらゆる機能を使用してください。

- K&R スタイルの関数の宣言は使用せず、代わりにプロトタイプスタイルを使用してください。ARM 用 IAR C/C++ 開発ガイドの `--require_prototypes` を参照してください。
- ランタイムチェックを実行する前に、あらゆるコンパイラのワーニングに注意してください。ほとんどの場合、一度ワーニングが表示された問題をチェックするコードは出力されません。次に例を示します。

  ```
  unsigned char ch = 1000; /* ワーニング : 整数の切捨て */
  ```

  整数の変換チェックが有効の場合であっても、出力されたコードにはこの場合のチェックはまったく含まれません。単に値 232（1000 & 255）を `ch` に割り当てるだけのコードです。

C-RUN は、ARM のセミホスティングインタフェースに依存する点に注意してください（ライブラリ関数 `__iar_ReportCheckFailed` は、セミホスティングインタフェースを通じて C-SPY とやりとりします）。別の低レベルの I/O インタフェースは、非対話型モードでのみ使用可能です。320 ページの*非対話型モードでの C-RUN の使用 non-interactive mode* を参照してください。

エラーの検出方法については、324 ページの*さまざまなランタイムエラーの検出*を参照してください。

## ライブラリにより提供されるチェック済みヒープ

*チェック済みヒープ*を提供するライブラリがあり、これを使用してヒープの使用をチェックすることができます。チェック済みヒープは、ヒープブロックのユーザ部分の前後にガードバイトを挿入するほか、各ブロックに追加情報（連続する割当て番号を含む）を格納して、レポートに役立てます。

それぞれのヒープ処理では通常、ガードバイトあるいは新しく割り当てられたヒープメモリの内容の変更に対して、関係する各ヒープブロックがチェックされます。特定の場合（特定の呼出しによるトリガ、または設定したヒープ処理の回数を超過した場合）、ヒープの整合性チェックが実行され、問題がないかどうかヒープ全体がチェックされます。

チェック済みヒープは間違ったリードアクセスを検出することはできない点を十分に理解してください。解放されたヒープブロックからのリードや、割り当てられたヒープブロックの境界の外でのリードなどです。これらのアクセスやチェック済みヒープで見過ごされがちな数多くの間違ったライトアクセスは境界チェックで検出することができます。境界チェックではガードバイトやチェック済みバイトにライトすることがないためです。または、チェック済みヒープはヒープ処理の使用時にのみチェックを行い、実際のアクセス時点にはチェックを実行しません。

## IAR EMBEDDED WORKBENCH IDE での C-RUN の使用

C-RUN は IAR Embedded Workbench IDE に完全に統合されており、以下の機能を提供します。

- 検出された各エラーについて詳しいエラー情報がコールスタック情報とともに提供され、エラーに関するコードとの相関関係とフィードバックがエディタウィンドウに表示されます。
- プロジェクトレベル、ファイルレベルまたは指定のコード位置における個々のランタイムエラーに対して、実行の停止、ログ、無視するようにエラールール管理できます。フィルタ設定をロード / セーブすることができます。
- 各メッセージについてエディタウィンドウでブックマークを設定できるため、メッセージ間の移動が容易（F4 を使用）。

IDE では、C-RUN に以下のウィンドウが用意されています。

- ［C-RUN メッセージ］ウィンドウは C-RUN が生成するすべてのメッセージが一覧表示されます。各メッセージにはメッセージタイプ（実行されたチェックを反映）、問題を説明する文、コールスタックが含まれます。対応するソースコード文は、エディタウィンドウで強調表示されます。346 ページの *［C-RUN メッセージ］ウィンドウ*を参照してください。
- ［C-RUN メッセージルール］ウィンドウには、すべてのルールが一覧表示されます。348 ページの *［C-RUN メッセージルール］ウィンドウ*を参照してください。ルールによって、［C-RUN メッセージ］ウィンドウに表示されるメッセージが決まります。

## 非対話型モードでの C-RUN の使用 NON-INTERACTIVE MODE

C-RUN でチェック済みのプログラムは、cspybat を用いて C-SPY をバッチモードで使用して実行できます。cspybat は、Workbench IDE で設定されたルールや他の設定を使用することができます。cspybat の C-RUN メッセージは、デフォルトでホスト stdout に報告されますが、ファイルにリダイレクトすることができます。

アプリケーションと C-RUN メッセージのホスト間で独自の通信チャンネルを使用する場合は、関数 __iar_ReportCheckFailed（通信にセミホスティングインタフェースを使用）を独自のバージョンに置換すれば、どれでも希望する通信インタフェースを使用できます。ソースファイル ReportCheckFailedStdout.c (arm¥src¥lib¥crun) には、アプリケーションの stdout に報告する派生型があります。セミホスティングのものではなく、独自のレポート関数を使用するには、リンカオプション --redirect __iar_ReportCheckFailed=__iar_ReportCheckFailedStdout を使用します。

**注 :** レポート関数のモジュールがプロジェクトに挿入されている場合、そのモジュールは C-RUN のソースコードオプションでコンパイルしないでください。

`__iar_ReportCheckFailedStdout` からの出力は、raw データのみが含まれるため、ユーザが読取り可能な形式ではありません。`cspybat` をオフラインモードで使用（オプション `--rtc_filter` と `--rtc_filter_file` を利用）すれば、raw テキストを通常の C-RUN メッセージに非常に近いものに変換することができます。

オプション `--rtc_enable` を使用して、`cspybat` で C-RUN を有効化します。C-RUN 用の `cspybat` オプションはすべて、`--rtc_*` で始まります。これらのオプションについては、357 ページの *C-RUN の cspybat オプション*を参照してください。

### ランタイムエラー解析の要件

ランタイムエラー解析を実行するには、IAR Embedded Workbench のアドオン製品である C-RUN が必要です。

## C-RUN の使用

以下のタスクについて解説します。

- C-RUN ランタイムエラー解析を使用するにあたって
- メッセージのルール作成

### C-RUN ランタイムエラー解析を使用するにあたって

一般的に C-RUN を使用するには、以下の手順に従いいます。

- 必要な C-RUN のチェックを決定し、C-RUN オプションでそれらを指定します。
- IAR Embedded Workbench IDE でアプリケーションを実行して、個々の C-RUN メッセージを対話形式でチェックしてください。各メッセージについて、重要な問題かどうかを判断します。そうでない場合は、今後その特定のメッセージや同じようなメッセージを無視するルールを適用することができます。メッセージが重要な問題である場合は、特定の状況に応じて、問題を修正して再度実行したり、他に問題があるかどうかをチェックすることができます。
- 終わったら C-SPY を閉じます。C-RUN のウィンドウは開いたままですので、見つかった問題に関して確認するよいタイミングとなります。ルールの設定を確認して編集し、後で実行するときのために保存します。
- すべての問題が解決するまで、このプロセスを繰り返します。

より詳しくランタイムエラー解析を実行してランタイムエラーを検出するには、一般的なプロセスの次の例に従います。

1 ランタイムチェックのプロジェクトオプションを設定するには、**［プロジェクト］>［オプション］>［ランタイムチェック］**を選択して、たとえば**［境界チェック］**など、実行するランタイムチェックを選びます。

ランタイムチェックはプロジェクトのレベルで有効化してから、使用する各タイプのチェックを有効にする必要があります。**［チェック済みヒープの使用］**や**［境界チェックを有効化］**などチェックオプションによっては、プロジェクトレベルで有効化する必要があります。その他はプロジェクトやファイルのレベルで有効化できます。

2 アプリケーションをビルドします。低い最適化レベルの方が、より適切な情報が得られる点に注意してください。

3 デバッグセッションを開始します。

4 アプリケーションプログラムの実行を開始します。

5 C-RUN で潜在的なエラーが検出されると、プログラムが実行を停止し、エディタウィンドウで対応するソースコードが強調表示されます。

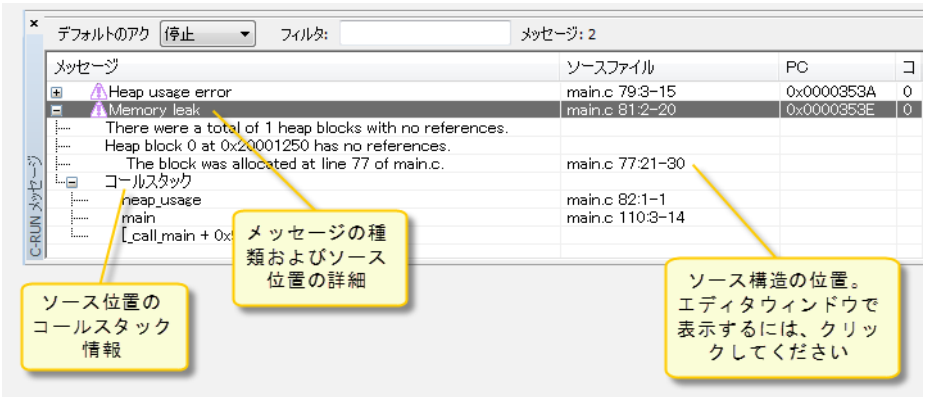
```
char *p = malloc(10);

free(p + 200);

__iar_check_leaks(); // Leakage

return 0;
```

まだ開いていない場合は［C-RUN メッセージ］ウィンドウが表示され、ソースコードの構成やチェックの種類、ソース位置のコールスタック情報が示されます。



問題の検出が実際のアクセス時点で発生するとは限りません。チェックがループ外に移動されたり、異なるアクセスに対する複数のチェックが統合されている可能性もあります。この場合は問題のソース（問題となるアクセスのソース）が現在の文にはない場合や、問題が複数ある可能性があります。

6 ソースコードの構成によっては、潜在的なエラーが検出された後にもプログラムの実行を継続できることがあります。一部の種類のエラーによって、たとえばデータやコードのオーバライドが原因で、実行時に予期しない動作が発生することがあります。

7 必要があれば、［C-RUN メッセージルール］ウィンドウを使用して、特定のチェックやソースコードの位置、特定のチェックとソースファイル、または特定のチェックのみに基づいて、特定のメッセージを除外するルールを指定することができます。また、特定のチェックで実行を停止せず、記録のみを停止するように指定することも可能です。323 ページの*メッセージのルール作成*を参照してください。

実行するさまざまなランタイムチェックについて、この手順を繰り返します。

### メッセージのルール作成

使用するソースコードによっては、［C-RUN メッセージ］ウィンドウのメッセージ数が非常に多くなることがあります。集中しやすくするために、表示するメッセージを制御するルールを作成できます。

**ルールを作成するには、次の手順に従います。**

1 ［C-RUN メッセージ］ウィンドウでフィルタルールを作成するメッセージを選択します。

2 右クリックして、表示されるコンテキストメニューからルールを 1 つ選択します。

ルールが［C-RUN ルール］ウィンドウに表示されます。

3 すべてのルールの概要を見るには、**［表示］> ［C-RUN ルール］**を選択します。

チェックに失敗したときは、ルールによってメッセージを報告する方法が決まります。上から下に向かってルールがスキャンされ、最初に一致したルールの処理が実行されます。

**注 :** フィルタ設定を保存して、後で新しいデバッグセッションにロードすることができます。

# さまざまなランタイムエラーの検出

以下のタスクについて解説します。

- 暗黙的または明示的な整数変換の検出
- 符号付きまたは符号なしのオーバフローの検出
- ゼロによる除算の検出
- シフトする際のビット損失または未定義の動作の検出
- switch 文における未処理のケースの検出
- 配列およびその他のオブジェクトの境界の外にあるアクセスの検出
- ヒープ使用エラーの検出
- ヒープメモリのリークの検出
- ヒープの整合性違反の検出

## 暗黙的または明示的な整数変換の検出

**説明**

整数変換（暗黙的または明示的）、あるいはビットフィールドへのライトアクセスによって値が変更されないことを確認します。

**チェックを実行する理由**

C ではより大きな型から小さい整数型への変換が可能なため、一部の変換で値の上位のビットが意図せずに削除されることがあります。このチェックは暗黙的な整数変換を限定することができ、データの損失が明示的な変換による意図的とはっきりさせる場合に便利です。

**使用方法**

コンパイラオプション：

```
--runtime_checking integer_conversion|implicit_integer_conversion
```

IDE: **［プロジェクト］>［オプション］>［ランタイムチェック］>［整数変換］**

1 つまたは複数のモジュールにチェックを適用できます。

明示的なマスクを挿入することで、チェックを回避できます。

```
short f(int x)
{
  return x & 0xFFFF;  /* 値の変更を報告しません */
}
```

**仕組み**

コンパイラは、整数変換およびビットフィールドへのライトアクセスがあるたびにチェックを実行するコードを挿入します。ただし、チェックが失敗しないとコンパイラが判断した場合を除きます。明示的な定数からの変換はチェックされません。

インクリメント / ディクリメント演算子 (++/--) および複合代入（+=、-= など）は、longhand (*var = var op val*) として記述されたときと同じようにチェックされます。

たとえば、++i と i += 1 はどちらも、i = i + 1 と記述されたかのようにチェックされます。この場合、オーバフローのチェックが有効化されていれば加算がチェックされ、変換のチェックが有効になっていれば代入がチェックされます。int と同じ、またはそれ以上のサイズを持つ整数型の場合、変換チェックが失敗することはありません。しかし、より小さい整数型の場合は、この種類の式におけるあらゆる失敗が一般的に変換エラーとなります。この例は以下のようになります。

```
signed char a = 127;
void f(void)
{
  ++a;     /* 変換チェックエラー (128 -> -128) */
  a -= 1; /* 変換チェックエラー (-129 -> 127) */
}
```

コードサイズは大きくなります。つまり、アプリケーションにリソースの制約がある場合、オーバヘッドを最小限に抑えるため、このチェックはモジュール単位で使用してください。

**例**

321 ページの *C-RUN ランタイムエラー解析を使用するにあたって*に記載された手順に従ってください。**[整数変換]** オプションを使用します。

これは、実行時に識別されるソースコードの一例です。

```
int i = 5,j = 0;
char ch = 0;

void conv(void)
{
  ch = i * 100;
}
```

C-RUN は、整数変換の失敗またはビットフィールドのオーバフローのいずれかを報告します。これは、リストされるメッセージの一例です。

C-RUN メッセージ

| メッセージ | | ソースファイル |
|---|---|---|
| Integer conversion failure | | main.c 14:8-14 |
| Conversion changes the value from | 500 (0x000001f4) | |
| to | 244 (0xf4). | |
| コールスタック | | |
| conv | | main.c 15:1-1 |
| main | | main.c 115:3-8 |
| [_call_main + 0x9] | | |

## 符号付きまたは符号なしのオーバフローの検出

**説明**

式の結果がその型の値の表現可能な範囲にあること、およびシフト量が有効であることをチェックします。

シフト演算のオーバフローはチェックされません。これは個別のチェックで処理されます。327 ページの *シフトする際のビット損失または未定義の動作の検出*を参照してください。

**チェックを実行する理由**

符号付きオーバフローの動作は定義されておらず、符号なしのオーバフローの結果が望ましくない切捨てになることがあるためです。シフト演算はチェックされませんが、シフト量はチェックされます。これは、シフト量がマイナスまたは拡張された左オペランドの幅以上の場合に、シフト演算の動作が定義されないためです。

**使用方法**

コンパイラオプション :

```
--runtime_checking signed_overflow|unsiged_overflow
```

IDE: **［プロジェクト］>［オプション］>［ランタイムチェック］>［整数オーバフロー］**

1 つまたは複数のモジュールにチェックを適用できます。

チェックは、たとえばより大きな型で処理をするなどして、以下のように回避することができます。

```
int f(int a, int b)
  { return (int) ((long long) a + (long long) b); }
short g(short a, short b)
  { return (short) ( a + b); } /* 整数拡張が発生 */
```

**仕組み**

コンパイラは、チェックが失敗することがないと判断しない限り、オーバフローする可能性のある各整数演算（+、-、*、/、%、単項の - を含む）、および各シフト演算でチェックを実行するコードを挿入します。

インクリメント / ディクリメント演算子 (++/--) および複合代入（+=、-= など）は、longhand (*var = var op val*) として記述されたときと同じようにチェックされます。

たとえば、`++i` と `i += 1` はどちらも、`i = i + 1` と記述されたかのようにチェックされます。この場合、オーバフローのチェックが有効化されていれば加算がチェックされ、変換のチェックが有効になっていれば代入がチェックされます。`int` と同じ、またはそれ以上のサイズを持つ整数型の場合、変換チェックが失敗することはありません。しかし、より小さい整数型の場合は、この種類の式におけるあらゆる失敗が一般的に変換エラーとなります。この例は以下のようになります。

```
signed char a = 127;
void f(void)
{
  ++a;     /* 変換チェックエラー (128 -> -128) */
  a -= 1; /* 変換チェックエラー (-129 -> 127) */
}
```

コードサイズは大きくなります。アプリケーションにリソースの制約がある場合、オーバヘッドを最小限に抑えるため、このチェックはモジュール単位で使用してください。

**例**

321 ページの *C-RUN ランタイムエラー解析を使用するにあたって*に記載された手順に従ってください。**[整数オーバフロー]** オプションを使用します。

これは、実行時に識別されるソースコードの一例です。

```
unsigned long ovfl(void)
{
  unsigned long ul = i + 0x7fffffff;
  return ul;
}
```

C-RUN は符号付き整数オーバフロー、符号なし整数オーバフロー、またはシフトカウントオーバフローのいずれかを報告します。これは、リストされるメッセージの一例です。

C-RUN メッセージ

| メッセージ | ソースファイル |
|---|---|
| Signed integer overflow | main.c 23:22-35 |
| Result is greater than the largest representable number: | |
| 5 (0x5) + 2147483647 (0x7fffffff). | |
| コールスタック | |
| ovfl | main.c 23:17-36 |
| main | main.c 119:3-8 |
| [_call_main + 0x9] | |

## シフトする際のビット損失または未定義の動作の検出

**説明**

シフト演算にオーバフローがないか、シフトカウントが有効化どうかをチェックします。

**チェックを実行する理由**

符号付きオーバフローの動作は定義されておらず、符号なしのオーバフローの結果が望ましくない切捨てになることがあるためです。

オーバフローは、左シフト演算 `E1<<E2` で、`E1` がマイナスの場合、あるいは $E1*2^{E2}$ で定義された結果が、その型の表現可能な値の範囲にない場合に発生します。

**使用方法**

コンパイラオプション：`--runtime_checking signed_shift|unsigned_shift`

IDE: **［プロジェクト］** > **［オプション］** > **［ランタイムチェック］** > **［整数シフトオーバフロー］**

1 つまたは複数のモジュールにチェックを適用できます。

シフトの前にマスクをすれば、チェックを回避できます。

```
/* オーバフローしません */
int f(int x) { return (x & 0x00007FFF) << 16; }
```

**仕組み**

コンパイラは、チェックが失敗することがないと判断した場合を除いて、各シフト演算についてチェックを実行するコードを挿入します。

コードサイズは大きくなります。つまり、アプリケーションにリソースの制約がある場合、オーバヘッドを最小限に抑えるため、このチェックはモジュール単位で使用してください。

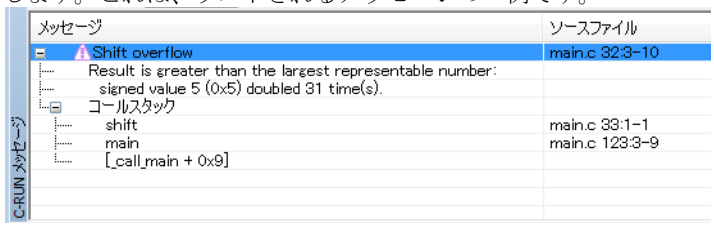
**例**

321 ページの *C-RUN ランタイムエラー解析を使用するにあたって*に記載された手順に従ってください。**［整数シフトオーバフロー］**オプションを使用します。

これは、実行時に識別されるソースコードの一例です。

```
void shift(void)
{
  i <<= 31;
}
```

C-RUN は、シフトオーバフローとシフトカウントオーバフローのいずれかを報告します。これは、リストされるメッセージの一例です。




## ゼロによる除算の検出

**説明**

ゼロによる除算とゼロによる剰余がないかチェックします。浮動小数点演算に、正確な（正の）ゼロによる除算がないかチェックします。

**チェックを実行する理由**

ゼロによる整数の除算の動作は定義されておらず、正確なゼロによる浮動小数点の除算には通常問題があるためです。

**使用方法**

コンパイラオプション：`--runtime_checking division_by_zero`

IDE: **［プロジェクト］>［オプション］>［ランタイムチェック］>［ゼロによる除算］**

1 つまたは複数のモジュールにチェックを適用できます。

**仕組み**

コンパイラは、チェックが失敗することがないと判断した場合を除いて、それぞれの除算と剰余の演算についてチェックを実行するコードを挿入します。

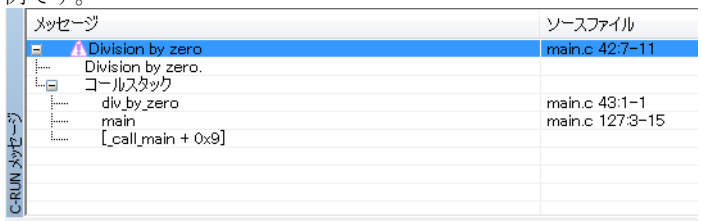
**例**

321 ページの *C-RUN ランタイムエラー解析を使用するにあたって*に記載された手順に従ってください。**［ゼロによる除算］**オプションを使用します。

これは、実行時に識別されるソースコードの一例です。

```
void div(void)
{
  j = i / j;
}
```

C-RUN はゼロによる除算を報告します。これは、リストされるメッセージの一例です。




## switch 文における未処理のケースの検出

**説明**

デフォルトのラベルを持たない `switch` 文に、足りない `case` ラベルがないかチェックします。

**チェックを実行する理由**

このチェックは、たとえば `enum` 型にまだ `switch` 文で処理されていない新しい値が追加されている場合などに便利です。

**使用方法**

コンパイラオプション：`--runtime_checking switch`

IDE: **［プロジェクト］>［オプション］>［ランタイムチェック］>［Switch］**

1 つまたは複数のモジュールにチェックを適用できます。

このチェックは、`default` ラベルを追加すると回避できます。

**仕組み**

コンパイラは、`default` ラベルを持たない各 `switch` 文について、チェックを実行する暗黙的な `default` ラベルを挿入します。

**例**

321 ページの *C-RUN ランタイムエラー解析を使用するにあたって*に記載された手順に従ってください。**[Switch]** オプションを使用します。

これは、実行時に識別されるソースコードの一例です。

```
void sw(void)
{
  switch(ch)
  {
  case 0: i = 3; break;
  case 5: i = 2; break;
  }
}
```

C-RUN は `switch` 文における未処理のケースを報告します。これは、リストされるメッセージの一例です。


## 配列およびその他のオブジェクトの境界の外にあるアクセスの検出

**説明**

ポインタ式からのアクセスがオブジェクトの境界内にあるかチェックします。オブジェクトはどんな型でもよく、グローバルやスタックまたはヒープ上など、どこにでも存在することができます。

**チェックを実行する理由**

このチェックは、アプリケーションが本来は許可されていない場所にリードやライトを行うときに便利です。次に例を示します。

```
int arr[10] = {0};
int f(int i)
{
  return arr[i];
}
int g(void)
{
  return f(20); /* arr [20 は境界の外にあります] */
}
```

**使用方法**

コンパイラオプション：`--runtime_checking bounds`

IDE:［**プロジェクト**］>［**オプション**］>［**ランタイムチェック**］>［**境界チェックを有効化**］

これによって、境界チェックをグローバルで有効化します。グローバルの境界チェック、および各ソースファイルのチェックを微調整するためのサブオプションが使用できます。

**仕組み**

ポインタ境界が追跡されているコード内。

- ポインタ値が渡されるたびに、そのポインタ値の境界も渡されます。
- 何らかのオブジェクトをポイントするようにポインタが初期化される場合、そのポインタの境界はオブジェクトの境界に設定されます。オブジェクトが配列の場合は、境界は配列全体をカバーします。単独のインスタンスであれば、境界はその単独のインスタンスをカバーします。
- ポインタが絶対アドレスに初期化される場合、ポインタは指定した型の単一のオブジェクトをポイントするという前提になります。次に例を示します。

  ```
  uint32_t * p = (uint_32_t *)0x100;
  ```

  この場合、`p` はアドレス `0x100` で 32 ビットの符号なしの整数をポイントし、境界は `0x100` および `0x104` となります。
- null ポインタには、いずれのアクセスもカバーしない境界が割り当てられます。つまり、それを通したアクセスは間違ったものになります。
- ポインタの値がパラメータとして関数に渡されるとき、境界は追加のパラメータとして渡されます。
- ポインタの値が関数から返されるとき、返される値と境界は実際のリターン値として `struct` で渡されます。
- ポインタ経由でアクセス可能な形でポインタの値がメモリに保存される場合、その境界はグローバル境界テーブルに格納されます。ポインタの値がアクセスされるときは、グローバル境界テーブルにある関連の境界も常に読み込まれます。グローバル境界テーブルのサイズは、［**エントリ数**］（リンカオプション `--bounds_table_size number_of_records[:number_of_buckets]|(number_of_bytes)`）を使用すれば変更できます。
- その他の場合は、境界は追加のローカル変数内に格納されます。

ポインタ式を介してアクセスがあるたびに、計算されたアドレス、計算済みアドレスにアクセスサイズを足したものが境界に対してチェックされます。2 つのアドレスのうちどちらかが境界の外にある場合、C-RUN メッセージが生成されます。

任意のパラメータでポインタを受け取る関数、またはポインタ値を返す関数は、2 つの派生型（境界チェックありと境界チェックなし）を持つことができます。

### リソースの使用

境界チェックのオーバヘッドによって、使用可能な ROM や RAM にアプリケーションが配置できなくなることがあります。この問題を解決するには、以下の方法があります。

- 間接的にアクセスされるポイントがアプリケーションに多くない場合、グローバル境界テーブルを縮小して、使用される RAM の量を少なくします。350 ページの *--bounds_table_size（リンカオプション）*を参照してください（IDE では**エントリ数**）。

  デフォルトでは、約 190 KB を必要とする 4 KB のエントリが使用されます。
- 一部のモジュールでは実際の境界チェックをオフにすることができます。これによって、実装によって追加されるコードの量が少なくなります。
- 一部のモジュールではポインタの境界追跡をオフにすることができます。これにより、それらのモジュールにおけるコードサイズが大きくなることは一切なくなりますが、ポインタ境界を追跡するコードと追跡しないコード間のインタフェースに問題が発生します。これについて詳しくは、次のセクションを参照してください。

### チェックされていないコード

アプリケーション全体で境界チェックを有効化できない場合もあります。たとえば、アプリケーションの一部が外部でビルドされたライブラリであったり、アセンブラで記述されているときです。境界チェックのためにコードが機能するように何らかのソースコードを追加する場合、プリプロセッサシンボル `__AS_BOUNDS__` を使用して追加のソースコードを条件付きにしてください。

- 境界を追跡するコードから境界を追跡しないコードを呼び出す

  これは、パラメータまたはリターン型としてポインタを持つ関数のみに影響します。

  宣言で `#pragma no_bounds` または `#pragma default_no_bounds` を使用することにより、特定の関数でポインタ境界を追跡しないように指定できます。ポインタ境界を追跡しないコードからこうした関数を呼び出す場合、追加の非表示パラメータは渡されず、オプション［**C-RUN が有効でない関数からのポインタをチェック**］が使用されているかどうかに応じて（コンパイラオプション `--ignore_uninstrumented_pointers`）、返されたポインタはすべて「安全でない」（こうしたポインタ経由のチェック済みアクセスはすべてエラーとなります）または「安全」（このポインタを介したアクセスは失敗しません）のどちらかと見なされます。このような値に明示的に境界を指定するには、内蔵の演算子 `__as_make_bounds` を使用します。

次に例を示します。

```
#pragma no_bounds
struct X * f1(void);
...
{
  struct X *px = f1();
  /* 1 つの X 構造体へのアクセスを許可する境界を設定。
     （ポインタが NULL でも問題ない場合は、それもチェックの必要あり） */
  if (px)
    px = __as_make_bounds(px, 1);
  /* ここから、ポインタを介したすべてのアクセスがチェックされ、
     構造体の内部にあることを確認。*/
```

- 境界を追跡しないコードから境界を追跡するコードを呼び出す

  境界を追跡する関数、パラメータとしてポインタを持つ関数、境界を追跡しないコードからポインタを返す関数を呼び出す場合、リンクの際には通常 undefined external エラーが表示されます。このような呼出しを有効にするには、#pragma generate_entry_without_bounds またはオプション［**C-RUN が有効でないコードから呼出し可能な関数を生成**］（コンパイラオプション --generate_entries_without_bounds）を使用して、境界を追跡しないコードから呼出し可能な 1 つまたは複数の追加関数を出力するようコンパイラに指示することができます。こうした関数は単に、それぞれがデフォルトの境界を持つ関数を呼び出します。こうした関数は単に、それぞれのデフォルトの境界を持つ関数を呼び出します。デフォルトの境界は、オプション［**有効でない関数からのポインタ**］（コンパイラオプション --ignore_uninstrumented_pointers）を使用しているかどうかに応じて、"safe"（このポインタを介したアクセスではエラーは一切発生しません）または "unsafe"（このポインタを介したアクセスでは常にエラーが発生します）のどちらかです。

  この場合に、より正確な境界を指定するには、
  #pragma define_without_bounds を使用します。

  このプラグマディレクティブには 2 つの使用方法があります。問題の関数がポインタ境界を追跡しないコードからのみ呼び出され、境界が既知であるか、他のパラメータから推測できる場合、2 つの関数は必要ありません。#pragma define_without_bounds を使用して定義を変更するだけですみます。

  次に例を示します。

```
#pragma define_without_bounds
int f2(int * p, int n)
{
  p = __as_make_bounds(p, n); /* p 境界を指定 */
  ...
}
```

この例では、p は n 個の整数からなる配列をポイントすると想定します。割当ての後、p の境界は p および p + n となります。

関数がポインタ境界を追跡するコードと追跡しないコードの両方から呼び出せる場合、代わりに #pragma define_without_bounds を使用して、境界情報を持たない関数の追加の派生型で境界情報を持つ派生型を呼び出すものを定義することができます。

同じ翻訳単位に、境界を持たない派生型と境界を持つ派生型を定義することはできません。

次に例を示します。

```
#pragma define_without_bounds
int f3(int * p, int n)
{
  return f3(__as_make_bounds(p, n), n);
}
```

この例では、p は n 個の整数からなる配列をポイントすると想定します。ここで定義されている追加の境界情報を持たない f3 の派生型は、追加の境界情報 ("f3 [with bounds]") を持つ f3 の派生型を呼び出し、ポインタのパラメータに p と p + n の境界を渡します。

- 境界を追跡しないコードで定義されるポインタを持ったグローバル変数

  これらのポインタは、あらゆるアクセスに対してエラーを発信する境界、あるいはリンクの際にオプション［**C-RUN が有効でない関数からのポインタをチェック**］（リンカオプション --ignore_unistrumented_pointers）が使用される場合は、発信するエラーが一切発生しない境界のどちらかを取得します。より具体的な境界が必要な場合は、__as_make_bounds を使用してください。

  次に例を示します。

  ```
  extern struct x * gptr;
  int main(void)
  {
    /* サイズ N の gptr 境界を提供 */
    gptr = __as_make_bounds(gptr, N);
    ...
  }
  ```

- RTOS タスク

  タスクを実装する関数は、ポインタであるパラメータにより呼び出されることがあります。RTOS 自体がポインタ境界を追跡しない場合は、#pragma define_without_bounds および __as_make_bounds を使用して正しい境界情報を取得する必要があります。

次に例を示します。

```
#pragma define_without_bounds
void task1(struct Arg * p)
{
  /* p は 1 つの Arg 構造体をポイントします */
  p = __as_make_bounds(p, 1);
  ...
}
```

一部の制限事項:

- 関数ポインタ

  関数ポインタを境界を追跡するコードと追跡しないコード間で共有すると、問題が発生する可能性があります。

  境界を追跡するコードと追跡しないコードでは、型に違いがありません。どちらの種類の関数も関数ポインタに割り当てたり、関数ポインタのパラメータを受け入れる関数に渡すことができます。ただし、署名にポインタを含む関数が一致しない状況で呼び出された場合（境界を追跡しないコードから境界を追跡する関数を呼び出したり、その逆)、正しく機能しなくなります。最も理想的な場合には、これは紛らわしい境界の違反ということになりますが、これらの関数は間違った数の引数によって呼び出されているため、どんな動作でも実際に起こる可能性があります。

  正しく機能するためには、ポインタが署名に含まれるすべての関数、および関数ポインタを介して呼び出されるすべての関数が適切な種類でなければなりません。境界を追跡しないライブラリからのコールバックの単純なケースでは、適切な関数上で `#pragma no_bounds` を使用すれば通常は十分です。

- K&R スタイル関数

  K&R 関数は使用しないでください。すべての関数に正しいプロトタイプがあることを確認するには、`--require_prototypes` と共有のヘッダファイルを使用します。C では `void f()` は K&R 関数ですが、`f(void)` はそうではない点に注意してください。

- 境界を追跡しないコードにより更新されるポインタ

  ポインタに新しい境界を設定しないコードによってポインタが更新される場合は、常に潜在的な問題があります。新しいポインタ値が古いポインタ値と同じオブジェクトにポイントしない場合、境界が正しいものではなくなり、チェック済みコードでこのポインタを介したアクセスはエラーを発信します。

### 絶対アドレス

`#pragma location` または `@` 演算子を使用して絶対アドレスに変数を配置する場合、これらの変数へのポインタは、他の任意の変数へのポインタと同じように正しい境界を取得します。

整数からポインタへ明示的なキャストを使用する場合、ポインタは境界を取得します。このポインタが指定した型の 1 つのオブジェクトをポイントすることが前提となります。他の境界が必要な場合は、__as_make_bounds を使用します。

次に例を示します。

```
/* p は 1 つの構造体をポイントすることが前提の境界を取得します
   アドレス 0x1000 のポート */
p = (struct Port *)0x1000;
/* 3 つの構造体からなる配列をポイントする場合、以下を追加できます */
p = __as_make_bounds(p, 3);
```

**例**

321 ページの *C-RUN ランタイムエラー解析を使用するにあたって*に記載された手順に従ってください。［**境界チェック**］オプションを使用します。

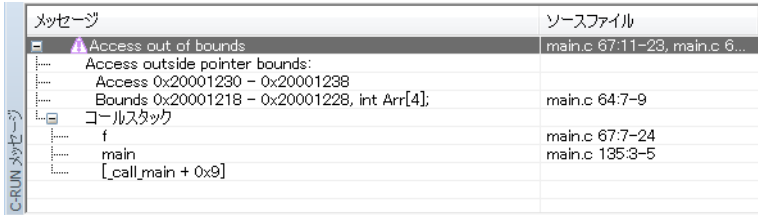
これは、実行時に識別されるソースコードの一例です。

```
int Arr[4] = { 0, 1 ,2, 3};

int ArrI = 5;

int f(void)
{
  int i = Arr[ArrI + 1]; // Double fail global
  i += Arr[ArrI + 2];
  return i;
}
```

C-RUN は、境界の外のアクセスまたは無効な関数ポインタのどちらかを報告します。これは、リストされるメッセージの一例です。




## ヒープ使用エラーの検出

**説明**

ヒープのインタフェース（malloc、new、free など）がアプリケーションにより適切に使用されているかどうかをチェックします。以下の不適切な使用がチェックされます。

- 割当て（malloc、new など）に対する正しくない割当て解除（free、delete など）。たとえば以下のようになります。

```
char * p1 = (char *)malloc(23); /* malloc を使用した割当て */
char * p2 = new char[23];       /* new[] を使用した割当て */
char * p3 = new int;            /* new を使用した割当て */
delete p1         /* エラー。malloc を使用した割当て */
free(p2);         /* エラー。new[] を使用した割当て */
delete[] p3;      /* エラー。new を使用した割当て */
```

- ヒープブロックを 2 回以上解放しているかどうか。
- 大きすぎるヒープブロックを割り当てようとしているかどうか。

**チェックを実行する理由**

ヒープのインタフェースが正しく使用されているか確認します。

**使用方法**

リンカオプション：`--debug_heap`

IDE: **［プロジェクト］>［オプション］>［ランタイムチェック］>［チェック済みヒープの使用］**

チェック済みヒープは、アプリケーション全体にわたって通常のヒープを置換します。チェック済みヒープには、追加のヒープとスタックリソースが必要です。アプリケーションに 10 KB 以上のヒープと 4 KB のスタックがあることを確認してください。

割当て時のヒープブロックの最大サイズは、デフォルトで 1 GB です。この上限は以下の関数により変更できます。

```
size_t __iar_set_request_report_limit(size_t value);
```

この関数によって古い上限が返されます。この関数の宣言は、`iar_dlmalloc.h` にあります。詳細については、*『ARM 用 IAR C/C++ 開発ガイド』* を参照してください。

**仕組み**

すべてのヒープインタフェースの間違った使用について、メッセージが表示されます。

319 ページの *ライブラリにより提供されるチェック済みヒープ*を参照してください。

**例**

321 ページの *C-RUN ランタイムエラー解析を使用するにあたって*に記載された手順に従ってください。**［デバッグヒープ］**オプションを使用します。

これは、実行時に識別されるソースコードの一例です。

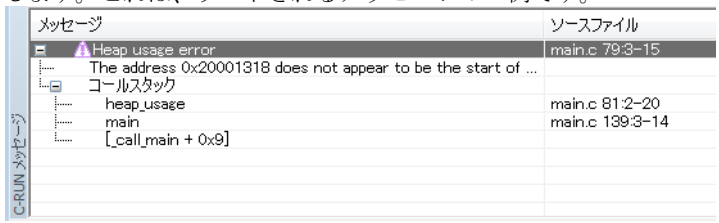
```
int main(void)
{
  char *p = malloc(10);

  free(p + 200);

  __iar_check_leaks(); // Leakage

  return 0;
}
```

C-RUN は、ヒープの整合性違反またはヒープ使用エラーのどちらかについて報告します。これは、リストされるメッセージの一例です。




## ヒープメモリのリークの検出

**説明**

アプリケーション内の選択したポイントに参照を持たないヒープブロックがあるか否かをチェックします。

**チェックを実行する理由**

リークしたヒープブロックは参照できなくなるため、使用したり解放することができません。このチェックを使用して、ヒープブロックへの参照を検出し、参照されていないと思われるブロックを報告します。リークの検出で潜在的なあらゆるメモリリークを見つけることはできない点に注意してください。参照されていないと思われるヒープブロックが実際は参照されていて、参照されているように見えるヒープブロックが実際はリークしてることがあります。

**使用方法**

リンカオプション：`--debug_heap`

IDE: **［プロジェクト］>［オプション］>［ランタイムチェック］>［チェック済みヒープの使用］**

チェック済みヒープは、アプリケーション全体にわたって通常のヒープを置換します。チェック済みヒープには、追加のヒープとスタックリソースが必要です。アプリケーションに 10 KB 以上のヒープと 4 KB のスタックがあることを確認してください。

リークの検出チェックは手動で呼び出す必要があります。アプリケーションの終了時に呼び出すか、2 つのソースポイント間でリークしたヒープブロック

を検出するために使用できます。これらの関数は iar_dlmalloc.h: に定義されています。

- `void __iar_leaks_ignore_all(void);`

  この関数を使用して、以降のヒープのリークチェックで無視する現在割り当てられたすべてのヒープブロックにマークを付けます。

- `void __iar_leaks_ignore_block(void *block);`

  この関数を使用して、以降のヒープのリークチェックで無視する割り当てられた特定のヒープブロックにマークを付けます。

- `void __iar_check_leaks(void);`

  この関数を使用して、リークがないかチェックします。

### 仕組み

チェック済みヒープは、アプリケーション全体にわたって通常のヒープを置換します。ヒープのリークアルゴリズムには以下の 3 つのフェーズがあります。

1 ヒープをスキャンして、割り当てられたすべてのヒープブロックのリストを作成します。

2 ヒープ内にアドレスがあるかどうか、静的に使用される RAM、スタックなどをスキャンします。上記のリストにあるヒープブロックに一致するアドレスがある場合、それらはリストから削除されます。

3 リストにある残りのヒープブロックをリークしたものとして報告します。

319 ページの *ライブラリにより提供されるチェック済みヒープ*を参照してください。

### 例

321 ページの *C-RUN ランタイムエラー解析を使用するにあたって*に記載された手順に従ってください。［**デバッグヒープ**］オプションを使用します。

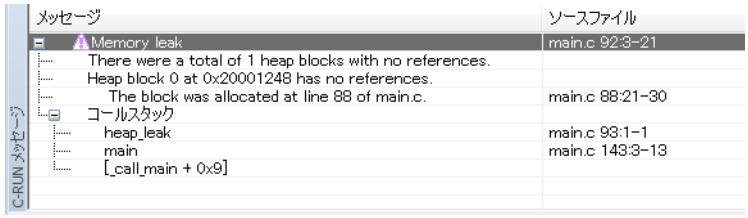
これは、実行時に識別されるソースコードの一例です。

```
char *p = malloc(10);
p = malloc(20);

__iar_check_leaks();

return *p;
```

C-RUN はメモリリークを報告します。これは、リストされるメッセージの一例です。




## ヒープの整合性違反の検出

**説明**

さまざまなヒープの整合性違反をチェックします。チェックは手動でトリガしたり、ヒープインタフェースの使用に対して定期的な間隔でトリガされるよう設定できます。このチェックを有効にして検出できる整合性の問題には、以下のものがあります。

- 内部ヒープ構造の破壊。ほとんどの場合、これはポインタ式からのライトアクセスが正しくないことが原因です。間違ったライトアクセスを探すには、境界チェックを使用してください。
- 割り当てられたメモリの領域外でのライトアクセス。たとえば以下のようなものです。

```
char * p = (char *)malloc(100); /* メモリが割り当てられます。*/
...
p[100] = ... /* このライトアクセスは境界の外にあります。*/
```

  ヒープブロックの境界の外でのライトアクセス、およびヒープブロックの前後でガードを変更するライトアクセスが検出されます。その他のライトアクセスは検出されません。
- 解放済みメモリに対するライトアクセス。たとえば以下のようなものです。

```
char * p = (char *)malloc(...); /* メモリが割り当てられます。*/
...
free(p); /* メモリが解放されます。*/
...
p[...] = ... /* 解放済みメモリへのライトアクセス。*/
```

  元の p を含むメモリが p の書込みより前に再び割り当てられる場合、このエラーは通常は検出されません。遅延のある解放リスト（以下を参照）を使用することで、このエラーを検出できます。

**チェックを実行する理由**

たとえばヒープブロックの誤用などの理由で、アプリケーションが間違ってある時点でヒープに書き込んでいると思われる場合は、チェック済みヒープを使用してください。

**使用方法**

リンカオプション：`--debug_heap`

IDE: **［プロジェクト］>［オプション］>［ランタイムチェック］>［チェック済みヒープの使用］**

チェック済みヒープは、アプリケーション全体にわたって通常のヒープを置換します。チェック済みヒープには、追加のヒープとスタックリソースが必要です。アプリケーションに 10 KB 以上のヒープと 4 KB のスタックがあることを確認してください。

ヒープの整合性違反を検出する目的で、以下の関数が `iar_dlmalloc.h` に定義されています。

- `void __iar_check_heap_integrity(void);`

  この関数を使用して、ヒープの整合性を検証します。損傷が見つかった場合は、それが報告されます。報告される損傷エラーの件数には限りがあります。この上限は、`__iar_set_integrity_report_limit` 関数を使用して変更できます。最終のメッセージが生成されたときのみ、実行が停止します。報告されるメッセージのデフォルト件数は 10 です。
  `__iar_check_heap_integrity` の呼出しでは、ヒープに損傷がある場合に呼出し元に返すという保証はありません。

- `size_t __iar_set_heap_check_frequency(size_t interval);`

  この関数を使用して、定期的にヒープ整合性チェックを実行する頻度を指定します。デフォルトでは、定期的チェックはオフに設定されています (*interval* = 0)。*interval* が正の数値の場合、整合性は `interval`: 回目のヒープ演算ごとにチェックされ、`free`/`malloc`/`new`/`delete`/`realloc`/ などの呼出しが 1 回の演算として数えられます。この関数は古い間隔を返します。つまり、必要があれば状態を復元できます。ヒープチェックは、アプリケーションプログラムの信頼できる部分を実行する際は間隔を長くしたり、オフにすることも可能です。また、アプリケーションでヒープエラーが含まれる可能性が高い部分を実行するときは、チェックの間隔を小さくすることができます。

- `size_t __iar_set_delayed_free_size(size_t size);`

  この関数を使用して、解放済み遅延リストの最大サイズを指定します。デフォルトでは、解放済み遅延リストはオフに設定されています (*size* = 0)。この関数はリストの実際のサイズには影響せず、最大値のみを変更します。この関数は前回の値を返すため、必要があれば復元が可能です。

  解放済み遅延リストを使用して、解放済みヒープブロックを使用するアプリケーション内のアドレスを見つけます。以下を検出する際に役立ちます。

  - 解放済みの古いヒープブロックポインタが、新しい割当済みのヒープブロックポインタと混在している場合。解放済み遅延リストは解放済み

ヒープブロックの再利用を遅らせるため、アプリケーションの動作が変わり、この種類の問題を検出できることがあります。

- すでに解放されているヒープブロックに書き込みます。ヒープブロックが解放済み遅延リストにあると、実際に解放されるときのものとは異なる特定の内容を取得し、ヒープ整合性チェックでヒープブロックに対するこれらの間違ったライトアクセスを検出することができます。

- `size_t __iar_free_delayed_free_size(size_t count);`

  この関数を使用して、解放済み遅延リストにほとんどの *count* エレメントがあるか確認します。不要なエレメントは解放されます（最も古いものから変更されます)。リストの最大サイズには影響しません。現在のエレメントの数のみが変更されます。この関数を呼び出しても、*count* がリストの現在のサイズより大きい場合には効果はありません。この関数は解放済みエレメントの数を返します。

### 仕組み

チェック済みヒープは、アプリケーション全体にわたって通常のヒープを置換します。

*解放済み遅延リスト*は、`free` 呼出しのキューメカニズムです。`free`、またはメモリをヒープに返す同等のメモリ処理を呼び出す際、最近解放されたポインタが、実際に解放される代わりに、解放待ちのキューに入ります。遅延リストの最大サイズを超過すると、解放済み遅延リストで最大サイズを超えた最も古いエレメントが実際に解放されます。

チェック済みヒープで報告されるすべてのエラーは、何らかの理由で損傷したヒープブロックについて言及します。チェック済みヒープは、ヒープブロックを損傷したユーザや損傷した時期については情報を提供しません。`__iar_debug_check_heap_integrity` 関数を呼び出して、アプリケーション実行時の整合性を検証し、潜在的な候補のリストを絞り込むことができます。

次に例を示します。

```
...
__iar_debug_check_heap_integrity(); /* 事前のチェック */
my_function(..., ..., ...);
__iar_debug_check_heap_integrity(); /* 事後のチェック */
...
```

事後のチェックで事前のチェックで報告されない問題が見つかった場合、`my_function` でヒープが損傷した可能性があります。

チェック済みヒープはリソースを消費します。

- チェック済みヒープは、通常のヒープ実装に比べてより多くの ROM 容量を必要とします。
- すべてのヒープ処理にはチェック済みヒープでより多くの時間が必要です。

- チェック済みヒープの各ヒープブロックには、記録のための追加の容量が含まれます。このため、アプリケーションの RAM 使用量が増えることになります。

319 ページの *ライブラリにより提供されるチェック済みヒープ*を参照してください。

**例**

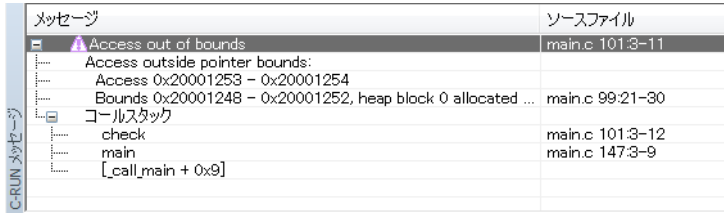
321 ページの *C-RUN ランタイムエラー解析を使用するにあたって*に記載された手順に従ってください。**［デバッグヒープ］**オプションを使用します。

これは、実行時に識別されるソースコードの一例です。

```
void check(void)
{
  char *p = malloc(10);

    p[11] = 1;
  __iar_check_heap_integrity();
}
```

C-RUN はヒープの整合性違反を報告します。これは、リストされるメッセージの一例です。



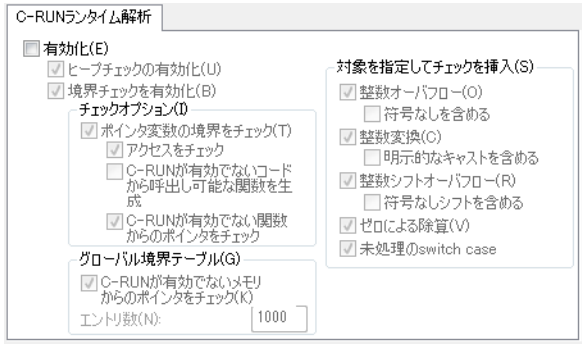

## ランタイムエラー解析のリファレンス情報

リファレンス情報：

- 344 ページの *C-RUN ランタイム解析のオプション*
- 346 ページの *［C-RUN メッセージ］ウィンドウ*
- 348 ページの *［C-RUN メッセージルール］ウィンドウ*

## C-RUN ランタイム解析のオプション

［C-RUN ランタイム解析］オプションにより、実行時に実行するチェックを決定します。




### 有効化

ランタイム解析を有効にします。

### ヒープチェックの有効化

ヒープチェックを使用して、ヒープの使用に関するエラーを検出します。

### 境界チェックを有効化

配列およびその他オブジェクトの境界の外にあるアクセスをチェックします。使用可能なチェック：

#### ポインタ変数の境界をチェック

コンパイラでポインタの境界を追跡するコードを追加します。以下から選択してください。

| | |
|---|---|
| **アクセスをチェック** | ポインタ経由のアクセスがあるかどうかチェックするコードを挿入します。 |
| **C-RUN が有効でないコードから呼出し可能な関数を生成** | ポインタ境界を追跡するためにコードが追加される際、ポインタを含む型を返す関数あるいは受け取る関数は、ポインタ境界を返す / 受け取るために修正されます。このオプションを使用して、そのような関数（未チェックのコードから呼出し可能です）ごとに追加のエントリを生成します。 |

**C-RUN が有効でない関数からのポインタをチェック**
境界チェックが有効なコード（つまり、ポインタ境界を追跡するコード）においては、境界チェックが有効でない関数から発生するポインタには、このオプションに応じて境界が設定されます。このオプションを使用する場合、そのようなポインタを経由するすべてのアクセスでエラーが発生します。このオプションを使用しない場合、こうしたポインタを経由するすべてのアクセスでエラーが発生しません。

**C-RUN が有効でないメモリからのポインタをチェック**
境界チェックが有効なコード（つまり、ポインタ境界を追跡するコード）では、ポインタがメモリからロードされるたびに、グローバル境界テーブルでその境界が照会されます。このポインタのエントリがテーブルに見つからない場合は通常、ポインタが有効でないコードにより作成されたためで、このオプションの設定に応じて境界が設定されます。このオプションを使用する場合、そのようなポインタを経由するすべてのアクセスでエラーが発生します。このオプションを使用しない場合、こうしたポインタを経由するアクセスではエラーが発生しません。

**エントリ数**
境界チェックシステムは、別のテーブルを使用してメモリ内のポインタの境界を追跡します。このオプションを使用して、同時に追跡可能なこうした境界の数を設定します。テーブルは、ポインタあたり約 50 バイトを使用します。

### チェックを挿入する対象

チェックを挿入する対象 :

**整数オーバフロー**
整数演算に符号付きオーバフローがあるかチェックします。整数演算に符号なしのオーバフローがあるかもチェックする場合は［**符号なしを含める**］を使用して下さい。

**整数変換**

暗黙的な整数変換によって値が変更されているかチェックします。明示的なキャストがあるかどうかもチェックするには、**[明示的なキャストを含める]**を使用してください。

**整数シフトオーバフロー**

シフト演算にオーバフローがあるかチェックします。シフト演算に符号なしのオーバフローがあるかどうかもチェックするには、**[符号なしシフトを含める]**を使用してください。

**ゼロによる除算**

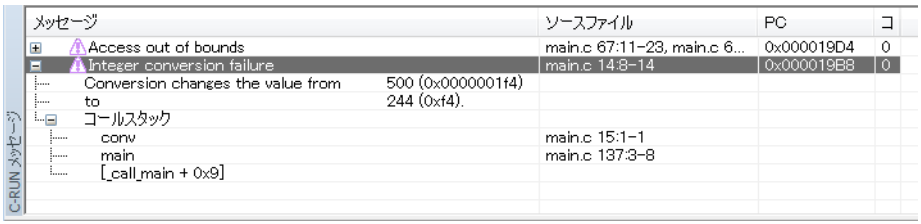
ゼロによる除算があるかチェックします。

**未処理の switch case**

`switch` 文に未処理のケースがあるかチェックします。

## [C-RUN メッセージ] ウィンドウ

[C-RUN メッセージ] ウィンドウは **[表示]** メニューから利用できます。




このウィンドウには、ランタイムチェックで検出されたランタイムエラーに関する情報が表示されます。同じソース文、コールスタック、メッセージを持つウィンドウグループのメッセージ。

### 要件

C-RUN 製品のライセンス。

### ツールバー

ツールバーの内容は以下のとおりです。

**デフォルトのアクション**

他のルールの条件が満たされない場合のデフォルトのアクションを設定します。**[停止]**、**[ログ]**、**[無視]** から選択してください。

**フィルタ**

指定したテキストを含むメッセージのみが一覧表示されるように、メッセージのリストをフィルタリングします。これは、メッセージテキスト、コールスタックのエントリ、ファイル名を検索する際に役立ちます。

## 表示エリア

表示エリアには、前回のリセット以降に検出されたすべてのエラーが表示されます。

表示エリアには以下の情報が表示されます。

**メッセージ**

検出されたランタイムエラーについての情報。各メッセージには見出し、エラーの詳細な情報、エラーの位置に関するコールスタック情報が含まれます。

**PC**

ランタイムエラーが検出されたときの PC の値。

**ソースファイル**

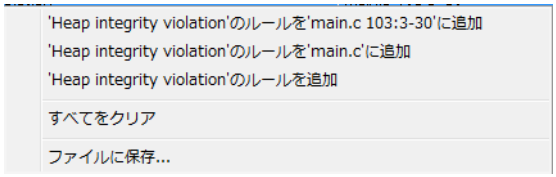
ランタイムエラーが検出されたソースファイル名、または変数の定義など適切な位置。

**コア**

チェックを実行した CPU コア（マルチコア環境の場合）。

## コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**[*のルールを範囲に追加*]**

この位置のランタイムチェックに一致するルールを追加します。

**[*ファイル名にルールを追加*]**

指定したファイルに含まれる、この種類のランタイムチェックに一致するルールを追加します。

**ルールを追加**

この種類のランタイムチェックに一致するすべてのルールを追加します。

**すべてをクリア**

ウィンドウからすべての内容をクリアします。

**ファイルに保存**

ダイアログボックスを開き、テキストまたは XML 形式のどちらで内容をファイルに保存するか選択します。

## ［C-RUN メッセージルール］ウィンドウ

［C-RUN メッセージルール］ウィンドウは［**表示**］メニューから利用できます。


このウィンドウには、［C-RUN メッセージ］ウィンドウでどのようにメッセージを報告するかを制御するルールが表示されます。潜在的なエラーが検出されると、そのエラーはこれらのルールとチェックされ（上から下の順に）、最初に一致したルールによって実行するアクションが決まります。一番下には常に、すべてのメッセージに一致する全条件をカバーするルールがあります。このルールは、［C-RUN メッセージ］ウィンドウの［**デフォルトのアクション**］を使用して変更できます。

* をワイルドカードとして使用します。

### 要件

C-RUN 製品のライセンス。

### 表示エリア

表示エリアでは、以下の列に情報が表示されます。

**チェック**

このルールに一致するランタイムエラーの名称。

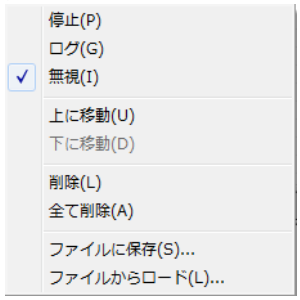
**ソースファイル**
一致するソースファイル名、および一致するファイル内の位置。

**アクション**
ルールに一致したエラーに対して実行するアクション。

- **［停止］**は実行を停止し、エラーを記録します。
- **［ログ］**はエラーを記録しますが、実行を続けます。
- **［無視］**では記録と停止のどちらも行われません。

## コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**停止 / ログ / 無視**
メッセージが選択したルールに一致するときに実行するアクションを選択します。

**上に移動 / 下に移動**
選択したルールを上または下に 1 つ移動します。

**削除**
選択したルールを削除します。

**全て削除**
すべてのルールを削除します。

**ファイルに保存**
ダイアログボックスを開き、ルールを保存することができます（**［ファイルからロード］**を参照）。358 ページの *--rtc_rules* を参照してください。

**ファイルからロード**

ダイアログボックスを開き、ファイルからルールをロードすることができます。

# C-RUN のためのコンパイラおよびリンカのリファレンス

リファレンス情報 :



## --bounds_table_size（リンカオプション）

**構文**

```
--bounds_table_size records[:buckets]|(bytes)
```

**パラメータ**

| | |
|---|---|
| *records* | レコード件数。 |
| :*buckets* | バケット数。 |
| (*bytes*) | バイト数。 |

**説明**

このリンカオプションを使用して、メモリ内のポインタの境界を追跡するために使用するグローバル境界テーブルのサイズを指定します。

テーブル内のレコード数を指定できます（境界を保持できるポインタの数）。その場合、バケット数（2 乗）も指定でき、ルックアップの速度が影響を受けます。指定しない場合はバケット数は 2 乗となり、レコード数の少なくとも 6 倍です。

または、レコードとバケットに使用する合計バイト数を指定することもできます。

**関連項目**

330 ページの*配列およびその他のオブジェクトの境界の外にあるアクセスの検出*。

**［プロジェクト］>［オプション］>［ランタイムチェック］>［エントリ数］**

## --debug_heap（リンカオプション）

**構文**

```
--debug_heap
```

**説明**

このリンカオプションを使用して、チェック済みヒープを使用します。

**関連項目**

319 ページの*ライブラリにより提供されるチェック済みヒープ*。

**［プロジェクト］>［オプション］>［ランタイムチェック］>［チェック済みヒープの使用］**

## --generate_entries_without_bounds

**構文**

```
--generate_entries_without_bounds
```

**説明**

このコンパイラオプションを使用して、C-RUN が有効化されていないコードから追加の関数を生成します。このオプションでは、境界チェックを有効化する必要があります。

**関連項目**

330 ページの*配列およびその他のオブジェクトの境界の外にあるアクセスの検出*。

**［プロジェクト］>［オプション］>［ランタイムチェック］>［C-RUN が有効でないコードから呼出し可能な関数を生成］**

## --ignore_uninstrumented_pointers

**構文**

```
--ignore_uninstrumented_pointers
```

**説明**

このコンパイラオプションを使用して、C-RUN が有効でない関数からのポインタを経由するアクセスのチェックを無効にします。

**関連項目**

330 ページの*配列およびその他のオブジェクトの境界の外にあるアクセスの検出*。

**［プロジェクト］>［オプション］>［ランタイムチェック］>［C-RUN が有効でない関数からのポインタをチェック］**

## --ignore_uninstrumented_pointers（リンカオプション）

**構文**

```
--ignore_uninstrumented_pointers
```

**説明**

このリンカオプションを使用して、境界が設定されていないメモリ内でのポインタを経由したアクセスのチェックを無効化します。

**関連項目**

330 ページの*配列およびその他のオブジェクトの境界の外にあるアクセスの検出*。

**［プロジェクト］>［オプション］>［ランタイムチェック］>［C-RUN が有効でないメモリからのポインタをチェック］**

## --runtime_checking

**構文**

```
--runtime_checking param ,param, ...
```

**パラメータ**

*param* は以下のいずれかです。

| | |
|---|---|
| `signed_overflow` \| `unsigned_overflow` | 整数演算に符号付きまたは符号なしのオーバフローがあるかチェックします。 |
| `integer_conversion` \| `implicit_integer_conversion` | 暗黙的または明示的な整数変換によって値が変更されているかチェックします。 |
| `division_by_zero` | ゼロによる除算があるかチェックします。 |
| `signed_shift` \| `unsigned_shift` | シフト時にビットの損失または実装に依存する結果があるかどうかチェックします。 |

| | |
|---|---|
| switch | switch 文に未処理のケースがあるかチェックします。 |
| bounds | 配列およびその他オブジェクトの境界の外にあるアクセスをチェックします。 |
| bounds_no_checks | ポインタ境界を追跡しますが、チェックは実行しません。#pragma disable_check = bounds も参照してください。 |

**説明** このコンパイラオプションを使用して、ランタイムエラー解析を有効にします。

**関連項目** 317 ページの *ランタイムエラー解析の概要*。

オプションを設定するには、以下のように選択します。

**[プロジェクト] > [オプション] > [ランタイムチェック]**

## #pragma default_no_bounds

**構文** `#pragma default_no_bounds [=on|=off]`

**パラメータ**

| | |
|---|---|
| on | このポイントから宣言されたすべての関数のデフォルトを、#pragma no_bounds を使用して宣言されたかのようにします。 |
| off | デフォルトをオフにします。 |

**説明** このプラグマディレクティブを使用して、#pragma no_bounds を、たとえば未チェックのコードへのインタフェースを宣言するヘッダファイルの周辺というように、関数の全セットに適用します。

**関連項目** 330 ページの *配列およびその他のオブジェクトの境界の外にあるアクセスの検出*。

## #pragma define_with_bounds

**構文** `#pragma define_with_bounds`

**説明** このプラグマディレクティブは、#pragma no_bounds（または同等のもの）を使用して宣言された関数にしか使用できません。続いて、この関数はポイ

ンタ境界を追跡するように有効化されますが、境界チェックは一切実行されません。この関数の呼出しはすべて、追加の境界情報を持たないバージョンへのものになります。

これは、チェックのないバージョンに基づいてチェックのあるバージョンの関数を記述する際に役立ちます。

## #pragma define_without_bounds

**構文**

```
#pragma define_without_bounds
```

**説明**

このプラグマディレクティブを使用して、追加の境界情報を持たない関数のバージョンを定義します。この関数のコードは、変わりなくポインタの境界を追跡するように有効化されます（また、`#pragma disable_check = bounds` を使用しない限り、チェックも挿入されます）。

ポインタ境界を追跡しないコードから関数が独占的に呼び出され、境界が他の引数から、あるいは他の何らかの方法で推測できる場合に便利です。

**例**

```
/* p は n 個の整数の配列をポイントします */
void fun(int * p, int n)
{
  /* p に境界を設定します */
  p = __as_make_bounds(p, n);
  ...
}
```

## #pragma disable_check

**構文**

```
#pragma disable_check = bounds
```

**パラメータ**

| | |
|---|---|
| `bounds` | 境界に対してアクセスのチェックを実行しません。 |

**説明**

このプラグマディレクティブを使用して、関数の直後に境界に対してアクセスをチェックしないように指示します。境界チェックとともにコンパイルされた場合、関数は境界を追跡するように有効化されますが、チェックは実行されません。

## #pragma generate_entry_without_bounds

**構文**

```
#pragma generate_entry_without_bounds
```

**説明**

このプラグマディレクティブを使用して、関数の直後の境界を持たない追加のエントリの生成を有効化します。この追加エントリ（関数）は、境界チェック用に有効化されていないコードから呼び出すことができます。非表示の追加のパラメータは必要とせず、返されたポインタについて境界の情報も一切追加されません。こうした関数に渡されたポインタにはすべて境界が設定され、あらゆるアクセスについてエラーが発生します。`--ignore_uninstrumented_pointers` を使用する場合、設定された境界によってエラーが発生することはありません。

そのようなエントリが生成されない場合に、関数にこのプラグマディレクティブを使用することがエラーです。これには、引数の変数値を受け入れる関数や、ポインタを含む値を受け入れたり返す 1 つ以上の関数ポインタを受け入れる関数が含まれます。

こうしたエントリを必要としない関数にこのプラグマディレクティブを使用することはエラーではありません（ポインタを受け入れないか、または `#pragma no_bounds` を使用して宣言されているため）。この場合、追加のエントリは生成されません。

**関連項目**

330 ページの*配列およびその他のオブジェクトの境界の外にあるアクセスの検出*。

## #pragma no_bounds

**構文**

```
#pragma no_bounds
```

**説明**

このプラグマディレクティブを使用して、直後の関数を境界チェックのために有効化しないように指示します。この関数を呼び出す際は、追加の境界パラメータは渡されず、ポインタの境界があったとしてもリターン値として返されません。

**関連項目**

330 ページの*配列およびその他のオブジェクトの境界の外にあるアクセスの検出*。

## __as_get_base

**構文**

```
__as_get_base(ptr)
```

**パラメータ**

| | |
|---|---|
| *ptr* | ポインタ。 |

**説明** この演算子を使用して、*ptr* と同じ型のポインタを作成し、*ptr* によってポイントされるベース領域を表します。

**例**
```
base = __as_get_base(my_ptr);
```

## __as_get_bound

**構文**
```
__as_get_bound(ptr)
```

**パラメータ**

| | |
|---|---|
| *ptr* | ポインタ。 |

**説明** この演算子を使用して、*ptr* と同じ型のポインタを作成し、*ptr* によってポイントされる上部のベース領域を表します。

**例**
```
bound = __as_get_bound(ptr);
```

## __as_make_bounds

**構文**
```
__as_make_bounds(ptr, number)
__as_make_bounds(ptr, base, bound)
```

**パラメータ**

| | |
|---|---|
| *ptr* | 境界を持たないポインタ。 |
| *number* | エレメントの数。 |
| *base* | ポイントされるオブジェクトの開始部分。 |
| *bound* | ポイントされるオブジェクトの終了部分。 |

**説明** この演算子を使用して、境界情報を持つポインタを作成します。最初の構文は、*ptr* について、*ptr* から最大 *ptr* と *size* を合わせた分までの領域を作成します。2 番目の構文には明示的な境界があり、*base* 領域の最初のエレメントに対するポインタです。*bound* は、その領域より先に対するポインタです。それぞれの式が 1 度だけ評価される点を除いて、2 つの一パラメータの呼び出しは __as_make_bounds(*ptr*, *ptr*, *ptr* + *size*) と同等です。

**例**
```
/* ここから開始し、p は 1 つのエレメントをポイントします */
p = __as_make_bounds(p, 1);
/* 指定の境界を持つポインタで fun を呼出し */
fun(__as_make_bounds(q, start, end));
```

# C-RUN の cspybat オプション

リファレンス情報 :



## --rtc_enable

**構文**

`--rtc_enable`

このオプションはコマンドライン上で `--backend` オプションより前に配置しなければならない点に注意してください。

**適用範囲**

`cspybat`

**説明**

このオプションを使用して、`cspybat` で C-RUN ランタイムチェックを有効化します。このオプションは、その他 `-rtc_*` オプションを使用する場合に自動的に有効になります。

このオプションは、IDE では使用できません。

## --rtc_output

**構文**

`--rtc_output` *`file`*

このオプションはコマンドライン上で `--backend` オプションより前に配置しなければならない点に注意してください。

**パラメータ**

| | |
|---|---|
| *`file`* | 出力メッセージのファイル。 |

**適用範囲**

`cspybat`

**説明**

このオプションを使用して、C-RUN メッセージ出力の `cspybat` ファイルを、テキスト（ファイル名拡張子 `txt`）または XML（ファイル名拡張子 `xml`）の形式で指定します。

このオプションは、IDE では使用できません。

## --rtc_raw_to_txt

**構文**

```
--rtc_raw_to_txt=file
```

このオプションはコマンドライン上で --backend オプションより前に配置しなければならない点に注意してください。

**適用範囲**

cspybat

**説明**

このオプションを使用して、cspybat をランタイムチェックのメッセージフィルタとして機能させます。このオプションはファイルを読み込み、各メッセージを正しい形式のメッセージに変換します（[C-RUN メッセージ] ウィンドウの場合と同様）。唯一の制約は、コールスタック情報が提供されないことです。

このオプションは、IDE では使用できません。

## --rtc_rules

**構文**

```
--rtc_rules file
```

このオプションはコマンドライン上で --backend オプションより前に配置しなければならない点に注意してください。

**パラメータ**

*file* ルールの入力ファイル。

**適用範囲**

cspybat

**説明**

このオプションを使用して、cspybat に対して C-RUN ルールのファイル名を指定します。

**関連項目**

**ファイルに保存**に関しては 348 ページの *[C-RUN メッセージルール] ウィンドウ*。

このオプションは、IDE では使用できません。

# パート 3. 高度なデバッグ

*ARM 用 C-SPY® デバッガガイド*のこのパートは、以下の章で構成されています。

- マルチコアデバッグ
- 割込み
- C-SPY マクロ
- C-SPY コマンドラインユーティリティ — cspybat
- フラッシュローディング機構

# マルチコアデバッグ

- マルチコアデバッグの概要
- マルチコアのデバッグ
- マルチコアデバッグに関するリファレンス情報

## マルチコアデバッグの概要

以下のトピックについて説明します:

- マルチコアデバッグの概要
- 対称マルチコアデバッグ
- 非対称マルチコアデバッグ
- マルチコアデバッグの要件および制限

### マルチコアデバッグの概要

マルチコアデバッグとは、複数のコアを持つターゲットをデバッグできるという意味です。C-SPY デバッガは、以下の 2 つの方法でマルチコアデバッグをサポートしています。

- *対称なマルチコアデバッグ* (SMP)。2 つ以上の同じコアをデバッグします。これは、IAR Embedded Workbench IDE の 1 つのインスタンスを使用して処理されます。
- *非対称なマルチコアデバッグ* (AMP)。異なるアーキテクチャに基づいた 2 つのコアをデバッグします。たとえば、Cortex–A9 と Cortex–M0 のように、2 つの異なる ARM コアとすることができます。これは、協調する IAR Embedded Workbench IDE の 2 つのインスタンスを使用して処理されます。

### 対称マルチコアデバッグ

対称マルチコアデバッグとは、ターゲットのボード上に 2 つ以上の同じコア（一般的には同じチップ上）があり、通常 1 つのデバッグプローブを通じてそれらにアクセス可能であることを意味します。

デバッガではいつでも、フォーカスが合っているコア 1 つについてのみ、その状態がウィンドウに表示されます。

対称マルチコアデバッグの特別なサポートの概要は以下のとおりです。

- アプリケーション全体を自動的に起動および停止するか、コアを独立して個々に実行するかを制御することができます。
- デバッガでフォーカスするコアを設定することも可能です。これはエディタウィンドウ、［逆アセンブリ］、［登録］、［ウォッチ］、［ローカル］、［コールスタック］の各ウィンドウに影響します。
- ［コア］ウィンドウには使用可能なすべてのコアの一覧が表示され、実行の状態など各コアの情報が示されます。**［コア］**ツールバーは、［コア］ウィンドウを補完するものです。
- ［スタック］ウィンドウには、専用のスタックセクションを通じて各コアのスタックを表示することができます。
- RTOS サポートは、個別のマルチコアプラグインで使用できます。通常は対応するシングルコアのプラグインのように機能しますが、別々のコアで複数のアクティブなタスクを処理します。プラグインでは、［スタック］ウィンドウで選択したタスクのスタックを表示するために必要な情報も提供されます。

## 非対称マルチコアデバッグ

非対称マルチコアデバッグは、ターゲットに異なるアーキテクチャに基づいた 2 つのコアがあることになります。2 つの IDE インスタンスが使用され、それぞれが 1 つのコアに接続されます。この 2 つの IDE インスタンスは同期しており、デバッグセッションが開始および停止して、どちらのインスタンスからもコアが制御できるようになっています。共有メモリを除いて、各デバッグセッションではそれ自体のコアに関する情報（変数、コールスタックなど）しか表示されません。

1 つの IDE インスタンスを手動で開始すると、そのインスタンスはマスターとして参照されます。非対称マルチコアデバッグセッションを開始するとき、マスターによって 2 番目のインスタンスである*スレーブ*が開始されます。スレーブインスタンスがすでに実行中であれば、それが再利用されます。

マスターとスレーブは、それぞれ独自のプロジェクトを必要とします。各プロジェクトを、正しい派生プロセッサ、リンカ、デバッグオプションにより設定する必要があります。また、マスタープロジェクトは、マルチコアマスターとして機能するように設定するか、マルチコアマスターモードを有効にしてください。

ダウンロードの有効な方式の 1 つは、コアのイメージを 1 つに結合して、マスタープロジェクトで結合されたイメージをダウンロードすることです。このシナリオでは、すべてのダウンロードを抑制するため、実行中のターゲットにアタッチするようにスレーブを設定する必要があります。

もうひとつの方式は、マスターとスレーブを別々のバイナリイメージとしてダウンロードします。この場合、メモリ内で意図しない重複がないように注意しなければなりません。

非対称マルチコアデバッグの特別なサポートの概要は以下のとおりです。

- アプリケーション全体を自動的に起動および停止するか、コアを独立して個々に実行するかを制御することができます。
- IDE の各インスタンスに、接続先のコアに関するデバッグ情報が表示されます。
- ［コア］ウィンドウには使用可能なすべてのコアの一覧が表示され、実行の状態など各コアの情報が示されます。**［コア］**ツールバーは、［コア］ウィンドウを補完するものです。
- ブレークポイントを設定すると 1 つのコアのみに接続され、ブレークポイントがトリガされると、そのコアは停止します。

### マルチコアデバッグの要件および制限

C-SPY シミュレータはマルチコアデバッグをサポートしており、特定の要件や制限はありません。

ハードウェアデバッグシステムでマルチコアデバッグを使用するには、C-SPY ドライバとデバッグプローブの固有な組合せが必要です。

- IAR C-SPY I-jet/JTAGjet ドライバ
- I-jet、I-jet Trace、JTAGjet、JTAGjet-Trace のデバッグプローブ

**注 :** 使用するハードウェアに制限によって、トレースのサポートが制限を受けることがあります。

## マルチコアのデバッグ

以下のタスクについて説明します :

- 対称マルチコアデバッグの設定
- 非対称マルチコアデバッグの設定
- マルチコアデバッグセッションの開始と停止

### 対称マルチコアデバッグの設定

1 **［プロジェクト］**>**［オプション］**>**［デバッガ］**>**［マルチコア］**を選択して、使用するコアの数を指定します。

2 デバッグセッションを開始する準備ができました。

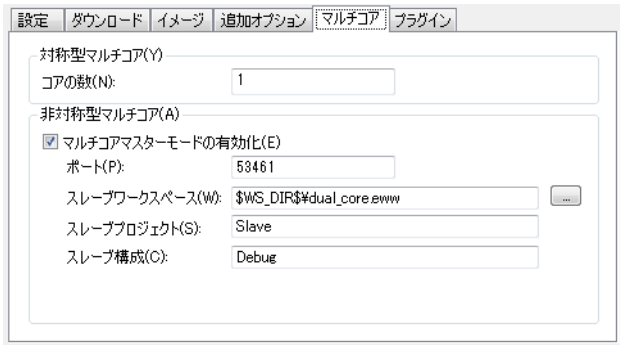
### 非対称マルチコアデバッグの設定

マルチコアデバッグを設定する方法はいくつもありますが、この方式を推奨します。

1 各コアに 1 つずつ、2 つのプロジェクトを持つワークスペースを作成します。

2 マスタープロジェクトを設定するには、次の手順に従います。

- **［プロジェクト］**>**［オプション］**>**［デバッガ］**>**［マルチコア］**を選び、**［マルチコアマスターモードの有効化］**を選択します。スレーブセッションの開始時に使用するワークスペースのパス、プロジェクト名、構成名を指定します。



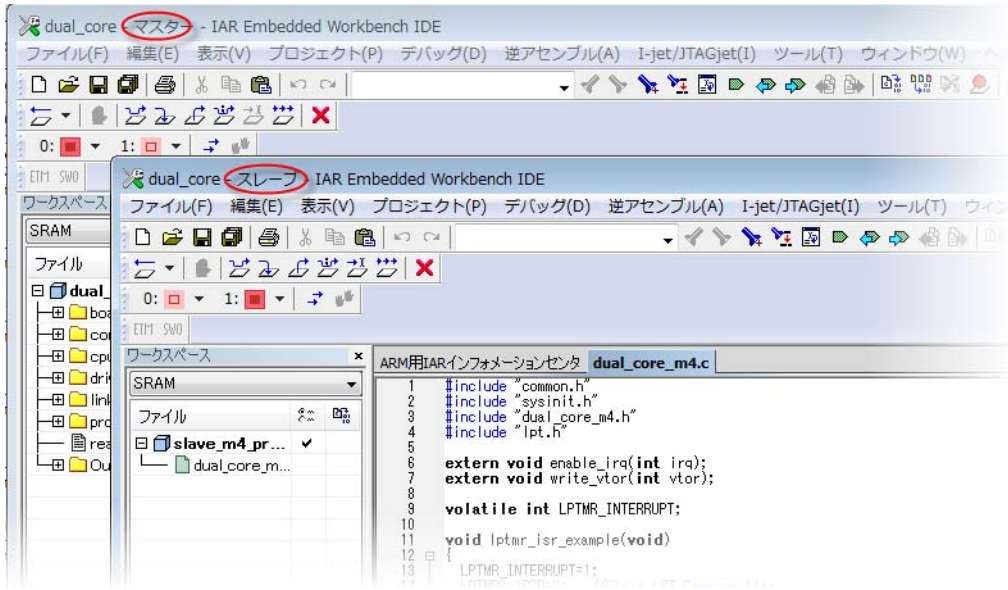

- **［プロジェクト］**>**［オプション］**>*［C-SPY ドライバ］*>**［選択］**を選び、**リセット**方式（通常は**ハードウェア**）を選択します。

3 スレーブプロジェクトを設定するには、次の手順に従います。

- ワークスペースウィンドウでそのプロジェクトを選び、**［プロジェクト］**>**［オプション］**>**［デバッガ］**>**［ダウンロード］**を選択して、**［実行中のターゲットにアタッチ］**を選択します。
- **［プロジェクト］**>**［オプション］**>*［C-SPY ドライバ］*>**［設定］**を選び、マスターセッションに影響しない**リセット**方式を選択します。通常は**ソフトウェア**です。

4 両方のプロジェクトについて、デバッグプローブに互換性のある設定を使用してください。

5 マスターとスレーブのインスタンスは、メインウィンドウのタイトルバーに示されます。




### マルチコアデバッグセッションの開始と停止

1 マルチコアデバッグセッションを開始するには、たとえばマスターまたはスレーブのセッションで、標準の［**ダウンロードしてデバッグ**］コマンドを使用します。

2 マルチコアデバッグセッションを停止するには、たとえば標準の［**デバッグ停止**］コマンドを使用すると、両方のデバッグセッションが停止します。

## マルチコアデバッグに関するリファレンス情報

リファレンス情報：

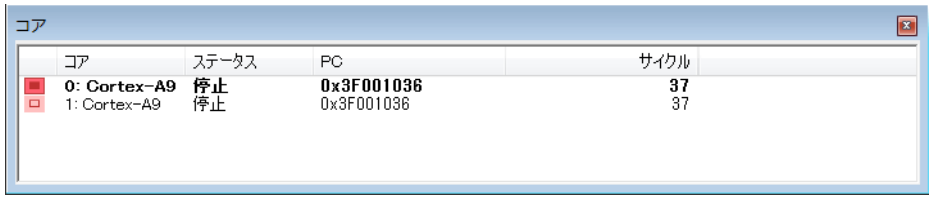
- 366 ページの *［コア］ウィンドウ*
- 367 ページの *コアツールバー*

関連項目：

- 421 ページの *__getSelectedCore*
- 443 ページの *__selectCore*

# ［コア］ウィンドウ

［コア］ウィンドウは［**表示**］メニューから使用できます。




このウィンドウには使用可能なすべてのコアの一覧が表示され、実行の状態など各コアの情報が示されます。太字で強調表示された行は、現在フォーカスされているコアで、コアに固有の情報を表示するウィンドウはすべて、フォーカスされているコアの情報を反映して更新されることを意味します。これには、エディタウィンドウ、［逆アセンブリ］、［登録］、［ウォッチ］、［ローカル］、［コールスタック］の各ウィンドウが含まれます。行をダブルクリックすると、そのコアがフォーカスされます。

**注：**非対称マルチコアデバッグの場合、ローカルのコアのみフォーカスすることができます。

両方のコアが実行中で、どちらかがブレークポイントに達した場合（または、プログラムの実行が停止するような他の状態になったとき）、デバッガはそのコアに自動的にフォーカスしようとします。

## 要件

以下のいずれかが必要です。

- C-SPY シミュレータ
- I-jet、I-jet Trace、JTAGjet、または JTAGjet-Trace のデバッグプローブ

## 表示エリア

このエリアの各行には、以下の列にコアのいずれかに関する情報が表示されます。

***実行の状態***

以下のアイコンのいずれかが表示され、コアの実行状態を示します。


フォーカス中、ただし実行中ではない


フォーカス中ではなく、実行中でもない


フォーカス中、実行中


フォーカス中ではないが、実行中

フォーカス中、スリープモード

フォーカス中ではなく、スリープモード

**コア**

コアの名前。

**ステータス**

実行のステータス。**停止**、**実行中**、**スリープ**のいずれかです。

**PC**

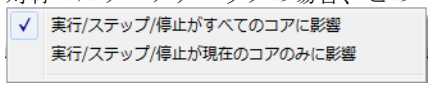
プログラムカウンタの値。

**サイクル | 時間**

サイクルカウンタや実行開始からの実行時間の値。使用するデバッガドライバによって異なります。

### コンテキストメニュー

対称マルチコアデバッグの場合、このコンテキストメニューが使用できます。




以下のコマンドがあります。

**実行 / ステップ / 停止がすべてのコアに影響**

**実行**、**ステップ**、**停止**の各コマンドは、すべてのコアに影響します。

**実行 / ステップ / 停止が現在のコアのみに影響**

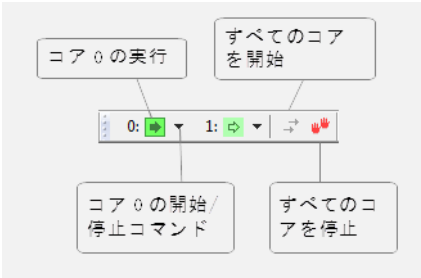
**実行**、**ステップ / 停止**の各コマンドは、現在のコアのみに影響します。このメニューコマンドは、デバイスでサポートされている場合にのみ利用可能です。

**注 :** 以下のコマンドは、一部のターゲットハードウェアではサポートされていません。

## コアツールバー

［**コア**］ツールバーは、マルチコアデバッグを有効にした場合に［**表示**］メニューから使用できます。363 ページの *対称マルチコアデバッグの設定*または 364 ページの *非対称マルチコアデバッグの設定*をそれぞれ参照してくださ

い。




このツールバーは［コア］ウィンドウを補完するもので、同じステータスが表示されます。各コアには、横にドロップダウンメニューのついたボタンがあります。そのコアで C-SPY にフォーカスするには、ボタンをクリックします。

**注：**非対称マルチコアデバッグの場合、ツールバーのコマンドを使用して、関連するデバッグセッションでコアを開始および停止することができます。

# 割込み

- 割込みの概要
- 割込みシステムの使用
- 割込みのリファレンス情報

## 割込みの概要

以下のトピックについて説明します:

- 割込みログの概要
- 割込みシミュレーションシステムの概要について
- 割込み特性
- 割込みシミュレーションの状態
- 割込みシミュレーション用の C-SPY システムマクロ
- ターゲットに合せた割込みシミュレーションシステムの調整

関連項目:

- 410 ページの *C-SPY システムマクロについてのリファレンス情報*
- 133 ページの *ブレークポイント*
- *ARM 用 IAR C/C++ 開発ガイド*

### 割込みログの概要

割込みログを使用すると、割込みイベントに関する包括的な情報を得ることができます。この情報は、たとえば、高速化するためにどの割込みを微調整したらよいか調べるのに便利です。割込みの開始と終了を記録できます。C-SPY シミュレータを使用している場合、トリガ済や期限切れなど、内部の割込みステータス情報も記録できます。ログは［割込みログ］ウィンドウに表示されます。概要は［割込みログサマリ］ウィンドウに表示されます。［タイムライン］ウィンドウの［割込みグラフ］には、アプリケーションプログラム実行中の割込みイベントがグラフィック表示されます。

### 割込みログの要件

割込みログは C-SPY シミュレータでサポートされています。

割込みログを行うには、Cortex-M デバイスが必要です。以下のいずれかも必要となります。

- I-jet または I-jet Trace インサーキットデバッグプローブ、または JTAGjet デバッグプローブ、およびデバッグプローブとターゲットシステム間の SWD インタフェース
- プローブとターゲットシステム間の SWD インタフェースの J-Link または J-Trace デバッグプローブ
- ST-LINK デバッグプローブと、デバッグプローブおよびターゲットシステム間の SWD インタフェース

377 ページの*割込みログを使用するにあたって*も参照してください。

## 割込みシミュレーションシステムの概要について

割込みをシミュレーションすることで、ハードウェアが入手可能になるよりかなり前に、割込みサービスルーチンのロジックをテストして、ターゲットシステムで割込み処理をデバッグできます。擬似割込みと C-SPY のマクロとブレークポイントを連携させることによって、割込み駆動型の周辺デバイスのような複雑なシミュレーションを構築できます。

C-SPY シミュレータには、デバッグ中に割込みの実行をシミュレーションできる割込みシミュレーションシステムが用意されています。割込みシミュレーションシステムをハードウェア割込みシステムと同じように構成することができます。

割込みシステムの特長を以下に示します。

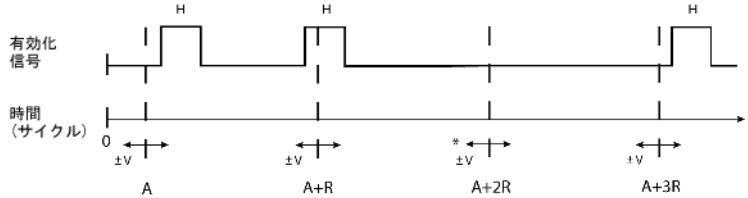
- ARM コアの割込みシミュレーションの提供
- 単発もしくはサイクルカウンタに基づく周期割込み
- さまざまなデバイス用の定義済割込み
- 保持時間、確率、タイミングのばらつきの設定
- タイミングの問題を特定するためのステータス情報
- ダイアログボックスまたは C-SPY システムマクロ、つまり対話型インタフェースと自動インタフェースを 1 つずつ使用した割込みの設定。また、割込みを即時強制することができます
- 定義された割込みごとにイベントを継続的に表示するログウィンドウ
- 現在の割込みアクティビティを示すステータスウィンドウ

**［割込み設定］**ダイアログボックスで定義した割込みはすべて、削除しない限りデバッグセッション終了後も保持されます。一方で強制割込みはサービスを受けるまで存在し、セッション終了後は保持されません。

割込みシミュレーションシステムはデフォルトでは有効になっていますが、必要がない場合は無効にすることでシミュレーションの実行速度を向上させることができます。無効にするには、**[割込み設定]**ダイアログボックスか、システムマクロを使用します。

## 割込み特性

擬似割込みは、ターゲットハードウェアの実際の割込みに似せるために、各割込みを微調整する属性のセットから構成されています。*初回割込み待機時間*、*繰返し間隔*、*保持時間*、*ばらつき*、*確率*のいずれかを指定できます。




* 確率が 100% 未満の場合、一部の割込みが省略されることがあります。

A = 有効化の時間
R = リピート間隔
H = 保留時間
V = 差異

割込みシミュレーションシステムは、サイクルカウンタを時計として使用し、シミュレータで割込みを発生させるタイミングを決定します。*初回割込み待機時間*は、サイクルカウンタ単位で指定します。C-SPY は、サイクルカウンタが指定された初回割込み待機時間を超えると、割込みを生成します。ただし、割込みが生成されるのは命令と命令の間だけです。すなわち、1 つのアセンブラ命令の実行が完了するまでは、その命令に必要なサイクル数に関係なく、割込みの生成は待機させられます。

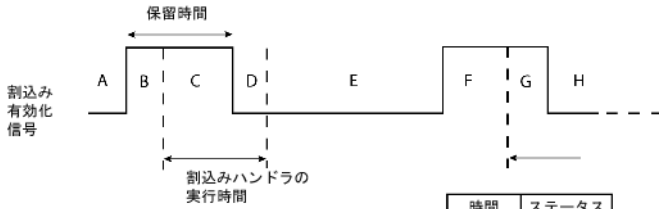
周期的に生成される割込みを定義するには、*繰返し間隔*を指定します。この値は、次の割込みを生成するまでの間隔を表すサイクル数を定義します。繰返し間隔の他に、その間隔が経過した後に実際に割込みを発生させる*確率*（パーセント値）と、繰返し間隔に対する*ばらつき*（パーセント値）の 2 つのオプションによって、実際の発生間隔を制御できます。この 2 つのオプションを使用して、割込みシミュレーションをランダム化できます。その他に、割込みの*保持時間*を指定できます。この時間が経過しても処理されない割込みは削除されます。保持時間を*無限*に設定すると、割込みが確認され削除されるまで対応する保持ビットが設定されます。

## 割込みシミュレーションの状態

割込みシミュレーションシステムには、アプリケーションでのタイミングの問題発見に使用できるステータス情報が含まれています。[割込みステータ

ス］ウィンドウには、使用可能なステータス情報が表示されます。割込みに関して、下記の状態が表示されます。*待機*、*保留*、*実行中*、*停止*。

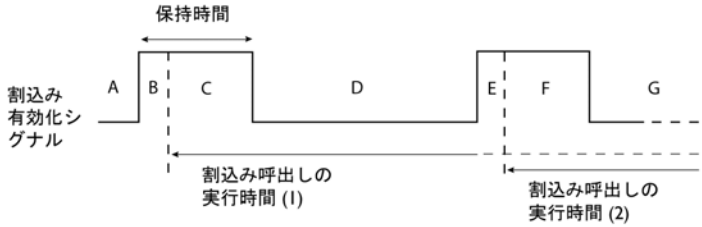
通常は、繰返し割込みの場合は 実行時間よりも長い繰返し間隔が指定されています。この場合、ステータス情報の経時変化は以下のようになります。




| 時間 | ステータス |
|---|---|
| A | アイドル |
| B | 保留 |
| C,D | 実行中 |
| E | アイドル |
| F | 保留 |
| G,H | 実行中 |

**注 :** 割込み有効信号（保持ビットともいう）は、割込みハンドラが割込みを認識するとすぐに、自動的に無効となります。

ただし、割込みの繰返し間隔が実行時間よりも短く、割込みがリエントラント（またはノンマスカブル）の場合、ステータス情報の経時変化は以下のようになります。




| 時間 | ステータス |
|---|---|
| A | 待機 |
| B | 保留 |
| C, D, E | 実行 |
| F,G | 割込み 1 : 停止<br>割込み 2 : 実行 |

実行時間より繰返し間隔が長い 場合は、割込みハンドラを記述し直して速くするか、割込みシミュレーションシステムの繰返し間隔を長く指定した方がよいことがあります。

## 割込みシミュレーション用の C-SPY システムマクロ

マクロによる定義は、要求どおりのブレークポイント設定ができない場合に便利です。擬似割込みの定義を含むマクロ関数を記述することによって、C-SPY の起動時にそのマクロ関数を実行できます。他にも、マクロファイルを使用した場合に擬似割込みの定義がドキュメント化されたり、開発プロジェクトに複数のエンジニアが携わっている場合にグループ内でマクロファイルを共有できたりといった長所があります。

C-SPY シミュレータには、割込みに関連する以下の定義済みシステムマクロが用意されています。

```
__enableInterrupts

__disableInterrupts

__orderInterrupt

__cancelInterrupt

__cancelAllInterrupts

__popSimulatorInterruptExecutingStack
```

最初の 5 つのマクロのパラメータは、**[割込み]** ダイアログボックスの該当するエントリに対応します。

各マクロの詳細については、410 ページの *C-SPY システムマクロについてのリファレンス情報*を参照してください。

## ターゲットに合せた割込みシミュレーションシステムの調整

割込みシミュレーションシステムは使いやすく設計されています。ただし、割込みシミュレーションシステムの力を最大限に利用するには、使用しているプロセッサに適合させる方法に習熟している必要があります。

割込みシミュレーションは、ハードウェアと同じように動作します。これは、割込みの実行はグローバルな割込みイネーブルビットのステータスに依存することを意味します。マスカブル割込みの実行も、個々の割込みイネーブルビットのステータスに依存します。

デバイス固有の割込みをシミュレーションするには、使用できる割込みの詳細な情報を割込みシステムに通知する必要があります。この情報は、デバイス記述ファイルに記述されています。

デバイス記述ファイルの詳細については、55 ページの *デバイス記述ファイルの選択*を参照してください。

# 割込みシステムの使用

以下のタスクについて説明します:

- シンプルな割込みシミュレーション
- マルチタスクシステムでの割込みシミュレーション
- 割込みログを使用するにあたって

関連項目:

- 設定ファイルを使用して C-SPY 起動時にシミュレーションされた割込みを定義する方法については、396 ページの *C-SPY マクロの使用*
- インフォメーションセンタのチュートリアル「*割込みのシミュレーション*」

## シンプルな割込みシミュレーション

次の例は、OKI ML674001 でのタイマ割込みをシミュレーションする方法を示します。ただし、他のタイプの割込みも同じ手順でシミュレーションできます。

### 割込みをシミュレーションしてデバッグするには、次の手順に従います。

1. このシンプルなアプリケーションには、システムタイマ割込みを処理する IRQ ハンドラルーチンが含まれます。tick 変数をインクリメントします。main 関数は必要に応じてステータスレジスタを設定します。アプリケーションは、100 回割込みが生成されると終了します。

```
/* 拡張キーワードの利用を有効にします。*/
#pragma language=extended

#include <intrinsics.h>
#include <oki/ioml674001.h>
#include <stdio.h>

unsigned int ticks = 0;

/* IRQ ハンドラ */
__irq __arm void IRQ_Handler(void)
{
  /* システムタイマ割込みのみを使用するため
     割込みソースをチェックする必要はありません。*/
  ticks += 1;
  TMOVFR_bit.OVF = 1; /* システムタイマのオーバフローフラグをクリアします
*/
}

int main( void )
{
  __enable_interrupt();
  /* タイマ設定コード */
  ILC0_bit.ILR0 = 4;        /* システムタイマ割込みの優先順位 */
  TMRLR_bit.TMRLR = 0x1E5; /* システムタイマの再ロード値 */
  TMEN_bit.TCEN = 1;        /* システムタイマを有効にします */
  while (ticks < 100);
  printf("Done¥n");
}
```

**2** 割込みサービルルーチンをアプリケーションソースコードに追加して、そのファイルをプロジェクトに追加します。

**3** プロジェクトをビルドして、シミュレータを起動します。

**4** **［シミュレータ］>［割込み設定］**を選択して、**［割込み設定］**ダイアログボックスを表示します。**［割込みシミュレーションを有効にする］**オプションを選択して、割込みシミュレーションを有効にします。**［新規作成］**をクリックして**［割込みの編集］**ダイアログボックスを開きます。Timer の例については、以下の設定を確認してください。

| オプション | 設定 |
|---|---|
| 割込み | IRQ |
| 初回割込み | 4000 |
| 繰返し間隔 | 2000 |

*表 15: タイマ割込み設定*

| オプション | 設定 |
|---|---|
| 保持時間 | 10 |
| 確率 (%) | 100 |
| ばらつき (%) | 0 |

*表 15: タイマ割込み設定（続き）*

**[OK]** をクリックします。

5 アプリケーションを実行します。アプリケーションソースコードで正常に割込みが有効になっている場合、C-SPY は以下を実行します。

- サイクルカウンタが 4000 を超えると割込みを生成
- その後は約 2000 サイクルごとに周期割込みを生成

6 動作中の割込みをモニタするには、**［シミュレータ］>［割込みログ］**を選択して［割込みログ］ウィンドウを開きます。

7 ［割込みログ］ウィンドウのコンテキストメニューから、**［有効化］**を選択してロギングを有効にします。プログラムの実行を再開する場合は、割込みの入口と出口のステータス情報が［割込みログ］ウィンドウに表示されます。

時間軸に沿って割込みをグラフィック表示する方法については、244 ページの*［タイムライン］ウィンドウ*を参照してください。

## マルチタスクシステムでの割込みシミュレーション

割込みハンドラから復帰する際に通常の命令以外の方法で割込みを使用する場合、たとえばタスク切替えが行われるオペレーティングシステムの場合、シミュレータは割込みの実行が完了したことを自動的には検出できません。割込みシミュレーションシステムは正常に動作しますが、**［割込み設定］**ダイアログボックスのステータス情報は予想したようには表示されない可能性があります。同時実行される割込みの数が多すぎると、ワーニングが出力される場合があります。

### 通常の割込み終了をシミュレーションするには、次の手順に従います。

1 割込み関数から復帰する命令にコードブレークポイントを設定します。

2 `__popSimulatorInterruptExecutingStack` マクロを条件としてブレークポイントに指定します。

ブレークポイントがトリガされると、マクロが実行され、それが完了するとアプリケーションが自動的に実行を継続します。

### 割込みログを使用するにあたって

1 割込みログを設定するには、*[C-SPY ドライバ]* > **[SWO 設定]** を選択します。ダイアログボックスで、トレースデータの SWO 通信チャンネルを設定します。特に **[CPU クロック]** オプションに注意してください。CPU クロックは、**[プロジェクト]** > **[オプション]** > **[ST-LINK]** ページでも設定できます。

C-SPY シミュレータの場合、特別な設定は必要ありません。

2 *[C-SPY ドライバ]* > **[割込みログ]** を選択して、[割込みログ] ウィンドウを開きます。または、以下のように選択することもできます。

- *[C-SPY ドライバ]* > **[割込みログ概要]** を選択して、[割込みログ概要] ウィンドウを開く。
- *[C-SPY ドライバ]* > **[タイムライン]** を選択して [タイムライン] ウィンドウを開き、割込みグラフを表示する。

3 [割込みログ] ウィンドウのコンテキストメニューから、**[有効化]** を選択してロギングを有効にします。

**[SWO 設定]** ダイアログボックスの **[ログイベントの割込み]** エリアで、割込みログが有効になっていることが確認できます。

4 アプリケーションプログラムの実行を開始して、ログ情報を収集します。

5 割込みログ情報を表示するには、[タイムライン] ウィンドウで割込みログ、割込みログ概要、または割込みグラフを参照します。

6 ログまたは概要をファイルに保存する場合は、対象のウィンドウのコンテキストメニューから **[ログファイルを保存]** を選択します。

7 割込みログを無効にするには、[割込みログ] ウィンドウのコンテキストメニューから **[有効化]** をオフにします。

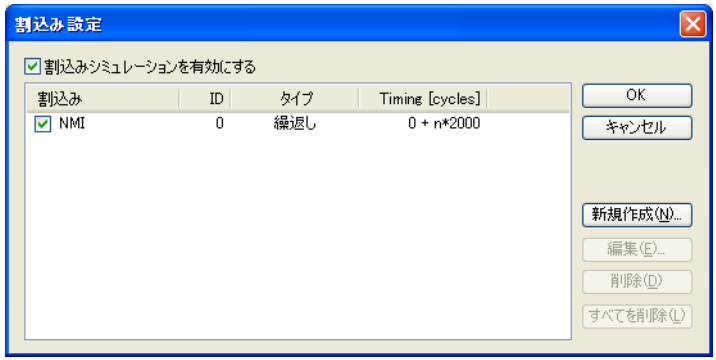
## 割込みのリファレンス情報

リファレンス情報 :

- 378 ページの *[割込み設定] ダイアログボックス*
- 380 ページの *[割込みの編集] ダイアログボックス*
- 382 ページの *[強制割込み] ウィンドウ*
- 383 ページの *[割込みステータス] ウィンドウ*
- 385 ページの *[割込みログ] ウィンドウ*
- 389 ページの *[割込みログ概要] ウィンドウ*

## ［割込み設定］ダイアログボックス

［割込み設定］ダイアログボックスは、［シミュレータ］>［割込み設定］を選択することにより使用できます。




このダイアログボックスには、定義済みの割込みがすべて一覧表示されます。このダイアログボックスを使用して、割込みシミュレーションシステムおよび個々の割込みを有効 / 無効にします。

### 要件

C-SPY シミュレータ。

### 割込みシミュレーションの有効化

割込みシミュレーションを有効または無効にします。割込みシミュレーションが無効の場合、定義はそのまま残りますが、割込みは発生しません。インストール済みの割込みも、リストの割込み名の左にあるチェックボックスを使用して、個別に有効か無効にできます。

### 表示エリア

このエリアには以下の列が含まれます。

**割込み**

すべての割込みが表示されます。チェックボックスを使用して、割込みを有効 / 無効にします。

**ID**

一意の割込み識別子。

**タイプ**

割込みタイプが表示されます。タイプは以下のいずれかです。

強制。単発の割込みで、［強制割込み］ウィンドウで定義します。

単一。単発の割込み。

繰返し。定期的に発生する割込みです。

割込みが C-SPY マクロから設定された場合、追加部分（マクロ）が追加されます。たとえば、`Repeat(macro)` というようになります。

**タイミング**

割込みのタイミング。単一と強制割込みの場合、割込み待機時間が表示されます。繰返し割込みの場合、情報は［割込み待機時間］+［n* 繰返し回数］の形式になります。たとえば、`2000 + n*2345` というようになります。これは、この割込みが初めてトリガされるときが 2000 サイクルで、その後は 2345 サイクル間隔になることを示します。

## ボタン

以下のボタンを選択できます。

**新規作成**

**［割込みの編集］**ダイアログボックスが開きます（380 ページの *［割込みの編集］ダイアログボックス*を参照）。

**編集**

**［割込みの編集］**ダイアログボックスが開きます（380 ページの *［割込みの編集］ダイアログボックス*を参照）。

**削除**

選択した割込みを削除します。

**全て削除**

すべての割込みを削除します。

## ［割込みの編集］ダイアログボックス

［**割込みの編集**］ダイアログボックスは、［**割込みの設定**］ダイアログボックスから使用できます。


このダイアログボックスを使用して、割込みパラメータを対話形式で微調整します。パラメータを追加して、すぐに要求どおり割込みが発生するかどうかをテストできます。

**注：**強制割込み以外の割込みのみ編集や削除ができます。

### 要件

C-SPY シミュレータ。

### 割込み

編集する割込みを選択します。ドロップダウンリストに使用可能なすべての割込みが表示されます。ここで割込みを選択すると、自動的に［**説明**］ボックスが更新されます。Cortex-M デバイスの場合、リストには選択したデバイス記述ファイルに記述されているエントリが表示されます。他のデバイスの場合、IRQ と FIQ の 2 つの割込みだけが使用できます。

### 説明

選択された割込みの内容。記述は選択されたデバイス記述ファイルから読み込まれ、これは優先順位やベクタオフセット、イネーブルビット、保持ビットがスペース文字で区切られて記述された文字列から構成されます。イネーブルビットと保持ビットはオプションです。何も使用しないか、イネーブルビットのみ、または両方を使用することができます。システムマクロ `__orderInterrupt` を使用して指定された割込みの場合は、［**説明**］ボックスには何も表示されません。

Cortex-M デバイスの場合、記述は選択したデバイス記述ファイルから読み込まれ、これは編集可能です。イネーブルビットと保持ビットは ddf ファイルからは利用できません。必要な場合は手動で編集する必要があります。優先

順位はハードウェアの場合と同じで、数値が小さいほど優先順位は高くなります。NMI と HardFault は特殊なため、記述を編集しないでください。Cortex-M の割込みは、レジスタ PRIMASK、FAULTMASK、BASEPRI の影響も受けます。詳細は ARM のマニュアルを参照してください。

他のデバイスについては、IRQ および FIQ の記述文字列がハードコード化されており、編集はできません。これらの記述では、優先順位の番号が大きいほど優先順位が高くなります。

**初回割込み**

指定されたタイプの割込みを生成するまでの待機時間を表すサイクルカウンタ値を指定します。

**繰返し間隔**

割込みの繰返し間隔をサイクル単位で指定します。

**ばらつき (%)**

タイミングのばらつきを繰返し間隔に対するパーセント値として選択します。このばらつきの範囲内に割込みが発生する可能性があります。たとえば、繰返し間隔が 100、ばらつきが 5% の場合、割込みは T=95 〜 105 の間に発生する可能性があり、これによってタイミングのばらつきがシミュレーションされます。

**保持時間**

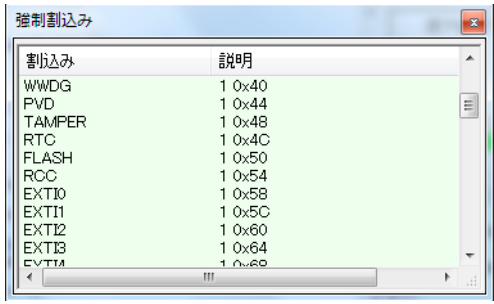
割込みの保持時間がサイクル単位で指定します。この時間が経過しても処理されない割込みは削除されます。［**無限**］を選択すると、対応する保持ビットは割込みが確認、削除されるまで設定されます。

**確率 (%)**

割込みが指定された期間内に実際に発生する確率をパーセント単位で選択します。

## ［強制割込み］ウィンドウ

［強制割込み］ウィンドウは、C-SPY ドライバメニューから使用できます。




このウィンドウを使用して、割込みを即時に強制します。割込みロジックや割込みルーチンをチェックする場合に便利です。割込み名を入力して、その名前に該当する最初の行に注目してください。

強制割込みの保持時間は無制限です。割込みはサービスを受けるまで、またはデバッグセッションをリセットするまで存在します。

ウィンドウの内容をソートするには、［割込み］または［説明］の列の見出しをクリックします。同じ列のヘッダを 2 回クリックすると、ソートの順序が逆になります。

**割込みを強制するには、次の手順に従います。**

1. 割込みシミュレーションシステムを有効にするには、378 ページの*［割込み設定］ダイアログボックス*を参照してください。
2. ［強制割込み］ウィンドウで割込みをダブルクリックするか、コンテキストメニューの**［強制］**コマンドを使用して有効にします。

### 要件

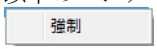
C-SPY シミュレータ。

### 表示エリア

このエリアには使用可能なすべての割込みとその定義がリストされます。この情報は、選択したデバイス記述ファイルから取得されます。詳しい記述はこのファイルを参照してください。

### コンテキストメニュー

以下のコンテキストメニューがあります。



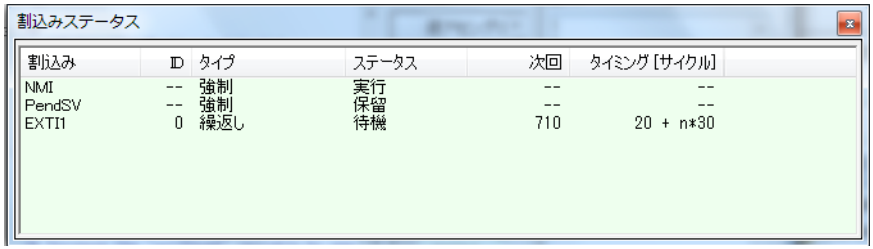

次のコマンドを使用できます。

**強制**

表示エリアで選択した割込みをトリガします。

## ［割込みステータス］ウィンドウ

［割込みステータス］ウィンドウは、C-SPY ドライバメニューから使用できます。




このウィンドウには、現在アクティブなすべての割込み、つまり実行中または実行を待つ割込みのステータスが表示されます。

### 要件

C-SPY シミュレータ。

### 表示エリア

このエリアには以下の列が含まれます。

**割込み**

すべての割込みが表示されます。

**ID**

一意の割込み識別子。

***タイプ***

割込みのタイプ。タイプは以下のいずれかです。

強制。単発の割込みで、［強制割込み］ウィンドウで定義します。

単一。単発の割込み。

繰返し。定期的に発生する割込みです。

割込みが C-SPY マクロから設定された場合、追加部分（マクロ）が追加されます。たとえば、Repeat(macro) というようになります。

**ステータス**

割込みの状態

**待機**。低い割込み有効信号です（無効）。

**保留**。割込み有効信号はアクティブですが、割込みが割込みハンドラにまだ認識されていません。

**実行**。割込みは現在サービス中です。すなわち、割込みハンドラ機能が実行中です。

**停止**。優先度の高い割込みが実行中のため、割込みは現在停止中です。現在アクティブな割込みを削除した場合、「実行」と「停止」に。

**（削除済）**が追加されます。**（削除済）**は、割込みの実行が完了すると除去されます。

**次回**

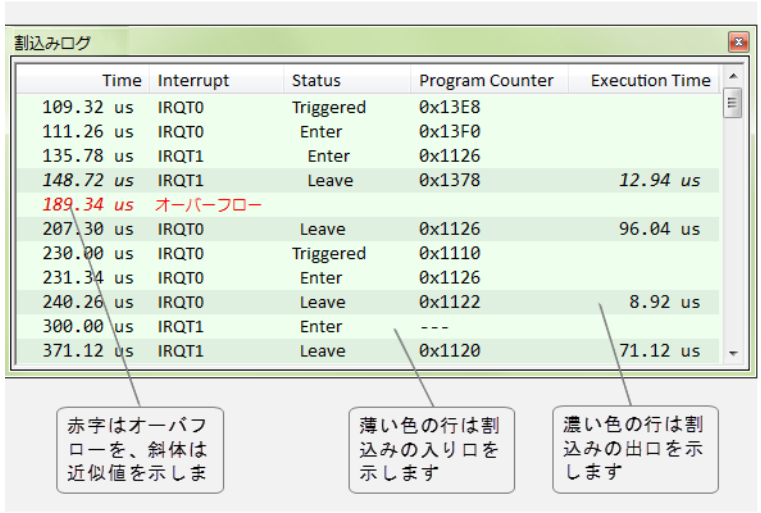
次に待機中の割込みがトリガされるとき。繰返し可能な割込みが実行を開始すると、その割込みのコピーが表示されて状態が「待機」になり「次回」が設定されます。次回が設定されていない割込み（保留や実行、停止）の場合、この列には -- と表示されます。

**タイミング**

割込みのタイミング。単一と強制割込みの場合、割込み待機時間が表示されます。繰返し割込みの場合、情報は［割込み待機時間］+［n* 繰返し回数］の形式になります。たとえば、2000 + n*2345 というようになります。これは、この割込みが初めてトリガされるときが 2000 サイクルで、その後は 2345 サイクル間隔になることを示します。

## ［割込みログ］ウィンドウ

［割込みログ］ウィンドウは、C-SPY ドライバメニューから使用できます。




このウィンドウには割込みの開始と終了が記録されます。C-SPY シミュレータには、内部の状態の変化も記録されます。

この情報は、ターゲットシステムの割込み処理をデバッグする場合に役に立ちます。［割込みログ］ウィンドウが開いている間は、実行時にステータスが常時更新されます。

**注 :** 保存されるログの数には制限があります。この制限を超過すると、バッファの最初のエントリが消去されます。

詳細については、377 ページの*割込みログを使用するにあたって*を参照してください。

アプリケーションの実行中に割込みイベントをグラフィック表示する方法については、244 ページの*［タイムライン］ウィンドウ*を参照してください。

### 要件

以下のいずれかが必要です。

- C-SPY シミュレータ
- I-jet または I-jet Trace インサーキットデバッグプローブ、または JTAGjet デバッグプローブ、およびデバッグプローブとターゲットシステム間の SWD インタフェース

- デバッグプローブとターゲットシステム間の SWD インタフェースの J-Link または J-Trace デバッグプローブ
- ST-LINK デバッグプローブと、デバッグプローブおよびターゲットシステム間の SWD インタフェース

### C-SPY ハードウェアデバッガドライブの表示エリア

このエリアには以下の列が含まれます。

**時間**

［**SWO 設定**］ダイアログボックスで指定したクロック周波数に基づく、割込み開始の時間。

ターゲットシステムが正確な時間を収集できなかった場合は、おおよその時間が斜体で表示されます。

この列は、コンテキストメニューから［**時間表示**］を選択した場合に有効になります。［**時間表示**］コマンドが使用可能でない場合、［**時間**］列がデフォルトで表示されます。

**サイクル**

実行の開始からイベントまでのサイクルの数。

斜体表示のサイクルカウントは、おおよその値を示します。ターゲットシステムが正確な値を収集できなかった場合は、おおよその値が斜体で表示されます。

この列は、コンテキストメニューから［**サイクル表示**］を選択したときに使用可能になります。ただし、使用する C-SPY ドライバでサポートされている場合に限ります。

**割込み**

割込みが発生する割込みソース名。赤色で `Overflow` と表示されている場合、通信チャンネルがすべての割込みログをターゲットシステムから送信できなかったことを示します。

**ステータス**

割込みのイベントステータス：

**開始**。割込みが現在実行中です。

**終了**。割込みの実行が完了しました。

**プログラムカウンタ ***

割込みハンドラのアドレス。

**実行時間 / サイクル**

割込みの経過時間。開始と終了のタイムスタンプを使用して算出されます。特定の割込みで発生した他の割込みやサブルーチンの時間も含まれます。

* アドレスをダブルクリックできます。ソースコードで使用可能な場合、エディタウィンドウに、たとえば、割込みハンドラなど、対応するソースコードが表示されます（ライブラリソースコードは除く）。

### C-SPY シミュレータの表示エリア

このエリアには以下の列が含まれます。

**時間**

内部で指定したクロック周波数に基づいた、割込み入口の時間。

この列は、コンテキストメニューから［**時間表示**］を選択した場合に有効になります。

**サイクル**

実行の開始からイベントまでのサイクルの数。

この列は、コンテキストメニューから［**サイクル表示**］を選択した場合に有効になります。

**割込み**

デバイス記述ファイルで定義された割込み。

**ステータス**

割込みイベントステータスが表示されます :

**トリガ済み**。待機時間がすでに経過している割込み。

**強制**。トリガ済みと同じですが、割込みは［強制割込み］ウィンドウから強制されています。

**開始**。割込みが現在実行中です。

**終了**。割込みが実行済みです。

**期限切れ**。割込みが実行されることなく保持時間が経過しました。

**却下**。割込みを受け入れるのに必要な割込みレジスタが設定されていないため、割込みが却下されました。

**プログラムカウンタ**

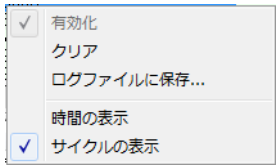
イベントが発生したときのプログラムカウンタの値。

**実行時間 / サイクル**

割込みの経過時間。開始と終了のタイムスタンプを使用して算出されます。特定の割込みで発生した他の割込みやサブルーチンの時間も含まれます。

## コンテキストメニュー

このコンテキストメニューは、［データログ］ウィンドウ、［データログサマリ］ウィンドウ、［割込みログ］ウィンドウ、［割込みログ概要］ウィンドウで使用できます。




**注 :** 各コマンドはどのウィンドウにも表示されますが、特定のウィンドウでのみ機能します。

以下のコマンドがあります。

**有効化**

ロギングシステムを有効にします。ロギングシステムでは、ウィンドウを閉じるときにも情報が記録されます。

**クリア**

ログ情報を削除します。デバッガをリセットしたときにも同じことになります。

**ログファイルを保存**

ログ情報の保存先のファイルを指定する、標準のファイル選択用ダイアログボックスを表示します。ログファイルのエントリは、タブおよび LF（ラインフィード）で区切ります。［近似］列の値 x は、タイムスタンプが概算値であることを示します。

**時間表示**

［データログ］ウィンドウおよび［割込みログ］ウィンドウに**［時間］**列を表示します。

このメニューコマンドは、使用する C-SPY ドライバでは利用できないことがあります。つまり、［データログ］ウィンドウに**［時間］**列がデフォルトで使用されている場合です。

**サイクル表示**

［データログ］ウィンドウおよび［割込みログ］ウィンドウに**［サイクル］**列を表示します。

このメニューコマンドは、使用する C-SPY ドライバで利用できないことがあります。つまり、［**サイクル**］列がサポートされていない場合です。

## ［割込みログ概要］ウィンドウ

［割込みログ概要］ウィンドウは、C-SPY ドライバメニューから使用できます。


このウィンドウには、記録された割込みの入口と出口の概要が表示されます。

詳細については、377 ページの*割込みログを使用するにあたって*を参照してください。

アプリケーションの実行中に割込みイベントをグラフィック表示する方法については、244 ページの*［タイムライン］ウィンドウ*を参照してください。

### 要件

以下のいずれかが必要です。

- C-SPY シミュレータ。
- I-jet または I-jet Trace インサーキットデバッグプローブ、または JTAGjet デバッグプローブ、およびデバッグプローブとターゲットシステム間の SWD インタフェース。
- デバッグプローブとターゲットシステム間の SWD インタフェースの J-Link または J-Trace デバッグプローブ。
- ST-LINK デバッグプローブと、デバッグプローブおよびターゲットシステム間の SWD インタフェース。

### C-SPY シミュレータの表示エリア

このエリアの各行の以下の列には、ログ情報に基づいて特定の割込みに関する統計が表示されます。

**割込み**

発生した割込みのタイプ。

* 列の下には、現在の時間やサイクル（実行開始からのサイクル数や実行時間）が表示されます。オーバフローカウントとおおよその時間は常にゼロです。

**カウント**

割込みが発生した回数。

**初回**

割込みの初回の実行時間。

**合計（時間）****

割込みで経過した累計時間。

**合計 (%)**

現在の時間に占めるパーセンテージ。

**最速 ****

このタイプの割込み 1 回の最短実行時間。

**最長 ****

このタイプの割込み 1 回の最長実行時間。

**最低間隔**

このタイプの 2 つの割り込み間の最短時間。

間隔は 2 つの連続する割込み間のエントリ時間として指定します。

**最長間隔**

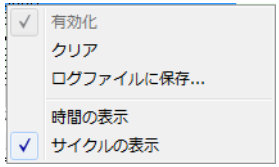
このタイプの 2 つの割込みの最長間隔。

この間隔は、2 つの連続した割込みの開始時間の間隔です。

**［割込みログ］ウィンドウの［実行時間 / サイクル］と同じ方法で計算されます。

## コンテキストメニュー

このコンテキストメニューは、［データログ］ウィンドウ、［データログサマリ］ウィンドウ、［割込みログ］ウィンドウ、［割込みログ概要］ウィンドウで使用できます。




**注 :** 各コマンドはどのウィンドウにも表示されますが、特定のウィンドウでのみ機能します。

以下のコマンドがあります。

**有効化**

ロギングシステムを有効にします。ロギングシステムでは、ウィンドウを閉じるときにも情報が記録されます。

**クリア**

ログ情報を削除します。デバッガをリセットしたときにも同じことになります。

**ログファイルを保存**

ログ情報の保存先のファイルを指定する、標準のファイル選択用ダイアログボックスを表示します。ログファイルのエントリは、タブおよびLF（ラインフィード）で区切ります。［近似］列の値xは、タイムスタンプが概算値であることを示します。

**時間表示**

［データログ］ウィンドウおよび［割込みログ］ウィンドウに**［時間］**列を表示します。

このメニューコマンドは、使用するC-SPYドライバでは利用できないことがあります。つまり、［データログ］ウィンドウに**［時間］**列がデフォルトで使用されている場合です。

**サイクル表示**

［データログ］ウィンドウおよび［割込みログ］ウィンドウに**［サイクル］**列を表示します。

このメニューコマンドは、使用するC-SPYドライバで利用できないことがあります。つまり、**［サイクル］**列がサポートされていない場合です。

# C-SPY マクロ

- C-SPY マクロの概要
- C-SPY マクロの使用
- マクロ言語についてのリファレンス情報
- 予約済みのセットアップマクロ関数名についてのリファレンス情報
- C-SPY システムマクロについてのリファレンス情報
- マクロのグラフィカル環境

## C-SPY マクロの概要

以下のトピックについて説明します:

- C-SPY マクロを使用する理由
- C-SPY マクロの使用の概要
- セットアップマクロ関数およびファイルの概要
- マクロ言語の概要

### C-SPY マクロを使用する理由

C-SPY マクロは、単独で使用することもできますが、複雑なブレークポイントおよび割込みシミュレーションとともに使用することにより、さまざまなタスクを実行できます。マクロが役に立つ例をいくつか示します。

- トレース出力、変数値の出力、ブレークポイントの設定などによるデバッグセッションの自動化。
- ハードウェアレジスタの初期化などのハードウェア設定。
- 実行中のアプリケーションへのシミュレーションしたデータの入力。
- 周辺デバイスのシミュレーション。「*割込み*」を参照してください。シミュレータドライバを使用している必要があります。
- たとえば、スタックの深さを計算する関数など、簡単なデバッグユーティリティ関数の開発。`¥arm¥src¥` ディレクトリの例、`stack.mac` を参照してください。

## C-SPY マクロの使用の概要

C-SPY マクロを使用するには、以下のことを行う必要があります。

- マクロ変数と関数を記述して、1 つまたは複数の*マクロファイル*に収集
- マクロを登録
- マクロを実行

マクロの登録と実行については、いくつかの方法から選択できます。どの方法を選択するかは、操作や自動化のレベル、どの段階でマクロを登録および実行するかによって異なります。

## セットアップマクロ関数およびファイルの概要

予約済の*セットアップマクロ関数名*がいくつかあります。これらは以下のような特定のタイミングに呼び出されるマクロ関数を定義する際に使用できます。

- ターゲットシステムとの通信確立後、アプリケーションソフトウェアをダウンロードするまでの間
- アプリケーションソフトウェアのダウンロードが完了した直後
- リセットコマンドが発行されるたび
- デバッグセッションの終了直後

マクロ関数を呼び出すタイミングを定義するには、予約済の名前でマクロ関数を定義、登録する必要があります。たとえば、アプリケーションソフトウェアをロードする前に特定のメモリエリアをクリアする必要がある場合は、マクロセットアップ関数 `execUserPreload` が適しています。アプリケーションソフトウェアをロードする前に一部の CPU レジスタやメモリにマッピングされた周辺ユニットを初期化する必要がある場合にも、この関数は適しています。

これらの関数を*セットアップマクロファイル*に定義します。これは C-SPY の起動前にロードできます。この場合、マクロ関数は C-SPY を起動するたびに自動的に登録されます。これは、C-SPY の初期化を自動化する場合や、複数のセットアップマクロファイルを登録する必要がある場合にも使用できます。

各セットアップマクロ関数の詳細については、406 ページの*予約済みのセットアップマクロ関数名についてのリファレンス情報*を参照してください。

## メモリの再配置

ARM を基本とする多くのプロセッサに共通する機能は、メモリの再配置機能です。メモリコントローラでは、リセット後に、フラッシュのような不揮発性メモリにアドレスのゼロをマッピングするのが一般的です。メモリコントローラを構成することで、RAM をアドレスマップのゼロに配置し、不揮発性

メモリをアドレスマップの上位に配置するように、システムメモリを再配置することができます。再配置することで、例外テーブルは RAM に置かれ、ターゲットハードウェアにコードをダウンロードしたときに簡単に修正できます。C-SPY でこれを処理するには、セットアップマクロの `execUserPreload()` 関数が適しています。例については、60 ページの*メモリの再配置*を参照してください。

## マクロ言語の概要

マクロ言語の構文は C 言語に非常によく似ています。以下の共通点があります。

- *マクロ文*があります。
- *マクロ関数*を、パラメータとリターン値の有無を指定して定義できます。
- C ライブラリ関数に似た定義済み組込み*システムマクロ*。ファイルのオープンやクローズ、ブレークポイントの設定、割込みシミュレーションの定義など便利なタスクを実行できます。
- *マクロ変数*。グローバルかローカルのどちらかで、C-SPY 式で使用できます。
- *マクロ文字列*。定義済システムマクロを使用して操作することができます。

マクロ言語のコンポーネントの詳細については、401 ページの*マクロ言語についてのリファレンス情報*を参照してください。

## 例

以下に示すマクロ関数の例では、マクロ言語のさまざまなコンポーネントが示されています。

```
__var oldVal;
CheckLatest(val)
{
 if (oldVal != val)
 {
  __message "Message: Changed from ", oldVal, " to ", val, "¥n";
  oldVal = val;
 }
}
```

**注 :** マクロの予約語は、名前の衝突を避けるために、2 連のアンダースコアで始まります。

# C-SPY マクロの使用

以下のタスクについて説明します:

- C-SPY マクロの登録 — 概要
- C-SPY マクロの実行 — 概要
- セットアップマクロとセットアップファイルによる登録と実行
- ［クイックウォッチ］によるマクロの実行
- ブレークポイントにマクロを接続して実行
- C-SPY マクロの中止

その他の C-SPY マクロの使用例については、以下を参照してください。

- インフォメーションセンタのチュートリアル「割込みのシミュレーション」
- 59 ページの *C-SPY の起動前にターゲットハードウェアを初期化する*

## C-SPY マクロの登録 — 概要

次に、定義したマクロ関数を使用することを C-SPY に通知する必要があるため、マクロを*登録*する必要があります。マクロ関数の登録方法はいろいろあります。

- C-SPY の起動シーケンス中にマクロ関数を登録できます（397 ページの *セットアップマクロとセットアップファイルによる登録と実行*を参照）。
- ［マクロ登録］ウィンドウで対話的にマクロを登録できます（463 ページの *マクロ登録ウィンドウ*を参照）。登録されたマクロは［デバッグマクロ］ウィンドウに表示されます（465 ページの *［デバッグマクロ］ウィンドウ*を参照）。
- システムマクロ `__registerMacroFile` を使用すると、マクロ関数定義を含むファイルを登録できます。これは、実行時の条件に応じて、登録するマクロファイルを動的に選択できることを意味します。さらに、システムマクロを使用する場合は、同時に複数のファイルを登録できます。システムマクロの詳細については、442 ページの *__registerMacroFile* を参照してください。

どの方法を選択するかは、操作や自動化のレベル、どの段階でマクロを登録するかによって異なります。

## C-SPY マクロの実行 — 概要

マクロ関数の実行方法はいろいろあります。

- セットアップマクロファイルでセットアップマクロ関数を定義することにより、C-SPY の起動シーケンス中およびデバッグセッションの他の定義済

の段階で マクロ関数を実行できます（397 ページの *セットアップマクロとセットアップファイルによる登録と実行*を参照）。

- ［クイックウォッチ］ウィンドウでは式を評価できるため、それによってマクロ関数を実行できます。例については、398 ページの *［クイックウォッチ］によるマクロの実行*を参照してください。
- ［マクロクイック起動］ウィンドウは［クイックウォッチ］ウィンドウに似ていますが、より C-SPY マクロ向けに設計されています。466 ページの *［マクロクイック起動ウィンドウ］* を参照してください。
- マクロはブレークポイントに接続でき、ブレークポイントがトリガされると、マクロが実行されます。例については、399 ページの *ブレークポイントにマクロを接続して実行*を参照してください。

どの方法を選択するかは、操作や自動化のレベル、どの段階でマクロを実行するかによって異なります。

### セットアップマクロとセットアップファイルによる登録と実行

C-SPY 起動シーケンスの途中でマクロファイルを登録すると便利なときがあります。そのためには、デバッグセッションを開始する前にロードするマクロファイルを指定します。この場合、マクロ関数はデバッガを起動するたびに自動的に登録されます。

予約済のセットアップマクロ関数名を使用してマクロ関数を定義する場合、マクロ関数を実行するタイミングを正確に定義できます。

#### セットアップマクロ関数を定義して C-SPY 起動中にロードするには、次の手順に従います。

1 マクロ関数を定義するテキストファイルを作成します。

次に例を示します。

```
execUserSetup()
{
 ...
 __registerMacroFile("MyMacroUtils.mac");
 __registerMacroFile("MyDeviceSimulation.mac");

}
```

このマクロ関数は、`MyMacroUtils.mac` と `MyDeviceSimulation.mac` の 2 つの追加マクロファイルを登録します。このマクロ関数は `execUserSetup` という関数名で定義されているため、アプリケーションのダウンロードが完了した直後に実行されます。

2 ファイル名拡張子を `mac` としてこのファイルを保存します。

3 C-SPY を起動する前に、**［プロジェクト］> ［オプション］> ［デバッガ］> ［設定］**を選択します。**［マクロファイルの使用］**チェックボックスを選択して、作成したマクロファイルを選択します。

マクロが C-SPY の起動シーケンス中に登録されるようになります。

### ［クイックウォッチ］によるマクロの実行

［クイックウォッチ］ウィンドウでは、マクロ関数を実行するタイミングを動的に選択できます。

1 以下に示す、タイマのイネーブルビットのステータスをチェックする単純なマクロ関数について考えます。

```
TimerStatus()
{
  if ((TimerStatreg & 0x01) != 0)/* reg の状態を確認 */
    return "Timer enabled"; /* 使用した C-SPY マクロ文字列 */
  else
    return "Timer disabled"; /* 使用した C-SPY マクロ文字列 */
}
```

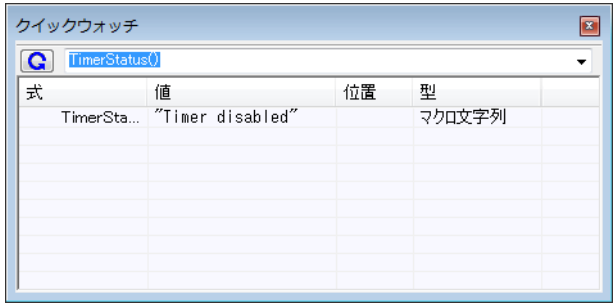
2 ファイル名の拡張子を mac としてこのマクロ関数を保存します。

3 マクロファイルをロードするには、**［表示］> ［マクロ］> ［マクロ登録］**を選択します。**［マクロ登録］**ウィンドウが表示されます。**［追加］**をクリックし、ファイルブラウザを使用してファイルを探します。マクロファイルが［マクロ登録］ウィンドウのマクロリストに表示されます。

4 登録するマクロを選択すると、そのマクロが**［デバッグマクロ］**ウィンドウに表示されます。

5 **［表示］> ［クイックウォッチ］**を選択して［クイックウォッチ］ウィンドウを開き、テキストフィールドにマクロ呼出し `TimerStatus()` と入力して [Enter] を押します。

または、マクロファイルエディタウィンドウで、マクロ関数名 `TimerStatus()` を選択します。右クリックして、表示されるコンテキストメニューで［**クイックウォッチ**］を選択します。




マクロは、［クイックウォッチ］ウィンドウに自動的に表示されます。

詳細については、120 ページの*［クイックウォッチ］ウィンドウ*を参照してください。

### ブレークポイントにマクロを接続して実行

マクロは、ブレークポイントに接続できます。これにより、ブレークポイントがトリガされると、マクロが実行されます。この方法では、特定の場所で実行を停止して、そこで特定のアクションを実行できるという長所があります。

たとえば、変数、シンボル、レジスタの値が変更された経緯などの情報を含むログレポートを簡単に作成できます。これを実現するには、疑わしい位置にブレークポイントを設定し、そのブレークポイントにログマクロを接続します。これにより、処理を実行した後で、レジスタの値が変更された経緯を調べることができます。

**ログマクロを作成してブレークポイントに接続するには、次の手順に従います。**

1. アプリケーションソースコードに以下の C 関数のスケルトンが定義されていると仮定します。

```
int fact(int x)
{
 ...
}
```

**2** 以下の例のような簡単なログマクロ関数を作成します。

```
logfact()
{
 __message "fact(" ,x, ")";
}
```

__message 文で、メッセージが［ログ］ウィンドウにロギングされます。

マクロファイルのファイル名の拡張子を mac として、ログマクロ関数を保存します。

**3** マクロを登録するには、**［表示］**>**［マクロ］**>**［マクロ登録］**を選択して、**［マクロ登録］**ウィンドウを開き、マクロファイルをリストに追加します。登録するファイルを選択します。マクロ関数が**［デバッグマクロ］**ウィンドウに表示されます。

**4** コードブレークポイントを設定するには、アプリケーションのソースコードの関数 fact 内の最初の文で**［ブレークポイントの切替え］**ボタンをクリックします。**［表示］**>**［ブレークポイント］**を選択して、［ブレークポイント］ウィンドウを開きます。ブレークポイントのリストで自分のブレークポイントを選択し、コンテキストメニューで**［編集］**コマンドを選びます。

**5** ログマクロ関数をブレークポイントに接続するには、マクロ関数名 logfact() を**［アクション］**フィールドに入力して**［適用］**をクリックします。ダイアログボックスを閉じます。

**6** アプリケーションのソースコードを実行します。ブレークポイントがトリガされると、マクロ関数が実行されます。結果は［ログ］ウィンドウに表示されます。

- **［アクション］**フィールドの式は、ブレークポイントによって実行が実際に停止する場合にのみ評価されます。値をログに記録して自動的に実行を継続する場合、以下のいずれかの方法を使用します。

  ログブレークポイントを使用（155 ページの *［ログ］ブレークポイントダイアログボックス*を参照）
- **［アクション］**フィールドではなく**［条件］**フィールドを使用 例については、145 ページの *タスクを処理して実行を継続する*を参照してください。

**7** ログマクロ関数は簡単に拡張できます。たとえば、__fmessage 文を使用すると、ファイルにログ情報を出力できます。__fmessage 文については、405 ページの *フォーマットした出力*を参照してください。

マクロをブレークポイントに接続することによってシリアルポート入力バッファをシミュレーションする例については、インフォメーションセンタのチュートリアル、*「割込みのシミュレーション」*を参照してください。

### C-SPY マクロの中止

**C-SPY マクロを中止するには、以下の手順に従います。**

1. Ctrl+Shift+.（ピリオド）をしばらく押したままにします。
2. マクロが終了したというメッセージが、［デバッグログ］ウィンドウに表示されます。

この方法は、たとえば実行が適切な時間内に終了しないなど、実行に関して何か問題があると思われる場合に使用できます。

## マクロ言語についてのリファレンス情報

リファレンス情報 :


- 401 ページの *マクロ関数*
- 402 ページの *マクロ変数*
- 402 ページの *マクロパラメータ*
- 403 ページの *マクロ文字列*
- 403 ページの *マクロ文*
- 405 ページの *フォーマットした出力*


### マクロ関数

C-SPY のマクロ関数は、C-SPY 変数定義と、マクロが呼び出されたときに実行されるマクロ文で構成されます。マクロ関数には任意の個数のパラメータを引き渡すことができます。また、マクロ関数は終了時に値を返すことができます。

C-SPY マクロの形式は、以下のとおりです。

```
macroName (parameterList)
{
  macroBody
}
```

ここで、*parameterList* にはコンマ区切りのマクロパラメータリスト、*macroBody* には C-SPY 変数定義および C-SPY 文を記述します。

タイプチェックは、マクロ関数に引き渡される値とリターン値のいずれでも実行されません。

## マクロ変数

マクロ変数は、アプリケーション外で定義して配置される変数です。C-SPY 式で使用できるほか、アプリケーションデータ（アプリケーションの変数値）を割り当てることができます。C-SPY 式の詳細については、100 ページの *C-SPY 式*を参照してください。

マクロ変数を定義する構文は、以下のとおりです。

```
__var nameList;
```

ここで、*nameList* にはコンマ区切りの C-SPY 変数名リストを指定します。

マクロ本体の外で定義したマクロ変数は、グローバルスコープになり、デバッグセッション全体に存在します。マクロ本体内部で定義されたマクロ変数は、その定義文の実行時に作成され、マクロから戻るときに破棄されます。

デフォルトでは、マクロ変数は符号付き整数として処理され、0 に初期化されます。式で C-SPY 変数に値を割り当てると、その式の型も変数に適用されます。次に例を示します。

| 式 | 意味 |
|---|---|
| `myvar = 3.5;` | `myvar` は `double` 型、値 `3.5` になりました。 |
| `myvar = (int*)i;` | `myvar` は `int` 型ポインタになり、値は `i` と同一です。 |

*表 16: C-SPY マクロ変数の例*

C のシンボルと C-SPY マクロ変数の間で名前が重複する場合は、C-SPY マクロ変数の方が C の変数よりも優先されます。 マクロ変数はデバッガホストに割り当てられるため、アプリケーションは影響されないことに注意してください。

## マクロパラメータ

マクロパラメータは、デバイスサポートをパラメータ化するためのものです。指定されたパラメータは、通常の C-SPY マクロ変数として機能し、異なる点は以下の通りです。

- パラメータの定義はイニシャライザを持つことができます。
- パラメータの値はオプションを使用して設定できます（IDE または `cspybat` を使用）。
- オプションから設定した値は、イニシャライザで設定した値より優先します。
- パラメータにはイニシャライザを含めるか、オプションを通して設定するか、あるいはその両方を満たす必要があります。そうでなければ、未定義エラーとなり、そのパラメータにアクセスするとランタイムエラーが発生します。

1 つまたは複数のマクロパラメータを定義する際の構文は以下の通りです。

```
__param param[ = value, ...;]
```

コマンドラインオプション --macro_param を使用して、パラメータの値を指定します（510 ページの *--macro_param* を参照）。

### マクロ文字列

C のデータ型に加えて、マクロ変数に*マクロ文字列*の値を保持できます。マクロ文字列は C 言語文字列と異なることに、注意してください。

C-SPY 式に "Hello!" などの文字列リテラルを書き込む場合、この値はマクロ文字列になります。char* はターゲットメモリの文字列を参照する必要がありますが、C-SPY ではターゲットメモリに実際に存在するいかなる文字列も使用できないため、これは C- 形式文字ポインタ char* ではありません。

__strFind または __subString などの組込みマクロ関数を使用して、マクロ文字列を操作できます。結果として新しいマクロ文字列が作成されます。*str* + "tail" のような + 演算子を使用して、マクロ文字列を連結できます。*str*[3] などのサブスクリプションを使用して、個々の文字も取得できます。sizeof(*str*) を使用して、文字列の長さを取得できます。マクロ文字列は NULL 終了ではないことに注意してください。

マクロ関数 __toString を使用して、アプリケーション内の NULL 終了 C 文字列（char* または char[]）からマクロ文字列に変換します。たとえば、アプリケーションに以下の C 文字列の定義があると仮定します。

```
char const *cstr = "Hello";
```

次に、以下のマクロの例を検討します。

```
__var str;          /* マクロ変数 */
str = cstr          /* str は現在、char へのポインタです */
sizeof str          /* sizeof (char*) と同様で、通常は 2 か 4 です */
str = __toString(cstr,512)  /* str は現在、マクロ文字列です */
sizeof str          /* 5、文字列の長さ */
str[1]              /* 101、'e' の ASCII コード */
str += " World!"    /* str はここで "Hello World!" になります */
```

405 ページの*フォーマットした出力*も参照してください。

### マクロ文

マクロ文は、相当する C 文と同様に機能します。以下の C-SPY マクロ文を使用できます。

### 式

*式*;

C-SPY 式の詳細については、100 ページの *C-SPY 式*を参照してください。

### 条件文

```
if（式）
  文

if（式）
  文
else
  文
```

### ループ文

```
for (init_expression; cond_expression; update_expression)
  文

while（式）
  文

do
  文
while（式）;
```

### return 文

```
return;

return 式;
```

リターン値が明示的に設定されていない場合は、デフォルトでは `signed int 0` が返されます。

### ブロック

マクロ文をブロックとしてグループ化できます。

```
{
  statement1
  statement2
  .
  .
  .
  statementN
}
```

### フォーマットした出力

C-SPY では、フォーマットした出力をさまざまな方法で生成できます。

| | |
|---|---|
| `__message` *argList*; | ［デバッグログ］ウィンドウに出力します。 |
| `__fmessage` *file*, *argList*; | 指定ファイルに出力します。 |
| `__smessage` *argList*; | フォーマットした出力を文字列に格納して戻します。 |

ここで、*argList* は C-SPY の式か文字列をコンマで区切ったリストで、*file* は `__openFile` システムマクロの実行結果です（436 ページの *__openFile* を参照）。

［デバッグログ］ウィンドウにメッセージを出力するには、以下の手順に従います。

```
var1 = 42;
var2 = 37;
__message "This line prints the values ", var1, " and ", var2,
" in the Log window.";
```

この手順を実行すると、以下のメッセージが［ログ］ウィンドウに出力されます。

```
This line prints the values 42 and 37 in the Log window.
```

出力を指定のファイルに書き込む場合：

```
__fmessage myfile, "Result is ", res, "!¥n";
```

文字列を生成する場合：

```
myMacroVar = __smessage 42, " is the answer.";
```

`myMacroVar` に、ここで文字列 `"42 is the answer."` が格納されます。

### 引数の表示フォーマットの指定

*argList* のスカラ引数（数値またはポインタ）のデフォルトの表示フォーマットは、後に：およびフォーマット指定子を記述することで 変更することができます。使用可能な指定子は次のとおりです。

| | |
|---|---|
| `%b` | 2 進数のスカラ引数 |
| `%o` | 8 進数のスカラ引数 |
| `%d` | 10 進数のスカラ引数 |

| | |
|---|---|
| `%x` | 16 進数のスカラ引数 |
| `%c` | 文字のスカラ引数 |

これらの指定子は、［ウォッチ］ウィンドウや［ロケール］ウィンドウで使用可能なフォーマットと同一ですが、プレフィックスや文字列 / 文字の前後の引用符は出力されません。別の例を示します。

```
__message "The character '", cvar:%c, "' has the decimal value
", cvar;
```

変数の値に応じて、以下のメッセージが出力されます。

```
The character 'A' has the decimal value 65
```

**注 :** 単一引用符で括った文字（文字定数）は整数定数として処理され、文字としてフォーマットされません。次に例を示します。

```
__message 'A', " is the numeric value of the character ",
'A':%c;
```

この場合、次のように出力されます。

```
65 is the numeric value of the character A
```

**注 :** 特定タイプ用デフォルトフォーマットは主に［ウォッチ］ウィンドウやその他の関連するウィンドウで使用できるように設計されています。たとえば、タイプ `char` は 'A' (0x41) に、文字（主に C 文字列）用ポインタは 0x8102 "Hello" にフォーマットされるように、文字列部分には文字列の始めの部分（現在、60 文字まで）が表示されます。

`char*` 型の値の出力時にフォーマット指定子 `%x` を使用して、ポインタ値を 16 進数表記で出力するか、またはシステムマクロ `__toString` を使用して全文字列値を取得します。

## 予約済みのセットアップマクロ関数名についてのリファレンス情報

セットアップマクロの定義に使用できる、予約済みのセットアップマクロ関数があります。これらの予約済みの名前を使用することにより、実行中に関数が定義された段階で実行されます。詳細については、394 ページの*セットアップマクロ関数およびファイルの概要*を参照してください。

リファレンス情報 :

- `execUserPreload`
- `execUserExecutionStarted`
- `execUserExecutionStopped`
- `execUserFlashInit`

- execUserSetup
- execUserFlashReset
- execUserPreReset
- execUserReset
- execUserExit
- execUserFlashExit

## execUserPreload

**構文** execUserPreload

**適用範囲** すべての C-SPY ドライバ。

**説明** ターゲットシステムとの通信確立後、ターゲットアプリケーションのダウンロード前に呼び出します。

このマクロは、データを適切にロードするために必要なメモリアドレス（ロケーション）/ レジスタを初期化する場合に実装します。

## execUserExecutionStarted

**構文** execUserExecutionStarted

**適用範囲** すべての C-SPY ドライバ。

**説明** デバッガが実行を開始または再開しようとするときに呼び出されます。このマクロは、命令が 1 つのアセンブラステップ（つまり、[逆アセンブリ] ウィンドウの [ステップ] または [ステップイン]）を実行するときには呼び出されません。

## execUserExecutionStopped

**構文** execUserExecutionStopped

**適用範囲** すべての C-SPY ドライバ。

**説明** デバッガが実行を停止したときに呼び出されます。このマクロは、命令が 1 つのアセンブラステップ（つまり、[逆アセンブリ] ウィンドウの [ステップ] または [ステップイン]）を実行するときには呼び出されません。

## execUserFlashInit

**構文** `execUserFlashInit`

**適用範囲** C-SPY ハードウェアデバッガドライバ。

**説明** フラッシュローダが RAM にダウンロードされる前に一度呼び出します。通常、このマクロは、フラッシュローダで必要なメモリマップを設定する場合に実装します。このマクロは、フラッシュのプログラミング時に一度だけ呼び出します。また、フラッシュローダ機能用にのみ使用します。

## execUserSetup

**構文** `execUserSetup`

**適用範囲** すべての C-SPY ドライバ。

**説明** ターゲットアプリケーションのダウンロード後に一度呼び出します。

このマクロは、メモリマップ、ブレークポイント、割込み、レジスタマクロファイルなどの設定を行う場合に実装します。

システム起動時に実行されるマクロファイル（`execUserSetup` を使用）で割込みやブレークポイントを定義する場合は、システム終了時にそれらが削除（`execUserExit` を使用）されていることを確認してください。サンプルは `SetupSimple.mac` にあります。インフォメーションセンタのチュートリアルを参照してください。

シミュレータでは割込み設定がセッション終了後も保持されるため、削除していない場合、`execUserSetup` の実行ごとに重複して作成されます。これが原因で、実行速度が大幅に低下します。

## execUserFlashReset

**構文** `execUserFlashReset`

**適用範囲** C-SPY ハードウェアデバッガドライバ。

**説明** フラッシュローダが RAM にダウンロードされた後、フラッシュローダの実行前に一度呼び出します。このマクロは、フラッシュのプログラミング時に一度だけ呼び出します。また、フラッシュローダ機能用にのみ使用します。

## execUserPreReset

**構文** `execUserPreReset`

**適用範囲** すべての C-SPY ドライバ。

**説明** リセットコマンドの実行直前に呼び出します。

このマクロを実装して、必要なデバイスの状態を設定します。

## execUserReset

**構文** `execUserReset`

**適用範囲** すべての C-SPY ドライバ。

**説明** リセットコマンドの実行直後に呼び出します。

このマクロは、データの設定 / 復元を行う場合に実装します。

## execUserExit

**構文** `execUserExit`

**適用範囲** すべての C-SPY ドライバ。

**説明** デバッグセッションの終了時に一度呼び出します。

このマクロは、状態データなどの保存を行う場合に実装します。

## execUserFlashExit

**構文** `execUserFlashExit`

**適用範囲** C-SPY ハードウェアデバッガドライバ。

**説明** フラッシュプログラミングの終了時に一度呼び出します。

このマクロは、状態データなどの保存を行う場合に実装します。このマクロは、フラッシュローダ機能用に使用します。

# C-SPY システムマクロについてのリファレンス情報

ここでは、各 C-SPY システムマクロのリファレンスを収録しています。

以下の表に定義済みシステムマクロをまとめています。

| マクロ | 説明 |
|---|---|
| __cancelAllInterrupts | 設定されたすべての割込みを取り消します。 |
| __cancelInterrupt | 割込みを取り消します。 |
| __clearBreak | ブレークポイントを削除します。 |
| __closeFile | __openFile で開かれたファイルを閉じます。 |
| __delay | 実行を遅らせます。 |
| __disableInterrupts | 割込み生成を無効にします。 |
| __driverType | ドライバタイプを確認します。 |
| __emulatorSpeed | エミュレータクロック周波数を設定します。 |
| __emulatorStatusCheckOnRead | 各リード処理後の CPSR レジスタの検証を有効 / 無効にします。 |
| __enableInterrupts | 割込み生成を有効にします。 |
| __evaluate | 入力文字列を式として解釈して、評価します。 |
| __fillMemory8 | 1 バイトの値で指定したメモリエリアをフィルします。 |
| __fillMemory16 | 2 バイトの値で指定したメモリエリアをフィルします。 |
| __fillMemory32 | 4 バイトの値で指定したメモリエリアをフィルします。 |
| __gdbserver_exec_command | 文字列やコマンドを GDB サーバに送信します。 |
| __getSelectedCore | 現在のコアの番号を取得します。 |
| __getTracePortSize | トレースポートの幅を返します。 |
| __hasDAPRegs | C-SPY ドライバがマクロ __readAPReg、__readDPReg、__writeAPReg、__writeDPReg をサポートする場合、True を返します。 |
| __hwJetResetWithStrategy | ハードウェアリセットを実行し、ターゲット CPU を停止します。 |
| __hwReset | ハードウェアリセットを実行し、ターゲット CPU を停止します。 |

*表 17: システムマクロのまとめ*

| マクロ | 説明 |
|---|---|
| __hwResetRunToBp | ハードウェアリセットを実行した後、指定されたアドレスまで実行します。 |
| __hwResetWithStrategy | ハードウェアリセットを実行し、ターゲット CPU を遅れて停止します。 |
| __isBatchMode | C-SPY がバッチモードで実行中かどうかをチェックします。 |
| __jlinkExecCommand | 低レベルコマンドを J-Link/J-Trace ドライバに送信します。 |
| __jtagCommand | 低レベルコマンドを JTAG 命令レジスタに送信します。 |
| __jtagCP15IsPresent | コプロセッサ CP15 が使用できるかどうかを確認します。 |
| __jtagCP15ReadReg | コプロセッサ CP15 のレジスタ値を返します。 |
| __jtagCP15WriteReg | コプロセッサ CP15 レジスタにライトします。 |
| __jtagData | 低レベルデータ値を JTAG データレジスタに送信します。 |
| __jtagRawRead | JTAG インタフェースからリードデータを戻します。 |
| __jtagRawSync | 蓄積されたデータを JTAG インタフェースにライトします。 |
| __jtagRawWrite | JTAG に転送されるデータを蓄積します。 |
| __jtagResetTRST | TRST JTAG 信号経由で ARM TAP コントローラをリセットします。 |
| __loadImage | イメージをロードします。 |
| __memoryRestore | ファイルの内容を指定したメモリゾーンに復元します。 |
| __memorySave | 指定したメモリエリアの内容をファイルに保存します。 |
| __messageBoxYesNo | ユーザとの対話形式による「はい / いいえ」のダイアログボックスが表示されます。 |
| __openFile | ファイルを入出力処理用に開きます。 |
| __orderInterrupt | 割込みを生成します。 |
| __popSimulatorInterruptExecutingStack | 割込みシミュレーションシステムに、割込みハンドラの実行が終了したことを通知します。 |
| __readAPReg | AP レジスタからリードします。 |

*表 17: システムマクロのまとめ（続き）*

| マクロ | 説明 |
|---|---|
| __readDPReg | DP レジスタからリードします。 |
| __readFile | 指定したファイルからリードします。 |
| __readFileByte | 指定したファイルから 1 バイトリードします。 |
| __readMemory8,<br>__readMemoryByte | 指定したメモリアドレス（ロケーション）から 1 バイトリードします。 |
| __readMemory16 | 指定したメモリアドレス（ロケーション）から 2 バイトリードします。 |
| __readMemory32 | 指定したメモリアドレス（ロケーション）から 4 バイトリードします。 |
| __registerMacroFile | 指定したファイルからマクロを登録します。 |
| __resetFile | __openFile で開かれたファイル内の位置を先頭に戻します。 |
| __restoreSoftwareBreakpoints | システム起動中に破壊されたブレークポイントを復元します。 |
| __selectCore | 現在のコアから指定したコアにフォーカスを移します。 |
| __setCodeBreak | コードブレークポイントを設定します。 |
| __setDataBreak | データブレークポイントを設定します。 |
| __setDataLogBreak | データログブレークポイントを設定します。 |
| __setLogBreak | ログブレークポイントを設定します。 |
| __setSimBreak | シミュレーションブレークポイントを設定します。 |
| __setTraceStartBreak | トレース開始ブレークポイントを設定します。 |
| __setTraceStopBreak | トレース停止ブレークポイントを設定します。 |
| __sourcePosition | 現在の実行位置がソース位置に対応する場合、ファイル名とソース位置を返します。 |
| __strFind | 指定した文字列で別の文字列を検索します。 |
| __subString | 文字列から部分文字列を抽出します。 |
| __targetDebuggerVersion | ターゲットデバッガのバージョンを返します。 |
| __toLower | パラメータ文字列のコピーを、すべての文字を小文字に変換して返します。 |
| __toString | 文字列を出力します。 |
| __toUpper | パラメータ文字列のコピーを、すべての文字を大文字に変換して返します。 |

*表 17: システムマクロのまとめ（続き）*

| マクロ | 説明 |
|---|---|
| __unloadImage | デバッグイメージをアンロードします。 |
| __writeAPReg | AP レジスタにライトします。 |
| __writeDPReg | DP レジスタにライトします。 |
| __writeFile | 指定したファイルにライトします。 |
| __writeFileByte | 指定したファイルに 1 バイトライトします。 |
| __writeMemory8,<br>__writeMemoryByte | 指定したメモリアドレス（ロケーション）に 1 バイトライトします。 |
| __writeMemory16 | 指定したメモリアドレス（ロケーション）に 2 バイトワードをライトします。 |
| __writeMemory32 | 指定したメモリアドレス（ロケーション）に 4 バイトワードをライトします。 |

*表 17: システムマクロのまとめ（続き）*

## __cancelAllInterrupts

**構文** `__cancelAllInterrupts()`

**リターン値** `int 0`

**適用範囲** C-SPY シミュレータ。

**説明** 設定されたすべての割込みを取り消します。

## __cancelInterrupt

**構文** `__cancelInterrupt(`*`interrupt_id`*`)`

**パラメータ**

*`interrupt_id`*

対応する `__orderInterrupt` マクロ呼出し (`unsigned long`) によって返される値。

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | `int 0` |
| 失敗 | ゼロ以外のエラー番号 |

*表 18: __cancelInterrupt のリターン値*

**適用範囲** C-SPY シミュレータ。

**説明** 指定した割込みを取り消します。

## __clearBreak

**構文** `__clearBreak(break_id)`

**パラメータ** *break_id*
設定したブレークポイントマクロのいずれかが返した値。

**リターン値** `int 0`

**適用範囲** すべての C-SPY ドライバ。

**説明** ユーザ定義ブレークポイントを削除します。

**関連項目** 133 ページの *ブレークポイント*。

## __closeFile

**構文** `__closeFile(fileHandle)`

**パラメータ** *fileHandle*
`__openFile` マクロでファイルハンドルとして使用するマクロ変数。

**リターン値** `int 0`

**適用範囲** すべての C-SPY ドライバ。

**説明** 先に `__openFile` で開いたファイルを閉じます。

## __delay

**構文** `__delay(value)`

**パラメータ** *value*
実行を遅らせる時間（ミリ秒）。

**リターン値** `int 0`

適用範囲　すべての C-SPY ドライバ。

説明　指定したミリ秒数だけ実行を遅らせます。

## __disableInterrupts

構文　`__disableInterrupts()`

リターン値

| 結果 | 値 |
|---|---|
| 成功 | `int 0` |
| 失敗 | ゼロ以外のエラー番号 |

*表 19: __disableInterrupts のリターン値*

適用範囲　C-SPY シミュレータ。

説明　割込み生成を無効にします。

## __driverType

構文　`__driverType(`*`driver_id`*`)`

パラメータ　*`driver_id`*

チェックするドライバに対応する文字列。以下のいずれかです。

`"sim"` はシミュレータドライバに対応します。

`"angel"` は C-SPY Angel ドライバに対応します。

`"cmsisdap"` は C-SPY CMSIS-DAP ドライバに対応します。

`"gdbserv"` は C-SPY GDB サーバドライバに対応します。

`"generic"` はサードパーティのドライバに対応します。

`"ijet"` は C-SPY I-jet/JTAGjet ドライバに対応します。

`"jlink"` は C-SPY J-Link/J-Trace ドライバに対応します。

`"jtag"` は C-SPY Macraigor ドライバに対応します。

`"lmiftdi"` は C-SPY TI Stellaris ドライバに対応します。

`"xds"` は C-SPY TI XDS ドライバに対応します。

`"rdi"` は C-SPY RDI ドライバに対応します。

"rom" は C-SPY IAR ROM モニタドライバに対応します。

"stlink" は C-SPY ST-LINK ドライバに対応します。

リターン値

| 結果 | 値 |
|---|---|
| 成功 | 1 |
| 失敗 | 0 |

*表 20: __driverType のリターン値*

適用範囲　すべての C-SPY ドライバ。

説明　現在の C-SPY ドライバが、*driver_id* パラメータで指定したドライバタイプと同一かどうかを確認します。

例

```
__driverType("sim")
```

シミュレータが現在のドライバである場合は、1 が返されます。それ以外の場合は 0 が返されます。

## __emulatorSpeed

構文

```
__emulatorSpeed(speed)
```

パラメータ

| | |
|---|---|
| *speed* | エミュレータ速度（単位は Hz）です。0（ゼロ）を使用すると、自動的に検出された速度にします。-1 を使用すると、適用する速度にします（アダプティブ速度をサポートしているエミュレータのみ）。 |

リターン値

| 結果 | 値 |
|---|---|
| 成功 | 前の速度、あるいは未知の場合は 0（ゼロ） |
| 不成功、エミュレータが未サポートの速度 | -1 |

*表 21: __emulatorSpeed のリターン値*

適用範囲　C-SPY ハードウェアドライバ。

説明　エミュレータクロック周波数を設定します。JTAG インタフェースの場合、TCK 信号に見られる JTAG クロック周波数です。

例

```
__emulatorSpeed(0)
```

自動的に検出するエミュレータ速度を設定します。

## __emulatorStatusCheckOnRead

構文

```
__emulatorStatusCheckOnRead(status)
```

パラメータ

| | |
|---|---|
| *status* | 0 は、チェックを有効にします（デフォルト）。1 は、チェックを無効にします。 |

リターン値

```
int 0
```

適用範囲

C-SPY J-Link/J-Trace ドライバ。

C-SPY I-jet/JTAG-jet ドライバの場合、このマクロは認識されますが動作しません。

説明

各リード処理後に行う CPSR（最新プロセッサステータスレジスタ）のドライバ検証を有効 / 無効にします。一般的にこのマクロは、Texas Instruments TMS470R1B1M など、いくつかの CPU で JTAG 接続を開始する場合に使用できます。

**注 :** この検証を有効にすると、たとえば無効な CPSR 値が戻された場合などに、CPU で問題の発生する原因となる場合があります。ただし、この検証が無効であると (`SetCheckModeAfterRead = 0`)、リード処理の成功を検証できず、またデータが中断する可能性を検出することができません。

例

```
__emulatorStatusCheckOnRead(1)
```

メモリリード時のデータ中止のチェックを無効にします。

## __enableInterrupts

構文

```
__enableInterrupts()
```

リターン値

| 結果 | 値 |
|---|---|
| 成功 | `int 0` |
| 失敗 | ゼロ以外のエラー番号 |

*表 22: __enableInterrupts のリターン値*

**適用範囲** C-SPY シミュレータ。

**説明** 割込み生成を有効にします。

## __evaluate

**構文** `__evaluate(string, valuePtr)`

**パラメータ**

*string*
式文字列。

*valuePtr*
結果を保持するマクロ変数ポインタ。

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | int 0 |
| 失敗 | int 1 |

*表 23: __evaluate リターン値*

**適用範囲** すべての C-SPY ドライバ。

**説明** このマクロは入力文字列を式として解釈して、評価します。結果は *valuePtr* で参照される変数に格納されます。

**例** 以下の例では、変数 i が定義され、値 5 に設定されていると仮定しています。

```
__evaluate("i + 3", &myVar)
```

マクロ変数 myVar に値 8 が割り当てられます。

## __fillMemory8

**構文** `__fillMemory8(value, address, zone, length, format)`

**パラメータ**

*value*
値を指定する整数。

*address*
メモリの開始アドレスを指定する整数。

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

*length*

影響が及ぶバイト数を指定する整数。

*format*

以下のいずれかが必要です。

| | |
|---|---|
| COPY | *[ 値 ]* に入力した値が指定したメモリエリアにコピーされます。 |
| AND | [AND] を指定すると、*[ 値 ]* の値とメモリの既存値の論理積がメモリに書き込まれます。 |
| OR | [OR] を指定すると、*[ 値 ]* の値とメモリの既存値の論理積がメモリに書き込まれます。 |
| XOR | [XOR] を指定すると、*[ 値 ]* の値とメモリの既存値の論理積がメモリに書き込まれます。 |

**リターン値**

```
int 0
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

1 バイトの値で指定したメモリエリアをフィルします。

**例**

```
__fillMemory8(0x80, 0x700, "", 0x10, "OR");
```

## __fillMemory16

**構文**

```
__fillMemory16(value, address, zone, length, format)
```

**パラメータ**

*value*

値を指定する整数。

*address*

メモリの開始アドレスを指定する整数。

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

*length*

影響が及ぶ 2 バイトのエンティティ数を定義する整数。

*format*

以下のいずれかが必要です。

| | |
|---|---|
| Copy | *[value]* に入力した値が指定したメモリエリアにコピーされます。 |
| AND | [AND] を指定すると、*[ 値 ]* の値とメモリの既存値の論理積がメモリに書き込まれます。 |
| OR | [OR] を指定すると、*[ 値 ]* の値とメモリの既存値の論理積がメモリに書き込まれます。 |
| XOR | [XOR] を指定すると、*[ 値 ]* の値とメモリの既存値の論理積がメモリに書き込まれます。 |

**リターン値** int 0

**適用範囲** すべての C-SPY ドライバ。

**説明** 2 バイトの値で指定したメモリエリアをフィルします。

**例**

```
__fillMemory16(0xCDCD, 0x7000, "", 0x200, "Copy");
```

## __fillMemory32

**構文**

```
__fillMemory32(value, address, zone, length, format)
```

**パラメータ**

*value*

値を指定する整数。

*address*

メモリの開始アドレスを指定する整数。

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

*length*

影響が及ぶ 4 バイトのエンティティ数を定義する整数。

*format*

以下のいずれかが必要です。

| | |
|---|---|
| COPY | *[ 値 ]* に入力した値が指定したメモリエリアにコピーされます。 |
| AND | [AND] を指定すると、*[ 値 ]* の値とメモリの既存値の論理積がメモリに書き込まれます。 |
| OR | [OR] を指定すると、*[ 値 ]* の値とメモリの既存値の論理積がメモリに書き込まれます。 |
| XOR | [XOR] を指定すると、*[ 値 ]* の値とメモリの既存値の論理積がメモリに書き込まれます。 |

**リターン値** int 0

**適用範囲** すべての C-SPY ドライバ。

**説明** 4 バイトの値で指定したメモリエリアをフィルします。

**例**

```
__fillMemory32(0x0000FFFF, 0x4000, "", 0x1000, "XOR");
```

## __gdbserver_exec_command

**構文**

```
__gdbserver_exec_command("string")
```

**パラメータ**

| | |
|---|---|
| "*string*" | GDB サーバに送信する文字列またはコマンドです。詳細についてはドキュメントを参照してください。 |

**適用範囲** C-SPY C-SPY GDB サーバドライバ。

**説明** このオプションは、文字列やコマンドを GDB サーバに送信するときに使用します。

## __getSelectedCore

**構文**

```
__getSelectedCore()
```

**リターン値** 現在のコア。コアの番号は 0 から順に大きくなります。

**適用範囲**

C-SPY シミュレータ。

C-SPY I-jet/JTAGjet ドライバ。

**説明**

現在のコアの番号を取得します。

**例**

```
test ()
{
  __message "Core: ", __getSelectedCore(), " pc = ", #PC:%x, ¥n;
  __selectCore(0);
  __message "Core: ", __getSelectedCore(), " pc = ", #PC:%x, ¥n;
  __selectCore(1);
  __message "Core: ", __getSelectedCore(), " pc = ", #PC:%x, ¥n;
```

上記のマクロの一般的な結果は次のようになります（当初のコアの番号が 1 の場合）：

```
Core: 1 pc = 0000213C
Core: 0 pc = 00000494
Core: 1 pc = 0000213C
```

**関連項目**

443 ページの *__selectCore*。

## __getTracePortSize

**構文**

```
__getTracePortSize
```

**リターン値**

| 結果 | 値 |
|---|---|
| トレースポート幅（ビット単位）。 | 1、2、4、8、16 |

*表 24: __getTracePortSize のリターン値*

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ。

C-SPY J-Link/J-Trace ドライバ。

**説明**

トレースポートの幅を返します。

**関連項目**

226 ページの *［ETM トレース設定］ダイアログボックス*および 229 ページの *［ETM トレース設定］ダイアログボックス (J-Link/J-Trace)*。

## __hasDAPRegs

**構文**

```
__hasDAPRegs()
```

**リターン値**

| 結果 | 値 |
|---|---|
| C-SPY ドライバは、最新の CPU コアのマクロ __readAPReg、__readDPReg、__writeAPReg、__writeDPReg をサポートします。 | true |
| C-SPY ドライバは、最新の CPU コアのマクロ __readAPReg、__readDPReg、__writeAPReg、__writeDPReg をサポートしません。 | false |

*表 25: __hasDAPRegs のリターン値*

**適用範囲**

C-SPY ハードウェアドライバ。

**説明**

このマクロは、C-SPY ドライバが最新の CPU コアでマクロ __readAPReg、__readDPReg、__writeAPReg、__writeDPReg をサポートする場合に、true を返し、それ以外は false を返します。

## __hwJetResetWithStrategy

**構文**

```
__hwJetResetWithStrategy(halt_delay, strategy)
```

**パラメータ**

| | |
|---|---|
| *halt_delay* | リセットパルスの終わりと CPU の停止との間の遅延（マイクロ秒）です。0（ゼロ）を設定すると、リセット後すぐに CPU を停止します（*strategy* が 0 に設定されている場合にのみ）。 |
| *strategy* | リセット方式の番号。サポートされているリセット方式については、502 ページの *--jet_standard_reset* を参照してください。 |

**リターン値**

| 結果 | 値 |
|---|---|
| 成功。この遅延機能はデバッグプローブではサポートされていません。 | -1 |
| 不成功。このリセット方式はデバッグプローブではサポートされていません。 | -3 |
| 不成功。その他 | -4 |

*表 26: __hwJetResetWithStrategy のリターン値*

**適用範囲** C-SPY I-jet/JTAGjet ドライバ。

**説明** 実行するリセット方法を指定します。

**例**

```
__hwJetResetWithStrategy(0,2)
```

ハードウェアリセットを実行します。

## __hwReset

**構文**

```
__hwReset(halt_delay)
```

**パラメータ**

| | |
|---|---|
| *halt_delay* | リセットパルスの終わりと CPU の停止との間の遅延（マイクロ秒）です。0（ゼロ）を設定すると、リセット後すぐに CPU を停止します。 |

**リターン値**

| 結果 | 値 |
|---|---|
| 成功。実際の遅延値がエミュレータで実現 | >=0 |
| 成功。この遅延機能はエミュレータが未サポート | -1 |
| 不成功。ハードウェアリセットはエミュレータが未サポート | -2 |

*表 27: __hwReset のリターン値*

**適用範囲** このシステムマクロは、すべての JTAG インタフェースに使用できます。

**説明** ハードウェアリセットを実行し、ターゲット CPU を停止します。

**例**

```
__hwReset(0)
```

CPU をリセットし、ただちに停止します。

## __hwResetRunToBp

**構文**

```
__hwResetRunToBp(strategy, breakpoint_address, timeout)
```

**パラメータ**

| | |
|---|---|
| *strategy* | C-SPY I-jet/JTAG-jet ドライバでサポートされているリセット方式については、502 ページの *--jet_standard_reset* を参照してください。C-SPY J-Link ドライバでサポートされているリセット方式について詳しくは、*『ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド』* を参照してください。 |
| *breakpoint_address* | 実行を停止するブレークポイントのアドレスです。整数値を指定します（シンボルは使用できません）。 |
| *timeout* | ブレークポイントのタイムアウト（ミリ秒）。指定した時間内にブレークポイントに達しない場合、コアが停止します。 |

**リターン値**

| 値 | 結果 |
|---|---|
| >=0 | 成功。ブレークポイントにヒットするまでのおおよその実行時間 (ms) です。 |
| -2 | 不成功。ハードウェアリセットはエミュレータでサポートされていません。 |
| -3 | 不成功。このリセット方式はエミュレータでサポートされていません。 |

*表 28: __hwResetRunToBp のリターン値*

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

**説明**

ハードウェアリセット、指定アドレスでのブレークポイントの設定、ブレークポイントまでの実行、ブレークポイントの削除を行います。ブレークポイントアドレスは、ダウンロードされたイメージが RAM にコピーされた後の開始アドレスです。

このマクロは、アプリケーションイメージをフラッシュから RAM にコピーするブートローダの実行を目的としています。イメージがフラッシュにダウンロードされた後、イメージが検証される前に実行されます。このマクロは、`execUserFlashExit` または `execUserPreload` で実行できます。

例

```
__hwResetRunToBp(0,0x400000,10000)
```

リセット方法 0 を使用して CPU をリセットし、アドレス 0x400000 まで実行します。10 秒以内にブレークポイントに達しない場合には、指定されたタイムアウト時間に従って停止を実行します。

## __hwResetWithStrategy

構文

```
__hwResetWithStrategy(halt_delay, strategy)
```

パラメータ

*halt_delay* リセットパルスの終わりと CPU の停止との間の遅延（マイクロ秒）です。0（ゼロ）を設定すると、リセット後すぐに CPU を停止します（*strategy* が 0 に設定されている場合にのみ）。

*strategy* C-SPY I-jet/JTAGjet ドライバは方式 2（ハードウェアリセット）のみサポートしています。C-SPY J-Link ドライバでサポートされているリセット方式について詳しくは、*『ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド』*を参照してください。

リターン値

| 結果 | 値 |
|---|---|
| 成功。実際の遅延（ミリ秒）。エミュレータで実現 | >=0 |
| 成功。この遅延機能はエミュレータが未サポート | -1 |
| 不成功。ハードウェアリセットはエミュレータが未サポート | -2 |
| 不成功。このリセット方式はエミュレータが未サポート | -3 |

*表 29: __hwResetWithStrategy のリターン値*

適用範囲

C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

このマクロは他の C-SPY ハードウェアドライバにも存在しますが、そこでは何の働きもしません。

説明

ハードウェアリセットを実行し、ターゲット CPU を遅れて停止します。

例

```
__hwResetWithStrategy(0,1)
```

CPU をリセットし、メモリアドレスがゼロのブレークポイントを使用して停止します。

## __isBatchMode

構文

```
__isBatchMode()
```

リターン値

| 結果 | 値 |
|---|---|
| True | int 1 |
| False | int 0 |

*表 30: __isBatchMode のリターン値*

適用範囲

すべての C-SPY ドライバ。

説明

このマクロは、デバッガがバッチモードで実行中の場合に True（真）を、それ以外の場合は False（偽）を返します。

## __jlinkExecCommand

構文

```
__jlinkExecCommand(cmdstr)
```

パラメータ

| | |
|---|---|
| *cmdstr* | J-Link/J-Trace コマンド文字列 |

リターン値

```
int 0
```

適用範囲

C-SPY J-Link/J-Trace ドライバ。

説明

低レベルコマンドを J-Link/J-Trace ドライバに送信します。使用可能なコマンドの一覧は、*『ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド』*を参照してください。

例

*『ARM コア向け JTAG エミュレータ用 IAR J-Link および IAR J-Trace ユーザガイド』*を参照してください。

関連項目

506 ページの *--jlink_exec_command*。

## __jtagCommand

構文

```
__jtagCommand(ir)
```

パラメータ

*ir* は以下のいずれかです。

| | |
|---|---|
| 2 | SCAN_N |
| 4 | RESTART |
| 12 | INTEST |
| 14 | IDCODE |
| 15 | BYPASS |

**リターン値** int 0

**適用範囲** C-SPY J-Link/J-Trace ドライバ。

**説明** 低レベルコマンドを JTAG 命令レジスタ IR に送信します。

**例**

```
__jtagCommand(14);
Id = __jtagData(0,32);
```

ARM ターゲットデバイスの JTAG ID を戻します。

## __jtagCP15IsPresent

**構文** `__jtagCP15IsPresent()`

**リターン値** 1（CP15 が使用できる場合）。それ以外の場合は 0。

**適用範囲** C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

**説明** コプロセッサ CP15 が使用できるかどうかを確認します。

## __jtagCP15ReadReg

**構文** `__jtagCP15ReadReg(CRn, CRm, op1, op2)`

**パラメータ** MRC 命令のパラメータ（レジスタおよびオペランド）。詳細については、*『ARM Architecture Reference Manual』* を参照してください。*op1* は常に 0 である必要がある点に注意してください。

**リターン値** レジスタ値です。

**適用範囲** C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

**説明** CP15 レジスタの値を呼び込み、その値を戻します。

## __jtagCP15WriteReg

**構文** `__jtagCP15WriteReg(CRn, CRm, op1, op2, value)`

**パラメータ** MRC 命令のパラメータ（レジスタおよびオペランド）。詳細については、『*ARM Architecture Reference Manual*』を参照してください。*op1* は、常に 0、*value* が書き込まれる値である点に注意してください。

**適用範囲** C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

**説明** CP15 レジスタに値を書込みます。

## __jtagData

**構文** `__jtagData(dr, bits)`

**パラメータ**

| | |
|---|---|
| *dr* | 32 ビットデータレジスタ値 |
| *bits* | *dr* の有効ビット数です。マクロパラメータおよびリターン値の両方に対応します。最下位ビットから始まります (1...32) |

**リターン値** 処理結果を戻します。結果のビット数は *bits* パラメータで指定されます。

**適用範囲** C-SPY J-Link/J-Trace ドライバ。

**説明** 低レベルデータ値を JTAG データレジスタ DR に送信します。DR からシフトされたビットが戻されます。

**例**

```
__jtagCommand(14);
Id = __jtagData(0,32);
```

ARM ターゲットデバイスの JTAG ID を戻します。

## __jtagRawRead

**構文**

```
__jtagRawRead(bitpos, numbits)
```

**パラメータ**

| | |
|---|---|
| *bitpos* | 戻された JTAG ビットの開始ビット位置で、そこからデータを戻します。 |
| *numbits* | 読み込みビット数です。最大値は 32 です。 |

**適用範囲**

J-Link/J-Trace ドライバ

**説明**

JTAG TDO から読み込まれたデータを戻します。最下位ビットだけはデータを含みます。最下位ビットリードは最下位ビットから行われます。任意の数だけこの関数を呼び出すと、操作で戻すすべてのビットを取得することができます。この関数は、蓄積された書込みビットの同期を間接的に取ることも行います。

**例**

以下は、TMS および TDI ピンにある JTAG へデータを書き込む方法と、TDO からデータを読み込む擬似コードを示します。

```
__var Id;
__var BitPos;
/*****************************************************************
*
* ReadId()
*/
ReadId() {
__message "Reading JTAG Id¥n";
__jtagRawWrite(0, 0x1f, 6); /*RESET ステート経由で IDLE へ移動 */
__jtagRawWrite(0, 0x1, 3); /*DR スキャンチェインを入力 */
BitPos = __jtagRawWrite(0, 0x80000000, 32); /*32 ビットを
                    DR にシフト。Read 操作の BitPos を記憶 */
__jtagRawWrite(0, 0x1, 2); /*IDLE へ移動 */
Id = __jtagRawRead(BitPos, 32); /*Id を読み込み */
__message "JTAG Id: ", Id:%x, "¥n";
}
```

## __jtagRawSync

**構文**

```
__jtagRawSync()
```

**リターン値**

```
int 0
```

**適用範囲**　C-SPY J-Link/J-Trace ドライバ。

**説明**　任意のデータを JTAG インタフェースに送信します。__jtagRawWrite を使用して蓄積されたすべてのビットは、JTAG スキャンチェーンに書き込まれます。このデータは TCK と同期を取って送信され、通常は TCK の立上がりエッジ上のデバイスによってサンプリングされます。

**例**　以下は、TMS および TDI ピンにある JTAG へデータを書き込む方法と、TDO からデータを読み込む擬似コードを示します。

```
int i;
U32 tdo;
for (i = 0; i < numBits; i++) {
  TDI = tdi & 1; /* TDI ピンを設定 */
  TMS = tms & 1; /* TMS ピンを設定 */
  TCK = 0;
  TCK = 1;
  tdo <<= 1;
  if (TDO) {
    tdo |= 1;
  }
  tdi >>= 1;
  tms >>= 1;
}
```

## __jtagRawWrite

**構文**　__jtagRawWrite(*tdi, tms, numbits*)

**パラメータ**

| | |
|---|---|
| *tdi* | TDI ピンに出力されるデータです。このデータは、最初に最下位ビットから送信されます。 |
| *tms* | TMS ピンに出力されるデータです。このデータは、最初に最下位ビットから送信されます。 |
| *numbits* | 転送ビット数です。各ビットは、JTAG TCK ラインの立ち下がりエッジ、および立ち上がりエッジとなります。最大値は 64 です。 |

**リターン値**　蓄積されたパケットのデータのビット位置を戻します。通常、この値は JTAG からデータを読み込むときに使用されます。

**適用範囲**　C-SPY J-Link/J-Trace ドライバ

**説明**

JTAG に転送されたビットを蓄積します。32 ビットで不十分な場合、この関数を何回も呼び出すことができます。両方のデータ出力ライン（TMS および TDI）は、別々に制御できます。

**例**

```
/*TMS の 1 ビットを 5 つ送信し、TAP-RESET 状態に移動する */
__jtagRawWrite(0x1F, 0, 5);    /* バッファにビットを格納 */
__jtagRawSync();  /* 転送バッファ、書込み tms、tdi 読込み tdo */
```

ARM ターゲットデバイスの JTAG ID を戻します。

## __jtagResetTRST

**構文**

```
__jtagResetTRST()
```

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | int 0 |
| 失敗 | ゼロ以外のエラー番号 |

*表 31: __jtagResetTRST のリターン値*

**適用範囲**

C-SPY J-Link/J-Trace ドライバ

**説明**

TRST JTAG 信号経由で ARM TAP コントローラをリセットします。

## __loadImage

**構文**

```
__loadImage(path, offset, debugInfoOnly)
```

**パラメータ**

*path*

ダウンロードするイメージのパスを識別する文字列。パスは絶対パスか、引数変数を使用する必要があります。引数変数について詳しくは、*『ARM 用 IDE プロジェクト管理およびビルドガイド』*を参照してください。

*offset*

ダウンロードされたイメージの目的地のアドレスのオフセットを識別する整数。

*debugInfoOnly*

ゼロ以外の値の場合、ターゲットシステムにコードやデータはダウンロードされません。すなわち、C-SPY はデバッグファイルからデバッグ情報を読み込むのみとなります。0 の場合、ダウンロードされます。

リターン値

| 値 | 結果 |
|---|---|
| ゼロ以外整数値 | 固有のモジュール識別子。 |
| int 0 | ロードに失敗しました。 |

*表 32: __loadImage のリターン値*

適用範囲

すべての C-SPY ドライバ。

説明

イメージ（デバッグファイル）をロードします。

**注 :** フラッシュロードは実行されません。[イメージ] オプションは、イメージを RAM にダウンロードするためだけに使用できます。

例 1

システムが ROM ライブラリとアプリケーションで構成されているとします。アプリケーションはアクティブプロジェクトですが、ライブラリに対応するデバッグファイルがあります。この場合には、C-SPY マクロファイルの execUserSetup マクロに以下のマクロ呼出しを追加し、これを自分のプロジェクトに関連付けます。

```
__loadImage(ROMfile, 0x8000, 1);
```

このマクロ呼出しは、ROM ライブラリ *ROMfile* のデバッグ情報をロードし、その内容はダウンロードしません（内容はすでに ROM にあると推定されるため）。すると、ライブラリと一緒にアプリケーションのデバッグが可能になります。

例 2

システムが ROM ライブラリとアプリケーションで構成されており、主にライブラリに心配があるとします。ライブラリは、デバッグセッションの前にフラッシュメモリにプログラミングする必要があります。したがって、ライブラリの開発中は、ライブラリプロジェクトが IDE でアクティブプロジェクトでなければなりません。この場合には、C-SPY マクロファイルの execUserSetup マクロに以下のマクロ呼出しを追加し、これを自分のプロジェクトに関連付けます。

```
__loadImage(ApplicationFile, 0x8000, 0);
```

このマクロ呼出しは、アプリケーションのデバッグ情報をロードし、その内容をダウンロードします（通常は RAM に）。すると、アプリケーションと一緒にライブラリのデバッグが可能になります。

関連項目

533 ページの*イメージ*、58 ページの*複数イメージのロード*。

## __memoryRestore

**構文**

```
__memoryRestore(zone, filename)
```

**パラメータ**

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

*filename*

読み込むファイルを指定する文字列。filename にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、*『ARM 用 IDE プロジェクト管理およびビルドガイド』* を参照してください。

**リターン値**

```
int 0
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

ファイルの内容を読み込んで、指定したメモリゾーンに保存します。

**例**

```
__memoryRestore("", "c:¥¥temp¥¥saved_memory.hex");
```

**関連項目**

185 ページの *［メモリリストア］ダイアログボックス*。

## __memorySave

**構文**

```
__memorySave(start, stop, format, filename)
```

**パラメータ**

*start*

保存するメモリエリアの最初の位置を指定する文字列。

*stop*

保存するメモリエリアの最後の位置を指定する文字列。

*format*

保存したメモリに使用されるフォーマットを指定する文字列。以下から選択します。

```
Intel-extended

motorola

motorola-s19
```

```
motorola-s28

motorola-s37.
```

*filename*

書き込むファイルを指定する文字列。filename にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、*『ARM 用 IDE プロジェクト管理およびビルドガイド』* を参照してください。

**リターン値**

```
int 0
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

指定したメモリエリアの内容をファイルに保存します。

**例**

```
__memorySave(":0x00", ":0xFF", "intel-extended",
"c:¥¥temp¥¥saved_memory.hex");
```

**関連項目**

184 ページの *［メモリセーブ］ダイアログボックス*。

## __messageBoxYesNo

**構文**

```
__messageBoxYesNo(string message, string caption)
```

**パラメータ**

*message*

メッセージボックスに表示されるメッセージ。

*caption*

メッセージボックスに表示されるタイトル。

**リターン値**

| 結果 | 値 |
|---|---|
| はい | 1 |
| いいえ | 0 |

*表 33: __messageBoxYesNo return values*

**適用範囲**

すべての C-SPY ドライバ。

**説明**

呼び出されたときに「はい / いいえ」のダイアログボックスを表示し、ユーザの入力値を返します。これは通常、ユーザとのやりとりを必要とするマクロを作成する際に役立ちます。

## __openFile

**構文**

```
__openFile(filename, access)
```

**パラメータ**

*filename*

開くファイル。filename にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、*『ARM 用 IDE プロジェクト管理およびビルドガイド』* を参照してください。

*access*

アクセスタイプ（文字列）。

以下は必須ですが、混在できません（互いに排他的）。

`"a"` 付加。開かれたファイルの最後に新しいデータが付加されます

`"r"` リード（テキストモードのデフォルト ; `b`（バイナリモード）と組み合わせる場合 : `rb`）

`"w"` ライト（テキストモードのデフォルト ; `b`（バイナリモード）と組み合わせる場合 : `wb`）

以下はオプションです。これらは混在できません（互いに排他的）。

`"b"` バイナリ。ファイルをバイナリモードで開きます

`"t"` ASCII テキスト。ファイルをテキストモードで開きます

以下のアクセスタイプはオプションです。

`"+"` と `r`、`w`、または `a`; `r+`、あるいは `w+` を併せて使用すると、*read* および *write* となり、`a+` は *read* と *append* となります

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | ファイルハンドル |
| 失敗 | 無効なファイルハンドル（評価結果は False） |

*表 34: __openFile のリターン値*

**適用範囲**

すべての C-SPY ドライバ。

**説明**

ファイルを入出力処理用に開きます。このマクロのデフォルトの基準ディレクトリは、開かれているプロジェクトファイル (`*.ewp`) のある場所になります。`__openFile` の引数では、このディレクトリとの相対位置を指定できます。また、`$PROJ_DIR$` や `$TOOLKIT_DIR$` などの引数変数をパスとして指定することもできます。

**例**

```
__var myFileHandle;          /* ファイルハンドルを保持する */
                             /* マクロ変数 */
myFileHandle = __openFile("$PROJ_DIR$¥¥Debug¥¥Exe¥¥test.tst",
"r");
if (myFileHandle)
{
    /* 正常に開きます */
}
```

**関連項目**

引数変数について詳しくは、*『ARM 用 IDE プロジェクト管理およびビルドガイド』*を参照してください。

## __orderInterrupt

**構文**

```
__orderInterrupt(specification, first_activation,
                 repeat_interval, variance, infinite_hold_time,
                 hold_time, probability)
```

**パラメータ**

*specification*

割込み（文字列）。specification には、デバイス記述ファイル (ddf) で使用されている仕様全体か、名前のみを指定できます。名前のみを指定した場合は、割込みシステムはデバイス記述ファイルから詳細を自動的に取得します。

*first_activation*

サイクル単位で指定した最初の実行時間（整数）。

*repeat_interval*

サイクル単位で指定した周期（整数）。

*variance*

パーセントで指定したタイミング変動範囲（0 ～ 100 の 整数）。

*infinite_hold_time*

無制限の場合は 1、それ以外の場合は 0。

*hold_time*

ホールド時間（整数）。

*probability*

パーセントで指定した確立（0 ～ 100 の 整数）。

**リターン値**

このマクロは、割込み識別子 (unsigned long) を返します。

*specification* の構文に誤りがある場合は、-1 を返します。

**適用範囲** C-SPY シミュレータ。

**説明** 割込みを生成します。

**例** 以下の例では、4000 サイクルの後に初めて実行された保持時間無制限の周期割込みを生成します。

```
__orderInterrupt( "IRQ", 4000, 2000, 0, 1, 0, 100 );
```

## __popSimulatorInterruptExecutingStack

**構文** `__popSimulatorInterruptExecutingStack(void)`

**リターン値** `int 0`

**適用範囲** C-SPY シミュレータ。

**説明** 割込みシミュレーションシステムに、割込みハンドラ実行から戻る際に使用される通常の命令と同様に、割込みハンドラの実行が終了したことを通知します。

割込みハンドラから復帰する際に通常の命令を使用しない場合（タスク切替え機能のあるオペレーティングシステムの場合など）に使用します。この場合、割込みシミュレーションシステムは割込みハンドラの実行が終了したことを自動的には検出できません。

**関連項目** 376 ページの *マルチタスクシステムでの割込みシミュレーション*。

## __readAPReg

**構文** `__readAPReg`（*レジスタ*）

**パラメータ**

| | |
|---|---|
| *register* | 8 ビットの AP レジスタオフセット。 |

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | true |
| 失敗 | false |

*表 35: __readAPReg のリターン値*

適用範囲

C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY TI Stellaris ドライバ

説明

現在選択されているアクセスポートの AP レジスタからのリード操作を実行します。

## __readDPReg

構文

```
__readDPReg（レジスタ）
```

パラメータ

*register* 8 ビットの DP レジスタオフセット。

リターン値

| 結果 | 値 |
|---|---|
| 成功 | true |
| 失敗 | false |

*表 36: __readDPReg リターン値*

適用範囲

C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY TI Stellaris ドライバ

説明

DP レジスタからのリード操作を実行します。

## __readFile

構文

```
__readFile(fileHandle, valuePtr)
```

パラメータ

*fileHandle*

__openFile マクロでファイルハンドルとして使用するマクロ変数。

*valuePtr*

変数へのポインタ。

リターン値

| 結果 | 値 |
|---|---|
| 成功 | 0 |
| 失敗 | ゼロ以外のエラー番号 |

*表 37: __readFile リターン値*

適用範囲　すべての C-SPY ドライバ。

説明　指定したファイルから 16 進数を順にリードし、unsigned long に変換して、*value* パラメータに代入します。この値は、マクロ変数へのポインタになります。

16 進数および空白文字を表す印刷可能な文字のみ使用できます。その他の文字は使用できません。

例

```
__var number;
if (__readFile(myFileHandle, &number) == 0)
{
  // 数字を処理します
}
```

この例では、myFileHandle によってポイントされるファイルに ASCII 文字の 1234 abcd 90ef が含まれる場合、連続した読取りによって値 0x1234 0xabcd 0x90ef が変数 number に割り当てられます。

## __readFileByte

構文　__readFileByte(*fileHandle*)

パラメータ

*fileHandle*

__openFile マクロでファイルハンドルとして使用するマクロ変数。

リターン値　エラー、ファイル終端時に -1 それ以外は、0 ～ 255。

適用範囲　すべての C-SPY ドライバ。

説明　file で指定したファイルから 1 バイトリードします。

例

```
__var byte;
while ( (byte = __readFileByte(myFileHandle)) != -1 )
{
  /* バイトを処理します */
}
```

## __readMemory8, __readMemoryByte

**構文**

```
__readMemory8(address, zone)
__readMemoryByte(address, zone)
```

**パラメータ**

*address*

メモリアドレス（整数）。

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

**リターン値**

このマクロは、メモリから取得した値を返します。

**適用範囲**

すべての C-SPY ドライバ。

**説明**

指定したメモリアドレス（ロケーション）から 1 バイトリードします。

**例**

```
__readMemory8(0x0108, "");
```

## __readMemory16

**構文**

```
__readMemory16(address, zone)
```

**パラメータ**

*address*

メモリアドレス（整数）。

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

**リターン値**

このマクロは、メモリから取得した値を返します。

**適用範囲**

すべての C-SPY ドライバ。

**説明**

指定したメモリアドレス（ロケーション）から 2 バイトワードをリードします。

**例**

```
__readMemory16(0x0108, "");
```

## __readMemory32

**構文**

```
__readMemory32(address, zone)
```

**パラメータ**

*address*

メモリアドレス（整数）。

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

**リターン値**

このマクロは、メモリから取得した値を返します。

**適用範囲**

すべての C-SPY ドライバ。

**説明**

指定したメモリアドレス（ロケーション）から 4 バイトワードをリードします。

**例**

```
__readMemory32(0x0108, "");
```

## __registerMacroFile

**構文**

```
__registerMacroFile(filename)
```

**パラメータ**

*filename*

登録するマクロが記述されたファイル（文字列）filename にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、*『ARM 用 IDE プロジェクト管理およびビルドガイド』*を参照してください。

**リターン値**

```
int 0
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

セットアップマクロファイルからマクロを登録します。この関数を使用して、複数のマクロファイルを C-SPY 起動時に登録できます。

**例**

```
__registerMacroFile("c:¥¥testdir¥¥macro.mac");
```

**関連項目**

396 ページの *C-SPY マクロの使用*。

## __resetFile

**構文** `__resetFile(fileHandle)`

**パラメータ** *fileHandle*

`__openFile` マクロでファイルハンドルとして使用するマクロ変数。

**リターン値** `int 0`

**適用範囲** すべての C-SPY ドライバ。

**説明** 先に `__openFile` で開いたファイルを先頭に戻します。

## __restoreSoftwareBreakpoints

**構文** `__restoreSoftwareBreakpoints()`

**リターン値** `int 0`

**適用範囲** C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY TI Stellaris ドライバ

C-SPY Macraigor ドライバ

**説明** システム起動中に破壊されたブレークポイントを自動的に復元します。

起動中に RAM にコピーしてから RAM で実行しているアプリケーションの場合に有効です。たとえば、リンカ構成ファイルのコードに `initialize by copy` ディレクティブを使用する場合、あるいはアプリケーションに `__ramfunc` 宣言関数がある場合に有効となることがあります。この場合、アプリケーションの実行が開始されると、すべてのブレークポイントは RAM のコピー中にオーバライドされます。

C-SPY はこのマクロを使用して、破棄されたブレークポイントを復元します。

## __selectCore

**構文** `__selectCore(int core)`

**パラメータ** *core*

切り替える先のコア。コアの番号は `0` から順に大きくなります。

**リターン値**

`int 0`

**適用範囲**

C-SPY シミュレータ

C-SPY I-jet/JTAGjet ドライバ

**説明**

マクロの呼出しの間、または次に `__selectCore` の呼出しがあるまで、現在のコアから指定したコアにフォーカスを切り替えます。

**例**

```
test ()
{
  __message "Core: ", __getSelectedCore(), " pc = ", #PC:%x, ¥n;
  __selectCore(0);
  __message "Core: ", __getSelectedCore(), " pc = ", #PC:%x, ¥n;
  __selectCore(1);
  __message "Core: ", __getSelectedCore(), " pc = ", #PC:%x, ¥n;
```

上記のマクロの一般的な結果は次のようになります（当初のコアの番号が `1` の場合）：

```
Core: 1 pc = 0000213C
Core: 0 pc = 00000494
Core: 1 pc = 0000213C
```

**関連項目**

421 ページの *__getSelectedCore*。

## __setCodeBreak

**構文**

`__setCodeBreak(`*`location, count, condition, cond_type, action`*`)`

**パラメータ**

*`location`*

ブレークポイントのコード位置を定義する文字列。有効な C-SPY 式で、値が有効なアドレスや絶対アドレス、ソース位置に評価されるもののいずれかです。位置タイプの詳細については、168 ページの *[位置入力] ダイアログボックス*を参照してください。

*`count`*

実行を停止するまでのブレークポイント条件発生回数（整数）。

*`condition`*

ブレークポイント条件（文字列）。

*`cond_type`*

条件の種類。`"CHANGED"` または `"TRUE"`（文字列）。

*action*

ブレークポイント検出時に評価される式（通常はマクロ呼出し）。

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | ブレークポイントを一意に特定する符号なし整数。ブレークポイントを消去する際には、この値を使用する必要があります。 |
| 失敗 | 0 |

*表 38: __setCodeBreak のリターン値*

**適用範囲**

C-SPY ハードウェアデバッガドライバ。

**説明**

コードブレークポイント（プロセッサが指定位置で命令をフェッチする直前にトリガされるブレークポイント）を設定します。

**例**

```
__setCodeBreak("{D:¥¥src¥¥prog.c}.12.9", 3, "d>16", "TRUE",
"ActionCode()");
```

以下の例は、ソース中の main というラベルにコードブレークポイントを設定します。

```
__setCodeBreak("main", 0, "1", "TRUE", "");
```

**関連項目**

133 ページの*ブレークポイント*。

## __setDataBreak

**構文**

シミュレータ：

```
__setDataBreak(location, count, condition, cond_type, access,
               action)
```

I-jet/JTAGjet ドライバの場合：

```
__setDataBreak(location, access, extend, match, data, mask)
```

**パラメータ**

*location*

ブレークポイントのデータ位置を定義する文字列。有効な C-SPY 式で、値が有効なアドレスまたは絶対アドレスに評価されるもののいずれかです。位置タイプの詳細については、168 ページの*［位置入力］ダイアログボックス*を参照してください。

*count*

実行を停止するまでのブレークポイント条件発生回数（整数）。

このパラメータはシミュレータのみに適用されます。

*condition*

ブレークポイント条件（文字列）。

このパラメータはシミュレータのみに適用されます。

*cond_type*

条件の種類。`"CHANGED"` または `"TRUE"`（文字列）。

このパラメータはシミュレータのみに適用されます。

*access*

メモリアクセスタイプ : リードの場合は `"R"`、ライトの場合は `"W"`、リード / ライトの場合は `"RW"`。

*action*

ブレークポイント検出時に評価される式（通常はマクロ呼出し）。

このパラメータはシミュレータのみに適用されます。

*extend*

データ構造体がカバーされるようにブレークポイントを拡張します。ハードウェアブレークポイント装置で提供できるブレークポイント範囲のサイズと合わないデータ構造（たとえば 3 バイト）の場合、ブレークポイントの範囲はデータ構造全体を対象としません。ブレークポイントの範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。`"TRUE"` か `"FALSE"` のどちらかを選択します。

このパラメータは I-jet/JTAGjet のみに適用されます。

*match*

アクセスされるデータの照合を有効にします。`"TRUE"` か `"FALSE"` のどちらかを選択します。

このパラメータは I-jet/JTAGjet のみに適用されます。

*data*

照合するデータ値（符号なしの 32 ビット形式）。

このパラメータは I-jet/JTAGjet のみに適用されます。

*mask*

データ値のどの部分（ワード、ハーフワード、バイト）を照合するかを、符号なしの 32 ビット形式で指定します。

このパラメータは I-jet/JTAGjet のみに適用されます。

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | ブレークポイントを一意に特定する符号なし整数。ブレークポイントを消去する際には、この値を使用する必要があります。 |
| 失敗 | 0 |

*表 39: __setDataBreak のリターン値*

**適用範囲**

I-jet/JTAGjet ドライバ。

**説明**

データブレークポイント（プロセッサが指定位置でデータのリード / ライトを実行した直後にトリガされるブレークポイント）を設定します。

**例**

C-SPY シミュレータ：

```
__var brk;
brk = __setDataBreak(":0x4710", 3, "d>6", "TRUE",
      "W", "ActionData()");
...
__clearBreak(brk);
```

**関連項目**

133 ページの *ブレークポイント*。

## __setDataLogBreak

**構文**

```
__setDataLogBreak(location, access, extend)
```

**パラメータ**

*location*

ブレークポイントのデータ位置を定義する文字列。有効な C-SPY 式で、値が有効なアドレスまたは絶対アドレスに評価されるもののいずれかです。位置タイプの詳細については、168 ページの *［位置入力］ダイアログボックス*を参照してください。

*access*

メモリアクセスタイプ：リードの場合は `"R"`、ライトの場合は `"W"`、リード / ライトの場合は `"RW"`。

*extend*

データ構造体がカバーされるようにブレークポイントを拡張します。ハードウェアブレークポイント装置で提供できるブレークポイント範囲のサイズと合わないデータ構造（たとえば 3 バイト）の場合、ブレークポイントの範囲はデータ構造全体を対象としません。ブレークポイントの範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。`"TRUE"` か `"FALSE"` のどちらかを選択します。

使用する C-SPY ドライバによっては、このパラメータがサポートされていない場合がある点に注意してください。

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | ブレークポイントを一意に特定する符号なし整数。ブレークポイントを消去する際には、この値を使用する必要があります。 |
| 失敗 | 0 |

*表 40: __setDataLogBreak のリターン値*

**適用範囲**

すべての C-SPY ドライバ

C-SPY I-jet/JTAGjet ドライバ。

**説明**

データログブレークポイント（プロセッサが指定位置でデータのリード / ライトを実行したときにトリガされるブレークポイント）を設定します。データログブレークポイントは実行を停止せず、データログを生成するだけです。

**例**

```
__var brk;
brk = __setDataLogBreak("Memory:0x4710", "R", "TRUE");
...
__clearBreak(brk);
```

**関連項目**

133 ページの *ブレークポイント*、105 ページの *データログを開始するには*。

## __setLogBreak

**構文**

```
__setLogBreak(location, message, msg_type, condition,
              cond_type)
```

**パラメータ**

*location*

ブレークポイントのコード位置を定義する文字列。有効な C-SPY 式で、値が有効なアドレスや絶対アドレス、ソース位置に評価されるもののいずれかです。位置タイプの詳細については、168 ページの*［位置入力］ダイアログボックス*を参照してください。

*message*

メッセージテキスト。

*msg_type*

以下のメッセージタイプのどちらかを選択します。

TEXT：メッセージがそのまま書き込まれます。

ARGS：メッセージは、C-SPY の式または文字列をコンマで区切ったリストとして認識されます。

*condition*

ブレークポイント条件（文字列）。

*cond_type*

条件の種類。"CHANGED" または "TRUE"（文字列）。

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | ブレークポイントを一意に特定する符号なし整数。ブレークポイントを削除する際には、同じ値を使用する必要があります。 |
| 失敗 | 0 |

*表 41: __setLogBreak のリターン値*

**適用範囲**

C-SPY ハードウェアデバッガドライバ。

**説明**

ログブレークポイントを設定します。つまり、命令が指定の場所からフェッチされたときにトリガされるブレークポイントです。特定のマシン命令にブレークポイントを設定した場合は、命令の実行前にブレークポイントがトリガされ、実行が一時停止し、指定したメッセージが［C-SPY デバッグログ］ウィンドウに出力されます。

例

```
__var logBp1;
__var logBp2;

logOn()
{
  logBp1 = __setLogBreak ("{C:¥¥temp¥¥Utilities.c}.23.1",
    "¥"Entering trace zone at :¥", #PC:%X", "ARGS", "1", "TRUE");
  logBp2 = __setLogBreak ("{C:¥¥temp¥¥Utilities.c}.30.1",
    "Leaving trace zone...", "TEXT", "1", "TRUE");
}

logOff()
{
  __clearBreak(logBp1);
  __clearBreak(logBp2);
}
```

関連項目

405 ページの *フォーマットした出力*、133 ページの *ブレークポイント*。

## __setSimBreak

構文

```
__setSimBreak(location, access, action)
```

パラメータ

*location*

ブレークポイントのデータ位置を定義する文字列。有効な C-SPY 式で、値が有効なアドレスまたは絶対アドレスに評価されるもののいずれかです。位置タイプの詳細については、168 ページの *［位置入力］ダイアログボックス*を参照してください。

*access*

メモリアクセスタイプ : リードの場合は "R"、ライトの場合は "W"。

*action*

ブレークポイント検出時に評価される式（通常はマクロ呼出し）。

リターン値

| 結果 | 値 |
|---|---|
| 成功 | ブレークポイントを一意に特定する符号なし整数。ブレークポイントを消去する際には、この値を使用する必要があります。 |
| 失敗 | 0 |

*表 42: __setSimBreak のリターン値*

適用範囲

C-SPY シミュレータ。

**説明**

このシステムマクロを使用して、一時的にのみ命令の実行を停止するイミディエイトブレークポイントを設定します。このブレークポイントを使用すると、プロセッサがある位置からデータを読み込む直前かある位置にデータを書き込んだ直後に、C-SPY マクロ関数を呼び出すことができます。アクションが終了すると、命令の実行が再開されます。

イミディエイトブレークポイントは、メモリにマッピングされたさまざまな種類のデバイス（シリアルポートやタイマなど）をシミュレーションする場合に便利です。デバイスがメモリマッピングされた位置をプロセッサが読み込むと、C-SPY マクロ関数が実行されて適切なデータを供給します。逆に、デバイスがメモリマッピングされた位置にプロセッサが書き込むと、C-SPY マクロ関数が実行されて、書き込まれた値に応じた適切な動作を実行します。

## __setTraceStartBreak

**構文**

シミュレータの場合：

```
__setTraceStartBreak(location)
```

I-jet/JTAGjet ドライバの場合：

```
__setTraceStartBreak(location, access, extend, match, data,
mask)
```

**パラメータ**

*location*

ブレークポイントのコード位置を定義する文字列。有効な C-SPY 式で、値が有効なアドレスや絶対アドレス、ソース位置に評価されるもののいずれかです。位置タイプの詳細については、168 ページの *[位置入力] ダイアログボックス*を参照してください。

*access*

メモリアクセスのタイプ：`"F"`（フェッチ）、`"R"`（リード）、`"W"`（ライト）、`"RW"`（リード / ライト）。

このパラメータは I-jet/JTAGjet のみに該当します。

*extend*

データ構造全体をカバーするようにブレークポイントを拡張します。ハードウェアブレークポイントユニットで指定されるブレークポイント範囲のサイズに合わないデータ構造（3 バイトなど）の場合、ブレークポイント範囲はデータ構造全体をカバーしきれません。データ構造のサイズよりも大きくブレークポイントが拡張されますが、これによって隣接するデータで誤ったトリガが発生することがあります。`"TRUE"` または `"FALSE"` のどちらかを選択してください。

このパラメータは I-jet/JTAGjet のみに該当します。

*match*

アクセスされるデータのマッチングを可能にします。"TRUE" または "FALSE" のどちらかを選択してください。

このパラメータは I-jet/JTAGjet のみに該当します。

*data*

マッチングするデータ値（符号なしの 32 ビット形式)。

このパラメータは I-jet/JTAGjet のみに該当します。

*mask*

データ値のどの部分をマッチングするかを符号なしの 32 ビット形式で指定します（ワード、ハーフワード、バイト)。

このパラメータは I-jet/JTAGjet のみに該当します。

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | ブレークポイントを一意に特定する符号なし整数。ブレークポイントを削除する際には、同じ値を使用する必要があります。 |
| 失敗 | 0 |

*表 43: __setTraceStartBreak のリターン値*

**適用範囲**

C-SPY シミュレータ。

C-SPY I-jet/JTAGjet ドライバ。

**説明**

指定の位置にブレークポイントを設定します。そのブレークポイントがトリガされると、トレースシステムが起動します。

**例**

```
__var startTraceBp;
__var stopTraceBp;

traceOn()
{
  startTraceBp = __setTraceStartBreak
    ("{C:¥¥TEMP¥¥Utilities.c}.23.1");
  stopTraceBp = __setTraceStopBreak
    ("{C:¥¥temp¥¥Utilities.c}.30.1");
}

traceOff()
{
  __clearBreak(startTraceBp);
  __clearBreak(stopTraceBp);
}
```

**関連項目**

133 ページの*ブレークポイント*。

## __setTraceStopBreak

**構文**

シミュレータの場合 :

```
__setTraceStopBreak(location)
```

I-jet/JTAGjet ドライバの場合 :

```
__setTraceStopBreak(location, access, extend, match, data, mask)
```

**パラメータ**

*location*

ブレークポイントのコード位置を定義する文字列。有効な C-SPY 式で、値が有効なアドレスや絶対アドレス、ソース位置に評価されるもののいずれかです。位置タイプの詳細については、168 ページの*［位置入力］ダイアログボックス*を参照してください。

*access*

メモリアクセスのタイプ : `"F"`（フェッチ）、`"R"`（リード）、`"W"`（ライト）、`"RW"`（リード / ライト）。

このパラメータは I-jet/JTAGjet のみに該当します。

*extend*

データ構造全体をカバーするようにブレークポイントを拡張します。ハードウェアブレークポイントユニットで指定されるブレークポイント範囲のサイズに合わないデータ構造（3 バイトなど）の場合、ブレークポイント範囲はデータ構造全体をカバーしきれません。データ構造のサイズよりも大きくブレークポイントが拡張されますが、これによって隣接するデータで誤ったトリガが発生することがあります。`"TRUE"` または `"FALSE"` のどちらかを選択してください。

このパラメータは I-jet/JTAGjet のみに該当します。

*match*

アクセスされるデータのマッチングを可能にします。`"TRUE"` または `"FALSE"` のどちらかを選択してください。

このパラメータは I-jet/JTAGjet のみに該当します。

*data*

マッチングするデータ値（符号なしの 32 ビット形式)。

このパラメータは I-jet/JTAGjet のみに該当します。

*mask*

データ値のどの部分をマッチングするかを符号なしの 32 ビット形式で指定します（ワード、ハーフワード、バイト)。

このパラメータは I-jet/JTAGjet のみに該当します。

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | ブレークポイントを一意に特定する符号なし整数。ブレークポイントを削除する際には、同じ値を使用する必要があります。 |
| 失敗 | `int 0` |

*表 44: __setTraceStopBreak のリターン値*

**適用範囲**

C-SPY シミュレータ。

C-SPY I-jet/JTAGjet ドライバ。

**説明**

指定の位置にブレークポイントを設定します。そのブレークポイントがトリガされると、トレースシステムが停止します。

**例**

451 ページの *__setTraceStartBreak* を参照してください。

**関連項目**

133 ページの *ブレークポイント*。

## __sourcePosition

**構文**

```
__sourcePosition(linePtr, colPtr)
```

**パラメータ**

*linePtr*

行番号を格納する変数へのポインタ。

*colPtr*

列番号を格納する変数へのポインタ。

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | ファイル名文字列 |
| 失敗 | 空き ("") 文字列 |

*表 45: __sourcePosition リターン値*

**適用範囲**

すべての C-SPY ドライバ。

**説明**

現在の実行位置がソース位置に対応する場合、このマクロはファイル名を文字列として返します。パラメータで参照する変数の値を、ソース位置の行番号と列番号に設定します。

## __strFind

**構文**

```
__strFind(macroString, pattern, position)
```

**パラメータ**

*macroString*

マクロ文字列。

*pattern*

検索対象の文字列パターン。

*position*

検索の開始位置。最初の位置は `0` です。

**リターン値**

パターンが見つかった位置。文字列が見つからなかった場合は `-1`。

**適用範囲**

すべての C-SPY ドライバ。

**説明**

このマクロは指定した文字列 (*macroString*) で別の文字列 (*pattern*) を検索します。

**例**

```
__strFind("Compiler", "pile", 0)  = 3
__strFind("Compiler", "foo", 0)   = -1
```

**関連項目**

403 ページの *マクロ文字列*。

## __subString

**構文**

```
__subString(macroString, position, length)
```

**パラメータ**

*macroString*

マクロ文字列。

*position*

部分文字列の開始位置。最初の位置は 0 です。

*length*

部分文字列の長さ。

**リターン値**

指定したマクロ文字列から抽出した部分文字列。

**適用範囲**

すべての C-SPY ドライバ。

**説明**

このマクロは、文字列 (*macroString*) から部分文字列を抽出します。

**例**

```
__subString("Compiler", 0, 2)
```

生成されたマクロ文字列に `Co` が含まれます。

```
__subString("Compiler", 3, 4)
```

生成されたマクロ文字列に `pile` が含まれます。

**関連項目**

403 ページの *マクロ文字列*。

## __targetDebuggerVersion

**構文**

```
__targetDebuggerVersion()
```

**リターン値**

C-SPY デバッガのプロセッサモジュールのバージョン番号を表す文字列。

**適用範囲**

すべての C-SPY ドライバ。

**説明**

このマクロは、C-SPY デバッガのプロセッサモジュールのバージョン番号を返します。

例

```
__var toolVer;
toolVer = __targetDebuggerVersion();
__message "The target debugger version is, ", toolVer;
```

## __toLower

構文

```
__toLower(macroString)
```

パラメータ

*macroString*
マクロ文字列。

リターン値

変換後のマクロ文字列。

適用範囲

すべての C-SPY ドライバ。

説明

このマクロは、すべての文字を小文字に変換して、パラメータ *macroString* のコピーを返します。

例

```
__toLower("IAR")
```

生成されたマクロ文字列に `iar` が含まれます。

```
__toLower("Mix42")
```

生成されたマクロ文字列 `mix42` が含まれます。

関連項目

403 ページの *マクロ文字列*。

## __toString

構文

```
__toString(C_string, maxlength)
```

パラメータ

*C_string*
NULL 終端 C 文字列。

*maxlength*
返されるマクロ文字列の最大長。

リターン値

マクロ文字列。

適用範囲

すべての C-SPY ドライバ。

**説明** このマクロを使用して、C 文字列（char* または char[]）をマクロ文字列に変換します。

**例** アプリケーションに以下の定義が含まれていると仮定します。

```
char const * hptr = "Hello World!";
```

以下のマクロ呼出しを使用します。

```
__toString(hptr, 5)
```

Hello を含むマクロ文字列を返します。

**関連項目** 403 ページの *マクロ文字列*。

## __toUpper

**構文** `__toUpper(`*macroString*`)`

**パラメータ**

*macroString*
マクロ文字列。

**リターン値** 変換後の文字列。

**適用範囲** すべての C-SPY ドライバ。

**説明** このマクロは、すべての文字を大文字に変換して、パラメータ *macroString* のコピーを返します。

**例**

```
__toUpper("string")
```

生成されたマクロ文字列に STRING が含まれます。

**関連項目** 403 ページの *マクロ文字列*。

## __unloadImage

**構文** `__unloadImage(`*module_id*`)`

**パラメータ**

*module_id*
固有なモジュール ID を表す整数。対応する __loadImage C-SPY マクロからのリターン値として取得されます。

リターン値

| 値 | 結果 |
|---|---|
| *module_id* | 固有のモジュール ID（入力パラメータと同じ） |
| int 0 | アンロードが失敗 |

*表 46: __unloadImage のリターン値*

適用範囲　すべての C-SPY ドライバ。

説明　すでにダウンロード済みのイメージからデバッグ情報をアンロードします。

関連項目　58 ページの *複数イメージのロード*、533 ページの *イメージ*。

## __writeAPReg

構文　`__writeAPReg`（*データ*、*レジスタ*）

パラメータ

| | |
|---|---|
| *data* | 32 ビット値。 |
| *register* | 8 ビットの AP レジスタオフセット。 |

リターン値

| 結果 | 値 |
|---|---|
| 成功 | true |
| 失敗 | false |

*表 47: __writeAPReg のリターン値*

適用範囲　C-SPY I-Jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY TI Stellaris ドライバ

説明　現在選択されているアクセスポートの AP レジスタへのライト操作を実行します。

## __writeDPReg

**構文**

```
__writeDPReg（データ、レジスタ）
```

**パラメータ**

| | |
|---|---|
| *data* | 32 ビット値。 |
| *register* | 8 ビットの DP レジスタオフセット。 |

**リターン値**

| 結果 | 値 |
|---|---|
| 成功 | true |
| 失敗 | false |

*表 48: __writeDPReg のリターン値*

**適用範囲**

C-SPY I-Jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY TI Stellaris ドライバ

**説明**

DP レジスタへのライト操作を実行します。

**例**

```
__writeDPReg(0x010000F0、0x8)
  /* アクセスポート 1 とバンク 15 を選択 */
```

## __writeFile

**構文**

```
__writeFile(fileHandle, value)
```

**パラメータ**

*fileHandle*

__openFile マクロでファイルハンドルとして使用するマクロ変数。

*value*

整数。

**リターン値**

```
int 0
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

整数値を 16 進数フォーマット（後の空白文字を含む）でファイル *file* に出力します。

**注：** __fmessage 文でも同様の操作を実行できます。__writeFile マクロは、__readFile との対称性の観点から提供されています。

## __writeFileByte

**構文**

```
__writeFileByte(fileHandle, value)
```

**パラメータ**

*fileHandle*

__openFile マクロでファイルハンドルとして使用するマクロ変数。

*value*

整数。

**リターン値**

```
int 0
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

*fileHandle* ファイルに 1 バイト書き込みます。

## __writeMemory8, __writeMemoryByte

**構文**

```
__writeMemory8(value, address, zone)
__writeMemoryByte(value, address, zone)
```

**パラメータ**

*value*

整数。

*address*

メモリアドレス（整数）。

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

**リターン値**

```
int 0
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

指定したメモリアドレス（ロケーション）に 1 バイトライトします。

**例**

```
__writeMemory8(0x2F, 0x8020, "");
```

## __writeMemory16

**構文**

```
__writeMemory16(value, address, zone)
```

**パラメータ**

*value*

整数。

*address*

メモリアドレス（整数）。

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

**リターン値**

```
int 0
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

指定したメモリアドレス（ロケーション）に 2 バイトライトします。

**例**

```
__writeMemory16(0x2FFF, 0x8020, "");
```

## __writeMemory32

**構文**

```
__writeMemory32(value, address, zone)
```

**パラメータ**

*value*

整数。

*address*

メモリアドレス（整数）。

*zone*

メモリ範囲を指定する文字列（174 ページの *C-SPY メモリゾーン*を参照）。

**リターン値**

```
int 0
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

指定したメモリアドレス（ロケーション）に 4 バイトライトします。

**例**

```
__writeMemory32(0x5555FFFF, 0x8020, "");
```

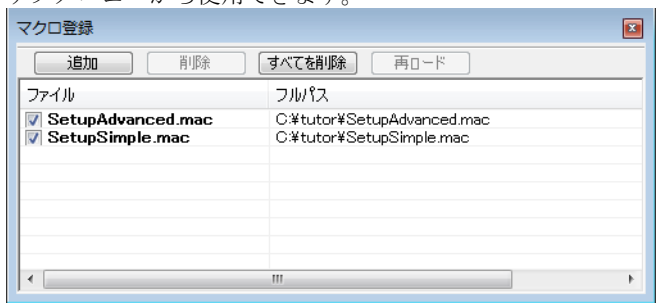
# マクロのグラフィカル環境

リファレンス情報：

- 463 ページの *マクロ登録ウィンドウ*
- 465 ページの *［デバッグマクロ］ウィンドウ*
- 466 ページの *［マクロクイック起動ウィンドウ］*

## マクロ登録ウィンドウ

［マクロ登録］ウィンドウは、デバッグセッション中に **［表示］> ［マクロ］** サブメニューから使用できます。




このウィンドウを使用して、デバッグマクロファイルの一覧表示、登録、編集が可能です。

エディタウィンドウでマクロファイルをダブルクリックして開き、編集します。

### 要件

ありません。このウィンドウは常に使用できます。

### 表示エリア

このエリアには以下の列が含まれます。

**ファイル**

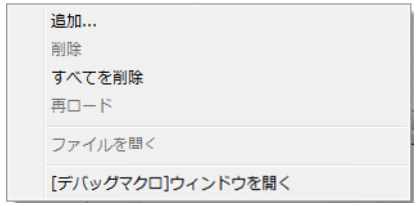
使用可能なマクロファイルの名前。マクロファイルを登録するには、ファイル名の左にあるチェックボックスを選択します。登録されたマクロファイルの名前が太字で表示されます。

**フルパス**

追加されたマクロファイルのパス。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**追加**

ファイルブラウザが開き、リストに追加するマクロファイルを探すことができます。このメニューコマンドは、ウィンドウ上部の機能ボタンとしても利用できます。

**削除**

選択されたデバッグマクロファイルをリストから削除します。このメニューコマンドは、ウィンドウ上部の機能ボタンとしても利用できます。

**すべてを削除**

すべてのマクロファイルをリストから削除します。このメニューコマンドは、ウィンドウ上部の機能ボタンとしても利用できます。

**再ロード**

選択したマクロファイルを登録します。通常、マクロファイルを編集し終わったときに便利です。このメニューコマンドは、ウィンドウ上部の機能ボタンとしても利用できます。

**ファイルを開く**

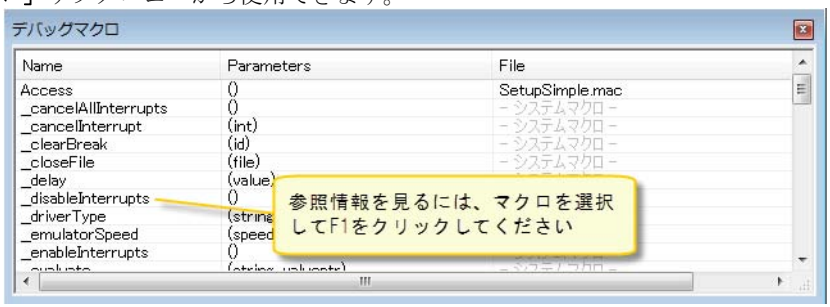
選択したマクロファイルがエディタウィンドウで開きます。

**[デバッグマクロ] ウィンドウを開く**

[デバッグマクロ] ウィンドウが開きます。

# ［デバッグマクロ］ウィンドウ

［デバッグマクロ］ウィンドウは、デバッグセッション中に［表示］>［マクロ］サブメニューから使用できます。




このウィンドウを使用して、定義済システムマクロあるいは独自のマクロを問わず、登録したデバッグマクロ関数をすべて一覧表示します。このウィンドウは、独自のマクロ関数を編集し、使用可能なすべてのマクロの概要が必要なときに役立ちます。

ファイル内に定義されたマクロをダブルクリックすると、そのファイルがエディタウィンドウで開きます。

［マクロクイック起動］ウィンドウでマクロを開くには、マクロを［デバッガマクロ］ウィンドウから［マクロクイック起動］ウィンドウにドラッグ & ドロップします。

マクロを選択して F1 を押すと、そのマクロのオンラインヘルプ情報を入手できます。

### 要件

ありません。このウィンドウは常に使用できます。

### 表示エリア

このエリアには以下の列が含まれます。

**名前**

デバッグマクロの名前。

**パラメータ**

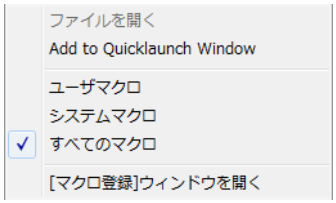
デバッグマクロのパラメータ。

**ファイル**

ファイル内に定義されたマクロの場合、ファイル名が表示されます。定義済システムマクロの場合、「- システムマクロ -」と表示されます。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**ファイルを開く**

選択したデバッグマクロファイルがエディタウィンドウで開きます。

**［クイック起動に追加］ウィンドウ**

選択したマクロを［マクロクイック起動］ウィンドウに追加します。

**ユーザマクロ**

自分で定義したデバッグマクロのみがリストされます。

**システムマクロ**

定義済システムマクロのみがリストされます。

**すべてのマクロ**

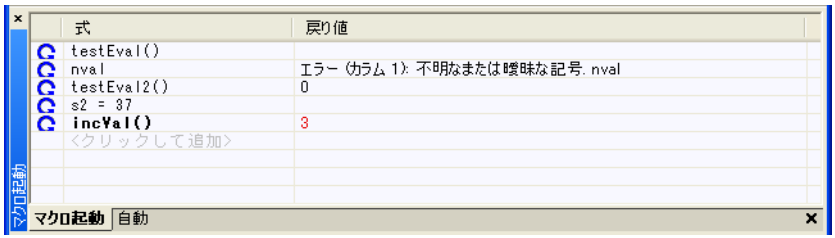
定義済のシステムマクロと自分で定義したマクロの両方を含む、すべてのデバッグマクロがリストされます。

**［マクロ登録］ウィンドウを開く**

［マクロ登録］ウィンドウが開きます。

## ［マクロクイック起動ウィンドウ］

［マクロクイック起動］ウィンドウは**［表示］**メニューから利用できます。




このウィンドウを使用して、通常は C-SPY マクロの式を評価します。

一部のデバイスでは、デバイスサポート付きの事前定義済み C-SPY マクロが用意されており、通常これらはチップメーカが提供します。これらのマクロは、デバイスに固有な特定のタスクを実行する際に便利です。これらのマクロは［マクロクイック起動］ウィンドウから利用でき、緑のアイコンで簡単に区別できます。

［マクロクイック起動］ウィンドウは［クイックウォッチ］ウィンドウに似ていますが、これは主に C-SPY マクロの評価用に設計されています。このウィンドウでは、式を評価するタイミングを正確に制御することができます。

**式を追加するには以下の手順に従います。**

1. 以下のいずれかの方法を選択します。
   - 式をウィンドウにドラッグ
   - **［式］**列に確認する式を入力します。

   追加して評価する式が C-SPY マクロの場合、このマクロをまず登録する必要があります（396 ページの *C-SPY マクロの使用*を参照）。

**式を評価するには、以下の手順に従います。**

1. **［再計算］**アイコンをダブルクリックすると、式の値が計算されます。

### 要件

ありません。このウィンドウは常に使用できます。

### 表示エリア

このエリアには以下の列が含まれます。

**再計算アイコン**

式を評価するには、このアイコンをダブルクリックします。最新の評価された式が太字で表示されます。

**式**

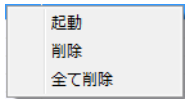
評価する 1 つまたは複数の式。式を追加するには、< クリックして追加 > をクリックします。リターン値が前回から変わっている場合、その値は赤で表示されます。

**結果**

式の評価からのリターン値が表示されます。

### コンテキストメニュー

以下のコンテキストメニューがあります。




以下のコマンドがあります。

**今すぐ評価**

選択した式を評価します。

**削除**

選択した式を削除します。

**すべてを削除**

選択した式をすべて削除します。

# C-SPY コマンドラインユーティリティ — cspybat

- C-SPY をバッチモードで使用
- C-SPY コマンドラインオプションの概要
- C-SPY コマンドラインオプションについてのリファレンス情報

## C-SPY をバッチモードで使用

以下のトピックについて説明します:

- cspybat の起動
- 出力
- 呼出し構文

コマンドラインユーティリティの cspybat を使用すると、C-SPY をバッチモードで実行できます。このユーティリティは、common¥bin ディレクトリにインストールされています。

### CSPYBAT の起動

cspybat を起動するには、まずバッチファイルを作成する必要があります。その簡単な方法は、IDE で C-SPY を起動する際に C-SPY が自動的に生成するバッチファイルのいずれかを使用することです。

C-SPY が初期化されるたびに、*projectname.buildconfiguration*.cspy.bat というバッチファイルが生成されます。また、あと 2 つのファイルが生成されます。

- *project.buildconfiguration*.general.xcl には、cspybat に固有のオプションが含まれます。
- *project.buildconfiguration*.driver.xcl には、使用する C-SPY ドライバに固有のオプションが含まれます。

このファイルは、$PROJ_DIR$¥settings ディレクトリに保存されています。これらのファイルには IDE と同じ設定が含まれており、使用可能なオプションについてのヒントになります。

**2** cspybat を起動するには、次のコマンドラインを使用できます。

```
project.cspybat.bat [debugfile]
```

*debugfile* は任意である点に注意してください。*project.buildconfiguration*.general.xcl ファイルで使用されているものとは異なるデバッグファイルを使用する場合に、それを指定することができます。

### 出力

cspybat の実行時、以下のタイプの出力を生成できます。

- cspybat 自身からのターミナル出力

  このターミナル出力はすべて stderr に転送されます。コマンドラインから引数なしで cspybat を実行する場合、cspybat のバージョン番号と利用可能なすべてのオプション（簡単な説明を含む）が stdout に転送され、画面に表示されます。
- デバッグ対象アプリケーションからのターミナル出力

  こうしたすべてのターミナル出力先は、stdout になります。ただし、--plugin オプションを使用していることが条件です。514 ページの *--plugin* を参照してください。
- エラーリターンコード

  cspybat は、バッチファイル内で評価可能なステータス情報をホストオペレーティングシステムに返します。*成功*の場合は値 int 0 が、*失敗*の場合は値 int 1 がそれぞれ返されます。

### 呼出し構文

cspybat の呼出し構文は以下のとおりです。

```
cspybat processor_DLL driver_DLL debug_file
        [cspybat_options] --backend driver_options
```

**注 :** ファイル名（DLL ファイルも含む）が要求される場合には、ファイル名のフルパスを指定することが推奨されます。

### パラメータ

パラメータを以下に示します。

| パラメータ | 説明 |
|---|---|
| *processor_DLL* | プロセッサ固有の DLL ファイルです。arm¥bin にあります。 |
| *driver_DLL* | C-SPY ドライバ DLL ファイルです。arm¥bin にあります。 |

*表 49: cspybat のパラメータ*

| パラメータ | 説明 |
|---|---|
| *debug_file* | デバッグ対象のオブジェクトファイルです（ファイル名拡張子 `out`）。480 ページの *–debugfile* も参照してください。 |
| *cspybat_options* | `cspybat` に渡すコマンドラインオプションです。これらのオプションは省略可能です。それぞれのオプションについては、478 ページの *C-SPY コマンドラインオプションについてのリファレンス情報*を参照してください。 |
| `--backend` | C-SPY ドライバに送信するパラメータの開始を示します。後に続くすべてのオプションがドライバに渡されます。このオプションは必須です。 |
| *driver_options* | C-SPY ドライバに渡すコマンドラインオプションです。これらのオプションには、必須のものと省略可能なものがあります。それぞれのオプションについては、478 ページの *C-SPY コマンドラインオプションについてのリファレンス情報*を参照してください。 |

*表 49: cspybat のパラメータ（続き）*

# C-SPY コマンドラインオプションの概要

リファレンス情報 :

- cspybat の一般オプション
- すべての C-SPY ドライバで使用可能なオプション
- シミュレータドライバで使用可能なオプション
- C-SPY Angel デバッグモニタドライバで使用可能なオプション
- C-SPY GDB サーバドライバで使用可能なオプション
- C-SPY IAR ROM-monitor ドライバで使用可能なオプション
- C-SPY I-jet/JTAGjet ドライバで使用可能なオプション
- C-SPY CMSIS-DAP ドライバで使用可能なオプション
- C-SPY J-Link/J-Trace ドライバで使用可能なオプション
- C-SPY TI Stellaris ドライバで使用可能なオプション
- C-SPY TI XDS ドライバで使用可能なオプション
- C-SPY Macraigor ドライバで使用可能なオプション
- C-SPY RDI ドライバで使用可能なオプション
- C-SPY ST-LINK ドライバで使用可能なオプション
- C-SPY サードパーティ製ドライバで使用可能なオプション

## CSPYBAT の一般オプション

| | |
|---|---|
| --backend | C-SPY ドライバに送信するパラメータの開始を示します（必須）。 |
| --code_coverage_file | コードカバレッジ情報の生成を可能にして、それを指定ファイルに記録します。 |
| --cycles | 実行するサイクルの最大回数を指定します。 |
| --debugfile | 別のデバッグファイルを指定します。 |
| --download_only | コードイメージを後でデバッグセッションを開始せずにダウンロードします。 |
| -f | コマンドラインを拡張します。 |
| --flash_loader | フラッシュローダ仕様 XML ファイルを指定します。 |
| --leave_running | ターゲット上で実行を開始し、終了した後もターゲットを実行したままにします。 |
| --macro | 使用するマクロファイルを指定します。 |
| --macro_param | C-SPY マクロパラメータに値を割り当てます。 |
| --plugin | 使用するプラグインファイルを指定します。 |
| --rtc_enable | cspybat で C-RUN のランタイムエラーチェックを有効にします。 |
| --rtc_output | cspybat に対して、C-RUN メッセージ出力用のファイルを指定します。 |
| --rtc_raw_to_txt | ファイルを入力として読み込むことにより、cspybat がランタイムチェックのメッセージフィルタとして機能します。 |
| --rtc_rules | cspybat に対して、C-RUN ルール用のファイルを指定します。 |
| --silent | サインオンメッセージを省略します。 |
| --timeout | 最長実行時間を設定します。 |

## すべての C-SPY ドライバで使用可能なオプション

| | |
|---|---|
| -B | バッチモードを有効化します（必須）。 |

| | |
|---|---|
| `--BE8` | ビッグエンディアンフォーマット BE8 を使用します。リファレンス情報については、*『ARM 用 IAR C/C++ 開発ガイド』* を参照してください。 |
| `--BE32` | ビッグエンディアンフォーマット BE32 を使用します。リファレンス情報については、*『ARM 用 IAR C/C++ 開発ガイド』* を参照してください。 |
| `--cpu` | 派生プロセッサを指定します。リファレンス情報については、*『ARM 用 IAR C/C++ 開発ガイド』* を参照してください。 |
| `--device` | デバイスの名前を指定します。 |
| `--drv_attach_to_program` | 実行中のアプリケーションの現在の位置にデバッガを接続します。リファレンス情報については、532 ページの *ダウンロード*のオプション **［実行中のターゲットにアタッチする］** を参照してください。 |
| `--drv_catch_exceptions` | 特定の例外の場合にアプリケーションを停止します。 |
| `--drv_communication` | 使用する通信リンクを指定します。 |
| `--drv_communication_log` | ログファイルを作成します。 |
| `--drv_default_breakpoint` | ブレークポイントの設定時に使用するブレークポイントリソースの種類を設定します。 |
| `--drv_reset_to_cpu_start` | アプリケーションをリセットする際、PC の設定を省略します。 |
| `--drv_restore_breakpoints` | システム起動中に破壊されたブレークポイントを自動的に復元します。 |
| `--drv_suppress_download` | 実行可能イメージのダウンロードを抑制します。リファレンス情報については、532 ページの *ダウンロード*のオプション **［ダウンロードしない］** を参照してください。 |
| `--drv_vector_table_base` | Cortex-M リセットベクタの位置およびスタックポインタの初期値を指定します。 |

| | |
|---|---|
| `--drv_verify_download` | ターゲットプログラムを検証します。リファレンス情報については、532 ページの*ダウンロード*のオプション［**ベリファイする**］を参照してください。<br><br>Angel、GDB Server、IAR ROM-monitor、J-Link/J-Trace、JTAGjet、Macraigor、RDI、ST-LINK、TI Stellaris、、TI XDS で使用できます。 |
| `--endian` | 生成されるコードおよびデータのバイトオーダを指定します。リファレンス情報については、*『ARM 用 IAR C/C++ 開発ガイド』*を参照してください。 |
| `--fpu` | 浮動小数点ユニット型を選択します。リファレンス情報については、*『ARM 用 IAR C/C++ 開発ガイド』*を参照してください。 |
| `--leave_running` | ターゲット上で実行を開始し、終了した後もターゲットを実行したままにします。 |
| `-p` | 使用するデバイス記述ファイルを指定します。 |
| `--proc_stack_stack` | C-SPY に予約済スタックの情報を提供します。 |
| `--semihosting` | セミホスティング I/O を有効にします。 |

## シミュレータドライバで使用可能なオプション

| | |
|---|---|
| `--disable_interrupts` | 割込みシミュレーションを無効化します。 |
| `--mapu` | メモリアクセスチェックをアクティブにします。 |

## C-SPY ANGEL デバッグモニタドライバで使用可能なオプション

| | |
|---|---|
| `--rdi_heartbeat` | C-SPY でターゲットシステムを定期的にポーリングします。リファレンス情報については、538 ページの *Angel* のオプション［**ハートビート送信**］を参照してください。 |
| `--rdi_step_max_one` | 命令を 1 つ実行します。 |

## C-SPY GDB サーバドライバで使用可能なオプション

| | |
|---|---|
| `--gdbserv_exec_command` | コマンド文字列を GDB サーバに送信します。 |

## C-SPY IAR ROM-MONITOR ドライバで使用可能なオプション

C-SPY IAR ROM-monitor ドライバに固有のオプションは他にありません。

## C-SPY I-JET/JTAGJET ドライバで使用可能なオプション

| | |
|---|---|
| `--drv_interface` | 通信インタフェースを選択します。 |
| `--drv_interface_speed` | JTAG と SWD の速度を指定します。 |
| `--jet_board_cfg` | プローブ設定ファイルを指定します。 |
| `--jet_board_did` | 複数コアシステムでデバッグする CPU を選択します。 |
| `--jet_cpu_clock` | 内部プロセッサクロックの周波数を指定します。 |
| `--jet_ir_length` | 接続先の ARM コアの前の IR ビット数を指定します。 |
| `--jet_power_from_probe` | I-jet または I-jet Trace プローブからの電源を指定します。 |
| `--jet_probe` | C-SPY I-jet/JTAGjet ドライバがインタフェースとなるデバッグシステムを指定します。 |
| `--jet_script_file` | リセットスクリプトファイルを指定します。 |
| `--jet_standard_reset` | C-SPY の起動時に使用するリセット方法を選択します。 |
| `--jet_startup_connection_timeout` | C-SPY ドライバがターゲットボードに接続しようとする時間を延長します。 |
| `--jet_swo_on_d0` | SWO トレースデータがトレースデータピンの `D0` に出力されるよう指定します。 |
| `--jet_swo_prescaler` | CPU クロック周波数の SWO プリスケーラを指定します。 |
| `--jet_swo_protocol` | SWO 通信プロトコルを選択します。 |

| | |
|---|---|
| --jet_tap_position | JTAG スキャンチェーン内の特定デバイスを選択します。 |
| --reset_style | デバッグ時に使用可能なリセット方式を指定します。 |

### C-SPY CMSIS-DAP ドライバで使用可能なオプション

| | |
|---|---|
| --drv_interface | 通信インタフェースを選択します。 |
| --drv_interface_speed | JTAG と SWD の速度を指定します。 |
| --jet_board_cfg | プローブ設定ファイルを指定します。 |
| --jet_board_did | 複数コアシステムでデバッグする CPU を選択します。 |
| --jet_probe | C-SPY ドライバがインタフェースとなるデバッグシステムを指定します。 |
| --jet_script_file | リセットスクリプトファイルを指定します。 |
| --jet_standard_reset | C-SPY の起動時に使用するリセット方法を選択します。 |
| --jet_startup_connection_timeout | C-SPY ドライバがターゲットボードに接続しようとする時間を延長します。 |
| --jet_tap_position | JTAG スキャンチェーン内の特定デバイスを選択します。 |
| --reset_style | デバッグ時に使用可能なリセット方式を指定します。 |

### C-SPY J-LINK/J-TRACE ドライバで使用可能なオプション

| | |
|---|---|
| --drv_interface | 通信インタフェースを選択します。 |
| --drv_interface_speed | JTAG と SWD の速度を指定します。 |
| --drv_swo_clock_setup | CPU クロックと必要な SWO 速度を指定します。 |
| --jlink_dcc_timeout | C-SPY からターゲット上の DCC エージェントへの保留中のリクエストについて、タイムアウトを指定します。 |

| | |
|---|---|
| --jlink_device_select | JTAG スキャンチェーン内の特定デバイスを選択します。 |
| --jlink_exec_command | ターゲット接続が確立された後に __jlinkExecCommand マクロを呼び出します。 |
| --jlink_initial_speed | 最初の JTAG 通信速度 (kHz) を指定します。 |
| --jlink_ir_length | 接続先の ARM コアの前の IR ビット数を設定します。 |
| --jlink_reset_strategy | デバッガの起動時に使用するリセット方法を選択します。 |
| --jlink_script_file | ハードウェアを設定するスクリプトファイルを指定します。 |
| --jlink_trace_source | ETB または ETM をトレースのソースとして選択します。 |

### C-SPY TI STELLARIS ドライバで使用可能なオプション

| | |
|---|---|
| --drv_interface | 通信インタフェースを選択します。 |
| --drv_interface_speed | JTAG と SWD の速度を指定します。 |

### C-SPY TI XDS ドライバで使用可能なオプション

| | |
|---|---|
| --xds_rootdir | TI XDS ドライバパッケージのインストールディレクトリを指定します。 |

### C-SPY MACRAIGOR ドライバで使用可能なオプション

| | |
|---|---|
| --drv_interface | 通信インタフェースを選択します。 |
| --drv_interface_speed | JTAG と SWD の速度を指定します。 |
| --mac_handler_address | Intel XScale デバイスで使用されるデバッグハンドラの位置を指定します。 |
| --mac_jtag_device | ハードウェアデバイスに対応するデバイスを選択します。 |

| | |
|---|---|
| --mac_multiple_targets | JTAG スキャンチェーンに複数のデバイスがある場合、接続するデバイスを指定します。 |
| --mac_reset_pulls_reset | C-SPY で最初のハードウェアリセットを生成します。 |
| --mac_set_temp_reg_buffer | ドライバにコプロセッサのアクセス用の物理 RAM アドレスを提供します。 |
| --mac_xscale_ir7 | XScale ir7 アーキテクチャの使用を指定します。 |

### C-SPY RDI ドライバで使用可能なオプション

| | |
|---|---|
| --rdi_allow_hardware_reset | ハードウェアリセットを実行します。 |
| --rdi_driver_dll | ドライバ DLL ファイルのパスを指定します。 |
| --rdi_step_max_one | 命令を 1 つ実行します。 |

### C-SPY ST-LINK ドライバで使用可能なオプション

| | |
|---|---|
| --drv_interface | 通信インタフェースを選択します。 |
| --drv_swo_clock_setup | CPU クロックと必要な SWO 速度を指定します。 |
| --stlink_reset_strategy | 使用するリセット方法を指定します。 |

### C-SPY サードパーティ製ドライバで使用可能なオプション

使用するサードパーティ製ドライバに固有なオプションについては、そのドライバのドキュメントを参照してください。

## C-SPY コマンドラインオプションについてのリファレンス情報

ここでは、cspybat の各オプションおよび C-SPY ドライバで使用可能な各オプションに関する詳細なリファレンス情報を提供します。

## -B

**構文**

```
-B
```

**適用範囲**

すべての C-SPY ドライバ。

**説明**

このオプションは、バッチモードを有効にするときに使用します。

このオプションは、IDE では使用できません。

## --backend

**構文**

```
--backend {driver options}
```

**パラメータ**

*driver options*

使用している C-SPY ドライバで使用可能なすべてのオプション。

**適用範囲**

cspybat への送信（必須）。

**説明**

このオプションは、各種オプションを C-SPY ドライバに送信するときに使用します。--backend の後に続くすべてのオプションは、C-SPY ドライバに送信され、cspybat 自身では処理されません。

このオプションは、IDE では使用できません。

## --code_coverage_file

**構文**

```
--code_coverage_file file
```

このオプションはコマンドライン上で --backend オプションより前に配置しなければならない点に注意してください。

**パラメータ**

*file*

コードカバレッジ情報の対象ファイル名。

**適用範囲**

cspybat

**説明**

このオプションを使用して、コードカバレッジ情報の生成を有効にします。コードカバレッジ情報は実行が完了した後に生成され、指定したファイルに保存されます。

このオプションでは、使用する C-SPY ドライバがコードカバレッジをサポートしている必要があります。このオプションをコードカバレッジをサポートしていない C-SPY ドライバで使用すると、エラーメッセージが stderr に出力されます。

**関連項目**

293 ページの *コードカバレッジ*。

このオプションを設定するには、**［表示］>［コードカバレッジ］**を選択して、C-SPY デバッガの実行中に**［名前を付けて保存］**を選択します。

## --cycles

**構文**

```
--cycles cycles
```

このオプションはコマンドライン上で --backend オプションより前に配置しなければならない点に注意してください。

**パラメータ**

*cycles*

実行するサイクル数。

**適用範囲**

cspybat

**説明**

このオプションは、実行するサイクルの最大回数を指定するときに使用します。ターゲットプログラムが指定サイクル数より長く実行された場合、ターゲットプログラムは異常終了されます。このオプションを使用するには、使用している C-SPY ドライバがサイクルカウンタをサポートし、実行中にサイクルカウントのサンプリングが可能であることが必要です。

このオプションは、IDE では使用できません。

## --debugfile

**構文**

```
--debugfile filename
```

**パラメータ**

*filename*

使用するデバッグファイル名。

**適用範囲**

cspybat

このオプションは、コマンドライン上で --backend オプションの前後どちらにも配置することができます。

**説明**

このオプションを使用して、cspybat で、生成された cpsybat.bat ファイルではなく、指定したデバッグファイルを使用するようにします。

このオプションは、IDE では使用できません。

## --device

**構文**

```
--device=device_name
```

**パラメータ**

| | |
|---|---|
| *device_name* | ADuC7030、AT91SAM7S256、LPC2378、STR912FM44、TMS470R1B1M などのデバイス名。 |

**適用範囲**

すべての C-SPY ドライバ。

**説明**

このオプションを使用して、デバイス名を指定します。

オプションを設定するには、以下のように選択します。

**[プロジェクト] > [オプション] > [一般オプション] > [ターゲット] > [デバイス]**

## --disable_interrupts

**構文**

```
--disable_interrupts
```

**適用範囲**

C-SPY シミュレータドライバ。

**説明**

このオプションは、命令シミュレーションを無効にするときに使用します。

このオプションを設定するには、**[シミュレータ] > [割込み設定]** を選択して、**[割込みシミュレーションを有効にする]** オプションの選択を解除します。

## --download_only

**構文**

```
--download_only
```

このオプションはコマンドライン上で --backend オプションより前に配置しなければならない点に注意してください。

**適用範囲**

cspybat

**説明**

このオプションを使用して、後でデバッグセッションを開始せずにコードイメージをダウンロードします。

**［プロジェクト］>［ダウンロード］>［アクティブなアプリケーションのダウンロード］**

または、関連オプションを設定するには以下を選択します。

**［プロジェクト］>［オプション］>［デバッガ］>［設定］**を選択し、**［指定位置まで実行］**の選択を解除します。

## --drv_catch_exceptions

**構文**

```
--drv_catch_exceptions=value
```

**パラメータ**

| | |
|---|---|
| *value*<br><br>（ARM9、Cortex-R4、ARM11、Cortex-A の場合） | 0-0x1FF の範囲の値です。各ビットは、以下のように、キャッチする例外を指定します。<br>ビット 0 = リセット<br>ビット 1 = 未定義の命令<br>ビット 2 = SWI<br>ビット 3 = プリフェッチアボート<br>ビット 4 = データアボート<br>ビット 5 = 未使用<br>ビット 6 = IRQ<br>ビット 7 = FIQ<br>ビット 8 = その他のエラー |

*value*

（Cortex-M の場合）

0-0x7FF の範囲の値です。各ビットは、以下のように、キャッチする例外を指定します。

Bit 0 = CORERESET - リセットベクタ

Bit 4 = MMERR - メモリ管理の障害

Bit 5 = NOCPERR - コプロセッサのアクセスエラー

Bit 6 = CHKERR - チェッキングエラー

Bit 7 = STATERR - ステートエラー

Bit 8 = BUSERR - バスエラー

Bit 9 = INTERR - 割込みサービスエラー

Bit 10 = HARDERR - ハード障害

**適用範囲**

C-SPY Angel デバッグモニタドライバ。

C-SPY I-jet/JTAGjet ドライバ。

C-SPY J-Link/J-Trace ドライバ。

C-SPY CMSIS-DAP ドライバ。

C-SPY RDI ドライバ。

**説明**

このオプションは、特定の例外発生時にアプリケーションを停止するときに使用します。

**関連項目**

143 ページの *例外ベクタ上へのブレークポイントの設定*。

C-SPY Angel デバッグモニタドライバの場合。

**［プロジェクト］>［オプション］>［デバッガ］>［追加オプション］**

C-SPY I-jet/JTAGjet ドライバおよび C-SPY J-Link/J-Trace ドライバの場合、以下を使用します。

**［プロジェクト］>［オプション］>［デバッガ］>** ***［ドライバ］*** **>［ブレークポイント］>［例外の取得］**

C-SPY RDI ドライバの場合

**［プロジェクト］>［オプション］>［デバッガ］>［RDI］>［例外の取得］**

## --drv_communication

**構文**

```
--drv_communication=connection
```

**パラメータ**

C-SPY Angel デバッグモニタドライバの場合、connection は以下のいずれかです。

イーサネット経由

`UDP:ip_address`

`UDP:ip_address,port`

`UDP:hostname`

`UDP:hostname,port`

シリアルポート経由

*port:baud,parity,stop_bit,handshake*

*port* = COM1-COM256（デフォルト COM1）

*ボー* = 9600、19200、38400、57600、115200（デフォルト値 9600 baud）

*parity* = N（パリティなし）

*stop_bit* = 1（ストップビットは 1 ビット）

*handshake* = NONE または RTSCTS（ハンドシェイクなしの場合はデフォルト NONE）

例：COM1:9600,N,8,1,NONE

C-SPY GDB サーバドライバの接続が以下のいずれかの場合 :

イーサネット経由

TCPIP:*ip_address*

TCPIP:*ip_address*,*port*

TCPIP:*hostname*

TCPIP:*hostname*,*port*

ポートを指定しない場合、デフォルトでポート 3333 が使用されます。

C-SPY IAR ROM-monitor ドライバの場合、*connection* は以下のいずれかです。

シリアルポート経由

*port*:*baud*,*parity*,*stop_bit*,*handshake*

*port* = COM1-COM256（デフォルト COM1）

*ボー* = 9600、19200、38400、57600、115200（デフォルト値 9600 baud）

*parity* = N（パリティなし）

*stop_bit* = 1（ストップビットは 1 ビット）

*handshake* = NONE または RTSCTS（ハンドシェイクなしの場合はデフォルト NONE）

例 : COM1:9600,N,8,1,NONE

C-SPY J-Link/J-Trace ドライバの場合、*connection* は以下のいずれかです。

USB ポート経由

USB:#*serial* で、*serial* は接続先のプローブを識別する数値と文字からなる文字列です。このシリアル番号はプローブ上に記載されているか、プローブを 1 つだけ接続してデバッグセッションを開始することにより入手できます。こうすることで、シリアル番号が［**デバッグログ**］ウィンドウに表示されます。シリアル番号は、［**デバッグプローブ選択**］ダイアログボックスにも表示されます。

USB:#select は、デバッグセッションを開始するたびに［**デバッグプローブ選択**］ダイアログボックスが表示されるようにします。

USB 経由でデバッグプローブに直接

`USB0-USB3`

`USB0` を使用し、USB 接続上に複数のデバッグプローブがある場合は、デバッグセッションを開始するとダイアログボックスが表示されます。このダイアログボックスを使用して、接続するデバッグプローブを選択します。

LAN 上の J-Link を経由

`TCPIP:`

コロン符号の後にアドレスやホスト名、シリアル番号が何もない場合は、J-Link ドライバはローカルネットワーク上のすべての J-Link デバッグプローブを検索して、それらをダイアログボックスに表示します。ここで接続するプローブを選択できます（自動検出）。

`TCPIP:`*`ip_address`*

`TCPIP:`*`ip_address,port`*

`TCPIP:`*`hostname`*

`TCPIP:`*`hostname,port`*

`TCPIP:#`*`serial`* は、ローカルネットワーク上のシリアル番号 *`number`* を持つ J-Link に接続します。

ポートを指定しない場合、デフォルトでポート `19020` が使用されます。

C-SPY I-jet/JTAGjet ドライバの場合、*`connection`* は以下のいずれかです。

USB ポート経由

`USB:#`*`serial`* で、*`serial`* は接続先のプローブを識別する数値と文字からなる文字列です。このシリアル番号はプローブ上に記載されているか、プローブを 1 つだけ接続してデバッグセッションを開始することにより入手できます。こうすることで、シリアル番号が［**デバッグログ**］ウィンドウに表示されます。シリアル番号は、［**デバッグプローブ選択**］ダイアログボックスにも表示されます。

`USB:#select` は、デバッグセッションを開始するたびに［**デバッグプローブ選択**］ダイアログボックスが表示されるようにします。

C-SPY Macraigor ドライバの場合、*connection* は以下のいずれかです。

mpDemon の場合

*port:baud*

*port* = COM1-COM4

*ボー* = 9600、19200、38400、57600、115200（デフォルト値 9600 baud）

mpDemon の場合

TCPIP:*ip_address*

TCPIP:*ip_address,port*

TCPIP:*hostname*

TCPIP:*hostname,port*

ポートを指定しない場合、デフォルトでポート 19020 が使用されます。

USB 経由で usbDemon および usb2Demon

USB ports = USB0-USB3

C-SPY ST-LINK ドライバの場合、*connection* は以下のいずれかです。

USB ポート経由

USB:#*serial* で、*serial* は接続先のプローブを識別する数値と文字からなる文字列です。このシリアル番号はプローブ上に記載されているか、プローブを 1 つだけ接続してデバッグセッションを開始することにより入手できます。こうすることで、シリアル番号が［**デバッグログ**］ウィンドウに表示されます。シリアル番号は、［**デバッグプローブ選択**］ダイアログボックスにも表示されます。

USB:#select は、デバッグセッションを開始するたびに［**デバッグプローブ選択**］ダイアログボックスが表示されるようにします。

USB*x* で、*x* は接続時のプローブの列挙順 (0-256) を示します。これは、シリアル番号が使用できない場合の代わりの記述法で、旧型のプローブに対する解決法です。ただし、プローブを次回接続したときやコンピュータを再起動した場合に順序が変わる可能性があるため、この方法は不正確です。USB ポートは、使用するすべてのプローブを接続すると取得できます。続いて、--drv_communication=USB:#select を使用して、［**デバッグプローブ選択**］ダイアログボックスに接続されたすべてのプローブを表示します。

C-SPY TI Stellaris ドライバの場合、*connection* は以下のいずれかです。

USB ポート経由

USB:#*serial* で、*serial* は接続先のプローブを識別する数値と文字からなる文字列です。このシリアル番号はプローブ上に記載されているか、プローブを 1 つだけ接続してデバッグセッションを開始することにより入手できます。こうすることで、シリアル番号が［**デバッグログ**］ウィンドウに表示されます。シリアル番号は、［**デバッグプローブ選択**］ダイアログボックスにも表示されます。

USB:#select は、デバッグセッションを開始するたびに［**デバッグプローブ選択**］ダイアログボックスが表示されるようにします。

USB*x* で、*x* は接続時のプローブの列挙順 (0-256) を示します。これは、シリアル番号が使用できない場合の代わりの記述法で、旧型のプローブに対する解決法です。ただし、プローブを次回接続したときやコンピュータを再起動した場合に順序が変わる可能性があるため、この方法は不正確です。USB ポートは、使用するすべてのプローブを接続すると取得できます。続いて、--drv_communication=USB:#select を使用して、［**デバッグプローブ選択**］ダイアログボックスに接続されたすべてのプローブを表示します。

C-SPY TI XDS ドライバの場合、*connection* は以下のいずれかです。

USB ポート経由（XDS100 のみ）

USB:#*serial* で、*serial* は接続先のプローブを識別する数値と文字からなる文字列です。このシリアル番号はプローブ上に記載されているか、プローブを 1 つだけ接続してデバッグセッションを開始することにより入手できます。こうすることで、シリアル番号が［**デバッグログ**］ウィンドウに表示されます。シリアル番号は、［**デバッグプローブ選択**］ダイアログボックスにも表示されます。

USB:#select は、デバッグセッションを開始するたびに［**デバッグプローブ選択**］ダイアログボックスが表示されるようにします。

USB*x* で、*x* は接続時のプローブの列挙順 (0-256) を示します。これは、シリアル番号が使用できない場合の代わりの記述法で、旧型のプローブに対する解決法です。ただし、プローブを次回接続したときやコンピュータを再起動した場合に順序が変わる可能性があるため、この方法は不正確です。USB ポートは、使用するすべてのプローブを接続すると取得できます。続いて、--drv_communication=USB:#select を使用して、［**デバッグプローブ選択**］ダイアログボックスに接続されたすべてのプローブを表示します。

COM ポート経由（XDS200 のみ）

COM*x* で、*x* はプローブ接続時の列挙順 (0-256) を示します。これは、プローブを次回接続したときや、コンピュータを再起動すると順序が変わる可能性があるため、不正確な方法です。

COM:#select は、デバッグセッションを開始するたびに［**デバッグプローブ選択**］ダイアログボックスが表示されるようにします。

**適用範囲**

C-SPY Angel デバッグモニタドライバ

C-SPY GDB サーバドライバ

C-SPY IAR ROM-monitor ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY Macraigor ドライバ

C-SPY ST-LINK ドライバ

C-SPY TI Stellaris ドライバ

C-SPY TI XDS ドライバ

**説明**

このオプションは、通信リンクを選択するときに使用します。

**［プロジェクト］>［オプション］>［デバッガ］>［Angel］>［通信］**

**［プロジェクト］>［オプション］>［デバッガ］>［GDB サーバ］>［TCP/IP アドレスまたはホスト名［,port］］**

**［プロジェクト］>［オプション］>［デバッガ］>［IAR ROM モニタ］>［通信］**

**［プロジェクト］>［オプション］>［デバッガ］>［J-Link/J-Trace］>［接続］>［通信］**

C-SPY Macraigor ドライバの関連オプションを設定するには、以下のように選択します。

**［プロジェクト］>［オプション］>［デバッガ］>［Macraigor］**

このオプションを C-SPY ST-LINK ドライバ、C-SPY TI Stellaris ドライバ、C-SPY TI XDS ドライバに設定するには、**［プロジェクト］>［オプション］>［デバッガ］>［追加オプション］**を使用します。

## --drv_communication_log

**構文**

```
--drv_communication_log=filename
```

**パラメータ**

| | |
|---|---|
| `filename` | ログファイルの名前。 |

**適用範囲**

すべての C-SPY ハードウェアドライバ。

**説明**

このオプションを選択すると、C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。結果を解析するには、通信プロトコルに関する詳しい知識が必要です。

**［プロジェクト］>［オプション］>［デバッガ］>*［ドライバ］*>［通信ログ］**

## --drv_default_breakpoint

**構文**

```
--drv_default_breakpoint={0|1|2}
```

**パラメータ**

| | |
|---|---|
| 0 | 自動（デフォルト） |
| 1 | ハードウェア |
| 2 | ソフトウェア |

**適用範囲**

C-SPY GDB サーバドライバ

C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY CMSIS-DAP ドライバ

C-SPY Macraigor ドライバ

**説明**

このオプションでは、ブレークポイントの設定時に使用するブレークポイントリソースの種類を選択します。

**関連項目**

164 ページの *［ブレークポイント］ダイアログボックス*。

**［プロジェクト］>［オプション］>［デバッガ］>［*ドライバ*］>［ブレークポイント］>［デフォルトのブレークポイントタイプ］**

## --drv_interface

**構文**

```
--drv_interface={SWD|JTAG}
```

**パラメータ**

| | |
|---|---|
| SWD | SWD インタフェースを指定します。 |
| JTAG（デフォルト） | JTAG インタフェースを指定します。 |

**適用範囲**

C-SPY CMSIS-DAP ドライバ

C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY Macraigor ドライバ

C-SPY P&E Micro ドライバ

C-SPY ST-LINK ドライバ

C-SPY TI Stellaris ドライバ

**説明**

このオプションは、デバッグプローブとターゲットシステム間の通信インタフェースを指定するときに使用します。

SWD インタフェースは、JTAG よりも少数のピンを使用します。serial-wire output (SWO) 通信チャンネルを使用する場合に `--drv_interface=SWD` を指定します。または、このオプションを `JTAG` に設定して、`--jet_swo_on_d0` オプションを指定することもできます。Trace_D0 上の SWO 出力は、C-SPY I-Jet/I-jet Trace ドライバでのみサポートされています。

［**一般オプション**］>［**ライブラリ構成**］ページで［**SWO 経由の stdout/stderr**］を選択した場合は、デバイスが Trace_D0 で SWO 出力をサポートしている場合を除き、SWD が自動的に選択されます。

**関連項目**

- 231 ページの *［SWO トレースウィンドウ設定］ダイアログボックス*
- 561 ページの *J-Link/J-Trace 接続オプション*
- 567 ページの *ST-LINK*

［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**CMSIS-DAP**］>［**JTAG/SWD/ インタフェース**］

［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**I-jet/JTAGjet**］>［**JTAG/SWD**］>［**インタフェース**］

［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**J-Link/J-Trace**］>［**接続**］>［**インタフェース**］

［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**Macraigor**］>［**インタフェース**］

［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**PE micro**］>［**インタフェース**］

［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**ST-LINK**］>［**ST-LINK**］>［**インタフェース**］

［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**TI Stellaris**］>［**インタフェース**］

## --drv_interface_speed

**構文**

```
--drv_interface_speed=Hz
```

パラメータ

| | |
|---|---|
| *Hz* | 周波数 (Hz) |

適用範囲

C-SPY CMSIS-DAP ドライバ

C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY Macraigor ドライバ

C-SPY P&E Micro ドライバ

C-SPY TI Stellaris ドライバ

説明

このオプションを使用して、JTAG および SWD の通信速度を Hz で指定します。

関連項目

556 ページの *J-Link/J-Trace の設定オプション*。

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet］>［JTAG/SWD］>［JTAG/SWD 速度］**

**［プロジェクト］>［オプション］>［デバッガ］>［Macraigor］>［JTAG 速度］**

**［プロジェクト］>［オプション］>［デバッガ］>［J-Link/J-Trace］>［設定］>［JTAG 速度］**

**［プロジェクト］>［オプション］>［デバッガ］>［TI Stellaris］>［設定］>［JTAG 速度］**

## --drv_reset_to_cpu_start

構文

```
--drv_reset_to_cpu_start
```

適用範囲

C-SPY Angel デバッグモニタドライバ

C-SPY GDB サーバドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY TI Stellaris ドライバ

C-SPY TI XDS ドライバ

C-SPY Macraigor ドライバ

C-SPY RDI ドライバ

C-SPY ST-LINK ドライバ

**説明**

通常はリセット時に、PC がアプリケーションのエントリポイントに設定されます。

このオプションは、アプリケーションがリセットされるたびに PC の設定を省略します。これは、リセット時に CPU が設定するリセット値を保持する場合に便利です。たとえば、ベクタテーブルがポイントする最初の命令から実行を開始したり、アプリケーションの開始アドレスに入る前にブートローダあるいは OS 起動コードを実行する場合などです。

このオプションは、CPU により設定された SP （Cortex-M の場合）または CPSR レジスタ（その他のデバイス）の値も保持します。

このオプションを設定するには、**[プロジェクト] > [オプション] > [デバッガ] > [追加オプション]** を使用します。

## --drv_restore_breakpoints

**構文**

```
--drv_restore_breakpoints=location
```

**パラメータ**

| | |
|---|---|
| *location* | アドレスまたは関数名ラベル |

**適用範囲**

C-SPY GDB サーバドライバ

C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY CMSIS-DAP ドライバ

C-SPY Macraigor ドライバ

**説明**

このオプションを使用すると、システム起動中にオーバライドされたすべてのソフトウェアブレークポイントを自動的に復元します。

**関連項目**

164 ページの *[ブレークポイント] ダイアログボックス*。

**[プロジェクト] > [オプション] > [デバッガ] > *[ドライバ]* > [ブレークポイント] > [ソフトウェアブレークポイント復元位置]**

## --drv_swo_clock_setup

**構文**

```
--drv_swo_clock_setup=frequency,autodetect,wanted
```

**パラメータ**

| | |
|---|---|
| *frequency* | 内部プロセッサクロック HCLK の正確なクロック周波数 (Hz)。この値は、SWO の通信速度の設定およびタイムスタンプの計算に使用します。 |
| *autodetect* | 0: パラメータ *wanted* を使用して、希望する周波数を指定します。<br>1: J-Link デバッグプローブで使用できる最大可能周波数を自動的に使用します。 |
| *wanted* | *autodetect* が 0 の場合に使用する周波数 (Hz)。*wanted* は、送信中にデータパケットが失われる場合に使用します。 |

**適用範囲**

J-Link ドライバおよび ST-LINK ドライバ。

**説明**

このオプションを使用して、CPU クロックを設定します。このオプションを使用しない場合、CPU クロック周波数はデフォルトで 72 MHz に設定されます。

**［J-Link］>［SWO 設定］>［CPU クロック］**

**［J-Link］>［SWO 設定］>［SWO クロック］>［自動検出］**

**［J-Link］>［SWO 設定］>［SWO クロック］>［希望する値］**

## --drv_vector_table_base

**構文**

```
--drv_vector_table_base=expression
```

**パラメータ**

| | |
|---|---|
| *expression* | ラベルまたはアドレス |

**適用範囲**

C-SPY GDB サーバドライバ

C-SPY I-jet/JTAGjet ドライバ

C-SPY J-Link/J-Trace ドライバ

C-SPY CMSIS-DAP ドライバ

C-SPY TI Stellaris ドライバ

C-SPY TI XDS ドライバ

C-SPY Macraigor ドライバ

C-SPY RDI ドライバ

C-SPY ST-LINK ドライバ

C-SPY シミュレータドライバ

**説明**

このオプションでは、リセットベクタの位置を指定します（これによって、Cortex-M の初期スタックポインタ値の配置も決まります）。このオプションは、アプリケーション内でデフォルトの `__vector_table` ラベル（システム起動コードで定義）をオーバライドする場合や、アプリケーションにこのラベルが欠落している場合に便利です。後者は、別のベンダ製ツールによってビルドされたコードをデバッグする場合に起こり得ます。

このオプションを設定するには、**[プロジェクト] > [オプション] > [デバッガ] > [追加オプション]** を使用します。

## -f

**構文**

`-f` *`filename`*

**パラメータ**

*`filename`*

コマンドを含むテキストファイル（デフォルトのファイル名拡張子は `xcl`）。

**適用範囲**

`cspybat`

このオプションは、コマンドライン上で `--backend` オプションの前後どちらにも配置することができます。

**説明**

このオプションを使用して、`cspybat` で指定ファイルからコマンドラインオプションを読み取ります。

コマンドファイルでは、項目はコマンドライン上でのフォーマットと同様に記述します。ただし、改行文字は空白文字やタブと同様に処理されるため、複数行にわたって記述できます。

ファイルでは、C と C++ の両スタイルのコメントを使用できます。二重引用符は、Microsoft Windows のコマンドライン環境の場合と同様に機能します。

このオプションを設定するには、**[プロジェクト] > [オプション] > [デバッガ] > [追加オプション]** を使用します。

## --flash_loader

**構文**

```
--flash_loader filename
```

このオプションはコマンドライン上で --backend オプションより前に配置しなければならない点に注意してください。

**パラメータ**

*filename*

ファイル名の拡張子が board のフラッシュローダの仕様 XML ファイル。

**適用範囲**

cspybat

**説明**

このオプションは、フラッシュのロードに関するすべての関連情報を記述したフラッシュローダ仕様 xml ファイルを指定するときに使用します。この引数は複数指定できます。その場合、それぞれの引数が指定の順序で処理され、複数のフラッシュプログラミングパスが得られます。

**関連項目**

*フラッシュローダ開発ガイド*

オプションを設定するには、以下のように選択します。

**[プロジェクト] > [オプション] > [デバッガ] > [フラッシュローダを使用する]**

## --gdbserv_exec_command

**構文**

```
--gdbserv_exec_command="string"
```

**パラメータ**

| | |
|---|---|
| "*string*" | GDB サーバに送信する文字列またはコマンドです。詳細についてはドキュメントを参照してください。 |

**適用範囲**

C-SPY GDB サーバドライバ。

**説明**

このオプションは、文字列やコマンドを GDB サーバに送信するときに使用します。

**[プロジェクト] > [オプション] > [デバッガ] > [追加オプション]**

## --jet_board_cfg

**構文**

```
--jet_board_cfg=probe_configuration_file
```

**パラメータ**

*probe_configuration_file*　プローブ設定ファイルのフルパス。

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ

C-SPY CMSIS-DAP ドライバ

**説明**

このオプションを使用して、ボード上のデバッグシステムを定義するプローブ設定ファイルを指定します。

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet］>［JTAG/SWD］>［プローブ設定ファイル］**

## --jet_board_did

**構文**

```
--jet_board_did={cpu|#cpu_number}
```

**パラメータ**

| | |
|---|---|
| *cpu* | ボード設定ファイルが指定され（`--jet_board_cfg` を使用）、定義されたデバッグシステムに複数の CPU がある場合は、このパラメータを使用して CPU を 1 つ選択します。*cpu* の値はテキスト文字列です。有効な値の範囲は、プローブ設定ファイルにあります。 |
| #*cpu_number* | デバッグシステムが複数コアの SWD システムの場合、DAP の CPU 番号を指定してください。<br><br>デバッグシステムが JTAG スキャンチェーンで、指定した TAP 位置に複数の CPU がある場合、ターゲット上の CPU 番号を指定してください。<br><br>ボード設定ファイルが `--jet_board_cfg` を使用して指定されていると、#*cpu_number* は機能しない点に注意してください。 |

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ

C-SPY CMSIS-DAP ドライバ

**説明**

このオプションを使用して、複数コアシステムでデバッグする CPU を指定します。

--jet_probe=cmsisdap を指定した場合、--jet_board_did=#*cpu_number* も適用可能です。

**例**

プローブ設定ファイルを持つ複数コアデバイスで CPU を選択する場合 :

```
--jet-board-cfg=device.ProbeConfig --jet_board_did=A9_1
```

JTAG スキャンチェーンを持つ複数コアデバイスで CPU を選択する場合。ただし、指定の TAP 位置に複数の CPU があることが条件です :

```
--jet_tap_position=1 --jet_ir_length=5 --jet_board_did=#2
```

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet］>［JTAG/SWD］>［プローブ設定ファイル］>［CPU］**

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet/JTAG/SWD］>［明示的なプローブ設定］>［ターゲット上の CPU］**

## --jet_cpu_clock

**構文**

```
--jet_cpu_clock=frequency
```

**パラメータ**

| | |
|---|---|
| *frequency* | クロック周波数 (Hz) |

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ。

**説明**

このオプションを使用して、内部プロセッサクロック HCLK の正確なクロック周波数を指定します。この値は、SWO の通信速度の設定およびタイムスタンプの計算に使用します。

**注 :** このオプションは、オプション --jet_swo_protocol が UART に設定されている場合にのみ適用されます。

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet］>［SWO］>［クロック設定］>［CPU クロック］**

## --jet_ir_length

**構文**

```
--jet_ir_length=length
```

**パラメータ**

| | |
|---|---|
| *length* | 接続先 ARM コアの前の IR ビット数で、ARM デバイスと他のデバイスが混在する JTAG スキャンチェーン用です。 |

**適用範囲** C-SPY I-jet/JTAGjet ドライバ。

**説明** このオプションは、接続先 ARM コアの前の IR ビット数を設定します。

**関連項目** 550 ページの *I-jet/JTAGjet の JTAG/SWD オプション*。

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet］>［JTAG/SWD］>［明示的なプローブ設定］>［先行ビット］**

## --jet_power_from_probe

**構文**

```
--jet_power_from_probe=[leave_on|switch_off]
```

**パラメータ**

| | |
|---|---|
| `leave_on` | デバッグセッションの停止後もターゲットに電源を供給し続けます。 |
| `switch_off` | デバッグセッションが停止すると、ターゲットの電源をオフにします。 |

**適用範囲** C-SPY I-jet/JTAGjet ドライバ。

**説明** このオプションを使用して、デバッグ後のプローブの電源のステータスを指定します。

このオプションを指定しない場合、プローブはボードに電源を供給しません。

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet］>［設定］>［ターゲットの電源］**

## --jet_probe

**構文**

```
--jet_probe=[ijet|cmsisdap]
```

**パラメータ**

| | |
|---|---|
| `ijet` | C-SPY I-jet/JTAGjet ドライバを I-jet、I-jet Trace、または JTAGjet プローブへのインタフェースとして指定します。 |
| `cmsisdap` | CMSIS-DAP システムへのインタフェースとして C-SPY I-jet/JTAGjet ドライバを指定します。 |

**適用範囲** C-SPY I-jet/JTAGjet ドライバ

C-SPY CMSIS-DAP ドライバ

**説明**

このオプションを使用して、デバッグシステムへのインタフェースとして C-SPY I-jet/JTAGjet ドライバを指定します。

［プロジェクト］>［オプション］>［デバッガ］>［ドライバ］

## --jet_script_file

**構文**

```
--jet_script_file=path
```

**パラメータ**

| | |
|---|---|
| *path* | スクリプトによるリセット方式が記述されているファイルのパス。 |

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ

C-SPY CMSIS-DAP ドライバ

**説明**

このオプションでは、使用可能なスクリプトによるリセット方式を記述するファイルがあればそれを指定します。

**関連項目**

517 ページの *--reset_style*、502 ページの *--jet_standard_reset*。

このオプションを設定するには、［プロジェクト］>［オプション］>［デバッガ］>［追加オプション］を使用します。

## --jet_standard_reset

**構文**

```
--jet_standard_reset=strategy,duration,delay
```

**パラメータ**

*strategy*

リセット方式。以下から選択します。

0: リセットは無効

1: ソフトウェアリセット

2: ハードウェアリセット

3: コアリセット

4: システムリセット

--jet_script_file で指定したファイルに存在し、対応する --reset_style のインスタンスにより定義されている場合は、以下のリセット方式が使用できます:

5: カスタムリセット

6: ウォッチドッグまたはリセットレジスタによるリセット

7: ブートローダの後にリセットして停止

8: ブートローダの前にリセットして停止

9: リセット中に接続

*duration*

デバイスをリセットするためにハードウェアリセットがリセット信号（nSRST/nRESET 行）（低）をアサートする時間（ミリ秒）。

一部のデバイスでは、デフォルトの 200 ms よりも長いリセット信号が必要な場合があります。

このパラメータはハードウェアリセット、およびハードウェアリセットを使用するカスタムのリセット方式に適用されます。

*delay* リセット信号のアサートが取り消されてから、デバッガがプロセッサを制御しようとするまでの遅延時間（ミリ秒）。

外部のリセット信号のアサートが取り消された後、プロセッサが内部的にリセットのままになって、デバッガからアクセスできないことがあります。

このパラメータはハードウェアリセット、およびハードウェアリセットを使用するカスタムのリセット方式に適用されます。

**適用範囲** C-SPY I-jet/JTAGjet ドライバ

C-SPY CMSIS-DAP ドライバ

**説明** このオプションでは、デバッガの開始時に使用するリセット方式を選択します。Cortex-M の場合、他のデバイスとは異なる方式を使用します。

**関連項目** 517 ページの *--reset_style*、501 ページの *--jet_script_file*。

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet］>［設定］>［リセット］**

## --jet_startup_connection_timeout

**構文** `--jet_startup_connection_timeout=`*milliseconds*

**パラメータ**

*milliseconds* ミリ秒単位の時間。

**適用範囲** C-SPY I-jet/JTAGjet ドライバ

C-SPY CMSIS-DAP ドライバ

**説明** このオプションを使用して、C-SPY ドライバがターゲットボードに接続しようとする時間を延長します。

このオプションを設定するには、**［プロジェクト］>［オプション］>［デバッガ］>［追加オプション］**を使用します。

## --jet_swo_on_d0

**構文**

```
--jet_swo_on_d0
```

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ。

**説明**

このオプションを使用して、トレースデータピンの D0 に SWO トレースデータを出力するよう指定します。このオプションを使用するときは、SWD と JTAG インタフェースが SWO トレースデータを処理できます。

［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**I-jet/JTAGjet**］>［**SWO**］>［**TraceD0 ピンの SWO**］

## --jet_swo_prescaler

**構文**

```
--jet_swo_prescaler=number
```

**パラメータ**

| | |
|---|---|
| *number* | プリスケーラの値 (1–100)。これによって CPU クロック周波数が決まります。 |

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ。

**説明**

このオプションを使用して、SWO クロックのプリスケーラを指定します。CPU クロック周波数は、プリスケーラとして指定された数値で除算されます。通信中にデータパケットが失われる場合、より大きいプリスケーラの値を使用してみてください。

このオプションを指定しない場合、プリスケーラの値は自動的に設定されます。この自動的に設定される値は、デバッグプローブで使用できる最高の周波数になります。

［**I-jet/JTAGjet**］>［**SWO 設定**］>［**SWO プリスケーラ**］

## --jet_swo_protocol

**構文**

```
--jet_swo_protocol={auto|Manchester|UART}
```

**パラメータ**

| | |
|---|---|
| `auto` | 通信プロトコルを自動的に選択します。 |
| `Manchester` | Manchester プロトコルを指定します。 |
| `UART` | UART プロトコルを指定します。 |

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ。

**説明**

このオプションを使用して、SWO チャンネルの通信プロトコルを指定します。このオプションを指定しない場合、`auto` が自動的に使用されます。

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet］>［SWO］>［プロトコル］**

## --jet_tap_position

**構文**

```
--jet_tap_position=tap_number|multidrop_id
```

**パラメータ**

| | |
|---|---|
| *tap_number* | 接続先のデバイスの TAP 位置。 |
| *multidrop_id* | マルチドロップシステムのターゲット ID。 |

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ

C-SPY CMSIS-DAP ドライバ

**説明**

JTAG インタフェースを使用し、JTAG スキャンチェーン上に複数のデバイスがある場合、このオプションを使用して特定のデバイスを選択します。SWD インタフェースを使用して、ボードにマルチドロップ SWD システムがある場合、このオプションを使用してターゲット ID を選択します。

**関連項目**

550 ページの *I-jet/JTAGjet の JTAG/SWD オプション*。

**［プロジェクト］>［オプション］>［デバッガ］>［I-jet/JTAGjet］>［JTAG/SWD］>［明示的なプローブ設定］>［ターゲット番号（TAP またはマルチドロップ ID）］**

## --jlink_dcc_timeout

**構文**

```
--jlink_dcc_timeout=milliseconds
```

**パラメータ**

| | |
|---|---|
| *milliseconds* | タイムアウト（ミリ秒）。有効な範囲は 5 から 5000 で、デフォルト値は 100 ミリ秒です。 |

**適用範囲**

C-SPY J-Link/J-Trace ドライバ。

**説明**

このオプションを使用して、C-SPY からターゲット上の DCC エージェントへの保留中のリクエストについてタイムアウトを指定します。

このオプションを設定するには、**［プロジェクト］> ［オプション］> ［デバッガ］> ［追加オプション］**を使用します。

## --jlink_device_select

**構文**

```
--jlink_device_select=tap_number
```

**パラメータ**

| | |
|---|---|
| *tap_number* | 接続先のデバイスの TAP 位置。 |

**適用範囲**

C-SPY J-Link/J-Trace ドライバ。

**説明**

JTAG スキャンチェーンに複数のデバイスがある場合に、このオプションを使用してデバイスを選択します。

**関連項目**

550 ページの *I-jet/JTAGjet の JTAG/SWD オプション*。

**［プロジェクト］> ［オプション］> ［デバッガ］> ［J-Link/J-Trace］> ［接続］> ［JTAG スキャンチェーン］> ［TAP 番号］**

## --jlink_exec_command

**構文**

```
--jlink_exec_commmand=cmdstr1; cmdstr2; cmdstr3 ...
```

**パラメータ**

| | |
|---|---|
| *cmdstrn* | J-Link/J-Trace コマンド文字列。 |

**適用範囲**

C-SPY J-Link/J-Trace ドライバ。

**説明**

このオプションは、ターゲットの接続が完了した後に、デバッガで 1 つまたは複数のコマンド文字列を指定して `__jlinkExecCommand` マクロを呼び出すときに使用します。

**関連項目**

427 ページの *__jlinkExecCommand*。

このオプションを設定するには、**［プロジェクト］> ［オプション］> ［デバッガ］> ［追加オプション］**を使用します。

## --jlink_initial_speed

**構文**

```
--jlink_initial_speed=speed
```

**パラメータ**

| | |
|---|---|
| *speed* | 最初の通信速度 (kHz)。速度を指定しない場合、最初の速度として 32kHz が使用されます。 |

**適用範囲**

C-SPY J-Link/J-Trace ドライバ。

**説明**

このオプションは、最初の JTAG 通信速度 (kHz) を指定するときに使用します。

**関連項目**

556 ページの *J-Link/J-Trace の設定オプション*。

**［プロジェクト］> ［オプション］> ［デバッガ］> ［J-Link/J-Trace］> ［設定］> ［JTAG/SWD 速度］> ［固定］**

## --jlink_ir_length

**構文**

```
--jlink_ir_length=length
```

**パラメータ**

| | |
|---|---|
| *length* | 接続先 ARM コアの前の IR ビット数で、ARM デバイスと他のデバイスが混在する JTAG スキャンチェーン用です。 |

**適用範囲**

C-SPY J-Link/J-Trace ドライバ。

**説明**

このオプションは、接続先 ARM コアの前の IR ビット数を設定します。

**関連項目**

561 ページの *J-Link/J-Trace 接続オプション*。

［プロジェクト］>［オプション］>［デバッガ］>［J-Link/J-Trace］>［接続］>［JTAG スキャンチェーン］>［先行ビット］

## --jlink_reset_strategy

**構文**

```
--jlink_reset_strategy=delay,strategy
```

**パラメータ**

| | |
|---|---|
| *delay* | Cortex-M および ARM 7/9/11 で方式が 1-9 の場合、*delay* は 0（無視）に設定してください。ARM 7/9/11 で方式が 0 の場合、delay の値は 0-10000 のいずれかにする必要があります。 |
| *strategy* | サポートされているリセット方式について詳しくは、*『ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド』*を参照してください。 |

**適用範囲**

C-SPY J-Link/J-Trace ドライバ。

**説明**

このオプションは、デバッガの起動時に使用するリセット方法を選択するときに使用します。

**関連項目**

556 ページの *J-Link/J-Trace の設定オプション*。

［プロジェクト］>［オプション］>［デバッガ］>［J-Link/J-Trace］>［設定］>［リセット］

## --jlink_script_file

**構文**

```
--jlink_script_file=filename
```

**パラメータ**

| | |
|---|---|
| *filename* | J-Link スクリプトファイル名。 |

**適用範囲**

C-SPY J-Link/J-Trace ドライバ。

**説明**

このオプションは、使用する J-Link スクリプトファイルを指定するときに使用します。

J-Link には、ハードウェアの設定に使用できるスクリプト言語が用意されています。特定のターゲットについては、既成のスクリプトファイルが IAR Embedded Workbench によって自動的に指定されます。コマンドラインモード

では、このオプションを使用してスクリプトファイルを手動で指定する必要があります。

**関連項目**

スクリプト言語について詳しくは、*『J-Link/J-Trace ARM ユーザガイド』*（JLinkARM.pdf、ドキュメント番号 UM08001）のセクション 5.10。

事前に定義されていないスクリプトファイルを使用してこのオプションを設定するには、**［プロジェクト］>［オプション］>［デバッガ］>［追加オプション］**を使用します。

## --jlink_trace_source

**構文**

```
--jlink_trace_source={ETB|ETM}
```

**パラメータ**

| | |
|---|---|
| ETB | ETB トレースを選択します。 |
| ETM | ETM トレースを選択します。 |

**適用範囲**

C-SPY J-Link/J-Trace ドライバ。

**説明**

このオプションを使用して、ETB または ETM をトレースのソースとして選択します。

**注 :** このオプションは、J-Trace にのみ適用されます。

**関連項目**

556 ページの *J-Link/J-Trace の設定オプション*。

**［プロジェクト］>［オプション］>［デバッガ］>［J-Link/J-Trace］>［設定］>［ETM/ETB］**

## --leave_running

**構文**

```
--leave_running
```

このオプションはコマンドライン上で --backend オプションより前に配置しなければならない点に注意してください。

**適用範囲**

cspybat

**説明**

cspybat によりターゲット上で実行を開始し、終了した後もターゲットを実行したままにします。

関連オプションを設定するには、以下のように選択します。

**[プロジェクト] > [オプション] > [デバッガ] > [実行中のターゲットにアタッチ]**

## --macro

**構文**

```
--macro filename
```

このオプションはコマンドライン上で `--backend` オプションより前に配置しなければならない点に注意してください。

**パラメータ**

*filename*

使用する C-SPY マクロファイル（ファイル名拡張子 `mac`）。

**適用範囲**

`cspybat`

**説明**

このオプションは、ターゲットアプリケーションを実行する前にロードする C-SPY マクロファイルを指定するときに使用します。このオプションは、1 つのコマンドラインで複数個使用できます。

**関連項目**

394 ページの *C-SPY マクロの使用の概要*。

**[プロジェクト] > [オプション] > [デバッガ] > [設定] > [セットアップマクロ] > [マクロファイルの使用]**

## --macro_param

**構文**

```
--macro_param [param=value]
```

このオプションは、コマンドライン上で `--backend` オプションより前に配置しなければならない点に注意してください。

**パラメータ**

*param = value*

*param* は、`__param` C-SPY マクロ構造を使用して定義するパラメータで、`value` は値を示します。

**適用範囲**

`cspybat`

**説明**

このオプションを使用して、C-SPY マクロパラメータに値を割り当てます。このオプションはコマンドライン上で複数回使用することができます。

関連項目　402 ページの *マクロパラメータ*。


［プロジェクト］>［オプション］>［デバッガ］>［追加オプション］

## --mac_handler_address

構文　`--mac_handler_address=`*`address`*

パラメータ

| | |
|---|---|
| *address* | デバッガハンドラのメモリエリアの開始アドレス。 |

適用範囲　C-SPY Macraigor ドライバ。

説明　このオプションでは、Intel XScale デバイスが使用するデバッグハンドラの場所（メモリアドレス）を指定します。

関連項目　563 ページの *Macraigor*。


［プロジェクト］>［オプション］>［デバッガ］>［Macraigor］>［デバッグハンドラアドレス］

## --mac_jtag_device

構文　`--mac_jtag_device=`*`device`*

パラメータ

| | |
|---|---|
| *device* | 使用するハードウェアインタフェースに対応するデバイス。Macraigor `mpDemon`、`usbdemon`、`usb2demon` から選択します。 |

適用範囲　C-SPY Macraigor ドライバ。

説明　このオプションは、使用するハードウェアインタフェースに対応するデバイスを選択するときに使用します。

関連項目　563 ページの *Macraigor*。


［プロジェクト］>［オプション］>［デバッガ］>［Macraigor］>［OCD インタフェースデバイス］

## --mac_multiple_targets

**構文**

```
--mac_multiple_targets=<tap-no>@dev0,dev1,dev2,dev3,...
```

**パラメータ**

| | |
|---|---|
| *tap-no* | 接続先のデバイスの TAP 番号です。最初のデバイスに接続する場合は `0`、2 番目のデバイスに接続する場合は `1` などとします。 |
| *dev0-devn* | Macraigor JTAG プローブ上で最も近い TDO ピン。 |

**適用範囲**

C-SPY Macraigor ドライバ。

**説明**

JTAG スキャンチェーンに複数のデバイスがある場合、それぞれのデバイスを定義する必要があります。このオプションは、接続先のデバイスを指定するときに使用します。

**例**

```
--mac_multiple_targets=0@ARM7TDMI,ARM7TDMI
```

**関連項目**

563 ページの *Macraigor*。

**[プロジェクト] > [オプション] > [デバッガ] > [Macraigor] > [JTAG スキャンチェーン（マルチターゲット)]**

## --mac_reset_pulls_reset

**構文**

```
--mac_reset_pulls_reset=time
```

**パラメータ**

| | |
|---|---|
| *time* | `0-2000`。リセット後の遅延（ミリ秒）です。 |

**適用範囲**

C-SPY Macraigor ドライバ。

**説明**

このオプションは、デバッガの起動時に最初のハードウェアリセットを C-SPY で実行するときに使用します。また、リセット後の遅延を指定します。

**関連項目**

563 ページの *Macraigor*。

**[プロジェクト] > [オプション] > [デバッガ] > [Macraigor] > [ハードウェアリセット]**

## --mac_set_temp_reg_buffer

**構文**

```
--mac_set_temp_reg_buffer=address
```

**パラメータ**

| | |
|---|---|
| *address* | RAM エリアの開始アドレス。 |

**適用範囲**

C-SPY Macraigor ドライバ。

**説明**

このオプションは、MMU の制御および CP15 コプロセッサ経由のキャッシュに使用する RAM エリアの開始アドレスを指定するときに使用します。

このオプションを設定するには、**[プロジェクト] > [オプション] > [デバッガ] > [追加オプション]** を使用します。

## --mac_xscale_ir7

**構文**

```
--mac_xscale_ir7
```

**適用範囲**

C-SPY Macraigor ドライバ。

**説明**

このオプションは、XScale ir5 コアに代えて XScale ir7 コアを使用することを指定するときに使用します。XScale ir7 コアを使用する場合、このオプションは必須です。

以下の XScale コアが IAR C-SPY Macraigor ドライバでサポートされます。

Intel XScale Core 1（5 ビット命令レジスタ — ir5）

Intel XScale Core 2（7 ビット命令レジスタ — ir7）

このオプションを設定するには、**[プロジェクト] > [オプション] > [デバッガ] > [追加オプション]** を選択します。

## --mapu

**構文**

```
--mapu
```

**適用範囲**

C-SPY シミュレータドライバ。

**説明**

このオプションは、デバッグファイル内のセクション情報をメモリアクセスチェックに使用する場合に指定します。すると、実行中に未指定のメモリ範囲へのアクセスがシミュレータでチェックされます。こうしたアクセスが見

つかった場合、C 関数のコールスタックとメッセージが、stderr に出力されて、実行が停止します。

**関連項目**

176 ページの*メモリアクセスチェック*。

オプションを設定するには、以下のように選択します。

**［シミュレータ］>［メモリアクセス設定］**

## -p

**構文**

```
-p filename
```

**パラメータ**

*filename*

使用するデバイス記述ファイル。

**適用範囲**

すべての C-SPY ドライバ。

**説明**

このオプションは、使用するデバイス記述ファイルを指定するときに使用します。

**関連項目**

55 ページの*デバイス記述ファイルの選択*。

**［プロジェクト］>［オプション］>［デバッガ］>［設定］>［デバイス記述ファイル］**

## --plugin

**構文**

```
--plugin filename
```

このオプションはコマンドライン上で --backend オプションより前に配置しなければならない点に注意してください。

**パラメータ**

*filename*

使用するプラグインファイル（ファイル名拡張子 dll）。

**適用範囲**

cspybat

**説明**

特定の C/C++ 標準ライブラリ関数（printf など）を、実際のハードウェアデバイスの代わりに、C-SPY （[C-SPY Terminal I/O] ウィンドウなど）でサポートできます。このようなサポートを cspybat で有効にするには、arm¥bin ディレクトリにある armbat.dll という専用のプラグインモジュールを使用してください。

このオプションは、このプラグインをデバッグセッション中にインクルードするときに使用します。このオプションは、1 つのコマンドラインで複数個使用できます。

**注 :** このオプションは、別のプラグインモジュールのインクルードにも使用できますが、その場合の条件として、特にモジュールが cspybat で動作可能であることが必要となります。これは、common¥plugins ディレクトリにある C-SPY プラグインモジュールは、通常、cspybat で使用できないことを意味します。

［プロジェクト］>［オプション］>［デバッガ］>［プラグイン］

## --proc_stack_*stack*

**構文**

```
--proc_stack_stack=startaddress,endaddress
```

Cortex-M の場合、*stack* は、main または proc のいずれかです。

ここで、他の ARM コアでは *stack* は usr、svc、irq、fiq、und、abt のいずれかです。

**パラメータ**

| | |
|---|---|
| *startaddress* | スタックの開始アドレス。値または式として指定します。 |
| *endaddress* | スタックの終了アドレス。値または式として指定します。 |

**適用範囲**

すべての C-SPY ドライバ このコマンドラインオプションは、C-SPY を IDE から使用する場合のみに有効です。cspybat を使用したバッチモードでは使用できません。

**説明**

このオプションを使用して、C-SPY に予約済みスタックについての情報を提供します。デフォルトでは、C-SPY はこの情報をシステム起動コードから受け取りますが、何らかの理由でデフォルト値をオーバライドする場合に、このオプションを活用できます。

**例**

```
--proc_stack_irq=0x8000,0x80FF
```

このオプションを設定するには、［プロジェクト］>［オプション］>［デバッガ］>［追加オプション］を使用します。

## --rdi_allow_hardware_reset

**構文**

```
--rdi_allow_hardware_reset
```

**適用範囲**

C-SPY RDI ドライバ。

**説明**

このオプションは、ターゲットのハードウェアリセットをエミュレータで実行可能にするときに使用します。エミュレータでサポートされていることが必要です。

**関連項目**

565 ページの *RDI*。

**[プロジェクト] > [オプション] > [デバッガ] > [RDI] > [ハードウェアリセットを許可する]**

## --rdi_driver_dll

**構文**

```
--rdi_driver_dll filename
```

**パラメータ**

| | |
|---|---|
| *filename* | ドライバ DLL ファイルまたはそのパス。 |

**適用範囲**

C-SPY RDI ドライバ。

**説明**

このオプションは、JTAG ポッドに付属のドライバ DLL ファイルのパスを指定するときに使用します。

**関連項目**

565 ページの *RDI*。

**[プロジェクト] > [オプション] > [デバッガ] > [RDI] > [メーカー RDI ドライバ]**

JTAGjet の場合、このオプションは IDE では使用できません。

## --rdi_step_max_one

**構文**

```
--rdi_step_max_one
```

**適用範囲**

C-SPY Angel デバッグモニタドライバ。

C-SPY RDI ドライバ。

**説明**

このオプションは、命令を 1 つだけ実行するときに使用します。デバッガは、ステップ動作中、割込みをオフにします。また、必要に応じて、命令を実行せずに命令のシミュレーションを行います。

このオプションを設定するには、**[プロジェクト] > [オプション] > [デバッガ] > [追加オプション]** を使用します。

## --reset_style

**構文**

```
--reset_style="reset_id,reset_name,selected,menu_command"
```

**パラメータ**

| | |
|---|---|
| *reset_id* | リセット方式の番号 (0-9)。`--jet_standard_reset` の記述に従います。 |
| *reset_name* | リセット方式の名称。`--jet_script_file` で指定したファイルに従います。<br><br>内蔵のリセット方式の場合、このパラメータは - です。内蔵のリセット方式をオーバライドするには、リセットスクリプトファイルにラベルまたは関数名を入力します。 |
| *選択済* | 0 または 1。1 は、**[リセット]** ドロップダウンボタンのデフォルトのリセット方式を設定します。 |
| *menu_command* | **[リセット]** ドロップダウンメニューに表示されるリセット方式の名称。 |

**適用範囲**

C-SPY I-jet/JTAGjet ドライバ

C-SPY CMSIS-DAP ドライバ

**説明**

このオプションを使用して、デバッグ中に使用できるリセット方式を指定します（各方式に 1 つずつ）。

**例**

次の例ではスクリプトファイルを指定し、標準のリセット方式を設定して、デバッグ時に使用できるリセット方式を指定します。

```
--jet_script_file=myDir¥myProbeScriptFile
--jet_standard_reset=9,0,0
--reset_style="0,-,0,Disabled (no reset)"
--reset_style="1,-,0,Software"
--reset_style="2,-,0,Hardware"
--reset_style="3,-,0,Core"
--reset_style="4,-,0,System"
--reset_style="5,Custom,0,Custom reset"
--reset_style="9,ConnectUnderReset,1,Connect during reset"
```

**関連項目** 501 ページの *--jet_script_file* と 502 ページの *--jet_standard_reset*。

このオプションを設定するには、**[プロジェクト] > [オプション] > [デバッガ] > [追加オプション]** を使用します。

## --semihosting

**構文**

```
--semihosting={none|iar_breakpoint}
```

**パラメータ**

| | |
|---|---|
| パラメータはありません | 標準セミホスティングを使用します。 |
| `none` | セミホスティング I/O は使用されません。 |
| `iar_breakpoint` | IAR 独自のセミホスティングが使用されます。 |

**適用範囲** すべての C-SPY ドライバ。

**説明** このオプションは、セミホスティング I/O を有効にし、使用するセミホスティングインタフェースの種類を選択するときに使用します。このオプションを使用しない場合、デフォルトでセミホスティングが有効化され、C-SPY は正しいセミホスティングモードを自動的に選択するように動作します。すなわち、アプリケーションがセミホストとリンクされている場合には、通常、このオプションを使用する必要はありません。

セミホスティングが動作するためには、アプリケーションがセミホスティングライブラリとリンクされていることが必要です。

**関連項目** セミホスティングとのリンクに関する詳細については、*『ARM 用 IAR C/C++ 開発ガイド』* を参照してください。

**[プロジェクト] > [オプション] > [一般オプション] > [ライブラリ構成]**

## --silent

**構文**

```
--silent
```

このオプションはコマンドライン上で `--backend` オプションより前に配置しなければならない点に注意してください。

**適用範囲**

`cspybat`

**説明**

このオプションは、サインオンメッセージを省略するときに使用します。

このオプションは、IDE では使用できません。

## --stlink_reset_strategy

**構文**

```
--stlink_reset_strategy=delay,strategy
```

**パラメータ**

| | |
|---|---|
| *delay* | 遅延時間（ミリ秒）。*delay* は無視されるため、`0` にしてください。 |
| *strategy* | リセット方式。<br>`0`:（通常）標準のリセットの手順を実行します。<br>`1`:（リセットピン）リセットピンを使用してハードウェアリセットを実行します。ST-LINK バージョン 2 でのみ使用できます。<br>`2`:（リセット中に接続）ST-LINK は、リセットをアクティブにしたままでターゲットに接続します（リセットは「低」になり、ターゲットに接続中はそのままになります）。ST-LINK バージョン 2 でのみ使用できます。 |

**適用範囲**

C-SPY ST-LINK ドライバ。

**説明**

このオプションは、デバッガの起動時に使用するリセット方法を選択するときに使用します。

**関連項目**

567 ページの *ST-LINK*。

**［プロジェクト］>［オプション］>［デバッガ］>［ST-LINK］>［設定］>［リセット］**

## --timeout

**構文**

```
--timeout milliseconds
```

このオプションはコマンドライン上で --backend オプションより前に配置しなければならない点に注意してください。

**パラメータ**

*milliseconds*

実行が停止するまでの時間（ミリ秒）。

**適用範囲**

cspybat

**説明**

このオプションを使用して、最長実行時間を制限します。

このオプションは、IDE では使用できません。

## --xds_rootdir

**構文**

```
--xds_rootdir=path
```

**適用範囲**

C-SPY TI XDS ドライバ

**説明**

このオプションは、TI XDS ドライバパッケージがインストールされるディレクトリのパスを指定するときに使用します。

このオプションを設定するには、**[プロジェクト]** > **[オプション]** > **[デバッガ]** > **[追加オプション]** を使用します。

# フラッシュローダ

- フラッシュローダの概要
- フラッシュローダの使用
- フラッシュローダについてのリファレンス情報

## フラッシュローダの概要

フラッシュローダは、ターゲットにダウンロードされるエージェントです。デバッガからアプリケーションをフェッチして、フラッシュメモリにプログラムします。フラッシュローダでは、ファイル I/O 機構を使用してホストからアプリケーションプログラムを読み込みます。1 つまたは複数のフラッシュローダを選択できます。各フラッシュローダでは、アプリケーションの選択した部分をロードします。すなわち、異なるフラッシュローダを使用して、アプリケーションのさまざまな部分をロードすることができます。

さまざまなマイクロコントローラに対するフラッシュローダセットが、ARM 用 IAR Embedded Workbench に用意されています。これらのローダに加えて、多くのフラッシュローダがチップメーカおよびサードパーティベンダから提供されています。独自のフラッシュローダを実装できるように、フラッシュローダの API、ドキュメント、およびいくつかの実装例が使用できます。

## フラッシュローダの使用

以下のトピックについて説明します:

- フラッシュローダの設定
- フラッシュローディング機構
- フラッシュローダの中止

### フラッシュローダの設定

**アプリケーションのダウンロードにフラッシュローダを使用するには、以下の手順に従います。**

1 ［**プロジェクト**］>［**オプション**］を選択します。

2 ［**デバッガ**］カテゴリを選択して、［**ダウンロード**］タブをクリックします。

3 ［**フラッシュローダを使用する**］オプションを選択します。指定したデバイスに設定されたデフォルトのフラッシュローダが使用されます。設定は、事前に定義された board ファイルで指定します。

4 デフォルトのフラッシュローダをオーバライドしたり、自分のボードに合わせてデフォルトのフラッシュローダの動作を変更するには、［**デフォルトの .board ファイルのオーバライド**］オプションに続いて［**編集**］を選択し、［**フラッシュローダの構成**］ダイアログボックスを開きます。.board ファイルのコピーがプロジェクトディレクトリに作成され、それに従って .board ファイルのパスが更新されます。

5 ［**フラッシュローダの概要**］ダイアログボックスでは、現在設定されているフラッシュローダをすべて一覧表示します。523 ページの*［フラッシュローダの概要］ダイアログボックス*を参照してください。フラッシュローダを選択できます。あるいは、［**フラッシュローダの構成**］ダイアログボックスを開くことができます。

［**フラッシュローダの構成**］ダイアログボックスでは、ダウンロードを設定できます。さまざまなフラッシュローダオプションの詳細については、525 ページの*［フラッシュローダの構成］ダイアログボックス*を参照してください。

## フラッシュローディング機構

［**フラッシュローダを使用する**］オプションが選択され、1 つまたは複数のフラッシュローダが設定されると、デバッグセッションの開始時に以下の手順が実行されます。

フラッシュローダ設定のフラッシュローダごとに、手順 1 から 4 が実行されます。

1 C-SPY では、フラッシュローダをターゲット RAM にダウンロードします。

RAM のサイズおよびアプリケーションイメージのサイズに応じて、手順 2 から 4 が 1 回以上実行されます。

2 C-SPY はアプリケーションイメージからのコード / データを、ターゲット RAM（RAM バッファ）に書き込みます。

3 C-SPY では、フラッシュローダの実行を開始します。

4 フラッシュローダは、RAM バッファからデータを読み取り、フラッシュメモリをプログラムします。

5 アプリケーションイメージがフラッシュメモリに書き込まれ、起動可能になります。以降、フラッシュローダーと RAM バッファは不要になり、RAM はすべてフラッシュメモリのアプリケーションに利用できます。

### フラッシュローダの中止

フラッシュローダを中止するには、以下の手順に従います。

1 Ctrl+Shift-（マイナス）をしばらく押します。

2 フラッシュローダが中止したというメッセージが、［デバッグログ］ウィンドウに表示されます。

この方法は、たとえば実行が適切な時間内に終了しないなど、実行に関して何か問題があると思われる場合に使用できます。

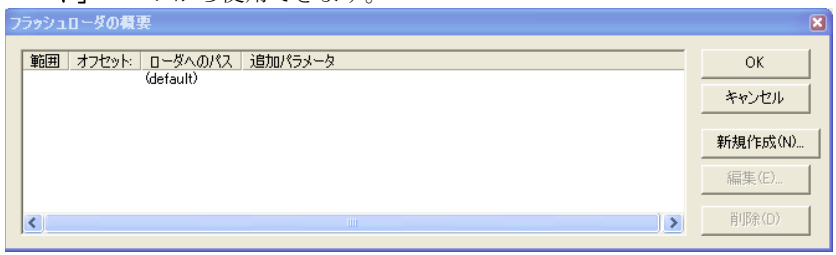
## フラッシュローダについてのリファレンス情報

リファレンス情報：


- 523 ページの *［フラッシュローダの概要］ダイアログボックス*
- 525 ページの *［フラッシュローダの構成］ダイアログボックス*


### ［フラッシュローダの概要］ダイアログボックス

**［フラッシュローダの概要］**ダイアログボックスは、**［デバッガ］＞［ダウンロード］**ページから使用できます。




このダイアログボックスには、定義済みのフラッシュローダがすべて一覧表示されます。**［一般オプション］＞［ターゲット］**ページでフラッシュローダのあるデバイスを選択した場合、デフォルトでは、このフラッシュローダは**［フラッシュローダの概要］**ダイアログボックスに一覧表示されます。

#### 表示エリア

表示エリアの各列には、メモリの特定の部分をフラッシュするフラッシュローダの設定が表示されます。

**範囲**

選択したフラッシュローダによってプログラミングされる、アプリケーションの部分。

**オフセット / アドレス**

アプリケーションがフラッシュされる、メモリの開始位置 アドレスの先頭に a が付いている場合は絶対アドレスです。それ以外の場合は、メモリの開始部分への相対オフセットです。

**ローダへのパス**

使用されるフラッシュローダ `*.flash` ファイルへのパス（古いスタイルのフラッシュローダの場合は `*.out`）

**追加パラメータ**

フラッシュローダに渡される追加パラメータの一覧

列のヘッダをクリックして、範囲、オフセット / アドレスなどの順にリストをソートします。

### 機能ボタン

以下の機能ボタンを使用できます。

**OK**

選択したフラッシュローダを使用して、アプリケーションをメモリにダウンロードします。

**キャンセル**

標準の「キャンセル」。

**新規作成**

使用するフラッシュローダを指定できるダイアログボックスが表示されます（525 ページの *［フラッシュローダの構成］ダイアログボックス*を参照）。

**編集**

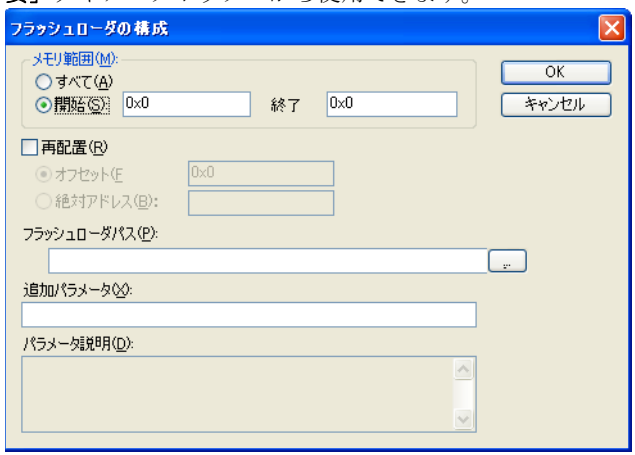
選択したフラッシュローダの設定を変更できるダイアログボックスが表示されます（525 ページの *［フラッシュローダの構成］ダイアログボックス*を参照）。

**削除**

選択されたフラッシュローダ設定を削除します。

# ［フラッシュローダの構成］ダイアログボックス

［フラッシュローダの構成］ダイアログボックスは、［フラッシュローダの概要］ダイアログボックスから使用できます。




［フラッシュローダの構成］ダイアログボックスを使用して、使用するボードに合ったダウンロードを設定します。デフォルトの board ファイルのコピーが、プロジェクトディレクトリに作成されます。

### メモリ範囲

フラッシュメモリにダウンロードするアプリケーションの部分を指定します。以下から選択します。

**すべて**

すべてのアプリケーションが、このフラッシュローダを使用してダウンロードされます。

**開始 / 終了**

アプリケーションをダウンロードするメモリエリアの開始と終了を指定します。

### 再配置

デフォルトのフラッシュベースアドレスをオーバライドします。つまりメモリ内のアプリケーション位置を再配置します。アプリケーションを、リンクされていた位置とは別の所にフラッシュできます。以下から選択します。

**オフセット**

相対オフセットの数値。このオフセットは、アプリケーションファイルのアドレスに追加されます。

**絶対アドレス**

アプリケーションがフラッシュされる、絶対ベースアドレスの数値。アプリケーションの最も小さいアドレスが、このアドレスに配置されます。絶対アドレスを指定する場合、アプリケーションに 1 つだけしかフラッシュロードを使用できない点に注意してください。

使用できる数値の形式は以下のとおりです。

- `123456`、10 進数
- `0x123456`、16 進数
- `0123456`、8 進数

最初のバイト（最も低いアドレス）をフラッシュに書き込むために使用されるデフォルトベースアドレスは、アプリケーション用のリンカ設定ファイルで指定されます。しかし、フラッシュベースアドレスをオーバライドして、アドレス空間の別の場所で起動することが必要な場合もあります。たとえば、フラッシュメモリの位置を再配置するデバイスに必要となることがあります。

**フラッシュローダパス**

このテキストボックスを使用して、ボード設定で使用するフラッシュローダファイル (`*.flash`) のパスを指定します。

**追加パラメータ**

フラッシュローダによっては、特別なオプションセットを独自に定義します。このテキストボックスを使用して、フラッシュローダを制御するオプションを指定します。使用可能なフラッシュロードについては、**[パラメータ説明]** フィールドを参照してください。

**パラメータ説明**

**[追加パラメータ]** テキストボックスで指定された追加パラメータの説明が表示されます。

# パート 4. 追加リファレンス情報

*ARM 用 C-SPY® デバッガガイド*のこのパートは、以下の章で構成されています。

- ［デバッガ］オプション
- C-SPY ドライバについての追加情報

# ［デバッガ］オプション

- デバッガオプションの設定
- デバッガオプションのリファレンス情報
- C-SPY ハードウェアデバッガドライバオプションのリファレンス情報

## デバッガオプションの設定

C-SPY デバッガを起動する前に、C-SPY の一般オプションとターゲットシステムで必要なオプション（C-SPY ドライバ固有のオプション）を設定しなければならない場合があります。このセクションでは、［**デバッガ**］のオプションについて説明します。

**IDE のデバッガオプションを設定するには、以下の手順に従います。**

1 ［**プロジェクト**］>［**オプション**］を選択して、［**オプション**］ダイアログボックスを開きます。

2 ［**カテゴリ**］リストで［**デバッガ**］を選択します。

汎用オプションの詳細は、530 ページの*デバッガオプションのリファレンス情報*を参照してください。

3 ［**設定**］ページで、［**ドライバ**］ドロップダウンリストから適切な C-SPY ドライバを選択します。

4 ドライバ固有オプションを設定するには、［**カテゴリ**］リストから該当するドライバを選択します。使用している C-SPY ドライバによっては、異なるオプションが使用できます。

| C-SPY ドライバ | 使用可能なオプションのページ |
|---|---|
| C-SPY Angel デバッグモニタドライバ | 538 ページの *Angel* |
| C-SPY GDB サーバドライバ | 544 ページの *GDB サーバ*<br>164 ページの *［ブレークポイント］ダイアログボックス* |
| C-SPY IAR ROM モニタドライバ | 545 ページの *IAR ROM モニタ* |

*表 50: 使用する C-SPY ドライバに固有のオプション*

| C-SPY ドライバ | 使用可能なオプションのページ |
|---|---|
| C-SPY CMSIS-DAP ドライバ | 539 ページの *CMSIS-DAP の設定オプション*<br>542 ページの *CMSIS-DAP の JTAG/SWD オプション*<br>164 ページの *［ブレークポイント］ダイアログボックス* |
| C-SPY I-jet/JTAGjet ドライバ | 546 ページの *I-jet/JTAGjet の設定オプション*<br>550 ページの *I-jet/JTAGjet の JTAG/SWD オプション*<br>552 ページの *I-jet/JTAGjet のトレースオプション*<br>164 ページの *［ブレークポイント］ダイアログボックス* |
| C-SPY J-Link/J-Trace ドライバ | 556 ページの *J-Link/J-Trace の設定オプション*<br>561 ページの *J-Link/J-Trace 接続オプション*<br>164 ページの *［ブレークポイント］ダイアログボックス* |
| C-SPY TI Stellaris ドライバ | 568 ページの *TI Stellaris の設定オプション* |
| C-SPY TI XDS ドライバ | 569 ページの *TI XDS の設定オプション* |
| C-SPY Macraigor ドライバ | 563 ページの *Macraigor* |
| RDI ドライバ | 565 ページの *RDI* |
| ST-LINK ドライバ | 567 ページの *ST-LINK* |
| サードパーティ製ドライバ | 570 ページの *サードパーティ製ドライバのオプション* |

*表 50: 使用する C-SPY ドライバに固有のオプション（続き）*

5 すべての設定をデフォルトの出荷時の設定に戻すには、［**工場出荷時設定**］ボタンをクリックします。

6 必要なオプションをすべて設定した後は、［**オプション**］ダイアログボックスで **[OK]** をクリックします。

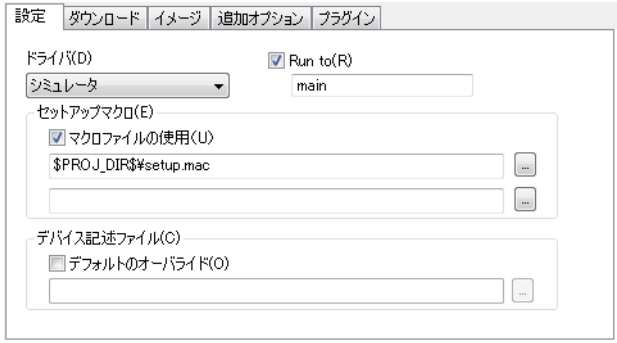
# デバッガオプションのリファレンス情報

リファレンス情報 :

- 設定
- ダウンロード
- イメージ
- 追加オプション
- プラグイン

# 設定

［設定］オプションは、使用する C-SPY ドライバ、セットアップマクロファイル、デバイス記述ファイルと、デフォルトでソースコードのどの位置まで実行するかを指定します。




### ドライバ

ターゲットシステムの C-SPY ドライバを選択します。

### 指定位置まで実行

リセット後にデバッガを起動したときに、C-SPY をどこまで実行するかを指定します。デフォルトでは、C-SPY は `main` 関数まで実行します。

デフォルトの位置をオーバライドするには、C-SPY の実行先となる別の位置名を指定してください。アセンブララベルかそれに相当するもの（関数名など）を指定できます。

オプションを選択していない場合は、リセットごとにプログラムカウンタに通常のハードウェアリセットアドレスが格納されます。

### セットアップマクロ

C-SPY 起動シーケンスのセットアップマクロファイルの内容を登録します。［マクロファイルの使用］を選択して、セットアップファイルのパスと名前を指定します。たとえば、`SetupSimple.mac` とします。拡張子を指定していない場合は、`mac` が使用されます。参照ボタンを使用して選択することもできます。

最大 2 つの異なるマクロファイルを指定できます。

### デバイス記述ファイル

デフォルトのデバイス記述ファイル（IAR 固有の `ddf` ファイルまたは CMSIS システムビュー記述ファイル）が、プロジェクトの設定に基づいて自動的に

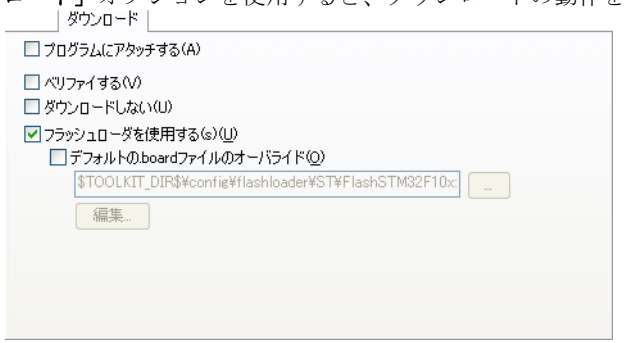
選択されます。デフォルトのファイルをオーバライドするには、**[デフォルトのオーバライド]** を選択し、他のファイルを指定します。参照ボタンを使用して選択することもできます。

デバイス記述ファイルの詳細については、『59 ページの*デバイス記述ファイルの修正*』を参照してください。

各 arm デバイスの IAR 固有のデバイス記述ファイルは、`arm¥config` ディレクトリにあり、ファイル名の拡張子は `ddf` です。

## ダウンロード

デフォルトでは、デバッグセッションが起動したときに、C-SPY によってアプリケーションが RAM またはフラッシュにダウンロードされます。**[ダウンロード]** オプションを使用すると、ダウンロードの動作を変更できます。




### 実行中のターゲットにアタッチ

ターゲットシステムのリセットや中止（J-Link および I-jet/JTAGjet のみ）をしないで、現在の位置で実行中のアプリケーションにデバッガをアタッチします。このオプションの使用時に予期しない動作を回避するには、**[デバッガ]** > **[設定]** オプションの **[指定位置まで実行]** の選択を解除してください。

### ベリファイする

ダウンロードしたコードイメージがターゲットメモリからリードバックでき、その内容が正しいことを確認します。

### ダウンロードを中止する

現在のフラッシュの内容を保持しながら、コードのダウンロードを無効にします。このコマンドは、ターゲットメモリにすでに格納されているアプリケーションをデバッグする場合に便利です。

このオプションと［**ベリファイする**］オプションを組み合せると、デバッガは不揮発性メモリからコードイメージをリードバックして、デバッグしたアプリケーションと同一かどうかをベリファイします。

### フラッシュローダを使用する

このオプションを使用して、フラッシュメモリへアプリケーションをダウンロードするときに 1 つまたは複数のフラッシュローダを使用します。フラッシュローダが、選択したチップで使用可能であれば、デフォルトで使用されます。［**編集**］ボタンを押し、［**フラッシュローダを使用する**］ダイアログボックスを表示します。

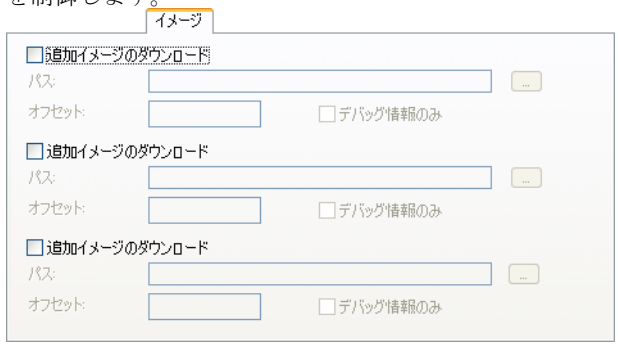
フラッシュローダオプションの詳細については、521 ページの *フラッシュローダ*を参照してください。

### デフォルトの .board ファイルのオーバライド

デフォルトのフラッシュローダの選択は、［**一般オプション**］>［**ターゲット**］ページで選択したデバイスに基づいて行われます。デフォルトフラッシュローダをオーバライドするには、［**デフォルトの .board ファイルのオーバライド**］を選択し、使用するフラッシュロードのパスを指定します。参照ボタンを使用して選択することもできます。［**編集**］をクリックして、［**フラッシュローダの概要**］ダイアログボックスを表示します。詳細については、523 ページの *［フラッシュローダの概要］ダイアログボックス*を参照してください。

## イメージ

［イメージ］オプションは、ダウンロードする追加のデバッグファイルの使用を制御します。




**注 :** フラッシュロードは実行されません。［**イメージ**］オプションは、イメージを RAM にダウンロードするためだけに使用できます。

### 追加イメージのダウンロード

ダウンロードする追加のデバッグファイルの使用を制御します。

**パス**

ダウンロードするデバッグファイルを指定します。参照ボタンを使用して選択することもできます。

**オフセット**

ダウンロードしたデバッグファイルの目的地のアドレスを確定する整数を指定します。

**デバッグ情報のみ**

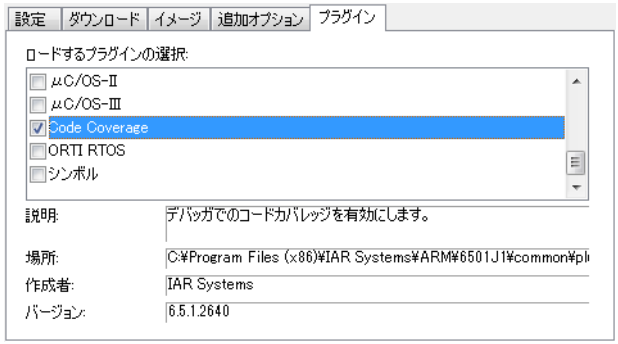
完全なデバッグファイルではなく、デバッグ情報のみをデバッガでダウンロードします。

4 つ以上のイメージをダウンロードするには、関連の C-SPY マクロを使用します（432 ページの *__loadImage* を参照）。

詳細については、58 ページの*複数イメージのロード*を参照してください。

## プラグイン

［**プラグイン**］オプションでは、デバッグセッションでロードして使用する C-SPY プラグインモジュールを選択します。




### ロードするプラグインの選択

デバッグセッション中にロードして使用可能にするプラグインモジュールを選択します。このリストには、製品のインストール時に同梱されたプラグインモジュールが含まれます。

### 説明

プラグインモジュールについて説明しています。

### 位置

プラグインモジュールの位置を知らせます。

一般プラグインモジュールは、common¥plugins ディレクトリに格納されます。ターゲット固有のプラグインモジュールは、arm¥plugins ディレクトリに格納されます。

### 作成者

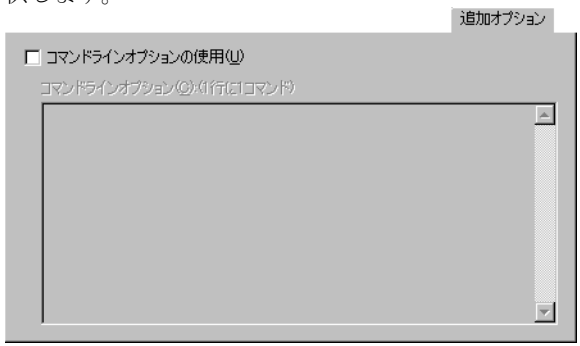
プラグインモジュールの提供元を示します。これは IAR システムズやサードパーティベンダなどです。

### バージョン

バージョン番号を示します。

## 追加オプション

［追加オプション］ページは、C-SPY へのコマンドラインインタフェースを提供します。




### コマンドラインオプションの使用

IDE で C-SPY への引渡しがサポートされていないコマンドライン引数を指定します。

/args オプションを使用して、コマンドライン引数をデバッグ済アプリケーションに引き渡すことは可能です。

構文：/args arg0 arg1 ...

/args を持つ行を複数使用できます。たとえば次のようになります。

```
/args --logfile log.txt
/args --verbose
```

/args を使用する場合、以下の変数をアプリケーションに定義する必要があります。

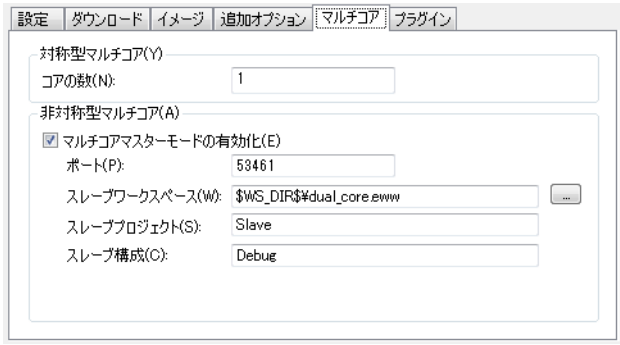
```
/* __argc, the number of arguments in __argv. */
__no_init int __argc;

/* __argv は、引数を保持する文字列へのポインタの配列で、パラメータ数に見合う大きさでなければなりません。*/
__no_init const char * __argv[MAX_ARGS];

/* __argvbuf は for __argv の記憶領域で、すべてのコマンドラインパラメータを保持するのに十分なサイズである必要があります。*/
__no_init __root char __argvbuf[MAX_ARG_SIZE];
```

## マルチコア

［マルチコア］オプションは、マルチコアのデバッグを構成します。




### コアの数

対称なマルチコアデバッグの場合、デバイス上のコア数を指定してください。

### マルチコアマスターモードの有効化

デバッグセッションを、非対称マルチコアデバッガマスターにします。デバッグセッションを開始すると、以下のいずれかのオプションを使用して IAR Embedded Workbench IDE の新しいインスタンスが起動します。

**ポート**

IDE インスタンス間の通信に使用する TCP ポート（通常は 1023 より大きいもの）を指定します。

**スレーブワークスペース**

スレーブインスタンスで開くワークスペースを指定します。

**スレーブプロジェクト**

スレーブインスタンスで開く、ワークスペース内のプロジェクト名を指定します。たとえば、プロジェクトのファイル名が `MySlaveProj.ewp` なら、`MySlaveProj` と指定します。

**スレーブ構成**

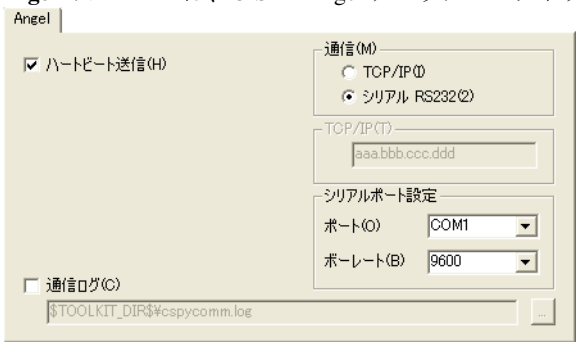
スレーブのデバッグ中に使用するビルド構成を指定します。たとえば、デバッグやリリースのように指定します。

## C-SPY ハードウェアデバッガドライバオプションのリファレンス情報

このセクションでは、以下のトピックについて説明します。

## Angel

**Angel** オプションは、C-SPY Angel デバッグモニタドライバを制御します。




### ハートビート送信

アプリケーションの実行中に C-SPY でターゲットシステムに定期的にポーリングを行います。ポーリングをすると、デバッガではターゲットアプリケーションが継続して実行しているか、または異常終了したかについて検出することができます。ハートビートを有効にすると、実行するプログラムから余分な CPU サイクルをある程度使用することになります。

### 通信

Angel の通信リンクを選択します。RS232 シリアルポート接続とイーサネット接続経由の TCP/IP がサポートされています。

### TCP/IP

ターゲットデバイスの IP アドレスをテキストボックスに指定します。

### シリアルポート設定

シリアルポートを設定します。以下を指定できます。

**ポート** Angel の通信リンクとして使用するホストコンピュータ上のポートを選択します。

**ボーレート** 通信速度を設定します。

Angel のシリアル初期速度は、常に 9600 baud です。最初のハンドシェイクの後に、リンク速度は指定した速度に変更されます。通信に関する問題は、非常に高速で発生する場合があります。Angel ベースのいくつかの評価ボードは 38,400 baud を超えては動作しません。

### 通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、Angel モニタプロトコルに対する十分な知識が必要です。

## CMSIS-DAP の設定オプション

**設定**オプションは、C-SPY CMSIS-DAP デバッグモニタドライバを制御します。


### リセット

デバッガの起動時に使用するリセット方法を選択します。**リセット**オプションは、Cortex-M デバイスでのみ使用可能な点に注意してください。使用するハードウェアに応じて、方式のいずれかがデフォルトになります。以下から選択します。

**無効（リセットなし）**リセットは実行されません。

**ソフトウェア**　PC をプログラムのエントリアドレスに設定します。これはソフトウェアリセットです。

**ハードウェア**

プローブはデバイスをリセットするために、JTAG コネクタの nSRST/nRESET のラインを切り替えます。通常はこうすることで周辺ユニットもリセットされます。リセットパルスのタイミングは、**［期間］**オプションと**［遅延（後）］**オプションによって制御されます。

プロセッサは、命令を実行する前にリセットハンドラで停止するはずです。一部のプロセッサはリセットベクタで停止しないことがありますが、何らかの命令を実行した後に停止します。

**コア**

コアは VECTRESET ビットでリセットされます。ペリフェラルユニットは影響を受けません。

**システム**

コアとペリフェラルをリセットします。

**リセット中に接続**

CMSIS-DAP は、リセットをアクティブにしたままターゲットに接続します。リセットは「Low」になり、ターゲットに接続中はそのままになります。

**カスタム**

デバイス固有のハードウェアリセット。デバイスによっては、デバッグを有効にしたり、命令を実行する前にプロセッサを停止するために、特別なリセットの手順やタイミングを必要とするものもあります。

ウォッチドッグタイマが無効になることがあります。

低消費電力モードなど、特殊なデバッグモードがオンになることがあります。

このオプションは一部のデバイスでのみ使用できます。

**ウォッチドッグまたはリセットレジスタによりリセット**

ソフトウェアリセットレジスタまたはウォッチドッグリセットを使用してプロセッサをリセットします。周辺ユニットがリセットされないことがあります。

ハードウェアリセットを使用してプロセッサをリセットベクタで停止できないときに、このリセット方式を推奨します。

デバイス固有のソフトウェアリセット。このオプションは一部のデバイスでのみ使用できます。

**ブートローダの後にリセットして停止**

一部のデバイスには、プロセッサがアプリケーションコードにジャンプする前に実行される ROM ブートローダがあります。このリセット方式は、ブートローダのコードを実行してアプリケーションコードの入口でプロセッサを停止するときに使用します。

デバイスに応じて、このリセット方式はハードウェア、コアまたはシステムリセットを使用して実装されます。

このオプションは一部のデバイスでのみ使用できます。

**ブートローダの前にリセットして停止**

このリセット方式は、**［ブートローダの後にリセットして停止］**の方式を補足するものです。デバイスに応じて、これはハードウェア、コアまたはシステムリセットを使用して実装されます。

このオプションは一部のデバイスでのみ使用できます。

これらの方式はすべて、JTAG および SWD インタフェースどちらにも使用できます。すべての方式は CPU をリセット後に停止します。

ターゲットのソフトウェアリセットを使用しても、ターゲットシステムの設定値を変更することはありません。プログラムカウンタをリセットするだけです。

一般的に、C-SPY リセットはソフトウェアリセットだけです。**［ハードウェア］**オプションを使用する場合、C-SPY では、デバッガの起動時に最初のハードウェアリセットを生成します。これはダウンロードの前に一度実行されます。**［フラッシュローダを使用する］**オプションが選択されている場合は、フラッシュダウンロード後にもう一度行われます。62 ページの *フラッシュのコードのデバッグ*および 63 ページの *RAM のコードのデバッグ*を参照してください。

ハードウェアリセットは、アプリケーションの低レベル設定が完全でないと、問題が発生する可能性があります。低レベル設定でメモリ構成とクロックを設定しないと、ハードウェアリセット後のアプリケーションは動作しません。C-SPY でこれを処理するには、セットアップマクロの `execUserReset()` 関数が適しています。同様な例（`execUserPreload()` を使用）については、60 ページの *メモリの再配置*を参照してください。

## リセット期間

デバイスをリセットするためにハードウェアリセットがリセット信号（`nSRST`/`nRESET` 行）（低）をアサートする時間（ミリ秒）。

一部のデバイスでは、デフォルトの 200 ms よりも長いリセット信号が必要な場合があります。

このオプションはハードウェアリセット、およびハードウェアリセットを使用するカスタムのリセット方式に適用されます。

**遅延（後）**

リセット信号のアサートが取り消されてから、デバッガがプロセッサを制御しようとするまでの遅延時間（ミリ秒）。

外部のリセット信号のアサートが取り消された後、プロセッサが内部的にリセットのままになって、デバッガからアクセスできないことがあります。

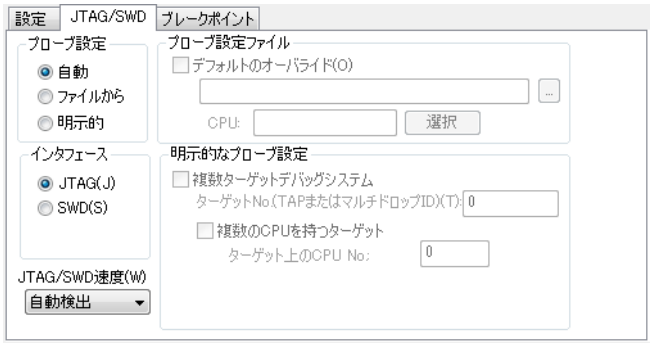
このオプションはハードウェアリセット、およびハードウェアリセットを使用するカスタムのリセット方式に適用されます。

**通信ログ**

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、インタフェースに対する十分な知識が必要です。

## CMSIS-DAP の JTAG/SWD オプション

［**JTAG/SWD**］オプションでは、CMSIS-DAP とターゲットシステム間のインタフェースを指定します。




**プローブ設定**

**自動**

CMSIS-DAP ドライバは自動的にターゲットの CPU を識別します。デフォルトのプローブ設定ファイルがある場合は、それを使用します。

これは、CPU が 1 つだけ存在する場合に正しく機能します。

**ファイルから**

プローブ設定ファイルのオーバライド、あるいは複数のターゲット CPU が必要となるよう指定します。

**明示的**

ターゲット CPU の検索方法を指定します。

### インタフェース

デバッグプローブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

| | |
|---|---|
| **JTAG** | JTAG インタフェースを使用します。 |
| **SWD** | SWD インタフェースを使用します。 |

### JTAG/SWD 速度

JTAG と SWD の通信速度を指定します。以下から選択します。

| | |
|---|---|
| **自動検出** | 信頼性の高い動作をするための最も高い周波数を自動的に使用します。 |
| **クロックに同期 (A)** | クロックをコアの外のプロセッサに同期します。RTCK JTAG 信号が使用可能な ARM デバイスでのみ機能します。 |
| ***n* MHz** | JTAG と SWD の通信速度を、選択した周波数に設定します。<br>JTAG 通信に関する問題や、ターゲットメモリへの書込みに関する問題がある場合（プログラムのダウンロード中など）、速度をより低い周波数に設定すると、これらの問題が解決できる可能性があります。 |

### プローブ設定ファイル

**デフォルトのオーバライド**

製品パッケージに付属のデフォルトのプローブ設定ファイルの代わりに使用するプローブ設定ファイルを指定します。

**選択**

ターゲット CPU の検索方法を指定します。

### 明示的なプローブ設定

**複数ターゲットデバッグシステム**

複数の CPU からなるデバッグシステムを指定します。

**ターゲット No.（TAP またはマルチドロップ ID）**

デバッグシステムがマルチドロップ SWD の場合に、CPU のある DAP のマルチドロップ ID（16 進数表記）を指定します。

デバッグシステムが JTAG スキャンチェーンの場合、接続先のデバイスの［**ターゲット番号 TAP**］（テストアクセスポート）位置を指定します。TAP 番号はゼロから始まります。TAP 位置に複数の CPU がある場合、［**ターゲット上の CPU No.**］も指定する必要があります。

**ターゲット上の CPU No.**

デバッグシステムが複数コアの SWD の場合、DAP の CPU 番号を指定してください。

## GDB サーバ

［**GDB サーバ**］オプションは、STR9-comStick 評価ボード用の C-SPY GDB サーバを制御します。


### TCP/IP アドレスまたはホスト名

GDB サーバの IP アドレスおよびポート番号を指定します。デフォルトでは、ポート番号 3333 が使用されます。TCP/IP 接続は、リモートコンピュータで動作する J-Link サーバに接続するために使用します。

### 通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、JTAG インタフェースに対する十分な知識が必要です。

# IAR ROM モニタ

**IAR ROM モニタ**オプションは、C-SPY IAR ROM モニタインタフェースを制御します。


## シリアルポート設定

シリアルポートを設定します。以下を指定できます。

**ポート** ROM の通信リンクとして使用するホストコンピュータ上のポートを選択します。

**ボーレート** 通信速度を設定します。シリアルポートの通信リンク速度は、ターゲットボードで選択された速度と一致する必要があります。

## 通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、ROM モニタプロトコルに対する十分な知識が必要です。

## I-jet/JTAGjet の設定オプション

［設定］オプションは、I-jet と I-jet Trace のインサーキットデバッグプローブおよび JTAGjet デバッグプローブを制御します。


### リセット

デバッガの起動時に使用するリセット方法を選択します。Cortex-M の場合、他のデバイスとは異なる方式を使用します。使用するハードウェアに応じて、方式のいずれかがデフォルトになります。以下から選択します。

**無効（リセットなし）** リセットは実行されません。

**ソフトウェア** PC をプログラムのエントリアドレスに、SP を初期スタックポインタの値に設定します。

これはソフトウェアリセットです。

**ハードウェア** プローブはデバイスをリセットするために、JTAG コネクタの nSRST/nRESET のラインを切り替えます。通常はこうすることで周辺ユニットもリセットされます。リセットパルスのタイミングは、**［期間］**オプションと**［遅延（後）］**オプションによって制御されます。

プロセッサは、命令を実行する前にリセットハンドラで停止するはずです。一部のプロセッサはリセットベクタで停止しないことがありますが、何らかの命令を実行した後に停止します。

**コア** コアは VECTRESET ビットでリセットされます。ペリフェラルユニットは影響を受けません。Cortex-M デバイスの場合のみ。

**システム**

AIRCR レジスタの SYSRESETREQ ビットを設定することにより、コアと周辺ユニットをリセットします。リセットベクタキャッチを使用して、最初の命令が実行される前にリセットベクタで CPU を停止します。Cortex-M デバイスの場合のみ。

**リセット中に接続**

I-jet/JTAGjet は、リセットをアクティブにしたままターゲットに接続します。リセットは「Low」になり、ターゲットに接続中はそのままになります。これは、STM32 デバイスの推奨リセット方式です。

**カスタム**

デバイス固有のハードウェアリセット。デバイスによっては、デバッグを有効にしたり、命令を実行する前にプロセッサを停止するために、特別なリセットの手順やタイミングを必要とするものもあります。

ウォッチドッグタイマが無効になることがあります。

低消費電力モードなど、特殊なデバッグモードがオンになることがあります。

このオプションは一部のデバイスでのみ使用できます。

**ウォッチドッグまたはリセットレジスタによりリセット**

ソフトウェアリセットレジスタまたはウォッチドッグリセットを使用してプロセッサをリセットします。周辺ユニットがリセットされないことがあります。

ハードウェアリセットを使用してプロセッサをリセットベクタで停止できないときに、このリセット方式を推奨します。

デバイス固有のソフトウェアリセット。このオプションは一部のデバイスでのみ使用できます。

**ブートローダの後にリセットして停止**

一部のデバイスには、プロセッサがアプリケーションコードにジャンプする前に実行される ROM ブートローダがあります。このリセット方式は、ブートローダのコードを実行してアプリケーションコードの入口でプロセッサを停止するときに使用します。

デバイスに応じて、このリセット方式はハードウェア、コアまたはシステムリセットを使用して実装されます。

このオプションは一部のデバイスでのみ使用できます。

**ブートローダの前にリセットして停止**

このリセット方式は、［**ブートローダの後にリセットして停止**］の方式を補足するものです。デバイスに応じて、これはハードウェア、コアまたはシステムリセットを使用して実装されます。

このオプションは一部のデバイスでのみ使用できます。

これらの方式はすべて、JTAG および SWD インタフェースどちらにも使用できます。すべての方式は CPU をリセット後に停止します。

ターゲットのソフトウェアリセットを使用しても、ターゲットシステムの設定値を変更することはありません。プログラムカウンタとモードレジスタ CPSR をリセット状態にするだけです。一部の ARM9、ARM11、Cortex-A のデバイスでは、効率的に仮想メモリ (MMU)、キャッシュ、メモリ保護を無効にして CP15 システム制御コプロセッサもリセットされます。

一般的に、C-SPY リセットはソフトウェアリセットだけです。［**ハードウェア**］オプションを使用する場合、C-SPY では、デバッガの起動時に最初のハードウェアリセットを生成します。これはダウンロードの前に一度実行されます。［**フラッシュローダを使用する**］オプションが選択されている場合は、フラッシュダウンロード後にもう一度行われます。62 ページの *フラッシュのコードのデバッグ*および 63 ページの *RAM のコードのデバッグ*を参照してください。

ハードウェアリセットは、アプリケーションの低レベル設定が完全でないと、問題が発生する可能性があります。低レベル設定でメモリ構成とクロックを設定しないと、ハードウェアリセット後のアプリケーションは動作しません。C-SPY でこれを処理するには、セットアップマクロの `execUserReset()` 関数が適しています。同様な例（`execUserPreload()` を使用）については、60 ページの*メモリの再配置*を参照してください。

### リセット期間

デバイスをリセットするためにハードウェアリセットがリセット信号（`nSRST`/`nRESET` 行）（低）をアサートする時間（ミリ秒）。

一部のデバイスでは、デフォルトの 200 ms よりも長いリセット信号が必要な場合があります。

このオプションはハードウェアリセット、およびハードウェアリセットを使用するカスタムのリセット方式に適用されます。

### 遅延（後）

リセット信号のアサートが取り消されてから、デバッガがプロセッサを制御しようとするまでの遅延時間（ミリ秒）。

外部のリセット信号のアサートが取り消された後、プロセッサが内部的にリセットのままになって、デバッガからアクセスできないことがあります。

このオプションはハードウェアリセット、およびハードウェアリセットを使用するカスタムのリセット方式に適用されます。

### ターゲット電源

ターゲットシステムの電源がプローブから供給される場合に、このオプションでデバッグ後の電源のステータスを指定します。以下から選択します。

**デバッグ後もオンにする** デバッグセッションの停止後もターゲットに電源を供給し続けます。

**デバッグ後にオフにする** デバッグセッションが停止すると、ターゲットの電源をオフにします。

### ETM よりも ETB/MTB を希望

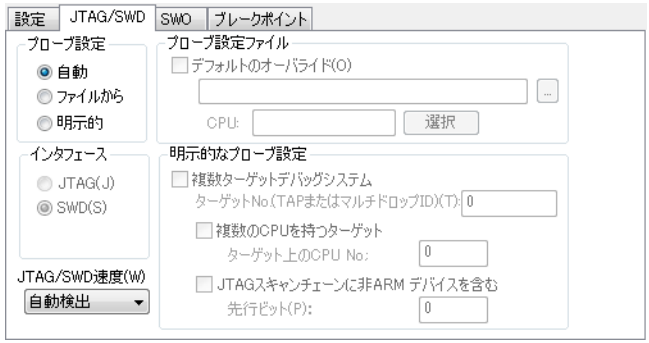
ETM ではなく ETB/MTB を使用します。これは通常、I-jet または I-jet Trace のインサーキットデバッグプローブを使用していて、オンチップの ETB または MTB がある場合に必要です。

### 通信ログ

トレース出力を記録します。これは、C-SPY とハードウェアに対する下位インタフェース間の内部的な通信アクティビティのシーケンスです。この記録は主に、IAR システムズのサポートに連絡する際のトラブルシューティングの参考情報として使用されます。

## I-jet/JTAGjet の JTAG/SWD オプション

**[JTAG/SWD]** オプションでは、I-jet、I-jet Trace、または JTAGjet とターゲットシステム間のインタフェースを指定します。




### プローブ設定

#### 自動

I-jet/JTAGjet ドライバが自動的にターゲットの CPU を識別します。デフォルトのプローブ構成ファイルが存在する場合はそれを使用します。

これは、CPU が 1 つだけ存在する場合に正しく機能します。

#### ファイルから

プローブ設定ファイルのオーバライド、あるいは複数のターゲット CPU が必要となるよう指定します。

#### 明示的

ターゲット CPU の検索方法を指定します。

### インタフェース

デバッグプローブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

**JTAG** JTAG インタフェースを使用します。

**SWD** JTAG よりも少数のピンを使用する SWD インタフェースを使用します。serial-wire output (SWO) 通信チャンネルを使用する場合は、SWD を選択します。**［一般オプション］>［ライブラリ構成］**ページで **［SWO 経由の stdout/stderr］**を選択すると、SWD が自動的に選択される点に注意してください。SWO 設定の詳細については、231 ページの *［SWO トレースウィンドウ設定］ダイアログボックス*を参照してください。

### JTAG/SWD 速度

JTAG と SWD の通信速度を指定します。以下から選択します。

**自動検出** 信頼性の高い動作をするための最も高い周波数を自動的に使用します。

**クロックに同期 (A)** クロックをコアの外のプロセッサに同期します。RTCK JTAG 信号が使用可能な ARM デバイスでのみ機能します。

***n* MHz** JTAG と SWD の通信速度を、選択した周波数に設定します。

JTAG 通信に関する問題や、ターゲットメモリへの書込みに関する問題がある場合（プログラムのダウンロード中など）、速度をより低い周波数に設定すると、これらの問題が解決できる可能性があります。

### プローブ設定ファイル

**デフォルトのオーバライド**

製品パッケージに付属のデフォルトのプローブ設定ファイルの代わりに使用するプローブ設定ファイルを指定します。

**選択**

ターゲット CPU の検索方法を指定します。

### 明示的なプローブ設定

**複数ターゲットデバッグシステム**

複数の CPU からなるデバッグシステムを指定します。

**ターゲット No.（TAP またはマルチドロップ ID）**

デバッグシステムがマルチドロップ SWD の場合に、CPU のある DAP のマルチドロップ ID（16 進数表記）を指定します。

デバッグシステムが JTAG スキャンチェーンの場合、接続先のデバイスの［**ターゲット番号 TAP**］（テストアクセスポート）位置を指定します。TAP 番号はゼロから始まります。TAP 位置に複数の CPU がある場合、［**ターゲット上の CPU No.**］も指定する必要があります。

**ターゲット上の CPU No.**

デバッグシステムが複数コアの SWD の場合、DAP の CPU 番号を指定してください。

**JTAG スキャンチェーンに非 ARM デバイスを含む**

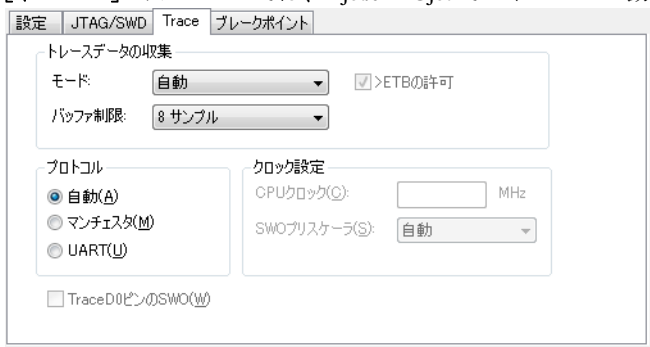
FPGA など、ARM デバイスと他のデバイスを混在させる JTAG スキャンチェーンを有効にします。

先行ビット

接続先のデバイスの TAP (Test Access Port) 位置を指定します。TAP 番号はゼロから始まります。

## I-jet/JTAGjet のトレースオプション

［トレース］オプションでは、I-jet/JTAGjet でのトレースの動作を指定します。




### モード

電力測定（プローブまたは I-scope による TrgPwr）は特定のトレースモードには依存せず、常に使用可能です（プローブでサポートされている場合）。

［デバッグログ］ウィンドウには、現在使用されているトレースモードについてのメッセージが含まれます。プローブやボード / デバイスの制限によって特定のモードが使用できない場合、トレースは無効になって［デバッグログ］ウィンドウにワーニングが表示されます。特定のトレースモードのサポートをチェックする方法は以下の通りです。

- プローブが特定のモードをサポートしている必要がある。
- プローブが特定のコアで特定のモードをサポートしている必要がある。たとえば、ARM9 上の ETM は I-jet Trace プローブではサポートされていません。
- 特定のコアが特定のモードをサポートしている必要がある。たとえば、Cortex-M0 は SWO/ETM/ETB をまったくサポートしておらず、ARM9 は SWO をサポートしていません。
- 使用されているアダプタが指定のモードをサポートしている必要がある。たとえば、ARM20 アダプタが I-jet Trace と併用されている場合には ETM トレースは使用できません。
- 特定のデバイスが特定のモードをサポートしている必要がある。たとえば、ETM トレースは ETM のない Cortex-M3 では使用できず、これはオンチップの TPIU 設定レジスタを読み込むまでは検出できません。

［モード］オプションは、トレースデータ収集のメカニズムとインタフェースを指定します。以下から選択します。

**自動**

プローブとボード / デバイスの機能に応じて、最適なメカニズムとインタフェースを自動的に選択します。

基本モードはプローブに依存する順序で実行されます。

- I-jet: 最初に SWO、次に ETB の順に実行（ETM はサポートされていません）。
- I-jet Trace: 最初に ETM、次に SWO、続いて ETB の順。
- JTAGjet-Trace: 最初に ETM、次に ETB の順に実行（SWO はサポートされていません）。
- JTAGjet: ETB のみ（SWO と ETM はサポートされていません）。

これらのモードのいずれも使用できない場合は、トレースは無効になります（［なし］を選択したときと同じです）。［自動］モードでは、トレースに関連したオンチップのリソースへの初期アクセスがより多く行われることがあります。このため、特定のプローブとモードを使用する場合は、C-SPY がトレースのリソースをより効率的に初期化 / 構成できるようにモードを明示的に設定した方が便利です。

**なし**

トレースを無効にします。このモードでは、C-SPY はトレース関連のオンチップのリソースには一切アクセスしません。このモードは以下の場合に使用できます。

- 接続に問題があります。トレースのリソース初期化からの干渉がない状態で接続の問題を診断する方が簡単です。
- トレースによって内部のクロックおよび / または GPIO mux 設定が変わることがあり、結果として一部のアプリケーションが特定のトレースモードで正しく機能しない可能性があります。
- 低消費電力モードで実行してください。内部のオンチップトレースロジックとトレースピンの切替えには、追加の電流が必要なことがあり、低消費電力測定に干渉する恐れがあります。極端な場合には、トレース /GPIO でクロックを有効にすると、CPU 内部の一部のクロックがアクティブな状態のままになるため、CPU が実際に低消費電力モードに入るのを妨げることがあります。

**シリアル (SWO)**

シリアル (SWO) インタフェースを通じてトレースデータを収集します。

**パラレル (ETM)**

パラレル (ETM) インタフェースを通じてトレースデータを収集します。

**オンチップ (ETB/MTB)**

オンチップ (ETB/MTB) インタフェースを通じてトレースデータを収集します。

### ETB の許可

オンチップ (ETB) の同時トレースを許可します。このオプションは、**［モード］**が**［シリアル (SWO)］**の場合にのみ使用できます。

### バッファの制限

収集するメガサンプル数を選択します。このオプションは、パラレル (ETM) モードの使用時（**［パラレル (ETM)］**または**［自動］**により暗示的に）にのみ利用できます。

ETM はトレースクロックの各端部で 4 ビットのトレースデータを提供します。このような最小のトレースデータを、*ETM サンプル*といいます。トレースプローブは、MIPI20 コネクタを介して ETM 対応のデバイスから 4 ビットのトレースを収集することができます。収集されたトレースデータはプローブ内に保管されます。

大量のトレースデータの読取りとデコードには時間がかかることがあるため、トレースデータの収集が停止したときに（CPU の停止またはバッファがフルになったときなど）、ETM メモリのどの部分を C-SPY で読み取るかを制限することが可能です。**［バッファの制限］**オプションを使用して、この制限を指定した数の Msamples に限定します。（1 Msample = 最大 100 万の 4 ビットサンプル）**［バッファの制限］**で高い値を指定するほど、より役に立つトレースデータが収集可能ですが、結果が表示されるまでの時間も長くなり、保存するためのメモリも多く必要となります。C-SPY はトレースプローブから最も新しいサンプルを取得し、収集されたトレースデータの残りはプローブにより破棄されます。

ETM のローサンプル数と［ETM トレース］ウィンドウで表示される PC サンプル数に単純な相関関係はありません。ETM プロトコル自体は高度に圧縮されており、プローブによって ETM アイドルサイクルがさらに圧縮されます。したがって、トレースプローブにより収集された特定の数の ETM ローサンプルからどれだけの命令をデコードできるかを推測することは不可能です。アプリケーションで PC が頻繁に変わる場合、ETM はより多くの PC ビットを送るためにさらに多くのサンプルを必要とします。このため、トレースデータは十分に圧縮されません。特定のアプリケーションプロファイルの場合、この値は通常は定数（各 4 ビットサンプルに対して 0.5 から 2 の命令）です。こ

のため、デコードされたデータと C-SPY のパフォーマンスのバランスを良好に保つバッファ制限を自分で判断する必要があります。

**注：**JTAGjet-Trace プローブでは、このオプションは使用できません。JTAGjet-Trace のバッファ制限は、ハードウェアの制限に応じて 1M/2M/4M サンプルに固定されています。

### SWO プロトコル

SWO チャンネルの通信プロトコルを指定します。以下から選択します。

| | |
|---|---|
| **自動** | 使用するデバイスに応じて、最適なプロトコルと速度を自動的に選択します。 |
| **Manchester** | Manchester プロトコルを指定します。 |
| **UART** | UART プロトコルを指定します。 |

### CPU クロック

内部プロセッサクロック HCLK の正確なクロック周波数を指定します (MHz)。10 進数で指定できます。この値は、SWO の通信速度を設定するときに使用します。

### SWO プリスケーラ

SWO 通信チャンネルのクロックプリスケーラ (KHz) を指定します。プリスケーラによって、SWO クロック周波数が決まります。

［**自動**］を選択すると、I-jet または I-jet Trace デバッグプローブで処理が可能な最高の周波数が自動的に使用されます。この設定は、データパケットが送信中に失われる場合に使用してください。

［**SWO クロック設定**］オプションをオーバライドするには、［**SWO 設定**］ダイアログボックスで［**プロジェクトのデフォルト設定をオーバライド**］オプションを使用します（235 ページの *プロジェクトデフォルトのオーバライド* を参照）。

### TraceD0 ピンの SWO

SWO トレースデータがトレースデータの D0 ピンに出力されるよう指定します。このオプションを使用するときは、SWD と JTAG インタフェースが SWO トレースデータを処理できます。

使用するデバイスとボードの両方がこのピンをサポートしている必要があります。

## J-Link/J-Trace の設定オプション

［設定］オプションでは、J-Link/J-Trace プローブを指定します。


### リセット

デバッガの起動時に使用するリセット方法を選択します。Cortex-M の場合、他のデバイスとは異なる方式を使用します。実際のリセット方法の種別番号は、使用可能な選択肢ごとに指定します。以下から選択します。

**ノーマル**
（0、デフォルト）
デフォルトの方式です。ターゲットデバイスをリセットする最良の方法です。大半のデバイスでは、これはリセット方式［**コアとペリフェラル**］(8) と同じです。一部のデバイスでは特殊な処理が必要なこともあります。たとえば、リセット後やアプリケーション起動後に実行する必要がある ROM ブートローダを持つデバイスなどです。

**コア** (1)
コアは VECTRESET ビットでリセットされます。ペリフェラルユニットは影響を受けません。

**コアとペリフェラル** (8)
コアとペリフェラルをリセットします。

**リセットピン** (2)
J-Link は RESET ピンを low に設定して、コアとペリフェラルユニットをリセットします。通常、ターゲットデバイスの CPU RESET ピンも low になり、その結果、CPU とペリフェラルユニット両方がリセットされます。

**リセット中に接続** (3) J-Link は、リセットをアクティブにしたままでターゲットに接続します（リセットは「低」になり、ターゲットに接続中はそのままになります）。これは、STM32 デバイスの推奨リセット方式です。この方式は、STM32 デバイスでのみ使用できます。

**ブートロード後に停止**（4 または 7） NXP Cortex-M0 デバイス。これは通常のセット方式と同じですが、ブートローダの実行完了後にターゲットが停止します これは、LPC11xx および LPC13xx デバイスの推奨リセット方式です。

Analog Devices Cortex-M3 デバイス (7)。AIRCR の SYSRESETREQ ビットを設定することにより、コアとペリフェラルユニットをリセットします。コアは ADI カーネルを実行できますが（これによりデバッグインタフェースが有効になります）、リセット後にユーザアプリケーションが実行されないよう徹底するために、カーネルが実行された後、最初の命令の前にコアが停止します。

**ブートロード前に停止** (5) これは通常のセット方式と同じですが、ブートローダの実行開始前にターゲットが停止します ブートローダのデバッグが必要な場合を除いて、この方式は通常は使用しません。この方式は、LPC11xx および LPC13xx デバイスでのみ使用できます。

**ノーマル、ウォッチドッグの無効化**（6、9 または 10） まずノーマルのリセットを実行してコアとペリフェラルユニットをリセットし、リセット直後に CPU を停止します。CPU の停止後はウォッチドッグは無効になります。これは、ウォッチドッグはデフォルトでリセット後に実行されるためです。ターゲットアプリケーションがウォッチドッグにフィードしない場合、永久にリセットされるため J-Link からデバイスへの接続が解除されます。この方式は、Freescale Kinetis デバイス (6) および NXP LPC 1200 デバイス (9)、Samsung S3FN60D デバイス (10) で利用できます。

これらの方式はすべて、JTAG および SWD インタフェースどちらにも使用できます。すべての方式は CPU をリセット後に停止します。

その他のコアの場合は、以下の方式から選択します。

**ハードウェア、停止までの遅延時間を指定 (ms)** (0)

ハードウェアリセットからプロセッサの停止までの遅延時間を指定します。これは、C-SPY がアクセスを開始したときに、チップが完全な動作状態であることを確認するために使用されます。デフォルトでは遅延はゼロに設定され、できるだけ速くプロセッサを停止します。

これはハードウェアリセットです。

**ハードウェア、ブレークポイントを使用して停止** (1)

リセット後、J-Link はブレークポイントを使用して CPU の停止を継続的に試行します。通常、CPU は、リセット後間もなく停止されます。ほとんどのシステムでは、いくつかの命令を実行してから CPU を停止できます。

これはハードウェアリセットです。

**ハードウェア、0 で停止** (4)

ブレークポイントをアドレスゼロに設定することでプロセッサを停止します。なお、この機能はすべての ARM マイクロコントローラでサポートされているわけではありません。

これはハードウェアリセットです。

**ハードウェア、DBGRQ を使用して停止** (5)

リセット後、J-Link は DBGRQ を使用して CPU の停止を継続的に試行します。通常、CPU は、リセット後間もなく停止されます。ほとんどのシステムでは、いくつかの命令を実行してから CPU を停止できます。

これはハードウェアリセットです。

**ソフトウェア** (-)

PC をプログラムのエントリアドレスに設定します。

これはソフトウェアリセットです。

**ソフトウェア、Analog デバイス** (2)

Analog Devices ADuC7xxx ファミリ専用のリセットシーケンスを使用します。この方式は、**[一般オプション] > [ターゲット]** ページで、**[デバイス]** ドロップダウンリストからこの種類のデバイスを選択した場合にのみ使用できます。

これはソフトウェアリセットです。

| | |
|---|---|
| **ハードウェア、NXP LPC** (9) | この方式は、**［一般オプション］**>**［ターゲット］**ページで、**［デバイス］**ドロップダウンリストからこの種類のデバイスを選択した場合にのみ使用できます。<br><br>これは NXP LPC デバイス専用のハードウェアリセットです。 |
| **ハードウェア、Atmel AT91SAM7** (8) | この方式は、**［一般オプション］**>**［ターゲット］**ページで、**［デバイス］**ドロップダウンリストからこの種類のデバイスを選択した場合にのみ使用できます。<br><br>これは **Atmel AT91SAM7** ファミリ専用のハードウェアリセットです。 |

さまざまなリセット方式に関する詳細については、arm¥doc ディレクトリの *『ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド』* を参照してください。

ターゲットのソフトウェアリセットを使用しても、ターゲットシステムの設定値を変更することはありません。プログラムカウンタとモードレジスタ CPSR をリセット状態にするだけです。一般的に、C-SPY リセットはソフトウェアリセットだけです。［ハードウェアリセット］オプションを使用する場合、C-SPY では、デバッガの起動時に最初のハードウェアリセットを生成します。これはダウンロードの前に一度実行されます。［フラッシュローダを使用する］オプションが選択されている場合は、フラッシュダウンロード後にもう一度行われます。62 ページの *フラッシュのコードのデバッグ*および 63 ページの *RAM のコードのデバッグ*を参照してください。

ハードウェアリセットは、アプリケーションの低レベル設定が完全でないと、問題が発生する可能性があります。低レベル設定でメモリ構成とクロックを設定しないと、ハードウェアリセット後のアプリケーションは動作しません。C-SPY でこれを処理するには、セットアップマクロの `execUserReset()` 関数が適しています。同様な例（`execUserPreload()` を使用）については、60 ページの *メモリの再配置*を参照してください。

### JTAG/SWD 速度

JTAG 通信速度 (kHz) を指定します。以下から選択します。

**自動**

信頼性の高い動作をするための最も高い周波数を自動的に使用します。初期速度には、最大可能周波数が見つかるまで固定周波数が使用されます。一般的に、デフォルトの初期周波数 (32kHz) を使用できますが、初期リセット後にできるだけ短時間で CPU の停止が必要な場合、初期周波数を上げてください。

速い初期速度が必要な場合に設定します。リセット後にフラッシュまたは RAM から CPU で望ましくない命令（電源停止の命令など）が実行された場合などです。このような場合、初期速度が速いとデバッガではリセット後に短時間で確実に CPU を停止できます。

初期値は 1 ～ 12,000kHz の範囲である必要があります。

**固定**

JTAG 通信速度 (kHz) を指定します。値は 1 ～ 12,000kHz の範囲である必要があります。

JTAG 通信に関する問題や、ターゲットメモリへの書込みに関する問題がある場合（プログラムのダウンロード中など）、速度をより低い周波数に設定すると、これらの問題が解決できる可能性があります。

**クロックに同期 (A)**

クロックをコアの外のプロセッサに同期します。`RTCK` JTAG 信号が使用可能な ARM デバイスでのみ機能します。クロックに同期した速度について詳しくは、`arm¥doc` ディレクトリの *『ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド』* を参照してください。

### クロック設定

CPU クロックを指定します。以下から選択します。

**CPU クロック**

内部プロセッサクロック `HCLK` の正確なクロック周波数を指定します (MHz)。10 進数で指定できます。この値は、SWO の通信速度の設定およびタイムスタンプの計算に使用します。

**SWO クロック**

SWO 通信チャンネルのクロック周波数を指定します (KHz)。

**自動**

J-Link デバッグプローブで使用できる最大可能周波数を自動的に使用します。［自動］が選択されていない場合、希望する SWO クロックの値をテキストボックスに入力できます。10 進数で指定できます。このオプションは、送信中にデータパケットが失われる場合に使用します。

［**クロック設定**］オプションをオーバライドするには、［**SWO 設定**］ダイアログボックスの［**プロジェクトのデフォルトのオーバライド**］オプションを使用します（233 ページの *［SWO 設定］ダイアログボックス*を参照）。

### ETM/ETB

**[Prefer ETB]（ETB を優先）**オプションは、ETM トレースの代わりにデフォルトの ETB トレースを選択します。

**注 :** このオプションは、J-Trace にのみ適用されます。

## J-Link/J-Trace 接続オプション

［**接続**］オプションでは、J-Link/J-Trace プローブとの接続を指定します。


### 通信

C-SPY と J-Link デバッグプローブ間の通信チャンネルを選択します。以下から選択します。

**USB**

USB 接続を選択します。ドロップダウンリストでシリアル番号が選択されている場合、指定したシリアル番号の J-Link デバッグプローブが選択されます。

**TCP/IP**

J-Link サーバの IP アドレスを指定します。TCP/IP 接続は、リモートコンピュータで動作する J-Link サーバに接続するために使用します。

**IP アドレス** : LAN に接続された J-Link プローブの IP アドレスを指定します。

**自動検出** : J-Link プローブを探してネットワークを自動的にスキャンします。このダウンロードを使用して、検出された J-Link プローブから選択します。

**シリアル番号** : 指定したシリアル番号を持つネットワーク上の J-Link プローブに接続します。

### インタフェース

J-Link デバッグプローブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

**JTAG**（デフォルト）

JTAG インタフェースを使用します。

**SWD**

JTAG よりも少数のピンを使用します。serial-wire output (SWO) 通信チャンネルを使用する場合は、SWD を選択します。**［一般オプション］>［ライブラリ構成］**ページで［SWO 経由の stdout/stderr］を選択すると、SWD が自動的に選択される点に注意してください。SWO 設定の詳細については、231 ページの *［SWO トレースウィンドウ設定］ダイアログボックス*を参照してください。

### JTAG スキャンチェーン

JTAG スキャンチェーンを指定します。以下から選択します。

**JTAG スキャンチェーン（マルチターゲット）**

JTAG スキャンチェーンに複数のデバイスがあることを指定します。

**TAP 番号**

接続先のデバイスの TAP (Test Access Port) 位置を指定します。TAP 番号はゼロから始まります。

**スキャンチェーンに非 ARM デバイスが含まれています**

FPGA など、ARM デバイスと他のデバイスを混在させる JTAG スキャンチェーンを有効にします。

**先行ビット**

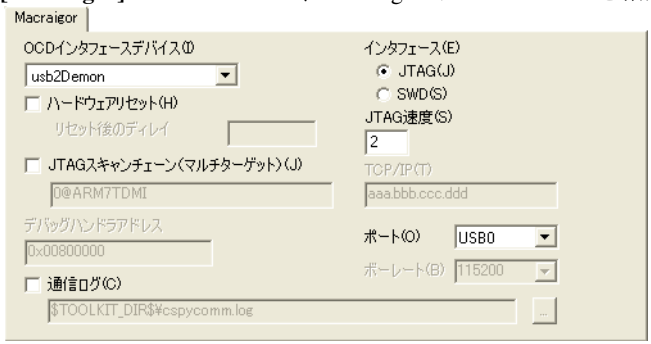
デバッグ対象の ARM デバイスの前の IR ビット数を指定します。

### 通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、JTAG インタフェースに対する十分な知識が必要です。

## Macraigor

**[Macraigor]** オプションでは、Macraigor インタフェースを指定します。




### OCD インタフェースデバイス

使用しているハードウェアインタフェースに対応するデバイスを選択します。サポートされている Macraigor JTAG プローブは、Macraigor **mpDemon** です。

### インタフェース

J-Link デバッグプローブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

**JTAG**（デフォルト）　JTAG インタフェースを使用します。

**SWD**　JTAG よりも少数のピンを使用します。serial-wire output (SWO) 通信チャンネルを使用する場合は、SWD を選択します。**［一般オプション］>［ライブラリ構成］**ページで［SWO 経由の stdout/stderr］を選択すると、SWD が自動的に選択される点に注意してください。SWO 設定の詳細については、231 ページの *［SWO トレースウィンドウ設定］ダイアログボックス*を参照してください。

### JTAG 速度

JTAG プローブと ARM JTAG ICE ポート間の速度を指定します。この値は 1 ～ 8 の範囲にあり、スキャンクロックの生成時に JTAG プローブクロックが分割された係数を設定する必要があります。

mpDemon インタフェースには、低速である大きな値（2 や 3 など）の設定が必要となる場合があります。

### TCP/IP

イーサネット /LAN ポートに接続された JTAG プローブの IP アドレスを指定します。

### ポート

通信リンクとして使用するホストコンピュータ上のシリアルポートまたはパラレルポートを選択します。JTAG プローブが接続されるホストのポートを選択します。

パラレルポートでは、コンピュータが 1 つのパラレルポートを搭載している場合、通常は LPT1 を使用してください。なお、ラップトップコンピュータにはその 1 つのパラレルポートを LPT2 または LPT3 にマッピングしている場合があります。できれば、EPP モードが高速であるため、パラレルポートをこのモードに設定してください。双方向で互換性のあるモードでは、動作はしますが低速となります。

### ボーレート

シリアル通信速度を選択します。

### ハードウェアリセット

デバッガの起動時に最初のハードウェアリセットを生成します。これはダウンロードの前に一度実行されます。**［フラッシュローダを使用する］**オプションが選択されている場合は、フラッシュダウンロード後にもう一度行われます。62 ページの *フラッシュのコードのデバッグ*および 63 ページの *RAM のコードのデバッグ*を参照してください。

ターゲットのソフトウェアリセットを使用しても、ターゲットシステムの設定値を変更することはありません。プログラムカウンタをリセット状態にするだけです。一般的に、C-SPY リセットはソフトウェアリセットだけです。

ハードウェアリセットは、アプリケーションの低レベル設定が完全でないと、問題が発生する可能性があります。低レベル設定でメモリ構成とクロックを設定しないと、ハードウェアリセット後のアプリケーションは動作しません。C-SPY でこれを処理するには、セットアップマクロの `execUserReset()` 関数が適しています。同様な例（`execUserPreload()` を使用）については、

60 ページの*メモリの再配置*を参照してください。

### JTAG スキャンチェーン（マルチターゲット）

JTAG スキャンチェーン上に複数のデバイスがある場合に、各デバイスを定義します。また、どのデバイスに接続するか指定する必要があります。構文は以下のとおりです。

```
<0>@dev0,dev1,dev2,dev3,...
```

ここで、0 は接続先デバイスの TAP 番号で、*dev0* は Macraigor JTAG プローブで最も近い TDO ピンです。

### デバッグハンドラアドレス

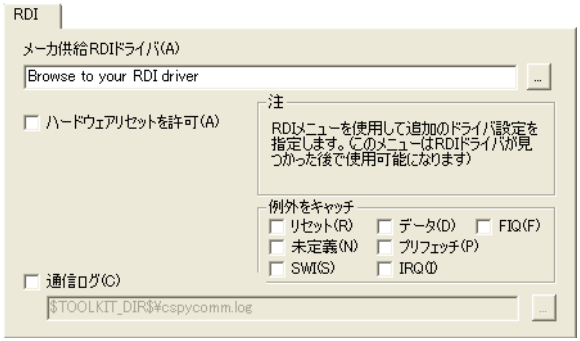
Intel XScale デバイスで使用されるデバッグハンドラの位置（メモリアドレス）を指定します。メモリ空間を保存するには、キャッシュ RAM の一部がマッピングできるアドレスを指定してください。その位置には物理メモリは含まれていません。できれば、下位 16MB メモリの未使用領域を見つけて、そこにハンドラのアドレスを置きます。

### 通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、JTAG インタフェースに対する十分な知識が必要です。

## RDI

**[RDI]** オプションを使用すると、ARM Ltd. RDI 1.5.1 仕様に準拠する JTAG インタフェースを使用できます。こうしたインタフェースの一例が、ARM RealView Multi-ICE JTAG インタフェースです。



### Manufacturer RDI driver

JTAG ポッドを提供する RDI ドライバ DLL ファイルのファイルパスを指定します。

### ハードウェアリセットを許可

エミュレータでターゲットのハードウェアリセットを実行可能にします。

ターゲットのソフトウェアリセットを使用しても、ターゲットシステムの設定値を変更することはありません。プログラムカウンタをリセット状態にするだけです。

アプリケーションの低レベル設定が完全である場合にのみ、ハードウェアリセットを行うことができます。低レベル設定でメモリ構成とクロックを設定しないと、ハードウェアリセット後のアプリケーションは動作しません。C-SPY でこれを処理するには、セットアップマクロの `execUserReset()` 関数が適しています。同様な例（`execUserPreload()` を使用）については、60 ページの*メモリの再配置*を参照してください。

**注 :** このオプションを使用するには、使用する RDI ドライバでハードウェアリセットがサポートされている必要があります。

### 例外をキャッチ

例外がブレークポイントとして処理されるようにします。実行中のプログラムが定義したように例外処理を行うのではなく、デバッガが停止します。

取得できる ARM コアの例外は以下のとおりです。

| 例外 | 説明 |
|---|---|
| リセット | リセット |
| 未定義 | 未定義の命令 |
| SWI | ソフトウェア割込み |
| データ | データの異常終了（データアクセスのメモリ障害） |
| プリフェッチ | プリフェッチの異常終了（命令フェッチのメモリ障害） |
| IRQ | 通常の割込み |
| FIQ | 高速割込み |

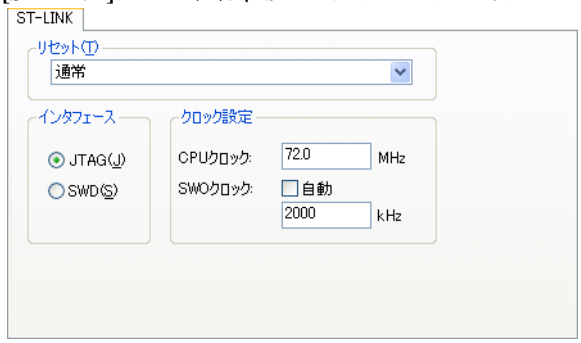
*表 51: 例外をキャッチ*

### RDI とのコミュニケーションのログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、RDI インタフェースに対する十分な知識が必要です。

## ST-LINK

**[ST-LINK]** ページには、ST-LINK プローブのオプションが含まれます。




### リセット

デバッガの起動時に使用するリセット方法を選択します。実際のリセット方法の種別番号は、使用可能な選択肢ごとに指定します。以下から選択します。

**ノーマル** (0)　標準のリセット処理を実行します。

**リセットピン** (1)　リセットピンを使用して、ハードウェアリセットを実行します。ST-LINK バージョン 2 でのみ使用できます。

**リセット中の接続** (2)　ST-LINK は、リセットピンをアクティブにしたままでターゲットに接続します（リセットピンは「低」になり、ターゲットに接続中はそのままになります）。ST-LINK バージョン 2 でのみ使用できます。

### インタフェース

ST-LINK デバッグプローブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

**JTAG**（デフォルト）　JTAG インタフェースを使用します。

**SWD**　JTAG よりも少数のピンを使用します。

### クロック設定

CPU クロックを指定します。以下から選択します。

| | |
|---|---|
| **CPU クロック** | 内部プロセッサクロック HCLK の正確なクロック周波数を指定します (MHz)。10 進数で指定できます。この値は、SWO の通信速度の設定およびタイムスタンプの計算に使用します。 |
| **SWO クロック** | SWO 通信チャンネルのクロック周波数を指定します (KHz)。 |
| **自動** | J-Link デバッグプローブで使用できる最大可能周波数を自動的に使用します。［自動］が選択されていない場合、希望する SWO クロックの値をテキストボックスに入力できます。10 進数で指定できます。このオプションは、送信中にデータパケットが失われる場合に使用します。 |

［**クロック設定**］オプションをオーバライドするには、［**SWO 設定**］ダイアログボックスの［**プロジェクトのデフォルトのオーバライド**］オプションを使用します（235 ページの*プロジェクトデフォルトのオーバライド*を参照）。

## TI Stellaris の設定オプション

［**設定**］オプションでは、TI Stellaris インタフェースを指定します。

### インタフェース

J-Link デバッグプローブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

**JTAG**（デフォルト）　JTAG インタフェースを使用します。

**SWD**　JTAG よりも少数のピンを使用します。serial-wire output (SWO) 通信チャンネルを使用する場合は、SWD を選択します。**［一般オプション］>［ライブラリ構成］**ページで［SWO 経由の stdout/stderr］を選択すると、SWD が自動的に選択される点に注意してください。SWO 設定の詳細については、231 ページの *［SWO トレースウィンドウ設定］ダイアログボックス*を参照してください。

### JTAG/SWD 速度

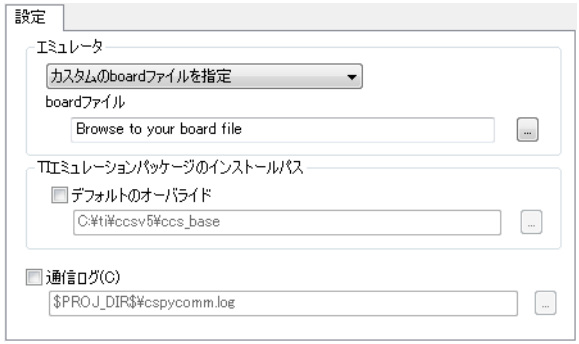
JTAG 通信速度 (kHz) を指定します。

### 通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。結果を解析するには、通信プロトコルに関する詳しい知識が必要です。

## TI XDS の設定オプション

**［設定］**オプションでは、TI XDS インタフェースを制御します。




### エミュレータ

使用するエミュレータを特定します。デフォルトの board ファイルをオーバライドするには、**［カスタムの board ファイルを指定］**を選択します。

### TI エミュレーションパッケージのインストールパス

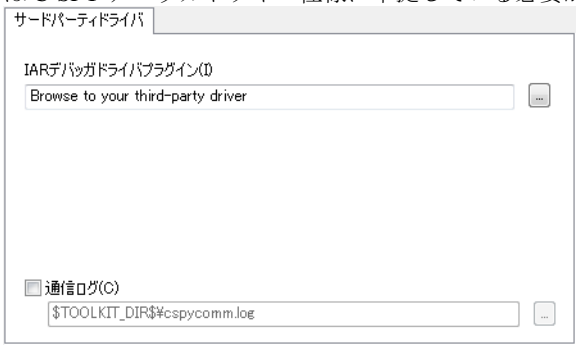
Texas Instruments エミュレーションパッケージのデフォルトのインストールパスをオーバライドするには、［**デフォルトのオーバライド**］を選択します。

### 通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、インタフェースに対する十分な知識が必要です。

## サードパーティ製ドライバのオプション

［**サードパーティ製ドライバ**］オプションは、サードパーティベンダが提供するドライバプラグインをロードする場合に使用されます。これらのドライバは C-SPY デバッガドライバ仕様に準拠している必要があります。




ここで設定可能なオプションのほかに、サードパーティのドライバ用オプションを［**プロジェクト**］>［**オプション**］>［**デバッガ**］>［**追加オプション**］ページを使用して設定できます。

### IAR デバッガのドライバプラグイン

サードパーティ製ドライバプラグインの DLL ファイルのファイルパスを指定します。参照ボタンを使用して選択することもできます。

### 通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、インタフェースに対する十分な知識が必要です。

# C-SPY ドライバについての追加情報

この章では、C-SPY® ドライバの追加メニューや機能について説明します。また、問題の解決に役立つヒントも記載されています。

## C-SPY ドライバメニューのリファレンス情報

このセクションでは、C-SPY ドライバに固有のメニューに関するリファレンス情報を提供します。具体的には以下の項目を解説します。



### *C-SPY ドライバ*

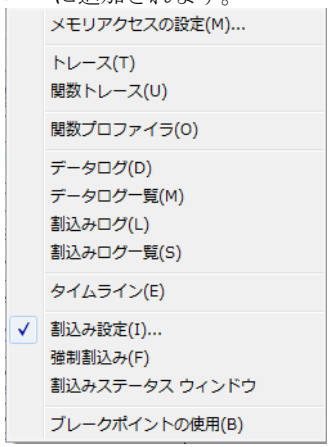
C-SPY デバッガを起動する前に、オプション［**デバッガ**］>［**設定**］>［**ドライバ**］を使用して、［**オプション**］ダイアログボックスで C-SPY ドライバを指定する必要があります。

デバッグセッションを開始すると、その C-SPY ドライバに固有のメニューが、ドライバに固有のコマンドとともにメニューバーに表示されます。

本ガイドで *［C-SPY ドライバ］>* に続いてメニューコマンドを選択する場合、*［C-SPY ドライバ］* とはメニューを指します。ドライバで機能がサポートされている場合、そのコマンドがメニューに表示されます。

# ［シミュレータ］メニュー

シミュレータドライバを使用する場合、**［シミュレータ］**メニューがメニューバーに追加されます。




## メニューのコマンド

メニューから実行できるコマンドは、以下のとおりです。

**メモリアクセスの設定**

さまざまなアクセスタイプでメモリエリアを指定して、メモリアクセスチェックをシミュレーションするダイアログボックスを開きます（210 ページの*［メモリアクセス設定］ダイアログボックス*を参照）。

**メモリ構成**

ダイアログボックスが表示され、そこで使用するデバイスに合うように C-SPY を構成できます（200 ページの*［メモリ構成］ダイアログボックス*、*C-SPY シミュレータ*を参照）。

**トレース**

収集されたトレースデータを表示するウィンドウが開きます（237 ページの*［トレース］ウィンドウ*を参照）。

**関数トレース**

関数の呼出しや復帰を示すトレースデータを表示するウィンドウが開きます（243 ページの*［関数トレース］ウィンドウ*を参照）。

**関数プロファイラ**

関数のタイミング情報を示すウィンドウを表示します（287 ページの*［関数プロファイラ］ウィンドウ*を参照）。

**データログ**

最大 4 つの異なるメモリ位置または領域へのアクセスを記録するウィンドウが開きます（125 ページの*［データログ］ウィンドウ*を参照）。

**データログ一覧**

特定のメモリ位置またはメモリエリアへのデータアクセスの概要を表示するウィンドウが開きます（127 ページの*［データログ一覧］ウィンドウ*を参照）。

**割込みログ**

定義されているすべての割込みのステータスを表示するウィンドウを開きます（385 ページの*［割込みログ］ウィンドウ*を参照）。

**割込みログ概要**

定義されているすべての割込みのステータス概要を表示するウィンドウを開きます（389 ページの*［割込みログ概要］ウィンドウ*を参照）。

**タイムライン**

タイムライン上に多様な情報をグラフィック表示するウィンドウが開きます（244 ページの*［タイムライン］ウィンドウ*を参照）。

**シミュレート周波数**

**［シミュレート周波数］**ダイアログボックスが開き、ログのウィンドウなどにシミュレータが時間の情報を表示する際に使用されるシミュレータの周波数を指定することができます。これによって、シミュレータの速度は影響を受けません。

**割込み設定**

C-SPY の割込みシミュレーションを設定するためのダイアログボックスを開きます（378 ページの*［割込み設定］ダイアログボックス*を参照）。

**強制割込み**

割込みをすぐにトリガできるウィンドウを開きます（382 ページの*［強制割込み］ウィンドウ*を参照）。

**割込みステータス**

割込みをすぐにトリガできるウィンドウを開きます（383 ページの*［割込みステータス］ウィンドウ*を参照）。

**ブレークポイントの使用**

すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの*［ブレークポイントの使用］ウィンドウ*を参照）。

# CMSIS-DAP メニュー

C-SPY CMSIS-DAP ドライバを使用する場合、［**CMSIS-DAP**］メニューがメニューバーに追加されます。


## メニューのコマンド

メニューから実行できるコマンドは、以下のとおりです。

### メモリ構成

*ダイアログボックスが開きます（204 ページの［メモリ構成］ダイアログボックス、C-SPY ハードウェアデバッガドライバを参照）。*

### メモリキャッシュをバイパス

C-SPY でのメモリのキャッシングとメモリ範囲のチェックを無効にします。

C-SPY は通常、［**メモリ構成**］ダイアログボックスのメモリ範囲情報を使用して、ターゲットメモリの特定範囲へのアクセスを制限したり、C-SPY のパフォーマンスを高めるためにターゲットメモリの内容をキャッシュ化したりします。ごく稀な状況では、これは適切ではないため、［**メモリキャッシュのバイパス**］を選択して、キャッシングとメモリ範囲チェックを完全にオフにすることができます。C-SPY からのすべてのアクセスは、対応するターゲットシステムへのアクセスということになります。次のような場合が該当します。

- メモリが実行時にマッピングし直される場合に、固定の範囲として指定できない。
- メモリ範囲の設定が間違っているか、不完全である。

**ステップ実行中に割込みを無効化**

ステップ実行済みの命令のみが実行されるように徹底します。割込みは実行されません。このコマンドは、フルスピードで実行中でない場合に使用可能です。また、一部の割込みはデバッグプロセスを妨害します。

**ETM トレース設定**

*ダイアログボックスが開きます*（226 ページの*［ETM トレース設定］ダイアログボックス*を参照）。

**ETM トレースの保存**

*ダイアログボックスが開きます*（277 ページの*［トレースの保存］ダイアログボックス*を参照）。

**ETM トレース**

［ETM トレース］ウィンドウが開きます（237 ページの*［トレース］ウィンドウ*を参照）。

**関数トレース**

ウィンドウを開きます（243 ページの*［関数トレース］ウィンドウ*を参照）。

**ベクタキャッチ**

割込みベクタテーブルのベクタに直接ブレークポイントを設定するダイアログボックスが表示されます（167 ページの*［ベクタキャッチ］ダイアログボックス*を参照）。なお、このコマンドはすべての ARM コアに使用できるわけではありません。

**タイムライン**

ウィンドウを開きます（244 ページの*［タイムライン］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**関数プロファイラ**

関数のタイミング情報を示すウィンドウを表示します（287 ページの*［関数プロファイラ］ウィンドウ*を参照）。

**セッションの概要**

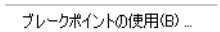
プロジェクト設定やセッションの設定、セッションの状態など、デバッグセッションに関する情報を一覧するウィンドウが表示されます。ウィンドウの内容をファイルに保存するには、コンテキストメニューから**［名前を付けて保存］**を選択します。

**ブレークポイントの使用**

すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの*［ブレークポイントの使用］ウィンドウ*を参照）。

## ［GDB サーバ］メニュー

C-SPY GDB サーバドライバを使用する場合、**［GDB サーバ］**メニューがメニューバーに追加されます。




### メニューのコマンド

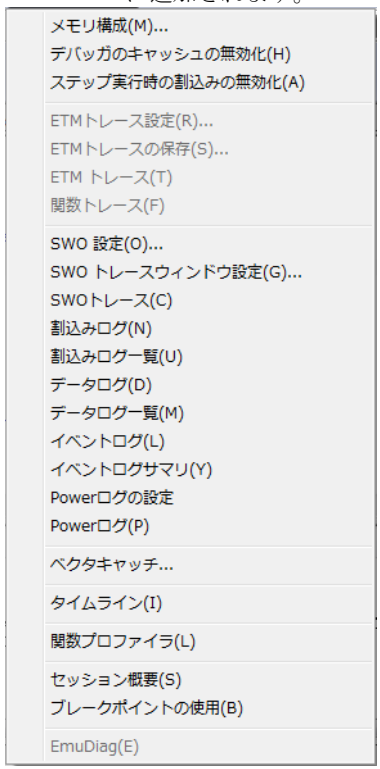
メニューから実行できるコマンドは、以下のとおりです。

**ブレークポイントの使用**

すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの*［ブレークポイントの使用］ウィンドウ*を参照）。

## I-jet/JTAGjet メニュー

C-SPY I-jet/JTAGjet ドライバを使用する際は、**[I-jet/JTAGjet]** メニューがメニューバーに追加されます。




### メニューのコマンド

メニューから実行できるコマンドは、以下のとおりです。

**メモリ構成**

ダイアログボックスが開きます（204 ページの *［メモリ構成］ダイアログボックス、C-SPY ハードウェアデバッガドライバ*を参照）。

**デバッガのキャッシュの無効化**

C-SPY でのメモリのキャッシングとメモリ範囲のチェックを無効にします。

C-SPY は通常、［**メモリ設定**］ダイアログボックスのメモリ範囲情報を使用して、ターゲットメモリの特定範囲へのアクセスを制限したり、C-SPY のパフォーマンスを高めるためにターゲットメモリの内容をキャッシュ化したりします。ごく稀な状況では、これは適切ではないため、［**デバッガキャッシュの無効化**］を選択して、キャッシングとメモリ範囲チェックを完全にオフにすることができます。C-SPY からのすべてのアクセスは、対応するターゲットシステムへのアクセスということになります。次のような場合が該当します。

- メモリが実行時にマッピングし直される場合に、固定の範囲として指定できない。
- メモリ範囲の設定が間違っているか、不完全である。

**ステップ実行中に割込みを無効化**

ステップ実行済みの命令のみが実行されるように徹底します。割込みは実行されません。このコマンドは、フルスピードで実行中でない場合に使用可能です。また、一部の割込みはデバッグプロセスを妨害します。

**ETM トレース設定**

*ダイアログボックスが開きます*（226 ページの*［ETM トレース設定］ダイアログボックス*を参照）。

**ETM トレースの保存**

*ダイアログボックスが開きます*（277 ページの*［トレースの保存］ダイアログボックス*を参照）。

**ETM トレース**

［ETM トレース］ウィンドウを開きます（237 ページの*［トレース］ウィンドウ*を参照）。

**関数トレース**

ウィンドウを開きます（243 ページの*［関数トレース］ウィンドウ*を参照）。

**SWO 設定**

*ダイアログボックスが開きます*（233 ページの*［SWO 設定］ダイアログボックス*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**SWO トレースウィンドウ設定**

*ダイアログボックスが開きます*（231 ページの*［SWO トレースウィンドウ設定］ダイアログボックス*を参照）。

**SWO トレース**

［SWO トレース］ウィンドウを開いて、収集したトレースデータを表示します（237 ページの*［トレース］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**割込みログ**

ウィンドウを開きます（385 ページの*［割込みログ］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**割込みログ一覧**

ウィンドウを開きます（389 ページの*［割込みログ概要］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**データログ**

ウィンドウを開きます（125 ページの*［データログ］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**データログ一覧**

ウィンドウを開きます（127 ページの*［データログ一覧］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**イベントログ**

ウィンドウを開きます（129 ページの*［イベントログ］ウィンドウ*を参照）。

**イベントログサマリ**

ウィンドウを開きます（131 ページの*［イベントログサマリ］ウィンドウ*を参照）。

**Power ログの設定**

ウィンドウを開きます（309 ページの*［Power ログ設定］ウィンドウ*を参照）。

**Power ログ**

ウィンドウを開きます（311 ページの*［Power ログ］ウィンドウ*を参照）。

**ベクタキャッチ**

割込みベクタテーブルのベクタに直接ブレークポイントを設定するダイアログボックスが表示されます（167 ページの*［ベクタキャッチ］ダイアログボックス*を参照）。なお、このコマンドはすべての ARM コアに使用できるわけではありません。

**タイムライン**

ウィンドウを開きます（244 ページの*［タイムライン］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**関数プロファイラ**

関数のタイミング情報を示すウィンドウを表示します（287 ページの*［関数プロファイラ］ウィンドウ*を参照）。

**セッションの概要**

プロジェクト設定やセッションの設定、セッションの状態など、デバッグセッションに関する情報を一覧するウィンドウが表示されます。ウィンドウの内容をファイルに保存するには、コンテキストメニューから**［名前を付けて保存］**を選択します。

**ブレークポイントの使用**

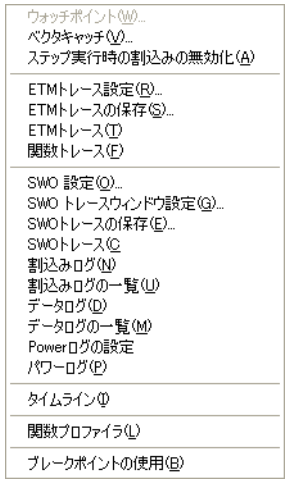
すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの*［ブレークポイントの使用］ウィンドウ*を参照）。

**EmuDiag**

**EmuDiag** アプリケーションを起動して、ホストコンピュータ、プローブ、ボード間の接続を診断できます。

## J-Link メニュー

C-SPYJ-Link ドライバを使用する場合、**[J-Link]** メニューがメニューバーに追加されます。




### メニューのコマンド

メニューから実行できるコマンドは、以下のとおりです。

#### ウォッチポイント

ウォッチポイントを設定するダイアログボックスが表示されます（150 ページの *[コード] ブレークポイントダイアログボックス*を参照）。

#### ベクタキャッチ

割込みベクタテーブルのベクタに直接ブレークポイントを設定するダイアログボックスが表示されます（167 ページの *[ベクタキャッチ] ダイアログボックス*を参照）。なお、このコマンドはすべての ARM コアに使用できるわけではありません。

#### ステップ実行中に割込みを無効化

ステップ実行済みの命令のみが実行されるように徹底します。割込みは実行されません。このコマンドは、フルスピードで実行中でない場合に使用可能です。また、一部の割込みはデバッグプロセスを妨害します。

**ETM トレース設定**

ETM トレースデータの生成と収集を設定するダイアログボックスを表示します（229 ページの *［ETM トレース設定］ダイアログボックス (J-Link/J-Trace)* を参照）。

このメニューコマンドは、ETB を持つ J-Link か ETM の使用時のみ利用できます。

**ETM トレースの保存**

収集されたトレースデータをファイルに保存するダイアログボックスを表示します（277 ページの *［トレースの保存］ダイアログボックス* を参照）。

このメニューコマンドは、ETB を持つ J-Link か ETM の使用時のみ利用できます。

**ETM トレース**

［ETM トレース］ウィンドウを開いて、収集したトレースデータを表示します（237 ページの *［トレース］ウィンドウ* を参照）。

このメニューコマンドは、ETB を持つ J-Link か ETM の使用時のみ利用できます。

**関数トレース**

関数の呼出しや復帰を示すトレースデータを表示するウィンドウが開きます（243 ページの *［関数トレース］ウィンドウ* を参照）。

このメニューコマンドは、ETB を持つ J-Link か ETM の使用時のみ利用できます。

**SWO 設定**

ダイアログボックスが開きます（233 ページの *［SWO 設定］ダイアログボックス* を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**SWO トレースウィンドウの設定**

ダイアログボックスが開きます（231 ページの *［SWO トレースウィンドウ設定］ダイアログボックス* を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**SWO トレースの保存**

収集されたトレースデータをファイルに保存するダイアログボックスを表示します（277 ページの *［トレースの保存］ダイアログボックス* を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**SWO トレース**

［SWO トレース］ウィンドウを開いて、収集したトレースデータを表示します（237 ページの*［トレース］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**割込みログ**

ウィンドウを開きます（385 ページの*［割込みログ］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**割込みログ一覧**

ウィンドウを開きます（389 ページの*［割込みログ概要］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**データログ**

ウィンドウを開きます（125 ページの*［データログ］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**データログ一覧**

ウィンドウを開きます（127 ページの*［データログ一覧］ウィンドウ*を参照）。

このメニューコマンドは、SWD/SWO インタフェース使用時にだけ使用できます。

**Power ログの設定**

ウィンドウを開きます（309 ページの*［Power ログ設定］ウィンドウ*を参照）。

**Power ログ**

ウィンドウを開きます（311 ページの*［Power ログ］ウィンドウ*を参照）。

**タイムライン**

ウィンドウを開きます（244 ページの*［タイムライン］ウィンドウ*を参照）。

このメニューコマンドは、ETM または SWD/SWO の使用時のみ利用可能です。

**関数プロファイラ**

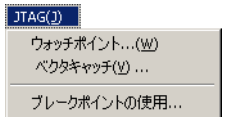
関数のタイミング情報を示すウィンドウを表示します（287 ページの*［関数プロファイラ］ウィンドウ*を参照）。

**ブレークポイントの使用**

すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの*［ブレークポイントの使用］ウィンドウ*を参照）。

## Macraigor の [JTAG] メニュー

C-SPY Macraigor ドライバを使用する際は、**[JTAG]** メニューがメニューバーに追加されます。




メニューから実行できるコマンドは、以下のとおりです。

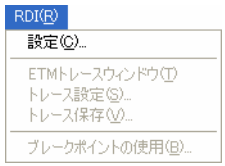
**ウォッチポイント** ウォッチポイントを設定するダイアログボックスを開きます（150 ページの*［コード］ブレークポイントダイアログボックス*を参照）。

**ベクタキャッチ** 割込みベクタテーブルのベクタに直接ブレークポイントを設定するダイアログボックスを開きます（167 ページの*［ベクタキャッチ］ダイアログボックス*を参照）。なお、このコマンドはすべての ARM コアに使用できるわけではありません。

**ブレークポイントの使用** すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの*［ブレークポイントの使用］ウィンドウ*を参照）。

## RDI メニュー

C-SPY RDI ドライバを使用する際は、**[RDI]** メニューがメニューバーに追加されます。




メニューから実行できるコマンドは、以下のとおりです。

**設定** RDI ドライバベンダから提供されたダイアログボックスを開きます。このダイアログボックスの詳細については、ドライバの資料を参照してください。

**トレース設定** ETM トレースを設定するダイアログボックスを表示します（229 ページの *［ETM トレース設定］ダイアログボックス (J-Link/J-Trace)* を表示）。

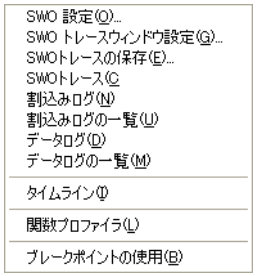
**トレースの保存** 収集されたトレースデータをファイルに保存するダイアログボックスを表示します（277 ページの *［トレースの保存］ダイアログボックス*を参照）。

**ブレークポイントの使用** すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの *［ブレークポイントの使用］ウィンドウ*を参照）。

**注 :** 設定ダイアログボックスのデフォルト設定値を取得するには、プロジェクトに特別な設定が必要ない場合であっても、ダイアログボックスを開く / 閉じるためだけにいくつかの RDI ドライバが必要です。

## ST-LINK メニュー

C-SPY ST-LINK ドライバを使用する際は、**[ST-LINK]** メニューがメニューバーに追加されます。




メニューから実行できるコマンドは、以下のとおりです。

**SWO 設定** [1] ダイアログボックスが開きます（233 ページの *[SWO 設定] ダイアログボックス*を参照）。

**SWO トレースウィンドウの設定** [1] ダイアログボックスが開きます（231 ページの *[SWO トレースウィンドウ設定] ダイアログボックス*を参照）。

**SWO トレースの保存** [1] 収集されたトレースデータをファイルに保存するダイアログボックスを表示します（277 ページの *[トレースの保存] ダイアログボックス*を参照）。

**SWO トレース** [1] [SWO トレース] ウィンドウを開いて、収集したトレースデータを表示します（237 ページの *[トレース] ウィンドウ*を参照）。

**割込みログ** [1] ウィンドウを開きます（385 ページの *[割込みログ] ウィンドウ*を参照）。

**割込みログ一覧** [1] ウィンドウを開きます（389 ページの *[割込みログ概要] ウィンドウ*を参照）。

**データログ** [1] ウィンドウを開きます（125 ページの *[データログ] ウィンドウ*を参照）。

**データログ一覧** [1] ウィンドウを開きます（127 ページの *[データログ一覧] ウィンドウ*を参照）。

**タイムライン** [2] ウィンドウを開きます（244 ページの *[タイムライン] ウィンドウ*を参照）。

**関数プロファイラ** 関数のタイミング情報を示すウィンドウを表示します（287 ページの *［関数プロファイラ］ウィンドウ*を参照）。

**ブレークポイントの使用** すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの *［ブレークポイントの使用］ウィンドウ*を参照）。

1 SWD/SWO インタフェースの使用時のみ。

2 ETM または SWD/SWO の使用時のみ。

### TI Stellaris メニュー

C-SPY TI Stellaris ドライバを使用する際は、**[TI Stellaris]** メニューがメニューバーに追加されます。




このコマンドはメニューから実行できます。

**ブレークポイントの使用** すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの *［ブレークポイントの使用］ウィンドウ*を参照）。

### TI XDS メニュー

C-SPY TI XDS ドライバを使用する際は、**[TI XDS]** メニューがメニューバーに追加されます。




このコマンドはメニューから実行できます。

**ブレークポイントの使用** すべてのアクティブなブレークポイントを一覧表示するウィンドウを開きます（149 ページの *［ブレークポイントの使用］ウィンドウ*を参照）。

## 問題の解決

以下のトピックについて説明します：

- ターゲットハードウェアとの接続ができない
- ステップ速度が遅い場合

C-SPY ハードウェアデバッガシステムを使用したデバッグには、それぞれに独立した数多くのシステム間のやりとりが必要です。このため、このデバッグシステムを設定するのは複雑な作業です。何か問題が起きると、問題の原因を特定することが難しいときがあります。

現在のデバッグセッションの詳細を見るには、ドライバメニューから［**セッション概要**］を選択します。使用する C-SPY ドライバによっては、このウィンドウがサポートされていない場合がある点に注意してください。

このセクションには、C-SPY ハードウェアデバッガシステムを使用したデバッグの際に起こりがちな、最も一般的な問題を解決するための推奨事項が記載されています。

評価ボードの動作に関する問題については、ボードに付属のマニュアルを参照するか、ハードウェア販売代理店にお問い合わせください。

## ターゲットハードウェアとの接続ができない

C-SPY がターゲットハードウェアと接続を確立できない理由は、いくつか考えられます。以下のようにしてください。

- ホストコンピュータ上の通信デバイスを確認します。
- ケーブルが破損していたり間違ったタイプではなく、正しく接続されているか確認します。
- 評価ボードに十分な電力が供給されているか確認します。
- IAR Embedded Workbench IDE で正しい通信オプションが指定されているか確認します。
- 正しいリセット方式が使用されているかチェックします。

リンカ設定ファイルを調べて、アプリケーションが間違ったアドレスにリンクされていないか確認してください。

## ステップ速度が遅い場合

ステップ速度が遅いと感じる場合、次のトラブルシューティングのヒントがステップの加速化に役立つことがあります。

- ハードウェアデバッガシステムを使用している場合、使用されているハードウェアブレークポイントの数を把握して、そのうちいくつかがステップ用に残っているようにします。

  C-SPY でのステップ実行は通常、ブレークポイントを使用して行われます。C-SPY がステップコマンドを実行する際、次の文にブレークポイントが設定されて、このブレークポイントに達するまでアプリケーションが実行されます。ハードウェアデバッガシステムを使用する場合、ハードウェアブレークポイントの数は限られています。通常これらは、フラッシュ

/ROM メモリに配置されたコード内にステップブレークポイントを設定するために使用されます。たとえば、C `switch` の文にステップインする場合、ブレークポイントは各分岐に設定されます。これによって、いくつかのハードウェアブレークポイントが使用されます。使用可能なハードウェアブレークポイントの数に達した場合、C-SPY はアセンブリレベルでのシングルステップに切り替わり、速度が非常に低下する可能性があります。

詳細については、137 ページの *C-SPY ハードウェアデバッガドライバのブレークポイント*、137 ページの *ブレークポイントの設定元*を参照してください。

- ［トレース］ウィンドウおよび［関数プロファイリング］ウィンドウの**［有効化 / 無効化］**ボタンを使用して、トレースデータの収集を無効にします。トレースデータの収集によって、ステップ実行が遅くなることがあります。これは収集されたデータが各ステップの後に処理されるためです。トレースデータの収集を無効にするには、対応するウィンドウを閉じるだけでは不十分な点に注意してください。
- SFR レジスタの限定された選択内容のみを表示するときに選択します。以下の 2 つのどちらかを選択します。［ウォッチ］ウィンドウに `#SFR_name`（`SFR_name` はモニタする SFR 名を示します）を入力するか、［レジスタ］ウィンドウで限られたグループの SFR を表示する独自のフィルタを作成します。多くの SFR レジスタを表示すると、ステップの実行が遅くなることがあります。これは各ステップの後にすべてのレジスタをハードウェアから読み取らなければならないためです。178 ページの *アプリケーション固有のレジスタグループの定義*を参照してください。
- ［メモリ］ウィンドウと［シンボルメモリ］ウィンドウが開いていれば閉じます。これは、各ステップの後に表示されているメモリを読み取らなければならず、そうなるとステップ実行が遅くなる可能性があるためです。
- ［ウォッチ］や［ライブウォッチ］、［ローカル］、［静的］など、式が表示されるウィンドウが開いていれば閉じます。これは、これらのウィンドウで各ステップの実行後にメモリが読み込まれて、ステップの実行が遅くなる可能性があるためです。
- ［スタック］ウィンドウが開いていれば閉じます。**［ツール］>［オプション］>［スタック］**を選択して、**［グラフィカルスタック表示とスタック使用トラッキングを有効にする］**オプションが有効になっていれば無効にします。
- 可能であれば、C-SPY とターゲットボード / エミュレータ間の通信速度を上げます。

# A



# B



# C

# D

# E



# F



# G



# H



# I

# J

# L



# M



# N



# O



# P

# R



# S

# T



# U



# V



# W



# X

# あ



# い



# う



# お



# か

# き



# く



# こ



# さ

# し



# す



# せ



# そ

# た



# ち



# つ



# て

# と

# の



# は



# ひ



# ふ

# へ



# ほ



# ま



# め

# も



# ゆ



# ら



# り



# る



# れ

# ろ



# 記号

# 数字