

# 導入ガイド

IAR Embedded Workbench®



GSEW-3-JP

The logo for IAR Systems, featuring a stylized globe icon to the left of the text "IAR" and "SYSTEMS" stacked vertically.

## 著作権事項

Copyright © 2009–2012 IAR Systems AB.

IAR Systems AB が事前に書面で同意した場合を除き、このドキュメントを複製することはできません。このドキュメントに記載するソフトウェアは、正当な権限の範囲内でインストール、使用、およびコピーすることができます。

## 免責事項

このドキュメントの内容は、予告なく変更されることがあります。また、IAR Systems 社では、このドキュメントの内容に関して一切責任を負いません。記載内容には万全を期していますが、万一、誤りや不備がある場合でも IAR Systems 社はその責任を負いません。

IAR Systems 社、その従業員、その下請企業、またはこのドキュメントの作成者は、特殊な状況で、直接的、間接的、または結果的に発生した損害、損失、費用、課金、権利、請求、逸失利益、料金、またはその他の経費に対して一切責任を負いません。

## 商標

IAR Systems、IAR Embedded Workbench、C-SPY、visualSTATE、The Code to Success、IAR KickStart Kit、I-jet\$ IAR および IAR システムズのロゴタイプは、IAR Systems AB が所有権を有する商標または登録商標です。

Microsoft および Windows は、Microsoft Corporation の登録商標です。

その他のすべての製品名は、その所有者の商標または登録商標です。

## 改版情報

第3版：2012年4月

部品番号：GSEW-3-JP

内部参照：Too6.4, Mys12, ISUD.

# 目次

はじめに .....	5
<b>本ガイドについて</b> .....	5
<b>表記規則</b> .....	5
<b>概要</b> .....	7
<b>製品ポートフォリオの概要</b> .....	7
<b>デバイスサポート</b> .....	9
<b>チュートリアル</b> .....	9
<b>ユーザドキュメンテーション</b> .....	10
<b>その他のリソース</b> .....	10
<b>IAR Embedded Workbench の概要</b> .....	11
<b>IDE</b> .....	11
<b>IAR C/C++ コンパイラ</b> .....	14
<b>IAR アセンブラ</b> .....	15
<b>IAR リンカと関連ツール</b> .....	16
<b>IAR C-SPY デバッガ</b> .....	18
<b>組込みアプリケーションの開発</b> .....	21
<b>開発サイクル</b> .....	21
<b>一般的に使用されるソフトウェアモデル</b> .....	22
<b>ビルド処理</b> .....	24
<b>パフォーマンス重視のプログラミング</b> .....	27
<b>ハードウェアおよびソフトウェア要因の考慮</b> .....	29
<b>アプリケーションの実行</b> .....	33
<b>アプリケーションプロジェクトの作成</b> .....	39
<b>ワークスペースの作成</b> .....	39
<b>新しいプロジェクトの作成</b> .....	40
<b>プロジェクトオプションの設定</b> .....	41
<b>プロジェクトへのソースファイルの追加</b> .....	42

ツール固有のオプションの設定 .....	43
コンパイル .....	45
リンク .....	46
デバッグ .....	49
デバッグの設定 .....	49
デバッグの起動 .....	51
アプリケーションの実行 .....	53
変数の検証 .....	54
メモリとレジスタのモニタ .....	56
ブレークポイントの使用 .....	57
ターミナル I/O の表示 .....	60
アプリケーションのランタイムでの動作の解析 .....	61

# はじめに

IAR Embedded Workbench の使用開始の手順へようこそ。

## 本ガイドについて

本ガイドの目的は、IAR Embedded Workbench の概要と IDE の使用方法、組み込みシステムソフトウェアの開発ツールの使用方法について説明することです。本ガイドでは一部の機能に重点を置いて、ツールの目的と機能について解説します。

C/C++ プログラミング言語や組み込みシステムのアプリケーション開発、使用するマイクロコントローラのアーキテクチャと命令セット（チップメーカーのマニュアルを参照してください）、ホストコンピュータのオペレーティングシステムについての実践的な知識が必要です。

**注：**本ガイドの説明の一部は、特定のマイクロコントローラや IAR Embedded Workbench の特定の派生製品パッケージにのみ該当します。たとえば、C++ をサポートしていないパッケージがありますがここではこの機能について説明しています。

## 表記規則

本ガイド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。

製品インストール先のディレクトリ（例：ターゲット¥doc）の記述がある場合、その場所までのフルパス（例：c:¥Program Files¥IAR Systems¥Embedded Workbench N.n¥ ターゲット¥doc）を意味します。



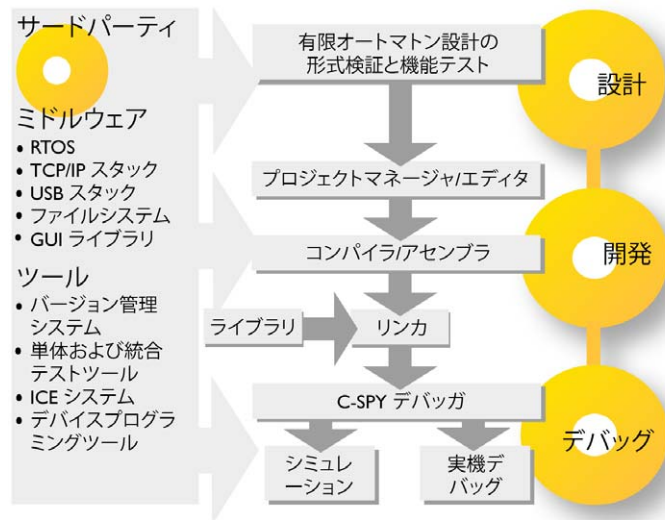
# 概要

- 製品ポートフォリオの概要
- デバイスサポート
- チュートリアル
- ユーザドキュメンテーション
- その他のリソース

インストールとライセンスについては、製品に同梱されているクリックリファレンスを参照してください。

## 製品ポートフォリオの概要

次の図は、IAR システムおよびサードパーティベンダのさまざまなツールと、それらがどのように相互に動作するかを示します。

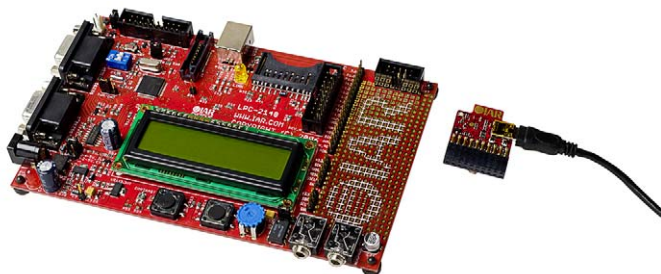


**IAR Embedded Workbench** は、組込みシステムの完全なアプリケーションプロジェクト開発および管理を可能にする統合開発環境を提供します。この環境は、コンパイル、リンク、デバッグのためのツールから構

成され、総合的かつ専用のターゲットをサポートしています。どのマイクロコントローラを使用するかに関係なく、ユーザインタフェースは同じです。

**IAR visualSTATE** は、有限オートマトンに基づいて組込みアプリケーションを設計、テスト、実装するための開発ツールのセットが統合されています。形式検証および検証ツールを提供し、設計から C ソースコードを生成します。使用を開始するあたって、製品のインストールにはチュートリアルや入門ガイド、いくつかのサンプルが用意されており、一般的な内容と特定の評価ボード用に分かれています。

**IAR 評価キット**は、特定のマイクロコントローラ用に組込みアプリケーションを開発するための統合型キットです。各キットには評価ボード、サンプルプロジェクトを含むソフトウェア開発ツール、ハードウェアデバッグプローブまたはエミュレータが含まれています。



**I-jet** はインサーキットデバッグプローブで、チップの JTAG/SWD ポートを USB 経由でホスト PC に接続します。I-jet は IAR Embedded Workbench に統合されており、プラグアンドプレイの互換性があります。

## サードパーティツールおよびユーティリティ

さまざまなサードパーティツールおよびユーティリティを、IAR Embedded Workbench に統合できます。こうした製品には、バージョン管理システム、エディタ、RTOS 対応デバッグおよびさまざまなデバッグプローブ用の C-SPY プラグインモジュール、プロトコルスタックなどがあります。

## 他のビルドツールとの相互運用

製品で使用するオブジェクト形式およびアプリケーションバイナリのインタフェースによっては、IAR ツールチェーンは他のベンダのツールチェーンと相互に運用可能です。相互運用のレベルは、



サードパーティのデバッガでの IAR ツールの実行可能ファイルのデバッグからリンクレベルの完全な互換性までさまざまです。

## デバイスサポート

製品開発を円滑に始めるために、IAR 製品にはさまざまなデバイス用に事前設定されたファイルが含まれています。

**周辺 I/O ヘッダファイル**は、周辺ユニットを定義するデバイス固有の I/O ヘッダです。

**リンカ設定ファイル**には、コードとデータをメモリに配置するためにリンカに必要な情報が含まれています。製品パッケージによっては、リンカ設定ファイル用のテンプレート、またはサポートされているデバイス用のすぐに使用可能なリンカ設定ファイルが提供されています。

**デバイス記述ファイル**は、周辺レジスタおよびそれらのグループの定義など、デバッグに必要なデバイス固有の詳細な部分を取り扱います。つまり、デバッグの実行中に SFR アドレスやビット名を参照できます。

**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、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。製品パッケージに応じて、一部のデバイス向けフラッシュローダが利用できます。使用するデバイスが含まれていない場合は、独自のフラッシュローダをビルドできます。一部のデバッグプローブは対応する機能を提供しますので、専用のフラッシュローダは必要ありません。

**導入のためのサンプルプロジェクト**(ソフトウェア開発用)を用意して、スムーズに開始できるようにしています。製品パッケージに応じて、数個から数百個の実際のソースコードのサンプルが含まれており、複雑さは単純な LED の点滅から USB のマスタストレージコントローラまでさまざまです。これらのサンプルには、[Help] (ヘルプ) メニューから表示できるインフォメーションセンタからアクセスできます。

## チュートリアル

チュートリアルでは、IAR Embedded Workbench IDE とそのツールの基本的な使用方法を確認するための実践トレーニングを提供します。チュートリアルは異なるパートに分かれており、チュートリアル全体をまとめて使用するか、個々のチュートリアルを見るよう選択することもできます。チュートリアルは、ハードウェアを用意せずにデバッガの使用を開始できるように、C-SPY シミュレータ用に設定されています。

チュートリアルには、インフォメーションセンタ (IDE の [Help] (ヘルプ) メニューにより表示) からアクセスできます。チュートリアルに必要なファイルはすべて `ターゲット\tutor` ディレクトリにあります。

## ユーザドキュメンテーション

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 (IAR Embedded Workbench IDE の **[Help]** (ヘルプ) メニューにより表示) からアクセスできます。オンラインヘルプシステムにも、IDE で F1 キーによりアクセスできます。

ユーザドキュメンテーションに含まれていない可能性のある直前の変更については、**[Help]** (ヘルプ) メニューからアクセスできるリリースノートを読むことをお勧めします。

すべてのドキュメンテーションは、ターゲット¥doc と common¥doc の各ディレクトリにあります。

## その他のリソース

IAR システムズの Web サイト (<http://www.iarsys.co.jp/customer>) では、テクニカルノートやアプリケーションノートを見ることができます。

保守契約 (SUA) を締結している場合、IAR システムズの Web サイトの MyPages から、最新の製品情報にアクセスして製品のアップデートおよび新しいデバイスのサポートファイルをダウンロードすることも可能です。

## 技術サポートのリクエスト

IAR システムズのツールに問題があった場合は、以下のトラブルシューティングのヒントのリストをチェックしてください。

- 1 まずユーザドキュメントの該当トピックをご覧ください。ガイドラインについては、インフォメーションセンタを参照してください。
- 2 リリースノートの「既知の問題」の項を読んで、報告済みの内容に関連があるかどうか調べます。
- 3 問題が解決しない場合、問題点の切り分けをしてみてください。再現する小さなコードのサンプル、プロジェクト設定、問題の再現方法についての説明があると、タイムリにサポートを提供する上で非常に役に立ちます。
- 4 Web サイト経由、または IAR システムズの担当者に連絡してレポートを送信してください。製品名、バージョン、ライセンス番号、保守契約者番号を必ず含めるようにしてください。

# IAR Embedded Workbench の概要

本章では、IAR Embedded Workbench® に含まれるさまざまなツールの概要を説明します。

## IDE

IDE は、アプリケーションのビルドに必要なすべてのツールが統合されたフレームワークです。C/C++ コンパイラ、アセンブラ、リンカ、ライブラリツール、エディタ、プロジェクトマネージャ、IAR C-SPY® デバッガが含まれます。

製品パッケージに付属のツールチェーンは、特定のマイクロコントローラをサポートしています。IDE では、さまざまなマイクロコントローラに対する複数のツールチェーンを同時に格納できます。つまり、いくつかのマイクロコントローラ用に IAR Embedded Workbench をインストールしている場合、どのマイクロコントローラ向けに開発するかを選択できます。

**注：**ビルド済みプロジェクト環境で外部ツールとして利用する場合は、コンパイラ、アセンブラ、リンカ、C-SPY デバッガを、コマンドライン環境で実行することもできます。

## IDE を起動するには：

Windows の [スタート] メニューのスタートボタンをクリックするか、ワークスペースのファイル名（ファイル拡張子 `eww`）をダブルクリック、あるいは `IarIdePm.exe` というファイル（IAR Embedded Workbench の `common\bin` ディレクトリにあります）を使用します。

IDE のメインウィンドウが開きます。



最初に IAR Embedded Workbench を開くと、IDE のメインウィンドウに IAR インフォメーションセンタが表示されます。ここでは、チュートリアルやサンプルプロジェクト、サポート情報、リリースノートなど、必要な情報がすべて得られます。

## IDE の設定

好みや要件に合わせて IDE を設定する方法は数多くあります：

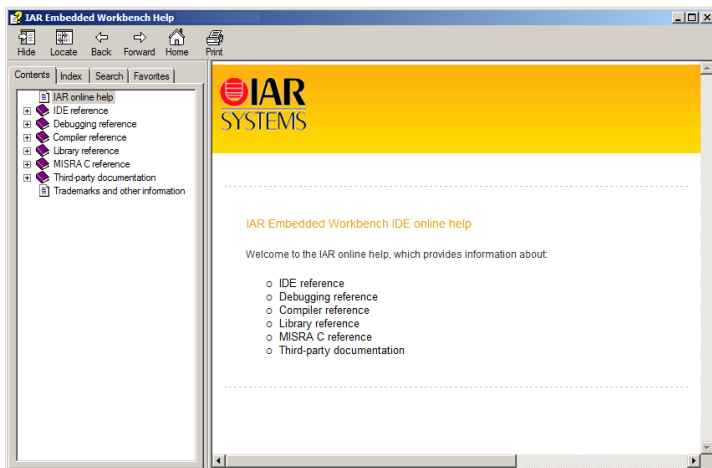
- ウィンドウの編成 — ウィンドウは特定の位置にドッキングして、タブグループとして編成できます。また、ウィンドウをフローティング化することができます。フローティングウィンドウは、常に他のウィンドウよりも前に表示されます。フローティングウィンドウのサイズや位置を変更しても、現在開かれている他のウィンドウは影響を受けません。メインウィンドウの下端にあるステータスバーには、ウィンドウのサイズを変更するためのヘルプが用意されています。

- レビジョン管理システムやエディタなど、外部ツールでツールチェーンを拡張。また、IAR visualSTATE をツールチェーンに追加して、IDE でステートマシンダイアグラムをプロジェクトに直接追加することも可能です。
- [Tools] (ツール) メニューからの外部ツールの呼出し。
- コマンドによる IDE のカスタマイズ。たとえば次のものが使用できます。
  - エディタの設定
  - 共通フォントの変更
  - キーカスタマイズの変更
  - 任意の外部エディタの使用
  - プロジェクトビルドコマンドの設定
- [Messages] (メッセージ) ウィンドウへの出力数の設定

オンラインヘルプシステムを参照するには：

[Help]>[Content] (ヘルプ>目次) を選択するか、IDE でウィンドウまたはダイアログボックスをクリックして F1 キーを押します。

オンラインヘルプシステムが表示されます：



たとえば次のようなコンテキストに合ったヘルプ情報が表示されます。

- IDE と C-SPY
- コンパイラ

- ライブラリ
- MISRA-C

## IAR C/C++ コンパイラ

### プログラミング言語

ほとんどの製品パッケージでは、IAR C/C++ コンパイラとともに使用可能な高度なプログラミング言語が2つあります。

**C言語は：**組込みシステム業界で最も幅広く使用されている高級プログラミング言語です。以下の標準に準拠したフリースタンディングアプリケーションのビルドが可能です。

- 標準 C – C99 とも呼ばれる標準の C。本ガイドでは、この規格を *C 規格* と呼びます。
- C89 – C94、C90、C89、ANSI C とも呼ばれ、この規格は MISRA-C が有効なときに必須です。

**C++** (製品パッケージによって異なります)。以下のすべての標準が使用できます。

- 標準 C++ – 例外およびランタイム型情報で、異なるレベルのサポートを使用できます (製品パッケージによって異なります)。
- Embedded C++ (EC++) – C++ プログラミング標準のサブセット。業界団体である Embedded C++ Technical Committee により定義されています。
- IAR 拡張 Embedded C++ – テンプレート、多重継承 (製品パッケージによって異なります)、名前空間、新しいキャスト演算子、標準テンプレートライブラリ (STL) などの追加機能をサポートしています。

## MISRA-C

MISRA-C は、安全性を重視するシステムの開発に適した規則のセットです。MISRA-C を構成する規則は、C プログラミング言語の ISO 標準により厳しい安全対策を適用することを意図しています。使用する製品パッケージに応じて、MISRA-C:2004 と MISRA-C:1998 の両方をサポートしています。

### コンパイラ拡張

コンパイラには、C/C++ 言語の標準機能に加えて、広範な拡張機能が用意されています。

**C 言語拡張**は次の2つのグループに分類できます。

- **組込みシステムプログラミングの拡張** — 一般的にメモリの制限を満たしたり、割込みなど特殊な関数を宣言するために使用する特定のマイクロコントローラでの効率的な組込みプログラミングに特化した拡張。
- **C 規格に対する緩和** — 規格を緩和した軽微な言語拡張、わずかな構文の拡張で得られる便利な機能。

**プラグマディレクティブ**は、ソースコードが移植可能であることを徹底する目的で、ベンダ固有の拡張機能で使用されるためにC規格で定義されたメカニズムです。事前定義されたディレクティブは、コンパイラの動作（メモリの配置方法、拡張キーワードの許可/禁止、ワーニングメッセージの表示/非表示など）を制御します。

プラグマディレクティブは、コンパイラでは常に有効になっています。

**プリプロセッサの機能**以下に例を示します。

- 事前定義されたプリプロセッサシンボルで、コンパイルの時間や異なるコンパイラ設定など、コンパイル時間の環境を調べることができます。
- #define ディレクティブに加えてコンパイラオプションまたはIDEで定義される、ユーザ定義のプリプロセッサシンボル。

**マイクロコントローラの低レベルの機能へのアクセス**は、不可欠です。コンパイラは、低レベルプロセッサの処理に直接アクセス可能にする組込み関数、Cモジュールとアセンブラモジュールの混在、インラインアセンブラなどの方法でこれをサポートしています。使用方法を慎重に選択してください。

## IAR アセンブラ

IAR アセンブラは、使用するマイクロコントローラに対して、柔軟なディレクティブと式演算子のセットを使用してマクロアセンブラを再配置します。C言語プリプロセッサを内蔵しており、条件アセンブリをサポートしています。

アセンブラは、シンボリックアセンブラ言語ニーモニックを実行可能なマシンコードに変換します。効率的なアセンブラアプリケーションを記述するためには、使用するマイクロコントローラのアーキテクチャと命令セットを理解しておく必要があります。

アプリケーション全体をアセンブラ言語で記述するのではない場合でも、正確なタイミングや特殊な命令シーケンスを要求するマイクロコントローラのメカニズムを使用する場合など、コードの一部をアセンブラで記述する必要が生じることがあります。

## IAE リンカと関連ツール

使用するパッケージに応じて、IAE Embedded Workbench には XLINK リンカまたは ILINK リンカのどちらかが用意されています。

これらはサイズが小さい 1 ファイルのアセンブラアプリケーションのリンクと、大規模で再配置可能な複数モジュール、C/C++、アプリケーションや、C/C++ とアセンブラ混在アプリケーションのリンクの両方に適しています。

どちらのリンカも、設定ファイルを使用します。このファイルで、ターゲットシステムのメモリマップのコードおよびデータ領域に個別の位置を指定し、コードとデータ配置を完全に制御することができます。

リンクする前に、入力順に関係なく（ライブラリを除く）、リンカはすべての入力ファイルにあるあらゆるシンボルの依存関係の完全な解決を実行します。また、すべてのモジュールについてコンパイラ設定の整合性をチェックし、正しいバージョンおよび派生品の C/C++ ランタイムライブラリが使用されていることを確認します。

リンカは、リンクするアプリケーションが実際に必要なライブラリモジュール（ユーザライブラリおよび標準 C/C++ の派生ライブラリ）だけを自動的にロードします。正確に言うと、実際に使用されるライブラリモジュールの関数のみがロードされます。

IAE ILINK リンカは、1 つまたは複数の再配置可能なオブジェクトファイルを、1 つまたは複数のオブジェクトライブラリの選択した部分と組み合わせて、実行可能イメージを生成します。

ILINK が作成する最終出力は、ELF（デバッグ情報の DWARF を含む）形式の実行可能なイメージを含む、絶対オブジェクトファイルです。このファイルは、ELF/DWARF をサポートする C-SPY などのデバッガにダウンロードできます。あるいは、任意の適切なフォーマットに変換した後で EPROM にプログラミングすることができます。

ELF ファイルを処理するために、アーカイバ、ELF ダンプ、フォーマットコンバータなどのさまざまなユーティリティがあります。

使用する製品パッケージによっては、多くのアプリケーションについて ILINK が最大スタック使用量を計算できます。



**IAR XLINK リンカ**は、IAR システムズのコンパイラまたはアセンブラによって生成された1つまたは複数の再配置可能オブジェクトファイルを結合して、使用するマイクロコントローラ用にマシンコードを生成します。**XLINK** は、**C-SPY** デバッガで使用される **UBROF** のほかに、30 以上の業界標準のローダフォーマットを生成できます。

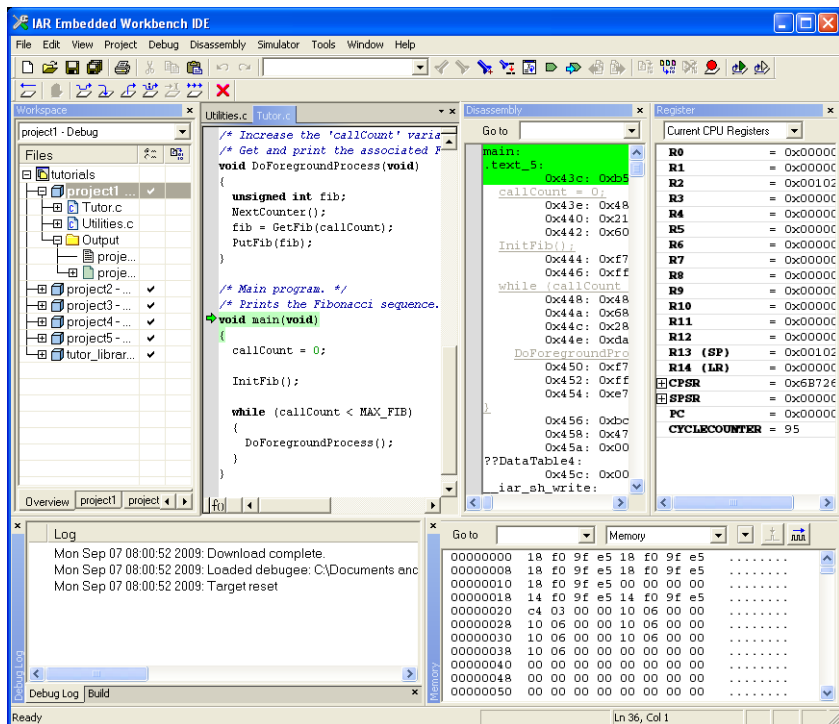
また、**XLINK** はすべてのモジュールに対して詳細な C/C++ タイプのチェックを実行します。

**XLINK** が生成する最終出力は、マイクロコントローラやハードウェアエミュレータにダウンロードでき、ターゲットで実行可能な絶対オブジェクトファイルです。出力ファイルに、選択した出力フォーマットに応じたデバッグ情報を含めるかどうかを選択することもできます。

ライブラリを処理するために、ライブラリツール **XAR** と **XLIB** が含まれています。

## IAR C-SPY デバッガ

IAR C-SPY デバッガは、組み込みアプリケーション開発用の高級言語デバッガです。

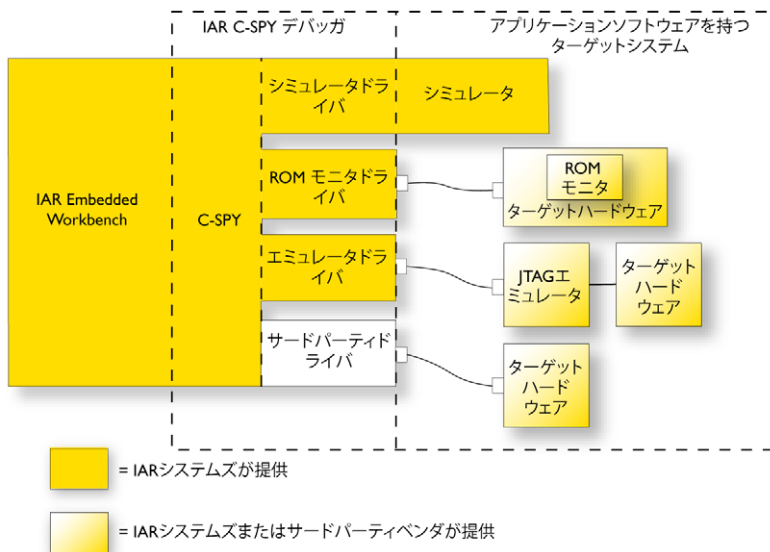


IAR システムズのコンパイラやアセンブラと併せて使用するように設計されており、IDE に完全統合されているため、開発とデバッグをシームレスに切り替えることができます。このため、以下のような操作が可能です。

- 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。
- デバッグ起動前に、ソースコードにブレークポイント設定可能。ソースコード中のブレークポイントは、コードを追加した後も、ソースコードの同一部分に対応付けられます。

C-SPY は、デバッガの基本機能セットを提供する部分とドライバで構成されています。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、専用のメニュー、ウィンドウ、ダイアログボックスを提供します。

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。



製品パッケージによっては、C-SPY と併せて、シミュレータドライバやさまざまなハードウェアデバッガシステム用の追加ドライバもインストールされます。

C-SPY の詳細については、本ガイドの「49 ページの デバッグ」を参照してください。

## C-SPY プラグインモジュール

C-SPY は、モジュール型のアーキテクチャとして設計されています。デバッガに追加機能を実装するための SDK（ソフトウェア開発キット）は、プラグインモジュールという形で提供されています。これらのモジュールは IDE に統合可能です。

IAR システムズが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。

- [Code Coverage]（コードカバレッジ）、[Symbols]（シンボル）、[Stack]（スタック）プラグイン（これらはすべて IDE に問題なく統合されます）。
- 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 対応のデバッグのための RTOS プラグインモジュール。
- 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。

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。

# 組込みアプリケーションの開発

組込みアプリケーションソフトウェアの開発を始める前に、次の概念に目を通すことをお勧めします。

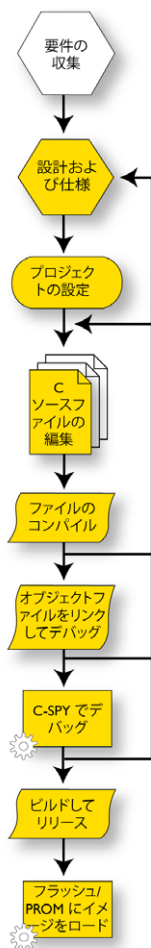
- 開発サイクル
- 一般的に使用されるソフトウェアモデル
- ビルド処理
- パフォーマンス重視のプログラミング
- ハードウェアおよびソフトウェア要因の考慮
- アプリケーションの実行

## 開発サイクル

実際の開発が始まる前に、要件と設計をまとめてアプリケーションアーキテクチャを指定（手動または visualSTATE などの自動コード生成ツールを使用）する必要があります。こうすることで、IAR Embedded Workbench IDE を使用する準備が整います。

以下は一般的な開発サイクルです。

- 一般オブジェクトおよびツール固有のオブジェクトを含むプロジェクトを設定する
- C/C++ またはアセンブラでソースコードファイルを作成
- デバッグのためにプロジェクトをビルド（コンパイルおよびリンク）
- ソースコードのエラーを修正
- アプリケーションのテストおよびデバッグ
- リリース用にビルド
- イメージをフラッシュメモリまたは PROM メモリにロード



## 一般的に使用されるソフトウェアモデル

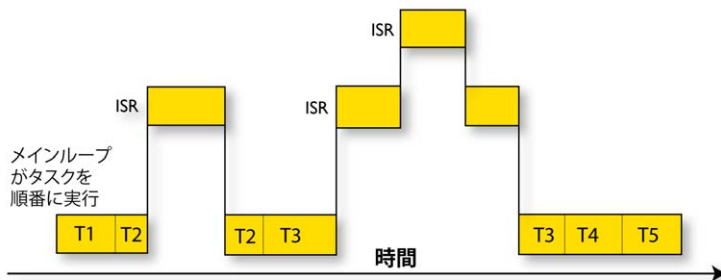
以下は一般的に使用されるソフトウェアモデルの一部です。

- スーパーループシステム (タスクが連続して実行されます)
- マルチタスクシステム (タスクはリアルタイム OS によってスケジュールされます)
- 有限オートマトンモデル

通常は、スーパーループシステムまたはマルチタスクシステムを使用し、アプリケーションのロジックを整理する一般的な方法は有限オートマトンを使用して設計するやり方です。

## スーパーループシステム

マルチタスキングカーネルがなくては、CPU によって一度に 1 つのタスクしか実行できません。これを、シングルタスクシステムまたはスーパーループと呼び、基本的には終わらないループで実行され、適切な操作を連続して実行するプログラムです。リアルタイムカーネルが使用されないため、ソフトウェアや重要な処理のリアルタイム部分には割り込みサービスルーチン (ISR) を使用する必要があります (割り込みレベル)。



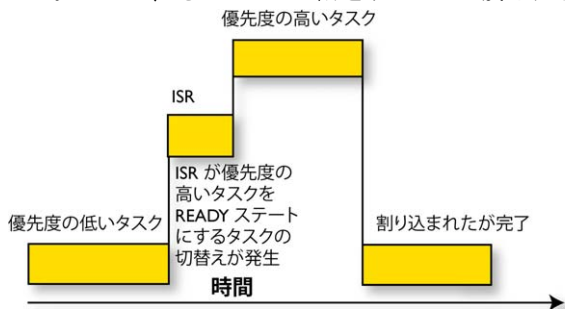
スーパーループは、プログラムが大きくなると保守が難しくなる可能性があります。あるコンポーネントが動作している場合、ISR 以外の他のコンポーネントは割り込むことができないため、1 つのコンポーネントの応答時間はシステムの他のすべてのコンポーネントの実行時間に依存します。このため、リアルタイム性が劣ります。

このタイプのシステムは通常、リアルタイムの動作が重要でない場合に使用されます。

## プリエンティブマルチタスクシステム

リアルタイムオペレーティングシステムを使用する際、単一の CPU 上で複数のタスクを同時に実行できます。すべてのタスクは、CPU 全体を完全に所有するかのごとく実行されます。RTOS は各タスクを有効/無効にすることによりスケジューリングします。マルチタスクシステムでは、異なるスケジューリングアルゴリズムがあり、CPU の算出能力はタスク間で分散されます。

リアルタイムシステムは、プリエンティブタスクとして処理をしています。リアルタイムオペレーティングシステムには、定義済みの時間にタスクに割り込み、必要があればタスク切替えを実行する通常のタイマ割り込みが必要です。そのため、割り込みタスクかどうかに関係なく、READY ステートで最も優先度の高いタスクは常に実行されます。割り込みサービスルーチン (ISR) でより高い優先度のタスクが準備されると、タスク切替えが発生して、そのタスクは割り込みタスクが戻される前に実行されます。



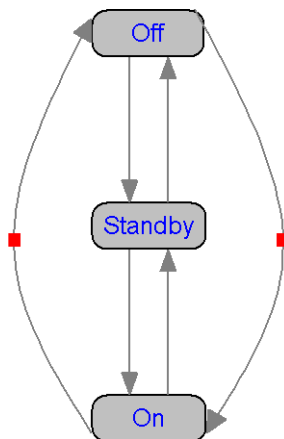
## 有限オートマトンモデル

有限オートマトンモデルは、単純に入ってくるイベントを推定される外向きのアクションに変換します。これは純粋に反作用を示すエンジンまたはコアであり、オペレーティングシステムとは混同しないでください。どの時点でも、システムは可能なステートのいずれかにあります。環境からの入力に応じて、システムはステートを変更することができます。ステートの変更が発生すると同時に、環境についてアクションを実行できます。

たとえば、電子機器のサブシステムは On または Off で、ドアは Open または ClosedAndUnlocked などというようにです。有限オートマトンでは、可能な物理ステートをすべてマップする必要はなく、ソリューションにとって重要なステートのみをマップすればすみます。

有限オートマトンモデルの重要な特徴のひとつは、並列処理を行う能力です。この場合において、並列処理という言葉は複数のパラレルステートシステムを同時に処理することを指します。

たとえば、自動販売機において考慮しなければならないすべてのケースを想定してください。



- カップがいっぱいになる前に取り出された
- 支払を処理中に顧客が注文をキャンセルしたときのクレジットカード口座の処理
- 前の注文が完了する前に新規の注文が開始された場合
- 注文の最中にいずれかの電子部品によってマシンが停止した場合、料金が顧客に返金されるかどうか

ステートチャート図は高度なレベルから設計を見ることができ、複雑性を処理するために必要な全体像を維持することが可能です。いったんステートチャートモデルが作成されると、意図したとおりに動作するかどうか検証することができます。また、IAR visualSTATE は設計と 100% 一致する C/C++ ソースコードを生成します。

ステートマシンの使用は、信頼性、サイズ、決定論的な実行が主要な基準であるモニタリング、測定、制御などのように、論理指向のアプリケーションに非常に有効です。

## ビルド処理

このセクションでは、ビルドプロセスの概要について説明します。つまり、コンパイラ、アセンブラ、リンカなどさまざまなビルドツールがどのように組み合わせたり、ソースコードから実行可能イメージに移行するかについて説明します。ビルド処理はさらに以下に分類されます。



- 変換プロセス
- リンク処理
- リンク後

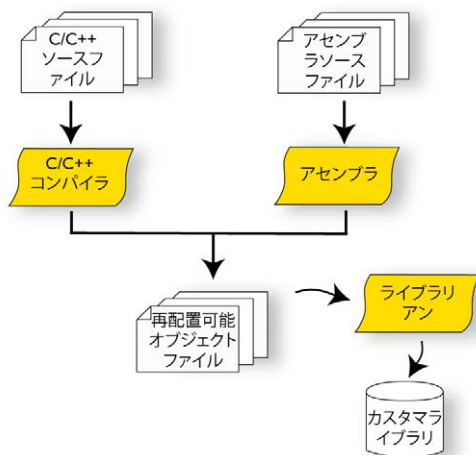
## 変換プロセス

アプリケーションソースファイルを中間オブジェクトファイルに変換するツールは IDE に 2 つあります。それは、IAR C/C++ コンパイラおよび IAR アセンブラです。どちらも、ILINK を使用する製品には ELF/DWARF フォーマットで、XLINK を使用する製品には UBROF フォーマットで再配置可能オブジェクトファイルをそれぞれ生成します。

**注：** このコンパイラは、C/C++ ソースコードをアセンブラソースコードに変換するときにも使用できます。必要な場合、アセンブラソースコードを修正してからオブジェクトコードにアセンブルできます。

以下の図は、変換プロセスを示します。

変換後は、任意の数のモジュールを 1 つのアーカイブ、つまりライブラリにパッキングしてファイルを整理することができます。



## リンク処理

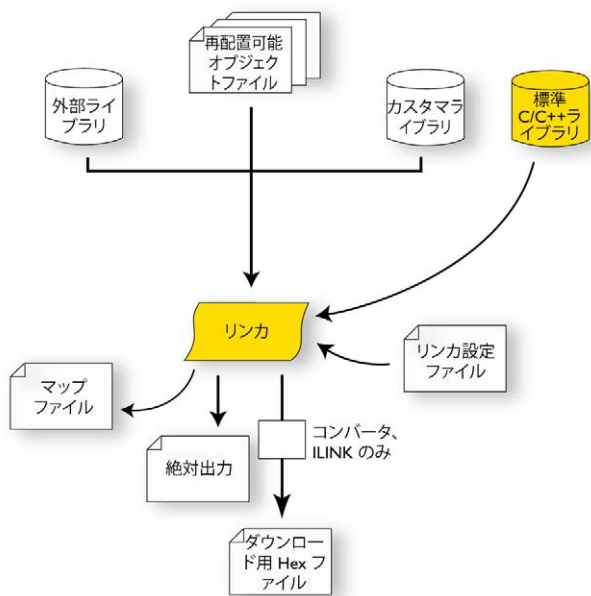
IAR コンパイラおよびアセンブラにより生成されるオブジェクトファイルおよびライブラリの再配置可能モジュールは、そのまま実行することはできません。これらが実行可能なアプリケーションとなるには、リンクされる必要があります。

**注：** 製品パッケージに応じて、別のベンダのツールセットにより生成されたモジュールもビルドに含めることができます。ただし、AEABI 準拠でない場合、同じベンダのコンパイラユーティリティライブラリが必要な点に注意してください。

リンカは、最終的なアプリケーションのビルドに使用します。通常は、リンカは入力として以下の情報を必要とします。

- いくつかのオブジェクトファイル、場合によってはライブラリ
- プログラムの開始ラベル (ILINK リンカのデフォルト設定、および XLINK リンカのユーザ設定)
- ターゲットシステムのメモリ内でのコードおよびデータの配置を記述したリンク設定ファイル

以下の図はリンクプロセスを示します。



**注：** 標準 C/C++ ライブラリには、コンパイラのサポートルーチンと、C/C++ 標準ライブラリ関数の実装が含まれます。

ILINK リンカは、実行可能イメージを含む ELF フォーマットの絶対オブジェクトファイルを生成します。XLINK は、C-SPY デバッガで使用される IAR システム独自のデバッグフォーマット、UBROFに加えて、30 以上の業界標準のローダフォーマットを生成できます。

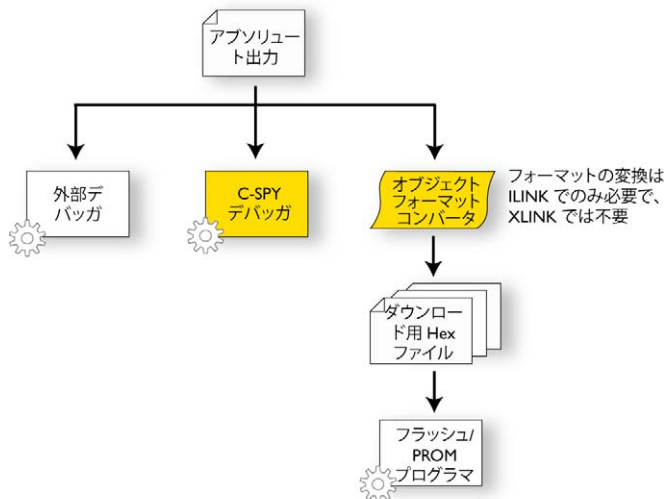
リンクの途中で、リンクが stdout および stderr に関するエラーメッセージを生成することがあります。ILINK はログメッセージも生成し、これはアプリケーションがなぜリンクされたかを理解する場合、たとえば、モジュールが含まれた理由やセクションが削除された理由を理解するときに役に立ちます。

## リンク後

リンク後、生成された絶対実行可能イメージは以下のことに使用できます。

- IAR C-SPY デバッガ、または生成されたフォーマットを読み取るその他の外部デバッガへのロード。
- フラッシュ /PROM メモリを使用したフラッシュ /PROM へのプログラミング。ILINK を使用してリンクする際、これを実現するには、イメージの実際のバイトを標準の Motorola S-record フォーマットまたは Intel-Hex フォーマットに変換する必要があります。XLINK はこれらのフォーマットのどれでも直接生成できるため、これ以外の変換は不要です。

次の図は出力の用法を示します。



## パフォーマンス重視のプログラミング

ここでは、以下についてのヒントを提供します。

- データ型の適切な使用
- レジスタ定義の容易化
- コンパイラ変換の容易化

## データ型の適切な使用

**データサイズ**は適切に使用する必要があります。8 ビットの処理は、たいていは 32 ビットの CPU 上では非効率的です。逆に、32 ビットの処理は 8 ビットの CPU にとっては効率が悪くなります。整数定数を使用する場合、36L というように必ず適切なサフィックスを追加してください。

**符号付きの値**は、マイナスの値が使用できることを意味します。ただし、コンパイラがこうした値を変更する場合、算術演算が使用されます。符号なしの値の場合、コンパイラはビットおよびシフト演算を使用します。これらは算術演算よりも通常は効果的です。マイナスの値が必要ない場合、符号なしの型を使用してください。

**浮動小数点数の処理**は通常、大きなライブラリ関数が必要になりかねないため、非常にコストがかかります。こうした演算を整数演算（より効率的です）に置き換えることを検討してください。

**メモリ配置およびポインタ型**は、8 ビットおよび 16 ビットのアーキテクチャ上では効率が上がり、小さいメモリ領域や小さいアドレス、小さいポインタを目指す場合は生成されるコードも小さくなります。最も大きいメモリ型やポインタの使用は避けてください。

**キャスト**でポインタを起点にしたり、異なる型の式を混在させることは避けるべきです。こうすると非効率的なコードが生成され、情報が失われるリスクがあります。

**構造体におけるパディング**は、CPU でアラインメントが必要な場合に発生します。メモリを浪費しないように、フィールドをサイズ別に整列して、パディングに使用されるメモリ量が最小限になるようにしてください。

## レジスタ定義の円滑化

**関数パラメータとローカル変数**（グローバル変数とは対照的に）は、レジスタに配置できてスコープ内にある間だけ存在すればすむため、メモリ消費量が少なくなります。グローバル変数を使用すると、変数にアクセスする関数の呼出しのたびに更新の必要があるため、オーバーヘッドが発生します。

**可変引数**（printf スタイル）は避けるべきです。その理由は、引数がスタックに強制適用されるためです。強制適用されない場合、これらの引数はレジスタで引き渡されます。

## コンパイラ変換の円滑化

**関数プロトタイプ**は、型変換（暗黙のキャスト）が不要でソースコードの問題が発見しやすくなるため、使用するべきです。また、プロトタイプピングによって、コンパイラがより効率的なコードを生成しやすくなります。

**静的宣言された変数および関数**は、最高の最適化を達成するために、宣言されたファイルやモジュールでのみ使用するべきです。

**インラインアセンブラ**は、コンパイラがコードを最適化する際に大きな障害となることがあります。使用するコンパイラで `asm` キーワードがどのように機能するか確認してください。

**「賢い」ソースコード**は、分かりやすいコードと置き換えるべきです。分かりやすいコードは保守が簡単で、プログラミングのエラーを含む確率が少なく、たいていはコンパイラが最適化しやすいからです。

**volatile キーワード**は、同時にアクセスされる変数を保護するために使用してください。つまり、割込みルーチンや別のスレッドで実行されるコードなどによって、非同期でアクセスされる変数ということです。こうすれば、こうした変数がアクセスされる場合、コンパイラは常にメモリから読み込み、メモリに書込みます。

**空白のループ**は、遅延を達成するほかに何も影響を与えないコードで、コンパイラによって削除されるかもしれません。代わりに、OS サービス、組込み関数、CPU タイマを使用するか、`volatile` 宣言された変数にアクセスします。

**長い基本ブロック**は、可能であれば作成してください。基本ブロックは、関数呼出しを持たないソースコードの割り込まれないシーケンスです。これによって、より効率的なレジスタ定義と最適化の良好な結果につながります。

## ハードウェアおよびソフトウェア要因の考慮

通常、専用マイクロコントローラに記述される組込みソフトウェアは、何らかの外部イベントの発生を待機するエンドレスループとして設計できます。このソフトウェアは、ROM に置かれ、リセット時に実行されます。この種のソフトウェアを作成する際には、いくつかのハードウェア要因およびソフトウェア要因を考慮する必要があります。

## CPU 機能および制限

命令セットの相互作用や異なるプロセッサモードなど、使用するマイクロコントローラで利用可能な機能、およびアラインメントの制約は、しっかり理解しておく必要があります。これらを正しく設定するには、ハードウェアのマニュアルを読んで理解することが重要です。

コンパイラは、こうした機能を拡張キーワードやプラグマディレクティブ、コンパイラオブジェクトなどの手段によってサポートしています。

プロジェクトを IDE で設定するとき、使用するデバイスに適したデバイスオブジェクトを選択する必要があります。以下の選択は自動的に行われます。

- 使用するデバイスに合った CPU 固有のオプションの設定
- デフォルトのリンカ設定ファイルの決定（製品パッケージによって異なります）

製品パッケージによっては、ターゲット¥config ディレクトリにリンカ設定ファイルのテンプレート、または一部あるいはサポートされているすべてのデバイス用の定義済みリンカ設定ファイルが含まれています。これらのファイルのファイル拡張子は、xcl (XLINK) または icf (ILINK) です。

- デフォルトのデバイス記述ファイルの決定。

これらのファイルはターゲット¥config ディレクトリにあり、ファイル名の拡張子は ddf か svd です（製品パッケージによって異なります）。

## 内部および外部メモリのマッピング

組込みシステムには、通常、オンチップ RAM、外部 DRAM、外部 SRAM、外部 ROM、外部 EEPROM、フラッシュメモリなど、さまざまなタイプのメモリが含まれます。

組込みソフトウェア開発者は、これらのさまざまなメモリタイプの機能を理解する必要があります。たとえば、オンチップ RAM は、通常、他のメモリタイプより高速なので、時間重視のアプリケーションでは、頻繁にアクセスされる変数をこのメモリに配置することでメリットを得ることができます。逆に、アクセスは頻繁に行われないが、電源を切った後でもその値を保持する必要がある設定データは、EEPROM またはフラッシュメモリに保存する必要があります。

メモリを効率的に使用するため、コンパイラでは、関数とデータオブジェクトのメモリへの配置を制御するためのさまざまな仕組みを提供しています。リンカは、リンカ設定ファイルで指定したディレクティブに従って、メモリにコードとデータを配置します。

## 周辺ユニットとの通信

外部デバイスがマイクロコントローラに接続される場合、シグナル伝達用インタフェースを初期化および制御し、たとえば、チップを使用して、ピンを選択し、外部割込みシグナルを検出および処理して行います。通常、初期化および制御はランタイムに実行する必要があります。通常、これは特殊関数レジスタ (SFR) を使用して行います。これらのレジスタは、通常、チップ設定を制御するビットを含む、専用アドレスで使用できます。

標準の周辺ユニットは、デバイス専用の I/O ヘッドファイル (拡張子は `h`) で定義されており、ターゲット `¥inc` ディレクトリにあります。該当するインクルードファイルをアプリケーションソースファイルにインクルードしてください。追加の I/O ヘッドファイルが必要な場合は、既存のヘッドファイルをテンプレートとして使用することにより作成できます。

## 割込みの処理

組込みシステムでは、ボタン押下の検出など、外部イベントを即座に処理するために *割込み* を使用します。通常はコード中で割込みが発生すると、マイクロコントローラは実行中のコードを停止し、その代わりに割込みルーチンの実行を開始します。

コンパイラは専用のキーワードでプロセッサの例外タイプをサポートしています。つまり、割込みルーチンを `C` で記述できます。

## システム起動

すべての組込みシステムでは、アプリケーションの `main` 関数が呼び出される前に、システム起動コードが実行され、ハードウェアとソフトウェアの両方のシステムが初期化されます。

組込みソフトウェア開発者は、この起動コードを専用メモリアドレスに配置するか、ベクタテーブルからポインタを使用してアクセスできるようにしなければなりません。つまり、起動コードおよび初期ベクタテーブルは、ROM、EPROM、フラッシュメモリなど、不揮発性メモリに配置する必要があります。

`C/C++` アプリケーションでは、さらに、すべてのグローバル変数を初期化する必要があります。この初期化は、リンカおよびシステム起動コードで扱われます。詳細については、33 ページの *アプリケーションの実行* を参照してください。

## ランタイムライブラリ

製品パッケージに応じて、次の2つのライブラリのどちらか一方、または両々が用意されています。使用するライブラリを選択する必要があります。

- IAR DLIB ライブラリは標準のCとC++に対応しています。このライブラリは、IEEE 754 フォーマットの浮動小数点数もサポートしています。また、ロケール、ファイル記述子、マルチバイト文字などのさまざまなレベルのサポートを指定して構成することができます。
- IAR CLIB ライブラリは軽量ライブラリで、標準のCに完全に対応しているわけではありません。IEEE 754 フォーマットの浮動小数点数の数値にも完全には対応しておらず、Embedded C++ もサポートしていません。旧型のCLIBライブラリが提供されている場合は、旧バージョンとの互換性のためです。新しいアプリケーションプロジェクトには使用しないでください。

**注：**アセンブラソースコードだけで構成されたプロジェクトの場合は、ランタイムライブラリを選択する必要はありません。

ランタイムライブラリは、さまざまなプロジェクトの設定用にビルドされたビルド済みライブラリとして提供されます。IDEは、プロジェクトの設定に合ったライブラリを自動的に使用します。製品パッケージによっては、使用する構成にビルド済みライブラリがない場合があります。その場合は、自分でライブラリをビルドする必要があります。

製品パッケージによっては、ライブラリがソースファイルとして用意されていることもあります。これらはディレクトリターゲット `¥src¥lib` にあります。つまり、ライブラリをカスタマイズして自分でビルドできます。IDEは、ライブラリプロジェクトテンプレートを提供しています。これを使用して自作ライブラリのビルドが可能です。

## ランタイム環境

ランタイム環境は、アプリケーションを実行するための環境です。このランタイム環境は、選択したランタイムライブラリ、ターゲットハードウェア、ソフトウェア環境、アプリケーションのソースコードによって異なります。

最もコード効率の高いランタイム環境を設定するには、アプリケーションやハードウェアの要件を特定する必要があります。必要な機能が多いほど、コードのサイズも大きくなります。



必要なランタイム環境を構成するには、これを以下の方法でカスタマイズすると便利です。

- `scanf` 入力フォーマットおよび `printf` 出力フォーマットなど、ライブラリオブジェクトの設定。
- スタックサイズの指定（マイクロコントローラによっては複数のスタック）。

マイクロコントローラによっては、非静的自動変数をスタック上または静的オーバーレイエリアのどちらに配置するかも指定する必要があります。スタックは実行時に動的に割り当てられるのに対し、スタックオーバーレイエリアはリンク時に静的に割り当てられます。

- ヒープサイズ、およびメモリ内の配置場所の指定。製品パッケージによっては、複数のヒープを使用してそれらを異なるメモリ領域に配置することも可能です。
- `cstartup` などの特定のライブラリ関数を、自分でカスタマイズした関数でオーバーライドする。
- ロケール、ファイル記述子、マルチバイト文字など特定の標準ライブラリ機能のサポートレベルを、**Normal/Full**（DLIB ライブラリの場合のみ可能）のいずれかのライブラリ構成を選択して指定する。また、自分でライブラリ構成を作成することもできますが、ライブラリのリビルドが必要になります。この機能により、ランタイム環境を完全管理できます。

ハードウェア上でアプリケーションを実行するには、キャラクターベースの I/O に低レベルのルーチンを実装する必要があります（通常は CLIB の場合が `putchar` と `getchar` で、DLIB の場合は `__read` と `__write` です）。

## アプリケーションの実行

このセクションでは、組込みアプリケーションの実行の概要を以下の3つのフェーズに分けて説明します。

- 初期化
- 実行
- 終了

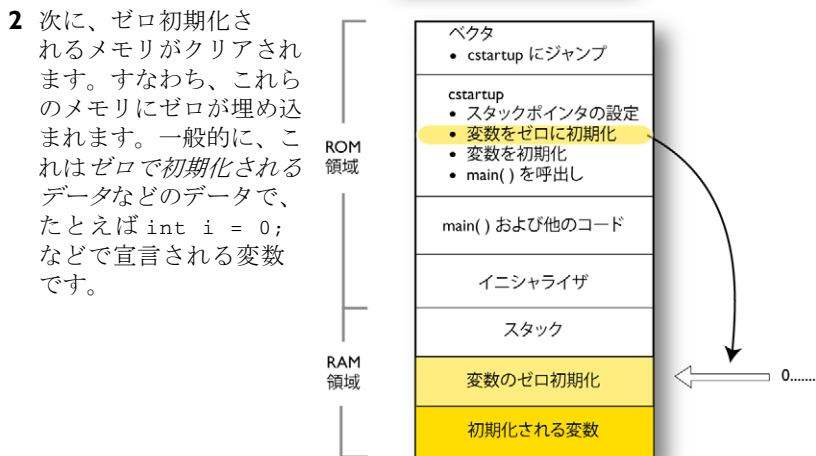
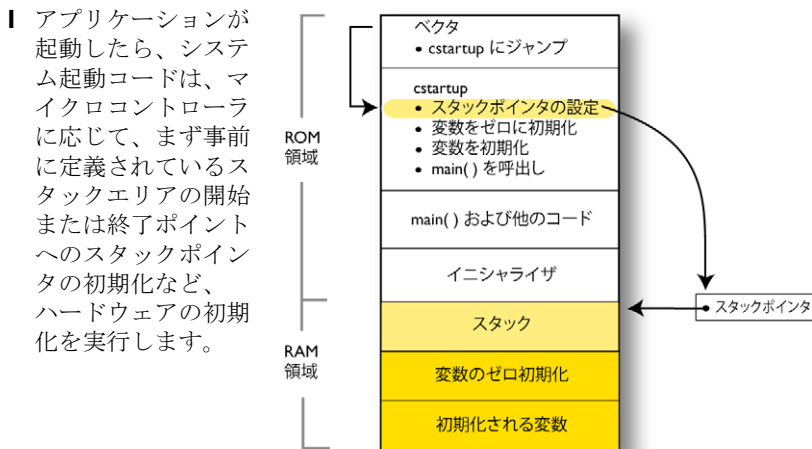
## 初期化フェーズ

初期化フェーズは、アプリケーションの起動時（CPUのリセット時）、main関数が入力される前に実行されます。初期化フェーズは、おおまかに以下のように分割できます。

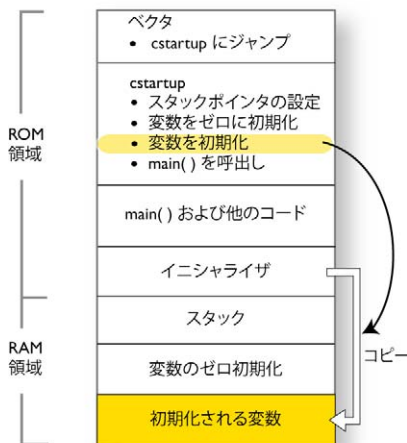
- ハードウェア初期化。たとえば、スタックポインタの初期化などハードウェア初期化は通常、システム起動コード `cstartup` で実行され、必要に応じて、ユーザが提供する追加の低レベルルーチンで実行されます。また、ハードウェアの残りの部分のリセット/起動や、ソフトウェア C/C++ システム初期化の準備のための CPU などの設定が行われる場合もあります。
- ソフトウェア C/C++ システム初期化  
一般的に、この初期化フェーズでは、main関数が呼び出される前に、すべてのグローバル（静的にリンカされた）C/C++ オブジェクトがその正しい初期化値を受け取っていることが前提です。
- アプリケーション初期化  
これは、使用しているアプリケーションにより異なります。一般的に、RTOS カーネルの設定や、RTOS が実行するアプリケーションの初期タスクの開始が含まれます。ベアボーンアプリケーションでは、さまざまな割込みの設定、通信の初期化、デバイスの初期化などが含まれます。

ROM/フラッシュベースのシステムでは、定数や関数がすでに ROM に配置されています。RAM に配置されたすべてのシンボルは、main関数が呼び出される前に初期化される必要があります。また、リンカにより、利用可能な RAM はすでに変数、スタック、ヒープなどの異なるエリアに分割されています。

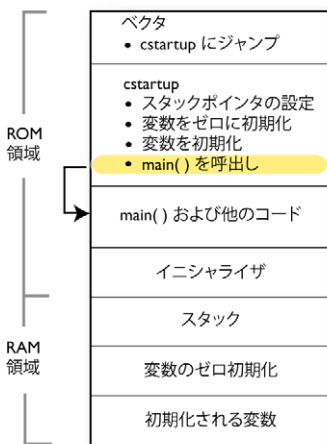
以下の一連の図は、初期化の各段階の概要を簡単に示します。図ではメモリのレイアウトは一般化されています。



- 3 初期化済みデータの場合、`int i = 6;` などゼロ以外の値で宣言されたデータは、イニシャライザが ROM から RAM にコピーされます。



- 4 最後に main 関数が呼び出されます。



## 実行フェーズ

組込みアプリケーションのソフトウェアは、通常、割り込み駆動型のループか、外部相互処理や内部イベントを制御するためのポーリングを使用するループのいずれかで実装されます。割り込み駆動型システムの場合、割り込みは、通常、main 関数の開始時に初期化されます。

リアルタイム動作システムで、応答性が重要な場合、マルチタスクシステムが必要になることがあります。つまり、アプリケーションソフトウェアは、リアルタイムオペレーティングシステムで補足する必要があります。この場合、RTOS およびさまざまなタスクは、main 関数の開始前に初期化される必要があります。

## 終了フェーズ

一般的に、組込み関数は実行を停止しません。終了する場合、正しい終了動作を定義する必要があります。

アプリケーションを制御したまま終了するには、標準 C ライブラリ関数の `exit`、`_Exit`、`abort` のいずれか呼び出すか、`main` から戻ります。`main` から戻ると、`exit` 関数が実行されます。すなわち、静的およびグローバル変数の C++ デストラクタが呼び出され (C++ のみ)、開いているすべてのファイルが閉じます。



# アプリケーションプロジェクトの作成

この章では、IDE でアプリケーションプロジェクトを設定するための開発サイクルについて解説します。開発サイクルは、以下の手順で構成されます。

- ワークスペースの作成
- 新しいプロジェクトの作成
- プロジェクトオプションの設定
- プロジェクトへのソースファイルの追加
- ツール固有のオプションの設定
- コンパイル
- リンク

ステップバイステップのチュートリアルのいずれかを使用して学習する場合は、[Help]（ヘルプ）メニューから表示できるインフォメーションセンタからチュートリアルにアクセスできます。

## ワークスペースの作成

[File]>[New]>[Workspace]（ファイル>新規>ワークスペース）を選択して、プロジェクトを追加するワークスペースを作成します。空白のワークスペースウィンドウが表示されます。

**注：**IDE を初めて起動したときは、既製のワークスペースが開いているので、それをプロジェクト用に使用できます。

プロジェクトを作成してワークスペースに追加する準備はこれで完了です。

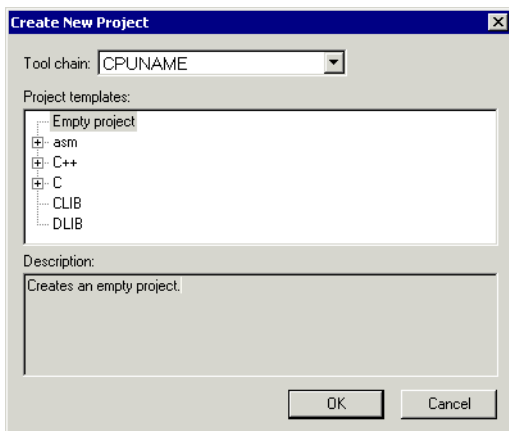
スムーズに開発を開始するためのサンプルが用意されています。製品パッケージに応じて、数個から数百個の実際のソースコードのサンプルが含まれており、複雑さは単純な LED の点滅から USB のマストレージコントローラまでさまざまです。使用する製品パッケージに応じて、サポートされているほとんどのデバイスのサンプルがあります。これらのサンプルには、[Help] (ヘルプ) メニューから表示できるインフォメーションセンタからアクセスできます。

## 新しいプロジェクトの作成

- 1 [Project]>[Create New Project] (プロジェクト>新規プロジェクト) を選択します。

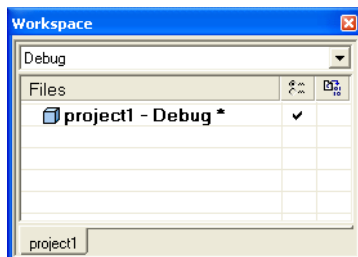
### [Tool chain]

(ツールチェーン) ドロップダウンリストから、使用するツールチェーンを選択します。複数のマイクロコントローラ用に IDE をインストールしている場合、それがドロップダウンリストに表示されます。



プロジェクトリストのリストから、新しいプロジェクトの基礎とするテンプレートを選択します。たとえば、[Empty project] (空白のプロジェクト) を選択すると、単にデフォルトのプロジェクト設定を使用した空のプロジェクトが作成されます。

- 2 プロジェクトを保存します。
- 3 [Workspace] (ワークスペース) ウィンドウにプロジェクトが表示されます。





デフォルトで、デバッグとリリースという2つのビルド構成が作成されます。これらを使用して、プロジェクト（プロジェクト設定およびビルドのファイルパート）の派生形を定義できます。独自のビルド構成を定義することもできます。構成は、ウィンドウ上端のドロップダウンメニューから選択します。

- 4 ファイルをプロジェクトに追加する前に、ワークスペースを保存してください。

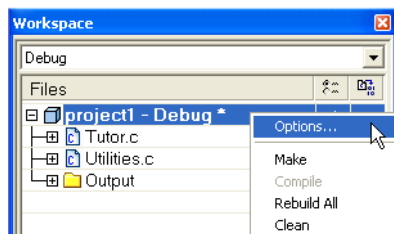
プロジェクトに関連する以下のファイルが作成されます。

- ワークスペースファイル（ファイル名の拡張子 `eww`）。このファイルには、ワークスペースに追加したすべてのプロジェクトがリストされます。
- プロジェクトファイル（ファイル名の拡張子 `ewp` と `ewd`）。これらのファイルには、ビルドオプションなど、プロジェクト固有の設定に関する情報が保存されます。
- ウィンドウの配置やブレークポイントなど、現在のセッションに関する情報は、`projects\settings` ディレクトリに作成されるファイルに保持されます。

## プロジェクトオプションの設定

ビルド構成全体に同じオプションを設定するには：

- 1 [Workspace]（ワークスペース）ウィンドウでプロジェクトフォルダのアイコンを選択し、右クリックして [Options]（オプション）を選択します。

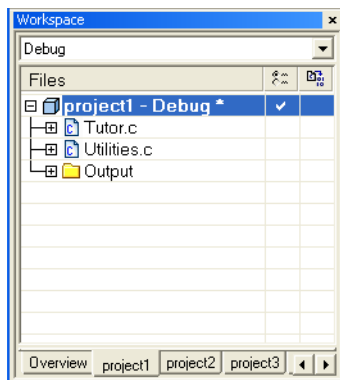


- 2 [General Options]（一般オプション）カテゴリには、ターゲットや出力、ライブラリ、ランタイム環境のオプションがあります。ここで行う設定は、ビルド構成全体と同じでなければなりません。

特に [Target]（ターゲット）ページで選択したデバイスによって、使用する製品パッケージに合わせて、デフォルトのデバッグデバイス記述ファイルとおよびデフォルトのリンク設定ファイルが自動的に決まる点に注意してください。さらに、選択したデバイスに合わせて他のオプションが自動的に設定されます。

## プロジェクトへのソースファイルの追加

- 1 [Workspace] (ワークスペース) ウィンドウで、ソースファイルの追加先としてグループかプロジェクトを選択します。この例では、直接プロジェクトに追加します。
- 2 [Project]>[Add Files] (プロジェクト>ファイルの追加) を選択して、標準の参照ダイアログボックスを開きます。ファイルを探し、[Open] (開く) をクリックしてそれらを自分のプロジェクトに追加します。



いくつかのファイルのグループを作成して、プロジェクトの要件に合わせてソースファイルを論理的に整理することができます。

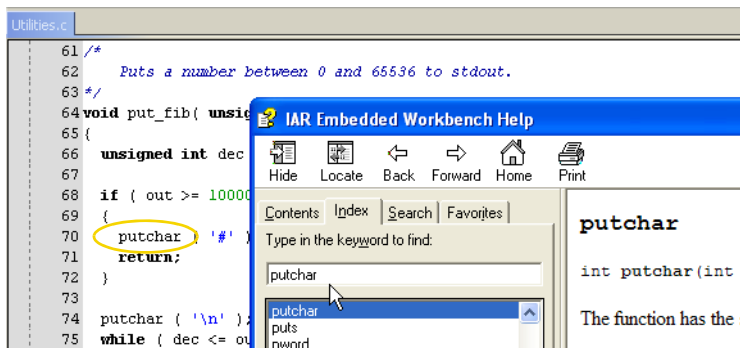
### 新しいドキュメントを作成するには：



ツールバーで [New Document] (新規ドキュメント) をクリックします。エディタウィンドウにファイルが表示されます。1つまたは複数のテキストファイルを作成または開くことができます。複数のファイルを開く場合、タブグループに整理されます。複数のエディタウィンドウを同時に開いておくことができます。

### 関数の参照を検索するには：

エディタウィンドウで、ヘルプが必要な項目を選択して F1 キーを押します。オンラインヘルプシステムが表示されます。



エディタウィンドウでは、すべての C または Embedded C++ のライブラリ関数、キーワードや組込み関数などすべてのコンパイラ言語拡張についてヘルプを参照できます。

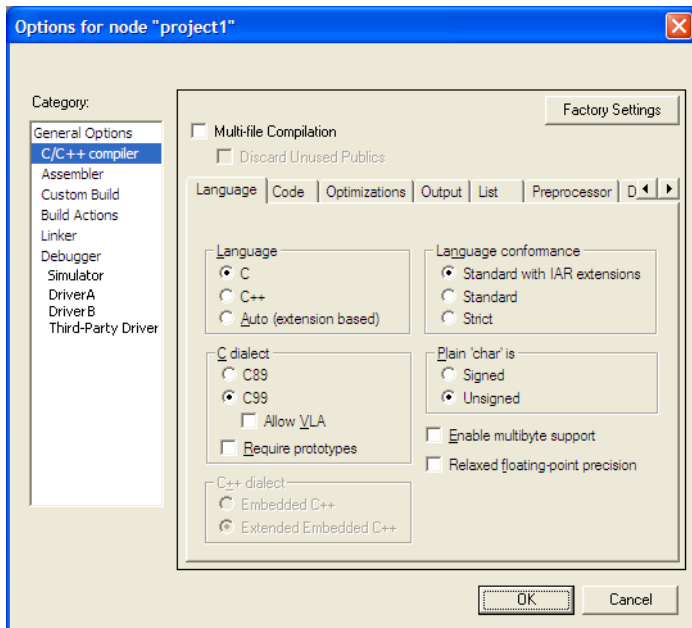
**エディタを設定するには：**

**[Tools]>[Options]** (ツール > オプション) を選択して、**[IDE Options]** (IDE オプション) ダイアログボックスで適切なオプションのカテゴリを選択します。

### **ツール固有のオプションの設定**

- 1 **[Workspace]** (ワークスペース) ウィンドウでプロジェクトを選択し、ファイルグループまたは個別のファイルを選びます。**[Project]>[Options]** (プロジェクト > オプション) を選択して、**[Options]** (オプション) ダイアログボックスを開きます。

- 2 **[Category]** (カテゴリ) リストでツールを選択し、適切なページで設定を行います。リストで使用可能なツールは、製品パッケージによって異なります。



ツールチェーンの標準ツールパートのほかに、ビルド前とビルド後のアクションのオプション、および外部ツールを呼び出すためのオプションを設定できます。

特定のコンパイラオプションを設定する前に、複数ファイルのコンパ



イルを使用するかどうかを指定できます。コンパイラの1回の呼び出しで複数のソースファイルのコンパイルを可能にすることにより、多くの場合に最適化がより効率的になります。ただし、ビルド時間に影響することがあります。このため、作業の開発フェーズではこのオプションを無効にしておくのが望ましいと言えます。

**注：** ご使用の製品パッケージが複数ファイルのコンパイルをサポートしない場合は、**[Multi-file Compilation]** (複数ファイルのコンパイル) オプションは使用できません。

## コンパイル

### 1 つまたは複数のファイルをコンパイルするには：

- 1 [Workspace] (ワークスペース) ウィンドウでファイルを選択するか、コンパイルするファイルが表示されているエディタウィンドウをクリックします。



- 2 ツールバーの [Compile] (コンパイル) をクリックします。

または、[Project] (プロジェクト) メニューで使用可能な以下のコマンドのいずれかを使用します：

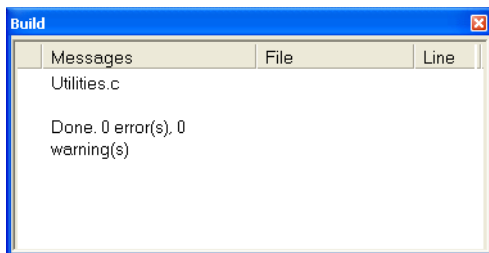


**[Make]** (メイク) — 最後のビルド以降に変更されたファイルだけをコンパイル、アセンブル、リンクして、現在のビルド構成を最新の状態に更新。

**[Rebuild All]** (すべてを再ビルド) — アクティブなプロジェクト構成のすべてのファイルをビルドして再度リンク。

**[Batch build]** (バッチビルド) — 指定したバッチビルド構成を作成し、指定したバッチをビルドするためのダイアログボックスを表示します。

- 3 ソースコードのエラーが生成された場合は、[Build] (ビルド) ウィンドウでエラーメッセージをダブルクリックすることで、適切なソースファイルの正しい位置に切り替えます。



- 4 1 つまたは複数のファイルをコンパイルあるいはアセンブルした後は、IDE によってプロジェクトディレクトリに 2 つの新しいディレクトリとファイルが作成されています。ビルド構成の名前が **Debug** だとすると、これらのディレクトリを含む同じ名前のディレクトリが作成されています。

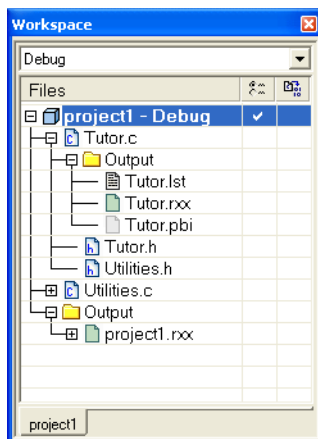
- List — リストファイルの保存先ディレクトリで、拡張子は 1st、map、log。
- Obj — コンパイラとアセンブラが生成したオブジェクトファイルの保存先ディレクトリ。これらのファイルはリンカへの入力として使用され、拡張子は XLINK と併用する製品の場合は rnn (nn は製品パッケージによって異なります)、ILINK と併用する製品では o です。
- Exe — 実行可能ファイルのディレクトリ。C-SPY への入力として使用され、拡張子は XLINK と使用する製品の場合は dnn (nn は製品パッケージによって異なります)、ILINK と共に使用する製品では out です。このディレクトリは、オブジェクトファイルをリンクするまでは空であることに注意してください。

**[Workspace]** (ワークスペース) ウィンドウで結果を参照するには :

コンパイルの後で、**[Workspace]** (ワークスペース) ウィンドウの **[+]** アイコンをクリックして、表示を拡大します。

**[Workspac]** (ワークスペース) ウィンドウには IDE によって作成された出力フォルダのアイコンが表示されています。このフォルダには、生成されたすべての出力ファイルが保存されます。同様に、インクルードされているすべてのヘッダファイルも表示され、ファイル間の依存関係を示しています。

生成されたファイルのファイル名の拡張子は、製品パッケージによって異なります。



## リンク

- 1 **[Workspace]** (ワークスペース) ウィンドウでプロジェクトを選択し、右クリックしてコンテキストメニューから **[Options]** (オプション) を選択します。続いて、**[Category]** (カテゴリ) リストで **[Linker]** (リンカ) を選択して、リンカのオプションページを表示します。
- 2 設定が終わったら、**[Project]>[Make]** (プロジェクト > 作成) を選択します。進捗状況は、**[Build]** (ビルド) メッセージウィンドウに表示されます。リンクの結果は、デバッグ情報を含む出力ファイルになります (デバッグ情報付きでビルドした場合)。

リンカオプションの設定時には、出力オプションや出力形式、リンカ設定ファイル、マップファイルおよびログファイルの選択に注意してください。

## 出力形式

**XLINK リンカ**は、数多くの形式を生成できます。出力形式は、目的に合わせて選択することが重要です。デバッガに出力をロードする場合は、デバッグ情報付きで出力する必要があります。最終のアプリケーションプロジェクトでは、別の方法として、出力を **PROM** プログラマにロードすることができます。この場合は、**Intel-hex**、**Motorola S-records** など、プログラマでサポートされている出力形式を使用する必要があります。

**ILINK リンカ**では、ELF 形式（デバッグ情報用の **DWARF** を含む）の出力ファイルを作成します。ELF 形式ではなく、**Motorola** または **Intel-standard** 形式のファイルが必要な場合（ファイルを **PROM** メモリにロードする場合など）、ファイルを変換する必要があります。**[Options]**（オプション）ダイアログボックスで **[Converter]**（コンバータ）カテゴリを選択し、適切なオプションを設定してください。

## リンカ設定ファイル

プログラムコードとデータは、リンカ設定ファイル（ファイル名の拡張子は **ILINK** では **icf**、**XLINK** では **xc1**）で指定された設定に基づいて、メモリに配置されます。セクションをメモリに配置する方法の構文について理解していることが重要です。

製品パッケージによっては、ターゲット¥config ディレクトリにリンカ設定ファイルのテンプレート、または一部あるいはサポートされているすべてのデバイス用の定義済みリンカ設定ファイルが含まれています。製品に備わっている **C-SPY** シミュレータに付属のファイルやテンプレートを使用できますが、それらをターゲットシステムに使用する場合は、実際のハードウェアのメモリレイアウトに合わせて調整する必要があります。

リンカ設定ファイルを詳細に確認するには、**IAR Embedded Workbench** エディタのようなテキストエディタか、ファイルの内容を印刷して定義がハードウェアメモリのレイアウト要件に合っているかどうか確認してください。

## リンカマップとログファイル

**XLINK** と **ILINK** はどちらも、詳細なリストを生成できます。

- **XLINK** はマップファイルを生成でき、これにはセグメントマップやシンボルリスト、モジュールサマリなどが任意で含まれます。
- **ILINK** は、一般的に配置のサマリを含むマップファイルを生成できます。また、**ILINK** はログファイルを生成できます。このファイルには、初期化やモジュールの選択、セクションの選択などについて、リンカが決定した事項が記録されます。

一般的に、この情報は次のことを詳しく調べるときに役立ちます。

- セグメント/セクションおよびコードがどのようにメモリに配置されたか
- どのソースファイルが実際に最終のイメージに含まれたか
- どのシンボルが実際にインクルードされたか、およびその値
- 個々の関数がメモリのどこに配置されたか



# デバッグ

C-SPY デバッグの機能の一部を紹介することにより、本章ではそれらの機能と使用方法について説明します。

- デバッグの設定
- デバッグの起動
- アプリケーションの実行
- 変数の検証
- メモリとレジスタのモニタ
- ブレークポイントの使用
- ターミナル I/O の表示
- アプリケーションのランタイムでの動作の解析

インストールした製品パッケージによって、C-SPY が含まれている場合とそうでない場合があります。

使用するハードウェアによっては、ここで紹介しない追加機能が使用する C-SPY ドライバで利用可能なことがあります。一般的には、これは異なるタイプのウォッチポイントの設定や追加のブレークポイントタイプ、さまざまなトリガシステム、より複雑なトレースシステムなどに該当します。

## デバッグの設定

- 1 C-SPY を起動する前に、**[Project]>[Options]>[Debugger]>[Setup]** (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。
- 2 **[Category]** (カテゴリ) リストで、適切な C-SPY ドライバを選択し、設定を確認します。
- 3 C-SPY の設定が終わったら、**[OK]** をクリックします。
- 4 **[Tools]>[Options]>[Debugger]** (ツール>オプション>デバッガ) を選択して以下を設定します。

- デバッガの動作
- デバッガによるスタック使用率のトラッキング

## C-SPY 起動前のハードウェア設定

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、デバッグする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。次に例を示します。

- 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。たとえば、外部の SDRAM を有効にするマクロは以下ようになります：

```
/* 使用するマクロ関数 */
enableExternalSDRAM()
{
    __message "Enabling external SDRAM\r\n";
    __writeMemory32 ( /* ここにコードを配置 */ );
    /* 必要に応じてここにコードを追加 */
}

/* セットアップマクロが実行時間を決定 */
execUserPreload()
{
    enableExternalSDRAM();
}
```

組み込みの `execUserPreload` セットアップマクロ関数が使用され、マクロ関数は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の間に直接実行されます。

- 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、**[Project]>[Options]>[Debugger]** (プロジェクト > オプション > デバッガ) を選択して、**[Setup]** (設定) タブをクリックします。オプション **[Use Setup file]** (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、起動マクロが 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。

## デバッガの起動

デバッガを起動するには、次のどちらかの手順に従います：



**ダウンロードしてデバッグ**は C-SPY を起動して、現在のプロジェクトをターゲットシステムにロードします。

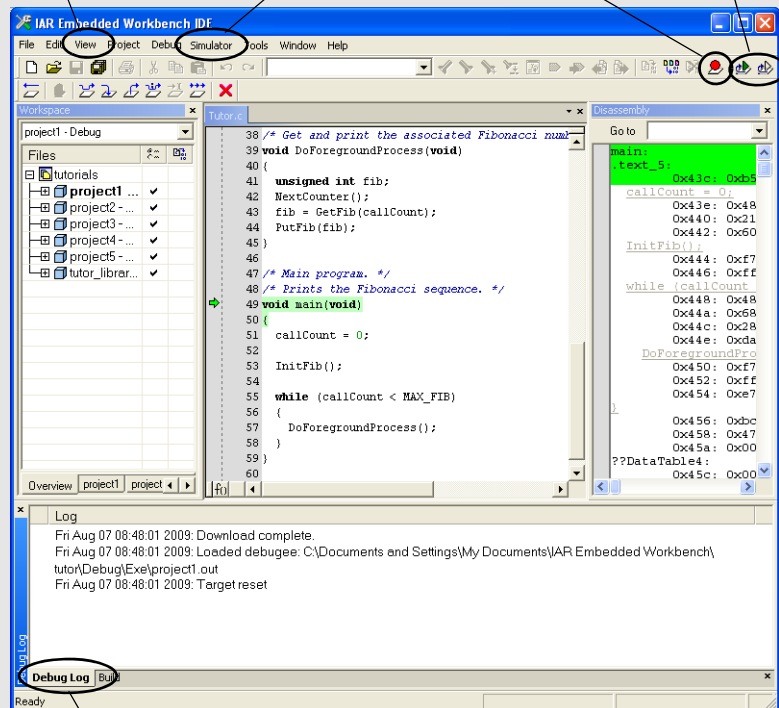


**ダウンロードなしにデバッグ**は、現在のプロジェクトをターゲットシステムに再ロードせずに C-SPY を起動します。コードイメージがすでにターゲットにあるために、ロードが必要ないときを想定しています。

複数のデバッグファイル（イメージ）をターゲットシステムにロードできます。IDE で追加のデバッグファイルをロードするには、**[Project]>[Options]>[Debugger]>[Images]**（プロジェクト>オプション>デバッガ>イメージ）を選択します。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。たとえば、アプリケーション（1つのイメージ）がブートローダ（別のイメージ）によって起動されるとします。アプリケーションイメージとブートローダは別のプロジェクトを使用してビルドされ、別の出力ファイルを生成します。

C-SPY が起動して、アプリケーションがロードされます。

[View] (表示) メニューで ドライバ固有のメニュー  
C-SPY ウィンドウが使用可能に (ドライバに関連する名前) ブレークポイントの切替え



[Debug Log] (デバッグログ) ウィンドウ  
デバッグ出力を表示



C-SPY は、ウィンドウの内容を更新するためにターゲットシステムから読み込む必要があります ([Memory] (メモリ) や [Trace] (トレース) ウィンドウなど、更新の必要があるウィンドウの場合)。これによって、デバッグ中に反応時間が影響を受けます。同時に複数のウィンドウを開いていて、反応時間が長すぎる場合 (特にハードウェア上でアプリケーションを実行中)、ウィンドウを1つか2つ閉じて反応時間を短縮します。

**C-SPY を終了するには：**



[Debug] (デバッグ) ツールバーの [Stop Debugging] (デバッグ停止) ボタンをクリックします。

## アプリケーションの実行

以下のような実行コマンドは、**[Debug]** (デバッグ) メニューと **[Debug]** (デバッグ) ツールバーにあります。



**[Step Over]** (ステップオーバー) は、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。



**[Step Into]** (ステップイン) は、C/C++ 関数やアセンブラサブルーチンに入って、次の文 / 命令を実行します。



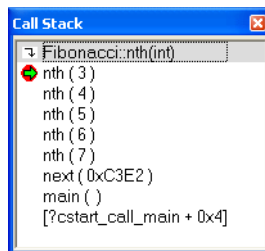
**[Next Statement]** (次のステートメント) は、個々の関数呼出しで停止せずに次の C/C++ 文まで直接実行します。

また、**[Go]** (実行)、**[Break]** (ブレーク)、**[Reset]** (リセット)、**[Run to Cursor]** (カーソルまで実行)、**[Autostep]** (自動ステップ) などのコマンドは、メニューおよびツールバーにあります。

C-SPY はステップポイントの機能によってステートメント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。複雑なステートメントの一部である個別の関数呼出し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。

関数呼出し検証するには：

- 1 **[View]>[Call Stack]** (表示 > 呼出しスタック) を選択して、**[Call Stack]** (呼出しスタック) ウィンドウを開きます。C/C++ 関数呼出しスタックが、現在の関数とともに上部に表示されます。いずれかの関数をダブルクリックすると、IDE で影響を受けるすべてのウィンドウの内容がが更新されて、特定の呼出しフレームのステートが表示されます。



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、呼出しチェーン内で問題が発生した関数を特定する場合

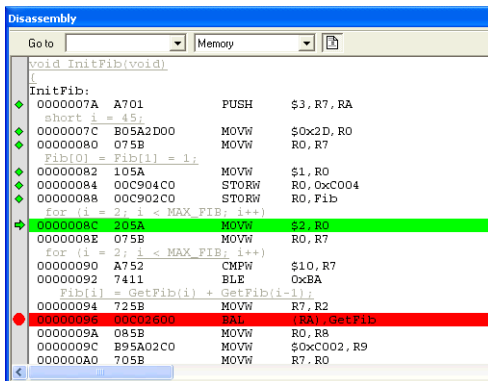
逆アセンブリモードでデバッグするには：

### [View]>[Disassembly]

(表示>逆アセンブリ)を選択して [Disassembly] (逆アセンブリ) ウィンドウを開きます (まだ開いていない場合)。現在の C 文に対応するアセンブラコードが表示されます。

逆アセンブリモードでは、一度に1つのアセンブラ命令単位でアプリケーション

を実行できます。対照的に、C/C++ モードでは一度に1つのステートメントまたは関数単位でアプリケーションを実行します。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。



モードを切り替えるには：

マウスポインタを使用して、使用するモードに応じて、[Editor] (エディタ) ウィンドウか [Disassembly] (逆アセンブリ) ウィンドウをアクティブにします。

コードカバレッジ情報を表示するには：

[Disassembly] (逆アセンブリ) ウィンドウで右クリックして [Code Coverage]>[Enable] (コードカバレッジ>有効) を選択し、コンテキストメニューから [Code Coverage]>[Show] (コードカバレッジ>表示) を選択します。実行済みのコードは、緑色のひし形で示されます。63 ページのコードカバレッジも参照してください。

## 変数の検証

C-SPY では、ソースコードの変数や式をモニタして、アプリケーションを実行したときのそれらの値をトレースできます。いくつかの方法で変数を参照できます。

ツールチップウォッチは、変数の値やより複雑な式をエディタウィンドウで参照する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。

**[Locals] (ローカル) ウィンドウ**は、**[View] (表示)** メニューから表示でき、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自動的に表示されます。

**[Watch] (ウォッチ) ウィンドウ**は**[View] (表示)** メニューから表示します。これを使用して、選択した C-SPY 式および変数の値をモニタリングできます。

**[Live Watch] (ライブウォッチ) ウィンドウ**は**[View] (表示)** メニューから表示します。アプリケーション実行中に継続的に式の値がサンプリングされて表示されます。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。このウィンドウでは、プログラム実行中にターゲットシステムがメモリの読取りをサポートしている必要があります。

**[Statics] (静的変数) ウィンドウ**は**[View] (表示)** メニューから表示します。静的記憶寿命変数の値が自動的に表示されます。さらに、表示するこうした変数を自由に選択できます。

**[Auto] (自動) ウィンドウ**は**[View] (表示)** メニューから表示します。現在の文の中、または近くにある変数と式の自動選択の内容が表示されます。

**[Quick Watch] (クイックウォッチ) ウィンドウ**は、変数または式の値をいつ評価したりモニタリングするかを、正確に制御できる素早い方法です。

**トレースデータの収集**は、はドライバ固有のメニューから表示します。イベントのシーケンス (通常は実行されたマシン命令) をターゲットシステムに収集できます。使用しているターゲットシステムに応じて、追加タイプのトレースデータが収集できます。たとえば、メモリのリード/ライトアクセス、C-SPY 式の値などを記録できます。65 ページのトレースも参照してください。

**注:** 最適化レベルとして **[None] (なし)** が使用されている場合、すべての非静的変数はそのスコープが継続している間は有効であり、したがって、そのような変数は完全にデバッグ可能です。それよりも高いレベルの最適化が使用されている場合は、変数は完全にデバッグできない可能性があります。

ウィンドウでは、式の追加、変更、削除、表示フォーマットの変更を行うことができます。コンテキストメニューはすべてのウィンドウで、操作コマンドとともに利用できます。ウィンドウ間でのドラッグアンドドロップは、該当する箇所ではサポートされません。

## 変数の値を調べるには：

- 1 たとえば、[Watch] (ウォッチ) ウィンドウを開くには、[View]>[Watch] (表示>ウォッチ) を選択します。
- 2 変数を選択するには、この手順に従います：
  - [Watch] (ウォッチ) ウィンドウで点線の長方形をクリックします。
  - 表示される入力フィールドに変数名を入力して、Enter キーを押します。
  - エディタウィンドウから [Watch] (ウォッチ) ウィンドウに変数をドラッグしても、ウォッチポイントを設定できます。

この例では、[Watch] (ウォッチ) ウィンドウに変数の現在の値 `i` と配列 `Fib` が表示されます。`Fib` 配列を展開すると、さらに詳細にモニタできます。

Expression	Value	Location	Type
i	5	0x7	short
Fib	<array>	Memory:0xC002	unsigned int[10]
[0]	1	Memory:0xC002	unsigned int
[1]	1	Memory:0xC004	unsigned int
[2]	2	Memory:0xC006	unsigned int
[3]	3	Memory:0xC008	unsigned int
[4]	5	Memory:0xC00A	unsigned int
[5]	0	Memory:0xC00C	unsigned int
[6]	0	Memory:0xC00E	unsigned int
[7]	0	Memory:0xC010	unsigned int
[8]	0	Memory:0xC012	unsigned int
[9]	0	Memory:0xC014	unsigned int

- 3 [Watch] (ウォッチ) ウィンドウから変数を削除するには、削除する変数を選択して、[Delete] (削除) キーをクリックします。

## メモリとレジスタのモニタ

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 [View] (表示) メニューから表示できます。

**[Memory] (メモリ) ウィンドウ**は、指定したメモリ領域 (C-SPY ではメモリゾーン) の最新の状態を表示します。内容は編集できます。データカバレッジ (製品パッケージによって異なります)、とアプリケーションの実行方法を示すために色が使用されます。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イ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。



**[Symbolic memory] (シンボルメモリ) ウィンドウ**は、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、制限を超えたバッファ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。

**[Stack] (スタック) ウィンドウ**は、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。詳細については、64 ページの *スタックの使用量* を参照してください。

**[Register] (レジスタ) ウィンドウ**は、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。

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。

## ブレイクポイントの使用

使用する C-SPY ドライバに応じて、さまざまな種類のブレイクポイントを設定できます。

**[Code] (コード) ブレイクポイント**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。

**[Log] (ログ) ブレイクポイント**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。

**[Trace Start] (トレース開始) および [Trace Stop] (トレース停止) ブレイクポイント**は、トレースデータの収集を開始および停止します。2つの実行ポイント間で命令を分析する便利な方法です。65 ページの *トレース* も参照してください。

**[Data] (データ) ブレイクポイント**は、リードまたはライトのメモリアクセスのためにトリガされます。データブレイクポイントは通常、データがいつどのように変化するかを調べるときに使用します。

これらのブレイクポイントに加えて、使用するデバッグシステムによっては、C-SPY ドライバはさらに複雑または異なる種類の他のブレイクポイントあるいはトリガをサポートすることがあります。

## ブレークポイントを設定するには：

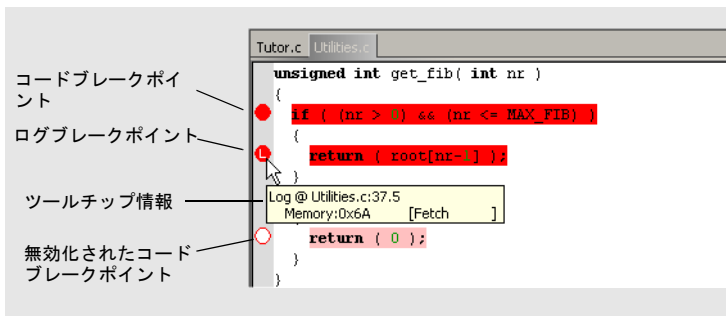


挿入ポイントを左側の余白、または文の中か近くに配置して、ダブルクリックしてコードブレークポイントを切り替えます。

または、**[Breakpoints]** (ブレークポイント) ダイアログボックスは、**[Editor]** (エディタ) ウィンドウ、**[Breakpoints]** (ブレークポイント) ウィンドウ、**[Disassembly]** (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。このダイアログボックスでは、異なるタイプのブレークポイントを細かく設定して編集することができます。

**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。

ブレークポイントは、**[Editor]** (エディタ) ウィンドウの左余白にアイコンで示されます。



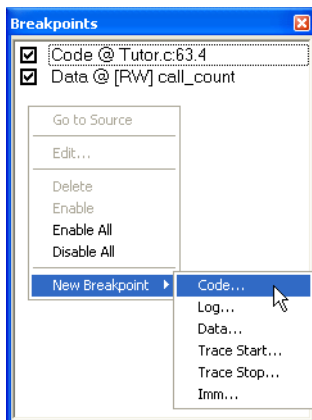
ブレークポイントアイコンが表示されない場合は、**[IDE Options]>[Editor]** (IDE オプション>エディタ) ダイアログボックスで **[Show bookmarks]** (ブックマークの表示) オプションが選択されていることを確認します。



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、**[Breakpoint Usage]** (ブレークポイントの使用) ダイアログボックスでも表示されます。

すべての定義済みブレークポイントを表示するには：

[View]>[Breakpoints] (表示>ブレークポイント) を選択して、[Breakpoints] (ブレークポイント) ウィンドウを開きます。このウィンドウにはすべてのブレークポイントがリストされます。ここでは、ブレークポイントのモニターや有効/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、削除も行うことができます。



ブレークポイントの使用状況調べるには：

- 1 [Breakpoints Usage] (ブレークポイントの使用) ウィンドウ (C-SPY ドライバ固有のメニューから表示) を開き、定義済みのものや C-SPY で内部的に使用されるものも含めて、すべてのブレークポイントを低レベルで表示します。

通常は、ターゲットハードウェアには限られた数のハードウェアブレークポイント (C-SPY でブレークポイントの設定に使用) があり、時には 1～2 しかないこともあります。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、デバッガが実行中に強制的に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。



限られた数のハードウェアブレークポイントしか持たないハードウェアデバッガシステムでは、[Breakpoint Usage] (ブレークポイントの使用) ウィンドウを使用して以下のことを実行します。

- すべてのブレークポイントの使用状況を確認
- 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

ブレークポイントまで実行するには：



- 1 ツールバーの [Go] (実行) ボタンをクリックします。

次に設定されたブレークポイントまでアプリケーションが実行されます。[Debug Log] (デバッグログ) ウィンドウには、ブレークポイントのトリガに関する情報が含まれます。

- 2 ブレークポイントを選択して右クリックし、コンテキストメニューから [Toggle Breakpoint] (ブレークポイントの切替え) (xxx) を選択してブレークポイントを削除します。

## ターミナル I/O の表示

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、ハードウェアを使用しないでデバッグする必要があります。C-SPY では、[Terminal I/O] (ターミナル I/O) ウィンドウを使用して、stdin と stdout をシミュレーションできます。

[Terminal I/O] ウィンドウを使用するには：

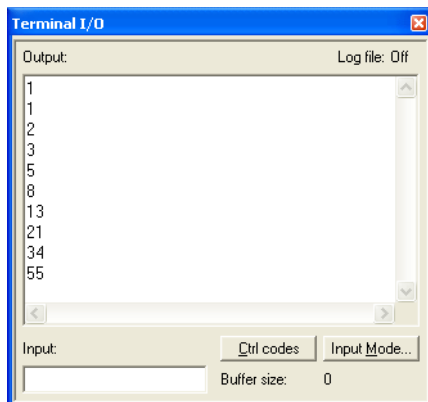
- 1 以下のオプションを使用してアプリケーションをビルドします：

カテゴリ	設定
[Linker]>[Config] (リンカ > 設定) (XLINK の場合)	With I/O emulation modules (I/O エミュレーションモジュール付き)
[General Options]>[Library Configuration] Library low-level interface implementation (一般オプション > ライブラリ設定) (ILINK の場合)	(低レベルインタフェースのライブラリ実装)

これは、stdin と stdout を [Terminal I/O] ウィンドウに接続する低レベルルーチンが、リンクされていることを意味します。

- 2 アプリケーションをビルドして C-SPY を起動します。

- 3 [View]>[Terminal I/O] ([表示]>[ターミナル I/O]) を選択して [Terminal I/O] ウィンドウを開きます。このウィンドウには、I/O 処理からの出力が表示されます。



## アプリケーションのランタイムでの動作の解析

C-SPY には、ボトルネックを発見して、アプリケーションのすべての部分がテスト済みであることを検証するために、アプリケーションのランタイム動作の解析に使用可能なさまざまな機能が用意されています。

- プロファイリング
- コードカバレッジ
- スタックの使用量
- トレース

## プロファイリング

2 種類のプロファイリングのどちらかを選択できます：

**関数プロファイリング**では、実行時間のほとんどが費やされる関数が見つけやすくなります。これらの関数は、コードを最適化するとき重点を置くべき部分です。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、最も効率的なアドレッシングモードを使用するメモリ内に関数を移動することもできます。

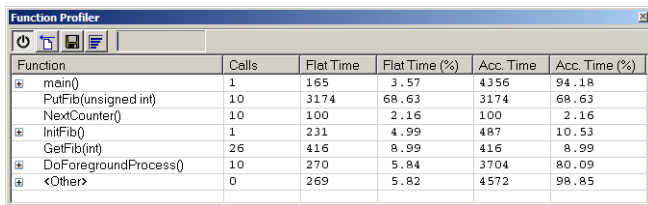
**命令プロファイリング**情報、すなわち各命令が何回実行されたかが [Disassembly] (逆アセンブリ) ウィンドウに表示され、非常に詳細なレベルでコード (特にアセンブラのソースコード) を微調整できます。

## プロファイリングを使用するには：

- 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	[Output]>[Generate debug information] (出力 > デバッグ情報の生成)
Linker (XLINK の場合)	[Format]>[Debug information for C-SPY] (形式 > C-SPY のデバッグ情報)
Linker (ILINK の場合)	[Output]>[Include debug information in output] (出力 > 出力ファイルにデバッグ情報を含める)

- 2 アプリケーションをビルドして C-SPY を起動します。
- 3 プロファイリングを使用する前に、設定する必要があります。設定は、C-SPY ドライバおよびターゲットシステムによって異なります。
- 4 [Function Profiler] (関数プロファイラ) ウィンドウを開くには、ドライバ固有のメニューから **[Profiling]** (プロファイリング) を選択します。
- 5 **[Enable]** (有効) ボタンをクリックして、プロファイラを有効にします。
- 6 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。
- 7 プロファイリング情報が [Function Profiler] (関数プロファイラ) ウィンドウに表示されます。



Function	Calls	Flat Time	Flat Time (%)	Acc. Time	Acc. Time (%)
main()	1	165	3.57	4356	94.18
PutFib(unsigned int)	10	3174	68.63	3174	68.63
NextCounter()	10	100	2.16	100	2.16
InitFib()	1	231	4.99	487	10.53
GetFib(int)	26	416	8.99	416	8.99
DoForegroundProcess()	10	270	5.84	3704	80.09
<Other>	0	269	5.82	4572	98.85

情報をソートするには、対象の列見出しをクリックします。

- 8 新しいサンプリングを開始する前に、**[Clear]** (クリア) ボタンをクリックします。
- 9 **[Graph]** (グラフ) ボタンをクリックすると、パーセント値を表す列の表示方法を数値や棒グラフに切り替えることができます。

## コードカバレッジ

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。

**注：**ハードウェア上でデバッグする際に、コードカバレッジ制限がある場合があります。特に、サイクルカウンタの統計が使用できない場合があります。

コードカバレッジを使用するには：

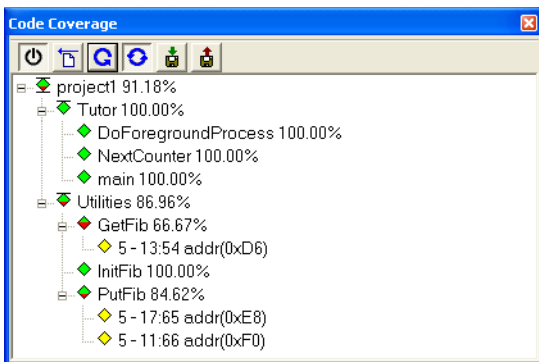
- 1 以下のオプションを使用してアプリケーションをビルドします：

カテゴリ	設定
C/C++ コンパイラ	[Output]>[Generate debug information] (出力 > デバッグ情報の生成)
Linker (XLINK の場合)	[Format]>[Debug information for C-SPY] (形式 > C-SPY のデバッグ情報)
Linker (ILINK の場合)	[Output]>[Include debug information in output] (出力 > 出力ファイルにデバッグ情報を含める)
Debugger	[Plugins]>[Code Coverage] (プラグイン > コードカバレッジ)

- 2 アプリケーションをビルドして C-SPY を起動します。

- 3 **[View]>[Code**

**Coverage]** (表示 > コードカバレッジ) を選択して、**[Code Coverage]** (コードカバレッジ) ウィンドウを開きます。



- 4 **[Activate]** (有効化) ボタンをクリックして、コードカバレッジアナライザを有効にします。

- 5 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、**[Refresh]** (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。

[Code Coverage] ウィンドウに、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。つまり、解析を開始してからコードのどの部分が少なくとも1回は実行されたかです。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。レポートには、すべてのモジュールと関数についての情報が表示されます。アプリケーション停止時点で実行済みのすべてのステップポイントの割合がレポートされ、実行されていないすべてのステップポイントが一覧表示されます。

## 6 カバレッジは、無効にするまで続行されます。

**注：**コードカバレッジは、[Disassembly] ウィンドウにも表示できます。実行されるコードは、緑色のひし形で示されます。

## スタックの使用量

[Stack] (スタック) ウィンドウは、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、スタックオーバーフローを検出してワーニングすることもできます。



[Stack] ウィンドウにスタックの内容が表示されます。これは、以下の場合に役立ちます：

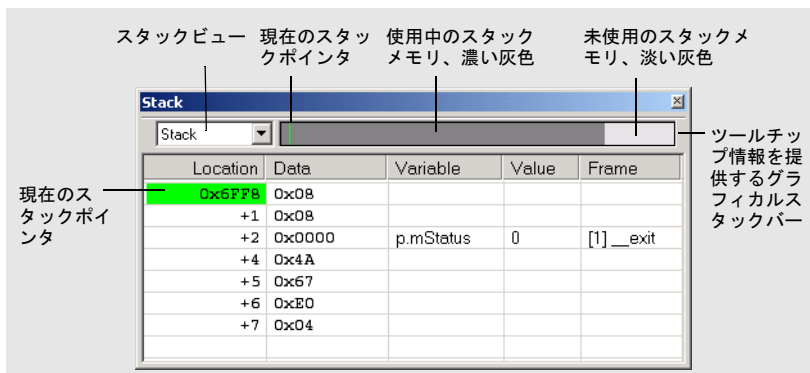
- Cモジュールからアセンブラモジュールを呼び出すか、その逆のときに、スタック使用量を調べる場合
- 適切なエレメントがスタック上に配置されているかどうかを調べる場合
- スタックが正しく復元されているかどうかを調べる場合

スタックの使用をトラッキングするには：

- 1 [Project]>[Options]>[Debugger]>[Plugins] (プロジェクト>オプション>デバッガ>プラグイン) を選択した後、プラグインのリストから [Stack] を選択します。
- 2 [Tools]>[Options]>[Stack] (ツール>オプション>スタック) を選択して、スタックのトラッキングを設定します。スタックポイントがいつ有効になるかを指定しなければならない場合があることに注意してください。
- 3 アプリケーションをビルドして C-SPY を起動します。



#### 4 [View]>[Stack] (表示>スタック) を選択します。



[Stack] ウィンドウの複数のインスタンスを表示し、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。



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。

#### スタックオーバフローを検出するには：

[Tools]>[Options]>[Stack] (ツール>オプション>スタック) を選択して、オプションの [Enable stack checks] (スタックチェックの有効化) を選択します。

これは、アプリケーションが実行を停止したときに、C-SPY がスタックオーバフローのワーニングを出力できることを意味します。ワーニングは、スタック使用量がユーザが指定するしきい値を超えたときか、スタックポインタがスタックメモリ範囲以外のエリアを指したときに出力されます。


#### トレース

トレースデータを収集することで、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分析したり、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。

トレースとは、収集されたマシン命令のシーケンスを記録したものです。使用する C-SPY ドライバによって、使用可能なトレースデータが決まります。

- C-SPY シミュレータは、[Trace Expressions] (トレース式) ウィンドウで選択した C-SPY 式の値を記録します。[Trace] (トレース) ウィンドウにはすべての命令が表示されますが、[Function Trace] (関数トレース) ウィンドウには、関数呼出しと関数からの呼び出しに対応するトレースデータだけが表示されます。
- ハードウェアデバッグシステムシステムの C-SPY ドライバは、使用するハードウェアがこれをサポートする場合に、トレースデータを収集できます。たとえば、トレース収集のための専用の通信チャンネルや専用のトレースバッファがある場合などです。この場合、[Trace] (トレース) ウィンドウには収集されたデータが反映されます。

### トレースデータを収集するには：

- 1 シミュレータでトレースデータを収集するために、特定のビルド設定は不要です。ハードウェアデバッグシステムを使用する場合、トレースデータを先に設定する必要があります。詳しくは、ドライバのマニュアルを参照してください。
- 2 アプリケーションをビルドして C-SPY を起動します。
- 3  ドライバ固有のメニューから [Trace] (トレース) を選択して [Trace] (トレース) ウィンドウを開き、[Activate] (有効化) ボタンをクリックし、トレースデータの収集を有効にします。
- 4 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、[Trace] ウィンドウにトレースデータが表示されます。

### ブレークポイントを使用してトレースデータの収集を開始するには：

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。エディタまたは [Disassembly] (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 [Trace Start] (トレース開始) または [Trace Stop] (トレース停止) ブレークポイントを切り替えます。C-SPY シミュレータで、C-SPY システムマクロ `__setTraceStartBreak` と `__setTraceStopBreak` を使用することも可能です。

# 導入ガイド

IAR Embedded Workbench<sup>®</sup>

[www.iar.com/jp](http://www.iar.com/jp)

GSEW-3-JP