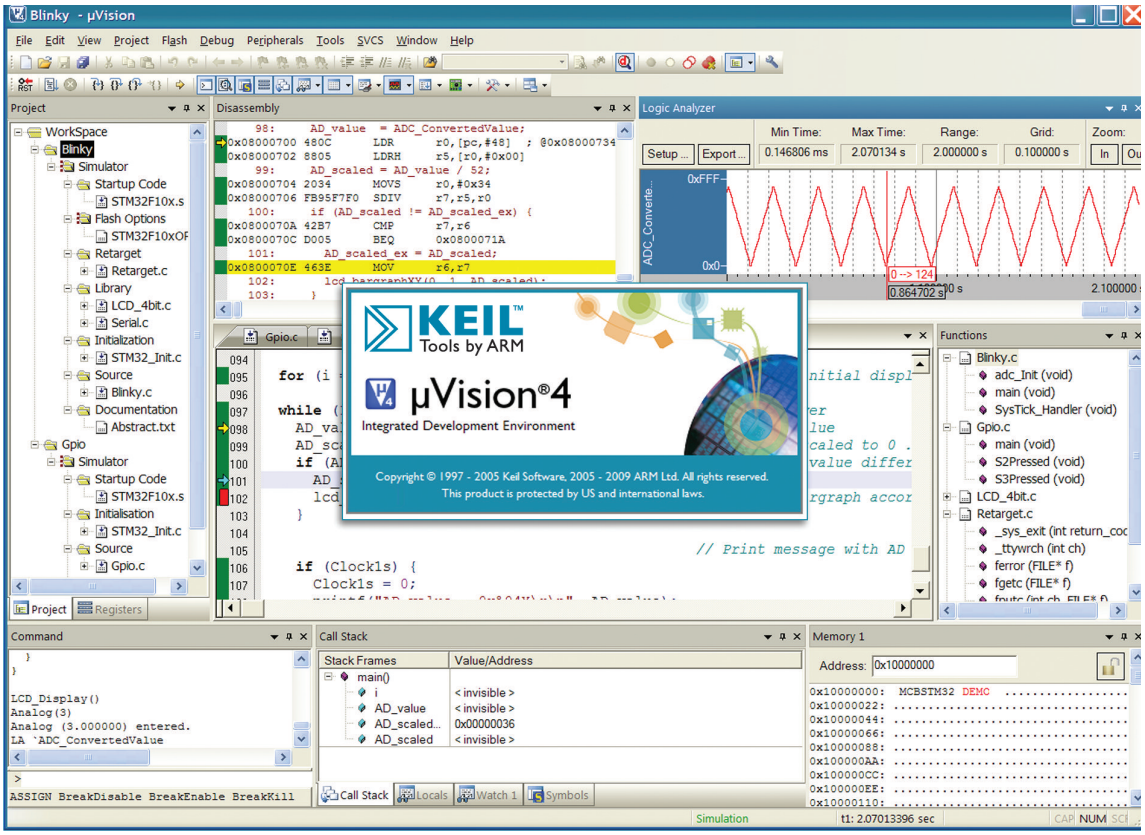
KEIL™
Tools by ARM

スターターガイド

µVision®4 を使用したアプリケーションの作成



8 ビット、16 ビット、32 ビット マイクロコントローラ

注

本書は、Microsoft Windows およびハードウェア、ARM7、ARM9、Cortex-Mx、C166、XE166、XC2000、または 8051 マイクロコントローラの命令セットに精通している読者を対象に作成しています。

本書では、記述の正確を期すため、また、参照されている個人、法人および商標の適切なクレジット表示をするため、あらゆる努力がなされています。

序文

本書は、Cortex-Mx、ARM7、ARM9、C166、XE166、XC2000、および 8051 マイクロコントローラの Keil 開発ツールの入門書です。ここでは、µVision 統合開発環境、シミュレータ、およびデバッガを紹介し、Keil 組み込み開発ツールの数多くの機能と性能について段階的に説明します。

本書の対象読者

本書は、習得中、初心者、上級および経験豊富な開発者すべてに有用です。

これまでに広範囲において µVision を使用した経験があり、µVision IDE の動作方法やデバッガ、シミュレータ、およびターゲットハードウェアでの作業方法に関する知識がある場合は、経験豊富な開発者または上級開発者と見なされます。さらに、マイクロコントローラを深く理解していることが望まれます。このようなエンジニアは、µVision に導入された拡張機能を熟知し、最新機能を理解することをお勧めします。

µVision での作業経験がない開発者は、習得中または初心者と見なされます。このような開発者は、最初に µVision IDE に関連した章を読み、説明されているインタフェースおよびコンフィギュレーションオプションに慣れるためにサンプルを使用して作業することをお勧めします。シミュレータがもたらす広範な可能性を活用できます。その後、RTOS とマイクロコントローラのアーキテクチャについて説明している章に進んで下さい。

ただし、上記はマイクロコントローラの使用方法について基本的な知識があり、適切なマイクロコントローラのいくつかの命令または命令セットに精通していることを前提としています。

本書は、各章が厳密に関連し合っていないため、個別に学習することができます。

章の概要

「**1. はじめに**」の章では、製品のインスールとライセンス取得の概要を説明し、Keil 開発ツールについてのサポートを受ける方法を示しています。

「**2. マイクロコントローラアーキテクチャ**」の章では、Keil 開発ツールがサポートするさまざまなマイクロコントローラアーキテクチャについて説明し、お客様のアプリケーションに最適なマイクロコントローラを選択するお手伝いをします。

「**3. 開発ツール**」の章では、µVision IDE とデバッガ、アセンブラ、コンパイラ、リンカ、その他の開発ツールの主要な機能について説明します。

「**4. RTX RTOS カーネル**」の章では、リアルタイムオペレーティングシステム（RTOS）を使用する利点と Keil RTX カーネルで使用可能な機能について説明します。

「**5. µVision の使用**」の章では、µVision ユーザインタフェースの特定の機能とその操作方法について説明します。

「**6. 組み込みプログラムの作成**」の章では、プロジェクトの作成、ソースファイルの編集、コンパイル、構文エラーの修正、実行可能コード生成の方法について説明します。

「**7. デバッグ**」の章では、組み込み後のプログラムをテストおよび検証するための µVision シミュレータとターゲットデバッガの使用方法について説明します。

「**8. ターゲットハードウェアの使用**」の章では、サードパーティ製 Flash プログラミングユーティリティとターゲットドライバを設定して使用する方法を示します。

「**9. サンプルプログラム**」の章では、4 つのサンプルプログラムについて説明し、それらのサンプルを使用して µVision の関連機能を示します。

本書における表記に関する定義

例	説明
README.TXT **（太字の大文字）**[1]	太字の大文字テキストは、実行可能プログラム、データファイル、ソースファイル、環境変数、コマンドプロンプトに入力できるコマンドの名前を強調するために使用されています。このテキストは通常、表記されたとおりに入力する必要があるコマンドを示しています。例： **ARMCC.EXE　　DIR　　LX51.EXE**
Courier （クーリエフォント）	この書体のテキストは、画面に表示する情報やプリンタで印刷する情報を示すために使用されます。 この書体は、コマンドライン項目について説明または記述しているテキスト内でも使用されています。
Variables （イタリック）	イタリック体のテキストは、ユーザが入力する必要がある情報を示します。例えば、構文文字列の中の projectfile は実際のプロジェクトファイル名を入力する必要があることを意味しています。 テキスト内の言葉を強調するためにイタリック体が使用されることもあります。
「...」を繰り返す表現	省略記号（…）は、繰り返される可能性のある項目を示すために使用されます。

省略記号
.
.
.
	縦に並んだ省略記号は、ソースコードのリストで使用され、プログラムのフラグメントが省略されていることを示します。以下に例を示します。
	void main (void) {
	.
	.
	.
	while (1);

«オプション項目»	二重括弧でくくられた部分は、コマンドラインおよび入力フィールドに任意で入力する項目を表します。以下に例を示します。 **C51 TEST.C PRINT** «ファイル名»
{ opt1 | opt2 }（縦線で区切られた中括弧内テキスト）	縦線で区切られた中括弧内のテキストは、選択項目を表します。中括弧が選択肢をすべて囲み、縦線がこの選択肢を分けています。リスト内の項目を 1 つ選択する必要があります。
Keys（キー）	sans serif 書式のテキストは、キーボード上の実際のキーを表しています。例えば、「Enter キーを押して続行します」となります。

[1] コマンドはすべて大文字で入力する必要はありません。

目次

第 1 章 はじめに

お客様の組み込みマイクロコントローラアプリケーションに Keil ソフトウェア開発ツールをお選びいただき、ありがとうございます。

本書『**スターターガイド**』では、µVision IDE、µVision デバッガおよび分析ツール、シミュレーション、デバッグおよびトレース機能について説明します。また、本書は µVision の基本的な動作および画面についての説明だけでなく、サポートされているマイクロコントローラアーキテクチャの種類やそれらの利点および要点についても包括的に説明し、適切なターゲットデバイスの選択に役立ちます。本書では、コードを記述するためのヒントも記載しています。他の**スターターガイド**と同様に、すべての局面や多くの使用可能なコンフィギュレーションオプションについて網羅しているわけではありません。µVision と付属のコンポーネントに慣れるためにサンプルを使用して作業することをお勧めします。

Keil 開発ツールは、ソフトウェア開発の専門家だけでなく、さまざまなレベルのプログラマがこのツールを使用して、サポートされている組み込みマイクロコントローラアーキテクチャを最大限に活用できるよう設計されています。

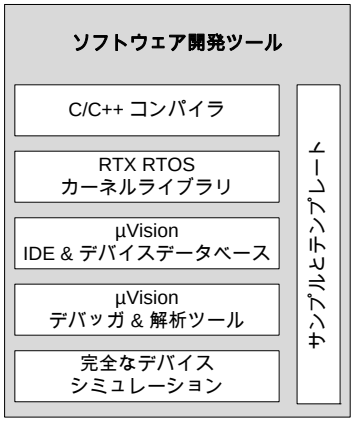
Keil が開発したツールは、よく使用されているマイクロコントローラのほとんどをサポートし、アーキテクチャに応じて、複数のパッケージおよびコンフィギュレーションで提供されます。

- **MDK-ARM**：マイクロコントローラ開発キット（ARM7、ARM9、および Cortex-Mx ベースのデバイス用）
- **PK166**：Keil プロフェッショナルデベロッパーズキット（C166、XE166、および XC2000 デバイス用）
- **DK251**：Keil 251 開発ツール（251 デバイス用）
- **PK51**：Keil 8051 開発ツール（クラシックおよび拡張 8051 デバイス用）

Keil では、ソフトウェアパッケージだけでなく、さまざまな評価ボード、USB-JTAG アダプタ、エミュレータ、サードパーティ製ツールなども提供しており、すべての製品を網羅しています。

以下の図は、Keil またはその他のベンダのツールと併用する µVision の一般的なコンポーネントブロックとコンポーネントの関連状況を示します。

ソフトウェア開発ツール

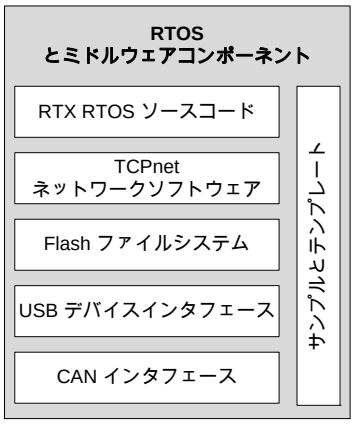


Keil の µVision IDE に基づいたすべてのソフトウェアと同様に、ツールセットが、組み込みアプリケーションを開発するための強力で使いやすく習得が容易な環境を提供しています。

C/C++ ソースファイルの作成、デバッグ、およびアセンブルに必要なコンポーネントが含まれており、マイクロコントローラと関連するペリフェラルのシミュレーションが組み込まれています。

RTX RTOS カーネルを使用すると、
複雑でタイムクリティカルなソフトウェアを実装できます。

RTOS とミドルウェアコンポーネント



これらのコンポーネントは、組み込みシステムの通信およびリアルタイムの課題を解決するために設計されています。リアルタイムカーネルを使用しなくても組み込みアプリケーションを実装することは可能ですが、実証済みのカーネルを使用することで、時間を節約し、開発サイクルを短縮できます。

また、このコンポーネントには、オペレーションシステムのソースコードファイルも含まれています。

ハードウェアデバッグアダプタ

µVision デバッガは、Keil とその他のベンダが提供しているいくつかのエミュレータを完全にサポートしています。Keil ULINK USB-JTAG ファミリのアダプタで、PC の USB ポートと対象のハードウェアを接続します。これにより、組み込みアプリケーションを実際のハードウェアでダウンロード、テスト、およびデバッグできます。

最終変更

他のハイテク製品と同様、最終変更が印刷済みのマニュアルに反映されない場合があります。ソフトウェアとマニュアルに対する最終変更と改善点は、製品に同梱されている**リリースノート**にリストされています。

ライセンス取得

各 Keil 製品は、ライセンスコードを使用して有効化する必要があります。このコードは、電子メールを介して製品登録の際に入手します。製品ライセンスには以下の 2 種類があります。

- **シングルユーザライセンス**は、Keil の全製品で利用が可能です。シングルユーザライセンスでは、1 ユーザが最大 2 台のコンピュータで製品を使用できます。インストールごとに、製品がインストールされるコンピュータ用の個別のライセンスコードが必要です。シングルユーザライセンスは、アンインストールして別のコンピュータに移すこともできます。
- **フローティングユーザライセンス**は、Keil の多数の製品で利用できます。フローティングユーザライセンスを使用すれば、同時に複数の開発者が複数のコンピュータで製品を使用できます。インストールごとに、製品がインストールされるコンピュータ用の個別のライセンスコードが必要です。

インストール

Keil 開発ツールを確実にインストールして、適切に機能させるには、ハードウェアおよびソフトウェアの最小要件が満たされていることを確認して下さい。インストールする前に、以下の条件を満たしていることを確認して下さい。

- Microsoft Windows XP または Windows Vista を実行する標準型 PC
- 1GB RAM、500 MB のハードディスク空き容量（推奨）
- 1024x768 以上のディスプレイ解像度、マウスまたはその他のポインティングデバイス
- CD-ROM ドライブ

Keil 製品は CD-ROM および**www.keil.com** からのダウンロードによって入手できます。
関連製品のアップデートは**www.keil.com/update** で定期的に入手できます。

Web サイトからのダウンロードによるインストール

1. **www.keil.com/demo** から製品をダウンロードします。
2. ダウンロードした実行可能ファイルを実行します。
3. **SETUP** プログラムで表示される指示に従います。

CD-ROM によるインストール

1. CD-ROM を CD-ROM ドライブに挿入します。CD-ROM ブラウザが自動的に開きます。ブラウザが開かない場合は、CD-ROM から **SETUP.EXE** を実行することもできます。
2. CD ブラウザメニューから［**Install Products & Updates**］を選択します。
3. **SETUP** プログラムで表示される指示に従います。

製品フォルダ構造

SETUP プログラムによって、開発ツールがサブフォルダにコピーされます。ベースフォルダは、デフォルトで **C:\KEIL** に配置されます。以下の表に、各マイクロコントローラアーキテクチャのインストールに対するデフォルトフォルダを示します。本書で使用されている例を調整して、実際のインストールディレクトリに合わせて下さい。

マイクロコントローラアーキテクチャ	フォルダ
MDK-ARM ツールセット	**C:\KEIL\ARM**
C166/XE166/XC2000 ツールセット	**C:\KEIL\C166**
8051 ツールセット	**C:\KEIL\C51**
C251 ツールセット	**C:\KEIL\C251**
µVision 共通ファイル	**C:\KEIL\UV4**

それぞれのツールセットには複数のサブフォルダがあります。

目次	サブフォルダ
実行可能プログラムファイル	**\BIN**
C インクルード/ヘッダファイル	**\INC**
オンラインヘルプファイルとリリースノート	**\HLP**
共通/汎用サンプルプログラム	**\EXAMPLES**
評価ボード用のサンプルプログラム	**\BOARDS**

サポートのリクエスト

Keil では、最高の組み込み開発ツール、マニュアル、サポートをお届けするよう心がけていますが、当社の製品に関してご提案、ご意見がある場合、ソフトウェアについて問題が生じた場合は、当社までご連絡下さい。その場合は、まず以下を試してみて下さい。

1. 実行しようとしている作業に関連のある本書のセクションをお読み下さい。
2. Keil の Web サイトのアップデートセクションをチェックして、お使いのソフトウェアとユーティリティのバージョンが最新のものかどうかを確認して下さい。
3. できるだけコードの行数を減らして、ソフトウェアの問題を切り離します。

それでも問題が解決しない場合は、当社のテクニカルサポートグループまで問題をお知らせ下さい。その際、お客様のライセンスコードと製品バージョン番号も併せてお知らせ下さい。［**Help**］→［**About**］メニューをご覧下さい。当社では、以下のサポートおよび情報チャネルも提供しています。すべて **www.keil.com/support**[1] でアクセスできます。

1. **Support Knowledgebase** は常時更新されており、サポート部門による最新の質問と回答が含まれています。
2. **Application Notes** では、割り込みやメモリの利用状況など、複雑な問題について確認できます。
3. オンラインの **Discussion Forum** を確認します。
4. **Contact Technical Support**（Web ベースの電子メール）でサポートをリクエストします。
5. 最後に、**support.intl@keil.com** または**support.us@keil.com** からサポート部門に直接問い合わせることができます。

[1] **www.keil.com/support** からテクニカルサポート、製品のアップデート、アプリケーションノート、およびサンプルプログラムをいつでも入手できます。

第 2 章 マイクロコントローラアーキテクチャ

Keil µVision 統合開発環境（µVision IDE）は、3 つの主なマイクロコントローラアーキテクチャをサポートし、さまざまなアプリケーションの開発に対応しています。

- **8 ビット（クラシックおよび拡張 8051）**デバイスは、リアルタイムパフォーマンス用に設計された効率的な割り込みシステムを備え、8 ビットアプリケーションの 65% 以上で使用されています。1000 を超えるバリアントを、アナログ I/O、タイマ/カウンタ、PWM、UART、I^2C、LIN、SPI、USB、CAN などのシリアルインタフェース、さらに低電力ワイヤレスアプリケーション用のオンチップ RF トランスミッタなどのペリフェラルと共に使用できます。一部のアーキテクチャ拡張機能は、最大で 16MB のメモリと豊富な 16/32 ビット命令セットを提供します。

  µVision IDE は、アプリケーション固有のペリフェラルをシングルチップに統合する IP コアに基づいたカスタムチップ設計などの最新の傾向にも対応しています。

- **16 ビット（Infineon C166**、**XE166**、および **XC2000**）デバイスは、最適なリアルタイムおよび割り込み性能用に調整され、マイクロコントローラコアと緊密に連動した豊富なオンチップペリフェラルのセットを提供します。これらのデバイスには、マイクロコントローラのオーバーヘッドが少ないか、まったくない高速データ収集用のペリフェラルイベントコントローラ（メモリ間 DMA に類似）が含まれています。

  このようなデバイスは、外部イベントにきわめて高速に対応する必要があるアプリケーションに最適です。

- **32 ビット（ARM7 および ARM9 ベース）**デバイスは、より強力な処理能力を必要とする複雑なアプリケーションをサポートしています。このようなコアでは、4GB のアドレス空間内で 32 ビット高速演算が可能です。RISC 命令セットは、コード密度を高めるため Thumb モードで拡張されています。

  ARM7 デバイスと ARM9 デバイスは、高速コンテキストスイッチ用に個別のスタック空間を備え、効率のよいマルチタスクオペレーティングシステムを実現します。ビットアドレス指定と専用のペリフェラルアドレス空間はサポートされていません。割り込み優先レベルは、割り込み要求（IRQ）と高速割り込み要求（FIQ）の 2 つのみ、使用できます。

- **32 ビット（Cortex-Mx ベース）**デバイスは、8 ビットデバイスと 16ビットデバイスのコスト上の利点と 32 ビットデバイスの柔軟性および性能上の利点が組み合わされた、超低消費電力デバイスです。このアーキテクチャによって、最先端の FPGA および SoC の実装が可能になります。Cortex-Mx[1] ベースのマイクロコントローラでは、Thumb2 命令セットの改良によって、4GB アドレス空間をサポートするとともに、ビットアドレス指定（ビットバンディング）と少なくとも 8 つの割り込み優先レベルを持つ複数の割り込みが可能です。

アーキテクチャの選択

組み込みアプリケーションに最適なデバイスを選択することは、複雑な作業です。Keil Device Database（**www.keil.com/dd**）では、適切なアーキテクチャを選択するための 3 つの方法をご用意しています。デバイスをアーキテクチャごとに検索する、マイクロコントローラの特性を指定して検索する、またはベンダごとに検索することができます。

以下のセクションでは、さまざまなアーキテクチャの利点について説明するとともに、組み込みアプリケーションに最適なマイクロコントローラを選択するためのガイドラインを提供します。

[1] Cortex-M0 デバイスは Thumb2 命令のサブセットを実装します。

8051 アーキテクチャの長所

- 高速 I/O 演算とデータ空間でのオンチップ RAM への高速アクセス
- 効率的で柔軟な割り込みシステム
- 低電力操作

8051 ベースのデバイスは、一般的に、高 I/O 処理能力を必要とする小規模および中規模アプリケーションで使用されています。柔軟なペリフェラルを備えた多くのデバイスを、最小のチップパッケージでも使用できます。

C166、XE166、およびXC2000 アーキテクチャの長所

- ペリフェラルイベントコントローラを介した超高速 I/O 演算
- 適切に調整されたペリフェラルを備えた高速割り込みシステム
- 効率的な演算および高速メモリアクセス

このようなデバイスは、高 I/O 処理能力を必要とする中規模から大規模アプリケーションで使用されます。このアーキテクチャは、従来のコントローラコードと DSP アルゴリズムの融合を伴う組み込みシステムのニーズに適しています。

ARM7 および ARM9 アーキテクチャの長所

- 大容量のリニアアドレス空間
- 16 ビット Thumb 命令セットによる高コード密度の実現
- ポインタアドレス指定を含むあらゆる C 整数データ型を効率よくサポート

ARM7 および ARM9 ベースのマイクロコントローラは、大量のメモリを必要とするアプリケーションや PC ベースのアルゴリズムを使用するアプリケーションで使用されます。

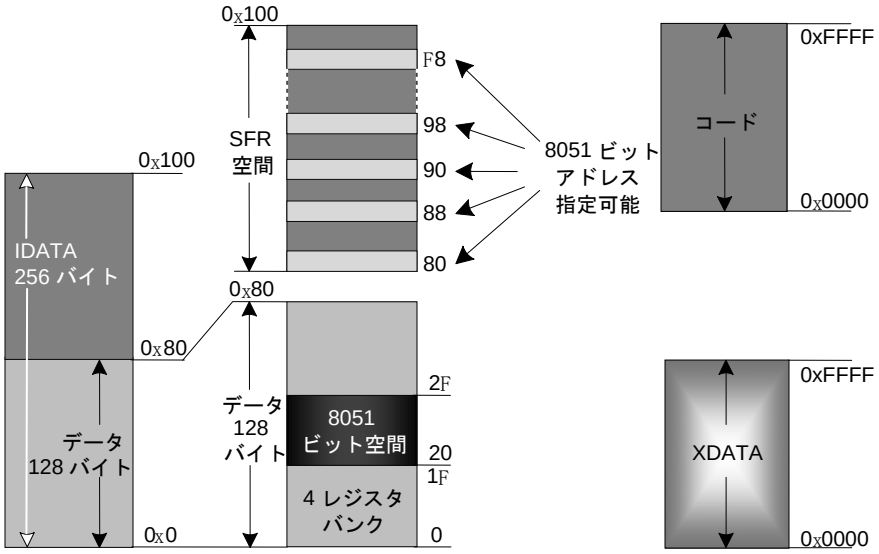
Cortex-Mx アーキテクチャの長所

- 1 つの命令セット（Thumb2）により、プログラムコードの複雑性を軽減し、ARM と Thumb 命令モードを切り替えるために必要なオーバーヘッドを排除します。
- ネストされたベクタ割り込みコントローラ（NVIC）により、割り込みのプロローグコードとエピローグコードを削除し、複数の設定可能な優先レベルを提供します。
- さまざまなスリープモードにより、超低消費電力を実現します。

Cortex-Mx マイクロコントローラアーキテクチャは、ハードリアルタイムシステム用に設計されていますが、複雑な System-on-Chip（SoC）アプリケーションでも使用できます。

クラシックおよび拡張 8051 デバイス

8051 デバイスは、コスト効率に優れたハードウェアと、最大限のコード効率とメモリアクセスの高速化を実現するためにさまざまなメモリ領域を使用するシンプルで効率的なプログラミングモデルを備えています。以下の図は、クラシック 8051 デバイスのメモリレイアウトを示しています。



8051 アーキテクチャには、以下の 3 つの物理メモリ領域があります。

- **DATA/IDATA** メモリは、レジスタバンクと、高速変数アクセスに使用されるビットアドレス指定可能な空間を備えた 256 バイトのオンチップ RAM で構成されています。一部のデバイスには、最大 64KB の拡張データ（EDATA）空間があります。
- **CODE** メモリは、プログラムコードと定数に使用される 64KB ROM 空間で構成されています。Keil リンカは、物理メモリ空間の拡張を可能にするコードバンキングをサポートしています。拡張されたバリアントでは、最大 16MB の ROM 空間を使用できます。

- **XDATA** メモリは、オフチップペリフェラルとメモリアドレス指定用の 64KB RAM 空間で構成されています。現在、大部分のデバイスに XDATA にマップされたオンチップ RAM が備えられています。
- **SFR** および IDATA メモリは同じアドレス空間に配置されていますが、アクセス時には異なるアセンブラ命令が使用されます。
- 拡張デバイスのメモリレイアウトは、1 つの 16MB アドレス領域にすべての 8051 メモリタイプを含むユニバーサルメモリマップを提供します。

8051 の特長

- 2 つまたは 4 つの優先レベルと最大 32 のベクトル割り込みを伴う高速割り込みサービスルーチン。
- 最小割り込みプロローグ/エピローグの 4 つのレジスタバンク。
- 効率的な論理演算のためのビットアドレス指定可能な空間。
- オンチップペリフェラルの緊密な統合のための 128 バイト特殊関数レジスタ（SFR）空間。一部のデバイスでは、ページングを使用して SFR 空間が拡張されます。
- 低電力で最大 100 MIPS の高速デバイスが利用可能。

8051 開発ツールサポート

Keil C51 コンパイラと Keil リンカ/ロケータは、以下の機能と C 言語拡張機能を備えた最適な 8051 アーキテクチャサポートを提供します。

- 直接 C 言語で書き込まれる、レジスタバンクサポートを備えた割り込み関数。
- 最適なブールデータ型をサポートするためのビットとビットアドレス指定可能変数。
- データオーバーレイにおけるコンパイル時間スタックでは、直接メモリアクセスを使用し、アセンブリプログラミングと比較してオーバーヘッドの少ない高速コードを実現します。

- 複数の割り込みまたはタスクスレッドで使用できる再入可能関数。
- 汎用のメモリ固有ポインタにより、柔軟なメモリアクセスが可能です。
- リンカコードパッキングでは、同一のプログラムシーケンスを再利用することにより、最大限のコード密度を実現します。
- コードと可変バンキングにより、物理メモリアドレス空間を拡張します。
- 絶対的な変数配置により、ペリフェラルアクセスとメモリ共有を実現します。

8051 メモリタイプ

メモリタイプ接頭文字は、定数を持つ式にメモリタイプを割り当てる場合に使用します。これは、出力コマンドのアドレスとして式が使用される場合などに必要です。通常、シンボル名にはメモリタイプが割り当てられているため、メモリタイプを指定する必要はありません。以下のメモリタイプが定義されています。

接頭文字	メモリ空間
C:	コードメモリ（CODE）
D:	内部直接アドレス指定可能 RAM メモリ（DATA）
I:	内部間接アドレス指定可能 RAM メモリ（IDATA）
X:	外部 RAM メモリ（XDATA）
B:	ビットアドレス指定可能 RAM メモリ
P:	ペリフェラルメモリ（VTREGD - 80x51 ピン）

接頭文字 **P:** は通常は名前の前に置かれる特殊な文字です。名前は、レジスタのピン名を含む特殊なシンボルテーブルで検索されます。

例:

C:0x100	CODE メモリ内のアドレス 0x100
ACC	DATA メモリ D: 内のアドレス 0xE0
I:100	内部 RAM 内のアドレス 0x64
X:0FFFFH	外部データメモリ内のアドレス 0xFFFF
B:0x7F	ビットアドレス 127 または 2FH.7
C	アドレス 0xD7（PSW.7）、メモリタイプ B:

Infineon C166、XE166、XC2000

これらのデバイスの 16 ビットアーキテクチャは、高速リアルタイムアプリケーション用に設計されています。このアーキテクチャでは、アドレス空間の一部にマップされた高速メモリ領域を有する最大 16MB のメモリ空間が提供されます。頻繁に使用する変数の高速メモリ領域への配置は、高性能アプリケーションに有効です。以下のメモリタイプによって、さまざまなメモリ領域がアドレス指定されます。

メモリタイプ	説明
bdata	**idata** メモリのビットアドレス指定可能な部分。
huge	高速 16 ビットアドレス演算を備えた完全な 16MB メモリ。オブジェクトサイズは 64KB までに制限されています。
idata	最大アクセス速度を実現する高速 RAM（**sdata** の一部）。
near	16 ビットポインタと 16 ビットアドレス演算を備えた効率的な変数と定数のアドレス指定（最大 64KB）。
sdata	ペリフェラルレジスタと追加のオンチップ RAM 空間を含むシステム領域。
xhuge	オブジェクトサイズが無制限のフルアドレス演算を備えた完全な 16MB メモリ。

C166、XE166、XC2000 の特長

- 優先レベルが 16 でベクトル割り込みが最大 128 の最高速割り込み処理。
- 最小割り込みプロローグ/エピローグの無制限レジスタバンク。
- 効率的な論理演算用のビット命令とビットアドレス指定可能空間
- ATOMIC 命令シーケンスは割り込みから保護されており、割り込み有効/無効シーケンスがありません。
- ペリフェラル割り込みによりトリガされる自動メモリ転送用のペリフェラルイベントコントローラ（PEC）。プロセッサインタラクションが必要ないため、割り込み応答時間がさらに向上します。
- 高速 DSP アルゴリズムの乗累算ユニット（MAC）。

C166、XE166、XC2000 開発ツールサポート

Keil C166 コンパイラは、C166、XE166、および XC2000 の特定機能をすべてサポートし、以下のような追加拡張機能を備えています。

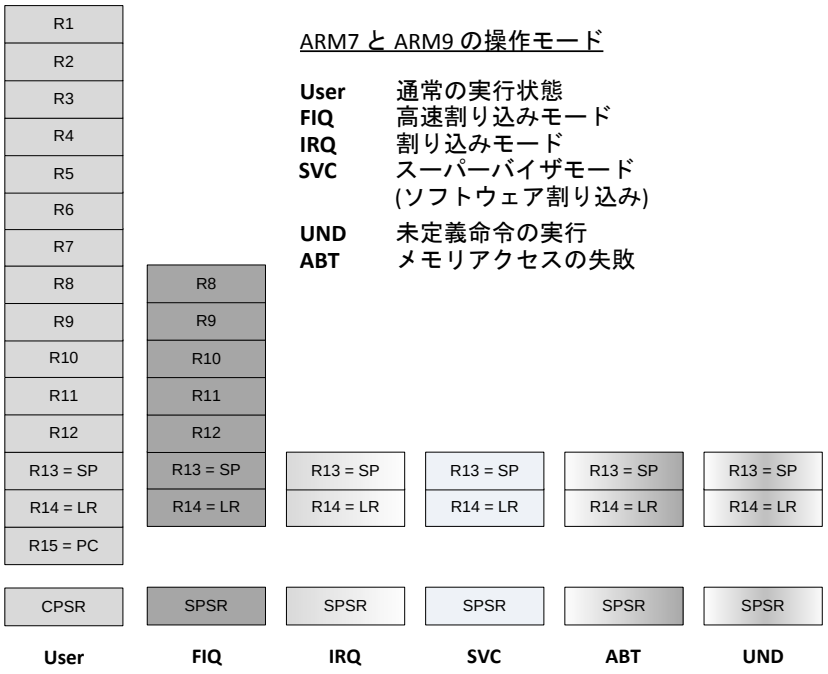
- メモリタイプサポートと、超高速変数アクセスの柔軟なデジタルパターン処理
- 呼び出し/戻り値オーバーヘッドを除去する関数インライン展開
- すべてのマイクロコントローラ命令と MAC 命令へのアクセス用のインラインアセンブリ

ARM7 および ARM9 ベースのマイクロコントローラ

ARM7 および ARM9 ベースのマイクロコントローラは、32 ビットレジスタと固定オペコード長を備えたロードストア RISC アーキテクチャ上で動作します。このアーキテクチャは、リニア 4GB メモリアドレス空間を備えています。上記の 8/16 ビットデバイスと異なり、メモリアドレス指定がマイクロコントローラレジスタの 32 ビットポインタを介して実行されるため、特定のメモリタイプは指定されていません。ペリフェラルレジスタは、リニアアドレス空間に直接マップされます。Thumb 命令セットにより、圧縮 16 ビット命令サブセットが提供され、コード密度が向上します。

ARM7 および ARM9 コアは使いやすく、コスト効率に優れ、最新のオブジェクト指向のプログラミング技術にも対応しています。このコアには、通常割り込み（IRQ）と高速割り込み（FIQ）ベクタを伴う 2 レベルの割り込みシステムが搭載されています。割り込みオーバーヘッドを最小限に抑えるために、通常の ARM7/ARM9 マイクロコントローラには、ベクタ割り込みコントローラが搭載されています。マイクロコントローラの動作モード、個別のスタック空間、およびソフトウェア割り込み（SVC）機能により、リアルタイムオペレーティングシステムを効率よく使用できます。

ARM7 および ARM9 コアには、13 の汎用レジスタ（R0−R12）、スタックポインタ（SP）R13、関数呼び出し時にリターンアドレスを保持するリンクレジスタ（LR）R14、プログラムカウンタ（PC）R15、およびプログラムステータスレジスタ（PSR）があります。シャドウレジスタは、さまざまな動作モードで使用可能で、レジスタバンクと類似しており、割り込みレイテンシを減少します。



ARM7 および ARM9 の特長

- **リニア 4 GB メモリ空間**。ペリフェラルが含まれており、特定のメモリタイプを指定する必要がありません。
- **効率的なポインタアドレス指定を行うロードストアアーキテクチャ**。複数のレジスタロード/ストアの使用により、高速タスクコンテキストスイッチを実現します。

- **標準（IRQ）割り込みと高速（FIQ）割り込み**。FIQ 上のバンクマイクロコントローラレジスタにより、レジスタ保存/復元オーバーヘッドが減少します。
- **ベクタ割り込みコントローラ**（ほとんどのマイクロコントローラで使用可能）。複数の割り込み処理を最適化します。
- **プロセッサモード**。予測可能なスタック要件のための個別の割り込みスタックを提供します。
- **コンパクト 16 ビット命令セット（Thumb）**。Thumb モードコードは、16 ビットメモリシステムから実行する際、ARM モードと比較するとコードサイズが約 65% になり、速度が 160% になります。

ARM7 および ARM9 開発ツールサポート

ARM コンパイルツールは、ARM 固有のすべての機能をサポートし、以下の機能を備えています。

- **関数インライン展開**により、呼び出し/リターンオーバーヘッドが削減され、パラメータの受け渡しが最適化されます。
- **インラインアセンブリ**により、C/C++ プログラムで特殊な ARM/Thumb 命令がサポートされます。
- **RAM 関数**により、高速割り込みコードおよびインシステム Flash プログラミングが有効になります。
- **ARM/Thumb のインターワーク**により、卓越したコード密度とマイクロコントローラパフォーマンスがもたらされます。
- **タスク関数と RTOS サポート**は、C/C++ コンパイラにビルドされます。

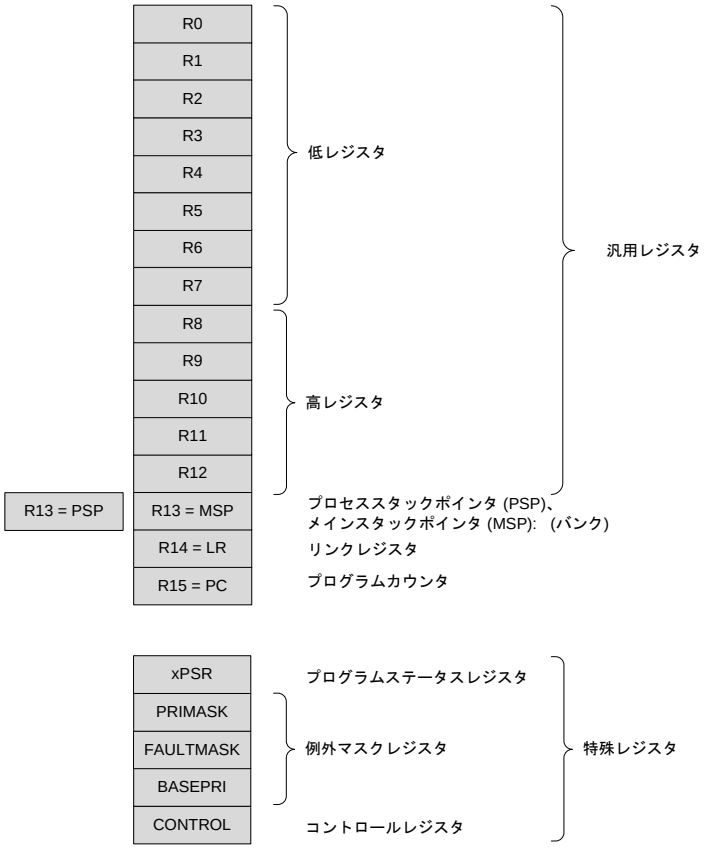
Cortex-Mx ベースのマイクロコントローラ

Cortex-Mx マイクロコントローラは、32 ビットマイクロコントローラ市場向けに設計され、低ゲート数での優れたパフォーマンスと、これまではハイエンドプロセッサにのみ搭載されていた機能を併せ持っています。

Cortex-Mx プロセッサは、4GB のリニア統合メモリ空間を備え、ビットバンディング機能を持ち、ビッグエンディアンおよびリトルエンディアンコンフィギュレーションに対応しています。事前に定義されたメモリタイプは使用可能ですが、一部のメモリ領域には追加の属性があります。コードは、外部 RAM である SRAM に配置できますが、コード領域に配置することをお勧めします。ペリフェラルレジスタは、メモリ空間にマップされます。プロセッサのバージョンに応じて、Thumb または Thumb2 命令セットによりコード密度が改善されます。

汎用レジスタは、R0 から R12 に順位付けされます。R13（SP）はバンク付きで、一度に表示可能な R13（MSP、PSP）のコピーは 1 つのみです。特殊レジスタは使用可能ですが、通常のデータ処理には使用されません。一部の 16 ビット Thumb 命令は、R0 ～ R7（低）のレジスタにのみアクセスできます。FIQ は存在しませんが、ネストされた割り込みと割り込み優先度の処理は、ネストされたベクタ割り込みコントローラ（NVIC）を介して行われ、割り込みレイテンシが大幅に減少します。

Cortex コアレジスタセット



Cortex-Mx の長所

- **ネストされたベクタ割り込みコントローラ**が複数の外部割り込みを最適化します（最大 240 + 1 NMI。8 以上の優先レベル）。
- **R0-R3、R12、LR、PSR、および PC** は、割り込みの開始時にスタックに**自動的にプッシュ**され、割り込み終了時にポップされます。
- **命令セットは Thumb2 のみ**で、ARM ロードマップ全体とのソフトウェアの上位互換性を保証します。
- **複数の超低電力モード**とウェイクアップ割り込みコントローラ（WIC）を備えています。

Cortex-Mx 開発ツールサポート

Keil MDK-ARM では、ARM 固有の特性に加え、Cortex-Mx マイクロコントローラソフトウェアインタフェース規格（CMSIS）に対応し、以下の機能を提供します。

- **コアレジスタとコアペリフェラル**に C/C++ 関数を使用してアクセスできます。
- RTOS カーネル用デバイス非依存デバッグチャネル
- **オブジェクト指向プログラミングをサポートし**、コードを再使用し、別のデバイスにコードを簡単に移植できます。
- **広範囲のデバッグ機能**により、プロセッサを停止することなく、メモリに直接アクセスできます。
- **CMSIS のサポート**により、Cortex-Mx アーキテクチャ全体でソフトウェアの互換性が維持されます。

アーキテクチャの比較結果

さまざまなアーキテクチャには、長所と短所があり、アプリケーション要件に応じて最適な選択は異なります。以下のコード比較のセクションでは、ターゲット組み込みシステムに最適なマイクロコントローラを選択する際に役立つ追加のアーキテクチャ情報を提供します。

コード比較

以下の簡潔で、典型的なコードサンプルを使用して、さまざまなマイクロコントローラアーキテクチャのそれぞれの長所について説明します。

I/O ポートアクセスの比較

ソースコード	説明

`if (IO_PIN == 1)  {`
`  i++;`
`}`
	I/O ピン設定時に値をインクリメントします。

- **8051** デバイスには、ビットアドレス指定可能な I/O ポートと、固定されたメモリ位置に直接アクセスするための命令があります。
- **C166**、**XE166**、および **XC2000** デバイスには、ビットアドレス指定可能な I/O ポートと、固定されたメモリ位置に直接アクセスするための命令があります。
- **ARM7 および ARM9** デバイスには、間接的メモリアクセス命令のみがあり、ビット演算はありません。
- **Cortex-Mx** デバイスにあるのは間接的メモリアクセス命令のみですが、アトミックビット演算を使用できます。

8051 コード	C166/XE166 および XC2000 コード	ARM7 および ARM9 Thumb コード	Cortex-Mx Thumb2 コード

`sfr  P0=0x80;`
`sbit P0_0=P0^0;`
 	
`unsigned char i;`
 	
`void main (void) {`
` if (P0_0) {`
`;  JNB  P0_0,?C0002`
 	
` i++;`
`;  INC  i`
 	
` }`
 	
`;  RET`
`}`
	`sfr  P0L=0xFF00;`
	`sbit P0_0=P0L^0;`
	
	`unsigned int i;`
	
	`void main (void) {`
	` if (P0_0) {`
	`;  JNB  P0_0,?C0001`
	
	` i++;`
	`;  SUB i,ONES`
	
	` }`
	
	`;  RET`
	`}`
		`#define IOP *(int*))`
		
		`unsigned int i;`
		
		`void main (void) {`
		` if (IOP & 1) {`
		`; LDR R0,=0xE0028000`
		`; LDR R0,[R0,#0x0]`
		`; MOV R1,#0x1`
		`; TST R0,R1`
		`; BEQ L_1`
		
		` i++;`
		`; LDR R0,=i ; i`
		`; LDR R1,[R0,#0x0];i`
		`; ADD R1,#0x1`
		`; STR R1,[R0,#0x0];i`
		` }`
		
		`; BX  LR`
		`}`
			
			`unsigned int i;`
			
			`void main (void) {`
			` if (GPIOA->ODR) {`
			`; STR   R0,[R1,#0xc]`
			`; LDR   R0,[R2,#0]`
			`; CBZ   R0,|L1.242|`
			
			` i++;`
			`; MOVS  R0,#2`
			`; STR   R0,[R1,#0xc]`
			`; |L1.242|`
			
			` }`
			
			`; BX    LR`
			`}`

6 バイト	**10 バイト**	**24 バイト**	**12 バイト**

ポインタアクセスの比較

ソースコード	説明

`typedef struct { int x; int arr[10]; } sx;`
 	
`int f (sx xdata *sp, int i) {`
`  return sp->arr[i];`
`}`
	構造体の一部であり、ポインタを介して間接的にアクセスされる値を返します。

- **8051** デバイスでは、アドレス演算用に複数のマイクロコントローラ命令を必要とするバイト演算が可能です。
- **C166**、**XE166**、および **XC2000** デバイスでは、大容量 16 MB アドレス空間の直接サポートによる効果的なアドレス演算が可能です。
- **ARM** デバイスは、ポインタアドレス指定に関して非常に効率がよく、常に 32 ビットアドレス指定モードを使用します。
- **Cortex-Mx** デバイスでは、データ構造体および配列のポインタとしてすべてのレジスタを使用できます。

8051 コード	C166、XE166、および XC2000 コード	ARM 7 および ARM9 Thumb コード	Cortex-Mx Thumb2 コード

`MOV DPL,R7`
`MOV DPH,R6`
`MOV A,R5`
`ADD A,ACC`
`MOV R7,A`
`MOV A,R4`
`RLC A`
`MOV R6,A`
`INC DPTR`
`INC DPTR`
`MOV A,DPL`
`ADD A,R7`
`MOV DPL,A`
`MOV A,DPH`
`ADDC A,R6`
`MOV DPH,A`
`MOVX A,@DPTR`
`MOV R6,A`
`INC DPTR`
`MOVX A,@DPTR`
`MOV R7,A`
	`MOV R4,R10`
	`SHL R4,#01H`
	`ADD R4,R8`
	`EXTS R9,#01H`
	`MOV R4,[R4+#2]`
		`LSL R0,R1,#0x2`
		`ADD R0,R2,R0`
		`LDR R0,[R0,#0x4]`
			`ADD R0,R0,R1,LSL #2`
			`LDR R0,[R0,#4]`

25 バイト	14 バイト	6 バイト	6 バイト

最適なコードの生成

Keil が提供する C/C++ コンパイラは、コード生成においてトップクラスであり、非常に効率的なコードを生成します。ただし、コードの生成と変換は、アプリケーションソフトウェアがどのように記述されているかに影響されます。以下のヒントは、アプリケーションパフォーマンスの最適化に役立ちます。

すべてのアーキテクチャのためのコーディングのヒント

ヒント	説明
割り込み関数は短くする	適切に構成された割り込み関数は、データ収集や時間管理のみを実行します。データ処理は、main 関数で実行されるか、または RTOS タスク関数により実行されます。これにより、割り込み関数のコンテキスト保存/復元に関連したオーバーヘッドが減少します。
アトミック演算の要件をチェックする	アトミックコードは、main 関数によって使用されるメモリにアクセスする複数の RTOS スレッドまたは割り込みルーチンの使用中にデータにアクセスする場合に必要です。アトミック演算が必要かどうかについてアプリケーションを注意深くチェックし、生成されたコードを検証します。アーキテクチャによって、チェックすべき内容は異なります。例えば、8051 と C166/XE166/XC2000 デバイスでの変数のインクリメントは、割り込みできないため単一のアトミック命令ですが、ARM デバイスでのインクリメントの場合、複数の命令が必要です。一方、8051 では、**int** 変数のメモリにアクセスするには、複数の命令が必要です。
割り込み、ハードウェアペリフェラル、またはその他の RTOS タスクによって変更される変数に、**volatile**属性を適用する	**volatile**属性は、C/C++ コンパイラによる変数アクセスの最適化を抑制します。デフォルトでは、C/C++ コンパイラは、変数値がメモリ読み出し操作の間で変化しないとみなす場合があります。これにより、リアルタイムアプリケーションでのアプリケーションの動作が不正確になる可能性があります。
可能な場合、ループとその他の一時演算に**自動**変数を使用する	最適化プロセスの一部として、Keil C/C++ コンパイラは、CPU レジスタのローカル変数（関数レベルで定義）を維持しようしとします。レジスタアクセスは、最も速いメモリアクセス方法であり、必要なプログラムコードは最小限です。

8051 アーキテクチャのためのコーディングのヒント

ヒント	説明
変数には可能な最小のデータ型を使用し、可能であれば **unsigned char** および **bit** にする	8051 では、幅広いビットサポートのある 8 ビット CPU を使用します。大部分の命令は、8 ビット値またはビットで動作します。このため、データ型が小さいほど、より効率的なコードが生成されます。
可能であれば、**unsigned** データ型を使用する	8051 は、signed データ型を直接サポートしません。符号付き演算には、追加命令が必要であるのに対して、unsigned データ型はアーキテクチャにより直接サポートされています。
可能であれば **SMALL** メモリモデルにする	大部分のアプリケーションは、**SMALL** メモリモデルを使用して書き込むことができます。明示的なメモリタイプを使用して、大きなオブジェクト（配列または構造）を **xdata** または **pdata** メモリに配置できます。Keil C51 ランタイムライブラリでは汎用ポインタを使用しており、どのようなメモリタイプを使用しても機能します。
他のメモリモデルを使用している場合、頻繁に使用される変数にメモリタイプ **data** を適用する	**data** アドレス空間の変数は、8051 命令セットにエンコードされた 8 ビットアドレスで直接アクセスされます。このメモリタイプにより最も効率的なコードが生成されます。
使用するデバイスでの **pdata** メモリタイプの使用方法を学ぶ	**pdata** メモリは、8 ビットアドレス指定で MOVX @Ri 命令を使用して、256 バイトへの効率的なアクセスを可能にします。だだし、ページングレジスタの設定が必要なため、さまざまな 8051 デバイスで **pdata** の動作が異なります。**xdata** メモリタイプは、汎用であり、大きなメモリ空間（最大 64KB）にアクセスします。
可能であれば、**メモリタイプ付きポインタ**を使用する	デフォルトでは、Keil C51 コンパイラでは、どのようなメモリタイプにもアクセスできる汎用ポインタが使用されます。メモリタイプ付きポインタがアクセスできるのは、固定メモリ空間のみですが、より高速で、小さなコードを生成できます。
再入可能関数の使用を減らす	8051 ではスタック変数をサポートしていません。再入可能関数は、メモリを最大限に利用するためにデータオーバーレイとコンパイル時間スタックを使用して、Keil C 51 コンパイラにより実行されます。8051 の再入可能関数には、スタックアーキテクチャのシミュレーションが必要です。組み込みアプリケーションで再入可能コードが必要な場合は少ないため、再入可能属性の使用を最少限にする必要があります。
プログラムサイズを小さくするために、**LX51 リンカ/ロケータ**と**リンカコードパッキング**を使用する	拡張 LX51 リンカ/ロケータ（PK51 プロフェッショナルデベロッパーキットのみで使用可能）によりプログラム全体が解析され、最適化されます。3 バイト **LJMP** や **LCALL** の代わりに 2 バイト **AJMP** と **ACALL** 命令を最大化するためにコードがメモリ内で再配列されます。**リンカコードパッキング**（C51 OPTIMIZE レベル 8 以上で有効）により共通コードブロックのサブルーチンが生成されます。

C166、XE166、XC2000 アーキテクチャのためのコーディングのヒント

ヒント	説明
可能であれば、**自動**変数と**パラメータ**変数に 16 ビットデータ型を使用する	パラメータの受け渡しは、16 ビット CPU レジスタで実行されます（16 ビットレジスタの多くは、自動変数に使用できます）。より多くの16 ビット変数（**signed/unsigned int/short**）を CPU レジスタに割り当てることができます。これにより、より効率的なコードが生成されます。
可能であれば、**long** を **int** データ型に置き換える	16 ビット型（**int** および **unsigned int** など）を使用する演算は、**long** 型を使用する演算よりも効率的です。
boolean 変数には **bit** データ型を使用する	これらの CPU には、**bit** データ型を使用する Keil C166 コンパイラによって完全にサポートされている効率的なビット命令があります。
可能であれば、**SMALL** または **MEDIUM** メモリモデルを使用する	これらのメモリモデルでは、デフォルトの変数位置は **near** メモリ内です。これは、CPU 命令でエンコードされる 16 ビット直接アドレスを介してアクセスできます。大きなオブジェクト（配列または構造）は、明示的なメモリタイプを使用して、**huge** または **xhuge** に配置できます。
他のメモリモデルを使用している場合、頻繁に使用される変数に **near**、**idata** または **sdata** メモリタイプを適用する	**near**、**idata** または **sdata** アドレス空間の変数は、単一の C166/XE166/XC2000 命令に直接エンコードされる 16 ビットアドレスを介してアクセスされます。これらのメモリタイプは、最も効率的なコードを生成します。
メモリモデル **COMPACT/LARGE** の代わりに **HCOMPACT/HLARGE** を使用する	メモリモデル COMPACT と LARGE では、現在使用されていない **far** メモリタイプを使用し、オブジェクトサイズは 16KB に制限されています。メモリモデル HCOMACT と HLARGE では、**huge** メモリタイプを使用し、オブジェクトサイズは 64KB に制限されています。**near** から **huge** ポインタへのキャスト操作も、より最適化されます。
可能であれば、**near ポインタ**を使用する	**near** ポインタは、**near**、**idata**、または **sdata** アドレス空間の変数にアクセスできるので、メモリへのアクセスが可能かどうかを確認します。**near** ポインタによって、より高速で小さなコードが生成されます。

ARM7 および ARM9 アーキテクチャのためのコーディングのヒント

ヒント	説明
可能であれば、**自動**変数と**パラメータ**変数には 32 ビットデータ型を使用する	パラメータの受け渡しは、32 ビット CPU レジスタで実行されます。すべての ARM 命令は 32 ビット値に対して演算を実行します。Thumb モードでは、すべてのスタック命令は 32 ビット値に対してのみ演算を実行します。32 ビットデータ型（**signed/unsigned int/long**）を使用すると、追加データ型のキャスト操作が必要ありません。
Thumb 命令セットを使用する	Thumb モードは、16 ビットメモリシステムから実行する場合、ARM モードと比較して、コードサイズが約 65% になり、速度が 160% になります。MDK-ARM コンパイラにより、必要な ARM/Thumb インターワーク命令が自動的に挿入されます。
アトミックシーケンスに **__swi** ソフトウェア割り込み関数を使用する	MDK-ARM コンパイラで、__swi 関数属性を使用すると、IRQ による割り込みが不可能なソフトウェア割り込み関数を直接生成できます（__swi 関数は FIQ 割り込みによって割り込み可能です）。他の組み込みアーキテクチャとは異なり、ARM では、**ユーザモード**での割り込み無効ビット **I** および **F** へのアクセスを禁止しています。
スカラを**先頭**に配置し、**後続の struct** メンバとして**配列**を配置して、`struct` ポインタのアクセスを強化する	Thumb 命令と ARM 命令によって、メモリアクセスの制限付き変位がエンコードされます。ポインタで`struct` にアクセスする場合、`struct` の先頭にあるスカラ変数に直接アクセスできます。配列は、常にアドレス演算を必要とします。このため、スカラ変数を`struct` の先頭に配置するとより効率的です。
高速割り込みコードを RAM に割り当てる	Flash ROM からのコード実行には、一般的にウェイト状態または CPU ストールが必要です。RAM からのコード実行には必要ありません。そこで、タイムクリティカルな関数（高速割り込みコードなど）は、そのファイルの**コンテキストメニュー**から**［Options for File］→［Properties］**を選択して**［Memory Assignment］**機能を使用し、RAM 内に直接配置できます。
サイズに合わせて最適化する	最小プログラムサイズになるようにアプリケーションを最適化するには、**［Options for Target］**で以下のツールチェーンを選択します。 ▪ ［Target］ダイアログページで、［Code Generation - Use Cross-Module Optimization］を有効にします。 ▪ **［C/C++］**ダイアログページで、［Optimization:Level 2 (-O2)］を選択し、**［Optimize for Time］**、**［Split Load and Store Multiple］**、および**［One ELF Section per Function］**オプションを無効にします。

ヒント	説明
MicroLIB	コンパイラには、アプリケーションの**コードサイズ**をさらに**小さくする**ために使用される MicroLIB が用意されています。MicroLIB は、高度な組み込みシステムを対象としており、ANSI には完全には準拠していません。 実行速度が重要でない場合は、MicroLIB を使用しないで下さい。
速度に合わせて最適化する	最大実行速度になるようにアプリケーションを最適化するには、［**Options for Target**］で以下のツールチェーンを選択します。 ▪ ［**Target**］ダイアログページで、［**Code Generation - Use Cross-Module Optimization**］を有効にします。 ▪ ［**C/C++**］ダイアログページで、［Optimization:Level 3 (-O3)］を選択し、［**Optimize for Time**］を有効にし、［**Split Load and Store Multiple**］を無効にします。

Cortex-Mxアーキテクチャのためのコーディングのヒント

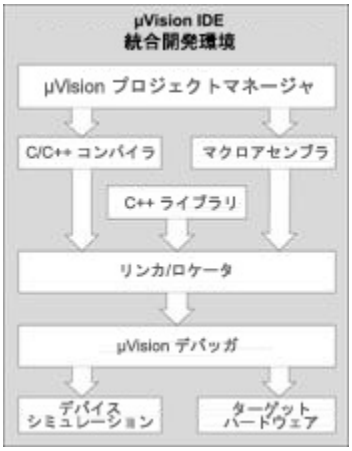
ヒント	説明
可能であれば、**自動**変数と**パラメータ**変数には 32 ビットデータ型を使用する	パラメータの受け渡しは、32 ビット CPU レジスタで実行されます。すべての ARM 命令は 32 ビット値に対して演算を実行します。Thumb モードでは、すべてのスタック命令は 32 ビット値に対してのみ演算を実行します。32 ビットデータ型（**signed/unsigned int/long**）を使用すると、追加データ型のキャスト操作が必要ありません。
サイズに合わせて最適化する	最小プログラムサイズになるようにアプリケーションを最適化するには、**［Options for Target］**で以下のツールチェーンを選択します。 ▪ ［Target］ダイアログページで、［Code Generation - Use Cross-Module Optimization］を有効にします。 ▪ **［C/C++］**ダイアログページで、［Optimization:Level 2 (-O2)］を選択し、**［Optimize for Time］**、**［Split Load and Store Multiple］**、および**［One ELF Section per Function］**オプションを無効にします。
MicroLIB	コンパイラには、アプリケーションの**コードサイズ**をさらに**小さくする**ために使用される MicroLIB が用意されています。MicroLIB は、高度な組み込みシステムを対象としており、ANSI には完全には準拠していません。 実行速度が重要でない場合は、MicroLIB を使用しないで下さい。
速度に合わせて最適化する	最大実行速度になるようにアプリケーションを最適化するには、**［Options for Target］**で以下のツールチェーンを選択します。 ▪ **［Target］**ダイアログページで、**［Code Generation - Use Cross-Module Optimization］**を有効にします。 ▪ **［C/C++］**ダイアログページで、［Optimization:Level 3 (-O3)］を選択し、**［Optimize for Time］**を有効にし、**［Split Load and Store Multiple］**を無効にします。
スリープモード機能	アプリケーションの消費電力を最適化するには、WFI 命令を使用して、次の割り込みを受け取るまでプロセッサをスリープモードにします。C プログラムでは、コンパイラ組み込み関数 **__wfi()** を使用して、この命令を目的のコードに挿入します。
スカラを**先頭**に配置し、**後続の struct** メンバとして**配列**を配置して、`struct` ポインタのアクセスを強化する	Thumb2 命令はメモリアクセスに制限付きのオフセットがエンコードされます。ポインタで `struct` にアクセスする場合、`struct` の先頭にあるスカラ変数に直接アクセスできます。配列は常にアドレス演算を必要とします。このため、スカラ変数を `struct` の先頭に配置するとより効率的です。

第 3 章 開発ツール

Keil 開発ツールには、組み込みアプリケーションを迅速に開発して、開発を成功させるために役立つ数多くの機能と長所があります。これらは使いやすく、お客様の設計目標を直ちに達成するのに役立つことが保証されています。

ソフトウェア開発サイクル

Keil ツールを使用する場合、プロジェクト開発サイクルは他のソフトウェア開発プロジェクトとほぼ同じです。



1. プロジェクトを作成し、デバイスデータベースからターゲットデバイスを選択し、ツール設定を構成します。
2. C/C++ またはアセンブリでソースファイルを作成します。
3. プロジェクトマネージャを使用してアプリケーションをビルドします。
4. ソースファイルのエラーをデバッグして修正し、アプリケーションを確認して最適化します。
5. コードをフラッシュ ROM または SRAM にダウンロードし、リンクされたアプリケーションをテストします。

ブロック図に示されている各コンポーネントについては、以下のセクションで説明します。

µVision IDE

µVision IDE は強力なエディタ、プロジェクトマネージャ、および Make ユーティリティツールを組み合わせた Windows ベースのソフトウェア開発プラットフォームです。µVision では、C/C++ コンパイラ、マクロアセンブラ、リンカ、ライブラリマネージャ、Object-HEX コンバータなどの Keil ツールをすべてサポートしています。µVision は以下を提供して開発プロセスを早めるのに役立ちます。

- 特定のマイクロコントローラ用のデバイスを選択し、開発ツールを設定するための**デバイスデータベース**
- プロジェクトの作成、メンテナンス用の**プロジェクトマネージャ**
- 組み込みアプリケーションをアセンブル、コンパイル、リンクするための **Make ユーティリティ**
- 十分な機能を備えたソースコードエディタ
- 共通テキストシーケンスまたはヘッダブロックを挿入するために使用できる**テンプレートエディタ**
- アプリケーション内のコードオブジェクトをすばやく探索し、データを見つけて分析するための**ソースブラウザ**
- プログラム内の関数をすばやくナビゲートするための**関数ブラウザ**
- ソースファイル内でビジュアルスコープを管理するための関数アウトライン
- ソースコードのコメント付け/コメント解除を行うための**ファイル内検索**や関数などのビルトインユーティリティ
- µVision **シミュレータ**と**ターゲットデバッガ**との完全な統合
- マイクロコントローラのスタートアップコードとコンフィギュレーションファイルをグラフィカルに編集できる**コンフィギュレーションウィザード**
- **ソフトウェアバージョンコントロールシステム**やサードパーティ製ユーティリティを設定するためのインタフェース

- Keil ULINK USB-JTAG アダプタのファミリなどの**フラッシュプログラミングユーティリティ**
- あらゆる開発ツール設定用の**ダイアログ**
- **オンラインヘルプ**と、マイクロコントローラデータシートおよびユーザガイドへのリンク

µVision デバイスデータベース

µVision デバイスデータベースは、デバイスおよびプロジェクトパラメータの選択と設定に役立ちます。事前定義の設定も用意されているので、アプリケーション要件に専念できます。また、独自のデバイスを追加したり、既存の設定を変更することもできます。デバイスデータベースの機能を使用すると、以下の作業が可能です。

- スタートアップコードおよびデバイス設定を初期化します。
- アセンブラ、コンパイラ、およびリンカのコンフィギュレーションオプションをロードします。
- マイクロコントローラコンフィギュレーション設定を追加および変更できます。

µVision デバッガ

µVision デバッガは、µVision IDE に完全に統合されています。これには以下の機能があります。

- アセンブラ、テキスト、混合モードなどのさまざまな**段階モード**と**ビューモード**でプログラムを実行することによる C/C++ ソースレベルまたはアセンブリレベルでのコードの**逆アセンブル**
- アクセスブレークポイントや複雑なブレークポイントなどの複数の**ブレークポイント**オプション
- すばやく重要な場所を見つけて定義するための**ブックマーク**
- メモリ、変数、およびレジスタ値の**確認**および**変更**
- スタック変数などのプログラムコールツリーのリスト

- オンチップマイクロコントローラ**ペリフェラル**のステータスの**確認**
- デバッグコマンドや C ライクスクリプト作成機能
- 経過時間と、各命令に必要なサイクル数を記録して表示する**実行プロファイリング**
- セーフティクリティカルアプリケーションのテスト用のコードカバレッジ統計
- 統計を表示し、変数およびペリフェラル I/O シグナルの値を記録して時系列に表示するさまざまな**解析ツール**
- 実行された命令の**履歴**を表示する**命令トレース**機能
- **カスタマイズ**された画面とウィンドウ**レイアウト**の定義

µVision デバッガには、**シミュレータモード**と**ターゲットモード**の 2 つの動作モードがあります。

シミュレータモードは、µVision デバッガを *ソフトウェアのみの製品*として設定し、命令や大部分のオンチップペリフェラルを含むターゲットシステムを精密にシミュレートします。このモードでは、ハードウェアを入手する前にアプリケーションコードをテストでき、信頼性の高い組み込みソフトウェアの迅速な開発が可能となります。シミュレータモードには以下の機能があります。

- ハードウェア環境がなくてもデスクトップでソフトウェアテストが可能になります。
- 関数ベースでの早期のソフトウェアデバッグにより、ソフトウェアの信頼性が向上します。
- ハードウェアデバッガでは不可能なブレークポイントが可能になります。
- 最適な入力シグナル。ハードウェアデバッガでは、余分なノイズが加わります。
- シグナル処理アルゴリズムを介したシングルステップが可能です。マイクロコントローラが停止すると、外部シグナルは停止します。
- 実際のハードウェアペリフェラルを破損する可能性のある間違ったシナリオを検出できます。

ターゲットモード[1] では µVision デバッガを*実際のハードウェア*に接続します。以下の項目にインタフェース接続するためのさまざまなターゲットドライバがあります。

- **ULINK JTAG/OCDS アダプタ**。オンチップデバッグシステムに接続します。
- **モニタ**。ユーザハードウェアと統合可能で、多くの評価ボードで使用できます。
- **エミュレータ**。ターゲットハードウェアのマイクロコントローラピンに接続します。
- **インシステムデバッガ**。ユーザアプリケーションプログラムの一部で、基本的なテスト機能があります。
- **ULINK*Pro* アダプタ**。JTAG/SWD/SWV 経由でオンチップデバッグシステムに接続する高速のデバッグおよびトレースユニット。Cortex-M3 ETM 命令トレース機能を備えています。

アセンブラ

アセンブラを使用すると、マイクロコントローラの命令を使用してプログラムを記述できます。最大限の速度、小さいコードサイズ、正確なハードウェア管理が必要不可欠な場合に使用されます。Keil アセンブラは、ソースレベルのシンボリックデバッグ機能をサポートしながら、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。また、マクロ処理などの強力な機能もサポートされます。

アセンブラにより、アセンブリソースファイルが再配置可能なオブジェクトモジュールに変換され、オプションでシンボルテーブルと相互参照の詳細が記載されたリスト型ファイルを作成できます。生成されたオブジェクトファイルには、完全な行番号、シンボル、および型情報が書き込まれます。この情報を使用すれば、プログラム変数を正確にデバッガに表示できます。行番号は µVision デバッガまたはその他のサードパーティ製デバッグツールを使用したソースレベルデバッグに使用されます。

[1] 一部のターゲットドライバにはハードウェア制限があり、ターゲットハードウェアのデバッグ時に µVision デバッガの機能が制限または除外されます。

Keil アセンブラでは、いくつかの異なるタイプのマクロプロセッサ（アーキテクチャにより異なる）をサポートしています。

- **標準マクロプロセッサ**はより使いやすいマクロプロセッサです。このプロセッサにより、他の多くのアセンブラで使用されている構文と互換性のある構文を使用してアセンブリプログラムでマクロを定義して使用できます。
- **マクロ処理言語（MPL）**は、Intel ASM-51 マクロプロセッサと互換性のある文字列置換機能です。MPL には、文字列処理や番号処理のような便利な動作を実行する、事前定義のマクロプロセッサ機能があります。

一般的に使用されるシーケンスは 1 度開発するだけでいいので、マクロを使用すれば、開発時間やメンテナンス時間が節約できます。

アセンブラのマクロプロセッサのもう 1 つの強力な特長は、条件付きアセンブリ機能です。条件付きアセンブリは、アセンブリプログラム内のコマンドラインディレクティブやシンボルを介して起動できます。コードセクションの条件付きアセンブリは、非常にコンパクトなコードを実現するのに役立ちます。これを使用すれば、単一のアセンブリソースファイルからさまざまなアプリケーションを生成することも可能です。

C/C++ コンパイラ

ARM C/C++ コンパイラは、ARM7、ARM9、および Cortex-Mx プロセッサアーキテクチャ用の高速でコンパクトなコードを生成するために設計されています。Keil ANSI C コンパイラは、8051、C166、XE166、および XC2000 アーキテクチャを対象としています。これらのコンパイラにより、アセンブリプログラミングの効率性とスピードに見合ったオブジェクトコードが生成可能です。C/C++ のような高レベルの言語を使用することにより、アセンブリ言語によるプログラミングよりも多くの利点が得られます。

- プロセッサ命令セットの知識は必要ありません。マイクロコントローラのアーキテクチャの基本的な知識があれば望ましいですが、必須ではありません。

- レジスタ割り当てやさまざまなメモリタイプおよびデータ型のアドレス指定などの詳細は、コンパイラにより管理されます。
- プログラムは、公式構造（C/C++ プログラミング言語により定められた）を受け取り、個別の関数に分けることができます。これは、アプリケーション構造の改善だけでなくソースコードの再利用に役立ちます。
- 人間の思考プロセスに近いキーワードと操作機能を使用できます。
- ソフトウェア開発とデバッグの時間が大幅に削減されます。
- ランタイムライブラリには、形式が指定された出力、数値化、浮動小数点演算などの標準ルーチンがあります。
- モジュールプログラミング技術により、既存のプログラム部分を新しいプログラムに容易に統合できます。
- C/C++ 言語は ANSI 標準に基づく移植可能な言語であり、一般に幅広く支持され、ほとんどのシステムにおいて容易に使用できます。既存のプログラムコードを必要に応じて他のプロセッサに簡単に適合させることができます。

Object-HEX コンバータ

Object-HEX コンバータにより、リンカによって作成された絶対オブジェクトモジュールから Intel HEX ファイルが作成されます。Intel HEX ファイルは、アプリケーションプログラムの 16 進数表現を含む ASCII ファイルです。これらは、ROM、EPROM、FLASH、またはその他のプログラム可能なメモリに書き込むために、デバイスプログラムに容易にロードできます。Intel HEX ファイルを操作して、チェックサムまたは CRC データを容易に取り込むことができます。

リンカ/ロケータ

リンカ/ロケータにより、複数のオブジェクトモジュールを組み合わせて 1 つの実行可能なプログラムにすることができます。これにより、外部またはパブリック参照が解決され、絶対アドレスが再配置可能なプログラムセグメントに割り当てられます。リンカにより、適切なランタイムライブラリモジュールが自動的に取り込まれ、コンパイラやアセンブラにより作成されたオブジェクトモジュールが処理されます。リンカは、コマンドラインまたは µVision IDE 内から呼び出すことができます。デフォルトのリンカディレクティブは、大部分のアプリケーションに適合するよう注意深く選択されており、追加オプションは必要ありません。ただし、アプリケーションのカスタム設定を追加指定することも簡単です。

ライブラリマネージャ

ライブラリマネージャは、C/C++ コンパイラおよびアセンブラにより作成されたオブジェクトモジュールのライブラリを作成およびメンテナンスします。ライブラリファイルを使用すると、リンカで使用する多数のモジュールを簡単に組み合わせて参照できます。

リンカには、アプリケーションで使用される外部変数および関数を解決するためのライブラリが含まれています。必要な場合にのみライブラリ内のモジュールが抽出され、プログラムに追加されます。モジュールに含まれているルーチンがプログラムで呼び出されない場合、そのモジュールは最終出力に含まれません。リンカによってライブラリから抽出されたオブジェクトモジュールは、他のオブジェクトモジュールと同様に処理されます。

ライブラリを使用すると、多くの利点があります。セキュリティ、スピード、ディスク容量の最小化は、そのほんの一部です。ライブラリを使用すると、オリジナルのソースコードを配布せずに多数の関数とルーチンを配布できます。例えば、ANSI C ライブラリは、ライブラリファイル一式として提供されています。

µVision **プロジェクトマネージャ**を使用すると、実行可能プログラムの代わりにライブラリファイルをビルドできます。これを実行するには、［**Options for Target**］→［**Output**］ダイアログで［**Create Library**］チェックボックスをオンにします。または、［**Command**］ウィンドウからライブラリマネージャを呼び出すこともできます。

第 4 章 RTX RTOS カーネル

本章では、リアルタイムオペレーティングシステム（RTOS）を使用する利点について説明し、Keil RTX カーネルで使用可能な機能を紹介します。Keil 開発ツールには多くのサードパーティ製 RTOS ソリューションとの互換性があります。Keil RTX を使用する必要はありませんが、RTX カーネルは開発ツールに完全に統合されており、多機能で、組み込みシステムの要件に合わせて十分にカスタマイズされています。

ソフトウェア概念

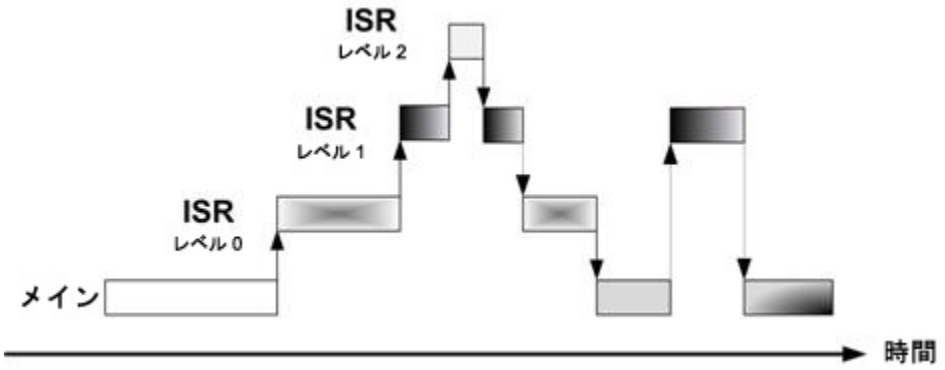
組み込みアプリケーションには、以下の 2 つの基本設計概念があります。

- **無限ループ設計：**この設計には、無限ループとしてのプログラムの実行が含まれています。割り込みサービスルーチン（ISR）によりデータ処理を伴うタイムクリティカルなジョブが実行されている間、プログラム関数（タスク）がループ内から呼び出されます。
- **RTOS 設計：**この設計には、**リアルタイムオペレーティングシステム（RTOS）**を使用したいくつかのタスクの実行が含まれています。RTOS は、タスク間通信と時間管理関数を提供します。プリエンプティブ RTOS により、タイムクリティカルなデータ処理が最優先タスクで実行されるので、割り込み関数の複雑さが緩和されます。

無限ループ設計

無限ループでの組み込みプログラムの実行は、単純な組み込みアプリケーションに適したソリューションです。通常はハードウェア割り込みによってトリガされるタイムクリティカルな機能は、必須のデータ処理が実行される ISR 内で実行されます。メインループには、タイムクリティカルではないが、バックグラウンドで実行される基本演算のみが含まれています。

このソフトウェア概念に必要なスタック空間は 1 つで、メモリに制限のあるデバイスに最適です。複数の割り込みレベルを備えたアーキテクチャにより、複雑な下位の ISR 関数実行が可能となります。タイムクリティカルなジョブは、上位の割り込みレベルで実行可能です。



8051、C166/XE166/XC2000、および ARM Cortex-Mx マイクロコントローラには複数の割り込みレベルがあります。上位の割り込みが、下位の割り込みまたは main 関数を中断することがあります。

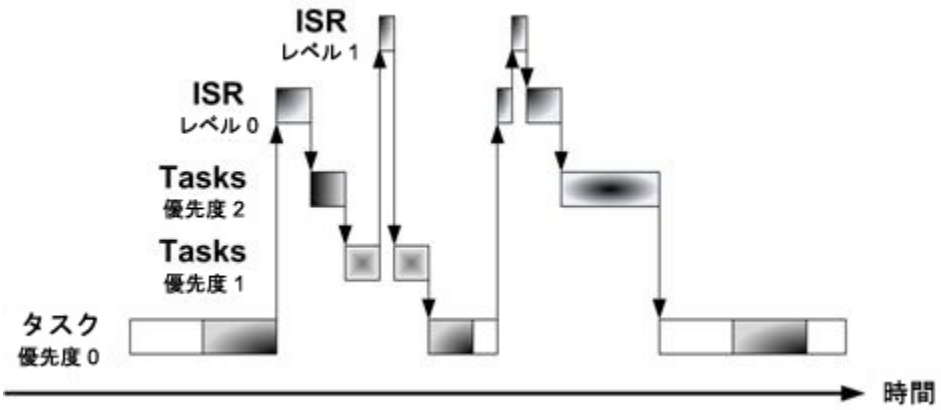
優先度の高い割り込みによる場合を除き、ISR の実行を中断することはできません。そのため、優先度の高い割り込みに CPU 処理時間のほとんどが費やされるため、複雑な ISR レベルが多い場合はタイミングを予測できません。

もう 1 つの問題は最悪の場合のスタックネストを特定することです。ISR 設計が複雑なアプリケーションではスタックリソースの問題が見過ごされる場合があり、突発的な実行エラーの原因になります。ARM アーキテクチャには ISR 用の追加のスタックがあり、メインループ実行時にスタックメモリの予想外のエラーを避けることができます。

RTOS 設計

RTOS 設計は、その性質により、順次タイムスライスで複数のタスク実行を可能にします。プリエンプティブ RTOS のタスク優先レベルでは、優先度の高いタスクが優先度の低いタスクの実行に割り込みます。大部分の RTOS システムでは、複雑なアプリケーション設計を可能にする、タスク間通信と時間遅延関数を実装しています。

ARM ベースのアーキテクチャは、RTOS 用に設計されています。RTOS は、いくつかの割り込みソースがある ARM7、ARM9、および Cortex-Mx ベースのシステムでほとんどの場合に必須です。ARM デバイスには、個別の ISR スタックがあります。このため、各タスクが 8051 および C166/XE166/XC2000 デバイスで必要な ISR を実行するために追加のスタックを確保しておく必要はありません。



プリエンプティブ RTOS は複数のタスク優先度をサポートしています。同じ優先度のタスクは順次実行されます。優先度の高いタスクが優先度の低いタスクを中断します。ISR は常にタスクの実行に割り込み、他のタスクとデータのやり取りを行う場合もあります。

RTOS は、組み込みアプリケーションに固有の他の問題を解決することもできます。メモリリソースとデータ通信機能の維持に役立ち、複雑なアプリケーションを単純なジョブに分割できます。

Keil では、さまざまなマイクロコントローラアーキテクチャのために複数の異なる RTOS システムをご用意しています。

- **RTX51 Tiny** と **RTX166 Tiny**（8051 および C166/XE166/XC2000 用）は非プリエンプティブ RTOS で、RAM が制限されているデバイスでもよく機能する特殊なスタックスワッピング技術が使用されています。
- **RTX**（ARM7/ARM9 および Cortex-Mx 用）と **ARTX166**（C166/XE166/XC2000 用）はプリエンプティブ RTOS カーネルでタスク優先レベルがあります。これらのカーネルは、ISR でのメッセージの受け渡しを処理し、スレッドセーフなメモリブロック割り当てを伴う実行時間が確定的な関数を実装します。ISR により、データがメッセージバッファに収集され、その後に複雑なデータ処理を実行する優先度の高いタスクにメッセージが送信されます。ISR は短くシンプルな構造を保ちます。

RTX の概要

多くのマイクロコントローラアプリケーションでは、複数のジョブまたはタスクを同時実行する必要があります。このようなアプリケーションの場合、RTOS を使用してシステムリソース（CPU、メモリなど）をいくつかのタスクに柔軟にスケジューリングできます。

RTX では、標準 C を使用してプログラムを記述してコンパイルします。タスク ID と優先度を指定するには、標準 C を少し変更するだけです。RTX-166 プログラムには、RTX166.H または RTX166T.H ヘッダファイルも含める必要があります。ARM デバイスでは RTX_CONFIG.C が必要です。［**Options for Target**］→［**Target**］ダイアログでオペレーティングシステムを選択することにより、µVision に含まれているリンカ L166 は適切な RTX-166 ライブラリファイルにリンクします。

単一タスクプログラム

標準 C プログラムは *main* 関数を使用して実行を開始します。組み込みアプリケーションで、*main* 関数は、通常、無限ループとしてコーディングされ、連続して実行される 1 つのタスクと見なされます。以下に例を示します。

```
int counter;

main (void)  {
  counter = 0;

  while (1)  {                                  // repeat forever
    counter++;                                  // increment counter
  }
}
```

ラウンドロビンタスク切り替え

ラウンドロビンタスク切り替えを使用すると、複数のタスクを疑似並列、同時実行できます。各タスクが実行される期間は事前に定義されています。タイムアウトになるとタスクの実行が中断され、別のタスクが開始されます。以下の例では、このラウンドロビンタスク切り替え技術を使用しています。

プログラムは、RTOS タスク関数 *job0* で実行を開始します。RTX 関数 *os_tsk_create* は *job1* を実行準備完了とマークします。タスク関数 *job0* および *job1* は単純なカウントループです。タイムスロットがなくなると、RTX は *job0* の実行を中断して *job1* の実行を開始します。そのタイムスロットがなくなると、*job0* の実行が継続されます。

ラウンドロビンタスク切り替えを使用した簡単な **RTX** プログラム

```
int counter0;
int counter1;

__task[1] void job0 (void)  {
  os_tsk_create (job1, 1);                  // start job 1

  while (1)  {                               // endless loop
    counter0++;                              // Increment counter 0
  }
}

__task void job1 (void)  {
  while (1)  {                               // Endless loop
    counter1++;                              // Increment counter 1
  }
}

main (void)  {                               // the main function
  os_sys_init (job0);                        // starts only job 0
}
```

Wait 関数

RTX カーネルには、現在のタスク関数の実行を中断し、指定されたイベントを待機する *wait*[2] 関数があります。この間、タスクはイベントを待機し、CPU は他のタスク関数を実行できます。

時間遅延の待機

RTX は、マイクロコントローラデバイスのハードウェアタイマを使用して、定期的な割り込み（タイマ Tick）を生成します。最も単純なイベントは、指定された数のタイマ Tick の間、現在実行中のタスクが割り込まれることによる時間遅延です。

以下のプログラムは前の例と似ていますが、*counter0* がインクリメントされた後に *job0* が *os_dly_wait* で中断される点が異なります。RTX は、*job0* が再度実行できるようになるまで、3 タイマ Tick の間、待機します。この間に *job1* が実行されます。また、この関数は 5 Tick の時間遅延で *os_dly_wait* も呼び出します。結果として、*counter0* は 3 Tick ごとにインクリメントされ、*counter1* は 5 タイマ Tick ごとにインクリメントされます。

---

[1] *For non-ARM devices the syntax is: void job0 (void) __task {…}.*

[2] *RTX Tiny では、時間遅延は関数 os_wait (K_TMO, …) で作成されます。*

時間遅延の待機のあるプログラム

```
int counter0;
int counter1;

__task void job0 (void)  {
  os_tsk_create (job1, 1);                // start job 1

  while (1)  {
    counter0++;                            // Increment counter 0
    os_dly_wait (3);                       // Wait 3 timer ticks
  }
}

__task void job1 (void)  {
  while (1)  {
    counter1++;                            // Increment counter 1
    os_dly_wait (5);                       // Wait 5 timer ticks
  }
}
```

イベントフラグの待機

イベントフラグはタスク調整に使用されます。つまり、イベントフラグを生成する別のタスクを待機しているタスクに使用されます。イベントフラグが以前に設定されていた場合、タスクは実行を継続します。

イベントフラグの待機のあるプログラム
(*job1* は *job0* を待機し、*i0* のオーバーフローをカウントします)

```
long i0, save_i0, i1;
OS_TID id1;                              // task ID for event transmits

__task void job0 (void)  {
  id1 = os_tsk_create (job1, 1);        // start job 1

  while (1)  {
    i0++;
    if (i0 > 1000000)  {                  // when i1 reaches 1000000
      i0 = 0;                             // clear i1
      os_evt_set (1, id1);                // set event '1' on job1
    }
  }
}

__task void job1 (void)  {
  while (1)  {
    os_evt_wait_or (1, 0xffff);           // wait for event '1'
    save_i0 = i0;                         // save value of i0
    i1;                                   // count events in i1
  }
}
```

プリエンプティブタスク切り替え

タスクが同じ優先度を持つ場合[1]（上記の例）、他のタスクを実行するにはラウンドロビンタイムアウト、または RTX *wait* 関数の明示的な呼び出しが必要です。そのため、上記の例では、予想されるように *save_i0* の値はゼロではありません。*job1* のタスク優先度が *job0* よりも高い場合、*job1* の実行は直ちに開始され、*save_i0* の値はゼロになります。*job1* は *job0* の実行に対してプリエンプティブになります（このタスク切り替えは高速で行われ数ミリ秒しかかかりません）。

高いタスク優先度を持つ *job1* の開始

```
   :
__task void job0 (void)  {
  id1 = os_tsk_create (job1, 2);          // start job 1 with priority 2
   :
   :
}
```

[1] *RTX Tiny にはタスク優先度がありません。代わりに、RTX Tiny にはシグナルと呼ばれるイベントフラグがタスクごとに 1 つあり、関数 os_wait (K_SIG, …) を使用してこのシグナルフラグを待機します。*

メールボックス通信

メールボックスは、タスク関数間でメッセージを転送するための FIFO（先入れ先出し）バッファです。メールボックス関数はポインタ値を受け取り、通常はメモリバッファを参照します。ただし、適切な型のキャストを使用することにより、任意の 32 ビットデータ型整数を渡すことができます。

メールボックス通信のあるプログラム[1]

```
os_mbx_declare(v_mail, 20);                // mailbox with 20 entries

__task void job0 (void)  {
  int i, res;

  os_mbx_init (v_mail, sizeof (v_mail));   // create mailbox first
  os_tsk_create (job1, 2);                 // before waiting tasks

  for (i = 0; i < 30; )  {                  // send 30 mail
    res = os_mbx_send (v_mail, (void *) i, 1000);
    if (res == OS_R_OK)  i++;              // check that mail send OK
  }
  os_tsk_delete_self ();                   // when done delete own task
}

__task void job1 (void)  {
  int v, res;

  while (1)  {
    res = os_mbx_wait (v_mail, (void **) &v, 0xFFFF);  // receive mail
    if (res == OS_R_OK || s == OS_R_MBX)  {            // check status
      printf ("\nReceived v=%d res=%d", v, res);       // use when correct
    }
  }
}
```

タスク *job0* はメールボックスを使用して *job1* に情報を送信します。*job1* を *job0* よりも高い優先度で実行した場合、メールは直ちに配信されます。*job1* を *job0* 以下の優先度で実行した場合、メールボックスでは最大 20 のメッセージがバッファに格納されます。

os_mbx_send および *os_mbx_wait* 関数はタイムアウト値を指定し、タイムアウト時間内にメールを配信できない場合は関数を終了します。

[1] メールボックスを使用する優先度の高いタスクを作成する場合は、優先度の高いタスクによって使用される前に、メールボックスを初期化します。

セマフォ

セマフォを利用し、アプリケーション内でタスクを同期します。セマフォには、オペレーティングシステムに対する単純な呼び出しセットがあり、競合状態を防ぐ標準的なソリューションです。ただし、リソースデッドロックは解決されません。RTX では、セマフォで使用するアトミック操作には割り込めません。

バイナリセマフォ

セマフォの最も単純な使用事例は、2 つのタスクの同期です。

```
os_sem semA;                                  // declare the semaphore

__task void job0 (void)  {
  os_sem_init(semA, 0);

  while(1) {
     do_func_A();
     os_sem_send(semA);                       // free the semaphore
  }
}

__task void job1 (void)  {

  while(1) {
     os_sem_wait(semA, 0xFFFF);               // wait for the semaphore
     do_func_B();
  }
}
```

この例では、セマフォを使用して *do_func_B()* を実行する前に *do_func_A()* を実行します。

セマフォのカウント（マルチプレックス）

マルチプレックスを使用して、重要なコードセクションにアクセスできるタスクの数を制限します。例えば、メモリリソースにアクセスするルーチンでは、サポートされる呼び出しの数が制限されます。

```
os_sem mplxSema;                            // declare the semaphore

__task void job0 (void)  {
  os_sem_init (mplxSema, 5);                 // init semaphore with 5 tokens

  while(1) {
     os_sem_wait (mplxSema);                 // acquire a token
     processBuffer();
     os_sem_send (mplxSema);                 // free the token
  }
}
```

この例では、5 つのトークンと共にマルチプレックスセマフォを初期化します。タスクは *processBuffer()* を呼び出す前に、セマフォトークンを取得する必要があります。関数は完了すると、セマフォにトークンを返します。*processBuffer()* を 5 回以上呼び出す場合、6 回目の呼び出しは 5 つの実行タスクのいずれかがトークンを返すまで待機します。このため、マルチプレックスセマフォでは *processBuffer()* を同時に使用できる呼び出しは 5 つまでです。

割り込みサービスルーチン

割り込みはハードウェアまたはソフトウェアからの非同期信号であり、実行状態を保存するようマイクロコントローラに指示します。割り込みは、割り込みハンドラにコンテキストスイッチをトリガします。ソフトウェア割り込みはマイクロコントローラの命令セットに命令として実装され、ハードウェア割り込みと同様に動作します。割り込みは、以下のように分類されます。

- マスク可能割り込み（IRQ）－ビットマスクにビットを設定することにより無視できるハードウェア割り込み。
- マスク不可割り込み（NMI）－設定できず無視できないハードウェア割り込み。
- ソフトウェア割り込み－命令を実行することによりプロセッサ内に生成されます。

RTX では、割り込みが正しく実行され、マシンを定義された状態にします。割り込みサービスルーチンは、割り込みハンドラとも呼ばれ、システム呼び出し、システムタイマ、ディスク I/O、電源シグナル、キーストローク、ウォッチドッグなど、ハードウェアデバイスや操作モード間の遷移の処理に使用されます。他の割り込みは UART またはイーサネットを使用してデータを転送します。

RTX で割り込み関数を使用するためのヒント：

- ***os_*** で始まる関数は、タスクから呼び出すことはできるが、割り込みサービスルーチンから呼び出すことはできません。
- ***isr_*** で始まる関数は、**IRQ** 割り込みサービスルーチンから呼び出すことはできるが、タスクから呼び出すことはできません。**FIQ** からは使用しないで下さい。
- カーネルを開始する前に、***isr_*** 関数を呼び出す **IRQ** 割り込みを有効にしないで下さい。
- ARM7/ARM9 ターゲットで **IRQ** 関数をネストしないで下さい。
- シグナルおよびメッセージを RTOS タスクに送信するには、短い **IRQ** 関数を使用します。
- 非 RTX プロジェクトの場合と同様にアプリケーションに割り込み関数を追加します。
- デフォルトでは、割り込みは起動時に全体的に有効になっています。

もう 1 つの重要な概念が割り込みレイテンシです。割り込みが生成されてから処理されるまでの時間として定義されます。この概念はマシンをリアルタイムで制御する必要があるシステムで特に重要であり、低割り込みレイテンシが要求されます。RTX では、承認された最大時間内にサブルーチンの実行を終了し、割り込みレイテンシが事前に定義された最大時間を越えることはありません。

ISR の一般的なロジックは以下のコードサンプルのようになります。割り込み関数 *ext0_int* は *process_task* にイベントを送信して終了します。タスク *process_task* は外部割り込みイベントを処理します。この例の *process_task* は単純で、割り込みイベントの数をカウントするだけです。

```
#define   EVT_KEY 0x00001

OS_TID    pr_task;
int       num_ints;

__irg   void ext0_int (void)  {        // external interrupt routine

   isr_evt_set (EVT_KEY, pr_task);     // send event to 'process_task'
   acknYourInterrupt ();               // acknowledge interrupt;
}


__task  void process_task (void)  {
   num_ints =0;

   while(1)  {
      os_evt_wait_or (EVT_KEY, 0xFFFF);
      num_ints++;
   }
}


__task  void init_task (void)  {

   enableYourInterrupt ();
   pr_task = os_tsk_create (process_task, 100);   // create task with prio
   os_tsk_delete_self ();
}
```

オンラインヘルプのさまざまな例と追加情報を参照するには、**F1** キーを押します。

メモリとメモリプール

マイクロコントローラのアーキテクチャに関係なく、Keil 開発ツールに付属するコンパイラはすべてのメモリ領域にアクセスできます。変数は、メモリタイプを宣言に含めることにより特定のメモリ空間に明示的に割り当てたり、メモリモデルに基づいて暗黙的に割り当てることができます。レジスタに配置できない関数の引数およびアトミック変数も、デフォルトのメモリ領域に格納されます。内部データメモリへのアクセスは、外部内部データメモリへのアクセスよりも大幅に速くなります。可能な場合は、使用頻度の高い変数を内部メモリ空間に格納し、使用頻度の低い変数を外部メモリ空間に格納します。

RTX では、サイズが固定されたメモリプール用にスレッドセーフで再入可能な[1] 割り当て関数が用意されています。これらの関数は実行時間が確定的で、プールの使用状況の影響を受けません。組み込みメモリ割り当てルーチンを使用すると、メモリプールを作成することによりシステムメモリを動的に使用したり、メモリプールから固定サイズブロックを使用することもできます。メモリプールは、オブジェクトのサイズに合わせて適切に初期化する必要があります。

[1] 可変長メモリ割り当て関数は再入不可です。malloc() および free() の実行時に tsk_lock() と tsk_unlock() を使用してシステムタイマ割り込みを無効または有効にして下さい。

```
#include <rtl.h>

os_mbx_declare (MsgBox, 16);                // declare an RTX mailbox

U32 mpool [16*( 2 * sizeof (U32) ) /4 + 3]; // memory for 16 messages

__task  void rec_task (void);               // task to receive a message

__task  void send_task (void)  {            // Task to send a message
   U32 *mptr;

   os_tsk_create (rec_task, 0);
   os_mbx_init (MsgBox, sizeof (MsgBox));    // init mailbox

   mptr = _alloc_box (mpool);             // alloc. memory for the message

   mptr[0] = 0x3215fedc;                  // set message content
   mptr[1] = 0x00000015;

   os_mbx_send (MsgBox, mptr, 0xffff);    // Send the message to 'MsgBox'

   os_tsk_delete_self ();
}

__task  void rec_task (void)  {
   U32 *rptr, rec_val[2];

   os_mbx_wait (MsgBox, &rptr, 0xffff);    // Wait for message
   rec_val[0] = rptr[0];                    // Store content to 'rec_val'
   rec_val[1] = rptr[1];

   _free_box (mpool, rptr);                // Release the memory block

   os_tsk_delete_self ();
}

void main (void)  {

   _init_box (mpool, sizeof (mpool), sizeof (U32));
   os_sys_init (send_task);
}
```

可変サイズのメッセージオブジェクトを送信し、可変サイズのメモリブロックを使用するには、メモリ割り当て関数を使用する必要があります。これらの関数は **stdlib.h** にあります。

RTX および ARTX166 関数の概要

関数グループ	RTX	ARTX166
タスク管理	create-task、delete-task、pass-task、change-priority、running-task-id、running-task-priority、lock-task、unlock-task、system-init、system-priority	create-task、delete-task、pass-task、change-priority、running-task-id、running-task-priority、lock-task、unlock-task、system-init、define-task
イベント/シグナル関数	clear-event、get-event、set-event、wait-event、isr-set-event	clear-event、get-event、set-event、wait-event、isr-set-event
セマフォ関数	initialize -semaphore、send-semaphore、wait-semaphore、isr-send-semaphore	initialize -semaphore、send-semaphore、wait-semaphore、isr-send-semaphore
メールボックス関数	check-mbx、declare-mbx、initialize -mbx、send-mbx、wait-mbx、isr-receive-mbx、isr-send-mbx isr-check-mbx	check-mbx、declare-mbx、initialize -mbx、send-mbx、wait-mbx、isr-receive-mbx、isr-send-mbx
メモリ管理	create-pool、check-pool、get-block、free-block	
ミューテックス管理	initialize-mutex、release-mutex、wait-mutex	initialize-mutex、release-mutex、wait-mutex
システムクロック (タイマ Tick)	delay-task、wake-up-task、set-slice、create-timer、kill-timer、call-timer	delay-task、wake-up-task、set-slice、create-timer、kill-timer、call-timer
一般的な WAIT 関数	interval-wait	interval-wait

RTX および ARTX166 の技術データ

技術データ	RTX		ARTX166
最大タスク数	250		250
イベント/シグナル	タスクあたり 16		タスクあたり 16
セマフォ、メールボックス、ミューテックス	無制限		無制限
最小 RAM	2 ～ 3 キロバイト		500 バイト
	ARM7/ARM9	**Cortex-Mx**	
最大コードスペース	4.2 キロバイト	4.0 キロバイト	4.0 キロバイト
ハードウェア要件	1 オンチップタイマ	SysTick タイマ	1 オンチップタイマ
タスク優先度	1 - 254	1 - 254	1-127
コンテキストスイッチ	< 7 µsec @ 60 MHz	< 4 µsec @ 72 MHz	< 15 µsec @ 20 MHz
割り込みロックアウト	3.1 µsec @ 60 MHz	RTX による無効不可	0.2 µsec @ 20 MHz

RTX51 Tiny および RTX166 Tiny 関数の概要

関数グループ	RTX51 Tiny	RTX166 Tiny
タスク管理	create-task、delete-task、running-task-id、switch-task、set-ready、isr-set-ready	create-task、delete-task、running-task-id、delay-task
シグナル関数	send-signal、clear-signal、isr-send-signal	send-signal、clear-signal、isr-send-signal、wait-signal
システムクロック（タイマ Tick)	reset-interval	delay-task
一般的な WAIT 関数	wait	wait

RTX51 Tiny および RTX166 Tiny の技術データ

技術データ	RTX51 Tiny	RTX166 Tiny
最大タスク数	16	32
シグナル	16	最大 32
RAM	7 + 3 バイト/タスク	8 + 4 バイト/タスク
最大コードスペース	900 バイト	1.5 キロバイト
ハードウェア要件	タイマなし	1 オンチップタイマ
コンテキストスイッチ	100 ～ 700 サイクル	400 ～ 4000 サイクル
割り込みロックアウト	< 20 サイクル	< 4 µsec、0 ws.

第 5 章 µVision の使用

µVision IDE を使用すると、ほとんどの開発者が組み込みシステムプログラムを容易に作成できます。本章では、一般的に使用される µVision の機能と、その使用方法について説明します。

概要と概念

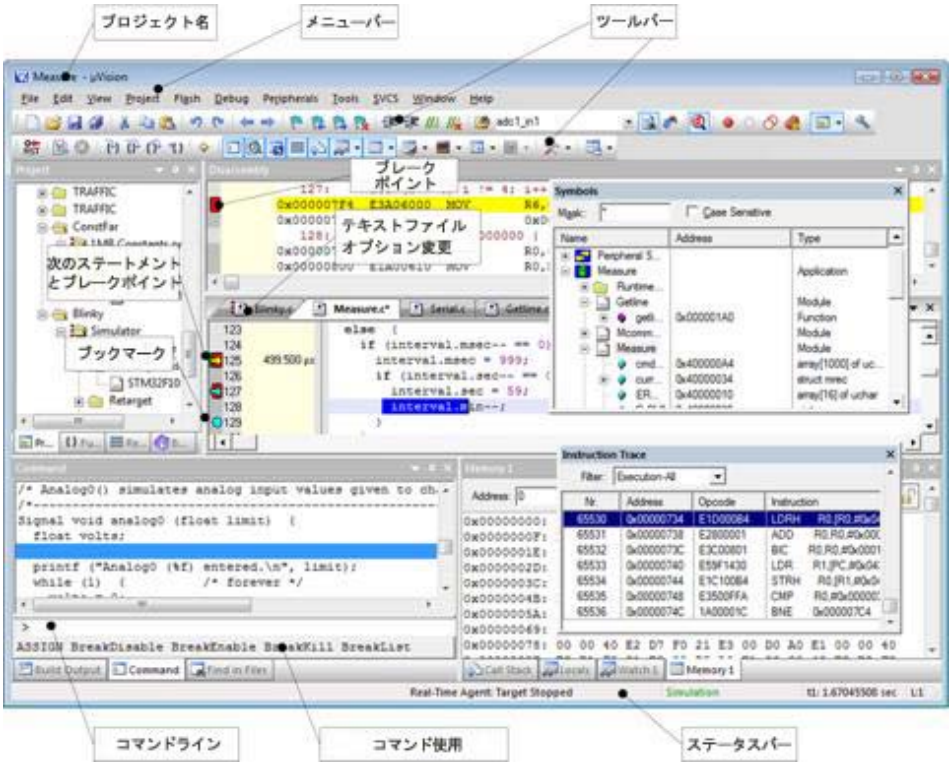
µVision の使用方法について説明する前に、多くの画面[1] や開発ツールの動作に共通する概要をご紹介します。お客様の日常業務をサポートするクラス最高の開発ツールのご提供に取り組む中で、汎用アプリケーションの外観を持つ µVision が構築されました。このアプローチによって µVision をすぐに使い始めることが可能となり、習熟に要する期間が短縮されます。

ウィンドウは、以下の概念に基づいています。

- µVision のウィンドウは、再編成したり、タイトルを付けたり、他の画面領域やウィンドウに個別に追加することもできます。
- ウィンドウ、オブジェクト、および変数をドラッグアンドドロップできます。
- 右マウスボタンを使用すると、大部分のオブジェクトでコンテキストメニューが表示されます。
- キーボードのショートカットを使用したり、独自のショートカットを定義したりできます。
- 最新のエディタのさまざまな機能を使用できます。
- 現在のコンテキストで使用できないメニュー項目やツールバーボタンはグレー表示になります。

[1] 次の章のスクリーンショットは、そのトピックの主な機能や特殊な機能を示すため、さまざまなサンプルプログラムとマイクロコントローラアーキテクチャから取得したものです。ウィンドウ、ダイアログ、タブカテゴリなどの外観が他のマイクロコントローラアーキテクチャとは多少異なる場合があります。

- 図示記号を使用して、オプションや保存されていない変更内容、プロジェクトに含まれていないオブジェクトを示します。
- ステータスバーにはコンテキスト駆動型の情報が表示されます。
- µVision は、サードパーティ製のツールに関連付けることができます。

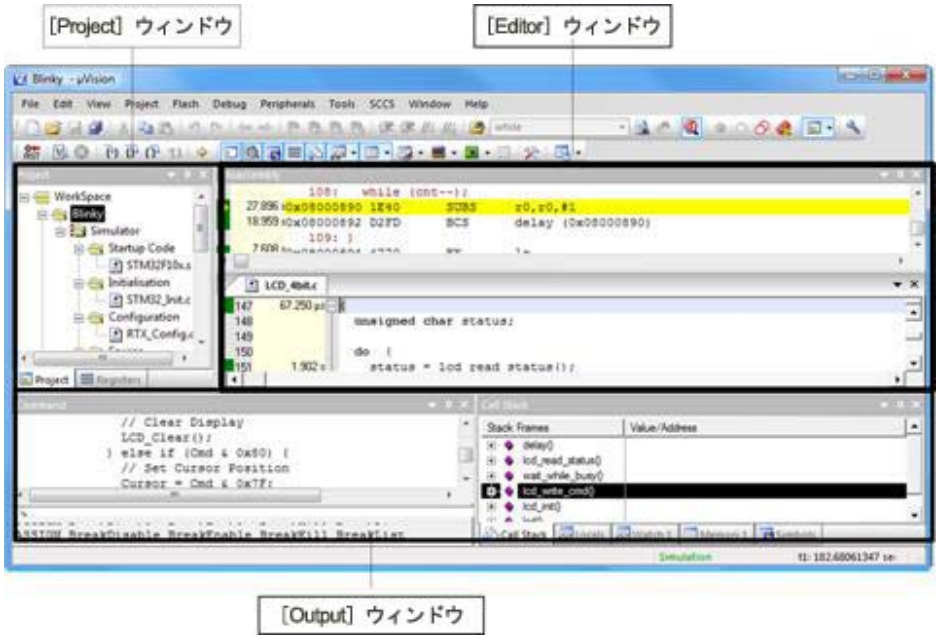


µVision を起動するには、デスクトップの µVision アイコンをクリックするか［スタート］メニューから［µVision］を選択します。

ウィンドウレイアウトの概念

µVision では作業環境[1] を自由に設定できますが、3 つの重要な画面領域を定義しておきます。後述のコメント、説明、および指示を理解しやすくするための定義です。

[1] どのウィンドウも µVision 画面の他の部分や外側、物理的に別の画面にも移動できますが、テキストエディタに関連するオブジェクトは例外です。



[Project] ウィンドウ領域は、デフォルトで［Project］ウィンドウ、［Functions］ウィンドウ、［Books］ウィンドウ、および［Registers］ウィンドウが表示される画面内の一部分です。

[Editor] ウィンドウ領域内では、ソースコードの変更、パフォーマンスと解析情報の表示、および逆アセンブリコードの確認を行うことができます。

[Output] ウィンドウ領域には、デバッグ、メモリ、シンボル、コールスタック、ローカル変数、コマンド、参照情報、およびファイル内での検索結果に関連する情報が表示されます。

何らかの理由で特定のウィンドウが表示されず、そのウィンドウの表示/非表示を何回か試しても表示されない場合は、**［Window］→［Reset Current Layout］**メニューを使用し、デフォルトのレイアウトで µVision を起動して下さい。

ウィンドウの位置設定

µVision のウィンドウは画面上のどの領域にも配置できます。µVision フレームの外側や物理的に別の画面に配置することもできます。

- 左マウスボタンでウィンドウの**タイトルバー**[1] をクリックして押したままにします。
- そのウィンドウを適切な領域やコントロール上にドラッグしてマウスボタンを離します。

ソースコードファイルは［**Editor**］**ウィンドウ**[2] の外側に移動できません。

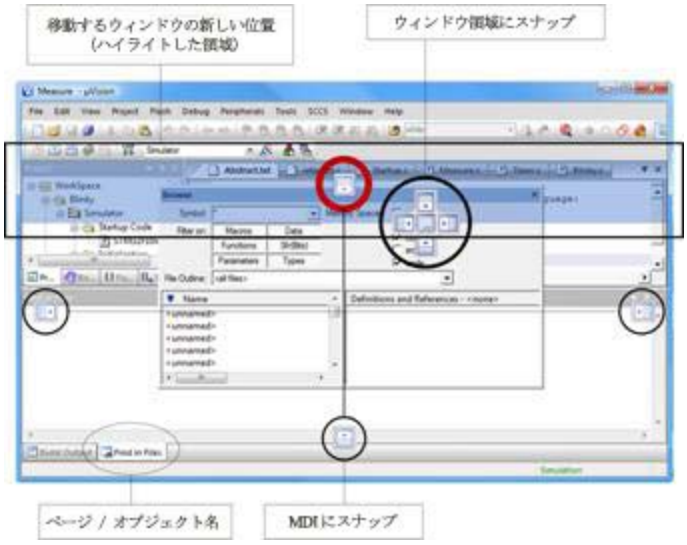
ウィンドウの**タイトルバー**の**コンテキストメニュー**を表示し、ウィンドウオブジェクトのドッキング属性を変更します。場合によっては、ウィンドウをドラッグアンドドロップする前にこの操作を実行する必要があります。

ドッキングヘルパコントロール[3] が表示されて、ウィンドウを貼り付ける領域が強調されます。新しいドッキング領域が、青色で強調表示されたセクションで示されます。適切なコントロールの上にマウスを移動し、ウィンドウをマルチドキュメントインタフェース（MDI）またはウィンドウ領域にスナップします。

[1] オブジェクトをドラッグアンドドロップするにはページ/オブジェクト名をクリックします。

[2] ソースコードファイルはテキストエディタのウィンドウ内に保持されます。

[3] コントロールは新しいウィンドウ位置の領域を示します。新しい位置が強調表示されます。



µVision のモード

µVision は、**ビルドモード**と**デバッグモード**の2 つのモードで動作します。画面設定、ツールバー設定、およびプロジェクトオプションは、モードのコンテキストに保存されます。**File ツールバー**はどちらのモードでも使用できます。**Debug ツールバー**と **Build ツールバー**は、それぞれのモードでのみ表示されます。ボタン、アイコン、およびメニューは、特定のモードに関連している場合に使用できます。

標準的な作業モードは**ビルドモード**です。このモードでは、アプリケーションの作成、プロジェクトの設定、環境設定、ターゲットハードウェアとデバイスの選択、プログラムのコンパイル、リンク付け、およびアセンブル、エラーの修正、アプリケーション全体に適用される一般的な設定を行います。

デバッグモードでは、一般的な一部のオプションの変更とソースコードファイルの編集もできますが、これらの変更内容は、**ビルドモード**に切り替えてアプリケーションを再ビルドするまで反映されません。デバック設定への変更は直ちに反映されます。

メニュー

ファイル操作、エディタ操作、プロジェクト保持、開発ツールの設定、プログラムのデバッグ、ウィンドウの選択と操作、オンラインヘルプなど、ほとんどの µVision コマンドは**メニュー**バーからアクセスできます。

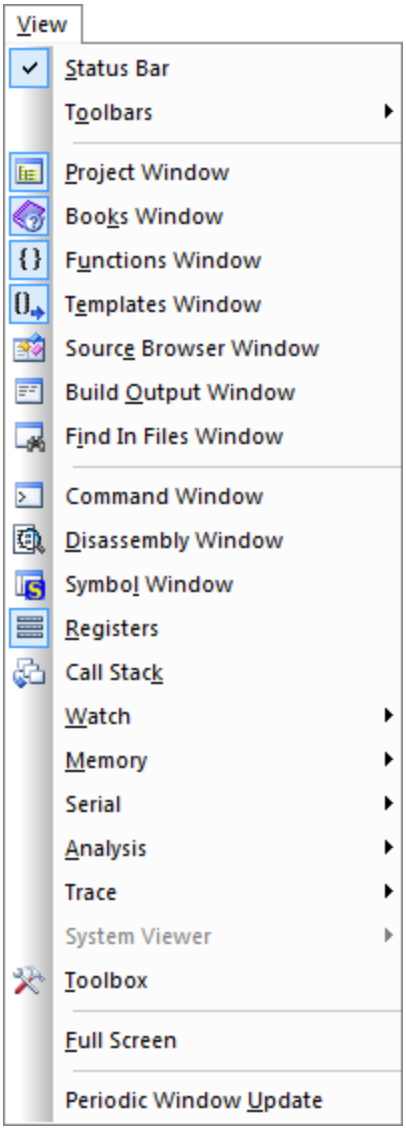
［File］メニュー

［**File**］メニューには、ソースファイルを開く、保存、印刷、および閉じるコマンドがあります。［**Device Database**］ダイアログと［**License Manager**］ダイアログへは、このメニューからアクセスします。

［Edit］メニュー

［**Edit**］メニューには、元に戻す、やり直し、切り取り、コピー、貼り付け、インデント、ブックマーク機能、さまざまな検索コマンドや置換コマンド、ソースアウトライン機能、高度なエディタ機能など、ソースコードを編集するためのコマンドがあります。エディタコンフィギュレーション設定にもこのメニューからアクセスします。

［View］メニュー



［**View**］メニューには、さまざまなウィンドウを表示または非表示にするためのコマンドがあります。**ステータスバー**も有効化/無効化できます。**デバッグモード**で画面を定期的に更新するには、［**Periodic Window Update**］オプションが役立ちます。このオプションを選択していない場合は、画面を**ツールボックス**から手動で更新できます。

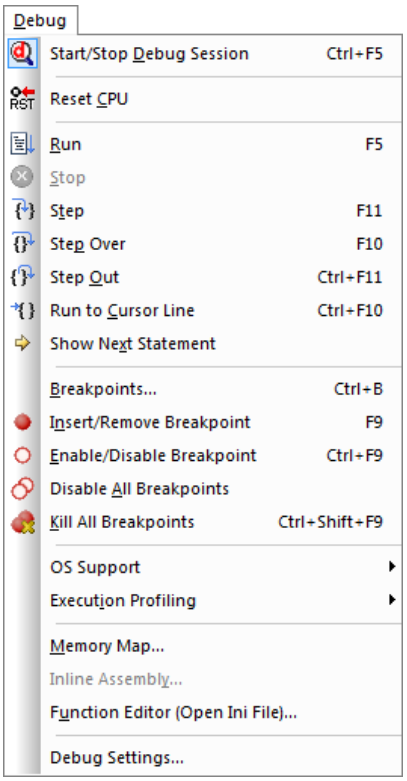
［Project］メニュー

［**Project**］メニューには、プロジェクトファイルを開く、保存、および閉じるコマンドがあります。プロジェクトを旧バージョンの µVision に**エクスポート**したり、プロジェクトコンポーネントの**管理**やプロジェクトの**作成**も実行できます。さらに、プロジェクト、グループ、およびファイルの**オプション**を設定できます。［**Multi-Project Workspace…**］メニューでは、マルチプロジェクトを管理できます。

［Flash］メニュー

［**Flash**］メニューには、組み込みターゲットシステムの Flash メモリを設定、消去、およびプログラミングするためのコマンドがあります。

［Debug］メニュー



［**Debug**］メニューには、デバックセッションの開始と停止、CPU のリセット、プログラムの実行と停止、および高レベルなアセンブリコードでのシングルステップを実行するコマンドがあります。さらに、ブレークポイントの管理、RTOS カーネル情報の表示、実行プロファイルの起動を行うコマンドを使用できます。メモリマップの変更、デバッガ関数および設定の管理もできます。

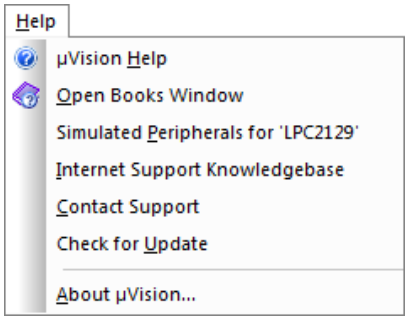
［Tools］メニュー

PC-Lint の設定および実行、サードパーティ製ユーティリティへの独自のツールショートカットの設定を行います。

［SVCS］メニュー

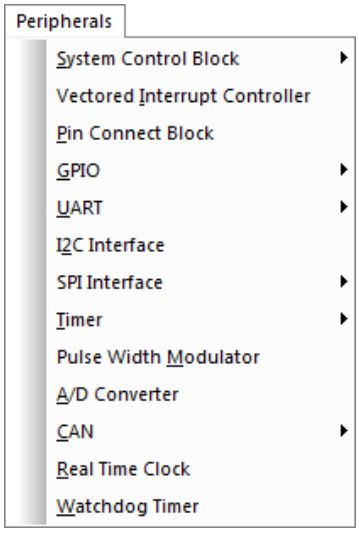
［**SVCS**］メニューでは、プロジェクト開発とサードパーティバージョンコントロールシステムの設定および統合を実行できます。

［Help］メニュー



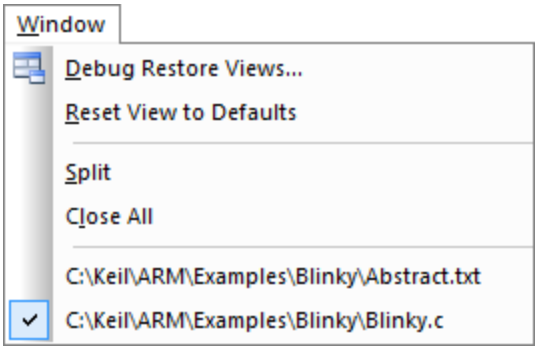
［**Help**］メニューには、オンラインヘルプシステムの起動、オンチップペリフェラル情報の表示、ナレッジベースへのアクセス、テクニカルサポートチームへの問い合わせ、製品アップデートの確認、製品のバージョン情報の表示を行うコマンドがあります。

［Peripherals］メニュー



［**Peripherals**］メニューは、オンチップペリフェラル設定を表示および変更するためのダイアログにアクセスできます。このメニューの内容は、アプリケーションで選択した特定のマイクロコントローラのオプションを表示するようにカスタマイズされます。通常、システムコンフィギュレーション、割り込み、UART、I2C、タイマ/カウンタ、汎用 I/O、CAN、パルス幅モジュレータ、リアルタイムクロック、およびウォッチドッグタイマのダイアログを使用できます。このメニューは**デバッグモード**でのみアクティブになります。

［Window］メニュー



［**Window**］メニューには、**テキストエディタ**でさまざまなウィンドウを分割、選択、および閉じるためのコマンドがあります。

さらに、［**Debug Restore Views…**］ダイアログでは独自の画面レイアウトを定義し、定義した複数の画面レイアウト間で切り替えることができます。

デフォルトのレイアウトは、［**Reset View to Defaults**］でいつでも復元できます。［**Window**］メニューの下部には、現在開いているソースコードのウィンドウがリストされます。

ツールバーとツールバーアイコン

µVision IDE には、頻繁に使用するコマンドのボタンを伴ったいくつかのツールバーが組み込まれています。

- **File ツールバー**には、ソースファイルの編集、µVision の設定、およびプロジェクト固有のオプションの設定に使用するコマンドに対応したボタンがあります。
- **Build ツールバー**には、プロジェクトをビルドする際に使用するコマンドに対応したボタンがあります。
- **Debug ツールバー**には、デバッガで使用するコマンドに対応したボタンがあります。

File ツールバーはいつでも使用できますが、**Build ツールバー**と **Debug ツールバー**はそれぞれのモードでのみ表示されます。**ビルドモード**と**デバッグモード** には、該当するツールバーの表示/非表示を切り替えるオプションがあります。

File ツールバー

［New File］– 空のテキストウィンドウを開きます。

［Open File］– 既存のファイルを開くダイアログを表示します。

［Save File］– 現在のファイルの内容を保存します。

［Save All］– 開いているすべてのファイルの変更内容を保存します。

［Cut］– 選択したテキストを削除し、クリップボードにコピーします。


［Copy］- 現在選択しているテキストをクリップボードにコピーします。


［Paste］- クリップボードのテキストを現在のカーソル位置に挿入します。


［Undo changes］- 編集ウィンドウでの以前の変更内容を削除します。


［Redo changes］- 取り消した直前の変更内容を復元します。


［Navigate Backwards］- カーソルを以前の後の位置に移動します。


［Navigate Forwards］- カーソルを以前の前の位置に移動します。


［Bookmark］- カーソル位置のブックマークを設定または削除します。


［Previous Bookmark］- 現在のカーソル位置より前にあるブックマークにカーソルを移動します。


［Next Bookmark］- 現在のカーソル位置より後にあるブックマークにカーソルを移動します。


［Clear All Bookmarks］- 現在のドキュメント内のブックマークを削除します。


［Indent］- 強調表示されているテキストの行をタブ 1 つ分右に移動します。


［Unindent］- 強調表示されているすべてのテキスト行をタブ 1 つ分左に移動します。


［Set Comment］- 選択されたコード/テキストをコメント行に変換します。


［Remove Comment］- 選択されたテキスト行をコード行に変換します。


［Find in Files］- ファイルでテキストを検索します。結果は別のウィンドウに表示されます。

［Find］– 現在のドキュメント内で指定されたテキストを検索します。

［Incremental Find］– 入力した式を検索します。

［Debug Session］– デバッグを開始または停止します。

［Breakpoint］– カーソル位置のブレークポイントを設定または削除します。

［Disable Breakpoint］– カーソル位置のブレークポイントを無効にします。

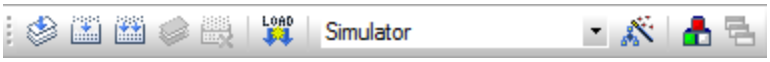
［Disable All Breakpoints］– すべてのドキュメント内のすべてのブレークポイントを無効にします。

［Kill All Breakpoints］– すべてのドキュメントからすべてのブレークポイントを削除します。

［Project Window］– プロジェクトに関連するウィンドウを有効または無効にするドロップダウンを表示します。

［Configure］– エディタ、ショートカット、キーワードなどを設定するダイアログを表示します。…

Build ツールバー



［Translate/Compile］– 現在の編集ウィンドウでファイルをコンパイルまたはアセンブルします。

［Build］– 変更が加えられたプロジェクトや依存関係が変更されたプロジェクトのファイルをビルドおよびリンクします。

［Rebuild］– プロジェクトのすべてのファイルを再度コンパイル、アセンブル、およびリンクします。

［Batch Build］– バッチ命令に基づいてアプリケーションを再ビルドします。この機能は、マルチプロジェクト環境でのみアクティブになります。

［Stop Build］- ビルドプロセスを停止します。

［Download］- アプリケーションをターゲットシステムフラッシュにダウンロードします。

［Target］ – ターゲットシステムを選択するためのドロップダウンボックスです（前述のツールバーの例では［Simulator］）。

［Target Options］- ツールおよびターゲットの設定を定義するダイアログを表示します。デバイス、ターゲット、コンパイラ、リンカ、アセンブラ、およびデバッグのオプションをここで設定します。フラッシュデバイスも設定できます。

［File Extensions, Environments, and Books］- ターゲット、グループ、デフォルトのフォルダ、ファイル拡張子、およびその他のブックを設定するダイアログを表示します。

［Manage Multi-Project Workspace］- 個々のプロジェクトやプログラムをマルチプロジェクトコンテナに対して追加または削除するダイアログを表示します。

Debug ツールバー

［Reset］- デバッグ中にマイクロコントローラ CPU またはシミュレータをリセットします。

［Run］- 次のブレークポイントまでターゲットプログラムの実行を継続します。

［Stop］- ターゲットプログラムの実行を停止します。

［Step One Line］- 次の命令またはプロシージャコールをステップ実行します。

［Step Over］- 1 つの命令およびプロシージャコールをステップオーバーします。


［Step Out］- 現在のプロシージャからステップアウトします。


［Run to Line］- 現在のカーソル行までプログラムを実行します。


［Show Current Statement］- 実行する次のステートメントを示します。


［Command Window］-［Command］ウィンドウの表示/非表示を切り替えます。


［Disassembly Window］-［Disassembly］ウィンドウの表示/非表示を切り替えます。


［Symbol Window］- シンボル、変数、ポートの表示/非表示を切り替えます。…


［Register Window］- レジスタの表示/非表示を切り替えます。


［Call Stack Window］- コールスタックツリーの表示/非表示を切り替えます。


［Watch Window］-［Locals］ウィンドウと［Watch］ウィンドウの表示/非表示を切り替えるドロップダウンを表示します。


［Memory Window］-［Memory］ウィンドウの表示/非表示を切り替えるドロップダウンを表示します。


［Serial Window］-［UART］ペリフェラルウィンドウと Debug printf() ビューの表示/非表示を切り替えるドロップダウンを表示します。


［Logic Analyzer］- 変数値をグラフィックで表示します。［Performance Analyzer and Code Coverage］ウィンドウの表示/非表示を切り替えるドロップダウンとしても使用します。


［Performance Analyzer］- モジュールおよび関数に要した時間と、関数の呼び出し回数をグラフィックで表示します。


［Code Coverage］-［Performance Analyzer］とは異なる方法でコード実行統計を示すダイアログを表示します。


［System Viewer］- ペリフェラルレジスタの値を表示します。


［Instruction Trace］-［Instruction Trace］ウィンドウの表示/非表示を切り替えます。


［Toolbox］-［Toolbox］ダイアログの表示/非表示を切り替えます。ターゲットシステムに応じて、さまざまなオプションを使用できます。


［Debug Restore Views］- デバッグ中に適切なウィンドウレイアウトを選択するダイアログを表示します。

その他のアイコン


［Print］- プリンタのダイアログを開きます。


［Books］- プロジェクトワークスペースに［Books］ウィンドウを開きます。


［Functions］- プロジェクトワークスペースに［Functions］ウィンドウを開きます。


［Templates］- プロジェクトワークスペースに［Templates］ウィンドウを開きます。


［Source Browser］- 出力ワークスペースに［Source Browser］ウィンドウを開きます。この機能を使用して、コード内で変数、関数、モジュール、およびマクロの定義またはそれ自体を検索します。


［µVision Help］– µVision ヘルプブラウザを開きます。


ファイル - ソースファイル。変更可能なファイルです。デフォルトオプションが使用されます。


ファイル - ソースファイル。変更可能なファイルです。ファイルオプションが変更されており、デフォルトのオプションとは異なっています。


ファイルまたはモジュール - ヘッダファイル。通常はプロジェクトに自動的に含まれます。これらのファイルタイプにはオプションを設定できません。


フォルダまたはグループ – 展開されています。展開されているフォルダまたはグループを示すアイコンです。オプションはデフォルト設定に対応します。


フォルダまたはグループ - 展開されています。展開されているフォルダまたはグループを示すアイコンです。オプションは変更されていて、デフォルト設定とは異なります。


フォルダまたはグループ - 折りたたまれています。オプションはデフォルト設定に対応します。


フォルダまたはグループ - 折りたたまれています。オプションは変更されていて、デフォルト設定とは異なります。


［Lock］– ウィンドウの内容を固定します。ウィンドウが定期的に更新されないようにします。ウィンドウの内容を手動で変更できません。


［Unlock］– ウィンドウの内容を固定解除します。ウィンドウが定期的に更新されるようにします。ウィンドウの内容を手動で変更できます。


［Insert］– 項目またはオブジェクトを作成したり、リストに追加したりします。


［Delete］– 項目またはオブジェクトをリストから削除します。

［Move Up］–リスト内の項目またはオブジェクトを上に移動します。


［Move Down］– リスト内の項目またはオブジェクトを下に移動します。


ペリフェラル SFR（ペリフェラルレジスタ、特殊関数レジスタ）


シミュレータ VTREG（仮想レジスタ）

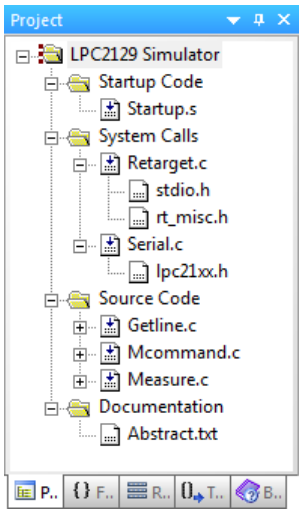

アプリケーション、コンテナ


変数


パラメータ


関数

［Project］ウィンドウ

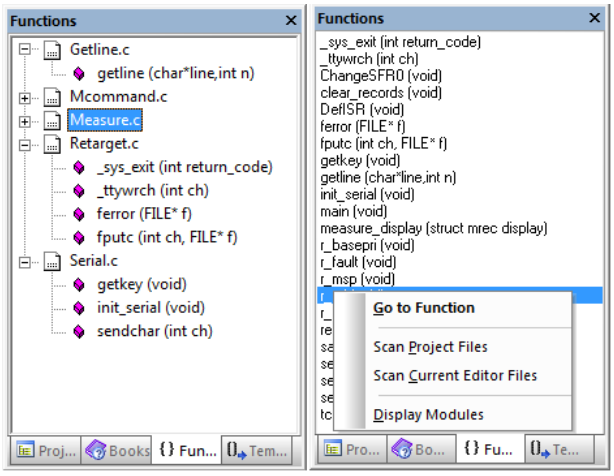


［Project］ウィンドウには、現在のプロジェクトに関する情報が表示されます。この領域の下部にあるタブからは、以下の項目にアクセスできます。

- **プロジェクト**構造と管理。ファイルをグループ化してプロジェクト概要をわかりやすくします。
- プロジェクトの**関数**。ソースコードの関数を簡単に見つけて移動できます。
- マイクロコントローラ**レジスタ**。デバッグ時にのみ使用できます。
- **Templates** for often-used text blocks. 定義をダブルクリックして、事前に定義されているテキストをカーソルの位置に挿入できます。
- 使用する µVision IDE およびプロジェクトに関する**ブック**。マイクロコントローラも含まれる場合があります。独自のブックを設定して任意のセクションに追加できます。

［**Functions**］ウィンドウには、プロジェクトまたは開いているエディタファイルのすべての関数が表示されます。

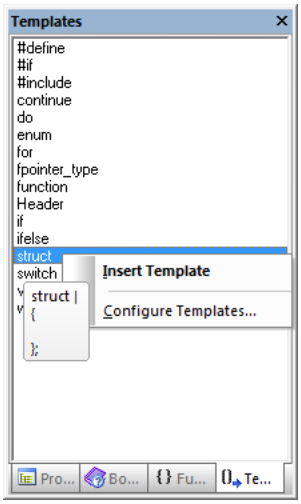
関数をダブルクリックすると、その関数の定義にジャンプします。このウィンドウの表示モードを切り替えたりファイルをスキャンするには、**コンテキストメニュー**を表示します。



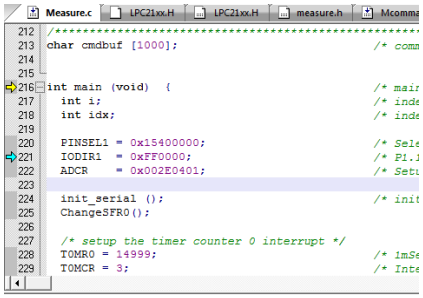
［**Configuration**］→［**Templates**］ダイアログから定義できる［**Templates**］ウィンドウには、ユーザ定義のテキストブロックが表示されます。

頻繁に使用する構文をコードファイルに挿入するには、定義をダブルクリックするか、**コンテキストメニュー**を表示します。

または、テンプレート名の先頭の数文字を入力してから **Ctrl+Space** キーを押してテキストを挿入します。

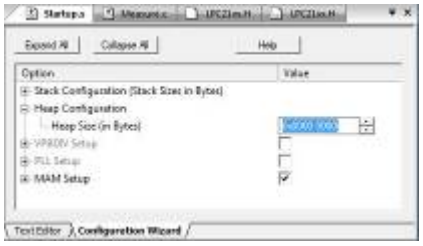


［Editor］ ウィンドウ



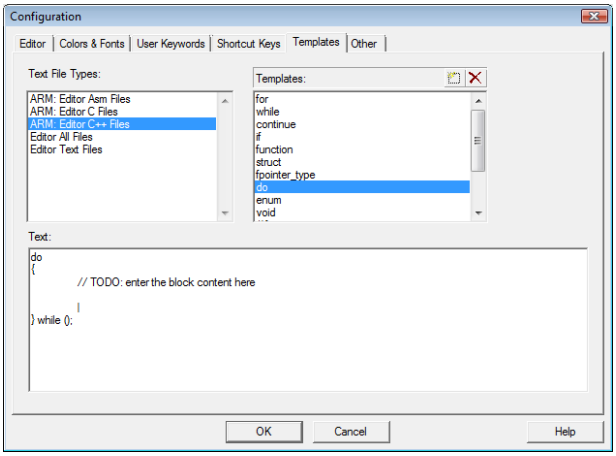
［**Editor**］**ウィンドウ**は以下の目的で使用します。

- ソースファイルの作成、編集、およびデバッグ。ヘルプを表示するには、言語要素で **F1** キーを押して下さい。
- ブレークポイントとブックマークの設定
- 強力なコンフィギュレーションウィザードを使用したプロジェクトオプションの設定とターゲットシステムの初期化
- デバッグ時の逆アセンブリコードとトレース命令の表示



通常、この領域内には、ソースコードファイルを表示した**テキストエディタ**、［**Disassembly**］**ウィンドウ**、**パフォーマンスアナライザ**、および**ロジックアナライザ**が表示されます。

エディタコンフィギュレーション



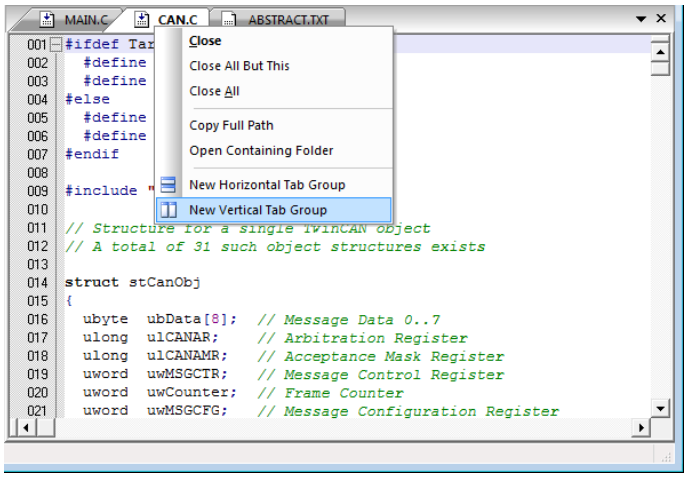
［**Configuration**］ダイアログから、エディタの設定、色とフォント、ユーザ定義のキーワード、ショートカットキー、およびテンプレートを設定します。

このダイアログは、［**Template**］ウィンドウの**コンテキストメニュー**、［**Edit**］→［**Configuration**］メニュー、または。

［**File Toolbar**］コマンドから表示できます。

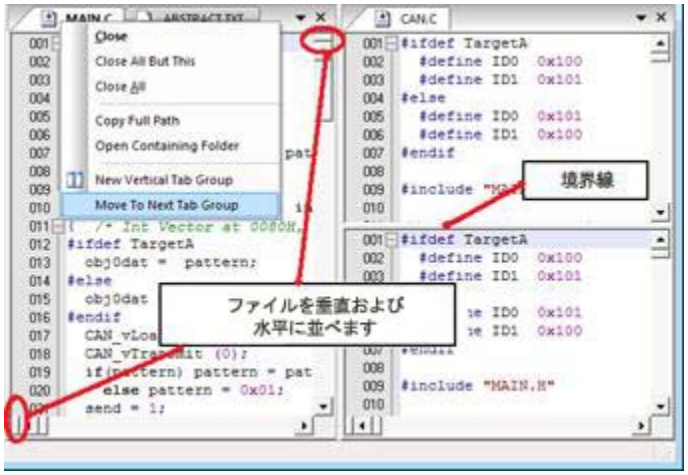
エディタの使用

エディタではテキストファイルを並べて表示できます。ファイルタブの**コンテキストメニュー**を表示して、水平と垂直のいずれの方向に並べるかを選択します。



ファイルは、ある**タブグループ**から別のタブグループにドラッグアンドドロップしたり、**コンテキストメニュー**から次のタブグループに移動することができます。

さらに、ファイルを垂直および水平に並べて表示できます。これらのフラグメントの任意の箇所でコードを完了します。



フラグメントを削除するには、境界線をダブルクリックします。

ファイルを閉じるには、ファイルのタブをダブルクリックします。

［Output］ ウィンドウ

［**Output**］**ウィンドウ**[1] はデフォルトで µVision 画面の下部に表示され、以下のウィンドウがあります。

- ［**Build Output**］**ウィンドウ**には、コンパイラ、アセンブラ、およびリンカのエラーと警告が表示されます。メッセージをダブルクリックすると、メッセージをトリガしたソースコードの位置にジャンプします。Press **F1** for on-line help.
- ［**Command**］**ウィンドウ**では、コマンドを入力してデバッガの応答を確認できます。ウィンドウの**ステータスバー**にヒントが表示されます。Press **F1** for on-line help.
- ［**Find in Files**］**ウィンドウ**では、結果をダブルクリックして、メッセージをトリガしたソースコードを検索できます。
- ［**Serial**］ウィンドウと［**UART**］ウィンドウには、ペリフェラルの I/O 情報が表示されます。
- ［**Call Stack**］**ウィンドウ**では、プログラムコールツリーを確認できます。
- ［**Locals**］**ウィンドウ**には、現在の関数のローカル変数に関する情報が表示されます。
- ［**Watch**］ウィンドウでは、トレースする一連の変数をカスタマイズできます。オブジェクト、構造、共用体、および配列を詳しく監視できます。
- ［**Symbols**］**ウィンドウ**は、オブジェクトの定義を検索するときに役立つオプションです。これらの項目を µVision の他の領域にドラッグアンドドロップできます。
- ［**Memory**］ウィンドウでは、メモリ領域の値を調べることができます。データを表示するには、適切なアドレスを定義します。
- ［**Source Browser**］**ウィンドウ**では、オブジェクトとその定義を簡単に検索できます。検索基準を入力し、出力を絞り込みます。

[1] ほとんどのオブジェクトは独自のウィンドウフレームに移動できるので、本書では「ページ」と「ウィンドウ」という語句が同じ意味で使用されています。

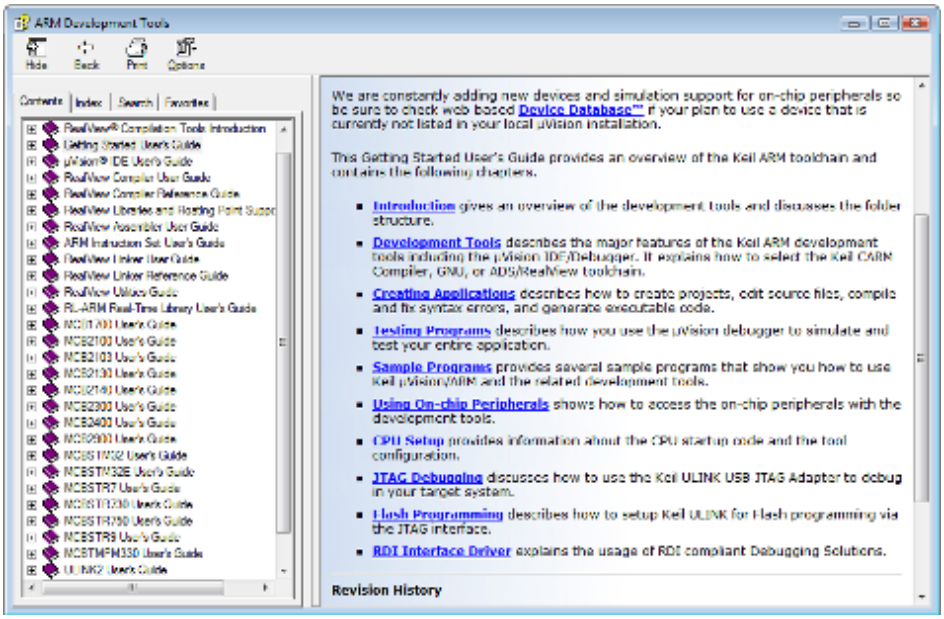
その他のウィンドウとダイアログ

ペリフェラルのダイアログとウィンドウ

ペリフェラルのダイアログとウィンドウでは、オンチップペリフェラルのステータスを確認および変更できます。これらのダイアログはプロジェクトの開始時に選択したターゲットシステムによって異なり、そのオプションもさまざまです。

オンラインヘルプ

µVision には、多数のページがあるオンラインマニュアルとコンテキスト依存ヘルプがあります。メインのヘルプシステムは、［**Help**］メニューから表示できます。



コンテキスト依存のオンラインヘルプは、µVision のほとんどのダイアログで使用できます。さらに、コンパイラディレクティブやライブラリルーチンのような言語要素のヘルプを表示するには、［**Editor**］**ウィンドウ**で **F1** キーを押します。デバッグコマンド、エラーメッセージ、および警告のヘルプについては、［**Output**］**ウィンドウ**で **F1** キーを使用して下さい。

第 6 章 組み込みプログラムの作成

µVision は、Keil マイクロコントローラ開発ツールだけでなく、複数のサードパーティ製ユーティリティも含む Windows アプリケーションです。µVision には、組み込みプログラムの作成をすぐに開始するために必要なものがすべて用意されています。

µVision には、高度なエディタ、プロジェクトマネージャ、および make ユーティリティが含まれています。これらが共に機能することによって、開発作業の軽減や操作方法の習得にかかる時間の短縮を実現し、組み込みアプリケーションの作成を迅速に開始できます。

新しい組み込みプロジェクトを作成するには、以下の作業を行います。

- プロジェクトファイルの作成
- ［Project］ウィンドウの使用
- ソースファイルの作成
- プロジェクトへのソースファイルの追加
- ターゲット、グループ、およびファイルの使用
- ターゲットオプション、グループオプション、およびファイルオプションの設定
- スタートアップコードの設定
- プロジェクトのビルド
- HEX ファイルの作成
- マルチプロジェクトの操作

このセクションでは、段階的なチュートリアルを通じて µVision IDE を使用した組み込みプロジェクトの作成方法を示します。

プロジェクトファイルの作成

新しい µVision プロジェクトは、以下の 3 つの手順のみで作成できます。

1. プロジェクトフォルダとプロジェクトのファイル名を選択します。
2. ターゲットのマイクロコントローラを選択します。
3. プロジェクトフォルダにスタートアップコードをコピーします。

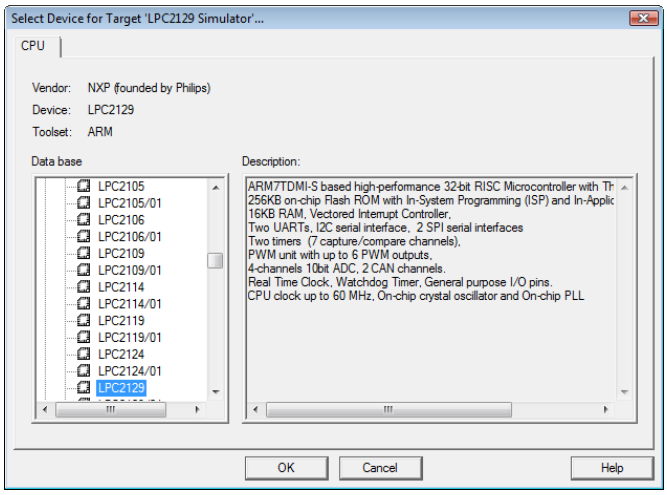
フォルダとプロジェクト名の選択

新しいプロジェクトファイルを作成するには、［**Project**］→［**New Project…**］メニューを選択します。これにより、標準的なダイアログが開き、新しいプロジェクトのファイル名を入力するよう求められます。プロジェクトごとに個別のフォルダを使用することをお勧めします。このダイアログの［**Create New Folder**］ボタンを押すと、新しい空のフォルダを作成できます。

適切なフォルダを選択し、新しいプロジェクトのファイル名を入力すると、新しい空のプロジェクトファイルが指定した名前で作成されます。このプロジェクトには、デフォルトのターゲットとファイルグループ名が含まれており、［**Project**］ウィンドウで表示できます。

ターゲットマイクロコントローラの選択

フォルダを選択し、プロジェクトのファイル名を指定すると、ターゲットマイクロコントローラを選択するよう求められます。µVision では、その特定のデバイスのツール設定、ペリフェラル、およびダイアログがカスタマイズされるので、この手順は非常に重要です。



［**Select Device**］[1] [2] ダイアログボックスに、µVision **デバイスデータベース**のすべてのデバイスがリストされます。

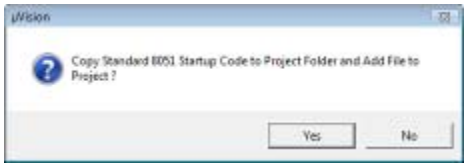
この画面を［**Project**］→［**Select Device for Target…**］メニューで呼び出して、後でターゲットを変更できます。

[1] 一部のデバイスでは、手動で入力する必要がある追加のパラメータがあります。デバイスコンフィギュレーションについて追加の手順が記載されている場合があるため、［Select Device］ダイアログのデバイスの説明をよくお読み下さい。

[2] 最終的に使用する実際のデバイスがわからなくても、µVision では、プロジェクトの作成後、ターゲットに合わせてデバイスの設定を変更できます。

スタートアップコードのコピー

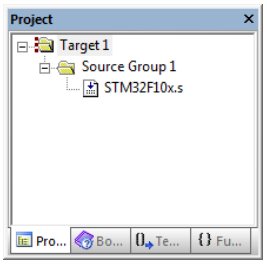
すべての組み込みプログラムには、何らかのマイクロコントローラ初期化またはスタートアップのコード[1] [2] が必要です。コードは、使用するツールチェーンとハードウェアによって異なります。また、ハードウェアの開始コンフィギュレーションを指定するために必要です。



すべての Keil ツールに、**デバイスデータベース**にリストされているほとんどのデバイスに対するチップ固有のスタートアップコードが含まれています。スタートアップコードはプロジェクトフォルダにコピーし、そこで修正します。µVision では、プロジェクトフォルダにスタートアップコードをコピーするかどうかを尋ねるダイアログが自動的に表示されます。この質問に対して［**YES**］を選択します。プロジェクトフォルダにスタートアップコードがコピーされ、プロジェクトにスタートアップファイルが追加されます。

スタートアップコードファイルは、コンフィギュレーションウィザードで使用される組み込みコメントと共に配布され、スタートアップコンフィギュレーションの GUI インタフェースが提供されます。

［Project］ウィンドウの使用

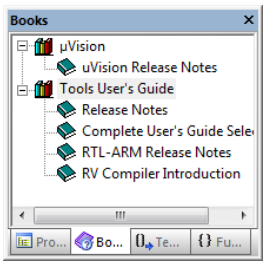


プロジェクトファイルを正常に作成すると、プロジェクトのターゲット、グループ、およびファイルが［**Project**］**ウィンドウ**に表示されます。デフォルトで、ターゲット名は**Target 1**に設定され、グループ名は**Source Group 1**になります。

[1] スタートアップコードのデフォルト設定は、ほとんどのシングルチップアプリケーションに適した開始点になります。ただし、スタートアップコードの変更が必要な場合もあります。

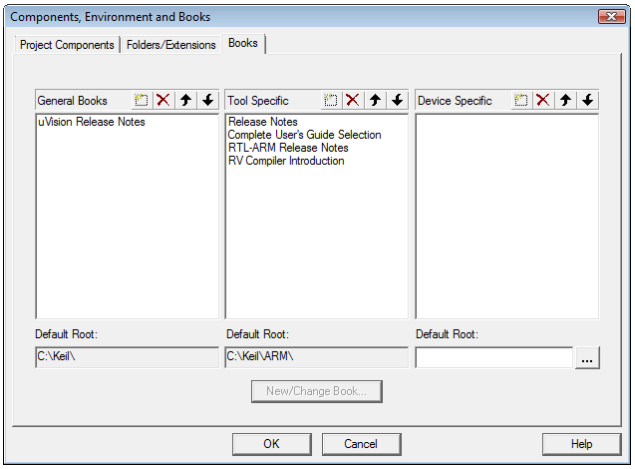
[2] ライブラリとアドオンプロジェクトには、スタートアップコードは必要ありません。

スタートアップコードを含んでいるファイルは、ソースグループに追加されます。スタートアップファイルを含むすべてのファイルを、他のグループに移動して後で定義できます。



［**Project**］ウィンドウの 1 つである［**Books**］ウィンドウにも、Keil 製品のマニュアル、データシート、および選択したマイクロコントローラのプログラマ用ガイドが表示されます。ブックをダブルクリックして開きます。

［**Books**］ウィンドウを右クリックして、**コンテキストメニュー**を開きます。［**Manage Books…**］を選択して［**Components, Environments and Books**］[1] ダイアログを開き、既存のマニュアルの設定を変更したり独自のマニュアルをブックのリストに追加したりします。



後で、プログラムの開発時には［**Functions**］ウィンドウと［**Templates**］ウィンドウも使用できます。

[1] ほとんどのマイクロコントローラのマニュアルは、ツールセットに含まれているか、Keil 開発ツール CD-ROM から入手できます。

ソースファイルの作成

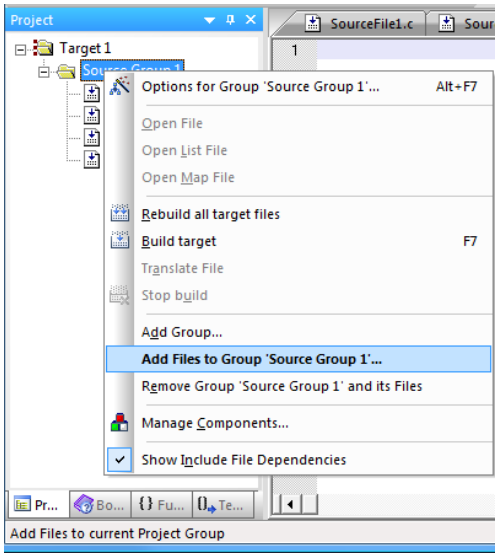
File ツールバーのボタンを使用するか、［**File**］→［**New…**］メニューを選択して、新しいソースファイルを作成します。

これにより、ソースコードを入力する空の［**Editor**］**ウィンドウ**が開きます。ファイルを保存すると、µVision では、ファイル拡張子に基づいて構文の色分け表示が有効になります。この機能をすぐに使用するには、コーディングを開始する前に空のファイルに目的の拡張子を付けて保存します。

File ツールバーのボタンまたは［**File**］→［**Save**］メニューを使用して、新しいソースファイルを保存します。

プロジェクトへのソースファイルの追加

ソースファイルを作成して保存したら、そのファイルをプロジェクトに追加します。ファイルは、プロジェクトフォルダに存在していても、現在のプロジェクト構造に含まれていなければ、コンパイルされません。



［**Project**］**ウィンドウ**でファイルグループを右クリックし、**コンテキストメニュー**から［**Add Files to Group**］を選択します。次に、追加するソースファイルを選択します。

ファイルを追加する手順がウィンドウにわかりやすく示されます。

ターゲット、グループ、およびファイルの使用

µVision の非常に柔軟なプロジェクト管理構造を使用すると、同じプロジェクトに複数の**ターゲット**を作成できます。

ターゲットとは、特定のプラットフォーム用に特定の方法でインクルードファイルをアセンブル、コンパイル、およびリンクするビルドオプションの定義済みのセットです。

複数のファイルグループをターゲットに追加し、複数のファイルを同じファイルグループに追加できます。

同じプロジェクトに対して**複数のターゲット**を定義することもできます。

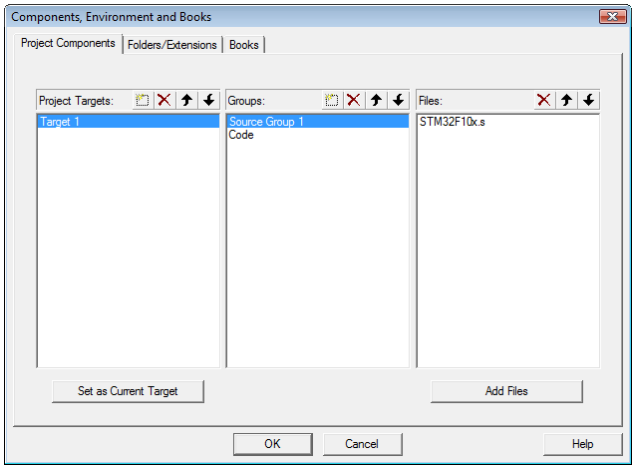
ターゲットやグループの名前は、アプリケーション構造や内部命名規則に合わせてカスタマイズする必要があります。マイクロコントローラコンフィギュレーションファイルには個別のファイルグループを作成することをお勧めします。

［**Components, Environment, and Books…**］ダイアログを使用して、ターゲット、グループ、およびファイルコンフィギュレーションを管理します。

ターゲット、グループ、またはファイルの名前を変更するには、以下のいずれかの操作を実行します。

- 目的の項目をダブルクリックするか、
- 項目を強調表示し、**F2** キーを押します。

名前を変更して［**OK**］ボタンをクリックします。変更内容は、このダイアログを閉じるとすぐに別のウィンドウで表示できます。



［Insert］- 新しいターゲットまたはグループを作成します。

［Delete］- ターゲット、グループ、またはソースファイルをプロジェクトから削除します。

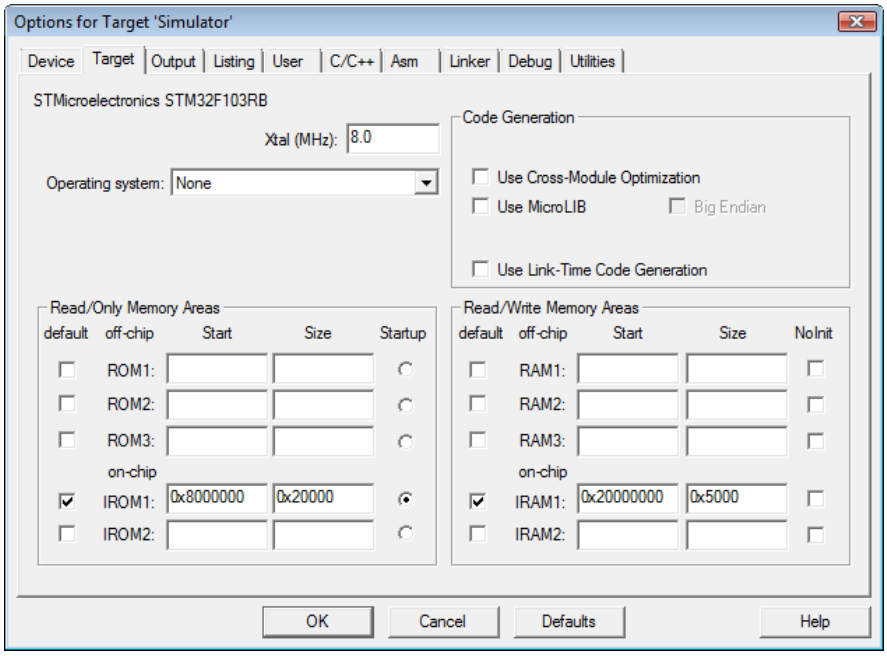
［Move Up］- ターゲット、グループ、またはソースファイルをリストの上位に移動します。

［Move Down］- ターゲット、グループ、またはソースファイルをリストの下位に移動します。

［Move Up］ボタンや［Move Down］ボタンを使用する代わりに、ソースファイルを［**Project**］ウィンドウ内にドラッグアンドドロップして、ファイルの順序を再配置できます。

ターゲットオプションの設定

Build ツールバーまたは［**Project**］メニューから［**Options for Target**］ダイアログを開きます。



このダイアログでは、以下の操作を実行できます。

- ターゲットデバイスの変更
- ターゲットオプションの設定
- 開発ツールとユーティリティの設定

通常、［**Target**］ダイアログと［**Output**］ダイアログのデフォルト設定は変更する必要はありません。

［**Options for Target**］ダイアログで使用できるオプションは、選択したマイクロコントローラデバイスによって異なります。使用できるタブとページは、選択したデバイスとターゲットに応じて変わります。

デバイスを切り替えた場合、メニューオプションは［**Device Selection**］ダイアログの［**OK**］ボタンをクリックするとすぐに使用できるようになります。

以下の表に、［**Target Options**］ダイアログの各ページで設定できるプロジェクトオプションを示します。

ダイアログページ	説明
Device	デバイスデータベースからターゲットデバイスを選択します。
Target	ターゲットシステムのハードウェア設定を指定します。
Output	生成する出力フォルダと出力ファイルを指定します。
Listing	生成するリストフォルダとリストファイルを指定します。
User	ビルドプロセスの前後でユーザプログラムを開始できます。
C/C++	プロジェクト全体の C/C++ コンパイラオプションを設定します。
Asm	プロジェクト全体のアセンブラオプションを設定します。
リンカ	プロジェクト全体のリンカオプションを設定します。リンカオプションでは通常、ターゲットシステムの物理メモリを設定し、メモリクラスとメモリセクションの位置を指定する必要があります。
Debug	ハードウェアやシミュレーションを使用するかどうかなどのデバッガオプションを設定します。
Utilities	Flash プログラミングのユーティリティを設定します。

グループオプションとファイルオプションの設定

µVision では、オブジェクトとオプションのプロパティをグループレベルで個別のファイルに設定できます。この優れた機能を使用して、デフォルト設定とは異なるコンフィギュレーションが必要なファイルとグループのオプションを設定できます。設定するには、［**Project**］ウィンドウを開きます。

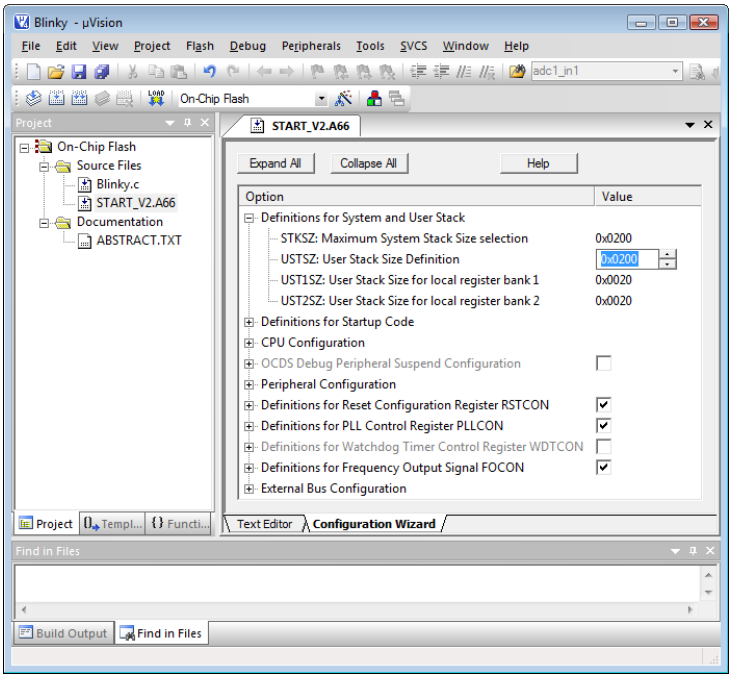
- ファイルグループの**コンテキストメニュー**を開いて［**Options for Group**］を選択し、そのファイルグループのプロパティ、コンパイラオプション、およびアセンブラオプションを指定します。
- ソースファイルの**コンテキストメニュー**を開いて［**Options for File**］を選択し、そのファイルのプロパティ、コンパイラオプション、またはアセンブラオプションを指定します。

［**Target**］オプションを一般的なオプションと同じように処理します。これらのオプションは、プロジェクト全体とそのターゲットに有効です。オプションによっては、グループレベルで個別のファイルに定義できます。ファイルレベルのオプションはグループレベルのオプションよりも優先され、同様に、ターゲットレベルで設定されたオプションよりも優先されます。

 アイコンの左側にある赤いドットは、その項目のオプションが一般的なターゲットオプションとは異なることを意味します。

スタートアップコードの設定

Keil ツールには、サポートされているデバイスの大部分についてチップ固有のスタートアップコードを含むファイルがあります。



Keil スタートアップファイルには、特定のターゲットシステムに合わせて調整できるオプションを持つアセンブラコードが含まれています。ほとんどのスタートアップファイルには、µVision の**コンフィギュレーションウィザード**の組み込みコマンドが含まれています。このウィザードには、スタートアップコードを編集するための直感的で使いやすいグラフィカルインタフェースが用意されています。

データを変更するには、目的の値をクリックするだけです。また、**テキストエディタ**を使用して、アセンブリソースファイルを直接編集することもできます。

Keil スタートアップファイルは、ほとんどのシングルチップアプリケーションに適した開始点になります。ただし、コンフィギュレーションをターゲットハードウェアに適合させる必要があります。マイクロコントローラ PLL クロックや BUS システムのように、ターゲット固有の設定は、手動で設定する必要があります。

プロジェクトのビルド

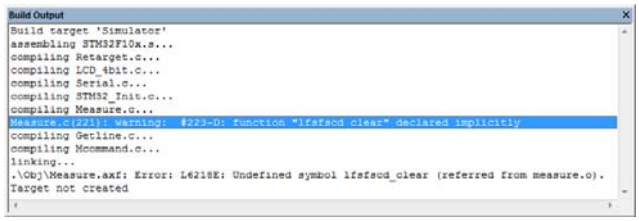
プロジェクト内のファイルをアセンブル、コンパイル、およびリンクするには、**Build ツールバー**または［**Project**］メニューから複数のコマンドを使用できます。以下の操作を実行する前に、ファイルが保存されます。

［Translate File］－現在アクティブなソースファイルをコンパイルまたはアセンブルします。

［Build Target］－変更されたファイルをコンパイルしてアセンブルし、プロジェクトにリンクします。

［Rebuild］－変更されたかどうかに関係なく、すべてのファイルをコンパイルしてアセンブルし、プロジェクトにリンクします。

アセンブル、コンパイル、およびリンク中は、［**Build Output**］**ウィンドウ**にエラーと警告が表示されます。



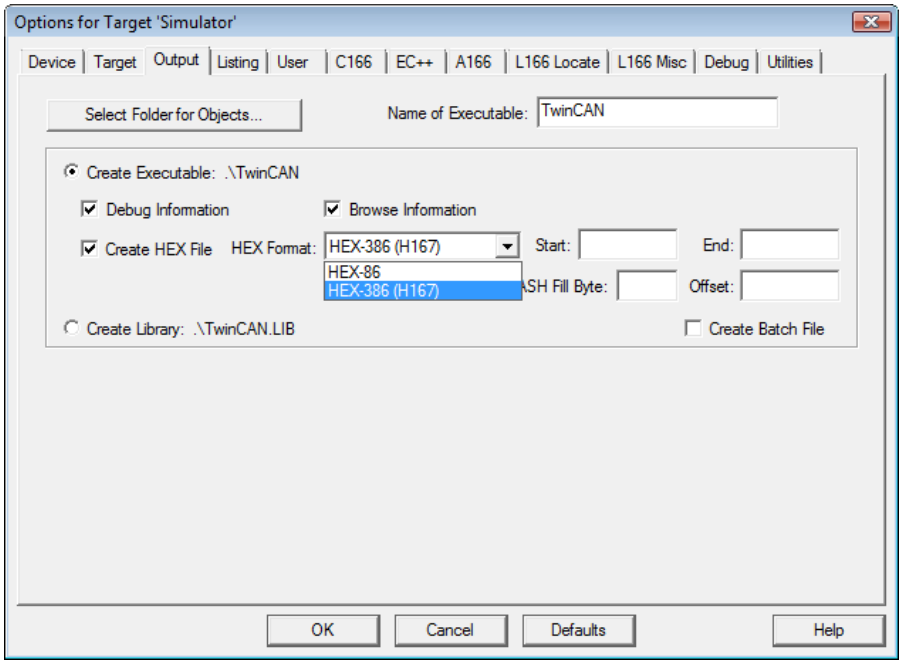
エラーまたは警告を強調表示して **F1** キーを押すと、特定のメッセージに関するヘルプを表示できます。メッセージをダブルクリックすると、エラーまたは警告の原因となったソース行にジャンプします。

ビルドプロセスが正常に行われた場合は、µVision に「**0 Error(s), 0 Warning(s)**」というメッセージが表示されます。プログラムが正常に実行されていても、警告があった場合は、時間の消費や望ましくない二次的な影響、プログラムに不必要なその他のアクションなど、悪影響を排除するために問題の解決を検討して下さい。

HEX ファイルの作成

［**Options for Target**］→［**Output**］を選択して［**Create HEX File**］ボックスをオンにすると、ビルドプロセス時に HEX ファイルが自動的に作成されます。

HEX 形式ファイルを生成するには、ドロップダウンコントロールで目的の HEX 形式を選択します。これらのファイルは、一部の Flash プログラミングユーティリティで必要です。



マルチプロジェクトの操作

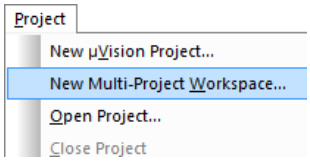
アプリケーション開発では、複数のプロジェクトを同時に操作する場合があります。シングルプロジェクトの場合は、現在のプロジェクトを閉じてから、新しいプロジェクトを開く必要があります。µVision のマルチプロジェクト機能を使用すると、プロジェクトのグループをマルチプロジェクトファイルとして定義し、それらのプロジェクトを 1 つの［Project］ウィンドウで操作できます。

論理的に依存し合っている µVision のプロジェクトを 1 つの**マルチプロジェクト**に組み合わせると、組み込みシステムアプリケーション設計の概要、一貫性、および透過性が高まります。µVision では、さまざまな独立したプロジェクトを 1 つのプロジェクト概要にグループ化できます。

シングルプロジェクト用に説明されている機能はすべてマルチプロジェクトにも適用されますが、その他の機能が必要であり、µVision IDE で使用できます。

マルチプロジェクトの作成

［Project］→［New Multi-Project Workspace…］を選択し、新しいマルチプロジェクトファイルを作成します。標準的な Windows ダイアログが開き、新しいプロジェクトファイル名の入力を求められます。



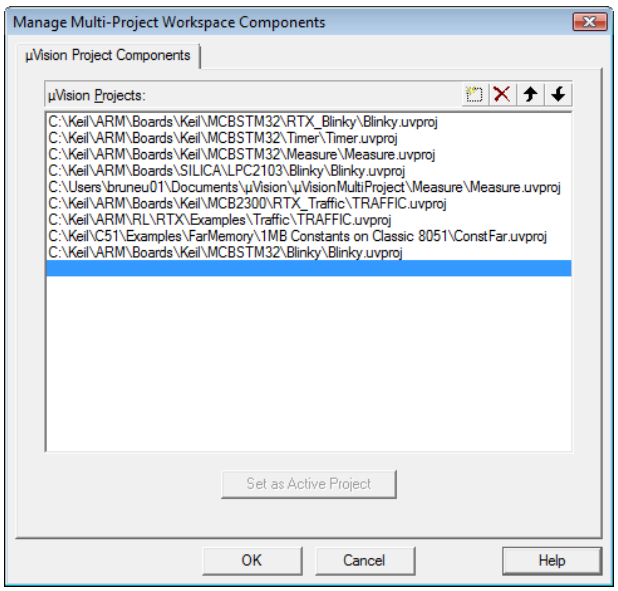
既存のマルチプロジェクトを開くには、**［Project］→［Open Project］**を選択します。ファイル拡張子によって、マルチプロジェクトファイルを独立したプロジェクトファイルと区別できます。マルチプロジェクトが含まれているファイルには、独立したプロジェクトの命名規則であるファイル名**.uvproj** ではなく、ファイル名**.uvmpv** の拡張子が付いています。

マルチプロジェクトの管理

［**Project**］→［**Manage − Multi Project Workspace…**］メニューから［**Manage Multi-Project Workspace Components**］ダイアログを開くか、**Build** ツールバーの［**Manage Multi-Project Workspace…**］ボタンを使用します。

［**Manage Multi-Project Workspace…**］– 個別のプロジェクトやプログラムをマルチプロジェクトに追加するダイアログです。

既存の独立したプロジェクト[1] [2] をマルチプロジェクトに追加します。ファイル順序の変更、プロジェクトファイルの追加または削除、アクティブプロジェクトの定義を行うには、コントロールを使用します。



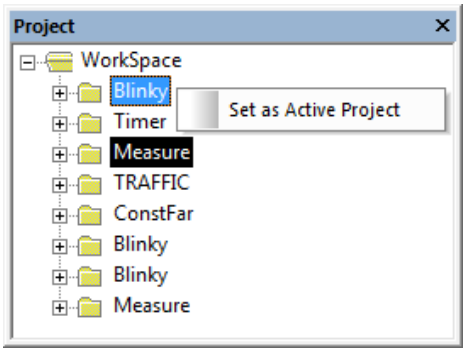
プロジェクトをこのリストから削除しても、プロジェクトファイル、つまりそれぞれのプロジェクトは保存場所から物理的には削除されません。

[1] 保持してマルチプロジェクトに追加できるのは既存のプロジェクトのみです。マルチプロジェクト環境で管理する前に、独立したプロジェクトを作成する必要があります。

[2] 別のフォルダに存在しているプロジェクトには、同じ名前を付けることができます。

マルチプロジェクトのアクティブ化

別のプロジェクトに切り替えるには、アクティブにするプロジェクト名を右クリックして、［**Set as Active Project**］をクリックします。



この例では、*Measure* が現在アクティブなプロジェクトですが、*Blinky* をアクティブなプロジェクトにしようとしています。

現在アクティブなプロジェクトを一意的に識別するため、µVision ではプロジェクト名が黒で強調表示されます。µVision IDE 内で実行されるすべての操作は、このプロジェクトにのみ適用されます。したがって、このプロジェクトは、独立したプロジェクトと同じ方法で処理できます。

マルチプロジェクトのバッチビルド

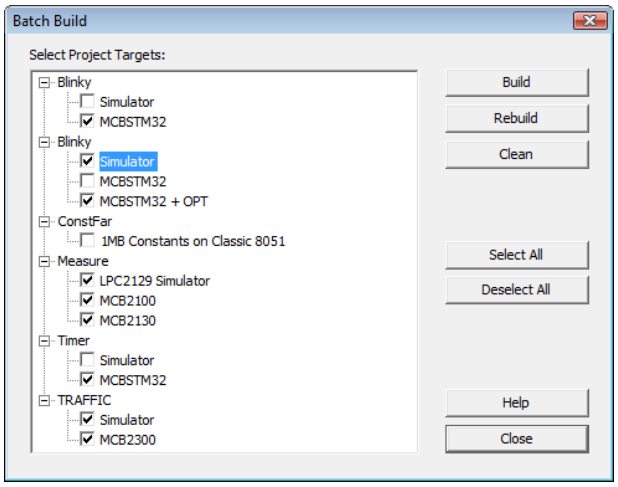
個々のプロジェクトは 1 つずつコンパイルできますが、マルチプロジェクト環境には、すべてのプロジェクトを 1 つの作業手順でコンパイルする便利な方法があります。

Build ツールバーまたは［**Project**］→［**Batch Build**］メニューから［**Batch Build**］[1] コマンドを使用して、プロジェクトターゲットをビルド、再ビルド、または削除します。

［Batch Build］– ターゲットおよび操作を選択できるウィンドウが開きます。

ビルド、再ビルド、または削除するプロジェクトと関連するターゲットのチェックボックスをオンにします。

それぞれのプロジェクトで指定された設定に基づいて、オブジェクトファイルが作成されます。共通のオブジェクトファイルがさらに作成されることはありません。



［**Build**］ボタンは、変更されたファイルをコンパイルしてアセンブルし、選択したターゲットにリンクします。

［**Rebuild**］ボタンは、すべてのファイルをコンパイルまたはアセンブルして、選択したターゲットにリンクします。

［**Clean**］ボタンは、選択したターゲットのオブジェクトファイルを削除します。

[1] Batch Build can be used in a Multi-Project setup only.

第 7 章 デバッグ

µVision デバッガは、シミュレータ[1] またはターゲットデバッガ[2] として設定できます。2 つのデバッグモードを切り替えたり、各モードを設定したりするには、［**Options for Target**］ダイアログの［**Debug**］タブに移動します。

シミュレータは、ターゲットハードウェアを必要とせずにマイクロコントローラのほとんどの機能をシミュレートするソフトウェアのみの製品です。シミュレータを使用すると、ターゲットハードウェアや評価ボードが使用できるようになる前に、組み込みアプリケーションをテストしてデバッグできます。また、µVision では、シリアルポート、外付け I/O、タイマ、割り込みなどのさまざまなペリフェラルをシミュレートします。ペリフェラルシミュレーション機能は、選択したデバイスによって異なります。

ターゲットデバッガは、ターゲットシステムに連結するハードウェアデバッガと µVision を組み合わせた複合型の製品です。以下のデバッグデバイスがサポートされています。

- **JTAG/OCDS アダプタ**。ARM 組み込み ICE などのオンチップデバッグシステムに接続します。
- **ターゲットモニタ**。ユーザハードウェアと統合され、多くの評価ボードで使用できます。
- **エミュレータ**。ターゲットハードウェアの MCU ピンに接続します。
- **インシステムデバッガ**。ユーザアプリケーションプログラムの一部で、基本的なテスト機能があります。

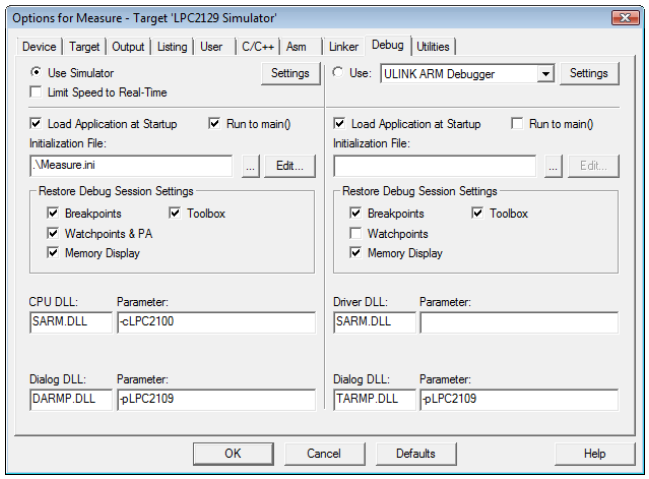
[1] シミュレータを使用すると、ターゲットハードウェアでのデバッグ時よりも多くの能力や機能を使用できます。シミュレータは PC ですべての機能を実行でき、ハードウェアの制限事項によって制限されません。

[2] プログラムはターゲットハードウェアで動作します。制限のあるアプリケーションをデバッグできます。

サードパーティ製ツールの開発者は、Keil Advanced GDI を使用して、µVision を独自のハードウェアデバッガにインタフェース接続できます。

デバッグにシミュレータとターゲットデバッガのいずれを使用する場合でも、µVision IDE では、操作をすぐにマスターできる単一ユーザインタフェースが提供されています。

シミュレータを使用してプログラムをデバッグするには、［**Debug**］ダイアログの左側にある［**Use Simulator**］を選択します。



ターゲットハードウェアで実行中のプログラムをデバッグするには、［**Debug**］ダイアログの右側にある［**Use <*Hardware Debugger*>**］を選択します。

デバッグにシミュレータとターゲットデバッガのいずれを使用するかを選択するほか、［**Debug**］ダイアログには、さまざまなデバッガコンフィギュレーションオプションがあります。

コントロール	説明
Settings	シミュレーションドライバまたは Advanced GDI ターゲットドライバのコンフィギュレーションダイアログを開きます。
Load Application at Startup	デバッガの起動時にアプリケーションプログラムをロードします。
Limit Speed to Real-Time	シミュレーションがターゲットハードウェアより高速で実行されないように、シミュレーション速度をリアルタイムに制限します。
Run to main()	メイン C 関数でプログラムの実行を停止します。このオプションが設定されていない場合、プログラムは main 関数より前にある暗黙のブレークポイントで停止します。
Initialization File	プログラムの実行が開始される前に、デバッガの起動時に読み出され、実行されるコマンドスクリプトファイルを指定します。
Breakpoints	前のデバッグセッションのブレークポイント設定を復元します。
Watchpoints & PA	前のデバッグセッションのウォッチポイントとパフォーマンスアナライザの設定を復元します。
Memory Display	前のデバッグセッションのメモリ表示設定を復元します。
Toolbox	前のデバッグセッションのツールボックスボタンを復元します。
CPU DLL	シミュレータの命令セット DLL を指定します。**この設定は変更しないで下さい。**
Driver DLL	ターゲットデバッガの命令セット DLL を指定します。**この設定は変更しないで下さい。**
Dialog DLL	シミュレータまたはターゲットデバッガのペリフェラルダイアログ DLL を指定します。**この設定は変更しないで下さい。**

シミュレーション

µVision により、読み出し、書き込み、実行、またはいずれかの組み合わせに使用する特定の領域をマップできるメモリを 4 GB までシミュレートできます。ほとんどの場合、µVision ではプログラムオブジェクトモジュールから適切なメモリマップを推測できます。不正なメモリアクセスがあれば、自動的にトラップされて報告されます。

µVision を使用すれば、デバイスに固有の多くのシミュレーション機能を実行できます。デバイスデータベースからマイクロコントローラを選択すると、µVision によりそれに従ってシミュレータが設定され、適切な命令セット、タイミング、およびペリフェラルが選択されます。

µVision シミュレータは以下の機能を備えています。

- ARM7、ARM9、Thumb、Thumb2、8051、C166/XE166/XC2000 命令セットを使用して書き込まれたプログラムを実行します。
- 可能な場合には、サイクルに誤差が生じず、命令とオンチップペリフェラルタイミングを正確にシミュレートします。
- 多くの 8051、C166/XE166/XC2000、ARM7、ARM9、および Cortex-Mx デバイスのオンチップペリフェラルをシミュレートします。
- デバッガ C スクリプト言語を使用して、外部入力を模擬できます。

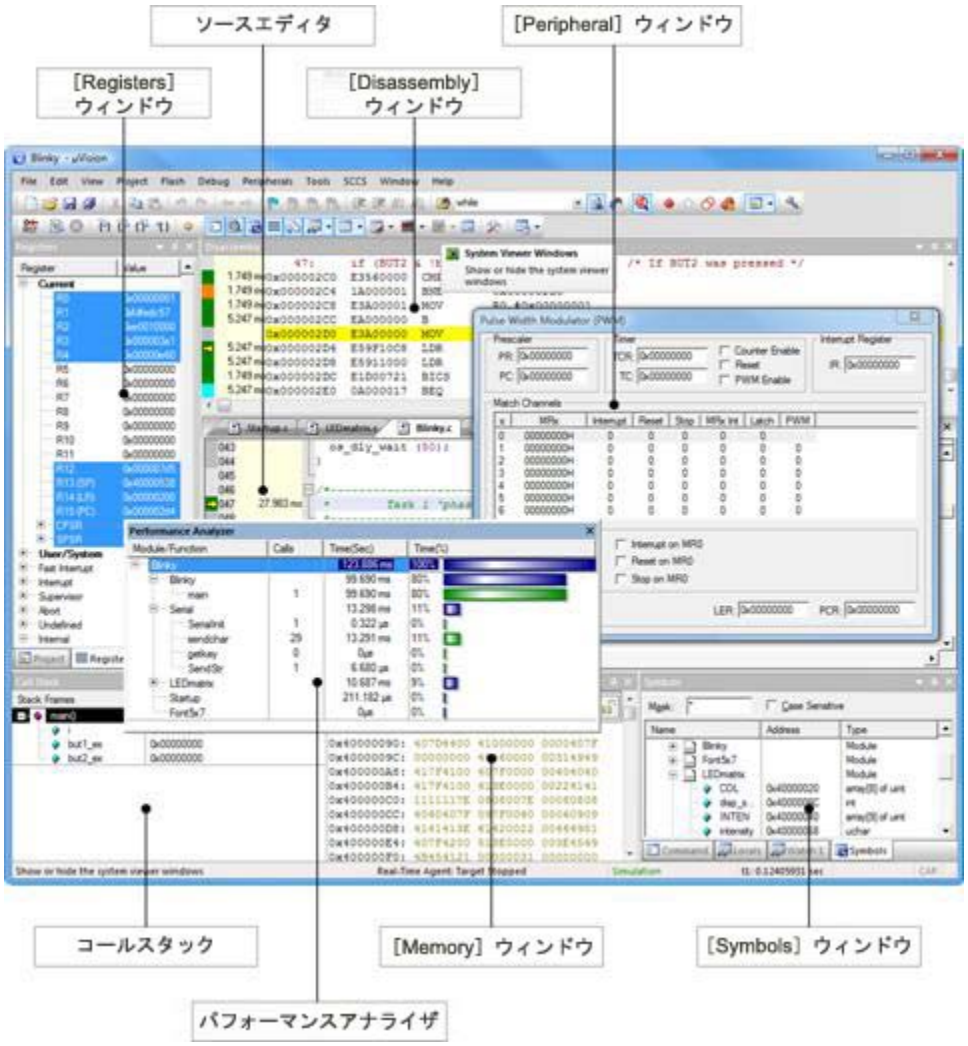
デバッグセッションの開始

デバッグセッションを開始すると、µVision によりアプリケーションがロードされ、スタートアップコードが実行されます。設定されている場合は、メイン C 関数で停止します。プログラムの実行が停止すると、［**Text Editor**］ウィンドウが開き、現在のソースコード行がハイライトされます。また、［**Disassembly**］ウィンドウが開き、逆アセンブルされたコードが表示されます。

Debug ツールバーの［**Start/Stop Debug Session**］コマンドを使用して、デバッグセッションを開始または停止します。画面のレイアウトは、デバッガの切り替え時に復元され、デバッガを閉じると、自動的に保存されます。

現在の命令または高レベルのステートメント（次の命令サイクルで実行されるもの）には黄色の矢印が付きます。先に進むたびに、新たな現在の行や命令に合わせて矢印が移動します。

以下の画面は、**デバッグモード**で使用できる主要なウィンドウの一部を示しています。



デバッグモード

デバッグ中も、ほとんどの編集機能を使用できます。例えば、［**Find**］コマンドを使用して、ソーステキストの位置を特定したり、ソースコードを変更したりできます。デバッガインタフェースの多くは、テキストエディタのインタフェースと同じです。

ただし、**デバッグモード**では、さらに以下の機能、メニュー、およびウィンドウを使用できます。

［**Debug**］メニューと **Debug ツールバー**。デバッグコマンドにアクセスできます。

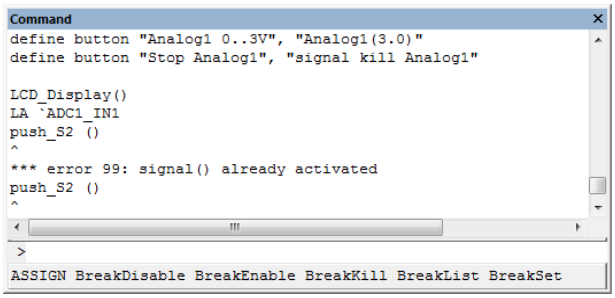
- ［**Peripherals**］メニュー。環境の監視に使用されるペリフェラルのダイアログが表示されます。
- ［**Command**］**ウィンドウ**。デバッグコマンドを実行し、デバッガのメッセージを表示します。
- ［**Disassembly**］**ウィンドウ**。ソースコードの逆アセンブリにアクセスできます。
- ［**Registers**］**ウィンドウ**。レジスタの値を直接表示し、変更できます。
- ［**Call Stack**］**ウィンドウ**。プログラムコールツリーを確認できます。
- ［**Memory**］、［**Serial**］、および［**Watch**］**の各ウィンドウ**。アプリケーションを監視できます。
- ［**Performance Analyzer**］**ウィンドウ**。パフォーマンスを最適化するためアプリケーションを調整できます。
- ［**Code Coverage**］**ウィンドウ**。セーフティクリティカルシステムのコードを検査できます。
- ［**Logic Analyzer**］**ウィンドウ**。シグナルと変数をグラフィックで確認できます。
- ［**Execution Profiler**］実行時間と呼び出しの回数を調べることができます。

- ［**Instruction Trace**］ウィンドウ。プログラムシーケンスの実行を追跡できます。
- ［**Symbol**］ウィンドウ。プログラムオブジェクトを容易に検索できます。
- **システムビューア**。ペリフェラルレジスタを監視できます。
- **複数のデバッグ復元レイアウト**。定義して目的のウィンドウ配置を切り替えることができます。

無効になっているビルドコマンドのほか、以下の操作はできません。

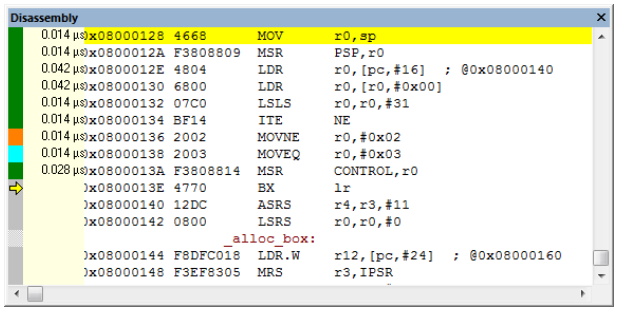
- プロジェクト構造の変更
- ツールパラメータの変更

［Command］ウィンドウの使用



このウィンドウには、コードのステップスルー時に、コンパイルおよびデバッグの一般的な情報が表示されます。例えば、メモリ領域にアクセスできない場合などは、通知が表示されます。［**Command**］ウィンドウの**コマンドライン**にデバッガコマンドを入力すると、パラメータおよびパラメータオプションのヒントと共に、有効な命令がステータスバーに表示されます。レジスタ、変数、およびメモリ領域の内容を表示または変更するための式を挿入します。デバッガスクリプト関数を呼び出すこともできます。**F1** キーを押して、詳細なオンラインヘルプ情報を参照することをお勧めします。使用可能な多くのオプションについての説明は、本書では割愛します。

［Disassembly］ウィンドウの使用



このウィンドウを設定するには、その**コンテキストメニュー**を開きます。このウィンドウを使用して、命令を実行する時刻や呼び出しの回数を表示できます。また、ブレークポイントとブックマークの設定や削除もできます。

以前に実行された命令のトレース履歴は、［**View**］→［**Trace**］→［**View Trace Records**］**メニュー**から表示します。トレース履歴を表示するには、［**View**］→［**Trace**］→［**Enable Trace Recording**］オプションを有効にします。

［**Disassembly**］**ウィンドウ**がアクティブなウィンドウの場合、シングルステップ実行は、プログラムソースレベルではなくアセンブラ命令レベルで機能します。

コードの実行

µVision では、以下のようにいくつかの方法でプログラムを実行できます。

- プログラムがメイン C 関数に直接実行されるように指定できます。［Options for Target］ダイアログの［Debug］タブでこのオプションを設定します。
- ［Debug］メニューまたは Debug ツールバーからデバッガコマンドを選択します。
- ［Command］ウィンドウにデバッガコマンドを入力します。
- 初期化ファイルからデバッガコマンドを実行します。

プログラムの起動

プログラムを実行するには、**Debug ツールバー**または［**Debug**］メニューから［**Run**］コマンドを選択するか、［**Command**］ウィンドウに「**GO**」と入力します。

プログラムの停止

Debug ツールバーまたは［**Debug**］メニューから［**Stop**］を選択するか、［**Command**］ウィンドウで **Esc** キーを押します。

CPU のリセット

シミュレートした CPU をリセットするには、**Debug ツールバー**または［**Debug**］→［**Reset CPU**］メニューから［**Reset**］を選択するか、［**Command**］ウィンドウに「**RESET**」と入力します。

シングルステップ実行

プログラムをステップスルーし、関数呼び出しにステップインするには、**Debug ツールバー**または［**Debug**］メニューから［**Step**］コマンドを使用します。または、［**Command**］ウィンドウに「**TSTEP**」と入力するか、**F11** キーを押します。

プログラムをステップスルーし、関数呼び出しをステップオーバーするには、**Debug ツールバー**または［**Debug**］メニューから［**Step Over**］コマンドを使用します。［**Command**］ウィンドウに「**PSTEP**」と入力するか、**F10** キーを押します。

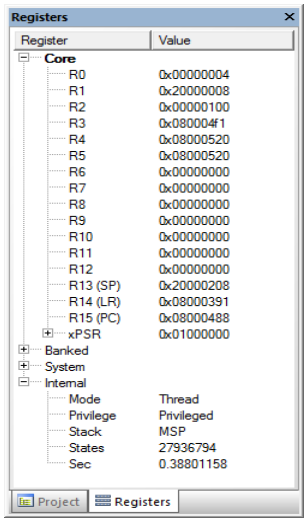
現在の関数からステップアウトするには、**Debug ツールバー**または［**Debug**］メニューから［**Step Out**］コマンドを使用します。［**Command**］ウィンドウに「**OSTEP**」と入力するか、**Ctrl+F11** キーを押します。

メモリの検証と変更

µVision では、さまざまな方法でプログラムおよびデータメモリを観察し、変更できます。いくつかのウィンドウでは、便利な形式でメモリの内容が表示されます。

レジスタの内容の表示

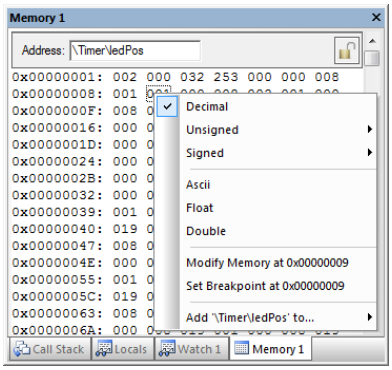
［**Registers**］ウィンドウには、マイクロコントローラのレジスタの内容が表示されます。レジスタの内容を変更するには、レジスタの値をダブルクリックします。**F2** キーを押して、選択した値を編集することもできます。



［Memory］ウィンドウ

メモリ領域は、4 つの個別の［**Memory**］**ウィンドウ**から監視できます。

Debug ツールバーまたは［**View**］→［**Memory**］→［**Memory[x]**］メニューから［**Memory**］ウィンドウを開きます。



コンテキストメニューを使用して、出力形式を選択できます。

［**Address**］フィールドに、目的の領域またはオブジェクトを監視するための式を入力します。アドレスの内容を変更するには、値をダブルクリックして変更します。

［**Memory**］**ウィンドウ**を定期的に更新するには、［**View**］→［**Periodic Window Update**］を有効にします。ウィンドウを手動で更新するには、［**Toolbox**］の［**Update Windows**］を使用します。

［**Memory**］**ウィンドウ**が更新されないようにするには、［**View**］→［**Periodic Window Update**］をオフにするか、［**Lock**］ボタンを使用して、ウィンドウのスナップショットを取得します。同じアドレス空間の値を比較するには、2 番目の［**Memory**］**ウィンドウ**で同じセクションのスナップショットを取得します。

メモリコマンド

以下のコマンドを［**Command**］ウィンドウに入力できます。

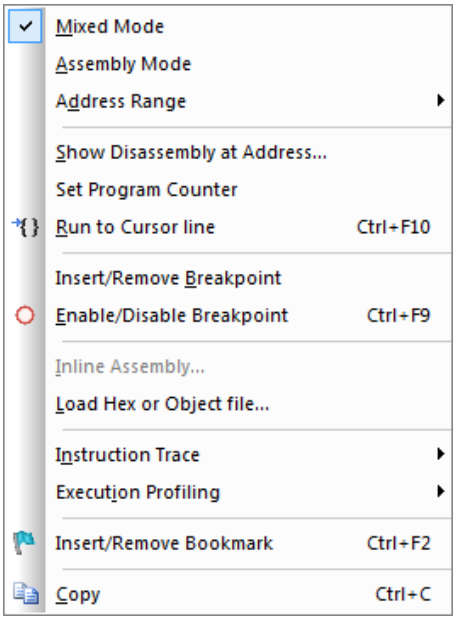
コマンド	説明
ASM	現在のアセンブリアドレスを表示または設定し、アセンブリ命令を入力できるようにします。命令を入力すると、生成された OP コードがコードメモリに保存されます。インラインアセンブラを使用して、ミスを修正したり、ターゲットプログラムに一時的な変更を加えたりできます。
DISPLAY	［Memory］ウィンドウ（開いている場合）または［Command］ウィンドウに一定範囲のメモリを表示します。メモリ領域は、HEX および ASCII 形式で表示されます。
ENTER	指定したアドレスで始まるメモリの内容を変更できます。
EVALUATE	指定した式を計算し、10 進数、8 進数、HEX、および ASCII 形式で結果を出力します。
UNASSEMBLE	コードメモリを逆アセンブルし、［Disassembly］ウィンドウに表示します。

ブレークポイントとブックマーク

µVision では、以下の操作の実行中にブレークポイントとブックマークを設定できます。

- プログラムのソースコードを作成または編集しているとき
- ［**Debug**］メニューから表示した［**Breakpoints**］ダイアログを使用して、デバッグしているとき
- ［**Command**］ウィンドウに入力したコマンドを使用してデバッグしているとき

ブレークポイントとブックマークの設定



ソースコードまたは［**Disassembly**］**ウィンドウ**で実行ブレークポイントを設定するには、**コンテキストメニュー**を開き、［**Insert/Remove Breakpoint**］コマンドを選択します。

［**Editor**］**ウィンドウ**または［**Disassembly**］**ウィンドウ**のグレーのサイドバーをダブルクリックしてブレークポイントを設定するか、**File ツールバー**のブレークポイントボタンを使用できます。

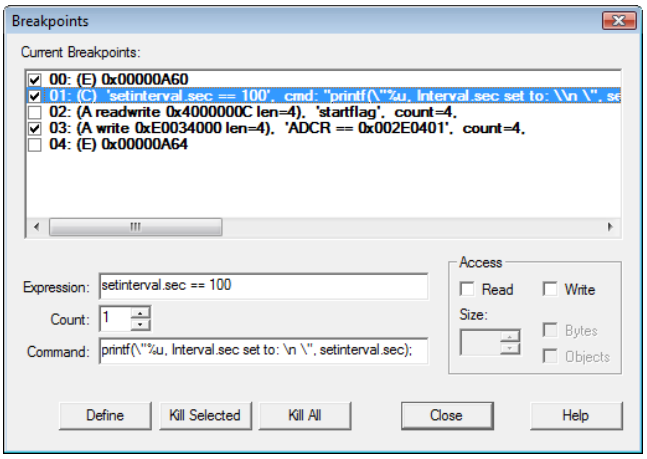
ブレークポイントとブックマークは、［**Editor**］**ウィンドウ**と［**Disassembly**］**ウィンドウ**で同様に表示されますが、色分けは異なります。ブレークポイントは赤で表示され、ブックマークは青で示されます。

ブックマークを定義するには、アナログ操作が必要です。ブレークポイントとは異なり、ブックマークによってプログラムの実行が停止することはありません。

ブックマークを使用して、ソースコードにリマインダとマーカを設定します。ブックマークナビゲーションコマンドを使用すると、重要な箇所を簡単に定義し、ブックマーク間を速やかに移動できます。同じコード行でブックマークとブレークポイントを同時に定義することもできます。

ブックマークにはこの他の説明はありませんが、ブレークポイントについては、以下のセクションで詳しく説明します。

［Breakpoints］ウィンドウ



［**Debug**］メニューから［**Breakpoints**］**ウィンドウ**を表示します。

このダイアログにアクセスするには、実行中のプログラムを停止する必要があります。

このダイアログで既存のブレークポイントを変更したり、新しいブレークポイントを追加したりできます。［**Current Breakpoints**］リストのチェックボックスを使用して、ブレークポイントを有効または無効にすることができます。既存のブレークポイントの定義を変更するには、そのブレークポイントをダブルクリックします。

ブレークポイントを定義するには、**式**を入力します。入力した式に従って、以下のブレークポイントタイプのいずれかが定義されます。

- **実行ブレークポイント（E）**は、式でコードアドレスが指定される場合に定義されます。このブレークポイントは、指定したコードアドレスに到達した場合にトリガされます。コードアドレスは、マイクロコントローラ命令の最初のバイトを参照する必要があります。
- **アクセスブレークポイント（A）**は、式でメモリアクセス（読み出し、書き込み、またはその両方）命令が指定される場合に定義されます。このブレークポイントは、特定のメモリアクセスが発生した場合にトリガされます。ブレークポイントをトリガするバイト数またはオブジェクト数（式に基づく）を指定できます。式は、メモリアドレスとメモリタイプに変換する必要があります。演算子（&、&&、<、<=. >、>=、= =、!=）は、**アクセスブレークポイント**がトリガされ、プログラムの実行が中断されるか、**コマンド**が実行される前の値を比較するのに使用できます。

- **条件ブレークポイント（C）**は、式で真/偽条件が指定され、式をアドレスに変換できない場合に定義されます。このブレークポイントは、指定した条件式が真である場合にトリガされます。条件式は、各命令の後で再計算されます。したがって、プログラムの実行速度が大幅に遅くなります。

ブレークポイントに**コマンド**を指定すると、µVision によってそのコマンドが実行され、ターゲットプログラムの実行が再開します。コマンドには、µVision デバッグ関数またはシグナル関数を指定できます。µVision 関数でプログラムの実行を中断するには、**_break_** システム変数を設定します。詳細については、オンラインヘルプの「*System Variables*」を参照して下さい。

［**Count**］値は、ブレークポイントがトリガされる前に、ブレークポイント式が真である回数を指定します。

ブレークポイントコマンド

以下のブレークポイントコマンドを［**Command**］**ウィンドウ**に入力できます。

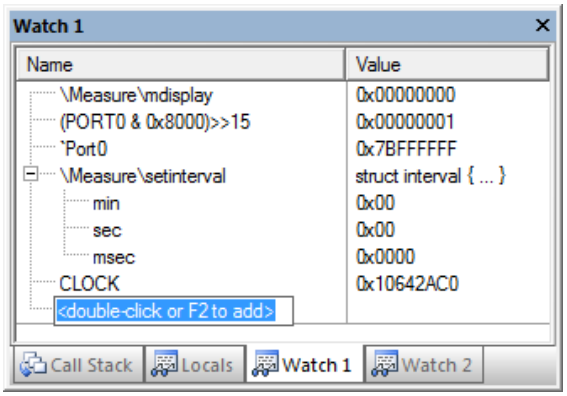
コマンド	説明
BREAKSET	指定した式にブレークポイントを設定します。ブレークポイントは、真の場合にターゲットプログラムの実行を中断するか、指定されたコマンドを実行するプログラムアドレスまたは式です。
BREAKDISABLE	以前に定義されたブレークポイントを無効にします。
BREAKENABLE	以前に定義され、無効になっているブレークポイントを有効にします。
BREAKKILL	以前に定義されたブレークポイントを削除します。
BREAKLIST	すべてのブレークポイントをリストします。

編集中またはデバッグ中に、**File ツールバー**のボタンを使用して実行ブレークポイントを設定することもできます。

ウォッチポイントと［Watch］ウィンドウ

デフォルトでは、［**Watch**］**ウィンドウ**は、現在の関数の変数が表示される［**Locals**］ページ、カスタマイズされたウォッチポイントが表示される2 つの［**Watch**］ページ、プログラムツリーが表示される［**Call Stack**］の4 つのページタブから構成されています。［**Watch**］**ウィンドウ**を使用して、プログラム変数を表示し、変更できます。このウィンドウには、ネストされた関数の呼び出しもリストされます。**デバッグモード**でコードをステップスルーする場合に、［**View**］→［**Periodic Window Update**］オプションが設定されていれば、内容が自動的に更新されます。［**Locals**］**ウィンドウ**にはローカル関数の変数がすべて表示されるのに対し、［**Watch**］**ウィンドウ**にはユーザ指定のプログラム変数が表示されます。

ウォッチポイント



ウォッチポイントを定義して、ターゲットプログラムによって影響を受ける変数、オブジェクト、およびメモリ領域を観察します。ウォッチポイントは、2 つの［**Watch**］ページで定義できます。［**Locals**］**ウィンドウ**には、現在実行されている関数の項目が表示されます。これらの項目は、［**Locals**］**ウィンドウ**に自動的に追加されます。

ウォッチポイントを追加するには、以下のようにいくつかの方法があります。

- 任意の［**Watch**］ページで、フィールド［**<double-click or F2 to add>**］を使用します。
- 既存のウォッチポイントをダブルクリックして、名前を変更します。
- **デバッグモード**で、変数の**コンテキストメニュー**を開き、［**Add *<item name>* to…**］→［**Watch**］**ウィンドウ**を使用します。マウスポインタの下にある変数名が自動的に選択されます。式をマークして［**Watch**］**ウィンドウ**に追加することもできます。
- 新しいウォッチポイントを作成するには、［**Command**］**ウィンドウ**で **WATCHSET** コマンドを使用します。
- さらに、［**Symbols**］**ウィンドウ**またはソースコードファイルから［**Watch**］**ウィンドウ**にオブジェクトをドラッグアンドドロップします。

ローカル変数とウォッチポイント値を変更するには、変更する値をダブルクリックするか、値をクリックして **F2** キーを押します。ウォッチポイントを削除するには、そのウォッチポイントを選択して **Del** キーを押します。

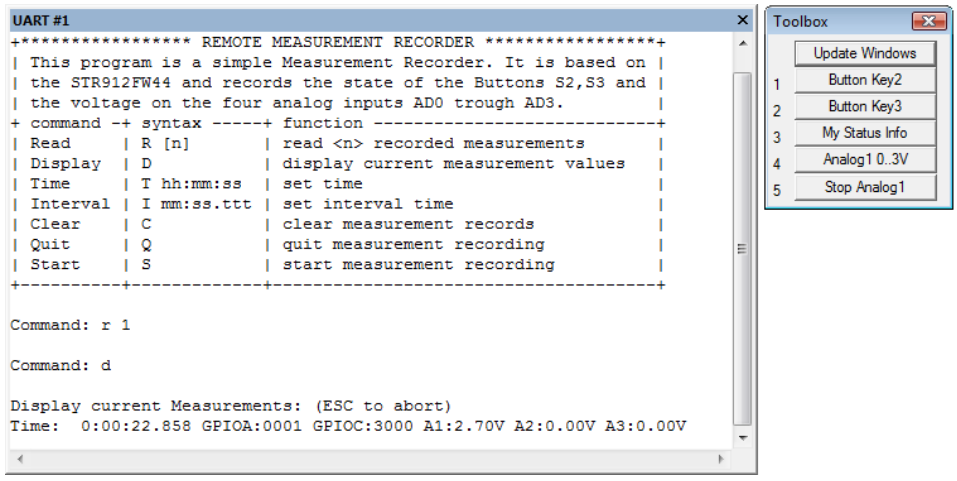
ウォッチポイントコマンド

以下のウォッチポイントコマンドを［**Command**］**ウィンドウ**に入力できます。

コマンド	説明
WATCHSET	［Watch］ウィンドウに表示するウォッチポイント式を定義します。
WATCHKILL	［Watch］ウィンドウで定義されたウォッチポイント式をすべて削除します。

シリアル I/O と UART

µVision には、シミュレートされるそれぞれのオンチップ UART 用の［**Serial**］ウィンドウが 3 つあり、«**UART #**{1|2|3}» という名前が付いています。シミュレートされるマイクロコントローラからのシリアルデータ出力がこのウィンドウに表示されます。［**Serial**］ウィンドウに入力した文字は、シミュレートされるマイクロプロセッサに入力されます。



シリアル出力は、**ASSIGN** デバッガコマンドを使用して PC COM ポートに割り当てることができます。

以下のようなデータ表示モードを使用できます。

- 基本 VT100 端末モード
- 混合モード
- ASCII モード
- HEX モード

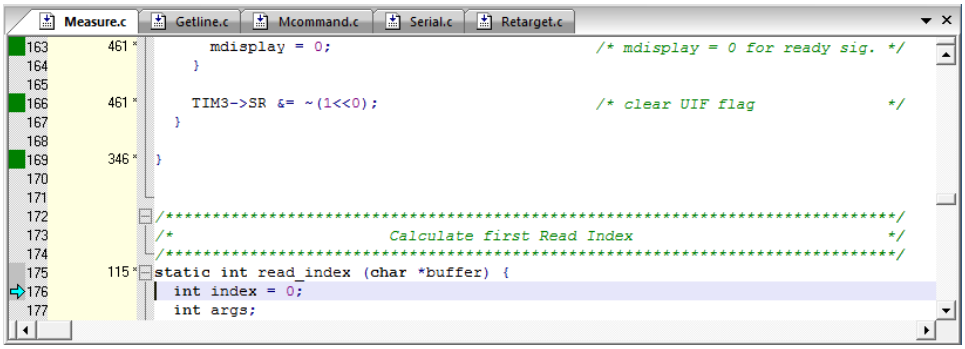
ウィンドウの内容をクリップボードにコピーするか、ファイルに保存できます。必要に応じて、［**Toolbox**］[1] の機能を使用して、プログラムを操作できます。

[1] ［Toolbox］のボタンは、いつでも追加、削除、および変更できます。それには、［Command］ウィンドウでコマンドラインを使用します。

実行プロファイラ

µVision の**実行プロファイラ**により、プログラム実行中の各アセンブラ命令と高レベルのステートメントの時間と回数が記録されます。

呼び出しの時間と回数は累積値です。これらの値は［**Disassembly**］ウィンドウと［**Editor**］ウィンドウに同様に表示されます。

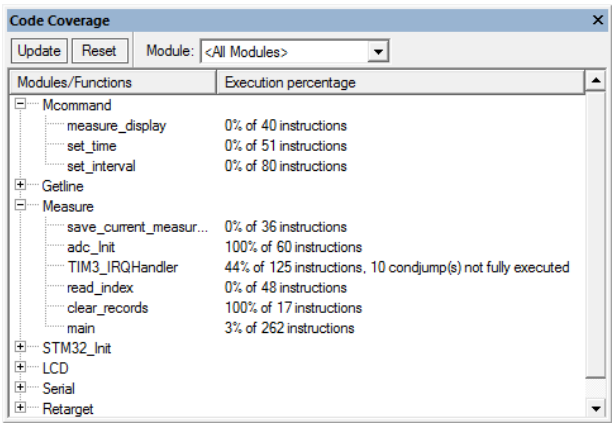


実行プロファイラは、［**Debug**］→［**Execution Profiling**］メニューから有効にします。

時間と呼び出しを切り替えるには、［**Disassembly**］ウィンドウの**コンテキストメニュー**を表示します。

パフォーマンスアナライザを使用してプログラムホットスポットの位置決めをする場合、**実行プロファイラ**を使用すれば、実際のパフォーマンスのボトルネックが簡単に見つかります。

コードカバレッジ



［Code Coverage］ウィンドウでは、実行されたコードにマークが付き、モジュールと関数に基づいて情報がグループ化されます。

この機能を使用して、認証や検証が必要なセーフティクリティカルアプリケーションをテストします。

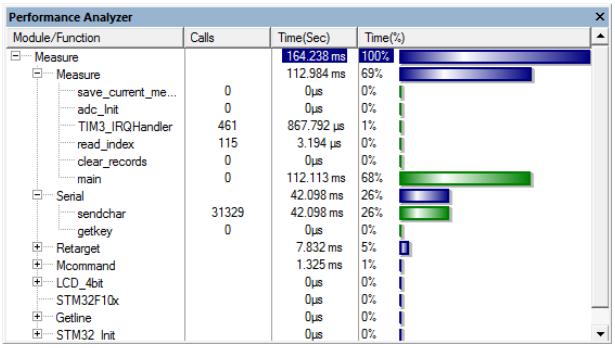
スキップされた命令、
完全に実行された命令、一部が実行された命令、またはまったく実行されなかった命令を検出できます。

コードカバレッジデータは、ファイルに保存できます。このレポートに詳細な CPU 命令のリストを含めることもできます。これらの機能を使用するには、**［Command］ウィンドウ**で **COVERAGE** コマンドを実行します。

µVision では、**［Code Coverage］ウィンドウ**のほかに、**［Disassembly］ウィンドウ**と**［Editor］ウィンドウ**のサイドバーに色分けされたヒントが表示されます。各色の意味は以下のとおりです。

- 実行されていない行は、**グレー**のブロックのマークが付きます。
- 完全に実行された行は、**緑色**のブロックのマークが付きます。
- スキップされた分岐は、**オレンジ色**のブロックのマークが付きます。
- 実行された分岐は、**青色**のブロックのマークが付きます。
- コードのない行は、**薄いグレーのチェック**のブロックのマークが付きます。

パフォーマンスアナライザ



µVision の**パフォーマンスアナライザ**により、アプリケーションプログラムの関数に関して記録された実行時間が表示されます。

結果は、呼び出し回数、関数の実行時間、および合計実行時間に対するこの関数のパーセンテージに従って棒グラフとして表示されます。

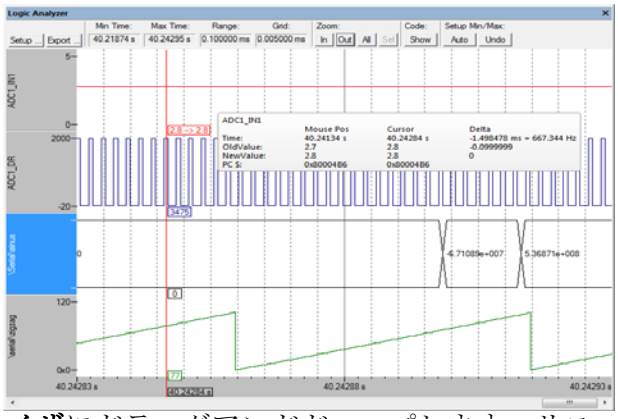
この情報を使用して、プログラムが大部分の時間を費やす場所とさらに調査が必要な部分を特定できます。

費やした時間に従って、オブジェクトが自動的にソートされます。

調査の他の表示に切り替えるには、**パフォーマンスアナライザ**の**コンテキストメニュー**を表示します。出力を操作して、モジュールや関数の統計を表示できます。最終的に新しいサマリーを表示するには、収集したデータを消去する必要があります。

ソースコード行にジャンプするには、モジュールまたは関数の列に表示されているオブジェクトをダブルクリックします。

ロジックアナライザ



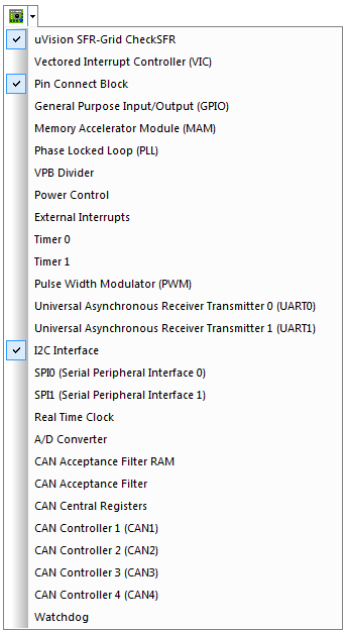
ロジックアナライザにより、変数の値や仮想レジスタが表示され、時間軸での変更が表示されます。

［**Setup …**］ボタンを使用して値を追加するか、他のウィンドウのオブジェクトを**ロジックアナライザ**にドラッグアンドドロップします。リストから項目を削除するには、**Del** キーを押すか、［**Setup …**］ボタンを使用するか、**コンテキストメニュー**を表示します。

［**Logic Analyzer**］ウィンドウには、複数のボタンがあり、データを詳細に分析するための複数のフィールドがあります。マウスポインタを目的の場所に動かし、1 秒待つと、追加情報が自動的にポップアップします。

コントロール	説明
Setup…	［**Logic Analyzer Setup**］ダイアログから変数とその設定を定義します。
Export…	現在記録されているシグナルをタブ区切りファイルに保存します。
Min Time	シグナル記録バッファの開始時間を表示します。
Max Time	シグナル記録バッファの終了時間を表示します。
Range	現在の表示の時間範囲を表示します。
Grid	グリッドラインの時間範囲を表示します。
Zoom	表示された時間範囲を変更します。［**Zoom All**］では、シグナル記録バッファの内容が表示されます。［**Zoom Sel**］では、現在の選択に対する表示がズームされます（**Shift** キーを押したまま、マウスをドラッグしてセクションをマークします）。
Show	シグナル遷移の原因となったコードで［**Editor**］**ウィンドウ**または［**Disassembly**］**ウィンドウ**を開きます。また、プログラムが実行されないようにします。
Setup Min/Max	シグナルの範囲を設定します。［**Auto**］ボタンをクリックすると、現在記録している内容の値に基づいた最小値と最大値が設定されます。［**Undo**］ボタンをクリックすると、［**Auto**］ボタンを使用する前の設定に戻ります。

システムビューア

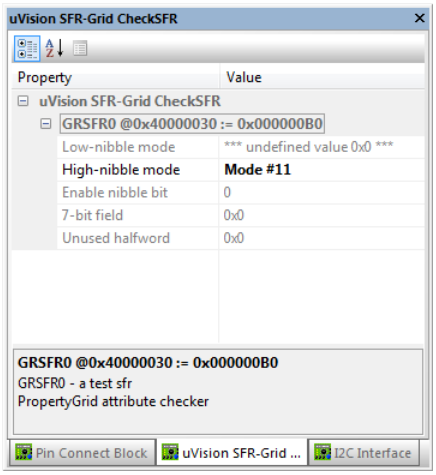


ペリフェラルレジスタは、ペリフェラルデバイスを制御するためにプロセッサによる書き込みおよび読み出しが可能なメモリマップされたレジスタです。µVision では、これらのペリフェラルレジスタを高度な方法で表示してデバッグできます。

Debug ツールバーから**システムビューア**を起動するか、［**View**］→［**System Viewer Windows**］メニューから起動します。動作を監視するために最大 100 のペリフェラルオブジェクトを定義できます。

システムビューアには以下の機能があります。

- マイクロコントローラデバイス C ヘッダファイルを解析して、バイナリ形式にします。
- ヘッダファイルにプロパティを追加して、ペリフェラルレジスタの記述やデータの概要など、追加情報を提供できます。



- ペリフェラルレジスタの値は、シミュレータまたはターゲットハードウェアから更新されます。更新は、ターゲットが停止したときに行われます。または、［**View**］→［**Periodic Window Update**］メニューを有効にすると、定期的に更新できます。
- ペリフェラルレジスタの内容は、**システムビューア**で値を上書きすることによって、いつでも変更できます。

［Symbols］ウィンドウ

［**Symbols**］**ウィンドウ**には、デバッガからの情報が順番にグループ分けされて表示されます。このウィンドウは、**Debug ツールバー**から表示するか、または［**View**］→［**Symbol Window**］メニューから表示できます。この機能により、以下のようなオブジェクトが表示されます。

- 仮想レジスタである**シミュレータ VTREG** としてシミュレートされたリソースのオブジェクト。I/O ピン、UART 通信、または CAN トラフィックにアクセスできます。
- ペリフェラルレジスタである**ペリフェラル SFR** からのオブジェクト。ペリフェラルにアクセスできます。
- 組み込みアプリケーションのオブジェクト。プログラム名によって認識できます。関数、モジュール、変数、構造体などのソースコード要素にアクセスできます。

この機能を使用して、項目を速やかに見つけることができます。オブジェクトを µVision の他のウィンドウにドラッグアンドドロップします。

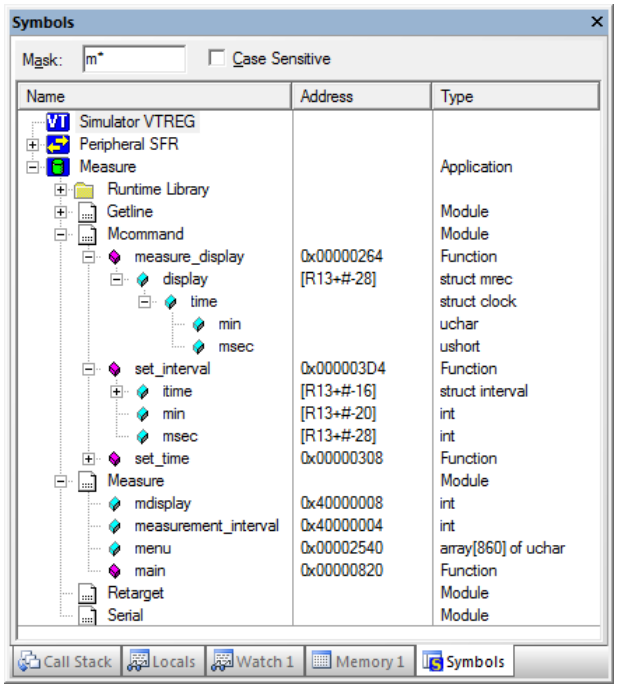
マスクは、検索機能と同様に動作します。検索基準を入力し、結果を参照します。ネストされたオブジェクトでは、リーフ項目がある場合、ツリー全体が表示されます。以下の検索基準を使用できます。

は数字（0～9）と一致します。

$ は 1 文字と一致します。

***** は 0 文字以上の文字と一致します。

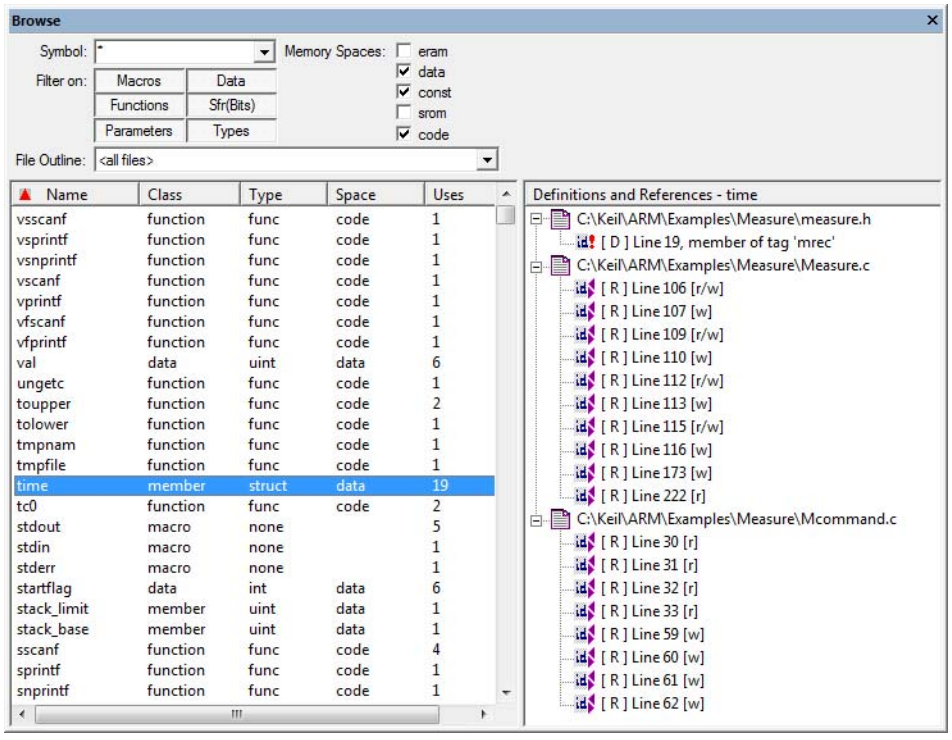
このウィンドウを設定するには、**コンテキストメニュー**を表示します。



［Browse］ウィンドウ

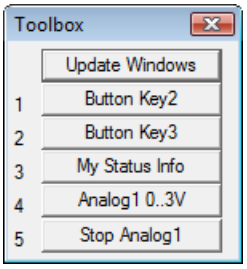
［Browse］ウィンドウでは、コード内のオブジェクトを検索できます。この機能は、**デバッグモード**および**ビルドモード**で使用できます。ただし、参照情報はコンパイル後にのみ入手できます。**［Options for Target］→［Output］→［Browser Information］**オプションを設定して、参照情報をオブジェクトファイルに含めるようコンパイラに指示する必要があります。このウィンドウは、**File ツールバー**または**［View］→［Source Browser Window］**から表示します。

［Filter on］ボタンを有効または無効にし、**［Symbol］**フィールドに検索基準を入力し、**［File Outline］**ドロップダウンを使用して結果を絞り込みます。ヘッダコントロールをクリックして、結果をソートできます。項目が使用されている場所を参照して特定するには、項目をクリックします。コード行にジャンプするには、**［Definition and References］**ページの行をダブルクリックします。



項目をポイントしながら**コンテキストメニュー**を表示します。表示されるオプションは、オブジェクトクラスによって異なります。関数の場合は、呼び出し元のグラフとコールグラフを表示できます。

［Toolbox］



［**Toolbox**］には、デバッガコマンドまたはユーザ定義関数を実行するユーザ設定可能なボタンが含まれています。［**Toolbox**］のボタンをクリックして、関連付けられているコマンドを実行します。
［**Toolbox**］のボタンは、プログラムの実行中を含め、いつでもクリックできます。

［**Toolbox**］のボタンは、［**Command**］ウィンドウで［**DEFINE BUTTON**[1] [2]］コマンドを使用して定義します。ボタンを再定義するには、同じコマンドを使用します。このコマンドの一般的な構文は、以下のようになります。

```
DEFINE BUTTON "button_label", "command"
```

各パラメータには以下の意味があります。

button_label　［Toolbox］内で表示される名前です。

command　そのボタンを押すと実行されるコマンドです。

[1] 最後の例で定義された printf() コマンドでは、ネストされた文字列が使用されています。形式文字列の二重引用符（"）とバックスラッシュ（\）文字は、構文エラーを避けるために \ でエスケープする必要があります。

[2] このコマンドを使用して、ボタンの意味を再定義したり、説明を変更したりします。

以下の例には、上記の［Toolbox］内のボタンを作成するために使用されるコマンドが示されています。

```
DEFINE BUTTON "Decimal Output", "radix=0x0A"
DEFINE BUTTON "My Status Info", "MyStatus ()"   /* call debug function  */
DEFINE BUTTON "Analog1 0..3V", "analog0 ()"     /* call signal function */

DEFINE BUTTON "Show R15", "printf (\"R15=%04XH\\n\")"
```

KILL BUTTON[1] コマンドを使用して、［**Toolbox**］のボタンを削除します。ボタンの左側に、このステートメントに必要なボタン番号を示します。以下に例を示します。

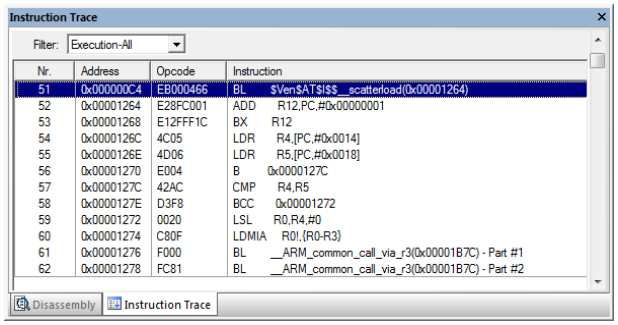
```
KILL BUTTON 5      /* resembles to: “Remove the ‘Stop Analog1’ button” */
```

[1] ［Toolbox］内の［Update Windows］ボタンは自動的に作成され、削除できません。このボタンを押すと、複数のデバッガウィンドウの内容が更新されます。

［Instruction Trace］ウィンドウ

命令シーケンス履歴を追跡するには、**Debug ツールバー**または［**View**］→［**Trace**］メニューから［**Instruction Trace**］**ウィンドウ**を表示します。このウィンドウは、［**Disassembly**］**ウィンドウ**と共に使用します。必要な情報を収集するには、トレース記録を有効にする必要があります。トレース記録を有効にするには、［**View**］→［**Trace**］→［**Enable Trace Recording**］メニューを使用します。

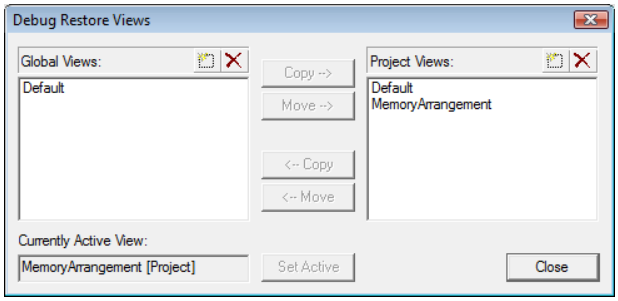
［**Disassembly**］**ウィンドウ**にジャンプするか、ウィンドウを開くには、［**Instruction Trace**］**ウィンドウ**で任意の行をダブルクリックします。命令ツリーを目的のモードで表示するには、事前に定義されている［**Filter**］オプションを使用します。



この機能は、ターゲットハードウェアのデバッグ中に**シミュレータ**に使用できます。このウィンドウの外観は、デバッグ環境のドライバ設定によって異なります。

デバッグ復元ビューの定義

デバッグモードでは、目的の画面設定とウィンドウ配置を速やかに切り替えるために、複数のウィンドウレイアウトを設定できます。［**Window**］→［**Debug Restore Views…**］ダイアログまたは **Debug** ツールバーからレイアウトを表示します。［**Window**］→［**Reset View to Defaults**］からデフォルトを復元します。



［**Window Restore Views…**］ダイアログから目的の外観を定義して保存します。

グローバルビューは、すべてのプロジェクトに反映されます。一方、**プロジェクトビュー**は、特定のプロジェクトに関連付けられます。

第 8 章 ターゲットハードウェアの使用

この章ではターゲットハードウエアとともにμVision のデバッグの可能性について述べています。Keil ULINK USB-JTAG アダプタファミリの詳細と、サードパーティ製アダプタについても説明します。

Keil **ULINK** アダプタでは、以下のデバイスファミリがサポートされています。

- 8051 Infineon XC8xx、ST µPSD3xxx、および NXP LPC95x 用 **ULINK**
- 166 Infineon C166、XE166、および XC2000 用 **ULINK**
- ARM ARM7、ARM9、および Cortex-Mx デバイス用 **ULINK**、**ULINK*Pro***

µVision デバッガは、Keil によって提供される以下のドライバを介してターゲットハードウェアとインタフェース接続します。

- 8051 **Monitor**、**FlashMon**、**MonADI**、**ISD51**、**EPM900**、**Infineon DAS**
- 251 **Monitor**
- 166 C166 用 **Monitor**
- 166 XE166、XC2000 用 **Monitor**、**Infineon DAS**
- ARM ARM7、ARM9、および Cortex-Mx 用 SEGGER **J-Link/J-Trace**

また、以下に示すように、多くのサードパーティベンダが独自のハードウェアに対応する µVision ドライバを提供しています。

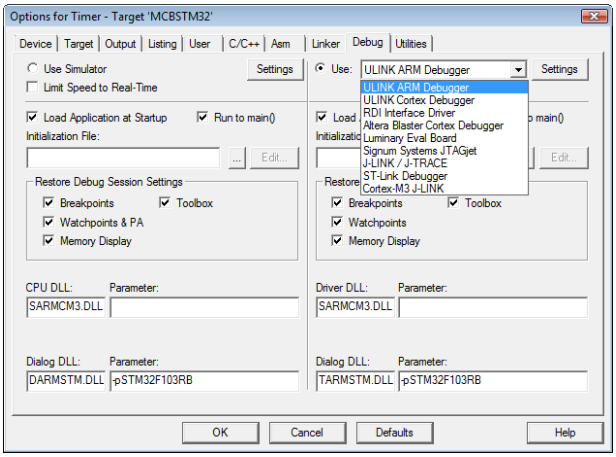
- 8051 EZ-USB デバイス用 Cypress **USB 開発キット**
- 8051 Quickcore **FPGA ベースの Pro8051** デバイス
- 8051 FlashFlex51 デバイス用 SST **SoftICE**
- 8051 C8051Fxxx デバイス用 Silabs **Debug Adapter**
- ARM ARM7、ARM9、および Cortex-Mx デバイス用 Signum Systems **JTAGjet**

デバッガの設定

Build ツールバーで［**Target Options**］を選択し、［**Debug**］タブを選択します。

または、［**Project**］→［**Options for Target**］メニューを使用してダイアログを開きます。

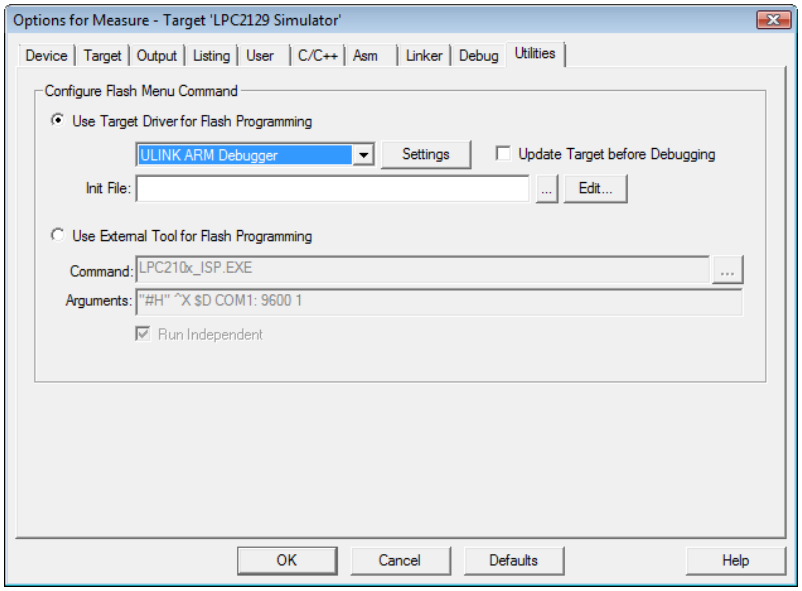
［**Use**］ラジオボタンをオンにして、該当するデバッグインタフェースを選択します。



コントロール	説明
Settings	シミュレーションドライバまたは Advanced GDI ターゲットドライバのコンフィギュレーションダイアログを開きます。
Load Application at Startup	デバッガの起動時にアプリケーションプログラムをロードします。
Limit Speed to Real-Time	シミュレーションがターゲットハードウェアより高速で実行されないように、シミュレーション速度をリアルタイムに制限します。
Run to main()	main C 関数でプログラムの実行を停止します。このオプションが設定されていない場合、プログラムは main 関数より前にある暗黙のブレークポイントで停止します。
Initialization File	プログラムの実行が開始される前に、デバッガの起動時に読み出され、実行されるコマンドスクリプトファイルを指定します。
Breakpoints	前のデバッグセッションのブレークポイント設定を復元します。
Watchpoints & PA	前のデバッグセッションのウォッチポイントとパフォーマンスアナライザの設定を復元します。
Memory Display	前のデバッグセッションのメモリ表示設定を復元します。
Toolbox	前のデバッグセッションのツールボックスボタンを復元します。
CPU DLL	シミュレータの命令セット DLL を指定します。**この設定は変更しないで下さい。**
Driver DLL	ターゲットデバッガの命令セット DLL を指定します。**この設定は変更しないで下さい。**
Dialog DLL	シミュレータまたはターゲットデバッガのペリフェラルダイアログ DLL を指定します。**この設定は変更しないで下さい。**

Flash デバイスのプログラミング

µVision IDEはターゲットシステムのフラッシュメモリをプログラムするように設定することができます。開発環境への追加や開発環境からの起動が可能なサードパーティ製の Flash プログラミングツールを使用できます。Flash プログラミングは、［**Options for Target**］ダイアログの［**Utilities**］タブから設定します。ターゲットドライバまたはサードパーティ製のコマンドラインツール（通常はチップベンダが提供）を選択する必要があります。



Keil ULINK USB-JTAG アダプタ、SEGGER J-Link、EPM900 エミュレータ、Silabs アダプタなどのターゲットアダプタを使用してシステムの Flash メモリをプログラミングするには、［**Use Target Driver for Flash Programming**］を選択します。

FlashMagic などのサードパーティ製のコマンドラインユーティリティを使用してシステムの Flash メモリをプログラミングするには、［**Use External Tool for Flash Programming**］を選択します。

設定後は、**Build ツールバー**の［**Download to Flash**］ボタンまたは［**Flash**］メニューを使用して、ターゲットシステムの Flash メモリにプログラムをダウンロードします。

Flash メモリに自動的にダウンロードするように µVision デバッガを設定できます。そのためには、［**Update Target before Debugging**］をオンにします。

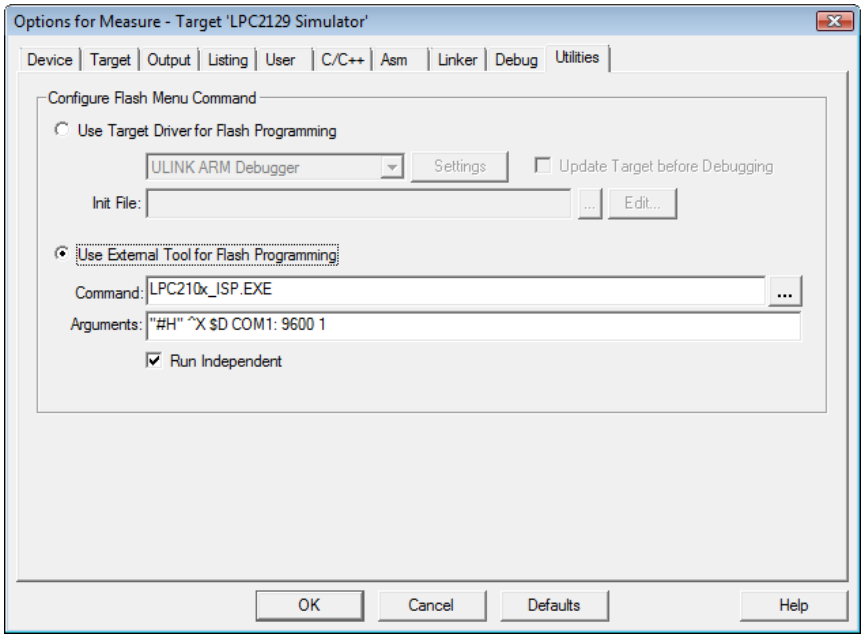
外部ツールの設定

コマンドラインユーティリティでの Flash[1] プログラミングに対応するように µVision を設定するには、［**Use External Tool for Flash Programming**］を選択して使用する**コマンド**と**引数**を指定します。

Build ツールバーで［**Target Options**］を選択し、［**Utilities**］タブを選択します。

または、［**Project**］→［**Options for Target**］メニューを使用して、［**Utilities**］ダイアログを開きます。

[1] µVision デバイスデータベースには、多くのマイクロコントローラデバイスの Flash メモリに適したコンフィギュレーションが用意されています。

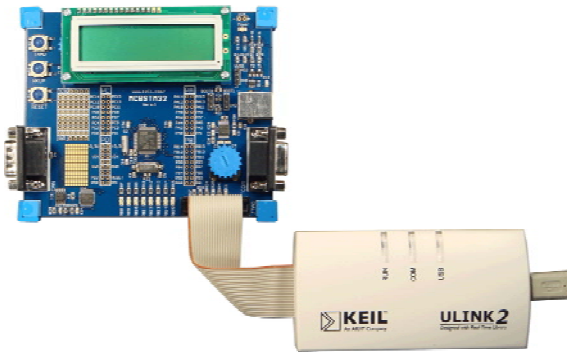


生成される HEX ファイルのパス、出力ファイル名、デバイス名、クロック周波数など、プロジェクト固有の項目は、［Arguments］フィールドで使用される可能性があります。

追加情報については、オンライン**ヘルプ**を参照して下さい。

ULINK アダプタの使用

Keil ULINK USB-JTAG ファミリのアダプタ（ULINK とも呼ばれる）によって、PC の USB ポートがターゲットシステムに接続されます。マイクロコントローラと ULINK ユニット間の接続は、組み込みシステムの JTAG[1] ポートピンを介して確立されます。ULINK アダプタを使用すると、以下の処理を行うことができます。



- ターゲットプログラムのダウンロード
- メモリとレジスタの検査
- プログラムのシングルステップ実行
- 複数のブレークポイントの挿入
- リアルタイムでのプログラムの実行
- Flash メモリのプログラミング

ターゲットハードウェアでデバッガを使用する前に、ULINK アダプタや Flash のプログラミングに適した他の外部ツールを使用するために µVision IDE を設定する必要があります。

µVision デバッガでは、いくつかのよく使用される形式でメモリの内容や変数を表示できます。メモリと変数は定期的に更新され、プログラムの実行中にも現在のプログラムステータスを示すインスタントビューが表示されます。特定の変数へのアクセスをトリガするブレークポイントも設定できます。

[1] ULINK アダプタは広範なデバイスおよびプロトコル、ターゲットハードウェアポートピン特性をサポートしています。

Keil ULINK アダプタファミリでは、設定可能なプログラミングアルゴリズムによる Flash デバイスのプログラミングがサポートされています。事前に設定されているプログラミングアルゴリズムから選択することも、必要に応じてアルゴリズムをカスタマイズすることもできます。外部 Flash メモリのプログラミングは、多くのターゲットシステムでもサポートされています。

ULINK 機能の比較

機能	ULINK2	ULINK*Pro*
実行制御デバッグ（ARM および Cortex-Mx）	あり	あり
実行制御デバッグ（8051 および C166）	あり	-
データトレース（Coretex-M3）	あり	あり
命令トレース（Cortex-M3）	-	あり
JTAG クロック速度	10MHz	50MHz
Flash のダウンロード	28KB/秒	600KB/秒

ULINK アダプタ用の µVision の設定

ULINK アダプタを使用する場合は、ULINK アダプタをデバッグでどのように使用するかを µVision IDE が認識できるように、いくつかの設定を変更する必要があります。詳しくは、以下の項目を設定する必要があります。

- デバッグ設定
- トレース設定（Cortex-Mx デバイスのみ）
- Flash のダウンロード

ULINK アダプタを PC に接続します。接続すると、µVision で ULINK コンフィギュレーションを設定できるようになります。

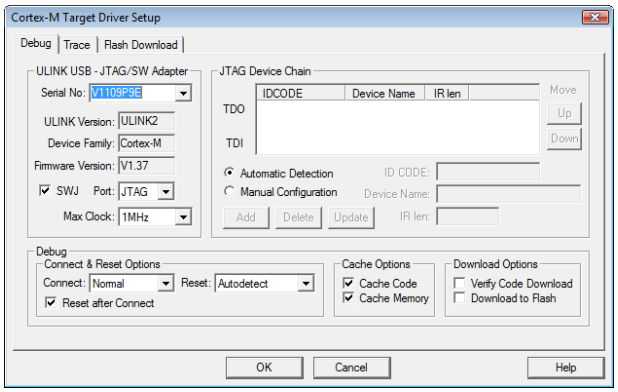
Build ツールバーで［**Target Options**］をクリックして［**Debug**］タブを選択するか、［**Project**］→［**Options for Target**］→［**Debug**］メニューからダイアログを開きます。

［**Settings**］ボタンをクリックして、［**Target Driver Setup**］ダイアログを開きます。

デバッグの設定

［**Target Driver Setup**］ダイアログは、プロジェクトで選択するターゲットデバイスによって異なります。

追加情報については、オンライン**ヘルプ**を参照して下さい。

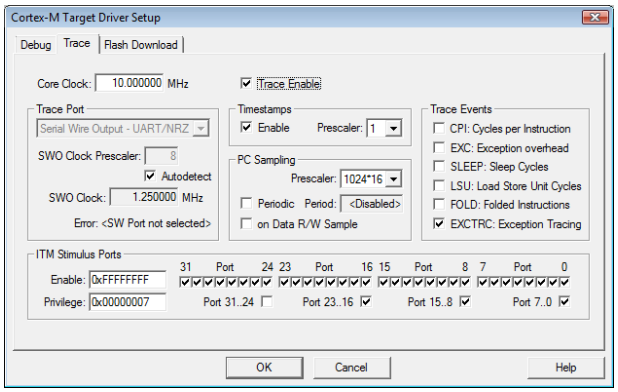


トレースの設定

［**Trace**］ダイアログタブでは、リアルタイムのトレース動作を制御します。

追加情報については、オンライン**ヘルプ**を参照して下さい。

トレース機能を使用できるのは、Cortex-Mx デバイスのみです。



Flash Download のコンフィギュレーション

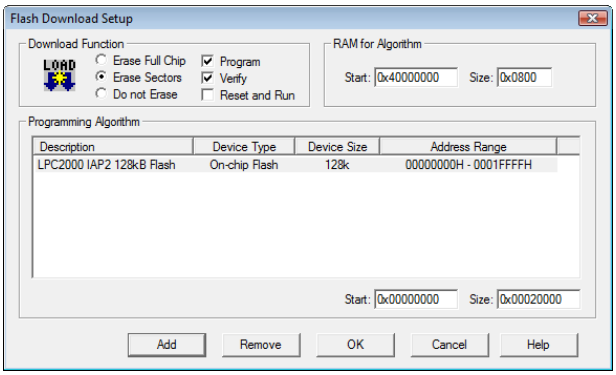
Keil ULINK ドライバは、さまざまな Flash ベースのマイクロコントローラをサポートしています。

Build ツールバーで［**Target Options**］をクリックして［**Utilities**］タブを選択するか、［**Project**］→［**Options for Target**］メニューからダイアログを開きます。

µVision を特定のドライバに合わせて設定するには、［**Use Target Driver for Flash Programming**］を選択し、ドロップダウンコントロールから該当するドライバを選択します。

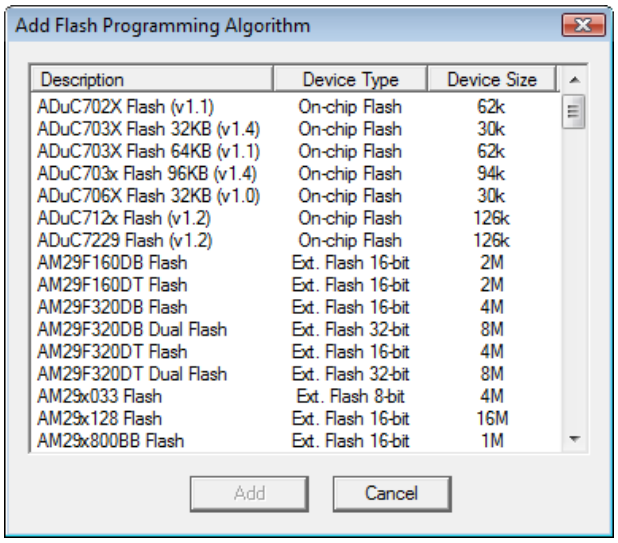
［**Settings**］ボタンを使用して、そのドライバ固有の［**Flash Download Setup**］ダイアログを開きます。

このダイアログでは、Flash のダウンロード動作を設定したり、ターゲットシステムに必要なプログラミングアルゴリズムを指定したりできます。



追加情報については、オンライン**ヘルプ**を参照して下さい。

プログラミングアルゴリズム



ULINK ドライバを使用すると、Flash メモリのプログラミングに使用されるプログラミングアルゴリズムを指定できます。

［**Flash Download Setup**］ダイアログの［**Add**］ボタンを使用して、［**Add Flash Programming Algorithm**］ダイアログを開きます。

このダイアログから、それぞれ異なる Flash デバイス用に 1 つ以上のプログラミングアルゴリズムを選択できます。適切なプログラミングアルゴリズムを強調表示してターゲットハードウェアに**追加**します。

使用する Flash デバイスがリストされない場合は、新しいアルゴリズムを定義できます。この操作は、現在 Keil で直接サポートされていない新しい Flash デバイスを使用する場合に実行して下さい。**\FLASH** フォルダ内のアルゴリズムは、新しいアルゴリズムのテンプレートとして使用できます。

キットに含まれているプログラミングアルゴリズムは、以下のフォルダに保存されています。

- ARM ツールセット：\KEIL\ARM\FLASH\
- C16x ツールセット：\KEIL\C166\FLASH\

Init ファイルの使用

アプリケーションやターゲットシステムの中には、Flash のプログラミングよりも先に、特定のデバッグコマンドまたは関数を実行しなければならないものがあります。この機能は、通常、デバイスに BUS コンフィギュレーションを定義したり、ターゲットプログラムの一部ではないコードまたはデータを含んでいる補助ファイルで Flash をプログラミングしたりする際に使用されます。デバッグコマンドおよび関数は、［**Init File**］テキストボックスで定義される初期化ファイルに保存されます。このファイルは、Flash のダウンロードが行われる前に実行されます。

BUS コンフィギュレーション

外付け Flash メモリが付いたデバイスでは、通常、BUS システムの設定を行って初めてそのデバイスがプログラム可能になります。ULINK USB-JTAG アダプタを使用する場合は、**_WBYTE**、**_WDWORD** などのあらかじめ定義されているデバッグ関数を使用してメモリへの書き込みや BUS の設定を行う初期設定ファイルを作成できます。以下に例を示します。

```
_WDWORD(0xFFE00000, 0x20003CE3);   // BCFG0: Flash Bus Configuration
_WDWORD(0xE002C014, 0x0E6001E4);   // PINSEL2: CS0, OE, WE, BLS0..3
```

補助メモリの内容

BUS の設定に加えて、初期設定ファイルには補助プログラムまたはデータをメモリにロードする命令も含まれています。以下に例を示します。

```
LOAD MyFile.HEX
```

［**Project**］→［**Options for Target**］→［**Output**］で指定された実行可能ファイルは、デフォルトで Flash にダウンロードされます。

第 9 章 サンプルプログラム

Keil の各ツールセットには、すぐに実行できる、使用の開始に役立つサンプルプログラムが用意されています。サンプルプログラムに目を通すことで、開発ツールがどのように動作するかを学び、µVision の外観と動作に慣れることができます。サンプルプログラムのコードをコピーしてそれぞれの用途にも使用できます。

サンプルプログラム[1] は、**\EXAMPLES** フォルダにあります。各プログラムは、サンプルプログラムを再ビルドして µVision の機能を迅速に評価できるように、プロジェクトファイルと共に個別のサブフォルダに格納されています。

検証のためのサンプルプログラムは数多くありますが、本書では、以下の 4 つのプログラムについて説明します。

- Hello： 最初の組み込みプログラム
- Measure： 遠隔測定システム
- Traffic： 交通信号と歩行者横断歩道システム
- Blinky： ターゲットハードウェアの使用方法を示すサンプルプログラム

前述の章で説明したように、µVision の多くの操作や関数はツールバーやメニューから呼び出したり、［**Command**］ウィンドウにコマンドを入力して呼び出すことができます。一部の操作は、キーの組み合わせにより呼び出すこともできます。

[1] サンプルプログラムはライセンスフリーです。

デバッグモードで µVision のさまざまな機能を試すことをお勧めします。前述の章で説明した機能を試して下さい。特に、ナビゲーションに慣れて、各種オブジェクトの**コンテキストメニュー**を開いたり、ウィンドウをドラッグして他の画面領域や物理的に別の画面にドロップしたり、カスタマイズしたレイアウトを作成および保存したりしてみて下さい。**パフォーマンスアナライザ**、**ロジックアナライザ**、**コードカバレッジ**、［**Symbols**］**ウィンドウ**を開き、あるウィンドウの項目を別のウィンドウにドラッグアンドドロップします。コードをシングルステップで実行し、［**Disassembly**］**ウィンドウ**に慣れて、［**Register**］**ウィンドウ**、［**Output** ］**ウィンドウ**、および［**Serial**］**ウィンドウ**とどのように連動するかを確認します。

Hello サンプルプログラム

どのようなプログラミング言語でも最初のプログラムは、単に「`Hello World`」と画面に出力します。組み込みシステムには画面がないため、Hello プログラムはオンチップシリアルポートに出力を送信します。プログラム全体で、ソースファイルは **HELLO.C** のみです。

この小規模アプリケーションは、アプリケーションをコンパイル、リンク、およびデバッグできることを確認するのに役立ちます。これらの操作は、バッチファイルを使用してコマンドラインから実行するか、提供されたプロジェクトファイルを使用して µVision から実行できます。

Hello プロジェクトのターゲットハードウェア[1] は、標準マイクロコントローラに基づいています。サポートされているすべてのアーキテクチャに対してサンプルプログラムが提供され、以下の表に示したフォルダに格納されています。

アーキテクチャ	サンプルフォルダ
ARM	**\KEIL\ARM\EXAMPLES\HELLO**
C166/XE166/XC2000	**\KEIL\C166\EXAMPLES\HELLO**
8051	**\KEIL\C51\EXAMPLES\HELLO**

[1] µVision ではこのプログラムに必要なターゲットハードウェアをシミュレートするので、実際にはターゲットハードウェアまたは評価ボードは必要ありません。

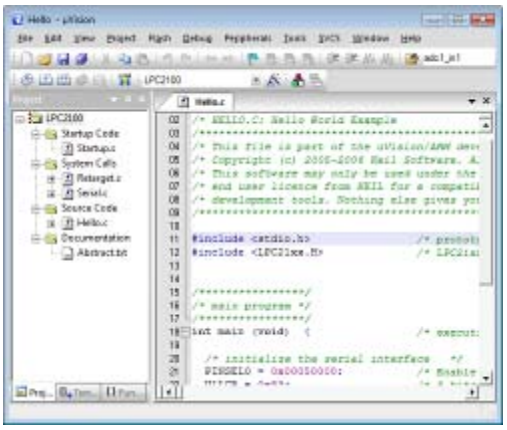
Hello プロジェクトを開く

Hello プロジェクトで作業を開始するには、適切なサンプルフォルダから **HELLO.UVPROJ** プロジェクトファイルを開きます。

［**Project**］→［**Open Project**］メニューを選択し、**...\EXAMPLES\HELLO** フォルダから **HELLO.UVPROJ** を開きます。

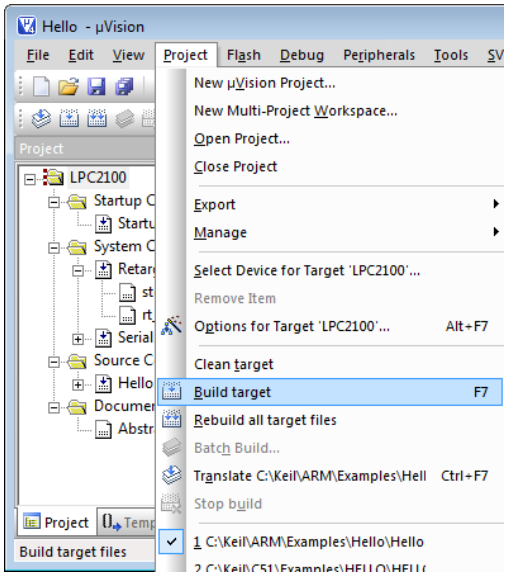
あるいは、µVision アプリケーションに **HELLO.UVPROJ** ファイルをドラッグアンドドロップするか、単にファイルをダブルクリックすることもできます。

このプロジェクトを開くと、µVision によりプロジェクトを構成するソースファイルが表示されます。ファイルは［**Project**］ウィンドウに表示されます。**HELLO.C** をダブルクリックして、ソースファイルを表示したり、編集したりします。µVision では、ファイルの内容が［**Editor**］ウィンドウにロードされ、表示されます。



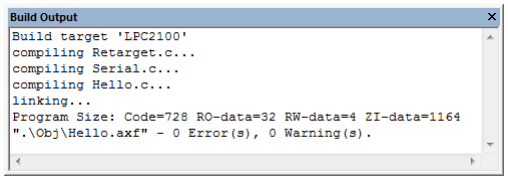
Hello プロジェクトのビルド

プロジェクトをコンパイルして、リンクするには、**Build ツールバー**の［**Build**］ボタンを使用するか、［**Project**］→［**Build Target**］メニューを選択します。



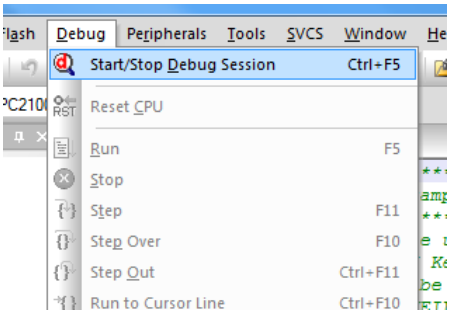
µVision では、アセンブラとコンパイラを実行して、プロジェクトのソースファイルをアセンブルしてコンパイルします。リンカにより、必要なオブジェクトモジュールが追加され、1 つの実行可能プログラムに組み合わされます。これを µVision デバッガでロードしてテストできます。

ビルドプロセスは［**Build Output**］**ウィンドウ**で確認できます。ここには、エラー、警告、およびその他のトレースメッセージが表示されます。エラーまたは警告メッセージをダブルクリックすると、メッセージをトリガしたソース行にジャンプします。



Hello プロジェクトのテスト

Hello プログラムを正常にコンパイルしてリンクしたら、µVision デバッガでテストします。メニューまたは **File ツールバー**から［**Debug**］→［**Start/Stop Debug Session**］を選択します。µVision がデバッガを初期化し、プログラムの実行を開始し、main() C 関数へのエントリで停止します。



プログラムの実行を制御するには、以下のデバッガコマンドを使用します。

［**Serial Window UART #1**］を開き、アプリケーションの出力を表示します。

Debug ツールバーの［**Run**］ボタンをクリックするか、［**Debug**］→［**Run**］を選択して Hello プログラムを起動します。［Serial］**ウィンドウ**に「Hello World」と出力され、プログラムは無限ループに入ります。

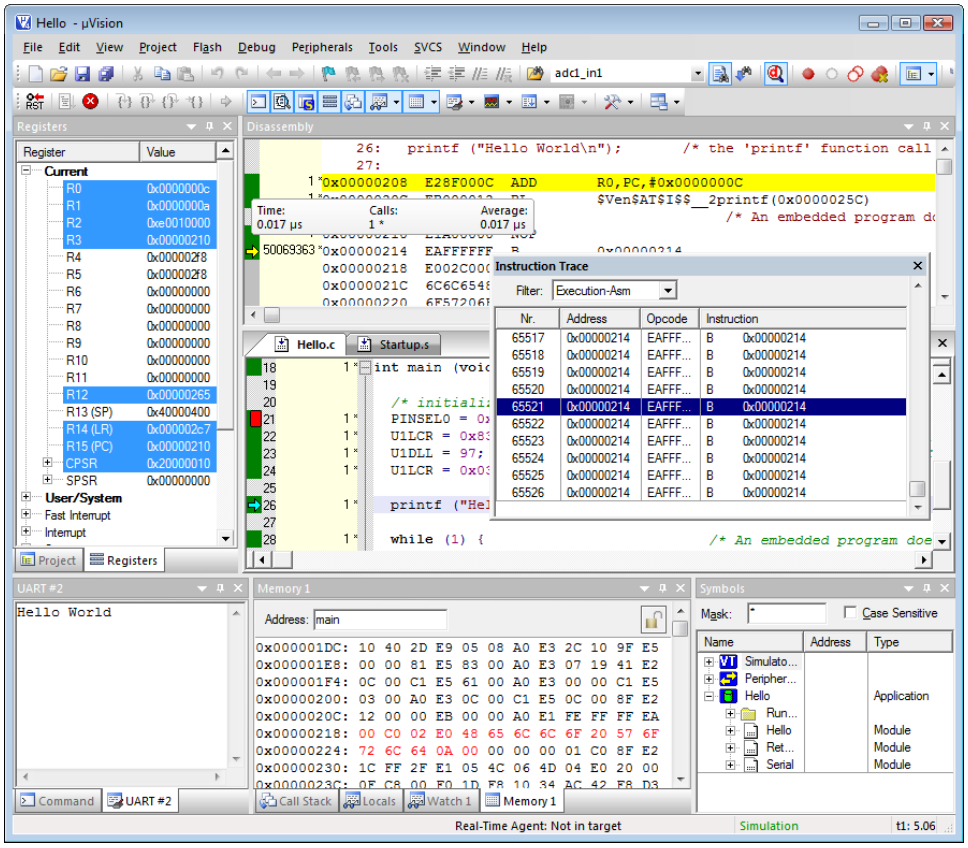
［**Stop**］ボタンをクリックしてプログラムを停止します。あるいは、［**Command**］**ウィンドウ**の**コマンドライン**で **Esc** キーを押します。

［**Insert/Remove Breakpoint**］コマンドを使用してブレークポイントを設定またはクリアします。

［**Reset**］コマンドをテストしてシミュレートされるマイクロコントローラをリセットします。プログラムがまだ実行されている場合は、最初のブレークポイントで停止します。

［**Step**］ボタンを使用してプログラムをシングルステップで実行します。次に実行される現在の命令には黄色の矢印が付きます。この黄色の矢印は、ステップごとに移動します。

デバッグ中、µVision には以下のレイアウトのデフォルト画面が表示されます。レイアウトを変更すると、µVision に自動的に保存され、次回デバッガを開いたときにそのレイアウトが使用されます。ただし、［**Window**］→［**Debug Restore Views…**］メニューを使用して保存しないと、変更したレイアウトを明示的に呼び出すことはできません。



ここで、新しい外観とナビゲーション機能に慣れて下さい。レジスタおよびメモリ領域の内容の変更を確認し、値を異なる形式で表示してみてください。時間を取って、このシンプルなサンプルプログラムを使用して µVision の機能を確認することをお勧めします。

Measure サンプルプログラム

Measure プログラムは、測候所やプロセス制御アプリケーションで使用されているような方法でアナログデータとデジタルデータを収集する単純なサンプルプログラムです。次の 3 つのソースファイル **GETLINE.C**、**MCOMMAND.C**、および **MEASURE.C** を使用します。

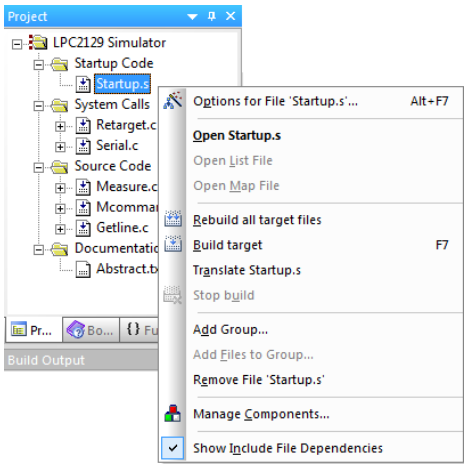
Measure プログラム[1] は、デジタルポートと A/D 入力から受け取るデータを記録します。1 ミリ秒から 60 分までに設定できるタイマにより、サンプルレートと間隔が制御されます。現在の時刻と入力チャネルからのすべてのデータが測定され、RAM バッファに保存されます。

以下の表には、各 Measure プログラムの場所を示しています。

アーキテクチャ	サンプルフォルダ
ARM	**\KEIL\ARM\EXAMPLES\MEASURE**
C166/XE166/XC2000	**\KEIL\C166\EXAMPLES\MEASURE**
8051	**\KEIL\C51\EXAMPLES\MEASURE**

Measure プロジェクトを開く

Measure プロジェクトを開始するには、必要なサンプルフォルダから **MEASURE.UVPROJ** プロジェクトファイルを開きます。



プログラムをコンパイルします。プロジェクト構造のインクルードファイルを有効または無効にし、［**Project**］ウィンドウの**コンテキストメニュー**を開き、［**Show Include File Dependencies**］を切り替えます。

Source Code グループには、アプリ

[1] µVision ではこのプログラムに必要なターゲットハードウェアをシミュレートするので、実際にはターゲットハードウェアまたは評価ボードは必要ありません。

ケーション関連の 3 つのソースコードファイルがあります。**System Calls** にはシリアル I/O およびシステムモジュールがあり、**Startup Code** グループにはスタートアップファイルが格納されています。

プロジェクトには、1 つ以上のターゲットを含めることができます。この機能により、プログラムのさまざまなバージョンをビルドできます。Measure プロジェクトには、シミュレータや評価ボードなどの異なるテスト環境用のいくつかのターゲットが含まれています。シミュレータターゲットのモデルを選択します。以下のファイルは、ソースコードで構成されています。

MEASURE.C このファイルには、main C 関数とタイマの割り込みサービスルーチンが含まれています。main 関数はすべてのペリフェラルを初期化し、コマンド処理を実行します。割り込みルーチンは、リアルタイムクロックとサンプリングを管理します。

MCOMMAND.C このファイルは、表示、時間、間隔のコマンドを処理します。これらの関数は main C 関数から呼び出されます。表示コマンドは、浮動小数点形式のアナログ値をリストし、0.00V と 3.00V の間の電圧を示します。

GETLINE.C このファイルには、シリアルポートから受信した文字のコマンドラインエディタが含まれます。

Measure プロジェクトのビルド

［**Project**］メニューと **Build ツールバー**には、プロジェクト内のファイルをコンパイルしてリンクするためのいくつかのコマンドがあります。

 プロジェクトワークスペースで選択したファイルをコンパイルするには、［**Translate File**］コマンドを使用します。

 最後にビルドおよびリンクした後に変更されたファイルをコンパイルするには、［**Build Target**］コマンドを使用します。


プロジェクト内のすべてのファイルをコンパイルしてリンクするには、［**Rebuild All Target Files**］コマンドを使用します。


実行中のビルドを停止するには、［**Stop Build**］コマンドを使用します。

［**Build Target**］コマンドを選択して、Measure プロジェクトのソースファイルをコンパイルしてリンクします。ビルドプロセスが完了すると、µVision の［**Command**］**ウィンドウ**にメッセージが表示されます。

ソースブラウザ

Measure プロジェクトは、完全なブラウザとデバッグ情報を生成するように設定されています。


File ツールバーまたは［**View**］メニューの［**Source Browse**］コマンドを使用して、プログラム変数とその他のオブジェクトについての情報を表示します。

Measure プロジェクトのテスト

Measure プログラムは、オンチップシリアルポートからのコマンドを受け入れるように設計されています。実際のターゲットハードウェアがある場合は、Hyperterm や別の端末プログラムを使用して、ボードと通信できます。ターゲットハードウェアがない場合、µVision を使用してハードウェアのすべての側面をシミュレートできます。例えば、µVision の［**Serial**］**ウィンドウ**はシリアル入力をシミュレートします。


Debug ツールバーまたは［**Debug**］メニューの［**Start/Stop Debug Session**］コマンドを使用して、µVision デバッガを開始します。

シリアルコマンドの使用

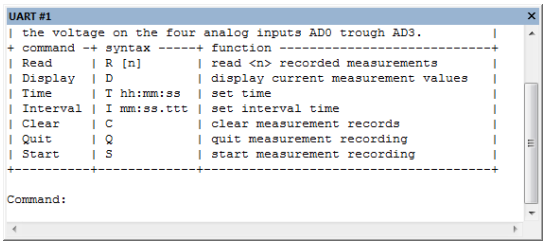
［**Serial**］ウィンドウの以下のコマンドを試して下さい。

アクション	コマンド	説明
Clear	C	測定記録バッファをクリアします。
Display	D	現在の時刻と入力値を継続的に表示します。
Interval	I *mm:ss.ttt*	測定記録の時間間隔を設定します。時間間隔は 0:00.001（1 ミリ秒）と 60:00.000（60 分）の間で指定する必要があります。
Quit	Q	測定記録を終了します。
Read	R [*count*]	保存された記録を表示します。表示するレコード数を指定します。カウントが指定されていない場合、すべてのレコードが送信されます。間隔が 1 秒より長い場合は、実行中にレコードを読み出すことができます。それ以外の場合は、記録を停止する必要があります。
Start	S	記録を開始します。指定した時間間隔でテータ入力が格納されます。
Time	T *hh:mm:ss*	現在の時刻を 24 時間形式で設定します。

シリアルインタフェースの使用

［**View**］メニューまたは **Debug ツールバー**からシリアル［**UART**］ウィンドウを開き、出力を表示します。

Measure プログラムの実行を開始する前に、［Serial］ウィンドウを開くと、コマンドを入力したり、プログラム出力を表示したりできます。



プログラムの実行

［Step］ボタンを使用して個々のコードのコマンドを実行します。**［Disassembly］ウィンドウ**がアクティブウィンドウの場合、デバッガはソースコードではなく、アセンブラ命令をステップスルーします。


現在の命令または高レベルのステートメント（次に実行されるもの）には黄色の矢印が付きます。この矢印は、ステップ実行するたび、新しい現在の命令行を反映して移動します。


Debug ツールバーまたは［**Debug**］メニューの［**Run**］コマンドを使用して、プログラムのデバッグを開始します。


［**Stop**］コマンドを使用してプログラムの実行を停止するか、［**Command**］**ウィンドウ**で **Esc** キーを押します。


Debug ツールバーまたは［**Debug**］メニューの［**Step Into**］コマンドを使用して、プログラムをステップスルーし、関数呼び出しにステップインします。


Debug ツールバーまたは［**Debug**］メニューの［**Step Over**］コマンドを使用して、プログラムをステップスルーし、関数呼び出しをステップオーバーします。

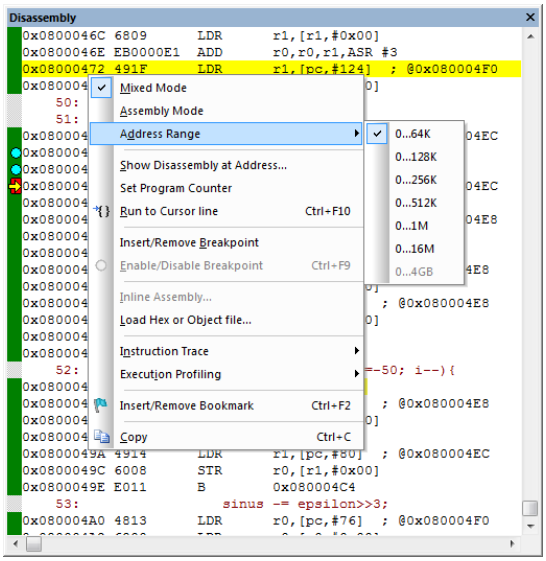

Debug ツールバーまたは［**Debug**］メニューの［**Step Out**］コマンドを使用して、現在の関数からステップアウトします。


Debug ツールバーまたは［**Debug**］メニューの［**Run To Cursor Line**］コマンドを使用して、強調表示した行に達するまでプログラムを実行します。

プログラムコードの表示

Debug ツールバーまたは［**View**］メニューの［**Disassembly Window**］コマンドを使用して、混合ソースとアセンブリコードを表示します。

さまざまなステップコマンドを、最初は［**Disassembly**］**ウィンドウ**で、次に［**Editor**］**ウィンドウ**で試して、デバッガの動作の違いを確認して下さい。

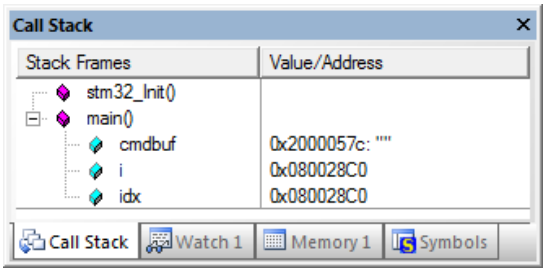


さまざまなコード行にマウスを移動しながら、ウィンドウの**コンテキストメニュー**を開きます。異なるオプションが表示されます。オプションは、ステートメントが実行可能かどうかに依存します。緑色やグレーでマークされた行と、色のない行があります。
メモリアドレスをポイントしながら**コンテキストメニュー**を開きます。

コールスタックの使用

Debug ツールバーまたは［**View**］メニューの［**Call Stack Window**］コマンドを使用します。

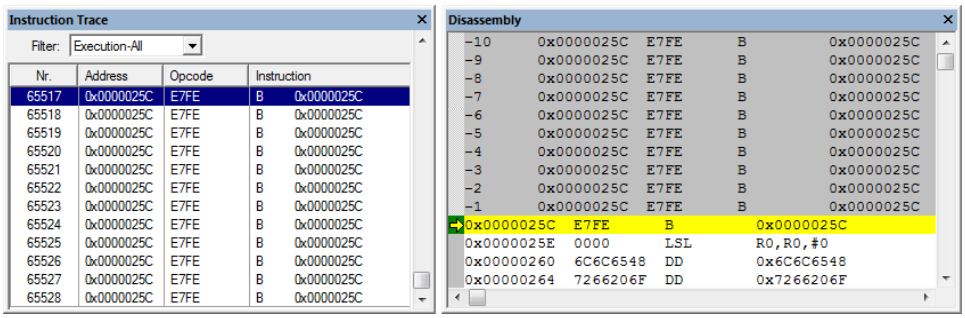
µVision により関数ネストが追跡され、［**Call Stack**］**ウィンドウ**にデータが表示されます。関数の行をダブルクリックすると、ソースコードにジャンプします。



トレースバッファの使用

プログラミングプロセスでは、ある状態を生じる状況を調査する場合があります。µVision デバッガにより、トレースメモリバッファに命令を記録できます。**デバッグモード**では、［**View**］→［**Trace**］→［**Show Records in Disassembly**］コマンドを使用してトレースバッファを確認できます。

Debug ツールバーまたは［**View**］→［**Trace**］→［**Instruction Trace Window**］メニューの［**Trace Windows**］コマンドを使用して、トレースバッファ内に格納されている実行済みの命令を表示します。



［**Disassembly**］**ウィンドウ**には、常にトレース情報が表示されますが、［**Instruction Trace**］**ウィンドウ**は ARM デバイスのみに有効です。

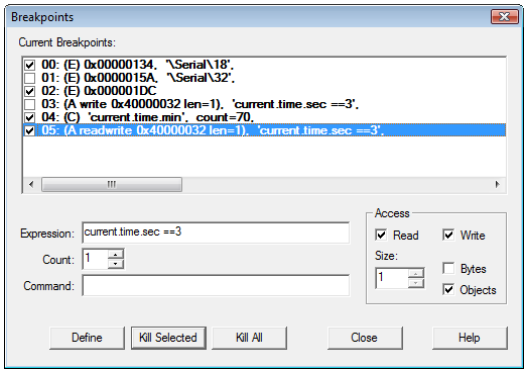
さらに、選択した命令のレジスタの内容を示す［**Registers**］**ウィンドウ**を確認します。

使用できるオプションを確認するには、［**Disassembly**］**ウィンドウ**の**コンテキストメニュー**を開きます。

［**Instruction Trace**］**ウィンドウ**でダブルクリックすると、［**Disassembly**］**ウィンドウ**に対応する命令が表示されます。

ブレークポイントの使用

µVision は、実行、アクセス、および複合ブレークポントをサポートしています。以下の例は、**current.time.sec** に値 3 が書き込まれたときにトリガされるブレークポイントの作成方法を示します。



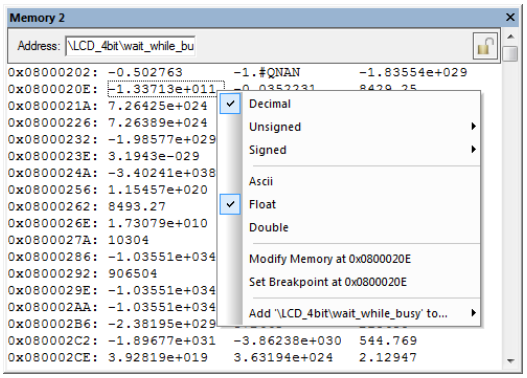
［**Debug**］→［**Breakpoints**］メニューから［**Breakpoints**］ダイアログを開きます。［**Expression**］に「**current.time.sec==3**」と入力し、［**Write**］チェックボックスをオンにします。これにより、プログラムが **current.time.sec** に値 3 を書き込んだときにブレークポイントがトリガされます。［**Define**］**ボタン**をクリックしてブレークポイントを設定します。ブレークポイント定義を再定義するには、定義をダブルクリックします。CPUを**リセット**して、ブレークポイントをテストします。これにより、**current.time.sec** に値 3 が書き込まれたときにブレークポイントがトリガされ、プログラムの実行が停止します。［**Debug**］**ウィンドウ**のプログラムカウンタ行には、ブレークポイントがトリガされた位置がマークされます。

メモリの内容の表示

Debug ツールバーまたは［**View**］メニューの［**Memory Window**］コマンドを使用して、メモリの内容を表示します。

［**Lock/Freeze**］アイコンを使用して、値が更新されないようにします。

µVision はさまざまな形式でメモリを表示し、4 つの異なる［**Memory**］**ウィンドウ**が用意されています。



内容を表示するには、開始**アドレス**を定義するか、［**Symbols**］**ウィンドウ**から［**Memory**］**ウィンドウ**にオブジェクトをドラッグアンドドロップします。

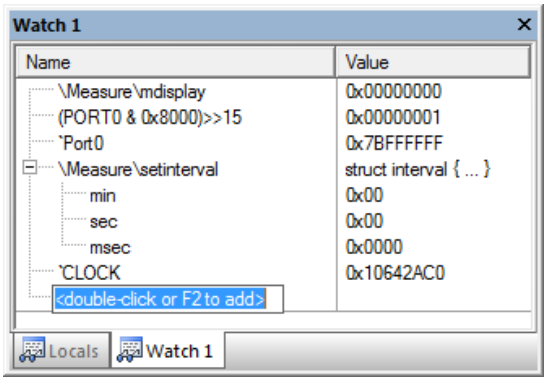
コンテキストメニューを開くと、形式の変更、メモリの変更、またはブレークポイントの設定が可能です。

変数の監視

µVision デバッガには2 つの［**Watch**］**ウィンドウ**があり、簡単に参照できるように変数、構造体、および配列を追加できます。［**Watch**］**ウィンドウ**は、各実行コマンドの終了時点で更新されます。［**View**］→［**Periodic Window Update**］メニューを有効にすると、プログラム実行中にこのウィンドウの内容が更新されます。

Debug ツールバーまたは［**View**］メニューの［**Watch Window**］コマンドを使用して、機能を起動するか、［**Locals**］を呼び出します。

［**Locals**］ページには、現在実行されている関数のローカルシンボルが表示されます。



［**Watch**］ページには、監視するプログラムオブジェクトが表示されます。［+］記号をクリックすると、構造体と配列が必要に応じて開きます。インデントされた行はネストを反映します。

［**Watch**］ページに変数を追加するには、いくつかの方法があります。

- ［**Watch**］**ウィンドウ**で、［Watch］ページの最後の行（**<double-click or F2 to add>**）を選択します。**F2** キーを押すか、この行をマウスでクリックします。監視する変数の名前を入力します。
- ［**Editor**］**ウィンドウ**で変数を選択し、**コンテキストメニュー**（右クリック）を開いて［**Add to Watch Window**］を選択します。
- ［**Output**］**ウィンドウ**の［**Command**］ページに「**WS**」（WatchSet）と入力し、続けて［**Watch**］**ウィンドウ**番号（1 または 2）、変数名の順に入力します。
- このウィンドウにオブジェクトをドラッグアンドドロップします。

［**Watch**］**ウィンドウ**から変数を削除するには、行を選択して **Del** キーを押すか、**コンテキストメニュー**を使用します。構造体および配列の個々の要素を個別に削除することはできません。

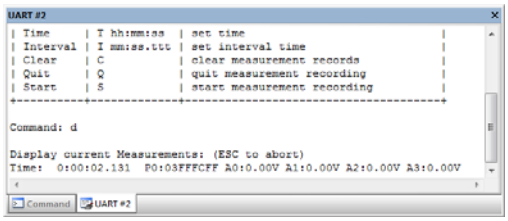
オンチップペリフェラルの表示と変更

Measure プログラムは、複数の I/O と A/D ポートからの入力を受け入れます。µVision デバッガを使用して、データを表示し、ペリフェラルを操作できます。入力に対して行った変更は各ペリフェラルのダイアログウィンドウに反映されます。［**Serial**］**ウィンドウ**で「**D**」を入力すると、出力と、入力値に適用された変更内容を監視できます。

シミュレートされた CPU を**リセット**します。

まだ実行されていない場合は、プログラムを**起動**します。

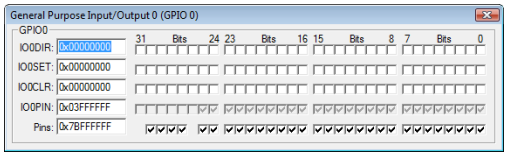
閉じている場合は、［**Serial**］**ウィンドウを開き**ます。

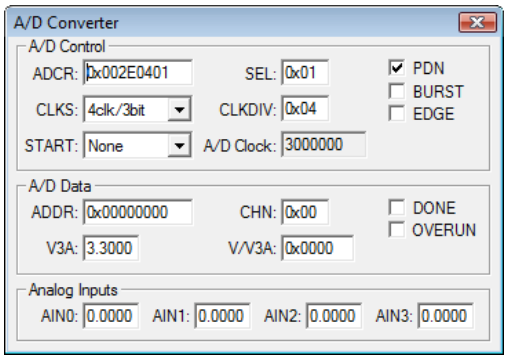


［**D**］コマンドを入力すると、Measure プログラムは常に時間、I/O ポート、および A/D 入力を更新します。I/O ポートおよび A/D コンバータチャネルからの入力は、［**Peripherals**］メニューから使用できるペリフェラルダイアログから制御できます。

［Peripheral］ウィンドウの使用

µVision デバッガには、I/O およびシリアルポート、A/D コンバータ、割り込み、タイマ、その他ほとんどのチップ固有のペリフェラルのウィンドウがあります。これらのウィンドウは、［**Peripherals**］メニューから開きます。





ウィンドウには、レジスタのステータスと、シミュレートされるデバイスのピンのステータスが表示されます。

［A/D Converter］ダイアログを開くと、A/D コントロールと A/D データのステータスが表示されます。アナログ入力の入力電圧を入力することができ、これが［Serial］ウィンドウに反映されます。

VTREG シンボルの使用

ペリフェラルダイアログのほか、仮想ターゲットレジスタ（VTREG）を使用して入力シグナルを変更することもできます。**［Command］ウィンドウ**では、VTREG シンボルに値を割り当てることができます。以下に例を示します。

```
PORT0=0xAA55       /* Set digital input PORT to 0xAA55   */
AIN1=3.3           /* Set analog input AIN1 to 3.3 volts */
```

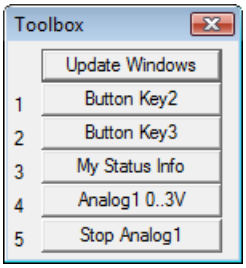
ユーザ関数とシグナル関数の使用

µVision デバッガでは、さらにプログラムで VTREG シンボルを使用できるようになる C 類似スクリプト言語がサポートされています。Measure プログラムには、デバッグシグナル関数が含まれています。この関数は、**［Toolbox］**のボタンを使用して呼び出すことができます。**［Command］ウィンドウ**を確認して下さい。詳細についてオンラインヘルプを表示するには、**F1** キーを押します。

［Toolbox］の使用

Debug ツールバーまたは**［View］**メニューの**［Toolbox］**コマンドを使用して、**［Toolbox］**ダイアログを表示します。

［Toolbox］には、デバッガコマンドまたはユーザ定義関数とリンクしたユーザ定義ボタンが含まれています。Measure プログラム用にいくつかのボタンが事前に定義されています。



［Analog0..3V］ボタンをクリックすると、シミュレートされたマイクロコントローラのアナログ入力 1 への入力を指定するユーザ定義シグナル関数が開始されます。

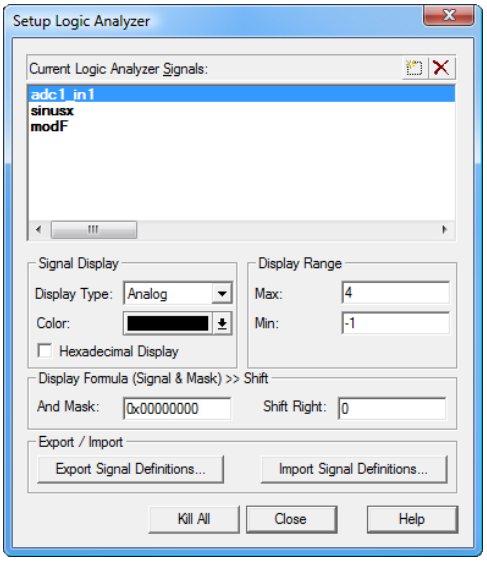
ロジックアナライザの使用

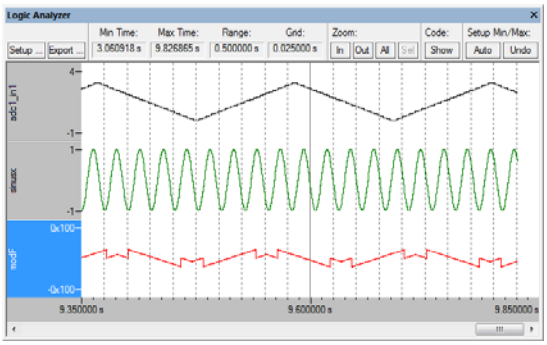
µVision デバッガには、プログラム実行中にシミュレートするシグナルと変数のトレースに使用できる、設定可能な**ロジックアナライザ**が用意されています。

Debug ツールバーまたは［**View**］**メニュー**から［**Logic Analyzer**］**ウィンドウ**を開きます。

シミュレートする A/D 入力シグナルなどのいくつかのシグナルを**ロジックアナライザ**に追加します。

［**Logic Analyzer**］**ウィンドウ**の［**Setup**］ボタンをクリックして［**Setup**］ダイアログを開きます。**Ins** キーを押して、「**ADC1_IN1**[1]」（A/D チャネル 1 の入力シグナルの名前）を入力し、［**Setup**］ダイアログを閉じます。単に［**Symbols**］ウィンドウから［**Logic Analyzer**］ウィンドウにオブジェクトをドラッグアンドドロップすることもできます。





[1] このスクリーンショットに使用されている追加信号は、Measure サンプルプログラムに組み込まれていません。

複雑な解析には、複数のシグナルを選択して記録できます。［**Export Signal Definitions…**］ボタンと［**Import Signal Definitions…**］ボタンを使用してシグナル定義を保存およびロードできます。

プログラムを実行し、［**Toolbox**］の［**Analog1 0..3**］ボタンを使用してアナログ入力 1 のシグナルの変更を開始します。このアナログ入力に変更内容が適用され、**ロジックアナライザ**に反映されます。

Traffic サンプルプログラム

Traffic プログラム[1] は、組み込みアプリケーションでどのようにリアルタイムオペレーティングシステムを使用できるかを示すサンプルです。このサンプルは、交通信号と歩行者信号の制御をシミュレートします。ラッシュ時には、停止信号により交差点の交通の流れが制御され、歩行者は一定時間ごと、または［**Push for Walk**］ボタンを押して横断します。ラッシュ時が過ぎると、交通信号は黄色に点滅します。

以下の項目を介して Traffic プログラムを操作します。

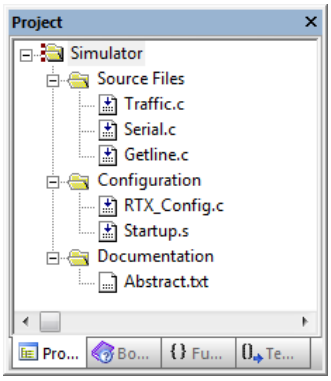
- シリアル［**UART**］ウィンドウ。現在の時刻と動作時間を変更できます。
- ［**Toolbox**］。［**Push for Walk**］ボタンをクリックして通りを横断できます。
- ［**Watch**］ウィンドウと［I/O Port］ダイアログ。交通信号と「歩く/止まれ」の歩行者信号の状態を監視できます。

以下の表には、各アーキテクチャの Traffic プロジェクトファイルの場所を示します。

アーキテクチャ	サンプルフォルダ
ARM	**\KEIL\ARM\RL\RTX\EXAMPLES\TRAFFIC**
C16x/XC16x	**\KEIL\C166\EXAMPLES\TRAFFIC**
8051	**\KEIL\C51\RTXTiny2\EXAMPLES\TRAFFIC**

[1] µVision ではこのプログラムに必要なハードウェアをシミュレートするので、ターゲットハードウェア、評価ボード、または交通信号は必要ありません。

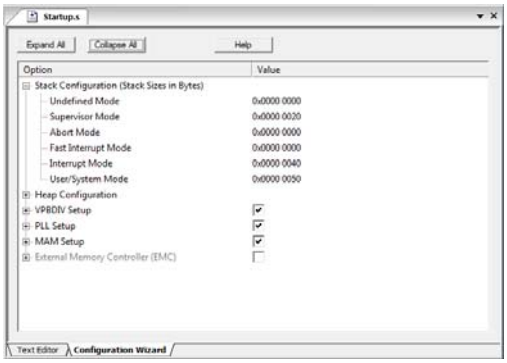
Traffic プロジェクトを開く



Traffic プロジェクトで作業を開始するには、適切なサンプルフォルダから **TRAFFIC.UVPROJ** プロジェクトファイルを開きます。

大部分の Keil のサンプルプロジェクトには、プログラムの特徴と目的を説明するテキストファイル **ABSTRACT.TXT** が含まれています。このファイルは［**Project**］ウィンドウにあります。

コンフィギュレーションウィザードの使用



µVisionには、スタートアップファイルとその他のコンフィギュレーションファイルの設定の選択に役立つ**コンフィギュレーションウィザード**が組み込まれています。

従来、これらのファイルは、ハードウェアコンフィギュレーションまたは優先度に応じて変更する必要があるマクロまたは定義を含むアセンブラまたはその他のソースファイルです。**コンフィギュレーションウィザード**により、このような選択プロセスが簡素化されます。

もちろんこれらのファイルは、［**Text Editor**］タブをクリックして、オリジナルソースの形式でいつでも編集できます。

Traffic プロジェクトのビルドとテスト

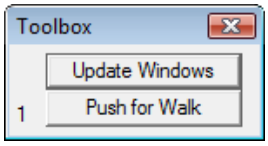
［**Rebuild**］コマンドを使用して、プロジェクトのすべてのファイルをコンパイルしてリンクします。

Traffic プログラムは、オンチップシリアルポート（µVision 内で完全にシミュレートされる）からコマンドを受け取り、交通信号（I/O ポートピンに接続されている）に出力を表示するように設計されています。

［Toolbox］の使用

Debug ツールバーまたは［**View**］メニューの［**Toolbox**］コマンドを使用して、［Toolbox］ダイアログを表示します。

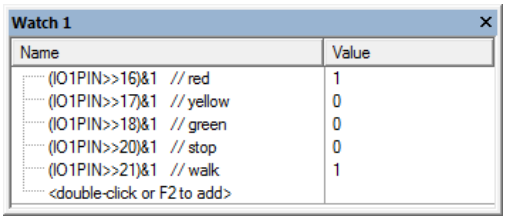
［**Toolbox**］に［**Push for Walk**］ボタンがあります。このボタンをクリックして、通りを横断する歩行者をシミュレートし、［**Watch**］**ウィンドウ**で「止まれ」と「歩く」の信号が変わるのを監視します。



［Watch］ウィンドウの使用

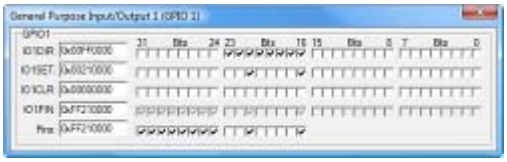
Debug ツールバーまたは［**View**］メニューの［**Watch Window**］コマンドを使用します。さらに［**Call Stack**］**ウィンドウ**も開きます。

歩行者用交通信号のステータスは、事前に定義されたウォッチ式を使用して［**Output**］**ウィンドウ**の［**Watch 1**］ページに表示されます。

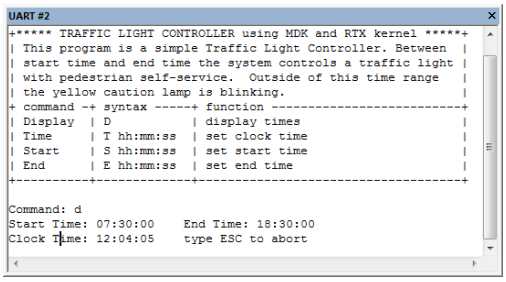


I/O ポートの使用

［**Peripherals** ］メニューから選択できる［I/O Port］ダイアログでも、交通信号の行を表示できます。



［Serial］ウィンドウの使用



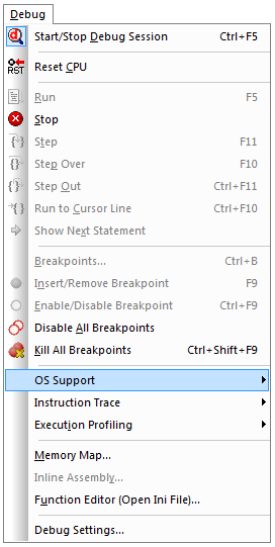
［**Serial** ］ウィンドウには情報が表示され、交通信号の時刻と動作時間を変更できます。黄色に点滅する信号を確認するには、現在の時刻をラッシュ時以外に設定します。［**Watch**］ウィンドウをチェックして、動作の変化を監視します。

以下の表にリストされたシリアルコマンドを使用できます。これらのコマンドはプレーン ASCII テキストです。各コマンドは、改行復帰で終わる必要があります。

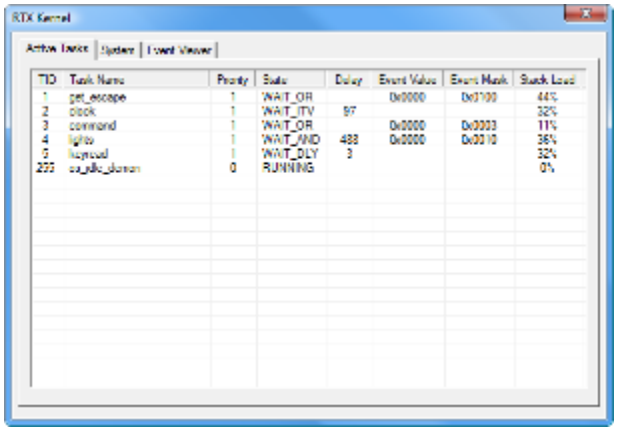
アクション	コマンド	説明
Display	D	現在の時刻、および動作の開始時間と終了時間を表示します。**Esc** キーを押すと表示モードが終了します。
Set Current Time	T *hh:mm:ss*	交通信号の現在の時刻を設定します。現在の時刻が開始時間と終了時間で指定された動作時間内であれば、交通信号は通常どおりに動作します。現在の時刻が動作時間外の場合、交通信号は黄色のライトが点滅します。
Set Start Time	S *hh:mm:ss*	通常動作の開始時間を設定します。
Set End Time	E *hh:mm:ss*	通常動作の終了時間を設定します。

カーネル認識デバッグ情報の表示

µVision シミュレータを使用すると、リアルタイムオペレーティングシステムで作成されたアプリケーションを実行してテストできます。リアルタイムアプリケーションは他のプログラムと同じようにロードされます。デバッグには特別なコマンドやオプションは必要ありません。



カーネル認識ウィンドウを開くには、［**Debug**］→［**OS Support**］メニューを使用します。



Blinky サンプルプログラム

Blinky プログラムは、評価ボードの LED を点滅するサンプルアプリケーションです。LED の点滅により、ターゲットハードウェアに正常にプログラムがロードされ、実行されていることを簡単に確認できます。

Blinky プログラムはボード固有のアプリケーションであり、ボードが異なると、このプログラムによってその他のボード固有の機能が表示される場合があります。詳細については、ボードのマニュアルを参照して下さい。

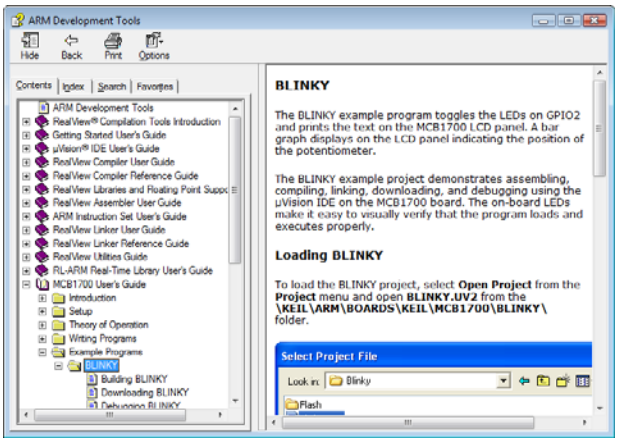
Blinky プロジェクトを開く

［**Project**］→［**Open Project**］メニューを選択し、以下のサブフォルダからそれぞれの **BLINKY.UVPROJ** プロジェクトを選択します。

アーキテクチャ	サンプルフォルダ
ARM	**\KEIL\ARM\BOARDS\ベンダ\ボード名\BLINKY**
C166/XE166/XC2000	**\KEIL\C166\BOARDS\ボード名\BLINKY**
8051	**\KEIL\C51\EXAMPLES\BLINKY**

各プロジェクトには、特定のボード用の Blinky プログラムの使用方法について説明する **ABSTRACT.TXT** ファイルが含まれています。

ボードの**ユーザガイド**にも Blinky プログラムの詳細な説明があります。

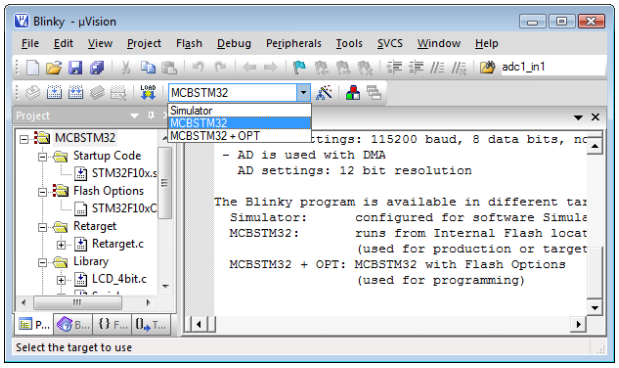


Blinky プロジェクトのビルド

このプロジェクトには、次のような複数のターゲットが含まれている場合があります。

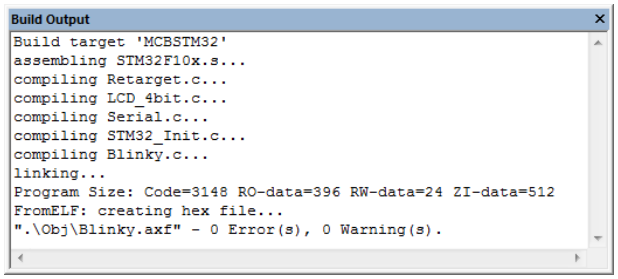
- **シミュレータ**：実際のターゲットハードウェアを使用せずにコードをデバッグするコンフィギュレーションです。
- **ボード固有のターゲット**：実際のターゲットハードウェアにプログラムをダウンロードしてテストするコンフィギュレーションです。

ハードウェアでテストする場合は、**ボード固有のターゲット**を選択して下さい。



Build ツールバーの［**Rebuild**］コマンド、または［**Project**］→［**Rebuild all target files**］メニューを使用して、すべてのプロジェクトファイルをコンパイルしてリンクします。

実行可能ファイルが出力フォルダに配置され、ダウンロードできるようになります。



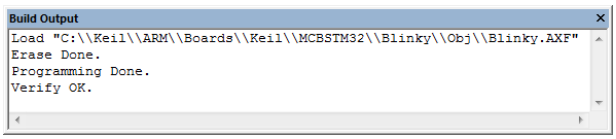
Blinky プログラムのダウンロード

［Download］コマンドを初めて使用する前に、［**Project**］→［**Options for Target**］→［**Utilities**］ダイアログの［Flash］オプションを確認します。または、［**Flash**］→［**Configure Flash Tools…**］メニューを使用して、このダイアログを開くこともできます。ボードを PC に接続します。

µVision IDE の設定を完了すると、［**Flash**］→［**Download**］メニューで Flash プログラミングに指定したアダプタが使用されます。

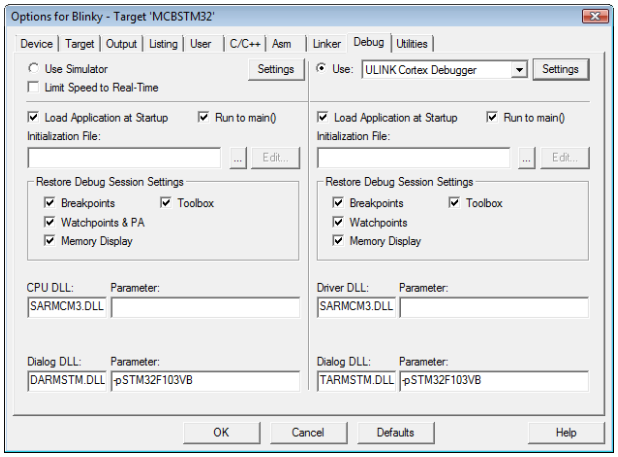
［**Download to Flash**］ツールバーボタンをクリックするか、［**Flash**］→［**Download**］メニューを使用して、ターゲットハードウェアにアプリケーションプログラムをフラッシュします。

プログラムがダウンロードされてターゲットハードウェアで正常に実行されると、同時に LED が点滅します。



Blinky プログラムのデバッグ

デバッガのコンフィギュレーション設定を確認するには、［**Options for Target**］→［**Debug**］ダイアログを開きます。



- ［**Use**］ラジオボタンをオンにして、ドロップダウンリストから該当するデバッグドライバを選択します。
- ［Load Application at Startup］と［Run to main()］をオンにします。
- ［**Settings**］ボタンをクリックしてドライバのコンフィギュレーションを確認します。

Debug ツールバーの［**Start/Stop Debug Session**］をクリックするか、［**Debug**］→［**Start/Stop Debug Session**］メニューを開き、アプリケーションのデバッグを開始します。

［**Step One Line**］－ステップコマンドを使用してターゲットハードウェアでアプリケーションをデバッグします。

［**Reset**］－デバッグ中にマイクロコントローラをリセットします。

［**Run**］－プログラムを実行して評価ボードの LED を点滅します。

［**Stop**］－プログラムの実行を停止します。

［**Show Current Statement**］－実行されるコードの次のステートメントを表示します。

用語集

ASCII
American Standard Code for Information Interchange
コンピュータにより使用されるコードセットで、数字、文字、句読点、その他の特殊記号を表します。最初の 128 文字は標準化されています。残りの 128 文字は、実装により定義されます。

CAN
Controller Area Network
バス標準。自動車用に設計されていますが、他の産業でも使用されています。これによりホストコンピュータがなくても、マイクロコントローラとデバイスが相互に通信できます。

CMSIS
Cortex Microcontroller Software Interface Standard
Cortex-Mx プロセッサ用のベンダに依存しないハードウェア抽象レイヤ。これにより、インタフェース接続したペリフェラル、リアルタイムオペレーティングシステム、およびミドルウェアで、プロセッサに対して一貫したスケーラブルでシンプルなソフトウェアインタフェースが実現し、ソフトウェアの再利用を簡素化し、新しいデバイスの開発期間を短縮できます。

CRC
Cyclic Redundancy Check（巡回冗長検査）
送信または格納中の偶発的なデータ変更を検出するための機能の一種

FPGA
Field-Programmable Gate Array
製造後に顧客が設定できる半導体デバイス

GPIO
General Purpose Input/Output（汎用入出力）
外部とデジタル形式で相互作用するためにマイクロコントローラデバイスで使用できるインタフェース。GPIO は通常、8 ピン、16 ピン、または 32 ピンのグループに配置されています。GPIO ポートのピンは、入力または出力として個別に設定できます。

ICE

In-Circuit-Emulator（インサーキットエミュレータ）
組み込みシステムのソフトウェアのデバックに使用するハードウェアデバイス。ハードウェアレベルの実行制御とブレークポイント機能を提供します。一部の ICE には、最新のマイクロコントローライベントを格納するトレースバッファが備わっています。

JTAG

Joint Test Action Group
標準テストアクセスポートおよびバウンダリスキャンアーキテクチャと呼ばれる IEEE 1149.1 標準の通称。JTAG は、多くの場合マイクロコントローラのデバッグポートまたはプローブポートとして使用され、デバイスメモリとのデータの送受信が可能になります。

LIN

Local Interconnect Network
現在の自動車ネットワークアーキテクチャ内で使用される車両バス標準またはコンピュータネットワークバスシステム。LIN バスは、小型の低速ネットワークシステムで、CAN バスの低価格なサブネットワークとして使用されます。

MDI

同じアプリケーションから複数のドキュメントを開くことのできるアプリケーション。アプリケーションの別のインスタンスを故意に起動する必要はありません。

MISRA

Motor Industry Software Reliability Association
組み込みシステムとの関連においてコードの安全性、移植性、および信頼性を重視して、C/C++ プログラミング言語のソフトウェア開発標準を策定するフォーラム

Monitor

8051 デバイスと C166 デバイス用のプログラム。ターゲットマイクロコントローラにロードすることで、高速ソフトウェアダウンロードによるデバッグと迅速な製品開発をサポートします。

OCDS
On Chip Debug Support
Infineon 166 デバイスのハードウェアエミュレーション機能を提供するデバッグポート

Thumb、Thumb2
ARM デバイスと Cortex デバイスの命令セット。「**命令セット**」を参照して下さい。

UART
Universal Asynchronous Receiver/Transmitter
個々の IC または IC の一部。シリアル通信に使用されます。

アセンブラ
アセンブリ命令（ニーモニック）をオペコードに変換し、メモリ位置やその他のエンティティのシンボル名を解決することで、オブジェクトコードを作成するコンピュータプログラム。アセンブリ言語で記述され、アセンブラにより変換されたプログラムは、メモリにロードして実行できます。

インクルードファイル
#include プリプロセッサディレクティブを使用してソースファイルに組み込まれるテキストファイル

オブジェクト
検証可能なメモリ領域。通常、変数または関数に関連するメモリ領域を指す場合に使用されます。

オブジェクトファイル
コンパイラにより作成されるこのファイルには、マシンコードフォーマットの一連の命令である、編成されたオブジェクト群が含まれますが、実行時に使用するデータ（再配置情報、スタック展開情報、コメント、リンク用の変数と関数の名前、およびデバッグ情報）が含まれる場合もあります。

オペコード

オペコード（演算コード）は、実行する演算を指定するマシン言語命令の一部です。これらの仕様と形式は、プロセッサの命令セットアーキテクチャに示されています。

コンパイラ

C/C++ などの高レベルプログラミング言語から、アセンブリ言語やマシンコードなどの低レベル言語にソースコードを変換するプログラム。コンパイラは通常、字句解析、前処理、解析、意味解析、コード生成、コードの最適化などの操作を実行します。µVision は、C/C++ コンパイラを実装します。

シミュレーション

実際のものまたはプロセスの模擬です。µVision では、実際のハードウェアを使用しないで組み込みアプリケーションを作成するためにシミュレーションを使用します。選択したシステムの主要な特性と動作を再現して、最適化、安全なエンジニアリング、テスト、およびデバッグを実施できます。

スタック

スタックポインタが間接的にアクセスするメモリ領域。動的に拡大および縮小してローカル関数データを保持します。スタックの項目は、LIFO（後入れ先出し）に従って削除されます。

デバッガ

ソフトウェアをテストするコンピュータプログラム。デバッガでは、ステップごとにプログラムを実行したり（シングルステップ）、ブレークポイントにより何らかのイベント発生時にプログラムを停止して現在の状態を調べたり、変数の値を追跡したりする高度な関数が使用されます。

トークン

プログラミング言語で名前またはエンティティを表す基本シンボル

マクロ

特定の入力シーケンスをどのように出力シーケンスにマップするかを指定した規則またはパターンを定義します。

命令セット
命令セット、すなわち命令セットアーキテクチャ（ISA）は、マイクロコントローラアーキテクチャのプログラミングに関連する部分で、ネイティブデータ型、命令、レジスタ、アドレス指定モード、メモリアーキテクチャ、割り込みと例外の処理、および外部 I/O が含まれます。ISA には、特定のマイクロコントローラにより実装されるネイティブコマンドであるオペコードセットの仕様が含まれます。

メモリモデル
関数の引数およびローカル変数に使用するメモリ領域を指定する定義。

ライブラリ
関連性のあるいくつかのオブジェクトモジュールを格納するファイル。リンカにより、ライブラリからモジュールを抽出して、オブジェクトファイルのビルドに使用できます。

リンカ
コンパイラによって作成され、ライブラリとオブジェクトを 1 つの実行可能プログラムに組み合わせたプログラム

リント
C/C++ コードのバグ、障害、矛盾、移植性、およびコードが MISRA に準拠しているかどうかをチェックするツール

索引

記号

数字

A

B

C

D

F

あ

い

う

お

か

く

こ

さ

し

す

せ

そ

た

ち

つ

て

と

は

ひ

ふ