参考資料

JAJU059A

Code Composer Studio™ v5.2 ユーザーズ・ガイド MSP430™版ユーザーズ・ガイド

目次

SLAU157U 翻訳版

最新の英語版資料
http://www.ti.com/lit/slau157

はじめに

最初にお読みください

このマニュアルについて

このマニュアルでは、超低消費電力マイクロコントローラMSP430™と併用してTexas Instruments™ Code Composer Studio™ v5.2 (CCS v5.2)を使う方法について説明します。このユーザーズ・ガイドでは、Windows版のCode Composer Studioのみに焦点を当てます。Linux版のセットアップはWindows版と同じであるため、別個の記載はありません。

このマニュアルの使用方法

「開発の前に(Get Started Now!)」章の手順を読んで実行してください。この章では、ソフトウェアのインストール手順を提示し、デモンストレーション・プログラムを実行する方法を説明します。開発ツールを短時間で簡単に使用できることを理解していただいた後に、このマニュアルを通読することをお勧めします。

このマニュアルでは、ソフトウェア開発環境のセットアップと基本的な動作のみを説明しますが、MSP430マイクロコントローラの完全な解説や、開発ソフトウェアおよびハードウェア・システムの完全な説明を提供するわけではありません。これらの項目の詳細については、セクション1.3「CD-ROMとウェブで入手できる重要なMSP430™関連資料」に記載された、該当するTIの関連資料の一覧を参照してください。

このマニュアルの内容は、Texas InstrumentsのMSP-FET430UIF、MSP-FET430PIF、eZ430 開発ツール・シリーズに適用できます。

上記のツールには、パッケージングの時点で入手可能な最新の資料が付属しています。最新の資料(データ・シート、ユーザーズ・ガイド、ソフトウェア、アプリケーション情報等) については、TI MSP430のウェブ・サイト(www.ti.com/msp430)をご覧になるか、最寄りのTIの営業所にお問い合わせください。

注意と警告についての情報

このドキュメントには、注意(Caution)と安全上の警告(Warning)が記載されている場合があります。

注意

これは注意の記述例です。
注意は、お使いのソフトウェアや機器が損傷するおそれのある場合を説明しています。

安全上の警告

これは警告の記述例です。
警告は、身体に危害が及ぶおそれのある場合を説明しています。

注意や安全上の警告に含まれる情報は、機器の損傷や身体の負傷からユーザーを保護するために提供されています。各注意や警告を注意してお読みください。

Texas Instruments から発行されている関連資料

CCS v5.2の関連資料

MSP430™ Assembly Language Tools User's Guide, (文書番号) SLAU131
MSP430™ Optimizing C/C++ Compiler User's Guide, (文書番号) SLAU132

MSP430™ 開発ツールの関連資料

MSP430™ Hardware Tools User's Guide, (文書番号) SLAU278
eZ430-F2013 Development Tool User's Guide, (文書番号) SLAU176
eZ430-RF2480 User's Guide, (文書番号) SWRA176
eZ430-RF2500 Development Tool User's Guide, (文書番号) SLAU227
eZ430-RF2500-SEH Development Tool User's Guide, (文書番号) SLAU273
eZ430-Chronos™ Development Tool User's Guide, (文書番号) SLAU292
MSP-EXP430G2 LaunchPad Experimenter Board User's Guide, (文書番号) SLAU318

MSP430デバイスのデータ・シート

MSP430x1xx Family User's Guide, (文書番号) SLAU049
MSP430x2xx Family User's Guide, (文書番号) SLAU144
MSP430x3xx Family User's Guide, (文書番号) SLAU012
MSP430x4xx Family User's Guide, (文書番号) SLAU056
MSP430x5xx and x6xx Family User's Guide, (文書番号) SLAU208

CC430デバイスのデータ・シート

CC430 Family User's Guide, (文書番号) SLAU259

サポートが必要な場合は

MSP430 マイクロコントローラとFET開発ツールのサポートは、Texas Instruments Product Information Center (PIC)から提供されています。PICの連絡先は、TIのウェブ・サイト www.ti.com/supportに記載されています。Code Composer Studio 専用のWiki ページ(FAQ)が利用可能であり、またMSP430とCode Composer Studio v5.2用のTexas Instruments E2E Community サポート・フォーラムでは同業のエンジニアやTIエンジニア等の専門家と自由な情報交換ができるようになっています。各デバイス特有の補足情報は、MSP430のウェブ・サイトから入手できます。

FCC(米連邦通信委員会)警告

この機器は、実験室のテスト環境のみで使用することが意図されています。高周波エネルギーの生成と使用を行い、高周波エネルギーを放射する可能性がありますが、(無線周波妨害からの適切な保護を提供するために策定されている)FCCルールのパート15のサブパートJに準じたコンピュータ機器の制限事項を順守しているかどうかのテストは受けていません。実験室のテスト環境以外でこの機器を動作させると、無線通信の妨害が発生する可能性があります。この場合はユーザー側が、妨害を解消するために必要と思われるあらゆる措置を、ユーザー自身の負担で講じる必要があります。

第 1 章 開発の前に(Get Started Now!)

この章ではソフトウェアのインストール手順を説明し、デモンストレーション・プログラムの実行方法を紹介します。

1.1 ソフトウェアのインストール

Code Composer Studio™ v5.2 (CCS)をインストールするには、DVDからsetup_CCS_x.x.x.x.exe を実行します。ダウンロードでCCSパッケージを入手した場合は、setup_CCS_x.x.x.x.exeを実行する前に、zipアーカイブ全体を確実に展開するようにしてください。画面に表示される手順に従ってインストールを進めます。USB JTAGエミュレータ用のハードウェア・ドライバ(MSP-FET430UIFシリーズとeZ430シリーズ)は、CCSのインストール時に自動的にインストールされます。パラレル・ポートFET用のドライバ (MSP-FET430PIF)はデフォルトではインストールされませんが、インストール・プロセス中に手動で選択することが可能です。

注: **MSP-FET430PIF (パラレル・ポート・エミュレータ)のサポート**
パラレル・ポート・インターフェイス製品MSP-FET430PIFに対応するドライバとIDEの部品(components)は、デフォルトではインストールされていないため、CCS v5.2のインストール・プロセス中に手動で選択する必要があります。
setup_CCS_x.x.x.x.exeを実行する前に、zipアーカイブ setup_CCS_x_x_x.zip ファイル全体を展開してください。

表1-1 システムの要件

	推奨されるシステム要件	最小システム要件
プロセッサ	デュアル・コア	1.5 GHz
RAM	2 GB	1 GB
ディスクの空き容量	2 GB	300MB(インストール中に選択された機能によって異なります)
オペレーティング・システム	次のいずれか Microsoft® Windows® XP SP2 (32 または 64 ビット) Windows Vista® SP1 (32 または 64 ビット) Windows 7® (32 または 64 ビット)	次のいずれか Microsoft® Windows® XP SP2 (32 または 64 ビット) Windows Vista® (32 または 64 ビット) Windows 7® (32 または 64 ビット)

1.2 LED を点滅させる

このセクションでは、C言語を学ぶ時に最初に書くプログラム "Hello world!" に当たるものをFET上で動作させてみます。CCS v5.2には、完全に事前構築されたプロジェクトの他に、CとASMのコード・ファイルが含まれています。LEDを点滅させるアプリケーションを開発し、FETにダウンロードして実行する方法の手順を次に示します。

1. Code Composer Studioを起動します。(Start → All Programs → Texas Instruments → Code Composer Studio → Code Composer Studioと選択)
2. 新規プロジェクトを生成します。(File → New → CCS Projectと選択)
3. プロジェクト名を入力し、Device Variant (デバイスの種類)を選択します。
4. MSP-FET430UIFやeZ430開発ツール等のUSBフラッシュ・エミュレーション・ツールを使用している場合は、それらのツールがデフォルトで構成済みになっている必要があります。MSP-FET430PIF LPTインターフェイスを使用している場合(かつ、インストール中にMSP430パラレル・ポート・ツールのサポートが選択された場合)は、TI MSP430 LPTxをユーザー側で選択する必要があります。

5. プロジェクトのテンプレートと例のセクションにある "Blink The LED(LEDを点滅させる)" という基本的な例を選択します。
6. Finishをクリックします。

注: 事前定義された例は、大部分のMSP430ボードで使用可能です。ただし、特定のMSP430x4xxボードでは、LED接続用にPort P5.0を使用します。また、MSP430L092ボードでは別のコード例が必要になります。このようなコード例も入手可能になっています。詳細については、表1-2を参照してください。

表 1-2 コード例

MSP430 デバイス	コード例
MSP430x1xx デバイス・ファミリ	<...>¥msp430x1xx¥C-Source¥msp430x1xx.c
MSP430x2xx デバイス・ファミリ	<...>¥msp430x2xx¥C-Source¥msp430x2xx.c
MSP430x4xx デバイス・ファミリ	<...>¥msp430x4xx¥C-Source¥msp430x4xx.c
MSP430x5xx デバイス・ファミリ	<...>¥msp430x5xx¥C-Source¥msp430x5xx.c
MSP430x6xx デバイス・ファミリ	<...>¥msp430x6xx¥C-Source¥msp430x6xx.c
MSP430L092	<...>¥msp430x5xx¥C-Source¥msp430l092.c

7. コードをコンパイルし、アプリケーションをターゲット・デバイスにダウンロードします。(Run → Debug (F11)と選択)
8. アプリケーションを開始できるようになります。(Run → Resume (F8)と選択するか、ツールバーのPlayボタンをクリック)

CCSデバッガがデバイスと通信できない場合は、FAQのデバッグ #1を参照してください。

これで、MSP430アプリケーションのビルドとテストが無事に完了しました。

事前定義されたプロジェクトは<Installation Root>\ccsv5\ccs_base\msp430\examples\example projects に入っており、次の手順で選択してインポートできます。Project → Import Existing CCS/CCE Eclipse Project

1.3 CD-ROM とウェブで入手できる重要な MSP430™ 関連資料

MSP430とCCS v5.2の主要な情報源は、デバイス固有のデータ・シートとユーザー・ガイドです。製造(production)時に入手可能な、これらの関連資料の最新バージョンは、このツールに同梱されたCD-ROMで提供されています。MSP430のウェブ・サイト(www.ti.com/msp430)には、これらの関連資料の最新版が記載されています。

第 2 章 開発フロー

この章では、Code Composer Studio (CCS)を使用してアプリケーション・ソフトウェアを開発する方法と、そのソフトウェアをデバッグする方法について説明します。

2.1 Code Composer Studio (CCS)を使用する

以降のセクションでは、CCSの使用方法の簡単な概要を説明します。CCSを使用した、アセンブリまたはC言語でのソフトウェア開発フローの完全な説明については、MSP430 Assembly Language Tools User's Guide (SLAU131)およびMSP430 Optimizing C/C++ Compiler User's Guide (SLAU132)を参照してください。

2.1.1 新規のプロジェクトを生成する

このセクションでは、アセンブリまたはCのプロジェクトを新規に作成する方法と、MSP430上でアプリケーションをダウンロードして実行するための手順を、順を追って説明します(セクション2.1.2「プロジェクトの設定」を参照)。また、MSP430 Code Composer Studioのヘルプ機能を使用すると、より包括的なプロセス概要をご覧いただけます。

1. CCSを起動します(Start → All Programs → Texas Instruments → Code Composer Studio → Code Composer Studio)。

2. 新規プロジェクトを生成します(File → New → CCS Project)。プロジェクト名を入力し、nextをクリックして、Device FamilyをMSP430に設定します。

3. 適切なデバイスの種類(variant)を選択します。アセンブリ専用の(assembly only)プロジェクトの場合は、"Project template and examples" セクションの"Empty Assembly-only Project"を選択してください。

4. MSP-FET430UIFやeZ430開発ツール等のUSBフラッシュ・エミュレーション・ツールを使用している場合は、それらがデフォルトで構成済みになっている必要があります。MSP-FET430PIF LPTインターフェイスを使用している場合(かつ、インストール中にMSP430パラレル・ポート・ツールのサポートが選択された場合)は、TIのMSP430 LPTxをユーザー側で選択する必要があります。

5. Cプロジェクトの場合は、これでセットアップ完了となります。main.cが表示され、コードが入力できるようになります。アセンブリ・プロジェクトの場合は、新規のソース・ファイルを生成する必要があります(File → New → Source File)。ファイル名を入力した後、忘れずに .asmというサフィックスを付加してください。そうせずに、プロジェクト用に既存のソース・ファイルを使用したい場合は、Project → Add Files... とクリックして、関心対象のファイルを表示します。ファイル上で1度クリックした後、Open をクリックするか、ファイル名をダブルクリックすると、ファイルがプロジェクト・フォルダに追加されます。

6. Finish をクリックします。

7. プログラム・テキストをファイルに入力します。

注: コード開発を簡素化するUse .h ファイル
CCSには、デバイスのレジスタとビット名を定義している各デバイス用のファイルが付属しています。これらのファイルを使用するとプログラム開発作業が大幅に簡素化できるため、使用が推奨されます。ターゲット・デバイスに対応する.hファイルをインクルードするには、Cの場合は #include <msp430xyyy.h> という行、アセンブリ・コードの場合は .cdecls C,LIST,"msp430xyyy" という行を追加します。ここで、xyyy にはMSP430の部品番号を指定します。

8. プロジェクトをビルドします(Project → Build Project)。

9. アプリケーションのデバッグを行います(Run → Debug (F11))。これによりデバッガが起動されます。デバッガではターゲットの制御権を得てターゲット・メモリを消去し、ターゲット・メモリにアプリケーションを書き込み、ターゲットをリセットします。
   デバッガがデバイスと通信できない場合は、FAQ のデバッグ #1 を参照してください。

10. Run → Resume (F8)をクリックして、アプリケーションを起動します。

11. Run → Terminate をクリックして、アプリケーションを停止し、デバッガを終了します。CCS は、C/C++ 表示(コード・エディタ)に自動的に戻ります。

12. File → Exit をクリックして、CCS を終了します。

2.1.2 プロジェクトの設定

CCSの構成に必要な設定は多数で詳細にわたります。ほとんどのプロジェクトには、デフォルトの工場出荷時設定を使用してコンパイルやデバッグを行うことが可能です。プロジェクトの設定には、Project → Properties for the active project とクリックしてアクセスします。次に示すプロジェクト設定は、推奨または必須となります。

- デバッグ・セッション用のターゲット・デバイスを指定します(Project → Properties → General → Device → Variant)。対応するリンカ・コマンド・ファイルとランタイム・サポート・ライブラリは自動的に選択されます。
- Cプロジェクトのデバッグをより容易にするには、最適化をディセーブルにします(Project → Properties → Build → MSP430 Compiler → Optimization → Optimization level off)。
- Cのプリプロセッサ用の検索パス(search path)を指定します(Project → Properties → Build → MSP430 Compiler → Include Options)。
- 使用されている任意のライブラリ用の検索パスを指定します(Project → Properties → Build → MSP430 Linker → File Search Path)。
- デバッガのインターフェイスを指定します(Project → Properties → General → Device → Connection)。TI MSP430 LPTxをパラレルFETのインターフェイスとして選択するか、TI MSP430 USBxをUSBのインターフェイスとして選択します。
- オブジェクト・コードのをダウンロードする前に、メイン・メモリと情報メモリの消去をイネーブルにします(Project → Properties → Debug → MSP430 Properties → Download Options → Erase Main and Information Memory)。
- 正しいスタンドアロン動作が確実に行われるようにするには、ソフトウェア・ブレークポイントをディセーブルにします(Project → Properties → Debug → MSP430 Properties → Enable Software Breakpoints : チェックをはずす)。ソフトウェア・ブレークポイントがイネーブルになっている場合は、ターゲットの接続中に各デバッグ・セッションが正しく終了するようにしてください。正しく終了しない場合は、デバイス上のアプリケーションにソフトウェア・ブレークポイント命令が含まれたままの状態になるため、ターゲットがスタンドアロンで動作できなくなる可能性があります。

2.1.3 既存の CCE v2、CCE v3、CCE v3.1、CCS v4.x プロジェクトを使用する

CCS v5.2 では、CCE のバージョン v2、v3、v3.1、CCSv4.x で生成された作業領域(workspace)とプロジェクトの CCS v5.2 形式への変換をサポートしています(File → Import → General → Existing Projects into Workspace → Next)。
インポート対象のプロジェクトが含まれている旧バージョン側の CCE 作業領域を表示させると、Import Wizard により、その作業領域にあるすべてのプロジェクトが一覧表示されます。この状態から、特定のプロジェクトを選択して変換できます。CCEv2 と CCEv3 のプロジェクトの場合は、インポート後にターゲット構成ファイル(*.ccxml)に手作業で変更を加える必要が生じる可能性があります。

使用されていた Code Generation Tools(CGT)の旧バージョンによっては、インポートされたプロジェクトが別バージョンの CGT でビルドされたものであるという警告を IDE 側が返す可能性があります。

アセンブリ・プロジェクト用のサポートは変更されませんが、C コード用のヘッダ・ファイルはわずかに変更されており、IAR Embedded Workbench® IDE (割り込みベクタ定義)との互換性が向上しています。CCE 2.x で使用されていた定義も従来通り用意されていますが、すべてのヘッダ・ファイルでコメントアウトされています。CCE 2.x の C コードに対応するには、#define 文(ステートメント)の前の "//" を削除します。#define 文は、各.h ファイルの末尾に置かれている"Interrupt Vectors" セクションにあります。

2.1.4 スタック管理

予約されているスタック・サイズは、project options ダイアログを介して構成できます(Project → Properties → Build → MSP430 Linker → Basic Options → Set C System Stack Size)。スタック・サイズは、RAM の最後の位置から 50～80 バイト分拡張するように定義されています(つまり、選択されたデバイスの RAM サイズに応じて、スタックは RAM から 50～80 バイト分下方向へ拡張されます)。

サイズの小ささやアプリケーション・エラーが原因で、スタックのオーバーフローが発生する可能性があることに注意してください。スタック・サイズを追跡する方法については、セクション 2.2.2.1 を参照してください。

2.1.5 バイナリ形式ファイルの生成方法 (TI-TXT および INTEL-HEX)

CCSのインストールには、hex430.exeという変換ツールが含まれています。このツールは、TIファクトリ・デバイス・プログラミング用のINTEL-HEX形式のファイルの他に、MSP-GANG430および MSP-PRGS430のプログラマが使用するためのTI-TXT 形式の出力オブジェクトを生成するように構成できます。このツールは、 (<Installation Root>\ccsv5\ccs_base\tools\compiler\msp430\binにある)コマンド・ラインでスタンドアロンで使用することも、CCS内で直接使用することもできます。後者の場合は、ポストビルド(ビルド後)・ステップにて、"Apply Predefined Step" プルダウン・メニューからTI-TXTやINTEL-HEX等の事前定義された形式を選択し、ビルドが終了するごとにファイルを自動的に生成するようにもできます(Project → Properties → Build → Build Steps Tab → Steps → Apply Predefined Step)。生成されたファイルは、<Workspace>\<Project>\Debug\ ディレクトリに格納されます。

2.1.6 プログラム例およびプロジェクト例の概要

MSP430デバイス用のプログラム例は、<Installation Root>\ccsv5\ccs_base\msp430\examplesに置かれています。 アセンブリ(言語)のソースとC言語のソースは、該当するサブディレクトリにあります。

例を使用するには、新規プロジェクトを作成し、次のようにクリックしてソース・ファイル例をプロジェクトに追加します(Project → Add Files...)。また、コード例に対応するプロジェクト例が <Installation Root>\ccsv5\ccs_base\msp430\examples\example projectsに置かれています。プロジェクトをインポートするには、次のようにクリックします。Project → Import Existing CCS/CCE Eclipse Project (詳細については、セクション1.2を参照してください)

2.2 統合デバッガ(Integrated Debugger)を使用する

CCS内のFET固有のメニューの説明は、付録Dに記載されています。

2.2.1 ブレークポイントのタイプ

デバッガのブレークポイント機構では、使用できるオンチップのデバッグ・リソースの数に制限があります(具体的に表すと、ブレークポイント・レジスタの数がN個までとなっています。表2-1参照)。セットされたブレークポイントの数がN以下であれば、アプリケーションは最大限のデバイス速度(あるいは"リアルタイム")で動作します。Nを超える数のブレークポイントがセットされ、Use Software Breakpoints がイネーブルにされた(Project → Properties → Debug → MSP430 Properties → Enable Software Breakpoints)場合は、セットできるソフトウェア・ブレークポイントの数に制限がなくなりますが、実際の動作速度に制限されます。

注: ソフトウェア・ブレークポイントでは、ブレークポイントのアドレスにある命令を、コード実行に割り込むための呼び出しに置き換えます。したがって、ソフトウェア・ブレークポイントをセットする際にはわずかに遅延が発生します。また、ソフトウェア・ブレークポイントの使用では常に、各デバッグ・セッションが正しく終了することが必須となります。正しく終了しない場合は、デバイス上のアプリケーションにソフトウェア・ブレークポイント命令が含まれたままの状態になるため、アプリケーションをスタンドアロンで使用できなくなります。

アドレス(コード)とデータ(値)の両方のブレークポイントがサポートされています。データのブレークポイントと範囲(range)のブレークポイントではそれぞれ、2つのMSP430ハードウェア・ブレークポイントが必要になります。

表 2-1. デバイスのアーキテクチャ、ブレークポイントとエミュレーション機能

デバイス	MSP430 アーキテクチャ	4線 JTAG	2線 JTAG(1)	ブレークポイント(N)	範囲ブレークポイント	クロック制御	ステートシーケンサ	トレースバッファ	LPMx.5 デバッグサポート
CC430F512x	MSP430Xv2	X	X	3	X	X			X
CC430F513x	MSP430Xv2	X	X	3	X	X			
CC430F514x	MSP430Xv2	X	X	3	X	X			X
CC430F612x	MSP430Xv2	X	X	3	X	X			
CC430F613x	MSP430Xv2	X	X	3	X	X			
CC430F614x	MSP430Xv2	X	X	3	X	X			X
MSP430AFE2xx	MSP430	X	X	2		X			
MSP430BT5190	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F11x1	MSP430	X		2					
MSP430F11x2	MSP430	X		2					
MSP430F12x	MSP430	X		2					
MSP430F12x2	MSP430	X		2					
MSP430F13x	MSP430	X		3	X				
MSP430F14x	MSP430	X		3	X				
MSP430F15x	MSP430	X		8	X	X	X	X	
MSP430F16x	MSP430	X		8	X	X	X	X	
MSP430F161x	MSP430	X		8	X	X	X	X	
MSP430F20xx	MSP430	X	X	2		X			
MSP430F21x1	MSP430	X		2		X			
MSP430F21x2	MSP430	X	X	2		X			
MSP430F22x2	MSP430	X	X	2		X			
MSP430F22x4	MSP430	X	X	2		X			
MSP430F23x	MSP430	X		3	X	X			
MSP430F23x0	MSP430	X		2		X			
MSP430F24x	MSP430	X		3	X	X			
MSP430F241x	MSP430X	X		8	X	X	X	X	
MSP430F2410	MSP430	X		3	X	X			
MSP430F261x	MSP430X	X		8	X	X	X	X	
MSP430G2xxx	MSP430	X	X	2		X			
MSP430F41x	MSP430	X		2		X			
MSP430F41x2	MSP430	X	X	2		X			
MSP430F42x	MSP430	X		2		X			
MSP430FE42x	MSP430	X		2		X			
MSP430FE42x2	MSP430	X		2		X			
MSP430FW42x	MSP430	X		2		X			
MSP430F42x0	MSP430	X		2		X			
MSP430FG42x0	MSP430	X		2		X			
MSP430F43x	MSP430	X		8	X	X	X	X	
MSP430FG43x	MSP430	X		2		X			
MSP430F43x1	MSP430	X		2		X			
MSP430F44x	MSP430	X		8	X	X	X	X	
MSP430F44x1	MSP430	X		8	X	X	X	X	
MSP430F461x	MSP430X	X		8	X	X	X	X	
MSP430FG461x	MSP430X	X		8	X	X	X	X	

(1) 2線JTAGデバッグ・インターフェイスは、Spy-Bi-Wire (SBW)インターフェイスとも呼ばれます。このインターフェイスは、USBエミュレータ(eZ430-xxxxおよびMSP-FET430UIF USB JTAG エミュレータ) と、MSP-GANG430生産用(production)プログラミング・ツールでのみサポートされていることに注意してください。MSP-FET430PIFパラレル・ポート JTAGエミュレータでは、2線JTAGモードでの通信をサポートしていません。

表 2-1. デバイスのアーキテクチャ、ブレークポイントとエミュレーション機能(続き)

デバイス	MSP430 アーキテクチャ	4線 JTAG	2線 JTAG(1)	ブレークポイント(N)	範囲ブレークポイント	クロック制御	ステートシーケンサ	トレースバッファ	LPMx.5 デバッグサポート
MSP430F461x1	MSP430X	X		8	X	X	X	X	
MSP430F47x	MSP430	X		2		X			
MSP430FG47x	MSP430	X		2		X			
MSP430F47x3	MSP430	X		2		X			
MSP430F47x4	MSP430	X		2		X			
MSP430F471xx	MSP430X	X		8	X	X	X	X	
MSP430F51x1	MSP430Xv2	X	X	3	X	X			
MSP430F51x2	MSP430Xv2	X	X	3	X	X			
MSP430F52xx	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F530x	MSP430Xv2	X	X	3	X	X			
MSP430F5310	MSP430Xv2	X	X	3	X	X			
MSP430F532x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F533x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F534x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F54xx	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F54xxA	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F550x	MSP430Xv2	X	X	3	X	X			
MSP430F5510	MSP430Xv2	X	X	3	X	X			
MSP430F552x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F563x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430FR57xx	MSP430Xv2	X	X	3	X	X			X
MSP430FR59xx	MSP430Xv2	X	X	3	X	X			X
MSP430F643x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F665x	MSP430Xv2	X	X	8	X	X	X	X	X
MSP430F663x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F67xx	MSP430Xv2	X	X	3	X	X			
MSP430L092	MSP430Xv2	X		2		X			

2.2.2 ブレークポイントの使用

N 個より多いブレークポイントをセットし、ソフトウェア・ブレークポイントをディセーブルにした状態でデバッガが起動された場合は、すべてのブレークポイントをイネーブルにすることはできませんというメッセージが表示されます。CCS では、CCS の Use Software Breakpoints の設定に関係なく、任意の数のブレークポイントをセットできることに注意してください。ソフトウェア・ブレークポイントがディセーブルにされている場合は、デバッガ内で設定可能なブレークポイントの数は最大 N 個になります。

プログラムをリセットするには、次の手順で定義されたアドレスにセットされたブレークポイントが必要になります。Project → Properties → Debug → Generic Debugger Options → Auto Run Options → Run to symbol(main)

Run To Line 動作には、一時的にブレークポイントが必要になります。

printf 等のコンソール入出力(CIO)関数には、ブレークポイントを使用する必要があります。これらの関数がコンパイルイン(compiled in)されてもブレークポイントを使用したくない場合は、次の手順でオプションを変更して、CIO 関数をディセーブルにします。Project → Properties → Debug → Generic Debug Options → Enable CIO function use

2.2.2.1 CCS v5.2のブレークポイント

CCS では、Breakpoint ウィンドウ内の Breakpoints アイコンの隣にあるメニューを開いて選択できる、事前定義されたブレークポイント・タイプを数多くサポートしています(Window → Show View → Breakpoints)。従来のブレークポイントの他に、

CCS ではウォッチポイント(watchpoints)をセットして、アドレス・アクセスではなくデータ・アドレス・アクセス時にプログラムを中断させる(break)ことが可能です。ブレークポイントとウォッチポイントのプロパティをデバッガ内で変更するには、ブレークポイントを右クリックして Properties を選択します。

・プログラム・アドレスの後でのブレーク(中断)
プログラムが特定のアドレスの後でコードの実行を試みた場合に、コードの実行を止めます。

・プログラム・アドレスの前でのブレーク(中断)
プログラムが特定のアドレスの前でコードの実行を試みた場合に、コードの実行を止めます。

・プログラム範囲でのブレーク(中断)
プログラムが特定の範囲内でコードの実行を試みた場合に、コードの実行を止めます。

・DMA 転送時のブレーク(中断)

・範囲内の DMA 転送時のブレーク(中断)
特定のアドレス範囲内で DMA アクセスが発生した場合に中断します。

・スタック・オーバーフロー時のブレーク(中断)
スタック・オーバーフローを発生させたアプリケーションをデバッグすることが可能です。Break on Stack Overflow を設定します(デバッグ・ウィンドウ内で右クリックした後、コンテクスト・メニューの"Break on Stack Overflow"を選択します)。プログラムの実行が、スタック・オーバーフローを起こした命令で停止します。スタックのサイズは、次のように補正できます。
Project → Properties → C/C++ Build → MSP430 Linker → Basic Options

・ブレークポイント
ブレークポイントをセットします。

・ハードウェア・ブレークポイント
ソフトウェア・ブレークポイントがディセーブルにされていない場合に、ハードウェア・ブレークポイントを強制的にセットします。

・データ・アドレス範囲の監視(ウォッチ)
特定範囲にあるアドレスへのデータ・アクセスが発生した場合に、コードの実行を止めます。

・ウォッチポイント
特定アドレスへの特定データ・アクセスが行われた場合に、コードの実行を止めます。

・データを使用したウォッチポイント(Watchpoint with data)
特定アドレスへ特定のデータ値でアクセスが行われた場合に、コードの実行を止めます。

制限 1:ウォッチポイントは、グローバル変数と非レジスタ・ローカル変数に適用可能です。後者の場合は、変数の観察が必要な関数で実行を停止するようにブレークポイント(BP)をセットします(コード BP をその場所にセットします)。次に、ウォッチポイントをセットして、関数内のコード・ブレークポイントを削除し(またはディセーブルにし)、アプリケーションを実行または再起動します。

制限 2: ウォッチポイントは、8 ビット幅と 16 ビット幅の変数に適用可能です。

注: すべてのオプションが、どのMSP430 派生品(derivative)でも使用可能なわけではありません(表2-1を参照)。したがって、ブレークポイント・メニューにある事前定義されたブレークポイント・タイプの数は、選択されたデバイスによって異なります。

CCS を使用した高度なデバッグの詳細については、アプリケーション・レポート Advanced Debugging Using the Enhanced Emulation Module (EEM) With CCS Version 4 (SLAA393)を参照してください。

付録 A 問い合わせの多い質問

この付録では、ハードウェア、プログラム開発、デバッグ・ツールに関して問い合わせの多い事項への回答を紹介します。

A.1 ハードウェア

ハードウェア関連の全FAQのリストは、MSP430 Hardware Tools User's Guide SLAU278に記載されています。

A.2 プログラム開発(アセンブラ、C コンパイラ、リンカ、IDE)

注: **CCSのリリース・ノートもご覧ください。**
予期しない振る舞いがあった場合は、CCS のリリース・ノートを参照して、既知のバグや現行版CCSの制限事項について調べてください。この情報には、次のメニュー項目を経由してアクセスできます。Start → All Programs → Texas Instruments → Code Composer Studio → Release Notes

1. **MSP430 の使用中によく起こる "誤操作" が、ウォッチドッグ機構をディセーブルにしないことです。**ウォッチドッグはデフォルトではイネーブルにされており、アプリケーションによりディセーブルにされなかったり、正しく管理されない場合はデバイスをリセットします。`WDTCL = WDTPW + WDTHOLD` を使用して、ウォッチドッグを明示的にディセーブルにしてください。このステートメントは、main()の前に実行される_system_pre_init() 関数に置くのが最適です。ウォッチドッグ・タイマがディセーブルにされず、CSTARTUP の期間中にウォッチドッグが起動(trigger)してデバイスをリセットした場合は、**ソース画面に何も表示されなくなります**。デバッガ側で CSTARTUP のソース・コードが見つけられなくなるためです。初期化されたグローバル変数が多数使用される場合は、CSTARTUP の実行に相当長い時間がかかることに注意してください。

```
int _system_pre_init(void)
{
/* Insert your low-level initializations here */
WDTCTL = WDTPW + WDTHOLD; // Stop Watchdog timer
/*===================================*/
/* Choose if segment initialization */
/* should be done or not. */
/* Return: 0 to omit initialization */
/*         1 to run initialization */
/*===================================*/
return (1);
}
```

2. **C ライブラリ内では、GIE (グローバル割り込みイネーブル)がハードウェア乗算器の使用前にディセーブルになり、使用後に再度復元されます。**

3. CCS 内で、アセンブリと C のプログラムをミックスすることが可能です。MSP430 Optimizing C/C++ Compiler User's Guide (文書番号 SLAU132)の"Interfacing C/C++ With Assembly Language" 章を参照してください。

4. **.h ファイル内で使用される定数定義(#define) は、事実上予約されており、**C、Z、N、V 等を含んでいます。これらの名前を使用してプログラムの変数を生成することはしないでください。

5. **コンパイラを最適化すると、効力がない未使用の変数や文が削除されることがあり、これはデバッグに影響することがあります。**これを防止するには、これらの変数を volatile(揮発性)と宣言します。例: `volatile int i;`.

A.3 デバッグ

デバッガはCCSの一部ですが、独立したアプリケーションとしても使用できます。このセクションは、デバッガを独立して使用する場合と、CCS IDEから使用する場合の両方に適用可能です。

注: CCSのリリース・ノートもご覧ください。
予期しない振る舞いがあった場合は、CCSのリリース・ノートを参照して、既知のバグや現行版CCSの制限事項について調べてください。この情報には、次のメニュー項目を経由してアクセスできます。Start → All Programs → Texas Instruments → Code Composer Studio → Release Notes

1. **デバッガに、デバイスと通信できないというメッセージが表示されます。**この問題の解決方法として考えられるのは次のようなことです。

- 正しいデバッグ・インターフェイスと対応するポート番号(port number)が選択されているかどうかを、次に示す手順で確認してください。Project → Properties → General → Device → Connection
- ターゲット・ハードウェアで、ジャンパの設定が正しく構成されているかどうかを確認してください。
- COMポートまたはパラレル・ポートの制御権を予約しているか持っているソフトウェア・アプリケーション(プリンタ・ドライバ等)が他に存在しないことを確認してください。そのようなアプリケーションがあると、デバッグ・サーバーがデバイスと通信できなくなります。
- デバイス・マネージャを開き、FETツール用のドライバが正しくインストールされているか、また、COMポートまたはパラレル・ポートがWindows OSに正常に認識されているかどうかを判断します。PCのBIOSをチェックして、パラレル・ポートの設定を確認します(FAQのデバッグ #5を参照)。IBMまたはLenovo ThinkPad®コンピュータのユーザーの場合は、パラレル・ポートがLPT1に存在するとオペレーティング・システムが報告している場合でも、ポート設定LPT2とLPT3を試してみてください。
- コンピュータを再起動します。

MSP430デバイスが(ソケットの「fingers」が完全にデバイスのピンと噛み合うように)しっかりとソケットに差し込まれ、デバイスのピン1(デバイス上面に丸いくぼみで示してあります)がPCB上の"1" のマークに合わせられているかどうかを確認してください。

注意

デバイスで発生する可能性のある損傷
MSP430デバイスは常に、真空吸着ツールのみを使用して取り扱ってください。デバイスのピンは曲がりやすく、指での取り扱うとデバイスが使えなくなるおそれがあるため避けてください。また、正しいESD(静電放電)の予防措置を常に守り、従ってください。

2. **デバッガでは、割り込みモードと低消費電力モードを利用するアプリケーションをデバッグできます。**FAQのデバッグ #17を参照してください。

3. **デバイスの動作中は、デバッガはデバイスのレジスタとメモリにアクセスできません。**デバイスのレジスタとメモリにアクセスするには、ユーザー側でデバイスを止める必要があります。

4. **デバッガにより、デバイスのJTAG安全ヒューズが切断されていることが報告されます。**現在のMSP-FET430PIFとMSP430-FET430UIF JTAGインターフェイス・ツールでは、外部電源から電力が供給してターゲット・ボードを利用する場

合に弱点があります。そのため、MSP430でヒューズのチェックが誤って(accidentally)行われ、JTAGセキュリティヒューズが切断されていないのに切れていると切断されてしまうことがあります。この問題はMSP-FET430PIFとMSPFET430UIFの場合に発生しますが、主にMSP-FET430UIFで見られます。

回避手段:

- デバイスの$\overline{\text{RST}}$/NMIピンをJTAGヘッダ(ピン11)に接続します。MSP-FET430PIFとMSP-FET430UIFインターフェイス・ツールでは$\overline{\text{RST}}$ラインをPull(リセット)ができます。これにより、デバイス内部のヒューズのロジックもリセットされます。
- JTAGヘッダのV_{CC}ツール(ピン2)とV_{CC}ターゲット(ピン4)は、両方同時に接続しないでください。デバッガでは、V_{CC}用として外部電源電圧と等価の値を指定してください。

5. パラレル・ポート指定子(designators) (LPTx)の持つ物理アドレスは次の通りです: LPT1 = 378h, LPT2 = 278h, LPT3 = 3BCh。パラレル・ポートの設定(ECP、互換性、双方向、ノーマル)は重要とはなりませんが、ECPモードで良好に動作すると思われます。デバッガ～デバイス間通信の問題の解決についてさらにヒントが必要な場合は、FAQのデバッグ #1を参照してください。

6. **デバッガが起動された時、またデバイスがプログラムされた時に、デバッガでは $\overline{\text{RST}}$/NMI をアサートしてデバイスをリセットします**。デバイスはまた、デバッガの Reset ボタンを押した時、(Reload を使用して)デバイスが手動で再プログラムされた時、また(Resynchronize JTAG を使用して)JTAG が再同期された時に、リセットされます。$\overline{\text{RST}}$/NMI がアサートされない場合(low の場合)は、デバッガではロジックに駆動される $\overline{\text{RST}}$/NMI をハイ・インピーダンスにセットし、$\overline{\text{RST}}$/NMI は PCB 上の抵抗を介して High になります。
   $\overline{\text{RST}}$/NMI は、デバッガの起動時に電力が印加された後にアサートおよびネゲートされます。$\overline{\text{RST}}$/NMI のアサートとネゲートはその後、デバイスの初期化完了後に再度行われます。

7. **デバッガでは、プログラムが $\overline{\text{RST}}$/NMI ピンの機能を NMI に再設定しているデバイスでもデバッグを行うことが可能です。**

8. **XOUT/TCLK ピンのレベルは、デバッガがデバイスをリセットする時は不定(undefined)になります。**その他すべての時には、ロジックに駆動される XOUT/TCLK はハイ・インピーダンスにセットされます。

9. **デバイスの電流測定を行う場合は、JTAG 制御信号を確実に解放するようにしてください。**そうでないと、JTAG ピンの信号によりデバイスに電力が供給され、正確な測定ができなくなります。FAQ のデバッグ #10 を参照してください。

10. **デバッガによりデバイスが制御されている時は、ステータス・レジスタの低消費電力モード・ビットがセットされているかどうかに関係なく、CPU がオン状態になります**(つまり、低消費電力モードにはなりません)。どのような低消費電力モード状態も、STEP または GO の前に復元されます。そのため、デバッガがデバイスを制御している間は、デバイスに消費される電力の測定は行わないでください。代わりに、Disconnect Target を使用してアプリケーションを実行してください。

11. MEMORY ウィンドウには、実装された領域のメモリ内容が正しく表示されます。ただし、**実装外の領域にあるメモリ内容は MEMORY ウィンドウに正しく表示されません**。デバイスのデータ・シートに指定されたアドレス範囲内でのみ、メモリを使用してください。

12.デバッガでは、デバッグ中にシステム・クロックを利用してデバイスを制御します。したがって、**デバッガがデバイスを制御する場合には、メイン・システム・クロック(MCLK)からクロックを供給されるデバイス・カウンタ等の部品が影響を受けます**。ウォッチドッグ・タイマへの影響を最小限にするために、特別な予防措置が取られます。
CPU のコア・レジスタは保持され、その他すべてのクロック源(SMCLK と ACLK)とペリフェラルでは、エミュレーション中も通常の動作を継続します。つまり、**フラッシュ・エミュレーション・ツールは部分的介入型の(intrusive) ツールであることになります。**

クロック制御をサポートするデバイスでは、デバッグ中にクロックを停止させることで、上記の影響をさらに小さく抑えることが可能です(Project → Properties → CCS Debug Settings → Target → Clock Control と選択)。

13. フラッシュ(・メモリ)をプログラムする場合は、**フラッシュへの書き込み動作の直後の命令(instruction)にブレークポイントをセットしないようにしてください**。この制限は、フラッシュへの書き込み動作後に NOP を使用して、その NOP に続く命令(instruction)にブレークポイントをセットすることで容易に回避できます。

14. フラッシュ・メモリのクリアとプログラミングには、複数の内部マシン・サイクルが必要になります。**フラッシュを操作する命令に対してシングル・ステップを実行する場合は、これらの動作が完了する前に制御がデバッガに返されます。その結果、デバッガのメモリ・ウィンドウが誤った情報でアップデートされることになります**。この振る舞いに対する回避手段は、フラッシュ・アクセス命令に続いて NOP を使用した後、フラッシュ・アクセス命令の結果を確認している NOP を通り越してから Step 実行します。

15. **通常のプログラム実行中に読み出されるとクリアされるビット(つまり割り込みフラグ)は、デバッグ中に読み出されるとクリアされます(つまりメモリ・ダンプ、ペリフェラル・レジスタ)**。
強化されたエミュレーション・ロジック付きの特定の MSP430 デバイス(MSP430F43x や MSP430F44x 等)を使用すれば、ビットが上記のようにふるまうことはなくなります(つまり、デバッガの読み出し動作ではビットがクリアされなくなります)。

16. **F12x および F41x デバイスの RAM で実行されるプログラムのデバッグ用にデバッガを使用することはできません。**この制限は、フラッシュ・メモリ内でプログラムをデバッグすることで回避できます。

17.**アクティブかつイネーブルにされた割り込みとともにシングル・ステップを実行している間は、割り込みサービス・ルーチン(ISR)のみがアクティブである(つまり、ISR 以外のコードが一度も実行されておらず、シングル・ステップ動作が ISR の先頭行で止まる)ように見えることがあります**。ただし、これは正しい振る舞いです。デバイスでは、ISR 以外の(つまりメインラインの)コードを処理する前に、アクティブでイネーブルにされた割り込みを処理するためです。この振る舞いの回避手段は、ISR 内にいる間にスタックにある GIE ビットをディセーブルにして、ISR の実行後に割り込みがディセーブルになるようにすることです。これにより、(割り込みを止めて)ISR 以外のコードをデバッグできるようになります。割り込みは、後から Register ウィンドウでステータス・レジスタの GIE をセットすることで、再度イネーブルにできます。
クロック制御機能(Clock Control)のあるデバイスであれば、シングル・ステップ間でクロックを保留(一時停止)にして、割り込み要求を遅延させることも可能です(Project → Properties → CCS Debug Settings → Target → Clock Control と選択)。

18. データ転送コントローラ(DTC: Data Transfer Controller)を装備したデバイスでは、**データ転送サイクルの完了の方が、低消費電力モード命令のシングル・ステップよりも先に実行されます。**デバイスは、割り込みが処理された後でないと、低消費電力モード命令より先には進みません。割り込みが処理されるまで、シングル・ステップは機能していないように見えます。この状態に対する回避手段は、低消費電力モードに続く命令にブレークポイントをセットしてから、このブレークポイントまで実行する(Run)ことです。

19. **DTC がシングル・ステップまたはブレークポイントに反応して停止した場合でも、データ転送コントローラ(DTC)によるデータの転送が正確に停止しない可能性があります**。DTC がイネーブルにされている状態でシングル・ステップが実行される場合は、1 つ以上のデータ・バイトを転送することが可能です。DTC がイネーブルにされ、かつ 2 ブロック転送モード用に構成されている場合は、シングル・ステップまたはブレークポイントに応答して停止しても、DTC がブロック境界上で正確に停止しない可能性があります。

20. **ブレークポイント**。CCS では、事前定義されたブレークポイントとウォッチポイントのタイプを多数サポートしています。詳細な概要については、セクション 2.2.2 を参照してください。

付録 B IAR 2.x、3.x、4.x から CCS C への移行

TI CCS Cコンパイラ用のソース・コードと、IAR Embedded Workbench コンパイラのソース・コードは、完全互換ではありません。標準的なANSI/ISO Cコードはこれらのツール間で移植可能ですが、実装固有の拡張が異なるため、この拡張部分を移植する必要があります。この付録には、2つのコンパイラの主な相違が記載されています。

B.1 割り込みベクタの定義

今回、IAR ISR 宣言(#pragma vector = を使用)がCCSで完全にサポートされるようになりました。ただし、他のすべてのIAR pragma 擬似命令(directive)でサポートされているわけではありません。

B.2 組み込み関数(Intrinsic Functions)

CCSツールとIARツールでは、MSP430プロセッサ固有の組み込み関数用に、同じ命令を使用します。

B.3 データと関数の配置

B.3.1 絶対的な位置でのデータ配置

IARコンパイラに実装されている、@演算子または #pragma location 擬似命令のどちらかを使用したスキームは、CCSコンパイラではサポートされていません。

```
/* IAR C Code */
__no_init char alpha @ 0x0200; /* Place ‘alpha' at address 0x200 */
#pragma location = 0x0202
const int beta;
```

絶対的なデータ配置が必要な場合は、リンカ・コマンド・ファイルへEntryを定義して、次のようにCコード内で変数を extern と宣言することで実現できます。

```
/* CCS Linker Command File Entry */
alpha = 0x200;
beta = 0x202;
/* CCS C Code */
extern char alpha;
extern int beta;
```

絶対的なRAM位置は、RAMセグメントから除外(excluded)される必要があります。そうでないと、リンカが動的にアドレスを割り当てる際にRAM位置の内容が上書きされる可能性があります。開始アドレスとRAMブロック長は、リンカ・コマンド・ファイル内で修正される必要があります。前の例では、RAMの開始アドレスが0x0200から0x0204へ4バイト分シフトされる必要があります。これにより、ブロック長は0x0080から0x007Cに(MSP430デバイスの場合、RAMは128バイトです)短縮されます。

```
/* CCS Linker Command File Entry */
/************************************************************************/
/* SPECIFY THE SYSTEM MEMORY MAP */
/************************************************************************/
MEMORY /* assuming a device with 128 bytes of RAM */
{
...
RAM :origin = 0x0204, length = 0x007C /* was: origin = 0x200, length = 0x0080 */
...
}
```

絶対位置にデータを配置した例として、リンカ・コマンド・ファイル(lnk_msp430xxxx.cmd)内のペリフェラル・レジスタ・マップの定義および、CCSとともに供給されるデバイス固有ヘッダ・ファイル(msp430xxxx.h)があります。

注: プロジェクトを生成する時に、CCSでは選択されたMSP430の派生品(derivative)に対応するリンカ・コマンド・ファイルを、インクルード・ディレクトリ(<Installation Root>\ccsv5\ccs_base\tools\compiler\MSP430\include)からプロジェクト・ディレクトリにコピーします。したがって、リンカ・コマンド・ファイルの変更はすべてプロジェクト・ディレクトリ内で行うようにしてください。これにより、同じデバイスを使用する様々なプロジェクトで、各プロジェクト固有のリンカ・コマンド・ファイルが使用できるようになります。

B.3.2 名前付き(Named)セグメントへのデータ配置

IARでは、次に示すように@演算子または#pragma 擬似命令を使用して、変数を名前付きセグメントに配置することが可能です。

```
/* IAR C Code */
__no_init int alpha @ "MYSEGMENT"; /* Place ‘alpha' into ‘MYSEGMENT' */
#pragma location="MYSEGMENT"       /* Place ‘beta' into ‘MYSEGMENT' */
const int beta;
```

CCSコンパイラでは、 #pragma DATA_SECTION() 擬似命令を使用する必要があります。

```
/* CCS C Code */
#pragma LOCATION(alpha, "MYSEGMENT")
int alpha;
#pragma LOCATION(beta, "MYSEGMENT")
int beta;
```

IARとCCSの間でメモリ・セグメント名を変換する方法については、セクションB.5.3を参照してください。

B.3.3 名前付きセグメントへの関数配置

IARコンパイラを使用すると、次に示すように@演算子または#pragma location 擬似命令を使用して、関数を名前付きセグメントに配置できます。

```
/* IAR C Code */
void g(void) @ "MYSEGMENT"
{
}
#pragma location="MYSEGMENT"
void h(void)
{
}
```

CCSコンパイラでは、#pragma CODE_SECTION() 擬似命令付きの次のようなスキームを使用する必要があります。

```
/* CCS C Code */
#pragma CODE_SECTION(g, "MYSEGMENT")
void g(void)
{
}
```

IARとCCSの間でメモリ・セグメント名を変換する方法については、セクションB.5.3を参照してください。

B.4 C の呼び出し規約

CCSとIARのCコンパイラでは、パラメータを関数に渡すために使用する呼び出し規約(calling conventions)が異なります。ミックスされたCとアセンブリのプロジェクトをTIのCCSコード生成ツールに移植する場合は、これらの変更を反映させるためにアセンブリ関数を修正する必要があります。呼び出し規約の詳細については、TI MSP430 Optimizing C/C++ Compiler User's Guide (SLAU132)とIAR MSP430 C/C++ Compiler Reference Guideを参照してください。

次に示す例は、32ビットのワード Data' をビッグエンディアンのバイト・オーダーで指定のメモリ位置に書き込む関数です。別個のCPUレジスタを使用してパラメータ 'Data' が渡されているのが分かります。

IAR 版:

```
;-------------------------------------------------------------------------------
; void WriteDWBE(unsigned char *Add, unsigned long Data)
;
; Writes a DWORD to the given memory location in big-endian format. The
; memory address MUST be word-aligned.
;
; IN: R12 Address (Add)
;     R14 Lower Word (Data)
;     R15 Upper Word (Data)
;-------------------------------------------------------------------------------
WriteDWBE
swpb R14         ; Swap bytes in lower word
swpb R15         ; Swap bytes in upper word
mov.w R15,0(R12) ; Write 1st word to memory
mov.w R14,2(R12) ; Write 2nd word to memory
ret
```

CCS版

```
;-------------------------------------------------------------------------------
; void WriteDWBE(unsigned char *Add, unsigned long Data)
;
; Writes a DWORD to the given memory location in big-endian format. The
; memory address MUST be word-aligned.
;
; IN: R12 Address (Add)
;     R13 Lower Word (Data)
;     R14 Upper Word (Data)
;-------------------------------------------------------------------------------
WriteDWBE
swpb R13         ; Swap bytes in lower word
swpb R14         ; Swap bytes in upper word
mov.w R14,0(R12) ; Write 1st word to memory
mov.w R13,2(R12) ; Write 2nd word to memory
ret
```

B.5 その他の相違点

B.5.1 静的変数とグローバル変数の初期化

ANSI/ISOのC言語規格では、明示的に初期化していない静的変数とグローバル変数(外部変数(extern))は、(プログラムの実行が開始される前に)あらかじめ0に初期化しておく必要があると規定しています。このタスクは通常、プログラムがロードされるときは実行されます。これはIARコンパイラに実装されています。

```
/* IAR, global variable, initialized to 0 upon program start */
int Counter;
```

ただし、TI CCSコンパイラではこれらの変数をあらかじめ初期化しないため、この要件が満たされるどうかはアプリケーション次第になります。

```
/* CCS, global variable, manually zero-initialized */
int Counter = 0;
```

B.5.2 カスタム・ブート・ルーチン(Custom Boot Routine)

IARコンパイラを使用する場合はCのスタートアップ関数(startup function)をカスタマイズできるため、アプリケーションでは早期初期化(ペリフェラルの構成等)を実行したり、データ・セグメントの初期化を省略したりすることが可能になります。これは次のように、カスタマイズされた__low_level_init() 関数を提供することで実現できます。

```
/* IAR C Code */
int __low_level_init(void)
{ =
/* Insert your low-level initializations here */
/*================================== */
/* Choose if segment initialization  */
/* should be done or not.            */
/* Return: 0 to omit initialization  */
/*         1 to run initialization   */
/*================================== */
return (1);
}
```

戻り値では、データ・セグメントをCのスタートアップ・コードで初期化するかどうかを制御します。CCSのCコンパイラでは、カスタム・ブート・ルーチン名が_system_pre_init()であり、IARコンパイラでと同様に使用されます。

```
/* CCS C Code */
int _system_pre_init(void)
{
/* Insert your low-level initializations here */
/*================================== */
/* Choose if segment initialization  */
/* should be done or not.            */
/* Return: 0 to omit initialization  */
/*         1 to run initialization   */
/*================================== */
return (1);
}
```

両方のコンパイラでセグメントの初期化を省略すると、明示的と非明示的両方の初期化が省略されることに注意してください。ユーザー側では、実行の際に、重要な変数が使用前に初期化されるようにする必要があります。

B.5.3 事前定義されたメモリ・セグメント名

CCSツールでもIARツールでも、データと関数の配置用のメモリ・セグメント名はデバイス固有のリンカ・コマンド・ファイルにより制御されます。ただし、使用されるセグメント名は異なります。詳細については、リンカ・コマンド・ファイルを参照してください。次の表には、最も一般的に使用されるセグメント名の変換方法が記載されています。

説明	CCS のセグメント名	IAR のセグメント名
RAM	.bss	DATA16_N DATA16_I DATA16_Z
スタック(RAM)	.stack	CSTACK
メイン・メモリ(フラッシュまたはROM)	.text	CODE
情報メモリ(フラッシュまたはROM)	.infoA .infoB	INFOA INFOB INFO
割り込みベクタ(フラッシュまたはROM)	.int00 .int01 … .int14	INTVEC
リセット・ベクタ(フラッシュまたはROM)	.reset	RESET

B.5.4 事前定義されたマクロ名

IARコンパイラとCCSコンパイラでは両方とも、ANSI/ISO標準以外の事前定義されたマクロ名を数個持っています。このマクロ名を使用すると、様々なコンパイラ・プラットフォームでコンパイルし、使用できるコードの生成に役立ちます。#ifdef 擬似命令を使用して、マクロ名が定義されているかどうかを確認してください。

説明	CCS マクロ名	IAR マクロ名
MSP430 がターゲットであり、特殊なコンパイラ・プラットフォームが使用されているか	_MSP430_	_ICC430_
特殊なコンパイラ・プラットフォームが使用されているか	__TI_COMPILER_VERSION__	__IAR_SYSTEMS_ICC__
C のヘッダ・ファイルがアセンブリのソース・コード内部からインクルードされているか	_ASM_HEADER_	_ IAR_SYSTEMS_ASM_

付録 C IAR 2.x、3.x、4.x から CCS アセンブラへの移行

TI CCSアセンブラ用のソースと、IARアセンブラ用のソース・コードは、100%互換可能ではありません。命令ニーモニックは同一ですが、アセンブラの擬似命令が若干異なります。この付録には、CCSアセンブラの擬似命令とIAR 2.x、3.xアセンブラの擬似命令の相違点が記載されています。

C.1 C/C++ のヘッダ・ファイルをアセンブリのソースと共有する

IAR A430アセンブラでは一定のC/C++ プリプロセッサ擬似命令をサポートしているため、MSP430デバイス固有のヘッダ・ファイル(msp430xxxx.h)等のC/C++ヘッダ・ファイルをアセンブリ・コードに直接インクルードすることが可能になっています。

```
#include "msp430x14x.h" // Include device header file
```

CCS Asm430アセンブラを使用する場合は、上記とは異なる、.cdecls 擬似命令を使用したスキームを使用する必要があります。この擬似命令を使用すると、ミクスド・アセンブリ環境とC/C++環境両方のプログラマが、C/C++コードとアセンブリ・コード間で宣言とプロトタイプの入ったC/C++ヘッダを共有できるようになります。

```
.cdecls C,LIST,"msp430x14x.h" ; Include device header file
```

.cdecls 擬似命令の詳細は、MSP430 Assembly Language Tools User's Guide (文書番号 SLAU131)に記載されています。

C.2 セグメントの制御

CCS Asm430アセンブラでは、ORG、ASEG、RSEG、COMMON等のIAR A430 セグメント制御擬似命令をサポートしていません。

説明	Asm430 擬似命令 (CCS)
.bss 未初期化セクションに空間を予約する	.bss
名前付き未初期化セクションに空間を予約する	.usect
デフォルトのプログラム・セクション(初期化済み)にプログラムを割り当てる	.text
名前付きの初期化済みセクションにデータを割り当てる	.sect

CCSアセンブラを使用してコード・セクションとデータ・セクションを特定のアドレスにそれぞれ割り当てるには、リンカ・コマンド・ファイル内に定義されたメモリ・セクションを生成して使用する必要があります。次の例では、違いを強調するために、IARとCCS両方のアセンブリでの割り込みベクタ割り当てを示しています。

```
;-------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x11x1 and 12x(2) - IAR Assembler
;-------------------------------------------------------------------------
ORG 0FFFEh ; MSP430 RESET Vector
DW RESET   ;
ORG 0FFF2h ; Timer_A0 Vector
DW TA0_ISR ;
;------------------------------------------------------------------------- ;
Interrupt Vectors Used MSP430x11x1 and 12x(2) - CCS Assembler
;-------------------------------------------------------------------------
.sect ".reset" ; MSP430 RESET Vector
.short RESET   ;
.sect ".int09" ; Timer_A0 Vector
.short TA0_ISR ;
```

どちらの例でも、標準的なデバイスのサポート・ファイル(ヘッダ・ファイル、リンカ・コマンド・ファイル)を使用していることが前提となっています。リンカ・コマンド・ファイルがIARとCCSでは異なるため、再利用ができないことに注意してください。メモリ・セグメント名をIARとCCS間で変換する方法については、セクションB.5.3を参照してください。

C.3 A430 アセンブラ擬似命令を Asm430 擬似命令に変換する

C.3.1 はじめに

次のセクションでは全体として、IAR A430アセンブラ(A430)のアセンブラ擬似命令を、Texas Instruments CCS Asm430アセンブラ(Asm430)の擬似命令に変換する方法を説明します。このC.3セクションは、変換の目安を示すことのみを目的としたものです。各擬似命令の詳細については、Texas Instrumentsから出ているMSP430 Assembly Language Tools User's Guide (SLAU131)か、IARから出ているMSP430 IAR Assembler Reference Guideを参照してください。

注: アセンブラの*擬似命令*のみ変換が必要です
変換が必要なのはアセンブラ擬似命令(directive)のみであり、アセンブラ命令(instruction)には必要ありません。どちらのアセンブラでも、同じ命令ニーモニック、オペランド、演算子(operators)の他、セクション・プログラム・カウンタ($)やコメントデリミタ(;)等の特殊記号(special symbol)を使用します。

A430アセンブラでは、デフォルトでは大文字と小文字を区別しません。以降のセクションでは、A430擬似命令を大文字、Asm430擬似命令を小文字で表記して区別します。

C.3.2 文字列

それぞれのアセンブラでは、使用する擬似命令が異なる他に、文字列に使用する構文(syntax)も異なります。A430 では文字列にCの構文を使用します。引用文(quote)は、バックスラッシュ文字をエスケープ文字として引用符と併用することで表現し(\")、バックスラッシュ自体は 2 つの連続したバックスラッシュ(\\)で表されます。Asm430 の構文では、引用文は 2 つの連続した引用符 ("")で表現されます。次に表に例を記載します。

文字列	Asm430 構文 (CCS)	A430 構文 (IAR)
PLAN "C"	"PLAN ""C"""	"PLAN \"C\""
\dos\command.com	"\dos\command.com"	"\\dos\\command.com"
連続した文字列(例: Error 41)	-	"Error " "41"

C.3.3 セクション制御擬似命令

Asm430には、3つの事前定義されたセクションがあります。この3つの中に、プログラムの各パートがアセンブルされます。未初期化データは.bss セクション、初期化済みデータは.dataセクション、実行可能コードは.textセクションにアセンブルされます。

A430でもセクションやセグメントを使用しますが、事前定義されたセグメント名はありません。多くの場合、Cコンパイラで使用される名前(未初期化データではDATA16_Z、定数(初期化)データではCONST、実行可能コードではCODE)をそのまま使用する方が便利です。次に示す表では、これらの名前を使用しています。

一対のセグメントを使用して、初期化された修正可能な(modifiable)データPROM-ableを生成するのに使用できます。ROMセグメントには初期化子(initializer)が入り、スタートアップ・ルーチンによりRAMセグメントにコピーされます。この場合は、両セグメントが正確に同じサイズとレイアウトである必要があります。

説明	Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)
.bss(未初期化データ) セクションにサイズ・バイトを予約する	.bss(1)	(2)
.data (初期化済みデータ) セクションにアセンブルする	.data	RSEG const
名前付きの(初期化済み)セクションにアセンブルする	.sect	RSEG
.text (実行可能コード)セクションにアセンブルする	.text	RSEG code
名前付き(未初期化)セクションに空間を予約する	.usect(1)	(2)
バイト境界上のアライメント	.align 1	(3)
ワード境界上のアライメント	.align 2	EVEN

(1) .bss と.usectでは、元のセクションと未初期化セクションを交互に切り替える必要はありません。例えば、次のようになります。

```
; IAR Assembler Example
            RSEG   DATA16_N       ; Switch to     DATA segment
            EVEN                  ; Ensure proper     alignment
ADCResult:  DS     2              ; Allocate 1     word in RAM
Flags:      DS     1              ; Allocate 1     byte in RAM
            RSEG   CODE           ; Switch back     to CODE segment
; CCS Assembler Example #1
ADCResult   .usect  ".bss",2,2    ; Allocate 1     word in RAM
Flags       .usect  ".bss",1      ; Allocate 1     byte in RAM
; CCS Assembler Example #2
            .bss    ADCResult,2,2 ; Allocate 1     word in RAM
            .bss    Flags,1       ; Allocate 1     byte in RAM
```

(2) まずそのセグメントに切り替えた後、適切なメモリ・ブロックを定義し、元のセグメントに戻ることによって、未初期化セグメント内に予約されます。例えば、次のようになります。

```
            RSEG    DATA16_Z
LABEL:      DS      16            ; Reserve 16     byte
            RSEG    CODE
```

(3) ビットフィールド定数(.field)の初期化はサポートされていないため、セクション・カウンタは常にバイト境界整列(byte-aligned)となります。

C.3.4 定数初期化擬似命令

説明	Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)
1 つ以上の連続したバイトまたはテキスト文字列を初期化する	.byte または .string	DB
32 ビット IEEE 浮動小数点定数を初期化する	.double または .float	DF
変数長フィールドを初期化する	.field	(1)

現在のセクション内にサイズ・バイトを予約する	.space	DS
ひとつ以上のテキスト文字列を初期化する	Initialize one or more text strings	DB
ひとつ以上の 16 ビット整数を初期化する	.word	DW
ひとつ以上の 32 ビット整数を初期化する	.long	DL

(1) ビットフィールド定数(.field)の初期化はサポートされていません。定数は、DWを使用して完全なワードに結合する必要があります。

```
; Asm430 code                ; A430     code
.field 5,3    \
.field 12,4   | ->           DW      (30<<(4+3))|(12<<3)|5   ; equals 3941
.field 30,8   /
```

C.3.5 一覧表示制御擬似命令

説明	Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)
false 条件コード・ブロックの一覧表示を許可する	.fclist	LSTCNDInhibit
false 条件コード・ブロックの一覧表示を抑制する	.fcnolist	LSTCND+
ソース一覧表示のページ長を設定する	.length	PAGSIZ
ソース一覧表示のページ幅を設定する	.width	COL
ソース一覧表示を再開する	.list	LSTOUT+
ソース一覧表示を停止する	.nolist	LSTOUTAllow
マクロ一覧表示とループ・ブロックを許可する	.mlist LSTEXP+ (macro)	LSTREP+ (loop blocks)
マクロ一覧表示とループ・ブロックを抑制する	.mnolist LSTEXP− (macro)	LSTREP− (loop blocks)
出力一覧表示オプションを選択する	.option	(1)
ソース一覧表示の 1 ページ分を排出する	.page	PAGE
拡張置換シンボルの一覧表示を許可する	.sslist	(2)
拡張置換シンボルの一覧表示を抑制する	.ssnolist	(2)
一覧表示のページ見出しにタイトルを出力する	.title	(3)

(1) .optionに直接対応するA430擬似命令はありません。個別の一覧表示制御擬似命令(上記)またはコマンドライン・オプション-c (サブオプション付き)を使用して、.option 擬似命令と置き換える必要があります。

(2) .sslist と .ssnolistに直接対応する擬似命令はありません。

(3) 一覧表示のページ見出しのタイトルは、ソース・ファイル名になります。

C.3.6 ファイル参照擬似命令

説明	Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)
ソース文を他のファイルからインクルードする	.copy または .include	#include または $
現在のモジュールで定義され、他のモジュールで使用されているひとつ以上のシンボルを識別する	.def	PUBLIC または EXPORT
ひとつ以上のグローバル(外部)・シンボルを識別する	.global	(1)
マクロ・ライブラリを定義する	.mlib	(2)
現在のモジュールで使用されているが、他のモジュールで定義されているひとつ以上のシンボルを識別する	.ref	EXTERN または IMPORT

(1) 擬似命令 .globalは、シンボルが現在のモジュールで定義されている場合は.def として、そうでない場合は.refとして機能します。PUBLICまたはEXTERNは、A430アセンブラを使用して適用できるため、.global 擬似命令を置換するために使用する必要があります。

(2) マクロ・ライブラリの概念はサポートされていません。この機能を実現するには、マクロ定義のあるインクルード・ファイル(Include files)を使用する必要があります。

モジュールをAsm430アセンブラとともに使用して、個別にリンク可能なルーチンを生成できます。ひとつのファイルに複数のモジュールまたはルーチンを入れることが可能です。DEFINE、#define (IARプリプロセッサの擬似命令)、MACROのいずれかにより生成されたシンボル以外のシンボルはすべて、モジュールの終わりで"未定義(undefined)"となります。さらに、ライブラリ・モジュールは条件付きでリンクされます。つまりライブラリ・モジュールは、モジュール内のパブリック・シンボルが外部的に参照される場合のみに、リンクされた実行可能(ファイル)にインクルードされます。次に示す擬似命令は、A430アセンブラのモジュールの開始と終了を示すために使用されます。

補助的な A430 擬似命令 (IAR)	A430 擬似命令 (IAR)
プログラム・モジュールを開始する	NAME または PROGRAM
ライブラリ・モジュールを開始する	MODULE または LIBRARY
現在のプログラムまたはライブラリ・モジュールを終了する	ENDMOD

C.3.7 条件付きアセンブリ擬似命令

説明	Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)
オプションの反復可能ブロック・アセンブリ	.break	(1)
条件付きアセンブリを開始する	.if	IF
オプションの条件付きアセンブリ	.else	ELSE
オプションの条件付きアセンブリ	.elseif	ELSEIF
条件付きアセンブリを終了する	.endif	ENDIF
反復可能ブロック・アセンブリを終了する	.endloop	ENDR
反復可能ブロック・アセンブリを開始する	.loop	REPT

(1) .breakに直接対応する擬似命令はありません。ただし、次のように反復可能ブロック・アセンブリがマクロ内で使用される場合は、EXITM擬似命令を他の条件付き(アセンブリ)とともに使用することは可能です。

```
SEQ     MACRO   FROM,TO      ; Initialize a sequence     of byte constants
        LOCAL   X
X       SET     FROM
        REPT    TO-FROM+1    ; Repeat from FROM to      TO
        IF      X>255        ; Break if X exceeds      255
        EXITM
        ENDIF
        DB      X            ; Initialize bytes to      FROM...TO
X       SET     X+1          ; Increment      counter
        ENDR
        ENDM
```

C.3.8 シンボル制御擬似命令

2つのアセンブリでは、アセンブリ時シンボルの範囲が異なっています。Asm430では、(シンボル)定義をファイルに対してグローバルにしたり、モジュールまたはマクロに対してローカルにすることが可能です。ローカル・シンボルは、.newblock擬似命令を使用して未定義にできます。A430のシンボルは、マクロに対してローカル(LOCAL)、モジュールに対してローカル(EQU)、ファイルに対してグローバル(DEFINE)のどれかになります。
また、プリプロセッサ擬似命令 #define を使用して、ローカル・シンボルを定義することもできます。

説明	Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)
置換シンボルに文字列を割り当てる	.asg	SET または VAR または ASSIGN
ローカル・シンボルを未定義にする	.newblock	(1)
値とシンボルを等価にする	.equ または .set	EQU または =
数値置換シンボルの算術演算を行う	.eval	SET または VAR または ASSIGN
構造体の定義を終了する	.endstruct	(2)
構造体の定義を開始する	.struct	(2)
構造体の属性をラベルに割り当てる	.tag	(2)

(1) .newblockに直接対応するA430擬似命令はありません。ただし、#define擬似命令を使用して定義されたシンボルをリセットするために#undefを使用できます。また、マクロまたはモジュールを使用して.newblock機能を実現することもできます。マクロまたはモジュールの終わりでは、ローカル・シンボルは暗黙で(implicitly)未定義であるためです。

(2) 構造体の型(structure types)の定義はサポートされていません。同様の機能は、次に示すように、マクロを使用して総データおよびベース・アドレス+シンボリック・オフセットを割り当てることで実現されます。

```
            DS 4
            ENDM
LO          DEFINE  0
HI          DEFINE  2
            RSEG    DATA16_Z
X           MYSTRUCT
            RSEG    CODE
            MOV     X+LO,R4
            ...
```

C.3.9 マクロ擬似命令

説明	Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)
マクロを定義する	.macro	MACRO
途中でマクロから抜ける	.mexit	EXITM
マクロ定義を終了する	.endm	ENDM

C.3.10 その他各種の(Miscellaneous)擬似命令

説明	Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)
ユーザー定義エラー・メッセージを出力デバイスに送信する	.emsg	#error
ユーザー定義メッセージを出力デバイスに送信する	.mmsg	#message(1)
ユーザー定義警告メッセージを出力デバイスに送信する	.wmsg	(2)
ロード・アドレス・ラベルを定義する	.label (3)	
絶対リスタにより生成された擬似命令	.setsect	ASEG(4)
絶対リスタにより生成された擬似命令	.setsym	EQU または =(4)
プログラムを終了する	.end	END

(1) #message 擬似命令の構文は次の通りです。#message ″<string>″ これにより、アセンブルおよびコンパイル期間中に '#message <string>' がプロジェクト・ビルド・ウィンドウに出力されるようになります。

(2) 警告メッセージをユーザー定義にすることはできません。#message は使用できますが、警告カウンタは増分されません。

(3) ロードタイム・アドレスの概念はサポートされません。ランタイム・アドレスとロードタイム・アドレスは同じであることが前提となっています。同じ効果を実現するために、EQU 擬似命令によりラベルを絶対(ランタイム)アドレスに付与できます。

```
; Asm430 code           ; A430 code
.label  load_start      load_start:
Run_start:                  <code>
   <code>               load_end:
Run_end:                run_start: EQU 240H
.label  load_end        run_end:   EQU run_start+load_end-load_start
```

(4) 絶対リスタでは生成されませんが、ASEGでは絶対セグメントを定義し、またEQUは絶対シンボルを定義するのに使用できます。

```
MYFLAG    EQU   23EH        ; MYFLAG is located at     23E
          ASEG  240H        ; Absolute segment at     240
MAIN:     MOV   #23CH, SP   ; MAIN is located at     240
 ...
```

C.3.11 Asm430 擬似命令のアルファベット順一覧表示と相互参照

Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)	Asm430 擬似命令 (CCS)	A430 擬似命令 (IAR)
.align	ALIGN	.loop	REPT
.asg	SET または VAR または ASSIGN	.macro	MACRO
.break	条件付きアセンブリ擬似命令を参照	.mexit	EXITM
.bss	シンボル制御擬似命令を参照	.mlib	ファイル参照擬似命令を参照
.byte または .string	DB	.mlist	LSTEXP+ (macro)
.cdecls	C プリプロセッサ宣言が本質的に(inherently)サポートされています。		LSTREP+ (loop blocks)
.copy または .include	#include または $	.mmsg	#message (XXXXXX)
.data	RSEG	.mnolist	LSTEXP− (macro)
.def	PUBLIC または EXPORT		LSTREP− (loop blocks)
.double	サポートされていません	.newblock	シンボル制御擬似命令を参照
.else	ELSE	.nolist	LSTOUT−
.elseif	ELSEIF	.option	一覧表示制御擬似命令を参照
.emsg	#error	.page	PAGE
.end	END	.ref	EXTERN または IMPORT
.endif	ENDIF	.sect	RSEG
.endloop	ENDR	.setsect	その他各種の擬似命令を参照
.endm	ENDM	.setsym	その他各種の擬似命令を参照
.endstruct	シンボル制御擬似命令を参照	.space	DS
.equ または .set	EQU または =	.sslist	サポートされていません
.eval	SET または VAR または ASSIGN	.ssnolist	サポートされていません
.even	EVEN	.string	DB
.fclist	LSTCND−	.struct	シンボル制御擬似命令を参照
.fcnolist	LSTCND+	.tag	シンボル制御擬似命令を参照
.field	定数初期化擬似命令を参照	.text	RSEG
.float	定数初期化擬似命令を参照	.title	一覧表示制御擬似命令を参照
.global	ファイル参照擬似命令を参照	.usect	シンボル制御擬似命令を参照
.if	IF	.width	COL
.label	その他各種の擬似命令を参照	.wmsg	その他各種の擬似命令を参照
.length	PAGSIZ	.word	DW
.list	LSTOUT+		

C.3.12 サポートされていない A430 擬似命令(IAR)

次に挙げるIARアセンブラ擬似命令は、CCS Asm430アセンブラではサポートされていません。

条件付きアセンブリ擬似命令	マクロ擬似命令	
REPTC(1)	LOCAL(2)	
REPTI		
ファイル参照擬似命令	**その他各種の擬似命令**	**シンボル制御擬似命令**
NAME または PROGRAM	RADIX	DEFINE
MODULE または LIBRARY	CASEON	SFRB
ENDMOD	CASEOFF	SFRW
一覧表示制御擬似命令	**C 形式のプリプロセッサ擬似命令(3)**	**シンボル制御擬似命令**
LSTMAC (+/−)	#define	ASEG
LSTCOD (+/−)	#undef	RSEG
LSTPAG (+/−)	#if, #else, #elif	COMMON
LSTXREF (+/−)	#ifdef, #ifndef	STACK
	#endif	ORG
	#include	
	#error	

(1) IAR REPTCおよびREPTI擬似命令は、CCSでは直接サポートされていません。ただし、CCSの .macro 擬似命令を次のように使用して、同等の機能を実現できます。

```
; IAR Assembler Example
            REPTI   zero,"R4","R5","R6"
            MOV     #0,zero
            ENDR
; CCS Assembler Example
zero_regs .macro list
            .var item
            .loop
            .break ($ismember(item, list) =     0)
            MOV #0,item
            .endloop
            .endm
```

″zero_regs R4,R5,R6″を呼び出すことで生成されるコードは次の通りです。

```
            MOV #0,R4
            MOV #0,R5
            MOV #0,R6
```

(2) CCSでは、$n（n=0…9となります）または NAME?を使用してローカル・ラベルが定義されます。例えば、$4、$7、または Test?のようになります。

(3) C形式のプリプロセッサ擬似命令の使用は、.cdeclsの使用を介して間接的にサポートされています。.cdecls擬似命令の詳細は、MSP430 Assembly Language Tools User's Guide (文書番号 SLAU131)に記載されています。

付録 D FET 固有のメニュー

この付録では、FETに固有のCCSメニューについて説明します。

D.1 メニュー

D.1.1 Debug View: Run → Free Run

デバッガでは、デバイスの JTAG 信号を使用してデバイスのデバッグを行います。MSP430 デバイスによっては、これらの JTAG 信号がデバイスのポート・ピンと共有されています。通常、デバイスのデバッグが可能になるように、デバッガではピンを JTAG モードに保持しています。この期間中は、共有ピンのポート機能(functionality)は使用できません。

ただし、(Debug View の上部にある Run アイコンの隣にあるプルダウン・メニューを開いて) Free Run を選択すると JTAG ドライバが 3 ステートに設定され、GO がアクティベートされた場合にデバイスが JTAG 制御から解放されます(TEST ピンが GND に設定されます)。任意のアクティブなオンチップ・ブレークポイントがそのまま保持され(retained)、共有 JTAG ポート・ピンがポート機能に戻ります。

この時、デバッガではデバイスを使用できないため、アクティブなブレークポイント(存在する場合)に達しているかどうかを判定できません。デバイスを停止するには、デバッガに手動でコマンドを送信する必要があります。この時に、デバイスのステート(つまり、ブレークポイントに達したかどうか)が判定されます。
FAQ のデバッグ #9 を参照してください。

D.1.2 Run → Connect Target

ティック(tick)されると、デバイスの制御を取り戻します。

D.1.3 Run → Advanced → Make Device Secure

ターゲット・デバイス上の JTAG ヒューズを切断します。ヒューズが切断された後は、JTAG を介したデバイスとの通信がそれ以上できなくなります。

D.1.4 Project → Properties → Debug → MSP430 Properties → Clock Control

デバイスの制御権をデバッガが持っている間、(STOP またはブレークポイント以降) 指定されたシステム・クロックをディセーブルにします。GO または単一ステップ(STEP または STEP INTO)に続いて、すべてのシステム・クロックがイネーブルになります。デバッガがアクティブでない場合のみに変更可能です。FAQ のデバッグ #12 を参照してください。

D.1.5 Window → Show View → Breakpoints

MSP430 の Breakpoints View ウィンドウをオープンします。このウィンドウを使用して、基本的なブレークポイントと高度なブレークポイントを設定できます。条件付きトリガやレジスタ・トリガ等の詳細な設定も、プロパティにアクセスする(対応するブレークポイント上で右クリック)ことにより、各ブレークポイントについて個別に選択可能です。Break on Stack Overflow(スタック・オーバーフローで中断)等の事前定義されたブレークポイントは、Breakpoint プルダウン・メニューを開いて選択できます。このプルダウン・メニューは、ウィンドウ上部の Breakpoint アイコンの隣にあります。複数のブレークポイント同士を Breakpoint View ウィンドウ内でドラッグ＆ドロップして結合する(combined)こともできます。結合してひとつになったブレークポイントは、すべてのブレークポイント条件が満たされた場合にトリガされます。

D.1.6 Window → Show View → Other... Debug → Trace Control

Trace View では、ステート格納モジュールの使用をイネーブルにします。ステート格納モジュールは、Enhanced Emulation Module (EEM)の完全版を含むデバイスにのみ存在します(表 2-1 参照)。ブレークポイントが定義された後、設定された通りの

トレース情報が State Storage View に表示されます。ウィンドウ右上隅の Configuration Properties アイコンをクリックすると、各種能トレース・モードを選択できます。EEM の詳細は、アプリケーション・レポート Advanced Debugging Using the Enhanced Emulation Module (EEM) With CCS Version 4 (SLAA393)に記載されています。

D.1.7 Project → Properties → Debug → MSP430 Properties → Target Voltage

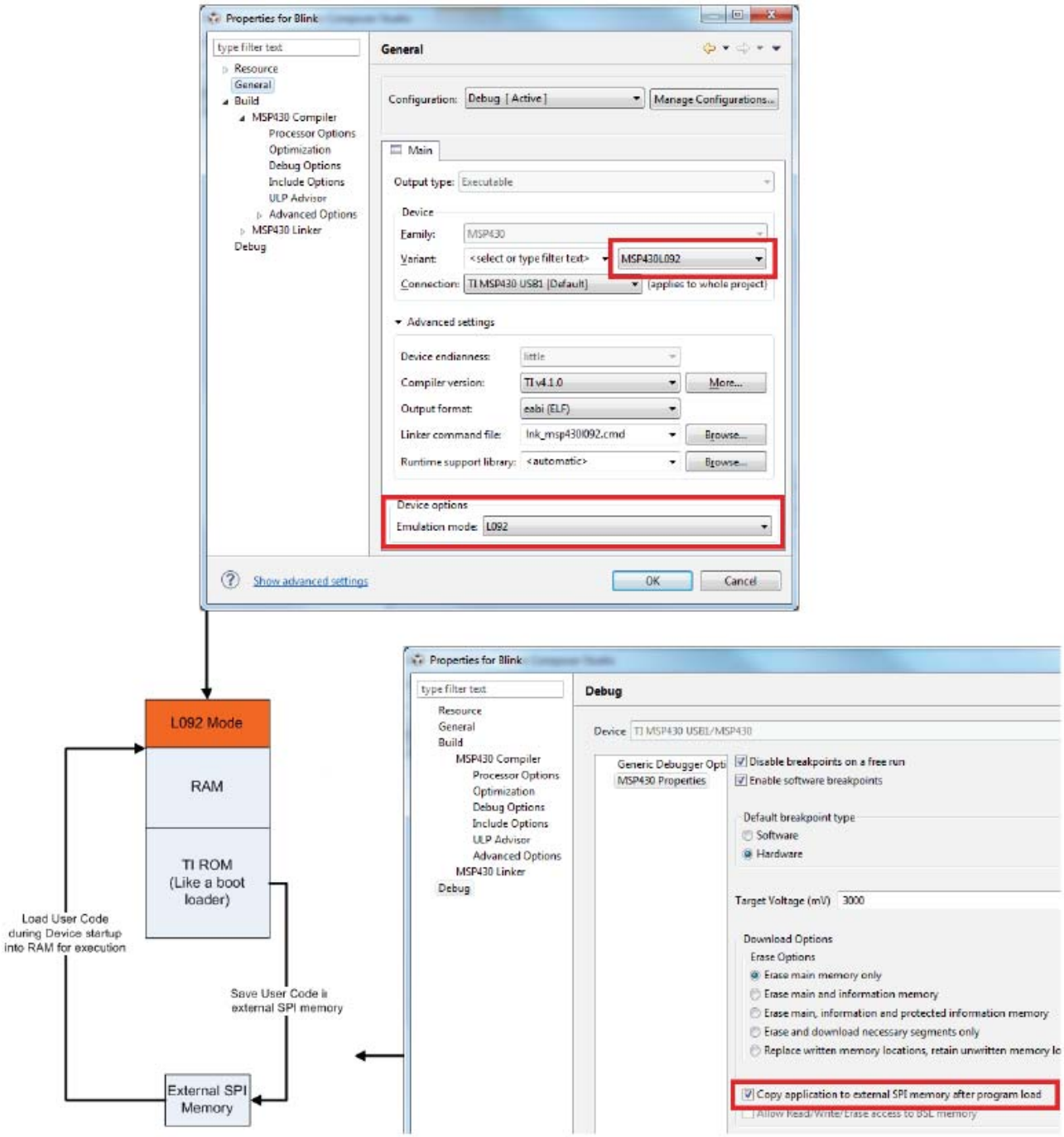
MSP-FET430UIF のターゲット電圧は、1.8 V～3.6V の範囲で補正できます。この電圧は、USB FET からターゲット(電圧)を供給するための 14 ピンのターゲット・コネクタのピン 2 上で利用可能です。ターゲット(電圧)が外部的に供給される場合は、外部電源電圧をターゲット・コネクタのピン 4 に接続して、USB FET がそれに基づいて出力信号のレベルを設定できるようにする必要があります。デバッガがアクティブでない場合のみに変更可能です。

付録 E デバイス固有のメニュー

E.1 MSP430L092

E.1.1 エミュレーション・モード

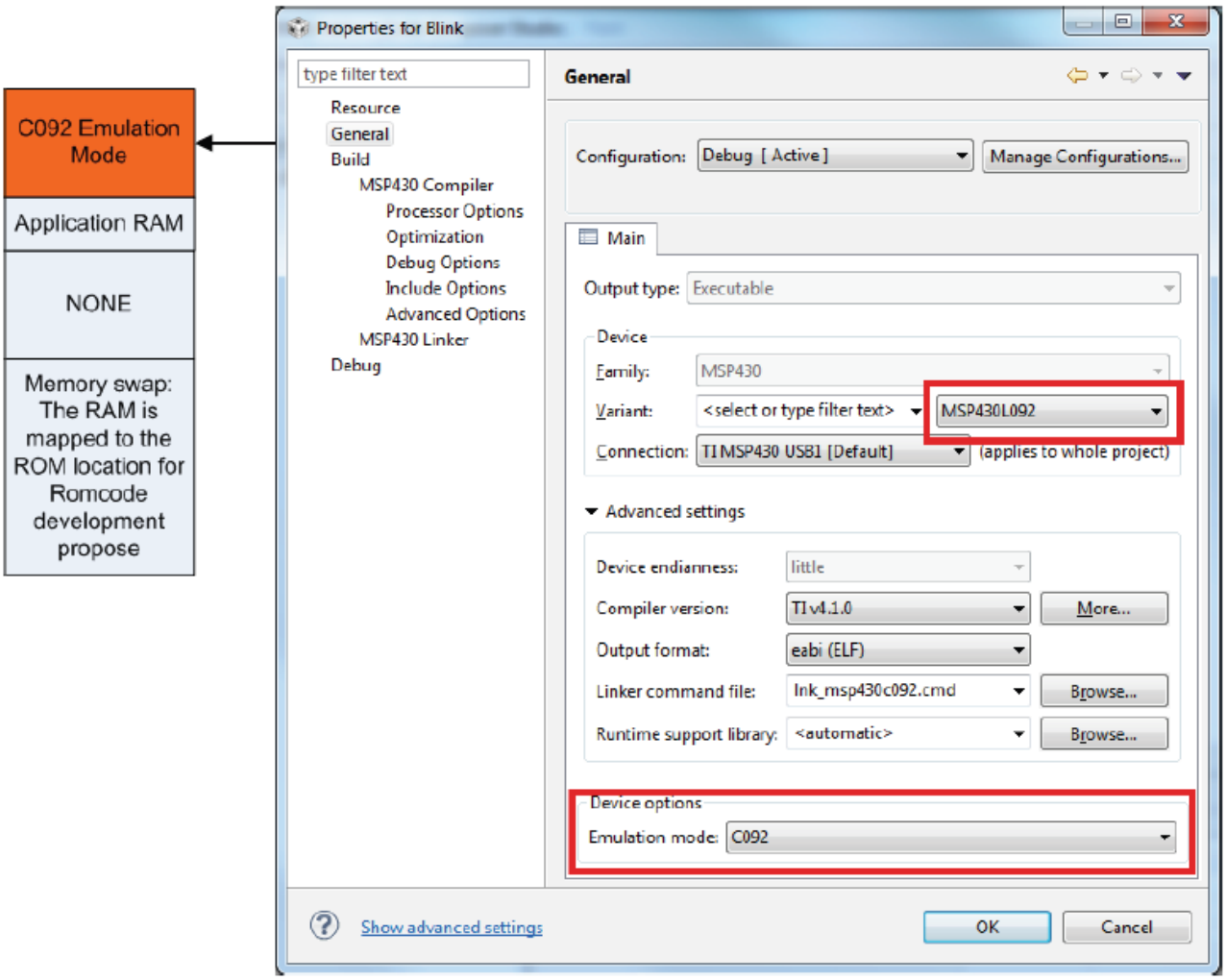
MSP430L092 は、L092 モードと C092 エミュレーション・モードという 2 つのモードで動作可能です。C092 エミュレーション・モードの目的は、L092 を使用してマスクを生成するために、最大 1920 バイトのコードを使用して C092 の機能を再現することです。動作モードは、デバッガを起動する前に CCS に設定する必要があります。モードの選択は図 E-1 に示すように、プロジェクト・プロパティで MSP430L092 を Device Variant(デバイスの種類)として選択した後、画面下部にある Device Options で行います。図 E-2 では、C092 モードを選択する方法を示しています。



図E-1. MSP430L092 のモード

E.1.2 ローダ・コード(Loader Code)

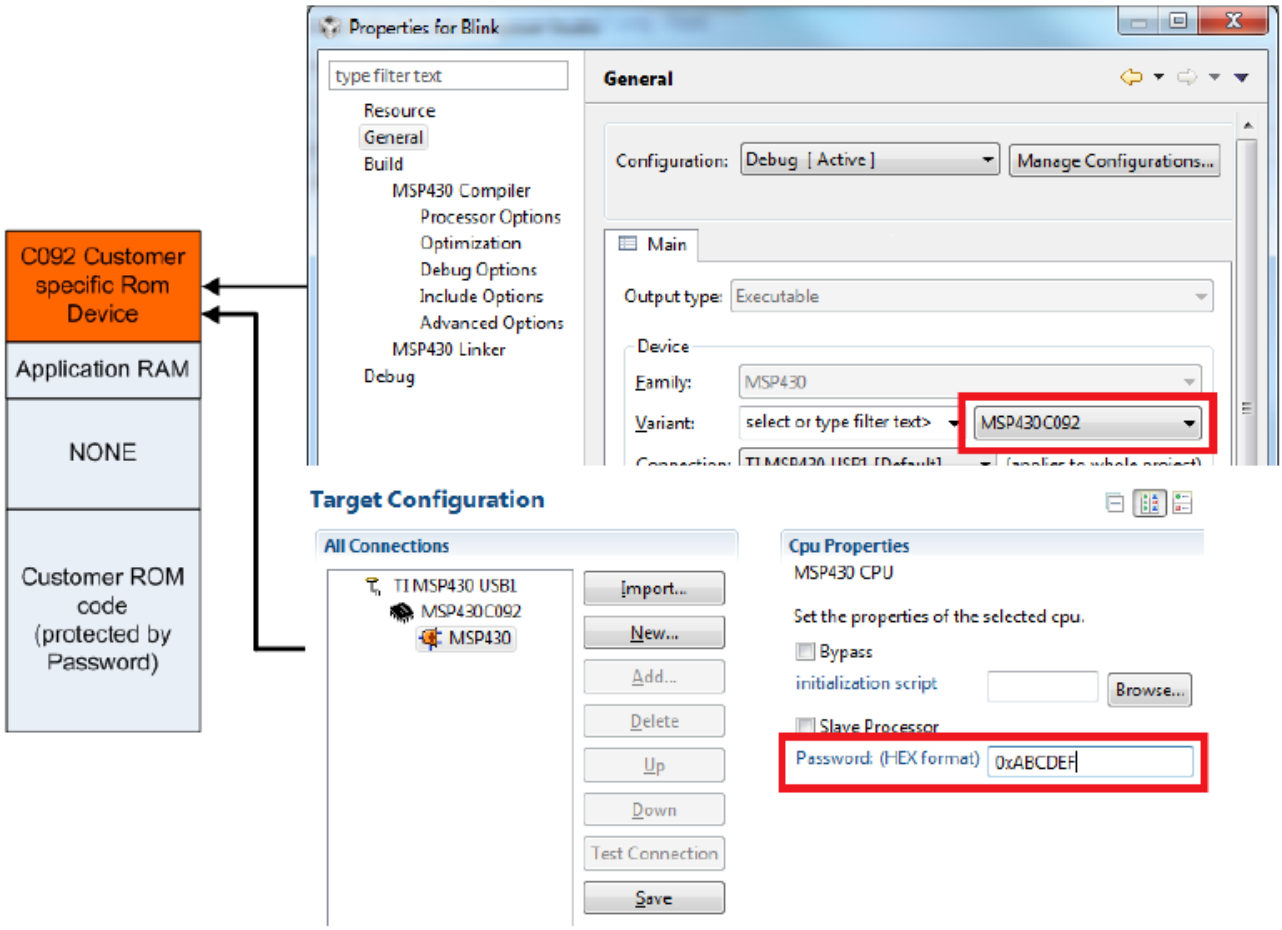
MSP430L092のローダ・コードは、一連の情報を提供するTIのROMコードです。このコードを使用すると、ROMマスクを開発しなくても自律性のある(autonomous) アプリケーションをビルドすることが可能になります。このようなアプリケーションは、ローダ(MSP430L092等)とSPIメモリ・デバイス('95512または '25640等)の組み込まれたMSP430デバイスで構成されています。それらおよび類似のデバイスは、様々なメーカーから入手可能です。ローダ・デバイスとネイティブ0.9V電源電圧用の外部SPIメモリを使用したアプリケーションは主に、後期の開発(late development)、プロトタイピング、小規模なシリーズの製造(small series production)に使用されます。外部コードのダウンロードは、次のように設定できます。CCS Project Properties → Debug → MSP430 Properties →Download Options → Copy application to external SPI memory after program load (プログラムのロード後に、外部SPIメモリにアプリケーションをコピーする) (図E-1参照)



図E-2. C092エミュ x レーション・モードのMSP430L092

E.1.3 C092 のパスワード保護

MSP430C092はカスタマー固有のROMデバイスであり、パスワードで保護されています。デバッグ・セッションを開始するには、パスワードをCCS側に設定する必要があります。お使いのプロジェクトのMSP430C092.CCXMLファイルを開き、Advanced Setup セクション内のTarget Configurations，Advanced Target Configurationをクリックします。MSP430が選択されると、CPU Propertiesが表示されます。図E-3に、CCSv5 ターゲット構成内でHEX(16進数)パスワードを設定する方法を示します。



図E-3. MSP430C092 パスワード・アクセス

E.2 MSP430F5xx と MSP430F6xx BSL のサポート

ほとんどのMSP430F5xxとMSP430F6xxデバイスでは、デフォルトで保護されたカスタムBSLをサポートしています。カスタムBSLをプログラムするには、この保護を次の手順でディセーブルにする必要があります。CCS Project Properties → Debug → MSP430 Properties → Download Options → Allow Read/Write/Erase access to BSL memory (BSLメモリへの読み出し/書き込み/消去アクセスを可能にする) (図E-4参照).

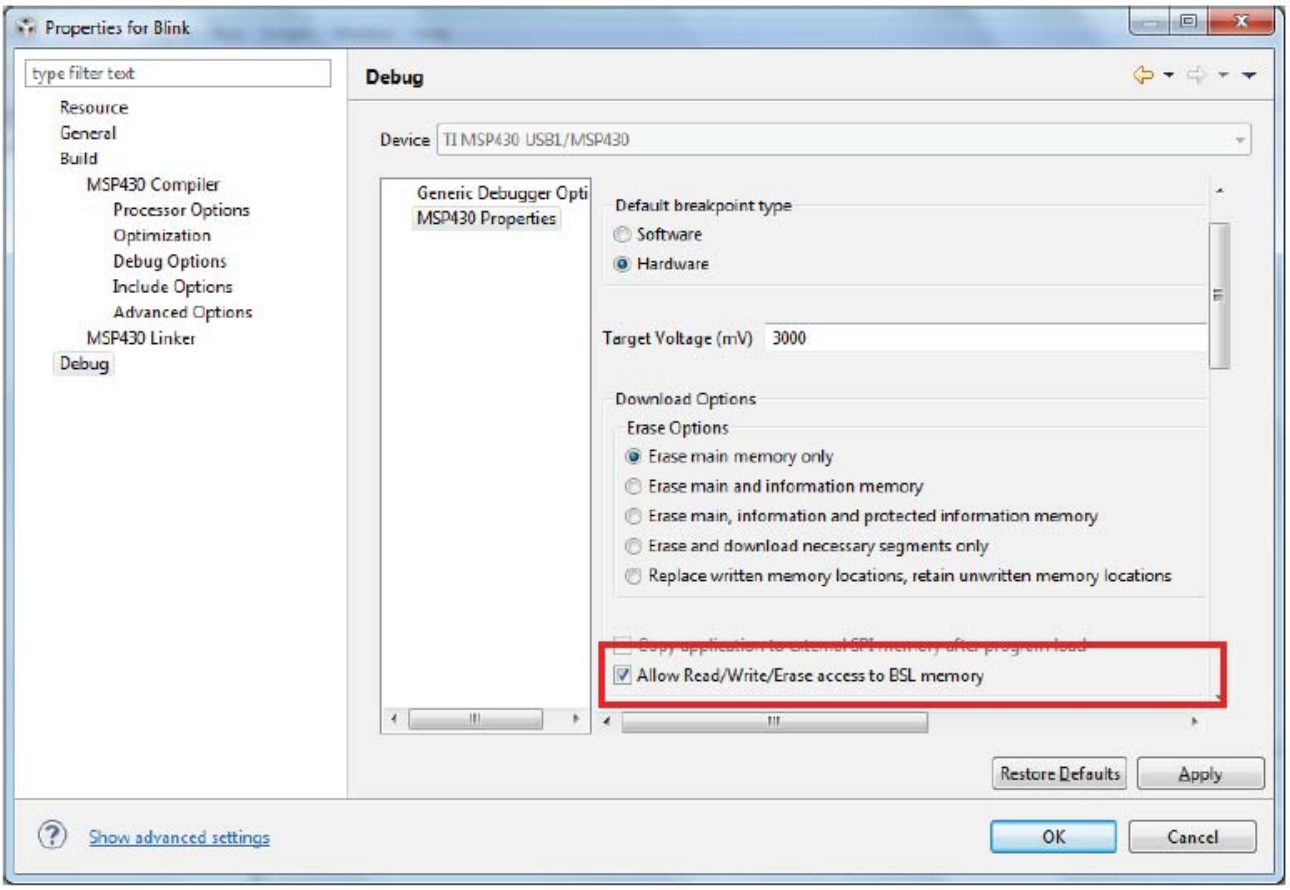
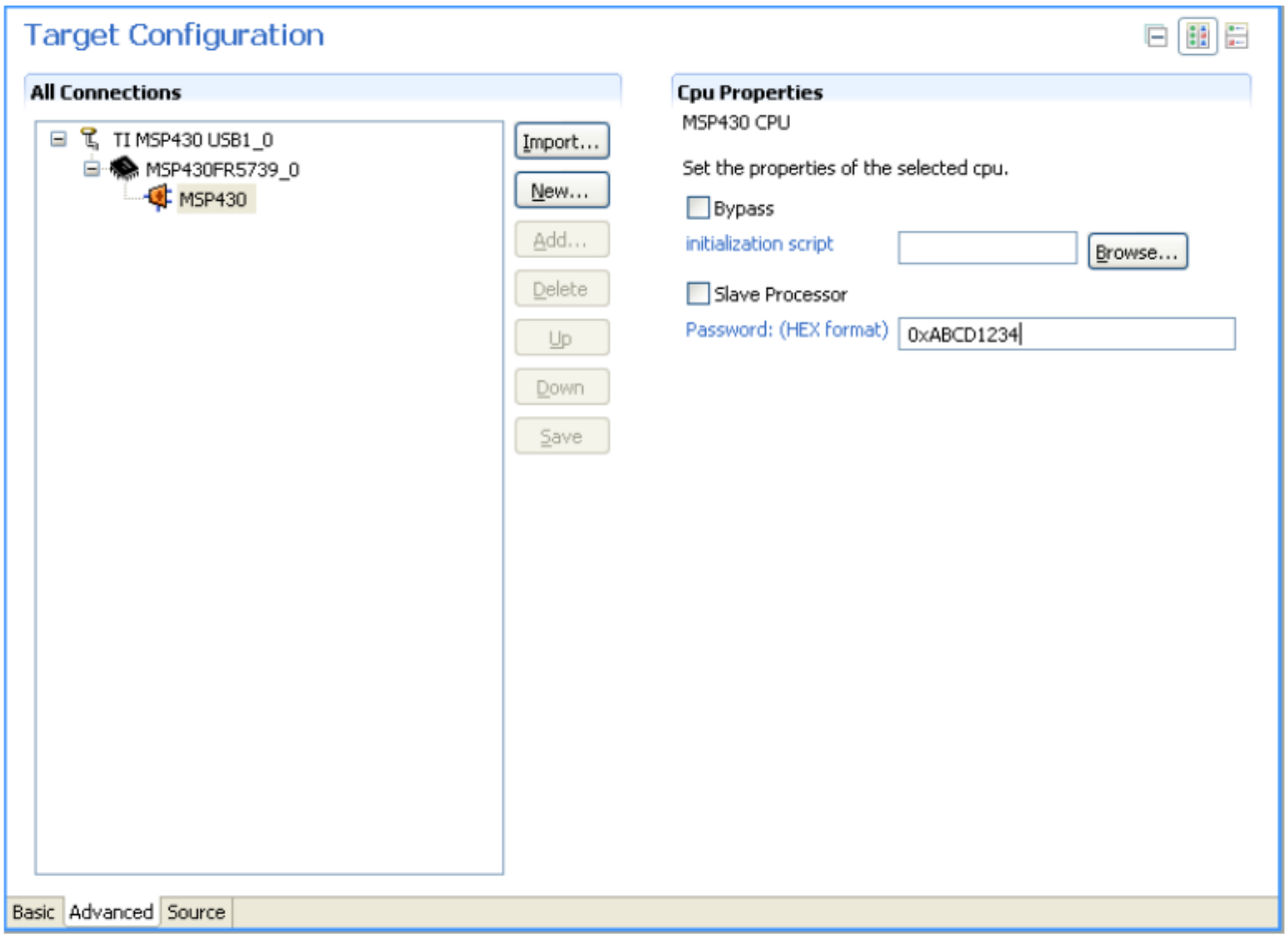


図 E-4. BSLへのアクセスを可能にする

E.3 MSP430F5xx と MSP430F6xx のパスワード保護

選択されたMSP430F5xxとMSP430F6xxデバイスでは、ユーザー・パスワードによるJTAG保護を提供します。このようなMSP430の派生品(derivatives)をデバッグする場合は、デバッグ・セッションを開始するために16進数のJTAGパスワードを設定する必要があります。お使いのプロジェクトのMSP430Fxxxx.CCXMLファイルを開き、Advanced Setup セクション内のTarget ConfigurationsであるAdvanced Target Configurationをクリックします。MSP430が選択されると、CPU Propertiesが表示されます(図E-5参照)。



図E-5. MSP430 パスワード・アクセス

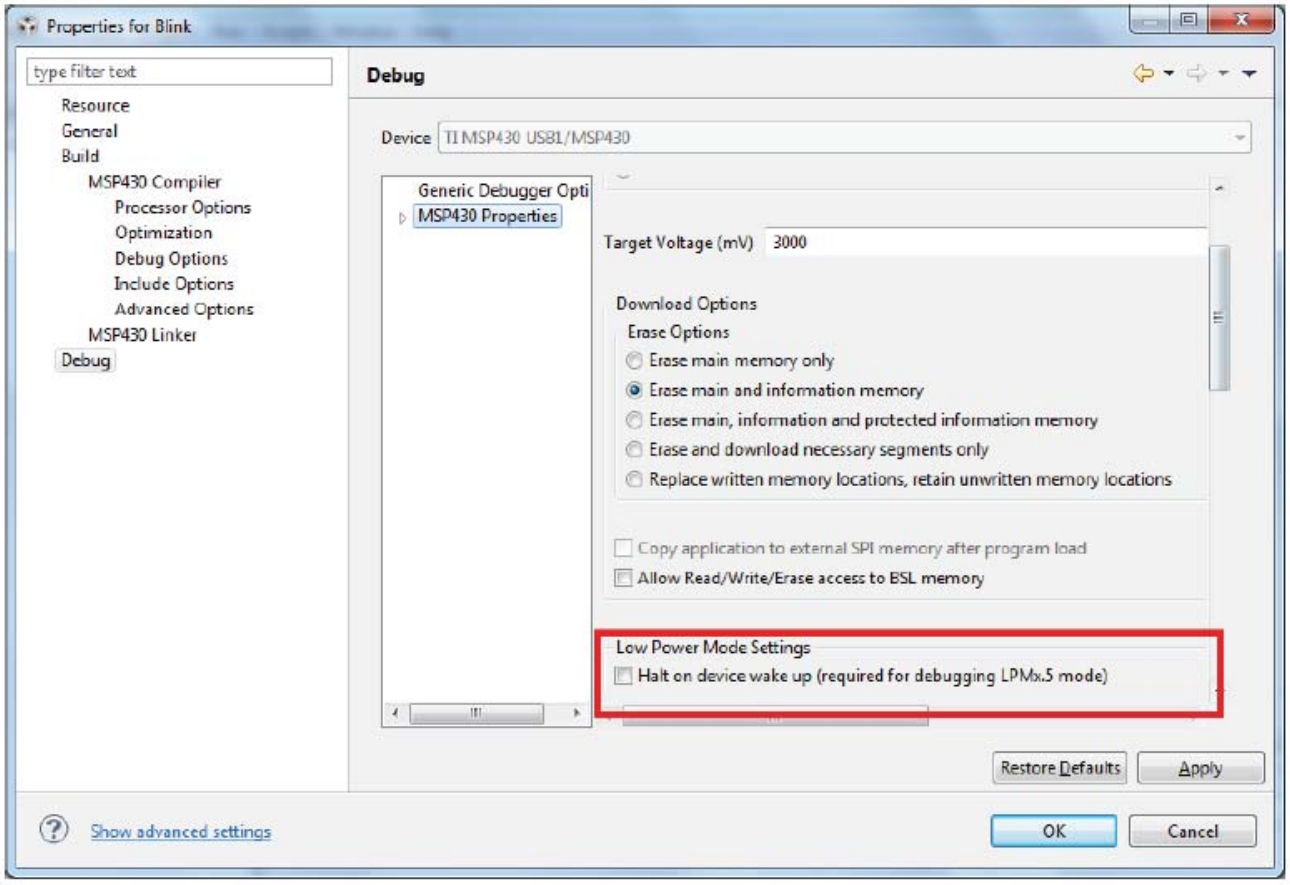
E.4 LPMx.5 CCS デバッグのサポート

LPMx.5 は、他の低消費電力モードとは異なり、開始(entry)と終了(exit)が別個に処理されるという新しい低消費電力モードです。LPMx.5を正しく使用すれば、デバイスの消費電力を最小にすることが可能です。これを実現するために、LPMx.5の開始(entry)によりPMMモジュールのLDOがディセーブルになり、デバイスのコアとJTAGモジュールから電源電圧が除去されます。電源電圧がコアから除去されるため、レジスタの内容とSRAMの内容はすべて失われます。LPMx.5の終了(Exit)によりBORイベントが発生し、システムが強制的に完全にリセットされます。

注: 補足的なLPMx.5の詳細については、対応するMSP430デバイス・ファミリのユーザーズ・ガイドを参照してください。

E.4.1 LPMx.5 を使用したデバッグ

LPMx.5のデバッグ機能をイネーブルにするには、Halt on device wake up(デバイス起動時に停止 (LPMx.5モードのデバッグに必須))チェックボックスをイネーブルにする必要があります(図E-6参照)。LPMx.5のデバッグをイネーブルにするには、次のようにクリックします。Project Properties → Debug → MSP430 Properties → Halt on device wakeup (LPMx.5モードのデバッグに必須)



図E-6. LPMx.5 デバッグ・サポートのイネーブル

LPMx.5のデバッグ・モードがイネーブルにされた場合は、ターゲット・デバイスがLPMx.5モードに入ったり抜けたりするごとに、通知がデバッガのコンソール・ログに表示されます。CCSのHaltボタンまたはResetボタンを押すと、LPMx.5からターゲット・デバイスが起動され、コード起動時に停止されます。LPMx.5の前にアクティブだったすべてのブレークポイントが復元され、自動的に再度アクティベートされます。

E.4.2 LPMx.デバッグの制限事項

ターゲット・デバイスが LPMx.5モードの時は、高度な(advanced)条件ブレークポイントやソフトウェア・ブレークポイントを設定したり削除したりすることはできません。ハードウェア・ブレークポイントを設定することは可能です。また、LPMx.5モードでは、LPMx.5の期間中に設定されたハードウェア・ブレークポイントのみが削除可能です。実行中のターゲットへのアタッチをLPMx.5モードのデバッグと組み合わせることはできません。組み合わせた結果、デバイスがリセットされるためです。

注: LPMx.5のデバッグがアクティブの場合は、オプション"RUN FREE" は、CCS v.5.1.x と CCS v5.2.xでは現在はサポートされません。
補足的なLPMx.5 の詳細については、対応するMSP430デバイス・ファミリのユーザーズ・ガイドを参照してください。

改版履歴

バージョン	説明
SLAU157U	Code Composer Studio v5.2 全体の情報を更新 MSP430FR59xx と MSP430F665x のエミュレーション機能を追加 表 2-1 の LPMx.5 デバッグ・サポートを更新し、セクション E.4 を更新。
SLAU157T	Code Composer Studio v5.1 全体の情報を更新。 CC430F512x, CC430F514x, CC430F614x のエミュレーション機能を追加
SLAU157S	MSP430F52xx, F533x, F643x, F67xx のエミュレーション機能を追加
SLAU157R	付録 E に MSP430FR57xx、LPMx.5、汎用 MSP430 パスワード保護命令のエミュレーション機能を追加
SLAU157Q	MSP430F5310 のエミュレーション機能を追加
SLAU157P	MSP430AFE253、MSP430F532x、MSP430F534x のエミュレーション機能を追加
SLAU157O	MSP430BT5190, MSP430F530x, MSP430F563x のエミュレーション機能を追加 付録 E に MSP430F5xx と MSP430F6xx の BSL サポートを追加し、付録 A の System Pre Init セクションを拡張
SLAU157N	付録 E に MSP430L092 と MSP430C092 のエミュレーション機能とメモリ情報を追加
SLAU157M	MSP430G2xxx, MSP430F51x1, MSP430F51x2, MSP430F550x, MSP430F5510, MSP430F551x, MSP430F552x, MSP430F663x のエミュレーション機能を追加
SLAU157L	Code Composer Studio v4.1 全体の情報を更新 MSP430F44x1, MSP430F461x, MSP430F461x1 のエミュレーション機能を追加
SLAU157K	MSP430F54xxA, MSP430F55xx のエミュレーション機能を追加 アーキテクチャ情報を追加して表 2-1 を更新・拡張
SLAU157J	Code Composer Studio v4 全体の情報を更新 ハードウェアに関する情報を削除 (削除された情報は、MSP430 Hardware Tools User's Guide (SLAU278)に移動)
SLAU157I	セクション 1.7 の MSP-FET430U100A キット、MSP-TS430PZ100A ターゲット・ソケット・モジュールのスキーマ(図 B-19)、PCB (図 B-20)を追加 表 2-1 に CC430F513x, CC430F612x, CC430F613x, MSP430F41x2, MSP430F47x,MSP430FG479, MSP430F471xx のエミュレーション機能を追加 430F47x と MSP430FG47x の情報で MSP-TS430PN80 ターゲット・ソケット・モジュールの回路図(図 B-15) を更新 MSP-FET430Pxx0 キットと MSP-FET430X110 キット全体の情報を削除
SLAU157H	Code Composer Essentials v3.1 全体の情報を更新
SLAU157G	MSP-FET430U5x100 キット と MSP-TS430PZ5x100 ターゲット・ソケット・モジュールの回路図を追加
SLAU157F	セクション 1.7 にクリスタル情報を追加 表 1-1 として、デバッグ・インターフェイスの概要を追加 eZ430-F2013, T2012, eZ430-RF2500 を追加 Code Composer Essentials v3 の全体の情報を更新
SLAU157E	MSP-TS430PW28 ターゲット・ソケット・モジュールの回路図(図 B-5)と PCB (図 B-6)を追加 セクション 1.7 の MSP-FET430U28 キットの内容情報 (DW または PW パッケージのサポート)を更新 MSP430F21x2 のエミュレーション機能を表 2-1 に追加 MSP-TS430PW14 ターゲット・ソケット・モジュールの回路図(図 B-1)を更新 MSP-TS430DA38 ターゲット・ソケット・モジュールの回路図(図 B-7)を更新
SLAU157D	セクション 1.12 を追加 表 2-1 を更新 付録 F を更新
SLAU157C	付録 F を更新 MSP430F22x2, MSP430F241x, MSP430F261x, MSP430FG42x0, MSP430F43x のエミュレーション機能を表 2-1 に追加
SLAU157B	MSP-FET430U40 を MSP-FET430U23x0 に改称 MSP-FET430U40 のスキーマと PCB 図を改称後の MSP-FET430U23x0 の図に差し替え セクション A.1 に FAQ ハードウェア #2 を追加 セクション A.3 に FAQ デバッグ#4 を追加

注: 現在のバージョンのページ番号は、以前のバージョンのページ番号とは異なる場合があります。

ご注意

日本テキサス・インスツルメンツ株式会社（以下TIJといいます）及びTexas Instruments Incorporated（TIJの親会社、以下TIJないしTexas Instruments Incorporatedを総称してTIといいます）は、その製品及びサービスを任意に修正し、改善、改良、その他の変更をし、もしくは製品の製造中止またはサービスの提供を中止する権利を留保します。従いまして、お客様は、発注される前に、関連する最新の情報を取得して頂き、その情報が現在有効かつ完全なものであるかどうかご確認下さい。全ての製品は、お客様とTIJとの間に取引契約が締結されている場合は、当該契約条件に基づき、また当該取引契約が締結されていない場合は、ご注文の受諾の際に提示されるTIJの標準販売契約約款に従って販売されます。

TIは、そのハードウェア製品が、TIの標準保証条件に従い販売時の仕様に対応した性能を有していること、またはお客様とTIJとの間で合意された保証条件に従い合意された仕様に対応した性能を有していることを保証します。検査およびその他の品質管理技法は、TIが当該保証を支援するのに必要とみなす範囲で行なわれております。各デバイスの全てのパラメーターに関する固有の検査は、政府がそれ等の実行を義務づけている場合を除き、必ずしも行なわれておりません。

TIは、製品のアプリケーションに関する支援もしくはお客様の製品の設計について責任を負うことはありません。TI製部品を使用しているお客様の製品及びそのアプリケーションについての責任はお客様にあります。TI製部品を使用したお客様の製品及びアプリケーションについて想定されうる危険を最小のものとするため、適切な設計上および操作上の安全対策は、必ずお客様にてお取り下さい。

TIは、TIの製品もしくはサービスが使用されている組み合せ、機械装置、もしくは方法に関連しているTIの特許権、著作権、回路配置利用権、その他のTIの知的財産権に基づいて何らかのライセンスを許諾するということは明示的にも黙示的にも保証も表明もしておりません。TIが第三者の製品もしくはサービスについて情報を提供することは、TIが当該製品もしくはサービスを使用することについてライセンスを与えるとか、保証もしくは是認するということを意味しません。そのような情報を使用するには第三者の特許その他の知的財産権に基づき当該第三者からライセンスを得なければならない場合もあり、またTIの特許その他の知的財産権に基づきTI からライセンスを得て頂かなければならない場合もあります。

TIのデータ・ブックもしくはデータ・シートの中にある情報を複製することは、その情報に一切の変更を加えること無く、かつその情報と結び付られた全ての保証、条件、制限及び通知と共に複製がなされる限りにおいて許されるものとします。当該情報に変更を加えて複製することは不公正で誤認を生じさせる行為です。TIは、そのような変更された情報や複製については何の義務も責任も負いません。

TIの製品もしくはサービスについてTIにより示された数値、特性、条件その他のパラメーターと異なる、あるいは、それを超えてなされた説明で当該TI製品もしくはサービスを再販売することは、当該TI製品もしくはサービスに対する全ての明示的保証、及び何らかの黙示的保証を無効にし、かつ不公正で誤認を生じさせる行為です。TIは、そのような説明については何の義務も責任もありません。

TIは、TIの製品が、安全でないことが致命的となる用途ないしアプリケーション（例えば、生命維持装置のように、TI製品に不良があった場合に、その不良により相当な確率で死傷等の重篤な事故が発生するようなもの）に使用されることを認めておりません。但し、お客様とTIの双方の権限有る役員が書面でそのような使用について明確に合意した場合は除きます。たとえTIがアプリケーションに関連した情報やサポートを提供したとしても、お客様は、そのようなアプリケーションの安全面及び規制面から見た諸問題を解決するために必要とされる専門的知識及び技術を持ち、かつ、お客様の製品について、またTI製品をそのような安全でないことが致命的となる用途に使用することについて、お客様が全ての法的責任、規制を遵守する責任、及び安全に関する要求事項を満足させる責任を負っていることを認め、かつそのことに同意します。さらに、もし万一、TIの製品がそのような安全でないことが致命的となる用途に使用されたことによって損害が発生し、TIないしその代表者がその損害を賠償した場合は、お客様がTIないしその代表者にその全額の補償をするものとします。

TI製品は、軍事的用途もしくは宇宙航空アプリケーションないし軍事的環境、航空宇宙環境にて使用されるようには設計もされていませんし、使用されることを意図されておりません。但し、当該TI製品が、軍需対応グレード品、若しくは「強化プラスティック」製品としてTIが特別に指定した製品である場合は除きます。TIが軍需対応グレード品として指定した製品のみが軍需品の仕様書に合致いたします。お客様は、TIが軍需対応グレード品として指定していない製品を、軍事的用途もしくは軍事的環境下で使用することは、もっぱらお客様の危険負担においてなされるということ、及び、お客様がもっぱら責任をもって、そのような使用に関して必要とされる全ての法的要求事項及び規制上の要求事項を満足させなければならないことを認め、かつ同意します。

TI製品は、自動車用アプリケーションないし自動車の環境において使用されるようには設計されていませんし、また使用されることを意図されておりません。但し、TIがISO/TS 16949の要求事項を満たしていると特別に指定したTI製品は除きます。お客様は、お客様が当該TI指定品以外のTI製品を自動車用アプリケーションに使用しても、TIは当該要求事項を満たしていなかったことについて、いかなる責任も負わないことを認め、かつ同意します。

弊社半導体製品の取り扱い・保管について

半導体製品は、取り扱い、保管・輸送環境、基板実装条件によっては、お客様での実装前後に破壊/劣化、または故障を起こすことがあります。

弊社半導体製品のお取り扱い、ご使用にあたっては下記の点を遵守して下さい。

1. 静電気
   - 素手で半導体製品単体を触らないこと。どうしても触る必要がある場合は、リストストラップ等で人体からアースをとり、導電性手袋等をして取り扱うこと。
   - 弊社出荷梱包単位（外装から取り出された内装及び個装）又は製品単品で取り扱いを行う場合は、接地された導電性のテーブル上で（導電性マットにアースをとったもの等）、アースをした作業者が行うこと。また、コンテナ等も、導電性のものを使うこと。
   - マウンタやはんだ付け設備等、半導体の実装に関わる全ての装置類は、静電気の帯電を防止する措置を施すこと。
   - 前記のリストストラップ・導電性手袋・テーブル表面及び実装装置類の接地等の静電気帯電防止措置は、常に管理されその機能が確認されていること。
2. 温・湿度環境
   - 温度：0～40℃、相対湿度：40～85%で保管・輸送及び取り扱いを行うこと。（但し、結露しないこと。）
   - 直射日光があたる状態で保管・輸送しないこと。
3. 防湿梱包
   - 防湿梱包品は、開封後は個別推奨保管環境及び期間に従い基板実装すること。
4. 機械的衝撃
   - 梱包品（外装、内装、個装）及び製品単品を落下させたり、衝撃を与えないこと。
5. 熱衝撃
   - はんだ付け時は、最低限260℃以上の高温状態に、10秒以上さらさないこと。（個別推奨条件がある時はそれに従うこと。）
6. 汚染
   - はんだ付け性を損なう、又はアルミ配線腐食の原因となるような汚染物質（硫黄、塩素等ハロゲン）のある環境で保管・輸送しないこと。
   - はんだ付け後は十分にフラックスの洗浄を行うこと。（不純物含有率が一定以下に保証された無洗浄タイプのフラックスは除く。）

以上

2001.11