TEXAS INSTRUMENTS

Code Composer Studio

ユーザーズ・マニュアル

2000 ***Digital Signal Processing Solutions***

SPRU328 翻訳版

JAJU020 WAS SPRU486

Code Composer Studio
ユーザーズ・マニュアル

2000 年 11 月

TEXAS INSTRUMENTS

ご 注 意

日本テキサス・インスツルメンツ株式会社（以下 TI といいます）は、通知をすることなくその製品を変更し、もしくは半導体集積回路製品またはサービスの製造または提供を中止することがありますので、お客様は、発注される前に、これから参照しようとする資料が最新のものであることを確実にするため、最新版の資料を取得するようお勧めします。

TI は、その半導体集積回路製品および関連するソフトウェアが、TI の標準保証条件に従い販売の際の現行の仕様書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特定の検査は、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。

半導体集積回路製品を使用する或る種の用途の中には、死亡、傷害、または財産もしくは環境に深刻な被害をもたらす危険の可能性を包含するものがあります。（以下、これらを「重大用途」といいます。）

TI の半導体集積回路製品は、生命維持の用途、装置、システム、その他の重大用途に使用できるように設計も、意図も、承認も、また保証もされておりません。

TI の製品を当該重大用途に組込むことは、お客様独自のリスクでなされることと解釈されます。TI 製品を当該用途に使用される場合は、事前に TI の役員の書面による承諾を必要とします。危険な可能性を有する用途に関する質問は、TI の営業所を通じて、TI 迄お寄せ下さい。

お客様の用途に TI 製品を使用することに伴う危険を最小のものとするため、製品固有の危険性を最小にするための、適切な設計上および作動する上での安全対策は、お客様がとらなくてはなりません。

TI は製品の使用用途に関する支援、お客様の製品の設計、ソフトウェアの性能、または特許侵害もしくはサービスに対する責任を負うものではありません。また TI は、その半導体集積回路製品もしくはサービスが使用されうる、もしくは使用されている組み合せ、機械装置、もしくは方法をカバーしている、またはそれ等に関連している特許権、著作権、回路配置利用権、その他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示もしておりません。

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、露結しないこと。）
   - 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

TEXAS INSTRUMENTS

最初にお読みください

このマニュアルについて

本書では、Code Composer Studio が提供する開発環境により、リアルタイムの組み込み型ソフトウェア・アプリケーションを作成する方法、およびそのデバッグ方法について説明します。

表記規則

本書では、以下の表記規則を使用します。

- プログラム・リスト、プログラム例、および対話表示は、タイプライタの活字に似た特殊な活字 (`special typeface`) で示してあります。例は強調のため、ボールド (**`bold version`**) で示してあります。対話表示についても、ユーザが入力するコマンドとシステムが表示する項目 (プロンプト、コマンド出力、エラー・メッセージなど) とを区別するために、ボールド (**`bold version`**) で示してあります。

  C コードの例を、以下に示します。

  ```
  #include <stdio.h>
  main()
  {
    printf("hello, world\n");
  }
  ```

- 構文の記述、命令、コマンド、疑似命令はボールド (**bold**)、パラメータはイタリック (*italic*) で示します。構文でボールドの部分は、その表記通り入力する箇所です。構文でイタリックの部分は、入力する情報の種類を示しています。コマンド行で入力する構文は、中心に示します。テキスト・ファイルで使用する構文は、左詰めで示します。
- 大括弧 ([ ]) は、任意のパラメータを示します。任意のパラメータを使用する場合、この括弧内の情報を入力します。大括弧がボールドである場合を除き、大括弧そのものは入力不要です。

当社発行の関連文献

お客様のターゲット・プロセッサ、および関連のサポート・ツールに関する詳しい情報は、Code Composer Studio のオンライン・ヘルプから関連文献を参照してください。

関連文献

本書の参考文献として、次の文献を参考にできます。

American National Standard for Information Systems-Programming Language C X3.159-1989 ― 米国規格協会刊行 (C 言語に関する ANSI 規格)

The C Programming Language **(** 第 **2** 版 **)** ― Brian W. Kernighan と Dennis M. Ritchie との共著、Prentice-Hall, Englewood Cliffs (New Jersey) 刊行 (1988)

Programming in C ― Kochan, Steve G. 著、Hayden Book Company 刊行

商標

Code Composer Studio、DSP/BIOS、Probe Point(s)、および RTDX は、Texas Instruments Incorporated の商標です。

Pentium は、Intel Corporation の登録商標です。

Windows と Windows NT は、Microsoft Corporation の登録商標です。

サポートが必要な場合・・・

❑ World-Wide Web Sites

TI Online	http://www.ti.com
日本 TI	http://www.tij.co.jp
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/pic/home.htm
DSP Solutions	http://www.ti.com/dsps
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
DSP Hotline	(281) 274-2320	Fax: (281) 274-2324	Email: dsph@ti.com
DSP Modem BBS	(281) 274-2323		

DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32	Email: epic@ti.com
Deutsch	+33 1 30 70 11 68		
English	+33 1 30 70 11 65		
Francais	+33 1 30 70 11 64		
Italiano	+33 1 30 70 11 67		
EPIC Modem BBS	+33 1 30 70 11 99		
European Factory Repair	+33 4 93 22 25 40		
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10	

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200	
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002	
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828	
Korea DSP Modem BBS	+82 2 551 2914		
Singapore DSP Hotline		Fax: +65 390 7179	
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718	
Taiwan DSP Modem BBS	+886 2 376 2592		

Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/

❑ 日本

TI 製品についてのお問い合わせ および資料の請求	プロダクト・インフォメーション・センター（PIC）へご連絡ください。 0120-81-0026	Fax: 0120-81-0036	Email: pic-japan@ti.com

❑ Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: comments@books.sc.ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

目次

第 1 章

Code Composer Studio のセットアップ

本章では、Code Composer Studio のインストール方法およびセットアップ方法について説明します。

項目 **ページ**

1.1 システム要件

Code Composer Studio を使用するには、最低限、次の動作環境が必要です。

- ❑ IBM 製 PC (または互換 PC)
- ❑ Microsoft Windows 95、Windows 98、または Windows NT 4.0
- ❑ 32 MB の RAM、100 MB の空きスペースのあるハードディスク、Pentium プロセッサ、SVGA ディスプレイ (800x600)

1.2 Code Composer Studio のインストール

このインストレーション・プロセスを完了するには、次の 2 つのフェーズが必要です。

1) ユーザのシステムに Code Composer Studio ソフトウェアをインストールします。
2) Code Composer Studio のセットアップ･アプリケーションを実行して、Code Composer Studio がユーザのターゲット・ボードまたはシミュレータと通信できるように、インターフェイスの設定を行います。

この節の後半では、ユーザ・システムに本ソフトウェアをインストールする方法について説明します。Code Composer Studio のセットアップ・アプリケーションの実行方法の詳細は、1.3 節「Code Composer Studio のセットアップ」を参照してください。

Windows 95/98/NT に Code Composer Studio をインストールする方法

Windows 95/98/NT に Code Composer Studio をインストールする手順は、次のとおりです。

1) Windows を起動した状態で、インストレーション CD を CD-ROM ドライブに挿入します。
2) Windows エクスプローラから CD-ROM ドライブを選択して、setup.exe を実行します。なお、Windows NT の場合は、必ず Windows NT 用の管理者権限を使用して Code Composer Studio をインストールしなければなりません。

インストーレションが完了すると、「CCStudio」および「Setup CCStudio」のプログラム・アイコンがデスクトップ上に作成されます。

1.3 Code Composer Studio のセットアップ

Code Composer Studio Setup は、Code Composer Studio がユーザのターゲット・ボードやシミュレータとの通信を可能にするインターフェイスを確立します。セットアップ･プログラムを実行する前に、1.2 節「Code Composer Studio のインストール」の説明に従ってソフトウェアをインストールしておく必要があります。

デスクトップ上の「Setup CCStudio」アイコンをダブルクリックして、Code Composer Studio Setup をスタートさせます。

Code Composer Studio Setup の画面のプロンプトに従って、システム設定を定義してください。さらに詳しいヘルプが必要な場合は、Help メニューを参照してください。

注 : トラブルシューティング
Code Composer Studio **ソフトウェアのセットアップに問題がある場合は、付録 「問い合わせの多い質問」に記載されているトラブルシューティングのヒントを参照してください。**

1.4 Code Composer Studio の起動

インストレーションおよびセットアップが完了したら、Code Composer Studio Tutorial を実行してみてください。このチュートリアルでは、本バージョンに組み込まれた新しい機能や、本ソフトウェアがもつ多くの機能に精通してもらうことを目的としています。Code Composer Studio で実際の仕事を始める前に、このチュートリアルで実習をしておくと、本ソフトウェアの習得時間が短縮され、また多様な基礎的手順についての知識を得ることができます。

Code Composer Studio チュートリアルにアクセスするには、次のステップを踏みます。

1) デスクトップの「CCStudio」アイコンをダブルクリックして、Code Composer Studio を始動させます。
2) 「Code Composer Studio Help」メニューから、「Tutorial」→「Code Composer Studio Tutorial」の順に選択します。

1.5 オンライン・ヘルプの使用

Code Composer Studio 全般に関するヘルプを得るには、「Help」→「General Help」の順に選択します。

「Code Composer Studio General Help」画面の目次および索引から、Code Composer Studio 製品のあらゆるツール、特徴、または機能に関する情報を見たり、検索したりできます。

第 2 章

Code Composer Studio の基本

本章では、Code Composer Studio の基本概念およびその特徴について説明します。本章で説明する概念は、すべてのデバッグ・セッションに共通する重要な概念です。

項目 **ページ**

2.1 Code Composer Studio のウィンドウとツールバー

Code Composer Studio 環境内では、「Edit」ウィンドウを除くすべてのウィンドウ、およびすべてのツールバーは、自由にドッキング可能です。つまり、Code Composer Studio メイン・ウィンドウ枠内のどの場所にでもウィンドウやツールバーを移動させ、配置することができます。

Code Composer Studio のドッキング可能なウィンドウとツールバーは、メイン・ウィンドウの枠を越えて移動させることができ、デスクトップ上の任意の場所に配置することもできます。ツールバーを移動するには、目的のツールバーを新しい位置までクリック・アンド・ドラッグするだけです。

ウィンドウをメイン・ウィンドウの枠外に移動する方法

1) ウィンドウ内で右クリックして、オープンしたコンテキスト・メニューから「Allow Docking」を選択します。
2) 目的のウィンドウのタイトル・バーを左クリックし、そのウィンドウをデスクトップ上の任意の場所にドラッグします。

ドッキング可能なウィンドウにはコンテキスト・メニューの機能があります。このメニューは、ウィンドウの配置を制御する次の 3 つのオプションを表示します。

Allow Docking	ウィンドウのドッキングのオン／オフを切り換えます。
Hide	アクティブ・ウィンドウを、他のすべてのウィンドウの背面に隠します。
Float in the Main Window	ドッキングをオフにして、アクティブ・ウィンドウをメイン・ウィンドウ内の任意の場所に配置できるようにします。

2.1.1 コンテキスト依存型メニュー

Code Composer Studio のウィンドウには、コンテキスト・メニューの機能があります。コンテキスト・メニューを開くには、目的のウィンドウ内で右クリックします。

コンテキスト・メニューは、その特定のウィンドウに適用可能な一連のオプションおよびまたはコマンドを表示します。たとえば、「Project View」ウィンドウに表示されるプロジェクト・ファイルを右クリックし、表示されるコンテキスト・メニューから適切なアクションを選択すると、プロジェクトの操作（ソース／ GEL ファイルの追加／削除、ビルド・オプションの設定など）を行うことができます（プロジェクトの管理方法については、第 10 章「プロジェクト環境」を参照）。

2.2 「Dis-Assembly」ウィンドウ

ユーザがプログラムを、ターゲットまたはシミュレータにロードすると、Code Composer Studio デバッガは、自動的に「Dis-Assembly」ウィンドウを開きます。

「Dis-Assembly」ウィンドウには、デバッグに必要な逆アセンブルされた命令やシンボル情報が表示されます。逆アセンブリは、アセンブリの逆プロセスによって、メモリの内容をアセンブリ言語コードで表示します。シンボル情報は、シンボルおよび英数字の文字列から構成されるもので、ターゲットのアドレスや値を表します。

「Dis-Assembly」ウィンドウは、アセンブリ言語命令ごとに、逆アセンブルした命令、その命令の場所を示すアドレス、および対応するオペコード（その命令を表すマシン・コード）を表示します。デバッガは、ターゲット・ボードからオペコードを読み取り、それらを逆アセンブルし、アクティブなプログラム・カウンタ (PC) が示す場所から始まるシンボル情報を追加することによって、逆アセンブリ・リストを作成します。現在の PC が置かれている行は、黄色で強調表示されます。

ステップ実行コマンドを使用してプログラムをステップ実行すると、PC は次の有効な命令まで進みます。プログラム・コードの C で書かれたセクションに対しては、C ソースとアセンブリの混合コード表示をすることができます（2.2.6 節 「C ソースとアセンブリを混合コードで表示する方法」を参照）。

2.2.1 複数の 「Dis-Assembly」ウィンドウを開く方法

複数の 「Dis-Assembly」ウィンドウを開くには、「View」→「Dis-Assembly」の順に選択するか、または 「Debug 」ツールバーから、「View disassembly」ボタンを使用します。

View disassembly: 

最初に表示されるウィンドウは、PC の現在位置を追尾します。複数の「Dis-Assembly」ウィンドウを表示すると、タイトル・バーには Dis-Assembly <n> と表示されます。この場合の「n」は、ウィンドウの固有番号を示します。

2.2.2 開始アドレスの変更

「Dis-Assembly」ウィンドウの開始アドレスを変更するには、そのウィンドウのアドレス・フィールド上をダブルクリックして、「View Address」ダイアログ・ボックスを表示します。次に、ユーザが指定する開始アドレスを入力します。この指定には、任意の絶対数、または有効なプログラム・シンボルによる C 表記を使用します。

2.2.3 ブレークポイント、プローブ・ポイント、プロファイル・ポイントを「Dis-Assembly」ウィンドウから管理する方法

ブレークポイント、プローブ・ポイント、およびプロファイル・ポイントを設定または消去するには、「Dis-Assembly」ウィンドウの該当行にカーソルを置き、「Debug」メニュー、または 「Profiler」メニューから適切なコマンドを選択するか、または「Project」ツールバーの適切なボタンをクリックします。ブレークポイントは、該当行をダブルクリックすると簡単に設定することができます。これらの設定ポイントは、背景に色をつけて表示されます。背景色は、設定したポイントの種類によって異なります。たとえば、ブレークポイントとプローブ・ポイントを同じ行に設定すると、その背景色は紫色と青色で表示されます。

2.2.4 強調表示色の変更

「Option」→「Colors」の順に選択すると、現在の PC およびデバッグ・ポイント（ブレークポイント／プロファイル・ポイント／プローブ・ポイント）用のデフォルトの表示色を変更することができます。

2.2.5 「Dis-Assembly Style」オプションの設定

Code Composer Studio には、「Dis-Assembly」ウィンドウに表示する情報の表示形態を変更するためのオプションが複数あります。「Dis-Assembly Style Options」ダイアログ・ボックスを使用すると、デバッグ・セッション用の特別な表示オプションを入力することができます。たとえば、アドレッシングの基数として 16 進数か 10 進数を選択することができます。

「Dis-Assembly Style」オプションを設定する方法

1) 「Option」→「Dis-Assembly Style」の順に選択します。

   または

   「Dis-Assembly」ウィンドウ内で右クリックします。コンテキスト・メニューから、「Properties」→「Dis-Assembly Options」の順に選択します。

2) 「Dis-Assembly Style Options」ダイアログ・ボックス内の選択項目を指定します。

3) 「OK」をクリックします。

「Dis-Assembly」ウィンドウの内容は、直ちに新しいスタイルに更新されます。

2.2.6 C ソースとアセンブリを混合コードで表示する方法

「Dis-Assembly」ウィンドウ内で逆アセンブルされた命令を表示することに加えて、Code Composer Studio デバッガでは、逆アセンブル・コードがインターリーブされた C ソース・コードを表示することができます。

C ソースとアセンブリを混合コードで表示する方法

ご使用のターゲットまたはシミュレータにプログラムをロードした後で、次の手順を実行します。

1) 「View」→「Mixed Source/ASM」の順に選択します。このオプションが選択されると、チェック・マークが付きます。

2) 「Debug」→「Go Main」の順に選択します。

   デバッガは、プログラムの実行を開始し、main() で実行を停止します。関連した C ソース・ファイルが、自動的に 「Edit」ウィンドウに表示されます（詳細は、2.8 節「ターゲット・プロセッサのリセット」を参照）。C 言語のソース・コードの各行の下に、対応する逆アセンブリ言語の命令が表示されることに注意してください。「Dis-Assembly」ウィンドウの場合と同じに、PC の位置は黄色で強調表示されます。

C ソース・ファイルは、アセンブリ・コードと一緒に表示するか、アセンブリ・コードなしに表示するかを選択することができます。選択内容を変更するには、「View」→「Mixed Source/ASM」を切り換えるか、「Edit」ウィンドウ内で右クリックしてコンテキスト・メニューを開き、現在の選択内容に応じて 「Mixed Mode」、または「Source Mode」を選択します。

2.3 「Memory」ウィンドウ

Code Composer Studio デバッガでは、特定位置のメモリの内容を表示することができます。

メモリの内容を表示する方法

1) 「View」→「Memory」の順に選択します。

   または

   「Debug」ツールバー上で 「View Memory」ボタンをクリックします。

   View Memory: 

2) 「Memory」ウィンドウが表示される前に、まず 「Memory Window Options」ダイアログ・ボックスが表示されます。このダイアログでは、「Memory」ウィンドウのさまざまな特性を指定することができます。

   「Memory Window Options」ダイアログ・ボックス内に、希望する特性を入力します（2.3.1 節 「「Memory Window」オプションの設定」を参照）。

3) 「OK」をクリックします。「Memory」ウィンドウが表示されます。

アクティブな 「Memory」ウィンドウの特性のいずれかを変更するには、「Memory」ウィンドウ内で右クリックし、コンテキスト・メニューから「Properties」を選択します。「Memory Window Options」ダイアログ・ボックスが表示されます。

メモリ位置の内容を編集するには、「Memory」ウィンドウ内の該当するアドレスをダブルクリックするか、「Edit」→「Memory」→「Edit」の順に選択します。「Edit Memory」ダイアログ・ボックスが表示されます（2.3.2 節 「メモリ位置の編集」を参照）。

2.3.1 「Memory Window」オプションの設定

「Memory Window Options」ダイアログ・ボックスでは、「Memory」ウィンドウのさまざまな特性を指定することができます。

「Memory Window Options」ダイアログでは、次のオプションを設定できます。

Address 表示するメモリ位置の開始アドレスを入力します。

Q Value Q 値を使用して整数を表示できます。この値は、より精度の高い 2 進数値として、整数値を表すために使用されます。小数点が、その 2 進数値に挿入されます。最下位ビット (LSB) からのオフセットは、次のように Q 値によって決定されます。

$$New\_integer\_value = integer / 2^{Q\ value}$$

Q value が xx である場合は、小数点が最下位ビット (LSB) から xx 桁変位している符号付き 2 の補数の整数であることを示します。

Format ドロップダウン・リストから、メモリ表示のフォーマット設定を選択します。

Use IEEE Float IEEE 浮動小数点フォーマットを使用して表示する場合は、このオプションを選択します。

Page ドロップダウン・リストから、ページのタイプ（Program、Data、または I/O）を選択します。

Enable Reference Buffer

このチェック・ボックスを選択すると、指定されたメモリ・エリアのスナップショットを保管して、後で比較に使用することができます。

たとえば、「Enable Reference Buffer」を選択し、アドレス範囲を 0x0000..0x002F に指定するとします。指定された範囲のメモリの内容は、ホスト・メモリに保管されます。ターゲットの停止、ブレークポイントのヒット、メモリのリフレッシュなどを行うたびに、デバッガは、Reference Buffer の内容を現在のメモリの内容と比較します。「Memory」ウィンドウ内でこのメモリ空間をスクロールすると、変更点が赤色で表示されます。

Start Address Reference Buffer に保管するメモリ位置の開始アドレスを入力します。このフィールドがアクティブになるのは、「Enable Reference Buffer」が選択されているときだけです。

End Address Reference Buffer に保管するメモリ位置の終了アドレスを入力します。このフィールドがアクティブになるのは、「Enable Reference Buffer」が選択されているときだけです。

Update Reference Buffer Automatically
このチェック・ボックスを選択すると、指定されたアドレス範囲で、Reference Buffer の内容を、現在のメモリの内容で自動的に上書きします。このチェック・ボックスが選択されない場合は、Reference Buffer の内容は変更されません。このオプションがアクティブになるのは、「Enable Reference Buffer」が選択されているときだけです。

すべての入力フィールドは、C 式入力フィールドです。シンボル名を含む式を、「Memory Window Options」ダイアログの開始アドレスを指定するために使用できます。詳細は、2.3.3.1 節 「式の中でシンボルを使用する方法」を参照してください。

表示フォーマット

「Memory」ウィンドウには、複数の異なるフォーマットでメモリの内容を表示することができます。サポートされている表示フォーマットは、以下のとおりです。

C-style hex ワードは、プレフィックス 0x を付けて表示されます。

Hex 16 進数を表示するための TI フォーマット

Signed integer 値は、符号付き整数として表示されます。

Unsigned integer 値は、符号なし整数として表示されます。

Character 各ワードの LSB に相当する文字が表示されます。

Packed character 各ワードは、8 ビット文字の集まりとして表示されます。

Floating point 値は、10 進数の浮動小数点フォーマットで表示されます。

Exponential float 値は、指数浮動小数点フォーマットで表示されます。

Binary 値は、2 進数フォーマットで表示されます。

2.3.2 メモリ位置の編集

メモリ位置を編集するには、次の方法のどちらかを行います。

- ❏ 「Memory」ウィンドウで、編集するデータをダブルクリックします。または
- ❏ 「Edit」→「Memory」→「Edit」コマンドの順に選択します。

どちらの方法でも、「Edit Memory」ダイアログ・ボックスが開きます。「Memory」ウィンドウ内でメモリ位置をダブルクリックした場合、「Address」フィールドと「Data」フィールドには選択された位置のアドレスとデータ値が入っています。

メニューのコマンドを使用した場合、「Address」フィールドと「Data」フィールドにはデフォルト値が入っています。希望するアドレス位置を指定するには、編集したいアドレスを「Address」フィールドに入力します。Tab キーを押すか、「Data」フィールドをクリックすると、指定されたアドレスの内容が、指定された値に更新されます。その位置のデータ値を変更するには、「Data」フィールドに希望する値を入力し、「Done」をクリックします。また、「Address」フィールド上でスクロール・ボタンを使用して、メモリ位置上を移動することもできます。

「Memory」ウィンドウ内のある位置をダブルクリックした場合、「Data」フィールドのデフォルト・フォーマットは、「Memory」ウィンドウ内のフォーマットと同じになります。表示フォーマットが整数の場合、16 進数フォーマット（プレフィックス 0x 付き）か、10 進数フォーマット（プレフィックスなし）のどちらかで値を入力することができます。また、表示フォーマットに互換性がある場合は、浮動小数点値を入力することもできます。

すべての入力フィールドは、C 式入力フィールドです。

2.3.3 C 式入力フィールド

数値を必要とするすべての入力フィールドは､C 式入力フィールドです。C 式入力フィールドでは、C 関数またはアセンブリ言語ラベルの名前を含む任意の有効な C の式を入力することができます。この C の式は単一の数値に縮小され、次の例のように表示されます。

```
MyFunction
0x1000 + 2 * 35
(int) MyFunction + 0x100
PC + 0x10
```

上記のフィールドに対するデフォルトの表示フォーマットは 16 進数ですが、特殊なフォーマット記号を使用して変更することができます。この記号は、「Watch」ウィンドウ の記号とほぼ同じです（8.3 節 「「Watch」ウィンドウの表示フォーマット」を参照）。

2.3.3.1 式の中でシンボルを使用する方法

数値入力を必要とするすべてのフィールドに、シンボル名を含む式を使用することができます。しかし、Code Composer Studio がシンボルを解釈する方法は、オブジェクト・ファイルにシンボリック・デバッグ情報が入っているかどうかによって異なります。

シンボルが C ソース・ファイル内で定義されていて、ファイルをビルドするときにシンボリック・デバッグ情報 (-g) が指定されている場合、そのシンボルは、指定されたアドレスにあるメモリの内容を表わす変数として扱われます。

シンボリック・デバッグ情報がない場合、すべてのシンボルはアドレスとして扱われます。

たとえば、「Memory Window Options」ダイアログ、または「Graph Property」ダイアログ・ボックスで、シンボル名を使用して開始アドレスを指定する場合は、次のようになります。

シンボリック・デバッグ情報が使用可能であるときに、単純変数のアドレスを使用したい場合は、シンボル名（変数）の前にアンパーサンド (&) を付ける必要があります。このアンパーサンドは、アドレスを表現するための標準 C 言語表記です。アンパーサンドを指定しない場合、その式は評価され、シンボルの内容になります。ただし、配列を表す変数は例外です。C 表記では、アンパーサンド付きの名前を指定すると、その配列の先頭のアドレスを指すことが暗黙に指定されます。

シンボリック・デバッグ情報が使用できない場合は、シンボル名（アドレス）だけを指定してください。

2.4 CPU レジスタ

ターゲット・プロセッサの CPU レジスタとペリフェラル・レジスタ（C5000）は、「Register」ウィンドウを使用して表示することができます。レジスタの内容は、「Edit Registers」ダイアログを使用して編集できます。

2.4.1 レジスタの表示

ターゲット・プロセッサの CPU レジスタの内容を表示するには、コマンド 「View」→「CPU Registers」→「CPU Register」の順に選択します。または、「Debug」ツールバーの「View Registers」ボタンを選択しても、「CPU Register」ウィンドウを表示できます。

Register Window: 

この 「Register」ウィンドウでは、「Edit Registers」ダイアログを使用してレジスタを編集できます。

2.4.2 レジスタの編集

レジスタの内容を編集する方法

「Edit」→「Edit Register」の順に選択します。「Edit Register」ダイアログ・ボックスが表示されます。

または

「Register」ウィンドウで、レジスタをダブルクリックするか、またはそのウィンドウ内で右クリックし、コンテキスト・メニューから 「Edit Register」を選択します。

「Edit Register」ダイアログには、次のオプションが用意されています。

Register	レジスタ名を入力するか、ドロップダウン・リストからレジスタを選択して、編集したいレジスタを指定します。
Value	このフィールドには、指定されたレジスタの現在の値が 16 進数で表示されます。このフィールドには、16 進数（プレフィックス 0x 付き）または 10 進数（プレフィックスなし）で別の値を入力できます。また、任意の有効な C 言語の式を入力することもできます。

レジスタの値を変更した後で、「Close」をクリックして変更内容を適用します。このダイアログ・ボックスもクローズします。

注： シミュレータは、ペリフェラル・レジスタをサポートしません

シミュレータには、ペリフェラル・レジスタのサポートが組み込まれていません。

2.5 COFF ファイルのロード

実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードに有効な COFF ファイルをロードするには、「File」→「Load Program」の順に選択します。「Load Program」ダイアログ・ボックスで、必要なファイルを選択し、「開く (O)」をクリックします。

このコマンドは、COFF ファイルからシンボル情報だけでなく、データもロードします。

2.5.1 シンボル情報だけのロード

シンボル情報だけをロードするには、「File」→「Load Symbol」の順に選択します。「Load Symbol Info」ダイアログ・ボックスで、必要なファイルを選択し、「開く (O)」をクリックします。

この機能は、デバッガがオブジェクト・コードをロードできないか、またはロードする必要がないようなデバッグ環境においては便利です。コードが ROM に入っている場合などがこれに該当します。

このコマンドは、既存のシンボル・テーブルを消去してから、新しいシンボル・テーブルをロードします。ただし、メモリの変更やプログラム・エントリ・ポイントの設定は行いません。

2.5.2 COFF ファイルの再ロード

実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードに有効な COFF ファイルを再ロードするには、「File」→「Reload Program」の順に選択します。プログラムを再ロードする前に、前回のロード以降に、そのファイルが変更されたかどうかを確かめるチェックが行われます。そのファイルが変更されていた場合は、プログラムとそのシンボル情報の両方が再ロードされます。変更が検出されない場合は、プログラムだけが再ロードされ、そのプログラムに関連したシンボル・テーブルは再ロードされません。

このコマンドは、ターゲット・メモリが破壊された場合にプログラムを再ロードするときに便利です。

2.5.3 「Program Load」オプションの設定

「Program Load Options」ダイアログ・ボックスを開くには、「Option」→「Program Load」の順に選択します。「Program Load Options」ダイアログ・ボックスには、次のオプションが用意されています。

Perform verification after Program Load

デフォルトでは、このチェック・ボックスにチェックが付いています。つまり、Code Composer Studio は、選択されたメモリを読み出すことによってプログラムのロードが正常に行われたことを検証します。このオプションのチェックをオフにすると、Code Composer Studio は、この検証を実行しません。

Load Program After Build

このオプションにチェックを付けると、プロジェクトのビルド直後に実行可能プログラムがロードされます。これにより、1 つのビルド後に生成された最新のシンボル情報は、必ずターゲットに含まれるようになります。

2.6 シングル・ステップ

コードをシングル・ステップ実行するには、「Debug」ツールバーの中から次のボタンを使用します。

 **Single Step**

「Debug」ツールバー上のこのボタンをクリックするか、「Debug」→「StepInto」の順に選択すると、コードをシングル・ステップで実行させることができます。C ソース・モードに入っている場合、このコマンドは単一の C 命令をステップ実行します。それ以外の場合は、単一のアセンブリ命令をステップ実行します。

 **Step Over**

このコマンドを使用すると、現行の関数内で個々の文をステップ実行することができます。関数呼び出しが行われると、その関数は完了するまで実行されます。ただし、関数呼び出しの後、停止するまでの間にブレークポイント が検出されると、実行はその時点で停止します。step over を実行するには、「Debug」ツールバー上のこのボタンをクリックするか、「Debug」→「StepOver」の順に選択します。

C だけでそのファイルを表示するか、または同時にアセンブリ・ファイルを表示することもできます。C ソース・モード（2.2.6 節 「C ソースとアセンブリを混合コードで表示する方法」を参照）では、このコマンドは C だけの命令をステップ実行します。それ以外の場合は、単一のアセンブリ命令をステップ実行します。しかし、プロセッサのパイプラインを保護するために、遅延分岐または遅延呼び出しの後に続く複数の命令は、同じ文の一部と見なすことができます。この場合、step over コマンドは、一度に複数の命令を実行することになります。

 **Step Out**

サブルーチンの実行途中である場合には、step out コマンドを選択して、そのサブルーチンの実行を完了させることができます。この実行は、現行の関数が呼び出し元の関数に戻った後で停止します。step out を実行するには、「Debug」ツールバー上のこのボタンをクリックするか、「Debug」→「StepOut」の順に選択します。

C ソース・モードでは、ローカル・フレーム・ポインタを使用して、標準ランタイム C スタックから呼び出し元の関数が判別されます。それ以外のモードでは、呼び出し元の関数への戻りアドレスが、スタックの最上部にセットされているものと見なされます。アセンブリ・ルーチンがスタックを使用してその他の情報を保管している場合、この step out コマンドは正しく機能しません。

 **Run to Cursor**

このコマンドを使用すると、実行が「Dis-Assembly」ウィンドウのカーソル位置に到達するまで、ロード済みのプログラムを実行することができます。run to cursor を実行するには、「Debug」→「Run to Cursor」の順に選択します。

2.6.1 複数のステップ操作

ステップ実行コマンドを複数回実行する方法

1) コマンド 「Debug」→「Multiple Operation」の順に選択します。
2) 「Multiple Operation」ダイアログ・ボックスで、ドロップダウン・リストからステップ実行コマンドを選択します。
3) このコマンドを実行する回数を指定します。
4) 「OK」をクリックします。

同じステップ実行コマンド、または別のステップ実行コマンドを複数回実行するには、上記の手順を繰り返します。

2.7 Run、Halt、Animate、Run Free

 **Run**

「Debug」ツールバー上でこのボタンをクリックするか、「Debug」→「Run」の順に選択すると、現在の PC 位置からプログラムを実行することができます。ブレークポイントが検出されるまで、継続して実行されます。

 **Halt**

「Debug」ツールバー上でこのボタンをクリックするか、「Debug」→「Halt」の順に選択すると、プログラムの実行を停止させることができます。

 **Animate**

「Debug」ツールバー上でこのボタンをクリックするか、「Debug」→「Animate」の順に選択すると、プログラムをアニメーション表示することができます。このプログラムは、ブレークポイントを検出するまで実行されます。ブレークポイントが検出されると、プローブ・ポイントに接続されていないウィンドウが更新されてから、実行が再開されます。アニメーションを停止するには、「Debug」→「Halt」の順に選択します。アニメーションの速度を制御するには、「Option」→「Animate Speed」の順に選択します。

Run Free

すべてのブレークポイント（プローブ・ポイントとプロファイル・ポイントを含む）をディセーブルにした後、現行の PC 位置からプログラムの実行を開始します。このコマンドを実行するには、「Debug」→「Run Free」の順に選択します。free run の間にターゲット・プロセッサにアクセスするオペレーションがあると、ブレークポイントは復元されます。実行を停止するには、「Debug」→「Halt」コマンドを使用します。JTAG ベースのデバイス・ドライバを使用してエミュレーションを行う場合、このコマンドを発行すると、ターゲット・プロセッサも切り離されるので、JTAG ケーブルまたは MPSD ケーブルを取り外すことができます。free run の間に、ターゲット・プロセッサのハードウェア・リセットを実行することもできます。

注 : シミュレータは Run Free をサポートしません
シミュレータを実行しているときは、run free **機能は使用できません。**

2.7.1 アニメーション速度の設定

アニメーション速度とは、ブレークポイント相互間の最小時間です。アニメーション・モードでは、ブレークポイントが検出されるまで、プログラムは継続して実行されます。ブレークポイントが検出されると、プログラムの実行は停止し、プローブ・ポイントに接続されていないすべてのウィンドウが更新されます。この後、このプログラムは実行を再開し、次のブレークポイントが検出されるまで続きます。この機能では、各ブレークポイントでプロセッサの状態がアニメーション表示されます。プローブ・ポイントでは、プローブ・ポイントに接続されているウィンドウを更新した後、実行が再開されます。

アニメーション速度を設定する方法

1) コマンド「Option」→「Animate Speed」の順に選択します。
2) 「Animate Speed Properties」ダイアログ・ボックスに、アニメーション速度を秒単位で入力します。
3) 「OK」をクリックします。

前回のブレークポイントからこの最小時間が経過した後に、プログラムの実行が再開されます。

animate を実行するには、「Debug」→「Animate」の順に選択します。animate を停止するには、「Debug」→「Halt」の順に選択します。

2.8 ターゲット・プロセッサのリセット

ターゲット・プロセッサをリセットするには、次のコマンドが使用できます。

Reset DSP

「Debug」→「Reset DSP」の順にコマンドを選択すると、レジスタのすべての内容が電源投入時の状態に初期化され、プログラムの実行が停止されます。ターゲット・ボードがこのコマンドに応答せず、しかもカーネル・ベースのデバイス・ドライバを使用している場合には、DSP カーネルが壊れる恐れがあります。この場合は、そのカーネルを再ロードする必要があります。シミュレータは、ターゲット・シミュレーション指定に従って、レジスタのすべての内容を電源投入時の状態に初期化します。

Load Kernel

カーネル・ベースのデバッガ（JTAG 以外）を使用する場合は、DSP カーネルがホスト・コンピュータとの通信を行います。このカーネルが壊れると、デバイス・ドライバがターゲットと通信できなくなることがあります。この場合は、「Debug」→「Load Kernel」コマンドを実行して、カーネルを正しい状態に復元しなければなりません。

Restart

「Debug」→「Restart」コマンドを選択すると、PC は、現在ロードされているプログラムのエントリ・ポイントに戻されます。ただし、このコマンドはプログラムの実行を開始しません。

Go Main

「Debug」→「Go Main」コマンドを選択すると、プログラム内のシンボル main に一時的なブレークポイントが設定され、実行が開始されます。このブレークポイントは、実行の停止時、またはブレークポイントの検出時に削除されます。この方法は、C プログラマがアプリケーションを始動する場合に便利です。実行が main() で停止すると、関連したソース・ファイルが自動的にロードされます。

2.9 データ値のコピー

メモリのブロックを新しい場所にコピーする方法

1) 「Edit」→「Memory」→「Copy」の順に選択します。
2) 「Setup for Copying」ダイアログ・ボックスで、Source 情報を入力します。

   Address ― コピーされるメモリ・ブロックの開始アドレス

   Length ― コピーされるメモリ・ブロックの長さ
3) Destination 情報を入力します。

   Address ― メモリ・ブロックのコピー先のアドレス
4) 「OK」をクリックして、コピーを実行します。

すべての入力フィールドは、C 式入力フィールドです。

2.10 メモリ位置の埋め込み

メモリ・ブロックに指定値を埋め込む方法

1) 「Edit」→「Memory」→「Fill」の順に選択します。
2) 「Setup Filling Memory」ダイアログ・ボックスで、次の情報を入力します。

   Address －埋め込まれるメモリ・ブロックの開始アドレス

   Length －埋め込まれるメモリ・ブロックの長さ

   Fill Pattern －メモリ・ブロックに埋め込まれるパターン
3) 「OK」をクリックして、埋め込みを実行します。

開始アドレスから始まり、開始アドレス + 長さ－ 1 までのすべての位置に、「Fill Pattern」フィールドに入力された埋めこみパターンが埋め込まれます。

すべての入力フィールドは、C 式入力フィールドです。

2.11 変数の編集

変数を編集する方法

1) 「Edit」→「Edit Variable」の順に選択します。
2) 「Edit Variable」ダイアログ・ボックスで、次の情報を入力します。

   Variable －編集対象の変数の名前

   Value －新しい値
3) 「OK」をクリックして、編集を実行します。

「Watch」ウィンドウ 式を編集する場合にも、「Edit Variable」ダイアログ・ボックスが使用されます（8.2 節 「「Watch」ウィンドウで変数を編集する方法」を参照）。

すべての入力フィールドは、C 式入力フィールドです。

TI 固定小数点プロセッサに対しては、実際のターゲットまたはシミュレーション・ターゲットが複数のページから構成されている場合には、@ 記号を使用して特定のページを指定できます。この記号を入力した後、キーワード（prog、data、または io）のどれか 1 つを入力してください。このキーワードは、このページがプログラム・ページであるか、データ・ページであるか、I/O ページであるかを指定します。たとえば、次のとおりです。

```
*0x1000@prog = 0
*0x1000@data = myVar
*0x2000@io = 0
```

2.12 コマンド行の編集

「Command Line」ダイアログは、式の入力、または GEL 関数の実行に便利な方法を提供します。組み込み GEL 関数のいずれかを実行することも、ロードされているユーザ専用の GEL 関数を実行することもできます（12.5 節 「GEL 関数のロードとアンロード」を参照）。

コマンドを実行する方法

1) 「Edit」→「Edit Command」の順に選択します。
2) 「Command Line」ダイアログ・ボックスの「Command」フィールドに、式または GEL 関数を入力します。
3) 「OK」をクリックして、そのコマンドを実行します。

また、「GEL」ツールバーを使用して、組み込み GEL 式、またはユーザ定義の GEL 関数を入力し、実行することもできます。このツールバーにアクセスするには、「View」→「GEL Toolbar」の順に選択します。「GEL」ツールバー内のスクロール可能リストには、最近実行された GEL 関数の履歴が入っています。以前に使用されたコマンドを実行するには、そのコマンドを選択し、次のボタンをクリックします。

Execute Command: 

次の例は、「GEL」ツールバー、または 「Command Line」ダイアログで入力できるコマンドを示しています。

式を入力して、変数を変更する例

```
PC = c_int00
```

組み込み GEL 関数を使用して、プログラムをロードする例

```
GEL_Load("c:\\myprog.out")
```

ユーザ専用の GEL 関数を実行する例

```
MyFunc()
```

2.13 ウィンドウのリフレッシュ

通常、すべてのウィンドウは、ターゲット・ボードの現在のステータスを表示しています。しかし、ウィンドウをプローブ・ポイントに接続すると、このウィンドウには最新の情報が含まれていない可能性があります（第 4 章「ブレークポイントとプローブ・ポイント」を参照）。ウィンドウを更新するには、「Window」→「Refresh」の順に選択します。このコマンドは、アクティブ・ウィンドウを現在のターゲット・データで更新します。

2.14 コール・スタックの表示

「Call Stack」ウィンドウを使用すると、ユーザがデバッグ中のプログラム内の現在の位置に至るまでの、一連の関数呼び出しを調べることができます。コール・スタックを表示するには、「View」→「Call Stack」の順に選択するか、「Debug」ツールバー上の 「View Stack」ボタンをクリックします。

View Stack: 

コール・スタック上の関数のソース・コードを表示するには、その関数の上でダブルクリックします。そのソース・コードが入っている 「Edit」ウィンドウが表示されます。このウィンドウでは、選択した関数内の現在実行中の行にカーソルが設定されます。「Call Stack」ウィンドウで関数を選択すると、その関数のスコープ内にあるローカル変数を監視することもできます。

コール・スタックは、C プログラムでのみ作動します。呼び出し元関数は、ランタイム・スタック上でフレーム・ポインタのリンク・リストを検索して決定されます。プログラムには、スタック・セクションと main 関数が必要です。これらがない場合は、コール・スタックは次のメッセージを表示します。

C source is not available.

2.14.1 ローカル変数の監視

現在実行中ではないが、コール・スタック内に存在するような関数がある場合、この関数のローカル変数（自動変数）を監視することができます。「Call Stack」ウィンドウを使用して、興味のある関数のローカル変数にスコープを変更します。スコープを変更した後、その関数のすべてのローカル変数を監視（または、「Watch」ウィンドウに追加）することができます。

2.15 ワークスペースのセーブとロード

複数のデバッグ・セッション間で、ワークスペースと呼ばれる現在の作業環境をセーブしたり、ロードしたりすることができます。また、同じデバッグ・セッション内で、複数の異なる作業環境を切り換えることもできます。

ワークスペースをセーブする方法

1) 「File」→「Workspace」→「Save Workspace」の順に選択します。
2) 「Save Workspace」ダイアログ・ボックスで、「ファイル名 (N)」フィールドにワークスペースの名前を入力します。
3) 「保存 (S)」をクリックします。

Code Composer Studio を終了すると、現在のワークスペースは、default.wks ファイルに自動的にセーブされます（2.15.2 節 「デフォルトのワークスペース」を参照）。

ワークスペースをロードする方法

1) 「File」→「Workspace」→「Load Workspace」の順に選択します。
2) 「Load Workspace」ダイアログ・ボックスで、「ファイル名 (N)」フィールドにワークスペース・ファイルの名前を入力します。
3) 「開く (O)」をクリックします。

Code Composer Studio を始動するたびに、特定のワークスペースを自動的にロードさせることができます（2.15.1 節 「ワークスペースを自動的にロードする方法」を参照）。

ワークスペース内にセーブされるものは、次のとおりです。

- ❏ 親ウィンドウ（サイズと位置を含む）
- ❏ 子ウィンドウ（サイズと位置を含む）
- ❏ ブレークポイント、プローブ・ポイント、プロファイル・ポイント
- ❏ プロファイラ・オプション
- ❏ 現在のプロジェクト
- ❏ 現在ロードされている GEL 関数
- ❏ メモリ・マップ
- ❏ アニメーション速度オプション
- ❏ ファイル I/O セットアップ

ワークスペース内にセーブされないものは、次のとおりです。

- ❑ 現在のフォント
- ❑ 現在の色設定
- ❑ ターゲット・メモリ、ターゲット・プログラム、またはターゲット・プロセッサの状態
- ❑ Edit と find/replace の浮動ツール
- ❑ 「build」ウィンドウ内のエラー・メッセージと進捗メッセージ
- ❑ GEL 出力ウィンドウ
- ❑ スキャン依存ウィンドウ
- ❑ Disassembly style オプション

注 : フォントと色設定のセーブ

フォントと色設定は、プロファイラ・オプション、メモリ・マップ、およびアニメーション速度と一緒に、cc_user.dat **ファイルに自動的にセーブされ、別のセッションでロードされます。**

注 : ターゲット・プロセッサの初期化

GEL **拡張言語を使用して、ターゲット・プロセッサの状態を初期化することができます (**3.6 **節 「**GEL **関数の自動実行」を参照)。**

2.15.1 ワークスペースを自動的にロードする方法

Code Composer Studio を始動するたびに、特定のワークスペースを自動的にロードさせることができます。自動的にロードさせるには、アプリケーションの始動時に、最初のコマンド行パラメータとして、そのワークスペースの名前を指定します。このパラメータの末尾には、拡張子 .wks を指定しなければなりません。これは、Code Composer Studio が、そのパラメータをワークスペース・ファイルとして認識できるようにするためです。末尾に .wks を指定しないと、Code Composer Studio は、そのワークスペースを GEL ファイルとしてロードしようとします。

ワークスペースを自動的にロード する方法 (Windows 95/98/NT)

1) ウィンドウズのエクスプローラで、Code Composer Studio の実行プログラムを選択します。
2) その実行プログラムを右クリックし、「ショートカットの作成 (S)」を選択して、実行プログラムへのショートカットを作成します。
3) 作成されたショートカット上で右クリックし、「プロパティ (P)」を選択します。
4) 「プロパティ」ダイアログ・ボックスで、「ショートカット」タブを選択します。
5) 「リンク先 (T)」フィールドに、Code Composer Studio 実行プログラムのパス名とファイル名が入っていることを確認します。たとえば、次のとおりです。
   c:\ti\cc\bin\cc_app.exe
6) このパス名の後ろに、使用するワークスペース・ファイルの名前（末尾は .wks でなければなりません）を追加します。たとえば、次のとおりです。
   c:\ti\cc\bin\cc_app.exe myspace.wks.

注 : デフォルトのワークスペース

ファイル default.wks **を指定すると、**Code Composer Studio **を始動するたびに、デフォルトのワークスペースが自動的にロードされます。**

2.15.2 デフォルトのワークスペース

Code Composer Studio を終了すると、現在のワークスペースは、default.wks ファイルに自動的にセーブされます。終了した場所から再開したい場合は、Code Composer Studio を始動し、「File」→「Workspace」→「Load Workspace」コマンドの順に選択して、default.wks をロードします。コマンド行上で、最初のプログラム・パラメータとして default.wks を指定して Code Composer Studio を始動すると、Code Composer Studio を始動するたびに、デフォルトのワークスペースが自動的にロードされます。

アプリケーションを始動するたびにデフォルトのワークスペースを自動的にロードするように、Code Composer Studio をセットアップすることができます。このセットアップを行うと、複数のセッション間で、自分の環境を自動的にセーブしたり、ロードしたりすることができます。

始動時に、ワークスペースと GEL ファイルを自動的にロードする方法

Code Composer Studio の始動時に、ワークスペース、およびこれに関連した GEL ファイルの両方をロードすることができます。

1) Code Composer Studio の実行中に、必要な GEL 関数をロードし、環境全体をワークスペースとしてセーブします。セーブを行うには、「File」→「Workspace」→「Save Workspace」の順に選択します（12.5 節「GEL 関数のロードとアンロード」を参照）。
2) 始動時に、このワークスペースを自動的にロードするように、Code Composer Studio をセットアップします（2.15.1 節 「ワークスペースを自動的にロードする方法」を参照）。

マルチプロセッサ環境で作業している場合は、3.6 節 「GEL 関数の自動実行」を参照してください。

第 3 章

Code Composer Studio によるマルチプロセッシング

Code Composer Studio は、複数のプロセッサを同時にデバッグする機能をもっています。パラレル・デバッグ・マネージャを使用して、すべてのプロセッサにコマンドをブロードキャストしたり、プロセッサを個別に選択したりします。パラレル・デバッグ・マネージャを使用する前に、Code Composer Studio Setup プログラムを使用して、マルチプロセッシング環境を設定しておく必要があります。この構成がセットアップされた後に Code Composer Studio が起動されると、画面上に 「Parallel Debug Manager」メニュー・バーが表示されます。

注 : シミュレータはマルチプロセッシングをサポートしません

シミュレータは、複数の DSP **システムをサポートしません。エミュレータ・バージョンの** Code Composer Studio **デバッガを、マルチ** DSP **ターゲット・ボードと一緒に使用する必要があります。**

項目 **ページ**

3.1 パラレル・デバッグ・マネージャ

パラレル・デバッグ・マネージャを使用すると、複数のプロセッサを同期させることができるようになります。複数のプロセッサ、およびその複数のプロセッサをサポートするデバイス・ドライバを使用している場合、Code Composer Studio を始動すると、パラレル・デバッグ・マメージャが有効になります。「Parallel Debug Manager」メニューでは、各プロセッサを制御するために、個々の親ウィンドウを開くことができます。また、プロセッサのグループにコマンドをブロードキャストすることもできます (3.3 節「プロセッサのグループ化」を参照)。

「Parallel Debug Manager」は、フローティング・ツールバーです。このツールバーを他のウィンドウの手前に表示するには、「Options」→「Always On Top」を選択します。メニュー・バーのサイズを変更すると、メニュー・バーの形状を変えることができます。コマンドのボタンとメニューは、ウィンドウのサイズに合わせて折り返します。

3.2 個々の親ウィンドウを開く方法

「Parallel Debug Manager」メニューから、各プロセッサを制御するウィンドウを開くことができます。

プロセッサ上でデバッグ・セッションを選択する方法

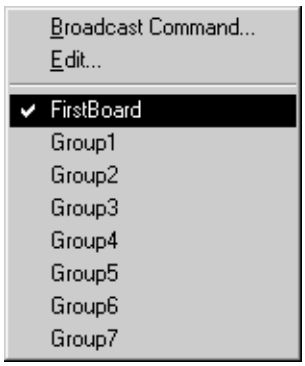
1) 「Parallel Debug Manager」メニュー・バーから 「Open」を選択します。
2) プロセッサの名前を選択します。これで、選択されたプロセッサ用の親ウィンドウが開きます。

注： 現在のシステム構成

「Open」メニューには、現在のシステム構成で定義されている物理ターゲット・ボードまたは DSP、およびシミュレートする DSP のリストが含まれています。「Open」メニューの下に正しいプロセッサ・リストが表示されない場合には、Code Composer Studio が Code Composer Studio Setup プログラムを使用して正しく設定されているかどうかを確認してください。

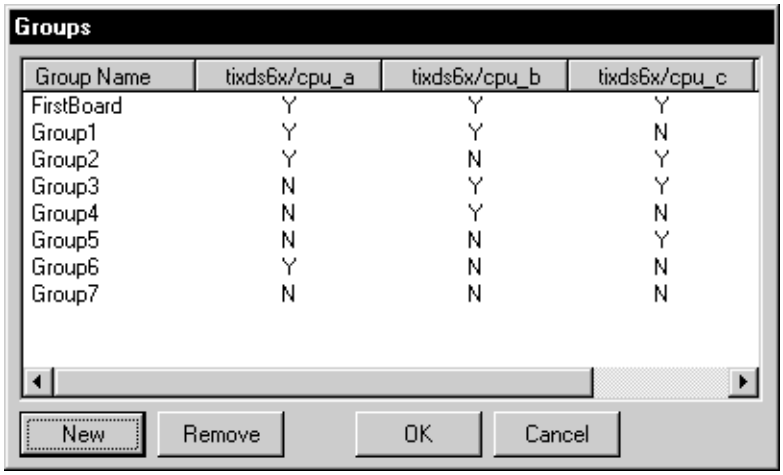
3.3 プロセッサのグループ化

パラレル・デバッグ・マネージャでのすべてのコマンドは、現在のグループ内のすべてのターゲット・プロセッサにブロードキャストされます。Code Composer Studio では、プロセッサのグループを最高 32 個定義できます。各プロセッサは、1 つまたは複数のグループに組み込むことができます。このグループのリストを表示するには、「Parallel Debug Manager」メニュー・バー上の 「Group」を選択します。「Group」は、すべてのグループを名前順にリストします。使用したいグループを選択するとアクティブ・グループになり、次の図のように、名前の隣にチェック・マークが付きます。



グループを編集する方法

「Parallel Debug Manager」メニューで 「Group」→「Edit」の順に選択します。「Groups」ダイアログ・ボックスが表示されます。



この 「Groups」ダイアログ・ボックスには、最初の列にグループ名が表示され、最上部にプロセッサ名が表示されます。この表内の各エントリには、Y (Yes) か N (no) のどちらかが表示されます。このエントリが Y である場合、その列のプロセッサは同じ行にあるグループに含まれます。現在のグループに含まれているプロセッサだけが、ブロードキャスト・メッセージを受信します。

グループ名は直接編集できますが、固有の名前でなければなりません。デフォルトでは、名前が FirstBoard のグループが作成され、そのグループにはすべてのプロセッサが含まれています。このデフォルト・グループは変更できますが、削除することはできません。つまり、少なくとも 1 つのグループが常に使用可能な状態にあります。

新規グループを作成する方法

「Groups」ダイアログ・ボックスで、「New」をクリックします。Code Composer Studio は、固有の名前を付けた新規グループを生成します。生成された直後は、新規グループにプロセッサは含まれていません。

グループにプロセッサを組み込む方法

1) 組み込むプロセッサの列を見つけます。
2) プロセッサを組み込むグループの行を見つけます。
3) 列と行が交差するセルをクリックします。「Y」が表示されたら、そのプロセッサはそのグループに組み込まれます。

グループを削除する方法

1) マウスまたは矢印キーを使用して、削除するグループ名を強調表示します。
2) 「Remove」を選択します。

3.4 マルチプロセッサ・ブロードキャスト・コマンド

パラレル・デバッグ・マネージャでのすべてのコマンドは、現在のグループ内のすべてのターゲット・プロセッサにブロードキャストされます（3.3 節「プロセッサのグループ化」を参照）。DSP デバイス・ドライバが同期操作をサポートする場合、次の各コマンドが同期して、各プロセッサ上で同時に開始します。

Locked Step

このコマンドを使用すると、実行中ではないすべてのプロセッサのシングル・ステップ実行ができます。

Locked Step-Over

このコマンドを使用すると、実行中ではないすべてのプロセッサ上で step-over を実行することができます。

Locked Step-Out

すべてのプロセッサが 1 つのサブルーチン内にある場合に、このコマンドを使用すると、実行中ではないすべてのプロセッサ上で Step-Out コマンドを実行することができます。

Synchronous Run

このコマンドは、実行中ではないすべてのプロセッサに、グローバル Run コマンドを送信します。

Synchronous Halt

このコマンドは、すべてのプロセッサを同時に一時停止します。

Synchronous Animation

このコマンドは、実行中ではないすべてのプロセッサのアニメーション表示を開始させます。詳細は、2.7 節 「Run、Halt、Animate、Run Free」を参照してください。

3.5 GEL コマンドのブロードキャスト

GEL コマンドをブロードキャストする方法

1) 「Parallel Debug Manager」メニューで 「Group」→「Broadcast Command」の順に選択します。「Broadcast Command」ダイアログ・ボックスが開きます。
2) 組み込み GEL 関数、またはユーザ定義の GEL 関数を、「Command」ボックスに入力します（12.5 節 「GEL 関数のロードとアンロード」を参照）。
3) 「OK」をクリックするか、Enter キーを押して、現在のグループ内の各プロセッサにこのコマンドをブロードキャストします。

注： GEL コマンドのブロードキャスト

GEL コマンドのブロードキャストは、親ウィンドウも開いているプロセッサに限定されます。

3.6 GEL 関数の自動実行

GEL 関数を使用すると、ユーザのニーズに合わせて開発環境を構成できるようになります。たとえば、特定の CPU に対する制御ウィンドウを、ウィンドウを開くたびにメモリ・システムのウェイト状態を初期化するように構成することができます。または、制御ウィンドウを開くたびに、多くのタスクを実行するようにセットアップすることもできます。

毎回、「File」→「Load GEL」コマンドを使用して GEL ファイルをロードした後に GEL 関数を実行する代わりに、「Open」→「CPU」コマンドを使用すると、各制御ウィンドウを開くたびにそのウィンドウに GEL ファイル名を渡すことができます。このコマンドは、指定された GEL ファイルをスキャンし、そのファイルをロードすることを制御ウィンドウに通知します。ユーザは、自分でこの機能を実行することもできます。この機能を実現するには、指定されたファイルの中にある自分の GEL 関数の 1 つに StartUp() という名前を付けます。Code Composer Studio は、GEL ファイルをロードすると、StartUp() という名前の関数を検索します。この関数は、検出されると自動的に実行されます。

制御ウィンドウのオープン時に GEL ファイルを自動的にロードし、実行する方法

1) 「Parallel Debug Manager」メニューで、「Options」→「Startup」の順に選択して、「StartUp Files」ダイアログ・ボックスを開きます。このダイアログ・ボックスは、すべての CPU をリストアップし、CPU ごとに GEL ファイルを指定できるフィールドを表示します。
2) このフィールドで、GEL 関数が入っている GEL ファイルの名前を指定します。たとえば、cpu_a myfile.gel です。

これによって、cpu_a 用の制御ウィンドウをユーザが開くたびに、GEL ファイル myfile.gel が自動的にスキャンされ、Code Composer Studio にロードされます。StartUp() 関数が定義されている場合には、その GEL 関数も実行されます。

始動時にすべての CPU がロードする共通ファイルがある場合には、Code Composer Studio アイコンの「プロパティ」ダイアログ・ボックスの「リンク先 (T)」フィールドの後ろに、このファイル名を入力すれば便利です。

始動時にワークスペースと関連 GEL ファイルを自動的にロードする方法

Code Composer Studio では、次のようにして、始動直後にワークスペースと関連 GEL ファイルの両方をロードすることができます。

1) 「Parallel Debug Manager」メニューで、「Options」→「Startup」の順に選択します。
2) 「StartUp Files」ダイアログ・ボックスで、各 CPU に対応する GEL ファイル名を入力し、「OK」をクリックします。
3) 「File」→「Save Workspace」の順に選択して、そのセットアップをワークスペースとしてセーブします。
4) 「Save Workspace」ダイアログ・ボックスで、「ファイル名 (N)」フィールドにそのワークスペースの名前（拡張子 .wks 付き）を入力します（2.15.1 節 「ワークスペースを自動的にロードする方法」を参照）。
5) 「保存 (S)」をクリックします。

3.7 グローバル・ブレークポイント

グローバル・ブレークポイントを使用すると、特定のプロセッサ上のブレークポイントで、マルチプロセッサ環境内の他のプロセッサを一時停止させることができます。JTAG ベースのデバイス・ドライバは、EMU0/1 ピンを使用して、他のプロセッサをトリガして同時に停止させます。このオプションをイネーブルにすると、現在のグループに含まれているいずれかのプロセッサがブレークポイントを検出すると、現在のグループ内のすべてのプロセッサが一時停止します。

現在のグループ内のすべてのプロセッサ上でグローバル・ブレークポイントをイネーブルにするには、「Parallel Debug Manager」メニュー・バーから 「Options」→「Global Breakpoints」の順に選択します。このメニュー項目の横にチェック・マークが付いている場合、グローバル・ブレークポイントがイネーブルになっていることを示します。現在のグループを変更することにより（3.3 節 「プロセッサのグループ化」を参照）、グローバル・ブレークポイントによってどのプロセッサがトリガーされるかを制御することができます。

第 4 章

ブレークポイントとプローブ・ポイント

本章では、ブレークポイントを設定してプログラムの実行を制御する方法、およびプローブ・ポイントを設定して信号を分析する方法について説明します。

項目 **ページ**

4.1 ブレークポイント

ブレークポイントは、プログラムの実行を停止させます。プログラムが停止すると、ユーザはプログラムの状態の検査、変数の検査または変更、コール・スタックの検査などを行うことができます。ブレークポイントを設定するには、「Project」ツールバー上の「Toggle Breakpoint」ボタンを使用するか、「Debug」→「Breakpoints」の順に選択します。「Debug」→「Breakpoints」を選択すると、「Break/Probe/Profile Points」ダイアログ・ボックスが立ち上がります。ブレークポイントが設定されていると、ユーザはそのブレークポイントをイネーブルにしたり、ディセーブルにしたりできます。

4.1.1 設計者用の注意事項（カーネル・ベースの Code Composer Studio デバッガ）

プロセッサ・パイプラインの破壊をできるだけ避けるために、次のガイドラインに従ってください。

- ❑ 遅延分岐または遅延呼び出しの一部として実行される命令に、ブレークポイントを設定しないでください。
- ❑ ブロック・リピート命令の終了直前の 1 つまたは 2 つの命令に、ブレークポイントを設定しないでください。

4.1.2 ブレークポイントの追加と削除

「Breakpoint」ダイアログ・ボックスを使用してブレークポイントを追加する方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) 「Breakpoint Type」フィールドで、「Break at Location（無条件）」、または「Break at Location if Expression is TRUE（条件付き）」のどちらかを選択します。
3) 次のフォーマットのどちらかを使用して、ブレークポイントを設定する位置を入力します。
   - ■ 絶対アドレスの場合、任意の有効な C 式、C 関数の名前、またはシンボル名を入力します。
   - ■ C ソース・ファイルに基づいて、ブレークポイントの位置を入力します。この方法は、C 命令が実行可能プログラムのどこに対応するかが分からない場合に便利です。C ソース・ファイルに基づいて位置を入力する場合のフォーマットは、次のとおりです。

     fileName line *lineNumber*
4) ステップ 2 で条件付きブレークポイントを選択した場合は、「Expression」フィールドに条件を入力する必要があります。
5) 「Add」をクリックして、新規ブレークポイントを作成します。これで、新しいブレークポイントが作成され、イネーブルになります。
6) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

ツールバーを使用してブレークポイントを追加する方法

プログラム内の任意の位置でブレークポイントの設定や消去を行う最も簡単な方法は、「Project」ツールバー上の 「Toggle Breakpoint」ボタンを使用することです。「Breakpoint」ダイアログ・ボックスを使用すると、条件付きブレークポイントやハードウェア・ブレークポイントのような、さらに複雑なブレークポイントが設定できます。

1) ブレークポイントを設定する行にカーソルを置きます。「Dis-Assembly」ウィンドウ、または C ソース・コードが入っている「Edit」ウィンドウのどちらかで、ブレークポイントを設定できます。
2) 「Project」ツールバー上の 「Toggle Breakpoint」ボタンをクリックします。その行が強調表示されます。

Toggle Breakpoint: 

ツールバーを使用してブレークポイントを削除する方法

1) ブレークポイントが設定されている行にカーソルを置きます。
2) 「Project」ツールバー上の 「Toggle Breakpoint」ボタンをクリックします。

既存のブレークポイントを変更する方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) 「Breakpoint」ウィンドウ内でブレークポイントを選択します。選択されたブレークポイントは強調表示され、「Breakpoint Type」、「Location」、および「Expression」の各フィールドが、選択されたブレークポイントと一致するように更新されます。
3) 必要に応じて、「Breakpoint Type」、「Location」、および「Expression」の各フィールドを編集します。
4) 「Replace」をクリックして、現在選択されているブレークポイントを変更します。
5) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

既存のブレークポイントを削除する方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) 「Breakpoint」ウィンドウ内でブレークポイントを選択します。
3) 「Delete」をクリックして、そのブレークポイントを削除します。
4) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

「Breakpoint」ダイアログ・ボックスを使用して、すべてのブレークポイントを削除する方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) 「Delete All」をクリックします。
3) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

ツールバーを使用して、すべてのブレークポイントを削除する方法

「Project」ツールバー上の 「Remove All Breakpoints」ボタンをクリックします。

Remove All Breakpoints: 

4.1.3 ブレークポイントのイネーブルとディセーブル

ブレークポイントが設定されていると、そのブレークポイントをディセーブルにしたり、イネーブルにしたりできます。ブレークポイントをディセーブルにすると、そのブレークポイントの位置とタイプを保持しながら、その動作を直ちに抑止させることができます。

ブレークポイントをイネーブルにする方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) イネーブルにするブレークポイントを 「Breakpoint」ウィンドウから選択します。ブレークポイントの横にあるチェック・ボックスにチェック・マークが付いていない場合は、そのブレークポイントが現在ディセーブルであることを示します。
3) そのブレークポイントのチェック・ボックスをクリックします。これで、そのチェック・ボックスにチェック・マークが付き、そのブレークポイントがイネーブルになったことを示します。
4) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

ブレークポイントをディセーブルにする方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) ディセーブルにするブレークポイントをリストから選択します。このブレークポイントのチェック・ボックスにはチェック・マークが付き、現在イネーブルであることを示します。
3) そのブレークポイントのチェック・ボックスをクリックします。これで、そのチェック・ボックスからチェック・マークが削除され、そのブレークポイントがディセーブルになったことを示します。
4) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

すべてのブレークポイントをイネーブルにする方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) 「Enable All」をクリックします。このボタンを使用すると、ブレークポイント・リスト内のすべてのブレークポイントを直ちにイネブールにできます。
3) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

すべてのブレークポイントをディセーブルにする方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) 「Disable All」をクリックします。このボタンを使用すると、ブレークポイント・リスト内のすべてのブレークポイントを直ちにディセーブルにできます。
3) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

4.2 条件付きブレークポイント

すべての条件付きブレークポイントには、各々に独自の条件式を設定できます。条件付きブレークポイントの位置に達すると、その式が計算されます。式の結果が偽の場合、プロセッサは、表示を更新せずに実行を再開します。式の結果が真の場合、プロセッサは、通常のブレークポイントが検出された場合と同じように表示を更新し、実行を停止します。

また、GEL (General Extension Language) ファイルを定義して、ブレークポイントをイネーブルにするのに必要な条件を満たすこともできます。

GEL ファイル名を入力する方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) 「Breakpoint type」ドロップダウン・リストから、「Break at Location if expression is TRUE」を選択します。これを選択したときだけ、「Expression」フィールドはアクティブになります。
3) 「Expression」フィールドに GEL ファイル名を入力します。こうすると、ユーザが指定する条件が満たされたときにだけ、ブレークポイントはイネーブルになります。

GEL ファイルとその設定の詳細は、第 12 章「General Extension Language (GEL)」を参照してください。

注 : ターゲット・プロセッサの一時停止

上記の式をホストが計算している間、ターゲット・プロセッサは一時停止します。つまり、条件付きブレークポイントが設定されていると、ターゲット・アプリケーションは、リアルタイムの制約条件を満たすことができない場合があります。

4.3 ハードウェア・ブレークポイント

ハードウェア・ブレークポイントは、ターゲット・プログラムを変更しない点で、ソフトウェア・ブレークポイントとは異なります。ハードウェア・ブレークポイントは、ROM メモリ内にブレークポイントを設定したり、命令の獲得をしないで、メモリ・アクセスを中断したりする場合に便利です。ブレークポイントは、個々のメモリの読み取り、メモリの書き込み、またはメモリの読み書きに対して設定できます。メモリ・アクセスのブレークポイントは、「Edit」ウィンドウにも「Memory」ウィンドウにも表示されません。

注 : シミュレータは、ハードウェア・ブレークポイントをサポートしません
ハードウェア・ブレークポイントは、シミュレーション DSP ターゲット上には設定できません。

ハードウェア・ブレークポイントには、カウントを指定することもできます。カウントを使用すると、特定の位置に指定した回数だけ到達したときにブレークポイントを生成するように設定することができます。このカウントが 1 の場合、ブレークポイントは毎回生成されます。

ハードウェア・ブレークポイントを追加する方法

1) 「Debug」→「Breakpoints」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Breakpoints」タブが選択された状態で表示されます。
2) 「Breakpoint type」フィールドで、命令獲得ブレークポイントの場合は、「*H/W Break*」を選択し、メモリ・アクセス・ブレークポイントの場合は、「*Break on<bus> <Read|Write|R/W>*」を選択します。
3) ブレークポイントを設定するプログラム位置またはメモリ位置を入力します。次の方法のどちらかを使用してください。
   - 絶対アドレスの場合、任意の有効な C 式、C 関数の名前、またはシンボル名を入力します。
   - C ソース・ファイルに基づいて、ブレークポイントの位置を入力します。この方法は、C 命令が実行可能プログラム内のどこに対応するかが分からないときに便利です。C ソース・ファイルに基づいて位置を入力する場合のフォーマットは、次のとおりです。

     fileName line *lineNumber*
4) ブレークポイントの生成までに繰り返されるその位置の検出回数を、「Count」フィールドに入力します。毎回ブレークさせたい場合は、このカウントを 1 に設定します。
5) 「Add」をクリックして、新規ブレークポイントを作成します。これで、新しいブレークポイントが作成され、イネーブルになります。
6) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

4.4 プローブ・ポイント

プローブ・ポイントを使用すると、特定のウィンドウを更新させたり、アルゴリズム内の特定ポイントでファイルからサンプルを読み出したり、ファイルにサンプルを書き込んだりできるようになります。プローブ・ポイントは、ご使用のアルゴリズム内の特定ポイントにプローブを接続します。プローブ・ポイントが設定されていると、ユーザは、ブレークポイントの場合と同様に、プローブ・ポイントをイネーブルにしたり、ディセーブルにしたりできます。

デフォルトでは、ウィンドウが作成されると、どのブレークポイントでもそのウィンドウは更新されます。しかし、接続されているプローブ・ポイントにプログラムが達したときにだけウィンドウが更新されるように、変更することができます。ウィンドウが更新されると、プログラムの実行は続行されます。

Code Composer Studio のファイル I/O 機能により、プローブ・ポイントを使用すると、DSP コード内の特定ポイントにデータのストリームを接続できます。アルゴリズム内でそのプローブ・ポイントに達すると、特定のメモリ・エリアからファイルに、またはファイルからメモリに、データが転送されます。詳細は、5.1 節「ファイル入出力 (I/O)」を参照してください。

注 : ターゲット・プロセッサはプローブ・ポイントで一時停止します

プローブ・ポイントが検出されると、ターゲット・プロセッサはホスト・プロセッサによって一時的に停止されます。したがって、プローブ・ポイントを使用する場合、ターゲット・アプリケーションは、リアルタイムの制約条件を満たすことができない場合があります。

4.4.1 プローブ・ポイントの追加と削除

プローブ・ポイントを追加する方法

ソース・ファイルまたは「Dis-Assembly」ウィンドウ内の行にカーソルを置いて、「Project」ツールバー上の 「Toggle Probe Point」ボタンをクリックすると、プローブ・ポイントを設定することができます。プローブ・ポイントは、ウィンドウまたはファイルに接続されていなければなりません（4.4.2 節 「プローブ・ポイントの接続」を参照）。「Debug」→「Probe Points」の順に選択すると、条件付きプローブ・ポイントやハードウェア・プローブ・ポイントなどの、さらに複雑なプローブ・ポイントを設定できるようになります。

Toggle Probe Point: 

既存のプローブ・ポイントを削除する方法

1) 「Debug」→「Probe Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。
2) 「Probe Point」ウィンドウ内で、削除するプローブ・ポイントを選択します。
3) 「Delete」をクリックします。
4) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

「Probe Point 」ダイアログ・ボックスを使用して、すべてのプローブ・ポイントを削除する方法

1) 「Debug」→「Probe Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。
2) 「Delete All」をクリックします。
3) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

ツールバーを使用して、すべてのプローブ・ポイントを削除する方法

「Project」ツールバー上の 「Remove All Probe Points」ボタンをクリックします。

Remove All Probe Points: 

4.4.2 プローブ・ポイントの接続

表示ウィンドウとプローブ・ポイントを接続する方法

1) 接続するウィンドウを開きます。
2) プローブ・ポイントを設定する行にカーソルを置き、 「Project」 ツールバー上の「Toggle Probe Point」ボタンをクリックして、プローブ・ポイントを設定します。

   Toggle Probe Point: 
3) 「Debug」→「Probe Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。「Probe Point」ウィンドウに、作成された新規プローブ・ポイントが表示されます。次のプローブ・ポイントは、現在接続されていないことを示しています。

   ```
   echocan.c line 191 --> No Connection
   ```

   このプローブ・ポイントを選択して、現在のプローブ・ポイントにします。これで、ダイアログ・ボックス内で、そのプローブ・ポイントのフィールドを編集できるようになります。
4) 「Probe type」ドロップダウン・リストから、プローブ・ポイントのタイプを選択します。デフォルトでは、無条件です。つまり、コードの実行中にそのプローブ・ポイントに達するたびに、このプローブ・ポイントに接続されているファイルまたはウィンドウが更新され、更新後にそのコードが引き続き実行されます。式が真の場合にだけプローブをアクティブにするように、そのプローブ・ポイントを条件付きプローブ・ポイントに変更することができます。

5) 次の方法のどちらかを使用して、そのプローブ・ポイントを設定する位置を指定します。「Toggle Probe Point」ボタンを使用した場合、このフィールドには、すでに該当する値が埋め込まれています。

   ■ 絶対アドレスの場合、任意の有効な C 式、C 関数の名前、またはアセンブリ言語ラベルの名前を指定します。

   ■ C ソース・ファイルに基づいて、プローブ・ポイントの位置を指定します。この方法は、C 命令が実行可能プログラム内のどこにあるかが分からない場合に便利です。C ソース・ファイルに基づいて位置を指定する場合のフォーマットは、次のとおりです。

   fileName line *lineNumber*

6) ステップ 4 で条件付きプローブ・ポイントを選択した場合には、「Expression」フィールドに条件を入力する必要があります。

7) ウィンドウまたはファイルを、そのプローブ・ポイントに接続します。「Connect To」ドロップダウン・リストには、そのプローブ・ポイントに接続可能なすべてのファイルとウィンドウが表示されます。このリストの中から、該当する項目を選択してください。

8) 「Add」をクリックして新規プローブ・ポイントを作成するか、「Replace」をクリックして既存のプローブ・ポイントを変更します。

4.4.3 プローブ・ポイントをイネーブルまたはディセーブルにする方法

プローブ・ポイントを設定すると、そのプローブ・ポイントをイネーブルにしたり、ディセーブルにしたりできます。プローブ・ポイントをディセーブルにすると、そのプローブ・ポイントの位置とタイプを保持しながら、その動作を直ちに抑止させることができます。

注 : ウィンドウが更新されない場合

ウィンドウに接続されているプローブ・ポイントがディセーブルに設定されている場合、そのウィンドウは更新されません。

プローブ・ポイントをイネーブルにする方法

1) 「Debug」→「Probe Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。
2) イネーブルにするプローブ・ポイントを 「Probe Point」ウィンドウから選択します。プローブ・ポイントがディセーブルのときは、そのプローブ・ポイントのチェック・ボックスにチェック・マークが付きません。
3) プローブ・ポイントのチェック・ボックスをクリックします。これで、そのチェック・ボックスにチェック・マークが付き、そのプローブ・ポイントがイネーブルになったことを示します。
4) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

プローブ・ポイントをディセーブルにする方法

1) 「Debug」→「Probe Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。
2) ディセーブルにするプローブ・ポイントを「Probe Point」ウィンドウから選択します。プローブ・ポイントがイネーブルのときは、そのプローブ・ポイントのチェック・ボックスにチェック・マークが付きます。
3) プローブ・ポイントのチェック・ボックスをクリックします。これで、そのチェック・ボックスからチェック・マークが削除され、そのプローブ・ポイントがディセーブルになったことを示します。
4) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

すべてのプローブ・ポイントをイネーブルにする方法

1) 「Debug」→「Probe Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。
2) 「Enable All」をクリックします。これで、すべてのチェック・ボックスにチェック・マークが付きます。
3) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

すべてのプローブ・ポイントをディセーブルにする方法

1) 「Debug」→「Probe Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。
2) 「Disable All」をクリックします。すべてのチェック・ボックスのチェック・マークがオフになります。
3) 「OK」をクリックして、そのダイアログ・ボックスをクローズします。

4.5 条件付きプローブ・ポイント

すべての条件付きプローブ・ポイントには、各々に独自の条件式を設定できます。プロセッサは、条件付きプローブ・ポイントの位置に達すると、その式を計算します。この式の結果が偽の場合、プロセッサは、プローブ・ポイントを検出しなかった場合と同じように実行を再開します。式が真の場合、標準のプローブ・ポイントが検出された場合と同じように、プロセッサは実行します（4.4 節 「プローブ・ポイント」を参照）。

また、ユーザが定義する GEL (General Extension Language) ファイルを接続して、特定のプローブ・ポイントをイネーブルにする条件を満たすこともできます。

GEL ファイル名を入力する方法

1) 「Debug」→「Probe Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。
2) 「Probe type」ドロップダウン・リストから、「Probe at Location if expression is TRUE」を選択します。これを選択したときにだけ、「Expression」フィールドはアクティブになります。
3) 「Expression」フィールドに GEL ファイル名を入力します。GEL ファイル名を入力すると、ユーザが指定する条件が満たされたときにだけ、プローブ・ポイントがイネーブルになります。

GEL ファイルとその設定方法の詳細は、第 12 章「General Extension Language (GEL)」を参照してください。

4.6 ハードウェア・プローブ・ポイント

ハードウェア・プローブ・ポイントは、通常のプローブ・ポイントと同じように動作します。ただし、ハードウェア・プローブ・ポイントは、ソフトウェア・ブレークポイントではなく、ハードウェア・ブレークポイントを使用して設定されます（4.3 節「ハードウェア・ブレークポイント」を参照）。ハードウェア・プローブ・ポイントは、ROM メモリ内のプローブの設定またはメモリ・アクセスのトレースを行う場合に便利です。

注 : ターゲット・プロセッサはプローブ・ポイントで一時停止します

ホスト・プロセッサがハードウェア・プローブ・ポイントを検出すると、ターゲット・プロセッサは、ホスト・プロセッサによって一時的に停止されます。したがって、ハードウェア・プローブ・ポイントを使用すると、ターゲット・アプリケーションは、リアルタイムの制約条件を満たすことができない場合があります。

注 : シミュレータはハードウェア・プローブ・ポイントをサポートしません

ハードウェア・ブレークポイント（およびプローブ・ポイント）は、シミュレーション DSP ターゲット上には設定できません。

メモリ・アクセスをトレースする方法

以下の手順を行うと、トレースするメモリ位置でデータを保管するファイル・オブジェクトを作成することができます（5.1 節 「ファイル入出力 (I/O)」を参照）。

1) 「File」→「File I/O」の順に選択します。
2) 「File I/O」ダイアログ・ボックスで、「Add Probe Point」をクリックします。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。
3) 「Probe type」ドロップダウン・リストで、「probe on <bus> <Read|Write|R/W> at location」を選択します。
4) 「Location」フィールドに、トレースするメモリ位置を入力します。
5) 「Connect To」ドロップダウン・リストから、ファイル・オブジェクトを選択します。
6) 「Add」をクリックして、新規プローブ・ポイントを作成し、イネーブルにします。
7) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

第 5 章

ファイル入出力機能の使用方法

本章では、実際のターゲットまたはシミュレーション・ターゲットへファイルに含まれるデータを信号として転送する方法について説明します。また、ターゲット・メモリの値を PC ファイルに保存したり、逆にロードしたりする方法についても説明します。

項目 **ページ**

5.1 ファイル入出力 (I/O)

Code Composer Studio を使用すると、実際のターゲットまたはシミュレーション・ターゲットと PC ファイルとの間でデータを両方向に転送することができるようになります。これは、既存のサンプル値を使用してコードをシミュレートするのに優れた方法です。ファイル I/O 機能では、プローブ・ポイントの概念を使用します。この概念では、ユーザが定義するポイント（プローブ・ポイント）で、サンプルの取り出しまたは挿入を行うか、メモリ位置のスナップショットを取ることができます。プローブ・ポイントは、（ブレークポイントの設定とほぼ同じ方法で）アルゴリズム内の任意のポイントに設定できます。プログラムの実行中にプローブ・ポイントに達すると、そのプローブ・ポイントに接続されているオブジェクト（ファイル、グラフ、またはメモリ・ウィンドウのどれであるかに関係なく）が更新されます。接続されているオブジェクトが更新された後、引き続きプログラムが実行されます。この概念を使用すると、コード内の特定のポイントにプローブ・ポイントを設定して、そのプローブ・ポイントにファイルを接続すると、ファイル I/O 機能を設定することができます。

入力信号、または出力信号のどちらかに、ファイルを関連付けることができます。特定のプローブ・ポイントで、指定されたファイルからデータのストリームを読み取ったり、指定されたファイルにデータのストリームを書き込んだりできます。

注 : リアルタイムのデータ転送

ファイル I/O **では、リアルタイムのデータ転送はサポートされません。**Real-Time Data Exchange (RTDX) **の使用方法についての情報は、「**Help**」→「**Tools**」→「**RTDX**」の順に選択して入手してください。**

ファイルとの間で、両方向にデータの転送を行う方法

1) ファイルについての情報を指定する前に、プローブ・ポイントを設定する場所にカーソルを置いて、プローブ・ポイントを設定します。「Project」ツールバー上の「Toggle Probe Point」ボタンをクリックします。このプローブ・ポイントは、接続されていないままにしてください。このプローブ・ポイントは、ファイルとの間で両方向のデータの転送を開始する時点を、Code Composer Studio デバッガに指示します。つまり、コードの実行中にこのポイントに達すると、Code Composer Studio デバッガは、そのプローブ・ポイントに接続されているファイルを更新します（または、そのファイルから読み取ります）。ファイルの更新が終了すると、引き続きプログラムが実行されます。

   Toggle Probe Point: 

2) 「File」→「File I/O」の順に選択します。File I/O ダイアログ・ボックスが表示されます。「File I/O」ダイアログでは、特定の情報を入力するように求められます。「File Input 」タブか 「File Output」タブのどちらかを選択してください。

3) 「File Input」タブか 「File Output」タブのどちらかの下にある 「Add File」をクリックします。「File Input」ダイアログ・ボックスが表示されます。

   - ❏ 使用するファイルが入っているフォルダまでナビゲートします。
   - ❏ そのダイアログのメイン・ウィンドウで、そのファイル名を強調表示します。その名前が「ファイル名 (N)」フィールドに表示されます。データ・ファイルは、COFF オブジェクト・ファイルか、Code Composer Studio データ・ファイルのどちらでも構いません（5.1.2 節 「データ・ファイル・フォーマット」を参照）。
   - ❏ 「開く (O)」をクリックします。「File I/O 」ダイアログ・ボックスにファイル名が表示されます。「File Input」または「File Output」用の追加ファイルを選択するには、この手順を繰り返します。

4) 「File I/O」ダイアログ・ボックスにファイルを挿入しても、そのファイルはプローブ・ポイントに接続されません。「Probe」フィールドに 「Not Connected」という語が表示されることに注意してください。

   ファイルをプローブ・ポイントに接続するために、「Add Probe Point」をクリックします。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Probe Points」タブが選択された状態で表示されます。

   - ❏ 「Probe Point」リストで、接続するプローブ・ポイントを強調表示します。そのプローブ・ポイントが 「No Connection」であることに注意してください。
   - ❏ 「Connect To」ドロップダウン・リストから、適切なファイルを選択します。
   - ❏ 「Replace」をクリックします。この時点で、「Probe Point」リストが、選択したファイルにそのプローブ・ポイントが接続されたことを表示していることに注目してください。
   - ❏ 「OK」 をクリックします。ファイルが正常にプローブ・ポイントに接続されると、「File I/O」ダイアログ・ボックス内の 「Probe」フィールドに、「Connected」という語が表示されます。

5) 「File I/O」ダイアログ・ボックスで、選択されたファイルごとに、「Address」フィールドと「Length」フィールドに値を入力します。

   「Address」フィールドでは、データの転送元 (File Input) 、または転送先 (File Output) を指定します。このフィールドには、有効なラベルまたは数値アドレスのどちらかを入力できます。

   「Length」フィールドでは、選択されたプローブ・ポイントに達するたびに、ターゲット・ボードに転送される (File Input) サンプル数、またはターゲット・ボードから転送される (File Output) サンプル数を指定します。

   「Address」フィールドと 「Length」フィールドには、任意の有効な C 式を入力することができます。これらの式は、サンプルがターゲットから読み取られるか、ターゲットに書き込まれるたびに計算し直されます。つまり、このフィールドにシンボルを入力し、その値が後で変更されても、このパラメータを入力し直す必要はありません。

6) 「OK」をクリックします。入力されたパラメータが検証されます。

折り返しモード

「File Input」タブで、「Wrap Around」チェック・ボックスを選択することができます。折り返しモードを使用すると、ファイルの内容を繰り返し使うように、つまりファイルの終わりに達すると、再び先頭からアクセスするように設定できます。この機能は、ファイルから周期的な信号を生成したい場合に便利です。折り返しモードが選択されていないときに、ファイルの終わりに達すると、ファイルの終わり状態を示すメッセージと一緒にプロンプトが出され、プログラムが一時停止します。

5.1.1 ファイル I/O の制御

「File I/O」ダイアログ・ボックスにデータを入力し、「OK」をクリックすると、制御ウィンドウが表示されます。このウィンドウでは、ファイル I/O 動作の進行状況をモニタし、制御することができます。

制御ウィンドウの特徴は、次のとおりです。

- **Play** ボタン

  一時停止後に、ファイル I/O トランザクションを再開します。

- **Stop** ボタン

  プローブ・ポイントが検出されたかどうかにかかわらず、ファイルとの間の両方向のデータ転送を、すべて一時停止します。このボタンを使用すると、ファイル I/O 転送を一時的に停止できます。

- **Rewind to Beginning** ボタン

  ファイルをリセットします。ファイル入力の場合、次のサンプルはファイルの先頭から読み取られます。ファイル出力の場合、すべての既存サンプルが削除され、新規サンプルがファイルの先頭に書き込まれます。

- **Fast Forward** ボタン

  プローブ・ポイントの検出をシミュレートします。このボタンをクリックすると、ターゲットがプローブ・ポイントを検出したときと同じファイル I/O が行われます。

- **File I/O** プログレス・フィールド

  ファイル・トランザクションの進行状況を表示します。ファイル入力の場合、ファイルから読み取られたサンプルと、ターゲットに書き込まれたサンプルのパーセンテージが、プログレス・バーに表示されます。ファイル出力の場合、ファイルに現在書き込まれているサンプルの数を数字で示します。

5.1.2 データ・ファイル・フォーマット

コマンド「File」→「Data」→「Load」、「File」→「Data」→「Save」、および「File」→「File I/O」は、すべてファイル・フォーマット（COFF および Code Composer Studio データ・ファイル）を使用します。

COFF

共通オブジェクト・ファイル・フォーマット (COFF) を使用するバイナリ・ファイル。COFF は、PC で作成された大きなデータ・ブロックを保存する最も簡潔な方法です。

Code Composer Studio データ・ファイル

1 行のヘッダ情報を使用し、1 行当たり 1 つのサンプル・データをストアするテキスト・ファイル。このデータは、次のいずれかのフォーマットにできます。

- Hexadecimal
- Integer
- Long
- Float

データ・ファイル用のヘッダ情報は、次の構文を使用します。ここで、イタリック体の項目は変数です。

MagicNumber Format Starting Address PageNum Length

MagicNumber	1651 に固定されています。
Format	1 ～ 4 の数字。ファイル内のサンプルのフォーマットを示します。この数字は、データ・フォーマット（hexadecimal、integer、long、または float）を表します。
StartingAddress	セーブされているブロックの開始アドレス
PageNum	取り出されるブロックが入っているページの番号
Length	ブロック内のサンプル数

すべてのヘッダ値は、TI スタイルの 16 進値であると仮定されます。

Code Composer Studio データ・ファイルの例は、次のとおりです。

```
1651 1 800 1 10
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
0x0000
```

注 : ヘッダ情報の上書き

ヘッダ値は、デフォルトのアドレスと長さだけを指定します。「File**」→「**Data**」→「**Load**」コマンドを使用してファイルをメモリにロードすると、**Code Composer Studio **デバッガは、これらの値を上書きするかどうかをユーザに聞いてきます。ファイル** I/O **で** Code Composer Studio **データ・ファイル・フォーマットを使用する場合、「**File I/O**」ダイアログ・ボックス（**Address **と** Length**）に入力する情報は、すべて** Code Composer Studio **データ・ファイルのヘッダ情報を自動的に上書きします。ヘッダに次の値が含まれている場合は、ファイルごとにヘッダ情報を設定する必要はありません。**1651 1 0 0 0.

5.2 データ・ファイルのロード

データ・ファイルは、ターゲット・ボードの任意の有効なアドレスにロードすることができます。このデータ・ファイルは COFF オブジェクト・ファイルか、Code Composer Studio データ・ファイルのどちらかであっても構いません。

データ・ファイルをロードする方法

1) 「File」→「Data」→「Load」の順に選択します。「Load Data」ダイアログ・ボックスが表示されます。
2) データ・ファイルがウィンドウ内に表示されていない場合、ロードするファイルまでナビゲートします。ファイル名を選択した後、「開く (O)」をクリックします。「Loading File into Memory」ダイアログ・ボックスが表示されます。
3) 「Loading File into Memory」ダイアログで、データをロードする先のアドレスと、そのデータの長さを指定します。
4) 「OK」をクリックします。

すべての入力フィールドは、C 式入力フィールドです。

5.3 データ・ファイルへの保存

ターゲット・ボードからのメモリ値は、データ・ファイルに保存することができます。このデータ・ファイルは COFF オブジェクト・ファイルか、Code Composer Studio データ・ファイルのどちらかであっても構いません。

データをファイルに保存する方法

1) 「File」→「Data」→「Save」の順に選択します。「Store Data」ダイアログ・ボックスが表示されます。
2) データ・ファイル名を指定した後、「保存 (S)」をクリックします。「Storing Memory into File」ダイアログ・ボックスが表示されます。
3) 「Storing Memory into File」ダイアログで、保存するデータの開始アドレスおよび長さを指定します。
4) 「OK」をクリックします。

すべての入力フィールドは、C 式入力フィールドです。

第 6 章

「Graph」ウィンドウ

Code Composer Studio には、先進的な信号分析インターフェイスが組み込まれています。このインターフェイスを使用すると、開発者は、厳格かつ完全に信号データをモニタすることができます。この新しい機能は、汎用的な DSP のアプリケーションはもちろん、通信、無線、イメージ処理用のアプリケーションを開発する際に便利です。

本章では、Code Composer Studio のグラフ作成機能を使用して、実際のターゲット・システム、またはシミュレーション・ターゲット・システム上で信号を表示する方法について説明します。

項目 **ページ**

6.1 Time/Frequency

グラフ・メニューには、多数のオプションが含まれています。これらのオプションを使用すると、データを柔軟に表示できるようになります。time/frequency グラフを使用すると、時間領域または周波数領域のどちらかに信号を表示できます。周波数領域を分析する場合、表示バッファが FFT ルーチンを通じて実行され、データの周波数領域の分析を行います。周波数グラフには、FFT Magnitude、FFT Waterfall、Complex FFT、および FFT Magnitude and Phase の種類があります。時間領域の分析では、グラフ化される前に、表示データに対して前処理は行われません。時間領域グラフは、Single Time、または Dual Time のどちらかで表示できます。

「Graph Property」ダイアログ・ボックス を表示するには、「View」→「Graph」→「Time/Frequency」の順に選択します。フィールド名が左の欄に表示されます。必要に応じて、右欄の値を調整できます。「OK」をクリックすると、設定されたプロパティを使用して、グラフ・ウィンドウが表示されます。このグラフ・ウィンドウでこれらのパラメータを変更するには、マウスを右クリックし、「Properties」を選択し、必要に応じてパラメータを調整します。また、プログラム内の任意のポイントでグラフを更新することもできます（詳細は、4.4.2 節 「プローブ・ポイントの接続」を参照）。

すべての入力フィールドは、C 式入力フィールドです。数値入力を必要とするすべてのフィールド（たとえば、「Start Address」や「Acquisition Buffer Size」) に、シンボルを含む式を使用することができます。詳細は、2.3.3.1 節 「式の中でシンボルを使用する方法」を参照してください。

6.1.1 Time/Frequency グラフの働き

「Graph」ウィンドウには、2 つのバッファが関連付けられています。獲得 (acquisition) バッファと表示 (display) バッファです。獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードに置かれています。獲得バッファには、ユーザの関心があるデータが入っています。グラフが更新されるときには、獲得バッファが実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから読み取られ、表示バッファが更新されます。表示バッファはホスト・メモリに置かれているので、データの履歴を保持します。このグラフは、表示バッファ内のデータから生成されます。

パラメータを入力し、「OK」を押すと、「Graph」ウィンドウは、「Acquisition Buffer Size」フィールドで指定された長さの DSP データを含む獲得バッファを受け取ります。この獲得バッファは、DSP データ・メモリ空間内の「Start Address」フィールドで指定された位置から開始します。「Display Data Size」に指定されたサイズの表示バッファは、ホスト・メモリ内で割り当てられ、その値は、すべて 0 に初期化されます。

「Left-Shifted Data Display」フィールドをイネーブルにすると、表示バッファ全体が、「Acquisition Buffer Size」フィールドで指定された値だけ左にシフトされ、獲得バッファの値が右端からシフトインされます。表示バッファの値は、獲得バッファの値によって上書きされます。

信号を逐次に処理する場合は、「left-shifted data display」が便利です。「left-shifted data display」を使用すると、サンプルは一度に 1 つしか利用できませんが、それらのサンプルの履歴を表示することができます。関連付けられているプローブ・ポイントに達すると（4.4 節 「プローブ・ポイント」を参照）、DSP データのブロックが読み取られ、表示が更新されます。

以下の節では、「Graph Property」ダイアログ・ボックス 内の入力フィールドについて説明します。

6.1.2 Display Type

「Graph Property」ダイアログ・ボックス 内の「Display Type」オプションには、右欄のドロップダウン・メニューに複数のオプションが含まれています。このフィールドのオプションは、constellation (6.2 節 「Constellation」)、eye (6.3 節 「Eye Diagram」)、または image グラフ (6.4 節 「Image Graph」) と対応しています。

次に説明するグラフ・オプションを選択すると、「Graph Property」ダイアログ・ボックスに、さらに別のフィールドが表示されます。

❑ **Single Time**

表示バッファ内のデータを、前処理なしで大きさ対時間のグラフにプロットします。信号の単一時間トレースが、そのグラフ上に表示されます。このオプションを選択すると、「Graph Property」ダイアログ・ボックス に次のフィールドが表示されます。

■ **Time Display Unit**

グラフの時間軸に計測単位を指定します。s (秒)、ms (ミリ秒)、us (マイクロ秒)、および sample（表示バッファ・インデックスによって時間軸上に値を表示する）の値の中から選択します。

■ **Start Address**

グラフ化されるデータが入っている（実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上の）獲得バッファの開始位置。このグラフが更新されるときには、この位置から開始する獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。次に、獲得バッファは表示バッファを更新し、この表示バッファがグラフ化されます。「Start Address」フィールドには、任意の有効な C 式を入力できます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。つまり、このフィールドにシンボルを入力し、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

■ **Index Increment**

データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。

❑ **Dual Time**

表示バッファ内のデータを、前処理なしで大きさ対時間のグラフにプロットします。信号の二重時間トレースがグラフ上に表示されます。このオプションを選択すると、1 つのグラフ・ウィンドウ内に 2 つの時間領域トレースを作図できます。このオプションを選択すると、「Graph Property」ダイアログ・ボックス に次の追加フィールドが表示されます。

■ **Time Display Unit**

グラフの時間軸の計測単位を指定します。s (秒)、ms (ミリ秒)、us (マイクロ秒)、および sample（表示バッファ・インデックスによって時間軸上に値を表示する）の値の中から選択します。

■ **Interleaved Data Sources**

信号ソースがインターリーブされるかどうかを指定します。この表示オプションを切り換えると、1 つのバッファ入力が 2 つのソースを表すことができるようになります。このオプションを「Yes」に設定すると、奇数サンプルが最初のソースを表し、偶数サンプルが 2 番目のソースを表すことになります。このオプションを「 Yes」に設定すると、ダイアログ・ボックス に次の追加フィールドが作成されます。

▮ **Start Address**

グラフ化されるデータが入っている（実際のターゲット・ボード、またはシミュレーション・ターゲット・ボード上の）獲得バッファの開始位置。このグラフが更新されるときには、この位置から開始する獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。次に、獲得バッファは表示バッファを更新します。そして、この表示バッファがグラフ化されます。「Start Address」フィールドには、任意の有効な C 式を入力することができます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。つまり、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

「Interleaved Data Sources」を「No」に設定すると、次の追加フィールドが作成されます。

▮ **Start Address - upper display**

▮ **Start Address - lower display**

▮ **Index Increment**

データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。

❑ **FFT Magnitude**

表示バッファ内のデータに FFT を実行し、大きさ対周波数のグラフをプロットします。この FFT ルーチンは、FFT フレーム・サイズ（最も近い 2 の累乗に切り上げ）を使用して、サンプルの最小数を判別します。このオプションを選択すると、「Graph Property」ダイアログ・ボックス に次の追加項目が表示されます。

■ **Frequency Display Unit**

グラフの周波数軸に計測単位を指定します。Hz (ヘルツ)、kHz (キロヘルツ)、MHz (メガヘルツ) の値の中から選択します。

■ **Signal Type**

グラフを作成するための信号ソースのタイプを指定します。このプロパティには Real と Complex の 2 つのオプションがあります。Real は、単一時間表示の場合のように単一ソース表示に対応し、Complex は、2 つの信号ソースに対応します。Complex を選択すると、ダイアログ・ボックス に次の追加オプションが表示されます。

■ **Interleaved Data Sources**

信号ソースがインターリーブされるかどうかを指定します。この表示オプションを切り換えると、1 つのバッファ入力が 2 つのソースを表すことができるようになります。このオプションを「Yes」に設定すると、奇数サンプルが最初のソースを表し、偶数サンプルが 2 番目のソースを表すことになります。このオプションを「Yes」に設定すると、ダイアログ・ボックス に次の追加フィールドが作成されます。

▮ **Start Address**

グラフ化されるデータが入っている（実際のターゲット・ボード、またはシミュレーション・ターゲット・ボード上の）獲得バッファの開始位置。このグラフが更新されるときには、この位置から開始する獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。次に、獲得バッファは表示バッファを更新します。そして、この表示バッファがグラフ化されます。「Start Address」フィールドには、任意の有効な C 式を入力することができます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。つまり、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

「Interleaved Data Sources」を「No」に設定すると、次の追加フィールドが作成されます。

- **Start Address - real data**
- **Start Address - imaginary data**
- **Index Increment**

  データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。

■ **FFT Framesize**

各 FFT 計算で使用されるサンプル数を指定します。

注 : 獲得バッファのサイズが異なる場合

獲得バッファのサイズは、FFT フレームのサイズとは異なる場合があります。

■ **FFT Order**

FFT サイズ = $2^{\text{FFT order}}$ となるように指定します。

■ **FFT Windowing Function**

Rectangle (矩形)、Bartlett、Blackman(ブラックマン)、Hanning (ハニング)、Hamming(ハミング) の窓関数の中から選択できます。これらの操作は、FFT の計算が実施される前に、データに対して実行されます。

■ **Display Peak and Hold**

サンプルの履歴がグラフ上でどのように保持されるかについての詳細な情報を入力できます。

❑ **Complex FFT**

同じグラフ表示ウィンドウに、実数部分と虚数部分からなる 2 つのグラフが表示されます。このオプションを選択すると、「Graph Property」ダイアログ・ボックス に次の追加項目が表示されます。

■ **Frequency Display Unit**

グラフの周波数軸に計測単位を指定します。Hz (ヘルツ)、kHz (キロヘルツ)、MHz (メガヘルツ) の値の中から選択します。

■ **Signal Type**

グラフを作成するための信号ソースのタイプを指定します。このプロパティには Real と Complex の 2 つのオプションがあります。Real は、単一時間表示の場合のように単一ソース表示に対応し、Complex は、2 つの信号ソースに対応します。Complex を選択すると、ダイアログ・ボックス に次の追加オプションが表示されます。

■ **Interleaved Data Sources**

信号ソースがインターリーブされるかどうかを指定します。「Signal Type」フィールドを Complex に設定すると、このフィールドが表示されます。この表示オプションを切り換えると、1 つのバッファ入力が 2 つのソースを表すことができるようになります。このオプションを「Yes」に設定すると、奇数サンプルが最初のソースを表し、偶数サンプルが 2 番目のソースを表すことになります。このオプションを「Yes」に設定すると、ダイアログ・ボックス に次の追加フィールドが作成されます。

▮ **Start Address**

グラフ化されるデータが入っている（実際のターゲット・ボード、またはシミュレーション・ターゲット・ボード上の）獲得バッファの開始位置。このグラフが更新されるときには、この位置から開始する獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。次に、獲得バッファは表示バッファを更新します。そして、この表示バッファがグラフ化されます。「Start Address」フィールドには、任意の有効な C 式を入力することができます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。つまり、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

「Interleaved Data Sources」を「No」に設定すると、次の追加フィールドが作成されます。

- **Start Address - real data**
- **Start Address - imaginary data**
- **Index Increment**

  データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。

■ **FFT Framesize**

各 FFT 計算で使用されるサンプル数を指定します。

注 : 獲得バッファのサイズが異なる場合
獲得バッファのサイズは、FFT フレームのサイズとは異なる場合があります。

■ **FFT Order** ― FFT サイズ =2 $^{\text{FFT order}}$ になるように指定します。

■ **FFT Windowing Function**

Rectangle (矩形)、Bartlett、Blackman(ブラックマン)、Hanning (ハニング)、Hamming(ハミング) の窓関数の中から選択できます。これらの操作は、FFT の計算が実施される前に、データに対して実行されます。

❑ **FFT Magnitude and Phase**

同じグラフ表示ウィンドウに大きさ部分と位相部分からなる 2 つのグラフが表示されます。「FFT Magnitude and Phase」オプションを選択すると、「Graph Property」ダイアログ・ボックス に次の追加オプションが表示されます。

■ **Frequency Display Unit**

グラフの周波数軸に計測単位を指定します。Hz (ヘルツ)、kHz (キロヘルツ)、MHz (メガヘルツ) の値の中から選択します。

■ **Signal Type**

グラフを作成するための信号ソースのタイプを指定します。このプロパティには Real と Complex の 2 つのオプションがあります。Real は、単一時間表示の場合のように単一ソース表示に対応し、Complex は、2 つの信号ソースに対応します。Complex を選択すると、ダイアログ・ボックス に次の追加オプションが表示されます。

■ **Interleaved Data Sources**

信号ソースがインターリーブされるかどうかを指定します。このフィールドは、「Signal Type」フィールドを Complex に設定すると表示されます。この表示オプションを切り換えると、1 つのバッファ入力が 2 つのソースを表すことができるようになります。このオプションを「Yes」に設定すると、奇数サンプルが最初のソースを表し、偶数サンプルが 2 番目のソースを表すことになります。このオプションを「Yes」に設定すると、ダイアログ・ボックス に次の追加フィールドが作成されます。

▮ **Start Address**

グラフ化されるデータが入っている（実際のターゲット・ボード、またはシミュレーション・ターゲット・ボード上の）獲得バッファの開始位置。このグラフが更新されるときには、この位置から開始する獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。次に、獲得バッファは表示バッファを更新します。そして、この表示バッファがグラフ化されます。「Start Address」フィールドには、任意の有効な C 式を入力することができます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。つまり、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

「Interleaved Data Sources」を「No」に設定すると、次の追加フィールドが作成されます。

- **Start Address - real data**
- **Start Address - imaginary data**
- **Index Increment**

  データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。

■ **FFT Framesize**

各 FFT 計算で使用されるサンプル数を指定します。

注 : 獲得バッファのサイズが異なる場合
獲得バッファのサイズは、FFT フレームのサイズとは異なる場合があります。

■ **FFT Order**

FFT サイズ = $2^{\text{FFT order}}$ となるように指定します。

■ **FFT Windowing Function**

Rectangle (矩形)、Bartlett、Blackman(ブラックマン)、Hanning (ハニング)、Hamming(ハミング) の窓関数の中から選択できます。これらの操作は、FFT の計算が実施される前に、データに対して実行されます。

❑ **FFT Waterfall**

表示バッファ内のデータに FFT を実行し、大きさ対周波数のグラフをフレームとしてプロットします。これらのフレームの発生順に、FFT ウォータフォール型グラフが作成されます。「FFT Waterfall」オプションを選択すると、「Graph Property」ダイアログ・ボックス に次の追加オプションが表示されます。

■ **Frequency Display Unit**

グラフの周波数軸に計測単位を指定します。Hz (ヘルツ)、kHz (キロヘルツ)、MHz (メガヘルツ) の値の中から選択します。

■ **Signal Type**

グラフを作成するための信号ソースのタイプを指定します。このプロパティには Real と Complex の 2 つのオプションがあります。Real は、単一時間表示の場合のように単一ソース表示に対応し、Complex は、2 つの信号ソースに対応します。Complex を選択すると、ダイアログ・ボックスに次の追加オプションが表示されます。

■ **Interleaved Data Sources**

信号ソースがインターリーブされるかどうかを指定します。「Signal Type」フィールドを Complex に設定すると、このフィールドが表示されます。この表示オプションを切り換えると、1 つのバッファ入力が 2 つのソースを表すことができるようになります。このオプションを「Yes」に設定すると、奇数サンプルが最初のソースを表し、偶数サンプルが 2 番目のソースを表すことになります。このオプションを「Yes」に設定すると、ダイアログ・ボックス に次の追加フィールドが作成されます。

▮ **Start Address**

グラフ化されるデータが入っている（実際のターゲット・ボード、またはシミュレーション・ターゲット・ボード上の）獲得バッファの開始位置。このグラフが更新されるときには、この位置から開始する獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。次に、獲得バッファは表示バッファを更新します。そして、この表示バッファがグラフ化されます。「Start Address」フィールドには、任意の有効な C 式を入力することができます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。つまり、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

「Interleaved Data Sources」を「No」に設定すると、次の追加フィールドが作成されます。

- **Start Address - real data**
- **Start Address - imaginary data**
- **Index Increment**

  データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。

■ **FFT Framesize**

各 FFT 計算で使用されるサンプル数を指定します。

注 : 獲得バッファのサイズが異なる場合

獲得バッファのサイズは、FFT フレームのサイズとは異なる場合があります。

■ **FFT Order**

FFT サイズ = $2^{\text{FFT order}}$ となるように指定します。FFT フレーム・サイズが FFT order より小さい場合は、0 が埋めこまれます。

■ **FFT Windowing Function**

Rectangle (矩形)、Bartlett、Blackman(ブラックマン)、Hanning (ハニング)、Hamming(ハミング) の窓関数の中から選択できます。これらの操作は、FFT の計算が実施される前に、データに対して実行されます。

■ **Number of Waterfall Frames**

表示されるウォータフォール型フレームの数を指定します。

■ **Waterfall Height(%)**

ウォータフォール型フレームの表示に使用される垂直方向のウィンドウの高さの割合を % で指定します。

6.1.3 Graph Title

作成するグラフの各々に固有の名称を付けて識別することができます。固有の名称を付けると、複数のウィンドウが開いているときに結果を区別するのに役立ちます。

6.1.4 Data Page

実際のターゲット、またはシミュレーション・ターゲットが複数のページ（たとえば、プログラム、データ、I/O）で構成されている場合、「Data Page」オプションを使用してページを指定することができます。リストの中から、Prog、Data、I/O のいずれかを選択します。これらの選択項目は、グラフ表示されている変数位置またはメモリ位置のページが、プログラムであるか、データであるか、I/O であるかを示します。

注 : シミュレータは I/O メモリ・ページをサポートしません

'C54x DSP **用のシミュレータは、**I/O **メモリ・ページをサポートしません。**

6.1.5 Start Address

これは、グラフ化されるデータが入っている（実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上の）獲得バッファの開始位置です。このグラフが更新されるときには、この位置から開始する獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。次に、獲得バッファは表示バッファを更新します。そして、この表示バッファがグラフ化されます。

「Start Address」フィールドには、任意の有効な C 式を入力できます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。つまり、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

「Display Type」フィールドの値と「Interleaved Data Sources」フィールドの状況に応じて、1 つか 2 つの「Start Address」フィールドに値を設定する必要があります。詳細は、6.1.2 節 「Display Type」を参照してください。

6.1.6 Acquisition Buffer Size

これは、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上で、ユーザが使用している獲得バッファのサイズです。たとえば、一度に 1 つずつサンプルを処理する場合は、このフィールドに 1 を入力します。「 Left-Shifted Data Display」フィールドをイネーブルにし、プログラム内の正しい位置にその表示を接続します（詳細は、4.4.2 節 「プローブ・ポイントの接続」を参照)。

プログラムが一度にフレーム全体（複数のサンプル）を処理し、そのフレームだけに関心がある場合は、「Acquisition Buffer Size」フィールドと「Display Data Size」フィールドに同じ値を入力します。そして、「Left-Shifted Data Display」オプションをオフにします。

グラフが更新されるときには、獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから読み取られ、表示バッファを更新します。この表示バッファはグラフ化されます。

「Acquisition Buffer Size」フィールドには、任意の有効な C 式を入力することができます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。したがって、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

6.1.7 Display Data Size

これは、ユーザが使用する表示バッファのサイズです。この表示バッファの内容は、画面上にグラフ表示されます。表示バッファはホスト上に置かれているので、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上に信号が存在しなくなった場合でも、その信号の履歴を表示することができます。

この表示のサイズは、時間／周波数領域（Display Type）オプションに対してユーザが選択した内容に応じて異なります。時間領域グラフの場合、「Display Data Size」には、グラフが表示するサンプル数が指定されています。表示バッファでは、前処理は行われません。通常、「Display Data Size」は、「Acquisition Buffer Size」より大きいか、等しくなります。「Display Data Size」が「Acquisition Buffer Size」より大きい場合は、表示バッファ全体が左シフトされ、獲得バッファのデータが右端からシフトインされます。周波数領域グラフ (FFT Magnitude、Complex FFT、FFT Magnitude and Phase) の場合、「Display Data Size」は、FFT 周波数分析に使用される FFT フレーム・サイズ（最も近い 2 の累乗に切り上げ）を示します（6.1.2 節 「Display Type」を参照）。

「Display Data Size」フィールドには、任意の有効な C 式を入力できます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。したがって、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

6.1.8 DSP Data Type

このフィールドでは、次のデータ型の中から選択することができます。

- ❏ 32 ビット符号付き整数
- ❏ 32 ビット符号なし整数
- ❏ 32 ビット浮動小数点
- ❏ 32 ビット IEEE 浮動小数点
- ❏ 16 ビット符号付き整数
- ❏ 16 ビット符号なし整数
- ❏ 8 ビット符号付き整数
- ❏ 8 ビット符号なし整数

符号付き整数を Q-Value と組み合わせて使用すると、固定小数点値を表現することができます。

6.1.9 Q-Value

このフィールドには、ゼロ以外の Q 値が入ります。Q 値は、整数の小数表記です。実際のターゲットまたはシミュレーション・ターゲット上のデータは、この Q 値を使用して表現されます。データは、整数のバイナリ表記に小数点を挿入して作成されます。その結果、精度が高くなります。Q-Value には、次の公式に従って変位の量を指定します。

$$New\_integer\_value = integer / 2^{Q\text{-}Value}$$

Q-Value が xx の場合、これは、小数点が最下位ビット (LSB) から xx 桁ずらされている、符号付き 2 の補数表記の整数であることを示します。

6.1.10 Sampling Rate (Hz)

このフィールドには、アナログからディジタルへの変換などの場合のように、獲得バッファのサンプル用のサンプリング周波数が入っています。このサンプリング・レートは、グラフ上に表示される時間値と周波数値の計算に使用されます。

時間領域グラフの場合、このフィールドでは、時間軸の値を計算します。この軸には 0 〜 (Display Data Size * 1/Sampling Rate) のラベルが付きます。

周波数領域グラフ (FFT Magnitude、Complex FFT、FFT Magnitude and Phase) の場合、このフィールドには、FFT 周波数分析に使用されるサンプル数（最も近い 2 の累乗に切り上げ）が入ります。このグラフは、0 〜 Sampling Rate/2 の範囲で信号の周波数の内容を表示します。

6.1.11 Plot Data From

このフィールドでは、獲得バッファ内のデータの順序を決定します。「Left to Right」と「Right to Left」のオプションを切り換えることができます。「Left to Right」では、獲得バッファ内の最初のサンプルが、最新または最近着信したサンプルであると見なされます。「Right to Left」では、獲得バッファ内の最初のサンプルが最も古いサンプルであると見なされます。

6.1.12 Left-Shifted Data Display

このオプションは、獲得バッファを表示バッファにマージする方法を制御します。「Yes」を選択してこのオプションをイネーブルにするか、「No」を選択してディセーブルにすることができます。

グラフが更新されるとき、獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされ、表示バッファにマージされます。「Left-Shifted Data Display」をイネーブルにすると、表示バッファ全体が左にシフトされ、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボードの獲得バッファの値が、右端からシフトインされます。始動時に、表示バッファ内のすべての値が 0 に初期化されることに注意してください。「Left-Shifted Data Display」オプションがイネーブルでないと、表示バッファの値は、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボードの獲得バッファによって上書きされます。

「Left-Shifted Data Display」オプションは、信号を逐次に処理する場合に便利です。サンプルは一度に 1 つしか利用できませんが、「Left-Shifted Data Display」を使用すると、そのサンプルの履歴を表示することができます。

ウィンドウに関連付けられているプローブ・ポイントに達すると（詳細は 4.4.2 節 「プローブ・ポイントの接続」を参照）、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボードのデータのブロックが読み取られ、表示が更新されます。データを表示上に左シフトする場合、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボードのデータが有効である場合だけ、グラフ・ウィンドウが更新されることを確認してください。

6.1.13 Display Peak and Hold

このオプションを使用すると、連続したグラフのピーク値を表示することができます。「On」を選択するとイネーブルになり、「Off」を選択するとディセーブルになります。

「Display Peak and Hold」オプションをイネーブルにすると、連続したデータの獲得または更新を通じて得られたピークの履歴が保持されます。新しいバッファが獲得されると、新しい FFT 計算が実行されます。この新しい計算における特定サンプルの絶対値が、グラフ表示された直前のサンプルのピーク値を超える場合、その新しいグラフは、そのピークを含むように調整されます。サンプルがピーク値より小さい場合には、そのグラフの現在のピーク値が保持されます。

「Display Peak and Hold」オプションをディセーブルにすると、グラフ上のサンプルのピーク値を保持するための調整は行われません。表示される FFT 値の大きさは、現在のフレーム・バッファ上の計算だけが反映されます。

6.1.14 Autoscale

このオプションでは、Y 軸の最大値が自動的に判別できるようになります。「On」を選択するとイネーブルになり、「Off 」を選択するとディセーブルになります。

「Autoscale」をイネーブルにすると、グラフは表示バッファ内の最大値を使用して Y 軸の範囲を設定し、それに応じてすべての値をグラフ表示します。「Autoscale」をディセーブルにすると、「Graph Property」ダイアログ・ボックス に追加フィールドが表示されます。

- **Maximum Y-Value**

  グラフ上に表示される Y 軸の最大値を設定します。

6.1.15 DC Value

このオプションは、Y 軸の範囲の中点を設定します。Y 軸は、「DC Value」フィールドに入力された値に関して対称です。この値は、「Autoscale」フィールドがイネーブルであるかどうかに関係なく、イネーブルです。「FFT Magnitude」表示の場合、このフィールドは無視されます。

6.1.16 Axes Display

このオプションは、グラフ・ウィンドウ内の X 軸と Y 軸の表示のオン／オフを切り換えます。「On」を選択すると、軸はイネーブルになり、「Off」を選択するとディセーブルになります。

6.1.17 Status Bar Display

このオプションは、グラフ・ウィンドウ下部にあるステータス・バー表示のオン／オフを切り換えます。「On」を選択すると、その表示がイネーブルになり、「Off」を選択するとディセーブルになります。

6.1.18 Magnitude Display Scale

このフィールドでは、グラフ内のデータ値に使用されるスケーリング関数を設定します。次のオプションの中から選択することができます。

- Linear: 未変更の整数値を使用します。
- Logarithmic: 関数 20 ×log(x) を使用します。

6.1.19 Data Plot Style

このフィールドでは、グラフ内でデータがどのように表現されるかを設定します。次のオプションの中から選択することができます。

- ❏ Line: データ値を直線で結びます。
- ❏ Bar: 値の表示に縦線を使用します。

6.1.20 Grid Style

このフィールドでは、グラフ内の水平および垂直の背景線のパターンを設定します。次のオプションの中から選択することができます。

- ❏ No Grid
- ❏ Zero Line: 0 軸だけを表示します。
- ❏ Full Grid: 完全なグリッドを表示します。

6.1.21 Cursor Mode

このフィールドでは、グラフ内のカーソルの外観と機能を設定します。次のオプションの中から選択することができます。

- ❏ No Cursor
- ❏ Data Cursor: グラフの画面上に表示され、グラフのステータス・バー上にカーソルの座標が表示されます。
- ❏ Zoom Cursor: グラフの区域が拡大できるようになります。その区域の 1 つの隅にカーソルを置き、左マウス・ボタンを押したまま、希望する区域を囲む長方形をドローします。

6.2 Constellation

このグラフ・メニューには、多数のオプションが含まれています。これらのオプションを使用すると、データを柔軟に表示できるようになります。constellation グラフを使用すると、入力信号から情報が抽出される際の効率を測定できます。入力信号は 2 つのコンポーネントに分割され、この結果得られたデータは、時間をベースにした直交座標系を使用して作図されます。この作図は、一方の信号をもう一方の信号と対比させます（Y ソース対 X ソース。この場合、Y は Y 軸上に、X は X 軸上にプロットされます）。

「Graph Property」 ダイアログ・ボックス を表示するには、「View」→「Graph」→「Constellation」コマンドを使用します。フィールド名が左の欄に表示されます。必要に応じて、右欄の値を調整することができます。その後 「OK」をクリックします。設定されたプロパティを使用して、グラフ・ウィンドウが表示されます。このグラフ・ウィンドウでこれらのパラメータを変更するには、マウスを右クリックし、「Properties」を選択し、必要に応じてパラメータを調整します。また、プログラム内の任意のポイントでグラフを更新することもできます（詳細は、4.4.2 節 「プローブ・ポイントの接続」を参照）。

すべての入力フィールドは、C 式入力フィールドです。数値入力を必要とするすべてのフィールド（たとえば、「Start Address」や「Acquisition Buffer Size」）に、シンボルを含む式を使用することができます。詳細は、2.3.3.1 節 「式の中でシンボルを使用する方法」を参照してください。

6.2.1 Constellation の働き

「Graph」ウィンドウには 2 つのバッファが関連付けられています。獲得 (acquisition) バッファと表示 (display) バッファです。獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードに置かれています。獲得バッファには、ユーザの関心があるデータが入っています。グラフが更新されるときには、獲得バッファが、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから読み取られ、表示バッファを更新します。表示バッファはホスト・メモリに置かれているので、データの履歴を保持します。このグラフは、表示バッファ内のデータから生成されます。

すべてのオプション項目に入力し、「OK」を押すと、「Graph」ウィンドウは、「Acquisition Buffer Size」フィールドに入力された長さの DSP データを含む獲得バッファを受け取ります。この獲得バッファは、DSP データ・メモリ空間の「Start Address」で指定された位置から始まります。「Constellation Points」に指定されたサイズの表示バッファは、ホスト・メモリ内で割り当てられます。初めは、表示するデータを含んでいません。

このグラフが更新されるときには、表示バッファ全体が、「Acquisition Buffer Size」フィールドで指定された値だけ左にシフトされ、獲得バッファの値が右端からシフトインされます。この方法は、信号を逐次に処理する場合に便利です。サンプルは一度に 1 つしか利用できませんが、この方法により、そのサンプルの履歴を表示することができます。関連付けられているプローブ・ポイントに達すると（4.4 節 「プローブ・ポイント」を参照）、DSP データのブロックが読み取られ、表示が更新されます。

以下の節では、「Graph Property」ダイアログ・ボックス 内の入力フィールドについて説明します。

6.2.2 Display Type

「Graph Property」ダイアログ・ボックス 内の「Display Type」オプションには、右欄のドロップダウン・メニューに複数のオプションが含まれています。「View」→「Graph」→「Constellation」コマンドを使用すると、この「Constellation」オプションはデフォルトで表示されます。このフィールドのその他のオプションは、time/frequency（6.1 節 「Time/Frequency」を参照）、eye（6.3 節 「Eye Diagram」を参照）、または image グラフ（6.4 節 「Image Graph」を参照）に対応しています。

6.2.3 Graph Title

作成するグラフの各々に、固有の名称を付けて識別することができます。固有の名称を付けると、複数のウィンドウが開いているときに結果を区別するのに役立ちます。

6.2.4 Interleaved Data Sources

信号ソースがインターリーブされるかどうかを指定します。この表示オプションを切り換えると、1 つのバッファ入力が 2 つのソースを表すことができるようになります。このオプションを「Yes」に設定すると、奇数サンプルが最初のソース（X ソース）を表し、偶数サンプルが 2 番目のソース（Y ソース）を表すことになります。このオプションを「Yes」に設定すると、「Graph Property」ダイアログ・ボックス に次の追加フィールドが作成されます。

- ❑ **Start Address**

  グラフ化されるデータが入っている（実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上の）獲得バッファの開始位置。このグラフが更新されるときには、この位置から開始する獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。次に、この獲得バッファは表示バッファを更新します。そして、この表示バッファがグラフ化されます。「Start Address」フィールドには、任意の有効な C 式を入力できます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。つまり、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

「Interleaved Data Sources」を「No」に設定すると、次の追加フィールドが作成されます。

- **Start Address - X Source**
- **Start Address - Y Source**
- **Index Increment**

  データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。

6.2.5 Data Page

実際のターゲット、またはシミュレーション・ターゲットが複数のページ（たとえば、プログラム、データ、I/O）で構成されている場合は、「Data Page」オプションを使用してページを指定することができます。リストの中から、Prog、Data、I/O のいずれかを選択します。これらの選択項目は、グラフ表示されている変数位置またはメモリ位置のページが、プログラムであるか、データであるか、I/O であるかを示します。

注 : シミュレータは I/O メモリ・ページをサポートしません

'C54x DSP 用のシミュレータは、I/O メモリ・ページをサポートしません。

6.2.6 Acquisition Buffer Size

これは、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上でユーザが使用している獲得バッファのサイズです。たとえば、一度に 1 つずつサンプルを処理する場合は、このフィールドに 1 を入力します。プログラム内の正しい位置に表示を接続していることを確認してください（詳細は、4.4.2 節「プローブ・ポイントの接続」を参照）。

グラフが更新されるときには、獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから読み取られ、表示バッファを更新します。この表示バッファはグラフ化されます。

「Acquisition Buffer Size」フィールドには、任意の有効な C 式を入力できます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。したがって、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

6.2.7 Index Increment

このフィールドでは、データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。

このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。「Interleaved Data Sources」オプション（6.1.2 節「Display Type」を参照）をイネーブルにすると、この「Index Increment」オプションはディセーブルになります。

6.2.8 Constellation Points

これは、画面上にグラフ表示される表示バッファのサイズです。表示バッファはホスト上に置かれているので、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上に信号が存在しなくなった場合でも、その信号の履歴が表示されます。

「Constellation points」は、グラフが表示するサンプルの最大数です。通常、「Constellation Points」フィールドは「Acquisition Buffer Size」フィールドより大きいか、等しくなります。「Constellation Points」が「Acquisition Buffer Size」より大きい場合、表示バッファ全体が左シフトされ、獲得バッファのデータが右端からシフトインされます。

「Constellation Points」フィールドには、任意の有効な C 式を入力することができます。「Graph Property」ダイアログ・ボックス で 「OK」をクリックすると、この式が計算されます。

6.2.9 DSP Data Type

このフィールドでは、次のデータ型の中から選択することができます。

- ❏ 32 ビット符号付き整数
- ❏ 32 ビット符号なし整数
- ❏ 32 ビット浮動小数点
- ❏ 32 ビット IEEE 浮動小数点
- ❏ 16 ビット符号付き整数
- ❏ 16 ビット符号なし整数
- ❏ 8 ビット符号付き整数
- ❏ 8 ビット符号なし整数

符号付き整数を Q-Value と組み合わせて使用すると、固定小数点値を表現することができます。

6.2.10 Q-Value

このフィールドには、ゼロ以外の Q 値が入ります。Q 値は、整数の小数表記です。実際のターゲットまたはシミュレーション・ターゲット上のデータは、この Q 値を使用して表現されます。データは、整数のバイナリ表記に小数点を挿入して作成されます。その結果、精度が高くなります。Q 値は、次の公式に従って変位の量を指定します。

$New\_integer\_value = integer / 2^{Q\text{-}Value}$

Q-Value が xx の場合、これは、小数点が最下位ビット (LSB) から xx 桁ずらされている、符号付き 2 の補数表記の整数であることを示します。

6.2.11 Minimum X-Value

この値は、グラフ上に表示される X 軸の最小値を設定します。

6.2.12 Maximum X-Value

この値は、グラフ上に表示される X 軸の最大値を設定します。

6.2.13 Minimum Y-Value

この値は、グラフ上に表示される Y 軸の最小値を設定します。

6.2.14 Maximum Y-Value

この値は、グラフ上に表示される Y 軸の最大値を設定します。

6.2.15 Symbol Size

このプロパティでは、各記号の表示サイズを設定することができます。各 constellation は、X 記号で表示されます。次のオプションは、この表示プロパティに対応しています。

- ❏ Dot: X シンボルではなく、ドットとして各ポイントを表示します。
- ❏ Small
- ❏ Medium
- ❏ Large
- ❏ Extra Large

6.2.16 Axes Display

このオプションは、グラフ・ウィンドウ内の X 軸と Y 軸の表示のオン／オフを切り換えます。「On」を選択するとイネーブルになり、「Off」を選択するとディセーブルになります。

6.2.17 Status Bar Display

このオプションは、グラフ・ウィンドウ下部にあるステータス・バーの表示のオン／オフを切り替えます。「On」を選択すると、表示はイネーブルになり、「Off」を選択するとディセーブルになります。

6.2.18 Grid Style

このフィールドでは、グラフ内の水平および垂直の背景線のパターンを設定します。次のオプションの中から選択することができます。

- ❏ No Grid
- ❏ Zero Line: 0 軸だけを表示します。
- ❏ Full Grid: 完全なグリッドを表示します。

6.2.19 Cursor Mode

このフィールドでは、グラフ内のカーソルの外観と機能を設定します。次のオプションの中から選択することができます。

- ❏ No Cursor
- ❏ Data Cursor: グラフの画面上に表示され、グラフのステータス・バー上にカーソルが表示されます。
- ❏ Zoom Cursor: グラフの区域が拡大できるようになります。その区域の 1 つの隅にカーソルを置き、左マウス・ボタンを押したまま、希望する区域を囲む長方形をドローします。

6.3 Eye Diagram

「eye diagram」を使用すると、信号の精度を定性的に検査することができます。着信信号は、指定された表示範囲内で連続して重ね合わせられ、目の形状で表示されます。信号を連続的に作図し、0 交差が検出されるときにその信号を前に検出された 0 交差と重ね合わせることにより、時間の経過にともなった信号の周期が表示されます。0 交差は、(データ・ソースによって指定された）信号がウィンドウ・フレームの先頭に戻ることができる参照点です。重ね合わせが行われるのは、次のどちらかの場合です。

- ❑ 0 交差が検出され、「Minimum Interval Between Triggers」の条件が満たされる場合
- ❑ 表示長さに達した場合

0 交差のレベルは、「Trigger Level」フィールド内の値によって設定されます。0 交差は、「Trigger Level」フィールド内の値を各サンプルの値と比較し、信号の動向に留意して判別されます。信号の動向が 0 交差レベルを超えて上る場合は、そのレベルと等しいか、そのレベル以下の次のサンプルが、新しい 0 交差ポイントになります。この後、この信号の動向は、その 0 交差レベルより下であると見なされます。同様に、この動向がそのレベル以下である場合は、そのレベルと等しいか、そのレベルより上の次のサンプルが、新しい 0 交差ポイントになります。これ以降、信号の動向は、0 交差レベルより上であると見なされます。その動向は、初めは最初の信号サンプルの値から決定されます。

0 交差が検出される場合、0 交差は、データ・ソース信号を重ね合わせるためのトリガ・ポイント（6.3.4 節 「Trigger Source」を参照）の役目を果たします。ただし、これは「Minimum Interval Between Triggers」フィールド内の値が条件に合っている場合です。

0 交差が検出されない場合、データ・ソース信号は、「Display Length」フィールド内の値（最大の重ね合わせ長）に従って重ね合わされます。この値は、Y 軸範囲の中点でもあります。Y 軸は、「Display Length」フィールド内の値に関して対称です。「Trigger Level」と「Maximum Y-Value」を組み合わせると、Y 軸の最小値を得ることができます。

「Graph Property」ダイアログ・ボックス を表示するには、「View」→「Graph」→「Eye」コマンドを使用します。フィールド名が左の欄に表示されます。必要に応じて、右欄の値を調整することができます。その後 「OK」をクリックします。設定されたプロパティをもって、グラフ・ウィンドウが表示されます。このグラフ・ウィンドウでこれらのパラメータを変更するには、マウスを右クリックし、「Properties」を選択し、必要に応じてパラメータを調整します。また、プログラム内の任意のポイントでグラフを更新することもできます（詳細は、4.4.2 節 「プローブ・ポイントの接続」を参照)。

すべての入力フィールドは、C 式入力フィールドです。数値入力を必要とするすべてのフィールド（たとえば、Start Address や Acquisition Buffer Size）に、シンボルを含む式を使用することができます。詳細は、2.3.3.1 節 「式の中でシンボルを使用する方法」を参照してください。

6.3.1 Eye Diagram の働き

「Graph」ウィンドウには 2 つのバッファが関連付けられています。獲得 (acquisition) バッファと表示 (display) バッファです。獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードに置かれています。獲得バッファには、ユーザの関心があるデータが入っています。グラフが更新されるときには、獲得バッファが実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから読み取られ、表示バッファが更新されます。表示バッファはホスト・メモリに置かれているので、データの履歴を保持します。このグラフは、表示バッファ内のデータから生成されます。

各オプションの選択項目をすべて入力し、「OK」を押すと、そのグラフ・ウィンドウが更新されます。そのグラフ・ウィンドウは、「Acquisition Buffer Size」フィールドで指定された長さの DSP データを含む獲得バッファを受け取ります。この獲得バッファは、DSP データ・メモリ空間内の「Start Address」フィールドで指定された位置から開始します。「Persistence Size」フィールドで指定されたサイズの表示バッファは、ホスト・メモリ内で割り当てられ、その値はすべて 0 に初期化されます。

このグラフが更新されるときには、表示バッファ全体が、「Acquisition Buffer Size」フィールドで指定された値だけ左にシフトされ、獲得バッファの値が右端からシフトインされます。この方法は、信号を逐次に処理する場合に便利です。サンプルは一度に 1 つしか利用できませんが、この方法により、そのサンプルの履歴を表示することができます。関連付けられているプローブ・ポイントに達すると（4.4 節 「プローブ・ポイント」を参照）、DSP データのブロックが読み取られ、表示が更新されます。

以下の節では、「Graph Property」ダイアログ・ボックス 内の入力フィールドについて説明します。

6.3.2 Display Type

「Graph Property」ダイアログ・ボックス 内の「Display Type」オプションには、右欄のドロップダウン・メニューに複数のオプションが含まれています。「View」→「Graph」→「Eye」コマンドを使用すると、この「Eye Diagram」オプションはデフォルトで表示されます。このフィールドのその他のオプションは、time/frequency（6.1 節「Time/Frequency」を参照）、constellation（6.2 節「Constellation」を参照）、または image グラフ（6.4 節「Image Graph」を参照）に対応しています。

6.3.3 Graph Title

作成するグラフの各々に固有の名称を付けて識別することができます。固有の名称を付けると、複数のウィンドウが開いているときに結果を区別するのに役立ちます。

6.3.4 Trigger Source

「trigger source」は、実際のデータ・ソース信号値が測定される場合に比較対象となる、信号の理想的な表記です。「Yes」を選択してこのオプションをイネーブルにする場合、「*trigger source*」が 0 ラインと交差すると、必ずデータ・ソース信号はウィンドウ・フレームの先頭に重ね合わされます。この重ね合わせにより、その信号表記内に目の形が生成されます。「Trigger Source」フィールドをイネーブルにすると、「Graph Property」ダイアログ・ボックス に次の別のオプションが表示されます。

❑ **Interleaved Data Sources**

信号ソースがインターリーブされるかどうかを指定します。この表示オプションを切り換えると、1 つのバッファ入力が 2 つのソースを表すことができるようになります。このオプションを「Yes」に設定すると、奇数サンプルがデータ・ソースを表し、偶数サンプルがトリガ・ソースを表すことになります。このオプションを「Yes」に設定すると、ダイアログ・ボックス に次の追加フィールドが作成されます。

■ **Start Address**

グラフ化されるデータが入っている（実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上の）獲得バッファの開始位置。このグラフが更新されるときには、この位置から開始する獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。次に、この獲得バッファは表示バッファを更新します。そして、この表示バッファがグラフ化されます。「Start Address」フィールドには、任意の有効な C 式を入力できます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。つまり、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

「Interleaved Data Sources」を「No」に設定すると、次の追加フィールドが作成されます。

■ **Start Address - Data Source**

■ **Start Address - Trigger Source**

■ **Index Increment**

データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。

「Trigger Source」に対して「No」を選択する場合、0 ラインと交差するデータ・ソースは、信号がウィンドウ・フレームの先頭に重ね合わされるトリガとなります。この重ね合わせにより、その信号表記内に目の形が生成されます。

6.3.5 Data Page

実際のターゲット、またはシミュレーション・ターゲットが複数のページ（たとえば、プログラム、データ、I/O）で構成されている場合は、「Data Page」オプションを使用してページを指定することができます。リストの中から、Prog、Data、I/O のいずれかを選択します。これらの選択項目は、グラフ表示されている変数位置またはメモリ位置のページが、プログラムであるか、データであるか、I/O であるかを示します。

注 : シミュレータは I/O メモリ・ページをサポートしません

'C54x DSP **用のシミュレータは、**I/O **メモリ・ページをサポートしません。**

6.3.6 Acquisition Buffer Size

これは、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上でユーザが使用している獲得バッファのサイズです。たとえば、一度に 1 つずつサンプルを処理する場合は、このフィールドに 1 を入力します。プログラム内の正しい位置に表示を接続していることを確認してください（詳細は、4.4.2 節 「プローブ・ポイントの接続」を参照）。

グラフが更新されるときには、獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから読み取られ、表示バッファを更新します。表示バッファ内のデータは、「Acquisition Buffer Size」フィールドで指定された値だけ左にシフトされます。獲得バッファ内のデータは右端から表示バッファにシフトインされます。この表示バッファはグラフ化されます。

「Acquisition Buffer Size」フィールドには、任意の有効な C 式を入力することができます。この式は、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。したがって、このフィールドにシンボルを入力した場合には、そのシンボルの値が後で変更されても、このパラメータを入力し直す必要はありません。

6.3.7 Index Increment

このフィールドでは、データ・グラフに表示するサンプルのインデックス増分を指定します。このフィールドへの指定は、インターリーブされていないソースに対するサンプル・オフセット値 1 と等価になります。これにより、複数のソースから、1 つのグラフに使用する信号データを抽出できるようになります。たとえば、Index Increment 2 はサンプル・オフセット値 2 に対応します。つまり、獲得バッファ内のサンプルを 1 つおきにグラフ表示します。したがって、このフィールドに対応するオフセット値を指定すると、表示用の複数のデータ・ソースを指定することができます。

このオプションは、インターリーブされたソースを取り出すための一般的な手法を提供します。「Interleaved Data Sources」オプション（6.1.2 節「Display Type」を参照）をイネーブルにすると、この「Index Increment」オプションはディセーブルになります。

6.3.8 Persistence Size

これは、表示バッファのサイズです。この表示バッファの内容は、画面上にグラフ表示されます。表示バッファはホスト上に置かれているので、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上に信号が存在しなくなった場合でも、その信号の履歴が表示されます。

「Persistence Size」フィールドには、そのグラフが表示する履歴内のサンプル数が入ります。通常、「persistence size」は、「Acquisition Buffer Size」フィールド内の値より大きいか、等しくなります。「Persistence Size」フィールドが「Acquisition Buffer Size」フィールドより大きい場合、表示バッファ全体が左シフトされ、獲得バッファのデータが右端からシフトインされます。

グラフの表示は累積され、その結果、グラフ・ウィンドウは、指定された「Persistence Size」より多いサンプルを表示する場合があります。このグラフ・ウィンドウ上でマウスを右クリックし、「Refresh」を選択すると、指定された「Persistence Size」を超えた着信サンプルのある表示バッファから、古いデータを消去することができます。この消去により、「Persistence Size」は表示バッファの長さに設定されます。

Persistence Size フィールドには、任意の有効な C 式を使用できます。「Graph Property」ダイアログ・ボックス で 「OK」をクリックすると、この式が計算されます。

6.3.9 Display Length

このフィールドでは、ウィンドウ内に表示される時間枠を設定します。また、2 つのトリガ・ポイント間の最大重ね合わせ長も設定されます。0 交差が検出されないときに、現行サンプルと前回のトリガ・ポイント間の間隔が「Display Length」フィールド内の値より大きい場合、データ・ソース内の信号は、画面上の左トリガ・ポイントから重ね合わされます。

6.3.10 Minimum Interval Between Triggers

このフィールドでは、2 つの連続したトリガ・ポイント間の最小サンプル間隔を設定します。0 交差が検出されるときに、このポイントと前回のトリガ・ポイント間の間隔が、この最小間隔と同じか、それより大きい場合、データ・ソース内の信号はウィンドウ・フレームの先頭に重ね合わされます。

しかし、0 交差が検出されるときに、このポイントと前回のトリガ・ポイント間の間隔が、この最小間隔より小さい場合、信号は重ね合わされません。この信号は、次の条件のどちらかが満たされるまでプロットされ続けます（重ね合わされません）。

- ❏ 「Display Length」フィールド内の値（最大重ね合わせ長）に達した場合、信号は、画面上の左トリガ・ポイントまで重ね合わされます。
- ❏ 0 交差および最小間隔条件が満たされた場合、信号は、その 0 交差ポイントに従って重ね合わされます。

6.3.11 Pre-Trigger (in samples)

このオプションでは、左トリガ・ポイントより前に表示されるサンプルの数を設定します。このオプションは、画面の左側または右側に左トリガ・ポイントを水平移動させます。このトリガ・ポイントの周囲の信号を表示する場合に、このオプションが便利です。

このオプションを 0 に設定すると、左トリガ・ポイントはグラフ・ウィンドウの左境界上に置かれます。このオプションを正の値に設定すると、左トリガ・ポイントはウィンドウの右境界方向に移動します。このオプションを負の値に設定すると、左トリガ・ポイントは左境界ウィンドウの左側（ウィンドウの外）に移動します。重ね合わせが有効である場合、そのポイントはウィンドウの右境界から遠くなります。

6.3.12 DSP Data Type

このフィールドでは、次のデータ型の中から選択することができます。

- ❏ 32 ビット符号付き整数
- ❏ 32 ビット符号なし整数
- ❏ 32 ビット浮動小数点
- ❏ 32 ビット IEEE 浮動小数点
- ❏ 16 ビット符号付き整数
- ❏ 16 ビット符号なし整数
- ❏ 8 ビット符号付き整数
- ❏ 8 ビット符号なし整数

符号付き整数を Q -Value と組み合わせて使用すると、固定小数点値を表現することができます。

6.3.13 Q-Value

このフィールドには、ゼロ以外の Q 値が入ります。Q 値は、整数の小数表記です。実際のターゲットまたはシミュレーション・ターゲット上のデータは、この Q 値を使用して表現されます。データは、整数のバイナリ表記に小数点を挿入して作成されます。その結果、精度が高くなります。Q 値は、次の公式に従って変位の量を指定します。

$New\_integer\_value = integer / 2^{Q\text{-}Value}$

Q-Value が xx の場合、これは、小数点が最下位ビット (LSB) から xx 桁ずらされている、符号付き 2 の補数表記の整数であることを示します。

6.3.14 Sampling Rate

このフィールドには、アナログからディジタルへの変換などの場合のように、獲得バッファのサンプル用のサンプリング周波数が入っています。軸の値は、0 〜 (Display Length * 1/Sampling Rate) です。軸の値には、これらの値から Pre-Trigger (in samples) パラメータの値を引いた値が使われます。

6.3.15 Trigger Level

「Trigger Source」フィールドがイネーブルの場合、このフィールドでは 0 交差レベルが設定されます。

6.3.16 Maximum Y-Value

この値は、グラフ上に表示される Y 軸の最大値を設定します。「Maximum Y-Value」フィールドと「Trigger Level」フィールドの値により、Y 軸の最小値が決定されます。

6.3.17 Axes Display

このオプションは、グラフ・ウィンドウ内の X 軸と Y 軸の表示のオン／オフを切り換えます。「On」を選択するとイネーブルになり、「Off」を選択するとディセーブルになります。

6.3.18 Time Display Unit

このフィールドでは、グラフの時間軸の計測単位を指定します。このオプションおよび「Sampling Rate」フィールドの値により、軸の値が決定されます。

次の値の中から選択できます。

- ❏ s: 秒
- ❏ ms: ミリ秒
- ❏ us: マイクロ秒
- ❏ sample: 表示バッファ・インデックスによって値を表示します。

6.3.19 Status Bar Display

このオプションは、グラフ・ウィンドウ下部にあるステータス・バーの表示のオン／オフを切り換えます。「On」を選択すると、表示がイネーブルになり、「Off」を選択するとディセーブルになります。

6.3.20 Grid Style

このフィールドでは、グラフ内の水平および垂直の背景線のパターンを設定します。次のオプションの中から選択することができます。

- ❏ No Grid
- ❏ Zero Line: 0 軸だけを表示します。
- ❏ Full Grid: 完全なグリッドを表示します。

6.3.21 Cursor Mode

このフィールドでは、グラフ内のカーソルの外観と機能を設定します。次のオプションの中から選択することができます。

- ❏ No Cursor
- ❏ Data Cursor: グラフの画面上に表示され、グラフのステータス・バー上にカーソルが表示されます。
- ❏ Zoom Cursor: グラフの区域が拡大できるようになります。その区域の 1 つの隅にカーソルを置き、左マウス・ボタンを押したまま、希望する区域を囲む長方形をドローします。

6.4 Image Graph

このグラフ・メニューには、複数のオプションが含まれています。これらのオプションを使用すると、データを柔軟に表示できるようになります。「Image Graph」を使用すると、画像処理アルゴリズムをテストできます。画像データは、RGB および YUV データ・ストリームに基づいて表示されます。「Graph Property」ダイアログ・ボックス を表示するには、「View」→「Graph」→「Image」コマンドを使用します。フィールド名が左の欄に表示されます。必要に応じて、右欄の値を調整できます。その後 「OK」をクリックします。設定されたプロパティをもって、グラフ・ウィンドウが表示されます。このグラフ・ウィンドウでこれらのパラメータを変更するには、マウスを右クリックし、「Properties」を選択し、必要に応じてパラメータを調整します。また、プログラム内の任意のポイントでグラフを更新することもできます（詳細は、4.4.2 節 「プローブ・ポイントの接続」を参照）。

すべての入力フィールドは、C 式入力フィールドです。数値入力を必要とするすべてのフィールド（たとえば、Start Address や Acquisition Buffer Size）に、シンボルを含む式を使用することができます。詳細は、2.3.3.1 節 「式の中でシンボルを使用する方法」を参照してください。

6.4.1 Image Graph の働き

「Graph」ウィンドウには 2 つのバッファが関連付けられています。獲得 (acquisiton) バッファと表示 (display) バッファです。獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードに置かれています。獲得バッファには、ユーザに関係するデータが入っています。グラフが更新されるときには、獲得バッファが実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから読み取られ、表示バッファが更新されます。表示バッファはホスト・メモリに置かれているので、データの履歴を保持します。このグラフは、表示バッファ内のデータから生成されます。

各オプションの選択項目をすべて入力し、「OK」を押すと、そのグラフ・ウィンドウが更新されます。そのグラフ・ウィンドウは、「Start Address」フィールドで指定されたデータ・メモリ位置から始まる、DSP の獲得バッファを受け取ります。表示バッファは、ホスト・メモリ内で割り当てられます。最初は、表示するデータは含まれていません。獲得バッファと表示バッファは、いずれも画像全体を保持します。

このグラフが更新されるときには、画像全体が入っている獲得バッファは、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードからフェッチされます。そして、表示バッファはこの獲得バッファによって上書きされます。関連付けられているプローブ・ポイントに達すると（4.4 節 「プローブ・ポイント」を参照）、DSP データのブロックが読み取られ、表示が更新されます。

以下の節では、「Graph Property」ダイアログ・ボックス 内の入力フィールドについて説明します。

6.4.2 Graph Title

作成するグラフの各々に固有の名称を付けて識別することができます。固有の名称を付けると、複数のウィンドウが開いているときに結果を区別するのに役立ちます。

6.4.3 Color Space

このフィールドでは、データが解釈され、表示される方法を指定します。次の「Color Space」オプションの中から選択することができます。

❑ **YUV**

どの Y、U、および V のサンプルも、8 ビットを使用して表示されます。YUV を選択すると、「Graph Property」ダイアログ・ボックス に次の追加フィールドが表示されます。

■ **YUV Ratio**

Y、U、および V のサンプル間の関係を指定します。次のオプションの中から選択します。

4:1:1 － 4 つの水平 Y サンプルごとに、1 つの U および V サンプルがあります。垂直方向の U と V の縮小は行われません。

4:2:2 － 2 つの水平 Y サンプルごとに、1 つの U および V サンプルがあります。垂直方向の U と V の縮小は行われません。

4:2:0 －垂直方向と水平方向の両方で、U と V の 2:1 の縮小が行われます。つまり、2x2 の Y サンプルごとに、1 つの U と V のサンプルがあります。

■ **Transformation of YUV Values**

YUV を RGB に変換します。YUV を変換するには、2 つのステップがあります。YUV から Y'U'V' への変換と、Y'U'V' から RGB への変換です。次のオプションの中から選択できます。

Unity (none): 単一マトリックスを使用して、YUV を Y'U'V' に変換します。

ITU-R BT 601 (CCIR601): 推奨事項 ITU-R BT.601 (旧 CCIR 601) luma に従って、YUV を Y'U'V' に変換するための CCIR601 マトリックスを使用して、YUV を RGB に変換します。

■ **Start Address - Y Source**

■ **Start Address - U Source**

■ **Start Address - V Source**

❑ **RGB**

R、G、および B のサンプル間の関係を指定します。RGB を選択すると、「Graph Property」ダイアログ・ボックス に次の別のフィールドが表示されます。

■ **Interleaved Data Sources**

信号ソースがインターリーブされるかどうかを指定します。このオプションを「Yes」に設定すると、「Graph Property」ダイアログ・ボックス に次の追加フィールドが表示されます。

▮ **Start Address**

三重にインターリーブドされたソースのある単一バッファの開始位置を表します。これは、インターリーブの長さが 1 である、3 ソース入力バッファを意味します。たとえば、シーケンス「$R_0G_0B_0R_1G_1B_1$...」は、ピクセル 0 の RGB コンポーネントの後に、ピクセル 1 のコンポーネントが続いたストリームを表しています。

▮ **Bits Per Pixel**

次のオプションの中から選択することができます。

8 (256 Color Palette): 各ピクセルは 8 ビット値であり、パレットに連動します。

16 (6 Bits for Green): 各ピクセルは、2 バイト値です。その中の 5 ビットは赤色用、6 ビットは緑色用、5 ビットは青色用です。

24: 各ピクセルは 3 バイト値です。赤色、緑色、および青色用にそれぞれ 8 ビットずつです。

32: 各ピクセルは 4 バイト値です。赤色、緑色、および青色用にそれぞれ 8 ビットずつであり、最上位バイトは使用されません。

「Bits Per Pixel」に 16 (6 Bits for Green)、24、または 32 を選択すると、次の追加プロパティが表示されます。

▮ **Image RGB Order**

赤色、緑色、および青色の順序を指定します。RGB、BGR、RBG、BRG、GBR、GRB のオプションの中から選択します。

「Bits Per Pixel」で 8 (256 Color Palette) を選択すると、次の追加プロパティが表示されます。

▮ **Palette Option**

8 ビット・ピクセル値（パレット・インデックス）を RGB カラー値に変換することを指定します。Uniform Palette of 256 Colors、Gray Scale of 256 Colors、User Defined (256 Colors) のオプションの中から選択します。

「Palette Option」で User Defined (256 Colors) を選択すると、次の追加プロパティ項目が表示されます。

▮ **Palette Address**

ユーザ提供パレットを取り出す始点となる開始アドレスを指定します。このアドレスは、任意の有効な C 式であり、実際のターゲット、またはシミュレーション・ターゲットからサンプルが読み取られるたびに、計算し直されます。

- **Palette Entry 4-Byte Aligned**

  「Yes」を選択すると、各パレット項目は 4 バイト値（最下位バイトは使用されません）になります。赤色、緑色、および青色用にそれぞれ 8 ビットを使用します。「No」を選択すると、各パレット項目は 3 バイト値になります。赤色、緑色、および青色用にそれぞれ 8 ビットずつです。

- **Palette RGB Order**

  赤色、緑色、および青色の順序を指定します。RGB、BGR、RBG、BRG、GBR、GRB のオプションの中から選択します。

- **Read Palette Once Only**

  「No」を選択すると、グラフが更新されるたびにパレットがフェッチされます。「Yes」を選択すると、グラフがたとえ何回更新されても、パレットは実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから 1 回しかフェッチされません。グラフのプロパティを変更すると、パレットは強制的に再ロードされます。

「Interleaved Data Sources」フィールドに「No」を選択すると、各ピクセルの R、G および B のコンポーネントが、それぞれ 8 ビット幅になり、0 ～ 255 の範囲の値になります。「Graph Property」ダイアログ・ボックス に次の追加フィールドが作成されます。

- **Start Address - R Source**
- **Start Address - G Source**
- **Start Address - B Source**

6.4.4 Data Page

実際のターゲット、またはシミュレーション・ターゲットが複数のページ（たとえば、プログラム、データ、I/O）で構成されている場合は、「Data Page」オプションを使用してページを指定することができます。リストの中から、Prog、Data、I/O のいずれかを選択します。これらの選択項目は、グラフ表示されている変数位置またはメモリ位置のページが、プログラムであるか、データであるか、I/O であるかを示します。

注 : シミュレータは I/O メモリ・ページをサポートしません

'C54x DSP **用のシミュレータは、**I/O **メモリ・ページをサポートしません。**

6.4.5 Lines Per Display

このオプションでは、画像全体の高さをピクセル数で指定します。この値と「Pixels Per Line」フィールドで指定された値により、画像のサイズが決まります。グラフが更新されるときには、画像全体が、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから取り出され、表示されます。

6.4.6 Pixels Per Line

このオプションでは、画像全体の幅をピクセル数で指定します。この値と「Lines Per Display」フィールドで指定された値により、画像全体のサイズが決まります。グラフが更新されるときには、画像全体が、実際のターゲット・ボード、またはシミュレーション・ターゲット・ボードから取り出され、表示されます。

6.4.7 Byte Packing to Fill 32 Bits

このオプションでは、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上のデータのパッキング・フォーマットを指定します。

「No」を選択すると、バイト型のデータ・ストリームはパックされず、実際のターゲットまたはシミュレーション・ターゲット上の各データ値はバイト型と見なされます。

「Yes」を選択すると、バイト型のデータ・ストリームはパックされ、4 バイトごとに 1 つのパケットとしてグループ化されたものとして扱われます。このパケットは、32 ビット型符号なし整数の、実際のターゲットまたはシミュレーション・ターゲット上のデータ値です。このデータ値の最下位バイトは、そのパケット内の最初のバイトです。「Yes」を選択すると、「Graph Property」ダイアログ・ボックスに次の追加フィールドが表示されます。

- **Image Row 4-Byte Aligned**

  このオプションに「Yes」を選択すると、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上の各イメージ行は、4 バイトに桁合わせされます。「No」を選択すると、実際のターゲット・ボードまたはシミュレーション・ターゲット・ボード上の各イメージ行は、4 バイトに桁合わせされません。データ値には、1 つの行の終わりと次行の先頭用のデータのサンプルが入っている場合があります。

6.4.8 Image Origin

このフィールドでは、グラフ・ウィンドウ上のイメージの起点を指定します。Bottom Left、Top Left、Top Right、Bottom Right のオプションの中から選択します。

6.4.9 Uniform Quantization to 256 Colors

このオプションは、オリジナル画像が 256 色でない場合にだけ使用できます。「Yes」を選択すると、画像は、256 色の画像に均一に量子化されます。量子化された画像には赤色と緑色用に 8 つのレベルがあり、青色用に 4 つのレベルがあります。オリジナルの赤色、緑色、および青色の値が、これらのレベルのどれか 1 つにマップされます。「Yes」を選択すると、「Graph Property」ダイアログ・ボックス に次の追加フィールドが表示されます。

- **Error Diffusion**

  「Yes」を選択すると、量子化によって発生したエラーを拡散させて、色をより滑らかにします。「No」を選択すると、量子化エラーは調整されません。

「Uniform Quantization to 256 Colors」に対して「No」を選択すると、画像は量子化されずに RGB 色空間に表示されます。表示用のハードウェアが 256 色を超える色を表示できない場合、このオプションは強制的に「Yes」に設定されることに注意してください。

6.4.10 Status Bar Display

このオプションは、グラフ・ウィンドウ下部にあるステータス・バーの表示のオン／オフを切り換えます。「On」を選択すると、その表示がイネーブルになり、「Off」を選択するとディセーブルになります。

6.4.11 Cursor Mode

このフィールドでは、グラフ内のカーソルの外観と機能を設定します。次のオプションの中から選択することができます。

- No Cursor
- Data Cursor: グラフの画面上に表示され、グラフのステータス・バー上にカーソルが表示されます。
- Zoom Cursor: グラフの区域が拡大できるようになります。その区域の 1 つの隅にカーソルを置き、左マウス・ボタンを押したまま、希望する区域を囲む長方形をドローします。

第 7 章

メモリ・マップ

メモリ・マップは、メモリ内のアクセス可能なエリアとアクセス不可能なエリアを Code Composer Studio デバッガに知らせます。通常、このマップは、ユーザのリンカ・コマンド・ファイル内のメモリ定義と一致しています。MEMORY 疑似命令、およびリンカ・コマンド・ファイルの設定方法については、*Code Generation Tools* オンライン・ヘルプを参照してください。

項目 **ページ**

7.1 メモリ・マップへのアクセス

メモリ・マッピングをイネーブルにすると、Code Composer Studio デバッガは、提供されたメモリ・マップと照らして、各メモリ・アクセスをチェックします。ユーザが未定義エリア、または保護エリアにアクセスしようとすると、デバッガは、ターゲットへのアクセスを試みる代りに、デフォルト値を表示します。

注 : シミュレータ – メモリ・マップ設定

シミュレータは、ハードコーディングされたメモリ・マップ設定を使用して、ソフトウェアによってシミュレートされる DSP ファミリーを総称的に表示します。シミュレータのメモリ・マップ設定については、オンライン・ヘルプのトピック *Simulator – Memory Map Specifications* を参照してください。本章に記載されている方法を使用して、メモリ・マップ設定を操作することもできます。しかし、異常な動作を避けるために、プログラムのサイズは、指定されたメモリ・マップ範囲内に収めるようにしてください。

存在しないメモリへのアクセス

Code Composer Studio デバッガは、メモリ・アクセスをメモリ・マップと比較するときに、ハードウェアではなく、ソフトウェアでこのチェックを実行します。このデバッガは、プログラムが存在しないメモリにアクセスしようとするのを防止することができません。

次のように、ターゲットに有効なメモリ・マップ範囲を定義することができます。

- ❏ デバッガの使用中に、対話式にコマンドを入力できます。
- ❏ GEL 組み込み関数を使用して、メモリ・マップを定義できます。デバッガには、完全な一連のメモリ・マッピング・コマンドが用意されています。これらのコマンドは、General Extension Language (GEL) とメニュー・バーを使用して起動することができます。メモリ・マップを実装する最も簡単な方法は、このメモリ・マッピング・コマンドを GEL テキスト・ファイルに入れ、始動時に実行することです。

7.2 メモリ・マップの定義

「Memory Map」ダイアログ・ボックスを使用すると、対話式にメモリ・マップを定義し、リストすることができます。「Option」→「Memory Map」の順に選択して、ダイアログ・ボックスを起動します。

初めて Code Composer Studio を起動すると、メモリ・マップはオフになっています。任意のメモリ位置にアクセスでき、メモリ・マップは干渉しません。

新規メモリ・マップ範囲を追加する方法

アクセスしたいメモリ範囲を定義する手順は、次のとおりです。

1) 「Option」→「Memory Map」の順に選択します。「Memory Map」ダイアログ・ボックスが表示されます。
2) 「Enable Memory Mapping」チェック・ボックスにチェック・マークが付いていることを確認します。チェック・マークが付いていない場合、Code Composer Studio デバッガは、ターゲット上のアドレス指定可能なメモリ (RAM) のすべてが有効であると見なします。
3) 変更したいページ (Program、Data、または IO) と対応するフォルダを選択します。メモリ・ページが 1 つしかないプロセッサを使用している場合は、このステップをスキップします。プロセッサにメモリ・ページが 1 つしかない場合には、1 つのフォルダしか作成されません。
4) 新しいメモリ・マップ範囲の開始アドレスを、「Starting Address」入力フィールドに入力します。
5) この新しい範囲の長さを、「Length」入力フィールドに入力します。
6) この新しいメモリ範囲のリード／ライト特性を、「Attributes」フィールドの中から選択します。
7) 「Add」をクリックします。

このデバッガを使用すると、既存のメモリ範囲とオーバーラップする新しいメモリ範囲を入力することができます。この新しい範囲は有効であると見みなされ、オーバーラップした範囲の属性が、新しい属性に変更されます。

メモリ・マップの範囲が定義済みのときには、その範囲のリード／ライト属性を変更することもできます。この属性を変更するには、新しいメモリ・マップを（同じ開始アドレスと長さを指定して）定義し、「Add」をクリックします。デバッガは、既存の属性に新しい属性を上書きします。

既存のメモリ・マップ範囲を削除する方法

既存のメモリ・マップ範囲を削除することもできます。「Attributes」フィールドを「None - No Memory/Protected」に変更することができます。こうすると、このメモリ位置に対してリードもライトもできなくなります。次のようにすれば、メモリ・マップ範囲を削除することもできます。

1) 「Option」→「Memory Map」の順に選択します。「Memory Map」ダイアログ・ボックスが表示されます。
2) 「Memory Map List」ボックスで、削除したいメモリ・マップ範囲を選択します。
3) 「Delete」をクリックします。

メモリ・マップによって保護されているメモリ位置から読み取ろうとすると、Code Composer Studio デバッガは、そのターゲットからの読み取りを試みる代りに、保護された値を使用します。始動時のデフォルト値は 0 です。したがって、すべての無効なメモリ位置には、値 0 が表示されます。このデフォルト値を変更するには、「Memory Map」ダイアログ・ボックスの「Protected Value」入力フィールドに、ユーザ独自の値を入力します。0XDEAD のような値を代わりに使用すると、無効なメモリ位置を読み取ろうとしたことを明確に示すことができます。

7.3 GEL を使用してメモリ・マップを定義する方法

初めて Code Composer Studio を起動すると、メモリ・マップはオフになっています。任意のメモリ位置にアクセスでき、メモリ・マップは干渉しません。オプションの GEL ファイル名をパラメータとして指定して Code Composer Studio を起動すると、Code Composer Studio は、この GEL ファイルを自動的にロードします。StartUp() という名前の GEL 関数を使用する場合は、この GEL 関数が実行されます。このファイルでマップ関数を指定すると、ユーザの環境用のメモリ・マッピング要件を自動的に指定することができます。

次の GEL 関数を使用して、メモリ・マップを定義することができます。

GEL_MapAdd()	メモリ・マップを追加する
GEL_MapDelete()	メモリ・マップを削除する
GEL_MapOn()	メモリ・マップをイネーブルにする
GEL_MapOff()	メモリ・マップをディセーブルにする
GEL_MapReset()	メモリ・マップをリセットする

GEL_MapAdd() 関数は、有効なメモリ範囲を定義し、そのメモリ範囲のリード／ライト特性を指定します。次に示すものは、読み取りも書き込みも可能な、長さ 0xF000 の 2 つのブロックを定義するために使用することができる GEL ファイルの例です。

```
StartUp()
{
  GEL_MapOn();
  GEL_MapReset();
  GEL_MapAdd(0, 0, 0xF000, 1, 1);
  GEL_MapAdd(0, 1, 0xF000, 1, 1);
}
```

メモリ・マップを設定すると、「Option」→「Memory Map」コマンドを使用して、そのメモリ・マップを表示することができます。

上記の組み込み GEL メモリ操作関数の実装方法の詳細は、第 12 章「General Extension Language (GEL)」を参照してください。

第 8 章

「Watch」ウィンドウの使用

「Watch」ウィンドウを使用すると、変数と C 式の検査および編集を行うことができます。「Watch」ウィンドウでは、複雑な式の展開と縮小を行うことができます。また、項を評価し、異なる複数のフォーマットで結果を表示することもできます。Quick Watch 機能を使用すると、「Watch」ウィンドウに迅速に変数を追加できるようになります。本章では、デバッグ時に、これらの機能をどのように使用するかについて説明します。

項目 **ページ**

8.1 「Watch」ウィンドウに式を追加または削除する方法

「Watch」ウィンドウに式を追加する手順は、次のとおりです。

1) 「View」→「Watch Window」を選択します。

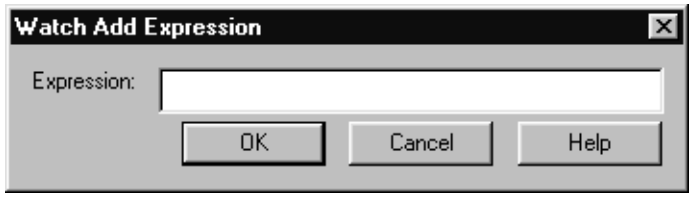
   または

   「Debug」ツールバー上で、「Watch」ウィンドウのボタンをクリックします。

   Watch Window: 

2) 「Watch」ウィンドウに新しい式を追加するには、以下の方法があります。

   - 4 つの 「Watch」ウィンドウのタブのどれか 1 つを選択します。キーボードの Insert キーを押します。Insert キーを押すと、「Watch Add Expression 」ダイアログ・ボックスが表示されます。検査する式を「Expression」フィールドに入力し、「OK」を押します。
   - 「Watch」ウィンドウで右クリックし、コンテキスト・メニューから 「Insert New Expression」を選択します。「Watch Add Expression 」ダイアログ・ボックスが表示されます。検査する式を「Expression」フィールドに入力し、「OK」を押します。
   - ソースまたは 「Dis-Assembly」ウィンドウ内で変数をダブルクリックして反転させた後、右クリックして、コンテキスト・メニューから「Add to Watch Window」を選択します。



シンボルを式として使用する方法

「Watch」ウィンドウでは、シンボル名を式として指定することができます。しかし、Code Composer Studio がシンボルを解釈する方法は、オブジェクト・ファイルにシンボリック・デバッグ情報が入っているかどうかによって異なります。

シンボルが C ソース・ファイル内で定義され、そのファイルの作成時にシンボリック・デバッグ情報 (-g) が指定されている場合、そのシンボルは、指定されたアドレスのメモリの内容を示す変数として扱われます。

シンボリック・デバッグ情報がないと、すべてのシンボルはアドレスとして扱われます。

たとえば、シンボル名を使用して、「Watch」ウィンドウに式を指定すると、次のようになります。

シンボリック・デバッグ情報が利用可能な場合、「Watch」ウィンドウには、そのシンボル名（変数）が表すメモリ位置の値が表示されます。

シンボリック・デバッグ情報が利用できない場合、「Watch」ウィンドウは、特定のアドレス・ラベルが存在することだけを示します。「Watch」ウィンドウには、このシンボルのアドレスが表示されます。このシンボルの表すメモリ位置の値を表示するには、そのシンボル名の前にアスタリスク (*) を付ける必要があります。

「Watch」ウィンドウ内の式を削除する方法

1) マウスでクリックするか、上または下矢印キーを使用して移動し、アクティブな「Watch」ウィンドウから削除したい式を選択します。
2) キーボードの Delete キーを押します。その式が展開表示される場合は、すべての部分式が 「Watch」ウィンドウから削除されます。

TI 固定小数点プロセッサでは、実際のターゲット、またはシミュレーション・ターゲットが複数のページから構成されている場合には、@ 記号を使用して特定のページを指定できます。@ を入力した後、prog、data、または io のどれか 1 つの用語を入力します。この用語は、このページがプログラム・ページであるか、データ・ページであるか、I/O ページであるかを指定します。たとえば、次のとおりです。

```
*(int *)0x1000@prog
*(int *)0x1000@data
```

8.1.1 Watch 変数の展開と縮小

配列、構造体、ポインタなどの複数の要素を含む変数は、その変数の前に + 符号、または ― 符号のどちらかが付いて表示されます。+ 記号は、この変数に要素が含まれ、展開可能であることを示します。― 記号は、この変数が完全に展開されていて、縮小可能であることを示します。

変数を展開または縮小する方法

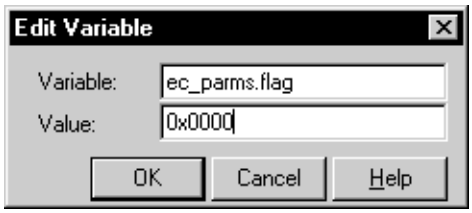
1) マウスを使用するか、下または上矢印キーを使用して、展開したい変数を選択します。
2) 変数が現在の変数であるときは、Enter キーを押せば展開状態（展開または縮小に）を切り換えることができます。

8.2 「Watch」ウィンドウで変数を編集する方法

「Watch」ウィンドウの式およびその値は、以下の手順で変更します。

1) 「Watch」ウィンドウのタブを選択します。「Watch」ウィンドウで、編集したい変数をマウスでクリックするか、下または上矢印キーを使用して、その変数を選択します。

2) その変数をダブルクリックして、「Edit Variable」ダイアログ・ボックスを表示します。



3) 必要に応じて、「Value」フィールド内の情報を編集します。

4) 既存の Watch 式を新しい式に置き換えることもできます。たとえば、既存の式を変更して、その表示フォーマットを変更することができます（8.3 節「「Watch」ウィンドウの表示フォーマット」を参照）。

注：展開された式は編集できません

変数が展開済みの式である場合、または変数が展開済み変数の要素である場合は、「Variable」フィールドを編集することはできません。その変数を変更する場合は、まずその変数を縮小してから、編集します。

8.3 「Watch」ウィンドウの表示フォーマット

フォーマット記号を使用すると、「Watch」ウィンドウ内の変数の表示フォーマットを変更することができます。デフォルトの表示フォーマットは、表示される変数のタイプに応じて異なります。そのフォーマットを変更するには、次のように、変数の後にコンマおよびフォーマット記号を入力します。

```
myVar,x = 0x1234
```

記号	フォーマット
d	10 進数
e	指数浮動小数点
f	10 進数浮動小数点
x	16 進数
o	8 進数
u	符号なし整数
c	ASCII 文字 (バイト)
p	ビッグ・エンディアン・フォーマットを使用するパック済み ASCII 文字。最初の文字は、ターゲットの最大重みバイト (MS Byte) 内にあります。
P	リトル・エンディアン・フォーマットを使用するパック済み ASCII 文字。最初の文字は、ターゲットの最小重みバイト (LS Byte) 内にあります。

注 : P および p の表示文字列

p フォーマット、および P フォーマットは、ターゲット上の文字列を表示します。この変数は、ターゲット上の文字列の先頭文字を指す文字ポインタでなければなりません。

8.4 Quick Watch

Code Composer Studio デバッガの Quick Watch 機能を使用すると、変数の表示または変更、もしくは「Watch」ウィンドウ への項目の追加を簡単に実行することができます。「Quick Watch」ダイアログ・ボックスは、「Watch」ウィンドウとほぼ同じです。したがって、機能の多くも非常に似ています。変数を変更するには、その変数をダブルクリックします。式の展開または縮小を行うには、その式が選択されていることを確認してから、Enter キーを押します。

Quick Watch を使用して変数を表示する方法

1) 「Edit」ウィンドウで、調べたい変数の上にカーソルを置きます。
2) その変数の上で右クリックし、コンテキスト・メニューから 「Quick Watch」を選択します。「Quick Watch」ダイアログ・ボックスが表示されます。

「Debug」ツールバー上で、「Quick Watch」ボタンを使用することもできます。

Quick Watch: 

第 9 章

組み込みエディタ

本章では、Code Composer Studio で使用できるソース・プログラムを編集するための機能について説明します。

項目 **ページ**

9.1 機能の概要

Code Composer Studio は、次の編集機能を備えています。

構文の強調表示

言語キーワード、コメント、文字列、およびアセンブラ疑似命令を、異なる色で強調表示します。

検索と置換

文字列の検索と置換を行います。この機能は、「Standard」ツールバーから起動することができます。

文脈依存ヘルプの表示

強調表示されたワードについてのヘルプを検索します。これは、アセンブリ命令または GEL 組み込み関数についてのヘルプを表示する場合に便利です。

複数ウィンドウ

複数のファイル、または同一ファイルの複数のビューを開きます。

ウィンドウの分割

Code Composer Studio 環境内の「Edit」ウィンドウを分割します (*.c、*.cmd、*.asm、*.h ファイル)。この機能を使用すると、1 つのアクティブ・ウィンドウ内で複数のコピーを作成することができます。ウィンドウを横に分割するには、スクロール・バーの最上部にある小さいバーをクリックして、下にドラッグします。ウィンドウを縦に分割するには、スクロール・バーの左にある小さいバーをクリックして、右にドラッグします。どちらの場合も、ウィンドウのコピーガ希望するサイズになるまで、その区画をドラッグします。

「Edit」ツールバー

拡張エディタ機能への高速アクセス

右ボタン・アクセス

拡張エディタ機能への簡単なアクセス。「Edit」ウィンドウ内の任意の位置で右クリックし、コンテキスト・メニューから機能を選択します。

9.1.1 「Standard」ツールバー

「Standard」ツールバーは、Code Composer Studio の始動時に、自動的に表示されます。「View」→「Standard Toolbar」の順に選択すると、「Standard」ツールバーのオン／オフを切り換えることができます。

「Standard」ツールバーでは、次のボタンを使用することができます。


New ― 新しいファイルを作成します。


Open ― 既存のファイルを開きます。


Save ― アクティブ・ウィンドウ内のファイルをセーブします。


Cut ― マークされたテキストを切り取って、クリップボードに入れます。


Copy ― マークされたテキストを、クリップボードにコピーします。


Paste ― クリップボードからカーソル位置に、テキストを貼り付けます。


Undo ― 直前の編集操作を取り消します。


Redo ― 直前の undo 操作を再実行します。


Find Next ― アクティブ・ウィンドウ内で、検索文字列の次の候補を検索します。


Find Previous ― アクティブ・ウィンドウ内で、検索文字列の直前の候補を検索します。


Search Word ― カーソルが置かれているワードを、検索テキストとして使用します。または、テキストの一部分が強調表示されている場合は、その部分を検索テキストとして使用します。このボタンをクリックすると、検索テキストの次の候補にウィンドウが移動します。


Find in Files ― 複数のファイルを検索して、指定された文字列を見つけます。


Print ― アクティブなソース・ファイルを印刷します。


Help ― このボタンをクリックした後、オブジェクトをクリックすると、文脈依存ヘルプが表示されます。

9.1.2 「Edit」ツールバー

「Edit」ツールバーは、Code Composer Studio の始動時に、自動的に表示されます。「View」→「Edit Toolbar」の順に選択すると、「Edit」ツールバーのオン／オフを切り換えることができます。

マルチプロセッシングを行っているとき、編集機能は、現在選択されている親ウィンドウ内のアクティブな子ウィンドウで作動します。この親ウィンドウの名前（マルチプロセッサ・システム内の CPU ごとに、1 つの親ウィンドウがあります）が、「Edit」ツールバーのタイトルに表示されます。

「Edit」ツールバーでは、次のボタンを使用することができます。


Mark To ― 括弧または中括弧の前にカーソルを置くと、対応する括弧を含めて、テキストをマーク付けします。


Mark Next ― 次の左括弧または左中括弧を検索します。これらの括弧を検出すると、対応する右括弧または右中括弧を検出し、囲まれたテキストをマーク付けします。このボタンをもう一度押すと、ネストされたブロックまで深く調べることができます。


Find Match ― 対応する括弧または中括弧に、カーソルを移動します。


Find Next Open ― 次の左括弧または左中括弧に、カーソルを移動します。


Outdent Marked Text ― 選択されたテキストのブロックを、タブ 1 つ分左に移動させます。


Indent Marked Text ― 選択されたテキストのブロックを、タブ 1 つ分右に移動させます。


Edit:Toggle Bookmark ― アクティブ・ドキュメント内の現在の行に、ブックマークを作成および削除します。


Edit:Next Bookmark ― アクティブ・ドキュメント内で、次のブックマークを検索します。


Edit:Previous Bookmark ― アクティブ・ドキュメント内で、直前のブックマークを検索します。


Edit Bookmarks ― 「Bookmark」プロパティ・ダイアログ・ボックスを開きます。

「Edit」ツールバーへのアクセスは、右クリックして、「Tools」→「Edit Toolbar」の順に選択してもできます。

9.2 キーボード・ショートカット

エディタ機能（およびその他の機能）に素早くアクセスするには、キーボードのショートカットを使用します。

表 9–1 デフォルトのキーボード・ショートカット

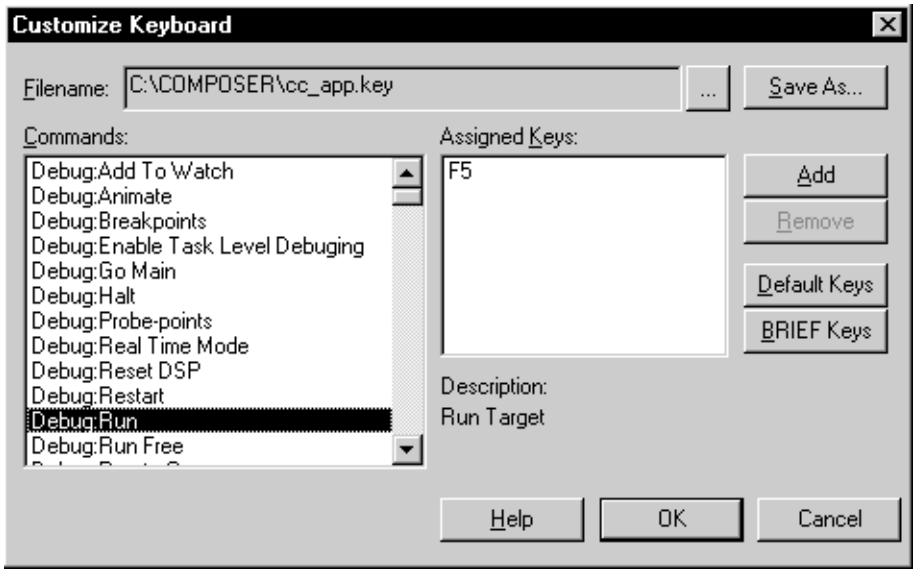
キーボード操作	目的	キー操作
ブックマークの管理	「Bookmarks」ダイアログ・ボックスを立ち上げる	Alt + F2
	ブックマークを切り換える	Ctrl + F2
	ブックマークを切り換えて編集する	Ctrl + Alt + F2
	ファイル内の次のブックマークに進む	F2
列編集	列編集モードを切り換える	Ctrl + Shift + F8
カーソルの移動	1 文字左に移動する	左矢印
	1 文字右に移動する	右矢印
	1 ワード左に移動する	Ctrl + 左矢印
	1 ワード右に移動する	Ctrl + 右矢印
	1 行上に移動する	上矢印
	1 行下に移動する	下矢印
	現在の行の最初のインデントに移動する	Home
	現在の行の先頭に移動する	Home, Home
	行の末尾に移動する	End
	ファイルの先頭に移動する	Ctrl + Home
	ファイルの終わりに移動する	Ctrl + End
削除、挿入、コピー	左側の 1 文字を削除する	Backspace
	右側の 1 文字を削除する	Delete
	選択したテキストを削除し、クリップボードにコピーする	Ctrl + X、Shift + Delete

キーボード操作	目的	キー操作
	キーボードの挿入モードのオン／オフを切り換える	Insert
	選択したテキストをクリップボードにコピーし、そのテキストを保持する	Ctrl + C、Ctrl + Insert
	選択したテキストをクリップボードにコピーし、そのテキストを削除する	Ctrl + X、Shift + Delete
	クリップボードの内容を挿入する	Ctrl + V、Shift + Insert
	直前の編集操作を取り消す	Ctrl + Z
タブ	選択されている複数行を、タブ 1 つ分右に移動する	Tab
	選択されている複数行を、タブ 1 つ分左に移動する	Shift + Tab
テキストのスクロール	一度に 1 ページずつ、スクロールアップする	Page Up
	一度に 1 ページずつ、スクロールダウンする	Page Down
テキストの選択	左側の文字を選択する	Shift + 左矢印
	右側の文字を選択する	Shift + 右矢印
	左側の 1 ワードを選択する	Shift + Ctrl + 左矢印
	右側の 1 ワードを選択する	Shift + Ctrl + 右矢印
	カーソルがホームにある場合、現在の行を選択する	Shift + 下矢印
	カーソルがホームにある場合、上の行を選択する	Shift + 上矢印
	行の末尾まで選択する	Shift + End
	行の先頭まで選択する	Shift + Home
	1 画面上を選択する	Shift + Page Up
	1 画面下を選択する	Shift + Page Down
ウィンドウの管理	次のウィンドウに切り換える	Ctrl + F6
	直前のウィンドウに切り換える	Shift + Ctrl + F6

キーボード操作	目的	キー操作
	直前のアクティブ・ウィンドウに切り換える	Ctrl + Tab
	アクティブ・ウィンドウをクローズする	Ctrl + F4

9.2.1 キーボード・ショートカットをカスタマイズする方法

編集コマンド用のキーボード・ショートカットだけでなく、Code Composer Studio 内のすべてのメニュー・コマンド用のキーボード・ショートカットも、カスタマイズすることができます。キーボード・ショートカットの割り当てを行う 「Customize Keyboard」ダイアログを開くには、「Option」→「Keyboard」の順に選択します。



「Commands」ウィンドウから、カスタマイズしたいコマンドを選択します。「Assigned Keys」ウィンドウに、現在のキーボード・ショートカットが表示されます。

選択されたコマンドを起動させるための新しいキー・シーケンスを割り当てるには、「Add」ボタンをクリックします。「Assign Shortcut 」ダイアログ・ボックスが表示されます。このダイアログ・ボックスで、新しいキー・シーケンスを入力してから、「OK」を押します。

コマンドの特定のキー・シーケンスを削除するには、「Assigned Keys」ウィンドウでそのキー・シーケンスを選択し、「Remove」ボタンをクリックします。

「Save As」オプションを使用すると、キーボード設定をファイルに保存することががができます。このボタンを押すと、「ファイル名を付けて保存」ダイアログ・ボックスが表示されます。このダイアログ・ボックスで、設定を保存したい位置までナビゲートできます。

以前に保存されたコンフィギュレーション・ファイルまでナビゲートし、それをロードするには、「Browse」ボタン (...) を使用します。

「BRIEF Keys」ボタンをクリックすると、直ちに brief エディタのショートカット・キーに切り換えることができます。

9.3 ファイル操作

以下の節では、ソース・ファイルに対して実行できる操作について説明します。

9.3.1 新規ファイルの作成

新規ソース・ファイルは、次の手順で作成します。ソース・ファイルを作成しても、既存のソース・ファイルには影響を与えません。

1) 「File」→「New」→「Source File」の順に選択します。新しい 「Edit」ウィンドウが開きます。「Standard」ツールバー上の 「New」ボタンを使用することもできます。

New: 

2) この新しいウィンドウに、ソース・コードを入力します。「Edit」 ウィンドウのタイトル・バーで、そのファイル名の隣にアスタリスク (*) が表示されていることに注意してください。このアスタリスクは、そのソース・ファイルが変更されていることを示します。そのファイルが保管されると、このアスタリスクが消えます。

3) 「File」→「Save」、または 「File」→「Save As」の順に選択します。「名前を付けて保存」ダイアログ・ボックスが表示されます。「Standard」ツールバー上の 「Save」ボタンを使用することもできます。

Save: 

4) 「名前を付けて保存」ダイアログ・ボックスのメイン・ウィンドウで、ソース・ファイルを保存するディレクトリをダブルクリックします。希望するディレクトリが表示されていない場合は、正しいディレクトリまでナビゲートします。

5) ファイル名が、「ファイル名 (N)」フィールドに表示されます。ファイルの拡張子を変更する場合は、別の拡張子を入力するか、「ファイルの種類 (T) 」フィールドで別の拡張子を選択します。

6) 「保存 (S)」をクリックします。

新規ファイルには、保存されるまでの間、Untitled という名前が付けられます。ウィンドウをセーブするか、クローズする前に、そのウィンドウをアクティブにしておく必要があります。ウィンドウをアクティブにするには、そのウィンドウ内の任意の位置でクリックするか、「Window」→「Untitled」の順に選択します。

9.3.2 ファイルを開く

Open コマンドを使用すると、既存のソース・ファイルが開きます。任意のエディタで作成されたどの ASCII ファイルでも、開くことができます。

ファイルを開く方法

1) 「File」→「Open」の順に選択します。「開く」ダイアログ・ボックスが表示されます。「Standard」ツールバー上の 「Open」ボタンを使用することもできます。

   Open: 

2) 「開く」ダイアログ・ボックスのメイン・ウィンドウで、開きたいファイルをダブルクリックします。希望するファイルが表示されていない場合は、正しいディレクトリまでナビゲートし、そのファイルをダブルクリックします。
3) ファイル名が、「ファイル名 (N)」フィールドに表示されます。ファイルの拡張子を変更する場合は、別の拡張子を入力するか、「ファイルの種類 (F)」フィールドで別の拡張子を選択します。
4) 「開く (O)」をクリックします。

9.3.3 ファイル・ビューの重複

同一ファイルの複数ビューを表示するには、「Window」→「New Window」の順に選択します。1 つのファイルの複数コピーが表示されると、タイトル・バーに *filename:*<*n*> と表示されます。この場合の「n」は、固有のウィンドウ番号です。1 つのウィンドウ内で加えられた変更は、すべてその他のウィンドウに反映されます。

9.3.4 ファイルの保存

Save コマンドを使用すると、タイトル・バーに表示されている名前を使用してファイルが保存されます。

ファイルを保存する方法

1) 「Edit」ウィンドウをクリックして、そのファイルをアクティブにします。「File」→「Save」の順に選択するか、「Standard」ツールバー上の 「Save」ボタンをクリックします。

   Save: 

2) ファイルに名前が指定されていない場合は、「名前を付けて保存」ダイアログ・ボックスが表示されます。「ファイル名 (F)」フィールドに、使用したい名前を入力します。
3) ファイルを、保存先のドライブとディレクトリに移動します。
4) ファイルの拡張子を変更したい場合は、別の拡張子を入力するか、「ファイルの種類 (T)」フィールド・ボックスで別の拡張子を選択します。
5) 「保存 (S)」をクリックします。

ファイル名、またはファイル拡張子を変更する方法

1) 「Edit」ウィンドウをクリックして、そのファイルをアクティブにします。「File」→「Save As」の順に選択すると、「名前を付けて保存」ダイアログ・ボックスが表示されます。
2) 「ファイル名 (N)」フィールドに、使用する名前を入力します。
3) ファイルを、保存先のドライブとディレクトリに移動します。
4) ファイルの拡張子を変更する場合は、別の拡張子を入力するか、「ファイル名の種類 (T)」フィールド・ボックスで別の拡張子を選択します。
5) 「保存 (S)」をクリックします。

開いているファイルをすべて保存する方法

「File」→「Save All」の順に選択します。

9.3.5 ファイルの印刷

Print コマンドを使用すると、ソース・ファイルを印刷することができます。

ファイルを印刷する方法

1) 「Edit」ウィンドウをクリックして、そのファイルをアクティブにします。「File」→「Print」の順に選択するか、「Standard」ツールバー上の 「Print」ボタンをクリックします。

   Print: 

2) 「印刷」ダイアログ・ボックスの「 プリンタ名 (N)」ドロップダウン・リストで、使用するプリンタを選択します。
3) 「印刷範囲」エリアで、使用するページ範囲を指定します。
4) 「OK」をクリックします。

9.3.6 テキストのカット、コピー、ペースト

選択したテキストをアクティブ・ウィンドウから削除し、そのテキストをクリップボードにコピーするには、「Edit」→「Cut」コマンドを使用します。選択したテキストをアクティブ・ウィンドウからクリップボードにコピーするには、「Edit」→「Copy」コマンドを使用します。クリップボードからテキストを取り出して挿入するには、「Edit」→「Paste」コマンドを使用します。

テキストをカット、コピー、およびペーストする方法

1) カットまたはコピーを行いたいテキストを強調表示します。
2) 「Edit」→「Cut」、または 「Edit」→「Copy」を選択します。「Standard」ツールバー上の 「Cut」ボタンと 「Copy」ボタンを使用することもできます。

Cut: 

Copy: 

3) そのテキストを挿入したい任意の 「Edit」ウィンドウに、カーソルを移動させます。
4) 「Edit」→「Paste」の順に選択するか、「Standard」ツールバー上の「Paste」ボタンを使用します。

Paste: 

9.3.7 テキストの削除

「Edit」→「Delete」の順に選択すると、強調表示されたテキストを、クリップボードにコピーせずに削除することができます。このテキストは、別の位置に貼り付けることができません。また、キーボード上の Delete キーを使用して削除することもできます。

9.3.8 列の編集

行全体ではなく、テキストの列を選択、カット、ペーストすることができます。

「Edit」ウィンドウをクリックして、そのファイルをアクティブにします。「Edit」→「Column Editing」の順に選択するか、次のキーボード・シーケンスを押して、列モードに入ります。

Ctrl + Shift + F8

マウスの左ボタンを押したまま、選択したい列にカーソルを移動し、左ボタンを離すことによって列エリアを選択します。また、シフト・キーを押したまま、矢印キーを使用してカーソルを移動させても、列を選択できます。

必要に応じて、選択された列のカット、コピー、ペースト、および削除を行うことができます（9.3.6 節 「テキストのカット、コピー、ペースト」を参照）。

9.3.9 操作の取り消しと再実行

アクティブ・ウィンドウ内で直前の編集操作を元に戻すには、「Edit」→「Undo」コマンドを実行します。再実行するには、「Edit」→「Redo」コマンドを実行します。

取り消し方法

「Edit」→「Undo」の順に選択します。キーボード・ショートカット (Ctrl + Z) を使用するか、「Standard」ツールバー上の 「Undo」ボタンを押しても、取り消すことができます。

Undo: 

再実行方法

「Edit」→「Redo」の順に選択します。キーボード・ショートカット (Ctrl + A) を使用するか、「Standard」ツールバー上の 「Redo」ボタンを押しても、再実行することができます。

Redo: 

9.3.10 複数行にタブ設定する方法

1 つのグループの行のインデントを変更するには、その部分全体を選択し、キーボードの Tab キーを押してインデントするか、Shift + Tab キーを押してインデントを戻します。また、「Edit」ツールバーの 「Outdent Marked Text」ボタンと「Indent Marked Text」ボタンを使用することもできます。

Outdent Marked Text: 

Indent Marked Text: 

9.3.11 ソース行へ移動 (GO TO) する方法

Go To コマンドを使用すると、ソース・ファイル内の特定の行またはブックマークに素早く移動することができます。

特定の行、またはブックマークへ移動 (Go To) する方法

1) 「Edit」→「Go To」の順に選択します。「Go To」ダイアログ・ボックスが表示されます。または「Edit」ウィンドウ内で右クリックし、コンテキスト・メニューから Go To を選択します。
2) 表示したい行またはブックマークを指定します。
3) 「OK」をクリックします。

9.3.12 フォントの変更

「Option」→「Font」コマンドを使用すると、テキストのフォントとサイズを変更することができます。

フォントとサイズを変更する方法

1) 「Option」→「Font」の順に選択します。「フォントの指定」ダイアログ・ボックスが表示されます。
2) 使用するフォント名、スタイル、およびサイズを選択します。
3) 「OK」をクリックします。

9.4 テキストの検索と置換

Code Composer Studio では、現在のファイル、または複数のファイルを検索して、文字列を見つけることができます。また、ある文字列を別の文字列に置換することもできます。

9.4.1 現在のファイル内のテキストを検索する方法

アクティブ・ウィンドウを素早く検索して文字列を見つけるには、「Standard」ツールバー内の「Find」フィールドを使用します。

文字列を検索する方法

1) 「Find」フィールドに検索文字列を入力します。このフィールドは、「Standard」ツールバーにあります。この「Find」フィールドは、スクロール可能なリストであり、検索文字列の履歴が入っています。そのリストをスクロールして、直前の検索パラメータを見つけることができます。
2) 検索の方向に応じて、「Standard」ツールバー上の「Find Next」ボタン、または「Find Previous」ボタンをクリックして、検索を開始します。

Find Next: 

Find Previous: 

または、「Edit」→「Find/Replace」コマンドを使用して、文字列を検索することもできます。

Find/Replace コマンドを使用して、文字列を検索する方法

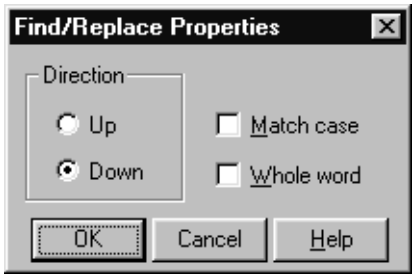
1) 検索を開始したい位置にカーソルを移動させます。
2) 「Edit」→「Find/Replace」の順に選択します。「Find and Replace」ダイアログ・ボックスが表示されます。

3) 「Find」フィールドに検索テキストを入力します。
4) 「Find next」をクリックします。

9.4.2 Find/Replace プロパティの設定

「Find/ReplaceProperties」ダイアログでオプションを設定すると、検索方法を制御することができます。

1) 「Find and Replace」ダイアログ・ボックスで、「Properties」をクリックします。「Find/Replace Properties」ダイアログ・ボックスが表示されます。



2) オプションを選択して、次の項目を指定します。

Direction ― 検索の方向を指定します。

Match case ― 大文字と小文字を区別して、その文字列と一致するテキストを検索します。

Whole word ― 英数字または下線が前後に付いていない文字列とだけ一致します。

3) 「OK」をクリックします。

9.4.3 テキストの検索と置換

Find/Replace コマンドを使用すると、ファイル内のテキストの検索に加えて、文字列を検索して、それを別の文字列に置換することもできます。

テキストを検索する方法と置換する方法

1) 検索を開始したい位置にカーソルを移動させます。
2) 「Edit」→「Find/Replace」の順に選択します。「Find and Replace」ダイアログ・ボックスが表示されます。
3) 「Find」フィールドに検索テキストを入力します。
4) そのテキストが置換される新しいテキストを、「Replace」フィールドに入力します。
5) 「Find next」をクリックします。
6) そのテキストが見つかると、「Replace」をクリックして、選択したテキストを置換します。

   または、「Replace All」をクリックして、選択したテキストに一致するすべてのテキストを置換します。

9.4.4 複数ファイル内のテキストの検索

「Edit」→「Find in Files」コマンドを使用すると、複数のテキスト・ファイルを検索して、特定の文字列を見つけることができます。

文字列を検索する方法

1) 「Edit」→「Find in Files」の順に選択します。「Standard」ツールバー上の 「Find in Files」ボタンを使用することもできます。

   Find in Files: 

2) 「Find in Files」ダイアログ・ボックスで、次の情報を指定します。

   Find What ― 文字列を入力します。直前の検索文字列のリストの中から選択するには、ドロップダウン・リストを使用します。

   Files of Type ― 検索したいファイル・タイプを選択します。共通ファイル・タイプのドロップダウン・リストの中から選択するか、ファイル・タイプを指定するテキストを入力します。

   In Folder ― 検索したい基本のフォルダを選択します。ドライブとフォルダを指定するテキストを入力するか、「Browse」ボタン (...) を使用してフォルダを選択します。

   Look in subfolders ― 指定された基本のフォルダのサブフォルダを検索します。

   Match case ― 大文字と小文字を区別して、その文字列と一致するテキストを検索します。

   Match Whole word only ― 英数字または下線 (_) が前後に付いていない文字列とだけ一致します。

   Look In Project Files ― 特定プロジェクト内で検索します。

3) 「Find」をクリックして、検索を開始します。

「Output」ウィンドウに検索の結果が表示されます。一致するごとに、「Output」ウィンドウには完全修飾ファイル名が表示されます。そしてその後に、一致した行の行番号とテキストが続きます。

「Output」ウィンドウ内で一致をダブルクリックすると、指定されたファイルが「Edit」ウィドウ内で開きます。「Edit」ウィンドウ内のカーソルは、一致が検出された行の先頭に置かれます。

「Output」ウィンドウをクローズするには、そのウィンドウ内で右クリックしてコンテキスト・メニューを表示し、「Hide」を選択します。

現在のセッション時に、前回の複数ファイル検索の出力を表示するには、「View」→「Output」の順に選択してから、「Output」ウィンドウで「Find in Files」タブを選択します。

9.5 エディタ・プロパティの設定

Code Composer Studio では、ユーザが頻繁に使用するエディタ・オプションをカスタマイズすることができます。

エディタ・プロパティを設定する方法

1) 「Option」→「Editor Properties」の順に選択します。「Editor Properties」ダイアログ・ボックスが表示されます。

2) 「Editor Properties」ダイアログには、次のオプションが表示されます。

Tab Stops

「Tab」Stops ボックスに、希望するタブ・ストップ数を入力します。デフォルトは 4 です。

Open files as read only

開いたウィンドウに意図しない修正を加えるのを防止するには、「Open files as read only」チェック・ボックスにチェック・マークを付けます。このオプションをオフに切り換えると、開いた任意のファイルに修正を加えることができます。

Save before running tools

「Save before running tools」チェック・ボックスにチェック・マークを付けないと、「Code Composer」ダイアログ・ボックスが表示され、ファイルを保存するように求められます。ファイルの保存が求められるのは、現在開いているプロジェクト・ファイルのいずれかに変更を加えた後で、Project Build を起動した場合です。

Recent Files

「File」→「Recent Source Files」、「Project」→「Recent Project Files」などのメニュー項目に表示される、最近使用したファイルの数を指定します。

3) 「OK」をクリックします。

9.6 ブックマークの使用

ブックマークを設定すると、ソース・ファイル内の重要な位置を検索し、保持することができます。ブックマークは、任意のファイルの任意の行に設定することができます。設定されたブックマークは、Code Composer Studio ワークスペースと一緒に保存されるので、いつでも再呼び出しをすることができます。

「Edit」ウィンドウからブックマークを設定する方法

「Edit」ウィンドウでソース・コードを編集しているときにブックマークを設定する手順は、次のとおりです。

1) ブックマークを指定する行にカーソルを置きます。
2) 「Edit」ウィンドウ内で右クリックし、コンテキスト・メニューから「Bookmarks」を選択します。「Bookmarks」サブメニューから、「Set a Bookmark」を選択します。

   または

   「Edit」ツールバー上の 「Edit:Toggle Bookmark」ボタンをクリックします。

Edit:Toggle Bookmark: 

ブックマークが設定された行が、「Edit」ウィンドウ内で強調表示されていることに注意してください。

あるブックマークから別のブックマークに素早く進むには、「Edit」ツールバー上の「Edit:Next Bookmark」ボタンと 「Edit:Previous Bookmark」ボタンを使用します。

Edit:Next Bookmark: 

Edit:Previous Bookmark: 

ブックマークのリストを表示する方法

現在設定されているブックマークのリストを表示するには、次の方法のいずれかを使用します。

- ❑ 「Project View」ウィンドウの 「Bookmarks」タブを選択します。

  リスト内の任意のブックマークをクリックすると、そのブックマークが含まれているファイルが開き、そのブックマークの位置にカーソルが置かれます。
- ❑ 「Edit」→「Bookmarks」コマンドを選択して、「Bookmarks」ダイアログ・ボックスを開きます。
- ❑ 「Edit」ツールバー上の「Edit:Bookmarks」ボタンをクリックして、「Bookmarks」ダイアログ・ボックスを開きます。

Edit:Bookmarks: 

9.6.1 ブックマークの管理

すべてのブックマークを管理するには、「Bookmarks」ダイアログを使用します。

「Bookmarks」ダイアログには、現在使用可能なブックマークの完全なリストが表示されます。そのリストからブックマークを選択するには、必要なブックマークをクリックするだけです。

「Bookmarks」ダイアログには、次のオプションがあります。

Go To ― リストからブックマークを選択した後、「Go To」を選択すると、そのブックマークが入っているファイルが開き（まだ開いていない場合)、そ のブックマークの位置にカーソルが置かれます。

Close ― 「Bookmarks」ダイアログ・ボックスをクローズします。

Help ― 「Bookmarks」ダイアログ・ボックスの使用方法についてのヘルプを表示します。

Edit ― リストからブックマークを選択した後、「Edit」を選択すると、「Bookmark Properties」ダイアログ・ボックスが開きます。

Add ― 「Bookmark Properties」ダイアログ・ボックスを開きます。

Remove ― リストからブックマークを選択した後、「Remove」を選択すると、そのブックマークは現在の位置から削除されます。

9.6.2 ブックマーク・プロパティの編集

新規ブックマークの追加、または既存のブックマークの編集を行うには、「Bookmark Properties」ダイアログ・ボックスを使用します。

次の情報を指定します。

File ― 「Bookmarks」ダイアログで 「Add」を選択した後、「Browse」ボタンがアクティブになります。この 「Browse」ボタンを押すと、「ファイルを開く」ダイアログ・ボックスが表示されます。そのブックマークが入っているファイルを選択し、「開く (O)」ボタンをクリックします。

Line ― ブックマークを設定する行番号を指定します。

Description ― ブックマークの分かりやすい説明を入力します。この説明は、Bookmarks のリストに表示されます。

「OK」をクリックして、設定したパラメータを受け入れます。

第 10 章

プロジェクト環境

Code Composer Studio は、プロジェクトを使用する統合型のプログラム管理機構を備えています。プロジェクトは、ターゲット・プログラム、またはターゲット・ライブラリの構築に必要な、あらゆる情報を保持します。

プロジェクトは、次のものを記録します。

- ❏ ソース・コードおよびオブジェクト・ライブラリのファイル名
- ❏ コンパイラ・オプション、アセンブラ・オプション、およびリンカ・オプション
- ❏ インクルード・ファイルの依存関係

プログラム管理を最も簡単に実行するには、「Project View」ウィンドウを使用します。「Project View」ウィンドウには、プロジェクト全体の内容が表示されます。そこでは、プロジェクトに関連付けられたファイルは、タイプによって分離され、組織付けられています。すべてのプロジェクト操作は、「Project View」ウィンドウの中で実行することができます。

このプロジェクト環境では、プロジェクトのビルド用にさまざまなコマンドが提供されているので、開発時間が短縮されます。プロジェクトに多くのソース・ファイルが含まれているが、前回プロジェクトをビルドしてからは数個のファイルしか編集していないような場合には、「Incremental Build」コマンドを使用して、変更したファイルだけを再コンパイルします。「Rebuild All」コマンドを使用すると、すべてのファイルが強制的にコンパイルされます。個々のソース・ファイルをコンパイルするには、「Compile File」コマンドを使用します。

項目 **ページ**

10.1 プロジェクトの作成、オープン、およびクローズ

各プロジェクトの情報は、1 つのプロジェクト・ファイル (*.mak) に保存されます。プロジェクト・ファイルの作成、オープン、およびクローズを行う手順は、次のとおりです。

新規プロジェクトを作成する方法

1) 「Project」→「New」の順に選択します。使用したいプロジェクト・ディレクトリが、「Save New Project As」ダイアログ・ボックスに表示されていない場合は、正しいディレクトリをブラウズします。コンパイラとアセンブラによって生成されたオブジェクト・ファイルの保存だけでなく、プロジェクト・ファイルの保存にも、このディレクトリを使用します。新規プロジェクトごとに異なるディレクトリを使用することをお勧めします。異なるディレクトリを使用すると、異なるプロジェクトのオブジェクト・ファイルが別々に保存されるので、プロジェクトごとに、コンパイラ、アセンブラ、およびリンカに異なるオプションを指示することができます。

2) 「ファイル名　(N)」フィールドで、新規プロジェクトのファイル名を入力し、「保存 (S)」をクリックします。新しいプロジェクト・ファイルが、空のプロジェクト・リストと一緒に作成されます。既存のプロジェクトがすでに開いている場合は、そのコンパイラ・オプション、アセンブラ・オプション、およびリンカ・オプションがその新規プロジェクトにコピーされ、既存のプロジェクトは自動的にクローズします。開いているプロジェクトがない場合、その新規プロジェクトは、デフォルトのプロジェクト・オプションを継承します。Code Composer Studio のタイトル・バーが変わり、新規プロジェクトの名前が表示されます。

新規プロジェクト・ファイルを作成した後、ソース・コード、オブジェクト・ライブラリ、およびリンカ・コマンド・ファイルのファイル名を、そのプロジェクト・リストに追加します。詳細は、10.3 節 「プロジェクトにファイルを追加する方法」を参照してください。

既存プロジェクトを開く方法

1) 「Project」→「Open」の順に選択します。プロジェクト・ファイルの存在するディレクトリが 「Project Open」ダイアログ・ボックスで選択されていない場合は、正しいディレクトリをブラウズします。

2) 「Project Open」ダイアログ・ボックスで、使用するプロジェクト・ファイルを強調表示し、「開く (O)」をクリックします。既存のプロジェクトがすでに開いている場合、そのプロジェクトは自動的にクローズします。プロジェクト・ファイルが正常にロードされると、Code Composer Studio のタイトル・バーが変わり、新規プロジェクトの名前が表示されます。ファイルがロードされない場合は、ファイルが壊れていることを知らせるエラー・メッセージが表示されます。正しいプロジェクト・ファイルを選択したかどうかを確認してください。ファイルが壊れている場合は、新規プロジェクトを最初から作り直す必要があります。旧バージョンの Code Composer からアップグレードした場合は、プロジェクト・ファイルが旧フォーマットであることを知らせる警告メッセージが表示される場合があります。この場合、「OK」ボタンをクリックして、プロジェクト・ファイルを新しいフォーマットに変換すると、データを失うことはありません。プロジェクト・ファイルを変換しないと、ファイルを開くことができません。

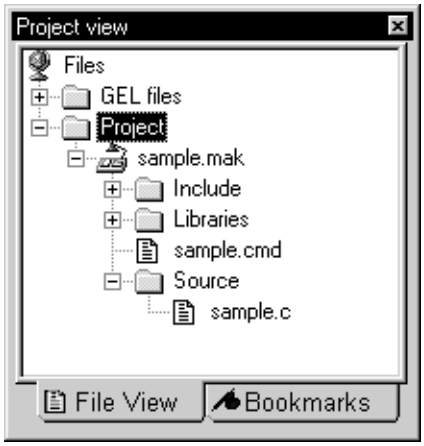
プロジェクトをクローズする方法

プロジェクトをクローズするには、次の操作のいずれかを実行します。

- ❏ 「Project」→「Close」の順に選択します。
- ❏ 新規プロジェクトを作成します。
- ❏ 別のプロジェクトを開きます。

10.2「Project View 」ウィンドウの使用

「Project View」ウィンドウは、プロジェクトの内容をグラフ表示します。



Code Composer Studio が始動すると、「Project View」が自動的に開きます。「Project View」が表示されない場合は、「View」→「Project」の順に選択します。「Bookmarks」アイコンが選択されている場合は、「Project View」ウィンドウの下部にある「File View」アイコンをクリックします。

プロジェクト内のファイルは、次のように、別々のフォルダにグループ分けされます。

Include ― すべてのヘッダ／インクルード・ファイルを含む : *.h

Libraries ― すべてのライブラリ・ファイルを含む : *.lib

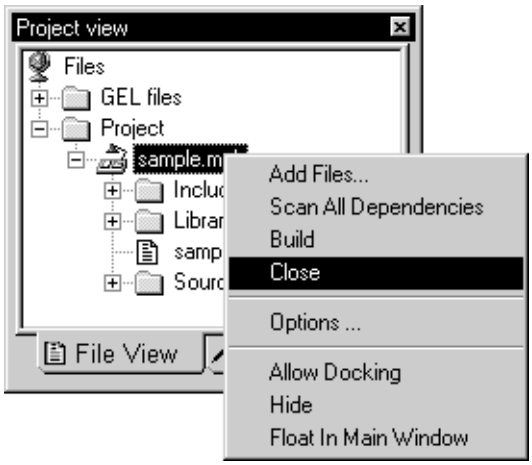
Source ― すべてのソース・ファイルを含む : *.c、*.asm

リンカ・コマンド・ファイル (*.cmd) は、プロジェクト・ファイル (*.mak) の真下に表示されます。

Project リストを展開、または縮小するには、Project フォルダ、プロジェクト名 (*.mak)、および Include、Libraries、Source の各フォルダの横にある + または ― 符号をクリックします。

10.2.1 Project View のコンテキスト・メニューを使用する方法

コンテキスト・メニューを使用すると、「Project View」ウィンドウ内で、さまざまな操作を実行することができます。たとえば、プロジェクトを素早くクローズするには、そのプロジェクト名を右クリックし、「Close」を選択します。



その他のショートカットは、次のとおりです。

- ❑ 既存のプロジェクトを開くには、「Project」を右クリックし、「Open project」を選択します。
- ❑ プロジェクトにファイルを追加するには、そのプロジェクト名を右クリックし、「Add Files」を選択します。
- ❑ ファイルを編集するには、そのファイル名を右クリックし、「Open」を選択します。
- ❑ 個々のソース・ファイルに対してコンパイラ・オプションを設定するには、そのファイル名を右クリックし、「File Specific Options」を選択します。
- ❑ 1 つのファイルをコンパイルするには、そのファイル名を右クリックし、「Compile file」を選択します。
- ❑ プロジェクト全体を構築するには、そのプロジェクト名を右クリックし、「Build」を選択します。
- ❑ プロジェクトからファイルを削除するには、そのファイル名を右クリックし、「Remove from project」を選択します。

10.2.2 ドラッグ・アンド・ドロップ機能

Code Composer Studio は、Windows 95/98/NT 用のドラッグ・アンド・ドロップ機能をサポートします。エクスプローラから、「Project View」ウィンドウにファイルを直接ドラッグすると、任意のプロジェクト (*.mak) ファイル、またはソース (*.c、*.asm、*.cmd) ファイルをロードすることができます。

10.3 プロジェクトにファイルを追加する方法

ソース・コード・ファイル、オブジェクト・ファイル、ライブラリ・ファイル、およびリンカ・コマンド・ファイルを、プロジェクトに追加したり、プロジェクト・リストから削除したりするには、次の手順を使用します。

プロジェクトにファイルを追加する方法

1) 「Project」→「Add Files to Project」の順に選択します。「Add Files to Project」ダイアログ・ボックスが表示されます。

   または

   「Project View」ウィンドウ内でプロジェクト名を右クリックし、「Add Files」を選択します。

2) 「Add Files to Project」ダイアログ・ボックスで、追加するファイルを指定します。そのファイルが現在のディレクトリに存在しない場合は、正しい位置をブラウズします。「ファイル名 (F)」フィールドに表示されているファイルのタイプを設定するには、「ファイルの種類 (T)」ドロップダウン・リストを使用します。

注 : ヘッダ・ファイルやインクルード・ファイルを指定しないでください
ヘッダ・ファイル、またはインクルード・ファイル (*.h) を、手作業でプロジェクトに追加しようとしないでください。ビルド・プロセスの過程でソース・ファイルをスキャンして依存関係を見つけるときに、これらのファイルは自動的に追加されます。

3) 「開く (O)」をクリックして、指定されたファイルをプロジェクトに追加します。

ファイルがプロジェクトに追加されると、「Project View」は自動的に更新されます。

プロジェクトからファイルを削除する方法

1) 「Project View」が表示されない場合は、「View」→「Project」の順に選択します。
2) 必要に応じて、Project リストを展開します。
3) 削除するファイル名を右クリックします。
4) コンテキスト・メニューから、「Remove from Project」を選択します。

10.3.1 ファイル拡張子

ファイルは、ファイル拡張子で識別されます。次の表は、ファイル拡張子に基づく前提事項を掲載しています。

拡張子	前提事項
. または .c*	C ソース・ファイル。コンパイルとリンクが行われます。
.a* または .s*	アセンブリ・ソース・ファイル。アセンブルとリンクが行われます。
.o* または .lib	オブジェクト・ファイル、またはライブラリ・ファイル。リンク時にのみ使用されます。
.cmd	リンカ・コマンド・ファイル。リンク時にのみ使用されます。リンカ・コマンド・ファイルの詳細は、*Code Generation Tools* オンライン・ヘルプを参照してください。
その他	認識できないファイル。認識されないファイルは、プロジェクトに追加されません。

リンカ・コマンド・ファイルは、1 つのプロジェクトに対して、1 つしか指定できません。それ以外のファイルには、1 つのプロジェクトに追加できるファイル数に制限はありません。

プロジェクトに追加されたすべてのファイルは、絶対パス名と一緒に表示されます。しかし、これらのファイルは相対パス名と一緒に保存されるので、そのプロジェクトを異なるディレクトリに簡単に移動させることができます。プロジェクトが開くたびに、絶対パス名が判別されます。パス名は、プロジェクトの make ファイルに関連させて保存されます。たとえば、ご使用の make ファイルがパス c:\version1\linker\make\ 内にあり、ソース・ファイルがパス c:\version1\source\ 内にある場合、そのソース・ファイルへの相対パスは ..\..\source\ になります。各 ..\ は、1 つのディレクトリ・レベルをバックアップすることを示します。ユーザのソース・ファイルが別のドライブに保存されている場合は、相対パスが存在しないので、そのソース・ファイルは絶対パスと一緒に保存されます。プロジェクトの make ファイルを c:\version1\linker\make\ から c:\version2\linker\make\ に移動、またはコピーした場合、そのプロジェクトを開くときには、そのソース・ファイルは c:\version2\source に置かれているものと見なされます。make ファイルを移動するときには、すべての依存関係を再スキャンして、すべての参照が解決されることを確認してください。

10.4 依存関係のスキャン

プロジェクトは、インクリメンタル・コンパイル時にどのファイルをコンパイルする必要があるかを判別するために、ソース・ファイルごとにインクルード・ファイルの依存関係リストを保持する必要があります。プロジェクトのビルド、「Project」→「Show Dependencies」の実行、または「Project」→「Scan All Dependencies」の実行を行うたびに、依存関係ツリーが作成されます。依存関係ツリーを作成するために、プロジェクト・リスト内のすべてのソース・ファイルは、#include、.include、および .copy 疑似命令を検索するために、再帰的にスキャンされます。そして、インクルードされた各ファイル名は、そのプロジェクト・リストに追加されます。インクルード・ファイルは自動的にプロジェクトに追加されるので、ユーザ自身でインクルード・ファイルを追加しないでください。

インクルード・ファイルの検索に使用される検索パスは、ソース・ファイルのタイプに基づいています。カレント・ディレクトリは、そのソース・ファイルのパスです。相対パスは、カレント・ディレクトリを基準にして解決されます。検索は、次の順序で実行されます。

C ソース・ファイルの場合

1) カレント・ディレクトリ
2) コンパイラ・オプション (-i) で指定されたインクルード・パスのリスト（左から右へ）
3) C_DIR 環境変数によって指定されたインクルード・パスのリスト（左から右へ）（下の注を参照）

アセンブリ・ソース・ファイルの場合

1) カレント・ディレクトリ
2) アセンブラ・オプション (-i) で指定されたインクルード・パスのリスト（左から右へ）
3) A_DIR 環境変数によって指定されたインクルード・パスのリスト（左から右へ）（下の注を参照）

注 : 環境変数

使用される環境変数は、ターゲット・プロセッサに応じて変わる場合があります（たとえば、TMS320C6x **は、標準名** C_DIR **と** A_DIR **に加えて、**C6X_C_DIR **と** C6X_A_DIR **も使用します）。使用する正確な変数名は、コード生成ツールによって定義されます。**

このプロジェクト環境は、インクリメンタルに依存関係をスキャンすることにより、時間を短縮しています。インクリメンタルな依存関係のスキャンでは、新規ファイル、または前回のスキャン以降に変更されたファイルだけを対象とします。ファイルへの変更の有無は、依存関係のスキャン間でのファイルのタイム・スタンプの比較によって判断されます。この変更には、旧バックアップ・バージョンによって置き換えられたファイルが含まれます。

インクルード・ファイルの依存関係を再検出する方法保存

インクルード・ファイルの依存関係を再検出するには、次の方法のいずれかを使用します。

- ❑ 「Project」→「Show Dependencies」の順に選択します。依存関係のインクリメンタルなスキャンが完了した後、プロジェクト全体の依存関係ツリーが表示されます。
- ❑ 「Project」→「Build」の順に選択します。依存関係のインクリメンタルなスキャンが完了した後、インクリメントなビルドが実行されます。
- ❑ 「Project」→「Scan All Dependencies」の順に選択します。

  または

  「Project View」内でプロジェクト名を右クリックし、コンテキスト・メニューから「Scan All Dependencies」を選択します。

  前回の依存関係スキャン以降にファイルが変更されたかどうかに関係なく、すべてのファイルの依存関係がスキャンされます。

インクルード・ファイルの依存関係を表示する方法

1) プロジェクトがまだ開いていない場合は、「Project」→「Open」の順に選択します。
2) 「Project Open」ダイアログ・ボックスで、表示したいプロジェクトのファイル名を選択します。そのプロジェクトがカレント・ディレクトリにない場合は、正しいディレクトリをブラウズします。
3) 「Project」→「Show Dependencies」の順に選択します。依存関係のインクリメンタルなスキャンが実行されます。このスキャンにより、依存関係ツリーは最新のものになります。

依存関係スキャンが行われると、必ず 「Dependencies」ステータス・ウィンドウが表示されます。「Dependencies」ステータス・ウィンドウ内にリストされているファイルのいずれかが、赤色で表示されている場合は、これらのファイルの依存関係が解決されていないことを示しています。インクリメンタルなビルドが実行されると、これらのファイルは再ビルドされます。

依存関係スキャンからファイルを除外する方法

特定のファイルに対する依存関係スキャンを行わないようにするには、排他ファイル exclude.dat を使用します。初期状態で exclude.dat には、変更される可能性が低いシステム・インクルード・ファイルのリストが含まれています。このファイルを編集して、その他のファイル（たとえば、決して変更されないヘッダ・ファイル）のスキャンを除外させたり、変更が必要なシステム・ファイルのスキャンを実施させたりできます。

10.5 ビルド・オプションの設定

プログラムのビルド時に使用されるコンパイラ・オプション、アセンブラ・オプション、およびリンカ・オプションを設定するには、次の手順を使用します。

プロジェクト内のすべてのファイルに適用される、一連のプロジェクト・レベルのオプションを定義します。次に、個々のソース・コード・ファイルにファイル固有のオプションを定義して、プログラムを最適化します。

プロジェクト・レベル・オプションを設定する方法

プロジェクト・レベルで設定されたオプションは、すべてのプロジェクト・ファイルに影響を与えます。

1) 「Project」→「Options」の順に選択します。

   または

   「Project View」ウィンドウ内でプロジェクト名を右クリックし、コンテキスト・メニューから 「Options」を選択します。

2) 「Build Options」ダイアログ・ボックスで、適切なタブ（Compiler、Assembler、または Linker）を選択します。

3) プログラムのビルド時に使用されるオプションを選択します。

4) 「OK」をクリックして、選択内容を受け入れます。

ファイル固有オプションを設定する方法

特定のファイルに対して設定されたオプションは、プロジェクト・レベルの設定を上書きします。

1) 「Project View」ウィンドウ内でソース・ファイルの名前を右クリックし、コンテキスト・メニューから 「File Specific Options」を選択します。

2) ファイルのコンパイル時に使用されるオプションを選択します。

3) 「OK」をクリックして、選択内容を受け入れます。

ファイル固有のオプションは、プロジェクト・ファイルに保存されます。この保存の際には、プロジェクト・オプションと、そのファイルに設定されたオプションとの相違だけが記録されます。

ファイルに対してビルド・オプションを設定するときには、十分注意してください。プロジェクト・レベルで整合性を保持する必要があるオプションを変更することに対する保護はありません。

10.6 プログラムのビルド

次のコマンドを使用すると、ソース・ファイルのコンパイルとリンクを実行することができます。メニュー・コマンドの代りに、ツールバー・ボタンを使用できます。

 **Compile File**

「Project」→「Compile File」を選択して、現在のソース・ファイルだけをコンパイルします。リンクは実行されません。

 **Incremental Build**

「Project」→「Build」を選択して、現在のプロジェクトをビルドします。このコマンドは、前回のビルド以降に変更されたファイルだけをコンパイルします。ファイルのコンパイルが必要かどうかを判別するには、ソース・ファイルのタイム・スタンプを、オブジェクト・ファイルのタイム・スタンプと比較します。ソース・ファイルのタイム・スタンプが、対応するオブジェクト・ファイルのタイム・スタンプより大きい場合、そのファイルは再コンパイルされます。出力ファイルの再リンクが必要かどうかを判別するには、各オブジェクト・ファイルのタイム・スタンプを、その出力ファイルのタイム・スタンプと比較します。オブジェクト・ファイルのタイム・スタンプの方が大きい場合、その出力ファイルは再リンクされます。

 **Rebuild All**

「Project」→「Rebuild All」を選択して、現在のプロジェクト内のすべてのファイルを再コンパイルし、出力ファイルを再リンクします。

 **Stop Build**

「Project」→「Stop Build」を選択して、ビルド・プロセスを中止します。ビルド・プロセスが停止するのは、現在のファイルのコンパイルが完了した後だけです。

第 11 章

コード実行のプロファイリング

Code Composer Studio を使用すると、コード内の特定エリアについての実行の統計情報を収集することができます。これはプロファイリングと呼ばれ、アプリケーションのパフォーマンスについて即時にフィードバックし、コードを最適化できるようにします。たとえば、ユーザは、アルゴリズムが使用する CPU 時間量を判断することができます。また、その他のプロセッサ・イベント（たとえば、分岐、サブルーチン・コール、または発生した割り込みの数）もプロファイルすることができます。

項目 **ページ**

11.1 プロファイル・クロック

プロファイル・クロックは、プロファイリング時に、実行やシングル・ステップ操作の間に実行するプロセッサの命令サイクル数、またはその他のイベントの発生数をカウントします。

プロファイル・クロックは、CLK という名前の変数として、「Clock」ウィンドウからアクセスすることができます。この CLK 変数は 「Watch」ウィンドウで表示し、「Edit Variable」ダイアログ・ボックスで変更できます。CLK は、ユーザ定義の GEL 関数に対しても使用できます。

命令サイクルの測定方法は、ユーザがどのデバイス・ドライバを使用しているかに応じて異なります。JTAG スキャン・パスを通じて通信するデバイス・ドライバの場合、命令サイクルは、プロセッサのオンチップ分析機能を使用してカウントされます。これ以外のデバイス・ドライバでは、他のタイプのタイマを使用しなければならない場合があります。これらのリソースを管理するには、プロファイル・クロックをイネーブルにしたり、ディセーブルにしたりする必要があります。

シミュレータは、DSP の模擬オンチップ分析インターフェイスを使用して、プロファイリング・データを収集します。クロックがイネーブルであると、Code Composer Studio は、命令サイクルのカウントの実行に必要なリソースを引き継ぎます。クロックがディセーブルであると、このリソースはユーザが使用できます。サイクルのカウント方法は、11.1.2 節 「プロファイル・クロックの精度」を参照してください。

上記の機能は、次のように使用することができます。

プロファイル・クロックをイネーブルまたはディセーブルにする方法

「Profiler」→「Enable Clock」の順に選択します。クロックがイネーブルであると、このメニュー項目の隣にチェック・マークが表示され、クロックがディセーブルであると、チェック・マークが表示されません。

プロファイル・クロックを表示する方法

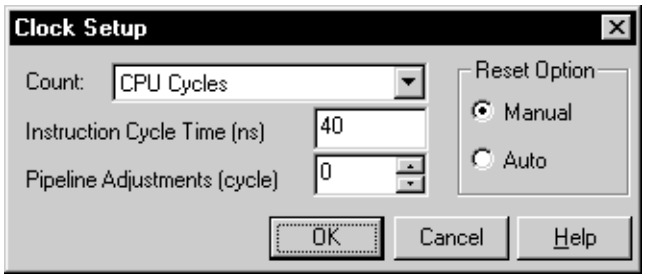
「Profiler」→「View Clock」の順に選択します。「Clock」ウィンドウが立ち上がり、CLK 変数の値が表示されます。

プロファイル・クロックをリセットする方法

CLK 変数を編集して 0 に設定するか、「Clock」ウィンドウの内容をダブルクリックします。

11.1.1 プロファイル・クロックのセットアップ

プロファイル・クロックのセットアップを変更するには、メニューから 「Profiler」→「Clock Setup」を選択します。「Clock Setup」ダイアログ・ボックスが開きます。



「Instruction Cycle Time」フィールドに、1 つの命令の実行にかかる時間を入力することができます。この情報は、プロファイラ統計情報を表示するために、サイクル数を時間または周波数に変換するのに使用されます。

「Count」フィールドのドロップダウン・リストで、プロファイル対象のイベントを選択します。使用するデバイス・ドライバによっては、「CPU Cycles」だけしかリストにない場合があります。しかし、いくつかのデバイス・ドライバは、その他のイベントのプロファイリングに、オンチップ分析機能を利用します。こうしたイベントには、割り込みの回数、サブルーチンまたは割り込みのリターン回数、分岐の回数、サブルーチン・コールの回数などが含まれます。たとえば、分岐を選択すると、CLK 変数は、CPU サイクルをカウントするのではなく、発生した分岐の数をカウントします。

注 : シミュレータ - イベントのプロファイル

シミュレータは、「Count」フィールドに「CPU Cycles」パラメータだけを表示します。これは、このシミュレータが、シミュレートされる DSP ターゲットとのインターフェイスを介して他の DSP イベントをプロファイルしないからです。

「Reset Option」パラメータを使用すると、CLK 変数の集計方法を決定することができます。「Manual」ラジオ・ボタンを選択すると、CLK 変数は、クロックをリセットすることなく命令サイクル数を集計します。これは、TI シミュレータの動作とほぼ同じです。「Auto」ラジオ・ボタンを選択すると、CLK 変数は、ターゲット・プロセッサの実行またはステップ実行を行う前に、自動的にリセット（0 に設定）されます。したがって、この CLK 変数は、前回の実行またはステップ実行以降のサイクル数だけを表示することになります。これは、TI エミュレータの動作とほぼ同じです。

「Pipeline Adjustment」フィールドを使用すると、ブレークポイントの処理時にプロセッサのパイプラインをフラッシュするのに使用されるサイクル数を、差し引きすることができます。プロセッサは、ブレークポイントの処理、プロセッサの一時停止、またはプロセッサのステップ実行のために停止するたびに、パイプラインをフラッシュする必要があります。このフラッシュに必要なサイクル数は、プログラムのウェイト・ステート数に応じて異なり、このウェイト・ステート数も状況により異なります。より正確なサイクル数を入手するために、サイクル数から差し引く値を指定して、この影響を補正することができます。

11.1.2 プロファイル・クロックの精度

プログラムの実行中に、プロファイル・クロックは、ウェイト・ステートやパイプライン・コンフリクト用のサイクル数を含む命令サイクル数を、正確にカウントします。しかし、ターゲット・プロセッサからこのサイクル数を読み取るために、プロセッサは一時停止しなければなりません。プロセッサの停止時には、パイプラインのフラッシュ、パイプライン・コンフリクトの欠落、および余分なプログラム・フェッチを原因とする、いくつかの種類の測定エラーが生じます。

プロセッサが停止するたびに、パイプラインはフラッシュされなければなりません。この結果、余分な命令サイクル数がカウントされます。パイプラインがフラッシュされた後では、次の命令が実行されないので、次の命令とのコンフリクトが回避され、余分な命令サイクルがカウントされなくなります。この結果、あるべきはずの命令サイクル数より少なくなります。プログラムがソフトウェア・ブレークポイントによって一時停止する場合は、ブレークポイントの位置にある命令のフェッチとデコードを行うために、余分な命令サイクル数が必要になります。通常、この余分なフェッチとデコードは、パイプラインをフラッシュするサイクル数とオーバラップします。しかし、命令のフェッチがゼロ・ウェイト・ステートで実施されない場合は、余分なウェイト・ステートがカウントされます。

「Pipeline Adjustment」フィールドを適切な数に設定しても、すべての測定エラー、特にパイプライン・コンフリクトの欠落によるエラーを十分に補正することはできません。その結果、プログラムをステップ実行または実行させる回数が多いほど、プロファイル・クロックの精度は低下します。同様に、検出されるブレークポイント、プローブ・ポイント、およびプロファイル・ポイントが増加するほど、クロックの精度は低下します。

正確な命令サイクル数を取得する方法

次の手順を実行すると、プログラム内の 2 つのポイント（A と B）間の正確なサイクル数を取得することができます。

1) プログラム・フローの中で、ポイント B から 4 つ以上の命令を経過したポイント C に、ブレークポイントを設定します。
2) ポイント A にブレークポイントを設定し、そのブレークポイントまで実行します。
3) クロックをリセットし、ポイント A のブレークポイントを削除します。
4) ポイント C のブレークポイントまで実行し、CLK 変数の値を記録します。この値は、ポイント A と C の間のサイクル数を表します。
5) ポイント A の代わりにポイント B を使用して、上記のステップ 2 ～ 4 を繰り返します。プログラムの状態が、ポイント A とポイント C 間のサイクル数を測定した場合と同じであることを確認してください。
6) ポイント A とポイント C 間のサイクル数から、ポイント B とポイント C 間のサイクル数を差し引きます。これにより、ポイント B でプロセッサが停止することによる測定エラーが除去されます。

11.2 プロファイル・ポイント

プロファイル・ポイントとは、プログラム内の特定の位置でプロファイリング情報を取り込む特殊なブレークポイントです。各プロファイル・ポイントは、プロファイル・ポイントに到達した回数をカウントし、直前のプロファイル・ポイントが検出された以降に経過したサイクル数、またはその他のイベントの発生数に関する統計を保持します。プロファイル・ポイントは、ブレークポイントとは異なり、その統計を集計した後に実行を再開します。プロファイル・ポイントを設定すると、それをイネーブルにしたり、ディセーブルにしたりすることができます。

プロファイル・ポイントが命令サイクル数、またはその他のイベント数についての統計情報を保持するには、プロファイル・クロックがイネーブルになっていなければなりません。プロファイル・クロックがディセーブルであると、プロファイル・ポイントは、各プロファイル・ポイントに到達した回数をカウントできますが、他の統計を生成することができません。

プロファイル・ポイントを追加する方法

プロファイル・ポイントを作成するには、そのプロファイル・ポイントを置きたいソース・ファイルまたは「Dis-Assembly」ウィンドウ内の行にカーソルを置き、「Project」ツールバー上の 「Profile Point 」 ボタンをクリックします。

Profile Point: 

既存のプロファイル・ポイントを削除する方法

1) 「Profiler」→「Profile Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Profile Points」タブが選択された状態で表示されます。
2) 「Profile Point」ウィンドウで、プロファイル・ポイントを選択します。
3) 「Delete」をクリックします。
4) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

すべてのプロファイル・ポイントを削除する方法

「Project」ツールバーから、「Remove All Profile Points」ボタンをクリックします。

Remove All Profile Points: 

次のようにメニュー・コマンドを使用して、すべてのプロファイル・ポイントを削除することもできます。

1) 「Profiler」→「Profile Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Profile Points」タブが選択された状態で表示されます。
2) 「Delete All」をクリックします。
3) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

11.2.1 プロファイル・ポイントのイネーブルとディセーブル

プロファイル・ポイントを設定すると、それをディセーブルにしたり、イネーブルにしたりすることができます。プロファイル・ポイントをディセーブルにすると、そのプロファイル・ポイントの位置とタイプ、および集計された統計情報を保持しながら、そのプロファイル・ポイントの動作を素早く一時的に中断することができます。

プロファイル・ポイントをイネーブルにする方法

1) 「Profiler」→「Profile Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Profile Points」タブが選択された状態で表示されます。
2) 「Profile Point」ウィンドウで、イネーブルにしたいプロファイル・ポイントをリストの中から選択します。プロファイル・ポイントが現在ディセーブルの場合、チェック・ボックスは空白になっています。
3) プロファイル・ポイントのチェック・ボックスをクリックします。これで、そのチェック・ボックスには、そのプロファイル・ポイントがイネーブルであることを示すチェック・マークが付きます。
4) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

プロファイル・ポイントをディセーブルにする方法

1) 「Profiler」→「Profile Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Profile Points」タブが選択された状態で表示されます。
2) 「Profile Point」ウィンドウで、ディセーブルにしたいプロファイル・ポイントをリストの中から選択します。プロファイル・ポイントが現在イネーブルの場合、チェック・ボックスにはチェック・マークが付いています。
3) プロファイル・ポイントのチェック・ボックスをクリックします。これで、そのチェック・ボックスは、そのプロファイル・ポイントがイネーブルであることを示す空白になります。
4) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

すべてのプロファイル・ポイントをイネーブルにする方法

1) 「Profiler」→「Profile Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Profile Points」タブが選択された状態で表示されます。
2) 「Enable All」をクリックします。
3) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

すべてのプロファイル・ポイントをディセーブルにする方法

1) 「Profiler」→「Profile Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Profile Points」タブが選択された状態で表示されます。
2) 「Disable All」をクリックします。
3) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

11.3 ハードウェア・プロファイル・ポイント

ハードウェア・プロファイル・ポイントは、通常のプロファイル・ポイントと同じように作動します。ただし、ソフトウェア・ブレークポイントではなく、ハードウェア・ブレークポイントを使用して実現されます（4.3 節「ハードウェア・ブレークポイント」を参照）。ハードウェア・プロファイル・ポイントは、読み取り専用メモリでプロファイルする場合、または所定の位置で N 回目ごとにだけプロファイルしたい場合に便利です。

注 : ターゲット・プロセッサの一時停止

ハードウェア・プロファイル・ポイントが検出されると、ターゲット・プロセッサは一時的に停止します。したがって、ハードウェア・プロファイル・ポイントを使用すると、ターゲット・アプリケーションは、リアルタイムの制約条件を満たすことができない場合があります。

注 : ハードウェア・プロファイル・ポイントがサポートされない場合

ハードウェア・プロファイル・ポイントは、通常、JTAG ベースのデバイス・ドライバでのみ使用することができます。シミュレートされる DSP ターゲットでは、実現できません。

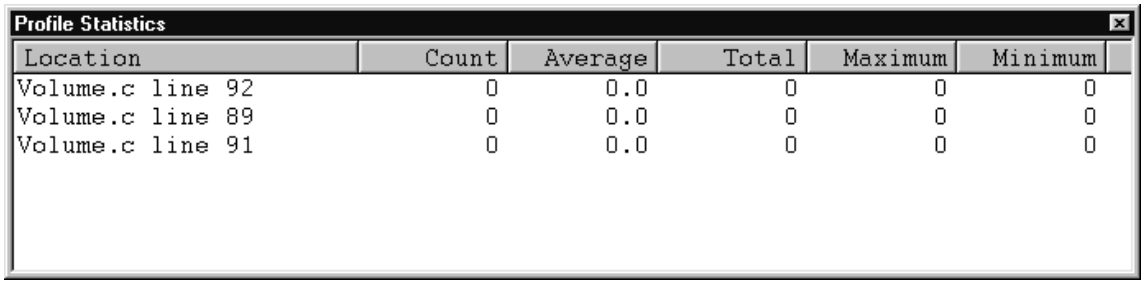
ハードウェア・プロファイル・ポイントを追加する方法

1) 「Profiler」→「Profile Points」の順に選択します。「Break/Probe/Profile Points」ダイアログ・ボックスは、「Profile Points」タブが選択された状態で表示されます。
2) 「Profile Type」フィールドで、「H/W profile at location」を選択します。
3) 「Location」フィールドで、そのプロファイル・ポイントを設定する位置を入力します。次の方法のどちらかを使用することができます。
   - 絶対アドレスの場合は、任意の有効な C 式、C 関数の名前、またはシンボル名を入力します。
   - ご使用の C ソース・ファイルに基づいて、プロファイル・ポイントの位置を入力します。この方法は、実行可能プログラム内の C 命令の位置が分からない場合に便利です。C ソース・ファイルに基づいて位置を入力する場合のフォーマットは、次のとおりです。

     fileName line *lineNumber*
4) 「Add」をクリックして、新しいハードウェア・プロファイル・ポイントを作成します。
5) 「OK」をクリックして、ダイアログ・ボックスをクローズします。

11.4 統計情報の表示

プロファイラの統計情報を表示するには、メニュー・バーから 「Profiler-View Statistics」を選択します。「Profile Statistics」ウィンドウが開きます。



「Profile Statistics」ウィンドウには、プロファイル・ポイントごとの結果が表示されます。各ポイントには、そのポイントに到達した回数、および直前のプロファイル・ポイントに到達してから経過したサイクル数、またはその他のイベントの発生数についての統計情報が表示されます。この統計情報には、サイクルの最小数、最大数、合計数、および平均数が含まれます。

「Profile Statistics」ウィンドウは、プロファイル・ポイントに到達するたびに更新されますが、ウィンドウの更新回数が多すぎると、プロファイリングのパフォーマンスが低下する可能性があります。ウィンドウの更新回数を減らすには、2 つの方法があります。そのウィンドウをプローブ・ポイントに接続するか、または必要に応じてそのウィンドウを開閉するかです。「Profile Statistics」ウィンドウをプローブ・ポイントに接続すると、このウィンドウは、そのプローブ・ポイントが検出されるときにだけ更新されます。したがって、ユーザは、自分のアプリケーションで 「Profile Statistics」ウィンドウがいつ更新されるかを制御することができます。このウィンドウは、単にプロファイラの結果を表示するものであり、プロファイラ統計情報を収集するためには必要ありません。したがって、このウィンドウは常に開いている必要はありません。

プロファイル・ポイント用の統計情報を消去する方法

1) マウス、またはカーソル・キーを使用して、「Profile Statistics」ウィンドウ内で、統計を消去するプロファイル・ポイントを選択します。選択されたプロファイル・ポイントの輪郭が、点線で表示されます。
2) 右クリックし、コンテキスト・メニューから「Clear Selected」を選択します。そのプロファイル・ポイントの「count」、「average」、「total」、「maximum」、および「minimum」の各フィールドが消去され、0 に設定されます。「Edit」メニューには、このアクションに対する取り消し (undo) はありません。

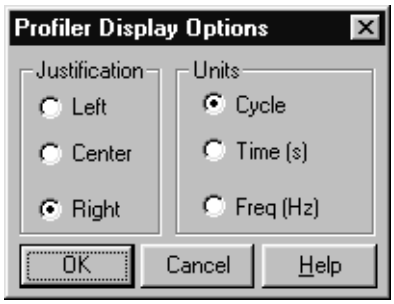
すべてのプロファイル・ポイント用の統計情報を消去する方法

右クリックし、コンテキスト・メニューから 「Clear All」を選択します。すべてのプロファイル・ポイントの「count」、「average」、「total」、「maximum」、および「minimum」各フィールドが消去され、0 に設定されます。「Edit」メニューには、このアクションに対する取り消し (undo) はありません。

表示オプションを変更する方法

1) 「Profile Statistics」ウィンドウをクリックして、このウィンドウをカレント・ウィンドウにします。

2) 右クリックし、コンテキスト・メニューから 「Properties」→「Display Options」の順に選択します。「Profiler Display Options」ダイアログ・ボックスが表示されます。



「Justification」フィールド内の 「Left」、「Center」、または 「Right」ラジオ・ボタンのいずれかを選択して、データの表示方法を変えても構いません。

「Profile Statistics」ウィンドウの計測単位を変更するには、「Units」フィールド内の「Cycle」、「Time」、または 「Frequency」ラジオ・ボタンのいずれかを選択します。計測単位は、サイクル数、時間（秒）、または周波数（ヘルツ）のいずれかにできます。「Clock Setup」ダイアログ・ボックスでは、命令のサイクル・タイムを設定することができます（11.1.1 節「プロファイル・クロックのセットアップ」を参照）。サイクル・タイムは、逆数を取ると周波数に変換されます。単位が周波数であると、たとえば、プログラムが所定のサンプリング周波数を処理できるかどうかを判別する際に役立つ場合があります。サイクル数は、そのサイクル数に命令サイクル・タイムを乗算すると、時間に変換されます。

11.5 プロファイル・ポイントを使用した分割と最適化

この節では、プロファイル・ポイントを使用して、プログラムの CPU 使用を最適化するための手順について説明します。この手順は、アセンブリ言語プログラムと C 言語プログラムの両方に適しています。

1) 完全最適化をイネーブルにして、C プログラムをコンパイルします。
2) 入力ファイルとプローブ・ポイントをセットアップして、ソース・コードに変更を加えずに、アルゴリズムをシミュレートします。プロファイル・ポイントは、プログラムの実行速度を低下させるので、アルゴリズムをシミュレートする必要があります。
3) アルゴリズムのメイン・ループ内に、プロファイル・ポイントを設定します。アプリケーションを実行して、プログラム全体の実行についてのプロファイル統計情報を生成します。プログラムが、すでにリアルタイムの制約条件を満たしている場合は、これで作業が完了します。
4) メイン・ループに数個のプロファイル・ポイントを追加することによって、アルゴリズムのメイン・ループを分割して、プログラムを複数のセクションに分割します。各プロファイル・ポイントでは、プログラムの先行セクションの実行に必要なサイクル数が測定されます。
5) アプリケーションを実行して、統計情報を集計します。
6) 「Profile Statistics」ウィンドウ内のプロファイル・ポイントのリストを分類して、使用するサイクル数が最も多いコードのセクションを見つけます。そのセクションをさらに分割するには、設定するプロファイル・ポイントを増やします。
7) ステップ 5 と 6 を繰り返して、さらに細かいプロファイル解決を行います。
8) 使用しているサイクル数が最も多いコードの最小セクションを書き直し、ステップ 1 に進みます。

第 12 章

General Extension Language (GEL)

General Extension Language (GEL) は、Code Composer Studio の有用性を拡張する関数をユーザが作成できるようにする、C に似たインタプリタ言語です。ユーザは、GEL 文法を使用して自身の GEL 関数を作成してから、その関数を Code Composer Studio にロードします。GEL を使用すると、実際のターゲット・メモリ位置、またはシミュレータ上のターゲット・メモリ位置にアクセスし、Code Composer Studio の GEL メニューにオプションを追加することができます。GEL は特に、テストの自動化とユーザ・ワークスペースのカスタマイズに便利です。GEL 関数は、式を入力できる任意の場所から呼び出すことができます。また、GEL 関数を 「Watch」ウィンドウに追加して、どのブレークポイントでも GEL 関数が実行されるようにすることもできます。

項目 **ページ**

12.1 GEL 文法

GEL は、C プログラム言語の 1 つのサブセットです。しかし、ユーザはホスト変数を宣言することはできません。すべての変数は、ユーザの DSP プログラムで定義され、実際のターゲットまたはシミュレーション・ターゲット上に存在しなければなりません。ターゲット上で定義されない識別子は、GEL 関数とそのパラメータだけです。変数が評価されるときに、Code Composer Studio デバッガは、ターゲットから必要な情報を取得します。シンボル情報を含んだ COFF ファイルが、すでにロードされていなければなりません。

GEL は、次のタイプの文をサポートします。

- ❏ 関数定義
- ❏ 関数パラメータ
- ❏ GEL 関数の呼び出し
- ❏ Return 文
- ❏ If-else 文
- ❏ while 文
- ❏ GEL コメント
- ❏ 前処理文

例 12-1「基本的な Gel 関数」は、GEL 関数の定義方法を示しています。いったん関数がロードされると、ユーザは、正しいパラメータを指定してその関数を呼び出すことによって、その関数をいつでも実行することができます。MyFunc(100, 0)、または MyFunc(200) のような呼び出しは、両方とも有効です。

例 12-1. 基本的な Gel 関数

```
MyFunc(parameter1, parameter2)
{
  if (parameter1 == parameter2)
  {
    a = parameter1;
  }
else
  {
    while (c)
    {
      b = parameter2;
      c--;
    }
  }
}
```

上記のシンボル a、b、および c は、DSP ターゲット変数であると仮定されます。

12.2 GEL 関数定義

GEL 関数は、次のように定義されます。ここでは、イタリック体の項目は変数です。

```
funcName( [parameter1 [ , parameter2 ...[, parameter6 ] ] ] )
{
    statements
 }
```

funcName	GEL 関数
parameters	有効な GEL パラメータ
statements	有効な GEL 文

GEL 関数は、.gel 拡張子をもつテキスト・ファイル内で定義されます。1 つの GEL ファイルには、複数の GEL 関数定義が含まれている場合があります。

GEL 関数は、戻りの型を識別しません。また、必要とするパラメータの型を定義するのにヘッダ情報を必要とすることもありません。このヘッダ情報は、データ値から自動的に得ることができます。

標準 C の場合と同じように、GEL 関数定義は、別の GEL 関数定義内に組み込むことはできません。

例 12-2「Square 関数」では、ユーザが関数に渡す値を二乗しています。

例 12-2. Square 関数

```
square(a)
{
  return a*a;
}
```

Watch ウィンドウ内でこの関数への呼び出しを追加すると、次のようになります。

```
square(1.2) = 1.44
square(5) = 25
```

a は GEL パラメータなので、DSP ターゲットで a を定義する必要はありません。

各パラメータの後に、そのパラメータの使用方法を解説するオプションの文字列を続けることができます。たとえば、例 12-3「解説的なパラメータ文字列」に示されているとおりです。この記述は、dialog 関数用に作成されるダイアログ・ボックスで使用されます。

例 12-3. 解説的なパラメータ文字列

```
dialog Init(filename "File to be Loaded", CPUname "CPU Name",
initValue "Initialization Value")
{
  GEL_Load(filename, CPUname);
  a = initValue;
}
```

このダイアログは、Init 関数をメニュー・バーに追加します。パラメータ入力ダイアログ・ボックスにおいて、パラメータに対するフィールド名を提供するために、パラメータの後ろに文字列が指定されています。文 a = initValue では、文字 a はパラメータ・リストに定義されていません。したがって、実際のターゲット、またはシミュレーション・ターゲット上で定義する必要があります。文字 a が定義されていない場合、この関数を呼び出すときにエラーが発生します。組み込み関数 GEL_Load に対する呼び出しに注意してください。この関数には、最初のパラメータのファイル名と CPU 名を識別する文字列が必要です。この CPUname パラメータはオプションであり、複数のプロセッサをセットアップするときに便利です。ユーザは、最初のパラメータ用の文字列を渡す必要があります。この関数に対する有効な呼び出しの例は、次のとおりです。

```
Init("c:\\mydir\\myfile.out", "cpu_a", 0).
```

12.3 GEL 関数パラメータ

GEL 関数定義でパラメータを定義すると、GEL 関数に引数を渡すことができます。C 関数パラメータとは異なり、このパラメータ・タイプは定義されません。パラメータ名だけが必要です。このパラメータ・タイプは、渡された引数から自動的に判別されます。GEL パラメータは、次のいずれかにすることができます。

- ❏ 実際のターゲットまたはシミュレーション・ターゲットのシンボル値―ターゲット・シンボルが渡される場合
- ❏ 数値定数 ― 任意の式、または定数値が渡される場合
- ❏ 文字列定数 ― 文字列定数が渡される場合

実行時に渡される引数により、パラメータが取る値が決定されます。

以下は、GEL 関数定義の例です。

```
Initialize(a, filename, b)
{
  targVar = b;
  a = 0;
        /*a DSP symbol must be passed for parameter 'a'  */
  GEL_Load(filename);
        /* a string constant must be passed for filename */
  return b*b;
}
```

上記の GEL 関数に対する正しい呼び出しの例は、次のとおりです。

```
Initialize(targetSymbol, "c:\\myfile.out", 23 * 5 + 1.22);
```

この関数が実行されると、パラメータ a は、DSP シンボル targetSymbol であると判別され、パラメータ filename は文字列定数「c:\myfile.out」であると判別され、パラメータ b は、定数値 116.22 であると計算されます。これらの値は、それらのパラメータの代わりに、その関数で使用されます。

パラメータ a に対する DSP シンボルが渡されなかった場合、2 番目の文 a = 0 の実行時に、ランタイム・エラーが発生します。たとえば、定数値 20 を渡した場合、2 番目の文は 20 = 0 になります。これは有効な代入文ではありません。

最初のパラメータに対する有効な DSP シンボルが渡される場合であっても、GEL 関数の実行時に、そのシンボル情報が Code Composer Studio デバッガにロードされていることを確認する必要があります。シンボル targetSymbol が定義される場合、シンボル target Symbol が渡されると、上記の関数呼び出しでは、targetSymbol に 0 が代入されます。

GEL パラメータは、1、3.1415、0x100、c:\\filename などの数値、または文字列にすることができます。数値パラメータの場合、GEL では、任意の有効な C 式を渡すことができます。この式は、評価された後、その GEL 関数に渡されます。最後の値にピリオドまたは指数符号が含まれている（たとえば、1.2 または 1.34e4）場合、その値は、real 型であると仮定されます。含まれていない場合は、整数であると仮定されます。

次のフォーマットのどちらかで、initialize GEL 関数を呼び出すことができます。

```
Initialize(targetSymbol, "c:\\mydir\\myfile.out",10);
Initialize(targetSymbol, "c:\\filename.out", 1.2);
```

最初の呼び出しでは、パラメータ b は整数値であると仮定されます。2 番目の呼び出しでは、その入力が real 型であると判定されます。ターゲット変数 targVar が int 型の場合は、targVar への代入時に、パラメータ b は切り捨てられます。

パラメータ・シンボルを使用して GEL 関数を定義すると、その関数に引数を渡すかどうかは任意選択です。これは、そのパラメータ値が、数値の場合は 0 に初期化され、それ以外の場合はヌル文字列に初期化されるからです。パラメータが渡されない場合、その関数は、そのパラメータのデフォルト値を使用します。つまり、次のように上記の関数を呼び出すこともできます。

```
Initialize();
Initialize(targSymbol,"c:\\myfile.out");
```

最初の呼び出しでは、targVar に 0 を代入します。しかし、最初の呼び出しで文 GEL_Load (filename) を実行しようとすると、エラーが検出されます。その filename に対して渡される引数がないので、その文は GEL_Load("") と等しくなり、その結果、Invalid File Name エラーになります。また、文 a = 0 についてのエラーが発生します。Initialize 関数に対する 2 番目の呼び出しでは、その関数に 2 つの引数だけが渡されます。3 番目の引数には、自動的に 0 が代入されます。

GEL は、非常におおまかに定義されるので、デフォルト値で構わない場合には、GEL 関数呼び出しを簡単にする柔軟性を備えています。また、ユーザが希望すれば、より多くのパラメータを渡すこともできます。最高 6 つの引数を取ることができる TextOut などの GEL 組み込み関数を使用するときに、この柔軟性が分かります。拡張アプリケーションの場合、出力に対するいろいろな制御を提供するためにあらゆるパラメータを使用したい場合があります。また、次のように、1 つのパラメータだけを使用してこの関数を呼び出すこともできます。

```
GEL_TextOut("Hello World!")
```

12.4 GEL 関数と文の呼び出し

GEL 関数は、有効な C 式を入力できる任意の場所から呼び出すことができます。また、有効な C 式を受け入れる任意のダイアログ・ボックスから、または別の GEL 関数内から呼び出すことができます。しかし、再帰的な GEL 関数呼び出しはサポートされません。GEL 関数の実行中、ユーザは、その関数の別のインスタンスを実行することはできません。

GEL 関数に引数を渡すかどうかは、任意選択です。省略された引数は、デフォルト値を取ります。詳細は、12.3 節 「GEL 関数パラメータ」を参照してください。

以下の節では、C 構文を使用するいくつかの GEL 文について解説します。

12.4.1 GEL Return 文

GEL は、関数内の標準 C の return 文をサポートします。一般的な形式は、次のとおりです。

```
return expression;
```

関数は、値を戻す必要はありません。expression のない return 文により、呼び出し側に制御が戻されます。ただし、有効な値は戻されません。return 文を検出せずに関数の終わりに達したときにも、同じことが起きます。呼び出し元関数は、関数が戻す値を無視できます。

C の場合と異なり、GEL 関数定義は、戻り値の型を指定しません。戻り値の型は、実行時に判別されます。

12.4.2 GEL If-Else 文

GEL は、標準 C の if-else 文をサポートします。一般的な形式は、次のとおりです。

```
if (expression)
    statement1
else
    statement2
```

1 つの if-else 文に対応している 2 つの文の一方だけが実行されます。*expression* が真の場合は、*statement1* が実行されます。真でない場合は、*statement2* が実行されます。例 12-4「If-Else 文」に示されているように、各 statement は 1 つの文でも、中括弧に囲まれた複数の文でも構いません。

例 12-4. If-Else **文**

```
if (a == 25)
  b = 30;

if (b == 20)
{
  a = 30;
  c = 30;
}
else
{
  d = 20;
}
```

12.4.3 GEL While 文

GEL while 文は標準 C の while 文とほぼ同じですが、GEL バージョンは、continue 文も break 文もサポートしません。一般的な形式は、次のとおりです。

while (*expression*)
    statement

この文では、*expression* が評価されます。この expression が真（ゼロ以外）の場合、*statement* は実行され、*expression* は再評価されます。このサイクルは、*expression* が偽 (0) になるまで続きます。偽になると、その文の後から実行が再開されます。例 12-5「While 文」に示されているように、この *statement* は 1 つの文でも、中括弧に囲まれた複数の文でも構いません。

例 12-5. While **文**

```
while (a != Count)
{
  dataspace[a] = 0;
  a--;
}
```

12.4.4 GEL コメント

GEL は、ファイル内の標準 C のコメントをサポートします。GEL コメントは、文字 /* および */ によって区切られ、複数行にわたる場合があります。

12.4.5 GEL 前処理文

GEL は、標準 #define 前処理キーワードをサポートします。これは、現在使用可能な唯一の前処理キーワードです。

次のような制御行により、プリプロセッサは、その識別子の後続のインスタンスを、所定のシーケンスのトークンに置き換えます。

#define *identifier token-sequence*

token-sequence の前の空白、および後の空白は廃棄されます。

最初の識別子と左小括弧との間にスペースのない次のような制御行は、識別子リストによってパラメータが指定されているマクロ定義です。

#define *identifier*(*identifier-list*) *token-sequence*

#define キーワードを使用してマクロが定義されていると、そのファイル内の任意の位置だけでなく、後で Code Composer Studio にロードされるその他の任意のファイル内の位置で、そのマクロを使用することができます。

12.5 GEL 関数のロードとアンロード

ユーザ自身の GEL 関数が入っているファイルを定義した場合は、そのファイル内の関数にアクセスする前に、ファイルを Code Composer Studio にロードしておく必要があります。ファイルがロードされると、GEL 関数は Code Composer Studio のメモリに常駐し、いつでも実行できるようになります。この GEL 関数は、対応するファイルが削除されるまでメモリ内に存在します。ロードされたファイルを変更した場合には、その変更内容を有効にするために、そのファイルをいったんアンロードしてから、再ロードする必要があります。

GEL ローダは、ファイルをロードするときに、ファイル内の構文エラーをチェックします。ただし、変数が定義されているかどうかを調べるための変数のチェックは行いません。したがって、シンボル情報が入っている COFF ファイルをロードする前であっても、GEL 関数をロードすることができます。また、まだ定義やロードが行われていない GEL 関数を参照することもできます。GEL 関数の実行時には、シンボルが定義されなければなりません。Code Composer Studio は、ファイルのロード中に構文エラーを検出すると、そのロード処理を中止し、該当するエラー・メッセージを表示します。ユーザはそのエラーを修正してから、そのファイルを再ロードする必要があります。

GEL ファイルのロード方法

1) 「File」→「Load GEL」の順に選択します。
2) 「Load GEL File」ダイアログ・ボックスで、ユーザ自身の GEL 関数が入っているファイルをブラウズします。
3) そのファイル名をダブルクリックするか、そのファイル名をクリックした後、「開く(O)」をクリックします。

または

1) 「Project View」ウィンドウが表示されない場合は、「View」→「Project」の順に選択します。
2) 「GEL Files」フォルダを右クリックし、コンテキスト・メニューから 「Load GEL」を選択します。
3) 「Load GEL File」ダイアログ・ボックスで、ユーザ自身の GEL 関数が入っているファイルをブラウズします。
4) そのファイル名をダブルクリックするか、そのファイル名をクリックした後、「開く(O)」をクリックします。

GEL ファイルのアンロード方法

1) 「Project View」ウィンドウが表示されない場合は、「View」→「Project」の順に選択します。
2) GEL files フォルダの横にある + 符号をクリックして、個々の GEL ファイルを表示します。
3) 削除する GEL ファイルを右クリックします。
4) コンテキスト・メニューから 「Remove」を選択します。

12.6 キーワードを使用して「GEL」メニューに GEL 関数を追加する方法

頻繁に利用する GEL 関数を「GEL」メニューに追加することができます。「GEL」メニューに追加するには、menuitem キーワードを使用して、メニュー・バー上の「GEL」メニューの下に、新しいメニュー項目のドロップダウン・リストを作成します。このリストの作成後、キーワードの hotmenu、dialog、または slider を使用して、最新のドロップダウン・リスト内に新しいメニュー項目を追加できるようになります。ユーザ定義のメニュー項目（「GEL」メニューの下にある）を選択すると、ダイアログ・ボックス、またはスライダ・オブジェクトが表示されます。

12.6.1 hotmenu キーワード

「GEL」メニューに、選択すると直ちに実行される GEL 関数を追加するには、hotmenu キーワードを使用します。この構文は、次のとおりです。

```
hotmenu funcName()
{
    statements
}
```

このキーワードは、例 12-6「Hotmenu キーワード」に示されているように、パラメータを渡す必要がない GEL 関数に使用されます。

例 12-6. Hotmenu キーワード

```
menuitem "My Functions";
hotmenu InitTarget()
{
  *waitState = 0x11;
}
hotmenu LoadMyProg()
{
  GEL_Load("c:\\mydir\\myfile.out");
}
```

上記の例では、「GEL」メニューの下に、次の項目が副選択項目として追加されます。

「InitTarget」コマンドを選択すると、このコマンドは直ちに実行されます。パラメータを渡す必要がある GEL 関数を呼び出すには、dialog キーワードを使用します。

12.6.2 dialog キーワード

GEL 関数を「GEL」メニューに追加し、パラメータ入力用のダイアログ・ウィンドウを作成するには、dialog キーワードを使用します。「GEL」メニューからこの関数を選択すると、パラメータの入力を求めるダイアログ・ウィンドウが表示されます。関数宣言内のパラメータの横にある文字列は、ダイアログ・ボックス内でフィールド名として使用されます。dialog GEL 関数の構文は、次のとおりです。

```
dialog funcName( paramName1 "param1 definition", paramName2 "param2
definition", ......)
{
    statements
}
```

paramName[1-6]　パラメータ変数名で、関数の内部で使用されます。

"param1 definition"　パラメータに対する解説で、パラメータの左にフィールド名として表示されます。

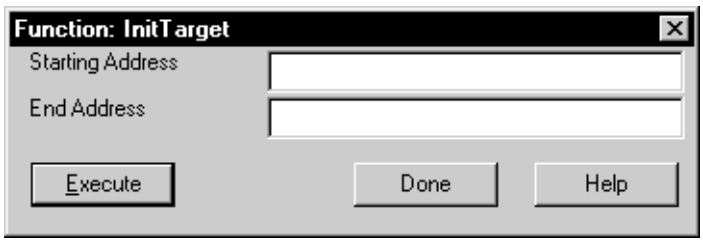
このダイアログ・ウィンドウを使用して追加された GEL 関数には、最高 6 つのパラメータを渡すことができます。例 12-7「Dialog キーワード」は、dialog キーワードを使用して 2 つのメニュー項目を追加する方法を示しています。

例 12-7.　Dialog キーワード

```
menuitem "My Functions";
dialog InitTarget(startAddress "Starting Address", EndAddress "End
Address")
{
  statements
}
dialog LoadMyProg()
{
  statements
}
```

上記の例では、GEL メニューの下に、次の項目が副選択項目として追加されます。

InitTarget コマンドを使用すると、「Function: InitTarget」ダイアログ・ボックスが表示され、開始アドレスと終了アドレスを入力するように求められます。



入力フィールドに値を入力し、「Execute」をクリックすると、これらのパラメータをもつ GEL 関数を呼び出します。

12.6.3 slider キーワード

GEL 関数を「GEL」メニューに追加するには、slider キーワードを使用することもできます。「GEL」メニューから関数を選択すると、GEL 関数に渡された値を制御するスライダ・オブジェクトが表示されます。スライダの位置を動かすたびに、スライダの新しい位置を反映する新しいパラメータ値をもつ GEL 関数が呼び出されます。slider GEL 関数には、1 つのパラメータしか渡すことができません。slider GEL 関数の構文は、次のとおりです。

```
slider param_definition( minVal, maxVal, increment, pageIncrement,
        paramName )
{
    statements
}
```

param_definition	スライダ・オブジェクト上に印刷されるパラメータの説明
minVal	整定数で、スライダの位置が最低レベルであるときに、関数に渡される値を指定します。
maxVal	整定数で、スライダの位置が最高レベルであるときに、関数に渡される値を指定します。
increment	整定数で、スライダの位置が 1 つ動かされるたびに、値に追加される増分を指定します。
pageIncrement	整定数で、スライダが 1 ページ動くたびに、値に追加される増分を指定します。
paramName	パラメータ定義で、関数内部で使用されます。

例 12-8「Slider キーワード」では、slide キーワードを使用して、ボリューム制御スライダを追加しています。

例 12-8. Slider キーワード

```
menuitem "My Functions";
slider VolumeControl(0, 10, 1, 1, volume)
{
  /* initialize the target variable with the parameter passed
     by the slider object. */
  targVarVolume = volume;
}
```

12.7「Output」ウィンドウのアクセス

Code Composer Studio の出力ウィンドウに結果を表示するために、複数の GEL 組み込み関数を使用することができます。

これらのコマンドは、以下のことを実行できます。

- ❏ 無限数の出力ウィンドウを作成する
- ❏ スクロール・ウィンドウ、または固定ウィンドウを作成する
- ❏ 任意のウィンドウに出力をパイプ接続する
- ❏ マルチカラーで表示する
- ❏ 強調表示テキストを変更する
- ❏ 実際のターゲットまたはシミュレーション・ターゲットから、フォーマットされた文字列を表示する

上記のタスクを実行できるようにするコマンドは、次のとおりです。

GEL_OpenWindow	出力ウィンドウを開きます。
GEL_CloseWindow	既存の出力ウィンドウをクローズします。
GEL_TargetTextOut	フォーマットされたターゲット文字列を出力します。
GEL_TextOut	出力ウィンドウにテキストを表示します。

12.8 始動時の GEL 関数の自動実行

GEL 関数を使用すると、ユーザのニーズに合わせて Code Composer Studio 環境を設定できるようになります。Code Composer Studio を始動するたびに、自分の環境を設定したい場合があります。毎回「File」→「Load Gel」を使用して GEL ファイルをロードし、それから GEL 関数を実行する代りに、始動時に GEL ファイル名を Code Composer Studio に渡すことができます。この方法では、指定された GEL ファイルをスキャンし、ロードするように、Code Composer Studio に指示します。さらに、GEL ファイルを単にロードするだけでは十分でなく、関数も実行したい場合があります。関数を実行するには、指定されたファイルに含まれる GEL 関数のどれか 1 つの名前を StartUp() にします。GEL ファイルが Code Composer Studio にロードされると、Code Composer Studio は、StartUp() として定義された関数を検索します。Code Composer Studio は、ファイル内でこの関数を検出すると、自動的にこの関数を実行します。

GEL 関数を自動的にロードし、実行する方法 (Windows 95/NT)

1) エクスプローラで、Code Composer Studio の実行プログラムを選択します。
2) 実行プログラムを右クリックし、コンテキスト・メニューから「ショートカットの作成 (S)」を選択します。
3) 作成されたショートカット上で右クリックし、コンテキスト・メニューから 「プロパティ (R)」を選択します。
4) 「ショートカットのプロパティ」ダイアログ・ボックスで、「ショートカット」タブを選択します。「リンク先 (T)」フィールドに、Code Composer Studio 実行プログラムのパス名とファイル名が表示されます。たとえば、次のとおりです。

   c:\composer\cc_app.exe

5) このパス名の末尾に、GEL 関数が入っている GEL ファイルの名前を追加します。たとえば、次のとおりです。

   c:\composer\cc_app.exe myfile.gel

こうすると、Code Composer Studio のショートカット・アイコンをダブルクリックするたびに、GEL ファイル myfile.gel が自動的にスキャンされ、Code Composer Studio にロードされるようになります。StartUp() として定義された GEL 関数がある場合は、その GEL 関数も実行されます。例 12-9「Startup GEL 関数」は、始動時にロードできる標準的な GEL ファイルを示しています。

例 12-9. Startup GEL 関数

```
StartUp()
{
  /*Everything in this function will be executed on startup*/
  /*turn on our memory map*/
  GEL_MapOn();
  GEL_MapAdd(0, 0, 0xF000, 1, 1);
  GEL_MapAdd(0, 1, 0xF000, 1, 1);
}
dialog LoadMyFile()
{
  /* load my coff file, and start at main */
  GEL_Load("myfile.out");
  GEL_Go(main);
}
```

上記の例では、Code Composer Studio を始動するたびにメモリ・マッピング機能がオンになり、LoadMyFile() 関数が GEL メニューに追加されます。

12.9 式待ち行列の表示

すべての GEL 関数および式は、式の計算機能 (evaluator) を使用して計算されます。必要に応じて、多くの式を計算機能への待ち行列に入れることができます。「View」→「Expression List」の順に選択します。「Expressions Executing」ダイアログ・ボックスが表示されます。このダイアログ・ボックスでは、式の計算機能によって現在計算されている式を表示することができます。

式の計算機能が現在計算している式を中止するには、その式を選択し、「Abort」をクリックします。この Abort は、無限ループに入っているか、実行に時間がかかり過ぎている GEL 関数を実行している場合に便利です。

12.10 組み込み GEL 関数

実際のターゲットまたはシミュレーション・ターゲットの状態を制御したり、メモリ位置にアクセスしたり、出力ウィンドウに結果を表示したりできるようにする、複数の組み込み GEL 関数があります。

すべての GEL 組み込み関数の前には、プレフィックス GEL_ が付いていて、ユーザ定義の GEL 関数と混同されないようになっています。GEL 組み込み関数のすべての呼び出しの前に GEL_ が付かないようにしたいときには、独自の名前を使って関数を定義し、その関数内で GEL 組み込み関数を呼び出すようにします。たとえば、次の GEL 関数では、Load と入力するだけで、GEL_Load() 組み込み関数を呼び出すことができるようになります。

```
Load(a)
{
  GEL_Load(a);
}
```

注:

複数の GEL **文から構成されているすべての組み込み** GEL **関数とユーザ定義** GEL **関数は、**Code Composer Studio **の** 「GEL」**ツールバーから直接起動することができます。このツールバーは、**expression **フィールドと** 「Execute Command」**ボタンで構成されています。任意の** GEL **文またはユーザ定義の関数を起動するには、このフィールド・ボックスに適切な関数呼び出しを入力し、**「Execute Command」**を押して、その式を評価します。**「expression」**ダイアログ・ボックスには、最近起動された** GEL **文またはユーザ定義関数の履歴が保持されています。スクロール・ボタンを使用すると、これらの履歴の中から任意のものを選択することができます。この** 「GEL」**ツールバーはデフォルトでは表示されませんが、**「View」→「Gel Toolbar」**を選択して、オン／オフを切り換えることができます。**

GEL_Animate() DSP ターゲットをアニメーション表示する

構文 GEL_Animate();

パラメータ なし

解説 この関数は、DSP ターゲットのアニメーション表示を開始します。

例 GEL_Animate();

参照先 GEL_Go、GEL_Halt、GEL_Run

GEL_BreakPtAdd() ブレークポイントを追加する

構文 GEL_BreakPtAdd(*address, "Condition"*);

パラメータ *address*: (必須) ブレークポイントの位置を指定します。

Condition: (オプション) 引用符で囲みます。条件付きブレークポイントで使用される条件

解説 この関数は、特定アドレスにソフトウェア・ブレークポイントを設定します。条件が指定された場合には、その条件が真の場合にだけブレークポイントは有効になり、プログラムの実行が停止します。このアドレスには、絶対アドレス、任意の C 式、C 関数の名前、またはアセンブリ言語ラベルの名前を指定することができます。

例
```
GEL_BreakPtAdd(0x2000);
GEL_BreakPtAdd(TargetLabel + 100);
GEL_BreakPtAdd(0x2000, "a < b");
```

参照先 GEL_BreakPtDel、GEL_BreakPtReset

GEL_BreakPtDel() ブレークポイントを削除する

構文 GEL_BreakPtDel(*address*);

パラメータ *address*: (必須)

解説 この関数は、特定アドレスのソフトウェア・ブレークポイントを消去します。そのアドレスに設定されているソフトウェア・ブレークポイントがない場合は、何も行われません。

このアドレスには、絶対アドレス、任意の C 式、C 関数の名前、またはアセンブリ言語ラベルの名前を指定することができます。

例
```
GEL_BreakPtDel(0x2000);
GEL_BreakPtDel(TargetLabel + 100);
```

参照先 GEL_BreakPtAdd、GEL_BreakPtReset

GEL_BreakPtReset() すべてのブレークポイントを消去する

構文 GEL_BreakPtReset();

パラメータ なし

解説 この関数は、すべてのソフトウェア・ブレークポイントを消去します。

例
```
GEL_BreakPtReset();
```

参照先 GEL_BreakPtAdd、GEL_BreakPtDel

GEL_CloseWindow() 出力ウィンドウをクローズする

構文 GEL_CloseWindow(*"windowName"*);

パラメータ *windowName*: (必須) 引用符で囲みます。クローズされるウィンドウの名前

解説 この関数は、既存の出力ウィンドウをクローズします。windowName には、クローズされるウィンドウの名前を指定します。このパラメータは、GEL_OpenWindow() 関数で指定したパラメータと同じでなければなりません。

例
```
GEL_CloseWindow("My Window");
GEL_CloseWindow("Macro Output");
```

参照先 GEL_OpenWindow、GEL_TargetTextOut、GEL_TextOut

GEL_Exit() アクティブな制御ウィンドウをクローズする

構文 GEL_Exit();

パラメータ なし

解説 この関数は、アクティブな制御ウィンドウをクローズします。シングル・プロセッサ・システムの場合、この関数は、Code Composer Studio を完全にクローズします。

例 GEL_Exit();

参照先 GEL_CloseWindow

GEL_Go() 指定されたアドレスまで実行する

構文 GEL_Go(*address*);

パラメータ *address*: (オプション) 停止アドレス

解説 GEL_Go 関数は、プログラム内の特定ポイントまで、コードを実行します。この address パラメータは、プログラム・メモリのアドレスとして扱われます。アドレスを指定しないと、GEL_Go は GEL_Run 関数と等しくなります。

ご使用のコードが、指定されたアドレスに到達しない場合があります。この場合、この GEL 関数は完了しません。このような式を中止するには、「View」→「Expression List」の順に選択して、計算されているすべての式を表示します。GEL_Go() 式を選択し、「Abort」をクリックしてください。

例
```
GEL_Go();
GEL_Go(main);
```

参照先 GEL_Run、GEL_Halt

GEL_Halt() 実行を停止する

構文 GEL_Halt();

パラメータ なし

解説 この関数は、ターゲット・プログラムが実行している場合に、このプログラムを停止させます。

例 GEL_Halt();

参照先 GEL_Go、GEL_Run、GEL_RunF

GEL_Load() データ・ファイルをロードする

構文 GEL_Load("*fileName*","*cpuName*");

パラメータ *fileName*: (必須) 引用符で囲みます。ロードされるオブジェクト・ファイル

cpuName: (オプション) 引用符で囲みます。ファイルをロードする先の CPU の名前（マルチプロセッサ環境で便利）

解説 この関数は、オブジェクト・ファイルと、それに関連したシンボル・テーブルの両方をメモリにロードします。このファイルが Code Composer Studio のカレント・ディレクトリ内にない場合は、その文字列内で絶対パス名を指定してください。fileName パラメータにバックスラッシュを入れるには、バックスラッシュを 2 つ連結させる必要があることに注意してください。cpuName パラメータは、Code Composer Studio マルチプロセッサ・セットアップ内で設定されている CPU 名と一致しなければなりません。シングル・プロセッサ・システムでは、このフィールドに入力する必要はありません。

例 GEL_Load("c:\\workdir\\test.out", “cpu_b”);

参照先 GEL_SymbolLoad

GEL_MapAdd() メモリ・マップへ追加する

構文

GEL_MapAdd(*Address, Page, Length, Readable, Writeable*);

パラメータ

Address: (必須) メモリ内の範囲の開始アドレス。このパラメータには、絶対アドレス、任意の C 式、C 関数の名前、またはアセンブリ言語ラベルを指定することができます。

Page: (必須) 埋め込むメモリのタイプを識別します。

メモリ・タイプ	Page パラメータ
プログラム・メモリ	0
データ・メモリ	1
I/O 空間	2

1 つのタイプのメモリしかないプロセッサの場合は、このパラメータに 0 を使用します。シミュレーション・ターゲットの場合、I/O 空間を示すパラメータ 2 はサポートされません。

Length: (必須) 範囲の長さを定義します。このパラメータは、任意の C 式にすることができます。

Readable: (必須) そのメモリ範囲が読み取り可能かどうかを定義します。
0 - 読み取り不可
1 - 読み取り可能

Writeable: (必須) そのメモリ範囲が書き込み可能かどうかを定義します。
0 - 書き込み不可
1 - 書き込み可能

解説

この関数は、ターゲット・メモリの範囲に対する読み取り／書き込み権限を、メモリ・マップに追加します。その範囲が既存の項目とオーバラップする場合、メモリ・マップでは新しい範囲の属性が優先されます。

例

GEL_MapAdd(0x1000, 0, 0x300, 1, 1);

参照先

GEL_MapDelete、GEL_MapOn、GEL_MapOff、GEL_MapReset

GEL_MapDelete() メモリ・マップから削除する

構文

GEL_MapDelete(*Address, Page*);

パラメータ

Address: (必須) メモリ・マップから削除されるメモリ範囲を識別します。Address は、削除対象のメモリ・マップ範囲内の、任意の有効なアドレスにすることができます。このパラメータには、絶対アドレス、任意の C 式、C 関数の名前、またはアセンブリ言語ラベルを指定することができます。

Page: (必須) 埋め込むメモリのタイプを識別します。

メモリ・タイプ	Page パラメータ
プログラム・メモリ	0
データ・メモリ	1
I/O 空間	2

1 つのタイプのメモリしかないプロセッサの場合は、このパラメータに 0 を使用します。シミュレーション・ターゲットの場合、I/O 空間を示すパラメータ 2 はサポートされません。

解説

この関数は、メモリ・マップからメモリのある範囲を削除します。この範囲が削除されると、Code Composer Studio デバッガはそのターゲットとの間で読み取りも書き込みも行いません。読み取り不可のメモリ位置を表示しようとすると、デバッガは、ターゲットの値を表示する代わりに、デフォルト値を表示します。

例

GEL_MapDelete(0x1000, 0);

参照先

GEL_MapAdd、GEL_MapOn、GEL_MapOff、GEL_MapReset

GEL_MapOff() メモリ・マップをディセーブルにする

構文 GEL_MapOff();

パラメータ なし

解説 この関数は、メモリ・マップをディセーブルにします。メモリ・マップをディセーブルにすると、ターゲットでバス障害が発生する可能性があることに注意してください。これは、Code Composer Studio デバッガが、存在しないメモリにアクセスしようとする場合があるからです。デフォルトでは、電源投入後、メモリ・マップはオフになっています。

例 GEL_MapOff();

参照先 GEL_MapAdd、GEL_MapOn、GEL_MapDelete、GEL_MapReset

GEL_MapOn() メモリ・マップをイネーブルにする

構文 GEL_MapOn()

パラメータ なし

解説 この関数は、メモリ・マップをイネーブルにします。Code Composer Studio デバッガは、読み取り不可のマップ・セグメントからの読み取りも、書き込み不可のマップ・セグメントへの書き込みも行いません。マップを初めてオンにするときには、メモリ範囲全体は、読み取り不可でかつ書き込み不可であるものと見なされます。デバッガが有効なセクションにアクセスできるようにするには、(GEL_MapAdd() 関数を使用して) メモリ・セクションを追加する必要があります。デフォルトでは、電源投入後、メモリ・マップはオフになっています。

例 GEL_MapOn();

参照先 GEL_MapAdd、GEL_MapOff、GEL_MapDelete、GEL_MapReset

GEL_MapReset() メモリ・マップをリセットする

構文 GEL_MemoryReset()

パラメータ なし

解説 この関数は、すべてのメモリを読み取り不可および書き込み不可にして、メモリ・マップをリセットします。

例 MapReset();

参照先 GEL_MapAdd、GEL_MapOff、GEL_MapOn、GEL_MapDelete

GEL_MemoryFill() メモリ・ブロックに埋め込む

構文

GEL_MemoryFill(*Startaddress, Page, Length, Pattern*)

パラメータ

Startaddress: (必須) ブロック内の最初のアドレス

Page: (必須) 埋め込むメモリのタイプを識別します。

メモリ・タイプ	Page パラメータ
プログラム・メモリ	0
データ・メモリ	1
I/O 空間	2

1 つのタイプのメモリしかないプロセッサの場合は、このパラメータに 0 を使用します。シミュレーション・ターゲットの場合、I/O 空間を示すパラメータ 2 はサポートされません。

Length: (必須) 埋め込むワード数を定義します。

Pattern: (必須) ブロック内の各ワードに指定される値

解説

GEL_MemoryFill() は、ターゲット・メモリのブロックを、指定されたパターンで埋め込む場合に使用することができます。

例

GEL_MemoryFill(0x1000, 0, 0x100, 0xa5a5);

参照先

GEL_MemoryLoad、GEL_MemorySave

GEL_MemoryLoad() ファイルからメモリ・ブロックをロードする

構文

GEL_ MemoryLoad(*Startaddress, Page, Length, "fileName"*)

パラメータ

Startaddress: (必須) ブロック内の最初のアドレス

Page: (必須) 埋め込むメモリのタイプを識別します。

メモリ・タイプ	Page パラメータ
プログラム・メモリ	0
データ・メモリ	1
I/O 空間	2

1 つのタイプのメモリしかないプロセッサの場合は、このパラメータに 0 を使用します。シミュレーション・ターゲットの場合、I/O 空間を示すパラメータ 2 はサポートされません。

Length: (必須) 埋め込むワード数を定義します。

fileName: (必須) 引用符で囲みます。ターゲット・データをロードするファイルの名前

解説

GEL_MemoryLoad() を使用すると、指定されたファイルから、ターゲット・メモリのブロックをロードすることができます。このデータのブロックは、Startaddress、Page、および Length によって指定されます。ファイル名のファイル拡張子に .out が指定されている場合は、COFF 構文が使用されます。それ以外の場合、Code Composer Studio デバッガは、ファイル内のヘッダ情報を使用してファイル・構文を判別します。

例

```
GEL_MemoryLoad(0x1000, 1, 0x100, "c:\\workdir\\temp.dat");
```

参照先

GEL_MemorySave、GEL_MemoryFill

GEL_MemorySave() ファイルへメモリ・ブロックをセーブする

構文

GEL_MemorySave(*Startaddress, Page, Length, "fileName"*)

パラメータ

Startaddress: (必須) ブロック内の最初のアドレス

Page: (必須) 埋め込むメモリのタイプを識別します。

メモリ・タイプ	Page パラメータ
プログラム・メモリ	0
データ・メモリ	1
I/O 空間	2

1 つのタイプのメモリしかないプロセッサの場合は、このパラメータに 0 を使用します。シミュレーション・ターゲットの場合、I/O 空間を示すパラメータ 2 はサポートされません。

Length: (必須) 埋め込むワード数を定義します。

fileName: (必須) 引用符で囲みます。ターゲット・データをセーブするファイルの名前

解説

GEL_MemorySave() は、指定されたファイルに、ターゲット・メモリのブロックをセーブする場合に使用できます。このデータのブロックは、Startaddress、Page、および Length によって指定されます。ファイル名のファイル拡張子に .out が指定されている場合は、COFF 構文が使用されます。それ以外の場合は、C-Style Hex が使用されます。

例

GEL_MemorySave(0x1000, 1, 0x100, "c:\\workdir\\temp.dat");

参照先

GEL_MemoryLoad、GEL_MemoryFill

GEL_OpenWindow() 表示用に出力ウィンドウをオープンする

構文

GEL_OpenWindow(*"windowName", windowType, MaxLines)*;

パラメータ

windowName: (オプション) 引用符で囲みます。出力ウィンドウのユーザ定義名。名前が指定されない場合は、Macro Output と仮定されます。

windowType: (オプション) 作成される出力ウィンドウのタイプ
0 - スクロール・ウィンドウ
1 - 非スクロール・ウィンドウ

このパラメータに値を指定しない場合は、スクロール・ウィンドウと仮定されます。

MaxLines: (オプション) 非スクロール・ウィンドウを指定する場合、このパラメータに、ウィンドウが保持できる最大行数を指定します。スクロール・ウィンドウを指定する場合、この引数は無視されます。

解説

この関数は、windowName という名前の出力ウィンドウを作成します。この出力ウィンドウにアクセスする場合、他の GEL 関数は、この windowName を使用します。無限数の出力ウィンドウを作成できます。

例

```
GEL_OpenWindow();
GEL_OpenWindow("Macro Output", 1, 20);
```

参照先

GEL_CloseWindow、GEL_TargetTextOut、GEL_TextOut

GEL_PatchAssembly() アセンブリ・パッチ文字列を使用してメモリにパッチする

構文

GEL_PatchAssembly(*Address, Page,* "*PatchString*");

パラメータ

Address: (必須) アセンブリ命令をパッチする先のアドレス

Page: (必須) 埋め込むメモリのタイプを識別します。

メモリ・タイプ	Page パラメータ
プログラム・メモリ	0
データ・メモリ	1
I/O 空間	2

1 つのタイプのメモリしかないプロセッサの場合は、このパラメータに 0 を使用します。

PatchString: (必須) メモリにパッチするアセンブリ文字列

解説

この関数は、指定されたメモリ上のアドレスにアセンブリ文字列をロードします。

例

GEL_PatchAssembly(0x1000,1, "LAR AR4,#01h");

> **注 : 'C6000 プロセッサ**
> **パッチ・アセンブリは、**'C6000 **プロセッサ（実際、またはシミュレートされる）に対してサポートされません。**

GEL_ProjectBuild() 現行プロジェクトをビルドする

構文

GEL_ProjectBuild();

パラメータ

なし

解説

この関数は、現行のプロジェクトをビルドします。

例

GEL_ProjectBuild();

参照先

GEL_ProjectLoad、GEL_ProjectRebuildAll

GEL_ProjectLoad() プロジェクトをロードする

構文 GEL_ProjectLoad(“*fileName*”);

パラメータ *fileName*: (必須) ロードするプロジェクト・ファイル

解説 この関数は、指定されたプロジェクト・ファイルをロードします。

例 GEL_ProjectLoad(“d:\\mydir\\myproject.mak”);

参照先 GEL_ProjectBuild、GEL_ProjectRebuildAll

GEL_ProjectRebuildAll() 現行プロジェクトを完全にリビルドする

構文 GEL_ProjectRebuildAll();

パラメータ なし

解説 この関数は、現行のプロジェクトを完全にリビルドします。

例 GEL_ProjectRebuildAll();

参照先 GEL_ProjectBuild、GEL_ProjectLoad

GEL_Reset() ターゲット・システムをリセットする

構文 GEL_Reset();

パラメータ なし

解説 Reset() 関数は、ターゲット・システムをリセットし、モニタを再ロードします。これは、ソフトウェア・リセットであることに注意してください。

例 GEL_Reset();

参照先 GEL_Restart、GEL_Run、GEL_Halt

GEL_Restart() プログラム・エントリ・ポイントに PC をリセットする

構文

GEL_Restart();

パラメータ

なし

解説

GEL_Restart() 関数は、プログラムを、そのエントリ・ポイントにリセットします。これは、プログラム（シンボル情報が入っている）が、ターゲットにロードされているものと仮定します。

例

GEL_Reset();

参照先

GEL_Reset、GEL_Run、GEL_Halt

GEL_Run() コードを実行する

構文

GEL_Run(["*Condition*"]);

パラメータ

Condition: (オプション) 引用符で囲みます。ターゲットの実行中に満足していなければならない条件。ターゲットの実行中にブレークポイントに達し、その条件が偽であると評価されると、デバッガは、そのアドレスで停止します。

解説

この関数は、ターゲット上でコードの実行を開始します。条件が指定される場合、この run 関数は条件付き run 文になります。つまり、この文が真である間、実行は継続されます。この文は、遭遇する各ブレークポイントで計算されます。

例

```
GEL_Run();
GEL_Run("A != B");
```

参照先

GEL_Restart、GEL_Go、GEL_Halt、GEL_RunF

GEL_RunF() 自由に実行する

構文 GEL_RunF();

パラメータ なし

解説 この関数は、ブレークポイントをディセーブルにした後、ターゲット上でコードの実行を開始します。また、この関数は、ターゲット・システムから切り離されます。これは、ターゲット・システム上でハードウェア・リセットを実行する必要がある場合、または JTAG ケーブルや MPSD ケーブルを切り離す必要がある場合に便利です。ターゲット・システム上でアクセスが要求される（たとえば、メモリの読み取り）場合、またはユーザがプロセッサを停止させる場合、Code Composer Studio デバッガは、ターゲット・システムに再接続し、ブレークポイントをイネーブルにします。

例 GEL_RunF();

参照先 GEL_Restart、GEL_Go、GEL_Halt、GEL_Run

GEL_SymbolLoad() シンボル情報のみをロードする

構文 GEL_SymbolLoad("*fileName*", “*cpuName*”);

パラメータ

fileName: (必須) 引用符で囲みます。シンボル情報が入っている COFF ファイル

cpuName: (オプション) 引用符で囲みます。シンボル情報をロードする先の CPU の名前（マルチプロセッサ環境で便利）

解説 この関数は、指定されたオブジェクト・ファイルのシンボル情報をロードします。GEL_SymbolLoad() は、デバッガがオブジェクト・コードをロードできないか、ロードする必要がないデバッグ環境（たとえば、コードが ROM に入っている場合）で便利です。エントリ・ポイントは変更されません。

例 GEL_SymbolLoad("d:\\mydir\\myfile.out", “cpu_b”);

参照先 GEL_Load

GEL_System()　DOS コマンドを実行する

構文

GEL_System(*"dosCommand", param1, param2, .. param4*);

パラメータ

dosCommand: (必須) 実行される DOS コマンド（オプションの書式制御文字列が含まれている場合があります）

param1..param4: (オプション) 書式制御文字列が検出されると、dosCommand 内で置換される追加パラメータ。これらのパラメータを使用すると、ユーザは、ターゲットからの値、またはユーザから渡された値を DOS コマンドに渡すことができます。

解説

GEL_System 関数を使用すると、Code Composer Studio 内から DOS コマンドを実行できるようになります。DOS コマンドの出力は、Code Composer Studio 内の出力ウィンドウに送られます。DOS コマンドだけが、テキスト出力を作成するコマンドであり、いったん実行を開始すると、追加のユーザ入力は必要ありません。

実行される DOS コマンドは、実際には、dosCommand と追加パラメータ (parm1..parm4) によって決定される書式化された文字列です。これにより、ユーザは、（DSP ターゲット上で定義される値を含めて）追加パラメータを DOS コマンドに渡すことができます。

書式制御文字列は、常にパーセント記号 (%) で始まり、左から右に読み取られます。最初の書式制御文字列（ある場合）が検出されると、コマンドの後の最初の引数の値が変換され、dosCommand に書き込まれます。2 番目の書式制御文字列では、2 番目の引数が変換され、書き込まれ、以下同様になります。書式制御文字列よりも多くの引数がある場合、余分の引数は無視されます。

この関数 GEL_System() 関数は、専有技術を使用して実装されていて、Code Composer Studio の機能を拡張するために使用できます。この関数を使用すると、バックグラウンドでタスク（たとえばコンパイル）を実行し、Code Composer Studio の出力ウィンドウに、その結果をパイプ接続することができます。

例

```
GEL_System("dir");

GEL_System("dir *.dat") ; は、
GEL_System("dir %s", "*.dat"); と等しくなります。

GEL_System("myfunc %f %d %s", targVar, 3, "-ol");
```

targVar が、DSP ターゲット上で定義された変数であり、その値が 3.14 であると仮定すると、実行される DOS コマンドは、次のようになります。

```
>>myfunc 3.14 3 -ol
```

書式制御文字列のフィールド

書式制御文字列は、次の形式で指定します。

%type

C の場合と異なり、GEL 書式制御文字列には、パーセント記号と型を示す文字（たとえば %s）だけが含まれています。型を示す文字は、（以下に詳細があるように）関連した引数が文字列として解釈されるか、数値として解釈されるかを決定します。

型を示す文字 type	型	出力フォーマット
d	int	符号付き 10 進整数
u	int	符号なし 10 進整数
x	int	16 進形式
f	double	形式 「-」dddd.dddd の符号付き値
e	double	形式 「-」d.dddd e 「sign」ddd の符号付き値
s	string	最初のヌル文字まで表示される文字。この書式制御型に渡される文字列は、ホスト上で宣言された定数文字列でなければなりません。

GEL_TargetTextOut() ターゲットの書式化された文字列を表示する

構文

GEL_TargetTextOut(*startAddress, Page, maxLength, format, "windowName", textColor, lineNumber, appendToEnd, changeHighlight*);

パラメータ

startAddress: (必須) 事前に構文された文字列が入っている、ブロックの最初のアドレス

Page: (オプション) メモリのタイプを識別します。

メモリ・タイプ	Page パラメータ
プログラム・メモリ	0
データ・メモリ	1
I/O 空間	2

1 つのタイプのメモリしかないプロセッサの場合は、このパラメータに 0 を使用します。ページ番号のデフォルト値は、0 です。シミュレーション・ターゲットの場合、I/O 空間を示すパラメータ 2 はサポートされません。

maxLength: (オプション) ブロックの長さが 400 バイトを超える場合のブロックの最大長。ターゲット上の書式化された文字列は、ヌルで終了する文字列でなければなりません。しかし、ヌルが検出されない場合は、その文字列の 400 バイト、または maxLength バイト（どちらか大きい方）だけが表示されます。

format: (オプション) ターゲット上でテキストがパック・構文であるか、アンパック・構文であるかを指定します。
0 - ASCII 文字 (バイト)

1 - ビッグ・エンディアン・フォーマットを使用するパック済み ASCII 文字 ― 先頭文字は、ターゲットの最大重みバイト内にあります。

2 - リトル・エンディアン・フォーマットを使用するパック済み ASCII 文字 ― 先頭文字は、ターゲットの最小重みバイト内にあります。

windowName: (オプション) 出力が送られる先のウィンドウの名前。このウィンドウが開いていない場合は、GEL_OpenWindow() 関数のデフォルトのパラメータを使用して、ウィンドウが作成されます（12-30 ページ「GEL_OpenWindow()」を参照）。

textColor: (オプション) テキストの表示カラー
0 - 黒色
1 - 青色
2 - 赤色

lineNumber: (オプション) 非スクロール・ウィンドウの場合、表示が開始されるウィンドウの行番号

appendToEnd: (オプション) 非スクロール・ウィンドウに表示する場合にだけ使用されるフラグ。このフラグが 1 に設定されると、このテキストは、既存の行に追加されます。それ以外の場合は、既存の行が消去され、新しい行が出力ウィンドウに置かれます。

changeHighlight: (オプション) 非スクロール・ウィンドウに表示し、特定行を置き換える場合にだけ使用されるフラグ。このフラグがイネーブルな場合、新しいテキストが旧テキストと比較されます。変更が検出されると、新しいテキストが強調表示されます。

解説

この関数は、Code Composer Studio の出力ウィンドウに書式化された文字列を表示する場合に使用されます。この文字列は、ターゲット上にすでに存在している必要があり、ヌルで終了していなければなりません。

例

```
GEL_TargetTextOut(0x800);
GEL_TargetTextOut(0x1000, 0, 400, 1, "My Window", 1);
```

参照先

GEL_CloseWindow、GEL_OpenWindow、GEL_TextOut

GEL_TextOut() 出力ウィンドウへのテキストを表示する

構文

GEL_TextOut(*"Text", "windowName", textColor, lineNumber, appendToEnd, parm1, parm2, .. parm4*);

パラメータ

Text: (必須) 表示対象の書式化されたテキスト（書式制御文字列を含む)。書式制御文字列の数は、検出される追加パラメータ数（*parm1.. parm4*) と一致しなければなりません。

windowName: (オプション) 出力が送られる先のウィンドウの名前。このウィンドウが開いていない場合は、GEL_OpenWindow() 関数のデフォルトのパラメータを使用して、ウィンドウが作成されます (12-30 ページ「GEL_OpenWindow()」を参照)。

textColor: (オプション) テキストの表示カラー
0 - 黒色
1 - 青色
2 - 赤色

lineNumber: (ウィンドウが非スクロール・ウィンドウの場合のオプション) このパラメータは、表示が開始されるウィンドウの行番号を指定します。

appendToEnd: (オプション) 非スクロール・ウィンドウに印刷する場合にだけ使用されるフラグ。このフラグが 1 に設定されると、このテキストは既存の行に追加されます。それ以外の場合は、既存の行が消去され、新しい行が出力ウィンドウに置かれます。

param1..param4: (オプション) 書式制御文字列が検出されると、テキスト内で置換される追加パラメータ。これらのパラメータを使用すると、ユーザは、ターゲットからの値、またはユーザから渡された値を、出力ウィンドウに渡すことができます。

書式制御文字列は常に、パーセント記号 (%) で始まり、左から右に読み取られます。最初の書式制御文字列（ある場合）が検出されると、最初の引数 (parm1) の値が変換され、Code Composer Studio の出力ウィンドウに表示されます。2 番目の書式制御文字列では、2 番目の引数が変換され、表示され、以下同様になります。書式制御文字列よりも多くの引数がある場合、余分の引数は無視されます。

解説

この関数は、指定された出力ウィンドウに、固定の文字列を表示します。この関数は、メッセージを表示する場合に理想的な関数です。

例

```
GEL_TextOut("All Tests Passed\n");
GEL_TextOut("Failed Memory Test\n","Diagnostic Results", 2 );
GEL_TextOut("Tests Executed: %d, Tests Passed %d ",targExe, targPass);
```

参照先

GEL_OpenWindow、GEL_TargetTextOut、GEL_CloseWindow

書式制御文字列のフィールド

書式制御文字列は、次の形式で指定します。

%type

C の場合と異なり、GEL 書式制御文字列には、パーセント記号と型を示す文字（たとえば %s）だけが含まれています。型を示す文字は、（以下に詳細があるように）関連した引数が文字列として解釈されるか、数値として解釈されるかを決定します。

型を示す文字 type	型	出力フォーマット
d	int	符号付き 10 進整数
u	int	符号なし 10 進整数
x	int	16 進形式
f	double	形式 「-」dddd.dddd の符号付き値
e	double	形式 「-」d.dddd e 「sign」ddd の符号付き値
s	string	最初のヌル文字まで表示される文字。この書式制御型に渡される文字列は、ホスト上で宣言された定数文字列でなければなりません。

GEL_WatchAdd() 「Watch」ウィンドウへ式を追加する

構文 GEL_WatchAdd(*"expression", "label"*);

パラメータ *expression*: (必須) 「Watch」ウィンドウに追加される式。この式には、「Watch」ウィンドウのフォーマットによって指定される書式化された文字列が含まれている場合があります。

label: (オプション) watch エントリの表示に使用されるラベル

解説 この関数は、GEL 環境から「Watch」ウィンドウに式を追加する場合に使用できます。詳細は、「Watch」ウィンドウの解説を参照してください。

例
```
GEL_WatchAdd("*(int *)0x1000,x", "Task Number");
GEL_WatchAdd("i");
```

参照先 GEL_WatchDel、GEL_WatchReset

GEL_WatchDel() 「Watch」ウィンドウから既存の式を削除する

構文 GEL_WatchDel("*expression*");

パラメータ *expression*: (必須) 「Watch」ウィンドウから削除したい式

解説 この関数は、「Watch」ウィンドウから既存の式を削除します。この式は、「Watch」ウィンドウ内の式と正確に同じものでなければなりません。

例
```
GEL_WatchDel("*(int *)0x1000,x");
```

参照先 GEL_WatchAdd、GEL_WatchReset

GEL_WatchReset() 「Watch」ウィンドウをクリアする

構文 GEL_WatchReset();

パラメータ なし

解説 この関数は、「Watch」ウィンドウからすべての式を消去します。

例
```
GEL_WatchReset();
```

参照先 GEL_WatchAdd、GEL_WatchDel

GEL_XMDef() 拡張メモリ範囲を定義する (C548 のみ)

構文

GEL_XMDef(*Map, RegAddr, Type, Start, Mask*);

パラメータ

Map : 拡張メモリ・マッピング用のメモリ空間のタイプ

メモリ・タイプ	Map パラメータ
プログラム空間	0
データ空間	1

RegAddr : マッパ・レジスタ (XPC) の位置 (0x1E)

Type : マッパ・レジスタのメモリ・タイプ

メモリ・タイプ	Type パラメータ
プログラム空間	0
データ空間	1

Start : マップされたメモリ範囲の先頭 (OVLY が 1 の場合は、0x8000 を使用)

Mask : マッパ・レジスタのサイズを表すビット・マスク

解説

この関数は、'C548/'C549 プロセッサ用の拡張メモリ・アドレス範囲を定義するのに使用されます。

例

GEL_XMDef(0,0x1E,1,0x8000,0x7EF);

参照先

GEL_XMOn

注 : シミュレータ - GEL_XMDef のサポートなし
この関数は、シミュレータではサポートされません。

GEL_XMOn()　拡張メモリ・マッピングをイネーブルにする (C548/C549 のみ)

構文　GEL_XMOn();

パラメータ　なし

解説　この関数は、'C548/'C549 プロセッサ用の拡張メモリ・マッピングをイネーブルにするのに使用されます。

例　GEL_XmOn();

参照先　GEL_XMDef

> **注 : シミュレータ - GELXMOn のサポートなし**
> **この関数は、シミュレータではサポートされません。**

付録 A

問い合わせの多い質問

ここでは、Code Composer Studio の使用に関して、問い合わせの多い質問の概要を記載します。

項目 **ページ**

A.1 Code Composer Studio のインストレーションとロード

1) **初めて Code Composer Studio を実行しようとするときに、次のようなエラー・メッセージが断続的に表示される。**

ERROR MESSAGE 1:「Can't Initialize Target DSP
Trouble with JTAG controller
Check your Cabling and your Multiprocessing Configuration」

ERROR MESSAGE 2:「Can't Initialize Target DSP
I/O Port = <address>」

このエラーが検出された場合には、トラブルシューティングの際にいくつかの点を考慮しなければなりません。このエラー・メッセージの原因となった可能性が最も高いものから順に示します。

DSP ターゲットの I/O 設定

a) DSP ターゲットが、無効な I/O アドレスで設定されている可能性があります。ターゲット・カード上の DIP スイッチの設定が、Code Composer Setup ユーティリティの実行時に設定された I/O アドレスと一致していることを確認してください。

b) ターゲットに対して設定された I/O アドレスが、競合する可能性があります。PC 上の他のハードウェアが、この I/O 設定を使用していないことを確認してください。Windows 95 を実行している場合は、次のようにして競合があるかどうかをチェックできます。「スタート」→「設定」→「コントロール パネル」→「システム」の順に選択し、「デバイス マネージャ」タブを選択します ― これは、システム構成の概要を示し、ターゲット・ボードとシステム上のその他のハードウェアとの間に、競合が存在するかどうかを示します。

Code Composer Setup の構成

a) Code Composer Setup ユーティリティの実行時に設定したデバイス・ドライバが、誤っている可能性があります。このドライバが、DSP ターゲットの構成に適したものであるかどうかを確認してください。

b) マルチプロセッシングの構成が、正しく設定されていない可能性があります。Code Composer Multiprocessing System の正しい構成方法は、Code Composer Setup オンライン・ヘルプを参照してください。

c) Code Composer Setup ユーティリティが、Code Composer Studio の実行プログラムと同じディレクトリに存在していない可能性があります。このセットアップ・プログラムが、実行プログラムと同じディレクトリにあることを確認してください。

DSP ターゲットのセットアップ

a) ご使用の DSP が 「ホールド」状態でも、「リセット」状態でもなく、正常に電源がオンになっていることを確認してください。

b) ターゲット・プロセッサ・ピンがアクティブであることを確認してください。デバッガが実行するには、ターゲット・プロセッサが 「レディ」状態でなければなりません。レディ信号線上にハードウェア障害がある場合は、可能であれば、プロセッサをマイクロコンピュータ・モードにしてから、システムをリセットし、デバッガを再び立ち上げてみてください。マイクロコンピュータ・モードでは、あらゆるメモリ・アクセスがチップ上で行われ、レディ信号は効力をもちません。詳細は、ご使用の装置の「使用者の手引き」を参照してください。

c) プロセッサのホールド・ピンがアクティブである可能性があります。b) と同じ問題です。

d) JTAG からの信号が明瞭でない可能性があります。エミュレータとターゲット・プロセッサとの間で高品質の信号をやり取りするためには、エミュレータ・ヘッダとプロセッサとの間のバッファのない距離が 6 インチ未満でなければなりません。この距離が 6 インチを超える場合は、エミュレーション信号をバッファする必要があります。

e) プロセッサがクロックを生成していない可能性があります。プロセッサは、正しいクロックを受信し、生成する必要があります。回路内のクロックとクロック・モードをチェックしてください。

f) EMU0/1 ピンがハイでない可能性があります。EMU0/1 ピンの値とリセット・ピンは、デバイス・ピンをオフにしたり、デバイス・テスト・モードを起動したりするのに使用できます。ユーザは、ターゲット・システム内の抵抗器を使用して、これらのピンをハイにする必要があります。

2) Code Composer Studio は 2000 年問題に対応しているか。

Code Composer Studio のフトウェアの動作に関する 2000 年問題はありません。Code Composer Studio と 2000 年問題の対応については、プロダクト・インフォメーション・センターにお問い合わせください。

Tel: 0120-81-0026

Fax: 0120-81-0036

Email: pic-japan@ti.com

または、次の Web サイトを参照してください。

http://www.ti.com/corp/docs/year2000

または、次の Web サイトで TI 製品マトリックスを参照してください。

http://www.ti.com/corp/docs/year2000/dspsds.htm

A.2 DSP プロジェクト管理システム

1) **Code Composer Studio 環境内から BUILD または COMPILE を起動しようとすると、次のエラー・メッセージが表示される。**

   「error 1010: can't initialize loader LINEXE_LOADER [1]」

   または 「Out of Memory」エラー・メッセージが表示される。何が起きているのか分からない。

   最も可能性が高い原因は、システム内で使用可能なコンベンショナル・メモリが不足していることです。DOS シェル内で mem と入力して、その DOS シェル内で使用できるコンベンショナル・メモリの量をチェックしてください。590K バイト以上の RAM が必要です。また、DOS シェルから TI ツールを起動してみるべきです ― Code Composer Studio がツールを起動するのに使用するのと同じコマンド行を入力してみてください。たとえば、ソース・コード・ファイルが「Mydirectory」内にある場合は、このディレクトリに移り、Code Composer Studio が TI ツールを起動するのに使用するコマンド行を入力してみてください。この問題を解決するには、コンベンショナル・メモリの一部を解放する必要があります。

2) **(Windows 95) Code Composer Studio 内から BUILD を起動すると、「Build」ウィンドウに次のエラー・メッセージが表示される。**

   「**General failure error reading drive E**」

   このエラーは、16 ビット TI コード生成ツールと 32 ビット Code Composer Studio アプリケーションとの間の相互作用によって発生します。WIN.INI ファイルに次の行を追加すると、この状態が改善されるはずです。

   [Code Composer Studio]
   BackgroundCompile = SAFE

3) **(Windows 95) Code Composer Studio 内から BUILD を起動すると、「Build」ウィンドウに、起動中のコード生成ツールが表示されている。しかし、構文エラーは（コンパイラ段階で）検出されず、実行可能な .out ファイルは構築されない。DOS シェルからコンパイラ、アセンブラ、およびリンカを起動すると、問題なく実行可能プログラムが生成される。**

   この問題は、TI 16 ビット・コード生成ツールと 32 ビット Code Composer Studio アプリケーションとの間の非互換性を表しています。WIN.INI ファイルに次の行を追加すると、この状態が解決されるはずです。

   [Code Composer Studio]
   BackgroundCompile = SAFE

4) **Code Composer Studio 内から BUILD、または COMPILE を起動すると、「Build」ウィンドウが表示されるが、空のままである。コンパイラも、アセンブラも、リンカも起動されず、このウィンドウは空のままである。**

この問題も、使用可能なコンベンショナル・メモリが不足する場合に生じます。詳細は、1) を参照してください。

5) **Code Composer Studio 環境内から BUILD、または COMPILE を起動すると、TI ツールが「Build」ウィンドウ内で起動されるときに、「bad command or filename」エラーが表示される。**

autoexec.bat ファイル (Windows 95) またはシステム環境変数 (Windows NT) 内で TI ツールに対して設定されたパスは、TI コード生成ツールの正しいディレクトリを示していなければなりません。正しいディレクトリに設定されていることを確認してください。

6) **リンカを起動するたびに、次のエラー・メッセージが表示される。**

「entry point symbol _c_int100 undefined」

このエラー・メッセージは、Code Composer Studio が提供するリンカ・オプション以外のオプションを使用していることが原因で表示されることがあります。これらのオプションは、autoexec.bat ファイル (Windows 95) またはシステム環境変数 (Windows NT) 内の環境変数を使用して設定されているか、Code Composer Studio の「Build Options」ダイアログ・ボックスに、ユーザ自身が入力したオプションによって設定されている可能性があります。解決法としては、autoexec.bat ファイル内の「set C_OPTION」行 (Windows 95) から、またはシステム環境変数 (Windows NT) から「-z」オプションを削除し、Code Composer Studio の「Build Options」ダイアログ・ボックスでユーザ自身が入力したアセンブラ・オプション、またはリンカ・オプションを削除することが考えられます。

7) **Code Composer Studio の使用時に設定したオプションが、プロジェクトの構築時に使用されない。たとえば、Code Composer Studio の「Build Options」ダイアログ・ボックスから最適化をオフにしたとする。コンパイル・プロセスを開始すると、TI コンパイラが正しいオプションを指定して起動されているように見えるが、最終コードを調べると、最適化がオフになっていないことがわかる。**

TI コード生成ツールは、Code Composer Studio の「Build Options」ダイアログ・ボックスでユーザが指定したオプションを使用して起動されます。しかし、これらのオプションは、「C_OPTIONS」などの環境変数を使用して設定されたオプションによって上書きされます。したがって、autoexec.bat ファイル内で (Windows 95) またはシステム環境変数の一部として (Windows NT) 環境変数を定義した場合、これらのオプションは、ユーザが Code Composer Studio の「Build Options」ダイアログ・ボックスで選択したすべての競合するオプションを上書きします。したがって、競合を避けるために、その環境変数を削除することを強くお勧めします。

8) **プロジェクトの一部として、パス名の異なる複数の「インクルード」ファイルを組み込むには、どのようにしたらよいか。**

プロジェクトのファイルの「インクルード」集合の一部として複数のファイルを組み込むには、次のようにします。

次の 4 つのステップを実行すると、ユーザのプロジェクトに関連したすべての「インクルード」ファイルが、プロジェクトの一部として組み込まれます。インクルード・ファイルのパス名は、必ずしも、ソース・ファイルのパス名と等しいとは限りません。

a) 「Project」→「Options」の順に選択します。

b) 「Compiler」タブを選択します。

c) 「Category」リスト・ボックスで Preprocessor を選択します。

d) 「Include Search Path」ダイアログ・ボックスで、複数の「インクルード」ファイルの完全なパス名を、セミコロンで区切って入力します。

9) **プロジェクトの構築プロセスを開始するときに、コンパイラが起動されると、「Build」ウィンドウの内部に次のエラー・メッセージが表示される。**

「Can't run cl6x – too many arguments」

TI コード生成ツールの起動に使用されるコマンド行に入力できる文字数には、80 文字の限界があります。したがって、Code Composer Studio の「Build Options」ダイアログ・ボックス内部の文字数を、80 文字に制限する必要があります。大部分の場合、コンパイラの段階で、コマンド行の文字数は、インクルード（ヘッダ）ファイルへのパスの指定に使用されてしまいます。この状態を改善し、コマンド行に渡すことができるオプション数を増やすには、次のように、環境変数を利用して、インクルード検索パスを指定します。

(Windows 95) 環境変数は、次の構文を使用して autoexec.bat ファイル内部で設定する必要があります。

SET TEMP=<pathname>

TEMP、=、および <pathname> の間にはスペースを入れないように注意してください。

(Windows NT) 環境変数は、システム環境変数を使用して設定する必要があります。スタート・ボタンから、「設定」→「コントロール・パネル」→「システム」の順に選択します。ダイアログ・ボックス内の「環境」タブの下に、次の情報を入力します。

❑ 「変数」フィールドに TEMP を追加します。

❑ 「値」フィールドに、インクルード・ファイルの完全なパス名を入力します。

TEMP は、この例に使用される環境変数の名前を意味します。変数名には、任意の名前を使用できます。Code Composer Studio の「Build Options (Compiler) 」ダイアログ・ウィンドウ内の 「Include Search Path」では、そのダイアログ・ウィンドウ内に %TEMP% と入力すると、環境変数を参照することができます。複数の環境変数を「Include Search Path」ウィンドウに入力するには、%TEMP%;%TEMP1%;...;%TEMPn% と入力します（ここで、TEMP..TEMPn は、すべて事前定義された環境変数とします）。

10) **プロジェクトの構築を開始すると、「Build」ウィンドウに、起動されたコンパイラ・コマンドが青色で表示されるが、応答が表示されず、リンカの段階で「obj」ファイルが検出できない。何が問題か。**

コンパイラ、アセンブラ、リンカのツールを起動するコマンド行の長さが 80 文字を超えると、この状態が発生します。この問題を改善するには、「Build Options (Compiler)」ダイアログ・ウィンドウ内の「Include Search Path」に指定された「インクルード」ディレクトリを削除し、環境変数（たとえば、「C_OPTIONS」）を使用して Compiler オプションを定義するようにしてください。詳細は、9) を参照してください。

A.3 一般的なデバッグ

1) **「Edit」ウィンドウで有効な C 行にブレークポイントを設定したときに、プログラムの実行を開始すると、次のエラー・メッセージが表示される。**

「Unable to move breakpoint to a valid line at source line: <filename> at line xxx. It has been disabled」

次の原因が考えられます。

a) プログラムをターゲットにロードしていない。Code Composer Studio デバッガでは、ユーザは、ターゲットにプログラムをロードして、すべてのシンボル情報を取得する必要があります。このシンボル情報は、C ソース行ごとに、正確なターゲット・アドレスを Code Composer Studio に指示します。

b) 指定されたソース行に対して有効なアセンブリ行が存在しない。

注 : ブレークポイントの設定に問題がある場合、この問題を解決する最善の方法は、「Mixed Source/ASM」オプション（「VIEW」メニューの下にある）をオンにすることです。この表示は、関連した命令だけでなく、すべての C 行を表示します。ファイル内のアセンブリ行が表示されない場合は、この C ファイルに対するシンボル情報が存在しません。

A.4 エディタ

1) **エディタ内で選択したエリア全体を Tab 位置 1 つ分シフトするには、どのようにしたらよいか。**

マウスを使って、必要なエリアを選択します。次に、Tab キー（または shift-Tab キー）を使って、そのエリア全体をタブ位置 1 つ分シフトしてください。

A.5 「Watch」ウィンドウ

1) **「Watch」ウィウドウ内で変数の表示オプションを選択するには、どのようにしたらよいか。**

式の後に「,D」のようにフォーマット記号を続けて指定すると、さまざまな表示オプションを選択することができます。ここで、D は任意の有効な表示オプションを示しています。詳細は、「Watch」ウィンドウにあるオンライン・ヘルプを参照してください。

A.6 General Extension Language – GEL

1) **前後に引用符が必要なシンボルと、必要でないシンボルがあるのはなぜか。たとえば、WatchAdd には引用符が必要であり、BreakPtAdd には必要でないのはなぜか。**

引用符は、実行前に構文解析も評価も行われていない式または文字列の前後に置かれます。式を入力すると必ず、その式は構文解析され、実行されます。GEL 関数（組み込み関数を含む）を呼び出す場合も、これと同じことが当てはまります。式を引数として GEL 関数に渡す場合、その式は評価され、最終結果がその GEL 関数に渡されます。たとえば、組み込み関数に対する呼び出し GEL_BreakPtAdd(StartAddress + 0x100) では、式「StartAddress + 0x100」が計算され、その後、その組み込み関数に結果が渡されます。

組み込み関数 GEL_WatchAdd() の場合、式を引数として渡したくありません。渡したいのは、「Watch」ウィンドウに追加される文字列です。その文字列に入っている式を評価するのは、「Watch」ウィンドウの役割です。

2) **Code Composer Studio を始動するたびに GEL 関数を実行できるか。**

できます。Code Composer Studio を始動するコマンド行の末尾に GEL ファイル名を指定すると、始動時に、自動的に GEL 関数をロードできます。このアクションは、GEL ファイル内の GEL 関数にアクセスできるように、その GEL ファイルを Code Composer Studio のメモリにロードするものであることを認識しておく必要があります。このファイルに、名前が「StartUp()」の GEL 関数が含まれている場合、Code Composer Studio は、この関数を実行します。したがって、この関数内に初期化タスクを入れることができます。

3) **コードがブレークポイントを検出するたびに GEL 関数を実行するには、どのようにしたらよいか。**

ブレークポイントで GEL 関数を実行するには、条件付きブレークポイントを設定することをお勧めします。呼び出す関数を、条件式として入力してください。その GEL 関数を呼び出すだけで、その後も実行を継続させる場合は、GEL 関数に FALSE（つまり 0）を戻させます。それ以外の場合は、TRUE（つまり 1）を戻させてください。

4) **Code Composer Studio を起動するたびに GEL ファイルとワークスペースを自動的にロードするには、どのようにしたらよいか。**

Code Composer Studio の環境内で、ご自分の GEL ファイルをロードし、デバッグ環境をセットアップします。次に、そのセットアップをワークスペースとしてセーブします。Code Composer Studio を始動するコマンド行の末尾に、このワークスペースの名前（.wks 拡張子付き）を指定して始動してください。(2.15 節 「ワークスペースのセーブとロード」を参照)。マルチプロセッサの場合にも、これと同じ手順が適用されます（3.6 節 「GEL 関数の自動実行」を参照)。

A.7 「Graph」ウィンドウ

1) **少なくとも、コードをステップ実行しているときに、グラフが更新されないようにすることができるか。**

できます。必要な作業は、そのグラフをプローブ・ポイントに接続することです。グラフがプローブ・ポイントに接続されていない場合（「Graph」ウィンドウを初めて開いたときが、これに当てはまります）は、ブレークポイントまでシングル・ステップ実行、または実行を行うたびに、そのグラフが更新されます。しかし、グラフをプローブ・ポイントに接続すると、コードの実行中にそのプローブ・ポイントに達したときだけ、このグラフは更新されます。

2) **「Graph」ウィンドウで変数の値をトレースしたい。この機能は、Code Composer Studio でサポートされているか。**

サポートされています。「Graph」ウィンドウは、変数の値をトレースできる柔軟性を備えています。実行するステップは、次のとおりです。

a) 次のパラメータを指定して、「Graph」ウィンドウを開きます。

Start Address – 対象の変数が「ErrorPower」である場合は、この位置に「&ErrorPower」と入力します。ここで最もよく起きる間違いは、ユーザが、変数の前に「&」を入力しないことです。このフィールドには任意の有効な 「C」式を入力することができ、その結果、ユーザが必要とする変数のアドレスが指定されることに注意してください。

Acquisition Buffer Size – 「1」と入力します。

Display Data Size – 履歴を表示したいサンプルの数を入力します。

Left-Shifted Data Display – このオプションを選択します。

b) 「Graph」 ウィンドウをプローブ・ポイントに接続します。 「Left shifted Data Display」オプションを使用しているときには、グラフをプローブ・ポイントに接続して、「Graph」ウィンドウを更新したくないときに、このウィンドウが更新されないようにすることをお勧めします。

これによって、このグラフ表示は、変数の値をグラフ表示するようになります。

付録 B

用語集

A

Animate： プログラムは、ブレークポイントが検出されるまで実行されます。ブレークポイントが検出されると、プログラムの実行が停止し、デバッガはプローブ・ポイントに接続されているウィンドウ以外のすべてのウィンドウを更新します。プログラムの実行が再開し、次のブレークポイントが検出されるまで続きます。

C

「Call Stack」ウィンドウ： デバッグ中のプログラム内の現在の位置に至るまでの、一連の関数呼び出しを表示します。

D

「Dis-Assembly」ウィンドウ： デバッグに必要な逆アセンブル命令とシンボル情報を表示します。

G

General Extension Language (GEL)： ユーザが、Code Composer Studio の環境を設定する関数を書き、ターゲット・プロセッサにアクセスできるようにするインタープリタ言語

「Graph」ウィンドウ： アプリケーション・プログラムによって作成されるデータの分析を可能にします。

M

「Memory」ウィンドウ： 指定されたアドレスから開始するターゲット・メモリの内容を表示します。

R

「Register」ウィンドウ： CPU またはペリフェラル・レジスタの内容の表示、および編集を可能にします。

W

「Watch」ウィンドウ： 変数と式の表示、および編集を可能にします。

あ

アクティブ・ウィンドウ： 編集、移動、サイズ変更、クローズなどのために、現在選択されているウィンドウ

アニメーション速度： ブレークポイント相互間の最小時間を定義します。animate コマンドを使用してプログラムを実行すると、前回のブレークポイントからこの最小時間が経過するまで、プログラムの実行は再開されません。

え

エミュレータ： ターゲット・プロセッサの動作をエミュレートするハードウェア開発システム

エントリ・ポイント： ターゲット・メモリでの実行開始位置

お

オブジェクト・ファイル： 機械語オブジェクト・コードを含む、アセンブルまたはリンクされたファイル

オブジェクト・ライブラリ： 複数のオブジェクト・ファイルで構成されたアーカイブ・ライブラリ

オプション： ソフトウェア・ツールの起動時に、追加の機能や特定の機能を実行させるために使用するコマンド・パラメータ

き

逆アセンブリ： ターゲット・メモリの内容を逆方向にアセンブリすることによって形成されるアセンブリ言語コード

共通オブジェクト・ファイル・フォーマット (COFF)： バイナリ・オブジェクト・フォーマットの一種。セクションの概念を採用することによりプログラミングを促進します。すべての COFF セクションは、メモリ空間の中で別々に再配置することができます。ユーザは、ターゲット・メモリのどの割り当て済みブロックに、どのセクションを入れても構いません。

し

自動ロード： 始動時に、ファイルを自動的にロードします。

シミュレータ： ターゲット・プロセッサの動作をシミュレートするソフトウェア開発システム

シングル・ステップ： プログラムは文ごとに実行されます。これにより、ユーザは各文の効果を確認することができます。

シンボル： アドレス、または値を表す英数字からなる文字列

シンボル情報： ターゲット上のアドレスまたは値を表す、シンボルおよび英数字からなる文字列

シンボル・テーブル： COFF オブジェクト・ファイルの一部分。ファイルで定義されて使用されるシンボルについての情報を含みます。

せ

セクション： コード、またはデータの再配置可能なブロック。最終的には、ターゲット・メモリ・マップ内の連続した空間を占めます。

そ

ソース・ファイル： C コード、またはアセンブリ言語コードを含むファイル。コードをコンパイルやアセンブルすることによって、オブジェクト・ファイルを作成します。

た

ターゲット・システム： 開発したオブジェクト・コードを実行するシステム

と

ドッキング可能なウィンドウ： Code Composer Studio 内にある多くのウィンドウは、ドッキング可能です。Code Composer Studio のメイン・ウィンドウのどの部分にでも、ドッキング可能なウィンドウを移動し、位置を調整することができます。「Allow Docking」プロパティをディセーブルにすると、Code Composer Studio の親ウィンドウからそのウィンドウを削除し、デスクトップ上の任意の場所に配置することができます。

は

バイト： 1 つの単位として処理される、隣接する 8 ビットのシーケンス

ふ

ブックマーク： ソース・ファイル内の重要な位置にマークを付けます。

ブレークポイント： プログラムの実行が停止する位置を定義します。プログラムの実行が停止している間に、ユーザはプログラムの状態を分析することができます。

プロジェクト： アプリケーション・プログラムの開発を管理するためのフレームワーク

プロジェクト環境： アプリケーション・プログラムの開発速度を上げるツールの集合。アプリケーション・プログラムは、Code Composer Studio 環境内のプロジェクトとして開発されます。プロジェクトに関するすべての情報は、プロジェクト・ファイルに保存されます。

プロジェクト・ファイル： 特定のプロジェクト用の情報を保存する単一のファイル。この情報とは、アプリケーション・プログラムまたはライブラリの作成に必要なソース・ファイル、オブジェクト・ファイル、オブジェクト・ライブラリ、ソフトウェア・ツール・オプション、依存関係などです。

プロファイル・ポイント： ブレークポイントとほぼ同じですが、プログラム実行を停止するのではなく、直前のプロファイル・ポイントが検出されて以降に発生したイベントの統計情報を収集します。

プローブ・ポイント： プログラム実行中に、いつウィンドウが更新されるかを定義します。プローブ・ポイントがウィンドウに接続されている場合、このウィンドウが更新されるのは、プローブ・ポイントに達したときだけです（ブレークポイントが検出されても、このウィンドウは更新されません）。このウィンドウが更新された後、プログラムの実行は再開されます。

へ

変数： 一連の値のうちのいずれかを想定する、ある量を表すシンボル

ま

マルチプロセッシング： Code Composer Studio は、複数のプロセッサを同時にデバッグする機能をもっています（エミュレータのみ）。

め

メモリ・マップ： ターゲット・システムのメモリ空間のマップ。複数の機能ブロックに区画分けされています。メモリ・マップを使って、メモリ内のアクセス可能なエリアとアクセス不可のエリアをデバッガに指示します。

わ

ワークスペース： Code Composer Studio の稼働環境。ワークスペースはセーブすることができます。以前にセーブされたワークスペースを再ロードすることもできます。

索引

A

B

C

D

E

F

G

H

I

L

M

O

P

Q

R

S

T

U

W

あ

い

う

え

お

か

き

く

こ

さ

し

せ

そ

た

ち

つ

て

と

に

は

ひ

ふ

へ

ほ

ま

め

よ

り

れ

ろ

わ

日本テキサス・インスツルメンツ株式会社

本　　社　〒160-8366 東京都新宿区西新宿6丁目24番1号　西新宿三井ビルディング3階

大阪営業所　〒530-6026 大阪市北区天満橋1丁目8番30号　ＯＡＰオフィスタワー26階　06(6356)4500(代表)
名古屋営業所　〒460-0003 名古屋市中区錦2丁目4番3号　錦パークビル10階　052(232)5601(代表)
松本営業所　〒390-0815 長野県松本市深志1丁目2番11号　松本昭和ビル6階　0263(33)1060(代表)
立川営業所　〒190-0012 東京都立川市曙町2丁目36番2号　ファーレ立川センタースクエア10階　042(527)6760(代表)
京都営業所　〒600-8216 京都市下京区塩小路通西洞院東入東塩小路町843-2　日本生命京都ヤサカビル5階　075(341)7713(代表)
大宮営業所　〒330-0802 埼玉県大宮市宮町1丁目38番1号　野村不動産大宮共同ビル4階　048(647)2622(代表)
米国ダラス営業所　TEXAS INSTRUMENTS Inc., 12500 TI Boulevard, M/S 8673 Dallas, Texas, 75243, U.S.A
1-214-480-7464

工　　場　大分県・日出町　茨城県・美浦村　静岡県・小山町（制御機器事業部）
筑波テクノロジーセンター　茨城県・つくば市

■お問い合わせ先
プロダクト・インフォメーション・センター（PIC）　0120-81-0026　FAX 0120-81-0036
E-mail: pic-japan@ti.com

Code Composer Studio
ユーザーズ・マニュアル

第1版 2000**年**11**月**

発行所 **日本テキサス・インスツルメンツ株式会社**
〒160-8366
東京都新宿区西新宿6-24-1**（西新宿三井ビルディング）**

[redacted]

Printed in Japan
'00.11

日本テキサス・インスツルメンツ株式会社

SPRU486

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