2008 年 10 月 14 日
株式会社インテリジェントシステムズ

# IS-TWL-DEBUGGER
# チュートリアル

INTELLIGENT SYSTEMS CO.,LTD.

# 目次

1. はじめに ..... 4
1.1 IPL のアップデート(重要) ..... 4
1.2 はじめに ..... 4
1.3 文章中のアイコンについて ..... 4
2. チュートリアルの環境の準備 ..... 5
2.1 ハードウェアのセットアップ ..... 5
2.2 ソフトウェアのインストール ..... 5
2.3 環境変数の設定 ..... 6
3. プログラム開発の流れ ..... 7
4. IS-TWL-DEBUGGER の起動と終了 ..... 8
4.1 IS-TWL-DEBUGGER の起動 ..... 8
4.2 IS-TWL-DEBUGGER の終了 ..... 8
5. IS-TWL-DEBUGGER のウィンドウ ..... 9
5.1 ウィンドウの操作 ..... 9
5.2 ウィンドウの種類 ..... 10
6. デバッグの準備 ..... 12
6.1 サンプル プログラムの格納場所 ..... 12
6.2 サンプル プログラムのコンパイル ..... 12
7. プログラムを読み込む ..... 14
7.1 プログラムを読み込む ..... 14
7.2 プログラム読み込み後の[プロジェクト]ウィンドウ ..... 15
8. プログラムの実行と停止 ..... 16
8.1 プログラムの実行 ..... 16
8.2 プログラムの停止 ..... 16
8.3 プログラムを最初から実行する ..... 17
9. プリントデバッグ ..... 18
10. [ソース]ウィンドウの表示 ..... 18
11. ソース指定行での停止 ..... 19
12. 意図しないメモリ書き換えの検出 ..... 20
13. 行単位、関数単位の実行 ..... 21
13.1 行単位、関数単位での実行の種類 ..... 21
13.2 1行分実行(トレース実行) ..... 21
13.3 1行分実行(ステップ実行) ..... 22
13.4 関数を抜けるまで実行(ステップ アウト実行) ..... 22
14. 変数値の確認 ..... 23
14.1 変数値の確認方法一覧 ..... 23
15. [ソース]ウィンドウ上で変数値を確認(データチップ) ..... 24
16. 変数値の参照と変更(クイックウォッチ) ..... 25
17. 変数値の参照と変更(ウォッチウィンドウ) ..... 26
18. 関数呼び出し履歴の確認 ..... 27
19. ローカル変数の表示と変更(ローカルウィンドウ) ..... 27
20. スレッドのデバッグ ..... 28
20.1 カレントスレッド ..... 28
20.2 スレッドを切り替える ..... 28
21. メモリ内容の表示 ..... 29
21.1 IS-TWL-DEBUGGER のメモリ空間 ..... 29
21.2 TWL 本体内メモリ内容の表示と変更 ..... 29
22. [ソース]ウィンドウでのアセンブラ混合表示 ..... 30
23. 逆アセンブル表示 ..... 31
23.1 ソースウィンドウに対応する逆アセンブルの表示 ..... 31
24. ソースの任意行にプログラムカウンタを移動する ..... 32
25. I/O レジスタの表示と変更 ..... 33
26. ソース ファイルの編集 ..... 34
26.1 テキスト エディタの起動設定 ..... 34
27. プログラムのビルド設定 ..... 35
28. プログラムのビルド ..... 36
29. エラータグジャンプ ..... 36
30. プログラムを自動的に読み込む ..... 37
31. プロジェクト設定の保存 ..... 37

32. TWL 本体内メモリ内容の保存 ..... 38
33. コマンドによる制御 ..... 39
33.1 コマンドの実行 ..... 39
33.2 繰り返し操作の自動化 ..... 40
34. EL ライブラリを使用したデバッグ ..... 41
34.1 EL ライブラリの概要 ..... 41
34.2 EL ライブラリの制限 ..... 41
34.3 EL ライブラリのロード状態を確認する ..... 41
35. ROM 内登録データの確認 ..... 42
36. サポート ..... 43
36.1 サポート窓口 ..... 43
36.2 著作権と商標について ..... 43

# 1. はじめに

この章では、IS-TWL-DEBUGGER を使用するにあたって重要な内容、IS-TWL-DEBUGGER チュートリアルの使い方や文章中に使用するアイコンなどについて説明します。

## 1.1 IPL のアップデート（重要）

TWL 用開発ツール導入の際は、かならず IPL のアップデートを行なってください。
IPL の更新に関する手順や詳細は任天堂株式会社提供の TWL SystemUpdater をご参照ください。

## 1.2 はじめに

『IS-TWL-DEBUGGER チュートリアル』(以降チュートリアル)は、IS-TWL-DEBUGGER の使い方を説明します。チュートリアルは、サンプル プログラムを使用し、操作を確認しながら読み進めることができるように構成されています。

画面や各機能の詳細については、『IS-TWL-DEBUGGER のヘルプ』を別途参照してください。『IS-TWL-DEBUGGER のヘルプ』は、Windows の[スタートメニュー]-[すべてのプログラム]-[IS-TWL-DEBUGGER]-[IS-TWL-DEBUGGER のヘルプ]から起動できます。

なお、本チュートリアルに記載されている内容は Windows XP 上で実行した方法となり、他の OS では若干方法が異なる可能性があります。

また、IS-TWL-DEBUGGER は開発中のものです。 画面の様子および製品の仕様は、断りなく変更になることもございます。

## 1.3 文章中のアイコンについて

チュートリアルには、次のアイコンが登場します。


チュートリアルの確認事項を示します。


使用上の注意、警告を示します。


IS-TWL-DEBUGGER 活用のためのヒントを示します。

# 2. チュートリアルの環境の準備

この章では、チュートリアルの実行に必要なハードウェア、およびソフトウェアについて説明します。以下の環境をセットアップおよびインストールしてください。

## 2.1 ハードウェアのセットアップ

IS-TWL-DEBUGGER に同梱されている『IS-TWL-DEBUGGER セットアップマニュアル』を参考にして、IS-TWL-DEBUGGER ハードウェアをセットアップしてください。

なお、使用する前に IS-TWL-DEBUGGER コントローラにて、DSi メニューの本体設定を行ってください。設定を行っていない場合、タッチパネル等正しく動作しない可能性があります。

## 2.2 ソフトウェアのインストール

以下のソフトウェアをインストールし、チュートリアルを確認できる環境をご用意ください。
チュートリアルは、各ソフトウェアを以下のフォルダにインストールすることを想定しています。チュートリアルで登場するフォルダは、お客様の環境に合わせて読み替えてご使用ください。

| ソフトウェア | インストールするフォルダ |
|---|---|
| IS-TWL-DEBUGGER | C:¥Program Files¥INTELLIGENT SYSTEMS¥IS-TWL-DEBUGGER |
| Code Warrior for NINTENDO TWL V1.0 | C:¥Program Files¥Freescale¥CW for NINTENDO TWL V1.0 |
| TWL-SDK | C:¥TWLSDK |
| Cygwin | C:¥Cygwin |

ソフトウェアインストール後に環境変数を反映させるため、一旦ログオフしてください。

## 2.3 環境変数の設定

TWL-SDK を使用したビルドはいくつかの環境変数が必要です。TWL-SDK が使用する環境変数の詳しい説明は任天堂株式会社作成の『Quick Start Guide』の 2.環境変数、および、ビルドスイッチ使用表を参照してください。それぞれ以下のフォルダに格納されています。

C:¥TWLSDK¥docs¥README¥QuickStartForSDK.pdf
C:¥TWLSDK¥docs¥SDKRules¥Rule-Defines.html

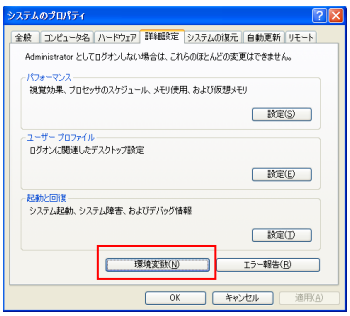
チュートリアルでは、環境変数を次のように設定しています。

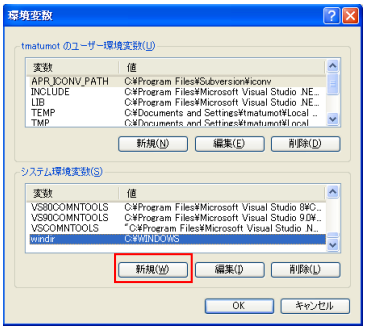
| 環境変数名 | 値 |
|---|---|
| TWLSDK_ROOT | C:¥TWLSDK |
| TWLSDK_PLATFORM | TWL |
| CWFOLDER_TWL | C:¥Program Files¥Freescale¥CW for NINTENDO TWL V1.0 |

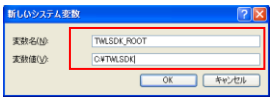
環境変数の設定は、Windows の[システムのプロパティ]ダイアログから永続的に設定することや Cygwin の bash から一時的に設定できます。ご使用の状況に合わせて環境変数を設定してください。ここでは、Windows の[システムのプロパティ]から環境変数を永続的に設定する方法を紹介します。

① Windows の[スタート]メニュー-[マイコンピュータ]を右クリックし、メニュー[プロパティ]をクリック、[システムのプロパティ]ダイアログを表示します。
② [詳細設定ページ]の[環境変数]ボタンをクリックし、[環境変数]ダイアログを表示します。
③ [システム環境変数]の[新規]をクリックし、[新しい変数]ダイアログを表示します。
④ [変数名]には TWLSDK_ROOT を、[値]には C:¥TWLSDK を設定し、[OK]ボタンをクリックして環境変数を登録します。

つづけて TWLSDK_PLATFORM など必要な環境変数を設定します。



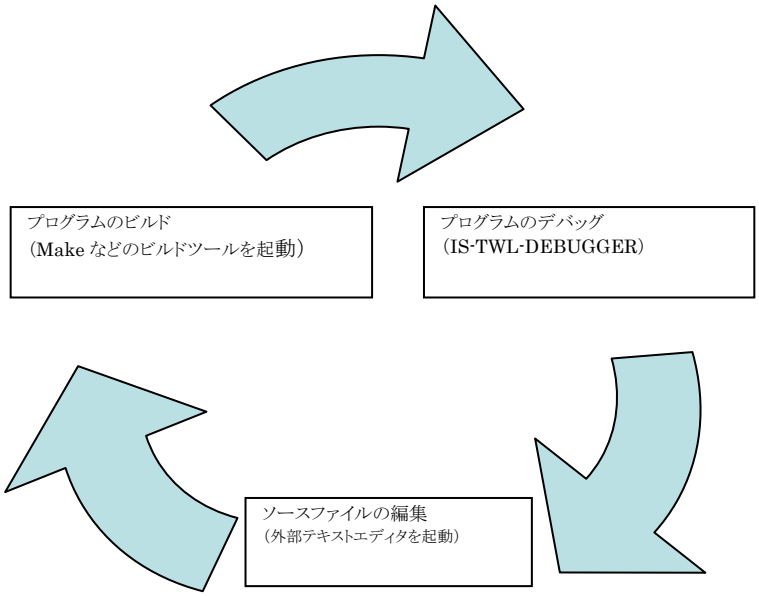




⚠ IS-TWL-DEBUGGER の画面レイアウトやショートカットキーをはじめ、各ソフトウェアの設定は出荷時の状態を想定しています。

# 3. プログラム開発の流れ

この章では IS-TWL-DEBUGGER を使用したプログラム開発の流れを説明します。

IS-TWL-DEBUGGER は、プログラムのデバッグ、ソース ファイルの編集、プログラムのビルド、再びプログラムのデバッグという、一連のプログラム開発の流れを IS-TWL-DEBUGGER 上から行うことができます。チュートリアルでは、プログラムのデバッグ方法、IS-TWL-DEBUGGER 上からのソース ファイルの編集方法、IS-TWL-DEBUGGER 上からのプログラムのビルド方法について説明します。

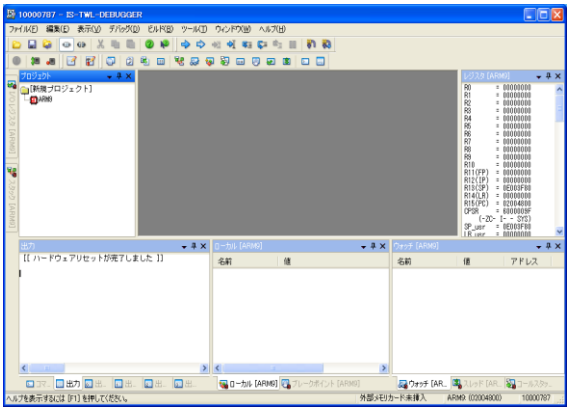


# 4. IS-TWL-DEBUGGER の起動と終了

この章では、IS-TWL-DEBUGGER の起動、および終了方法を確認します。

## 4.1 IS-TWL-DEBUGGER の起動

IS-TWL-DEBUGGER を起動するには、Windows の[スタート]ボタンから[すべてのプログラム]-[IS-TWL-DEBUGGER]-[IS-TWL-DEBUGGER]の順にクリックします。

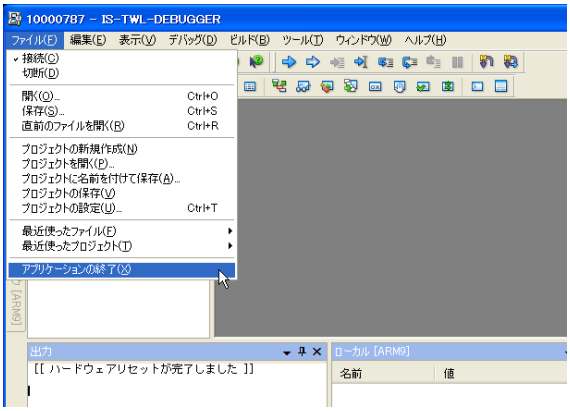


## 4.2 IS-TWL-DEBUGGER の終了

IS-TWL-DEBUGGER を終了するには、次のいずれかの操作を行います。

- [ファイル] メニューの[アプリケーションの終了]をクリックする。
- タイトル バーアイコンをクリックして表示されるメニューから[閉じる]をクリックする。
- タイトル バーアイコンをダブルクリックする。
- タイトル バーの[閉じる]ボタンをクリックする。
- [Alt]キーを押しながら[F4]キーを押す。

チュートリアルを途中で終了したい場合、この操作で IS-TWL-DEBUGGER を終了してください。



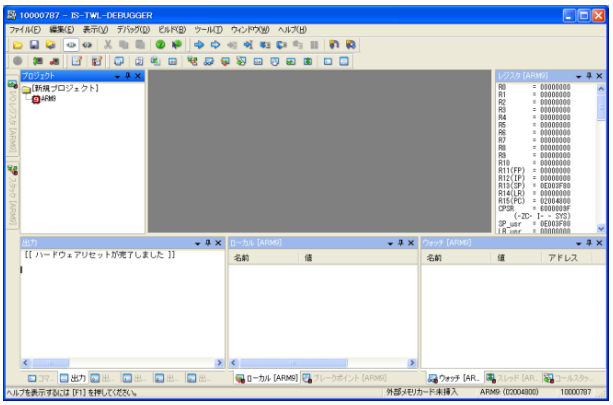

この章では IS-TWL-DEBUGGER の起動と終了方法について確認しました。

# 5. IS-TWL-DEBUGGER のウィンドウ

この章では、IS-TWL-DEBUGGERの起動直後のウィンドウレイアウト、およびIS-TWL-DEBUGGERを構成するウィンドウについて確認します。

## 5.1 ウィンドウの操作

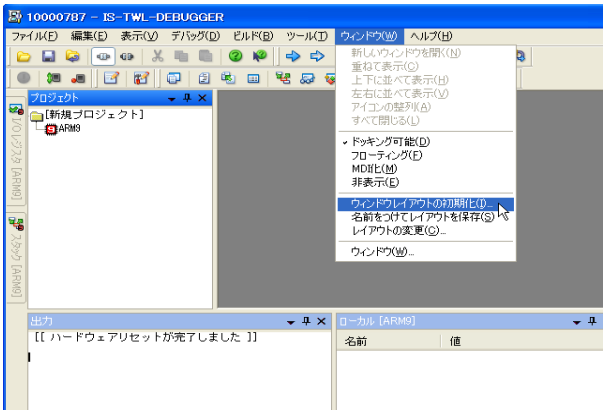
IS-TWL-DEBUGGER を初めて起動すると、IS-TWL-DEBUGGER のウィンドウは右図のようにレイアウトされます。ウィンドウレイアウトは、自由に変更できます。



チュートリアルではウィンドウレイアウトの操作説明は行いません。ウィンドウ操作に関する詳しい機能や操作方法については、『IS-TWL-DEBUGGER のヘルプ』を参照してください。

ウィンドウレイアウトは、いつでも IS-TWL-DEBUGGER インストール直後の状態に初期化できます。
ウィンドウレイアウトを初期化するには、次の操作を行います。

① [ウィンドウ]メニューの[ウィンドウレイアウトの初期化]をクリックします。
② [ウィンドウレイアウトを初期化しますか？]ダイアログ ボックスの[OK]ボタンをクリックします。

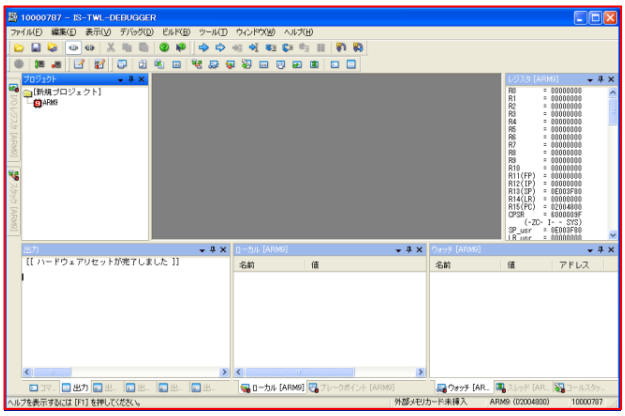


## 5.2 ウィンドウの種類

IS-TWL-DEBUGGER は、次のウィンドウにより構成されています。

**メイン ウィンドウ**
メイン ウィンドウは、IS-TWL-DEBUGGER の最も外側のウィンドウです。

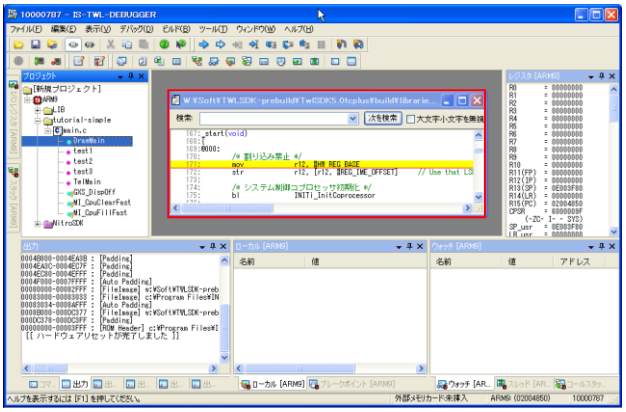
メイン ウィンドウの操作は Windows に準拠しています。詳しくは Windows のヘルプをご覧ください。



**ドキュメント ウィンドウ**
ドキュメント ウィンドウは、MDI 子ウィンドウ形式のウィンドウです。

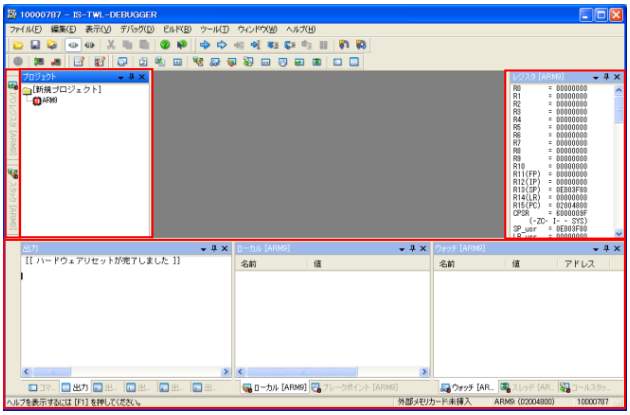
ドキュメント ウィンドウは、同じ種類のウィンドウを複数開くことができます。

- [ソース]ウィンドウ
- [逆アセンブル]ウィンドウ
- [ダンプ]ウィンドウ



**ドッキング ウィンドウ**
ドッキング ウィンドウは、ウィンドウ モードを切り替えることができるウィンドウです。

ウィンドウ モードは、メイン ウィンドウや他のドッキング ウィンドウにドッキングできる[ドッキング モード]、メイン ウィンドウから独立するウィンドウになる[フローティング モード]、必要な時だけ画面に表示する[自動的に隠すモード]など、好みに合わせてウィンドウごとにウィンドウ モードを設定できます。



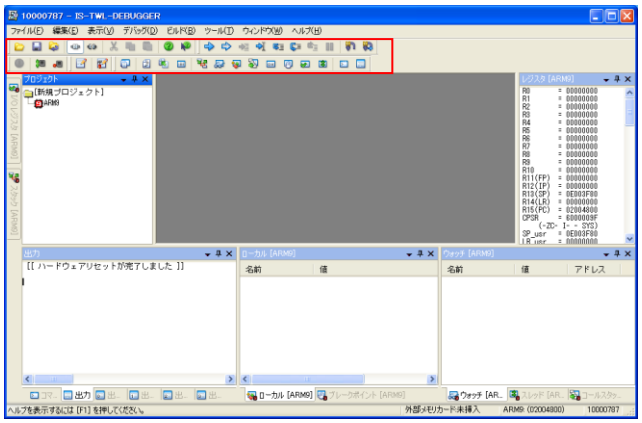
- [プロジェクト]ウィンドウ
- [コールスタック]ウィンドウ
- [ローカル]ウィンドウ
- [ウォッチ]ウィンドウ
- [レジスタ]ウィンドウ
- [スタック]ウィンドウ
- [I/O レジスタ]ウィンドウ
- [ブレークポイント]ウィンドウ
- [出力]ウィンドウ
- [スレッド]ウィンドウ

**ツール バー**

ツール バーは、メイン ウィンドウの上部にドッキングしているコマンド ボタンを備えたウィンドウです。

メイン ウィンドウの上下左右にドッキング、またメイン ウィンドウから独立してフローティングできます。

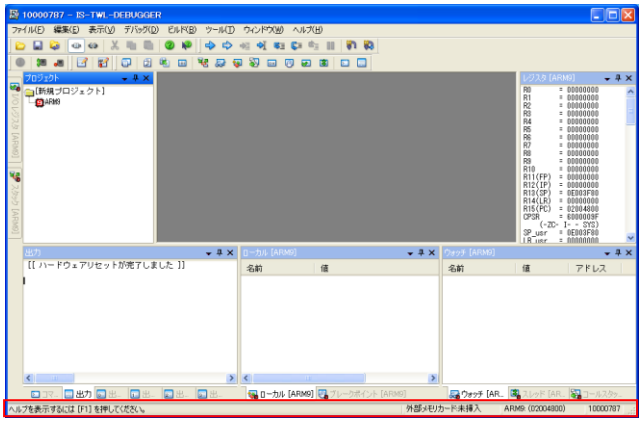
主なメインメニューのコマンドをボタンで用意していて素早くアクセスできます。



**ステータス バー**

ステータス バーは、メイン ウィンドウの下部にドッキングし、アプリケーションの情報を表示するウィンドウです。

プログラムの実行・停止状態、DS カードスロットの電源状態などを表示します。



# 6. デバッグの準備

この章では、デバッグに必要なビルドツールの設定、デバッグに使用するサンプルについて確認します。

## 6.1 サンプル プログラムの格納場所

チュートリアルは、IS-TWL-DEBUGGER の付属サンプル プログラム[tutorial-simple]を使用してデバッグ方法を確認します。なお、サンプル プログラムはマンデルブロー図を表示するもので、IS-TWL-DEBUGGER のインストール フォルダ以下に用意しています。

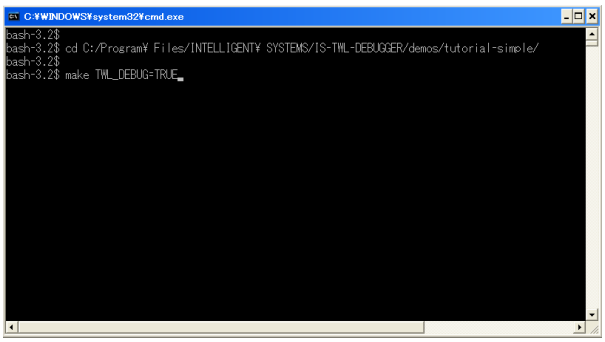
**C:¥Program Files¥INTELLIGENT SYSTEMS¥IS-TWL-DEBUGGER¥demos¥Tutorial-simple**

## 6.2 サンプル プログラムのコンパイル

IS-TWL-DEBUGGER でソースレベルデバッグを行うには、プログラムをコンパイルする際にデバッグ情報を生成するように指定する必要があります。TWL-SDK を使用した開発では、以下の手順でデバッグ情報を生成することができます。

以下 TWL-SDK を使用してサンプルプログラムをビルドする例です。

サンプル プログラムをコンパイルする手順は次の通りです。




① エクスプローラから、**C:¥Cygwin¥Cygwin.bat** をダブルクリックして、コマンドプロンプトを起動します。

② コマンドプロンプトからサンプル プログラムのフォルダに移動します。

**cd**
**C:/Program¥ Files/INTELLIGENT¥ SYSTEMS/IS-TWL-DEBUGGER/demos/Tutorial-simple**

③ 続いてコマンドプロンプトからプログラムをビルドします

**make TWL_DEBUG=TRUE**

ビルドが完了すると、
**C:¥Program Files¥INTELLIGENT SYSTEMS¥IS-TWL-DEBUGGER¥demos¥Tutorial-simple¥bin¥ARM9-TS.HYB¥Debug¥main.tlf**
にプログラムとデバッグ情報の指定ファイルが作成されます。

ビルドに失敗する場合には、TWL-SDK に必要な環境変数が設定されていない可能性があります。「環境変数の設定」を参照し、環境変数を設定してください。

最適化したプログラムや、出荷用プログラムはコンパイラの最適化により正しくデバッグすることができません。make の引数 TWL_DEBUG=TRUE は最適化をオフし、デバッグ情報を出力します。make.exe に付加するコマンドオプションについては、TWL-SDK に含まれる C:¥TWLSDK¥docs¥SDKRules¥Rule-Defines.html を参照してください。

IS-TWL-DEBUGGER は、IS-TWL-DEBUGGER のメニューやツール バーからプログラムをビルドできます。この方法については、後の章「プログラムをビルドする」で説明します。

この章では IS-TWL-DEBUGGER で、デバッグするために必要なビルド方法について確認しました。

# 7. プログラムを読み込む

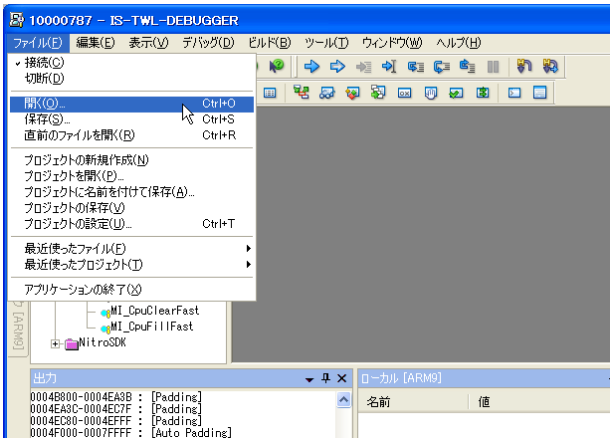
この章では、IS-TWL－DEBUGGER にデバッグ対象のプログラムを読み込む方法を確認します。

## 7.1 プログラムを読み込む

TLF ファイルは、ROM を構成するファイル一覧と、そのレイアウト方法を記載したファイルです。IS-TWL-DEBUGGER は TLF ファイルを読み込み、ROM を構成するファイル一覧をエミュレーションメモリに読み込みます。エミュレーションメモリとは、DS カードの ROM に相当するメモリです。

ここでは、前章[デバッグの準備]で作成したデバッグ対象のサンプル プログラム tutorial-simple を読み込みます。

IS-TWL-DEBUGGER にプログラムを読み込むには、次のいずれかの操作を行います。

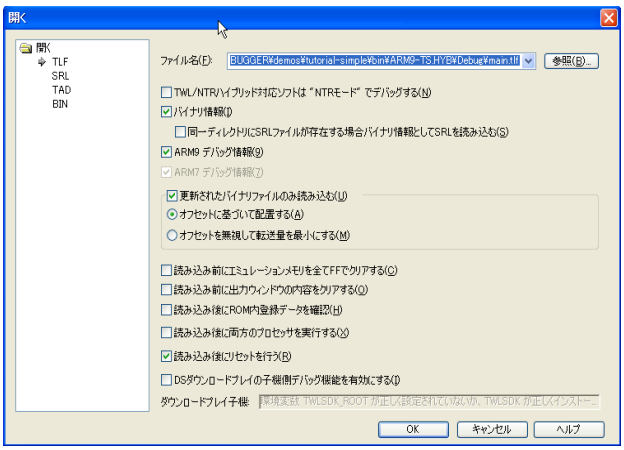



① [開く]ダイアログ ボックスを開きます。
- [ファイル]メニューの[開く]をクリックする。
- ツール バーの[開く]ボタンをクリックする。
- Ctrl キーを押しながら O を押す。

② [TLF]ページをクリックし、TLF ページを表示します。

③ [TLF]ページの[ファイル名]入力ボックスにサンプル プログラムを指定します。

**C:¥Program Files¥INTELLIGENT SYSTEMS¥IS-TWL-DEBUGGER¥demos¥Tutorial-simple¥bin¥ARM9-TS.HYB¥Debug¥main.tlf**



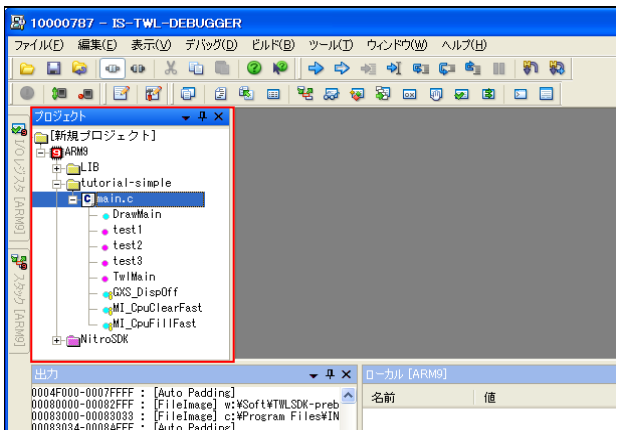

④ 続いて[OK]ボタンをクリックすると、読み込みを開始します。

## 7.2 プログラム読み込み後の[プロジェクト]ウィンドウ

IS-TWL-DEBUGGER にプログラムを読み込むと、[プロジェクト]ウィンドウにプログラム構造を表示します。プログラム構造は［ディレクトリ構造]-[ファイル]-[関数］のツリーで構成されています。

サンプル プログラムの場合、プログラム読み込み後の[プロジェクト]ウィンドウは右図のように表示します。

[プロジェクト]ウィンドウから[ソース]ウィンドウを開くなどの具体的な操作方法については、後の章で説明します。




この章では IS-TWL-DEBUGGER のプログラムの読み込み方法、およびプログラム読み込み後にプロジェクトウィンドウが更新されることを確認しました。

# 8. プログラムの実行と停止

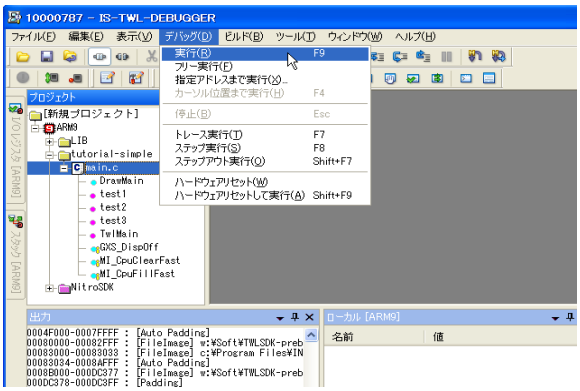
この章ではプログラムの実行方法、およびプログラムの停止方法について確認します。

## 8.1 プログラムの実行

ここでは、前章で読み込んだサンプル プログラムの実行方法を確認します。

プログラムを実行するには、次のいずれかの操作を行います。

- ツール バーの[実行]ボタンをクリックする。
- メニュー[デバッグ]-[実行]をクリックする。
- F9 キーを押す。




サンプル プログラムを実行すると、IS-TWL-DEBUGGER ハードウェア コントローラ部(以降コントローラとします)の液晶には、緑色のマンデルブロー図が表示されます。

プログラム実行中は、[デバッグ]-[停止]以外のメニューやツール バーのほとんどは灰色になり、操作することはできません。

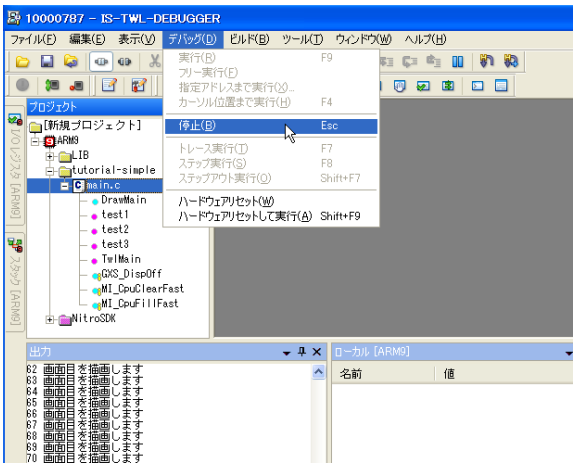
## 8.2 プログラムの停止

続いて、実行中のサンプル プログラムの停止方法を確認します。

プログラムを停止すると、ツール バーのボタンや、メニュー項目のほとんどが有効になり、クリックできる状態になります。プログラム停止中は、TWL 内のメモリやプログラム変数の値、I/O レジスタ、CPU レジスタの値をはじめ、TWL の任意の情報を表示、変更できます。

実行中のプログラムを停止するには、次のいずれかの操作を行います。

- ツール バーの[停止]ボタンをクリックする。
- メニュー[デバッグ]-[停止]をクリックする。
- ESC キーを押す。



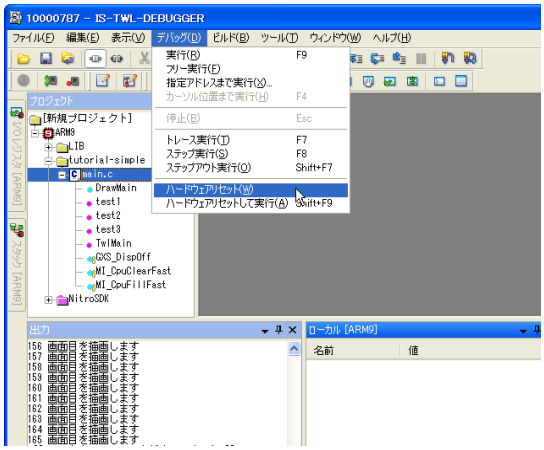
## 8.3 プログラムを最初から実行する

続いて、プログラムを最初から実行する手順について説明します。

プログラムのデバッグ中、プログラムを最初から実行しなおしたいことがあります。IS-TWL-DEBUGGERのハードウェア リセット機能を使用することで、TWL を電源投入直後の状態に戻し 再びプログラムを実行することができます。

ハードウェア リセットするには、次のいずれかの操作を行います。

- ツール バーの[両プロセッサをハードウェア リセット]ボタンをクリックする。
- メニュー[デバッグ]-[ハードウェア リセット]をクリックする。




ハードウェア リセットを完了すると、プログラム カウンタをはじめ、CPU レジスタ、I/O レジスタなどを TWL の電源投入直後の状態にエミュレーションします。

ハードウェア リセット後、プログラムを最初から実行できます。

IS-TWL-DEBUGGER は、ハードウェア リセット後に続けて実行することもできます。メニュー[デバッグ]-[ハードウェアリセットして実行]から行うことができます。

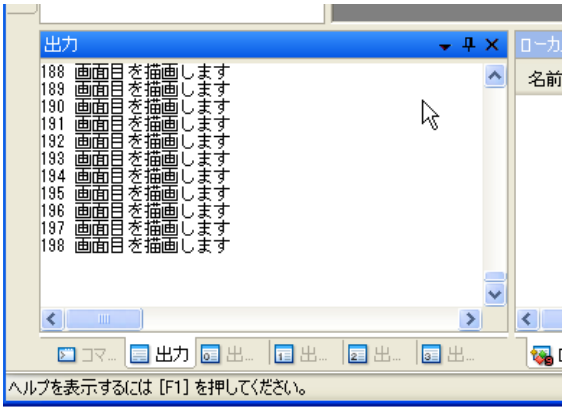
以上でプログラムの実行方法、プログラムの停止方法について確認しました。

# 9. プリントデバッグ

この章では、プリントデバッグを行う方法について確認します。
プリントデバッグとは、プログラム中にコンソール出力関数を埋め込むことで、通過順序、そのときの変数の状態などを[出力]ウィンドウに表示するデバッグ手法のことです。

プリントデバッグを行うには、TWL-SDK のコンソール出力関数 OS_Printf()を使用します。

サンプル プログラムの main.c 80 行には、既に OS_Printf()を埋め込んでいます。このサンプルでは、[出力]ウィンドウに右図の出力を行います。




IS-TWL-DEBUGGER は[出力]ウィンドウを複数持ち、それぞれ[出力 0]ウィンドウ、[出力 1]ウィンドウのように番号づけしています。ISTDPrintfEx()関数を使用して任意の[出力]ウィンドウに出力結果を振り分けることもできます。プリントデバッグの詳しい機能は『IS-TWL-DEBUGGER のヘルプ』を参照してください。

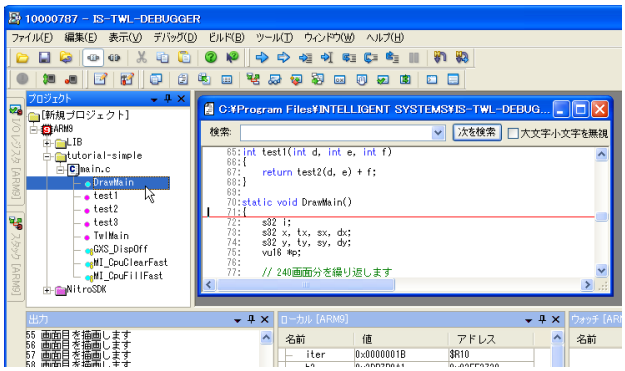
# 10. [ソース]ウィンドウの表示

この章では、[ソース]ウィンドウにソース ファイルを表示する方法を確認します。
[ソース]ウィンドウは、プログラムのソース ファイルを表示するウィンドウです。[ソース]ウィンドウから、ステップ実行、ブレークポイントの設置、変数の値を表示や変更など、ソースレベル デバッグの多くの機能を利用できます。

ここでは、サンプル プログラムの main.c の DrawMain 関数を[ソース]ウィンドウに表示する手順を説明します。

[ソース]ウィンドウに DrawMain 関数を表示するには、次の操作を行います。

① [プロジェクト]ウィンドウのツリーから **[ARM9]-[TutorialSimple]-[main.c]** の順にツリーをクリックします。
② さらに、ツリーの **DrawMain** をダブルクリックします。
以上で[ソース]ウィンドウに、main.c の DrawMain 関数を表示します。



# 11. ソース指定行での停止

この章では、前章で表示した[ソース]ウィンドウにブレークポイントを設置し、プログラムを任意のソース行でプログラムを停止する方法を確認します。

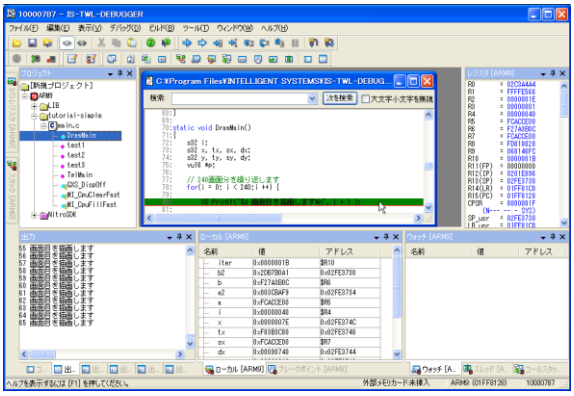
ソフトウェア ブレークポイントを使用すると、ソースの指定行を通過する時にプログラムを停止できます。
ここでは、サンプル プログラム main.c の 80 行でプログラムを停止する手順について説明します。

ソフトウェア ブレークポイントを設置するには、次のいずれかの方法を行います。

① プログラムを停止後、ブレークポイントを次のいずれかの方法で設置します。
- [ソース]ウィンドウの 80 行目をダブルクリックする。
- [ソース]ウィンドウの 80 行目にカーソルを位置づけ F2 キーを押す。
- [ソース]ウィンドウの 80 行目を右クリックしてメニューの[ブレークポイントの追加]をクリックする。

② ソフトウェア ブレークポイント設置後は、ブレークポイント設置行が緑色反転します。

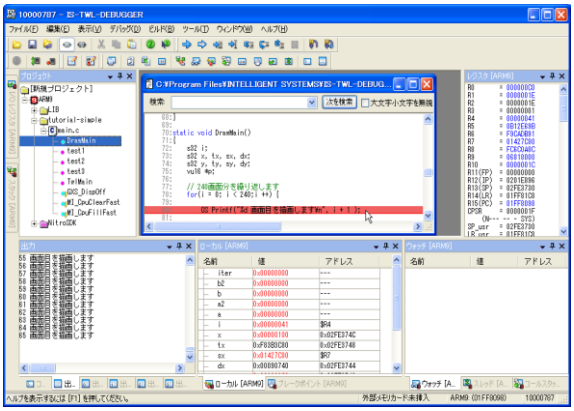
以上でブレークポイントの設置は完了です。



続いてプログラムを実行し、main.c の 80 行のブレークポイントでプログラムが停止することを確認します。

① サンプル プログラムを実行します。
② [ソース]ウィンドウの 80 行が赤色反転し、プログラムが 80 行のブレークポイントで停止します。

[ソース]ウィンドウの赤色反転は、プログラムがソフトウェアブレークポイントで停止していることを示しています。



再びブレークポイント設置行をダブルクリックするか、F2 キーを押すとソフトウェア ブレークポイントを解除できます。

[ブレークポイント]ウィンドウを併用すると、指定回数ブレークポイントを通過すると停止するなど、詳細な条件を設定できます。

以上で、ソース ファイルの任意の行でプログラムを停止する方法を確認しました。

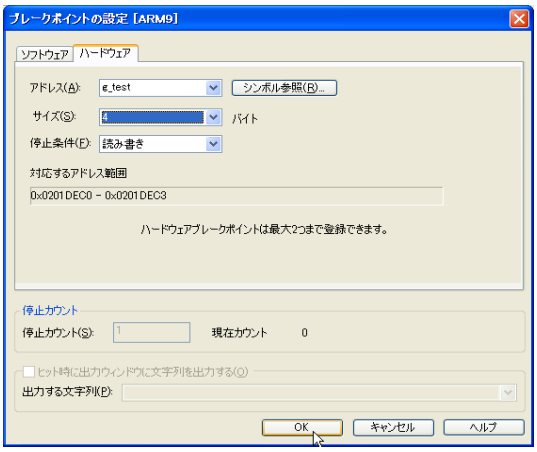
# 12. 意図しないメモリ書き換えの検出

この章では、意図しないメモリの書き換えを検出し、プログラムを停止する方法を確認します。
この機能はハードウェア ブレークポイントと呼び、主にプログラムのバグによりメモリや変数が意図せず書き換えられる原因箇所を特定する場合に使用します。

ここでは、サンプル プログラムのグローバル変数 g_test の書き換えを検出する手順を確認します。

① [ブレークポイント]ウィンドウを右クリックし、[追加]をクリックします。
② [ハードウェア]ページを開き、
1.[アドレス]ボックスに g_test を設定する。
2. [サイズ]ボックスに 4 を設定する（int 型）。
③ [OK]ボタンをクリックする。
以上で、ハードウェア ブレークポイントの設定は完了です。

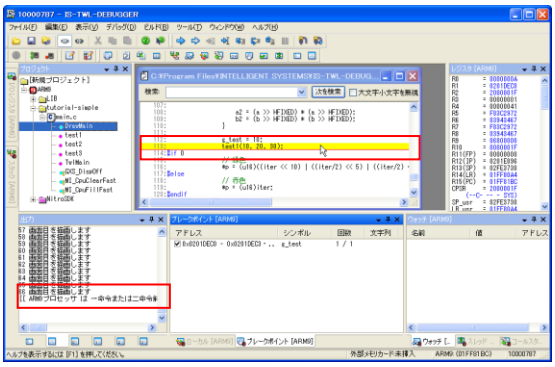
以降、プログラムを実行すると、変数 g_test の読み書きを検出し、プログラムを停止します。




続いてプログラムを実行し、g_test アクセス時にプログラムが停止することを確認します。
なお、サンプル プログラムは、main.c 112 行で変数 g_test に書き込みを行っています。

① プログラムを実行します。
② プログラムは [出力]ウィンドウに**[ARM9 プロセッサは 一命令または二命令前にハードウェア ブレークポイントが発生したため実行を停止しました]**を表示し、main.c 113 行でプログラムは停止します。

※ハードウェア ブレークポイントは、データを読み書きしたアセンブラの一命令または二命令後にプログラムを停止します。 サンプル プログラムの場合、main.c 112 行 g_test=0;のアセンブラ二命令後である main.c 113 行で停止します。



ハードウェア ブレークポイントによるメモリ読み書き検出は、ARM9CPU からのアクセスに限定されています。
ARM9CPU を経由しないアクセス(DMA の読み書き、ARM7CPU からの読み書きなど)は検出できません。

# 13. 行単位、関数単位の実行

この章では、プログラムを行単位、関数単位で実行する方法を確認します。

## 13.1 行単位、関数単位での実行の種類

IS-TWL-DEBUGGER は以下の行単位、関数単位のプログラム実行方法を使用できます。

| 実行方法 | 概要 |
|---|---|
| **トレース実行** | ソース1行分を実行。関数呼び出しの場合には、関数内の1行分を実行して、その関数内で停止する。 |
| **ステップ実行** | ソース1行分を実行。関数呼び出しの場合には、関数の中身をすべて実行して次の行で停止する。 |
| **ステップ アウト実行** | 現在プログラム カウンタのある関数から抜けだすまで、複数行まとめて実行し 関数呼び出し元で停止する。 |

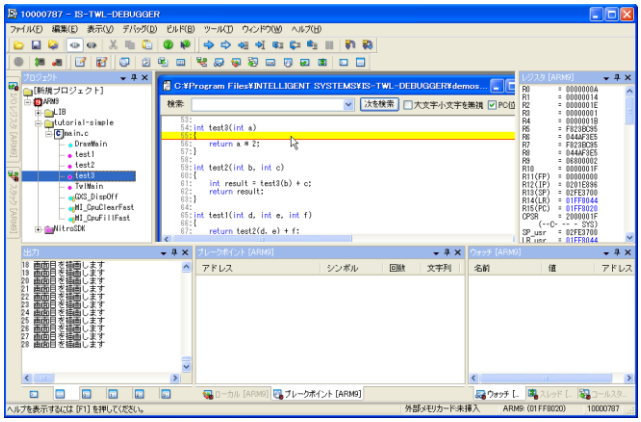
## 13.2 1行分実行(トレース実行)

プログラムのソース1行分を実行する機能をトレース実行と呼びます。
トレース実行は、ソース行が関数呼び出しの場合、関数の中身を1行分だけ実行して、[ソース]ウィンドウには関数内ソースを表示し停止します。

この項では、前項から続いてサンプル プログラム main.c の 61 行 test3()の呼び出しをトレース実行する手順について説明します。あらかじめプログラムを、main.c の 61 行に位置付けてください。

トレース実行を行うには、次のいずれかの操作を行います。

- ツール バーの[トレース実行]をクリックする。
- メニュー[デバッグ]-[トレース実行]をクリックする。
- F7 キーを押す。



トレース実行の完了後、[ソース]ウィンドウ main.c 55 行の test3()関数入口が黄色反転していることを確認します。

## 13.3 1行分実行（ステップ実行）

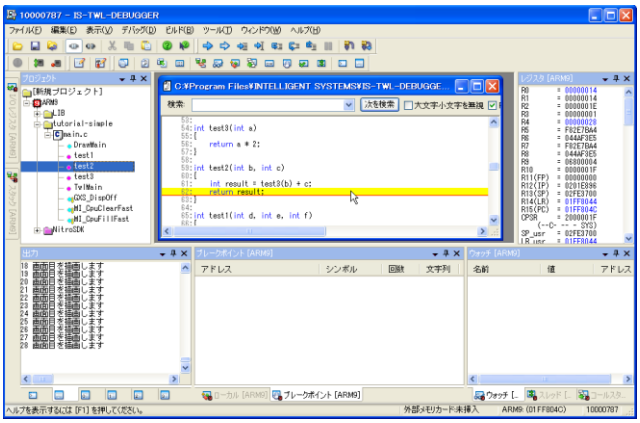
プログラムのソース1行分を実行する機能をステップ実行と呼びます。
ステップ実行は、ソース行が関数呼び出しの場合、関数の内容をすべて実行して次の行で停止します。

この項では、前項から続いてサンプル プログラムmain.cの61行 test3()の呼び出しをステップ実行する手順について説明します。あらかじめプログラムを、main.c の 61 行に位置付けてください。

ステップ実行を行うには、次のいずれかの操作を行います。

- ツール バーの[ステップ実行]をクリックする。
- メニュー[デバッグ]-[ステップ実行]をクリックする。
- F8 キーを押す。

ステップ実行完了後、[ソース]ウィンドウには main.c の 62 行が黄色反転し、test3()関数の内容をすべて実行したことを確認します。



## 13.4 関数を抜けるまで実行（ステップ アウト実行）

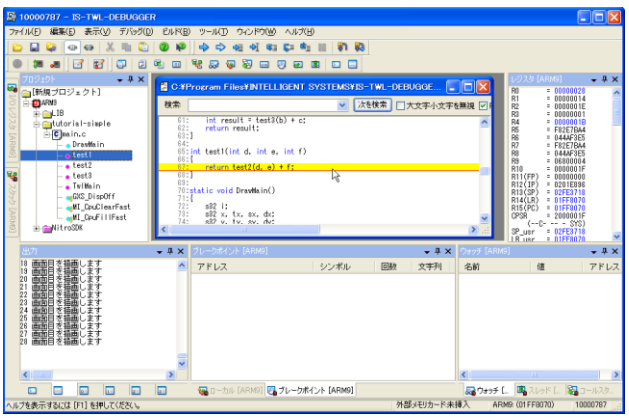
現在の関数を抜けるまで実行する機能をステップ アウト実行と呼びます。
ステップ アウト実行は C 言語、C++言語関数の関数内で使用できます。アセンブラで作成した関数はステップアウト実行できません。

この項では、前項から続いてサンプル プログラム main.c の 62 行目をステップ アウト実行する手順について説明します。

ステップ アウト実行を行うには、次のいずれかの操作を行います。

- ツール バーの[ステップ アウト実行]をクリックする。
- メニュー[デバッグ]-[ステップ アウト実行]をクリックする。
- Shift を押しながら F7 キーを押す。

ステップ アウト実行後、[ソース]ウィンドウは、test2 関数を抜けて、関数の呼び出し元 main.c の 67 行目を表示していることを確認します。



以上で1行単位、関数単位のプログラム実行方法を確認しました。

# 14. 変数値の確認

この章では、プログラム内の変数の値を確認する方法を確認します。

## 14.1 変数値の確認方法一覧

IS-TWL-DEBUGGER はプログラムの変数の値を参照、変更するため多くの方法を用意しています。
ここでは、プログラム内の変数参照、変更方法を紹介します。

| 変数確認方法 | 主な特徴や補足 |
|---|---|
| **データチップ** | ● [ソース]ウィンドウで、マウスを変数名にポイントするだけで変数の値をデータチップウィンドウと呼ばれるウィンドウに表示します。<br>● マウスで範囲選択した選択部分を C 言語式として評価できます。<br>● グローバル変数、ローカル変数に関わらず値を参照できます。<br>● 変数の値は変更できません。<br>● 配列などツリー展開表示が必要な変数は表示できません。 |
| **クイックウォッチ** | ● [ソース]ウィンドウを右クリックしメニューから[クイックウォッチ]ダイアログを表示します。配列や構造体など多階層のデータを表示できます。<br>● マウスで範囲選択した選択部分を C 言語式として評価できます。<br>● ローカル変数やグローバル変数に関わらず C 言語式を記述できます。<br>● 変数の値を変更できます。 |
| **ウォッチウィンドウ** | ● 配列や構造体など多階層のデータを表示できます。<br>● ローカル変数やグローバル変数にかかわらず、C 言語式を評価できます。<br>● ステップ実行やトレース実行後、前回と値が異なれば強調表示します。<br>● 変数の値を変更できます。 |
| **ローカルウィンドウ** | ● プログラム カウンタや[コールスタック]ウィンドウのカレントフレームに合わせて、関数のローカル変数を自動的に表示します。<br>● 配列や構造体など多階層のデータを表示できます。<br>● ステップ実行やトレース実行後、前回と値が異なれば強調表示します。<br>● 変数の値を変更できます。 |

# 15. [ソース]ウィンドウ上で変数値を確認（データチップ）

この章では、データチップにより変数の値を確認する方法を確認します。

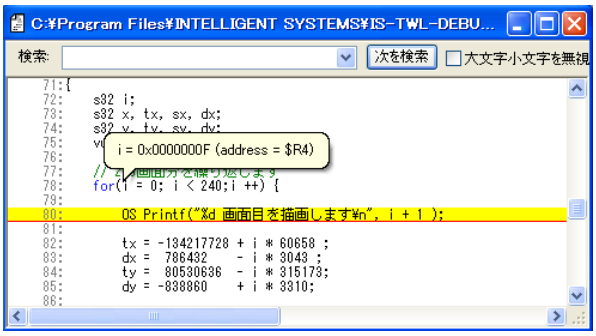
[ソース]ウィンドウ上の変数名にマウスをポイントすると、変数の値をツールチップウィンドウに表示します。この機能をデータチップと呼びます。データチップは最もすばやく変数の値を確認できます。

ここでは、サンプル プログラム main.c の 78 行 変数 i の値を確認します。あらかじめステップ機能などを使用して、main.c の 78 行目までプログラムを進めてください。

データチップによる変数 i を確認する方法は次の通りです。

① [ソース]ウィンドウ main.c の 78 行の変数 i にマウスをポイントします。

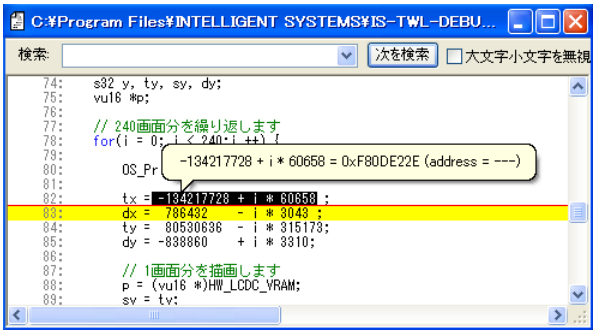
以上の操作で変数 i の値、および格納アドレスを表示します。変数が CPU レジスタに割り当てられている場合には、先頭に$マークをつけ、続いて R4 などの CPU レジスタ名を表示します。




データチップは、C 言語式も評価できます。
続いて、サンプル プログラム main.c の 82 行 の C 言語式 **-134217728 + i * 60658** の値を確認します。

① [ソース]ウィンドウ main.c の 82 行の C 言語式 **-134217728 + i * 60658** をマウスの左ボタンでドラッグし、式を範囲選択します。
② **-134217728 + i * 60658** をマウスでポイントします。

以上の操作でデータチップに C 言語式の値を表示します。なお**-134217728 + i * 60658** のように演算結果がアドレス上に存在しない場合には、アドレスに[---]を表示します。




データチップは変数の値を参照のみで、値を変更することはできません。

# 16. 変数値の参照と変更（クイックウォッチ）

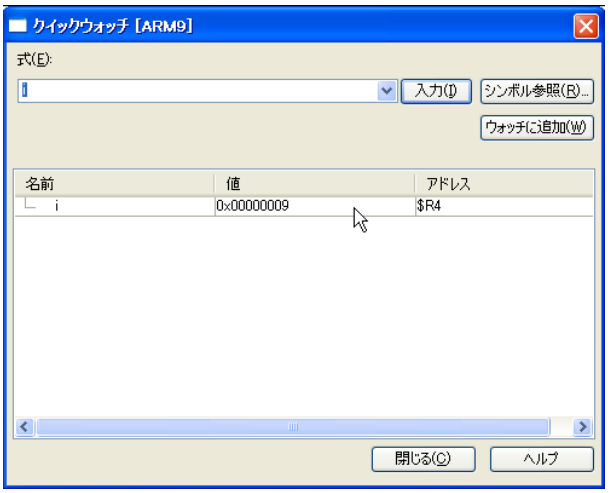
この章ではクイックウォッチによる変数の値表示、および変数の値変更方法を確認します。

［クイックウォッチ］ダイアログ ボックスを表示し、変数や C 言語式の内容を表示、変更する機能をクイックウォッチと呼びます。ここではサンプル プログラム main.c の 78 行 変数 i の値を表示、変更します。

サンプル プログラムの main.c 78 行変数 i の値を参照するには、以下の操作を行います。

① ［ソース］ウィンドウ main.c の 78 行の変数 i にマウスをポイントします。
② 右クリックし、メニューから［クイックウォッチ］をクリックし、［クイックウォッチ］ダイアログ ボックスを表示します。

［アドレス］列には、変数を格納するアドレスを表示します。変数が CPU レジスタに割り当てられている場合には、R4 など頭に$（ドルマーク）をつけた CPU レジスタ名を表示します。




つづいて変数 の値を変更する方法を確認します。ここでは、変数 i の値を 5 に変更します。

① ［値］列の変数の値をダブルクリックします。
② ［値］表示が［値］編集ボックスに変化するので、値 5 を入力し、Enter キーを押します。

以上で変数 i の値を変更できます。


クイックウォッチは、データチップ同様に［-134217728 + i * 60658］などの C 言語式も評価できます。

# 17. 変数値の参照と変更（ウォッチウィンドウ）

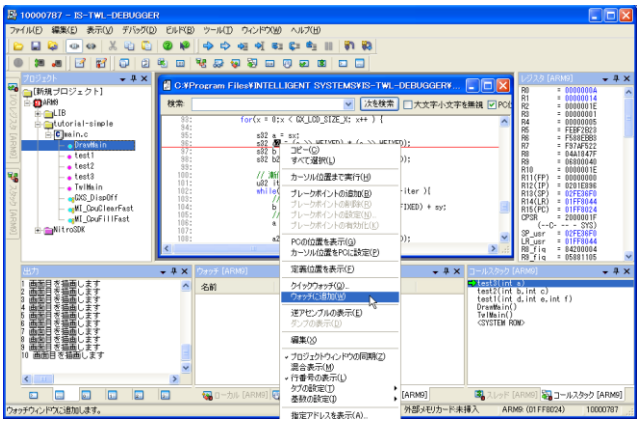
この章では、[ウォッチ]ウィンドウの使用方法を確認します。

[ウォッチ]ウィンドウは、変数や C 言語式を評価し表示、変更できるウィンドウです。[ウォッチ]ウィンドウは、変数の値を継続して表示し変数の値変化を確認する場合に有効です。

ここでは、サンプル プログラムの 96 行目　変数 a2 の値を継続して表示、確認します。

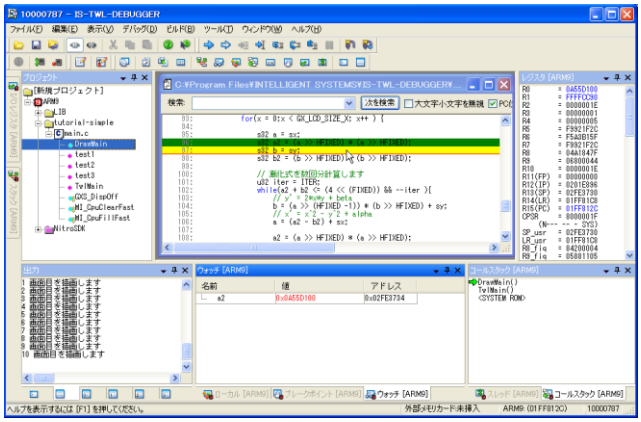
[ウォッチ]ウィンドウに変数を追加するには、つぎのいずれかの操作を行います。

- main.c の 96 行の変数 a2 を右クリックし、メニューから[ウォッチに追加]をクリックする。
- main.c の 96 行の変数 a2 を範囲選択する。範囲選択した領域を、ウォッチウィンドウにドラッグする。



[ソース]ウィンドウで単語を範囲選択するには、SHIFT キーを押しながら、単語を左クリックします。

サンプル プログラムは、78 行目から複数回プログラムをループします。ステップ実行を複数回実行し、変数 a2 の値が変化するたびに変数値が赤色表示されることを確認します。



以上で変数の値が変化を確認しながらステップ実行する方法を確認しました。

# 18. 関数呼び出し履歴の確認

この章では[コールスタック]ウィンドウにより、関数の呼び出し履歴を確認する方法を確認します。

[コールスタック]ウィンドウは、関数の呼び出し履歴であるコールスタックおよびカレントフレームを表示します。カレントフレームは、デバッガが注目する関数で、プログラム カウンタのアドレスに相当する関数です。カレントフレームは、[コールスタック]ウィンドウの緑色の矢印が指し示す関数です。またカレントフレームは[コールスタック]ウィンドウから変更できます。

[ローカル]ウィンドウには、カレントフレームのローカル変数を表示します。[コールスタック]ウィンドウの使用方法は、次章の[ローカル変数を表示、変更する]で[ローカル]ウィンドウの操作と合わせて説明します。

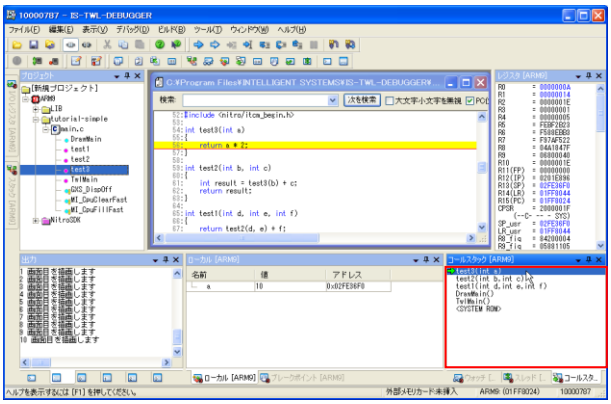
# 19. ローカル変数の表示と変更(ローカルウィンドウ)

この章では[ローカル]ウィンドウで、ローカル変数を確認、表示する方法を確認します。

[ローカル]ウィンドウは、カレントフレームの示すローカル変数一覧を表示します。カレントフレームが変更されると、[ローカル]ウィンドウもカレントフレームの変更に追随してローカル変数を表示します。

ここでは、[コールスタック]ウィンドウのカレントフレームの操作と、[ローカル]ウィンドウのローカル変数の表示と変更方法について確認します。サンプル プログラムを main.c の test3()でプログラムを停止させ、test3()に至るまでの経由した関数とそのローカル変数を表示する方法を確認します。なお、サンプル プログラムの場合、test3()に至るまでに、TwlMain()->DrawMain()->test1()->test2()->test3()の順に関数が呼びだされます。

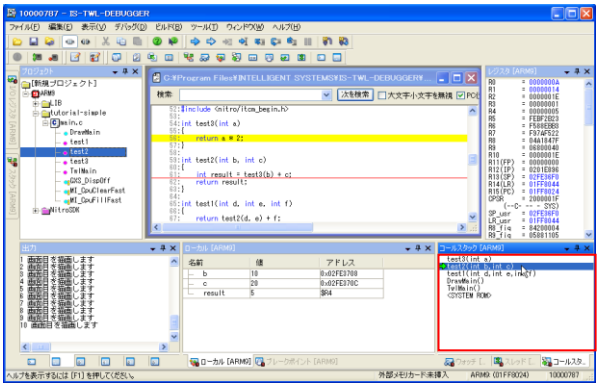
test3()に至るまでのコールスタックを確認するには、次の手順を行います。



① test3()にブレークポイントを設置し、test3()までプログラムを実行して停止させます。

[ローカル]ウィンドウには test3()関数のローカル変数 a が表示され、[コールスタック]ウィンドウは、test3()がカレントフレームになり、同時にTwlMain()->DrawMain()->test1()->test2()->test3()と関数が経由していることを確認します。

続いてカレントフレームを test2() に変更して、test2()関数のローカル変数を確認します。



① カレントフレームを変更するには、[コールスタック]ウィンドウの test2()をダブルクリックします。

以上でカレントフレームが test2()に移動します。

[ローカル]ウィンドウには、test2()のローカル変数 b と c を表示し、同時に[ソース]ウィンドウには test2()関数を表示することを確認します。

# 20. スレッドのデバッグ

この章では、TWLSDK のスレッドシステムのデバッグ方法について確認します。TWLSDK のスレッドシステムについては、『TWL プログラミングマニュアル』を参照してください。

## 20.1 カレントスレッド

IS-TWL-DEBUGGER は、TWL-SDK のスレッドシステムのデバッグに対応しています。[スレッド]ウィンドウはスレッドの一覧を表示し、カレントスレッドを切り替えてユーザーやシステムが作成した任意のスレッドをデバッグできます。

カレントスレッドとは、IS-TWL-DEBUGGER が注目するスレッドです。カレントスレッドに基づいて[レジスタ]ウィンドウ、[コールスタック]ウィンドウ、[ソース] ウィンドウ、[ローカル]ウィンドウの値を表示します。

## 20.2 スレッドを切り替える

[スレッド]ウィンドウはスレッドの一覧を表示し、カレントスレッドを緑色の矢印で表現します。

ここでは、サンプル プログラムを使用し、カレントスレッドをユーザースレッド ID0 からシステムスレッド ID3 に変更する手順を確認します。合わせて、カレントスレッド変更により、[レジスタ]ウィンドウや[コールスタック]ウィンドウなどの各ウィンドウの値が切り替わることを確認します。

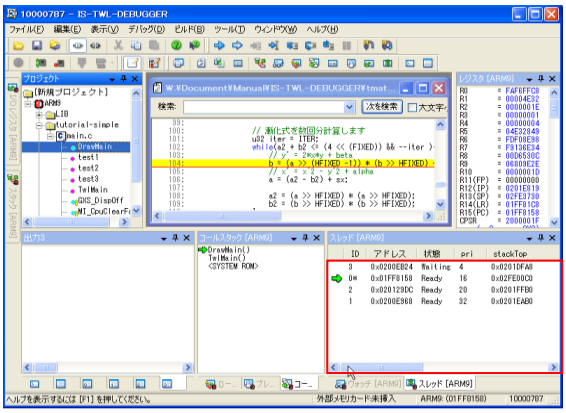
カレントスレッドを切り替える手順は次の通りです。

① カレントスレッドが ID 0 であることを確認します。
合わせて現在の[コールスタック]ウィンドウ、[レジスタ]ウィンドウの値や、[ソース]ウィンドウの表示位置を確認します。

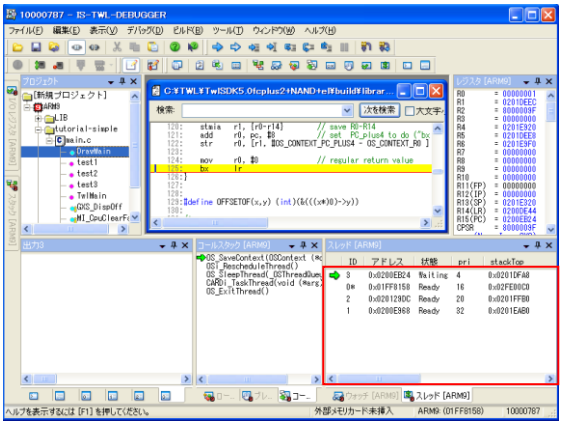
② [スレッド]ウィンドウの 1 行目 ID 3 をダブルクリックし、カレントスレッドを ID 3 に切り替えます。

以上で ID3 にカレントスレッドを切り替わります。
カレントスレッドの変更により、[コールスタック]ウィンドウ[レジスタ]ウィンドウの値や、[ソース]ウィンドウのソース表示位置が変化することを確認します。

**カレントスレッド変更前**



**カレントスレッド変更後**



# 21. メモリ内容の表示

この章では、[ダンプ]ウィンドウを使用して、TWL 本体内のメモリ、およびエミュレーションメモリの内容を表示、変更する方法を確認します。

## 21.1 IS-TWL-DEBUGGER のメモリ空間

TWL 本体のメモリとエミュレーションメモリは、別のメモリ空間に存在します。[ダンプ]ウィンドウは、[メモリ空間選択]ボックスにより、表示するメモリ空間を選択します。[メモリ空間選択]ボックスから[ARM9]を選択すると TWL 本体内に搭載しているビデオ RAM やプログラムワークメモリ、I/O レジスタなどを表示、変更できます。[エミュレーションメモリ]を選択すると DS カードに相当するエミュレーションメモリを表示、変更できます。

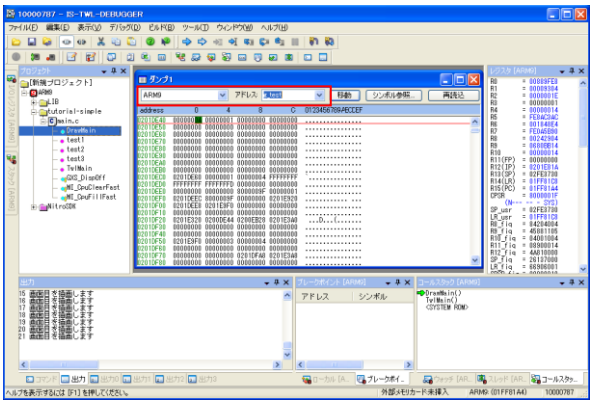
なお、TWL 本体内に搭載している I/O レジスタなど一部のメモリは書き込み専用、または読み出し専用のメモリがあり、これらメモリを [ダンプ]ウィンドウに表示しても正しい値は表示しません。各メモリの特性については、『TWL プログラミングマニュアル』を参照してください。

## 21.2 TWL 本体内メモリ内容の表示と変更

この項では、[ダンプ]ウィンドウのメモリ空間を変更する方法を確認し、サンプル プログラムのグローバル変数 g_test の値を確認する手順を確認します。

[ダンプ]ウィンドウで、グローバル変数 g_test の値を確認する手順は次の通りです。

① [ダンプ]ウィンドウの上部[メモリ空間選択]ボックスから[ARM9]を選択します。
② [アドレス]ボックスに g_test を入力し、[移動]ボタンをクリックします。
③ 以上で変数 g_test のアドレスを緑色反転し値を表示します。



[アドレス]ボックスには、変数名以外に[0x02000000]など直接アドレスを指定できます。

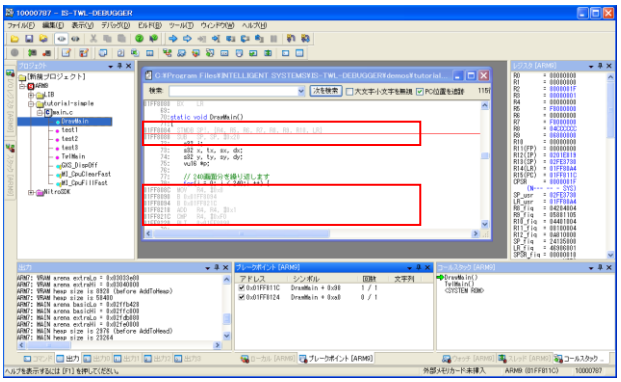
# 22. [ソース]ウィンドウでのアセンブラ混合表示

この章では、[ソース]ウィンドウに対応するアセンブラを混合表示する方法を確認します。

[ソース]ウィンドウは、ソース1行1行に対応するアセンブラを埋め込んで混合表示できます。コンパイラの出力したアセンブルコードを確認したい、またアセンブラレベルで挙動を確認したい場合などに利用します。

ここでは、サンプル プログラムの main.c の DrawMain 関数付近を混合表示する手順を説明します。

サンプル プログラム main.c の DrawMain 関数付近を混合表示する手順を説明します。



[ソース]ウィンドウを混合表示するには、次の操作を行います。

① [プロジェクト]ウィンドウから DrawMain をダブルクリックし、[ソース]ウィンドウに DrawMain を表示します。

② [ソース]ウィンドウの DrawMain 関数 70 行を右クリックし、[混合表示]をクリックします。

以上で[ソース]ウィンドウは混合表示に切り替わります。

混合表示中の[ソース]ウィンドウは、次のような挙動になります。

- ブレークポイントは、逆アセンブル表示行に設置します。
- トレース実行、ステップ実行は、アセンブラ 1 命令を実行後、プログラムを停止します。
- アセンブラ表示行の行番号表示部分には、プログラムのアドレスを表示します。

以上で[ソース]ウィンドウにアセンブラを混合表示する方法を確認しました。

# 23. 逆アセンブル表示

この章では、[逆アセンブル]ウィンドウにアセンブラを表示する方法を確認します。
[逆アセンブル]ウィンドウは、プログラムの逆アセンブルを表示します。コンパイラの出力結果を確認や、デバッグ情報のないプログラムをアセンブラレベルデバッグしたい場合に利用します。

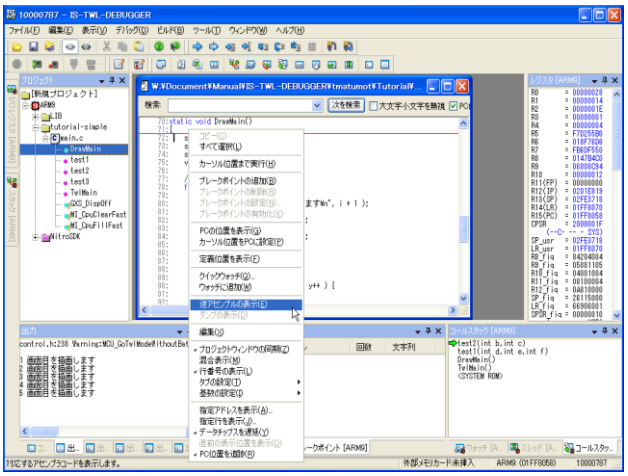
## 23.1 ソースウィンドウに対応する逆アセンブルの表示

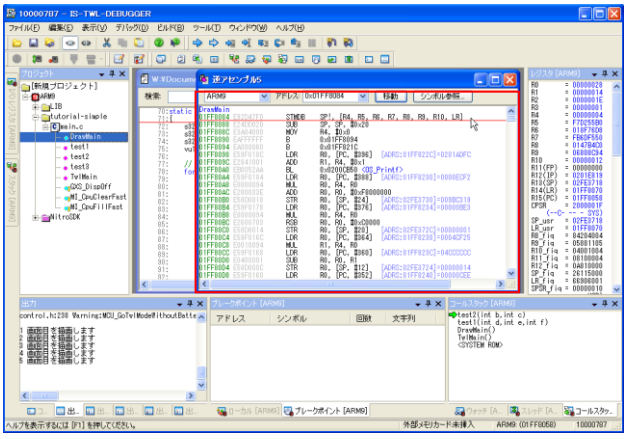
[ソース]ウィンドウから、表示中のソースに対応する逆アセンブルを表示できます。ここでは、サンプル プログラム main.c の DrawMain 関数を逆アセンブル表示する手順を確認します。

[ソース]ウィンドウに対応する逆アセンブルを表示する手順は次の通りです。

① [ソース]ウィンドウの DrawMain 関数を右クリックし、[逆アセンブルの表示]をクリックします。

以上で、DrawMain 関数を[逆アセンブル]ウィンドウに表示します。



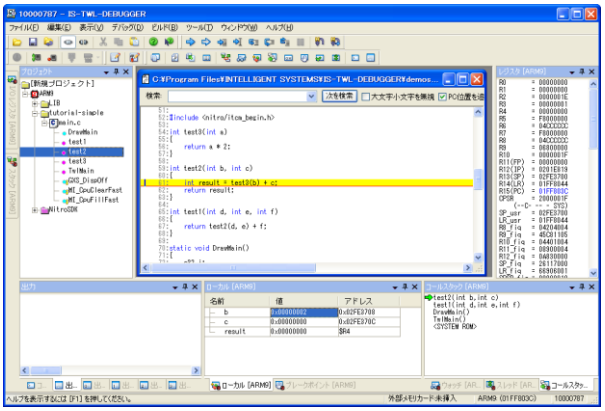


# 24. ソースの任意行にプログラムカウンタを移動する

この章ではソースの任意の行にプログラムカウンタを移動する方法を確認します。
この処理は[レジスタ]ウィンドウで、プログラムカウンタを変更する動作と等価です。関数の引数を変更し、同じ関数を繰り返し関数の動作結果をする場合に便利な機能です。

ここでは、サンプル プログラムの main.c の test3()関数呼び出し時の引数を変更して、繰り返し test3()関数の動作を確認する方法を確認します。

同じ関数を繰り返し動作確認するには、次の操作を行います。



① あらかじめ main.c 61 行の test3 関数の呼び出し直前までプログラムを実行します。
**61 行 int result = test3(b) + c**
② [ローカル変数を表示、変更する]を参考に変数 b に 1 c に 0 を設定して、ステップ実行を行います。
③ プログラムカウンタは
62 行の return result;
に移動します。変数result の値は 2 になっています。
④ 続いて[ソース]ウィンドウの 61 行をクリックし、[カーソル位置を PC に設定]をクリックし、プログラムカウンタを 61 行に戻します。
⑤ 再び[ローカル]ウィンドウから変数 b に 2 を設定し、再びステップ実行を行います。
プログラムカウンタは再び 62 行に移動し、result の値が 4 になっていることを確認します。

以上で、関数の引数を変更し、繰り返し関数の動作を確認する方法を確認しました。

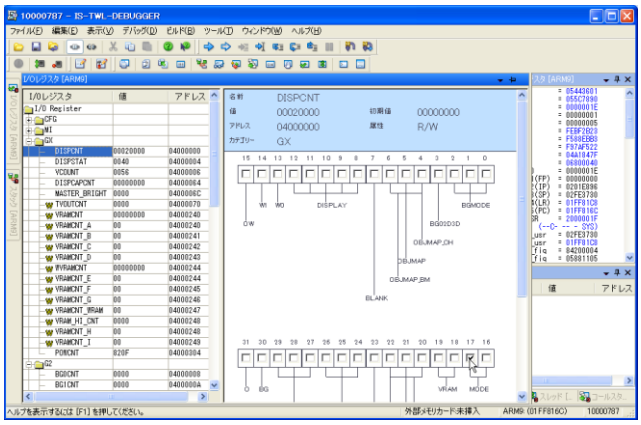
# 25. I/O レジスタの表示と変更

この章では I/O レジスタの表示、変更方法を確認します。
スクリーンの表示、非表示や画面スクロール位置などレジスタなど、任意の I/O レジスタを表示、変更できます。

ここでは、TWL-SDK の メイングラフィックスエンジンの表示オフ、オン制御関数 GX_DispOff()に相当する処理を[I/O レジスタ]ウィンドウから確認します。あらかじめサンプル プログラムを読み込み、プログラムを実行しマンデブロー図を IS-TWL-DEBUGGER ハードウェアの液晶表示部に表示した時点でプログラムを停止してください。

[I/O レジスタ[ARM9]]ウィンドウから、I/O レジスタを変更するには、以下の操作を行います。

① メイン ウィンドウの左端の[I/O レジスタ[ARM9]]ウィンドウをクリックします。
② [I/O レジスタ]ツリー[GX]-[DISPCNT]を表示します。
③ 右側[I/O 詳細表示]からビット 17 のチェックをオフします。
以上で IS-TWL-DEBUGGER ハードウェア コントローラ部の液晶画面が非表示されることを確認します。
④ 続いて右側[I/O 詳細表示]からビット 17 のチェックをオンします。
以上で、再び IS-TWL-DEBUGGER ハードウェア コントローラ部の液晶画面が表示されます。



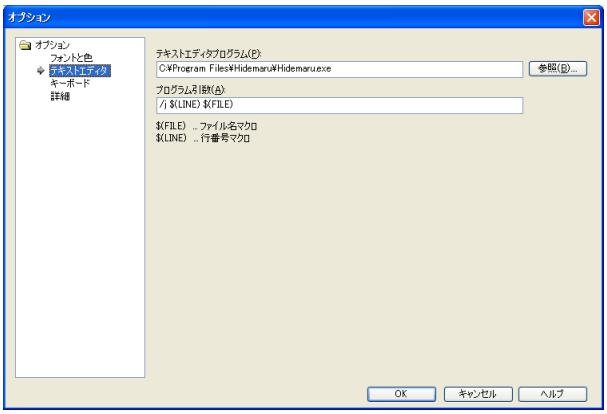
# 26. ソース ファイルの編集

この章では、IS-TWL-DEBUGGER 上から、市販のテキスト エディタを起動する方法を確認します。
[ソース]ウィンドウに表示中の指定ソース行や、[プロジェクト]ウィンドウから任意の関数を市販のテキスト エディタに表示できます。

## 26.1 テキスト エディタの起動設定

IS-TWL-DEBUGGER から外部テキスト エディタを開くには、あらかじめ以下の設定が必要です。

① メニューの[ツール]-[オプション]をクリックし、[オプション]ダイアログを表示し、[テキスト エディタ]ページを表示します。
② [テキスト エディタプログラム]入力ボックスに、テキスト エディタのプログラム名を指定します。
③ [プログラム引数]入力ボックスに、ファイル名および行番号を指定する文字列を指定します。
（なお、文字列中の$(FILE) は[ソース]ウィンドウのファイル名に、 $(LINE)は[ソース]ウィンドウの編集中の行に置き換えられます。）
④

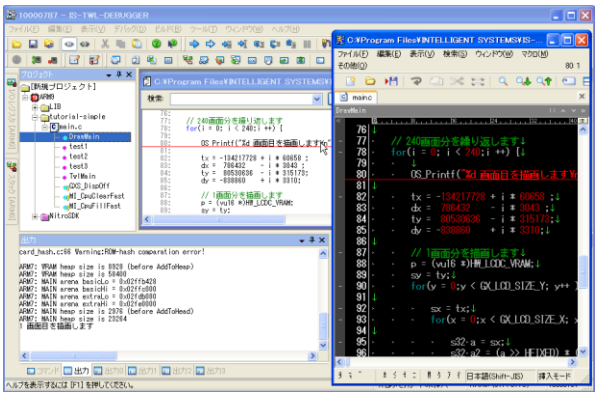



以上で外部テキスト エディタを起動するための設定は完了です。

任意のソース ファイルを開くには、次のいずれかの操作を行います。

- [ソース]ウィンドウで、SHIFT キーを押しながら Enter キーを押す。
- [プロジェクトウィンドウ]のツリーの関数ノードを右クリックし、メニューから[編集]をクリックする。

以上でソース ファイルの任意行を開くことができます。



ここでは、サンプル プログラムの main.c の114行目の#if 1 を #if 0 に変更し、以下の挙動になるよう修正します。

変更前 `*p = (u16)((iter << 10) | ((iter/2) << 5) | ((iter/2) << 5));`
変更後 `*p = (u16)iter;`

この変更はマンデルブロー図の色を青色から赤色に変更するものです。次章ではこの変更を反映するため、IS-TWL-DEBUGGER からビルドを行います。

# 27. プログラムのビルド設定

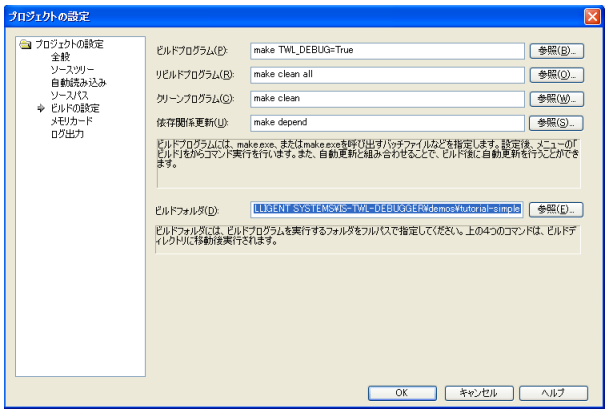
この章では、IS-TWL-DEBUGGER からプログラムをビルドするために必要な設定を行います。
IS-TWL-DEBUGGER は外部プログラム make.exe を起動してプログラムをビルドします。

ビルド時のカレントフォルダ、ビルド時に実行するコマンドなど、プロジェクトに固有なビルド設定を行います。ここでは、サンプル プログラムをビルドするための手順を確認します。

プロジェクトのビルドを行う設定は次の通りです。

① メニューの[ビルド]-[ビルドの設定]をクリックし、[ビルドの設定]ダイアログを表示します。
② [ビルドプログラム]や[リビルドプログラム]には、プログラムをビルドするためのコマンドを指定します。
③ [ビルドフォルダ]入力フォルダには、
**C:¥Program Files¥INTELLIGENT SYSTEMS¥IS-TWL-DEBUGGER¥demos¥tutorial-simple**
を入力します。

以上でサンプルをビルドするための設定は終了です。




[プロジェクトの設定]ダイアログで設定した外部プログラムは、次のメニューから呼び出すことができます。

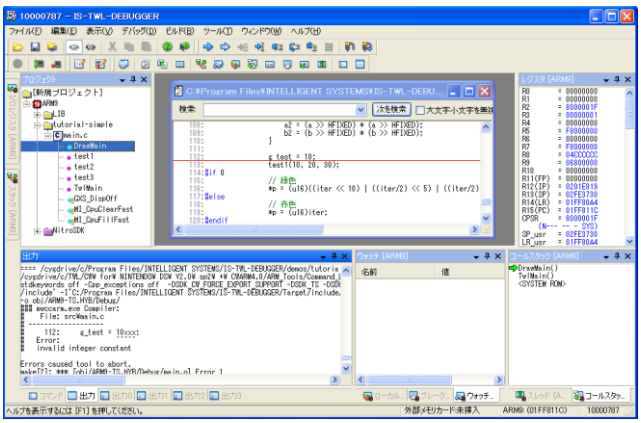
| ダイアログの項目 | 該当するメニュー項目 |
|---|---|
| **ビルドプログラム** | [ビルド]-[ビルド] |
| **リビルドプログラム** | [ビルド]-[リビルド] |
| **クリーンプログラム** | [ビルド]-[クリーン] |
| **依存関係更新** | [ビルド]-[依存関係の更新] |

以上で IS-TWL-DEBUGGER からビルドを行うために必要な設定方法を確認しました。

# 28. プログラムのビルド

この章では、IS-TWL-DEBUGGER からプログラムをビルドする方法を確認します。
前章『プログラムのビルド設定を行う』の設定を使用して、サンプル プログラムをビルドする方法を確認します。

① メニューの[ビルド]-[ビルド]をクリックします。
② ビルド中のコンパイルエラーや警告表示は、すべて[出力]ウィンドウに表示します。



以上でプログラムのビルドは完了します。

# 29. エラータグジャンプ

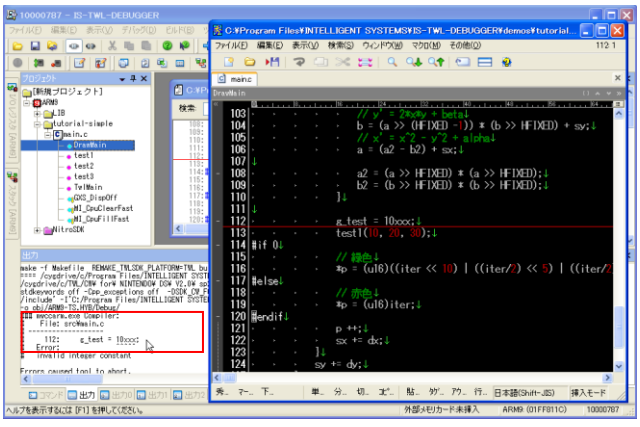
この章では、プログラムのビルド時のコンパイルエラー行をテキスト エディタで表示する方法を確認します。
ビルド時のコンパイルエラーは [出力]ウィンドウに表示します。このコンパイルエラーダブルクリックし、外部テキスト エディタを起動、すぐに編集できます。この機能をエラータグジャンプと呼びます。

ここでは、あらかじめサンプル プログラム main.c 112 行を以下のように編集し、プログラムビルド時にエラーがでるようにしておきます。

**main.c 112 行 g_test = 10xxx;**

エラータグジャンプを行う手順は次の通りです。

① メニューの[ビルド]-[ビルド]をクリックし、プログラムをビルドします。
IS-TWL-DEBUGGER はビルド時に、[出力]ウィンドウにコンパイルエラーを表示します。
② [出力ウィンドウ]のコンパイルエラー行をダブルクリックします。



以上で、対応するソース ファイルを外部テキスト エディタで起動します。なお、テキスト エディタの設定については、16.1 テキスト エディタの起動設定をご確認ください。

以上でコンパイルエラーの発生したソース ファイルのエラー行をテキスト エディタで起動し、編集する方法を確認しました。

# 30. プログラムを自動的に読み込む

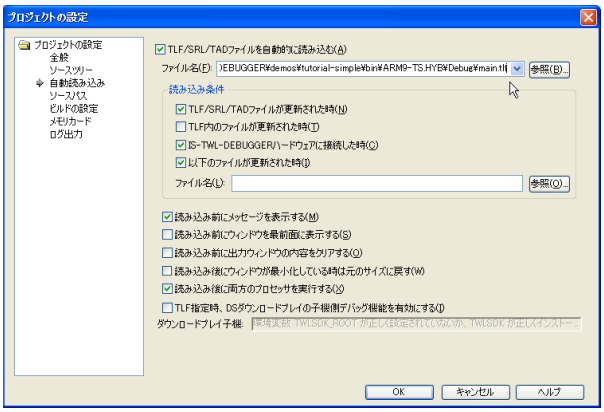
この章では、プログラムのビルド成功後や IS-TWL-DEBUGGER 起動時にプログラムを自動読み込みする機能について確認します。

ここではサンプル プログラムを自動読み込みする設定を確認します。

① メニューの[ファイル]-[プロジェクトの設定]をクリックし、[自動読み込み]ページを表示します。
② [TLF/SRL/TAD ファイルを自動的に読み込む]チェックボックスをチェックします。
③ [ファイル名]入力ボックスにサンプル プログラムを入力します。

C:¥Program Files¥INTELLIGENT SYSTEMS¥IS-TWL-DEBUGGER¥demos¥tutorial-simple¥bin¥ARM9-TS.HYB¥Debug¥main.tlf

④ [OK]ボタンをクリックして設定を反映します。




以降、プログラムをビルド時や、IS-TWL-DEBUGGER の起動時にファイルを自動読み込みします。

読み込み条件や、読み込み後の動作など各種設定については、『IS-TWL-DEBUGGER のヘルプ』を参照してください。

# 31. プロジェクト設定の保存

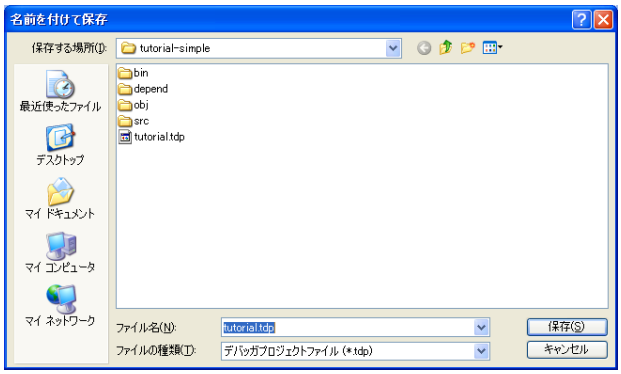
この章では、プロジェクトごとの設定を保存する方法を確認します。

IS-TWL-DEBUGGER では、ビルドの設定やプログラムの自働読み込み機能など、プロジェクト固有の設定をプロジェクトファイルに保存できます。複数のプロジェクトを並行して進める場合など、プロジェクトを切り替える必要がある場合には、プロジェクトファイルに保存すると

ここではサンプル プログラムの設定をプロジェクトファイルに保存する手順を説明します。

① メニューの[ファイル]-[プロジェクトの保存]をクリックし、[開く]ダイアログを表示します。
② [ファイル名]ボックスに tutorial.tdp を入力し、プロジェクトを保存します。

保存したプロジェクトは、[ファイル]-[プロジェクトを開く]をクリックし、後から読み込むことができます。



# 32. TWL 本体内メモリ内容の保存

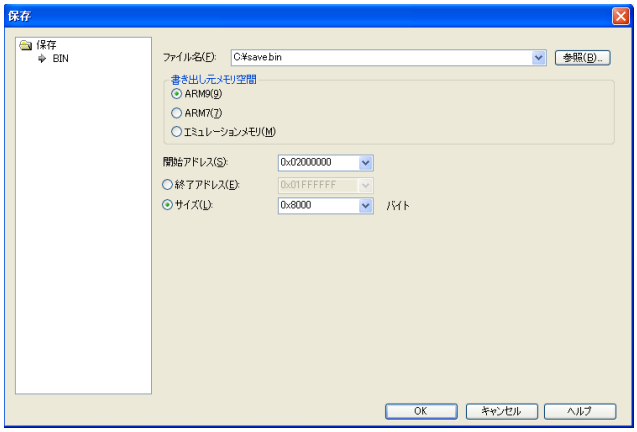
この章では、TWL 本体内のメモリやエミュレーションメモリをファイルに保存する方法を確認します。

プログラム中で保存したキー入力のログを保存するなど、TWL 本体内のメモリをファイルに保存したい場合に使用します。
ここでは、TWL 本体内のメモリ 0x02000000 から 0x8000 分をファイル C:¥save.bin に保存する方法を確認します。

TWL 本体内メモリ 0x02000000 から 0x8000 分をファイルに保存する手順は次の通りです。

① メニューの[ファイル]-[保存]をクリックして、[保存]ダイアログを表示します。
② 右図のように、ダイアログ項目を設定します。
[ファイル名]ボックスに C:¥save.bin、[書き出し元メモリ空間]を ARM9、[開始アドレス]に 0x02000000、[サイズ]に 0x8000 を設定する。
③ [OK]ボタンをクリックします。

以上で TWL 本体内メモリ 0x02000000 から 0x8000 分をファイル C:¥save.bin に保存します。



# 33. コマンドによる制御

この章では、IS-TWL-DEBUGGER をコマンドで制御する方法を確認します。

IS-TWL-DEBUGGER は、主な機能をコマンドとして[コマンド]ウィンドウから実行できます。
一連のコマンドをファイルに保存し、そのファイルを読み込んでまとめてコマンドを実行することもできます。
使用できるコマンドの一覧やコマンドの詳細『IS-TWL-DEBUGGER のヘルプ』を参照してください。

## 33.1 コマンドの実行

コマンドは、[コマンド]ウィンドウから実行できます。
ここでは、コマンドによりサンプル プログラムを読み込む方法を確認します。

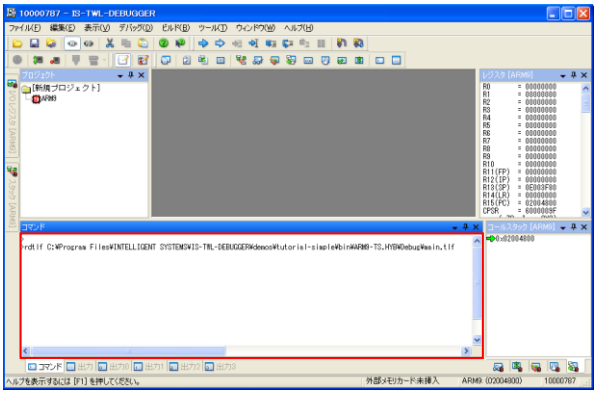
コマンドによるサンプル プログラムの読み込み手順は次の通りです。

① [コマンド] ウィンドウをクリックし、[コマンドウィンドウ]にカーソルを表示します。

② TLF ファイルの読み込みコマンド RDTLF を入力し、Enter を押します。

```
rdtlf C:¥Program Files¥INTELLIGENT SYSTEMS¥IS-TWL-DEBUGGER¥demos¥tutorial-simple¥bin¥ARM9-TS.HYB¥Debug¥main.tlf
```

以上でコマンドにより IS-TWL-DEBUGGER にサンプルプログラム main.tlf を読み込みます。



## 33.2 繰り返し操作の自動化

IS-TWL-DEBUGGERは、一連の操作をコマンドとしてテキストファイルに保存し、テキストファイルを読み込み、繰り返し行う操作を自動化できます。このテキストファイルは ICE(アイス)ファイルと呼び、拡張子は.ICE として保存します。

サンプル プログラムのフォルダには、サンプルの ICE ファイル sample.ice を用意しています。

**C:¥Program Files¥INTELLIGENT SYSTEMS¥IS-TWL-DEBUGGER¥demos¥tutorial-simple¥sample.ice**

サンプルの ICE ファイル sample.ice の内容は次の通りです。

① サンプル プログラム main.tlf を読み込む。
② 現行のブレークポイントをすべて削除する。
③ DrawMain 関数にブレークポイントを設置する。
④ ソフトウェア実行し、DrawMain 関数まで実行する。

sample.ice の内容

```
rdtlf   bin¥ARM9-TS.HYB¥Debug¥main.tlf
bc   *
bp   DrawMain
gs
```

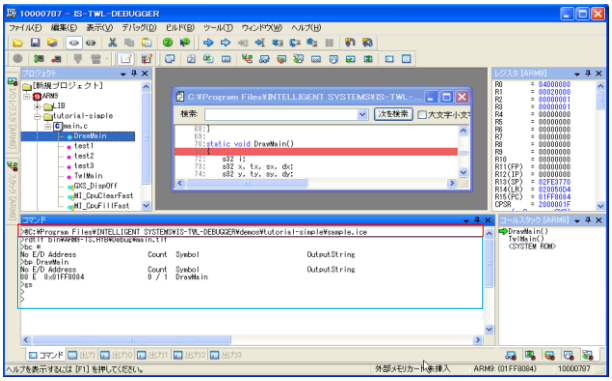
ここでは、sample.ice を使用して ICE ファイルの実行手順を確認します。

sample.ice を実行する手順は次の通りです。

① [コマンド] ウィンドウをクリックし、[コマンドウィンドウ]にカーソルを表示します。
② @(アットマーク)につづいて、ICE ファイル名を入力し、Enter を押します。

**@C:¥Program Files¥INTELLIGENT SYSTEMS¥IS-TWL-DEBUGGER¥demos¥tutorial-simple¥sample.ice**

以上で sample.ice 内に記述された一連のコマンドを実行します。右図の青枠は sample.ice により実行された内容です。



# 34. EL ライブラリを使用したデバッグ

この章ではELライブラリを使用してデバッグする場合の注意点、およびELライブラリを使用しない場合との相違点などを確認します。なお、EL ライブラリに関する詳しい説明は、『TWL プログラミングマニュアル』をご確認ください。

## 34.1 EL ライブラリの概要

EL ライブラリは、プログラム実行時にメイン プログラムとリンク、アンリンクできる動的ロードライブラリです。

IS-TWL-DEBUGGER は EL ライブラリを利用した動的ロードライブラリのデバッグに対応しています。便宜的にメモリにロード中の動的ロードライブラリを「ロード済みライブラリ」、メモリにロードしていない動的ロードライブラリを「未ロードのライブラリ」と呼びます。

## 34.2 EL ライブラリの制限

EL ライブラリは、ライブラリをメモリにロードした後アドレスを確定するため、「未ロードのライブラリ」はアドレスをもっていません。アドレスを必要とするデバッグ操作は、「未ロードのライブラリ」には使用できません。「未ロードのライブラリ」には以下の制限があります。

- 「未ロードのライブラリ」は、[逆アセンブル]ウィンドウからブレークポイントを設置できません。
- 「未ロードのライブラリ」は、[ソース]ウィンドウで混合表示することはできません。
- 「未ロードのライブラリ」は、[ソース]ウィンドウから、[逆アセンブルウィンドウを表示]メニューを使用することはできません。

「未ロードのライブラリ」もアドレスを使用しないデバッグ操作は制限なく利用できます。

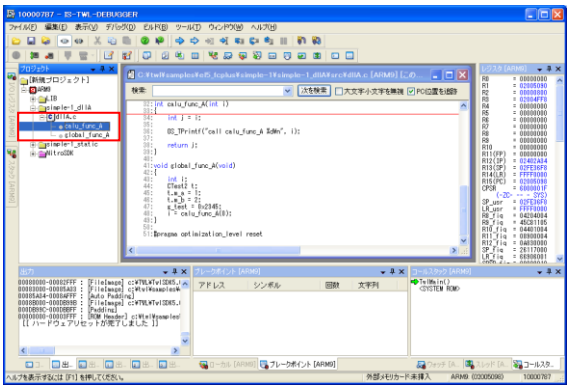
- 「未ロードのライブラリ」に含まれるソース ファイルも、[プロジェクト]ウィンドウのツリー構造に表示します。
- 「未ロードのライブラリ」にも、[ソース]ウィンドウからソフトウェアブレークポイントを設置できます。未ロードのライブラリをメモリにロードし、確定したアドレスにブレークポイントを自動的に設置します。※「未ロードのライブラリ」はメモリにはロードされていないため、[逆アセンブル]ウィンドウからブレークポイントを設置できません。

## 34.3 EL ライブラリのロード状態を確認する

動的ロードライブラリが、メモリにロード済みかの判断は次の手順で確認できます。

① [プロジェクト]ウィンドウのツリー構造をクリックし、確認したい動的ライブラリのツリーをクリックします。
② 関数ノードが灰色の場合、動的ライブラリはメモリにロードされていません(「未ロードのライブラリ」)。灰色以外の場合、動的ライブラリはメモリにロード済みです(「ロード済みのライブラリ」)。

右図のサンプルの場合、動的ライブラリ dllA.c は「未ロード状態」です。



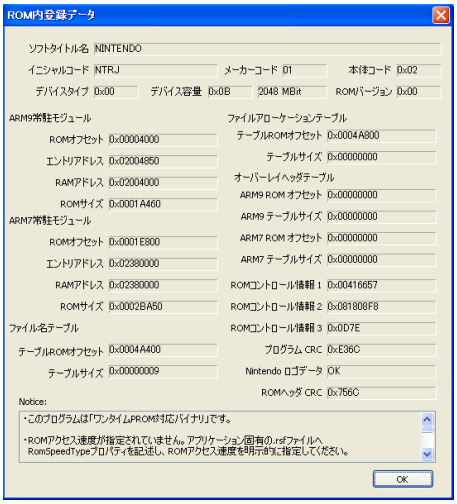
# 35. ROM 内登録データの確認

この章では ROM 内登録データの確認方法を確認します。
IS-TWL-DEBUGGER は、ROM 内登録データを確認し誤りを見つける方法を用意しています。ここでは、サンプルの ROM 内登録データを確認します。

サンプルの ROM 内登録データを確認するには、次の手順を行います。

① メニューの[ツール]-[ROM 内登録データ]をクリック、[ROM 内登録データ]ダイアログ ボックスを開きます。

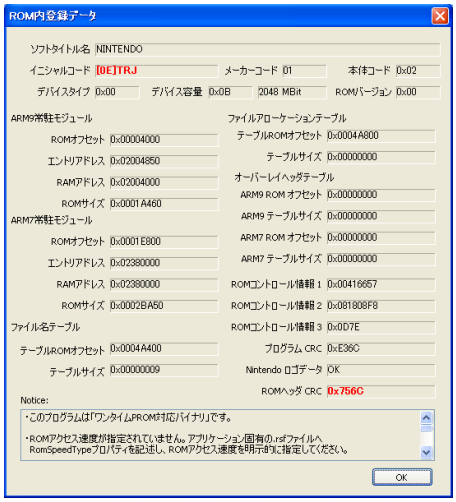
以上で ROM 内登録データを表示、確認できます。また Notice には、ROM 情報に関する注意点を表示します。




右図は ROM 内登録データのイニシャルコードが正しくない場合の例です。

ROM 内登録データで問題のある個所は赤色で表示し修正を促します。IS-TWL-DEBUGGER はプログラムに問題があっても内部的に ROM ヘッダを自動訂正しプログラムを実行できるようにします。ただし、TWL 実機では誤った ROM 内登録データのままではプログラムを実行できません。

出荷前の ROM 内登録データの確認は十分に行ってください。



# 36. サポート

インテリジェントシステムズでは、IS-TWL-DEBUGGER をご使用のお客様をサポートするために電子メールによるサポート窓口を用意しています。

## 36.1 サポート窓口

IS-TWL-DEBUGGER のお問い合わせは、サポート窓口（TWL_Support@intsys.co.jp）まで電子メールにてご連絡ください。

## 36.2 著作権と商標について