

CMOS 32-BIT SINGLE CHIP MICROCONTROLLER
(S1C33 Family C/C++コンパイラパッケージ) (Ver. 4.0.0)

S5U1C33001C

マニュアル

本資料のご使用につきましては、次の点にご留意願います。

本資料の内容については、予告なく変更することがあります。

1. 本資料の一部、または全部を弊社に無断で転載、または、複製など他の目的に使用することは堅くお断りいたします。
2. 本資料に掲載される応用回路、プログラム、使用方法等はあくまでも参考情報であり、これら起因する第三者の知的財産権およびその他の権利侵害あるいは損害の発生に対し、弊社はいかなる保証を行うものではありません。また、本資料によって第三者または弊社の知的財産権およびその他の権利の実施権の許諾を行うものではありません。
3. 特性値の数値の大小は、数直線上の大小関係で表しています。
4. 本資料に掲載されている製品のうち「外国為替及び外国貿易法」に定める戦略物資に該当するものについては、輸出する場合、同法に基づく輸出許可が必要です。
5. 本資料に掲載されている製品は、生命維持装置その他、きわめて高い信頼性が要求される用途を前提としていません。よって、弊社は本(当該)製品をこれらの用途に用いた場合のいかなる責任についても負いかねます。
6. 本資料に掲載されている会社名、商品名は、各社の商標または登録商標です。

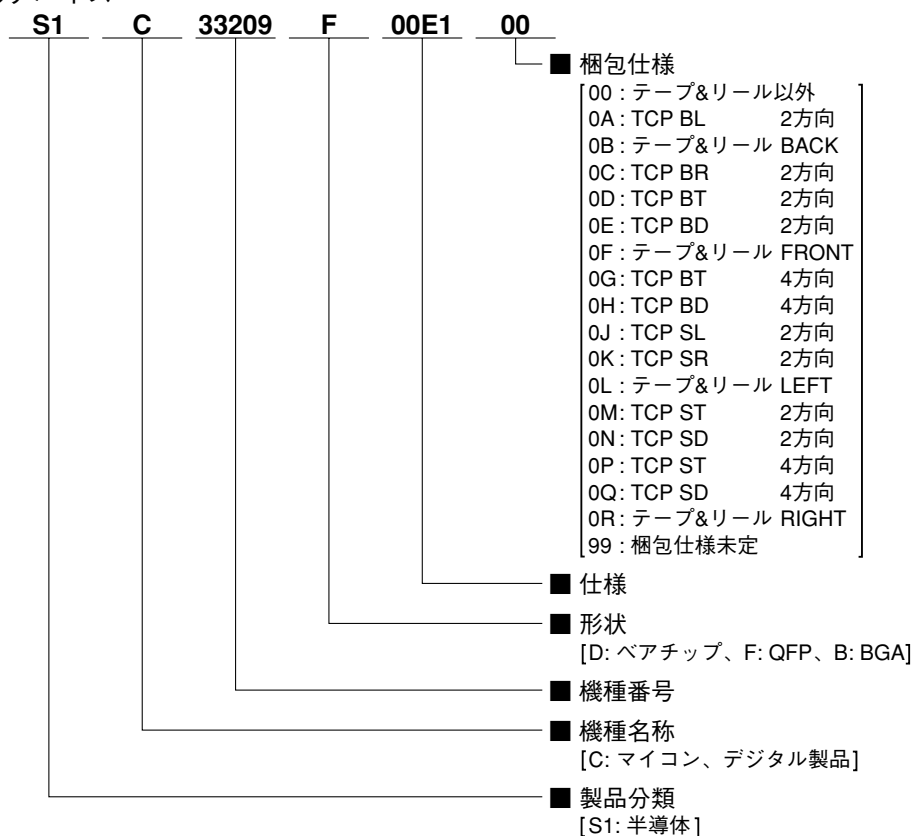
Windows XPおよびWindows Vistaは米国マイクロソフト社の登録商標です。

PC/ATおよびIBMは米国International Business Machines社の登録商標です。

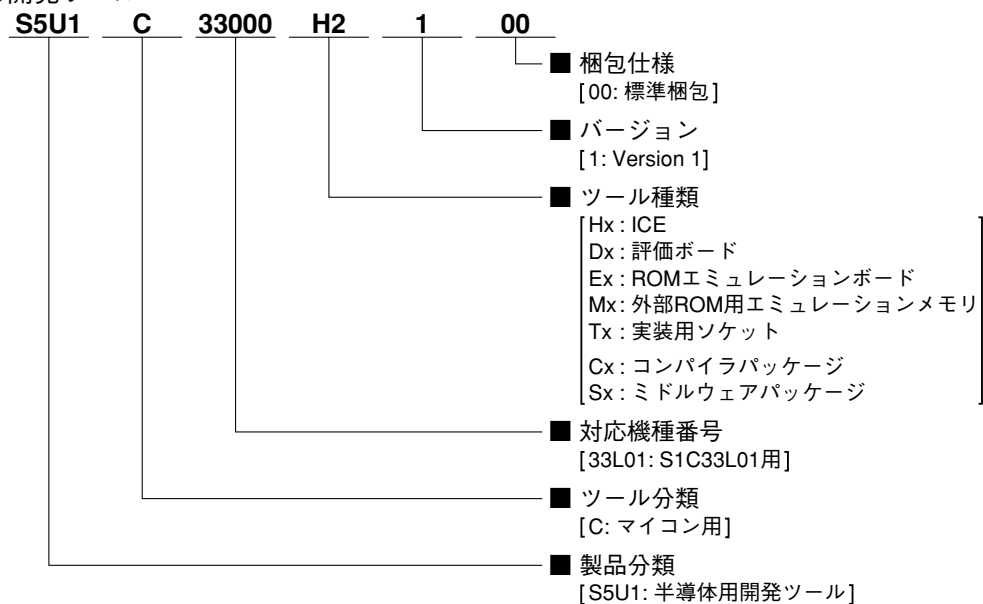
その他のブランド名または製品名は、それらの所有者の商標もしくは登録商標です。

製品型番体系

●デバイス



●開発ツール



はじめに

本書はS1C33 Familyの全機種に共通するソフトウェア開発ツール「S1C33 Family C/C++コンパイラパッケージ」によるC/C++ソースプログラムのコンパイルからデバッグ、最終的に提出していただくマスクデータの作成までの手順および各ツールの使用方法を説明します。

マニュアルの読み方

本書はプログラム開発に従事されている方を対象に編集されています。したがって、以下の内容が予備知識として必要です。

- C/C++言語(ANSI C準拠)に関する知識およびC/C++ソースプログラムの作成方法
- GNU C/C++, binutil, GNU makeおよびGNUリンカ(ld)用リンカスクリプトに関する知識
- アセンブリ言語に関する一般的な知識
- C/C++コンパイラおよびアセンブラを使用したプログラム開発全般の基礎知識
- Windows XPまたはWindows Vistaの基本的な操作方法

これらの内容につきましては、ANSI Cを解説した一般の書籍、GNUツール、Windowsのマニュアル等を参照してください。

■インストールの前に

1章をお読みください。パッケージの構成品や各ツールの概要が記載されています。

■インストール

2章に記載されたインストール手順に従って、ツールのインストールを行ってください。

■プログラム開発全体の流れと、その操作手順を理解するには

3章のチュートリアルをお読みください。コンパイルからデバッグ、最終的なマスクデータの作成までの概要を把握していただけます。

■ソースプログラム作成時

4章の必要箇所をお読みください。ソースファイル作成上の注意事項や、アセンブリソースファイルの内容等が記載されています。ソースファイルを作成する際には、以下のマニュアルも併せて参照してください。

S1C33xxxテクニカルマニュアル

デバイスの仕様、周辺回路の動作と制御方法について

S1C33000コアCPUマニュアル

コアCPUの機能と動作、命令セットの詳細について

■プログラムのデバッグについて

10章でデバッグの詳細を解説しています。10.1節～10.6節でデバッグ機能の概要を把握していただけます。各デバッグコマンドの詳細については10.7節を参照してください。

なお、デバッグ前に、使用するデバッグ用ツールのマニュアルも参照してください。

S1C33 Family In-Circuit Debugger Manual

インサーキットデバッグS5U1C3300xHの取り扱い方法について

S1C33 Family Debug Monitor Operation Manual

デバッグモニタS5U1C330M2Sの機能と実装方法について

■各ツールの詳細について

5～11の各章で、それぞれのツールの詳細を解説しています。必要に応じて参照してください。

マニュアルの表記

このマニュアルは、次の表記規則に従って記述されています。

■サンプル

サンプルの画面はすべてWindows XPでの表示例です。システムや搭載されているフォントにより表示が異なる場合があります。

■各部の名称

ウィンドウ名、メニューおよびメニューコマンド名、ボタン名、ダイアログボックス名、キーの名称は、[]で囲んで記述されています。たとえば、[Command]ウィンドウ、[File]メニュー、[Stop]ボタン、[q]キーなどのように記述します。

■命令などの名称

CPUの命令やコマンド名など、大文字、小文字の両方が許されるものについては小文字を使用しています。ただし、ユーザが指定するシンボル例などは除きます。これらの記述にはfixed-widthフォントを使用しています。

■数値の表記

数値は次のように記述しています。

10進数: 数値の前後に何も付けない。(例: 123、1000)

16進数: 数値の前に"0x"を付ける。(例: 0x0110、0xffff)

2進数: 数値の前に"0b"を付ける。(例: 0b0001、0b10)

ただし、表示例などには、16進数、2進数に何も付かない場合があります。

■マウス操作

クリック: 目的の位置にカーソル(ポインタ)を合わせ、マウスの左ボタンを一度押す操作をすべて「クリック」で表現します。右ボタンのクリック操作は「右クリック」という言葉を使います。

ダブルクリック: 目的の位置にカーソル(ポインタ)を合わせ、マウスの左ボタンを二度連続的に押す操作をすべて「ダブルクリック」で表現します。

ドラッグ: クリックにより選択したファイル(アイコン)などを、左ボタンを押したまま別の場所に移動させる操作を「ドラッグ」で表現します。

選択: メニューコマンドなどをクリックして選択する操作を、「選択」のみで表現します。

■キー操作

特定のキーを押す操作は「キーを入力」、または「キーを押す」と表現しています。

[Ctrl]+[C]キーのように+を使ったキーの組み合わせは、[Ctrl]キーを押しながら[C]キーを押す操作を表します。

キーボードからの入力例については特に[]で囲んでいません。

なお、このマニュアルでは、マウスで可能な操作は、すべてマウス操作方法のみを記述しています。キーボードによる操作についてはWindowsのマニュアルやヘルプ等を参考にしてください。

■コマンドや起動時オプション、メッセージの一般形

[]で囲まれた項目はユーザ選択項目で、入力しなくても動作することを表しています。

<>で囲まれた内容は、特定の名称が用いられることを表しています。たとえば、<file name>や<ファイル名>は実際のファイル名に置き換えることを示しています。

目 次

1 概要	1-1
1.1 特長	1-1
1.2 ツール構成	1-2
2 インストール	2-1
2.1 動作環境	2-1
2.2 インストール方法	2-2
3 ソフトウェア開発手順	3-1
3.1 ソフトウェア開発フロー	3-1
3.2 IDEによるソフトウェア開発	3-3
3.3 チュートリアル1(IDEとデバッグの基本操作)	3-7
3.3.1 IDEの起動	3-7
3.3.2 プロジェクトの作成	3-9
3.3.3 ソースファイルの作成、追加、編集	3-12
3.3.4 ビルドオプションとリンクスクリプトの編集	3-17
3.3.5 プログラムのビルド	3-25
3.3.6 デバッグ	3-26
3.3.7 ROMデータの作成	3-39
3.4 チュートリアル2(ユーザmakeファイルの使用)	3-40
3.4.1 プロジェクトの作成	3-40
3.4.2 ソースファイルのインポート	3-42
3.4.3 GNU33ファイルビルダの無効化	3-44
3.4.4 makeファイルの設定と修正	3-45
3.4.5 ビルド	3-47
3.4.6 デバッグ起動用ファイルのインポート	3-48
3.4.7 デバッグ起動用ファイルの修正	3-50
3.4.8 デバッグの起動	3-51
3.5 チュートリアル3(IDEプロジェクトのインポート)	3-53
3.6 デバッグ環境	3-57
3.6.1 デバッグモニタS5U1C330M2S	3-57
3.6.2 インサーキットデバッグS5U1C33000H	3-60
3.6.3 インサーキットデバッグS5U1C33001H	3-62
3.7 データエリア、セクション	3-64
3.7.1 データエリア	3-64
3.7.2 セクション	3-67
4 ソースファイル	4-1
4.1 ファイル形式とファイル名	4-1
4.2 C/C++ソースの文法	4-2
4.2.1 データ型	4-2
4.2.2 ライブラリ関数とヘッダファイル	4-3
4.2.3 インラインアセンブル	4-4
4.2.4 プロトタイプ宣言	4-4
4.2.5 <code>alloca()</code> /可変長配列	4-5
4.3 アセンブリソースの文法	4-6
4.3.1 ステートメント	4-6
4.3.2 オペランドの表記	4-10
4.3.3 拡張命令	4-14
4.3.4 プリプロセッサ擬似命令	4-15
4.4 ソース作成上の注意事項	4-16

5 GNU33 IDE	5-1
5.1 概要	5-1
5.1.1 特長	5-1
5.1.2 IDE使用上の注意	5-1
5.2 IDEの起動と終了	5-3
5.2.1 起動方法	5-3
5.2.2 終了方法	5-4
5.3 IDEウィンドウ	5-5
5.3.1 メニューバー	5-6
5.3.2 ウィンドウツールバー	5-14
5.3.3 エディタ領域	5-15
5.3.4 [C/C++ プロジェクト]ビュー	5-18
5.3.5 [ナビゲーター]ビュー	5-21
5.3.6 [アウトライン]ビュー	5-24
5.3.7 [コンソール]ビュー	5-25
5.3.8 [問題]ビュー	5-26
5.3.9 [プロパティ]ビュー	5-27
5.3.10 [Make ターゲット]ビュー	5-28
5.3.11 [検索]ビュー	5-29
5.3.12 [ブックマーク]ビュー	5-32
5.3.13 [タスク]ビュー	5-33
5.3.14 ビューの操作	5-34
5.3.15 パースペクティブ	5-37
5.4 プロジェクト	5-38
5.4.1 プロジェクトとは	5-38
5.4.2 新規プロジェクトの作成	5-38
5.4.3 プロジェクトのオープンとクローズ	5-41
5.4.4 ワークスペースの切り換え	5-42
5.4.5 既存プロジェクトのインポート	5-43
5.4.6 プロジェクトの削除	5-47
5.4.7 プロジェクトの名称変更	5-48
5.4.8 プロジェクト内のリソース操作	5-49
5.4.9 ファイルフィルタ	5-71
5.4.10 ワーキングセット	5-72
5.4.11 プロジェクトプロパティ	5-76
5.5 エディタとソースファイルの編集	5-78
5.5.1 エディタの起動	5-78
5.5.2 基本編集機能	5-80
5.5.3 C/C++ソース用編集機能	5-81
5.5.4 [アウトライン]ビュー	5-88
5.5.5 ナビゲーションヒストリ	5-89
5.5.6 ブックマーク	5-90
5.5.7 タスク	5-95
5.5.8 エディタのカスタマイズ	5-101
5.5.9 外部エディタを使用するには	5-103
5.5.10 外部エディタを行番号指定で起動するには	5-105
5.6 検索	5-107
5.6.1 テキスト検索	5-107
5.6.2 File検索	5-107
5.6.3 C/C++検索	5-109
5.6.4 コンテキストメニューによるC/C++検索	5-110
5.6.5 検索の中止	5-110
5.6.6 検索結果	5-111

5.7 プログラムのビルド.....	5-113
5.7.1 CPUタイプの設定.....	5-113
5.7.2 2パス/1パスメイクの選択.....	5-115
5.7.3 コンパイラオプションの設定.....	5-116
5.7.4 アセンブラオプションの設定.....	5-123
5.7.5 リンカオプションの設定.....	5-125
5.7.6 生成されるmakeファイル.....	5-129
5.7.7 リンカスクリプトの編集.....	5-133
5.7.8 ビルドの実行.....	5-162
5.7.9 クリーンとリビルド.....	5-164
5.7.10 ユーザ作成のmakeファイルを使用するには.....	5-166
5.8 デバッガの起動.....	5-168
5.8.1 パラメータファイルの生成.....	5-168
5.8.2 デバッガ起動コマンドの設定.....	5-173
5.8.3 デバッガの起動方法.....	5-176
5.9 IDEのカスタマイズ(設定).....	5-186
5.10 ダイアログ補足説明.....	5-217
5.10.1 プロジェクトのプロパティ.....	5-217
5.10.2 リソースの保管.....	5-248
5.10.3 新規 C++ クラス.....	5-249
5.10.4 ベース・クラスの選択.....	5-251
5.10.5 インポート>ファイル・システム.....	5-252
5.10.6 エクスポート>ファイル・システム.....	5-254
5.10.7 フィルター.....	5-256
5.11 IDEがプロジェクト内に作成するファイル.....	5-258
6 C/C++コンパイラ.....	6-1
6.1 機能.....	6-1
6.2 入出力ファイル.....	6-1
6.2.1 入力ファイル.....	6-1
6.2.2 出力ファイル.....	6-2
6.3 起動方法.....	6-2
6.3.1 起動フォーマット.....	6-2
6.3.2 コマンドラインオプション.....	6-2
6.4 コンパイラの出力.....	6-9
6.4.1 出力内容.....	6-9
6.4.2 データ表現.....	6-10
6.4.3 レジスタ使用法.....	6-12
6.4.4 関数呼び出し.....	6-14
6.4.5 スタックフレーム.....	6-15
6.4.6 C/C++ソースの文法.....	6-15
6.5 標準ヘッダファイル/標準ヘッダ.....	6-16
6.6 C++使用時のポイント.....	6-18
6.7 gccコアのバージョンアップおよび機能の追加/削除.....	6-20
6.8 Shift JISコードのフィルタ機能.....	6-21
6.9 Cとアセンブラの混在法.....	6-22
6.10 xgccの機能と注意事項について.....	6-27
6.11 既知の問題.....	6-28
7 ライブラリ.....	7-1
7.1 ライブラリ概要.....	7-1
7.1.1 ライブラリの構成.....	7-1
7.1.2 ライブラリ追加時の注意事項.....	7-2

7.2 エミュレーションライブラリ	7-3
7.2.1 概要	7-3
7.2.2 浮動小数点演算関数	7-4
7.2.3 整数剰余演算関数	7-6
7.3 long long型エミュレーションライブラリ	7-7
7.3.1 概要	7-7
7.3.2 long long型演算関数	7-7
7.4 ANSIライブラリ	7-8
7.4.1 概要	7-8
7.4.2 ANSIライブラリ関数一覧	7-8
7.4.3 グローバル変数の宣言と初期化	7-14
7.4.4 下位レベル関数	7-15
7.5 C++ライブラリ	7-17
7.5.1 概要	7-17
7.5.2 STLコンテナメンバ関数一覧	7-19
8 アセンブラ	8-1
8.1 機能	8-1
8.2 入出力ファイル	8-1
8.2.1 入力ファイル	8-1
8.2.2 出力ファイル	8-2
8.3 起動方法	8-3
8.3.1 起動フォーマット	8-3
8.3.2 コマンドラインオプション	8-3
8.4 スコープ	8-5
8.5 アセンブラ擬似命令	8-6
8.5.1 Textセクション定義擬似命令(.text)	8-6
8.5.2 Dataセクション定義擬似命令(.rodata, .data)	8-7
8.5.3 Bssセクション定義擬似命令(.bss)	8-8
8.5.4 Commセクション定義擬似命令(.comm)	8-9
8.5.5 Ctorsセクション定義擬似命令(.ctors)	8-10
8.5.6 Dtorsセクション定義擬似命令(.dtors)	8-11
8.5.7 Gcc_except_tableセクション定義擬似命令(.gcc_except_table)	8-12
8.5.8 データ定義擬似命令(.long, .short, .byte, .ascii, .space)	8-13
8.5.9 領域確保擬似命令(.zero)	8-14
8.5.10 アライメント擬似命令(.align)	8-15
8.5.11 グローバル宣言擬似命令(.global)	8-16
8.5.12 シンボル定義擬似命令(.set)	8-17
8.6 擬似オペランド	8-18
8.7 拡張命令	8-19
8.7.1 算術演算命令	8-19
8.7.2 比較命令	8-20
8.7.3 論理演算命令	8-21
8.7.4 シフト&ローテート命令	8-22
8.7.5 スタック〜レジスタ間データ転送命令	8-23
8.7.6 メモリ〜レジスタ間データ転送命令	8-24
8.7.7 即値ロード命令	8-26
8.7.8 ビット操作命令	8-27
8.7.9 分岐命令	8-28
8.7.10 C33 ADVコア用メモリ〜レジスタ間データ転送命令	8-31
8.7.11 C33 ADVコア用%dpエリア〜レジスタ間データ転送命令	8-32
8.8 拡張命令の最適化	8-33

8.9 エラー /ワーニングメッセージ.....	8-39
8.10 注意事項.....	8-40
9 リンカ.....	9-1
9.1 機能.....	9-1
9.2 入出力ファイル.....	9-1
9.2.1 入力ファイル.....	9-1
9.2.2 出力ファイル.....	9-2
9.3 起動方法.....	9-3
9.3.1 起動フォーマット.....	9-3
9.3.2 コマンドラインオプション.....	9-3
9.4 リンク処理.....	9-4
9.4.1 データエリアポインタの指定.....	9-4
9.4.2 デフォルトリンクスクリプト.....	9-4
9.4.3 リンク例.....	9-7
9.5 エラー /ワーニングメッセージ.....	9-11
9.6 注意事項.....	9-12
10 デバッガ.....	10-1
10.1 特長.....	10-1
10.2 入出力ファイル.....	10-1
10.2.1 入力ファイル.....	10-2
10.2.2 出力ファイル.....	10-3
10.3 起動方法.....	10-4
10.3.1 起動フォーマット.....	10-4
10.3.2 起動時オプション.....	10-4
10.3.3 終了方法.....	10-5
10.4 ウィンドウ.....	10-6
10.4.1 デバッグパースペクティブ.....	10-6
10.4.1.1 デバッグパースペクティブの切り替え.....	10-6
10.4.1.2 デバッグパースペクティブの構成.....	10-7
10.4.1.3 ビューを開く/閉じる.....	10-8
10.4.1.4 パースペクティブのカスタマイズ.....	10-9
10.4.1.5 メニュー /ツールバー.....	10-9
10.4.1.6 設定の変更.....	10-12
10.4.2 [デバッグ]ビュー.....	10-16
10.4.2.1 画面構成.....	10-16
10.4.2.2 メニュー /ツールバー.....	10-16
10.4.2.3 表示内容.....	10-18
10.4.2.4 操作.....	10-19
10.4.2.5 制限事項.....	10-22
10.4.3 [ソース]エディター.....	10-23
10.4.3.1 画面構成.....	10-23
10.4.3.2 メニュー /ツールバー.....	10-24
10.4.3.3 表示内容.....	10-25
10.4.3.4 操作.....	10-27
10.4.3.5 制限事項.....	10-30
10.4.4 [逆アセンブル]ビュー.....	10-31
10.4.4.1 画面構成.....	10-31
10.4.4.2 メニュー /ツールバー.....	10-31
10.4.4.3 表示内容.....	10-32
10.4.4.4 操作.....	10-32
10.4.4.5 制限事項.....	10-35
10.4.5 [ブレークポイント]ビュー.....	10-36
10.4.5.1 画面構成.....	10-36

10.4.5.2	メニュー / ツールバー	10-36
10.4.5.3	ブレークポイント全般の仕様	10-38
10.4.5.4	表示内容	10-42
10.4.5.5	操作	10-44
10.4.5.6	制限事項	10-48
10.4.6	[変数]ビュー	10-49
10.4.6.1	画面構成	10-49
10.4.6.2	メニュー / ツールバー	10-49
10.4.6.3	表示内容	10-51
10.4.6.4	操作	10-51
10.4.6.5	制限事項	10-52
10.4.7	[式]ビュー	10-53
10.4.7.1	画面構成	10-53
10.4.7.2	メニュー / ツールバー	10-53
10.4.7.3	表示内容	10-55
10.4.7.4	操作	10-56
10.4.7.5	制限事項	10-58
10.4.8	[レジスター]ビュー	10-59
10.4.8.1	画面構成	10-59
10.4.8.2	メニュー / ツールバー	10-59
10.4.8.3	表示内容	10-61
10.4.8.4	操作	10-63
10.4.8.5	制限事項	10-64
10.4.9	[メモリー]ビュー	10-65
10.4.9.1	画面構成	10-65
10.4.9.2	メニュー / ツールバー	10-65
10.4.9.3	表示内容	10-67
10.4.9.4	操作	10-68
10.4.9.5	制限事項	10-72
10.4.10	[コンソール]ビュー	10-73
10.4.10.1	画面構成	10-73
10.4.10.2	メニュー / ツールバー	10-73
10.4.10.3	表示内容	10-74
10.4.10.4	操作	10-74
10.4.10.5	制限事項	10-78
10.4.11	[シミュレーテッドI/O]ビュー	10-79
10.4.11.1	画面構成	10-79
10.4.11.2	メニュー / ツールバー	10-79
10.4.11.3	表示内容	10-79
10.4.11.4	操作	10-80
10.4.11.5	制限事項	10-80
10.4.12	[トレース]ビュー	10-81
10.4.12.1	画面構成	10-81
10.4.12.2	メニュー / ツールバー	10-81
10.4.12.3	表示内容	10-81
10.4.12.4	操作	10-82
10.4.12.5	制限事項	10-82
10.5	コマンド実行方法	10-83
10.5.1	コマンドのキーボード入力	10-83
10.5.2	パラメータの入力形式	10-84
10.5.3	メニュー / ツールバーによる実行	10-85
10.5.4	コマンドファイルによる実行	10-88
10.5.5	ログファイル	10-89
10.6	デバッグ機能	10-90
10.6.1	コネクトモード	10-90
10.6.2	ファイルの読み込み	10-94
10.6.3	メモリ、変数およびレジスタの操作	10-95

10.6.4	プログラムの実行	10-100
10.6.5	ブレーク機能	10-106
10.6.6	トレース機能	10-113
10.6.7	シミュレーテッドI/O	10-121
10.6.8	フラッシュメモリ操作	10-123
10.6.9	ビッグエンディアンへの対応について	10-126
10.6.10	プロファイラ・カバレッジ機能	10-127
10.7	コマンドリファレンス	10-137
10.7.1	コマンド一覧	10-137
10.7.2	コマンド詳細説明の見方	10-138
10.7.3	メモリ操作コマンド	10-139
	c33 fb (領域のフィル, バイト単位)	10-139
	c33 fh (領域のフィル, ハーフワード単位)	10-139
	c33 fw (領域のフィル, ワード単位)	10-139
	x (メモリダンプ)	10-141
	set {} (データ入力)	10-143
	c33 mvb (領域のコピー, バイト単位)	10-144
	c33 mvh (領域のコピー, ハーフワード単位)	10-144
	c33 mvw (領域のコピー, ワード単位)	10-144
	c33 df (メモリ内容の保存)	10-146
	c33 rm (ターゲットメモリリード)	10-148
	c33 readmd (メモリリードモード)	10-149
10.7.4	レジスタ操作コマンド	10-150
	info reg (レジスタ表示)	10-150
	set \$ (レジスタ変更)	10-153
10.7.5	プログラム実行コマンド	10-154
	continue (連続実行)	10-154
	until (テンポラリブレーク付き連続実行)	10-156
	step (ステップ実行、行単位)	10-158
	stepi (ステップ実行、ニーモニック単位)	10-158
	next (スキップ付きステップ実行、行単位)	10-160
	nexti (スキップ付きステップ実行、ニーモニック単位)	10-160
	finish (関数終了)	10-162
	c33 callmd (ユーザ関数コールモード設定)	10-163
	c33 call (ユーザ関数コール)	10-164
10.7.6	CPUリセットコマンド	10-166
	c33 rstc (コールドリセット)	10-166
	c33 rsth (ホットリセット)	10-168
	c33 rstt (ターゲットリセット)	10-169
10.7.7	割り込みコマンド	10-170
	c33 int (割り込み)	10-170
10.7.8	ブレーク設定コマンド	10-171
	break (ソフトウェアPCブレーク設定)	10-171
	tbreak (テンポラリソフトウェアPCブレーク設定)	10-171
	hbreak (ハードウェアPCブレーク設定)	10-174
	tbreak (テンポラリハードウェアPCブレーク設定)	10-174
	watch (データライトブレーク設定)	10-177
	rwatch (データリードブレーク設定)	10-177
	awatch (データリード/ライトブレーク設定)	10-177
	delete (ブレーク番号によるブレークの解除)	10-179
	clear (ブレーク位置によるブレークの解除)	10-180
	enable (ブレークポイントの有効化)	10-182
	disable (ブレークポイントの無効化)	10-182
	ignore (回数指定付きブレークの無効化)	10-184
	info breakpoints (ブレークポイントリストの表示)	10-185
	c33 oab (オンチップエリアブレークの設定)	10-186
	c33 obb (オンチップバスブレーク設定)	10-188

c33 timebrk (時間経過後ブレークの設定).....	10-191
10.7.9 シンボル情報表示コマンド.....	10-192
info locals (ローカルシンボル表示).....	10-192
info var (グローバルシンボル表示).....	10-192
print (シンボル値の変更).....	10-193
10.7.10 ファイル読み込みコマンド.....	10-194
file (デバッグ情報の読み込み).....	10-194
load (プログラムのロード).....	10-195
10.7.11 マップ情報コマンド.....	10-196
c33 rpf (マップ情報の設定).....	10-196
c33 map (マップ情報の表示).....	10-197
10.7.12 フラッシュメモリ操作コマンド.....	10-198
c33 fls (フラッシュメモリ設定).....	10-198
c33 fle (フラッシュメモリ消去).....	10-199
10.7.13 トレースコマンド.....	10-200
c33 tm (PCトレースモード設定).....	10-200
c33 td (PCトレース内容の表示).....	10-206
c33 autotd (PCトレース内容の自動表示).....	10-207
c33 ts (PCトレース内容の検索).....	10-209
c33 tf (PCトレース内容の保存).....	10-210
c33 autotf (PCトレース内容の自動保存).....	10-212
c33 obt (オンチップバストレース設定).....	10-214
c33 otd (オンチップバストレース内容の表示).....	10-216
c33 otf (オンチップバストレース内容の保存).....	10-218
10.7.14 シミュレーテッドI/Oコマンド.....	10-219
c33 stdin (データ入力シミュレーション).....	10-219
c33 stdout (データ出力シミュレーション).....	10-220
10.7.15 フラッシュライタコマンド.....	10-221
c33 fwe (プログラム/データの消去).....	10-221
c33 fwlp (プログラムのロード).....	10-222
c33 fwld (データのロード).....	10-223
c33 fwdc (ターゲットメモリのコピー).....	10-224
c33 fwd (フラッシュライタ情報の表示).....	10-225
10.7.16 プロファイラ・カバレッジコマンド.....	10-226
c33 profilemd (プログラム・カバレッジモードの設定).....	10-226
c33 profile (プロファイラウィンドウの起動).....	10-227
c33 coverage (カバレッジウィンドウの起動).....	10-228
10.7.17 その他のコマンド.....	10-229
c33 log (ロギング).....	10-229
source (コマンドファイルの実行).....	10-230
c33 clockmd (実行カウンタモード設定).....	10-232
c33 clock (実行カウンタ表示).....	10-232
target (ターゲットの接続).....	10-234
detach (ターゲットの切断).....	10-236
c33 das (デバッグユニットベースアドレス設定).....	10-237
c33 lpt (パラレルロードモード設定/解除).....	10-238
pwd (カレントディレクトリの表示).....	10-239
cd (カレントディレクトリの変更).....	10-239
c33 firmupdate (ファームウェア更新).....	10-240
c33 dclk (DCLKの分周値変更).....	10-241
c33 oscwait (OSC1モード時のウェイト設定).....	10-242
c33 cachehit (キャッシュヒット率の表示).....	10-243
c33 logiana (ロジックアナライザモード設定).....	10-244
c33 help (ヘルプ).....	10-245
quit (デバッグ終了).....	10-247
10.8 パラメータファイル.....	10-248
10.9 ステータス/エラーメッセージ.....	10-251

10.9.1 ステータスメッセージ	10-251
10.9.2 エラーメッセージ	10-251
11 その他のツール	11-1
11.1 make.exe	11-1
11.1.1 機能	11-1
11.1.2 入力ファイル	11-1
11.1.3 起動方法	11-2
11.1.4 makeファイル	11-3
11.1.5 マクロ定義と参照	11-6
11.1.6 依存リスト	11-7
11.1.7 サフィックス定義	11-10
11.1.8 clean	11-12
11.1.9 sh.exeによる起動	11-12
11.1.10 メッセージ	11-13
11.1.11 注意事項	11-13
11.2 ccap.exe	11-14
11.2.1 機能	11-14
11.2.2 出力ファイル	11-14
11.2.3 使用方法	11-14
11.2.4 エラーメッセージ	11-15
11.3 objdump.exe	11-16
11.3.1 機能	11-16
11.3.2 入力ファイル	11-16
11.3.3 使用方法	11-16
11.3.4 ダンプフォーマット	11-17
11.3.5 エラーメッセージ	11-20
11.3.6 注意事項	11-20
11.4 objcopy.exe	11-21
11.4.1 機能	11-21
11.4.2 入出力ファイル	11-21
11.4.3 使用方法	11-22
11.4.4 HEXファイルの作成方法	11-22
11.5 ar.exe	11-23
11.5.1 機能	11-23
11.5.2 入出力ファイル	11-23
11.5.3 使用方法	11-24
11.6 moto2ff.exe	11-26
11.6.1 機能	11-26
11.6.2 入出力ファイル	11-26
11.6.3 起動フォーマット	11-26
11.6.4 エラー /ワーニングメッセージ	11-27
11.6.5 マスクROMデータの作成方法	11-27
11.7 旧バージョンのデバugg	11-28

Quick Reference

1 概要

1.1 特長

S1C33 Family C/C++コンパイラパッケージには、C/C++ソースプログラムのコンパイル、アセンブリソースプログラムのアセンブルからデバッグ、さらにROM書き込みデータや提出用マスクデータの作成までを行う一連のソフトウェアツールやユーティリティが含まれています。

すべてのツール/ユーティリティはS1C33 Familyの全機種で共通に使用することができます。
主な特長を以下に示します。

- **強力なオプティマイズ機能**

C/C++コンパイラはS1C33アーキテクチャに対して最適化されており、非常にコンパクトなコードを生成します。また、高いオプティマイズ能力にもかかわらず、デバッグ情報が失われることが少なく、C/C++ソースレベルのデバッグを可能とします。

- **使いやすい拡張命令セットによりS1C33000命令セットをサポート**

S1C33000命令セットの即値拡張機能(ext命令)や複数の命令を1命令で記述可能な拡張命令を提供します。これにより、データサイズを特に意識せずにアセンブリソースを作成することができます。

- **シミュレータ機能を持つ、C/C++ソースおよびアセンブリソースレベルデバッグ**

デバッグはC/C++ソースレベルデバッグおよびアセンブリソースレベルデバッグに対応しています。インサーキットデバッグ(S5U1C33000H、S5U1C33001H)等を使用することにより、ターゲットボードを動作させた状態でデバッグが行えます。また、パソコンのみでもデバッグが行えるシミュレータモードを持っています。

- **Windowsに対応した統合開発環境**

Microsoft Windows XPおよびWindows Vistaに対応したGNU33 IDEは、ソース作成からデバッグまでのシームレスな統合開発環境を提供します。

1.2 ツール構成

本パッケージに含まれる主要なソフトウェアツールの概要を以下に示します。

(1) C/C++コンパイラ (xgcc.exe)

ANSI C/C++に準拠したC/C++コンパイラで、GNU C/C++コンパイラがベースとなっています。このツールは**cpp.exe**および**cc1.exe/cc1plus.exe**を連続して呼び出し、C/C++ソースファイルをコンパイルしてS1C33 Family用のアセンブリソースファイルを生成します。強力な最適化能力を持ち、非常にコンパクトなコードを生成します。**xgcc.exe**は、さらにアセンブラ**as.exe**を呼び出してオブジェクトファイルを生成することもできます。

(2) アセンブラ (as.exe)

C/C++コンパイラが出力するアセンブリソースファイルをアセンブルし、ソースファイルのニーモニックをS1C33000のオブジェクト(機械語)コードに変換します。**as.exe**は**xgcc.exe**から**cpp.exe**に続けて実行させることも可能です。これにより、アセンブリソースファイル内でプリプロセッサ擬似命令を使用することも許されます。結果はリンクおよびライブラリ化が可能なオブジェクトファイルとして出力されます。

(3) リンカ (ld.exe)

C/C++コンパイラおよびアセンブラによって生成したオブジェクトコードのメモリロケーションを決定し、実行可能なオブジェクトコードを生成します。複数のオブジェクトファイルやライブラリも、このツールによって1つにまとめられます。

(4) デバッグ (gdb.exe)

ハードウェアツール(S5U1C33001H、S5U1C33000H)やデバッグモニタ(S5U1C330M2S)を制御してソースレベルのデバッグを行うソフトウェアです。特別なツールを使用せずにパソコン上でデバッグを行うシミュレーション機能も持っています。

gdb.exeはWindows GUIに対応しています。ブレークやステップ実行など、頻繁に使用するコマンドはツールバーに登録されており、キーボード操作の量を抑えています。また、各種のデータもマルチウィンドウとして表示できるため、デバッグ作業が効率良く行えます。

(5) ライブラリアン (ar.exe)

ライブラリの編集ツールです。**ar.exe**はC/C++コンパイラおよびアセンブラが生成したオブジェクトファイルをライブラリに登録することができます。ライブラリからオブジェクトを削除したり、オブジェクトファイルに復元することもできます。

(6) Make (make.exe)

コンパイルからアセンブルまでを自動実行します。基本的な実行内容を記述したmakeファイルはIDEによって生成可能です。

(7) GNU33 IDE (eclipse.exe)

ソース作成からデバッグまでの統合開発環境を提供する開発用ワークベンチです。

本パッケージには、その他のGNUツール、サンプルプログラムおよびいくつかのユーティリティプログラムが含まれています。それらの内容につきましては、ディスク上の"readmeVxx.txt"をご覧ください。

注: 各ツールのコマンドオプションについては、各章で説明されているオプションのみ動作保証の対象となります。その他のオプションはお客様の責任においてご使用ください。

2 インストール

この章ではS1C33 Family C/C++コンパイラパッケージに収められたツールの動作環境とインストール方法について説明します。

2.1 動作環境

S1C33 Family C/C++コンパイラパッケージを使用するには、次の動作環境が必要です。

●パーソナルコンピュータ

IBM PC/ATまたは完全互換機が必要です。Pentium3 800MHz以上のCPUと512MB以上のRAMを搭載した機種を推奨します。

別売のインサーキットデバッグS5U1C33000Hや、S5U1C330M1D1ボードとデバッグモニタS5U1C330M2Sを使用するにはシリアルポート(D-sub 9pin)が必要です。別売のインサーキットデバッグS5U1C33001Hを使用する場合はUSBポートが必要となります。

●ディスプレイ

1,024 × 768ドット以上のディスプレイを推奨します。

注: Windowsの"画面のプロパティ"で特大のフォントおよびハイコントラストを設定すると、IDEの画面が正しく表示されないことがあります。

●ハードディスク

S1C33 Family C/C++コンパイラパッケージをインストールするには、ハードディスクに約500MB以上の空き領域が必要です。

●マウス

ツールの操作にマウスが必要です。

●デバッグツール

ターゲットシステムも含めデバッグを行うには、本パッケージとは別売のインサーキットデバッグ(S5U1C33000HまたはS5U1C33001H)、あるいはデバッグモニタ(S5U1C330M2SとS5U1C330M1D1)等が必要です。

●システムソフトウェアについて

Microsoft Windows XPまたはWindows Vista(日本語版と英語版)に対応しています。

注: 64-bit OS には対応していません。

●ユーザーアカウント

管理者権限(Administrator権限)で実行してください。

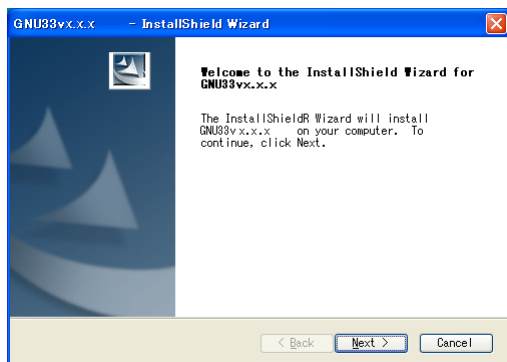
●その他

- ・ ディスク上の"readmeVxx.txt"(英語/日本語、xxはバージョン番号を表します)に注意点や制限事項が載っていますので、必ずお読みください。
- ・ 本パッケージのツールの実行にはcygwin1.dllが必要になります。このcygwin1.dllは¥gnu33フォルダに置かれていますが、お客様のシステム内にすでにcygwin1.dllがインストールされている場合、混在により問題が発生する場合があります。その場合は、お客様のシステム内にあるcygwin1.dllを削除、もしくは環境変数PATHの設定から外すことにより、¥gnu33フォルダのcygwin1.dllが確実に参照されるようにしてください。

2.2 インストール方法

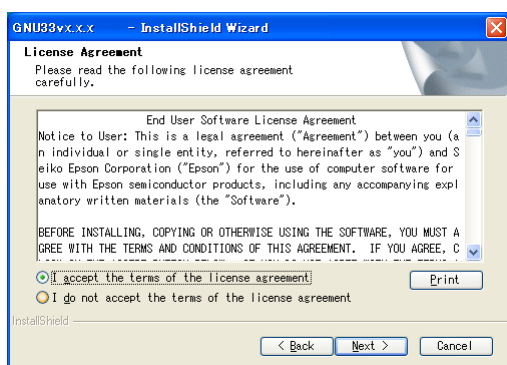
●ツールのインストール

- (1) Windows XP/Vistaを起動します。
起動している場合は、開いているプログラムをすべて終了させてください。
- (2) S5U1C33001Cの圧縮ファイルをSEIKO EPSONのユーザサイトよりダウンロードして、任意のフォルダに解凍してください。
- (3) **Setup.exe**をダブルクリックしてインストーラを起動します。



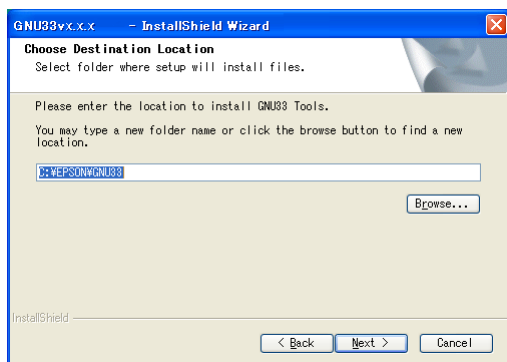
インストールウィザードのスタート画面が表示されます。

- (4) [Next >]ボタンをクリックして次に進めてください。



エンドユーザソフトウェアライセンス契約画面が表示されます。
表示されたライセンス契約内容は必ずお読みください。

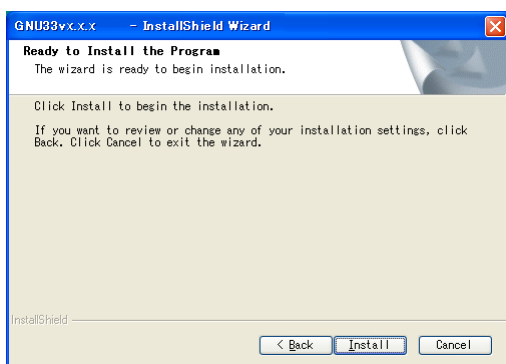
- (5) ライセンス契約に同意される場合は"I accept the terms of the license agreement"を選択して、[Next >]ボタンをクリックしてください。
同意できない場合は[Cancel]ボタンをクリックしてインストーラを終了させてください。



gnu33ツールをインストールするフォルダを指定する画面が表示されます。

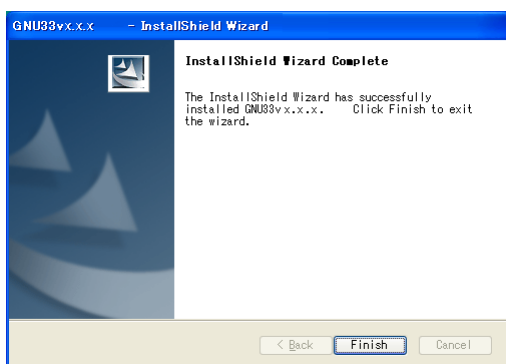
- (6) 表示されたインストール先を確認します。
インストール先を変更する場合は、[Browse...]ボタンでフォルダ選択ダイアログボックスを表示させます。インストールするフォルダをダイアログボックス上のリストから選択するか、[Path]テキストボックスにフォルダへのパスを入力して[OK]ボタンをクリックしてください。

- (7) [Next >]ボタンをクリックします。



インストールのスタート画面が表示されます。

(8) [Install]ボタンをクリックしてインストールを開始してください。

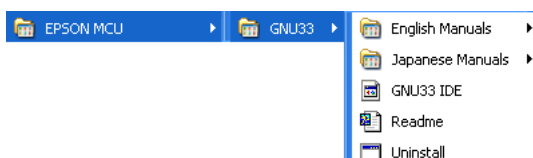


インストールが完了するとコンプリート画面が表示されます。

(9) [Finish]ボタンをクリックしてインストーラを終了させてください。

以上でツールのインストールは完了です。

インストールが正常に終了すると、Windowsのスタートアップメニューには[EPSON MCU] > [GNU33]メニューが追加されます。



2 インストール

●インストールされるファイル

コピー後のディレクトリとファイルの構成を以下に示します。

¥EPSON(デフォルト)

¥gnu33(gnu33ツールのルートDIR)

readmeVxx.txt

Copying.GNU

xgcc.exe, cpp.exe, cc1.exe/cc1plus.exe

xgcc_filt.exe

as.exe

as_org.exe

ld.exe

ar.exe

gdb.exe

make.exe

objdump.exe

objcopy.exe

moto2ff.exe

rm.exe

sed.exe

cp.exe

sh.exe

ccap.exe

gdbtk.ini

resetcold.gdb

resethot.gdb

userdefine.gdb

savebreak.gdb

loadbreak.gdb

gnuEdit.gdb

gnuCvrg.exe

gnuProf.exe

kill.exe

c33_cmd_ref_eng.chm

c33_cmd_ref_jpn.chm

cygtcl30.dll, cygtcl80.dll,

cygtk80.dll, tix4180.dll

cygwin1.dll

cygiconv-2.dll

cygintl-3.dll

cygintl-8.dll

¥eclipse

eclipse.exe

eclipse.ini

.eclipseproduct

epl-v10.html

notice.html

gnu33_32_trans.ico

artifacts.xml

¥configuration

¥cpuinfo

¥dropins

¥features

¥jre

ツールの内容説明(英語と日本語)

xxはバージョン番号を表しています。

GNUコピーライト

C/C++コンパイラ

漢字フィルター(xgcc.exeと同一のもの)

アセンブラフィルタ

アセンブラ本体

リンク

ライブラリアン

デバッグ

make

オブジェクトファイル情報表示ユーティリティ

オブジェクトファイル形式変換ユーティリティ

マスクROMファイル生成(ff埋め込み)ツール

ファイル削除ユーティリティ

ストリームエディタ

ファイルコピーユーティリティ

Bシェルユーティリティ

コンソールキャプチャユーティリティ

デバッグ設定ファイル

デバッグ[コールドリセット]ボタン用コマンドファイル

デバッグ[ホットリセット]ボタン用コマンドファイル

デバッグ[ユーザコマンド]ボタン用コマンドファイル

ブレークポイントを保存するコマンドファイル

ブレークポイントを再設定するコマンドファイル

外部エディタ名及び

パラメータを保存するコマンドファイル

カバレッジ実行ファイル

プロファイル実行ファイル

強制ブレーク

旧GDBコマンドリファレンス(英語版)

旧GDBコマンドリファレンス(日本語版)

デバッグ用dllファイル

開発ツール用dllファイル

cygwin1.dll用のdll

cygwin1.dll用のdll

cygwin1.dll用のdll

GNU33 IDEの実行ファイル

Eclipse設定ファイル

Eclipseのバージョン

EPLライセンス

ソフトウェア合意書

gnu33アイコンファイル

Eclipseアップデートマネージャ用ファイル

起動構成ファイル等

ウィザード用CPUファイルフォルダ

拡張プラグインフォルダ(空フォルダ)

フィーチャー

Java仮想マシン

¥plugins	プラグイン
¥readme	リリースノート
¥lib	ライブラリファイル
¥std	C33 STDコア用ライブラリ
libc.a	ANSI Cライブラリ
libgcc.a	エミュレーションライブラリ
libgccP.a	高精度エミュレーションライブラリ
libgcc2.a	long longエミュレーションライブラリ
libstdc++.a	C++ライブラリ
libstdiao.a	シミュレーテッドI/O用ライブラリファイル
¥pe	C33 PEコア用ライブラリ(構成はstdと同様)
¥33401	C33 ADVコア用ライブラリ(構成はstdと同様)
¥include	C/C++インクルードファイル
¥sample_ide	GNU33 IDE用サンプルファイル
¥utility	ユーティリティ
¥sample_std	C33 STDコア用サンプルファイル
¥sample_pe	C33 PEコア用サンプルファイル
¥sample_adv	C33 ADVコア用サンプルファイル
¥sample_c++	C++サンプルファイル
¥doc	マニュアル等ドキュメント
¥tool(ミドルウェア用フォルダ)	
¥tps33g	Toppersフォルダ(チュートリアルで使用)

sample_ideおよびutilityディレクトリの内容については、"readmeVxx.txt"を参照してください。

●旧サンプルプログラム

旧サンプルプログラム(¥sample_std、¥sample_pe、¥sample_adv、¥sample_c++)は¥utilityディレクトリに移動されました。サンプルを使用される場合は、これらのフォルダを¥gnu33ディレクトリに移動してください。

●旧バージョンに上書きインストールした場合の注意

古いバージョンに上書きインストールした場合、まれにGNU33 IDEの変更内容が反映されないことがあります。この場合には、GNU33 IDEを一旦終了し、コマンドプロンプトから以下のように起動してください。

```
C:¥EPSON¥gnu33¥eclipse>eclipse.exe -clean
```

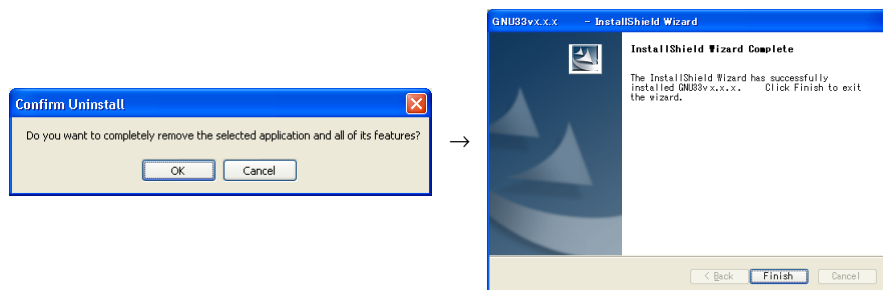
●OS設定上の注意

- 画面表示は、"画面のプロパティ"で"通常"のフォントサイズに設定してください。
- ネットワーク上のドライブをツールまたは作業用として使用する場合は、必ずドライブ名を割り当ててください。ネットワーク名を使用することはできません。
- デバッグツール(S5U1C33000H、S5U1C33001H、S5U1C330M2S)との通信に使用するCOMポートまたはUSBポートは、他のドライバやアプリケーションで使用しないでください。また、ノートPCではCOMポートとUSBポートがディセーブル状態になっている場合がありますので、デバッグツールを使用する前に使用できる状態になっていることを確認してください。
- デバッガgdbやGNU33 IDEにおいて、正常な表示ができないなど、GUI上の問題がおきる場合はグラフィックカードのアクセラレート機能のレベルを下げるか、Windowsに標準添付されている機能の低いドライバを使用してください。

2 インストール

●ツールのアンインストール

ツールをアンインストールするには、Windowsのスタートアップメニューで[EPSON MCU]>[GNU33]から[UnInstall]を選択し、表示されるダイアログボックスで[OK]ボタンをクリックしてください。



アンインストールは、コントロールパネルの[アプリケーションの追加と削除]でも行えます。

- 注: • ¥EPSON¥gnu33¥eclipse¥workspaceディレクトリをワークスペースに設定した場合は、その中にプロジェクトのデータが保存されていますので、ツールを削除する前にworkspaceフォルダをバックアップしてください。
- アンインストールしても、Windowsのスタートアップメニューで[EPSON MCU]>[GNU33]フォルダが削除されない場合があります。その場合は手動で削除してください。

●ライセンスについて

GNU

本パッケージのC/C++コンパイラツールはFree Software Foundation, Inc.のGNU C/C++コンパイラがベースとなっています。ライセンスに関してはテキストファイル"Copying.GNU"に記述されていますので、ご使用前にご覧ください。

EPL

GNU33 IDEはOpen Source InitiativeのEPL(Eclipse Public License)1.0に準拠しています。EPLに関しては、ご使用前に¥gnu33¥eclipseフォルダ内のepl-v10.htmlをご覧ください。

3 ソフトウェア開発手順

3.1 ソフトウェア開発フロー

ソフトウェア開発の流れを図3.1.1に示します。

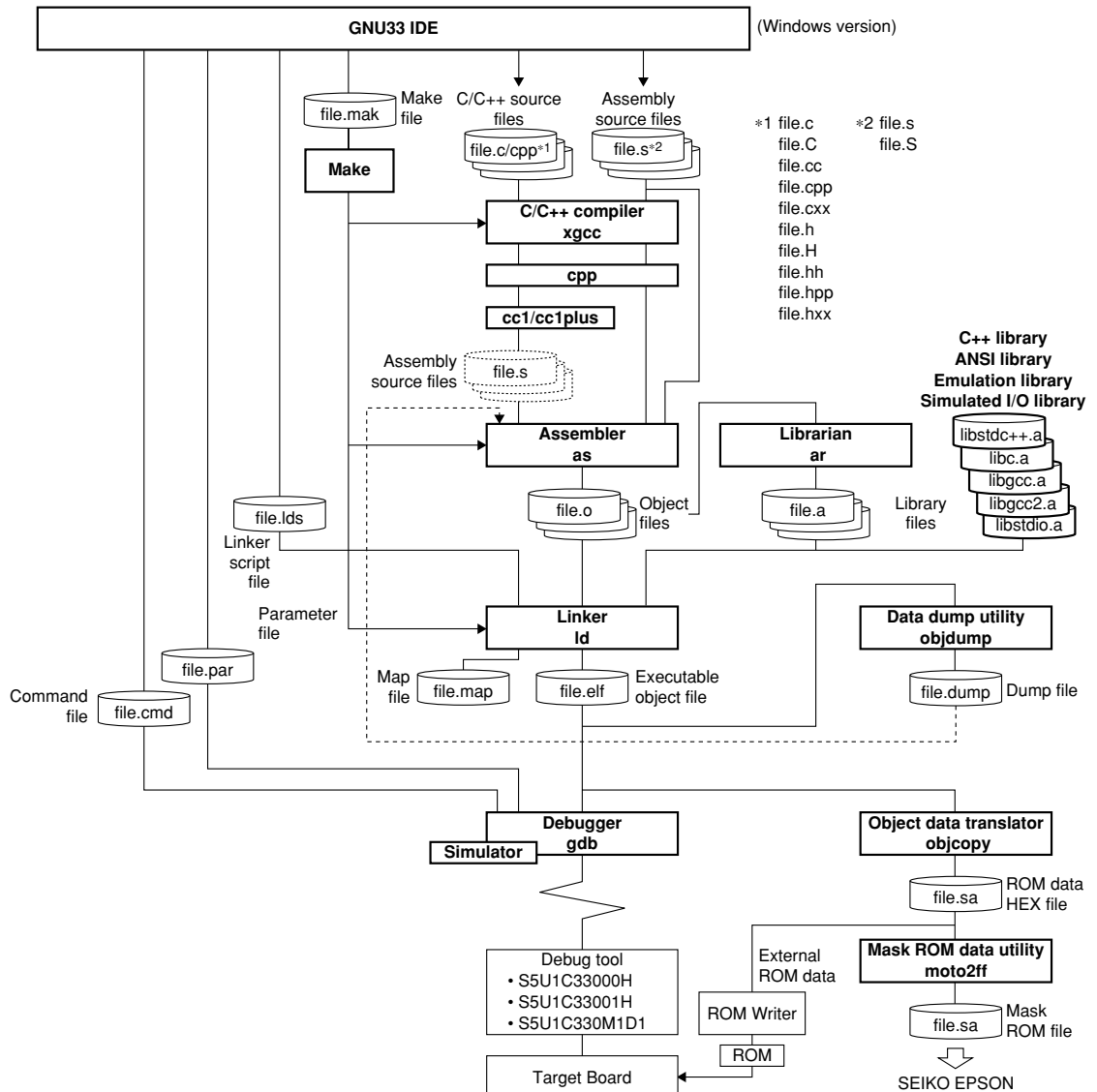


図3.1.1 ソフトウェア開発フロー

3 ソフトウェア開発手順

図に示したとおり、本パッケージのツールはソースプログラム作成後のすべてのソフトウェア処理を受け持っています。デバッグを除く基本的な操作はすべて**GNU33 IDE**(以下**IDE**)上で行います。開発の流れは次のようになります。

(1) プロジェクトの作成

IDEを使用し、新規プロジェクトを作成します。開発するアプリケーションのソフトウェアリソースを一元的に管理するためのプロジェクトファイルとリソースを置くワークスペースディレクトリが設定されます。

(2) ソースプログラムの作成

IDEのエディタまたは汎用のエディタ等を使用してソースファイルを作成し、プロジェクトに追加します。

(3) プログラムのビルド

まず、**IDE**を使用し、C/C++コンパイラからリンカまでの起動オプションやリンカスクリプトを設定します。

その後、**IDE**からビルドを実行すると、設定した内容に従って生成された**make**ファイルを使用して**make.exe**が実行され、デバッグ可能なelf形式のオブジェクトファイルが生成されます。

makeファイルにより下記の中で必要な処理が順次自動的に実行されます。

- **コンパイル(CまたはC++ソースの場合)**

ソースファイルをC/C++コンパイラ**xgcc**でコンパイルし、リンカ**ld**に inputsするオブジェクトファイル(.o)を生成します。

- **アセンブル(アセンブラソースの場合)**

アセンブラソースファイルをアセンブラ**as**によってアセンブルし、リンカ**ld**に inputsするオブジェクトファイル(.o)を生成します。

ソースファイル内でプリプロセッサ命令を使用している場合は、**xgcc**で前処理とアセンブルを行ってください。**xgcc**はオプション指定により、プリプロセッサ**cpp**とアセンブラ**as**を実行します。

- **リンク**

コンパイルとアセンブルにより、1個または複数のオブジェクトファイルが揃います。これらのファイルを、複数のものは1つにまとめ、ROM上に配置して実行可能なオブジェクトファイルとして生成するツールがリンカ**ld**です。

リンカ**ld**は、デバッグに必要な情報などを含んだelf形式のオブジェクトファイルを出力します。

(4) デバッグ

リンカ**ld**が生成したelf形式のオブジェクトファイルを使用して、動作の確認とデバッグをデバッガ**gdb**で行います。S5U1C33000H、S5U1C33001H、デバッグモニタ等を使用することにより、ハードウェアの動作まで含めたデバッグが行えますが、パソコン上でS1C33000コアCPUとメモリモデルとしての動作をシミュレートするシミュレータモードも備わっています。

デバッグ用の設定と起動は**IDE**から行えます。

(5) ROMデータ/マスクデータの作成

オブジェクトファイル形式変換ユーティリティ **objcopy**を使用して、リンカ**ld**が生成したelf形式のオブジェクトファイルから、外部ROMおよび内部ROM書き込み用のHEXファイルを作成します。最終的に、内部ROM用のHEXファイルを**moto2ff**によりマスクデータファイルに変換し、セイコーエプソンに提出していただきます。

また、これらのツール以外にライブラリアン**ar**が用意されています。汎用的な処理を行うモジュール(アセンブラ**as**が出力したオブジェクトファイル)をライブラリとしてまとめておくことができますので、今後のS1C33 Familyを使用したアプリケーションの開発に有効です。

3.2 IDEによるソフトウェア開発

ここでは、**IDE**によるソフトウェア開発手順を、いくつかのケースに分けて見ていきます。実際の操作方法については、別途チュートリアル1の節を用意しています。

まず、**IDE**でソフトウェア開発を行う場合、アプリケーションごとにプロジェクトというフォルダを作成し、そこで必要なリソースを管理します。

IDEでプロジェクトが作成されていない場合、**IDE**によるソフトウェア開発はプロジェクトの作成から始まります。これは、まったく新しいアプリケーションを作成する場合も、旧バージョンで作成したプログラムを利用する場合も同じです。

すでに**IDE**でプロジェクトが作成されている場合、そのプロジェクトフォルダをインポートすることで、他の環境からの移行あるいはバージョンアップの作業などを行うことができます。

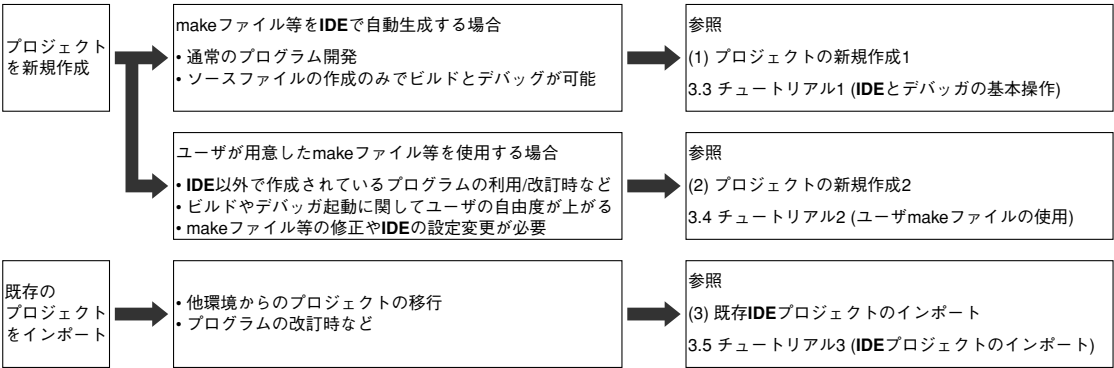


図3.2.1 IDEによる開発

(1)プロジェクトの新規作成1

IDEの通常のソフトウェア開発手順です。ユーザはソースファイルを作成するのみで、他のビルドやデバッグの起動に必要なファイルはIDEが自動生成します。

基本的な操作の流れは次のとおりです。

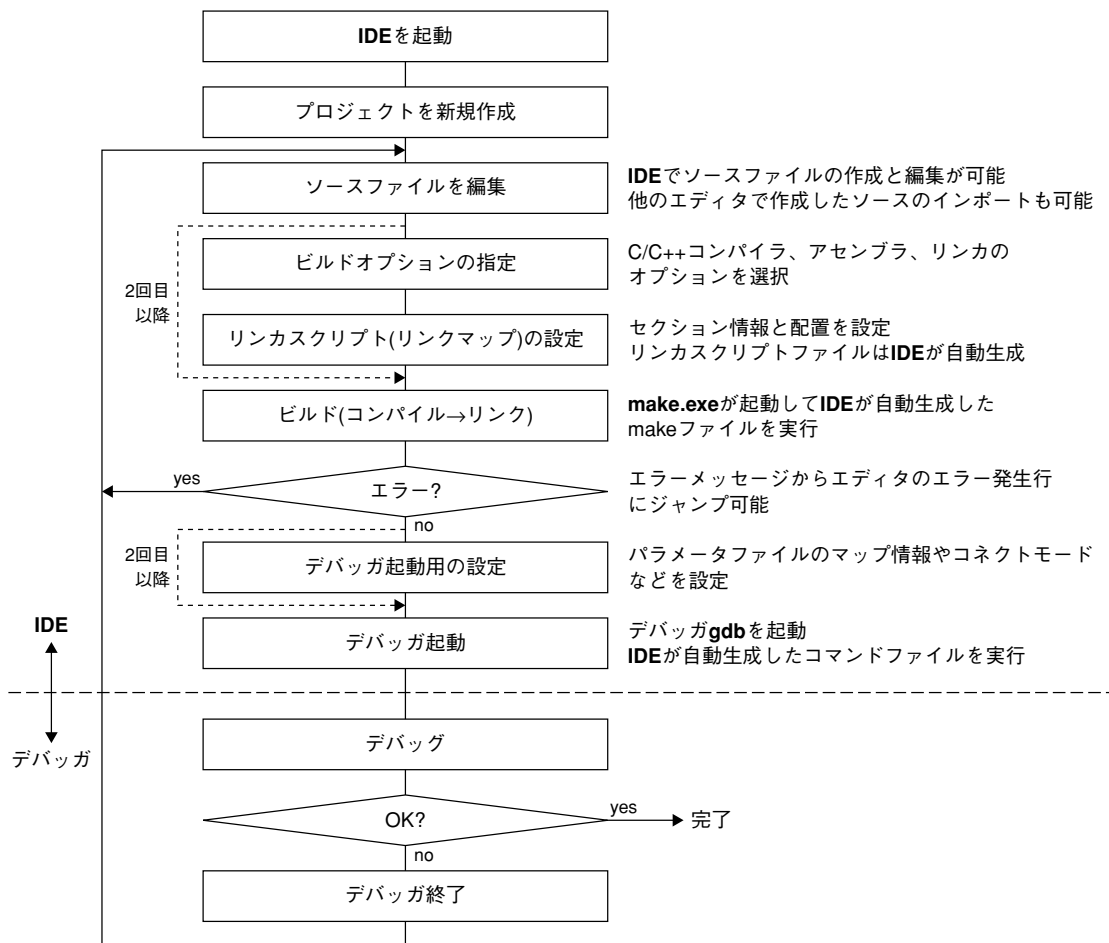


図3.2.2 操作の流れ(makeファイル等をIDEで自動生成)

IDEの起動からデバッグまでの基本操作を"3.3 チュートリアル1(IDEとデバッグの基本操作)"で説明していますので、参考にしてください。

(2)プロジェクトの新規作成2

旧gnu33ツールで作成したプログラムの改訂や、そのリソースを利用して新たなソフトウェア開発を行う際に、そのとき作成したmakeファイルやデバッグコマンドファイルなどを使用する必要があります。IDEによる自動生成に変えてそれらを使用することができます。ソフトウェアを新規開発する際にユーザ独自のmakeファイルやデバッグコマンドファイルを使用したい場合も同様です。基本的な操作の流れは次のとおりです。

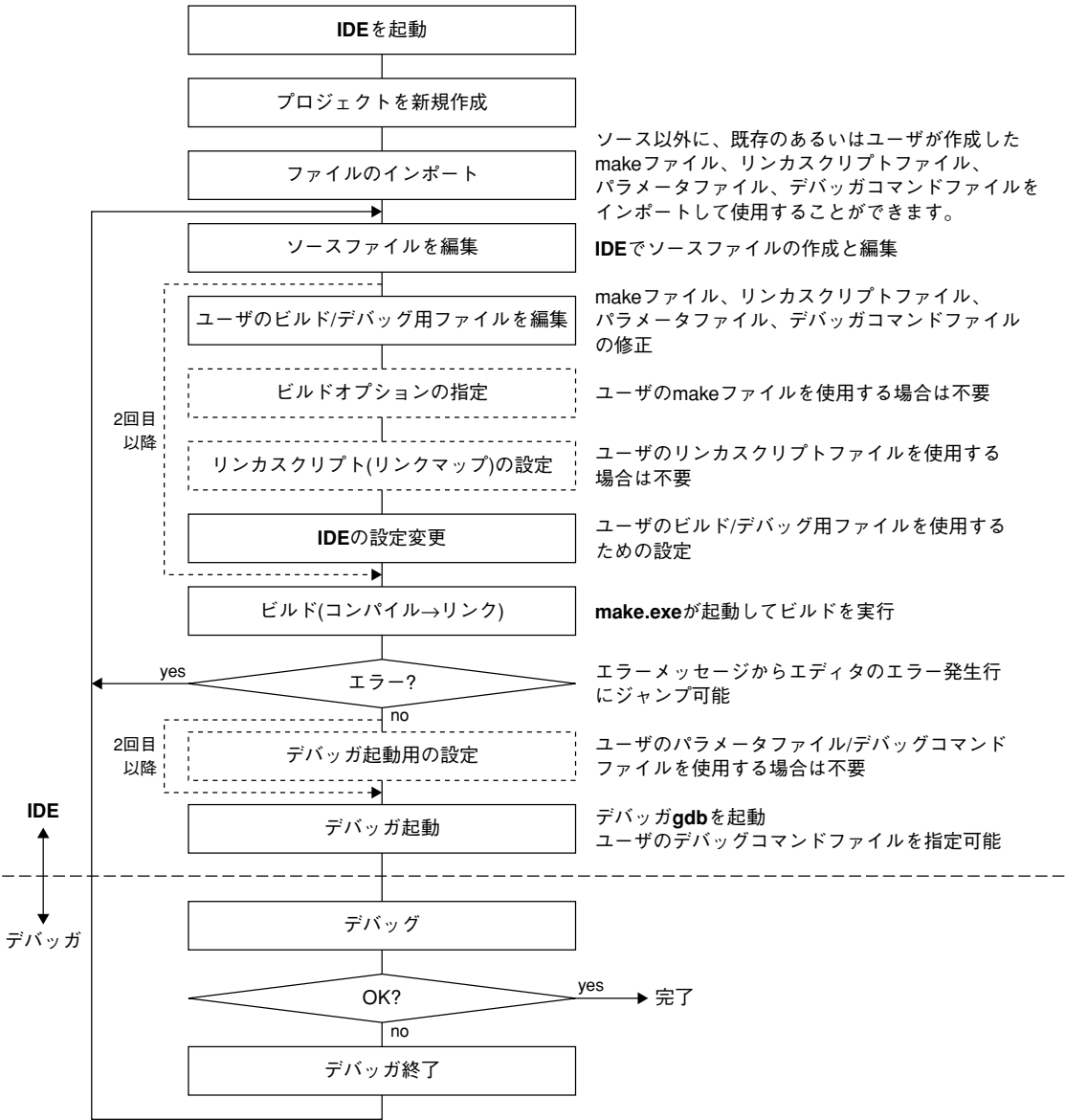


図3.2.3 操作の流れ(ユーザのmakeファイル等を使用)

"3.4 チュートリアル2(ユーザmakeファイルの使用)"では、旧gnu33ツールで作成されたサンプルプログラムをそのときのmakeファイルを利用してIDEでビルドする手順を説明していますので、参考にしてください。

既存makeファイルを使用する場合、makeファイル自体の修正やIDE上の設定変更が必要となりますので、特に問題がない場合はIDEで自動生成されるファイルの使用を推奨します。

(3)既存IDEプロジェクトのインポート

すでに**IDE**でプロジェクトが作成されている場合は、プロジェクトフォルダを**IDE**でインポートするだけで、作業の継続あるいは改訂作業などが行えます。プロジェクトのプロパティは継承されますので、特に変更する必要がなければ再設定などの操作は不要です。ただし、インポートするプロジェクトフォルダ内にプロジェクト管理ファイルが残っている必要があります。

基本的な操作の流れは次のとおりです。

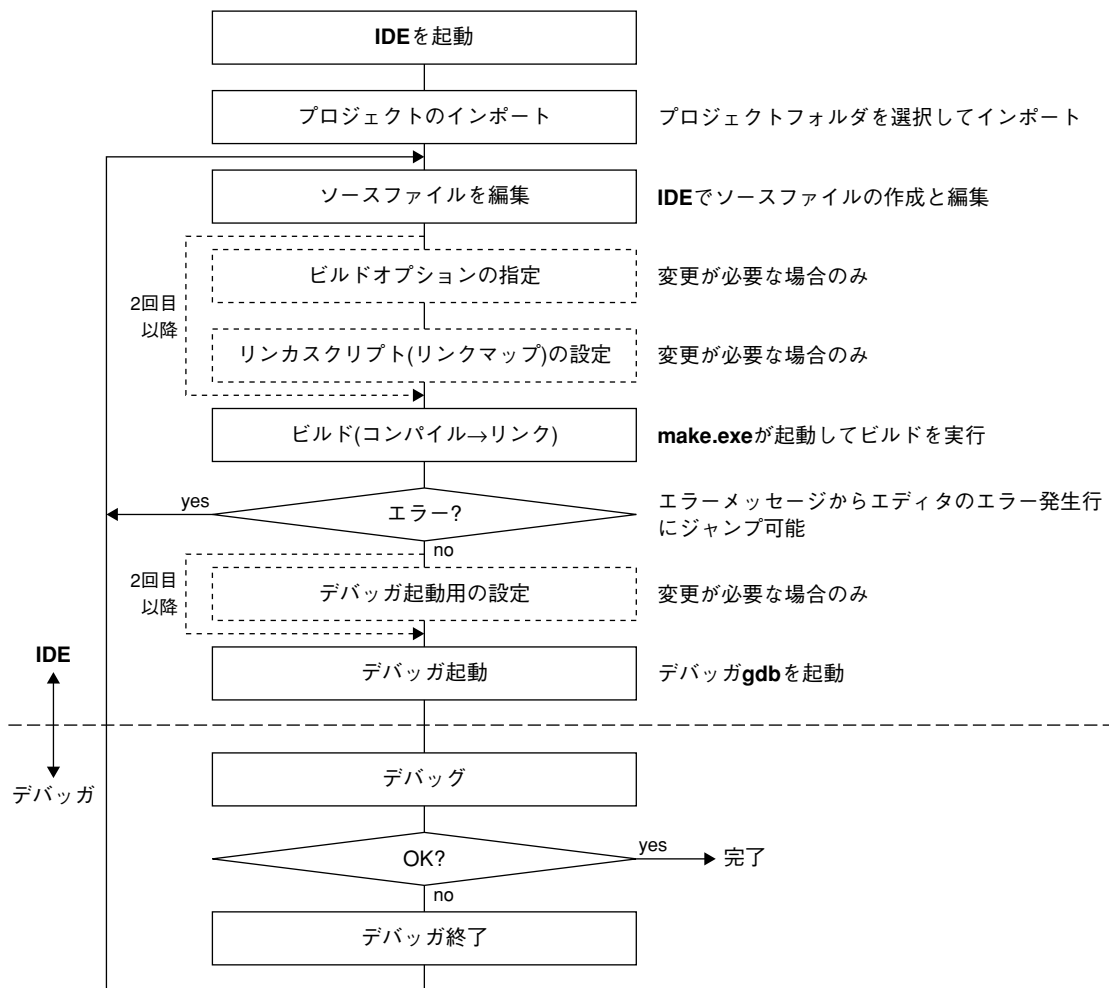


図3.2.4 操作の流れ(IDEプロジェクトのインポート)

"3.5 チュートリアル3(IDEプロジェクトのインポート)"では、**IDE**で作成されたサンプルプログラムをインポートする手順を説明していますので、参考にしてください。

3.3 チュートリアル1 (IDEとデバグの基本操作)

ここではチュートリアルとして**IDE**による操作の流れを見ていきます。各ツールの詳細については、それぞれのツールの章を参照してください。

使用するファイル

説明は `c:\¥EPSON¥gnu33¥sample_ide¥std¥simulator¥tst` ディレクトリに以下のサンプルソースファイルが存在するものとして行います。

`boot.s` アセンブラソースファイル
`main.c` Cソースファイル

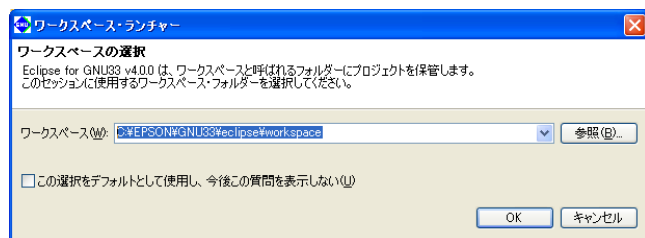
以下、上記の2つのソースファイルを使用して、プロジェクトの作成からプログラムのビルド、デバグで動作を確認するまでの操作方法を説明します。なお、ツールをインストール後、初めて使用するものとして説明が書かれていますので、すでに、何らかの操作を行っている場合には、画面表示例などが異なっていることがあります。

3.3.1 IDEの起動



操作1: `c:\¥EPSON¥gnu33¥eclipse` ディレクトリ内の **eclipse.exe** のアイコンをダブルクリックし、**IDE** を起動させます。Windows のスタートアップメニューから [EPSON MCU] > [GNU33] > [GNU33 IDE] を選択しても起動します。

Eclipse のスプラッシュに続き、[ワークスペース・ランチャー] ダイアログボックスが表示されます。ここではプロジェクトのリソースや出力ファイルを格納するワークスペース(ディレクトリ)を指定します。



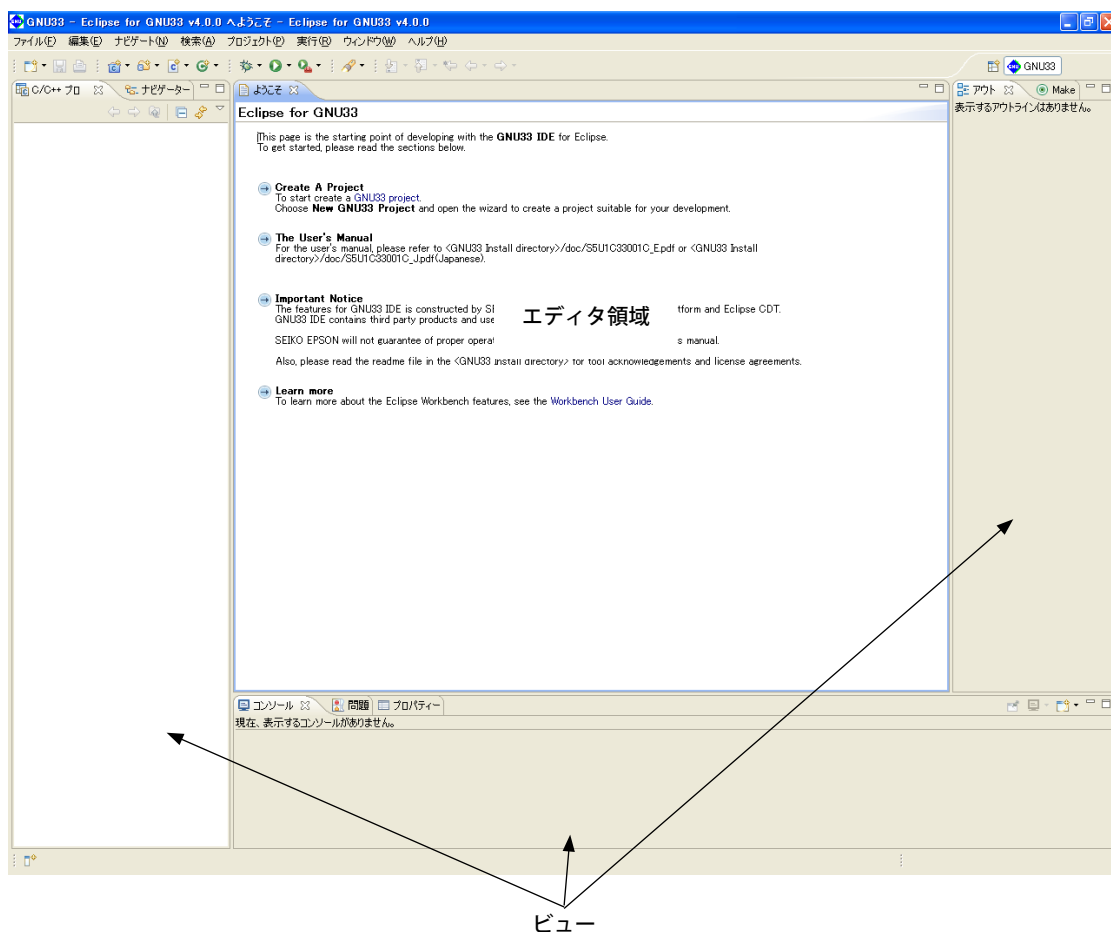
任意のディレクトリを選択または新規作成してワークスペースに設定可能ですが、このチュートリアルでは表示されたデフォルトのワークスペースディレクトリを使用します。

※ ワークスペースディレクトリには、プロジェクトディレクトリ(.projectファイルが含まれるディレクトリ)を指定しないでください。プロジェクトインポート(プロジェクトをワークスペースにコピーがONのとき)に失敗することがあります。

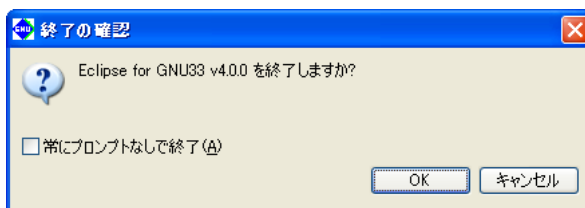
操作2: [OK] ボタンをクリックします。

IDE ウィンドウが表示されます。

3 ソフトウェア開発手順



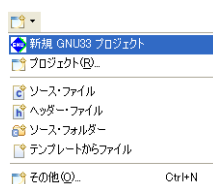
チュートリアルを途中で終了させるには、**IDE**の[ファイル]メニューから[終了]を選択してください。ウィンドウの✕(閉じる)ボタンでも終了可能です。ただし、この場合は次のダイアログボックスが表示されますので、終了する場合は[OK]ボタンを、終了を取り消す場合は[キャンセル]ボタンをクリックします。



3.3.2 プロジェクトの作成

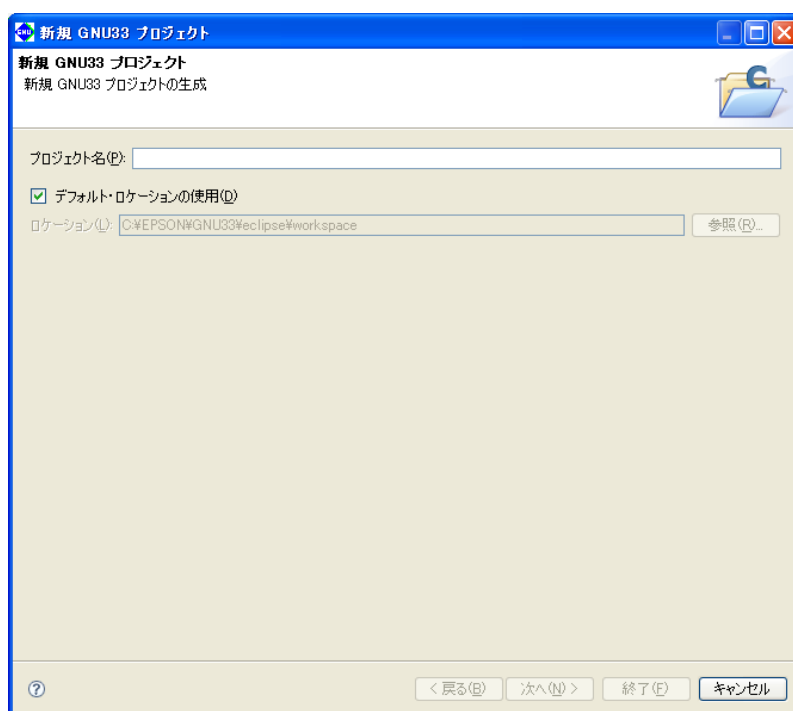
アプリケーション開発においては、複数のソースファイルからひとつの実行形式のプログラムファイルを作成します。これらのファイルなどを一元的に管理するためにひとつのプロジェクトを作成します。**IDE**では、このプロジェクト単位にプログラムを生成します。プロジェクトは、開発するアプリケーションプログラム自体ともいえますが、実際に作成されるプロジェクトは、指定したプロジェクト名を持つフォルダです。その中にプロジェクトの情報を含むファイル(.cproject、.gnu33project、.project)が生成されます。

●プロジェクトを新規作成するには

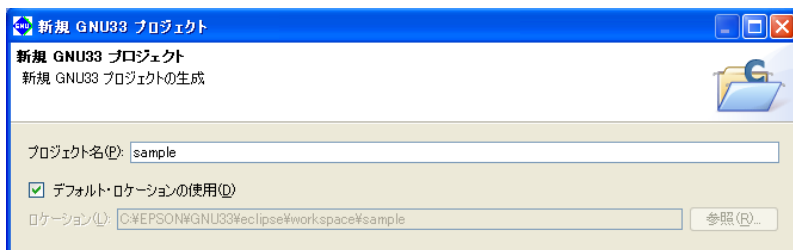


操作3: ツールバーの[新規]プルダウンメニューから[新規 GNU33 プロジェクト]を選択します。

このほかに、[ファイル]メニューまたは[C/C++ プロジェクト][ナビゲーター]ビュー内のコンテキストメニュー（右クリックにより表示）の[新規]からも[新規 GNU33 プロジェクト]が選択できます。



●プロジェクト名の指定

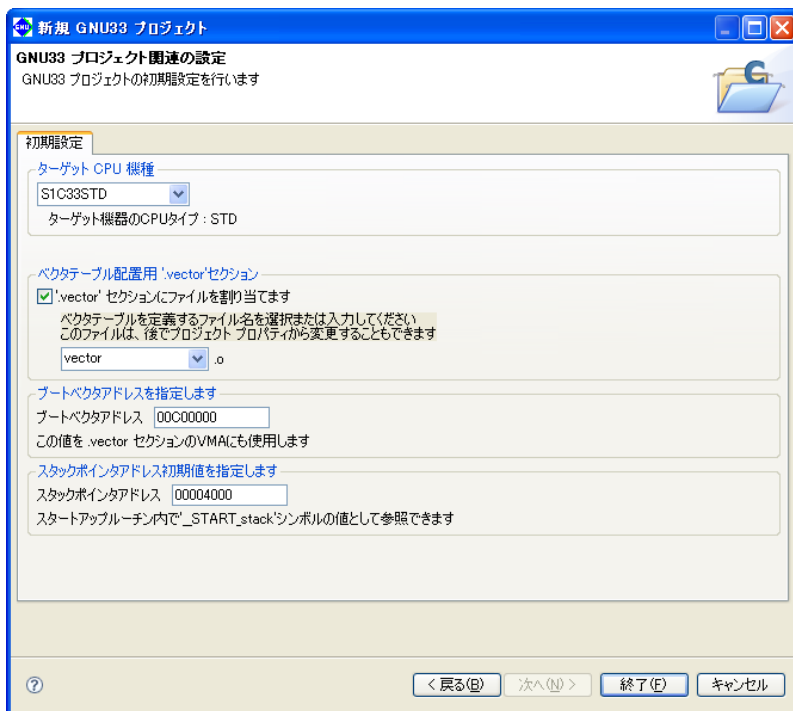


操作4: [プロジェクト名:]ボックスにプロジェクト名"sample"を入力します。

[デフォルト・ロケーションの使用]チェックボックスを選択しておくと、**IDE**起動時に指定したワークスペースディレクトリ内に"sample"というプロジェクトフォルダが生成されます。
また、プロジェクトのビルドにより生成される実行形式のオブジェクトファイル(.elf)にはここで指定した名前が付けられます。

操作5: [次へ>]ボタンをクリックします。

ターゲットCPUとベクタテーブルファイルの選択画面に移行します。



●ターゲットCPUの指定

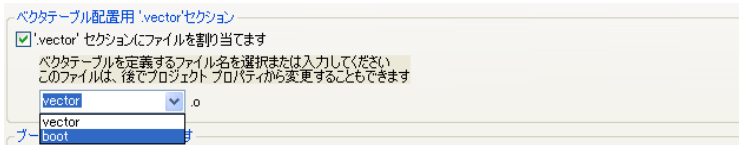


操作6: [ターゲット CPU 機種]コンボボックスからターゲットプロセッサの種類を選択します。ここでは"S1C33STD"を選択してください。

●ベクタテーブルファイルの指定

IDEでは割り込みベクタを確実にベクタテーブルベースアドレスから配置するために、`.vector`というベクタテーブル用のセクションをリンクスクリプトに定義するようになっています。この画面の[ベクタテーブル配置用'.vector'セクション]フィールドでは、チェックボックスで`.vector`セクションに特定のオブジェクトを配置するか否かを選択し、コンボボックスにはオブジェクトファイル名を設定します。ここでは、`.vector`セクションに`boot.o`を配置するものとして操作を進めます。

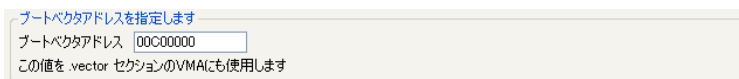
操作7: プルダウンリストから`boot.o`を選択します。



`.vector`セクションの詳細については、「5.7.7 リンクスクリプトの編集」を参照してください。

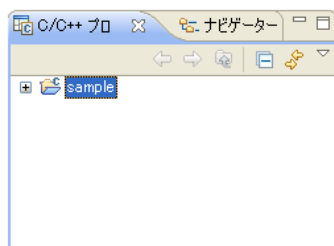
●ブートベクタアドレスの指定

[ブートベクタアドレスを指定します]フィールドでは、ブートベクタアドレスを指定します。デフォルト設定値は、ターゲットCPUに"S1C33STD"または"S1C33PE"を選択した場合は"00C00000"、"S1C33401"(ADV)を選択した場合は"20000000"となります。ここに設定した値は、IDEが自動生成するデバッグ起動用コマンドファイル内のTTBR設定コマンドのパラメータ、およびリンクスクリプト内の`.vector`セクションのVMAとして使用されます。このチュートリアルでは、変更する必要はありません。



操作8: [終了]ボタンをクリックします。

[新規 GNU33 プロジェクト]ウィザードが終了し、指定した名称のプロジェクトが作成されます。



ターゲットCPUやベクタテーブルファイルは、後から変更が可能です。

3.3.3 ソースファイルの作成、追加、編集

IDEはC、C++、アセンブラをサポートしており、これらの言語のソースファイルからオブジェクトを生成可能です。

オブジェクトの生成に必要なソースファイルはすべて、先に作成したプロジェクトに追加しておく必要があります。

●ソースファイルの作成

ソースファイルはIDEのエディタまたは汎用のエディタで作成します。また、S1C33 Family用アプリケーションの既存のソースファイルを使用することもできます。

このチュートリアルではサンプルとして用意されたソースファイルを使用します。

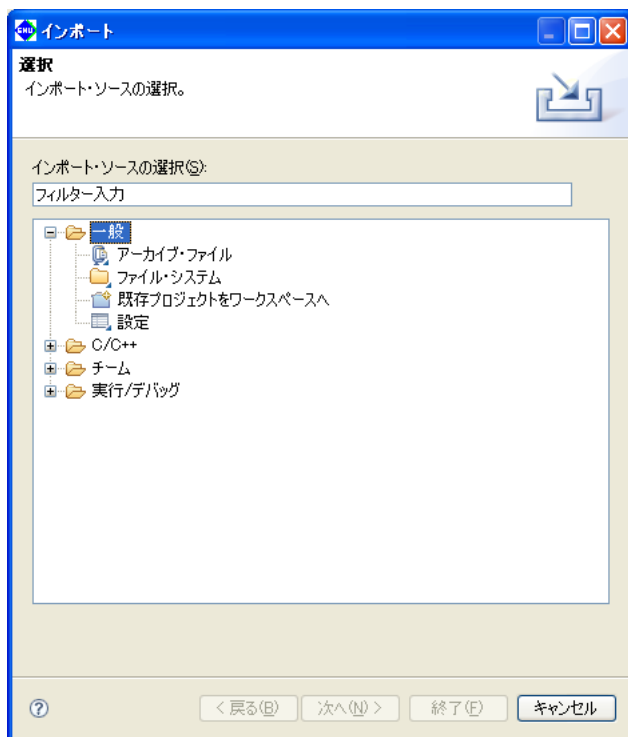
IDEによるソースの新規作成方法については"5.5 エディタとソースファイルの編集"を参照してください。

●ソースファイルの追加

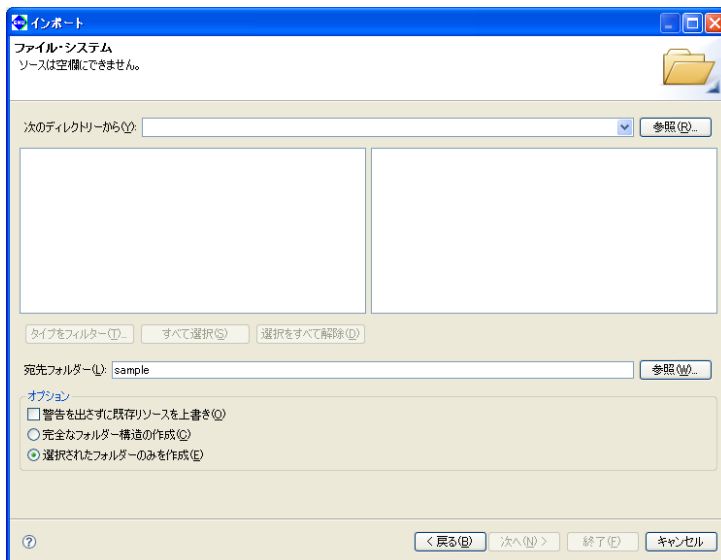
サンプルとして用意されているソースファイルをプロジェクトに読み込みます。

操作9: [ファイル]メニューから[インポート...]を選択します。

[インポート]ウィザードが起動します。



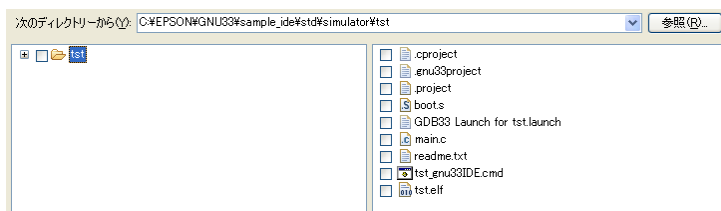
操作10: 表示されている一覧の中から[一般]>[ファイル・システム]を選択し、[次へ>]ボタンをクリックします。



操作11: [次のディレクトリから:]の[参照...]ボタンをクリックします。[ディレクトリからインポート]ダイアログボックスが表示されますので、**IDE**をインストールしたドライブ(C)から¥EPSON¥gnu33¥sample_id¥std¥simulator¥tstディレクトリを選択し、[OK]をクリックします。



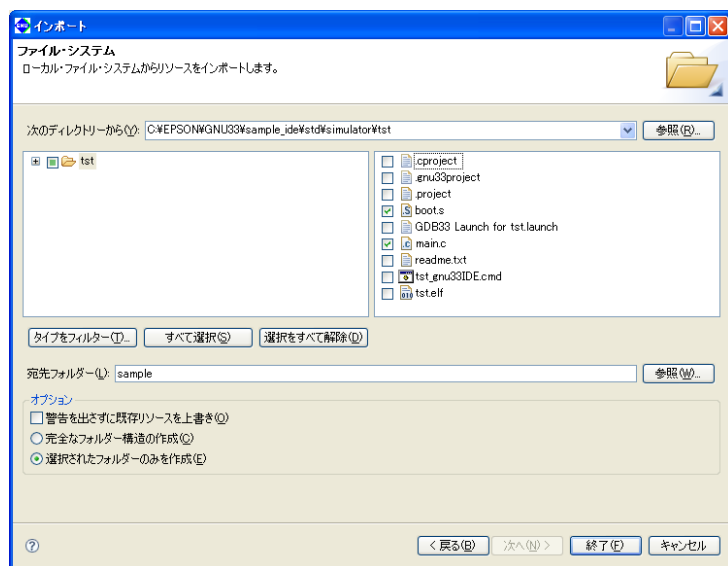
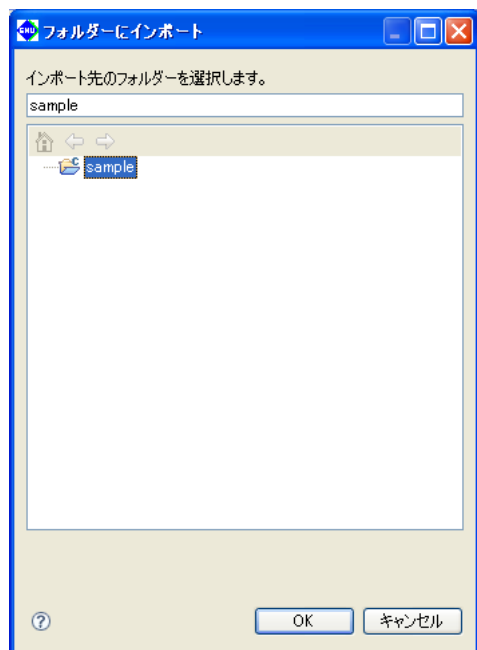
左のリストボックスに選択したディレクトリが、右のリストボックスにディレクトリ内のファイルの一覧が表示されます。



3 ソフトウェア開発手順

操作12: ファイルの一覧から"boot.s"と"main.c"を選択します。ファイル名の前のチェックボックスをクリックしてチェックを付けてください。

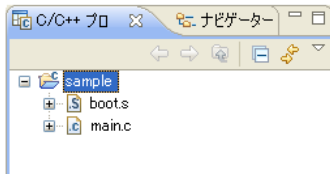
操作13: [宛先フォルダー:]の[参照...]ボタンをクリックします。[フォルダーにインポート]ダイアログボックスが表示されますので、"sample"フォルダを選択し、[OK]をクリックします。



操作14: 上記のとおり表示されていることを確認後、[終了]ボタンをクリックします。

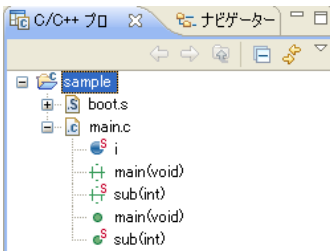
上記の操作により"boot.s"と"main.c"がプロジェクトに追加されました。

操作15: [C/C++ プロジェクト]ビューの"sample"をダブルクリック、または"sample"の前の[+]をクリックします。



[C/C++ プロジェクト]ビューの"sample"フォルダ内に追加したソースファイルが表示されます。

操作16: [C/C++ プロジェクト]ビューの"main.c"の[+]をクリックします。

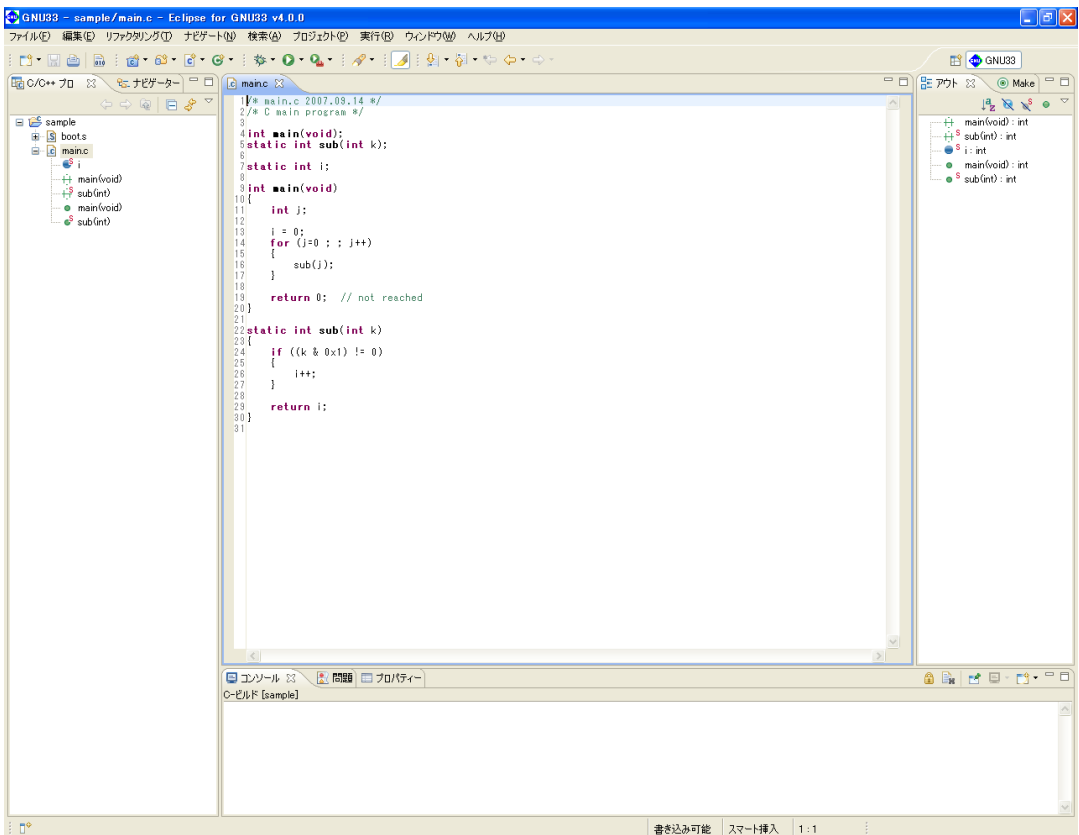


C/C++ソースの場合、そのファイルに定義されているグローバル変数と関数が表示されます。

●ソースファイルの表示と編集

プロジェクトに追加したソースファイルはIDEのエディタで表示および編集することができます。

操作17: [C/C++ プロジェクト]ビューの"main.c"をダブルクリックします。

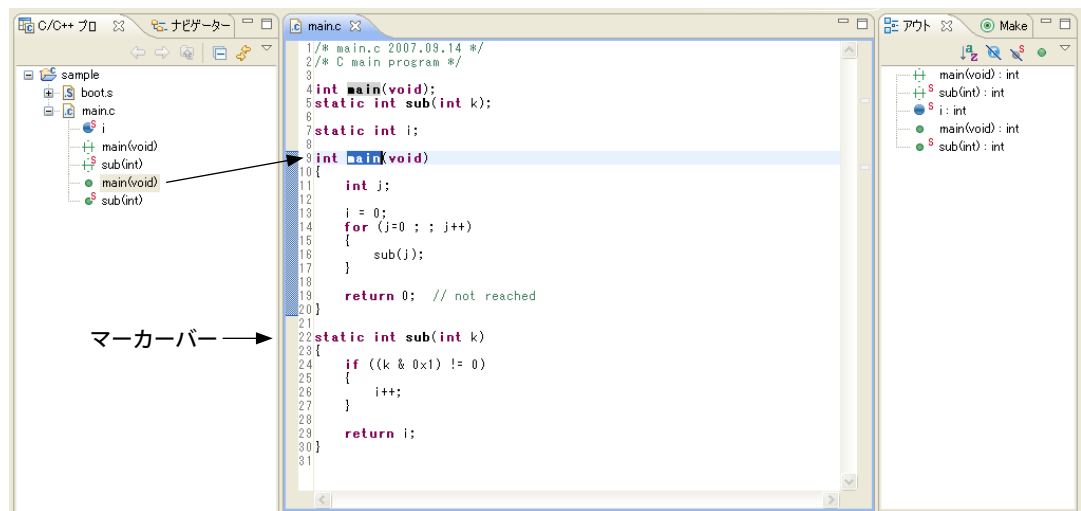


main.cの内容がエディタ上に表示されます。ここで汎用エディタと同様にソースの修正が行えます。また、通常使用している汎用エディタで選択したファイルが開くように設定することも可能です。詳細については"5.5 エディタとソースファイルの編集"を参照してください。

CまたはC++ソースを表示させた場合、C/C++の予約語、コメント、文字列が色付きで強調表示されます。

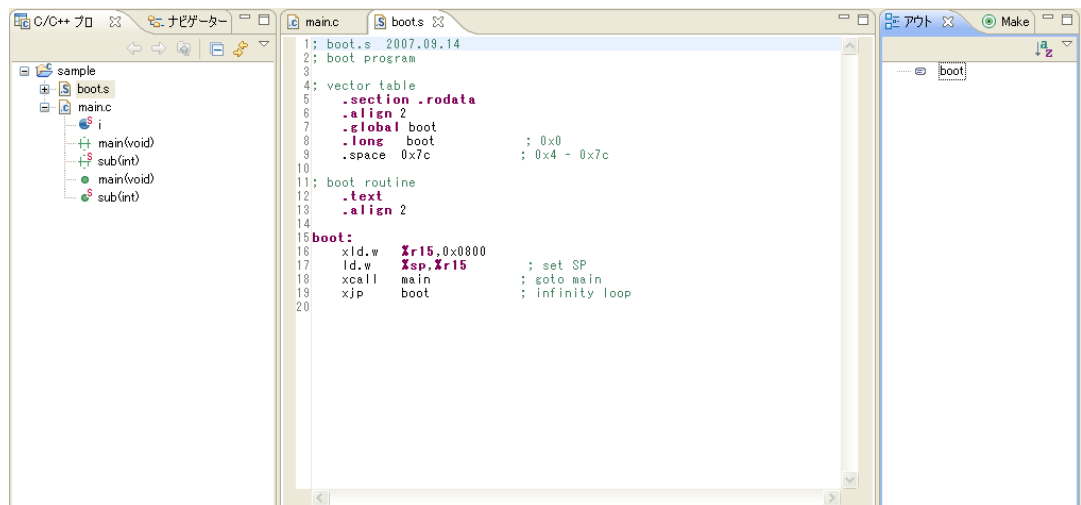
3 ソフトウェア開発手順

操作18: [C/C++ プロジェクト]ビューの"● main"をダブルクリックしてください。



エディタは"int main(void)"の行に飛んでハイライト表示します。また、エディタ左のマーカーバーにmain() 関数の範囲を示すバーが表示されます。このように、関数などを簡単に参照できます。[アウトライン]ビュー(右側にあるビュー)の"● main"をクリックしても同様です。

操作19: [C/C++ プロジェクト]ビューの"boot.s"をダブルクリックします。



複数のソースを同時に開くことができます。エディタ上部のタブ(ファイル名を表示)をクリックし、表示あるいは編集するソースを選択します。

アセンブラソースを表示させた場合、ラベル、擬似命令、レジスタが強調表示されます。

操作20: それぞれのソースのエディタタブにある  (閉じる) ボタンをクリックし、エディタを閉じます。

3.3.4 ビルドオプションとリンクスクリプトの編集

プロジェクトのビルド(実行形式オブジェクトファイルの生成)は**make.exe**によりコンパイラ、アセンブラ、リンカを起動して行います。ビルドに必要なmakeファイルは**IDE**で自動的に生成されますが、その中に記述するビルドオプション(コンパイラ、アセンブラ、リンカの起動オプション)はあらかじめ設定しておかなければなりません。また、リンクに必要なリンクスクリプトファイルの内容もビルド前に設定しておく必要があります。

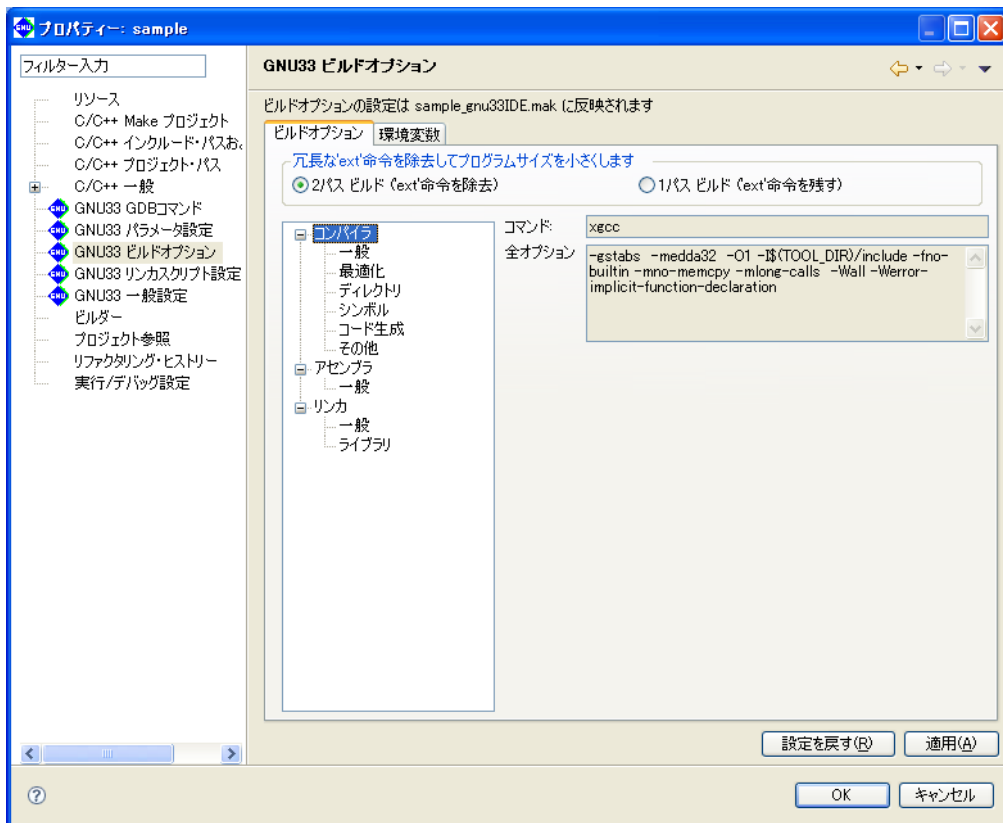
以下、それらの設定方法を簡単に説明します。

●ビルドオプションの設定

操作21: [プロジェクト]メニューから[プロパティ]を選択します。[プロパティ]は[C/C++ プロジェクト]ビュー内のプロジェクト名"sample"を右クリックして表示されるコンテキストメニューからも選択可能です。

[プロパティ]ダイアログボックスが表示されます。

操作22: 左側にあるプロパティのリストから[GNU33 ビルドオプション]を選択(クリック)し、[ビルドオプション]タブのページを表示させます。



ここで、コンパイラ、アセンブラ、リンカのコマンドラインオプションを設定できます。

ツリー表示されたツール名(コンパイラ、アセンブラ、リンカ)をクリックして選択すると、現在選択されているオプションが[全オプション]に表示されます。ツリー表示からオプションの分類をクリックして選択すると、個々のオプションを設定できる状態になります。

現在表示されているオプションは、プロジェクトを新規作成した際にデフォルトとして設定された内容です。

オプションの内容については各ツールの章を、オプション選択画面の詳細については"5.7 プログラムのビルド"を参照してください。

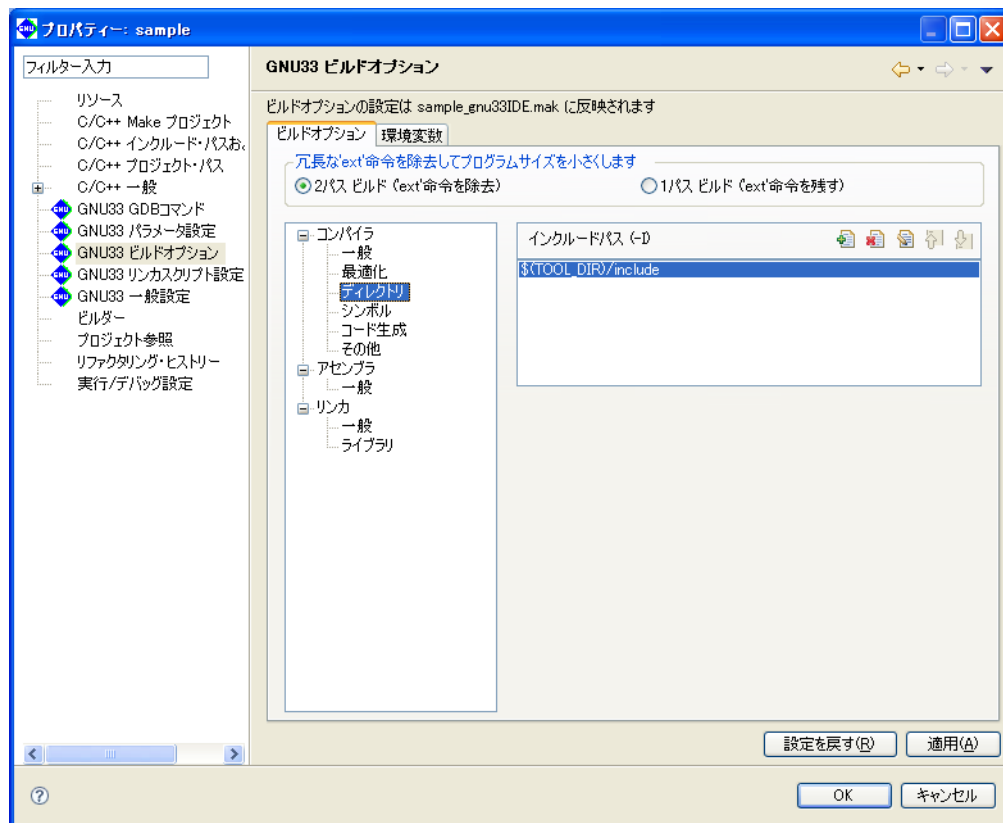
このチュートリアルでは、特に変更は必要ありませんが、ユーザのインクルードパスやライブラリファイルの追加方法をここで見ておきます。

●インクルードパスの追加

以下、操作23～操作26は参考例ですので、特に操作する必要はありません。

操作23: [ビルドオプション]のツリー表示から[コンパイラ]>[ディレクトリ]を選択します。

C/C++コンパイラの-E(インクルードパス指定)オプションを設定するページが表示されます。

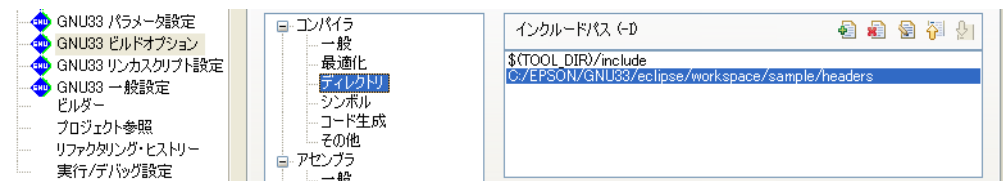


ユーザのヘッダファイルが別のディレクトリに用意されている場合は、このリストに次のように追加します。



操作24: [追加]ボタンをクリックします。ディレクトリ選択ダイアログボックスが表示されますので、パスを入力するか、[ファイル・システム...]ボタンで表示されるフォルダ選択ダイアログボックスから選択します。

ディレクトリ選択ダイアログボックスを終了すると、入力/選択したパスが次のようにリストに追加されます。



その他のボタンの機能は以下のとおりです。

-  **[削除]** リスト内で選択したパスを削除します。
-  **[編集]** リスト内で選択したパスを編集します。パスを書き換えるダイアログボックスが表示されます。
-  **[上へ移動]** リスト内で選択したパスを一覧内の一つ上に移動します。インクルードファイルは一覧内の上にあるパスから順に検索されます。
-  **[下へ移動]** リスト内で選択したパスを一覧内の一つ下に移動します。

操作25: [適用]ボタンで変更内容を確定します。

以上の設定により、C/C++コンパイラはインクルードファイルを、ここで指定したディレクトリからも読み出せるようになります。

パスを指定するマクロと環境変数について

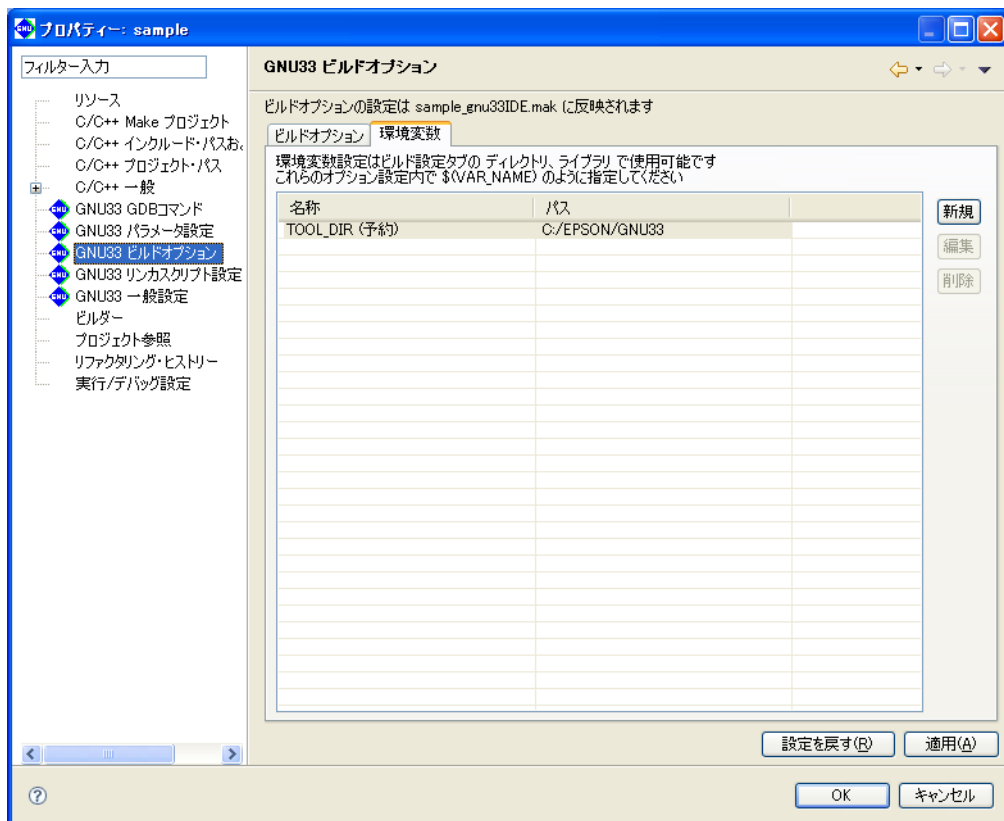
デフォルト設定で、[インクルードパス (-I)]には"\$ (TOOL_DIR) /include"が表示されています。
\$(環境変数) は、makeファイル(ビルド時に生成)に記述されるマクロです。TOOL_DIRは、gnu33ツールをインストールしたディレクトリへのパスが定義されている環境変数で、定義内容は[環境変数]タブのページで確認できます。

例: gnu33ツールをc:\EPSON\gnu33ディレクトリにインストールした場合

```
TOOL_DIR = c:/EPSON/gnu33
```

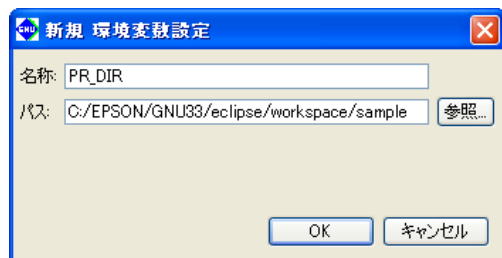
make.exeの実行時は、マクロが () 内の環境変数の内容に置き換えられますので、-I\$(TOOL_DIR)/includeは-Ic:/EPSON/gnu33/includeの指定となります。

[環境変数]タブのページでは、TOOL_DIRと同様なユーザ独自の環境変数を定義することができます。



定義方法は次のとおりです。

操作26: [新規]ボタンをクリックして[新規 環境変数設定]ダイアログボックスを表示させます。
 [名称:]テキストボックスに環境変数名を入力します。
 定義するパスはキーボードから、あるいは[参照...]ボタンで選択して[パス:]テキストボックスに入力します。
 入力後、[OK]ボタンでダイアログボックスを閉じます。

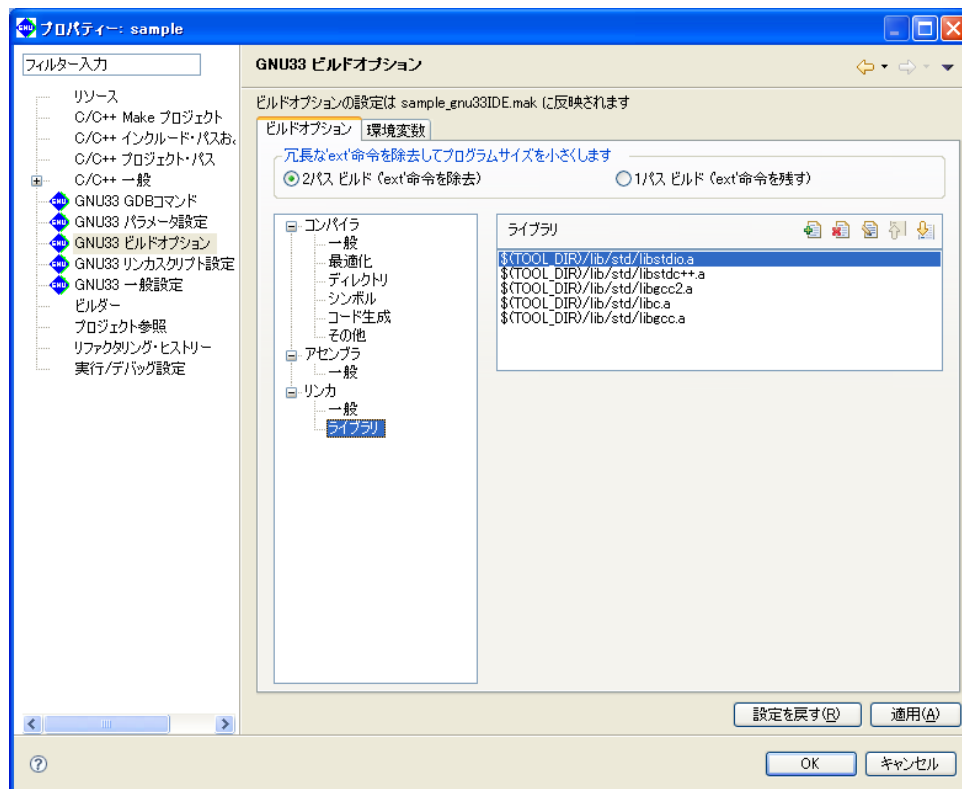


ここで定義した環境変数は、ビルドオプションのインクルードパスおよびライブラリファイルのパスの指定に使用可能です。使用する場合は、上記例のように\$(環境変数)のマクロとして入力してください。

●ライブラリファイルの追加

以下、操作27～操作29は参考例ですので、特に操作する必要はありません。

操作27: [ビルドオプション]のツリー表示から[リンカ] > [ライブラリ]を選択します。
 ライブラリファイルを設定するページが表示されます。



[ライブラリ]にはデフォルト設定で本パッケージに含まれるシミュレーテッドI/Oライブラリ、C++ライブラリ、long long型エミュレーションライブラリ、ANSIライブラリ、エミュレーションライブラリが表示されています。

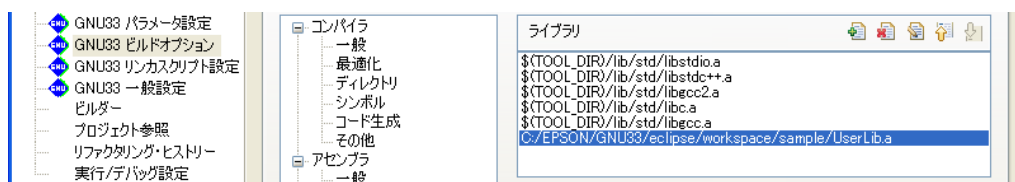
ユーザのライブラリファイルが用意されている場合は、このリストに次のように追加します。



操作28: [追加]ボタンをクリックします。ファイル選択ダイアログボックスが表示されますので、パスを入力するか、[ファイル・システム...]ボタンで表示される[開く]ダイアログボックスから選択します。

パスの指定には、[環境変数]タブのページで定義した環境変数を使用可能です。

ファイル選択ダイアログボックスを終了すると、入力/選択したファイルが次のようにリストに追加されます。



その他のボタンの機能は、前述のインクルードパスの場合と同様です。

操作29: [適用]ボタンで変更内容を確定します。

以上の設定により、ここで指定したライブラリもリンクの対象になります。

● リンカスクリプトの設定

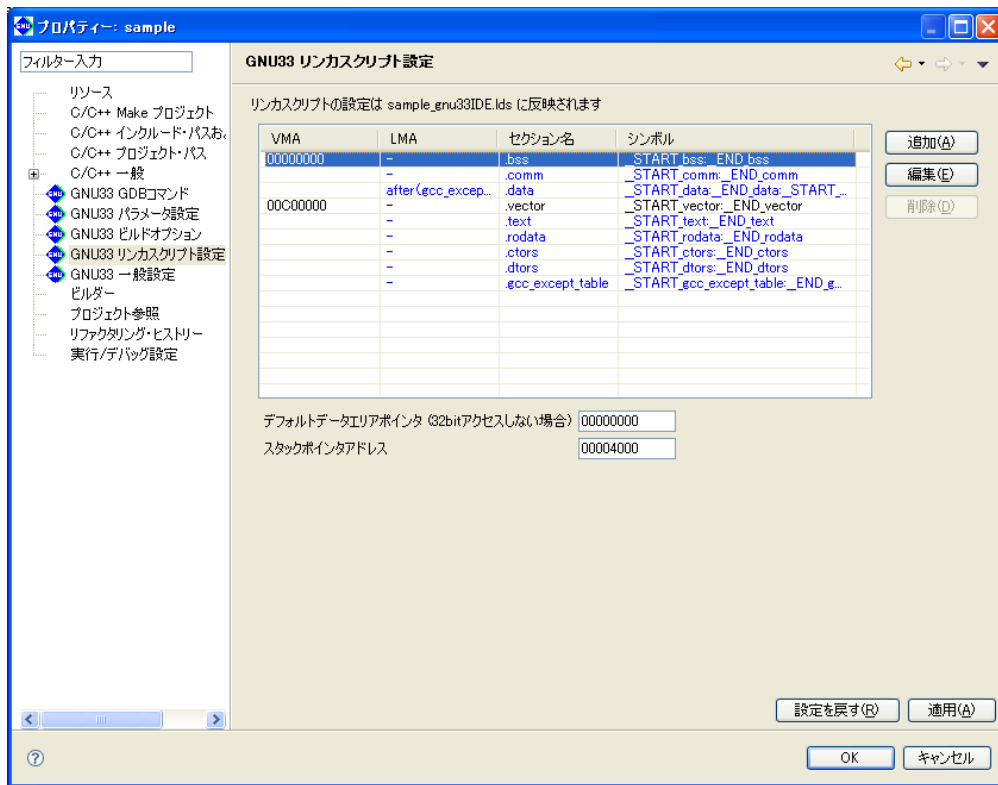
ビルドの実行にはリンカスクリプトファイルが必要で、このファイルもIDEで作成することができます。

リンカスクリプトファイルは、リンクにセクションの配置と構成を指示するために使用します。たとえば、アセンブラで生成された1つのオブジェクトファイルを見ると、プログラムコード部分、静的データ部分、変数部分というように、データ属性で分類されるコードのまとまりがあります。これらのまとまりが1つのセクションです。リンカから見ると、これは入力セクションとなります。リンカは複数の同種入力セクションを1つにまとめて(出力セクションに構成し直し)、実行形式のオブジェクトファイルを生成します。また、同じ属性でも、外部ROMのアドレスAにはソース1と2から生成されたオブジェクトのプログラムコード部を配置し、アドレスBには内部RAMに転送して実行させるソース3から生成されたオブジェクトのプログラムコード部を配置するというように、配置するアドレスやデバイスによっても分けておく必要があります。

このように、どの入力セクションをまとめて1つの出力セクションを構成し、どのアドレスから格納し、またどのアドレスで実行させるかという情報を指定するのが、リンカスクリプトファイルの役目です。もう少し詳しい説明については、"3.7 データエリア、セクション"を参照してください。

操作30: [プロパティ]ダイアログボックスを閉じてしまった場合は、[プロジェクト]メニューから [プロパティ]を選択して再度開いてください。

操作31: プロパティのリストから[GNU33 リンカスクリプト設定]を選択(クリック)します。



セクション名欄を見ると、.bss、.comm、.data、.vector、.text、.rodata、.ctors、.dtors、.gcc_except_tableの9つの基本的なセクション(出力セクション)があらかじめ設定されていることが分かります。

.bss	初期値を持たない変数を置くセクション(通常はRAMに置きます)
.comm	初期値を持たない変数を置くセクション(通常はRAMに置きます)
.data	初期値を持つ変数を置くセクション(初期値をROMに置いておき、使用時はRAMにコピーしてアクセスします)
.vector	ベクタテーブルを置くセクション(実データはROMに置きます)
.text	プログラムコードを置くセクション(実データはROMに置き、その位置で、あるいは高速なRAMなどにコピーして実行します)
.rodata	定数(実データはROMに置きます)
.ctors	グローバルクラスのコンストラクタ関数へのポインタ配列(実データはROMに置きます)
.dtors	グローバルクラスのデストラクタ関数へのポインタ配列(実データはROMに置きます)
.gcc_except_table	例外処理用テーブルデータ(実データはROMに置きます)

VMA(Virtual Memory Address)は実行時にセクションを置く位置(先頭アドレス)です。VMAにアドレスの記載のないセクションはその一つ上のセクションに続いて配置することを表します。

LMA(Load Memory Address)は実データを置いておくROM内の位置(先頭アドレス)です。"- "はVMAと同じ(実データが置かれた位置で実行またはアクセスされる)であることを表します。"after(.gcc_except_table)"は()内のセクション、この場合は.gcc_except_tableセクションに続いて実データが配置されることを示します。

[シンボル]にはセクションが配置される領域の先頭および終了アドレスを示すラベルが表示されます。LMAが指定されない場合は<VMAの先頭>:<VMAの終了>の2つのラベルが、LMAが指定されている場合は<VMAの先頭>:<VMAの終了>:<LMAの先頭>:<LMAの終了>の順に4つのラベルが表示されます。これらのラベルは、ROMからRAMへのセクションのコピーの際など、ソースファイル内でのアドレス指定に使用できます。

実際のプログラム開発時は、[追加]ボタンでユーザ独自のセクションを追加することもできます。

セクション情報は、.vectorのみ黒色、その他のセクションは青色で表示されています。青色はデフォルトで定義済みの標準セクションを表し、黒色はそれ以外の新たに定義したユーザセクションを表します。

ユーザセクションは、セクション名、セクションの基本属性、配置アドレス、配置するオブジェクトを任意に編集可能です。標準セクションは配置アドレスのみ指定可能で、ユーザセクションに配置された以外のオブジェクトが自動的に配置されるようになっています。

[デフォルトデータエリアポインタ]テキストボックスは、リンカスクリプトに記述するデフォルトデータエリアポインタ(アドレス)の設定に使用します。データエリアポインタは、C33コアの内部レジスタをポインタとして使用し、そのレジスタをベースアドレスとして規定範囲内のデータをアクセスすることで、命令数を減らすことを目的としています。

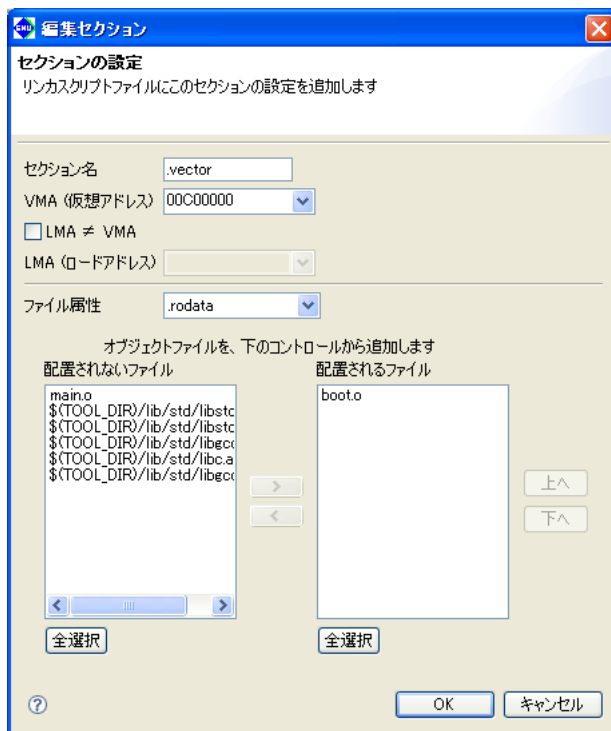
デフォルト設定では、ベースアドレスを0番地とするデータポインタ定義がリンカスクリプトに記述されます。データエリアについては、"3.7 データエリア、セクション"を参照してください。

上記の各セクションには、あらかじめプロジェクトのオブジェクトファイルが配置されるように定義されています。

.vectorセクションの例を見えます。

操作32: セクションの一覧から".vector"を選択し、[編集]ボタンをクリックします。

[編集セクション]ダイアログボックスが表示されます。



ダイアログボックスの上部は、前の画面にあったセクションの設定を行います。

下部右側のリストボックスには、.vectorセクションに配置されるオブジェクトが表示されます。新規プロジェクトウィザードで.vectorセクションに配置するように設定したboot.oが表示されています。

左側のリストボックスには、プロジェクトの残りのオブジェクトファイルとライブラリファイルの一覧が表示されています。

もし、左側のリスト中のいずれかのオブジェクトをこのセクションに配置する場合は、リストからそのファイルを選択して[>]ボタンをクリックします。選択したファイルが右のリストボックスに移動し、このセクションを構成するファイルリストに加えられます。

右側のリストに複数のファイルが表示されている場合は、リストの上から順に配置されることを表しています。配置順序は[上へ]または[下へ]ボタンで変更可能です。

この時点では、まだオブジェクトファイルが生成されていませんが、プロジェクトに追加したソースファイルから同じ名前のオブジェクトファイル(`boot.o`、`main.o`)が生成されるものとして表示されています。

ここで、[ファイル属性]を見ると`.rodata`となっています。これは、`boot.o`内の`.rodata`セクションのみが`.vector`セクションに配置されることを示しています。`boot.o`内の他のセクションはそれぞれ同じ属性のセクションに配置されるようになっていきますので、その他のセクション情報を見ると同属性の`.rodata`セクション以外は、すべて`boot.o`を配置するように設定されています。

IDEの初期設定では、ベクタテーブルが`.rodata`セクションに記述されているものとしています(C/C++ソースでは`const`宣言した定数、アセンブラソースでは`.rodata`セクションの範囲内の定数)。ソース内のベクタテーブルが他の属性のセクション(たとえば`.text`セクション)に記述されている場合は、そのセクションが`.vector`セクションに配置されるように[ファイル属性]から選択してください。

また、[VMA(仮想アドレス)]は、プロジェクトの新規作成時に指定したブートベクタアドレスの値に設定されています。プロセッサに設定されているブートベクタアドレスの値がこれと異なる場合は、[VMA(仮想アドレス)]の値を正しく入力し直してください。

操作33: [OK]ボタンをクリックしてダイアログボックスを閉じます。

操作34: [プロパティ]ダイアログボックスの[OK]ボタンをクリックして、リンクスクリプトの編集を終了します。

上記のようにユーザセクション(黒で表示)に配置するオブジェクトや属性を編集すると、同じ属性を持つ標準セクションのオブジェクトの構成も自動的に更新されます。

以上で、プログラムのビルドの準備が完了しました。

3.3.5 プログラムのビルド

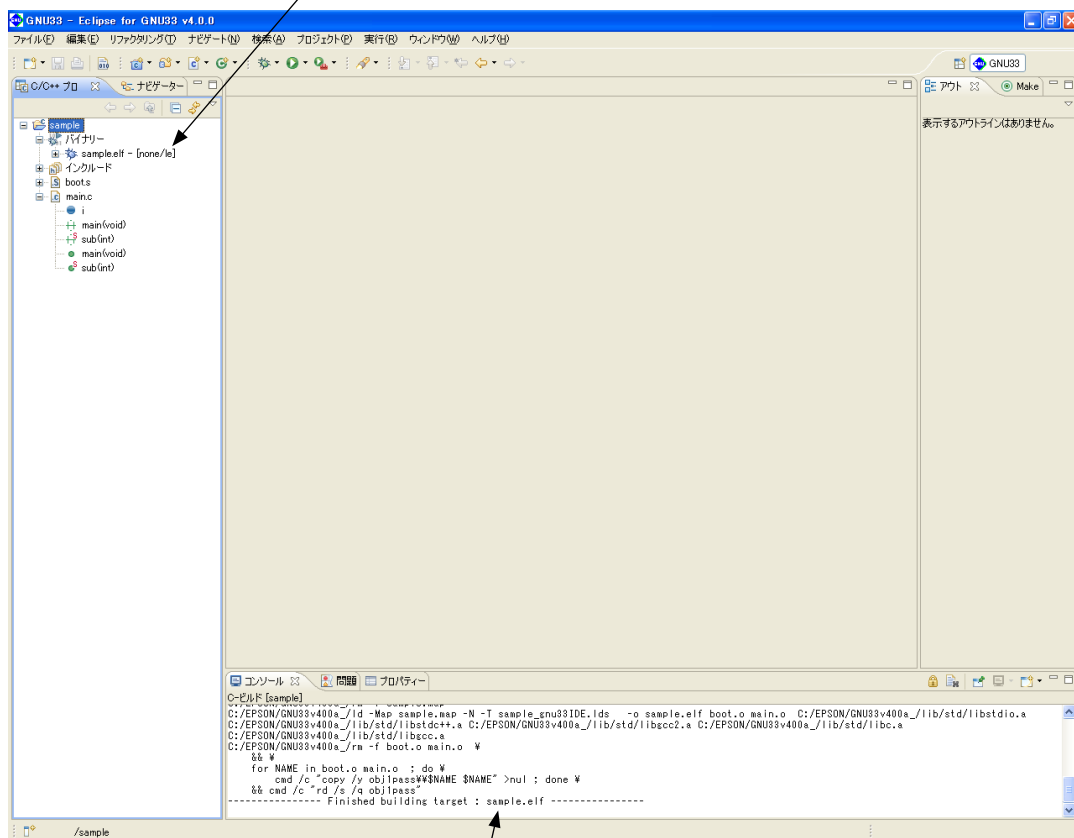
前節までの作業が終わると、プログラムのビルド(コンパイル、アセンブル、リンク)が行えます。

●ビルドを実行するには

操作35: [C/C++ プロジェクト]ビュー内のプロジェクト名"sample"を選択します。

操作36: [プロジェクト]メニューから[プロジェクトのビルド]を選択します。[プロジェクトのビルド]は[C/C++ プロジェクト]ビュー内のプロジェクト名"sample"を右クリックして表示されるコンテキストメニューからも選択可能です。

この操作により、設定されている内容でmakeファイルを生成後**make.exe**が実行され、実行形式のオブジェクトファイル**sample.elf**が生成されます。



ビルド中に実行されるコマンドやツールメッセージが[コンソール]ビューに表示されます。

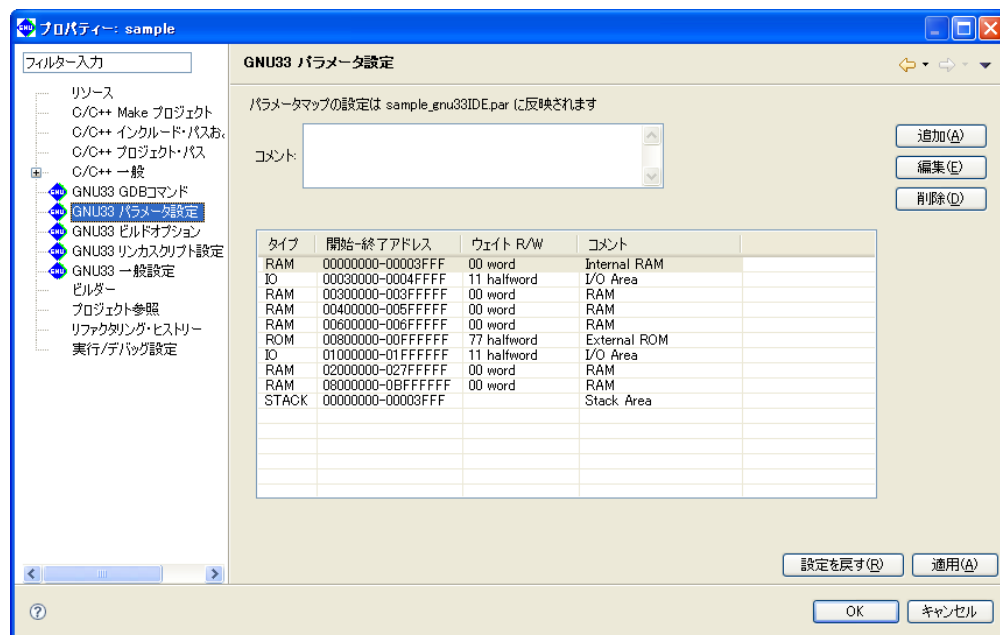
3.3.6 デバッグ

デバッグを行う前にデバッグ用のパラメータファイルを作成します。パラメータファイルは、ターゲットシステムのメモリマップ情報を記述しておくファイルで、デバッグに読み込んでマップの設定を行います。ターゲットシステムのメモリ構成に合わせて作成し、必ずデバッグに読み込ませてください。また、デバッグの起動オプションもあらかじめ設定しておく必要があります。

●パラメータマップの設定

操作37: [ナビゲーター]ビューまたは[C/C++ プロジェクト]ビューでsampleプロジェクトを選択後、[プロジェクト]メニューから[プロパティ]を選択します。

操作38: プロパティのリストから[GNU33 パラメータ設定]を選択(クリック)します。



デフォルト設定のエリア情報が10個表示されます。

たとえば、一番上のRAMの情報は、エリア0の0x0～0x3fff(16Kバイト)をRAM領域として使用することを定義しています。"00 word"とあるのは、読み出しに0サイクル、書き込みに0サイクルのウェイトを挿入し、ワードでアクセスするデバイスであることを示します(このアクセス条件の設定はシミュレータモードでのみ有効です)。

その他も同様で、I/Oメモリ領域、RAM領域、ROM領域、最後にRAM内のスタック領域が定義されています。

これは、C33 STDまたはC33 PEコア搭載のS1C33マイクロコンピュータの基本構成です。

これ以外のメモリや外部デバイスを使用する場合は、[追加]ボタンをクリックして追加するエリアの設定を行います。

サンプルプログラムは、特にデフォルト以上のメモリ構成を必要としませんので、このままパラメータファイルを作成しても問題ありません。

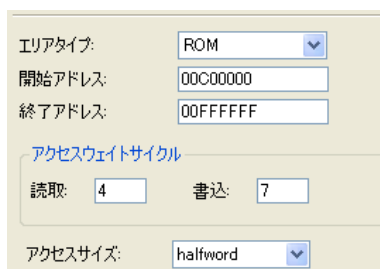
ここでは操作の参考として、ROM領域(0x800000～0xfffffff)の開始アドレスを0xc00000に、読み出しのウェイト数を4に変更してみます。

操作39: リストボックス内のROMの行をクリックして反転表示させ、[編集]ボタンをクリックします。

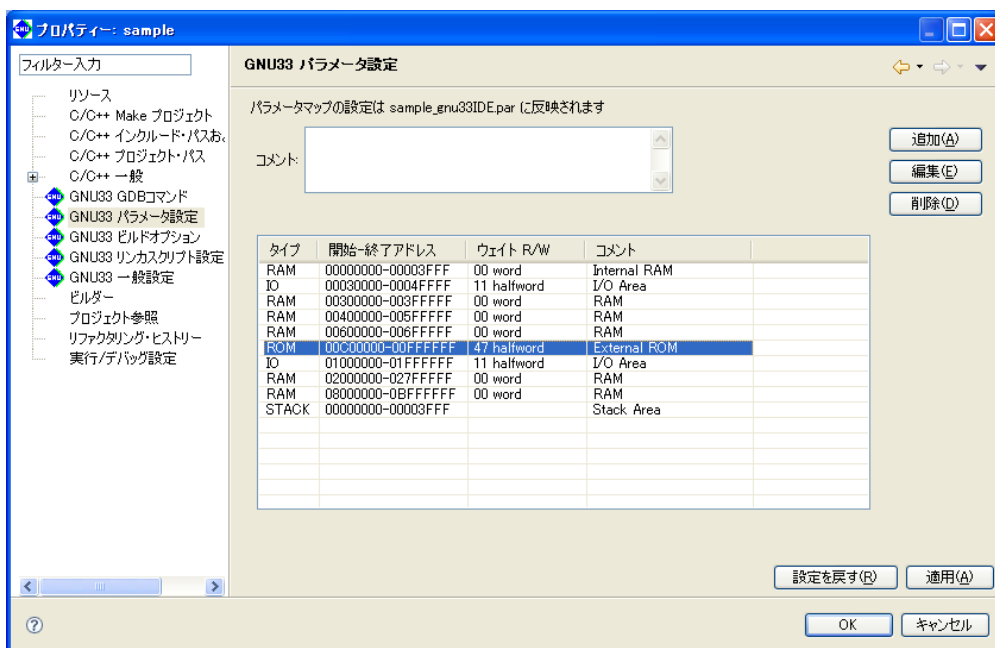
[編集 パラメータ]ダイアログボックスが表示されます。



操作40: [開始アドレス:]テキストボックスに00C00000を、[読取:]テキストボックスに4(4ウェイト)を入力します。



操作41: [OK]ボタンをクリックします。



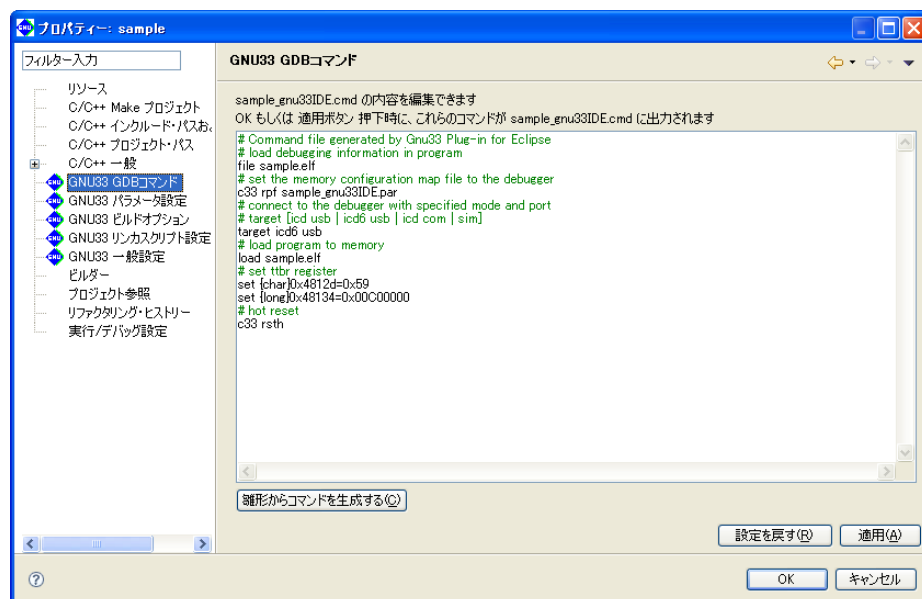
領域の表示が"00C00000-00FFFFFF"に、アクセス条件の表示が"47 halfword"に変更されました。

操作42: [適用]ボタンをクリックします。

以上の設定により、デバッグ起動時にsample_gnu33IDE.parというファイルが生成され、コマンドファイルを介してデバッグに渡されます。

●デバッガ起動オプションの設定

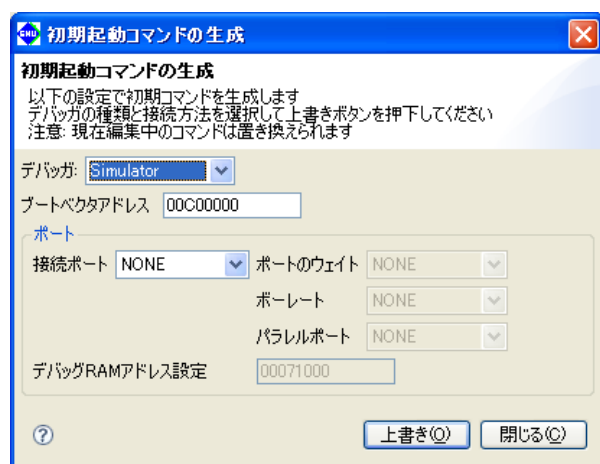
操作43: プロパティのリストから[GNU33 GDBコマンド]を選択(クリック)します。



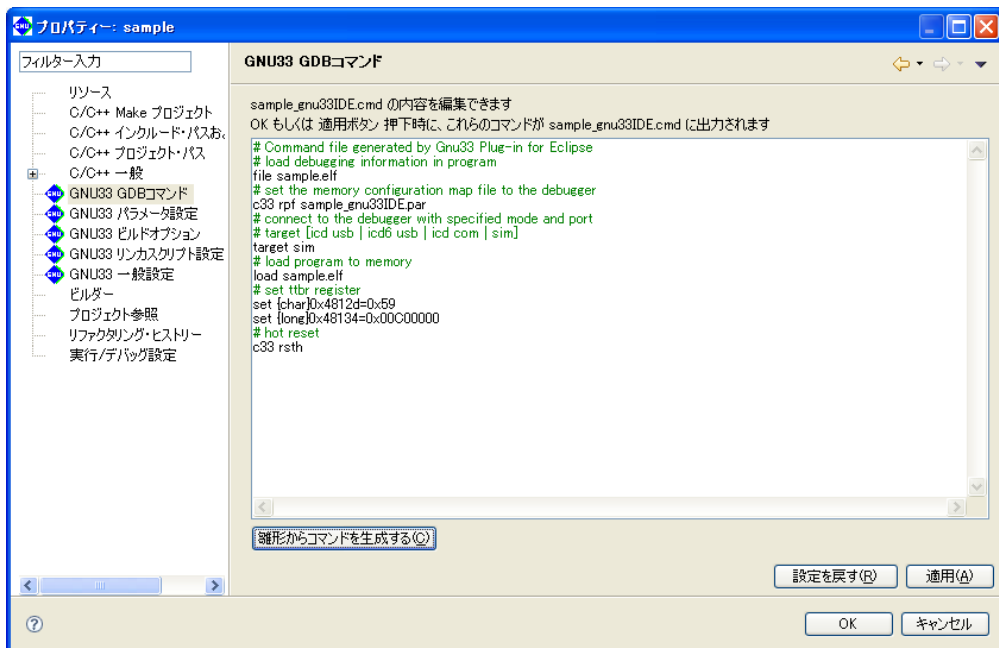
ここには、**IDE**が生成するデバッガ起動用コマンドファイルの内容が表示されます。デバッガは使用するICDなどに合わせたモードに設定して動作させる必要があります。詳細は、「3.6 デバッグ環境」を参照してください。ここでは、外部機器の不要なシミュレータモードに設定してデバッグを行うことにします。デフォルトで表示されるコマンドファイルの内容は、ICD Ver.6を使用してデバッグを行うようになっています。以下の操作でシミュレータモード用に変更できます。

操作44: [雛形からコマンドを生成する]ボタンをクリックして、[初期起動コマンドの生成]ダイアログボックスを表示させます。

操作45: [デバッガ:]コンボボックスから"Simulator"を選択します。



操作46: [上書き]ボタンをクリックします。



コマンドファイルの表示がシミュレータ用のコマンド構成に変更されました。
コマンドの修正や追加が必要な場合は、テキストボックス内で直接編集が可能です。

操作47: [OK]ボタンをクリックします。

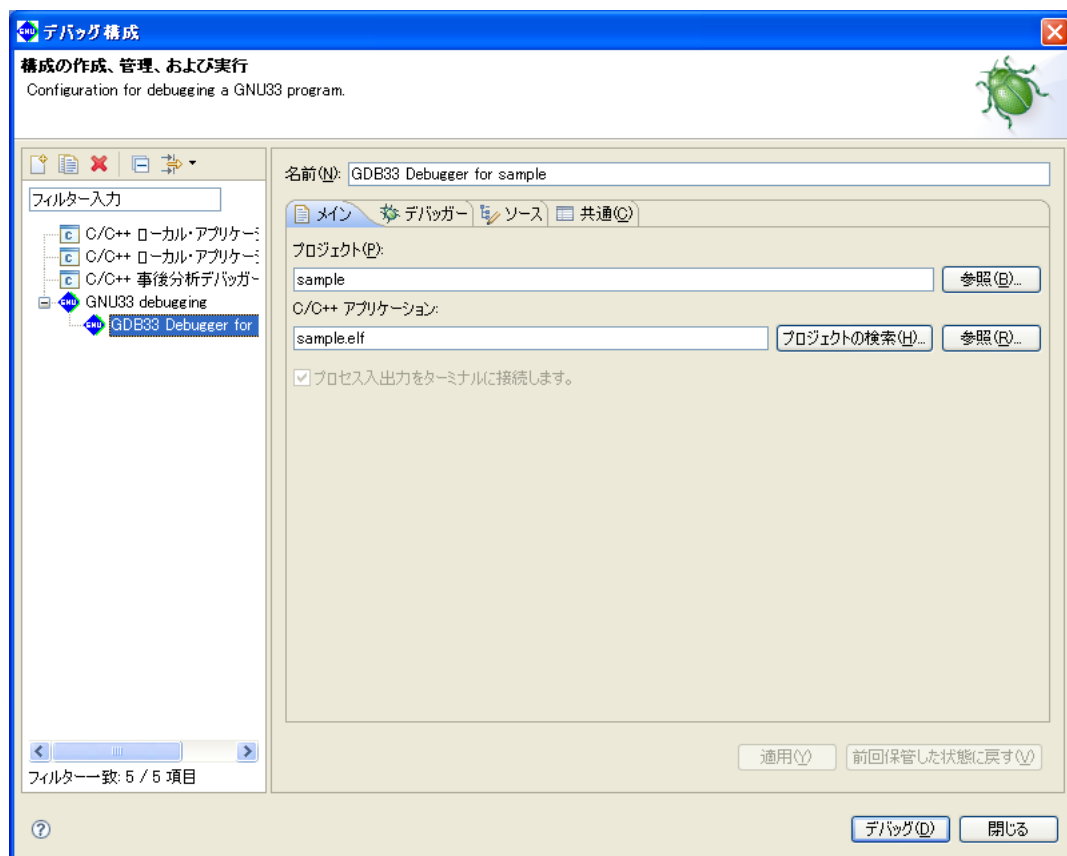
以上の設定により、sample_gnu33IDE.cmdというコマンドファイルが生成され、デバッガに渡されます。

●デバッガの起動

操作48: [実行]メニューから[デバッグの構成]を選択します。

[デバッグ構成]ダイアログボックスが表示されます。

操作49: リストから[GDB33 Debugger for sample]を選択します。

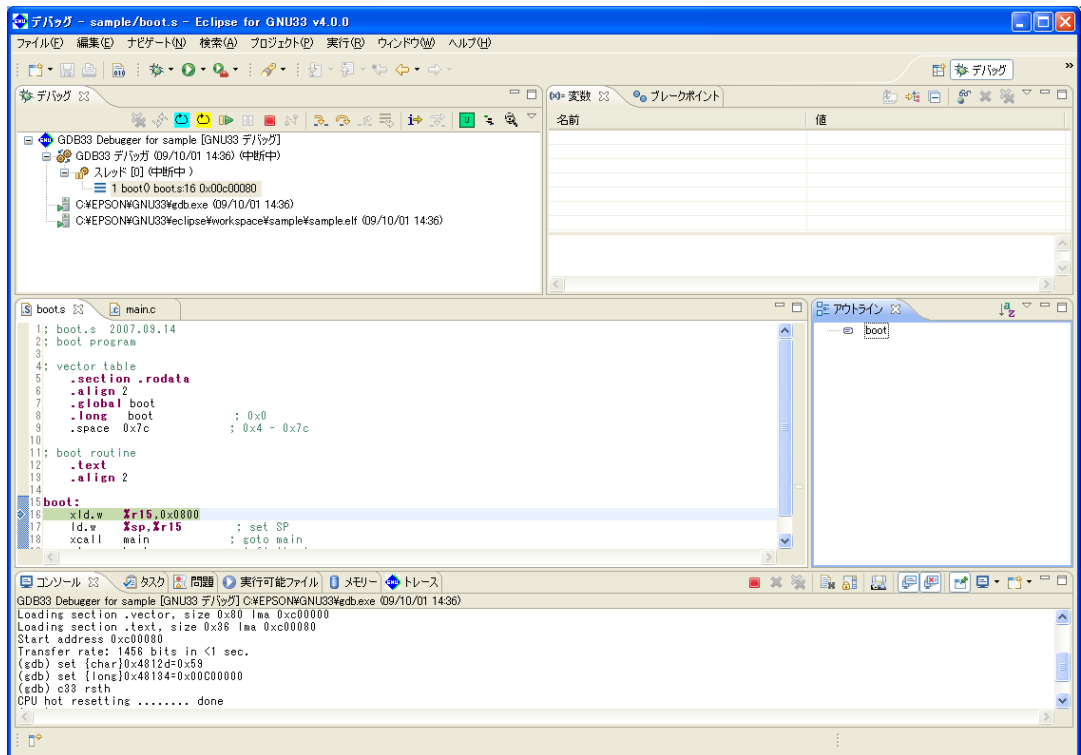


このダイアログでデバッガgdbのコマンドラインを編集することができます。サンプルの実行には特に変更の必要はありませんので、このままの設定でデバッガを起動します。

操作50: [デバッグ]ボタンをクリックします。

デバッガgdbが起動します。

デバッガが起動すると次のウィンドウが開き、[GNU33 GDBコマンド]ダイアログボックス等で設定したコマンドファイルを実行します。



コマンドファイルによってオブジェクトファイルがロードされ、ホットリセットされます。PC(プログラムカウンタ)がプログラム実行開始位置に設定され、すぐにデバッグを開始できる状態になります。

●プログラムを実行させるには

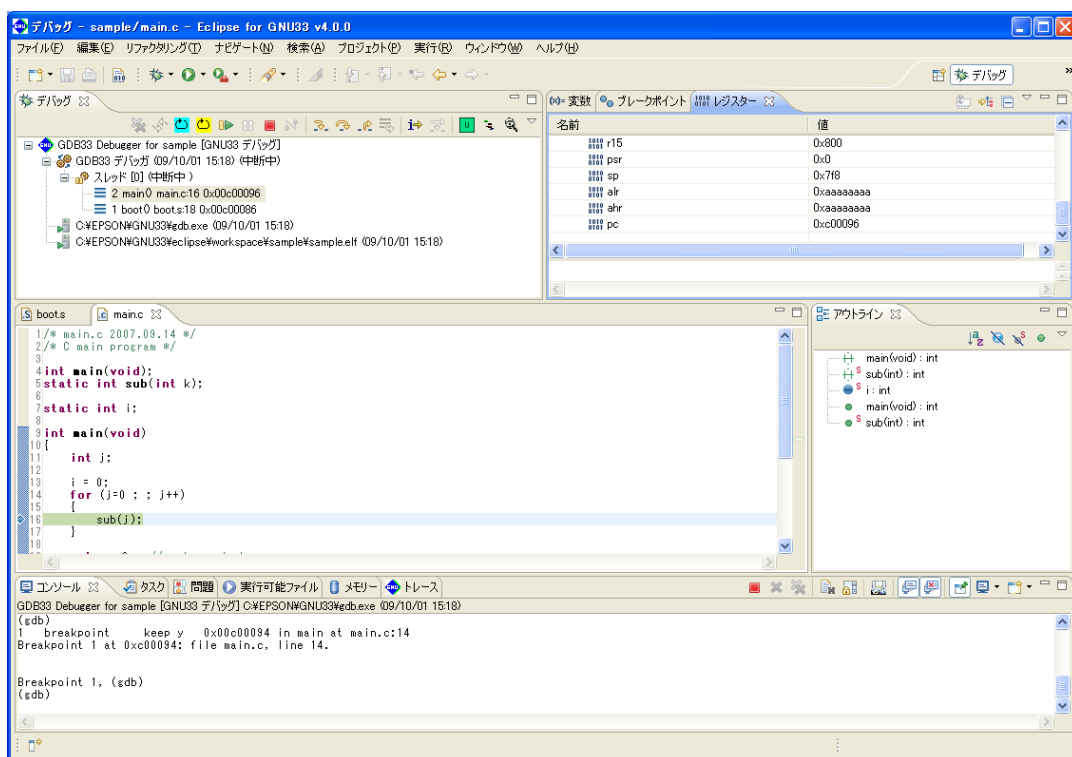
 **操作51:** ツールバーの[再開]ボタンをクリックします。

サンプルプログラムはRAM領域のint変数i(0x00000000~0x00000003番地)をカウンタとしてインクリメントを永遠に繰り返します。

このような永久ループは、強制ブレークによって停止させます。

●強制的にブレークさせるには

 **操作52:** ツールバーの[中断]ボタンをクリックします。



[ソース]エディタ内の"`sub(j);`"が緑色で強調表示されているのは、現在のPC(プログラムカウンタ)のアドレスがそこにあり、プログラムがそこで停止していることを示しています。また、[レジスター]ビューのpcを見ると、0x00c00096番地を示しています。これで、この番地の命令を実行する直前に停止したことが分かります。

実行開始時はboot.sが[ソース]エディタに表示されていましたが、現在はmain.cの中で停止しているため、main.cのソースが表示されています。

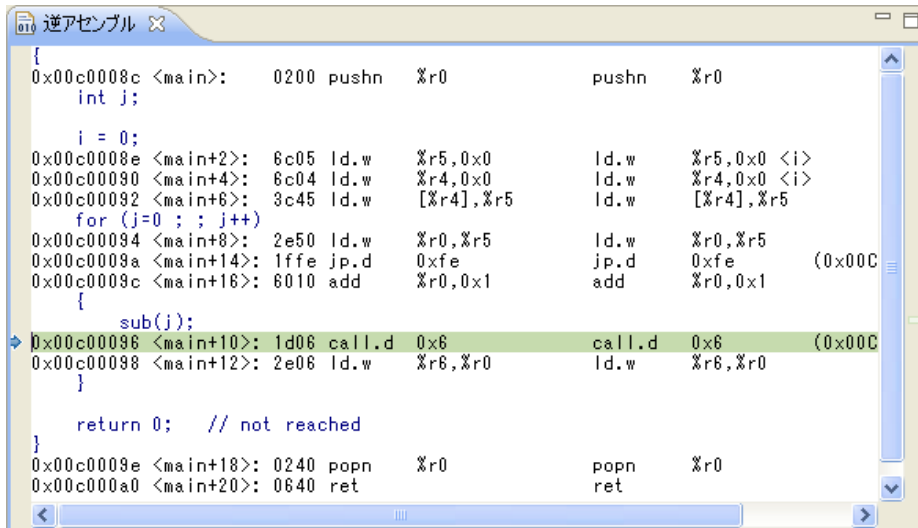
[ソース]エディタに表示されているソースは、[逆アセンブル]ビューに逆アセンブルした形式のプログラムを表示することができます。

●[逆アセンブル]ビューの表示方法

操作53: [ウィンドウ]メニューから[ビューの表示]>[逆アセンブル]を選択します。
[逆アセンブル]ビューが表示されます。

逆アセンブル

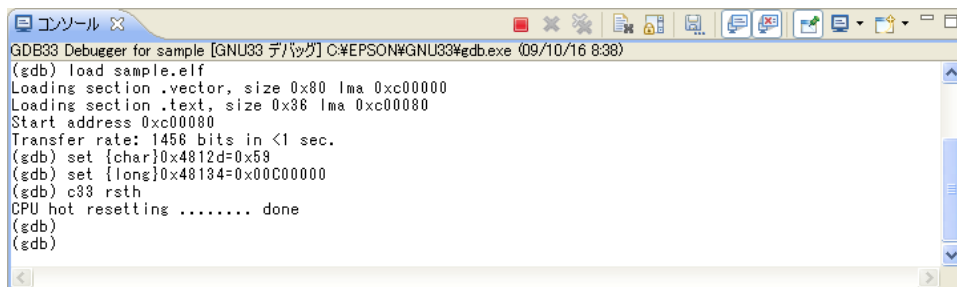
逆アセンブルは、ソース表示の各行ごとに、対応するアセンブラ表示を挿入します。この表示により、プログラムが停止しているソース行とともに、アドレスとその命令コードも[逆アセンブル]ビュー上で分かります。この場合も、Cソースは関数単位の表示となります。



操作54: [ソース]エディタを選択し、ビューを元に戻してください。

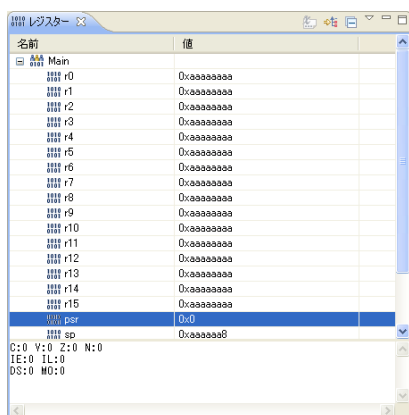
ここで、その他のビューも見ておきましょう。

●デバッガのウィンドウについて

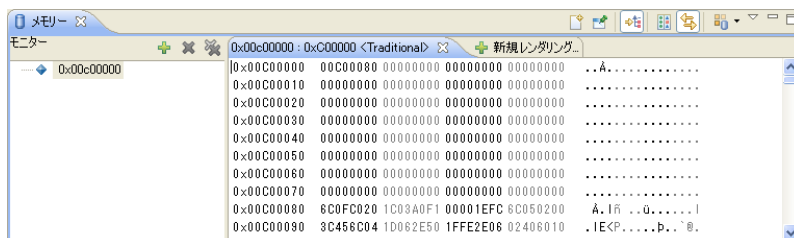


これは[コンソール]ビューです。(gdb)というプロンプトの位置にデバッグコマンドを入力し、実行させます。

たとえば、操作51のプログラム実行はボタン操作でしたが、ここにcontと入力し[Enter]キーを押したのと同じことです。



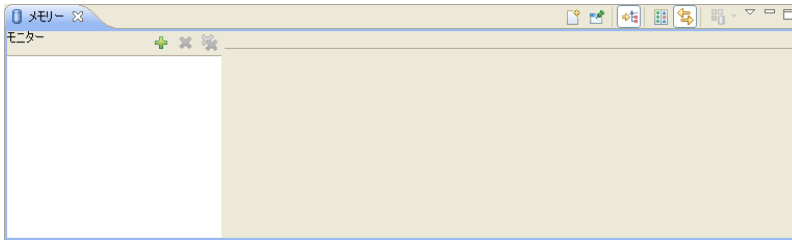
これは[レジスター]ビューです。CPUのレジスタの内容を表示します。ここでレジスタデータの書き換えも行えます。



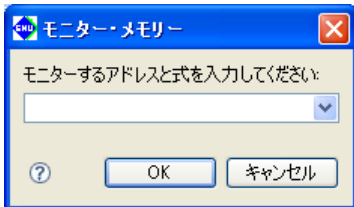
これは[メモリ]ビューです。ターゲットメモリの内容を表示します。ここで、メモリデータの書き換えも行えます。

サンプルプログラムは、先ほどの実行でint変数i(0x00000000~0x00000003番地)をインクリメントしました。[メモリー]ビューで変数iの内容を見てみましょう。

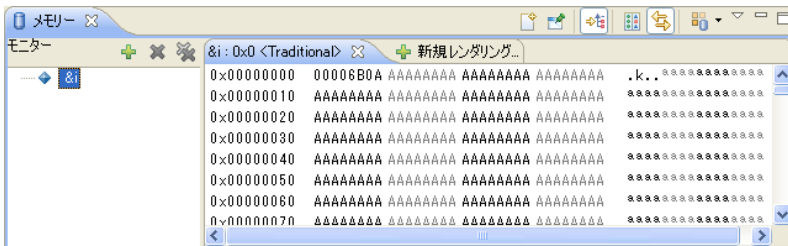
操作55: [ウィンドウ]メニューから[ビューの表示]>[メモリー]を選択します。
[メモリー]ビューが表示されます。



操作56: [メモリー・モニターの追加]ボタンをクリックすると、[モニター・メモリー]ダイアログボックスが表示されます。

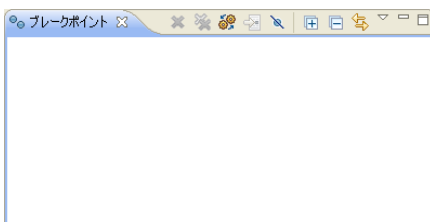


操作57: [モニター・メモリー]ダイアログボックスのテキストボックスに&iまたは0と入力し、[OK]をクリックします。



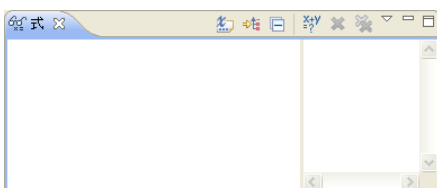
変数i(0番地)からメモリーの内容が表示されました。これによるとiは00006B0A(=27402)までカウントされたことを示しています。

操作58: [ウィンドウ]メニューから[ビューの表示]>[ブレークポイント]を選択します。



これは[ブレークポイント]ビューです。指定位置でプログラムの実行を停止させるソフトウェアPCブレークポイントの管理に使用します。

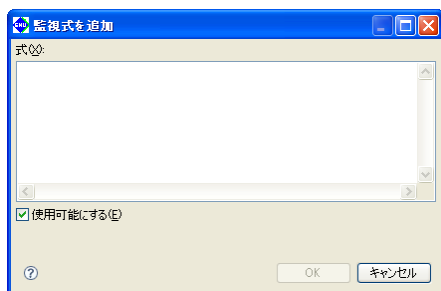
操作59: [ウィンドウ]メニューから[ビューの表示]>[式]を選択します。



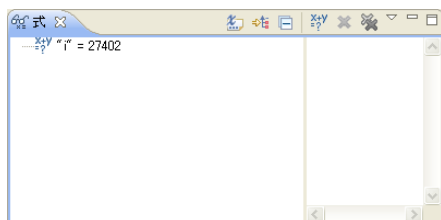
3 ソフトウェア開発手順

これは[式]ビューです。グローバル変数などの値をモニタするのに使用します。たとえば、先ほど[メモリー]ビューで確認した変数*i* (*i*はグローバル変数です)をこのウィンドウでモニタできるようにしてみましょう。

操作60: [式]ビューの[新規監視式を追加]ボタンをクリックします。
[監視式を追加]ダイアログボックスが表示されます。

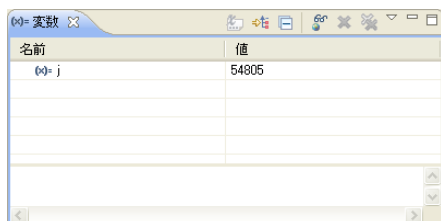


操作61: [監視式を追加]ダイアログボックスのテキストボックスに*i*と入力し、[OK]をクリックします。



ウィンドウのリストボックスに*i*が現れ、その内容が10進数(デフォルトの表示形式)で表示されます。

操作62: [ウィンドウ]メニューから[ビューの表示]>[変数]を選択します。



これは[変数]ビューです。現在の関数内で定義されているローカル変数を表示します。現在のPCアドレスはmain () 関数内にありますので、この関数で定義されている変数*j*のシンボルとその内容が表示されています。

以上、gdbのウィンドウを見てきました。各ビューには、表示以外にも機能が備わっています。それらの詳細は、"10.4 ウィンドウ"に説明されています。

また、プログラムの実行に戻ります。

先ほどは、プログラムを連続実行させ、強制ブレーク機能で停止させました。

今度は、停止する位置をあらかじめ指定してからプログラムを実行させてみます。

●ブレークポイントを指定するには

操作63: [ソース]エディタ内にソース行番号が表示されますので、16という数字の前にマウスポインタを移動し、そこでダブルクリックします。

ソース行16の先頭にが表示されれば、そこがソフトウェアPCブレークポイントに設定されました。ソース行16以外にが付いた場合は、そこをもう一度ダブルクリックすると元に戻りますので、やり直してください。

```

12
13     i = 0;
14     for (j=0 ; ; j++)
15     {
16         sub(j);
17     }
18

```



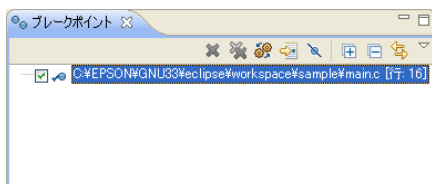
操作64: [再開]ボタンをクリックします。

操作51とは違い、今度はブレークポイントに設定した行でプログラムが停止したはずです。何度か[再開]ボタンを押してみてください。毎回同じところで停止します。

[変数]ビューがまだ開いていれば、ブレークするたびに表示される変数jがインクリメントされているのが確認できます。

同様に、[式]ビューに表示されているグローバル変数iは、実行2回ごとにインクリメントされることが確認でき、プログラムが期待通りの動きをしていることが分かります。

操作65: [ウィンドウ]メニューから[ビューの表示]>[ブレークポイント]を選択します。
[ブレークポイント]ビューが表示されます。



先ほどは何も表示されていなかった[ブレークポイント]ビューに、今回設定したブレークポイントの情報が表示されています。情報の先頭にあるチェックはブレークポイントが現在有効であることを示しています。選択を解除するとブレークポイントは一時的に無効となり、次にプログラムを実行すると、先ほどの位置では停止しなくなります。チェックボックスを選択すれば、また有効な状態に戻ります。

操作66: [ソース]エディタ上に表示されているをダブルクリックして消します。

これで、ブレークポイントが解除されます。

このほかにも、プログラムの実行1回に限り有効なテンポラリブレークや、指定アドレスをアクセスした場合に停止させるデータブレークが使用できますが、ここでは省略します。ブレーク機能の詳細については、「10.6.5 ブレーク機能」を参照してください。

プログラムの動きに問題がある場合、もう少し細かく動作確認する必要があります。
最後にステップ動作を見てみましょう。

●ステップ実行するには



操作67: [デバッグ]ビューの[ステップイン]ボタンをクリックしてください。

[ソース]エディタ内の緑で強調表示されたソース行(現在のPCアドレスがある行)を実行し、強調表示が次に実行するソース行に移ります。

操作67を繰り返すことで、1ステップずつ順次実行します。プログラムに問題がなければ、レジスタの表示などがステップごとに正しく変わっていることが分かります。

[ステップイン]ボタンではソース行単位に実行します。命令(ニーモニック)単位に実行させるには、[命令ステップ・モード]ボタンをクリックして、[命令ステップ・モード]ボタンを押し込んだ状態で[ステップイン]ボタンをクリックして使用します。



[命令ステップ・モード]ボタン



操作68: [デバッグ]ビューの[ステップ・オーバー]ボタンをクリックしてください。

操作68を繰り返して、[ステップイン]ボタンとの違いを[ソース]エディタで確認してください。

関数`sub()`をスキップしましたが、変数`i`の内容が更新され、関数内は連続実行されたことが分かります。

Next動作は基本的にはステップ動作と同じですが、関数やサブルーチンをスキップ(1ステップとして実行)します。デバッグの終わったサブルーチンなどは1ステップずつ実行させる必要もないため、Next実行が便利です。

以上、デバッガの基本的な操作を見てきましたが、[コンソール]ビューにキーボードからコマンドを入力して、さらに高度なデバッグを行うことができます。その詳細については"10 デバッグ"を参照してください。

デバッグは次の方法で終了します。

●デバッグを終了するには

操作69: [実行]メニューから[終了]を選択します。

デバッグが終了します。IDEのウィンドウに戻りたい場合は、ウィンドウ右上にある[「GNU33」パースペクティブ]ボタンをクリックしてください。

ここで行ったシミュレータモード以外にも、インサーキットデバッグS5U1C33000HまたはS5U1C33001Hにターゲットボードを接続して、あるいはデバッグモニタ(S5U1C30M2S)を実装したターゲットボードとS5U1C330M1D1を使用してデバッグを行うこともできます。それぞれのモードでのデバッグ方法については"3.6 デバッグ環境"を参照してください。

最後に、IDEを終了させます。

●IDEを終了するには

操作70: [ファイル]メニューから[終了]を選択します。

3.3.7 ROMデータの作成

ターゲットボードに搭載する外部ROM用のデータや内蔵ROMのマスクデータを作成するにはHEXファイルが必要です。このため、プログラムが完成後は、elf形式オブジェクトファイルをもとローラS3形式のHEXファイルに変換し、また、外部ROM用と内蔵ROM用にファイルを分ける作業を行います。この作業には、gnu標準のオブジェクトファイル形式変換ユーティリティ **objcopy.exe** を使用します。**objcopy.exe**の詳細については、gnuのユーティリティに関する資料もしくは11.4項を参照してください。また、オプションや出力ファイル名の指定は行えませんが、IDEの[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー上でも、コンテキストメニューからモトローラS3形式への変換が行えます。

●IDE上でのHEXファイル作成

操作71: [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー上のelf形式オブジェクトファイルを右クリックしてコンテキストメニューを表示させ、[オブジェクトファイル変換] > [S レコードファイル生成]を選択します。



同じファイル名で拡張子が".sa"のモトローラS3形式ファイルが生成されます。

●objcopyでHEXファイルを作成するには

コマンドプロンプトウィンドウを開き、次のコマンドラインで**objcopy**を実行します。

```
C:\¥EPSON¥gnu33>objcopy -I elf32-little -O srec --srec-forceS3 InputFile(.elf)
OutputFile(.sa)
```

-O srec: 出力ファイルのレコードフォーマットをモトローラ形式に指定します。
 -R .gbss: .gbssという名称のセクションを出力ファイルに含めないことを指定するオプションです。
 --srec-forceS3: モトローラS3形式の出力を指定します。
 InputFile: 変換するelf形式のオブジェクトファイルを指定します。
 OutputFile: 出力されるHEXファイルの名称を指定します。

例: input.elfからすべてのHEXデータを抜き出し、output.saに書き出します。

```
C:\¥EPSON¥gnu33>objcopy -O srec input.elf output.sa
```

●提出用マスクデータの作成

ROMを内蔵する機種のプログラムが完成後は、内蔵ROM用のマスクデータをセイコーエプソンに提出していただくことになります。

objcopy.exeによりモトローラS3フォーマットのHEXファイルを作成後、**moto2ff.exe**を使用してHEXファイル内の空きアドレスを0xffのデータで埋めていただく必要があります。**moto2ff.exe**の詳細は、その他のツールの章を参照してください。

moto2ff.exeによってマスクデータを作成する方法を以下に示します。

コマンドプロンプトウィンドウを開き、次のようにコマンドラインで**moto2ff**を実行します。

```
C:\¥EPSON¥gnu33>moto2ff StartAddress BlockSize InputFile
```

上記形式で、InputFile内のStartAddressからBlockSizeバイトのデータが出力ファイル(入力ファイル名+拡張子.saf)に書き出されます。このとき、空きアドレスには0xffが埋め込まれます。

例: モトローラS3形式のinput.sa内の0x600000番地から0x100000バイト分のデータをinput.safに出力します。

```
C:\¥EPSON¥gnu33>moto2ff 600000 100000 input.sa
```

input.sa内のアドレス0x600000から0x6fffffの範囲の全データがファイルに出力されます。この範囲内に空きアドレスが存在した場合、0xffのデータで空きは埋められます。

上記の手順で内蔵ROM用のマスクROMファイル作成後、このファイルを使用して最終動作確認を行ってください。

動作確認終了後、マスクROMファイルをセイコーエプソンが指定する名称にリネームして提出してください。

3.4 チュートリアル2(ユーザmakeファイルの使用)

ここでは、ユーザのmakeファイルを使用する例として、旧gnu33ツール用に作成されたToppers33のサンプルプログラムとmakeファイルなどを使用してIDE上でビルドし、デバッガを起動するまでの手順を見ていきます。IDEの基本的な操作方法等については、チュートリアル1を参照してください。

使用するサンプルのディレクトリ

Toppers33およびgnu33ツールは、以下のディレクトリにインストールされているものとして説明を行います。

Toppers33インストールディレクトリ: C:\¥EPSON¥gnu33¥tool¥tps33g

gnu33インストールディレクトリ: C:\¥EPSON¥gnu33

注: このサンプルプログラムはICD Ver.3以降のデバッグ環境に対応して作成されています。本チュートリアルでは、ICD Ver.3とDMT33209のデバッグ環境で実行させるものとして説明を行います。ただし、このチュートリアルの目的はデバッガを起動するところまでとなりますので、実際にプログラムを実行させない場合、このデバッグ環境は特に必要ではありません。
その他に対応しているデモンストレーションボードの種類、ICDとの接続やボードの設定については、C:\¥EPSON¥gnu33¥tool¥tps33gディレクトリ内のreadme_ja_slc33.txt(日本語)またはreadme_slc33.txt(英語)を参照してください。

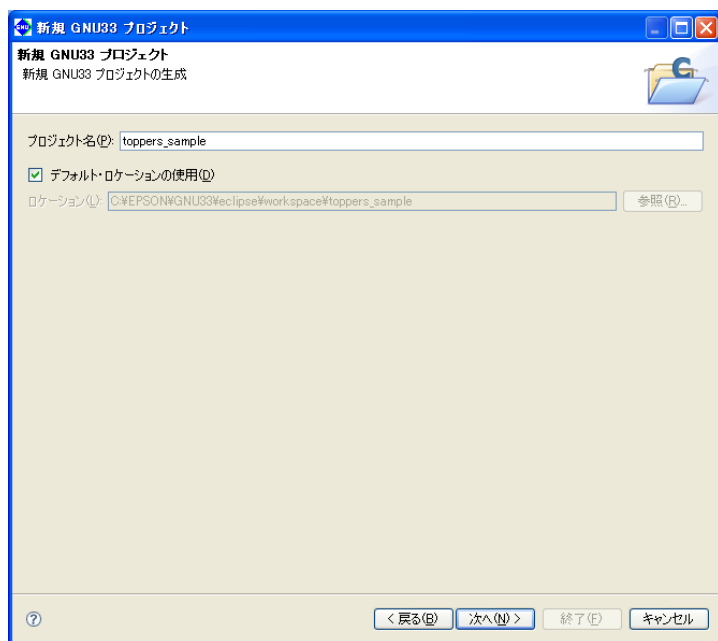
3.4.1 プロジェクトの作成

まず、IDEで新規プロジェクトを作成します。

操作1: IDEを起動します。

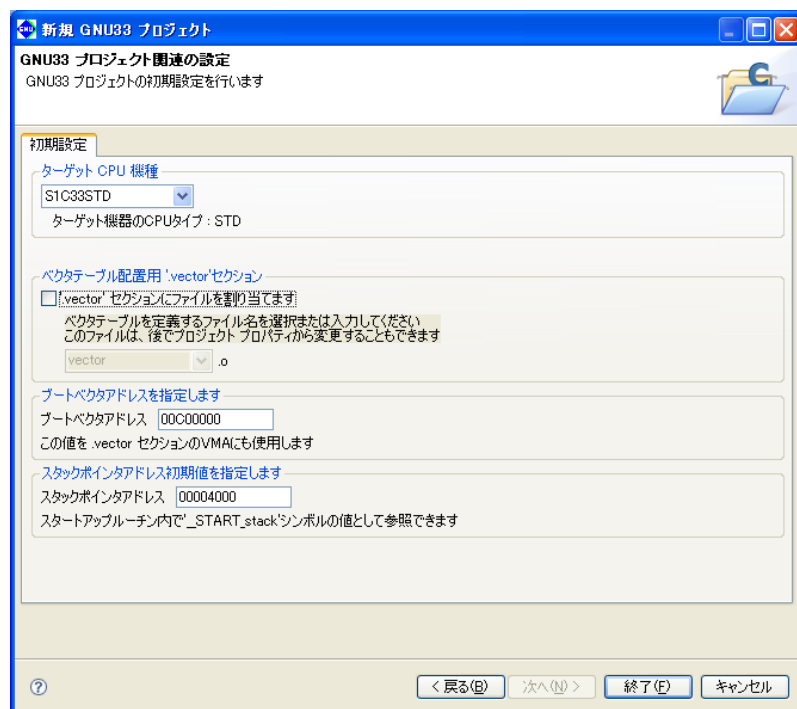
操作2: [ファイル]メニューから[新規] > [新規 GNU33 プロジェクト]を選択し、[新規 GNU33 プロジェクト]ウィザードを開始させます。

操作3: [プロジェクト名:]にプロジェクト名"toppers_sample"を入力します。



このチュートリアルでは、プロジェクトフォルダをワークスペース(デフォルト)に作成しますので、[デフォルト・ロケーションの使用]チェックボックスは選択したままにしておきます。

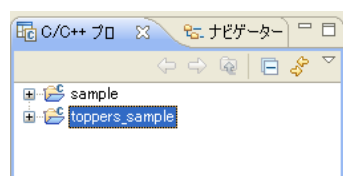
操作4: [次へ>]ボタンをクリックします。



このチュートリアルでは、**IDE**が生成するmakeファイルやリンカスクリプトファイルなどを使用しますので、ターゲットCPUと.vectorセクションの選択は必要ありません。

操作5: ['.vector'セクションにファイルを割り当てます]のチェックは外しておきます。

操作6: [終了]ボタンをクリックして、プロジェクトを作成します。

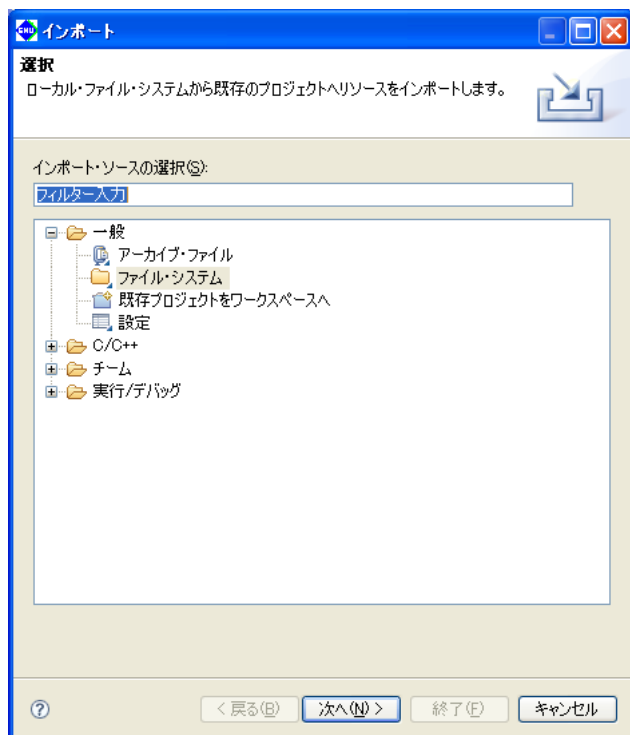


3.4.2 ソースファイルのインポート

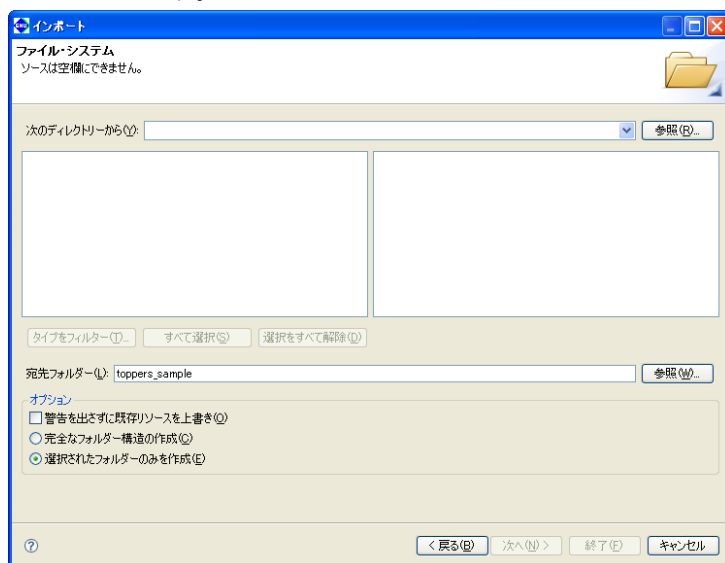
プロジェクトが作成できましたら、次にソースファイルをインポートします。また、同じディレクトリにあるmakeファイルもここでインポートしておくことにします。

操作7: [ナビゲーター]ビューでプロジェクト"toppers_sample"を選択して、[ファイル]メニューから[インポート...]を選択します。

[インポート]ウィザードが起動します。



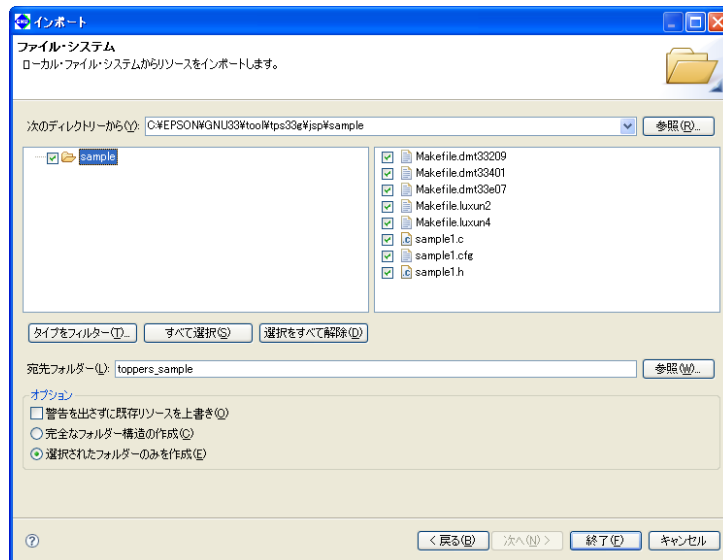
操作8: 表示されている一覧の中から[一般] > [ファイル・システム]を選択し、[次へ>]ボタンをクリックします。



操作9: [次のディレクトリから:]の[参照...]ボタンで、インポートするファイルがあるC:\¥EPSON¥gnu33¥tool¥tps33g¥jsp¥sampleディレクトリを選択します。

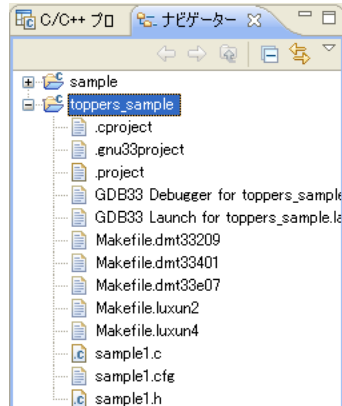
左のリストボックスに選択したディレクトリが、右のリストボックスにディレクトリ内のファイルの一覧が表示されます。

操作10: 左のリストボックスに表示されているsampleフォルダのチェックボックスを選択します。
右のリスト内のファイルがすべて選択されます。



操作11: [終了]ボタンをクリックします。

[ナビゲーター]ビューで、プロジェクトに追加されたファイルが確認できます。



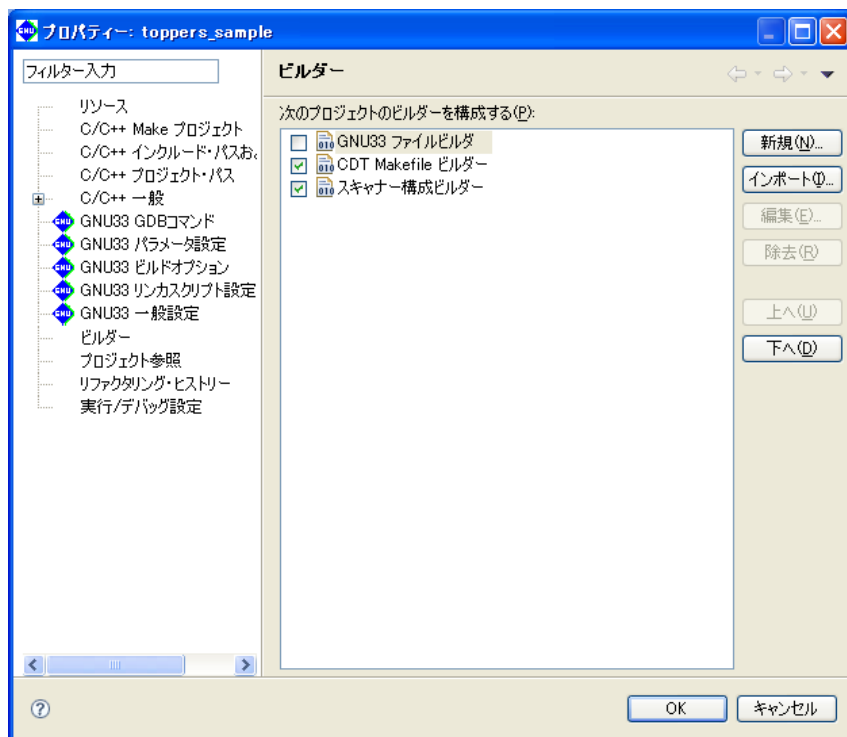
3.4.3 GNU33ファイルビルダの無効化

IDEは、makeファイル、リンカスクリプトファイル、パラメータファイル、デバッグコマンドファイルを自動生成してビルドやデバッグの起動に使用するように初期設定されています。このチュートリアルでは別途用意したファイルを使用するため、IDEがこれらのファイルを生成して使用しないように設定を変更します。

まず、ここではこれらのファイルが自動生成されないように、ファイルビルダを無効に設定する方法を説明します。

操作12: [ナビゲーター]ビューまたは[C/C++ プロジェクト]ビューでプロジェクト"toppers_sample"を選択後、[プロジェクト]メニューまたはコンテキストメニューから[プロパティ]を選択して[プロパティ]ダイアログボックスを表示させます。

操作13: プロパティの一覧から[ビルダー]を選択し、[GNU33ファイルビルダ]のチェックを外します。



操作14: [OK]ボタンをクリックします。

この設定を行わない場合でも、ユーザが作成したmakeファイル、リンカスクリプトファイル、パラメータファイル、デバッグコマンドファイルを使用することは可能です。ただし、ビルドのたびに不要なファイルが生成されます。また、自動生成されるファイルと同じ名称を使用している場合は、上書きされてしまいます。

なお、これらのファイルの中で1つでもIDEに自動生成させる場合は、ファイルビルダを無効にすることはできません。

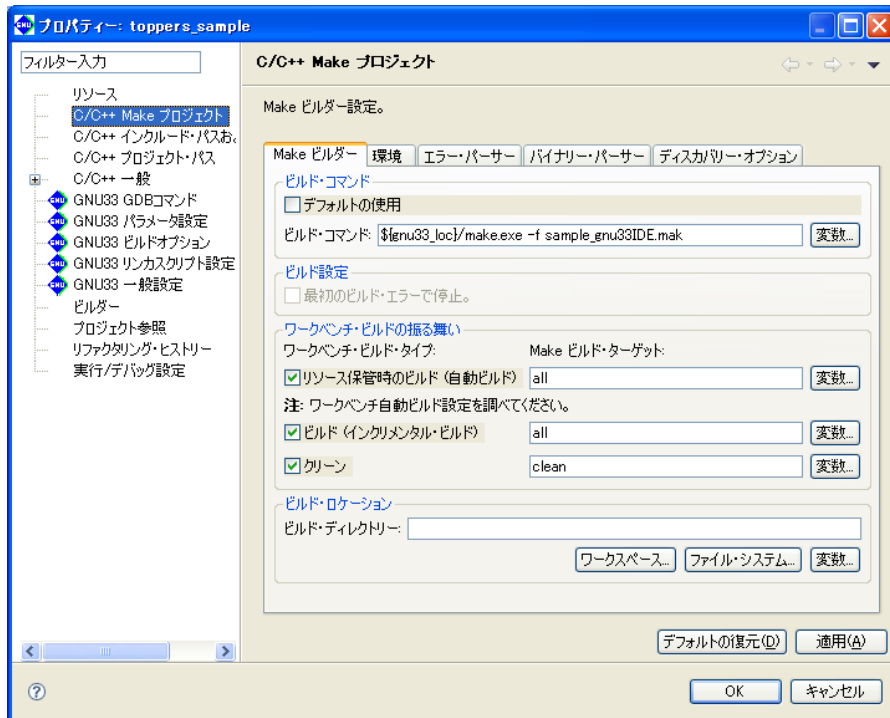
3.4.4 makeファイルの設定と修正

●ユーザmakeファイルを指定するには

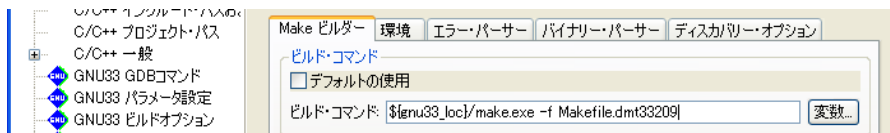
次に、別途用意したmakeファイルでビルドが行われるように設定します。ここでは、先にプロジェクトにインポートしたMakefile.dmt33209を使用します。

操作15: [ナビゲーター]ビューまたは[C/C++ プロジェクト]ビューでプロジェクト"toppers_sample"を選択後、[プロジェクト]メニューまたはコンテキストメニューから[プロパティ]を選択して[プロパティ]ダイアログを表示させます。

操作16: プロパティの一覧から[C/C++ Make プロジェクト]を選択し、[Make ビルダー]タブのページを表示させます。



操作17: [ビルド・コマンド:]に設定されているmakeファイル名"toppers_sample_gnu33IDE.mak"を"Makefile.dmt33209"に変更します。



Makefile.dmt33209の場合は必要ありませんが、ユーザが作成したmakeファイル内のターゲット名がall(ビルド)とclean(クリーン)以外の場合は、以下の設定も変更する必要があります。

[ビルド(インクリメンタル・ビルド)]

ビルド実行時に呼び出すmakeファイル内のターゲットを指定します。

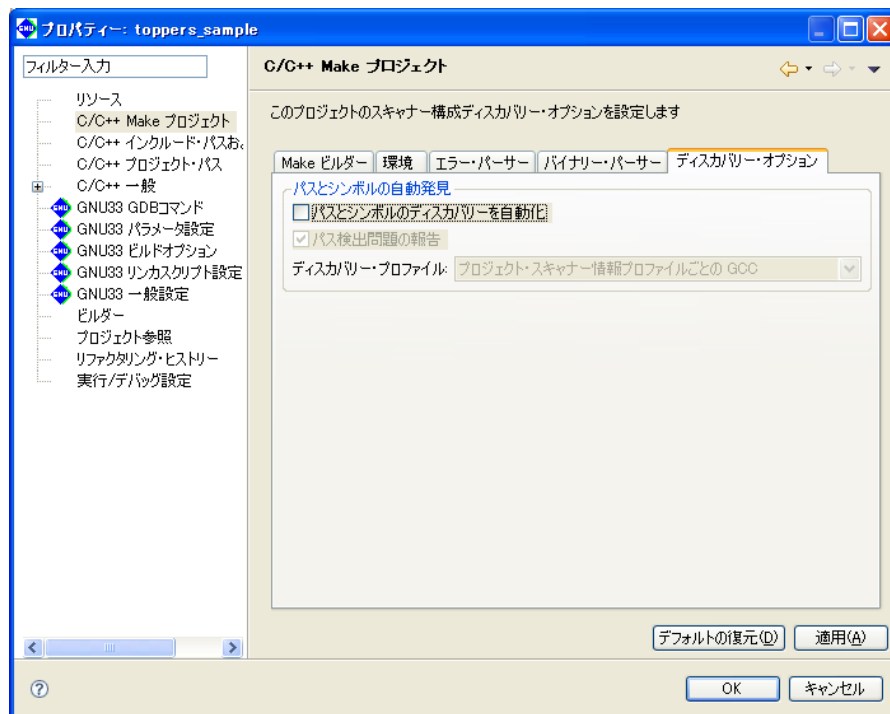
[クリーン]

クリーン(生成ファイルのクリア)実行時に呼び出すmakeファイル内のターゲットを指定します。

[リソース保管時のビルド(自動ビルド)]は、IDEでは使用しません。

操作18: [ディスカバリー・オプション]タブのページを表示させます。

操作19: [パスとシンボルのディスカバリーを自動化]のチェックを外します。



操作20: [OK]ボタンをクリックします。

●Makeファイルの内容修正

旧バージョン用に作成されているmakeファイルを使用する場合、記述されたパスの変更が必要なことがあります。Makefile.dmt33209もソースディレクトリが相対パスで記述されていますので、プロジェクトディレクトリからも参照できるようにcygwin形式のパスに変更します。内容の修正はIDEのエディタで行えます。

操作21: [ナビゲーター]ビューに表示されているファイル名Makefile.dmt33209をダブルクリックしてエディタで開きます。

操作22: ソースディレクトリを示すマクロ"SRCDIR"の定義内容を次のように変更します。

変更前 SRCDIR= ..

変更後 SRCDIR=/cygdrive/c/EPSON/gnu33/tool/tps33g/jsp

※ 新規プロジェクト作成時にプロジェクトフォルダをC:¥EPSON¥gnu33¥tool¥tps33g¥jsp¥sampleに設定した場合、この修正は不要です。

tps33gを他のディレクトリにインストールした場合は、そのディレクトリに変更してください。

操作23: ツールバーの[保管]ボタンをクリック、または[ファイル]メニューから[保管]を選択してファイルを保存します。

操作24: エディタタブのクローズ(✕)ボタンでファイルを閉じます。

次の操作25～28はgnu33ツールをC:¥EPSON¥gnu33ディレクトリ以外にインストールした場合に必要です。C:¥EPSON¥gnu33ディレクトリにインストールした場合、操作の必要はありません。

Makefile.dmt33209等、Toppersのサンプルで提供されるmakeファイルには、includeディレクトリによりMakefile.configというファイルがインクルードされています。

```
include $(SRCDIR)/config/$(CPU)-$(TOOL)/Makefile.config
```

このファイルの中にgnu33ツールディレクトリを定義したマクロが記述されていますので、別のディレクトリにインストールした場合は変更します。

操作25: [ファイル]メニューから[ファイルを開く...]を選択し、ファイル選択ダイアログボックスから¥tps33g¥jsp¥config¥s1c33-gnu33¥dmt33209ディレクトリ内のMakefile.configを選択してエディタで開きます。

操作26: マクロ"TOPDIR"の定義内容を、gnu33ツールをインストールしたディレクトリに変更します。
 変更前 TOPDIR = /cygdrive/c/EPSON/gnu33
 変更例 TOPDIR = /cygdrive/d/EPSON/gnu33
 (D:¥EPSON¥gnu33にインストールした場合)

操作27: ツールバーの[保管]ボタンをクリック、または[ファイル]メニューから[保管]を選択してファイルを保存します。

操作28: エディタタブのクローズ(✕) ボタンでファイルを閉じます。

本チュートリアルでは不要ですが、リンカスクリプトファイルをサンプル以外のものに入れ替えたり、内容を修正したりする場合のために、makeファイル内の変更箇所と、サンプルのリンカスクリプトファイルを記載しておきます。

- makeファイル内でリンカスクリプトファイルを指定するマクロの定義箇所
 ファイル: C:¥EPSON¥gnu33¥tool¥tps33g¥jsp¥config¥s1c33-gnu33¥dmt33209ディレクトリ内のMakefile.config
 マクロ定義: LDSCRIPT = \$(CPU)/\$(SYS)/dmt33209.ld
 他のmakeファイルを使用する場合は、LDSCRIPTの定義を変更します。
- Makefile.dmt33209で使用しているリンカスクリプトファイル
 C:¥EPSON¥gnu33¥tool¥tps33g¥jsp¥config¥s1c33¥dmt33209¥dmt33209.ld
 リンカスクリプトの内容を変更する場合は、このファイルを開いて修正します。
 (本チュートリアルはDMT33209を使用しますが、その他のボードの場合、¥dmt33209の代わりに各ボードに対応するディレクトリが用意されています。)
 修正は操作25と同様に、ファイルを[ファイルを開く...]によって開いて行います。

3.4.5 ビルド

makeファイルの設定が終われば、通常の方法でビルドを実行できます。ビルドオプションの設定およびリンカスクリプトの編集は必要ありません。

操作29: [ナビゲーター]ビューまたは[C/C++ プロジェクト]ビュー内のプロジェクト名"toppers_sample"を選択します。

操作30: [プロジェクト]メニューから[プロジェクトのビルド]を選択します。

この操作により、指定のmakeファイルでmake.exeが実行され、実行形式のオブジェクトファイルsample1.elfが生成されます。(自動生成したmakeファイルではないため、オブジェクトファイルはプロジェクト名と同じにはなりません。)

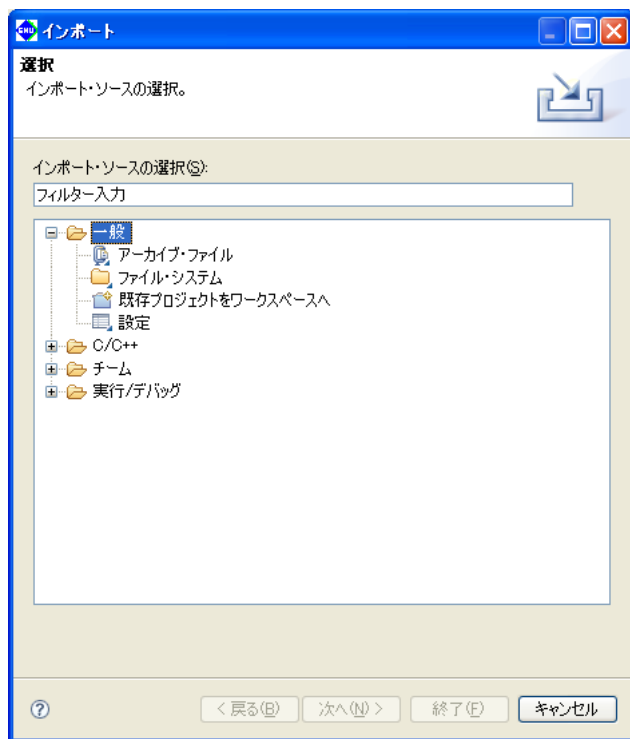
3.4.6 デバッガ起動用ファイルのインポート

デバッガの起動時に使用するコマンドファイルとパラメータファイルについても、ユーザが用意したファイルを使用することができます。ここでは、C:\¥EPSON¥gnu33¥tool¥tps33g¥jsp¥config¥s1c33-gnu33¥dmt33209ディレクトリに用意されているicdv3.cmd(コマンドファイル)とdmt33209.par(パラメータファイル)を使用します。(本チュートリアルはDMT33209を使用しますが、その他のボードの場合、¥dmt33209の代わりに各ボードに対応するディレクトリとファイルが用意されています。)

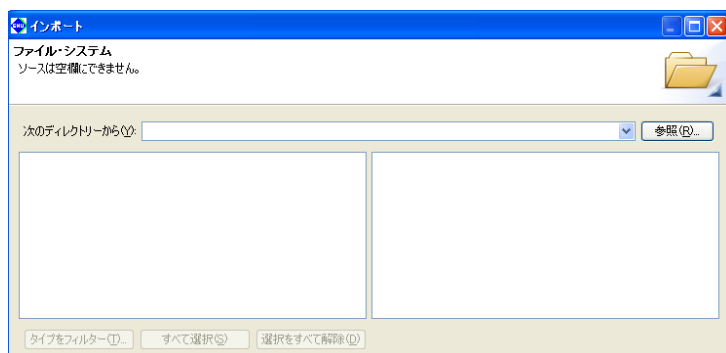
まず、これらのファイルをプロジェクトにインポートします。

操作31: [ナビゲーター]ビューでプロジェクト"toppers_sample"を選択して、[ファイル]メニューから[インポート...]を選択します。

[インポート]ウィザードが起動します。



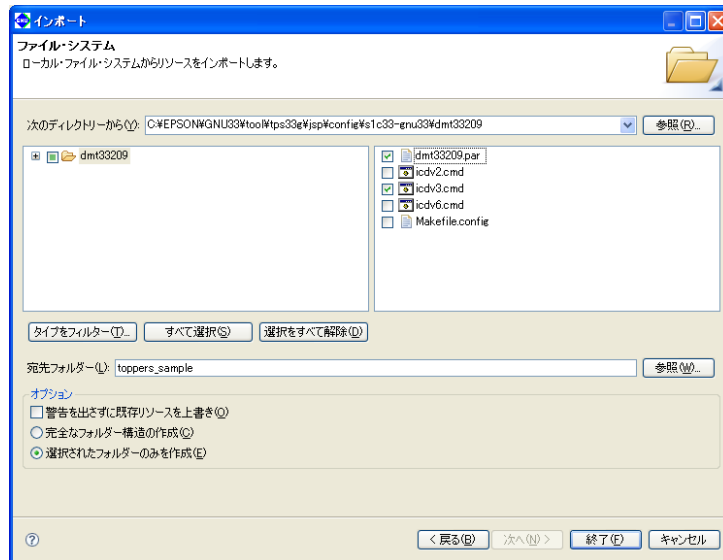
操作32: 表示されている一覧の中から[一般] > [ファイル・システム]を選択し、[次へ>]ボタンをクリックします。



操作33: [次のディレクトリーから:]の[参照...]ボタンで、インポートするファイルがあるC:\¥EPSON¥gnu33¥tool¥tps33g¥jisp¥config¥s1c33-gnu33¥dmt33209ディレクトリーを選択します。

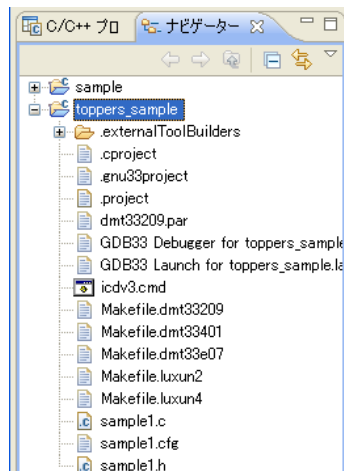
左のリストボックスに選択したディレクトリが、右のリストボックスにディレクトリ内のファイルの一覧が表示されます。

操作34: 右のリストボックスに表示されている"dmt33209.par"と"icdv3.cmd"のチェックボックスを選択します。



操作35: [終了]ボタンをクリックします。

[ナビゲーター]ビューで、プロジェクトに追加されたファイルが確認できます。



3.4.7 デバッガ起動用ファイルの修正

旧バージョン用に作成されているコマンドファイルとパラメータファイルを使用する場合、記述されたパスの変更などが必要ことがあります。icdv3.cmdの場合も、使用するフラッシュプログラムを相對パスで指定していますので、プロジェクトディレクトリからも参照できるようにcygwin形式のパスに変更します。

操作36: [ナビゲーター]ビューに表示されているファイル名"icdv3.cmd"をエディタにドラッグ&ドロップしてエディタで開きます。

操作37: fileコマンドを次のように変更します。

```
変更前 file ../config/s1c33-gnu33/dmt33209/fls/am29f800.elf
変更後 file /cygdrive/c/EPSON/gnu33/tool/tps33g/jsp/config/s1c33-gnu33/
      dmt33209/fls/am29f800.elf
```

※ 新規プロジェクト作成時にプロジェクトフォルダをC:¥EPSON¥gnu33¥tool¥tps33g¥jsp¥sampleに設定した場合、この修正は不要です。

tps33gを他のディレクトリにインストールした場合は、そのディレクトリに変更してください。

操作38: その他、コマンドの追加が必要であれば、記述します。(本チュートリアルでは不要)

操作39: ツールバーの[保管]ボタンをクリック、または[ファイル]メニューから[保管]を選択してファイルを保存します。

操作40: エディタタブのクローズ()ボタンでファイルを閉じます。

本コマンドファイルは、同じディレクトリにあるパラメータファイルdmt33209.parを指定してc33rpfコマンド(メモリマップの設定)を実行します。本チュートリアルでは不要ですが、パラメータファイルをサンプル以外のものに入れ替える場合は、"c33 rpf dmt33209.par"のコマンド行を変更します。また、サンプルのパラメータファイルでマップの変更が必要な場合は、dmt33209.parを開いて編集してください。

3.4.8 デバッガの起動

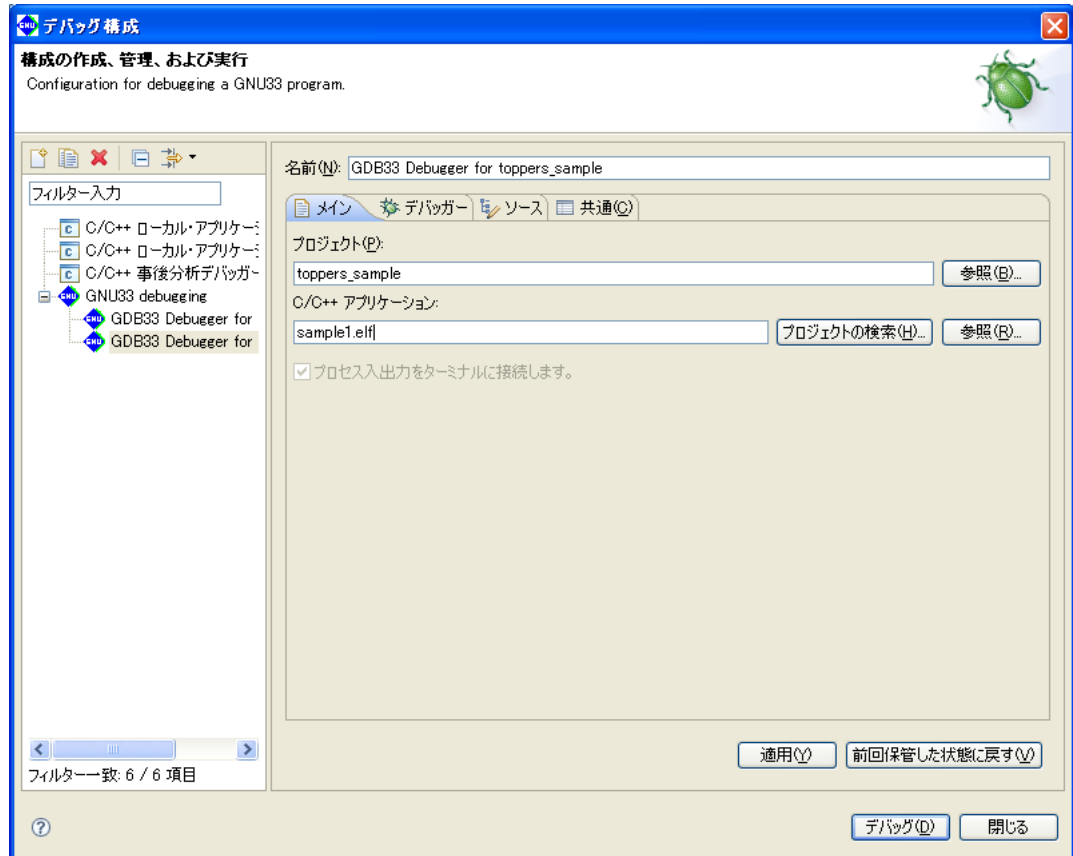
チュートリアル2の最後として、用意したコマンドファイルをデバッガ起動時に実行させるための設定方法を説明します。

その後、実際にデバッガを起動してサンプルプログラムを実行させる場合は、先にICD(S5U1C33001H)とデモンストレーションボードを接続し、電源をONしてデバッグを開始できる状態にしておきます。ICDとの接続やボードの設定、動作などについては、C:\¥EPSON¥gnu33¥tool¥tps33gディレクトリ内のreadme_ja_s1c33.txt(日本語)またはreadme_s1c33.txt(英語)を参照してください。

操作41: [実行]メニューから[デバッグの構成...]を選択します。
[デバッグ構成]ダイアログボックスが表示されます。

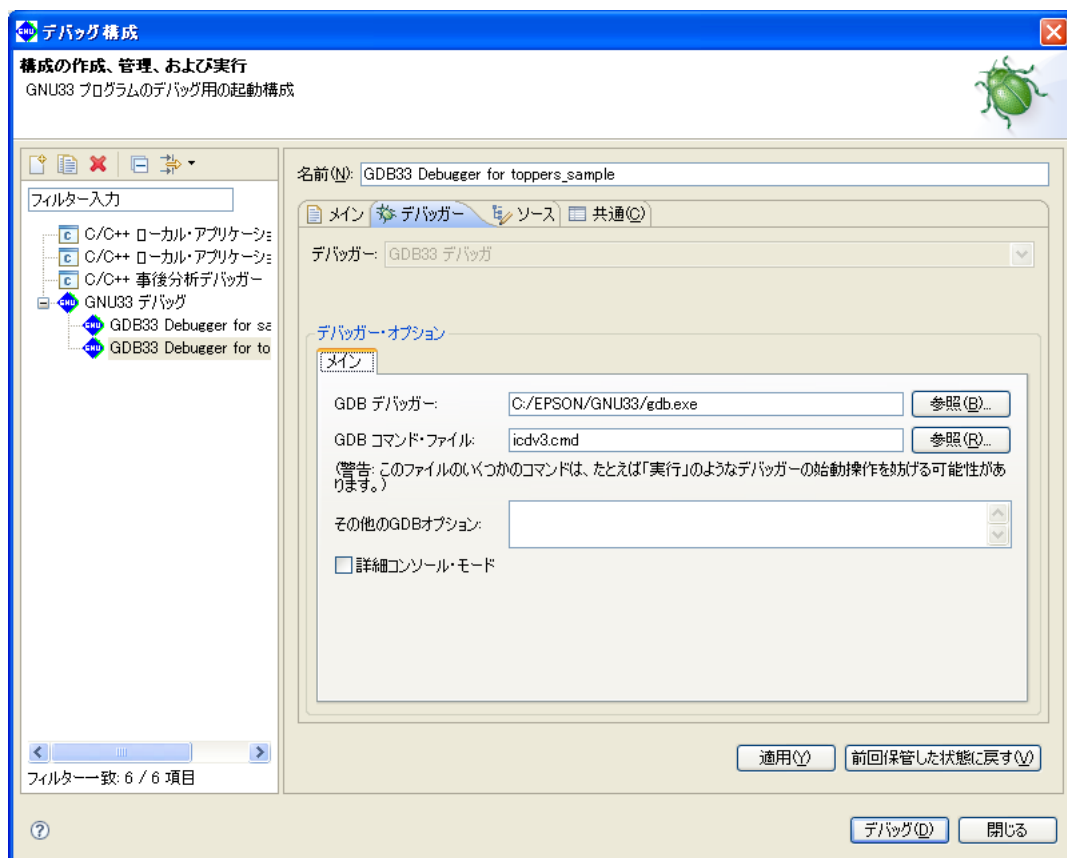
操作42: リストから[GDB33 Debugger for topplers_sample]を選択し、[メイン]タブのページを表示させます。

操作43: [C/C++ アプリケーション:]にあるelfファイルを「sample1.elf」に変更します。



3 ソフトウェア開発手順

操作44: [デバッガー]タブを表示させて、[GDB コマンド・ファイル:]にあるコマンドファイルを `icdv3.cmd` に変更します。



操作45: [適用]ボタンをクリックして、変更を確定します。

この後、デバッガを起動すると、`icdv3.cmd`が実行されます。

操作46: ここで終了する場合は[閉じる]ボタンをクリックします。

実際にデバッガを起動する場合は、[デバッグ]ボタンをクリックします。

デバッガの基本操作については、チュートリアル1を参照してください。

3.5 チュートリアル3(IDEプロジェクトのインポート)

S1C33 FamilyのアプリケーションをすでにIDEで開発している場合、そのプロジェクトを他のPC上のIDEにインポートして開発の継続や改訂作業を行ったり、そのプロジェクトを元に別のアプリケーションを開発したりすることができます。ここでは、プロジェクトをインポートする方法を説明します。それ以外の操作についてはチュートリアル1と2を参照してください。

使用するサンプルプロジェクトフォルダ

C:\¥EPSON¥gnu33¥sample_ide¥std¥simulator¥tst

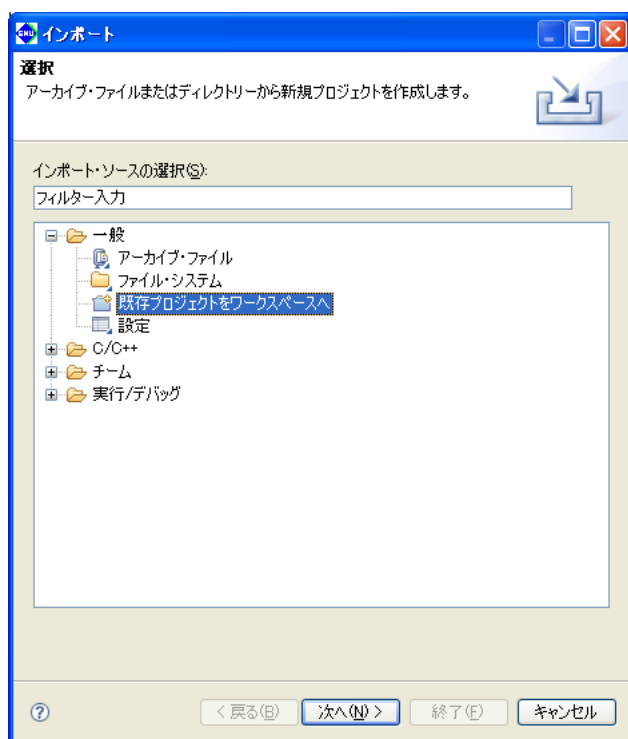
●プロジェクトをインポートするには

インポートするプロジェクトは、HDD上にコピーされているものとして始めます。

操作1: IDEを起動します。

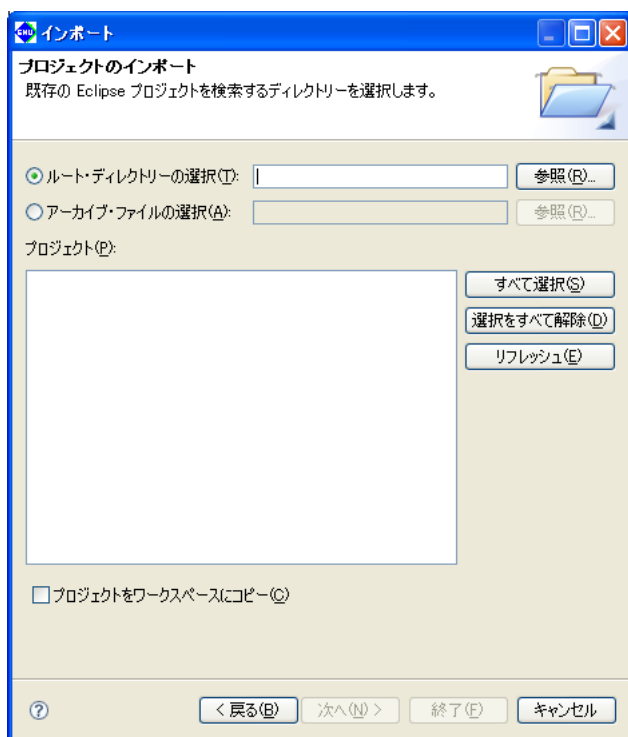
操作2: [ファイル]メニューから[インポート...]を選択します。

[インポート]ウィザードが起動します。

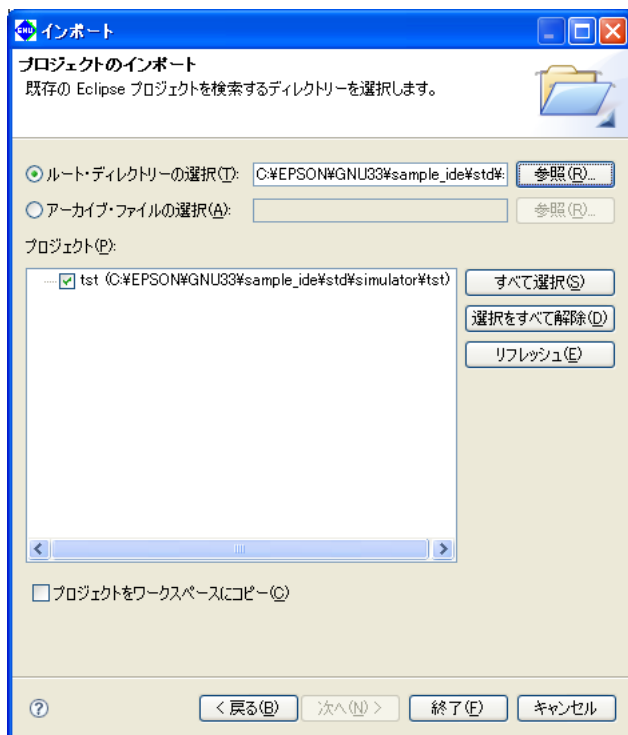


操作3: 表示されている一覧の中から[既存プロジェクトをワークスペースへ]を選択し、[次へ>]ボタンをクリックします。

3 ソフトウェア開発手順



操作4: [ルート・ディレクトリの選択]の[参照...]ボタンで、インポートするプロジェクトのフォルダC:\¥EPSON¥gnu33¥sample_id¥std¥simulator¥tstを選択します。



操作5: [プロジェクトをワークスペースにコピー]チェックボックスを選択します。

これにより、プロジェクトのコピーがワークスペースディレクトリに作成され、オリジナルが変更されることはありません。

※ ワークスペースディレクトリには、プロジェクトディレクトリ(.projectファイルが含まれるディ

レクトリ)を指定しないでください。プロジェクトインポート(プロジェクトをワークスペースにコピーがONのとき)に失敗することがあります。

現在のワークスペースディレクトリは、[ファイル]-[ワークスペースの切り替え]-[その他]よりワークスペース・ランチャーダイアログで確認できます。

操作6: [終了]ボタンをクリックします。

これで**IDE**にプロジェクトとしてインポートされました。

プロジェクトのインポートの場合、プロジェクトフォルダはワークスペースにコピーされません。インポートで選択した位置のまま作業が行われます。オリジナルのプロジェクトフォルダを指定した場合、内容が変更されてしまいますので、コピーを作成してインポートしてください。

あるいは、インポートした後で、次のようにコピーを作成することもできます。

●プロジェクトのコピーを作成するには

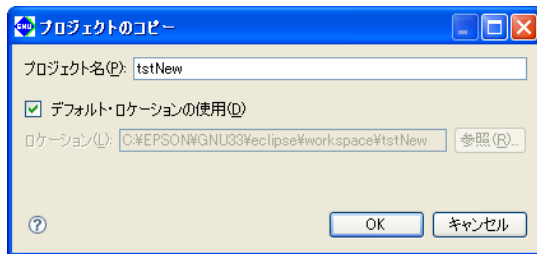
操作7: [C/C++ プロジェクト]ビューでインポートしたプロジェクト"tst"を選択します。

操作8: [編集]メニューから[コピー](Ctrl+C)を選択し、続けて[編集]メニューから[貼り付け](Ctrl+V)を選択します。

[プロジェクトのコピー]ダイアログボックスが表示されます。

操作9: [プロジェクト名:]にプロジェクト名(ここでは"tstNew")を入力します。

操作10: [デフォルト・ロケーションの使用]チェックボックスを選択します。



このチェックボックスを選択することにより、ワークスペース内にtstNewというプロジェクトとしてコピーされます。

ワークスペース以外にコピーしたい場合は、そのパスを[ロケーション:]に入力するか、[参照...]ボタンで選択します。

操作11: [OK]ボタンをクリックしてダイアログボックスを閉じます。

[C/C++ プロジェクト]ビューにtstNewのプロジェクトが作成されます。このプロジェクトフォルダはワークスペース内に新たに作成されています。

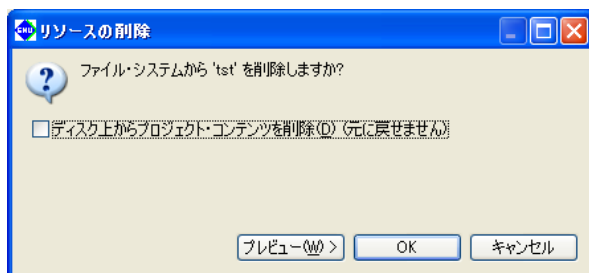
オリジナルのtstプロジェクトを[C/C++ プロジェクト]ビューから削除するには次のように操作します。

●プロジェクトの削除

操作12: [C/C++ プロジェクト]ビューでプロジェクト"tst"を選択します。

操作13: [Delete]キーを押します。

[リソースの削除]ダイアログボックスが表示されます。



操作14: [OK]ボタンをクリックします。

[C/C++ プロジェクト]ビューから`tst`が削除されます。

[ディスク上からプロジェクト・コンテンツを削除]を選択するとファイルシステム上のフォルダの内容も削除されますので、十分に注意してください。

●プロジェクトファイルの自動更新

旧バージョンのIDEで作成されたプロジェクトをインポートすると、プロジェクトファイル(`.project/``cproject/gnu33project/`)が現IDEのバージョンと互換性を持つように自動的に更新されます。

なお、旧バージョンのプロジェクトで使用されていた`.cdtproject`ファイルは、`.cproject`ファイルに置き換えられます。プロジェクトインポート時に`.cproject`ファイルが生成され、`.cdtproject`ファイルの内容が自動的に移行されます。

●ディレクトリ構造やリソースの位置について

プロジェクトフォルダ内にすべてのリソースが集約されていれば、どこにコピーしても問題はありません。コピー先でも修正なしにビルドが行えます。

オリジナルのプロジェクトフォルダ内から参照している外部ファイルがあった場合、同じディレクトリ構造の環境であれば修正等は必要ありません。チュートリアル2でも説明したように、`make`ファイルなどをIDEの自動生成ではなく、外部に用意している場合などは、パスの記述に修正が必要になる場合があります。

標準ライブラリやインクルードディレクトリは、ツールを`C:\¥EPSON¥gnu33`など、同じディレクトリにインストールしてあれば問題はありません。ツールディレクトリが異なる環境のIDEで作業する場合や、ユーザのライブラリとインクルードディレクトリは必要に応じてプロジェクトの[プロパティ]ダイアログボックスの[GNU33 ビルドオプション]で変更してください。

3.6 デバッグ環境

デバッグは、デバッグモニタS5U1C330M2S、インサーキットデバッグS5U1C33000HおよびS5U1C33001Hをサポートしており、ターゲットシステムを実際に動作させてデバッグを行うことができます。

以下にそれぞれのシステムの概要とデバッグの開始手順を説明します。詳細についてはそれぞれのマニュアルを参照してください。また、オンボードフラッシュ書き込みについては¥gnu33¥utility¥fls33内にあるReadmeja.txt(日本語)またはReadme.txt(英語)を参照してください。

注: ・各システムの接続と切り離しは、必ずすべての機器の電源をOFFにして行ってください。

- ・本節で使用するサンプルプログラムのディレクトリは¥EPSON¥gnu33¥utility¥sample_stdです。本節に記載の動作確認はこのディレクトリで可能ですが、ディレクトリ内のmakeファイルを使用してmake.exeを実行する場合は、¥sample_stdディレクトリを¥gnu33ディレクトリ直下に移す必要があります。

3.6.1 デバッグモニタS5U1C330M2S

デバッグモニタS5U1C330M2SはS1C33 Family用のミドルウェアで、S5U1C33xxxDボードやユーザーターゲットボード上でのプログラムデバッグ機能を提供します。S5U1C330M2Sを実装したボードをS5U1C330M1D1ボードを介してパーソナルコンピュータに接続することにより、デバッグgdbを使用してプログラムのデバッグが行えます。ここでは、デバッグモニタが実装されているS5U1C33xxxDボードを開発ツールとして使用してプログラムのデバッグを行う方法を説明します。

●システム構成と接続

図3.6.1.1にS5U1C33xxxDボードを使用したデバッグシステムの構成を示します。

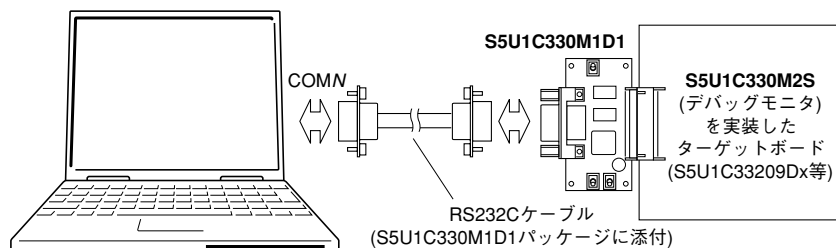


図3.6.1.1 S5U1C330M1D1とS5U1C33xxxDボードを使用するデバッグシステム

●起動と動作確認

以下のサンプルプログラムで動作確認が行えます。

- ・¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥led.elf
- ・¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥led2.elf

これらは、すべてのS5U1C33xxxDボードに対応したサンプルプログラムで、ボード上のLEDを点滅させます。led.elfはRAM(0x600000～)上で、led2.elfはフラッシュメモリ(0x200000～)上で実行するように作成されています。

プログラム内容については、ディレクトリ内のソースファイル(led.s)を参照してください。実行形式のオブジェクトファイルがサンプルとして用意されていますので、ソースの修正が不要な場合はmake.exeを実行する必要はありません。ソースを修正して使用する場合は、各プログラムに対応するmakeファイルを使用してmake.exeを実行してください。

(1) デバッグモニタの起動

S5U1C33xxxDの0xc00000からマップされているブートルーチンは、K63入力ポートが"0" (S5U1C330M1D1の[DEBUG]スイッチがON)に設定されている場合にデバッグモニタを起動するようになっています。ターゲットシステムとパーソナルコンピュータを接続後、次の手順でデバッグモニタを起動してください。

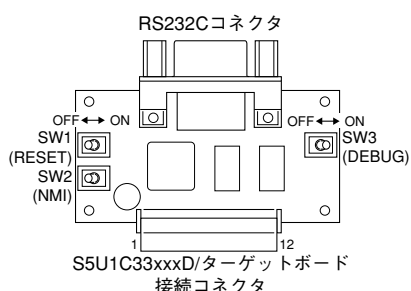


図3.6.1.2 S5U1C330M1D1ボードレイアウト

1. S5U1C330M1D1のSW3[DEBUG]をON
2. S5U1C33xxxDの電源をON
3. S5U1C33xxxDをリセット (S5U1C33xxxDのSW6[RESET]をON→OFF)
4. パーソナルコンピュータの電源をONし、Windowsを起動
5. デバグ **gdb** を起動 (起動方法は後述)

注: S5U1C330M1D1のSW3[DEBUG]をOFFした状態でS5U1C33xxxDの電源をONすると、デバグモニタは起動せずにフラッシュメモリ (0x200000～) の先頭をブートアドレスとして実行を開始します。この場合はSW3[DEBUG]をONし、S5U1C33xxxDのSW6[RESET]でリセットしてデバグモニタを起動してください。

(2) RAM上でのデバグ

S5U1C33xxxDのRAM (0x600000～) 上でデバグを行う場合のサンプルプログラムは `led.elf` です。ベクタテーブルアドレスをRAMの先頭に設定し `led.elf` をRAMにロードするデバグコマンドファイル `icdv2_led.cmd` が用意されていますので、デバグ起動時に `-x` オプションで指定してください。コマンドプロンプトからデバグを起動させる操作手順は次のとおりです。

1. 前述のとおりデバグモニタを起動
2. `¥EPSON¥gnu33¥utility¥sample_std¥dmt33209` をカレントディレクトリに設定
3. **gdb.exe** へのPATHを設定
4. コマンドプロンプト上から次のコマンドでデバグを起動

```
C:¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥>gdb -x icdv2_led.cmd
```

デバグが起動後デバグモニタモードに設定され、`led.elf` のデバグが行える状態になります。たとえば、`continue` コマンドを実行するとS5U1C33xxxDボード上のLEDが点滅します。

デバグモニタではキーブレークなどの強制ブレーク機能は使用できません。

このため、`icdv2_led.cmd` では `led.elf` のNMIルーチン内のラベル位置にブレークポイントを設定しています。S5U1C33xxxDのSW5[NMI]をONするとNMIが発生し、プログラムの実行を擬似的に強制ブレークすることができます。

(3) フラッシュメモリ上でのデバグ

S5U1C33xxxDのフラッシュメモリ (0x200000～) 上でデバグを行う場合のサンプルプログラムは `led2.elf` です。

フラッシュメモリにサンプルプログラムを書き込むには、まずフラッシュ消去/書き込みルーチン `am29f800.elf` をロードします。

さらに、`c33 fls`、`c33 fle` コマンドを実行してフラッシュ関係の初期化とイレースを行い、`load` コマンドでサンプルプログラムをロードします。コマンドの実行手順については、サンプルのデバグコマンドファイル `icdv2_led2.cmd` を参照してください。

デバグコマンドファイル `icdv2_led2.cmd` にはフラッシュ消去/書き込みルーチンのロード、ベクタテーブルアドレスの設定、`led2.elf` のフラッシュメモリへの書き込みを行うデバグコマンドが記述されていますので、デバグ起動時に `-x` オプションで指定してください。

コマンドプロンプトからデバグを起動させる操作手順は次のとおりです。

1. 前述のとおりデバグモニタを起動
2. `¥EPSON¥gnu33¥utility¥sample_std¥dmt33209` をカレントディレクトリに設定
3. **gdb.exe** へのPATHを設定
4. コマンドプロンプト上から次のコマンドでデバグを起動

```
C:¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥>gdb -x icdv2_led2.cmd
```

デバグが起動後デバグモニタモードに設定され、`led2.elf` のデバグが行える状態になります。たとえば、`continue` コマンドを実行するとS5U1C33xxxDボード上のLEDが点滅します。

デバッグモニタではキープレークなどの強制ブレーク機能は使用できません。
S5U1C33xxxDのSW5[NMI]をONするとNMIが発生し、プログラムの実行を擬似的に強制ブレークすることができます。

フラッシュメモリに書き込んだプログラムをS5U1C33xxxD単体で実行させることもできます。
デバッグを終了後、システムの電源をOFFにしてRS232Cケーブルを切り離し、S5U1C330M1D1のSW3[DEBUG]をOFFにしてS5U1C33xxxDの電源をONしてください。led2.e1fがフラッシュメモリ上で実行され、LEDが点滅します。

●注意事項

- ユーザプログラムをS5U1C33xxxDを使用してデバッグする場合は、以下の点に注意してください。
- (1) S5U1C33xxxDのデバッグモニタは、mon33ch0.libをリンクして実装しています。したがって、ユーザプログラムから内蔵シリアルインタフェースCh.0を使用することはできません。
 - (2) デバッグモニタではキープレークなどの強制ブレーク機能は使用できません。
デバッグするプログラムのNMIやキー入力割り込み処理ルーチン内にラベルを設け、そこにハードウェアPCブレークを設定して対応してください。
 - (3) RAMへのダウンロード速度は約8KB/S、フラッシュメモリへのダウンロード速度は約7KB/Sです。
ただし、使用するPCや条件により変わります。
 - (4) デバッグするプログラムはS5U1C33xxxDのRAMまたはフラッシュメモリの空き領域にロードして実行できるように作成してください。デバッグからはプログラムのロードアドレスを指定できませんので、リンク時にアドレスを決定しておく必要があります。
特に、内蔵RAMの0～0x2fと外部SRAMの0x6ff640～0x6fffffはS5U1C330M2Sが使用しており、ここを書き換えると動作しなくなりますので注意してください。
また、メモリをメモリ操作コマンドで書き換える場合もこの領域が変更されないように注意してください。

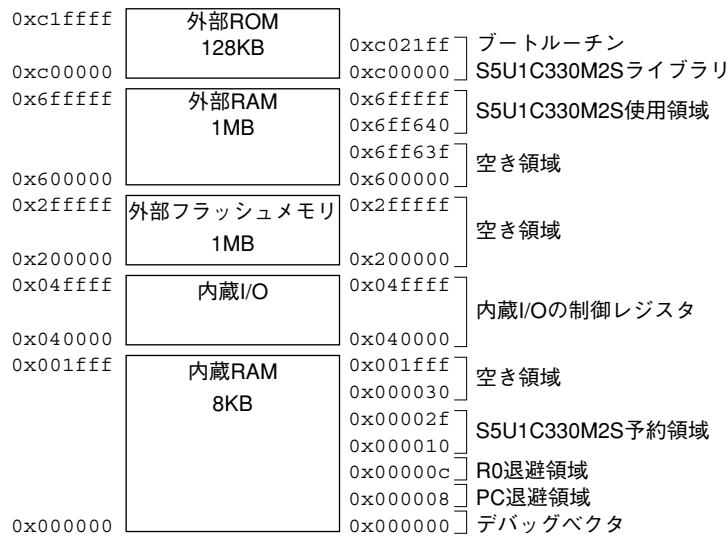


図3.6.1.3 S5U1C33xxxDメモリマップ

- (5) その他デバッグモニタの制限事項については、"S1C33 Family Debug Monitor Operation Manual"を参照してください。

3.6.2 インサーキットデバッガS5U1C33000H

インサーキットデバッガS5U1C33000Hはデバッガgdbのコマンドに従ってS1C33チップ上のデバッグ機能を制御するツールです。デバッグモニタと同等のデバッグ機能に加え、内蔵のトレースメモリによりトレース機能も提供します。ここでは、S5U1C33000HとS5U1C33xxxDボードを開発ツールとして使用してプログラムのデバッグを行う方法を説明します。

●システム構成と接続

図3.6.2.1にS5U1C33000HとS5U1C33xxxDボードを使用したデバッグシステムの構成を示します。

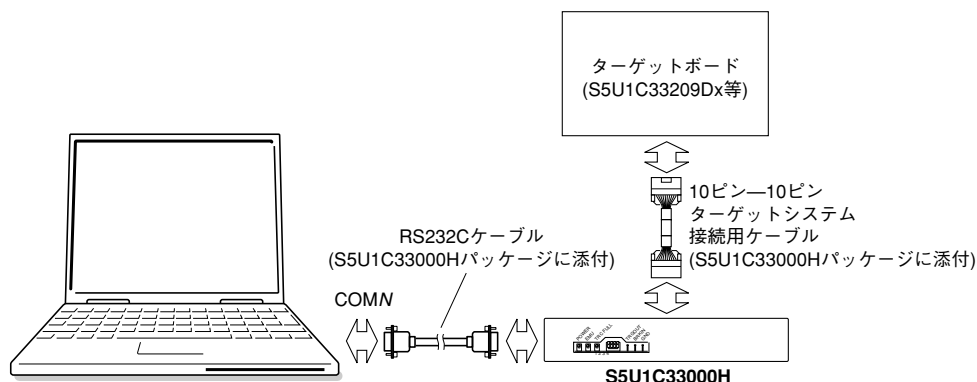


図3.6.2.1 S5U1C33000HとS5U1C33xxxDボードによるデバッグシステム

注: S5U1C33000Hとデバッグモニタの併用はできません。S5U1C33xxxDボードにS5U1C330M1D1は接続しないでください。取り付けて使用する場合は、S5U1C330M1D1の[DEBUG]スイッチ(SW3)は必ずOFFにしてください。

●起動と動作確認

以下のサンプルプログラムで動作確認が行えます。

- ¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥led.elf
- ¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥led2.elf

これらは、すべてのS5U1C33xxxDボードに対応したサンプルプログラムで、ボード上のLEDを点滅させます。led.elfはRAM(0x600000～)上で、led2.elfはフラッシュメモリ(0x200000～)上で実行するように作成されています。

プログラム内容については、ディレクトリ内のソースファイル(led.s)を参照してください。実行形式のオブジェクトファイルがサンプルとして用意されていますので、ソースの修正が不要な場合はmake.exeを実行する必要はありません。ソースを修正して使用する場合は、各プログラムに対応するmakeファイルを使用してmake.exeを実行してください。

(1) システムの起動

S5U1C33000H、S5U1C33xxxD、パーソナルコンピュータを接続後、次の手順でシステムを起動してください。

1. S5U1C33000HのディップスイッチをすべてOPEN(上)
2. S5U1C33000Hの電源をON
3. S5U1C33xxxDの電源をON
4. パーソナルコンピュータの電源をONし、Windowsを起動
5. デバッガgdbを起動(起動方法は後述)

(2) RAM上でのデバッグ

S5U1C33xxxDのRAM(0x600000～)上でデバッグを行う場合のサンプルプログラムはled.elfです。ベクタテーブルアドレスをRAMの先頭に設定しled.elfをRAMにロードするデバッグコマンドファイルicdv2_led.cmdが用意されていますので、デバッガ起動時に-xオプションで指定してください。

コマンドプロンプトからデバッグを起動させる操作手順は次のとおりです。

1. 前述のとおりシステムを起動
2. ¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥をカレントディレクトリに設定
3. **gdb.exe**へのPATHを設定
4. コマンドプロンプト上から次のコマンドでデバッグを起動

```
C:¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥>gdb -x icdv2_led.cmd
```

デバッグが起動後ICD2モードに設定され、led.elfのデバッグが行える状態になります。たとえば、continueコマンドを実行するとS5U1C33xxxDボード上のLEDが点滅します。

S5U1C33000Hではキープレーク機能も使用可能です。デバッグの[中断]ボタンでプログラムの実行を中断することができます。また、トレース機能も使用可能です。トレース方法については"10 デバッグ"を参照してください。

(3) フラッシュメモリ上でのデバッグ

S5U1C33xxxDのフラッシュメモリ(0x200000～)上でデバッグを行う場合のサンプルプログラムは、led2.elfです。

フラッシュメモリにサンプルプログラムを書き込むには、まずフラッシュ消去/書き込みルーチンam29f800.elfをロードします。さらに、c33 fls、c33 fleコマンドを実行してフラッシュ関係の初期化とイレースを行い、loadコマンドでサンプルプログラムをロードします。コマンドの実行手順については、サンプルのデバッグコマンドファイルicdv2_led2.cmdを参照してください。

デバッグコマンドファイルicdv2_led2.cmdにはフラッシュ消去/書き込みルーチンのロード、ベクタテーブルアドレスの設定、led2.elfのフラッシュメモリへの書き込みを行うデバッグコマンドが記述されていますので、デバッグ起動時に-xオプションで指定してください。

コマンドプロンプトからデバッグを起動させる操作手順は次のとおりです。

1. 前述のとおりシステムを起動
2. ¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥をカレントディレクトリに設定
3. **gdb.exe**へのPATHを設定
4. コマンドプロンプト上から次のコマンドでデバッグを起動

```
C:¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥>gdb -x icdv2_led2.cmd
```

● 注意事項

ユーザプログラムをS5U1C33000HとS5U1C33xxxDを使用してデバッグする場合は、以下の点に注意してください。

- (1) デバッグするプログラムはS5U1C33xxxDのRAMまたはフラッシュメモリの空き領域にロードして実行できるように作成してください。デバッグからはプログラムのロードアドレスを指定できませんので、リンク時にアドレスを決定しておく必要があります。S5U1C33xxxDのメモリマップについては図3.6.1.3を参照してください。
- (2) S5U1C33000HをRS232CケーブルでCOMポートに接続するのみで、S5U1C33000Hの全機能を使用することができます。
RAMへのダウンロード速度は約8KB/S、フラッシュメモリへのダウンロード速度は約7KB/Sです。ただし、使用するPCや条件により変わります。
- (3) その他S5U1C33000Hの制限事項については、"S1C33 Family In-Circuit Debugger Manual"を参照してください。

3.6.3 インサーキットデバグS5U1C33001H

インサーキットデバグS5U1C33001Hはデバグガ**gdb**のコマンドに従ってS1C33チップ上のデバグ機能を制御するツールです。デバグモニタと同等のデバグ機能に加え、内蔵のトレースメモリによりトレース機能も提供します。ここでは、S5U1C33001HとS5U1C33xxxDボードを開発ツールとして使用してプログラムのデバグを行う方法を説明します。

注: サポートしているOSは、Windows XPおよびWindows Vistaです。

- S5U1C33001Hを使用する場合は、専用のUSBドライバが必要になります。USBドライバは¥gnu33¥utility¥drv_usbフォルダにあります。インストール方法についてはICDに添付のマニュアルを参照してください。

●システム構成と接続

図3.6.3.1にS5U1C33001HとS5U1C33xxxDボードを使用したデバグシステムの構成を示します。

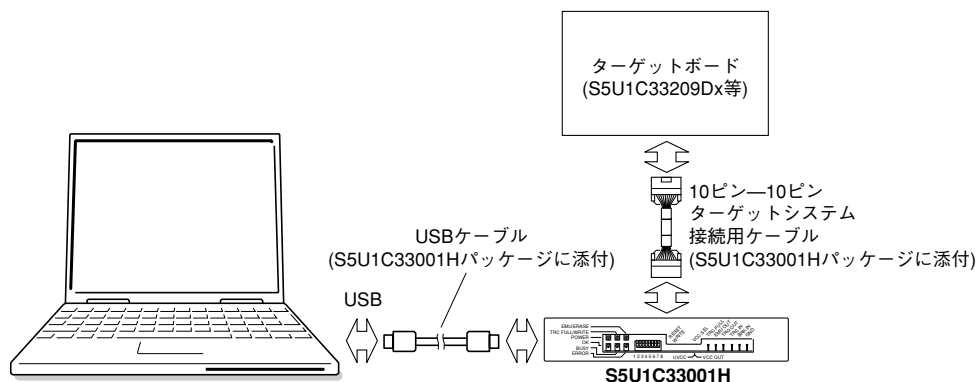


図3.6.3.1 S5U1C33001HとS5U1C33xxxDボードによるデバグシステム

注: S5U1C33001Hとデバグモニタの併用はできません。S5U1C33xxxDボードにS5U1C33001Hは接続しないでください。取り付けて使用する場合は、S5U1C33001Hの[DEBUG]スイッチ(SW3)は必ずOFFにしてください。

●起動と動作確認

以下のサンプルプログラムで動作確認が行えます。

- ¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥led.elf
- ¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥led2.elf

これらは、すべてのS5U1C33xxxDボードに対応したサンプルプログラムで、ボード上のLEDを点滅させます。led.elfはRAM(0x600000～)上で、led2.elfはフラッシュメモリ(0x200000～)上で実行するように作成されています。プログラム内容については、ディレクトリ内のソースファイル(led.s)を参照してください。実行形式のオブジェクトファイルがサンプルとして用意されていますので、ソースの修正が不要な場合は**make.exe**を実行する必要はありません。ソースを修正して使用する場合は、各プログラムに対応するmakeファイルを使用して**make.exe**を実行してください。

(1) システムの起動

S5U1C33001H、S5U1C33xxxD、パーソナルコンピュータを接続後、次の手順でシステムを起動してください。

1. S5U1C33001HのディップスイッチをすべてOPEN(上)
2. S5U1C33001Hの電源をON
3. S5U1C33xxxDの電源をON
4. パーソナルコンピュータの電源をONし、Windowsを起動
5. デバグガ**gdb**を起動(起動方法は後述)

(2) RAM上でのデバッグ

S5U1C33xxxDのRAM(0x600000～)上でデバッグを行う場合のサンプルプログラムはled.elfです。ベクタテーブルアドレスをRAMの先頭に設定しled.elfをRAMにロードするデバッグコマンドファイルicdv3_led.cmdが用意されていますので、デバッグ起動時に-xオプションで指定してください。コマンドプロンプトからデバッグを起動させる操作手順は次のとおりです。

1. 前述のとおりシステムを起動
2. ¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥をカレントディレクトリに設定
3. **gdb.exe**へのPATHを設定
4. コマンドプロンプト上から次のコマンドでデバッグを起動
C:¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥>**gdb -x icdv3_led.cmd**

デバッグが起動後ICD3モードに設定され、led.elfのデバッグが行える状態になります。たとえば、continueコマンドを実行するとS5U1C33xxxDボード上のLEDが点滅します。

S5U1C33001Hではキープレーク機能も使用可能です。デバッグの[中断]ボタンでプログラムの実行を中断することができます。また、トレース機能も使用可能です。トレース方法については"10 デバッグ"を参照してください。

(3) フラッシュメモリ上でのデバッグ

S5U1C33xxxDのフラッシュメモリ(0x200000～)上でデバッグを行う場合のサンプルプログラムは、led2.elfです。

フラッシュメモリにサンプルプログラムを書き込むには、まずフラッシュ消去/書き込みルーチンam29f800.elfをロードします。さらに、c33 fls、c33 fleコマンドを実行してフラッシュ関係の初期化とイレースを行い、loadコマンドでサンプルプログラムをロードします。コマンドの実行手順については、サンプルのデバッグコマンドファイルicdv3_led2.cmdを参照してください。

デバッグコマンドファイルicdv3_led2.cmdにはフラッシュ消去/書き込みルーチンのロード、ベクタテーブルアドレスの設定、led2.elfのフラッシュメモリへの書き込みを行うデバッグコマンドが記述されていますので、デバッグ起動時に-xオプションで指定してください。

コマンドプロンプトからデバッグを起動させる操作手順は次のとおりです。

1. 前述のとおりシステムを起動
2. ¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥をカレントディレクトリに設定
3. **gdb.exe**へのPATHを設定
4. コマンドプロンプト上から次のコマンドでデバッグを起動
C:¥EPSON¥gnu33¥utility¥sample_std¥dmt33209¥>**gdb -x icdv3_led2.cmd**

● 注意事項

ユーザプログラムをS5U1C33001HとS5U1C33xxxDを使用してデバッグする場合は、以下の点に注意してください。

- (1) デバッグするプログラムはS5U1C33xxxDのRAMまたはフラッシュメモリの空き領域にロードして実行できるように作成してください。デバッグからはプログラムのロードアドレスを指定できませんので、リンク時にアドレスを決定しておく必要があります。S5U1C33xxxDのメモリマップについては図3.6.1.3を参照してください。
- (2) S5U1C33001HをUSBケーブルで接続するのみで、S5U1C33001Hの全機能を使用することができません。RAMへのダウンロード速度は約8KB/S、フラッシュメモリへのダウンロード速度は約7KB/Sです。ただし、使用するPCや条件により変わります。
- (3) その他S5U1C33001Hの制限事項については、"S1C33 Family In-Circuit Debugger Manual"を参照してください。

3.7 データエリア、セクション

ここでは、ソースファイルの作成およびリンクの際に必要なデータエリアおよびセクション管理の概念を説明しておきます。

3.7.1 データエリア

プログラムが頻繁に使用する領域は、ベースアドレスをレジスタに設定し、`ext`命令でディスプレースメントを設定してアクセスする方法が一般的です。

```
例: ext      imm13                      ; 13-bit displacement (8K bytes)
     ld.b    %rd, [%rb]                 ; = ld.b  %rd, [%rb+imm13]
```

この例では、2命令で、8Kバイトの領域をベースアドレス(`%rb`)からの相対アドレスでアクセスできます。頻繁に使用する変数などをこの領域に配置しておくと、他のアクセス方法に比べコードサイズを小さく、処理速度を高くすることができます。

8Kバイトを超える領域についても、64Mバイトまでは`ext`命令を2つ使用するディスプレースメント付きのレジスタ間接アドレッシング(全3命令)でアクセス可能です。

C/C++コンパイラはこの手法を使用するために、`%r15`をベースアドレス(データエリアポインタ)を設定するレジスタとして確保しています。そこから始まる64Mバイトがデフォルトデータエリアです。

データエリアの基本的な使用方法を以下に示します。

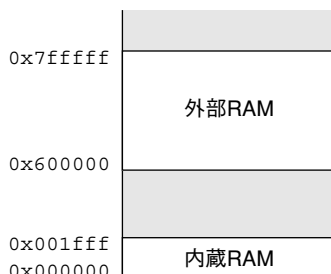


図3.7.1.1 メモリ構成例

図3.7.1.1に示すとおり、アドレス0x0から内蔵RAMが配置されているものとします。

説明ではRAMのみを使用しますが、ROMやI/Oメモリもデータエリアとして扱うことができます。

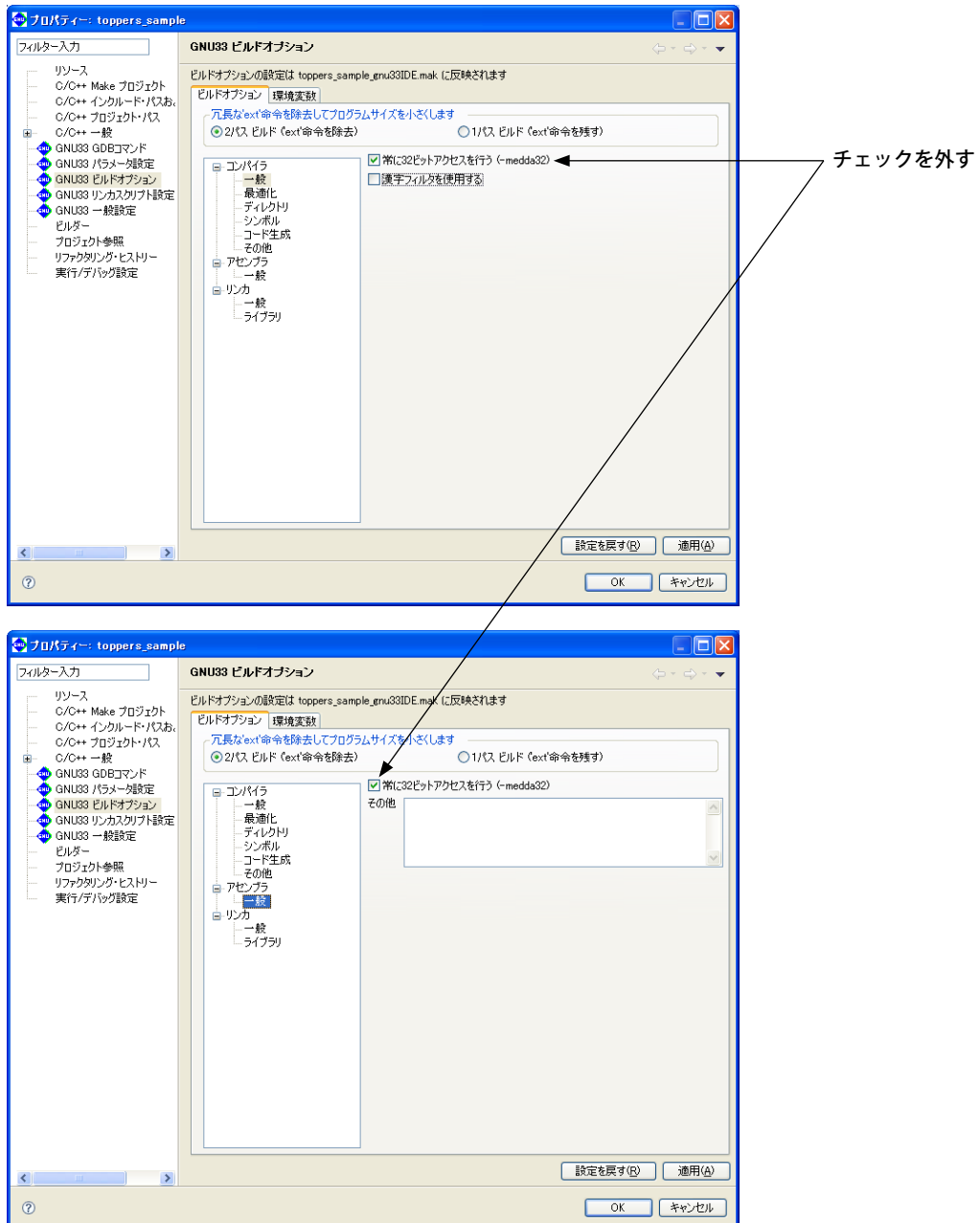
ここでは、アドレス0x0から始まる64Mバイトをデフォルトデータエリアとして使用します。

`xgcc`の`-medda32`オプションを付加すると、デフォルトデータエリアを使用しません。したがって、デフォルトデータエリアポインタ(`__dp`)を参照せずにデータをアクセスします。

1. デフォルトデータエリアを使用するには

IDEではC/C++コンパイラのビルドオプションとして`-medda32`がデフォルトで指定されています。これはデフォルトデータエリアポインタを使用せずに32ビットアドレス空間すべてをアクセスできるようにするオプションです。

デフォルトデータエリアを使用する場合は、このオプションを指定しないでください。



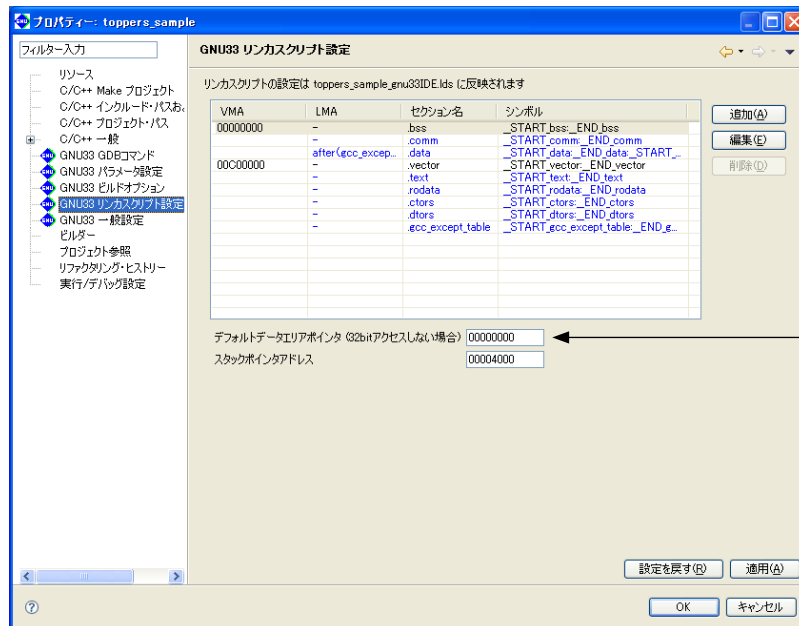
2. データエリア先頭アドレスの指定

リンカスクリプトファイル内でのデータエリアアドレスの指定

リンク用に作成するリンカスクリプトファイルでは、データエリアポインタシンボルを用いてデータエリアの先頭アドレスを指定します。

例: `__dp = 0x0;`

IDEでは、[プロパティ]ダイアログボックスの[GNU33 リンカスクリプト設定]で[デフォルトデータエリアポインタ]テキストボックスに0を入力します。



ソース内でのデータエリアアドレスの指定

デフォルトデータエリアポインタとして使用される`%r15`は、C/C++コンパイラ等では初期設定されないため、ユーザプログラムで初期化する必要があります。これには、リンカスクリプトファイルで定義するデータエリアポインタシンボルを使用することができます。

例: `xld.w %r15, __dp ; Set default data area start address`

3. 外部変数定義

外部変数を定義すると、すべてデフォルトデータエリアに配置されます。

4. コンパイル結果

外部変数を参照しているステートメントは以下のようにコンパイルされます。

例1: `SzDefaultData[0] = 0;`を`-medda32`オプションなしでコンパイルした場合

```
xld.w %r4,0
ext    doff_hi (szDefaultData)
ext    doff_lo (szDefaultData)
ld.b   [%r15],%r4
```

`doff_hi (szDefaultData)`は`__dp`から`szDefaultData`までのオフセットの上位13ビット、`doff_lo (szDefaultData)`は下位13ビットです。デフォルトデータエリアは64Mバイトのため、3命令でアクセスされます。

例2: `SzDefaultData[0] = 0;`を`-medda32`オプションありでコンパイルした場合

```
xld.w %r4,0 ; 0x0
xld.w %r5,SzDefaultData
ld.w [%r5],%r4
```

このようにデータポインタを使用することなく、データをロードします。

例3: `i = SzDefaultData[0];` (セーブ命令)を`-medda32`オプションありでコンパイルした場合

```
xld.w %r4,SzDefaultData
ld.w %r4, [%r4]
```

このようにデータポインタを使用することなく、データをセーブします。

3.7.2 セクション

ソースファイルにはプログラムコード、定数、変数など、いろいろな属性を持つデータが記述されます。また、組み込み型システムでは、データをROMやRAMなど、異なるデバイスへの割り付けを前提に管理する必要があります。このため、データを属性で管理できるようにセクションという論理的な領域を用意しています。

たとえば、複数のソースファイル内にあるプログラムコードを同一のセクションに置くものとして作成すると、リンクの際にもそれらを容易に1つにまとめることができ、結果として同じROMに配置されます。もちろん、ファイルごとに配置アドレスを指定可能ですので、内蔵ROMと外部ROMといった別のデバイスに配置することもできます。

C/C++コンパイラ**xgcc**では、大きく分けて8種類(属性)のセクションが設定されており、データはソースの内容に従って、適切なセクションに配置されます。

(1) **.text**セクション

プログラムコードを配置します。最終的にはROMに書き込みます。

(2) **.data**セクション

初期値を持ちリード/ライト可能なデータを配置します。データはROMに書き込み、プログラムでRAMに転送して使用します。

(3) **.rodata**セクション

const宣言された変数を配置します。最終的にはROMに書き込みます。

(4) **.bss**セクション

初期値を持たない変数を配置します。RAM空間に領域のみ確保されます。

(5) **.comm**セクション

初期値を持たない変数を配置します。

変数が**.bss**と**.comm**のどちらに配置されるかについては、C/C++コンパイラが決定しています。

アセンブラソース上で初期値を持たない変数を配置する場合は、**.bss**セクションに配置してください。

RAM空間に領域のみ確保されます。

(6) **.ctors**セクション

グローバルクラスのコンストラクタ関数へのポインタ配列を配置します。

C++ソースの場合に、C/C++コンパイラにより生成されます。

最終的にはROMに書き込みます。

(7) **.dtors**セクション

グローバルクラスのデストラクタ関数へのポインタ配列を配置します。

C++ソースの場合に、C/C++コンパイラにより生成されます。

最終的にはROMに書き込みます。

(8) **.gcc_except_table**セクション

例外処理用テーブルデータを配置します。

C++で例外処理を使用するときに、C/C++コンパイラにより生成されます。

最終的にはROMに書き込みます。

.vectorセクション

IDEではベクタテーブル用に**.rodata**属性の**.vector**セクションが用意されます。

C/C++ソースの場合はベクタテーブルをconst宣言で作成し、そのオブジェクトを**.vector**セクションに配置します。

アセンブラソースの場合は、ベクタテーブルを**.rodata**セクションまたは**.text**セクションに記述することができます。ただし、**.text**セクションに配置した場合は、IDEで**.vector**セクションの属性を**.text**に変更する必要があります。

詳細は、「5.7.7 リンカスクリプトの編集」を参照してください。

3 ソフトウェア開発手順

次に、セクションと実際のメモリとの対応について説明します。

ソースファイル例

(file1.s)

```
        :
        .set SP_INI,0x0800    ; sp is in end of 2KB internal RAM

        .text
        .long BOOT          ; BOOT VECTOR
BOOT:
        xld.w %r15,SP_INI
        ld.w  %sp,%r15      ; set SP
        xld.w %r15,__dp     ; set default data area pointer
        xcall main          ; goto main
        jp   BOOT           ;
```

(file2.s)

```
        :
        .data                ;.data section
        :
        :
        .rodata              ;.rodata section
        :
        :
```

(file3.c)

```
        :
#include <string.h>

int      i_bss;                /* .bss section */
int      i_data = 1;           /* .data section */
const    int   i_rodata = 0x12345678; /* .rodata section */
const    char  sz_rodata[] = "ABCDEFGH"; /* .rodata section */
        :
int main()                     /* .text section */
{
    char sz_buf[10];
    int i;

    for( i = 0; i < 5; ++i ){
        sz_buf[i] = sz_rodata[i];
    }

    :
    :
    return 0;
}
```

リンカスクリプトファイル例

```

(sample.lds)
OUTPUT_FORMAT("elf32-c33", "elf32-c33",
              "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);
SECTIONS
{
    __dp = 0x0;                                     ... (1)
    . = 0x0;                                         ... (2)
    .bss : { *(.bss) }                               ... (3)
    .data :
        AT (__load_data)
        { __start_data = . ; *(.data) ; __end_data = . ; } ... (4)
    . = 0xc00000;                                     ... (5)
    .text : { *(.text) }                             ... (6)
    .rodata : { *(.rodata) }                         ... (7)
    __load_data = . ;                                ... (8)
}

```

例に示したソースファイルは以下のセクションを含んでいます。

```

file1    .textセクション
file2    .dataおよび.rodataセクション
file3    .text、.bss、.dataおよび.rodataセクション

```

これらのセクションは、リンカスクリプトファイルに記述されたSECTIONSコマンドに従って再配置されます。リンカスクリプトファイル例の内容は以下のとおりです。

- (1) デフォルトデータエリアポインタ(__dp)を0x0に設定します。このシンボルはソースファイル例"file1.s"で%r15レジスタを設定するために使用しています。
- (2) ロケーションカウンタを0x0(RAM開始アドレス)に設定します。続いて指定されるセクションは、このアドレスから配置されます。
- (3) 実行形式オブジェクトファイルに出力する.bss出力セクションを定義します。{ *(.bss) }は入力ファイル内のすべての.bssセクションを、ここで定義する.bssセクションに配置することを指定します。
- (4) .data出力セクションを定義します。このセクション(VMA)は.bssセクションの直後に配置されます。入力ファイル内のすべての.dataセクションは、この出力セクションに入ります。定義されたセクションはVMA(実際にコードを実行したりデータをリード/ライトする仮想メモリアドレス)に配置されますが、通常、VMAはLMA(データを記憶しておくロードメモリアドレス)と同じになります。.dataセクションの場合、初期値をROMに書き込んでおき、RAMにコピーしてから使用しなければなりません。このため、LMA(ROMアドレス)を別に指定する必要があります。AT文は.dataセクションの実コードをステップ(8)で指定する__load_data(ROM内)から配置することを指定しています。__start_dataと__end_dataは.dataセクションの開始アドレスと終了アドレス(VMA)を後で参照するために定義しています。これらのシンボルはソース例"file3.c"内でデータをLMAからVMAにコピーするために使用しています。
- (5) ロケーションカウンタを0xc00000(ROM開始アドレス)に設定します。続いて指定されるセクションは、このアドレスから配置されます。
- (6) プログラムコード用の.text出力セクションを定義します。入力ファイル内のすべての.textセクションは、アドレス0xc00000から配置されます。
- (7) .rodata出力セクションを定義します。このセクションは.textセクションの直後に配置されます。入力ファイル内のすべての.rodataセクションは、この出力セクションに配置されます。
- (8) .dataセクションのLMAを指定するためのシンボル__load_dataを定義します。このシンボルには現在のロケーションカウンタ値(.rodataセクション直後のアドレス)が定義されます。

3 ソフトウェア開発手順

このスクリプト例により構成されるメモリマップを図3.7.2.1に示します。

```
ld -o sample.elf file1.o file2.o file3.o -T sample.lds
```

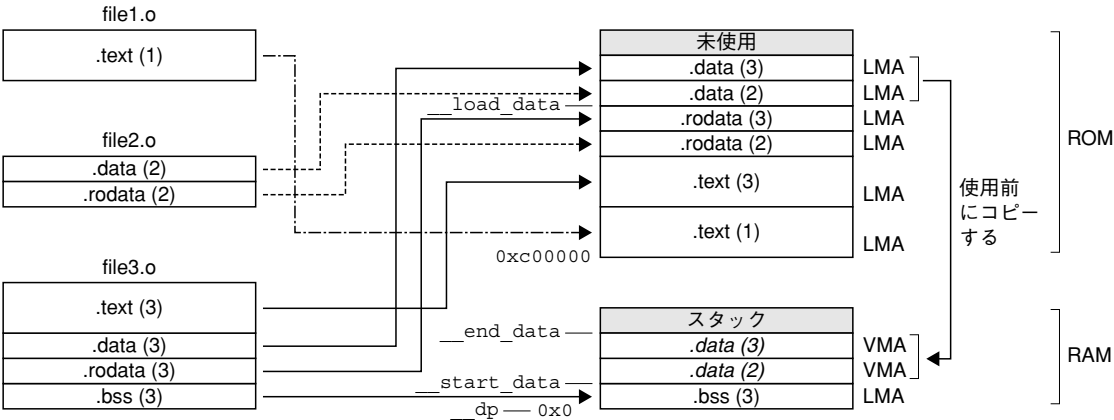


図3.7.2.1 sample.ldsにより構成されるメモリマップ

4.2 C/C++ソースの文法

本パッケージのC/C++コンパイラ**xgcc**はANSI C/C++準拠のGNU C/C++コンパイラ(ver. 3.3.2)です。C/C++ソースはANSI C/C++の規定に従って作成してください。文法について不明な点がある場合は、ANSI C/C++を解説している一般の書籍の参照をお願いいたします。

4.2.1 データ型

C/C++コンパイラ**xgcc**はANSI C/C++のすべてのデータ型に対応しています。各型のサイズ(バイト数)、表現できる数値の有効範囲を表4.2.1.1に示します。

表4.2.1.1 型の種類とサイズ

型	サイズ	数値の有効範囲
char	1	-128～127
unsigned char	1	0～255
short	2	-32768～32767
unsigned short	2	0～65535
int	4	-2147483648～2147483647
unsigned int	4	0～4294967295
long	4	-2147483648～2147483647
unsigned long	4	0～4294967295
pointer	4	0～4294967295
float	4	1.175e-38～3.403e+38(正規化数)
double	8	2.225e-308～1.798e+308(正規化数)
long long	8	-9223372036854775808～9223372036854775807
unsigned long long	8	0～18446744073709551615
bool	1	true/false
wchar_t (C)	2	0～65535
wchar_t (C++)	4	-2147483648～2147483647

float型、double型はIEEE標準規格のフォーマットに準拠しています。

long long型の定数を扱う場合は、接尾子LLまたはll(long long型)/ULLまたはull(unsigned long long型)が必要になります。この接尾子がないと、コンパイラはlong long型の定数として認識できないため、ワーニングになります。

```
例: long long ll_val;
    ll_val = 0x1234567812345678;
        → warning: integer constant is too large for "long" type
    ll_val = 0x1234567812345678LL;
        → OK
```

bool型はtrue(真)/false(偽)を扱うためのデータ型です。Cでは、stdbool.hをインクルードする必要があります。C++では独立した型になります。

wchar_t型はワイド文字を扱うためのデータ型であり、Cではstdlib.h/stddef.h内でunsigned short型として定義されています。

また、C++では符号付き4バイトの独立した型になります。C++において、wchar_tを使用するには、接頭辞Lを指定する必要があります。これは、コンパイラにワイド文字列として扱うように指示するためです。

```
例: wchar_t buf[] = L"123";
```

4.2.2 ライブラリ関数とヘッダファイル

本パッケージには、ANSI標準ライブラリと浮動小数点/整数剰余演算用のエミュレーションライブラリが用意されています。また、"include"ディレクトリ内のヘッダファイルには、ライブラリの関数宣言、マクロなどが定義されています。ライブラリ関数を使用する場合は、その宣言を含んだヘッダファイルを#include命令でインクルードしてください。ライブラリファイルの種類およびヘッダファイルとの対応は次のとおりです。

表4.2.2.1 ライブラリファイル/関数一覧

ANSI標準ライブラリ

ファイル名	関数/マクロ	対応ヘッダファイル
libc.a	tmpfile*, tmpnam*, remove*, fopen*, freopen, fclose*, setbuf*, setvbuf*, fflush*, clearerr*, feof*, ferror*, perror, fseek*, fgetpos*, fsetpos*, ftell*, rewind*, getchar, fgetc,getc, gets, fgets, fscanf, scanf, sscanf, fread, putchar, fputc,putc, puts, fputs, ungetc, fprintf, printf, sprintf, vfprintf, vprintf, vsprintf, fwrite	stdio.h
	abort, exit, atexit*, getenv*, system*, malloc, calloc, realloc, free, atoi, atol, atof, strtol, strtoul, strtod, abs, labs, div, ldiv, rand, srand, bsearch, qsort	stdlib.h
	time, difftime*, clock*, mktime, localtime*, gmtime, asctime*, ctime*	time.h
	acos, asin, atan, atan2, ceil, cos, cosh, exp, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan, tanh	math.h, errno.h, float.h, limits.h
	memchr, memmove, strchr, strcspn, strncat, strpbrk, strstr, memcmp, memset, strcmp, strerror, strncmp, strchr, strtok, memcpy, strcat, strcpy, strlen, strncpy, strspn	string.h
	isalnum, iscntrl, isgraph, isprint, isspace, isxdigit, toupper, isalpha, isdigit, islower, ispunct, isupper, tolower	ctype.h
	va_start, va_arg, va_end	stdarg.h
	assert	assert.h
	setlocale*	locale.h
libgcc2.a	strtol, strtoull	stdlib.h

*の付いた関数はダミー関数です。

エミュレーションライブラリ

ファイル名	関数
libgcc.a	adddf3, subdf3, muldf3, divds3, negdf2, addds3, subds3, mulds3, divds3, negds2, fixunsdfsi, fixdfsi, floatsidf, fixunssfsi, fixfsi, floatsisf, truncdfsf2, extendsdfsf2, fcmpd, fcmps, divsi3, udivsi3, modsi3, umodsi3

long long型エミュレーションライブラリ

ファイル名	関数
libgcc2.a	muldi3, divdi3, udivdi3, moddi3, umoddi3, negdi2, lshrdi3, ashldi3, ashldi3, fixunsdfdi, fixdfdi, floatdidf, fixunssfdi, fixsfdi, floatdisf, cmpdi2, ucmpdi2, ffsdi2

ライブラリに含まれる関数の機能については"7 ライブラリ"を参照してください。また、ライブラリ関数を使用した場合は、リンク時にその関数を含むライブラリファイルを指定してください。リンクは指定されたライブラリファイルの中から必要なオブジェクトモジュールのみを取り出してリンクします。

4.2.3 インラインアセンブル

C/C++コンパイラ**xgcc**はインラインアセンブルに対応しており、**asm**文が使用可能です。これに伴い、**"asm"**は予約語となりますので注意してください。

形式: **asm("<文字列>");**

例1:

```
/* HALT mode */
asm("halt");
```

例2:

```
/* Trap Table*/
asm(".long      BOOT¥n¥
    .space     8¥n¥
    .long      ZERO_DIV¥n¥
    .space     4¥n¥
    .long      ADDR_ERR¥n¥
    .long      NMI¥n¥
    .space     32¥n¥
    .long      INT1¥n¥
    .long      INT2");
```

例3:

```
BOOT() {
    asm("xld.w %r15,0x0800");
    asm("ld.w  %sp,%r15");    /* set SP */
    asm("ld.w  %r15,0x0000"); /* set data area pointer */
    :
}
```

アセンブリソースの記述方法の詳細については"4.3 アセンブリソースの文法"を参照してください。

4.2.4 プロトタイプ宣言

●割り込み処理関数の宣言

割り込み処理関数は次の形式でプロトタイプ宣言します。

<型> <関数名> **__attribute__ ((interrupt_handler));**

例:

```
void foo(void) __attribute__ ((interrupt_handler));
```

```
int int_num;
void foo()
{
    int_num = 5;
}
```

アセンブラコード

```
foo:
    pushn    %r2
    xld.w    %r2,5          ;0x5
    xld.w    %r1,int_num
    ld.w     [%r1],%r2
    popn     %r2
    reti
```

4.2.5 alloca()/可変長配列

C/C++コンパイラ**xgcc**は**alloca()**/可変長配列に対応しています。

alloca()を使用する場合は、**alloca.h**をインクルードしてください。**alloca()**は、スタックオーバーフローのチェックをしていませんので、その時点でのスタックのサイズに注意してください。

alloca()/可変長配列を使用した場合、フレームポインタとして**%r0**が割り当てられます。

例: `#include <alloca.h>`

```
int main()
{
    char* cp_pt;
    cp_pt = (char*)alloca(3);
    cp_pt[0] = 5;
    cp_pt[1] = 6;
    cp_pt[2] = 7;

    return 0;
}
```

アセンブラコード

```
main:
    pushn    %r0
    ld.w     %r0,%sp      ← フレームポインタ(%r0)に%spの値をセーブ
    ld.w     %r5,%sp
    ld.w     %r4,%sp
    xsub     %r4,4        ;0xffffffffc
    ld.w     %sp,%r4
    xld.w    %r4,5        ;0x5
    ld.b     [%sp],%r4
    xld.w    %r4,6        ;0x6
    xld.b    [%sp+1],%r4
    xld.w    %r4,7        ;0x7
    xld.b    [%sp+2],%r4
    ld.w     %sp,%r5
    xld.w    %r4,0        ;0x0
    popn     %r0
    ret
```

4.3 アセンブリソースの文法

4.3.1 ステートメント

アセンブリソースの個々の命令や定義をステートメントと呼びます。基本的なステートメントの構成は次のとおりです。

構文パターン

1	<ニーモニック>	(<オペランド>)	(;<コメント>)
2	<アセンブラ擬似命令>	(<パラメータ>)	(;<コメント>)
3	<ラベル>:		(;<コメント>)
4	;<コメント>		
5	<拡張命令>	<オペランド>	(;<コメント>)
6	<プリプロセッサ擬似命令>	(<パラメータ>)	(;<コメント>)

例:

	ステートメント	構文パターン
; boot.s 2002.2.14		4
; boot program		4
#define SP_INI,0x0800 ; Stack pointer value		6
.text		2
.long BOOT ; BOOT VECTOR		2
BOOT:		3
xld.w %r15,SP_INI		5
ld.w %sp,%r15 ; set SP		1
ld.w %r15,__dp ; set data area pointer		1
xcall main ; goto main		5
xjp BOOT ; infinity loop		5
:	:	:

上記の例が一般的なソース記述方法です。視認性を高めるため、各ステートメントを構成する要素はタブやスペースで位置を揃えています。

●制限事項

- 各行には1つのステートメントのみ記述できます。1行に2つ以上の命令を記述するとエラーとなります。ただし、コメントは命令やラベルの行にも記述することができます。

例: ;OK

```
BOOT: ld.w %r1,%r2
      ld.w %r0,%r1
;Error
BOOT: ld.w %r1,%r2      ld.w %r0,%r1
```

- 1つのステートメントを複数行に分けて記述することはできません。1行で完結していないステートメントはエラーとなります。

例: ;OK

```
ld.w %r1,%r2
;Error
ld.w %r1,
      %r2
```

- コメントを除き、使用可能な文字はASCIIキャラクタ(英数記号)に限られます。また、使用できる記号にも制限があります(詳細は後述)。コメントにはASCIIキャラクタ以外の漢字なども使用できます。漢字をコメントにする場合は、/* ... */を使用してください。

(1) 命令(ニーモニック&オペランド)

C33コアに対する命令は<ニーモニック>+<オペランド>の構成となります。命令によっては、オペランドを持たないものもあります。

●命令表記の一般形

一般形: <ニーモニック>

<ニーモニック> タブまたはスペース <オペランド>

<ニーモニック> タブまたはスペース <オペランド1>, <オペランド2>

例: nop
 call SUB1
 ld.w %r0, 0x4

行の中で、ニーモニックの記述開始位置に制限はありません。ニーモニックの前にあるタブやスペースは無視されます。一般的にはタブによってニーモニックの頭揃えを行います。

オペランドを持つ命令の場合、ニーモニックとオペランド間は1個以上のタブまたはスペースで区切る必要があります。また、オペランドが複数の場合は、オペランド間を1個のカンマ(,)で区切ります。オペランド間のスペースは無視されます。

オペランドの要素については後述します。

●ニーモニックの種類

S1C33 Familyでは、以下のコア命令を使用できます。

C33 STDコア命令

nop	slp	halt	pushn	popn	brk	ret.d
int	reti	call	call.d	ret	ret.d	j p
jp.d	jrgt	jrgt.d	jrgc	jrgc.d	jrlt	jrlt.d
jrle	jrle.d	jrugt	jrugt.d	jrugc	jrugc.d	j r u l t
jrult.d	jrulc	jrulc.d	jreq	jreq.d	jrnc	jrnc.d
ld.b	ld.ub	ld.h	ld.uh	ld.w	add	sub
cmp	and	or	xor	not	srl	sll
sra	sla	rr	rl	scan0	scan1	s w a p
mirror	div0s	div0u	div1	div2s	div3s	b t s t
bclr	bset	bnor	adc	sbc	mlt.h	mltu.h
mlt.w	mltu.w	mac	ext			

C33 ADVコア命令

jpr	jpr.d	pushs	pops	mac.w	mac.hw	macclr
ld.cf	div.w	divs.w	repeat	retm	swaph	s a t . b
sat.ub	mlt.hw	mac1.h	mac1.hw	mac1.w	ld.c	s a t . h
sat.uh	sat.w	sat.uw	loop	do.c	psrset	psrclr
ext						

C33 PEコア命令

push	pop	pushs	pops	jpr	jpr.d	psrclr
psrset	swaph	ld.c	do.c	ld.cf		

各命令の詳細については各コアのマニュアルを参照してください。

●文字の制限

ニーモニックは大文字(A~Z)、小文字(a~z)のどちらで記述しても受け付けられます。

たとえば、"ld.w"、"LD.W"、"Ld.w"はすべて"ld.w"命令として受け付けられます。

シンボル等と区別するため、本マニュアルでは小文字を使用します。

オペランドについては後述します。

(2) アセンブラ擬似命令

アセンブラ`as`は、GNUアセンブラに標準で用意されている擬似命令をサポートしています。標準の擬似命令についてはGNUアセンブラのマニュアルを参照してください。アセンブラ擬似命令はピリオド(.)で始まります。よく使われるアセンブラ擬似命令を以下に示します。

<code>.text</code>		<code>.text</code> セクションを宣言
<code>.section .data</code>		<code>.data</code> セクションを宣言
<code>.section .rodata</code>		<code>.rodata</code> セクションを宣言
<code>.section .bss</code>		<code>.bss</code> セクションを宣言
<code>.long</code>	<code><data></code>	4バイトデータを定義
<code>.short</code>	<code><data></code>	2バイトデータを定義
<code>.byte</code>	<code><data></code>	バイトデータを定義
<code>.ascii</code>	<code><string></code>	ASCII文字列を定義
<code>.space</code>	<code><length></code>	空白領域(0x0)を定義
<code>.zero</code>	<code><length></code>	空白領域(0x0)を定義
<code>.align</code>	<code><value></code>	指定境界アドレスにアライメント
<code>.global</code>	<code><symbol></code>	シンボルのグローバル宣言
<code>.set</code>	<code><symbol>, <address></code>	シンボルに絶対アドレスを定義

(3) ラベル

ラベルはプログラム中の任意のアドレスを参照するための識別子です。プログラムの分岐先アドレスや`.text/.data`セクション内の任意のアドレスを、ラベルとして定義したシンボルを使って参照することができます。

●ラベルの定義

次の形式で記述したシンボルがラベルとみなされます。

<シンボル>:

先頭のスペースやタブは無視されます。一般的には行の先頭から記述します。

定義されたシンボルは記述された位置のアドレスを示します。

実際のアドレス値はリンク時に決定します。

●制限事項

使用できる文字は以下のものに限られます。

A～Z a～z _ 0～9

ただし、数字で始まることはできません。大文字と小文字は区別されます。

例: `;OK` `;Error`
`FOO:` `l1label:`
`_Abcd:` `0_ABC:`
`L1:`

(4) コメント

コメントは一連のルーチン、あるいは各ステートメントの意味を記述しておくもので、コード化の対象とはなりません。

●コメントの定義

セミコロン(;)で始まり、改行で終わる文字列がコメントとみなされます。

"/*"で始まり"*/"で終わる文字列もコメントとして扱われます。

コメントにはASCIIキャラクタのほかに漢字などの非ASCIIキャラクタも使用可能です。

また、ラベルや命令と同じ行に記述することもできます。

例: ; This line is a comment line.

```

    LABEL:                ;Comment for LABEL.
                        ld.w  %a,%b  ;Comment for the instruction on the left.

/*
    This type of comment can include
    newline characters.
*/
```

●制限事項

コメントが複数行に渡る場合は、各行ごとにセミコロンで始めるか、"/*"と"*/"を使用する必要があります。

例: ;These are
comment lines. 2行目はコメントとはみなされません。エラーとなります。

 ;These are
; comment lines. 2行ともコメントとみなされます。

/*
These are
comment lines. 2行ともコメントとみなされます。
*/

(5) 空白行

本アセンブラでは、復帰/改行コードのみの空白行も許されます。一連のルーチンの区切りなど、コメント行にする必要はありません。

4.3.2 オペランドの表記

ここでは命令のオペランドに使用するレジスタ名、シンボル、定数の表記法を説明します。

(1) レジスタ名

C33コアの内部レジスタ名には、すべてパーセント(%)を付けます。レジスタ名は大文字でも、小文字でも受け付けます(区別されません)。

汎用レジスタ(%rd, %rs, %rb)	表記
汎用レジスタR0～R15	%r0～%r15または%R0～%R15

特殊レジスタ(%sd, %ss)	表記
プロセッサステータスレジスタPSR	%psrまたは%PSR
スタックポインタSP	%spまたは%SP
算術演算ローレジスタALR	%alrまたは%ALR
算術演算ハイレジスタAHR	%ahrまたは%AHR

C33 ADVコアの特殊レジスタ(%sd)	表記
ループカウンタ	%lcoまたは%LCO
ループスタートアドレスレジスタ	%lsaまたは%LSA
ループエンドアドレスレジスタ	%leaまたは%LEA
シフトアウトレジスタ	%sorまたは%SOR
ベクタテーブルベースレジスタ	%ttbrまたは%TTBR
データポインタ	%dpまたは%DP
プロセッサ識別レジスタ	%idirまたは%IDIR
デバッグベースレジスタ	%dbbrまたは%DBBR
ユーザスタックポインタ	%uspまたは%USP
スーパーバイザスタックポインタ	%sspまたは%SSP

C33 PEコアの特殊レジスタ(%sd)	表記
ベクタテーブルベースレジスタ	%ttbrまたは%TTBR
プロセッサ識別レジスタ	%idirまたは%IDIR

[]を伴うレジスタ間接アドレッシングの指定にも、レジスタ名に%が必要です。

例: [%r8] [%r1]+ [%sp+imm6]

注: レジスタ名の先頭に%がない場合、シンボルとみなされます。逆に%で始まるものはすべてレジスタとみなされ、存在しないレジスタ名を使用するとエラーとなります。

(2) 数値表記

アセンブラasは10進形式、16進形式、2進形式の3種類の数値表記に対応しています。

●10進形式の数値表記

0～9の数値のみで表記されたものは10進数とみなされます。負の数を指定する場合は、マイナス(-)符号を数値の前に付けてください。

例: 1 255 -3

0～9、および符号(-)以外の文字は使用できません。

●16進形式の数値表記

16進数を指定する場合は数値の前に"0x"を付けてください。

例: 0x1a 0xff00

"0x"の後ろには0～9、a～f、A～F以外の文字は使用できません。

●2進形式の数値表記

2進数を指定する場合は数値の前に"0b"を付けてください。

例: 0b1001 0b01001100

"0b"の後ろには0と1以外の文字は使用できません。

●数値の指定範囲

即値データのサイズ(指定範囲)は各命令により異なります。
各即値データの指定可能範囲は次のとおりです。

表4.3.2.1 即値データの種類と指定可能範囲

記号	種類	10進数	16進数	2進数
imm2	2ビット即値	0～3	0x0～0x3	0b0～0b11
imm3	3ビット即値	0～7	0x0～0x7	0b0～0b111
imm4	4ビット即値	0～15	0x0～0xf	0b0～0b1111
imm6	6ビット即値	0～63	0x0～0x3f	0b0～0b111111
sign6	符号付き6ビット即値	-32～31	0x0～0x3f	0b0～0b111111
imm8	8ビット即値	0～255	0x0～0xff	0b0～0b11111111
sign8	符号付き8ビット即値	-128～127	0x0～0xff	0b0～0b11111111
imm10	10ビット即値	0～1023	0x0～0x3ff	0b0～0b1111111111
imm13	13ビット即値	0～8191	0x0～0x1fff	0b0～0b111111111111
imm32	32ビット即値	0～4294967295	0x0～0xffffffff	0b0～0b11111111111111111111111111111111
sign32	符号付き32ビット即値	-2147483648 ～2147483647	0x0～0xffffffff	0b0～0b11111111111111111111111111111111

(3)シンボル

即値データによるアドレス指定には、他の場所で定義されているシンボルを使用することができます。

●シンボル定義

使用可能なシンボルは次のいずれかの方法によって32ビットの値として定義されます。

1. ラベルとして記述(text、dataまたはbssセクション内)

例: LABEL1:

LABEL1は.text、.dataまたは.bssセクション内の記述された位置(アドレス)を示すシンボル

2. .set擬似命令で定義

例: .set ADDR1,0xff00

ADDR1は絶対アドレス0x0000ff00を示すシンボル

●文字の制限

使用できる文字は以下のものに限られます。

A～Z a～z _ 0～9

ただし、数字で始まることはできません。大文字と小文字は区別されます。

●ローカルシンボルとグローバルシンボル

通常、定義したシンボルはローカルシンボルとなり、そのファイル内でのみ参照可能です。このため、複数のファイルで同じ名称のシンボルを定義することもできます。他のファイルで定義されているシンボルを参照するには、シンボルを定義したファイル内で、.global擬似命令によるグローバル宣言が必要です。

●シンボルの拡張表記

通常、シンボルでアドレスを参照する場合、アドレスを指定するオペランドにそのシンボル名を記述します。

例: call LABEL ← LABEL = sign8

ld.w %rd,LABEL ← LABEL = sign6

アセンブラasでは、次のようなディスプレイメント付きの参照も許可します。

LABEL + imm32 LABEL + sign32

例: call LABEL+0x10

●シンボルマスク

C33コア命令セットの基本命令は、個々の命令のオペランドで指定可能な即値サイズが限られています。このため、そのサイズを超える値を持つシンボルを使用するとアセンブラでエラーとなります。基本命令を使用する場合はext命令に上位ビットを振り分けて記述する必要があります、これにはシンボルマスクを使用します。

シンボルマスクは、シンボルの値からext命令と基本命令に振り分ける値を取得するためのもので、シンボルの直後に記述します。

拡張命令を使用する場合は、アセンブラasが必要なシンボルマスクを付けて展開しますので、特にext命令やシンボルマスクを意識することはありません。

シンボルマスクの種類

次に示す8種類のシンボルマスクが使用可能です。

シンボルマスク	機能
@rhまたは@RH	相対アドレスの上位10ビットを取得
@rmまたは@RM	相対アドレスの中間13ビットを取得
@rlまたは@RL	相対アドレスの下位8ビットを取得
@hまたは@H	絶対アドレスの上位13ビットを取得
@mまたは@M	絶対アドレスの中間13ビットを取得
@lまたは@L	絶対アドレスの下位6ビットを取得
@ahまたは@AH	相対アドレスの上位13ビットを取得
@alまたは@AL	相対アドレスの下位13ビットを取得

例:

```
ext LABEL@rh
ext LABEL@rm
call LABEL@rl      ← "call LABEL"として働きます。
```

```
ext LABEL@h
ext LABEL@m
ld.w %rd, LABEL@l   ← "ld.w %rd, LABEL"として働きます。
```

```
ext LABEL@ah
ext LABEL@al
ld.w %rd, [%rb]      ← "ld.w %rd, [%rb+LABEL]"として働きます。
```

注: ・シンボルマスクは定義されたシンボルにのみ有効です。数値にシンボルマスクを使用した場合はエラーとなります。

- ・シンボルマスクを省略すると、その命令に有効な下位側のビットが使用されます。ただし、指定可能な範囲を超えるとエラーまたはワーニングとなります。

●擬似オペランド

アセンブラ`as`には、デフォルトデータエリア開始アドレスからのオフセットアドレスを指定するための擬似オペランドが用意されており、シンボルマスクと同様に使用することができます。

擬似オペランドの種類

次に示す5種類の擬似オペランドが使用可能です。

擬似オペランド	機能
<code>doff_hi()</code> または <code>DOFF_HI()</code>	デフォルトデータエリアの先頭(<code>%r15; __dp</code>)からのオフセットアドレス上位13ビットを取得
<code>doff_lo()</code> または <code>DOFF_LO()</code>	デフォルトデータエリアの先頭(<code>%r15; __dp</code>)からのオフセットアドレス下位13ビットを取得
<code>dpoff_h()</code> または <code>DPOFF_H()</code>	デフォルトデータエリアの先頭(<code>%dp; __dp</code>)からのオフセットアドレス上位13ビットを取得 このオペランドはC33 ADVコア用で、 <code>-mc33adv</code> オプション指定時に使用可能です。
<code>dpoff_m()</code> または <code>DPOFF_M()</code>	デフォルトデータエリアの先頭(<code>%dp; __dp</code>)からのオフセットアドレス中間13ビットを取得 このオペランドはC33 ADVコア用で、 <code>-mc33adv</code> オプション指定時に使用可能です。
<code>dpoff_l()</code> または <code>DPOFF_L()</code>	デフォルトデータエリアの先頭(<code>%dp; __dp</code>)からのオフセットアドレス下位6ビットを取得 このオペランドはC33 ADVコア用で、 <code>-mc33adv</code> オプション指定時に使用可能です。

例:

```
ext doff_hi(FOO)
ext doff_lo(FOO)
ld.w %r0, [%r15]      ← "ld.w %r0, [%r15+(FOO address-__dp)]"として働きます。
```

- 注: • 擬似オペランドは定義されたシンボルにのみ有効です。数値に擬似オペランドを使用した場合はエラーとなります。
- 擬似オペランドを使用する場合は、デフォルトデータエリアポインタにデフォルトデータエリアの開始アドレスを設定しておく必要があります("3.7.1 データエリア"参照)。

4.3.3 拡張命令

拡張命令は、通常ext命令を含む複数の命令で記述する内容を1つの命令として記述できるようにしたもので、アセンブラasにより必要最小限の基本命令に展開されます。

●拡張命令の種類

C33 STDコア命令

xadd	xsub	xcmp	xand	xoor	xxor	xnot
xsll	xsrl	xsla	xsra	xrl	xrr	xld.b
xld.ub	xld.h	xld.uh	xld.w	xbset	xbclr	xbtst
xbnot	xjp	xjreq	xjrne	xjrgt	xjrge	xjrslt
xjrle	xjrugt	xjruge	xjrult	xjrle.d	xcall	xjp.d
xjreq.d	xjrne.d	xjrgt.d	xjrge.d	xjrslt.d	xjrle.d	xjrugt.d
xjruge.d	xjrult.d	xjrle.d	xcall.d			
sjp	sjreq	sjrne	sjrgt	sjrge	sjrslt	sjrle
sjrugt	sjruge	sjrult	sjrule	scall	sjp.d	sjreq.d
sjrne.d	sjrgt.d	sjrge.d	sjrslt.d	sjrle.d	sjrugt.d	sjruge.d
sjrult.d	sjrule.d	scall.d				

C33 ADVコア命令

ald.b	ald.ub	ald.h	ald.w

●拡張命令の使用方法

オペランドには即値拡張後のサイズの数値、シンボルを直接記述可能です。

```

例: xcall LABEL          ; ext LABEL@rh
                                ; ext LABEL@rm
                                ; call LABEL@rl

    xld.w %r1,sign32      ; ext sign32@h
                                ; ext sign32@m
                                ; ld.w %r1,sign32@l
  
```

また、基本命令の即値拡張機能以外にも、以下のようなオペランドの指定も命令により可能です。

```

例: xld.w %r0,[symbol + 0x10] ; R0 ← [symbol + 0x10]
    xjp LABEL + 5             ; LABEL+5のアドレスにジャンプ
    xrl %r0,15                ; R0の内容を15ビット左にローテート
  
```

オペランドを含む拡張命令の詳細については"8.7 拡張命令"を参照してください。

4.3.4 プリプロセッサ擬似命令

アセンブリソースファイル内でもCプリプロセッサ**cpp**の擬似命令を使用できます。
主な擬似命令を以下に示します。

#include	ファイルの挿入
#define	文字列および数値の定義
#if-#else-#endif	条件アセンブル

例: #include "define.h"
#define NULL 0
#ifdef TYPE1
ld.w %r0,0

#else
ld.w %r0,-1
#endif

プリプロセッサ擬似命令の詳細についてはGNU Cプリプロセッサのマニュアルを参照してください。

注: プリプロセッサ擬似命令を含むアセンブリソースはプリプロセッサを通す必要があります(**xgcc**の**-c**と**-xassembler-with-cpp**オプションで指定)、アセンブラ**as**に直接入力することはできません(直接入力するとエラーになります)。

4.4 ソース作成上の注意事項

- (1) タブストップはできるだけ4文字ごとに設定してください。4文字以外のタブ設定をしたソースをデバッガgdbでソース表示/ミックス表示すると、ソース部分がずれて出力される場合があります。
- (2) C/C++ソース、アセンブリソースをデバッグ情報付きでコンパイル/アセンブルする場合には、他のソースファイルをインクルード(#include)しないでください。デバッガgdbが正しく動作しなくなります。ソースを含まない通常のヘッダファイルは問題ありません。
- (3) C/C++モジュールとアセンブラモジュールを混在して使用する場合、引数や戻り値のサイズ、受け渡し方法には十分注意してください。
- (4) ターゲットがC33 ADVコアを使用する場合、必ず%psrレジスタのOCビットを1に初期設定してください(OCビットの初期値は0です)。OCビットが1のときに論理演算命令が実行されると、%psrのVビットが0クリアされます。C/C++コンパイラはこの動作を前提に作成されています。

ブート時の設定例:

```
asm("xld.w %r15,0x2000");           // Set SP in end of 8KB internal RAM
asm("ld.w %sp,%r15");
asm("xld.w %r15,0x00200000");       // OC=1 // 必須
asm("ld.w %psr,%r15");             // 必須
asm("xld.w %r15,__dp");             // Set DP (default data area)
asm("ld.w %dp,%r15");
```

- (5) 関数は必ずプロトタイプ宣言もしくはextern宣言を行ってください。
プロトタイプ宣言もしくはextern宣言がされてなく、かつ、同じファイル内の前方に定義部のない関数をコールした場合、関数をコールする側で想定している型と、実際にコールされる関数の型が一致せずに誤動作する可能性があります。この場合でも、コンパイル時にエラーは発生しません。ただし、同じファイル内にコールされる関数の定義部がある場合には、ワーニングが発生します。他のファイル内にコールされる関数の定義部がある場合は、-Wallオプションを付けない限りワーニングも発生しません。
戻り値に関しては、この場合、暗黙のうちにint型と判断されます。そのため、int型より大きい型の戻り値は正しく取得できなくなります。

例:

```
long long ll_Val_1 = 0x1122334455667788LL;
long long ll_Val_2;

int main()
{
    ll_Val_2 = sub();           // ll_Val_2の値が0x55667788になってしまいます。
    return 0;
}

long long sub()
{
    long long ll_wk;

    ll_wk = ll_Val_1;
    return ll_wk;
}
```

- (6) アドレス不整例外が発生しますので、char以外のポインタを使用して奇数番地のメモリをリード/ライトしないでください。

例:

```
int *ip_Pt;

void sub()
{
    ip_Pt = (int *)0x3;
    (*ip_Pt) = 0x2;           // ここでアドレス不整例外が発生します。
}
```


- (7) C/C++ソースソースでは関数名を関数へのポインタとしても使用できますが、関数名を使用してその値を実数型(float/double)の変数や配列に代入することはできません。

整数型の変数や配列には代入できます。

ただし、Cソースではグローバル変数またはグローバル配列の宣言時に関数名で値を代入する場合は、int/unsigned int/long/unsigned long型のグローバル変数/配列にのみ、関数名による代入が行えます。

宣言時以外であれば、任意の整数型のグローバル変数/配列に関数名で値を代入することができます。

ローカル変数またはローカル配列の場合は、整数型であれば宣言時か否かにかかわらず、いつでも関数名で値を代入することができます。

例:

- 1) long l_Global_Val = (long)boot;
→ long型のグローバル変数l_Global_Valの宣言時に、関数名による代入は行えます。
- 2) short s_Global_Val = (short)boot;
→ short型のグローバル変数s_Global_Valの宣言時に関数名による代入を行うと、エラーが発生します。
error: initializer element is not constant
C++ソースでは、エラーの代わりに以下のワーニングが発生します。
warning: cast from pointer to integer of different size
- 3) long l_local_val = (long)boot;
→ long型のローカル変数l_local_valへの関数名による代入は行えます。
- 4) short s_local_val = (short)boot;
→ short型のローカル変数s_local_valへの関数名による代入は行えます。
- 5) char c_Global_Val;
void sub()
{
 c_Global_Val = (char)boot;
}
→ 宣言時以外であれば、char型のグローバル変数c_Global_Valへの関数名による代入は行えます。

- (8) C/C++ソースでは関数ポインタも使用できますが、この関数ポインタも(7)と同様に実数型(float/double)の変数や配列には代入できません。

整数型の変数や配列には代入できます。

ただし、Cソースではグローバル変数またはグローバル配列の宣言時は関数ポインタを代入することはできません。

宣言時以外であれば、整数型のグローバル変数/配列に関数ポインタを代入することができます。

整数型のローカル変数またはローカル配列の場合は、宣言時か否かにかかわらず、関数ポインタを代入することができます。

int/unsigned int/long/unsigned long型以外の変数/配列に関数ポインタを代入するとワーニングが発生します。

例:

void (* fp_Pt)(void); // 戻り値と引数の型が共にvoid型の関数ポインタの宣言

- 1) long l_Global_Val = (long)fp_Pt;
→ グローバル変数l_Global_Valの宣言時に関数ポインタを代入すると、エラーが発生します。
error: initializer element is not constant
(C++ソースでは、エラーは発生しません)
- 2) long l_local_val = (long)fp_Pt;
→ long型のローカル変数l_local_valへの関数ポインタの代入は行えます。
- 3) short s_local_val = (short)fp_Pt;
→ short型のローカル変数s_local_valへの関数ポインタの代入はできますが、ワーニングが発生します。
warning: cast from pointer to integer of different size

4 ソースファイル

```
4) long l_Global_Val;
    void sub()
    {
        l_Global_Val = (long)fp_Pt;
    }
```

→ 宣言時以外であれば、long型のグローバル変数s_Global_Valへの関数ポインタの代入は行えます。

- (9) C/C++言語の規格上、動作が未定義の処理を行った場合、最適化オプション(-O0/-O/-O2/-O3/-Os)や、ローカル変数/外部変数などの違いにより演算結果が異なる場合がありますので、注意してください。

未定義の処理としては、以下のケースなどが挙げられます。

- ・ 浮動小数点型→整数型に型変換した時に、オーバーフローしていた場合。
- ・ シフト演算を負数もしくは、型昇格された後の演算対象のビット長と同じかより大きい値で行う場合。

- (10) C言語の規格上、非互換の型のポインタ経由で変数にアクセスしようとした時に-Wallオプションが付いている場合、以下のワーニングが表示されることがあります。

この場合、ポインタを経由した変数への参照及び代入は正しく行われない場合があります。

warning: dereferencing type-punned pointer will break strict-aliasing rules

例: -O3 オプションが指定されている場合

```
int sub()
{
    int p1 = 0 ;
    short *p2 = (short *)&p1 ;
```

p2と&p1は非互換の型のため、ワーニングが発生します。

この場合、ポインタp2を経由した変数p1への参照及び、代入は正しく行われない場合があります。

5 GNU33 IDE

この章では、GNU33 IDEの機能と操作方法を説明します。

5.1 概要

5.1.1 特長

GNU33 IDE(以下IDE)は、S1C33 Family C/C++コンパイラパッケージ(S5U1C33001C)を使用したプログラムの開発を支援する統合開発環境を提供します。

IDEの主な機能は次のとおりです。

- **プロジェクトの管理**
1つのアプリケーションの作成に必要な全ソースファイルなどをプロジェクトとして一元管理できます。
- **GNU準拠のC、C++、およびアセンブラをサポート**
GNU準拠のC、C++、およびアセンブリ言語でのソース作成/編集機能があります。他のエディタで作成したソースも読み込み可能です。
- **makeファイルの作成と実行機能**
コンパイラからリンカまでの実行シーケンスが記述されたmakeファイルを、ビルドオプションを選択することで自動的に生成します。このファイルをもとに実行形式のオブジェクトファイルを生成するビルド処理は1ボタンで行えます。
- **各種定義ファイルの作成機能**
上記のmakeファイル以外にビルドに必要なリンカスクリプトファイル、デバッグの起動に必要なパラメータファイルやコマンドファイルが、簡単な操作により編集/作成できます。
- **デバッグgdbを呼び出すためのランチャー機能**
ビルド処理終了後、デバッグgdbを呼び出し、デバッグを行うことができます。

5.1.2 IDE使用上の注意

●IDEの動作保証について

IDEは開発プラットフォームEclipse上で動作し、Eclipseの機能も利用しています。ただし、IDEでは、本マニュアルに記載されていない機能については動作保証対象外としていますのでご注意ください。

●Eclipseプラグインのバージョン

IDEが依存するEclipseプラグインとそのバージョンは以下のとおりです。

表5.1.2.1 Eclipseプラグインのバージョン

プラグイン	バージョン
Eclipse Platform	3.4.0
Eclipse CDT	5.0.0
Eclipse DSDP メモリビュー	1.1.0

これらのプラグインを変更、削除、またはアップデートした場合、IDEの動作は保証できませんのでご注意ください。また、IDEはJavaアプリケーションであり、Java仮想マシンもインストールされます。これ以外のJava仮想マシン上でのIDEの動作も保証できませんのでご注意ください。

●IDE上での日本語の使用について

IDE上では、ファイルやフォルダ名、ファイル内の文字列に日本語(文字コードShift-JIS/MS-932)を使用可能ですが、ビルドなどに使用するGNU33ツールは日本語をサポートしておりません。したがって、ファイルやフォルダ名、およびソースファイル等の実行行では日本語を使用しないでください。(ソースファイル内のコメント行のみ日本語を使用可能です。)

●IDEのユーザーインターフェースの日本語表示について

IDEでは、Eclipseの翻訳プラグインblancoプラグインをパッケージングしています。

これにより、IDEのユーザーインターフェース(メニュー・ダイアログ・エラーメッセージ)が日本語で表示されます。(一部を除く)

通常、IDEを起動したときにOSの地域と言語オプションの設定により自動的にユーザーインターフェースの表示が切り替わります。(日本語OS上は日本語表記になり、その他のOSでは英語表記になります。)

IDEを以下のように起動すると、手動で英語/日本語を切り替えることも可能です。

eclipse.exe -nl en_US : 英語

eclipse.exe -nl ja_JP : 日本語

5.2 IDEの起動と終了

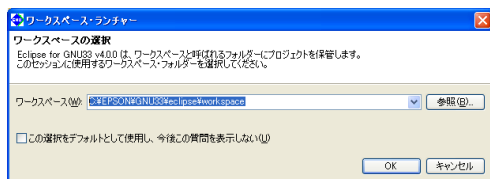
5.2.1 起動方法

IDEの起動方法は以下のとおりです。

- (1)  `c:\¥EPSON¥gnu33¥eclipse`ディレクトリ内の`eclipse.exe`のアイコンをダブルクリックし、**IDE**を起動させます。

Windowsのスタートアップメニューから[EPSON MCU] > [GNU33] > [GNU33 IDE]を選択しても起動します。また、コマンドラインからも引数なしで起動可能です。

- (2) Eclipseのスプラッシュに続き、[ワークスペース・ランチャー]ダイアログが表示されます。ここではプロジェクトの格納などに使用する作業用ディレクトリ(ワークスペース)を指定します。



`c:\¥EPSON¥gnu33¥eclipse¥workspace`がデフォルトディレクトリとして表示されますが、任意のディレクトリを選択または新規作成してワークスペースに設定可能です。[ワークスペース:]コンボボックスにディレクトリ名を入力するか、[参照...]ボタンで表示されるディレクトリ選択ダイアログで選択してください。

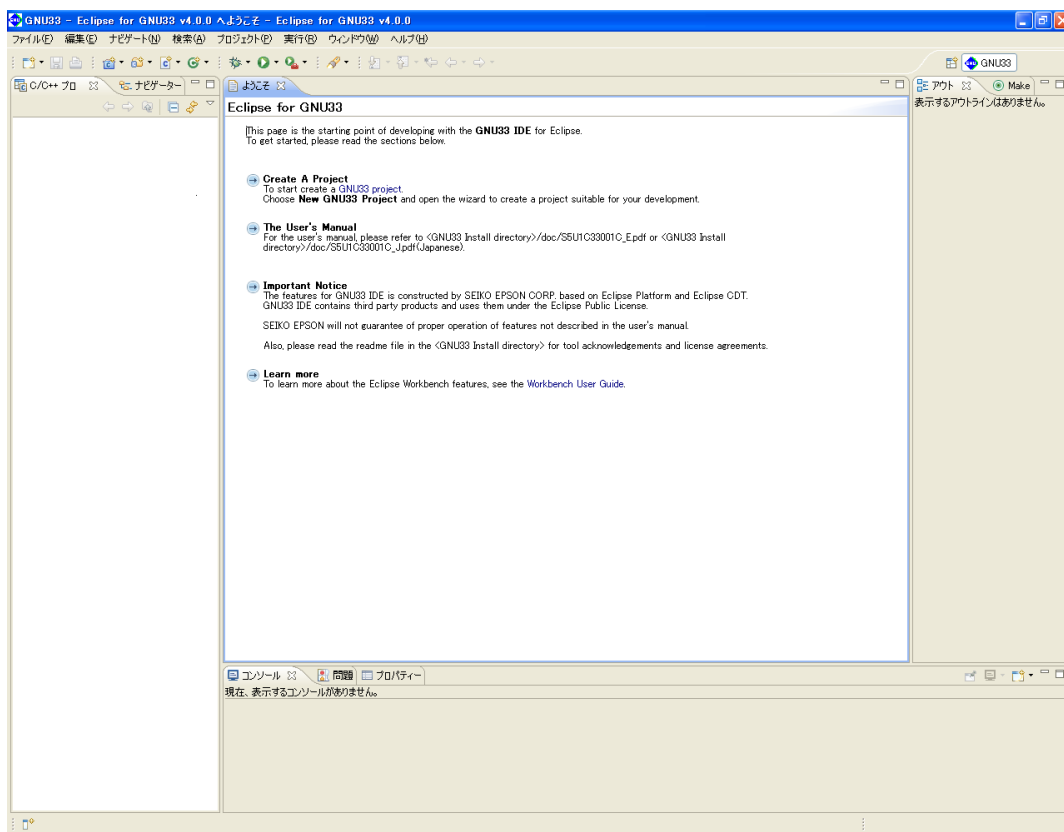
[ワークスペース・ランチャー]ダイアログは**IDE**を起動するたびに表示されます。今後、同じワークスペースで作業する場合は、[この選択をデフォルトとして使用し、今後この質問を表示しない]をチェックすることで、[ワークスペース・ランチャー]ダイアログを開かなくすることができます(ワークスペースは、起動後でも[ファイル]メニューの[ワークスペースの切り替え...]で変更可能です)。

※ワークスペースディレクトリには、プロジェクトディレクトリ(.projectファイルが含まれるディレクトリ)を指定しないでください。プロジェクトインポート(プロジェクトをワークスペースにコピーがONのとき)に失敗することがあります。

現在のワークスペースディレクトリは、[ファイル]-[ワークスペースの切り替え]-[その他]よりワークスペース・ランチャーダイアログで確認できます。

(3)[OK]ボタンをクリックします。

IDEのウィンドウが表示されます。

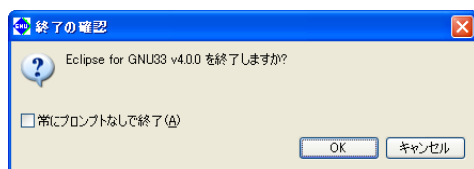


5.2.2 終了方法

IDEを終了させるには、[ファイル]メニューから[終了]を選択してください。

エディタで開いているファイルが保存されていない場合は、保存するか否かを選択するダイアログが表示されます。いずれかを選択して終了させてください。

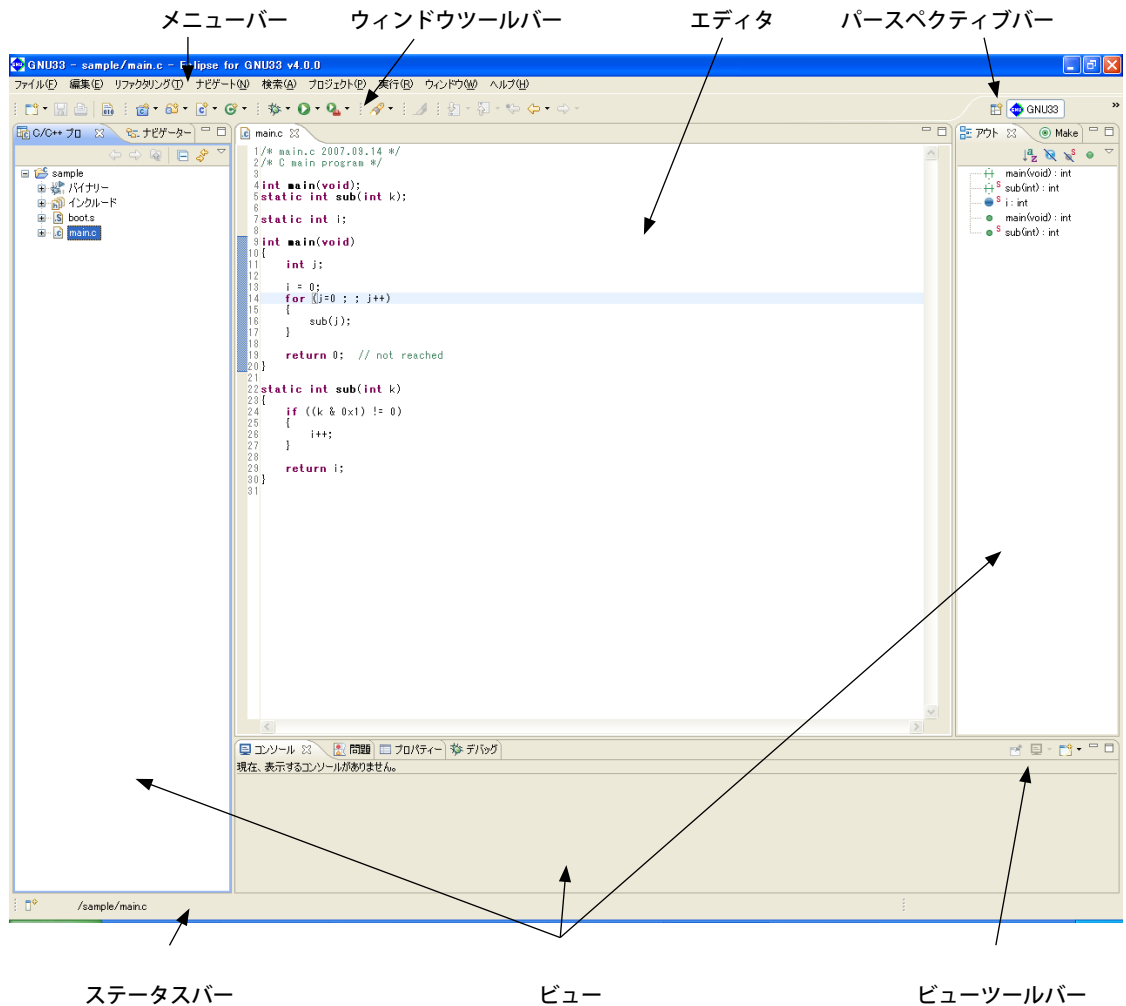
ウィンドウの[閉じる]ボタンでも終了可能です。ただし、この場合は次のダイアログが表示されますので、終了する場合は[OK]ボタンを、終了を取り消す場合は[キャンセル]ボタンをクリックします。



[常にプロンプトなしで終了]をチェックしておくと、以後はこのダイアログが表示されなくなります。

5.3 IDEウィンドウ

IDEのウィンドウはエディタを中心に、それを取り囲むいくつかのビュー、およびメニューバーとツールバーで構成されています。



5.3.1 メニューバー

ファイル(F) 編集(E) リファクタリング(R) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)

●[ファイル]メニュー

ファイル(F)	
新規(N)...	Alt+Shift+N ▶
ファイルを開く(O)...	
閉じる(C)	Ctrl+W
すべて閉じる(L)	Ctrl+Shift+W
保存(S)	Ctrl+S
別名保管(A)...	
すべて保管(E)	Ctrl+Shift+S
前回保管した状態に戻す(T)	
移動(V)...	
名前変更(H)...	
リフレッシュ(F)	F5
行区切り文字の変換(Y)	▶
印刷(P)...	Ctrl+P
ワークスペースの切り替え(W)	▶
再始動	
インポート(I)...	
エクスポート(E)...	
プロパティ(R)	Alt+Enter
1 main.c [sample]	
2 boots [sample]	
終了(X)	

新規 (Alt+Shift+N)

新規 GNU33 プロジェクト	
プロジェクト(O)...	
ソース・ファイル	
ヘッダー・ファイル	
ソース・フォルダー	
テンプレートからファイル	
その他(O)...	Ctrl+N

新規 GNU33 プロジェクト

GNU33の新規プロジェクト作成ウィザードを開始します。

プロジェクト...

新規プロジェクト作成ウィザードを選択します。表示されるダイアログには複数のウィザードが登録されていますが、ここでは[GNU33 プロジェクト]内の[新規 GNU33 プロジェクト]を選択してください。

ソース・ファイル

ソースファイルを新規作成します。

ヘッダー・ファイル

ヘッダファイルを新規作成します。

ソース・フォルダー

ソースフォルダを新規作成します。

テンプレートからファイル

テキストファイルを新規作成します。

その他...

[プロジェクト...]と同じ機能です。

ファイルを開く...

エディタで開くファイルを選択します。

閉じる (Ctrl+W)

エディタが最前面に開いているファイルを閉じます。最新の編集内容が保存されていない場合は、保存するか否かを選択するダイアログが表示され、そのいずれか、あるいは処理の中止を選択できます。

すべて閉じる (Ctrl+Shift+W)

エディタが開いているすべてのファイルを閉じます。最新の編集内容が保存されていない場合は、保存するファイルを選択するダイアログが表示され、その選択を行うか、あるいは処理を中止できます。

保管 (Ctrl+S)

エディタが最前面に開いているファイルを保存します。最新の内容が保存済みの場合は無効です。

別名保管...

エディタが最前面に開いているファイルを別名で、あるいは別の場所に保存します。

すべて保管 (Ctrl+Shift+S)

エディタが開いているすべてのファイルを保存します。

前回保管した状態に戻す

エディタが最前面に開いているファイルの編集内容を破棄し、前回の保存内容に戻します。

移動...

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで選択されているファイルまたはフォルダを別の場所に移動します。

名前変更.. (F2)

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで選択されているファイル名またはフォルダ名を編集モード(名前が変更できる状態)にします。

リフレッシュ (F5)

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューの表示内容を最新の状態に更新します。

行区切り文字の変換

行の区切り文字を選択します。

印刷... (Ctrl+P)

エディタが最前面に開いているファイルを印刷します。

ワークスペースの切り替え...

別のワークスペースを選択します。新しいワークスペースを開く前に現在のワークスペース情報が保存され、エディタが開いているファイルに対しても[すべて保管]の処理が実行されます。

※ ワークスペースディレクトリには、プロジェクトディレクトリ(.projectファイルが含まれるディレクトリ)を指定しないでください。

再始動

IDEを再起動します。

インポート...

既存のプロジェクトやソースファイルを現在のワークスペースやプロジェクトに追加するウィザードが起動します。

エクスポート...

現在のプロジェクト内のファイルを別のディレクトリに書き出します。

プロパティ (Alt+Enter)

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで選択されているプロジェクト、ファイルまたはフォルダのプロパティを表示/編集するダイアログを開きます。

終了

IDEを終了します。

●[編集]メニュー

編集(E)	
元に戻す(U)	Ctrl+Z
やり直し(B) 入力	Ctrl+Y
切り取り(T)	Ctrl+X
コピー(C)	Ctrl+C
貼り付け(P)	Ctrl+V
削除(D)	Delete
すべて選択(A)	Ctrl+A
検索/置換(F)...	Ctrl+F
Find Word	
次を検索(N)	Ctrl+K
前を検索(B)	Ctrl+Shift+K
次をインクリメンタル検索(Q)	Ctrl+J
前をインクリメンタル検索(M)	Ctrl+Shift+J
ブックマークの追加(S)...	
タスクの追加(S)...	
✓ スマート挿入モード(B)	Ctrl+Shift+Insert
ツールチップ記述の表示(Q)	F2
単語補完(W)	Alt+/
クイック・ファックス(Q)	Ctrl+I
コンテンツ・アシスト(N)	Ctrl+Space
パラメーター・ヒント(H)	Ctrl+Shift+Space
右へシフト(S)	
左へシフト(B)	Shift+Tab
フォーマット(O)	Ctrl+Shift+F
Include の追加	Ctrl+Shift+N
エンコードの設定(Q)...	

元に戻す (Ctrl+Z)

直前に行ったエディタ上の操作を取り消します。

やり直し (Ctrl+Y)

[元に戻す]で取り消した操作を元に戻します。

切り取り (Ctrl+X)

選択した文字列やファイル/フォルダを削除するとともにペーストボードにコピーします。

コピー (Ctrl+C)

選択した文字列やファイル/フォルダをペーストボードにコピーします。

貼り付け (Ctrl+V)

クリップボードにコピーされている内容をエディタ内のカーソル位置、あるいは現在のビュー内にペーストします。

削除 (Delete)

選択された文字列やファイル/フォルダを削除します。

すべて選択 (Ctrl+A)

エディタ上の現在アクティブな文書の内容をすべて選択します。

検索/置換... (Ctrl+F)

エディタ上で文字列の検索と置き換えを行います。

Find Word

選択文字列の次の一致を検索します。

次を検索 (Ctrl+K)

検索時に次の候補にジャンプします。

前を検索 (Ctrl+Shift+K)

検索時に前の候補にジャンプします。

次をインクリメンタル検索 (Ctrl+J)

本コマンドを選択して検索する文字列をタイプすると、エディタ上の現在アクティブな文書内から、入力した文字列を検索します(現在のカーソル位置より後方を検索)。1文字タイプするごとに再検索されます。[↑]または[↓]キーで現在の検索内容の次候補にジャンプします。この検索モードは[←]、[→]、[Enter]、または[Esc]キーを押すことにより解除されます。

前をインクリメンタル検索

本コマンドを選択して検索する文字列をタイプすると、エディタ上の現在アクティブな文書内から、入力した文字列を検索します(現在のカーソル位置より前方を検索)。1文字タイプするごとに再検索されます。[↑]または[↓]キーで現在の検索内容の次候補にジャンプします。この検索モードは[←]、[→]、[Enter]、または[Esc]キーを押すことにより解除されます。

ブックマークの追加...

エディタ上のアクティブな文書内で現在カーソルがある行をブックマークとして登録します。ブックマークの詳細は"5.5.6 ブックマーク"を参照してください。

タスクの追加...

エディタ上のアクティブな文書内で現在カーソルがある行をタスク(To Do項目)として登録します。登録したタスクは[タスク]ビューで管理できます。

スマート挿入モード (Ctrl+Shift+Insert)

エディタのスマート挿入モードを切り替えます。

ツールチップ記述の表示 (F2)

ツールチップ表示中に[F2]キーを押下すると、ツールチップがフォーカスされます。

単語補完 (Alt+/)

エディタで入力中の文字で始まる単語を現在位置に挿入します。最近入力した単語が選択されます。

クイック・フィックス (Ctrl+1)

エラーやワーニングの修正候補を表示します。

コンテンツ・アシスト (Ctrl+Space)

ダイアログが表示され、そこから選択したCソースの予約語やテンプレートなどをエディタ上の現在のカーソル位置に挿入します。Cソースの編集時にのみ有効です。

パラメータ・ヒント (Ctrl+Shift+Space)

関数引数のヒントを表示します。

右ヘシフト

行の先頭をタブ1個分右にシフトします。

左ヘシフト(Shift+Tab)

行の先頭をタブ1個分左にシフトします。

フォーマット (Ctrl+Shift+F)

フォーマットの設定にしたがってテキストをフォーマットします。

エンコードの設定...

テキストのエンコード形式を選択します。

●[リファクタリング]メニュー

リファクタリング(R)	
名前変更(N)...	Alt+Shift+R
定数の抽出(A)...	Alt+C
関数の抽出(F)...	Alt+Shift+M
メソッドの隠蔽...	
メソッドの実装(E)...	
Getter および Setter の生成...	

名前変更.. (Alt+Shift+R)

選択した関数や変数の名称を一括して変更します。

定数の抽出 (Alt+C)

ソースファイルの定数を変数に切り出します。

関数の抽出 (Alt+Shift+M)

ソースファイルのコードの一部を関数に切り出します。

●[ナビゲート]メニュー

ナビゲート(N)	
次へジャンプ(N)	
ジャンプ(Q)	
宣言を開く(O)	F3
型階層を開く(E)	F4
呼び出し階層を開く(I)	Ctrl+Alt+H
組み込みブラウザーを開く(L)	Ctrl+Alt+I
ソース/ヘッダーの切り替え(S)	Ctrl+Tab
表示(W)	Alt+Shift+W
クイック・アウトライン(L)	Ctrl+O
次の注釈(N)	Ctrl+.
前の注釈(N)	Ctrl+,
最後の編集位置(O)	Ctrl+Q
指定行へジャンプ(Q)...	Ctrl+L
戻る(B)	Alt+←
進む(F)	Alt+→

次へジャンプ

[C/C++ プロジェクト]または[ナビゲーター]ビューを、現在選択しているフォルダ内のみを表示するように切り換えます。

ジャンプ

戻る(B)
進む(F)
1 つ上のレベルへ(L)

次のメンバー(E)
直前のメンバー(O)
対応する大括弧(B)
Next Bookmark

戻る

[C/C++ プロジェクト]または[ナビゲーター]ビューを、一つ前の表示に戻します。

進む

上記の[ジャンプ]>[戻る]で遡った表示を一つ新しい状態に戻します。

1 つ前のレベルへ

[C/C++ プロジェクト]または[ナビゲーター]ビューを、一つ上の階層が表示されるようにします。

次のメンバー (Ctrl+Shift+Up)

一つ後の関数/変数定義位置にジャンプします(Cエディタのみ)。

直前のメンバー (Ctrl+Shift+Down)

一つ前の関数/変数定義位置にジャンプします(Cエディタのみ)。

対応する大括弧 (Ctrl+Shift+P)

対応するカッコにジャンプします(Cエディタのみ)。

Next Bookmark

一つ後のブックマークにジャンプします(Cエディタのみ)。

宣言を開く (F3)

選択しているオブジェクトの宣言・定義を開きます(インデクサON時)。

型階層を開く (F4)

選択している変数の型階層を開きます(インデクサON時)。

呼び出し階層を開く (Ctrl+Alt+H)

選択している関数の呼び出し階層を開きます(インデクサON時)。

組み込みブラウザーを開く (Ctrl+Alt+I)

選択しているソースファイルのインクルード階層を開きます(インデクサON時)。

ソース/ヘッダーの切り替え (Ctrl+Tab)

対応するソースファイルとヘッダファイルをエディタで切り替えます。

表示

選択したエレメント(関数名、変数名、型)を含むリソースをハイライト表示するビュー(エディタ以外で可能なビューがある場合のみ)を選択します。

次の注釈 (Ctrl+.)

[問題]や[検索]ビューなどに表示される一覧内の選択を次の項目に進めます。

前の注釈 (Ctrl+,)

[問題]や[検索]ビューなどに表示される一覧内の選択を前の項目に戻します。

最後の編集位置 (Ctrl+Q)

エディタ内の最後に編集した位置にジャンプします。

指定行へジャンプ... (Ctrl+L)

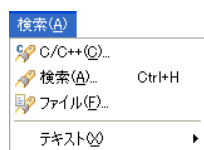
行番号を指定して、エディタ上のアクティブな文書内の指定位置にジャンプします。

戻る (Alt+Left)

エディタ上の文書内の移動も含め、一つ前に参照/編集していた位置に戻ります。

進む (Alt+Right)

上記の[戻る]で遡った表示を一つ新しい状態に戻します。

●[検索]メニュー**検索... (Ctrl+H)**

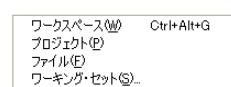
ファイル検索、C検索を行う[検索]ダイアログを表示します。

ファイル...

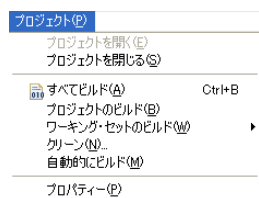
指定の文字列を含むファイルを検索します。([検索]ダイアログのファイル検索ページを表示します。)

C/C++...

指定の文字列を含むCソースを検索します。([検索]ダイアログのC検索ページを表示します。)

テキスト

現在カーソルが置かれている文字列を、サブメニューで選択した範囲(ワークスペース、現在のプロジェクト、現在のファイル、または指定のワーキングセット)から検索します。

●[プロジェクト]メニュー**プロジェクトを開く**

[C/C++ プロジェクト]または[ナビゲーター]ビューで選択されている閉じた状態のプロジェクトを開きます。

プロジェクトを閉じる

[C/C++ プロジェクト]または[ナビゲーター]ビューで選択されているプロジェクトを閉じます。

すべてビルド (Ctrl+B)

[C/C++ プロジェクト]または[ナビゲーター]ビュー上で開いているプロジェクトすべてのビルドを実行します。

プロジェクトのビルド

[C/C++ プロジェクト]または[ナビゲーター]ビュー上で選択されているプロジェクトのビルドを実行します。

ワーキング・セットのビルド

指定のワーキングセットに含まれるリソースのビルドを実行します。

クリーン...

すべてのリソースからビルドを再実行させるため、前回のビルド時に生成されたファイルをすべて削除します。

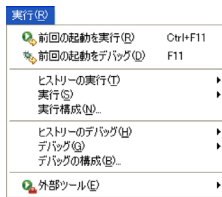
自動的にビルド

オートビルド機能をON/OFFします。オートビルドは、エディタで編集しているソースを保存することにより自動的にビルドを実行する機能ですが、**IDE**では使用できません。

プロパティ

[C/C++ プロジェクト]または[ナビゲーター]ビュー上で選択されているプロジェクトのプロパティを表示/編集する[プロパティ]ダイアログを表示します。

●[実行]メニュー



前回の起動を実行

サポートとしません。

前回の起動をデバッグ

前回起動した構成でデバッグを開始します。

履歴の実行

サポートとしません。

実行

サポートとしません。

実行構成...

サポートとしません。

履歴のデバッグ

サブメニューに最近起動したデバッグ構成のショートカットを表示します。

デバッグ

サポートとしません。

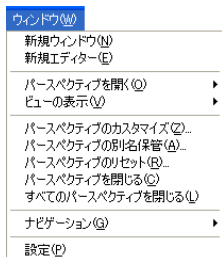
デバッグの構成...

デバッグがgdbの起動構成ダイアログを開きます。

外部ツール

サポートとしません。

●[ウィンドウ]メニュー



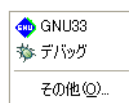
新規ウィンドウ

現在選択されているパースペクティブの初期設定のビューレイアウトで新しいウィンドウを開きます。現在開いているプロジェクトもそのまま新しいウィンドウに継承されます。

新規エディター

現在編集中的のファイルを新しいエディタタブで開きます。

パースペクティブを開く



GNU33

GNU33パースペクティブを開きます。

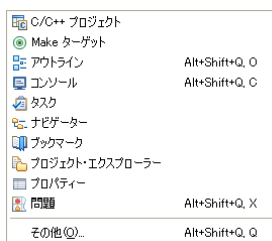
デバッグ

デバッグパースペクティブを開きます。

その他...

その他のパースペクティブを開きます。

ビューの表示



サブメニューで選択したビューを開きます。すでに開いている場合は、ビューがアクティブになります。

パースペクティブのカスタマイズ...

現在のパースペクティブに定義されているツールバーのショートカットやメニューコマンドの設定を変更します。

パースペクティブの別名保管...

現在のパースペクティブの設定を別名で保存します。

パースペクティブのリセット

パースペクティブを初期状態に再設定し、変更したビューのレイアウトなどを元に戻します。

パースペクティブを閉じる

現在アクティブなパースペクティブを閉じます。

すべてのパースペクティブを閉じる

読み込まれているすべてのパースペクティブを閉じます。

ナビゲーション

「システム・メニュー」を表示(S)	Alt+-
「ビューの表示」メニュー(V)	
クイック・アクセス(Q)	Ctrl+3
アクティブ・ビューまたはエディターの最大化 アクティブ・ビューまたはエディターの最小化	Ctrl+M
エディターをアクティブにする(A)	F12
次のエディター(E)	Ctrl+F6
前のエディター(P)	Ctrl+Shift+F6
エディターの切り替え(W...)	Ctrl+Shift+E
次のビュー(O)	Ctrl+F7
前のビュー(Q)	Ctrl+Shift+F7
次のパースペクティブ(B)	Ctrl+F8
前のパースペクティブ(U)	Ctrl+Shift+F8

「システム・メニュー」を表示 (Alt+-)

現在アクティブなビューやエディタに有効なシステムメニュー（高速ビュー、リサイズ、クローズなど）を表示します。

「ビューの表示」メニュー (Ctrl+F10)

現在アクティブなビューのビューメニューを表示します。

アクティブ・ビューまたはエディターの最大化 (Ctrl+M)

現在アクティブなビューまたはエディタを最大化します。すでに最大化している場合は元のサイズに戻ります。

アクティブ・ビューまたはエディターの最小化

現在アクティブなビューまたはエディタを最小化します。

エディターをアクティブにする (F12)

エディタで現在最も前面に表示されている文書をアクティブにします。

次のエディター (Ctrl+F6)

エディタでアクティブにする文書(デフォルト: 使用履歴で現在アクティブな文書の次に開いた文書)を選択します。

前のエディター (Ctrl+Shift+F6)

エディタでアクティブにする文書(デフォルト: 使用履歴で現在アクティブな文書の前に開いた文書)を選択します。

エディターの切り替え... (Ctrl+Shift+E)

エディタでアクティブにする文書をダイアログから選択します。

次のビュー (Ctrl+F7)

アクティブにするビュー(デフォルト: 使用履歴で現在のビューの次に開いたビュー)を選択します。

前のビュー (Ctrl+Shift+F7)

アクティブにするビュー(デフォルト: 使用履歴で現在のビューの前に開いたビュー)を選択します。

次のパースペクティブ (Ctrl+F8)

アクティブにするパースペクティブ(デフォルト: 使用履歴で現在アクティブなパースペクティブの次に開いたパースペクティブ)を選択します。

前のパースペクティブ (Ctrl+Shift+F8)

アクティブにするパースペクティブ(デフォルト: 使用履歴で現在アクティブなパースペクティブの前に開いたパースペクティブ)を選択します。

設定...

IDEの環境設定を行う[設定]ダイアログを表示します。

●[ヘルプ]メニュー

ヘルプ(H)	
ヘルプ目次(H)	
検索(E)	
ダイナミック・ヘルプ(D)	
キー・アシスト(K)...	Ctrl+Shift+L
ヒント(I)...	
Cheat Sheets...	
ようこそ	
ソフトウェア更新...	
Eclipse for GNU33 v4.0.0 について(A)	

ヘルプ目次

ヘルプコンテンツをブラウザに表示します。

検索

ヘルプトピックの検索ビューを表示します。

ダイナミック・ヘルプ

現在アクティブなビューに関連するヘルプトピックを表示します。

キー・アシスト... (Ctrl+Shift+L)

現在使用可能なメニューコマンドの一覧を表示します。

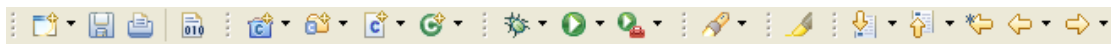
ソフトウェア更新...

アップデートのインストールやプラグインの更新などを管理します。通常は操作しないでください。

Eclipse for GNU33 Vx.xについて

IDEのバージョンやプラグインの詳細などを表示します。

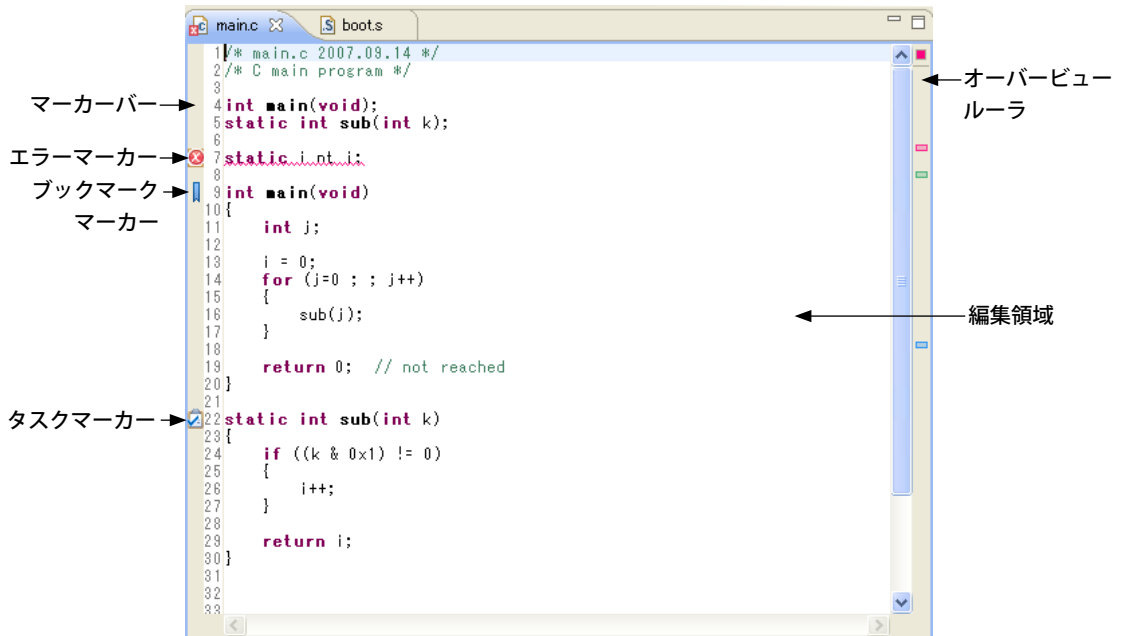
5.3.2 ウィンドウツールバー



ここでは、ウィンドウメニューの中からよく使用するコマンドをショートカットとして配置しています。各ボタンの機能については、メニューバーの説明を参照してください。

	新規	= [ファイル]>[新規]
	保管	= [ファイル]>[保管]
	印刷	= [ファイル]>[印刷...]
	すべてビルド	= [プロジェクト]>[すべてビルド]
	新規 C/C++ プロジェクト	= [ファイル]>[新規]>[新規 GNU33 プロジェクト]
	新規 C/C++ ソース・フォルダー	= [ファイル]>[新規]>[ソース・フォルダー]
	新規 C/C++ ソース・ファイル	= [ファイル]>[新規]>[ソース・ファイル]
	新規 C/C++ クラス	(新規クラスを作成する。)
	デバッグ	= [実行]>[ヒストリーのデバッグ]>[前回起動した構成]
	実行	未サポート
	外部ツール	= [実行]>[外部ツール]
	検索	= [検索]>[検索...]
	次の注釈	= [ナビゲート]>[次の注釈]
	前の注釈	= [ナビゲート]>[前の注釈]
	最後の編集位置	= [ナビゲート]>[最後の編集位置]
	戻る	= [ナビゲート]>[戻る]
	進む	= [ナビゲート]>[進む]

5.3.3 エディタ領域



ここは、ソースなどを編集するエディタの領域です。**IDE**には、C/C++ソース用、アセンブラソース用のエディタが組み込まれています。これらのエディタは汎用のエディタと同様の編集機能があり、他のビューに表示されるエラーメッセージや変数/関数名ともリンクします。一度に複数の文書を開くことができ、それぞれの文書は文書名が表示される上部のタブで選択します。

エディタ領域左端のマーカーバーは、エラー発生行、ブックマークやタスクを設定した行などを示すマーカーを表示します。マーカーにマウスポインタを重ねると、エラーの内容やブックマーク名、タスクの説明などが現れます。

右端のオーバービュールーラも、マーカーバーと同様にエラー発生位置やブックマーク/タスク設定位置を四角のシンボルで示します。こちらは現在の表示位置とは対応しておらず、ファイル全体から見た位置関係を示します。シンボルにマウスを重ねるとマーカーと同様に説明などが現れ、シンボルをクリックすることで、その位置にジャンプします。

組み込みのエディタ以外にも、**IDE**から外部エディタを起動してソースを編集することもできます。編集機能については、「5.5 エディタとソースファイルの編集」を参照してください。

●コンテキストメニュー

編集領域内を右クリックすると、次のコンテキストメニューが表示されます(特に説明のないメニューコマンドについては、メニューバーの説明を参照してください)。

元に戻す(U) 入力	Ctrl+Z
ファイルを前回保管した状態に戻す(U)	
保管(S)	Ctrl+S
宣言を開く(O)	F3
型階層を開く	F4
呼び出し階層を開く(P)	Ctrl+Alt+H
クイック・アウトライン(L)	Ctrl+O
クイック型階層	Ctrl+T
マクロ拡張の探索(M)	Ctrl+=
ソース/ヘッダーの切り替え(Q)	Ctrl+Tab
表示	Alt+Shift+W(W) ▶
切り取り(T)	Ctrl+X
コピー(C)	Ctrl+C
貼り付け(P)	Ctrl+V
クイック・フィックス(Q)	Ctrl+I
ソース	▶
リファクタリング	▶
宣言(D)	▶
参照(R)	▶
検索テキスト	▶
実行(E)	▶
デバッグ(D)	▶
チーム(T)	▶
比較(A)	▶
置換(L)	▶
オブジェクトファイル変換	▶
設定..	
構成のビルド	▶
Make ターゲット	▶

元に戻す = [編集]>[元に戻す]

ファイルを前回保管した状態に戻す

編集中の文書を前回ファイル保存した内容に戻します。

保管 = [ファイル]>[保管]

宣言を開く = [ナビゲート]>[宣言を開く]

型階層を開く = [ナビゲート]>[型階層を開く]

呼び出し階層を開く = [ナビゲート]>[呼び出し階層を開く]

クイック・アウトライン = [ナビゲート]>[クイック・アウトライン]

ソース/ヘッダーの切り替え = [ナビゲート]>[ソース/ヘッダーの切り替え]

表示

サブメニューで選択したビュー ([C/C++ プロジェクト]、[ナビゲーター]、[プロジェクト・エクスプローラー]、[アウトライン]ビュー)をアクティブにして、現在編集中のファイルを示します(Cエディタのみ)。

切り取り = [編集]>[切り取り]

コピー = [編集]>[コピー]

貼り付け = [編集]>[貼り付け]

クイック・フィックス = [編集]>[クイック・フィックス]

ソース

現在カーソルがある行を編集するためのサブメニューが表示されます。

コメント/コメント解除

現在カーソルがある行をコメント行/通常のソース行にします(行先頭に"/"を付加/削除します)(Cエディタのみ)。

ブロックコメントの追加

現在選択されている文字列/行を"/* */"で囲んでコメントにします(Cエディタのみ)。

ブロックコメントの除去

現在選択されている文字列/行の"/* */"を削除して通常のソース行にします(Cエディタのみ)。

右ヘシフト = [編集]>[右ヘシフト]

左ヘシフト = [編集]>[左ヘシフト]

インデントの訂正

編集行のインデントをそろえます。

フォーマット = [編集]>[フォーマット]

Include の追加 = [編集]>[Include の追加]

コンテンツ・アシスト = [編集]>[コンテンツ・アシスト] (Cエディタのみ)

リファクタリング

名前変更..

選択した型、関数やメンバの名称をソース内の他の場所も含めすべて変更します。

宣言

編集領域で選択した文字列(関数名や変数名など)を宣言している箇所を、サブメニューで選択した範囲(ワークスペース、現在のプロジェクト、指定のワーキングセット)で検索します(Cエディタのみ)。

参照

編集領域で選択した文字列(関数名や変数名など)を参照している箇所を、サブメニューで選択した範囲(ワークスペース、現在のプロジェクト、指定のワーキングセット)で検索します(Cエディタのみ)。

検索テキスト

編集領域で選択した文字列を、サブメニューで選択した範囲(ワークスペース、現在のプロジェクト、現在のファイル、指定のワーキングセット)で検索します(Cエディタのみ)。

設定...

エディタの環境設定ダイアログを表示します。

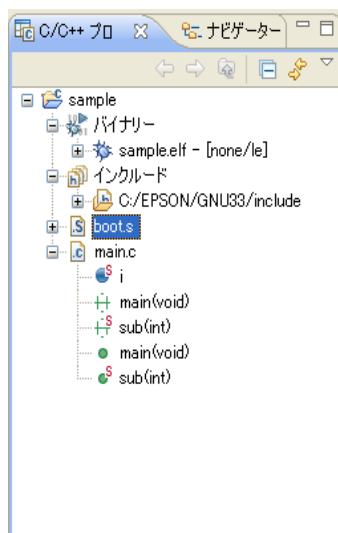
構成のビルド...

[Make ターゲット]で選択するターゲットを定義します。

Make ターゲット

ターゲットを選択してmake.exeを実行します。

5.3.4 [C/C++ プロジェクト]ビュー



ワークスペース内のプロジェクトと、そこに含まれるC/C++およびアセンブラソース、インクルードファイル、生成された実行形式オブジェクトファイルの一覧を表示します(表示させるファイルの種類はビューメニューの[フィルター...]で選択可能です)。C/C++ソース内の関数名やグローバル変数名なども表示可能です。編集などの操作を行う場合に、ここで対象となるプロジェクトあるいはソースを選択します。

ツリー構造で表示されるファイルの一覧は、Windowsエクスプローラ等と同様にナビゲートできます。

+、**-**アイコンのクリックによりフォルダ/ファイル内を表示させたり、親フォルダのみに畳み込むことができます。

特定のフォルダ内のみを表示させるには、フォルダを選択してメニューから[次へジャンプ]を選択します。元に戻すには、下記ツールバーの[上へ]ボタンをクリックします。

ナビゲーション操作は履歴に保存され、メニューやツールバーボタンの[戻る]、[進む]で元の状態に戻したり進めたりできます。

注: アセンブラソースについては、表示が正しく行われな場合があります。

● ツリーリストのアイコン

ツリーリストに表示される主なアイコンの意味は次のとおりです。

	プロジェクト		変数
	バイナリコンテナ		関数
	実行形式ファイル		クラス(class)
	インクルードコンテナ		構造体(struct)
	インクルードフォルダ		メンバ変数
	ヘッダファイル		共用体(union)
	ソースフォルダ		列挙型(enum)
	C/C++ソースファイル		列挙子(enumerator)
	アセンブラソースファイル		関数定義(プロトタイプ宣言)
	テキストファイル等		マクロ定義
	オブジェクトファイル		型定義(typedef)
	インクルード		名前空間(namespace)

● ツールバー

- 戻る**
ビュー内の表示を履歴に沿って一つ前に戻します。
- 進む**
ビュー内の表示を履歴に沿って一つ後に進めます。
- 上へ**
ビュー内の表示を一つ上の階層まで広げます。



すべて縮小表示

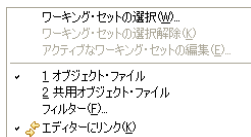
階層を展開している表示(⊖)をすべて最上位の階層で閉じます(⊕)。



エディターにリンク

このボタンを押し込んだ状態にしておくことで、ビュー内の選択内容にエディタの表示が追従します。たとえば、ビュー内でファイルを選択(クリック)すると、その文書がエディタの最前面に表示されます(すでにエディタで開かれている場合のみ)。また、ビュー内に表示されているCソースの関数名や変数名を選択すると、エディタの表示はその関数の先頭、あるいは変数が定義された位置にジャンプします。

ビュー・メニュー



ワーキング・セットの選択...

ワーキングセットの選択、作成、削除などを行います。ワーキングセットは、ビューに表示させるリソースを特定のものに限定するなどの目的に使用します。

ワーキング・セットの選択解除

ワーキングセットを未選択の状態に戻します。

アクティブなワーキング・セットの編集

現在選択されているワーキングセットの内容を編集します。

フィルター ...

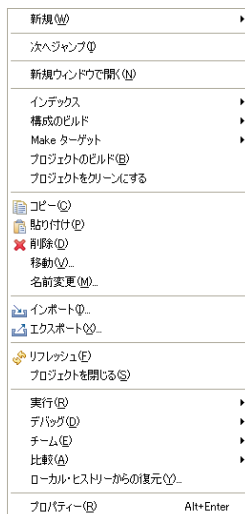
ビューに表示するファイルの種類を指定します。

エディターにリンク

ビュー内の選択状態にエディタの表示を追従させます。

●コンテキストメニュー

ビュー内を右クリックすると、次のコンテキストメニューが表示されます(特に説明のないメニューコマンドについては、メニューバーの説明を参照してください)。



新規 = [ファイル]>[新規]

開く

選択したファイルをエディタで開きます(ファイル選択時)。

アプリケーションから開く

選択したファイルを以下のサブメニューで選択したエディタで開きます(ファイル選択時)。

次へジャンプ = [ナビゲート]>[次へジャンプ](プロジェクト選択時)

新規ウィンドウで開く = [ウィンドウ]>[新規ウィンドウ](プロジェクト選択時)

コピー = [編集]>[コピー]

貼り付け = [編集]>[貼り付け]

削除 = [編集]>[削除]

移動... = [ファイル]>[移動...]

名前変更... = [ファイル]>[名前変更...]

インポート... = [ファイル]>[インポート...]

エクスポート... = [ファイル]>[エクスポート...]

プロジェクトのビルド = [プロジェクト]>[プロジェクトのビルド](プロジェクト選択時)

リフレッシュ = [ファイル]>[リフレッシュ]

ブックマークの追加 = [編集]>[ブックマークの追加](ファイル選択時)

プロジェクトを閉じる = [プロジェクト]>[プロジェクトを閉じる](プロジェクト選択時)

比較

選択した2つまたは3つのファイルの内容を比較します。

ローカル・ヒストリーからの復元

ファイル(削除したものなど)をプロジェクト内に復元します(プロジェクト選択時)。

置換(ファイル選択時)**ローカル・ヒストリー ...**

選択したファイルを以前の保存内容に置き換えます(履歴から選択)。

ローカル・ヒストリーの前回のもとの置換...

選択したファイルを一つ前の保存内容に置き換えます。

オブジェクトファイル変換(ファイル選択時)**S レコードファイル生成(elfファイル選択時)**

選択したelf形式オブジェクトファイルをモトローラS3形式に変換し、HEXファイルを生成します。呼び出されるコマンドは、`objcopy -I elf32-little -O srec --srec-forc-eS3 <ファイル名>.elf <ファイル名>.sa`です。

バイナリファイル生成(elfファイル選択時)

選択したelf形式オブジェクトファイルからデバッグ情報等を削除し、バイナリファイルを生成します。呼び出されるコマンドは、`objcopy -I elf32-little -O binary <ファイル名>.elf <ファイル名>.bin`です。

プロパティー

現在のプロジェクトのプロパティを表示/変更する[プロパティー]ダイアログを表示します。

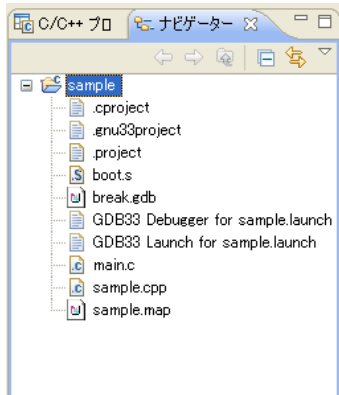
構成のビルド...

[Make ターゲット]で選択するターゲットを定義します。

Make ターゲット

ターゲットを選択してmake.exeを実行します。

5.3.5 [ナビゲーター]ビュー



ワークスペース内のディレクトリおよびファイルの一覧を表示します(表示させるファイルの種類はビューメニューの[フィルター ...]で選択可能です)。編集などの操作を行う場合に、ここで対象となるプロジェクトあるいはソースを選択します。

ツリー構造で表示されるファイルの一覧は、Windowsエクスプローラ等と同様にナビゲートできます。

、アイコンのクリックによりフォルダ/ファイル内を表示させたり、親フォルダのみに畳み込むことができます。

特定のフォルダ内のみを表示させるには、フォルダを選択してメニューから[次へジャンプ]を選択します。元に戻すには、下記ツールバーの[上へ]ボタンをクリックします。

ナビゲーション操作は履歴に保存され、メニューやツールバーボタンの[戻る]、[進む]で元の状態に戻したり進めたりできます。

● ツールバー



戻る

ビュー内の表示を履歴に沿って一つ前に戻します。



進む

ビュー内の表示を履歴に沿って一つ後に進めます。



上へ

ビュー内の表示を一つ上の階層まで広げます。



すべて縮小表示

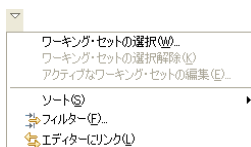
階層を展開している表示()をすべて最上位の階層で閉じます()。



エディターにリンク

このボタンを押した状態にしておくことで、ビュー内の選択内容にエディタの表示が追従します。たとえば、ビュー内でファイルを選択(クリック)すると、その文書がエディタの最前面に表示されます(すでにエディタで開かれている場合のみ)。

ビュー・メニュー



ワーキング・セットの選択...

ワーキングセットの選択、作成、削除などを行います。ワーキングセットは、ビューに表示させるリソースを特定のものに限定するなどの目的に使用します。

ワーキング・セットの選択解除

ワーキングセットを未選択の状態に戻します。

アクティブなワーキング・セットの編集

現在選択されているワーキングセットの内容を編集します。

ソート

名前順

ビューの表示をファイルの種類にかかわらず、アルファベット順に並べます。

タイプ順

ビューの表示をファイルの種類ごとのアルファベット順に並べます。

フィルター ...

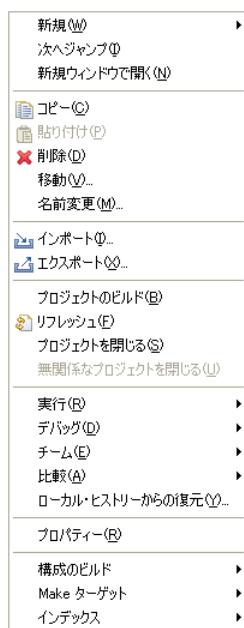
ビューに表示するファイルの種類を指定します。

エディターにリンク

ビュー内の選択状態にエディタの表示を追従させます。

●コンテキストメニュー

ビュー内を右クリックすると、次のコンテキストメニューが表示されます(特に説明のないメニューコマンドについては、メニューバーの説明を参照してください)。



新規 = [ファイル]>[新規]

開く

選択したファイルをエディタで開きます(ファイル選択時)。

アプリケーションから開く

選択したファイルを以下のサブメニューで選択したエディタで開きます(ファイル選択時)。

次へジャンプ = [ナビゲート]>[次へジャンプ](プロジェクト選択時)

新規ウィンドウで開く = [ウィンドウ]>[新規ウィンドウ](プロジェクト選択時)

コピー = [編集]>[コピー]

貼り付け = [編集]>[貼り付け]

削除 = [編集]>[削除]

移動... = [ファイル]>[移動...]

名前変更 = [ファイル]>[名前変更..]

インポート... = [ファイル]>[インポート...]

エクスポート... = [ファイル]>[エクスポート...]

プロジェクトのビルド = [プロジェクト]>[プロジェクトのビルド](プロジェクト選択時)

リフレッシュ = [ファイル]>[リフレッシュ]

プロジェクトを閉じる = [プロジェクト]>[プロジェクトを閉じる](プロジェクト選択時)

無関係なプロジェクトを閉じる

現在選択されているものと無関係の他のプロジェクトを閉じます(プロジェクト選択時)。

比較

選択した2つまたは3つのファイルの内容を比較します。

ローカル・ヒストリーからの復元

ファイル(削除したものなど)をプロジェクト内に復元します(プロジェクト選択時)。

置換(ファイル選択時)

ローカル・ヒストリー ...

選択したファイルを以前の保存内容に置き換えます(履歴から選択)。

ローカル・ヒストリーの前回のもとの置換...

選択したファイルを一つ前の保存内容に置き換えます。

オブジェクトファイル変換(ファイル選択時)

S レコードファイル生成(elfファイル選択時)

選択したelf形式オブジェクトファイルをモトローラS3形式に変換し、HEXファイルを生成します。呼び出されるコマンドは、`objcopy -I elf32-little -O srec --srec-force-s3 <ファイル名>.elf <ファイル名>.sa`です。

バイナリファイル生成(elfファイル選択時)

選択したelf形式オブジェクトファイルからデバッグ情報等を削除し、バイナリファイルを生成します。呼び出されるコマンドは、`objcopy -I elf32-little -O binary <ファイル名>.elf <ファイル名>.bin`です。

プロパティ

現在のプロジェクトのプロパティを表示/変更する[プロパティ]ダイアログを表示します。

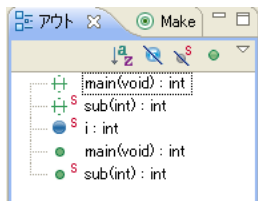
構成のビルド

[Make ターゲット]で選択するターゲットを定義します。

Make ターゲット

ターゲットを選択してmake.exeを実行します。

5.3.6 [アウトライン]ビュー



エディタで表示中のC/C++ソース内に記述されている関数やクラス、グローバル変数などを表示します。それらをクリックすることで、エディタ内の関数/変数の記述位置にジャンプします。アセンブラソースの表示中は何も表示されません。

ツリーリストのアイコンは[C/C++ プロジェクト]ビューと同様です。

● ツールバー



ソート

このボタンを押し込んだ状態にしておくことで、表示内容がアルファベット順に並びます。通常は、エディタ内の出現順に表示されます。



フィールドの非表示

このボタンを押し込んだ状態にしておくことで、フィールドが非表示になります。



static メンバーの非表示

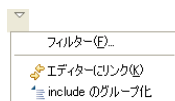
このボタンを押し込んだ状態にしておくことで、静的メンバが非表示になります。



非 public メンバーの非表示

このボタンを押し込んだ状態にしておくことで、public以外のメンバが非表示になります。

ビュー・メニュー



フィルター ...

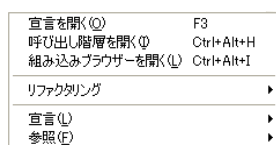
ビューに表示する項目を指定します。

include のグループ化

インクルードファイルをグループ化して表示させるか、個別に表示させるかを選択します。

● コンテキストメニュー

ビュー内を右クリックすると、次のようなコンテキストメニューが表示されます。



リファクタリング

名前変更...

選択したクラス、型、関数やメンバの名称をソース内の他の場所も含めすべて変更します。

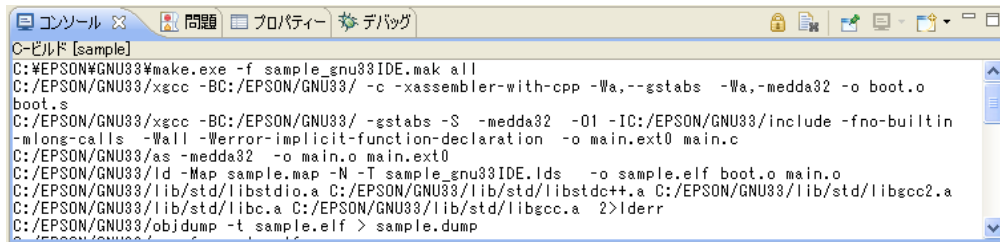
宣言

選択した関数名や変数名などを宣言している箇所を、サブメニューで選択した範囲(ワークスペース、現在のプロジェクト、指定のワーキングセット)で検索します。

参照

編集領域で選択した関数名や変数名などを参照している箇所を、サブメニューで選択した範囲(ワークスペース、現在のプロジェクト、指定のワーキングセット)で検索します。

5.3.7 [コンソール]ビュー



実行されたコマンドラインやGNU33ツールが出力するメッセージを表示します。

● ツールバー



スクロール・ロック

このボタンを押し込んだ状態にしておくことで、自動スクロールを禁止します。



コンソールのクリア

ビューの表示内容をクリアします。



コンソールのピン留め

このボタンを押し込んだ状態にしておくことで、[コンソール]ビューがメッセージを出力中でも同ペインの他のビューをアクティブにすることができます。ビルドに時間がかかる場合などに有効です。



選択されたコンソールの表示

ビルド時のコンソールとデバッガ起動時のコンソールなど、複数のコンソールを開いている場合に、[コンソール]ビューに表示するコンソールを選択します。



コンソールを開く

新しいコンソールを開きます。



終了

デバッガ起動時などのコンソールに表示されるボタンです。このボタンをクリックすると、コンソールに対応するツール(デバッガ)が処理を中止して閉じます。コンソールは閉じません。ツール側の操作で終了した場合もコンソールは閉じません。



起動の除去

デバッガ起動時などのコンソールに表示されるボタンです。現在表示されているコンソールを閉じます。

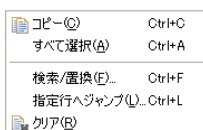


終了したすべての起動を除去

デバッガ起動時などのコンソールに表示されるボタンです。終了しているツールのコンソールをすべて閉じます。

● コンテキストメニュー

ビュー内を右クリックすると、次のようなコンテキストメニューが表示されます。



コピー = [編集]>[コピー]

すべて選択 = [編集]>[すべて選択]

検索/置換... = [編集]>[検索/置換...]

指定行へジャンプ...

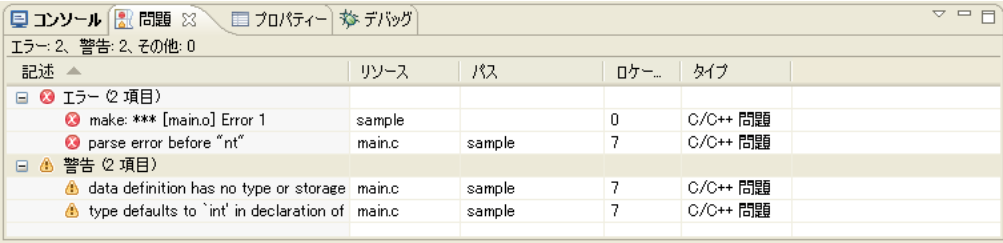
ビュー内の指定の行にジャンプします。

クリア

ビューの表示内容をクリアします。

(特に説明のないメニューコマンドについては、メニューバーの説明を参照してください。)

5.3.8 [問題]ビュー



記述	リソース	パス	ローケ...	タイプ
エラー 2 項目)				
make: *** [main.o] Error 1	sample		0	C/C++ 問題
parse error before "nt"	main.c	sample	7	C/C++ 問題
警告 2 項目)				
data definition has no type or storage	main.c	sample	7	C/C++ 問題
type defaults to 'int' in declaration of	main.c	sample	7	C/C++ 問題

ビルド時に発生したエラーを表示します。ソースファイル内のエラーはエラーメッセージをクリックすることで、エディタ上のエラー発生箇所にジャンプします。

●ビューメニュー

表示(S)	▶
グループ(G)	▶
ソート順	▶
新規問題ビュー	
内容の構成...	
列...	
設定	

表示

適用するフィルタを選択します。

グループ

エラーをグループ化する対象を選択します。

ソート順

一覧のどの項目を優先して並べ替えるかを選択します。また、[昇順]をチェックすると、それぞれの項目を昇順で並べ替え、チェックを外すと降順で並べ替えます。

新規問題ビュー

新たなエラービューを作成します。

内容の構成...

フィルタ設定を作成、編集します。
ここで条件を設定し、表示させるエラーの対象や数を制限します。フィルタ設定については“5.10.7 フィルター”を参照してください。

列...

表示する項目(列)の順番を設定します。

設定...

表示するエラーの最大数、項目(列)の表示/非表示を設定します。

●コンテキストメニュー

ビュー内を右クリックすると、次のようなコンテキストメニューが表示されます。

ジャンプ	
コピー	Ctrl+C
削除	Delete
すべて選択	Ctrl+A
表示	Alt+Shift+W ▶
クイック・フィックス	Ctrl+I
プロパティ	Alt+Enter
外部エディタで開く(Q)	

ジャンプ

エディタ内のエラーが発生した行にジャンプします。

表示

選択したエラーが発生しているリソースをハイライト表示するビュー(エディタ以外で可能なビューがある場合のみ)を選択します。

コピー = [編集]>[コピー]

すべて選択 = [編集]>[すべて選択]

プロパティ

ビュー内で選択されているエラーの情報を表示します。

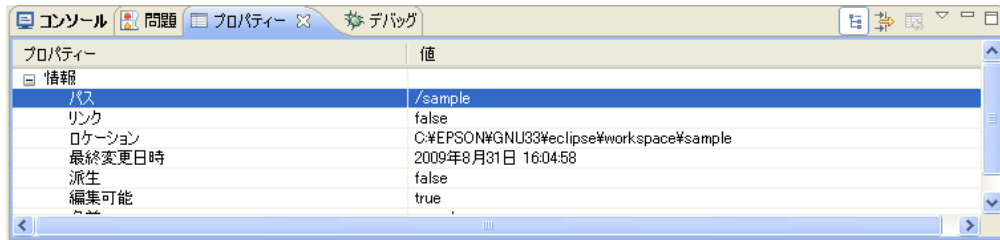
外部エディタで開く

“5.5.10 外部エディタを行番号指定で起動するには”で外部エディタの設定をしている場合は、コンテキストメニューからエラー発生行にジャンプすることができます。

詳細は“5.5.10 外部エディタを行番号指定で起動するには”を参照してください。

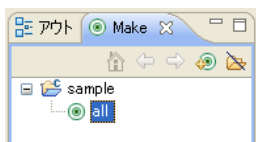
(特に説明のないメニューコマンドについては、メニューバーの説明を参照してください。)

5.3.9 [プロパティ]ビュー



[C/C++ プロジェクト]ビュー、[ナビゲーター]ビューまたは[アウトライン]ビューで選択されたリソースやメンバなどの情報を表示します。

5.3.10 [Make ターゲット]ビュー



ユーザが編集したmakeファイルを使用する場合に、ここにターゲットを定義して実行します。

● ツールバー



ホーム

ツリーリストの最上位の階層に戻ります。



戻る

ツリーリストの一つ上の階層に戻ります。



次へジャンプ

ツリーリストの一つ下の階層に進みます。



Make ターゲットのビルド

選択されたターゲットのメイクを実行します。



空のフォルダを隠す

フォルダを隠し、登録したターゲットのみを表示します。

● コンテキストメニュー

ビュー内を右クリックすると、次のようなコンテキストメニューが表示されます。



Make ターゲットのビルド

選択したターゲットのメイクを実行します。

Make ターゲットの追加

メイクターゲットを定義します。

Make ターゲットの削除

選択したターゲットを削除します。

Make ターゲットの編集

選択したターゲットを編集します。

ホームへ戻る

ツリーリストの最上位の階層に戻ります。

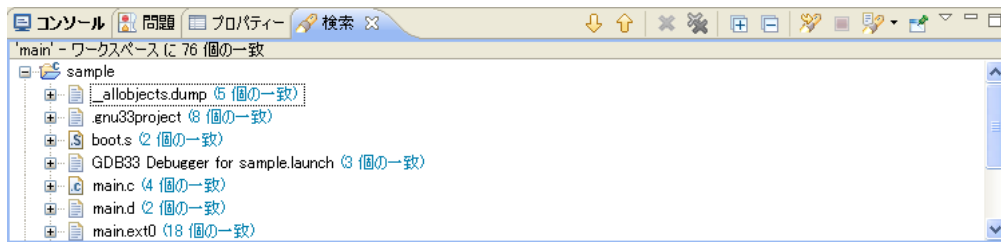
戻る

ツリーリストの一つ上の階層に戻ります。

次へジャンプ

ツリーリストの一つ下の階層に進みます。

5.3.11 [検索]ビュー



[検索]ダイアログを使用した検索結果を表示します。このビューは初期状態では表示されません。検索を実行すると開きます。

● ツールバー



次の一致を表示

検索結果の一つ後の候補にジャンプします。



直前の一致を表示

検索結果の一つ前の候補にジャンプします。



選択された一致を除去

選択した検索結果をビューから削除します。



すべての一致を除去

ビュー内の検索結果をすべて削除します。



すべて展開

ビュー内の階層表示をすべて展開します。



すべて縮小表示

展開されている階層表示を畳みます。



現在の検索をもう1度実行する

直前の検索を再度実行します。



現行検索のキャンセル

進行中の検索動作をキャンセルします。



直前の検索を表示

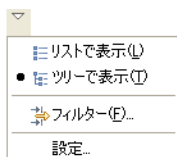
以前行った検索結果を選択して表示します。



検索ビューをピン留め

このボタンを押し込んだ状態にしておくことで、[検索]ビューが検索結果を出力中でも同ペインの他のビューをアクティブにすることができます。検索に時間がかかる場合などに有効です。

ビュー・メニュー



リストで表示

検索結果を階層化せずに表示します。

ツリーで表示

検索結果を階層表示にします。

設定...

検索に関する環境設定ダイアログを表示します。

●コンテキストメニュー

ビュー内を右クリックすると、次のようなコンテキストメニューが表示されます。

開く(E)	
アプリケーションから開く(H)	▶
表示	Alt+Shift+W ▶
↓ 次の一致	Ctrl+,
↑ 直前の一致	Ctrl+,
🔍 すべて展開	
📄 コピー	Ctrl+C
✖ 選択された一致を除去	Delete
✖ すべての一致を除去	
🔄 選択対象を置換(L)...	
🔄 すべて置換(E)...	
🔍 再検索	
実行(R)	▶
デバッグ(D)	▶
チーム(E)	▶
比較(A)	▶
置換(L)	▶
オブジェクトファイル変換	▶
プロパティ(R)	
構成のビルド	▶
Make ターゲット	▶

開く

選択したファイルをエディタで開きます。(ファイル検索時)

アプリケーションから開く

選択したファイルを以下のサブメニューで選択したエディタで開きます。(ファイル検索時)

C/C++ エディタ (アセンブリ・エディター)

C/C++ エディタ (C/C++ソース選択時) またはアセンブリエディタ (アセンブラソース選択時)

テキスト・エディター

テキストエディタ

システム・エディター

Windows プログラム (Notepad など)

In-Place エディター

C/C++ エディタ (C/C++ソース選択時) またはアセンブリエディタ (アセンブラソース選択時)

デフォルト・エディター

C/C++ エディタ (C/C++ソース選択時) またはアセンブリエディタ (アセンブラソース選択時)

表示

選択されている候補をサブメニューから選択したビューでハイライト表示します。

次の一致

検索結果の一つ後の候補にジャンプします。

直前の一致

検索結果の一つ前の候補にジャンプします。

選択された一致を除去

選択した検索結果をビューから削除します。

すべての一致を除去

ビュー内の検索結果をすべて削除します。

選択対象を置換...

現在選択されている候補のみを別の語句と置き換えます。(ファイル検索時)

すべて置換...

ファイル内の全候補を別の語句と置き換えます。(ファイル検索時)

再検索

前回の検索を再度行います。

比較

相互

選択した2つまたは3つのファイルの内容を比較します。

ローカル・ヒストリー ...

選択したファイルの内容を以前保存したときの内容と比較します。(ファイル選択時)

ローカル・ヒストリーからの復元...

ファイル(削除したものなど)をプロジェクト内に復元します。(プロジェクト選択時)

置換 (ファイル選択時)**ローカル・ヒストリーの前回のものと置換**

選択したファイルを一つ前の保存内容に置き換えます。

ローカル・ヒストリー ...

選択したファイルを以前の保存内容に置き換えます (履歴から選択)。

オブジェクトファイル変換 (ファイル選択時)**S レコードファイル生成 (elfファイル選択時)**

選択したelf形式オブジェクトファイルをモトローラS3形式に変換し、HEXファイルを生成します。呼び出されるコマンドは、`objcopy -I elf32-little -O srec --srec-forceS3<ファイル名> .elf <ファイル名> .sa`です。

バイナリファイル生成 (elfファイル選択時)

選択したelf形式オブジェクトファイルからデバッグ情報等を削除し、バイナリファイルを生成します。呼び出されるコマンドは、`objcopy -I elf32-little -O binary<ファイル名> .elf <ファイル名> .bin`です。

プロパティ

ビュー内で選択されている検索結果の情報を表示します。

構成のビルド

[Make ターゲット]で選択するターゲットを定義します。

Make ターゲット

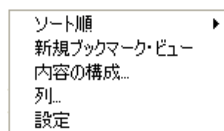
ターゲットを選択して**make.exe**を実行します。

5.3.12 [ブックマーク]ビュー



エディタで登録したブックマークの表示、ブックマークへのジャンプ、ブックマークの削除などを行います。このビューは初期状態では表示されません。[ウィンドウ]メニューの[ビューの表示]から選択して開きます。

●ビューメニュー



ソート順

一覧のどの項目を優先して並べ替えるかを選択します。また、[昇順]をチェックすると、それぞれの項目を昇順で並べ替え、チェックを外すと降順で並べ替えます。

新規ブックマーク・ビュー

新たなブックマークビューを作成します。

内容の構成...

フィルタ設定を作成、編集します。

ここで条件を設定し、表示させるブックマークの対象や数を制限します。

フィルタ設定については“5.10.7 フィルター”を参照してください。

列...

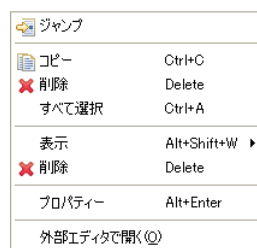
表示する項目(列)の順番を設定します。

設定...

表示するブックマークの最大数、項目(列)の表示/非表示を設定します。

●コンテキストメニュー

ビュー内を右クリックすると、次のようなコンテキストメニューが表示されます。



ジャンプ

エディタ内のブックマークの位置にジャンプします。

表示

選択したブックマークが定義されているリソースをサブメニューから選択したビューでハイライト表示します。

コピー = [編集]>[コピー]

削除 = [編集]>[削除]

すべて選択 = [編集]>[すべて選択]

プロパティ

ビュー内で選択されているブックマークの情報を表示します。

(特に説明のないメニューコマンドについては、メニューバーの説明を参照してください。)

5.3.13 [タスク]ビュー

項目	記述	リソース	パス	ロケー...	タイプ
☒	!	main.c	sample	行 16	タスク
☐		main.c	sample	行 22	タスク
☐	↓	main.c	sample	行 26	タスク

エディタで登録したタスクの表示、タスクへのジャンプ、タスクの削除などを行います。ここで言うタスクとは、“To-Do”項目のことで、行の先頭の□は完了時にチェックするためのアイコンです。次の列には優先順位を示すアイコン(高 = !、標準 = 空白、低 = ↓)が表示されます。このビューは初期状態では表示されません。[ウィンドウ]メニューの[ビューの表示]から選択して開きます。

●ビューメニュー

表示(S)	▶
グループ(G)	▶
ソート順	▶
新規タスク・ビュー	
内容の構成...	
列...	
設定	

表示

適用するフィルタを選択します。

グループ

タスクをグループ化する対象を選択します。

ソート順

一覧のどの項目を優先して並べ替えるかを選択します。また、[昇順]をチェックすると、それぞれの項目を昇順で並べ替え、チェックを外すと降順で並べ替えます。

新規タスク・ビュー

新たなタスクビューを作成します。

内容の構成...

フィルタ設定を作成、編集します。

ここで条件を設定し、表示させるタスクの対象や数を制限します。フィルタ設定については“5.10.7 フィルター”を参照してください。

列...

表示する項目(列)の順番を設定します。

設定...

表示するタスクの最大数、項目(列)の表示/非表示を設定します。

●コンテキストメニュー

ビュー内を右クリックすると、次のようなコンテキストメニューが表示されます。

ジャンプ	
コピー	Ctrl+C
削除	Delete
すべて選択	Ctrl+A
表示	
タスクの追加...	Alt+Shift+W ▶
削除	Delete
完了タスクの削除	
完了のマークを付ける	
プロパティ	Alt+Enter
外部エディタで開く(Q)	

タスクの追加... = [編集]>[タスクの追加...]

ジャンプ

エディタ内のタスク設定位置にジャンプします。

表示

選択したタスクが定義されているリソースをサブメニューから選択したビューでハイライト表示します。

コピー = [編集]>[コピー]

削除 = [編集]>[削除]

すべて選択 = [編集]>[すべて選択]

完了のマークを付ける

選択したタスクに完了マークを付けます。

完了タスクの削除

完了しているタスクをすべて削除します。

プロパティ

ビュー内で選択されているタスクの情報を表示します。

(特に説明のないメニューコマンドについては、メニューバーの説明を参照してください。)

5.3.14 ビューの操作

ここでは、すべてのビューに関わるオープン/クローズやレイアウトを変更する操作について説明します。

●ビューのオープン/クローズ

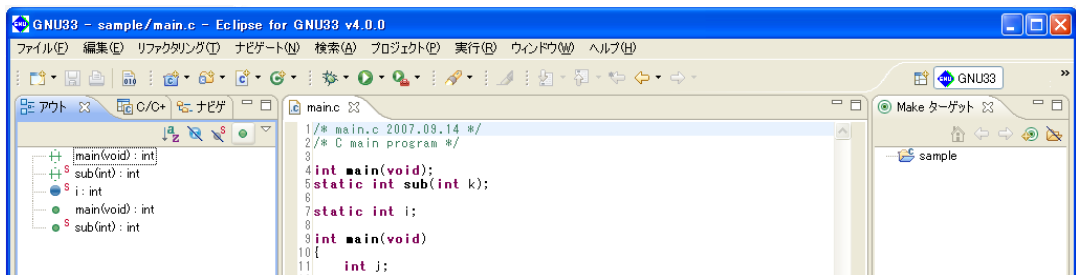
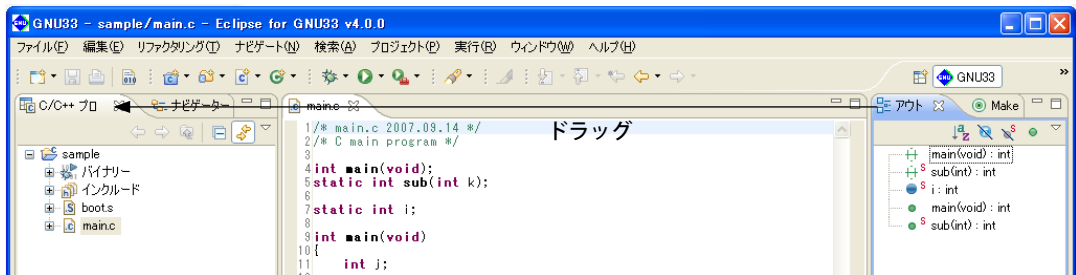
表示されているビューは、タブの  ボタンをクリックすると閉じます。一つのペイン上のすべてのビューを閉じると、そのペインも表示されなくなります。

閉じたビューは[ウィンドウ]メニューの[ビューの表示]から選択して開きます。このとき、選択したビューがどのペインに表示されるかは、後述のパーспекティブの設定により決まります。

一つのペイン内に複数のビューが重なっている場合、表示させるビューは上部のタブで選択します。

●ビューレイアウトの変更

タブをドラッグして、ビューの表示位置を変更することができます。タブを移動可能な位置にドラッグすると、移動先を示す四角の枠が表示されます。たとえば、他のペイン内にドラッグした際に、そのペインのサイズの枠とフォルダアイコンが表示されれば、そのペインへ移動します。タブの位置でタブサイズの枠が表示された場合も、そのペインへの移動となりますが、タブの重なりの中でどの位置に挿入するかを選択できます。ドラッグ時に矢印アイコンとともに、現在のペインとは異なるサイズの枠が表示された場合は、ペインが分割され、新たなペインの中にビューが移動します。



各ペインのサイズも境界の枠をドラッグして変更可能です。

●ビューの最大化

ビューのタブをダブルクリックすると、そのビューを含むペインがIDEウィンドウのサイズまで拡大され、その他のビューは下に隠れます。最大化されたビューのタブをダブルクリックすると、ビューが元のサイズに戻ります。また、各ペインの右上には最大化ボタンがあり、このボタンのクリックでも同様の操作が行えます。最小化ボタンは最大化されたビューを元のサイズに戻します。通常の表示で最小化ボタンをクリックすると、ビューはタブのみの表示となります。



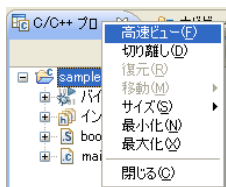
最大化ボタン



最小化ボタン

●高速ビュー

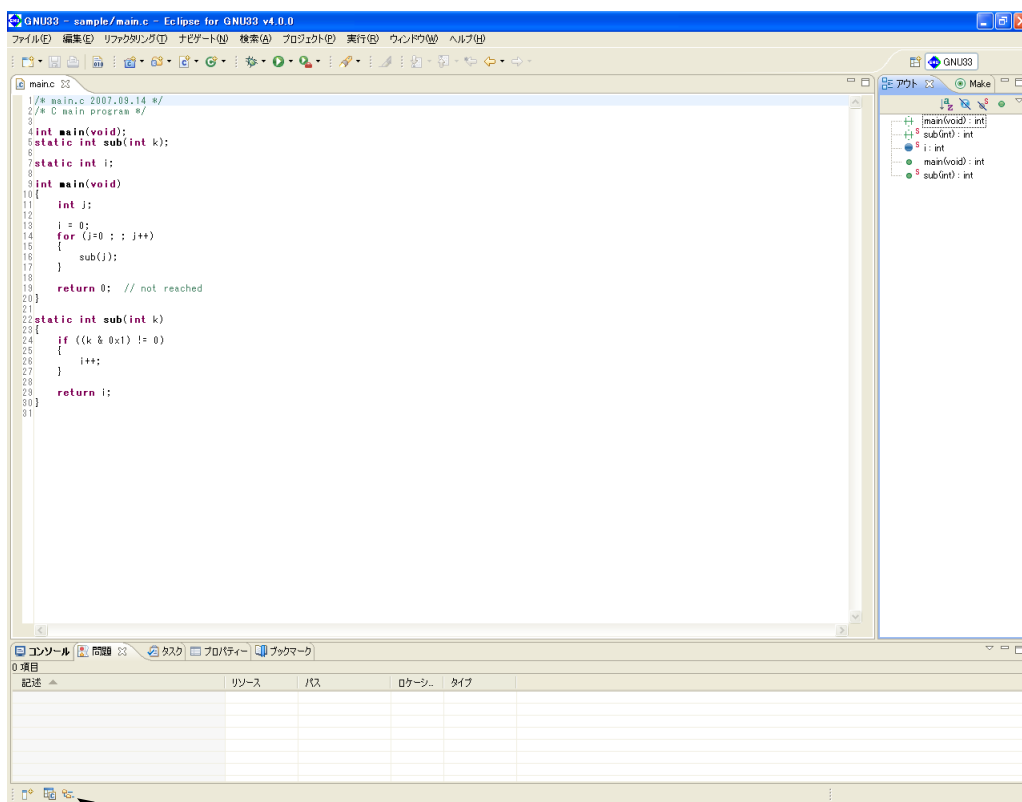
たとえばソースの編集時など、エディタ領域を広げたいことがあります。エディタ領域を最大化する方法もありますが、これとは別に高速ビューを使用する方法があります。これは、とりあえず不要なビューをアイコンにしてウィンドウ左下の高速ビューエリアに置き、そのアイコンをクリックすることでビューが拡大表示される機能です。この方法では、他のビューすべてを隠してしまうことはありません。



ビューを高速ビューにするには、ビューのタブを右クリックして表示されるコンテキストメニューから[高速ビュー]を選択します。タブを高速ビューエリアまでドラッグ&ドロップして高速ビュー化することも可能です。

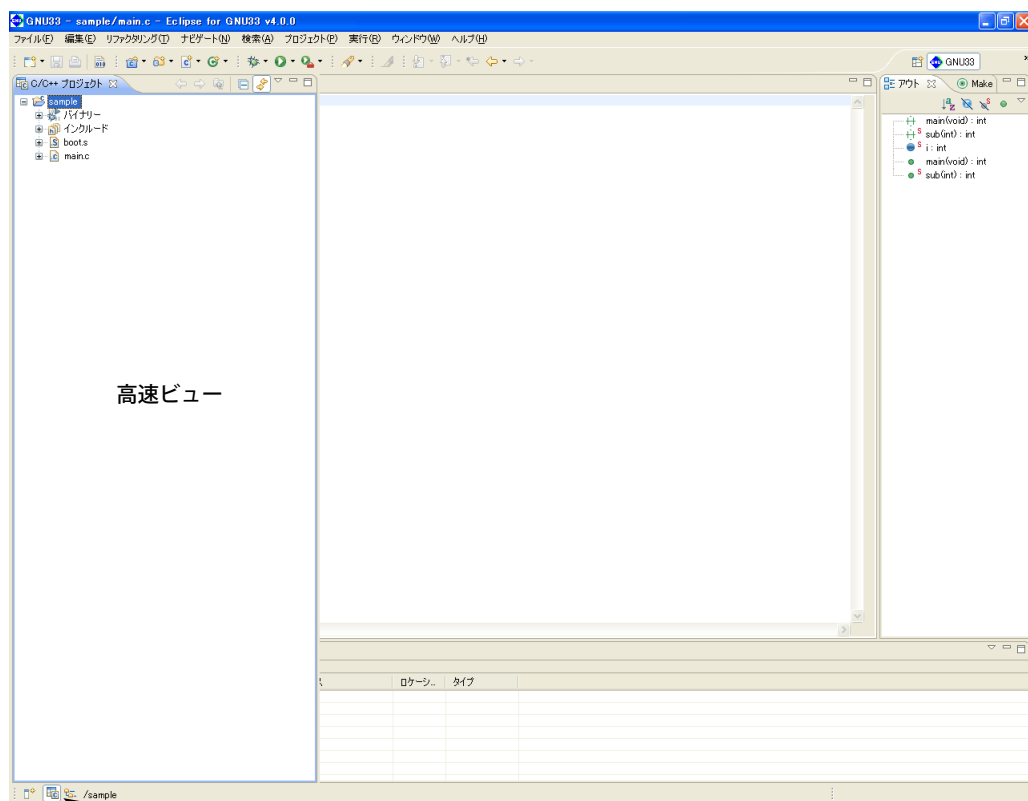
たとえば、[C/C++ プロジェクト]と[ナビゲーター]タブのコンテキストメニューから[高速ビュー]を選択してください。それぞれのビューが高速ビュー化され、ビューのアイコンがウィンドウ左下の高速ビューエリアに現れます。左ペイン内のビューが閉じたため、その分エディタや下のビューエ

リアが拡大されます。

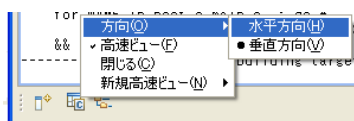


高速ビューエリア

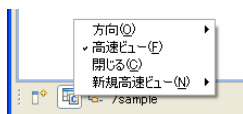
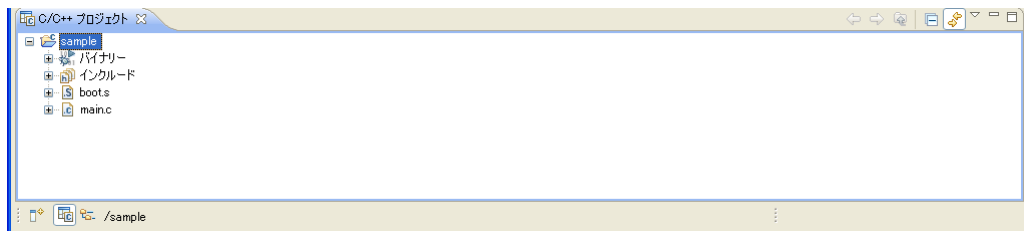
高速ビューを開くには、高速ビューエリアのアイコンをクリックします。閉じるには、それ以外のビューをクリックするか、高速ビューエリアのアイコンまたは開いた高速ビューの最小化ボタンをクリックします。



アイコンのクリックで高速ビューを開開



高速ビューを開くと、デフォルト設定では左端に垂直に表示されます。高速ビューアイコンのコンテキストメニューから[方向]>[水平方向]を選択すると、高速ビューがウィンドウの下側に水平に開くようになります。



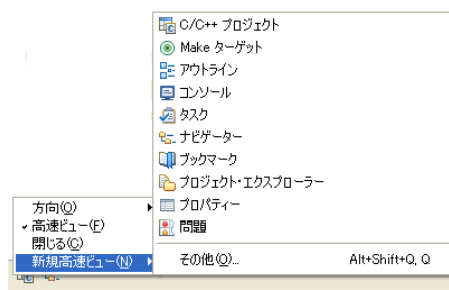
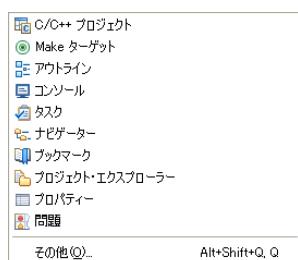
高速ビューを通常のビューに戻すには、高速ビューアイコンのコンテキストメニューから[高速ビュー]を選択します。

このコンテキストメニューから[閉じる]を選択するとそのビューが閉じ、アイコンも消えます。

上記の他に、[ビューを高速ビューとして表示する]ボタンで表示されるメニューからビューを選択し、高速ビューとして開くことができます。コンテキストメニューの[新規高速ビュー]からでも同じ操作は行えます。



[ビューを高速ビューとして表示する]ボタン



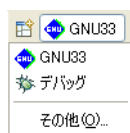
●ビューレイアウトをデフォルト設定に戻すには

[ウィンドウ]メニューから[パースペクティブのリセット]を選択します。確認のダイアログが表示されますので、[OK]をクリックしてください。IDEのデフォルト設定のレイアウト(初めてIDEを起動したときの状態)に戻ります。

ビューレイアウトはIDEを終了する際に保存され、次回は終了時のレイアウトで起動します。再起動してもデフォルト設定には戻りません。

5.3.15 パースペクティブ

表示させるビューの構成やエディタ領域を含めたレイアウト、メニューやツールバーの構成などの定義をパースペクティブと呼んでいます。IDEが採用しているEclipseは、開発環境に合わせてパースペクティブを切り換えられるようになっています。IDEでは前節に示したビューの構成とレイアウトで"GNU33"という名称のパースペクティブが定義されています。



パースペクティブショートカットアイコンの横に表示されている"GNU33"は、現在GNU33パースペクティブが選択されていることを示しています。IDEはパースペクティブの変更や切り替えにも対応していますが、初期設定の"GNU33"以外は使用しないでください。

5.4 プロジェクト

5.4.1 プロジェクトとは

IDEでは個々のアプリケーション開発をプロジェクト名で管理します。具体的には、1つのアプリケーション開発を始めるに当たって指定したプロジェクト名を持つディレクトリを作成し、そこでソースファイルなどのリソースやコンパイラなどのツールが生成するファイルを管理します。

プロジェクトディレクトリにはプロジェクトの管理用ファイル(.cproject、.gnu33project、.project)も生成され、IDEにより随時更新されます。

注: プロジェクトディレクトリ内の管理用ファイルは、絶対にIDE上の操作以外で編集、移動、削除しないでください。プロジェクトが再開できなくなることがあります。

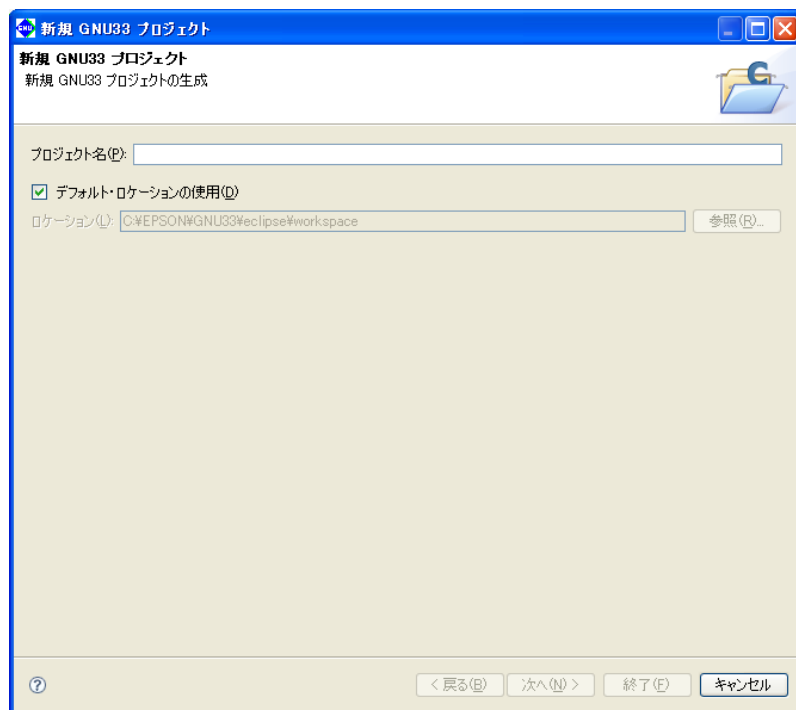
5.4.2 新規プロジェクトの作成

IDEによるアプリケーション開発は、新規プロジェクトの作成から始めます。その手順は次のとおりです。

(1) 以下のいずれかの方法で、[新規 GNU33 プロジェクト]ウィザードを開始させます。

- [ファイル]メニューから[新規]>[新規 GNU33 プロジェクト]を選択します。
- ツールバーの[新規]ショートカットから[新規 GNU33 プロジェクト]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[新規 GNU33 プロジェクト]を選択します。

ウィザードが起動し、次のダイアログが表示されます。



(2) [プロジェクト名:]にプロジェクト名を入力します。

注: ・ プロジェクト名は100文字以内にしてください。

- プロジェクト名には、半角英数字とアンダーバー以外の文字は使用できません。

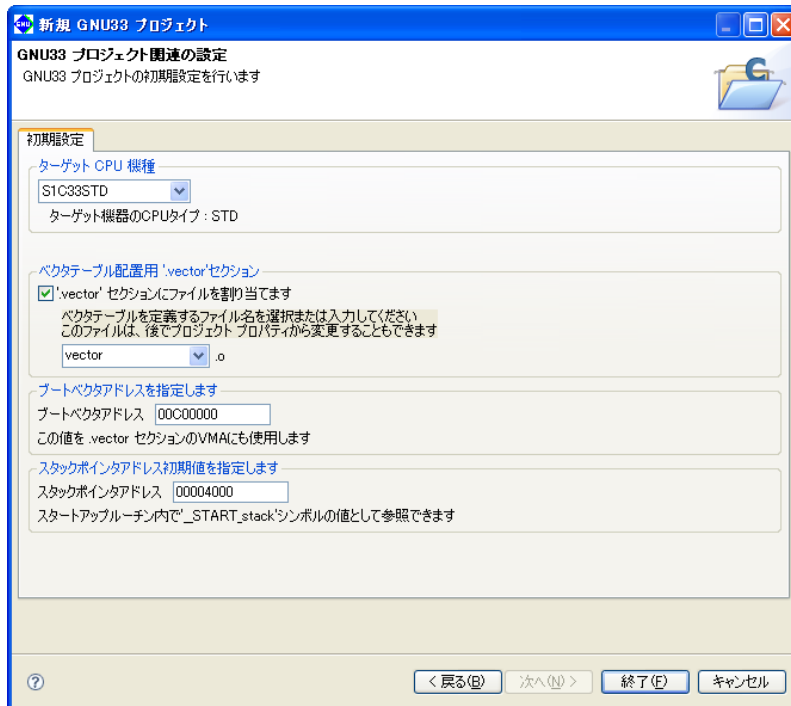
- (3) プロジェクトディレクトリを作成する場所を指定します。(特に必要な場合のみ)

デフォルト設定では、[デフォルト・ロケーションの使用]チェックボックスが選択されており、**IDE** 起動時に指定したワークスペースディレクトリ内にプロジェクトフォルダが生成されます。通常はこのまま次に進んでください。

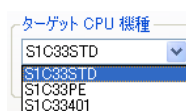
ワークスペース以外の場所にプロジェクトディレクトリを作成したい場合は、[デフォルト・ロケーションの使用]のチェックを外し、[ロケーション:]にパスを入力するか、[参照...]ボタンをクリックして既存のディレクトリを選択します。

注: パスは最大200文字までに制限されています。

- (4) [次へ>]ボタンをクリックします。
ターゲットCPU選択画面に移行します。



- (5) [ターゲット CPU 機種]コンボボックスからターゲットCPUの種類を選択します。



S1C33STD: C33 STDコアを内蔵したデバイスをターゲット機種とする場合

S1C33PE: C33 PEコアを内蔵したデバイスをターゲット機種とする場合

S1C33401: S1C33401をターゲット機種とする場合

ターゲットCPUは、後からの変更も可能です("5.7.1 CPUタイプの設定"参照)。

- (6) .vectorセクション(ベクタテーブル用セクション)に配置するオブジェクトをコンボボックスで選択(vector.oとboot.oが選択可能)、もしくは入力します。



.vectorセクションに配置しない場合は、['.vector' セクションにファイルを割り当てます]のチェックを外してください。

.vectorセクションの設定は、後からの変更も可能です("5.7.7 リンカスクリプトの編集"参照)。

(7) ブートベクタアドレスを指定します。

デフォルト設定値は、ターゲットCPUに"S1C33STD"または"S1C33PE"を選択した場合は"00C00000"、"S1C33401"を選択した場合は"20000000"となります。ここに設定した値は、**IDE**が自動生成するデバッグ起動用コマンドファイル内のTTBR設定コマンドのパラメータ、およびリンクスクリプト内の.vectorセクションのVMAとして使用されます。

(8) スタックポインタアドレスを指定します。

デフォルト設定値は、ターゲットCPUに"S1C33STD"または"S1C33PE"を選択した場合は"00004000"、"S1C33401"を選択した場合は"00008000"となります。ここに設定した値は、**IDE**が自動生成するリンクスクリプトファイルの__START_stackシンボルの値となり、スタック領域の開始アドレスとしてシンボルを使用することができます。

例) ブートルーチン内で以下のように記述できます。

boot:

```
xld.w %r15, __START_stack
ld.w %sp, %r15
```

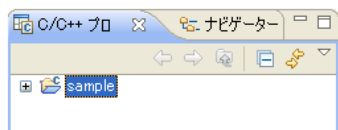
(9) [終了]ボタンをクリックします。

[新規 GNU33 プロジェクト]ウィザードが終了し、指定した名称のプロジェクトが作成されます。

プロジェクトを新規作成すると、現在のワークスペースまたは(3)で指定したディレクトリ内にプロジェクトと同じ名称のディレクトリが作成されます。プロジェクト名のディレクトリがすでに存在する場合は、そのディレクトリをそのままプロジェクトディレクトリとして使用します。

[C/C++ プロジェクト]または[ナビゲーター]ビュー上では、プロジェクトが次のようなフォルダアイコンを伴って表示されます。

例: "sample"の名称でプロジェクトを作成

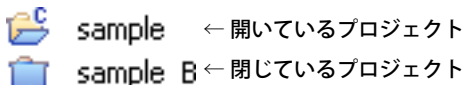


ワークスペース内に複数のプロジェクトを作成した場合は、そのすべてが[C/C++ プロジェクト]または[ナビゲーター]ビュー上に現れます。

5.4.3 プロジェクトのオープンとクローズ

プロジェクトを作成すると、そのプロジェクトは開いた状態になります。開いているプロジェクトはクローズの操作を行うまで閉じられることはありません。**IDE**を終了してその後再起動した場合も、終了時に開いていたプロジェクトは開いた状態で作業を継続することができます。

例: 開いているプロジェクトと閉じているプロジェクトのアイコン



プロジェクト内のソースファイルの編集やビルドなどの操作は、プロジェクトが開いており、かつ[C/C++ プロジェクト]または[ナビゲーター]ビューでプロジェクトフォルダまたはそこに含まれるファイルが選択されている必要があります。

●プロジェクトのクローズ

ワークスペースに複数のプロジェクトが存在する場合、作業中以外のプロジェクトを閉じておくことができます。その手順は次のとおりです。

- (1) [C/C++ プロジェクト]または[ナビゲーター]ビューで閉じるプロジェクトをクリックして選択します。
- (2) 以下のいずれかの方法で、プロジェクトをクローズします。
 - [プロジェクト]メニューから[プロジェクトを閉じる]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[プロジェクトを閉じる]を選択します。

以上の操作でプロジェクトが閉じます。このとき、そのプロジェクト内のソースファイルなどがエディタで開かれていた場合は、それらのファイルも同時に閉じます。エディタで編集内容が保存されていない場合は、[リソースの保管]ダイアログ(5.10.2節参照)が現れ、ファイル個々に保存するかどうかを選択することができます。

●プロジェクトのオープン

ワークスペース内に存在し、[C/C++ プロジェクト]または[ナビゲーター]ビュー上で閉じているプロジェクトは、次の方法で開くことができます。

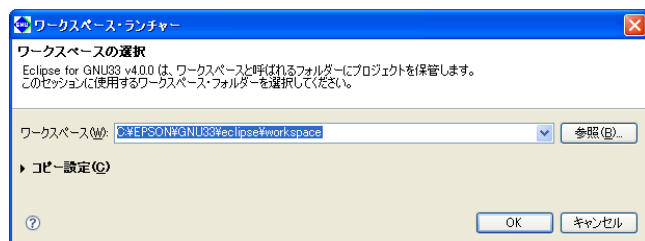
- (1) [C/C++ プロジェクト]または[ナビゲーター]ビューで開くプロジェクトをクリックして選択します。
- (2) 以下のいずれかの方法で、プロジェクトをオープンします。
 - [プロジェクト]メニューから[プロジェクトを開く]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[プロジェクトを開く]を選択します。

この操作は[C/C++ プロジェクト]または[ナビゲーター]ビューに表示され、閉じているプロジェクトにのみ有効です。現在のワークスペース以外のディレクトリにあるプロジェクトを開くには、ワークスペースをプロジェクトの置かれたディレクトリに切り換えます(5.4.4節参照)。または、現在のワークスペースに既存のプロジェクトをインポートしてください(5.4.5節参照)。

5.4.4 ワークスペースの切り換え

[C/C++ プロジェクト]または[ナビゲーター]ビューには、現在のワークスペースに存在するプロジェクトのみが表示されます。他のディレクトリにあるプロジェクトの作業を行うには、ワークスペースをそのディレクトリに切り換える必要があります。また、新たなディレクトリを作成してワークスペースにすることもできます。その手順は次のとおりです。

- (1) エディタで編集中の文書がある場合は、すべて保存してください。
- (2) [ファイル]メニューから[ワークスペースの切り替え...]を選択します。
[ワークスペース・ランチャー]ダイアログが表示されます。



- (3) [ワークスペース:]コンボボックスにパスを入力するか、[参照...]ボタンをクリックして表示されるディレクトリ選択ダイアログから既存のディレクトリを選択します。
以前使用したワークスペースであれば、[ワークスペース:]コンボボックスの▼ボタンで表示される一覧から選択することもできます。

- (4) [OK]ボタンをクリックします。

以上の操作で、一旦**IDE**ウィンドウが閉じ、指定のディレクトリをワークスペースに設定した後に新たなウィンドウが開きます。このとき、ソースファイルなどがエディタで開かれていた場合は、それらのファイルも同時に閉じます。エディタで編集内容が保存されていない場合は、[リソースの保管]ダイアログ(5.10.2節参照)が現れ、ファイル個々に保存するか否かを選択することができます。

切り換え後のワークスペースに既存のプロジェクトが存在する場合は、前回そのプロジェクトの作業を終了したときの状態でウィンドウが開きます。

- (3) で存在しないディレクトリを指定した場合は、新たに作成されます。

5.4.5 既存プロジェクトのインポート

ここでは、既存のプロジェクトを現在のワークスペースに取り込む方法を説明します。

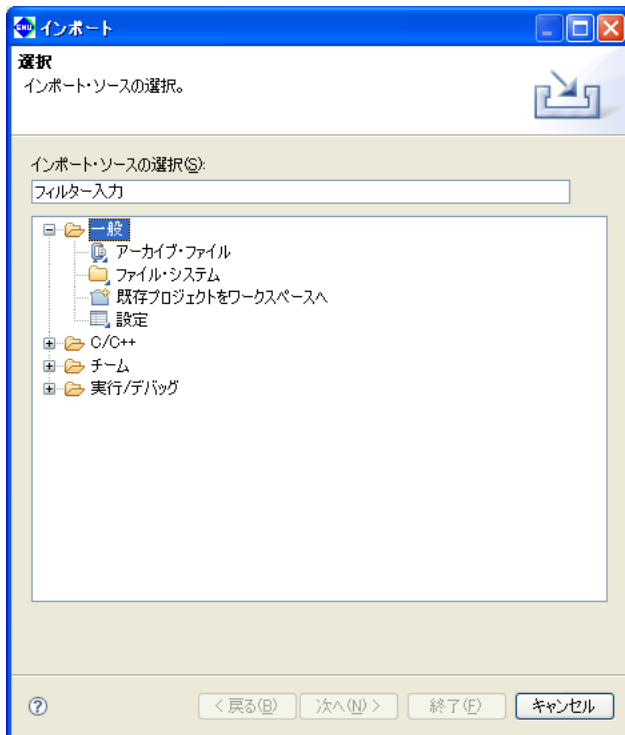
以下の手順でインポートすることで、旧バージョン(V3.1以降)の**IDE**で作成したプロジェクトで引き続き開発することができます。

インポートの手順は次のとおりです。

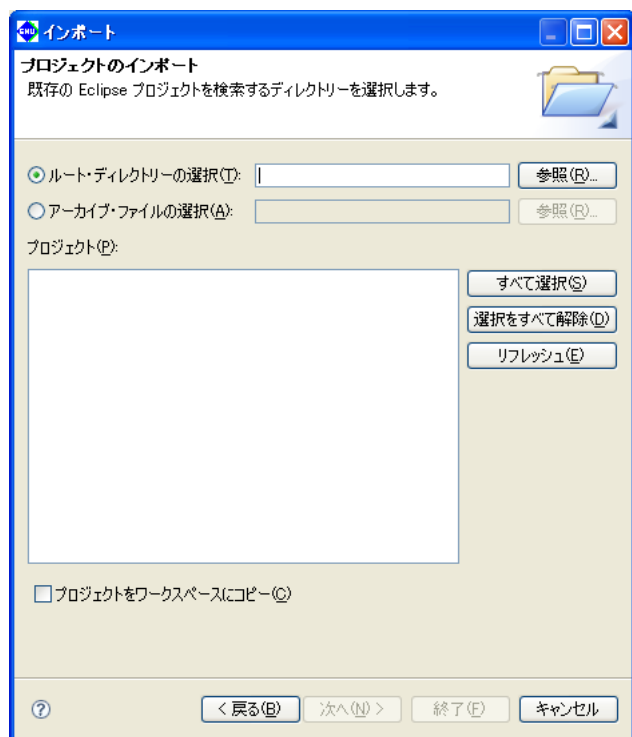
(1) 以下のいずれかの操作を行います。

- [ファイル]メニューから[インポート...]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[インポート...]を選択します。

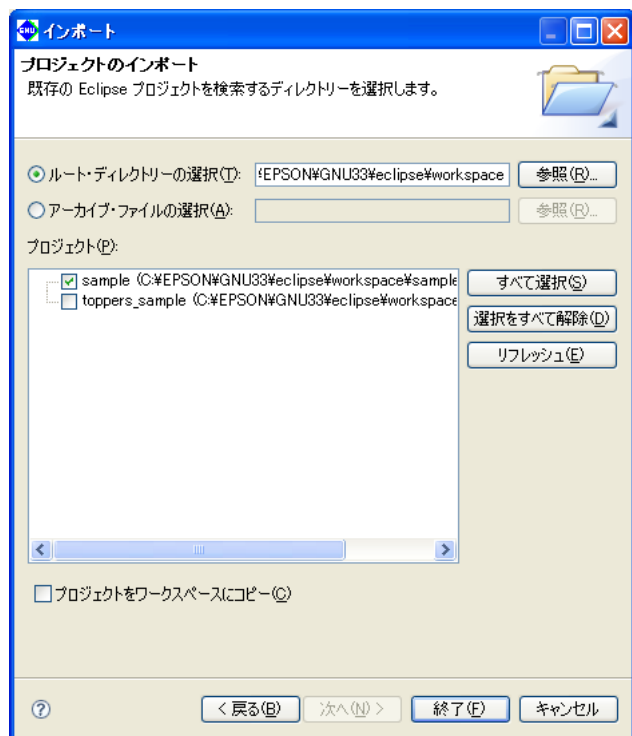
[インポート]ウィザードが起動します。



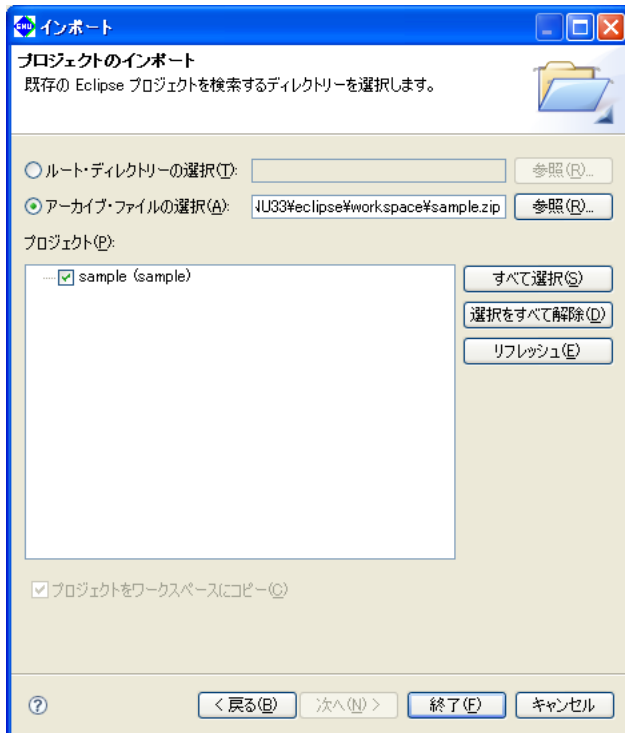
(2) 一覧から[既存プロジェクトをワークスペースへ]を選択し、[次へ>]をクリックします。



(3) プロジェクトフォルダが圧縮されていない場合は、[ルート・ディレクトリーの選択]ラジオボタンを選択し、[参照...]ボタンをクリックして表示されるフォルダ選択ダイアログで、インポートするプロジェクトフォルダが存在するディレクトリを選択してください。



プロジェクトフォルダが圧縮されている場合は、[アーカイブ・ファイルの選択]ラジオボタンを選択し、[参照...]ボタンをクリックして表示されるファイル選択ダイアログで、プロジェクトのアーカイブファイルを選択してください。



ルートディレクトリまたはアーカイブファイルの選択により、その中に含まれるプロジェクトが[プロジェクト:]リストボックス内に表示されます。インポートするプロジェクトのチェックボックスを選択してください(複数選択可)。

[すべて選択]ボタンはリスト内のすべてのプロジェクトを選択し、[選択をすべて解除]ボタンはリスト内のすべてのプロジェクトの選択を解除します。

[リフレッシュ]ボタンは一覧表示を最新の状態に更新します。

[プロジェクトをワークスペースにコピー]チェックボックスは、プロジェクトフォルダをワークスペースディレクトリにコピーするかどうかの選択に使用します。

プロジェクトをコピーしない場合

[プロジェクトをワークスペースにコピー]のチェックを外します。プロジェクトはワークスペースにコピーされず、編集操作等は元のプロジェクトフォルダ内のファイルに対して行われます。また、プロジェクトを削除すると、元のプロジェクトフォルダが削除されます。

プロジェクトをコピーする場合

[プロジェクトをワークスペースにコピー]を選択します。指定のプロジェクトフォルダがワークスペースにコピーされ、編集操作等はワークスペース内のファイルに対して行われます。元のプロジェクトフォルダ内のファイルに変更の影響は及びません。

元のファイルを変更したくない場合は、必ず[プロジェクトをワークスペースにコピー]を選択してください。

アーカイブファイルの選択時は、常にプロジェクトがワークスペースにコピーされます。

※ワークスペースディレクトリには、プロジェクトディレクトリ(.projectファイルが含まれるディレクトリ)を指定しないでください。プロジェクトインポート(プロジェクトをワークスペースにコピーがONのとき)に失敗することがあります。

(4) [終了]ボタンをクリックします。

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューにインポートしたプロジェクトが表示されます。

他の環境(PC)で作成したプロジェクトについても、プロジェクトフォルダごと現在のPC上にコピーしてインポートすると、ビルドを実行することができます。ただし、ビルドオプションで独自のインクルード検索パスやライブラリのパスを設定している場合は、インポート後に変更する必要があります。

●プロジェクトファイルの自動更新

旧バージョンのIDEで作成されたプロジェクトをインポートすると、プロジェクトファイル(.project/.cproject/.gnu33project/)が現IDEのバージョンと互換性を持つように自動的に更新されます。

なお、旧バージョンのプロジェクトで使用されていた.cdtprojectファイルは、.cprojectファイルに置き換えられます。

プロジェクトインポート時に.cprojectファイルが生成され、.cdtprojectファイルの内容が自動的に移行されます。

●ディレクトリ構造やリソースの位置について

プロジェクトフォルダ内にすべてのリソースが集約されていれば、どこにコピーしても問題はありません。コピー先でも修正なしにビルドが行えます。

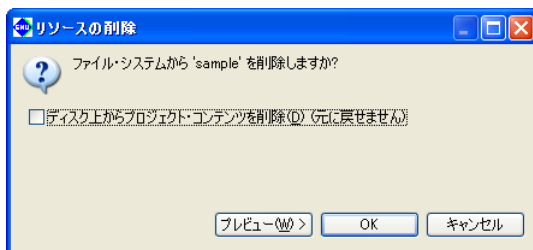
オリジナルのプロジェクトフォルダ内から参照している外部ファイルがあった場合、同じディレクトリ構造の環境であれば修正等は必要ありません。チュートリアル2でも説明したように、makeファイルなどをIDEの自動生成ではなく、外部に用意している場合などは、パスの記述に修正が必要になる場合があります。

標準ライブラリやインクルードディレクトリは、ツールをC:\¥EPSON¥gnu33など、同じディレクトリにインストールしてあれば問題はありません。ツールディレクトリが異なる環境のIDEで作業する場合、ユーザのライブラリとインクルードディレクトリは必要に応じてプロジェクトの[プロパティ]ダイアログボックスの[GNU33 ビルドオプション]で変更してください。

5.4.6 プロジェクトの削除

不要なプロジェクトは次の手順で削除できます。

- (1) [C/C++ プロジェクト]または[ナビゲーター]ビューで削除するプロジェクトをクリックして選択します。
 - (2) 以下のいずれかの操作を行います。
 - [編集]メニューから[削除]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[削除]を選択します。
 - [Del]キーを押します。
- [リソースの削除]ダイアログが表示されます。



※ファイルシステム上のディレクトリとファイルはそのまま残す場合

- (3) [ディスク上からプロジェクト・コンテンツを削除]のチェックボックスはOFFにします。
この場合、[C/C++ プロジェクト]と[ナビゲーター]ビューからはプロジェクトの表示が消えますが、ファイルは残ります。同じプロジェクトをインポートすれば(5.4.5節参照)、再度プロジェクトとして作業を継続可能です。

※ファイルシステム上のディレクトリとファイルもすべて削除する場合

- (3') [ディスク上からプロジェクト・コンテンツを削除]のチェックボックスをONにします。

注: この選択は、ディスク上のファイルもすべて削除され、プロジェクトを復活できなくなりますので十分に注意してください。

- (4) [OK]ボタンをクリックすると削除されます。中止する場合は[キャンセル]ボタンをクリックしてください。

注: 制限文字数以上のプロジェクト名でプロジェクトを作成してしまった場合にプロジェクトフォルダが削除できない場合があります。その場合は、IDEを終了し、コマンドプロンプト等のシェル上から同プロジェクトのフォルダ名をリネームした上で削除してください。

●Windows Vistaでプロジェクトの削除ができない場合

Windows Vista上で、[リソースの削除]ダイアログの[ディスク上からプロジェクト・コンテンツを削除]を選択してプロジェクトを削除しようとしたとき、[リソースの削除のリファクタリングの処理中に例外がキャッチされました]のエラーメッセージが表示され、プロジェクトフォルダが削除できないことがあります。

これは、プロジェクトをビルドした時、プロジェクトフォルダをカレントディレクトリとしてconime.exe(コマンドプロンプトで日本語入力を可能にする)が起動しビルド後も終了しないため、プロジェクトフォルダが削除できなくなります。

この場合には、タスクマネージャなどからconime.exeのプロセスを終了させるか、PCを再起動させてから、プロジェクトフォルダを削除してください。

5.4.7 プロジェクトの名称変更

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内でのプロジェクト名の変更は以下のように行います。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内で、名称を変更するプロジェクトを選択します。
- (2) 以下のいずれかの操作を行います。
 - [ファイル]メニューから[名前変更..]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[名前変更]を選択します。
- (3) ビュー内のプロジェクト名が編集モードになりますので、新しい名称を入力して[Enter]キーを押します。

この操作はファイルシステムにも反映されます。

プロジェクト名を変更した場合、プロジェクトのフォルダ名が変更されるとともに、以下の部分も自動的に変更されます。

- コマンドファイル(<プロジェクト名>_gnu33IDE.cmd)の名称
元の</!プロジェクト名>_gnu33IDE.cmdは削除されません。
- プロジェクト内の以下のファイルの名称(次回ビルド時に変更)
 - 実行形式オブジェクトファイル(<プロジェクト名>.elf)
 - メイクファイル(<プロジェクト名>_gnu33IDE.mak)*
 - リンカスクリプトファイル(<プロジェクト名>_gnu33IDE.lds)*
 - パラメータファイル(<プロジェクト名>_gnu33IDE.par)*
- * これらのファイルは、プロジェクト名の変更時に削除され、次のビルド時に新たに生成されます。
すでに生成されているelfファイルは削除されず、前の名称のまま残ります。
- デバッガの起動設定
[デバッグ構成]ダイアログの設定内容が新たなプロジェクト名に従って変更されます。

注: • プロジェクト名には半角英数字とアンダーバーのみが使用可能です。それ以外の記号が含まれている場合、ビルドやその他の操作時にエラーが発生することがありますので注意してください。

- プロジェクト名を変更した場合、コマンドファイルはテンプレートの内容で新たに生成され、元のコマンドファイルの内容はコピーされません。</!プロジェクト名>_gnu33IDE.cmdファイルの内容をコピーし、[プロパティ]ダイアログの[GNU33 GDBコマンド]ページに貼り付けするなどしてコマンドを修正してください。
- 元のプロジェクトのコピーを同じワークスペースに共存させたい場合は、“プロジェクトの貼り付け”のあと、さらに“プロジェクトのリネーム”を行ってください。

- 例) projectプロジェクトと、project2プロジェクトを共存させる場合。
1. projectプロジェクトをコピーします。
 2. projectTempという名称でプロジェクトの貼り付けをします。
 3. projectTempプロジェクトをproject2にリネームします。

上記の手順で、元のプロジェクトprojectと、新しいプロジェクトproject2を同じワークスペース内で共存できます。

- [C/C++ プロジェクト]ビューや[ナビゲーター]ビューのコンテキストメニューでプロジェクトをコピー/ペーストした場合は、上記のようなプロジェクト内容の自動的な変更は行われませんので注意してください。

5.4.8 プロジェクト内のリソース操作

ここでは、[C/C++ プロジェクト]または[ナビゲーター]ビュー上で可能なリソースの作成やインポート、コピーや移動、削除といった操作の方法について説明します。

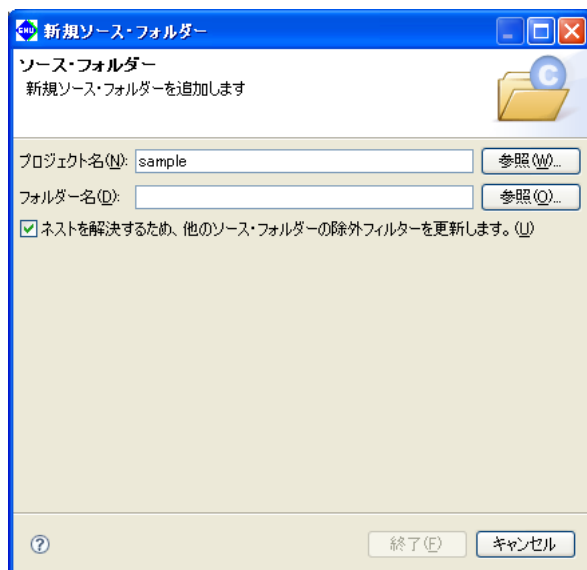
●新規ソースフォルダの作成

プロジェクト内にソースファイル用フォルダを作成できます。
プロジェクトで使用するソースファイルはソースフォルダ内に格納してください。ソースフォルダに格納したソースファイルは、**IDE**により自動的にメイクファイルに追加されます。また、これらのソースから生成されるオブジェクトファイルがリンクされるようにリンカスクリプトファイルも更新されます。なお、プロジェクトフォルダは、プロジェクト作成時にソースフォルダに設定されます。ソースフォルダを作成する手順は次のとおりです。

(1) 以下のいずれかの操作を行います。

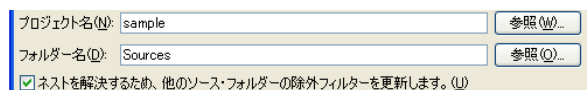
- ・ [ファイル]メニューから[新規]>[ソース・フォルダー]を選択します。
- ・ [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[ソース・フォルダー]を選択します。
- ・ ツールバーの[新規]ショートカットから[ソース・フォルダー]を選択します。
- ・ ツールバーの[新規 C/C++ ソース・フォルダー]ボタンをクリックします。

[新規ソース・フォルダー]ダイアログが表示されます。



(2) [プロジェクト名:]には現在のプロジェクト名が表示されます。他のプロジェクト内にソースフォルダを作成する場合は、[参照...]ボタンでプロジェクトフォルダを選択します。

(3) [フォルダー名:]に作成するソースフォルダの名称を入力します。

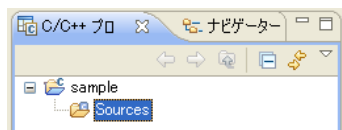


プロジェクトフォルダ直下などにすでにソースファイルがインポートされている場合は、[ネストを解決するため、他のソース・フォルダーの除外フィルターを更新します。]を選択します。このチェックボックスを選択せずにソースフォルダを作成すると、これらのソースファイルの再インポートが必要になります。

プロジェクトフォルダにソースファイルを置いている場合は、必ずこのチェックを入れてください(デフォルトはチェックがONになっています)。

(4) [終了]ボタンをクリックします。中止する場合は[キャンセル]ボタンをクリックしてください。

作成したフォルダがビューに表示されます。



ファイルシステム上には通常のフォルダ(ディレクトリ)が作成されますが、ソースフォルダとして作成することにより、その中のソースファイルがメイクの対象となります。

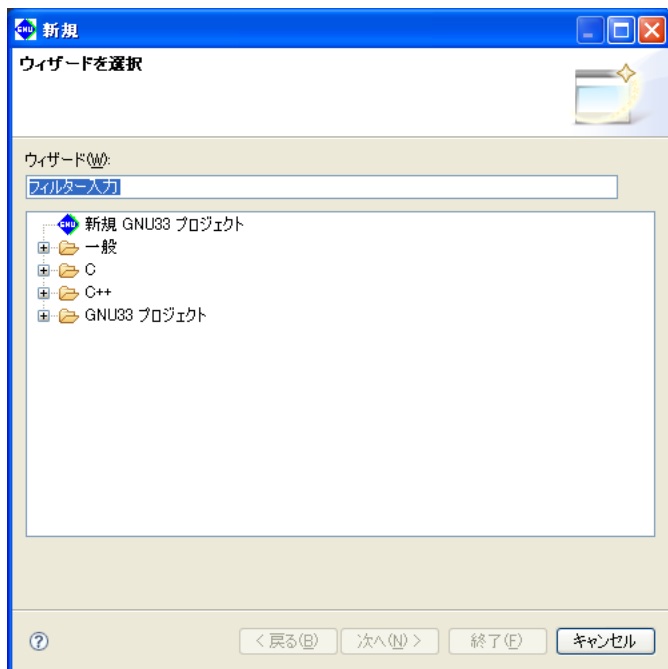
●新規汎用フォルダの作成

プロジェクトまたはプロジェクト内にあるフォルダの中に新たなフォルダを作成できます。その手順は次のとおりです。

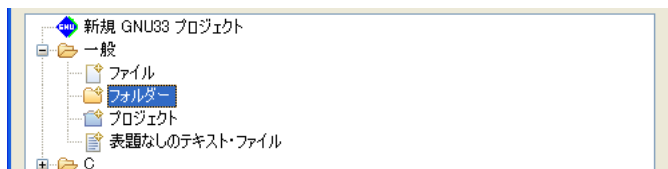
(1) 以下のいずれかの操作を行います。

- [ファイル]メニューから[新規]>[その他...]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[その他...]を選択します。
- ツールバーの[新規]ショートカットから[その他...]を選択します。

[新規]ダイアログが表示されます。



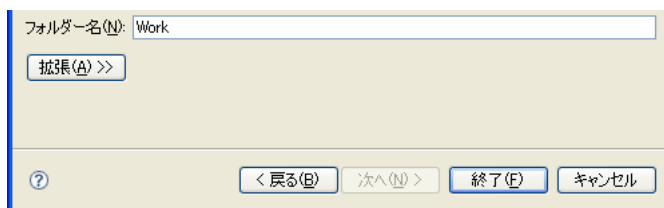
(2) リストの[一般]から[フォルダー]を選択します。



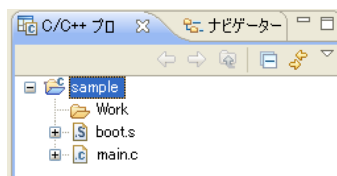
(3) [次へ>]をクリックします。



- (4) フォルダのツリー表示から、新たなフォルダ(サブディレクトリ)を作成するプロジェクトまたは親フォルダをクリックして選択します。
- (5) [フォルダー名:]に作成するフォルダの名称を入力します。



- (6) [終了]ボタンをクリックします。中止する場合は[キャンセル]ボタンをクリックしてください。作成したフォルダがビューに表示されます。



注: このフォルダ内のソースファイルはメイクの対象となりません。メイクの対象とする場合は、ソースファイルをソースフォルダに置いてください。

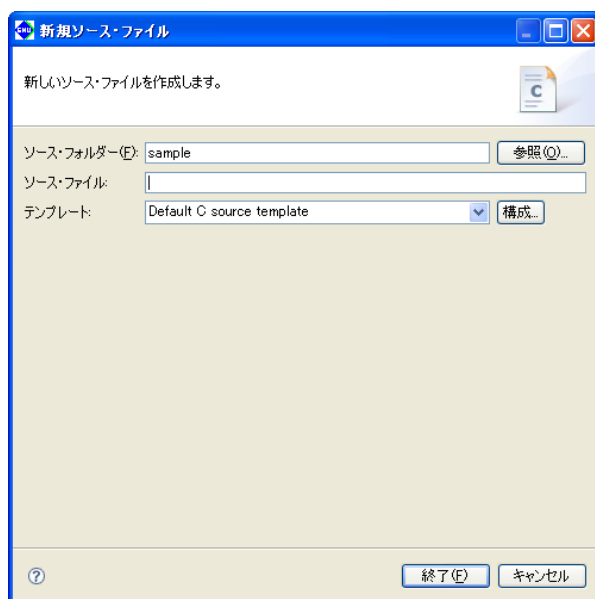
●新規ソースファイル/ヘッダファイルの作成

プロジェクト内に新たなソースファイル/ヘッダファイルを作成できます。その手順は次のとおりです。

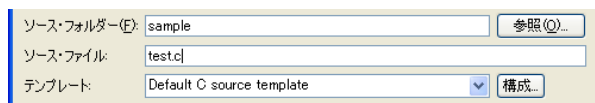
(1) 以下のいずれかの操作を行います。

- [ファイル]メニューから[新規]>[ソース・ファイル](ソースファイルを作成する場合)または[ヘッダー・ファイル](ヘッダファイルを作成する場合)を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[ソース・ファイル]または[ヘッダー・ファイル]を選択します。
- ツールバーの[新規]ショートカットから[ソース・ファイル]または[ヘッダー・ファイル]を選択します。
- ツールバーの[新規 C/C++ ソース・ファイル]ショートカットから[ソース・ファイル]または[ヘッダー・ファイル]を選択します。
- ツールバーの[新規 C/C++ ソース・ファイル]ボタンをクリックします(ソースファイルを作成する場合)。

[新規ソース・ファイル](または[新規ヘッダー・ファイル])ダイアログが表示されます。

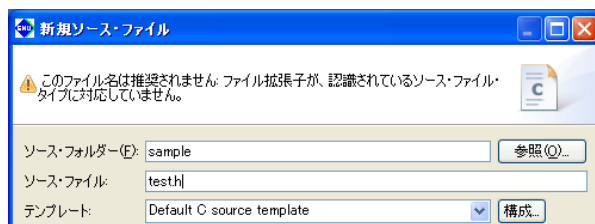


(2) [ソース・ファイル:] (または [ヘッダー・ファイル:]) に作成するファイルの名称を入力します。



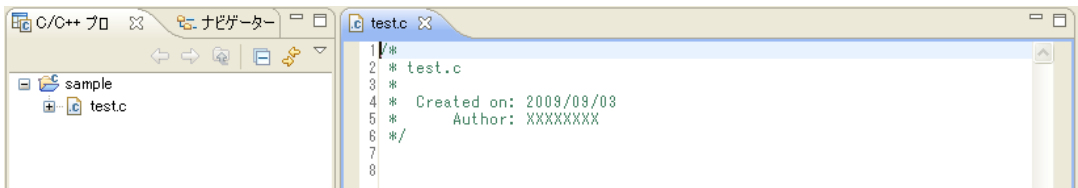
Cソースファイルを作成する場合は拡張子を".c"、C++ソースファイルを作成する場合は".cpp"、アセンブラソースファイルを作成する場合は".s"として入力してください。

対応しているソースファイルの拡張子が入力されないと、次のメッセージが表示されます。ただしこの場合でも、その名称でファイルを作成することは可能です。

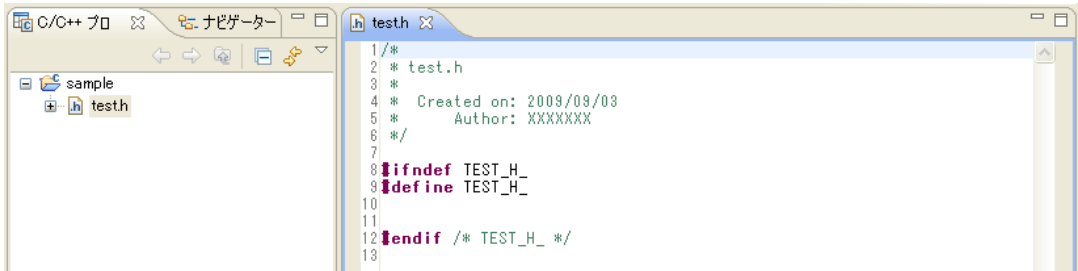


[ソース・フォルダー:]には、現在作業中のプロジェクトフォルダ名が表示されます。他のディレクトリに作成する場合は、パスを入力するか、[参照...]ボタンでディレクトリを選択してください。

(3)[終了]ボタンをクリックします。中止する場合は[キャンセル]ボタンをクリックしてください。
作成したファイルがビューに表示され、エディタで開かれます。



メニューから[ヘッダー・ファイル]を選択して新規のヘッダファイルを生成した場合は、ファイル内に次のようなマクロ定義(<ファイル名>_H_)が自動的に記述されます。



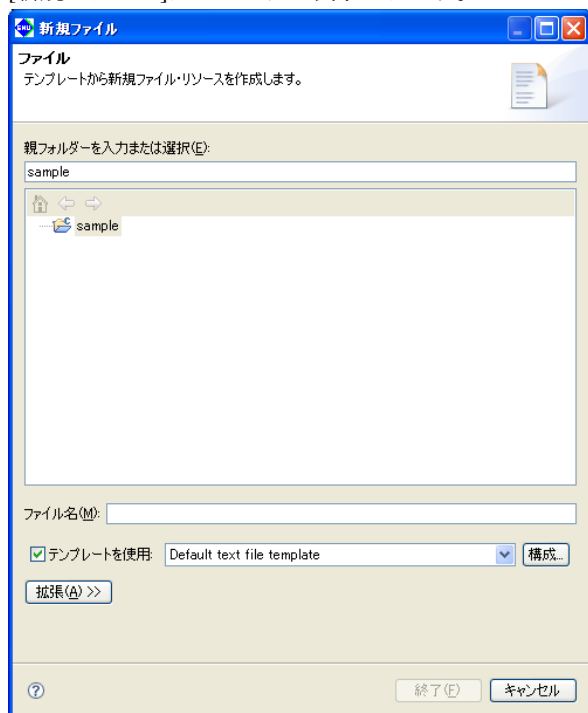
●新規汎用テキストファイルの作成

プロジェクトまたはプロジェクト内にあるフォルダの中に新たなテキストファイルを作成できます。その手順は次のとおりです。

(1) 以下のいずれかの操作を行います。

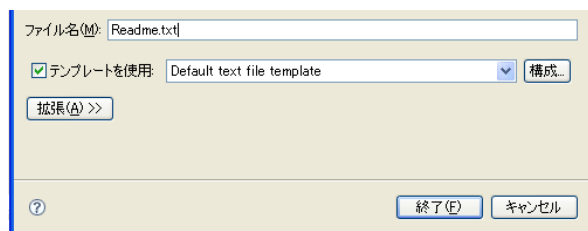
- [ファイル]メニューから[新規]>[テンプレートからファイル]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[テンプレートからファイル]を選択します。
- ツールバーの[新規]ショートカットから[テンプレートからファイル]を選択します。
- ツールバーの[新規 C/C++ ソース・ファイル]ショートカットから[テンプレートからファイル]を選択します。

[新規ファイル]ダイアログが表示されます。

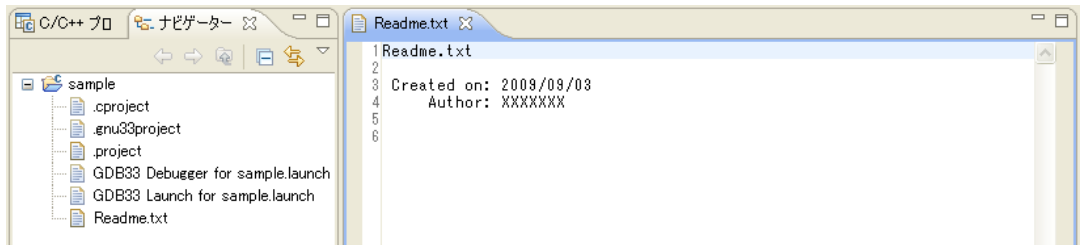


(2) フォルダのツリー表示から、新たなファイルを作成するプロジェクトまたは親フォルダを選択します。

(3) [ファイル名:]に作成するファイルの名称を入力します。



(4)[終了]ボタンをクリックします。中止する場合は[キャンセル]ボタンをクリックしてください。
作成したファイルがビュー *に表示され、エディタで開かれます。



* 作成したファイルの拡張子が[C/C++ プロジェクト]/[ナビゲーター]ビューのフィルタで表示禁止に設定されている場合は表示されません。

●新規クラスの作成

プロジェクト内にC++の新規クラスを記述したソースファイルとヘッダファイルを生成することができます。その手順は次のとおりです。

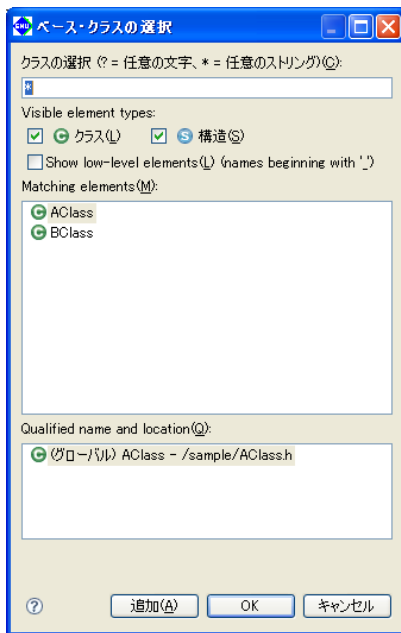
- (1) 新規クラスを作成するプロジェクトを選択します。
- (2) 以下のいずれかの操作を行います。
 - ・ ツールバーの[新規 C/C++ クラス]ボタンをクリックします。
 - ・ [ファイル]メニューから[新規]>[その他...]を選択し、表示されるダイアログ上のリストから[C++]>[クラス]を選択します。
 - ・ [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[その他...]を選択し、表示されるダイアログ上のリストから[C++]>[クラス]を選択します。
 - ・ ツールバーの[新規]ショートカットから[その他...]を選択し、表示されるダイアログ上のリストから[C++]>[クラス]を選択します。

[新規 C/C++ クラス]ダイアログ(詳細は5.10.3節参照)が表示されます。

- (3) [クラス名:]に作成するクラスの名称を入力します。

- (4) 他のクラスを継承する場合は、[追加...]ボタンで表示される[ベース・クラスの選択]ダイアログ(詳細は5.10.4節参照)で基本クラスを選択し、[ベース・クラス:]の一覧に入力しておきます。

注: 基本クラスや名前空間を選択するには、プロジェクトプロパティの[C/C++ 一般]>[インデクサー]で[高速 C/C++ インデクサー]を選択しておく必要があります。



[ベース・クラスの選択]ダイアログの[Matching elements:]に現在のプロジェクト内で定義されているクラスと構造体が一覧表示されますので、基本クラスを選択して[OK]ボタンをクリックします。

一覧が多数の場合は、[構造]のチェックを外して表示内容をクラスに絞り込みます。

さらに、[クラスの選択(?=任意の文字、*=任意のストリング)]にクラス名の一部の文字とワイルドカード(?: 任意の1文字、*: 任意の文字列)を入力して指定文字を含むクラスのみを表示させることもできます。

(5) 必要に応じてその他の設定を行ってください(詳細は5.10.3節参照)。

(6) [終了]ボタンをクリックします。中止する場合は[キャンセル]ボタンをクリックしてください。

作成例: AClassを基本クラスとしてBClassを作成

ヘッダファイル (BClass.h)

```
#ifndef BCLASS_H
#define BCLASS_H                ...二重インクルード防止マクロ

#include "AClass.h"              ...基本クラスの宣言をインクルード

class BClass : public AClass{    ...クラス宣言
public:

    BClass();                    ...コンストラクタ
    virtual ~BClass();           ...仮想デストラクタ
};

#endif // BCLASS_H
```

ヘッダファイルには、二重インクルード防止マクロとクラス宣言(コンストラクタ、仮想デストラクタ)が記述されます([メソッド・スタブ:]の設定による。詳細は5.10.3節参照)。

ソースファイル (BClass.cpp)

```
#include "BClass.h"              ...ヘッダファイルのインクルード

AClass::BClass()                 ...コンストラクタ定義
{
}

AClass::~BClass()                ...デストラクタ定義
{
}
```

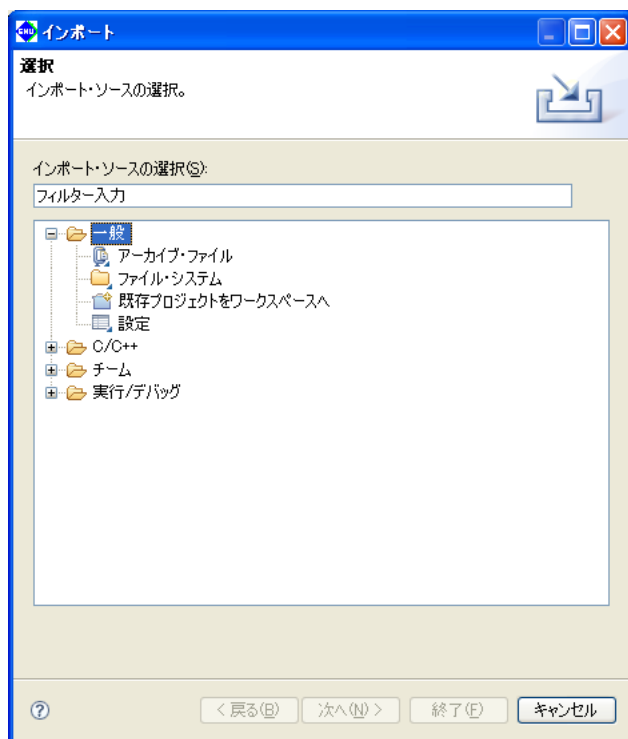
ソースファイルにはヘッダファイルのインクルード、コンストラクタ定義、デストラクタ定義が記述されます([メソッド・スタブ:]の設定による。詳細は5.10.3節参照)。

●既存ファイル、フォルダのインポート

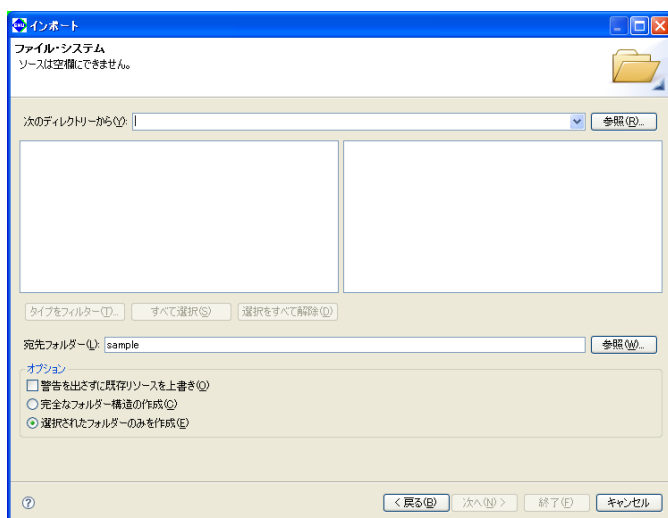
ここでは、既存のファイルやフォルダをプロジェクトに取り込む方法を説明します。ここで説明するインポートは、ファイルシステム上でもファイル/フォルダがプロジェクトディレクトリにコピーされます。既存のソースなどを修正して新たなソースを作成する場合など、オリジナルファイルをそのままの状態に保存しておくことができます。インポートの手順は次のとおりです。

- (1) [C/C++ プロジェクト]または[ナビゲーター]ビューでファイル/フォルダをインポートするプロジェクトまたはフォルダをクリックして選択します。
- (2) 以下のいずれかの操作を行います。
 - [ファイル]メニューから[インポート...]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[インポート...]を選択します。

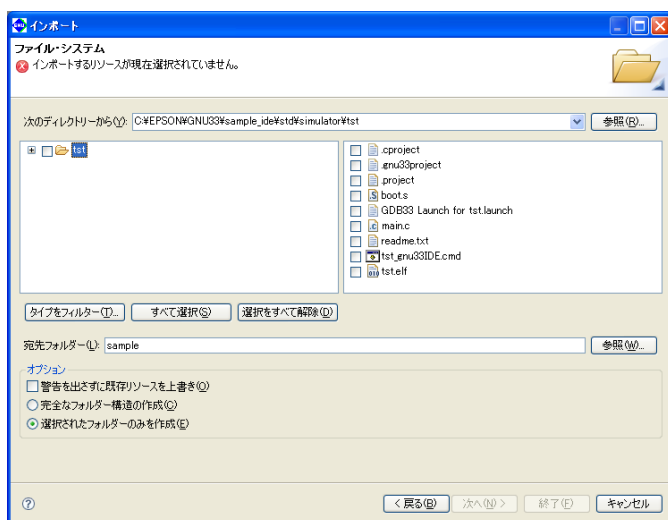
[インポート]ウィザードが起動します。



- (3) 一覧から[一般]>[ファイル・システム]を選択し、[次へ>]をクリックします。



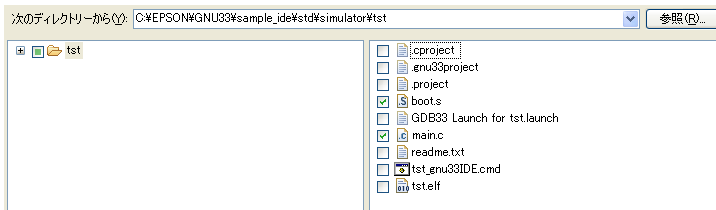
- (4) [次のディレクトリーから]の[参照...]ボタンをクリックして表示されるフォルダ選択ダイアログで、インポートするファイル/フォルダがあるフォルダを選択してください。選択したフォルダへのパスが[次のディレクトリーから]に入力されます。以前インポート元となったフォルダであれば、[次のディレクトリーから]の▼ボタンで表示される履歴から選択することも可能です。



注: ここでは、必ずインポートするファイル/フォルダの親フォルダを選択してください。左側のフォルダリストにルートとして表示されているフォルダ内のファイル/フォルダがインポート対象となります。

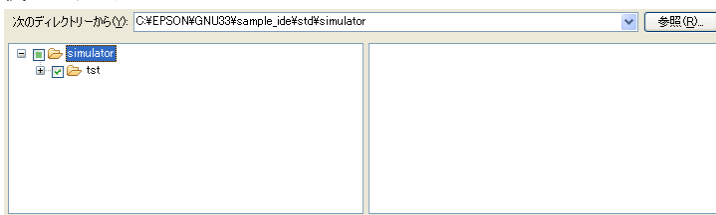
- (5) ファイルをインポートする場合は、右側のリストボックスでインポートするファイルを選択し（チェックマークを付け）ます。

例: ファイルのインポート



フォルダをインポートする場合は左側のリストボックスでツリーリストを展開し、子フォルダの一覧からインポートするフォルダを選択します。孫以下のフォルダを選択してインポートすることも可能ですが、この場合は子フォルダからのディレクトリ構造を含めてインポートされます（ファイルは選択したフォルダ内のもののみインポートされます）。

例: フォルダのインポート



- (6) [終了]ボタンをクリックします。

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューにインポートしたファイル/フォルダが表示されます。

[インポート>ファイル・システム]ダイアログのその他のコントロールについては、5.10.5節を参照してください。

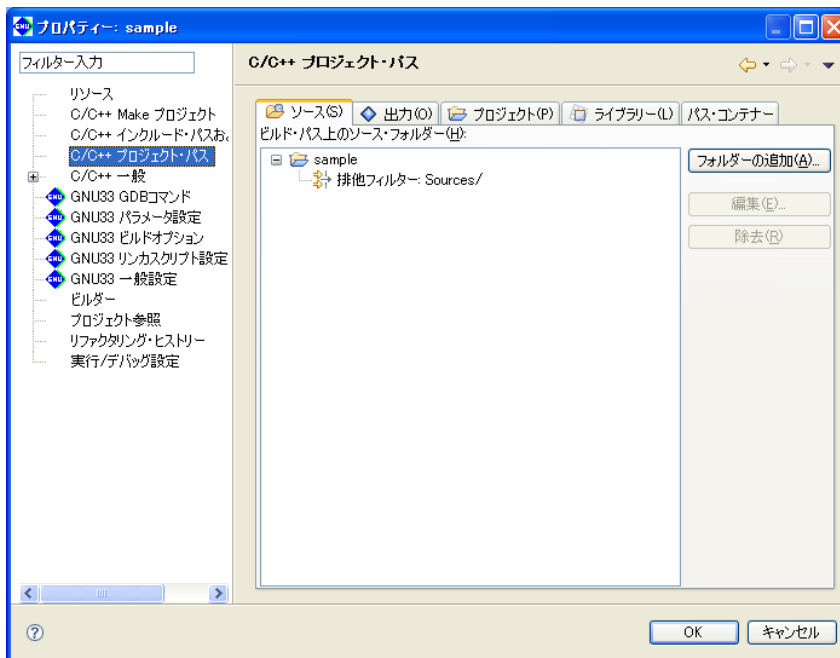
●ソースフォルダの指定

IDEは、プロジェクトフォルダ直下に置かれたソースファイルをmakeファイルに記述してコンパイル/アセンブルの対象にします。プロジェクト内のフォルダに含まれるソースファイルは初期設定状態ではmakeファイルには記述されません。ただし、プロジェクトのプロパティにソースフォルダとして登録することで、その中のソースファイルもメイクの対象になります。その手順は次のとおりです。なお、登録するフォルダがすでにプロジェクト内に作成あるいはインポートされているものとします。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューでソースフォルダを選択するプロジェクトを選択します。
- (2) 以下のいずれかの操作を行います。
 - [プロジェクト]メニューから[プロパティ]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[プロパティ]を選択します。

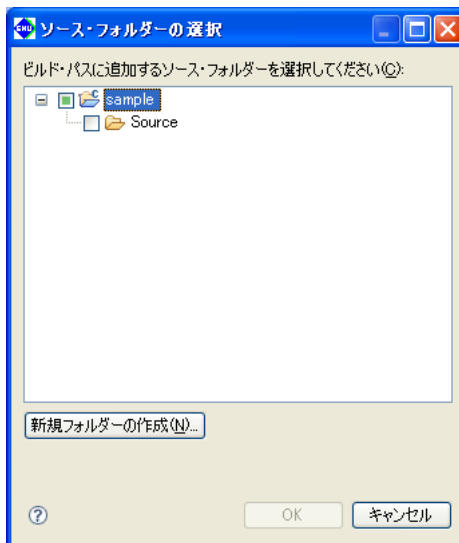
[プロパティ]ダイアログが表示されます。

- (3) プロパティの一覧から[C/C++ プロジェクト・パス]を選択し、[ソース]タブを選択します。

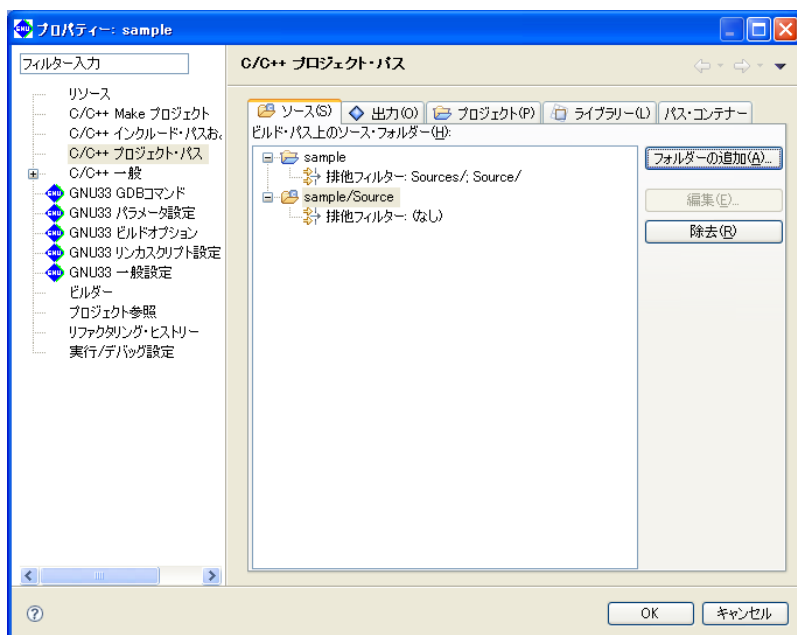


プロジェクトフォルダはデフォルト設定で登録済みです。

- (4) [フォルダーの追加...]ボタンをクリックします。
[ソース・フォルダーの選択]ダイアログが表示されます。



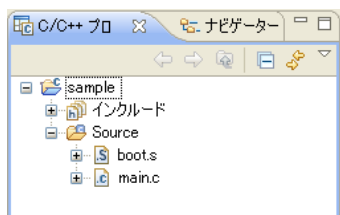
(5) 登録するフォルダを選択し(チェックを付け)、[OK]ボタンをクリックします。



ソースフォルダのリストに登録されます。登録を解除する場合は、ソースフォルダを選択して[除去]ボタンをクリックします。[編集...]ボタン(リストから排他フィルターを選択して使用)は、ソースフォルダ内にあっても、makeファイルに含めないファイルを選択するときに使用します。

(6) [OK]ボタンをクリックして[プロパティ]ダイアログを閉じます。

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内のソースフォルダは、次のようにアイコンにCの印が付きます。



●ファイル、フォルダのコピー / ペースト

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内でも、リソースのコピー / ペーストが行えます。

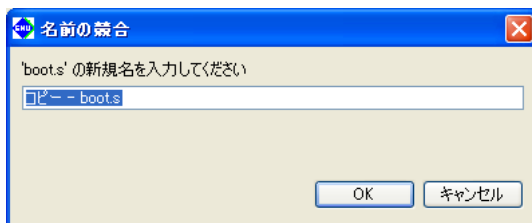
コピー

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内でコピーするファイルまたはフォルダを選択します。
- (2) 以下のいずれかの操作を行います。
 - ・ [編集]メニューから[コピー]を選択します。
 - ・ [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[コピー]を選択します。

選択したリソースがクリップボードにコピーされます。

ペースト

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内でコピーしたファイルまたはフォルダをペーストするプロジェクトまたはフォルダを選択します。
- (2) 以下のいずれかの操作を行います。
 - ・ [編集]メニューから[貼り付け]を選択します。
 - ・ [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[貼り付け]を選択します。
- (3) コピー元にペーストすると次のダイアログが表示されますので、必要に応じて名称を変更してください。



[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内のコピー / ペースト操作はファイルシステム上にも反映されます。

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内でコピーしたリソースは、Windowsのエクスプローラ上にペーストすることも可能です。また、Windowsのエクスプローラ上でコピーしたリソースも、[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内にペーストできます。

●ファイル、フォルダの移動

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内のファイルまたはフォルダの移動は次のように行います。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内で移動するファイルまたはフォルダを選択します。
- (2) 以下のいずれかの操作を行います。
 - ・ [ファイル]メニューから[移動...]を選択します。
 - ・ [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[移動...]を選択します。
- (3) フォルダ選択ダイアログが表示されますので、移動先のフォルダを選択して[OK]をクリックします。

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内の移動操作はファイルシステム上にも反映されます。

この方法は、現在のワークスペース内の移動に限られます。ワークスペース外への移動は、コピー / ペーストまたはエクスポートによりリソースをコピー後、ビュー内のリソースを削除してください。

●ファイル、フォルダのエクスポート

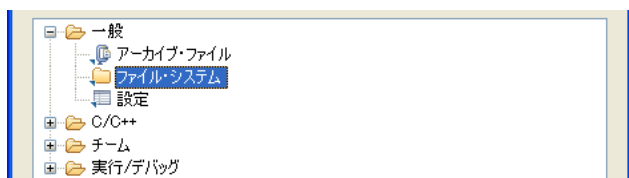
[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内のファイル/フォルダを次の手順でワークスペース外に書き出すことができます。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内で、外部に書き出すファイルまたはフォルダを選択します。
- (2) 以下のいずれかの操作を行います。
 - [ファイル]メニューから[エクスポート...]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[エクスポート...]を選択します。

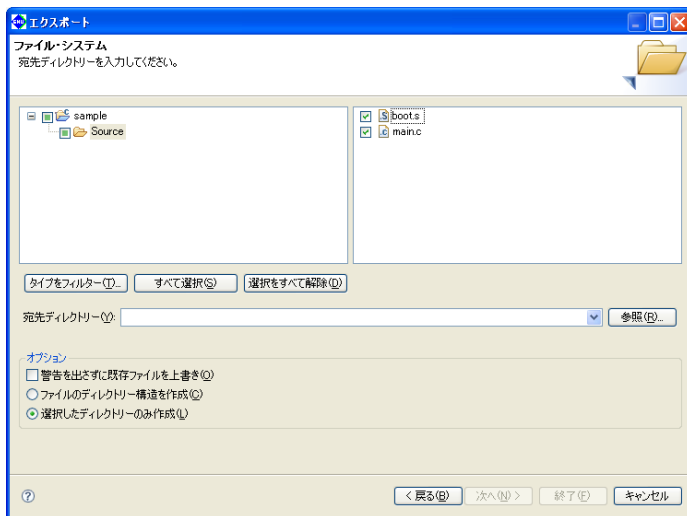
[エクスポート]ウィザードが起動します。



- (3) [一般]を開きます。



- (4) ウィザードの一覧から[ファイル・システム]を選択し、[次へ>]をクリックします。



(5) [宛先ディレクトリ:]の[参照...]ボタンをクリックして表示されるフォルダ選択ダイアログで、書き出し先のフォルダを選択してください。選択したフォルダへのパスが[宛先ディレクトリ:]に入力されます。以前書き出し先となったフォルダであれば、[宛先ディレクトリ:]の▼ボタンで表示される履歴から選択することも可能です。

(6) 必要に応じ、フォルダー一覧とファイル一覧内のチェックボックスを操作して書き出すフォルダ、ファイルを編集します。

(7) [終了]ボタンをクリックします。

選択したフォルダ/ファイルが指定のディレクトリに書き出されます。

[エクスポート>ファイル・システム]ダイアログのその他のコントロールについては、5.10.6節を参照してください。

●ファイル、フォルダの名称変更

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内でのファイル名、フォルダ名の変更は以下のように行います。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内で、名称を変更するファイルまたはフォルダを選択します。
- (2) 以下のいずれかの操作を行います。
 - [ファイル]メニューから[名前変更...]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[名前変更]を選択します。
- (3) ビュー内のファイル/フォルダ名が編集モードになりますので、新しい名称を入力して[Enter]キーを押します。

この操作はファイルシステムにも反映されます。

●ファイル、フォルダの削除

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内でのファイル/フォルダの削除は以下のように行います。

注: • [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューからファイル/フォルダを削除すると、ファイルシステム上の実際のファイル/フォルダも削除されますので注意してください。

- 外部ファイルへのリンクもしくは外部フォルダへのリンクを削除した場合は、リンクのみが削除され、ファイルもしくはフォルダの実体は削除されません。

(1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内で、削除するファイルまたはフォルダを選択します。

(2) 以下のいずれかの操作を行います。

- [Del]キーを押します。
- [編集]メニューから[削除]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[削除]を選択します。

(3) 確認のダイアログが表示されますので、削除する場合は[OK]を、中止する場合は[キャンセル]をクリックします。

●プロジェクト外のファイルへのリンクを張る

チーム開発中では、プロジェクトフォルダの外にあるソースファイルを参照してビルドしたいことがあります。

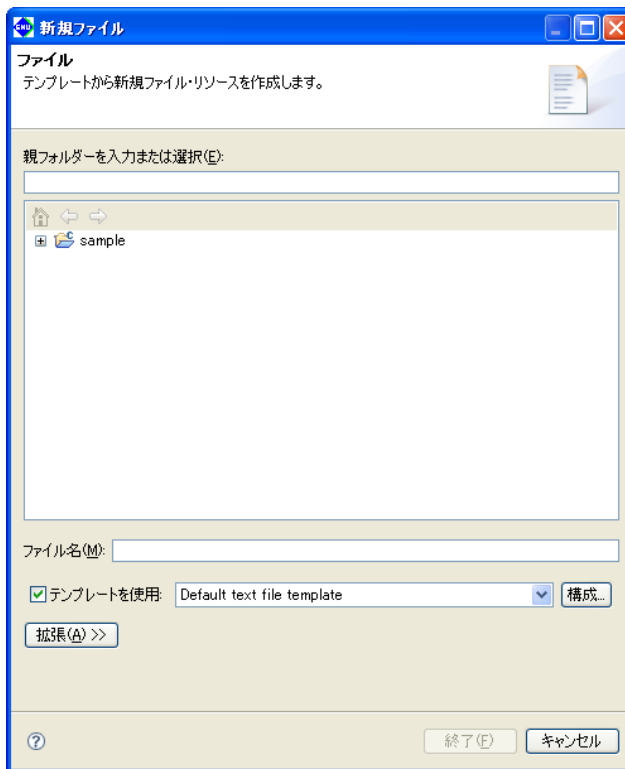
以下の手順で、プロジェクト外にあるファイルに対してリンクを張ることができます。

リンクが張られたファイルは、プロジェクト内に存在するファイルと同じように編集することができます。

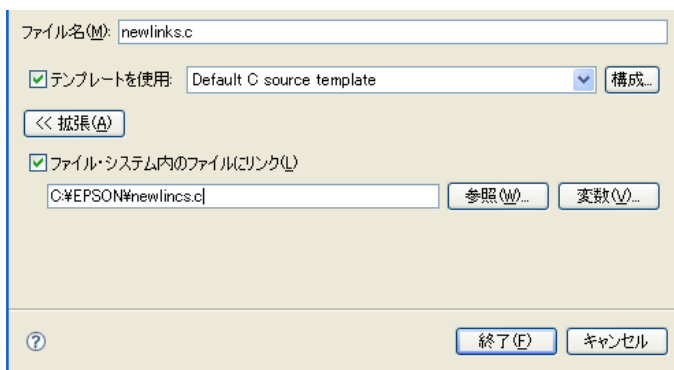
(1) 以下のいずれかの操作を行います。

- [ファイル]メニューから[新規]>[テンプレートからファイル]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[テンプレートからファイル]を選択します。
- ツールバーの[新規]ショートカットから[テンプレートからファイル]を選択します。
- ツールバーの[新規 C/C++ ソース・ファイル]ショートカットから[テンプレートからファイル]を選択します。

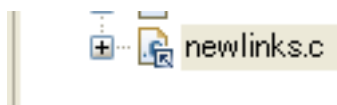
[新規ファイル]ダイアログが表示されます。



- (2) フォルダのツリー表示から、新たなファイルを作成するプロジェクトまたは親フォルダを選択します。
- (3) [ファイル名:]に作成するファイルの名称を入力します。
- (4) [拡張>>]ボタンをクリックします。
- (5) ファイル・システム内のファイルにリンクにチェックマークを付けます。
- (6) [参照...]ボタンをクリックして、参照先のファイルを選択します。



- (7) [終了]ボタンをクリックします。中止する場合は[キャンセル]ボタンをクリックしてください。
作成したファイルがビューに表示され、エディタで開かれます。



注：・リンク元ファイル名(ファイル名)とリンク先ファイル名(ファイル・システム内のファイルにリンク)は同じにしてください。

- ・作成先フォルダには、ソースフォルダを選択してください(プロジェクトフォルダはデフォルトでソースフォルダです)。ソースフォルダ内に作成しなかった場合、ビルド対象になりません。
- ・変数ボタンによるパス指定には対応しておりません。変数は使用しないでください。

●リンクリソースから生成されるオブジェクトファイルについて

プロジェクト外部にリンクされたソースファイルのオブジェクトファイルは、プロジェクトフォルダ直下に生成されます。

●プロジェクト外のフォルダへのリンクを張る

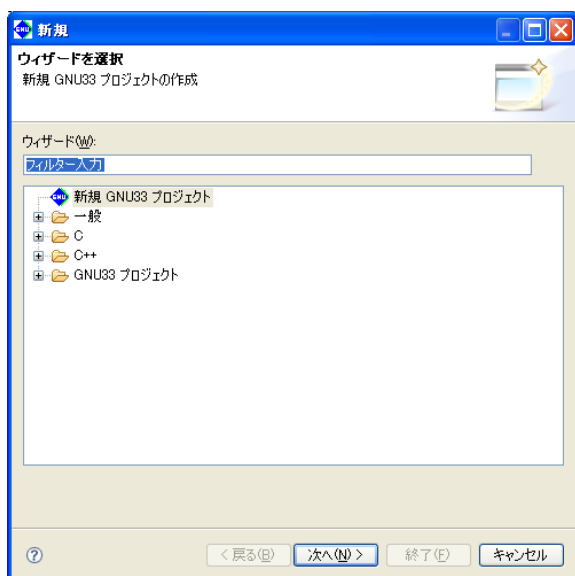
チーム開発中では、プロジェクトフォルダの外にあるソースファイルをフォルダ単位に参照してビルドしたいことがあります。

以下の手順で、プロジェクト外にあるファイルに対してリンクを張ることができます。

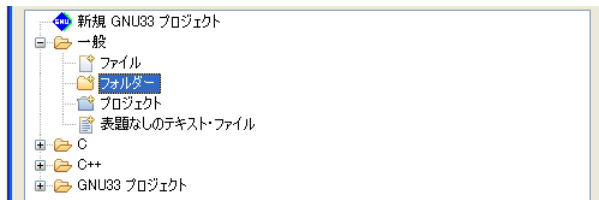
(1) 以下のいずれかの操作を行います。

- ・[ファイル]メニューから[新規]>[その他...]を選択します。
- ・[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[その他...]を選択します。
- ・ツールバーの[新規]ショートカットから[その他...]を選択します。

[新規]ダイアログが表示されます。



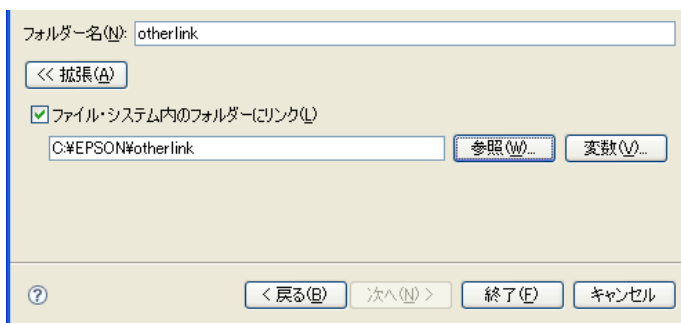
- (2) リストの[一般]から[フォルダー]を選択します。



- (3) [次へ>]をクリックします。

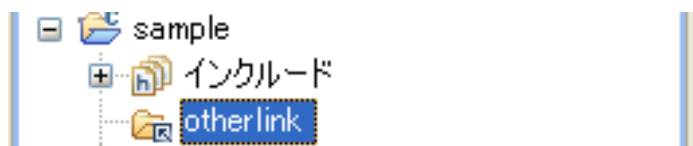


- (4) フォルダのツリー表示から、新たなファイルを作成するプロジェクトまたは親フォルダを選択します。
- (5) [フォルダー名:]に作成するフォルダの名称を入力します。
- (6) [拡張>>]ボタンをクリックします。
- (7) ファイル・システム内のフォルダにリンクにチェックマークを付けます。
- (8) [参照...]ボタンをクリックして、参照先のフォルダを選択します。



- (9) [終了]ボタンをクリックします。中止する場合は[キャンセル]ボタンをクリックしてください。
作成したフォルダがビューに表示されます。

- (10) このフォルダをソースフォルダとして登録します。登録したソースフォルダは、アイコンにCの印が付きます。



注：・リンク元フォルダ名(フォルダー名)とリンク先フォルダ名(ファイル・システム内のフォルダーにリンク)は同じにしてください。

- ・変数ボタンによるパス指定には対応しておりません。変数は使用しないでください。
- ・リンクされたフォルダ内のソースファイルをビルド対象とするため、このフォルダをソースフォルダとして指定する必要があります。詳細については“●ソースフォルダの指定”を参照してください。

● リンクリソースから生成されるオブジェクトファイルについて

プロジェクト外部にリンクされたソースファイルのオブジェクトファイルは、プロジェクトフォルダ直下に生成されます。

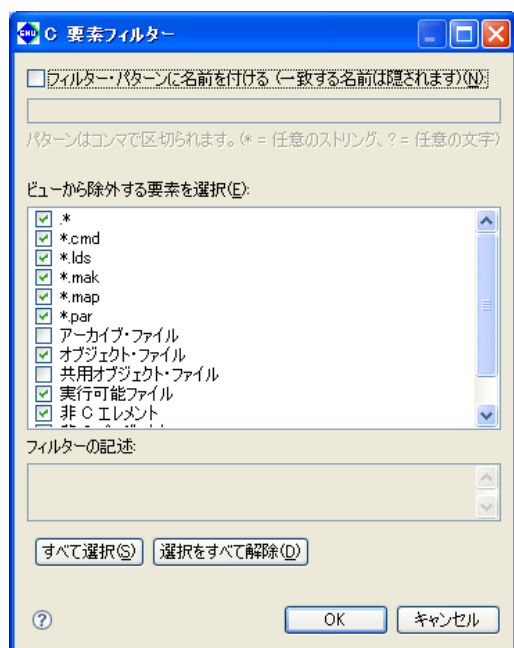
5.4.9 ファイルフィルタ

IDEではファイルの管理やナビゲーションを[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで行います。これらのビューはファイルフィルタを持っており、表示させないファイルの種類を選択できるようになっています。

ファイルフィルタはビューの表示にのみ影響します。たとえば、ビルドに必要なソースファイルを非表示にしても、ビルドは非表示のファイルも含め正しく行われます。

●[C/C++ プロジェクト]ビューのファイルフィルタ

ビューのツールバーメニュー(▽)から[フィルター ...]を選択すると次のダイアログが表示されます。



ここで、表示させないファイルの種類を選択します。

[すべて選択]はすべてを選択、[選択をすべて解除]はすべての選択を解除します。

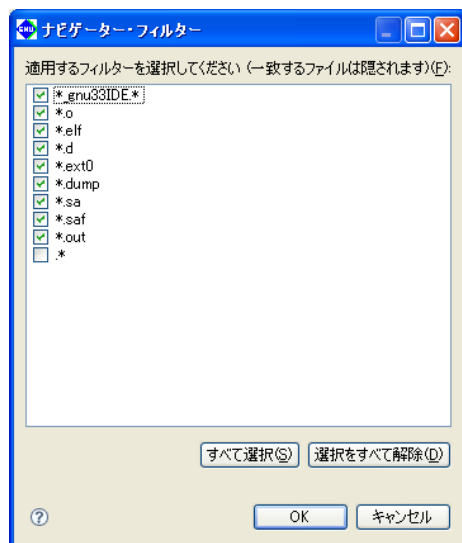
また、[フィルター・パターンに名前を付ける:]チェックボックスを選択し、そのテキストボックスに文字パターンを入力することで、その文字パターンに一致するファイルを非表示にすることができます。このパターン指定には、*(文字列)と?(1文字)のワイルドカードが使用できます。

[C/C++ プロジェクト]ビューの初期設定では、以下のファイルが表示されないようになっています。

- *.*に該当するプロジェクトファイル等
- 実行ファイル(.elfファイル)
- オブジェクトファイル(.oファイル)
- C/C++関連以外のテキストファイル

●[ナビゲーター]ビューのファイルフィルタ

ビューのツールバーメニュー(▽)から[フィルター ...]を選択すると次のダイアログが表示されます。



ここで、表示させないファイルの種類を選択します。

[すべて選択]はすべてを選択、[選択をすべて解除]はすべての選択を解除します。

[ナビゲーター]ビューの初期設定では、以下のファイルが表示されないようになっています。

- *.oに該当するファイル(オブジェクトファイル)
- *.elfに該当するファイル(実行ファイル)
- *.dに該当するファイル(ビルド用依存関係ファイル)
- *.gnu33IDE.*に該当するファイル(IDE用ファイル)
- *.ext0に該当するファイル(コンパイラ出力アセンブラソースファイル)
- *.dumpに該当するファイル(2passビルド用シンボルファイル)
- *.saに該当するファイル(elf実行ファイルのS3形式ファイル)
- *.safに該当するファイル(moto2ffが出力したS3形式ファイル)

5.4.10 ワーキングセット

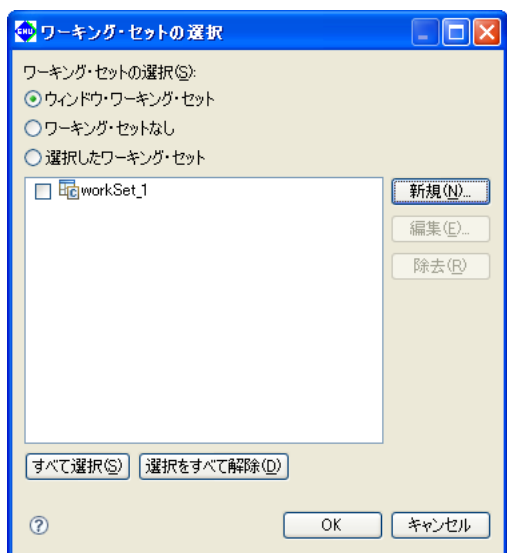
必要なリソースのみを一つのワーキングセットとしてグループ化し、[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューの表示内容を指定のワーキングセットに限定することができます。ここでは、[C/C++ プロジェクト]ビュー上でのワーキングセットの操作方法を説明します。以下の操作は[ナビゲーター]ビューでも同様です。

また、下記の方法で作成したワーキングセットは検索時やビルド時にも、検索範囲の指定やビルドするプロジェクトの指定に利用できます。

●ワーキングセットの選択

注: ワーキングセットはあらかじめ作成しておく必要があります。

- (1) [C/C++ プロジェクト]ビューのツールバーメニュー(▽)から[ワーキング・セットの選択...]を選択します。
[ワーキング・セットの選択]ダイアログが現れ、作成されているワーキングセットの一覧が表示されます。



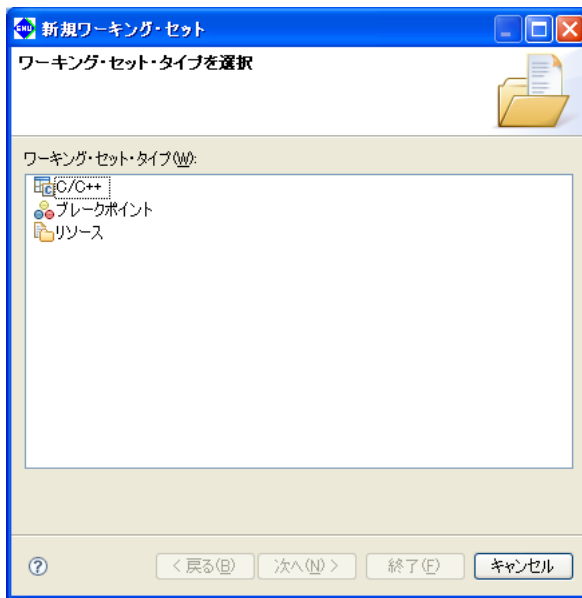
- (2) 一覧からワーキングセットを選択し、[OK]をクリックします。

[C/C++ プロジェクト]ビューの表示が、選択したワーキングセットに定義されているリソースのみになります。

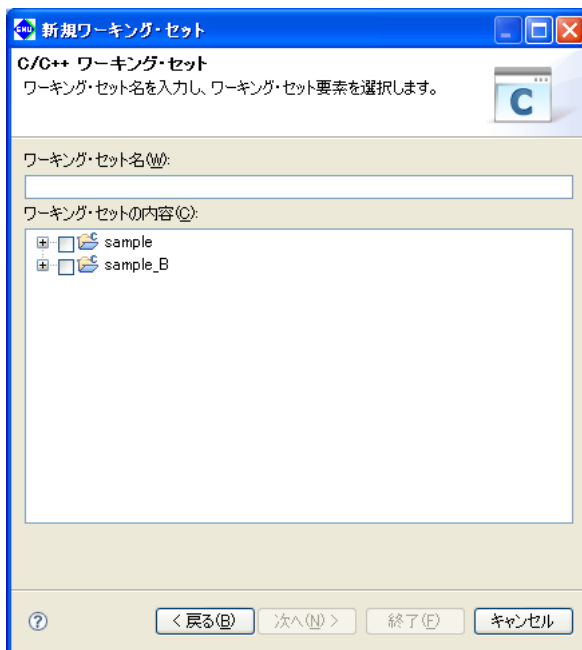
[C/C++ プロジェクト]ビューの表示を元に戻すには、[C/C++ プロジェクト]ビューのツールバーメニュー(▽)から[ワーキング・セットの選択解除]を選択します。

●ワーキングセットの作成

- (1) [C/C++ プロジェクト]ビューのツールバーメニュー(▽)から[ワーキング・セットの選択...]を選択します。
[ワーキング・セットの選択]ダイアログ(上図)が表示されます。
- (2) [新規...]ボタンをクリックします。
[新規ワーキング・セット]ウィザードが起動します。



(3) [ワーキング・セット・タイプ:]の一覧から[C/C++]を選択し[次へ>]をクリックします。

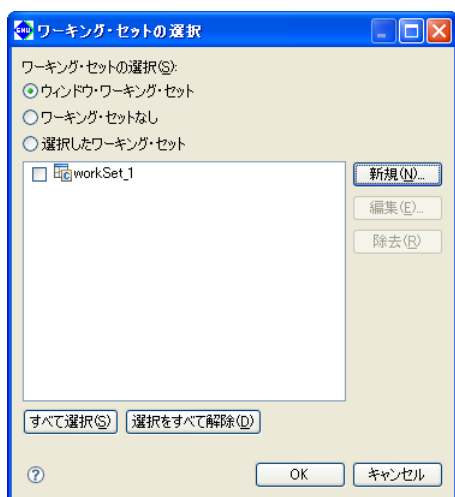


(4) [ワーキング・セット名:]にワーキングセットの名称を入力します。

(5) プロジェクト内のリソースをすべて選択する場合は、プロジェクトフォルダのチェックボックスを選択します。

プロジェクトフォルダ内のリソースの中で表示させるものを個別に選択する場合は、プロジェクトの[+]アイコンをクリックしてリソース一覧を表示させ、必要なものを選択してください。

(6) [終了]ボタンをクリックします。

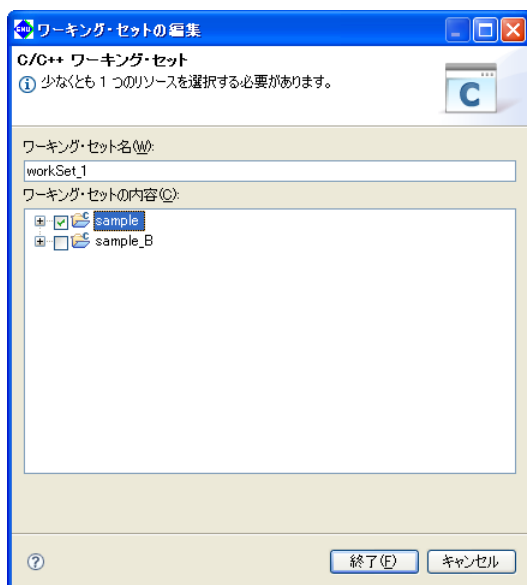


(7) [OK]をクリックするとビューに適用されます。[キャンセル]をクリックすると上記の操作が無効になります。

●ワーキングセットの編集

作成されているワーキングセットのリソース構成は、以下の方法で変更できます。

- (1) [C/C++ プロジェクト]ビューのツールバーメニュー(▽)から[ワーキング・セットの選択...]を選択します。
[ワーキング・セットの選択]ダイアログ(上図)が表示されます。
- (2) 一覧から編集するワーキングセットを選択し、[編集...]をクリックします。
[ワーキング・セットの編集]ダイアログが表示されます。



- (3) 新規作成時と同様にリソースの選択内容を変更し、[終了]をクリックします。
- (4) [OK]をクリックするとビューに適用されます。[キャンセル]をクリックすると上記の操作が無効になります。

現在のビューに選択されているワーキングセットを編集する場合は、[C/C++ プロジェクト]ビューのツールバーメニュー(▽)から[アクティブなワーキング・セットの編集]を選択して直接[ワーキング・セットの編集]ダイアログを表示させることもできます。この場合は、[ワーキング・セットの編集]

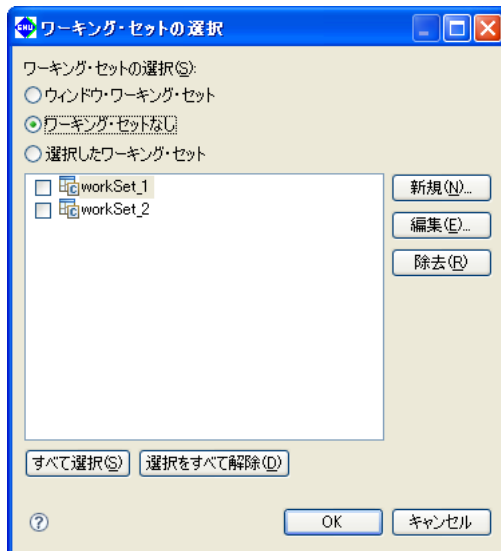
ダイアログを終了すると、変更内容がそのままビューに反映されます。

●ワーキングセットの削除

不要となったワーキングセットは次のように削除します。

- (1) [C/C++ プロジェクト]ビューのツールバーメニュー(▽)から[ワーキング・セットの選択...]を選択します。

[ワーキング・セットの選択]ダイアログが現れ、作成されているワーキングセットの一覧が表示されます。



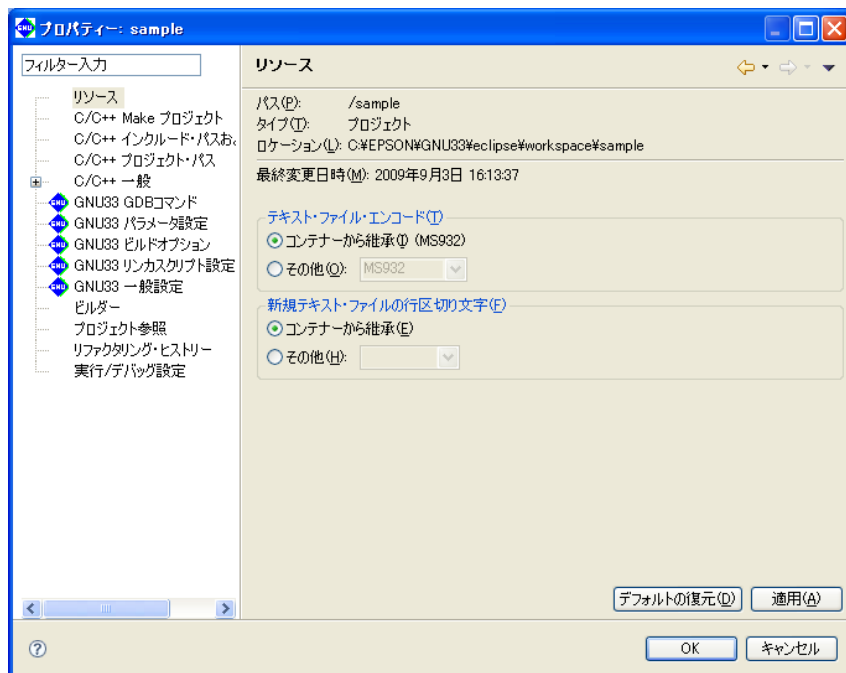
- (2) 一覧から削除するワーキングセットを選択し、[除去]をクリックします。
ビューの表示はワーキングセット非選択の状態に戻ります。
- (3) [OK]ボタンをクリックします。
別のワーキングセットを使用する場合は、[OK]をクリックする前に一覧から選択してください。

5.4.11 プロジェクトプロパティ

プロジェクトは、個々に各種のプロパティを持ち、[プロパティ]ダイアログでそれらを参照/設定できるようになっています。

[プロパティ]ダイアログは次の手順で開きます。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内で、プロジェクトを選択します。
- (2) 以下のいずれかの操作を行います。
 - [プロジェクト]メニューから[プロパティ]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[プロパティ]を選択します。



左のプロパティ一覧にある目的の項目をクリックすると、その設定内容が表示され変更もできます。一覧に表示されているプロパティは以下のとおりです。

1. リソース

プロジェクトディレクトリの位置情報を表示します。また、ソースファイルなどのテキストエンコード形式と行の区切り文字が設定できます。

2. ビルダ

プロジェクトのビルドに使用するビルダを登録、選択します。

3. C/C++ 一般

C/C++に関する一般的な諸設定です。

以下の項目は、**IDE**では使用しませんので通常は操作する必要はありません。

詳しくは、HelpメニューのC/C++ Development User Guide > Reference > C/C++プロパティ > C/C++ Projectプロパティ > C/C++ Generalを参照してください。

コード・スタイル

フォーマッタの設定です。

ドキュメンテーション

プロジェクトで使用するヘルプ用ドキュメントを選択します。

通常は操作しないでください。

ファイル・タイプ

Cリソースとして扱うファイルの名称や拡張子を定義します。
通常は操作しないでください。

インデクサー

Cの検索やコンテンツアシストに使用するインデクサに関する設定を行います。
通常は操作しないでください。

言語マッピング

言語とファイルのマッピングを変更します。
通常は操作しないでください。

4. C/C++ インクルード・パスおよびシンボル

インクルードファイルを検索するパスやプリプロセッサ用のシンボルを定義します。
通常は操作しないでください。

5. C/C++ Make プロジェクト

メイクに関する設定を行います。

6. C/C++ プロジェクト・パス

ソースフォルダ、生成ファイルの出力先などを設定します。

7. GNU33 ビルドオプション

コンパイラ、アセンブラ、リンカのコマンドラインオプションを設定します。

8. GNU33 GDBコマンド

デバグが起動用コマンドファイルを編集します。

9. GNU33 一般設定

ターゲットプロセッサを選択します。

10. GNU33 リンカスクリプト設定

リンカスクリプトを編集します。

11. GNU33 パラメータ設定

デバグで使用するパラメータファイルを編集します。

12. プロジェクト参照

現在のプロジェクトが参照する他のプロジェクトを選択します。

13. リファクタリング・ヒストリー

リファクタリングの履歴を表示します。

14. 実行/デバグ設定

選択中のファイルに起動構成を指定することができます。
通常は操作しないでください。

詳細は、それぞれの内容に関連する節、または5.10.1節を参照してください。

5.5 エディタとソースファイルの編集

IDEにはエディタが組み込まれており、他のエディタ等を使用せずにソースファイルの作成/編集等が行えます。外部エディタを起動させてソースの編集ができるようにも設定できます。

5.5.1 エディタの起動

●エディタの種類

IDEに組み込まれているエディタは3種類に分けられます。使用されるエディタはファイルの種類(拡張子)で決まります。

1. C/C++ エディタ

C/C++ソース(*.c、*.cpp、*.cxx、*.cc、*.C)やヘッダファイル(*.h、*.hh、*.hpp、*.hxx、*.H)の作成/編集に使用します。このエディタでは、C/C++の予約語、コメント、文字列が強調表示されます。また、C/C++の予約語やコードのテンプレートをリストから選択して入力できるコンテンツアシスト機能が使用できます。

このエディタで開かれる文書は、[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー上では次のアイコンで表示されます。



2. アセンブラエディタ

アセンブラソース(*.s、*.S)の作成/編集に使用します。このエディタでは、ラベル、擬似命令、レジスタ名が強調表示されます。

このエディタで開かれる文書は、[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー上では次のアイコンで表示されます。



[C/C++ プロジェクト]ビューのツリー表示は、アセンブラソース内のシンボルおよびラベルの表示には対応していません。

3. テキストエディタ

上記以外の形式(拡張子)のテキストファイルの作成/編集に使用します。

このエディタで開かれる文書は、[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー上では次のアイコンで表示されます。



C/C++ エディタおよびアセンブラエディタが起動するファイルの種類は、C/C++プロジェクトのリソースファイルとして登録されています。登録されているファイルの種類については、[プロパティ]ダイアログの[C/C++ 一般]>ファイル・タイプ]([プロジェクト]メニューから[プロパティ]を選択)、または[設定]ダイアログの[C/C++]>[ファイル・タイプ]([ウィンドウ]メニューから[設定]を選択)を参照してください。

※Word/Excelファイルやバッチファイル

IDEのベースとなっているEclipseはOLEドキュメントに対応しています。このため、".doc"や".xls"などのファイルを開くと、Windowsで定義されているプログラム(WordやExcel)が起動し、IDEのエディタ領域で編集することもできるようになっています。

なお、OLEドキュメントを開くと編集中的の状態になってしまうため、編集作業を行っていない場合でも閉じる際には保存を確認するダイアログが表示されます。

".cmd"や".bat"などの拡張子のファイルは、ダブルクリックするとコマンドプロンプトが起動してしまいます。これらのファイルを編集したい場合は、エディタ領域にドラッグすることで開くことができます。

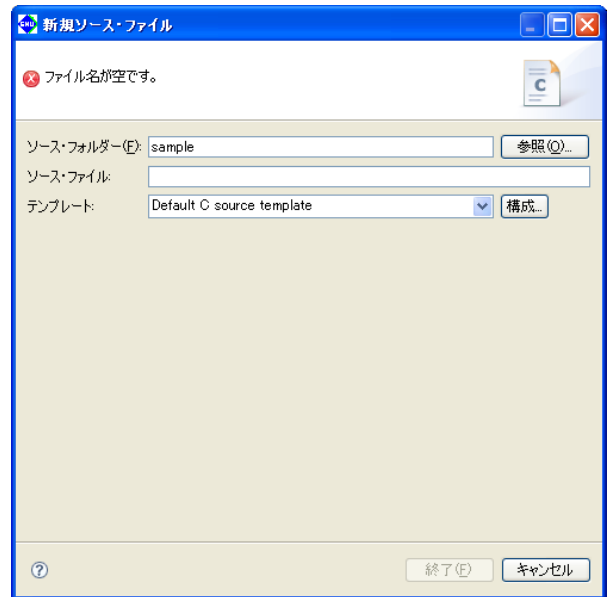
●ソースの新規作成

新規のソース(テキスト)ファイルは次のように作成します。

(1)以下のいずれかの操作を行います。

- [ファイル]メニューから[新規]>[ソース・ファイル]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[ソース・ファイル]を選択します。
- ツールバーの[新規]ショートカットから[ソース・ファイル]を選択します。
- ツールバーの[新規 C/C++ ソース・ファイル]ショートカットから[ソース・ファイル]を選択します。
- ツールバーの[新規 C/C++ ソース・ファイル]ボタンをクリックします。

[新規ソース・ファイル]ダイアログが表示されます。



(2)[ソース・ファイル:]に作成するファイルの名称を入力します。C/C++ソースまたはアセンブラソースを作成する場合は、必ず作成するソースに合った拡張子を付けてください。

(3)[終了]ボタンをクリックします。

●ファイルのオープン

ファイルをエディタで開くには、[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのファイル名(またはアイコン)をダブルクリックします。

5.5.2 基本編集機能

エディタの基本機能や操作方法は一般のエディタと同様です。
代表的な編集機能とメニューコマンドを以下に示します。

表5.5.2.1 基本編集メニューコマンド

編集機能	メニューバー (キーショートカット)	コンテキストメニュー (クリックするビュー)	ボタン/その他の操作
ファイルの新規作成	[ファイル]>[新規]>[ソース・ファイル], [ヘッダー・ファイル], [テンプレートからファイル]	[新規]>[ソース・ファイル], [ヘッダー・ファイル], [テンプレートからファイル] (C/C++ プロジェクト, ナビゲーター)	 , 
ファイルを開く	[ファイル]>[ファイルを開く...]	[開く], [アプリケーションから開く] (C/C++ プロジェクト, ナビゲーター)	[C/C++ プロジェクト]/[ナビゲーター]ビュー上のファイル名をダブルクリック
ファイルを閉じる	[ファイル]>[閉じる] (Ctrl+W)	—	エディタタブの  ボタンをクリック
ファイルをすべて閉じる	[ファイル]>[すべて閉じる] (Ctrl+Shift+W)	—	—
ファイルの保存	[ファイル]>[保管] (Ctrl+S)	[保管] (Editor)	
別名で保存	[ファイル]>[別名保管...]	—	—
すべて保存	[ファイル]>[すべて保管] (Ctrl+Shift+S)	—	—
前回の保存内容に戻す	[ファイル]>[前回保管した状態に戻す]	[ファイルを前回保管した状態に戻す] (Editor)	—
印刷	[ファイル]>[印刷...] (Ctrl+P)	—	
元に戻す	[編集]>[元に戻す] (Ctrl+Z)	[元に戻す] (Editor)	—
やり直し	[編集]>[やり直し] (Ctrl+Y)	—	—
切り取り	[編集]>[切り取り] (Ctrl+X)	[切り取り] (Editor)	—
コピー	[編集]>[コピー] (Ctrl+C)	[コピー] (Editor)	—
貼り付け	[編集]>[貼り付け] (Ctrl+V)	[貼り付け] (Editor)	—
削除	[編集]>[削除] (Delete)	—	—
すべて選択	[編集]>[すべて選択] (Ctrl+A)	—	—
検索	[編集]>[検索/置換...] (Ctrl+F)	—	—
置き換え	[編集]>[検索/置換...] (Ctrl+F)	—	—
検索次候補	[編集]>[次を検索] (Ctrl+K)	—	—
検索前候補	[編集]>[前を検索] (Ctrl+Shift+K)	—	—

5.5.3 C/C++ソース用編集機能

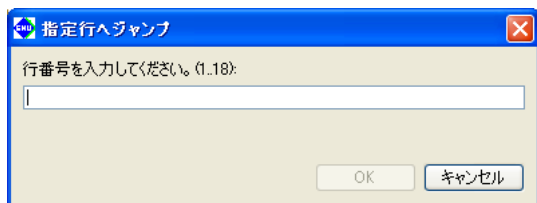
C/C++ エディタは前述の基本編集機能に加え、C/C++ソース専用の機能を持っています。

●指定行へのジャンプ

行番号を指定して、その行にジャンプする機能です。操作手順は次のとおりです。

(1)[ナビゲート]メニューから[指定行へジャンプ...]を選択します。

[指定行へジャンプ]ダイアログが表示されます。



(2)テキストボックスに行番号を入力し、[OK]をクリックします。

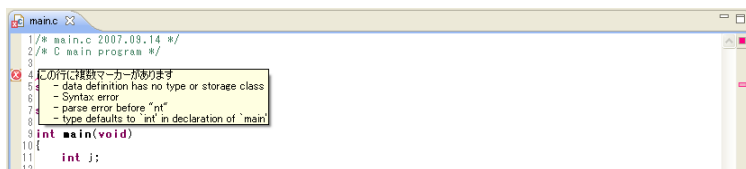
指定行にジャンプします。

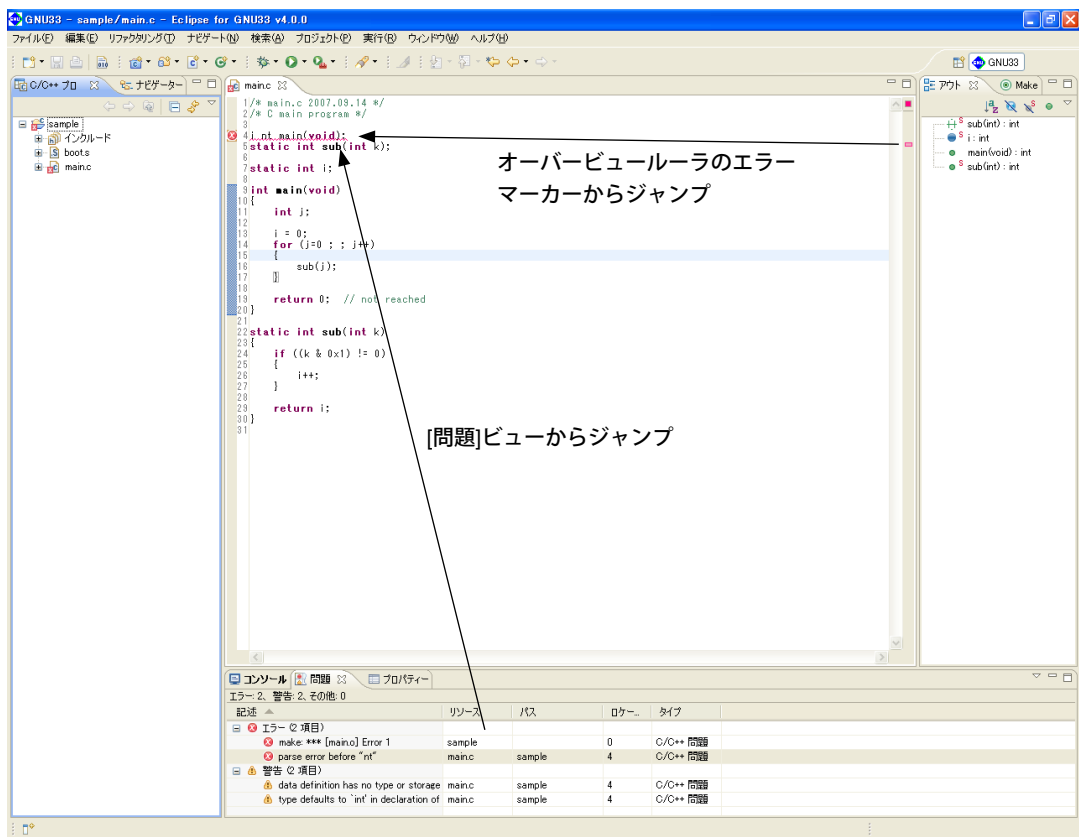
●エラー発生行へのジャンプ

ビルド中にエラーが発生すると、[問題]ビューに発生したエラーの一覧がエラー箇所とともに表示されます。



また、エディタが開いているソースファイルでエラーが発生している場合は、エラー発生行にエラーを示すマーカー (X) が表示されます。このマーカーにマウスカーソルを重ねると、エラーの内容が表示されます。





[問題]ビューはエディタとリンクしており、[問題]ビューに表示されたコンパイルエラーをダブルクリックするとエディタ上の該当行にジャンプします(該当ファイルが閉じている場合は開きます)。ただし、[問題]ビューのエラー情報にソースファイルとエラー発生行が表示されている場合に限られます。

[問題]ビューからのジャンプはアセンブラソースにも対応していますが、C/C++ソースではエディタ上でエラー発生行のナビゲーションが可能となっています。

後方のエラー発生行へのジャンプ

次のいずれかの操作で、現在のカーソル位置より後方の最も近いエラー発生行にジャンプします。

- [ナビゲート]メニューから[次の注釈]を選択します。
- ツールバーの[次の注釈]ボタンをクリックします。

前方のエラー発生行へのジャンプ

次のいずれかの操作で、現在のカーソル位置より前方の最も近いエラー発生行にジャンプします。

- [ナビゲート]メニューから[前の注釈]を選択します。
- ツールバーの[前の注釈]ボタンをクリックします。

このほか、エディタのオーバービューラ(垂直スクロールバーの右)にもファイル全体から見たエラーの位置を示すマーカーが表示されますので、そのクリックによってもエラー行へのジャンプが可能です。

●外部エディタを行番号指定して起動

“5.5.10 外部エディタを行番号指定で起動するには”で外部エディタの設定をしている場合は、コンテキストメニューからエラー発生行にジャンプすることができます。

詳細は“5.5.10 外部エディタを行番号指定で起動するには”を参照してください。

●インクリメンタルサーチ

C/C++ エディタでは文字列の検索に、インクリメンタルサーチ機能が使用可能です。これは、検索文字列を入力していくに従って、1文字ずつ結果に反映される検索機能です。使用方法是次のとおりです。

- (1) エディタ上で検索を行うファイルをアクティブ(最前面)にします。
- (2) [編集]メニューから[次をインクリメンタル検索]または[前をインクリメンタル検索]を選択します。
ウィンドウのステータスバー（最下部）に"インクリメンタル検索"と表示され、エディタがインクリメンタルサーチモードになります。
- (3) 検索を行う文字列を入力します。
文書中には入力されませんのでカーソル位置は問いませんが、[次をインクリメンタル検索]では現在位置より後方が、[前をインクリメンタル検索]では現在位置より前方が検索対象となります。入力に一致した文字列の中で、現在位置から最も近い候補が反転表示されます。検索結果は1文字入力するごとに変わります。誤った文字を入力した場合は[Backspace]キーによって取り消します。入力した検索文字列はステータスバーに現れます。

例: main と入力

```
/* main.c 2007.09.14 *
/* C main program */
:
[m]キー入力
:
/* main.c 2007.09.14 *
/* C main program */
:
[a] [i] [n]キー入力
:
/* main.c 2007.09.14 *
/* C main program */
:
```

注: [←]、[→]、[Enter]、[Esc]キーはインクリメンタルサーチモードを終了しますので、検索終了前に誤って入力しないように注意してください。

- (4) 検索する文字列をすべて入力し終わると、最も近い候補が現れます。
この後は、[↑]または[↓]キーで前後の候補に移動できます。[編集]メニューから[前をインクリメンタル検索]または[次をインクリメンタル検索]を選択しても同様です。

```
/* main.c 2007.09.14 *
/* C main program */
:
[↓]キー入力
:
/* main.c 2007.09.14 *
/* C main program */
```

注: [次をインクリメンタル検索]または[前をインクリメンタル検索]の選択による検索方向に一致する文字列がない場合、ステータスバーには"<文字列> not found"が表示されます。この場合でも文書中には存在する可能性がありますので、[↑]または[↓]キーで前後を確認してください。

- (5) インクリメンタルサーチモードを終了するには、[Enter]キーまたは[Esc]キーを押します。

●インデント

C/C++ソースでは、複数行の記述内容をまとめてタブ1個分右または左にシフトさせることができます。ループ文をコピーした際などに入れ子の状態を合わせるためのインデント調整に利用できます。その手順は次のとおりです。

- (1) エディタ上で、インデントを変更する行にカーソルを置くか、複数行の場合はドラッグして選択します。
- (2) [編集]メニューまたはエディタのコンテキストメニューから、右シフトの場合は[右ヘシフト]を、左シフトの場合は[左ヘシフト]を選択します。

例: main()
 { ←[左ヘシフト]実行
 int j; ←[右ヘシフト]実行
 ↓
 main()
 {
 int j;

選択されたすべての行の記述位置が、設定されているタブ1個分右または左に移動します。

右にシフトした場合、行の先頭にタブが1個挿入されます。

左にシフトした場合、行の先頭がタブであれば1個削除されます。スペースでインデントされている場合は、設定されているタブサイズ分(初期設定は4文字)のスペースが削除されます。たとえば、タブが4文字に設定されている場合に6文字のスペースでインデントされている行は左シフトにより、先頭に2文字分のスペースが残ります。4文字に満たないスペースでインデントされていた場合、左シフトを行ってもインデントは変わりません。

タブのサイズは、[ウィンドウ]>[設定]で表示される[設定]ダイアログ([一般]>[エディター])の[テキスト・エディター]設定)上で変更できます。

●コメントアウト

C/C++ソースでは、複数行のコメントアウトとその解除が簡単にできるようになっています。コメントアウトの手順は次のとおりです。

- (1) エディタ上で、コメントアウトする行にカーソルを置くか、複数行の場合はドラッグして選択します。
- (2) エディタのコンテキストメニューから、[ソース]>[コメント/コメント解除]を選択します。

例: for (j=0 ; ; j++)
 {
 disp_j(); ←[ソース]>[コメント/コメント解除]を実行
 sub(j);
 }
 ↓
 for (j=0 ; ; j++)
 {
 // disp_j();
 sub(j);
 }

選択されたすべての行の先頭に"//"が挿入されます。

コメントアウトした行を元に戻す手順は次のとおりです。

- (1) エディタ上で、コメントアウトを解除する行にカーソルを置くか、複数行の場合はドラッグして選択します。
- (2) エディタのコンテキストメニューから、[ソース]>[コメント/コメント解除]を選択します。

例:

```

for (j=0 ; ; j++)
{
    //          disp_j();   ←[ソース]>[コメント/コメント解除]を実行
    sub(j);
}
↓
for (j=0 ; ; j++)
{
    disp_j();
    sub(j);
}

```

行の先頭にある"//"が削除されます。"//"の前にスペースまたはタブが挿入されていた場合でも、スペースとタブは残し、"//"のみを削除します。

"/*"で始まるコメント行には影響を与えません。

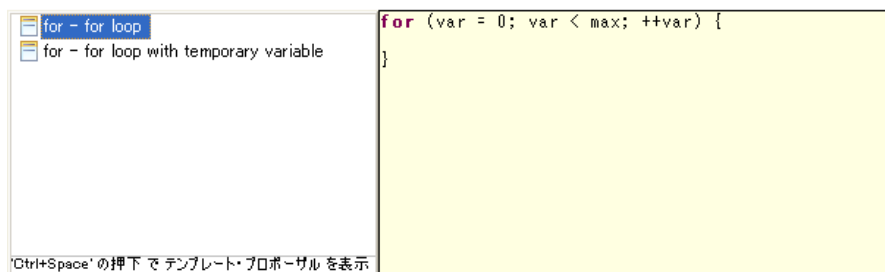
●コンテンツアシスト

C/C++ エディタはコンテンツアシスト機能を持っています。これは、テキストカーソルの位置に記述可能なC/C++の予約語やテンプレートを一覧からの選択によって入力できる機能です。この機能の使用方法は次のとおりです。

- (1) 新しいステートメントを挿入する位置にテキストカーソルを置きます。
- (2) 入力するコードが分かっている場合は、その最初の1～2文字を入力します。これにより、一覧の内容を絞り込むことができます。

例: for文を記述する場合はfを入力します。

- (3) [編集]メニューまたはエディタのコンテキストメニューから[コンテンツ・アシスト]を選択します。



- (4) 一覧から、使用する予約語またはテンプレートを選択し、ダブルクリックします。カーソル位置に選択した内容が挿入されます。テンプレートはループ文や条件文などの定型書式で、文書のアイコンでリストされています。テンプレートの内容は一覧からクリックして選択すると、横に表示されます。汎用のテンプレートはあらかじめ定義されていますが、ユーザ独自のテンプレートを定義することもできます("5.9 IDEのカスタマイズ(設定)"の"C/C++>エディター>テンプレート"参照)。

●リファクタリング

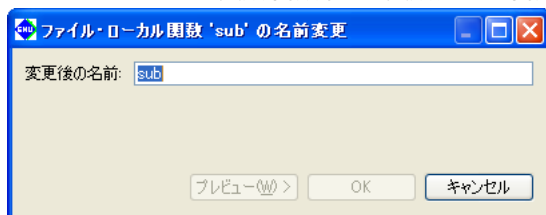
注: リファクタリングは、Eclipse CDTの開発途上にある機能です。

この機能を使用してソースファイルを編集した後は、十分な動作確認を行ってください。

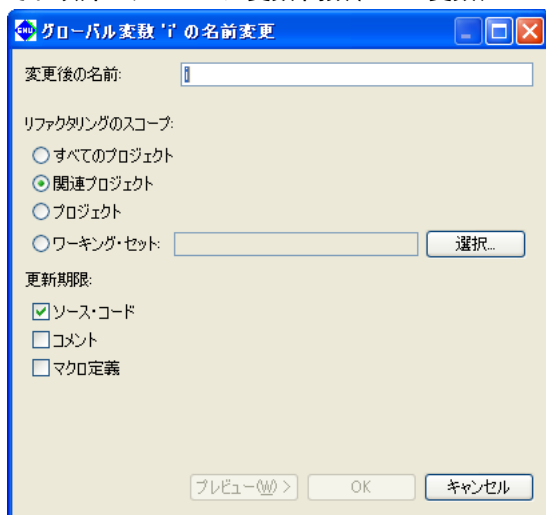
クラス、型、関数などの名称を宣言箇所やすべての参照箇所も含め、一括して変更することができます。操作方法は次のとおりです。

- (1) エディタ、[C/C++ プロジェクト]ビューまたは[アウトライン]ビューから名称を変更するエレメントを選択します。
- (2) [リファクタリング]メニューから[名前変更...]を選択します。あるいは、コンテキストメニューを表示させ、[リファクタリング]>[名前変更...]を選択します。
[名前変更]ダイアログが表示されます。

ファイルスコープの変数/関数 (static変数、static関数、ローカル変数、関数パラメータなど) の場合



それ以外のグローバル変数/関数 (extern変数、extern関数など) の場合



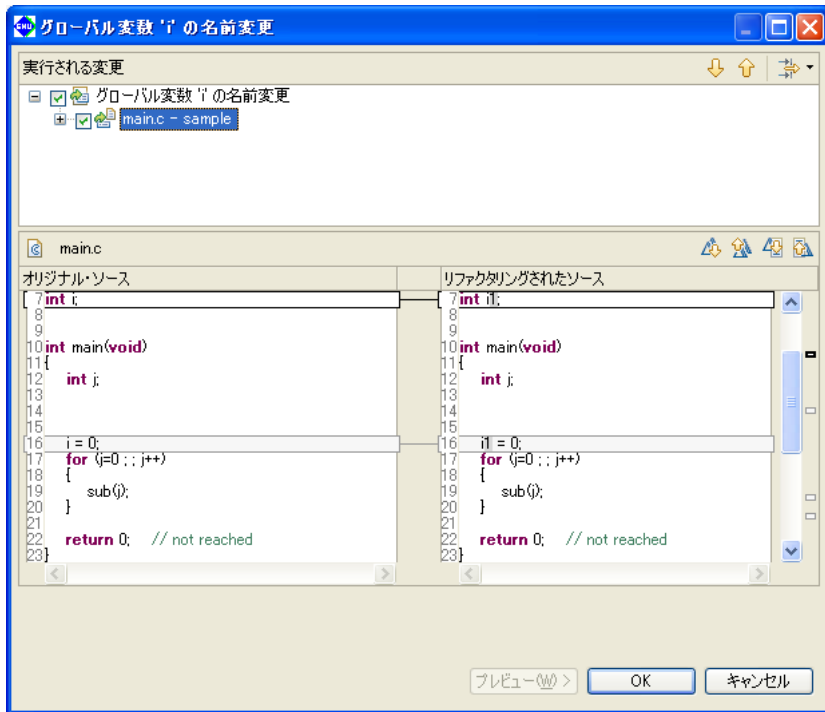
- (3) [変更後の名前:]に新しい名称を入力します。
- (4) [リファクタリングのスコープ:]のラジオボタンで変更する対象を選択します。(グローバル変数/関数)

[すべてのプロジェクト]	開かれているプロジェクトすべてを変更の対象とします。
[関連プロジェクト]	現在編集集中のプロジェクトに關係するプロジェクトすべてを変更の対象とします。
[プロジェクト]	現在編集集中のプロジェクトのみを変更の対象とします。
[ワーキング・セット:]	指定のワーキングセット内のプロジェクトまたはファイルを変更の対象とします。ワーキングセットは[選択...]ボタンで選択します。
- (5) [更新期限]のチェックボックスで変更範囲を選択します。(グローバル変数/関数)

[ソース・コード]	ソースコード内を変更します。
[コメント]	コメント内を変更します。
[マクロ定義]	マクロ定義内を変更します。

ファイルスコープ変数/関数の場合、現在のファイル内のみを変更範囲とします。

- (6) [プレビュー>]ボタンをクリックすると、該当するリソースの一覧が表示されますので、変更しないリソースがあれば、そのチェックを外します。



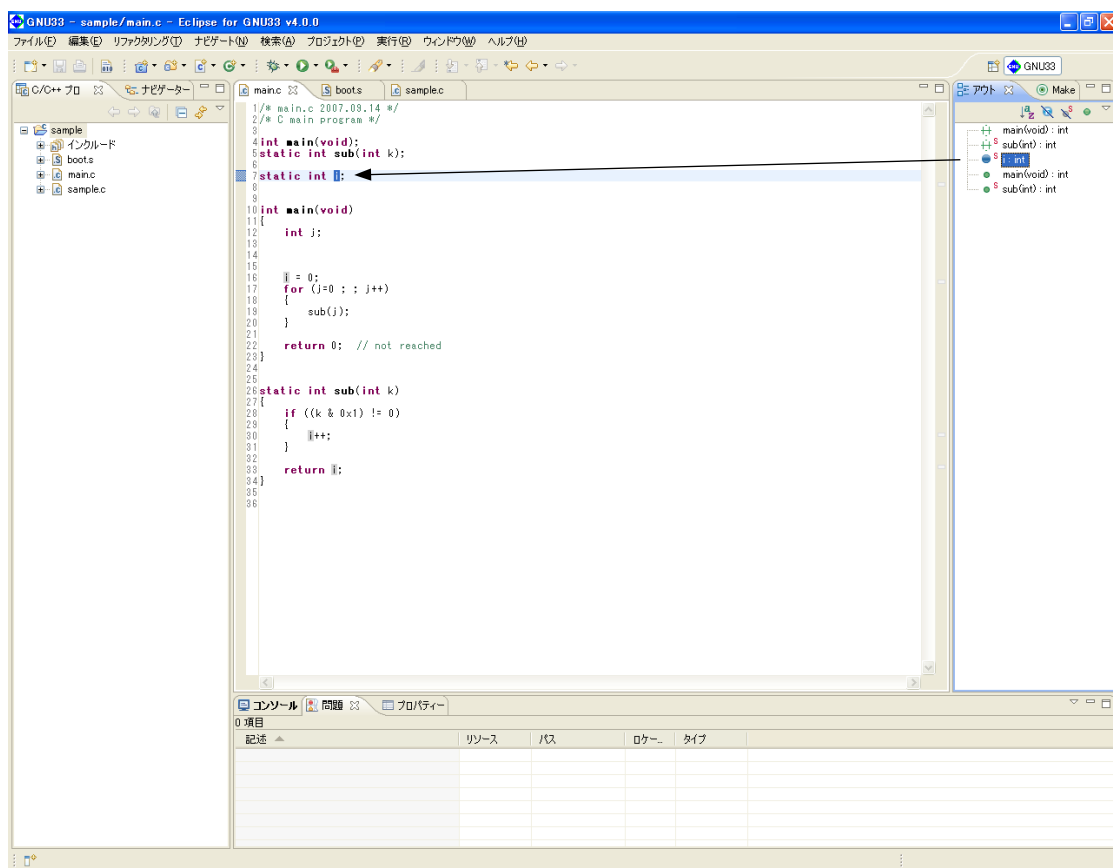
矢印のボタンで、変更箇所を確認することができます。

- (7) 変更する場合は[OK]ボタンを、中止する場合は[キャンセル]ボタンをクリックします。

5.5.4 [アウトライン]ビュー

[アウトライン]ビューは、現在エディタ領域の最前面に開いているC/C++ソース内に定義されている変数や関数などを表示します。

その名称をクリックすることにより、ソース内の定義位置にジャンプします。



5.5.5 ナビゲーションヒストリ

IDEのエディタは、オープンして行き来したファイルの履歴を保持しており、ブラウザのように履歴を前後にたどることができます。これはナビゲーション機能のみで、編集内容は現在の状態から変わりません。

●履歴に沿って戻る

以下のいずれかの操作を行います。

- [ナビゲート]メニューから[戻る]を選択します。
- ツールバーの[戻る]ボタンをクリックします。

これらの操作では、一つ前の履歴に戻ります。

ツールバーの[戻る]ボタンの横にある[▼]をクリックすると、履歴内のファイル一覧が表示されます。この履歴の中からファイルを選択することもできます。

●履歴に沿って進める

以下のいずれかの操作を行います。

- [ナビゲート]メニューから[進む]を選択します。
- ツールバーの[進む]ボタンをクリックします。

これらの操作では、一つ先の履歴に進めます。

ツールバーの[進む]ボタンの横にある[▼]をクリックすると、履歴内のファイル一覧が表示されます。この履歴の中からファイルを選択することもできます。

●直前に編集した箇所にジャンプ

これは、最後に編集したソース行に戻る機能です。以下のいずれかの操作を行います。

- [ナビゲート]メニューから[最後の編集位置]を選択します。
- ツールバーの[最後の編集位置]ボタンをクリックします。

この操作では、常に直前の編集箇所にジャンプし、別の箇所を編集するまで変わりません。

文書中で何らかの入力が行われると、編集したものと見なされます。入力した文字を削除したり、操作を取り消しても履歴は元に戻りません。最後の編集が行の削除だった場合は、1行上に戻ります。

5.5.6 ブックマーク

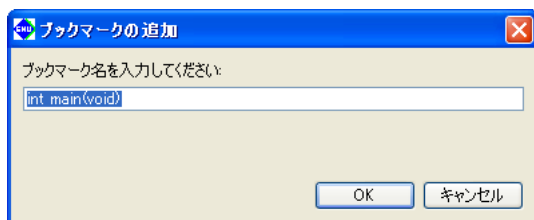
よく参照する箇所(行)には、ブックマーク(しおり)の付加が可能です。ブックマークを付けた行は[ブックマーク]ビューにリストされ、そこから簡単にジャンプすることができます。

●ブックマークの付加

ブックマークの付加は次のように行います。

- (1) ブックマークを付けるソース行にカーソルを置きます。
- (2) 以下のいずれかの操作を行います。
 - [編集]メニューから[ブックマークの追加...]を選択します。
 - エディタのマーカーバー (エディタ領域の左端)を右クリックしてコンテキストメニューを表示させ、[ブックマークの追加...]を選択します。

[ブックマークの追加]ダイアログが表示されます。

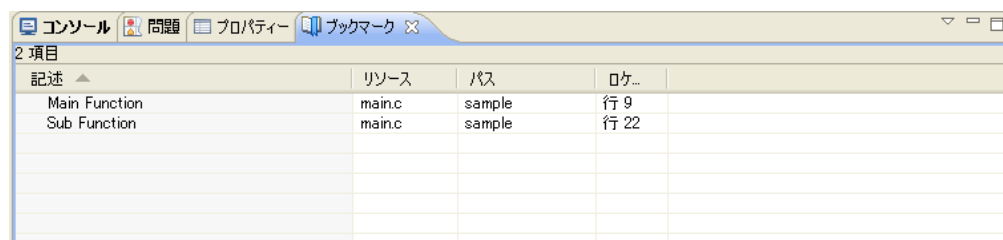


- (3) ブックマーク名を設定します。[ブックマーク名を入力してください]に表示された名称をそのまま使用するか、入力し直して[OK]をクリックします。

エディタのマーカーバーにブックマークのマーカーが表示されます。

ブックマークマーカー → `int main(void)`
`{`
 `int j;`

また、[ブックマーク]ビューを開くと、設定したブックマークが一覧に加えられています。



ブックマーク名は、[記述]欄をクリックして変更できます。

●ブックマークへのジャンプ

ブックマークを付加したソース行へは、[ブックマーク]ビューからジャンプできます。

- (1)[ブックマーク]ビューをアクティブにします。開かれていない場合は、[ウィンドウ]メニューから[ビューの表示]>[ブックマーク]を選択してください。
- (2)以下のいずれかの操作を行います。
 - ・ 目的のブックマークの行内をダブルクリックします。
 - ・ 目的のブックマークの行内を右クリックしてコンテキストメニューを表示させ、[ジャンプ]を選択します。

エディタはブックマーク位置にジャンプします。

●ブックマークの削除

ブックマークが不要となった場合は、エディタ上または[ブックマーク]ビュー上で削除できます。

エディタ上での削除

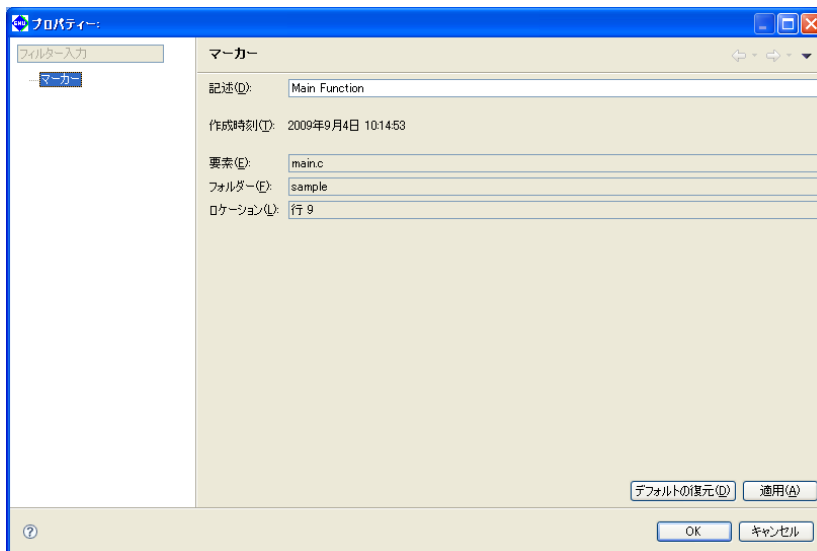
削除するブックマークマーカーを右クリックしてコンテキストメニューを表示させ、[ブックマークの除去]を選択します。

[ブックマーク]ビュー上での削除

- (1)削除するブックマークをクリックして選択します。
- (2)コンテキストメニューを表示させ、[削除]を選択します。

●ブックマーク情報の表示

目的のブックマークの行内を右クリックしてコンテキストメニューを表示させ、[プロパティ]を選択します。[プロパティ]ダイアログが現れ、ビュー内の情報に加え作成日時を表示します。



ここでブックマーク名の変更も行えます。

●ブックマークリストのフィルタと並べ替え

ブックマークが増え[ブックマーク]ビューの一覧が大きくなりすぎたときのために、一覧の表示を制限したり、項目で並べ替えたりすることができます。

フィルタ

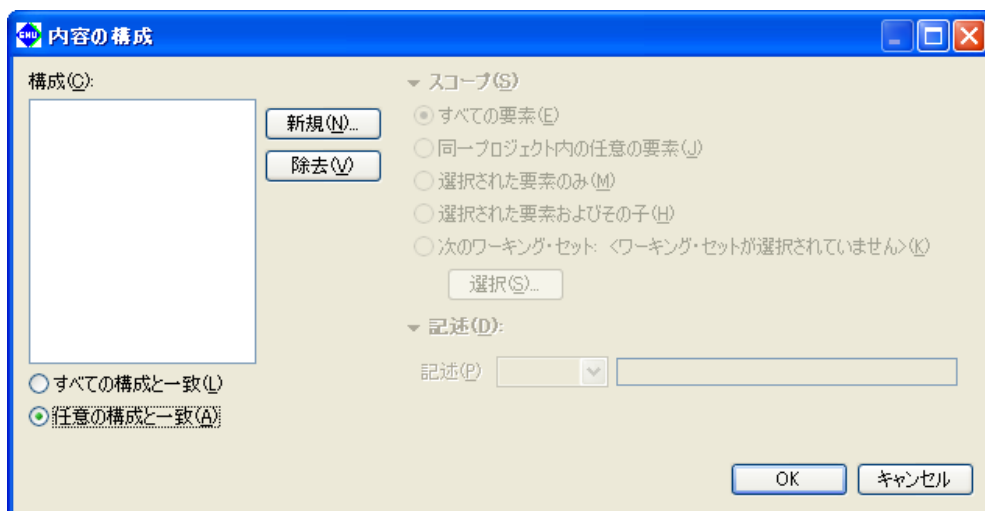
フィルタを使用し、必要なブックマーク以外を非表示にできます。

また、複数のフィルタを作成し、必要に応じて使い分けることができます。

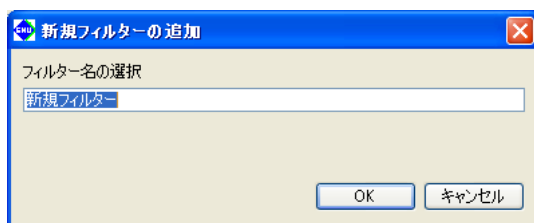
新規のフィルタは以下の手順で作成します。

- (1) [ブックマーク]ビューをアクティブにします。
- (2) ビューメニュー(▽)から[内容の構成...]を選択します。

[内容の構成]ダイアログが表示されます。



- (3) [新規]ボタンをクリックします。

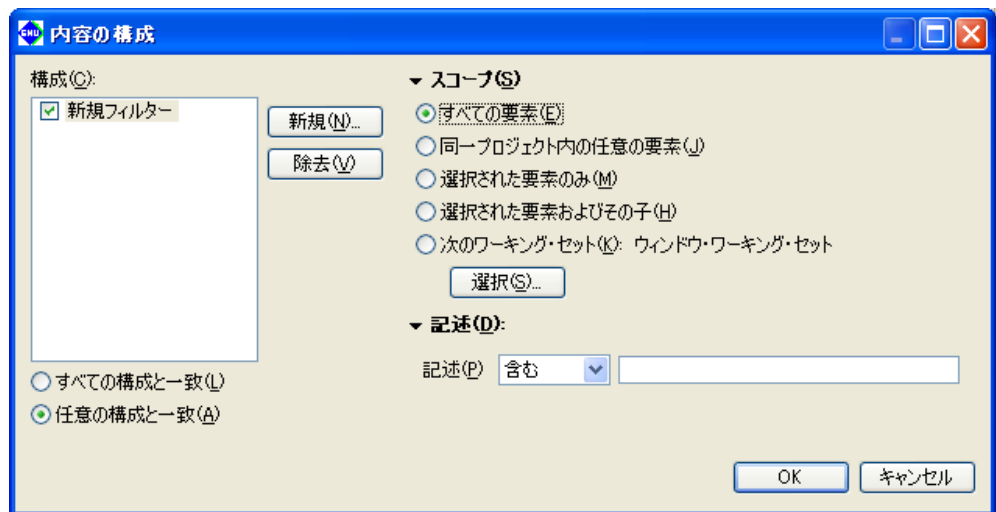


作成するフィルタの名前を入力し、[OK]をクリックします。

- (4) [構成:]内に他のフィルタが表示されている場合は、そのフィルタのチェックを外し、新たに作成したフィルタのチェックボックスをOnにします。

(5) 表示させる条件を設定します。

[すべての要素]	開かれているプロジェクトすべてを対象に、設定されているブックマークを表示します。
[同一プロジェクト内の任意の要素]	現在選択されているプロジェクト内に設定されているブックマークを表示します。
[選択された要素のみ]	[C/C++ プロジェクト]/[ナビゲーター]ビューで選択したファイル、またはエディタでアクティブにしたファイル内に設定されているブックマークのみを表示します。
[選択された要素およびその子]	[C/C++ プロジェクト]/[ナビゲーター]ビューで選択したプロジェクトまたはフォルダ内の全ファイル、もしくは選択したファイル内に設定されているブックマークのみを表示します。
[次のワーキング・セット:]	指定のワーキングセット内に設定されているブックマークを表示します。ワーキングセットは[選択...]ボタンで選択します。
[記述] - [含む]	上記の範囲指定に加え、[記述]項目がテキストボックスに入力した文字列を含むブックマークのみを表示します。この条件を使用しない場合はテキストボックスを空にしておきます。
[記述] - [含まない]	上記の範囲指定に加え、[記述]項目がテキストボックスに入力した文字列を含まないブックマークのみを表示します。この条件を使用しない場合はテキストボックスを空にしておきます。



(6) [OK]をクリックします。

[ブックマーク]ビューは設定した条件に合ったブックマークのみを表示します。

現在のフィルタの条件を変更する場合は、(3)と(4)を省略して条件選択のみを行ってください。

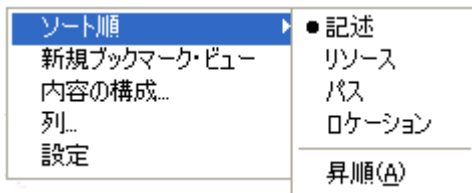
複数のフィルタを作成した場合は、上記ダイアログを表示させ、[構成:]で使用するフィルタを選択します。あるいは、ビューメニューの[表示]のサブメニューから使用するフィルタを選択します。

[内容の構成]ダイアログの詳細については5.10.7節を参照してください。

並べ替え

一覧の項目に優先順位を付け、表示を並べ替えることができます。

- (1)[ブックマーク]ビューをアクティブにします。
- (2)ビューメニュー (▽)から[ソート順]を選択し、一覧のどの項目を優先して並べ替えるかを選択します。また、[昇順]をチェックすると、それぞれの項目を昇順で並べ替え、チェックを外すと降順で並べ替えます。



5.5.7 タスク

ソースの作成時、一部を保留にしたまま先に進む場合など、その位置情報と作業内容などをタスクとして記録しておくことができます。エディタ上ではブックマークと同様なジャンプ先のマーカーですが、タスクには優先順位を指定可能で、[タスク]ビューに表示されるタスクの一覧をTo-Doリストとして使用することができます。

●タスクの作成

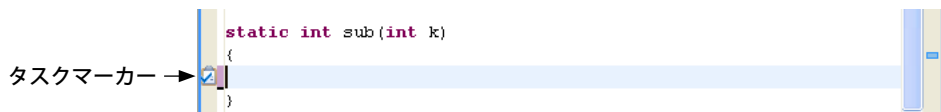
タスクは次のように作成します。

ソース行情報を含める場合

- (1) タスクを設定するソース行にカーソルを置きます。
 - (2) 以下のいずれかの操作を行います。
 - ・ [編集]メニューから[タスクの追加...]を選択します。
 - ・ エディタのマーカーバー(エディタ領域の左端)を右クリックしてコンテキストメニューを表示させ、[タスクの追加...]を選択します。
- [プロパティ]ダイアログが表示されます。

- (3) [記述:]にタスクの説明を入力します。
- (4) [優先度:]から優先順位(高、標準、低)を選択します。
- (5) 完了したタスクとして作成する場合は[完了]を選択します。
- (6) [OK]ボタンをクリックします。

エディタのマーカーバーにタスクのマーカーが表示されます。



また、[タスク]ビューを開くと、作成したタスクが一覧に加えられています。

1 項目	記述	リソース	パス	ロケ...	タイプ
☐	Create Subfunction routine	main.c	sample	行 24	タスク

一覧の左端のチェックボックスはタスクが完了したか否かを示します。終了したタスクはこのボックスをクリックしてチェックを付けます。

その隣の欄は優先順位をアイコンで表示します。

🔴 = 高, 空白 = 標準, ⬇️ = 低

この欄をクリックするとプルダウンリストボックスが表示され、優先順位を再設定することができます。

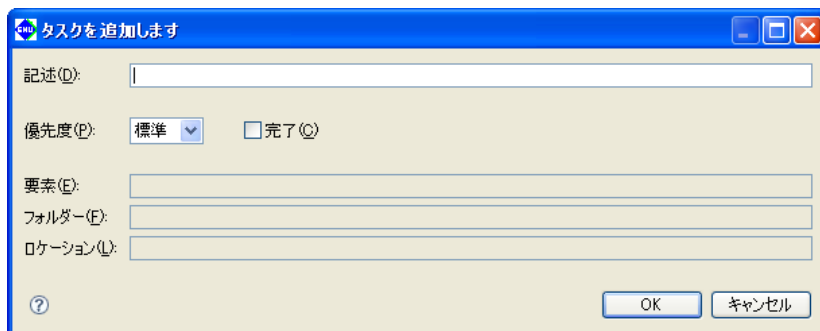
タスクの説明は、[記述]欄をクリックして変更できます。

ソース行情報を含めない場合

特定のソースファイルに関連しないTo-Do項目も作成可能です。

- (1) [タスク]ビューのコンテキストメニューから[タスクの追加...]を選択します。

[タスクを追加します]ダイアログが表示されます。



- (2) [記述:]にタスクの説明を入力します。

- (3) [優先度:]から優先順位(高、標準、低)を選択します。

- (4) 完了したタスクとして作成する場合は[完了]を選択します。

- (5) [OK]ボタンをクリックします。

この場合、リソースや行番号は設定されません。

●タスク設定位置へのジャンプ

タスクを設定したソース行へは、[タスク]ビューからジャンプできます。

- (1) [タスク]ビューをアクティブにします。開かれていない場合は、[ウィンドウ]メニューから[ビューの表示]>[タスク]を選択してください。

- (2) 以下のいずれかの操作を行います。

- 目的のタスクの行内をダブルクリックします。
- 目的のタスクの行内を右クリックしてコンテキストメニューを表示させ、[ジャンプ]を選択します。

エディタはタスク設定位置にジャンプします。

●タスクの削除

タスクの表示が不要となった場合は、エディタ上または[タスク]ビュー上で削除できます。

エディタ上での削除

削除するタスクマーカーを右クリックしてコンテキストメニューを表示させ、[タスクの除去]を選択します。

[タスク]ビュー上での削除

- (1) 削除するタスクをクリックして選択します。
- (2) コンテキストメニューを表示させ、[削除]を選択します。

完了したタスクの削除

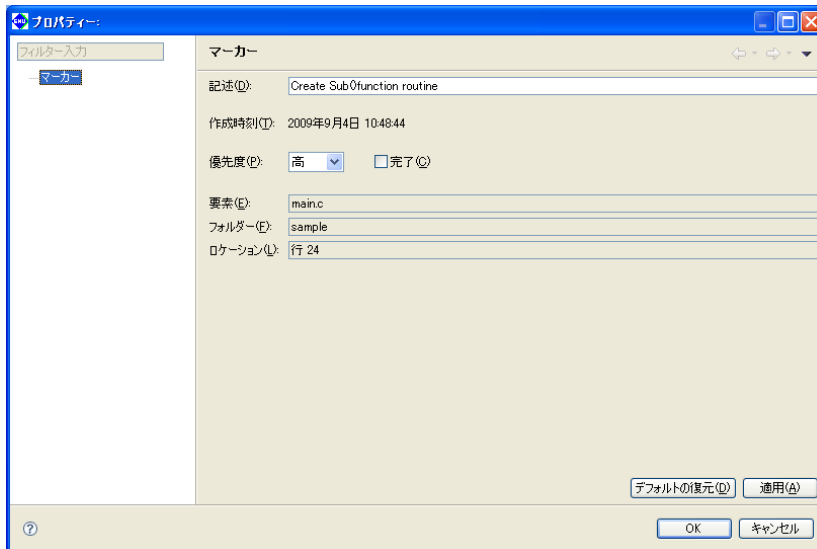
完了したタスクのみを一度に削除することができます。

- (1) [タスク]ビューのコンテキストメニューから[完了タスクの削除]を選択します。
- (2) 確認のダイアログが表示されますので、削除する場合は[OK]、中止する場合は[キャンセル]をクリックします。

完了したタスクを削除はせずに、フィルタ(後述)によって表示しないように設定することもできます。

●タスク情報の表示

目的のタスクの行内を右クリックしてコンテキストメニューを表示させ、[プロパティ]を選択します。[プロパティ]ダイアログが現れ、ビュー内の情報に加え作成日時を表示します。



ここでタスク説明の変更も行えます。

●タスクリストのフィルタと並べ替え

タスクが増え[タスク]ビューの一覧が大きくなりすぎたのために、一覧の表示を制限したり、項目で並べ替えたりすることができます。

フィルタ

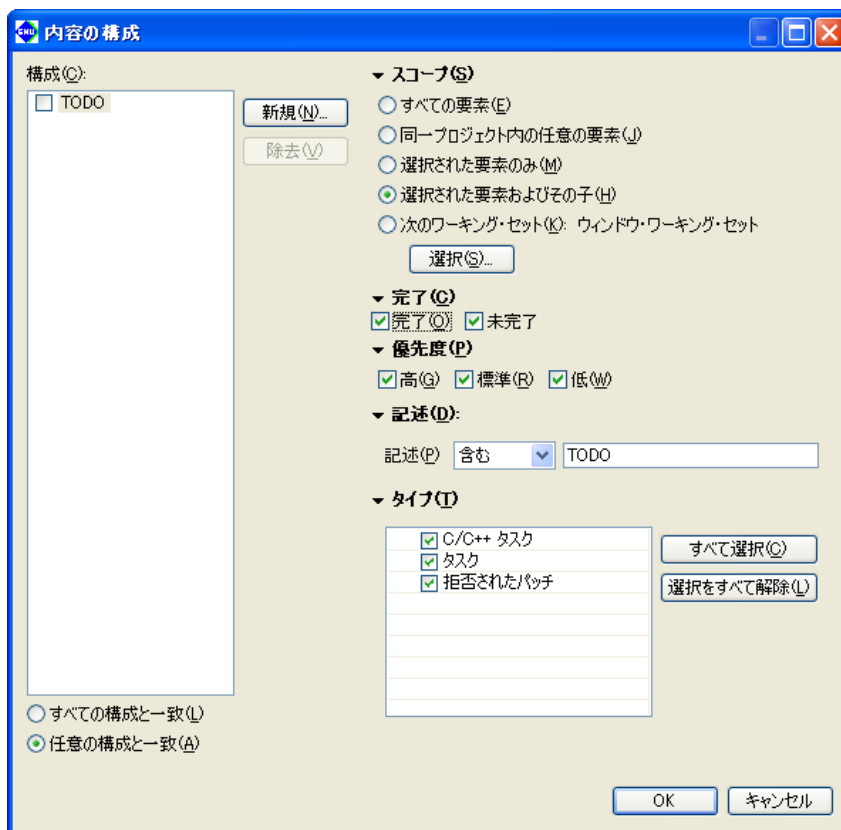
フィルタを使用し、必要なタスク以外を非表示にできます。

また、複数のフィルタを作成し、必要に応じて使い分けることができます。

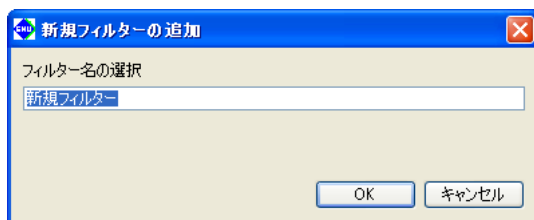
新規のフィルタは以下の手順で作成します。

- (1) [タスク]ビューをアクティブにします。

- (2) ビューメニュー(▽)から[内容の構成...]を選択します。
[内容の構成]ダイアログが表示されます。



- (3) [新規]ボタンをクリックします。



作成するフィルタの名前を入力し、[OK]をクリックします。

- (4) [構成:]内に他のフィルタが表示されている場合は、そのフィルタのチェックを外し、新たに作成したフィルタのチェックボックスをOnにします。
- (5) 表示させる条件を設定します。

[すべての要素]

[同一プロジェクト内の任意の要素]

[選択された要素のみ]

[選択された要素およびその子]

開かれているプロジェクトすべてを対象に、設定されているタスクを表示します。

現在選択されているプロジェクト内に設定されているタスクを表示します。

[C/C++ プロジェクト]/[ナビゲーター]ビューで選択したファイル、またはエディタでアクティブにしたファイル内に設定されているタスクのみを表示します。

[C/C++ プロジェクト]/[ナビゲーター]ビューで選択したプロジェクトまたはフォルダ内の全ファイル、もしくは選択したファイル内に設定されているタスクのみを表示します。

[次のワーキング・セット:]

[完了] - [完了] / [未完了]
[優先度:] - [高] / [標準] / [低]

[記述] - [含む]

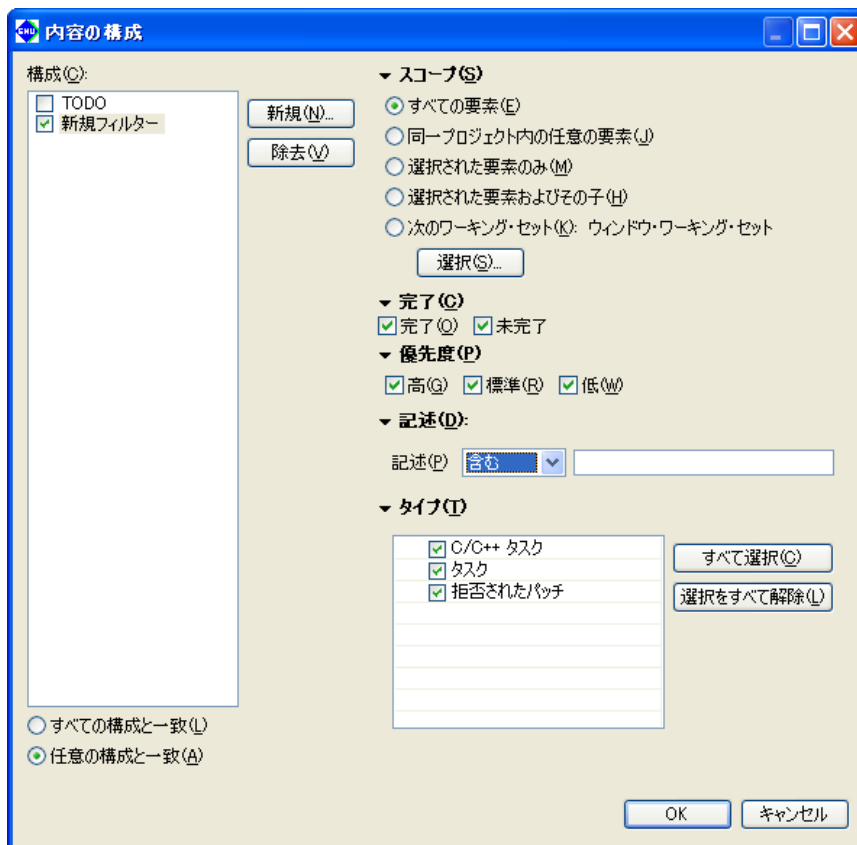
[記述] - [含まない]

指定のワーキングセット内に設定されているタスクを表示します。ワーキングセットは[選択...]ボタンで選択します。

完了/未完了を表示させる場合に選択します。指定優先順位(高、標準、低)のタスクを表示させる場合に選択します。

上記の範囲指定に加え、[記述]項目がテキストボックスに入力した文字列を含むタスクのみを表示します。この条件を使用しない場合はテキストボックスを空にしておきます。

上記の範囲指定に加え、[記述]項目がテキストボックスに入力した文字列を含まないタスクのみを表示します。この条件を使用しない場合はテキストボックスを空にしておきます。



(6) [OK]をクリックします。

[タスク]ビューは設定した条件に合ったタスクのみを表示します。

現在のフィルタの条件を変更する場合は、(3)と(4)を省略して条件選択のみを行ってください。

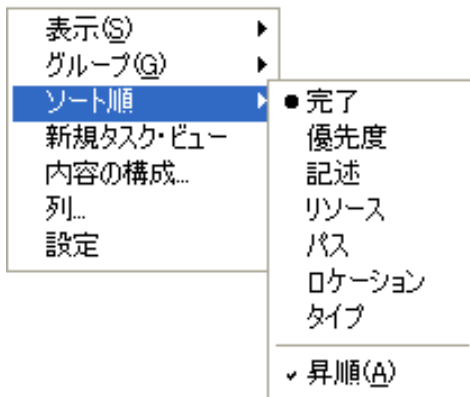
複数のフィルタを作成した場合は、上記ダイアログを表示させ、[構成:]で使用するフィルタを選択します。あるいは、ビューメニューの[表示]のサブメニューから使用するフィルタを選択します。

[内容の構成]ダイアログの詳細については5.10.7節を参照してください。

並べ替え

一覧の項目に優先順位を付け、表示を並べ替えることができます。

- (1) [タスク]ビューをアクティブにします。
- (2) ビューメニュー (▽) から [ソート順] を選択し、一覧のどの項目を優先して並べ替えるかを選択します。また、[昇順] をチェックすると、それぞれの項目を昇順で並べ替え、チェックを外すと降順で並べ替えます。

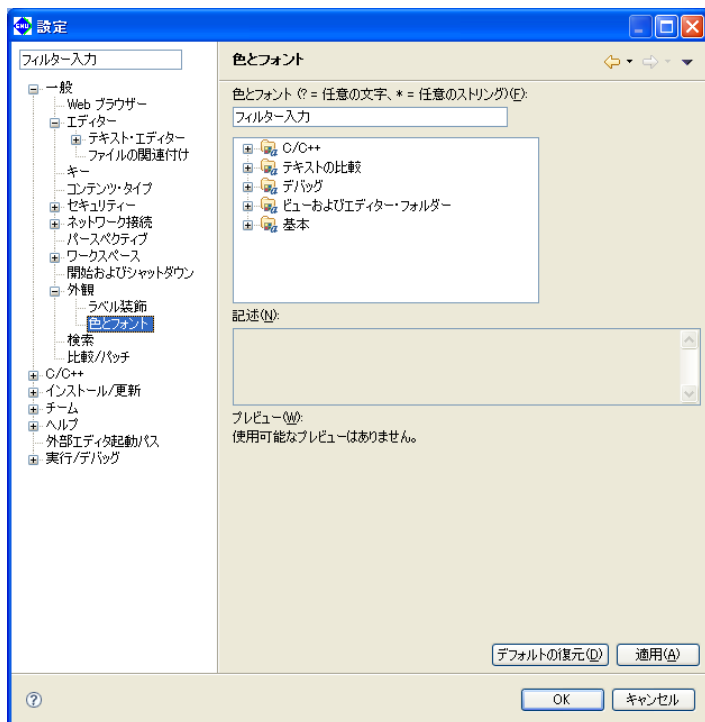


5.5.8 エディタのカスタマイズ

エディタが使用するフォントやタブサイズなどは[設定]ダイアログで変更できるようになっています。[設定]ダイアログは[ウィンドウ]メニューから[設定...]を選択すると表示されます。

[設定]ダイアログのエディタに関する主なページとカスタマイズ項目を以下に示します。[設定]ダイアログの詳細は"5.9 IDEのカスタマイズ(設定)"を参照してください。

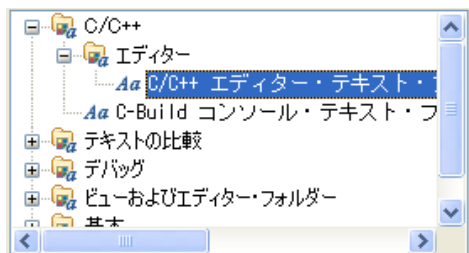
●テキストのフォントと色([一般]>[外観]>[色とフォント])



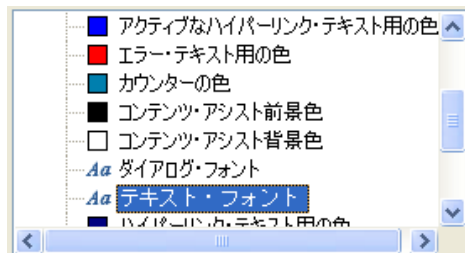
ここで、C/C++ エディタとアセンブラエディタのテキストフォントを変更することができます。

- (1) C/C++ エディタのフォントを変更するには、ツリーリストから[C/C++]>[エディター]>[C/C++ エディター・テキスト・フォント]を選択します。
アセンブリエディタのフォントを変更するには、ツリーリストから[基本]>[テキスト・フォント]を選択します。

C/C++ エディタフォント



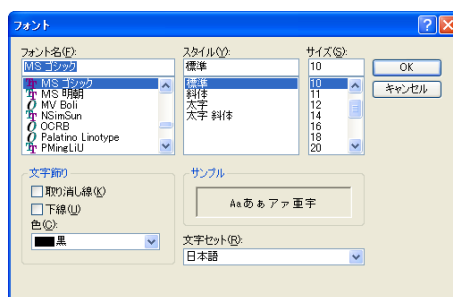
アセンブリエディタフォント



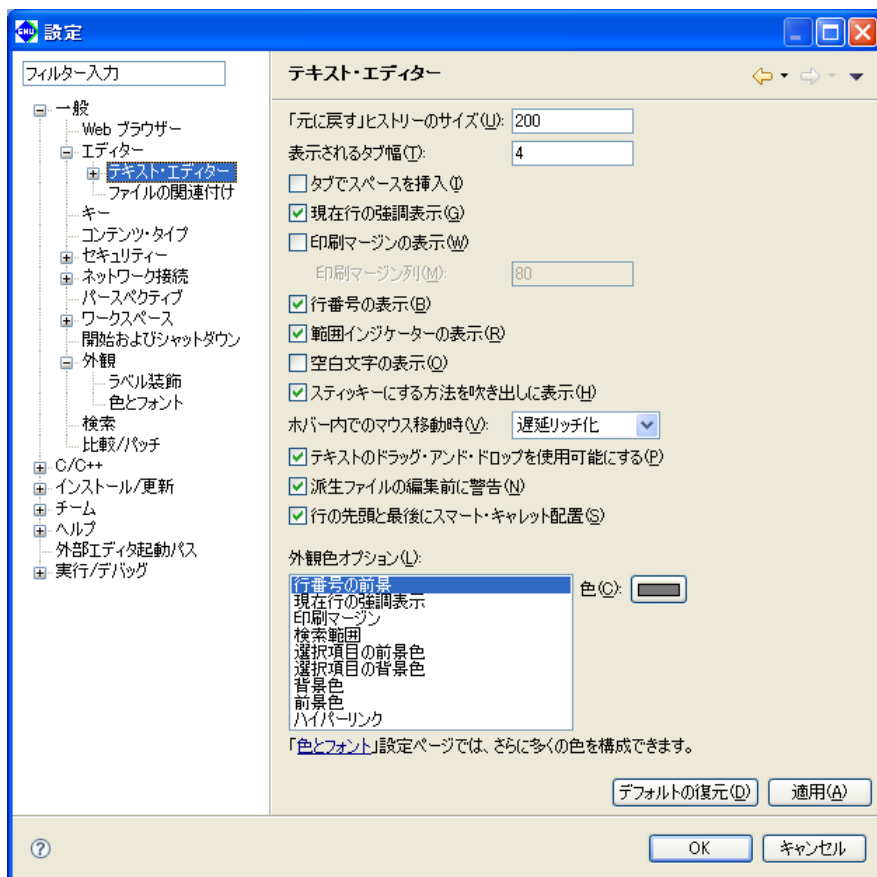
- (2) [変更...]をクリックしてフォント選択ダイアログを表示させ、フォントとスタイルおよび表示色を選択します。

[システム・フォントの使用]ボタンでWindowsの標準フォントを選択することもできます。

- (3) [適用]または[OK]をクリックして設定を終了します。



●エディタのタブサイズ、行番号表示([一般]>[エディター]>[テキスト・エディター])



タブサイズを変更するには、**[表示されるタブ幅:]**に文字数を設定します。

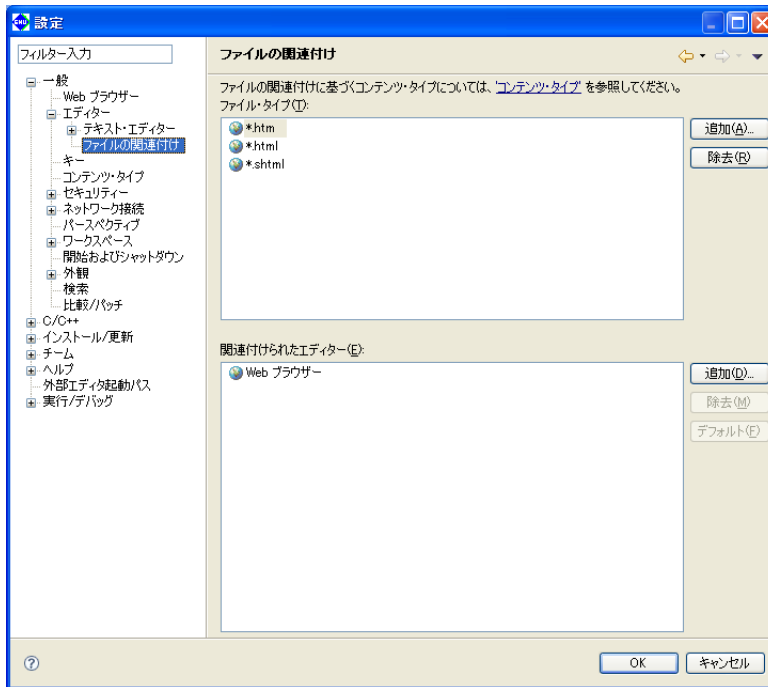
行番号を表示させるには、「行番号の表示」を選択します。

このページでは、このほかに強調表示の設定などが行えます。

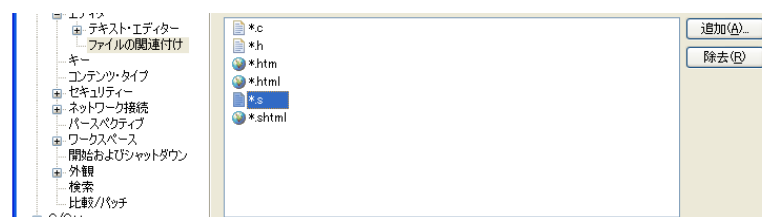
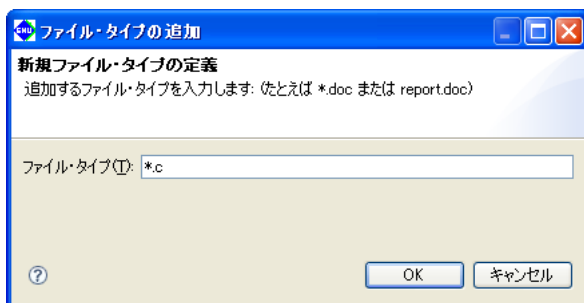
5.5.9 外部エディタを使用するには

日頃使用しているエディタをIDEに登録して、リソースの編集に使用することができます。
外部エディタに登録する手順は次のとおりです。

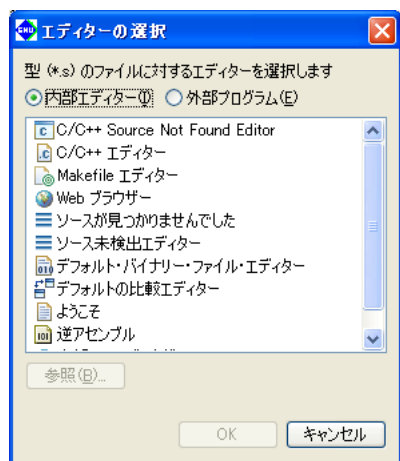
- (1) [ウィンドウ]メニューから[設定...]を選択します。
[設定]ダイアログが表示されます。
- (2) 左側にある設定項目のツリー表示から[一般]>[エディター]>[ファイルの関連付け]を選択します。



- (3) 登録するエディタで編集するファイルの種類(拡張子)を[ファイル・タイプ:]から1つ選択します。
表示されていない場合は[ファイル・タイプ:]の右にある[追加...]ボタンで次のダイアログを表示させ、
拡張子(*.c、*.cc、*.cpp、*.cxx、*.h、*.hh、*.hpp、*.hxx、*.s)を入力して一覧に加えてください。



- (4) [関連付けられたエディター:]の[追加...]ボタンをクリックします。
[エディターの選択]ダイアログが表示されます。



- (5) [外部プログラム]ラジオボタンを選択します。
(6) 一覧から、もし表示されない場合は[参照...]ボタンで表示されるファイル選択ダイアログで登録するエディタを選択します。
(7) [OK]ボタンをクリックし、[エディターの選択]ダイアログを閉じます。
(8) [OK]ボタンをクリックし、[設定]ダイアログを閉じます。

これで、選択した拡張子と外部エディタとの関連づけができました。編集するファイルの種類すべてについて上記の作業を行います。

ファイルを、登録したエディタで開く方法は次のとおりです。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューでファイルを選択します。
- (2) ファイルを右クリックしてコンテキストメニューを表示させ、[アプリケーションから開く]から登録したエディタを選択します。

外部エディタが起動して、選択したファイルを開きます。

注: 現在IDEのエディタで開かれているファイルを外部エディタで開くことはできません。開いているファイルを一度閉じてから、再度上記の操作を行ってください。

5.5.10 外部エディタを行番号指定で起動するには

●外部エディタ設定画面

IDEでプロジェクトをビルドしてコンパイルエラーがあるとき、IDEの[問題]ビューにエラーが発生しているファイルとその行番号が一覧表示されます。

このエラー一覧から、外部エディタを行番号指定で起動することができます。

起動する外部エディタを登録する手順は次のとおりです。

- (1) [ウィンドウ]メニューから[設定]を選択します。
[設定]ダイアログが表示されます。
- (2) 左側にある設定項目のツリー表示から[外部エディタ起動パス]を選択します。

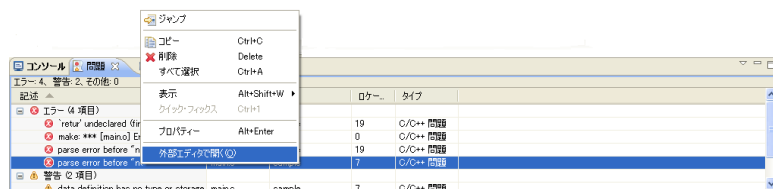


- (3) [参照...]ボタンをクリックして、起動したいエディタを選択します。
パス名、ファイル名は、最大255文字まで入力可能です。また、初期状態では“notepad.exe”(メモ帳)が設定されています。
 - (4) [パラメータ:]に、エディタ起動時のパラメータを指定します。
以下のパラメータが使用できます。
 %F： 起動時にファイル名に置換されます(省略不可)。
 %L： 起動時に行番号に置換されます(省略可)。
 - (5) [OK]ボタンをクリックし、[設定]ダイアログを閉じます。
- これで、選択したエラーと外部エディタとの関連づけができました。

●外部エディタ起動方法

IDEでプロジェクトをビルドしてコンパイルエラーがあるとき、IDEの[問題]ビューにエラーが発生しているファイルとその行番号が一覧表示されます。このエラー一覧から、外部エディタを行番号指定で起動することができます。

[問題]ビューでエラーを右クリックし、コンテキストメニューから[外部エディタで開く]を選択すると、[設定]ダイアログの[外部エディタ起動パス]で設定した外部エディタが起動します。



外部エディタの設定が間違っているときなど、起動できない場合には“外部エディタを開けませんでした”エラーが表示されます。

5.6 検索

IDEには、エディタの文書内検索機能の他に、ワークスペースやプロジェクト全体を対象としたテキスト検索機能とリソースやC/C+++エレメントの条件設定が可能な検索機能が搭載されています。検索結果は[検索]ビューに表示されます。ここでは、その操作方法について説明します。

5.6.1 テキスト検索

現在エディタ上で選択している文字列を、検索範囲をファイル外まで拡張して検索することができます。この操作手順は次のとおりです。

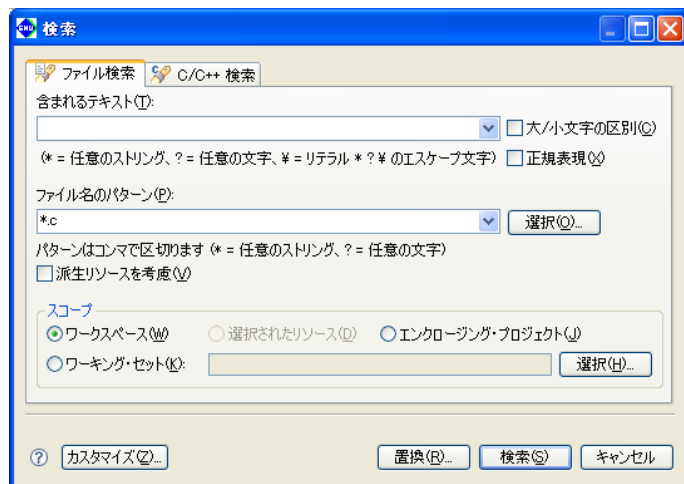
- (1) 検索する文字列をエディタ上でドラッグして選択します。
- (2) [検索]メニューの[テキスト]を選択し、そのサブメニューから検索範囲(ワークスペース、現在のプロジェクト、現在のファイル、指定のワーキングセット)を選択します。

検索結果は[検索]ビューに表示されます。

5.6.2 File検索

ワークスペース内、現在のプロジェクト内、または指定のワーキングセット内のファイル、もしくは現在のファイルに含まれる文字データを検索することができます。ファイル検索を行うには、以下のいずれかの方法で[検索]ダイアログの[ファイル検索]ページを表示させます。

- [検索]メニューから[ファイル...]を選択
- [検索]メニューから[検索...]を選択し、[検索]ダイアログの[ファイル検索]タブを選択
- ウィンドウツールバーの[検索]ボタンをクリックし、[検索]ダイアログの[ファイル検索]タブを選択



下記の説明に従って検索内容を設定し、[検索]ボタンをクリックします。検索が終わると[検索]ビューが開き、結果を表示します。

[含まれるテキスト:]

検索する文字データを入力します。ファイルのみを検索する場合は、空白にします。以前にこのページで検索したデータは▼のプルダウンリストから選択して入力することもできます。検索する文字データには、以下のワイルドカードを使用可能です。

*: 任意の文字列

?: 任意の1文字

¥: *, ?, ¥を検索する文字として指定する場合に前置きします。(¥*, ¥?, ¥¥)

[大/小文字の区別]

入力した文字データの太文字/小文字を区別して検索する場合に選択します。

[正規表現]

このチェックボックスを選択すると、正規表現での検索モードになります。
このモードでは正規表現の入力アシスト機能が使用できます。その方法は次のとおりです。

- (1) [正規表現]を選択します。
- (2) [含まれるテキスト:]テキストフィールドにカーソルを置きます。
テキストフィールドの前に"**I**"が表示され、入力アシスト機能が使用可能になります。
- (3) [Ctrl]+[Space]キーを押します。
- (4) 表示されたプルダウンリストから入力する構文を選択します。

[ファイル名のパターン:]

検索するファイルの種類、または名称パターンを入力します。複数のパターンをカンマ(,)で区切って入力可能です。複数指定はOR条件となります。

ファイルの種類は、[選択...]ボタンで表示されるダイアログ上で選択可能です。

以前にこのページで設定したパターンは▼のプルダウンリストから選択して入力することもできます。

検索する文字データには、以下のワイルドカードを使用可能です。

*: 任意の文字列

?: 任意の1文字

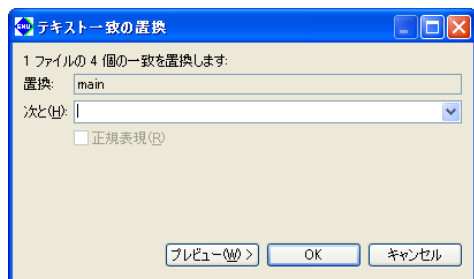
[スコープ]

検索対象を以下のラジオボタンで選択した範囲に絞ります。

[ワークスペース]	ワークスペース全体
[選択されたリソース]	[C/C++ プロジェクト]または[ナビゲーター]ビューで選択されているリソース
[エンクロージング・プロジェクト]	[C/C++ プロジェクト]または[ナビゲーター]ビューで選択されているリソースを含むプロジェクト
[ワーキング・セット:]	ワーキングセット内のリソース。ワーキングセットは[選択...]ボタンで表示されるダイアログ上で選択します。

[カスタマイズ...]

[検索]ダイアログ上に表示する検索ページ([ファイル検索]、[C/C++ 検索])を選択します。

**[置換...]**

上記の設定条件で検索を行い、最初の検索位置で候補を選択した状態で停止します。同時に、[テキスト一致の置換]ダイアログを表示します。ここで、検索文字列の置き換えが行えます。

[次と]

置き換える文字列を入力します。

[プレビュー>]

押下すると[テキスト一致の置換]ダイアログが開き、一致箇所がまとめて一覧表示されます。置換する箇所を確認してください。

[OK]

現在選択されている候補を入力した文字列と置き換え、次の候補を検索します。

[キャンセル]

検索を中止します。

[検索]

上記の設定条件で検索を行います。

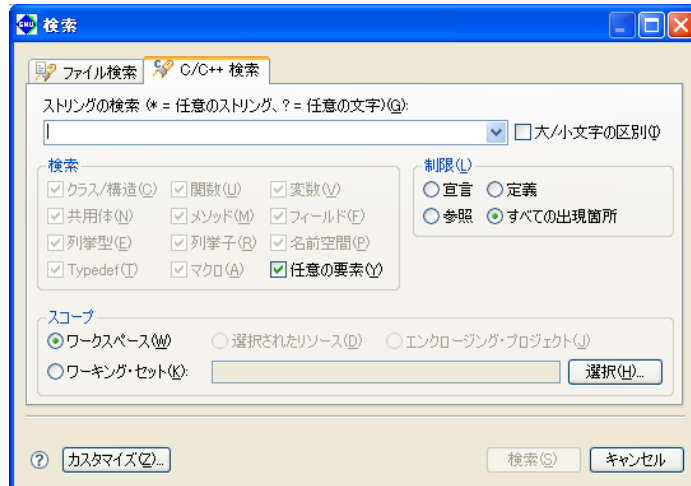
[キャンセル]

検索を中止します。

5.6.3 C/C++検索

ワークスペース内のリソースに含まれる文字列、および関数名などエレメントを検索できます。
C/C++検索を行うには、以下のいずれかの方法で[検索]ダイアログの[C/C++ 検索]ページを表示させます。

- [検索]メニューから[C/C++...]を選択
- [検索]メニューから[検索...]を選択
- ウィンドウツールバーの[検索]ボタンをクリック



下記の説明に従って検索内容を設定し、[検索]ボタンをクリックします。検索が終わると[検索]ビューが開き、結果を表示します。

[ストリングの検索:]

検索する文字列を入力します。エディタ上で文字列を選択した後に[検索]ダイアログを開くと、その文字列がこのフィールドに入力されます。

以前にこのページで検索したデータは▼のプルダウンリストから選択して入力することもできます。

検索する文字データには、以下のワイルドカードを使用可能です。

*: 任意の文字列

?: 任意の1文字

[大/小文字の区別]

入力した文字列の大文字/小文字を区別して検索する場合に選択します。

[検索]

検索対象のエレメントをチェックボックスで選択します。

[クラス/構造]	クラスまたは構造体
[関数]	グローバル関数または名前空間内の関数(クラス、構造体、または共用体のメンバ関数を除く)
[変数]	変数(クラス、構造体、または共用体のメンバを除く)
[共用体]	共用体
[メソッド]	メソッド(クラス、構造体、または共用体のメンバ)
[フィールド]	フィールド(クラス、構造体、または共用体のメンバ)
[列挙型]	列挙型
[列挙子]	列挙子
[名前空間]	名前空間
[Typedef]	型定義
[マクロ]	マクロ定義
[任意の要素]	すべてのエレメントを検索対象にします。このチェックボックスを選択すると、他のエレメントのチェックボックスは無効となります。

[制限]

検索対象を以下の選択に従って制限します。

[宣言]	宣言箇所
[定義]	定義箇所(関数、メソッド、変数、フィールド)
[参照]	参照箇所
[すべての出現箇所]	上記をすべて含む

[スコープ]

検索対象を以下のラジオボタンで選択した範囲に絞ります。

[ワークスペース]	ワークスペース全体
[選択されたリソース]	[C/C++ プロジェクト]または[ナビゲーター]ビューで選択されているリソース
[エンクロージング・プロジェクト]	[C/C++ プロジェクト]または[ナビゲーター]ビューで選択されているリソースを含むプロジェクト
[ワーキング・セット:]	ワーキングセット内のリソース。ワーキングセットは[選択...]ボタンで表示されるダイアログ上で選択します。

[カスタマイズ...]

[検索]ダイアログ上に表示する検索ページ([ファイル検索]、[C/C++ 検索])を選択します。

[検索]

上記の設定条件で検索を行います。

[キャンセル]

検索を中止します。

5.6.4 コンテキストメニューによるC/C++検索

エディタまたは[アウトライン]ビューのコンテキストメニューからでも、選択したエレメントの宣言箇所、または参照箇所の検索が行えます。

- (1) エディタ上のソースまたは[アウトライン]ビューから変数や関数などのエレメントを選択し、右クリックでコンテキストメニューを表示させます。
- (2) 宣言箇所を検索するには、[宣言]のサブメニューで検索範囲(ワークスペース、現在のプロジェクト、ワーキングセット)を選択します。
- (3) 参照箇所を検索するには、[参照]のサブメニューで検索範囲(ワークスペース、現在のプロジェクト、ワーキングセット)を選択します。

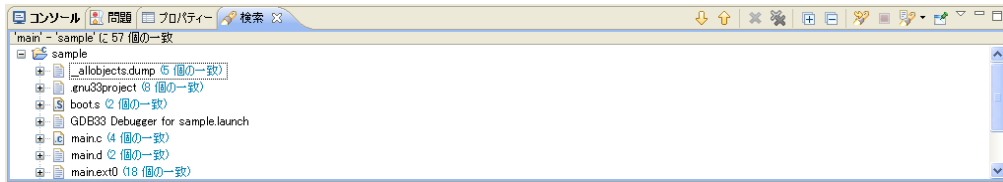
検索結果は[検索]ビューに表示されます。

5.6.5 検索の中止

検索を開始すると[検索]ビューが表示されます。検索中は[検索]ビューツールバーの[現行検索のキャンセル]ボタンがアクティブとなり、このボタンのクリックにより検索動作を中止できます。

5.6.6 検索結果

File検索、C/C++検索、テキスト検索の結果は[検索]ビューに一覧として表示されます。



●検索位置の参照

[検索]ビュー内の検索結果をクリックすることにより、エディタ内の記述位置にジャンプできます。対象ファイルが開いていない場合は検索結果をダブルクリックすると開きます。

また、[検索]ビューのツールバーボタンで検索結果を上下にナビゲートできます。

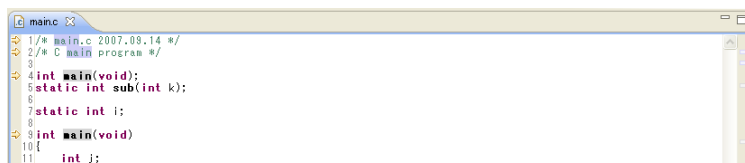
[次の一致を表示]

一覧内の一つ下の検索位置にジャンプします。([ナビゲート]メニューの[次の注釈]でも同様)

[直前の一致を表示]

一覧内の一つ上の検索位置にジャンプします。([ナビゲート]メニューの[前の注釈]でも同様)

エディタ上では検索された文字列が強調表示され、その行のマーカバーに矢印のマーカが表示されます。

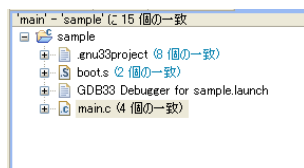


ファイル検索で文字列を空白としてファイル名を検索した場合、検索位置はファイルの先頭となります。ファイル名を検索して[ナビゲーター]ビュー上でファイルを参照したい場合は、[検索]ビュー内のファイル名を選択し、そのコンテキストメニューまたは[ナビゲート]メニューから[表示]>[ナビゲーター]を選択します。[ナビゲーター]ビュー内の該当ファイルが強調表示されます(一覧に表示されている場合のみ)。

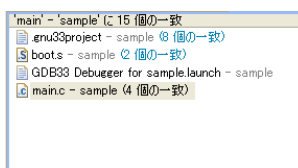
●[検索]ビューの表示形式の変更

[検索]ビューは検索結果をツリー表示するように初期設定されています。階層をなくして表示させるには、[検索]ビューのメニュー(▽)から[リストで表示]を選択してください。

ツリーで表示

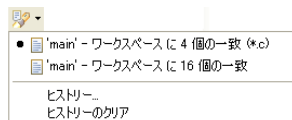


リストで表示



●検索履歴

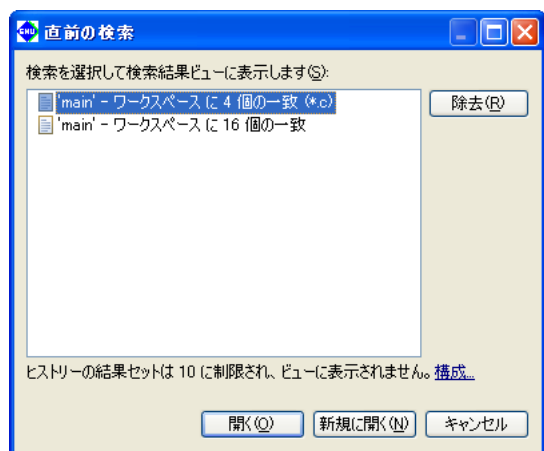
[検索]ビューツールバーの[直前の検索を表示]ショートカットは、これまでに実行したFile検索、C/C++検索、テキスト検索の一覧を表示します。



ここから、検索内容を選択して以前の検索結果をビューに再表示させることができます。

履歴の一覧は、[直前の検索を表示]ショートカットの[ヒストリークリア]を選択するとすべて削除されます。

[直前の検索を表示]ショートカットから[ヒストリー ...]を選択すると次のダイアログが表示され、表示させる履歴を選択できます。また、履歴を個別に削除することもできます。



[除去]

一覧から選択した検索履歴を削除します。

[開く]

一覧から選択した検索結果を現在の[検索]ビューに表示します。

[新規に開く]

一覧から選択した検索結果を新しい[検索]ビューを開いて表示します。

[キャンセル]

ダイアログを閉じます。

●検索結果の削除

[検索]ビューのツールバーボタンで検索結果を削除することができます。

-  [選択された一致を除去] ビュー内で選択されている検索結果を削除します。
-  [すべての一致を除去] ビュー内の検索結果の一覧をすべて削除します。

ここで削除した検索結果は、検索履歴を選択しても再表示されません。

5.7 プログラムのビルド

プログラムのビルドとは、必要なソースをコンパイル/アセンブルし、ライブラリ等も含めてリンクして実行形式のオブジェクトファイルを生成することをいいます。実際の動作は、コンパイラやリンカの実行手順を記述したmakeファイルを**make.exe**で実行させます。

ここでは、ビルドに必要なツールオプションやリンカスクリプトの設定と、ビルドの実行方法を説明します。

5.7.1 CPUタイプの設定

開発するアプリケーションシステムが搭載するS1C33プロセッサの種類により、ツールの起動コマンドオプションやリンクするエミュレーションライブラリを切り替えます。そのために、正しいターゲットCPUタイプを選択しておく必要があります。

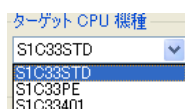
通常はプロジェクトの新規作成時にターゲットCPUタイプを選択しますのでその後の選択は不要ですが、次のように再設定することができます。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、CPUタイプを変更するプロジェクトを選択します。
- (2) [プロジェクト]メニューまたは上記ビューのコンテキストメニューから[プロパティ]を選択します。
[プロパティ]ダイアログが表示されます。
- (3) プロパティの一覧から[GNU33 一般設定]を選択します。



[ターゲット CPU 機種]に現在選択されているターゲットCPUの種類が表示されます(未選択時のデフォルト設定はS1C33STD)。

- (4) [ターゲット CPU 機種]のリストからターゲットCPUの種類を選択します。



S1C33STD: C33 STDコアを内蔵したデバイスをターゲット機種とする場合

S1C33PE: C33 PEコアを内蔵したデバイスをターゲット機種とする場合

S1C33401: S1C33401をターゲット機種とする場合

(5) 変更を確定する場合は[OK]ボタンを、変更を中止する場合は[キャンセル]ボタンをクリックします。

ボタンの機能は次のとおりです。

- [OK] 変更内容を確定してダイアログを終了します。
- [キャンセル] 変更内容を破棄してダイアログを終了します。
- [適用] 変更内容を確定しますが、ダイアログは閉じられません。他のプロパティを変更する場合はそのページに移動する前に[適用]ボタンをクリックしてください。
上記の設定を変更した場合は、"clean"ビルド(5.7.9節参照)を実行するかどうかを確認するダイアログが表示され、以前の設定で生成されているファイルの削除(およびリビルド)が行えます。
- [設定を戻す] 変更した内容をこのページを開いたときの状態([適用]ボタンがクリックされている場合は、その時点で確定した内容)に戻すことができます。

この選択に従って変更される設定を以下に示します。

表5.7.1.1 CPUタイプによるオプション設定

CPUタイプ	影響を受ける設定	効 果
S1C33STD	ビルドオプション設定	オプションから-mc33peおよび-mc33advが取り除かれます。 リンクするエミュレーションライブラリをC33 STDコア用に切り替えます。
	リンカスクリプト設定	リンクするエミュレーションライブラリをC33 STDコア用に切り替えます。
	パラメータマップ設定	パラメータファイル出力時にCPUタイプを出力しません。
	gdb起動コマンド設定	ひな型設定をC33 STDコアのデフォルト値に戻します。(ICD6+USB接続) 編集領域のコマンドは更新されません。
S1C33PE	ビルドオプション設定	オプションに-mc33peを追加します。 リンクするエミュレーションライブラリをC33 PEコア用に切り替えます。
	リンカスクリプト設定	リンクするエミュレーションライブラリをC33 PEコア用に切り替えます。
	パラメータマップ設定	パラメータファイル出力時にCPUのタイプ(C33 PE)を出力します。
	gdb起動コマンド設定	ひな型設定をC33 PEコアのデフォルト値に戻します。(ICD6+USB接続) 編集領域のコマンドは更新されません。
S1C33401	ビルドオプション設定	オプションに-mc33advおよび-mc33401を追加します。リンクするエミュレーションライブラリをS1C33401用に切り替えます。
	リンカスクリプト設定	リンクするエミュレーションライブラリをS1C33401用に切り替えます。
	パラメータマップ設定	パラメータファイル出力時に、CPUのタイプ(C33 拡張)を出力します。
	gdb起動コマンド設定	ひな型設定をS1C33401のデフォルト値に戻します。(ICD6+USB接続) 編集領域のコマンドは更新されません。

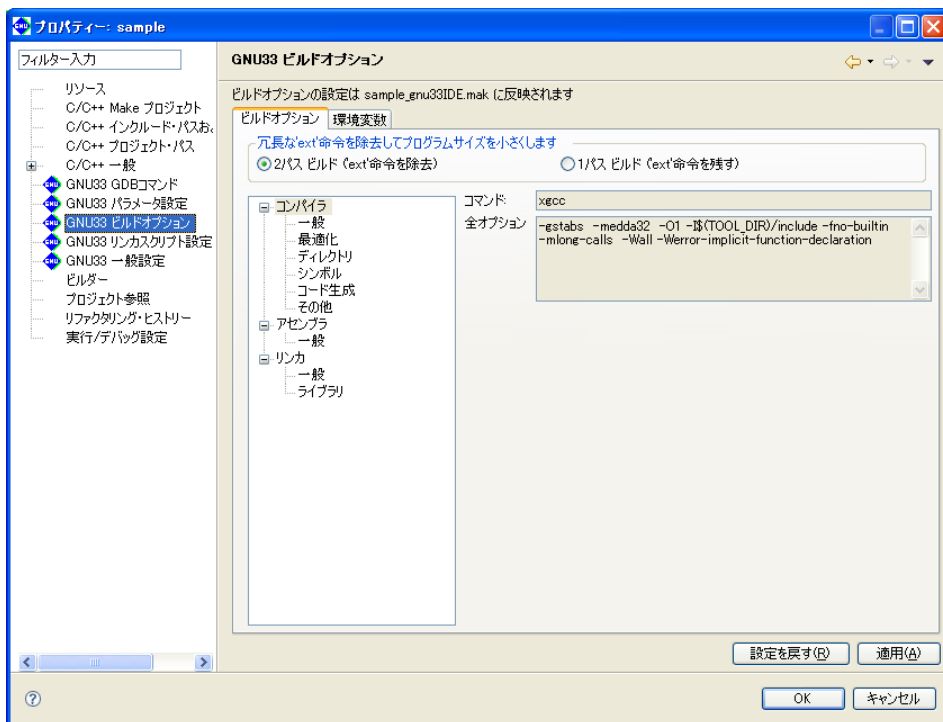
5.7.2 2パス/1パスメイクの選択

2パスメイクは、リンクにより確定したシンボル情報を元にアセンブラを再実行して冗長なext命令を削除するための手法です。たとえば、関数コールはコンパイル/アセンブルによりext命令付きのcall命令に展開されますが、分岐距離によっては不要なext命令が付加されます。これを削除することにより、プログラムサイズを小さく抑えることができます。この詳細については、"8.8 拡張命令の最適化"を参照してください。

IDEは通常の1パスメイクに加え、2パスメイク用のメイクファイルを生成することができます。

この選択は次のように行います。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ビルドするプロジェクトを選択します。
- (2) [プロジェクト]メニューまたは上記ビューのコンテキストメニューから[プロパティ]を選択します。
[プロパティ]ダイアログが表示されます。
- (3) プロパティの一覧から[GNU33 ビルドオプション]を選択します。



- (4) [冗長な'ext'命令を除去してプログラムサイズを小さくします]のラジオボタンでビルド方法を選択します。

[2パス ビルド](デフォルト : ON)

このボタンを選択すると、拡張命令を最適化するための2パスメイクファイルが生成されます。ビルド時間は若干増えますが、プログラムサイズは小さくなります。

[1パス ビルド]

このボタンを選択すると、ext命令を削除しない1パスメイクファイルが生成されます。

- (5) 他のプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。

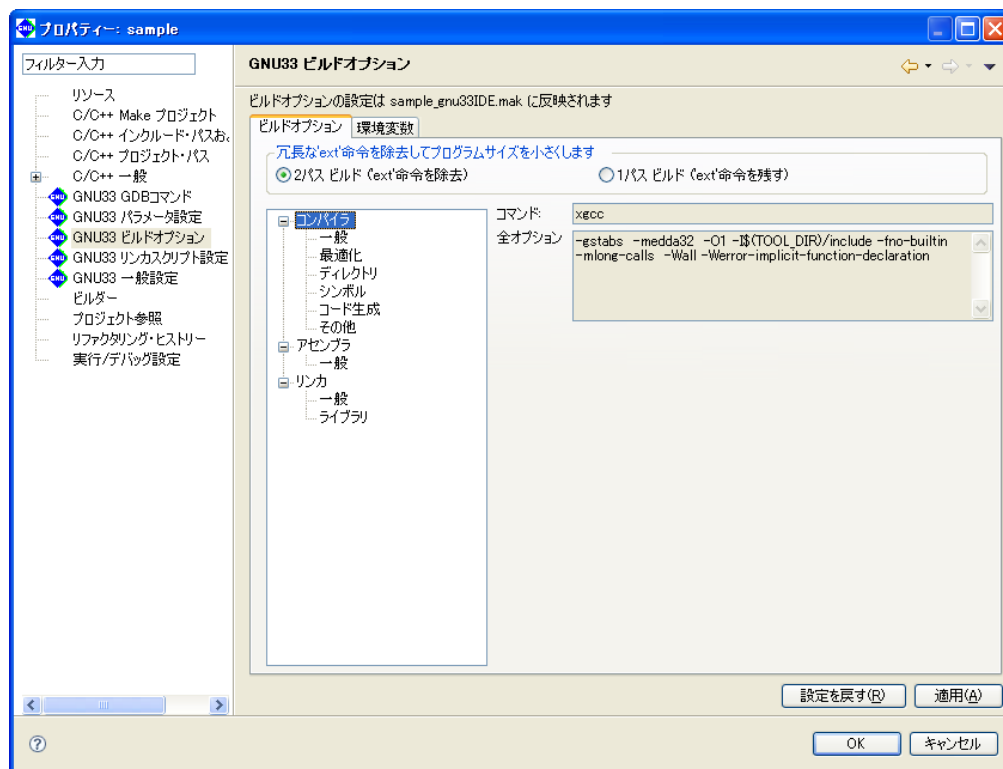
設定を変更した場合は、"clean"ビルド(5.7.9節参照)を実行するかどうかを確認するダイアログが表示され、以前の設定で生成されているファイルの削除(およびリビルド)が行えます。

[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

5.7.3 コンパイラオプションの設定

C/C++コンパイラのコマンドオプションは、次のように設定します。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ビルドするプロジェクトを選択します。
- (2) [プロジェクト]メニューまたは上記ビューのコンテキストメニューから[プロパティ]を選択します。
[プロパティ]ダイアログが表示されます。
- (3) プロパティの一覧から[GNU33 ビルドオプション]を選択します。
- (4) [ビルドオプション]のツリーから[コンパイラ]を選択します。

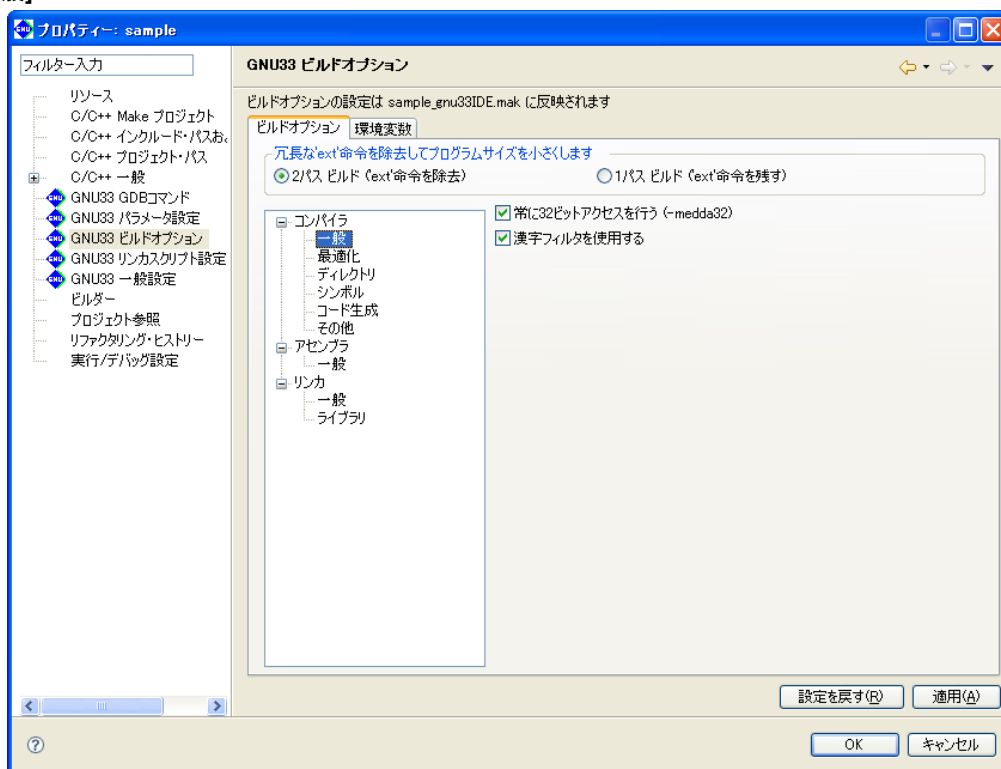


[コマンド:]はC/C++コンパイラのプログラム名、[全オプション]は現在設定されているコンパイラオプションを表示します。

- (5) [コンパイラ]のツリーリストからカテゴリを選択し、必要なオプションを設定します。
- (6) 他のプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。
設定を変更した場合は、"clean"ビルド(5.7.9節参照)を実行するかどうかを確認するダイアログが表示され、以前の設定で生成されているファイルの削除(およびリビルド)が行えます。
[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

コンパイラオプションのカテゴリ別設定ページを以下に示します。オプションの詳細は、C/C++コンパイラの章を参照してください。

[一般]



このページではコンパイラの基本オプションを選択します。

[常に32ビットアクセスを行う(-medda32)](デフォルト: ON)

このオプションを指定すると、デフォルトデータエリアを使用せずに、32ビットアクセス用のコードを出力します。

-fPICオプション([コード生成]ページ)との同時指定はできません。

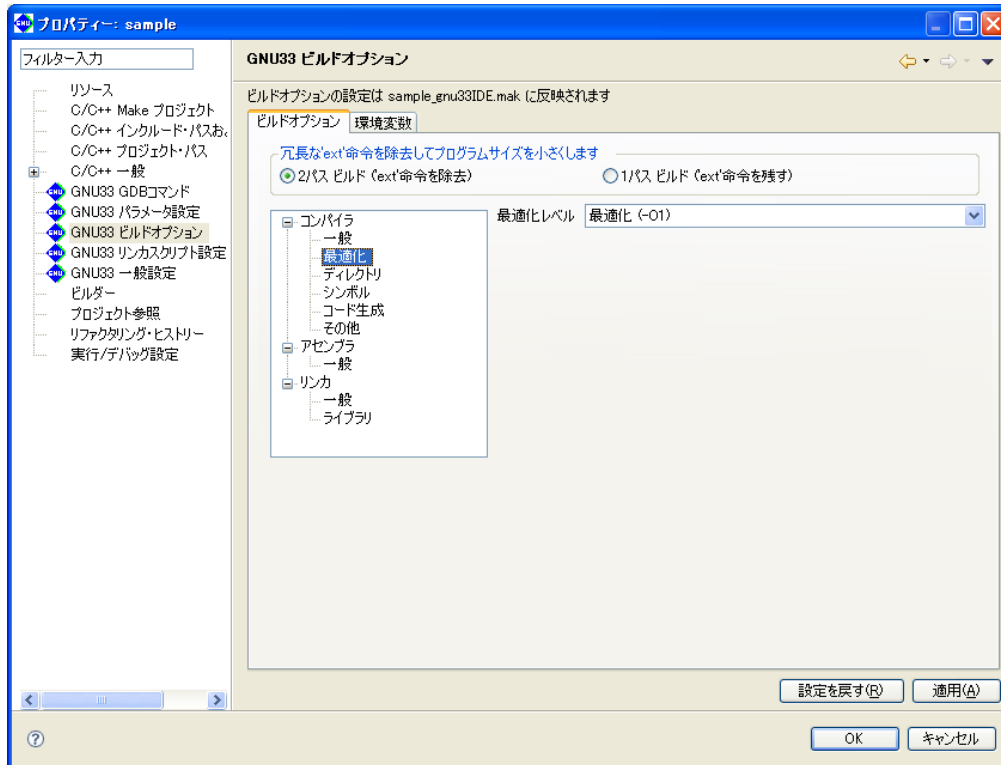
[漢字フィルタを使用する]

このオプションをONにした場合は、コンパイル時にソース内のシフトJISコードを適切に読み込みます。

OFFにした場合は、コンパイラ呼び出し時に-mno-sjis-filtオプションが付与され、上記の処理は行われません。

このチェックボックスのデフォルト設定状態は、IDEを起動したOSの言語により決まります。日本語版OS環境でIDEを起動した場合はON、その他の言語ではOFFに設定されます(OFF時は、コンパイル時に-mno-sjis-filtオプションが付与されます)。

[最適化]

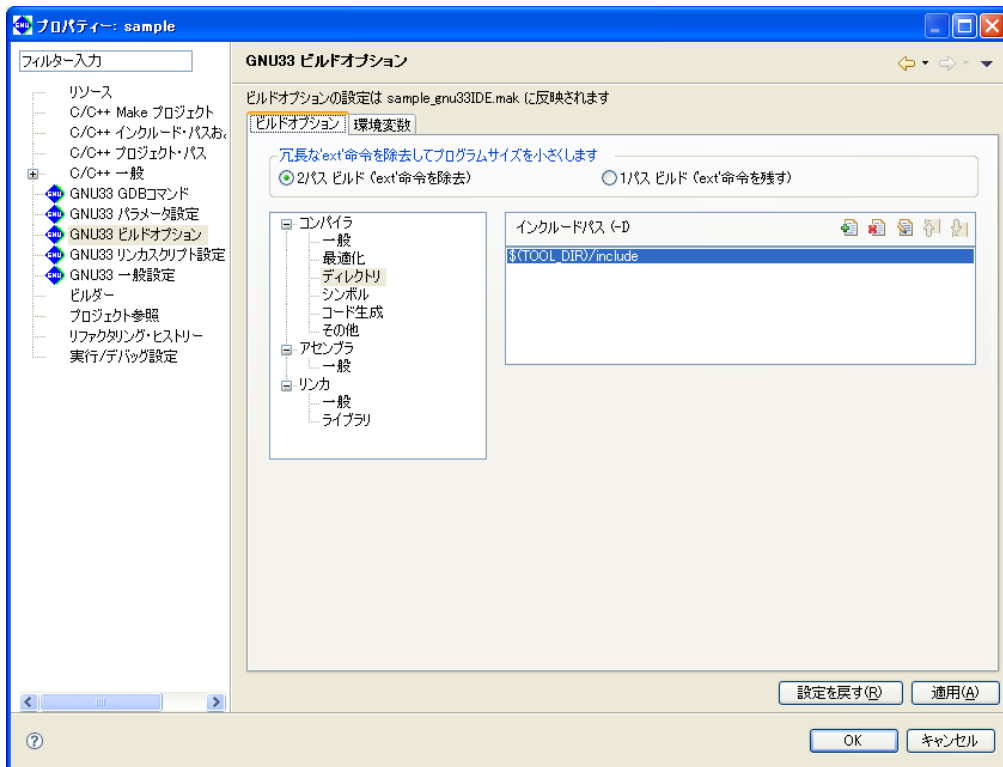


このページではコンパイラの最適化オプションを選択します。

[最適化レベル](デフォルト: -O1)

最適化のレベル(-O0~-O3、-Os)を選択します。

[ディレクトリ]



このページではコンパイラの検索パスオプションを設定します。

[インクルードパス (-I)](デフォルト: -I\$(TOOL_DIR)/include)

インクルードファイルの検索パスを設定します。ボタンの機能は以下のとおりです。

-  **[追加...]** ディレクトリを追加します。パスを入力あるいは[ファイル・システム...]ボタンで選択するダイアログが表示されます。
-  **[削除]** リスト内で選択したパスを削除します。
-  **[編集...]** リスト内で選択したパスを編集します。パスを書き換えるダイアログが表示されます。
-  **[上へ移動]** リスト内で選択したパスを一覧内の一つ上に移動します。インクルードファイルは一覧内の上にあるパスから順に検索されます。
-  **[下へ移動]** リスト内で選択したパスを一覧内の一つ下に移動します。

※\$(TOOL_DIR)について

デフォルト設定で、[インクルードパス (-I)]には"\$ (TOOL_DIR) /include"が表示されています。
\$(環境変数) は、ビルド時に生成されるmakeファイルに記述されるマクロです。TOOL_DIRは、gnu33ツールをインストールしたディレクトリへのパスが定義されている環境変数で、定義内容は[環境変数]タブのページで確認できます。

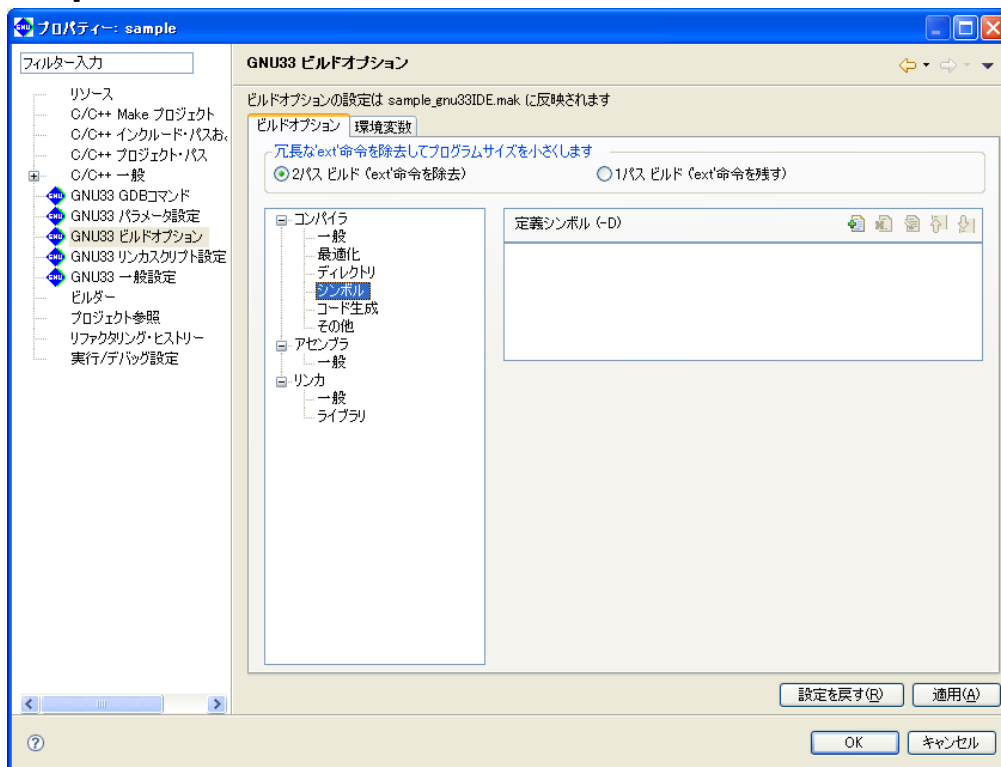
例: gnu33ツールをc:\EPSON\gnu33ディレクトリにインストールした場合

TOOL_DIR = c: / EPSON / gnu33

make.exeの実行時は、マクロが () 内の環境変数の内容に置き換えられますので、-I\$(TOOL_DIR) /includeは-Ic: / EPSON / gnu33 / includeの指定となります。

[環境変数]タブのページでは、TOOL_DIRと同様なユーザ独自の環境変数を定義することができます。ここで定義した環境変数は、ビルドオプションのインクルードパスおよびライブラリファイルのパスの指定に使用可能です。[環境変数]ページの詳細については、5.10.1節を参照してください。

[シンボル]



このページではコンパイラのマクロ定義オプションを設定します。

[定義シンボル (-D)](デフォルト: なし)

マクロ名と置き換え文字を指定します。ボタンの機能は以下のとおりです。

 [追加...] マクロ定義を追加します。マクロ定義を入力するダイアログが表示されますので、次の形式で入力します。

<マクロ名>

または

<マクロ名>=<置き換え文字>

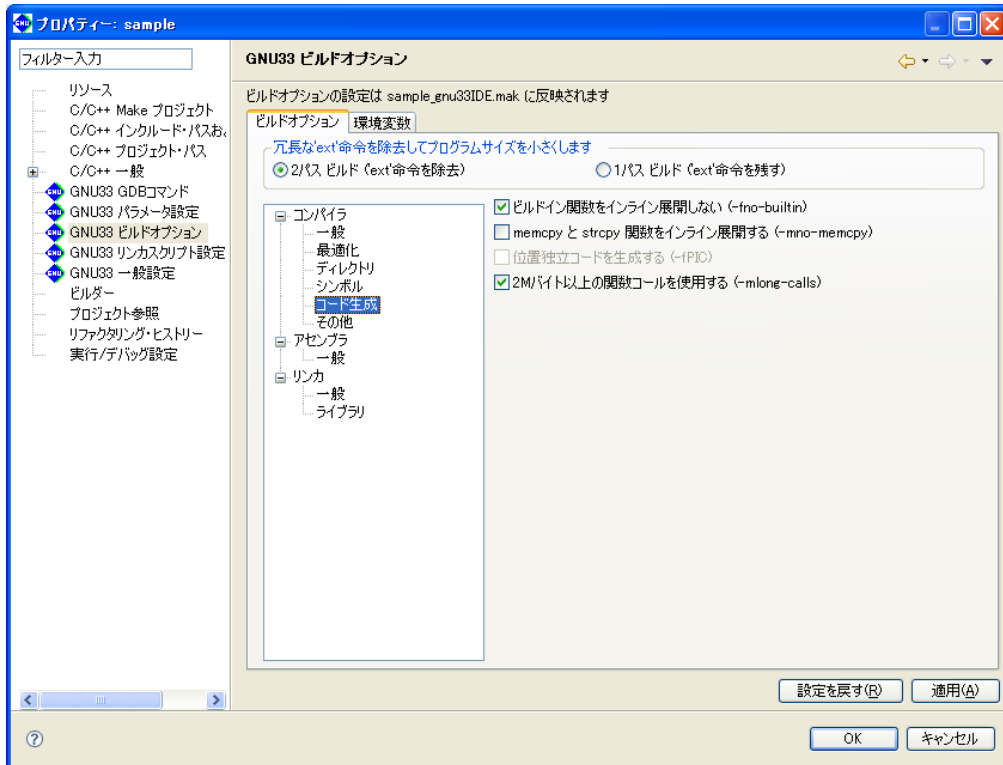
 [削除] リスト内で選択したマクロ定義を削除します。

 [編集...] リスト内で選択したマクロ定義を編集します。マクロ定義を書き換えるダイアログが表示されます。

 [上へ移動] リスト内で選択したマクロ定義を一覧内の一つ上に移動します。

 [下へ移動] リスト内で選択したマクロ定義を一覧内の一つ下に移動します。

[コード生成]



このページではコンパイラのコード生成に関するオプションを選択します。

[ビルドイン関数をインライン展開しない (-fno-builtin)](デフォルト: ON)

このオプションを指定すると、ビルトイン関数が無効となり、必ずコールされます。
対象の関数については、"6.3.2 コマンドラインオプション"を参照してください。

[memcpyとstrcpy関数をインライン展開する (-mno-memcpy)](デフォルト: OFF)

このオプションを指定すると、memcpy() およびstrcpy() 関数をインライン展開します。

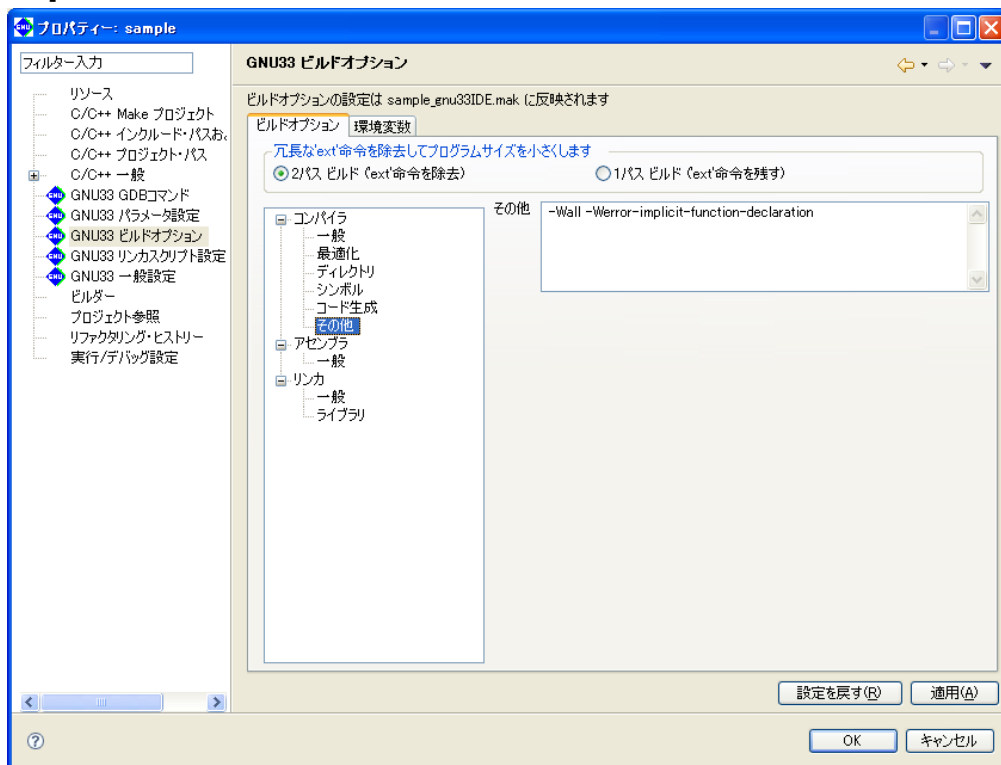
[位置独立コードを生成する (-fPIC)](デフォルト: OFF)

このオプションを指定すると、位置独立コードを生成します。
-medda32オプション([一般]ページ)との同時指定はできません。

[2Mバイト以上の関数コールを使用する (-mlong-calls)](デフォルト: ON)

2Mバイト以上離れた関数コールを行います。

[その他]



このページではコンパイラのその他のオプションを設定します。

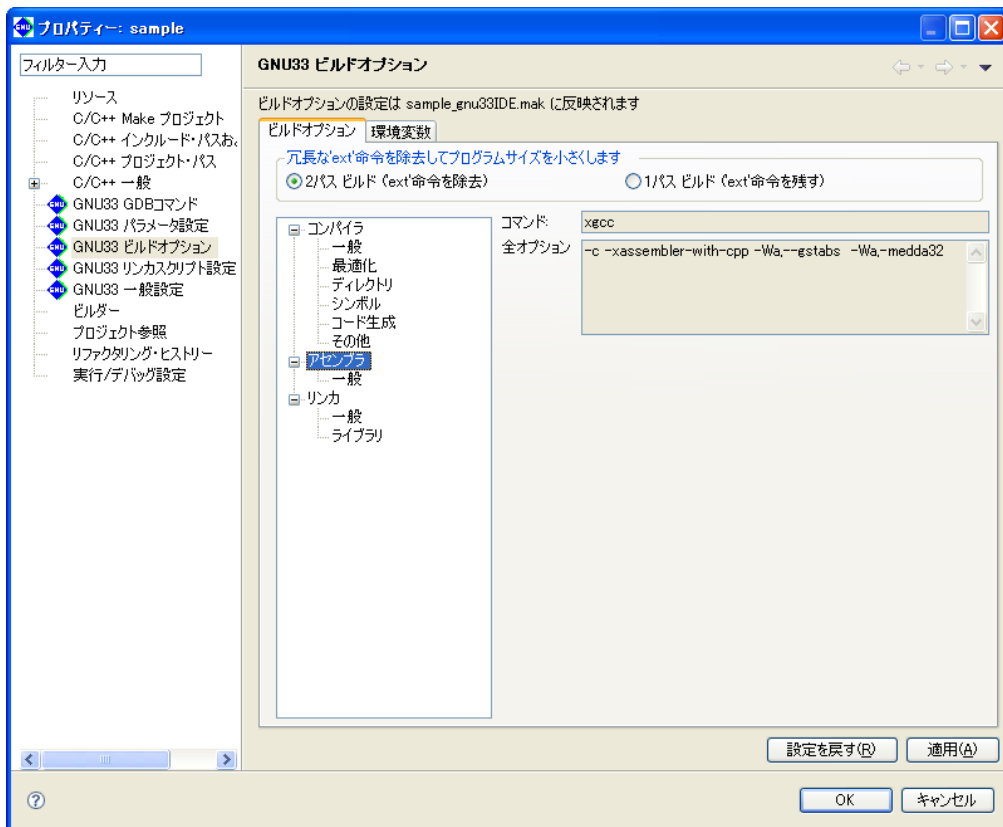
[その他](デフォルト: -Wall -Werror-implicit-function-declaration)

その他のオプションを直接入力します。オプション間には1個以上のスペースを挿入してください。
 2パスメイクを選択した場合、コンパイル時は自動的に-Sオプション指定でコンパイルしてアセンブリファイル(拡張子.ext0)を出力し、このファイルをアセンブラでアセンブルします。
 [その他]には、コンパイラの-Sオプション、-cオプションを指定する必要はありません。
 Cソースのアセンブルイメージは、<Cソースファイル名.ext0>ファイルで確認できます。
 また、-gstabsオプションも自動的に指定されます。

5.7.4 アセンブラオプションの設定

アセンブラのコマンドオプションは、次のように設定します。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ビルドするプロジェクトを選択します。
- (2) [プロジェクト]メニューまたは上記ビューのコンテキストメニューから[プロパティ]を選択します。
[プロパティ]ダイアログが表示されます。
- (3) プロパティの一覧から[GNU33 ビルドオプション]を選択します。
- (4) [ビルドオプション]のツリーから[アセンブラ]を選択します。



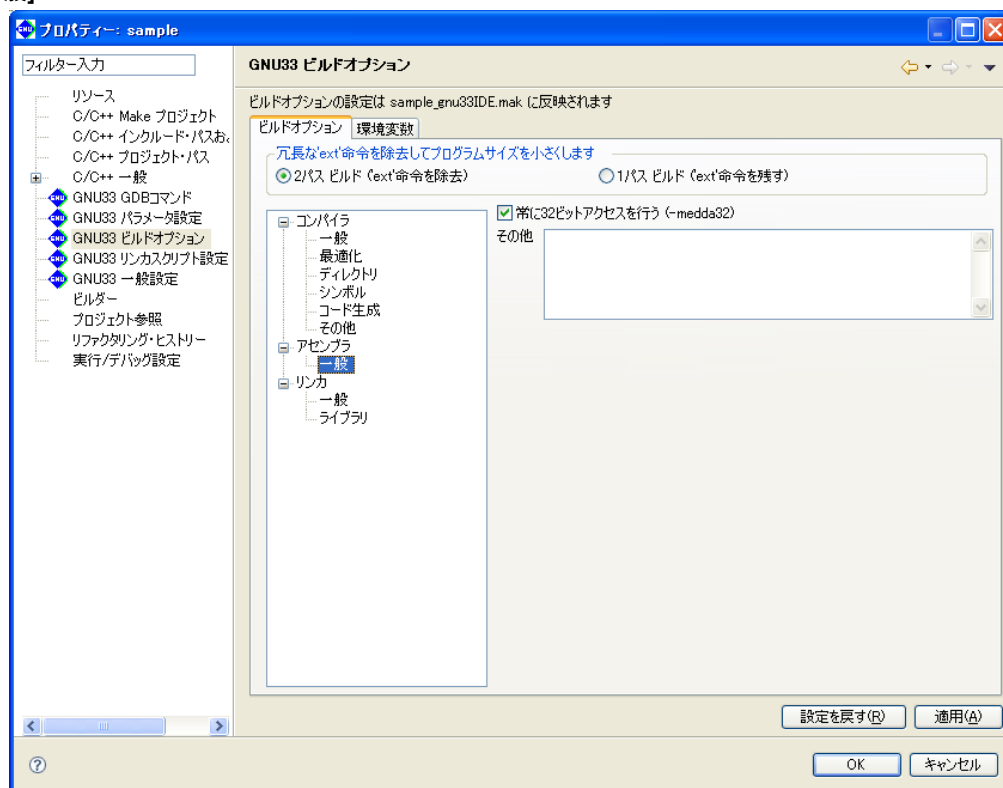
[コマンド:]はコンパイラ*のプログラム名、[全オプション]は現在設定されているオプションを表示します。

注：IDEではアセンブラソースを、`-c -xassembler-with-cpp`オプションを指定したCコンパイラでアセンブルします。全オプションには、コンパイラの`-c`オプション、`-xassembler-with-cpp`は指定する必要はありません。

- (5) [アセンブラ]のツリーリストから[一般]を選択し、必要なオプションを設定します。
- (6) 他のプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。
設定を変更した場合は、"clean"ビルド(5.7.9節参照)を実行するかどうかを確認するダイアログが表示され、以前の設定で生成されているファイルの削除(およびリビルド)が行えます。
[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

アセンブラオプションの設定ページを以下に示します。オプションの詳細は、アセンブラの章を参照してください。

[一般]



このページではアセンブルに関するオプションを選択します。-c、-xassembler-with-cpp、-Wa、--gstabsオプションは以下の選択にかかわらず、常に付加されます。

[常に32ビットアクセスを行う(-medda32)](デフォルト: ON)

このオプションを指定すると、デフォルトデータエリアを使用せずに、32ビットアクセス用のコードを出力します。

[その他](デフォルト: なし)

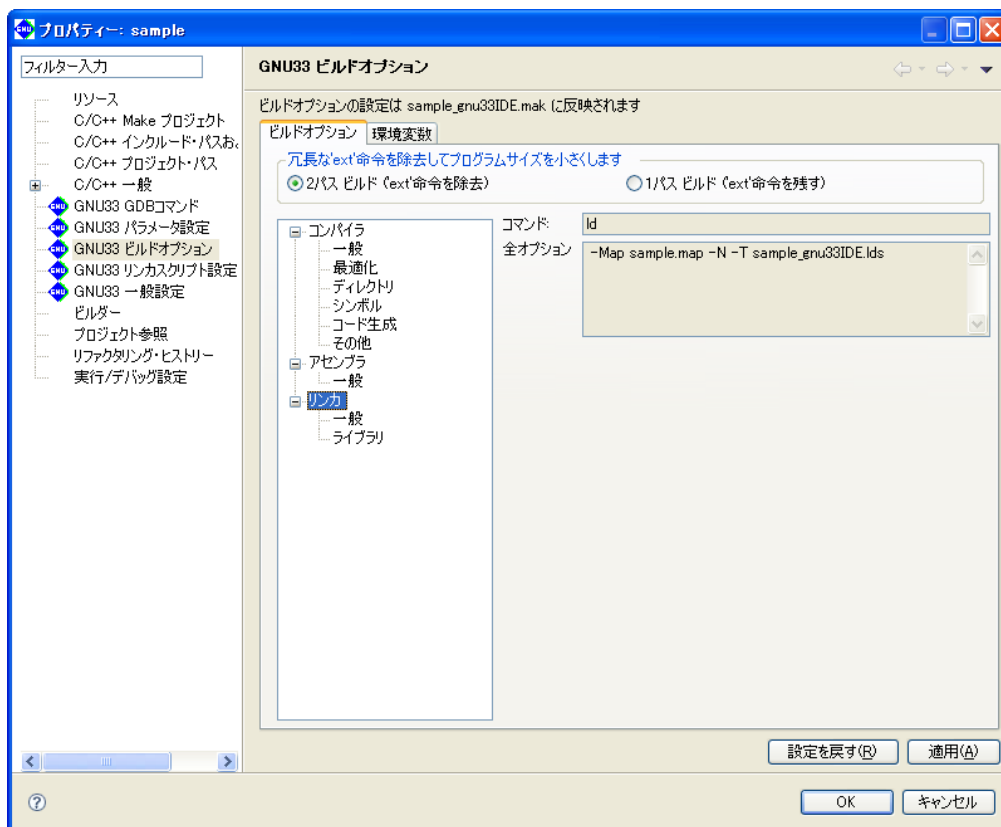
アセンブラに渡すオプションを入力します。オプション間には1個以上のスペースを挿入してください。

入力したオプションは"-Wa,<オプション>, ..."としてアセンブラに渡されます。

5.7.5 リンカオプションの設定

リンカのコマンドオプションは、次のように設定します。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ビルドするプロジェクトを選択します。
- (2) [プロジェクト]メニューまたは上記ビューのコンテキストメニューから[プロパティ]を選択します。
[プロパティ]ダイアログが表示されます。
- (3) プロパティの一覧から[GNU33 ビルドオプション]を選択します。
- (4) [ビルドオプション]のツリーから[リンカ]を選択します。



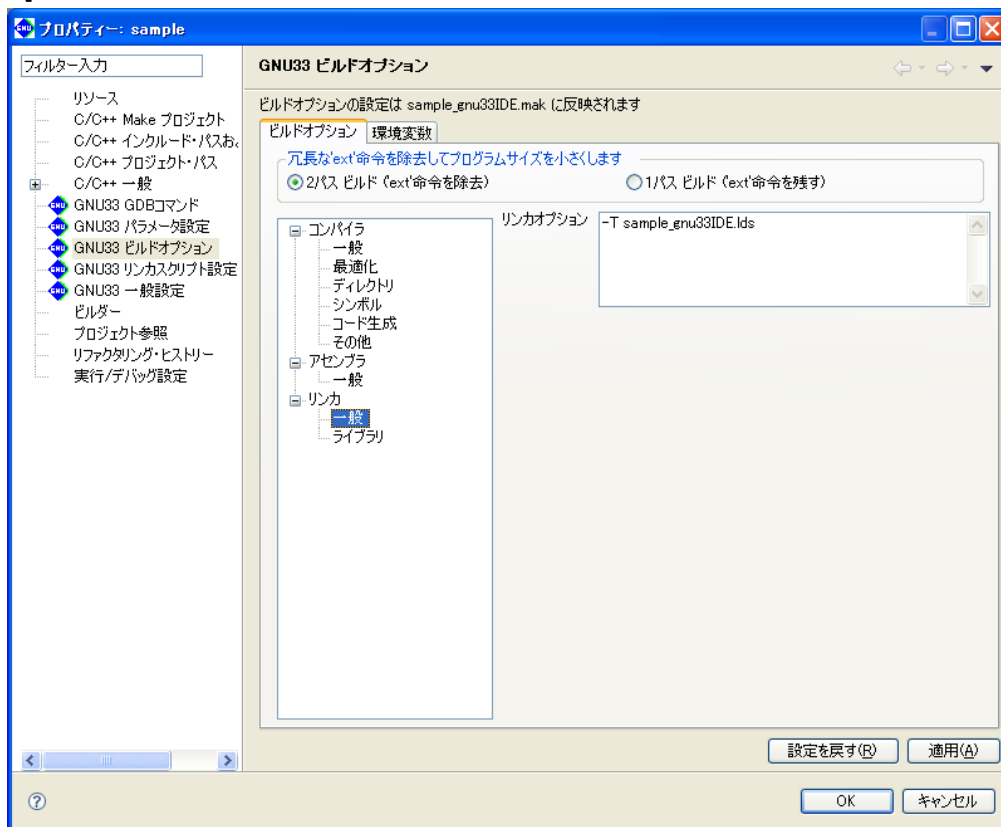
[コマンド:]はリンカのプログラム名、[全オプション]は現在設定されているオプションを表示します。

- (5) [リンカ]のツリーリストからカテゴリを選択し、必要なオプションを設定します。
- (6) 他のプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。
設定を変更した場合は、"clean"ビルド(5.7.9節参照)を実行するかどうかを確認するダイアログが表示され、以前の設定で生成されているファイルの削除(およびリビルド)が行えます。

[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

リンカオプションの設定ページを以下に示します。オプションの詳細は、リンカの章を参照してください。

[一般]

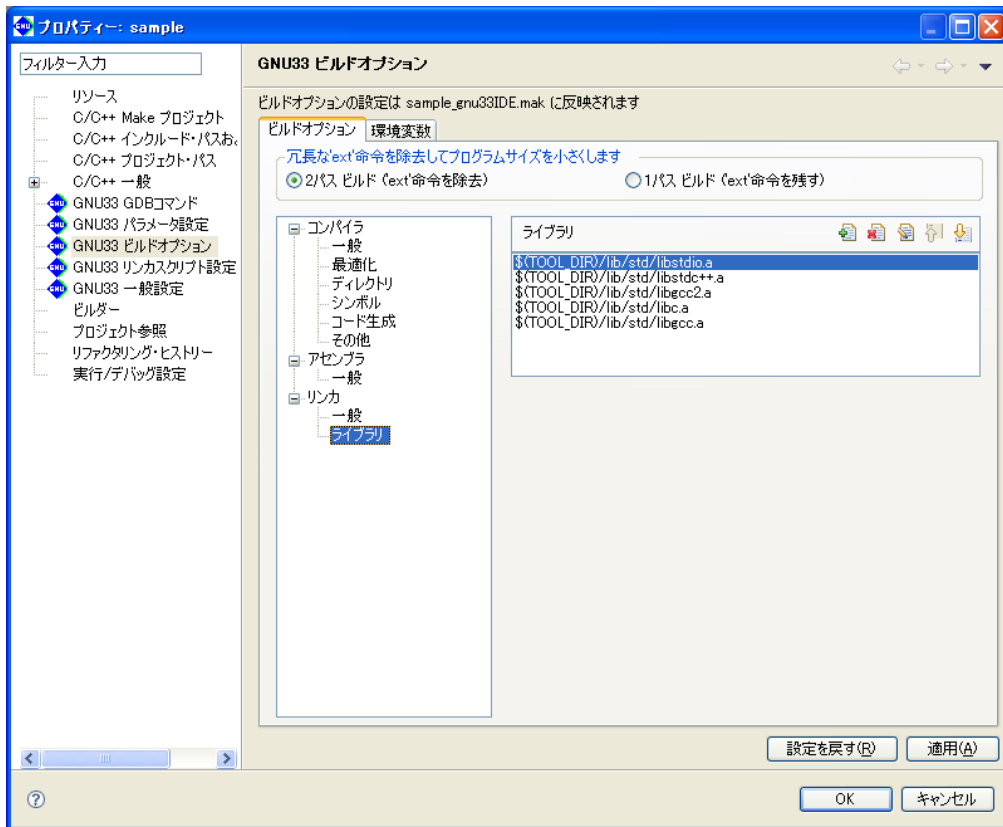


リンカの -Map、-N オプションは常に指定されます。このページではリンカのその他のオプションを選択します。

[リンカオプション](デフォルト: -T<プロジェクト名>_gnu33IDE.lids)

その他のリンカオプションを入力します。オプション間には1個以上のスペースを挿入してください。

[ライブラリ]



このページではリンクするライブラリを設定します。

[ライブラリ](デフォルト: libstdio.a, libstdc++.a, libgcc2.a, libc.a, libgcc.a)

リンクするライブラリを設定します。ボタンの機能は以下のとおりです。

-  **[追加...]** ライブラリを追加します。パスを入力あるいは[ファイル・システム...]ボタンで選択するダイアログが表示されます。
-  **[削除]** リスト内で選択したライブラリを削除します。
-  **[編集...]** リスト内で選択したライブラリを編集します。パスを書き換えるダイアログが表示されます。
-  **[上へ移動]** リスト内で選択したライブラリを一覧内の一つ上に移動します。ライブラリは一覧内の上から順にリンクされます。
-  **[下へ移動]** リスト内で選択したライブラリを一覧内の一つ下に移動します。

ここで設定したライブラリは、makeファイル内に記述され、ソースから生成されたオブジェクトリンクされます。ただし、リンクスクリプトファイル内でセクションにマップする必要があります。

※\$(TOOL_DIR)について

デフォルト設定で、[ライブラリ]には"\$ (TOOL_DIR) /lib/std/libxxx.a"が表示されています。

\$ (環境変数) は、ビルド時に生成されるmakeファイルに記述されるマクロです。TOOL_DIRは、gnu33ツールをインストールしたディレクトリへのパスが定義されている環境変数で、定義内容は[環境変数]タブのページで確認できます。

例: gnu33ツールをc:\EPSON\gnu33ディレクトリにインストールした場合

```
TOOL_DIR = c:/EPSON/gnu33
```

make.exeの実行時は、マクロが () 内の環境変数の内容に置き換えられますので、-I\$(TOOL_DIR) /lib/std/libxxx.aは-Ic:/EPSON/gnu33/lib/std/libxxx.aの指定となります。

[環境変数]タブのページでは、TOOL_DIRと同様なユーザ独自の環境変数を定義することができます。ここで定義した環境変数は、ビルドオプションのインクルードパスおよびライブラリファイルのパスの指定に使用可能です。[環境変数]ページの詳細については、5.10.1節を参照してください。

5.7.6 生成されるmakeファイル

CPUタイプおよび前記のツールオプションの設定に従い、ビルド時には"<プロジェクト名>_gnu-33IDE.mak"という名称のmakeファイルが生成され、**make.exe**によって実行されます。

生成されるmakeファイルの例を以下に示します。

例:

```
# Make file generated by Gnu33 Plug-in for Eclipse
# This file should be placed directly under the project folder

# macro definitions for target file (1)
TARGET= sample
GOAL= $(TARGET).elf

# macro definitions for tools (2)
TOOL_DIR= C:/EPSON/GNU33
CC= $(TOOL_DIR)/xgcc
CXX= $(TOOL_DIR)/xgcc
AS= $(TOOL_DIR)/xgcc
AS_CC= $(TOOL_DIR)/as
LD= $(TOOL_DIR)/ld
RM= $(TOOL_DIR)/rm
SED= $(TOOL_DIR)/sed
CP= $(TOOL_DIR)/cp
OBJDUMP= $(TOOL_DIR)/objdump

# macro definitions for tool flags (3)
CFLAGS= -B$(TOOL_DIR)/ -gstabs -S -medda32 -O1 -I$(TOOL_DIR)/include -fno-builtin
-mlong-calls -Wall -Werror-implicit-function-declaration
CXXFLAGS= -gstabs -S -medda32 -O1 -I$(TOOL_DIR)/include -fno-builtin -mlong-calls
-Wall
ASFLAGS= -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs -Wa,-medda32
ASFLAGS_CC= -medda32
LDFLAGS= -Map sample.map -N -T sample_gnu33IDE.lds
ALLOBJDUMP_FILE= __allobjects
EXTFLAGS= -Wa,-mc33_ext -Wa,$(TARGET).dump -Wa,$(ALLOBJDUMP_FILE).dump
EXTFLAGS_CC= -mc33_ext $(TARGET).dump $(ALLOBJDUMP_FILE).dump
OBJDUMPFLAGS= -t

# macro for switching 2pass or 1pass build (4)
PASS= 2pass

# search paths for source files
vpath %.c
vpath %.cpp
vpath %.cc
vpath %.cxx
vpath %.C
vpath %.s

# macro definitions for object files (5)
OBS= boot.o ¥
main.o ¥

# macro definitions for library files (6)
OBJLDS= $(TOOL_DIR)/lib/std/libstdio.a ¥
$(TOOL_DIR)/lib/std/libstdc++.a ¥
$(TOOL_DIR)/lib/std/libgcc2.a ¥
$(TOOL_DIR)/lib/std/libc.a ¥
$(TOOL_DIR)/lib/std/libgcc.a ¥

# macro definitions for assembly files generated from c source files (7)
CEXTTEMPS= main.ext0 ¥

# macro definitions for dependency files (8)
DEPS= $(OBS:%.o=%.d)
SED_PTN= 's/[[:space:]]¥([a-zA-Z]¥)¥:/ ¥/cygdrive¥/¥1/g'
SED_PTN2= 's/^¥($ (subst .,¥.,$(@F))¥)¥:/$(subst /,¥/,$(@))¥:/g'
```

```

# macro definitions for creating dependency files (9)
DEPCMD_CC= @$ (CC) -M -MG $(CFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e $(SED_PTN2)
>$(@:%.o=%.d)
DEPCMD_CXX= @$ (CC_KFILT) -M -MG $(CXXFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e
$(SED_PTN2) >$(@:%.o=%.d)
DEPCMD_AS= @$ (AS) -M -MG $(ASFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e $(SED_
PTN2) >$(@:%.o=%.d)

# targets and dependencies (10)
.PHONY : all clean

all : $(GOAL) (11)

$(TARGET).elf : $(OBJS) sample_gnu33IDE.mak sample_gnu33IDE.lds
ifeq ($(PASS), 1pass)
# 1pass linking
$(LD) $(LDFLAGS) -o $$ $(OBJS) $(OBJLDS)
else
# 1pass linking
-$(LD) $(LDFLAGS) -o $$ $(OBJS) $(OBJLDS) 2>lderr
@if [ -s lderr ]; then ¥
    cmd /c "type lderr" ¥
    && $(RM) -f $(TARGET).elf ¥
    && exit 1; ¥
else $(RM) -f lderr ; ¥
fi
$(OBJDUMP) $(OBJDUMPFLAGS) $$@ > $(TARGET).dump
$(RM) -f $(TARGET).elf
# create all objets dump file
$(RM) -f $(ALLOBJDUMP_FILE).dump
for NAME in $(OBJS) ; do ¥
    $(OBJDUMP) $(OBJDUMPFLAGS) $$NAME >> $(ALLOBJDUMP_FILE).dump ; done
# save 1pass object files
@if [ -e obj1pass ]; then ¥
    cmd /c "rd /s /q obj1pass" ; ¥
fi
cmd /c "md obj1pass"
for NAME in $(subst /,¥¥,$(OBJS)) ; do ¥
    cmd /c "copy /y $$NAME obj1pass¥¥$$NAME" >nul ; done ¥
&& $(RM) -f $(OBJS)
# 2pass for assembly files
$(AS) $(ASFLAGS) $(EXTFLAGS) -o boot.o boot.s
# 2pass for c files
for NAME in $(basename $(CEXTTEMPS)) ; do ¥
    $(AS_CC) $(ASFLAGS_CC) $(EXTFLAGS_CC) -o $$NAME.o $$NAME.ext0 ; done
$(RM) -f $(TARGET).map
# 2pass linking
$(LD) $(LDFLAGS) -o $$ $(OBJS) $(OBJLDS)
# restore 1pass object files
$(RM) -f $(OBJS) ¥
&& ¥
for NAME in $(subst /,¥¥,$(OBJS)) ; do ¥
    cmd /c "copy /y obj1pass¥¥$$NAME $$NAME" >nul ; done ¥
&& cmd /c "rd /s /q obj1pass"
endif
@cmd /c "echo ----- Finished building target : $$@ -----"

## boot.s (12)

boot.o : boot.s
$(AS) $(ASFLAGS) -o $$@ $<
$(DEPCMD_AS)

## main.c
main.o : main.c main.ext0
$(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<

```

```

$(AS_CC) $(ASFLAGS_CC) -o $$ $(@:%.o=%.ext0)
$(DEPCMD_CC)

## cpp.c
cpp.o : cpp.cpp cpp.ext0
$(CC) $(CXXFLAGS) -o $$ (@:%.o=%.ext0) $<
$(AS_CC) $(ASFLAGS_CC) -o $$ (@:%.o=%.ext0)
$(DEPCMD_CXX)

# dependencies for assembled c source files
main.ext0 : main.c

# include dependency files
-include $(DEPS)

# clean files
clean :
$(RM) -f $(OBSJS) $(TARGET).elf $(TARGET).map $(DEPS) $(CEXTTEMPS) $(TARGET).dump
$(ALLOBJDUMP_FILE).dump lderr
@if [ -e objlpass ]; then ¥
    cmd /c "rd /s /q objlpass" ; ¥
fi

```

(13)

番号を付けて示した各フィールドの内容は以下のとおりです。

- (1) プロジェクト名をTARGETとして定義しています。この名前がelfオブジェクトファイルやマップファイルにも使用されます。
 - (2) ツールのディレクトリと、コンパイラ、アセンブラ、リンカのコマンドを定義しています。TOOL_DIRにはツールが存在するディレクトリが設定されます。パス内の空白文字は、"¥" (¥ + スペース) に変換されます。ただし、パスが空白を含んでいる場合、リンク処理が正常に行われません。したがって、S5U1C33001Cツール一式を、空白を含まないディレクトリにインストールしてください。
 - (3) コンパイラ、アセンブラ、リンカのオプションを定義しています。これらのオプションにはプロジェクトプロパティ ([GNU33 ビルドオプション]) の選択内容が反映されます。
 - (4) 2パスメイクまたは1パスメイクのビルドオプションを定義しています。このオプションにはプロジェクトプロパティ ([GNU33 ビルドオプション]) の選択内容が反映されます。
 - (5) "OBSJS="に続き、プロジェクト内のソースファイルに対応するオブジェクトファイル名が記述されます。ソースファイルの追加や削除により、このフィールドの記述内容は変更されます。
 - (6) "OBJLDS="の後には、[GNU33 ビルドオプション]>[ビルドオプション]>[リンカ]>[ライブラリ]で設定したライブラリファイル名が"OBSJS="と同様に記述されます。
 - (7) 2パスメイクによる拡張命令の最適化を行うため、アセンブラソースファイルが記述されます。C/C++ソースファイルから生成されるアセンブラソースファイルは拡張子が".ext0"になります。
 - (8) 依存関係ファイルを作成するためのマクロを定義しています。依存関係ファイルは、各ソースに対応して生成されます。オブジェクトファイルの生成に必要なソースとインクルードファイルが定義されます。
例: 依存ファイル(main.d)
main.o: main.c

依存ファイル(boot.d)
boot.o: boot.s
- これらのファイルは依存リストに記述するツールコマンドの作成に使用されます。
- (9) 依存リスト内に置く実行コマンドを定義しています。
 - (10) ターゲットを定義しています。

- (11) ビルドのターゲットと、実行形式オブジェクトファイルの依存リストです。
2パスメイクを指定した場合、実行形式のオブジェクトファイルは拡張命令が最適化されて出力されます。
 - (12) 各ソースから生成するオブジェクトファイルの依存リストです。
ソースファイルの追加や削除により、このフィールドの記述内容は変更されます。
 - (13) ターゲット"clean"で実行する、生成ファイルの削除コマンドが記述されています。
- makeファイルの詳細は、"11.1 make.exe"を参照してください。

5.7.7 リンカスクリプトの編集

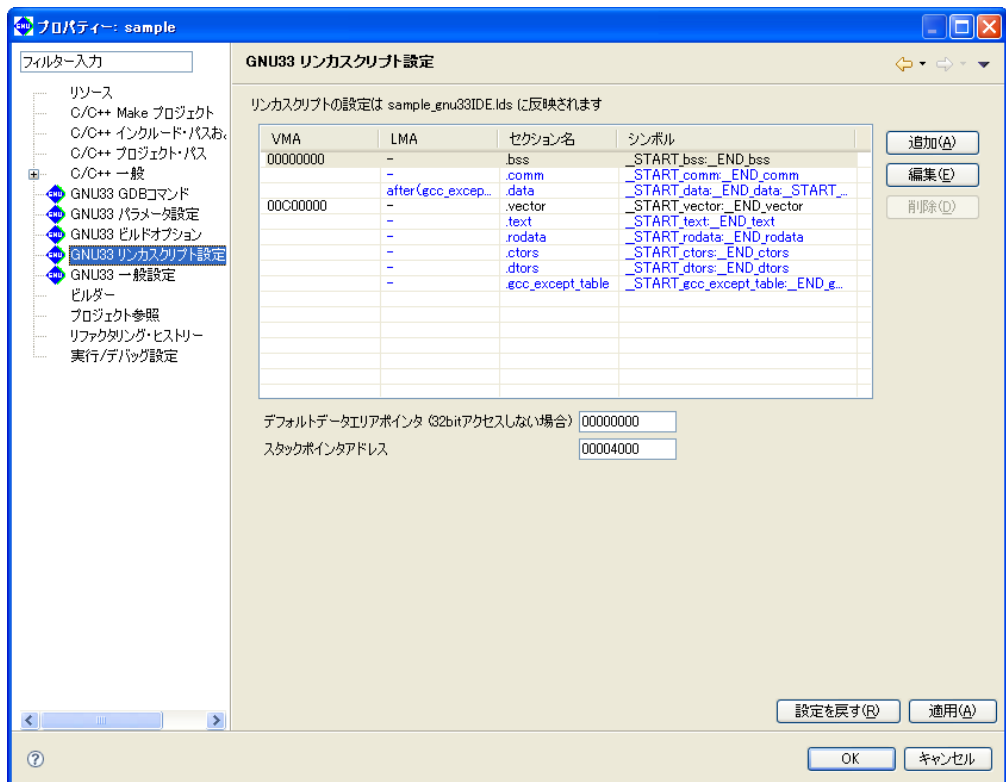
リンカスクリプトファイルは、実行ファイル(.elf)を構成するオブジェクトファイルの配置情報をリンカに渡すためのファイルです。IDEは、本節で説明する設定により"<プロジェクト名>_gnu33IDE.lds"というリンカスクリプトファイルを生成します。

以下、リンカスクリプトの設定手順を説明します。セクションとリンカスクリプトの詳細は、"3.7 データエリア、セクション"および"9 リンカ"を参照してください。

●リンカスクリプト設定ページ

リンカスクリプトは、プロジェクトプロパティの[GNU33 リンカスクリプト設定]ページで設定します。次の手順で設定ページを表示させます。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ビルドするプロジェクトを選択します。
- (2) [プロジェクト]メニューまたは上記ビューのコンテキストメニューから[プロパティ]を選択します。
[プロパティ]ダイアログが表示されます。
- (3) プロパティの一覧から[GNU33 リンカスクリプト設定]を選択します。



一覧は実行ファイル(.elf)内のセクション構成と配置を示しています。このメモリマップを図5.7.7.1に示します。

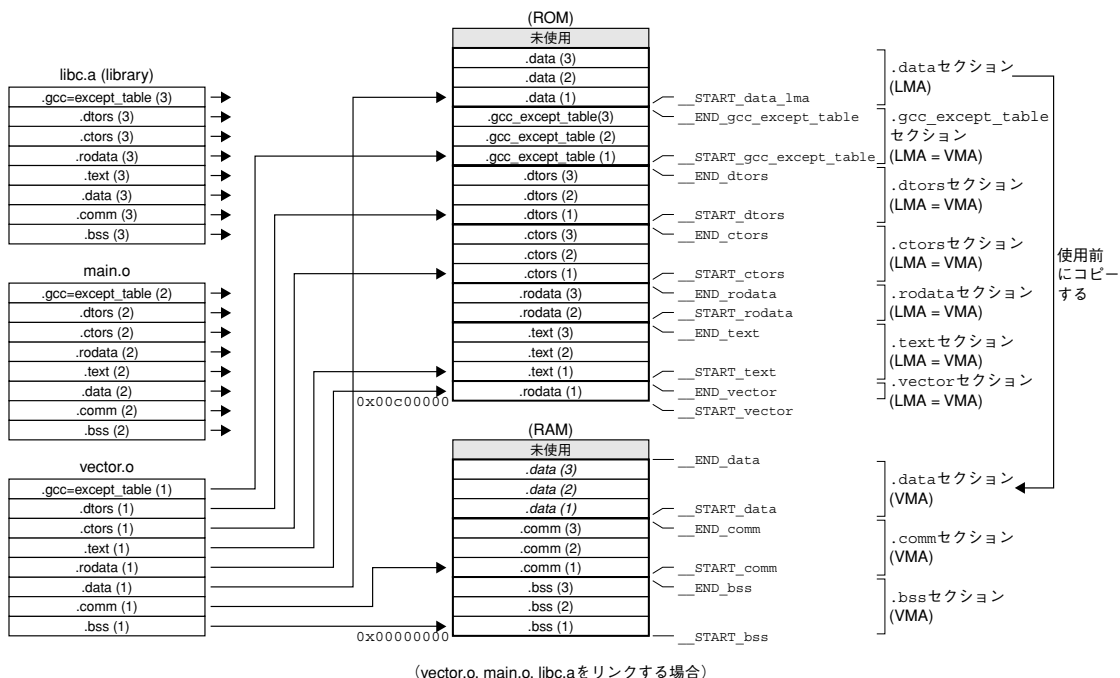


図5.7.7.1 デフォルト設定のセクション配置

[セクション名]欄に表示されているとおり、以下の9つの基本的なセクションがあらかじめ設定されています。

- .bss: 初期値を持たない変数を置くセクション(通常はRAMに置きます)
- .comm: 初期値を持たない変数を置くセクション(通常はRAMに置きます)
- .data: 初期値を持つ変数を置くセクション(初期値をROMに置き、使用時はRAMにコピーしてアクセスします)
- .vector: ベクタテーブルを置くセクション(実データはROMに置きます)
- .text: プログラムコードを置くセクション(実データはROMに置き、その位置で、あるいは高速なRAMなどにコピーして実行します)
- .rodata: 定数(実データはROMに置きます)
- .ctors: グローバルクラスのコンストラクタ関数へのポインタ配列(実データはROMに置きます)
- .dtors: グローバルクラスのデストラクタ関数へのポインタ配列(実データはROMに置きます)
- .gcc_except_table: 例外処理用テーブルデータ(実データはROMに置きます)

セクション情報は、.vectorのみ黒色、その他のセクションは青色で表示されています。青色はデフォルトで定義済みの標準セクションを表し、黒色はそれ以外の新たに定義したユーザセクションを表します。

ユーザセクションは、セクション名、セクションの基本属性、配置アドレス、配置するオブジェクトを任意に編集可能です。標準セクションは配置アドレスのみ指定可能で、ユーザセクションに配置された以外のオブジェクトが自動的に配置されるようになっています。

[VMA](Virtual Memory Address)は実行時にセクションを置く位置(先頭アドレス)です。VMAにアドレスの記載のないセクションはその一つ上のセクションに続いて配置することを表します。

[LMA](Load Memory Address)は実データを置いておくROM内の位置(先頭アドレス)です。"- "はVMAと同じ(実データが置かれた位置で実行またはアクセスされる)ことを表します。"after (.gcc_except_table)"は()内のセクション、この場合は.gcc_except_tableセクション、に続いて実データが配置されることを示します。

[シンボル]にはセクションが配置される領域の先頭および終了アドレスを示すラベルが表示されます。LMAが指定されない場合は2つのラベルが、LMAが指定されている場合はVMAの先頭/終了、LMAの先頭/終了の順に4つのラベルが表示されます。これらのラベルは、ROMからRAMへのセクションのコピーの際など、ソースファイル内でのアドレス指定に使用できます。これらのラベル名はセクション名から自動的に生成されます。

例: `__START_bss: __END_bss`

.bssセクションの先頭、終了アドレスを示すラベル

`__START_data: __END_data: __START_data_lma: __END_data_lma`

.dataセクションのVMA先頭、VMA終了、LMA先頭、LMA終了を示すラベル

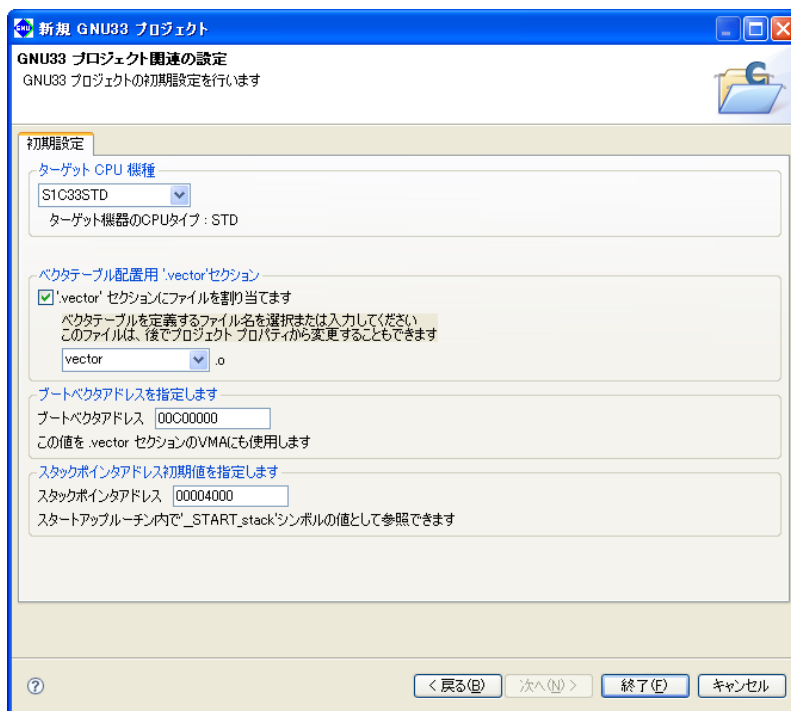
●.vectorセクションについて

.vectorセクションはベクタテーブルを確実にベクタテーブルベースアドレスから配置するために、IDEが独自に設けているセクションです。

IDEの初期設定では、.textセクションは.vectorセクションの直後に配置されます。

このセクションに配置するオブジェクトとしてベクタテーブルを含むファイルを指定することで、他のオブジェクトとの配置順序を考慮することなく確実にベクタテーブルを上記のアドレスから配置することができます。

配置するオブジェクトは新規プロジェクト作成ウィザードで、ターゲットCPUの選択と同じページで行えます。新規プロジェクト作成ウィザードについては、「5.4.2 新規プロジェクトの作成」を参照してください。



.vectorセクションに配置するオブジェクトをコンボボックスで選択(vector.oとboot.oが選択可能)、もしくは入力します。

.vectorセクションを配置するアドレスは、[ブートベクタアドレス]テキストボックスで指定します。

デフォルト設定値は、ターゲットCPUに"S1C33STD"または"S1C33PE"を選択した場合は"00C00000"、"S1C33401"を選択した場合は"20000000"となります。ここに設定した値は、リンカスクリプト内の`.vector`セクションのVMAとして使用されるほか、**IDE**が自動生成するデバッグ起動用コマンドファイル内のTTBR設定コマンドのパラメータとしても使用されます。

`.vector`セクションに配置しない場合は、`['.vector'` セクションにファイルを割り当てます]のチェックを外してください。その場合でも、`.vector`セクションはオブジェクトを持たないセクションとして定義されます。

ウィザードで設定した内容は、[編集セクション]ダイアログ("●セクション情報の編集"参照)で変更も可能です。

`.vector`セクションは`.rodata`属性で定義されます。したがって、ソースファイルにはベクタテーブルを次のように記述してください。

C/C++ソースの場合(`vector.c`または`vector.cpp`)

`const`宣言をして、`.rodata`セクションに配置されるようにしてください。

```
例: const unsigned long vector[] = {
    (unsigned long)boot,           // 0    0
    0,                            // 4    1
    0,                            // 8    2
                                :
    (unsigned long)dummy,         // 280  70
    (unsigned long)dummy         // 284  71
};
```

アセンブラソースの場合(`boot.s`)

`.rodata`セクションを宣言してベクタテーブルを記述してください。

```
例: .section .rodata, "a"
    .long    BOOT                ; 0    0
    .long    0,                  ; 4    1
    .long    0,                  ; 8    2
                                :
    .long    DUMMY               ; 280  70
    .long    DUMMY               ; 284  71
```

ベクタテーブルが`.text`セクションに記述されている既存のアセンブラソースを使用し、`.vector`セクションに配置する場合は、以下の方法から選択してください。

方法1 ソースファイルを変更する

- (1) ソースファイルのベクタテーブルの前に、上記のような`.rodata`疑似命令を挿入します。ベクタテーブルの後にプログラムが記述されている場合は、その前に`.text`疑似命令を挿入し`.text`セクションを宣言してください。
- (2) 新規プロジェクトウィザードで`['.vector'` セクションにファイルを割り当てます]を選択し、コンボボックスで`boot.o`を選択します(ソースが`boot.s`以外の場合は、そのファイル名を入力してください)。

方法2 IDEでセクション情報を変更する(ソースファイルはそのまま使用)

- (1) 新規プロジェクトウィザードで`['.vector'` セクションにファイルを割り当てます]を選択し、コンボボックスで`boot.o`を選択します(ソースが`boot.s`以外の場合は、そのファイル名を入力してください)。
- (2) [編集セクション]ダイアログ上で`.vector`セクションの属性を`.text`に変更します。(次ページ以降の説明を参照してください。)

`.vector`セクションを使用しない場合は、新規プロジェクトウィザードで`['.vector'` セクションにファイルを割り当てます]のチェックを外し、[編集セクション]ダイアログによる`.text`セクションの編集で`boot.o`を先頭に配置してください。

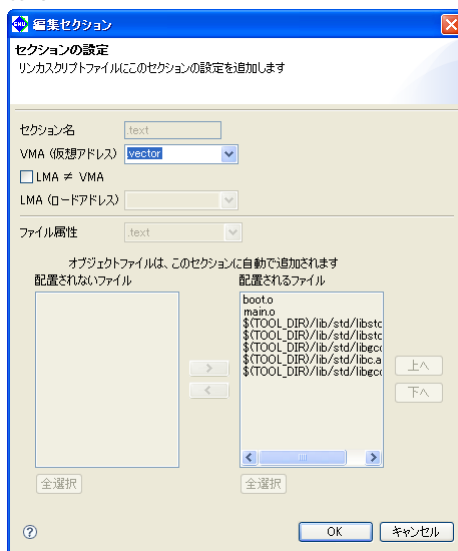
●セクション情報の編集

上記の各セクションの配置情報、およびセクション内に配置するオブジェクトあるいはライブラリは、システムに合わせて変更可能です。その手順を以下に示します。

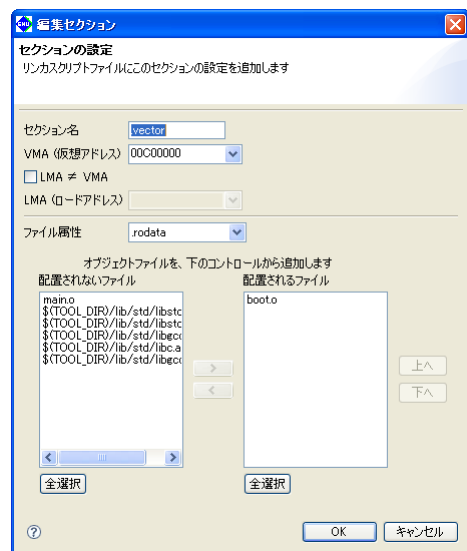
- (1) [GNU33 リンカスクリプト設定] ページのセクション一覧から編集するセクションをクリックして選択します。
- (2) [編集] ボタンをクリックします。
[編集セクション] ダイアログが表示されます。
- (3) 下記の説明に従って必要な変更を行い、[OK] をクリックします。
- (4) 他のセクションまたはプロパティを変更する場合は[適用] ボタンを、プロパティの設定を終了する場合は[OK] ボタンをクリックします。
[適用] をクリックする前であれば、[設定を戻す] ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

[編集セクション] ダイアログ

標準セクション



ユーザセクション



ダイアログの上部では、実行ファイル内のセクションの配置情報を設定します。

[セクション名]

セクション名(出力セクション)を設定します。

注: セクション名を入力する場合、次の点に注意してください。

- ・セクション名は"."で始まる必要があります。
- ・セクション名に使用可能な文字は、半角英数字、"_"、"."のみです。

[VMA (仮想アドレス)]

実行時のセクション配置(先頭アドレス)を設定します。他のセクションに続けて配置する場合は、直前のセクション名をプルダウンリストから選択します。デバイスの先頭、あるいは前のセクションとは離して配置する場合は、そのアドレス(16進数)を入力します。

注: アドレスを入力する場合は、0~9とA~Fのみを使用し、8文字以内としてください。それ以外はアドレスではなくセクション名を入力したものと見なされます。

[LMA ≠ VMA]

初期値を持つ変数(.dataセクションなど)やRAM上で実行するプログラムなど、実行アドレス(VMA)と格納アドレス(LMA)が異なる場合に選択します。ROMなどの格納場所でそのまま実行させる場合は非選択とします。

[LMA(ロードアドレス)]

[LMA ≠ VMA]を選択した場合に、格納アドレスを設定します。他のセクションに続けて配置する場合は、直前のセクション名をプルダウンリストから選択します。デバイスの先頭、あるいは前のセクションとは離して配置する場合は、そのアドレス(16進数)を入力します。

注: アドレスを入力する場合は、0～9とA～Fのみを使用し、8文字以内としてください。それ以外はアドレスではなくセクション名を入力したものと見なされます。

ダイアログの下部では、セクション内のオブジェクトの配置情報を設定します。

以下の設定は、ユーザセクションのみ変更可能です。標準セクションは、あらかじめプロジェクトの全オブジェクトファイルおよびビルドオプションで設定済みのライブラリが配置されるように定義されています。同属性のユーザセクションが定義されると、標準セクションの設定内容は、ユーザセクションと重複しないように自動的に更新されます。

[ファイル属性]

ユーザセクションの基本属性(標準セクションの各属性と同じ)をプルダウンリストから選択します。

標準セクションでは、変更できません。

[配置されないファイル](左)

ユーザセクションの場合、プロジェクトのオブジェクトファイルおよびビルドオプションで設定済みのライブラリの中で、本セクションに配置されないファイルの一覧を表示します。ビルド前でも、プロジェクト内のソースファイル名からオブジェクトファイルの一覧が作成されます。この中から、本セクションに配置するオブジェクトを選択します。

標準セクションの場合、このリストは空白になるか、同属性のユーザセクションに配置されているオブジェクトが選択不可の状態が表示されます。

[配置されるファイル](右)

本セクションに配置するオブジェクトファイルとライブラリの一覧を表示します。

標準セクションの場合、プロジェクト内のオブジェクトファイルおよびビルドオプションで設定済みのライブラリの中で本セクションに配置可能なファイル(同属性のユーザセクションに配置されていないもの)がすべてリストされます。リンク時は、これらのファイル内から出力セクションと同じ属性のセクションが抽出され、出力セクション内に一覧の上から順に配置されます。オブジェクトファイルはアルファベット順に並べられ、続いてライブラリがビルドオプションの設定順に並びます。

標準セクションのリストはユーザセクションの設定に従って自動的に更新されます。ユーザセクションのように手動で変更することはできません。

ユーザセクションの場合、[<]、[>]、[上へ]、[下へ]ボタンを使用してファイルの構成を変更できます。

[>]

本セクションに配置するファイルを[配置されないファイル]から選択します。標準セクションでは無効です。

[<]

本セクションに配置しないファイルを[配置されるファイル]から[配置されないファイル]に移します。標準セクションでは無効です。

[上へ]

セクション内のオブジェクトの配置を変更します。[配置されるファイル]内のオブジェクトを選択して、本ボタンをクリックすると、一つ上のオブジェクトと位置が入れ替わります。標準セクションでは無効です。

[下へ]

セクション内のオブジェクトの配置を変更します。[配置されるファイル]内のオブジェクトを選択して、本ボタンをクリックすると、一つ下のオブジェクトと位置が入れ替わります。標準セクションでは無効です。

[全選択]

それぞれのリスト内のファイルをすべて選択します。標準セクションでは無効です。

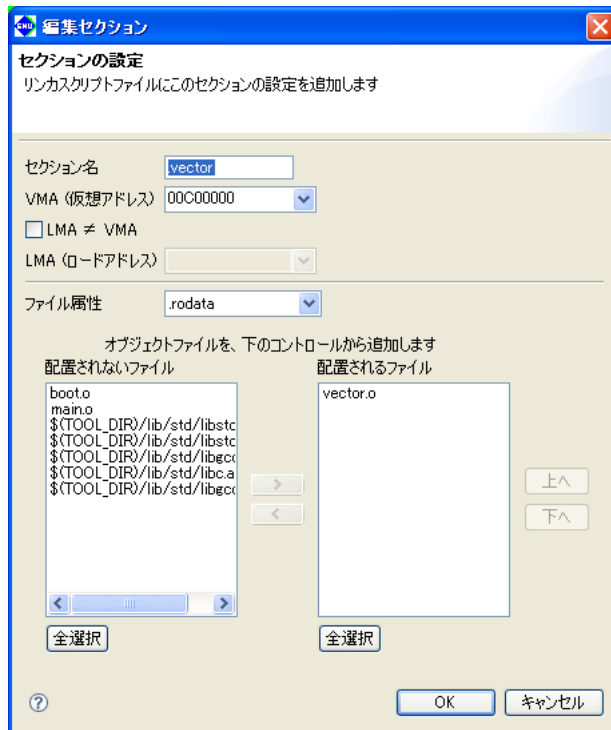
セクション情報編集例(.vectorセクション)

セクション情報の編集例として、新規プロジェクトウィザードのデフォルト設定で作成された.vectorセクション情報を変更します。

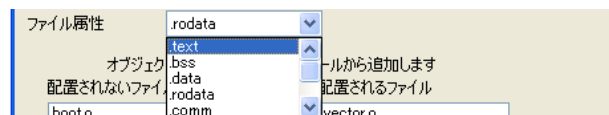
セクション属性: .rodata → .text

オブジェクト: vector.o → boot.o

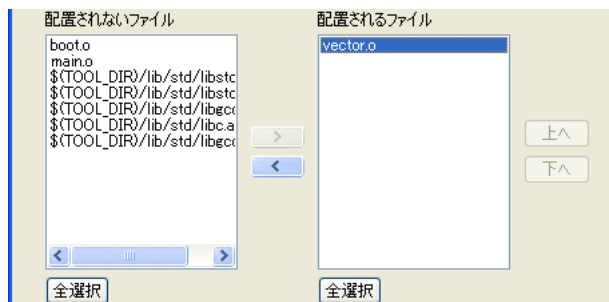
- (1) プロジェクトプロパティの[GNU33 リンカスクリプト設定]ページで.vectorセクションを選択し、[編集]ボタンをクリックします。
[編集セクション]ダイアログが開きます。



- (2) [ファイル属性]のプルダウンリストから.textを選択します。

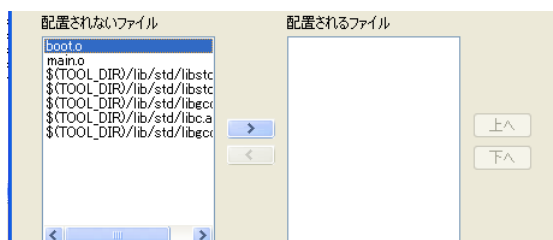


- (3) [配置されるファイル]からvector.oを選択し、[<]ボタンをクリックします。

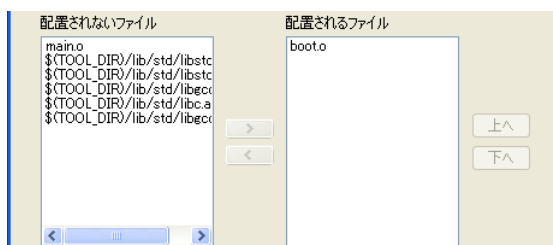


vector.oはソースファイルがないため、[配置されないファイル]には移動せずに削除されます。この操作のみで、vector.oは他のセクション情報からも削除されます。

- (4) [配置されないファイル]からboot.oを選択し、[>]ボタンをクリックします。

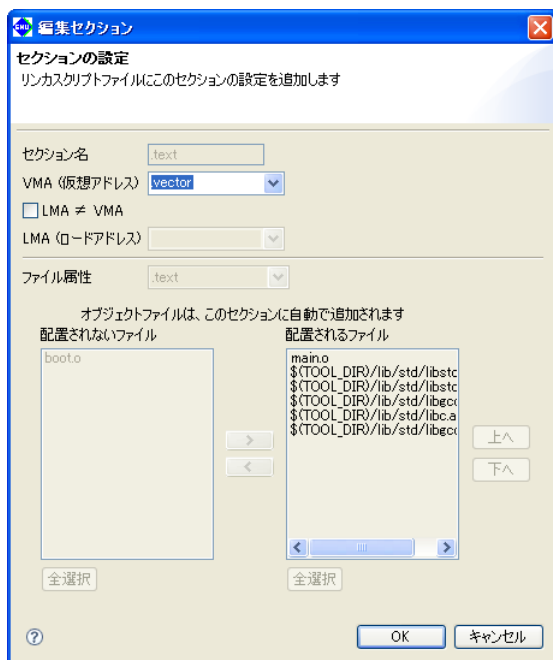


boot.oが[配置されるファイル]に移動します。



- (5) [OK]ボタンで編集を終了します。

boot.oの.textセクションが.vectorセクションに配置されたため、.textセクション情報のboot.oは[配置されないファイル]に移動します。(1つのファイルを同属性の複数のセクションに配置することはできません。)



- (6) [編集セクション]ダイアログを[OK]ボタンで終了しただけでは、リンクスクリプトに反映されません。必ず[プロパティー]ダイアログの[適用]ボタンまたは[OK]ボタンをクリックして編集内容を確定させてください。

●オブジェクトファイルの自動更新(標準セクション)

標準セクションの場合、セクションに配置するオブジェクトファイルの自動更新機能が有効となります。後からソースを追加/削除した場合なども自動的にリンカスクリプトが修正されますので、ソースを追加/削除するたびにリンカスクリプトを編集する必要はありません。

自動更新の処理内容は以下のとおりです。

同じ属性のセクションが他にない場合

プロジェクトのオブジェクトファイルおよびビルドオプションで設定済みのライブラリがすべて選択され、そのセクション内に配置されます。

同じ属性のセクションが複数ある場合

同属性のユーザセクションに配置されていないオブジェクトファイルとライブラリが選択され、標準セクション内に配置されます。すでに同属性のユーザセクションに含まれているファイルは自動更新では配置されません。

ユーザセクションの場合は、他の同属性のセクションに含まれているファイルも[配置されないファイル]から選択して追加することができます。他のセクションに配置し直すと、そのファイルはそれまで配置されていたセクションから取り除かれます。

自動更新が有効となる標準セクションの制限事項を以下に示します。

1. セクション属性([ファイル属性])の変更はできません。
2. ファイルリストは参照のみで操作できません。
3. オブジェクトファイルはアルファベットの昇順で追加されます。

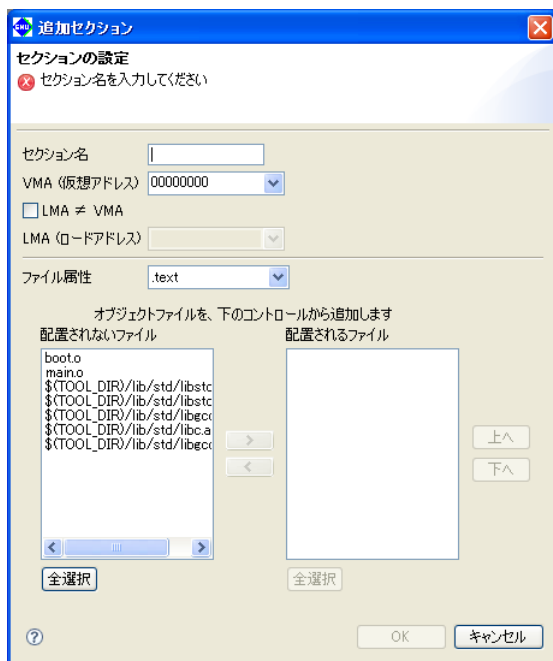
これらを変更する場合はユーザセクションを作成してください。

●セクションの追加

新規のセクションは次のように追加します。

- (1) [追加]ボタンをクリックします。

[追加セクション]ダイアログが表示されます。



- (2) 前記の説明に従って必要な設定を行い、[OK]をクリックします。

- (3) 他のセクションまたはプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。

[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

一覧内には、設定したVMAに従って挿入されます。

新規のセクションの配置やオブジェクトが他のセクションと重複しないように、他のセクションの情報も見直し、必要であれば変更してください。

●セクションの削除

不要なセクションは次の手順で削除します。

ただし、削除可能なセクションはユーザ定義セクションのみで、標準セクションは削除できません。

- (1) [GNU33 リンカスクリプト設定]ページのセクション一覧から削除するセクションをクリックして選択します。
- (2) [削除]ボタンをクリックします。
- (3) 操作を確認するダイアログが表示されますので、削除する場合は[OK]を、中止する場合は[キャンセル]をクリックします。
- (4) 他のセクションまたはプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。
[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

※削除による参照セクションの変更について

```
Section1    ([VMA (仮想アドレス)] = 00000000)
Section2    ([VMA (仮想アドレス)] = Section1)
Section3    ([VMA (仮想アドレス)] = Section2)
Section4    ([VMA (仮想アドレス)] = Section3)
```

上記のようなセクション構成でSection2が削除された場合、Section3はSection1を参照する設定に自動的に変更されます。

```
Section1    ([VMA (仮想アドレス)] = 00000000)
Section3    ([VMA (仮想アドレス)] = Section1)
Section4    ([VMA (仮想アドレス)] = Section3)
```

Section1を削除した場合、Section2の配置アドレスは(0x)00000000となります。問題が生じる場合は、セクション情報を再編集してください。

```
Section2    ([VMA (仮想アドレス)] = 00000000)
Section3    ([VMA (仮想アドレス)] = Section2)
Section4    ([VMA (仮想アドレス)] = Section3)
```

●データポインタの設定

[GNU33 リンカスクリプト設定]ページの[デフォルトデータエリアポインタ]テキストボックスは、リンカスクリプトに記述するデフォルトデータエリアポインタ(アドレス)の設定に使用します。データエリアポインタは、C33コアの内部レジスタをポインタとして使用し、そのレジスタをベースアドレスとして規定範囲内のデータをアクセスすることで、命令数を減らすことを目的としています。デフォルト設定では、ベースアドレスを0番地とするデータポインタ定義がリンカスクリプトに記述されます。

```
/* data pointer symbols */
__dp = 0x00000000;
```

注: デフォルト設定のコンパイラとアセンブラのビルドオプションでは、データポインタを無効とする `-medda32` オプションが選択されています。データポインタを使用するには、上記の設定に加え、`-medda32` オプションを非選択にするとともに、ソース内にもデータポインタを初期化する命令が必要です。詳細は、"3.7.1 データエリア"を参照してください。

●スタックポインタについて

[GNU33 リンカスクリプト設定]ページの[スタックポイントアドレス]テキストボックスは、IDEが自動生成するリンカスクリプトファイルの `__START_stack` シンボルの値となり、スタック領域の開始アドレスとしてシンボルを使用することができます。

新規プロジェクトウィザードでデフォルトの値が設定されます。

デフォルト設定値は、ターゲットCPUに"S1C33STD"または"S1C33PE"を選択した場合は"00004000"、"S1C33401"を選択した場合は"00008000"となります。

例) ビルド時、リンカスクリプトファイルには以下のように__START_stackシンボルが出力されます。

```
/* stack pointer symbols */
__START_stack = 0x00004000;
```

例) ブートルーチン内で以下のように記述できます。

```
boot:
    xld.w %r15, __START_stack
    ld.w %sp, %r15
```

●注意事項

- [編集セクション]ダイアログの[セクション名]、[VMA (仮想アドレス)]、[LMA (ロードアドレス)]に入力可能な文字数は最大255文字です。
- セクション名は"."で始まる必要があります。また、使用可能な文字は、半角英数字、"_","."のみです。
- [VMA (仮想アドレス)]、[LMA (ロードアドレス)]にアドレスを入力する場合は、0～9とA～Fのみを使用し、8文字以内としてください。それ以外はアドレスではなくセクション名を入力したものと見なされます。
- IDEはセクション同士の後方参照や循環参照についてはチェックしますが、完全ではありません。その場合は、リンク時にエラーが発生します。
- [新規 GNU33 プロジェクト]ウィザードで指定した、.vectorセクションに割り当てるオブジェクトファイル(vector.o)は、プロジェクトフォルダ直下に作成されるようにしてください。オブジェクトファイルを別フォルダに置く場合は、リンカスクリプト設定の.vectorセクションを変更してください。

●リンカスクリプトの設定例

ここでは、いくつかのセクション構成に対応したリンカスクリプトの設定例を、画面サンプル等で示します。操作方法については、前述の説明を参照してください。

例1. 最小のセクション構成

例2. 基本レイアウトの変更

例3. RAMエリアを複数の変数で共用

例4. RAM上でプログラムを実行

なお、説明の簡略化のため、オブジェクトファイルには.comm、.ctors、.dtors、.gcc_except_tableセクションは含まれないものとします。(リンカスクリプトはこれらのセクションにも対応して作成しています。)

また、アセンブラソース(boot.o)内のベクタテーブルは.rodataセクションに記述され、新規プロジェクト作成ウィザードで.vectorセクションに配置するように設定されたものとします。

例1. 最小のセクション構成例

内蔵RAMと外部ROMを1つ使用する最も簡単な構成のシステムを説明します。

外部ROMをアドレス0xc00000から割り当て、図5.7.7.2のようにプログラムとデータを配置します。プログラムはROM上の格納アドレス(LMA)でそのまま実行するものとし、静的データもROMから直接読み出して使用するものとします。アドレス0x0から始まる内蔵RAMには初期値を持たない変数領域を先頭から配置し、その後を初期値を持つ変数領域として使用します。変数の初期値はROMに格納しておき、アプリケーションプログラムがRAMにコピーします。

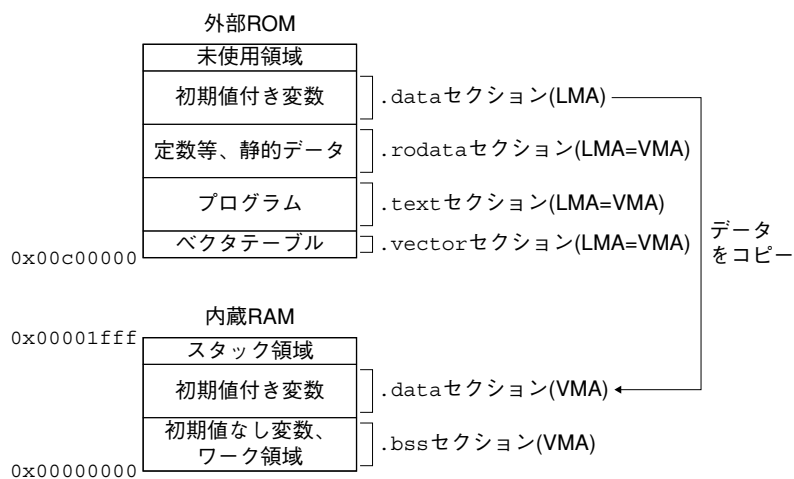


図5.7.7.2 メモリ構成例1

このようなメモリの使い方に対応する最も簡単な方法は、デフォルトのリンクスクリプトファイルを使用することです。
[GNU33 リンカスクリプト設定]ダイアログで設定の追加や修正を行わなくても、このセクション配置が実現できます。

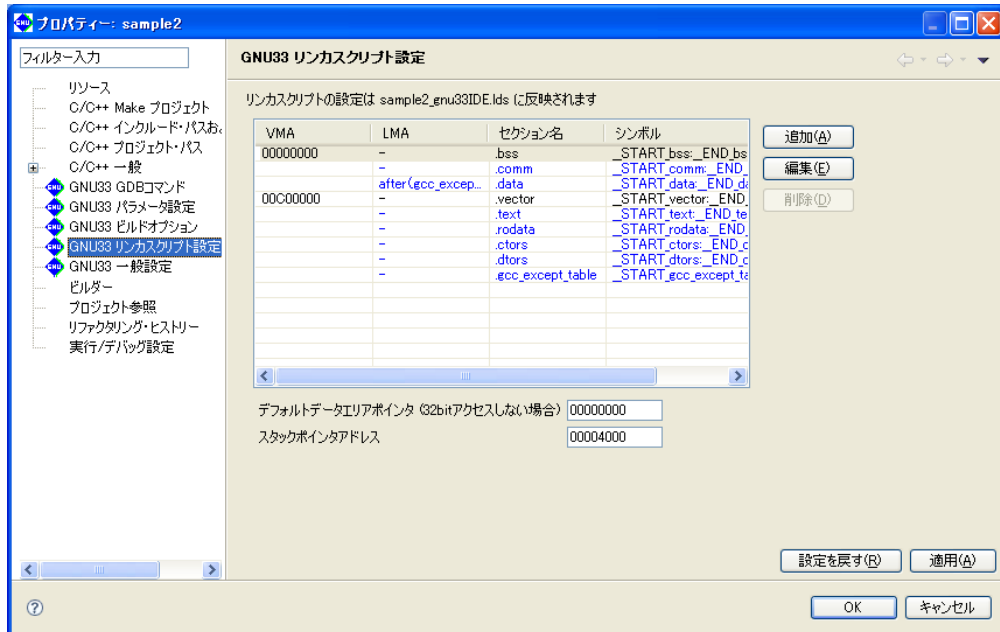
ソースファイルの構成例

boot.s (ベクタテーブルとスタックの初期化等)
main.c (メインおよびその他の関数)

プログラムはこの2つのソースで構成されるものとします。

それぞれのセクションの配置アドレスを個々に指定する場合を除き、リンク時の配置はファイル名のアルファベット順になります。この場合はboot.s、main.cの順に配置されます。boot.sの先頭にあるベクタテーブル(.rodataセクション)がROMの先頭(0xc00000～)に置かれます。

セクションの構成 ([GNU33 リンカスクリプト設定] ダイアログ上の設定内容)



これはデフォルトの設定内容です。

VMA(実行アドレス)は、.bssセクションがアドレス0x00000000(RAMの先頭)に設定され、.vectorセクション(boot.oの.rodataセクション)がアドレス0x00c00000の(ROMの先頭)に設定されています。その他のセクションは、この2つのセクションに続いて配置されています。.dataを除くセクションは格納位置で使用するため、LMA(格納アドレス)は設定されていません。.dataはROMからRAMにコピーして使用するため、LMAを.gcc_except_tableセクションの後に設定しています。C++ソースを使用しないこの例の場合、.ctors、.dtors、.gcc_except_tableセクションは実体がありませんが、特に削除はしていません。

図5.7.7.2に示すメモリマップの構成が、このままで実現できます。

リンカスクリプトとセクションの配置

リンカスクリプトは次のように生成されます。

```
/* Linker Script file generated by Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

SECTIONS
{
    /* data pointer symbols */
    __dp = 0x00000000;

    /* stack pointer symbols */
    __START_stack = 0x00004000;

    /* location counter */
    . = 0x0;

    /* section information */
}
```

```

.bss 0x00000000 :
{
    __START_bss = . ;
    boot.o(.bss)
    main.o(.bss)
    C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
    C:/EPSON/gnu33/lib/std/libc.a(.bss)
    C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
}
__END_bss = . ;

.comm __END_bss :
{
    __START_comm = . ;
    boot.o(.comm)
    main.o(.comm)
    C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
    C:/EPSON/gnu33/lib/std/libc.a(.comm)
    C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
}
__END_comm = . ;

.data __END_comm : AT( __END_gcc_except_table )
{
    __START_data = . ;
    boot.o(.data)
    main.o(.data)
    C:/EPSON/gnu33/lib/std/libstdio.a(.data)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
    C:/EPSON/gnu33/lib/std/libc.a(.data)
    C:/EPSON/gnu33/lib/std/libgcc.a(.data)
}
__END_data = . ;

.vector 0x00C00000 :
{
    __START_vector = . ;
    boot.o(.rodata)
}
__END_vector = . ;

.text __END_vector :
{
    __START_text = . ;
    boot.o(.text)
    main.o(.text)
    C:/EPSON/gnu33/lib/std/libstdio.a(.text)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
    C:/EPSON/gnu33/lib/std/libc.a(.text)
    C:/EPSON/gnu33/lib/std/libgcc.a(.text)
}
__END_text = . ;

.rodata __END_text :
{
    __START_rodata = . ;
    main.o(.rodata)
    C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
    C:/EPSON/gnu33/lib/std/libc.a(.rodata)
    C:/EPSON/gnu33/lib/std/libgcc.a(.rodata)
}
__END_rodata = . ;

.ctors __END_rodata :

```

```

{
    . = ALIGN(4);
    __START_ctors = . ;
    boot.o(.ctors)
    main.o(.ctors)
    C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
    C:/EPSON/gnu33/lib/std/libc.a(.ctors)
    C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
}
__END_ctors = . ;

.dtors __END_ctors :
{
    . = ALIGN(4);
    __START_dtors = . ;
    boot.o(.dtors)
    main.o(.dtors)
    C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
    C:/EPSON/gnu33/lib/std/libc.a(.dtors)
    C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
}
__END_dtors = . ;

.gcc_except_table __END_dtors :
{
    __START_gcc_except_table = . ;
    boot.o(.gcc_except_table)
    main.o(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
}
__END_gcc_except_table = . ;

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
}

```

ファイル構成を含めたセクション配置は次のようになります。

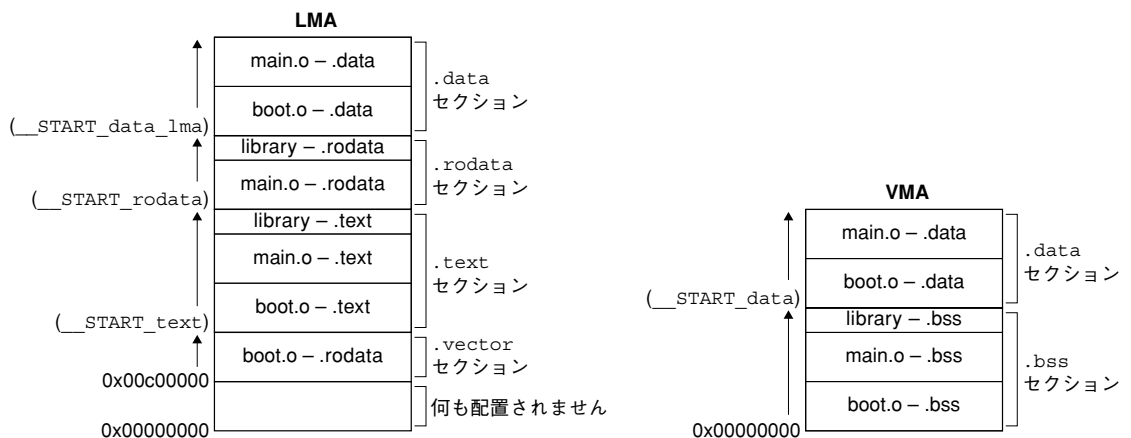


図5.7.7.3 セクション配置例1

例2. 基本レイアウトの変更

ここでは、例1のシステムに定数格納用のROMを追加し、.rodataセクションを配置する例を示します。

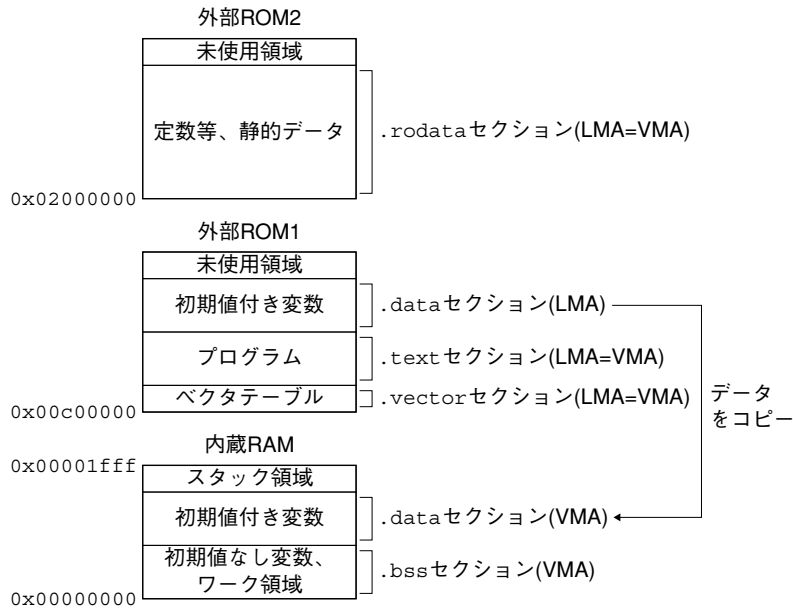


図5.7.7.4 メモリ構成例2

ソースファイルの構成例

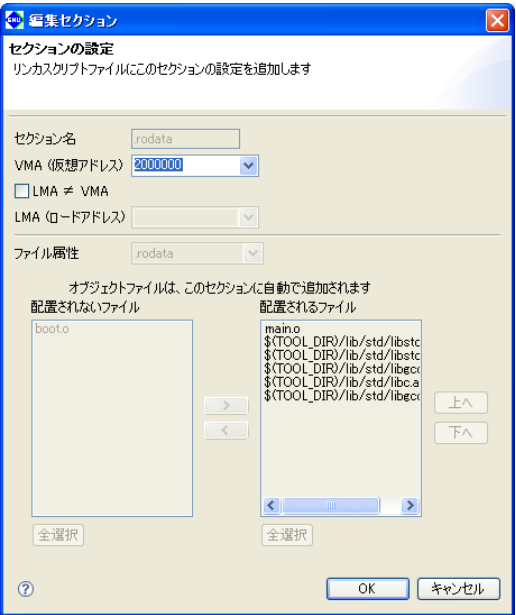
boot.s (ベクタテーブルとスタックの初期化等)
main.c (メインおよびその他の関数)

セクションの編集([編集セクション]ダイアログ上の設定内容)

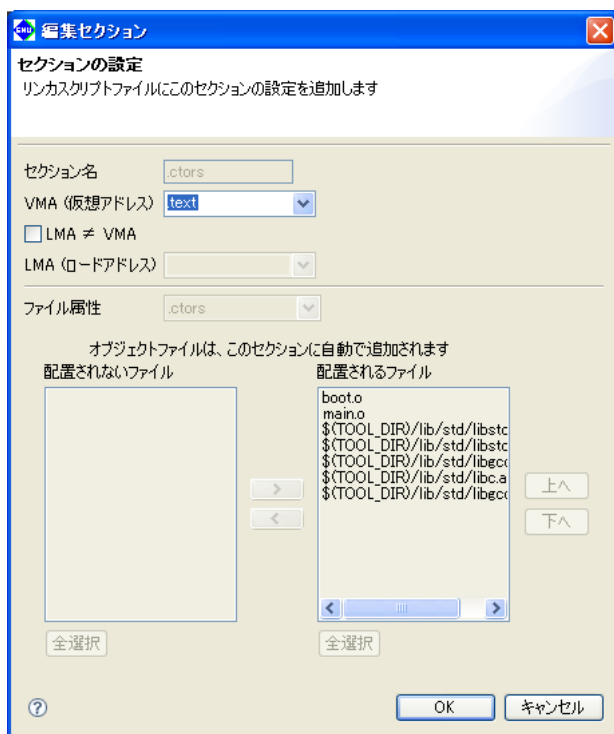
以下のように.rodataセクションと.ctorsセクション情報の修正を行います。その他のセクションはデフォルト設定のまま使用します。

1 .rodataセクションの修正

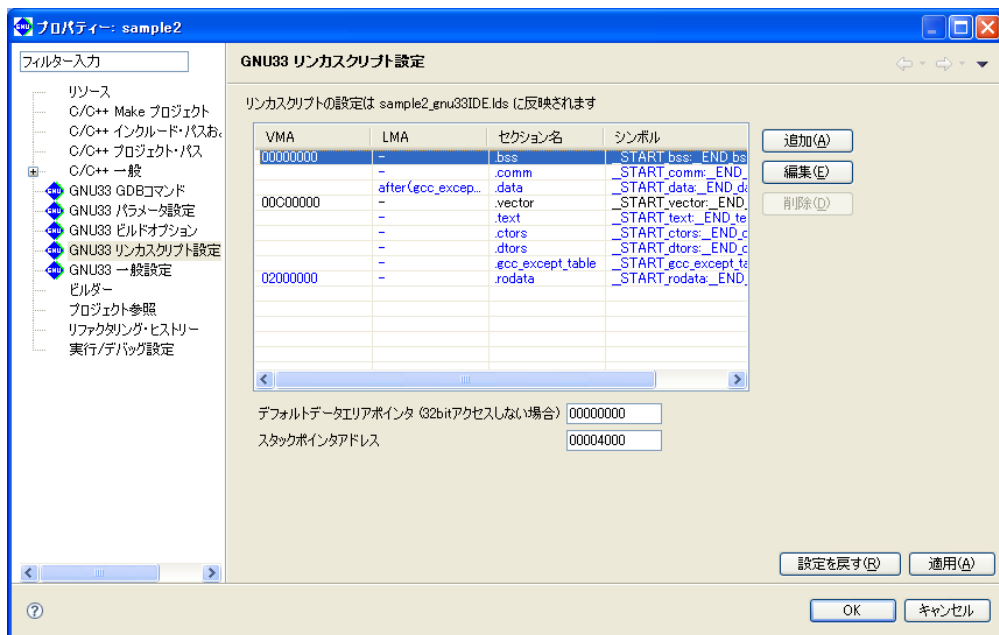
[VMA (仮想アドレス)]を0x02000000に修正します。



- 2 .ctorsセクションの修正
[VMA (仮想アドレス)]を".text"に変更します。



セクションの構成 ([GNU33 リンカスクリプト設定] ダイアログ上の設定内容)



リンカスクリプトとセクションの配置

リンカスクリプトは次のように生成されます。

```
/* Linker Script file generated by Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

SECTIONS
{
    /* data pointer symbols */
    __dp = 0x00000000;

    /* stack pointer symbols */
    __START_stack = 0x00004000;

    /* location counter */
    . = 0x0;

    /* section information */
    .bss 0x00000000 :
    {
        __START_bss = . ;
        boot.o(.bss)
        main.o(.bss)
        C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
        C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
        C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
        C:/EPSON/gnu33/lib/std/libc.a(.bss)
        C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
    }
    __END_bss = . ;

    .comm __END_bss :
    {
        __START_comm = . ;
        boot.o(.comm)
        main.o(.comm)
        C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
        C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
        C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
        C:/EPSON/gnu33/lib/std/libc.a(.comm)
        C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
    }
    __END_comm = . ;

    .data __END_comm : AT( __END_gcc_except_table )
    {
        __START_data = . ;
        boot.o(.data)
        main.o(.data)
        C:/EPSON/gnu33/lib/std/libstdio.a(.data)
        C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
        C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
        C:/EPSON/gnu33/lib/std/libc.a(.data)
        C:/EPSON/gnu33/lib/std/libgcc.a(.data)
    }
    __END_data = . ;

    .vector 0x00C00000 :
    {
        __START_vector = . ;
        boot.o(.rodata)
    }
    __END_vector = . ;

    .text __END_vector :
    {
        __START_text = . ;
        boot.o(.text)
        main.o(.text)
        C:/EPSON/gnu33/lib/std/libstdio.a(.text)
```



```

C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
C:/EPSON/gnu33/lib/std/libc.a(.text)
C:/EPSON/gnu33/lib/std/libgcc.a(.text)
}
__END_text = . ;

.ctors __END_text :
{
    . = ALIGN(4);
    __START_ctors = . ;
    boot.o(.ctors)
    main.o(.ctors)
    C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
    C:/EPSON/gnu33/lib/std/libc.a(.ctors)
    C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
}
__END_ctors = . ;

.dtors __END_dtors :
{
    . = ALIGN(4);
    __START_dtors = . ;
    boot.o(.dtors)
    main.o(.dtors)
    C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
    C:/EPSON/gnu33/lib/std/libc.a(.dtors)
    C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
}
__END_dtors = . ;

.gcc_except_table __END_dtors :
{
    __START_gcc_except_table = . ;
    boot.o(.gcc_except_table)
    main.o(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
}
__END_gcc_except_table = . ;

.rodata 0x02000000 :
{
    __START_rodata = . ;
    main.o(.rodata)
    C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
    C:/EPSON/gnu33/lib/std/libc.a(.rodata)
    C:/EPSON/gnu33/lib/std/libgcc.a(.rodata)
}
__END_rodata = . ;

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
}

```

ファイル構成を含めたセクション配置は次のようになります。

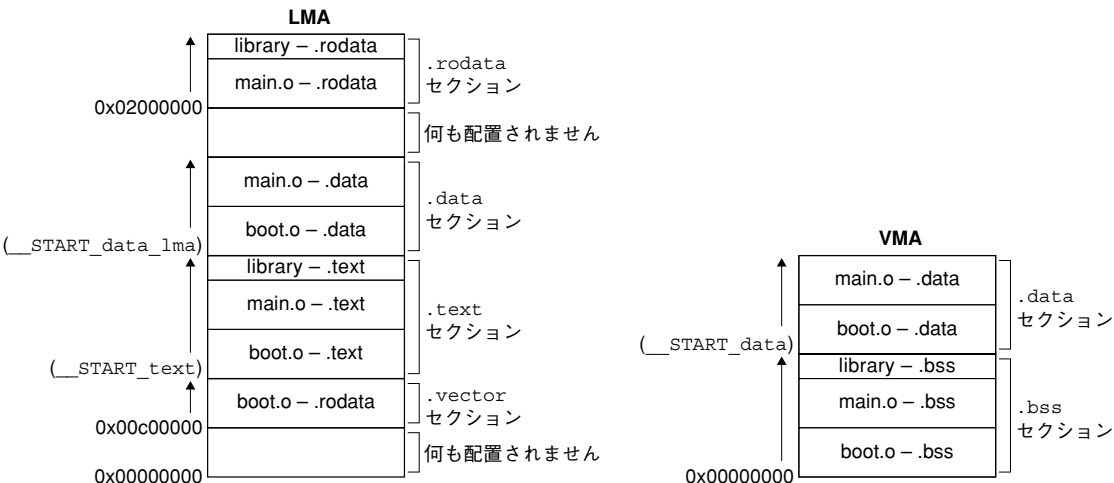


図5.7.7.5 セクション配置例2

例3. RAMエリアを複数の変数で共用

ここでは、例1と同じメモリ構成のシステムで、複数の変数に同じRAM領域を割り当てる例を示します。

図5.7.7.6のように複数のセクションを同じアドレスに割り付け、複数の変数で同じデータを共有したり、データを入れ替えて使用します。これによりメモリを節約することができます。

ただし、同じデータ領域を共有するセクションは.bss属性でなければなりません。初期値付きの変数(.dataセクション)領域の共有はできません。

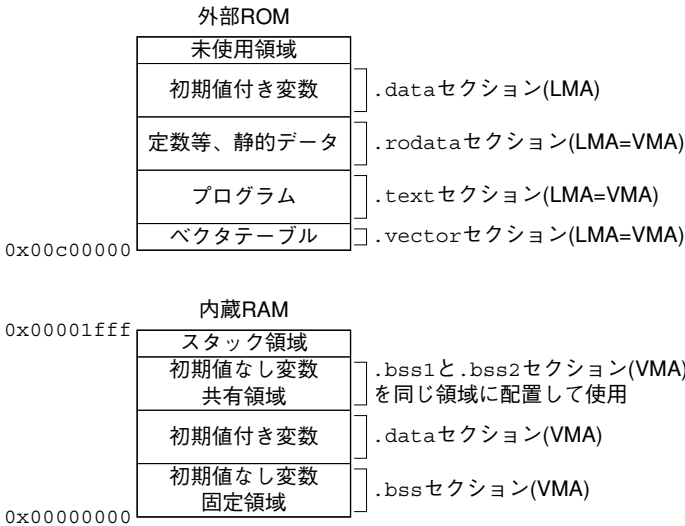


図5.7.7.6 データ領域の共有

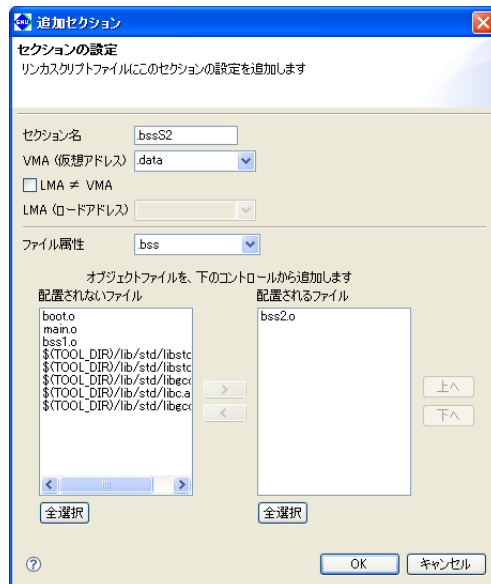
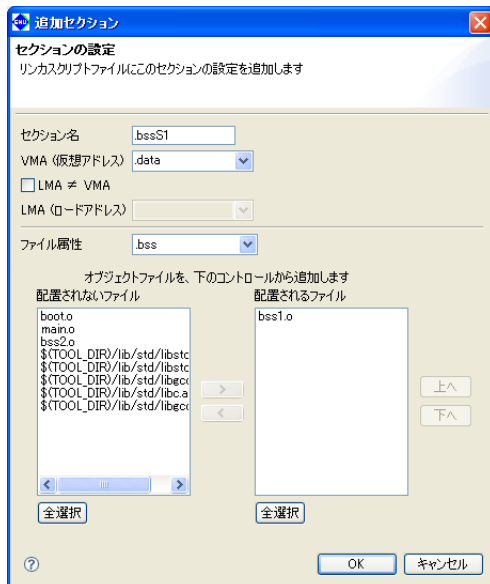
ソースファイルの構成例

- boot.s (ベクタテーブルとスタックの初期化等)
- main.c (メインおよびその他の関数)
- bss1.c (グローバル変数の定義ファイル1)
- bss2.c (グローバル変数の定義ファイル2)

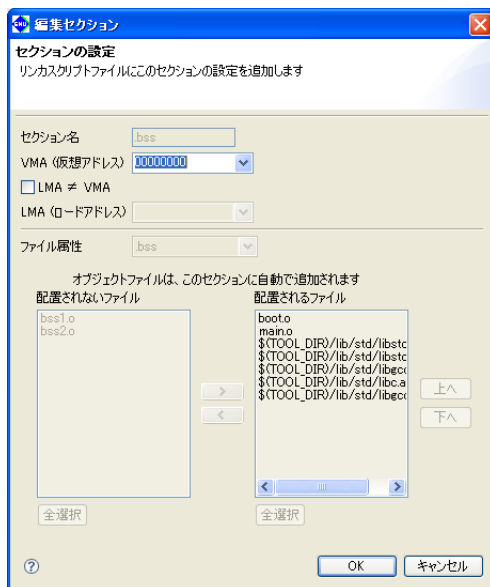
bss1.cとbss2.cには領域を共有する初期値を持たないグローバル変数のみが定義されているものとします。これらのファイルに関数や初期値を持つ変数、あるいは定数が含まれている場合、この例の設定では、.text、.data、.rodataセクションに配置されます。

セクションの編集 ([追加/編集セクション]ダイアログ上の設定内容)

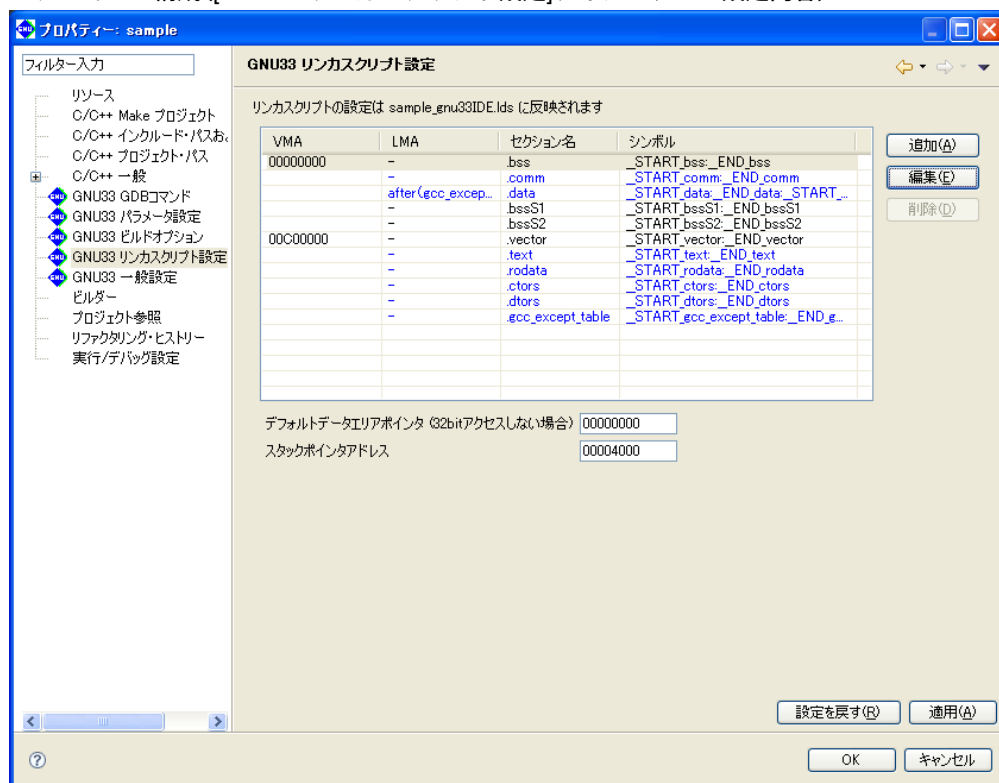
RAM領域を共有する .bssS1、.bssS2セクションを新規に作成します。どちらも .bss属性で .dataセクション (VMA) の直後に配置します。 .bssS1セクションには bss1.o のみを、 .bssS2セクションには bss2.o のみをファイルとして選択します。



.bssセクション情報は自動更新が設定されていますが、bss1.oとbss2.oは上記のセクションで指定されたため、配置するファイル一覧には含まれません。



セクションの構成 ([GNU33 リンカスクリプト設定] ダイアログ上の設定内容)



リンカスクリプトとセクションの配置

リンカスクリプトは次のように生成されます。

```
/* Linker Script file generated by Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);
```

```
SECTIONS
{
  /* data pointer symbols */
  __dp = 0x00000000;

  /* stack pointer symbols */
  __START_stack = 0x00004000;

  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x00000000 :
  {
    __START_bss = . ;
    boot.o(.bss)
    main.o(.bss)
    C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
    C:/EPSON/gnu33/lib/std/libc.a(.bss)
    C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
  }
  __END_bss = . ;

  .comm __END_bss :
  {
    __START_comm = . ;
    boot.o(.comm)
```

```

    bss1.o(.comm)
    bss2.o(.comm)
    main.o(.comm)
    C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
    C:/EPSON/gnu33/lib/std/libc.a(.comm)
    C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
}
__END_comm = . ;

.data __END_comm : AT( __END_gcc_except_table )
{
    __START_data = . ;
    boot.o(.data)
    bss1.o(.data)
    bss2.o(.data)
    main.o(.data)
    C:/EPSON/gnu33/lib/std/libstdio.a(.data)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
    C:/EPSON/gnu33/lib/std/libc.a(.data)
    C:/EPSON/gnu33/lib/std/libgcc.a(.data)
}
__END_data = . ;

.bssS1 __END_data :
{
    __START_bssS1 = . ;
    bss1.o(.bss)
}
__END_bssS1 = . ;

.bssS2 __END_data :
{
    __START_bssS2 = . ;
    bss2.o(.bss)
}
__END_bssS2 = . ;

.vector 0x00C00000 :
{
    __START_vector = . ;
    boot.o(.rodata)
}
__END_vector = . ;

.text __END_vector :
{
    __START_text = . ;
    boot.o(.text)
    bss1.o(.text)
    bss2.o(.text)
    main.o(.text)
    C:/EPSON/gnu33/lib/std/libstdio.a(.text)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
    C:/EPSON/gnu33/lib/std/libc.a(.text)
    C:/EPSON/gnu33/lib/std/libgcc.a(.text)
}
__END_text = . ;

.rodata __END_text :
{
    __START_rodata = . ;
    bss1.o(.rodata)
    bss2.o(.rodata)
    main.o(.rodata)
    C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
    C:/EPSON/gnu33/lib/std/libc.a(.rodata)
}

```

```

    C:/EPSON/gnu33/lib/std/libgcc.a(.rodata)
}
__END_rodata = . ;

.ctors __END_rodata :
{
    . = ALIGN(4);
    __START_ctors = . ;
    boot.o(.ctors)
    bss1.o(.ctors)
    bss2.o(.ctors)
    main.o(.ctors)
    C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
    C:/EPSON/gnu33/lib/std/libc.a(.ctors)
    C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
}
__END_ctors = . ;

.dtors __END_ctors :
{
    . = ALIGN(4);
    __START_dtors = . ;
    boot.o(.dtors)
    bss1.o(.dtors)
    bss2.o(.dtors)
    main.o(.dtors)
    C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
    C:/EPSON/gnu33/lib/std/libc.a(.dtors)
    C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
}
__END_dtors = . ;

.gcc_except_table __END_dtors :
{
    __START_gcc_except_table = . ;
    boot.o(.gcc_except_table)
    bss1.o(.gcc_except_table)
    bss2.o(.gcc_except_table)
    main.o(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
}
__END_gcc_except_table = . ;

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
}

```

ファイル構成を含めたセクション配置は次のようになります。

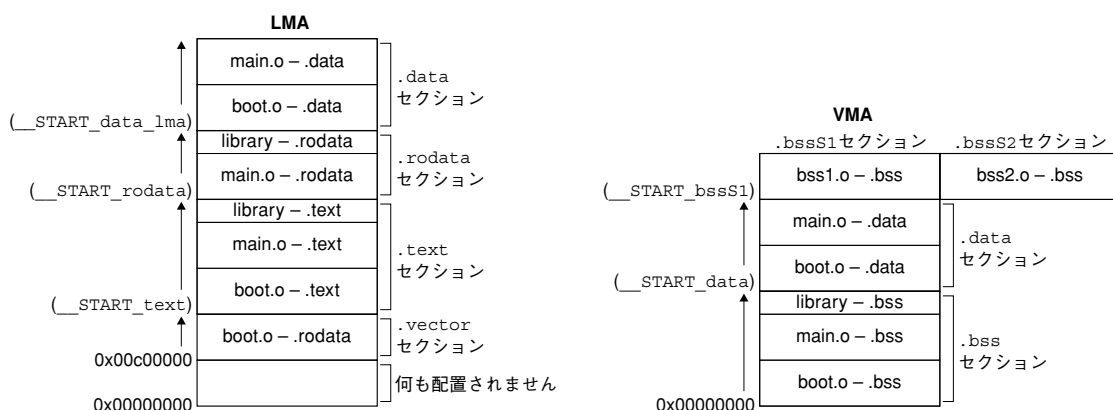


図5.7.7.7 セクション配置例3

.bssS1と.bssS2セクションは並列の配置になります。どちらのセクションとして使用するか、およびそのデータについてはアプリケーションプログラムが管理します。

例4. RAM上でプログラムを実行

高速な処理が必要なルーチンは、ノーウェイトでアクセス可能な内蔵RAMで実行させることで対応します。ここまでの例では、.textセクションのVMAのみを指定し、ROM上で実行させていました。.dataセクションと同様にVMAとLMAを指定し、実行前にRAMにコピーすることでRAM上での実行が実現できます。また、例3と同様に複数のセクションを同じ領域に割り当てて、プログラムを必要に応じて入れ替えながら実行させることもできます。

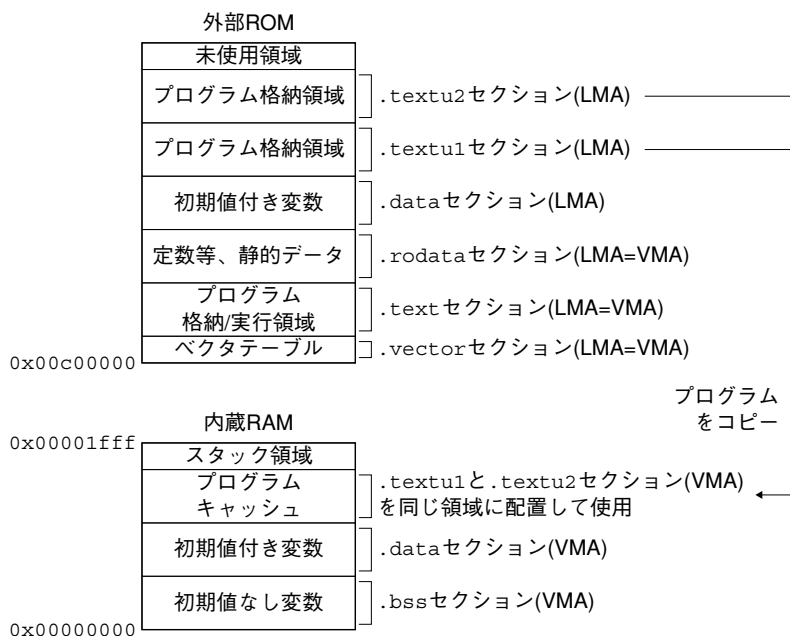


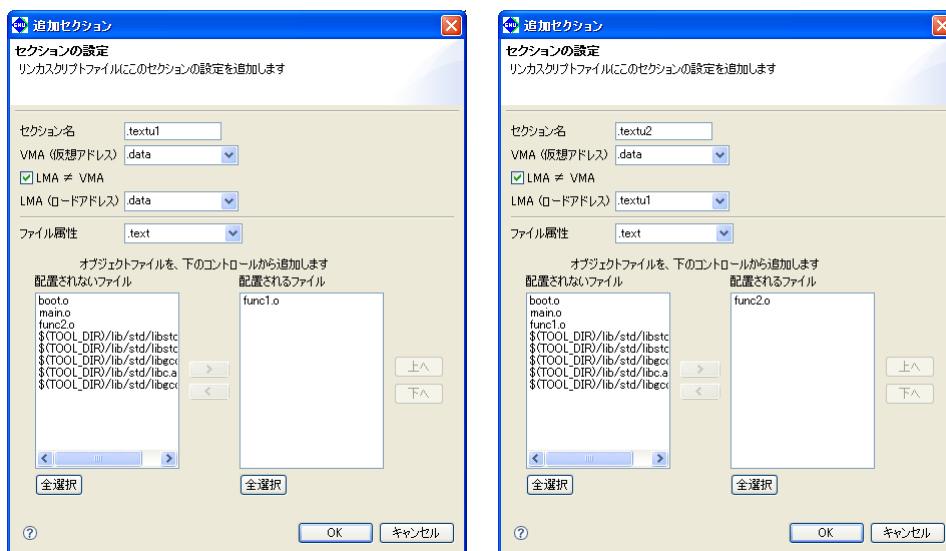
図5.7.7.8 RAM上にプログラム領域を確保

ソースファイルの構成例

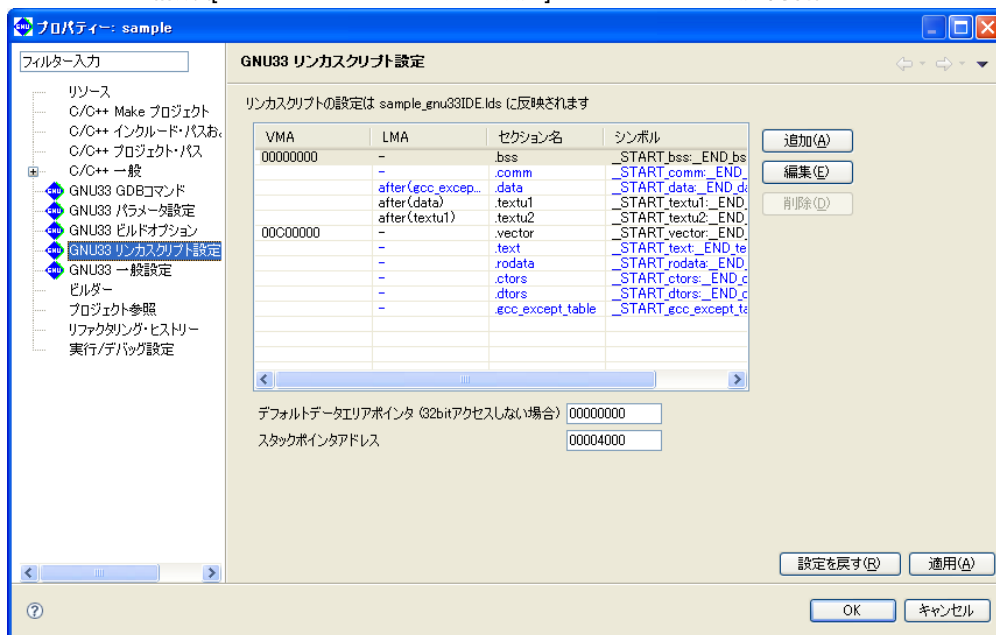
boot.s (ベクタテーブルとスタックの初期化等)
 main.c (メインおよびその他の関数)
 func1.c (RAM上で実行するプログラム1)
 func2.c (RAM上で実行するプログラム2)

セクションの編集 ([追加セクション]ダイアログ上の設定内容)

.textu1、.textu2セクションを、以下のとおり新規作成します。



セクションの構成 ([GNU33 リンカスクリプト設定]ダイアログ上の設定内容)



リンカスクリプトとセクションの配置

リンカスクリプトは次のように生成されます。

```
/* Linker Script file generated by Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);
```

```
SECTIONS
{
    /* data pointer symbols */
    __dp = 0x00000000;

    /* stack pointer symbols */
    __START_stack = 0x00004000;
```



```

/* location counter */
. = 0x0;

/* section information */
.bss 0x00000000 :
{
    __START_bss = . ;
    boot.o(.bss)
    func1.o(.bss)
    func2.o(.bss)
    main.o(.bss)
    C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
    C:/EPSON/gnu33/lib/std/libc.a(.bss)
    C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
}
__END_bss = . ;

.comm __END_bss :
{
    __START_comm = . ;
    boot.o(.comm)
    func1.o(.comm)
    func2.o(.comm)
    main.o(.comm)
    C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
    C:/EPSON/gnu33/lib/std/libc.a(.comm)
    C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
}
__END_comm = . ;

.data __END_comm : AT( __END_gcc_except_table )
{
    __START_data = . ;
    boot.o(.data)
    func1.o(.data)
    func2.o(.data)
    main.o(.data)
    C:/EPSON/gnu33/lib/std/libstdio.a(.data)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
    C:/EPSON/gnu33/lib/std/libc.a(.data)
    C:/EPSON/gnu33/lib/std/libgcc.a(.data)
}
__END_data = . ;

.textul __END_data : AT( __START_data_lma + SIZEOF( .data ) )
{
    __START_textul = . ;
    func1.o(.text)
}
__END_textul = . ;

.textu2 __END_data : AT( __START_textul_lma + SIZEOF( .textul ) )
{
    __START_textu2 = . ;
    func2.o(.text)
}
__END_textu2 = . ;

.vector 0x00C00000 :
{
    __START_vector = . ;
    boot.o(.rodata)
}
__END_vector = . ;

.text __END_vector :

```

```

{
    __START_text = . ;
    boot.o(.text)
    main.o(.text)
    C:/EPSON/gnu33/lib/std/libstdio.a(.text)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
    C:/EPSON/gnu33/lib/std/libc.a(.text)
    C:/EPSON/gnu33/lib/std/libgcc.a(.text)
}
__END_text = . ;

.rodata __END_text :
{
    __START_rodata = . ;
    func1.o(.rodata)
    func2.o(.rodata)
    main.o(.rodata)
    C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
    C:/EPSON/gnu33/lib/std/libc.a(.rodata)
    C:/EPSON/gnu33/lib/std/libgcc.a(.rodata)
}
__END_rodata = . ;

.ctors __END_ctors :
{
    . = ALIGN(4);
    __START_ctors = . ;
    boot.o(.ctors)
    func1.o(.ctors)
    func2.o(.ctors)
    main.o(.ctors)
    C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
    C:/EPSON/gnu33/lib/std/libc.a(.ctors)
    C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
}
__END_ctors = . ;

.dtors __END_dtors :
{
    . = ALIGN(4);
    __START_dtors = . ;
    boot.o(.dtors)
    func1.o(.dtors)
    func2.o(.dtors)
    main.o(.dtors)
    C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
    C:/EPSON/gnu33/lib/std/libc.a(.dtors)
    C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
}
__END_dtors = . ;

.gcc_except_table __END_dtors :
{
    __START_gcc_except_table = . ;
    boot.o(.gcc_except_table)
    func1.o(.gcc_except_table)
    func2.o(.gcc_except_table)
    main.o(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
    C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
}
__END_gcc_except_table = . ;

```

```

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
__START_textu1_lma = LOADADDR( .textu1 );
__END_textu1_lma = __START_textu1_lma + SIZEOF( .textu1 );
__START_textu2_lma = LOADADDR( .textu2 );
__END_textu2_lma = __START_textu2_lma + SIZEOF( .textu2 );
}

```

ファイル構成を含めたセクション配置は次のようになります。

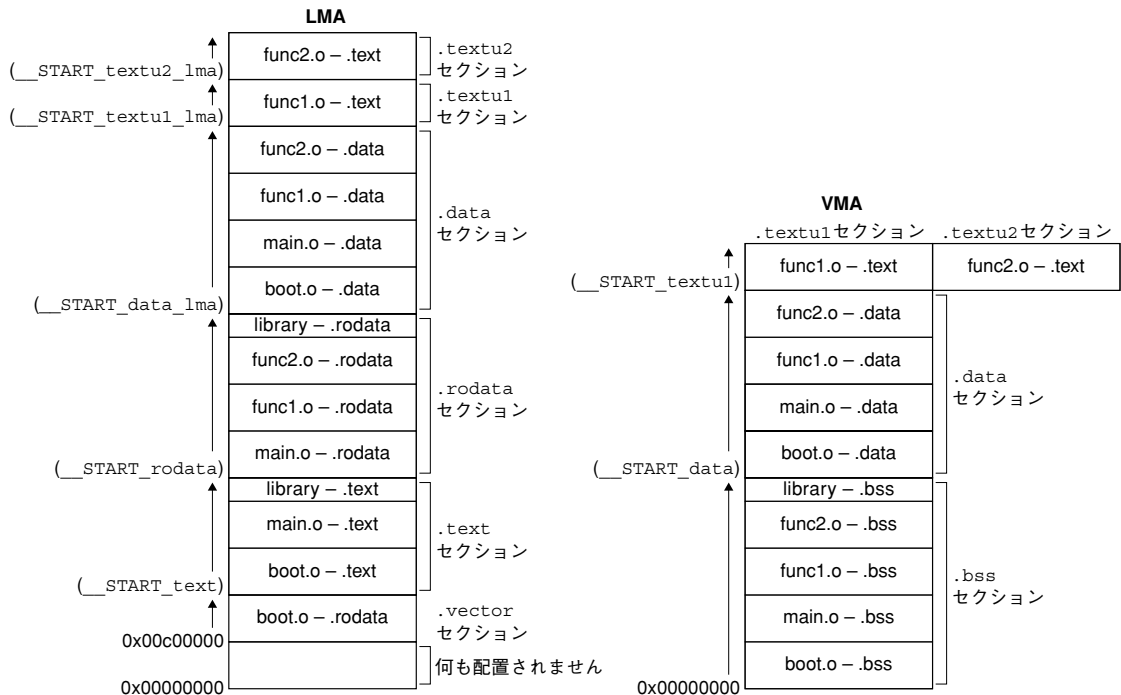


図5.7.7.9 セクション配置例4

ROMからRAMへのプログラム転送ルーチンは、main.c内に作成してください。

注: .textu1、.textu2セクションを、たとえばLMAの指定されていない.rodataセクションの前に配置すると、.rodataセクションは.textu2セクションのVMAの終了アドレスの後ろに配置されてしまいます。したがって、LMA ≠ VMAのセクションの後ろにVMAのみ指定のセクションを配置しないでください。

5.7.8 ビルドの実行

ソースファイルの作成、ビルドオプションの設定、リンカスクリプトの編集が終わると、ビルドを実行できます。

ビルドの実行方法を以下に示します。

●ワークスペース内の全プロジェクトのビルド

ワークスペース内のすべてのプロジェクトをビルドするには、以下の操作のいずれかを行います。

- [プロジェクト]メニューから[すべてビルド]を選択します。
- ウィンドウツールバーの[すべてビルド]ボタンをクリックします。

●選択したプロジェクトのビルド

プロジェクト別のビルドは次のように行います。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ビルドするプロジェクトを選択します。
- (2) 以下のいずれかの方法でビルドを実行します。
 - [プロジェクト]メニューから[プロジェクトのビルド]を選択します。
 - [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[プロジェクトのビルド]を選択します。

[プロジェクト]メニューの[ワーキング・セットのビルド]からワーキングセットを選択し、セットに含まれるプロジェクトのみをビルドすることも可能です。

●ビルド処理

ビルドを開始すると、IDEは次のような処理を行います。

1. エディタ上に保存されていないファイルがある場合は保存します。
2. プロジェクトプロパティの設定に従い、以下のファイルを生成します。
 - makeファイル(<プロジェクト名>_gnu33IDE.mak)と依存関係ファイル(<ソース名>.d)
 - リンカスクリプトファイル(<プロジェクト名>_gnu33IDE.lds)
 - パラメータファイル(<プロジェクト名>_gnu33IDE.par)*
 - コマンドファイル(<プロジェクト名>_gnu33IDE.cmd)*

* これらのファイルはデバッグに必要なファイルで、ビルドには影響を与えません。また、コマンドファイルは、通常ビルド時には生成されません。<プロジェクト名>_gnu33IDE.cmdが存在しない場合のみ、起動に必要な最低限のコマンドでファイルが生成されます。
3. **make.exe**を実行します。以下のファイルが生成されます。
 - ソースごとのオブジェクトファイル(<ソース名>.o)
 - 実行形式オブジェクトファイル(<プロジェクト名>.elf)
 - リンクマップファイル(<プロジェクト名>.map)
 - ダンプファイル(<プロジェクト名>.dump)

ビルド実行中は、各ツールを実行するコマンドラインが[コンソール]ビューに表示されます。

ビルド中に発生したエラーは[問題]ビューで確認でき、そこからエディタ上のエラー発生箇所にジャンプすることができます。この機能については5.5.3節内の"●エラー発生行へのジャンプ"を参照してください。

make.exeはmakeファイルに記述されたソースファイルとオブジェクトファイル(*.o)の依存関係リストから、生成されていないか、あるいは更新が必要なオブジェクトファイル(*.o)のみを生成してリンクするようになっています。

したがって、最初のビルドではすべてのソースをコンパイル/アセンブルしてオブジェクトファイル(*.o)を生成し、リンクします。それ以降は変更されたソース(インクルードファイルの変更も含む)のみをコンパイル/アセンブルしてリンクを行います。

上記の2に示したファイルは、ビルド前にエディタ等で内容を直接修正しても、ビルド開始時には上書きされてしまいます。ビルド時に再生成されないようにするには、[プロパティ]ダイアログの[ビルダー]ページで選択されている[GNU33 ファイルビルダ]のチェックを外してください。

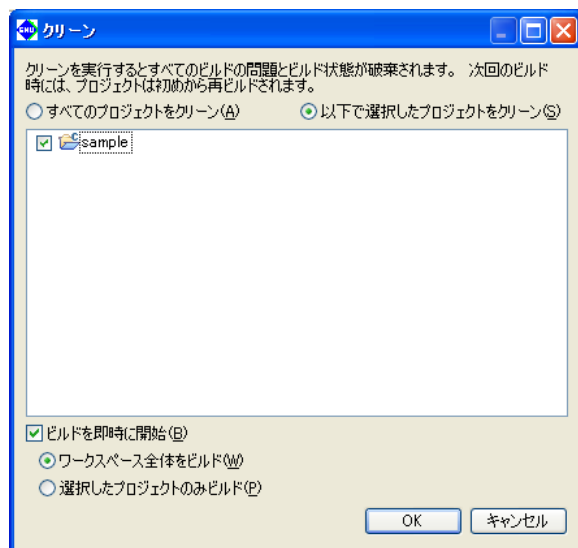
●リンクリソースから生成されるオブジェクトファイルについて

プロジェクト外部にリンクされたソースファイルのオブジェクトファイルは、プロジェクトフォルダ直下に生成されます。

5.7.9 クリーンとリビルド

前述のとおり、ソースまたはインクルードファイルが変更されていない場合はオブジェクトファイル(*.o)は再生成されません。生成されているオブジェクトファイル(*.o)をすべて消去すると、再度すべてのソースからのビルド(リビルド)を実行できますので、**IDE**が生成するmakeファイルには、生成されたファイルをすべて消去するコマンドが"clean"のターゲット名で記述されています。このクリーンを利用してリビルドを行う方法を以下に示します。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、リビルドするプロジェクトを選択します。
- (2) [プロジェクト]メニューから[クリーン...]を選択します。
[クリーン]ダイアログが表示されます。



- (3) [以下で選択したプロジェクトをクリーン]ラジオボタンを選択し、リストからクリーン(リビルド)するプロジェクトを選択します。
ワークスペース内のすべてのプロジェクトをリビルドする場合は、[すべてのプロジェクトをクリーン]を選択します。
[ビルドを即時に開始]を選択した状態にしておきます。これで、クリーンを実行後、他の操作なしにビルドに移行できます。
[ワークスペース全体をビルド]を選択すると、ワークスペース内のすべてのプロジェクトをビルドします。
[選択したプロジェクトのみビルド]を選択すると、上で選択中のプロジェクトのみをビルドします。
- (4) [OK]ボタンをクリックします。

選択したプロジェクト内の生成されたオブジェクトファイルがすべて削除され、引き続いてビルドを実行します。

[ビルドを即時に開始]を非選択にした場合はファイルの削除だけで終了しますので、ビルドは前節の手順で実行する必要があります。

上記の他に、次の方法でもクリーンを実行できます。

- (1) [C/C++ プロジェクト]ビューで、クリーンするプロジェクトを選択します。
- (2) [C/C++ プロジェクト]ビューのコンテキストメニューから[プロジェクトをクリーンする]を選択します。

この方法の場合、ダイアログは表示されずにクリーンのみが実行されます。

意図的にリビルドを行う以外、通常はソースファイルまたはヘッダファイルを変更してもビルドを行うだけで済みますが、以下の場合にはリビルドが必要になります。

- ヘッダファイルをプロジェクトから削除した場合
リビルドを行うことで依存関係ファイルを再作成します。
- 依存関係ファイルを削除してしまった場合
依存関係ファイルは削除しないでください。
- インクルードしているヘッダファイルがプロジェクト内でない場合
プロジェクトフォルダの下にない(プロジェクトの外部にある)ヘッダファイルを変更した場合は、リビルドを行ってください。
- プロジェクトをインポート直後にビルドしたとき、ファイル権限の影響によりビルドできない場合。

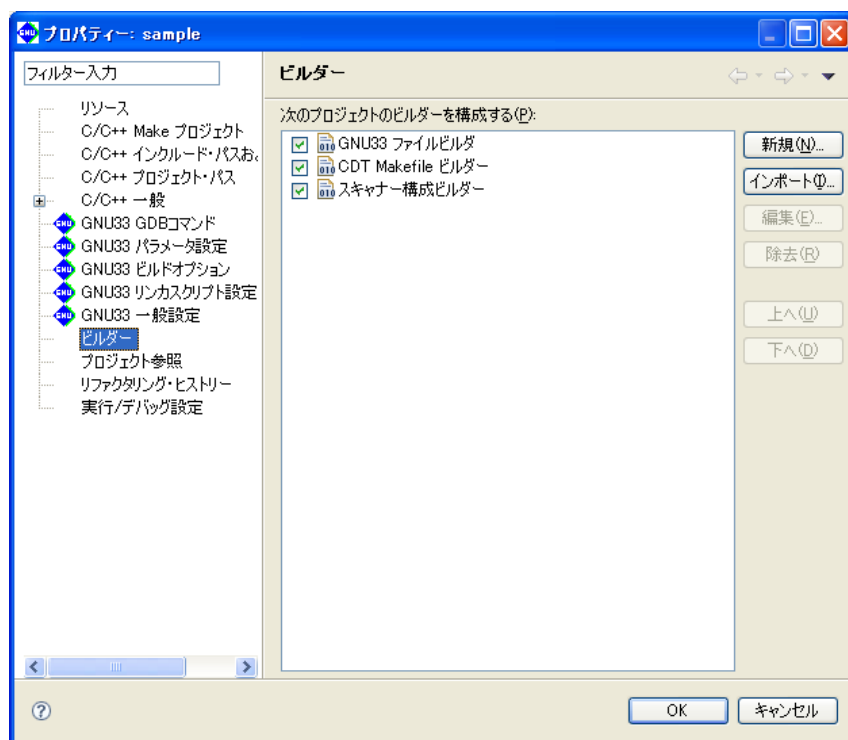
5.7.10 ユーザ作成のmakeファイルを使用するには

IDEが自動作成するmakeファイルを使用せずに、ユーザが独自に作成したmakeファイルを使用してビルドを行うことができます。その手順を以下に示します。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ユーザが作成したmakeファイルでビルドするプロジェクトを選択します。
- (2) makeファイルをIDE上で新規作成するか、作成済みのファイルをインポートします。("5.4.8 プロジェクト内のリソース操作"参照)
すでにIDEでビルドを行っている場合は、IDEが生成したmakeファイルを修正して使用することもできます。
- (3) [プロジェクト]メニューまたは選択したプロジェクトのコンテキストメニューから[プロパティ]を選択します。
[プロパティ]ダイアログが表示されます。

makeファイルを"<プロジェクト名>_gnu33IDE.mak"の名称で作成した場合や、IDEが生成したmakeファイルを修正した場合は、次の(4)と(5)の操作が必要です。別名のmakeファイルを使用し、リンカスクリプトファイル(.lds)、パラメータファイル(.par)、コマンドファイル(.cmd)についてはIDEが生成するファイルを使用する場合は(4)と(5)を省略し、(6)に進みます。

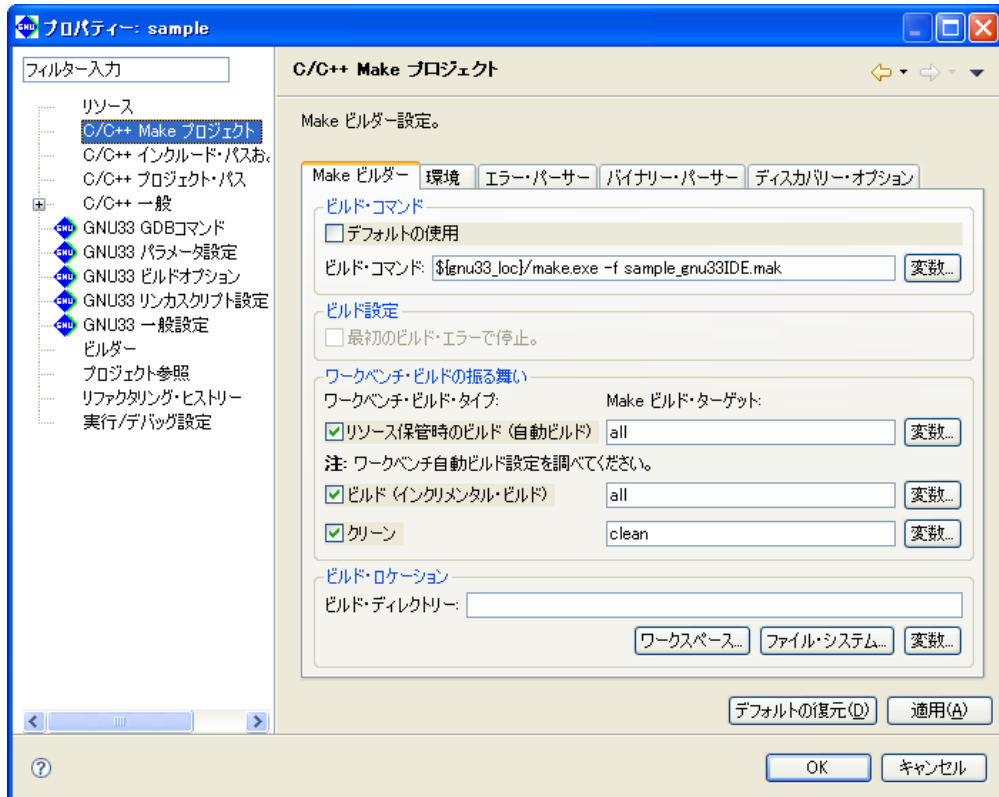
- (4) プロパティの一覧から[ビルダー]を選択します。



- (5) [GNU33 ファイルビルダ]のチェックを外します。
これによって、ビルド時に以下のファイルが生成されなくなります。
 - makeファイル(<プロジェクト名>_gnu33IDE.mak)
 - リンカスクリプトファイル(<プロジェクト名>_gnu33IDE.lds)
 - パラメータファイル(<プロジェクト名>_gnu33IDE.par)
 - コマンドファイル(<プロジェクト名>_gnu33IDE.cmd)

[GNU33 ファイルビルダ]が選択されていると、ビルドを実行するたびにこれらのファイルが**IDE**によって上書きされます。makeファイル以外は**IDE**が生成するファイルを使用したい場合は、makeファイルを上記以外の名称で作成して[GNU33 ファイルビルダ]を選択しておきます。または、[GNU33 ファイルビルダ]を選択してビルドを実行し、あらかじめ上記ファイルを生成した後に[GNU33 ファイルビルダ]のチェックを外します。

- (6) プロパティの一覧から[C/C++ Make プロジェクト]を選択し、[Make ビルダー]タブのページを表示させます。



- (7) [デフォルトの使用]のチェックを外し、[ビルド・コマンド]のコマンドラインを修正します。

例: user.makというmakeファイルに変更

```
$(gnu33_loc)/make.exe -f user.mak
```

- (8) 以下のターゲット名をユーザ作成の名称に変更します。チェックボックスは選択したままにしておきます。

[ビルド (インクリメンタル・ビルド)]

ビルド実行時に呼び出すmakeファイル内のターゲットを指定します。デフォルト設定では"all"を呼び出します。

[クリーン]

クリーン実行時に呼び出すmakeファイル内のターゲットを指定します。デフォルト設定では"clean"を呼び出します。

ユーザのmakeファイルでも同じターゲット名を使用している場合は、変更の必要はありません。

[リソース保管時のビルド (自動ビルド)]はデフォルト設定のままにしておいてください(**IDE**のデフォルト設定では使用しません)。

- (9) [OK]ボタンをクリックして[プロパティ]ダイアログを終了します。

5.8 デバッガの起動

デバッガgdbはIDEとは別のプログラムとして用意されていますが、IDE上でコマンドオプション等を設定して起動することができます。

5.8.1 パラメータファイルの生成

パラメータファイルは、デバッガにターゲットシステムのメモリマップ情報を設定するためのファイルです。デバッガは、その設定に基づいて以下の処理を行います。

- ソフトウェアPCブレイクアドレスが正当なマップ領域内かどうかのチェック
- ROM領域への書き込みによるブレイク(シミュレータモードのみ)
- 未定義領域へのアクセス(シミュレータモードのみ)
- スタックのオーバーフロー(シミュレータモードのみ)
- トレース結果の取得(ICDモードのみ)

また、パラメータファイルを読み込んだ場合、ファイルに記述されているすべてのメモリ領域を合わせたサイズのメモリがパソコン上のメモリ内に確保されます。

パラメータファイルの詳細については、「10.8 パラメータファイル」を参照してください。

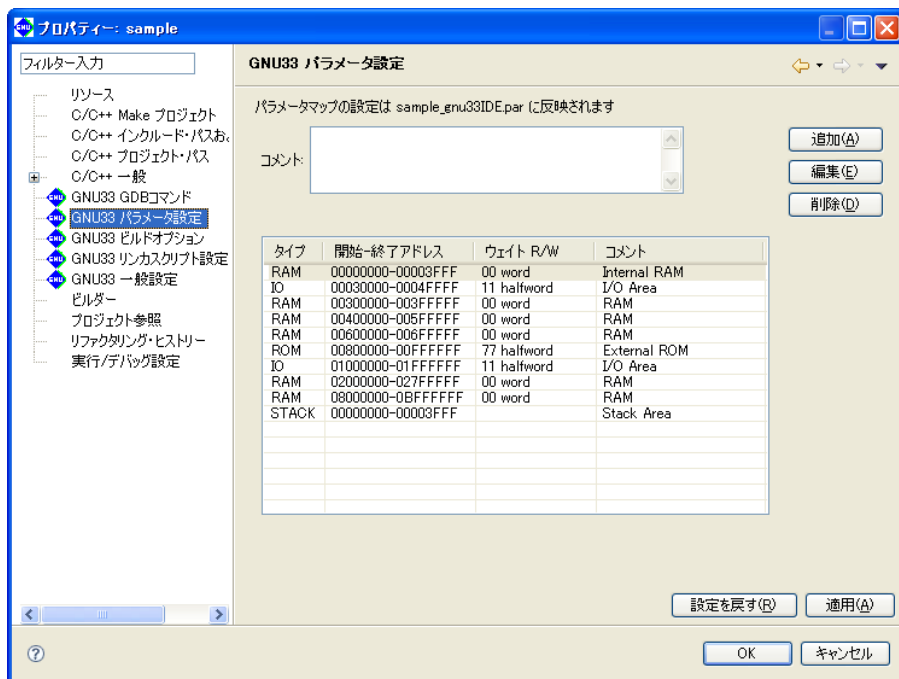
IDEはプロパティの設定に従い、ビルド時に"<プロジェクト名>_gnu33IDE.par"という名称のパラメータファイルを作成します。その後プロパティが変更された場合は、デバッガの起動時にファイルは更新され、デバッガには最新のパラメータファイルが渡されます。

以下、パラメータファイルの生成方法を説明します。

●パラメータ設定ページ

パラメータファイルの内容は、プロジェクトプロパティの[GNU33 パラメータ設定]ページで設定します。次の手順で設定ページを表示させます。

- (1)[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのプロジェクトを選択します。
- (2)[プロジェクト]メニューまたは選択したプロジェクトのコンテキストメニューから[プロパティ]を選択します。
[プロパティ]ダイアログが表示されます。
- (3)プロパティの一覧から[GNU33 パラメータ設定]を選択します。



このページには、現在の設定内容が表示されます。

[コメント:]

任意のコメントを入力することができます。最大255文字まで入力可能です。ここに入力したコメントはパラメータファイル内に記述されます。

エリア一覧

各行に1つのエリア情報が表示されます。一覧は、アドレス順に表示されます。スタックエリア情報のみ最後にまとめられます。

[タイプ]

エリアの種類(ROM、RAM、IO、STACK)が表示されます。

[開始-終了アドレス]

エリアの開始-終了アドレスが16進数で表示されます。

[ウェイト R/W]

先頭の2桁の数値は、前が読み出しサイクルのウェイト数、後が書き込みサイクルのウェイト数を表します。"byte"、"halfword"、"word"はエリアのアクセスサイズを示します。その後が空白の場合は、エリアがリトルエンディアンでアクセスされることを示しています。ビッグエンディアンに設定したエリアの場合、"Big"が表示されます。

[コメント]

各エリアの情報に入力されているコメントを表示します。コメントの先頭を示す"#"を入力する必要はありません。

プロジェクト作成時の初期設定内容は以下のとおりです。

CPUタイプに"S1C33STD"または"S1C33PE"選択時

エリア	開始-終了アドレス	ウェイト数(R/W)	アクセスサイズ
RAM	0x00000000-0x00003FFF	0/0	word
IO	0x00030000-0x0004FFFF	1/1	halfword
RAM	0x00300000-0x003FFFFF	0/0	word
RAM	0x00400000-0x004FFFFF	0/0	word
RAM	0x00600000-0x006FFFFF	0/0	word
ROM	0x00800000-0x008FFFFF	7/7	halfword
ROM	0x00C00000-0x00CFFFFF	7/7	halfword
IO	0x01000000-0x010FFFFF	1/1	halfword
RAM	0x02000000-0x020FFFFF	0/0	word
RAM	0x08000000-0x080FFFFF	0/0	word
STACK	0x00000000-0x00003FFF	—	—

CPUタイプに"S1C33401"選択時

エリア	開始-終了アドレス	ウェイト数(R/W)	アクセスサイズ
RAM	0x00000000-0x00007FFF	0/0	word
IO	0x00030000-0x0004FFFF	1/1	halfword
RAM	0x00200000-0x002FFFFF	0/0	word
RAM	0x00300000-0x003FFFFF	0/0	word
RAM	0x00600000-0x006FFFFF	0/0	word
RAM	0x01000000-0x010FFFFF	0/0	word
RAM	0x10000000-0x100FFFFF	0/0	word
ROM	0x20000000-0x200FFFFF	7/7	halfword
RAM	0x80000000-0x800FFFFF	0/0	word
STACK	0x00000000-0x00007FFF	—	—

パラメータファイルの内容

CPUタイプに"S1C33STD"選択時

Parameter file generated by Gnu33 Plug-in for Eclipse

```
RAM      00000000 00003FFF 00W      # Internal RAM
IO       00030000 0004FFFF 11H      # I/O Area
RAM      00300000 003FFFFF 00W      # RAM
RAM      00400000 004FFFFF 00W      # RAM
RAM      00600000 006FFFFF 00W      # RAM
ROM      00800000 008FFFFF 77H      # External ROM
ROM      00C00000 00CFFFFF 77H      # External ROM
IO       01000000 010FFFFF 11H      # I/O Area
RAM      02000000 020FFFFF 00W      # RAM
RAM      08000000 080FFFFF 00W      # RAM
STACK    00000000 00003FFF          # Stack Area
```

CPUタイプに"S1C33PE"選択時

Parameter file generated by Gnu33 Plug-in for Eclipse

C33_PE

```
RAM      00000000 00003FFF 00W      # Internal RAM
IO       00030000 0004FFFF 11H      # I/O Area
RAM      00300000 003FFFFF 00W      # RAM
RAM      00400000 004FFFFF 00W      # RAM
RAM      00600000 006FFFFF 00W      # RAM
ROM      00800000 008FFFFF 77H      # External ROM
ROM      00C00000 00CFFFFF 77H      # External ROM
IO       01000000 010FFFFF 11H      # I/O Area
RAM      02000000 020FFFFF 00W      # RAM
RAM      08000000 080FFFFF 00W      # RAM
STACK    00000000 00003FFF          # Stack Area
```

CPUタイプに"S1C33401"選択時

Parameter file generated by Gnu33 Plug-in for Eclipse

C33_ADVANCED

```
RAM      00000000 00007FFF 00W      # Internal RAM
IO       00030000 0004FFFF 11H      # I/O Area
RAM      00200000 002FFFFF 00W      # RAM
RAM      00300000 003FFFFF 00W      # RAM
RAM      00600000 006FFFFF 00W      # RAM
RAM      01000000 010FFFFF 00W      # RAM
RAM      10000000 100FFFFF 00W      # RAM
ROM      20000000 200FFFFF 77H      # External ROM
RAM      80000000 800FFFFF 00W      # RAM
STACK    00000000 00007FFF          # Stack Area
```

●エリアの編集

上記の各エリアの情報は、システムに合わせて変更可能です。その手順を以下に示します。

- (1) [GNU33 パラメータ設定]ページのエリア一覧から編集するエリアをクリックして選択します。
- (2) [編集]ボタンをクリックします。
[編集パラメータ]ダイアログが表示されます。
- (3) 下記の説明に従って必要な変更を行い、[OK]をクリックします。

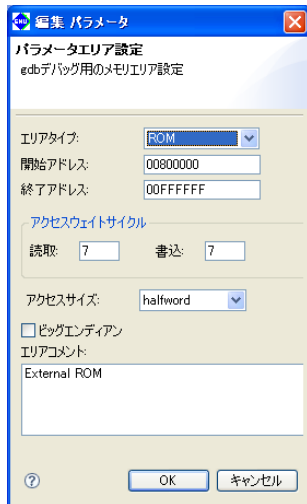
すでに設定されているエリアとアドレスが重複すると、ダイアログ上部に以下のようなエラーメッセージを表示します。

"アドレス範囲が他のエリアと重なっています"

この場合は、編集中のエリアのアドレスを修正するか、[キャンセル]で終了し、重複しているエリアの情報を修正してから再度このエリアの情報を編集してください。

- (4) 他のエリアまたはプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。
[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

[編集パラメータ]ダイアログ



[エリアタイプ:]

エリアの種類を次の4種類から選択します。

ROM 内蔵ROMエリア、あるいは外部ROMエリアを設定する場合に選択します。

RAM 内蔵RAMエリア、あるいは外部RAMエリアを設定する場合に選択します。

IO 内蔵のI/Oメモリエリア、あるいは外部デバイスをマップする場合に選択します。

STACK スタックエリアを設定する場合に選択します。

ROM、RAM、IOの設定は、ソフトウェアPCブレイクポイントが有効なアドレスかどうかを判断するためにデバッグで使用されます。また、シミュレータモード時は、設定したすべてのエリアを合わせたサイズのシミュレーションメモリがパソコン上のメモリ内に確保されます。これらのエリアはアドレスの重複が許されません。

STACKの設定は、シミュレータモード時にスタックオーバーフローを検出してプログラムの実行をブレイクさせるためのもので、それ以外のモードやスタックポインタ等には影響を与えません。これは物理的なメモリではありませんので、他のエリアとの重複が認められます。STACK選択時は、[アクセスウェイトサイクル]、[アクセスサイズ:]、[ビッグエンディアン]は無効となります。

[開始アドレス:]

エリアの開始アドレスを16進数(0xは不要)で入力します。終了アドレスより大きい場合、あるいは他の設定済みエリアのアドレスと重複している場合はエラーとなります。

[終了アドレス:]

エリアの終了アドレスを16進数(0xは不要)で入力します。開始アドレスより小さい場合、あるいは他の設定済みエリアのアドレスと重複している場合はエラーとなります。

[アクセスウェイトサイクル]

エリアをアクセスする際のウェイト数を設定します。0からF(16進数)で0から15サイクルを指定します。STACKエリアの場合は無効となります。

[読取:] 読み出しサイクルに挿入するウェイト数を入力します。

[書込:] 書き込みサイクルに挿入するウェイト数を入力します。

[アクセスサイズ:]

エリアのアクセスサイズを次の3種類から選択します。STACKエリアの場合は無効となります。

byte 8ビット

halfword 16ビット

word 32ビット

[ビッグエンディアン]

ビッグエンディアンでアクセスするエリアの場合に選択します。非選択のエリアはリトルエンディアンでアクセスされます。STACKエリアの場合は無効となります。

[エリアコメント:]

エリアの内容などをコメントとして入力します。コメントの先頭を示す"#"を入力する必要はありません。

[OK]

ダイアログを閉じ、[GNU33 パラメータ設定]ページのエリア一覧を設定した内容で更新します。必要な設定内容が入力されていない場合、このボタンは無効です。また、[OK]をクリック後には設定内容がチェックされ、不具合(他のエリアとアドレスが重複しているなど)がある場合はダイアログ上部にエラーメッセージを表示して、ダイアログを終了しません。編集中のエリアのエラー内容を修正するか、[キャンセル]で終了して他のエリアの修正を先に行う必要があります。

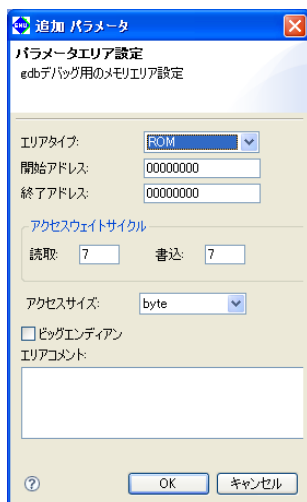
[キャンセル]

設定内容を破棄してダイアログを閉じます。[GNU33 パラメータ設定]ページのエリア一覧は変更されません。

●エリアの追加

新規のエリアは次のように追加します。

- (1) [追加]ボタンをクリックします。
[追加パラメータ]ダイアログが表示されます。



- (2) 前記の"エリアの編集"の説明に従って必要な設定を行い、[OK]をクリックします。
- (3) 他のエリアまたはプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。
[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

一覧内には、設定したアドレスに従って挿入されます。

●エリアの削除

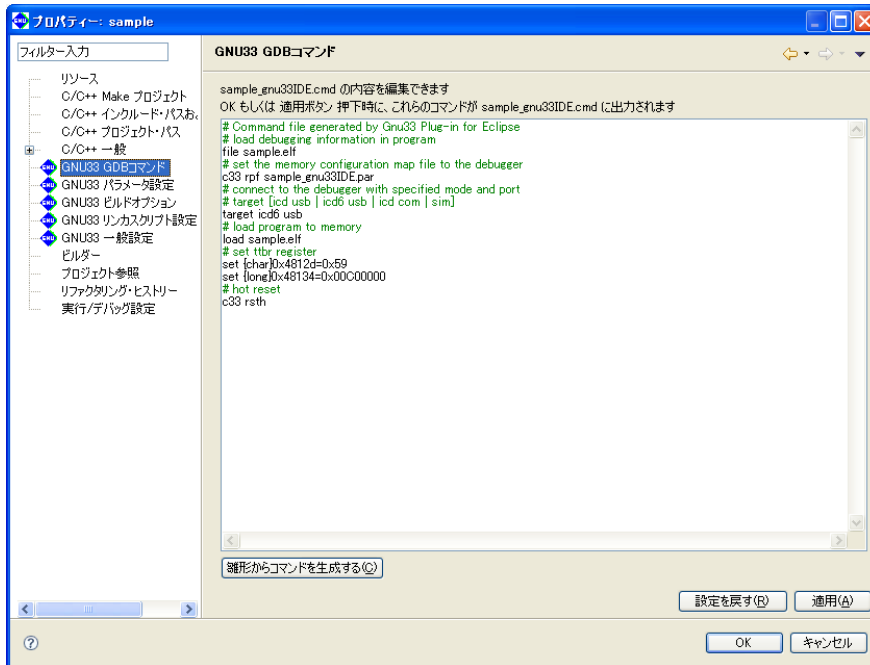
不要なエリアは次の手順で削除します。

- (1) [GNU33 パラメータ設定]ページのエリア一覧から削除するエリアの行をクリックして選択します。
- (2) [削除]ボタンをクリックします。
- (3) 操作を確認するダイアログが表示されますので、削除する場合は[OK]を、中止する場合は[キャンセル]をクリックします。
- (4) 他のエリアまたはプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。
[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

5.8.2 デバッグ起動コマンドの設定

デバッグの起動コマンドは、プロジェクトプロパティとして次のように設定しておくことができます。

- (1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのプロジェクトを選択します。
- (2) [プロジェクト]メニューまたは選択したプロジェクトのコンテキストメニューから[プロパティ]を選択します。
[プロパティ]ダイアログが表示されます。
- (3) プロパティの一覧から[GNU33 GDBコマンド]を選択します。



このページでデバッグ起動用コマンドファイルを直接編集可能です。編集領域には、<プロジェクト名>_gnu33IDE.cmdの内容が表示されます。このファイルが存在しない場合は、プロジェクトファイルに保存されている設定に基づいて下記のようなデフォルト設定のコマンドが表示されます。

- (4) コマンドの編集後、他のプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。
[適用]をクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

[OK]ボタンもしくは[適用]ボタンがクリックされると、ここで設定した内容からコマンドファイル(<プロジェクト名>_gnu33IDE.cmd)が生成され、デバッグ起動時にデバッグに渡されます。

例: デフォルト設定で生成されるコマンドファイル

```
# Command file generated by Gnu33 Plug-in for Eclipse
# load debugging information in program
file sample.elf                ... デバッグ情報の読み込み
# set the memory configuration map file to the debugger
c33 rpf sample_gnu33IDE.par     ... パラメータファイルの読み込み
# connect to the debugger with specified mode and port
# target [icd usb | icd6 usb | icd com | sim]
target icd6 usb                ... コネクトモードの設定(ICD Ver6の場合)*1
# load program to memory
load sample.elf                ... elf オブジェクトファイルの読み込み
# set ttbr register
set {char}0x4812d=0x59         ... ブートベクタアドレスの設定(C33 STDの場合)*2
set {long}0x48134=0x00C00000
# hot reset
c33 rsth                       ... ホットリセット
```

- *1 デフォルト設定では、デバッグをICD Ver6モードに設定するコマンドが表示されます。その他のモードで使用する場合、直接コマンドを書き換えるか、[雛形からコマンドを生成する]ボタンで表示される[初期起動コマンドの生成]ダイアログ(下記参照)を使用して変更します。

ICD Ver3/Ver4以外のコネクトモードを指定する場合は次のようになります。

```
target icd comN      ICD Ver2またはMON選択時(NはCOMポート番号)
target icd6 usb      ICD Ver6選択時
target sim           Simulator選択時
```

- *2 プロジェクト作成時、あるいは[プロパティ]ダイアログの[GNU33 一般設定]ページで設定したターゲットCPUとブートベクタアドレスの設定により変わります。ブートベクタアドレスは[初期起動コマンドの生成]ダイアログ(下記参照)でも変更可能です。

S1C33401、C33 PEの場合、ブートベクタアドレス設定コマンドは次のようになります(アドレスはデフォルト設定値)。

```
set $ttbtr=0x20000000    S1C33401選択時
set $ttbtr=0x00C00000    S1C33PE選択時
```

このコマンドファイルはデバッグの起動時に実行され、以下の初期設定を行います。

1. デバッグ情報とパラメータマップ情報の取り込み
2. コネクトモードの設定
3. デバッグするオブジェクトファイルの読み込み
4. PC(プログラムカウンタ)をブートアドレスに設定

これにより、デバッグ起動後すぐにブートアドレスからプログラムを実行できます。

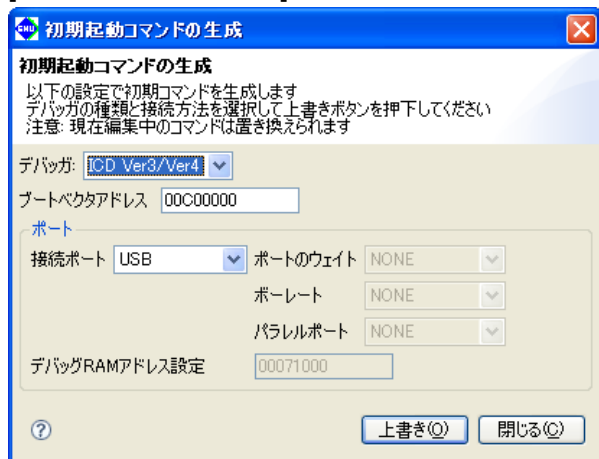
このほかに実行したいコマンドがあれば、キーボードから入力して追加してください。

デバッグコマンドについては"10 デバッグ"を参照してください。

デバッグ起動コマンドの設定は、コマンドファイルをエディタで開いて編集することも可能です。

ただし、コマンドファイルを外部エディタで開いている状態で[プロパティ]ダイアログを開いても、その内容が[GNU33 GDBコマンド]ページに読み込まれない場合があります。この場合は、外部エディタでコマンドファイルを閉じてから、[プロパティ]ダイアログを開いてください。

[初期起動コマンドの生成]



[デバッグ:]

接続するデバッグ(コネクトモード)を選択します。5種類の選択肢がありますが、実際に選択できるのは、[GNU33 一般設定]ページで選択されているターゲットCPUによって変わります。

ICD Ver6 ICD Ver. 6(S5U1C33001H1400)を使用してデバッグを行う場合

ICD Ver3/Ver4 ICD Ver. 3(S5U1C33001H1100)またはVer. 4(S5U1C33001H1200)を使用してデバッグを行う場合

ICD Ver2 ICD Ver. 2(S5U1C33000H)を使用してデバッグを行う場合

MON デバッグモニタ(S5U1C330M1D1+S5U1C330M2S)を使用してデバッグを行う場合

Simulator パーソナルコンピュータのみでデバッグを行う場合

表5.8.2.1 ターゲットCPUとコネクトモード

モード	C33 STDコア	C33 PEコア	S1C33401
ICD Ver6	○	○	○
ICD Ver3/Ver4	○	○	○
ICD Ver2	○	×	×
MON	○	×	×
Simulator	○	○	○

[ブートベクタアドレス]

ブートベクタアドレスをテキストボックスに16進数(0xは不要)で入力します。C33 STDまたはC33 PE コア搭載機種を使用する場合、アドレスは400KB単位で指定する必要があります。S1C33401では1KB 単位となります。

IDEが生成するコマンドファイルには、ブートベクタアドレスをこの値に設定するコマンドが記述されます。

ここにデフォルトで表示されている値は、プロジェクト作成時に指定したアドレスです。

[ポート]

[デバッグ:]にICD Ver2またはMONを選択した場合に、使用するCOMポートの設定を行います。ICD Ver3/Ver4、ICD Ver6またはSimulatorを選択した場合、ポートはUSB、NONEに固定されますので設定の必要はありません。

[接続ポート]

S5U1C33000H(ICD Ver2)またはS5U1C330M1D1(MON)を接続するCOMポート(COM1～COM8)を選択します。

[ポートのウェイト]

PC側のCOMポートを開く際に入れるウェイトを指定します。0～20秒までのウェイトを設定可能です。ノートPC等の一部PCでCOMポートを開いた後にウェイトを挿入しないと正常に通信を開始できない場合があります。そのような場合にウェイトを設定してください。

[ボーレート]

COMポートのボーレートを115200bpsまたは38400bpsから選択します。[デバッグ:]にMONを指定した場合は、115200bpsを指定する必要があります。

[パラレルポート]

ターゲットにプログラムを転送するときに使用するパラレルポートの番号を指定します。

NONE、LPT1、LPT2を指定可能で、それぞれパラレル使用なし、パラレルポート1、2に対応します。

[デバッグ:]でICD Ver2を指定した場合のみ選択可能です。

[デバッグRAMアドレス設定]

デバッガがデバッグ時に使用するターゲット側メモリのアドレスを設定します。[GNU33 一般設定]ページのターゲットCPUにS1C33PE、[デバッグ:]にICD Ver3/Ver4またはICD Ver6を選択した場合のみ設定可能です。この組み合わせ以外では[デバッグRAMアドレス設定]を変更することはできません。ここで指定した領域はプログラム内で使用しないようにしてください。

[上書き]

上記の設定内容を[GNU33 GDBコマンド]ページのコマンドファイルに反映させて、ダイアログを開じます。このボタンをクリックすると上書きを確認するダイアログが表示され、実行のキャンセルも可能です。

[閉じる]

本ダイアログを閉じます。変更内容は[GNU33 GDBコマンド]ページのコマンドファイルに反映されません。

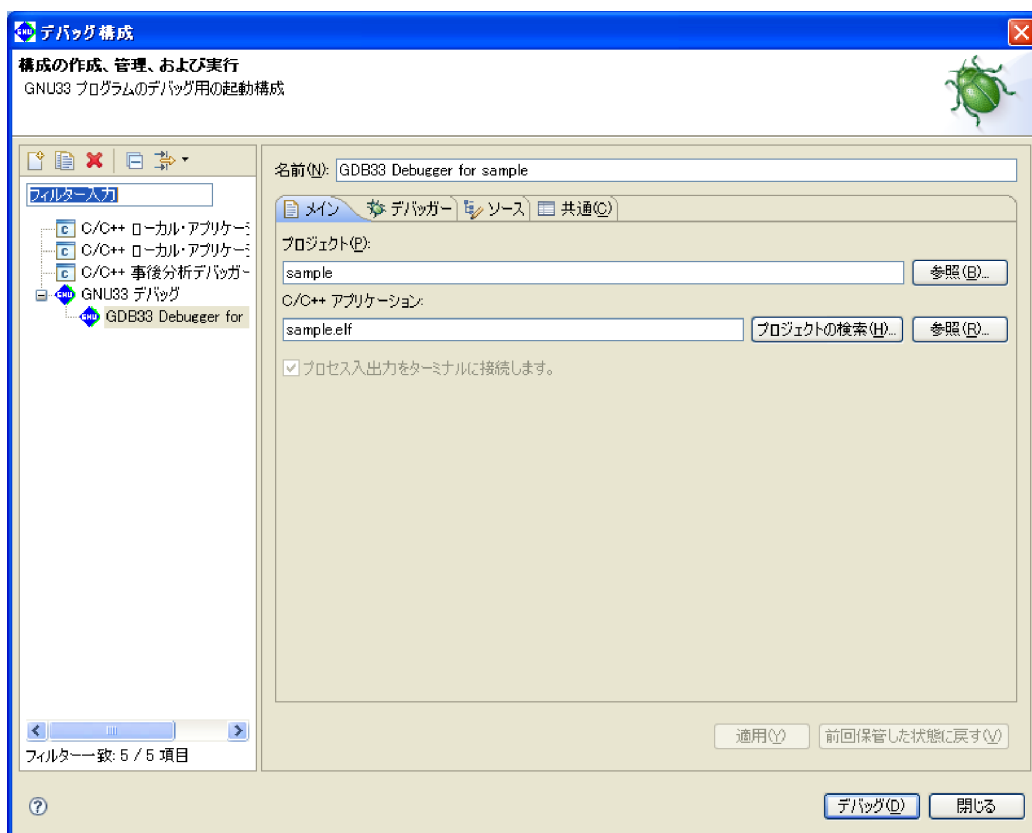
5.8.3 デバッガの起動方法

ビルドによって実行ファイル(.elf)が生成され、前節までの準備が完了すると、デバッグを開始できます。デバッグの開始は、起動構成画面から行います。

起動構成画面では、デバッグを開始するための各種設定を行い、デバッガ(GDB)を起動します。

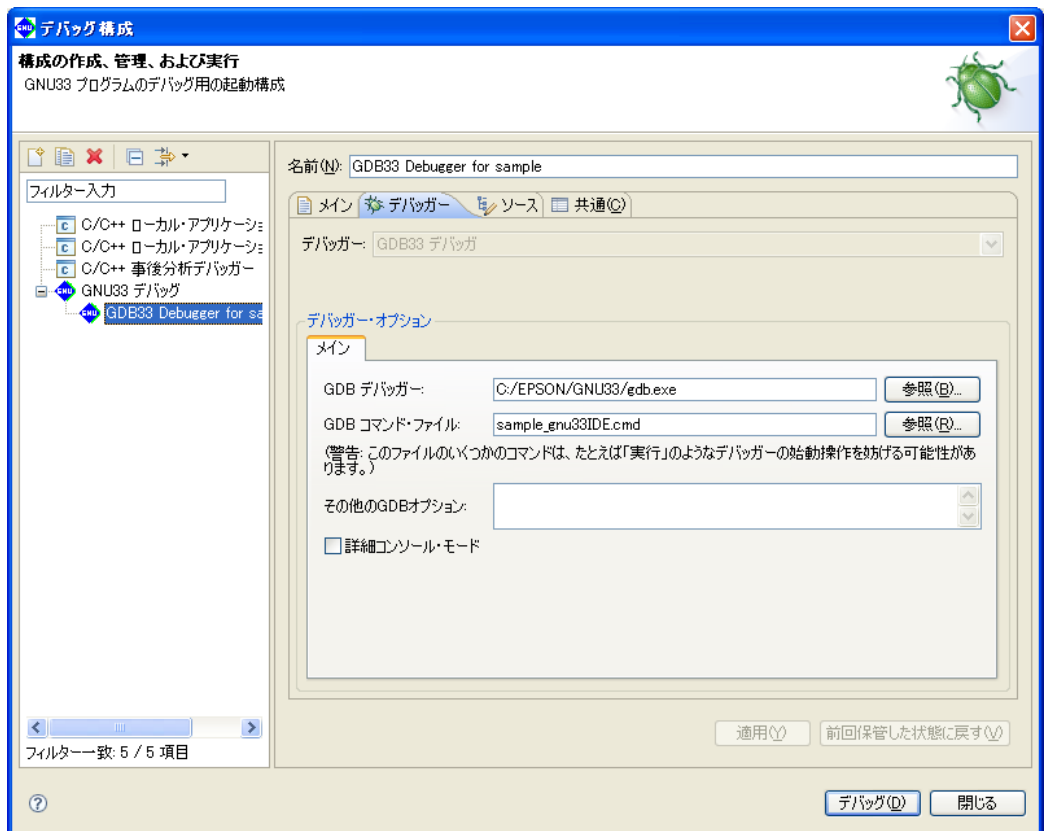
●デバッグ開始

- (1) [実行]メニューから[デバッグの構成...]を選択して起動構成ダイアログを表示させます。ツールバーの[デバッグ]ボタンのメニューからも表示させることができます。
- (2) ツリーリストから[GDB33 Debugger for <プロジェクト名>]を選択します。



- (3) 必要に応じて[デバッガ]タブの[その他のGDBオプション:](デバッガコマンドラインの引数)を変更します。

IDEが生成したコマンドファイルを使用してデバッガを起動する場合は、変更する必要はありません。



(4) [デバッグ] ボタンをクリックします。

デバッガ **gdb** が起動して指定のコマンドファイルを実行します。

これ以降のデバッグ操作については "10 デバッガ" を参照してください。

● デバッガの終了

以下のいずれかの方法でデバッグを終了することができます。

デバッグ終了後、[デバッグ] ビューの表示が終了状態に変わります。

- [実行] メニューから [終了] を選択します。
- [デバッグ] ビューの [終了] ボタンをクリックします。
- [コンソール] ビューの [終了] ボタンをクリックします。
- [デバッグ] ビューのコンテキストメニューから [終了] を選択します。

● デバッガの再起動

デバッガ再起動は、停止しているデバッガ (GDB) を再度起動しなおすことを指します。

一度デバッグを行った後は、起動構成ダイアログを開くことなく同じ起動方法でデバッグを開始することができます。

以下のいずれかの方法でデバッガを再起動することができます。

- [実行] メニューから [前回の起動をデバッグ (F11)] を選択します。
- [実行] メニューから [履歴デバッグ] の一覧から前回の [GDB33 Debugger for <プロジェクト名>] を選択します。
- ツールバーの [デバッグ] ボタンをクリックします。
- [デバッグ] ビューのコンテキストメニューから [再起動] を選択します。

●起動構成ダイアログ

[起動構成の新規作成]

• 起動構成を新規作成する場合

ツリーから[GNU33 デバッグ]をダブルクリックすることで、起動構成の新規作成を行うことができます。新規に作成された起動構成は[新規作成]の名前で作成されます。

起動に必要な設定(プロジェクト名、実行ファイル、使用するコマンドファイル)は、以降の手順から入力してください。

注: 起動構成の設定をプロジェクトフォルダに保存するためには、[共通]タブの[共用ファイル]の設定でプロジェクトフォルダを指定してください。

• 既存のプロジェクトから起動構成を新規作成する場合

[C/C++ プロジェクト]ビューでプロジェクトを選択した状態で起動構成ダイアログを開き、[GNU33 デバッグ]をダブルクリックすると、選択されたプロジェクトから、プロジェクト名、実行ファイル、使用するコマンドファイルをあらかじめ設定します。

注: [C/C++ プロジェクト]ビューでプロジェクトを選択し、GNU33/パースペクティブで起動構成ダイアログを開いてください。

デバッグパースペクティブから起動構成ダイアログを開くと、[C/C++ プロジェクト]ビューで選択されているプロジェクトが取得できないため、[新規作成]で作成されてしまいます。

[起動構成の設定]

起動構成を選択すると、右ペインに各タブ画面が表示されます。
これらのタブ画面にデバッガに関する各種設定を入力します。

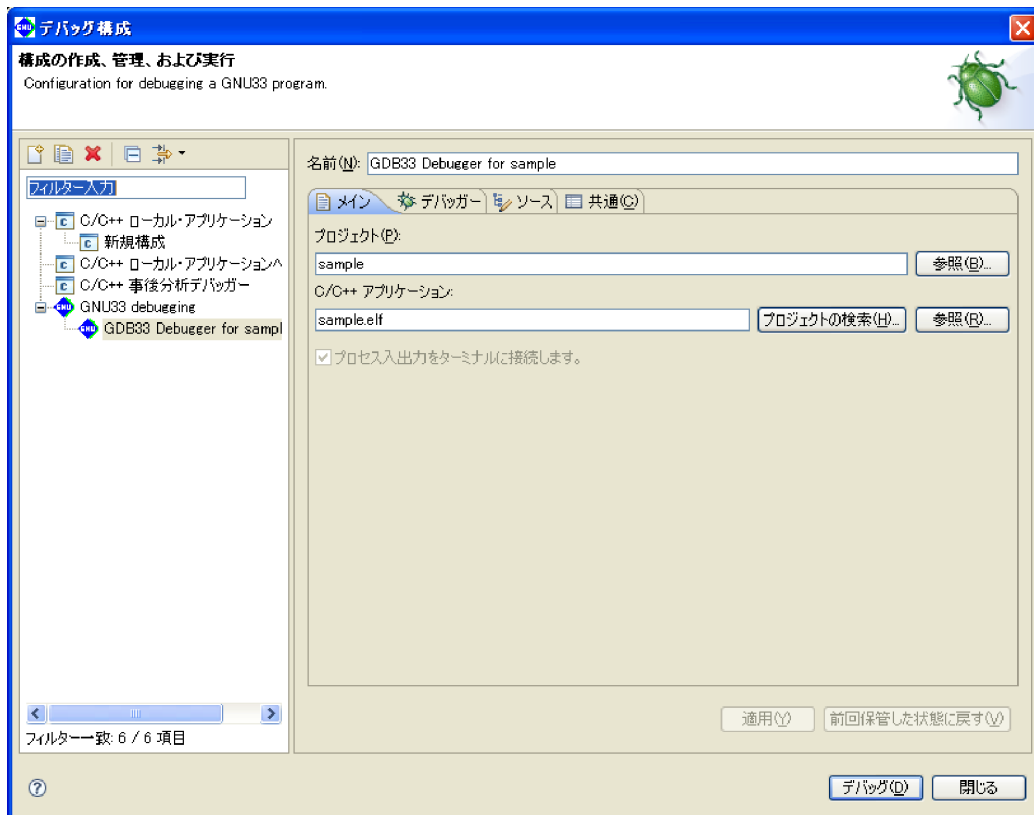


表5.8.3.1 起動構成画面のタブ一覧

タブ	設定内容
メイン	デバッグ対象のプロジェクトとターゲットプログラム
デバッガ	デバッガへのパス・引数
ソース	ソース検索パス
共通	起動構成に関する共通設定

[メイン]タブ

デバッグ対象のプロジェクトとターゲットプログラムに関する情報を入力・表示します。

名前(N): GDB33 Debugger for sample

メイン

デバッガー

ソース

共通(Q)

プロジェクト(P):

sample

参照(R)...

C/C++ アプリケーション:

sample.elf

プロジェクトの検索(H)...

参照(R)...

☒ プロセス入出力をターミナルに接続します。

適用(Y)

前回保管した状態に戻す(V)

表5.8.3.2 [メイン]タブ

入力項目	内容
プロジェクト	デバッグ対象のプロジェクト名を入力します。 [参照]ボタンから、ワークスペース内のプロジェクトを選択することができます。
C/C++ アプリケーション	デバッグ対象の実行ファイル名(elf)を入力します。 [プロジェクトの検索]ボタンから、プロジェクト内のelfファイルを選択することができます。(プロジェクトが入力されている場合) [参照]ボタンからファイルダイアログが開き、任意のelfファイルを選択することができます。(プロジェクトが入力されている場合)

[デバッグ]タブ

デバッガ(GDB)へのパス・引数を設定します。

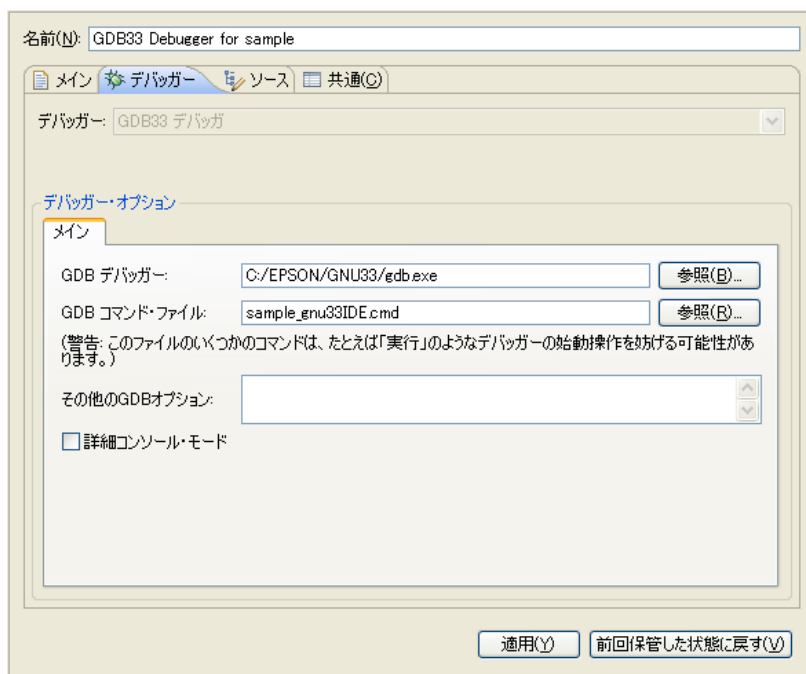


表5.8.3.3 [デバッグ]タブ

入力項目	内容
デバッガ	起動するデバッガの種類を選択します。 GNU33 Debuggerに固定です。
GDB デバッガ	起動するデバッガのパスを表示・入力します。 [参照]ボタンのファイルダイアログから選択することができます。
GDB コマンド・ファイル	起動時に使用するコマンドファイルを指定します。 (デフォルトは<プロジェクト名>_gnuXXIDE.cmd) 空のときは-nx [参照]ボタンのファイルダイアログから選択することができます。
その他のGDBオプション	gdb.exe 起動のために追加で指定する引数を入力できます。 指定できるパラメータ例) --c33_cmw=n --double_starting 以下のオプションを指定すると誤動作しますので指定しないでください。 (-x、--command、--cd、--directory)
詳細コンソール・モード	IDE-GDB間のMIコマンドの通信を表示します。 デフォルト：OFF

[ソース]タブ

ソースレベルデバッグのためのソース検索パスを指定します。

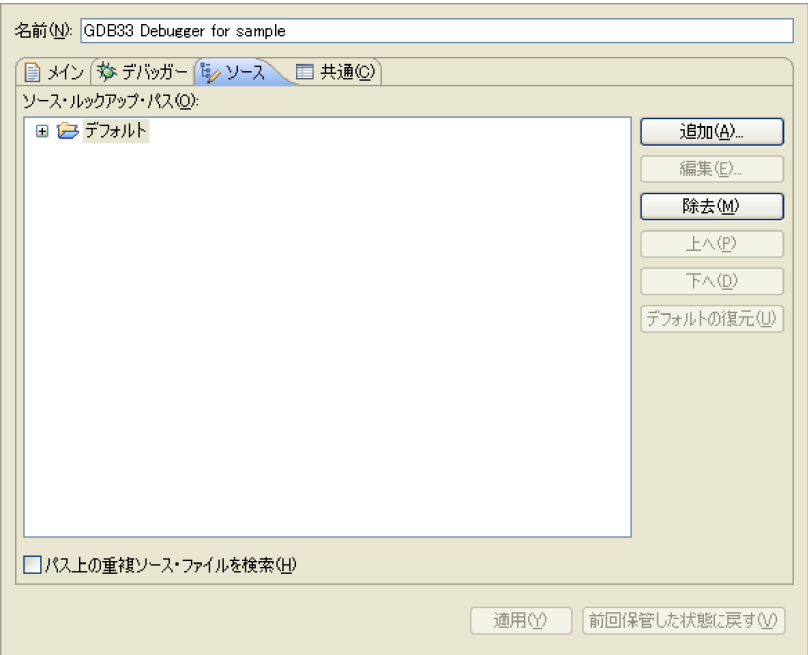


表5.8.3.4 [ソース]タブ

入力項目	内容
ソース・ルックアップ・パス	ソース検索パスを表示します。 デフォルトは、プロジェクトフォルダが指定されます。
追加	新しいパスを追加します。
編集	ソース検索パスを編集します。
除去	ソース検索パスを削除します。
上へ/下へ	パスの検索順序を変更します。([上へ]優先)
デフォルトの復元	ソース検索パスをデフォルトの設定に戻します。
パス上の重複ソース・ファイルを検索	ソース検索パス上の重複ファイルを探します。 サポートしていません。

[共通]タブ

起動構成に関する共通設定を行います。

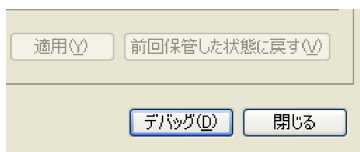
表5.8.3.5 [共通]タブ

入力項目	内容
ローカル・ファイル	この起動構成をIDEが内部に保存します。
共用ファイル	この起動構成をlaunchファイルとしてプロジェクト内等に保存します。保存するパスを指定してください。 デフォルトでは、"GDB33 Debugger for <プロジェクト名>.launch"の名前でプロジェクトフォルダ直下に保存されます。
お気に入りのメニューに表示	この起動構成のショートカットを、ツールバーの[デバッグ]ボタンのメニューに登録します。
コンソール・エンコード	コンソールの出力エンコーディングを指定します。 通常は設定する必要はありません。
コンソールに割り当て (入力に必要)	[コンソール]ビューでGDBへのコマンド入力を可能にします。必ずONに設定してください。
ファイル	[コンソール]ビューのGDBコマンドの出力をファイルにリダイレクトします。([コンソール]ビューにも出力されます) ファイル名を指定してください。 [ワークスペース...]/[ファイル・システム...]からファイルやパスを指定することができます。
追加	[ファイル]を指定したとき、アペンドモードで出力します。
バックグラウンドでの起動	バックグラウンドで起動します。 通常はONに設定してください。

注: 起動構成の設定をプロジェクトフォルダに保存するためには、[共用ファイル]の設定でプロジェクトフォルダを指定してください。

[起動構成設定の決定]

起動構成ダイアログの設定は以下の各ボタンで確定します。

**[適用]**

現在表示中のタブ画面の設定を確定します。

[前回保管した状態に戻す]

現在表示中のタブ画面の設定を前回保存した状態に戻します。

[閉じる]

起動構成ダイアログを閉じます。

設定を変更して保存していない場合は、確認ダイアログが表示されます。

[デバッグ]

タブ画面の設定を確定し、デバッグを起動します。

初回の起動後は、当ボタンからではなく、ツールバーもしくは[実行]メニューのショートカットから起動できます。

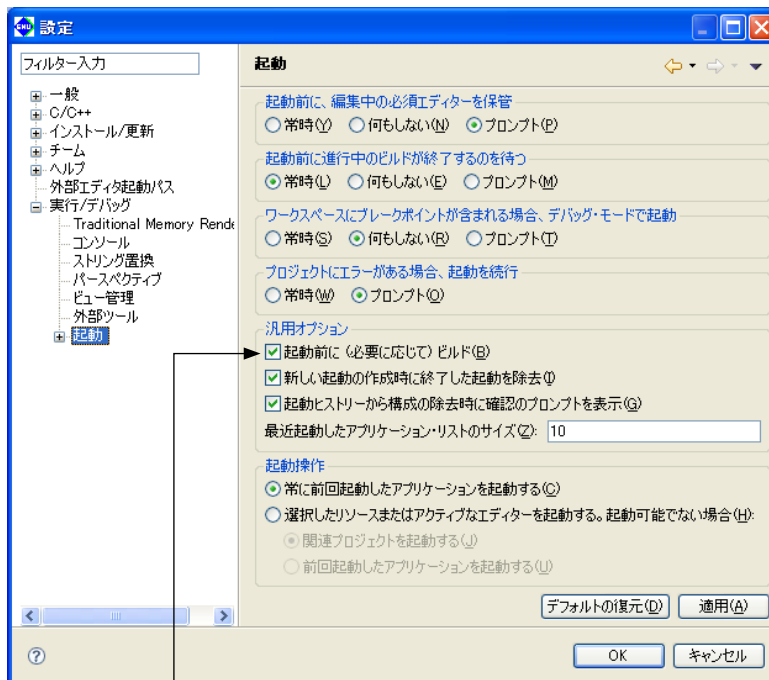
[ブレークポイント]ビュー一覧にブレークポイントがある場合、さらにブレークポイントをデバッグに対して設定します。

注: コマンドファイル内にエラーがある場合、エラーメッセージは[コンソール]ビューに出力され、以降のコマンドは実行されずにデバッグを起動します。

●注意事項

- デバッガを起動後は、IDEでビルドを実行することはできません。ビルドを行う場合は、デバッガを終了してください。
- デバッガ起動中にIDEを終了するとデバッガも終了します。
- [デバッグ]のメニューよりデバッガを起動すると、起動前にビルドが行われることがあります(ソースファイルが変更されている場合など)。また、このビルドでエラーが発生してもデバッガが起動します。
デバッガ起動前にビルドを実行させたくない場合は、以下の手順で起動前のビルドを無効に設定してください。

- (1) [ウィンドウ]メニューから[設定...]を選択し、[設定]ダイアログを表示させ、[実行/デバッグ]>[起動]のページを表示させます。

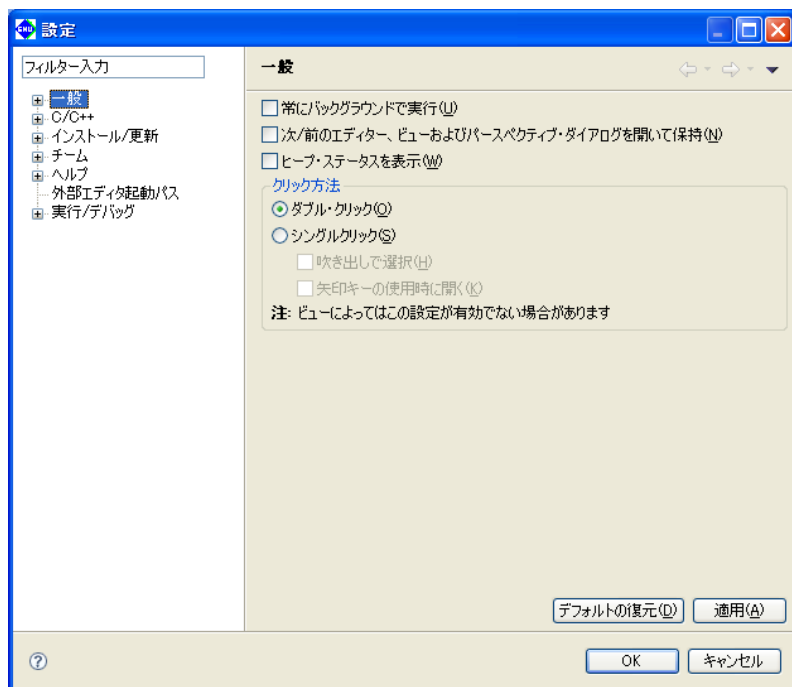


- (2) [起動前に (必要に応じて) ビルド]のチェックを外します。

5.9 IDEのカスタマイズ(設定)

IDEの操作や表示に関する初期設定内容を、ユーザの使い方に合わせてカスタマイズできるようになっています。

この設定は、[ウィンドウ]メニューから[設定]を選択することにより表示される[設定]ダイアログで行います。



カスタマイズ可能な項目はダイアログの左にツリー表示されていますので、変更したい項目を選択し、設定ページを表示させます。[フィルター入力]に入力した文字列で始まる単語を含む設定項目のみをツリーリストに表示させることもできます。

各ページに共通なボタンの機能は次のとおりです。

 [戻る]

前に参照/編集していたページに戻ります。

 [進む]

上記の[戻る]で遡った表示を新しい方に戻します。

[デフォルトの復元]

各ページの設定内容をダイアログを開いたときの状態、または[適用]ボタンで設定を確定した状態に戻します。

[適用]

そのページで設定した内容を適用します。他の項目を変更する場合は、[適用]をクリックしてから新たなページに移動してください。

[OK]

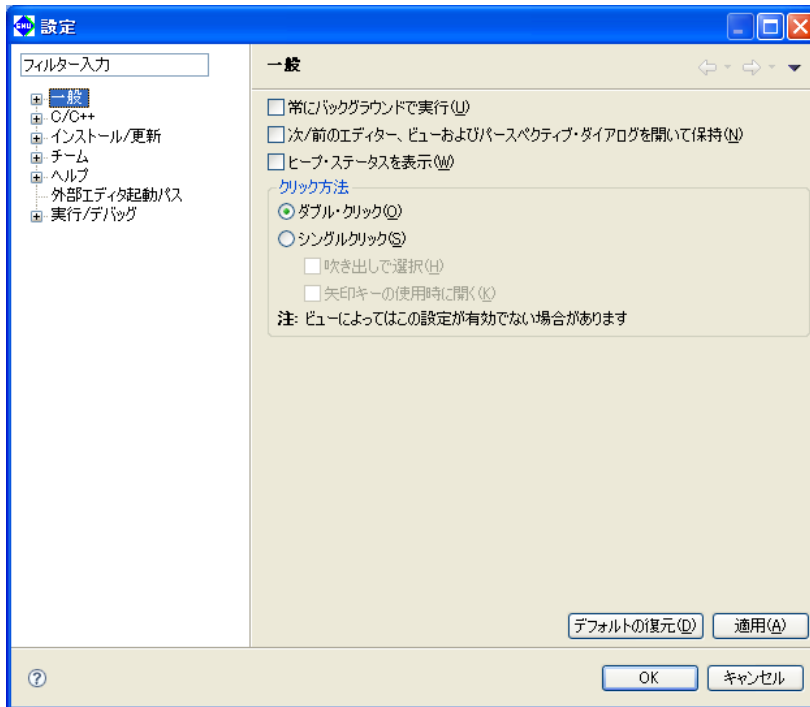
現在のページの設定内容を適用し、ダイアログを閉じます。

[キャンセル]

設定を中止して、ダイアログを閉じます。その前に[適用]ボタンで適用済みの設定内容は元には戻りません。

以下、カスタマイズ項目のページと設定内容を示します。ここで説明されていない設定内容は変更しないでください。

●一般



IDEの動作に関する設定を行います。

[常にバックグラウンドで実行] (デフォルト: OFF)

このチェックボックスを選択すると、ビルド処理などをバックグラウンドで行い(ビルド中に表示されるダイアログが表示されなくなります)、その間も他の作業を可能にします。

[次/前のエディター、ビューおよびパースペクティブ・ダイアログを開いて保持] (デフォルト: OFF)

IDEには[Ctrl]+[F6](エディタ)、または[Ctrl]+[F7](ビュー)により、現在のエディタ/ビューと直前参照していたエディタ/ビューを切り替える機能があります。通常はキー入力により即時に切り替わりますが、このチェックボックスを選択すると、履歴がプルダウンメニューとして現れ、その一覧からアクティブにするエディタ/ビューを選択できるようになります。

[ヒープ・ステータスを表示] (デフォルト: OFF)

このチェックボックスを選択すると、Javaヒープの使用状況をワークベンチウィンドウの右下に表示します。

[クリック方法]

[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューからリソースをエディタで開く操作を選択します。

[ダブル・クリック] (デフォルト: ON)

リソースをシングルクリックすると選択され、ダブルクリックすると開きます。

[シングルクリック] (デフォルト: OFF)

リソースをシングルクリックすると開きます。

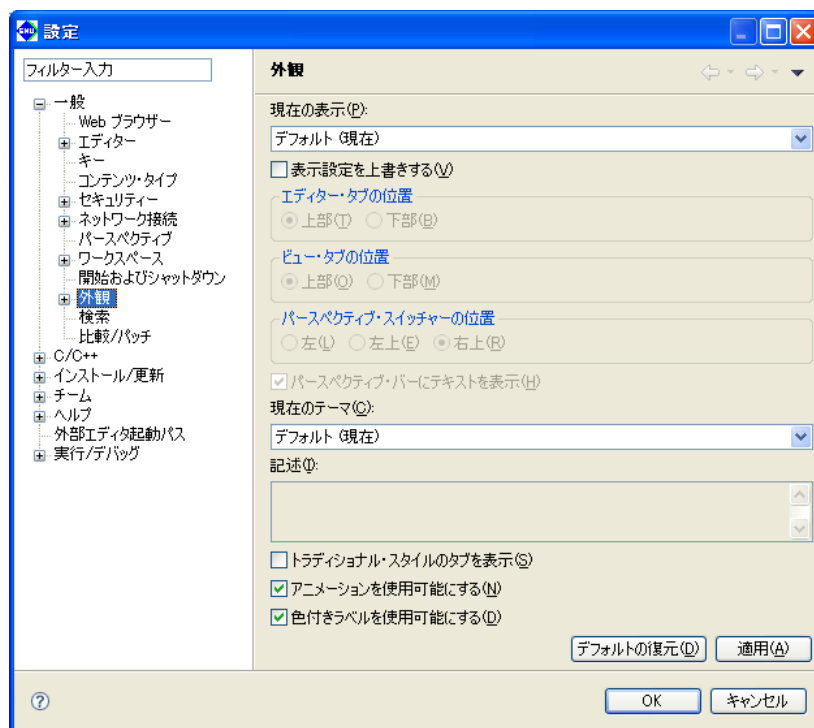
[吹き出しで選択] (デフォルト: OFF)

このチェックボックスを選択すると、ビュー内のリソースがマウスカーソルを重ねることで選択状態になります。([シングルクリック]選択時のみ有効)

[矢印キーの使用時に開く] (デフォルト: OFF)

このチェックボックスを選択すると、矢印キーでリソースを選択することによってもエディタで開くことができます。([シングルクリック]選択時のみ有効)

●一般 > 外觀



IDEの外観やエディタとビューのタブの表示位置などを指定します。

[現在の表示:]

IDEの外観をEclipse 2.1のスタイルに切り替えることができます。ただし、この変更には、**IDE**をリスタートさせる必要があります。

[表示設定を上書きする] (デフォルト: OFF)

エディタのタブの表示位置を変更する場合に選択します。

[エディター・タブの位置]

エディタのタブの表示位置を指定します。

[上部] (デフォルト: ON)

ビューの上部に表示します。

[下部] (デフォルト: OFF)

ビューの下部に表示します。

[ビュー・タブの位置]

ビューのタブの表示位置を指定します。

[上部] (デフォルト: ON)

ビューの上部に表示します。

[下部] (デフォルト: OFF)

ビューの下部に表示します。

[パースペクティブ・スイッチャーの位置]

パースペクティブバーの表示位置を指定します。

[左] (デフォルト: OFF)

ウィンドウの左端に表示します。

[左上] (デフォルト: OFF)

ウィンドウの左上部(ツールバーの下)に表示します。

[右上] (デフォルト: ON)

ウィンドウの右上部(ツールバーの右)に表示します。

[パースペクティブ・バーにテキストを表示] (デフォルト: ON)

このチェックボックスを選択すると、パースペクティブバーにパースペクティブ名も表示します。
OFFの場合はアイコンのみ表示します。

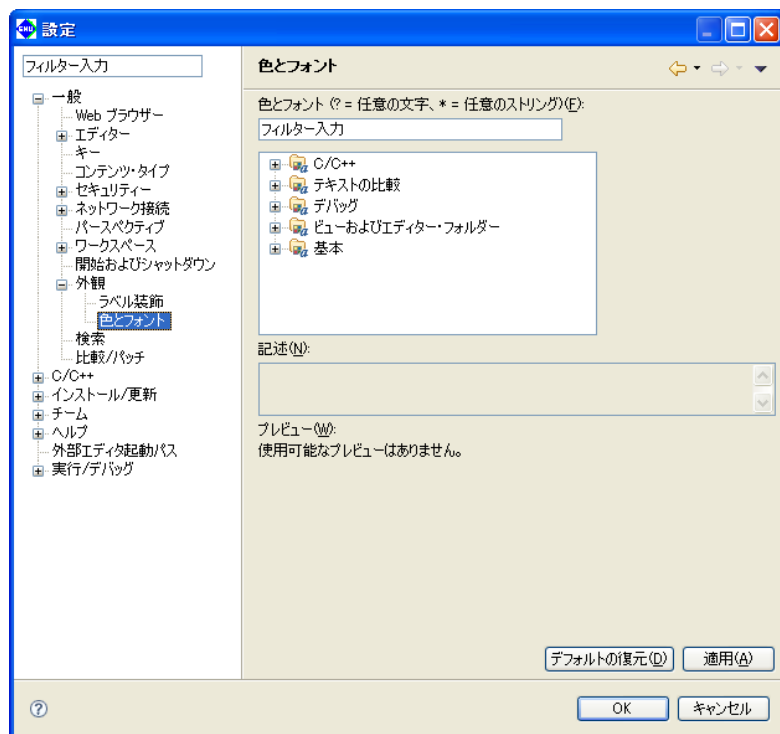
[トラディショナル・スタイルのタブを表示] (デフォルト: OFF)

このチェックボックスを選択すると、エディタやビューのタブの形状を角張ったスタイルに変更します。

[アニメーションを使用可能にする] (デフォルト: ON)

このチェックボックスを選択すると、高速ビューのオープン/クローズ時のアニメーション機能が有効になります。

●一般 > 外観 > 色とフォント



エディタなどで使用するフォントとカラーを設定します。

[色とフォント:]

一覧に表示させる項目を指定します。"? "は任意の1文字、"* "は任意の文字列を指定するワイルドカードとして使用可能です。

リストボックス

カラーとフォントの設定がカテゴリ別にリストされます。この一覧から変更するカラーまたはフォントを選択します。

[記述:]

リストから選択したカラー / フォントが使用される場所などの説明が表示されます。

[プレビュー:]

リストから選択したカラー / フォントの表示サンプルが用意されていれば、ここに表示されます。

以下のボタンは、リストからフォントを選択すると表示されます。

[システム・フォントの使用]

リストで選択したフォントをシステムフォントに変更します。

[変更...]

リストで選択したフォントを任意のフォントに変更します。[フォント]ダイアログが表示されますので、そこでフォントやサイズを選択します。

[リセット]

変更したフォントをデフォルト設定に戻します。フォントが変更されている場合にアクティブになります。

以下のボタンはリストからカラーを選択すると表示されます。

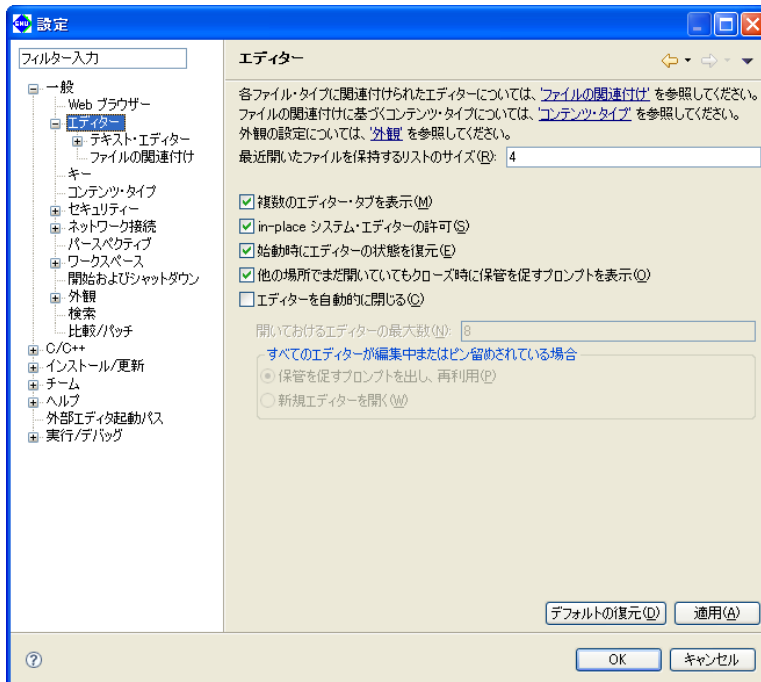
カラー選択ボタン

[色の設定]ダイアログが表示され、新しいカラーを選択することができます。

[リセット]

変更したカラーをデフォルト設定に戻します。カラーが変更されている場合にアクティブになります。

●一般 > エディター



すべてのエディタに関わる項目の設定を行います。

[最近開いたファイルを保持するリストのサイズ] (デフォルト: 4)

[ファイル]メニューに表示する最近開いたファイルの数を設定します。

[複数のエディター・タブを表示] (デフォルト: ON)

エディタビューに複数のタブを一度に表示するか選択します。

チェックを外すと、最前面の文書のタブのみが表示されます。この場合、他の文書はタブ上のショートカットメニュー(>>)から選択します。

[エディターを自動的に閉じる] (デフォルト: OFF)

このチェックボックスを選択すると、エディタが指定数(デフォルトは8)以上開かれると古いものから自動的に閉じられます。

この機能を選択すると、ツールバーに[エディターのピン留め]ボタンが現れます。このボタンをクリックして押し込まれた状態にしておくと(タブのコンテキストメニューから[エディターのピン留め]を選択しても同様)、そのリソースは自動的に閉じる対象から外れます。

閉じられるファイルが保存されていない場合はダイアログが表示され、保存するか否か、あるいはエディタ数の制限を超えて新たに開くかを選択できます。

[開いておけるエディターの最大数] (デフォルト: 8)

エディタで何個以上のリソースが開かれた場合に上記の自動クローズ機能を有効にするかを指定します。このフィールドは[エディターを自動的に閉じる]が選択された場合にのみ設定可能です。

[すべてのエディターが編集集中またはピン留めされている場合]

自動クローズ機能使用時に、制限数のエディタがすでに開かれ、そのすべてが未保存か"エディターのピン留め"が選択されている場合の処理方法を設定します。このフィールドは[エディターを自動的に閉じる]が選択された場合にのみ設定可能です。

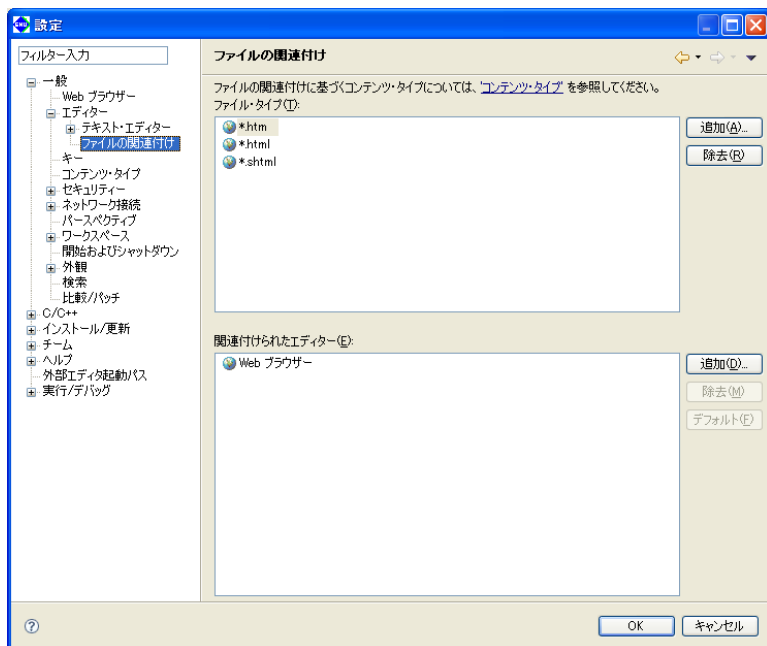
[保管を促すプロンプトを出し、再利用] (デフォルト: ON)

閉じる対象のファイルを保存するか否か、あるいはエディタ数の制限を超えて新たに開くかを選択するダイアログを表示します。

[新規エディターを開く] (デフォルト: OFF)

確認なしに、エディタ数の制限を超えて新たに開きます。

●一般 > エディター > ファイルの関連付け



ファイルの種類(拡張子)と、その編集に使用するエディタを設定します。

[ファイル・タイプ:]

IDEで編集するファイルの種類がリストされます。

[関連付けられたエディター:]

[ファイル・タイプ:]で選択されているファイルの編集に使用するエディタがリストされます。(default)が付記されているエディタが、[C/C++ プロジェクト]ビューや[ナビゲーター]ビュー内のファイル名をダブルクリックして開く際に使用されます。リスト内のその他のエディタは、ファイルをコンテキストメニューの[アプリケーションから開く]で開く際に指定できます。

[追加...]

リストにファイルの種類やエディタを追加します。ボタンをクリックすると表示されるダイアログで入力あるいは選択が行えます。

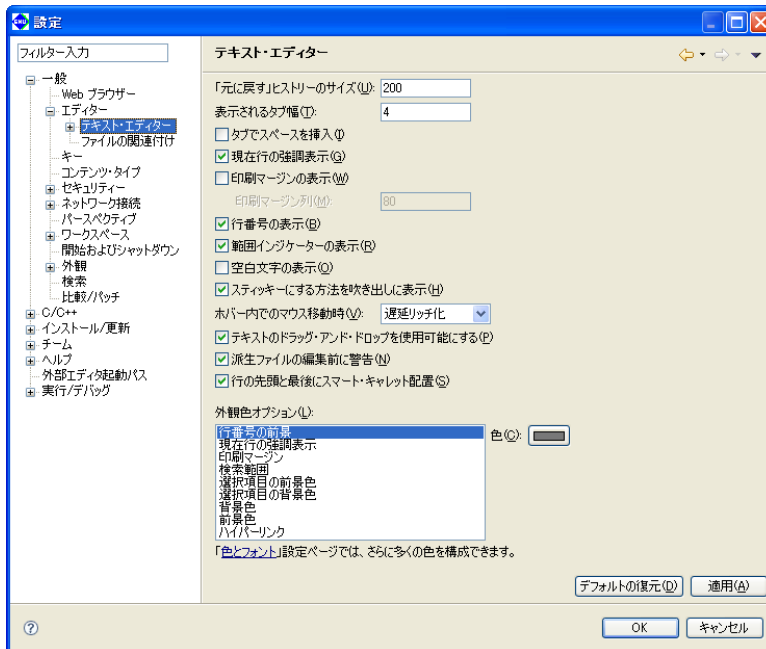
[除去]

リストボックスで選択されているファイルの種類やエディタを一覧から取り除きます。

[デフォルト]

[関連付けられたエディター:]で選択されているエディタをデフォルトエディタに設定します。

●一般 > エディター > テキスト・エディター



IDEが搭載しているテキストエディタに関する設定を行います。

[表示されるタブ幅:] (デフォルト: 4)

タブスペースを文字数で指定します。

[「元に戻す」履歴のサイズ:] (デフォルト: 200)

操作を現在から遡って取り消し可能な回数を指定します。

[現在行の強調表示] (デフォルト: ON)

このチェックボックスを選択すると、現在行を色付きでハイライト表示します。

[印刷マージンの表示] (デフォルト: OFF)

このチェックボックスを選択すると、プリントマージン([印刷マージン列]で設定されている1行の印刷可能範囲)を示す垂直線が表示されます。

[印刷マージン列] (デフォルト: 80)

印刷時の1行あたりの文字数を指定します。

[行番号の表示] (デフォルト: ON)

このチェックボックスを選択すると、各行の先頭に行番号が表示されます。

[範囲インジケータの表示] (デフォルト: ON)

このチェックボックスを選択すると、[アウトライン]ビューで選択した関数などの位置を示す範囲インジケータがエディタ左端のマーカーバー上に表示されます。

[外観色オプション:]

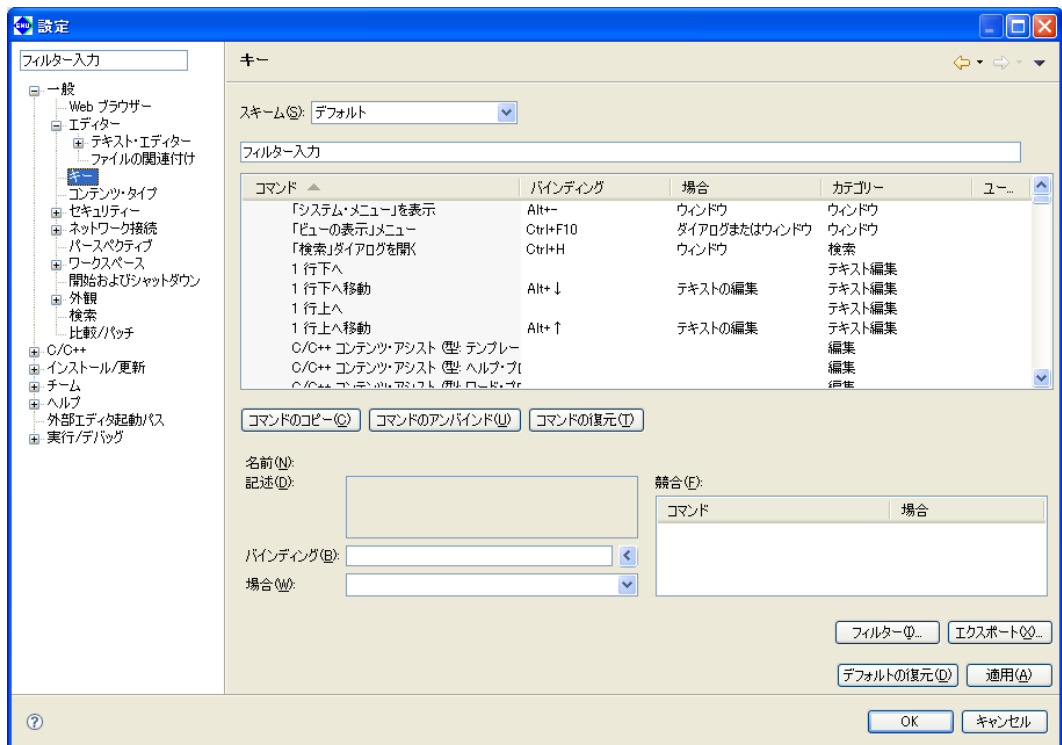
以下の表示色を設定します。一覧から表示色を変更する項目を選択し、[色:]ボタンで表示されるダイアログから色を選択します。

行番号の前景	行番号の文字色
現在行の強調表示	現在行のハイライト色
印刷マージン	プリントマージンの垂直線の色
検索範囲	検索範囲
選択項目の前景色	選択範囲の文字色*
選択項目の背景色	選択範囲の背景色*
背景色	背景色*
前景色	文字色*
ハイパーリンク	ハイパーリンクの文字色

* [システム・デフォルト]をチェックすると、システムのデフォルト設定が使用されます。

●一般 > キー

キーボードショートカットの設定を行います。



キーボードショートカットの一覧を表示します。

[バインディング:]

キーの組み合わせを押下することでキーシーケンスを割り当てることができます。

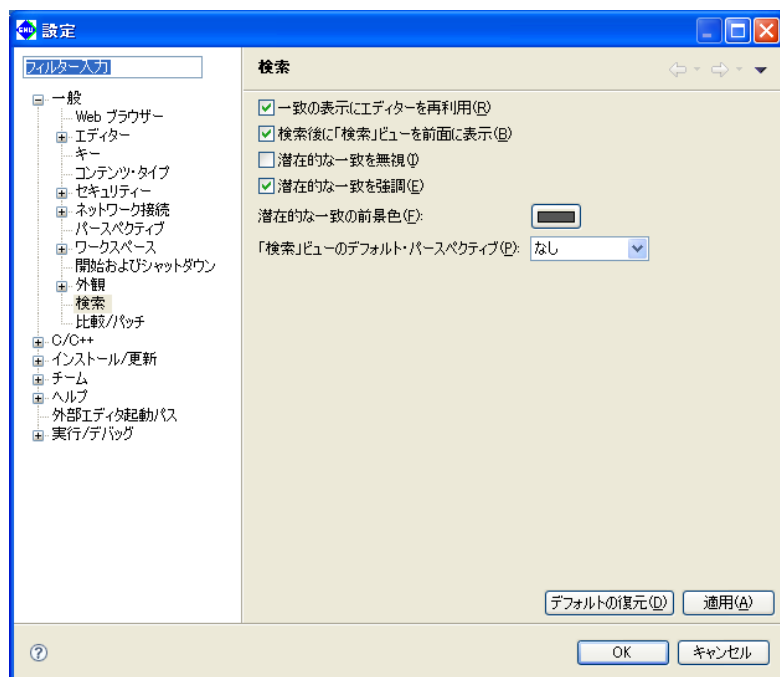
[場合:]

[バインディング:]に設定したキーシーケンスをコマンドに割り当てる際に、コマンドが有効になる場所/状況を選択します。

[エクスポート...]

一覧の内容をCSV形式のファイルに書き出します。

●一般 > 検索



[検索]メニュー/ボタンによる検索に関わる設定を行います。

[一致の表示にエディターを再利用] (デフォルト: ON)

このチェックボックスを選択すると、結果の表示に同じエディタを使用します。同じエディタで表示可能な別のファイルの検索位置に移動する場合、現在のファイルは閉じられます。

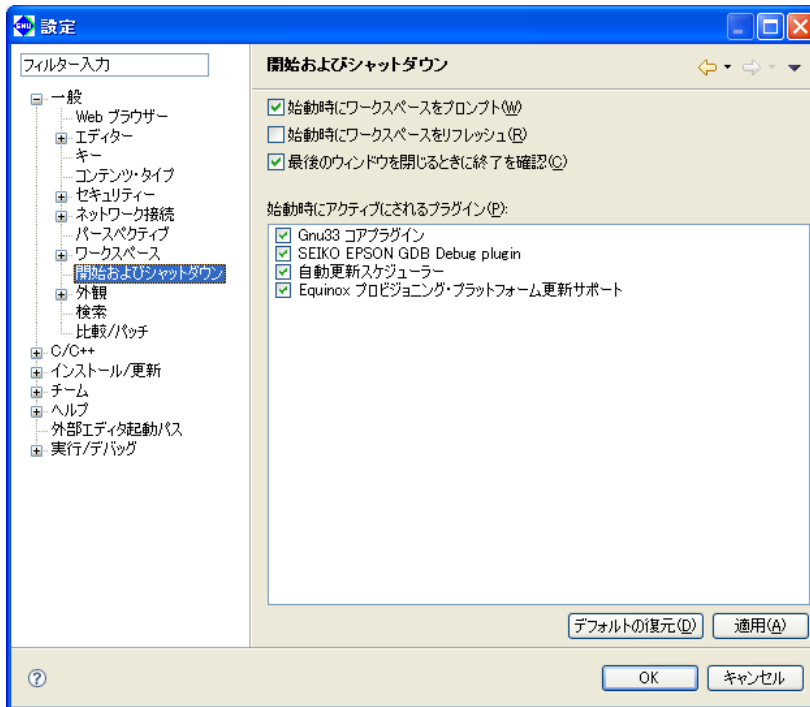
[検索後に「検索」ビューを前面に表示] (デフォルト: ON)

このチェックボックスを選択すると、検索実行後に[検索]ビューが前面に表示されます。

[潜在的な一致を無視] (デフォルト: OFF)

このチェックボックスを選択すると、検索内容に完全に一致した結果のみを表示します。

●一般 > 開始およびシャットダウン



IDEの起動/終了に関する設定を行います。

[始動時にワークスペースをプロンプト] (デフォルト: ON)

このチェックボックスを選択すると、**IDE**の起動時にワークスペースディレクトリを指定するダイアログが表示されます。

[始動時にワークスペースをリフレッシュ] (デフォルト: OFF)

このチェックボックスを選択すると、**IDE**の起動時にワークスペースの情報がファイルシステムの最新の状態に更新されます。

[最後のウィンドウを閉じるときに終了を確認] (デフォルト: ON)

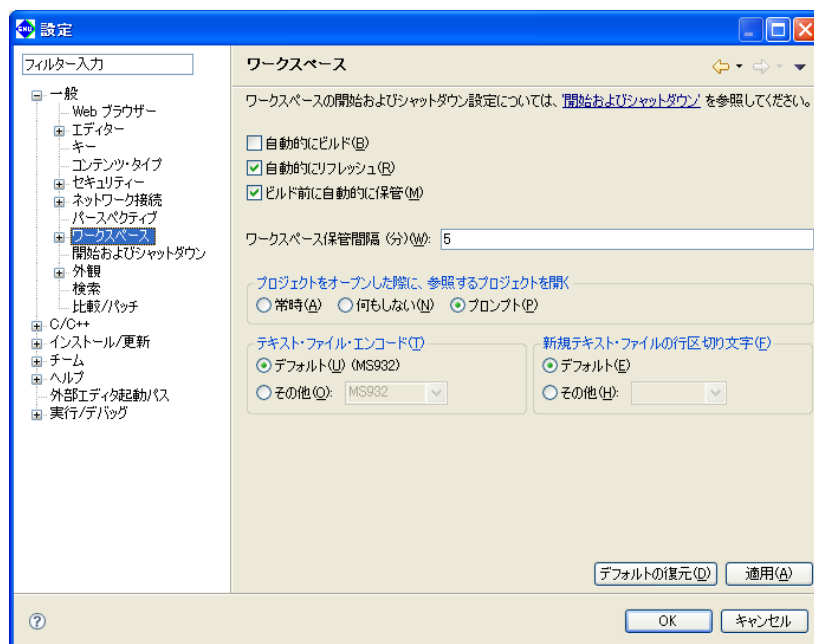
このチェックボックスを選択すると、**IDE**の最後のウィンドウを  (閉じる) ボタンで閉じようとした場合に、操作を確認するダイアログが表示されます。

[始動時にアクティブにされるプラグイン]

IDEの起動時にアクティブにするプラグインを選択します。

この設定は変更しないでください。

●一般 > ワークスペース



IDEの動作に関する設定を行います。

[自動的にビルド] (デフォルト: OFF)

オートビルド機能(エディタで編集しているソースを保存することにより自動的にビルドを実行)を有効にするチェックボックスですが、**IDE**ではオートビルド機能は使用できません。

[自動的にリフレッシュ] (デフォルト: ON)

このチェックボックスを選択すると、ファイルシステム上のリソースの追加や削除が自動的に[C/C++ プロジェクト]ビューと[ナビゲーター]ビューに反映されます。**OFF**にした場合は、[ファイル]メニューまたはビューのコンテキストメニューから[リフレッシュ]を選択して、ビューの表示を更新します。

[ビルド前に自動的に保管] (デフォルト: ON)

このチェックボックスを選択すると、ビルド実行前にエディタで編集中の保存されていないリソースが自動的に保存されます。

[ワークスペース保管間隔 (分):] (デフォルト: 5分)

ワークスペース情報の自動保存間隔を分単位で設定します。

[プロジェクトをオープンした際に、参照するプロジェクトを開く]

プロジェクトを開くときに、そのプロジェクトが参照している他のプロジェクトも開くかどうかを選択します。

[常時] (デフォルト: OFF)

常に開きます。

[何もしない] (デフォルト: OFF)

開きません。

[プロンプト] (デフォルト: ON)

確認のためのダイアログを表示します。

[テキスト・ファイル・エンコード]

テキストのエンコード形式を設定します。

[デフォルト(*1)] (*1: Cp1252、MS932等、Windowsの言語対応により変わります。)

Windows標準の文字コードセットです。(デフォルト)

[その他:]

その他の文字コードを選択します。

[新規テキスト・ファイルの行区切り文字]

行の区切り文字を選択します。この選択は、この後に作成されるファイルに有効で、既存のファイルを開いても変更されません。

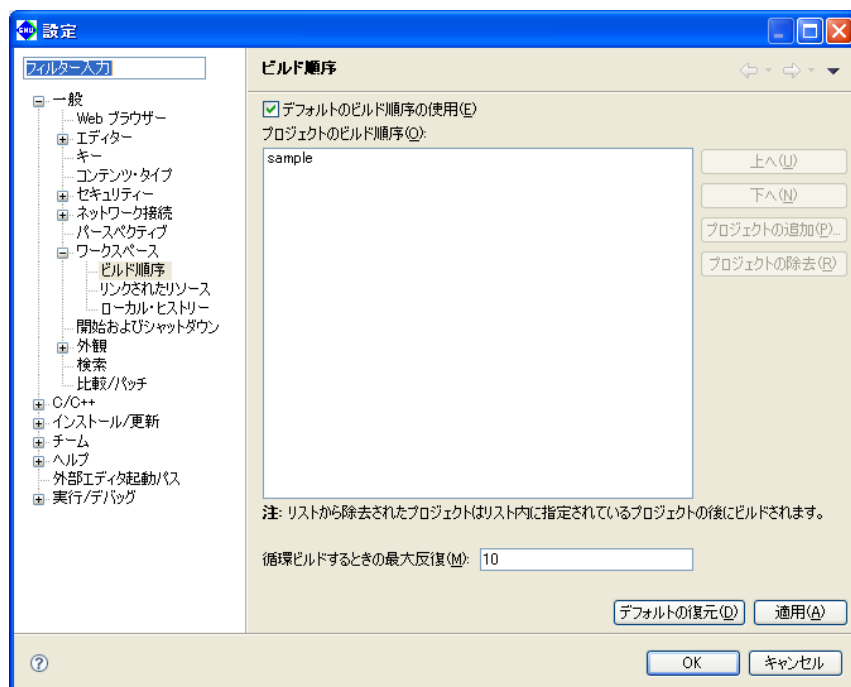
[デフォルト]

Windows標準の区切り文字を使用します。(デフォルト)

[その他:]

その他の区切り文字を選択します。

●一般 > ワークスペース > ビルド順序



プロジェクトをビルドする順序を設定します。

[デフォルトのビルド順序の使用] (デフォルト: ON)

このチェックボックスを選択して[すべてビルド](開いている全プロジェクトのビルド)を実行した場合、プロジェクトは[C/C++ プロジェクト]ビューの表示順(アルファベット順)にビルドされます。OFFにすると、本ページの一覧の順にビルドされます。

[プロジェクトのビルド順序:]

[デフォルトのビルド順序の使用]をOFFにした場合のビルドの順序をここで設定します。一覧のプロジェクトは上から順にビルドされます。また、一覧にないプロジェクトが開いている場合は、一覧内のプロジェクトのビルドが終了後に、アルファベット順に処理されます。

[上へ]

一覧内で選択されているプロジェクトを一つ上に移動します。

[下へ]

一覧内で選択されているプロジェクトを一つ下に移動します。

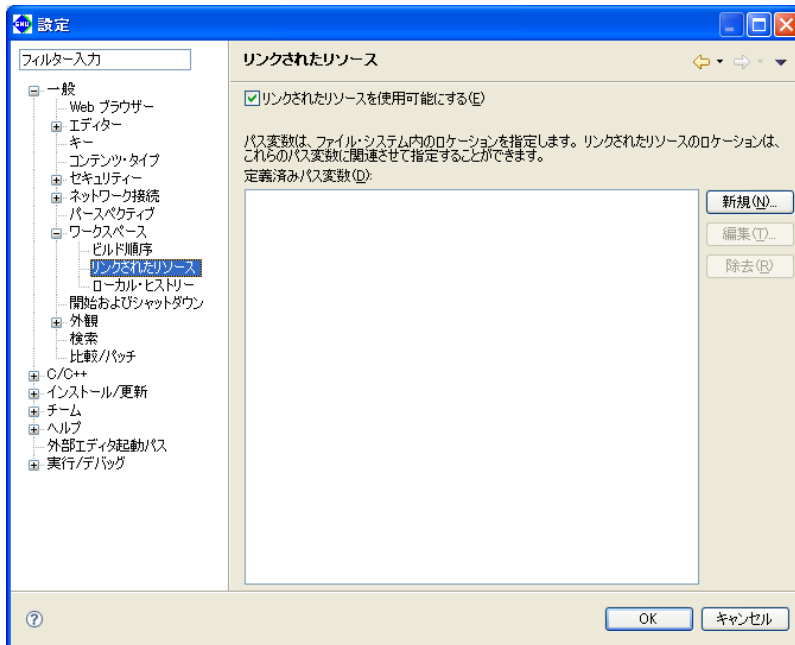
[プロジェクトの追加...]

プロジェクトを一覧に追加します。

[循環ビルドするときの最大反復:] (デフォルト: 10)

一覧のプロジェクト順でのビルドは、ここで設定した回数までに制限されます。

●一般>ワークスペース>リンクされたリソース



リンクリソースのON/OFFを切り替えます。

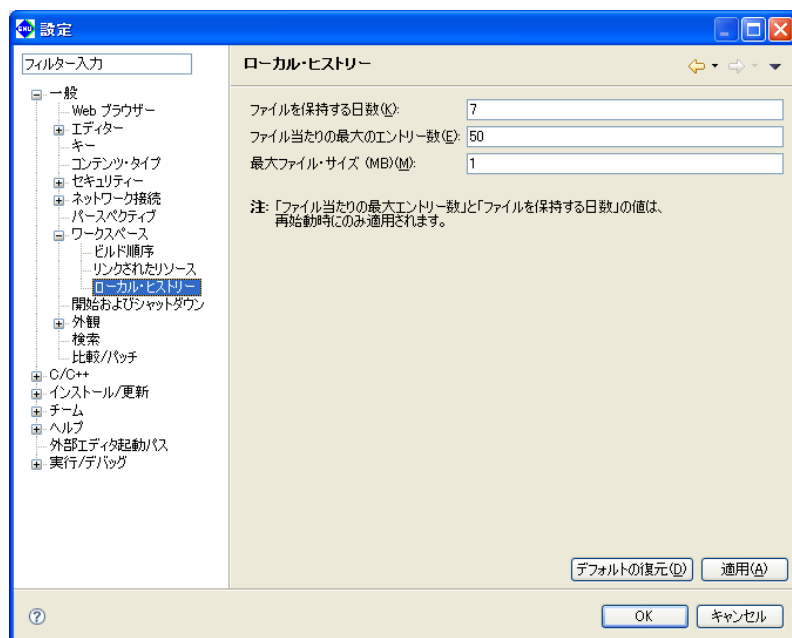
詳細については“5.4.8 プロジェクト内のリソース操作”の“●プロジェクト外のファイルへのリンクを張る”および“●プロジェクト外のフォルダへのリンクを張る”を参照してください。

[リンクされたリソースを使用可能にする](デフォルト：ON)

OFFにすると、リンクリソース機能が使用不可になります。

通常は操作しないでください。

●一般 > ワークスペース > ローカル・ヒストリー



リソースの変更履歴に関する設定を行います。

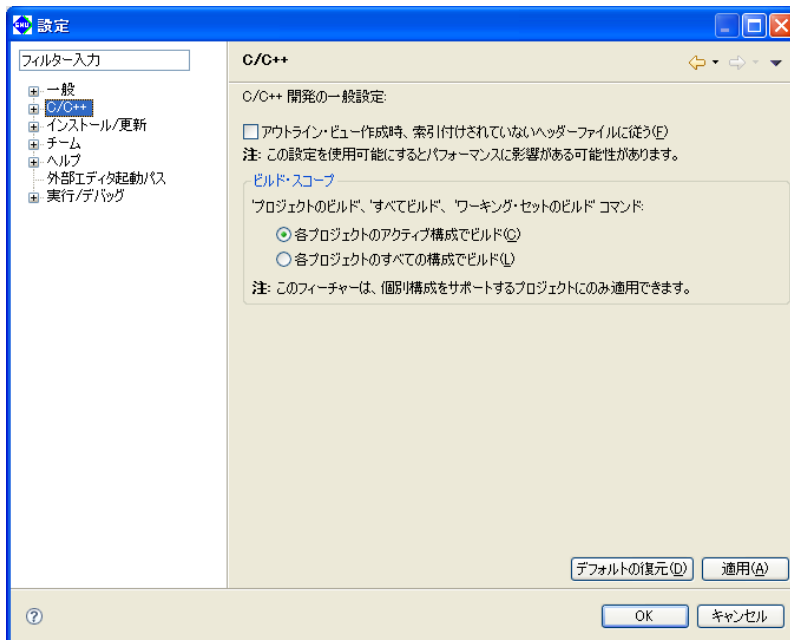
[ファイルを保持する日数:] (デフォルト: 7日)
変更履歴を保持しておく日数を設定します。

[ファイル当たりの最大のエントリー数:] (デフォルト: 50個)
1つのファイルについて、何個の変更履歴を保持するか設定します。

[最大ファイル・サイズ (MB):] (デフォルト: 1MB)
個々の変更履歴を保持しておくファイルの最大サイズを設定します。
この設定を超えるサイズのファイルは履歴が保存されません。

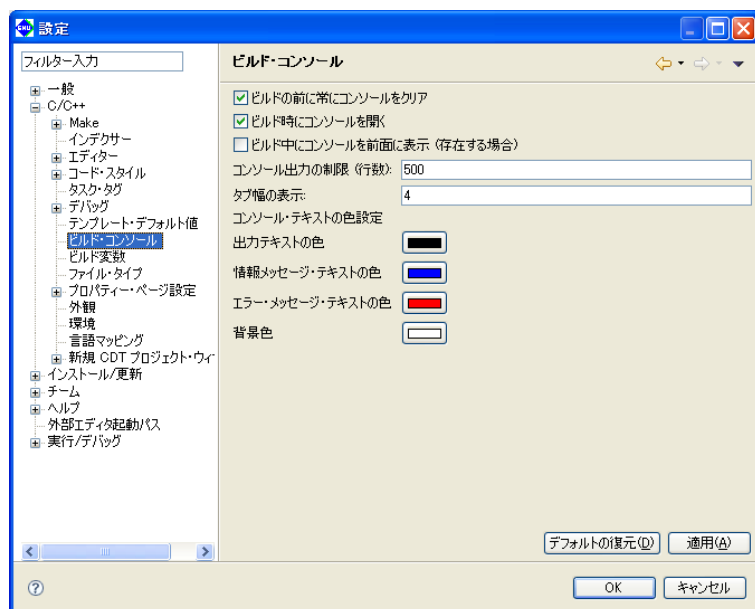
設定した上限を超えた履歴は消去されます。
変更した設定を有効にするには、**IDE**を再起動してください。

● C/C++



この画面は操作しないでください。

●C/C++ > ビルド・コンソール



[コンソール]ビューに関する設定を行います。

[ビルドの前に常にコンソールをクリア] (デフォルト: ON)

このチェックボックスを選択すると、ビルド開始時に[コンソール]ビューがクリアされます。

[ビルド時にコンソールを開く] (デフォルト: ON)

このチェックボックスを選択すると、ビルド時に[コンソール]ビューが開きます。

[ビルド中にコンソールを前面に表示 (存在する場合)] (デフォルト: OFF)

このチェックボックスを選択すると、ビルド時に[コンソール]ビューが前面に表示されます(すでに開いている場合)。

[コンソール出力の制限 (行数):] (デフォルト: 500行)

[コンソール]ビューの最大表示行数を指定します。

[タブ幅の表示:] (デフォルト: 4文字)

[コンソール]ビューの表示におけるタブ幅を文字数で指定します。

[コンソール・テキストの色設定]

[出力テキストの色]

実行したコマンドラインの表示色です。

[情報メッセージ・テキストの色]

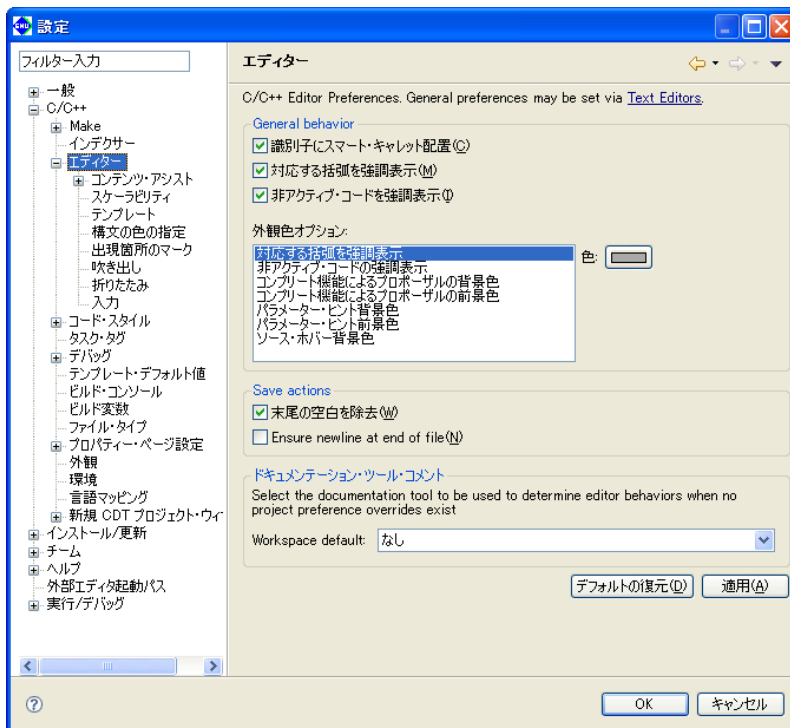
ツールが出力したステータスメッセージなどの表示色です。

[エラー・メッセージ・テキストの色]

ツールが出力したエラーメッセージなどの表示色です。

●C/C++ > エディター

IDEが搭載しているCエディタに関する設定を行います。



[General behavior]

[識別子にスマート・キャレット配置] (デフォルト: ON)

このチェックボックスが選択されると、識別子内部のワード間の境界が表示されます。

[対応する括弧を強調表示] (デフォルト: ON)

このチェックボックスが選択された状態で、括弧({または})の行にカーソルを置くと、対応する括弧が枠で囲まれて表示されます。

[非アクティブ・コードを強調表示] (デフォルト: ON)

このチェックボックスが選択された状態で、アクティブでないコードの行にカーソルを置くと、アクティブでないコードをハイライト表示します。

[外観色オプション:]

コメントやキーワードなどの文字色を設定します。一覧からステートメントの種類を選択し、[色]ボタンで希望の色を選択します。

[Save actions]

[末尾の空白を除去]

ファイル保存時に行末の空白を除去します。

[Ensure newline at end of file]

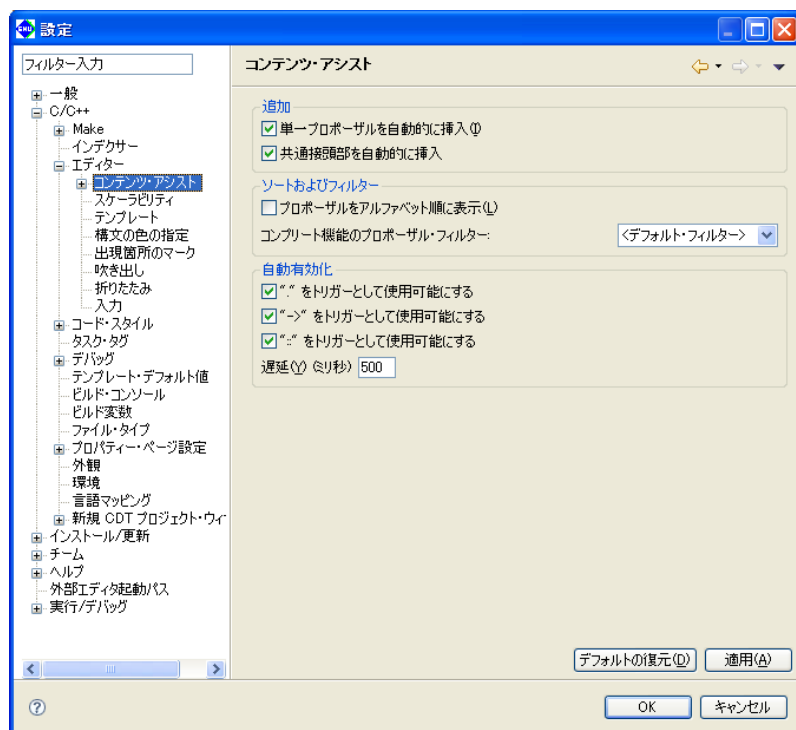
ファイル保存時に最終行が改行で終了するようにします。

[ドキュメンテーション・ツール・コメント]

エディタの表示や動作を決めるドキュメンテーションツールの種類を選択します。エディタのコンテンツアシスト、色分け、コメント生成などの機能を、ここで選択したツールに合わせて設定します。

ここで選択された設定は、プロジェクトに関連付けされていないファイルや、プロジェクトレベルが設定されていないファイルに適用されます。

●C/C++ > エディター > コンテンツ・アシスト



コンテンツアシストに関する設定を行います。
通常は操作しないでください。

[追加]

候補の表示や挿入に関する設定を行います。

[単一プロポーザルを自動的に挿入] (デフォルト：ON)

このチェックボックスを選択すると、候補が1つのみの場合は自動的に入力されます。

[共通接頭部を自動的に挿入] (デフォルト：ON)

このチェックボックスを選択すると、一般的なプレフィックスは自動的に入力されます。

[ソートおよびフィルター]

ソートとフィルタリングに関する設定を行います。

[プロポーザルをアルファベット順に表示] (デフォルト：OFF)

このチェックボックスを選択すると、コンテンツアシストの候補一覧をアルファベット順に表示します。OFFの場合は、位置関係やスコープ、プレフィックスなどの条件により、適合度の高いものから順に表示します。

[自動有効化:]

シンボル(":", ":->", "::-:")の入力により、コンテンツアシストを自動的に起動することができます。この機能を無効にする場合は、チェックボックスをOFFにします。

[":" をトリガーとして使用可能にする] (デフォルト：ON)

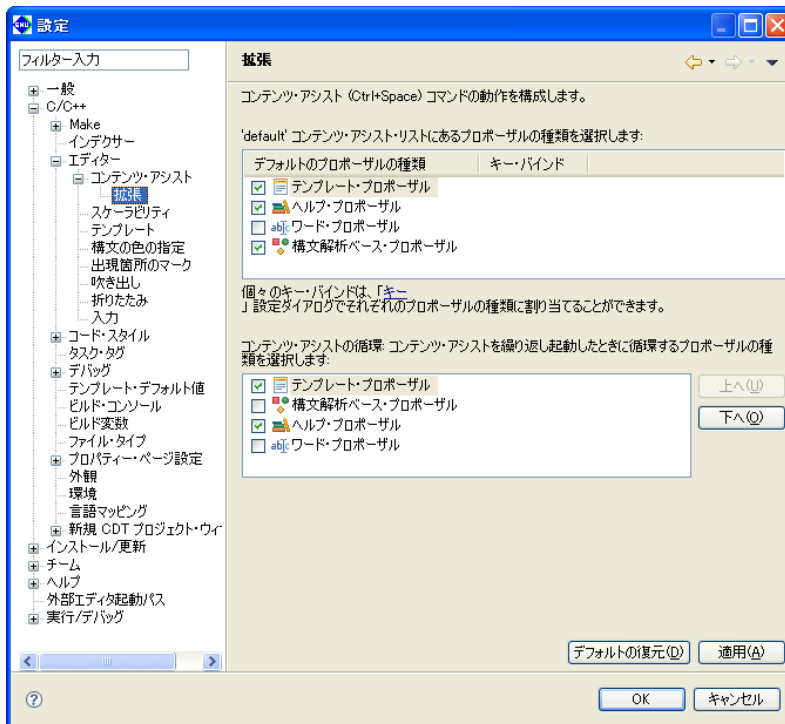
[":->" をトリガーとして使用可能にする] (デフォルト：ON)

["::-:" をトリガーとして使用可能にする] (デフォルト：ON)

[遅延(ミリ秒)] (デフォルト：500ms)

上記のシンボルの入力からコンテンツアシストを自動表示するまでの時間をms単位で指定します。

●C/C++ > エディター > コンテンツ・アシスト > 拡張



コンテンツアシストに関する高度な設定を行います。

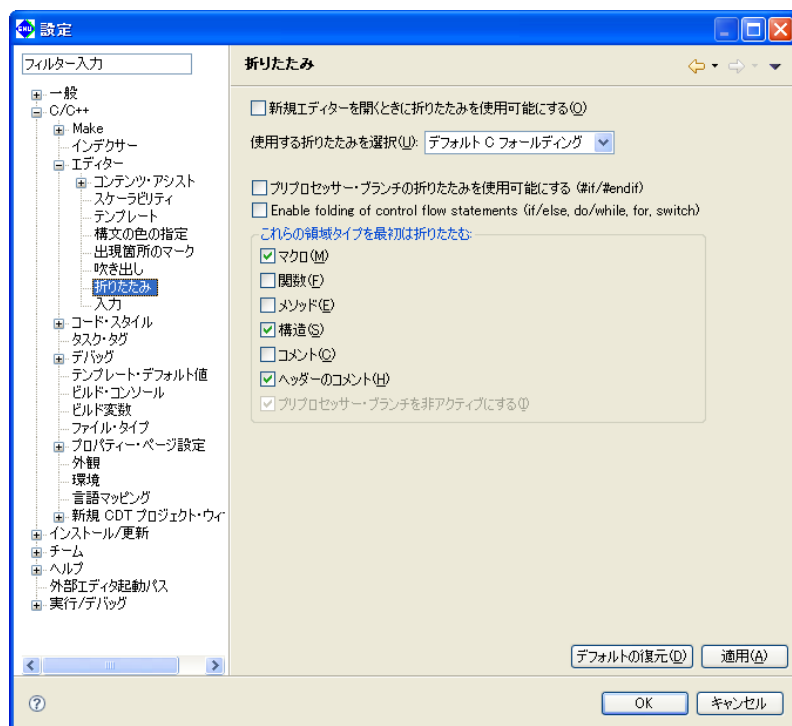
[**'default' コンテンツ・アシスト・リストにあるプロポーザルの種類を選択します:**

標準でコンテンツアシストに含める機能を選択します。

[**コンテンツ・アシストの循環:コンテンツ・アシストを繰り返し起動したときに循環するプロポーザルの種類を選択します:**

コンテンツアシストが繰り返し呼び出されたときに、これに含める機能を選択します。項目を選択して[上へ]/[下へ]ボタンを押すことによって、項目の優先順位を設定することができます。

●C/C++ > エディター > 折りたたみ



折りたたみ機能に関する設定を行います。

[新規エディターを開くときに折りたたみを使用可能にする] (デフォルト：OFF)

このチェックボックスを選択すると、新たに開いたエディタに折りたたみ機能を適用します。

[使用する折りたたみを選択:]

折りたたみ機能の種類を選択します。

[プリプロセッサ・ブランチの折りたたみを使用可能にする (#if/#endif)] (デフォルト：OFF)

このチェックボックスを選択すると、プリプロセッサ分岐の折りたたみを行います。

[Enable folding of control flow statements (if/else, do/while, for, switch)] (デフォルト：OFF)

このチェックボックスを選択すると、フロー制御宣言文の折りたたみを行います。

[これらの領域タイプを最初は折りたたむ:]

エディタを起動時、折りたたんだ状態にするものをチェックします。

[マクロ] (デフォルト：ON)

マクロ

[関数] (デフォルト：OFF)

関数

[メソッド] (デフォルト：OFF)

メソッド

[構造] (デフォルト：ON)

構造

[コメント] (デフォルト：OFF)

コメント

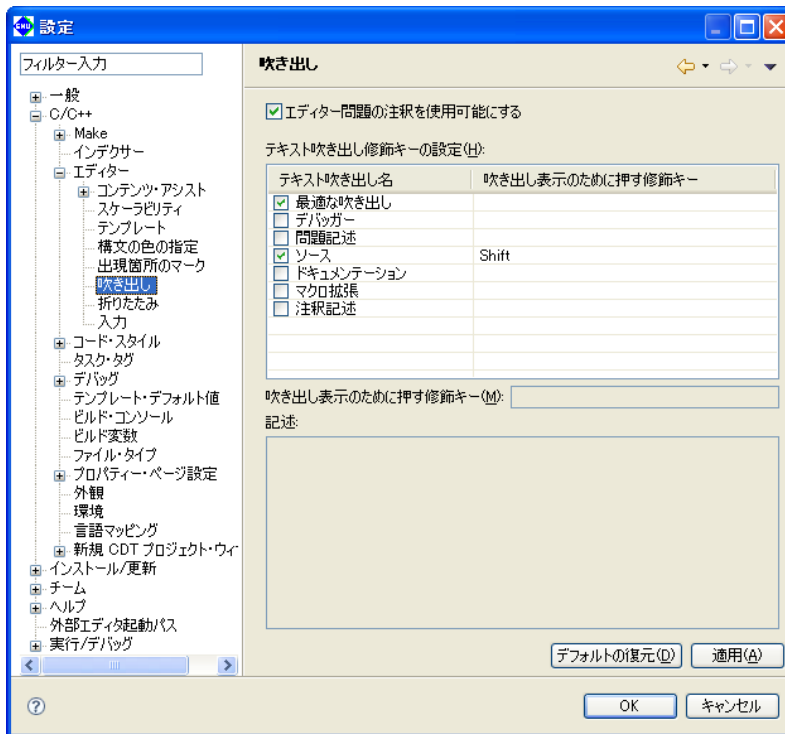
[ヘッダーのコメント] (デフォルト：ON)

ヘッダー

[プロセッサ・ブランチを非アクティブにする] (デフォルト：ON)

アクティブでないプリプロセッサ分岐

●C/C++ > エディター > 吹き出し



吹き出しに関する設定を行います。

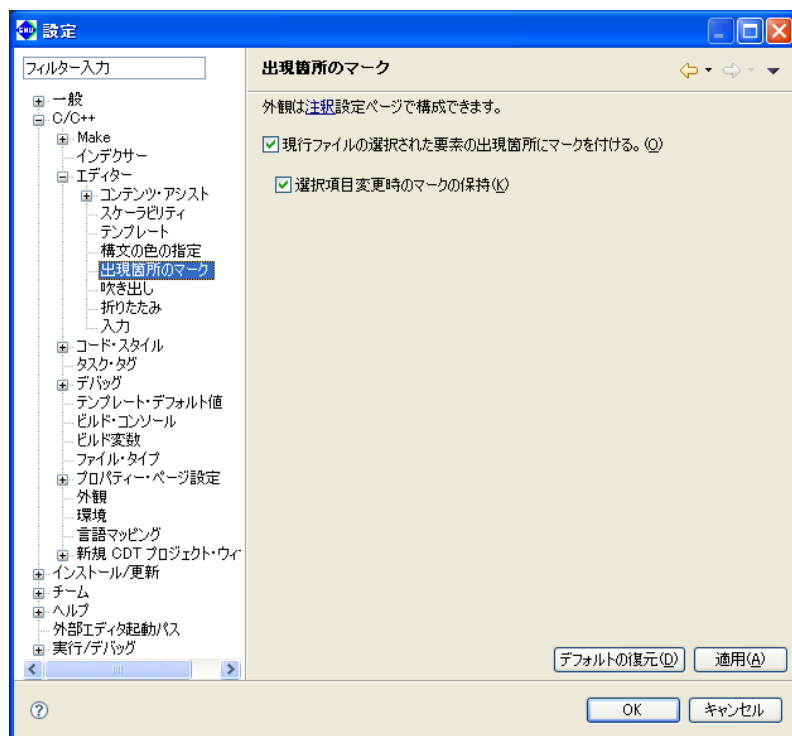
[エディター問題の注釈を使用可能にする] (デフォルト：ON)

このチェックボックスを選択すると、発見された問題がハイライト表示されます。

[テキスト吹き出し修飾キーの設定]

吹き出しに代わるホットキーを選択できます。例えば、[Ctrl]キーを押しながらマウスカーソルを合わせると、対象が元の宣言にリンクされます。

●C/C++ > エディター > 出現箇所のマーク



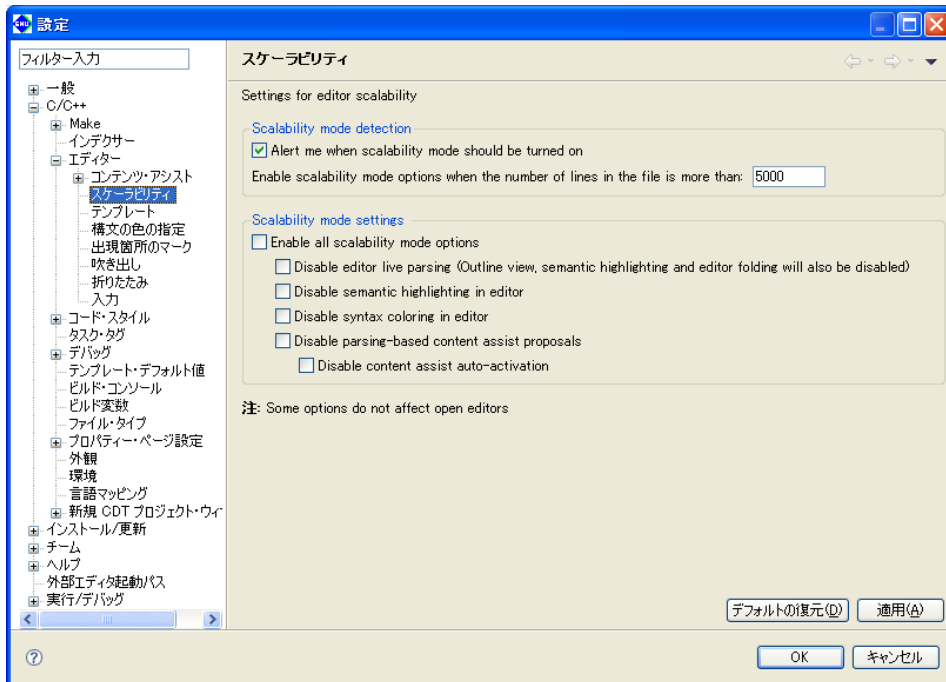
[現行ファイルの選択された要素の出現箇所にマークを付ける。] (デフォルト：OFF)

このチェックボックスを選択すると、変数、機能、メソッド、タイプ、マクロまたは他の要素のマークを表示します。

[選択項目変更時のマークの保持] (デフォルト：ON)

このチェックボックスを選択すると、選択を変更してもマークを保持します。

●C/C++ > エディター > スケーラビリティ



エディタの性能を調節します。

行数の多いファイルなどをエディタで開くと、エディタの強調表示などのパフォーマンスが落ちるため、エディタの表示機能を制限することができます。

通常は、操作する必要はありません。

[Scalability mode detection]

[Alert me when scalability mode should be turned on]

初期設定では、5000行以上のソースファイルをエディタで開くと、エディタのパフォーマンスに関する警告ダイアログが表示されます。

[Scalability mode settings]

[Enable all scalability mode options]

大きなファイルを開いたときにエディタの機能を制限するオプションを有効にします。

[Disable editor live parsing]

エディタのライブ構文解析を無効にします(アウトライン表示、強調表示、折りたたみ表示も無効になります)。

[Disable semantic highlighting in editor]

エディタの強調表示を無効にします。

[Disable syntax coloring in editor]

エディタの構文色分けを無効にします。

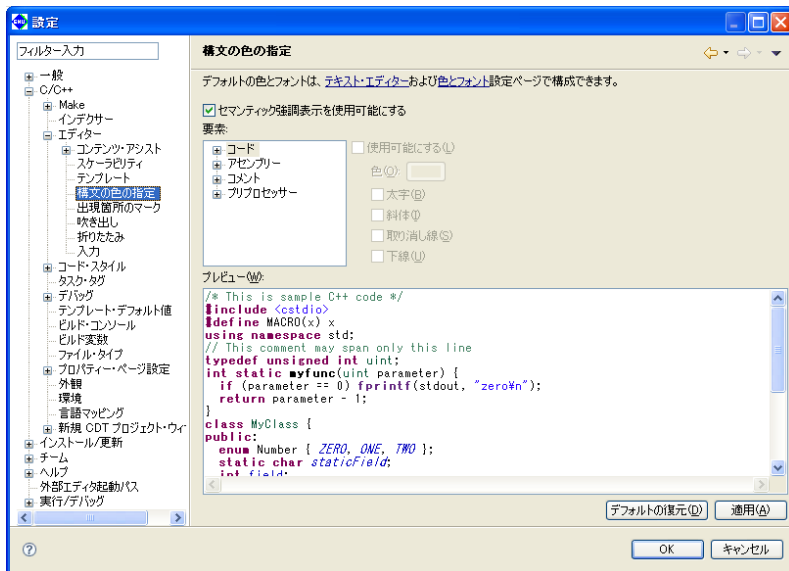
[Disable parsing-based content assist proposals]

構文解析ベースのコンテンツアシストを無効にします。

[Disable content assist auto-activation]

コンテンツアシストの自動起動を無効にします。

●C/C++ > エディター > 構文の色の指定



構文の強調表示の設定を行います。

[セマンティック強調表示を使用可能にする] (デフォルト：ON)

このチェックボックスを選択すると、カラーの設定を有効にします。

[要素:]

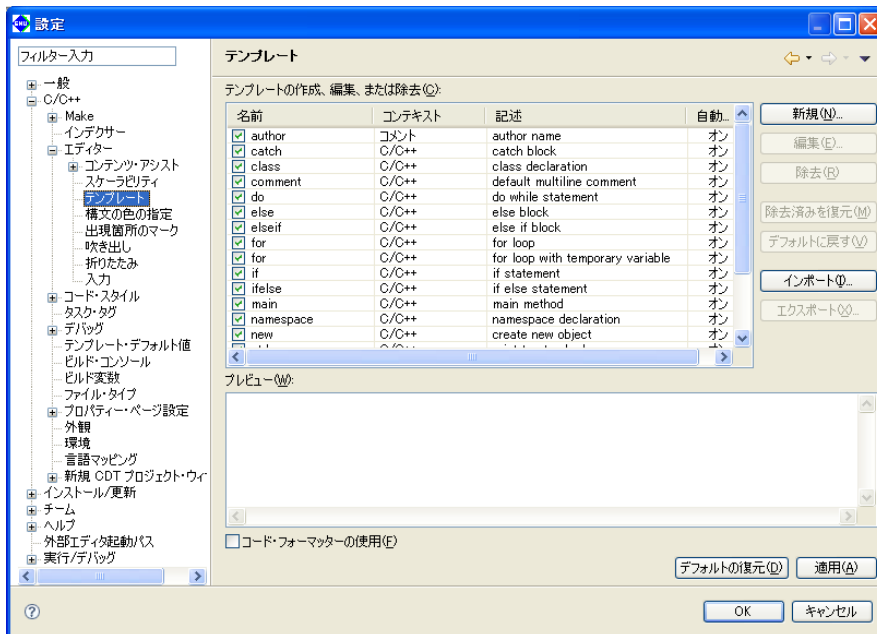
コメントやキーワードなどの文字色を設定します。一覧からステートメントの種類を選択し、[色:] ボタンで希望の色を選択します。以下のチェックボックスで文字のスタイルを変更できます。

[太字]	太字
[斜体]	斜体
[取り消し線]	取り消し線
[下線]	下線

[プレビュー:]

上記の設定を確認するサンプルを表示します。

●C/C++ > エディター > テンプレート



コンテンツアシスト機能で入力するテンプレート(定型書式)を管理します。

[テンプレートの作成、編集、または除去]

定義されているテンプレートの一覧をアルファベット順に表示します。

[名前]

テンプレートの名称で、チェックボックスはテンプレートが有効か無効かを示します。チェックが付いているテンプレートが有効で、コンテンツアシストのリストに表示されます。

[コンテキスト]

記述可能な場所を示します。"C"はC/C++ソース内に記述可能なことを示します。

[記述]

テンプレートの概要説明です。

[自動挿入]

"オン"は候補が1つの場合に自動挿入されることを示します。

[プレビュー:]

一覧内で選択されているテンプレートの内容を表示します。

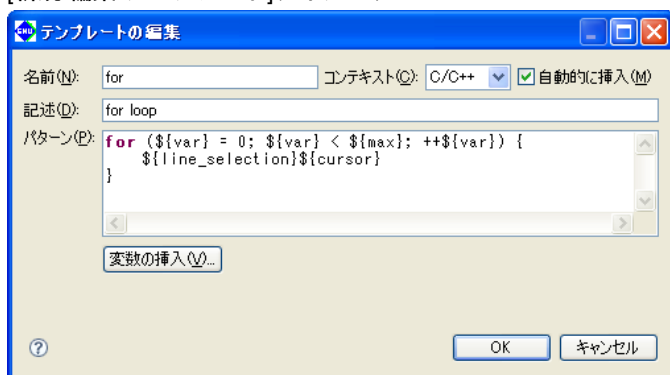
[新規...]

[新規テンプレート]ダイアログが表示され、テンプレートを新規作成できます。

[編集...]

[テンプレートの編集]ダイアログが表示され、一覧内で選択されているテンプレートを編集できます。

[新規/編集 テンプレート]ダイアログ



このダイアログでテンプレートの作成/編集を行います。

[名前:]

テンプレートの名称を入力します。

[コンテキスト:]

コンテキストを選択します。

[自動的に挿入]

自動挿入するかどうかを選択します。

[記述:]

テンプレートの説明を入力します。

[パターン:]

ここにテンプレートを記述します。

[変数の挿入...]

変数の一覧が表示され、ファイル名や日時などを表す変数をテンプレート内に挿入することができます。

[OK]/[キャンセル]

[OK]で入力/編集した内容を確認します。[キャンセル]で作成/編集を中止します。

[除去]

一覧内で選択されているテンプレートを削除します。

[除去済みを復元]

削除したテンプレートを復帰します。

[デフォルトに戻す]

デフォルト状態に戻します。

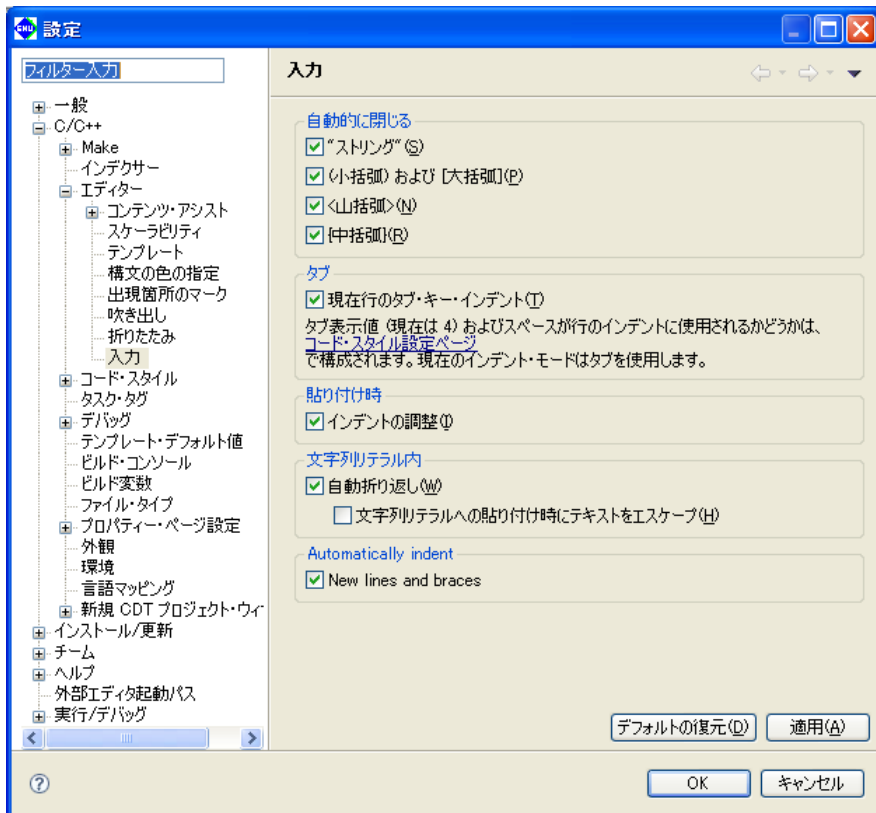
[インポート...]

テンプレートの定義をファイルから読み込みます。

[エクスポート...]

一覧内で選択されているテンプレートの定義内容をファイルに書き出します。このファイルは[インポート...]で読み込み可能です。

● C/C++ > エディター > 入力



タイピング(入力)に関する設定を行います。

[自動的に閉じる]

以下の項目を自動的に閉じます。

["ストリング"](デフォルト：ON)

このチェックボックスを選択すると、文字列を自動的に閉じます。

[(小括弧) および [大括弧]](デフォルト：ON)

このチェックボックスを選択すると、カッコ () と大カッコ [] を自動的に閉じます。

[<山括弧>](デフォルト：ON)

このチェックボックスを選択すると、山カッコ <> を自動的に閉じます。

[[中括弧]](デフォルト：ON)

このチェックボックスを選択すると、中カッコ { } を自動的に閉じます。

[タブ]

[Tab]キーの動作を設定します。

[現在行のタブ・キー・インデント](デフォルト：ON)

このチェックボックスを選択すると、[Tab]キーを押した行をインデントします。

[貼り付け時]

貼り付け時の処理を設定します。

[インデントの調整](デフォルト：ON)

このチェックボックスを選択すると、貼り付けられたテキストのインデントを、現在のインデントに合わせて調整します。

[文字列リテラル内]

文字列の処理を設定します。

[自動折り返し](デフォルト：ON)

このチェックボックスを選択すると、1 行の最大文字数を超えた文字列を、自動的にラップします。

[文字列リテラルへの貼り付け時にテキストをエスケープ](デフォルト：OFF)

このチェックボックスを選択すると、貼り付けられた文字列にある特殊文字をエスケープで囲みます。

[Automatically indent]

自動インデントを設定します。

[New lines and braces](デフォルト：ON)

このチェックボックスを選択すると、新規の行と中カッコ { } を自動的にインデントします。

●その他の設定項目について

ここで説明されていない以下の設定項目は、通常、操作しないでください。

[環境]

[ファイル・タイプ]

[インデクサー]

[言語マッピング]

[Make]

[新規 CDT プロジェクト・ウィザード]

[プロパティ・ページ設定]

[タスク・タグ]

[テンプレート・デフォルト値]

5.10 ダイアログ補足説明

ここでは、特に補足が必要と思われるダイアログについて説明します。本文中に説明のあるもの、Windowsの標準ダイアログまたはそれに準じるもの、[OK]と[キャンセル]ボタンのみの操作を必要とするようなダイアログについては省略しています。

5.10.1 プロジェクトのプロパティ

このダイアログは現在選択されているプロジェクトの各種プロパティの表示と設定変更を行います。[プロジェクト]メニューまたは[C/C++ プロジェクト]/[ナビゲーター]ビューのコンテキストメニューから[プロパティ]を選択すると表示されます。左のプロパティ一覧に14のカテゴリが用意されており、そのクリックによりカテゴリのページを表示します。以下、カテゴリのページ別に説明します。なお、各ページに共通なボタンの機能は次のとおりです。

[デフォルトの復元]

各ページの設定内容をダイアログを開いたときの状態、または[適用]ボタンで設定を確定した状態に戻します。

[適用] *

そのページで設定した内容を適用します。他のプロパティを変更する場合は、[適用]をクリックしてから新たなページに移動してください。

[OK] *

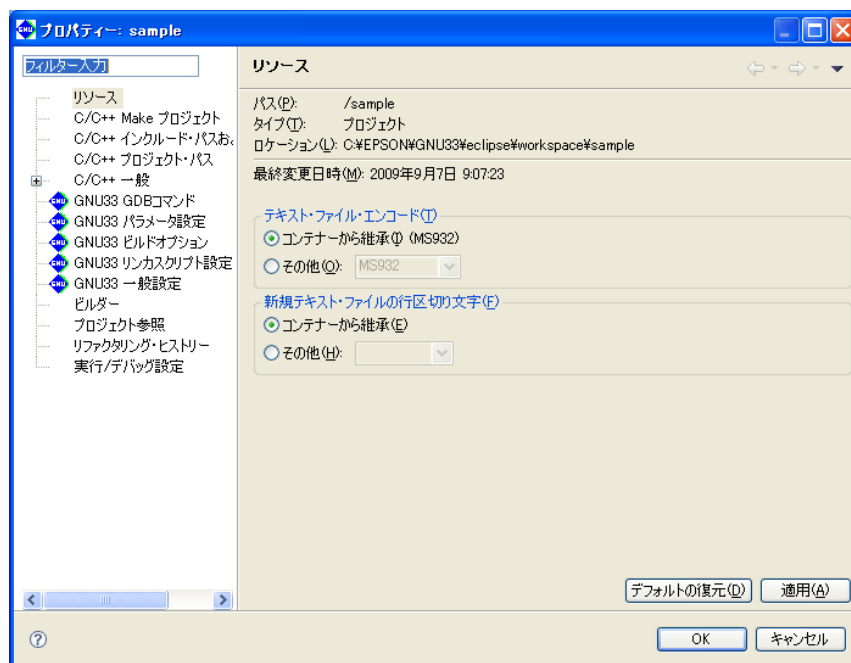
現在のページの設定内容を適用し、ダイアログを閉じます。

[キャンセル]

設定を中止して、ダイアログを閉じます。その前に[適用]ボタンで適用済みの設定内容は元には戻りません。

- * [GNU33 ビルドオプション]または[GNU33 一般設定]のページで設定を変更後に[OK]または[適用]ボタンをクリックした場合は、"clean"ビルド(5.7.9節参照)を実行するかどうかを確認するダイアログが表示され、以前の設定で生成されているファイルの削除(およびリビルド)が行えます。

● リソース



プロジェクトディレクトリの位置情報を表示します。また、ソースファイルなどのテキストエンコード形式と行の区切り文字が設定できます。

[テキスト・ファイル・エンコード]

テキストのエンコード形式を設定します。

[コンテナから継承 (*1)] (*1: Cp1252、MS932等、Windowsの言語対応により変わります。)

Windows標準の文字コードセットです。(デフォルト)

[その他:]

その他の文字コードを選択します。

[新規テキスト・ファイルの行区切り文字]

行の区切り文字を選択します。この選択は、この後に作成されるファイルに有効で、既存のファイルを開いても変更されません。

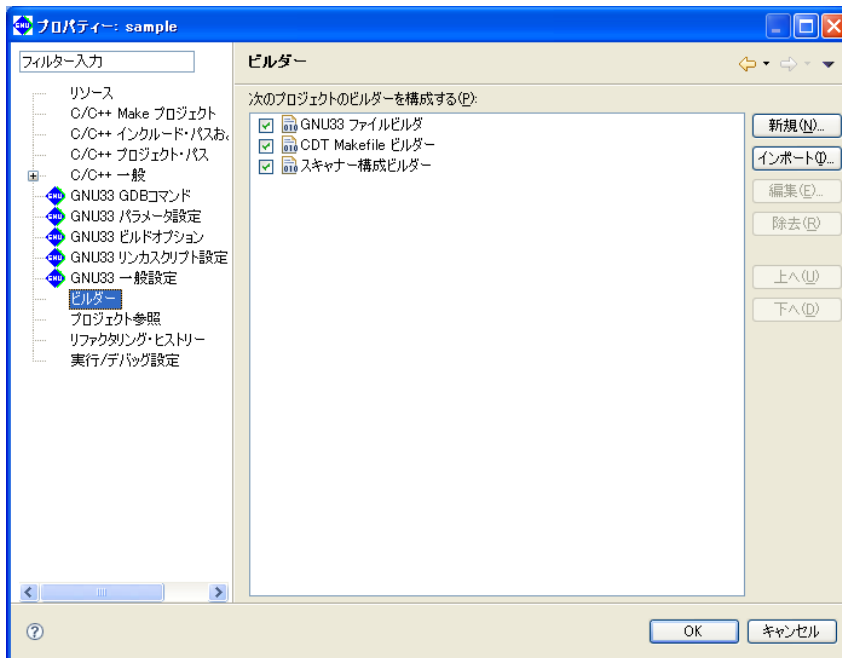
[コンテナから継承]

Windows標準の区切り文字を使用します。(デフォルト)

[その他:]

その他の区切り文字を選択します。

●ビルダー

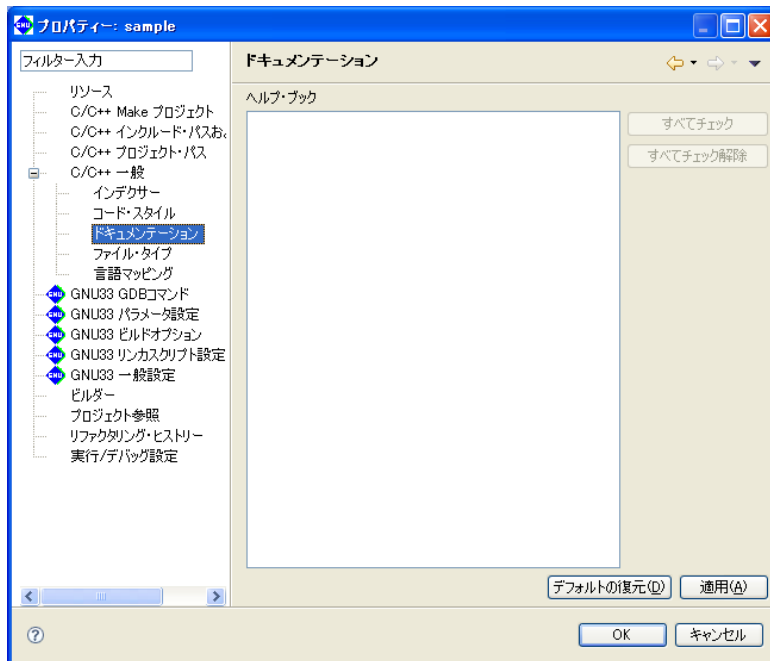


プロジェクトで使用するビルダを選択します。

[GNU33 ファイルビルダ]

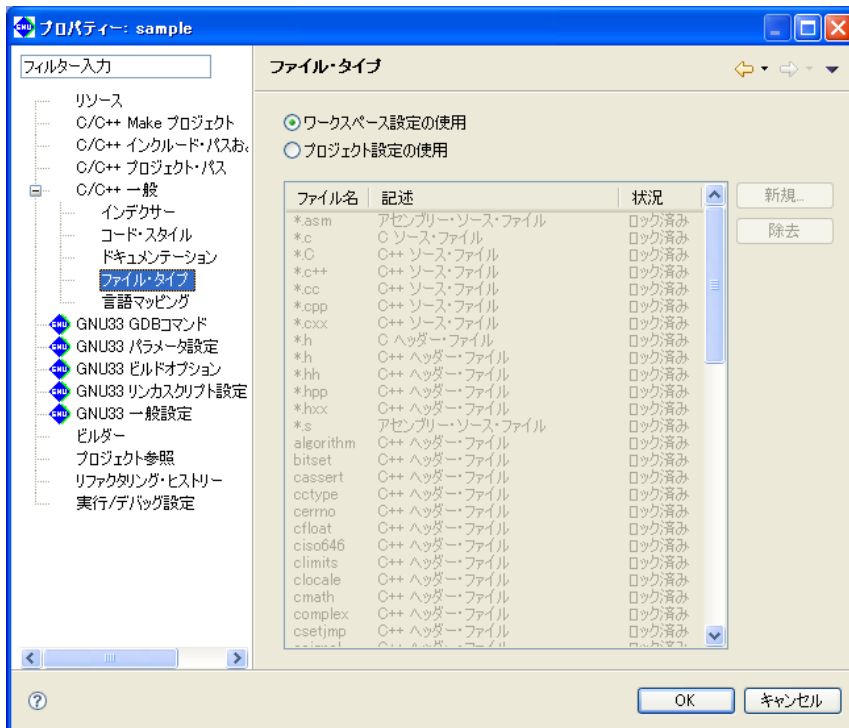
ビルド実行時にmakeファイルなどビルドに必要なファイルの自動生成を行わない場合にチェックを外します("5.7.10 ユーザ作成のmakeファイルを使用するには"参照)。それ以外は、本ページの設定を変更しないでください。

●C/C++ 一般 > ドキュメンテーション



プロジェクトに使用するヘルプ用ドキュメントを選択します。**IDE**では使用しません。

●C/C++ 一般 > ファイル・タイプ



C/C++リソースとして扱うファイルの名称や拡張子を定義します。**IDE**では、変更する必要はありません。

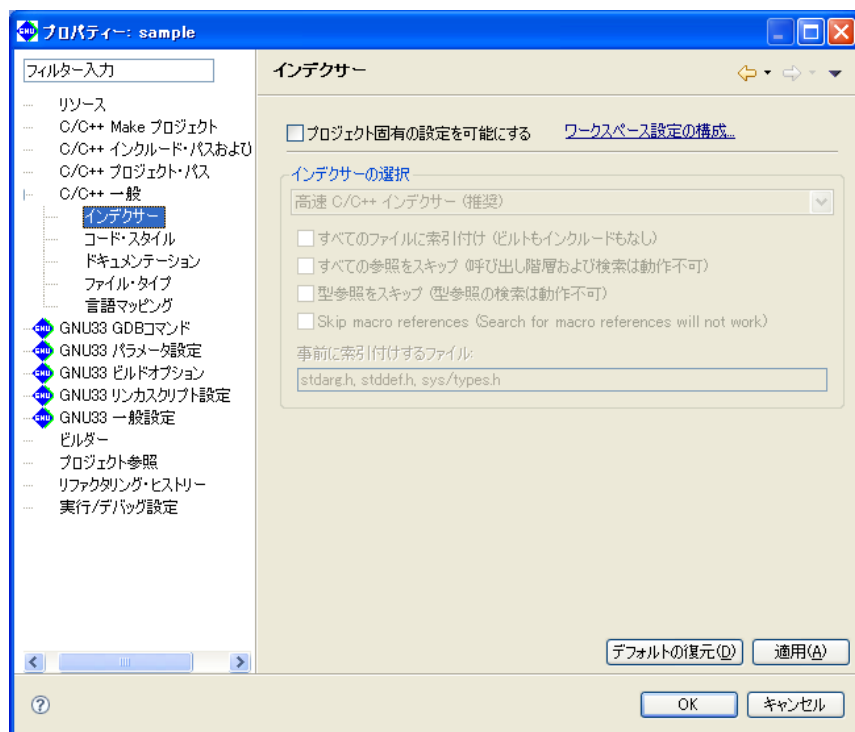
[ワークスペース設定の使用]

[設定]ダイアログの[C/C++]>[ファイル・タイプ]ページに記載されているファイル形式セットを使用します。

[プロジェクト設定の使用]

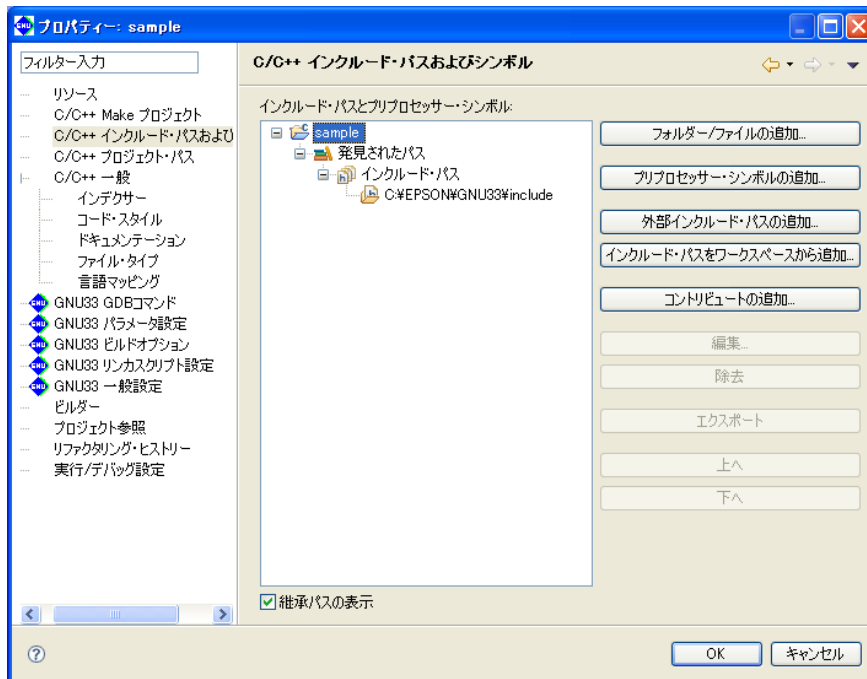
プロジェクト独自のファイル形式セットを使用する場合にラジオボタンを選択します。新しいファイル拡張子は[新規...]ボタンで追加します。不要な拡張子とファイル名は、一覧から選択して[除去]ボタンで削除できます。

●C/C++ 一般 > インデクサー



C/C++の検索やコンテンツアシストに使用するインデクサーに関する設定を行います。このページの設定は変更しないでください。

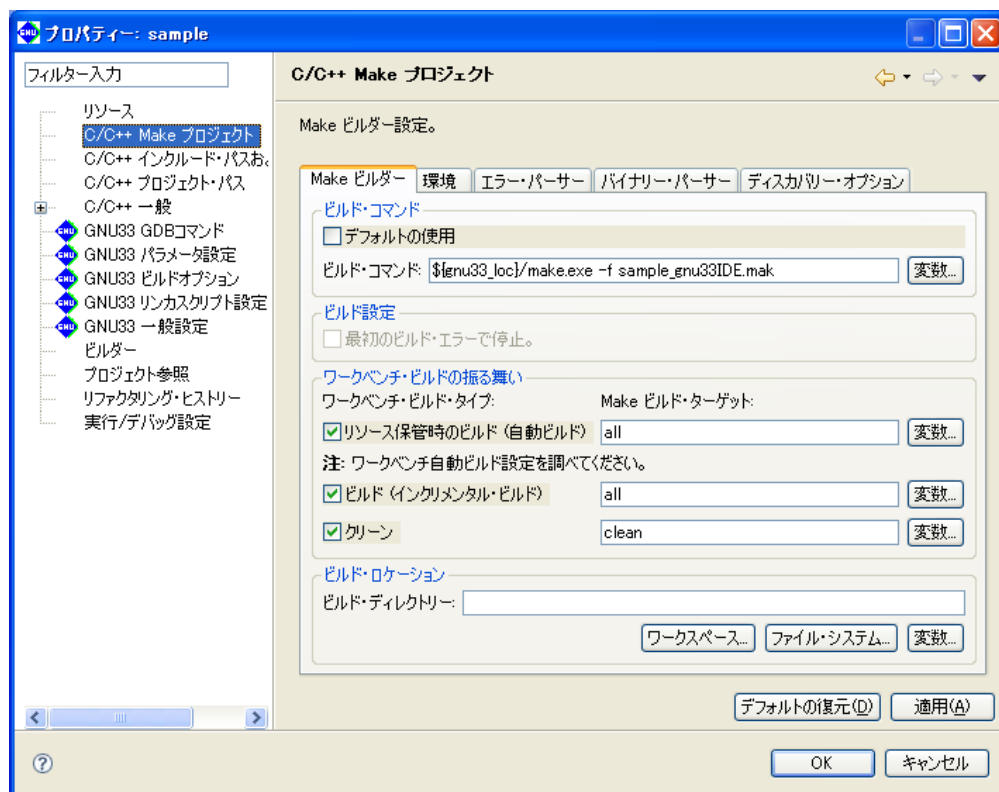
●[C/C++ インクルード・パスおよびシンボル]



インクルードファイルを検索するパスやプリプロセッサ用のシンボルを定義します。
一度でもビルドを実行すると、そのときに検索したインクルードフォルダとプリプロセッサに渡したマクロ定義がツリーリスト内の[発見されたパス]に追加されます。

注: 通常、このページは操作しないでください。

●C/C++ Make プロジェクト



ビルドを行う **make.exe** に関する設定を行います。ユーザ独自に作成した **make** ファイルを使用する場合に、[Make ビルダー] タブのページを変更します。それ以外のタブのページは変更しないでください。

[ビルド・コマンド]

[デフォルトの使用]

make.exe のコマンドラインを編集する場合は非選択とします。

[ビルド・コマンド:]

make.exe のコマンドラインを編集します。(make ファイル名をユーザが作成したファイルに変更します。)

[ビルド設定]

[最初のビルド・エラーで停止。]

選択すると、エラーが発生した時点でビルドを中止します。このチェックボックスは[デフォルトの使用]を選択した場合に有効です。

[ワークベンチ・ビルドの振る舞い]

[リソース保管時のビルド (自動ビルド)]

自動ビルド時に呼び出す **make** ファイル内のターゲットを指定します。デフォルト設定では "all" を呼び出します。(IDE のデフォルト設定では使用しません。)

[ビルド (インクリメンタル・ビルド)]

ビルド実行時に呼び出す **make** ファイル内のターゲットを指定します。デフォルト設定では "all" を呼び出します。

[クリーン]

クリーン実行時に呼び出す **make** ファイル内のターゲットを指定します。デフォルト設定では "clean" を呼び出します。

[ビルド・ロケーション]

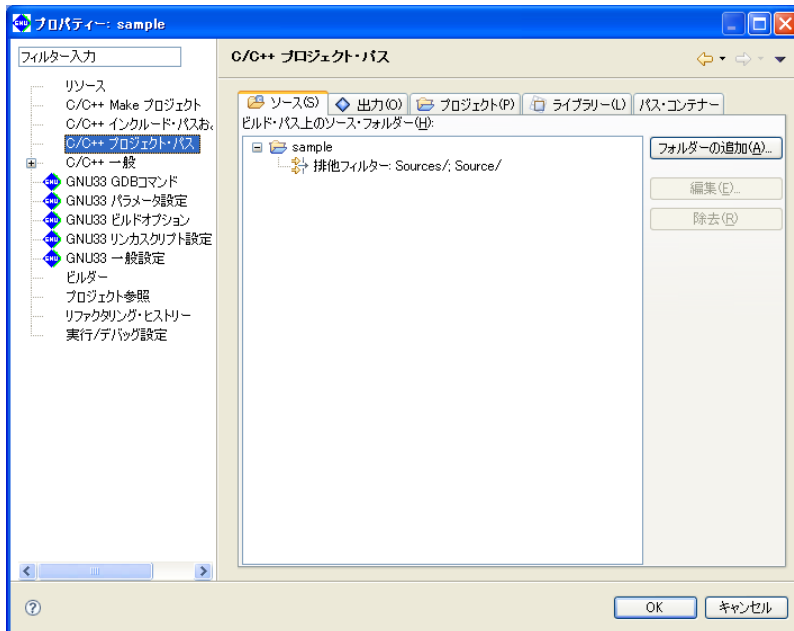
ビルド時の作業ディレクトリを指定するフィールドですが、特に指定する必要はありません。

ユーザ独自に作成した **make** ファイルを使用するには、このページ以外の操作も必要です。詳細については、"5.7.10 ユーザ作成の **make** ファイルを使用するには" を参照してください。

●C/C++プロジェクト・パス

ソースフォルダを設定します。[ソース]タブ以外の設定は不要です。

[ソース]タブ



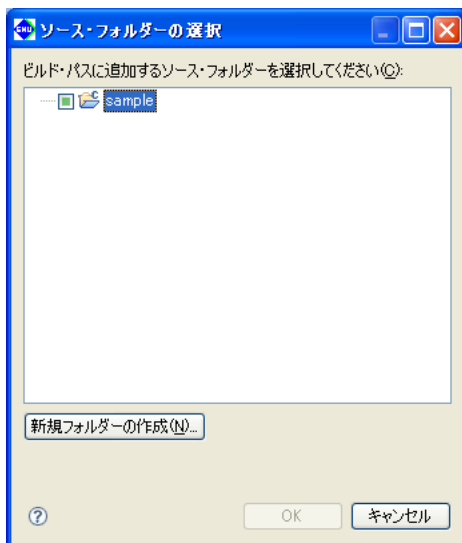
ソースフォルダを設定します。ここで設定したフォルダ内のソースファイルがビルド用のmakeファイル内に記述されます。

[ビルド・パス上のソース・フォルダ:]

現在のソースフォルダの一覧です。デフォルト設定でプロジェクトフォルダがリストされます。

[フォルダーの追加...]

一覧にソースフォルダを追加します。



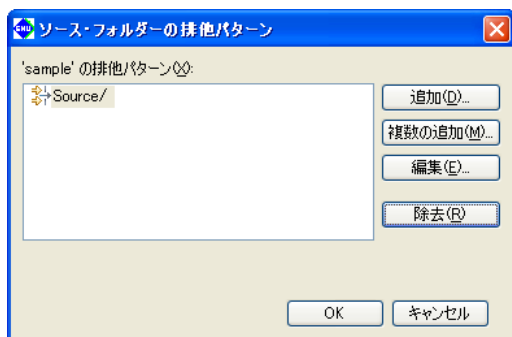
[ソース・フォルダーの選択]ダイアログが表示されます。

ツリーリストからソースフォルダを選択します。または、[新規フォルダーの作成...]でソースフォルダを新規作成するか、プロジェクトフォルダ外にある既存のソースフォルダをリンクします。

[編集...]

ソースフォルダ内のC/C++リソースの中で、ソースファイルと見なさない(makeファイルに含めない)ファイル名パターンを編集します。

ソースフォルダツリーリストから編集するソースフォルダの[排他フィルター]を選択して、このボタンをクリックすると、[ソース・フォルダーの排他パターン]ダイアログが表示されます。

**[<ソースフォルダー>の排他パターン:]**

ソースファイルと見なさない(makeファイルに含めない)ファイル名パターンの一覧を表示します。

[追加...]

ファイル名パターンを追加します。"?"を1文字のワイルドカード、"*"を文字列のワイルドカードとして使用できます。

例: src?.c ?の箇所の1文字は任意(src1.c、src2.c等)

test.* 拡張子は任意の文字列(test.c、test.s等)

[複数の追加...]

ソースフォルダ内のファイル一覧が表示されますので、その中からソースと見なさないファイルを選択します([Ctrl]キーの併用で一度に複数のファイルを選択可能)。

[編集...]

一覧から選択したファイルパターンを編集します。

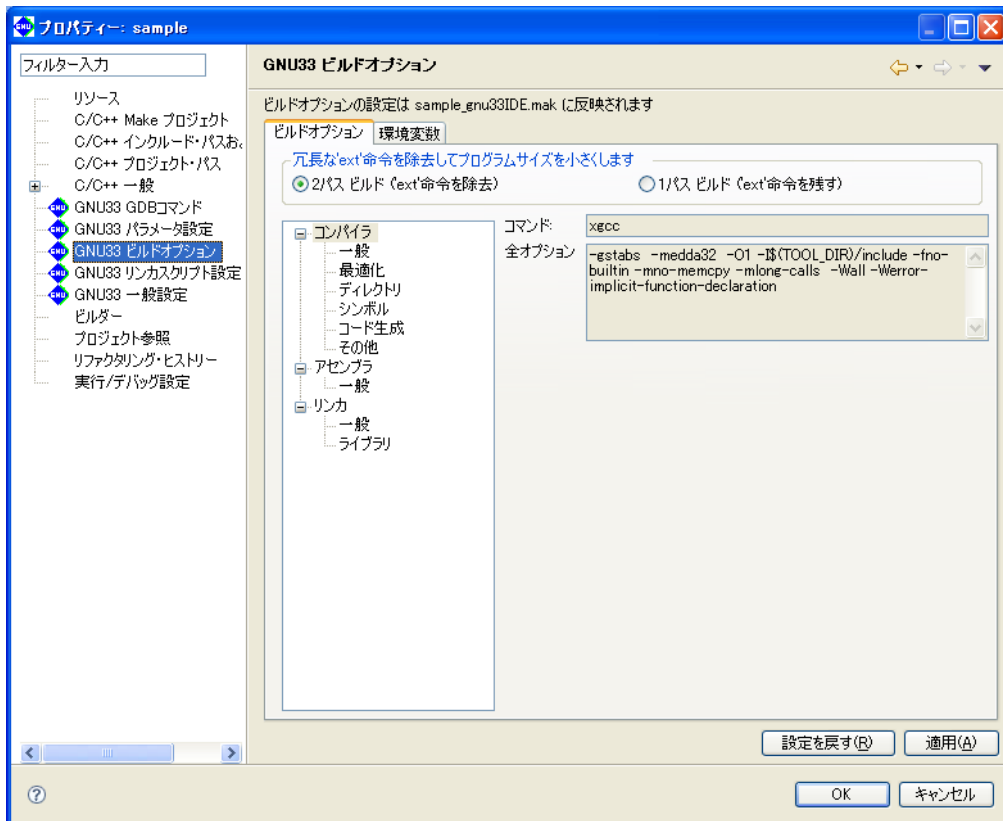
[除去]

一覧から選択したファイルパターンを削除します。

[除去]

一覧から選択したソースフォルダの登録を削除します。ファイルシステムからは削除されません。

●GNU33 ビルドオプション



コンパイラ、アセンブラ、リンカのコマンドラインオプションを設定します。

また、2パスメイク(拡張命令の最適化)を行うか、通常の1パスメイクを行うか、以下のラジオボタンで選択できます。

[2パス ビルド]

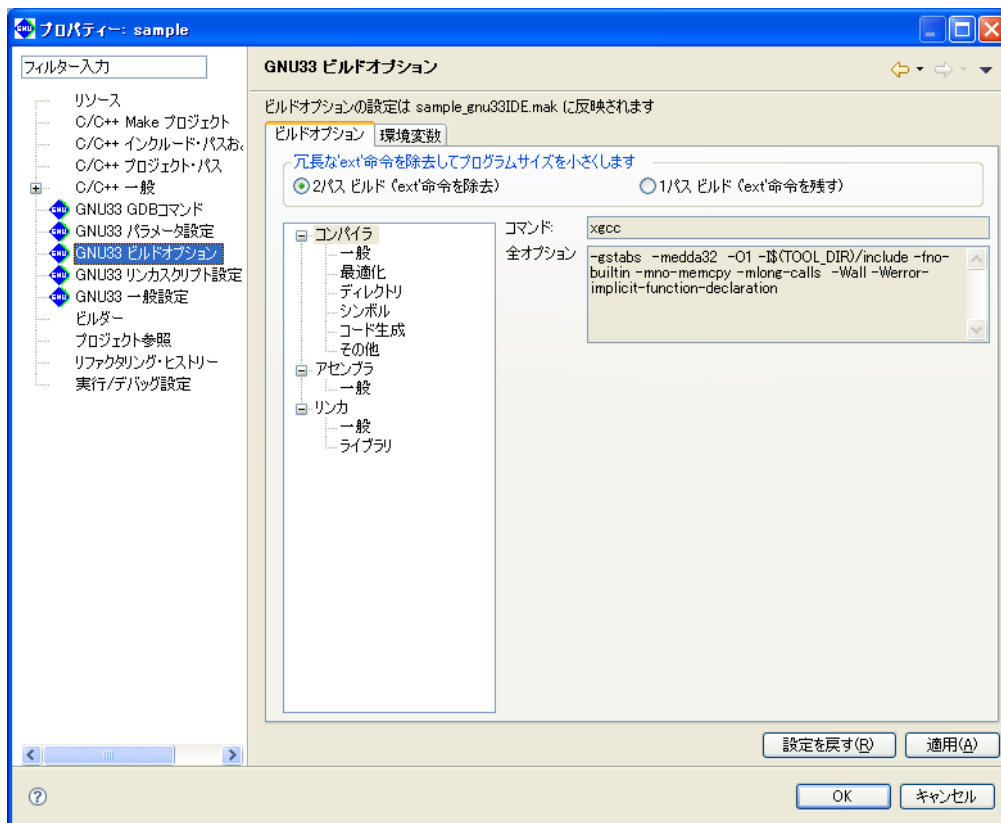
このボタンを選択すると、拡張命令を最適化するための2パスメイクファイルが生成されます。ビルド時間は若干増えますが、プログラムサイズは小さくなります。

[1パス ビルド]

このボタンを選択すると、ext命令を削除しない1パスメイクファイルが生成されます。

- * [GNU33 ビルドオプション]のページで設定を変更後に[OK]または[適用]ボタンをクリックした場合は、“clean”ビルド(5.7.9節参照)を実行するかどうかを確認するダイアログが表示され、以前の設定で生成されているファイルの削除(およびリビルド)が行えます。

[ビルドオプション]タブ>[コンパイラ]



コンパイラオプションの設定内容を表示します。

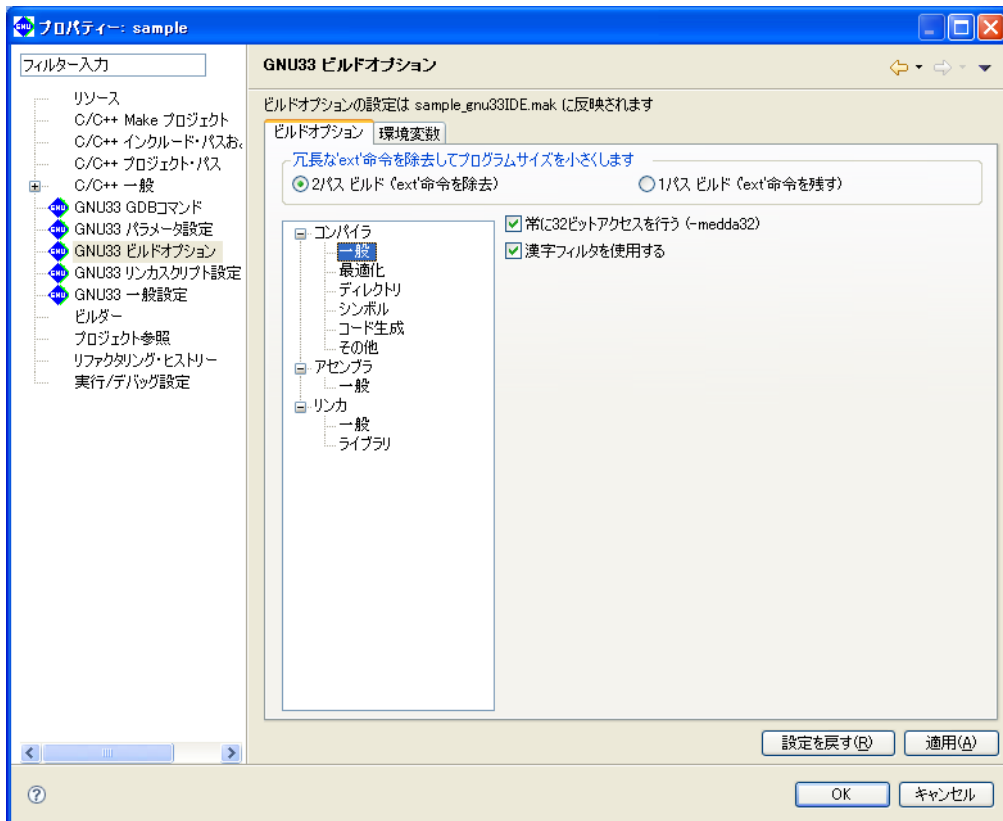
[コマンド:]

C/C++コンパイラのプログラム名を表示します。

[全オプション]

現在設定されているコンパイラオプションを表示します。

[ビルドオプション]タブ>[コンパイラ]>[一般]



このページではコンパイラの基本オプションを選択します。

[常に32ビットアクセスを行う(-medda32)]

このオプションを指定すると、デフォルトデータエリアを使用せずに、32ビットアクセス用のコードを出力します。`-fPIC`オプション([コード生成]ページ)との同時指定はできません。

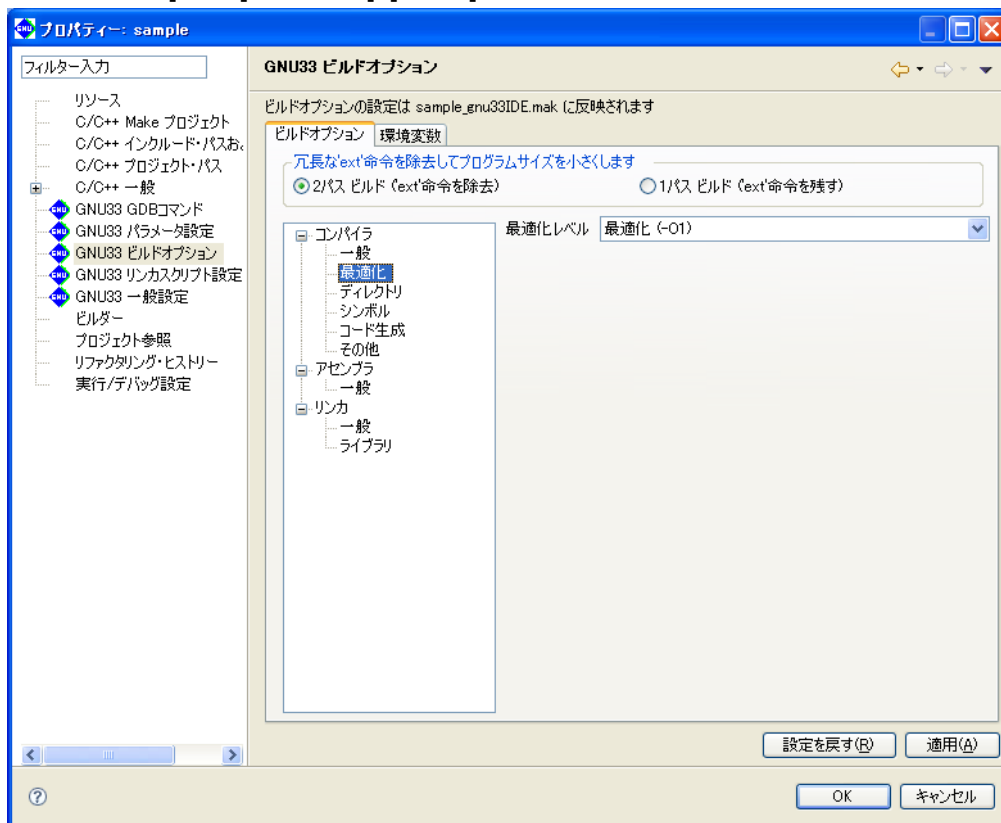
[漢字フィルタを使用する]

このオプションをONにした場合は、コンパイル時にソース内のシフトJISコードを適切に読み込みます。

OFFにした場合は、コンパイラ呼び出し時に`-mno-sjis-filt`オプションが付与され、上記の処理は行われません。

このチェックボックスのデフォルト設定状態は、IDEを起動したOSの言語により決まります。日本語版OS環境でIDEを起動した場合はON、その他の言語ではOFFに設定されます(OFF時は、コンパイル時に`-mno-sjis-filt`オプションが付与されます)。

[ビルドオプション]タブ>[コンパイラ]>[最適化]

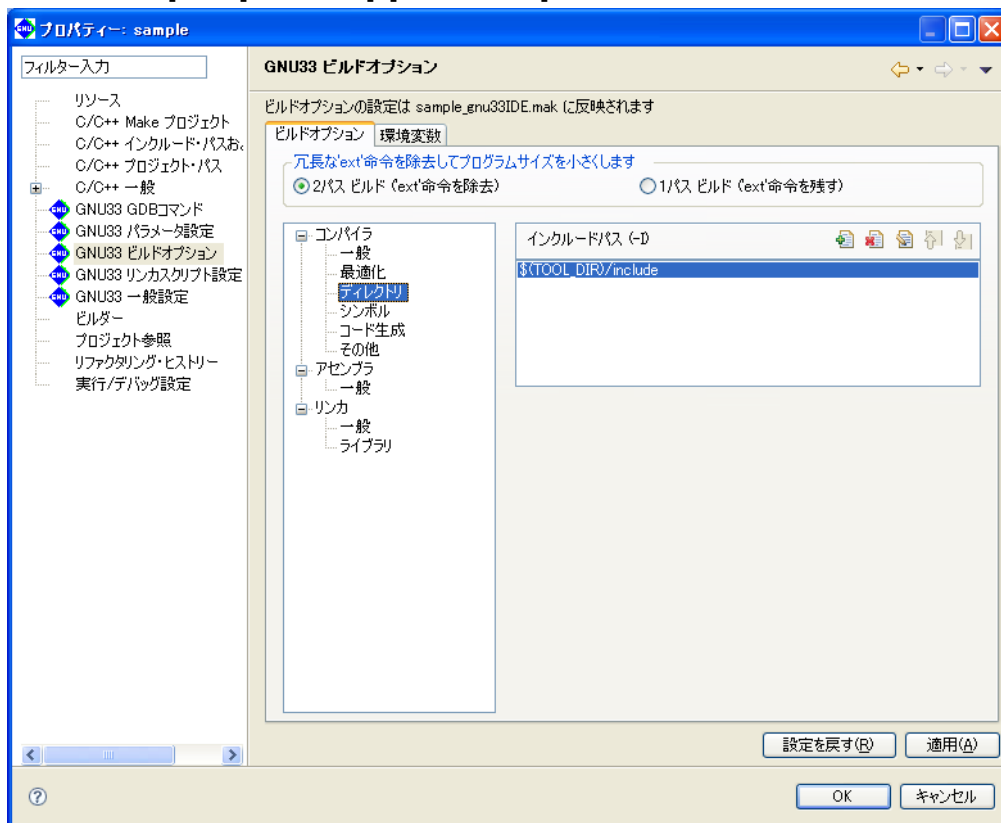


このページではコンパイラの最適化オプションを選択します。

[最適化レベル]

最適化のレベル(-O0~-O3、-Os)を選択します。

[ビルドオプション]タブ>[コンパイラ]>[ディレクトリ]



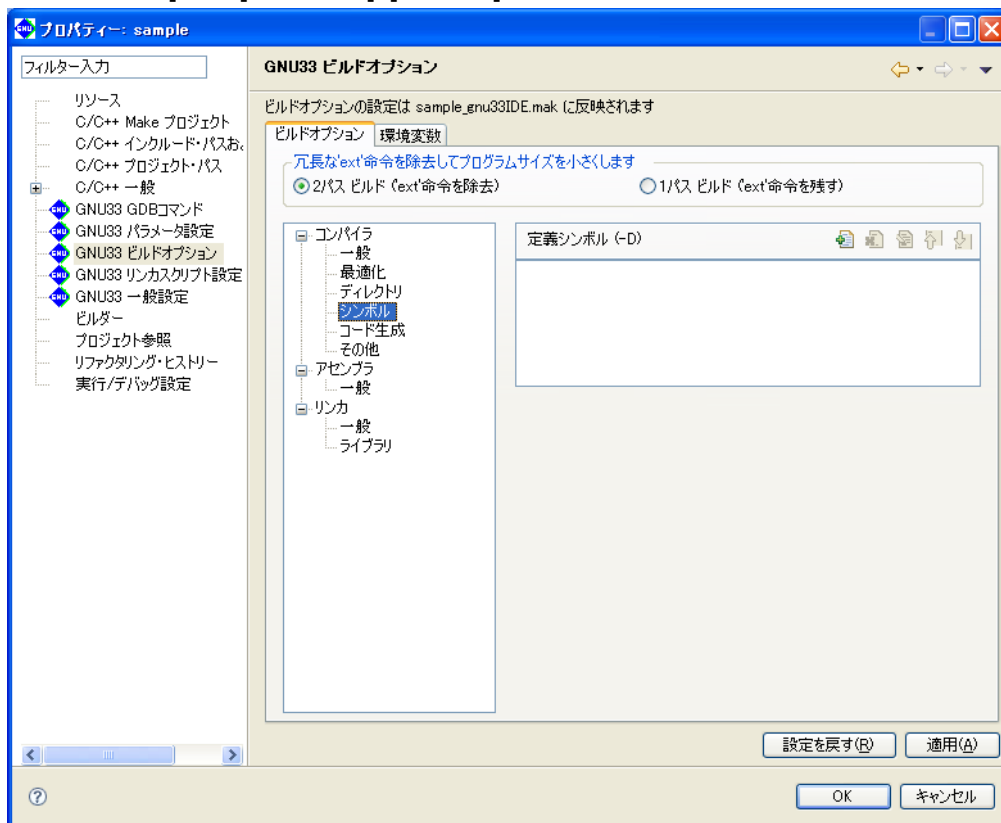
このページではコンパイラの検索パスオプションを設定します。

[インクルードパス (-I)]

インクルードファイルの検索パスを設定します。ボタンの機能は以下のとおりです。

 [追加...]	ディレクトリを追加します。パスを入力あるいは[ファイル・システム...]ボタンで選択するダイアログが表示されます。
 [削除]	リスト内で選択したパスを削除します。
 [編集...]	リスト内で選択したパスを編集します。パスを書き換えるダイアログが表示されます。
 [上へ移動]	リスト内で選択したパスを一覧内の一つ上に移動します。インクルードファイルは一覧内の上にあるパスから順に検索されます。
 [下へ移動]	リスト内で選択したパスを一覧内の一つ下に移動します。

[ビルドオプション]タブ>[コンパイラ]>[シンボル]



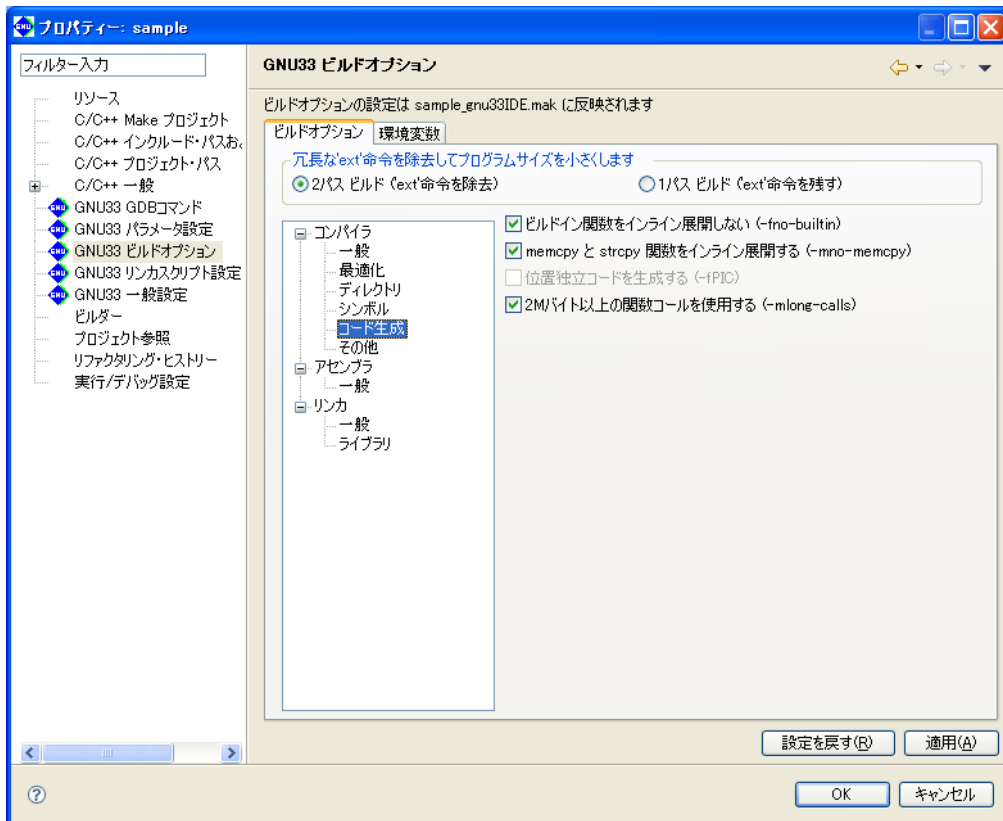
このページではコンパイラのマクロ定義オプションを設定します。

[定義シンボル (-D)]

マクロ名と置き換え文字を指定します。ボタンの機能は以下のとおりです。

 [追加...]	マクロ定義を追加します。マクロ定義を入力するダイアログが表示されますので、次の形式で入力します。 <マクロ名> または <マクロ名>=<置き換え文字>
 [削除]	リスト内で選択したマクロ定義を削除します。
 [編集...]	リスト内で選択したマクロ定義を編集します。マクロ定義を書き換えるダイアログが表示されます。
 [上へ移動]	リスト内で選択したマクロ定義を一覧内の一つ上に移動します。
 [下へ移動]	リスト内で選択したマクロ定義を一覧内の一つ下に移動します。

[ビルドオプション]タブ>[コンパイラ]>[コード生成]



このページではコンパイラのコード生成に関するオプションを選択します。

[ビルドイン関数をインライン展開しない (-fno-builtin)]

このオプションを指定すると、ビルトイン関数が無効となり、必ずコールされます。
対象の関数については、"6.3.2 コマンドラインオプション"を参照してください。

[mempcpy と strcpy 関数をインライン展開する (-mno-memcpy)]

このオプションを指定すると、mempcpy () および strcpy () 関数をインライン展開します。

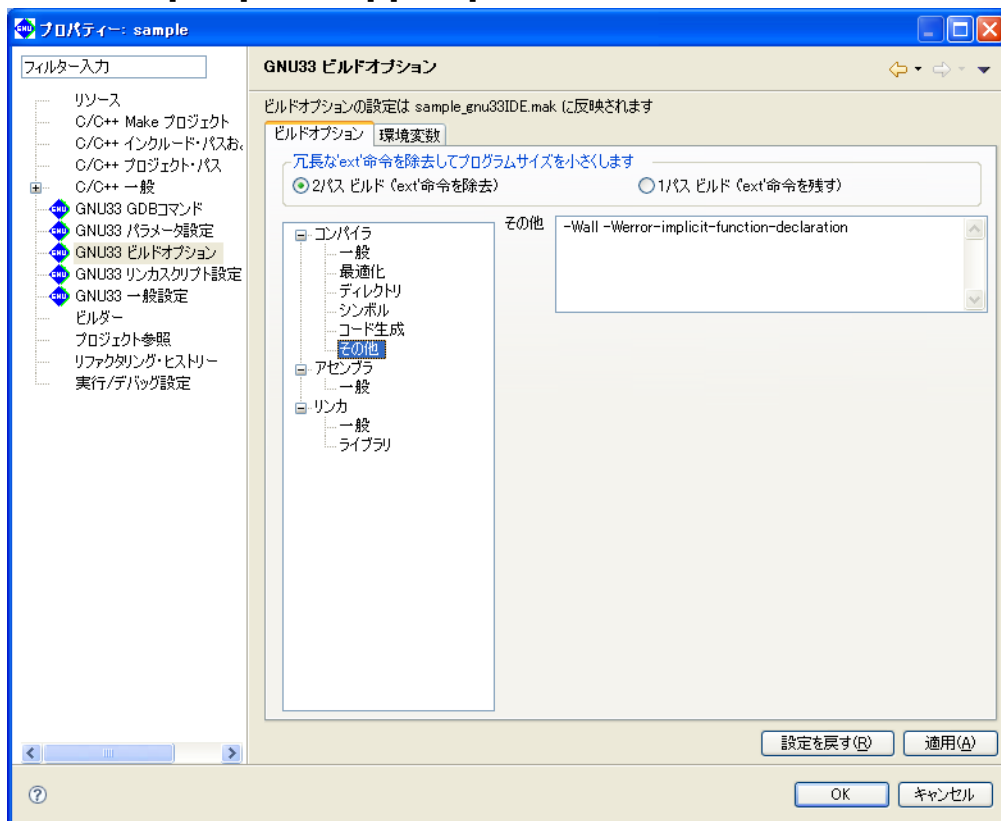
[位置独立コードを生成する (-fPIC)]

このオプションを指定すると、位置独立コードを生成します。
-medda32 オプション([一般]ページ)との同時指定はできません。

[2Mバイト以上の関数コールを使用する (-mlong-calls)]

2Mバイト以上離れた関数コールを行います。

[ビルドオプション]タブ>[コンパイラ]>[その他]



このページではコンパイラのその他のオプションを設定します。

[その他](デフォルト: -Wall -Werror-implicit-function-declaration)

その他のオプションを直接入力します。オプション間には1個以上のスペースを挿入してください。CPUタイプに"S1C33401"を選択した場合は、-mc33advと-mc33401オプションもここで指定されます。同様に、CPUタイプに"S1C33PE"を選択した場合は、-mc33peオプションがここで指定されます。

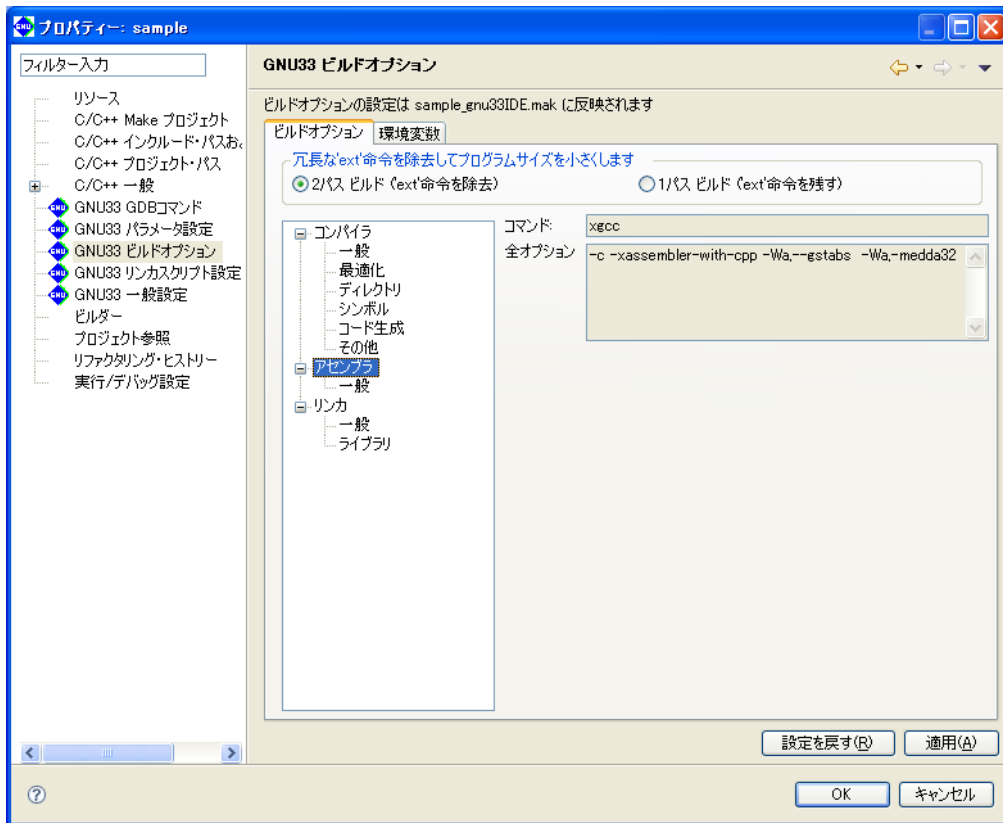
2パスメイクを選択した場合、コンパイル時は自動的に-sオプション指定でコンパイルしてアセンブリファイル(拡張子.ext0)を出力し、このファイルをアセンブラでアセンブルします。

[その他]には、コンパイラの-sオプション、-cオプションを指定する必要はありません。

Cソースのアセンブリイメージは、<Cソースファイル名.ext0>ファイルで確認できます。

また、-gstabsオプションも自動的に指定されます。

[ビルドオプション]タブ>[アセンブラ]



アセンブルオプションの設定内容を表示します。

[コマンド:]

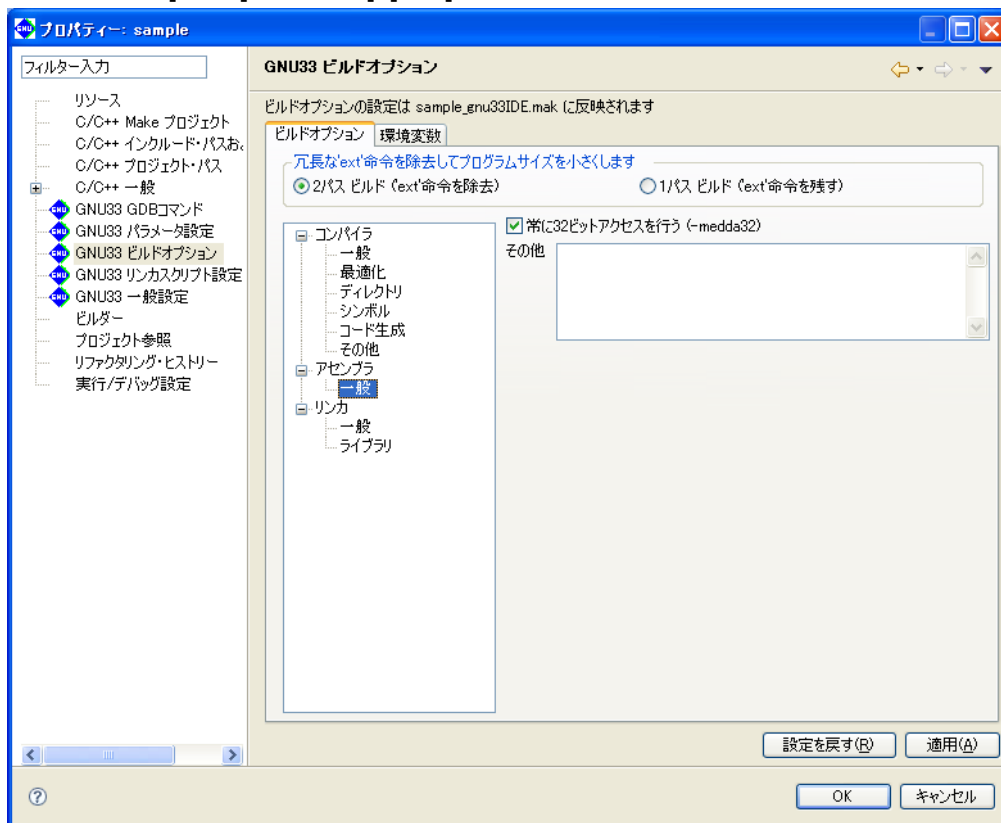
コンパイラ*のプログラム名を表示します。

[全オプション]

現在設定されているオプションを表示します。

注：IDEではアセンブラソースを、`-c -xassembler-with-cpp`オプションを指定したCコンパイラでアセンブルします。全オプションには、コンパイラの`-c`オプション、`-xassembler-with-cpp`は指定する必要はありません。

[ビルドオプション]タブ>[アセンブラ]>[一般]



このページではアセンブルに関するオプションを選択します。-c、-xassembler-with-cpp、-Wa,--gstabsオプションは以下の選択にかかわらず、常に付加されます。

[常に32ビットアクセスを行う(-medda32)]

このオプションを指定すると、デフォルトデータエリアを使用せずに、32ビットアクセス用のコードを出力します。

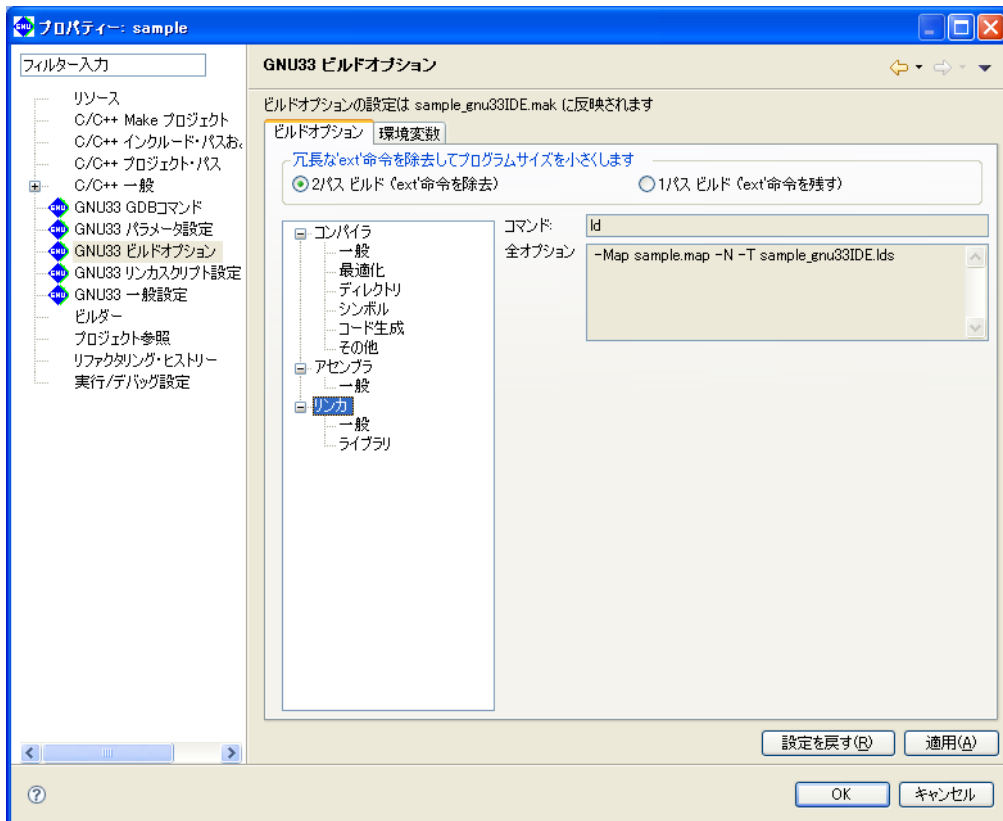
[その他]

アセンブラに渡すオプションを入力します。オプション間には1個以上のスペースを挿入してください。

CPUタイプに"S1C33401"を選択した場合は、-mc33advと-mc33401オプションがここで指定されます。同様に、CPUタイプに"S1C33PE"を選択した場合は、-mc33peオプションがここで指定されます。

入力したオプションは"-Wa,<オプション>, ..."としてアセンブラに渡されます。

[ビルドオプション]タブ>[リンカ]



リンカオプションの設定内容を表示します。

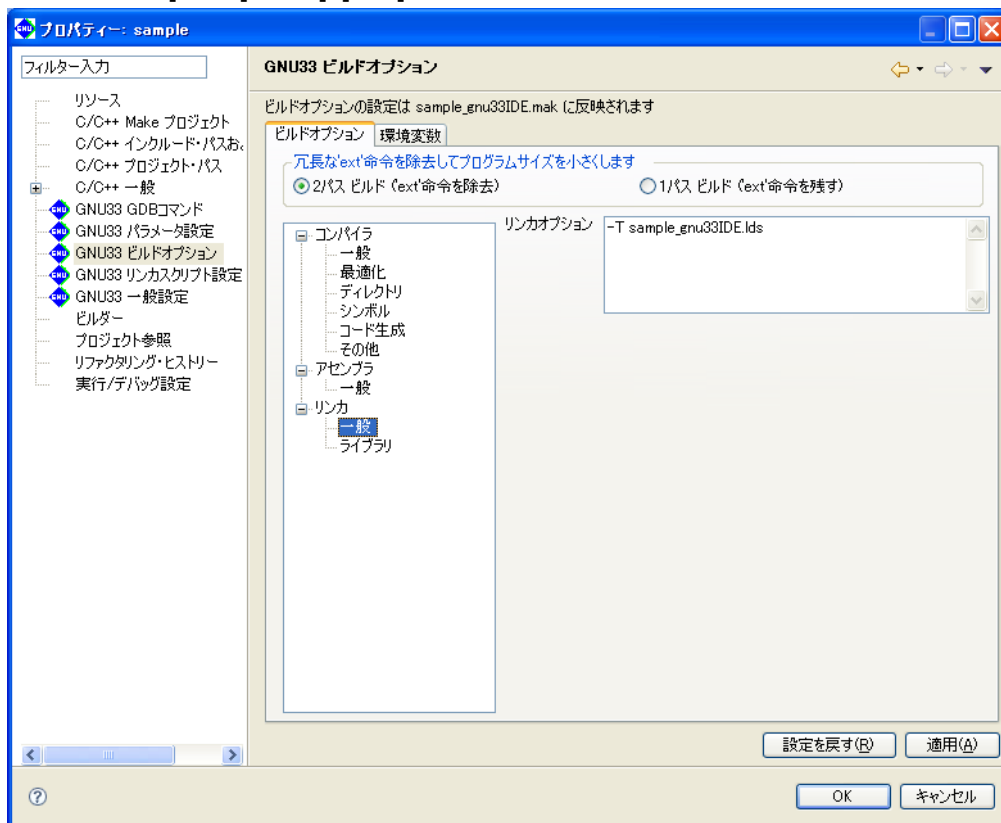
[コマンド:]

リンカのプログラム名を表示します。

[全オプション]

現在設定されているオプションを表示します。

[ビルドオプション]タブ>[リンカ]>[一般]

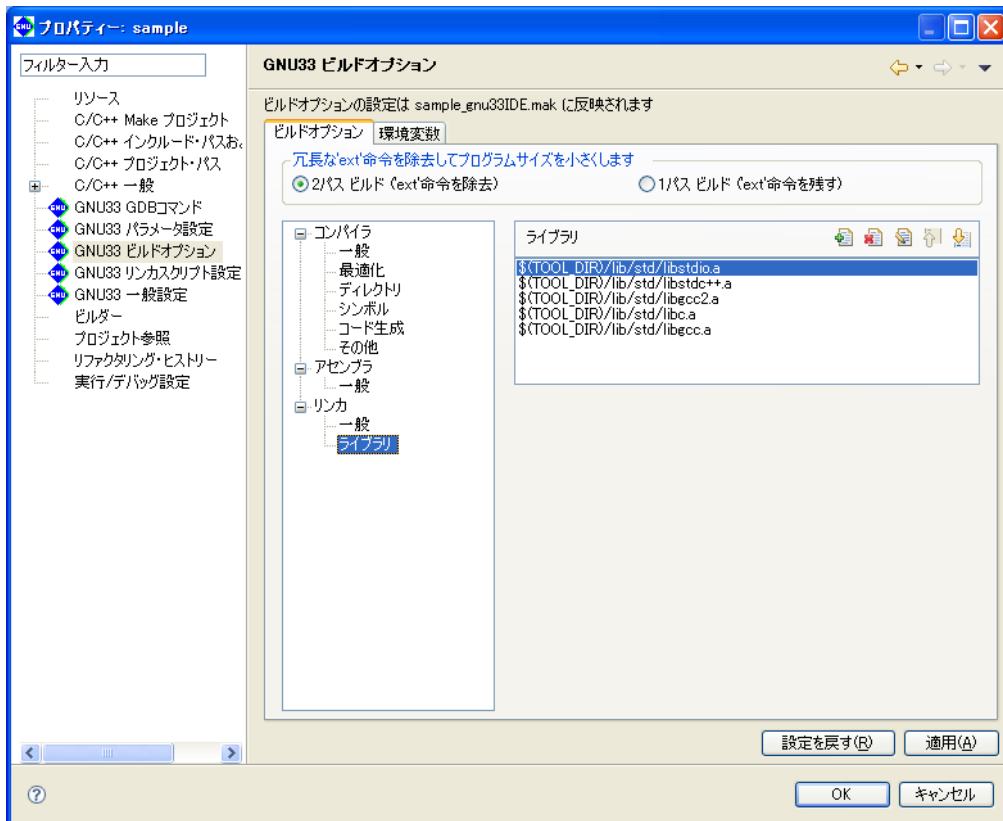


リンカの-Map、-Nオプションは常に指定されます。このページではリンカのその他のオプションを選択します。

[リンカオプション]

その他のリンカオプションを入力します。オプション間には1個以上のスペースを挿入してください。

[ビルドオプション]タブ>[リンカ]>[ライブラリ]



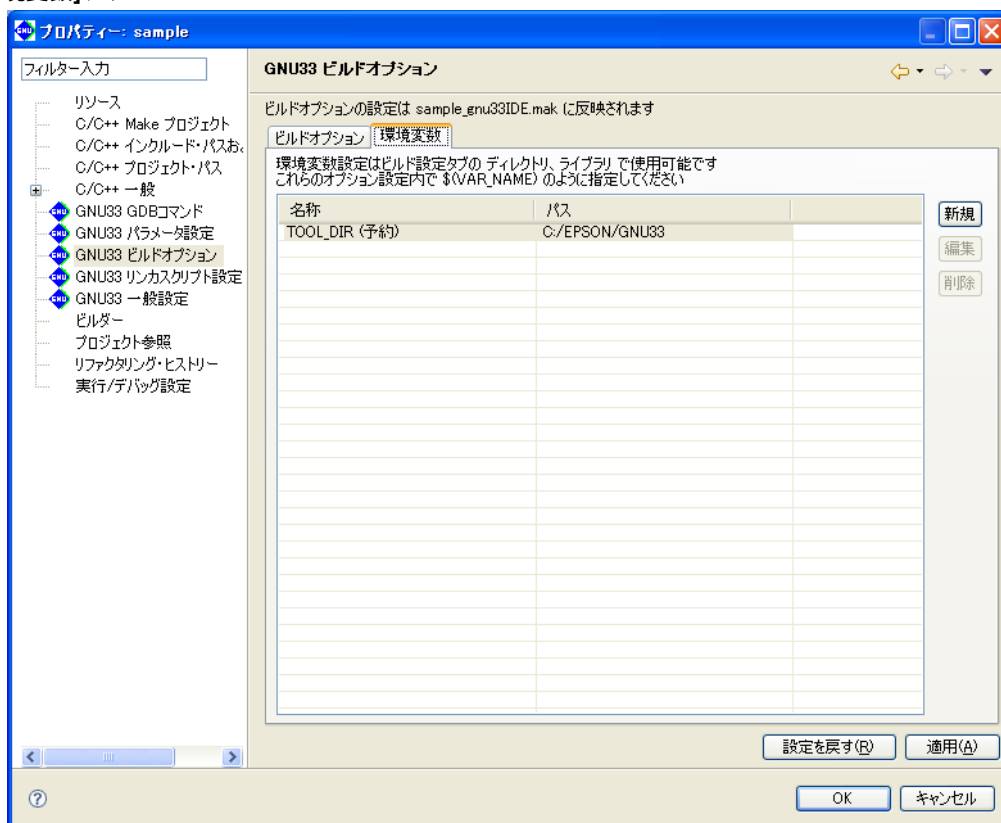
このページではリンクするライブラリを設定します。

[ライブラリ](デフォルト: libstdio.a, libstdc++.a, libgcc2.a, libc.a, libgcc.a)

リンクするライブラリを設定します。ボタンの機能は以下のとおりです。

-  [追加...] ライブラリを追加します。パスを入力あるいは[ファイル・システム...]ボタンで選択するダイアログが表示されます。
-  [削除] リスト内で選択したライブラリを削除します。
-  [編集...] リスト内で選択したライブラリを編集します。パスを書き換えるダイアログが表示されます。
-  [上へ移動] リスト内で選択したライブラリを一覧内の一つ上に移動します。ライブラリは一覧内の上から順にリンクされます。
-  [下へ移動] リスト内で選択したライブラリを一覧内の一つ下に移動します。

[環境変数]タブ



[ビルドオプション]タブの[コンパイラ]>[ディレクトリ]と[リンカ]>[ライブラリ]のパス指定に使用する環境変数を管理します。

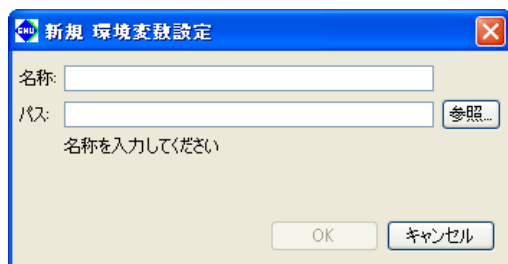
環境変数リスト

環境変数名と定義されているパスの一覧を表示します。TOOL_DIRはgnu33ツールがインストールされたディレクトリを定義した変数で変更や削除は行えません。

[新規]

環境変数を新たに定義します。

[新規 環境変数設定]ダイアログが表示されますので、[名称:]に変数名、[パス:]に定義するパスを入力して[OK]をクリックします。パスは[参照...]ボタンで表示されるダイアログによる選択も可能です。



[編集]

リスト内で選択した環境変数の変数名、パスを変更します。[新規]ボタンと同様のダイアログが表示されます。

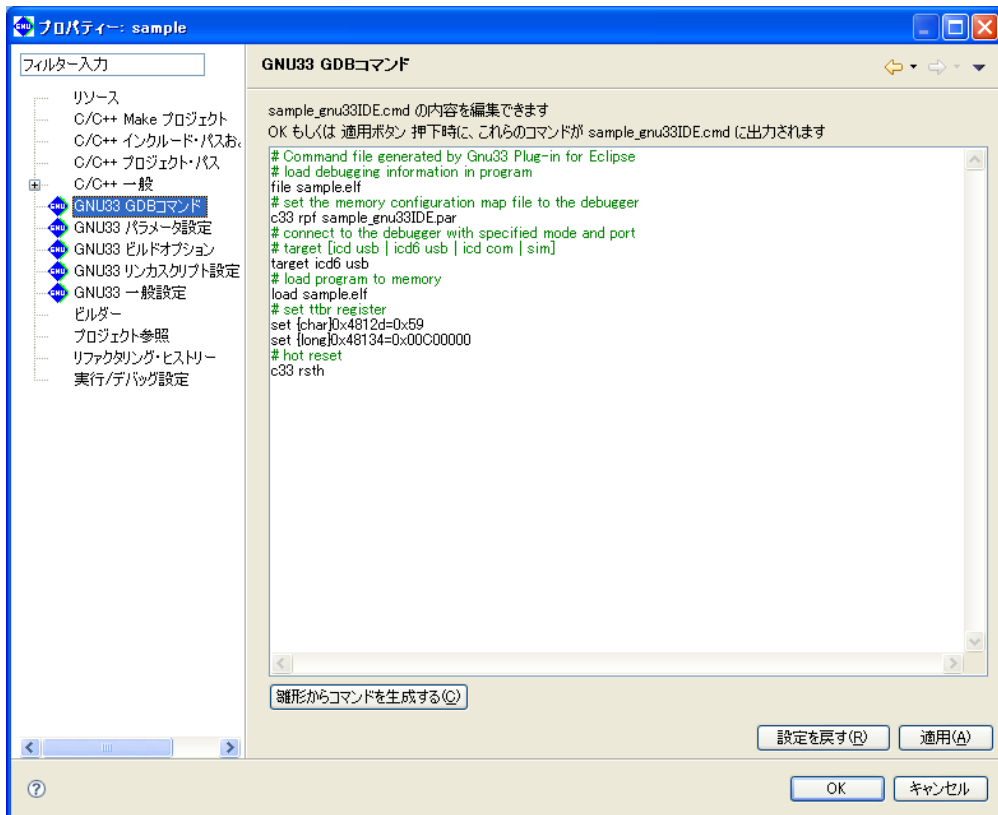
[除去]

リスト内で選択した環境変数を削除します。

注: • [新規 環境変数設定]ダイアログにパスとして入力可能な文字は、英数字、'_'(アンダーバー)、':'(コロン)、'\'(スラッシュ)、'\'または'¥'(バックスラッシュまたは円マーク)のみです。日本語のパスは指定できません。

- 定義した環境変数は、\$ (環境変数) の形式で使用してください。これ以外の形式では置き換えが行われずにビルド時にエラーとなります。

●GNU33 GDBコマンド

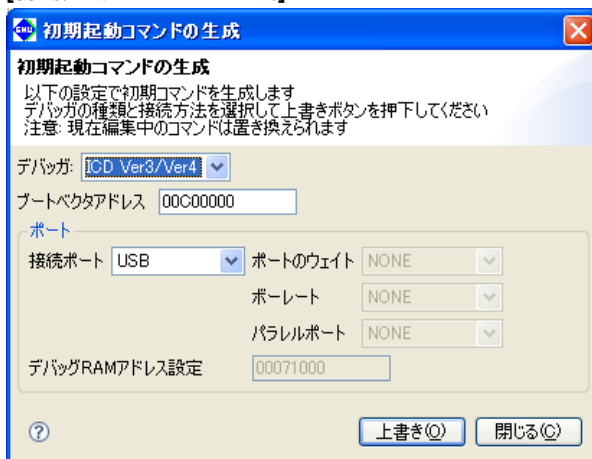


デバッガ起動用コマンドファイルを編集します。このページに表示されている内容がそのままコマンドファイルに書き出されます。

[雛形からコマンドを生成する]

コネクトモードなどを設定する[初期起動コマンドの生成]ダイアログ(下記参照)を表示します。
[初期起動コマンドの生成]ダイアログによる設定内容は、このページ上のコマンドファイルの内容に反映されます。

[初期起動コマンドの生成]



デバッガのモードやICDを接続するポートの設定を行います。

[デバッグ:]

接続するデバッグ(コネクトモード)を選択します。5種類の選択肢がありますが、実際に選択できるのは、[GNU33 一般設定]ページで選択されているターゲットCPUによって変わります。

ICD Ver6 ICD Ver. 6(S5U1C33001H1400)を使用してデバッグを行う場合

ICD Ver3/Ver4 ICD Ver. 3(S5U1C33001H1100)またはVer. 4(S5U1C33001H1200)を使用してデバッグを行う場合

ICD Ver2 ICD Ver. 2(S5U1C33000H)を使用してデバッグを行う場合

MON デバッグモニタ(S5U1C330M1D1+S5U1C330M2S)を使用してデバッグを行う場合

Simulator パーソナルコンピュータのみでデバッグを行う場合

表5.10.1.1 ターゲットCPUとコネクトモード

モード	C33 STDコア	C33 PEコア	S1C33401
ICD Ver6	○	○	○
ICD Ver3/Ver4	○	○	○
ICD Ver2	○	×	×
MON	○	×	×
Simulator	○	○	○

[ブートベクタアドレス]

ブートベクタアドレスをテキストボックスに16進数(0xは不要)で入力します。C33 STDまたはC33 PEコア搭載機種を使用する場合、アドレスは400KB単位で指定する必要があります。S1C33401では1KB単位となります。

IDEが生成するコマンドファイルには、ブートベクタアドレスをこの値に設定するコマンドが記述されます。

ここにデフォルトで表示されている値は、プロジェクト作成時に指定したアドレスです。

[ポート]

[デバッグ:]にICD Ver2またはMONを選択した場合に、使用するCOMポートの設定を行います。ICD Ver3/Ver4またはSimulatorを選択した場合、ポートはそれぞれUSB、NONEに固定されますので設定の必要はありません。

[接続ポート]

S5U1C33000H(ICD Ver.2) またはS5U1C330M1D1(MON)を接続するCOMポート(COM1～COM8)を選択します。

[ポートのウェイト]

PC側のCOMポートを開く際に入れるウェイトを指定します。0～20秒までのウェイトを設定可能です。ノートPC等の一部PCでCOMポートを開いた後にウェイトを挿入しないと正常に通信を開始できない場合があります。そのような場合にウェイトを設定してください。

[ボーレート]

COMポートのボーレートを115200bpsまたは38400bpsから選択します。[デバッグ:]にMONを指定した場合は、115200bpsを指定する必要があります。

[パラレルポート]

ターゲットにプログラムを転送するときに使用するパラレルポートの番号を指定します。

NONE、LPT1、LPT2を指定可能で、それぞれパラレル使用なし、パラレルポート1、2に対応します。[デバッグ:]でICD Ver2を指定した場合のみ選択可能です。

[デバッグRAMアドレス設定]

デバッグがデバッグ時に使用するターゲット側メモリのアドレスを設定します。[GNU33 一般設定]ページのターゲットCPUにS1C33PE、[デバッグ:]にICD Ver3/Ver4またはICD Ver6を選択した場合のみ設定可能です。この組み合わせ以外では[デバッグRAMアドレス設定]を変更することはできません。ここで指定した領域はプログラム内で使用しないようにしてください。

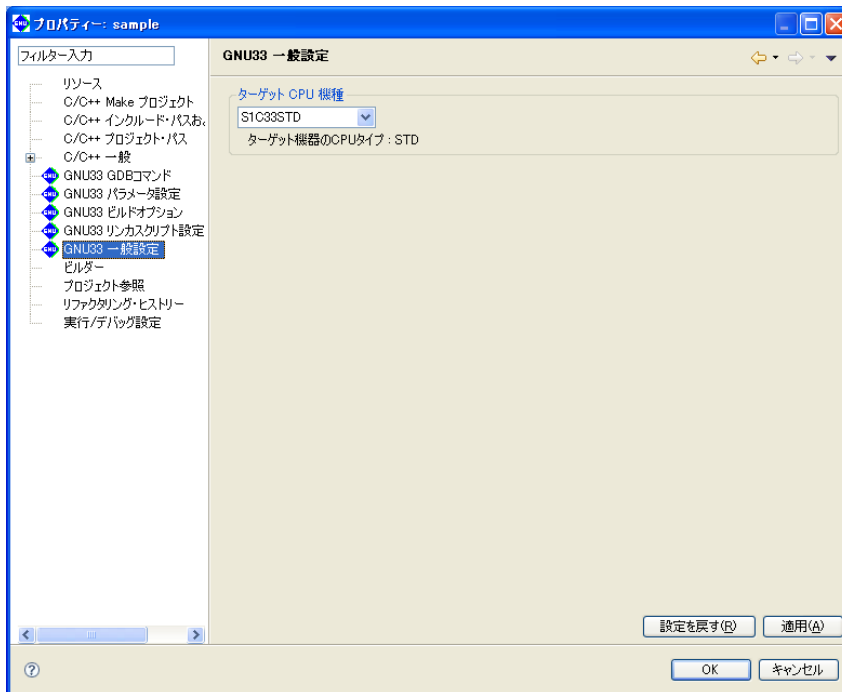
[上書き]

上記の設定内容を[GNU33 GDBコマンド]ページのコマンドファイルに反映させて、ダイアログを閉じます。このボタンをクリックすると上書きを確認するダイアログが表示され、実行のキャンセルも可能です。

[閉じる]

本ダイアログを閉じます。変更内容は[GNU33 GDBコマンド]ページのコマンドファイルに反映されません。

●GNU33 一般設定



ターゲットプロセッサを選択します。

[ターゲット CPU 機種]

以下の3種類からターゲットプロセッサを選択します。

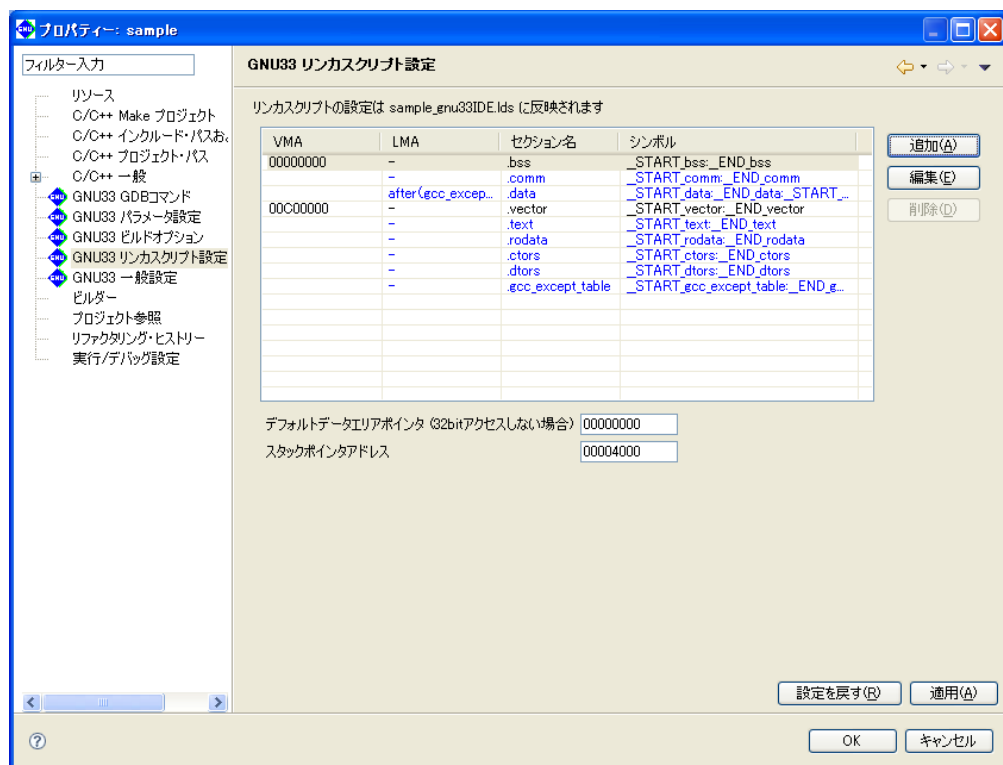
S1C33STD: C33 STDコアを内蔵したデバイスをターゲット機種とする場合

S1C33PE: C33 PEコアを内蔵したデバイスをターゲット機種とする場合

S1C33401: S1C33401をターゲット機種とする場合

* 上記の設定を変更後に[OK]または[適用]ボタンをクリックした場合は、"clean"ビルド(5.7.9節参照)を実行するかどうかを確認するダイアログが表示され、以前の設定で生成されているファイルの削除(およびリビルド)が行えます。

●GNU33 リンカスクリプト設定



リンカスクリプトを編集します。

セクションリスト

実行ファイル(.elf)内のセクション構成と配置を示しています。

青色はデフォルトで定義済みの標準セクションを表し、黒色はそれ以外の新たに定義したユーザセクションを表します。ユーザセクションは、セクション名、セクションの基本属性、配置アドレス、配置するオブジェクトを任意に編集可能です。標準セクションは配置アドレスのみ指定可能で、ユーザセクションに配置された以外のオブジェクトが自動的に配置されるようになっています。

[VMA]

実行時にセクションを置く位置(先頭アドレス)を表示します。VMAにアドレスの記載のないセクションはその一つ上のセクションに続いて配置することを表します。

[LMA]

実データを置いておくROM内の位置(先頭アドレス)を表示します。"- "はVMAと同じ(実データが置かれた位置で実行またはアクセスされる)ことを表します。"after(<セクション名>)"は()内のセクションに続いて実データが配置されることを示します。

[セクション名]

セクション名を表示します。

[シンボル]

セクションが配置される領域の先頭および終了アドレスを示すラベルが表示されます。LMAが指定されない場合は2つのラベルが、LMAが指定されている場合はVMAの先頭/終了、LMAの先頭/終了の順に4つのラベルが表示されます。これらのラベルは、ROMからRAMへのセクションのコピーの際など、ソースファイル内でのアドレス指定に使用できます。これらのラベル名はセクション名から自動的に生成されます。

[追加...]

セクション情報を追加します。

[編集]

一覧で選択したセクション情報を編集します。

[削除]

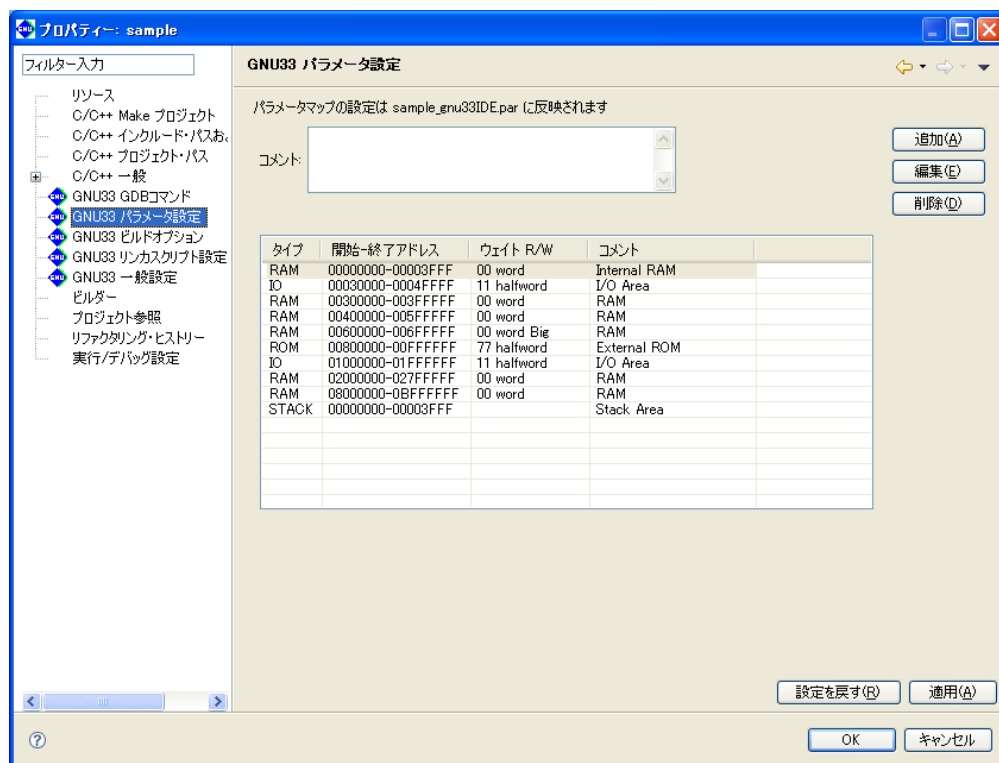
一覧で選択したセクション情報を削除します。

[デフォルトデータエリアポインタ]

リンカスクリプトに記述するデフォルトデータエリアポインタ(アドレス)を設定します。

リンカスクリプトの編集方法については、"5.7.7 リンカスクリプトの編集"を参照してください。

●GNU33 パラメータ設定



デバッガで使用するパラメータファイルを編集します。

[コメント:]

任意のコメントを入力することができます。最大255文字まで入力可能です。ここに入力したコメントはパラメータファイル内に記述されます。

エリアリスト

各行に1つのエリア情報が表示されます。一覧は、アドレス順に表示されます。スタックエリア情報のみ最後にまとめられます。

[タイプ]

エリアの種類(ROM、RAM、IO、STACK)が表示されます。

[開始-終了アドレス]

エリアの開始-終了アドレスが16進数で表示されます。

[ウェイト R/W]

先頭の2桁の数値は、前が読み出しサイクルのウェイト数、後が書き込みサイクルのウェイト数を表します。byte、halfword、wordはエリアのアクセスサイズを示します。その後が空白の場合は、エリアがリトルエンディアンでアクセスされることを示しています。ビッグエンディアンに設定したエリアの場合、Bigが表示されます。

[コメント]

各エリアの情報に入力されているコメントを表示します。コメントの先頭を示す"#"を入力する必要はありません。

[追加]

エリア情報を追加します。

[編集]

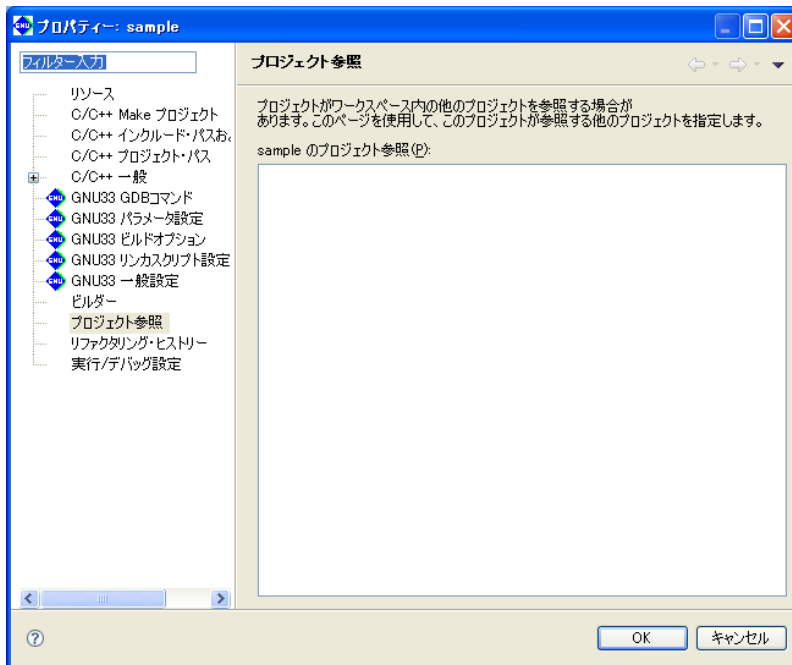
一覧で選択したエリア情報を編集します。

[削除]

一覧で選択したエリア情報を削除します。

パラメータファイルの編集方法については、"5.8.1 パラメータファイルの生成"を参照してください。

●プロジェクト参照

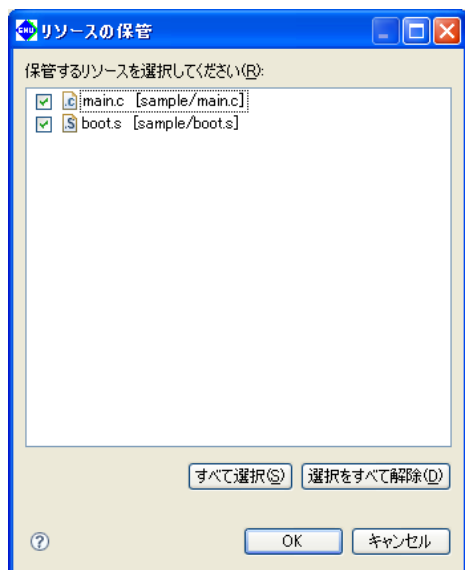


参照するプロジェクトを選択します。

[sample のプロジェクト参照:]

現在のプロジェクトが参照する他のプロジェクトを選択します。

5.10.2 リソースの保管



このダイアログは、エディタで編集中の文書が保存されていない場合に、複数の文書を閉じるような操作を行った場合に表示されます。

ファイルのチェックボックス

保存するファイルのチェックボックスを選択します。

[すべて選択]

すべてのファイルのチェックボックスを選択します。

[選択をすべて解除]

すべてのファイルのチェックを外します。

[OK]

選択されたファイルを保存後、エディタの文書を閉じます。

[キャンセル]

このダイアログを表示させた操作をキャンセルします。文書の保存やクローズ処理は行われません。

5.10.3 新規 C++ クラス

このダイアログは、[ファイル]メニューから[新規]>[その他...]>[C++]>[クラス]を選択(ツールバーの[新規 C++ クラス]ボタンをクリック等)して新規クラスを作成する際に表示されます。

[ソース・フォルダー:]

ソースおよびヘッダファイルを作成するフォルダ名を入力するか、[参照...]ボタンで選択します。

[名前空間:]

特定の名前空間に属するクラスを定義する場合にチェックボックスをONにし、テキストボックスにその名称を入力するか、[参照...]ボタンで選択します。

[クラス名:]

作成するクラスの名称を入力します。

[ベースクラス:]

現在のプロジェクト内ですでに作成されているクラスの派生クラスを作成する場合に、継承元となる基本クラスの名称をリストしておきます。現在のプロジェクトに定義されているクラスを[追加...]ボタンで表示される[ベース・クラスの選択]ダイアログから、すでに定義されているクラスを選択できます。

例: AClassを基本クラスに指定し、BClassという派生クラスを作成した場合のクラス宣言(アクセス指定子にpublicを選択)

```
class BClass : public AClass{
public:

    BClass();
    virtual ~BClass();
};
```

リスト内の不要なクラスは[除去]ボタンで削除できます。また、リストの並びを[上へ]および[下へ]ボタンで編集できます。

注: 基本クラスや名前空間を選択するには、プロジェクトプロパティの[C/C++一般]>[インデクサー]で[高速 C/C++ インデクサー]を選択しておく必要があります。

[メソッド・スタブ:]

コンストラクタ、デストラクタを定義する場合にチェックボックスをONにします。

[アクセス]

コンストラクタ、デストラクタ内に記述するアクセス指定子をpublic、protected、privateから選択します。

[仮想]

仮想デストラクタ宣言を行う場合は"はい"を、仮想でないデストラクタ宣言を行う場合は"いいえ"を選択します。

[インライン化]

"はい"を選択すると、コンストラクタ定義とデストラクタ定義をヘッダファイルに含め、ソースファイルにはヘッダファイルのインクルード文のみ記述されます。"いいえ"を選択するとはコンストラクタ定義とデストラクタ定義をソースファイルに含めます。

[デフォルトを使用]

ヘッダファイルとソースファイルを新規作成する場合はチェックボックスをONにします。既存のファイルに追加する場合はチェックを外し、ファイル名を入力するか、[参照...]ボタンで選択します。

[終了]

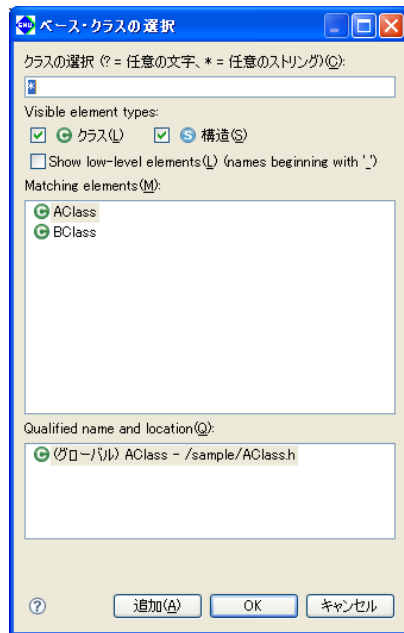
新規クラスを生成します。

[キャンセル]

新規クラスの作成を中止します。

5.10.4 ベース・クラスの選択

注: [ベース・クラスの選択]ダイアログを表示して基本クラスを選択するには、プロジェクトプロパティの[C/C++ 一般]>[インデクサー]で[高速 C/C++ インデクサー]を選択しておく必要があります。



このダイアログは、新規クラスを作成する際に[新規 C++ クラス]ダイアログ上の[ベース・クラス:](基本クラス)を選択する[追加...]ボタンをクリックすると表示されます。

[クラスの選択 (?=任意の文字、*=任意のストリング)]

[Matching elements:]に一覧表示させるクラスを一部のものに制限するための文字列を入力します。"? "は1文字のワイルドカード、"* "は文字列のワイルドカードとして使用します。

例: ?Class 先頭の1文字は任意(AClass等)

*Class Classの前は任意の文字列(GetTimeClass等)

[Visible element types:]

[Matching elements:]に一覧表示させるタイプを選択します。

[クラス]

クラスを表示

[構造]

構造体を表示

[Show low-level elements]

下位レベルタイプを表示

[Matching elements:]

現在のプロジェクト内に定義されている、上記条件を満たすタイプの一覧を表示します。ここで目的の基本クラスを選択します。

[Qualified name and location:]

修飾子を表示します。

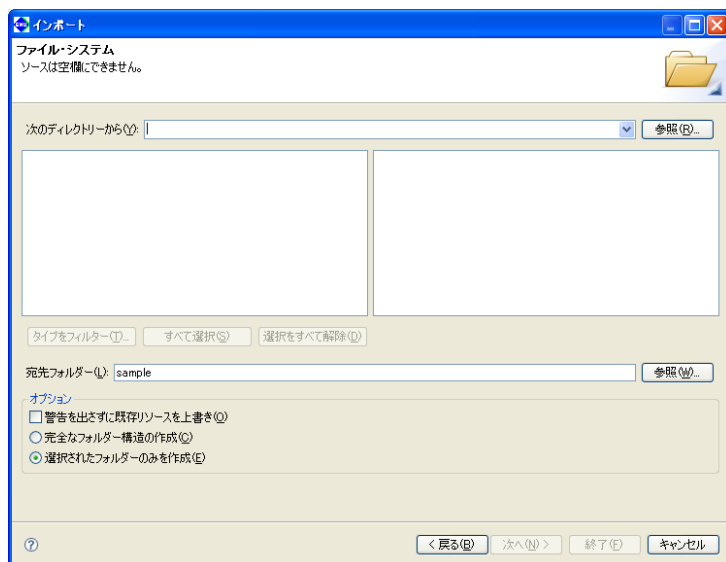
[OK]

[Matching elements:]で選択した基本クラスを[新規 C++ クラス]ダイアログ上の[ベース・クラス:]に入力します。

[キャンセル]

選択を中止します。

5.10.5 インポート>ファイル・システム



このダイアログは、ファイル/フォルダをインポートするために[インポート]ダイアログで[一般]>[ファイル・システム]を選択し、[次へ>]ボタンをクリックすると表示されます。

[次のディレクトリーから]

インポートするファイル/フォルダの親フォルダへのパスを入力するか、[参照...]ボタンで選択します。

フォルダリスト(左)

[次のディレクトリーから]で選択したフォルダをルートとして、子フォルダの一覧が表示されます。

フォルダをインポートする場合は、このリストから選択します。

ファイルをインポートする場合は、その親フォルダを選択します。

ファイルリスト(右)

フォルダリストで選択したフォルダ内にあるファイルの一覧が表示されます。ファイルをインポートする際の選択に使用します。

[タイプをフィルター ...]

ファイルリストで選択した中から、さらにファイル形式(拡張子)でインポートするファイルの絞り込みを行います。

このボタンで表示されるダイアログで拡張子を選択すると、その拡張子以外のファイルが非選択状態になります。

[すべて選択]

リストボックスに表示されたファイルをすべて選択します。

[選択をすべて解除]

リストボックスに表示されたファイルをすべて非選択状態にします。

[宛先フォルダー:]

[C/C++ プロジェクト]または[ナビゲーター]ビューで選択したプロジェクト/フォルダが表示されます。他のプロジェクト/フォルダにインポートする場合は、[参照...]ボタンで選択します。

[警告を出さずに既存ソースを上書き]

このチェックボックスを選択すると、インポート先のフォルダに同名のファイル/フォルダがあっても、確認メッセージなしに上書きされます。非選択(デフォルト)の場合、上書きを確認するメッセージが表示されます。

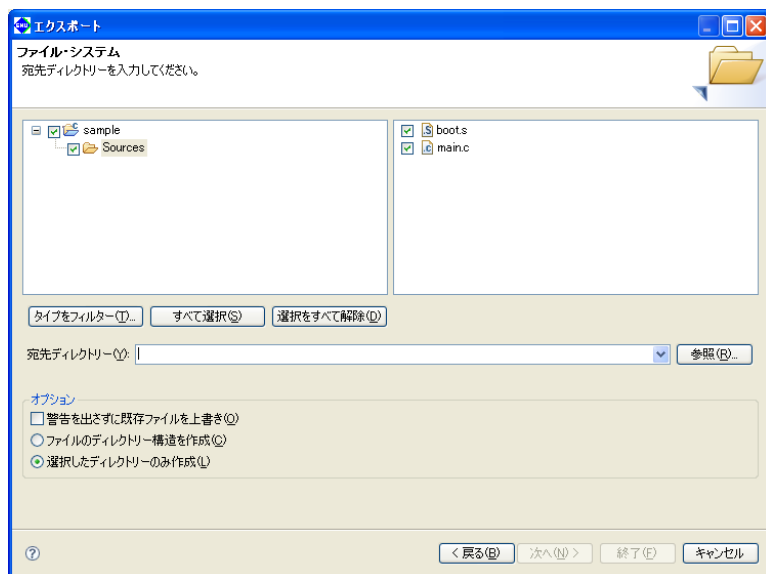
[完全なフォルダー構造の作成]

このボタンを選択すると、フォルダインポート時にファイルシステムのディレクトリ構造(ルートから選択したフォルダまでのツリー構造)も含めてインポートされます。

[選択されたフォルダーのみを作成]

このボタンを選択すると、選択したフォルダのみインポートされます。ディレクトリ構造はインポートされません。

5.10.6 エクスポート>ファイル・システム



このダイアログは、ファイル/フォルダを書き出すために[エクスポート]ダイアログで[一般]>[ファイル・システム]を選択し、[次へ>]ボタンをクリックすると表示されます。

フォルダリスト(左)

プロジェクトフォルダをルートとして、子フォルダの一覧が表示されます。
 フォルダを書き出す場合は、このリストから選択します。
 ファイルを書き出す場合は、その親フォルダを選択します。

ファイルリスト(右)

フォルダリストで選択したフォルダ内にあるファイルの一覧が表示されます。ファイルを書き出す際の選択に使用します。

[タイプをフィルター ...]

ファイルリストで選択した中から、さらにファイル形式(拡張子)で書き出すファイルの絞り込みを行います。
 このボタンで表示されるダイアログで拡張子を選択すると、その拡張子以外のファイルが非選択状態になります。

[すべて選択]

リストに表示されたフォルダとファイルをすべて選択します。

[選択をすべて解除]

リストに表示されたフォルダとファイルをすべて非選択状態にします。

[警告を出さずに既存ソースを上書き]

このチェックボックスを選択すると、書き出し先のフォルダに同名のファイル/フォルダがあっても、確認メッセージなしに上書きされます。非選択(デフォルト)の場合、上書きを確認するメッセージが表示されます。

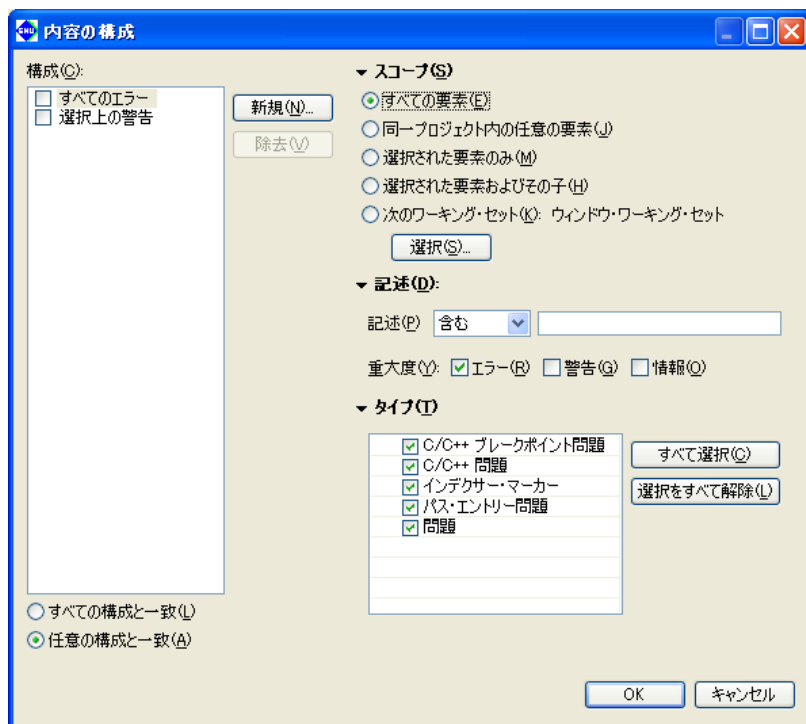
[ファイルのディレクトリー構造を作成]

このボタンを選択すると、ファイルシステムにディレクトリ構造(プロジェクトフォルダからのツリー構造)も含めて書き出されます。

[選択したディレクトリーのみ作成]

このボタンを選択すると、選択したフォルダのみ書き出されます。プロジェクトのディレクトリ構造は書き出されません。

5.10.7 フィルター



このダイアログは[問題]、[ブックマーク]、[タスク]ビューのビューメニュー (▽)から[内容の構成]を選択すると表示されます。ビューの一覧表示が大きくなった場合などに、ここで条件を設定し、表示させるリソースの対象や数を制限します。

[構成:]

設定されているフィルタの一覧が表示されます。それぞれのチェックボックスにより、そのフィルタを適用するかどうかを選択します。

フィルタの設定内容を変更する場合は、ここでフィルタ名を選択し、条件を選択します。

複数のフィルタのチェックボックスを選択すると、すべての表示条件がORされます。

[新規]

新しいフィルタを作成します。表示される[新規フィルタの追加]ダイアログにフィルタ名を入力すると、[構成:]の一覧に加えられます。一覧でそのフィルタを選択した状態で、条件を選択してください。

[除去]

[構成:]の一覧で選択されているフィルタを削除します。

[すべての要素]

開かれているプロジェクト内のすべてリソースを対象にします。

[同一プロジェクト内の任意の要素]

現在エディタでアクティブになっているリソース、または[C/C++ プロジェクト]/[ナビゲーター]ビューで選択したリソースと同じプロジェクト内のみを表示の対象とします。

[選択された要素のみ]

現在エディタでアクティブになっているリソース、または[C/C++ プロジェクト]/[ナビゲーター]ビューで選択したリソースのみを表示の対象とします。

[選択された要素およびその子]

[C/C++ プロジェクト]/[ナビゲーター]ビューで選択したフォルダ内のリソースのみを表示の対象とします。

[次のワーキング・セット:]

指定のワーキングセット内のみを表示の対象とします。

[選択...]

[次のワーキング・セット:]を選択した場合に、表示対象とするワーキングセットを選択します。

[完了] ([タスク]ビューのみ)

[完了] 完了したタスクを表示します。

[未完了] 未完了のタスクを表示します。

[優先度] ([タスク]ビューのみ)

タスクの優先順位で表示を制限する場合に選択し、さらに表示させる優先順位(高、標準、低)をチェックボックスで選択します。

[記述]

含む[記述]項目に、テキストボックスに入力した文字列を含むもののみを表示します。

含まない[記述]項目に、テキストボックスに入力した文字列を含まないもののみを表示します。

この条件を無効にするには、テキストボックス内を空白にしてください。

[重大度:] ([問題]ビューのみ)

エラーの程度で表示を制限する場合に選択し、さらに表示させる内容(エラー、警告、情報)をチェックボックスで選択します。

[タイプ] ([問題]ビュー、[タスク]ビューのみ)

ビューに表示させる一覧の種類を選択します。

[すべて選択] [タイプ]の表示項目をすべて選択します。

[選択をすべて解除] [タイプ]の表示項目をすべて非選択にします。

[OK]

設定した条件でのフィルタリングを開始します。

[キャンセル]

フィルタ設定を中止します。

5.11 IDEがプロジェクト内に作成するファイル

表5.11.1 IDE作成ファイル一覧

ファイル名	ファイル内容	編集可	要管理 ファイル
.project	IDEプロジェクトファイル	×	○
.cproject	IDEプロジェクトファイル(CDT)	×	○
.cdtproject	IDEプロジェクトファイル(CDT) GNU33v3.2.1以前に作成されたプロジェクト	×	○
.gnu33project	IDEプロジェクトファイル(GNU33)	×	○
GDB33 Debugger for <プロジェクト名>.launch	GDB起動設定用ファイル	×	○
<プロジェクト名>_gnu33IDE.cmd	GDBコマンドファイル	○ *1	○
¥.settings	プロジェクト設定格納ディレクトリ	×	○
¥.externalToolビルダー	プロジェクト設定格納ディレクトリ(ビルダ)	×	○
<プロジェクト名>_gnu33IDE.mak	makeファイル	×	×
<プロジェクト名>_gnu33IDE.lids	リンカスクリプトファイル	×	×
<プロジェクト名>_gnu33IDE.par	パラメータファイル	×	×
<ソースファイル名>.elf	実行ファイル	×	×
<ソースファイル名>.map	マップファイル	×	×
<ソースファイル名>.dump	2パスメイク用シンボルファイル	×	×
<ソースファイル名>.d	makeファイル用依存関係ファイル	×	×
<ソースファイル名>.o	オブジェクトファイル	×	×
<ソースファイル名>.ext0	2パスメイク用アセンブラソースファイル	×	×

*1: プロジェクトの[プロパティ]ダイアログを閉じているときのみエディタ等で編集可

要管理ファイルとは、ソース管理アプリケーションなどで管理する必要のあるファイル等を指します。

6 C/C++コンパイラ

この章ではC/C++コンパイラ**xgcc**の使用方法和アセンブリソースとのインタフェースに関する内容を説明します。C/C++コンパイラの標準機能やC/C++ソースの文法については、ANSI C/C++を解説している一般の書籍をご覧ください。

6.1 機能

C/C++コンパイラ**xgcc**はC/C++ソースファイルをコンパイルし、C33コア命令セットのニーモニック、拡張命令、アセンブラの擬似命令を含むアセンブリソースファイルを生成します。**xgcc**はANSI C/C++準拠のGNU C/C++コンパイラです。特殊な文法はサポートしていませんので、他機種用のプログラムの移植なども容易に行えます。

また、強力なオプティマイズ能力を持ち、非常にコンパクトなコードを生成しますので、組み込み用アプリケーションの開発に最適です。

C/C++コンパイラは**xgcc.exe**、**cpp.exe**、**cc1.exe**、**cc1plus.exe**の4つのファイルで構成されています。

xgccはFree Software Foundation, Inc.のC/C++コンパイラをベースとしています。ライセンスに関してはテキストファイル"Copying.GNU"に記述されていますので、ご使用前にご覧ください。

また、このC/C++コンパイラは日本ノーベル株式会社のコンパイラ評価サービスをクリアしています。

この評価で検出されたバグの中で、C/C++コンパイラの制限事項となっているもの"[¥gnu33¥doc¥release_history_j.pdf](#)"及び、"6.11 既知の問題"に記載されていますので、参照してください。

6.2 入出力ファイル

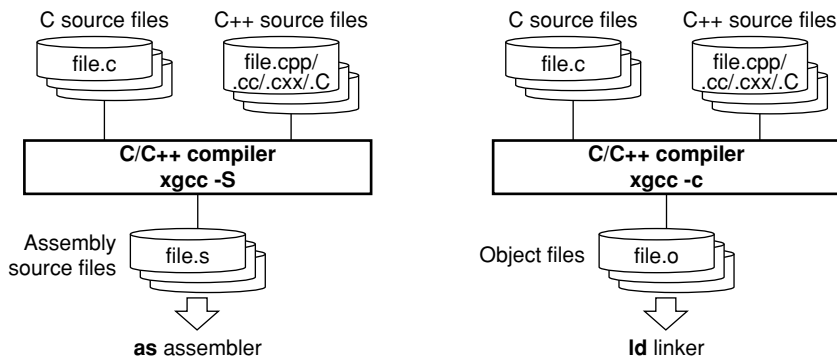


図6.2.1 フローチャート

6.2.1 入力ファイル

●Cソースファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.c

内容: Cソースプログラムを記述したファイルです。

●C++ソースファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.cpp, <ファイル名>.cc, <ファイル名>.cxx, <ファイル名>.C

内容: C++ソースプログラムを記述したファイルです。

6.2.2 出力ファイル

●アセンブリソースファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.s

内容: アセンブラasに入力するアセンブリソースファイルです。-sオプションを指定した場合に生成されます。

●オブジェクトファイル

ファイル形式: バイナリファイル

ファイル名: <ファイル名>.o

内容: リンカldに入力するリロケータブルオブジェクトファイルです。-cオプションを指定した場合に生成されます。

注: コマンドオプションの指定により、C/C++コンパイラxgccはelf形式の実行可能なオブジェクトファイルやプリプロセッサのみを通したソースファイルも出力できます。

6.3 起動方法

6.3.1 起動フォーマット

C/C++コンパイラxgccは次のコマンドにより起動します。

xgcc <オプション> <ファイル名>

<オプション> 6.3.2項参照

<ファイル名> C/C++ソースファイルを拡張子(.c, .cpp, .cc, .cxx, .C)も含めて指定します。

6.3.2 コマンドラインオプション

本パッケージのコンパイラは、以下のコマンドラインオプションを正式にサポートします。ここに記載されているもの以外は動作保証対象外です。お客さまの責任で使用してください。

-c

機能: リロケータブルオブジェクトファイルの出力

説明: リロケータブルオブジェクトファイル(<入力ファイル名>.o)を出力するためのオプションです。
このオプションを指定すると、C/C++コンパイラxgccはアセンブルが終了した時点で処理を中断し、リンクは行いません。このオプションを使用する場合、同時に-sまたは-Eを指定しないでください。

デフォルト: C/C++コンパイラxgccはelf形式の実行可能なオブジェクトファイルを生成します。

-S

機能: アセンブリコードの出力

説明: アセンブリソースファイル(<入力ファイル名>.s)を出力するためのオプションです。
このオプションを指定すると、C/C++コンパイラxgccはコンパイルが終了した時点で処理を中断し、アセンブルは行いません。IDEが生成する基本的なmakeファイルは、C/C++コンパイラxgccの起動コマンドにこのオプションを使用しています。このオプションを使用する場合、同時に-cまたは-Eを指定しないでください。

デフォルト: C/C++コンパイラxgccはelf形式の実行可能なオブジェクトファイルを生成します。

-E

機能: Cプリプロセッサのみの実行

説明: このオプションを指定すると、C/C++コンパイラxgccはプリプロセッサの実行が終了した時点で処理を中断し、コンパイルやアセンブルは行いません。プリプロセッシング結果は標準出力に出力されます。このオプションを使用する場合、同時に-sまたは-cを指定しないでください。

デフォルト: C/C++コンパイラxgccはelf形式の実行可能なオブジェクトファイルを生成します。

-B<ディレクトリ>

機能: コンパイラの検索パス指定

説明: C/C++コンパイラ**xgcc**がサブプログラム(**cpp**や**cc1/cc1plus**など)やコンパイラ自体のデータファイルなどを探すための検索パスに<ディレクトリ>を追加します。
<ディレクトリ>は-Bに続けて入力してください。複数のディレクトリを指定することができます。その場合は、-B<ディレクトリ>を必要数入力してください。検索は、コマンドライン上の出現順に行われます。ファイル検索の優先順位は、カレントディレクトリ、-Bオプション、PATHの順です。

デフォルト: C/C++コンパイラ**xgcc**はカレントディレクトリおよびPATH指定ディレクトリ内のサブプログラムを検索します。

-I<ディレクトリ>

機能: インクルードファイルディレクトリを指定

説明: C/C++ソース中にインクルードしたファイルが存在するディレクトリを指定します。
<ディレクトリ>は-Iに続けて入力してください。複数のディレクトリを指定することができます。その場合は、-I<ディレクトリ>を必要数入力してください。インクルードファイルの検索は、コマンドライン上の出現順に行われます。
環境変数C_INCLUDE_PATHにディレクトリが登録されていれば-Iオプションは不要です。ファイル検索の優先順位は、カレントディレクトリ、-Iオプション、C_INCLUDE_PATHの順です。

デフォルト: C/C++コンパイラ**xgcc**はカレントディレクトリおよびC_INCLUDE_PATHディレクトリ内のインクルードファイルを検索します。

-fno-builtin

機能: ビルトイン関数を無効化

説明: 以下の関数がビルトイン関数としてコンパイルされず、必ずcallされます。このオプションがない場合、コンパイラは以下の関数を状況に応じてインライン展開、もしくは他の関数への置き換えを行うことにより、コードの効率化を行う場合があります。
abort, abs, alloca, cos, exp, fabs, fprintf, fputs, labs, log, memcmp, memcpy, memset, printf, putchar, puts, scanf, sin, sprintf, sqrt, sscanf, strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, vprintf, vsprintf

デフォルト: ビルトイン関数が有効になります。

-D<マクロ名> [=<置き換え文字>]

機能: マクロ名の定義

説明: #defineと同じ働きをします。=<置き換え文字>が指定されている場合はマクロにその値を定義します。指定されていない場合、マクロの値は1に設定されます。
<マクロ名> [=<置き換え文字>]は-Dに続けて入力してください。複数のマクロ名を指定することができます。その場合は、-D<マクロ名> [=<置き換え文字>]を必要数入力してください。

※マクロ名の自動生成について

コンパイル時、以下のマクロ名が自動的に定義されます。これらのマクロ名はどのソースファイルからでも利用可能です。

マクロ名	生成条件
__c33	コアの種類にかかわらず、生成されます。
__c33std	-mc33adv、-mc33peのオプション指定がない場合に生成されます。
__c33adv	-mc33advオプションが指定されている場合に生成されます。
__c33pe	-mc33peオプションが指定されている場合に生成されます。

-O0, -O, -O2, -O3, -Os**機能: 最適化****説明:** いずれか1つのスイッチを指定し、最適化処理を行います。

速度とサイズを重視して最適化されたコードが生成されます。ただし、-O2/-O3は速度のみ、-Osはサイズのみを最適化の対象にしています。スイッチが指定されない場合は最適化は行われません。-Oの数値が上がるほど最適化機能は強化されますが、デバッグ情報が一部出力されないなどの問題が発生することがありますので、注意してください。正常に実行できないときは、最適化の数値を下げてください。最適化の段階でレジスタのインターロックは考慮されません。-O2/-O3は速度の最適化を目的としていますので、-Oよりもサイズが大きくなる場合があります。通常は、-Oでコンパイルすることを推奨します。

IDEが生成する基本的なmakeファイルは、C/C++コンパイラ**xgcc**の起動コマンドに-Oオプションを使用しています。

以下に、各オプションの特徴を説明します。

-O0

最適化を行いません。

未使用のローカル変数が宣言された場合でも、スタックに領域を確保します。コードはそのままコンパイルし、どこからも参照されないローカル変数に値を代入するというような、不要のコードもそのまま生成します。レジスタにロードした変数の値を再利用しません。ただし、register宣言されたローカル変数は最適化され、不要であれば削除します。

-O/-O1

コードのサイズと実行速度の最適化を行います。

実行される最適化としては、以下の点などが挙げられます。

不要なコードは削除します。(どこからも参照されないローカル変数に値を代入するなど)

変数の処理にレジスタを割り当て、その値を再利用することにより、メモリへのリード/ライトの回数を減らすようにします。

そのため、メモリへのアクセスの発生を保障できなくなりますので、確実にメモリのリード/ライトが必要な変数はvolatile宣言が必要になります。

ループ処理の最適化を実行します。

分岐条件を予測し最適化を行うことより、重複した比較命令を繰り返さないようにします。

-O2

-Oよりも更に、コードの実行速度の最適化を行います。

-Oよりも強化される最適化としては、以下の点などが挙げられます。

グローバル領域での共通の演算処理を一度の演算に置き換えます。(グローバル共通部分式の削除)

ループ処理の最適化を2回、実行します。

ld命令などの単純な命令でのオペランドにおいて、レジスタの割り当ての最適化を行います。

到達することのない分岐条件ブロックを無視し、コードの生成を行いません。

ほとんどの最適化は有効になりますが、inline宣言のない関数のインライン展開は行いません。

-O3

-O2よりも更に、コードの実行速度の最適化を行います。ただし、ソースコードの内容によっては、-O3の結果が最速の実行速度にならないこともあります。

-O2よりも最適化の強化される点として、inline宣言のない関数のインライン展開を行い、単純なコードのサブルーチンはコールする代わりにその関数本体のコードをコピーします。これにより、関数コールのオーバーヘッドを削除します。

-Os

コードサイズの最適化を行います。

-O2の最適化の中でコードのサイズを増加させない処理をすべて行います。

コードの実行速度は、-O1/-O2/-O3に比べて遅くなる傾向があります。

デフォルト: コードの最適化は行われません。

-gstabs

機能: ソースファイルの相対パスを含むデバッグ情報の付加

説明: デバッグ情報を含む出力ファイルが生成されます。

ソースファイルの位置情報は相対パスで出力されます。**IDE**が生成する基本的なmakeファイルは、C/C++コンパイラ**xgcc**の起動コマンドにこのオプションを使用しています。デバッグ情報はデバッガ**gdb**がデバッグ時に使用するのみです。このオプションは、必ず付けてください。

デフォルト: デバッグ情報は出力されません。

-fPIC

機能: 位置独立コードの生成

説明: このオプションを指定すると、C/C++コンパイラ**xgcc**はダイナミックリンクに適した位置独立コード(PIC)を生成します。ライブラリモジュールや再入可能なモジュールを生成する場合に指定してください。

このオプションと-medda32オプションは併用できません(C33 STDコア/C33 PEコアのときのみ)。

デフォルト: 位置独立コードは生成されません。

標準オプションに加え、以下のS1C33専用のオプションも用意されています。

-mc33adv

機能: C33 ADVコア用コードの生成

説明: このオプションを指定すると、C/C++コンパイラ**xgcc**はC33 ADVコア用のコードを生成します。標準命令に加えC33 ADVコア用の命令も出力されます。また、デフォルトデータエリアポインタとして、%r15レジスタの代わりに%dpレジスタが使用されます。

デフォルト: C33 STDコア用のコードを生成します。

-mc33401

機能: C33 ADVコアのアセンブラフィルタ

説明: C33 ADVコアのアセンブラ命令の組み合わせによっては、割り込みが発生したときに正常に動作しないことがあります。本オプションは、この問題を回避するためのアセンブラフィルタ機能を有効にします。

本オプションは、必ず-mc33advオプションと一緒に指定してください。

アセンブラ命令の組み合わせ等の詳細は、"[¥gnu33¥doc¥as_fil_readmeja.txt](#)"を参照してください。

デフォルト: アセンブラフィルタ機能は無効です。

-mc33pe

機能: C33 PEコア用コードの生成

説明: このオプションを指定すると、C/C++コンパイラ**xgcc**はC33 PEコア用のコードを生成します。標準命令に加えC33 PEコア用の命令も出力されます。

デフォルト: C33 STDコア用のコードを生成します。

-medda32

機能: デフォルトデータエリアの未使用

説明: 外部変数へのアクセス方法を指定するオプションです。

32ビットのメモリ空間を使用したい場合に指定してください。

指定しない場合は、外部変数へのアクセスがデータポインタから26ビットまでに制限されますが、コードサイズは小さくなります。このオプションはC33 STDコアおよびC33 PEコアにのみ有効です。

表6.3.2.1 -medda32オプション

CPU	-medda32	説明	Cコード	アセンブラコード	コードサイズ
C33 STD/ C33 PE	あり	デフォルトデータエリアポインタを使用せずに外部変数にアクセスします。	i = i_Global_Val;	xld.w %r4,i_Global_Val ld.w %r4,[%r4]	8バイト
		コンパイラは%r15を使用したコードを出力しません。 -fPICオプションとは併用できません。	i_Global_Val = 1;	xld.w %r5,1 ;0x1 xld.w %r4,i_Global_Val ld.w [%r4],%r5	14バイト
	なし	%r15をデフォルトデータエリアポインタとして使用します。プログラムの先頭で%r15を初期化してください。	i = i_Global_Val;	ext imm13-25(i_Global_Val) ext imm0-12(i_Global_Val) ld.w %r4,[%r15]	6バイト
		外部変数へのアクセスが%r15から26ビットまでに制限されますが、コードサイズは小さくなります。	i_Global_Val = 1;	xld.w %r4,1 ;0x1 ext imm13-25(i_Global_Val) ext imm0-12(i_Global_Val) ld.w [%r15],%r4	12バイト
C33 ADV	指定 無効	C33 ADVコアでは、このオプションによる出力コードの違いはありません。	i = i_Global_Val;	ext imm19-31(i_Global_Val) ext imm6-18(i_Global_Val) ld.w %r4,[%dp+imm0-5(i_Global_Val)]	6バイト
		%dpをデフォルトデータエリアポインタとして使用します。プログラムの先頭で%dpを初期化してください。	i_Global_Val = 1;	xld.w %r4,1 ;0x1 ext imm19-31(i_Global_Val) ext imm6-18(i_Global_Val) ld.w [%dp+imm0-5(i_Global_Val)],%r4	12バイト

* i: ローカル変数、i_Global_Val: 外部変数

デフォルト: デフォルトデータエリアを使用します。

-mlong-calls

機能: 関数コールの展開

説明: このオプションを指定すると、C/C++コンパイラ**xgcc**は関数コールを32ビットディスプレースメント付きの3命令に展開します。プログラムの実行領域が2Mバイトを超える場合に指定してください。

デフォルト: 通常、すべての関数コールは、±2Mバイトの範囲で分岐が可能な22ビットディスプレースメント付きの2命令に展開されます。

-mno-memcpy機能: **strcpy**、**memcpy**のインライン展開説明: **strcpy**および**memcpy**関数コールがインライン展開されます。

デフォルト: インライン展開されません。

-Wall

機能: ワーニングオプションの有効化

説明: 以下のワーニングオプションをすべて有効にします。
これらのワーニングオプションは、**-Wno-** と付けることで個別に無効にすることができます。例えば **-Wcomment** のワーニングのみを無効にしたい場合は、**-Wall** の後ろに **-Wno-comment**を追加します。**-Wchar-subscripts**配列の添字の型が **char** 型である場合にワーニングを出力します。**-Wcomment**コメントの開始文字列である **"/**** が **"/**** で始まるコメントの中にある場合にワーニングを出力します。または、**"//"** で始まるコメントがバックスラッシュで終わっている場合にワーニングを出力します。**-Wformat****printf** 関数や **scanf** 関数などを呼び出した時に、変換文字列に対し引数が適切であるかを確認します。また、変換文字列で指定された変換が妥当であるかを確認します。**-Wimplicit-int**

変数や関数を宣言する時に型が指定されていない場合にワーニングを出力します。

-Wimplicit-function-declaration

宣言される前に関数を使用した場合にワーニングを出力します。

-Wimplicit**-Wimplicit-int** および **-Wimplicit-function-declaration** を有効にしたものです。

-Wmain

main 関数の型が正しくない場合にワーニングを出力します。**main** 関数は、外部リンケージを持ち、戻り値は **int** 型であり、0個または2個または3個の適切な型の引数を持つべきとされています。

-Wmissing-braces

配列などを初期化する時に、十分に括弧でくくられていない場合にワーニングを出力します。多次元配列を初期化する時に、各次元ごとに括弧でくくられていない場合などが該当します。

例： `long l_Array_1[3][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };`
(ワーニングを出力します)

`long l_Array_2[3][3] = { { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 } };`
(ワーニングを出力しません)

-Wparentheses

括弧が省略されて記述が混乱する場合に、ワーニングを出力します。ネストした **if** 文などにおいて `{ }` が省略されている場合などが該当します。

-Wsequence-point

標準のC言語仕様においては、実行する順序が正確に規定されていないため、振る舞いが未定義になる可能性のあるコードを記述した場合にワーニングを出力します。

例： `i_Array[i_Val++] = i_Val;`

-Wreturn-type

関数を定義する時に戻り値の型が指定されていない為、デフォルトの **int** 型として定義される場合にワーニングを出力します。または、戻り値が **void** 型以外の関数であるにも関わらず、値を返していない場合にワーニングを出力します。

-Wswitch

switch 文が **enum** 型の変数をインデックスとする場合に、**enum** 型のすべての値に対応する **case** 文がない場合にワーニングを出力します(**default** ラベルがある場合は、このワーニングは出力されません)。または、**enum** 型の範囲を超えた値を **case** 文として指定した場合にワーニングを出力します。

-Wunused-function

static 関数が宣言されているにも関わらず定義されていない場合にワーニングを出力します。または、**inline** でない **static** 関数が定義されているにも関わらず未使用の場合にワーニングを出力します。

-Wunused-label

ラベルが宣言されているにも関わらず未使用の場合にワーニングを出力します。

-Wunused-variable

ローカル変数や **const** でない **static** 変数が宣言されているにも関わらず未使用の場合にワーニングを出力します。

-Wunused-value

使用されないことが明白にも関わらず、計算をする場合にワーニングを出力します。

-Wunused

上記の **-Wunused-xxxx** をすべて有効にしたものです。

-Wuninitialized

ローカル変数が初期化されずに使用されている場合にワーニングを出力します。このワーニングは **-O0** の時は出力されません。

デフォルト： 上記のワーニングオプションは無効になります。

-Werror-implicit-function-declaration

機能: 未宣言の関数をエラー出力

説明: Cソースファイルで、宣言される前に関数を使用した場合にエラーを出力します。
このオプションはCソースファイル(*.c)のみに指定可能です。
C++ソースファイル(*.cpp, *.cc, *.cxx, *.C)には指定できません。
C++ソースファイルでは、宣言される前に関数を使用した場合は、必ずエラーを出力します。

デフォルト: Cソースファイルで、宣言される前に関数を使用した場合でも、エラーを出力しません。

-mno-sjis-filt

機能: Shift JISコードのフィルタ機能を無効化

説明: Shift JISコードのフィルタ機能を無効化します。
このフィルタ機能の詳細については、「6.8 Shift JISコードのフィルタ機能」を参照してください。

デフォルト: プリプロセッサは、Shift JISコードのフィルタ処理を行います。

-xassembler-with-cpp

機能: Cプリプロセッサの呼び出し

説明: アセンブル前にCプリプロセッサ**cpp**を実行します。本オプションを付加することにより、アセンブラソース内で擬似命令(**#define**、**#include**など)を使用することが可能になります。

デフォルト: Cプリプロセッサの呼び出しは行いません。

コマンドラインにオプションを入力する場合、オプションの前後には1個以上のスペースが必要です。

例: `xgcc -c -gstabs test.c`

注: • ANSIライブラリは、**%r15**をデフォルトデータエリアポインタとして使用しません。

ミドルウェアについては、各ミドルウェアマニュアルを参照してください。

- 同じコンパイラオプションを異なる設定で複数指定した場合、動作は不定となります。
- **xgcc**を呼び出す場合は、必ず**-S**、**-E**または**-c**オプションのいずれかを指定してください。これらのオプションが指定されない場合、**xgcc**はリンク処理まで実行しますが、必要なリンクのオプションを指定することができません。
- **-medda32**オプションと**-fPIC**オプションは併用できません(C33 STDコア/C33 PEコアのときのみ)。

6.4 コンパイラの実出力

ここではC/C++コンパイラ**xgcc**が出力するアセンブリソースや使用するレジスタについて説明します。

6.4.1 出力内容

C/C++コンパイラ**xgcc**はC/C++ソースをコンパイルし、以下の内容を出力します。

- C33 STDコア命令セットのニーモニック
- C33 ADVコア命令セットのニーモニック(-mc33advオプション指定時)
- C33 PEコア命令セットのニーモニック(-mc33peオプション指定時)
- 拡張命令のニーモニック
- アセンブラ擬似命令

基本命令以外は拡張命令で出力されます。

システム制御命令やmac命令はC/C++ソースでは表現できませんので、asmによるインラインアセンブルまたはアセンブリソースファイルで対応してください。

例: `asm("mac %r12");`

セクションやデータ定義にはアセンブラ擬似命令を出力します。

命令やデータが設定されるセクションは以下のとおりです。

命令

すべて、.textセクションに配置されます。

定数

.rodataセクションに配置されます。

```
例: const int i=1;          .global    i
                             .section   .rodata
                             .align     2
                             .type      i,@object
                             .size      i,4

                             i:
                             .long      1
```

初期値を持つグローバル変数とスタティック変数

.dataセクションに配置されます。

```
例: int i=1;               .global    i
                             .section   .data
                             .align     2
                             .type      i,@object
                             .size      i,4

                             i:
                             .long      1
```

初期値を持たないグローバル変数とスタティック変数

.bssセクションに配置されます。

```
例: int i;                 .global    i
                             .section   .bss
                             .align     2
                             .type      i,@object
                             .size      i,4

                             i:
                             .zero      4
```

関数名やラベル等、すべてのシンボルにはアセンブラ擬似命令.stabによるシンボル情報が挿入されます(-gstabsオプション指定時)。

6.4.2 データ表現

C/C++コンパイラ**xgcc**はANSI C/C++のすべてのデータ型に対応しています。各型のサイズ(バイト数)、表現できる数値の有効範囲を表6.4.2.1に示します。

表6.4.2.1 型の種類とサイズ

型	サイズ	数値の有効範囲
char	1	-128～127
unsigned char	1	0～255
short	2	-32768～32767
unsigned short	2	0～65535
int	4	-2147483648～2147483647
unsigned int	4	0～4294967295
long	4	-2147483648～2147483647
unsigned long	4	0～4294967295
pointer	4	0～4294967295
float	4	1.175e-38～3.403e+38(正規化数)
double	8	2.225e-308～1.798e+308(正規化数)
long long	8	-9223372036854775808～9223372036854775807
unsigned long long	8	0～18446744073709551615
bool	1	true/false
wchar_t (C)	2	0～65535
wchar_t (C++)	4	-2147483648～2147483647

float型、double型はIEEE標準規格のフォーマットに準拠しています。

long long型の定数を扱う場合は、接尾子LLまたはll(long long型)/ULLまたはull(unsigned long long型)が必要になります。この接尾子がないと、コンパイラはlong long型の定数として認識できないため、ワーニングになります。

```
例: long long ll_val;
    ll_val = 0x1234567812345678;
        → warning: integer constant is too large for "long" type
    ll_val = 0x1234567812345678LL;
        → OK
```

bool型はtrue(真)/false(偽)を扱うためのデータ型です。Cでは、stdbool.hをインクルードする必要があります。C++では独立した型になります。

wchar_t型はワイド文字を扱うためのデータ型であり、Cではstdlib.h/stddef.h内で、unsigned short型として定義されています。

また、C++では符号付き4バイトの独立した型になります。C++において、wchar_tを使用するには、接頭辞Lを指定する必要があります。これは、コンパイラにワイド文字列として扱うように指示するためです。

```
例: wchar_t buf[] = L"123";
```

●メモリ上の格納位置

データのメモリ上の格納位置はデータの型に依存します。データは、グローバルかローカルかにかかわらずメモリアドレスがそのデータ型のサイズで割り切れるような場所からサイズ分配置されます。double型は4バイトの境界アドレスにアライメントされ、データの低位4バイト(仮数部(31:0))、がメモリの低位側の4バイトに、上位4バイトデータ(符号、指数部、仮数部(51:32))がメモリの上位側4バイトに格納されます。

●構造体データ

構造体データはdouble型と同様4バイト境界(4で割り切れるアドレス)から配置されます。メンバは定義の出現順に各データ型のサイズに従って配置されます。構造体の定義と配置の例を次に示します。

```
例: struct Sample {
    char      cData;
    short     hData;
    char      cArray[3];
    int       iData;
    double    dData;
};
```

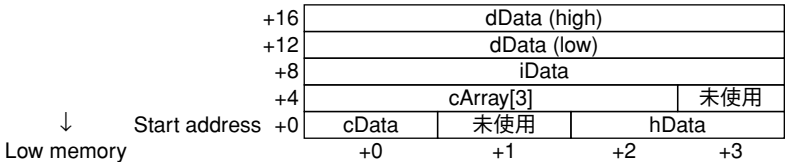


図6.4.2.1 構造体データのメモリ配置例

図のように、メンバのデータ型により未使用領域ができることがあります。

6.4.3 レジスタ使用法

C/C++コンパイラxgccは汎用レジスタを次のように使用します。

表6.4.3.1 汎用レジスタの使用法

レジスタ	使用法
%r0	フレームポインタとして使用されるレジスタ 関数呼び出し時に値を保存する必要があるレジスタ
%r1 %r2 %r3	関数呼び出し時に値を保存する必要があるレジスタ
%r4 %r5	戻り数格納用レジスタ(8/16/32ビットデータ、double型データの下位32ビット) 戻り数格納用レジスタ(double型データの上位32ビット)
%r6 %r7 %r8 %r9	引数渡し用レジスタ(第1ワード) 引数渡し用レジスタ(第2ワード) 引数渡し用レジスタ(第3ワード) 引数渡し用レジスタ(第4ワード)
%r10 %r11 %r12 %r13 %r14	スクラッチレジスタ/未使用
%r15	デフォルトデータエリアポインタレジスタ*
%dp	デフォルトデータエリアポインタレジスタ(-mc33advオプション(C33 ADVコア)指定時)

* -medda32オプション(デフォルトデータエリア未使用)未指定時

●関数呼び出し時に値を保存するレジスタ(%r0～%r3)

式の計算結果やローカル変数が格納されます。これらのレジスタは、関数からリターンした際に呼び出し時の値を保持していなければなりません。したがって、呼び出された関数内でこれらのレジスタを変更する場合、その関数はsave/restoreによりレジスタの内容を退避/復帰する必要があります。

●フレームポインタとして使用されるレジスタ(%r0)

alloca() や可変長配列を使用する場合に、フレームポインタが必要になります。その場合に、フレームポインタとして使用されます。ただし、フレームポインタが必要でない場合は、従来どおり関数呼び出し時に値を保存するレジスタとして使用されます。

●戻り値格納用レジスタ(%r4,%r5)

関数の戻り値を格納します。戻り値を格納する前はスクラッチレジスタとして使用されます。

●引数渡し用レジスタ(%r6～%r9)

関数呼び出し時に引数を格納します。4ワードを超えた分はスタック内に格納して渡します。引数を格納する前はスクラッチレジスタとして使用されます。

●スクラッチレジスタ(%r10～%r14)

式の計算結果やローカル変数が一時的に格納されます。これらのレジスタは関数呼び出し時に保存する必要はありません。

●デフォルトデータエリアポインタレジスタ(%r15または%dp)

デフォルトデータエリアの開始アドレスを示すデータエリアポインタ値を格納します。-mc33advオプションが指定されないと%r15が、指定されると%dpをデフォルトデータエリアポインタとして使用します。

-medda32オプション指定時はデフォルトデータエリアを使用しないため、%r15もデフォルトデータエリアポインタとしては使用されません。

ただし、C33 ADVコアは-medda32オプション指定のあり/なしにかかわらず、%dpをデフォルトデータエリアとして使用します。

注: C/C++ルーチンから呼び出されるアセンブラルーチンの中では、レジスタの使い方に注意してください。

- `%r0`～`%r3`を使用する場合は、使用する前に必ず`pushn`命令で内容を退避させ、リターン前に`popn`命令で復帰させてください。
- `%r4`と`%r5`はリターン前に戻り値をセットするまでは自由に使用できます。
- `%r6`～`%r9`は引数を使い終わった後は自由に使用できます。リターン前に復帰させる必要はありません。
- 構造体を返す関数への引数渡し
戻り値となる構造体データが8バイトに収まる場合、そのデータは戻り値格納用レジスタ`%r4`と`%r5`に入れて返されます。この場合、通常は第1引数となる構造体へのポインタが関数に渡されることはありません。

6.4.4 関数呼び出し

●引数の渡し方

関数呼び出しの際、引数は4ワードまでが引数渡し用レジスタ(%r6~%r9)に、それを超える分は呼び出し側関数のスタックフレーム(次節で説明)に格納されて呼び出される関数に渡されます。

例: `func(w1, w2, w3, w4, w5, w6);`

... `w1→%r6, w2→%r7, w3→%r8, w4→%r9, w5&w6→スタック`

(wN: ワードサイズ以下の引数)

基本的には、引数の指定順に%r6~%r9に格納されます。

●引数のデータサイズ

4バイト以下のデータサイズの引数は、データ型にかかわらずワード(4バイト)単位で扱われます。`char`型、`short`型のデータは符号拡張、`unsigned char`型、`unsigned short`型はゼロ拡張されます。

`double`型のみ8バイト単位で扱われます。`double`型引数を渡す場合、%r6~%r9に2つのレジスタの空きがなければスタック渡しとなります。

例: `func(w1, d2, d3, w4);`

... `w1→%r6, d2(L)→%r7, d2(H)→%r8, w4→%r9, d3→スタック`

(wN: ワードサイズ以下の引数, dN: `double`型の引数)

●構造体引数の扱い(注)

引数が構造体データの場合、構造体のメンバの値はスタックを介して渡されます。

例: `struct _foo {`

`int    a;`

`short  b;`

`char   c;`

`};`

`callee(struct _foo foo, int d);`

上記例ではdのみが引数渡し用レジスタ(%r6)に格納され、fooはすべてスタックに格納されます。

●構造体を返す関数への引数渡し(注)

構造体データを返す関数を呼び出す場合、結果を格納する構造体のアドレスが第1引数として%r6レジスタに格納され、関数に渡されます。したがって、ソース上で記述されている引数は1つずつ繰り下げられます。戻り値となる構造体を使用されない場合、コンパイラはダミーの構造体データを呼び出し側関数のローカル変数領域に割り当て、そのアドレスを渡します。

呼び出された関数は第1引数で渡されたポインタを戻り値として返します。

戻り値となる構造体データが8バイトに収まる場合、そのデータは戻り値格納用レジスタ%r4と%r5に入れて返されます。この場合、通常は第1引数となる構造体へのポインタが関数に渡されることはありません。

●レジスタの退避

呼び出された関数内で%r0~%r3レジスタを変更する場合、その関数は`save/restore`により呼び出された時点のレジスタの内容を退避/復帰する必要があります。

%r4 ~ %r14レジスタは、自由に使用することができます。

●戻り値

ワードサイズ以下の戻り値は%r4レジスタに格納されます。

ダブルワード(`double`)サイズの戻り値は%r4(下位ワード)と%r5(上位ワード)に格納されます。

注: 構造体をサブルーチンに渡したり、サブルーチンから構造体を返す場合、その都度構造体全体を`memcpy`関数でコピーするという特殊な処理を行う実コードが生成されます。これはコードサイズや実行速度という面からも望ましくありません。

構造体はできるだけポインタで渡して処理することを推奨します。

6.4.5 スタックフレーム

C/C++コンパイラ**xgcc**は関数呼び出し時に図6.4.5.1に示すスタックフレームを生成します。スタックフレームの開始アドレスは常にワード(32ビット)境界アドレスとなります。

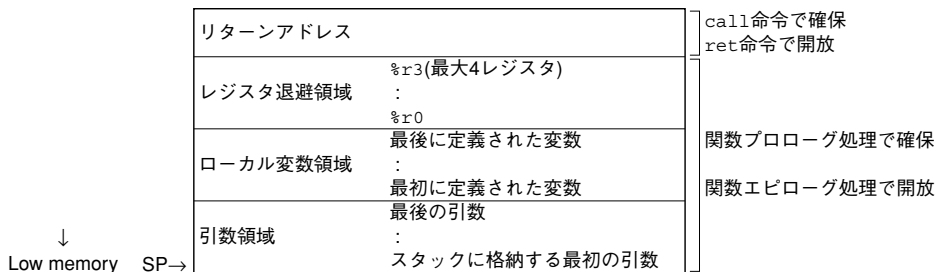


図6.4.5.1 スタックフレーム

●リターンアドレス

呼び出し側の関数へのリターンアドレス(1ワード)です。

●レジスタ退避領域

%r0～%r3レジスタの中で、呼び出し側の関数が使用しているレジスタがあれば、番号の大きい順に格納されます。

呼び出し側の関数が%r0～%r3レジスタを使用していない場合、この領域は確保されません。

●ローカル変数領域

呼び出された関数内で定義されているローカル変数で、レジスタに格納できないものについては、スタックフレーム内に領域を確保します。その後、最後に宣言された変数から順に、データ型に従った境界アドレス(double型は4バイト境界)に格納されます。

例: {

```
char    c;
short  s;
int    i;
      :
```

}

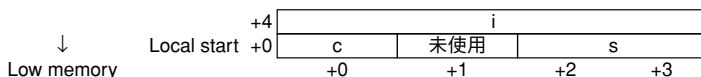


図6.4.5.2 ローカル変数の配置例

スタックに割り当てるローカル変数がない場合、この領域は確保されません。

●引数領域

呼び出された関数内にある別の関数呼び出しの引数の中で、引数渡し用レジスタに格納できないものについてスタックフレーム内に領域を確保します(前節参照)。引数はすべて4バイト境界に配置されます。double型の下位32ビットは下位側アドレスに、上位32ビットは上位側のアドレスに格納されます。関数呼び出しがない場合、この領域は確保されません。

●領域の確保と開放

リターンアドレス用の領域確保と格納はcall命令が行い、取り出しと領域の開放はret命令が行います。それ以外の領域は、関数のプロローグ処理が必要に応じて確保され、エピローグ処理で開放されます。

6.4.6 C/C++ソースの文法

データ型、ライブラリ関数とヘッダファイル、インラインアセンブル、プロトタイプ宣言(割り込み処理関数の宣言、セクションを指定するプロトタイプ宣言)については、"4.2 C/C++ソースの文法"を参照してください。

6.5 標準ヘッダファイル/標準ヘッダ

¥gnu33¥includeフォルダ内の".h"の付いているものは標準ヘッダファイル、".h"の付いていないものは標準ヘッダという分類になります。

Cソース内では標準ヘッダファイルをインクルードしてください。

C++ソース内では標準ヘッダをインクルードすることを推奨します。標準ヘッダは、名前空間stdに対応しています。

標準ヘッダファイル/標準ヘッダのディレクトリ構成は、以下の構成になっています。

```
include
  /backward
  /bits
  /c
  /c_compatibility
  /c_std
  /ext
  /machine
  /std
  /sys
```

includeディレクトリ以下に複数のサブフォルダがありますが、インクルードパスはincludeフォルダのみ指定します。これはC、C++共通です。

例: -I/cygdrive/c/EPSON/gnu33/include

以下に本パッケージで提供している標準ヘッダの一覧表を記載します。これらの標準ヘッダは、ISO/IEC 14882で定められているものです。

表6.5.1 C++ライブラリ用標準ヘッダー一覧

ヘッダ名	内容
algorithm	C++プログラムに適用できるアルゴリズムのテンプレートを定義しています。
bitset	bitsetコンテナのテンプレートを定義しています。
complex	複素数を扱うテンプレートを定義しています。
deque	dequeコンテナのテンプレートを定義しています。
exception	例外処理に関する型と関数を定義しています。
fstream	ファイルストリームクラスを定義しています。 ただし本パッケージでは、ファイルストリームはダミーの機能になります。
functional	関数オブジェクトを生成するためのテンプレートを定義しています。
io manip	入出力マニピュレータを定義しています。
ios	入出力ストリームクラスを定義しています。
iosfwd	入出力ストリームクラスの前参照を宣言しています。
iostream	標準ストリームを定義しています。
istream	入力ストリームクラスを定義しています。
iterator	反復子を定義しています。
limits	numeric_limitsクラスを定義しています。
list	listコンテナのテンプレートを定義しています。
locale	ロケールを定義しています。 ただし本パッケージでは、ロケール名はデフォルトの"C"/"POSIX"のみに対応しています。
map	mapおよびmultimapコンテナのテンプレートを定義しています。
memory	オブジェクトの割り当て、所有権の管理、解放を行うテンプレートを定義しています。
new	メモリ割り当てと開放の関数を宣言しています。
numeric	数値演算用のテンプレートを定義しています。
ostream	出力ストリームクラスを定義しています。
queue	queueおよびpriority_queueコンテナのテンプレートを定義しています。
set	setおよびmultisetコンテナのテンプレートを定義しています。
sstream	入出力文字列ストリームクラスを定義しています。
stack	stackコンテナのテンプレートを定義しています。
stdexcept	標準例外処理に関する型と関数を定義しています。
streambuf	ストリームバッファクラスを定義しています。
strstream	文字列シーケンス入出力クラスを定義しています。
string	basic_stringクラスを定義しています。
typeinfo	実行時型情報(RTTI)を扱うクラスを定義しています。
utility	比較演算子を定義しています。
valarray	数値を扱う配列のテンプレートを定義しています。
vector	vectorコンテナのテンプレートを定義しています。

表6.5.2 Cライブラリ用標準ヘッダー一覧

ヘッダ名	内容
cassert	assert.hに対応しています。
cctype	ctype.hに対応しています。
cerrno	errno.hに対応しています。
cfloat	float.hに対応しています。
ciso646	iso646.hに対応しています。
climits	limits.hに対応しています。
locale	locale.hに対応しています。
cmath	math.hに対応しています。
csetjmp	setjmp.hに対応しています。
csignal	signal.hに対応しています。
cstdarg	stdarg.hに対応しています。
cstddef	stddef.hに対応しています。
cstdio	stdio.hに対応しています。
cstdlib	stdlib.hに対応しています。
cstring	string.hに対応しています。
ctime	time.hに対応しています。
wchar	wchar.hに対応しています。 ただし、本パッケージにはwchar.hはありませんので、wchar.hに対応する箇所はコメントになっています。
wctype	wctype.hに対応しています。 ただし、本パッケージには wctype.hはありませんので、wctype.hに対応する箇所はコメントになっています。

注: 本パッケージが提供するANSI Cライブラリは、上記のCライブラリ用標準ヘッダー一覧内で宣言されている関数のすべてを実装していません。具体的な実装状況は、"7.4.2 ANSIライブラリ関数一覧"を参照してください。

6.6 C++使用時のポイント

C++の詳細な文法については、C++の文法を解説している一般の書籍を参照してください。
ここでは、本パッケージでC++を使用するときの主なポイントを説明します。

●必須の処理

初期化: .bss/.commセクションの0クリア
.dataセクションのROM→RAMへのコピー
グローバルクラスのコンストラクタ
libstdc++.aから提供されている_init()関数をコールしてください。
main()関数の直前で、実行する必要があります。
ヒープ領域の確保
STL(Standard Template Library)やstringクラスを使用する場合、ヒープ領域が必要になります。
lib.cから提供されているansi_InitMalloc()関数をコールしてください。

終了処理: グローバルクラスのデストラクタ
libstdc++.aから提供されている_fini()関数をコールしてください。
main()関数の直後で、実行する必要があります。

コンストラクタ関数/デストラクタ関数とは、クラスの生成時/消滅時に実行される関数のことです。
ローカルなクラスの場合には、そのクラスの生成/消滅時、すなわちそのローカルクラスを宣言している関数の最初/最後で実行されます。
それに対してグローバルなクラスの場合では、グローバル空間に存在し続けるため、その生成/消滅時の処理はmain()関数の前後で行います。

以下に初期化/終了処理の例を示します。

```
_init_section();          .bss/.commセクションの0クリア
_init_sys();              シミュレーテッドI/Oの初期化
_init_lib();              ANSIライブラリ用の変数の初期化
ansi_InitMalloc(ALLOC_START, ALLOC_END);
                           ヒープ領域の確保

_init();                  グローバルクラスのコンストラクタ関数を実行
main();
_fini();                  グローバルクラスのデストラクタ関数を実行
_exit();
```

¥gnu3¥utility¥sample_c++フォルダ内のvector.cに雛形がありますので、参照してください。
ヒープ領域は、サンプルではvector.c内のALLOC_START/ALLOC_ENDにて、0x600000～0x6fffffに定義されていますが、ご使用の環境に合わせて修正してください。

●ストリーム

標準ストリーム(cin/cout/cerr/clog)は、標準入出力(stdin/stdout/stderr)と同様に対応しています。これは、デバugggdbのシミュレーテッドI/O機能を通して、使用することができます。
これに対し、ファイルとの入出力を行うファイルストリームは未対応です。これは、ファイルストリームは内部的に、fopen()/fclose()などのファイル処理を行う標準ライブラリを呼び出しますが、本パッケージでは、これらの関数はダミー関数であるためです。

●デバuggdb上での処理

デバuggdbは、C++に対応しています。ただし、以下の制約があります。

- [式]ビューでクラスのメンバ変数まで、値を表示できます。ただし、C++ライブラリの一部のクラスは正確に型情報が取得できず、"error type"と表示される場合があります。
- [ステップ・オーバー]実行時に、Step/Goの動作をしてしまう場合があります。これが不便な場合は、-ooオプションを付けて再コンパイルし、デバuggdbで実行してみてください。[ステップ・オーバー]実行時の動作が大幅に改善されます。そして、リリース時には-oオプションで再コンパイルをするというやり方も一つの方法として提案します。ただし、-oで再コンパイルした後で、最終の動作確認は必ず行ってください。

●string型の初期化

string型に複数ある初期化方法の中で、変数のポインタを初期値として渡すという方法があります。このとき、初期化に使用する変数が0番地に配置されていると初期化に失敗しますので注意してください。

本パッケージでは0番地から外部変数を配置可能ですが、0番地から配置された外部変数がstring型の初期化に使用された場合、コンパイラはNULLポインタが渡されたと判断し、_exit()関数を呼び出してエラー終了します。

このような場合は、string型の初期化に使用する外部変数が0番地以外に置かれるようにメモリ配置を変更してください。

```
例: char c_array[5];           ← 0番地に配置
    std::string str1 = c_array; ← NULLポインタが渡されたと判断し、_exit()関数を
                                呼び出します。
```

6.7 gccコアのバージョンアップおよび機能の追加/削除

本パッケージ V3.0より、gccコアのバージョンアップ(ver.2.95.2→ver.3.3.2)および機能の追加/削除が行われました。

●追加された機能

- C++への対応
- long long型への対応
- `alloca()`/可変長配列への対応

`alloca()`を使用する場合は、`alloca.h`をインクルードしてください。

`alloca()`は、スタックオーバーフローのチェックをしていませんので、その時点でのスタックのサイズに注意してください。

●削除された機能

- S/T/Z/Gデータエリア

S/T/Z/Gデータエリアは、正式にサポートする機能ではなくなります。

データエリアは、デフォルトデータエリアのみ正式にサポートします。それに伴い、以下のコンパイラオプションは正式にサポートしません。

`-mdp`, `-mgda`, `-mgdp`, `-mezda`, `-metda`, `-mesda`

機能自体は残っていますが、正式なサポート対象ではなくなります。使用する場合は、お客さまの責任においての使用となります。

●gccコアver.2.95.2とver.3.3.2の相違点

gccコアのバージョンアップによるレジスタの使用方法や、スタックフレームの構成など基本的な構成には違いはありません。ただし、主に以下の相違点があります。

- インラインアセンブラを複数行で宣言するときは、行末に"`¥n`"が必要になります。

例:

```
ver.2.95.2)
asm("xxxxx
    xxxxx
    xxxxx");
```

```
ver.3.3.2)
asm("xxxxx¥n¥
    xxxxx¥n¥
    xxxxx");
```

- 型のチェックが厳密になり、`extern`宣言と実体の宣言で`volatile/const`が一致しない場合、エラーになります。

例:

```
test.h)
extern int i_Val;

test.c)
#include <test.h>
const int i_Val;
```

```
ver.2.95.2)
OK
```

```
ver.3.3.2)
error: conflicting types for 'int i_Val'
error: previous declaration as 'const int i_Val'
```

- ソースファイルの最後が改行コードでない場合、以下のワーニングになります。

```
warning: no newline at end of file
```


6.8 Shift JISコードのフィルタ機能

●機能の説明

GNU オリジナルのプリプロセッサ/コンパイラは、Shift JISコード(以下、SJISコードと記載)に完全には対応できていません。そのため、SJISコード上で、例えば"能"(0x945c)という文字があると、0x5c部分を行連結子(¥)と誤って判断し、正しく処理できません。

例:

```
i_Val = 0;           // 機能
i_Val = 1;           ← この行が上の行に連結されてしまい、コメントとして処理されてしまいます。
```

この誤動作に対応するため、本パッケージのVer.3.3.0以降のプリプロセッサ/コンパイラには、SJISコードをフィルタする機能が組み込まれています。

このフィルタ機能は0x5cというコードがある場合、その直前の1バイトをチェックし、0x5cが行連結子(¥)を表すものか、それともSJISコードの2バイト目のものかを判断します。

そして、SJISコードの2バイト目であると判断された場合は、行連結子(¥)として誤動作しないように処理します。

この機能の対象ファイルは以下の通りです。

- C/C++ ソースファイル
- C/C++ ソースファイルからインクルードされたヘッダファイル
- アセンブリソースファイル
- アセンブリソースファイルからインクルードされたヘッダファイル

このフィルタ機能を無効にする場合は、"-mno-sjis-filt"オプションを指定してください。

-mno-sjis-filt は IDE上から、以下の手順により設定できます。

設定するプロジェクトのコンテキストメニューから[プロパティ] > [GNU33 ビルドオプション] > [コンパイラ] > [一般]を選択します。

ここで、[漢字フィルタを使用する]のチェックを外します。

●注意点

- 未サポートですが、プリプロセッサに用意されている"-traditional-cpp"というオプションを指定してビルドした場合、フィルタ機能は正しく動作しません。
"-traditional-cpp"は、ISO の規格以前の古いルールに従ってプリプロセスの処理を行うように指定するオプションです。
- フィルタ処理が有効な場合は、シングルクォテーション(')で全角文字1文字を囲っているコードの出力結果がVer3.3.0より前のバージョンと変わります。
正しい出力を得るには接頭子Lを付ける必要があります。

例:

全角文字'空'(SJISコード上で0x8bf3)を代入するケース

- Ver3.3.0より前のバージョン、もしくはVer3.3.0以降のバージョンで-mno-sjis-filt オプションが指定されている場合

```
int i_Val = '空';      → i_Val に0x8bf3 が代入されます。
```

- Ver3.3.0 以降のバージョンで-mno-sjis-filtオプションが指定されていない場合

```
int i_Val = L'空';     → 接頭子Lを付けることにより、i_Val に0x8bf3が代入されます。  
                        接頭子Lを付けない場合は、i_Val に0xffff3が代入されます。
```

6.9 Cとアセンブラの混在法

引数、戻り値、レジスタ内容保護などのルールを守ることにより、Cとアセンブラルーチン間を自由に行き来することができます。

●Cからコールされるアセンブラルーチンの作成法

```

;*****
; strcpy
;     string copy from src to dest until 0 terminate
;
; arguments : %r6:dest addr, %r7:src addr (0 terminate string)
; return    : %r4:dest addr
;*****

        .section      .text
        .align        1
        .global       strcpy
        .type          strcpy,@function

strcpy:
    ld.w            %r4, %r6            ; return dest add

strcpy_loop:
    ld.ub           %r10, [%r7]+        ; copy src 1 byte to dest
    ld.b             [%r6]+, %r10
    cmp             %r10, 0             ; continue until 0 terminate
    jrne            strcpy_loop
    ret

```

このルーチンは、Cルーチンから次のようにコールされます。

```

#include <string.h>
int main()
{
    char *pchMem;           /* for malloc, strcpy */
    strcpy(pchMem, "This is strcpy test");
}

```

第1引数は%r6レジスタ、第2引数は%r7レジスタに入れて渡され、戻り値は%r4レジスタに返されます。

このように

- 第1引数から第4引数は%r6から%r9レジスタに入れて渡されます。
 - 特殊なケースや第5以降の引数はスタックに入れて渡されます(実際にコンパイルされたコードを参考にしてください)。
 - 戻り値は、%r4レジスタに入れて返されます。
- というのが引数と戻り値に使用するレジスタのルールです。

また、Cからコールされるアセンブラルーチン内でのレジスタ使用に関する制限は次のとおりです。

- %r0～%r3を使用する場合は、使用する前に必ずpushn命令で内容を退避させ、リターン前にpopn命令で復帰させてください。
- %r4と%r5はリターン前に戻り値をセットするまでは自由に使用できます。
- %r6～%r9は引数を使い終わった後は自由に使用できます。リターン前に復帰させる必要はありません。
- 構造体を返す関数への引数渡し
戻り値となる構造体データが8バイトに収まる場合、そのデータは戻り値格納用レジスタ%r4と%r5に入れて返されます。この場合、通常は第1引数となる構造体へのポインタが関数に渡されることはありません。

次の例では%r0～%r3レジスタの内容を保存し、リターン前に戻しています。

```

__adddf3:
    pushn %r3          ; save register values
    |
    popn  %r3          ; restore register values
    ret

```

●Cの関数をコールするアセンブラルーチンの作成法

Cの関数は上記のルールに沿ってコンパイルされます。したがって、Cの関数をコールするアセンブラルーチンは以下の点に注意して作成してください。

引数、戻り値の受け渡しルール

- 第1引数から第4引数は%r6から%r9レジスタに入れて渡します。
- 戻り値は、%r4レジスタで受け取ります。

リターン時のレジスタの状態

- %r0から%r3レジスタはコール時の内容を保持しています。
- %r4から%r15レジスタ、%ahr、%alr、%psrは変更されている可能性があります。

●ヘッダファイルのアセンブラとCでの共有

S5U1C33001Cではヘッダファイル(*.h)をアセンブリソースとCソースで共有することが可能です。アセンブラ、Cソース共に、プリプロセッサ定義を使用する個所よりも前でプリプロセッサ命令#include "filename.h"を使用することで参照できます。プリプロセッサの機能を使用するため、アセンブリソースをアセンブルするときは必ず**xgcc**を通し、**-xassembler-with-cpp**オプションを指定するようにしてください。**as**単体のみでアセンブルしようとした場合、エラーが発生します。アセンブラ固有の機能を#define命令で定義し、それをCソース内で使用することはできません。また、C言語固有の機能をアセンブリソース内の#defineで使用することはできません。

悪い例1:

```

(header1.h)
#define REF_SYMBOL .extern symbol          ← アセンブラ固有の機能です。

(source.c)
    REF_SYMBOL;                             ← 構文エラーが発生します。
    int main( void ) {
    }

```

悪い例2:

```

(header1.h)
#define SUCCESS_FLAG(x) ( x>= 0 )          ← C固有の機能です。

(source.s)
    cmp SUCCESS_FLAG(%r0)                   ← 構文エラーが発生します。

```

アセンブリソース内で#if、#ifdef、#ifndefといったプリプロセッサに依存した分岐を行わないでください。デバッガ上でソースと実際のアセンブラ行がずれて表示されます。また、アセンブリソース内で#includeによりヘッダファイルを参照する場合、参照するヘッダファイル中で#includeを使用しないでください。この場合もソースと実際のコードがずれて表示されます。

悪い例:

```

(header1.h)
#include "header2.h"
|
(src.s)
#include "header1.h"
|
ネストしてヘッダが参照されています。

```

良い例:

```

(header1.h)
/* #include "header2.h" */
|
(src.s)
#include "header2.h"
#include "header1.h"
|

```

以下にヘッダを共有した使用例を示します。

この例ではタスク登録をする際に共通ヘッダで定義されているTASK_Xをアセンブリソース内で使用しています。ここではアセンブラとCソースで共に使用できる定数のマクロ置換機能を使用しています。

次のヘッダファイルは、アセンブラとCソースで共通に使用できます。

#defineプリプロセッサ命令を使用して、プリプロセッサが実行される時点で決定する値のみを定義しています。外部のファイル、もしくはリンクによって決定する値を#defineにより定義する場合は、Cもしくはアセンブラそれぞれの方法に従って記述する必要があります。

(share.h)

```
/* Macros in this header are used in both C and assembler sources. */

#define TASK_MASK                0x00ff
#define TASK_DISABLE            0x0000
#define TASK_ENABLE             0x0001
#define TASK_DEFAULT            TASK_ENABLE
#define TASK_USINGMASK          0xff00
#define TASK_N( x )             ( 0x0100 << x )
#define TASK_0                  TASK_N( 0 )
#define TASK_1                  TASK_N( 1 )
#define TASK_2                  TASK_N( 2 )
#define TASK_3                  TASK_N( 3 )
#define TASK_4                  TASK_N( 4 )
#define TASK_5                  TASK_N( 5 )
#define TASK_6                  TASK_N( 6 )
#define TASK_7                  TASK_N( 7 )
#define MAX_TASK_NUM            8

#define TASK_0_7                ( TASK_0 + TASK_1 + TASK_2 + TASK_3 +
                                TASK_4 + TASK_5 + TASK_6 + TASK_7 )
#define TASK_NOT0               TASK_0_7 - TASK_0
#define TASK_0_3               TASK_0 * 0xf

/*****
/* when defined, application go to start routine if exit.      */
*****/
#define GOSTART_IFAPPEND

/*****
/* if you change max task number, enable below line.          */
*****/
// #define OVERRIDE_MAXTASK 8
/* new MAX_TASK_NUM ( max task number is 8. ) */
#define OVERRIDE_MAX_TASK_NUM 8

/*****
/* if you don't use any tasks, enable below line.              */
*****/
// #define NO_TASK

/*****
/* psr default setting                                          */
*****/
/* This configuration psr is set at the top of program. */
/* if you don't permit interruption, comment here.*/
#define IE_BIT 0x10
/* set default interrupt-level( In many case, below should be 0). */
#define DEFAULT_INTERRUPT_LEVEL 0

#define DEFAULT_PSR ( IE_BIT | ( DEFAULT_INTERRUPT_LEVEL << 8 ) )

/* dummy definition */
#define MACRO_DUMMY
#undef MACRO_DUMMY
```

次のファイルではアセンブラでのみ参照可能な定義を行っています。アセンブラでプリプロセッサ定義を使用する場合、文字列や数字の定義による置換が有効です。

(only.s)

```
#define GETTASK_H               ext doff_hi( ulTaskManage )
#define GETTASK_L               ext doff_hi( ulTaskManage )
#define GETTASK( register )    ld.w register, [%r15]
#define PUTTASK_H               ext doff_hi( ulTaskManage )
#define PUTTASK_L               ext doff_hi( ulTaskManage )
#define PUTTASK( register )    ld.w [%r15], register
```

以下はアセンブリソース内でプリプロセッサ定義を使用した例です。

(asm.s)

```

/* Nesting header files is not supported in assembler sources. */
#include "shareh.h"
#include "onlys.h"

        |
        xld.w      %r0,DEFAULT_PSR      ← プリプロセッサ定義を使用
                                           /* Set IE, IL. See shareh.h About DEFAULT_PSR. */

init_task:
        xld.w      %r6,TASK_0           ← プリプロセッサ定義を使用
        xld.w      %r7,DO_INT
        call       add_task
        ret

        |
SOFT_INT:
        pushn      %r14

        /* delete old task0 */
        xld.w      %r6,TASK_0           ← プリプロセッサ定義を使用
        call       delete_task
        /* add dummy task0 */
        xld.w      %r6,TASK_0           ← プリプロセッサ定義を使用
        xld.w      %r7,vfnSetDummy
        call       add_task

        /* add exit task */
        xld.w      %r6,TASK_7           ← プリプロセッサ定義を使用
        xld.w      %r7,EXIT_MAIN
        call       add_task

        popn       %r14
        reti

        |
/* exit all program */
/* --- no arguments --- */
EXIT_MAIN:
        /* read ulTaskManage */
        GETTASK_H                                     ← アセンブラ固有の
        GETTASK_L                                     ← 定義を使用
        GETTASK( %r4 )                                ←

        /* clear TASK_MASK part, and set TASK_DISABLE */
        xld.w      %r5,TASK_MASK
        not        %r5,%r5
        and        %r4,%r5

        /* write ulTaskManage */
        PUTTASK_H                                     ← アセンブラ固有の
        PUTTASK_L                                     ← 定義を使用
        PUTTASK( %r4 )                                ←
        ret

```

以下はCソースファイルでのみ使用可能なヘッダファイルの例です。ここではプリプロセッサ定義と `do { ... } while(1)` というCの構文を使用することで、擬似的にローカルな変数を作成しています。また、ファイルの先頭でヘッダファイルが二重に参照されてもエラーが出ないように細工してあります。ヘッダファイルの相互関係が複雑になる場合非常に有用です。

(csrc.h)

```

#ifndef _ONLYC_H_
/* _ONLYC_H_ */

#define _ONLYC_H_
#include "shareh.h"

/* macro definition changing a task bit to a task number. */
/* This definition can be used in only C-source files. */
#define TASK_NUM( task_bit, returned_tasknum ) ¥
        do { ¥
                unsigned long ulBuf; ¥
                ¥

```

```

        ulBuf = task_bit; ¥
        returned_tasknum = 0; ¥
        while ( (ulBuf & 0x100) == 0x0 ) { ¥
            returned_tasknum++; ¥
            ulBuf >>= 1; ¥
        }; ¥
    } while ( 0 );

/* #if, #ifdef or other branch prepro-instructions are
 *   used in only C sources. */
#ifdef OVERRIDE_MAX_TASK_NUM
#undef MAX_TASK_NUM
#define MAX_TASK_NUM      OVERRIDE_MAX_TASK_NUM
#endif

#ifdef NO_TASK
/* disable all tasks. */
#undef TASK_DEFAULT
#define TASK_DEFAULT      TASK_DISABLE
#endif

#endif                                     /* _ONLYC_H_ */

```

以下はプリプロセッサ定義を使用したCソースファイルの例です。

(csrc.c)

```

#include "onlyc.h"
|
int main_loop( void )
{
    while ( 1 ) {
        volatile unsigned long *ulpTask = &ulTaskManage;
        unsigned long          ulTaskBuf;
        unsigned int iTaskBit;
        int iTaskNum ;

        if ( ( *ulpTask & TASK_MASK ) == TASK_ENABLE ) {
            ulTaskBuf = *ulpTask;
            for ( iTaskNum = 0, iTaskBit = TASK_0 ;
                  iTaskNum < MAX_TASK_NUM ; iTaskNum++, iTaskBit <= 1 ) {
                if ( (ulTaskBuf & iTaskBit) != 0 ) {
                    /* Execute task if registered. */
                    if ( fnpTask[ iTaskNum ] != (void *)0x0 ) {
                        fnpTask[ iTaskNum ]();
                    }
                }
            }
        } else {
            /* illegal interruption occurred */
            break;
        }
    }

#ifdef GOSTART_IFAPPEND
    ENDLESS_LOOP:
        goto ENDLESS_LOOP
#else
    return 0;
#endif
}

void add_task( unsigned long ulTaskFlag, void *fpFunc )
{
    int iNum = 0;

    TASK_NUM( ulTaskFlag, iNum )

    fnpTask[ iNum ] = fpFunc;
    ulTaskManage = ulTaskManage | ulTaskFlag ;
}
|

```

6.10 xgccの機能と注意事項について

- **xgcc**の詳細についてはGNU C/C++コンパイラのドキュメントを参照してください。ドキュメントは、世界各地にあるGNUのミラーサイトからインターネット等を利用して入手可能です。
- 割り込みハンドラ用の関数の宣言方法については、"4 ソースファイル"を参照してください。
- `.gnu.linkonce`セクションおよび`weak`シンボルには未対応です。
- `#pragma`は動作保証対象外です。お客さまの責任で使用してください。
- gccコアのバグなどの情報は、"[%gnu33%doc%release_history_j.pdf](#)"及び、"6.11 既知の問題"を参照してください。
- gcc3.3で対応しているC99規格については、以下のサイトを参照してください。
<http://gcc.gnu.org/gcc-3.3/c99status.html>

6.11 既知の問題

以下に GNU33 C/C++ コンパイラで認識されている不具合のケースを記載します。

No.1	不具合の内容
	以下のコンパイルエラーになります。 internal compiler error: Segmentation fault Please submit a full bug report, with preprocessed source if appropriate. See <URL:http://gcc.gnu.org/bugs.html> for instructions.
	再現コード
	<pre>#include <stdio.h> int main(void) { struct data *p ; printf("OK¥n") ; return(0) ; } struct data { int i ; };</pre>
	対応方法
	①構造体の型の宣言を変数の宣言より前で行ってください。 ②あるいは、<stdio> をインクルードしてください。
	コンパイル条件
	C++ のみ。-gstabs をつけた時にエラーになります。

No.2	不具合の内容
	以下のコンパイルエラーになります。 In function `int main()': error: parse error before numeric constant
	再現コード
	<pre>#include <stdio> struct A { A(void) {} A(int a) {} }; struct B { B(void) {} B(A a) {} B(A arg1, A arg2) { printf("OK¥n"); } }; int main(void) { B b(A(), A(1)); return(0); }</pre>
	対応方法
	B b(A(), A(1)); は、struct B のメンバ関数の宣言ですが、B 型の戻り値を返す関数の宣言と誤認識されています。 以下のように、宣言すると正しく認識されます。 B b((0,A()),A(1)); 詳細、および類似のケースについては、http://gcc.gnu.org/bugs.html の "Parse errors for "simple" code."の項目を参照してください。
	コンパイル条件
	C++ のみ。

No.3	不具合の内容
	以下のコンパイルエラーになります。 internal compiler error: in pop_binding, at cp/decl.c:1428 Please submit a full bug report, with preprocessed source if appropriate. See <URL:http://gcc.gnu.org/bugs.html> for instructions.
	再現コード
	<pre>#include <stdio> struct A { class D {} ; }; struct B { typedef int D ; }; struct C : public A { struct E : public B {} ; class D {} ; }; int main(void) { printf("OK¥n"); return(0); }</pre>
	対応方法
	以下の二つの条件を満たした時に、エラーになります。 ・継承クラス(struct C)と基本クラス(struct A)の中に同じ名前のメンバクラスがある。 ・継承クラス内のメンバクラス(struct E)が継承されていて、その基本クラス(struct B)の中に同じ名前のメンバ変数/typedef がある。 http://gcc.gnu.org/ml/gcc-prs/2002-02/msg00024.html と同現象のエラーになります。 クラス名とメンバ変数/typedef 名とは、不必要な重複を避けることを推奨します。
	コンパイル条件
	C++ のみ。

No.4	不具合の内容
	if 文で比較した時点では、c を反転した値が 0 として認識されません。 else 文(②) を実行すべきところが、if 文(①) を実行してしまいます。 <pre>const signed short c = 0xffff; const signed int c = 0xffffffff; const signed long c = 0xffffffff; const signed long long c = 0xffffffffffffffffLL; でも、同様の結果になります。</pre>
	再現コード
	<pre>#include <stdio.h> int main(void) { const signed char c = 0xff; if((signed int)(~c)) { printf("NG¥n"); -- ① } else { printf("OK¥n"); -- ② } return(0); }</pre>
	対応方法
No.5	①const を外すと、正常に実行します。 ②あるいは一度、ワーク変数に代入してから if 文で比較すると OK になります。 ex) <pre>i_wk = (signed int)(~c); if(i_wk){</pre> ③また、上記の型でも、配列の場合は正常に実行します。 ex) <pre>const signed char c[2] = { 0xff,0xff }; if((signed int)(~c[0])) {</pre>
	コンパイル条件
	C++ のみ。
	不具合の内容
	以下のコンパイルエラーになります。 <pre>error: invalid use of undefined type 'class A' error: forward declaration of 'class A'</pre>
	再現コード
	<pre>class A { public: void proc_1(void) throw(A); };</pre>
	対応方法
	上記のようにメンバ関数が自らのクラスを throw 文でスローすることはできません。
	コンパイル条件
	C++ のみ

No.6	不具合の内容
	以下のコンパイルエラーになります。
	cc1.exe: out of memory allocating mmmmmmmm bytes after a total of nnnnnnnn bytes
	再現コード
	<pre>unsigned char uc_array[] = { 0x00,0x01, ... }; int main() { ※配列の大きさ -- 数十万バイト }</pre>
	対応方法
	コンパイル時にコンパイラが確保しているメモリ領域が足りなくなるエラーです。 添え字なしの配列のサイズが大きすぎる為に発生しています。 同様のエラーは、ソースコードの行数の多いファイルで発生する場合があります。 配列やソースコードを分割するなどして、一度にコンパイラが確保するメモリ領域が小さくて済むようにして下さい。
No.7	コンパイル条件
	C / C++。
	不具合の内容
	再現コードで b の値が 126 になりません。
	再現コード
	<pre>#include <stdio.h> signed int a ; int main(void) { signed char tmp ; signed int b ; a = 128 ; tmp = a - 1 ; -- ① b = tmp - 1 ; -- ② if (b != 126) { printf("NG¥n") ; } else { printf("OK¥n") ; } return(0) ; }</pre>
	対応方法
	最適化によるエラーです。最適化により、① & ②の処理がまとめてコンパイルされます。 その為、128(=0x80) の符号拡張(signed char -> signed int)と減算が一度に行われ、0xfffff80 - 2 という処理になります。その結果、-130(=0xfffff7e) という値になります。 このケースでは、signed char tmp; を volatile をつけて宣言することで回避して下さい。
	コンパイル条件
	C / C++ で最適化レベルが -O / -O2 / -O3 の時にエラーになります。

No.8	不具合の内容	
	strcmp() の結果が等しくなりません。 Kanji filter は、コンパイル時にシフトJIS コードをASCII エスケープに変換するツールでデフォルトで有効になっています。 文字列化演算子のマクロを使用した場合、コンパイル時には以下の順序で文字列が変換されていく為、等しい文字列になりません。	
	ソースコード	<pre>str("字") "¥"字¥"</pre>
	↓	
	Kanji filter による変換	<pre>str("¥x8e¥x9a") "¥"¥x8e¥x9a¥"</pre>
	↓	
	プリプロセッサ による変換	<pre>"¥"¥¥x8e¥¥x9a¥" "¥"¥x8e¥x9a¥"</pre> → Ver3.3.0 以降はこの問題は解決済みであり、エラーになることはありません。
再現コード		
<pre>#include <stdio.h> #include <string.h> #define str(a) #a // 文字列化演算子のマクロ int main(void) { if(!strcmp(str("字"), "¥"字¥")) { printf("OK¥n"); } else { printf("NG¥n"); } return(0); }</pre>		
対応方法		
Kanji filter を無効にしてください。 コマンドライン上でコンパイルする時は、メイクファイル(*.mak) 内の "CC=xgcc_filt" を "CC=xgcc" に変更してください。 IDE 上でコンパイルする時は、プロジェクトプロパティ内の Kanji filter 使用の項目を無効にしてください。		
コンパイル条件		
C / C++で Kanji filter が有効な時にエラーになります。		

7 ライブラリ

この章では本パッケージに含まれるエミュレーションライブラリ(浮動小数点演算関数および整数剰余演算関数)、long long型エミュレーションライブラリ、ANSIライブラリ、C++ライブラリについて説明します。

7.1 ライブラリ概要

本パッケージが提供するライブラリ全般についての概要について説明します。

7.1.1 ライブラリの構成

ライブラリは、以下の構成になっています。

libstdc++.a	C++ライブラリ 例外処理/実行時型情報(RTTI)などC++の機能やSTL(Standard Template Library)の機能を提供します。
libgcc2.a	long long型エミュレーションライブラリ long long型の演算関数を提供します。
libc.a	ANSIライブラリ ANSI標準の関数を提供します。
libgcc.a	エミュレーションライブラリ 単精度(32ビット)および倍精度(64ビット)浮動小数点の四則演算、比較、型変換、および整数の剰余演算を提供します。
libgccP.a	エミュレーションライブラリ(高精度) libgcc.aの高精度版です。
libstdio.a	シミュレーテッドI/O用ライブラリ ANSI Cライブラリの初期化関数(7.4.3節参照)と入出力用の下位レベル関数(7.4.4節参照)を提供します。

ライブラリは、ターゲットプロセッサの種類別に以下のフォルダにインストールされます。

¥EPSON

¥gnu33

¥lib

¥std C33 STD用フォルダ

¥pe C33 PE用フォルダ

¥33401 S1C33401用フォルダ

std、pe、33401それぞれのフォルダ内に、上記6個のライブラリが配置されます。

使用するプロセッサに合わせて選択してください。

注: 本バージョンのエミュレーションライブラリ(libgcc.a/libgccP.a)は、旧バージョンのS/T/Zデータエリアの機能には対応していません。S/T/Zエリアを使用する場合は、¥gnu33¥utility¥interrupt_mask_emulib以下のディレクトリ内にあるエミュレーションライブラリをリンクしてください。

7.1.2 ライブラリ追加時の注意事項

ライブラリには依存関係があります。

.makファイル/.ldsファイルにライブラリを記述する場合は、以下の順序で指定してください。

1. 追加ライブラリ
2. libstdio.a
3. libstdc++.a
4. libgcc2.a
5. libc.a
6. libgcc.a(またはlibgccP.a)

オブジェクトファイル(もしくはライブラリ)は、リンクに渡すファイル順で自身よりも後方にあるライブラリのみ参照可能です。追加ライブラリを最後に指定した場合、追加ライブラリ内では他のライブラリをいっさい使用することができません。floatやdouble演算、ANSIライブラリといった基本的な関数が使えませんので、追加ライブラリは必ずエミュレーションライブラリ、ANSIライブラリよりも前に配置するようにしてください。

例: 1. NG

```
ld.exe -T withmylib.lds -o withmylib.elf boot.o libc.a libgcc.a mylib.a
```

mylib.aがエミュレーションライブラリ、ANSIライブラリを使用している場合、必ずリンク時にエラーが出ます。

2. OK

```
ld.exe -T withmylib.lds -o withmylib.elf boot.o mylib.a libc.a libgcc.a
```

リンク時にエラーは発生せず、mylib.aは正常にエミュレーションライブラリ、ANSIライブラリを使用できます。

追加したライブラリ同士で依存関係を持つ場合は、基本的なライブラリを最後に配置するようにしてください。

例: lib1.a エミュレーションライブラリ、ANSIライブラリのみを呼び出す

lib2.a エミュレーションライブラリ、ANSIライブラリに加え、lib1.aを呼び出す

lib3.a エミュレーションライブラリ、ANSIライブラリに加え、lib1.a、lib2.aを呼び出す

```
ld.exe -T withmylib.lds -o withmylib.elf boot.o lib3.a lib2.a lib1.a libc.a libgcc.a
```

IDE上でのライブラリの追加方法については、"5.7.5 リンカオプションの設定"を参照してください。

7.2 エミュレーションライブラリ

7.2.1 概要

本パッケージには、IEEE形式に準拠した単精度(32ビット)および倍精度(64ビット)浮動小数点数の四則演算、比較、型変換、および整数の剰余演算をサポートするエミュレーションライブラリ(libgcc.aとlibgccP.a)が含まれています。C/C++コンパイラxgccは、浮動小数点演算、整数剰余演算が行われると、これらのライブラリ内の関数を呼び出します。ライブラリ関数は、既定の汎用レジスタを介してデータのやり取りを行いますので、アセンブリソースから呼び出すこともできます。エミュレーションライブラリ関数を使用する場合は、リンク時にlibgcc.aまたはlibgccP.aを指定してください。

libgccP.aは、libgcc.aと同一の関数で構成されています。ただし、以下の関数の演算精度がlibgcc.aのものより上がっています。

`__adddf3`, `__subdf3`, `__muldf3`, `__divdf3`

libgcc.aでは仮数部の有効範囲以下の桁は切り捨てとなりますが、libgccP.aの上記4つの関数では、有効範囲以下の桁も演算対象となり、四捨五入されて仮数部の最下位ビットに反映されます。sin、cos、tan等の数学関数で、高い演算精度が必要となる場合に使用してください。

すべてのソースコードは、¥gnu33¥utility¥lib_src¥emulib以下のディレクトリ内にあります。

ライブラリで使用しているレジスタ

- %r0から%r14を使用しています。
- %r15は未使用です。
- %r0から%r3は関数が開始時にスタックにプッシュ、終了時にポップして保護されます。

7.2.2 浮動小数点演算関数

●関数一覧

表7.2.2.1に浮動小数点演算関数を示します。

表7.2.2.1 浮動小数点演算関数一覧

分類	関数名	機能
倍精度浮動小数点演算	__adddf3	加算 $(\%r5, \%r4) \leftarrow (\%r7, \%r6) + (\%r9, \%r8)$
	__subdf3	減算 $(\%r5, \%r4) \leftarrow (\%r7, \%r6) - (\%r9, \%r8)$
	__muldf3	乗算 $(\%r5, \%r4) \leftarrow (\%r7, \%r6) * (\%r9, \%r8)$
	__divdf3	除算 $(\%r5, \%r4) \leftarrow (\%r7, \%r6) / (\%r9, \%r8)$
	__negdf2	符号反転 $(\%r5, \%r4) \leftarrow -(\%r7, \%r6)$
単精度浮動小数点演算	__addsf3	加算 $\%r4 \leftarrow \%r6 + \%r7$
	__subsf3	減算 $\%r4 \leftarrow \%r6 - \%r7$
	__mulsf3	乗算 $\%r4 \leftarrow \%r6 * \%r7$
	__divsf3	除算 $\%r4 \leftarrow \%r6 / \%r7$
	__negsf2	符号反転 $\%r4 \leftarrow -\%r6$
型変換	__fixunsdfsi	double → unsigned int $\%r4 \leftarrow (\%r7, \%r6)$
	__fixdfsi	double → int $\%r4 \leftarrow (\%r7, \%r6)$
	__floatsidf	int → double $(\%r5, \%r4) \leftarrow \%r6$
	__fixunssf3	float → unsigned int $\%r4 \leftarrow \%r6$
	__fixsf3	float → int $\%r4 \leftarrow \%r6$
	__floatsisf	int → float $\%r4 \leftarrow \%r6$
	__truncdfsf2	double → float $\%r4 \leftarrow (\%r7, \%r6)$
	__extendsfdf2	float → double $(\%r5, \%r4) \leftarrow \%r6$
浮動小数点比較	__fcmpd	double型比較 $\%psr\text{変更} \leftarrow (\%r7, \%r6) - (\%r9, \%r8)$
	__eqdf2	double型比較 (a=b) $\%psr\text{変更} \leftarrow (\%r7, \%r6) - (\%r9, \%r8), \%r4 \leftarrow 0 \mid 1 *$
	__nedf2	double型比較 (a≠b) $\%psr\text{変更} \leftarrow (\%r7, \%r6) - (\%r9, \%r8), \%r4 \leftarrow 1 \mid 0 *$
	__gtdf2	double型比較 (a>b) $\%psr\text{変更} \leftarrow (\%r7, \%r6) - (\%r9, \%r8), \%r4 \leftarrow 1 \mid 0 *$
	__gedf2	double型比較 (a≥b) $\%psr\text{変更} \leftarrow (\%r7, \%r6) - (\%r9, \%r8), \%r4 \leftarrow 0 \mid -1 *$
	__ltdf2	double型比較 (a<b) $\%psr\text{変更} \leftarrow (\%r7, \%r6) - (\%r9, \%r8), \%r4 \leftarrow -1 \mid 0 *$
	__ledf2	double型比較 (a≤b) $\%psr\text{変更} \leftarrow (\%r7, \%r6) - (\%r9, \%r8), \%r4 \leftarrow 0 \mid 1 *$
	__fcmps	float型比較 $\%psr\text{変更} \leftarrow \%r6 - \%r7$
	__eqsf2	float型比較 (a=b) $\%psr\text{変更} \leftarrow \%r6 - \%r7, \%r4 \leftarrow 0 \mid 1 *$
	__nesf2	float型比較 (a≠b) $\%psr\text{変更} \leftarrow \%r6 - \%r7, \%r4 \leftarrow 1 \mid 0 *$
	__gtsf2	float型比較 (a>b) $\%psr\text{変更} \leftarrow \%r6 - \%r7, \%r4 \leftarrow 1 \mid 0 *$
	__gesf2	float型比較 (a≥b) $\%psr\text{変更} \leftarrow \%r6 - \%r7, \%r4 \leftarrow 0 \mid -1 *$
	__ltsf2	float型比較 (a<b) $\%psr\text{変更} \leftarrow \%r6 - \%r7, \%r4 \leftarrow -1 \mid 0 *$
	__lesf2	float型比較 (a≤b) $\%psr\text{変更} \leftarrow \%r6 - \%r7, \%r4 \leftarrow 0 \mid 1 *$

* a=(%r7,%r6) or %r6、b=(%r9,%r8) or %r7、条件成立の場合%r4=左側の値、条件不成立の場合%r4=右側の値

- 演算結果がオーバーフローまたはアンダーフローした場合は、無限大または-無限大(次節参照)を返します。
- 比較関数は $op1 - op2$ の結果により、PSRのC、V、Z、Nを次のように変更します。他のフラグは変更されません。

比較結果	C	V	Z	N
$op1 > op2$	0	0	0	0
$op1 = op2$	0	0	1	0
$op1 < op2$	1	0	0	1

- 型変換時は0方向への丸めが行われます。

●浮動小数点フォーマット

C/C++コンパイラ**xgcc**はIEEE標準規格に準拠したfloat型(単精度浮動小数点、32ビット)とdouble型(倍精度浮動小数点、64ビット)をサポートしています。

浮動小数点数の内部形式を以下に示します。

倍精度浮動小数点数のフォーマット

double型の実数は次のように64ビットで構成されます。

63 62	52 51	0
[S]	指数部	仮数部

ビット63: 符号ビット(1ビット)

ビット62～52: 指数部(11ビット)

ビット51～0: 仮数部(52ビット)

レジスタに格納する場合は2つのレジスタを占有します。たとえば、浮動小数点演算の結果は次のように%r5および%r4レジスタに格納されます。

%r5レジスタ: 符号ビット、指数部および仮数部の上位20ビット(51:32)

%r4レジスタ: 仮数部の下位32ビット(31:0)

double型の有効範囲、浮動小数点表記と内部データの対応は次のとおりです。

+0:	0.0e+0	0x00000000 00000000
-0:	-0.0e+0	0x80000000 00000000
最大正規化数:	1.79769e+308	0x7fefffff ffffffff
最小正規化数:	2.22507e-308	0x00100000 00000000
最大非正規化数:	2.22507e-308	0x000fffff ffffffff
最小非正規化数:	4.94065e-324	0x00000000 00000001
無限大:		0x7ff00000 00000000
-無限大:		0xfff00000 00000000

0x7ff00000 00000001～0x7fffffff ffffffffおよび0xfff00000 00000001～0xffffffff ffffffffは数値としては認められません。

単精度浮動小数点数のフォーマット

float型の実数は次のように32ビットで構成されます。

31 30	23 22	0
[S]	指数部	仮数部

ビット31: 符号ビット(1ビット)

ビット30～23: 指数部(8ビット)

ビット22～0: 仮数部(23ビット)

float型データは1つのレジスタに格納できます。

float型の有効範囲、浮動小数点表記と内部データの対応は次のとおりです。

+0:	0.0e+0f	0x00000000
-0:	-0.0e+0f	0x80000000
最大正規化数:	3.40282e+38f	0x7f7fffff
最小正規化数:	1.17549e-38f	0x00800000
最大非正規化数:	1.17549e-38f	0x007fffff
最小非正規化数:	1.40129e-45f	0x00000001
無限大:		0x7f800000
-無限大:		0xff800000

0x7f800001～0x7fffffffおよび0xff800001～0xffffffffは数値としては認められません。

注意

無限大の扱い等を含め、IEEE準拠のFPUとは精度および機能的な差異があります。

7.2.3 整数剰余演算関数

表7.2.3.1に整数剰余演算関数を示します。

表7.2.3.1 整数剰余演算関数一覧

分類	関数名	機能	
整数除算	divsi3	符号付き整数除算	%r4 ← %r6 / %r7
	udivsi3	符号なし整数除算	%r4 ← %r6 / %r7
剰余演算	modsi3	符号付き剰余演算	%r4 ← %r6 % %r7
	umodsi3	符号なし剰余演算	%r4 ← %r6 % %r7

これらの関数は入力レジスタの値をチェックしません。そのため、%r7レジスタに0が設定されている場合、関数内のdiv0sまたはdiv0u命令でゼロ除算例外が発生します。

7.3 long long型エミュレーションライブラリ

7.3.1 概要

本パッケージには、long long型の演算をサポートするlong long型エミュレーションライブラリ (libgcc2.a)が含まれています。

C/C++コンパイラxgccは、long long型の演算が行われると、libgcc2.a内の関数を呼び出します。ライブラリ関数は、既定の汎用レジスタを介してデータのやり取りを行いますので、アセンブリソースから呼び出すこともできます。long long型エミュレーションライブラリ関数を使用する場合は、リンク時にlibgcc2.aを指定してください。

このライブラリは、GNU C/C++コンパイラver.3.3.2が提供するlibgcc2.cがベースになっています。すべてのソースコードは¥gnu33¥utility¥lib_src¥emulib2以下のディレクトリ内にあります。

ライブラリで使用しているレジスタ

- %r0から%r14を使用しています。
- %r15は未使用です。
- %r0から%r3は関数が開始時にスタックにプッシュ、終了時にポップして保護されます。

7.3.2 long long型演算関数

表7.3.2.1にlong long型演算関数を示します。

表7.3.2.1 long long型演算関数一覧

分類	関数名	機能
演算	__muldi3	乗算 (%r5,%r4) ← (%r7,%r6) * (%r9,%r8)
	__divdi3	符号付き除算 (%r5,%r4) ← (%r7,%r6) / (%r9,%r8)
	__udivdi3	符号なし除算 (%r5,%r4) ← (%r7,%r6) / (%r9,%r8)
	__moddi3	符号付き剰余演算 (%r5,%r4) ← (%r7,%r6) % (%r9,%r8)
	__umoddi3	符号なし剰余演算 (%r5,%r4) ← (%r7,%r6) % (%r9,%r8)
	__negdi2	符号反転 (%r5,%r4) ← (%r7,%r6)
	__lshrdi3	論理右シフト (%r5,%r4) ← (%r7,%r6) >> (%r9,%r8)
	__ashrdi3	算術右シフト (%r5,%r4) ← (%r7,%r6) >> (%r9,%r8)
	__ashldi3	算術左シフト (%r5,%r4) ← (%r7,%r6) << (%r9,%r8)
型変換	__fixunsdfdi	double → unsigned long long (%r5,%r4) ← (%r7,%r6)
	__fixdfdi	double → long long (%r5,%r4) ← (%r7,%r6)
	__floatdidf	long long → double (%r5,%r4) ← (%r7,%r6)
	__fixunssfdi	float → unsigned long long (%r5,%r4) ← %r6
	__fixsfdi	float → long long (%r5,%r4) ← %r6
	__floatdisf	long long → float %r4 ← (%r7,%r6)
比較	__cmpdi2	long long型比較 %psr変更 ← (%r7,%r6) - (%r9,%r8), %r4 ← 2 1 0 *1
	__ucmpdi2	unsigned long long型比較 %psr変更 ← (%r7,%r6) - (%r9,%r8), %r4 ← 2 1 0 *1
その他	__ffsdi2	ビットスキャン (%r5,%r4) ← (%r7,%r6) *2

*1 比較関数はopt1 - opt2の結果により、PSRのC、V、Z、Nおよび%r4(戻り値)を次のように変更します。他のフラグは変更されません。

比較結果	C	V	Z	N	%r4
opt1 > opt2	0	0	0	0	2
opt1 = opt2	0	0	1	0	1
opt1 < opt2	1	0	0	1	0

*2 LSBビットから1のビットをスキャンして、最初に見つかった1のビットの位置を返します。

最初の1のビットがLSBのとき: 1
最初の1のビットがMSBのとき: 64
1のビットが見つからなかったとき: 0

7.4 ANSIライブラリ

7.4.1 概要

本パッケージには、ANSIライブラリが含まれています。
各関数はANSI標準の機能を持っています。ただし、組み込み型マイコンの仕様上、ファイル関連の関数の中にはダミーとなっているものもあります。
ANSIライブラリファイルはlibc.aで、C33コアの種類別にlib¥std、lib¥33401、lib¥peディレクトリにインストールされます。また、long long型のANSIライブラリがlibgcc2.aに含まれています。各関数に関する定義が記録されている以下のヘッダファイルが、includeディレクトリにインストールされます。
stdio.h stdlib.h time.h math.h errno.h float.h limits.h ctype.h
string.h stdarg.h setjmp.h smcvals.h stddef.h assert.h locale.h
また、上記のヘッダファイルに対応する以下のC++ヘッダが¥includeディレクトリにインストールされます。C++ヘッダは名前空間stdに対応しています。
cstdio cstdlib ctime cmath cerrno cfloat climits cctype
cstring cstdarg csetjmp cstddef cassert clocale
ライブラリの使用方法については、¥gnu33¥utility¥sample_std¥ansilibディレクトリ内のサンプルファイルを参照してください。また、すべてのソースコードは¥gnu33¥utility¥lib_src¥ansilib(libc.a)/¥gnu33¥utility¥lib_src¥emulib2(libgcc2.a)以下のディレクトリ内にあります。詳細はソースを参照し、必要に応じて修正してください。

ライブラリで使用しているレジスタ

- %r0から%r11を使用しています。
- %r12から%r15は未使用です。
- %r0から%r3は関数が開始時にスタックにプッシュ、終了時にポップして保護されます。

7.4.2 ANSIライブラリ関数一覧

一覧表のリエントラント欄に記載されている記号の意味は以下のとおりです。
○ リエントラントな関数
× ノンリエントラントな関数
ー ダミー関数(お客さまのシステムに対応させた修正が必要な関数です。)
△ ノンリエントラントな関数(グローバル変数を参照しているもの、ダミー関数をコールしているものです。グローバル変数の変更がなく、お客さまが記述するread()およびwrite()がリエントラントな関数である場合、リエントラントな関数として使えます。)

●入出力関数

以下、libc.aに含まれる入出力関数の一覧を示します。

表7.4.2.1 入出力関数一覧

ヘッダファイル: stdio.h/cstdio

関数	機能	リエントラント	備考
FILE *fopen(const char *filename, const char *mode);	ダミー	ー	
FILE *freopen(const char *filename, const char *mode, FILE *stream);	ダミー	ー	
int fclose(FILE *stream);	ダミー	ー	
int fflush(FILE *stream);	ダミー	ー	
int fseek(FILE *stream, long offset, int origin);	ダミー	ー	
long ftell(FILE *stream);	ダミー	ー	
void rewind(FILE *stream);	ダミー	ー	
int fgetpos(FILE *stream, fpos_t *ptr);	ダミー	ー	
int fsetpos(FILE *stream, const fpos_t *pos);	ダミー	ー	
size_t fread(void *ptr, size_t size, size_t count, FILE *stream);	stdinから配列要素を入力	△	グローバル変数stdin, _iobを参照、read関数をコール
size_t fwrite(const void *ptr, size_t size, size_t count, FILE *stream);	stdoutから配列要素を出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール

関数	機能	リエントラント	備考
int fgetc (FILE *stream);	stdinから1文字入力	△	グローバル変数stdin, _iobを参照、read関数をコール
int getc (FILE *stream);	stdinから1文字入力	△	グローバル変数stdin, _iobを参照、read関数をコール
int getchar (void);	stdinから1文字入力	△	グローバル変数stdin, _iobを参照、read関数をコール
int ungetc (int c, FILE *stream);	入力バッファに1文字押し戻す	×	グローバル変数stdin, stdout, stderr, _iobを参照、戻した文字が上書きされる
char * fgets (char *s, int n, FILE *stream);	stdinから文字列を入力	△	グローバル変数stdin, _iobを参照、read関数をコール
char * gets (char *s);	stdinから文字列を入力	△	グローバル変数stdin, _iobを参照、read関数をコール
int fputc (int c, FILE *stream);	stdoutに1文字出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int putc (int c, FILE *stream);	stdoutに1文字出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int putchar (int c);	stdoutに1文字出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int fputs (char *s, FILE *stream);	stdoutへ文字列を出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int puts (char *s);	stdoutへ文字列を出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int remove (const char *filename);	ダミー	—	
int rename (const char *oldname, const char *newname);	ダミー	—	
void setbuf (FILE *stream, char *buf);	ダミー	—	
int setvbuf (FILE *stream, char *buf, int type, size_t size);	ダミー	—	
FILE * tmpfile (void);	ダミー	—	
char * tmpnam (char *buf);	ダミー	—	
int feof (FILE *stream);	ダミー	—	
int ferror (FILE *stream);	ダミー	—	
void clearerr (FILE *stream);	ダミー	—	グローバル変数stdout, _iobを参照、
void perror (const char *s);	stdoutへのエラー情報出力	×	errnoを変更、read関数をコール グローバル変数stdout, _iobを参照、
int fscanf (FILE *stream, const char *format, ...);	stdinからの書式指定付き入力	×	errnoを変更、read関数をコール グローバル変数stdout, _iobを参照、
int scanf (const char *format, ...);	stdinからの書式指定付き入力	×	errnoを変更、read関数をコール グローバル変数errnoを変更
int sscanf (const char *s, const char *format, ...);	文字列からの書式指定付き入力	×	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int fprintf (FILE *stream, const char *format, ...);	stdoutへの書式指定付き出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int printf (const char *format, ...);	stdoutへの書式指定付き出力	△	参照、write関数をコール
int sprintf (char *s, const char *format, ...);	配列への書式指定付き出力	○	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int vfprintf (FILE *stream, const char *format, va_list arg);	stdoutへの変換出力	△	グローバル変数stdout, stderr, _iobを参照、write関数をコール
int vprintf (const char *format, va_list arg);	stdoutへの変換出力	△	参照、write関数をコール
int vsprintf (char *s, const char *format, va_list arg);	配列への変換出力	○	

注: ファイルシステムは扱えません。stdin、stdoutのみです。stdinはread、stdoutはwrite関数が必要です。詳細は7.4.4節を参照してください。

●ユーティリティ関数

以下、libc.a、libgcc2.aに含まれるユーティリティ関数の一覧を示します。

表7.4.2.2 ユーティリティ関数一覧(libc.a)

ヘッダファイル: **stdlib.h/cstdlib**

関数	機能	リエントラント	備考
void * malloc (size_t size);	領域の確保	×	グローバル変数errno, ansi_ucStartAlloc, ansi_ucEndAlloc, ansi_ucNxtAlcP, ansi_ucTblPtr, ansi_ulRowを変更
void * calloc (size_t elt_count, size_t elt_size);	配列領域の確保	×	メモリアロケート中から呼び出しのため不正
void free (void *ptr);	領域の解放	×	メモリアロケート中から呼び出しのため不正
void * realloc (void *ptr, size_t size);	領域サイズの変更	×	メモリアロケート中から呼び出しのため不正
int system (const char *command);	ダミー	—	
void exit (int status);	プログラムの正常終了	○	exitを参照 後から呼び出された側で終了
void abort (void);	プログラムの異常終了	○	exitを参照 後から呼び出された側で終了
int atexit (void (*func)(void));	ダミー	—	
char * getenv (const char *str);	ダミー	—	
void * bsearch (const void *key, const void *base, size_t count, size_t size, int (*compare)(const void *, const void *));	バイナリサーチ	○	
void qsort (void *base, size_t count, size_t size, int (*compare)(const void *, const void *));	クイックソート	○	
int abs (int x);	絶対値を返す (int型)	○	
long labs (long x);	絶対値を返す (long型)	○	
div_t div (int n, int d);	int型除算	×	グローバル変数errnoを変更
ldiv_t ldiv (long n, long d);	long型除算	×	グローバル変数errnoを変更
int rand (void);	擬似乱数を返す	×	グローバル変数errnoを変更
void srand (unsigned int seed);	擬似乱数種の設定	×	グローバル変数errnoを変更
long atol (const char *str);	文字列をlong型に変換	×	グローバル変数errnoを変更
int atoi (const char *str);	文字列をint型に変換	×	グローバル変数errnoを変更
double atof (const char *str);	文字列をdouble型に変換	×	グローバル変数errnoを変更
double strtod (const char *str, char **ptr);	文字列をdouble型に変換	×	グローバル変数errnoを変更
long strtol (const char *str, char **ptr, int base);	文字列をlong型に変換	×	グローバル変数errnoを変更
unsigned long strtoul (const char *str, char **ptr, int base);	文字列をunsigned long型に変換	×	グローバル変数errnoを変更

表7.4.2.3 ユーティリティ関数一覧(libgcc2.a)

ヘッダファイル: **stdlib.h/cstdlib**

関数	機能	リエントラント	備考
long long strtoll (const char *str, char **endptr, int base);	文字列をlong long型に変換	×	グローバル変数errnoを変更
unsigned long long strtoull (const char *str, char **endptr, int base);	文字列をunsigned long long型に変換	×	グローバル変数errnoを変更

●非局所分岐関数

以下、libc.aに含まれる非局所分岐関数の一覧を示します。

表7.4.2.4 非局所分岐関数一覧

ヘッダファイル: **setjmp.h/csetjmp**

関数	機能	リエントラント	備考
int setjmp (jmp_buf env);	非局所分岐	○	
void longjmp (jmp_buf env, int status);	非局所分岐	○	

●日付と時刻関数

以下、libc.aに含まれる日付と時刻関数の一覧を示します。

表7.4.2.5 日付と時刻関数一覧

ヘッダファイル: **time.h/ctime**

関数	機能	リエントラント	備考
<code>clock_t clock(void);</code>	ダミー	—	
<code>char *asctime(const struct tm *ts);</code>	ダミー	—	
<code>char *ctime(const time_t *timeptr);</code>	ダミー	—	
<code>double difftime(time_t ti, time_t t2);</code>	ダミー	—	
<code>struct tm *gmtime(const time_t *t);</code>	カレンダー時間を標準時間に変換	×	スタティック変数を変更
<code>struct tm *localtime(const time_t *t);</code>	ダミー	—	
<code>time_t mktime(struct tm *tmptr);</code>	標準時間をカレンダー時間に変換	×	スタティック変数を変更。ロケール情報と夏時間の設定は反映されません。
<code>time_t time(time_t *tpttr);</code>	現在のカレンダー時間を返す	△	グローバル変数gm_secを参照

●数学関数

以下、libc.aに含まれる数学関数の一覧を示します。

表7.4.2.6 数学関数一覧

ヘッダファイル: **math.h, errno.h, float.h, limits.h/cmath, cerrno, cfloat, climits**

関数	機能	リエントラント	備考
<code>double fabs(double x);</code>	絶対値を返す (double型)	○	
<code>double ceil(double x);</code>	double型小数部の切り上げ	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double floor(double x);</code>	double型小数部の切り捨て	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double fmod(double x, double y);</code>	double型の剰余計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double exp(double x);</code>	指数計算(e ^x)	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double log(double x);</code>	自然対数の計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double log10(double x);</code>	常用対数の計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double frexp(double x, int *nptr);</code>	浮動小数点数の仮数と指数を返す	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double ldexp(double x, int n);</code>	仮数と指数から浮動小数点数を返す	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double modf(double x, double *nptr);</code>	浮動小数点数の整数部と少数部を返す	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double pow(double x, double y);</code>	x ^y を計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double sqrt(double x);</code>	平方根を計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double sin(double x);</code>	正弦(サイン)計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double cos(double x);</code>	余弦(コサイン)計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double tan(double x);</code>	正接(タンジェント)計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double asin(double x);</code>	逆正弦(アークサイン)計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double acos(double x);</code>	逆余弦(アークコサイン)計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double atan(double x);</code>	逆正接(アークタンジェント)計算	×	ALR, AHRを破壊
<code>double atan2(double y, double x);</code>	y/xの逆正接(アークタンジェント)計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double sinh(double x);</code>	双曲線正弦(ハイパボリックサイン)計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double cosh(double x);</code>	双曲線余弦(ハイパボリックコサイン)計算	×	グローバル変数errnoを変更、ALR, AHRを破壊
<code>double tanh(double x);</code>	双曲線正接(ハイパボリックタンジェント)計算	×	ALR, AHRを破壊

●文字関数

以下、libc.aに含まれる文字関数の一覧を示します。

表7.4.2.7 文字関数一覧

ヘッダファイル: **string.h/cstring**

関数	機能	リエントラント	備考
void *memchr(const void *s, int c, size_t n);	記憶域内の指定文字の位置を返す	○	
int memcmp(const void *s1, const void *s2, size_t n);	記憶域の比較	○	
void *memcpy(void *s1, const void *s2, size_t n);	記憶域のコピー	○	
void *memmove(void *s1, const void *s2, size_t n);	記憶域のコピー(オーバーラップ可能)	○	
void *memset(void *s, int c, size_t n);	記憶域に文字を設定	○	
char *strcat(char *s1, const char *s2);	文字列の連結	○	
char *strchr(const char *s, int c);	文字列内で最初に現れた指定文字の位置を返す	○	
int strcmp(const char *s1, const char *s2);	文字列の比較	○	
char *strcpy(char *s1, const char *s2);	文字列のコピー	○	
size_t strcspn(const char *s1, const char *s2);	文字列の先頭から指定文字(複数候補)が現れるまでの文字数を返す	○	
char *strerror(int code);	エラーメッセージの文字列を返す	○	
size_t strlen(const char *s);	文字列の長さを返す	○	
char *strncat(char *s1, const char *s2, size_t n);	文字列の連結(文字数指定付き)	○	
int strncmp(const char *s1, const char *s2, size_t n);	文字列の比較(文字数指定付き)	○	
char *strncpy(char *s1, const char *s2, size_t n);	文字列のコピー(文字数指定付き)	○	
char *strpbrk(const char *s1, const char *s2);	文字列内で最初に現れた指定文字(複数候補)の位置を返す	○	
char *strrchr(const char *str, int c);	文字列内で最後に現れた指定文字の位置を返す	○	
size_t strspn(const char *s1, const char *s2);	文字列の先頭から指定文字(複数候補)以外が現れるまでの文字数を返す	○	
char *strstr(const char *s1, const char *s2);	指定文字列が最初に現れた位置を返す	○	
char *strtok(char *s1, const char *s2);	文字列をトークンに分割	×	スタティック変数を変更

※ strerrorを除き、アセンブリソースで作成、チューニングされています。

●文字種判定/変換関数

以下、libc.aに含まれる文字種判定/変換関数の一覧を示します。

表7.4.2.8 文字種判定/変換関数一覧

ヘッダファイル: **ctype.h/cctype**

関数	機能	リエントラント
int isalnum(int c);	文字種判定(10進数またはアルファベット)	○
int isalpha(int c);	文字種判定(アルファベット)	○
int iscntrl(int c);	文字種判定(制御文字)	○
int isdigit(int c);	文字種判定(10進数)	○
int isgraph(int c);	文字種判定(図形文字)	○
int islower(int c);	文字種判定(アルファベット小文字)	○
int isprint(int c);	文字種判定(印刷可能文字)	○
int ispunct(int c);	文字種判定(区切り文字)	○
int isspace(int c);	文字種判定(空白文字)	○
int isupper(int c);	文字種判定(アルファベット大文字)	○
int isxdigit(int c);	文字種判定(16進数)	○
int tolower(int c);	文字種変換(アルファベット大文字 → 小文字)	○
int toupper(int c);	文字種変換(アルファベット小文字 → 大文字)	○

●可変引数マクロ

以下、stdarg.hに定義された可変引数マクロの一覧を示します。

表7.4.2.9 可変引数マクロ一覧

ヘッダファイル: **stdarg.h/cstdarg**

マクロ	機能
<code>void va_start(va_list ap, type lastarg);</code>	可変個引数集の初期化
<code>type va_arg(va_list ap, type);</code>	実引数の値を返す
<code>void va_end(va_list ap);</code>	可変個引数関数からの正常復帰

●診断マクロ

以下、libc.aに含まれる診断マクロを示します。

表7.4.2.10 診断マクロ

ヘッダファイル: **assert.h/cassert**

マクロ	機能	リエントラント	備考
<code>void assert(int test);</code>	条件式が偽のとき、診断メッセージを出力して処理を中止する	△	printf/abort関数をコール

●ロケール関数

以下、libc.aに含まれるロケール関数を示します。

表7.4.2.11 ロケール関数

ヘッダファイル: **locale.h/clocale**

関数	機能	リエントラント	備考
<code>char *setlocale(int category, const char *locale);</code>	ダミー	－	
<code>struct lconv *localeconv(void);</code>	ダミー	－	

7.4.3 グローバル変数の宣言と初期化

ANSIライブラリ関数は表7.4.3.1に示すグローバル変数を参照します。これらの変数はlibstdio.aライブラリ内に定義されています。C/C++ソースプログラム内に`¥include¥libstdio.h`をインクルードした上で、ライブラリ関数を呼び出す前に`_init_lib()`関数を呼び出してこれらの変数を初期化してください。

表7.4.3.1 宣言が必要なグローバル変数

グローバル変数	初期設定	関連するヘッダファイル/関数
<code>FILE _iob[FOPEN_MAX + 1];</code> <code>FOPEN_MAX=3, stdio.hにて定義</code> 標準入出力ストリーム用 ファイル構造体データ	<code>_iob[N]._flg = _UGETN;</code> <code>_iob[N]._buf = 0;</code> <code>_iob[N]._fd = N;</code> ($N=0\sim2$) <code>_iob[0]: stdin入力データ</code> <code>_iob[1]: stdout出力データ</code> <code>_iob[2]: stderr出力データ</code>	<code>stdio.h, smcvals.h</code> <code>fgets, fread, fscanf, getc, getchar, gets, scanf,</code> <code>ungetc, perror, fprintf, fputs, fwrite, printf,</code> <code>putc, putchar, puts, vfprintf, vprintf</code>
<code>FILE *stdin;</code> 標準入力ファイル構造体 データ <code>_iob[0]</code> へのポインタ	<code>stdin = &_iob[0];</code>	<code>stdio.h</code> <code>fgets, fread, fscanf, getc, getchar, gets, scanf,</code> <code>ungetc</code>
<code>FILE *stdout;</code> 標準出力ファイル構造体 データ <code>_iob[1]</code> へのポインタ	<code>stdout = &_iob[1];</code>	<code>stdio.h</code> <code>fprintf, fputs, fwrite, printf, putc, putchar,</code> <code>puts, vfprintf, vprintf</code>
<code>FILE *stderr;</code> 標準入出力ファイル構造体 データ <code>_iob[2]</code> へのポインタ	<code>stderr = &_iob[2];</code>	<code>stdio.h</code> <code>fprintf, fputs, fwrite, printf, perror, putc,</code> <code>putchar, puts, vfprintf, vprintf</code>
<code>int errno;</code> エラー番号を格納する変数	<code>errno = 0;</code>	<code>errno.h</code> <code>fopen, freopen, fseek, fsetpos, perror, remove,</code> <code>rename, tmpfile, tmpnam, fprintf, printf, sprintf,</code> <code>vprintf, vfprintf, fscanf, scanf, sscanf</code> <code>atof, atoi, calloc, div, ldiv, malloc, realloc, str-</code> <code>tod, strtol, strtoul</code> <code>acos, asin, atan2, ceil, cos, cosh, exp, fabs, floor,</code> <code>fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh,</code> <code>sqrt, tan</code>
<code>unsigned int seed;</code> 乱数の種を格納する変数	<code>seed = 1;</code>	<code>stdlib.h</code> <code>rand, srand</code>
<code>time_t gm_sec;</code> タイマー関数の70年1月1日 0時0分0秒からの秒数	<code>gm_sec = -1;</code>	<code>time.h</code> <code>time</code>

ANSIライブラリ関数より参照されるグローバル変数のうち、`malloc`、`calloc`、`realloc`、`free`の各関数で使用するグローバル変数は次の初期化関数をコールすることにより初期化されます。この関数は`stdlib.h`で宣言されています。

```
int ansi_InitMalloc(unsigned long START_ADDRESS, unsigned long END_ADDRESS);
```

`START_ADDRESS`には使用するメモリの先頭アドレスを、`END_ADDRESS`には終了アドレスを設定します。これらのアドレスは関数内で4バイト境界に調整されます。

この関数により、以下のグローバル変数が初期化されます。これらの変数は`stdlib.h`で宣言されています。

```
unsigned char *ansi_ucStartAlloc; ヒープ領域の先頭アドレスを指すポインタ
unsigned char *ansi_ucEndAlloc;   ヒープ領域の終端アドレスを指すポインタ
unsigned char *ansi_ucNxtAlcP;    次に割り当てられる新規領域の先頭を指すアドレスポインタ
unsigned char *ansi_ucTblPtr;     次に割り当てられる管理領域の先頭を指すアドレスポインタ
unsigned long ansi_ulRow;         次に割り当てられる管理領域を指す行ポインタ
```

ヒープ領域を確保すると、終端アドレス(`ansi_ucEndAlloc`)から始まって、先頭方向に8バイトのヒープ領域の管理データが配置されていきます。したがって、`ansi_InitMalloc()`関数によりヒープ領域として指定したエリアは、スタックポインタなどによりデータが書き換えられないように注意してください。

※ `ansi_InitMalloc()`関数は`libstdio.a`ライブラリに含まれていません。`malloc()`などをコールする前にユーザーチンからコールする必要がありますので注意してください。

(`ansi_InitMalloc()`がコールされないと、ヒープ領域が確保されません。)

7.4.4 下位レベル関数

以下に示す3つの関数(read、write、_exit)はライブラリ関数が呼び出す下位レベル関数です。readおよびwrite関数についてはlibstdio.aライブラリ内に定義されています。これを使用するには、C/C++ソースプログラム内に#include<libstdio.h>をインクルードした上で、_init_sys()関数を呼び出してください。

なお、_exit関数についてはユーザプログラム上で定義してください。

●read関数

read関数の内容

形式: `int read(int fd, char *buf, int nbytes);`

引数: `int fd;` 入力を示すファイル記述子
ライブラリ関数から呼ばれる場合、0(stdin)が渡されます。

`char *buf;` 入力データを格納するバッファへのポインタ
`int nbytes;` 転送バイト数

機能: ユーザが定義する入力用バッファからデータを最大nbytes分読み込み、bufで示されるバッファに格納します。

戻り値: 実際に入力用バッファから読み込んだデータのバイト数
入力用バッファが空(EOF)またはnbytesが0のときは、0を返します。
エラー発生時は、-1を返します。

read関数を呼び出すライブラリ関数:

直接コール: `fread, getc, _doscan`(`_doscan`はscanf系内部関数)

間接コール: `fgetc, fgets, getchar, gets`(`getc`をコール)
`scanf, fscanf, sscanf`(`_doscan`をコール)

入力用バッファの定義

形式: `unsigned char READ_BUF[65];` (変数名は任意、サイズは65バイト固定)
`unsigned char READ_EOF;`

説明: バッファの先頭(`READ_BUF[0]`)には、入力されたデータのサイズ(1~Max. 64)が格納されます。0はEOFを示します。

入力データは`READ_BUF[1]`以降に格納されます。

`READ_EOF`はすでにEOFになったかどうかを示すステータスを格納します。

シミュレーテッドI/Oを利用する場合の注意

デバッガのシミュレーテッドI/O機能(詳細はデバッガの章を参照)を利用する場合、read関数内にデバッガが入力用バッファを更新するためのグローバルなラベル`READ_FLASH`を定義し、その位置で入力用バッファに新たなデータが読み込まれるように関数を作成してください。

read関数はlibstdio.aライブラリに含まれています。これを使用するにはlibstdio.aをリンクし、ユーザプログラムのブートルーチンで_init_sys()を呼び出してください。

●write関数

write関数の内容

形式: `int write(int fd, char *buf, int nbytes);`

引数: `int fd;` 出力を示すファイル記述子
 ライブラリ関数から呼ばれる場合、1(stdout)または2(stderr)が渡されます。

`char *buf;` 出力データを格納するバッファへのポインタ

`int nbytes;` 転送バイト数

機能: `buf`で示されるバッファに格納されたデータを、ユーザが定義する出力用バッファに
`nbytes`分書き込みます。

戻り値: 実際に出力用バッファに書き込んだデータのバイト数
 書き込みが正常に行われたときは`nbytes`を返します。
 書き込みエラー発生時は、`nbytes`以外の数値を返します。

write関数を呼び出すライブラリ関数:

直接コール: `fwrite`, `putc`, `_doprint`(`_doprint`は`printf`系内部関数)

間接コール: `fputc`, `fputs`, `putchar`, `puts`(`putc`をコール)
`printf`, `fprintf`, `sprintf`, `vprintf`, `vfprintf`(`_doprint`をコール)
`perror`(`fprintf`をコール)

出力用バッファの定義

形式: `unsigned char WRITE_BUF[65];` (変数名は任意、サイズは65バイト固定)

説明: バッファの先頭(`WRITE_BUF[0]`)には、出力するデータのサイズ(1~Max. 64)を格納し
 ます。0はEOFを示します。
 出力データは`WRITE_BUF[1]`以降に格納します。

シミュレーテッドI/Oを利用する場合の注意

デバッガのシミュレーテッドI/O機能(詳細はデバッガの章を参照)を利用する場合、`write`関数
 内にデバッガが出力用バッファを更新するためのグローバルなラベル`WRITE_FLASH`を定義し、
 その位置で出力用バッファ内のデータが出力されるように関数を作成してください。

`write`関数は`libstdio.a`ライブラリに含まれています。これを使用するには`libstdio.a`をリ
 ンクし、ユーザプログラムのブートルーチンで`_init_sys()`を呼び出してください。

●_exit関数

_exit関数の内容

形式: `void _exit(void);`

機能: プログラムの終了処理を行います。

引数/戻り値:
 なし

`_exit`関数を呼び出すライブラリ関数:
 直接コール: `abort`, `exit`

7.5 C++ライブラリ

7.5.1 概要

本パッケージには、C++ライブラリ(libstdc++.a)が含まれています。
このライブラリは以下の機能を提供しています。

- グローバルクラスのコンストラクタ/デストラクタ — `_init()`/`_fini()`
- 例外処理
- 実行時型情報(RTTI)
- `string`型
- メモリの割り当てと開放 — `new/delete`
- ストリーム
ファイルストリームはダミーの機能になります。
標準ストリームを使用する場合、`cin`には`read`関数、`cout/cerr/clog`には`write`関数が必要です。
- ロケール
ロケール名はデフォルトの"C"/"POSIX"ロケールのみに対応しています。
ロケール名の切り替えには対応していません。
- STL(Standard Template Library)
STLの正確な範囲については、正式には定義されていません。
このマニュアル上では、以下のものをSTLとして分類しています。
 - コンテナ(`deque/list/map/multimap/multiset/priority_queue/queue/set/stack/vector`)
 - アルゴリズム
 - 反復子
 - アロケータ
 - 関数オブジェクト
- その他のクラス — `bitset/complex/valarray`

本パッケージのC++ライブラリでは、以下の機能は`wchar_t`型に対応していません。

- `string`型
`wstring/char_traits`クラス
- ストリーム
`wcin/wcerr/wclog`
`wios/wstreambuf/wistream/wostream/wiostream/wstringbuf/wstringstream/
wostringstream/wstringstream/wfilebuf/wifstream/wofstream/wfstream/wstreampos`
- ロケール
`money_punct`クラス/`money_punct_byname`クラス/`money_get`クラス/`money_put`クラス/
`num_punct`クラス/`num_punct_byname`クラス/`num_get`クラス/`num_put`クラス/
`__time_punct`クラス/`time_put`クラス/`time_put_byname`クラス/`time_get`クラス/
`time_get_byname`クラス/`messages`クラス/`messages_byname`クラス/`ctype`クラス/
`codecvt`クラス/`ctype_byname`クラス/`codecvt_byname`クラス/`collate`クラス/
`collate_byname`クラス

それぞれの機能を使用するために必要なヘッダについては、次の"表7.5.1.1 C++ライブラリ用標準ヘッダ一覧"を参照してください。

C++/STLの機能の具体的な使用方法については、C++/STLを解説している一般の書籍を参照してください。
また、`¥gnu33¥utility¥sample_c++`ディレクトリ以下にC++ライブラリを使用したサンプルがありますので、あわせて参照してください。

表7.5.1.1 C++ライブラリ用標準ヘッダー一覧

ヘッダ名	内容
algorithm	C++プログラムに適用できるアルゴリズムのテンプレートを定義しています。
bitset	bitsetコンテナのテンプレートを定義しています。
complex	複素数を扱うテンプレートを定義しています。
deque	dequeコンテナのテンプレートを定義しています。
exception	例外処理に関する型と関数を定義しています。
fstream	ファイルストリームクラスを定義しています。 ただし本パッケージでは、ファイルストリームはダミーの機能になります。
functional	関数オブジェクトを生成するためのテンプレートを定義しています。
io manip	入出力マニピュレータを定義しています。
ios	入出力ストリームクラスを定義しています。
iosfwd	入出力ストリームクラスの前参照を宣言しています。
iostream	標準ストリームを定義しています。
istream	入力ストリームクラスを定義しています。
iterator	反復子を定義しています。
limits	numeric_limitsクラスを定義しています。
list	listコンテナのテンプレートを定義しています。
locale	ロケールを定義しています。 ただし本パッケージでは、ロケール名はデフォルトの"C"/"POSIX"のみに対応しています。
map	mapおよびmultimapコンテナのテンプレートを定義しています。
memory	オブジェクトの割り当て、所有権の管理、解放を行うテンプレートを定義しています。
new	メモリ割り当てと開放の関数を宣言しています。
numeric	数値演算用のテンプレートを定義しています。
ostream	出力ストリームクラスを定義しています。
queue	queueおよびpriority_queueコンテナのテンプレートを定義しています。
set	setおよびmultisetコンテナのテンプレートを定義しています。
sstream	入出力文字列ストリームクラスを定義しています。
stack	stackコンテナのテンプレートを定義しています。
stdexcept	標準例外処理に関する型と関数を定義しています。
streambuf	ストリームバッファクラスを定義しています。
stringstream	文字列シーケンス入出力クラスを定義しています。
string	basic_stringクラスを定義しています。
typeinfo	実行時型情報(RTTI)を扱うクラスを定義しています。
utility	比較演算子を定義しています。
valarray	数値を扱う配列のテンプレートを定義しています。
vector	vectorコンテナのテンプレートを定義しています。

このライブラリは、GNU C/C++コンパイラver.3.3.2が提供するlibstdc++-v3ディレクトリ以下のソースコードが主にベースになっています。

すべてのソースコードは、¥gnu33¥utility¥lib_src¥libstdc++¥srcディレクトリ内にあります。

7.5.2 STLコンテナメンバ関数一覧

以下に、本パッケージのC++ライブラリでサポートしているSTLコンテナのメンバ関数一覧を示します。

表7.5.2.1 コンテナのメンバ関数一覧

		コンテナ名									
		deque	list	map	multimap	multiset	priority_queue	queue	set	stack	vector
メンバ関数	assign	○	○								○
	at	○									○
	back	○	○					○			○
	begin	○	○	○	○	○			○		○
	capacity										○
	clear	○	○	○	○	○			○		○
	count			○	○	○			○		
	empty	○	○	○	○	○	○	○	○	○	○
	end	○	○	○	○	○			○		○
	equal_range			○	○	○			○		
	erase	○	○	○	○	○			○		○
	find			○	○	○			○		
	front	○	○					○			○
	get_allocator	○	○	○	○	○			○		○
	insert	○	○	○	○	○			○		○
	key_comp			○	○	○			○		
	lower_bound			○	○	○			○		
	max_size	○	○	○	○	○			○		○
	merge		○								
	operator[]	○		○							○
	pop						○	○		○	
	pop_back	○	○								○
	pop_front	○	○								
	push						○	○		○	
	push_back	○	○								○
	push_front	○	○								
	rbegin	○	○	○	○	○			○		○
	remove		○								
	remove_if		○								
	rend	○	○	○	○	○			○		○
	reserve										○
	resize	○	○								○
	reverse		○								
	size	○	○	○	○	○	○	○	○	○	○
	sort		○								
	splice		○								
	swap	○	○	○	○	○			○		○
	top						○			○	
	unique		○								
	upper_bound			○	○	○			○		
	value_comp			○	○	○			○		

8 アセンブラ

この章ではアセンブラ`as`の機能を説明します。アセンブリソースの文法については"4.3 アセンブリソースの文法"を参照してください。

8.1 機能

アセンブラ`as`はC/C++コンパイラが出力したアセンブリソースファイルをアセンブル(翻訳)し、機械語のオブジェクトファイルを生成します。シンボリックデバッグ用のデバッグ情報も出力可能です。このアセンブラはGNUアセンブラ(`as`)をベースとしています。アセンブラ`as`の詳細については、GNUアセンブラのドキュメントを参照してください。ドキュメントは、世界各地にあるGNUのミラーサイトからインターネット等を利用して入手可能です。

8.2 入出力ファイル

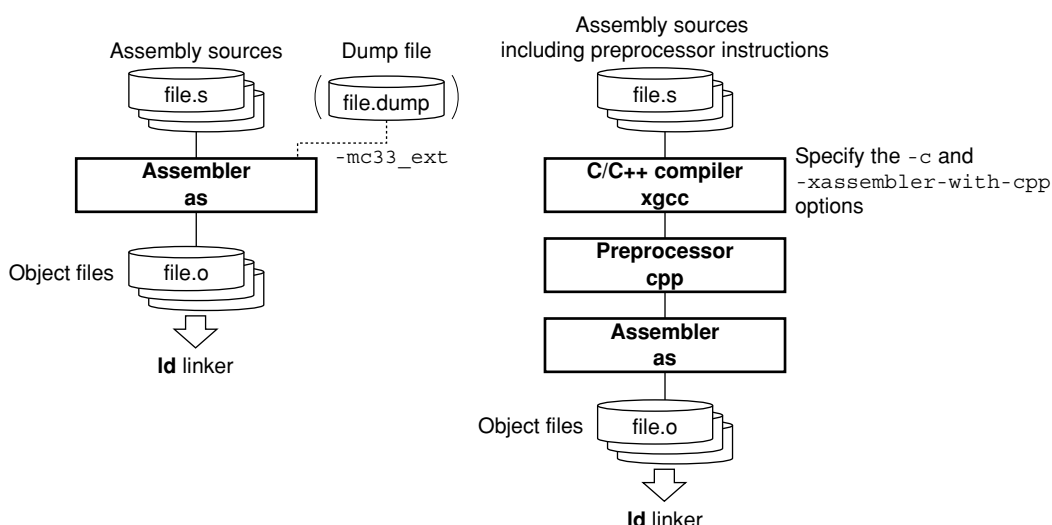


図8.2.1 フローチャート

8.2.1 入力ファイル

●アセンブリソースファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.s (" .s"以外の拡張子が使用可能、パス指定も可能)

内容: ソースプログラムを記述したファイルです。通常、C/C++コンパイラ`xgcc`が出力したファイルを入力します。

基本命令とアセンブラ擬似命令のみを記述したソースファイルを作成した場合は、直接アセンブラ`as`に入力することも可能です。

●ダンプファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.dump

内容: グローバルおよびローカルシンボル情報が記載されたファイルで、`objdump`で次のように生成します。

```
objdump -t <ファイル名>.elf > <ファイル名>.dump
```

拡張子は`.dump`である必要があります。`-mc33_ext`オプション指定時に、シンボル参照部などの最適化に使用されます。

8.2.2 出力ファイル

●オブジェクトファイル

ファイル形式: elf形式のバイナリファイル

ファイル名: <ファイル名>.o (<ファイル名>は入力ファイルと同じです。)

内容: プログラムコード(機械語)にシンボル情報やデバッグ情報等を付加したファイルです。

8.3 起動方法

8.3.1 起動フォーマット

アセンブラ`as`は次のコマンドにより起動します。

`as <オプション> <ファイル名>`

<オプション> 8.3.2項参照

<ファイル名> アセンブリソースファイルを拡張子(.s)も含めて指定します。

8.3.2 コマンドラインオプション

アセンブラ`as`はGNUアセンブラ標準のコマンドオプションを受け付けます。

ここでは、よく使用するオプションのみを説明しますので、その他のオプションも含め、詳細についてはGNUアセンブラのマニュアルを参照してください。

`-o<ファイル名>`

機能: 出力ファイル名の指定

説明: アセンブラ`as`が出力するオブジェクトファイルの名称を指定します。

<ファイル名>は`-o`に続けて入力してください。

デフォルト: `a.out`という名称のファイルを生成します。

`-a[<サブオプション>]`

機能: アセンブリリストファイルの出力

説明: アセンブリリストファイルを出力します。<サブオプション>の指定により、出力内容も選択できます。

例: `-adh1` ハイレベルな出力と、デバッグ擬似命令の削除を指定します。

デフォルト: アセンブリリストファイルは出力されません。

`--gstabs`

機能: ソースファイルの相対パスを含むデバッグ情報の付加

説明: デバッグ情報を含む出力ファイルが生成されます。

ソースファイルの位置情報は相対パスで出力されます。

デフォルト: デバッグ情報は出力されません。

標準オプションに加え、以下のS1C33専用のオプションも用意されています。

`-mc33adv`

機能: C33 ADVコア用コードの生成

説明: このオプションを指定すると、アセンブラ`as`はC33 ADVコア用のコードを生成します。標準命令に加えC33 ADVコア用の命令も出力されます。

デフォルト: C33 STDコア用のコードを生成します。

`-mc33401`

機能: C33 ADVコアのアセンブラフィルタ

説明: C33 ADVコアのアセンブラ命令の組み合わせによっては、割り込みが発生したときに正常に動作しないことがあります。本オプションは、この問題を回避するためのアセンブラフィルタ機能を有効にします。

本オプションは、必ず`-mc33adv`オプションと一緒に指定してください。

アセンブラ命令の組み合わせ等の詳細は、"`¥gnu33¥doc¥as_fil_readmeja.txt`"を参照してください。

デフォルト: アセンブラフィルタ機能は無効です。

`-mc33pe`

機能: C33 PEコア用コードの生成

説明: このオプションを指定すると、アセンブラ`as`はC33 PEコア用のコードを生成します。標準命令に加えC33 PEコア用の命令も出力されます。

デフォルト: C33 STDコア用のコードを生成します。

8 アセンブラ

-medda32

機能: デフォルトデータエリアの未使用

説明: デフォルトデータエリアを使用せず、データアクセス領域を32ビットにします。
これにより、データポインタを意識せずに、データを4G空間のどこにでも配置することが可能になります。

この指定により、コアがC33 STDまたはC33 PEであれば、デフォルトデータポインタの初期化等の処理は不要となります。(C33 ADVコアでは必要です。)

デフォルト: デフォルトデータエリアを使用します。

-mc33_ext <ダンプファイル名> <全オブジェクトダンプファイル名>

機能: 拡張命令の最適化

説明: 1パス目のアセンブルによってs*/x*拡張命令展開時に挿入されたext命令の数を、リンク時に決定した実際の参照シンボルまでの距離に応じて削除する最適化処理を指定します。一旦リンクまで処理した後の2パス目に指定します。拡張命令および最適化の詳細については、それぞれ8.7節および8.8節を参照してください。

デフォルト: 拡張命令の最適化は行われません。

コマンドラインにオプションを入力する場合、オプションの前後には1個以上のスペースが必要です。

例: as -mc33_ext test.dump allobjects.dump -o test.o test.s

8.4 スコープ

各ソースファイル内で定義されたシンボルは、そのファイル内では自由に参照することができます。このようなシンボルの参照範囲をスコープといいます。

通常シンボルは定義されたファイル内でのみ参照可能です。そのファイル内に見つからないシンボルが参照されるとアセンブラ`as`はそのシンボルを未定義シンボルとしてオブジェクトファイルを作成し、解決をリンカ`ld`に任せます。

複数のソースファイルを使用する開発ではスコープを他のソースファイルにも広げる必要があり、アセンブラ`as`にはそのための擬似命令が用意されています。それによってシンボルのグローバル宣言を行い、他のソースファイルでも参照可能にします。

定義したファイル内でのみ参照可能なシンボルをローカルシンボル、グローバル宣言されたシンボルをグローバルシンボルと呼びます。ローカルシンボルの場合、複数のファイルで同一のシンボル名を指定しても、それぞれ別のシンボルとして扱われます。グローバルシンボルは複数のファイルで多重定義するとリンカ`ld`でワーニングとなります。

例: `file1` (グローバルシンボルを定義するファイル)

```
.global  SYMBOL      ...シンボルを定義するファイルでグローバル宣言
.global  VAR1

SYMBOL:
:
:
LABEL:      ...ローカルシンボル
:          (このファイル内でのみ参照)

.section .bss
.align    2

VAR1:
.zero     4
```

`file2` (グローバルシンボルを参照するファイル)

```
ext  SYMBOL@rh
ext  SYMBOL@m
call SYMBOL@r1      ...シンボルを外部参照
:
ext  VAR1@h
ext  VAR1@m
ld.w %r1, VAR1@l    ...シンボルを外部参照
LABEL:      ...ローカルシンボル
:          (file1のLABELとは別のシンボルとして扱われます。)
```

アセンブラ`as`は`file2`のシンボル`SYMBOL`および`VAR1`をアドレス未定としてアセンブルし、その情報を出力するオブジェクトファイルに含めます。それらのアドレスはリンクの処理によって最終的に決定します。

8.5 アセンブラ擬似命令

アセンブラ擬似命令は、実行コードに変換される命令ではなく、アセンブルを制御したりデータを設定する命令です。

他の命令と区別するため、アセンブラ擬似命令はすべてピリオド(.)で始まります。

命令はすべて小文字で記述してください。パラメータは大文字と小文字が区別されます。

アセンブラ`as`ではGNUアセンブラの擬似命令がすべて使用可能です。アセンブラの擬似命令の詳細については、GNUアセンブラのドキュメントを参照してください。

8.5.1 Textセクション定義擬似命令(.text)

●命令の形式

`.text`

●機能

`.text`セクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、`.text`セクションとしてアセンブルされます。

8.5.2 Dataセクション定義擬似命令(.rodata, .data)

●命令一覧

<code>.rodata</code>	定数を配置する.rodataセクションを宣言
<code>.data</code>	初期値を持つデータを配置する.dataセクションを宣言

●命令の形式

```
.section .rodata, "a"
.section .data, "aw"
```

●機能

(1) .section .rodata, "a"

定数データセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.rodataセクションに配置されるものとしてアセンブルされます。"a"はデータの配置のみを指定する.section擬似命令のフラグで、読み出し専用の属性を持つセクションが作成されます。通常、このセクションはリンク時に読み出し専用メモリ(ROM)に配置します。

例: .section .rodata, "a" .rodataセクションを設定します。

(2) .section .data, "aw"

初期値付きデータセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.dataセクションに配置されるものとしてアセンブルされます。"aw"はデータの配置と書き込み可を指定する.section擬似命令のフラグで、リード/ライト可能な属性を持つセクションが作成されます。通常、このセクションはリンク処理で読み出し専用メモリ(ROM)に配置し、実行時はデータを使用する前にユーザプログラムでRAMなどのリード/ライト可能なメモリに転送する必要があります。

例: .section .data, "aw" .dataセクションを設定します。

●注意事項

データ定義擬似命令で1つのデータに割り当てられるメモリのサイズは以下のとおりです。

1バイト: .byte

2バイト: .short, .hword, .word

4バイト: .int, .long

8.5.3 Bssセクション定義擬似命令(.bss)

●命令一覧

.bss 初期値を持たないデータを配置する.bssセクションを宣言

●命令の形式

.section .bss

●機能

非初期化データセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.bssセクションに配置されるものとしてアセンブルされます。

例: **.section .bss** .bssセクションを設定します。

●注意事項

- デフォルト設定では、.bssセクションに記載したラベルはすべてローカルシンボルとして定義されます。グローバルシンボルとして定義するには、.global擬似命令を使用してください。

例: **.section .bss**
 .align 2

VAR1:
 .skip 4 4バイトのローカル変数VAR1を定義

.section .bss
 .global VAR2
 .align 2
VAR2:
 .skip 4 4バイトのグローバル変数VAR2を定義

- .bssセクション内では、.skip擬似命令で領域が確保できます。.space擬似命令は初期値を持つため使用できません。

8.5.4 Commセクション定義擬似命令(.comm)

●命令一覧

`.comm` `.comm`セクションに割り当てるシンボルを宣言

●命令の形式

`.comm` <シンボル名>,<サイズ>,<アライメント>

●機能

`.comm`セクションに割り当てるシンボルを宣言します。

ただし、`.comm`セクションに割り当てられるシンボルは、直前に`.local`宣言がされていないものに限りです。直前に`.local`宣言がされているシンボルは、`.bss`セクションに割り当てられます。

C++を使用するときにC/C++コンパイラによって生成されるセクションであり、ユーザが独自に作成するセクションではありません。

8.5.5 Ctorsセクション定義擬似命令(.ctors)

●命令一覧

.ctors グローバルクラスのコンストラクタ関数へのポインタ配列を配置する.ctorsセクションを宣言

●命令の形式

```
.section .ctors, "aw"
```

●機能

.ctorsセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.ctorsセクションに配置されるものとしてアセンブルされます。

"aw"はデータの配置と書き込み可を指定する.section擬似命令のフラグで、リード/ライト可能な属性を持つセクションが作成されます。

C++を使用するときにC/C++コンパイラによって生成されるセクションであり、ユーザが独自に作成するセクションではありません。

8.5.6 Dtorsセクション定義擬似命令(.dtors)

●命令一覧

.dtors グローバルクラスのデストラクタ関数へのポインタ配列を配置する.dtorsセクションを宣言

●命令の形式

```
.section .dtors,"aw"
```

●機能

.dtorsセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.dtorsセクションに配置されるものとしてアセンブルされます。

"aw"はデータの配置と書き込み可を指定する.section擬似命令のフラグで、リード/ライト可能な属性を持つセクションが作成されます。

C++を使用するときにC/C++コンパイラによって生成されるセクションであり、ユーザが独自に作成するセクションではありません。

8.5.7 Gcc_except_tableセクション定義擬似命令(.gcc_except_table)

●命令一覧

`.gcc_except_table` 例外処理用テーブルデータを配置する.gcc_except_tableセクションを宣言

●命令の形式

```
.section .gcc_except_table,"aw",@progbits  
.section .gcc_except_table
```

●機能

.gcc_except_tableセクションの開始を宣言します。本命令以降のステートメントは、他のセクションが宣言されるまで、.gcc_except_tableセクションに配置されるものとしてアセンブルされます。

"aw"はデータの配置と書き込み可を指定する.section擬似命令のフラグで、リード/ライト可能な属性を持つセクションが作成されます。

@progbitsはこのセクションが実データの存在するセクションであることを示します。

C++で例外処理を使用するときにC/C++コンパイラによって生成されるセクションであり、ユーザが独自に作成するセクションではありません。

8.5.8 データ定義擬似命令(.long, .short, .byte, .ascii, .space)

ここで説明する擬似命令は、.dataセクションまたは.textセクションにデータを定義するために使用します。

●命令一覧

.long 4バイトのデータを定義
 .short 2バイトのデータを定義
 .byte 1バイトのデータを定義
 .ascii ASCII文字列を定義
 .space 領域へのバイトデータ埋め込み

●命令の形式

.long <ワードデータ>[, <ワードデータ> ... , <ワードデータ>]
 .short <ハーフワードデータ>[, <ハーフワードデータ> ... , <ハーフワードデータ>]
 .byte <バイトデータ>[, <バイトデータ> ... , <バイトデータ>]
 .ascii "<文字列>"[, "<文字列>" ... , "<文字列>"]
 .space <バイト長>[, <バイトデータ>]

<ワードデータ> 4バイトデータ(0x0~0xffffffff)
 <ハーフワードデータ> 2バイトデータ(0x0~0xffff)
 <バイトデータ> 1バイトデータ(0x0~0xff)
 <文字列> ASCII文字列
 <バイト長> 埋め込む領域のサイズ

●機能

(1) .long, .short, .byte

ワードデータ、ハーフワードデータ、バイトデータを定義します。2つ以上のデータを指定する場合は、データ間をカンマで区切ります。定義したデータは直前に.align擬似命令がない限り、定義するデータのサイズに従った境界アドレスから配置されます。現在位置が境界アドレス以外の場合、データを配置する境界アドレスまでの間は0x00が設定されます。

例: .long 0x0, 0x1, 0x2
 .byte 0xff

これらの擬似命令以外にも、以下の擬似命令が使用可能です。

.hword .shortと同機能
 .word .shortと同機能
 .int .longと同機能

(2) .ascii

ASCII文字列を定義します。文字列は二重引用符(")で囲んで指定します。文字列にはASCII文字および¥記号で始まるエスケープシーケンスが記述可能です。たとえば、文字列中に二重引用符を設定する場合は¥", ¥を設定する場合は¥¥と記述します。2つ以上のデータを指定する場合は、データ間をカンマで区切ります。定義したデータは直前に.align擬似命令がない限り、現在のアドレスから配置されます。

例: .ascii "abc", "xyz"
 .ascii "abc¥"D¥"efg" (= abc"D"efg)

(3) .space

<バイト長>で指定される領域に<バイトデータ>を埋め込みます。データを埋め込む領域は直前に.align擬似命令がない限り、現在のアドレスから始まります。

<バイトデータ>の指定を省略すると0x0を埋め込みます。0x0の埋め込みは.zero擬似命令(次ページ参照)でも行えます。

例: .space 4, 0xff 現在のアドレスから始まる4バイトの領域に0xffを埋め込みます。
 .zero 4 (= .space 4, 0x0)

8.5.9 領域確保擬似命令(.zero)

●命令の形式

```
.zero <バイト長>
<バイト長> 領域のサイズ
```

●機能

現在の.bssセクションに<バイト長>で指定されるサイズの空白領域を確保します。直前に.align 擬似命令がない限り、現在のアドレスから領域を確保します。

例:

```
.section .bss
.global VAR1
.align 2

VAR1:
.zero 4      4バイトのグローバル変数VAR1用の領域を確保
```

8.5.10 アライメント擬似命令(.align)

●命令の形式

`.align <アライメント>`

<アライメント> 境界を指定する数値

●機能

この擬似命令の直後に出現するデータを2ⁿバイト境界にアライメントします(n=<アライメント>)。

例: `.align 2` 4バイト境界へのアライメント

●注意事項

`.align`擬似命令は直後にあるデータ定義または領域確保擬似命令に対してのみ有効です。したがって、アライメントが必要なデータを定義する場合には、データ定義擬似命令個々に`.align`擬似命令を使用する必要があります。

8.5.11 グローバル宣言擬似命令(.global)

●命令の形式

`.global <シンボル>`

<シンボル> 現在のファイルで定義するグローバルシンボル

●機能

シンボルのグローバル宣言を行います。指定のシンボルは、他のモジュールから参照可能なグローバルシンボルに設定されます。

例: `.global SUB1`

●注意事項

本擬似命令で宣言されないシンボルは、すべてローカルシンボルになります。

8.5.12 シンボル定義擬似命令(.set)

●命令の形式

`.set <シンボル>,<アドレス>`

<シンボル> メモリアクセス(アドレス参照)用のシンボル

<アドレス> 絶対アドレス

●機能

シンボルに絶対アドレス(32ビット)を定義します。

例: `.set DATA1,0x80000` 絶対アドレス0x80000を示すシンボルDATA1を定義します。

●注意事項

設定したシンボルはローカルシンボルとなります。グローバルシンボルとして使用する場合は、`.global`擬似命令によるグローバル宣言を行ってください。

8.6 擬似オペランド

アセンブラasには、データエリア開始アドレスからのオフセットアドレスを指定するための擬似オペランドが用意されています。データエリアについては、"3.7.1 データエリア"を参照してください。以下に示す5種類の擬似オペランドが使用可能です。

●擬似オペランド一覧

doff_hi (<シンボル>) デフォルトデータエリア先頭(%r15/ __dp)からのオフセットの上位13ビットを取得
doff_lo (<シンボル>) デフォルトデータエリア先頭(%r15/ __dp)からのオフセットの下位13ビットを取得
 以下のオペランドはC33 ADVコア用で、-mc33advオプション指定時に使用可能です。
dpoff_h (<シンボル>) デフォルトデータエリア先頭(%dp/ __dp)からのオフセットの上位13ビットを取得
dpoff_m (<シンボル>) デフォルトデータエリア先頭(%dp/ __dp)からのオフセットの中間13ビットを取得
dpoff_l (<シンボル>) デフォルトデータエリア先頭(%dp/ __dp)からのオフセットの下位6ビットを取得

●機能

データエリアをアクセスする場合、データエリアのベースアドレス(データエリアポインタ)をレジスタ(%r15, %dp)に設定し、ベースアドレスからのオフセットをext命令で指定します。擬似オペランドはその際に、ext命令のオペランドとして使用することができます。

例: ext doff_hi(FOO)
 ext doff_lo(FOO)
 ld.w %r0, [%r15] ← "ld.w %r0, [%r15 + (FOO address - __dp)]"として機能

これらの擬似オペランドにより得られるオフセット値は次のように計算されます。

<シンボルの再配置アドレス> - <リンクスクリプトファイルで指定されたデータエリアポインタ値(__*dp)>

●注意事項

- 擬似オペランドは定義されたシンボルにのみ有効です。数値に擬似オペランドを使用した場合はエラーとなります。
- シンボルはデータエリアポインタ(__dp)より上位のアドレスに配置されている必要があります。データエリアポインタがシンボルより上位のアドレスを示していると、リンク時にエラーとなります。
- 擬似オペランドを使用する場合は、データエリアポインタにデータエリアの開始アドレスを設定しておく必要があります。
 %r15 = __dp

8.7 拡張命令

アセンブラ`as`は以下に説明する拡張命令に対応しています。拡張命令は、通常`ext`命令を含む複数の命令で記述する内容を1つの命令として記述できるようにしたもので、命令の機能およびオペランドの即値サイズに従って必要最小限の基本命令に展開されます。

説明に使用するシンボル

`immX` 符号なし X ビット即値
`signX` 符号付き X ビット即値
`symbol` メモリアドレスを示すシンボル
`label` 分岐先ラベル
 $(X:Y)$ ビット X からビット Y のビットフィールド

8.7.1 算術演算命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
<code>xadd %rd, imm32</code>	$\%rd \leftarrow \%rd + imm32$	(1)
<code>xsub %rd, imm32</code>	$\%rd \leftarrow \%rd - imm32$	(1)

これらの拡張命令により、加減算で32ビット即値が直接指定可能となります。

●展開後の基本命令

`xadd` `add`命令に展開
`xsub` `sub`命令に展開

●展開形式

(1) `xOP    %rd, imm32` ($OP = \text{add, sub}$)

例: `xadd    %rd, imm32`

$imm32 \leq 0x3f$	$0x3f < imm32 \leq 0x7fff$	$imm32 > 0x7fff$
<code>add %rd, imm32(5:0)</code>	<code>ext imm32(18:6)</code> <code>add %rd, imm32(5:0)</code>	<code>ext imm32(31:19)</code> <code>ext imm32(18:6)</code> <code>add %rd, imm32(5:0)</code>

8.7.2 比較命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
<code>xcmp %rd, sign32</code>	<code>%rd-sign32</code> (PSRのC, V, Z, Nフラグを変更)	(1)

この拡張命令により、汎用レジスタと符号付き32ビット即値との比較が行えます。

●展開後の基本命令

`xcmp` `cmp`命令に展開

●展開形式

(1) `xcmp    %rd, sign32`

$-32 \leq sign32 \leq 31$	$-262144 \leq sign32 < -32$ または $31 < sign32 \leq 262143$	$sign32 < -262144$ または $262143 < sign32$
<code>cmp %rd, sign32(5:0)</code>	<code>ext sign32(18:6)</code> <code>cmp %rd, sign32(5:0)</code>	<code>ext sign32(31:19)</code> <code>ext sign32(18:6)</code> <code>cmp %rd, sign32(5:0)</code>

8.7.3 論理演算命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
xand <i>%rd, sign32</i>	$\%rd \leftarrow \%rd \ \& \ sign32$	(1)
xoor <i>%rd, sign32</i>	$\%rd \leftarrow \%rd \ \ sign32$	(1)
xxor <i>%rd, sign32</i>	$\%rd \leftarrow \%rd \ \wedge \ sign32$	(1)
xnot <i>%rd, sign32</i>	$\%rd \leftarrow !sign32$	(1)

これらの拡張命令により、論理演算で符号付き32ビット即値が直接指定可能となります。

●展開後の基本命令

xand and命令に展開
xoor or命令に展開
xxor xor命令に展開
xnot not命令に展開

●展開形式

(1) **xOP** *%rd, sign32* (OP = and, oor, xor, not)

例: **xand** *%rd, sign32*

$-32 \leq sign32 \leq 31$	$-262144 \leq sign32 < -32$ または $31 < sign32 \leq 262143$	$sign32 < -262144$ または $262143 < sign32$
and <i>%rd, sign32(5:0)</i>	ext <i>sign32(18:6)</i> and <i>%rd, sign32(5:0)</i>	ext <i>sign32(31:19)</i> ext <i>sign32(18:6)</i> and <i>%rd, sign32(5:0)</i>

8.7.4 シフト&ローテート命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
<code>xshr %rd, imm5</code>	論理右シフト	(1)
<code>xshl %rd, imm5</code>	論理左シフト	(1)
<code>xsra %rd, imm5</code>	算術右シフト	(1)
<code>xsla %rd, imm5</code>	算術左シフト	(1)
<code>xrr %rd, imm5</code>	右ローテート	(1)
<code>xrl %rd, imm5</code>	左ローテート	(1)

これらの拡張命令により、シフト・ローテートのシフト量を31ビットまで指定することができます。

●展開後の基本命令

xshr srl命令に展開
xshl shl命令に展開
xsra sra命令に展開
xsla sla命令に展開
xrr rr命令に展開
xrl rl命令に展開

●展開形式

(1) **xOP %rd, imm5** ($OP = srl, shl, sra, sla, rr, rl$)

例: `xshr %rd, imm5`

$imm5 \leq 8$	$8 < imm5 < 16$	$imm5 = 16$
<code>srl %rd, imm5(3:0)</code>	<code>srl %rd, 0x8</code> <code>srl %rd, imm5(2:0)</code>	<code>srl %rd, 0x8</code> <code>srl %rd, 0x8</code>
$16 < imm5 \leq 24$	$imm5 > 24$	
<code>srl %rd, 0x8</code> <code>srl %rd, 0x8</code> <code>srl %rd, imm5(3:0)</code>	<code>srl %rd, 0x8</code> <code>srl %rd, 0x8</code> <code>srl %rd, 0x8</code> <code>srl %rd, imm5(2:0)</code>	

C33 ADVコア用の`-mc33adv`オプションを指定した場合、これらの拡張命令は基本命令1個に展開されます。

例: `xshr %rd, imm5` → `srl %rd, imm5`

8.7.5 スタック～レジスタ間データ転送命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
xld.b %rd, [%sp+imm32]	%rd ← B[%sp+imm32] (符号拡張)	(1)
xld.ub %rd, [%sp+imm32]	%rd ← B[%sp+imm32] (ゼロ拡張)	(1)
xld.h %rd, [%sp+imm32]	%rd ← H[%sp+imm32] (符号拡張)	(2)
xld.uh %rd, [%sp+imm32]	%rd ← H[%sp+imm32] (ゼロ拡張)	(2)
xld.w %rd, [%sp+imm32]	%rd ← W[%sp+imm32]	(3)
xld.b [%sp+imm32], %rs	B[%sp+imm32] ← %rs(7:0)	(1)
xld.h [%sp+imm32], %rs	H[%sp+imm32] ← %rs(15:0)	(2)
xld.w [%sp+imm32], %rs	W[%sp+imm32] ← %rs	(3)

これらの拡張命令により、32ビットまでのディスプレースメントが直接指定可能となります。
imm32は省略可能です。

●展開後の基本命令

xld.b ld.b命令に展開
xld.ub ld.ub命令に展開
xld.h ld.h命令に展開
xld.uh ld.uh命令に展開
xld.w ld.w命令に展開

●展開形式

imm32を省略した場合、[%sp+0x0]を指定したものととして展開されます。

(1) バイトデータ転送(xld.b, xld.ub)

例: xld.b %rd, [%sp+imm32]

imm32 ≤ 0x3f	0x3f < imm32 ≤ 0x7fff	imm32 > 0x7fff
ld.b %rd, [%sp+imm32(5:0)]	ext imm32(18:6) ld.b %rd, [%sp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.b %rd, [%sp+imm32(5:0)]

(2) ハーフワードデータ転送(xld.h, xld.uh)

例: xld.h %rd, [%sp+imm32]

imm32 ≤ 0x7f	0x7f < imm32 ≤ 0x7fff	imm32 > 0x7fff
ld.h %rd, [%sp+imm32(6:1)]	ext imm32(18:6) ld.h %rd, [%sp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.h %rd, [%sp+imm32(5:0)]

(3) ワードデータ転送(xld.w)

例: xld.w %rd, [%sp+imm32]

imm32 ≤ 0xff	0xff < imm32 ≤ 0x7fff	imm32 > 0x7fff
ld.w %rd, [%sp+imm32(7:2)]	ext imm32(18:6) ld.w %rd, [%sp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.w %rd, [%sp+imm32(5:0)]

8.7.6 メモリ～レジスタ間データ転送命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
xld.b %rd, [symbol+imm26]	%rd ← B[symbol+imm26] (符号拡張) *	(1)
xld.ub %rd, [symbol+imm26]	%rd ← B[symbol+imm26] (ゼロ拡張) *	(1)
xld.h %rd, [symbol+imm26]	%rd ← H[symbol+imm26] (符号拡張) *	(1)
xld.uh %rd, [symbol+imm26]	%rd ← H[symbol+imm26] (ゼロ拡張) *	(1)
xld.w %rd, [symbol+imm26]	%rd ← W[symbol+imm26] *	(1), (2)
xld.b [symbol+imm26], %rs	B[symbol+imm26] ← %rs(7:0) *	(1)
xld.h [symbol+imm26], %rs	H[symbol+imm26] ← %rs(15:0) *	(1)
xld.w [symbol+imm26], %rs	W[symbol+imm26] ← %rs *	(1)
xld.b %rd, [%rb+imm26]	%rd ← B[%rb+imm26] (符号拡張)	(3)
xld.ub %rd, [%rb+imm26]	%rd ← B[%rb+imm26] (ゼロ拡張)	(3)
xld.h %rd, [%rb+imm26]	%rd ← H[%rb+imm26] (符号拡張)	(3)
xld.uh %rd, [%rb+imm26]	%rd ← H[%rb+imm26] (ゼロ拡張)	(3)
xld.w %rd, [%rb+imm26]	%rd ← W[%rb+imm26]	(3)
xld.b [%rb+imm26], %rs	B[%rb+imm26] ← %rs(7:0)	(3)
xld.h [%rb+imm26], %rs	H[%rb+imm26] ← %rs(15:0)	(3)
xld.w [%rb+imm26], %rs	W[%rb+imm26] ← %rs	(3)

* C33 ADVコアでは、imm26がimm32に拡張されています。

これらの拡張命令により、シンボルや即値でアドレスを指定するメモリアクセスが可能となります。ただし、ポストインクリメント機能([+])は使用できません。imm26(imm32)は省略可能です。

●展開後の基本命令

xld.b 1d.b命令に展開
xld.ub 1d.ub命令に展開
xld.h 1d.h命令に展開
xld.uh 1d.uh命令に展開
xld.w 1d.w命令に展開

●展開形式

(1) xld.* %rd, [symbol+imm26] (* = b, ub, h, uh, w)
xld.* [symbol+imm26], %rs (* = b, h, w)

例: xld.w %rd, [symbol+imm26]

無条件	
ext	(symbol+imm26) (25:13)
ext	(symbol+imm26) (12:0)
ld.w	%rd, [%r15]

imm26を省略した場合、[symbol+0x0]を指定したものとして展開されます。

-medda32オプション指定時

xld.* %rd, [symbol+imm26]	xld.* [symbol+imm26], %rs (%rs ≠ %r0)	xld.* [symbol+imm26], %rs (%rs = %r0)
ext (symbol+imm26)@h	pushn %r0	pushn %r1 (or push %r1)*1
ext (symbol+imm26)@m	ext (symbol+imm26)@h	ext (symbol+imm26)@h
ld.w %rd, (symbol+imm26)@l	ext (symbol+imm26)@m	ext (symbol+imm26)@m
ld.* %rd, [%rd]	ld.w %r0, (symbol+imm26)@l	ld.w %r1, (symbol+imm26)@l
	ld.* [%r0], %rs	ld.* [%r1], %r0
	popn %r0	popn %r1 (or pop %r1)*1

*1 -mc33advまたは-mc33peオプション指定時

(2) xld.w %rd, [symbol+imm32]

-mc33advオプション指定時

無条件	
ext	(symbol+imm32) (31:19)
ext	(symbol+imm32) (18:6)
ld.w	%rd, [%dp+(symbol+imm32) (5:0)]

(3) xld.* %rd, [%rb+imm26] (* = b, ub, h, uh, w)**xld.* [%rb+imm26], %rs** (* = b, h, w)

例: xld.w %rd, [%rb+imm26]

imm26 = 0		1 ≤ imm26 ≤ 0x1fff	imm26 > 0x1fff
ld.w	%rd, [%rb]	ext imm26 (12:0)	ext imm26 (25:13)
		ld.w %rd, [%rb]	ext imm26 (12:0)
			ld.w %rd, [%rb]

imm26を省略した場合、[%rb+0x0]を指定したものとして展開されます。

8.7.7 即値ロード命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
xld.w %rd, symbol+imm32	%rd ← symbol+imm32	(1)
xld.w %rd, sign32	%rd ← sign32	(2)

* "symbol+imm32"は"symbol+imm32"と"symbol-imm32"の指定が可能

これらの拡張命令により、32ビットの即値を直接汎用レジスタにロードすることができます。即値指定にはシンボルも使用可能です。

●展開後の基本命令

xld.w ld.w命令に展開

●展開形式

(1)xld.w %rd, symbol+imm32
xld.w %rd, symbol-imm32
例: xld.w %rd, symbol+imm32

無条件	
ext	(symbol+imm32)@h
ext	(symbol+imm32)@m
ld.w	%rd, (symbol+imm32)@l

imm32は省略可能です。imm32を省略した場合、symbol+0x0を指定したものとして展開されます。

(2)xld.w %rd, sign32

-32 ≤ sign32 ≤ 31	-262144 ≤ sign32 < -32 または 31 < sign32 ≤ 262143	sign32 < -262144 または 262143 < sign32
ld.w %rd, sign32(5:0)	ext sign32(18:6) ld.w %rd, sign32(5:0)	ext sign32(31:19) ext sign32(18:6) ld.w %rd, sign32(5:0)

8.7.8 ビット操作命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
xbtst <i>[symbol+imm26], imm3</i>	B <i>[symbol+imm26]</i> のビットテスト *	(1)
xbclr <i>[symbol+imm26], imm3</i>	B <i>[symbol+imm26]</i> のビットクリア *	(1)
xbset <i>[symbol+imm26], imm3</i>	B <i>[symbol+imm26]</i> のビットセット *	(1)
xbnot <i>[symbol+imm26], imm3</i>	B <i>[symbol+imm26]</i> のビット反転 *	(1)
xbtst <i>[%rb+imm26], imm3</i>	B[%rb+imm26] のビットテスト	(2)
xbclr <i>[%rb+imm26], imm3</i>	B[%rb+imm26] のビットクリア	(2)
xbset <i>[%rb+imm26], imm3</i>	B[%rb+imm26] のビットセット	(2)
xbnot <i>[%rb+imm26], imm3</i>	B[%rb+imm26] のビット反転	(2)

* C33 ADVコアでは、*imm26*が*imm32*に拡張されています。

これらの拡張命令により、シンボルや即値でビットを操作するメモリアドレスが指定可能となります。*imm26*(*imm32*)は省略可能です。

●展開後の基本命令

xbtst btst命令に展開
xbclr bclr命令に展開
xbset bset命令に展開
xbnot bnot命令に展開

●展開形式

(1) xOP *[symbol+imm26], imm3* (OP = btst, bclr, bset, bnot)

例: xbtst *[symbol+imm26], imm3*

無条件	
ext	(<i>symbol+imm26</i>) (25:13)
ext	(<i>symbol+imm26</i>) (12:0)
btst	[%r15], <i>imm3</i>

*imm26*を省略した場合、*[symbol+0x0]*を指定したものとして展開されます。

-medda32オプション指定時

例: xbtst *[symbol+imm26], imm3*

無条件	
pushn	%r0
ext	(<i>symbol+imm26</i>)@h
ext	(<i>symbol+imm26</i>)@m
ld.w	%r0, (<i>symbol+imm26</i>)@l
btst	[%r0], <i>imm3</i>
popn	%r0

(2) xOP *[%rb+imm26], imm3* (OP = btst, bclr, bset, bnot)

例: xbtst *[%rb+imm26], imm3*

<i>imm26</i> = 0	$1 \leq imm26 \leq 0x1fff$	<i>imm26</i> > 0x1fff
btst %rd, [%rb]	ext <i>imm26</i> (12:0)	ext <i>imm26</i> (25:13)
	btst %rd, [%rb]	ext <i>imm26</i> (12:0)
		btst %rd, [%rb]

*imm26*を省略した場合、*[%rb+0x0]*を指定したものとして展開されます。

8.7.9 分岐命令

●拡張命令の種類と機能

拡張命令		機能	展開形式
scall	label+imm32	サブルーチンコール	(1)
scall.d	label+imm32	サブルーチンコール(ディレイド分岐)	(1)
sjp	label+imm32	無条件ジャンプ	(1)
sjp.d	label+imm32	無条件ジャンプ(ディレイド分岐)	(1)
sjreq	label+imm32	条件付きジャンプ	(1)
sjreq.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
sjrne	label+imm32	条件付きジャンプ	(1)
sjrne.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
sjrgt	label+imm32	条件付きジャンプ	(1)
sjrgt.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
sjrge	label+imm32	条件付きジャンプ	(1)
sjrge.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
sjrlt	label+imm32	条件付きジャンプ	(1)
sjrlt.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
sjrle	label+imm32	条件付きジャンプ	(1)
sjrle.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
sjrugt	label+imm32	条件付きジャンプ	(1)
sjrugt.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
sjruge	label+imm32	条件付きジャンプ	(1)
sjruge.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
sjrult	label+imm32	条件付きジャンプ	(1)
sjrult.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
sjrule	label+imm32	条件付きジャンプ	(1)
sjrule.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(1)
scall	sign22	サブルーチンコール	(2)
scall.d	sign22	サブルーチンコール(ディレイド分岐)	(2)
sjp	sign22	無条件ジャンプ	(2)
sjp.d	sign22	無条件ジャンプ(ディレイド分岐)	(2)
sjreq	sign22	条件付きジャンプ	(2)
sjreq.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)
sjrne	sign22	条件付きジャンプ	(2)
sjrne.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)
sjrgt	sign22	条件付きジャンプ	(2)
sjrgt.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)
sjrge	sign22	条件付きジャンプ	(2)
sjrge.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)
sjrlt	sign22	条件付きジャンプ	(2)
sjrlt.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)
sjrle	sign22	条件付きジャンプ	(2)
sjrle.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)
sjrugt	sign22	条件付きジャンプ	(2)
sjrugt.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)
sjruge	sign22	条件付きジャンプ	(2)
sjruge.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)
sjrult	sign22	条件付きジャンプ	(2)
sjrult.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)
sjrule	sign22	条件付きジャンプ	(2)
sjrule.d	sign22	条件付きジャンプ(ディレイド分岐)	(2)

拡張命令		機能	展開形式
xcall	label+imm32	サブルーチンコール	(3)
xcall.d	label+imm32	サブルーチンコール(ディレイド分岐)	(3)
xjp	label+imm32	無条件ジャンプ	(3)
xjp.d	label+imm32	無条件ジャンプ(ディレイド分岐)	(3)
xjreq	label+imm32	条件付きジャンプ	(3)
xjreq.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xjrne	label+imm32	条件付きジャンプ	(3)
xjrne.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xjrgt	label+imm32	条件付きジャンプ	(3)
xjrgt.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xjrge	label+imm32	条件付きジャンプ	(3)
xjrge.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xjrlt	label+imm32	条件付きジャンプ	(3)
xjrlt.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xjrle	label+imm32	条件付きジャンプ	(3)
xjrle.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xjrugt	label+imm32	条件付きジャンプ	(3)
xjrugt.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xjruge	label+imm32	条件付きジャンプ	(3)
xjruge.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xjrult	label+imm32	条件付きジャンプ	(3)
xjrult.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xjrle	label+imm32	条件付きジャンプ	(3)
xjrle.d	label+imm32	条件付きジャンプ(ディレイド分岐)	(3)
xcall	sign32	サブルーチンコール	(4)
xcall.d	sign32	サブルーチンコール(ディレイド分岐)	(4)
xjp	sign32	無条件ジャンプ	(4)
xjp.d	sign32	無条件ジャンプ(ディレイド分岐)	(4)
xjreq	sign32	条件付きジャンプ	(4)
xjreq.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)
xjrne	sign32	条件付きジャンプ	(4)
xjrne.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)
xjrgt	sign32	条件付きジャンプ	(4)
xjrgt.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)
xjrge	sign32	条件付きジャンプ	(4)
xjrge.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)
xjrlt	sign32	条件付きジャンプ	(4)
xjrlt.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)
xjrle	sign32	条件付きジャンプ	(4)
xjrle.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)
xjrugt	sign32	条件付きジャンプ	(4)
xjrugt.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)
xjruge	sign32	条件付きジャンプ	(4)
xjruge.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)
xjrult	sign32	条件付きジャンプ	(4)
xjrult.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)
xjrle	sign32	条件付きジャンプ	(4)
xjrle.d	sign32	条件付きジャンプ(ディレイド分岐)	(4)

これらの拡張命令により、ディスプレイースメント付きラベル、符号付き22/32ビット即値で分岐先を指定することができます。条件付きジャンプ命令の分岐条件は基本命令と同じです。

●展開後の基本命令

<code>scall/scall.d/xcall/xcall.d</code>	<code>call/call.d</code> 命令に展開
<code>sjp/sjp.d/xjp/xjp.d</code>	<code>jp/jp.d</code> 命令に展開
<code>sjreq/sjreq.d/xjreq/xjreq.d</code>	<code>jreq/jreq.d</code> 命令に展開
<code>sjrne/sjrne.d/xjrne/xjrne.d</code>	<code>jrne/jrne.d</code> 命令に展開
<code>sjrgt/sjrgt.d/xjrgt/xjrgt.d</code>	<code>jrgt/jrgt.d</code> 命令に展開
<code>sjrge/sjrge.d/xjrge/xjrge.d</code>	<code>jrge/jrge.d</code> 命令に展開
<code>sjrlt/sjrlt.d/xjrlt/xjrlt.d</code>	<code>jrlt/jrlt.d</code> 命令に展開
<code>sjrle/sjrle.d/xjrle/xjrle.d</code>	<code>jrle/jrle.d</code> 命令に展開
<code>sjrugt/sjrugt.d/xjrugt/xjrugt.d</code>	<code>jrugt/jrugt.d</code> 命令に展開
<code>sjruge/sjruge.d/xjruge/xjruge.d</code>	<code>jruge/jruge.d</code> 命令に展開
<code>sjrult/sjrult.d/xjrult/xjrult.d</code>	<code>jrult/jrult.d</code> 命令に展開
<code>sjrule/sjrule.d/xjrule/xjrule.d</code>	<code>jrle/jrule.d</code> 命令に展開

●展開形式

- (1) **sOP label+imm32** (OP = call, call.d, jp, jp.d, jr*, jr*.d)

例: `scall label+imm32`

無条件	
ext	(label+imm32)@rm
call	(label+imm32)@r1

imm32は省略可能です。imm32を省略した場合、label+0x0を指定したものとして展開されます。

- (2) **sOP sign22** (OP = call, call.d, jp, jp.d, jr*, jr*.d)

例: `scall sign22`

-256 ≤ sign22 ≤ 254	-2097152 ≤ sign22 < -256 または 254 < sign22 ≤ 2097150
call sign22(8:1)	ext sign22(21:9) call sign22(8:1)

- (3) **xOP label+imm32** (OP = call, call.d, jp, jp.d, jr*, jr*.d)

例: `xcall label+imm32`

無条件	
ext	(label+imm32)@rh
ext	(label+imm32)@rm
call	(label+imm32)@r1

imm32は省略可能です。imm32を省略した場合、label+0x0を指定したものとして展開されます。

- (4) **xOP sign32** (OP = call, call.d, jp, jp.d, jr*, jr*.d)

例: `xcall sign32`

-256 ≤ sign32 ≤ 254	-2097152 ≤ sign32 < -256 または 254 < sign32 ≤ 2097150	sign32 < -2097152 または 2097150 < sign32
call sign32(8:1)	ext sign32(21:9) call sign32(8:1)	ext sign32(31:22) < 0x03 ext sign32(21:9) call sign32(8:1)

8.7.10 C33 ADVコア用メモリ～レジスタ間データ転送命令

ここで説明する拡張命令はC33 ADVコア専用で、-mc33advオプションを指定した場合にのみ使用可能です。

●拡張命令の種類と機能

拡張命令	機能	展開形式
ald.b %rd, [symbol+imm19]	%rd ← B[symbol+imm19] (符号拡張)	(1)
ald.ub %rd, [symbol+imm19]	%rd ← B[symbol+imm19] (ゼロ拡張)	(1)
ald.h %rd, [symbol+imm19]	%rd ← H[symbol+imm19] (符号拡張)	(1)
ald.uh %rd, [symbol+imm19]	%rd ← H[symbol+imm19] (ゼロ拡張)	(1)
ald.w %rd, [symbol+imm19]	%rd ← W[symbol+imm19]	(1)
ald.b [symbol+imm19], %rs	B[symbol+imm19] ← %rs(7:0)	(1)
ald.h [symbol+imm19], %rs	H[symbol+imm19] ← %rs(15:0)	(1)
ald.w [symbol+imm19], %rs	W[symbol+imm19] ← %rs	(1)

これらの拡張命令により、シンボルや32ビット即値でアドレスを指定するメモリアクセスが可能となります。ただし、ポストインクリメント機能([+])は使用できません。imm19は省略可能です。

●展開後の基本命令

ald.b ld.b命令に展開
 ald.ub ld.ub命令に展開
 ald.h ld.h命令に展開
 ald.uh ld.uh命令に展開
 ald.w ld.w命令に展開

●展開形式

imm19を省略した場合、[symbol+0x0]を指定したものとして展開されます。

(1) ald.* %rd, [symbol+imm19] (* = b, ub, h, uh, w)

ald.* [symbol+imm19], %rs (* = b, h, w)

例: ald.w %rd, [symbol+imm19]

無条件	
ext	(symbol+imm19)(18:6)
ld.w	%rd, [%dp+(symbol+imm19)(5:0)]

-medda32オプション指定時

ald.* %rd, [symbol+imm19]	ald.* [symbol+imm19], %rs (%rs ≠ %r0)	ald.* [symbol+imm19], %rs (%rs = %r0)
ext (symbol+imm19)@m	pushn %r0	pushn %r1 (or push %r1)*1
ld.w %rd, (symbol+imm19)@l	ext (symbol+imm19)@m	ext (symbol+imm19)@m
ld.* %rd, [%rd]	ld.w %r0, (symbol+imm19)@l	ld.w %r1, (symbol+imm19)@l
	ld.* [%r0], %rs	ld.* [%r1], %r0
	popn %r0	popn %r1 (or pop %r1)*1

*1 -mc33advまたは-mc33peオプション指定時

8.7.11 C33 ADVコア用%dpエリア～レジスタ間データ転送命令

ここで説明する拡張命令はC33 ADVコア専用で、-mc33advオプションを指定した場合にのみ使用可能です。

●拡張命令の種類と機能

拡張命令	機能	展開形式
xld.b %rd, [%dp+imm32]	%rd ← B[%dp+imm32] (符号拡張)	(1)
xld.ub %rd, [%dp+imm32]	%rd ← B[%dp+imm32] (ゼロ拡張)	(1)
xld.h %rd, [%dp+imm32]	%rd ← H[%dp+imm32] (符号拡張)	(2)
xld.uh %rd, [%dp+imm32]	%rd ← H[%dp+imm32] (ゼロ拡張)	(2)
xld.w %rd, [%dp+imm32]	%rd ← W[%dp+imm32]	(3)
xld.b [%dp+imm32], %rs	B[%dp+imm32] ← %rs(7:0)	(1)
xld.h [%dp+imm32], %rs	H[%dp+imm32] ← %rs(15:0)	(2)
xld.w [%dp+imm32], %rs	W[%dp+imm32] ← %rs	(3)

これらの拡張命令により、32ビットまでのディスプレイースメントが直接指定可能となります。ただし、ポストインクリメント機能([+])は使用できません。imm32は省略可能です。

●展開後の基本命令

xld.b 1d.b命令に展開
 xld.ub 1d.ub命令に展開
 xld.h 1d.h命令に展開
 xld.uh 1d.uh命令に展開
 xld.w 1d.w命令に展開

●展開形式

imm32を省略した場合、[%dp+0x0]を指定したものとして展開されます。

(1) バイトデータ転送(xld.b, xld.ub)

例: xld.b %rd, [%dp+imm32]

imm32 ≤ 0x3f	0x3f < imm32 ≤ 0x7fff	imm32 > 0x7fff
ld.b %rd, [%dp+imm32(5:0)]	ext imm32(18:6) ld.b %rd, [%dp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.b %rd, [%dp+imm32(5:0)]

(2) ハーフワードデータ転送(xld.h, xld.uh)

例: xld.h %rd, [%dp+imm32]

imm32 ≤ 0x7f	0x7f < imm32 ≤ 0x7fff	imm32 > 0x7fff
ld.h %rd, [%dp+imm32(6:1)]	ext imm32(18:6) ld.h %rd, [%dp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.h %rd, [%dp+imm32(5:0)]

(3) ワードデータ転送(xld.w)

例: xld.w %rd, [%dp+imm32]

imm32 ≤ 0xff	0xff < imm32 ≤ 0x7fff	imm32 > 0x7fff
ld.w %rd, [%dp+imm32(7:2)]	ext imm32(18:6) ld.w %rd, [%dp+imm32(5:0)]	ext imm32(31:19) ext imm32(18:6) ld.w %rd, [%dp+imm32(5:0)]

8.8 拡張命令の最適化

C/C++コンパイラは、すべてのグローバルシンボルおよび一部のローカルシンボルのアドレス参照を`s*`または`x*`拡張命令にコンパイルします。これらの拡張命令は前節に示したとおり、アセンブルにより`ext`命令付きの基本命令に展開されます。

拡張命令のオペランドが即値の場合、アセンブラは即値のサイズに従い、基本命令に0～2個の`ext`命令を付加しますので、その時点で最適化は完了します。

拡張命令のオペランドがシンボルの場合、リンクが完了するまでその値が確定しませんので、アセンブラは全メモリ空間が参照可能となる数の`ext`命令を付加します。このため、不要な`ext`命令も出力され、コードサイズが大きくなります。この不要な`ext`命令を削除するための最適化オプションが`-mc33_ext`です。

● `-mc33_ext` オプション

```
-mc33_ext filename.dump allobjects.dump
```

`filename.dump` グローバルおよびローカルシンボル情報が記載されたファイルで、**objdump**で次のように生成します。拡張子は`.dump`である必要があります。

```
objdump -t filename.elf > filename.dump
```

`allobjects.dump` 1パス目のアセンブル終了時点の、各オブジェクトファイルのグローバルシンボル情報が記載されたファイルです。すべてのオブジェクトファイルを対象として次のように**objdump**を実行し、このファイルを生成します。拡張子は`.dump`である必要があります。

```
objdump -t *.o >> allobjects.dump
```

このファイルは、オブジェクトファイル間で重複する可能性のあるグローバルシンボルを、最適化の対象としないようにするために参照されます。

```
例: objdump -t test.elf > test.dump
     objdump -t *.o >> allobjects.dump
     as -mc33_ext test.dump allobjects.dump -o test.o test.s
```

● 最適化手順

最適化にはリンクにより確定するシンボルのアドレス値が必要で、アセンブラはその情報を`-mc33_ext`オプションで指定されるダンプファイルから取得します。したがって、次の手順のとおりアセンブルをリンクの前後で実行するように**make**ファイルを作成し、2パスメイクを行う必要があります。

1. コンパイル
2. アセンブル(`-mc33_ext`オプションなし)
3. リンク(ダンプファイルの出力に必要)
4. **objdump**でelfファイルからダンプファイルを生成
5. 全オブジェクトダンプファイル(`allobjects.dump`)の生成
6. 再アセンブル(`-mc33_ext`オプション付き) *2と同じ入出力ファイルを指定

IDEで2パスメイクを指定した場合、生成される**make**ファイルにはこの手順が記述されています。

●最適化パターン

最適化の処理は、以下の5つのパターンに分けられます。

最適化1: データ転送命令の場合(メモリ→レジスタ)

```
例:  sub:
      ret

      main:
      ...
      xld.w    %r0, [sub]
      ...
      ret
```

-medda32指定時

展開形式

```
xld.*    %r0, [sub]                (*はb、ub、h、uh、wのいずれか)
      ↓
ext      <シンボルの上位13ビット>
ext      <シンボルの中間13ビット>
ld.w     %rd, <シンボルの下位6ビット>
ld.*     %rd, [%rd]
```

シンボル(sub)のアドレスをダンプファイルから取得し、extの数を決定します。

シンボルのアドレス = 0～0x1f: extなしで展開

シンボルのアドレス = 0x20～0x3fff: ext 1個で展開

シンボルのアドレス = 0x40000～0xfffff: ext 2個で展開

-medda32未指定時

展開形式1: C33 STD、C33 PEの場合

```
xld.w    %r0, [sub]
      ↓
ext      imm13
ext      imm13
ld.w     %r0, [%r15]
```

シンボル(sub)のアドレスと、%r15が表すデータポインタの値からextの数を決定します。

シンボルのアドレス - データポインタ = 0～0x1fff: ext 1個で展開

シンボルのアドレス - データポインタ = 0x2000～0x3fffff: ext 2個で展開

展開形式2: S1C33401の場合

```
xld.w    %r0, [sub]
      ↓
ext      imm13
ext      imm13
ld.w     %r0, [%dp+imm6]
```

シンボル(sub)のアドレスと、%dpが表すデータポインタの値からextの数を決定します。

シンボルのアドレス - データポインタ = 0～0x7fff: ext 1個で展開

シンボルのアドレス - データポインタ = 0x80000～0xfffff: ext 2個で展開

最適化2: データ転送命令の場合(レジスタ→メモリ)

```

例:  label:
      .skip    4

      main:
      ...
      xld.w    [label],%r0
      ...
      ret

```

-medda32指定時

展開形式

```

xld.*    [label],%r0                                (*はb、h、wのいずれか)
      ↓
pushn    %r1
ext      <シンボルの上位13ビット>
ext      <シンボルの中間13ビット>
ld.w     %r1,<シンボルの下位6ビット>
ld.*     [%r1],%rs
popn     %r1

```

シンボル(label)のアドレスをダンプファイルから取得し、extの数を決定します。

シンボルのアドレス = 0～0x1f: extなしで展開

シンボルのアドレス = 0x20～0x3fff: ext 1個で展開

シンボルのアドレス = 0x4000～0xfffff: ext 2個で展開

-medda32未指定時

展開形式1: C33 STD、C33 PEの場合

```

xld.w    [label],%r0
      ↓
ext      imm13
ext      imm13
ld.w     [%r15],%r0

```

シンボル(label)のアドレスと、%r15が表すデータポインタの値からextの数を決定します。

シンボルのアドレス - データポインタ = 0～0x1fff: ext 1個で展開

シンボルのアドレス - データポインタ = 0x2000～0xfffff: ext 2個で展開

展開形式2: S1C33401の場合

```

xld.w    [label],%r0
      ↓
ext      imm13
ext      imm13
ld.w     [%dp+imm6],%r0

```

シンボル(label)のアドレスと、%dpが表すデータポインタの値からextの数を決定します。

シンボルのアドレス - データポインタ = 0～0x7fff: ext 1個で展開

シンボルのアドレス - データポインタ = 0x80000～0xfffff: ext 2個で展開

最適化3: ビット操作命令の場合

例: main:

```

    ...
    xbtst    [SYMBOL], 1
    ...
    ret

```

-medda32指定時

展開形式

```

    xbtst    [SYMBOL], 1
      ↓
    pushn    %r0
    ext      <シンボルの上位13ビット>
    ext      <シンボルの中間13ビット>
    ld.w     %r0, <シンボルの下位6ビット>
    b*       [%r0], imm3                (*はtst、clr、set、notのいずれか)
    popn     %r0

```

シンボル(SYMBOL)のアドレスをダンプファイルから取得し、extの数を決定します。

シンボルのアドレス = 0~0x1f: extなしで展開

シンボルのアドレス = 0x20~0x3fff: ext 1個で展開

シンボルのアドレス = 0x40000~0xfffff: ext 2個で展開

-medda32未指定時

展開形式

```

    xbtst    [SYMBOL], 1
      ↓
    ext      imm13
    ext      imm13
    b*       [%r15], imm3                (*はtst、clr、set、notのいずれか)

```

シンボル(SYMBOL)のアドレスと、%r15が表すデータポインタの値からextの数を決定します。

シンボルのアドレス - データポインタ = 0~0x1fff: ext 1個で展開

シンボルのアドレス - データポインタ = 0x2000~0x3fffff: ext 2個で展開

最適化4: 相対分岐命令の場合 (scall/xcall/sjp/xjp/sjrrx/xjrrx命令)

```

例:  sub:
      ret
     main:
      ...
      xcall    sub          ← ADDR1
      ...
      ret

```

分岐先(sub)のアドレスはダンプファイルから取得します。分岐元(ADDR1)のアドレスは、mainの先頭アドレスをダンプファイルから取得し、mainからADDR1までの距離をアセンブラがカウントして算出します。分岐先アドレス - 分岐元アドレス(分岐距離)の値によりextの数を決定します。

分岐距離 = -256～254: extなしで展開

分岐距離 = -2,097,152～-257または255～2,097,150: ext 1個で展開

分岐距離 = -2,147,483,648～-2,097,153または2,097,151～2,147,483,647: ext 2個で展開

なお、削除されるext命令は"ext 0"のみとは限りません。下記の例のように、オペランド1とオペランド2のすべてのビットが1であり、かつオペランド3の最上位ビットが1のときは自動的に符号拡張されるため、ext命令は不要となり削除されます。

```

例:  L1:
      nop
      xjp  L1      ; (L1 - 分岐元のアドレス)の値を確認

```

最適化前

```

nop
ext  0x1ff8  ← オペランド1(imm10 << 3)
ext  0x1fff  ← オペランド2(imm13)
jp   0xfd   ← オペランド3(sign9 = (8:1))

```

最適化後

```

nop
jp   0x7e

```

最適化5: オペランドがシンボルを使用した計算式の場合

```

例:  sub:
      nop
      ret
     main:
      ...
      xld.w    %r0, [sub+0x2]
      ...
      ret

```

アセンブラがオペランドの式を計算し、extの数を決定します。

上記例は最適化1の展開形式が適用されます。

●注意事項

- アセンブラソースにext命令を直接記述して基本命令を拡張した場合、最適化の対象とはなりません。アセンブラソースを記述する場合、シンボルをオペランドに持つ転送命令、ビット操作命令、分岐命令にはs*またはx*拡張命令を使用してください。
- 相対分岐命令の分岐先のシンボルが他のセクションに属している場合、最適化の対象外になります。

例:

```
.text_A 0x00000000 :
{
    __START_text_A = . ;
    sub_1.o(.text)
    sub_2.o(.text)
    sub_3.o(.text)
    sub_4.o(.text)
}
__END_text_A = . ;

.text_B 0x00C00000 :
{
    __START_text_B = . ;
    main.o(.text)
}
__END_text_B = . ;
```

この例の場合、main.oとsub_X.oの間のシンボル参照や分岐を行う拡張命令は最適化されません。したがって、できるだけ、新規のセクションをリンカスクリプトファイル(.lds)に追加しないでください。

- ダンプファイル内に同じファイル名が複数記載されていた場合(同名のソースファイル(拡張子とパスを除く)が複数存在する場合)、ダンプファイルを使用したローカルシンボルの最適化は行われません。ただし、同じ名称でも大文字と小文字の構成が異なる場合、ローカルシンボルの最適化は行われます。

この状態にかかわらず、グローバルシンボルの最適化は行われます。

- medda32オプションなしでビルドする場合は、データポインタの値に関して以下の設定が必要です。以下の設定がない場合、拡張命令の最適化は行われません。

- リンカスクリプトファイルでデータポインタの値を指定します。

リンカスクリプトファイルに次の式を記述してください。

```
__dp=0xN;
```

この式は、データポインタの値を指定するのに必要です。

IDEのプロジェクトプロパティの[GNU33 リンカスクリプト設定]でも指定可能です。

- ブートルーチンで__dpの値を%r15および%dpに設定します。

__dpシンボルの値を%r15に設定してください。

S1C33401の場合は、%dpにも同じ値を設定してください。

```
xld.w  %r15, __dp
ld.w   %dp, %r15          (S1C33401の場合)
```

以降、%r15は変更しないでください。

8.9 エラー /ワーニングメッセージ

以下に、アセンブラasが出力する主なエラーおよびワーニングメッセージを示します。

表8.9.1 エラーメッセージ

エラーメッセージ	内容
Error: Unrecognized opcode: 'XXXXX'	XXXXXは未定義のオペコードです。
Error: XXXXXX: invalid operand	オペランド書式のエラーです。
Error: XXXXXX: invalid register name	使用できないレジスタを指定しています。
Error: There are too many characters of one line in assembler source file.	アセンブラソースファイルの1行の文字数(改行文字を除く)が2,047文字を超えています。
Error: Cannot allocate memory.	malloc()によるメモリ確保に失敗しました。
Error: Cannot specify plurality source files.	コマンドラインに複数のソースファイル名が指定されています。
Error: Cannot find the dump file.	-mc33_extが指定されていますが、ダンプファイル名が指定されていません。あるいは、指定されたダンプファイルが存在しません。
Error: The format of the dump file is invalid.	-mc33_extで指定されているダンプファイルの内容が不正です。
Error: There are too many characters of one line in dump file.	-mc33_extで指定されたダンプファイルの1行の文字数(改行文字を除く)が2,047文字を超えています。
Error: Cannot find the all objects' dump file.	-mc33_extが指定されていますが、全オブジェクトダンプファイル名が指定されていません。あるいは、指定されたファイルが存在しません。
Error: The format of the all objects' dump file is invalid.	-mc33_extで指定されている全オブジェクトダンプファイルの内容が不正です。

表8.9.2 ワーニングメッセージ

ワーニングメッセージ	内容
Warning: operand out of range (XXXXXX: XXX not between AAA and BBB)	オペランドで指定されている値は有効範囲外です。有効範囲内(AAA~BBB)に補正します。
Warning: Unrecognized .section attribute: want a, w, x	セクションの属性がa、w、x以外です。
Warning: Bignum truncated to AAA bytes	定数宣言(.long、.intなど)のサイズが最大値を超えています。値をAAAバイトサイズに補正します。 例) 0x100000012 → 0x12
Warning: Value XXXX truncated to AAA	定数宣言の値が最大値を超えています。値をAAAに補正します。例) .byte 0x10000012 → .byte 0xff

8.10 注意事項

- デバッガgdbでアセンブラソースレベルデバッグを行う場合には、アセンブラasのオプション--gstabsを指定し、ソース情報を付加してください。
- ソースファイル内のデバッグ情報(.stab擬似命令)は、**xgcc**および**as**以外で作成しないでください。また出力されたデバッグ情報を修正しないでください。**as**、**ld**、**gdb**の誤動作につながります。
- アセンブラ拡張命令の表記に`symbol+imm26`とありますが、実際に指定可能な値は
 $0 \leq (\text{symbol} + \text{imm26} - \text{__dp}) < 0x04000000$
 となります。
 また、ジャンプ系の拡張命令の表記で`label+imm32`がありますが、実際に指定可能な分岐範囲は、`scall`や`sjp`等の短いジャンプ命令では±2MBまで、`xcall`や`xjp`等の長いジャンプ命令では±2GBとなります。この値を超えて指定した場合、リンク時にエラーが発生します。
- アセンブラソースに`ext`命令を直接記述して基本命令を拡張した場合、最適化の対象とはなりません。アセンブラソースを記述する場合、シンボルをオペランドに持つ転送命令、ビット操作命令、分岐命令には`s*`または`x*`拡張命令を使用してください。
- 相対分岐命令の分岐先のシンボルが他のセクションに属している場合、最適化の対象外になります。

```
例: .text_A 0x00000000 :
{
    __START__text_A = . ;
    sub_1.o(.text)
    sub_2.o(.text)
    sub_3.o(.text)
    sub_4.o(.text)
}
__END__text_A = . ;

.text_B 0x00C00000 :
{
    __START__text_B = . ;
    main.o(.text)
}
__END__text_B = . ;
```

この例の場合、`main.o`と`sub_X.o`の間のシンボル参照や分岐を行う拡張命令は最適化されません。したがって、できるだけ、新規のセクションをリンクスクリプトファイル(.lds)に追加しないでください。

- ダンプファイル内に同じファイル名が複数記載されていた場合(同名のソースファイル(拡張子とパスを除く)が複数存在する場合)、ダンプファイルを使用したローカルシンボルの最適化は行われません。ただし、同じ名称でも大文字と小文字の構成が異なる場合、ローカルシンボルの最適化は行われます。この状態にかかわらず、グローバルシンボルの最適化は行われます。
- `-medda32`オプションなしでビルドする場合は、データポインタの値に関して以下の設定が必要です。以下の設定がない場合、拡張命令の最適化は行われません。

1. リンカスクリプトファイルでデータポインタの値を指定します。
 リンカスクリプトファイルに次の式を記述してください。

```
__dp=0xN;
```

この式は、データポインタの値を指定するのに必要です。

IDEのプロジェクトプロパティの[GNU33 リンカスクリプト設定]でも指定可能です。

2. ブートルーチンで`__dp`の値を`%r15`および`%dp`に設定します。

`__dp`シンボルの値を`%r15`に設定してください。

S1C33401の場合は、`%dp`にも同じ値を設定してください。

```
xld.w  %r15, __dp
ld.w   %dp, %r15          (S1C33401の場合)
```

以降、`%r15`は変更しないでください。

9 リンカ

この章ではリンカldの機能を説明します。

9.1 機能

リンカldは実行可能なオブジェクトファイルを生成するソフトウェアで、以下の機能があります。

- ライブラリを含め、複数のオブジェクトモジュールをリンクし、1つの実行可能なオブジェクトファイルを生成
- モジュール間の外部参照を解決
- リロケータブルアドレスをアブソリュートアドレスに再配置
- リンク後のオブジェクトファイルに行番号やシンボル情報などのデバッグ情報を出力
- リンクマップファイルを出力可能

このリンカはGNUリンカ(ld)をベースとしています。リンカldの詳細については、GNUリンカのドキュメントを参照してください。ドキュメントは、世界各地にあるGNUのミラーサイトからインターネット等を利用して入手可能です。

9.2 入出力ファイル

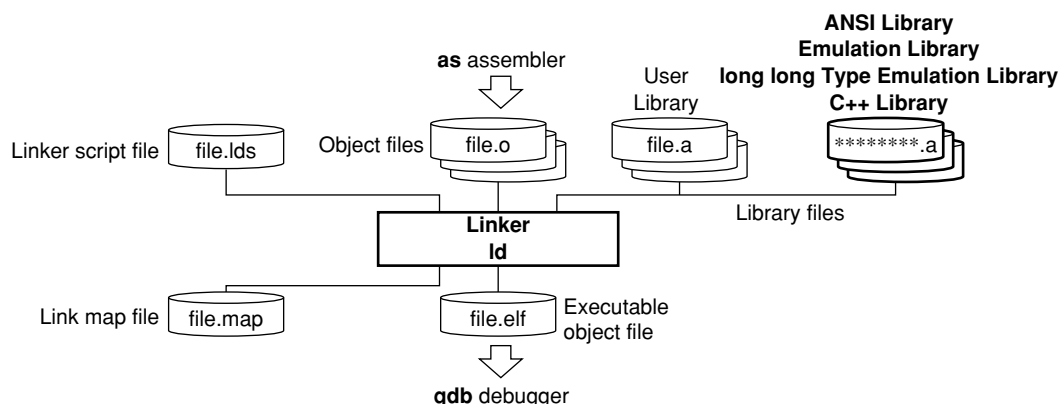


図9.2.1 フローチャート

9.2.1 入力ファイル

●オブジェクトファイル

ファイル形式: elf形式のバイナリファイル

ファイル名: <ファイル名>.o

内容: アセンブラasで生成したモジュール個々のオブジェクトファイルです。

●ライブラリファイル

ファイル形式: ライブラリ形式のバイナリファイル

ファイル名: <ファイル名>.a

内容: ANSIライブラリ、エミュレーションライブラリ、long long型エミュレーションライブラリ、C++ライブラリおよびユーザのライブラリファイルです。

注: リンカに入力するオブジェクトファイルおよびライブラリファイルは、すべて同じターゲットCPU (C33 STD、C33 PE、またはS1C33401) の指定によりコンパイルされている必要があります。

ターゲットCPUが異なるオブジェクトファイル同士をリンクすることはできません。

●リンカスクリプトファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.lds

内容: 各セクションの開始アドレスなどリンクに必要な情報を指定するファイルです。**IDE**によってリンカスクリプトファイルが作成できます。このファイルは**-T**オプションが指定された場合にリンカ**ld**に入力されます。

9.2.2 出力ファイル

●実行形式オブジェクトファイル

ファイル形式: elf形式のバイナリファイル

ファイル名: <ファイル名>.elf

内容: デバグガ**gdb**に入力可能な実行形式のオブジェクトファイルです。1つのプログラムを構成するすべてのモジュールがリンクされ、すべてのコードが配置される絶対アドレスが決定した内容となっています。デバグに必要な情報もelf形式で含まれています。**-o**オプションによる出力ファイル名の指定がない場合、オブジェクトファイルは**a.out**という名称で生成されます。

●リンクマップファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.map

内容: 各入力ファイルが各セクションのどのアドレスから配置されたかを示すマップ情報ファイルです。起動時オプションでの**-M**または**-Map**が指定された場合に出力されます。

9.3 起動方法

9.3.1 起動フォーマット

リンカldは次のコマンドにより起動します。

ld <オプション> <ファイル名>

<オプション> 9.3.2項参照

<ファイル名> リンクするオブジェクトファイルおよびライブラリファイルを指定します。

例: ld -o sample.elf boot.o sample.o ..¥lib¥libc.a ..¥lib¥libgcc.a -defsym _dp=0

9.3.2 コマンドラインオプション

リンカldはGNUリンカ標準のコマンドオプションを受け付けます。

ここでは、よく使用するオプションのみを説明しますので、その他のオプションも含め、詳細についてはGNUリンカのマニュアルを参照してください。

-o <ファイル名>

機能: 出力ファイル名の指定

説明: リンカldが出力するオブジェクトファイルの名称を指定します。

デフォルト: a.outという名称のファイルを生成します。

-T <リンクスクリプトファイル名>

機能: リンカスクリプトファイルの読み込み

説明: リンカコスクリプトファイルから配置情報を入力する場合に指定します。

デフォルト: デフォルトリンクスクリプト(9.4.2項参照)が使用されます。

-M

-Map <ファイル名>

機能: リンクマップファイルの出力

説明: -Mオプションを指定するとリンクマップ情報がstdioデバイスに出力されます。

-Mapオプションはリンクマップ情報を指定のファイルに出力します。

デフォルト: リンクマップ情報は出力されません。

-N

機能: データセグメントのアライメントチェックを禁止

説明: -Nオプションを指定すると、データセグメントのアライメントチェックを行いません。

通常は、このオプションを指定してください。("9.6 注意事項"参照)

デフォルト: リンカはデータセグメントのアライメントチェックを行います。

コマンドラインにオプションを入力する場合、オプションの前後には1個以上のスペースが必要です。

例: ld -o sample.elf -T sample.lds -N boot.o sample.o ..¥lib¥libc.a

9.4 リンク処理

9.4.1 データエリアポインタの指定

プログラムがデータエリアを使用している場合、リンク時にデータエリアポインタ(データエリア開始アドレス)を指定する必要があります。データエリアポインタはリンクスクリプトファイルに記述して読み込ませるか、リンカldを起動するコマンドラインにおいて-defsymオプションで指定します。

例: 次のようにデータエリアポインタを指定する場合

__dp(デフォルトデータエリアポインタ): 0xc00000

-defsymオプションによる指定

```
ld -defsym __dp=0xc00000 -o sample.elf sample.o
```

リンクスクリプトファイルによる指定(sample.lds)

以下の定義をリンクスクリプトファイルに記述します。

```
SECTIONS
{
    __dp = 0xc00000;
    :
```

リンカldを起動するコマンドラインでは、-Tオプションでリンクスクリプトファイルを指定します。

```
ld -o sample.elf sample.o -T sample.lds
```

9.4.2 デフォルトリンクスクリプト

-Tオプション未指定時のデフォルトリンクスクリプト

-Tオプションが指定されないと、リンカldは下記のリンクスクリプトを使用してリンクの処理をします。-defsymオプションでデフォルトデータエリアポインタ(__dp)が指定された場合は、デフォルトリンクスクリプトの__dp設定は無効となります。

```
OUTPUT_FORMAT("elf32-c33", "elf32-c33",
               "elf32-c33")
```

```
OUTPUT_ARCH(c33)
```

```
SEARCH_DIR(.);
```

```
SECTIONS
```

```
{
    __dp = 0x0;

    . = 0x0;
    .bss : { *(.bss) }
    .data : { *(.data) }

    . = 0xc00000;
    .text : { *(.text) }
    .rodata : { *(.rodata) }
}
```

このスクリプトでは、データが.bss、.dataの順にアドレス0から配置され、プログラムコードと定数データはアドレス0xc00000から配置されます。

図9.4.2.1にリンク後のメモリマップを示します。

このスクリプトはVMAがLMAと異なる場合を考慮していませんので、初期値を持つデータが置かれる.dataセクションをアプリケーションが使用している場合には使用することができません。

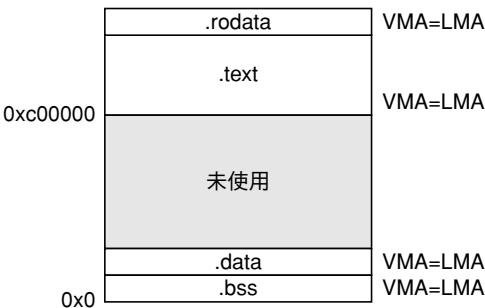


図9.4.2.1 デフォルトリンクスクリプトによるメモリマップ

IDEのデフォルト設定で作成されるリンカスクリプト

IDEのデフォルト設定では次のようなリンカスクリプトファイルが作成されます。

例: boot.oとmain.oをリンクする場合

```
/* Linker Script file generated by Gnu33 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

SECTIONS
{
    /* data pointer symbols */
    __dp = 0x00000000;

    /* stack pointer symbols */
    __START_stack = 0x00004000;

    /* location counter */
    . = 0x0;

    /* section information */
    .bss 0x00000000 :
    {
        __START_bss = . ;
        boot.o(.bss)
        main.o(.bss)
        C:/EPSON/gnu33/lib/std/libstdio.a(.bss)
        C:/EPSON/gnu33/lib/std/libstdc++.a(.bss)
        C:/EPSON/gnu33/lib/std/libgcc2.a(.bss)
        C:/EPSON/gnu33/lib/std/libc.a(.bss)
        C:/EPSON/gnu33/lib/std/libgcc.a(.bss)
    }
    __END_bss = . ;

    .comm __END_bss :
    {
        __START_comm = . ;
        boot.o(.comm)
        main.o(.comm)
        C:/EPSON/gnu33/lib/std/libstdio.a(.comm)
        C:/EPSON/gnu33/lib/std/libstdc++.a(.comm)
        C:/EPSON/gnu33/lib/std/libgcc2.a(.comm)
        C:/EPSON/gnu33/lib/std/libc.a(.comm)
        C:/EPSON/gnu33/lib/std/libgcc.a(.comm)
    }
    __END_comm = . ;

    .data __END_comm : AT( __END_gcc_except_table )
    {
        __START_data = . ;
        boot.o(.data)
        main.o(.data)
        C:/EPSON/gnu33/lib/std/libstdio.a(.data)
        C:/EPSON/gnu33/lib/std/libstdc++.a(.data)
        C:/EPSON/gnu33/lib/std/libgcc2.a(.data)
        C:/EPSON/gnu33/lib/std/libc.a(.data)
        C:/EPSON/gnu33/lib/std/libgcc.a(.data)
    }
    __END_data = . ;

    .vector 0x00C00000 :
    {
        __START_vector = . ;
        boot.o(.rodata)
    }
    __END_vector = . ;

    .text __END_vector :
    {
        __START_text = . ;
        boot.o(.text)
```

```

main.o(.text)
C:/EPSON/gnu33/lib/std/libstdio.a(.text)
C:/EPSON/gnu33/lib/std/libstdc++.a(.text)
C:/EPSON/gnu33/lib/std/libgcc2.a(.text)
C:/EPSON/gnu33/lib/std/libc.a(.text)
C:/EPSON/gnu33/lib/std/libgcc.a(.text)
}
__END_text = . ;

.rodata __END_text :
{
    __START_rodata = . ;
main.o(.rodata)
C:/EPSON/gnu33/lib/std/libstdio.a(.rodata)
C:/EPSON/gnu33/lib/std/libstdc++.a(.rodata)
C:/EPSON/gnu33/lib/std/libgcc2.a(.rodata)
C:/EPSON/gnu33/lib/std/libc.a(.rodata)
C:/EPSON/gnu33/lib/std/libgcc.a(.rodata)
}
__END_rodata = . ;

.ctors __END_rodata :
{
    . = ALIGN(4);
    __START_ctors = . ;
boot.o(.ctors)
main.o(.ctors)
C:/EPSON/gnu33/lib/std/libstdio.a(.ctors)
C:/EPSON/gnu33/lib/std/libstdc++.a(.ctors)
C:/EPSON/gnu33/lib/std/libgcc2.a(.ctors)
C:/EPSON/gnu33/lib/std/libc.a(.ctors)
C:/EPSON/gnu33/lib/std/libgcc.a(.ctors)
}
__END_ctors = . ;

.dtors __END_ctors :
{
    . = ALIGN(4);
    __START_dtors = . ;
boot.o(.dtors)
main.o(.dtors)
C:/EPSON/gnu33/lib/std/libstdio.a(.dtors)
C:/EPSON/gnu33/lib/std/libstdc++.a(.dtors)
C:/EPSON/gnu33/lib/std/libgcc2.a(.dtors)
C:/EPSON/gnu33/lib/std/libc.a(.dtors)
C:/EPSON/gnu33/lib/std/libgcc.a(.dtors)
}
__END_dtors = . ;

.gcc_except_table __END_dtors :
{
    __START_gcc_except_table = . ;
boot.o(.gcc_except_table)
main.o(.gcc_except_table)
C:/EPSON/gnu33/lib/std/libstdio.a(.gcc_except_table)
C:/EPSON/gnu33/lib/std/libstdc++.a(.gcc_except_table)
C:/EPSON/gnu33/lib/std/libgcc2.a(.gcc_except_table)
C:/EPSON/gnu33/lib/std/libc.a(.gcc_except_table)
C:/EPSON/gnu33/lib/std/libgcc.a(.gcc_except_table)
}
__END_gcc_except_table = . ;

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
}

```

IDEはリンクスクリプトファイルの編集機能を持っていますので、ユーザのアプリケーションに合わせて修正したうえで使用してください。**IDE**を使用したリンクスクリプトファイルの編集方法については、"5.7.7 リンカスクリプトの編集"を参照してください。

9.4.3 リンク例

例1) デフォルトデータエリアを使用する場合

デフォルトデータエリアに対応した最小構成のリンカスクリプト例を以下に示します。

```

OUTPUT_FORMAT("elf32-c33", "elf32-c33",
              "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);
SECTIONS
{
    /* data pointer symbol By GWB33 */
    __dp = 0x00000000;          ...1

    . = 0x0;

    .bss :                      ...2
    {
        __START_bss = . ;
        *(.bss) ;
    }
    __END_bss = . ;

    .data __END_bss : AT( __END_rodata ) ...3
    {
        __START_data = . ;
        *(.data) ;
    }
    __END_data = . ;
    __START_data_lma = LOADADDR( .data );

    .text 0x00c00000 :          ...4
    {
        __START_text = . ;
        *(.text) ;
    }
    __END_text = . ;

    .rodata __END_text :        ...5
    {
        __START_rodata = . ;
        *(.rodata) ;
    }
    __END_rodata = . ;
}

```

1. デフォルトデータエリアポインタ(__dp)を0x0に設定します。
2. 入力ファイルのすべての.bssセクションをアドレス0x0からはじまる.bss出力セクションに配置します。このセクションは実際のコードを持ちませんので、LMAを指定する必要はありません。
3. .bssセクションの直後に続くメモリ空間は、入力ファイルの.dataセクションに割り当てられます。初期値(実コード)は、.rodataセクションに続いて配置されます。このLMAはAT文で指定します。__END_rodataは.rodataコマンドで定義されたシンボルで、.rodataセクション直後のロケーションカウンタ値を示します。

__START_data_lmaは.dataセクション(LMA)の開始アドレスを参照するために定義されたシンボルです。__START_dataと__END_dataは.dataセクション(VMA)の開始/終了アドレスを参照するために定義されたシンボルです。これらのシンボルは、次のように、初期化ルーチンでLMAからVMAに初期値をコピーするのに使用します。

```
xld.w  %r0, __START_data_lma
xld.w  %r1, __START_data
xld.w  %r2, __END_data

loop:
  ld.b  %r3, [%r0]+
  ld.b  [%r1]+, %r3
  cmp   %r1, %r2
  jrne  loop
```

注: リンカスクリプトで生成したシンボルをC/C++コード内で参照すると、シンボル値によってはリンカでエラーとなる場合があります。

```
extern char __START_data_lma, __START_data, __END_data;

char *src = &__START_data_lma;      /* 正常に値を取得できません。 */
char *dst = &__START_data;          /* 正常に値を取得できません。 */
```

この例の__START_data_lmaのアドレス取得部は、以下のように生成されます。

```
ext doff_hi(__START_data_lma)      ; 上位13ビット
ext doff_lo(__START_data_lma)      ; 下位13ビット
ld.b %r4, [%r15]
```

__START_data_lmaのシンボル値(この場合はアドレス)が0x3fffffを超えている場合、オーバーフローが生じ、アドレス(32ビット)を取得できません。この制約はデータエリアをサポートするS5U1C33001Cに特有のもので、GNU i386コンパイラ等ではこの問題は発生しません。

4. 入力ファイルのすべての.textセクションはアドレス0xc00000から配置されます。
5. 入力ファイルのすべての.rodataセクションは、.textセクションの直後に配置されます。.dataセクションの実コードをこの直後に配置するため、LMAを指定する__END_rodataを定義しています。

図9.4.3.1にこのスクリプト例で構成されるメモリマップを示します。

```
ld -o sample.elf file1.o file2.o -T default.lds
```

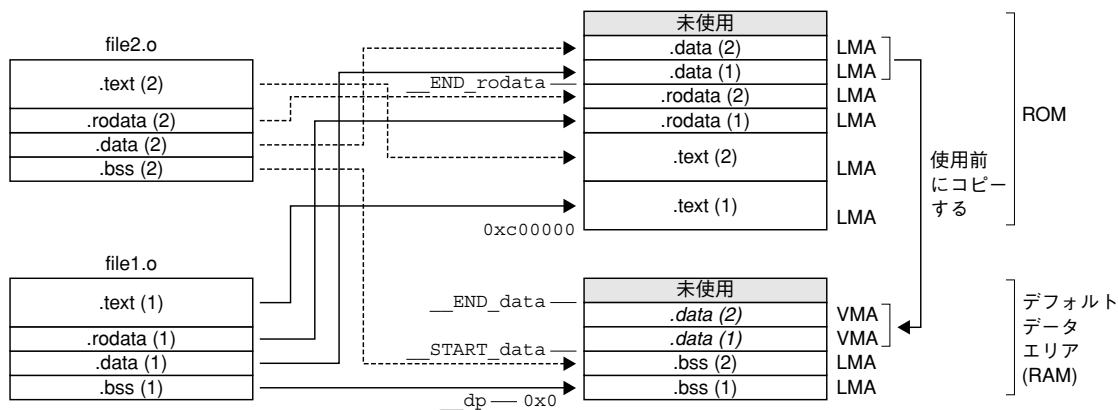


図9.4.3.1 メモリマップ(例1)

例2)仮想・共用セクションを使用する場合

仮想セクションおよび共用セクションを使用する場合のリンカスクリプト例を以下に示します。

```

OUTPUT_FORMAT("elf32-c33", "elf32-c33",
              "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);
SECTIONS
{
    /* data pointer symbol */
    __dp = 0x0;

    /* section information */
    . = 0x0;

    .bss 0x00000000 :
    {
        __START_bss = . ;
        *(.bss) ;
    }
    __END_bss = . ;

    .data __END_bss : AT( __END_rodata )
    {
        __START_data = . ;
        *(.data) ;
    }
    __END_data = . ;
    __START_data_lma = LOADADDR( .data );

    .text_foo1 __END_data : AT( __START_data_lma+SIZEOF( .data )
    {
        __START_text_foo1 = . ;
        foo1.o(.text) ;
    }
    __END_text_foo1 = . ;
    __START_text_foo1_lma = LOADADDR( .text_foo1 );

    .text_foo2 __END_data : AT( __START_text_foo1_lma+SIZEOF( .text_foo1 )
    {
        __START_text_foo2 = . ;
        foo2.o(.text) ;
    }
    __END_text_foo2 = . ;
    __START_text_foo2_lma = LOADADDR( .text_foo2 );

    .text_foo3 __END_data : AT( __START_text_foo2_lma+SIZEOF( .text_foo2 )
    {
        __START_text_foo3 = . ;
        foo3.o(.text) ;
    }
    __END_text_foo3 = . ;
    __START_text_foo3_lma = LOADADDR( .text_foo3 );

    .text 0x00600000 :
    {
        __START_text = . ;
        *(.text) ;
    }
    __END_text = . ;

    .rodata __END_text :
    {
        __START_rodata = . ;
        *(.rodata) ;
    }
    __END_rodata = . ;
}

```

図9.4.3.2にセクションの配置を示します。

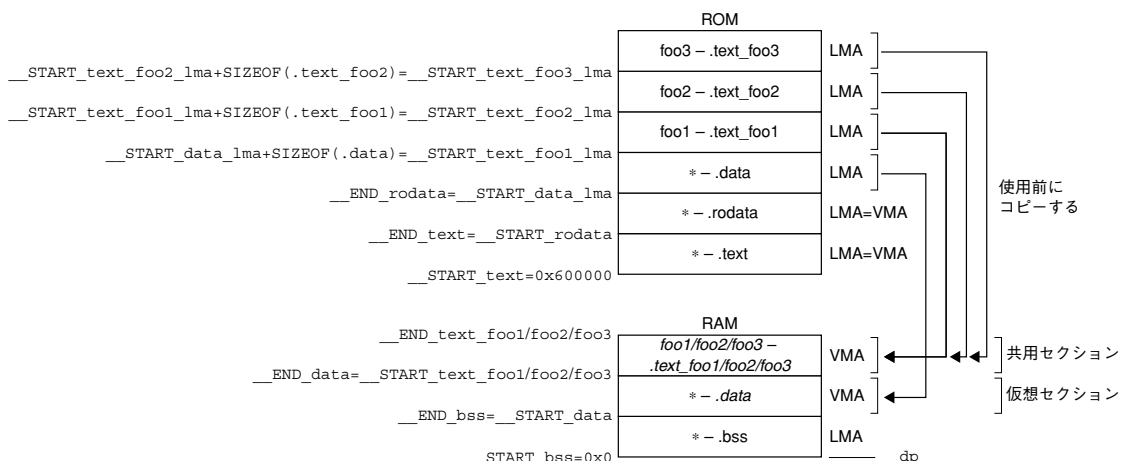


図9.4.3.2 メモリマップ(例2)

.dataセクションはROM上のLMAに実体を持ち、RAM上のVMA(.bssセクションの直後)にコピーして使用するよう設定しています。RAM上の.dataセクション(VMA)はプログラムの実行開始時にはデータが存在しない仮想セクションと見ることができます。初期値を持つ変数などは、このように使用する必要があります。この例の場合、すべてのファイルの.dataセクションが1つにまとめられています。

.text_foo1はfoo1.oファイルの.textセクションで、ROM上のLMAに実コードを持ち、RAM上のVMAで実行するよう設定しています。.text_foo2セクションと.text_foo3セクションも同様ですが、これら3つのセクションは同じVMAが設定されています。同じ実行アドレスを共用しているため、RAM上の.text_foo1/2/3セクションは共用セクションと見ることができます。プログラムを高速実行させるためのキャッシュのような使い方はこのように実現します。これら3つ以外のファイルの.textセクションは0x600000～の.textセクションに配置され、そのままROM上で実行されます。

9.5 エラー /ワーニングメッセージ

エラーおよびワーニングメッセージは標準出力(stdout)に表示/出力されます。

リンカldでは、以下に示すエラーおよびワーニングメッセージがGNUリンカの標準エラーメッセージに追加されています。

表9.5.1 エラーメッセージ

エラーメッセージ	内容
Error: Default Data area pointer value is larger than symbol address value.	__dp(デフォルトデータエリアポインタ)の値が定義されたシンボルのアドレス値を超えています。
Error: The offset value of a symbol is over 64MB. (default data area)	__dp(デフォルトデータエリアポインタ)からシンボルまでのオフセットが64MBの範囲を超えています。
Error: The offset value of a symbol is over 512KB. (default data area)	__dp(デフォルトデータエリアポインタ)からシンボルまでのオフセットが512KBの範囲を超えています。
Error: The offset value of a symbol is over 64byte. (default data area)	__dp(デフォルトデータエリアポインタ)からシンボルまでのオフセットが64バイトの範囲を超えています。
Error: Cannot link [STD PE ADV] object <objectfile> included from <archivefile> with [STD PE ADV] object <first objectfile>	<archivefile>に含まれている<objectfile>はターゲットCPUが異なるオブジェクトであるため、<first objectfile>とはリンクできません。
Error: Input object file <objectfile> [included from <archivefile>] is not for C33.	<archivefile>に含まれている<objectfile>はC33のオブジェクトではありません。

表9.5.2 ワーニングメッセージ

ワーニングメッセージ	内容
Warning: __dp symbol cannot be referred to.	1. doff_hi、dpoff_h、dpoff_m、dpoff_lまたはdoff_lo擬似オペランドが使用されていますが、__dpが指定されていません。 2. オペランドに[symbol+imm]が使用されていますが、__dpが指定されていません。 (オペランド[symbol+imm]を持つ命令は__dpの値を参照するコードに展開されます。) __dpは0x0として処理されます。

9.6 注意事項

- リンカに入力するオブジェクトファイルおよびライブラリファイルは、すべて同じターゲットCPU (C33 STD、C33 PE、またはS1C33401)の指定によりコンパイルされている必要があります。ターゲットCPUが異なるオブジェクトファイル同士をリンクすることはできません。

- リンカ実行時、以下のようなエラーメッセージが表示されることがあります。

```
ld: test.elf: Not enough room for program header, try linking with -N
```

これは、リンカがデータセグメントのアライメントチェックを行っているために発生します。このエラーは、リンカldのオプションに-Nを指定することにより回避できますので、通常は-Nを指定してリンクを行ってください。(IDEの作成するmakeファイルでは、-Nオプションが付加されています。)

- ldのコマンドラインで指定するオブジェクトファイル名とリンカスクリプトに記述する同じオブジェクトファイル名は大文字、小文字も含め、完全に一致させてください。大文字、小文字のみの違いでも、異なるファイルと見なされます。

例:

コマンドライン

```
ld -T sample.lds -o sample.elf prg1.o prg2.o
```

リンカスクリプトファイル(sample.lds)

```

:
__dp = 0;

.text 0xc00000:
{
  PRG1.o (.text) ← 小文字のprg1.oに変更する必要があります。
  prg2.o (.text)
}

:

```

- サイズの異なる同じ関数名のファイルをリンクしたとき、以下のメッセージが表示されます。

```
Warning: size of symbol 'AAA' changed from BBB to CCC in DDD.o
```

AAA: 重複する関数名

BBB, CCC: 関数のサイズ

DDD: 対象のファイル名

サイズが一致する場合はこのメッセージは表示されません。ファイルをリンクする場合は、同じ関数名は指定しないようにしてください。特に、ライブラリファイル(*.a)に含まれている関数名と同じ関数名を定義しないように注意してください。

- 下記の例のようにリンカスクリプトファイルで"*"と個別のオブジェクトファイル指定を混在すると、正しくリンクされないことがあります。個別のオブジェクトファイル指定が不要な場合は"*"のみを使用し、それ以外はすべて個別にオブジェクトファイルを指定するようにしてください。

例:

```

.text 0x00600000 :
{
  *(.text) ;
}
.text2 :
{
  main.o(.text) ;
}

```

10 デバッガ

この章ではデバッガgdbの操作方法を説明します。

10.1 特長

デバッガgdbはリンカによって生成したelf形式のオブジェクトファイルを読み込み、プログラムのデバッグを行うためのソフトウェアです。

以下の特長と機能を持ちます。

- 統合開発環境(IDE)のデバッグ機能でデバッグ
- マルチウィンドウにより各種のデータを一度に参照可能
- S5U1C33000H、S5U1C33001H、デバッグモニタを使用したデバッグのほかに、パソコン上でデバッグ可能なソフトウェアシミュレータ機能を搭載
- C/C++ソースコードおよびアセンブリソースコードに対応したソースレベルデバッグ機能
- プログラムの連続実行とC/C++ソースレベル、アセンブラレベルのステップ実行機能
- ハードウェア、ソフトウェアPCブレイク機能、およびメモリアクセス条件付きのデータブレイク機能
- 時間とサイクル数による実行時間の計測機能
- 保存が可能なトレース機能
- コマンドファイルによるコマンドの自動実行機能
- デバッガ上で入出力の評価が可能なシミュレーテッドI/O機能
- S5U1C33001Hのフラッシュライタ機能に対応

10.2 入出力ファイル

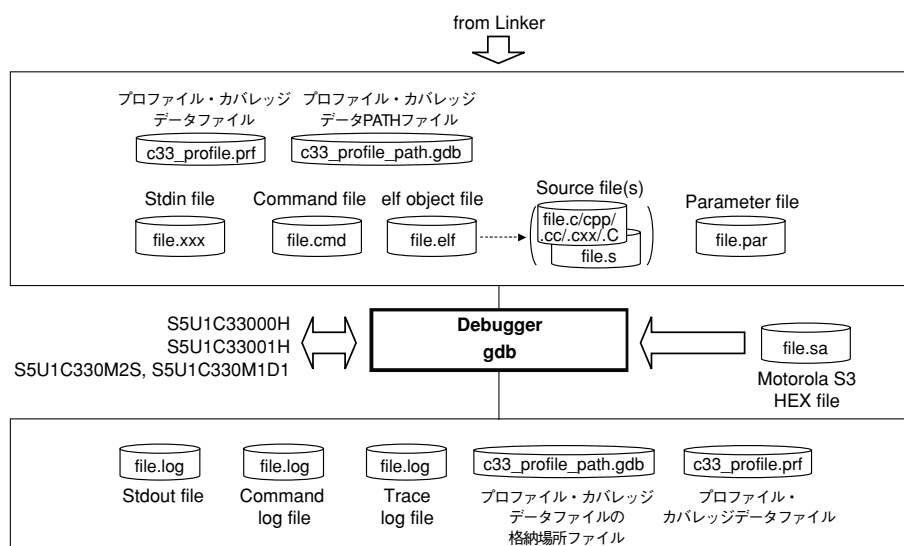


図10.2.1 フローチャート

10.2.1 入力ファイル

●パラメータファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.par

内容: デバッグのメモリマップ情報を設定する内容が記録されたファイルです。IDEの[プロジェクト]>[プロパティ]>[GNU33 パラメータ設定]ダイアログによってパラメータファイルを作成することができます。パラメータファイルの詳細については、"10.8 パラメータファイル"を参照してください。

●オブジェクトファイル

ファイル形式: elf形式のバイナリファイル

ファイル名: <ファイル名>.elf

内容: リンカldで生成したelf形式のアブソリュートオブジェクトファイルです。fileおよびloadコマンドにより読み込みます。デバッグ情報を含んだオブジェクトファイルを読み込むことで、ソース表示およびシンボリックデバッグが行えます。

●ソースファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.`c`/`.cpp`/`.cc`/`.cxx`/`.C` (C/C++ソース)

<ファイル名>.`s` (アセンブリソース)

内容: 上記オブジェクトファイルのソースファイルで、ソース表示を行うときに読み込まれます。

●コマンドファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.cmd

内容: 連続して実行させるデバッグコマンドを記述したファイルです。頻繁に使用する一連のコマンドを書き込んでおくことで、キーボードからのコマンド入力の手間を省くことができます。このファイルは起動オプション-xまたはsourceコマンドにより読み込み、実行させます。

●ROMデータHEXファイル

ファイル形式: モトローラS3形式HEXファイル

ファイル名: <ファイル名>.sa

内容: デバッグ情報を含まないため、ソースレベルデバッグは行えません。フラッシュライター機能を使用する場合、モトローラS3形式のファイルが必要です。c33 fwlpまたはc33 fwldコマンドにより読み込みます。

この形式のファイルは、リンカldが生成したelf形式ファイルをファイル形式変換ツールobjcopyで変換して作成可能です。

objcopy書式) objcopy -I elf32-little -O srec --srec-forceS3 <ファイル名>.elf <ファイル名>.sa

例) objcopy -I elf32-little -O srec --srec-forceS3 sample.elf sample.sa

●入力シミュレーション用データファイル

ファイル形式: テキストファイル

ファイル名: 任意のファイル名

内容: シミュレーテッド入力機能によりデバッグに読み込まれるファイルです。

c33 stdinコマンドにより指定します。

●プロファイル・カバレッジデータファイル

ファイル形式: バイナリ形式

ファイル名: c33_profile.gdb

内容: c33 profile または、c33 coverageコマンド実行時に作成される計測結果データです。

●プロファイル・カバレッジデータパスファイル

ファイル形式: テキスト形式

ファイル名: `c33_profile_path.gdb`

内容: プロファイル・カバレッジデータファイル(`c33_profile.prf`)の存在するパスです。
プロファイルウィンドウとカバレッジウィンドウが参照します。

10.2.2 出力ファイル

●トレース情報ファイル

ファイル形式: テキストファイル

ファイル名: 任意のファイル名

内容: トレース結果を出力したファイルです。`c33 tf`、`c33 otf`または`c33 tm`コマンドで指定して出力します。

●ログファイル

ファイル形式: テキストファイル

ファイル名: 任意のファイル名

内容: 実行したコマンドと実行結果が出力されます。`c33 log`コマンドで指定して出力します。

●出力シミュレーションデータファイル

ファイル形式: テキストファイル

ファイル名: 任意のファイル名

内容: シミュレーテッド出力機能により作成されるファイルです。
`c33 stdout`コマンドにより指定します。

●プロファイル・カバレッジデータファイル

ファイル形式: バイナリ形式

ファイル名: `c33_profile.prf`

内容: プロファイル・カバレッジの計測結果が記載されたファイルです。
プロファイルウィンドウ、カバレッジウィンドウが使用します。

●プロファイル・カバレッジデータパスファイル

ファイル形式: テキスト形式

ファイル名: `c33_profile_path.gdb`

内容: プロファイル・カバレッジデータファイル(`c33_profile.prf`)のパスが記載されたファイルです。プロファイルウィンドウとカバレッジウィンドウが使用します。

10.3 起動方法

10.3.1 起動フォーマット

●コマンドラインの一般形

`gdb [<起動時オプション>]`

[]は省略可能なことを示します。

●IDEの操作

[プロジェクト]>[プロパティ]>[GNU33 GDBコマンド]ダイアログボックスで必要に応じて起動コマンドを設定後、[実行]>[デバッグの構成...]で表示されるデバッグ開始/起動構成ダイアログで[GNU33 デバッグ]>[GDB Debugger for <プロジェクト名>]を選択し[デバッグ]ボタンをクリックします。

デバッグ開始/起動構成ダイアログボックスは、ツールバーの[デバッグ]ボタンのメニューからも開くことができます。

デバッグ開始/起動構成ダイアログボックスの詳細については、「5.8.3 デバッグの起動方法」の「●起動構成ダイアログ」を参照してください。

●注意事項

- "target icd usb2"コマンドを使用して2台目のS5U1C33001Hを接続する場合を除き、デバッガを二重起動させないでください。二重起動させた場合、正常に動作しないことがあります。

10.3.2 起動時オプション

デバッガには6種類の起動時オプションが用意されています。IDE上での選択方法については、「5.8.3 デバッガの起動方法」を参照してください。

`--command=<コマンドファイル名>`

`-x <コマンドファイル名>`

機能: コマンドファイルの指定

説明: このオプションを指定すると、デバッガは起動時に指定のコマンドファイルを読み込んで実行します。

`--c33_cmw=<ウェイト秒数>`

機能: コマンドファイル内のコマンド実行間隔の指定

説明: `-x/--command`オプション、または`source`コマンドによるコマンドファイルの実行時に、各コマンド間に指定秒数の待ち時間を挿入します。指定可能な秒数は1～256で、それ以外の数値を指定した場合は1秒の待ち時間となります。

`--cd=<ディレクトリパス文字列>`

機能: カレントディレクトリの変更

説明: このオプションを指定すると、デバッガは起動時に指定のパスをカレントディレクトリに設定します。省略時は`gdbtk.ini`ファイルに記述されたディレクトリ(`gdbtk.ini`ファイルがなければ`gdb.exe`が存在するディレクトリ)に設定されます。

`--directory=<ディレクトリパス文字列>`

機能: ソースファイルディレクトリの変更

説明: このオプションを使用して、ソースファイルのあるディレクトリを指定できます。
このオプションは複数指定することができます。

`--double_starting`

機能: 二重起動の許可

説明: このオプションを指定すると、デフォルトで禁止されているデバッガの二重起動が許可されます。2台のS5U1C33001Hを使用してデバッグを行う場合に、デバッガはこのオプションを指定して起動させます。それ以外では、デバッガを二重起動させないでください。

コマンドラインにオプションを入力する場合、オプションの前後には1個以上のスペースが必要です。

例: `c:\¥EPSON¥gnu33¥gdb -x sample.cmd --cd=/cygdrive/d/test/sample`

10.3.3 終了方法

●[Console]ビューの操作

`quit(q)` コマンド入力によって終了します。

(gdb)

`q`

●IDEの操作

以下のいずれかの方法でデバッグを終了することができます。

デバッグ終了後、[デバッグ]ビューの表示が終了状態に変わります。

- [実行]メニューから[終了]を選択します。
- [デバッグ]ビューの[終了]ボタンをクリックします。
- [コンソール]ビューの[終了]ボタンをクリックします。
- [デバッグ]ビューのコンテキストメニューから[終了]を選択します。

[コンソール]ビュー、[デバッグ]ビューについては、各ビューの項目を参照してください。

注: S5U1C33001Hを使用してデバッグを行っている場合は、S5U1C33001Hの電源を切る前に必ずデバッグを終了させてください。デバッグを実行中にS5U1C33001Hの電源を切ると再接続ができなくなることがあります。この場合、パーソナルコンピュータの再起動が必要です。

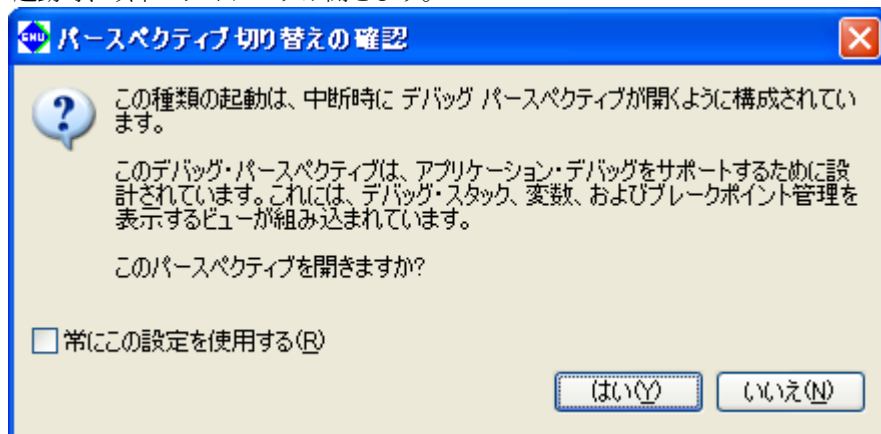
10.4 ウィンドウ

ここでは、デバッガで使用するビューの種類を説明します。

10.4.1 デバッグパースペクティブ

10.4.1.1 デバッグパースペクティブの切り替え

デバッガの起動時、以下のダイアログが開きます。



[はい]： 自動的にデバッグパースペクティブに切り替わります。

[いいえ]： パースペクティブは切り替わりません。

[常にこの設定を使用する]：

常に現在の設定でデバッグパースペクティブを開きます。

次回以降このダイアログを表示しないで、自動的にデバッグパースペクティブに切り替わります。

10.4.1.2 デバッグパースペクティブの構成

デバッグパースペクティブはデバッグに使用する各ビューを画面上に開きます。デフォルト設定では以下のビューが開きます。

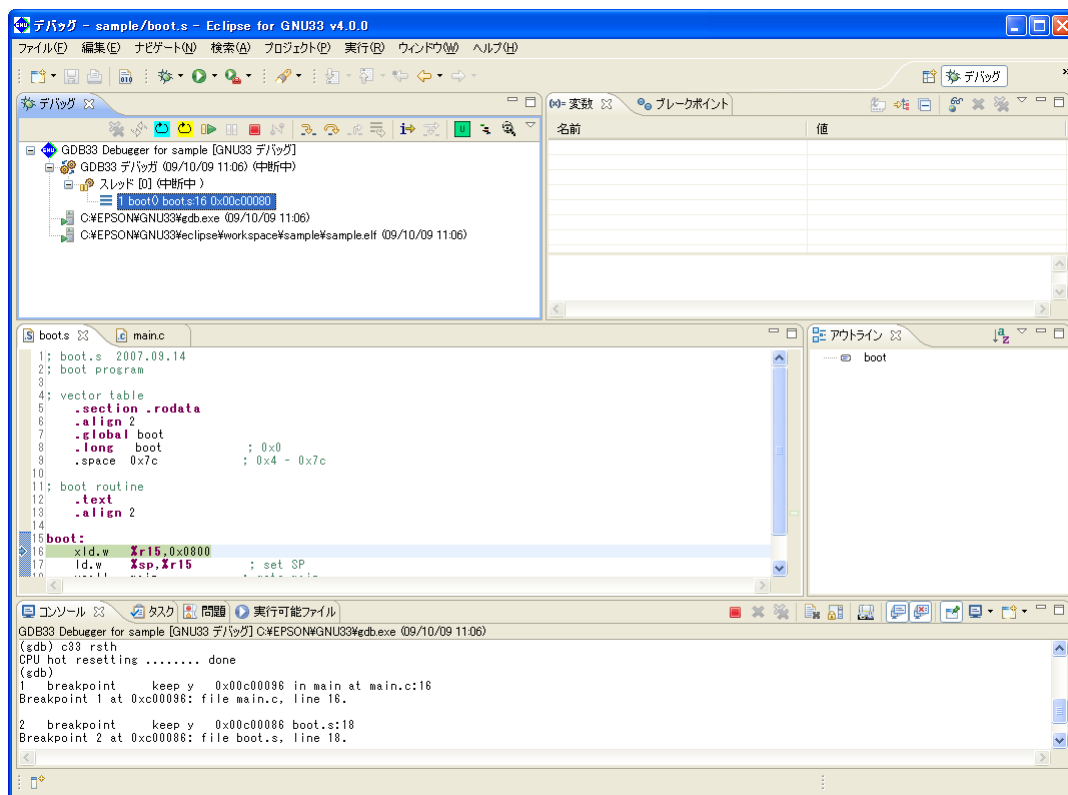


表10.4.1.2.1 デバッグパースペクティブの各ビュー

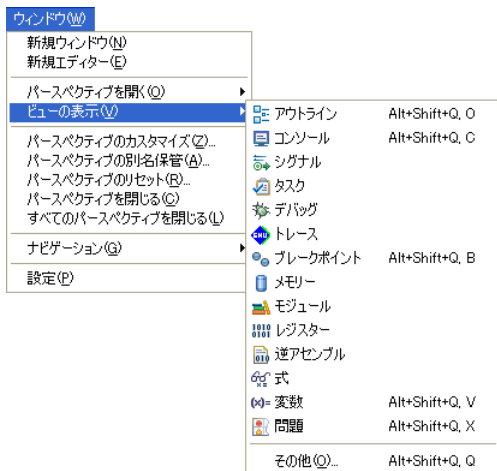
ビュー	機能
デバッグ	デバッグ時の操作画面です。 デバッグ中のプログラムのステップ実行・停止・再開を行います。 デバッグ中のプログラムの状態およびスタックフレームを表示します。
ソース	実行する行をハイライト表示します。 ブレークポイントの設定/解除を行います。
コンソール	デバッガにコマンド実行と実行結果を表示するためのコンソールです。
変数	ローカル変数を表示します。
アウトライン	[ソース]エディターで表示中のソースの構造(変数/関数)を表示します。
ブレークポイント	ブレークポイントの一覧を表示します。
レジスター	レジスタの値を表示します。
メモリー	メモリー領域を表示します。

注: 以下のビューは、デバッグ終了時に自動的に閉じられます。次回デバッグ開始時には自動的に開きます。(ただし、デバッグ中に閉じる操作をした場合は自動的に開きません。)

- ・[メモリー]ビュー
- ・[レジスター]ビュー

10.4.1.3 ビューを開く/閉じる

上記以外のビューを開く場合や、閉じてしまったビューを開く場合は、[ウィンドウ]>[ビューの表示]から開くことができます。



デバッグ時に使用できるビューは以下の通りです。

以下のビュー以外はサポートしていません。

表10.4.1.3.1 [ウィンドウ]>[ビューの表示]メニュー

デバッグ時に使用できるビュー	機能
ブレークポイント	ブレークポイントの一覧
コンソール	GDBへのコマンド入力およびSimulatedI/O出力
デバッグ	デバッグの開始・終了、実行・停止
逆アセンブル	逆アセンブル表示
式	監視式
メモリー	メモリ
レジスター	レジスタ
変数	ローカル変数
トレース	PCトレース

各ビューは、タブのxボタンから閉じることができます。

これらのビューは、すべて移動、拡大縮小、最小化、最大化の操作に対応しています。

10.4.1.4 パースペクティブのカスタマイズ

各ビューは、使いやすいように配置を変えることができます。

配置を変えるには、各ビューのタブ部分をお好みの位置までドラッグします。

不要なビューを閉じてしまうこともできます。

各ビューの配置は、IDEを閉じたときに記憶され、次回起動したとき復元されます。

パースペクティブの設定を出荷時の状態に戻したい場合は、[ウィンドウ]>[パースペクティブのリセット...]から行うことができます。

10.4.1.5 メニュー / ツールバー

デバッグパースペクティブで使用するメニュー / ツールバーについて説明します。

●[実行] メニュー

実行(E)	
● ブレークポイントの切り替え(K)	Ctrl+Shift+B
● 行ブレークポイントの切り替え(L)	
● メソッド・ブレークポイントの切り替え(M)	
🔍 監視ポイントの切り替え(W)	
🚫 すべてのブレークポイントをスキップ(K)	
🗑️ すべてのブレークポイントを除去(U)	
🔄 コールリセット	
🔥 ホットリセット	
📄 ユーザコマンド	
🔄 再開	
🔄 シグナルなしで再開	
🔍 指定行で再開(N) (C/C++)	
▶▶▶ 再開(M)	F8
⏸ 中断(S)	
🛑 終了(T)	Ctrl+F2
🔍 ステップイン(P)	F5
🔍 ステップ・オーバー(O)	F6
🔍 ステップ・リターン(R)	F7
➡ 指定行まで実行(L)	Ctrl+R
🔍 ステップ・フィルターの使用(E)	Shift+F5
🔍 前回の起動を実行(R)	Ctrl+F11
🔍 前回の起動をデバッグ(D)	F11
履歴の実行(T)	▶
実行(S)	▶
実行構成(N)...	
履歴のデバッグ(H)	▶
デバッグ(G)	▶
デバッグの構成(B)...	
🔍 外部ツール(E)	▶

表10.4.1.5.1 ツールバー [実行]メニュー

メニュー	機能
ブレークポイントの切り替え	カーソル位置に行ブレークポイントを設定します。 ([ソース]エディターにカーソルがあるときのみ)
行ブレークポイントの切り替え	カーソル位置に行ブレークポイントを設定します。 ([ソース]エディターにカーソルがあるときのみ)
メソッド・ブレークポイントの切り替え	関数の開始位置に関数ブレークポイントを設定します。 ([ソース]エディターにカーソルがあるときのみ)
監視ポイントの切り替え	選択した変数にデータブレークを設定する[監視ポイントの追加]ダイアログを開きます。 ([ソース]エディターにカーソルがあるときのみ)
すべてのブレークポイントをスキップ	すべてのブレークポイントを一時的にスキップします。
すべてのブレークポイントを除去	すべてのブレークポイントを削除します。
コールドリセット	コールドリセットを実行します。
ホットリセット	ホットリセットを実行します。
ユーザコマンド	ユーザ定義コマンドを実行します。
再開	サポートしていません。
シグナルなしで再開	サポートしていません。
指定行で再開	サポートしていません。
再開	実行を再開します。(デバッグプログラム中断時)
中断	実行を中断します。(デバッグプログラム実行時)
終了	デバッグ(GDB)を停止し、デバッグを終了します。(デバッグプログラム実行時/中断時)
ステップイン	ステップインします。([デバッグ]ビューでスタックフレーム選択時)
ステップ・オーバー	ステップオーバーします。([デバッグ]ビューでスタックフレーム選択時)
ステップ・リターン	ステップリターンします。([デバッグ]ビューでスタックフレーム選択時)
指定行まで実行	[ソース]エディター / [逆アセンブル]ビューのカーソル行の位置まで実行します。 (デバッグプログラム中断中で[ソース]エディターにカーソルがあるとき)
ステップ・フィルターの使用	サポートしていません。
前回の起動を実行	サポートしていません。
前回の起動をデバッグ	前回起動した構成でデバッグを開始します。
履歴の実行	サポートしていません。
実行	サポートしていません。
実行構成...	サポートしていません。
履歴のデバッグ	サブメニューに最近起動したデバッグ構成のショートカットを表示します。
デバッグ	サポートしていません。
デバッグの構成...	起動構成ダイアログを開きます。
外部ツール	外部起動設定ダイアログを開きます。 旧バージョンのGUIでデバッグを行う場合はこちらから起動します。

●[ウィンドウ]メニュー

ウィンドウ(W)	
新規ウィンドウ(N)	
新規エディター(E)	
パースペクティブを開く(O)	▶
ビューの表示(V)	▶
パースペクティブのカスタマイズ(C)...	
パースペクティブの別名保管(A)...	
パースペクティブのリセット(R)...	
パースペクティブを閉じる(C)	
すべてのパースペクティブを閉じる(L)	
ナビゲーション(Q)	▶
設定(P)	

表10.4.1.5.2 ツールバー [ウィンドウ]メニュー

メニュー	機能
新規ウィンドウ	別のIDEウィンドウを開くことができます。
新規エディター	現在アクティブなエディターをもうひとつ開きます。
パースペクティブを開く	デバッグパースペクティブやGNU33/パースペクティブに切り替えます。
ビューの表示	デバッグ時に使用するビューを開きます。
パースペクティブのカスタマイズ...	パースペクティブのカスタマイズ/保存/リセットを行います。
パースペクティブの別名保管...	
パースペクティブのリセット...	
パースペクティブを閉じる	パースペクティブを閉じます。
すべてのパースペクティブを閉じる	
ナビゲーション	"5.3.1 メニューバー"の「●[ウィンドウ]メニュー」を参照してください。
設定	IDEの設定ダイアログを開きます。

10.4.1.6 設定の変更

[ウィンドウ]>[設定]より、デバッグ時のIDEの設定を変更することができます。
ここでは、デバッグに使用する主な設定についてのみ説明します。

●[C/C++]>[デバッグ]

C/C++デバッグ時の一般的な設定です。

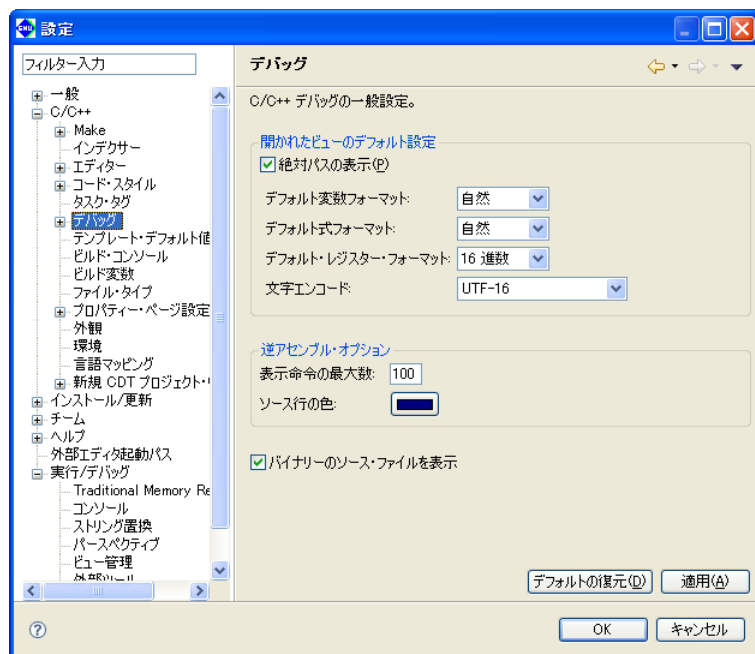


表10.4.1.6.1 [C/C++]>[デバッグ]

設定	設定内容
開かれたビューのデフォルト設定	ビューのデフォルト設定に関する項目です。
絶対パスの表示	サポートしていません。
デフォルト変数フォーマット	[変数]ビューの表示形式。デフォルト：自然
デフォルト式フォーマット	[式]ビューの表示形式。デフォルト：自然
デフォルト・レジスター・フォーマット	[レジスター]ビューの表示形式。デフォルト：16進数
文字エンコード	文字のエンコードを設定できます。デフォルト：UTF-16
逆アセンブル・オプション	[逆アセンブル]ビューの設定に関する項目です。
表示命令の最大数	逆アセンブルの最大表示行数を設定できます。デフォルト：100 この値は、現在のPCが含まれる関数の先頭からの表示行数です。この値で設定された行数以降にステップした場合は、ソースとのMIX表示はされず現在の関数の終わりまで表示されます。その場合には、設定値としてより大きな値を設定してください。
ソース行の色	ソース行の文字色を設定できます。デフォルト：濃い青
バイナリーのソース・ファイルを表示	サポートしていません。

このツリーの以下の設定画面は、操作しないでください。

- [C/C++]>[デバッグ]>[GDB MI]
- [C/C++]>[デバッグ]>[デバッガー・タイプ]
- [C/C++]>[デバッグ]>[ブレークポイント・アクション]
- [C/C++]>[デバッグ]>[共通ソース・ルックアップ・パス]

●[実行/デバッグ]

デバッグ時の一般的な設定です。

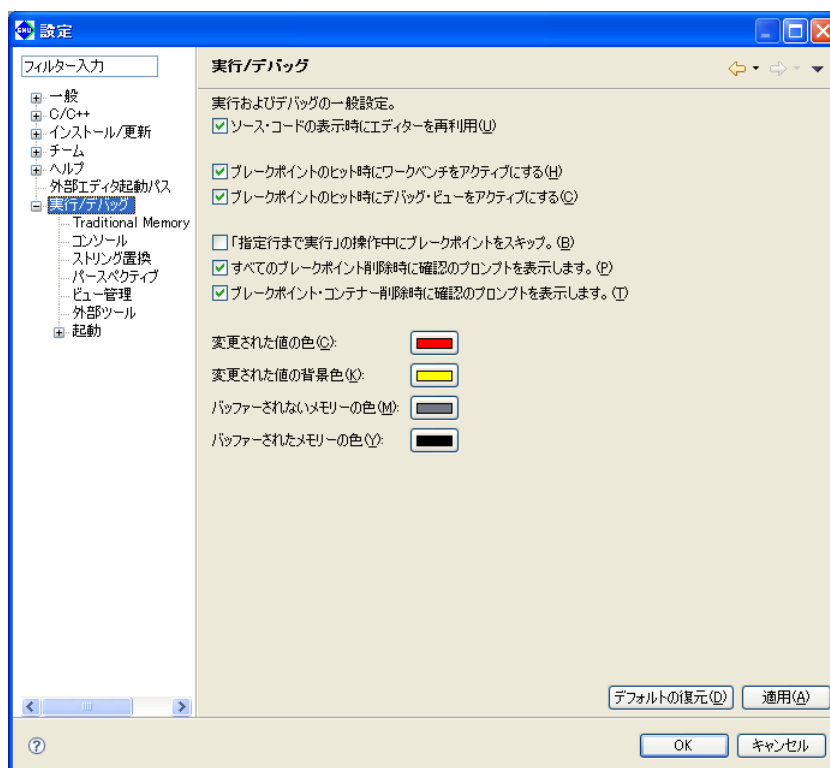


表10.4.1.6.2 [実行/デバッグ]

設定	設定内容
ソース・コードの表示時にエディターを再利用	ソース表示にエディターを再利用します。デフォルト：ON 変更しないでください。
ブレークポイントのヒット時にワークベンチをアクティブにする	ブレークポイントに停止したときにワークベンチをアクティブにします。デフォルト：ON
ブレークポイントのヒット時にデバッグ・ビューをアクティブにする	ブレークポイントに停止したときに[デバッグ]ビューをアクティブにします。デフォルト：ON
「指定行まで実行」の操作中にブレークポイントをスキップ。	[指定行まで実行]時にブレークポイントを一時的にスキップします。デフォルト：OFF
すべてのブレークポイント削除時に確認のプロンプトを表示します。	ブレークポイントを全削除するときに確認します。デフォルト：ON
ブレークポイント・コンテナー削除時に確認のプロンプトを表示します。	ブレークポイントのグループを削除するときに確認します。デフォルト：ON
変更された値の色	[変数]/[レジスター]ビューでの変更された値の色。デフォルト：赤（一覧形式時のみ）
変更された値の背景色	[変数]/[レジスター]ビューでの変更された値の背景色。デフォルト：黄色（表形式時のみ）
バッファーされないメモリーの色	サポートしていません。
バッファーされたメモリーの色	サポートしていません。

●[実行/デバッグ]>[起動]

デバッグ起動時の挙動に関する設定です。

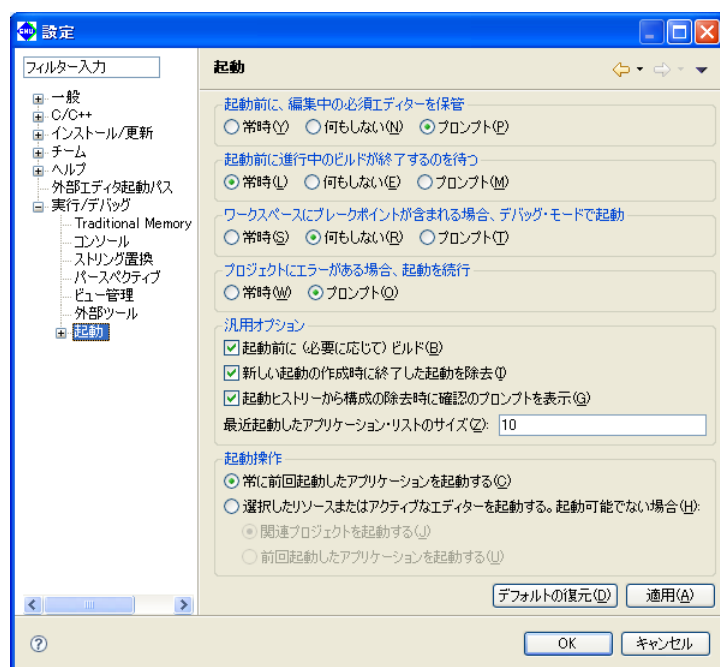


表10.4.1.6.3 [実行/デバッグ]>[起動]

設定	設定内容
起動前に、編集中の必須エディターを保管	起動時、編集集中で保存していないエディターを保存します。 常時 : 保存する 何もしない : 保存しない プロンプト : 確認する(デフォルト)
起動前に進行中のビルドが終了するのを待つ	ビルド中であれば、ビルド終了を待ちます。 常時 : 待つ(デフォルト) 何もしない : 待たない プロンプト : 確認する
ワークスペースにブレークポイントが含まれる場合、デバッグ・モードで起動	サポートしていません。
プロジェクトにエラーがある場合、起動を続行	プロジェクトにエラーがあっても起動します。 常時 : 常に起動する プロンプト : 確認する(デフォルト)
汎用オプション	
起動前に(必要に応じて)ビルド	起動前に必要に応じてビルドします。(実行ファイルがソースファイルより古いとき) デフォルト : ON
新しい起動の作成時に終了した起動を除去	新しく起動したとき、終了済みのものは[デバッグ]ビューより削除します。 デフォルト : ON
起動履歴から構成の除去時に確認のプロンプトを表示	サポートしていません。
最近起動したアプリケーション・リストのサイズ	[実行]、[デバッグ]、[外部ツール]の起動履歴の数。 デフォルト : 10

設定	設定内容
起動操作	
常に前回起動したアプリケーションを起動する	F11もしくはデバッグボタン押下時、前回のデバッグ構成で起動します。 デフォルト：ON この設定は変更しないでください。
選択したリソースまたはアクティブなエディターを起動する。起動可能でない場合	F11もしくはデバッグボタン押下時、現在選択されているエディターに対応する構成で起動します。 起動できない場合、以下のいずれかの方法で起動します。
関連プロジェクトを起動する	現在選択されているプロジェクトに対応する構成で起動します。
前回起動したアプリケーションを起動する	前回のデバッグ構成で起動します。

●[実行/デバッグ]>[Traditional Memory Reading]

[メモリー]ビューに関する設定です。

詳細は"10.4.9 [メモリー]ビュー"を参照してください。

●[実行/デバッグ]>[コンソール]

[コンソール]ビューの表示に関する設定です。

詳細は"10.4.10 [コンソール]ビュー"を参照してください。

このツリーの以下の設定画面は、操作しないでください。

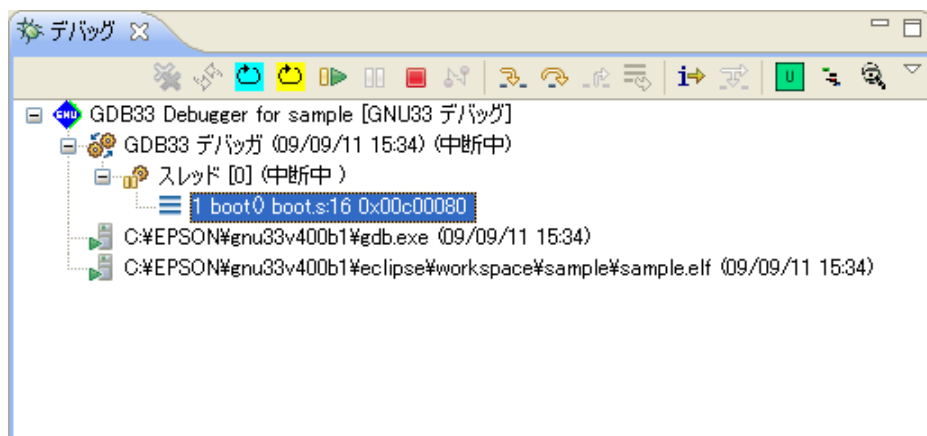
- [実行/デバッグ]>[ストリング置換]
- [実行/デバッグ]>[パースペクティブ]
- [実行/デバッグ]>[ビュー管理]
- [実行/デバッグ]>[外部ツール]
- [実行/デバッグ]>[起動]>[デフォルト・ランチャー]
- [実行/デバッグ]>[起動]>[起動構成]

10.4.2 [デバッグ]ビュー

[デバッグ]ビューはデバッグ時のメインウィンドウで、デバッグに関わるメニューやツールバーを持ちます。ステップ実行やプログラムの停止・再開は本画面で行います。

[デバッグ]ビューは常に開いている状態にしてください。

10.4.2.1 画面構成



10.4.2.2 メニュー / ツールバー

● ツールバー

表10.4.2.2.1 ツールバー

ボタン	機能
	終了したすべての起動を除去
	再開
	コールドリセット
	ホットリセット
	再開
	中断
	終了
	切断
	ステップイン
	ステップ・オーバー
	ステップ・リターン
	フレームにドロップ
	命令ステップ・モード
	ステップ・フィルターの使用
	ユーザコマンド
	プロファイラ
	カバレッジ

●コンテキストメニュー

 スタックのコピー(C)	Ctrl+C
 検索(E)...	Ctrl+F
 フレームにドロップ	
 再開	
 コールドリセット	
 ホットリセット	
 ステップイン(I)	F5
 ステップ・オーバー(O)	F6
 ステップ・リターン(U)	F7
 命令ステップ・モード	
 ステップ・フィルターの使用(U)	
 シグナルなしで再開	
 再開(M)	F8
 中断(S)	
 終了(I)	Ctrl+F2
 終了して再起動	
 切断(E)	
 終了したすべてを除去(A)	
 再起動(U)	
 GDB33 Debugger for sample の編集...	
 ソース・ルックアップの編集(K)...	
 ソースをルックアップ(S)	
 終了および除去(U)	
 すべて終了/切断(U)	
 プロパティ(R)	
 ユーザコマンド	
 プロファイラ	
 カバレッジ	

表10.4.2.2.2 コンテキストメニュー

メニュー	機能
スタックのコピー	[デバッグ]ビュー内の選択したアイコン以下の階層構造を文字列としてコピーします。
検索...	[デバッグ]ビュー内のアイコンを検索します。
フレームにドロップ	サポートしていません。
再開	サポートしていません。
コールドリセット	コールドリセットを実行します。
ホットリセット	ホットリセットを実行します。
ステップイン	ステップインします。([デバッグ]ビューでスタックフレーム選択時)
ステップ・オーバー	ステップオーバーします。([デバッグ]ビューでスタックフレーム選択時)
ステップ・リターン	ステップリターンします。([デバッグ]ビューでスタックフレーム選択時)
命令ステップ・モード	押し込んだ状態で、[ステップイン]/[ステップ・オーバー]がニーモニック1命令単位でステップ実行します。
ステップ・フィルターの使用	サポートしていません。
シグナルなしで再開	サポートしていません。
再開	実行を再開します。(デバッグプログラム中断時)
中断	実行を中断します。(デバッグプログラム実行時)
終了	デバッガ(GDB)を停止し、デバッグを終了します。(デバッグプログラム実行時/中断時)
終了して再起動	[終了]実行した後、[再起動]します。
切断	サポートしていません。
終了したすべてを除去	[デバッグ]ビューで終了済みのアイコンをすべて削除します。
再起動	終了したデバッガを再起動します。
GDB33 Debugger for **** の編集...	起動構成ダイアログを開き、編集することができます。
ソース・ルックアップの編集...	サポートしていません。
ソースをルックアップ	サポートしていません。
終了および除去	[デバッグ]ビューで選択中のデバッガを終了し、アイコンを削除します。
すべて終了/切断	起動中のすべてのデバッガを終了します。
プロパティ	サポートしていません。

メニュー	機能
ユーザコマンド	ユーザ定義コマンドを実行します。
プロファイラ	プロファイラ画面を起動します。
カバレッジ	カバレッジ画面を起動します。

●ビューメニュー

✓ 絶対パスの表示(P)
ビュー管理(V)...

表10.4.2.2.3 ビューメニュー

メニュー	機能
絶対パスの表示	スタックフレームでソースファイルのパス表示を切り替えます。
ビュー管理...	サポートしていません。

10.4.2.3 表示内容

[デバッグ]ビューでは以下の項目をツリー表示します。

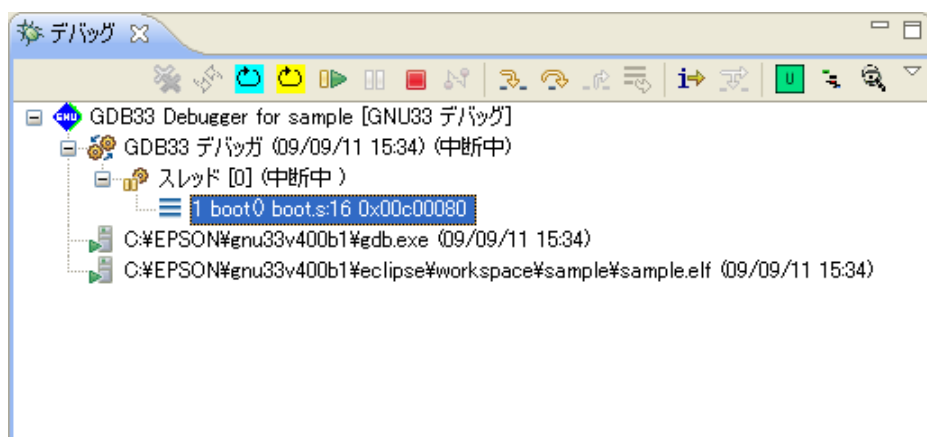


表10.4.2.3.1 表示項目

項目	説明
起動構成	デバッガを起動したときの起動構成名
デバッグターゲット	起動中のデバッガ/起動時間/状態
スレッド	実行中のスレッド(ただし、常に1つ)
スタックフレーム	ターゲットプログラムの停止位置 選択すると停止位置に関する情報を表示します。
デバッグプロセス	起動中のGDBプロセス： 選択すると[コンソール]ビューがアクティブになり、GDBのコマンドを入力できます。 ("10.4.10 [コンソール]ビュー"参照)
ターゲットプログラム	デバッグ中のプログラム： 選択すると[コンソール]ビューがアクティブになり、シミュレーテッドI/Oが表示されます。 ("10.4.11 [シミュレーテッドI/O]ビュー"参照)

10.4.2.4 操作

●ビューを開く/閉じる

[デバッグ]ビューは、デバッガを起動すると自動的に開きます。
デバッグ中は、[デバッグ]ビューは常に開いている状態にしてください。

ビューを再度開くには、[ウィンドウ]>[ビューの表示]から開いてください。
ビューのxボタンを押下して閉じることもできます。(ただし、デバッグ中は閉じないでください)

●デバッグプログラムの実行

[デバッグ]ビューの各メニューおよびボタンから、ターゲットプログラムのデバッグ(ステップ実行・実行・停止など)が行えます。

注：デバッグプログラムを実行するためには[デバッグ]ビューが常に開いているようにしてください。

CPUのリセット

[コールドリセット]ボタン：

ユーザが編集可能なコマンドファイル(¥gnu33¥resetcold.gdb)を実行します。

出荷時にはコマンドファイルには

```
c33 rstc
```

命令が記述されています。

[ホットリセット]ボタン：

ユーザが編集可能なコマンドファイル(¥gnu33¥resethot.gdb)を実行します。

出荷時にはコマンドファイルには

```
c33 rsth
```

命令が記述されています。

CPUのリセット時の初期設定内容は、以下のとおりです。

表10.4.2.4.1 CPUリセット時の初期値

C33レジスタ	初期値
R0-R15	0xffffffff
PC	ブートアドレス (0xc000000の内容が示すアドレス)
PSR	0x00000000
SP	0x0aaaaaaaa8 (C33 STD) 0x00000000 (C33 ADV, C33 PE)
AHR, ALR	0xffffffff
LCO	0x00000000 (C33 ADV)
LSA	0x00000000 (C33 ADV)
LEA	0x00000000 (C33 ADV)
SOR	0x00000000 (C33 ADV)
TTBR	0x00C00000 (C33 PE)*1 0x20000000 (C33 ADV)*1
DP	0x00000000 (C33 ADV)
USP	0x00000000 (C33 ADV)
SSP	0x00000000 (C33 ADV)

*1：TTBRレジスタは、コールドリセットのときのみ初期化されます。ホットリセットのときは初期化されません。

特に指定のないレジスタはC33コアすべてに共通です。

注：コネクトモードによりc33 rstcコマンドの動作が異なります。

- ICD2/ICD3/ICD6モード
上記の処理を行い、S1C33チップもリセットします。ターゲットボードはリセットされません。また、c33 rstcコマンド実行時にターゲットがフリーランしている場合は、強制ブレークをかけてからリセットを行います。
S5U1C33000H、S5U1C33001Hに接続されたターゲットをリセットすると、ターゲットシステムはフリーラン状態となりますが、c33 rstcコマンドでこれを停止させることができます。
- デバッグモニターモード
c33 rstcコマンドはc33 rsthコマンドと同じ動作をします。
S1C33チップのリセットやTTBRレジスタの初期化は行いません。
- シミュレータモード
パラメータファイルのMCU/MPU指定によりブートアドレスが決定します。
シミュレータモードかつC33 ADVモード時で、パラメータファイル内にアドレス0x20000000から1バイト以上のマップが設定されていない場合、PCの初期設定内容は0xc00000の内容が示すアドレスになります。

連続実行

[再開]ボタン：

- 停止中のターゲットプログラムを現在のPCから連続実行します。
連続実行中のプログラムは次のいずれかの要因によってブレークするまで停止しません。
- ブレークポイントに達する(指定行まで実行/untilコマンドのテンポラリブレークも含みます。)
 - [中断]ボタンをクリック(デバッグモニターモードを除きます。)
 - その他ブレーク要因の発生

[中断]ボタン：

- 実行中のターゲットプログラムを途中で強制的に停止させます。
ターゲットプログラムの実行によりCPUがスタンバイモード(HALT、SLEEP)になった場合やプログラムが永久ループ状態になった場合にブレークさせることができます。
プログラムが停止すると、以下のビューの表示が更新されます。
- [コンソール]ビュー (gdb)プロンプトが表示され、コマンドが入力できるようになります。
 - [ソース]エディター PC位置の行がハイライトされます。
 - [変数]ビュー フレーム上のローカル変数が表示・更新されます。
 - [レジスター]ビュー レジスタが表示・更新されます。
 - [メモリー]ビュー メモリモニタが登録されていればメモリが表示されます。

注：デバッグモニターモード(MON)でデバッグ中は、このボタンでターゲットを停止できません。

ステップ実行

[ステップイン]ボタン：

- ターゲットプログラムを現在のPCからソース1行分実行します。
[命令ステップ・モード]が選択されている場合は、ターゲットプログラムを現在のPCからニーモニック1命令分実行します。

[ステップ・オーバー]ボタン：

- ターゲットプログラムを現在のPCからソース1行分実行します。
関数コールやサブルーチンコールは、呼び出される関数/サブルーチンもすべて含め1ステップとして実行します。
[命令ステップ・モード]が選択されている場合は、ターゲットプログラムを現在のPCからニーモニック1命令分実行します。

[ステップ・リターン]ボタン：

- ターゲットプログラムを現在のPCから実行します。現在の関数から上位レベルにリターンしたところで停止します。
最下のスタックフレームが選択されている場合は、クリックできません。

[命令ステップ・モード]ボタン：

選択状態によって、[ステップイン]/[ステップ・オーバー]の各ボタンクリック時の処理が切り替わります。

表10.4.2.4.2 [命令ステップ・モード]ボタン状態

状態	処理
選択された状態(ON)	[ステップイン]/[ステップ・オーバー]の各ボタンクリック時に、ニーモニック1命令分で実行されます。
選択されない状態(OFF)	[ステップイン]/[ステップ・オーバー]の各ボタンクリック時に、C/C++ソース1行分で実行されます。

注：ローカル変数の初期化を行わないで変数を参照するようなプログラムのとき、GDBは関数のプロローグがどこまでか判別できません。これにより関数に[ステップイン]が正常に動作しないことがあります。

不定な変数を参照するプログラムにならないよう注意してください。

デバッグ終了**[終了]ボタン：**

実行中のデバッガ(GDB)を終了します。

●スタックフレーム

ターゲットプログラム停止時にスタックフレームが表示されます。

スタックフレームを選択すると、以下の各デバッグ操作を行うことができます。

スタックフレーム選択：

スタックフレームは、プログラム停止時に停止位置と呼び出し階層を表示します。

スタックフレームを選択すると、以下の各ビュー表示が更新されます。

- ・ [デバッグ]ビュー 各ボタンの有効・無効状態
- ・ [ソース]エディター 現在行をハイライト表示
- ・ [逆アセンブル]ビュー 現在の関数の逆アセンブリ表示
- ・ [ブレークポイント]ビュー 停止したブレークポイントのハイライト表示
- ・ [変数]ビュー 選択したフレーム上のローカル変数の表示
- ・ [レジスター]ビュー レジスタの表示
- ・ [メモリー]ビュー メモリの表示

注：各ビューの状態を参照する場合は必ずスタックフレームを選択する必要があります。スタックフレームは、カレントPCアドレス・現在の関数・ソース行番号を表示します。(最大99まで)

コンソール：

デバッガプロセス(gdb.exe)を選択すると[コンソール]ビューがアクティブになり、GDBのコマンドが入力可能な状態になります。

注：[コンソール]ビューを閉じてしまうと、コマンド入力ができなくなります。

[コンソール]ビューを閉じてしまった場合は、以下の手順で[コンソール]ビューを開いてください。

1. [ウィンドウ]>[ビューの表示]>[コンソール]を選択します。
2. [デバッグ]ビューのデバッガプロセス(gdb.exe)をクリックします。
3. [コンソール]ビューの[コンソールのピン留め]ボタンを選択してコンソールをピン留めしてください。

デバッガ起動直後に(gdb)のコンソールが表示されていない場合は、一旦[コンソール]ビューの[コンソールのピン留め]からピン留めをはずし、[デバッグ]ビューのデバッガプロセス(gdb.exe)のアイコンをクリックし、[コンソールのピン留め]ボタンを押下してからコンソールをピン留めしてください。

詳細は"10.4.10 [コンソール]ビュー"を参照してください。

シミュレーテッドI/O:

ターゲットプログラム(デバッグ中の`elf`)を選択すると[コンソール]ビューがアクティブになり、シミュレーテッドI/Oの出力が表示されます。"10.4.11 [シミュレーテッドI/O]ビュー"を参照してください。

注:[コンソール]ビューを閉じてしまうと、シミュレーテッド I/O出力が表示されなくなります。シミュレーテッド I/Oを使用する場合は、[コンソール]ビューを開いておいてください。

10.4.2.5 制限事項

- スタックフレームは最大99個までしか表示されません。
- 以下の場合、スタックフレームが正常に表示できません。
 - RAM上のプログラムをロード後に、この領域の内容を書き換えた場合
 - アセンブラ上で、現PCから上位アドレス方向4096命令までの間で`ret/ret.d/reti`命令が無い場合
 - PCリセット直後
- コマンドファイル内に、実行コマンド(`next / nexti / finish / until / continue`など)もしくは`quit`コマンドが含まれており、実行してもブレークポイントにヒットしないプログラムはデバッグが開始できません。
コマンドファイルにこれらの実行コマンドが含まれている場合は、ブートルーチンに停止させるか、ブレークポイントにヒットするように設定する必要があります。
このようなプログラムを実行させてしまった場合は、タスクマネージャなどからGDBを停止させる必要があります。

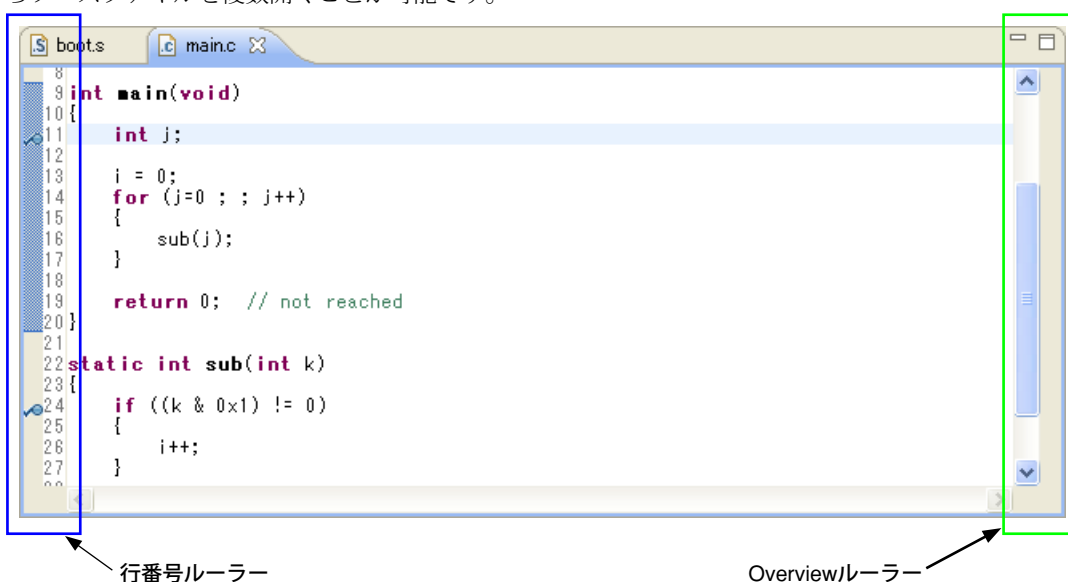
10.4.3 [ソース]エディター

IDE上でソース編集に使用しているエディターは、そのままデバッグ時のソースの現在行表示に使用されます。また、ブレークポイントの設定も[ソース]エディターで行います。

逆アセンブル表示は、[逆アセンブル]ビューで表示されます。"10.4.4 [逆アセンブル]ビュー"を参照してください。

10.4.3.1 画面構成

[ソース]エディターでは、デバッグ中でもGNU33パースペクティブの[C/C++ プロジェクト]ビューなどからソースファイルを複数開くことが可能です。



[ソース]エディターは、中央のエディター領域のほかに、左側の行番号ルーラーと右側のOverviewルーラーからなり、行番号ルーラーにブレークポイントを貼ることができます。

10.4.3.2 メニュー / ツールバー

"10.4.1.5 メニュー / ツールバー"の「●[実行]メニュー」と同じ操作が可能です。
[ソース]エディター独自のメニュー / ツールバーはありません。

●コンテキストメニュー

元に戻す(U)	Ctrl+Z
ファイルを前回(保管した状態に戻す(V))	
保管(S)	Ctrl+S
宣言を開(O)	F3
型階層を開	F4
呼び出し階層を開(Q)	Ctrl+Alt+H
クイック・アウトライン(L)	Ctrl+O
クイック型階層	Ctrl+T
マクロ拡張の探索(M)	Ctrl+=
ソース/ヘッダーの切り替え(G)	Ctrl+Tab
表示	Alt+Shift+W(W) ▶
切り取り(T)	Ctrl+X
コピー(C)	Ctrl+C
貼り付け(P)	Ctrl+V
クイック・フィックス(Q)	Ctrl+1
ソース	▶
リファクタリング	▶
宣言(L)	▶
参照(F)	▶
検索テキスト	▶
指定行まで実行(L)	Ctrl+R
指定行で再開(L)	
監視式を追加...	
実行(R)	▶
デバッグ(D)	▶
チーム(E)	▶
比較(A)	▶
置換(L)	▶
オブジェクトファイル変換	▶
設定...	
構成のビルド	▶
Make ターゲット	▶

ここでは、デバッグ中に使用できる関連メニューについてのみ説明しています。

表10.4.3.2.1 コンテキストメニュー

メニュー	機能
指定行まで実行	[ソース]エディター / [逆アセンブル]ビューのカーソル行の位置まで実行します。(デバッグプログラム中断中で[ソース]エディターにカーソルがある場合)
指定行で再開	サポートしていません。
監視式を追加...	[式]ビューに監視式を登録するダイアログを開きます。
実行	サポートしていません。
デバッグ	起動構成画面を開くメニューが表示されます。

10.4.3.3 表示内容

●現在行の表示

ターゲットプログラムを実行すると、カレントPCアドレスの行(次に実行される行)が緑で強調表示されます。プログラムを実行中は表示内容が更新されません。

現在行のソースファイルが自動的に開きますが、[デバッグ]ビューのスタックフレームをダブルクリックするなどして手動で開くこともできます。

現在行を表示する条件

- 現在行を表示するには、デバッグ中かつ[デバッグ]ビュー内のスタックフレームを選択する必要があります
- ソース行番号とソースコードは、ソースコード表示用のデバッグ情報を含む実行ファイル(elfファイル)を読み込んだ場合に表示可能です。
- C/C++ソースコードを表示するには、C/C++コンパイラの-gstabsオプションを指定してコンパイルしている必要があります。
- アセンブラソースコードを表示するには、アセンブラの-gstabsオプションを指定してアセンブルしている必要があります。
- デバッグ情報が含まれていない場合、あるいは対応するソースファイルが見つからない場合は表示されません。

●ブレークポイント表示

行番号ルーラーにブレークポイントを貼ることができます。

行ルーラーでは、ブレークポイントの状態および種類を表示します。

表10.4.3.3.1 ブレークポイント表示一覧

項目	説明
アイコン	[ブレークポイントの種類をアイコンで表現します。 アイコンの有り無しは、ソース上にブレークポイントが貼られているかどうかを示します。 有効/無効状態を表します。 有効・無効は、そのブレークポイントで停止するか停止しないかの設定を示します。 解決・未解決状態も表します。 解決・未解決は、ブレークポイントがデバッガに対して実際に貼られているかどうかを示します。

項目		説明
状態	● 有効・未解決状態	有効： ソースファイル上にブレークポイントは貼られています。 未解決： デバッグを起動時に - デバッグに実際に貼ります。 - 有効状態で貼られます。 - デバッグが起動中のときに、ソースファイル上にブレークポイントを貼ります。未解決状態になったものは、行ルーラーに  のアイコンを表示します。
	✔ 有効・解決状態	有効： ソースファイル上にブレークポイントは貼られています。 解決： デバッグが起動中であり、 - ブレークポイントが貼られています。 - ヒットしたときにこの位置で止まります。 ※ブレークポイントがこの状態にならないと停止しません。
	○ 無効・未解決状態	無効： ソースファイル上にブレークポイントが貼られています。 ただし、停止しない設定になっています。 未解決： デバッグを起動時に - デバッグに実際に貼ります。 - ただし、無効状態で貼られます。 - デバッグが起動中のときに、ソースファイル上にブレークポイントを貼ります。未解決状態になったものは、行ルーラーに  のアイコンを表示します。
	✔ 無効・解決状態	無効： ソースファイル上にブレークポイントは貼られています。 ただし、停止しない設定になっています。 未解決： デバッグが起動中であり、 - ブレークポイントが貼られています。 - ヒットしたときにこの位置で止まりません。
種類	● ソフトブレーク	ソフトウェアPCブレークポイント
		 [ソース]エディターに設定した場合
		 [逆アセンブル]ビューに設定した場合
		 関数ブレークポイントを設定した場合
		 テンポラリソフトウェアPCブレークポイントを設定した場合
	● ハードブレーク	ハードウェアPCブレークポイント
		 [ソース]エディターに設定した場合
		 [逆アセンブル]ビューに設定した場合
		 テンポラリハードウェアPCブレークポイントを設定した場合

●シンボルの値の表示

マウスポインタをシンボル名(単語として完全一致すれば、コメント内等でも可)の上に重ねると、シンボルの値をバルーン表示します。

```

13   i = 0;
14   for (j=0 ; ; j++)
15   {
16       sub(j);
17   }
18       i = 87251
19   return 0; // not reached
20 }

```

ローカル変数をポイントしたときは、現在のスタックフレーム(関数)に属するシンボルのみ表示し、他の関数内のシンボル値は表示されません。

また、最適化されたローカル変数のシンボル値は表示されません。

この機能は、[ウィンドウ]>[設定]>[C/C++]>[エディター]>[吹き出し]で無効にすることができます。

10.4.3.4 操作

●エディターを開く

[ソース]エディターは、プログラムが停止したとき[デバッグ]ビューでスタックフレームが選択されると対応するソースが自動的に開きます。エディターを再度開くには、[C/C++ プロジェクト]ビューからソースファイルをダブルクリックしてください。

また、エディターは複数開くこともできます。開きたいエディターをアクティブにした状態で[ウィンドウ]>[新規エディター]から開いてください。

エディターのxボタンを押下して閉じることもできます。

注：デバッグ中に編集しないでください。デバッグ中に編集してもデバッグ中の実行ファイルは再読み込みされませんので、プログラムの実行位置とソースファイルの実行位置があわなくなります。

この場合、デバッグを終了し再度デバッグを起動してください。

●ブレークポイント設定/解除

[エディターに表示中のソースファイルにソフトウェア/ハードウェアPCブレークポイントの設定と解除が行えます。

ブレークポイントの設定/解除は以下のいずれかの方法で行います。

・行ルーラーをダブルクリック

ブレークポイントを貼りたい位置で行ルーラーをダブルクリックすると、ソフトウェアPCブレークポイント ● が設定されます。設定されている位置で再度ダブルクリックすると、ブレークポイントが解除されます。

ダブルクリックでブレークポイントを設定できるのは、ソフトウェアPCブレークポイントのみです。ダブルクリックによるブレークポイント解除は、テンポラリソフトウェアPCブレークポイント、ハードウェアPCブレークポイント、テンポラリハードウェアPCブレークポイントでも可能です。

・行ルーラーのコンテキストメニュー>[ブレークポイントの切り替え]

・行ルーラーのコンテキストメニュー>[テンポラリソフトブレークの切り替え]

ブレークポイントの切り替え(B)
テンポラリソフトブレークの切り替え(T)

ブレークポイントを貼りたい位置で行ルーラーのコンテキストメニューからソフトウェアPCブレークポイント ● またはテンポラリソフトウェアPCブレークポイントが設定できます。

テンポラリソフトウェアPCブレークポイントは、1回のヒットのみ有効なブレークポイントです。設定されている位置で再度同じメニューを選択すると、ブレークポイントが解除されます。

・行ルーラーのコンテキストメニュー>[ハードブレークの切り替え]

・行ルーラーのコンテキストメニュー>[テンポラリハードブレークの切り替え]

ハードブレークの切り替え(H)
テンポラリハードブレークの切り替え(I)

ブレークポイントを貼りたい位置で行ルーラーのコンテキストメニューからハードウェアPCブ

ブレークポイント  またはテンポラリハードウェアPCブレークポイントが設定できます。
 テンポラリハードウェアPCブレークポイントは、1回のヒットのみ有効なブレークポイントです。
 設定されている位置で再度同じメニューを選択すると、ブレークポイントが解除されます。

・ [実行]>[ブレークポイントの切り替え]

・ [実行]>[行ブレークポイントの切り替え]

ブレークポイントを貼りたい行にカーソルを置き、[実行]メニューからソフトウェアPCブレークポイント  を設定できます。

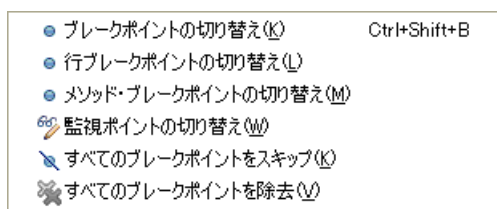
上記メニューどちらでも、同じようにソフトウェアPCブレークポイントが貼られます。

設定されている位置で再度同じメニューを選択すると、ブレークポイントが解除されます。

・ [実行]>[メソッド・ブレークポイントの切り替え]

カーソルを、関数内("{ " ~ "}")に置き、[実行]>[メソッド・ブレークポイントの切り替え]を選択すると関数ブレークポイント  を設定できます。関数ブレークポイントは、ソフトウェアPCブレークポイントです。

関数ブレークポイントで停止する位置は、関数プロログの次に実行されるソース行です。



注：・ ブレークポイントを設定できる位置

ブレークポイントは、[ソース]エディターのどの行に対しても設定することができます。ただし、コメントや空行に設定した場合は、設定した位置から下方向の一番近い命令で停止します。

・ ブレークポイントを設定できない位置

ソースファイルの最後尾以降の空白などの命令のない(デバッグ情報のない)部分には設定できません。

また、ROMエリアのプログラムには、ソフトウェアPCブレークポイントを設定することができません。

ROMエリアのプログラムにブレークポイントを設定するには、ハードウェアPCブレークポイントを使用してください。

・ ブレークポイントが設定されるタイミングについて

ソースファイル上にブレークポイントを貼った時点で[ブレークポイント]ビューに表示されますが、デバッグに対して設定が完了するまではブレークポイントで停止しません。

1. デバッグがすでに起動しているときにブレークポイントを貼る

設定時の動作：

デバッグが起動しているときにソースファイルにブレークポイントを貼ると、デバッグに対してブレークポイントが設定されます。

その結果、ブレークポイントにチェックのオーバーレイがつきます。  (解決)

ブレークポイントがこの状態にならないとこの位置では停止しません。

エラー時の動作：

エラーが発生してデバッグに対してブレークポイントが設定されない場合は、チェックのオーバーレイがつきません。  (未解決)

この場合、ダイアログでエラーが表示されます。また、  のアイコンがルーラーに表示されます。

また、有効・無効状態は以下のように変わります。

[ソース]エディターに設定したもの 有効・無効状態は変わりません

[逆アセンブル]ビューに設定したもの 無効状態に変更されます

デバッグが起動している間は、未解決状態  から解決状態  に移行させることはできません。

デバッグ起動時にエラーになったブレークポイントを再度設定するには一度削除してから設定しなおす必要があります。

2. デバッガがまだ起動していないときにブレークポイントを貼る

設定時の動作：

ブレークポイントは、デバッガが起動していなくてもソースファイル上に貼ることができます。設定した[ブレークポイント]ビューの一覧に表示されます。

これらのブレークポイントは、デバッガを起動した時にデバッガに対してブレークポイントが設定されます。

• ブレークポイントの設定数

ブレークポイントの設定数には上限があります。

- ソフトウェアPCブレークポイント：最大200カ所（テンポラリ含む）

- ハードウェアPCブレークポイント：最大2カ所

●ブレークポイントの有効/無効の切り替え

• ブレークポイント使用可能/使用不可

ブレークポイントには、有効・無効の状態があります。

有効は、ブレークポイントの位置で停止させることができます。ブレークポイントを設定したときは有効状態で設定されます。

無効は、ブレークポイントは設定されているが、一時的にそのブレークポイントで停止させたくないときに使用します。

• すべてのブレークポイントをスキップ

[実行]>[すべてのブレークポイントをスキップ]を選択すると、すべてのブレークポイントが一時的に無効になり停止しなくなります。

もう一度[すべてのブレークポイントをスキップ]を選択すると無効から有効に切り替わります。

●ブレークポイントの削除

• すべてのブレークポイントの除去

[実行]>[すべてのブレークポイントを除去]を選択すると、現在設定されているすべてのブレークポイントを削除します。

●データブレークポイントの設定/解除

特定の変数に対しての書き込み/読み出しアクセスを監視するデータブレークポイント(監視ポイント)を設定することができます。

[監視ポイントの切り替え]：

ソースファイル上のグローバル変数名またはC++クラスのフィールドを選択状態にして[実行]>[監視ポイントの切り替え]から、データブレークポイントを設定/解除することができます。

データブレークの設定は[ソース]エディターでは表示されません。[ブレークポイント]ビューで確認してください。

注：• データブレークは1カ所のみ設定可能です。

• ローカル変数をデータブレークポイントに設定することはできません。

• シンボルの型が、double、配列、構造体のときは、設定されません。

• 構造体が以下のように定義されているとき、

```
struct {
    char c;
}
ST_DATA stdata;
```

構造体のメンバstdata.cを範囲指定して[監視ポイントの切り替え]でデータブレークを設定することはできません。[ブレークポイント]ビューの[監視ポイントの追加]ダイアログに、構造体メンバstdata.cを入力して登録してください。詳細については、"10.4.5 [ブレークポイント]ビュー"を参照してください。

●指定位置までのプログラム実行

[指定行まで実行]：

コンテキストメニューの[指定行まで実行]を選択すると、ターゲットプログラムがカレントPCアドレスから実行を開始し、カーソルのある行で停止します(プログラムがこの行を通過しない場合は停止しません)。

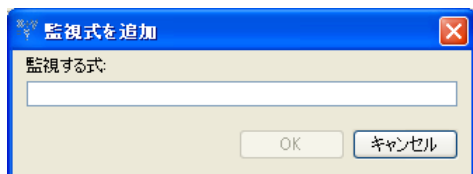
注：プログラムはテンポラリソフトウェアPCブレークポイントを設定して停止させます。

ただし、この機能はソフトウェアPCブレークポイントが既に200カ所(最大)に設定されている場合でも有効です。

●監視式の登録

[監視式を追加...]：

[ソース]エディター上でコンテキストメニューを開き、[監視式を追加...]を選択するとポップアップメニューが表示されます。



監視式を入力してOKを押下すると[式]ビューに登録されます。評価できない無効な式は、登録後に[式]ビューでエラーが表示されます。

また、ソース上のシンボルを選択状態にした上でコンテキストメニューを開くと、選択されたテキストが入力された状態で開きます。

●設定の変更

[ソース]エディターでは、[ウィンドウ]>[設定]から以下の設定が変更可能です。

表10.4.3.4.1 変更可能な項目

変更可能な設定	パス
フォント	[一般]>[外観]>[色とフォント]>[基本]-[テキスト・フォント]
現在行(カレントPC)の色	[一般]>[エディター]>[テキスト・エディター]>[注釈]>[現在の命令ポインターのデバッグ]
変数値/バレーンON/OFF	[C/C++]>[エディター]>[吹き出し]>[最適な引き出し]

10.4.3.5 制限事項

シンボル指定のメモリダンプの機能はありません。[メモリー]ビューでシンボルを登録してください。コマンドから設定したブレークポイントに対しては、以下の各操作はできません。

表10.4.3.5.1 制限事項一覧

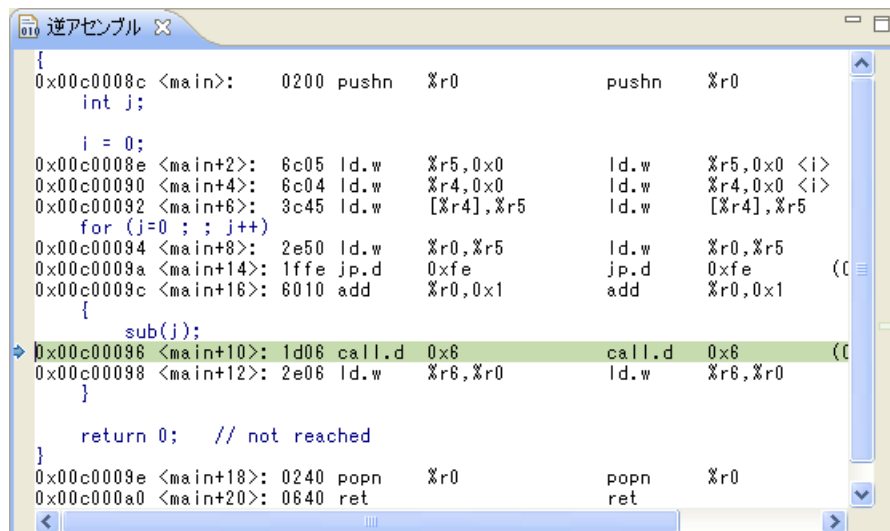
操作	操作内容
[ブレークポイント]ビュー表示	コマンドから設定したブレークポイントは[ブレークポイント]ビューで表示されない場合があります。
[ブレークポイントの切り替え]	ソフトウェアPCブレークポイントを解除する。
[テンポラリソフトブレークの切り替え]	テンポラリソフトウェアPCブレークポイントを解除する。
[ハードブレークの切り替え]	ハードウェアPCブレークポイントを解除する。
[テンポラリハードブレークの切り替え]	テンポラリハードウェアPCブレークポイントを解除する。
[行ブレークポイントの切り替え]	ソフトウェアPCブレークポイントを解除する。
[メソッド・ブレークポイントの切り替え]	関数ブレークポイントを解除する。
[監視ポイントの切り替え]	データブレークポイントを解除する。
[ブレークポイント使用可能/使用不可]	ブレークポイントを有効化・無効化する。
[すべてのブレークポイントをスキップ]	すべてのブレークポイントを有効化・無効化する。
[ブレークポイント・プロパティ]	ブレークポイントの詳細情報を参照する。
[ブレークポイントのエクスポート]	ブレークポイントのエクスポート。

10.4.4 [逆アセンブル]ビュー

[逆アセンブル]ビューは[デバッグ]ビューで選択されたスタックフレームに対応するプログラムの逆アセンブリを表示します。

10.4.4.1 画面構成

[逆アセンブル]ビューは、[ウィンドウ]>[ビューの表示]>[逆アセンブル]から開くことができます。



10.4.4.2 メニュー / ツールバー

● ツールバー / ビューメニュー

ツールバー / ビューメニューはありません。

● コンテキストメニュー

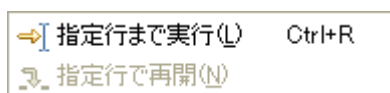


表10.4.3.5.1 コンテキストメニュー

メニュー	機能
指定行まで実行	[ソース]エディター / [逆アセンブル]ビューのカーソル行の位置まで実行します。(デバッグプログラム中断中で[ソース]エディターにカーソルがあるとき)
指定行で再開	サポートしていません。

10.4.4.3 表示内容

デバッグ起動中、現在行付近の逆アセンブリを表示します。

●逆アセンブル表示

実行ファイル(elfファイル)を逆アセンブルして表示しています。

C/C++ソースファイルまたはアセンブラソースファイルのソースコードの各行ごとに、対応するアドレス、ラベル、アセンブラ基本命令、およびアセンブラ拡張命令を表示します。

逆アセンブリを表示するには、デバッグ中かつ[デバッグ]ビュー内のスタックフレームを選択する必要があります。

元のソースコードを表示する条件

- 逆アセンブリに対応するソースコードは、ソースコード表示のデバッグ情報を含む実行ファイル(elfファイル)を読み込んだ場合に表示可能です。
- C/C++ソースコードを表示するには、C/C++コンパイラの-gstabsオプションを指定してコンパイルしている必要があります。
- アセンブラソースコードを表示するには、アセンブラの--gstabsオプションを指定してアセンブルしている必要があります。
- デバッグ情報が含まれていない場合、あるいは対応するソースファイルが見つからない場合は強制的に逆アセンブリのみの表示となります。

注：・アセンブラの拡張命令の最適化のため、アセンブラソースファイルと逆アセンブリ表示の内容は必ずしも一致しません。

- 逆アセンブリの表示行数は100行です。

元のソースコードのソース行は濃い青で表示されます。

これらの設定は、「10.4.1.6 設定の変更」[●C/C++ >デバッグ]の[逆アセンブル・オプション]から変更可能です。この値で設定された行数以降にステップした場合は、ソースとのMIX表示はされず現在の関数の終わりまで表示されます。その場合には、設定値としてより大きな値を設定してください。

- デバッグ中でない場合は逆アセンブル表示されません。

10.4.4.4 操作

●ビューを開く/閉じる

[変数]ビューは、[ウィンドウ]>[ビューの表示]から開いてください。「10.4.1.3 ビューを開く/閉じる」を参照してください。ビューのxボタンをクリックして閉じるができます。

●ブレークポイント設定/解除

[逆アセンブル]ビューに表示中のソースファイルにソフトウェア/ハードウェアPCブレークポイントの設定と解除が行えます。

ブレークポイントの設定/解除は以下のいずれかの方法で行います。

・行ルーラーをダブルクリック

ブレークポイントを貼りたい位置で行ルーラーをダブルクリックすると、ソフトウェアPCブレークポイント  が設定されます。設定されている位置で再度ダブルクリックすると、ブレークポイントが解除されます。

ダブルクリックでブレークポイントを設定できるのは、ソフトウェアPCブレークポイントのみです。ダブルクリックによるブレークポイント解除は、テンポラリソフトウェアPCブレークポイント、ハードウェアPCブレークポイント、テンポラリハードウェアPCブレークポイントでも可能です。

- ・行ルーラーのコンテキストメニュー>[ブレークポイントの切り替え]
- ・行ルーラーのコンテキストメニュー>[テンポラリソフトブレークの切り替え]

ブレークポイントの切り替え(B)
テンポラリソフトブレークの切り替え(T)

ブレークポイントを貼りたい位置で行ルーラーのコンテキストメニューからソフトウェアPCブレークポイント  またはテンポラリソフトウェアPCブレークポイントが設定できます。

テンポラリソフトウェアPCブレークポイントは、1回のヒットのみ有効なブレークポイントです。設定されている位置で再度同じメニューを選択すると、ブレークポイントが解除されます。

- ・行ルーラーのコンテキストメニュー>[ハードブレークの切り替え]
- ・行ルーラーのコンテキストメニュー>[テンポラリハードブレークの切り替え]

ハードブレークの切り替え(H)
テンポラリハードブレークの切り替え(T)

ブレークポイントを貼りたい位置で行ルーラーのコンテキストメニューからハードウェアPCブレークポイント  またはテンポラリハードウェアPCブレークポイントが設定できます。

テンポラリハードウェアPCブレークポイントは、1回のヒットのみ有効なブレークポイントです。設定されている位置で再度同じメニューを選択すると、ブレークポイントが解除されます。

注：・ブレークポイントが設定できるタイミングについて

ブレークポイントは、デバッグが起動中にのみ貼ることができ、逆アセンブリ上にブレークポイントを貼った時点で[ブレークポイント]ビューに表示されます。

ブレークポイントは、アドレスがある逆アセンブリの命令行にのみ設定可能です。C/C++ソース行には設定できません。

```
xcall    main                ; goto main
0x00c00088 <boot+8>: 1c02 call    0x2
```

また、以下の箇所も設定できません。

- ・先頭のext命令を除く拡張命令の行
例)
ext xxx ○ 設定可
ext xxx × 設定不可
ld.w xxx × 設定不可
- ・遅延命令の行(遅延分岐命令の次の行)
例)
jr.d xxx ○ 設定可
cmp × 設定不可

ブレークポイントを貼ると、デバッグに対してブレークポイントが設定されます。その結果、ブレークポイントにチェックのオーバーレイ  がつきます。ブレークポイントがこの状態にならないとこの位置では停止しません。

エラーが発生して使用可能でない場合は、チェックのオーバーレイがつかず、無効状態  になります。

エラーのためブレークポイントが設定できない場合は、ダイアログで表示されます。

- ・ブレークポイントの設定数
ブレークポイントの設定数には上限があります。
- ソフトウェアPCブレークポイント：最大200カ所（テンポラリ含む）
- ハードウェアPCブレークポイント：最大2カ所
- ・ROMエリアのプログラムには、ソフトウェアPCブレークポイントを設定することができません。ROMエリアのプログラムにブレークポイントを設定するには、ハードウェアPCブレークポイントを使用してください。

●ブレークポイントの有効/無効の切り替え

・ブレークポイント使用可能/使用不可

ブレークポイントには、有効・無効の状態があります。

有効は、ブレークポイントの位置で停止させることができます。ブレークポイントを設定したときは有効状態で設定されます。

無効は、ブレークポイントは設定されているが、一時的にそのブレークポイントで停止させたくないときに使用します。

・すべてのブレークポイントをスキップ

[実行]>[すべてのブレークポイントをスキップ]を選択すると、すべてのブレークポイントが一時的に無効になり停止しなくなります。

もう一度[すべてのブレークポイントをスキップ]を選択すると無効から有効に切り替わります。

●ブレークポイントの削除

・すべてのブレークポイントの除去

[実行]>[すべてのブレークポイントを除去]を選択すると、現在設定されているすべてのブレークポイントを削除します。

●ブレークポイントのプロパティ

・ブレークポイント・プロパティ

ブレークポイントが設定されている行の行ルーラーのコンテキストメニュー>[ブレークポイント・プロパティ]からブレークポイントの詳細情報を見ることができます。

"10.4.5.5 操作「●ブレークポイントの詳細情報」"を参照してください。

●指定位置までのプログラム実行

[指定行まで実行]：

コンテキストメニュー>[指定行まで実行]を選択すると、ターゲットプログラムがカレントPCアドレスから実行を開始し、カーソルのある行で停止します。(プログラムがこの行を通過しない場合は無効です)。

注：プログラムはテンポラリソフトウェアPCブレークポイントを設定して停止させます。ただし、この機能はソフトウェアPCブレークポイントが既に200カ所(最大)に設定されている場合でも有効です。

10.4.4.5 制限事項

逆アセンブリとソースコードの混在表示にのみ対応しています。アセンブリのみの表示には対応しません。

コマンドから設定したブレークポイントに対しては、以下の各操作はできません。

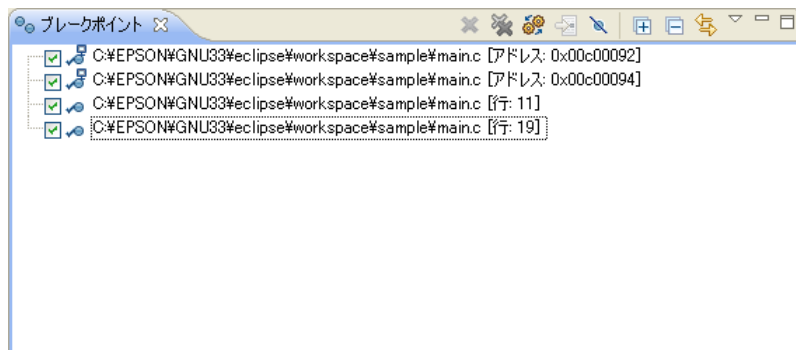
表10.4.4.5.1 制限事項一覧

操作	操作内容
[ブレークポイント]ビュー表示	コマンドから設定したブレークポイントは[ブレークポイント]ビューで表示されない場合があります。
[ブレークポイントの切り替え]	ソフトウェアPCブレークポイントを解除する。
[テンポラリソフトブレークの切り替え]	テンポラリソフトウェアPCブレークポイントを解除する。
[ハードブレークの切り替え]	ハードウェアPCブレークポイントを解除する。
[テンポラリハードブレークの切り替え]	テンポラリハードウェアPCブレークポイントを解除する。
[行ブレークポイントの切り替え]	ソフトウェアPCブレークポイントを解除する。
[メソッド・ブレークポイントの切り替え]	関数ブレークポイントを解除する。
[監視ポイントの切り替え]	データブレークポイントを解除する。
[ブレークポイント使用可能/使用不可]	ブレークポイントを有効化・無効化する。
[すべてのブレークポイントをスキップ]	すべてのブレークポイントを有効化・無効化する。
[ブレークポイント・プロパティ]	ブレークポイントの詳細情報を参照する。
[ブレークポイントのエクスポート]	ブレークポイントのエクスポート。

10.4.5 [ブレークポイント]ビュー

[ブレークポイント]ビューは、ブレークポイントの表示と管理に使用します。ブレークポイントの設定は、[ソース]エディターと[逆アセンブル]ビューより行います。
ワークスペース内のすべてのプロジェクトのブレークポイントを表示します。

10.4.5.1 画面構成



10.4.5.2 メニュー / ツールバー

● ツールバー

表10.4.5.2.1 ツールバー

ボタン	機能
選択されたブレークポイントを除去	選択したブレークポイントを削除します。
すべてのブレークポイントを除去	すべてのブレークポイントを削除します。
選択したターゲットでサポートされるブレークポイントを表示	サポートしていません。
ブレークポイント用のファイルへジャンプ	選択したブレークポイントをエディター上で開きます。
すべてのブレークポイントをスキップ	すべてのブレークポイントを一時的にスキップします。
すべて展開	閉じているブレークポイント一覧を開きます。
すべて縮小表示	表示しているブレークポイント一覧をたたみます。
デバッグ・ビューにリンク	ブレークポイントで停止したとき、ヒットした[ブレークポイント]ビュー内のブレークポイントをハイライトします。

●ビューメニュー

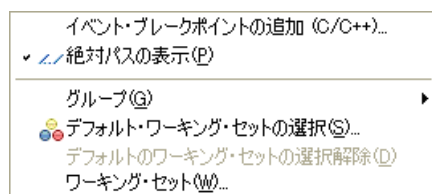


表10.4.5.2.2 ビューメニュー

メニュー	機能
イベント・ブレークポイントの追加	サポートしていません。
絶対パスの表示	ブレークポイントが設定されているファイルのフルパス表示を切り替えます。
グループ	ブレークポイントをグループごとに表示します。
デフォルト・ワーキング・セットの選択...	サポートしていません。
デフォルト・ワーキング・セットの選択解除	サポートしていません。
ワーキング・セット...	サポートしていません。

●コンテキストメニュー

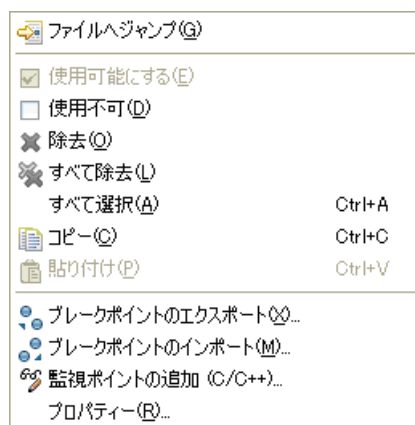


表10.4.5.2.3 コンテキストメニュー

メニュー	機能
ファイルへジャンプ	選択したブレークポイントをエディター上で開きます。
使用可能にする	ブレークポイントを有効化します。(ブレークします)
使用不可	ブレークポイントを無効化します。(ブレークしません)
除去	選択中のブレークポイントを表示から削除します。
すべて除去	すべてのブレークポイントを表示から削除します。
すべて選択	ブレークポイント一覧を全選択します。
コピー/貼り付け	ブレークポイントのコピー/貼り付けを行います。
ブレークポイントのエクスポート...	ブレークポイントを保存します。
ブレークポイントのインポート...	ブレークポイントを復元します。
監視ポイントの追加...	データブレークを設定する[監視ポイントの追加]ダイアログを開きます。
プロパティ ...	ブレークポイントのプロパティを表示するダイアログを開きます。

10.4.5.3 ブレークポイント全般の仕様

●ブレークポイントの状態(有効/無効と解決/未解決)

表10.4.5.3.1 ブレークポイントの状態一覧

ブレークポイントの状態	状態の内容
アイコンあり/なし	ソースファイル上にブレークポイントが貼られているかどうかを示します。有効/無効を設定することができます。 有効/無効に限らずデバッグ起動時にGDBに設定されます。
有効/無効	ソースファイル上にブレークポイントが貼られています。 有効：ブレークポイント位置で停止させる設定 無効：ブレークポイント位置で停止させない設定
解決/未解決	ソースファイル上にブレークポイントが貼られています。 解決/未解決の状態はデバッグ起動時に変化します。 解決：ターゲットに対してブレークポイントの設定が成功したことを示します 未解決：ターゲットに対してブレークポイントの設定が失敗したことを示します

表10.4.5.3.2 アイコンと状態の関係

アイコン	有効	解決	IDE上の状態	ブレークする
			GDB上の状態	
	○	×	IDEのソースファイル上に設定済 GDB上は未設定	×
	×	×	IDEのソースファイル上に設定済 GDB上は未設定	×
	○	○	IDEのソースファイル上に設定済 GDB上は設定済 (プログラムが停止するのは、この状態のみ)	○
	×	○	IDEのソースファイル上にブレークポイントが設定済 GDB上は設定済 ただし、実行時にはスキップされます	×
なし	×	×	ブレークポイントはソースファイルに設定されていません GDBにブレークポイントは設定されていません	×

●ブレークポイントの状態遷移

有効/無効状態および解決/未解決状態は以下の図のように遷移します。

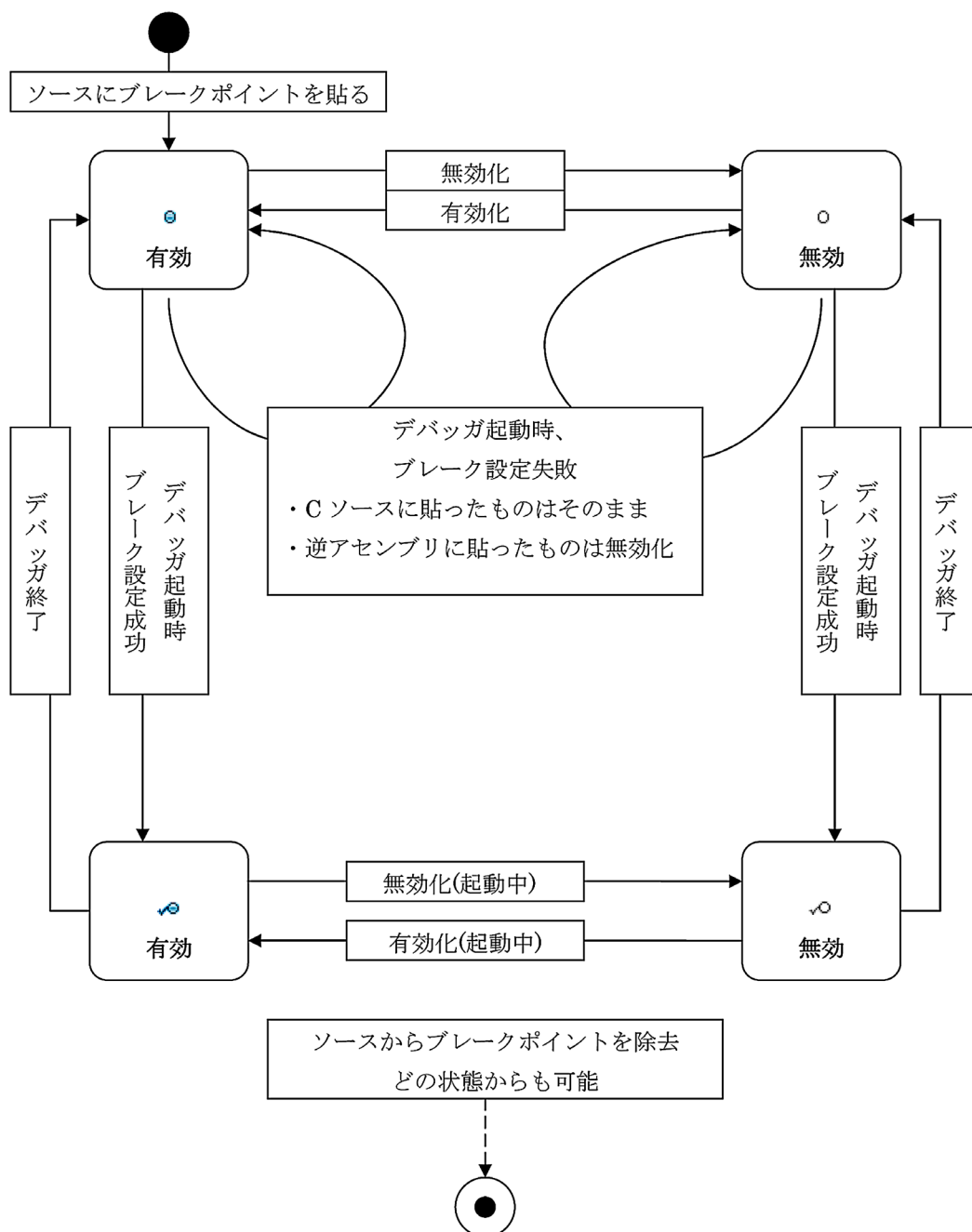


図10.4.5.3.1 ブレークポイントの状態遷移

●ブレークポイントが設定されるタイミング

・[ソース]エディターへのブレークポイント設定

デバッグが起動していてもブレークポイントを貼ることができます。ソースファイル上にブレークポイントを貼った時点で[ブレークポイント]ビューに表示されます。

ブレークポイントは、どの行に対しても設定可能です。ただし、コメントや空行に貼った場合に実際に停止する位置は、下方向の直近の命令の位置です。自動的に位置補正はされません。

注：以下の箇所には設定できません。

- ・ソースファイルの最後尾以降の空白などの命令のない(デバッグ情報のない)部分
- ・パラメータファイルのRAM範囲外には、ソフトウェア/テンポラリブレークポイントを設定できません
- ・ターゲットのROM上のアドレスには、ソフトウェア/テンポラリブレークポイントを設定できません

・[逆アセンブル]ビューへのブレークポイント設定

デバッグが起動中にのみブレークポイントを貼ることができます。逆アセンブリ上にブレークポイントを貼った時点で[ブレークポイント]ビューに表示されます。

ブレークポイントは、アドレスがある逆アセンブリの命令行にのみ設定可能です。

C/C++ソース行には設定できません。

```

xcall    main                ; goto main
0x00c00088 <boot+8>: 1c02 call    0x2

```

注：以下の箇所には設定できません。

- ・パラメータファイルのRAM範囲外には、ソフトウェア/テンポラリブレークポイントを設定できません
- ・ターゲットのROM上のアドレスには、ソフトウェア/テンポラリブレークポイントを設定できません
- ・先頭のext命令を除く拡張命令の行

例)

```

ext xxx      ○ 設定可
ext xxx      × 設定不可
ld.w xxx     × 設定不可

```

・遅延命令の行(遅延分岐命令の次の行)

例)

```

jr.d xxx     ○ 設定可
cmp          × 設定不可

```

・デバッグが起動しているときのブレークポイント設定

設定時の動作

デバッグが起動しているときにソースファイルにブレークポイントを貼ると、デバッグに対してブレークポイントが設定されます。その結果、ブレークポイントにチェックのオーバーレイ  (解決)がつけます。

ブレークポイントがこの状態にならないとこの位置では停止しません。

エラー時の動作

エラーが発生して使用可能でない場合は、チェックのオーバーレイ  (未解決)がつけません。エラーのためブレークポイントが設定できない場合はダイアログで表示されます。行ルーラーには  のアイコンが表示されます。

有効・無効状態は以下のように変わります。

- ・[ソース]エディターに設定 有効・無効状態は変わりません。
- ・[逆アセンブル]ビューに設定 無効状態に変更されます。

- 注：・ デバッガが起動している間は、未解決状態 ● から解決状態 ◐ に移行させることはできません。
- ・ デバッガ起動時にエラーになったものを再度貼るには一度削除してから貼り直す必要があります。
 - ・ デバッガが起動していないときのブレークポイント設定

設定時の動作

ブレークポイントは、デバッガが起動していなくてもソースファイル上に貼ることができます。設定したブレークポイント[ブレークポイント]ビューの一覧に表示されます。これらのブレークポイントは、「デバッガが起動しているときのブレークポイント設定」の記述の通り、デバッガを起動した時にコマンドが送信されブレークポイントが使用可能になります。

●コマンドからブレークポイント命令を実行したときの制限

[ブレークポイント]ビューは、画面上およびコンソールからコマンドによって設定されたブレークポイントを表示します。

ただし、コマンドから設定されたブレークポイントに関しては、以下の制限があります。

表10.4.5.3.3 コマンドからブレークポイントを設定したときの制限一覧

発行したコマンド	挙動 および 制限
break main (tbreak/hbreak/thbreakも同様)	行ブレークポイントが設定されます。 関数ブレークポイントでは設定されません。関数ブレークポイントは、[メソッド・ブレークポイントの切り替え]から設定することができます。アドレス位置的にはプロログのあと(関数ブレークとして)に設定されます。 ただし、ライブラリなど、デバッグ情報(アドレス位置のソース行)がない場所は、一覧に表示されません。
break file:lineno (tbreak/hbreak/thbreakも同様)	行ブレークポイントが設定されます。 ソースファイル上のどこにでも設定できますが、実際の停止箇所は下方向の直近の命令となります。
break *0xXXXX (tbreak/hbreak/thbreakも同様)	指定したアドレスに対応するソース行をもとに、行ブレークポイントが設定されます。 ただし、ブレークポイントを設定したアドレスにデバッグ情報がない場合は、ソース行が取得できないため[ソース]エディター [逆アセンブル]ビューにアイコンは表示されません。 [ブレークポイント]ビューには、アドレスブレークポイントとして表示されます。
delete	ブレークポイントをすべて解除します。 一覧のブレークポイントは ● (未解決)状態になりGDBを起動しなおさないと解決状態に移行できません。 一覧から削除するためには、ブレークポイントビューの[除去]を行う必要があります。

●テンポラリブレークポイントに関して

ブレークポイント一覧には、テンポラリブレークポイントも表示します。

テンポラリブレークポイントを設定したとき

表10.4.5.3.4 テンポラリブレークポイント

GDBの状態	表示
GDB停止時	[ブレークポイント]ビュー一覧に未解決状態 ● で表示します。
GDB実行時	[ブレークポイント]ビュー一覧に解決状態 ◐ で表示します。 ただし、ブレークすると未解決状態 ● になります。

10.4.5.4 表示内容

ワークスペース内のすべてのプロジェクトのブレークポイントを表示します。
ブレークポイントの設定は、[ソース]エディターと[逆アセンブル]ビューより行います。

●ブレークポイント一覧表示

各行ごとに以下のブレークポイント情報を表示します。

表10.4.5.4.1 ブレークポイント表示一覧

項目		説明
チェックボックス		ブレークポイントが有効か無効かを示します。 チェックあり：有効 チェックなし：無効
アイコン		ブレークポイントの種類をアイコンで表現します。 アイコンの有り無しは、ソース上にブレークポイントが貼られているかどうかを示します。 有効/無効状態を表します。 有効・無効は、そのブレークポイントで停止するか停止しないかの設定を示します。 解決・未解決状態も表します。 解決・未解決は、ブレークポイントがデバッガに対して実際に貼られているかどうかを示します。
状態	● 有効・未解決状態	有効： ソースファイル上にブレークポイントは貼られています。 未解決： デバッガを起動時に ・デバッガに実際に貼ります。 ・有効状態で貼られます。 ・デバッガが起動中のときに、ソースファイル上にブレークポイントを貼ります。未解決状態になったものは、行ルラーに🚧のアイコンを表示します。
	✔ 有効・解決状態	有効： ソースファイル上にブレークポイントは貼られています。 解決： デバッガが起動中であり、 ・ブレークポイントが貼られています。 ・ヒットしたときにこの位置で止まります。 ※ブレークポイントがこの状態にならないとこの位置では停止しません。
	○ 無効・未解決状態	無効： ソースファイル上にブレークポイントが貼られています。 ただし、停止しない設定になっています。 未解決： デバッガを起動時に ・デバッガに実際に貼ります。 ・ただし、無効状態で貼られます。 ・デバッガが起動中のときに、ソースファイル上にブレークポイントを貼ります。未解決状態になったものは、行ルラーに🚧のアイコンを表示します。
	✔○ 無効・解決状態	無効： ソースファイル上にブレークポイントが貼られています。 ただし、停止しない設定になっています。 解決： デバッガが起動中であり、 ・ブレークポイントが貼られています。 ・ヒットしたときにこの位置で止まりません。

項目	説明
種類	● ソフトブレーク ソフトウェアPCブレークポイント
	● [ソース]エディターに設定した場合
	🔍 [逆アセンブル]ビューに設定した場合
	🔍 関数ブレークポイントを設定した場合
	🔍 テンポラリソフトウェアPCブレークポイントを設定した場合
	● ハードブレーク ハードウェアPCブレークポイント
	● [ソース]エディターに設定した場合
	🔍 [逆アセンブル]ビューに設定した場合
	🔍 テンポラリハードウェアPCブレークポイントを設定した場合
	データブレーク データブレーク (監視ポイント)
ファイル名	🔍 [読み取り]ブレーク
	🔍 [書き込み]ブレーク
	🔍 [読み取り]/[書き込み]ブレーク
設定箇所	ブレークポイントが設定されているソースファイル名を表示します。[絶対パスの表示]選択時は、フルパス表示されます。 ブレークポイントをダブルクリックすると[ソース]エディター上の対応する行が強調表示されます。
	アドレス ソフトウェア/ハードウェアPCブレークが設定されているアドレスを表示します。[逆アセンブル]ビューに貼ったときに表示します。
	行 ソフトウェア/ハードウェアPCブレークが設定されている行番号を表示します。[ソース]エディターのソース行に貼ったときに表示します。
	関数 ソフトウェア/ハードウェアPCブレークが設定されている関数名を表示します。[メソッド・ブレークポイントの切り替え]より、関数ブレークポイントを貼ったときに表示します。 関数ブレークポイントは、プロローグ処理後の最初の命令に貼られます。
	式 データブレークが設定されているシンボルを表示します。 [監視ポイントの追加]または[監視ポイントの切り替え]により、データブレークポイントを貼ったときに表示します。

●グループ表示

[グループ]からグループを選択すると、ブレークポイントはグループごとに表示されます。

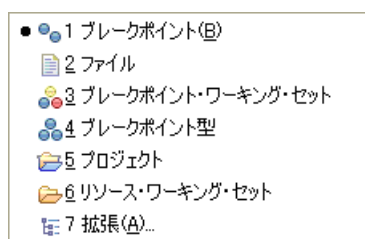


表10.4.5.4.2 グループ表示項目

グループ	表示方法
ブレークポイント	全ブレークポイントを表示します。(デフォルト)
ファイル	ソースファイルごとに表示します。
ブレークポイント・ワーキング・セット	サポートしていません。
ブレークポイント型	ブレークポイントの種類ごとに表示します。
プロジェクト	プロジェクトごとに表示します。
リソース・ワーキング・セット	サポートしていません。
拡張...	サポートしていません。

グループ表示時は[すべて展開]/[すべて縮小表示]より表示/非表示できます。

注：・ブレイクポイントが表示される順は、ブレイクポイントを貼った順と異なります。

- ・オンチップエリアブレイク (C33 ADV) は表示されません。
- ・オンチップバスブレイク (C33 ADV) は表示されません。

10.4.5.5 操作

●ビューを開く/閉じる

[ブレイクポイント]ビューは、[ウィンドウ]>[ビューの表示]から開いてください。"10.4.1.3 ビューを開く/閉じる"を参照してください。ビューのxボタンをクリックして閉じるができます。

●ソフトウェアPCブレイクポイントの設定

以下の画面から設定可能です。各ブレイクポイントの設定方法については[ソース]エディター / [逆アセンブル]ビューを参照してください。

[ソース]エディター

- ・行ルーラーをダブルクリック
- ・行ルーラーのコンテキストメニュー>[ブレイクポイントの切り替え]
- ・行ルーラーのコンテキストメニュー>[テンポラリソフトブレイクの切り替え]
- ・[実行]>[ブレイクポイントの切り替え]
- ・[実行]>[行ブレイクポイントの切り替え]
- ・[実行]>[メソッド・ブレイクポイントの切り替え]

[逆アセンブル]ビュー

- ・行ルーラーをダブルクリック
- ・行ルーラーのコンテキストメニュー>[ブレイクポイントの切り替え]
- ・行ルーラーのコンテキストメニュー>[テンポラリソフトブレイクの切り替え]

例) 行ルーラーのコンテキストメニュー

ブレイクポイントの切り替え(B)
 テンポラリソフトブレイクの切り替え(T)
 ハードブレイクの切り替え(H)
 テンポラリハードブレイクの切り替え(T)
 ブレイクポイントを使用不可にする(D)
 ブレイクポイント・プロパティ(P)...

設定したブレイクポイントは、●のアイコンで[ブレイクポイント]ビューに表示されます。

デバッグが起動中で、ブレイクポイントが貼れずにエラーが発生した場合は、エラーダイアログを表示します。ルーラーに ⚠ のアイコンが表示されます。

注：ソフトウェアPCブレイクポイントの設定数には上限があります。最大200カ所(テンポラリ含む)まで設定可能です。デバッグが起動中にこれを超えた場合には、エラーが表示されます。デバッグが停止中であればソースファイル上に上限を越えて設定することができます。また、ROMエリアのプログラムには、ソフトウェアPCブレイクポイントを設定することができません。

ROMエリアのプログラムにブレイクポイントを設定するには、ハードウェアPCブレイクポイントを使用してください

●ハードウェアPCブレークポイントの設定

以下の画面から設定可能です。各ブレークポイントの設定方法については[ソース]エディター/[逆アセンブル]ビューを参照してください。

[ソース]エディター

- ・行ルーラーのコンテキストメニュー>[ハードブレークの切り替え]
- ・行ルーラーのコンテキストメニュー>[テンポラリハードブレークの切り替え]

[逆アセンブル]ビュー

- ・行ルーラーのコンテキストメニュー>[ハードブレークの切り替え]
- ・行ルーラーのコンテキストメニュー>[テンポラリハードブレークの切り替え]

設定したブレークポイントは、のアイコンで[ブレークポイント]ビューに表示されます。デバッガが起動中で、ブレークポイントが貼れずにエラーが発生した場合は、エラーダイアログを表示します。ルーラーにのアイコンが表示されます。

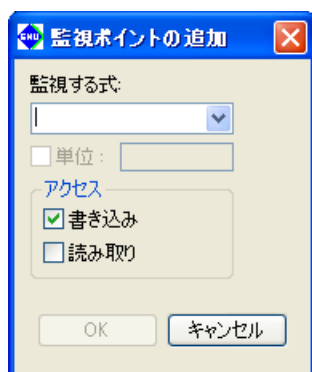
注：ハードウェアPCブレークポイントの設定数には上限があります。最大2カ所まで設定可能です。

デバッガが起動中にこれを超えた場合には、エラーが表示されます。

デバッガが停止中であればソースファイル上に上限を越えて設定することができます。

●データブレークポイントの設定

特定の変数またはメモリ領域に対しての書き込み/読み出しアクセスを監視するデータブレークポイント(監視式)を設定することができます。



[監視ポイントの追加...] :

[監視ポイントの追加]ダイアログからデータブレークポイントを設定できます。

このダイアログは各ビューによって開き方が変わります。

各ビューからの開き方は次の通りです。

- ・[ブレークポイント]ビュー
コンテキストメニュー>[監視ポイントの追加...]
- ・[ソース]エディター
グローバル変数を選択した状態 [実行]>[監視ポイントの切り替え]
- ・[メモリー]ビュー
コンテキストメニュー>[監視ポイントの追加...]
- ・[変数]ビュー
コンテキストメニュー>[監視ポイントの追加...]

ダイアログ内で以下の項目を入力してOKを押下すると、設定された内容でデータブレークポイントが設定されます。

表10.4.5.5.1 [監視ポイントの追加]ダイアログ

入力	値
監視する式	監視したいグローバル変数もしくはアドレスを入力します。 例) i グローバルシンボルiへのアクセス 例) 0x8000 0x8000番地へのアクセス アドレス入力時の設定可能範囲は0x0 ~ 0xFFFFFFFFです。 入力文字数に制限はありません。
単位	サポートしていません。
書き込み/読み取り	[書き込み]: 書き込み時にブレークします。 [読み取り]: 読み出し時にブレークします。 両方にチェックを入れた場合は、書き込み時および読み出し時にブレークします。

注: • データブレークは1カ所のみ設定可能です。

- ローカル変数をデータブレークポイントに設定することはできません。
- シンボルの型が、long long、double、配列、構造体のときは、設定されません。
- ブレークポイントが貼れずにエラーが発生した場合は、エラーダイアログを表示します。

●ブレークポイント有効/無効の切り替え

有効はプログラム実行中にこのポイントでブレーク可能となり、無効のブレークポイントは無視されブレークしなくなります。

チェックボックス:

ブレークポイントリストの行先頭にあるチェックボックスをクリックして有効/無効を切り換えます。チェックが付くと有効になり、チェックが付いていない場合は無効になります。

[使用可能にする]/[使用不可]:

ブレークポイントを選択してコンテキストメニューから有効にする場合は [使用可能にする]を選択します。無効にする場合は[使用不可]を選択します。

複数のまたはすべてのブレークポイントを有効/無効にする場合は、[すべて選択]から全ブレークポイントを選択して、メニューより[使用可能にする]もしくは[使用不可]を選択してください。

[すべてのブレークポイントをスキップ]:

ブレークポイントビューのツールバーメニューから[すべてのブレークポイントをスキップ]を選択すると、すべてのブレークポイントが一時的に無効になり停止しなくなります。もう一度[すべてのブレークポイントをスキップ]を選択すると無効から有効に切り替わります

●ブレークポイント箇所へのジャンプ

[ブレークポイント]ビューのリストをダブルクリックもしくはコンテキストメニュー>[ファイルへジャンプ]を選択すると、[ソース]エディターで該当する箇所を表示します。

●ブレークポイントの削除

[除去]/[すべて除去]:

削除するブレークポイントの行を選択し、コンテキストメニュー>[除去]を選択します。すべてのPCブレークポイントを削除するには、[すべて除去]を選択します。

●ブレークポイントの保存/復元

登録したブレークポイントは、IDEを終了したときに自動的に保存され、IDEを次回起動したときに復元されます。デバッガを再度起動したときに、登録されているブレークポイントをGDBデバッガに対して設定します。

ブレークポイントは外部ファイル(EclipseのXMLファイル形式)に保存・外部ファイルから復元することができます。

保存:

コンテキストメニュー>[ブレークポイントのエクスポート...]

ブレークポイント一覧から、保存したいブレークポイントを選択し、ファイル名を指定して保存します。

復元:

コンテキストメニュー>[ブレークポイントのインポート...]

外部ファイルを指定し、ブレークポイント一覧にブレークポイントを復元します。

●ブレークポイントの詳細情報

以下の操作から、ダイアログが開きブレークポイントの詳細情報を確認することが可能です。

- ・ コンテキストメニュー>[プロパティ]
 - ・ [ソース]エディターの行ルーラー>[ブレークポイントのプロパティ ...]
 - ・ [逆アセンブル]ビューの行ルーラー>[ブレークポイントのプロパティ ...]
- ツリーの[共通]を選択すると、以下の情報を表示します。

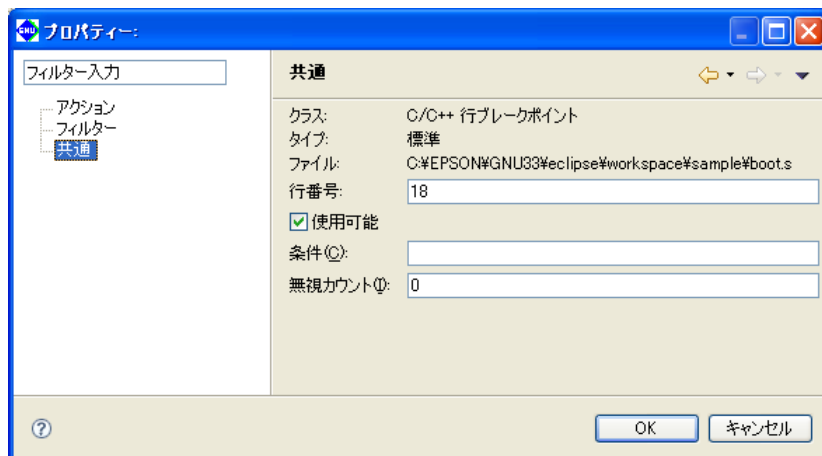


表10.4.5.5.2 ブレークポイントのプロパティ

項目		意味
クラス		ブレークポイントの内部タイプ C/C++ 行ブレークポイント ([ソース]エディター設定時) C/C++ 関数ブレークポイント ([ソース]エディターから関数ブレークポイントを設定時) C/C++ 監視ポイント (データブレーク設定時) 使用不可 (逆アセンブリに設定したとき)
タイプ		ブレークポイントの種類 標準 (ソフトウェアPCブレーク) ハードウェア (ハードウェアPCブレーク) 一時的 (テンポラリソフトウェアPCブレーク) ハードウェア一時的 (テンポラリハードウェアPCブレーク)
Classによる表示	ファイル/行番号	ファイル名 ([ソース]エディターに設定したとき、または変数にデータブレークを設定したとき)
	関数	関数名 ([ソース]エディターから関数ブレークポイントを設定したとき)
	アドレス	アドレス (逆アセンブリに設定したとき)
	監視する式	式 (データブレークを設定したとき)
使用可能		有効・無効状態
条件		ブレーク条件式
無視カウント		ヒットまでのスキップカウント

[アクション]および[フィルター]についてはサポートしていません。

10.4.5.6 制限事項

- On Chip Area Break (C33 ADV) の表示は対応していません。
- On Chip Bus Break (C33 ADV) の表示は対応していません。
- コマンドからブレークポイント命令を実行した場合、ブレークポイント一覧に正しく反映されません。ブレークポイントは、画面上から設定してください。
- デバッグを終了しても、プロジェクトのブレークポイント情報は[ブレークポイント]ビューに残っています。この状態で、別プロジェクトのデバッグを起動すると、前回のプロジェクトのブレークポイントが設定されます。
ただし、この場合、[ソース]エディター上にはブレークポイントが表示されません。前回のプロジェクトのブレークポイントが設定されないようにするためには、ツールバーの[すべてのブレークポイントを除去]ボタンなどから前回のブレークポイントの設定を削除しておく必要があります。
- `--double-starting`を使用して2つのプロジェクトを同時にデバッグする場合は、異なるワークスペースで**IDE**を2つ起動する必要があります。異なるワークスペースにそれぞれのプロジェクトをインポートしてからデバッグを起動してください。

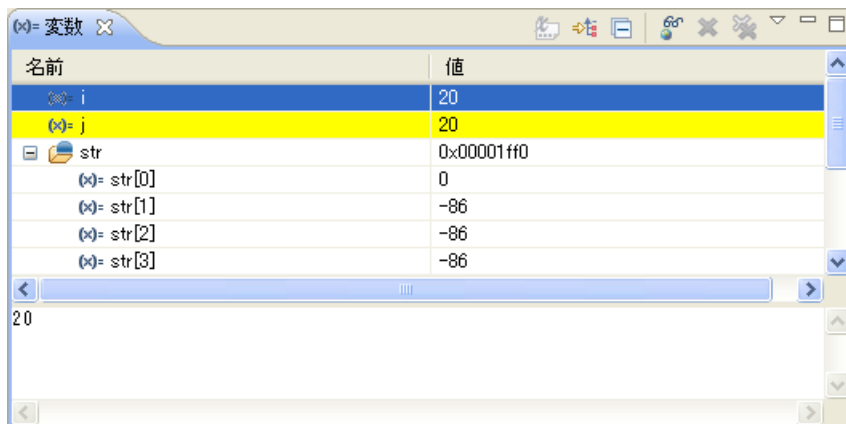
10.4.6 [変数]ビュー

[変数]ビューは、ローカル変数の値をモニタするときに使用します。

ローカル変数は[デバッグ]ビューのスタックフレームに対応して自動的に表示されます。また、登録すればグローバル変数の値も参照可能です。

10.4.6.1 画面構成

[デバッグ]ビューで選択されているスタックフレームで定義されている関数引数およびローカル変数名とその値を表示します。



10.4.6.2 メニュー / ツールバー

● ツールバー

表10.4.6.2.1 ツールバー

ボタン	機能
	型名の表示 一覧に式の型を表示します。
	論理構造の表示 サポートしていません。
	すべて縮小表示 表示している変数一覧をたたみます。
	グローバル変数の追加 グローバル変数を一覧から選択して追加します。
	選択されたグローバル変数の除去 選択中のグローバル変数を表示から削除します。
	すべてのグローバル変数の除去 すべてのグローバル変数を表示から削除します。

● ビューメニュー

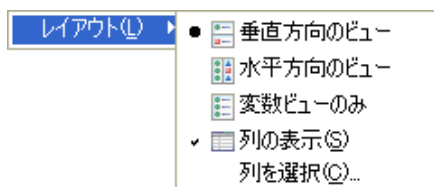


表10.4.6.2.2 ビューメニュー

メニュー	機能
レイアウト	ビュー表示のレイアウトを変更します。
垂直方向のビュー	詳細ペインを縦に表示します。
水平方向のビュー	詳細ペインを横に表示します。
変数ビューのみ	詳細ペインを表示しません。
列の表示	表形式での表示を切り替えます。
列の選択...	サポートしていません。

●コンテキストメニュー

すべて選択(A)	Ctrl+A
変数のコピー(U)	Ctrl+C
☒ 使用可能にする	
☐ 使用不可	
×[] 配列として表示...	
型にキャスト...	
オリジナル型の復元	
View Memory	
検索(F)...	Ctrl+F
値の変更(C)...	
監視ポイントの追加 (C/C++)...	
グローバル変数の追加...	
グローバル変数の除去	
すべてのグローバル変数を除去	
監視式の作成(I)	
フォーマット	▶

表10.4.6.2.3 コンテキストメニュー

メニュー	機能
すべて選択	ビューの表示を全選択します。
変数のコピー	選択した内容をコピーします。
使用可能にする	変数の更新が行われるようにします。
使用不可	変数の更新を行いません。
配列として表示...	サポートしていません。
型にキャスト...	サポートしていません。
オリジナル型の復元	サポートしていません。
View Memory	変数の値のアドレスを[メモリー]ビューで表示します。
検索...	変数を検索します。
値の変更...	変数の値を変更するダイアログを開きます。
監視ポイントの追加...	データブレークを設定する[監視式を追加]ダイアログを開きます。
グローバル変数の追加...	グローバル変数を一覧から選択して追加します。
グローバル変数の除去	選択中のグローバル変数を表示から削除します。
すべてのグローバル変数の除去	すべてのグローバル変数を表示から削除します。
監視式の作成	変数を[式]ビューに登録します。
フォーマット	表示形式を変更します。
バイナリー	2進数
自然	整数型は符号付き10進数、浮動小数点型は指数表記
10進数	符号付き10進数
16進数	16進数

注：詳細ペインのコンテキストメニューからの操作はサポートしていません。

10.4.6.3 表示内容

●ローカル変数一覧

[デバッグ]ビューで選択されているスタックフレームで定義されている関数引数およびローカル変数名とその値を表示します。

変数値はプログラムの実行後に更新されます。

他の関数に移ると、表示されるローカル変数も自動的に変更されます。

変数が配列やポインタの場合、変数名の前に[+]または[▶]マークが表示されます。そこをクリックするとマークが[-]または[▼]に変わり、配列内の情報あるいはポインタで示されるアドレスの内容が表示されます。

[フォーマット]が[バイナリー]のときはシンボルサイズに相当する桁数を表示します。

例: `int iSymbol = 0x1a55;`

int型は32ビットのため、iSymbol 0000000000000000000000001101001010101と表示されます。

●表形式/一覧形式

[変数]ビューは、2通りの表示方法が選択できます。

表10.4.6.3.1 表示方法

表示方法	切り替え表示方法	特徴	
表形式	ビューメニューから [列の表示]選択	表示	[名前]列と[値]列に分かれて表示します。
		ハイライト	変更箇所は、黄色でハイライト表示されます。
		値の変更	[値]列から直接編集できます。
一覧形式	ビューメニューから [列の表示]非選択	表示	変数=値 の形式で表示します。
		ハイライト	赤色でハイライト表示されます。
		値の変更	コンテキストメニュー>[値の変更]で編集します。

ハイライトの色は、"10.4.1.6 設定の変更"●[実行/デバッグ]"の[変更された値の色]または[変更された値の背景色]で変更可能です。

注：・ブレークポイントなどでプログラム停止時に、[変数]ビューのスクロール領域外の表示されていない部分はハイライトされません。

・floatやdoubleなどの浮動小数点型の値を参照するときは、[フォーマット]を[自然]に設定してください。([フォーマット]が[16進数]その他の場合は、指数部+仮数部の形式で表示されます。)

10.4.6.4 操作

●ビューを開く/閉じる

[変数]ビューは、[ウィンドウ]>[ビューの表示]から開いてください。"10.4.1.3 ビューを開く/閉じる"を参照してください。ビューのxボタンをクリックして閉じるができます。

●変数値の変更

[値の変更...]：

変数の値を変更することができます。内容を変更する変数を選択し、[値の変更...]を選択します。

ダイアログが表示され、値を編集してOKを押下します。

値は10進数または16進数(0xつき)で入力します。

[値]列：

[変数]ビューを表形式で表示中の場合、[値]列から直接値を編集することができます。

注：floatやdoubleなどの浮動小数点型の値を変更するときは、[フォーマット]を[自然]に設定した上で入力してください。([フォーマット]が[16進数]その他の場合は、指数部+仮数部の形式で入力する必要があります。)

●グローバル変数の登録/削除

[変数]ビューでは、ローカル変数は自動的に表示されますが、グローバル変数の値を参照・編集したい場合にも使用できます。

[式]ビューではグローバル変数の値を変更できませんが、[変数]ビューに登録すると値を編集できます。

[グローバル変数の追加...]：

コンテキストメニュー>[グローバル変数の追加]から、グローバル変数の一覧がダイアログで表示されます。

この一覧から、表示させたい変数を選択すると[変数]ビューに値が表示されます。

登録された変数は、デバッグを終了したときに記憶されます。次回起動時にも同じ変数をモニタできます。

[グローバル変数の除去]：

[グローバル変数の除去]から、登録したグローバル変数を削除することができます。

[すべてのグローバル変数の除去]から、[変数]ビューに登録したすべてのグローバル変数を削除することができます。

●表示形式の変更

コンテキストメニュー>[フォーマット]

表示形式を以下の中から選択できます。

表10.4.6.4.1 表示形式一覧

選択	表示形式	デフォルト
バイナリー	2進数	
自然	符号付き10進数	○
10進数	符号付き10進数	
16進数	16進数	

表示形式の変更は、選択項目すべてに適用されます。

●メモリでの表示

[View Memory]：

特定の変数の値を[メモリー]ビューに登録して表示することができます。

登録したい変数のコンテキストメニュー>[View Memory]から、[メモリー]ビューが開くと変数が[メモリー]ビューに登録・表示されます。

●監視式の登録

[監視式の作成]：

特定の変数の値を[式]ビューに登録して表示することができます。

登録したい変数のコンテキストメニュー>[監視式の作成]から、[式]ビューに登録・表示します。

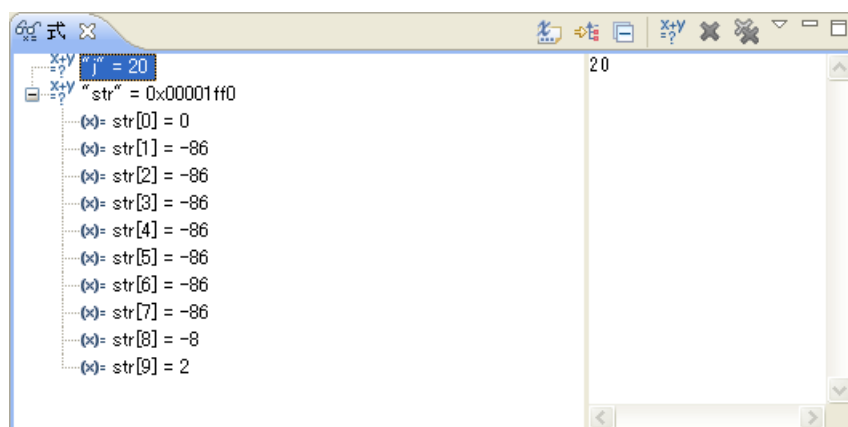
10.4.6.5 制限事項

- ・ [デバッグ]ビュー上のスタックフレーム選択時に、ローカル変数が複数個表示されることがあります。これは、以下のいずれかによります。
 - 最適化オプションあり(-O0以外)でビルドした場合
 - ローカル変数がレジスタに割り当てられている場合
 また、一度も使用されていないローカル変数は、ビルド時に最適化される場合があります。その場合には、ローカル変数は[変数]ビューに表示されません。
- ・ ローカル変数の内容をコマンド等で修正しても表示を更新しません。
- ・ プログラム停止時に値が更新されないことがあります。この場合は詳細ペインで値を確認してください。

10.4.7 [式]ビュー

[式]ビューは、任意の監視式(グローバルシンボルやレジスタ)を登録し、その値をモニタするのに使用します。ワークスペース内のすべてのプロジェクトの監視式を表示します。

10.4.7.1 画面構成



10.4.7.2 メニュー / ツールバー

● ツールバー

表10.4.7.2.1 ツールバー

ボタン	機能
	型名の表示
	論理構造の表示
	すべて縮小表示
	新規監視式を作成
	選択された式の除去
	すべての式を除去

● ビューメニュー

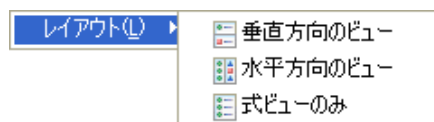


表10.4.7.2.2 ビューメニュー

メニュー	機能
レイアウト	ビュー表示のレイアウトを変更します。
垂直方向のビュー	詳細ペインを縦に表示します。
水平方向のビュー	詳細ペインを横に表示します。
式ビューのみ	詳細ペインを表示しません。

●コンテキストメニュー

- 監視式を選択中(ルートノードの  アイコン)

すべて選択(A)	Ctrl+A
 式のコピー(E)	Ctrl+C
 除去(Q)	
 すべて除去(L)	
検索(F)...	Ctrl+F
 値の変更(O)...	
 監視式を追加(A)...	
使用不可	
使用可能にする	
監視式を編集(E)...	
監視式を再評価(R)	

- 次の式を選択中(子ノードの  変数アイコン/  ポインタアイコン)

すべて選択(A)	Ctrl+A
 式のコピー(E)	Ctrl+C
 除去(Q)	
 すべて除去(L)	
検索(F)...	Ctrl+F
 値の変更(O)...	
 監視式を追加(A)...	
使用不可	
使用可能にする	
監視式を編集(E)...	
 監視式の作成(I)	
フォーマット	▶
 [] 配列として表示...	
 型にキャスト...	
オリジナル型の復元	
View Memory	

表10.4.7.2.3 コンテキストメニュー

メニュー	機能
すべて選択	ビューの表示を全選択します。
式のコピー	選択した内容をコピーします。
除去	選択中の式を表示から削除します。
すべて除去	すべての式を表示から削除します。
検索...	式を検索します。
値の変更...	式の値を変更するダイアログを開きます。 追加した監視式を選択した場合は表示されません。
監視式を追加...	監視式を追加します。
監視式を編集...	監視式を編集します。 追加した監視式を選択したときのみ表示されます。
監視式を再評価	監視式を再評価(再計算)します。 追加した監視式を選択したときのみ表示されます。
監視式の作成	選択した値を監視式として[式]ビューに登録します。 追加した監視式を選択した場合は表示されません。
使用可能にする	監視式の更新が行われるようにします。
使用不可	監視式の更新を行いません。

メニュー	機能
フォーマット	表示形式を変更します。 追加した監視式を選択した場合は表示されません。
バイナリー 自然 10進数 16進数	2進数 整数型は符号付き10進数、浮動小数点型は指数表記 符号付き10進数 16進数
配列として表示...	サポートしていません。
型にキャスト...	サポートしていません。
オリジナル型の復元	サポートしていません。
View Memory	式の値のアドレスを[メモリー]ビューで表示します。 追加した監視式を選択した場合は表示されません。

注：詳細ペインのコンテキストメニューからの操作はサポートしていません。

10.4.7.3 表示内容

●監視式一覧

監視リストに登録された式(シンボル名)とその評価結果値を表示します。式の評価結果値の表示内容は、プログラムの実行によりその値が変化した場合に更新されます。コマンド等で変更した場合は更新されません。

シンボルが配列やポインタの場合、シンボル名の前に[+]または[▶]マークが表示されます。そこをクリックするとマークが[-]または[▼]に変わり、配列内の情報あるいはポインタで示されるアドレスの内容が表示されます。

[フォーマット]が[バイナリー]のときはシンボルサイズに相当する桁数を表示します。

例: `int iSymbol = 0x1a55;`

int型は32ビットのため、iSymbol 0000000000000000000000001101001010101と表示されます。

詳細ペインでは、set output-radixコマンドの設定により、表示形式(10進数/16進数)が変わります。

●一覧形式

[式]ビューは、以下の表示を行います。

表10.4.7.3.1 表示方法

表示方法	切り替え表示方法	特徴	
一覧形式	ビューメニューから [列の表示]非選択	表示	「変数=値」の形式で表示します。
		ハイライト	ハイライトはされません。
		値の変更	コンテキストメニュー > [値の変更]で編集します。

表形式表示はありません。

注：floatやdoubleなどの浮動小数点型の値を参照するときは、[フォーマット]を[自然]に設定してください。([フォーマット]が、[16進数]その他に設定されている場合は、指数部+仮数部の形式で表示されます。)

10.4.7.4 操作

●ビューを開く/閉じる

[式]ビューは、[ウィンドウ]>[ビューの表示]から開いてください。"10.4.1.3 ビューを開く/閉じる"を参照してください。ビューのxボタンをクリックして閉じるができます。

●監視式(シンボル)の登録

[式]ビューに式を表示させるには、式の登録が必要です。

登録は以下のいずれかの方法で行います。

・ [式]ビューのコンテキストメニュー /ボタン

[監視式を追加...] :

ダイアログが表示されます。

ボックスに監視式・シンボル名などを入力します。改行を入力しないでください。

[使用可能にする]チェックボックスはONのままとしてください

[OK]ボタンを押下しますと、[式]ビューに式が登録されます。

[キャンセル]ボタンを押下しますと、登録せずにダイアログを閉じます。

・ [ソース]エディターのコンテキストメニュー

[監視式を追加...] :

"10.4.3 [ソース]エディター"の「●監視式の登録」を参照してください。

・ [変数]ビューのコンテキストメニュー

[監視式の作成] :

"10.4.6 [変数]ビュー"の「●監視式の登録」を参照してください。

・ [レジスター]ビューのコンテキストメニュー

[監視式の作成] :

"10.4.8 [レジスター]ビュー"の「●監視式の登録」を参照してください。

無効な式を登録した場合

無効な監視式を入力した場合、デバッガではその評価ができなかったことになります。

その場合には[式]ビューにエラーのまま登録されます。

監視式を正しい内容に編集してください。

ローカル変数を登録した場合

ローカル変数を登録した場合は、その変数が関数内に存在すれば値を参照できます。

関数外に移動すると、無効な監視式として扱われます。

配列要素を選択して登録した場合

[式]ビューで配列変数の要素を選択して[監視式の作成]から監視式を登録した場合、

例: `((symbol)+0)@10` [2]

のような式が登録されます。

式の読み方としては以下の通りです。

symbol+0のアドレスの

@10要素の配列の

[2] 番目の要素

を指す値

●監視式(シンボル)の編集

[監視式を編集...] :

[監視式を編集...]を選択します。

編集したい監視式が入力された状態でダイアログが開きます。

ボックスに監視式・シンボル名などを入力します。改行を入力しないでください。

[使用可能にする]チェックボックスはONのままとしてください

[OK]ボタンを押下しますと、[式]ビューの式が変更されます。

[キャンセル]ボタンを押下しますと、登録せずにダイアログを閉じます。

●監視式(シンボル)の再評価

[監視式を再評価]：

監視式を[使用不可]状態から[使用可能にする]状態に変更したときなどに、監視式の評価結果を直ちに行います。評価したい式を選択して[監視式を再評価]を選択します。

●監視式(シンボル)の削除

監視が不要となった式は次の方法でウィンドウから削除します。

削除する式をクリックし、コンテキストメニュー>[除去]または、ツールバー>[選択された式の除去]ボタンを選択します。

[すべて除去]を選択すると、すべての式が除去されます。

●監視式の保存/復元

登録した監視式は、**IDE**を終了したあとも自動的に保存され、**IDE**を次回起動したあとも復元されます。デバッガを再度起動したときに、登録されている監視式の評価が行われます。

●式の値の変更

登録した監視式(アイコン)は値を変更できません。式の値(変数(=アイコンもしくはポインタ アイコン)が付いた子要素のみ値を変更できます。

変更する式の値(変数(=アイコンもしくはポインタ[]アイコン)をクリックして選択します。

コンテキストメニュー>[値の変更]を選択します。

編集ダイアログが開き、値を編集して[OK]を押下します。

値は10進数または16進数(0xつき)で入力します。

注：[式]ビューでは、追加した式の値を変更することはできません。値の変更には[変数]ビューを使用してください。

●表示形式の変更

・コンテキストメニュー>[フォーマット]

表示形式を以下の中から選択できます。

表10.4.7.3.1 表示形式一覧

選択	表示形式	デフォルト
バイナリー	2進数	
自然	符号付き10進数	○
10進数	符号付き10進数	
16進数	16進数	

表示形式の変更は、選択項目すべてに適用されます。

●メモリでの表示

[View Memory]：

特定の変数の値を[メモリー]ビューに登録して表示することができます。

登録したい変数のコンテキストメニュー>[View Memory]から、[メモリー]ビューに登録・表示されます。

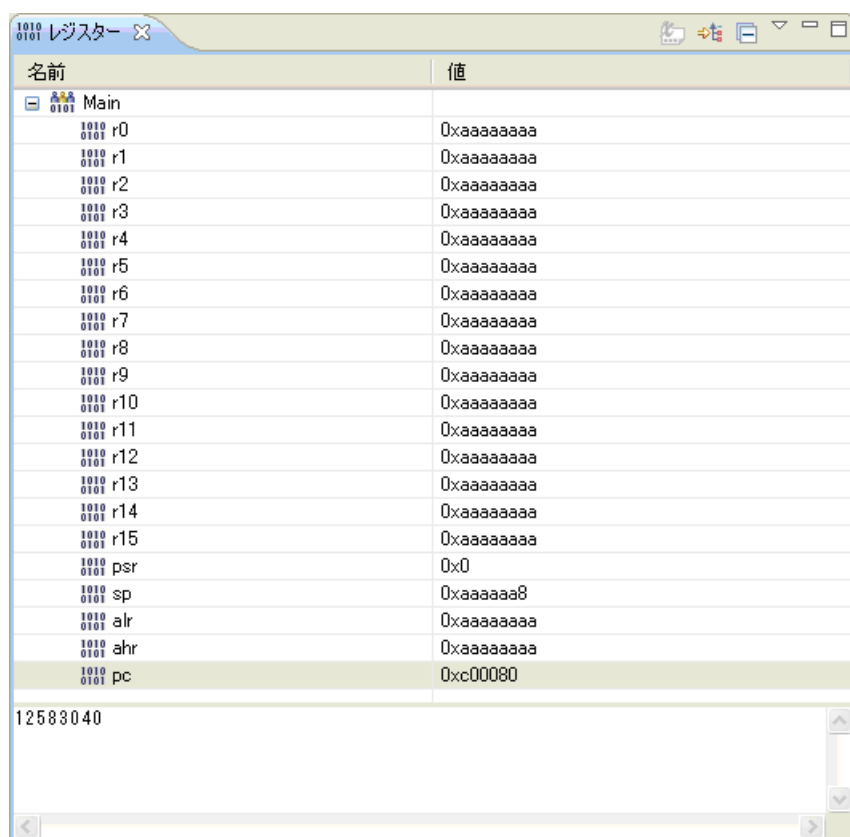
10.4.7.5 制限事項

- [式]ビューでは、追加した式の値を変更することはできません。値の変更には[変数]ビューを使用してください。
- [式]ビューには、ローカル変数は登録しないでください。ローカル変数の値は[変数]ビューで確認できます。
- 変数の内容をコマンド等で修正しても表示を更新しません。
- プログラム停止時に値が更新されないことがあります。この場合は詳細ペインで値を確認してください。

10.4.8 [レジスター]ビュー

[レジスター]ビューは、CPUレジスタの値の表示と修正に使用します。

10.4.8.1 画面構成



10.4.8.2 メニュー / ツールバー

● ツールバー

表10.4.8.2.1 ツールバー

ボタン	機能
	型名の表示
	論理構造の表示
	すべて縮小表示
	サポートしていません。
	サポートしていません。
	表示しているレジスター一覧をたたみます。

● ビューメニュー

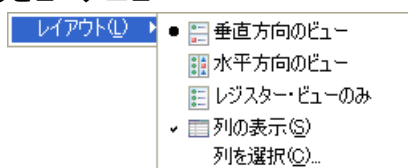


表10.4.8.2.2 ビューメニュー

メニュー	機能
レイアウト	ビュー表示のレイアウトを変更します。
垂直方向のビュー	詳細ペインを縦に表示します。
水平方向のビュー	詳細ペインを横に表示します。
レジスター・ビューのみ	詳細ペインを表示しません。
列の表示	表形式での表示を切り替えます。
列の選択...	サポートしていません。

●コンテキストメニュー

すべて選択(A)	Ctrl+A
レジスターのコピー(B)	Ctrl+C
☒ 使用可能にする	
☐ 使用不可	
×[I] 配列として表示...	
型にキャスト...	
オリジナル型の復元	
View Memory	
検索(E)...	Ctrl+F
値の変更(C)...	
レジスター・グループの追加	
デフォルト・レジスター・グループの復元	
監視式の作成(I)	
フォーマット	

表10.4.8.2.3 コンテキストメニュー

メニュー	機能
すべて選択	ビューの表示を全選択します。
レジスターのコピー	選択した内容をコピーします。
使用可能にする	レジスタの更新が行われるようにします。
使用不可	レジスタの更新を行いません。
配列として表示...	サポートしていません。
型にキャスト...	サポートしていません。
オリジナル型の復元	サポートしていません。
View Memory	レジスタの値のアドレスを[メモリー]ビューで表示します。 レジスタグループ名を選択した場合は表示されません。
検索...	レジスタを検索します。
値の変更...	レジスタの値を変更するダイアログを開きます。 レジスタグループ名を選択した場合は表示されません。
レジスター・グループの追加	特定のレジスタのみを表示するレジスタグループを作成します。
デフォルト・レジスター・グループの復元	デフォルトのレジスタグループの表示に戻します。 作成したレジスタグループも削除されます。
レジスター・グループの編集	レジスタグループを編集します。 レジスタグループを選択したときのみ表示されます。
レジスター・グループの除去	レジスタグループを削除します。 レジスタグループを選択したときのみ表示されます。
監視式の作成	レジスタを[式]ビューに登録します。 レジスタグループ名を選択した場合は表示されません。
フォーマット	表示形式を変更します。 レジスタグループ名を選択した場合は表示されません。
バイナリー	2進数
自然	符号付き10進数
10進数	符号付き10進数
16進数	16進数

注：詳細ペインのコンテキストメニューからの操作はサポートしていません。

10.4.8.3 表示内容

[レジスタ]ビューは、レジスタの内容を一覧表示します。

●レジスタ一覧

レジスタを表示するには、[デバッグ]ビューでスタックフレームを選択する必要があります。レジスタ値は、ターゲットプログラムが停止したときに更新されます。更新されたレジスタは、ハイライト表示されます。

表10.4.8.3.1 各CPUのレジスタ一覧

コア	レジスタ種類
C33 STD	r0-r15, psr, sp, alr, ahr, pc
C33 ADV	r0-r15, psr, alr, ahr, pc, lco, lsa, lea, sor, ttbr, dp, usp, ssp
C33 PE	r0-r15, psr, sp, alr, ahr, pc, ttbr

上記のレジスタは、Mainレジスタグループとしてツリー表示されます。

●表形式/一覧形式

[レジスタ]ビューは、2通りの表示方法が選択できます。

表10.4.8.3.2 表示方法

表示方法	切り替え表示方法	特徴	
表形式	ビューメニューから [列の表示]選択	表示	[名前]列と[値]列に分かれて表示します。
		ハイライト	変更箇所はハイライト表示されます。
		値の変更	[値]列から直接編集できます。
一覧形式	ビューメニューから [列の表示]非選択	表示	レジスタ=値 の形式で表示します。
		ハイライト	変更箇所はハイライト表示されます。
		値の変更	コンテキストメニュー>[値の変更...]で編集します。

ハイライトの色は、「10.4.1.6 設定の変更」●[実行/デバッグ]」の[変更された値の色]または[変更された値の背景色]で変更可能です。

注：[レジスタ]ビューの表示されていない部分は、値が変更されてもハイライトされません。

●詳細ペイン表示

ビューは通常2分割されており、詳細ペインが表示されます。詳細ペインには、選択されているレジスタの値のみが表示されます。

set output-radixコマンドの設定により、表示形式(10進数/16進数)が変わります。

詳細表示ペインのレイアウトは、ビューメニュー>[レイアウト]から以下のように変更できます。

表10.4.8.3.3 詳細ペイン表示方法

パス	レイアウト
垂直方向のビュー	詳細ペインを縦に表示します。
水平方向のビュー	詳細ペインを横に表示します。
レジスタ・ビューのみ	詳細ペインを表示しません。

●PSRレジスタ詳細表示

PSRレジスタを選択すると、レジスタの各フラグの値が詳細ペインに表示されます。

PSRレジスタの内容は、デバッグするコアによって変わります。

表10.4.8.3.4 PSRレジスタの詳細一覧

コア	PSRフラグ
C33 STD	IL: 割り込みレベル MO: MACオーバーフローフラグ DS: 被除数符号フラグ IE: 割り込み許可 Z: ゼロフラグ N: ネガティブフラグ C: キャリーフラグ V: オーバーフローフラグ

コア	PSRフラグ
C33 ADV	HE: HALT/SLEEPイネーブル RM: リピートモードイネーブル LM: ループモードイネーブル PM: PUSH/POPモード RC: レジスタカウンタ SW: スキャンワードイネーブル OC: オーバーフロークリアイネーブル SE: キャリー付きシフトイネーブル LC: ALRチェンジイネーブル HC: AHRチェンジイネーブル S: 飽和 DE: デバッグ例外 ME: MMU例外 SV: スーパーバイザモード IL: 割り込みレベル MO: MACオーバーフローフラグ DS: 被除数符号フラグ IE: 割り込み許可 Z: ゼロフラグ N: ネガティブフラグ C: キャリーフラグ V: オーバーフローフラグ
C33 PE	IL: 割り込みレベル IE: 割り込み許可 Z: ゼロフラグ N: ネガティブフラグ C: キャリーフラグ V: オーバーフローフラグ

表10.4.8.3.5 各フラグの表示方法

フラグ	表示方法
IL	0から始まる整数数値 例) IL: 7
それ以外のフラグ	立っている場合は1、下がっている場合は0 例) Z: 0 N: 0 C: 1 V: 1

10.4.8.4 操作

●ビューを開く/閉じる

[レジスター]ビューは、[ウィンドウ]>[ビューの表示]から開いてください。"10.4.1.3 ビューを開く/閉じる"を参照してください。ビューのxボタンをクリックして閉じるができます。

●表示形式の変更

・コンテキストメニュー>[フォーマット]

表示形式を以下の中から選択できます。

表10.4.8.4.1 表示形式一覧

選択	表示形式	デフォルト
バイナリー	2進数	
自然	符号付き10進数	
10進数	符号付き10進数	
16進数	16進数	○

表示形式の変更は、選択項目すべてに適用されます。

●レジスタデータの変更

[値の変更]：

レジスタの値を変更することができます。

内容を変更するレジスタを選択し、[値の変更]を選択します。

ダイアログが表示され、値を編集して[OK]を押下します。

[値]列：

[レジスター]ビューを表形式で表示中の場合、[値]列から直接値を編集することができます。レジスタサイズを超える値を入力した場合、下位32ビットが有効となり、33ビット目以上は無視されます。

●メモリでの表示

[View Memory]：

特定のレジスタの値を[メモリー]ビューに登録して表示することができます。

登録したいレジスタのコンテキストメニュー>[View Memory]から、[メモリー]ビューが開き"\$r0"(R0レジスタの場合)のようにレジスタが[メモリー]ビューに登録・表示されます。

●監視式の登録

[監視式の作成]：

特定のレジスタの値を[式]ビューに登録して表示することができます。

登録したいレジスタのコンテキストメニュー>[監視式の作成]から、[式]ビューが開き"\$r0"(R0レジスタの場合)のようにレジスタが[式]ビューに登録・表示されます。

●レジスタのグループ化

よく参照するレジスタのみをグループ化して表示することができます。

デフォルトで表示される全レジスタは、Mainというグループで登録されています。

[レジスター・グループの追加]：

グループ化したいレジスタを一覧から選択し、名前をつけます。

登録したグループがツリー表示されます。

[デフォルト・レジスター・グループの復元]：

登録したグループをすべて削除し、Mainのグループ(全レジスタの一覧)を復元します。

注： [使用不可]にしたレジスタは[使用可能にする]に変わりますが、[フォーマット]の設定は変わりません。

[レジスター・グループの編集]：

登録したグループを編集してレジスタを追加・削除できます。(グループ名は変更できません)

10 デバッグ

- ・ [レジスター・グループの除去]：
登録したグループを削除します。

10.4.8.5 制限事項

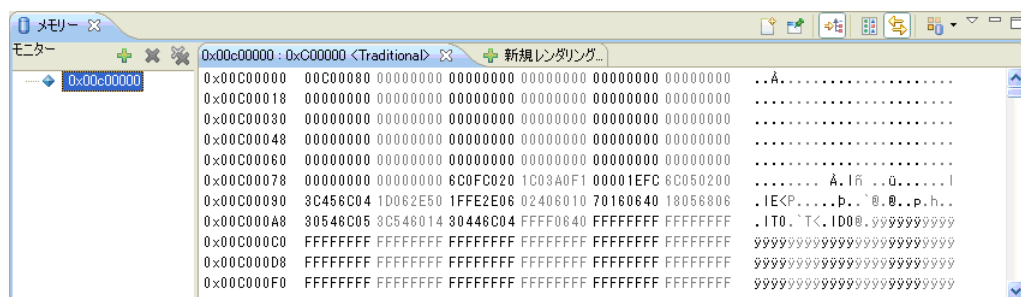
- ・ レジスタの内容をコマンドで修正した場合、表示が更新されません。
- ・ デバッグ起動直後[レジスター]ビューに初期表示されるレジスタは、ビューの表示領域によって変わります。このとき[すべて選択]の操作を行うと、現在表示されているレジスタのみが選択されます。

10.4.9 [メモリー]ビュー

[メモリー]ビューは、メモリ内容の表示と修正に使用します。

10.4.9.1 画面構成

左側にメモリモニタペイン（アドレス一覧）、右側にメモリレンダリングペイン（メモリデータ）に分割表示します。右側ペインは、アドレス部/データ部/ASCII部に分かれて表示します。



10.4.9.2 メニュー / ツールバー

● ツールバー

表10.4.9.2.1 ツールバー

ボタン	機能
	新規に[メモリー]ビューを開きます。
	[メモリー・モニターの追加]で新しいメモリアドレスを登録したときのメモリモニタ(左)ペインの選択状態をピン留めします。 ONのときは、登録したときでも現在選択されているアドレスが保持されます。 OFFのときは、登録したアドレスに切り替わります。
	メモリモニタ(左)ペインの表示/非表示を切り替えます。
	メモリレンダリング(右)ペインの2分割表示を切り替えます。
	サポートしていません。
	複数のメモリモニタ(表示アドレス)を切り替えます。

● ビューメニュー

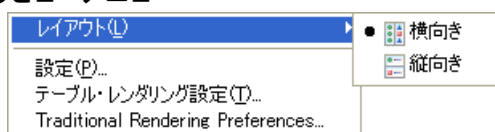


表10.4.9.2.2 ビューメニュー

メニュー	機能
レイアウト	ビュー表示のレイアウトを変更します。
横向き	メモリモニタとメモリレンダリングペインを横に表示します。
縦向き	メモリモニタとメモリレンダリングペインを縦に表示します。
設定...	[メモリー]ビュー全体の設定ダイアログを開きます。
テーブル・レンダリング設定...	サポートしていません。
Traditional Rendering Preferences...	メモリレンダリング(右ペイン)の設定を行います。

●コンテキストメニュー

- ・メモリモニタ(左ペイン)のコンテキストメニュー

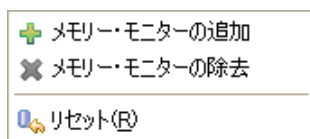


表10.4.9.2.3 コンテキストメニュー(メモリモニタ)

メニュー	機能
メモリ・モニターの追加	新しいメモリアドレスをメモリモニタに登録します。
メモリ・モニターの除去	選択したメモリアドレスをメモリモニタから削除します。
リセット	選択したメモリアドレスまでメモリの表示をリセットします。

- ・メモリレンダリング(右ペイン)のコンテキストメニュー



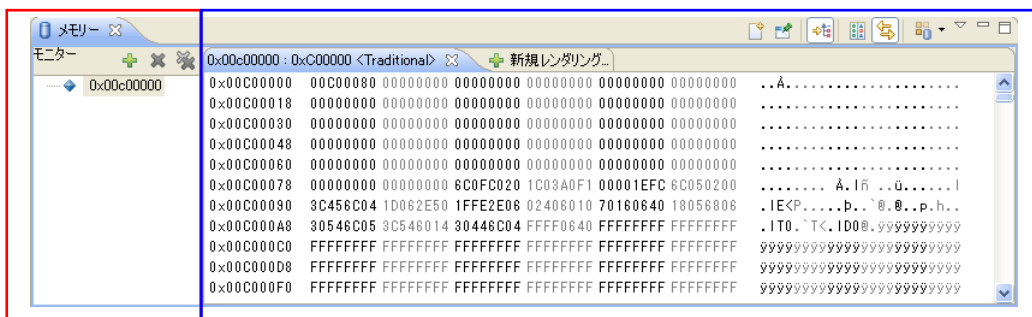
表10.4.9.2.4 コンテキストメニュー(メモリレンダリング)

メニュー	機能
監視ポイントの追加...	データブレークを設定する[監視ポイントの追加]ダイアログを開きます。
レンダリングの追加	メモリを表示するための表示形式を追加します。
レンダリングの除去	選択中のメモリ表示を削除します。
Panes	アドレス部/データ部/ASCII部の、表示/非表示を切り替えます。
Address	アドレス部
Binary	データ部
Text	ASCII部
Endian	リトルエンディアンとビッグエンディアンでの表示を切り替えます。
Big	ビッグエンディアン
Little(デフォルト)	リトルエンディアン
Text	ASCII部のエンコーディングを切り替えます。
ISO-8859-1(デフォルト)	左記のいずれかで表示します。
US-ASCII	
UTF-8.	
Cell Size	データ部の列ごとの表示バイト数を切り替えます。
1	1 byte
2	2 byte
4 (デフォルト)	4 byte
8バイト	8 byte
Radix	データ部の列ごとの表示フォーマットを切り替えます。
Hex(デフォルト)	16進
Decimal Signed	符号付10進
Decimal Unsigned	符号なし10進
Octal	8進
Binary	2進

メニュー	機能
Columns	データ部の列数を切り替えます。 (データは1行にCell Size×Columnsバイト表示されるようになります。)
Auto Size to Fill	列数ビューのリサイズにあわせます。
1 ~ 128	列数を指定します (1/2/4/8/16/32/64/128)。
Custom...	数値を入力して列数を指定します。
クリップボードにコピー	選択した部分をコピーします。
Copy Address	カーソル位置の境界アドレスをコピーします。
Reset To Base Address	データ部の表示を、メモリモニタに登録した時点のアドレス位置へ戻します。

10.4.9.3 表示内容

●メモリの表示



メモリモニタ ペイン (アドレス一覧)	メモリレンダリング ペイン (メモリデータ) アドレス部 データ部 ASCII部に分れます

[メモリー]ビューはメモリ領域のダンプ結果を表示します。
[メモリー]ビューは以下の2つのペインに分けて表示されます。

表10.4.9.3.1 各ペインの表示内容

ペイン	説明
メモリモニタペイン(左ペイン)	メモリ表示の開始位置となるベースアドレスを登録します。 複数のアドレスを登録することができます。
メモリレンダリングペイン(右ペイン)	メモリモニタペインで選択したアドレスに対応するアドレスを、アドレス部・データ部・ASCII部に分けて表示します。 メモリレンダリングペインの表示形式は、コンテキストメニューの各設定より変更可能です。 データ部のエンディアンは、表示形式の設定によります。 (パラメータファイルでの指定は使用しません。)

10.4.9.4 操作

●ビューを開く/閉じる

[メモリー]ビューは、[ウィンドウ]>[ビューの表示]から開いてください。"10.4.1.3 ビューを開く/閉じる"を参照してください。ビューのxボタンをクリックして閉じるができます。

●表示アドレスの設定

[メモリー・モニターの追加]：

デフォルトでは、[メモリー]ビューを開いたときに表示するメモリアドレスはありません。[メモリー]ビューの+アイコンを押下すると[モニター・メモリー]ダイアログが開きます。表示を開始したいアドレスをテキストボックスに入力し、OKボタンを押下すると、メモリモニタペイン(左ペイン)の一覧にアドレスが追加され、メモリレンダリング(右ペイン)には、そのアドレスのメモリ領域が表示されます。アドレスの指定は16進数(0xを頭に付加)、10進数、グローバルシンボルもしくは任意の式を入力できます。

数値を指定した場合、下位32ビットが有効です。

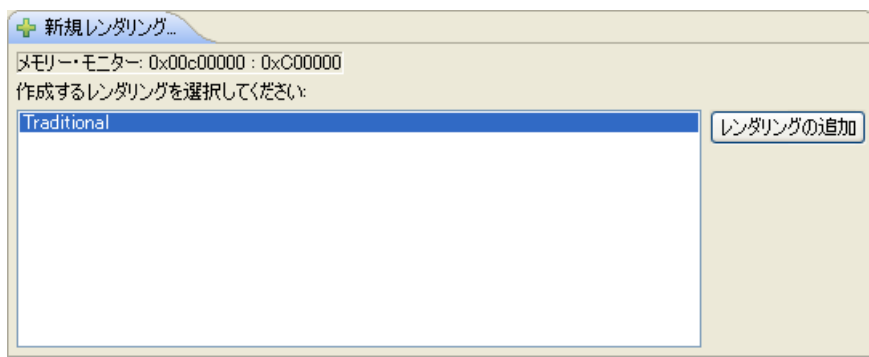
シンボルを指定した場合、シンボルの値がアドレスとして用いられます。シンボルが配置されているアドレスを指定する場合は、シンボルの前に&を付けてください。

また、メモリレンダリング(右ペイン)の表示は2回目以降のデバッグ起動時は[レンダリングの追加]を押下して表示しなおす必要があります。

[レンダリングの追加]：

[メモリー・モニターの追加]からアドレスを登録すると、メモリレンダリング(右ペイン)にメモリ領域が表示されますが、閉じてしまった場合は、以下の手順で開きます。

1. [新規レンダリング]タブを選択します。
2. 一覧から[Traditional]を選択します。
3. [レンダリングの追加]ボタンを押下します。



[レンダリングの除去]：

[レンダリングの追加]より追加したアドレスはメモリモニタペイン(左ペイン)に表示されます。

メモリ表示を削除する場合は、アドレスを選択して[レンダリングの除去]を押下すると該当するメモリ表示を削除できます。

[すべて除去]を押下するとすべてのメモリ表示が削除されます。

●他ビューからアドレス登録

他のビューから[メモリー]ビューにアドレスを登録することができます。

登録は以下のいずれの方法から可能です。

- ・ [変数]ビューのコンテキストメニュー>[View Memory]
- ・ [式]ビューのコンテキストメニュー>[View Memory]
- ・ [レジスター]ビューのコンテキストメニュー>[View Memory]

●スクロール

メモリレンダリング(右ペイン)のスクロールバー /Page Up/Page Downキーで、表示されていないメモリ領域を表示することができます。

[Reset To Base Address]ボタンを押下することで、データ部の表示をメモリモニタに登録した時点のアドレス位置に戻します。

注：連続的なスクロールを行うとパフォーマンスが落ちることがあります。

●メモリデータの変更

メモリデータの変更は、以下の手順で行います。

1. 変更したいデータの位置にカーソルをあわせめます。データ部もしくはASCII部で値を変更可能です。
2. 新しい値を入力します。入力上書きモードのように入力されます。入力形式は、現在の表示形式(16進/10進/文字コードなど)によります。
3. 入力されると、新しい値の背景色が変わります。(明るい緑)

```
0x00C0008C  FFFF0000 CC18A415
```

4. 値を確定する場合は、[Enter]を押下します。

確定された値は、背景色が変わります。(赤)

```
0x00C0008C  FFFF0000 CC18A415
```

5. 値を確定しない場合は、入力位置からカーソルをはずし、[Esc]を押下します。

```
0x00C0008C  0000p0000 CC18A415
```

値入力時は、以下のキーは使用できません。

- [Backspace]
- [Delete]
- [Insert]

誤って入力した場合は、カーソルを戻してから入力し直してください。矢印キーはカーソル移動に使用できます。

注：[メモリー]ビューの表示されていない部分は、値が変更されてもハイライトされません。

●表示形式の変更

メモリレンダリング(右ペイン)で表示されるメモリデータのコンテキストメニューから、各種表示形式を設定できます。

表10.4.9.4.1 表示形式一覧

表示形式		機能
Panels	Address	アドレス部
	Binary	データ部
	Text	ASCII部
		アドレス部/データ部/ASCII部の、表示/非表示を切り替えます。
Endian		リトルエンディアンとビッグエンディアンでの表示を切り替えます。
	Big	ビッグエンディアン
	Little(デフォルト)	リトルエンディアン
Text		ASCII部のエンコーディングを切り替えます。
	ISO-8859-1(デフォルト)	左記のいずれかで表示します。
	US-ASCII	
	UTF-8	
Cell Size		データ部の列ごとの表示バイト数を切り替えます。
	1	1 byte
	2	2 byte
	4(デフォルト)	4 byte
	8バイト	8 byte
Radix		データ部の列ごとの表示フォーマットを切り替えます。
	Hex(デフォルト)	16進
	Decimal Signed	符号付10進
	Decimal Unsigned	符号なし10進
	Octal	8進
	Binary	2進

表示形式	機能
Columns	データ部の列数を切り替えます。 (データは一行にCell Size×Columns/バイト表示されるようになります。)
Auto Size to Fill	列数ビューのリサイズにあわせてます。
1 ~ 128	列数を指定します (1/2/4/8/16/32/64/128)。
Custom...	数値を入力して列数を指定します。

●設定の変更

[メモリー]ビューに関する設定は、ビューメニューより開くことができます。

・設定

[メモリー・モニターのリセット]

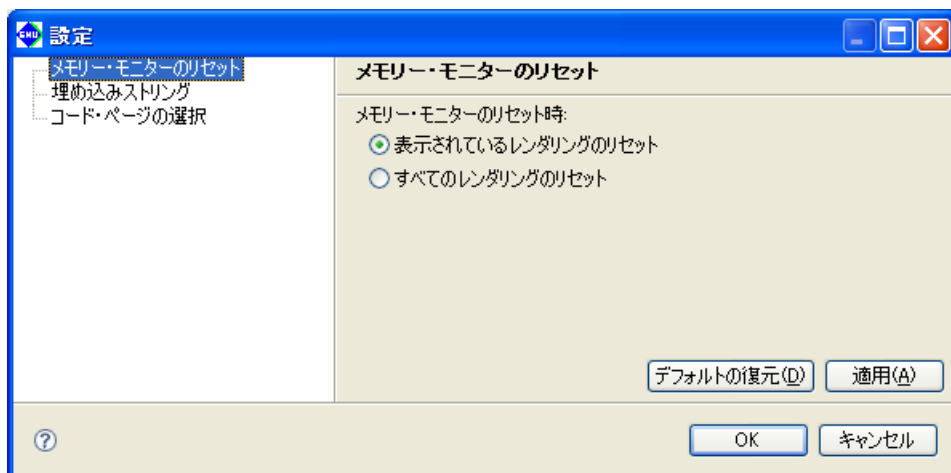


表10.4.9.4.2 [メモリー・モニターのリセット]ダイアログ

設定	設定内容
メモリー・モニターのリセット	メモリモニタ(左ペイン)で[リセット]したとき、メモリレンダリング(右ペイン)を複数開いている場合
表示されているレンダリングのリセット	現在表示中のメモリレンダリング(右ペイン)のみ表示をリセットする。通常、こちらを選択してください。
すべてのレンダリングのリセット	すべてのメモリレンダリング(右ペイン)の表示をリセットする。

[埋め込みストリング]

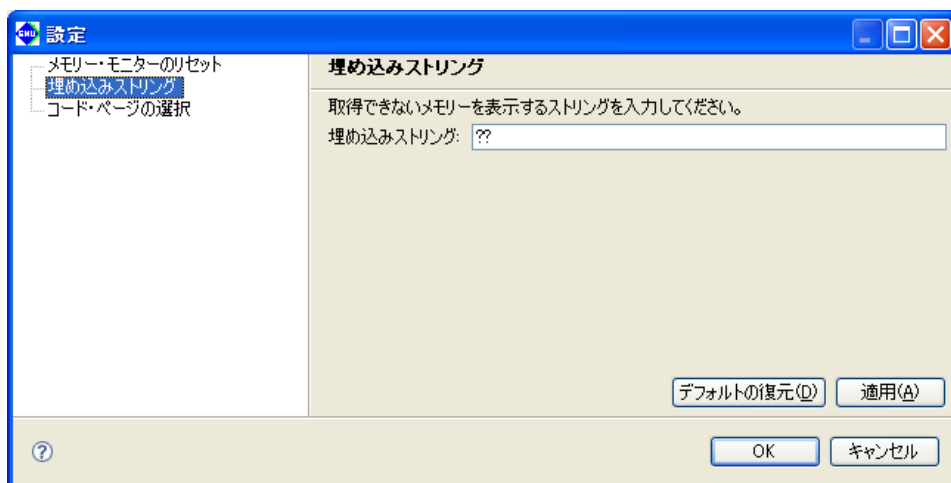


表10.4.9.4.3 [埋め込みストリング]ダイアログ

設定	設定内容
埋め込みストリング	表示できないメモリデータはこの文字列で表示されます。

[コード・ページの選択]
サポートしていません。

- Traditional Rendering Preferences
[Traditional Memory Reading]

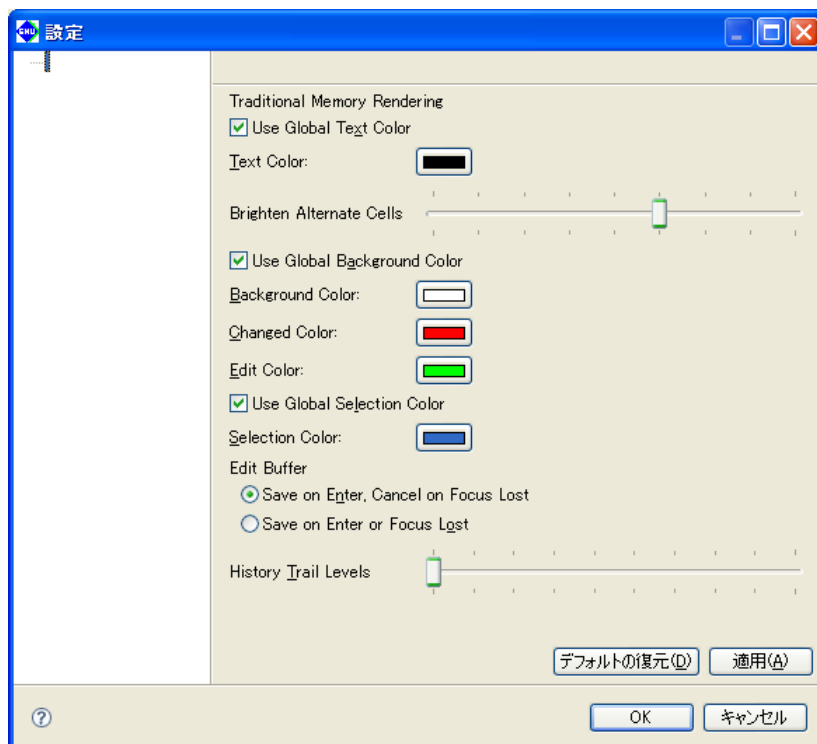


表10.4.9.4.4 [Traditional Memory Reading]ダイアログ

設定	設定内容
Use Global Text Color	文字列の色に、システム設定(黒)を使用します。
Text Color	文字列の色を指定します。 ([Use Global Text Color]がOFFのとき)
Brighten Alternate Cells	列ごとに文字のコントラストを変えます(左: 淡 右: 濃)
Use Global Background Color	背景色に、他のビューと同じ設定を使用します。
Background Color	背景色を指定します。 ([Use Global Background Color]がOFFのとき)
Changed Color	変更箇所の色を指定します。
Edit Color	編集箇所の色を指定します。
Use Global Selection Color	選択時の反転色に、他のビューと同じ設定を使用します。
Selection Color	選択時の反転色を指定します。 ([Use Global Selection Color]がOFFのとき)
Edit Buffer	編集方法を指定します。
Save on Enter, Cancel on Focus Lost	Enterで変更を確定し、入力位置からカーソルをはずすとキャンセルします。通常、こちらをご使用ください。
Save on Enter or Focus Lost	Enterもしくは入力位置からカーソルをはずすと変更を確定します。
History Trail Levels	サポートしていません。

●データブレイクポイントの設定

特定のメモリ領域に対しての書き込み/読み出しアクセスを監視するデータブレイクポイント(監視ポイント)を設定することができます。

[監視ポイントの追加...]:

右ペインのデータ部のコンテキストメニュー>[監視ポイントの追加...]から、データブレイクポイントを設定することができます。

詳細については、"10.4.5 [ブレイクポイント]ビュー "を参照してください。

10.4.9.5 制限事項

- ・ メモリアドレスの内容をコマンド等で修正しても、[メモリー]ビューの表示内容は変更されません。[メモリー]ビューをスクロールさせて更新してください。
- ・ [テーブル・レンダリング設定...]は、Eclipse付属の[メモリー]ビューを使用しないため、サポートしていません。

10.4.10 [コンソール]ビュー

[コンソール]ビューは、コマンドの実行と実行結果の表示に使用します。

また、シミュレーテッドI/Oの出力も表示します。"10.4.11 [シミュレーテッドI/O]ビュー"を参照してください。

本画面で入力可能なコマンドのコマンドリファレンスについては、"10.7 コマンドリファレンス"を参照してください。

10.4.10.1 画面構成



10.4.10.2 メニュー / ツールバー

● ツールバー

表10.4.10.2.1 ツールバー

ボタン		機能
	終了	コンソールが対応しているプロセスを終了します。
	起動の除去	コンソールが対応している[デバッグ]ビュー内のデバッグアイコンを削除します。
	終了したすべての起動を除去	終了済みのすべての[デバッグ]ビュー内のデバッグアイコンを削除します。
	コンソールのクリア	コンソールの表示をクリアします。
	スクロール・ロック	スクロールロックを切り替えます。
	標準出力に変更があった場合にコンソールを表示	標準出力があるときにコンソールをフォーカスします。
	標準エラーに変更があった場合にコンソールを表示	標準エラー出力があるときにコンソールをフォーカスします。
	Save console content	出力内容を保存します。
	コンソールのピン留め	現在表示中のコンソールをピン留め表示します。
	選択されたコンソールの表示	現在表示しているコンソールと前回表示したコンソールを切り替えます。付属のリストから、選択したコンソールに表示を切り替えることもできます。
	コンソールを開く	▼ボタンのリストから選択したコンソールを別のビューとして開きます。

● ビューメニュー

ビューメニューはありません。

●コンテキストメニュー

 切り取り(C)	Ctrl+X
 コピー(C)	Ctrl+C
 貼り付け(P)	Ctrl+V
 すべて選択(A)	Ctrl+A
検索/置換(F)...	
リンクを開く(O)	
 クリア(R)	
 終了したすべてを除去(A)	
 スクロール・ロック(S)	
設定(P)...	

表10.4.10.2.2 コンテキストメニュー

メニュー	機能
切り取り/コピー/貼り付け/すべて選択	コンソール出力文字列に対する各種編集操作を行います。
検索/置換...	コンソール内を検索します。
クリア	コンソールの表示をクリアします。
終了したすべてを除去	終了済みのすべての[デバッグ]ビュー内のデバッグアイコンを削除します。
スクロール・ロック	スクロールロックを切り替えます。
設定...	コンソールの設定ダイアログを開きます。

10.4.10.3 表示内容

●プロンプト

コマンドの入力が可能な状態になると、次のプロンプトを表示します。
(gdb)

|

コマンドを入力して実行すると、その結果を表示します(結果出力機能のあるコマンドのみ)。表示されるコマンドの実行結果については各コマンド説明を参照してください。(gdb) プロンプトの次の行にコマンドを入力してください。

なお、(gdb) プロンプトは、連続で出力されることがありますが、最後に表示された(gdb) プロンプトの次の行でコマンドを入力してください。

標準入力・標準出力・標準エラー出力はそれぞれ色別に表示されます。これらの色の設定は[設定...]より変更可能です。

10.4.10.4 操作

●ビューを開く/閉じる

[コンソール]ビューは、[ウィンドウ]>[ビューの表示]から開いてください。"10.4.1.3 ビューを開く/閉じる"を参照してください。ビューのxボタンをクリックして閉じるができます。

注：[コンソール]ビューを閉じてしまうと、コマンド入力ができなくなります。

[コンソール]ビューを閉じてしまった場合は、[ウィンドウ]>[ビューの表示]>[コンソール]から開き、[デバッグ]ビューのデバッグプロセス(gdb.exe)のアイコンをクリックし、[コンソールのピン留め]ボタンを押下してからコンソールをピン留めしてください。

デバッグ起動直後に(gdb)のコンソールが表示されていない場合は、一旦[コンソール]ビューの[コンソールのピン留め]からピン留めをはずし、[デバッグ]ビューのデバッグプロセス(gdb.exe)のアイコンをクリックし、[コンソールのピン留め]ボタンを押下してからコンソールをピン留めしてください。

●コマンド入力

[コンソール]ビューでは、デバッグコマンドを入力し実行させることができます。

[コンソール]ビューの最終行にプロンプト"(gdb)"が表示され、キーボードからのコマンド入力を受け付けます。

[デバッグ]ビューでデバッガプロセス(gdb.exeのアイコン)を選択することにより[コンソール]ビューがアクティブになり、GDBのコマンドが入力可能な状態となります。

注：・[デバッグ]ビューでデバッガプロセス(gdb.exeのアイコン)を選択しないとコマンド入力できません。

・以下の各コマンド入力後、ビューの表示が更新されます。(他のコマンドは更新されません。)

表10.4.10.4.1 ビューが更新されるコマンド一覧

カテゴリ	コマンド
ブレークポイント関連	break
	tbreak
	hbreak
	thbreak
	watch
	rwatch
	awatch
	info breakpoints
ステップ実行関連	step
	stepi
	next
	nexti
	finish
	continue
	until
CPUリセット関連	c33 rsth
	c33 rstc
	c33 rstt (ICD6のみ)

●編集に関する操作

[切り取り/コピー/貼り付け/すべて選択]：

[コンソール]ビューの文字列の切り取り・コピー・ペースト・全選択が行えます。

[検索/置換]：

[コンソール]ビューの文字列の検索が行えます。

注：置換は行えません。

[コンソールのクリア]：

[コンソールのクリア]ボタンもしくはコンテキストメニュー>[クリア]から、現在のコンソール出力がクリアされます。

[Save console content]：

[Save console content]ボタンより、現在のコンソール出力をファイルに保存することができます。

●表示に関する操作

[コンソールのピン留め]：

[コンソール]ビューでは複数のコンソールを1つのビュー内で表示しますが、このボタンを選択することで、現在表示されているコンソールをピン留めして常に表示状態にしておくことができます。

シミュレーテッドI/Oなど他のコンソールへの出力があったときにコマンド入力用のコンソールが裏に隠れないようにすることができます。

[選択されたコンソールの表示]：

[コンソール]ビューでは複数のコンソールを1つのビュー内で表示しますが、表示するコンソールを切り替えることができます。

[スクロール・ロック]：

スクロールをロックすることができます。

[標準出力に変更があった場合にコンソールを表示]：

[標準エラーに変更があった場合にコンソールを表示]：

それぞれのボタンを押し込むことで、標準出力もしくは標準エラー出力が書き込まれたときに[コンソール]ビューにフォーカスが移るようになります。

●プロセスの終了

[デバッグ]ビューと共通の処理を[コンソール]ビューから行うことができます。

[終了]：

デバッグ(GDB)を終了します。[デバッグ]ビューの表示も終了状態に変わります。

[起動の除去]：

[終了したすべての起動を除去]：

[デバッグ]ビューで、終了したデバッグの表示を削除します。

●設定の変更

[設定...]メニューよりコンソールの設定を変更することができます。

・[コンソール]

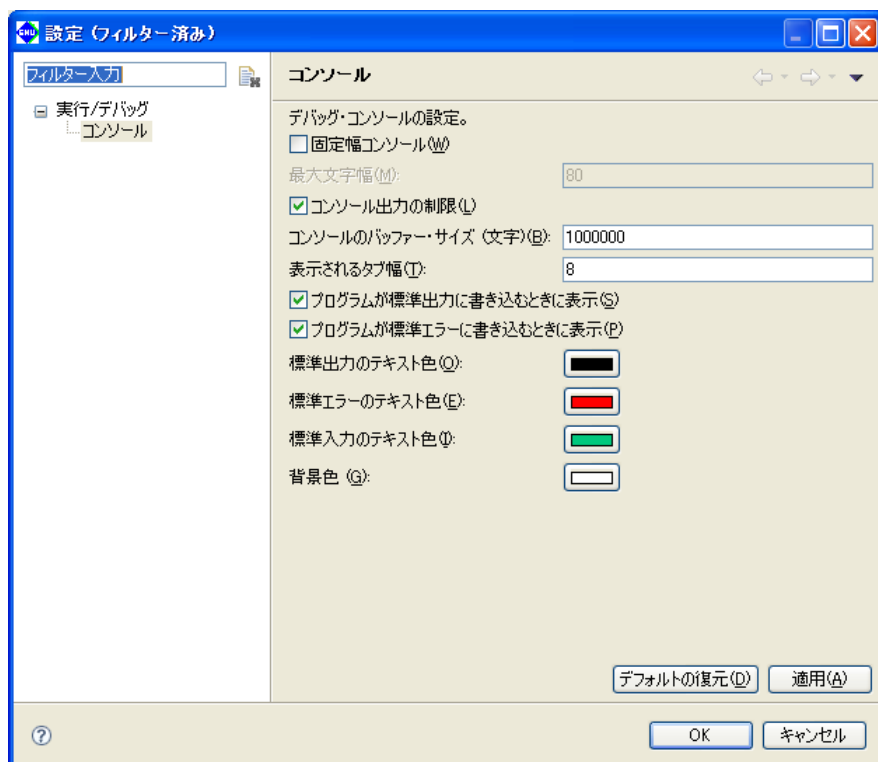


表10.4.10.4.2 [コンソール]設定ダイアログ

設定	設定内容
固定幅コンソール	コンソール幅を固定します。OFFのままご使用ください。
最大文字幅	幅を指定します。
コンソール出力の制限	出力バッファを制限します。
コンソールのバッファ・サイズ(文字)	バッファサイズを文字数で指定します。
表示されるタブ幅	タブ幅を指定します。
プログラムが標準出力に書き込むときに表示	出力があるときにコンソールをフォーカスします。
プログラムが標準エラーに書き込むときに表示	エラー出力があるときにコンソールをフォーカスします。
標準出力のテキスト色	出力の文字色を設定します。
標準エラーのテキスト色	エラー出力の文字色を設定します。
標準入力のテキスト色	入力の文字色を設定します。
背景色	背景色の文字色を設定します。

10.4.10.5 制限事項

- ・最大出力文字数は、[コンソールのバッファーク・サイズ]の設定+8000文字です。

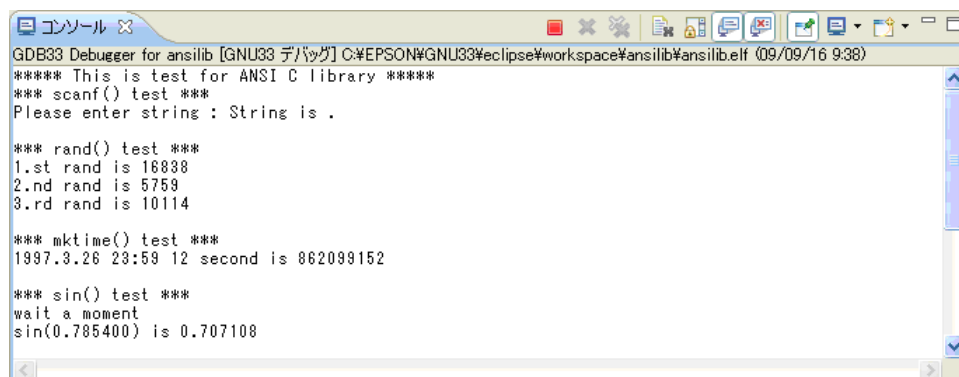
10.4.11 [シミュレーテッドI/O]ビュー

シミュレーテッドI/O機能の出力は、[コンソール]ビューに表示されます。

[デバッグ]ビューでターゲットプログラム(e1fのアイコン)選択時に[コンソール]ビューがアクティブになり、シミュレーテッドI/Oの出力画面が表示されます。

[コンソール]ビューには、シミュレーテッドI/Oの出力のみが表示され、キーボードからの入力はいけません。ファイルを使用した入力は使用可能ですので、入力を行いたい場合は、c33 stdinコマンドの引数にファイルを指定してください。

10.4.11.1 画面構成



10.4.11.2 メニュー / ツールバー

[コンソール]ビューのメニュー / ツールバーと同じです。

"10.4.10.2 メニュー / ツールバー"を参照してください。

10.4.11.3 表示内容

シミュレーテッドI/O機能のstdoutへの出力内容を表示します。

- 注：・ [コンソール]ビューを閉じてしまうと、シミュレーテッドI/O出力が表示されなくなります。[コンソール]ビューを閉じてしまった場合は、[ウィンドウ]>[ビューの表示]>[コンソール]から開き、[デバッグ]ビューのターゲットプログラムのアイコンをクリックして開いてください。
- ・ シミュレーテッドI/O使用時、printfなどの出力関数には必ず'¥n' (改行) を付与してください。改行が付与されていない行は[コンソール]ビューに出力することができません。

10.4.11.4 操作

[コンソール]ビューの以下の各操作が行えます。

●編集に関する操作

[切り取り/コピー /貼り付け/すべて選択]：

[コンソール]ビューの文字列の切り取り・コピー・ペースト・全選択が行えます。

[検索/置換]：

[コンソール]ビューの文字列の検索が行えます。

注：置換は行えません。

[コンソールのクリア]：

[コンソールのクリア]ボタンもしくはコンテキストメニュー>[クリア]から、現在のコンソール出力がクリアされます。

[Save console content]：

[Save console content]ボタンより、現在のコンソール出力をファイルに保存することができます。

●表示に関する操作

[コンソールのピン留め]：

[コンソール]ビューでは複数のコンソールを1つのビュー内で表示しますが、このボタンを選択することで、現在表示されているコンソールをピン留めして常に表示状態にしておくことができます。

シミュレーテッドI/Oなど他のコンソールへの出力があったときにコマンド入力用のコンソールが裏に隠れないようにすることができます。

[選択されたコンソールの表示]：

[コンソール]ビューでは複数のコンソールを1つのビュー内で表示しますが、表示するコンソールを切り替えることができます。

[スクロール・ロック]：

スクロールをロックすることができます。

[標準出力に変更があった場合にコンソールを表示]：

[標準エラーに変更があった場合にコンソールを表示]：

それぞれのボタンを押し込むことで、標準出力もしくは標準エラー出力が書き込まれたときに[コンソール]ビューにフォーカスが移るようになります。

なお、[コンソール]設定の変更がシミュレーテッドI/Oにも反映されます。

"10.4.10.4 操作"の「●設定の変更」を参照してください。

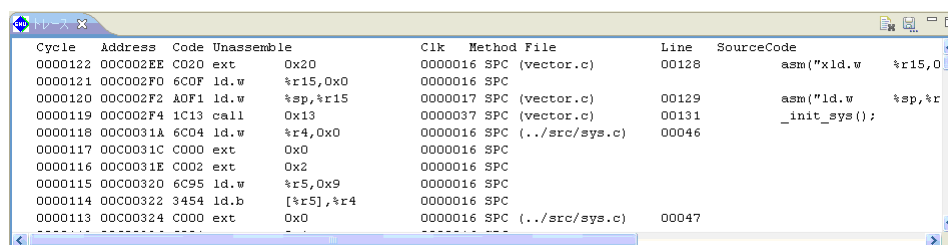
10.4.11.5 制限事項

- ・ [コンソール]ビューには、シミュレーテッドI/Oの出力のみが表示され、キーボードからの入力はいけません。ファイルを使用した入力の使用可能です。ファイルを使用した入力方法は、"10.7 コマンドリファレンス"の `c33 stdin` コマンドを参照してください。

10.4.12 [トレース]ビュー

[トレース]ビューでは、トレースデータを表示します。

10.4.12.1 画面構成



10.4.12.2 メニュー / ツールバー

● ツールバー

表10.4.12.2.1 ツールバー

ボタン		機能
	トレースのクリア	トレースの表示をクリアします。
	トレースの内容を保存	トレース出力内容を保存します。

● ビューメニュー

ビューメニューはありません。

● コンテキストメニュー

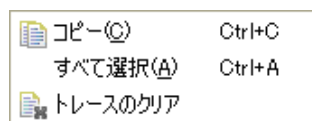


表10.4.12.2.2 コンテキストメニュー

メニュー	機能
コピー / すべて選択	出力文字列に対する各種編集操作を行います。
トレースのクリア	トレースの表示をクリアします。

10.4.12.3 表示内容

各命令の実行結果であるトレースデータを表示します。

トレースの開始・設定方法などは、“10.6.6 トレース機能”を参照してください。

トレースがONのときにデバッグ中にIDEがトレースデータをデバッガから受け取ると、[トレース]ビューが自動的に開き、データを表示します。

注：・一度に出力できるTraceデータは、“10.4.10.4 操作「●設定の変更」”の[コンソールのバッファ・サイズ] 1000000文字程度に制限されています。実行を開始した付近の古いデータは、表示されません。

・デバッグモニタモードにはトレース機能はありません。

●PCトレース

ICD2/ICD3/ICD6モードでは、S5U1C33000H/S5U1C33001HのトレースメモリからPCトレース結果を読み出して表示します。

PCトレース方式は「オーバーライドあり」「オーバーライドなしの全体トレース」「範囲トレース」の3種類に対応しています。

表示内容は次の通りです。

- ・ 実行サイクル番号
- ・ 実行アドレス・コード・逆アセンブル内容
- ・ クロック数
- ・ PC判定方式
- ・ ソース行番号・ソースコード

●PCトレース(シミュレータモード)

シミュレータモードでは、トレース機能をONにするとそれ以降のプログラム実行がすべてトレース表示されます。(ファイル出力を選択した場合を除く。)

表示内容は次の通りです。

- ・ 実行命令数
- ・ 実行アドレス・コード・逆アセンブル内容
- ・ メモリ内容(アドレス・R/W・データ)
- ・ ソースコード
- ・ レジスタ内容

10.4.12.4 操作

●ビューを開く/閉じる

[トレース]ビューは、[ウィンドウ]>[ビューの表示]から開いてください。"10.4.1.3 ビューを開く/閉じる"を参照してください。ビューのxボタンをクリックして閉じるができます。

●トレースデータ編集に関する操作

[トレースのクリア]：

[トレースのクリア]ボタンもしくはコンテキストメニュー>[トレースのクリア]から、現在の出力内容がクリアされます。

[トレースの内容を保存]：

[トレースの内容の保存]ボタンより、現在のトレース出力をファイルに保存することができます。

[コピー/すべて選択]

トレース出力の編集(コピー・全選択)が行えます。

10.4.12.5 制限事項

- ・ On Chip Bus Trace (C33 ADV)の表示は[コンソール]ビューからのコマンド入力により可能ですが、GUIでのトレース条件設定画面は対応しておりません。

10.5 コマンド実行方法

ここでは、コマンドを実行させる方法を説明します。コマンドのパラメータなど、詳細については各コマンド説明を参照してください。

10.5.1 コマンドのキーボード入力

コマンドは[コンソール]ビューに入力します。他のウィンドウが前面にでている場合は、[コンソール]ビューをクリックしてアクティブにしてください。[コンソール]ビューが表示されていない場合は、[ウィンドウ]>[ビューの表示]メニューから[コンソール]を選択してください。

●コマンド入力の一般形

(gdb)

コマンド [パラメータ [パラメータ ... パラメータ]]

コマンドとパラメータ間、およびパラメータ間にはスペースが必要です。

入力ミスの修正には矢印キー(←、→)、[Back Space]キー、[Delete]キーが使用できます。

最後に[Enter]キーを押すと、そのコマンドを実行します。

例: (gdb)

continue (コマンドのみの入力)

(gdb)

target icd usb (コマンドとパラメータの入力)

10.5.2 パラメータの入力形式

●数値入力

コマンドでアドレスやデータを指定するパラメータは、デフォルトで10進数を入力するように設定されています。16進数を入力するには、数値の先頭に0x(または0X)を付けてください。16進数として認められる文字は0～9、a～f、A～Fのみです。

ブレーク系コマンドで、即値アドレスを指定する場合は、次のように*をアドレス値の前に付けてください。

例: (gdb)
`break *0xc00040`

各コマンドの書式説明で、アドレスパラメータの前に*が付いていないものについては、この必要はありません。

●ソース行番号の指定

ブレーク系コマンドではソース行番号でブレークポイントを指定することができます。ただし、ソース行番号情報を含むelf形式のオブジェクトファイルをデバッグする場合に限られます。行番号の指定は次の形式で行います。

Filename:LineNo.

Filename ソースファイル名

*Filename:*は、カレントファイル(現在のPCに対応するコードを含むファイル)内の行番号を指定する場合は省略可能です。

LineNo. 行番号

行番号は10進数でのみ指定可能です。

例: `main.c:100`

●シンボルによるアドレス指定

アドレスの指定には、シンボルを使用することもできます。ただし、シンボル情報を含むelf形式のオブジェクトファイルをデバッグする場合に限られます。

●ファイル名の入力

カレントディレクトリ以外のファイル名は必ずパスを指定してください。

使用可能な文字は、a～z、A～Z、0～9、/、_のみです。大文字と小文字は区別されません。

また、ドライブ名は/cygdrive/<ドライブ名>/形式で指定し、パスの区切りは¥ではなく/を使用してください。

例: (gdb)
`file /cygdrive/c/EPSON/gnu33/sample/txt/sample.elf`

10.5.3 メニュー / ツールバーによる実行

[デバッグ]ビュー、[ソース]エディターのメニュー、ツールバーにはコマンドの一部が登録されています。メニューの選択、ツールバーボタンのクリックだけで指定のコマンドを実行できます。また、各ビューにもコマンド実行に相当する機能が実装されています。表10.5.3.1に登録されているコマンドの一覧を示します。

表10.5.3.1 メニュー、ツールバー、ビューで指定可能なコマンド

コマンド	ビュー	メニュー / その他	ボタン
continue	[デバッグ]	[実行]-[再開] コンテキストメニュー [再開]	
until	[ソース] [逆アセンブル]	[実行]-[指定行まで実行] コンテキストメニュー [指定行まで実行]	-
step	[デバッグ]	[実行]-[ステップイン] コンテキストメニュー [ステップイン]	
stepi	[デバッグ]	命令ステップ・モード時 [実行]-[ステップイン] 命令ステップ・モード時 コンテキストメニュー [ステップイン]	 選択時 
next	[デバッグ]	[実行]-[ステップ・オーバー] コンテキストメニュー [ステップ・オーバー]	
nexti	[デバッグ]	命令ステップ・モード時 [実行]-[ステップ・オーバー] 命令ステップ・モード時 コンテキストメニュー [ステップ・オーバー]	 選択時 
finish	[デバッグ]	[実行]-[ステップ・リターン] コンテキストメニュー [ステップ・リターン]	
ユーザコマンド*	[デバッグ]	[実行]-[ユーザコマンド] コンテキストメニュー [ユーザコマンド]	
c33 rstc *	[デバッグ]	[実行]-[コールドリセット] コンテキストメニュー [コールドリセット]	
c33 rsth *	[デバッグ]	[実行]-[ホットリセット] コンテキストメニュー [ホットリセット]	
c33 profile	[デバッグ]	コンテキストメニュー [プロファイラ]	
c33 coverage	[デバッグ]	コンテキストメニュー [カバレッジ]	
break	[ソース] [逆アセンブル]	[実行]-[ブレークポイントの切り替え] [実行]-[行ブレークポイントの切り替え] [実行]-[メソッド・ブレークポイントの切り替え] 行ルーラー ダブルクリック 行ルーラー コンテキストメニュー [ブレークポイントの切り替え]	-
tbreak	[ソース] [逆アセンブル]	行ルーラー コンテキストメニュー [テンポラリソフトブレークの切り替え]	-

コマンド	ビュー	メニュー / その他	ボタン
hbreak	[ソース] [逆アセンブル]	行ルーラー コンテキストメニュー [ハードブレークの切り替え]	—
thbreak	[ソース] [逆アセンブル]	行ルーラー コンテキストメニュー [テンポラリハードブレークの切り替え]	—
watch rwatch awatch	[ソース] [ブレークポイント] [メモリー]	シンボル選択時 [実行]-[監視ポイントの切り替え] コンテキストメニューの[監視ポイントの追加]	—
disable	[ソース]	コンテキストメニュー [ブレークポイントを使用不可にする]	—
	[ブレークポイント]	コンテキストメニュー [使用不可]	—
enable	[ソース]	コンテキストメニュー [ブレークポイントを使用可能にする]	—
	[ブレークポイント]	コンテキストメニュー [使用可能にする]	—
delete	[ソース]	行ルーラー ダブルクリック	—
	[ブレークポイント]	コンテキストメニュー [除去]または[すべて除去]	—
x /b, x /h, x /w	[メモリー]	キーボードよりアドレス入力	—
set {char}, set {short}, set {int}	[メモリー]	キーボードよりデータ入力	—
info reg	[レジスター]	レジスタグループ名 (mainなど) を開く	—
set \$Register	[レジスター]	コンテキストメニュー [値の変更]	—
info locals	[変数]	[ウィンドウ]-[ビューの表示]-[変数]	—
print	[ソース]	変数にカーソルをポイント コンテキストメニュー [監視式を追加]	—
	[式]	コンテキストメニュー [監視式を追加]	
	[変数]	コンテキストメニュー [グローバル変数の追加]	
quit	[デバッグ]	[実行]-[終了] コンテキストメニュー [終了]	

- * これらのメニュー / コマンド / ボタンは、それぞれコマンドファイルと関連付けられており、そのコマンドファイルをsourceコマンドにより実行します。コマンドファイルの内容はユーザが自由に編集可能です。ただし、コマンドファイルに自分自身を呼び出すコマンドを記述すると永久ループになりますので、注意してください

例: userdefine.gdbに"source userdefine.gdb"を記述した場合

また、コマンドファイル名やディレクトリを変更することはできません。既定の位置にコマンドファイルが存在しない場合、メニュー / ボタンによる実行時にエラーとなります。



[実行]-[ユーザコマンド]

コマンドファイル¥gnu33¥userdefine.gdbを実行します。

```
出荷時のuserdefine.gdb
#Edit user command
#c33 rsth
```

(何も実行しません)



[実行]-[コールドリセット]

コマンドファイル¥gnu33¥resetcold.gdbを実行します。

```
出荷時のresetcold.gdb
c33 rstc
```

(CPUをコールドリセットします)

**[実行]-[ホットリセット]**

コマンドファイル¥gnu33¥resethot.gdbを実行します。

出荷時のresethot.gdb

c33 rsth

(CPUをホットリセットします)

注: メニュー、ツールバーで実行したコマンドは、[コンソール]ビューのプロンプト部分には表示されません。

10.5.4 コマンドファイルによる実行

一連のデバッグコマンドを記述したコマンドファイルを読み込んで、それらのコマンドを実行させることができます。

●コマンドファイルの作成

コマンドファイルは汎用エディタ等でテキストファイルとして作成してください。

●コマンドファイル例

次の例は、¥GNU33¥sample_ide¥std¥dmt33301¥tst のサンプルに含まれているコマンドファイル例です。コマンドは1行にひとつのみ記述可能です。

例: ファイル名 = tst_gnu33IDE.cmd

file tst.elf	デバッグ情報の読み込み
c33 rpf tst_gnu33IDE.par	メモリマップ情報の設定
target icd6 usb	ターゲットの接続
load tst.elf	プログラムのロード
set {char}0x4812d=0x59	ブートベクタアドレス書き込み保護の解除
set {long}0x48134=0x00600000	ブートベクタアドレスの設定
c33 rsth	ホットリセット

●コマンドファイルの読み込み/実行

コマンドファイルを読み込み、実行させる方法は次の2通りです。

1. 起動時オプションによる実行

デバッグの起動コマンドで `-x` オプション (または `--command` オプション) を指定すると、1つのコマンドファイルをデバッグの起動時に実行させることができます。

例: `c:¥EPSON¥gnu33¥gdb -x startup.cmd`

2. コマンドによる実行

コマンドファイルの実行用に、`source` コマンドが用意されています。

`source` コマンドは指定されたファイルを読み込み、その中のコマンドを記述順に実行します。

例: (gdb)

```
source startup.cmd
```

コマンドファイルに記述されたコマンドは、[コンソール]ビューに表示されます。

●コマンド実行間隔

`--c33_cmw` オプションを指定すると、各コマンド間に指定秒数の待ち時間が挿入されます。指定可能な秒数は1～256で、それ以外の数値を指定した場合は1秒の待ち時間となります。`--c33_cmw` オプションを指定せずにデバッグを起動した場合、各コマンド間に待ち時間は挿入されません。

10.5.5 ログファイル

実行したコマンドと実行結果を、テキスト形式のログファイルとして保存することができます。これによって、後からデバッグの手順と内容を確認することができます。

●コマンド例

```
(gdb)
c33 log test.log   ログイング開始
                  :   ログモード
```

```
(gdb)
c33 log           ログイング終了
```

c33 logコマンドでログイング開始後は、次にc33 logコマンドが実行されるまでログの保存を続けます。

●ログの保存内容

実行したすべてのコマンドとその実行結果が保存されます。これには、メニューやツールバーボタンで実行した[コンソール]ビューに表示されないコマンドも含まれます。

10.6 デバッグ機能

ここでは、**gdb**のデバッグ機能の概要を機能別に説明します。
各デバッグコマンドの詳細については"10.7 コマンドリファレンス"を参照してください。

10.6.1 コネクトモード

gdbは4種類のコネクトモードに対応しており、**target**コマンドで使用するモードを設定します。

●ICD2モード

インサーキットデバッガS5U1C33000H(ICD ver.2)を使用してデバッグを行うモードです。プログラムはターゲットボード上で実行され、トレースデータはS5U1C33000H上のメモリに取得します。

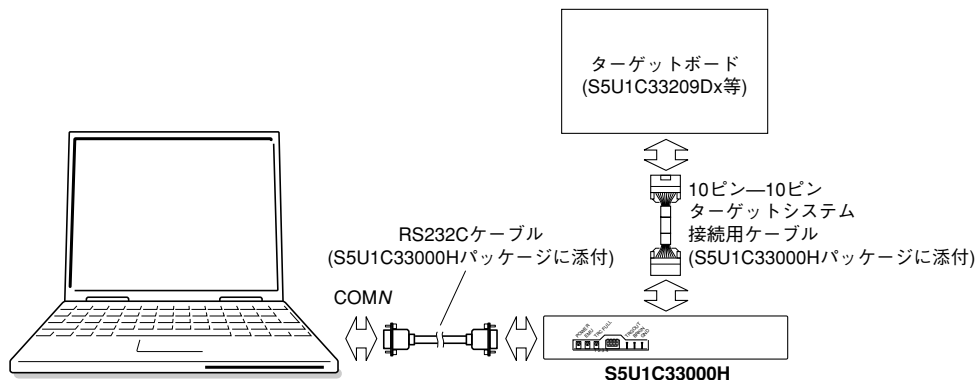


図10.6.1.1 S5U1C33000Hを使用するデバッグシステム例

指定方法

コマンド: (**gdb**)

target icd comN

comN: 使用するシリアルポート(com1～com9)を指定します。

IDEでの指定方法:

[初期起動コマンドの生成]ダイアログボックスの[デバッガ:]コンボボックスから"ICD Ver2"を選択して、起動用コマンドファイルを生成します。また、このダイアログボックスで、COMポートやボーレートの選択も行います。

ICD2モードで起動する場合、S5U1C33000Hおよびターゲットボードが正しく接続され、それらの電源も投入されている必要があります。

また、トレース機能を使用する場合はS5U1C33000HのディップスイッチSW4をOPENとしておきます。

ICD2モードでは、フラッシュライター機能を使用することはできません。

S5U1C33000Hの取り扱い等については、"S5U1C33000H Manual(S1C33 Family In-Circuit Debugger)"を参照してください。

注: ターゲットのCPUコアがC33 PEの場合、以下の2点の制限があります。

- ・アクセス可能なアドレス空間は28ビットまでです。
- ・C33 PEに内蔵されているスペシャルレジスタ(TTBR)はリード/ライトできません。

●ICD3モード

インサーキットデバッガS5U1C33001H1100(ICD ver.3)またはS5U1C33001H1200(ICD ver.4)を使用してデバッグを行うモードです。プログラムはターゲットボード上で実行され、トレースデータはS5U1C33001H1100/1200上のメモリに取得します。

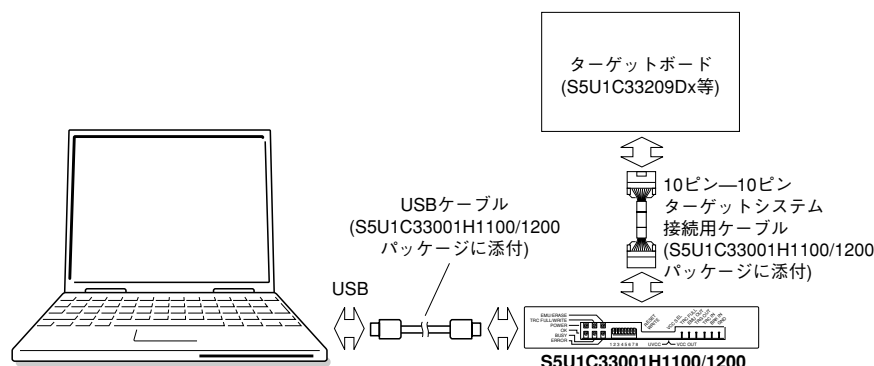


図10.6.1.2 S5U1C33001H1100/1200 (Ver.3/Ver.4)を使用するデバッグシステム例

指定方法

コマンド: (gdb)

target icd usb

IDEでの指定方法:

[初期起動コマンドの生成]ダイアログボックスの[デバッガ:]コンボボックスから"ICD Ver3/Ver4"を選択して、起動用コマンドファイルを生成します。

ICD3モードで起動する場合、S5U1C33001H1100/1200およびターゲットボードが正しく接続され、それらの電源も投入されている必要があります。また、トレース機能を使用する場合はS5U1C33001H1100/1200のディップスイッチSW4をOPENとしておきます。S5U1C33001H1100/1200の取り扱い等については、"S5U1C33001H Manual (S1C33 Family In-Circuit Debugger)"を参照してください。

●ICD6モード

インサーキットデバッガS5U1C33001H1400(ICD ver.6)を使用してデバッグを行うモードです。プログラムはターゲットボード上で実行され、トレースデータはS5U1C33001H1400上のメモリに取得します。

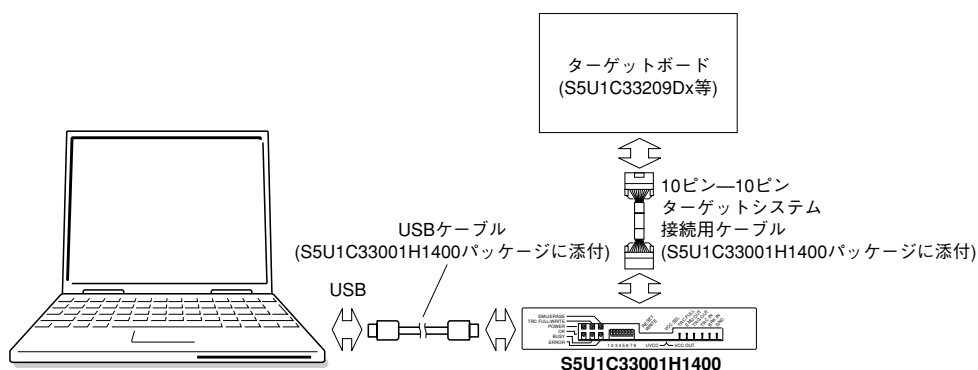


図10.6.1.3 S5U1C33001H1400 (Ver.6)を使用するデバッグシステム例

指定方法

コマンド: (gdb)

target icd6 usb

IDEでの指定方法:

[初期起動コマンドの生成]ダイアログボックスの[デバッガ:]コンボボックスから"ICD Ver6"を選択して、起動用コマンドファイルを生成します。

ICD6モードで起動する場合、S5U1C33001H1400およびターゲットボードが正しく接続され、それらの電源も投入されている必要があります。S5U1C33001H1400の取り扱い等については、"S5U1C33001H1400 Manual (S1C33 Family In-Circuit Debugger)"を参照してください。

●デバッグモニタ (MON) モード

デバッグモニタモードは、S5U1C330M1D1ボードを介して接続されたターゲットボード上でプログラムのデバッグを行うモードです。ターゲットボードにはデバッグモニタS5U1C330M2Sが実装されている必要があります。

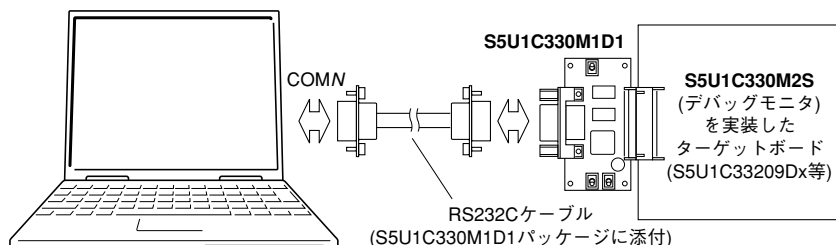


図10.6.1.4 S5U1C330M1D1、S5U1C330M2Sを使用するデバッグシステム

指定方法

コマンド: (gdb)

```
target icd comN
```

comN: 使用するシリアルポート (com1～com9) を指定します。

IDEでの指定方法:

[初期起動コマンドの生成]ダイアログボックスの[デバッグ:]コンボボックスから"MON"を選択して、起動用コマンドファイルを生成します。また、このダイアログボックスで、COMポートやボーレートの選択も行います。

デバッグモニタモードで起動する場合、S5U1C330M1D1およびターゲットボードが正しく接続され、それらの電源も投入されている必要があります。また、ターゲットボードのデバッグモニタを起動しておきます。デバッグモニタモードでは、トレース機能、フラッシュライタ機能、[中断]ボタンによる強制ブレークを使用することはできません。デバッグモニタの詳細については、"S1C33 Family Debug Monitor Operation Manual"を参照してください。

●シミュレータ (SIM) モード

シミュレータモードは、パソコンのメモリ上でターゲットプログラムの実行をシミュレートするモードで、他のツールを必要としません。ただし、S5U1C33000HやS5U1C33001Hに依存した機能は使用できません。

指定方法

コマンド: (gdb)

```
target sim
```

IDEでの指定方法:

[初期起動コマンドの生成]ダイアログボックスの[デバッグ:]コンボボックスから"Simulator"を選択して、起動用コマンドファイルを生成します。

シミュレータモードのトレース方法はICD2/ICD3/ICD6モードとは異なります。また、フラッシュライタ機能も使用できません。

シミュレータモードでは、各領域が次のように初期化されます。

- RAM: 0xaa
- ROM: 0xff
- I/O: 0x00

●ICD、デバッグモニタモードにおける注意事項

S5U1C33000H、S5U1C33001Hでプログラムの実行をブレイクするとCPUの動作クロックが高速クロック(OSC3)に切り換わり、ブレイク中はDRAMリフレッシュを除き、S1C33チップ内の全周辺機能に対するクロック供給が停止します。

S5U1C330M2Sでのプログラム実行時およびブレイク時は、瞬間的に上記と同じ状態になり、すぐに元の状態に戻ります。

このため、OSC3クロックを使用しないシステムはデバッグできません。

また、OSC3発振回路を停止して低速クロック(OSC1、32kHz)で動作している場合は、プログラムの実行をブレイクした瞬間にOSC3が発振を開始しますが、発振が不安定なためCPUが誤動作します。SLEEP時を含み、OSC3発振が停止状態でのブレイクは避けてください。

S5U1C33001Hを使用してデバッグを行っている場合は(ICD3/ICD6モード時)、S5U1C33001Hの電源を切る前に必ずデバッグを終了させてください。デバッグを実行中にS5U1C33001Hの電源を切ると再接続ができなくなることがあります。この場合、パーソナルコンピュータの再起動が必要です。

●コネクトモードと対応ターゲットCPU

各コネクトモードが対応しているターゲットCPUを表10.6.1.1に示します。

表10.6.1.1 コネクトモードと対応ターゲットCPU

コネクトモード	C33 STD	C33 PE	S1C33401
ICD2	○	×	×
ICD3	○	○	○
ICD6	○	○	○
デバッグモニタ	○	×	×
シミュレータ	○	○	○

10.6.2 ファイルの読み込み

●ファイルの種類

gdbはelf形式のオブジェクトファイルを読み込んでデバッグすることができます。

●ファイル読み込み手順

ファイルの読み込みには、次の2つのコマンドを使用します。

fileコマンド: デバッグ情報を読み込みます。

loadコマンド: オブジェクトコードをターゲットにロードします。

これとは別に、ターゲットのメモリマップ情報を設定するためにパラメータファイルを読み込む**c33 rpf**コマンドが用意されています。

fileコマンドは、**target**コマンドや**load**コマンドより先に実行する必要があります。また、**c33 rpf**コマンドは**file**コマンドの前に実行します。

ファイルを読み込んでデバッグを行うまでの基本的な実行順序は次のとおりです。

```
(gdb)
c33 rpf sample.par      (マップ情報設定)
(gdb)
file sample.elf         (デバッグ情報の読み込み)
(gdb)
target icd usb          (ターゲットの接続)
(gdb)
load                    (プログラムのロード)
(gdb)
c33 rsth                (ホトリセット)
```

ターゲットのROMに書き込まれているプログラムをデバッグする場合、**load**コマンドの実行は不要です。この場合も、**file**コマンドでデバッグ情報を読み込むことで、ソースレベルデバッグが行えます。

●注意点

loadコマンドはオブジェクトファイル内のコードおよびデータが存在する領域のみを読み込みます。それ以外の領域は**load**コマンド実行前の状態のまま残りますので注意してください。

gdbはソース表示を行うため、デバッグ情報に従ってソースファイルを読み込みます。このため、ソースファイルの内容、保存場所(ディレクトリ)とも**elf**オブジェクトファイルを生成したときと同じ状態になっている必要があります。

10.6.3 メモリ、変数およびレジスタの操作

gdbはメモリおよびレジスタに対する操作を行う機能を持っています。ハーフワードおよびワードデータのアクセスと表示はリトルエンディアン形式で行われます。ただし、シミュレータモードの場合は、パラメータファイルで特定の外部メモリ領域をビッグエンディアン形式に設定できます。

●メモリ領域の操作

メモリ領域に対しては以下の操作が行えます。アドレスの指定には変数名などのシンボルも使用可能です。また、どの機能も、バイト、ハーフワード、またはワード単位の処理に対応しています。

メモリダンプ(x /b、x /h、x /wコマンド)

指定のアドレスから指定データ数分のメモリの内容を16進数でダンプ表示します。

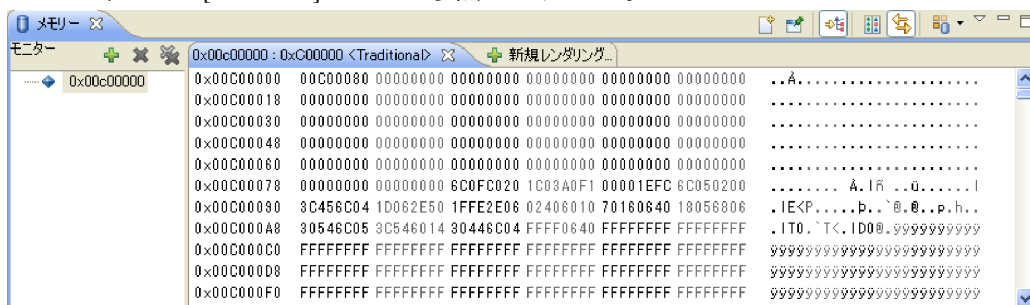
例: `_START_text`から16ハーフワードのメモリダンプ

(gdb)

`x /16h _START_text`

```
0xc00000 <_START_text>: 0x0004 0x00c0 0xc020 0x6c0f 0xa0f1 0xc000 0xc000 0x6c0f
0xc00010 <boot+12>:      0xc000 0xc000 0x1c04 0xdff8 0xdfff 0x1ef5 0x0200 0x6c04
```

[メモリ]ビューにメモリデータを表示させることもできます。[メモリ]ビューの表示と操作については、「10.4.9 [メモリ]ビュー」を参照してください。



データの入力(set {char}、set {short}、set {int}コマンド)

指定のアドレスに指定のデータを書き込みます。

データの入力/変更は[メモリ]ビュー上でも行えます。

例: `int i`を0x55555555に設定

(gdb)

`set {int}&i=0x55555555`

指定領域の書き換え(c33 fb、c33 fh、c33 fwコマンド)

指定した領域を指定したデータですべて書き換えます。

例: アドレス0x0～0xfに4ワードの0x1を書き込み

(gdb)

`c33 fw 0x0 0xf 0x1`

Start address = 0x0, End address = 0xc, Fill data = 0x1 done

指定領域のコピー(c33 mvb、c33 mvh、c33 mvwコマンド)

指定した領域の内容を、別の領域にコピーします。

例: アドレス0x0～0x7の8バイトを、アドレス0x8から始まる8バイトにコピー

(gdb)

`c33 mvb 0x0 0x7 0x8`

Start address = 0x0, End address = 0x7, Destination address = 0x8 done

メモリ内容の保存(c33 dfコマンド)

指定範囲のメモリの内容をバイナリ形式、テキスト形式、またはモトローラS3形式でファイルに出力します。

例: アドレス0x80000～0x80103の内容を、dump.motという名称のモトローラS3ファイルとして保存

(gdb)

`c33 df 0x80000 0x80103 3 dump.mot`

Start address = 0x80000, End address = 0x8011f, File type = Motorola-S3

Processing 00080000-0008011F address.

ターゲットメモリリードモードの指定(c33 readmdコマンド) ICD3/ICD6モードのみ

xコマンドやc33 dfコマンドによる通常のメモリリードでは、アクセスするデータサイズにかかわらずすべてバイト単位でデータを読み出します。この読み出し方法で不都合がある場合は、**c33 readmd**コマンドによりメモリデータが正しい境界アドレスから指定の単位で読み出されるように変更できます。ただし、メモリダンプにかかる時間が増加します。

例: メモリデータが正しい境界アドレスから指定サイズ単位で読み出されるように変更

(gdb)

```
c33 readmd 1
```

注: ・ 内蔵I/O領域の中でレジスタ等が割り付けられていないアドレスを読み出すと、チップの仕様上CPUが最後に読み出したデータが表示されます。

- ・ S5U1C33000H/S5U1C33001Hは、メモリからの読み出しを1回につき8バイト単位で行います。このため、コマンドで指定したアドレスより、最大7バイト先まで読み出す可能性があります。I/Oメモリをアクセスする際は、読み出しにより状態が変更されるレジスタに注意してください。
- ・ S5U1C33000H/S5U1C33001HあるいはS5U1C330M2SによりS5U1C33XXE上の内蔵ROMエミュレーションメモリにデータを書き込む場合はバイト書き込みのコマンド(set {char}、c33 fb、c33 mvb)は使用しないでください。必ず、ハーフワード以上のコマンド(set {short}、set {int}、c33 fh、c33 fw、c33 mvh、c33 mvw)を使用してください。ファイルロードコマンド(load)はハーフワード書き込みを行っています。

●変数一覧

変数の一覧を表示できます。

グローバル変数の表示(info var、printコマンド)

グローバル変数、スタティック変数、およびセクションシンボル等の一覧が**info var**コマンドで表示できます。また、**x**コマンドや**print**コマンドで変数の内容を表示することもできます。

例: グローバルシンボルの一覧表示

(gdb)

```
info var
```

```
All defined variables:
```

```
File ./main.c:
```

```
int i;
```

```
Non-debugging symbols:
```

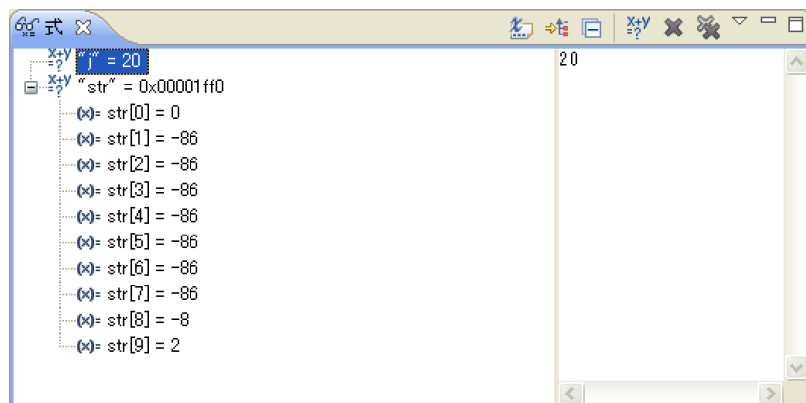
```
0x00000000 __START_bss
```

```
0x00000004 __END_bss
```

```
0x00000004 __END_data
```

```
0x00000004 __START_data
```

また、[式]ビューにグローバル変数を登録しておくことで、その値をモニタ可能です。[式]ビューの表示と操作については、"10.4.7 [式]ビュー"を参照してください。



ローカル変数の表示(info localsコマンド)

現在のPCアドレスを含む関数内で定義されているローカル変数の一覧が、その値とともに**info locals**コマンドで表示できます。

例: 現在の関数内に定義されているローカル変数を表示

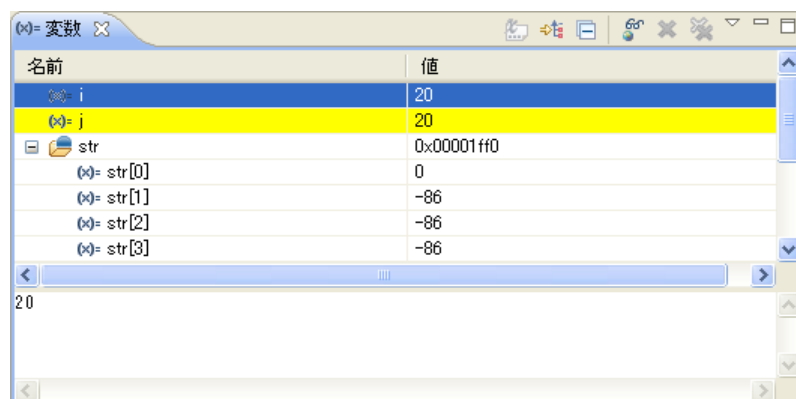
(gdb)

info locals

i = 0

j = 2

また、[変数]ビューを開いておくと、現在の関数内で定義されているすべてのローカル変数の値をモニタ可能です。[変数]ビューの表示と操作については、"10.4.6 [変数]ビュー"を参照してください。



●レジスタの操作

レジスタに対しては以下の操作が行えます。

レジスタの表示(info regコマンド)

CPUレジスタの内容をすべて、または指定したレジスタの内容を[コンソール]ビューに表示させることができます。

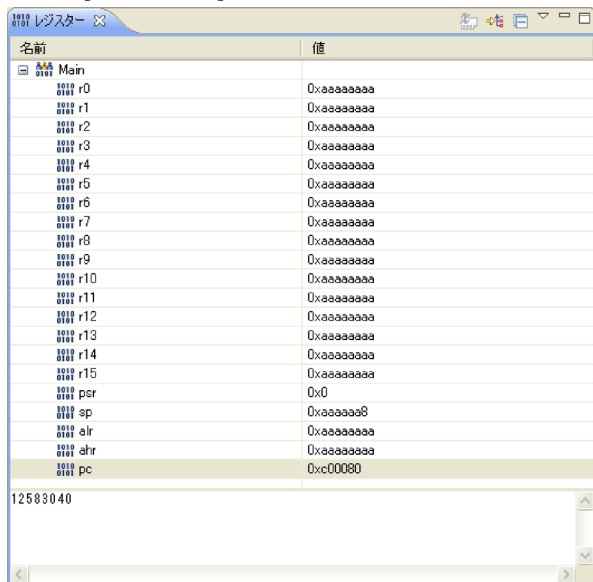
例: すべてのレジスタ値の表示(C33 ADVコアの場合)

(gdb)

info reg

r0	0xd20	3360
r1	0xaaaaaaaa	-1431655766
r2	0xaaaaaaaa	-1431655766
r3	0xaaaaaaaa	-1431655766
r4	0x690	1680
r5	0xaaaaaaaa	-1431655766
r6	0x0	0
r7	0xaaaaaaaa	-1431655766
r8	0xaaaaaaaa	-1431655766
r9	0xaaaaaaaa	-1431655766
r10	0xaaaaaaaa	-1431655766
r11	0xaaaaaaaa	-1431655766
r12	0xaaaaaaaa	-1431655766
r13	0xaaaaaaaa	-1431655766
r14	0xaaaaaaaa	-1431655766
r15	0x0	0
psr	0x2	2
sp	0x7f8	2040
alr	0x0	0
ahr	0x0	0
lco	0x0	0
lsa	0x0	0
lea	0x0	0
sor	0x0	0
ttbr	0x20000000	536870912
dp	0x0	0
idir	0x4000000	67108864
dbbr	0x60000	393216
usp	0x0	0
ssp	0x0	0
pc	0xc00030	12582960

[レジスター]ビューもレジスタ値を表示します。[レジスター]ビューの表示と操作については、"10.4.8 [レジスター]ビュー"を参照してください。



レジスタ値の変更 (set \$コマンド)

CPUレジスタの内容を任意の値に設定できます。

例: r1レジスタを0x10000に設定

(gdb)

```
set $r1=0x10000
```

レジスタ値は[レジスター]ビュー上で直接書き換えることもできます("10.4.8 [レジスター]ビュー"参照)。

10.6.4 プログラムの実行

デバッガにはターゲットプログラムを連続実行およびシングルステップ実行させる機能があります。

●連続実行

連続実行コマンド(continue、untilコマンド)

連続実行コマンドは、ロードされているプログラムを現在のPCアドレスから連続実行します。

continueコマンド: プログラムを連続実行する際、現在のブレークポイントを指定回数無効にすることができます。

例1: 現在のPCから連続実行

```
(gdb)
cont
Continuing.
```

例2: 現在のブレークポイントを4回スキップする指定で、現在のPCから連続実行

```
(gdb)
continue 5
Will ignore next 4 crossings of breakpoint 1. Continuing.
```

untilコマンド: 1回のみ有効なテンポラリPCブレークポイントを指定し、プログラムの実行をその位置で停止させることができます。

例: 現在のPCからmain.cの10行目まで連続実行。main.cの10行目を実行直前にブレーク

```
(gdb)
until main.c:10
main () at ./main.c:10
```

上記のコマンドを[デバッグ]ビュー上でも実行可能です。

continueコマンドの実行: • [実行]メニューから[再開]を選択

- [再開]ボタンをクリック



- * ブレーク無効回数は指定できません。

untilコマンドの実行:

- コンテキストメニューの[指定行まで実行]を選択
- * コンテキストメニューは、テンポラリPCブレークポイントに設定したソース行の先頭を右クリックして表示させます。

[ソース]エディター上の操作については、"10.4.3 [ソース]エディター"を参照してください。

連続実行の停止

実行中のプログラムは次のいずれかの要因によってブレークするまで停止しません。

- ブレーク設定コマンドで設定したブレーク条件が成立(untilコマンドのテンポラリブレークも含みます。)
- 強制ブレーク([中断]ボタンのクリック。デバッグモニタモードを除きます。)
- その他ブレーク要因の発生



[中断]ボタン

- * プログラムが停止しない場合は、このボタンで強制ブレークさせることができます。

プログラムが停止すると、[コンソール]ビューには、ブレーク要因と停止位置が表示されます。

また、[ソース]エディターや[レジスター]ビューの表示内容が更新されます。

● シングルステップ実行

シングルステップ実行の種類

3種類のシングルステップ実行機能があります。

全コードをシングルステップ実行(**step**、**stepi**コマンド)

現在のPCアドレスから1ステップまたは指定ステップ数分実行します。関数コールやサブルーチンコールを実行した場合、呼び出した関数/サブルーチン内もシングルステップ実行します。

stepコマンド: ソース行単位のシングルステップ実行

例: 現在のPCが示すソース行をシングルステップ実行

```
(gdb)
```

```
step
```

stepiコマンド: アセンブラ命令単位のシングルステップ実行

例: 現在のPCが示すアドレスから10命令をシングルステップ実行

```
(gdb)
```

```
stepi 10
```

```
main () at ./main.c:13
```

関数/サブルーチン以外をシングルステップ実行(**next**、**nexti**コマンド)

現在のPCアドレスから1ステップまたは指定ステップ数分実行します。関数コールやサブルーチンコールは、呼び出す関数/サブルーチン内のすべての行を含め1ステップとして実行します。それ以外は、**step**、**stepi**コマンドと同機能です。

nextコマンド: ソース行単位のシングルステップ実行

例: 現在のPCが示すソース行をシングルステップ実行。関数コールの場合は関数内もすべて実行

```
(gdb)
```

```
next
```

nextiコマンド: アセンブラ命令単位のシングルステップ実行

例: 現在のPCが示すアドレスから10命令をシングルステップ実行。サブルーチンコールはサブルーチン内を含め1ステップ分として実行

```
(gdb)
```

```
nexti 10
```

```
main () at ./main.c:13
```

関数/サブルーチンの終了(**finish**コマンド)

関数/サブルーチン内でプログラムが停止している場合に、呼び出し元にリターンするまでステップ実行します。

例: 現在の関数を終了

```
(gdb)
```

```
finish
```

```
Run till exit from #0  0x00c00040 in sub (k=1) at ./main.c:22
```

```
main () at ./main.c:14
```

```
Value returned is $1 = 480
```

コマンド入力による実行では、実行するステップ数を最大0x7fffffffステップまで指定することができます。

プログラムが停止すると、[コンソール]ビューには、停止位置のソースが表示されます。

また、[ソース]エディターや[レジスター]ビューの表示内容が更新されます。

上記のコマンドを[デバッグ]ビュー上のメニュー、ツールバーボタンでも実行可能です。ただし、実行ステップ数は指定できません(1ステップのみ実行)。

stepコマンドの実行:

- [実行]メニューの[ステップイン]を選択
- コンテキストメニューの[ステップイン]を選択
- [ステップイン]ボタンをクリック

 [ステップイン]ボタン

stepiコマンドの実行:

- [命令ステップ・モード]ボタンを選択状態で[実行]メニューの[ステップイン]を選択

- [命令ステップ・モード]ボタンを選択状態でコンテキストメニューの[ステップイン]を選択
- [命令ステップ・モード]ボタンを選択状態で[ステップイン]ボタンをクリック

 選択状態で  [ステップイン]ボタン

- nextコマンドの実行:
- [実行]メニューの[ステップ・オーバー]を選択
 - コンテキストメニューの[ステップ・オーバー]を選択
 - [ステップ・オーバー]ボタンをクリック

 [ステップ・オーバー]ボタン

- nextiコマンドの実行:
- [命令ステップ・モード]ボタンを選択状態で[実行]メニューの[ステップ・オーバー]を選択
 - [命令ステップ・モード]ボタンを選択状態でコンテキストメニューの[ステップ・オーバー]を選択
 - [命令ステップ・モード]ボタンを選択状態で[ステップ・オーバー]ボタンをクリック

 選択状態で  [ステップイン]ボタン

- finishコマンドの実行:
- [実行]メニューから[ステップ・リターン]を選択
 - コンテキストメニューの[ステップ・リターン]を選択
 - [ステップ・リターン]ボタンをクリック

 [ステップ・リターン]ボタン

シングルステップ実行中のブレーク

ステップ数を指定して実行する場合、その実行終了前でも以下の要因によって実行は停止します。

- 強制ブレーク([中断]ボタンのクリック。デバッグモニタモードを除きます。)
- ユーザ設定ブレーク以外のブレーク要因の発生

PCブレークポイント、データブレークポイントでは停止しません。



中断ボタン

* プログラムが停止しない場合は、このボタンで強制ブレークさせることができます。

プログラムが停止すると、[コンソール]ビューには、ブレーク要因と停止位置が表示されます。

●ユーザ関数のコール(callmd、callコマンド)

callコマンドを使用してユーザ関数をコールすることができます。たとえば、S1C33 FamilyリアルタイムOS内の関数をコールして、各種デバッグ情報を取得することができます。その詳細については、"S1C33 Family Realtime OSオペレーションマニュアル"を参照してください。

例: リアルタイムOSの管理データ一覧出力

```
(gdb)
call mng
```

callmdコマンドはcallコマンドの実行結果を画面またはファイルのどちらに出力するか設定します。

●HALT、SLEEP状態と割り込み

ICD2、ICD3、ICD6、デバッグモニタモードでは、連続実行またはシングルステップ実行にかかわらずhalt命令およびslp命令は実行され、それによってスタンバイモードになります。スタンバイモードは、外部割り込みを発生させることによって解除されます。また、[中断]ボタンで解除することもできます。

シミュレータモードはスタンバイモードと割り込みをサポートしていません。halt命令およびslp命令が実行されると、プログラムの実行はブレークします。

●実行サイクル/実行時間の測定

ICD2、ICD3、ICD6、シミュレータモードにはプログラムの実行サイクル/時間を測定する機能があります。デバッグモニタモードには、この測定機能はありません。

実行カウンタと測定モード

S5U1C33000Hは29ビット、S5U1C33001Hは31ビットの実行カウンタを1個内蔵しており、カウンタの測定単位を**c33 clockmd**コマンドで実行時間(μs単位または秒単位)またはサイクル数に設定可能です。シミュレータモードの実行カウンタは、サイクル数のカウントのみを行います。

表10.6.5.1 実行カウンタの測定単位と精度

測定単位	精度	
	ICD2/ICD3/ICD6モード	シミュレータモード
実行時間計測(秒単位)	±2μs *1	—
実行時間計測(μs単位)	±100ns *1	—
サイクル数計測(1サイクル単位)	±16サイクル *1	±0サイクル

*1 continue〜ブレイクのオーバーヘッドによりさらに2命令分のプリフェッチサイクルも加算されます。

S5U1C33000H、S5U1C33001Hの実行カウンタで測定可能な最大値は次のとおりです。

秒単位: S5U1C33000H: 約9分、S5U1C33001H: 約36分

μs単位: S5U1C33000H: 約27秒、S5U1C33001H: 約1.8分

サイクル数: S5U1C33000H: 2,147,483,644 サイクル、S5U1C33001H: 8,589,934,588サイクル

測定結果の表示

測定結果は**c33 clock**コマンドによって[コンソール]ビューに表示することができます。

例:

```
(gdb)
c33 clock
      330 cycle
```

カウンタの最大値を超えた場合は"clock timer overflow"を表示します。

積算モードとリセットモード

デバッグの初期設定では、実行サイクルカウンタは積算モードに設定されます。このモードでは、カウンタがリセットされるまで測定値を積算します。

リセットモードは**c33 clockmd**コマンドで設定することができ、プログラムの実行ごとにカウンタがリセットされます。

リセットモードに設定した場合、連続実行では実行コマンド(**continue**、**until**)の入力によるプログラムの実行開始時にリセットされ、その実行終了(ブレイク)までを測定します。

シングルステップ実行では、ステップ実行コマンド(**step**、**next**等)の入力による実行開始時にリセットされ、指定ステップ数の実行終了までを測定します。1ステップのみの指定またはツールバーボタン/メニューコマンドによる実行では、1ステップごとにリセットされます。

実行カウンタのリセット

実行カウンタは次の場合にリセットされます。

- 実行カウンタのモードを**c33 clockmd**コマンドにより切り換えた場合(積算モード↔リセットモード)
- リセットモードでプログラムの実行を開始した場合
- CPUをコールドリセットまたはホットリセットした場合

積算モードでも**continue**、**until**コマンド以外(**step**、**next**等)ではプログラム実行開始時カウンタはリセットされてしまいますので、時間計測する場合は**continue**、**until**コマンドを使用してください。

●CPUのリセット

CPUは**c33 rstc**コマンドによりコールドリセット、**c33 rsth**コマンドによりホットリセットされます。これらのコマンドは[デバッグ]ビュー上のメニュー、ツールバーボタンでも実行可能です。

c33 rstcコマンドの実行: • [実行]メニューから[コールドリセット]を選択
(¥gnu33¥resetcold.gdb) • [コールドリセット]ボタンをクリック



[コールドリセット]ボタン

c33 rsthコマンドの実行: • [実行]メニューから[ホットリセット]を選択
(¥gnu33¥resethot.gdb) • [ホットリセット]ボタンをクリック



[ホットリセット]ボタン

注: これらのメニューコマンド/ボタンは、それぞれ既定のコマンドファイルを実行します。出荷時の各コマンドファイルには上記のリセットコマンドのみが記述されています(ユーザによる編集が可能)。

CPUのリセットによる初期設定内容は以下のとおりです。

(1) CPUの内部レジスタ

```

R0～R15: 0xaaaaaaaa
PC:      ブートアドレス (0xc00000の内容が示すアドレス)
SP:      0x0aaaaaaaa8 (C33 STD)
         0x00000000 (C33 ADV, C33 PE)
PSR:     0x00000000
AHR, ALR: 0xaaaaaaaa
LCO:     0x00000000 (C33 ADV)
LSA:     0x00000000 (C33 ADV)
LEA:     0x00000000 (C33 ADV)
SOR:     0x00000000 (C33 ADV)
TTBR:    0x00c00000 (C33 PE)
         0x20000000 (C33 ADV)
DP:      0x00000000 (C33 ADV)
IDIR:    0x04000000 (C33 ADV)
         0x06000000 (C33 PE)
DBBR:    0x00060000 (C33 ADV, C33 PE)
USP:     0x00000000 (C33 ADV)
SSP:     0x00000000 (C33 ADV)

```

特に指定のないレジスタはすべてのコアに共通です。ホットリセット時のPC値(ブートアドレス)は**TTBR**レジスタにより指定されます。

(2) 実行カウンタを0に設定

(3) [ソース]エディター、[レジスター]ビューを再表示

PCがブートアドレスに設定されるため、[ソース]エディターはプログラムをそのアドレスから再表示します。[レジスター]ビューを上記の設定で再表示します。

メモリの内容は変更されません。

注: コネクトモードにより `c33 rstc` コマンドの動作が異なります。

ICD2/ICD3/ICD6モード 上記の処理を行い、S1C33チップもリセットします。ターゲットボードはリセットされません。また、`c33 rstc` コマンド実行時にターゲットがフリーランしている場合は、強制ブレークをかけてからリセットを行います。S5U1C33000H、S5U1C33001Hに接続されたターゲットをリセットするとターゲットシステムはフリーラン状態となりますが、`c33 rstc` コマンドでこれを停止させることができます。

デバッグモニタモード `c33 rstc` コマンドは `c33 rsth` コマンドと同じ動作をします。S1C33チップのリセットやTTBRレジスタの初期化は行いません。

シミュレータモード パラメータファイルのMCU/MPU指定によりブートアドレスが決定します。シミュレータモードかつC33 ADVモード時で、パラメータファイル内にアドレス0x20000000から1バイト以上のマップが設定されていない場合、PCの初期設定内容は0xc00000の内容が示すアドレスになります。

10.6.5 ブレーク機能

ターゲットプログラムは次の要因により実行を中断します。

- ブレーク設定コマンドで設定したブレーク条件が成立
- 強制ブレーク ([中断]ボタンのクリック。デバッグモニタモードを除きます。)
- 不正なメモリアクセス等の発生

●コマンドによるブレーク機能

gdbはコマンドによってブレーク条件が設定できる6種類のブレーク機能を持っています。

1. ソフトウェアPCブレーク
2. ハードウェアPCブレーク
3. データブレーク
4. オンチップエリア(CE)ブレーク(C33 ADV)
5. オンチップバスブレーク(C33 ADV)
6. 時間経過後ブレーク(ICD6)

それぞれ、条件が成立すると実行中のプログラムは停止します。

ソフトウェアPCブレーク(break、tbreakコマンド)

実行PCアドレスがコマンドで設定したアドレスに一致するとブレークする機能です。プログラムは、そのアドレスの命令を実行する前にブレークします。最大200カ所のアドレスをブレークポイントとして設定できます。

ソフトウェアPCブレークにも、通常のソフトウェアPCブレークとテンポラリソフトウェアPCブレークの2種類があります。ブレーク機能はどちらも同じですが、通常のソフトウェアPCブレークポイントはブレークヒットしても、コマンド等で削除するまで何度も使用可能です。一方、テンポラリソフトウェアPCブレークは、一度のブレークヒットで削除されます。最大200カ所の設定可能数は両方を併せた値です。

breakコマンド: 通常のソフトウェアPCブレークポイントを設定します。

例: アドレス0xc0001cをソフトウェアPCブレークポイントに設定

```
(gdb)
```

```
break *0xc0001c
```

```
Breakpoint 1 at 0xc0001c: file ./main.c, line 7.
```

tbreakコマンド: テンポラリソフトウェアPCブレークポイントを設定します。

例: アドレス0xc0001eをテンポラリソフトウェアPCブレークポイントに設定

```
(gdb)
```

```
tbreak *0xc0001e
```

```
Breakpoint 2 at 0xc0001e: file ./main.c, line 10.
```

ソフトウェアPCブレークが発生したときは、次のようなメッセージを表示してコマンド入力待ちとなります。

```
(gdb)
```

```
continue
```

```
Continuing.
```

```
Breakpoint 1, main () at ./main.c:7
```

ブレークポイントは直接アドレスを指定する以外に、ソース行番号、関数名/ラベル名でも指定できます。ソース行番号を指定した場合、その行が展開されたアセンブラ命令の中で最初に実行される命令のアドレスがブレークポイントとなります。初期化を伴わない変数宣言など、アセンブラ命令に展開されない行を指定した場合は、それ以降最初に実行される命令を持つ行をブレークポイントにします。

例: main.cの7行目をソフトウェアPCブレークポイントに設定

```
(gdb)
```

```
break main.c:7
```

```
Breakpoint 1 at 0xc0001c: file ./main.c, line 7.
```

関数名を指定した場合、関数内で最初に実行される命令を持つソース行がブレークポイントに設定されます。変数宣言のみの行には設定されません。

例: 関数mainにソフトウェアPCブレークポイントを設定

```
(gdb)
```

```
break main
```

```
Breakpoint 1 at 0xc0001c: file ./main.c, line 7.
```

コンパイル時に、関数の先頭にはレジスタ退避用のpush命令が付加されますが、ソース行に対応していないため、この命令のアドレスはブレークポイントにはなりません。アドレス指定によりブレークポイントに設定することは可能です。

[ソース]エディター上でもソフトウェアPCブレークポイントを設定することができます。

詳細については、"10.4.5 [ブレークポイント]ビュー"を参照してください。

注: ・ ソフトウェアPCブレークは、brk命令の埋め込みにより実現しています。そのため、命令の埋め込みができないターゲットボード上のROMに対しては使用できません。その際は、ハードウェアPCブレークを使用してください。

- ・ ソフトウェアPCブレークを、extによる拡張命令または遅延分岐命令のアドレスに設定する場合、先頭アドレス以外には設定できません。

ext xxxx	... ○ 設定可	jr*.d xxxx	... ○ 設定可
ext xxxx	... × 設定不可	遅延命令	... × 設定不可
拡張される命令	... × 設定不可		

- ・ デバッグによるソフトウェアPCブレークを設定可能なアドレスかどうかのチェックは、c33 rpf コマンドでパラメータファイルを読み込んで設定したメモリマップ情報に基づいて行われます。c33 rpfコマンドを実行していない場合、ターゲットシステムに合ったエラー処理は行われません。

ハードウェアPCブレーク(hbreak、thbreakコマンド)

ハードウェアPCブレークはC33コアが持つデバッグモードを使用したブレーク機能です。シミュレータモードでもシミュレーション可能です。実行PCアドレスがコマンドで設定したアドレスに一致すると、そのアドレスの命令を実行する前にブレークします。2カ所までのアドレスをハードウェアPCブレークポイントとして設定できます。

ハードウェアPCブレークにも、ソフトウェアPCブレークポイントと同様に通常のハードウェアPCブレークとテンポラリハードウェアPCブレークの2種類があります。

hbreakコマンド: 通常のハードウェアPCブレークポイントを設定します。

例: main.cの7行目をハードウェアPCブレークポイントに設定

```
(gdb)
```

```
hbreak main.c:7
```

```
Hardware assisted breakpoint 1 at 0xc0001c: file ./main.c, line 7.
```

thbreakコマンド: テンポラリハードウェアPCブレークポイントを設定します。

例: アドレス0xc0001eをテンポラリハードウェアPCブレークポイントに設定

```
(gdb)
```

```
thbreak *0xc0001e
```

```
Hardware assisted breakpoint 2 at 0xc0001e: file ./main.c, line 10.
```

ハードウェアPCブレークが発生したときは、次のようなメッセージを表示してコマンド入力待ちとなります。

```
(gdb)
```

```
continue
```

```
Continuing.
```

```
Breakpoint 1, main () at ./main.c:7
```

ブレークポイントの指定はソフトウェアPCブレークポイントと同様に、アドレス、ソース行番号、関数名/ラベル名で行えます。

[ソース]エディター上でもハードウェアPCブレークポイントを設定することができます。

詳細については、"10.4.5 [ブレークポイント]ビュー"を参照してください。

注: • ICD2、ICD3、ICD6モードで範囲トレースを設定中は、ハードウェアPCブレーク機能は無効となります。

- ハードウェアPCブレークを、extによる拡張命令または遅延分岐命令のアドレスに設定する場合、先頭アドレス以外には設定できません。

ext XXXX	... ○ 設定可	jr*.d XXXX	... ○ 設定可
ext XXXX	... × 設定不可	遅延命令	... × 設定不可
拡張される命令	... × 設定不可		

データブレーク機能(watch、rwatch、awatchコマンド)

データブレークは、指定したメモリアドレスをアクセスした場合にブレークを発生させる機能です。

watchコマンド: データを書き込んだ場合にブレークするデータブレークポイントを設定します。

例: 変数iのアドレスにデータライトブレークポイントを設定

```
(gdb)
watch i
Hardware watchpoint 1: i
```

rwatchコマンド: データを読み出した場合にブレークするデータブレークポイントを設定します。

例: 変数iのアドレスにデータリードブレークポイントを設定

```
(gdb)
rwatch i
Hardware read watchpoint 2: i
```

awatchコマンド: データの書き込みと読み出しのどちらでもブレークするデータブレークポイントを設定します。

例: アドレス0x0にデータリード/ライトブレークポイントを設定

```
(gdb)
awatch *0x0
Hardware access (read/write) watchpoint 3: *0
```

データブレークは指定のアドレスに対する指定のメモリアクセス終了後に発生します。

このブレークが発生したときは、次のようなメッセージを表示してコマンド入力待ちとなります。

例1: データライトブレークポイントでのブレーク

```
(gdb)
continue
Continuing.
Hardware watchpoint 1: i

Old value = 0
New value = 1
sub (k=1) at ./main.c:24
```

例2: データリードブレークポイントでのブレーク

```
(gdb)
continue
Continuing.
Hardware read watchpoint 2: i

Value = 2
0x00c00042 in sub (k=1) at ./main.c:22
```

例3: データリード/ライトブレークポイントでのブレーク

```
(gdb)
continue
Continuing.
Hardware access (read/write) watchpoint 3: *0

Value = 2
sub (k=1) at ./main.c:24
```

[ソース]エディター上でもデータブレークポイントを設定することができます。

詳細については、"10.4.5 [ブレークポイント]ビュー"を参照してください。

オンチップエリア(CE)ブレーク機能(c33 oabコマンド) ICD3/ICD6モード、C33 ADVコア専用

このブレークはC33 ADVコアの機能を使用し、指定したCEエリアをアクセスした場合にブレークを発生させる機能です。BBCUバスマスタ、CEエリア、リード/ライト条件を指定可能です。

例1: CEエリア4～6、13、17～21にDMAまたはHBCUが書き込みを行った場合にブレークする設定
(gdb)

```
c33 oab on 0x05 w 0x3e2070
Area break on
BBCU bus master select : DMA,CPU
Read/Write             : Write
Area                   : 4,5,6,13,17,18,19,20,21
```

例2: オンチップエリア(CE)ブレークの解除

(gdb)
c33 oab off

オンチップエリア(CE)ブレークは指定のエリアに対する指定のメモリアクセス終了後に発生します。このブレークが発生したときは、次のようなメッセージを表示してコマンド入力待ちとなります。

例:

```
(gdb)
continue
Continuing.
Hardware on chip Area Break!
<<<Area Break Status>>>
Bus master : HBCU(CPU)
Access      : Write
Area        : 8
Data type   : Data
Data size   : Word
```

オンチップバスブレイク機能(c33 obbコマンド) ICD3/ICD6モード、C33 ADVコア専用

このブレイクはC33 ADVコアの機能を使用し、指定のバスアクセスが発生した場合にブレイクさせる機能です。バスの種類、BBCUバスマスタ、アドレス、データの種類と値、リード/ライト条件、アクセス回数などを指定可能です。また、発生した割り込みレベルを指定してブレイクさせることもできます。

例1: CPU系データバスで、ワードデータ0x55555555の書き込みが3回発生した場合にブレイクさせます。また、バス上のデータ値を32ビットで比較します。

```
(gdb)
c33 obb on 0 1 1 w 1 0x04 0 0 0 0 0x55555555 0xffffffff 3 0
Bus break on
Bus select                : CPU data bus
BBCU bus master select    : ---
Trigger/Break             : Break
Read/Write                : Write
Instruction/Data          : Data
Access size               : Word
Break by interrupt level  : Disable
Interrupt level           : 0
Data compare              : 32bit
Address                   : 0
Address mask              : 0
Data                     : 0x55555555
Data mask                 : 0xffffffff
Break counter             : 3
Break counter Clear/Save : Save
```

例2: オンチップバスブレイクの解除

```
(gdb)
c33 obb off
```

オンチップバスブレイクは指定のバスアクセスが終了後に発生します。このブレイクが発生したときは、次のようなメッセージを表示してコマンド入力待ちとなります。

```
(gdb)
continue
Continuing.
Hardware on chip Bus Break!
```

時間経過後ブレーク機能(c33 timebrkコマンド) ICD6モード

プログラム実行開始後、指定時間(1~300000ミリ秒単位)が経過した時点で強制ブレークさせる機能です。このブレーク機能は、ICD6モード時のみ使用可能です。

例1: 実行開始から1秒後に強制ブレークさせます。

```
(gdb)
c33 timebrk 1000
timer break on. [1000 ms]
```

例2: 時間経過後ブレークを解除します。

```
(gdb)
c33 timebrk 0
timer break off.
```

ブレーク時間を設定すると、"c33 timebrk 0"で解除するまで時間経過後ブレークは有効です。

プログラムを実行させると毎回、設定した時間でブレークします。

設定時間の経過前に他のブレーク条件に一致、あるいは[中断]ボタンをクリックした場合は、その時点で直ちにブレークします。

ブレークポイントの管理(ソフトウェアPCブレーク, ハードウェアPCブレーク, データブレーク)

ソフトウェアPCブレーク、ハードウェアPCブレーク、データブレークを設定すると、ブレークの種類にかかわらず、1から順にブレーク番号が付けられ、ブレーク設定コマンド実行時に[コンソール]ビューにメッセージとして表示されます(前記例参照)。この番号は、後からブレークポイントを個別に無効化/有効化、あるいは削除するときが必要となります。なお、ブレークポイントを削除しても、デバッグを終了するまで、番号の繰り上げ(消えた番号の再利用)は起きません。

設定したブレークポイント进行操作するには、以下のコマンドを使用します。

disableコマンド: ブレークポイントの無効化(ブレーク設定時は有効です。)

例: ブレーク番号1を無効化

```
(gdb)
disable 1
```

enableコマンド: ブレークポイントの有効化

例: ブレーク番号1を有効化

```
(gdb)
enable 1
```

deleteまたは**clear**コマンド: ブレークポイントの削除

例1: ブレーク番号1と2を削除

```
(gdb)
delete 1 2
```

例2: main.cの10行目のブレークポイントを削除

```
(gdb)
clear main.c:10
```

ignoreコマンド: ブレークを無効にする回数を指定

例: ブレーク番号2を2回無効に設定

```
(gdb)
ignore 2 2
Will ignore next 2 crossings of breakpoint 2.
```

info breakpointsコマンド: ブレークポイントリストの表示

例: ブレークポイントリストの表示

```
(gdb)
info breakpoints
Num Type           Disp Enb Address      What
1  breakpoint      keep y   0x00c00026 in main at ./main.c:11
    breakpoint already hit 1 time
    ignore next 10 hits
2  hw breakpoint   del  n   0x00c00038 in sub at ./main.c:20
3  hw watchpoint   keep y           i
    breakpoint already hit 6 times
```

詳細は、各コマンドの説明を参照してください。

また、[ブレークポイント]ビューにPCブレークポイントの一覧を表示させることができ、そこで無効化/有効化および削除の操作が行えます。詳細については、"10.4.5 [ブレークポイント]ビュー"を参照してください。

注: ICD2、ICD3、ICD6、デバッグモニタモードでターゲットプログラムの実行中にコールドリセットをかけると、チップ上でサポートされるハードウェアPCブレーク(内部的に使用するテンポラリブレークも含む)およびデータブレークはクリアされ、ブレークが発生しなくなります。一度他の要因でブレークすると再設定されますが、それまでは無効となりますので注意してください。

●[中断]ボタンによる強制ブレーク



[中断]ボタン

[デバッグ]ビュー上の[中断]ボタンは、プログラムが永久ループまたはスタンバイ(HALT、SLEEP)状態から抜け出せない場合など、実行中のプログラムを強制的に終了させるのに使用できます。

注: デバッグモニタモードにはこの強制ブレーク機能はありません。

●マップブレーク、不当命令実行によるブレーク(シミュレータモード)

プログラムが、無効な領域に対してアクセスを行った場合もブレークします。

注: ・以下のブレーク機能はシミュレータモードでのみ有効です。

- ・メモリマップに関係するブレークは、c33_rpfコマンドで読み込んだパラメータファイルにより設定されたメモリマップ情報に従って発生します。正しいパラメータファイルが読み込まれていない場合は、不当なブレークが発生する可能性があります。

ROM領域への書き込み

パラメータファイルで設定されるROM領域に対して書き込みが行われた場合にブレークします。ブレーク時のメッセージは次のとおりです。

Break by writing ROM area.

未定義領域へのアクセス

パラメータファイルで設定される、メモリマップ以外の未定義領域をアクセスした場合にブレークします。

Break by accessing no map.

スタックのオーバーフロー

スタックへの書き込みが、パラメータファイルで設定されるスタック領域をオーバーフローした場合にブレークします。

Break by stack overflow.

不当命令を実行

不当命令(アセンブラで生成されないコード)を実行した場合にブレークします。

Illegal instruction.

不当アドレスの命令を実行

不当なアドレス上の命令を実行した場合にブレークします。

Illegal address exception.

不当な遅延命令を実行

不当な遅延命令を実行した場合にブレークします。

Illegal delayed instruction.

halt、slp命令を実行

halt命令またはslp命令を実行した場合にブレークします。

Break by sleep or halt.

10.6.6 トレース機能

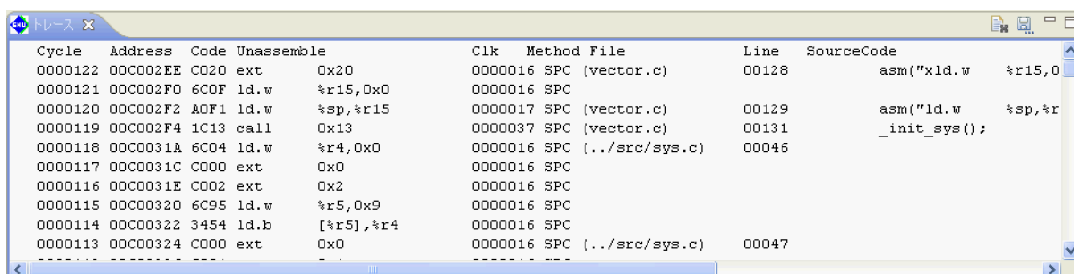
デバッグには、プログラムの実行をトレースする機能があります。
コネクトモードにより操作方法や機能が異なりますので注意してください。

注: デバッグモニタモードにはトレース機能はありません。

●ICD2/ICD3/ICD6モードのPCトレース機能

トレースメモリとPCトレース情報

S5U1C33000Hは131,072サイクル分、S5U1C33001Hは1,048,575サイクル分のトレースメモリを内蔵しており、S1C33チップが出力するデバッグ用信号等を利用して命令実行サイクルの情報を取得します。取得した情報は、コマンドにより[トレース]ビューに表示されます。また、ファイルに書き出すこともできます。



Cycle	Address	Code	Unassemble	Clk	Method	File	Line	SourceCode
0000122	00C002EE	C020	ext	0x20	0000016	SPC {vector.c}	00128	asm("xld.w %r15,0
0000121	00C002F0	6C0F	ld.w	%r15,0x0	0000016	SPC		
0000120	00C002F2	A0F1	ld.w	%sp,%r15	0000017	SPC {vector.c}	00129	asm("ld.w %sp,%r
0000119	00C002F4	1C13	call	0x13	0000037	SPC {vector.c}	00131	_init_sys();
0000118	00C0031A	6C04	ld.w	%r4,0x0	0000016	SPC {../src/sys.c}	00046	
0000117	00C0031C	C000	ext	0x0	0000016	SPC		
0000116	00C0031E	C002	ext	0x2	0000016	SPC		
0000115	00C00320	6C95	ld.w	%r5,0x9	0000016	SPC		
0000114	00C00322	3454	ld.b	[%r5],%r4	0000016	SPC		
0000113	00C00324	C000	ext	0x0	0000016	SPC {../src/sys.c}	00047	

トレース情報の内容は以下のとおりです。

Cycle:	トレースサイクル(10進数) トレースメモリ内に最後に取り込まれた情報が000000となります。
Address:	CPUの命令実行アドレス(16進数)
Code:	CPUが実行した命令コード(16進数)
Unassemble:	命令コードの逆アセンブル内容
Clk:	実行クロック数 c33 tmコマンドのパラメータにより、トレース開始からの積算表示、命令単位の表示が選択できます。(上記表示例は積算表示)
Method:	トレース解析(実行アドレス取得)方法 SPC: 初期PCによる解析 TRG: トリガアドレスによる解析 DPC: DPCO信号による解析 RET: call文、ret文による解析 MAP: マップ情報による解析 RTI: reti文による解析 ---: 解析不能
File:	実行命令を含むソースファイル名
Line:	ソース行番号
SourceCode:	ソースコード

PCトレースモードとトレース条件(c33 tmコマンド)

トレース範囲により3種類のPCトレースモードが設定されており、**c33 tm**コマンドで指定できます。

全体トレースモード(オーバーライトあり、オーバーライトなし)

プログラムの実行開始によりトレースを開始します。トレース条件として"オーバーライトあり"と"オーバーライトなし"が選択可能です。"オーバーライトあり"を選択した場合は、プログラムの実行を中断するまでトレース情報をトレースメモリに取り込みます。トレースメモリがいっぱいになった場合は古いデータから上書きされます。"オーバーライトなし"を選択した場合は、トレースメモリがいっぱいになった時点でトレースを終了します。

トレース情報のクロック数(Clk)を命令個別にカウントするか、累計するかについても選択できます。

例1: オーバーライトなしの全体トレースモード、実行サイクル数の積算表示を設定

(gdb)

```
c33 tm 1 1
```

例2: オーバーライトありの全体トレースモード、命令単位の実行サイクル数表示を設定

(gdb)

```
c33 tm 2 2
```

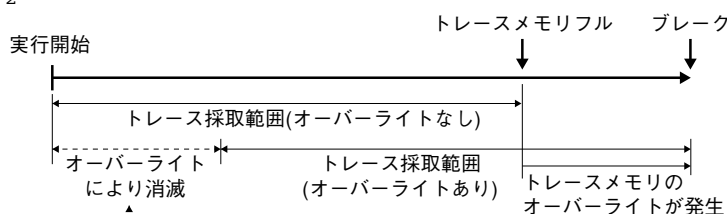


図10.6.7.1 全体トレースモード

範囲トレースモード

トレース範囲を指定し、プログラムがその範囲を実行した場合にのみトレース情報を取り込みます。トレース終了アドレスでプログラムの実行をブレークさせることもできます。また、時間測定条件(実行開始～ブレーク間の測定または範囲トレース時間の測定)も指定可能です。トレース情報のクロック数(Clk)を命令個別にカウントするか、累計するかについても選択できます。

例: アドレス0x600000から0x600100までの範囲トレースモード、実行サイクル数の積算表示、トレース後のブレークあり、トレース範囲内の実行時間測定を設定

(gdb)

```
c33 tm 3 1 0x600000 0x600100 2 2
```

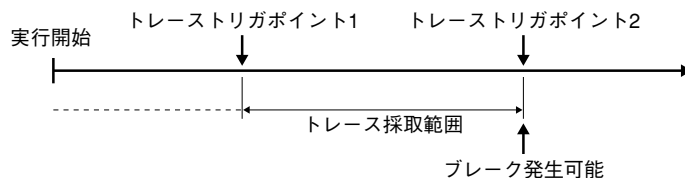


図10.6.7.2 範囲トレースモード

PCトレース情報の表示(c33 tdコマンド)

c33 tdコマンドにより、現在のトレースメモリ内のトレース情報を[トレース]ビューに表示することができます。サイクル番号で、表示範囲を指定することもできます。また、**c33 ts**コマンドを実行すると、検索された情報のみが表示されます。

例: 現在のトレースメモリにあるトレース情報の内、10Kクロック分を表示

(gdb)

```
c33 td
```

PCトレース情報の検索モード(c33 tsコマンド)

c33 tsコマンドにより、指定アドレスの命令を実行したサイクルとその前後のトレース情報のみを表示/保存対象にする検索モードに設定することができます。

例: 0x20100番地の命令を実行したサイクルとその前後5サイクルずつのトレース情報のみを表示/保存する検索モードに設定

```
(gdb)
c33 ts 0x20100 5 5
```

PCトレース情報の保存(c33 tfコマンド)

c33 tfコマンドにより、現在のトレースメモリ内のトレース情報をファイルに保存することができます。サイクル番号で保存範囲を指定することもできます。c33 tsコマンドを実行すると、検索された情報のみが保存されます。

例: 現在のトレースメモリにあるトレース情報の内、10Kクロック分をtrace.logファイルに保存

```
(gdb)
c33 tf trace.log
```

PCトレース情報の自動表示(c33 autotdコマンド)

c33 autotdコマンドにより、プログラムが停止するたびに自動的にトレース情報を[トレース]ビューに表示できます。これにより、c33 tdコマンドを実行する必要がなくなり、トレース結果の確認を速やかに行うことができます。

PCトレース情報の自動保存(c33 autotfコマンド)

c33 autotfコマンドにより、プログラムが停止するたびに自動的にトレース結果を保存できます。

注: • c33 td、c33 ts、c33 tf、c33 autotd、c33 autotfコマンドでトレース情報を取得するには、c33 rpfコマンドによるメモリマップ情報の設定が必要です。なお、c33 rpfコマンドは、targetコマンドより先に実行してください。プログラム実行直前にメモリマップ情報を設定すると、トレース情報を正常に取得できません。

- S5U1C33001Hはトレースメモリが大きいため、全トレースデータのファイル化にはかなりの時間が必要です。サイクル番号を指定するか、検索モードを設定して保存することをお勧めします。

例: サイクル番号0(最新)から1000までを保存

```
(gdb)
c33 tf trace.log 1000 0
```

ICD2/ICD3/ICD6モードのPCトレース動作と注意事項

ICD2/ICD3/ICD6モードでは以下の方法でトレースを行っています。

S5U1C33000H/S5U1C33001HにはターゲットCPUから以下の4本の信号が入力されます。

DST0, DST1, DST2 逐次命令実行、相対分岐実行、絶対分岐実行、アイドルなど、CPUの実行状況を伝えるステータス信号です。

DPCO 絶対分岐時、シリアルに分岐先のPC値を出力します。

この4ビットの情報をCPUクロックに同期して、S5U1C33000Hは最大128Kクロック分、S5U1C33001Hは最大1Mクロック分取り込みます。

gdbはこの情報と、gdb内部の逆アセンブル情報を基に次のようなフロー解析を行い、PC値を取得しています。

- DST0~2 = 逐次命令実行: +1命令
- DST0~2 = 相対分岐: 逆アセンブル情報から分岐命令数を計算
- DST0~2 = 絶対分岐: DPCO情報を参照し、分岐先を確定

ただし、トレースの起点とそれに対応したPC値が確定しないと解析ができません。

そこで、gdbは以下の手法によりPC値を確定しています。使用された手法はトレース情報内のMethodの表示内容で知ることができます。

Method: SPC	実行開始時のPC値が分かっている場合(オーバーライトなし全体トレース)
Method: DPC	絶対分岐で完全なDPCO情報が得られたことにより確定
Method: MAP	絶対分岐で不完全なDPCO情報を入力した場合はマップ情報を補完して確定 なお、絶対分岐には以下の命令と割り込みへの分岐が該当します。 call %rb, call.d %rb, jp %rb, jp.d %rb, ret, ret.d, reti, int
Method: TRG	範囲トレースではトリガアドレスを使用して確定
Method: RET	call文とret文が1対1に対応していることを前提として確定
Method: RTI	割り込み発生とretiが1対1に対応していることを前提として確定

このため、以下に示すような制約があります。

• オーバーライトありの場合の制約

相対分岐を繰り返すようなループでトレースを行うと、PC値が確定するまでトレース解析ができません。解決策として、8ビットタイマなどで数msごとに割り込みを発生させ、DPCO情報を出力させて、トレースを解析可能にする方法があります。(¥gnu33¥utility¥sample_std¥icdtrcのサンプル参照)

• オーバーライトなしの場合の制約

ソフトウェアPCブレークポイントからプログラムの実行を開始すると、**gdb**は次のような処理を行います。

1. 実行開始アドレスのソフトウェアPCブレークを解除
 2. 最初の命令をステップ実行
 3. 再度ソフトウェアPCブレークを設定
 4. 続く命令を連続実行
- ステップ実行したプログラムはトレースされません。

• 範囲トレースでの制約

通常、ハードウェアPCブレークは2本使用可能ですが、範囲トレースモードはそれらをトレーストリガとして使用するため、ハードウェアPCブレークが使用できなくなります。範囲トレースを解除すると、使用できる状態に戻ります。

以下は、全モード共通の制約です。

• 絶対分岐での制約

絶対分岐が27クロック以下で繰り返されると、PC値を完全に確定できません。マップ情報や、call~retのネスト情報によりリカバリ処理を試みますが、解析不能となる場合があります。

• 実行プログラムの制約

デバッグは、fileコマンドで読み込んだデバッグ情報を保存し、その逆アセンブル情報をもとに解析を進めます。

以下のようなケースでは内部の解析情報とターゲットのプログラムに食い違いがでます。

- 実行プログラムがROMにある場合
- プログラムが動的に実行ルーチンをコピー / 移動する場合
- **gdb**のコマンド等でプログラム領域を変更した場合

• シミュレーテッドI/O

シミュレーテッドI/Oは、ソフトウェアPCブレーク、ステップ実行、ソースステップ実行機能を内部で使います。トレース実行とは併用しないでください。

• 実行コマンド

トレースデータはcontinueコマンドによる連続実行時に採取されます。ステップ実行等では採取されません。また、continueコマンドが入力されると、**gdb**は以前のトレースデータを削除します。

以下は、S5U1C33000H/S5U1C33001Hハードウェアにかかわる注意点です。

• S5U1C33000H/S5U1C33001Hの設定

ICD2/ICD3モードでトレース機能を使用するには、S5U1C33000HのディップスイッチSW4をOPEN(上)に設定してください。

ICD6モードでトレース機能を使用するには、S5U1C33001HのディップスイッチSW9をON(下)に設定してください。また、ターゲットボードとS5U1C33000H/S5U1C33001Hは、10ピンケーブルを使用してトレースに必要な信号(DST0、DST1、DST2、DPCO)をすべて接続しておく必要があります。

・S5U1C33000H/S5U1C33001Hトレースの上限周波数

S5U1C33000H/S5U1C33001Hによるトレースは50MHzが動作の上限となります。それ以上ではデータ化けを引き起こします。

50MHz以上のCPU動作クロックを使用する場合は、DIP SWでトレース機能をOFFにしてください。また、S1C33チップのBCU(バス)の速度をCPUコアの1/2以下に設定(#X2SPD端子で設定)してください。

トレース以外の機能はバスと同じ速度で動作し、40MHzが上限となります。

●シミュレータモードのトレース機能

シミュレータモードでは**c33 tm**コマンドでトレース機能のON/OFF、表示方法およびファイル書き出しが指定できます。トレース機能をONすると、1命令実行ごとにトレース結果を表示あるいはファイルに保存します。

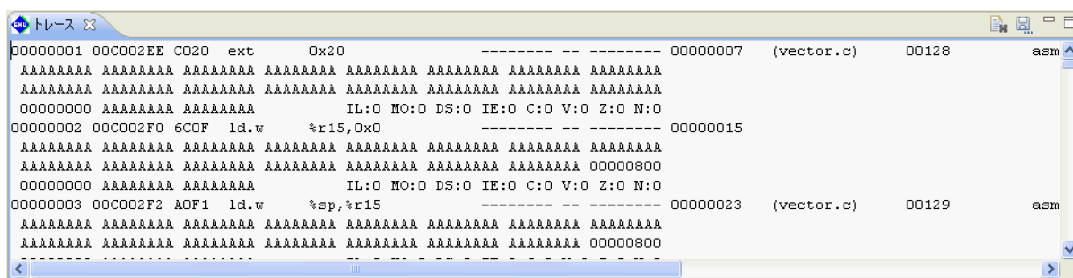
例1: ソース情報付きレジスタ表示のトレースを設定し、情報を保存するファイルtrace.logを指定(gdb)

```
c33 tm on 4 trace.log
```

例2: トレースモードを解除

(gdb)

```
c33 tm off
```



シミュレータモードで表示されるトレース情報は次のとおりです。

<各トレース情報の1行目>

Number Address Code Disassemble Address Type Data Clock [File Line SourceCode]

Number: 実行命令No.(10進数)

CPUをリセットまたはトレースをONにしてからの実行命令数

Address: 実行命令アドレス(16進数)

Code: 命令コード(16進数)

Disassemble: 実行命令の逆アセンブル内容

Address: アクセスしたメモリアドレス(16進数)

Type: バスオペレーションタイプ

rB: バイトデータリード、rH: ハーフワードデータリード、rW: ワードデータリード

wB: バイトデータライト、wH: ハーフワードデータライト、wW: ワードデータライト

Data: 読み出し/書き込みデータ(16進数)

Clock: 実行クロック数(10進数)

File: ソースファイル名(c33 tmコマンドでソース表示を選択した場合のみ)

Line: ソースの行番号(c33 tmコマンドでソース表示を選択した場合のみ)

SourceCode: ソースコード(c33 tmコマンドでソース表示を選択した場合のみ)

<各トレース情報の2~4(6)行目>

これらの行は、c33 tmコマンドでレジスタ内容の表示を選択した場合にのみ表示されます。

各レジスタ値は次のような配置で表示されます。

C33 STDの場合

R0	R1	R2	R3	R4	R5	R6	R7
R8	R9	R10	R11	R12	R13	R14	R15
SP	AHR	ALR		PSR(フラグ個別に表示)			

S1C33401 (C33 ADVコア)の場合

R0 R1 R2 R3 R4 R5 R6 R7
 R8 R9 R10 R11 R12 R13 R14 R15
 SP AHR ALR PSR(フラグ個別に表示)
 LCO LSA LEA SOR TTBR DP USP SSP
 PSR(C33 ADVコアの追加フラグを個別に表示)

表示はc33 tmコマンドで[トレース]ビューへの表示を選択した場合に[トレース]ビューに表示されます。

ファイルへの保存を選択した場合は、指定のファイルに出力され、[トレース]ビューには表示されません。

注: 表示されるクロック数(Clock)は、パラメータファイルに設定されたウェイト数情報を使い算出しています。パラメータファイルの設定に誤りがあると正しく表示されません。

●オンチップバストレース(ICD3/ICD6モード, C33 ADVコア専用)

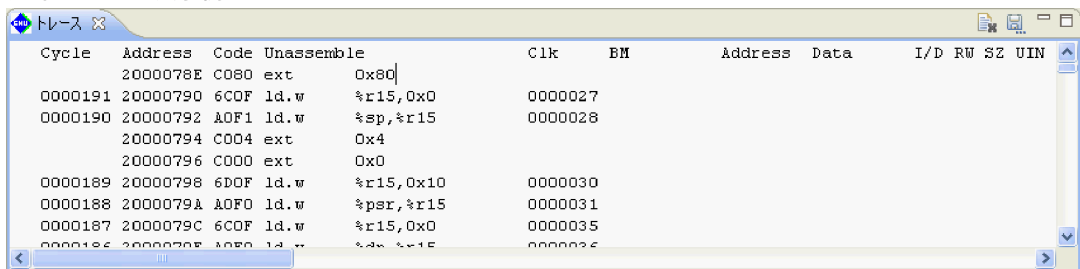
ICD3/ICD6モードでC33 ADVコアを搭載したターゲットのデバッグを行う場合、C33 ADVコアの内蔵機能を使用したオンチップバストレースが行えます。バストレースは、指定したバス上で設定した条件のアクセスが発生するとその情報をトレースメモリに取得する機能です。

C33 STDまたはC33 PEの場合も、c33 logianaコマンドを実行することにより、バストレース情報を表示/保存が可能となります。

バストレース情報

バストレース情報も前述のPCトレース情報と同様に、S5U1C33001Hのトレースメモリに記録されます。[トレース]ビューに表示されるバストレース情報を以下に示します。

通常モードの表示例



Cycle	Address	Code	Unassemble	Clk	BM	Address	Data	I/D	RW	SZ	UIN
	2000078E	C080	ext	0x80							
0000191	20000790	6C0F	ld.w	*r15,0x0	0000027						
0000190	20000792	A0F1	ld.w	*sp,*r15	0000028						
	20000794	C004	ext	0x4							
	20000796	C000	ext	0x0							
0000189	20000798	6D0F	ld.w	*r15,0x10	0000030						
0000188	2000079A	A0F0	ld.w	*psr,*r15	0000031						
0000187	2000079C	6C0F	ld.w	*r15,0x0	0000035						
0000186	2000079E	A0F0	ld.w	*psr,*r15	0000036						

Cycle:	トレースサイクル(10進数) トレースメモリ内に最後に取り込まれた情報が000000となります。
Clk:	実行クロック数 トレース開始から積算されたクロック数が表示されます。
BM:	バスの種類 CPU バスマスタは、CPU(HBCU設定時) CPU/CACHE バスマスタは、CPUまたはCACHE(BBCU設定時) DMA/USR バスマスタは、DMAまたはUSR(BBCU設定時)
Address:	アクセスしたアドレス(16進数)
Data:	リード/ライトしたデータ(16進数)
I/D:	データの種類 i 命令 d データ
RW:	リード/ライト r リード w ライト
SZ:	アクセスサイズ B バイト H ハーフワード W ワード

UIN: USER_IN信号の状態
0または1
Unassemble: 命令コードの逆アセンブル内容

ユーザモードの表示例

Cycle	Clk	DTD	DTS	UIN
0000118	0000000	00000000	00000	0
0000117	0000001	00000000	00000	0
0000116	0000002	00000000	00000	0
0000115	0000003	00000000	00000	0
0000114	0000004	00000000	00000	0
0000113	0000005	00000000	00000	0
0000112	0000006	00000000	00000	0
0000111	0000007	00000000	00000	0
0000110	0000008	00000000	00000	0

Cycle: トレースサイクル(10進数)
トレースメモリ内に最後に取り込まれた情報が000000となります。

Clk: 実行クロック数
トレース開始から積算されたクロック数が表示されます。

DTD: バストレースデータ(DTD端子出力)

DTS: バストレースデータステータス(DTS端子出力)

UIN: USER_IN信号の状態
0または1

上記の例はどちらのモードもバストレース情報のみを表示していますが、PCトレース情報も合わせて表示させるように設定することも可能です。

オンチップバストレースの設定(c33 obtコマンド)

バストレース条件はc33 obtコマンドで設定します。

例: トレース条件をUSERによるワードアクセス、アドレスバス幅を4バイト、圧縮出力はしない設定にします。 また、ユーザ入力(ブレードトリガブレード)のトレース信号として外部トレース入力信号(TRC IN信号)を指定します。

(gdb)

```
c33 obt 0 2 0x8 rw 2 0x04 0 4 0 0
```

```
Set bus trace
```

```
Trace mode : Normal mode
```

```
Bus select : BBCU
```

```
BBCU bus master select : USER
```

```
Read/Write : Read/Write
```

```
Instruction/Data : Instruction/Data
```

```
Access size : Word
```

```
Bus output : Address bus
```

```
Address bus size : 4 byte
```

```
Compression : Disable
```

```
Bus trace trigger select : TRC IN signal
```

オンチップバストレース情報の表示(c33 otdコマンド)

c33 otdコマンドにより、現在のトレースメモリ内のバストレース情報のみ、またはPCトレースおよびバストレース情報を[トレース]ビューに表示することができます。サイクル番号で、表示範囲を指定することもできます。

例1: 現在のトレースメモリにあるバストレース情報の内10Kクロック分を表示

(gdb)

```
c33 otd 0
```

例2: 現在のトレースメモリにある1000サイクルから最新サイクルのPCトレースおよびバストレース情報を表示

(gdb)

```
c33 otd 1 1000 0
```

オンチップバストレース情報の保存(c33 otfコマンド)

c33 otfコマンドにより、現在のトレースメモリ内のバストレース情報のみ、またはPCトレースおよびバストレース情報をファイルに保存することができます。サイクル番号で保存範囲を指定することもできます。

例1: 現在のトレースメモリにあるバストレース情報の内10Kクロック分をtrace.logファイルに保存

(gdb)

```
c33 otf 0 trace.log
```

例2: 現在のトレースメモリにある1000サイクルから最新サイクルのPCトレースおよびバストレース情報をtrace.logファイルに保存

(gdb)

```
c33 otf 1 trace.log 1000 0
```

●トレースデータ表示とファイル作成時間

PCの性能に伴う、トレースデータの表示およびファイル作成時間の目安を以下に示します。

10K Clkあたりの出力時間(10K Clkは、サイクル範囲指定しないときのサイズです。)

コマンド	Pentium-3 450MHz	Pentium-4 3GHz
c33 td	10秒	3秒
c33 otd 0	13秒	3秒
c33 otd 1	23秒	5秒
c33 tf	2秒	2秒
c33 otf 0	1秒	1秒
c33 otf 1	2秒	2秒

[トレース]ビューへの数百行～数千行の表示にはかなりの時間を要しますので、サイクル範囲を指定することをお勧めします。

10.6.7 シミュレーテッドI/O

gdbのシミュレーテッドI/O機能により、シリアルインタフェースなどの外部入出力機能を標準出力あるいはファイルの入出力により評価することができます。

●**stdin**による入力(**c33 stdin**コマンド)

c33 stdinコマンドでは、以下の条件を設定します。

- ブレークアドレス
- 入力バッファアドレス(バッファサイズは65バイト固定)
- 入力ファイル

これらの条件を設定後、プログラムを連続実行させます。

入力ファイルを指定した場合

gdbは設定したブレークアドレスになると指定のファイルからデータをバッファに入力します。その後、プログラムの実行をブレークしたアドレスから再開します。

例1: データ入力機能を設定

(gdb)

```
c33 stdin 1 READ_FLASH READ_BUF input.txt
```

例2: データ入力機能を解除

(gdb)

```
c33 stdin 2
```

入力ファイルを指定しない場合

c33 stdinコマンドは必ず入力ファイル名を指定してください。

指定しない場合はエラーになります。

●**stdout**による出力(**c33 stdout**コマンド)

c33 stdoutコマンドでは、以下の条件を設定します。

- ブレークアドレス
- 出力バッファアドレス(バッファサイズは65バイト固定)
- 出力ファイル(省略時は[コンソール]ビューへの出力)

これらの条件を設定後、プログラムを連続実行させます。

出力ファイルを指定した場合

gdbは設定したブレークアドレスになると[コンソール]ビューにバッファの内容を表示します。その後、プログラムの実行をブレークしたアドレスから再開します。

例1: データ出力機能を設定

(gdb)

```
c33 stdout 1 WRITE_FLASH WRITE_BUF output.txt
```

例2: データ出力機能を解除

(gdb)

```
c33 stdout 2
```

出力ファイルを指定しない場合

gdbは設定したブレークアドレスになると[コンソール]ビューにバッファの内容を表示します。その後、プログラムの実行をブレークしたアドレスから再開します。

例:[コンソール]ビューを使用したデータ出力機能を設定

(gdb)

```
c33 stdout 1 WRITE_FLASH WRITE_BUF
```

●プログラム上の対応

シミュレーテッドI/O機能を使用するには、プログラム内に以下の定義が必要です。

入出力バッファの定義

gdbがデータの入出力に使用するグローバルなバッファを次のような形式で定義しておきます。

入力バッファの定義: `unsigned char READ_BUF[65]`

出力バッファの定義: `unsigned char WRITE_BUF[65]`

バッファ名には、シンボル名の規定にそった任意の名称を使用することができます。

バッファサイズは65バイトに固定してください。

`c33 stdin`、`c33 stdout`コマンド実行時には、このシンボル名でバッファアドレスを指定します。

データが入力されると、`READ_BUF[0]`には実際に入力されたデータのサイズ(1~64)が入ります。

EOFが入力された場合`READ_BUF[0]`は0となります。入力データは`READ_BUF[1]`以降に格納されます。

データを出力する場合は`WRITE_BUF[0]`に出力するデータのサイズ(1~64)を書き込み、出力データを`WRITE_BUF[1]`以降に書き込みます。EOFを出力する場合は`WRITE_BUF[0]`に0を書き込みます。

gdbとプログラム間で、最大64バイトのデータ列として入出力を行えます。

データ更新用グローバルラベルの定義

gdbが入力バッファにデータを取り込む位置、および出力バッファのデータを出力する位置にそれぞれ次のようなグローバルラベルを定義しておきます。

入力位置: `.global READ_FLASH`

`READ_FLASH:`

出力位置: `.global WRITE_FLASH`

`WRITE_FLASH:`

ラベルには任意の名称が使用できます。

`c33 stdin`、`c33 stdout`コマンド実行時には、このシンボル名でブレークアドレスを指定します。

C/C++ソースでは、標準入出力ライブラリ関数の下位レベル関数`write`、`read`(7.4.4節参照)内にこれらのラベルを定義します。

実際のプログラム例については、サンプルプログラムやデバッガのコマンドファイルが`¥gnu33¥utility¥sample_std¥ansilib¥simulator`ディレクトリにインストールされていますので、参照してください。

gdbは`READ_FLASH`でブレークするとファイルのデータを読み込み、定義されている入力バッファにロードします。その後、プログラムの実行を再開します。

`WRITE_FLASH`でブレークした場合、**gdb**は出力バッファに書き込まれているデータをファイルに出力し、プログラムの実行を再開します。

●注意事項

- `c33 stdin`、`c33 stdout`コマンドで指定するブレークアドレスはソフトウェアPCブレークアドレスと重複することはできません。
- 内部的にはソフトウェアPCブレークを使用しているため、ターゲットボード上のROM領域は指定できません。
- 入出力にはASCIIキャラクタのみを使用してください。バイナリデータ(特に0x0や0x1a)では誤動作します。
- `c33 stdin`、`c33 stdout`により入出力を行う部分は`continue`コマンドで連続実行させ、ステップ実行は行わないでください。また、その近辺ではブレークが発生しないようにしてください。
- `c33 stdin`コマンドでは、必ず入力ファイル名を指定してください。

10.6.8 フラッシュメモリ操作

gdbには、ターゲットボード上のフラッシュメモリを操作する機能、およびS5U1C33001Hを使用するフラッシュライター機能があります。

●ターゲットボード上のフラッシュメモリ操作

gdbには、S1C33チップに内蔵あるいはターゲットボード上のフラッシュメモリに対してデータ書き込み/消去を行うためのユーティリティとコマンドが用意されています。

ICD2、ICD3、ICD6、デバッグモニタモードのデバッグ環境で使用することができます。

フラッシュメモリへの書き込みは以下の手順で行います。

より詳しい内容については、フラッシュサポートユーティリティ **fls33g**のreadme.txtを参照してください。(readme.txtは、¥gnu33¥tool¥fls33gディレクトリにあります。)

1. フラッシュルーチンのロード

loadコマンドで、フラッシュルーチン(消去ルーチン、書き込みルーチン)を内蔵RAM等へロードします。

例:

```
(gdb)
file 291v800t.elf           (デバッグ情報の読み込み)
(gdb)
target icd com1             (ターゲットの接続)
C33 ICD33 debugging
Connecting with target ... done
Target connection test ... done
Initializing ..... done
CPU cold resetting ..... done
ICD hardware version ... 00
ICD software version ... 2.0
Boot address ..... 0xC00000
0x00c00000 in ?? ()
(gdb)
set {char}0x4812d=0x59       (ブートベクタアドレスの設定)
(gdb)
set {int}0x48134=0x600000
(gdb)
load                         (フラッシュルーチンのロード)
```

このルーチンで、実際の消去、書き込みを行います。

セイコーエプソン提供のルーチンは、内蔵RAMの0x30～0x7ffを使用します。

注: フラッシュルーチンは、セイコーエプソン提供のものを使用するか、独自のものを作成してください。セイコーエプソン提供のルーチンは、¥gnu33¥tool¥fls33gディレクトリにあります。AMD系フラッシュを中心に用意されています。ソースコード等も含まれていますので、必要により修正して使用してください。

2. フラッシュの設定(c33 flsコマンド)

フラッシュメモリのスタートアドレス、エンドアドレス、ステップ1でロードした消去ルーチン、書き込みルーチンのエントリアドレスを**gdb**に設定します。

例: フラッシュメモリ領域が0xc00000～0xcfffff、消去ルーチンと書き込みルーチンの開始アドレスがFLASH_ERASEとFLASH_LOADの場合

```
(gdb)
c33 fls 0xc00000 0xcfffff FLASH_ERASE FLASH_LOAD
```

3. フラッシュの消去(c33 fleコマンド)

フラッシュメモリを全消去、あるいはセクタ消去します。

フラッシュの内容は0xffに変わります。

例: フラッシュの先頭アドレスが0xc00000で、全消去する場合

```
(gdb)
c33 fle 0xc00000 0 0
```

c33 flsコマンドの次に必ずc33 fleコマンドを実行してください。フラッシュを消去したくない場合は、セクタ範囲(上記例の"0 0")に"-1 0"を指定することにより、コントロールレジスタの変更のみを行います。

4. フラッシュへの書き込み

フラッシュメモリへのプログラムの書き込みにはloadコマンドを使用します。

例:

```
(gdb)
load sample.elf
```

フラッシュメモリへのデータ書き込みにはsetコマンドを使用します。

例: アドレス0xc00002に0x1234を書き込む場合

```
(gdb)
set {short}0xc00002 = 0x1234
(gdb)
x /8h 0xc00000
C00000: FFFF 1234 FFFF FFFF FFFF FFFF FFFF FFFF
```

16ビットデバイスの場合、set {short}コマンドを使用してください。バイト単位の書き換えはサポートされていません。

書き込んだ内容はxコマンドで確認できます。

ステップ2のc33 flsコマンドで設定したフラッシュメモリ領域に入るデータは、ステップ1でロードしたフラッシュ書き込みルーチンに渡してフラッシュ書き込みを行い、それ以外は通常どおり、RAMへの書き込みとして処理されます。フラッシュメモリが消去されていない(0xffでない)場合はエラーが返ります。

●S5U1C33001H(ICD3/ICD6モード)のフラッシュライター機能

S5U1C33001Hはフラッシュライター機能を内蔵しており、gdbにはこの機能を制御するコマンドが用意されています。

S5U1C33001Hをフラッシュライターとして使用するには、DSW1-1をON(下)にセットしてください。詳細については、"S5U1C33001H Manual(S1C33 Family In-Circuit Debugger)"を参照してください。

フラッシュライター機能用のコマンドは以下のとおりで、ICD3/ICD6モード時のみ使用可能です。

プログラム/データの消去(c33 fweコマンド)

S5U1C33001Hにロードしたデータ消去/書き込みプログラム、または書き込みデータとアドレス情報を消去します。

例1: 書き込みデータの消去

```
(gdb)
c33 fwe 0
```

例2: データ消去/書き込みプログラムの消去

```
(gdb)
c33 fwe 1
```

プログラムのロード(c33 fwlpコマンド)

データ消去/書き込みプログラムをホストからS5U1C33001Hにロードし、消去/書き込みルーチンのエントリ情報を設定します。

例: データ消去/書き込みプログラムファイルがwriter.sa、消去ルーチンと書き込みルーチンの開始アドレスが0x90と0xb4の場合

```
(gdb)
c33 fwlp writer.sa 0x90 0xb4
```

データのロード(c33 fwld、c33 fwdcコマンド)

c33 fwldコマンドは、フラッシュに書き込むデータをホストからS5U1C33001Hにロードします。

c33 fwdcコマンドは、ターゲットボード上のメモリに格納されているデータをS5U1C33001Hにロードします。フラッシュの消去範囲も設定します。

例1: 全ブロックの消去を指定し、sample.saをロード

```
(gdb)
```



```
c33 fwld sample.sa 0 0 0
```

例2: 全ブロックの消去を指定し、ターゲットメモリ上のアドレスFLASH_STARTから1MBのデータをロード

```
(gdb)
```

```
c33 fwdc FLASH_START 0x100000 0 0 0
```

フラッシュライタ情報の表示(c33 fwdコマンド)

S5U1C33001Hにロードされているデータの情報、および消去/書き込みプログラムの情報を表示します。

```
(gdb)
```

```
c33 fwd
```

```
CPU data address      : xxxxxxxx
Data size             : xxxxxxxx
Erase start block     : xxxxxxxx
Erase end block       : xxxxxxxx
Erase parameter       : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

CPU program address   : xxxxxxxx
Program size          : xxxxxxxx
Erase routine entry address : xxxxxxxx
Write routine entry address : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

10.6.9 ビッグエンディアンへの対応について

C/C++コンパイラからリンカまでのツールおよびライブラリは、リトルエンディアンにのみ対応しています。

C/C++コンパイラでは、ビッグエンディアン領域にロードできるelfファイルを作成できませんので注意してください。ただし、データの参照についてはデバッガがビッグエンディアンに対応しています。

●ビッグエンディアンの指定方法(シミュレータモード)

シミュレータモード時、デバッガにビッグエンディアン領域の情報を設定するには、パラメータファイルのマッピング情報で指定します。ビッグエンディアンに設定したい領域については、第5パラメータにBを記述します。この記述がない場合はリトルエンディアン領域になります。ただし、開発するS1C33チップがビッグエンディアンに対応した機種である必要があります。また、内蔵のROM、RAM、I/Oはビッグエンディアンに設定することはできません。

パラメータファイルについては、"10.8 パラメータファイル"を参照してください。

●デバッグコマンドの動作

メモリ操作コマンド(x, set, c33 fb/fh/fw, c33 mvb/mvh/mvw)

コマンドのデータタイプに従ってバイト、ハーフワード、ワード単位の読み出し/書き込みを行いますので、エンディアンに対応した動作、表示を行います。

loadコマンド

エンディアンに対応したデータスワップを行い、ハーフワード単位でデータを書き込みます。そのため、C/C++コンパイラで作成したプログラムをビッグエンディアン領域に正しくロードできません。

10.6.10 プロファイラ・カバレッジ機能

プロファイラ・カバレッジ機能は、プログラム中で性能を落とす原因としてボトルネックとなる箇所(関数など)の検出を主な目的とする。プロファイラ・カバレッジ機能は、シュミレータモードのみで動作する。

●機能概要

プロファイラ・カバレッジ機能は、以下の2つの機能に分けられます。

① GDB内のシミュレータ上での計測

プロファイラ計測 : 関数ごとに消費したサイクル数や実行回数を計測します。
 コード・カバレッジ計測 : 命令ごとに実行したか否かを記録します。
 計測範囲 : リセット+GOからブレーク間(simulated I/O内部ブレークは除く)

② 計測データ表示

プロファイラ結果表示実行ファイルとカバレッジ結果表示実行ファイルの2つのツールにより計測データを表示します。

[1]プロファイラ結果表示ツール

①で計測したデータを表示・保存・比較表示する機能を持つWindowsアプリケーションです。
gdbのc33 profileコマンド、[デバッグ]ビューの[プロファイラ]ボタン、または[実行]メニューの[プロファイラ]を選択することにより実行されます。

実行ファイル名	gnuProf.exe
入力ファイル	プロファイラ計測データファイル(*.prf) プロファイラ計測データファイルのパス格納ファイル(c33_profile_path.gdb)
出力ファイル	プロファイラ計測データファイル(*.prf) 表示結果テキストファイル(*.txt / *.csv)

[2]カバレッジ結果表示ツール

①で計測したデータを表示・保存・マージする機能を持つWindowsアプリケーションです。
gdbのc33 covergaeコマンド、[カバレッジ]ボタン、または[実行]メニューの[カバレッジ]を選択することにより実行されます。

実行ファイル名	gnuCvrg.exe
入力ファイル	プロファイラ計測データファイル(*.prf) プロファイラ計測データファイルのパス格納ファイル(c33_profile_path.gdb)
出力ファイル	プロファイラ計測データファイル(*.prf) 表示結果テキストファイル(*.txt / *.csv)

注：・このツールは、Windowsアプリケーションですが、単独で起動して使用することは想定されていません。また、そのときの動作は保障していません。

- ・「プロファイラ計測データファイル」は、プロファイラ・カバレッジ計測情報をすべて含むファイルで、上記2つのウィンドウで使用される共通のファイルです。

●機能一覧

主要な機能一覧を以下に示します。

表10.6.10.1 機能一覧

大分類	中分類	小分類
計測データ取得	計測処理	ユーザプログラム実行時、計測データを取得するかしないかを指定可能 (c33_profilemd)
		ユーザプログラム実行時、計測データを取得
	プロファイル コマンド (c33_profile)	計測データをファイルに保存
		プロファイラ・ウィンドウのオープン
	カバレッジコマンド (c33_coverage)	計測データをファイルに保存 カバレッジ・ウィンドウのオープン
計測データ表示	プロファイラ・ ウィンドウ	関数とそれよりも下の関数(子関数)を含めないサイクル数表示
		関数とそれよりも下の関数(子関数)を含めたサイクル数表示
		2つの計測結果を比較して、関数ごとのサイクル数やコードサイズの差分表示
		指定した項目をソートして表示
		プロファイラ計測データファイルの読み出し
		プロファイラ計測データファイルの保存
		表示結果をファイル(テキストまたはCSV形式)へ保存
	カバレッジ・ ウィンドウ	関数毎のコード・カバレッジの表示
		アドレス毎のコード・カバレッジの表示
		指定した項目をソートして表示
		プロファイラ計測データファイルの読み出し
		プロファイラ計測データファイルの保存
		表示結果をファイル(テキストまたはCSV形式)へ保存
		前回保存したカバレッジ結果とマージして表示
		関数名を指定して、実行/未実行箇所をソースレベルで表示 (ソースファイル・カバレッジウィンドウの表示)

●機能詳細

計測データの表示について説明します。

プロファイラ・ウィンドウ

C33 profileコマンド、[プロファイラ]ボタン、または[デバッグ]ビューのコンテキストメニューから[プロファイラ]を選択することによりプロファイラ・ウィンドウが開き、計測データがあれば表示します。計測データがない場合は、エラーメッセージ"No profiling data"が表示され、[OK]ボタンをクリックするとプロファイラ・ウィンドウが開きます(データは表示されません)。また、プロファイラ・ウィンドウは、GDBが終了すると連動して終了します。

比較結果状態：リストに比較結果を表示しているとき、"COMPARISON RESULT"と表示します(それ以外の場合は空白です)。

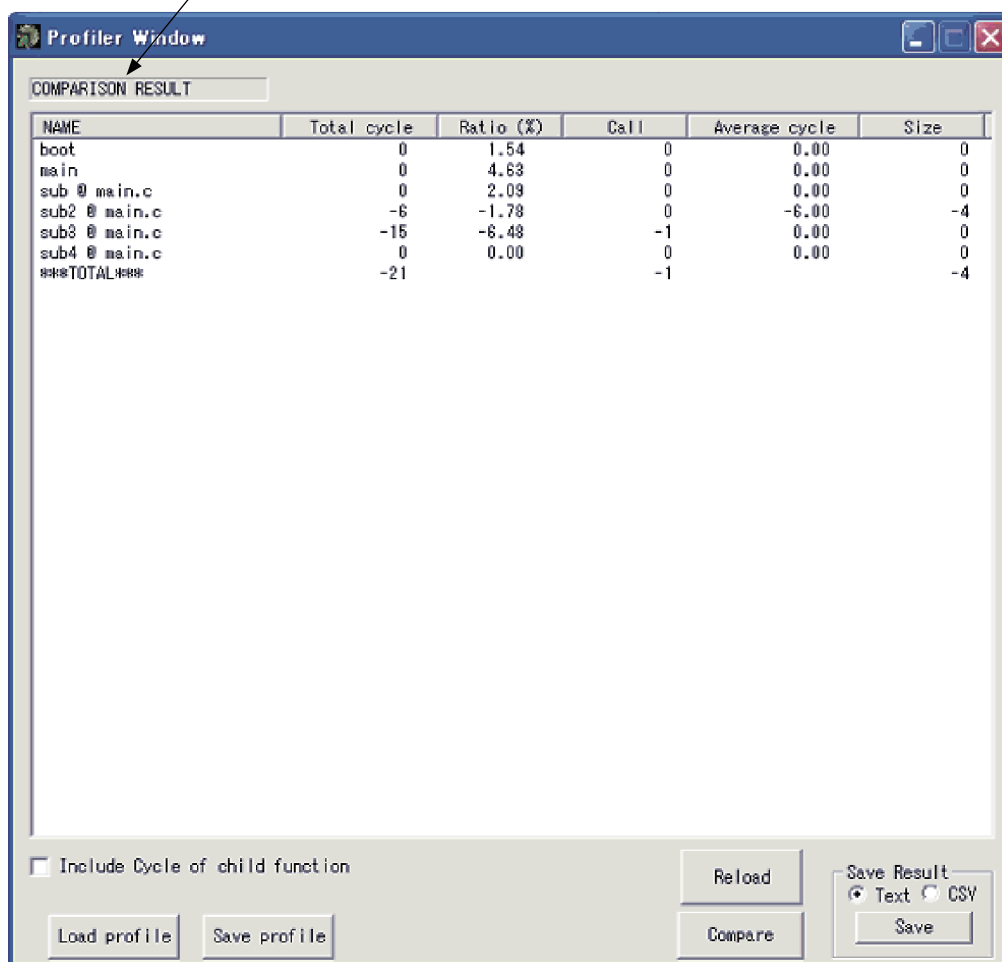


図10.6.1 プロファイラ・ウィンドウ

表示項目

Name :	関数名(最大256文字) Static関数は、「関数名@ファイル名」形式で表示します。 最終行の"***TOTAL***"は、合計を示します。
Total cycle :	関数の総実行サイクル(10進数、最大32bit長)
Ratio(%) :	関数の実行サイクルの比率(Total cycle÷合計のTotal cycle、単位：%、少数点以下第2位まで表示)
Call :	関数の呼ばれた回数(10進数、最大32bit長)
Average cycle :	関数の1回あたりの平均実行サイクル(Total cycle÷Call、10進数、最大32bit長)
Size :	関数内で実行した命令のサイズ(単位：バイト、10進数、最大24bit長)

- ・各項目名をクリックするとソートします。デフォルトはRatio(%)で値の高い順に表示します。ソートは、クリックするたびに昇順・降順表示が切り替わります。
- ・プロファイラ・ウィンドウ起動時、計測データがないとき、もしくはエラーのときは、何も表示しません。
- ・実行されなかった関数も表示します。
- ・比較結果は、[Load profile]ボタンによって別のファイルをロードしたり、[Include Cycle of child function]のチェックの状態を変更すると比較前の表示に戻ります。

[Include Cycle of child function]

チェックすると、子関数も含めたサイクル数を表示します。

チェックを外すと、子関数のサイクルを含めないサイクル数を表示します(デフォルト)。

[Load profile]ボタン

ファイルダイアログを開き、プロファイラ計測データファイルをロードします。

[Save profile]ボタン

ファイルダイアログを開き、プロファイラ計測データファイルにプロファイル情報を保存します。

比較後の表示状態でも、保存するデータは最初にロードしたデータとなります。

[Save Result]ボタン

ファイルダイアログを開き、表示されているデータを保存します。

テキストファイル形式(デフォルト)、またはCSVファイル形式での保存が可能です。

[Reload]ボタン

比較結果が表示されているときに、ウィンドウ起動後または、ロード後の表示に戻します。

[Compare]ボタン

現在表示されているプロファイル情報と、保存されているプロファイル情報との比較を行います。ファイルダイアログを開き、比較対象のプロファイラ計測データファイルを指定します。比較結果は、"Include Cycle of child function"の設定に合わせて表示されます。以下の比較計算処理を行います。

- (1) 表示されている関数名が比較対象に存在する場合、比較した結果を表示します。
(表示されている値－指定したファイルの値)で表示されます。
- (2) 表示されている関数名が比較対象に存在しない場合、関数が追加されたということなので、その関数行が追加されて(表示されている値)で表示されます。
- (3) 関数が比較対象にしか存在しない場合、関数が削除されたということなので、(0－指定したファイルの値)で表示されます。
- (4) TOTAL項目は、(表示されている値のTOTAL－指定したファイルの値のTOTAL)で表示されます。
表示された値の意味
正の値：増加分＝改悪されたという意味
負の値：減少分＝改善されたという意味
0 : 変化なし

これは、ソースの修正により、関数の追加/削除が行われることがあるためです。

表10.6.1.2 比較結果の例

表示されている関数	比較する関数	表示関数	値	ケース
A	B'	A	A	//(2)
B	C'	B	B-B'	//(1)
C	D'	C	C-C'	//(1)
E	E'	E	E-E'	//(1)
		D	0-D'	//(3)

<Save Resultボタンで作成されるテキストファイルのフォーマット>

(1) "text"選択時の出力例

Profile result

Total	cycle	Ratio(%)	Call	Average cycle	Size	Name
	17	10.43	1	17.00	40	boot
	51	31.29	1	51.00	44	main
	23	14.11	1	23.00	40	sub @ main.c
	27	16.56	1	27.00	38	sub2 @ main.c
	45	27.61	3	15.00	30	sub3 @ main.c
	0	0.00	0	0.00	30	sub4 @ main.c
163		100.00	7		222	*** TOTAL ***

(2) "csv"選択時の出力例

Total	cycle	Ratio(%)	Call	Average cycle	Size	Name
0,	1.54,	0,	0.00,	0,	boot	
0,	4.63,	0,	0.00,	0,	main	
0,	2.09,	0,	0.00,	0,	sub @ main.c	
-6,	-1.78,	0,	-6.00,	-4,	sub2 @ main.c	
-15,	-6.48,	-1,	0.00,	0,	sub3 @ main.c	
0,	0.00,	0,	0.00,	0,	sub4 @ main.c	
-21,	,	-1,	,	-4,	*** TOTAL ***	

カバレッジ・ウィンドウ

c33 coverageコマンド、[カバレッジ]ボタン、または[デバッグ]ビューのコンテキストメニューから[カバレッジ]を選択することによりカバレッジ・ウィンドウが開き、計測データがあれば表示します。計測データがない場合は、エラーメッセージ"No profiling data"が表示され、[OK]ボタンをクリックするとカバレッジ・ウィンドウが開きます(データは表示されません)。また、カバレッジ・ウィンドウは、GDBが終了すると連動して終了します。

マージ結果状態：リストにマージ結果を表示しているとき、"MERGING RESULT"と表示します(それ以外の場合は空白です)。

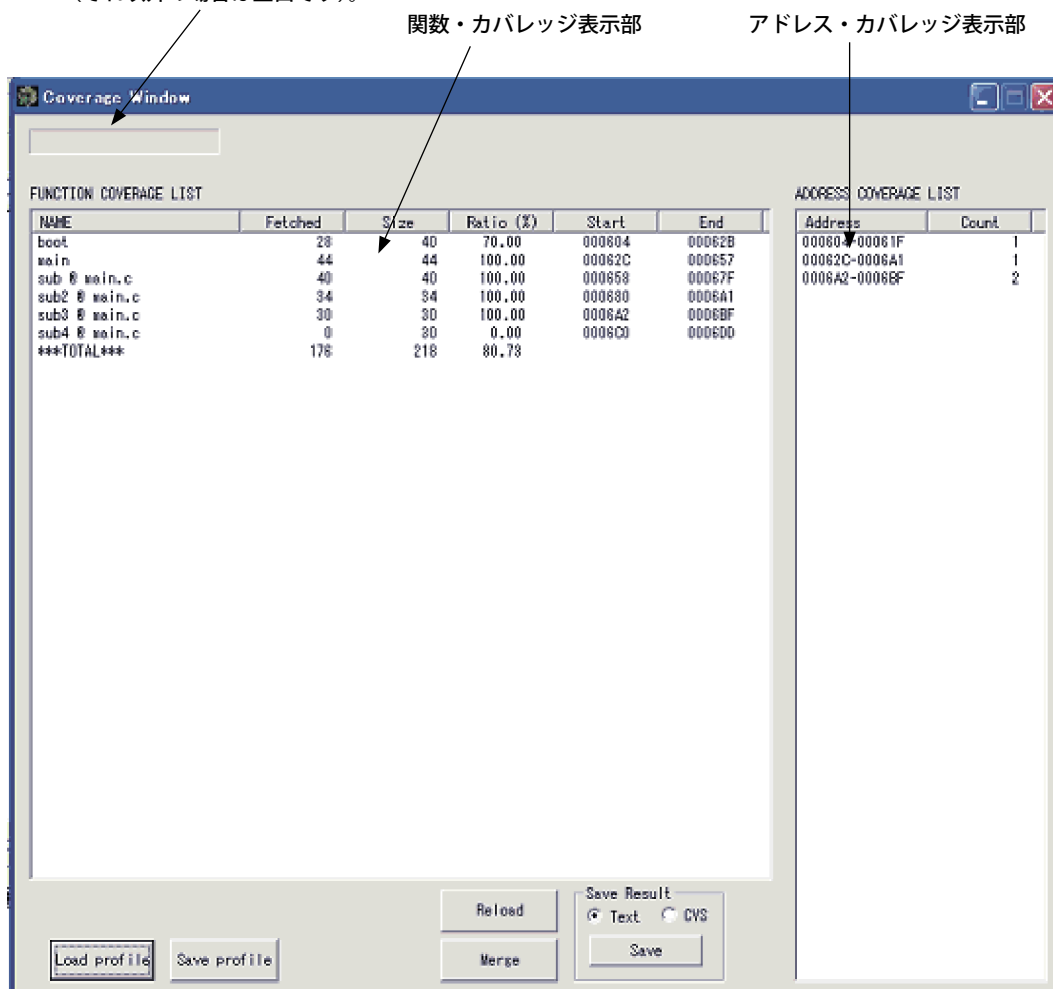


図10.6.2 カバレッジ・ウィンドウ

表示項目

関数・カバレッジ表示部

Name : 関数名(最大256文字)
 最終行の"***TOTAL***"は、トータル数を示します。
 Fetched : 関数内でフェッチしたコードサイズ(単位: バイト、10進数、最大24bit長)
 Size : 関数のサイズ(単位: バイト、10進数、最大24bit長)
 Ratio(%) : 関数内でフェッチした割合(Fetched ÷ Size、単位: %、小数点以下第2位まで表示)
 Start : 関数の開始アドレス(16進数、0 ~ FFFFFFFF)
 End : 関数の終了アドレス(16進数、0 ~ FFFFFFFF)

- ・各項目名をクリックするとソートします。項目名をクリックするたびに昇順・降順表示が切り替わります。また、デフォルトは"Start"項目で昇順に表示します。
- ・実行されなかった関数も表示します。

アドレス・カバレッジ表示部

Address : 実行したアドレス範囲(16進数、0 ~ FFFFFFFF)
 Count : 実行回数(10進数、最大31bit長)

[Load profile]ボタン

ファイルダイアログを開き、プロファイラ計測データファイルをロードします。

[Save profile]ボタン

ファイルダイアログを開き、プロファイラ計測データファイルにプロファイル情報を保存します。
 マージ後の表示状態でも、保存するデータは最初にロードしたデータとなります。

[Save Result]ボタン

ファイルダイアログを開き、表示されているデータを保存します。
 テキストファイル形式(デフォルト)、またはCSVファイル形式での保存が可能です。

[Reload]ボタン

マージ結果が表示されているときに、ウィンドウ起動後または、ロード後の表示に戻します。

[Merge] ボタン

現在表示されているカバレッジ情報に、保存されているカバレッジ情報をマージします。ファイルダイアログを開き、マージ対象のプロファイラ計測データファイルを指定します。以下のマージ結果を表示します。

(1) 関数・カバレッジ表示部

関数の開始アドレス順(デフォルト)にソートされて表示されます。

関数名 : 変更なし。

Fetched : フェッチしたアドレスが異なるとき加算されます。

Size : 変更なし。

Ratio : 再計算されます。

Start, End : 変更なし。

マージ先の関数の総数と関数名がすべて一致しない場合、エラーとなります。また、すべて一致した場合でも、各関数のStartとEndアドレスが一致しない場合はエラーになります(プログラムの変更後に取得したデータどうしはマージできません)。

(2) アドレス・カバレッジ表示部

マージされた結果のアドレス範囲とCountを更新して、アドレス順(固定)に表示されます。

Address : 追加してフェッチされたアドレス範囲を表示します。

Count : 同じアドレスのときでマージした値が大きい場合、値を更新します。

<Save Resultボタンで作成されるテキストファイルのフォーマット>

(1) "text"選択時

```
-----
Function Coverage result
Fetched      Size      Ratio(%)      Start      End      Name
    28         40         70.00      000604      00062C      boot
    44         44        100.00      00062C      000658      main
    40         40        100.00      000658      000680      sub @ main.c
    38         38        100.00      000680      0006A6      sub2 @ main.c
    30         30        100.00      0006A6      0006C4      sub3 @ main.c
   180        192         93.75
*** TOTAL ***

Address Coverage result
Address      Count
000604-00061F      1
00062C-0006A5      1
0006A6-0006C3      3
-----
```

(2) "csv"選択時

```
-----
Function Coverage result
Fetched,Size,Ratio(%),Start,End,Name
  28,  40, 70.00,0x000604,0x00062C,boot
  44,  44,100.00,0x00062C,0x000658,main
  40,  40,100.00,0x000658,0x000680,sub @ main.c
  38,  38,100.00,0x000680,0x0006A6,sub2 @ main.c
  30,  30,100.00,0x0006A6,0x0006C4,sub3 @ main.c
 180, 192, 93.75,,,*** TOTAL ***

Address Coverage result
Address,Count
0x000604-0x00061F,  1
0x00062C-0x0006A5,  1
0x0006A6-0x0006C3,  3
-----
```

●エラーメッセージ

エラーは、メッセージボックス([OK]ボタン付)で表示します。

[OK]ボタンをクリック後、計測結果は何も表示しません。

エラー表示のタイミングは、ウィンドウを開く時とプロファイラ計測データファイルの読み込みを行ったときです。

表10.6.10.3 エラーメッセージ

メッセージ	意味
No profiling data.	プロファイラ計測データファイルが見つかりません。
This file is not profiler format.	指定したプロファイラ計測データファイルのサイズが0です。
This file is not C33 architecture.	指定したファイルはC33用ではありません。IDが"C33PROF"ではありません。
not enough memory.	PCのメモリ確保に失敗しました。
Version mismatch.	プロファイラ計測データファイルのバージョンは、サポートしていません。ツールが計測データ作成時のツールのバージョンと異なっています。
Program called too many functions.	異なる10000個以上の関数をコールしているプログラムです。
Program called too deep functions.	トップの関数から見て、10000回以上深いコールをしているプログラムです。
Cycle counter was overflowed.	サイクル数カウンタが32ビットを超えています。
Function address were overlapped	重複している関数アドレスがあります。
unexpected error occured in GDB.	正しく計測が行われていません。制限事項に抵触している可能性があります(アセンブラソースtype宣言がないなど。「●制限事項」参照)
Merging mismatch (Function name)	マージ元とマージ先の関数名と関数の総数が一致していません。
Merging mismatch (Start or End address)	マージ元とマージ先の開始アドレスと終了アドレスが一致していません。

●制限事項

計測を正しく行うための制限事項を以下に示します。

- アセンブラ・ソースについて、関数名ラベルは.typeにてstab情報を付加しないと、正しいプロファイルが取れません。

例) .type sub,@function

また、ライブラリ化する場合も必要です。

Cソースは、Cコンパイラが自動的に.type文を出力するので不要です。

- プログラムの途中のブレークモード状態でset \$pc = XXXなどで、プログラム・カウンタ(%pc)を強制的に変更した場合には、正しいプロファイルは取れません。
- プログラムの途中のブレークモード状態で、"c33 profilemd 0"コマンド実施後、再度プログラムを実行させ、ブレークモード状態で:c33 profilemd 1:して再度プログラムを実行させるなど、途中で計測を中断すると正しいプロファイルは取れません。
- 計測範囲は、RESET(ブートアドレス)からブレークしたアドレスまで可能です。
ある特定のアドレスからブレークしたアドレス間のみの計測はできません。ただし、ブレーク後、%PCを変えずに再びGoしても次のブレークまで計測は行われます。
- RAM領域に動的にプログラムや関数をROM領域から転送して、実行するようなプログラムのプロファイルは取れません(.textセクションをLMA≠VMAとしてリンクしたプログラムなど)。
- 10000個以上の関数を持つプログラムで、10000個以上コールされたプロファイルは取れません。
- RESET(ブートアドレス)から見て、10000回以上深いコールをしたプログラムのプロファイルは取れません。
- デバッグ情報がない(アセンブラの-gstabsオプション、コンパイラの-gstabsオプションのない)static関数は、「関数名 + @ + ファイル名」と表示できず、ファイル名の部分は表示されません。
- until / finishコマンドでは、プロファイル・カバレッジデータは取得できません。プロファイラ・カバレッジ機能を起動すると、メッセージ"No Profiling data"が表示されます。

10.7 コマンドリファレンス

10.7.1 コマンド一覧

表10.7.1.1 コマンド一覧表

分 類	コマンド	機 能	モード対応					P No.
			ICD2	ICD3	ICD6	SIM	MON	
メモリ操作	c33 fb	領域のフィル(バイト単位)	●	●	●	●	●	10-139
	c33 fh	領域のフィル(ハーフワード単位)	●	●	●	●	●	10-139
	c33 fw	領域のフィル(ワード単位)	●	●	●	●	●	10-139
	x /b	メモリダンプ(バイト単位)	●	●	●	●	●	10-141
	x /h	メモリダンプ(ハーフワード単位)	●	●	●	●	●	10-141
	x /w	メモリダンプ(ワード単位)	●	●	●	●	●	10-141
	set {char}	データ入力(バイト単位)	●	●	●	●	●	10-143
	set {short}	データ入力(ハーフワード単位)	●	●	●	●	●	10-143
	set {int}	データ入力(ワード単位)	●	●	●	●	●	10-143
	c33 mvb	領域のコピー(バイト単位)	●	●	●	●	●	10-144
	c33 mvh	領域のコピー(ハーフワード単位)	●	●	●	●	●	10-144
	c33 mvw	領域のコピー(ワード単位)	●	●	●	●	●	10-144
	c33 df	メモリ内容の保存	●	●	●	●	●	10-146
	c33 rm	ターゲットメモリリード	●	●	●	—	—	10-148
	c33 readmd	メモリリードモード	—	●	●	—	—	10-149
レジスタ操作	info reg	レジスタ表示	●	●	●	●	●	10-150
	set \$	レジスタ変更	●	●	●	●	●	10-153
プログラム実行	continue	連続実行	●	●	●	●	●	10-154
	until	テンポラリブレーク付き連続実行	●	●	●	●	●	10-156
	step	ステップ実行(行単位)	●	●	●	●	●	10-158
	stepi	ステップ実行(ニーモニック単位)	●	●	●	●	●	10-158
	next	スキップ付きステップ実行(行単位)	●	●	●	●	●	10-160
	nexti	スキップ付きステップ実行(ニーモニック単位)	●	●	●	●	●	10-160
	finish	関数終了	●	●	●	●	●	10-162
	c33 callmd	ユーザ関数コールモード設定	●	●	●	●	●	10-163
	c33 call	ユーザ関数コール	●	●	●	●	●	10-164
	c33 rstc	コールドリセット(resetcold.gdbを実行)	●	●	●	●	●	10-166
CPUリセット	c33 rsth	ホットリセット(resethot.gdbを実行)	●	●	●	●	●	10-168
	c33 rstt	ターゲットのリセット	—	—	●	—	—	10-169
	c33 int	割り込み	—	—	—	●	—	10-170
ブレーク	break	ソフトウェアPCブレーク設定	●	●	●	●	●	10-171
	tbreak	テンポラリソフトウェアPCブレーク設定	●	●	●	●	●	10-171
	hbreak	ハードウェアPCブレーク設定	●	●	●	●	●	10-174
	thbreak	テンポラリハードウェアPCブレーク設定	●	●	●	●	●	10-174
	watch	データライトブレーク設定	●	●	●	●	●	10-177
	rwatch	データリードブレーク設定	●	●	●	●	●	10-177
	awatch	データリード/ライトブレーク設定	●	●	●	●	●	10-177
	delete	ブレーク番号によるブレークの解除	●	●	●	●	●	10-179
	clear	ブレーク位置によるブレークの解除	●	●	●	●	●	10-180
	enable	ブレークポイントの有効化	●	●	●	●	●	10-182
	disable	ブレークポイントの無効化	●	●	●	●	●	10-182
	ignore	回数指定付きブレークの無効化	●	●	●	●	●	10-184
	info breakpoints	ブレークポイントリストの表示	●	●	●	●	●	10-185
	c33 oab	オンチップエリア(CE)ブレーク設定 *1	—	●	●	—	—	10-186
	c33 obb	オンチップバスブレーク設定 *1	—	●	●	—	—	10-188
	c33 timebrk	時間経過後ブレークの設定	—	—	●	—	—	10-191
シンボル情報	info locals	ローカルシンボル表示	●	●	●	●	●	10-192
	info var	グローバルシンボル表示	●	●	●	●	●	10-192
	print	シンボル値の変更	●	●	●	●	●	10-193
ファイル読み込み	file	デバッグ情報の読み込み	●	●	●	●	●	10-194
	load	プログラムのロード	●	●	●	●	●	10-195
マップ情報	c33 rpf	マップ情報の設定	●	●	●	●	●	10-196
	c33 map	マップ情報の表示	●	●	●	●	●	10-197
フラッシュメモリ操作	c33 fls	フラッシュメモリ設定	●	●	●	—	●	10-198
	c33 fle	フラッシュメモリ消去	●	●	●	—	●	10-199
トレース	c33 tm	PCトレースモード設定	●	●	●	●	—	10-200
	c33 td	PCトレース内容の表示	●	●	●	—	—	10-206
	c33 autotd	PCトレース内容の自動表示	●	●	●	—	—	10-207
	c33 ts	PCトレース内容の検索	●	●	●	—	—	10-209
	c33 tf	PCトレース内容の保存	●	●	●	—	—	10-210
	c33 autotf	PCトレース内容の自動保存	●	●	●	—	—	10-212
	c33 obt	オンチップバストレースモード設定 *1	—	●	●	—	—	10-214
	c33 otd	オンチップバストレース内容の表示 *1	—	●	●	—	—	10-216
	c33 otff	オンチップバストレース内容の保存 *1	—	●	●	—	—	10-218
シミュレーション	c33 stdin	データ入力シミュレーション	●	●	●	●	●	10-219
	c33 stdout	データ出力シミュレーション	●	●	●	●	●	10-220

分類	コマンド	機能	モード対応					P No.
			ICD2	ICD3	ICD6	SIM	MON	
フラッシュライタ	c33 fwe	プログラム/データの消去	—	●	●	—	—	10-221
	c33 fwlp	プログラムのロード	—	●	●	—	—	10-222
	c33 fwld	データのロード	—	●	●	—	—	10-223
	c33 fwdc	ターゲットメモリのコピー	—	●	●	—	—	10-224
	c33 fwd	フラッシュライタ情報の表示	—	●	●	—	—	10-225
プロファイラ・カバレッジ	c33 profilemd	プロファイル・カバレッジモードの設定	—	—	—	●	—	10-226
	c33 profile	プロファイルウィンドウの表示	—	—	—	●	—	10-227
	c33 coverage	カバレッジウィンドウの表示	—	—	—	●	—	10-228
その他	c33 log	ロギング	●	●	●	●	●	10-229
	source	コマンドファイルの実行	●	●	●	●	●	10-230
	c33 clockmd	実行カウンタモード設定	●	●	●	●	—	10-232
	c33 clock	実行カウンタ表示	●	●	●	●	—	10-232
	target	ターゲットの接続	●	●	●	●	●	10-234
	detach	ターゲットの切断	●	●	●	●	●	10-236
	c33 das	デバッグユニットアドレス設定	—	●	●	—	—	10-237
	c33 lpt	パラレルロードモード設定/解除	●	—	—	—	—	10-238
	pwd	カレントディレクトリの表示	●	●	●	●	●	10-239
	cd	カレントディレクトリの変更	●	●	●	●	●	10-239
	c33 firmupdate	ICDファームウェア更新	—	●	●	—	—	10-240
	c33 dclk	DCLKの分周値変更	●	●	●	—	●	10-241
	c33 oscwait	OSC1モード時のウェイト設定	—	●	●	—	—	10-242
	c33 cachehit	キャッシュヒット率の表示 ^{*1}	—	●	●	—	—	10-243
	c33 logiana	ロジックアナライザモード設定	—	●	●	—	—	10-244
	c33 help	ヘルプ	●	●	●	●	●	10-245
	quit	デバッグ終了	●	●	●	●	●	10-247

モード対応: ● = 使用可(モードによらず同一機能/表示) ○ = 使用可(モードにより機能/表示が異なる) — = 使用不可

10.7.2 コマンド詳細説明の見方

コマンド個別の詳細説明は、以下の形式で記述されています。

コマンド名 (コマンド機能) [対応モード]

各コマンドの詳細説明は、この形式のコマンド名の見出しから始まります。

[対応モード]は[ICD2 / ICD3 / ICD6 / SIM / MON]のようにコマンドが使用できるモードを示します。ここに記載されていないモードでは、そのページのコマンドを使用することはできません。また[]が複数記載されている場合は、[]ごとのモードにより機能が異なります。

基本的にコマンドは個別に説明しますが、同機能で一部の違いのみにとどまる場合や、同時に説明した方が望ましいコマンドについては、複数をまとめて説明していることもあります。

機能

コマンドの機能が説明されています。

書式

[コンソール]ビューに入力する際のコマンド書式とパラメータの内容が記載されています。[]で囲まれたパラメータは省略可能です。それ以外は省略できません。*Italic*文字は数値やシンボルで指定するパラメータを表します。

使用例

コマンドの入力例とその結果などが記載されています。

注意

コマンドの制限事項や使用上の注意事項などが記載されています。

コマンドによっては、必要に応じて上記以外の項目を設けて説明しているものもあります。

10.7.3 メモリ操作コマンド

c33 fb (領域のフィル, バイト単位)

c33 fh (領域のフィル, ハーフワード単位)

c33 fw (領域のフィル, ワード単位) [ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

- c33 fb** 指定のメモリ領域を指定のバイトデータで書き換えます。
- c33 fh** 指定のメモリ領域を指定のハーフワードデータで書き換えます。
- c33 fw** 指定のメモリ領域を指定のワードデータで書き換えます。

書 式

c33 fb *StartAddr EndAddr Data*

c33 fh *StartAddr EndAddr Data*

c33 fw *StartAddr EndAddr Data*

StartAddr: フィル領域先頭アドレス(10進数、16進数、またはシンボル)

EndAddr: フィル領域終了アドレス(10進数、16進数、またはシンボル)

Data: 書き込みデータ(10進数または16進数)

条件: $0 \leq \text{StartAddr} \leq \text{EndAddr} \leq 0\text{xffffffff}$ 、 $0 \leq \text{Data} \leq 0\text{xff}$ (c33 fb)、 $0 \leq \text{Data} \leq 0\text{xffff}$ (c33 fh)、 $0 \leq \text{Data} \leq 0\text{xffffffff}$ (c33 fw)

使用例

■例1

```
(gdb)
c33 fb 0x0 0xf 0x1
Start address = 0x0, End address = 0xf, Fill data = 0x1 .....done
(gdb)
x /16b 0x0          (メモリダンプコマンド)
0x0: 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
0x8: 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01 0x01
```

アドレス0x0～0xfのメモリ領域をすべて0x01のデータ(16バイト)で書き換えます。

■例2

```
(gdb)
c33 fh 0x0 0xf 0x1
Start address = 0x0, End address = 0xe, Fill data = 0x1 .....done
(gdb)
x /8h 0x0          (メモリダンプコマンド)
0x0: 0x0001 0x0001 0x0001 0x0001 0x0001 0x0001 0x0001 0x0001
```

アドレス0x0～0xfのメモリ領域をすべて0x0001のデータ(8ハーフワード)で書き換えます(リトルエンディアンの場合)。

■例3

```
(gdb)
c33 fw 0x0 0xf 0x1
Start address = 0x0, End address = 0xc, Fill data = 0x1 .....done
(gdb)
x /4w 0x0          (メモリダンプコマンド)
0x0: 0x00000001 0x00000001 0x00000001 0x00000001
```

アドレス0x0～0xfのメモリ領域をすべて0x00000001のデータ(4ワード)で書き換えます(リトルエンディアンの場合)。

注 意

- ハーフワードおよびワード書き込みは、リトルエンディアン形式で行われます。ただし、シミュレータモードでデバッグする場合には、パラメータファイルで指定領域をビッグエンディアン形式に設定することも可能です。
- 書き込み指定範囲内のすべてあるいは一部が未使用領域であっても、エラーにはなりません。未使用領域以外はデータの書き換えが行われます。
- データ書き込み範囲はデータサイズに従った境界アドレスに調整されます。

(gdb)

c33 fw 0x3 0x9 0x0

たとえば、上記のように書き込み範囲を指定した場合、先頭アドレス0x3と終了アドレス0x9はワード境界アドレスではないため、下位2ビットを00として(ハーフワードの場合は最下位ビットを0として)境界アドレスに調整されます。実際に実行されるコマンドは次のとおりで、ワードアドレス0x0～0x8(バイトアドレス0x0～0xb)が0x00000000のデータで書き換えられます。

(gdb)

c33 fw 0x0 0x8 0x0

- アドレスのパラメータは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
- データのパラメータは、c33 fbは下位8ビット、c33 fhは下位16ビット、c33 fwは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、c33 fbでデータを0x100と指定した場合、0x00として処理されます。
- 終了アドレスが先頭アドレスより小さい場合、エラーとなります。
- 本コマンドでメモリの内容が変更されても、[メモリー]ビューや[ソース]エディターの表示内容は更新されません。それぞれのビューで表示を更新する操作を行ってください。また、プログラム領域を書き換えてもソースは書き換え前のまま表示されます。

X (メモリダンプ)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

メモリの内容を16進数でダンプ表示します。データサイズ、表示開始アドレス、表示データ数が指定できます。

書 式

x [/ [*Length*] *Size*] [*Address*]

Length: 表示するデータ数(10進数)
省略時は1です。

Size: データサイズ(表示単位)を指定する以下のシンボル

b バイト単位

h ハーフワード単位

w ワード単位(デフォルト)

Address: 表示開始アドレス(10進数、16進数、またはシンボル)

省略時は、前回の**x**コマンド実行時に表示した最後のアドレスの次になります。**gdb**起動時のデフォルトは0x0です。

条件: $0 \leq Length \leq 0xffffffff$ 、 $0 \leq Address \leq 0xffffffff$

表 示

メモリ内容は次のように表示されます。

Address[<*Symbol*>]: *Data* [*Data* ...]

Address: 各行のデータの先頭アドレスを16進数で表示します。

Symbol: 行先頭に表示されるアドレスにシンボル/ラベルが定義されている場合にそのシンボル名が表示されます。また、関数や変数の途中のアドレスを指定した場合には、そのシンボルと10進数のオフセット値(<*Symbol*+*n*>)が表示されます。

Data: *Address*から始まるデータが、1行に最大16バイト(8ハーフワード、4ワード)まで表示されます。

使用例**例1**

```
(gdb)
x
0x0:  0x00000000
```

起動後に、パラメータをすべて省略した場合、"**x** /1w 0x0"として実行されます。

例2

```
(gdb)
x /b 0
0x0 <i>:  0xe3
(gdb)
x /b 1
0x1 <i+1>: 0xa1
```

*Length*を省略して*Size*を指定すると、指定データサイズ1単位分を表示します。

*i*はアドレス0x0に定義されているシンボルです。変数などの先頭以外のアドレスを指定すると<シンボル+オフセット>がシンボルとして表示されます。

例3

```
(gdb)
x /16h _START_text
0xc00000 <_START_text>: 0x0004 0x00c0 0xc020 0x6c0f 0xa0f1 0xc000 0xc000 0x6c0f
0xc00010 <boot+12>:      0xc000 0xc000 0x1c04 0xdff8 0xdfff 0x1ef5 0x0200 0x6c04
```

*Length*を指定すると、そのデータ数分表示されます。

コード領域を表示すると、シンボルが定義されていないアドレスでも、[逆アセンブル]ビューのASSEMBLY表示と同様に<ラベル+オフセット>がシンボルとして表示されます。

■例4

```
(gdb)
x /4w 0
0x0 <i>: 0x00001ae3 0x00000000 0x00000000 0x00000000
(gdb)
x
0x10:      0x00000000
(gdb)
x
0x14:      0x00000000
```

一度xコマンドを実行した後は、xのみの入力で、前回に続くアドレスから前回と同じデータサイズ1単位分をダンプして表示します。

■例5

```
(gdb)
x /w &i
0x0 <i>: 0x00000010
(gdb)
x /w i
0x10:      0x00000000
```

アドレスをデータのシンボルで指定する場合、そのシンボルが割り付けられたアドレスを参照する場合は&を付けて入力します。シンボルのみを指定すると、そのデータの値がアドレスとして用いられます。プログラムコード中のラベルは割り付けられたアドレスを示しますので&は不要です。

注 意

- 表示は、リトルエンディアン形式で行われます。ただし、シミュレータモードでデバッグする場合には、パラメータファイルで指定領域をビッグエンディアン形式に設定することも可能です。
- メモリの未使用領域を指定した場合でも、エラーにはなりません。ただし、表示されるデータは無効です。
- データサイズに従った境界アドレス以外を指定しても、xコマンドはそのアドレスから表示します。
- アドレスのパラメータは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
- 本コマンドの実行は、[メモリー]ビューには影響を与えません。

set { } (データ入力)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

指定のアドレスに指定のデータを書き込みます。

書 式

set {Size}Address=Data

Size: データサイズを指定する以下のシンボル

char バイト単位

short ハーフワード単位

int ワード単位(デフォルト)

Address: データを書き込むアドレス(10進数、16進数、またはシンボル)

Data: 書き込むデータ(10進数、16進数、またはシンボル)

条件: $0 \leq \text{Address} \leq 0\text{xffffffff}$ 、 $0 \leq \text{Data} \leq 0\text{xff}$ (set {char})、 $0 \leq \text{Data} \leq 0\text{xffff}$ (set {short})、 $0 \leq \text{Data} \leq 0\text{xffffffff}$ (set {int})

使用例**例1**

```
(gdb)
set {char}0x1000=0x55
(gdb)
x /b 0x1000
0x1000: 0x55
```

アドレス0x1000にバイトデータ0x55を書き込みます。

例2

```
(gdb)
set {short}0x1000=0x5555
(gdb)
x /h 0x1000
0x1000: 0x5555
```

アドレス0x1000にハーフワードデータ0x5555を書き込みます。

例3

```
(gdb)
set {int}&i=0x55555555
(gdb)
x /w &i
0x0 <i>: 0x55555555
```

int変数iにワードデータ0x55555555を書き込みます。

注 意

- ハーフワードおよびワード書き込みは、リトルエンディアン形式で行われます。ただし、シミュレータモードでデバッグする場合には、パラメータファイルで指定領域をビッグエンディアン形式に設定することも可能です。
- 指定アドレスが未使用領域であっても、エラーにはなりません。
- アドレスのパラメータは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
- データのパラメータは、set {char}は下位8ビット、set {short}は下位16ビット、set {int}は下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、set {char}でデータを0x100と指定した場合、0x00として処理されます。
- 本コマンドでメモリの内容が変更されても、[メモリー]ビューや[ソース]エディターの表示内容は更新されません。それぞれのビューで表示を更新する操作を行ってください。また、プログラム領域を書き換えてもソースは書き換え前のまま表示されます。

c33 mvb (領域のコピー, バイト単位)**c33 mvh** (領域のコピー, ハーフワード単位)**c33 mvw** (領域のコピー, ワード単位)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

- c33 mvb** 指定のメモリ領域の内容をバイト単位で別の領域にコピーします。
c33 mvh 指定のメモリ領域の内容をハーフワード単位で別の領域にコピーします。
c33 mvw 指定のメモリ領域の内容をワード単位で別の領域にコピーします。

書 式**c33 mvb** *SourceStart SourceEnd Destination***c33 mvh** *SourceStart SourceEnd Destination***c33 mvw** *SourceStart SourceEnd Destination**SourceStart*: コピー元の先頭アドレス(10進数、16進数、またはシンボル)*SourceEnd*: コピー元の終了アドレス(10進数、16進数、またはシンボル)*Destination*: コピー先の先頭アドレス(10進数、16進数、またはシンボル)条件: $0 \leq \text{SourceStart} \leq \text{SourceEnd} \leq 0\text{xffffffff}$ 、 $0 \leq \text{Destination} \leq 0\text{xffffffff}$ **使用例****例1**

```
(gdb)
x /16b 0
0x0: 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x8: 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f
(gdb)
c33 mvb 0x0 0x7 0x8
Start address = 0x0, End address = 0x7, Destination address = 0x8 .....done
(gdb)
x /16b 0
0x0: 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x8: 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
```

アドレス0x0~0x7のメモリ領域の内容を、0x8以降の領域にコピーします。

例2

```
(gdb)
x /4w 0
0x0 <i>: 0x00000000 0x11111111 0x22222222 0x33333333
(gdb)
c33 mvw i i i+4
Start address = 0x0, End address = 0x0, Destination address = 0x4 .....done
(gdb)
x /4w 0
0x0 <i>: 0x00000000 0x00000000 0x22222222 0x33333333
```

int変数iの内容を1ワード(4バイト)後ろにコピーします。

注 意

- コピー元とコピー先のエンディアン形式が異なる場合は、コピーの際にデータ形式が変換されます。
- アドレスのパラメータは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
- c33 mvhとc33 mvwでは、アドレスはデータサイズに従った境界アドレスに調整されます。
c33 mvh: アドレスの最下位ビットを0として処理します。
c33 mvw: アドレスの下位2ビットを00として処理します。
- コピー元終了アドレスがコピー元先頭アドレスより小さい場合、エラーとなります。
- コピー元の指定範囲内に未使用領域がある場合、その領域のデータを0xf0としてコピーが行われます。
- コピー先の範囲内に未使用領域がある場合、その領域を除いた有効領域に対してコピーが行われます。
- コピー先アドレスがコピー元の先頭アドレスより小さい場合は、先頭アドレスから順にコピーします。逆に、コピー先アドレスがコピー元の先頭アドレスより大きい場合は、終了アドレスから順にコピーします。したがって、コピー元の領域内にコピー先アドレスを設定した場合でもコピーされます。
- コピー先の終了アドレスが0xffffffffを超える場合、0xffffffffまでがコピーの対象となります。
- 本コマンドでメモリの内容が変更されても、[メモリー]ビューや[ソース]エディターの表示内容は更新されません。それぞれのビューで表示を更新する操作を行ってください。また、プログラム領域を書き換えてもソースは書き換え前のまま表示されます。

c33 df (メモリ内容の保存)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

指定範囲のメモリの内容をバイナリ形式、テキスト形式、またはモトローラS3形式でファイルに出力します。

書 式

c33 df StartAddr EndAddr Type Filename [Append]

StartAddr: 保存領域先頭アドレス(10進数、16進数、またはシンボル)

EndAddr: 保存領域終了アドレス(10進数、16進数、またはシンボル)

Type: ファイルの種類を指定する以下の数値

- 1 バイナリファイル
- 2 テキストファイル
- 3 モトローラS3ファイル

Filename: ファイル名

Append: a 追記モード有効

Type = 1のとき、バイナリファイルの最後にダンプデータを追加出力します。

Type = 2のとき、テキストファイルの最後にダンプデータを追加出力します。

Type = 3のとき、モトローラファイルの最後にフッタレコードは付加しません。

f 追記モード有効

Type = 1のとき、バイナリファイルの最後にダンプデータを追加出力します。

Type = 2のとき、テキストファイルの最後にダンプデータを追加出力します。

Type = 3のとき、モトローラファイルの最後にフッタレコードを付加します。

省略時、ファイルは新規作成されます。

条件: $0 \leq \text{StartAddr} \leq \text{EndAddr} \leq 0\text{xffffffff}$

使用例**■例1**

(gdb)

c33 df 0x0 0xf 2 dump.txt

Start address = 0x0, End address = 0xf, File type = Text

Processing 00000000-0000000F address.

アドレス0x0～0xfの内容をファイルdump.txtにテキスト形式で書き出します。

(dump.txtの内容)

```
addr  +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00000000 00 01 02 03 04 05 06 07 00 01 02 03 04 0D 0E 0F
```

■例2

(gdb)

c33 df 0x80000 0x80103 3 dump.mot

Start address = 0x80000, End address = 0x80103, File type = Motorola-S3

Processing 00080000-0008010F address.

アドレス0x80000～0x80103の内容をファイルdump.motにモトローラS3形式で書き出します。

モトローラS3形式では1行に32バイト分が出力されます。最終行が32バイトに満たない場合、32バイトになるように終了アドレスが補正されます。この例では指定した終了アドレス0x80103が0x8010fに補正されて書き出されます。

(dump.motの内容)

```
S3250008000094D4BA020FCA086120800961881C6F0AA4D4BA020FCA086120800961881C6F0AA8
S3250008002008D3730A00000000000000000000000000000008D3730A09CA026139A505610CD309613F
:
S32500080100A4D5BA0278D5BA02A6D440004C0000000C4D6BA0238E1260A8886730A2CD3280A32
S705000000000FA
```

■例3

(gdb)

c33 df 0x10 0x1f 2 dump.txt a

アドレス0x10～0x1fの内容をファイルdump.txtの最後にテキスト形式で追加出力します。

■例4

```
(gdb)
c33 df 0x1000 0x1fff 3 dump.mot a ; フッタは出力しません。(最初)
(gdb)
c33 df 0x3000 0x3fff 3 dump.mot a ; フッタは出力しません。
(gdb)
c33 df 0x5000 0x5fff 3 dump.mot a ; フッタは出力しません。
(gdb)
c33 df 0x7000 0x7fff 3 dump.mot f ; フッタを出力します。(最後)
```

アドレス0x1000～0x7fff(0x1000番地ごと)の内容をファイルdump.motにモトローラS3形式で書き出します。

Append'パラメータがない場合、またはfを指定されるとモトローラS3形式ファイルにはフッタレコードが出力されます。

注 意

- アドレスのパラメータは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
- 終了アドレスが先頭アドレスより小さい場合、エラーとなります。

c33 rm (ターゲットメモリリード)

[ICD2 / ICD3 / ICD6]

機 能

ターゲットメモリ上のプログラムをデバッグに読み込みます。読み込んだデータはICD (S5U1C33000H/S5U1C33001H)によるPCトレース解析に使用されます。ターゲットボード上のプログラムをメモリ操作コマンドなどで書き換えた場合や、プログラムを転送し、転送先で実行したプログラムのPCトレース情報を取得するときに使用してください。

書 式

c33 rm Address1 Address2

Address1: 読み込む領域の先頭アドレス(10進数または16進数)

Address2: 読み込む領域の終了アドレス(10進数または16進数)

条件: $0 \leq \text{Address1} \leq \text{Address2} \leq 0\text{xffffffff}$

使用例

(gdb)

c33 rm 0 0x7fe

ターゲットボード上のアドレス0x0から0x7feのメモリ内容を、デバッグに読み込みます。

注 意

- c33 rmコマンドはICD2、ICD3またはICD6モード時のみ有効です。
- 本コマンドで読み込むアドレス領域は、パラメータファイルで指定されている必要があります。
- プログラムを転送後、そのまま転送先で実行した場合には正しいPCトレース情報(オペコード、逆アセンブル情報)が得られません。

転送先で実行する前に、転送先のメモリ情報(プログラムコード)をデバッグに読み込ませる必要があります。

例:

- 1) 0xc0100～0xc01fff番地の内容を内蔵RAMの0x1000番地～0x1fff番地へ転送し、0x1000番地から実行

この場合、内蔵RAMへ転送後、0x1000番地からプログラムの実行を開始する前に本コマンドを実行します。

(gdb)

c33 rm 0x1000 0x1fff

- 2) 転送から転送先での実行までのPCトレース情報を連続して取得

この場合は前準備として、転送元のプログラムを転送先へコピーし、本コマンドでその内容をデバッグに読み込んでおきます。その後、通常の手順(プログラムを実行→ブレーク→トレース取得)でトレース情報を取得します。

(gdb)

c33 mvh 0xc01000 0xc01fff 0x1000 プログラムを転送先へコピー

(gdb)

c33 rm 0x1000 0x1fff デバッグへの読み込み

(gdb)

c33 tm 1 1 トレースモードの設定 (必要な場合のみ)

(gdb)

c33 rsth

(gdb)

cont 実行

:

:

← ブレーク

(gdb)

c33 tf trace.txt PCトレースファイルの作成

c33 readmd (メモリリードモード)

[ICD3 / ICD6]

機 能

ターゲットメモリの読み込み方法を次の2種類から選択します。

1. 通常モード

メモリデータをすべてバイト単位で読み込みます。

2. 境界正確モード

ミニモニタの外部ルーチンで、

メモリをバイト(1バイト)単位でアクセスするときは1d.b命令

メモリをハーフワード(2バイト)単位でアクセスするときは1d.h命令

メモリをワード(4バイト)単位でアクセスするときは1d.w命令

を使用し、アクセス単位に合った正しい境界アドレスから読み込みます。

バイト単位での読み込みに不都合がある場合は、境界正確モードを選択してください。

書 式

c33 readmd Mode

Mode: メモリリードモード選択

0 通常モード(デフォルト)

1 境界正確モード

使用例

(gdb)

c33 readmd 1

境界正確モードに設定します。

注 意

- c33 readmdコマンドはICD3またはICD6モード時のみ有効です。
- 境界正確モードに設定した場合、メモリの読み込み速度が遅くなります。
- メモリリードモードの設定は、以下のコマンドによるメモリからの読み込みに影響します。
c33 df、x

10.7.4 レジスタ操作コマンド

info reg (レジスタ表示)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

CPUレジスタの内容を表示します。

書 式

info reg [*RegisterName*]

RegisterName: 表示するレジスタ名(小文字で指定)

r0-r15, psr, sp, alr, ahr, lco, lsa, lea, sor, ttbr, dp, idir, dbbr,
usp, ssp, pc

省略時は、すべてのレジスタの内容を表示します。

表 示

レジスタの内容は次のように表示されます。

Register Hexadecimal Decimal

Register: レジスタ名です。

Hexadecimal: レジスタ値を16進数で表示します。

Decimal: レジスタ値を10進数で表示します。

使用例

■例1

```
(gdb)
info reg r1
r1                0xffffffff -1431655766
(gdb)
info reg ttbr
ttbr              0x20000000 536870912
```

レジスタ名を指定すると、そのレジスタの内容のみを表示します。

■例2(ターゲットがC33 STDの場合)

```
(gdb)
info reg
r0                0xd20          3360
r1                0xffffffff -1431655766
r2                0xffffffff -1431655766
r3                0xffffffff -1431655766
r4                0x690          1680
r5                0xffffffff -1431655766
r6                0x0            0
r7                0xffffffff -1431655766
r8                0xffffffff -1431655766
r9                0xffffffff -1431655766
r10               0xffffffff -1431655766
r11               0xffffffff -1431655766
r12               0xffffffff -1431655766
r13               0xffffffff -1431655766
r14               0xffffffff -1431655766
r15               0x0            0
psr               0x2            2
sp                0x7f8          2040
alr               0x0            0
ahr               0x0            0
pc                0xc00030      12582960
```

■例3(ターゲットがS1C33401 (C33 ADVコア)の場合)

(gdb)

info reg

r0	0xd20	3360
r1	0xaaaaaaaa	-1431655766
r2	0xaaaaaaaa	-1431655766
r3	0xaaaaaaaa	-1431655766
r4	0x690	1680
r5	0xaaaaaaaa	-1431655766
r6	0x0	0
r7	0xaaaaaaaa	-1431655766
r8	0xaaaaaaaa	-1431655766
r9	0xaaaaaaaa	-1431655766
r10	0xaaaaaaaa	-1431655766
r11	0xaaaaaaaa	-1431655766
r12	0xaaaaaaaa	-1431655766
r13	0xaaaaaaaa	-1431655766
r14	0xaaaaaaaa	-1431655766
r15	0x0	0
psr	0x2	2
sp	0x7f8	2040
alr	0x0	0
ahr	0x0	0
lco	0x0	0
lsa	0x0	0
lea	0x0	0
sor	0x0	0
ttbr	0x20000000	536870912
dp	0x0	0
idir	0x40000000	67108864
dbbr	0x60000	393216
usp	0x0	0
ssp	0x0	0
pc	0xc00030	12582960

■例4(ターゲットがC33 PEの場合)

(gdb)

info reg

r0	0xd20	3360
r1	0xaaaaaaaa	-1431655766
r2	0xaaaaaaaa	-1431655766
r3	0xaaaaaaaa	-1431655766
r4	0x690	1680
r5	0xaaaaaaaa	-1431655766
r6	0x0	0
r7	0xaaaaaaaa	-1431655766
r8	0xaaaaaaaa	-1431655766
r9	0xaaaaaaaa	-1431655766
r10	0xaaaaaaaa	-1431655766
r11	0xaaaaaaaa	-1431655766
r12	0xaaaaaaaa	-1431655766
r13	0xaaaaaaaa	-1431655766
r14	0xaaaaaaaa	-1431655766
r15	0x0	0
psr	0x2	2
sp	0x7f8	2040
alr	0x0	0
ahr	0x0	0
ttbr	0x20000000	536870912
idir	0x60000000	1610612736
dbbr	0x60000	393216
pc	0xc00030	12582960

注 意

- デバッグ中のC33コアが持たないレジスタは表示されません。
- レジスタ名は小文字で指定してください。大文字で指定したり、存在しないレジスタ名を指定するとエラーとなります。

set \$ (レジスタ変更)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

CPUレジスタの値を変更します。

書 式

set \$RegisterName=Value

RegisterName: 変更するレジスタ名(小文字で指定)

r0-r15, psr, sp, alr, ahr, lco, lsa, lea, sor, ttbr, dp, usp, ssp, pc

Value: 設定するワードデータ(10進数、16進数、またはシンボル)

条件: $0 \leq Value \leq 0xffffffff$

使用例

```
(gdb)
set $r1=0x10000
(gdb)
info reg r1
r1                0x10000      65536
(gdb)
set $pc=main
```

設定値には数値以外にシンボルも使用可能です。

注 意

- 設定値は下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
- 設定値の内容はチェックしていません。PCに16ビット境界アドレス以外、同様にSPに32ビット境界アドレス以外の値を設定してもエラーとはなりません。ただし、実際にレジスタを変更する時点では下位ビットが切り捨てられ、強制的に境界アドレスに設定されます。
- 本コマンドを実行しても、[レジスター]ビューの表示内容は更新されません。

10.7.5 プログラム実行コマンド

continue (連続実行)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

現在のPCアドレスからターゲットプログラムを実行します。

プログラムの実行は次のいずれかの要因によってブレークするまで続きます。

- すでに設定してあるブレーク条件が成立
- [中断]ボタンのクリック(デバッグモニタモードを除く)

また、ブレーク条件の成立により停止しているターゲットプログラムを再実行させる場合、パラメータの指定により、そのブレークを指定回数だけ無効にすることができます。

書 式

continue [*IgnoreCount*]

cont [*IgnoreCount*] (省略形)

IgnoreCount: ブレーク回数指定(10進数または16進数)

設定した回数のブレーク条件が成立するまで実行を続けます。

使用例

■例1

```
(gdb)
continue
Continuing.

Breakpoint 1, main () at ./main.c:13
```

*IgnoreCount*を省略して**continue**を実行すると、ターゲットプログラムは現在のPCアドレスから実行を開始し、最初のブレーク条件成立により停止します。

■例2

```
(gdb)
cont 5
Will ignore next 4 crossings of breakpoint 1. Continuing.

Breakpoint 1, main () at ./main.c:13
```

*IgnoreCount*に5を指定しているため、実行開始後に現れる5 - 1 = 4回のブレーク条件成立を無視し、5回目の条件成立でブレークします。この例では、main.cの13行目に設定したPCブレークポイント(ブレーク番号1)で停止していたターゲットプログラムの実行を再開したため、そのPCブレークポイントに5回ヒットしたところで停止します。

これは、次のコマンドを実行した場合と同じです。

```
(gdb)
ignore 1 4
Will ignore next 4 crossings of breakpoint 1.
(gdb)
continue
```

注 意

- プログラムの先頭から実行するには、`continue`の前に`c33_rsth`(ホットリセット)を実行してください。
- *IgnoreCount*を指定した`continue`の実行は、ターゲットプログラムが一度以上実行され、ブレーク条件の成立により停止していることが条件です。なお、[中断]ボタンによるブレークはこの場合のブレーク条件の成立とは見なされません。ターゲットプログラムが一度もブレークしていない状態で*IgnoreCount*を指定しても無視されます。
- ターゲットプログラムがひとつのブレーク要因で停止しているとき、そのブレーク設定を解除してから*IgnoreCount*を指定して`continue`を実行するとエラーになります。これは、他のブレーク条件が設定されていても同じです。
- ターゲットプログラムが停止したブレーク以外のブレーク条件を指定回数分無視したいときには、`ignore`コマンドでブレーク条件とブレークヒットを無視する回数を指定し、`continue`コマンドをパラメータなしで実行してください。

until (テンポラリブレーク付き連続実行)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

現在のPCアドレスからターゲットプログラムを実行します。

テンポラリブレークを1カ所指定でき、そのブレークポイントを実行前に停止させることができます。このテンポラリブレークにはハードウェアPCブレークが使用され、一度ブレークすると解除されます。テンポラリブレークを指定した場合は、C/C++ソース以外も連続実行します。

指定したブレークポイントにヒットしない場合、プログラムの実行は次のいずれかの要因によってブレークするまで継続します。

- すでに設定してある他のブレーク条件が成立
- [中断]ボタンのクリック(デバッグモニターモードを除く)
- 現在のレベル(関数内)から上位レベルにリターンした時点
- アセンブリソースまたはソース情報がない場合は現在の命令のみを実行

書 式**until Breakpoint**

Breakpoint: テンポラリブレークポイント

以下の3種類で指定可能です。

- 関数名
- ソースファイル名:行番号、または行番号のみ
- *アドレス(10進数、16進数、またはシンボル)

条件: $0 \leq \text{アドレス} \leq 0\text{xffffffff}$

使用例**■例1**

```
(gdb)
until main
main () at ./main.c:10
```

テンポラリブレークポイントを関数名で指定し、ターゲットプログラムを実行します。

`main()`内の最初のC命令(ニーモニックに展開されるもの)を実行前にブレークします。ブレーク時のPCはその命令の先頭アドレス(展開後の先頭ニーモニックのアドレス)を示します。

■例2

```
(gdb)
until main.c:10
main () at ./main.c:10
```

テンポラリブレークポイントを行番号で指定し、ターゲットプログラムを実行します。

ここでは、"ソースファイル名:行番号"の書式で指定していますが、現在のPCアドレスがあるCソース内であれば**until 10**のように行番号のみで指定できます。アセンブリソースの場合は常にソースファイル名が必要です。

このコマンドの実行により、`main.c`内の10行目のC命令を実行前にブレークします。ブレーク時のPCはその命令の先頭アドレス(展開後の先頭ニーモニックのアドレス)を示します。10行目に実コードを持つ命令がない(ニーモニックに展開されない)場合は、それ以降最初に現れる実コードを持つ命令の先頭でブレークします。

■例3

```
(gdb)
until *0xc0001e
main () at ./main.c:10
```

テンポラリブレークポイントをアドレスで指定し、ターゲットプログラムを実行します。

そのアドレスの命令を実行前にブレークします。

次のようにシンボルも使用可能です。

```
(gdb)
until *main
main () at ./main.c:7
```

*を付加すると、関数名でもアドレスと見なされます。

■注 意

- プログラムの先頭から実行するには、`until`の前に`c33 rsth`(ホトリセット)を実行してください。
- ニーモニックに展開されないC/C++ソース行をテンポラリブレークポイントとして設定しても、その行ではブレークしません。それ以降の最初に実行されるニーモニックのアドレスでブレークします。
- テンポラリブレークポイントは、以下の行には設定できません。エラーとなります。
 - 先頭の`ext`命令を除く拡張命令の行
 - 遅延命令の行(遅延分岐命令の次の行)
- テンポラリブレークポイントを、存在しない関数名や行番号で指定するとエラーとなります。
- テンポラリブレークポイントをアドレス値で指定する場合、そのパラメータは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、`0x100000000`は`0x00000000`として処理されます。
- テンポラリブレークポイントをアドレス値で指定する場合、アドレスを奇数の値で指定すると最下位ビットを0として16ビット境界に補正されます。

step (ステップ実行、行単位)**stepi** (ステップ実行、ニーモニック単位)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

現在のPCからターゲットプログラムをシングルステップ実行します。呼び出される関数やサブルーチン内もステップ実行します。

step: ソースの行単位にシングルステップ実行します。C/C++ソースは1行のC/C++命令(展開された複数のニーモニックすべて)を1ステップとして実行します。アセンブリソースの場合は**stepi**と同機能です。

stepi: ニーモニック単位にシングルステップ実行します。

実行するステップ数を指定できます。

その全ステップが終了する前でも、プログラムの実行は次のいずれかの要因によって停止します。

- すでに設定してあるブレーク条件が成立
- [中断]ボタンのクリック(デバッグモニタモードを除く)

書 式

step [*Count*]

stepi [*Count*]

Count: 実行ステップ数(10進数、または16進数)
省略時は1ステップです。

条件: $1 \leq Count \leq 0x7fffffff$

使用例**■例1**

```
(gdb)
step
```

現在のPCが示すソース行を実行します。

■例2

```
(gdb)
stepi
```

現在のPCが示すアドレスの命令(ニーモニック単位)を実行します。

■例3

```
(gdb)
step 10
sub (k=5) at ./main.c:20
```

現在のPCが示すソース行から10行分を実行します。

■例4

```
(gdb)
stepi 10
main () at ./main.c:13
```

現在のPCが示すアドレスから10命令(ニーモニック単位)を実行します。

注 意

- ソース情報(オブジェクトファイルに含まれるデバッグ情報)のないアドレスからステップ実行することはできません。continueなどにより連続実行することは可能です。
- プログラムの先頭から実行するには、step/stepiの前にc33 rsth(ホットリセット)を実行してください。
- stepiであっても、extによる拡張命令は、拡張命令全体(2または3命令)を1ステップとして実行します。
- シングルステップ動作時も割り込みを受け付けます。
また、halt命令およびslp命令はシングルステップ動作時でも実行され、それによってスタンバイモードになります。これは、外部割り込みを発生させることによって解除されます。また、[中断]ボタンで解除することもできます。
ただし、シミュレータモードでは割り込み、halt命令およびslp命令によるスタンバイモードはサポートしていません。

next (スキップ付きステップ実行、行単位)**nexti** (スキップ付きステップ実行、ニーモニック単位) [ICD2 / ICD3 / ICD6 / SIM / MON]**機 能**

現在のPCからターゲットプログラムをシングルステップ実行します。基本的な動作はstep/stepiと同じですが、関数コールやサブルーチンコールは、それより下位レベルの呼び出される関数やサブルーチンをすべて含め1ステップとして実行します(呼び出される関数やサブルーチン内はリターンするまで連続実行されます)。

next: ソースの行単位にシングルステップ実行します。C/C++ソースは1行のC/C++命令(展開された複数のニーモニックすべて)を1ステップとして実行します。アセンブリソースの場合はnextiと同機能です。

nexti: ニーモニック単位にシングルステップ実行します。

実行するステップ数を指定できます。

その全ステップが終了する前でも、プログラムの実行は次のいずれかの要因によって停止します。

- すでに設定してあるブレーク条件が成立
- [中断]ボタンのクリック(デバッグモニタモードを除く)

書 式

next [*Count*]

nexti [*Count*]

Count: 実行ステップ数(10進数、または16進数)
省略時は1ステップです。

条件: $1 \leq Count \leq 0x7fffffff$

使用例**■例1**

```
(gdb)
next
```

現在のPCが示すソース行を実行します。ソースが関数コールまたはサブルーチンコールの場合は、呼び出す関数/サブルーチンもリターンするまで実行します。

■例2

```
(gdb)
nexti
```

現在のPCが示すアドレスの命令(ニーモニック単位)を実行します。命令がサブルーチンコールの場合は、呼び出すサブルーチンもリターンするまで実行します。

■例3

```
(gdb)
next 10
sub (k=5) at ./main.c:20
```

現在のPCが示すソース行から10行分を実行します。

■例4

```
(gdb)
nexti 10
main () at ./main.c:13
```

現在のPCが示すアドレスから10命令(ニーモニック単位)を実行します。

注 意

- ソース情報(オブジェクトファイルに含まれるデバッグ情報)のないアドレスからステップ実行することはできません。continueなどにより連続実行することは可能です。
- プログラムの先頭から実行するには、next/nextiの前にc33 rsth(ホトリセット)を実行してください。
- nextiであっても、extによる拡張命令は、拡張命令全体(2または3命令)を1ステップとして実行します。
- シングルステップ動作時も割り込みを受け付けます。
また、halt命令およびslp命令はシングルステップ動作時でも実行され、それによってスタンバイモードになります。これは、外部割り込みを発生させることによって解除されます。また、[変更]ボタンで解除することもできます。
ただし、シミュレータモードでは割り込み、halt命令およびslp命令によるスタンバイモードはサポートしていません。

finish (関数終了)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

現在のPCからターゲットプログラムを実行し、現在の関数から上位レベルにリターンした時点で停止します。リターン位置の命令は実行しません。

ただし、リターンの前でも、プログラムの実行は次のいずれかの要因によって停止します。

- すでに設定してあるブレイク条件が成立
- [中断]ボタンのクリック(デバッグモニタモードを除く)

書 式**finish****使用例**

(gdb)

finish

現在のPCアドレスからターゲットプログラムを実行し、リターン後に停止します。

注 意

ブートルーチンのような最上位レベルで**finish**を実行した場合、プログラムは停止しません。ブレークが設定されていない場合は、[中断]ボタンで停止させてください。

c33 callmd (ユーザ関数コールモード設定) [ICD2 / ICD3 / ICD6 / SIM / MON]**機 能**

c33 call(ユーザ関数コール)コマンドを実行した際の実行結果の出力先を設定します。

書 式

c33 callmd Mode [Filename]

Mode: 結果出力先を指定する以下の数値

- 1 [コンソール]ビューに表示(デフォルト)
- 2 ファイルに書き込み
- 3 [コンソール]ビューに表示、およびファイルに書き込み

Filename: 出力ファイル名(*Mode*が1の場合は無効)

使用例

(gdb)

c33 callmd 2 call.txt

これから実行するc33 callコマンドの実行結果をファイルcall.txtに書き込みます。

注 意

出力先にファイルを選択した場合、ファイルはc33 callコマンド実行時にカレントディレクトリに作成され、c33 callコマンド実行終了時に結果が書き込まれて閉じます。

既存のファイル名を指定した場合、新たな実行結果でファイルは上書きされます。

c33 call (ユーザ関数コール)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

ユーザ関数をコールします。
ただし、アセンブラのエントリプログラムが必要です。

書 式

c33 call Function [Arg1 [Arg2 [Arg3]]]

Function: コールする関数(関数名、または10進数あるいは16進数の先頭アドレス)

Arg1-3: 引数(10進数、16進数、またはシンボル)

条件: 引数は0個から3個まで指定可能です。

入力例

```
(gdb)
c33 callmd 1
(gdb)
c33 call print_data 1 2 3
arg1=1, arg2=2, arg3=3      (実行結果)
```

引数を3個指定して関数`print_data()`をコールします。

ユーザ関数とエントリルーチン

`c33 call`により**gdb**は指定のユーザ関数を実行し、`%r10`から戻り値を受け取ります。戻り値が-1 (0xffffffff)のときは何もせずに終了します。それ以外の値は関数から渡されるパケットの先頭アドレスと解釈し、パケット内のデータを[コンソール]ビューにテキストとして表示、またはそのままの形でファイルに書き出します。出力先は**c33 callmd**コマンドで指定します。デフォルトの出力先は[コンソール]ビューです。

ユーザ関数が返すパケットの構造は次のとおりです。

データサイズ(4バイト) データ....

パケットは必ずワード境界(4バイト境界)から始まらなければなりません。また、ユーザ関数は、このパケットの先頭アドレスを`%r10`に設定してください。パケットを返せない場合は`%r10`に-1を設定します。

コールされるユーザ関数の例を以下に示します。

ユーザ関数の最後は、必ず"`jp %r15`"で終了していなければなりません。

エントリプログラム例(アセンブラ)

```
#define SP_INI 0x0800          ; sp is in end of 1KB internal RAM
#define DP_INI 0x0000          ; default data area pointer dp (%r15) is 0x0

        .text
.global print_data
print_data:
        ld.w    %r5,%sp        ; save SP
        xld.w   %r4,SP_INI     ; set SP
        ld.w    %sp,%r4        ; save return address
        ld.w    %r4,%r15
        pushn   %r8            ; save registers
        ld.w    %r6,%r12
        ld.w    %r7,%r13
        ld.w    %r8,%r14
        xld.w   %r15,DP_INI    ; set default data area pointer for safety
        xcall   iprint_data    ; enter C program
        ld.w    %r10,%r4
        popn    %r8            ; restore registers
        ld.w    %sp,%r5        ; restore SP
        ld.w    %r15,%r4       ; restore return address
        jp      %r15           ; back to mini monitor
```


ユーザ関数

```

struct {
    int size;
    char buf[0x100];
} tmpbuf;

int *iprint_data(int arg1, int arg2, int arg3)
{
    int cnt;
    cnt = sprintf(tmpbuf.buf, "arg1=%d, arg2=%d, arg3=%d", arg1, arg2, arg3);
    tmpbuf.size = cnt+1;          // +1 is null
    return (int*)&tmpbuf;
}

```

注 意

- エントリプログラムがない関数をコールすると暴走します。
- `c33 call`コマンドを実行する前に、呼び出される関数内のブレーク設定はすべて削除するか無効にしてください。ブレークが設定されていると正常に動作しません。

10.7.6 CPUリセットコマンド

c33 rstc (コールドリセット)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

CPUをコールドリセットします。
 これによる初期設定内容は以下のとおりです。

(1) CPUの内部レジスタ

R0～R15:	0xaaaaaaaa	
PC:	ブートアドレス	(0xc00000の内容が示すアドレス)
SP:	0x0aaaaaaaa8	(C33 STD)
	0x00000000	(C33 ADV、C33 PE)
PSR:	0x00000000	
AHR, ALR:	0xaaaaaaaa	
LCO:	0x00000000	(C33 ADV)
LSA:	0x00000000	(C33 ADV)
LEA:	0x00000000	(C33 ADV)
SOR:	0x00000000	(C33 ADV)
TTBR:	0x00C00000	(C33 PE)
	0x20000000	(C33 ADV)
DP:	0x00000000	(C33 ADV)
IDIR:	0x04000000	(C33 ADV)
	0x06000000	(C33 PE)
DBBR:	0x00060000	(C33 ADV、C33 PE)
USP:	0x00000000	(C33 ADV)
SSP:	0x00000000	(C33 ADV)

特に指定のないレジスタはC33コアすべてに共通です。

(2) 実行カウンタを0に設定

(3) [ソース]エディター、[レジスター]ビューを再表示

PCがブートアドレスに設定されるため、[ソース]エディターはプログラムをそのアドレスから再表示します。

[レジスター]ビューを上記の設定で再表示します。

書 式

c33 rstc

使用例

```
(gdb)
c33 rstc
```

CPUをコールドリセットします。

注 意

- メモリ内容、ブレークやトレースなどのデバッグステータスはリセットされません。
- シミュレータモード以外で使用する場合、バスやI/Oの初期化については各機種の"テクニカルマニュアル"を参照してください。
- ICD2/ICD3/ICD6モードでは、上記の処理を行い、S1C33チップもリセットします。ターゲットボードはリセットされません。また、c33 rstcコマンド実行時にターゲットがフリーランしている場合は、強制ブレークをかけてからリセットを行います。S5U1C33000H/S5U1C33001Hに接続されたターゲットをリセットするとターゲットシステムはフリーラン状態となりますが、c33 rstcコマンドでこれを停止させることができます。
- シミュレータモードかつC33 ADVコアデバッグ時で、パラメータファイルによりアドレス0x20000000から1バイト以上のマップ設定がされていない場合、PCの初期設定内容は0xc00000の内容が示すアドレスになります。

c33 rsth (ホットリセット)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

CPUをホットリセットします。

PC、TTBR以外のレジスタと実行カウンタの初期設定、ビューの再表示については**c33 rstc**コマンドと同様です。PC値(ブートアドレス)はTTBRレジスタにより指定されます。TTBRの値は変更されません。

書 式

c33 rsth

使用例

(gdb)

c33 rsth

CPUをホットリセットします。

注 意

- メモリ内容、ブレークやトレースなどのデバッグステータスはリセットされません。
- シミュレータモード以外で使用する場合、バスやI/Oの状態は保持されます。
- シミュレータモード時に以下の条件が成立している場合には、0xc00000の内容が示すアドレスがPCに設定されます。
 - シミュレータモード
 - C33 ADVモード
 - TTBRレジスタ = 0x20000000
 - 読み込まれているパラメータファイルで0x20000000番地から1バイト以上ROMまたはRAMにマップ設定されている

c33 rstt (ターゲットリセット)

[ICD6]

機 能

S5U1C33001H1400(ICD Ver.6)からターゲットのリセット入力端子へリセット信号を出力します。

書 式

```
c33 rstt
```

使用例

```
(gdb)
c33 rstt
TARGET resetting ..... done
```

ターゲットをリセットします。

注 意

- c33 rsttコマンドはICD6モード時のみ使用可能です。
- このコマンドを実行するには、ターゲットボードにリセット入力端子を実装する必要があります。
- ターゲットのリセットに失敗した場合は次のメッセージが表示されます。

```
TARGET resetting ..... failure
```

10.7.7 割り込みコマンド

c33 int (割り込み)

[SIM]

機能

割り込みの発生をシミュレートします。

本コマンドで割り込み番号を指定すると、次のプログラム実行開始時にその割り込みが発生します。

書式

c33 int [*Type Level*]

Type: 割り込みタイプ(10進数、16進数、またはシンボル)

Level: 割り込みレベル(10進数、16進数、またはシンボル)

条件: $0 \leq Type \leq 0x7fff$, $0 \leq Level \leq 15$

使用例

■例1

```
(gdb)
c33 int
```

パラメータを指定しない場合、NMIが発生します。

■例2

```
(gdb)
c33 int 3 6
```

マスク可能な割り込みの番号と割り込みレベルが設定できます。

注意

- c33 int コマンドはシミュレータモード時のみ使用可能です。
- 割り込みタイプは0～0x7fffの範囲内で指定してください。これを超えた場合、エラーとなります。
- 割り込みレベルは0～15の範囲内で指定してください。これを超えた場合、エラーとなります。
- シミュレータモードでもブートベクタアドレス(TTBR)の設定は有効です。

10.7.8 ブレーク設定コマンド

break (ソフトウェア PC ブレーク設定)

tbreak (テンポラリソフトウェア PC ブレーク設定) [ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

ソフトウェアPCブレークポイントを設定します。最大200カ所のブレークポイントが設定可能です。プログラムの実行によりPCが設定したアドレスに一致すると、そのアドレスの命令を実行前にブレークします。ブレークポイントは関数名、行番号、アドレスで指定できます。

breakコマンドとtbreakコマンドのブレーク機能はどちらも同じで、違いは次のとおりです。

break: breakで設定したブレークポイントは、プログラムの実行によるブレークの発生では解除されません。

tbreak: tbreakで設定したブレークポイントは、一度ブレークすると解除されます。

書 式

break [*Breakpoint*]

tbreak [*Breakpoint*]

Breakpoint: ブレークポイント

以下の3種類で指定可能です。

- 関数名
 - ソースファイル名:行番号、または行番号のみ
 - *アドレス(10進数、16進数、またはシンボル)
- 省略時は、現在のPCが示すアドレスに設定されます。

条件: $0 \leq \text{アドレス} \leq 0\text{xffffffff}$

使用例

■例1

```
(gdb)
break main
Breakpoint 1 at 0xc0001e: file ./main.c, line 10.
(gdb)
continue
Continuing.

Breakpoint 1, main () at ./main.c:10
```

関数名を指定し、ソフトウェアPCブレークポイントを設定します。

ターゲットプログラムを実行すると、main ()内の最初のC命令(ニーモニックに展開されるもの)を実行前にブレークします。ブレーク時のPCはその命令の先頭アドレス(展開後の先頭ニーモニックのアドレス)を示します。

■例2

```
(gdb)
tbreak main.c:10
Breakpoint 1 at 0xc0001e: file ./main.c, line 10.
```

行番号を指定し、テンポラリソフトウェアPCブレークポイントを設定します。

ここでは、"ソースファイル名:行番号"の書式で指定していますが、現在のPCアドレスがあるCソース内であればtbreak 10のように行番号のみで指定できます。アセンブリソースの場合は常にソースファイル名が必要です。

指定の行に実コードを持つ命令がない(ニーモニックに展開されない)場合は、それ以降最初に現れる実コードを持つ命令の先頭にブレークポイントが設定されます。

ターゲットプログラムを実行すると、main.cの10行目のC命令を実行前にブレークします。ブレーク時のPCはその命令の先頭アドレス(展開後の先頭ニーモニックのアドレス)を示します。10行目に実コードを持つ命令がない場合は、それ以降最初に現れる実コードを持つ命令の先頭でブレークします。tbreakで設定したため、ブレークポイントはブレーク後に解除されます。

■例3

```
(gdb)
break *0xc0001e
Note: breakpoint 1 also set at pc 0xc0001e.
Breakpoint 2 at 0xc0001e: file ./main.c, line 10.
```

アドレスを指定し、ソフトウェアPCブレークポイントを設定します。
ターゲットプログラムを実行すると、そのアドレスの命令を実行前にブレークします。
次のようにシンボルも使用可能です。

```
(gdb)
tbreak *main
Breakpoint 3 at 0xc0001c: file ./main.c, line 7.
```

*を付加すると、関数名でもアドレスと見なされます。

ブレークポイントの管理方法

ブレークポイントを設定すると、ブレークの種類にかかわらず、1から順にブレーク番号が付けられ、ブレーク設定コマンド実行時に[コンソール]ビューにメッセージとして表示されます(上記例参照)。この番号は、後からブレークポイントを個別に無効化/有効化、あるいは削除するときに必要となります。なお、ブレークポイントを削除しても、デバッグを終了するまで、番号の繰り上げ(消えた番号の再利用)は起きません。

設定したブレークポイントを操作するには、以下のコマンドを使用します。

disable	ブレークポイントの無効化(ブレーク設定時は有効です)
enable	ブレークポイントの有効化
deleteまたはclear	ブレークポイントの削除
ignore	ブレークを無効にする回数を指定
info breakpoints	ブレークポイントリストの表示

詳細は、各コマンドの説明を参照してください。

注 意

- テンポラリブレークも含め、ソフトウェアPCブレークポイントは、最大200カ所まで設定可能です。これを超えるとエラーになります。
デバッグが他の機能で使用するソフトウェアPCブレークについてもこのブレーク本数に含まれますので注意してください。
- ニーモニックに展開されないC/C++ソース行をソフトウェアPCブレークポイントに指定しても、その行はブレークポイントとして設定されません。ソフトウェアPCブレークポイントは、それ以降の最初に実行される命令のアドレスに設定されます。
- 関数名や関数内先頭のC/C++ソース行をソフトウェアPCブレークポイントに設定した場合、プログラムの実行がブレークするのは、関数内の最初のC/C++ソース(ニーモニックに展開される命令)の先頭アドレスです。関数の先頭にはコンパイル時にレジスタを退避させるpush命令が挿入されますが、その命令はブレーク前に実行されます。この命令の実行前にブレークさせるには、そのアドレス値でソフトウェアPCブレークポイントを指定してください。
- ソフトウェアPCブレークポイントは、以下の行には設定できません。
 - 先頭のext命令を除く拡張命令の行
 - 遅延命令の行(遅延分岐命令の次の行)
- 存在しない関数名や行番号でソフトウェアPCブレークポイントを指定するとエラーとなります。
- ソフトウェアPCブレークポイントをアドレス値で指定する場合、そのパラメータは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
- ソフトウェアPCブレークポイントをアドレス値で指定する場合、アドレスを奇数の値で指定すると最下位ビットを0として16ビット境界に補正されます。

- ソフトウェアPCブレークは、`brk`命令の埋め込みにより実現しています。そのため、命令の埋め込みができないターゲットボード上のROMに対しては使用できません。この場合は、ハードウェアPCブレークを使用してください。
- シミュレータモード以外の時、パラメータファイル(*.par)で"ROM"属性にした領域には、ソフトウェアPCブレークおよびテンポラリソフトウェアPCブレークは設定できません。

hbreak (ハードウェア PC ブレーク設定)**thbreak** (テンポラリハードウェア PC ブレーク設定) [ICD2 / ICD3 / ICD6 / SIM / MON]**機 能**

ハードウェアPCブレークポイントを設定します。最大2カ所のブレークポイントが設定可能です。プログラムの実行によりPCが設定したアドレスに一致すると、そのアドレスの命令を実行前にブレークします。ブレークポイントは関数名、行番号、アドレスで指定できます。

hbreakコマンドとthbreakコマンドのブレーク機能はどちらも同じで、違いは次のとおりです。

hbreak: hbreakで設定したブレークポイントは、プログラムの実行によるブレークの発生では解除されません。

thbreak: thbreakで設定したブレークポイントは、一度ブレークすると解除されます。

書 式

hbreak [*Breakpoint*]

thbreak [*Breakpoint*]

Breakpoint: ブレークポイント

以下の3種類で指定可能です。

- 関数名
 - ソースファイル名:行番号、または行番号のみ
 - *アドレス(10進数、16進数、またはシンボル)
- 省略時は、現在のPCが示すアドレスに設定されます。

条件: $0 \leq \text{アドレス} \leq 0\text{xffffffff}$

使用例**例1**

```
(gdb)
hbreak main
Hardware assisted breakpoint 1 at 0xc0001e: file ./main.c, line 10.
(gdb)
continue
Continuing.

Breakpoint 1, main () at ./main.c:10
```

関数名を指定し、ハードウェアPCブレークポイントを設定します。

ターゲットプログラムを実行すると、main()内の最初のC命令(ニーモニックに展開されるもの)を実行前にブレークします。ブレーク時のPCはその命令の先頭アドレス(展開後の先頭ニーモニックのアドレス)を示します。

例2

```
(gdb)
thbreak main.c:10
Hardware assisted breakpoint 1 at 0xc0001e: file ./main.c, line 10.
```

行番号を指定し、テンポラリハードウェアPCブレークポイントを設定します。

ここでは、"ソースファイル名:行番号"の書式で指定していますが、現在のPCアドレスがあるCソース内であればthbreak 10のように行番号のみで指定できます。アセンブリソースの場合は常にソースファイル名が必要です。

指定の行目実コードを持つ命令がない(ニーモニックに展開されない)場合は、それ以降最初に現れる実コードを持つ命令の先頭にブレークポイントが設定されます。

ターゲットプログラムを実行すると、main.c内の10行目のC命令を実行前にブレークします。ブレーク時のPCはその命令の先頭アドレス(展開後の先頭ニーモニックのアドレス)を示します。10行目に実コードを持つ命令がない場合は、それ以降最初に現れる実コードを持つ命令の先頭でブレークします。thbreakで設定したため、ブレークポイントはブレーク後に解除されます。

■例3

```
(gdb)
hbreak *0xc0001e
Note: breakpoint 1 also set at pc 0xc0001e.
Hardware assisted breakpoint 2 at 0xc0001e: file ./main.c, line 10.
```

アドレスを指定し、ハードウェアPCブレークポイントを設定します。
 ターゲットプログラムを実行すると、そのアドレスの命令を実行前にブレークします。
 次のようにシンボルも使用可能です。

```
(gdb)
thbreak *main
Hardware assisted breakpoint 3 at 0xc0001c: file ./main.c, line 7.
```

*を付加すると、関数名でもアドレスと見なされます。

ブレークポイントの管理方法

ブレークポイントを設定すると、ブレークの種類にかかわらず、1から順にブレーク番号が付けられ、ブレーク設定コマンド実行時に[コンソール]ビューにメッセージとして表示されます(上記例参照)。この番号は、後からブレークポイントを個別に無効化/有効化、あるいは削除するときにより必要となります。なお、ブレークポイントを削除しても、デバッグを終了するまで、番号の繰り上げ(消えた番号の再利用)は起きません。

設定したブレークポイントを操作するには、以下のコマンドを使用します。

disable	ブレークポイントの無効化(ブレーク設定時は有効です)
enable	ブレークポイントの有効化
deleteまたはclear	ブレークポイントの削除
ignore	ブレークを無効にする回数を指定
info breakpoints	ブレークポイントリストの表示

詳細は、各コマンドの説明を参照してください。

注 意

- ・ テンポラリブレークも含め、ハードウェアPCブレークポイントは、最大2カ所まで設定可能です。これを超えるとエラーになります。デバッグがテンポラリブレーク付き連続実行で使用するハードウェアPCブレークについてもこのブレーク本数に含まれますので注意してください。
- ・ ICD2/ICD3/ICD6モードの範囲トレースはハードウェアPCブレーク機能を使用します。範囲トレースモードを設定すると、ハードウェアPCブレーク機能は無効となります。ただし、ハードウェアPCブレークアドレスの設定はクリアされず、範囲トレースモードを解除(全体トレースモードを設定)した時点で有効になります。
- ・ テンポラリハードウェアPCブレークポイントを設定しても、[ブレークポイント]ビューや[ソース]エディターには表示されません。
- ・ ニーモニックに展開されないC/C++ソース行をハードウェアPCブレークポイントに指定しても、その行はブレークポイントとして設定されません。ハードウェアPCブレークポイントは、それ以降の最初に実行される命令のアドレスに設定されます。
- ・ 関数名や関数内先頭のC/C++ソース行をハードウェアPCブレークポイントに設定した場合、プログラムの実行がブレークするのは、関数内の最初のC/C++ソース(ニーモニックに展開される命令)の先頭アドレスです。関数の先頭にはコンパイル時にレジスタを退避させるpush命令が挿入されますが、その命令はブレーク前に実行されます。この命令の実行前にブレークさせるには、そのアドレス値でブレークポイントを指定してください。
- ・ ハードウェアPCブレークポイントは、以下の行には設定できません。設定すると、ターゲットプログラムが実行できなくなります(解除すれば問題ありません)。
 - 先頭のext命令を除く拡張命令の行
 - 遅延命令の行(遅延分岐命令の次の行)
- ・ 存在しない関数名や行番号でハードウェアPCブレークポイントを指定するとエラーとなります。

10 デバッグ

- ハードウェアPCブレイクポイントをアドレス値で指定する場合、そのパラメータは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
- ハードウェアPCブレイクポイントをアドレス値で指定する場合、アドレスを奇数の値で指定すると最下位ビットを0として16ビット境界に補正されます。

watch (データライトブレーク設定)**rwatch** (データリードブレーク設定)**awatch** (データリード / ライトブレーク設定) [ICD2 / ICD3 / ICD6 / SIM / MON]**機 能**

データブレークポイントを設定します。設定可能なブレークポイントは1カ所です。

データブレークを設定すると、ターゲットプログラムが実行中に指定のアドレスをアクセスすると、アクセス直後にブレークします。ブレークポイントはシンボルまたはアドレスで指定できます。

3つのコマンドは、それぞれブレークを発生させるアクセス内容が異なります。

watch: watchでデータブレークを設定すると、ターゲットプログラムが指定のアドレスにデータを書き込むとブレークします。

rwatch: rwatchでデータブレークを設定すると、ターゲットプログラムが指定のアドレスからデータを読み出すとブレークします。

awatch: awatchでデータブレークを設定すると、ターゲットプログラムが指定のアドレスに対してデータの読み出しまたは書き込みを行うとブレークします。

書 式**watch Breakpoint****rwatch Breakpoint****awatch Breakpoint**

Breakpoint: ブレークポイント

以下の2種類で指定可能です。

- シンボル

条件: long long型、double型、構造体・共用体名、配列名以外であること。ただし、構造体・共用体のメンバ、配列のインデックス付の場合は除きます。

- *アドレス(10進数、16進数、またはシンボル)

条件: $0 \leq \text{アドレス} \leq 0\text{xffffffff}$

使用例**例1**

```
(gdb)
watch i
Hardware watchpoint 1: i
(gdb)
continue
Continuing.
Hardware watchpoint 1: i

Old value = 0
New value = 1
sub (k=1) at ./main.c:24
```

変数iのアドレスにデータライトブレークポイントを設定します。

プログラムを実行し、変数iにデータが書き込まれるとブレークします。

例2

```
(gdb)
rwatch i
Hardware read watchpoint 2: i
(gdb)
continue
Continuing.
Hardware read watchpoint 2: i

Value = 2
0x00c00042 in sub (k=1) at ./main.c:22
```

変数iのアドレスにデータリードブレークポイントを設定します。

プログラムを実行し、変数iが読み出されるとブレークします。

■例3

```
(gdb)
awatch *0x0
Hardware access (read/write) watchpoint 3: *0
(gdb)
continue
Continuing.
Hardware access (read/write) watchpoint 3: *0

Value = 2
sub (k=1) at ./main.c:24
```

アドレス0x0にデータリード/ライトブレークポイントを設定します。

プログラムを実行し、アドレス0x0のデータが読み出されるか、データが書き込まれるとブレークします。

ブレークポイントの管理方法

ブレークポイントを設定すると、ブレークの種類にかかわらず、1から順にブレーク番号が付けられ、ブレーク設定コマンド実行時に[コンソール]ビューにメッセージとして表示されます(上記例参照)。この番号は、後からブレークポイントを個別に無効化/有効化、あるいは削除するときに必要となります。なお、ブレークポイントを削除しても、デバッグを終了するまで、番号の繰り上げ(消えた番号の再利用)は起きません。

設定したブレークポイントを操作するには、以下のコマンドを使用します。

disable	ブレークポイントの無効化(ブレーク設定時は有効です)
enable	ブレークポイントの有効化
deleteまたはclear	ブレークポイントの削除
ignore	ブレークを無効にする回数を指定
info breakpoints	ブレークポイントリストの表示

詳細は、各コマンドの説明を参照してください。

注 意

- データブレークポイントは、1カ所のみ設定可能です。これを超えるとエラーになります。
- データブレークポイントを設定しても、[ブレークポイント]ビューやその他のビューには表示されません。
- 存在しないシンボルでデータブレークポイントを指定するとエラーとなります。
- データブレークポイントをアドレス値で指定する場合、そのパラメータは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
- データブレークポイントには、ターゲットプログラムがアクセスするデータサイズに合った境界アドレスを指定しないとブレークしません。たとえば、アドレス0x0に割り付けられているint変数へのアクセス(ワードアクセス)でブレークさせたい場合、アドレス0x1～アドレス0x3を指定してもブレークしません。アクセスデータサイズがワード(32ビット)の場合は指定アドレスの下位2ビットを00、ハーフワード(16ビット)の場合はアドレスの最下位ビットを0とする必要があります。
- プログラムはブレーク条件成立から1～数命令後に停止します。

delete (ブレーク番号によるブレークの解除)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

設定されているブレークポイントをすべて、あるいはブレーク番号を指定して個別に削除します。

書 式

delete [*BreakNo*]

BreakNo: ブレーク番号(10進数または16進数)

省略すると、すべてのブレークポイントが削除されます。

使用例

```
(gdb)
info breakpoints      (ブレークポイントリストの表示)
Num Type              Disp Enb Address      What
1  breakpoint         keep y   0x00c00038 in sub at ./main.c:20
2  breakpoint         keep y   0x00c00030 in main at ./main.c:14
3  hw watchpoint      keep y                   i
```

上記のとおり、ブレークポイントが設定されているものとします。

例1

```
(gdb)
delete 1 2
(gdb)
info breakpoints
Num Type              Disp Enb Address      What
3  hw watchpoint      keep y                   i
```

ブレーク番号を指定すると、そのブレークのみ解除できます。複数のブレーク番号を一度に指定可能です。

例2

```
(gdb)
delete
(gdb)
info breakpoints
No breakpoints or watchpoints.
```

ブレーク番号を省略すると、すべてのブレークを解除します。

注 意

- ブレーク番号は、ブレーク設定時に個々のブレークポイントに対して1から順に割り当てられます。削除する時点で不明な場合は、上記例のとおり `info breakpoints` コマンドで確認してください。
- `delete` はブレークの設定を完全に削除します。一時的にブレークポイントを無効にしたい場合は、`disable` コマンドや `ignore` コマンドを使用してください。
- 設定されていないブレーク番号が指定された場合、"No breakpoint number N." を表示して削除は行いません。
- `c33 oab`、`c33 obb` コマンドで設定したブレークを `delete` コマンドで解除することはできません。

clear (ブレーク位置によるブレークの解除)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

設定されているPCブレークポイントを、設定位置(関数名、行番号、アドレス)を指定して個別に削除します。clearは、テンポラリブレークを含むソフトウェアPCブレークポイントとハードウェアPCブレークポイントを削除するコマンドで、データブレークポイントは削除できません。

書 式

clear [*Breakpoint*]

Breakpoint: ブレークポイント

以下の3種類で指定可能です。

- 関数名
- ソースファイル名:行番号、または行番号のみ
- *アドレス(10進数、16進数、またはシンボル)

省略すると、現在のPCアドレスを含むソースに設定されているすべてのブレークポイントが削除されます。

条件: $0 \leq \text{アドレス} \leq 0\text{xfffffff}$

使用例

```
(gdb)
info breakpoints      (ブレークポイントリストの表示)
Num Type             Disp Enb Address      What
1  breakpoint        keep y   0x00c0001e in main at ./main.c:10
2  breakpoint        keep y   0x00c00038 in sub at  ./main.c:20
3  breakpoint        keep y   0x00c0003c in sub at  ./main.c:22
4  breakpoint        keep y   0x00c00042 in sub at  ./main.c:22
5  hw watchpoint     keep y                   i
```

上記のとおり、ブレークポイントが設定されているものとします。ブレーク番号3と4は異なるアドレスですが、Cソースで見ると1行に設定されています(ASSEMBLY表示のアドレスにブレークポイントを設定したケース)。

例1

```
(gdb)
clear main.c:22
Deleted breakpoints 4 3
(gdb)
info breakpoints
Num Type             Disp Enb Address      What
1  breakpoint        keep y   0x00c0001e in main at ./main.c:10
2  breakpoint        keep y   0x00c00038 in sub at  ./main.c:20
5  hw watchpoint     keep y                   i
```

行番号を指定すると、そのソース行に設定されているすべてのブレークポイントを解除します。

例2

```
(gdb)
clear main
Deleted breakpoint 1
(gdb)
info breakpoints
Num Type             Disp Enb Address      What
2  breakpoint        keep y   0x00c00038 in sub at  ./main.c:20
5  hw watchpoint     keep y                   i
```

関数名を指定すると、その関数内の最初のC命令(ニーモニックに展開されるもの)に設定されているブレークポイントを解除します。"break 関数名"等で設定したブレークポイントの削除に使用します。

■例3

```
(gdb)
clear
Deleted breakpoint 2
(gdb)
info breakpoints
Num Type             Disp Enb Address      What
5   hw watchpoint    keep y              i
```

パラメータを省略すると、現在のPCが示すアドレスを含むソース内(この例ではmain.c)に設定されているすべてのブレークポイントが削除されます。そのソース内にブレークポイントが設定されていない場合は"No source file specified."が表示され、削除は行いません。

■例4

```
(gdb)
clear *i
No breakpoint at *i.
(gdb)
info breakpoints
Num Type             Disp Enb Address      What
5   hw watchpoint    keep y              i
```

変数*i*に設定してあるデータブレークポイントはclearコマンドでは削除できません。

■注 意

- clearはデータブレークポイントを削除できません。deleteコマンドで対応してください。
- clearはブレークの設定を完全に削除します。一時的にブレークポイントを無効にしたい場合は、disableコマンドやignoreコマンドを使用してください。
- ブレークポイントに設定されていない関数名、行番号、アドレスを指定するとエラーとなります。
- c33 oab、c33 obbコマンドで設定したブレークをclearコマンドで解除することはできません。

enable (ブレークポイントの有効化)**disable** (ブレークポイントの無効化)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

enable: 無効に設定されているブレークポイントを有効な状態に切り換えます。

disable: 有効に設定されているブレークポイントを無効な状態に切り換えます。

ブレークコマンドでブレークポイントを設定した段階では、ブレークは有効となります。disable コマンドはこれを、ブレークポイントを削除せずに無効にします。enable コマンドで有効に戻すまでは、そのブレークポイントではブレークしません。

書 式

enable [*BreakNo*]

disable [*BreakNo*]

BreakNo: ブレーク番号(10進数または16進数)

省略すると、すべてのブレークポイントが対象になります。

使用例

```
(gdb)
info breakpoints      (ブレークポイントリストの表示)
Num Type             Disp Enb Address      What
1  breakpoint        keep y  0x00c0001c in main at ./main.c:7
2  breakpoint        keep y  0x00c0001e in main at ./main.c:10
3  breakpoint        keep y  0x00c00028 in main at ./main.c:13
4  breakpoint        keep y  0x00c00038 in sub at ./main.c:20
5  hw watchpoint     keep y                      i
```

上記のとおり、ブレークポイントが設定されているものとします。有効なブレークポイントはEnbがyと表示されます。

■例1

```
(gdb)
disable 1 3
(gdb)
info breakpoints
Num Type             Disp Enb Address      What
1  breakpoint        keep n  0x00c0001c in main at ./main.c:7
2  breakpoint        keep y  0x00c0001e in main at ./main.c:10
3  breakpoint        keep n  0x00c00028 in main at ./main.c:13
4  breakpoint        keep y  0x00c00038 in sub at ./main.c:20
5  hw watchpoint     keep y                      i
```

ブレーク番号を付けてdisableコマンドを実行すると、そのブレークのみ無効にできます。複数のブレーク番号を一度に指定可能です。無効なブレークポイントはEnbがnと表示されます。

■例2

```
(gdb)
disable 5
(gdb)
info breakpoints
Num Type             Disp Enb Address      What
1  breakpoint        keep n  0x00c0001c in main at ./main.c:7
2  breakpoint        keep y  0x00c0001e in main at ./main.c:10
3  breakpoint        keep n  0x00c00028 in main at ./main.c:13
4  breakpoint        keep y  0x00c00038 in sub at ./main.c:20
5  hw watchpoint     keep n                      i
```

データブレークポイントの有効/無効の切り換えも行えます。

■例3

```
(gdb)
enable
(gdb)
info breakpoints
```

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x00c0001c	in main at ./main.c:7
2	breakpoint	keep	y	0x00c0001e	in main at ./main.c:10
3	breakpoint	keep	y	0x00c00028	in main at ./main.c:13
4	breakpoint	keep	y	0x00c00038	in sub at ./main.c:20
5	hw watchpoint	keep	y		i

ブレーク番号を省略すると、すべてのブレークポイントが一度に有効(無効)に設定されます。

注 意

- ブレーク番号は、ブレーク設定時に個々のブレークポイントに対して1から順に割り当てられます。有効/無効にする時点で不明な場合は、上記例のとおり`info breakpoints`コマンドで確認してください。
- ブレークポイントは設定数に制限がありますので、使用しないブレークポイントは`delete`コマンドで削除してください。
- 設定されていないブレーク番号が指定された場合、"No breakpoint number N."を表示して設定は行いません。
- `c33 oab`、`c33 obb`コマンドで設定したブレークを`enable/disable`コマンドで有効/無効にすることはできません。

ignore (回数指定付きブレークの無効化)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

ブレークヒット数を指定し、その回数だけ特定のブレークを無効にします。

書 式

ignore BreakNo Count

BreakNo: ブレーク番号(10進数)

Count: 無効にするブレークヒット数(10進数または16進数)

使用例

```
(gdb)
info breakpoints
Num Type           Disp Enb Address      What
1  breakpoint      keep y   0x00c0003c in sub at ./main.c:22
2  breakpoint      keep y   0x00c00030 in main at ./main.c:14
```

```
(gdb)
```

ignore 2 2

Will ignore next 2 crossings of breakpoint 2.

ブレーク番号2を2回無効にします。

```
(gdb)
```

continue

Continuing.

Breakpoint 1, sub (k=1) at ./main.c:22

```
(gdb)
```

continue

Continuing.

Breakpoint 1, sub (k=1) at ./main.c:22

```
(gdb)
```

continue

Continuing.

Breakpoint 2, main () at ./main.c:14

上記2回の実行(continue)でターゲットプログラムはブレークポイント2を通過していますが、ブレークは発生しません。3度目の実行では無効設定が解除されているため、ブレークが発生します。

注 意

- ブレーク番号は、ブレーク設定時に個々のブレークポイントに対して1から順に割り当てられます。削除する時点で不明な場合は、上記例のとおりinfo breakpointsコマンドで確認してください。
- ブレーク無効回数は指定ブレークのヒット数をカウントする値で、ターゲットプログラムの実行回数をカウントするものではありません。指定のブレークポイントを通過しない場合、無効回数は減りません。
- ignoreコマンドでは、複数のブレークポイントをまとめて無効にすることはできません。
- 設定されていないブレーク番号が指定された場合、"No breakpoint number N."を表示して実行を中断します。
- c33 oab、c33 obbコマンドで設定したブレークをignoreコマンドで無効にすることはできません。

info breakpoints (ブレイクポイントリストの表示) [ICD2 / ICD3 / ICD6 / SIM / MON]**機 能**

設定されているブレイクポイントの一覧を表示します。

書 式

info breakpoints

表 示

一覧は次のように表示されます。

```
(gdb)
info breakpoints
Num Type           Disp Enb Address      What
1  breakpoint      keep y   0x00c00026 in main at ./main.c:11
    breakpoint already hit 1 time
    ignore next 10 hits
2  hw breakpoint   del  n   0x00c00038 in sub at ./main.c:20
3  hw watchpoint  keep y           i
    breakpoint already hit 6 times
```

Num: ブレイク番号を表示します。

Type: ブレイクポイントの種類を表示します。

breakpoint ソフトウェアPCブレイクポイント

hw breakpoint ハードウェアPCブレイクポイント

hw watchpoint データブレイクポイント

Disp: ブレイクヒット後のブレイクポイントの状態を表示します。

keep ブレイクポイントは削除されません。

del ブレイクポイントが削除されます。テンポラリブレイクであることを示します。

Enb: ブレイクポイントが有効か無効かを表示します。

y 有効

n 無効

Address: ブレイクポイントに設定されたアドレスを16進数で表示します。

データブレイクの場合は表示されません。

What: ブレイクポイントが設定された位置を表示します。

PCブレイクポイントは、"in 関数名 at ソースファイル名:行番号"の形式で表示されます。

データブレイクポイントは、指定したシンボルやアドレスが表示されます。

その他、各ブレイクポイントごとに、現在までのヒット回数を"breakpoint already hit N times"、ignoreコマンドの設定内容を"ignore next N hits"の形で表示します。

ブレイクポイントが1カ所も設定されていないときは、次のように表示されます。

```
(gdb)
info breakpoints
No breakpoints or watchpoints.
```

注 意

c33 oab、c33 obbコマンドで設定したブレイク情報をinfo breakpointsコマンドで表示することはできません。

c33 oab (オンチップエリアブレイクの設定)

[ICD3 / ICD6]

機 能

バスマスタやアクセスされたエリア、リード/ライト条件によりブレイクを発生可能なエリアブレイクを設定します。

本コマンドは、ターゲットCPUがC33 ADVコアの場合のみ有効です。

エリアブレイクの設定状態は、表示モード選択の右横に表示されます。

エリアブレイク有効時	area: on
エリアブレイク無効時	area: off

書 式**c33 oab Mode BBCU_BusMaster R/W Area**

Mode: オンチップエリアブレイクモード選択
on オンチップエリアブレイク設定
off オンチップエリアブレイク解除
 解除の場合、*BBCU_BusMaster*以降のパラメータは設定しないでください。

BBCU_BusMaster: BBCUバスマスタ選択(10進数または16進数)
 選択するビットを1にします。(複数指定可、1つ以上選択すること)
 bit3 = 1: USER(CPU)によるメモリアクセス
 bit2 = 1: DMAによるメモリアクセス
 bit1 = 1: キャッシュによるリフィル/ライトバック時のメモリアクセス
 bit0 = 1: CPUのメモリアクセス
 0x01～0x0fの範囲で指定します。

R/W: リード/ライト条件
r リード
w ライト
rw リード/ライト

Area: エリア指定(10進数または16進数)
 指定可能なエリアは0～22です。
 0x00000001～0x7ffffffの範囲で指定します。 bit0～bit22がエリア0～エリア22に相当し、ビットを1に設定すると対応するエリアがブレイク条件に含められ、0に設定するとブレイク条件から外れます。

使用例**■例1**

```
(gdb)
c33 oab on 0x05 w 0x3e2070
Area break on
BBCU bus master select :DMA,CPU
Read/Write             :Write
Area                   :4,5,6,13,17,18,19,20,21
```

エリア4～6、13、17～21にDMAまたはHBCUが書き込みを行った場合にブレイクします。
 ブレイク時は次のようなメッセージを表示します。

Hardware on chip Area Break!

<<<Area Break Status>>>

Bus master	: HBCU (CPU)	; バスマスタ	HBCU(CPU)/CCU/DMA/USER
Access	: Write	; アクセス	Read/Write
Area	: 4	; エリア番号	(0-22)
Data type	: Data	; データタイプ	Instruction/Data
Data size	: Word	; データサイズ	Byte/Half Word/Word

■例2

```
(gdb)
c33 oab off
Area break off
```

オンチップエリアブレイクを解除します。

注 意

- c33 oabコマンドはICD3またはICD6モード時のみ有効です。
- AREA2のキャッシュ、MMUエリアについては本機能の対象外とします。
- AREA0およびAREA3については、MMUによる変換先がエリア0、3であった場合のみブレイクします。
- ターゲットにオンチップエリアブレイク機能がない場合は、エラーになります。

c33 obb (オンチップバスブ레이크設定)

[ICD3 / ICD6]

機 能

バスの状態などの条件を細かく指定してブ레이크可能なバスブ레이크を設定します。

本コマンドは、ターゲットCPUがC33 ADVコアの場合のみ有効です。

バスブ레이크の設定状態は、表示モード選択の右横に表示されます。

バスブ레이크有効時 bus: on

バスブ레이크無効時 bus: off

書 式

c33 obb *Mode Bus BBCU_BusMaster Trig/Break R/W Inst/Data Access Int IntLevel DataComp*
Address AddressMask Data DataMask Counter CounterClr/Save

Mode: オンチップバスブ레이크モード選択

on オンチップバスブ레이크設定

off オンチップバスブ레이크解除

解除の場合、*Bus*以降のパラメータは設定しないでください。

Bus: バス選択

0 CPU系データバス

1 CPU系命令バス

2 BBCU系バス

BBCU_BusMaster: BBCUバスマスタ選択(10進数または16進数)

選択するビットを1にします。(Bus = 2の場合、1つ以上選択すること。複数指定可)

bit3 = 1: USER(CPU)によるメモリアクセス

bit2 = 1: DMAによるメモリアクセス

bit1 = 1: キャッシュによるリフィル/ライトバック時のメモリアクセス

bit0 = 1: CPUのメモリアクセス

0x01～0x0fの範囲で指定します。

Trig/Break: トリガ/ブ레이크選択

0 トリガ出力のみ

1 ブ레이크発生

R/W: リード/ライト条件

r リード

w ライト

rw リード/ライト

Inst/Data: 命令/データ条件

0 命令

1 データ

2 命令/データ

条件に命令(0、2)を選択した場合、*BBCU_BusMaster*でCPUのメモリアクセスを選択していないとブ레이크しません。

Access: アクセス条件(10進数または16進数。複数指定可)

bit2 = 1: ワードアクセス(Word)

bit1 = 1: ハーフワードアクセス(Half)

bit0 = 1: バイトアクセス(Byte)

0x01～0x07の範囲で指定します。

Int: 割り込み条件

0 割り込みレベルによるバスブ레이크は無効

1 割り込みレベルによるバスブ레이크は有効

<i>IntLevel:</i>	割り込みレベル(10進数または16進数) 0~15 <i>Int</i> が0(無効)の場合でも値を設定してください。ただし、設定した値は無効になります。
<i>DataComp:</i>	データ値分割比較指定 0 バス上のデータ値を比較する際、32ビットで比較します。 1 バス上のデータ値を比較する際、上位16ビット/下位16ビットを個別に比較し、一致条件をORします。
<i>Address:</i>	アドレス(10進数または16進数) 0x00000000~0xffffffffの範囲で指定します。
<i>AddressMask:</i>	アドレスマスク(10進数または16進数) 0x00000000~0xffffffffの範囲で指定します。 1を指定したアドレスビットのみバス上のアドレス情報と比較します。
<i>Data:</i>	データ(10進数または16進数) 0x00000000~0xffffffffの範囲で指定します。
<i>DataMask:</i>	データ マスク(10進数または16進数) 0x00000000~0xffffffffの範囲で指定します。 1を指定したデータビットのみバス上のデータ情報と比較します。
<i>Counter:</i>	回数カウンタ(10進数または16進数) バスブレーク要因が何回発生するとブレークするかを、0x00000001~0x7ffffffの範囲で指定します。
<i>CounterClr/Save:</i>	回数カウンタ保持/クリア選択(10進数または16進数) 0 プログラム実行時、回数カウンタを保持します。 1 プログラム実行時、回数カウンタをクリア(再設定)します。

使用例

■例1

```
(gdb)
c33 obb on 0 1 1 w 1 0x04 0 0 0 0 0x55555555 0xffffffff 3 0
Bus break on
Bus select           : CPU data bus
BBCU bus master select : ---
Trigger/Break       : Break
Read/Write          : Write
Instruction/Data     : Data
Access size         : Word
Break by interrupt level : Disable
Interrupt level      : 0
Data compare        : 32bit
Address             : 0
Address mask        : 0
Data               : 0x55555555
Data mask          : 0xffffffff
Break counter       : 3
Break counter Clear/Save : Save
```

CPU系データバスで、ワードデータ0x55555555の書き込みが3回発生した場合にブレークさせます。
また、バス上のデータ値を32ビットで比較します。
ブレークには次のメッセージが表示されます。

```
Hardware on chip Bus Break!
```

■例2

```
(gdb)
c33 obb off
Bus break off
```

オンチップバスブレークを解除します。

注 意

- c33 obbコマンドはICD3またはICD6モード時のみ有効です。
- 本コマンドでブレーク設定後のプログラム実行時は、ブレーク回数カウンタを表示します。
- ターゲットにオンチップバスブレーク機能がない場合は、エラーになります。
- 本コマンドは、バストレースと同じ設定にする必要があります。
バスブレークとバストレースのバス選択で異なる設定をした場合、後に実行した設定が有効になります。

c33 timebrk (時間経過後ブレークの設定)

[ICD6]

機 能

プログラム実行後、強制ブレークするまでの時間を設定、または解除します。

書 式**c33 timebrk *Timer***

Timer: プログラム実行開始からブレークするまでのミリ秒単位の時間(10進数または16進数)

1～300000(ミリ秒)の範囲で設定可能です。

0を指定すると、本ブレーク機能が無効となります(デバッガ起動時のデフォルト設定)。

使用例**■例1**

```
(gdb)
c33 timebrk 1000
      timer break on. [1000 ms]
(gdb)
cont
Continuing.
```

実行開始から1秒後に強制ブレークします。

■例2

```
(gdb)
c33 timebrk 0
      timer break off.
```

時間経過後ブレークを解除します。

注 意

- 本コマンドは、ICD6モード時のみ有効です。
- ブレーク時間を設定すると、"c33 timebrk 0"で解除するまで時間経過後ブレークは有効です。プログラムを実行させると毎回、設定した時間でブレークします。
- 設定時間の経過前に他のブレーク条件に一致、あるいは[中断]ボタンをクリックした場合は、その時点で直ちにブレークします。

10.7.9 シンボル情報表示コマンド

info locals (ローカルシンボル表示)

info var (グローバルシンボル表示)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機能

シンボルの一覧を表示します。

info locals: 現在の関数内で定義されているローカル変数の一覧を表示します。

info var: グローバルおよびスタティック変数の一覧を表示します。

書式

info locals

info var

使用例

■例1

```
(gdb)
info locals
i = 0
j = 2
```

現在のPCアドレスを含む関数内のローカルシンボルをすべて、その内容と共に表示します。

■例2

```
(gdb)
info var
All defined variables:

File ./main.c:
int i;

Non-debugging symbols:
0x00000000 __START_bss
0x00000004 __END_bss
0x00000004 __END_data
0x00000004 __START_data
```

定義されているすべてのグローバルおよびスタティック変数を、ソースファイルごとに一覧表示します。"Non-debugging symbols:"には、セクションシンボル等、ソースファイル以外で定義されたグローバルシンボルが表示されます。

注意

PCアドレスで示される現在位置が関数(スタックフレーム)外の場合(アセンブリソースのブートルーチン内など)、ローカルシンボルは表示されません。

```
(gdb)
info locals
No frame selected.
```

print (シンボル値の変更)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

シンボルの値を変更します。

書 式

print *Symbol* [=*Value*]

Symbol: 変数名

Value: 変更する値(10進数、16進数、またはシンボル)

省略時は現在のシンボル値を表示します。

条件: $0 \leq \text{Value} \leq \text{型の有効範囲}$

使用例**■例1**

```
(gdb)
info local
j = 0
(gdb)
print j
$1 = 0
```

変数名のみを指定すると、その値を表示します。 $\$N$ は後からこの値を参照するための番号です。**print** $\$1$ で、ここで表示した内容を参照できます。

■例2

```
(gdb)
print j=5
$2 = 5
(gdb)
info local
j = 5
```

値を指定すると、変数がその値に変更されます。

注 意

- 定義されていないシンボルを指定すると、エラーとなります。
- 変更する変数の型の範囲を超えた値を指定してもエラーとはなりません。変数のサイズに相当する下位側のビット数分のみが有効となり、それを超えた分は無視されます。たとえば、**int**変数に0x100000000を指定すると、0x00000000として処理されます。

10.7.10 ファイル読み込みコマンド

file (デバッグ情報の読み込み)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

elf形式のオブジェクトファイルからデバッグ情報のみを読み込みます。
オブジェクトコードはloadコマンドを使用して読み込みます。

書 式

file *Filename*

Filename: デバッグするelf形式のオブジェクトファイル名(パスも指定可能)

使用例

```
(gdb)
file sample.elf
```

カレントディレクトリのsample.elfからデバッグ情報を読み込みます。

注 意

- fileコマンドはデバッグ情報のみを読み込み、オブジェクトコードは読み込みません。したがって、ターゲットのROMにプログラムが書き込まれている場合を除き、fileコマンドの実行のみではデバッグを開始することはできません。
- fileコマンドは、targetコマンド、loadコマンドより先に実行してください。基本的な実行順序は次のとおりです。

(gdb)	
file sample.elf	(本コマンド)
(gdb)	
c33 rpf sample.par	(マップ情報設定)
(gdb)	
target sim	(ターゲットの接続)
(gdb)	
load	(プログラムのロード)
(gdb)	
c33 rsth	(ホットリセット)
- fileコマンドでは実行形式のelfオブジェクトファイル(リンカにより生成)以外はエラーとなり読み込めません。また、読み込んだファイルにデバッグ情報がない場合もエラーとなります。
- elf形式のオブジェクトファイルは、ディレクトリ構造も含めたソースファイル情報を持っています。このため、ソースファイルがカレントディレクトリから見てオブジェクトファイル内の指定ディレクトリ上にないと、ソースファイルが読み込めません。
基本的にはコンパイルからデバッグまでの操作を、同一のディレクトリで行ってください。
- fileコマンド実行後は、ファイルの読み込みを終了するまで中断はできません。
- ICDとの接続中(target icd/icd6 usbコマンド実行後)にfileコマンドを実行すると、c33 rpfコマンドで設定したマップ情報の設定はクリアされてしまいます。この場合、fileコマンドを実行した後に再度c33 rpfコマンドを実行してください。

load (プログラムのロード)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

elf形式のオブジェクトファイルからプログラム/データを読み込み、ターゲットのメモリにロードします。

書 式

load [*Filename*]

Filename: デバッグするelfまたはモトローラS3形式のオブジェクトファイル名(パスも指定可能)
省略すると、先にfileコマンドで指定したファイルが読み込まれます。通常は省略します。

使用例

```
(gdb)
file sample.elf
(gdb)
target sim
(gdb)
load
```

カレントディレクトリのsample.elf(fileコマンドで指定)からプログラム/データを読み込み、ターゲットメモリ(この例ではシミュレータモードのため、パソコンのメモリ)にロードします。

注 意

- loadコマンドは、fileコマンド、targetコマンドより後に実行してください。基本的な実行順序は次のとおりです。

```
(gdb)
c33 rpf sample.par           (マップ情報設定)
(gdb)
file sample.elf              (デバッグ情報の読み込み)
(gdb)
target sim                    (ターゲットの接続)
(gdb)
load                        (本コマンド)
(gdb)
c33 rsth                      (ホットリセット)
```

- loadコマンドはオブジェクトファイル内のコードおよびデータが存在する領域のみを読み込みます。それ以外の領域はloadコマンド実行前の状態のまま残りますので注意してください。

10.7.11 マップ情報コマンド

c33 rpf (マップ情報の設定)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

指定のパラメータファイルを読み込んで、メモリマップ情報を設定します。

書 式

c33 rpf *Filename*

Filename: パラメータファイル名(パスも指定可能)

使用例

(gdb)

c33 rpf sample.par

カレントディレクトリのsample.parからメモリマップ情報をロードします。

注 意

- パラメータファイルについては、"10.8 パラメータファイル"を参照してください。
- c33 rpfコマンドは、targetコマンドより先に一度だけ実行してください。基本的な実行順序は次のとおりです。

(gdb)

c33 rpf sample.par (本コマンド)

(gdb)

file sample.elf (デバッグ情報の読み込み)

(gdb)

target sim (ターゲットの接続)

(gdb)

load (プログラムのロード)

(gdb)

c33 rsth (ホトリセット)

メモリマップ情報を再設定する場合は、デバッグを一旦終了し、再起動してからc33 rpfコマンドを実行してください。

- デバッグはパラメータファイルから読み込んだメモリマップ情報に従って、パソコン上のメモリにその領域を確保します。領域が確保できないような場合は(エラーメッセージ"Cannot allocate memory."が表示されます)、ひとつのエリアにつき256MB程度を超えないように修正するか、エリアを複数に分割するなどしてください。
- シミュレータモード以外の時、パラメータファイル(*.par)で"ROM"属性にした領域には、ソフトウェアPCブレークおよびテンポラリソフトウェアPCブレークは設定できません。

c33 map (マップ情報の表示)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

パラメータファイルにより設定されているメモリマップ情報を表示します。

書 式

c33 map

表 示

```
(gdb)
c33 map
CPU : Standard
Memory map information
Type      Wait (r/w)  Size  Endian
00000000-000007ff RAM      7/7    16Bit Little
00040000-0004ffff IO        7/7    16Bit Little
00200000-002fffff RAM      7/7    16Bit Little
00600000-006fffff RAM      7/7    16Bit Little
00c00000-00cfffff ROM      7/7    16Bit Little
00600000-006fffff STACK
```

CPU: CPUの種類を表示します。
 Standard: C33 STD
 Advanced: S1C33401 (C33 ADV)
PE: C33 PE

Memory map information:

デバイスが割り付けられているメモリアドレス範囲(領域先頭アドレス - 領域終了アドレス)を16進数で表示します。

Type: メモリ/デバイスの種類を表示します。

Wait(r/w): 読み出し時/書き込み時に挿入するウェイト数を表示します。

Size: デバイスのデータ幅(ビット数)を表示します。

Endian: 領域のエンディアン(リトルエンディアン/ビッグエンディアン)を表示します。

c33 rpf コマンドによるメモリマップ情報の読み込みが済んでいない場合は、次のように表示されます。

```
(gdb)
c33 map
CPU : Standard
Memory map information
Type      Wait (r/w)  Size  Endian
No data.
```

10.7.12 フラッシュメモリ操作コマンド

c33 fls (フラッシュメモリ設定)

[ICD2 / ICD3 / ICD6 / MON]

機 能

ターゲットシステムのフラッシュメモリにデータを書き込むための設定を行います。

書 式

c33 fls *StartAddr EndAddr ErasePrg WritePrg*

StartAddr: フラッシュメモリ先頭アドレス(10進数、16進数、またはシンボル)

EndAddr: フラッシュメモリ終端アドレス(10進数、16進数、またはシンボル)

ErasePrg: 消去プログラム先頭アドレス(10進数、16進数、またはシンボル)

WritePrg: 書き込みプログラム先頭アドレス(10進数、16進数、またはシンボル)

条件: $0 \leq \textit{StartAddr} \leq \textit{EndAddr} \leq 0\text{xffffffff}$ 、 $0 \leq \textit{ErasePrg} \leq 0\text{xffffffff}$ 、 $0 \leq \textit{WritePrg} \leq 0\text{xffffffff}$

使用例

(gdb)

c33 fls 0x200000 0x2fffff FLASH_ERASE FLASH_LOAD

Flash start address = 0x200000, Flash end address = 0x2fffff

Flash erase routine address = 0x100, Flash write routine address = 0x200 done

ターゲットシステムのフラッシュメモリ領域(0x200000～0x2fffff)と消去および書き込みルーチンのアドレスを設定します。

注 意

- 本コマンドはシミュレータモードでは使用できません。
- ターゲットシステムのフラッシュメモリを消去、あるいはフラッシュメモリにデータを書き込むためには、本コマンドを実行するとともに、データ書き込みおよび消去プログラムも指定のアドレスに書き込んでおく必要があります。フラッシュ書き込みプログラムについては、tool¥fls33¥の内容を参照してください。
- アドレス指定は下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。

c33 fle (フラッシュメモリ消去)

[ICD2 / ICD3 / ICD6 / MON]

機 能

ターゲットシステムのフラッシュメモリの内容を消去します。

書 式

c33 fle *ControlReg StartBlock EndBlock* [*Timer*]

ControlReg: c33 fleで設定した先頭アドレス(10進数、16進数、またはシンボル)

StartBlock: 消去範囲の先頭ブロック(10進数、16進数、またはシンボル)

EndBlock: 消去範囲の終端ブロック(10進数、16進数、またはシンボル)

StartBlock = *EndBlock* = 0で全領域を消去します。

Timer: タイムアウト値(10進数または16進数)

秒数を指定します。省略時は150秒でタイムアウトになります。

使用例

(gdb)

```
c33 fle 0x200000 0 0
```

```
Control Register = 0x200000, Start block = 0x0, End block = 0x0 .....Finish with 0x00000000
```

フラッシュメモリの全領域を消去します。

注 意

- 本コマンドはシミュレータモードでは使用できません。
- ターゲットシステムのフラッシュメモリを消去するためには、データ書き込みおよび消去プログラムをターゲットシステムメモリに書き込み、c33 fleコマンドを実行しておく必要があります。消去プログラムが設定されていない状態でc33 fleコマンドを実行するとエラーとなります。
- ターゲットシステムのフラッシュメモリにデータを書き込む場合は、必ず先に本コマンドで消去を行ってください。

10.7.13 トレースコマンド

c33 tm (PC トレースモード設定)

[ICD2 / ICD3 / ICD6] [SIM]

注: c33 tm コマンドはICDモードとシミュレータモードでは機能と書式が異なりますので注意してください。以下、モードごとに分けて説明します。

ICDモード

機能 (ICD2/ICD3/ICD6)

ICDモードではトレースモード、トリガアドレス等を設定します。
設定する項目は以下のとおりです。

トレースモード

次の3種類のトレースモードが設定可能です。

1. 全体トレースモード、オーバーライトあり

プログラムの実行開始と同時にトレースも開始し、トレース情報をS5U1C33000H/S5U1C33001Hのトレースメモリに書き込みます。

S5U1C33000H/S5U1C33001Hのトレースメモリがいっぱいになった場合、古いデータから上書きされます。したがって、トレースメモリは常に最新のトレース情報を保持します。

2. 全体トレースモード、オーバーライトなし

プログラムの実行開始と同時にトレースも開始し、トレース情報をS5U1C33000H/S5U1C33001Hのトレースメモリに書き込みます。

S5U1C33000H/S5U1C33001Hのトレースメモリがいっぱいになるまでトレース情報を書き込み、それ以降は書き込みを行いません。

3. 範囲トレースモード

トリガアドレス1からトリガアドレス2までの範囲内を実行した場合にのみ、トレース情報をS5U1C33000H/S5U1C33001Hのトレースメモリに書き込みます。

実行サイクル数の表示

トレース情報のクロック数(Clk)を命令ごとの実行クロック数として表示させるか、トレース開始から積算した値で表示させるか選択できます。

範囲トレースモードの条件設定

範囲トレースモードでは以下の条件を指定できます。

1. トリガアドレス

トレース範囲の先頭アドレス(トリガアドレス1)と終了アドレス(トリガアドレス2)を指定します。

2. トリガアドレス2でのブレーク

指定範囲をトレース後、トリガアドレス2でプログラムの実行をブレークするか、継続するか指定することができます。

3. 時間測定条件

プログラムの実行時間を、実行開始からブレークするまで測定するか、トリガアドレス1からトリガアドレス2までのトレース範囲内の実行時間のみを測定するか選択できます。

書 式 (ICD2/ICD3/ICD6)

c33 tm Mode CycleInfo [Trigger1 Trigger2 Break Measure]

Mode: トレースモード

- 1 全体トレースモード、オーバーライトなし
- 2 全体トレースモード、オーバーライトあり(デフォルト)
- 3 範囲トレースモード

CycleInfo: 実行サイクル数表示内容

- 1 トレース開始からの累積(デフォルト)
- 2 命令単位

Trigger1: 範囲トレース開始アドレス(10進数、16進数、またはシンボル)

Trigger2: 範囲トレース終了アドレス(10進数、16進数、またはシンボル)

Break: 範囲トレース後のブレーク

- 1 *Trigger2*でブレークしない
- 2 *Trigger2*でブレークする

Measure: 範囲トレース時間測定

- 1 実行開始からブレークまでを測定
- 2 トレース範囲内の実行時間のみを測定

条件: $0 \leq \text{Trigger1} \leq 0\text{xffffffff}$ 、 $0 \leq \text{Trigger2} \leq 0\text{xffffffff}$

Trigger1、*Trigger2*、*Break*、*Measure*は、範囲トレースモードの場合は必ず設定してください。全体トレースモードの場合、これらのパラメータは指定しません。

使用例 (ICD2/ICD3/ICD6)**■例1**

```
(gdb)
c33 tm 2 2
```

オーバーライトありの全体トレースモード、命令単位の実行サイクル数表示を設定します。

■例2

```
(gdb)
c33 tm 3 1 0x600000 0x600100 2 2
```

アドレス0x600000から0x600100までの範囲トレースモード、実行サイクル数の積算表示、トレース後のブレークあり、トレース範囲内の実行時間測定を設定します。

トレース情報 (ICD2/ICD3/ICD6)

本コマンドによるトレースモードの設定後、ターゲットプログラムを連続実行させるとトレース情報がS5U1C33000H/S5U1C33001Hのトレースメモリに採取されます。ブレーク後、**c33 td**コマンドにより、採取したトレース情報を[トレース]ビューに表示させることができます。**c33 tf**コマンドによりファイルに書き出すことも可能です。

表示されるトレース情報の内容は以下のとおりです。

Cycle	Address	Code	Unassemble	Clk	Method	File	Line	SourceCode
000044	0C00004	C020	ext 0x20	000000	SPC	(./boot.s)	00009	xld.w ...
000043	0C00006	6C0F	ld.w %r15,0x0	000016	SPC			
000042	0C00008	A0F1	ld.w %sp,%r15	000032	SPC	(./boot.s)	00010	ld.w ...
			:					
000003	0C00046	2E54	ld.w %r4,%r5	000776	DPC			
000002	0C00048	7014	and %r4,0x1	000792	DPC			
000001	0C0004A	6804	cmp %r4,0x0	000808	DPC			
000000	0C0004C	1809	jreq 0x9	000824	DPC			

Cycle:	トレースサイクル(10進数) トレースメモリ内に最後に取り込まれた情報が000000となります。
Address:	CPUの命令実行アドレス(16進数)
Code:	CPUが実行した命令コード(16進数)
Unassemble:	命令コードの逆アセンブル内容
Clk:	実行クロック数 <i>CycleInfo</i> の指定により、トレース開始からの積算表示、命令単位の表示が選択できます。 (表示例は積算表示)
Method:	トレース解析(実行アドレス取得)方法 SPC: 初期PCによる解析 TRG: トリガアドレスによる解析 DPC: DPCO信号による解析 RET: call文、ret文による解析 MAP: マップ情報による解析 RTI: reti文による解析 ---: 解析不能
File:	実行命令を含むソースファイル名
Line:	ソース行番号
SourceCode:	ソースコード

注 意 (ICD2/ICD3/ICD6)

- 範囲トレースモードを設定すると、ハードウェアPCブレイク機能は無効となります。ただし、ハードウェアPCブレイクアドレスの設定はクリアされず、範囲トレースモードを解除した時点で有効になります。
- トリガアドレスは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。
また、アドレスを奇数の値で指定すると最下位ビットを0として16ビット境界に補正されます。

シミュレータモード

機能 (SIM)

シミュレータモードでは以下の選択のみが行えます。

トレース機能のON/OFF

ONに設定すると、プログラムの実行に従ってトレース情報が採取されます。

レジスタ値、ソースコードの表示

基本的なトレース情報のほかにレジスタの内容やソースコードも表示するか選択できます。

トレース情報の出力先

採取したトレース情報の出力先として、ウィンドウまたはファイルを選択できます。

[トレース]ビューを選択した場合は、トレース情報が[トレース]ビューに表示されます。

ファイル選択時は、ファイル名も指定します。

書式 (SIM)

c33 tm on Mode [Filename] (トレースモード設定)

c33 tm off (トレースモード解除)

Mode: トレースモード(トレース情報表示内容)

- 1 ノーマル表示
- 2 レジスタ表示
- 3 ソース情報付きノーマル表示
- 4 ソース情報付きレジスタ表示

Filename: トレース情報出力ファイル名

指定すると採取したトレース情報はファイルに出力され、[トレース]ビューには表示されません。

省略すると[トレース]ビューに表示され、ファイルには出力されません。

使用例 (SIM)

■例1

```
(gdb)
c33 tm on 4 trace.log
```

シミュレータモードで、ソース情報付きレジスタ表示のトレースを設定し、情報を保存するファイル `trace.log` を指定します。この設定以降にプログラムを実行すると、実行した命令ごとにトレース情報をファイルに出力します。ファイル名を省略すると、[トレース]ビューに表示されます。

■例2

```
(gdb)
c33 tm off
```

シミュレータのトレースモードを解除します。これ以降は、プログラムを実行してもトレース情報は採取しません。

トレース情報 (SIM)

本コマンドによるトレースモードの設定後、ターゲットプログラムを実行すると、実行した命令ごとにトレース情報を[トレース]ビューに表示、またはファイルに出力します。
表示/出力されるトレース情報の内容は以下のとおりです。

Mode = 1 (ノーマル表示)

```
00000001 00C00004 C020 ext 0x20 ----- 00000008
00000002 00C00006 6C0F ld.w %r15,0x0 ----- 00000016
00000003 00C00008 A0F1 ld.w %sp,%r15 ----- 00000024
:
00000015 00C00026 6C04 ld.w %r4,0x0 ----- 00000150
00000016 00C00028 5C04 ld.w [%sp+0x0],%r4 000007F8 wW 00000000 00000173
```

Mode = 2(レジスタ表示)

```

00000001 00C00004 C020  ext    0x20          ----- 00000008
AAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
0AAAAAA8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000002 00C00006 6C0F  ld.w    %r15,0x0      ----- 00000016
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
0AAAAAA8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000003 00C00008 A0F1  ld.w    %sp,%r15      ----- 00000024
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
0AAAAAA8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000000 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
:
00000015 00C00026 6C04  ld.w    %r4,0x0      ----- 00000150
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA 00000000 AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
000007F8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000016 00C00028 5C04  ld.w    [%sp+0x0],%r4 000007F8 wW 00000000 00000173
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA 00000000 AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
000007F8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0

```

Mode = 3(ソース情報付きノーマル表示)

```

00000001 00C00004 C020  ext    0x20          ----- 00000008 (./boot.s) 00009 xld.w ...
00000002 00C00006 6C0F  ld.w    %r15,0x0      ----- 00000016
00000003 00C00008 A0F1  ld.w    %sp,%r15      ----- 00000024 (./boot.s) 00010 ld.w ...
:
00000015 00C00026 6C04  ld.w    %r4,0x0      ----- 00000150 (./main.c) 00011 for ...
00000016 00C00028 5C04  ld.w    [%sp+0x0],%r4 000007F8 wW 00000000 00000173

```

Mode = 4(ソース情報付きレジスタ表示)

```

00000001 00C00004 C020  ext    0x20          ----- 00000008 (./boot.s) 00009 xld.w ...
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
0AAAAAA8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000002 00C00006 6C0F  ld.w    %r15,0x0      ----- 00000016
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
0AAAAAA8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000003 00C00008 A0F1  ld.w    %sp,%r15      ----- 00000024 (./boot.s) 00010 ld.w ...
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
0AAAAAA8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000000 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
:
00000015 00C00026 6C04  ld.w    %r4,0x0      ----- 00000150 (./main.c) 00011 for ...
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA 00000000 AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
000007F8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0
00000016 00C00028 5C04  ld.w    [%sp+0x0],%r4 000007F8 wW 00000000 00000173
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA 00000000 AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
AAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA
000007F8 00000000 00000000          IL:0 MO:0 DS:0 IE:0 C:0 V:0 Z:0 N:0

```

<各トレース情報の1行目>

Number Address Code Disassemble Address Type Data Clock [File Line SourceCode]

Number: 実行命令No.(10進数)

CPUをリセットまたはトレースをONにしてからの実行命令数

Address: 実行命令アドレス(16進数)

Code: 命令コード(16進数)

Disassemble: 実行命令の逆アセンブル内容

Address: アクセスしたメモリアドレス(16進数)

Type: バスオペレーションタイプ

rB: バイトデータリード、rH: ハーフワードデータリード、rW: ワードデータリード

wB: バイトデータライト、wH: ハーフワードデータライト、wW: ワードデータライト

Data: 読み出し/書き込みデータ(16進数)

Clock: 実行クロック数(10進数)

File: ソースファイル名(c33 tmコマンドでソース表示を選択した場合のみ)

Line: ソースの行番号(c33 tmコマンドでソース表示を選択した場合のみ)

SourceCode: ソースコード(c33 tmコマンドでソース表示を選択した場合のみ)

<各トレース情報の2~4(6)行目>

これらの行は、レジスタの表示を選択した場合にのみ表示されます。
各レジスタ値は次のような配置で表示されます。

C33 STDコアの場合

R0	R1	R2	R3	R4	R5	R6	R7
R8	R9	R10	R11	R12	R13	R14	R15
SP	AHR	ALR		PSR(フラグ個別に表示)			

C33 ADVコアの場合

R0	R1	R2	R3	R4	R5	R6	R7
R8	R9	R10	R11	R12	R13	R14	R15
SP	AHR	ALR		PSR(フラグ個別に表示)			
LCO	LSA	LEA	SOR	TTBR	DP	USP	SSP
PSR(C33 ADVコアの追加フラグを個別に表示)							

注 意 (SIM)

- 表示されるクロック数は、パラメータファイルに設定されたウェイト数情報を使い算出しています。パラメータファイルの設定に誤りがあると正しく表示されません。詳細については、"10.8 パラメータファイル"を参照してください。
- トレースモード(トレース情報表示内容)を変更する場合は、トレースモードを一旦解除(`c33 tm off`を実行)してから再設定してください。

c33 td (PC トレース内容の表示)

[ICD2 / ICD3 / ICD6]

機 能

S5U1C33000H/S5U1C33001Hのトレースメモリ内にあるトレース情報を[トレース]ビューに表示します。[トレース]ビューが閉じている場合でも、本コマンドの実行により開きます。

書 式

c33 td [*StartCycle EndCycle*]

StartCycle: 表示を開始するトレースサイクル番号(10進数または16進数)

EndCycle: 表示を終了するトレースサイクル番号(10進数または16進数)

0が最新データです。*StartCycle*と*EndCycle*を省略すると、10Kクロック分のサイクルを表示します。

条件: S5U1C33000H $0 \leq \text{EndCycle} \leq \text{StartCycle} \leq 131,071$

S5U1C33001H $0 \leq \text{EndCycle} \leq \text{StartCycle} \leq 1,048,575$

使用例**■例1**

(gdb)

c33 td

現在のトレースメモリにあるトレース情報の内、10Kクロック分を[トレース]ビューに表示します。

■例2

(gdb)

c33 td 1000 0

サイクル番号1000から最新までのトレース情報を[トレース]ビューに表示します。

注 意

- 本コマンドはICD2、ICD3またはICD6モード時のみ使用可能です。
- c33 ts**コマンドで検索モードに設定している場合は、指定したサイクル範囲内の検索条件に合ったトレースデータのみが表示されます。
- 本コマンドでトレース情報を取得するには、**c33 rpf**コマンドによるメモリマップ情報の設定が必要です。

なお、**c33 rpf**コマンドは、**target**コマンドより先に実行してください。プログラム実行直前にメモリマップ情報を設定すると、トレース情報を正常に取得できません。

コマンド実行順序の一例を示します。

(gdb)

c33 rpf sample.par (マップ情報設定)

(gdb)

file sample.elf (デバッグ情報の読み込み)

(gdb)

target icd com1 (ターゲットの接続)

(gdb)

load (プログラムのロード)

(gdb)

c33 rsth (ホットリセット)

(gdb)

c33 tm 1 1 (ICDトレースモード設定)

(gdb)

break 13 (ブレークポイント設定)

(gdb)

continue (連続実行)...トレース情報を採取

(gdb)

c33 td (トレース情報の表示)

c33 autotd (PC トレース内容の自動表示)

[ICD2 / ICD3 / ICD6]

機 能

プログラムが停止するたびにトレース結果を[トレース]ビューに表示します。

書 式

c33 autotd Mode [*StartCycle EndCycle*]

Mode: **on** トレース内容の自動表示機能を有効にします。
 表示は *Mode* を *off* にするまで行われ、トレース結果はプログラムが停止するたびに追加表示されます。
 off トレース内容の自動表示機能を無効にします。(デフォルト)

StartCycle: 表示を開始するトレースサイクル番号(10進数または16進数)

EndCycle: 表示を終了するトレースサイクル番号(10進数または16進数)
 0が最新データです。*StartCycle*と*EndCycle*を省略すると、10Kクロック分のサイクルを表示します。

条件: S5U1C33000H $0 \leq \text{EndCycle} \leq \text{StartCycle} \leq 131,071$
 S5U1C33001H $0 \leq \text{EndCycle} \leq \text{StartCycle} \leq 1,048,575$

使用例**■例1**

```
(gdb)
c33 autotd on
```

プログラムが停止するたびにトレース結果を表示します。

■例2

```
(gdb)
c33 autotd on 1000 0
```

プログラムが停止するたびに1000サイクル分表示します。

■例3

```
(gdb)
c33 autotd off
```

プログラムが停止してもトレース結果を表示しません。

注 意

- 本コマンドはICD2、ICD3またはICD6モード時のみ使用可能です。
- c33 ts**コマンドで検索モードに設定している場合は、指定したサイクル範囲内の検索条件に合ったトレースデータのみが表示されます。
- 本コマンドでトレース情報を取得するには、**c33 rpf**コマンドによるメモリマップ情報の設定が必要です。
 なお、**c33 rpf**コマンドは、**target**コマンドより先に実行してください。プログラム実行直前にメモリマップ情報を設定すると、トレース情報を正常に取得できません。
 コマンド実行順序の一例を示します。

```
(gdb)
c33 rpf sample.par       (マップ情報設定)
(gdb)
file sample.elf         (デバッグ情報の読み込み)
(gdb)
target icd6 usb         (ターゲットの接続)
(gdb)
load                    (プログラムのロード)
(gdb)
c33 rsth                (ホットリセット)
```

10 デバッガ

(gdb)	
c33 tm 1 1	(ICDトレースモード設定)
(gdb)	
c33 autotd on	(PCトレース内容の自動表示を有効にする)
(gdb)	
break 13	(ブレークポイント設定)
(gdb)	
continue	
ブレーク発生	…トレース情報の表示
(gdb)	
step	
Step終了	…トレース情報の表示
(gdb)	
step	
Step終了	…トレース情報の表示
(gdb)	
c33 autotd off	(PCトレース内容の自動表示を無効にする)

c33 ts (PC トレース内容の検索)

[ICD2 / ICD3 / ICD6]

機 能

指定アドレスの命令を実行したサイクルのトレース情報のみを表示/保存するための、検索モードを設定します。

書 式

c33 ts *Address* [*PreLine PostLine*]

Address: 実行アドレス(10進数、16進数、またはシンボル)

0xffffffffを指定すると、検索モードが解除されます。

PreLine: 検索された行より前の表示/保存する行数(10進数または16進数)

PostLine: 検索された行より後の表示/保存する行数(10進数または16進数)

条件: $0 \leq \text{Address} \leq 0xffffffff$ 、 $0 \leq \text{PreLine} \leq 256$ 、 $0 \leq \text{PostLine} \leq 256$

使用例**例1**

```
(gdb)
c33 ts 0x20100 5 5
```

0x20100番地の命令を実行したサイクルと、その前後5サイクルずつのトレース情報のみを表示/保存する検索モードに設定します。

例2

```
(gdb)
c33 ts 0xffffffff
```

検索モードを解除します。

注 意

- 本コマンドはICD2、ICD3またはICD6モード時のみ使用可能です。
- 本コマンドでトレース情報を取得するには、**c33 rpf**コマンドによるメモリマップ情報の設定が必要です。

なお、**c33 rpf**コマンドは、**target**コマンドより先に実行してください。プログラム実行直前にメモリマップ情報を設定すると、トレース情報を正常に取得できません。

コマンド実行順序の一例を示します。

```
(gdb)
c33 rpf sample.par           (マップ情報設定)
(gdb)
file sample.elf             (デバッグ情報の読み込み)
(gdb)
target icd com1             (ターゲットの接続)
(gdb)
load                         (プログラムのロード)
(gdb)
c33 rsth                     (ホットリセット)
(gdb)
c33 tm 1 1                   (ICDトレースモード設定)
(gdb)
break 13                     (ブレークポイント設定)
(gdb)
continue                     (連続実行)...トレース情報を採取
(gdb)
c33 ts 0x20100 5 5          (トレース情報の検索)
```

c33 tf (PC トレース内容の保存)

[ICD2 / ICD3 / ICD6]

機 能

S5U1C33000H/S5U1C33001Hのトレースメモリ内にあるトレース情報をファイルに保存します。保存される内容は、c33 tdコマンドで表示される内容と同じです。

書 式

c33 tf *Filename* [*StartCycle* *EndCycle*]

Filename: トレース情報を保存するファイル名

StartCycle: 保存を開始するトレースサイクル番号(10進数または16進数)

EndCycle: 保存を終了するトレースサイクル番号(10進数または16進数)

0が最新データです。*StartCycle*と*EndCycle*を省略すると、10Kクロック分のサイクルを保存します。

条件: S5U1C33000H $0 \leq \text{EndCycle} \leq \text{StartCycle} \leq 131,071$

S5U1C33001H $0 \leq \text{EndCycle} \leq \text{StartCycle} \leq 1,048,575$

使用例**■例1**

(gdb)

c33 tf trace.log

現在のトレースメモリにあるトレース情報の内、10Kクロック分をtrace.logファイルに保存します。

■例2

(gdb)

c33 tf trace.log 1000 0

サイクル番号1000から最新までのトレース情報をtrace.logファイルに保存します。

注 意

- 本コマンドはICD2、ICD3またはICD6モード時のみ使用可能です。
- c33 tsコマンドで検索モードに設定している場合は、指定したサイクル範囲内の検索条件に合ったトレースデータのみがファイルに書き出されます。
- 既存のファイルを指定すると、データを上書きします。
- 本コマンドでトレース情報を取得するには、c33 rpfコマンドによるメモリマップ情報の設定が必要です。
なお、c33 rpfコマンドは、targetコマンドより先に実行してください。プログラム実行直前にメモリマップ情報を設定すると、トレース情報を正常に取得できません。
コマンド実行順序の一例を示します。

(gdb)

c33 rpf sample.par (マップ情報設定)

(gdb)

file sample.elf (デバッグ情報の読み込み)

(gdb)

target icd com1 (ターゲットの接続)

(gdb)

load (プログラムのロード)

(gdb)

c33 rsth (ホトリセット)

(gdb)

c33 tm 1 1 (ICDトレースモード設定)

(gdb)

break 13 (ブレークポイント設定)

(gdb)

continue (連続実行)...トレース情報を採取

(gdb)

c33 tf trace.log

(トレース情報の保存)

- S5U1C33001Hはトレースメモリが大きいので、全トレースデータのファイル化にはかなりの時間が必要です。サイクル番号を指定するか、検索モードを設定して保存することをお勧めします。

c33 autotf (PC トレース内容の自動保存)

[ICD2 / ICD3 / ICD6]

機 能

プログラムが停止するたびにファイルへトレース結果を自動的に保存します。

書 式

c33 autotf Mode [Filename [StartCycle EndCycle]]

Mode: **on** トレース内容の自動保存機能を有効にします。
 ファイルへの保存は **Mode** を **off** にするまで行われ、トレース結果はプログラムが停止する
 たびに追加保存されます。
 off トレース内容の自動保存機能を無効にします。(デフォルト)

Filename: トレース情報を保存するファイル名

StartCycle: 保存を開始するトレースサイクル番号(10進数または16進数)

EndCycle: 保存を終了するトレースサイクル番号(10進数または16進数)
 0が最新データです。**StartCycle**と**EndCycle**を省略すると、10Kクロック分のサイクルを表示
 します。

条件: S5U1C33000H $0 \leq \text{EndCycle} \leq \text{StartCycle} \leq 131,071$
 S5U1C33001H $0 \leq \text{EndCycle} \leq \text{StartCycle} \leq 1,048,575$

使用例**■例1**

```
(gdb)
c33 autotf on trace.log
```

プログラムが停止するたびにトレース結果を保存します。

■例2

```
(gdb)
c33 autotf on trace.log 1000 0
```

プログラムが停止するたびに1000サイクル分保存します。

■例3

```
(gdb)
c33 autotf off
```

プログラムが停止してもトレース結果を保存しません。

注 意

- 本コマンドはICD2、ICD3またはICD6モード時のみ使用可能です。
- c33 ts**コマンドで検索モードに設定している場合は、指定したサイクル範囲内の検索条件に合ったトレースデータのみがファイルに書き出されます。
- 既存のファイルを指定すると、データを上書きします。
- 本コマンドでトレース情報を取得するには、**c33 rpf**コマンドによるメモリマップ情報の設定が必要です。
 なお、**c33 rpf**コマンドは、**target**コマンドより先に実行してください。プログラム実行直前にメモリマップ情報を設定すると、トレース情報を正常に取得できません。
 コマンド実行順序の一例を示します。

```
(gdb)
c33 rpf sample.par                   (マップ情報設定)
(gdb)
file sample.elf                   (デバッグ情報の読み込み)
(gdb)
target icd6 usb                   (ターゲットの接続)
(gdb)
load   (プログラムのロード)
(gdb)
```


<code>c33 rsth</code>	(ホットリセット)
<code>(gdb)</code>	
<code>c33 tm 1 1</code>	(ICDトレースモード設定)
<code>(gdb)</code>	
<code>c33 autotf on trace.log</code>	(PCトレース内容の自動保存を有効にする)
<code>(gdb)</code>	
<code>break 13</code>	(ブレークポイント設定)
<code>(gdb)</code>	
<code>continue</code>	
ブレーク発生	…トレース情報の表示
<code>(gdb)</code>	
<code>step</code>	
Step終了	…トレース情報の表示
<code>(gdb)</code>	
<code>step</code>	
Step終了	…トレース情報の表示
<code>(gdb)</code>	
<code>c33 autotd off</code>	(PCトレース内容の自動保存を無効にする)
- S5U1C33001Hはトレースメモリが大きいいため、全トレースデータのファイル化にはかなりの時間が必要です。サイクル番号を指定するか、検索モードを設定して保存することをお勧めします。

c33 obt (オンチップバストレース設定)

[ICD3 / ICD6]

機 能

バストレースを設定します。バストレースは、指定したバス上で設定した条件のアクセスが発生するとその情報をトレースメモリに取得する機能です。

本コマンドは、ターゲットCPUがC33 ADVコアの場合のみ有効です。

バストレース機能は、**gdb**起動時に無効となっています。バストレースを有効にするには、本コマンドのModeを0か1に設定して実行してください。

書 式

**c33 obt Mode Bus BBCU_BusMaster R/W Inst/Data Access BusOutput AddressBusSize
Compress UserInput**

Mode: トレースモード選択
 0 通常モードでトレース機能ON
 1 ユーザモードでトレース機能ON
 2 トレース機能OFF(デフォルト)

Bus: バス選択
 0 CPU系データバス(デフォルト)
 1 CPU系命令バス
 2 BBCU系バス
 Modeがユーザモードの場合は無効です。0を設定してください。

BBCU_BusMaster: BBCUバスマスタ選択(10進数または16進数)
 選択するビットを1にします。(Bus = 2の場合、1つ以上選択すること。複数指定可)
 bit3 = 1: USER(CPU)によるメモリアクセス(デフォルト)
 bit2 = 1: DMAによるメモリアクセス(デフォルト)
 bit1 = 1: キャッシュによるリフィル/ライトバック時のメモリアクセス(デフォルト)
 bit0 = 1: CPUのメモリアクセス(デフォルト)
 Modeがユーザモードの場合は無効です。0を設定してください。

R/W: リード/ライト条件
 r リード
 w ライト
 rw リード/ライト(デフォルト)

Inst/Data: 命令/データ条件
 0 命令
 1 データ
 2 命令/データ(デフォルト)

Access: アクセス条件(10進数または16進数、複数指定可)
 bit2 = 1: ワードアクセス(Word) (デフォルト)
 bit1 = 1: ハーフワードアクセス(Half) (デフォルト)
 bit0 = 1: バイトアクセス(Byte) (デフォルト)

BusOutput: バス出力条件
 0 アドレスバス
 1 データバス
 2 アドレスバス/データバス(デフォルト)

AddressBusSize: アドレスバス幅
 バス出力条件でアドレスバスを有効とした場合に出力されるアドレス情報のバイト数(下位から何バイトか)を指定します。指定範囲は1~4バイト(デフォルト = 4)です。バス出力条件でアドレス出力が無効の場合でも値を設定してください。ただし、設定した値は無効になります。

Compress: 圧縮出力
 0 圧縮しません。
 1 アドレスおよびデータの圧縮出力を行います。(デフォルト)

UserInput: ユーザ入力(ブレイクトリガブレイク)
 0 外部トレース信号入力(TRC IN) (デフォルト)
 1 ターゲットCPUトリガ信号入力(DBT: デバッグブレイクトリガ信号)

使用例

```
(gdb)
c33 obt 0 2 0x8 rw 2 0x04 0 4 0 0
Set bus trace
Trace mode           : Normal mode
Bus select           : BBCU
BBCU bus master select : USER
Read/Write           : Read/Write
Instruction/Data      : Instruction/Data
Access size          : Word
Bus output            : Address bus
Address bus size      : 4 byte
Compression          : Disable
Bus trace trigger select : TRC IN signal
```

トレース条件をUSERによるワードアクセス、アドレスバス幅を4バイト、圧縮出力はしない設定にします。また、ユーザ入力(ブレークトリガブレーク)のトレース信号として外部トレース入力信号(TRC IN信号)を指定します。

注 意

- c33 obtコマンドはICD3またはICD6モード時のみ有効です。
- ICDがトレース無効の場合、またはターゲットにオンチップバストレース機能がない場合は、エラーになります。
- 本コマンドは、バスブレークと同じ設定にする必要があります。
バスブレークとバストレースのバス選択で異なる設定をした場合、後に実行した設定が有効になります。

c33 otd (オンチップバストレース内容の表示)

[ICD3 / ICD6]

機能

オンチップバストレースで取得した内容を[トレース]ビューに表示します。

書式

c33 otd Mode [StartCycle EndCycle]

Mode: 出力モード選択

- 0 バストレース情報のみ出力
- 1 PCトレースとバストレース情報を出力

StartCycle: 表示を開始するトレースサイクル番号(10進数または16進数)

EndCycle: 表示を終了するトレースサイクル番号(10進数または16進数)

0が最新データです。**StartCycle**と**EndCycle**を省略すると、10Kクロック分のサイクルを表示します。

条件: $0 \leq \text{EndCycle} \leq \text{StartCycle} \leq 1,048,574$

使用例

例1

(gdb)

c33 otd 0

現在のトレースメモリにあるバストレース情報の内、10Kクロック分を[トレース]ビューに表示します。

例2

(gdb)

c33 otd 1 1000 0

サイクル番号1000から最新までのPCトレースとバストレース情報を[トレース]ビューに表示します。

表示

Mode = 0の場合

• 通常モードの出力例

Cycle	Clk	BM	Address	Data	I/D	RW	SZ	UIN	Unassemble
0000102	0000006	CPU		00000000	i	r	W	0	nop
									nop
0000101	0000020	CPU	006002240		i	r	W	0	
0000100	0000026	CPU	006000244		i	r	W	0	
0000099	0000034	CPU	006000248		i	r	W	0	
:	:	:	:	:	:	:	:	:	:

トレースデータの存在するCycleのみを出力します。

• ユーザモードの出力例

Cycle	Clk	DTD	DTS	UIN
0009999	0000000	00000000	00000	0
0009998	0000001	00000000	00000	0
0009997	0000002	00000000	00000	0
0009996	0000003	00000000	00000	0
:	:	:	:	:

Mode = 1の場合

• 通常モードの出力例

Cycle	Address	Code	Unassemble	Clk	BM	Address	Data	I/D	RW	SZ	UIN	...
0000012	0060039E	FC04	ld.w [%dp+0x0], %r4	0000000								...
				0000005	CPU	006003a8		i	r	W	1	...
				0000007								...
0000011	006003A0	2E40	ld.w %r0, %r4	0000013	CPU	005003ac		i	r	W	1	...
				0000021	CPU	005003b0		i	r	W	1	...
:	:	:	:	:	:	:	:	:	:	:	:	...

• ユーザモードの出力例

Cycle	Address	Code	Unassemble	Clk	DTD	DTS	UIN	Method	File	...
0000012	0060039E	FC04	ld.w [%dp+0x0],%r4	0000000	10000000	00000	0	SPC		...
				0000001	10000000	00000	0			...
				0000002	10100000	00000	1			...
				0000003	11100000	00001	1			...
:	:	:	:	:	:	:	:	:	:	...

BM: バスの種類

CPU バスマスタは、CPU(HBCU設定時)

CPU/CACHE バスマスタは、CPUまたはCACHE(BBCU設定時)

DMA/USR バスマスタは、DMAまたはUSR(BBCU設定時)

Address: アクセスアドレス

Data: アクセスデータ

I/D: データの種類

i 命令

d データ

RW: リード/ライト

w ライト

r リード

SZ: アクセスサイズ

B バイト

H ハーフワード

W ワード

UIN: USER_IN信号の状態

0または1

<Address/Data表記>

*: 不明値(アドレスとデータ)

+: 不明だが、前サイクルと同じ値(アドレスのみ)

<命令データバスのビット配置の見方>

命令バスは2命令同時フェッチです。

ABCDEFGH

A = I2[15:12]

B = I2[11: 8]

C = I2[7: 4]

D = I2[3: 0]

E = I1[15:12]

F = I1[11: 8]

G = I1[7: 4]

H = I1[3: 0]

注 意

- c33 ot_dコマンドはICD3またはICD6モード時のみ有効です。
- C33 STDまたはC33 PEコアのターゲットで本コマンドを使用するには、その前にc33 logianaコマンドを実行する必要があります。この場合、バストレース情報はユーザモードの出力のみとなります。

c33 otf (オンチップバストレース内容の保存)

[ICD3 / ICD6]

機 能

オンチップバストレースで取得した内容をファイルに保存します。出力内容はc33 otdコマンドと同様です。

書 式

c33 otf *Mode Filename [StartCycle EndCycle]*

Mode: 出力モード選択

0 バストレース情報のみ出力

1 PCトレースとバストレース情報を出力

Filename: トレース情報を保存するファイル名

StartCycle: 保存を開始するトレースサイクル番号(10進数または16進数)

EndCycle: 保存を終了するトレースサイクル番号(10進数または16進数)

0が最新データです。*StartCycle*と*EndCycle*を省略すると、10Kクロック分のサイクルを保存します。

条件: $0 \leq EndCycle \leq StartCycle \leq 1,048,574$

使用例**■例1**

(gdb)

```
c33 otf 0 trace.log
```

現在のトレースメモリにあるバストレース情報の内、10Kクロック分をtrace.logファイルに保存します。

■例2

(gdb)

```
c33 otf 1 trace.log 1000 0
```

サイクル番号1000から最新までのPCトレースとバストレース情報を、trace.logファイルに保存します。

注 意

- c33 otfコマンドはICD3またはICD6モード時のみ有効です。
- C33 STDまたはC33 PEコアのターゲットで本コマンドを使用するには、その前にc33 logianaコマンドを実行する必要があります。この場合、バストレース情報はユーザモードの出力のみとなります。

10.7.14 シミュレーテッドI/Oコマンド

c33 stdin (データ入力シミュレーション)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機能

データをファイルから入力して、プログラムに渡すための設定を行います。

c33 stdinコマンドでは、以下の条件を設定します。

- ブレークアドレス (gdbがデータを取り込む位置)
- 入力バッファアドレス (65バイトのバッファ)
- 入力対象 (ファイル)

プログラム側での対応については、"10.6.7 シミュレーテッドI/O"を参照してください。

書式

c33 stdin 1 BreakAddr BufferAddr [Filename] (設定)

c33 stdin 2 (解除)

BreakAddr: ブレークアドレス(10進数、16進数、またはシンボル)

BufferAddr: 入力バッファアドレス(10進数、16進数、またはシンボル)
バッファサイズは65バイト固定です。

Filename: 入力ファイル名(省略不可)

条件: $0 \leq \text{BreakAddr} \leq 0\text{xffffffff}$ 、 $0 \leq \text{BufferAddr} \leq 0\text{xffffffff}$

入力例

■例1

(gdb)

```
c33 stdin 1 READ_FLASH READ_BUF input.txt
```

ファイルからのデータ入力機能を設定します。

この設定の後でプログラムを連続実行させると、デバッガはプログラム内のラベルREAD_FLASHの位置で実行を中断します。そこでinput.txtファイルから1行分のデータを入力バッファ (READ_BUF)に取り込み、プログラムの実行を再開します。

■例2

(gdb)

```
c33 stdin 2
```

データ入力機能を解除します。ファイル入力の場合、指定ファイルは閉じられます。

注意

- c33 stdinコマンドで設定するブレークアドレスは、ソフトウェアPCブレークと重複することはできません。ソフトウェアPCブレークを解除してからc33 stdinコマンドを実行してください。ハードウェアPCブレークポイントとの重複は許されます。
- ブレークおよびバッファアドレスは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。また、ブレークアドレスを奇数の値で指定すると最下位ビットを0として16ビット境界に補正されます。
- *Filename*を省略するとエラーになり使用できません。

c33 stdout (データ出力シミュレーション)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

指定の出力バッファのデータをファイルまたは[コンソール]ビューに出力するための設定を行います。

c33 stdoutコマンドでは、以下の条件を設定します。

- ブレークアドレス (gdbがデータを出力する位置)
- 出力バッファアドレス (65バイトのバッファ)
- 出力対象 (ファイルまたは[コンソール]ビュー)

プログラム側での対応については、"10.6.7 シミュレートッドI/O"を参照してください。

書 式

c33 stdout 1 BreakAddr BufferAddr [Filename] (設定)

c33 stdout 2 (解除)

BreakAddr: ブレークアドレス(10進数、16進数、またはシンボル)

BufferAddr: 出力バッファアドレス(10進数、16進数、またはシンボル)
バッファサイズは65バイト固定です。

Filename: 出力ファイル名

ファイルと[コンソール]ビューへの出力をします。

省略すると、[コンソール]ビューへの出力となります。

条件: $0 \leq \text{BreakAddr} \leq 0\text{xffffffff}$ 、 $0 \leq \text{BufferAddr} \leq 0\text{xffffffff}$

入力例**■例1**

(gdb)

```
c33 stdout 1 WRITE_FLASH WRITE_BUF output.txt
```

ファイルへのデータ出力機能を設定します。

この設定の後でプログラムを連続実行させると、デバッグはプログラム内のラベルWRITE_FLASHの位置で実行を中断します。ここで、指定のバッファ(WRITE_BUF)内のデータを指定のファイルに出力し、プログラムの実行を再開します。

■例2

(gdb)

```
c33 stdout 1 WRITE_FLASH WRITE_BUF
```

[コンソール]ビューを使用するデータ出力機能を設定します。

この設定の後でプログラムを連続実行させると、デバッグはプログラム内のラベルWRITE_FLASHの位置で実行を中断します。ここで、[コンソール]ビューを開き、指定のバッファ(WRITE_BUF)内のデータをウィンドウに表示してプログラムの実行を再開します。

■例3

(gdb)

```
c33 stdout 2
```

データ出力機能を解除します。ファイル出力の場合、指定ファイルは閉じられます。

注 意

- c33 stdoutコマンドで設定するブレークアドレスは、ソフトウェアPCブレークと重複することはできません。ソフトウェアPCブレークを解除してからc33 stdoutコマンドを実行してください。ハードウェアPCブレークポイントとの重複は許されます。
- ブレークおよびバッファアドレスは下位32ビットのみが有効で、それを超えた分は無視されます。たとえば、0x100000000は0x00000000として処理されます。また、ブレークアドレスを奇数の値で指定すると最下位ビットを0として16ビット境界に補正されます。
- printf()関数を実行すると、[コンソール]ビュー表示が遅くなることがあります。このようなとき、C/C++コンパイラオプション-fno-builtinを付加すると速くなることがあります。

10.7.15 フラッシュライタコマンド

c33 fwe (プログラム / データの消去)

[ICD3 / ICD6]

機 能

S5U1C33001Hのフラッシュライタ機能用のコマンドです。

S5U1C33001Hにロードしたデータ消去/書き込みプログラム、または書き込みデータとアドレス情報を消去します。

書 式

c33 fwe 0 (書き込みデータの消去)

c33 fwe 1 (データ消去 / 書き込みプログラムの消去)

使用例

■例1

(gdb)

c33 fwe 0

S5U1C33001H内のフラッシュ書き込みデータ格納領域とアドレス情報を消去します。

■例2

(gdb)

c33 fwe 1

S5U1C33001H内のフラッシュ消去/書き込みプログラム格納領域とエントリ情報を消去します。

注 意

本コマンドはICD3またはICD6モード以外では使用できません。

c33 fwlp (プログラムのロード)

[ICD3 / ICD6]

機 能

S5U1C33001Hのフラッシュライター機能用のコマンドです。
データ消去/書き込みプログラムをホストからS5U1C33001Hにロードします。

書 式

c33 fwlp *Filename EraseEntryAddr WriteEntryAddr* [*Comment*]

Filename: データ消去/書き込みプログラムファイル名(モトローラS3形式ファイル)
EraseEntryAddr: 消去ルーチンエントリアドレス(CPU側RAMアドレス、10進数または16進数)
WriteEntryAddr: 書き込みルーチンエントリアドレス(CPU側RAMアドレス、10進数または16進数)
Comment: データ/アドレス情報識別コメント(省略可)
 空白文字(スペース)を含むときは、ダブルクォーテーションで囲みます。
 条件: $0 \leq \text{EraseEntryAddr} \leq 0\text{xffffffe}$ ($A0 = 0$)、 $0 \leq \text{WriteEntryAddr} \leq 0\text{xffffffe}$ ($A0 = 0$)、
 $0 \leq \text{コメントサイズ} \leq 128$ バイト

使用例

(gdb)

c33 fwlp writer.sa 0x90 0xb4

*writer.sa*の名称で用意されているフラッシュライター機能用のデータ消去/書き込みプログラムをS5U1C33001Hにロードし、消去ルーチンの開始アドレスを0x90、書き込みルーチンの開始アドレスを0xb4に設定します。

注 意

- 本コマンドはICD3またはICD6モード以外では使用できません。
- データ消去/書き込みプログラムのサイズは、ICD Ver.3/Ver.4(ICD3モード)は8KB以下、ICD Ver.6(ICD6モード)は64KB以下としてください。

c33 fwld (データのロード)

[ICD3 / ICD6]

機 能

S5U1C33001Hのフラッシュライター機能用のコマンドです。
フラッシュに書き込むデータをホストからS5U1C33001Hにロードします。

書 式

c33 fwld *Filename EraseStartBlock EraseEndBlock EraseParam* [*Comment*]

Filename: データファイル名(モトローラS3形式ファイル)

EraseStartBlock: 消去開始ブロック(CPU側フラッシュ、10進数または16進数)

EraseEndBlock: 消去終了ブロック(CPU側フラッシュ、10進数または16進数)

EraseParam: 消去パラメータ

外部ルーチン呼び出しで、消去ルーチンに引き渡されるパラメータ

Comment: データ/アドレス情報識別コメント(省略可)

空白文字(スペース)を含むときは、ダブルクォーテーションで囲みます。

条件: $0 \leq \text{EraseStartBlock} \leq 0\text{xffffffff}$ 、 $0 \leq \text{EraseEndBlock} \leq 0\text{xffffffff}$

$\text{EraseStartBlock} = \text{EraseEndBlock} = 0$ の場合はフラッシュ全ブロックの消去に設定されます。

$0 \leq \text{コメントサイズ} \leq 128$ バイト

使用例

(gdb)

c33 fwld sample.sa 0 0 0x200000

フラッシュの消去範囲(フラッシュ全ブロック)を設定し、`sample.sa`の名称で用意されているフラッシュ書き込みデータをS5U1C33001Hにロードします。

注 意

- 本コマンドはICD3またはICD6モード以外では使用できません。
- プログラムのサイズは、ICD Ver.3/Ver.4(ICD3モード)は8MB以下、ICD Ver.6(ICD6モード)は4MB以下です。

c33 fwdc (ターゲットメモリのコピー)

[ICD3 / ICD6]

機 能

S5U1C33001Hのフラッシュライタ機能用のコマンドです。
 ターゲットボード上のメモリに格納されているデータを、フラッシュに書き込むために
 S5U1C33001Hにロードします。

書 式

c33 fwdc *SourceAddr* *Size* *EraseStartBlock* *EraseEndBlock* *EraseParam* [*Comment*]

SourceAddr: コピー元ターゲットメモリアドレス
 (CPU側フラッシュ、10進数、16進数、またはシンボル)
Size: コピーバイト数(10進数または16進数)
EraseStartBlock: 消去開始ブロック(CPU側フラッシュ、10進数または16進数)
EraseEndBlock: 消去終了ブロック(CPU側フラッシュ、10進数または16進数)
EraseParam: 消去パラメータ
 外部ルーチン呼び出しで、消去ルーチンに引き渡されるパラメータ
Comment: データ/アドレス情報識別コメント(省略可)
 空白文字(スペース)を含むときは、ダブルクォーテーションで囲みます。
 条件: $0 \leq \text{SourceAddr} \leq 0\text{xffffffffe}$ ($A0 = 0$)、 $0 \leq \text{Size} \leq 0\text{xffffffffe}$ ($D0 = 0$)、
 $0 \leq \text{EraseStartBlock} \leq 0\text{xffffffff}$ 、 $0 \leq \text{EraseEndBlock} \leq 0\text{xffffffff}$
 $\text{EraseStartBlock} = \text{EraseEndBlock} = 0$ の場合はフラッシュ全ブロックの消去に設定され
 ます。
 $0 \leq \text{コメントサイズ} \leq 128$ バイト

使用例

(gdb)

```
c33 fwdc FLASH_START 0x100000 0 0 0x200000
```

フラッシュの消去範囲(フラッシュ全ブロック)を設定し、ターゲットメモリ上のFLASH_STARTから
 1Mバイトの領域をS5U1C33001Hにコピーします。

注 意

- 本コマンドはICD3またはICD6モード以外では使用できません。
- プログラムのサイズは、ICD Ver.3/Ver.4(ICD3モード)は8MB以下、ICD Ver.6(ICD6モード)は4MB以下です。

c33 fwd (フラッシュライタ情報の表示)

[ICD3 / ICD6]

機 能

S5U1C33001Hのフラッシュライタ機能用のコマンドです。

S5U1C33001Hにロードされているデータの情報、および消去/書き込みプログラムの情報を表示します。

書 式

c33 fwd

表 示

```
(gdb)
c33 fwd
CPU data address          : xxxxxxxx
Data size                 : xxxxxxxx
Erase start block        : xxxxxxxx
Erase end block          : xxxxxxxx
Erase parameter          : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

CPU program address       : xxxxxxxx
Program size              : xxxxxxxx
Erase routine entry address : xxxxxxxx
Write routine entry address : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

前半は、フラッシュ書き込みデータのアドレスとサイズ、フラッシュの消去範囲の情報で、**c33 fwd**または**c33 fwdc**コマンドで設定された内容です。

後半は、フラッシュ消去/書き込みプログラムの先頭アドレスとサイズ、消去/書き込みルーチンのエントリ情報で、**c33 fwdlp**コマンドで設定された内容です。

注 意

本コマンドはICD3またはICD6モード以外では使用できません。

10.7.16 プロファイラ・カバレッジコマンド

c33 profilemd (プログラム・カバレッジモードの設定)

[SIM]

機 能

ユーザプログラムの実行中に、プロファイル・カバレッジデータの取得処理を行うかどうか設定します。

本コマンドは、プロファイラ機能を使用しない場合で実行速度が遅いときに、モードを無効にし、実行速度を少しでも短縮したいときに実行します。

書 式**c33 profilemd Mode**

Mode: 0 以後プロファイル・カバレッジデータの取得を無効にします。無効にする前のデータは保持されています。

1 以後プロファイル・カバレッジデータの取得を有効にします(デフォルト)。

省略時には、状態を表示します。無効のとき、"Profiler mode is disabled."を、有効のとき、"Profiler mode is enabled."を表示します。

パラメータが2つ以上ある場合、"C33 command error, number of parameter."と表示されます。また、0、1以外のパラメータを付加すると、"C33 command error, invalid parameter."と表示されます。

使用例

```
(gdb)
c33 profilemd 0
Profiler mode is disabled.
(gdb)
c33 profilemd 1
Profiler mode is enabled.
(gdb)
c33 profilemd
Profiler mode is enabled.
```

注 意

以下のようにモードの設定を切り替えると、正しくプロファイルは計測できません。途中で有効、無効を切り替えないようにしてください。

モード：有効状態

```
(gdb)
C33 rst
(gdb)
Cont
    Break 発生
(gdb)
C33 profilemd 0
(gdb)
Cont
    Break 発生
(gdb)
C33 profilemd 1
(gdb)
Cont
    Break 発生
(gdb)
C33 profile ←ここでプロファイル計測は正しく行われません。
```

} この間の計測データがない。

c33 profile (プロファイラウィンドウの起動)

[SIM]

機 能

プロファイラウィンドウを開き、プロファイル結果の表示を行います。プロファイラウィンドウの詳細は、"10.6.10 プロファイラ・カバレッジ機能"を参照してください。

書 式

```
c33 profile
```

使用例

```
(gdb)
c33 rpf c33.par
(gdb)
file sample.elf
(gdb)
target sim
(gdb)
load
(gdb)
c33 rst
(gdb)
break _exit
(gdb)
cont                ← 計測開始
Break 発生          ← 計測停止
(gdb)
c33 profile          ← プロファイラウィンドウを開く。
```

注 意

- SIMモードのときプロファイラウィンドウを開き、実行サイクル数が0のとき計測結果は表示しません。実行サイクル数>0のとき計測結果を表示します。
- ICDモードのときエラーになります。
- 既にプロファイラウィンドウが開いているときは、2重には開きません。また、表示内容の更新も行いません。
- c33 profilemdコマンドでプロファイル機能を無効にしているときは、計測データがあってもプロファイラウィンドウを開きますが、計測結果は表示しません。

c33 coverage (カバレッジウィンドウの起動)

[SIM]

機 能

カバレッジウィンドウを開き、カバレッジ結果の表示を行います。カバレッジウィンドウの詳細は、"10.6.10 プロファイラ・カバレッジ機能"を参照してください。

書 式

c33 coverage

使用例

```
(gdb)
c33 rpf c33.par
(gdb)
file sample.elf
(gdb)
target sim
(gdb)
load
(gdb)
c33 rstc
(gdb)
break _exit
(gdb)
cont                ← 計測開始
Break 発生          ← 計測停止
(gdb)
c33 coverage        ← カバレッジウィンドウを開く。
```

注 意

- SIMモードのときカバレッジウィンドウを開き、実行サイクル数が0のとき計測結果は表示しません。実行サイクル数>0のとき計測結果を表示します。
- ICDモードのときエラーになります。
- 既にカバレッジウィンドウが開いているときは、2重には開きません。また、表示内容の更新も行いません。
- c33 profilemdコマンドでプロファイル機能を無効にしているときは、計測データがあってもカバレッジウィンドウを開きますが、計測結果は表示しません。

10.7.17 その他のコマンド

c33 log (ロギング)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

入力したコマンドと、[コンソール]ビューに表示されるコマンド実行結果をファイルに保存します。ログファイルには[コンソール]ビューに表示された内容がそのまま書き込まれます。また、[コンソール]ビュー上には現れない、メニューやその他の操作で実行したコマンドの内容も出力されます。

書 式

c33 log *Filename* (ロギング開始)
c33 log (ロギング終了)

Filename: ログファイル名

使用例

■例1

```
(gdb)
c33 log log.txt
log on
```

これ以降のコマンド入力と実行結果がlog.txtにテキスト形式で出力されます。
 ログの出力は次回のコマンド c33 log の実行まで有効です。

■例2

```
(gdb)
c33 log
log off
```

ログファイルを閉じて、ログ出力を終了します。

注 意

- 既存のファイル名が指定された場合、c33 log コマンドはファイルを上書きします。
- ログ出力を他のファイルに変更する場合は、一度ログ出力を終了し、新しいファイル名で再度ログ出力を開始してください。
- 起動オプション -x で指定するコマンドファイル内の c33 log コマンドは、[コンソール]ビューを開いた状態でのみ有効です。c33 log コマンドの実行前に[コンソール]ビューを閉じると、ログファイルは作成されませんので注意してください。

source (コマンドファイルの実行)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

コマンドファイルを読み込み、その中に記述されたデバッグコマンドを連続実行します。

書 式

source *Filename*

Filename: コマンドファイル名

使用例

```

ファイル名 = gwb33.cmd
# gwb33.cmd command file made by GWB33
# load symbol information
file /cygdrive/c/EPSON/gnu33/sample/tst/sample.elf
#decide debugger mode and its port
target sim
# load to memory
load /cygdrive/c/EPSON/gnu33/sample/tst/sample.elf
# set ttbr register
set {char}0x4812d=0x59
set {long}0x48134=0x00C00000
# hot reset
c33 rsth

```

#から行の終わりまではコメントとみなされます。

```

(gdb)
source gwb33.cmd
(gdb)
(gdb)
(gdb)
file /cygdrive/c/EPSON/gnu33/sample/tst/sample.elf
(gdb)
(gdb)
target sim
boot () at ./boot.s:9
Connected to the simulator.
Current language: auto; currently asm
(gdb)
(gdb)
load /cygdrive/c/EPSON/gnu33/sample/tst/sample.elf
Loading section .text, size 0xbc lma 0xc00000
Start address 0xc00000
Transfer rate: 1504 bits in <1 sec.
(gdb)
(gdb)
set {char}0x4812d=0x59
(gdb)
set {long}0x48134=0x00C00000
(gdb)
(gdb)
c33 rsth
CPU hot resetting ..... done

```

指定のコマンドファイルを読み込み、連続的に実行します。コマンドは例のように[コンソール]ビューに表示されます。

注 意

- コマンドファイル内の記述に誤りがあると、そこでコマンドファイルの実行を中止します。この場合、エラーメッセージは表示されませんので、コマンドファイルは注意して作成してください。
- `source`コマンドも他のコマンドと同様に、一度実行後は[Enter]キーによって繰り返し実行できます。この場合、コマンドファイルに記述された全コマンドが繰り返し実行されます。

c33 clockmd (実行カウンタモード設定)**c33 clock** (実行カウンタ表示)

[ICD2 / ICD3 / ICD6 / SIM]

機 能

c33 clockmd: 実行カウンタのモード設定を行います。設定する内容は次の2項目です。

カウンタモード

積算モード(カウンタがリセットされるまで測定値を積算)、あるいはリセットモード(プログラムの実行ごとにカウンタをリセット)に設定できます。

測定単位

ICD2/ICD3/ICD6モードでは、S5U1C33000H/S5U1C33001Hの実行カウンタの測定単位を、サイクル数、 μ s単位、秒単位から選択できます。

シミュレータモードの実行カウンタは、サイクル数のカウントのみを行います。

デバッグモニタモードには、この測定機能はありません。

c33 clock: プログラムの実行によりカウントされた結果を表示します。

実行カウンタの詳細については、"10.6.4 プログラムの実行"内の"●実行サイクル/実行時間の測定"を参照してください。

書 式

c33 clockmd Mode Function (実行カウンタモード設定)

c33 clock (実行カウンタ表示)

Mode: カウンタモード

- 1 リセットモード
- 2 積算モード(デフォルト)

Function: 測定単位

- 1 サイクル数(デフォルト) ※シミュレータモードでは1のみ指定可能です。
- 2 μ s単位
- 3 秒単位

使用例**■例1**

```
(gdb)
c33 clockmd 2 1
(gdb)
c33 rsth
CPU hot resetting ..... done
(gdb)
continue
Continuing.

Breakpoint 1, sub (k=0) at ./main.c:20
(gdb)
c33 clock
      218 cycle
(gdb)
continue
Continuing.

Breakpoint 1, sub (k=1) at ./main.c:20
(gdb)
c33 clock
      330 cycle
```

積算モードでサイクル数の測定に設定後、ホトリセットで実行カウンタをリセットし、プログラムを実行します。積算モードでは、ブレーク後に実行を再開しても実行カウンタはリセットされません。

■例2

```

(gdb)
c33 clockmd 1 1
(gdb)
c33 rsth
CPU hot resetting ..... done
(gdb)
continue

Breakpoint 1, sub (k=0) at ./main.c:20
(gdb)
c33 clock
                218 cycle
(gdb)
continue
Continuing.

Breakpoint 1, sub (k=1) at ./main.c:20
(gdb)
c33 clock
                112 cycle

```

こちらの例ではリセットモードでサイクル数の測定に設定後、ホトリセットしてプログラムを実行します。リセットモードでは、ブレーク後に実行を再開する時点で実行カウンタがリセットされるため、積算モードの例とはカウント値が違います。

測定単位の違いによる、c33 clockコマンドの表示は以下のとおりです。

```

(gdb)
c33 clock
9999999999 cycle      (サイクル数)

(gdb)
c33 clock
999999999.99 us      (μs 単位)

(gdb)
c33 clock
9999.999999 s        (秒単位)

```

■注 意

- デバッグモニタモードは実行カウンタをサポートしていないため、本コマンドは使用できません。
- 正確な計測値を得るには、計測を開始する位置にブレークが貼られていない状態で計測してください。
計測開始位置にブレークが貼られたまま計測を行うと、計測開始位置にある命令の実行時間が計測されません。
- 長時間の実行でカウンタの最大値を超えた場合は"clock timer overflow"を表示します。
- 実行カウンタは次の場合にリセットされます。
 1. 実行カウンタのモードをc33 clockmdコマンドで切り換えた場合(積算モード↔リセットモード)
 2. リセットモードでプログラムの実行を開始した場合
リセットモードの場合、ステップ数を1としたシングルステップ実行(または、ボタン/メニューによるステップ実行)では、1ステップごとにリセットされます。
 3. CPUをコールドリセットまたはホトリセットした場合

target (ターゲットの接続)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

ターゲットとの接続を確立し、コネクトモードを設定します。

ICD2モード: S5U1C33000H(ICD ver.2)とシリアルインタフェースで接続します。

ICD3モード: S5U1C33001H1100(ICD ver.3)またはS5U1C33001H1200(ICD ver.4)とUSBインタフェースで接続します。

ICD6モード: S5U1C33001H1400(ICD ver.6)とUSBインタフェースで接続します。

シミュレータモード: デバッグをシミュレータモードに設定します。

デバッグモニタモード: S5U1C330M1D1(+S5U1C330M2S)とシリアルインタフェースで接続します。

書 式

target *Type* [*Time*]

Type: ターゲットを指定する以下のシンボル

icd usb S5U1C33001H1100(ICD ver.3)またはS5U1C33001H1200(ICD ver.4)とUSBインタフェースで接続(ICD3モード)

icd usb2 S5U1C33001H1100(ICD ver.3)またはS5U1C33001H1200(ICD ver.4)とUSBインタフェースで接続(ICD3モード)

1台のPCと2台のICD ver.3/ver.4をUSB接続してデバッグするときに使用します。

デバッグは--double_startingオプションを指定して起動し、"target icd usb2"で接続します。これにより1台のPCでデバッグを2つ同時に起動できます。

icd6 usb S5U1C33001H1400(ICD ver.6)とUSBインタフェースで接続(ICD6モード)

icd comN S5U1C33000H(ICD ver.2)またはS5U1C330M1D1(+S5U1C330M2S)とシリアルインタフェースで接続(ICD2モードまたはMONモード)

Nはポート番号で、com1～com9から選択可能です。

S5U1C33000HとS5U1C330M1D1(+S5U1C330M2S)のどちらが接続されているかについては、デバッグが判断します。

sim シミュレータを起動(SIMモード)

Time: 通信開始遅延時間(秒数、10進数または16進数)

パソコンのCOMポートをオープンした後、ここで設定した秒数の待ち時間を取ってから通信を開始します。省略時はすぐに通信を開始します。また、ICD3/ICD6モードおよびシミュレータモードでは不要です。

使用例**例1**

(gdb)

target sim

boot () at ./boot.s:9

Connected to the simulator.

シミュレータモードに設定します。

例2

(gdb)

target icd usb

ICD3モードに設定します。

例3

(gdb)

target icd com1 10

ICD2モードまたはデバッグモニタモードに設定します。COM1ポートをオープン後、S5U1C33000HまたはS5U1C330M1D1(+S5U1C330M2S)との通信を開始する前に10秒の待ち時間が取られます。

注 意

- パラメータファイルを読み込んでメモリマップ設定を行う場合、c33 rpfコマンドは必ずtargetコマンドの前に実行してください。また、targetコマンドはloadコマンドの前、fileコマンドはtargetコマンドより前に実行してください。基本的な実行順序は次のとおりです。

```
(gdb)
c33 rpf sample.par           (マップ情報設定)
(gdb)
file sample.elf              (デバッグ情報の読み込み)
(gdb)
target icd com1             (本コマンド)
(gdb)
load                          (プログラムのロード)
(gdb)
c33 rsth                      (ホットリセット)
```

- パソコンによってはCOMポートオープン後、すぐに使用可能にならないものがあります。パラメータTimeは使用可能になるまでの待ち時間を取るために使用します。Timeを省略したtargetコマンドで、S5U1C33000Hとの接続に失敗した場合などに指定してください。
- シミュレータモード(target sim)の場合、一度切断(detach)しても2度目のtarget simコマンドはエラーになります。この場合、デバッガを再起動してからtarget simコマンドを実行してください。

detach (ターゲットの切断)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

ターゲットとの通信ポートを閉じ、現在のコネクトモードを終了します。

書 式

detach

使用例

```
(gdb)
target icd usb
      :
      デバッグ
      :
(gdb)
detach
```

ICD3モードを終了します。

注 意

本コマンドはシミュレータモードと他のモードの切り換えや、ターゲットボードの操作のためにS5U1C33000H/S5U1C33001HをOFFする場合などに使用します。デバッグを終了する場合に実行させる必要はありません。

c33 das (デバッグユニットベースアドレス設定)

[ICD3 / ICD6]

機 能

ICD ver.3/ver.4/ver.6(S5U1C33001H)に設定するCPUタイプとデバッグユニットのアドレスを指定します。

本コマンドはターゲットがC33 PEコアを使用し、かつデバッグRAMベースアドレスが0x71000以外のときに限り実行する必要があります。それ以外の場合、実行する必要はありません。

本コマンドで指定した内容は、targetコマンドによりICD(S5U1C33001H)に設定されます。したがって、本コマンドは、targetコマンドの前に実行しておく必要があります。

書 式

c33 das *DebugUnitBaseAddress* *DebugRAMAddress* [*CPUtype*]

DebugUnitBaseAddress: デバッグユニットが実装されているアドレス(10進数または16進数)

DebugRAMAddress: デバッグ用RAMが実装されているアドレス(10進数または16進数)

CPUtype: CPUタイプ

- 0 C33 STD(デフォルト)
- 1 C33 PE
- 3 S1C33401(C33 ADV)

使用例

```
(gdb)
c33 das 0x60000 0x71000 3
(gdb)
target icd usb
```

本コマンドで、デバッグ用ROMアドレスを0x60000番地、デバッグ用RAMアドレスを0x71000番地、CPUタイプをC33 ADVコアに指定し、targetコマンドでICDに設定します。

注 意

- c33 dasコマンドはICD3またはICD6モード時のみ有効です。
- 本コマンドを実行せずにtargetコマンドを実行した場合、ICDには以下のとおり設定されます。

デバッグ用ROMアドレス: 0x60000

デバッグ用RAMアドレス: 0x71000

CPUタイプ: C33 STD

- 通常、本コマンドはtargetコマンドの前に一度だけ実行します。ただし、その後デバッグユニットのベースアドレスが変更されるような場合は、変更ごとに新しいベースアドレスを指定して本コマンドを再実行する必要があります。このときはCPUタイプが無視されますので、省略してください。

例:

```
(gdb)
c33 das 0x60000 0x71000 1
(gdb)
target icd usb
    デバッグ開始
    ...途中メモリの配置がハード的に変更された。

(gdb)
c33 das 0x800000 0x811000
    デバッグ再開
```

c33 lpt (パラレルロードモード設定 / 解除)

[ICD2]

機 能

ICD ver.2(S5U1C33000H)で、パラレルケーブルを使用する高速ダウンロードを使用するかどうかを設定します。このコマンドはloadコマンドの前に実行する必要があります。
このコマンドを実行しない場合、シリアル(COM)ポートを使用してダウンロードを行います。

書 式

c33 lpt [*PortNumber*]

PortNumber: PCのプリンタ(LPT)ポートの番号(10進数または16進数)

1または**2**

省略時はシリアル(COM)ポートが選択されます。

使用例

```
(gdb)
file test.elf
(gdb)
target icd com1
(gdb)
c33 lpt 1           ←パラレルモード設定
(gdb)
load
(gdb)
c33 lpt           ←パラレルモード解除
(gdb)
load
```

1回目のloadコマンドは、パラレルポート(LPT1)を使用した高速ダウンロードを行います。

2回目のloadコマンドは、シリアルポートを使用した通常のダウンロードを行います。

注 意

- c33 lptコマンドはICD2モード時のみ有効です。
- 本コマンドはloadコマンドの前に実行する必要があります。

pwd (カレントディレクトリの表示)**cd** (カレントディレクトリの変更)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能**pwd:** カレントディレクトリを表示します。**cd:** カレントディレクトリを変更します。**書 式****pwd** (カレントディレクトリの表示)**cd *Directory*** (カレントディレクトリの変更)*Directory:* ディレクトリを指定する文字列**使用例**

(gdb)

pwd

Working directory /cygdrive/c/EPSON/gnu33/sample/tst.

(gdb)

cd /cygdrive/c/EPSON/gnu33/sample/asm

Working directory /cygdrive/c/EPSON/gnu33/sample/asm.

カレントディレクトリを確認後c:¥EPSON¥gnu33¥sample¥asm¥に変更します。

注 意ドライブ名の書式は/cygdrive/**ドライブ名**/です。"c:"の形式では指定しないでください。

また、ディレクトリの区切り文字は、¥ではなく/を使用してください。

c33 firmupdate (ファームウェア更新)

[ICD3 / ICD6]

機能

ICD ver.3/ver.4/ver.6(S5U1C33001H)のフラッシュメモリに書き込まれているファームウェアを更新します。

本コマンドは、ICD(S5U1C33001H)接続後に有効となります。したがって、本コマンドより先にtargetコマンドを実行しておく必要があります。

更新終了後はgdbを一旦終了し、ICD(S5U1C33001H)の電源を入れ直してください。

書式

c33 firmupdate *Filename*

Filename: ファームウェアファイル名(モトローラS3形式ファイル)

使用例

```
(gdb)
target icd usb
(gdb)
c33 firmupdate icd33dmt.sa
```

ICD(S5U1C33001H)と接続後、ファームウェアファイルicd33dmt.saでICD(S5U1C33001H)のファームウェアを更新します。

注意

c33 firmupdateコマンドはICD3またはICD6モード時のみ有効です。

c33 dclk (DCLK の分周値変更)

[ICD2 / ICD3 / ICD6 / MON]

機 能

コアクロックからDCLKを生成するための分周値を変更します。

書 式

c33 dclk *Number*

Number: 分周値

1、2、4、または8

使用例**■例1**

(gdb)

c33 dclk 2

DCLKをコアクロックの1/2分周に設定します。

■例2

(gdb)

c33 dclk 8

DCLKをコアクロックの1/8分周に設定します。

注 意

本コマンドは、シミュレータモードでは使用できません。

c33 oscwait (OSC1 モード時のウェイト設定)

[ICD3 / ICD6]

機 能

ターゲットをOSC1(低速クロック)モードに設定した場合、デバッガでGO/ブレークが正常に行われなくなることがあります。これは、ターゲット側の動作速度が落ちることによりICDとの送受信が正常に行われなくなることによります。

本コマンドは、ICDとターゲット間の通信タイミングを調整し、ターゲットの動作クロックが低速でも正常動作するために使用します。

書 式

c33 oscwait [*Counter*]

Counter: GO/ブレーク時に挿入するソフトウェアウェイト数(10進数または16進数)
省略時は0x30000を設定します。

条件: $0 \leq \textit{Counter} \leq 0\text{xfffffff}$

使用例

(gdb)

c33 oscwait 0x40000

ICDのファームウェアのGO/ブレーク時に0x40000回ループ分のウェイトが挿入されます。

注 意

- 本コマンドは、ICD3またはICD6モード時のみ有効です。
- 本コマンドを使用しない場合、カウントは0x300に設定されています。
通常(OSC1動作時以外)は、このカウンタ値のまま使用してください。
- カウント数を増やす程、デバッガのGO/STEP/ブレークのレスポンスは悪くなります。

c33 cachehit (キャッシュヒット率の表示)

[ICD3 / ICD6]

機 能

キャッシュのヒット率を表示します。

ヒット率は、 $(1 - (\text{ミスヒットカウンタ} \div \text{アクセスカウンタ})) \times 100\%$ で計算します。

書 式

```
c33 cachehit
```

使用例

```
(gdb)
c33 cachehit
The rate of cache hit is XX.XXXXX%
```

注 意

本コマンドは、ICD3またはICD6モードでターゲットCPUがS1C33401(C33 ADVコア)のときのみ使用可能です。

c33 logiana (ロジックアナライザモード設定)

[ICD3 / ICD6]

機 能

ターゲットCPUがC33 STDまたはC33 PEのときでもオンチップバストレース出力コマンド(c33 ot_d, c33 ot_f)をユーザモードで実行可能にします。

書 式

c33 logiana Mode USER_IN

Mode: ロジアナモード選択

On 有効

Off 無効(デフォルト)

USER_IN: 0 トレース入力(外部トレース入力をトレースする)(デフォルト)

1 トレーストリガ(ターゲットCPUトリガ信号をトレースする)

使用例

(gdb)

```
c33 logiana on 1
```

```
logic analyzer mode on.
```

```
target CPU trigger signal.
```

ロジアナモードをトレーストリガ入力で有効にします。

注 意

以下の場合、本コマンドは実行できません。

- ターゲットCPUがS1C33401(C33 ADVコア)の場合
- ICD2、MON、シミュレータモードの場合

c33 help (ヘルプ)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

コマンド説明を表示します。

書 式

c33 help [*Command*]

c33 help [*GroupNo.*]

Command: コマンド名

GroupNo.: コマンドグループ番号

使用例**例1**

```
(gdb)
c33 help
group 0: memory ..... c33 fb,c33 fh,c33 fw,x /b,x /h,x /w,set {char}, ...
group 1: register ..... info reg,set $
group 2: execution ..... continue,until,step,stepi,next,nexti,finish, ...
group 3: CPU reset ..... c33 rstc,c33 rsth,c33 rstt
group 4: interrupt ..... c33 int
group 5: break ..... break,tbreak,hbreak,thbreak,watch,rwatch,awatch, ...
group 6: symbol ..... info locals,info var,print
group 7: file ..... file,load,c33 lpt
group 8: map ..... c33 rpf,c33 map
group 9: flash memory .... c33 fls,c33 fle
group 10: trace ..... c33 tm,c33 td,c33 ts,c33 tf,c33 obt,c33 otd,c33 otf
group 11: simulated I/O .... c33 stdin,c33 stdout
group 12: flash writer .... c33 fwe,c33 fwlp,c33 fwld,c33 fwdc,c33 fwd
group 13: others ..... c33 dclk,c33 oscwait,c33 log,source,c33 clockmd, ...
Please type "c33 help 1" to show group 1 or type "c33 help c33 fb" to get usage ...
```

パラメータを省略すると、コマンドグループの一覧を表示します。

例2

```
(gdb)
c33 help 2
group 2: execution
continue      Execute continuously
until         Execute continuously with temporary break
step          Single-step every line
stepi         Single-step every mnemonic
next          Single-step with skip every line
nexti         Single-step with skip every mnemonic
finish        Quit function
c33 callmd    Set user function call mode
c33 call      Call user function
Please type "c33 help continue" to get usage of command "continue".
```

コマンドグループ番号を指定すると、そのグループに属するコマンドの一覧を表示します。

例3

```
(gdb)
c33 help step
step: Single-step,every line [ICD2/ICD3/ICD6/SIM/MON]

usage: step [Count]
Count: Number of steps to execute (decimal or hexadecimal)
       One step is assumed if omitted.
Conditions: 1-0x7fffffff

example:
(gdb)
step
```

10 デバッガ

```
(gdb)
step 10
```

コマンドを指定すると、詳細説明を表示します。

■例4

```
(gdb)
c33 help c33 rsth
c33 rsth: Hot reset                                [ICD2/ICD3/ICD6/SIM/MON]

usage: c33 rsth

example:
(gdb)
c33 rsth
The CPU is hot-reset.
```

C33系のコマンドの詳細説明を表示する場合は、"c33"の後に続けてコマンドを指定します。

注 意

- c33 helpコマンドではなくgnu標準のhelpコマンドを実行すると、gnuデバッガに設定されたコマンドクラスとコマンドのヘルプが表示され、その中には本デバッガがサポートしていないコマンドも含まれます。
本マニュアルで説明していないコマンドは、動作が保証されませんので注意してください。
- 詳細説明(例3、例4)で表示された"[ICD2/ICD3/ICD6/SIM/MON]"は、コマンドが有効なモードを示しています。
ICD2: ICD2モードで有効(S5U1C33000H(ICD ver.2)使用時)
ICD3: ICD3モードで有効(S5U1C33001H1100(ICD ver.3)またはS5U1C33001H1200(ICD ver.4)使用時)
ICD6: ICD6モードで有効(S5U1C33001H1400(ICD ver.6)使用時)
MON: デバッグモニタモードで有効(S5U1C330M1D1とS5U1C330M2S使用時)
SIM: シミュレータモードで有効(PCのみでのデバッグ時)
"[ICD2/MON]"のような表示は、そのコマンドがICD2モードまたはデバッグモニタモード以外では使用できないことを意味します。

quit (デバッガ終了)

[ICD2 / ICD3 / ICD6 / SIM / MON]

機 能

デバッガを終了します。

デバッガが使用したポートやファイルが開いているときは、すべて閉じます。

書 式

quit

q (省略形)

使用例

(gdb)

q

10.8 パラメータファイル

パラメータファイルはターゲットシステムのメモリマップ情報を記述したテキストファイルです。デバッグはこのファイルを読み込んでメモリマップ情報を構築し、この設定に基づいて以下の処理を行います。

- ソフトウェアPCブレイクアドレスが正当なマップ領域内かどうかのチェック
- ROM領域への書き込みによるブレイク(シミュレータモードのみ)
- 未定義領域へのアクセス(シミュレータモードのみ)
- スタックのオーバーフロー(シミュレータモードのみ)

シミュレータモードでパラメータファイルを読み込んだ場合、ファイルに記述されているすべてのメモリ領域を合わせたサイズのシミュレーションメモリがパソコン上のメモリ内に確保されます。

●パラメータファイルの読み込み方法

パラメータファイルは、c33 rpfコマンドの実行によってデバッグに読み込まれます。

(gdb)

c33 rpf *Filename.par* (パラメータファイルを読み込んでメモリマップを設定)

IDEで、デバッグが起動時に読み込まれるようなコマンドファイルを作成することもできます。IDEの詳細については、"5 GNU33 IDE"を参照してください。

c33 rpfコマンドは、file、target、loadコマンドより先に実行してください。

基本的な実行順序は次のとおりです。

(gdb)

c33 rpf sample.par (マップ情報設定)

(gdb)

file sample.elf (デバッグ情報の読み込み)

(gdb)

target sim (ターゲットの接続)

(gdb)

load (プログラムのロード)

(gdb)

c33 rsth (ホトリセット)

●パラメータファイルの作成方法

パラメータファイルは、IDEの[プロパティ]ダイアログボックスから[GNU33 パラメータファイル設定]を選択して作成できます。IDEの詳細については、"5 GNU33 IDE"を参照してください。パラメータファイルはテキストファイルのため、汎用エディタで作成/修正することも可能です。

注: ファイル名(拡張子も含む)、ファイル内のテキストに日本語を使わないでください。

●パラメータファイルの内容

以下にパラメータファイルの例を示します。

```
#gnu33 gdb parameter file (1)
C33_ADVANCED (2)

RAM 000000 001fff 00W #IRAM (3)
IO 040000 04ffff 00H #IO
RAM 600000 6fffff 11H #RAM
ROM c00000 cfffff 55B B #ROM
STACK 000000 001fff #STACK (4)
```

(1) コメント

#からその行の終わりまではコメントとみなされます。

(2) CPUのタイプ

S1C33401 (C33 ADVコア)またはC33 PEのデバッグを行う場合、それぞれC33_ADVANCED、C33_PEを記述します。

C33 STDでは指定の必要はありません。

実際には、この設定はシミュレータモードでのみ有効です。ICD2、ICD3、ICD6、デバッグモニタモードでは、ターゲットCPUを自動認識するため、この指定は必要ありません。ただし、シミュレータモードで動作させる場合のことを想定し、S1C33401(C33 ADV)またはC33 PEの場合はC33_ADVANCEDまたはC33_PEを記述しておくことを推奨します。

(3) メモリマップ情報

各行の構成は次のとおりです。

Device StartAddr EndAddr [Condition] [BigEndian] [#Comment]

Device メモリの種類を以下のシンボルで指定します。

ROM 書き込み専用メモリ領域
RAM 読み出し/書き込み可能メモリ領域
IO 周辺回路制御用メモリ領域

StartAddr 領域の先頭アドレスを指定します。
 アドレスは16進数で指定しますが、数値の前の0xは不要です。

EndAddr 領域の終端アドレスを指定します。
 アドレスは16進数で指定しますが、数値の前の0xは不要です。

Condition ウェイトサイクルとデバイスサイズを次のように指定します。
 <リードサイクルのウェイト数><ライトサイクルのウェイト数><デバイスサイズ>
 ウェイト数: 0~f 0~15サイクル
 データサイズ: **B** 8ビット
 H 16ビット
 W 32ビット

たとえば、リードサイクルが1ウェイト、ライトサイクルが2ウェイトの16ビットデバイスは12Hとなります。

このパラメータはシミュレータモードでのみ有効で、省略することも可能です。省略した場合は77Hとして処理されます。

シミュレータ以外のモードでは、これらの条件がBCUレジスタで実際のターゲットに設定されるため、このパラメータは無視されます。

BigEndian ビッグエンディアンデバイスの場合に**B**を記述します。
 このパラメータはシミュレータモードでのみ有効です。ただし、内蔵のROM、RAM、I/Oはビッグエンディアンに設定することはできません。省略した場合はリトルエンディアンとして処理されます。
 シミュレータ以外のモードでは、このパラメータは無視されます。

#Comment 必要なパラメータが記述されていれば、その後ろに#から始まるコメントを記述しておくことができます。

注: ・メモリマップ情報の*Device*、*StartAddr*、*EndAddr*は省略できません。

- ・メモリ領域を二重に指定した場合、先に指定した内容が有効となります。

RAM	600000	6fffff	11H	#RAM	有効
ROM	600000	6fffff	22H	#ROM	無効
IO	600000	7fffff	33H	#I/O	0x700000~0x7fffffがIOエリアとして有効

- ・メモリマップのひとつのエリアに256MB以上の範囲を指定しないでください。デバッグ起動時に領域の確保ができないことがあります。

(4) スタック領域情報

次の書式で、スタックとして使用する領域を指定します。

STACK StartAddr EndAddr

StartAddr 領域の先頭アドレスを指定します。
 アドレスは16進数で指定しますが、数値の前の0xは不要です。

EndAddr 領域の終端アドレスを指定します。

アドレスは16進数で指定しますが、数値の前の0xは不要です。

この設定はシミュレータモード時に有効で、スタックがオーバーフローした場合にブレークを発生します。

この設定は、プログラムによるSPの操作に影響を与えるものではありません。

●パラメータファイルを読み込まない場合

パラメータファイルはデバッグのための必須ファイルではありません。デバッグに読み込まなくてもデバッグは行えます。ただし、以下の制限が付きます。

ICD2、ICD3、ICD6、デバッグモニタモード時

- デバッグはソフトウェアPCブレークの設定アドレスのチェックを行いません。
- c33 tf、c33 otfコマンドによってトレースデータをファイルに保存することができません。

シミュレータモード時

- c33 rpfコマンドを実行していない状態で、target simコマンドを実行すると、以下の領域にRAMが割り当てられたものとしてシミュレーションメモリを確保します。この場合のメモリの初期値は0x00です。

0～0x1fff番地	(8KB)
0x40000～0x4ffff番地	(64KB)
0x600000～0x6ffff番地	(1MB)
0xc00000～0xcffff番地	(1MB)
- ソフトウェアPCブレークの設定アドレスのチェックを行いません。
- 以下のマップブレークが機能しません。
 1. ROM領域への書き込みによるブレーク
 2. 未定義領域へのアクセス
 3. スタックのオーバーフロー
- 実行カウンタのカウンタおよびトレース情報内のクロック数が正しい値を示しません。

10.9 ステータス/エラーメッセージ

10.9.1 ステータスメッセージ

ターゲットプログラムがブレークすると、コマンド入力待ちになる直前に以下のブレーク要因を示すメッセージを表示します。

表10.9.1.1 ステータスメッセージ

メッセージ	内容
Breakpoint #, function at file:line	設定したブレークポイントでブレーク
Break by hard pc break1.	ハードウェアPCブレークポイント1でブレーク
Break by hard pc break2.	ハードウェアPCブレークポイント2でブレーク
Break by accessing no map.	シミュレータモードでマップされていない領域へのアクセスによりブレーク
Break by writing ROM area.	シミュレータモードでリードオンリ領域へのアクセスによりブレーク
Break by stack overflow.	シミュレータモードでスタック領域のオーバーによるブレーク
Break by sleep or halt.	シミュレータモードでsleepまたはhalt命令実行によるブレーク
Illegal address exception.	シミュレータモードで不当アドレスによる命令の実行によりブレーク
Illegal instruction.	シミュレータモードで不当命令の実行によりブレーク
Illegal delayed instruction.	シミュレータモードで不当な遅延命令の実行によりブレーク
Hardware read watchpoint #: symbol	データリード条件でブレーク
Hardware access (read/write) watchpoint #: symbol	データアクセス(リード/ライト)条件でブレーク
Program received signal SIGTRAP, Trace/breakpoint trap.	[中断]ボタンによる強制ブレーク (シミュレータモード)
Program received signal SIGINT, Interrupt.	[中断]ボタンによる強制ブレーク(ICDモード)

10.9.2 エラーメッセージ

表10.9.2.1 エラーメッセージ(アルファベット順)

メッセージ	内容
A setup of a serial port was not completed.	ICDとのシリアル通信速度が115200または38400bps以外です。
Address (0x#) is ext or delayed instruction.	指定したアドレスは- 先頭のext命令を除く拡張命令または、- 遅延命令(遅延分岐命令の次)のため設定ができません。
C33 command error, command is not supported in present mode.	入力したコマンドは、現在の接続モードでは実行できません。
C33 command error, command is too long.	入力したコマンド長が256文字を超えています。
C33 command error, invalid command.	コマンドに誤りがあります。
C33 command error, number of parameter.	コマンドのパラメータ数が正しくありません。
C33 command error, start address > end address.	開始アドレスが終了アドレスを超えて指定されています。
C33 command error, start cycle < end cycle.	終了サイクルが開始サイクルを超えて指定されています。
C33 command error, cycle is 0 - 131071(ICD33V2) or 0 - 1048575(ICD33V3/V6).	指定したサイクル番号が指定可能範囲を超えています。
Cannot access memory at address #.	アドレス#をアクセスできません。
Cannot allocate memory.	パラメータで指定したサイズのメモリ領域が確保できません。
Cannot clear data break (0x#).	未設定のデータブレークアドレスを指定しました。
Cannot clear hard pc break (0x#).	未設定のハードウェアPCブレークアドレスを指定しました。
Cannot clear soft pc break (0x#).	未設定のソフトウェアPCブレークアドレスを指定しました。
Cannot open file (file).	ファイルがオープンできません。
Cannot open ICD33 usb driver.	USBドライブのオープンに失敗しました。
Cannot set data break any more.	データブレークポイントを設定できません。
Cannot set hard pc break any more.	ハードウェアPCブレークポイントの設定数が制限を超えました(Max.2)。
Cannot set soft pc break any more.	ソフトウェアPCブレークポイントの設定数が制限を超えました(Max.200)。
Cannot set hard pc break at ext or delayed instruction.	ハードPCブレークで指定したアドレスはext命令または遅延命令のため設定ができません。
Cannot set soft pc break at ext, delayed instruction or No RAM Area.	ソフトウェアPCブレークで指定したアドレスは- 先頭のext命令を除く拡張命令の行 - 遅延命令の行(遅延分岐命令の次の行)または、RAM領域でないため設定ができません。

メッセージ	内容
Cannot set same breakpoint address.	指定したアドレスには既にブレイクが設定されています。
Cannot set soft pc break at ROM area.	ソフトウェアPCブレイクで指定したアドレスはROM領域であるため設定ができません。
Cannot write file.	ファイルへの書き込みができません。
Clock timer overflow.	クロック計測でタイムオーバーになりました。
Communication error (bcc).	ICDからの受信メッセージにBCCエラーが発生しました。
Communication error (host->ICD33).	ICDへのUSB送信に失敗しました。
Communication error (ICD33-> host).	ICDへのUSB受信に失敗しました。
Communication internal error (#).	ICDとの通信中に内部エラーが発生しました。
Communication system error (#).	ICDとの通信中に切断してしまいました。
Coverage is not supported in present mode.	カバレッジ・ウィンドウは、現在のコネクモードではサポートしていません。シュミレータモードでご使用ください。
Coverage Window is already opened.	カバレッジ・ウィンドウが既に開いています。
"Editor path" exceeds 255 characters.	"Editor path"が255文字を超えています。
Expression cannot be implemented with read/access watchpoint.	指定したアドレスへのデータブレイクは設定できません。
Framing error.	ICDとの通信でフレーミングエラーが発生しました。
gnuEdit.gdb file save error	外部エディターパスの保存に失敗しました。
ICD33 is busy (#).	ICD側がBUSY状態です。
Initialization error of ICD33.	ターゲットの初期化に失敗しました。
Invalid parameter file (#: file).	ターゲットの初期化に失敗しました。
Invalid parameter file, start address > end address (#: file).	パラメータファイルのアドレス範囲設定で、開始アドレスが終了アドレスを超えています。
It has not connected with a target.	ICDとターゲットが接続できません。
It is not c33 architecture ELF file.	fileコマンドで指定したファイルはS5U1C33001Cが対応しているelf形式ファイルではありません。
Load max address (0x#) overflow.	ロードするアドレスが最大値を超えています。
Load motorola file format error.(file)	ロードするモトローラファイルのフォーマットに誤りがあります。
No memory map information.	マップ情報がありません。
Please input "Editor path".	"Editor path"に文字を入力してください。
Profiler is not supported in present mode.	プロファイラ・ウィンドウは、現在のコネクモードではサポートしていません。シュミレータモードでご使用ください。
Profiler Window is already opened.	プロファイラ・ウィンドウが既に開いています。
Receiving message is inaccurate.	ICDとの通信で最大サイズを超えるメッセージを受信しました。
Specification is required in the device for connecting.	targetコマンドでICD選択のデバイス名が足りません。
Target connection causes communication error at # times.	SIO接続チェックで異常が発生しました。
Target connection causes data error at # times.	SIO接続チェックが実行できません。
Target connection causes time out error at # times.	SIO接続チェックがタイムアウトです。
Target down.	ICD/ターゲット間で通信エラーが発生しました。
The simulator cannot be connected twice. please quit gdb.	シュミレータモードの接続が2度行えません。GDBを再起動してください。
There is no argument given to this command.	ターゲットの切断に失敗しました。
Timeout error # (ICD33 -> host).	ICDとの通信でICDからの受信待ちタイムアウトが発生しました。
Transmitting failure (#).	ICDとの通信でICDからNAKを受け取りました。
Warning: target file is not standard macro program.	fileコマンドで指定したELFファイルがスタンダードマクロ用ではありません。
Warning: target file is not PE program.	fileコマンドで指定したELFファイルがPE用ではありません。
Warning: target file is not advanced macro program.	fileコマンドで指定したELFファイルがアドバンスドマクロ用ではありません。

ICD: S5U1C33000HまたはS5U1C33001H

11 その他のツール

この章では本パッケージに含まれるその他のツールについて説明します。

11.1 make.exe

11.1.1 機能

本パッケージはコンパイルからリンクまでを効率よく処理するメイクツール(以下**make.exe**と記述)を含んでいます。

make.exeは、makeファイルに記述されたソースや各ツールが出力するファイルの依存関係を元に必要なツールを実行し、オブジェクトファイルを最新の状態に更新します。たとえば、すでに一度以上のメイク処理が行われた後で1つのソースファイルのみを修正した場合、**make.exe**はそのファイルのみを対象にコンパイルあるいはアセンブル処理を実行します。他のモジュールはリンク時にオブジェクトファイルが読み込まれるのみで、アセンブルまでの処理は行いません。

本パッケージの**make.exe**はGNU make(ver. 3.81)をベースに生成されていますが、上記の処理を行うための依存リスト、サフィックス定義およびマクロ定義のみをサポートしています。

11.1.2 入力ファイル

●makeファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.mak

内容: メイク処理の内容を記述したファイルです。通常、**IDE**で生成したmakeファイルを使用します。

11.1.3 起動方法

●コマンドラインの一般形

`make [<オプション>] [<ターゲット名>]`

[]は省略可能なことを示します。

例: `make -f test.mak clean`

●IDEからの起動

ビルドを実行すると、呼び出されます。

●オプション

`make.exe`には次の起動オプションが用意されています。

`-f <ファイル名>`

機能: `make` ファイルの指定

説明: `make.exe`は<ファイル名>(拡張子を含む)で指定されたmakeファイルを読み込み、その内容を処理します。

デフォルト: `-f`オプションが指定されない場合は、"makefile"の名称を持つファイルをmakeファイルとして入力します。

●ターゲット名

実行させるコマンドのターゲット名(コマンドの位置を示すラベル)を指定します。省略した場合は、makeファイル内で最初に現れるターゲットを実行します。

IDEで作成するmakeファイルには、メイクで生成されたファイルを削除するためのターゲット名(clean)が記録されています。

例: `make -f test.mak clean`

このように、cleanを指定することで、test.makの実行により生成されたオブジェクトファイルやマップファイルが削除されます。この機能は、すべてのソースファイルを対象に新規のメイク処理を行えるように付加されています。

IDEからcleanを実行するには、[プロジェクト]メニューから[クリーン...]を選択します。

S5U1C33001Cは、ターゲット名を省略した通常のmakeとターゲット名cleanおよびallを指定したmakeのみをサポートしています。

11.1.4 makeファイル

makeファイルはファイルの依存関係と実行するコマンドを記述したテキストファイルです。
IDEで生成されるmakeファイルの例を次に示します。

例:

```
# Make file generated by Gnu33 Plug-in for Eclipse
# This file should be placed directly under the project folder

# macro definitions for target file
TARGET= sample
GOAL= $(TARGET).elf

# macro definitions for tools
TOOL_DIR= C:/EPSON/GNU33
CC= $(TOOL_DIR)/xgcc
CXX= $(TOOL_DIR)/xgcc
AS= $(TOOL_DIR)/xgcc
AS_CC= $(TOOL_DIR)/as
LD= $(TOOL_DIR)/ld
RM= $(TOOL_DIR)/rm
SED= $(TOOL_DIR)/sed
CP= $(TOOL_DIR)/cp
OBJDUMP= $(TOOL_DIR)/objdump

# macro definitions for tool flags
CFLAGS= -B$(TOOL_DIR)/ -gstabs -S -medda32 -O1 -I$(TOOL_DIR)/include -fno-builtin
-mlong-calls -Wall -Werror-implicit-function-declaration
CXXFLAGS= -gstabs -S -medda32 -O1 -I$(TOOL_DIR)/include -fno-builtin -mlong-calls
-Wall
ASFLAGS= -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs -Wa,-medda32
ASFLAGS_CC= -medda32
LDFLAGS= -Map sample.map -N -T sample_gnu33IDE.lds
ALLOBJDUMP_FILE= __allobjects
EXTFLAGS= -Wa,-mc33_ext -Wa,$(TARGET).dump -Wa,$(ALLOBJDUMP_FILE).dump
EXTFLAGS_CC= -mc33_ext $(TARGET).dump $(ALLOBJDUMP_FILE).dump
OBJDUMPFLAGS= -t

# macro for switching 2pass or 1pass build
PASS= 2pass

# search paths for source files
vpath %.c
vpath %.cpp
vpath %.cc
vpath %.cxx
vpath %.C
vpath %.s

# macro definitions for object files
OBJS= boot.o ¥
      main.o ¥

# macro definitions for library files
OBJLDS= $(TOOL_DIR)/lib/std/libstdio.a ¥
        $(TOOL_DIR)/lib/std/libstdc++.a ¥
        $(TOOL_DIR)/lib/std/libgcc2.a ¥
        $(TOOL_DIR)/lib/std/libc.a ¥
        $(TOOL_DIR)/lib/std/libgcc.a ¥

# macro definitions for assembly files generated from c source files
CEXTTEMPS= main.ext0 ¥

# macro definitions for dependency files
DEPS= $(OBJS:%.o=%.d)
SED_PTN= 's/[[:space:]]¥([a-zA-Z]¥)¥:/ ¥/cygdrive¥/¥1/g'
SED_PTN2= 's/^¥$ (subst .,¥.,$(@F))¥)/$ (subst /,¥/, $(@))¥:/g'

# macro definitions for creating dependency files
DEPCMD_CC= @$(CC) -M -MG $(CFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e $(SED_PTN2)
>$(@:%.o=%.d)
DEPCMD_CXX= @$(CC_KFILT) -M -MG $(CXXFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e
$(SED_PTN2) >$(@:%.o=%.d)
```

11 その他のツール

```
DEPCMD_AS= @$ (AS) -M -MG $(ASFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e $(SED_PTN2) >$(@:%.o=%.d)

# targets and dependencies
.PHONY : all clean

all : $(GOAL)

$(TARGET).elf : $(OBJS) sample_gnu33IDE.mak sample_gnu33IDE.lds
ifeq ($(PASS), 1pass)
# 1pass linking
$(LD) $(LDFLAGS) -o $$ $(OBJS) $(OBJLDS)
else
# 1pass linking
-$(LD) $(LDFLAGS) -o $$ $(OBJS) $(OBJLDS) 2>lderr
@if [ -s lderr ]; then ¥
    cmd /c "type lderr" ¥
    && $(RM) -f $(TARGET).elf ¥
    && exit 1; ¥
else $(RM) -f lderr ; ¥
fi
$(OBJDUMP) $(OBJDUMPFLAGS) $$ > $(TARGET).dump
$(RM) -f $(TARGET).elf
# create all objets dump file
$(RM) -f $(ALLOBJDUMP_FILE).dump
for NAME in $(OBJS) ; do ¥
    $(OBJDUMP) $(OBJDUMPFLAGS) $$NAME >> $(ALLOBJDUMP_FILE).dump ; done
# save 1pass object files
@if [ -e obj1pass ]; then ¥
    cmd /c "rd /s /q obj1pass" ; ¥
fi
cmd /c "md obj1pass"
for NAME in $(subst /,¥¥,$(OBJS)) ; do ¥
    cmd /c "copy /y $$NAME obj1pass¥¥$$NAME" >nul ; done ¥
&& $(RM) -f $(OBJS)
# 2pass for assembly files
$(AS) $(ASFLAGS) $(EXTFLAGS) -o boot.o boot.s
# 2pass for c files
for NAME in $(basename $(CEXTTEMPS)) ; do ¥
    $(AS_CC) $(ASFLAGS_CC) $(EXTFLAGS_CC) -o $$NAME.o $$NAME.ext0 ; done
$(RM) -f $(TARGET).map
# 2pass linking
$(LD) $(LDFLAGS) -o $$ $(OBJS) $(OBJLDS)
# restore 1pass object files
$(RM) -f $(OBJS) ¥
&& ¥
for NAME in $(subst /,¥¥,$(OBJS)) ; do ¥
    cmd /c "copy /y obj1pass¥¥$$NAME $$NAME" >nul ; done ¥
&& cmd /c "rd /s /q obj1pass"
endif
@cmd /c "echo ----- Finished building target : $$ -----"

## boot.s
boot.o : boot.s
$(AS) $(ASFLAGS) -o $$ $<
$(DEPCMD_AS)

## main.c
main.o : main.c main.ext0
$(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
$(AS_CC) $(ASFLAGS_CC) -o $$ $(@:%.o=%.ext0)
$(DEPCMD_CC)

## cpp.c
cpp.o : cpp.cpp cpp.ext0
$(CC) $(CXXFLAGS) -o $(@:%.o=%.ext0) $<
$(AS_CC) $(ASFLAGS_CC) -o $$ $(@:%.o=%.ext0)
$(DEPCMD_CXX)

# dependencies for assembled c source files
main.ext0 : main.c

# include dependency files
```

```

-include $(DEPS)
# clean files
clean :
    $(RM) -f $(OBJS) $(TARGET).elf $(TARGET).map $(DEPS) $(CEXTTEMPS) $(TARGET).dump
    $(ALLOBJDUMP_FILE).dump lderr
    @if [ -e objlpass ]; then ¥
        cmd /c "rd /s /q objlpass" ; ¥
    fi

```

●makeファイル内のパスの記述

makeファイルはcygwin形式(またはUNIX形式)の記述に対応しています。
このため、特にパスを記述する場合には、以下の3点に注意してください。

1) ドライブ名とパスの区切り文字

Windowsの"¥"は"/"に置き換えてください。また、ドライブ名は/cygdrive/<ドライブ名>/の形式で記述してください。

例: c:/EPSON/gnu33/sample/tst/boot.o : c:/EPSON/gnu33/sample/tst/boot.s

as -o boot.o c:/EPSON/gnu33/sample/tst/boot.s



/cygdrive/c/EPSON/gnu33/sample/tst/boot.o : /cygdrive/c/EPSON/gnu33/sample/tst/boot.s

as -o boot.o c:/EPSON/gnu33/sample/tst/boot.s

ドライブ名を"<ドライブ名>:"の形式で記述した場合、メイク実行中にエラーとなります。ただし、アセンブラなどのWindowsのコマンドラインを記述する場合、コマンドライン中の引数として指定するパスの記述には"<ドライブ名>:"の形式が使用可能です。

2) スペース

ディレクトリやファイル名に含まれるスペースは、前に"¥"を付けて記述してください。

例: Tool Folder → Tool¥ Folder

3) 大文字/小文字を区別

ディレクトリやファイル名は大文字/小文字が区別されます。大文字/小文字も正確に記述してください。

make.exe実行時は、**make.exe**を呼び出したディレクトリがカレントディレクトリとなります。そのディレクトリからの相対パスも指定可能です。

例: ../libraries/lib1.a

●コメント

#からその行の終端まではコメントとみなされます。

11.1.5 マクロ定義と参照

文字列をマクロ定義すると、makeファイル内ではその文字列をマクロ名で参照することができます。マクロ定義と参照の形式は以下のとおりです。

マクロ定義: **<マクロ名> = <置き換え文字列>**

参照: **\$(マクロ名)**

例:

```
TARGET= sample
:
TOOL_DIR = /cygdrive/c/EPSON/gnu33
:
LD= $(TOOL_DIR)/ld
LIB_DIR= $(TOOL_DIR)/lib
:
LDFLAGS= -T $(TARGET).lds -Map $(TARGET).map -N
:
OBJS= boot.o ¥
      main.o ¥
:
OBJLDS=
:
LIBS= $(LIB_DIR)/libc.a $(LIB_DIR)/libgcc.a
:
$(LD) $(LDFLAGS) -o $$ $(OBJS) $(OBJLDS) $(LIBS)
```

この最後の参照例はリンカを実行するコマンドラインです。上記のようにマクロ定義されている場合、この行は次のように置き換えられて実行されます。

```
/cygdrive/c/EPSON/gnu33/ld -T sample.lds -Map sample.map -N -o sample.elf boot.o main.o
/cygdrive/c/EPSON/gnu33/lib/libc.a /cygdrive/c/EPSON/gnu33/lib/libgcc.a
```

●定義済みマクロ

上記例で使用されている\$@は定義済みマクロの1つです。以下の3つのマクロが使用可能です。ただし、使用できるのは、依存リストのコマンドライン内のみに限られます。

\$@ 現在処理中のターゲットファイル名(拡張子を含む)に置き変わります。

例:

```
sample.elf : ...
      ld -o $@ ...      (= ld -o sample.elf ...)
```

\$* 現在処理中のターゲットファイル名(拡張子を除く)に置き変わります。

例:

```
sample.elf : ...
      ld $*.o ...      (= ld sample.o ...)
```

\$< 現在処理中のターゲットの最初の依存関係(拡張子含む)に置き変わります。

例:

```
main.o : main.c ...
      xgcc -o $@ $< (= xgcc -o main.o main.c)
```

●注意事項

同じマクロ名で二重定義した場合、最後に定義された内容が有効となります。

11.1.6 依存リスト

ここでは、サフィックス定義を使用しない場合の依存リストについて説明します。

●依存リストの形式

メイクは次の形式で記述された依存リストに従って実行されます。

形式1: <ターゲットファイル名>: <依存ファイル名1> [^ <依存ファイル名2>...]

```
[
    TAB      <コマンド1>
    TAB      <コマンド2>
    :
]
```

形式2: <ターゲット名>:

```
[
    TAB      <コマンド1>
    TAB      <コマンド2>
    :
]
```

- ^はスペースを示します。
- []は省略可能なことを示します。
- <コマンド>の前にはタブ(スペースは不可)が必要です。

形式1

形式1ではターゲットファイルを得るために必要な依存ファイルを指定し、ターゲットファイルが作成されていない場合、あるいは依存ファイルにひとつでもターゲットファイルより新しく生成されたものがある場合に、続くコマンドを実行させます。

通常、コマンドとしてツールの起動コマンドラインを記述し、ターゲットファイルにはそのツールの出力ファイル、依存ファイルにはツールの入力ファイルを指定します。

例: main.o : \$(SRC1_DIR)/main.c
 \$(CC) \$(CFLAGS) \$(SRC1_DIR)/main.c

この例ではターゲットファイルmain.oがmain.cに依存していることを示します。

ターゲットファイルmain.oがない場合、あるいはmain.cがmain.oより新しい場合(コンパイル後にソースを修正した場合など)、コマンド"\$ (CC) \$(CFLAGS) \$(SRC1_DIR)/main.c" (xgccによるコンパイル)が実行されます。

形式2

依存ファイルが記述されていない場合、<ターゲット名>は単なるラベルとして使用されます。**make.exe**の起動コマンドでその<ターゲット名>を指定することにより、記述されたコマンドを実行させることができます。

例: make -f test.mak cleanで実行されるコマンド

```
clean:
    $(RM) -f $(OBSJ) $(TARGET).elf $(TARGET).map $(DEPS)
```

起動コマンドで<ターゲット名>を指定しなかった場合は、ファイル内で最初に記述されている依存リストによるメイクを行います。

コマンドとしては、.exeを持つ実行形式のコマンドとそのパラメータが記述できます。コマンドの記述がない場合は何も実行しません。ただし、ターゲットファイルと最初の依存ファイルの拡張子を持つサフィックスリスト(後述)が記述されていれば、そのコマンドを実行します。

●make.exeによる依存リストの処理

たとえば、`target.elf`が`boot.s`と`main.c`の2つのソースファイルから生成される場合、中間ファイル(.o)も含めたファイルの依存関係は、図11.1.6.1のようになります。つまり、ターゲットファイルを`target.elf`、`boot.o`、`main.o`とする3つの依存リストが必要です。

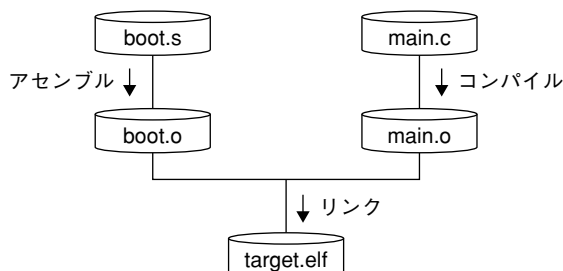


図11.1.6.1 ファイルの依存関係(例)

makeファイル例:

```

target.elf : boot.o main.o
    $(LD) $(LDFLAGS) -o target.elf boot.o main.o $(LIBS) ] (A)

boot.o : boot.s
    $(AS) $(ASFLAGS) -o boot.o boot.s ] (B)

main.o : main.c
    $(CC) $(CFLAGS) main.c ] (C)
  
```

(A) `target.elf`を生成するための依存リスト

(B) `boot.o`を生成するための依存リスト

(C) `main.o`を生成するための依存リスト

* マクロ`$(XXX)`の定義内容については前記の**IDE**が生成するmakeファイル例を参照してください。

このmakeファイルを初めて実行する場合のメイク処理は次のようになります。

1. **make.exe**は、makeファイル内で最初に現れる(A)の依存リスト(`target.elf: ...`)をチェックします。
2. 依存ファイルの`boot.o`と`main.o`はそれぞれ他の依存リストのターゲットになっているため、そちらの依存リストを先に評価します。
もし、`boot.o`または`main.o`が存在せず、それらを生成するための依存リストが記述されていない場合はエラーとなります。
3. (B)の依存リスト(`boot.o: ...`)が評価されます。この時点では`boot.o`が存在しないため、続くコマンド(`boot.s`のアセンブル)が実行されます。これにより`boot.o`が生成されます。
ここで、`boot.s`が存在しない場合は、それをターゲットファイルとする依存リストもないため、エラーとなります。この場合は、`boot.s`を指定のディレクトリ(この場合はカレントディレクトリ)に用意するか、`boot.s`や`boot.o`が関係している記述を削除する必要があります。
4. 3と同様に(C)の依存リスト(`main.o: ...`)が評価され、`main.o`が生成されます。
5. (A)の依存リストに戻ります。`target.elf`が生成されていないため、続くコマンド(リンク)が実行されます。これにより、`target.elf`が生成されます。

この後、main.cを変更してからメイクを再度実行した場合の処理は次のようになります。

1. **make.exe**はmakeファイル内で最初に現れる(A)の依存リスト(target.elf: ...)を評価します。
2. 依存ファイルのboot.oとmain.oはそれぞれ他の依存リストのターゲットになっているため、そちらの依存リストの評価を先に行います。
3. (B)の依存リスト(boot.o: ...)が評価されます。この時点ではboot.oが存在し、依存ファイルのboot.sよりも新しいため、続くコマンド(boot.sのアセンブル)は実行されません。
4. (C)の依存リスト(main.o: ...)が評価されます。依存ファイルのmain.cの作成時間が修正によりmain.oよりも新しくなっているため、続くコマンド(main.cのコンパイル)が実行されます。これによりmain.oが更新されます。
5. (A)の依存リストに戻ります。この時点では依存ファイルのmain.oがtarget.elfよりも新しいため、続くコマンド(リンク)が実行されます。これにより、target.elfが更新されます。

前回のメイクから依存ファイルがまったく変更されていない場合は、(A)～(C)のコマンドは実行されません。

●依存リストの記述に関する注意事項

- 依存リスト内で1つのファイルを頻繁に使用するような記述はなるべく避けてください。OS環境によっては実行時にファイルの日付が未来時間になってしまい、適切なメイクを実行することができません。

悪い例:

```
vector2.o : vector.c
    /cygdrive/c/EPSON/gnu33/sed.exe -f ../comm/place.sed vector.c > vector2.c
    /cygdrive/c/EPSON/gnu33/xgcc -B/cygdrive/c/EPSON/gnu33/ -c vector2.c -o vector2.o
```

この例ではvector.cにsed.exeによる置換を実行し、vector2.cを作成した後でコンパイルをしています。この場合は、次のような記述を推奨します。

良い例:

```
vector2.o : vector.c
    /cygdrive/c/EPSON/gnu33/xgcc -B/cygdrive/c/EPSON/gnu33/ -c vector2.c -o vector2.o
vector2.c : vector.c
    /cygdrive/c/EPSON/gnu33/sed.exe -f ../comm/place.sed vector.c > vector2.c
```

変更が難しい場合は、Make cleanを実行後に**make.exe**を再度実行してください。

- 依存リストの関係は3～4階層程度とし、極端に深い依存関係を作成しないようにしてください。
- 依存リストは4000件程度まで記述可能です。これ以上の依存リストを記述した場合は正常に動作しない可能性があります。
- ファイル名は最大で255文字までで、使用可能な文字は英数字のみです。漢字等の2バイトコードは使用できません。また、ファイルをフルパスで記述する場合も、パス全体で255文字を超えると、ファイルを正常に扱えない可能性があります。

11.1.7 サフィックス定義

たとえば、ターゲットファイルが`.o`、依存ファイルが`.c`の依存リストは通常、コンパイラのコマンドラインが記述されます。つまり、ファイル名のみを変更すれば、基本的に同じコマンドラインを同一種類のファイルの処理に共通に使用可能です。サフィックス定義は、ファイルの種類(拡張子)と実行するコマンドラインを定義しておくもので、個々の依存リスト内でのコマンドの記述を省略することが可能となります。この機能は、多数のソースファイルを扱う場合に、記述を簡略化するのに役立ちます。

サフィックス定義が記述されている場合、**make.exe**はサフィックス定義と同じファイル形式の構成でコマンドが記述されていない依存リストに対して、サフィックス定義内のコマンドを実行します。コマンドが記述されている場合は、依存リスト内のコマンドを実行します。したがって、特定のソースに対して他と異なる機能を実行する場合や、他と異なるディレクトリに置かれたソースを処理する場合などは、そのソースの依存リストのみ、コマンドを持つ通常の形で記述します。

以下に、サフィックス定義なしの依存リストと、サフィックス定義付きの依存リストの例を示します。

サフィックス未定義の依存リスト

```
# dependency list start

### src definition start
SRC1_DIR= .
### src definition end

$(TARGET).elf : $(OBJS) $(TARGET).mak $(TARGET).lds
    $(LD) $(LDFLAGS) -o $$@ $(OBJS) $(OBJLDS) $(LIBS)

## boot.s
boot.o : $(SRC1_DIR)/boot.s
    $(AS) $(ASFLAGS) -o boot.o $(SRC1_DIR)/boot.s

## main.c
main.o : $(SRC1_DIR)/main.c
    $(CC) $(CFLAGS) $(SRC1_DIR)/main.c

# dependency list end
```

サフィックスを定義した例

```
# suffix & rule definitions
.SUFFIXES : .c .s .o .elf

.c.o :
    $(CC) $(CFLAGS) -o $(SRC_DIR)/$.o $(SRC_DIR)/$.c
.s.o :
    $(AS) $(ASFLAGS) -o $(SRC_DIR)/$.o $(SRC_DIR)/$.s

# dependency list start

### src definition start
### src definition end

$(TARGET).elf : $(OBJS) $(TARGET).mak $(TARGET).lds
    $(LD) $(LDFLAGS) -o $$@ $(OBJS) $(OBJLDS) $(LIBS)

## boot.s
boot.o : $(SRC_DIR)/boot.s

## main.c
main.o : $(SRC_DIR)/main.c

## sub.c
sub.o : $(SRC_DIR)/sub.c

# dependency list end
```

サフィックス定義

サフィックスルール

依存リスト

● サフィックス定義の形式

拡張子の指定

形式: **.SUFFIXES** : .xxx .yyy .zzz

例: **.SUFFIXES** : .c .s .o .elf

サフィックスルールの適用に関係する拡張子をすべて指定します。

サフィックスルールの定義

サフィックスルールの形式は次のとおりです。

形式: **<依存ファイル1の拡張子>.<ターゲットファイルの拡張子>**:

```

                TAB      <コマンド1>
[                TAB      <コマンド2>
                :
```

例: **.c.o** :

```
$(CC) $(CFLAGS) -o $(SRC_DIR)/$.o $(SRC_DIR)/$.c
```

- **\$.***は依存リストのターゲットファイル名(拡張子なし)に置き換えられるマクロです。
- **<コマンド>**の前にはタブ(スペースは不可)が必要です。

この例のサフィックスルールは次のように、ターゲットファイルが.oおよび最初の依存ファイルが.cの依存リストに対応します。

<ファイル1>.o: <ファイル1>.c [<その他の依存ファイル>]

その依存リストのコマンドラインを省略した場合に、サフィックスルールのコマンドが実行されるようになっています。

例: 依存リスト

```
## main.c
main.o : $(SRC_DIR)/main.c

## sub.c
sub.o : $(SRC_DIR)/sub.c
```

この2つの依存リストには前記の例のサフィックスルール(**.c.o**)が適用され、以下の依存リストとして処理されます。

```
## main.c
main.o : $(SRC_DIR)/main.c
        $(CC) $(CFLAGS) -o $(SRC_DIR)/main.o $(SRC_DIR)/main.c

## sub.c
sub.o : $(SRC_DIR)/sub.c
        $(CC) $(CFLAGS) -o $(SRC_DIR)/sub.o $(SRC_DIR)/sub.c
```

サフィックスルールが適用された場合も依存ファイルの作成時間がチェックされ、ターゲットファイルの方が新しい場合、コマンドは実行されません。

● サフィックス定義使用上の注意事項

サフィックス定義を使用する場合、依存リスト内のターゲットファイル名と最初の依存ファイル名は拡張子を除き、同一(大文字/小文字を区別)である必要があります。

悪い例: **main.o : main1.c**

この場合、サフィックスルールは適用されません。メイクはこのような依存リストを無視し、何の処理も実行しません。

11.1.8 clean

IDEが作成するmakeファイルには、ソース以外の中間ファイルおよびオブジェクトファイルを削除するコマンドが記載されています。ターゲット名`clean`を指定してメイクを行うと(IDEでは[プロジェクト]メニューから[クリーン...]を選択)、このコマンドが実行されるようになっています。

記載されたコマンドは次のとおりです。

例:

```
TARGET= sample
:
TOOL_DIR = c:/EPSON/gnu33
:
RM= $(TOOL_DIR)/rm
:
OBS= boot.o ¥
      main.o ¥
:
DEPS = $(OBS:%.o=%.d)
clean:
    $(RM) -f $(OBS) $(TARGET).elf $(TARGET).map $(DEPS)
```

ファイル削除ユーティリティ **rm.exe**を使用して、メイクにより生成された`.o`ファイル、`.elf`ファイルおよび`.map`ファイルを削除します。

11.1.9 sh.exeによる起動

makefile内のコマンド行に、セミコロン(;)、切り替え記号(<, >, >>, |)、置換記号(*, ?, [], \$, =)、引用符、エスケープ文字、コメント("、'、\、¥、#など、:)などの、シェルのメタキャラクタが含まれている場合、makeはBourneシェルを起動してコマンド行を処理します。

シェルによるコマンド行の構文解析が必要ない場合は、makeはコマンドを直接実行します。また、このときのBourneシェルは、make.exeが存在する位置のsh.exeとします。

11.1.10 メッセージ

make.exeが出力するメッセージを以下に示します。

表11.1.10.1 通常メッセージ

メッセージ	内容
make: Nothing to be done for 'TARGET'.	<i>TARGET</i> を作成するさいに何も実行しませんでした。このエラーはターゲットを作成する実行内容が空の場合に発生します。
make: 'FILENAME' is up to date.	<i>FILENAME</i> はすでに最新の状態で更新されています。メイク処理は行われませんでした。

表11.1.10.2 エラーメッセージ

エラーメッセージ	内容
make: *** No rule to make target 'FILENAME1', needed by 'FILENAME2'. Stop.	<i>FILENAME2</i> を作成するための <i>FILENAME1</i> が見つからないか、 <i>FILENAME1</i> を作成する依存リストが定義されていません。メイク処理は中断されました。
make: <i>TOOL</i> : Command not found	<i>FILENAME</i> を作成する依存リストに記述された <i>TOOL</i> が見つかりませんでした。メイク処理は中断されます。
make: *** [<i>FILENAME</i>] Error 127	
make: *** [<i>FILENAME</i>] Error 1	呼び出したツールでエラーが発生しました。メイク処理は中断されました。
XXX.mak:LINE: *** missing separator. Stop.	XXX.makのLINE行目に不正な区別記号が入っています。メイク処理は中断されました。このエラーは実行内容の先頭がTABでなく空白の場合などに発生します。
XXX.mak:LINE: *** commands commence before first target. Stop.	XXX.makのLINE行目に最初のターゲット定義より前にコマンドが記述されています。make処理は中断されました。このエラーは最初のターゲットallなどよりも前にTABが記述されている場合に発生します。

表11.1.10.3 ワーニングメッセージ

ワーニングメッセージ	内容
make: *** Warning: File 'FILENAME' has modification time in the future (yyyy-mm-dd hh:mm:ss > yyyy-mm-dd hh:mm:ss)	<i>FILENAME</i> が未来の時間に更新されています。時刻設定に誤りが検出されました。メイク処理は正常に実行されていない可能性があります。
make: warning: Clock skew detected. Your build may be incomplete.	

11.1.11 注意事項

- 本パッケージの**make.exe**では、本書で説明した以外の機能については動作の保証はできません。
- 1つの**make**ファイルから呼び出せる実行ファイルの引数は最大で3万文字です。このため、**make**ファイルに長い名前のファイルを大量に追加した場合、リンク時にエラーが発生する可能性があります。

11.2 ccap.exe

11.2.1 機能

各ツールやコマンドがコンソール(標準出力あるいは標準エラー)に対して出力するメッセージをファイル化します。

11.2.2 出力ファイル

●メッセージファイル

ファイル形式: テキストファイル

ファイル名: <ファイル名>.err

内容: 取得したメッセージを書き出したテキストファイルです。

11.2.3 使用方法

●起動フォーマット

ccap [**<オプション>**] **<出力ファイル名>** "**<実行コマンド>**"

[]は省略可能なことを示します。

<出力ファイル名>: メッセージを出力するファイル名を指定します。

<実行コマンド>: 実行するツールの起動コマンドを入力します。

●オプション

ccapには4種類の起動時オプションが用意されています。

-a

機能: 既存ファイルへの追加

説明: -aオプションを指定すると出力内容を既存のファイルの後に追加します。
ファイルが存在しない場合は、新規に作成します。

デフォルト: -aオプションが指定されない場合、指定のファイルに上書き(ファイルが存在する場合)、または新規作成(ファイルが存在しない場合)します。

-o

機能: ファイルのみ出力

説明: 実行コマンドのメッセージをファイルにのみ出力します。コンソールには出力しません。
デフォルト: -oオプションが指定されない場合、コンソールとファイルの両方に出力します。

-c

機能: 実行コマンドラインの出力を停止

説明: -cオプションが指定されると**ccap**は実行コマンドラインをコンソールおよびファイルに出力しません。

デフォルト: -cオプションが指定されない場合、実行コマンドラインも出力されます。

-e

機能: エラーカウント

説明: -eオプションが指定されると実行コマンドが出力したエラーメッセージの数をカウントして出力します。

カウントの対象は、次の文字列で始まるメッセージです。

Error エラー数のカウント

Warning ワーニング数のカウント

デフォルト: カウントは行いません。

オプションの前後には1個以上のスペースが必要です。

例: c:\¥EPSON¥gnu33¥ccap -a -o -e Compile.err "xgcc -c -gstabs test.c"

●Usage出力

ファイル名や実行コマンドが指定されなかったり、オプション指定が正しくない場合、次の使用方法のメッセージを出力して終了します。

```
ccap
Console Capture Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
    ccap [options] <output-file> "command line"
Options:
    -a : append mode
    -o : disable console output
    -c : disable command echo
    -e : display error count
Example:
    ccap -a -o -c console.cap "gcc sample.c"
```

11.2.4 エラーメッセージ

ccapのエラーメッセージは次のとおりです。

表11.2.4.1 エラーメッセージ

エラーメッセージ	内容
Error: Cannot execute	指定した"実行コマンド"が実行できません。
Error: Cannot open output file	出力ファイルがオープンできません。

11.3 objdump.exe

11.3.1 機能

objdumpはelf形式バイナリファイルの内部データを表示するツールです。逆アセンブルコード、生データ、セクション構成、セクション配置アドレスとデータサイズ、リロケータブル情報シンボルテーブルを表示することができます。

11.3.2 入力ファイル

● 実行形式オブジェクトファイル

ファイル形式: elf形式のバイナリファイル

ファイル名: <ファイル名>.elf

内容: リンカにてリンク処理が終了した後の実行形式ファイルです。
ダンプ時のアドレスは、絶対アドレス表示になります。

● オブジェクトファイル

ファイル形式: elf形式のバイナリファイル

ファイル名: <ファイル名>.o

内容: アセンブル後のオブジェクトファイルです。
ダンプ時のアドレスは、ファイル先頭またはセクション先頭からの相対アドレス表示になります。

11.3.3 使用方法

● 起動フォーマット

objdump <オプション> <入力ファイル名>

<入力ファイル名>: ダンプするオブジェクトファイル名を指定します。

● オプション

以下の起動時オプションが指定可能です。

-d

機能: 逆アセンブル表示

説明: ファイル内の実行可能なすべてのセクションを逆アセンブルして表示します。ソースファイルの混合表示は行いません。

-h

機能: セクション情報表示

説明: セクション構成とそのサイズ、およびアドレスを表示します。

-g

機能: デバッグ情報の変換表示

説明: デバッグ情報を元にソースとアドレスとの関連を表示します。また、グローバルシンボルのデータ型も表示します。

-t

機能: グローバルシンボル情報表示

説明: グローバルシンボルの一覧を表示します。これにはローカルラベルも含まれます。

-s

機能: 16進ダンプ表示

説明: すべてのセクション情報を16進数で表示します。未解決シンボルに相当するデータの場合、正しい値は表示されません。

-D機能: **全セクションの逆アセンブル表示**

説明: すべてのセクションの逆アセンブル情報を表示します。

-G機能: **デバッグ情報の生データ表示**

説明: stab形式のデバッグ情報の生データを表示します。

-S機能: **ミックス表示**

説明: ファイル内の実行可能なすべてのセクションを逆アセンブルして表示します。可能な場合はソースとの混合表示を行います。

オプションの前後には1個以上のスペースが必要です。

例: `c:¥EPSON¥gnu33¥objdump -S test.elf`

11.3.4 ダンプフォーマット

各オプション指定による表示例を以下に示します。

-d (逆アセンブル表示)

実行可能なセクションの先頭から逆アセンブル情報を表示します。

```
C:¥EPSON¥gnu33¥>objdump -d main.o
main.o:      file format elf32-c33
```

```
Disassembly of section .text:
```

```
00000000 <main>:
    0: 0200  pushn    %r0                pushn    %r0

00000002 <.LBB2>:
    2: 6c04  ld.w      %r4,0x0            ld.w      %r4,0x0
    4: c000  ext        0x0
    6: c000  ext        0x0
    8: 3cf4  ld.w      [%r15],%r4            xld.w     [%r15+0x0],%r4
....
```

-h (セクション情報表示)

ファイル内のセクション構成、サイズ、配置アドレス(VMAおよびLMA)が表示されます。

.stab、.stabstr、.commentはデバッグ情報が入ったセクションです。

デバッグのloadコマンド実行時、これらのデバッグ情報のセクションはターゲットには送られません。

```
C:¥EPSON¥gnu33¥>objdump -h test.elf
```

```
test.elf:      file format elf32-c33
```

```
Sections:
```

Idx	Name	Size	VMA	LMA	File off	Algn
0	.bss	00000004	00000000	00000000	00000094	2**2
	ALLOC					
1	.data	00000008	00000004	00c000c4	00000158	2**2
	CONTENTS, ALLOC, LOAD, DATA					
2	.text	000000bc	00c00000	00c00000	00000094	2**2
	CONTENTS, ALLOC, LOAD, CODE					
3	.rodata	00000008	00c000bc	00c000bc	00000150	2**2
	CONTENTS, ALLOC, LOAD, DATA					
4	.stab	00000288	00c000c4	00c000c4	00000160	2**2
	CONTENTS, READONLY, DEBUGGING					
5	.stabstr	00000338	00c0034c	00c0034c	000003e8	2**0
	CONTENTS, READONLY, DEBUGGING					
6	.comment	0000004c	00c00684	00c00684	00000720	2**0
	CONTENTS, READONLY					

```
*** Debugging sections will not be loaded to the target ***
```

-g (デバッグ情報の変換表示)

ソース行と実行時のアドレスとの対比を表示します。表示フォーマットは次のとおりです。

```
/* file <ファイル名> line <対応ソース行番号> addr 0x<対応アドレス> */
```

ここではグローバルシンボルのデータ型も表示されます。

```
C:\¥EPSON¥gnu33¥>objdump -g test.elf
```

```
test.elf:      file format elf32-c33

boot.s:
/* file boot.s line 9 addr 0xc00004 */
/* file boot.s line 10 addr 0xc00008 */
/* file boot.s line 11 addr 0xc0000a */
/* file boot.s line 12 addr 0xc00010 */
/* file boot.s line 13 addr 0xc00016 */
./main.c:
typedef int32 int;
typedef int8 char;
typedef int32 long int;
typedef uint32 unsigned int;
.....
typedef void void;
long int aaaa /* 0xc000bc */;
long int bbbb /* 0x4 */;
long int aaaa2 /* 0xc000c0 */;
long int bbbb2 /* 0x8 */;
int main ()
{ /* 0xc0001c */
  /* file ./main.c line 12 addr 0xc0001c */
  { /* 0xc0001e */
    register int j /* 0x0 */;
    /* file ./main.c line 13 addr 0xc0001e */
    /* file ./main.c line 15 addr 0xc0001e */
    /* file ./main.c line 16 addr 0xc00026 */
    /* file ./main.c line 18 addr 0xc00028 */
    /* file ./main.c line 19 addr 0xc00030 */
    /* file ./main.c line 16 addr 0xc00030 */
  } /* 0xc00034 */
  /* file ./main.c line 20 addr 0xc00034 */
} /* 0xc00038 */
.....
int i /* 0x0 */;
```

-t (グローバルシンボル情報表示)

内部的なシンボルを含めたグローバルシンボル一覧を表示します。

表示フォーマットは次のとおりです。

SYMBOL TABLE:

<実行時のアドレス> <ローカル/グローバル> <シンボル種別> <配置セクション> <データサイズ> <シンボル名>

```
C:\¥EPSON¥gnu33¥>objdump -t test.elf
```

```
test.elf:      file format elf32-c33
```

SYMBOL TABLE:

```
00000000 1      d  .bss  00000000
00000004 1      d  .data  00000000
.....
```

-s (16進ダンプ表示)

各セクションの生データ表示を行います。表示フォーマットは次のとおりです。

Contents of section <セクション名>

<アドレス> <生データ> <生データ> <生データ> <生データ> <データのASCII表示>

.....

<アドレス> <生データ> <生データ> <生データ> <生データ> <データのASCII表示>

Contents of section <セクション名>

<アドレス> <生データ> <生データ> <生データ> <生データ> <データのASCII表示>

.....

```
C:\¥EPSON¥gnu33¥>objdump -s test.elf
```

```
test.elf:          file format elf32-c33
```

```
Contents of section .data:
```

```
0004 ffffffff 7f7f7f7f .....

```

```
Contents of section .text:
```

```
c00000 0400c000 20c00f6c f1a000c0 00c00f6c .... .l.....l
c00010 00c000c0 041cf8df ffdff51e 0002046c .....l
c00020 00c000c0 f43c402e 00c000c0 061d062e .....<@.....
c00030 fc1f1060 40024006 16700818 00c000c0 ...`@.@..p.....
c00040 f4301460 00c000c0 f43c4006 0302622e .0.`.....<@...b.
c00050 732e802e 912e272e ffc3f86f 762e8632 s.....'....ov..2
c00060 87888788 042e452e 85328488 848856aa .....E..2....V.
.....
```

-D (全セクションの逆アセンブル表示)

-dオプションを全セクションに適用、表示したものです。表示フォーマットは-dと同じとなります。

-g (デバッグ情報の生データ表示)

デバッグ情報の生データを表示します。通常はこのデータを使用することはありません。表示内容についてはGNU各種ドキュメントもしくはソースコードを参照してください。

```
Contents of .stab section:
```

```
Symnum n_type n_othr n_desc n_value n_strx String
```

```
-1      HdrSym 0      53      00000338 1
0       SO     0      0       00c00000 1      boot.s
1       SLINE  0      9       00c00004 0
2       SLINE  0      10      00c00008 0
3       SLINE  0      11      00c0000a 0
4       SLINE  0      12      00c00010 0
. . . . .
```

-S (ソース逆アセンブルのミックス表示)

実行可能なセクションの逆アセンブル情報とソースを混合表示します。

```
C:\¥EPSON¥gnu33¥>objdump -S test.elf
```

```
test.elf:          file format elf32-c33
```

```
Disassembly of section .text:
```

```
00c00000 <__START_text>:
   c00000: 0004  nop                      ***
   c00002: 00c0  pops      %psr             ***
```

```
00c00004 <boot>:
```

```
.align 2
.global boot
.long    boot
```

```
boot:
```

11 その他のツール

```
xld.w    %r15,0x0800
c00004: c020  ext      0x20
c00006: 6c0f  ld.w      %r15,0x0      xld.w    %r15,0x800
      ld.w    %sp,%r15      ; set SP
c00008: a0f1  ld.w      %sp,%r15      ld.w    %sp,%r15
      xld.w   %r15,__dp      ; set default data area pointer
.....
00c0001c <main>:
const long aaaa2 =0x87654321;
long bbbb2 =0x7f7f7f7f;

main()
{
  c0001c: 0200  pushn    %r0              pushn    %r0

00c0001e <.LBB2>:
  int j;

  i = 0;
  c0001e: 6c04  ld.w      %r4,0x0      ld.w      %r4,0x0
  c00020: c000  ext      0x0
  c00022: c000  ext      0x0
  c00024: 3cf4  ld.w      [%r15],%r4      xld.w      [%r15+0x0],%r4

00c00026 <.LM4>:
  for (j=0 ; ; j++)
  c00026: 2e40  ld.w      %r0,%r4      ld.w      %r0,%r4
```

11.3.5 エラーメッセージ

objdumpのエラーメッセージは次のとおりです。

表11.3.5.1 エラーメッセージ

エラーメッセージ	内容
/cygdrive/X/path to objdump/objdump: filename: File format not recognized	認識できないファイル(filename)が指定されました。 elf形式のファイルを指定してください。

11.3.6 注意事項

- 逆アセンブル表示を行った場合、表示する情報量が極端に多いと表示は途中で終了します。
- リンク前の.oファイルのダンプを行う場合、表示されるアドレスは絶対アドレスでなく、各セクションの先頭アドレスからの相対アドレス表示となります。また、各セクションの先頭アドレスは0x0番地として扱われます。
- mc33advオプション(S1C33401用)を指定してビルドした.oファイルやelfファイルを-dオプションでディスアセンブルした場合、ald命令は逆アセンブルされません。

例: ext 0x0
ld.w %r1,0x30
ld.w %r1,[%r1]
は、
xld.w %r1
ld.w %r1,[%r1]
として逆アセンブル表示されます。

ald.w %r1,0x30
として逆アセンブル表示されません。

11.4 objcopy.exe

11.4.1 機能

objcopyはgnu標準のオブジェクトファイル形式変換ユーティリティで、オブジェクトファイルのコピーやデータ形式の変換に使用します。

S1C33 Familyのアプリケーション開発においては、elf形式オブジェクトファイルをROM書き込み用のモトローラS3形式HEXファイルに変換するのに使用します。

objcopyは多種の機能(オプション)、ファイル形式に対応していますが、ここではelf形式ファイル→HEX形式ファイル変換に絞って説明を行います。**objcopy**の詳細については、gnuのユーティリティに関する資料を参照してください。

11.4.2 入出力ファイル

入力ファイル

●オブジェクトファイル

ファイル形式: elf形式のバイナリファイル

ファイル名: <ファイル名>.elf

内容: リンカにてリンク処理が終了した後の実行形式ファイルです。

出力ファイル

●HEXファイル

ファイル形式: モトローラS3形式ファイル

ファイル名: <ファイル名>.sa

内容: ROMに書き込むためのHEXデータファイルです。複数のROMを使用するシステムでは、elf形式オブジェクトファイル内の、各ROMに書き込むセクションのみをそれぞれ変換してファイルを作成します。

11.4.3 使用方法

●起動フォーマット

`objcopy <オプション> <入力ファイル名> [<出力ファイル名>]`

[]は省略可能なことを示します。

<入力ファイル名>: 変換するelf形式オブジェクトファイル名を指定します。

<出力ファイル名>: 変換後のモトローラS3形式HEXファイル名を指定します。

注: <出力ファイル名>を省略すると、**objcopy**は一時ファイルを作成して処理を実行後、そのファイルの名称を入力ファイルの名称に変更します。したがって、入力ファイルは削除されます。

●オプション

S1C33 Familyのアプリケーション開発においては、主に以下のオプションを使用します。

-I elf32-little

機能: 入力ファイル形式指定

説明: 入力ファイルのフォーマットをelfに指定します。

-O srec

機能: モトローラ形式出力

説明: 出力ファイルのフォーマットをモトローラ形式に指定します。'-I elf32-little'と一緒に指定してください。

-O binary

機能: バイナリ形式出力

説明: 出力ファイルのフォーマットをバイナリ形式に指定します。'-I elf32-little'と一緒に指定してください。

--srec-forceS3

機能: モトローラS3形式指定

説明: 出力ファイルのレコードフォーマットをモトローラS3形式にします。-O srecオプションと共に指定します。

例: ... -O srec --srec-forceS3 ...

-R SectionName

機能: セクションの削除

説明: SectionNameという名称のセクションを出力ファイルに含めないことを指定します。このオプションは複数回の指定が可能です。'-I elf32-little'と一緒に指定してください。

-v(または--verbose)

機能: 詳細出力モード

説明: 修正されたすべてのオブジェクトファイルを表示します。

-V(または--version)

機能: バージョン表示

説明: **objcopy**のバージョンを表示して終了します。

--help

機能: Usage出力

説明: **objcopy**のusageを表示して終了します。

11.4.4 HEXファイルの作成方法

コマンドプロンプトウィンドウを開き、次のコマンドラインで**objcopy**を実行します。

```
C:\¥EPSON¥gnu33¥>objcopy -I elf32-little -O srec -R SectionName --srec-forceS3
InputFile OutputFile
```

上記のコマンドにより、-Rオプションで指定した以外のセクションがS3レコードに変換され、出力ファイルが生成されます。

例: input.elfからすべてのセクションのデータを抜き出し、output.saに書き出します。

```
C:\¥EPSON¥gnu33¥>objcopy -I elf32-little -O srec --srec-forceS3 input.elf
output.sa
```

11.5 ar.exe

11.5.1 機能

arはアーカイブファイルとしてまとめられたファイル群を保守するための**gnu**標準のユーティリティです。通常、リンカ**ld**が使用するライブラリファイルの作成、更新に使用します。**ar**の詳細については、**gnu**のユーティリティに関する資料を参照してください。

11.5.2 入出力ファイル

●オブジェクトファイル

ファイル形式: elf形式のバイナリファイル

ファイル名: <ファイル名>.o

内容: リロケータブルオブジェクトファイルです。

arはこの形式のオブジェクトファイルを入力してアーカイブ(ライブラリ)に追加、あるいはアーカイブからオブジェクトを抽出してこの形式のファイルを作成します。

●アーカイブファイル(ライブラリファイル)

ファイル形式: バイナリ形式のアーカイブファイル

ファイル名: <ファイル名>.a

内容: リンカ**ld**に入力可能なライブラリファイルです。

11.5.3 使用方法

●起動フォーマット

ar <キー> [<修飾子>] [<追加位置>] <アーカイブ> [<オブジェクト>]

[]は省略可能なことを示します。

<キー>、<修飾子>: **ar**の処理内容を指定します。

<追加位置>: <オブジェクト>を追加する位置をアーカイブ中のオブジェクト名で指定します。

<アーカイブ>: 編集するアーカイブファイル名を指定します。

<オブジェクト>: 追加、抽出、移動、削除するオブジェクトファイル名を指定します。複数のオブジェクトファイルをスペースで区切って指定可能です。

●キー

- d** <オブジェクト>をアーカイブファイルから削除します。
- m** <オブジェクト>をアーカイブファイルの最後に移動します。修飾子aまたはbと併用してアーカイブ中の指定位置に移動することも可能です。
- q** <オブジェクト>をアーカイブファイルの最後に追加します。アーカイブファイル内のシンボルテーブルは更新されません。
- r** アーカイブファイル中の<オブジェクト>を置き換えます。アーカイブファイルに同名のオブジェクトが存在しない場合はファイルの最後に追加されます(修飾子aまたはbと併用して追加位置の指定も可能)。
- t** アーカイブファイルの内容一覧、または指定した<オブジェクト>の一覧を表示します。
- x** アーカイブファイルから指定した<オブジェクト>を抽出してファイルを作成します。<オブジェクト>が指定されない場合、アーカイブファイル内のすべてのオブジェクトが取り出されます。

●修飾子

- a** rまたはmキーと併用して<オブジェクト>をアーカイブファイル中の<追加位置>の後に置きます。<追加位置>にはアーカイブファイル内のオブジェクト名を指定します。
- b** aと同様ですが、<オブジェクト>を<追加位置>の前に置きます。
- s** アーカイブファイルのシンボルテーブルを強制的に更新します。
- u** rキーと併用し、<オブジェクト>の中でアーカイブファイルに収めた後に変更があったもののみを置き換えます。
- v** 冗長モード。実行した動作を表示します。

キーと修飾子は、間に空白を入れずにつなげて記述してください。

●使用例

(1)アーカイブファイルの新規作成

```
ar rs mylib.a func1.o func3.o
(mylib.a: func1.o + func3.o)
```

アーカイブファイル(mylib.a)が存在しない場合は新規に作成され、指定のオブジェクトファイル(func1.o、func3.o)が指定順にすべて追加されます。

(2)オブジェクトの追加

```
ar rs mylib.a func4.o func5.o
(mylib.a: func1.o + func3.o + func4.o + func5.o)
```

mylib.aの最後にfunc4.o、func5.oを追加します。

(3)指定位置へのオブジェクトの追加

```
ar ras func1.o mylib.a func2.o
(mylib.a: func1.o + func2.o + func3.o + func4.o + func5.o)
```

mylib.a内のfunc1.oの後ろにfunc2.oを追加します。

(4)オブジェクトの置き換え

```
ar rus mylib.a func1.o func2.o func3.o func4.o func5.o
(mylib.a: func1.o + func2.o + func3.o + func4.o + func5.o)
```

func1.o、func2.o、func3.o、func4.o、func5.oの中で、mylib.aに追加した後に更新されたファイルがあれば、そのオブジェクトのみ置き換えられます。

(5)オブジェクトの抽出

```
ar x mylib.a func5.o
(mylib.a: func1.o + func2.o + func3.o + func4.o + func5.o)
```

mylib.aからfunc5.oを取り出してオブジェクトファイルを作成します。アーカイブファイルの内容は変更されません。

(6)オブジェクトの削除

```
ar ds mylib.a func5.o
(mylib.a: func1.o + func2.o + func3.o + func4.o)
```

mylib.aからfunc5.oを削除します。

11.6 moto2ff.exe

11.6.1 機能

moto2ffはモトローラS3形式ファイルを読み込み、ファイル内のデータが存在しない空きアドレスを0xffのデータで埋めて出力ファイルを作成します。開始アドレスとブロックサイズを指定し、必要な領域のみを出力することができます。

S1C33 Familyのアプリケーション開発においては、**objcopy**により生成されたHEXファイルをセイコーエプソンに提出していただくマスクROMデータに変換するために使用します。

11.6.2 入出力ファイル

入力ファイル

●HEXファイル

ファイル形式: モトローラS3形式ファイル

ファイル名: <ファイル名>.sa

内容: 実行形式のelfファイルを**objcopy**で変換したモトローラS3形式のHEXファイルです。

出力ファイル

●マスクROMデータファイル

ファイル形式: モトローラS3形式ファイル(空き領域 = 0xff)

ファイル名: <ファイル名>.saf

内容: セイコーエプソンに提出していただくマスクROMデータファイルです。

11.6.3 起動フォーマット

moto2ff <データの開始アドレス> <データのブロックサイズ> <入力ファイル名>

<データの開始アドレス>: 入力ファイル内のアドレスで、ここからデータの出力を開始します。16進数値で指定します。

<データのブロックサイズ>: 出力するデータのブロックサイズ(バイト単位)です。16進数値で指定します。

<入力ファイル名>: 0xff埋めを行うモトローラS3形式HEXファイル名を指定します。
入力ファイル名はパス名、拡張子含めて128文字以内です。入力ファイルはパスの指定ができますが、出力ファイルはカレントディレクトリに出力されます。

- 引数の指定がない場合は使用方法メッセージが表示されます。
- 同名の出力ファイルがすでにあった場合は上書きされます。
- エラー発生時はエラーメッセージが表示され、出力ファイルは生成されません。
- 入力したモトローラS3形式ファイルに、開始アドレスとブロックサイズによって指定した範囲外のデータがある場合、警告メッセージが出力されます。出力ファイルは指定範囲のデータで生成されます。
- 入力したモトローラS3形式ファイルにアドレスが重複したデータがあった場合は、後のデータで上書きされ出力されます。
- 正常に終了した場合、以下のメッセージが標準出力に出力されます。
Convert Completed

11.6.4 エラー /ワーニングメッセージ

moto2ffが出力するエラー /ワーニングメッセージを以下に示します。

表11.6.4.1 エラーメッセージ

エラーメッセージ	内容
Input filename is over 128 letters.	入力ファイル名が128文字を超えています。
Cannot open input file "FILENAME".	入力ファイルFILENAMEが開けません。
Cannot open output file "FILENAME".	出力ファイルFILENAMEが開けません。
Motorola S3 checksum error.	モトローラS3形式ファイルの読み込みでチェックサムエラーが発生しました。
Cannot allocate memory.	メモリが確保できません。

表11.6.4.2 ワーニングメッセージ

ワーニングメッセージ	内容
"FILENAME" contains data outside of specified range ("STARTADDR": "SIZE")	FILENAMEには、指定された範囲(STARTADDRからSIZEバイト)以外のデータも含まれています。指定範囲以外のデータは出力されません。
Invalid file format in "FILENAME" line "NUMBER".	入力ファイルFILENAMEのNUMBER行目に不正な形式のデータがあります。

11.6.5 マスクROMデータの作成方法

objcopyによりモトローラS3フォーマットのHEXファイルを作成後、**moto2ff**によりマスクデータを作成します。

コマンドプロンプトウィンドウを開き、次のように**moto2ff**を実行します。

例: C:\¥EPSON¥gnu33¥>moto2ff 600000 100000 input.sa

このコマンドはinput.sa内のアドレス0x600000から始まる0x100000バイト分のデータをinput.safに出力します。input.saのアドレス0x600000から0x6fffffの範囲にある空きアドレスは、0xffのデータで埋められます。

上記手順で内蔵ROM用のマスクROMデータファイルを作成後、このファイルを使用して最終動作確認を行ってください。

動作確認終了後、マスクROMデータファイルをセイコーエプソンが指定する名称にリネームして提出してください。

ひとつのHEXファイルから2つのマスクROMデータを作成するには

上記例でinput.saにアドレス0x600000から0x6fffff以外のデータが含まれている場合は次のワーニングメッセージが出力されます。

Warning: input.sa contains data outside of specified range (600000:100000)

このワーニングメッセージが出力された場合でも出力ファイルは生成されます。

ひとつのHEXファイルから2つのマスクROMデータを作成する場合は、以下の手順に従ってください。

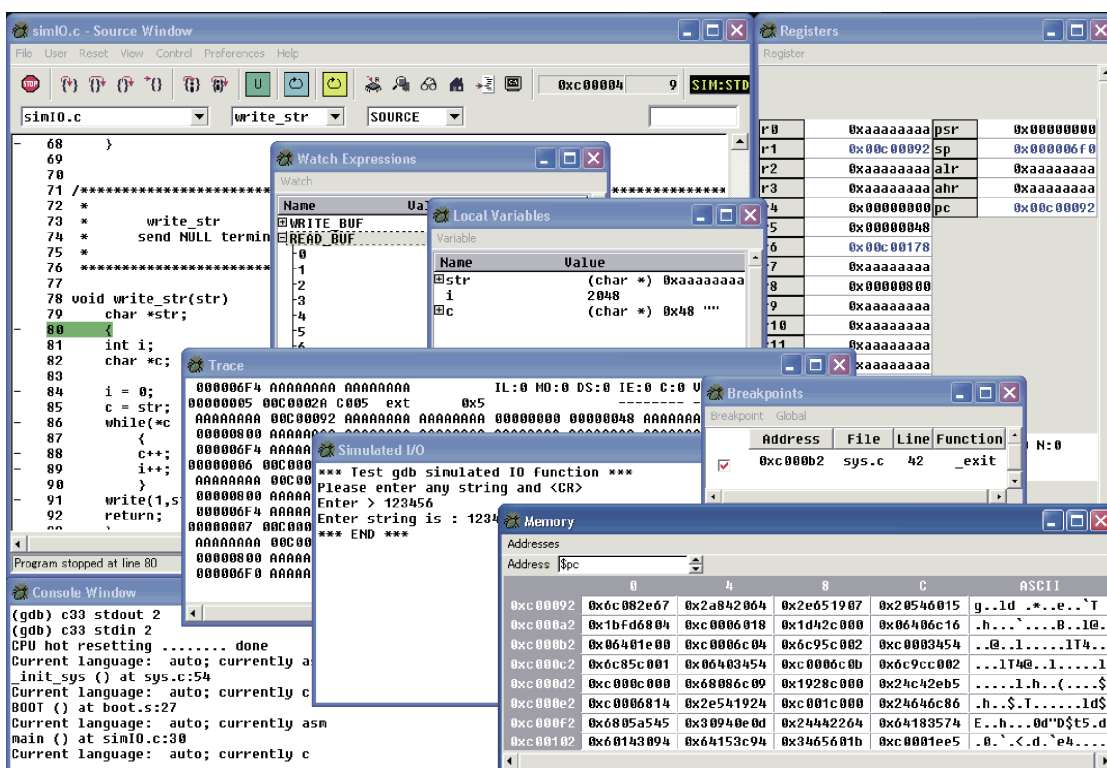
例: input.saに0x600000から始まるデータと0xc00000から始まるデータが含まれている場合

1. input.saを複製し、input2.saを作成します。
2. input.saを指定し**moto2ff**を実行します。
C:\¥EPSON¥gnu33¥>moto2ff 600000 100000 input.sa
3. input2.saを指定し**moto2ff**を実行します。
C:\¥EPSON¥gnu33¥>moto2ff C00000 100000 input2.sa

2および3の手順でワーニングが出力されますが、出力ファイルは生成されます。

上記手順によりinput.safにはアドレス0x600000からのマスクROMデータが、input2.safにはアドレス0xc00000からのマスクROMデータが出力されます。

11.7 旧バージョンのデバッガ



今回のバージョンのS1C33 Family C/C++コンパイラパッケージは、バージョン3.3.0以前のデバッガを使用することもできます。起動方法、操作方法などは、バージョン3.3.0のマニュアルを参照してください。バージョン3.3.0のマニュアルはSEIKO EPSONのユーザサイトよりダウンロードできます。

S1C33 Family C/C++ Compiler Package

Quick Reference

EPSON

CMOS 32-bit Single Chip Microcomputer
S1C33 Family C/C++ Compiler Package

Quick Reference for Development

メモリマップとベクタテーブル (C33 STDコア)

S1C33000 Core CPU

メモリマップ

		サイズ
0xFFFFFFF	エリア18 外部メモリ	64MB
	エリア17 外部メモリ	64MB
	エリア16 外部メモリ	32MB
	エリア15 外部メモリ	32MB
	エリア14 外部メモリ	16MB
	エリア13 外部メモリ	16MB
	エリア12 外部メモリ	8MB
	エリア11 外部メモリ	8MB
	エリア10 外部メモリ	4MB
	エリア9 外部メモリ	4MB
0x1000000 0x0C00000	エリア8 外部メモリ	2MB
	エリア7 外部メモリ	2MB
	エリア6 外部I/O	1MB
	エリア5 外部メモリ	1MB
0x0100000 0x0080000 0x0060000 0x0030000 0x0000000	エリア4 外部メモリ	1MB
	エリア3 内蔵ROM	512KB
	エリア2 Reserved	128KB
	エリア1 内蔵I/O	128KB
	エリア0 内蔵RAM	256KB

ベクタテーブル

	ベクタアドレス
リセット	base + 0
Reserved	base + 4~12
ゼロ除算	base + 16
Reserved	base + 20
アドレス不整例外	base + 24
NMI	base + 28
Reserved	base + 32~44
ソフトウェア例外0	base + 48
:	:
ソフトウェア例外3	base + 60
マスク可能な外部割り込み0	base + 64
:	:
マスク可能な外部割り込み215	base + 924

base: ベクタテーブル先頭アドレス

= 0x0080000 (内蔵ROMよりのブート時)

= 0x0C00000 (外部ROMよりのブート時)

レジスタ (C33 STDコア)

S1C33000 Core CPU

汎用レジスタ (16)

31	0
R15	
R14	
R13	
:	
R4	
R3	
R2	
R1	
R0	

特殊レジスタ (5)

31	0
PC	プログラムカウンタ
PSR	プロセッサステータスレジスタ
SP	スタックポインタ
ALR	算術演算ローレジスタ
AHR	算術演算ハイレジスタ

(AHR, ALR: 積和演算, 乗算, 除算用オプション)

PSR

31~12	11~8	7	6	5	4	3	2	1	0
Reserved	IL	MO	DS	—	IE	C	V	Z	N

- IL: 割り込みレベル (0~15: 割り込み許可レベル)
MO: MACオーバーフローフラグ (1: オーバーフローあり, 0: なし)
DS: 被除数符号フラグ (1: 負, 0: 正)
IE: 割り込み許可 (1: 許可, 0: 禁止)
Z: ゼロフラグ (1: ゼロ, 0: ゼロ以外)
N: ネガティブフラグ (1: 負, 0: 正)
C: キャリーフラグ (1: キャリー/ボローあり, 0: なし)
V: オーバーフローフラグ (1: オーバーフローあり, 0: なし)

メモリマップ(物理アドレス)

	サイズ
0xFFFFFFFF	
0x80000000	エリア22 外部メモリ 2GB
0x40000000	エリア21 外部メモリ 1GB
0x20000000	エリア20 外部メモリ 512MB
0x10000000	エリア19 外部メモリ 256MB
0x0C000000	エリア18 外部メモリ 64MB
0x08000000	エリア17 外部メモリ 64MB
0x06000000	エリア16 外部メモリ 32MB
0x04000000	エリア15 外部メモリ 32MB
0x03000000	エリア14 外部メモリ 16MB
0x02000000	エリア13 外部メモリ 16MB
0x01800000	エリア12 外部メモリ 8MB
0x01000000	エリア11 外部メモリ 8MB
0x00C00000	エリア10 外部メモリ 4MB
0x00800000	エリア9 外部メモリ 4MB
0x00600000	エリア8 外部メモリ 2MB
0x00400000	エリア7 外部メモリ 2MB
0x00300000	エリア6 外部メモリ 1MB
0x00200000	エリア5 外部メモリ 1MB
0x00100000	エリア4 外部メモリ 1MB
0x00080000	エリア3 内蔵RAM 512KB
0x00060000	エリア2 Reserved 128KB
0x00020000	エリア1 内蔵I/O 256KB
0x00000000	エリア0 内蔵RAM 128KB

メモリマップ(MMU論理アドレス)

0xFFFFFFFF	ブロック7 (512MB)
0xE0000000	
0xDFFFFFFF	ブロック6 (512MB)
0xC0000000	
0xBFFFFFFF	ブロック5 (512MB)
0xA0000000	
0x9FFFFFFF	ブロック4 (512MB)
0x80000000	
0x7FFFFFFF	ブロック3 (512MB)
0x60000000	
0x5FFFFFFF	ブロック2 (512MB)
0x40000000	
0x3FFFFFFF	ブロック1 (512MB)
0x20000000	
0x1FFFFFFF	ブロック0 (512MB)
0x00000000	

ベクタテーブル

	ベクタアドレス
リセット	TTBR + 0
Reserved	TTBR + 4~12
ゼロ除算	TTBR + 16
Reserved	TTBR + 20
アドレス不整例外	TTBR + 24
NMI	TTBR + 28
Reserved	TTBR + 32~44
ソフトウェア例外0	TTBR + 48
⋮	⋮
ソフトウェア例外3	TTBR + 60
マスク可能な外部割り込み0	TTBR + 64
⋮	⋮
マスク可能な外部割り込み239	TTBR + 1020

TTBR: ベクタテーブル先頭アドレス
= 0x20000000 (初期設定値)

汎用レジスタ (16)

31	0
R15	
R14	
R13	
:	
R6	
R5(AHR)	
R4(ALR)	
R3	
R2	
R1	
R0	

特殊レジスタ (13)

31	0
PC	プログラムカウンタ(R only)
SSP	スーパーバイザスタックポインタ*1
USP	ユーザスタックポインタ
DP	データポインタ
TTBR	ベクタテーブルベースレジスタ*1
SOR	シフトアウトレジスタ
LEA	ループエンドアドレス

31	0
LSA	ループスタートアドレス
LCO	ループカウンタ
AHR	算術演算ハイレジスタ
ALR	算術演算ローレジスタ
SP	スタックポインタ
PSR	プロセッサステータスレジスタ
IDIR	プロセッサ識別レジスタ(R only)
DBBR	デバッグベースレジスタ(R only)

PSR

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	HE	RM	LM	PM	RC[3:0]				—	SW	OC	SE	—	—	LC	HC
初期値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
スーパーバイザモード	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r	r/w	r/w	r/w	r	r	r/w	r/w
ユーザモード	r	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r	r/w	r/w	r/w	r	r	r/w	r/w

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	S	DE	ME	SV	IL[3:0]				MO	DS	—	IE	C	V	Z	N
初期値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
スーパーバイザモード	r/w	r	r	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r	r/w	r/w	r/w	r/w	r/w
ユーザモード	r/w	r	r	r	r	r	r	r	r/w	r/w	r	r	r/w	r/w	r/w	r/w

*1 ユーザモード時はリードオンリー

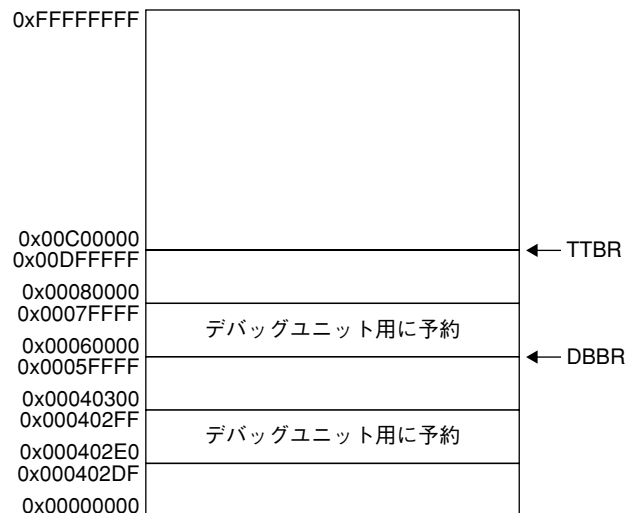
HE: HALT/SLEEPイネーブル
 RM: リピートモードイネーブル
 LM: ループモードイネーブル
 PM: PUSH/POPモード
 RC: レジスタカウンタ
 SW: スキャンワードイネーブル
 OC: オーバーフロークリアイネーブル
 SE: キャリー付きシフトイネーブル
 LC: ALRチェンジイネーブル
 HC: AHRチェンジイネーブル

(1: halt/slp命令有効, 0: 無効)
 (1: リピートモード, 0: 通常)
 (1: ループモード, 0: 通常)
 (1: push/pop実行中, 0: 通常)
 (0~15: push/pop開始レジスタ番号)
 (1: 32ビット, 0: 8ビット)
 (1: Vクリア有効, 0: 無効)
 (1: キャリー付き, 0: 無効)
 (1: ALR→R4, 0: 無効)
 (1: AHR→R5, 0: 無効)

S: 飽和
 DE: デバッグ例外
 ME: MMU例外
 SV: スーパーバイザモード
 IL: 割り込みレベル
 MO: MACオーバーフローフラグ
 DS: 被除数符号フラグ
 IE: 割り込み許可
 Z: ゼロフラグ
 N: ネガティブフラグ
 C: キャリーフラグ
 V: オーバーフローフラグ

(1: 飽和発生, 0: なし)
 (1: 例外発生, 0: なし)
 (1: 例外発生, 0: なし)
 (1: ユーザモード, 0: スーパーバイザモード)
 (0~15: 割り込み許可レベル)
 (1: オーバーフローあり, 0: なし)
 (1: 負, 0: 正)
 (1: 許可, 0: 禁止)
 (1: ゼロ, 0: ゼロ以外)
 (1: 負, 0: 正)
 (1: キャリー/ボローあり, 0: なし)
 (1: オーバーフローあり, 0: なし)

メモリマップ



ベクタテーブル

	ベクタアドレス
リセット	TTBR + 0
Reserved	TTBR + 4
ext例外	TTBR + 8
未定義命令例外	TTBR + 12
Reserved	TTBR + 16~20
アドレス不整合例外	TTBR + 24
NMI	TTBR + 28
Reserved	TTBR + 32~44
ソフトウェア例外0	TTBR + 48
:	:
ソフトウェア例外3	TTBR + 60
マスク可能な外部割り込み0	TTBR + 64
:	:
マスク可能な外部割り込み239	TTBR + 1020

TTBR: ベクタテーブル先頭アドレス
= 0xC00000 (初期設定値)

レジスタ (C33 PEコア)

S1C33000 Core CPU

汎用レジスタ (16)

31	0
R15	
R14	
R13	
:	
R6	
R5	
R4	
R3	
R2	
R1	
R0	

特殊レジスタ (8)

31	0
PC	プログラムカウンタ (R only)
DBBR	デバッグベースレジスタ (R only)
IDIR	プロセッサ識別レジスタ (R only)
TTBR	ベクタテーブルベースレジスタ
AHR	算術演算ハイレジスタ
ALR	算術演算ローレジスタ

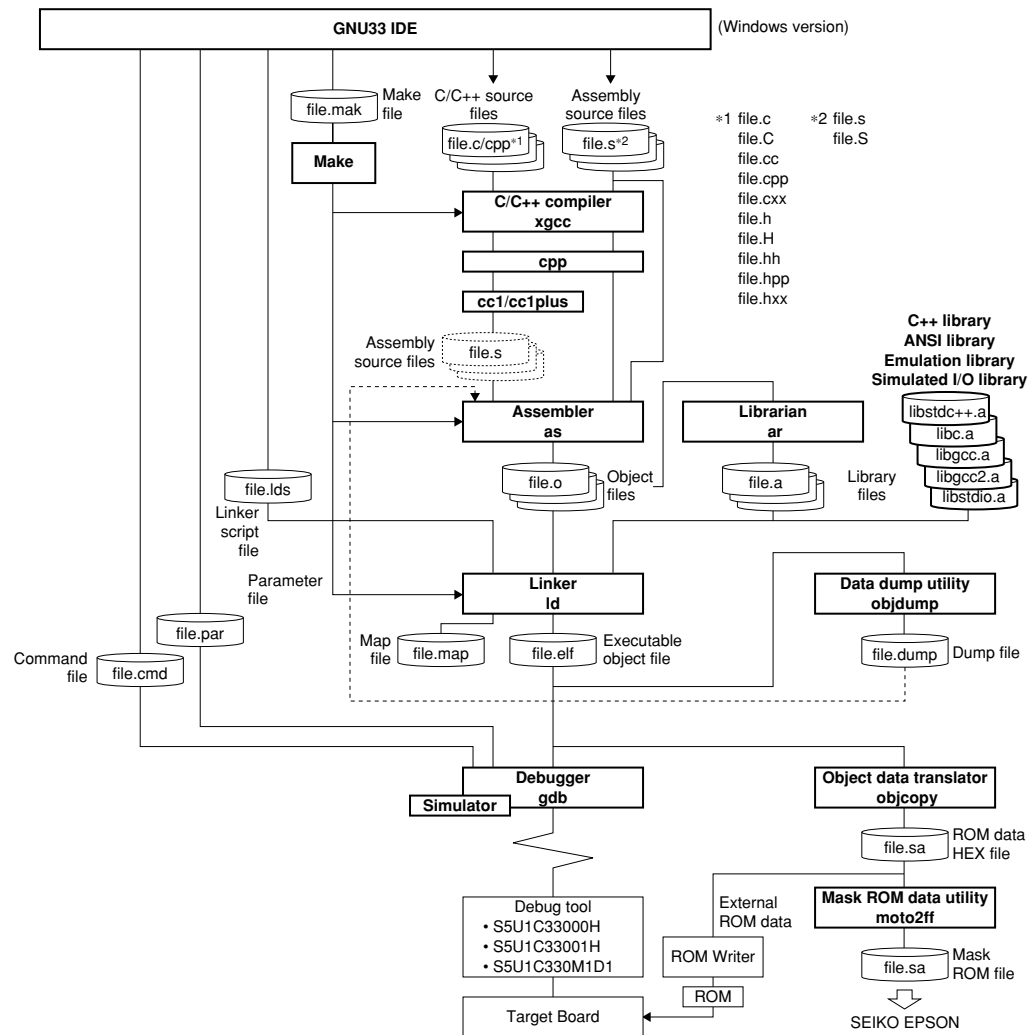
31	0
SP	スタックポインタ

PSR	プロセッサステータスレジスタ

PSR

31~12	11~8	7	6	5	4	3	2	1	0
Reserved	IL	-	-	-	IE	C	V	Z	N

IL: 割り込みレベル (0~15: 割り込み許可レベル)
 IE: 割り込み許可 (1: 許可, 0: 禁止)
 Z: ゼロフラグ (1: ゼロ, 0: ゼロ以外)
 N: ネガティブフラグ (1: 負, 0: 正)
 C: キャリーフラグ (1: キャリー/ボローあり, 0: なし)
 V: オーバーフローフラグ (1: オーバーフローあり, 0: なし)



1. プロジェクトの作成

IDEでプロジェクトを新規作成、または既存プロジェクトをインポートします。

2. ソースの編集

IDEまたは汎用のエディタを使用してソースファイルなどのリソースを編集します。

3. ビルド(コンパイル、アセンブル、リンク)

3-1) IDEのプロジェクトプロパティでビルドオプションとリンクスクリプトを編集します。(ユーザ作成のmakeファイルとリンクスクリプトファイルも使用可能です。)

3-2) IDEでビルドを実行します。

make.exeによりC/C++コンパイラ**xgcc**、アセンブラ**as**、およびリンカ**ld**が順次実行され、実行形式のオブジェクトファイル(.elf)が生成されます。

4. デバッグ

4-1) IDEのプロジェクトプロパティでパラメータファイルとデバッグ起動時に実行するコマンドファイルを編集します。

4-2) IDEからデバグガqdbを起動します。

4-3) デバッグコマンドを使用してデバッグを行います。

5. マスクデータの作成

プログラムの完成後、マスク化のためのマスクデータファイルを作成します。

5-1) objcopyとmoto2ffを使用して、マスクROMデータファイルを生成します。

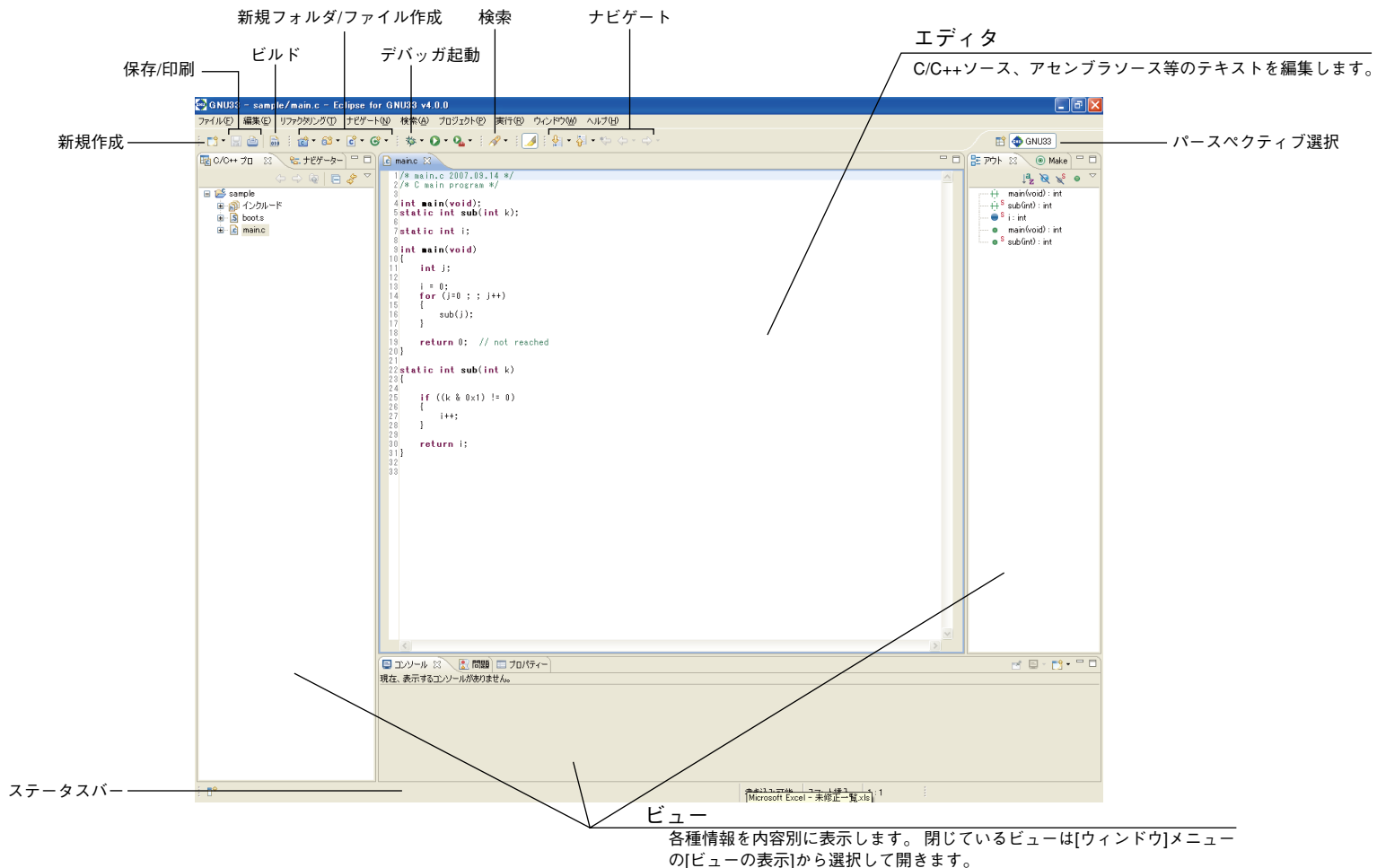
5-2) できあがったファイルをセイコーエプソンに提出してください。

概要



GNU33 IDEは、S1C33 Family C/C++コンパイラパッケージ(S5U1C33001C)を使用したプログラムの開発を支援する統合開発環境を提供します。

起動コマンド
eclipse



メニューバー

[ファイル]メニュー

ファイル(F)	
新規(N)...	Alt+Shift+N ▶
ファイルを開く(O)...	
閉じる(C)	Ctrl+W
すべて閉じる(L)	Ctrl+Shift+W
保存(S)	Ctrl+S
別名保存(A)...	
すべて保存(E)	Ctrl+Shift+S
前回保存した状態に戻す(R)	
移動(M)...	
名前変更(R)...	
リフレッシュ(F)	F5
行区切り文字の変換(V)	▶
印刷(P)...	Ctrl+P
ワークスペースの切り替え(W)	▶
再始動	
インポート(I)...	
エクスポート(E)...	
プロパティ(P)	Alt+Enter
1 main.c [sample]	
終了(X)	

[編集]メニュー

編集(E)	
元に戻す(U) / 入力	Ctrl+Z
やり直し(R)	Ctrl+Y
切り取り(C)	Ctrl+X
コピー(C)	Ctrl+C
貼り付け(P)	Ctrl+V
削除(D)	Delete
すべて選択(A)	Ctrl+A

新規 (Alt+Shift+N) プロジェクト、ファイル、フォルダを新規作成します。
ファイルを開く... エディタで開くファイルを選択します。
閉じる (Ctrl+W) エディタ最前面のファイルを閉じます。
すべて閉じる (Ctrl+Shift+W) エディタ上のすべてのファイルを閉じます。

保存 (Ctrl+S) エディタ最前面のファイルを保存します。
別名保存... エディタ最前面のファイルを別名で保存します。
すべて保存 (Ctrl+Shift+S) エディタ上のすべてのファイルを保存します。

前回保存した状態に戻す エディタ最前面のファイルの編集内容を破棄し、前回の保存内容に戻します。

移動... [C/C++ プロジェクト]/[ナビゲーター]ビューで選択されているファイル/フォルダを別の場所に移動します。

名前変更... (F2) [C/C++ プロジェクト]/[ナビゲーター]ビューで選択されているファイル/フォルダ名を変更します。

リフレッシュ (F5) [C/C++ プロジェクト]/[ナビゲーター]ビューの表示を最新の状態に更新します。

行区切り文字の変換 行の区切り文字を選択します。

印刷... (Ctrl+P) エディタ最前面のファイルを印刷します。
ワークスペースの切り替え... 別のワークスペースを選択します。

再始動 IDEを再起動します

インポート... 既存のプロジェクト/ファイルを現在のワークスペース/プロジェクトに追加します。

エクスポート... プロジェクト内のファイルを別のディレクトリに書き出します。

プロパティ (Alt+Enter) [C/C++ プロジェクト]/[ナビゲーター]ビューで選択されているプロジェクト、ファイルまたはフォルダのプロパティを表示/編集します。

終了 IDEを終了します。

元に戻す (Ctrl+Z) 直前の編集操作を取り消します。

やり直し (Ctrl+Y) [元に戻す]で取り消した操作を元に戻します。

切り取り (Ctrl+X) 選択した文字列/ファイル/フォルダをカットします。

コピー (Ctrl+C) 選択した文字列/ファイル/フォルダをコピーします。

貼り付け (Ctrl+V) カット/コピーした内容をペーストします。
削除 (Delete) 選択された文字列/ファイル/フォルダを削除します。

メニューバー

[編集]メニュー

編集(E)	
元に戻す(U) / 入力	Ctrl+Z
やり直し(R)	Ctrl+Y
切り取り(C)	Ctrl+X
コピー(C)	Ctrl+C
貼り付け(P)	Ctrl+V
削除(D)	Delete
すべて選択(A)	Ctrl+A
検索/置換(E)...	Ctrl+F
Find Word	
次を検索(N)	Ctrl+K
前を検索(O)	Ctrl+Shift+K
次をインクリメンタル検索(I)	Ctrl+J
前をインクリメンタル検索(M)	Ctrl+Shift+J
ブックマークの追加(A)...	
タスクの追加(S)...	
✓ スマート挿入モード(B)	Ctrl+Shift+Insert
ツールチップ記述の表示(Q)	F2
単語補完(W)	Alt+/
クイック・フィックス(F)	Ctrl+1
コンテンツ・アシスト(N)	Ctrl+Space
パラメーター・ヒント(H)	Ctrl+Shift+Space
右へシフト(S)	
左へシフト(B)	Shift+Tab
フォーマット(Q)	Ctrl+Shift+F
Include の追加	Ctrl+Shift+N
エンコードの設定(Q)...	

[リファクタリング]メニュー

リファクタリング(R)	
名前変更(N)...	Alt+Shift+R
定数の抽出(A)...	Alt+C
関数の抽出(E)...	Alt+Shift+M
メソッドの隠蔽	
メソッドの実装(E)...	
Getter および Setter の生成...	

すべて選択 (Ctrl+A) エディタ上の現在アクティブな文書の内容をすべて選択します。

検索/置換... (Ctrl+F) エディタ上で文字列の検索と置き換えを行います。

Find Word 選択文字列の次の一致を検索します。

次を検索 (Ctrl+K) 検索時に次の候補にジャンプします。

前を検索 (Ctrl+Shift+K) 検索時に前の候補にジャンプします。

次をインクリメンタル検索 (Ctrl+J) エディタの現在位置より後方を対象としたインクリメンタルサーチを行います。

前をインクリメンタル検索 エディタの現在位置より前方を対象としたインクリメンタルサーチを行います。

ブックマークの追加... エディタ上の現在位置をブックマークとして登録します。

タスクの追加... エディタ上の現在位置をタスク(To Do項目)に登録します。

スマート挿入モード (Ctrl+Shift+Insert) エディタのスマート挿入モードを切り替えます。

ツールチップ記述の表示 (F2) [F2]キーを押下すると、ツールチップがフォーカスされます。

単語補完 (Alt+/) エディタで入力中の文字で始まる単語を現在位置に挿入します。

クイック・フィックス (Ctrl+1) エラーやワーニングの修正候補を表示します。

コンテンツ・アシスト (Ctrl+Space) Cソースの予約語やテンプレートなどを入力アシストします。

パラメーター・ヒント (Ctrl+Shift+Space) 関数引数のヒントを表示します。

右へシフト 行の先頭を右にシフトします。

左へシフト (Shift+Tab) 行の先頭を左にシフトします。

フォーマット (Ctrl+Shift+F) フォーマッタの設定にしたがってテキストをフォーマットします。

エンコードの設定 テキストのエンコード形式を選択します。

名前変更... (Alt+Shift+R) 選択した関数や変数の名称を一括して変更します。

定数の抽出 (Alt+C) ソースファイルの定数を変数に切り出します。

関数の抽出 (Alt+Shift+M) ソースファイルのコードの一部を関数に切り出します。

メニューバー

[ナビゲート]メニュー

ナビゲート(N)		
次へジャンプ(J)		
ジャンプ(G)		
宣言を開く(O)	F3	
型階層を開く(R)	F4	
呼び出し階層を開く(I)	Ctrl+Alt+H	
組み込みブラウザーを開く(L)	Ctrl+Alt+I	
ソース/ヘッダーの切り替え(S)	Ctrl+Tab	
表示(M)	Alt+Shift+W	
クイック・アウトライン(U)	Ctrl+O	
次の注釈(N)	Ctrl+.	
前の注釈(N)	Ctrl+.	
最後の編集位置(C)	Ctrl+Q	
指定行へジャンプ(G)	Ctrl+L	
戻る(B)	Alt+←	
進む(F)	Alt+→	

[検索]メニュー

検索(A)		
C/C++(C)...		
検索(A)...	Ctrl+H	
ファイル(F)...		
テキスト(S)		

次へジャンプ [C/C++ プロジェクト]/[ナビゲーター]ビューを、現在選択しているフォルダ内のみを表示するように切り換えます。

ジャンプ [C/C++ プロジェクト]/[ナビゲーター]ビューの表示を、履歴に沿ってナビゲートします。

宣言を開く (F3) 選択しているオブジェクトの宣言・定義を開きます。

型階層を開く (F4) 選択している変数の型階層を開きます。

呼び出し階層を開く (Ctrl+Alt+H) 選択している関数の呼び出し階層を開きます。

組み込みブラウザーを開く (Ctrl+Alt+I) 選択しているソースファイルのインクルード階層を開きます。

ソース/ヘッダーの切り替え (Ctrl+Tab) 対応するソースファイルとヘッダファイルをエディタで切り替えます。

表示 (Alt+Shift+W) 選択した関数名、変数名、型を含むリソースをハイライト表示するビューを選択します。

次の注釈 (Ctrl+.) [問題]や[検索]ビューなどに表示される一覧内の選択を次の項目に進めます。

前の注釈 (Ctrl+.) [問題]や[検索]ビューなどに表示される一覧内の選択を前の項目に戻します。

最後の編集位置 (Ctrl+Q) エディタ内の最後に編集した位置にジャンプします。

指定行へジャンプ... (Ctrl+L) 行番号を指定して、エディタ上のアクティブな文書内の指定位置にジャンプします。

戻る (Alt+Left) エディタ上の文書内の移動も含め、一つ前に参照/編集していた位置に戻ります。

進む (Alt+Right) 上記の[戻る]で選った表示を一つ新しい状態に戻します。

検索... (Ctrl+H) ファイル検索、C検索を行う[検索]ダイアログを表示します。

ファイル... 指定の文字列を含むファイルを検索します。

C/C++... 指定の文字列を含むCソースを検索します。

テキスト 指定の文字列を、指定の範囲から検索します。

メニューバー

[プロジェクト]メニュー

プロジェクト(P)		
プロジェクトを開く(E)		
プロジェクトを閉じる(S)		
すべてビルド(A)	Ctrl+B	
プロジェクトのビルド(B)		
ワーキング・セットのビルド(W)		
クリーン(N)...		
自動的にビルド(M)		
プロパティ(P)		

[実行]メニュー

実行(E)		
再開(R)		
中断(S)		
終了(T)		
ステップイン(I)		
ステップ・オーバー(O)		
ステップ・リターン(U)		
[指定行まで実行(L)]		
ステップ・フィルターの使用(F)	Shift+F5	
前回の起動を実行(R)	Ctrl+F11	
前回の起動をデバッグ(D)	F11	
履歴の実行(T)		
実行(S)		
実行構成(N)...		
履歴のデバッグ(H)		
デバッグ(G)		
デバッグの構成(B)...		
外部ツール(E)		

プロジェクトを開く [C/C++ プロジェクト]/[ナビゲーター]ビューで選択されている閉じた状態のプロジェクトを開きます。

プロジェクトを閉じる [C/C++ プロジェクト]/[ナビゲーター]ビューで選択されているプロジェクトを閉じます。

すべてビルド (Ctrl+B) [C/C++ プロジェクト]/[ナビゲーター]ビュー上で開いているプロジェクトすべてのビルドを実行します。

プロジェクトのビルド [C/C++ プロジェクト]/[ナビゲーター]ビュー上で選択されているプロジェクトのビルドを実行します。

ワーキング・セットのビルド 指定のワーキングセットに含まれるリソースのビルドを実行します。

クリーン... クリーンまたはリビルドを実行します。

自動的にビルド... オートビルド機能をON/OFFします

プロパティ [C/C++ プロジェクト]/[ナビゲーター]ビュー上で選択されているプロジェクトのプロパティを表示/編集します。

前回の起動を実行 サポートとしません。

前回の起動をデバッグ 前回起動した構成でデバッグを開始します。

履歴の実行 サポートとしません。

実行構成... サポートとしません。

履歴のデバッグ サブメニューに最近起動したデバッグ構成のショートカットを表示します。

デバッグ サポートとしません。

デバッグの構成... デバッグgdbの起動構成ダイアログを開きます

外部ツール サポートとしません。

メニューバー [ウィンドウ]メニュー

ウィンドウ(W)	
新規ウィンドウ(N)	
新規エディター(E)	
パースペクティブを開く(O)	▶
ビューの表示(V)	▶
パースペクティブのカスタマイズ(C)...	
パースペクティブの別名保管(A)...	
パースペクティブのリセット(R)...	
パースペクティブを閉じる(C)	
すべてのパースペクティブを閉じる(L)	
ナビゲーション(G)	▶
設定(P)	

[ヘルプ]メニュー

ヘルプ(H)	
ヘルプ目次(H)	
検索(E)	
ダイナミック・ヘルプ(D)	
キー・アシスト(K)...	Ctrl+Shift+L
ヒント(T)...	
Cheat Sheets...	
ようこそ	
ソフトウェア更新...	
Eclipse for GNU33 v4.0.0 (について(A))	

新規ウィンドウ 新しいウィンドウを開きます。

新規エディター 編集中のファイルを新しいエディタタブで開きます。

パースペクティブを開く パースペクティブを開きます。

ビューの表示 ビューを開きます。

パースペクティブのカスタマイズ... 現在のパースペクティブの設定を変更します。

パースペクティブの別名保管... 現在のパースペクティブの設定を別名で保存します。

パースペクティブのリセット パースペクティブを初期状態に戻します。

パースペクティブを閉じる 現在のパースペクティブを閉じます。

すべてのパースペクティブを閉じる 読み込まれているすべてのパースペクティブを閉じます。

ナビゲーション エディタやビューをナビゲートします。

設定... IDEの環境設定を行います。

ヘルプ目次 ヘルプコンテンツを表示します。

検索 ヘルプトピックの検索ビューを表示します。

ダイナミック・ヘルプ アクティブなビューに関連するヘルプトピックを表示します。

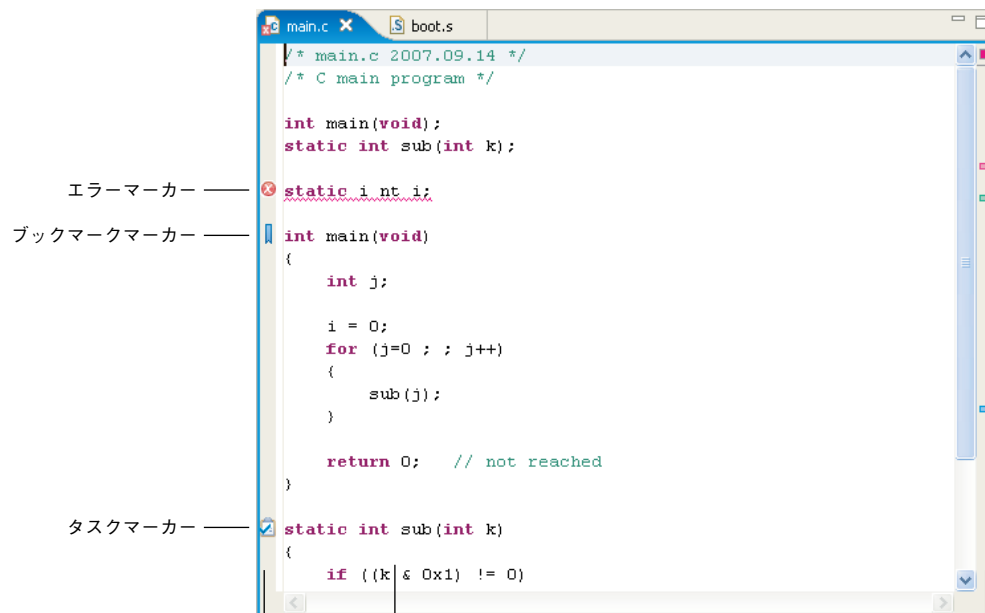
キー・アシスト... (Ctrl+Shift+L) 現在使用可能なメニューコマンドの一覧を表示します。

ソフトウェア更新... アップデータのインストールやプラグインの更新などを管理します。

Eclipse for GNU33 Vx.xについて IDEのバージョンやプラグインの詳細などを表示します。

エディタ領域

ソースファイルなどのテキストを編集するためのエディタ領域です。**IDE**ではファイルの種類に応じてC/C++エディタまたはアセンブラエディタが開きます。

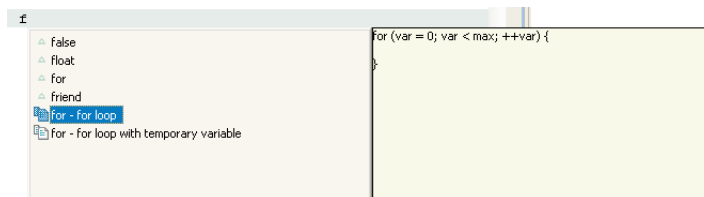


マーカーバー

エラー発生行、ブックマークやタスクが設定された行を示すマーカーが表示されます。マーカーは対応する行の先頭に表示されます。

編集領域

ソースなどテキストを編集します。
C/C++ソース編集時は、[Ctrl]+[Space]キーで下記のようなコンテンツアシストが表示されます。

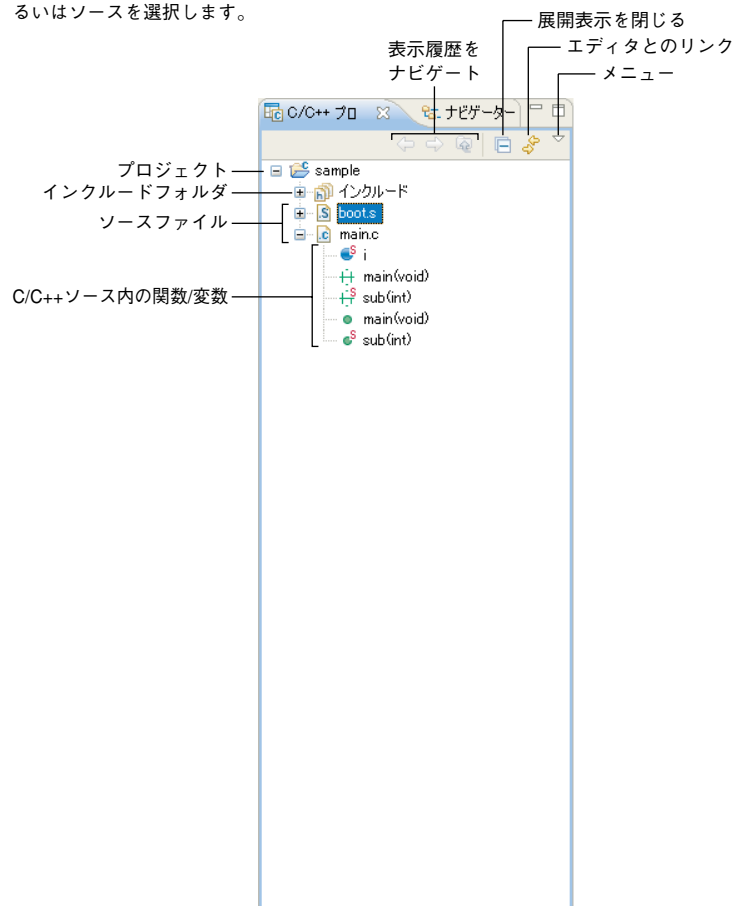


オーバービュールーラ

ファイル内のエラー発生位置、ブックマークやタスクが設定された位置を示すシンボルが表示されます。マーカーをクリックすると、その位置にジャンプします。

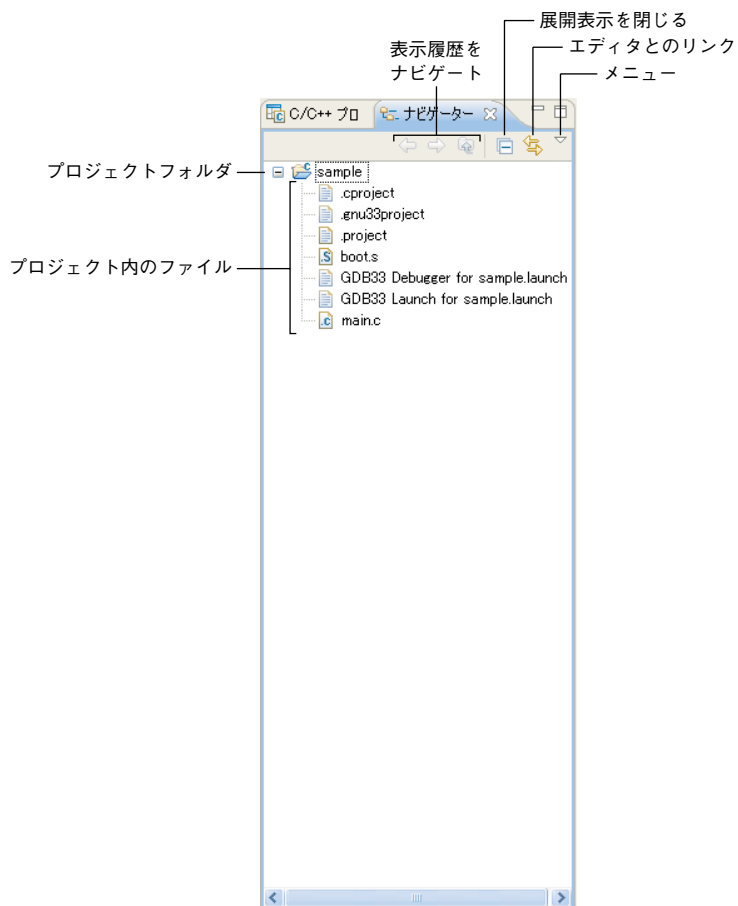
[C/C++ プロジェクト]ビュー

ワークスペース内のプロジェクトと、そこに含まれるC/C++およびアセンブラソース、インクルードファイル、生成された実行形式オブジェクトファイルの一覧を表示します(表示させるファイルの種類はビューメニューの[フィルター...]で選択可能です)。C/C++ソース内の関数名やグローバル変数名なども表示可能です。編集などの操作を行う場合に、ここで対象となるプロジェクトあるいはソースを選択します。



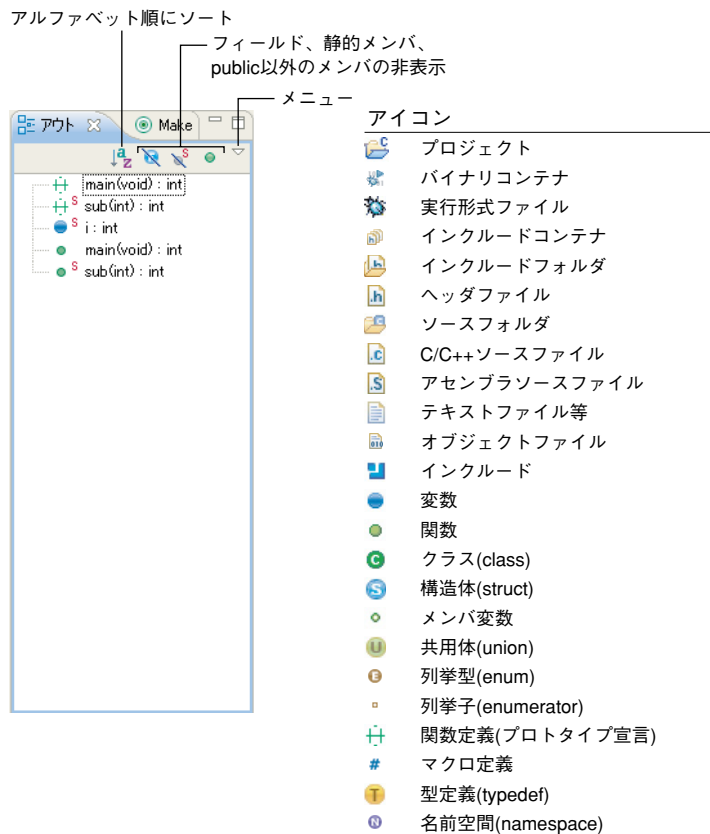
[ナビゲーター]ビュー

ワークスペース内のディレクトリおよびファイルの一覧を表示します(表示させるファイルの種類はビューメニューの[フィルター...]で選択可能です)。編集などの操作を行う場合に、ここで対象となるプロジェクトあるいはソースを選択します。



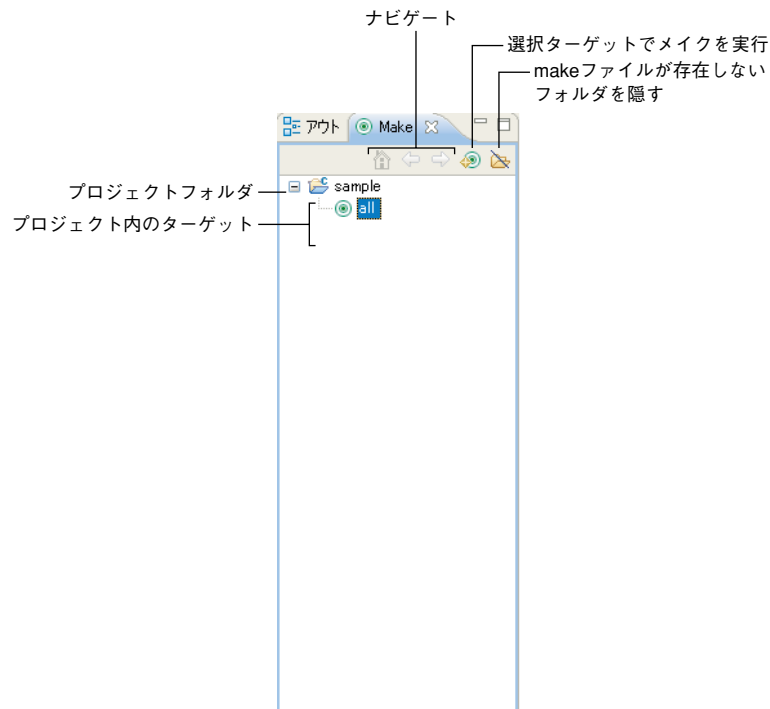
[アウトライン]ビュー

エディタで表示中のC/C++ソース内に記述されている関数やクラス、グローバル変数などを表示します。それらをクリックすることで、エディタ内の関数/変数の記述位置にジャンプします。アセンブラソースの表示中は何も表示されません。



[Make ターゲット]ビュー

ユーザが編集したmakeファイルを使用する場合に、ここにターゲットを定義して実行します。



[コンソール]ビュー

実行されたコマンドラインやGNU33ツールが出力するメッセージを表示します。

表示クリア
自動スクロールロック
[コンソール]出力継続中に
他のビューの選択を許可
[コンソール]の切り替え
新しい[コンソール]を開く

```

C-ビルド [sample]
C:\EPSON\GNU33\make.exe -f sample_gnu33IDE.mak all
C:\EPSON\GNU33\xgcc -BC:/EPSON/GNU33/ -c -xassembler-with-cpp -Wa,-gstabs -Wa,-medda32 -o boot.o
boot.s
C:\EPSON\GNU33\xgcc -BC:/EPSON/GNU33/ -gstabs -S -medda32 -O1 -IC:/EPSON/GNU33/include -fno-builtin
-mlong-calls -Wall -Werror-implicit-function-declaration -o main.ext0 main.c
C:\EPSON\GNU33\as -medda32 -o main.o main.ext0
C:\EPSON\GNU33\ld -Map sample.map -N -T sample_gnu33IDE.lids -o sample.elf boot.o main.o
C:\EPSON\GNU33/lib/std/libstdio.a C:/EPSON/GNU33/lib/std/libstdc++.a C:/EPSON/GNU33/lib/std/libgcc2.a
C:/EPSON/GNU33/lib/std/libc.a C:/EPSON/GNU33/lib/std/libgcc.a 2>lderr
C:\EPSON\GNU33\objdump -t sample.elf > sample.dump
  
```

[問題]ビュー

ビルド時に発生したエラーを表示します。ソースファイル内のエラーはエラーメッセージをクリックすることで、エディタ上のエラー発生箇所にジャンプします。

メニュー

エラー

ワーニング

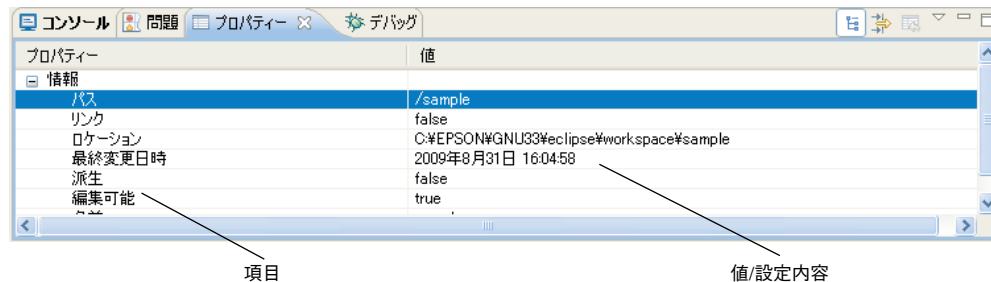
記述	リソース	パス	ロケ...	タイプ
エラー (2 項目)				
make: *** [main.o] Error 1	sample		0	C/C++ 問題
parse error before "nt"	main.c	sample	7	C/C++ 問題
警告 (2 項目)				
data definition has no type or storage	main.c	sample	7	C/C++ 問題
type defaults to 'int' in declaration of	main.c	sample	7	C/C++ 問題

エラーメッセージ

エラー発生箇所(ファイル/フォルダ/行番号)

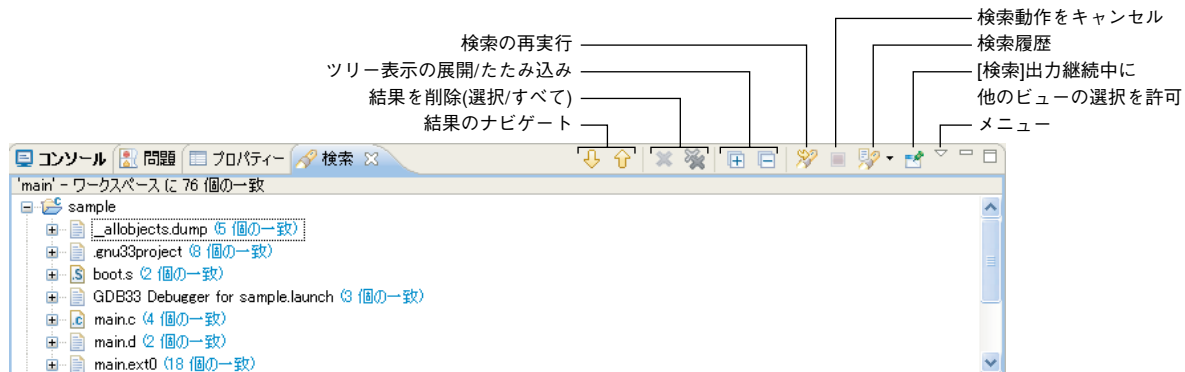
[プロパティ]ビュー

[C/C++ プロジェクト]ビュー、[ナビゲーター]ビューまたは[アウトライン]ビューで選択されたりソースやメンバなどの情報を表示します。



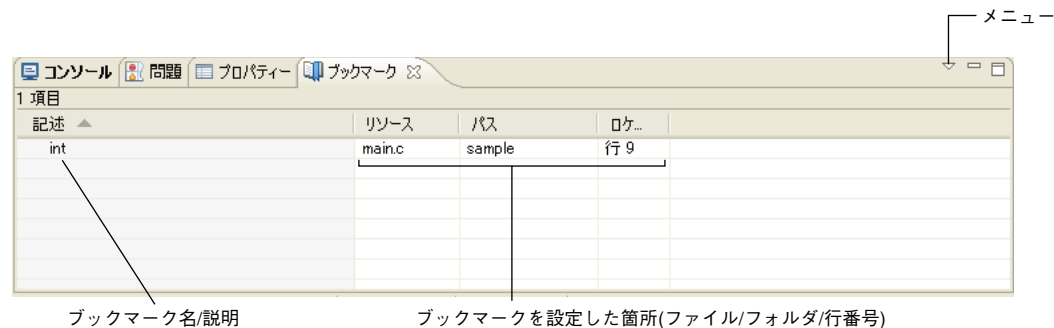
[検索]ビュー

[検索]ダイアログを使用した検索結果を表示します。このビューは初期状態では表示されません。検索を実行すると開きます。



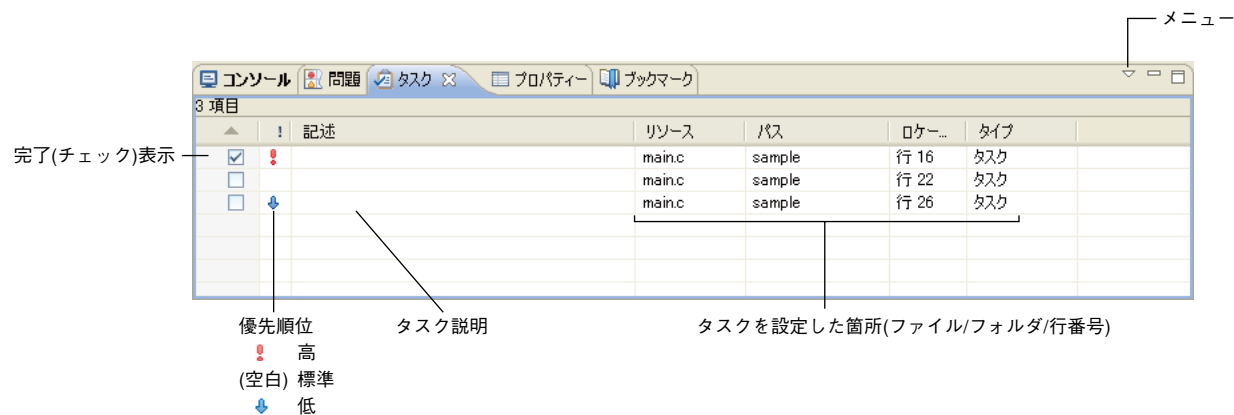
[ブックマーク]ビュー

エディタで登録したブックマークの表示、ブックマークへのジャンプ、ブックマークの削除などを行います。



[タスク]ビュー

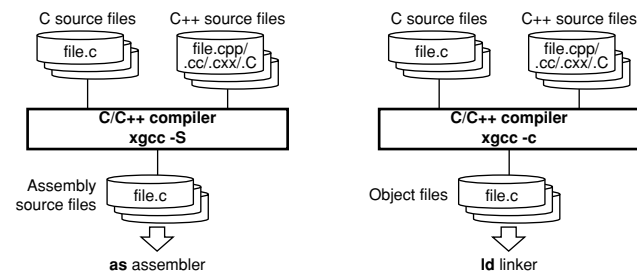
エディタで登録したタスク(To-Do)の表示、タスクへのジャンプ、タスクの削除などを行います。



概要

ANSI C/C++に準拠したC/C++コンパイラで、GNU C/C++コンパイラがベースとなっています。このツールは**cpp.exe**および**cc1.exe/cc1plus.exe**を連続して呼び出し、C/C++ソースファイルをコンパイルしてS1C33 Family用のアセンブリソースファイルを生成します。強力な最適化能力を持ち、非常にコンパクトなコードを生成します。**xgcc.exe**は、さらにアセンブラ**as.exe**を呼び出してオブジェクトファイルを生成することもできます。

フローチャート



起動コマンド

xgcc <options> <filename>

<filename> C/C++ソースファイル名

例: **xgcc -c -gstabs test.c** (Cソース)

xgcc -c -gstabs test.cc (C++ソース)

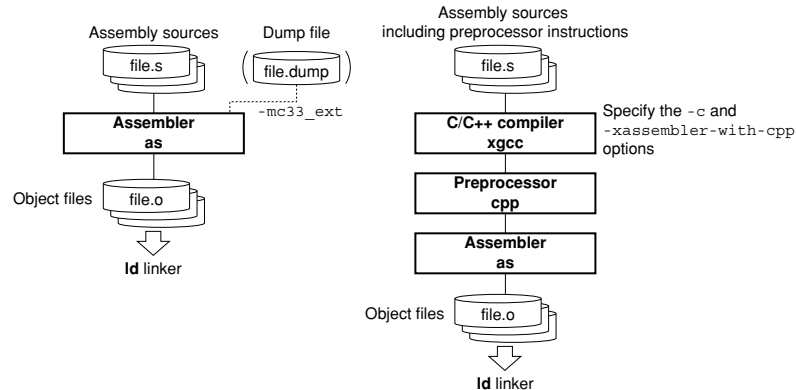
主要なコマンドラインオプション

-S	アセンブリコード(.s)の出力
-c	リロケータブルオブジェクトファイル(.o)の出力
-E	Cプリプロセッサのみの実行
-B<path>	コンパイラの検索パス指定
-I<path>	インクルードファイルディレクトリを指定
-fPIC	位置独立コードの生成
-fno-builtin	ビルトイン関数を無効化
-D<macro>[=<string>]	マクロ名の定義
-O0, -O, -O2, -O3, -Os	最適化
-gstabs	ソースファイルの相対パスを含むデバッグ情報の付加
-mc33adv	C33 ADVコア用コードの生成
-mc33401	C33 ADVコアのアセンブラフィルタ
-mc33pe	C33 PEコア用コードの生成
-medda32	デフォルトデータエリアの未使用
-mlong-calls	関数コールの展開
-mno-memcpy	strcpy、memcpyのインライン展開
-Wall	ワーニングオプションの有効化
-Werror-implicit-function-declaration	未宣言の関数をエラー出力
-mno-sjis-filt	Shift JISコードのフィルタ機能を無効化
-xassembler-with-cpp	cプリプロセッサの呼び出し

概要

C/C++コンパイラが出力するアセンブリソースファイルをアセンブルし、ソースファイルのニーモニックをS1C33000のオブジェクト(機械語)コードに変換します。
as.exeは**xgcc.exe**から**cpp.exe**に続けて実行させることも可能です。これにより、アセンブリソースファイル内でプリプロセッサ擬似命令を使用することも許されます。結果はリンクおよびライブラリ化が可能なオブジェクトファイルとして出力されます。

フローチャート



起動コマンド

as <options> <filename>

<filename> アセンブリソースファイル名

例: as -mc33_ext test.dump allobjects.dump -o test.o test.s

主要なコマンドラインオプション

-o<filename>	出力ファイル名の指定
-a[<suboption>]	アセンブリリストファイルの出力 例: -adh1 (ハイレベルな出力とデバッグ擬似命令の削除)
--gstabs	ソースファイルの相対バスを含むデバッグ情報の付加
-mc33adv	S1C33401(C33 ADVコア)用コードの生成
-mc33401	S1C33401(C33 ADVコア)のアセンブラフィルタ
-mc33pe	C33 PE用コードの生成
-medda32	デフォルトデータエリアの未使用
-mc33_ext <dumpfile> <allobjfile>	拡張命令の最適化

主要なプリプロセッサ擬似命令

#include	ファイルの挿入
#define	文字列および数値の定義
#if - #else - #endif	条件アセンブル

(xgccの-c -xassembler-with-cppオプション指定時に使用可能)

主要なアセンブラ擬似命令

.text	.textセクションを宣言
.section .data	.dataセクションを宣言
.section .rodata	.rodataセクションを宣言
.section .bss	.bssセクションを宣言
.section .comm	.commセクションを宣言
.section .ctors	.ctorsセクションを宣言
.section .dtors	.dtorsセクションを宣言
.section .gcc_except_table	.gcc_except_tableセクションを宣言
.long <data>	4バイトデータを定義
.short <data>	2バイトデータを定義
.byte <data>	バイトデータを定義
.ascii <string>	ASCII文字列を定義
.space <length>	空白領域(0x0)を定義
.zero <length>	空白領域(0x0)を定義
.align <value>	指定境界アドレスにアライメント
.global <symbol>	シンボルのグローバル宣言
.set <symbol>, <address>	シンボルに絶対アドレスを定義

シンボルマスク

@rh/@RH	相対アドレスの上位10ビットを取得
@rm/@RM	相対アドレスの中間13ビットを取得
@rl/@RL	相対アドレスの下位8ビットを取得
@h/@H	絶対アドレスの上位13ビットを取得
@m/@M	絶対アドレスの中間13ビットを取得
@l/@L	絶対アドレスの下位6ビットを取得
@ah/@AH	相対アドレスの上位13ビットを取得
@al/@AL	相対アドレスの下位13ビットを取得

擬似オペランド

doff_hi()/DOFF_HI()	dpからのオフセットアドレス上位13ビットを取得
doff_lo()/DOFF_LO()	dpからのオフセットアドレス下位13ビットを取得
dpoff_h()/DPOFF_H()*	dpからのオフセットアドレス上位13ビットを取得
dpoff_m()/DPOFF_M()*	dpからのオフセットアドレス中間13ビットを取得
dpoff_l()/DPOFF_L()*	dpからのオフセットアドレス下位6ビットを取得

(* S1C33401(C33 ADVコア)用)

エラー/ワーニングメッセージ

エラーメッセージ

Unrecognized opcode: 'XXXXX'	XXXXXは未定義のオペコードです。
XXXXX: invalid operand	オペランド書式のエラーです。
XXXXX: invalid register name	使用できないレジスタを指定しています。
There are too many characters of one line in assembler source file.	アセンブラソースファイルの1行の文字数(改行文字を除く)が2,047文字を超えています。
Cannot allocate memory.	malloc()によるメモリ確保に失敗しました。
Cannot specify plurality source files.	コマンドラインに複数のソースファイル名が指定されています。
Cannot find the dump file.	-mc33_extが指定されていますが、ダンプファイル名が指定されていません。あるいは、指定されたダンプファイルが存在しません。
The format of the dump file is invalid.	-mc33_extで指定されているダンプファイルの内容が不正です。
There are too many characters of one line in dump file.	-mc33_extで指定されたダンプファイルの1行の文字数(改行文字を除く)が2,047文字を超えています。
Cannot find the all objects' dump file.	-mc33_extが指定されていますが、全オブジェクトダンプファイル名が指定されていません。あるいは、指定されたファイルが存在しません。
The format of the all objects' dump file is invalid.	-mc33_extで指定されている全オブジェクトダンプファイルの内容が不正です。

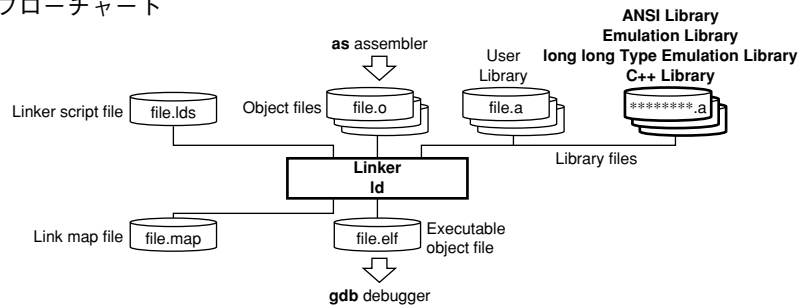
ワーニングメッセージ

operand out of range	オペランドで指定されている値は有効範囲外です。
(XXXXXX: XXX not between AAA and BBB)	有効範囲内(AAA~BBB)に補正します。
Unrecognized .section attribute: want a, w, x	セクションの属性がa、w、x以外です。
Bignum truncated to AAA bytes	定数宣言(.long、.intなど)のサイズが最大値を超えています。値をAAAバイトサイズに補正します。
Value XXXX truncated to AAA	定数宣言の値が最大値を超えています。値をAAAに補正します。

概要

C/C++コンパイラおよびアセンブラによって生成したオブジェクトコードのメモリロケーションを決定し、実行可能なオブジェクトコードを生成します。複数のオブジェクトファイルやライブラリも、このツールによって1つにまとめられます。

フローチャート



起動コマンド

ld <options> <filename>

<filename> リンクするオブジェクトファイルおよびライブラリファイル

例: ld -o sample.elf boot.o sample.o ../lib/libc.a ../lib/libgcc.a -defsym _dp=0

主要なコマンドラインオプション

-o <filename>	出力ファイル名の指定
-T <filename>	リンクスクリプトファイルの読み込み
-M	リンクマップのstdout出力
-Map <filename>	リンクマップのファイル出力
-N	データセグメントのアライメントチェックを禁止

エラー/ワーニングメッセージ

エラーメッセージ

Default Data area pointer value is larger than symbol address value.	__dp(デフォルトデータエリアポインタ)の値が定義されたシンボルのアドレス値を超えています。
The offset value of a symbol is over 64MB. (default data area)	__dp(デフォルトデータエリアポインタ)からシンボルまでのオフセットが64MBの範囲を超えています。
The offset value of a symbol is over 512KB. (default data area)	__dp(デフォルトデータエリアポインタ)からシンボルまでのオフセットが512KBの範囲を超えています。
The offset value of a symbol is over 64byte. (default data area)	__dp(デフォルトデータエリアポインタ)からシンボルまでのオフセットが64バイトの範囲を超えています。
Cannot link [STD PE ADV] object <objectfile> included from <archivefile> with [STD PE ADV] object <first objectfile>	<archivefile>に含まれている<objectfile>はターゲットCPUが異なるオブジェクトであるため、<first objectfile>とはリンクできません。
Input object file <objectfile> [included from <archivefile>] is not for C33.	<archivefile>に含まれている<objectfile>はC33のオブジェクトではありません。

ワーニングメッセージ

__dp symbol cannot be referred to.	1. doff_hi、dpoff_h、dpoff_m、dpoff_lまたはdoff_lo擬似オペランドが使用されていますが、__dpが指定されていません。 2. オペランドに[symbol+imm]が使用されていますが、__dpが指定されていません。 __dpは0x0として処理されます。

IDEが作成するデフォルトリンカスクリプトファイル

```

OUTPUT_FORMAT("elf32-c33", "elf32-c33", "elf32-c33")
OUTPUT_ARCH(c33)
SEARCH_DIR(.);

```

SECTIONS

```

{
/* data pointer symbols */
__dp = 0x00000000;

/* stack pointer symbols */
__START_stack = 0x00004000;

/* location counter */
. = 0x0;

/* section information */
.bss 0x00000000 :
{
    __START_bss = . ;
    boot.o(.bss)
    main.o(.bss)      files(.bss)
    libc.a(.bss)
}
__END_bss = . ;

```

```

.data __END_comm : AT( __END_gcc_except_table )
{
    __START_data = . ;
    files(.data)
}
__END_data = . ;

```

```

.vector 0x00C00000 :
{
    __START_vector = . ;
    boot.o(.rodata)
}
__END_vector = . ;

```

```

.text __END_vector :
{
    __START_text = . ;
    files(.text)
}
__END_text = . ;

```

```

.rodata __END_text :
{
    __START_rodata = . ;
    main.o(.rodata)
    libc.a(.rodata)
}
__END_rodata = . ;

```

```

.ctors __END_rodata :
{
    . = ALIGN(4);
    __START_ctors = . ;

```

```

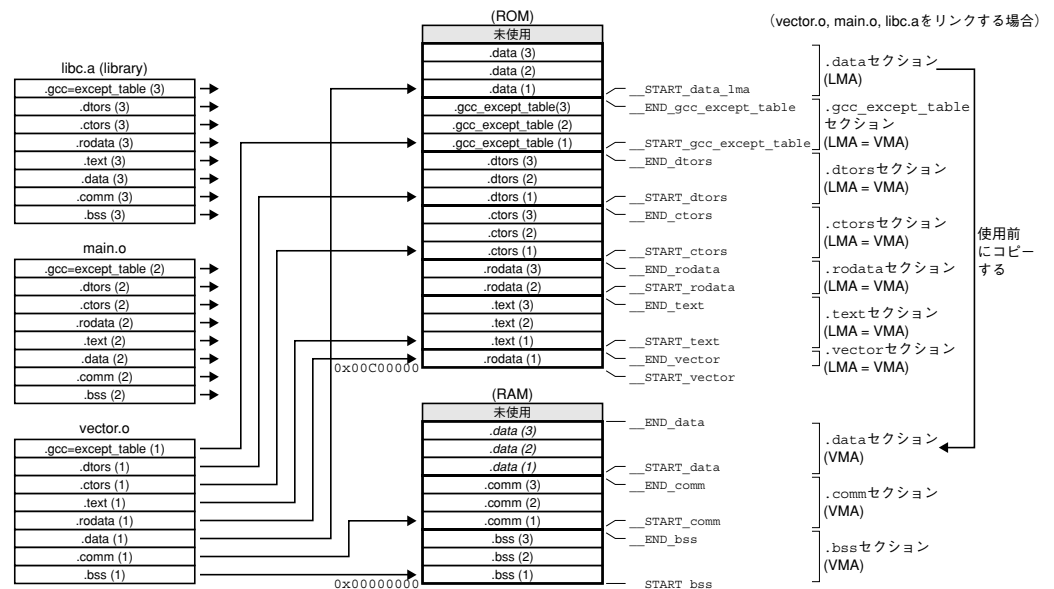
}
files(.ctors)
}
__END_ctors = . ;

.dtors __END_ctors :
{
    . = ALIGN(4);
    __START_dtors = . ;
    files(.dtors)
}
__END_dtors = . ;

.gcc_except_table __END_dtors :
{
    __START_gcc_except_table = . ;
    files(.gcc_except_table)
}
__END_gcc_except_table = . ;

/* load address symbols */
__START_data_lma = LOADADDR( .data );
__END_data_lma = __START_data_lma + SIZEOF( .data );
}

```



概要

ハードウェアツール(S5U1C33001H、S5U1C33000H)やデバッグモニタ(S5U1C330M2S)を制御してソースレベルのデバッグを行うソフトウェアです。特別なツールを使用せずにパソコン上でデバッグを行うシミュレーション機能も持っています。

gdb.exeはWindows GUIに対応しています。ブレークやステップ実行など、頻繁に使用するコマンドはツールバーに登録されており、キーボード操作の量を抑えています。また、各種のデータもマルチウィンドウとして表示できるため、デバッグ作業が効率良く行えます。

フローチャート

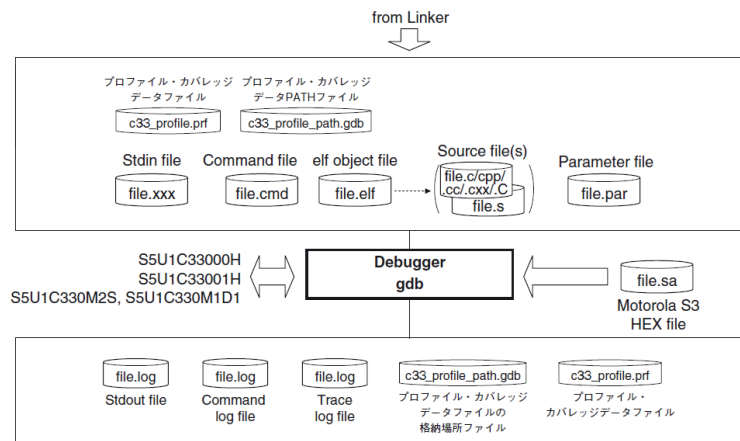


図10.2.1 フローチャート

起動コマンド

gdb <options>

例: `gdb -x sample.cmd --cd=/cygdrive/c/EPSON/gnu33/sample`

コマンドラインオプション

<code>--command=<filename></code>	コマンドファイルの指定
<code>-x <filename></code>	コマンドファイルの指定
<code>--c33_cmw=<seconds></code>	コマンドファイル内のコマンド実行間隔の指定
<code>--cd=<path></code>	カレントディレクトリの変更
<code>--directory=<path></code>	ソースファイルディレクトリの変更
<code>--double_starting</code>	二重起動の許可

デバッグパースペクティブ

[デバッグ]ビュー

デバッグ時のメインウィンドウで、デバッグに関わるメニューやツールバーを持ちます。ステップ実行やプログラムの停止・再開は本画面で行います。

[ソース]エディター

IDE上でソース編集に使用しているエディタは、そのままデバッグ時のソースの現在行表示に使用されます。また、ブレークポイントの設定も[ソース]エディタで行います。

[コンソール]ビュー

コマンドの実行と実行結果の表示に使用します。また、Simulated I/Oの出力も表示します。

[変数]ビュー

ローカル変数の値をモニターするのに使用します。

[ブレークポイント]ビュー

ブレークポイントの表示と管理に使用します。

[レジスタ]ビュー

CPUレジスタの値の表示と修正に使用します。

[式]ビュー

任意の監視式（グローバルシンボルやレジスタ）を登録し、その値をモニターするのに使用します。

[逆アセンブル]ビュー

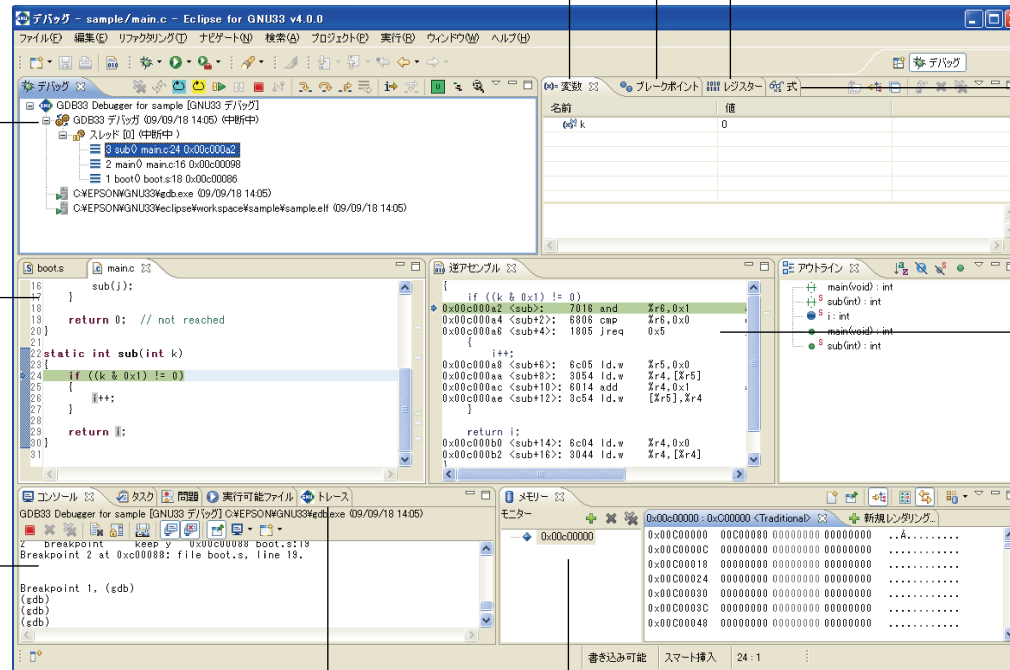
[デバッグ]ビューで選択されたスタックフレームに対応するプログラムの逆アセンブリを表示します。

[トレース]ビュー

トレースデータを表示します。

[メモリー]ビュー

メモリー内容の表示と修正に使用します。



コンテキストメニュー

[デバッグ]ビュー

	スタックのコピー (C)	Ctrl+C
	検索 (F)	Ctrl+F
	フレームにドロップ	
	再開	
	コールドリセット	
	ホットリセット	
	ステップイン (I)	F5
	ステップオーバー (O)	F6
	ステップリターン (R)	F7
	命令ステップ・モード	
	ステップフィルターの使用 (F)	
	シグナルなしで再開	
	再開 (I)	F8
	中絶 (I)	
	終了 (I)	Ctrl+F2
	終了して再起動	
	切り替 (I)	
	終了したすべてを除去 (A)	
	再起動 (I)	
	GDB33 Debugger for sample の編集...	
	ソース・ルックアップの編集 (I)	
	終了および除去 (I)	
	すべて終了/切断 (I)	
	プロパティ (I)	
	ユーザコマンド	
	プロファイラ	
	カバレッジ	

スタックのコピー 選択したアイコン以下の階層構造を文字列としてコピーします。

検索... アイコンを検索します。

フレームにドロップ サポートしません。

再開 サポートしません。

コールドリセット コールドリセットを実行します。

ホットリセット ホットリセットを実行します。

ステップイン ステップインします。

ステップ・オーバー ステップオーバーします。

ステップ・リターン ステップリターンします。

命令ステップ・モード 押し込んだ状態で、[ステップイン]/[ステップ・オーバー]がニーモニック1命令単位でステップ実行します。

ステップ・フィルターの使用 サポートしません。

シグナルなしで再開 サポートしません。

再開 実行を再開します。

中断 実行を中断します。

終了 デバッグ(GDB)を停止し、デバッグを終了します。

終了して再起動 [終了]実行した後、[再起動]します。

切断 サポートしません。

終了したすべてを除去 終了済みのアイコンをすべて削除します。

再起動 終了したデバッグを再起動します。

GDB33 Debugger for **** の編集... 起動構成ダイアログを開き、編集することができます。

ソース・ルックアップの編集... サポートしません。

ソースをルックアップ サポートしません。

終了および除去 [デバッグ]ビューで選択中のデバッグを終了し、アイコンを削除します。

すべて終了/切断 起動中のすべてのデバッグを終了します。

プロパティ サポートしません。

ユーザコマンド ユーザ定義コマンドを実行します。

プロファイラ プロファイラ画面を起動します。

カバレッジ カバレッジ画面を起動します。

[ソース]エディター

	指定行まで実行 (I)	Ctrl+R
	指定行で再開 (I)	
	監視式を追加...	
	実行 (I)	

[逆アセンブル]ビュー

	指定行まで実行 (I)	Ctrl+R
	指定行で再開 (I)	

コンテキストメニュー

[ブレイクポイント]ビュー

	ファイルヘジャンプ (I)	
	使用可能にする (I)	
	使用不可 (I)	
	除去 (I)	
	すべて除去 (I)	
	すべて選択 (A)	Ctrl+A
	コピー (I)	Ctrl+C
	貼り付け (I)	Ctrl+V
	ブレイクポイントのエクスポート (I)	
	ブレイクポイントのインポート (I)	
	監視ポイントの追加 (I/C++)	
	プロパティ (I)	

ファイルヘジャンプ 選択したブレイクポイントをエディタ上で開きます。

使用可能にする ブレイクポイントを有効化します。

使用不可 ブレイクポイントを無効化します。

除去 選択中のブレイクポイントを表示から削除します。

すべて除去 すべてのブレイクポイントを表示から削除します。

すべて選択 ブレイクポイント一覧を全選択します。

コピー/貼り付け ブレイクポイントのコピー/貼り付けを行います。

ブレイクポイントのエクスポート... ブレイクポイントを保存します。

ブレイクポイントのインポート... ブレイクポイントを復元します。

監視ポイントの追加... データブレイクを設定する[監視ポイントの追加]ダイアログを開きます。

プロパティ... ブレイクポイントのプロパティを表示するダイアログを開きます。

[変数]ビュー

	すべて選択 (A)	Ctrl+A
	変数のコピー (I)	Ctrl+C
	使用可能にする (I)	
	使用不可 (I)	
	×(I) 配列化して表示...	
	型にキャスト...	
	オリジナル型の復元	
	View Memory	
	検索 (F)	Ctrl+F
	値の変更 (I)	
	監視ポイントの追加 (I/C++)	
	グローバル変数の追加...	
	グローバル変数の除去...	
	全てのグローバル変数を除去...	
	監視式の作成 (I)	
	フォーマット	

すべて選択 ビューの表示を全選択します。

変数のコピー 選択した内容をコピーします。

使用可能にする 変数の更新が行われるようにします。

使用不可 変数の更新を行いません。

配列として表示... サポートしていません。

型にキャスト... 変数値を特定の型で表示します。

オリジナル型の復元 [型にキャスト...]の設定を元に戻します。

View Memory 変数の値のアドレスを[メモリー]ビューで表示します。

検索... 変数を検索します。

値の変更... 変数の値を変更するダイアログを開きます。

監視ポイントの追加... データブレイクを設定する[監視式を追加]ダイアログを開きます。

グローバル変数の追加... グローバル変数を一覧から選択して追加します。

グローバル変数の除去 選択中のグローバル変数を表示から削除します。

すべてのグローバル変数の除去 すべてのグローバル変数を表示から削除します。

監視式の作成 変数を[式]ビューに登録します。

フォーマット 表示形式を変更します。

コンテキストメニュー

[式]ビュー

すべて選択(A)	Ctrl+A
式のコピー(E)	Ctrl+C
除去(Q)	
すべて除去(U)	
検索(E)	Ctrl+F
値の変更(C)	
監視式を追加(A)...	
使用不可	
使用可能にする	
監視式を編集(E)	
監視式の作成(C)	
フォーマット	
配列として表示...	
型にキャスト...	
オリジナル型の復元	
View Memory	

すべて選択 ビューの表示を全選択します。
式のコピー 選択した内容をコピーします。
除去 選択中の式を表示から削除します。
すべて除去 すべての式を表示から削除します。
検索... 式を検索します。
値の変更... 式の値を変更するダイアログを開きます。
監視式を追加... 監視式を追加します。
監視式を編集... 監視式を編集します。
監視式を再評価 監視式を再評価(再計算)します。
監視式の作成 選択した値を監視式として[式]ビューに登録します。
使用可能にする 監視式の更新が行われるようにします。
使用不可 監視式の更新を行いません。
フォーマット 表示形式を変更します。
配列として表示... サポートしていません。
型にキャスト... 式の値を特定の型で表示します。
オリジナル型の復元 [型にキャスト...]の設定を元に戻します。
View Memory 式の値のアドレスを[メモリー]ビューで表示します。

[レジスター]ビュー

すべて選択(A)	Ctrl+A
レジスターのコピー(E)	Ctrl+C
使用可能にする	
使用不可	
配列として表示...	
型にキャスト...	
オリジナル型の復元	
View Memory	
検索(E)	
値の変更(C)	Ctrl+F
レジスター・グループの追加	
デフォルト・レジスター・グループの復元	
監視式の作成(C)	
フォーマット	

すべて選択 ビューの表示を全選択します。
レジスターのコピー 選択した内容をコピーします。
使用可能にする レジスタの更新が行われるようにします。
使用不可 レジスタの更新を行いません。
配列として表示... サポートしていません。
型にキャスト... レジスタの値を特定の型で表示します。
オリジナル型の復元 [型にキャスト...]の設定を元に戻します。
View Memory レジスタの値のアドレスを[メモリー]ビューで表示します。
検索... レジスタを検索します。
値の変更... レジスタの値を変更するダイアログを開きます。
レジスター・グループの追加 特定のレジスタのみを表示するレジスタグループを作成します。
デフォルト・レジスター・グループの復元 デフォルトのレジスタグループの表示に戻します。
レジスター・グループの編集 レジスタグループを編集します。
レジスター・グループの除去 レジスタグループを削除します。
監視式の作成 レジスタを[式]ビューに登録します。
フォーマット 表示形式を変更します。

コンテキストメニュー

[メモリー]ビュー

監視ポイントの追加 (C/C++)...	
レンダリングの追加	
レンダリングの除去	
Panes	
Endian	
Text	
Cell Size	
Radix	
Columns	
クリップボードにコピー	
Copy Address	
Reset To Base Address	

監視ポイントの追加... データブレイクを設定する[監視ポイントの追加]ダイアログを開きます。
レンダリングの追加 メモリを表示するための表示形式を追加します。
レンダリングの除去 選択中のメモリ表示を削除します。
Panes アドレス部/データ部/ASCII部の、表示/非表示を切り替えます。
Endian リトルエンディアンとビッグエンディアンでの表示を切り替えます。
Text ASCII部のエンコーディングを切り替えます。
Cell Size データ部の列ごとの表示バイト数を切り替えます。
Radix データ部の列ごとの表示フォーマットを切り替えます。
Columns データ部の列数を切り替えます。
クリップボードにコピー 選択した部分をコピーします。
Copy Address カーソル位置の境界アドレスをコピーします。
Reset To Base Address データ部の表示を、メモリモニタに登録した時点のアドレス位置へ戻します。

[コンソール]/[シミュレート]IOビュー

切り取り(Q)	Ctrl+X
コピー(C)	Ctrl+C
貼り付け(P)	Ctrl+V
すべて選択(A)	Ctrl+A
検索/置換(E)	
リンクを開く(C)	
クリア(R)	
終了したすべてを除去(A)	
スクロール・ロック	
設定(E)	

切り取り 選択した内容を切り取ります。
コピー 選択した内容をコピーします。
貼り付け コピーした内容を貼り付けます。
すべて選択 ビューの表示を全選択します。
検索/置換... コンソール内を検索します。
クリア コンソールの表示をクリアします。
終了したすべてを除去 終了済みのすべての[デバッグ]ビュー内のデバッグアイコンを削除します。
スクロール・ロック スクロールロックを切り替えます。
設定... コンソールの設定ダイアログを開きます。

[トレース]ビュー

コピー(C)	Ctrl+C
すべて選択(A)	Ctrl+A
トレースのクリア	

コピー 選択した内容をコピーします。
すべて選択 ビューの表示を全選択します。
トレースのクリア トレースの表示をクリアします。

デバグコマンド

メモリ操作

c33 fb <i>addr1 addr2 data</i>	領域のフィル (バイト単位)	ICD2/3/6/SIM/MON
c33 fh <i>addr1 addr2 data</i>	領域のフィル (ハーフワード単位)	ICD2/3/6/SIM/MON
c33 fw <i>addr1 addr2 data</i>	領域のフィル (ワード単位)	ICD2/3/6/SIM/MON
x <i>/[length]b [addr]</i>	メモリダンプ (バイト単位)	ICD2/3/6/SIM/MON
x <i>/[length]h [addr]</i>	メモリダンプ (ハーフワード単位)	ICD2/3/6/SIM/MON
x <i>/[length]w [addr]</i>	メモリダンプ (ワード単位)	ICD2/3/6/SIM/MON
set {char} <i>addr=data</i>	データ入力 (バイト単位)	ICD2/3/6/SIM/MON
set {short} <i>addr=data</i>	データ入力 (ハーフワード単位)	ICD2/3/6/SIM/MON
set {int} <i>addr=data</i>	データ入力 (ワード単位)	ICD2/3/6/SIM/MON
c33 mvb <i>addr1 addr2 addr3</i>	領域のコピー (バイト単位)	ICD2/3/6/SIM/MON
c33 mvh <i>addr1 addr2 addr3</i>	領域のコピー (ハーフワード単位)	ICD2/3/6/SIM/MON
c33 mvw <i>addr1 addr2 addr3</i>	領域のコピー (ワード単位)	ICD2/3/6/SIM/MON
c33 df <i>addr1 addr2 type file [append]</i>	メモリ内容の保存	ICD2/3/6/SIM/MON
c33 rm <i>addr1 addr2</i>	ターゲットメモリリード	ICD2/3/6
c33 readmd <i>mode</i>	メモリリードモード	ICD3/6

レジスタ操作

info reg <i>[register]</i>	レジスタ表示	ICD2/3/6/SIM/MON
set \$ <i>register=data</i>	レジスタ変更	ICD2/3/6/SIM/MON

プログラム実行

continue <i>[ignore]</i>	連続実行	ICD2/3/6/SIM/MON
until <i>addr</i>	テンポラリブレーク付き連続実行	ICD2/3/6/SIM/MON
step <i>[count]</i>	ステップ実行(行単位)	ICD2/3/6/SIM/MON
stepi <i>[count]</i>	ステップ実行(ニーモニック単位)	ICD2/3/6/SIM/MON
next <i>[count]</i>	スキップ付きステップ実行(行単位)	ICD2/3/6/SIM/MON
nexti <i>[count]</i>	スキップ付きステップ実行(ニーモニック単位)	ICD2/3/6/SIM/MON
finish	関数終了	ICD2/3/6/SIM/MON
c33 callmd <i>mode [file]</i>	ユーザ関数コールモードの設定	ICD2/3/6/SIM/MON
c33 call <i>func [arg1... [arg3]]</i>	ユーザ関数コール	ICD2/3/6/SIM/MON

CPUリセット

c33 rstc	コールドリセット(resetcold.gdbを実行)	ICD2/3/6/SIM/MON
c33 rsth	ホットリセット(resethot.gdbを実行)	ICD2/3/6/SIM/MON
c33 rstt	ターゲットリセット	ICD6

割り込み

c33 int <i>[type level]</i>	割り込み発生	SIM

ブレーク

break <i>[addr]</i>	ソフトウェアPCブレーク設定	ICD2/3/6/SIM/MON
tbreak <i>[addr]</i>	テンポラリソフトウェアPCブレーク設定	ICD2/3/6/SIM/MON
hbreak <i>[addr]</i>	ハードウェアPCブレーク設定	ICD2/3/6/SIM/MON
thbreak <i>[addr]</i>	テンポラリハードウェアPCブレーク設定	ICD2/3/6/SIM/MON
watch <i>addr</i>	データライトブレーク設定	ICD2/3/6/SIM/MON
rwatch <i>addr</i>	データリードブレーク設定	ICD2/3/6/SIM/MON
awatch <i>addr</i>	データリード/ライトブレーク設定	ICD2/3/6/SIM/MON
delete <i>[breakNo.]</i>	ブレーク番号によるブレークの解除	ICD2/3/6/SIM/MON
clear <i>[addr]</i>	ブレーク位置によるブレークの解除	ICD2/3/6/SIM/MON
enable <i>[breakNo.]</i>	ブレークポイントの有効化	ICD2/3/6/SIM/MON
disable <i>[breakNo.]</i>	ブレークポイントの無効化	ICD2/3/6/SIM/MON
ignore <i>breakNo. count</i>	回数指定付きブレークの無効化	ICD2/3/6/SIM/MON
info breakpoints	ブレークポイントリストの表示	ICD2/3/6/SIM/MON
c33 oab <i>parameters...</i>	オンチップエリア(CE)ブレーク設定 *1	ICD3/6
c33 obb <i>parameters...</i>	オンチップバスブレーク設定 *1	ICD3/6
c33 timebrk <i>timer</i>	時間経過後ブレークの設定	ICD6

シンボル情報

info locals	ローカルシンボル表示	ICD2/3/6/SIM/MON
info var	グローバルシンボル表示	ICD2/3/6/SIM/MON
print <i>symbol[=value]</i>	シンボル値の変更	ICD2/3/6/SIM/MON

ファイル読み込み

file <i>file</i>	デバグ情報の読み込み	ICD2/3/6/SIM/MON
load <i>[file]</i>	プログラムのロード	ICD2/3/6/SIM/MON

マップ情報

c33 rpf <i>file</i>	マップ情報の設定	ICD2/3/6/SIM/MON
c33 map	マップ情報の表示	ICD2/3/6/SIM/MON

フラッシュメモリ操作

c33 fls <i>addr1 addr2 erase write</i>	フラッシュメモリ設定	ICD2/3/6/MON
c33 fle <i>control block1 block2 [timer]</i>	フラッシュメモリの消去	ICD2/3/6/MON

トレース

c33 tm <i>md1 md2 [trg1 trg2 md3 md4]</i>	トレースモードの設定 (ICDモード)	ICD2/3/6
c33 tm on/off <i>[mode [file]]</i>	トレースモードの設定 (シミュレータモード)	SIM
c33 td <i>[cycle1 cycle2]</i>	トレース内容の表示	ICD2/3/6

デバッグコマンド

トレース

c33 autotd <i>parameters...</i>	PCトレース内容の自動表示	ICD2/3/6
c33 ts <i>addr [pre post]</i>	トレース内容の検索	ICD2/3/6
c33 tf <i>file [cycle1 cycle2]</i>	トレース内容の保存	ICD2/3/6
c33 autotf <i>parameters...</i>	PCトレース内容の自動保存	ICD2/3/6
c33 obt <i>parameters...</i>	オンチップバstreessモードの設定 *1	ICD3/6
c33 otd <i>mode [cycle1 cycle2]</i>	オンチップバstreess内容の表示 *1	ICD3/6
c33 otf <i>mode file [cycle1 cycle2]</i>	オンチップバstreess内容の保存 *1	ICD3/6

シミュレートDDI/O

c33 stdin <i>1/2 break buffer [file]</i>	データ入力シミュレーション	ICD2/3/6/SIM/MON
c33 stdout <i>1/2 break buffer [file]</i>	データ出力シミュレーション	ICD2/3/6/SIM/MON

フラッシュライタ

c33 fwe <i>0/1</i>	プログラム/データの消去	ICD3/6
c33 fwlp <i>file erase write [comment]</i>	プログラムのロード	ICD3/6
c33 fwld <i>file blk1 blk2 par [comment]</i>	データのロード	ICD3/6
c33 fwdc <i>addr sz blk1 blk2 par [comm]</i>	ターゲットメモリのコピー	ICD3/6
c33 fwd	フラッシュライタ情報の表示	ICD3/6

プロファイラ・カバレッジ

c33 profilemd	プロファイラ・カバレッジモード設定	sim
c33 profile	プロファイラウィンドウの表示	sim
c33 coverage	カバレッジウィンドウの表示	sim

その他

c33 log <i>[file]</i>	ロギング	ICD2/3/6/SIM/MON
source <i>file</i>	コマンドファイルの実行	ICD2/3/6/SIM/MON
c33 clockmd <i>mode func</i>	実行カウンタモード設定	ICD2/3/6/SIM
c33 clock	実行カウンタ表示	ICD2/3/6/SIM
target type <i>[time]</i>	ターゲットの接続	ICD2/3/6/SIM/MON
detach	ターゲットの切断	ICD2/3/6/SIM/MON
c33 das <i>ROMaddr RAMaddr [CPU]</i>	デバッグユニットアドレス設定	ICD3/6
c33 lpt <i>[port]</i>	パラレルロードモード設定/解除	ICD2
pwd	カレントディレクトリの表示	ICD2/3/6/SIM/MON
cd <i>directory</i>	カレントディレクトリの変更	ICD2/3/6/SIM/MON
c33 firmupdate <i>file</i>	ICDファームウェア更新	ICD3/6
c33 dclk <i>number</i>	DCLKの分周値変更	ICD2/3/6/MON
c33 oscwait <i>[counter]</i>	OSC1モード時のウェイト設定	ICD3/6
c33 cachehit	キャッシュヒット率の表示 *1	ICD3/6
c33 logiana <i>mode user</i>	ロジックアナライザモード設定	ICD3/6
c33 help <i>[Command/GroupNo.]</i>	ヘルプ	ICD2/3/6/SIM/MON
quit	デバッグ終了	ICD2/3/6/SIM/MON

*1 S1C33401(C33 ADVコア)のデバッグ時のみ使用可能

ステータス/エラーメッセージ

ステータスメッセージ

Breakpoint #, <i>function at file:line</i>	設定したブレークポイントでブレーク
Break by hard pc break1.	ハードウェアPCブレークポイント1でブレーク
Break by hard pc break2.	ハードウェアPCブレークポイント2でブレーク
Break by accessing no map.	シミュレータモードでマップされていない領域へのアクセスによりブレーク
Break by writing ROM area.	シミュレータモードでリードオンリ領域へのアクセスによりブレーク
Break by stack overflow.	シミュレータモードでスタック領域のオーバーによるブレーク
Break by sleep or halt.	シミュレータモードでsleepまたはhalt命令実行によるブレーク
Illegal address exception.	シミュレータモードで不当アドレスによる命令の実行によりブレーク
Illegal instruction.	シミュレータモードで不当命令実行によりブレーク
Illegal delayed instruction.	シミュレータモードで不当な遅延命令の実行によりブレーク
Hardware read watchpoint #: <i>symbol</i>	データリード条件でブレーク
Hardware access (read/write) watchpoint #: <i>symbol</i>	データアクセス(リード/ライト)条件でブレーク
Program received signal SIGTRAP, Trace/breakpoint trap.	[Stop]ボタンによる強制ブレーク (シミュレータモード)
Program received signal SIGINT, Interrupt.	[Stop]ボタンによる強制ブレーク (ICDモード)

エラーメッセージ

A setup of a serial port was not completed.	ICDとのシリアル通信速度が115200または38400bps以外です。
Address (0x#) is ext or delayed instruction.	指定したアドレスは- 先頭のext命令を除く拡張命令または、- 遅延命令 (遅延分岐命令の次) のため設定ができません。
C33 command error, command is not supported in present mode.	入力したコマンドは、現在の接続モードでは実行できません。
C33 command error, command is too long.	入力したコマンド長が256文字を超えています。
C33 command error, invalid command.	コマンドに誤りがあります。
C33 command error, number of parameter.	コマンドのパラメータ数が正しくありません。
C33 command error, start address > end address.	開始アドレスが終了アドレスを超えて指定されています。
C33 command error, start cycle < end cycle.	終了サイクルが開始サイクルを超えて指定されています。
C33 command error, cycle is 0 - 131071 (ICD33V2) or 0 - 1048575 (ICD33V3/V6).	指定したサイクル番号が指定可能範囲を超えています。
Cannot access memory at address #.	アドレス#をアクセスできません。
Cannot allocate memory.	パラメータで指定したサイズのメモリ領域が確保できません。
Cannot clear data break (0x#).	未設定のデータブレークアドレスを指定しました。
Cannot clear hard pc break (0x#).	未設定のハードウェアPCブレークアドレスを指定しました。
Cannot clear soft pc break (0x#).	未設定のソフトウェアPCブレークアドレスを指定しました。
Cannot open file (<i>file</i>).	ファイルがオープンできません。
Cannot open ICD33 usb driver.	USBドライブのオープンに失敗しました。
Cannot set data break any more.	データブレークポイントを設定できません。
Cannot set hard pc break any more.	ハードウェアPCブレークポイントの設定数が制限を超えました(Max.2)。
Cannot set soft pc break any more.	ソフトウェアPCブレークポイントの設定数が制限を超えました(Max.200)。
Cannot set hard pc break at ext or delayed instruction.	ハードPCブレークで指定したアドレスはext命令または遅延命令のため設定ができません。
Cannot set soft pc break at ext, delayed instruction or No RAM Area.	ソフトPCブレークで指定したアドレスは- 先頭のext命令を除く拡張命令の行 - 遅延命令の行 (遅延分岐命令の次の行) または、RAM領域でないため設定ができません。

ステータス/エラーメッセージ

エラーメッセージ

Cannot set some breakpoint address.	指定したアドレスには、既にブレークが設定されています。
Cannot set soft pc break at ROM area.	ソフトPCブレークで指定したアドレスはROM領域であるため設定ができません。
Cannot write file.	ファイルへの書き込みができません。
Clock timer overflow.	クロック計測でタイムオーバーになりました。
Communication error (bcc).	ICDからの受信メッセージにBCCエラーが発生しました。
Communication error (host->ICD33).	ICDへのUSB送信に失敗しました。
Communication error (ICD33-> host).	ICDへのUSB受信に失敗しました。
Communication internal error (#).	ICDとの通信中に内部エラーが発生しました。
Communication system error (#).	ICDとの通信中に切断してしまいました。
Coverage is not supported in present mode.	カバレッジ・ウィンドウは、現在のコネクトモードではサポートしていません。シュミレータモードでご使用ください。
Coverage Window is already opened.	カバレッジ・ウィンドウが既に開いています。
"Editor path" exceeds 255 characters.	"Editor path"が255文字を超えています。
Expression cannot be implemented with read/access watchpoint.	指定したアドレスへのデータブレークは設定できません。
Framing error.	ICDとの通信でフレーミングエラーが発生しました。
gnuEdit.gdb file save error	外部エディタバスの保存に失敗しました。
ICD33 is busy (#).	ICD側がBUSY状態です。
Initialization error of ICD33.	ターゲットの初期化に失敗しました。
Invalid parameter file (#: file).	ターゲットの初期化に失敗しました。
Invalid parameter file, start address > end address (#: file).	パラメータファイルのアドレス範囲設定で、開始アドレスが終了アドレスを超えています。
It has not connected with a target.	ICDとターゲットが接続できません。
It is not c33 architecture ELF file.	fileコマンドで指定したファイルはS5U1C33001Cが対応しているelf形式ファイルではありません。
Load max address (0x#) overflow.	ロードするアドレスが最大値を超えています。
Load motorola file format error.(file)	ロードするモトローラファイルのフォーマットに誤りがあります。
No memory map information.	マップ情報がありません。
Please input "Editor path".	"Editor path"に文字を入力してください。
Profiler is not supported in present mode.	プロファイラ・ウィンドウは、現在のコネクトモードではサポートしていません。シュミレータモードでご使用ください。

エラーメッセージ

Profiler Window is already opened.	プロファイラ・ウィンドウが既に開いています。
Receiving message is inaccurate.	ICDとの通信で最大サイズを超えるメッセージを受信しました。
Specification is required in the device for connecting.	targetコマンドでICD選択のデバイス名が足りません。
Target connection causes communication error at # times.	SIO接続チェックで異常が発生しました。
Target connection causes data error at # times.	SIO接続チェックが実行できません。
Target connection causes time out error at # times.	SIO接続チェックがタイムアウトです。
Target down.	ICD/ターゲット間で通信エラーが発生しました。
The simulator cannot be connected twice. please quit gdb.	シュミレータモードの接続が2度行えません。GDBを再起動してください。
There is no argument given to this command.	ターゲットの切断に失敗しました。
Timeout error # (ICD33 -> host).	ICDとの通信でICDからの受信待ちタイムアウトが発生しました。
Transmitting failure (#).	ICDとの通信でICDからNAKを受け取りました。
Warning: target file is not standard macro program.	fileコマンドで指定したELFファイルがスタンダードマクロ用ではありません。
Warning: target file is not PE program.	fileコマンドで指定したELFファイルがPE用ではありません。
Warning: target file is not advanced macro program.	fileコマンドで指定したELFファイルがアドバンスドマクロ用ではありません。

浮動小数点エミュレーションライブラリ libgcc.a

Double型演算

__adddf3	加算	(%r5, %r4) ← (%r7, %r6) + (%r9, %r8)
__subdf3	減算	(%r5, %r4) ← (%r7, %r6) - (%r9, %r8)
__muldf3	乗算	(%r5, %r4) ← (%r7, %r6) * (%r9, %r8)
__divdf3	除算	(%r5, %r4) ← (%r7, %r6) / (%r9, %r8)
__negdf2	符号反転	(%r5, %r4) ← -(%r7, %r6)

Float型演算

__addsf3	加算	%r4 ← %r6 + %r7
__subsf3	減算	%r4 ← %r6 - %r7
__mulsf3	乗算	%r4 ← %r6 * %r7
__divsf3	除算	%r4 ← %r6 / %r7
__negsf2	符号反転	%r4 ← -%r6

型変換

__fixunsdfsi	double → unsigned int	%r4 ← (%r7, %r6)
__fixdfsi	double → int	%r4 ← (%r7, %r6)
__floatsidf	int → double	(%r5, %r4) ← %r6
__fixunssfsi	float → unsigned int	%r4 ← %r6
__fixfsi	float → int	%r4 ← %r6
__floatsisf	int → float	%r4 ← %r6
__truncdfsf2	double → float	%r4 ← (%r7, %r6)
__extendsfdf2	float → double	(%r5, %r4) ← %r6

比較

__fcmpd	double type	Changes %psr by (%r7, %r6) - (%r9, %r8)
__**df2	double type	Changes %psr and %r4 by (%r7, %r6) - (%r9, %r8) **=eq, ne, gt, ge, lt, le (%r4 = 1 if true, 0 if false)
__fcmps	float type	Changes %psr by %r6 - %r13
__**sf2	float type	Changes %psr and %r4 by %r6 - %r13 **=eq, ne, gt, ge, lt, le (%r4 = 1 if true, 0 if false)

整数除算ライブラリ libgcc.a

整数除算

__divsi3	符号付き除算	%r4 ← %r6 / %r7
__udivsi3	符号なし除算	%r4 ← %r6 / %r7

剰余演算

__modsi3	符号付き演算	%r4 ← %r6 % %r7
__umodsi3	符号なし演算	%r4 ← %r6 % %r7

浮動小数点データフォーマット

Double型データ

63 62	52 51	0
S	指数部	仮数部

Double型有効範囲

+0:	0.0e+0 0x00000000 00000000
-0:	-0.0e+0 0x80000000 00000000
最大正規化数:	1.79769e+308 0x7fffffff ffffffff
最小正規化数:	2.22507e-308 0x00100000 00000000
最大非正規化数:	2.22507e-308 0x000fffff ffffffff
最小非正規化数:	4.94065e-324 0x00000000 00000001
無限大:	0x7ff00000 00000000
-無限大:	0xffff0000 00000000

Float型データ

31 30	23 22	0
S	指数部	仮数部

Float型有効範囲

+0:	0.0e+0f 0x00000000
-0:	-0.0e+0f 0x80000000
最大正規化数:	3.40282e+38f 0x7fffffff
最小正規化数:	1.17549e-38f 0x00800000
最大非正規化数:	1.17549e-38f 0x007fffff
最小非正規化数:	1.40129e-45f 0x00000001
無限大:	0x7ff80000
-無限大:	0xff800000

long long型エミュレーションライブラリ libgcc2.a

演算

<u>_muldi3</u>	乗算	(%r5, %r4) ← (%r7, %r6) * (%r9, %r8)
<u>_divdi3</u>	符号付き除算	(%r5, %r4) ← (%r7, %r6) / (%r9, %r8)
<u>_udivdi3</u>	符号なし除算	(%r5, %r4) ← (%r7, %r6) / (%r9, %r8)
<u>_moddi3</u>	符号付き剰余演算	(%r5, %r4) ← (%r7, %r6) % (%r9, %r8)
<u>_umoddi3</u>	符号なし剰余演算	(%r5, %r4) ← (%r7, %r6) % (%r9, %r8)
<u>_negdi2</u>	符号反転	(%r5, %r4) ← (%r7, %r6)
<u>_lshrdi3</u>	論理右シフト	(%r5, %r4) ← (%r7, %r6) >> (%r9, %r8)
<u>_ashrdi3</u>	算術右シフト	(%r5, %r4) ← (%r7, %r6) >> (%r9, %r8)
<u>_ashldi3</u>	算術左シフト	(%r5, %r4) ← (%r7, %r6) << (%r9, %r8)

型変換

<u>_fixunsdfdi</u>	double → unsigned long long	(%r5, %r4) ← (%r7, %r6)
<u>_fixdfdi</u>	double → long long	(%r5, %r4) ← (%r7, %r6)
<u>_floatdidf</u>	long long → double	(%r5, %r4) ← (%r7, %r6)
<u>_fixunssfdi</u>	float → unsigned long long	(%r5, %r4) ← %r6
<u>_fixsfdi</u>	float → long long	(%r5, %r4) ← %r6
<u>_floatdisf</u>	long long → float	%r4 ← (%r7, %r6)

比較

<u>_cmpdi2</u>	long long型比較	%psr変更 ← (%r7, %r6) - (%r9, %r8), %r4 ← 2 1 0
<u>_ucmpdi2</u>	unsigned long long型比較	%psr変更 ← (%r7, %r6) - (%r9, %r8), %r4 ← 2 1 0

その他

<u>_ffsdi2</u>	ビットスキャン	(%r5, %r4) ← (%r7, %r6)

入出力関数(ヘッダファイル: stdio.h/cstdio)

fopen()	FILE *fopen(const char *filename, const char *mode);	(dummy)*1
freopen()	FILE *freopen(const char *filename, const char *mode, FILE *stream);	(dummy)*1
fclose()	int fclose(FILE *stream);	(dummy)
fflush()	int fflush(FILE *stream);	(dummy)
fseek()	int fseek(FILE *stream, long offset, int origin);	(dummy)*1
ftell()	long ftell(FILE *stream);	(dummy)
rewind()	void rewind(FILE *stream);	(dummy)
fgetpos()	int fgetpos(FILE *stream, fpos_t *ptr);	(dummy)
fsetpos()	int fsetpos(FILE *stream, const fpos_t *pos);	(dummy)*1
fread()	size_t fread(void *ptr, size_t size, size_t count, FILE *stream);	*1, *2
fwrite()	size_t fwrite(const void *ptr, size_t size, size_t count, FILE *stream);	*1, *2
fgetc()	int fgetc(FILE *stream);	*2
getc()	int getc(FILE *stream);	*1, *2
getchar()	int getchar(void);	*1, *2
ungetc()	int ungetc(int c, FILE *stream);	*1
fgets()	char *fgets(char *s, int n, FILE *stream);	*1, *2
gets()	char *gets(char *s);	*1, *2
fputc()	int fputc(int c, FILE *stream);	*2
putc()	int putc(int c, FILE *stream);	*1, *2
putchar()	int putchar(int c);	*1, *2
fputs()	int fputs(char *s, FILE *stream);	*1, *2
puts()	int puts(char *s);	*1, *2
remove()	int remove(const char *filename);	(dummy)*1
rename()	int rename(const char *oldname, const char *newname);	(dummy)*1
setbuf()	void setbuf(FILE *stream, char *buf);	(dummy)
setvbuf()	int setvbuf(FILE *stream, char *buf, int type, size_t size);	(dummy)
tmpfile()	FILE *tmpfile(void);	(dummy)*1
tmpnam()	char *tmpnam(char *buf);	(dummy)*1
feof()	int feof(FILE *stream);	(dummy)
ferror()	int ferror(FILE *stream);	(dummy)
clearerr()	void clearerr(FILE *stream);	(dummy)
perror()	void perror(const char *s);	*1, *2
fscanf()	int fscanf(FILE *stream, const char *format, ...);	*1, *2
scanf()	int scanf(const char *format, ...);	*1, *2
sscanf()	int sscanf(const char *s, const char *format, ...);	*1, *2
fprintf()	int fprintf(FILE *stream, const char *format, ...);	*1, *2
printf()	int printf(const char *format, ...);	*1, *2
sprintf()	int sprintf(char *s, const char *format, ...);	*1, *2
vfprintf()	int vfprintf(FILE *stream, const char *format, va_list arg);	*1, *2
vprintf()	int vprintf(const char *format, va_list arg);	*1, *2
vsprintf()	int vsprintf(char *s, const char *format, va_list arg);	

ユーティリティ関数(ヘッダファイル: stdlib.h/cstdlib)

malloc()	void *malloc(size_t size);	*1
calloc()	void *calloc(size_t elt_count, size_t elt_size);	*1
free()	void free(void *ptr);	*1
realloc()	void *realloc(void *ptr, size_t size);	*1
system()	int system(const char *command);	
exit()	void exit(int status);	
abort()	void abort(void);	
atexit()	int atexit(void (*func)(void));	
getenv()	char *getenv(const char *str);	
bsearch()	void *bsearch(const void *key, const void *base, size_t count, size_t size, int (*compare)(const void *, const void *));	
qsort()	void qsort(void *base, size_t count, size_t size, int (*compare)(const void *, const void *));	
abs()	int abs(int x);	
labs()	long labs(long x);	
div()	div_t div(int n, int d);	*1
ldiv()	ldiv_t ldiv(long n, long d);	*1
rand()	int rand(void);	
srand()	void srand(unsigned int seed);	
atol()	long atol(const char *str);	
atoi()	int atoi(const char *str);	*1
atof()	double atof(const char *str);	*1
strtod()	double strtod(const char *str, char **ptr);	*1
strtol()	long strtol(const char *str, char **ptr, int base);	*1
strtoul()	unsigned long strtoul(const char *str, char **ptr, int base);	*1
strtoll()	long long strtoll(const char *str, char **endptr, int base);	*3
strtoull()	unsigned long long strtoull(const char *str, char **endptr, int base);	*3

日付と時刻関数(ヘッダファイル: time.h/ctime)

clock()	clock_t clock(void);	(dummy)
asctime()	char *asctime(const struct tm *ts);	(dummy)
ctime()	char *ctime(const time_t *timeptr);	(dummy)
difftime()	double difftime(time_t ti, time_t t2);	(dummy)
gmtime()	struct tm *gmtime(const time_t *t);	
localtime()	struct tm *localtime(const time_t *t);	(dummy)
mktime()	time_t mktime(struct tm *tmptr);	
time()	time_t time(time_t *tptr);	*1

非局所分岐関数(ヘッダファイル: setjmp.h/csetjmp)

setjmp()	int setjmp(jmp_buf env);	
longjmp()	void longjmp(jmp_buf env, int status);	

*1 グローバル変数の宣言と初期化が必要です。

*2 下位レベル関数と入出力バッファの定義が必要です。

*3 libgcc2.aに含まれるユーティリティ関数です。

数学関数(ヘッダファイル: math.h, errno.h, float.h, limits.h/cmath, cerrno, cfloat, climits)

fabs()	double fabs(double x);	*1
ceil()	double ceil(double x);	*1
floor()	double floor(double x);	*1
fmod()	double fmod(double x, double y);	*1
exp()	double exp(double x);	*1
log()	double log(double x);	*1
log10()	double log10(double x);	*1
frexp()	double frexp(double x, int *nptr);	*1
ldexp()	double ldexp(double x, int n);	*1
modf()	double modf(double x, double *nptr);	*1
pow()	double pow(double x, double y);	*1
sqrt()	double sqrt(double x);	*1
sin()	double sin(double x);	*1
cos()	double cos(double x);	*1
tan()	double tan(double x);	*1
asin()	double asin(double x);	*1
acos()	double acos(double x);	*1
atan()	double atan(double x);	
atan2()	double atan2(double y, double x);	*1
sinh()	double sinh(double x);	*1
cosh()	double cosh(double x);	*1
tanh()	double tanh(double x);	

文字種判定/変換関数(ヘッダファイル: ctype.h/cctype)

isalnum()	int isalnum(int c);
isalpha()	int isalpha(int c);
iscntrl()	int iscntrl(int c);
isdigit()	int isdigit(int c);
isgraph()	int isgraph(int c);
islower()	int islower(int c);
isprint()	int isprint(int c);
ispunct()	int ispunct(int c);
isspace()	int isspace(int c);
isupper()	int isupper(int c);
isxdigit()	int isxdigit(int c);
tolower()	int tolower(int c);
toupper()	int toupper(int c);

可変引数マクロ(ヘッダファイル: stdarg.h/cstdarg)

va_start()	void va_start(va_list ap, type lastarg);
va_arg()	type va_arg(va_list ap, type);
va_end()	void va_end(va_list ap);

*1 グローバル変数の宣言と初期化が必要です。

文字関数(ヘッダファイル: string.h/cstring)

memchr()	void *memchr(const void *s, int c, size_t n);
memcmp()	int memcmp(const void *s1, const void *s2, size_t n);
memcpy()	void *memcpy(void *s1, const void *s2, size_t n);
memmove()	void *memmove(void *s1, const void *s2, size_t n);
memset()	void *memset(void *s, int c, size_t n);
strcat()	char *strcat(char *s1, const char *s2);
strchr()	char *strchr(const char *s, int c);
strcmp()	int strcmp(const char *s1, const char *s2);
strcpy()	char *strcpy(char *s1, const char *s2);
strcspn()	size_t strcspn(const char *s1, const char *s2);
strerror()	char *strerror(int code);
strlen()	size_t strlen(const char *s);
strncat()	char *strncat(char *s1, const char *s2, size_t n);
strncmp()	int strncmp(const char *s1, const char *s2, size_t n);
strncpy()	char *strncpy(char *s1, const char *s2, size_t n);
strpbrk()	char *strpbrk(const char *s1, const char *s2);
strrchr()	char *strrchr(const char *s, int c);
strspn()	size_t strspn(const char *s1, const char *s2);
strstr()	char *strstr(const char *s1, const char *s2);
strtok()	char *strtok(char *s1, const char *s2);

診断マクロ(ヘッダファイル: assert.h/cassert)

assert()	void assert(int test);

ロケール関数(ヘッダファイル: locale.h/clocale)

setlocale()	char *setlocale(int category, const char *locale);	(dummy)
localeconv()	struct lconv *localeconv(void);	(dummy)

***1 グローバル変数の宣言と初期化**

FILE _iob[OPEN_MAX+1];	_iob[N]_flg= _UGETN; _iob[N]_buf=0; _iob[N]_fd=N; (N=0: stdin, N=1: stdout, N=2: stderr)
FILE *stdin;	stdin=&_iob[0];
FILE *stdout;	stdout=&_iob[1];
FILE *stderr;	stderr=&_iob[2];
int errno;	errno=0;
unsigned int seed;	seed=1;
time_t gm_sec;	gm_sec=-1;

***2 下位レベル関数の定義**

read()	int read(int fd, char *buf, int nbytes); unsigned char READ_BUF[65]; (他の名称も可) unsigned char READ_EOF;
write()	int write(int fd, char *buf, int nbytes); unsigned char WRITE_BUF[65]; (他の名称も可)

STLコンテナメンバ関数

メンバ関数	コンテナ名									
	deque	list	map	multimap	multiset	priority_queue	queue	set	stack	vector
assign	○	○								○
at	○									○
back	○	○					○			○
begin	○	○	○	○	○			○		○
capacity										○
clear	○	○	○	○	○			○		○
count			○	○	○			○		
empty	○	○	○	○	○	○	○	○	○	○
end	○	○	○	○	○			○		○
equal_range			○	○	○			○		
erase	○	○	○	○	○			○		○
find			○	○	○			○		
front	○	○					○			○
get_allocator	○	○	○	○	○			○		○
insert	○	○	○	○	○			○		○
key_comp			○	○	○			○		
lower_bound			○	○	○			○		
max_size	○	○	○	○	○			○		○
merge		○								
operator[]	○		○							○
pop						○	○		○	
pop_back	○	○								○
pop_front	○	○								
push						○	○		○	
push_back	○	○								○
push_front	○	○								
rbegin	○	○	○	○	○			○		○
remove		○								
remove_if		○								
rend	○	○	○	○	○			○		○
reserve										○
resize	○	○								○
reverse		○								
size	○	○	○	○	○	○	○	○	○	○
sort		○								
splice		○								
swap	○	○	○	○	○			○		○
top						○			○	
unique		○								
upper_bound			○	○	○			○		
value_comp			○	○	○			○		

命令一覧中のシンボル

レジスタ/レジスタデータ

%rd, rd: ディスティネーションとなる汎用レジスタ(R0~R15)、またはレジスタの内容
 %rs, rs: ソースとなる汎用レジスタ(R0~R15)、またはレジスタの内容
 %rb, rb: レジスタ間接アドレッシングのベースレジスタを保持している汎用レジスタ(R0~R15)、またはベースアドレス
 %sd, sd: ディスティネーションとなる特殊レジスタ(PSR, SP, ALR, AHR)、またはレジスタの内容
 %ss, ss: ソースとなる特殊レジスタ(PSR, SP, ALR, AHR)、またはレジスタの内容
 *C33 ADVコアとC33 PEコアの命令セットでは、以下の特殊レジスタも指定可能です。
 C33 ADV: LCO, LSA, LEA, SOR, TTBR, DP, USP, SSP, IDIR, DBBR
 C33 PE: TTBR, IDIR, DBBR
 %sp, sp: スタックポインタ(SP)またはその内容
 %dp, dp: データポインタ(DP)またはその内容(C33 ADVコア)
 %rc, rc: ループカウンタ(LCO)またはその内容(C33 ADVコア)
 %ra, ra: ループエンドアドレスレジスタ(LEA)またはその内容(C33 ADVコア)

メモリ/アドレス/メモリデータ

[%rb]: レジスタ間接アドレッシング指定
 [%rb]+: ポストインクリメント付きレジスタ間接アドレッシング指定
 [%sp+immX], [%rb+immX], [%rb+symbol+immX]:
 ディスプレースメント付きレジスタ間接アドレッシング指定
 B[XXX]: XXXで指定されるアドレス、またはそのアドレスにストアされているバイトデータ
 H[XXX]: XXXでベースアドレスが指定されるハーフワード領域、またはそこにストアされているハーフワードデータ
 W[XXX]: XXXでベースアドレスが指定されるワード領域、またはそこにストアされているワードデータ

即値

immX: 符号なしXビット即値
 signX: 符号付きXビット即値

シンボル/ラベル

Symbol: アドレスを示すシンボル
 Label: 分岐先ラベル

ビットフィールド

(X): データのビットX
 (X:Y): ビットXからビットYまでのビットフィールド
 {X, Y...}: ビット構成

機能

←: 右側の内容が左側の項目にロードされることを示します。
 +: 加算
 -: 減算
 &: 論理積
 |: 論理和
 ^: 排他的論理和
 !: 論理否定
 x: 乗算

フラグ

RM: リピートモードイネーブルフラグ(C33 ADVコア)
 LM: ループモードイネーブルフラグ(C33 ADVコア)
 PM: PUSH/POPモードフラグ(C33 ADVコア)
 RC: レジスタカウンタ(C33 ADVコア)
 S: 飽和フラグ(C33 ADVコア)
 ME: MMU例外フラグ(C33 ADVコア)
 MO: MACオーバーフローフラグ(C33 PEコアは未サポート)
 DS: 被除数符号フラグ(C33 PEコアは未サポート)
 C: キャリーフラグ
 V: オーバーフローフラグ
 Z: ゼロフラグ
 N: ネガティブフラグ
 -: 変更なし
 ↔: セット(1)またはリセット(0)
 0: リセット(0)

D

○: ディレイド命令として使用可能なことを示します。
 -: ディレイド命令として使用できないことを示します。

注意

- 命令一覧にはS1C33000命令セットの基本命令の他に、拡張命令(xorを除くxで始まる命令)も含まれています。
- "*Italic*" の文字で記載されている命令は、上位互換の拡張命令が用意されていることを示します。
- 命令一覧(2)~(7)はC33 STDコア、C33 ADVコアおよびC33 PEコアに共通です。ただし、一部の命令はC33 ADVとC33 PEで機能が拡張されています(注意に記載)。命令一覧(8)はC33 PEコア専用、(9)以降はC33 ADVコア専用の命令で、C33 STDコアでは使用できません。
- オペランドにsymbol+imm26とありますが、実際に設定可能な値は $0 \leq \text{オフセット値}(\text{symbol} + \text{imm26} - \text{__dp}) < 0x400000$ となります。

命令一覧 (3)

Assembly Programming

分類	ニーモニック		機能	フラグ						D
	オペコード	オペランド		MO	DS	C	V	Z	N	
符号なしハーフワードデータ転送	xld.uh	%rd, [symbol+imm26]	rd(15:0)←H[symbol+imm26], rd(31:16)←0 (*1)	—	—	—	—	—	—	—
		%rd, [%rb+imm26]	rd(15:0)←H[rb+imm26], rd(31:16)←0	—	—	—	—	—	—	—
		%rd, [%sp+imm32]	rd(15:0)←H[sp+imm32], rd(31:16)←0	—	—	—	—	—	—	—
ワードデータ転送	ld.w	%rd, %rs	rd←rs	—	—	—	—	—	—	○
		%sd, %rs	sd←rs	—	—	—	—	—	—	—
		%rd, %ss	rd←ss	—	—	—	—	—	—	—
		%rd, sign6	rd(5:0)←sign6(5:0), rd(31:6)←sign6(5)	—	—	—	—	—	—	○
		%rd, [%rb]	rd←W[rb]	—	—	—	—	—	—	—
		%rd, [%rb]+	rd←W[rb], rb←rb+4	—	—	—	—	—	—	—
		%rd, [%sp+imm6]	rd←W[sp+imm8]; imm8={imm6,00}	—	—	—	—	—	—	—
		[%rb], %rs	W[rb]←rs	—	—	—	—	—	—	—
		[%rb]+, %rs	W[rb]←rs, rb←rb+4	—	—	—	—	—	—	—
		[%sp+imm6], %rs	W[sp+imm8]←rs; imm8={imm6,00}	—	—	—	—	—	—	—
	xld.w	%rd, sign32	rd←sign32	—	—	—	—	—	—	*3
		%rd, symbol+imm32	rd←symbol+imm32	—	—	—	—	—	—	—
		%rd, symbol-imm32	rd←symbol-imm32	—	—	—	—	—	—	—
		%rd, [symbol+imm26]	rd←W[symbol+imm26] (*1)	—	—	—	—	—	—	—
		%rd, [%rb+imm26]	rd←W[rb+imm26]	—	—	—	—	—	—	—
		%rd, [%sp+imm32]	rd←W[sp+imm32]	—	—	—	—	—	—	—
		[symbol+imm26], %rs	W[symbol+imm26]←rs (*1)	—	—	—	—	—	—	—
		[%rb+imm26], %rs	W[rb+imm26]←rs	—	—	—	—	—	—	—
		[%sp+imm32], %rs	W[sp+imm32]←rs	—	—	—	—	—	—	—
論理演算	and	%rd, %rs	rd←rd & rs (*2)	—	—	—	—	↔	↔	○
		%rd, sign6	rd←rd & sign6(符号拡張) (*2)	—	—	—	—	↔	↔	*3
	xand	%rd, sign32	rd←rd & sign32	—	—	—	—	↔	↔	*3
	or	%rd, %rs	rd←rd rs (*2)	—	—	—	—	↔	↔	○
		%rd, sign6	rd←rd sign6(符号拡張) (*2)	—	—	—	—	↔	↔	*3
	xoor	%rd, sign32	rd←rd sign32	—	—	—	—	↔	↔	*3
	xor	%rd, %rs	rd←rd ^ rs (*2)	—	—	—	—	↔	↔	○
		%rd, sign6	rd←rd ^ sign6(符号拡張) (*2)	—	—	—	—	↔	↔	*3
	xxor	%rd, sign32	rd←rd ^ sign32	—	—	—	—	↔	↔	*3
	not	%rd, %rs	rd←!rs (*2)	—	—	—	—	↔	↔	○
		%rd, sign6	rd←!sign6(符号拡張) (*2)	—	—	—	—	↔	↔	*3
	xnot	%rd, sign32	rd←!sign32	—	—	—	—	↔	↔	*3

注意

*1) C33 ADVコアでは、imm26がimm32に拡張されています。

*2) C33 ADVコアでOCフラグが"1"の場合、命令実行後にVフラグがクリアされます。C33 PEコアは命令実行後にVフラグがクリアされます。

*3) オペランドの即値がsign6の範囲内に限り、ディレイド命令として使用可能です。

命令一覧 (4)

Assembly Programming

分類	ニーモニック		機能	フラグ						D
	オペコード	オペランド		MO	DS	C	V	Z	N	
算術演算	add	%rd, %rs	rd←rd + rs	-	-	↔	↔	↔	↔	○
		%rd, imm6	rd←rd + imm6(ゼロ拡張)	-	-	↔	↔	↔	↔	○
		%sp, imm10	sp←sp + imm12(ゼロ拡張); imm12={imm10,00}	-	-	-	-	-	-	○
	xadd	%rd, imm32	rd←rd + imm32	-	-	↔	↔	↔	↔	*1
	adc	%rd, %rs	rd←rd + rs + C	-	-	↔	↔	↔	↔	○
	sub	%rd, %rs	rd←rd - rs	-	-	↔	↔	↔	↔	○
		%rd, imm6	rd←rd - imm6(ゼロ拡張)	-	-	↔	↔	↔	↔	○
		%sp, imm10	sp←sp - imm12(ゼロ拡張); imm12={imm10,00}	-	-	-	-	-	-	○
	xsub	%rd, imm32	rd←rd - imm32	-	-	↔	↔	↔	↔	*1
	sbc	%rd, %rs	rd←rd - rs - C	-	-	↔	↔	↔	↔	○
	cmp	%rd, %rs	rd - rs	-	-	↔	↔	↔	↔	○
		%rd, sign6	rd - sign6(符号拡張)	-	-	↔	↔	↔	↔	○
	xcmp	%rd, sign32	rd - sign32	-	-	↔	↔	↔	↔	*1
	mlt.h	%rd, %rs	alr←rd(15:0) × rs(15:0); 符号付き乗算	-	-	-	-	-	-	○
	mltu.h	%rd, %rs	alr←rd(15:0) × rs(15:0); 符号なし乗算	-	-	-	-	-	-	○
	mlt.w	%rd, %rs	{ahr, alr}←rd × rs; 符号付き乗算	-	-	-	-	-	-	-
	mltu.w	%rd, %rs	{ahr, alr}←rd × rs; 符号なし乗算	-	-	-	-	-	-	-
	div0s (*6)	%rs	符号付き除算第1ステップ; alr = 被除数, rs = 除数	-	↔	-	-	-	↔	-
	div0u (*6)	%rs	符号なし除算第1ステップ; alr = 被除数, rs = 除数	-	0	-	-	-	0	-
	div1 (*6)	%rs	ステップ除算 (*2); alr←商, ahr←余り(符号なし)	-	-	-	-	-	-	-
div2s (*6)	%rs	符号付き除算データ補正1 (*3)	-	-	-	-	-	-	-	
div3s (*6)		符号付き除算データ補正2 (*3); alr←商, ahr←余り	-	-	-	-	-	-	-	
シフト& ローテート	srl	%rd, imm4	右方向論理シフト>>imm4ビット; imm4=0-8, MSBには0をセット (*4)	-	-	-	-	↔	↔	○
		%rd, %rs	右方向論理シフト>>rsビット; rs=0-8, MSBには0をセット (*4)	-	-	-	-	↔	↔	○
	xsrl	%rd, imm5	右方向論理シフト>>imm5ビット; imm5=0-31, MSBには0をセット	-	-	-	-	↔	↔	*5
	sll	%rd, imm4	左方向論理シフト>>imm4ビット; imm4=0-8, LSBには0をセット (*4)	-	-	-	-	↔	↔	○
		%rd, %rs	左方向論理シフト>>rsビット; rs=0-8, LSBには0をセット (*4)	-	-	-	-	↔	↔	○
	xsll	%rd, imm5	左方向論理シフト>>imm5ビット; imm5=0-31, LSBには0をセット	-	-	-	-	↔	↔	*5
	sra	%rd, imm4	右方向算術シフト>>imm4ビット; imm4=0-8, MSBには符号をコピー (*4)	-	-	-	-	↔	↔	○
		%rd, %rs	右方向算術シフト>>rsビット; rs=0-8, MSBには符号をコピー (*4)	-	-	-	-	↔	↔	○
xsra	%rd, imm5	右方向算術シフト>>imm5ビット; imm5=0-31, MSBには符号をコピー	-	-	-	-	↔	↔	*5	

注意

- *1) オペランドの即値がsign6の範囲内に限り、ディレイド命令として使用可能です。
- *2) 32ビット ÷ 32ビットの場合、div1命令を32回実行する必要があります。符号なし除算の場合はdiv1命令によって除算結果がalrおよびahrにロードされます。
- *3) div2s命令とdiv3s命令は符号なし除算では不要です。
- *4) C33 ADVコアとC33 PEコアでは、"imm4=0-8"が"imm5=0-31"、"rs=0-8"が"rs=0-31"に機能が拡張されています。また、C33 ADVではSEフラグが"1"の場合、Cフラグが変化します。
- *5) C33 ADVコアに限り、ディレイド命令として使用可能です。
- *6) C33 PEコアでは未サポートです。これらの命令を実行すると、未定義の命令コードがIDIRレジスタの下位16ビットに格納され、TTBR+12の未定義命令例外割り込みが発生します。

命令一覧 (5)

Assembly Programming

分類	ニーモニック		機能	フラグ						D
	オペコード	オペランド		MO	DS	C	V	Z	N	
シフト& ローテート	<i>sla</i>	<i>%rd, imm4</i>	左方向算術シフト>>imm4ビット; imm4=0-8, LSBには0をセット (*1)	-	-	-	-	↔	↔	○
		<i>%rd, %rs</i>	左方向算術シフト>>rsビット; rs=0-8, LSBには0をセット (*1)	-	-	-	-	↔	↔	○
	<i>xsla</i>	<i>%rd, imm5</i>	左方向算術シフト>>imm5ビット; imm5=0-31, LSBには0をセット	-	-	-	-	↔	↔	*2
	<i>rr</i>	<i>%rd, imm4</i>	右方向ローテート>>imm4ビット; imm4=0-8, LSB → MSB (*1)	-	-	-	-	↔	↔	○
		<i>%rd, %rs</i>	右方向ローテート>>rsビット; rs=0-8, LSB → MSB (*1)	-	-	-	-	↔	↔	○
	<i>xrr</i>	<i>%rd, imm5</i>	右方向ローテート>>imm5ビット; imm5=0-31, LSB → MSB	-	-	-	-	↔	↔	*2
	<i>rl</i>	<i>%rd, imm4</i>	左方向ローテート>>imm4ビット; imm4=0-8, MSB → LSB (*1)	-	-	-	-	↔	↔	○
		<i>%rd, %rs</i>	左方向ローテート>>rsビット; rs=0-8, MSB → LSB (*1)	-	-	-	-	↔	↔	○
ビット操作	<i>xrl</i>	<i>%rd, imm5</i>	左方向ローテート>>imm5ビット; imm5=0-31, MSB → LSB	-	-	-	-	↔	↔	*2
	<i>btst</i>	<i>[%rb], imm3</i>	Z flag←1 if B[rb](imm3)=0	-	-	-	-	↔	-	-
	<i>xbtst</i>	[symbol+imm26], imm3 [%rb+imm26], imm3	Z flag←1 if B[symbol+imm26](imm3)=0 Z flag←1 if B[rb+imm26](imm3)=0	-	-	-	-	↔	-	-
	<i>bclr</i>	<i>[%rb], imm3</i>	B[imm32](imm3)←0	-	-	-	-	-	-	-
	<i>xbclr</i>	[symbol+imm26], imm3 [%rb+imm26], imm3	B[symbol+imm26](imm3)←0 B[rb+imm26](imm3)←0	-	-	-	-	-	-	-
	<i>bset</i>	<i>[%rb], imm3</i>	B[rb](imm3)←1	-	-	-	-	-	-	-
	<i>xbset</i>	[symbol+imm26], imm3 [%rb+imm26], imm3	B[symbol+imm26](imm3)←1 B[rb+imm26](imm3)←1	-	-	-	-	-	-	-
	<i>bnot</i>	<i>[%rb], imm3</i>	B[rb](imm3)←!B[rb](imm3)	-	-	-	-	-	-	-
	<i>xbnot</i>	[symbol+imm26], imm3 [%rb+imm26], imm3	B[symbol+imm26](imm3)←!B[symbol+imm26](imm3) B[rb+imm26](imm3)←!B[rb+imm26](imm3)	-	-	-	-	-	-	-
分岐	<i>jrgt</i>	<i>sign8</i>	pc←pc+sign9 if !Z&!(N^V) is true; sign9={sign8,0}	-	-	-	-	-	-	-
	<i>jrgt.d</i>									
	<i>xjrgt</i>	label+imm32	pc←label+imm32 if !Z&!(N^V) is true	-	-	-	-	-	-	-
	<i>xjrgt.d</i>	sign32	pc←pc+sign32 if !Z&!(N^V) is true	-	-	-	-	-	-	-
	<i>jrge</i>	<i>sign8</i>	pc←pc+sign9 if !(N^V) is true; sign9={sign8,0}	-	-	-	-	-	-	-
	<i>jrge.d</i>									
	<i>xjrge</i>	label+imm32	pc←label+imm32 if !(N^V) is true	-	-	-	-	-	-	-
	<i>xirge.d</i>	sign32	pc←pc+sign32 if !(N^V) is true	-	-	-	-	-	-	-
	<i>jrlt</i>	<i>sign8</i>	pc←pc+sign9 if N^V is true; sign9={sign8,0}	-	-	-	-	-	-	-
	<i>jrlt.d</i>									
注意	<i>xjrlt</i>	label+imm32	pc←label+imm32 if N^V is true	-	-	-	-	-	-	-
	<i>xjrlt.d</i>	sign32	pc←pc+sign32 if N^V is true	-	-	-	-	-	-	-

注意

*1) C33 ADVコアとC33 PEコアでは、"imm4=0-8"が"imm5=0-31"、"rs=0-8"が"rs=0-31"に機能が拡張されています。また、C33 ADVではSEフラグが"1"の場合、Cフラグが変化します。

*2) C33 ADVコアに限り、ディレイド命令として使用可能です。

命令一覧 (6)

Assembly Programming

分類	ニーモニック		機能	フラグ						D
	オペコード	オペランド		MO	DS	C	V	Z	N	
分岐	<i>jrle</i>	<i>sign8</i>	$pc \leftarrow pc + sign9$ if Z (N^V) is true; $sign9 = \{sign8, 0\}$	-	-	-	-	-	-	-
	<i>jrle.d</i>									
	<i>xjrle</i>	$label + imm32$	$pc \leftarrow label + imm32$ if Z (N^V) is true	-	-	-	-	-	-	-
	<i>xjrle.d</i>	<i>sign32</i>	$pc \leftarrow pc + sign32$ if Z (N^V) is true	-	-	-	-	-	-	-
	<i>jrugt</i>	<i>sign8</i>	$pc \leftarrow pc + sign9$ if !Z&!C is true; $sign9 = \{sign8, 0\}$	-	-	-	-	-	-	-
	<i>jrugt.d</i>									
	<i>xjrugt</i>	$label + imm32$	$pc \leftarrow label + imm32$ if !Z&!C is true	-	-	-	-	-	-	-
	<i>xjrugt.d</i>	<i>sign32</i>	$pc \leftarrow pc + sign32$ if !Z&!C is true	-	-	-	-	-	-	-
	<i>jruge</i>	<i>sign8</i>	$pc \leftarrow pc + sign9$ if !C is true; $sign9 = \{sign8, 0\}$	-	-	-	-	-	-	-
	<i>jruge.d</i>									
	<i>xjruge</i>	$label + imm32$	$pc \leftarrow label + imm32$ if !C is true	-	-	-	-	-	-	-
	<i>xjruge.d</i>	<i>sign32</i>	$pc \leftarrow pc + sign32$ if !C is true	-	-	-	-	-	-	-
	<i>jrult</i>	<i>sign8</i>	$pc \leftarrow pc + sign9$ if C is true; $sign9 = \{sign8, 0\}$	-	-	-	-	-	-	-
	<i>jrult.d</i>									
	<i>xjrult</i>	$label + imm32$	$pc \leftarrow label + imm32$ if C is true	-	-	-	-	-	-	-
	<i>xjrult.d</i>	<i>sign32</i>	$pc \leftarrow pc + sign32$ if C is true	-	-	-	-	-	-	-
	<i>jrle</i>	<i>sign8</i>	$pc \leftarrow pc + sign9$ if Z C is true; $sign9 = \{sign8, 0\}$	-	-	-	-	-	-	-
	<i>jrle.d</i>									
	<i>xjrle</i>	$label + imm32$	$pc \leftarrow label + imm32$ if Z C is true	-	-	-	-	-	-	-
	<i>xjrle.d</i>	<i>sign32</i>	$pc \leftarrow pc + sign32$ if Z C is true	-	-	-	-	-	-	-
	<i>jreq</i>	<i>sign8</i>	$pc \leftarrow pc + sign9$ if Z is true; $sign9 = \{sign8, 0\}$	-	-	-	-	-	-	-
	<i>jreq.d</i>									
	<i>xjreq</i>	$label + imm32$	$pc \leftarrow label + imm32$ if Z is true	-	-	-	-	-	-	-
	<i>xjreq.d</i>	<i>sign32</i>	$pc \leftarrow pc + sign32$ if Z is true	-	-	-	-	-	-	-
	<i>jrne</i>	<i>sign8</i>	$pc \leftarrow pc + sign9$ if !Z is true; $sign9 = \{sign8, 0\}$	-	-	-	-	-	-	-
	<i>jrne.d</i>									
	<i>xjrne</i>	$label + imm32$	$pc \leftarrow label + imm32$ if !Z is true	-	-	-	-	-	-	-
	<i>xjrne.d</i>	<i>sign32</i>	$pc \leftarrow pc + sign32$ if !Z is true	-	-	-	-	-	-	-
	<i>call</i>	<i>sign8</i>	$sp \leftarrow sp - 4$, $W[sp] \leftarrow pc + 2$, $pc \leftarrow pc + sign9$; $sign9 = \{sign8, 0\}$	-	-	-	-	-	-	-
	<i>call.d</i>	<i>%rb</i>	$sp \leftarrow sp - 4$, $W[sp] \leftarrow pc + 2$, $pc \leftarrow rb$	-	-	-	-	-	-	-
	<i>xcall</i>	$label + imm32$	$sp \leftarrow sp - 4$, $W[sp] \leftarrow pc + 2$, $pc \leftarrow label + imm32$	-	-	-	-	-	-	-
	<i>xcall.d</i>	<i>sign32</i>	$sp \leftarrow sp - 4$, $W[sp] \leftarrow pc + 2$, $pc \leftarrow pc + sign32$	-	-	-	-	-	-	-
	<i>jp</i>	<i>sign8</i>	$pc \leftarrow pc + sign9$; $sign9 = \{sign8, 0\}$	-	-	-	-	-	-	-
	<i>jp.d</i>	<i>%rb</i>	$pc \leftarrow rb$	-	-	-	-	-	-	-
	<i>xjp</i>	$label + imm32$	$pc \leftarrow pc + label + imm32$	-	-	-	-	-	-	-
	<i>xjp.d</i>	<i>sign32</i>	$pc \leftarrow pc + sign32$	-	-	-	-	-	-	-

注意

命令一覧 (7)

Assembly Programming

分類	ニーモニック		機能	フラグ						D
	オペコード	オペランド		MO	DS	C	V	Z	N	
分岐	ret		pc←W[sp], sp←sp+4	-	-	-	-	-	-	-
	ret.d									
	reti		psr←W[sp], sp←sp+4, pc←W[sp], sp←sp+4	↔	↔	↔	↔	↔	↔	-
	ret.d		デバッグ例外処理ルーチンからのリターン (S5U1C33xxxHソフトウェア用)	-	-	-	-	-	-	-
	int	imm2	sp←sp-4, W[sp]←pc+2, sp←sp-4, W[sp]←psr, pc←ソフトウェア例外ベクタ	-	-	-	-	-	-	-
即値拡張	brk		デバッグ例外 (S5U1C33xxxHソフトウェア用)	-	-	-	-	-	-	-
	ext	imm13	次の命令の即値またはオペランドを拡張	-	-	-	-	-	-	-
プッシュ& ポップ	pushn	%rs	"sp←sp-4, W[sp]←rn"をrn=rs~r0まで繰り返し (*1)	-	-	-	-	-	-	-
	popn	%rd	"rn←W[sp], sp←sp+4"をrn=r0~rdまで繰り返し (*1)	-	-	-	-	-	-	-
積和演算	mac (*4)	%rs	"{ahr, alr}←{ahr, alr} + H[<rs+1>]+ × H[<rs+2>]+"をrs回繰り返し (*2)	↔	-	-	-	-	-	-
システム制御	nop		ノーオペレーション; pc←pc+2	-	-	-	-	-	-	-
	halt		Haltモードに設定	-	-	-	-	-	-	-
	slp		Sleepモードに設定	-	-	-	-	-	-	-
その他	scan0 (*4)	%rd, %rs	rsの最上位バイト中の"0"ビットをスキャン, rd←MSBからのオフセット (*3)	-	-	↔	0	↔	0	○
	scan1 (*4)	%rd, %rs	rsの最上位バイト中の"1"ビットをスキャン, rd←MSBからのオフセット (*3)	-	-	↔	0	↔	0	○
	swap	%rd, %rs	rd(31:24)←rs(7:0), rd(23:16)←rs(15:8), rd(15:8)←rs(23:16), rd(7:0)←rs(31:24)	-	-	-	-	-	-	○
	mirror (*4)	%rd, %rs	rd(31:24)←rs(24:31), rd(23:16)←rs(16:23), rd(15:8)←rs(8:15), rd(7:0)←rs(0:7)	-	-	-	-	-	-	○

注意

*1) C33 ADVコアでPMフラグが"1"の場合、RC[3:0]に格納されている番号のレジスタからプッシュ/ポップします。実行後、PMフラグとRC[3:0]が変化します。

*2) <rs+1>, <rs+2>: rsレジスタに続く2つのレジスタの内容 (例: rs=r0: <rs+1>=r1, <rs+2>=r2; rs=r15: <rs+1>=r0, <rs+2>=r1), 1回の演算ごとにポストインクリメント(+2) mac命令はオプションの乗除算回路を内蔵した機種に限り実行可能です。

*3) C33 ADVコアでSWフラグが"1"の場合、rsの全ビット(32ビット)をスキャンします。

*4) C33 PEコアでは未サポートです。これらの命令を実行すると、未定義の命令コードがIDIRレジスタの下位16ビットに格納され、TTBR+12の未定義命令例外割り込みが発生します。

分類	ニーモニック		機能	フラグ				D
	オペコード	オペランド		C	V	Z	N	
分岐	jpr	%rb	pc←pc + rb	–	–	–	–	–
	jpr.d							
システム制御	psrset	imm5	psr(imm5)←1	↔	↔	↔	↔	–
	psrclr	imm5	psr(imm5)←0	↔	↔	↔	↔	–
プッシュ& ポップ	push	%rs	sp←sp-4, W[sp]←rs	–	–	–	–	–
	pushs	%ss	sp←sp-4, W[sp]←ahr, sp←sp-4, W[sp]←alr (if ss=ahr) sp←sp-4, W[sp]←alr (if ss=alr)	–	–	–	–	–
	pop	%rd	rd←W[sp], sp←sp+4	–	–	–	–	–
	pops	%sd	alr←W[sp], sp←sp+4, ahr←W[sp], sp←sp+4 (if sd=ahr)	–	–	–	–	–
			alr←W[sp], sp←sp+4 (if sd=alr)					
交換	swaph	%rd, %rs	rd(31:24)←rs(23:16), rd(23:16)←rs(31:24), rd(15:8)←rs(7:0), rd(7:0)←rs(15:8)	–	–	–	–	–
コプロセッサ	ld.c	%rd, imm4	rd(7:0)←コプロセッサレジスタ(imm4)	–	–	–	–	–
		imm4, %rs	コプロセッサレジスタ(imm4)←rs(7:0)	–	–	–	–	–
	do.c	imm6	コプロセッサにコマンド(imm6)を発行	–	–	–	–	–
	ld.cf		psr(3:0)←コプロセッサフラグ	↔	↔	↔	↔	–

注意

命令一覧 (9) – C33 ADVコア用

Assembly Programming

分類	ニーモニック		機能	フラグ													D
	オペコード	オペランド		RM	LM	PM	RC	S	ME	MO	DS	C	V	Z	N		
符号付きバイトデータ転送	ld.b	%rd, [%dp+imm6]	rd(7:0)←B[dp+imm6], rd(31:8)←B[dp+imm6](7)	-	-	-	-	-	-	-	-	-	-	-	-		
		[%dp+imm6], %rs	B[dp+imm6]←rs(7:0)	-	-	-	-	-	-	-	-	-	-	-	-		
	ald.b	%rd, [symbol+imm19]	rd(7:0)←B[symbol+imm19], rd(31:8)←B[symbol+imm19](7)	-	-	-	-	-	-	-	-	-	-	-	-		
		[symbol+imm19], %rs	B[symbol+imm19]←rs(7:0)	-	-	-	-	-	-	-	-	-	-	-	-		
符号なしバイトデータ転送	xld.b	%rd, [%dp+imm32]	rd(7:0)←B[dp+imm32], rd(31:8)←B[dp+imm32](7)	-	-	-	-	-	-	-	-	-	-	-	-		
		[%dp+imm32], %rs	B[dp+imm32]←rs(7:0)	-	-	-	-	-	-	-	-	-	-	-	-		
	ld.ub	%rd, [%dp+imm6]	rd(7:0)←B[dp+imm6], rd(31:8)←0	-	-	-	-	-	-	-	-	-	-	-	-		
		ald.ub	%rd, [symbol+imm19]	rd(7:0)←B[symbol+imm19], rd(31:8)←0	-	-	-	-	-	-	-	-	-	-	-		
符号付きハーフワードデータ転送	xld.ub	%rd, [%dp+imm32]	rd(7:0)←B[dp+imm32], rd(31:8)←0	-	-	-	-	-	-	-	-	-	-	-	-		
		%rd, [%dp+imm6]	rd(15:0)←H[dp+imm7], rd(31:16)←H[dp+imm7](15); imm7={imm6,0}	-	-	-	-	-	-	-	-	-	-	-	-		
	ld.h	[%dp+imm6], %rs	H[dp+imm7]←rs(15:0); imm7={imm6,0}	-	-	-	-	-	-	-	-	-	-	-	-		
		ald.h	%rd, [symbol+imm19]	rd(15:0)←H[symbol+imm19], rd(31:16)←H[symbol+imm19](15)	-	-	-	-	-	-	-	-	-	-	-		
符号なしハーフワードデータ転送	xld.h	[symbol+imm19], %rs	H[symbol+imm19]←rs(15:0)	-	-	-	-	-	-	-	-	-	-	-	-		
		%rd, [%dp+imm32]	rd(15:0)←H[dp+imm32], rd(31:16)←H[dp+imm32](15)	-	-	-	-	-	-	-	-	-	-	-	-		
	ld.uh	[%dp+imm32], %rs	H[dp+imm32]←rs(15:0)	-	-	-	-	-	-	-	-	-	-	-	-		
		ald.uh	%rd, [%dp+imm6]	rd(15:0)←H[dp+imm7], rd(31:16)←0; imm7={imm6,0}	-	-	-	-	-	-	-	-	-	-	-		
ワードデータ転送	xld.uh	%rd, [symbol+imm19]	rd(15:0)←H[symbol+imm19], rd(31:16)←0	-	-	-	-	-	-	-	-	-	-	-	-		
		%rd, [%dp+imm32]	rd(15:0)←H[dp+imm32], rd(31:16)←0	-	-	-	-	-	-	-	-	-	-	-	-		
	ld.w	%rd, [%dp+imm6]	rd←W[dp+imm8]; imm8={imm6,00}	-	-	-	-	-	-	-	-	-	-	-	-		
		[%dp+imm6], %rs	W[dp+imm8]←rs; imm8={imm6,00}	-	-	-	-	-	-	-	-	-	-	-	-		
算術演算	ald.w	%rd, [symbol+imm19]	rd←W[symbol+imm19]	-	-	-	-	-	-	-	-	-	-	-	-		
		[symbol+imm19], %rs	W[symbol+imm19]←rs	-	-	-	-	-	-	-	-	-	-	-	-		
	xld.w	%rd, [%dp+imm32]	rd←W[dp+imm32]	-	-	-	-	-	-	-	-	-	-	-	-		
		[%dp+imm32], %rs	W[dp+imm32]←rs	-	-	-	-	-	-	-	-	-	-	-	-		
分岐	add	%rd, %dp	rd←rd + dp	-	-	-	-	-	-	-	-	-	-	-	○		
	mli.hw	%rd, %rs	{ahr, alr}←rd(31:0) × rs(15:0); 符号付き乗算	-	-	-	-	-	-	-	-	-	-	-	-		
	div.w	%rs	alr←alr/rs(商), ahr←alr/rs(余り); 符号付き除算	-	-	-	-	-	-	-	↔	-	-	↔	-		
	divu.w	%rs	alr←alr/rs(商), ahr←alr/rs(余り); 符号なし除算	-	-	-	-	-	-	-	0	-	-	0	-		
システム制御	jpr	%rb	pc←pc + rb	-	-	-	-	-	-	-	-	-	-	-	-		
	jpr.d																
	retm		r0←W[0x1C], pc←W[0x18]	-	-	-	-	-	0	-	-	-	-	-	-		
システム制御	psrset	imm5	psr(imm5)←1	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔		
	psrclr	imm5	psr(imm5)←0	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔		
注意																	

命令一覧 (10) – C33 ADVコア用

Assembly Programming

分類	ニーモニック		機能	フラグ													D
	オペコード	オペランド		RM	LM	PM	RC	S	ME	MO	DS	C	V	Z	N		
機能拡張	ext	%rs, op, imm2	3オペランド実行とimm2ビットのポストシフト; op=sll, srl, sra	-	-	-	-	-	-	-	-	-	-	-	-	-	
		op, imm2	imm2ビットのポストシフト; op=sll, srl, sra	-	-	-	-	-	-	-	-	-	-	-	-	-	
		%rs	直後の命令を3オペランドに拡張して実行	-	-	-	-	-	-	-	-	-	-	-	-	-	
		cond	条件付き実行; cond=gt, ge, lt, le, ugt, uge, ult, ule, eq, ne	-	-	-	-	-	-	-	-	-	-	-	-	-	
プッシュ& ポップ	push	%rs	sp←sp-4, W[sp]←rs	-	-	-	-	-	-	-	-	-	-	-	-		
	pushs	%ss	sp←sp-4, W[sp]←ss (if ss=psr, sp) "sp←sp-4, W[sp]←sn"をsn=ss~alrまで繰り返し(if ss=alr-pc) (*1)	-	-	↔	↔	-	-	-	-	-	-	-	-		
	pop	%rd	rd←W[sp], sp←sp+4	-	-	-	-	-	-	-	-	-	-	-	-		
	pops	%sd	sd←W[sp], sp←sp+4 (if sd=psr, sp) (sd==psr)→ "sn←W[sp], sp←sp+4"をsn=alr~sdまで繰り返し(if sd=alr-pc) (*1)	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	-		
積和演算	mac.hw	%rs	"{ahr, alr}←{ahr, alr} + W[<rs+1>] × H[<rs+2>]"をrs回繰り返し (*2)	-	-	-	-	-	↔	-	-	-	-	-	-		
	mac.w	%rs	"{ahr, alr}←{ahr, alr} + W[<rs+1>] × W[<rs+2>]"をrs回繰り返し (*2)	-	-	-	-	-	↔	-	-	-	-	-	-		
	mac1.h	%rd, %rs	{ahr, alr}←{ahr, alr} + rd(15:0) × rs(15:0)	-	-	-	-	-	↔	-	-	-	-	-	-		
	mac1.hw	%rd, %rs	{ahr, alr}←{ahr, alr} + rd(31:0) × rs(15:0)	-	-	-	-	-	↔	-	-	-	-	-	-		
	mac1.w	%rd, %rs	{ahr, alr}←{ahr, alr} + rd(31:0) × rs(31:0)	-	-	-	-	-	↔	-	-	-	-	-	-		
	macclr		{ahr, alr}←0, MO←0	-	-	-	-	-	0	-	-	-	-	-	-		
飽和处理付き データ転送	sat.b	%rd, %rs	rd←rs(b) if -128≤rs≤127, rd←0xFFFFF80 if rs<-128, rd←0x0000007F if rs>127	-	-	-	↔	-	-	-	-	-	-	-	-		
	sat.ub	%rd, %rs	rd←rs(ub) if rs≤255, rd←0x000000FF if rs>255	-	-	-	↔	-	-	-	-	-	-	-	-		
	sat.h	%rd, %rs	rd←rs(h) if -32768≤rs≤32767, rd←0xFFFF8000 if rs<-32768, rd←0x00007FFF if rs>32767	-	-	-	↔	-	-	-	-	-	-	-	-		
	sat.uh	%rd, %rs	rd←rs(uh) if rs≤65535, rd←0x0000FFFF if rs>65535	-	-	-	↔	-	-	-	-	-	-	-	-		
	sat.w	%rd, %rs	rd←0x80000000 if V=1&N=0, rd←0x7FFFFFFF if V=1&N=1, それ以外はrd←rs	-	-	-	↔	-	-	-	-	-	-	-	-		
	sat.uw	%rd, %rs	rd←0xFFFFFFFF if C=1, それ以外はrd←rs	-	-	-	↔	-	-	-	-	-	-	-	-		
ループ& リピート	loop	%rc, %ra	pc+2からraまでの命令をrc+1回実行	-	↔	-	-	-	-	-	-	-	-	-	-		
		%rc, imm4	pc+2からpc+2+(imm4×2)までの命令をrc+1回実行	-	↔	-	-	-	-	-	-	-	-	-	-		
		imm4', imm4	pc+2からpc+2+(imm4×2)までの命令をimm4'+1回実行	-	↔	-	-	-	-	-	-	-	-	-	-		
	repeat	%rc	pc+2の命令をrc+1回実行	↔	-	-	-	-	-	-	-	-	-	-	-		
	imm4	pc+2の命令をimm4+1回実行	↔	-	-	-	-	-	-	-	-	-	-	-			
交換	swaph	%rd, %rs	rd(31:24)←rs(23:16), rd(23:16)←rs(31:24), rd(15:8)←rs(7:0), rd(7:0)←rs(15:8)	-	-	-	-	-	-	-	-	-	-	-	-		
コプロセッサ	ld.c	%rd, imm4	rd(7:0)←コプロセッサレジスタ(imm4)	-	-	-	-	-	-	-	-	-	-	-	-		
		imm4, %rs	コプロセッサレジスタ(imm4)←rs(7:0)	-	-	-	-	-	-	-	-	-	-	-	-		
	do.c	imm6	コプロセッサにコマンド(imm6)を発行	-	-	-	-	-	-	-	-	-	-	-	-		
	ld.cf		psr(3:0)←コプロセッサフラグ	-	-	-	-	-	-	-	↔	↔	↔	↔	-		

注意

*1) PMフラグが"1"の場合、RC[3:0]に格納されている番号のレジスタからプッシュ/ポップします。

*2) <rs+1>, <rs+2>: rsレジスタに続く2つのレジスタの内容 (例: rs=r0: <rs+1>=r1, <rs+2>=r2; rs=r15: <rs+1>=r0, <rs+2>=r1), 1回の演算ごとにポストインクリメント

拡張命令の展開形式 (1)

Assembly Programming

拡張命令		展開形式				
オペコード	オペランド	条件1	条件2	条件3	条件4	条件5
xld.b	%rd, [symbol+imm26]	-medda32未指定時	—	—	—	—
xld.ub	[symbol+imm26], %rs *1	ext (symbol+imm26)(25:13)				
xld.h		ext (symbol+imm26)(12:0)				
xld.uh	例) xld.w %rd,[symbol+imm26]	ld.w %rd,[%r15]				
xld.w	%rd, [%rb+imm26]	imm26=0	1<imm26≤0x1fff	imm26>0x1fff	—	—
	[%rb+imm26], %rs *1	ld.w %rd,[%rb]	ext imm26(12:0)	ext imm26(25:13)		
	例) xld.w %rd,[%rb+imm26]		ld.w %rd,[%rb]	ext imm26(12:0)		
				ld.w %rd,[%rb]		
xld.b	%rd, [%sp+imm32]	imm32≤0x3f	0x3f<imm32≤0x7ffff	imm32>0x7ffff	—	—
xld.ub	[%sp+imm32], %rs *1	ld.b %rd,[%sp+imm32(5:0)]	ext imm32(18:6)	ext imm32(31:19)		
	例) xld.b %rd,[%sp+imm32]		ld.b %rd,[%sp+imm32(5:0)]	ext imm32(18:6)		
				ld.b %rd,[%sp+imm32(5:0)]		
xld.h	%rd, [%sp+imm32]	imm32≤0x7f	0x7f<imm32≤0x7ffff	imm32>0x7ffff	—	—
xld.uh	[%sp+imm32], %rs *1	ld.h %rd,[%sp+imm32(6:1)]	ext imm32(18:6)	ext imm32(31:19)		
	例) xld.h %rd,[%sp+imm32]		ld.h %rd,[%sp+imm32(5:0)]	ext imm32(18:6)		
				ld.h %rd,[%sp+imm32(5:0)]		
xld.w	%rd, [%sp+imm32]	imm32≤0xff	0xff<imm32≤0x7ffff	imm32>0x7ffff	—	—
	[%sp+imm32], %rs *1	ld.w %rd,[%sp+imm32(7:2)]	ext imm32(18:6)	ext imm32(31:19)		
	例) xld.w %rd,[%sp+imm32]		ld.w %rd,[%sp+imm32(5:0)]	ext imm32(18:6)		
				ld.w %rd,[%sp+imm32(5:0)]		
	%rd, sign32	32≤sign32≤31	-262144≤sign32<-32 or 31<sign32≤262143	sign32<-262144 or 262143<sign32	—	—
	例) xld.w %rd,sign32	ld.w %rd,sign32(5:0)	ext sign32(18:6)	ext sign32(31:19)		
			ld.w %rd,sign32(5:0)	ext sign32(18:6)		
				ld.w %rd,sign32(5:0)		
	%rd, symbol+imm32	無条件	—	—	—	—
	例) xld.w %rd,symbol+imm32	ext (symbol+imm32)@h				
		ext (symbol+imm32)@m				
		ld.w %rd,(symbol+imm32)@l				
	%rd, symbol-imm32	無条件	—	—	—	—
	例) xld.w %rd,symbol-imm32	ext (symbol-imm32)@h				
		ext (symbol-imm32)@m				
		ld.w %rd,(symbol-imm32)@l				

注意

*1) これらのオペランドはxld.b、xld.h、xld.w命令でのみ使用可能です。

拡張命令の展開形式 (2)

Assembly Programming

拡張命令		展開形式				
オペコード	オペランド	条件1	条件2	条件3	条件4	条件5
xld.b xld.ub xld.h xld.uh xld.w	%rd, [symbol+imm26] 例) xld.w %rd,[symbol+imm26]	-medda32指定時 ext (symbol+imm26)@h ext (symbol+imm26)@m ld.w %rd,(symbol+imm26)@l ld.w %rd,[%rd]	—	—	—	—
xld.b xld.h xld.w	[symbol+imm26], %rs 例) xld.w [symbol+imm26],%rs	%rs≠%r0, -medda32指定時 pushn %r0 ext (symbol+imm26)@h ext (symbol+imm26)@m ld.w %r0,(symbol+imm26)@l ld.w [%r0],%rs popn %r0	%rs=%r0, -medda32指定時 pushn %r1 (or push %r1)*1 ext (symbol+imm26)@h ext (symbol+imm26)@m ld.w %r1,(symbol+imm26)@l ld.w [%r1],%r0 popn %r1 (or pop %r1)*1	—	—	—
xand xor xxor xnot	%rd, sign32 例) xand %rd,sign32	-32≤sign32≤31 and %rd,sign32(5:0)	-262144≤sign32<-32 or 31<sign32≤262143 ext sign32(18:6) and %rd,sign32(5:0)	sign32<-262144 or 262143<sign32 ext sign32(31:19) ext sign32(18:6) and %rd,sign32(5:0)	—	—
xadd xsub	%rd, imm32 例) xadd %rd,imm32	imm32≤0x3f add %rd,imm32(5:0)	0x3f<imm32≤0x7fff ext imm32(18:6) add %rd,imm32(5:0)	imm32>0x7fff ext imm32(31:19) ext imm32(18:6) add %rd,imm32(5:0)	—	—
xcmp	%rd, sign32 例) xcmp %rd,sign32	-32≤sign32≤31 cmp %rd,sign32(5:0)	262144≤sign32<-32 or 31<sign32≤262143 ext sign32(18:6) cmp %rd,sign32(5:0)	sign32<-262144 or 262143<sign32 ext sign32(31:19) ext sign32(18:6) cmp %rd,sign32(5:0)	—	—
xslr xslr xsra xsla xrr xrl	%rd, imm5 例) xslr %rd,imm5	imm5≤8 srl %rd,imm5(3:0)	8<imm5<16 srl %rd,0x8 srl %rd,imm5(2:0)	imm5=16 srl %rd,0x8 srl %rd,0x8	16<imm5≤24 srl %rd,0x8 srl %rd,0x8 srl %rd,imm5(3:0)	imm5>24 srl %rd,0x8 srl %rd,0x8 srl %rd,0x8 srl %rd,imm5(2:0)

注意

*1) -mc33advまたは-mc33peオプション指定時

拡張命令の展開形式 (3)

Assembly Programming

拡張命令		展開形式				
オペコード	オペランド	条件1	条件2	条件3	条件4	条件5
xbtst xbclr xbset xbnot	[symbol+imm26], imm3 例) xbtst [symbol+imm26],imm3	-medda32未指定時	-medda32指定時	—	—	—
		ext (symbol+imm26)(25:13) ext (symbol+imm26)(12:0) bttst [%r15],imm3	pushn %r0 ext (symbol+imm26)@h ext (symbol+imm26)@m ld.w %r0,(symbol+imm26)@l bttst [%r0],imm3 popn %r0			
	[%rb+imm26], imm3 例) xbtst [%rb+imm26],imm3	imm26=0	1<imm26≤0x1fff	imm26>0x1fff	—	—
		bttst %rd,[%rb]	ext imm26(12:0) bttst %rd,[%rb]	ext imm26(25:13) ext imm26(12:0) bttst %rd,[%rb]		
scall scall.d sjp sjp.d sjr*1 sjr*1.d	label+imm32 例) scall label+imm32	無条件	—	—	—	—
		ext (label+imm32)@rm call (label+imm32)@rl				
	sign22 例) scall sign22	-256≤sign22≤254	-2097152≤sign22<-256 or 254<sign22≤2097150	—	—	—
		call sign22(8:1)	ext sign22(21:9) call sign22(8:1)			
xcall xcall.d xjp xjp.d xjr*2 xjr*2.d	label+imm32 例) xcall label+imm32	無条件	—	—	—	—
		ext (label+imm32)@rh ext (label+imm32)@rm call (label+imm32)@rl				
	sign32 例) xcall sign32	-256≤sign32≤254	-2097152≤sign32<-256 or 254<sign32≤2097150	sign32<-2097152 or 2097150<sign32	—	—
		call sign32(8:1)	ext sign32(21:9) call sign32(8:1)	ext sign32(31:22)<<0x3 ext sign32(21:9) call sign32(8:1)		

注意

*1) sjreq, sjreq.d, sjrne, sjrne.d, sjrgt, sjrgt.d, sjrge, sjrge.d, sjrlt, sjrlt.d, sjrle, sjrle.d, sjrugt, sjrugt.d, sjruge, sjruge.d, sjrult, sjrult.d, sjrule, sjrule.d

*2) xjreq, xjreq.d, xjrne, xjrne.d, xjrgt, xjrgt.d, xjrge, xjrge.d, xjrlt, xjrlt.d, xjrle, xjrle.d, xjrugt, xjrugt.d, xjruge, xjruge.d, xjrult, xjrult.d, xjrule, xjrule.d

拡張命令		展開形式				
オペコード	オペランド	条件1	条件2	条件3	条件4	条件5
ald.b ald.ub ald.h ald.uh ald.w	%rd, [symbol+imm19] [symbol+imm19], %rs *1 例) ald.w %rd,[symbol+imm19]	-medda32未指定時 ext (symbol+imm19) (18:6) ld.w %rd, [%dp+(symbol+imm19)(5:0)]	–	–	–	–
xld.b xld.ub xld.h xld.uh xld.w	%rd, [symbol+imm32] [symbol+imm32], %rs *2 例) xld.w %rd,[symbol+imm32]	-medda32未指定時 ext (symbol+imm32) (31:19) ext (symbol+imm32) (18:6) ld.w %rd, [%dp+(symbol+imm32)(5:0)]	–	–	–	–
xld.b xld.ub	%rd, [%dp+imm32] [%dp+imm32], %rs *2 例) xld.b %rd,[%dp+imm32]	imm32≤0x3f ld.b %rd,[%dp+imm32(5:0)]	0x3f<imm32≤0x7fff ext imm32(18:6) ld.b %rd,[%dp+imm32(5:0)]	imm32>0x7fff ext imm32(31:19) ext imm32(18:6) ld.b %rd,[%dp+imm32(5:0)]	–	–
xld.h xld.uh	%rd, [%dp+imm32] [%dp+imm32], %rs *2 例) xld.h %rd,[%dp+imm32]	imm32≤0x7f ld.h %rd,[%dp+imm32(6:1)]	0x7f<imm32≤0x7fff ext imm32(18:6) ld.h %rd,[%dp+imm32(5:0)]	imm32>0x7fff ext imm32(31:19) ext imm32(18:6) ld.h %rd,[%dp+imm32(5:0)]	–	–
xld.w	%rd, [%dp+imm32] [%dp+imm32], %rs *2 例) xld.w %rd,[%dp+imm32]	imm32≤0xff ld.w %rd,[%dp+imm32(7:2)]	0xff<imm32≤0x7fff ext imm32(18:6) ld.w %rd,[%dp+imm32(5:0)]	imm32>0x7fff ext imm32(31:19) ext imm32(18:6) ld.w %rd,[%dp+imm32(5:0)]	–	–
xsrl xsll xsra xsla xrr xrl	%rd, imm5 例) xsrl %rd,imm5	無条件 srl %rd,imm5	–	–	–	–

注意
本ページの拡張命令はC33 ADVコア専用で、-mc33advオプションを指定した場合にのみ使用可能です。

*1) これらのオペランドはald.b、ald.h、ald.w命令でのみ使用可能です。

*2) これらのオペランドはxld.b、xld.h、xld.w命令でのみ使用可能です。

拡張命令		展開形式				
オペコード	オペランド	条件1	条件2	条件3	条件4	条件5
ald.b ald.ub ald.h ald.uh ald.w	%rd, [symbol+imm19] 例) ald.w %rd,[symbol+imm19]	-medda32指定時 ext (symbol+imm19)@m ld.w %rd,(symbol+imm19)@l ld.w %rd,[%rd]	–	–	–	–
ald.b ald.h ald.w	[symbol+imm19], %rs 例) ald.w [symbol+imm19],%rs	%rs≠%r0, -medda32指定時 pushn %r0 ext (symbol+imm19)@m ld.w %r0,(symbol+imm19)@l ld.w [%r0],%rs popn %r0	%rs=%r0, -medda32指定時 push %r1 ext (symbol+imm19)@m ld.w %r1,(symbol+imm19)@l ld.w [%r1],%r0 pop %r1	–	–	–
xld.b xld.ub xld.h xld.uh xld.w	%rd, [symbol+imm32] 例) xld.w %rd,[symbol+imm32]	-medda32指定時 ext (symbol+imm32)@h ext (symbol+imm32)@m ld.w %rd,(symbol+imm32)@l ld.w %rd,[%rd]	–	–	–	–
xld.b xld.h xld.w	[symbol+imm32], %rs 例) xld.w [symbol+imm32],%rs	%rs≠%r0, -medda32指定時 pushn %r0 ext (symbol+imm32)@h ext (symbol+imm32)@m ld.w %r0,(symbol+imm32)@l ld.w [%r0],%rs popn %r0	%rs=%r0, -medda32指定時 push %r1 ext (symbol+imm32)@h ext (symbol+imm32)@m ld.w %r1,(symbol+imm32)@l ld.w [%r1],%r0 pop %r1	–	–	–

注意

本ページの拡張命令はC33 ADVコア専用で、-mc33advオプションを指定した場合にのみ使用可能です。

セイコーエプソン株式会社

半導体事業部 IC 営業部

<IC 国内営業グループ>

東京 〒191-8501 東京都日野市日野421-8
TEL (042) 587-5313 (直通) FAX (042) 587-5116

大阪 〒541-0059 大阪市中央区博労町3-5-1 エプソン大阪ビル15F
TEL (06) 6120-6000 (代表) FAX (06) 6120-6100

ドキュメントコード : 410560906
2006年 5月 作成 ㊦B
2009年10月 改訂 ㊦D