EPSON
EXCEED YOUR VISION

CMOS 16-BIT SINGLE CHIP MICROCONTROLLER
(S1C17 Family Cコンパイラパッケージ) (Ver. 2.4.0)

S5U1C17001C

マニュアル

セイコーエプソン株式会社

Rev.1.0

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、予告無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、これらに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. 製品および弊社が提供する技術を輸出等するにあたっては「外国為替および外国貿易法」を遵守し、当該法令の定める手続きが必要です。大量破壊兵器の開発等およびその他の軍事用途に使用する目的をもって製品および弊社が提供する技術を費消、再販売または輸出等しないでください。
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。

S5U1C17001C Manual 改訂履歴

コードNo.	ページ	章/節	内容
411086411	—	マニュアル全体	バージョン番号を、2.3.0から2.4.0へ変更
	—	マニュアル全体	説明変更 Windowsのバージョンに関する記述を変更または削除
	—	表紙裏	注意文を最新版に変更し、コピーライトを2013から2015に変更
	ii	マニュアルの表記	説明削除 「サンプル」の説明文から1文削除
	2-1	2.1 動作環境	説明削除 「ユーザーアカウント」の説明を削除
	2-4	2.2 インストール方法	説明変更 インストールされるファイルに「lcd17usb.dll」を追加
	12-4	12.1.4 makeファイル	説明変更 「MOTOSIZE」を「MOTOPROGSIZE」に変更

このページはブランクです。

製品型番体系

●デバイス

S1　C　17xxx　F　00E1　00

- 00 ■ 梱包仕様
  - 00 : テープ&リール以外
  - 0A : TCP BL　2方向
  - 0B : テープ&リール BACK
  - 0C : TCP BR　2方向
  - 0D : TCP BT　2方向
  - 0E : TCP BD　2方向
  - 0F : テープ&リール FRONT
  - 0G : TCP BT　4方向
  - 0H : TCP BD　4方向
  - 0J : TCP SL　2方向
  - 0K : TCP SR　2方向
  - 0L : テープ&リール LEFT
  - 0M : TCP ST　2方向
  - 0N : TCP SD　2方向
  - 0P : TCP ST　4方向
  - 0Q : TCP SD　4方向
  - 0R : テープ&リール RIGHT
  - 99 : 梱包仕様未定
- 00E1 ■ 仕様
- F ■ 形状
  - [D: ベアチップ、F: QFP、B: BGA]
- 17xxx ■ 機種番号
- C ■ 機種名称
  - [C: マイコン、デジタル製品]
- S1 ■ 製品分類
  - [S1: 半導体]

●開発ツール

S5U1　C　17000　Y2　1　00

- 00 ■ 梱包仕様
  - [00: 標準梱包]
- 1 ■ バージョン
  - [1: Version 1]
- Y2 ■ ツール種類
  - Hx : ICE
  - Dx : 評価ボード
  - Ex : ROMエミュレーションボード
  - Mx : 外部ROM用エミュレーションメモリ
  - Tx : 実装用ソケット
  - Cx : コンパイラパッケージ
  - Sx : ミドルウェアパッケージ
  - Yx : ライタソフトウェア
- 17000 ■ 対応機種番号
  - [17xxx: S1C17xxx用]
- C ■ ツール分類
  - [C: マイコン用]
- S5U1 ■ 製品分類
  - [S5U1: 半導体用開発ツール]

S5U1C17001C Manual

はじめに

本書はS1C17 Familyの全機種に共通するソフトウェア開発ツール「S1C17 Family Cコンパイラパッケージ」による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PAファイル（提出用データ）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言語（ANSI C準拠）に関する知識およびCソースプログラムの作成方法
- GNU 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PAファイル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章の必要個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。

S1C17*xxx*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
S1C17コアマニュアル
　CPUコアの機能と動作、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章でデバッガ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。

S1C17 Family In-Circuit Debugger Manual
　ICD Mini（S5U1C17001H）の取り扱い方法について

■各ツールの詳細について

5～12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

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。

■その他開発ツールの名称

ICD: ICD Mini(S5U1C17001H)またはICDボードを示しています。

ー 目 次 ー

S5U1C17001C Manual

1 概要

1 概要

1.1 特長

S1C17 Family Cコンパイラパッケージには、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PSAファイル(ROMデータ)やPAファイル(提出用データ)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S1C17 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コンパイラは S1C17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S1C17コア命令セットをサポート
S1C17コア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。

• 未使用関数の検出機能(C言語のみ)
プロジェクト内で一度も使用されていないC言語の関数を検出します。未使用関数の検出には、静的コード解析ツールSplintを使用します。関数の整理に有効な機能です。

• シミュレータ機能を持つ、Cソースおよびアセンブリソースレベルデバッガ
デバッガは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。ICD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、S1C17MCUコアのコアシミュレータ、およびS1C17各機種固有の周辺回路をシミュレートするES-Sim17を使用することができます。(ES-Sim17の対応機種は "10.11 組込みシステムシミュレータ(ES-Sim17)"を参照してください。)

• Windowsに対応した統合開発環境
Microsoft Windowsに対応した**GNU17 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 ソフトウェアツールの概要

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コンパイラ(xgcc.exe)

ANSI Cに準拠したCコンパイラで、GNU 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**を連続して呼び出し、Cソースファイルをコンパイルして**S1C17 Family**用のアセンブリソースファイルを生成します。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。**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**S1C17**コアのオブジェクト(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) デバッガ(gdb.exe)

ICD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) GNU17 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。

(8) Splint(Splint.exe)

静的コード解析ツールです。C言語の未使用関数の検出機能にて使用します。

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"readmeV*xx*.txt"(*xx*はバージョン番号を表します)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さまの責任においてご使用ください。

S5U1C17001C Manual

2 インストール

2 インストール

この章ではS1C17 Family 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

S1C17 Family 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。Pentium4 1.50GHz以上のCPUと1GB以上のRAMを搭載した機種を推奨します。

注: Itanium系のCPUは対応していません。

●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。また、画面の色は16ビット以上に設定してください。

●ハードディスク

S1C17 Family 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ICDが必要です。

●システムソフトウェアについて

Microsoft Windowsの日本語版と英語版に対応しています。

注: 64-bit OS では 32-bitアプリケーションとして動作します。

●その他

- ディスク上の"readmeV*xx*.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¥gnu17フォルダに置かれていますが、お客さま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、お客さま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、¥gnu17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。

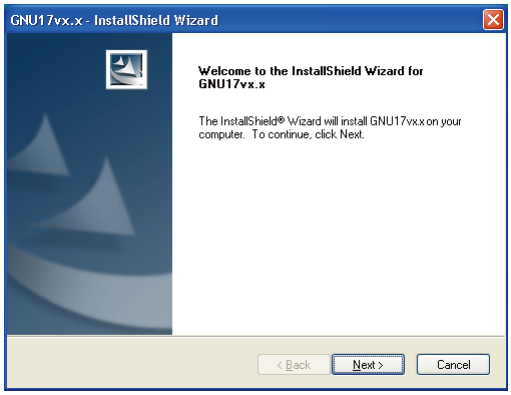
2.2 インストール方法

●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。

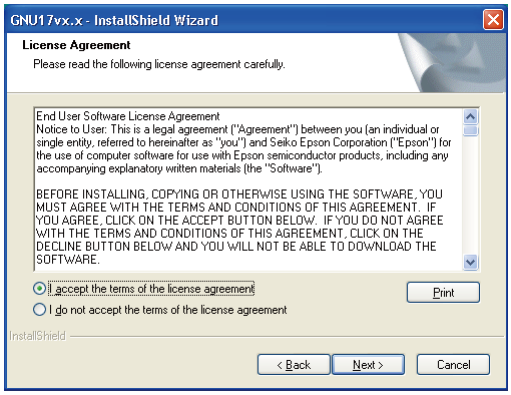
(2) S5U1C17001Cの圧縮ファイルをSEIKO EPSONのユーザサイトよりダウンロードして、任意のフォルダに解凍してください。

(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。



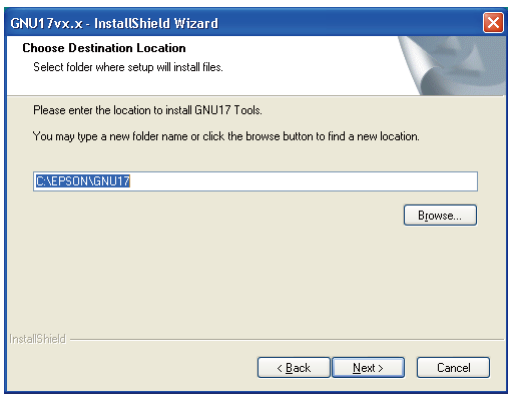
インストールウィザードのスタート画面が表示されます。

(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。

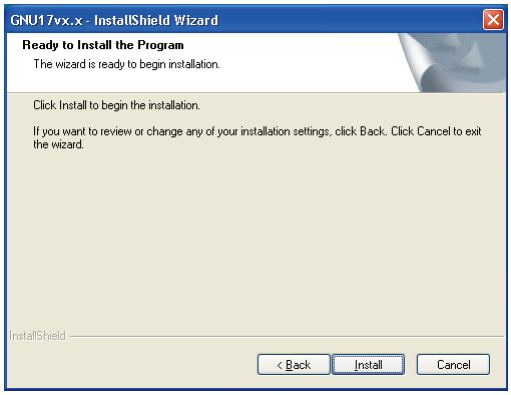
(5) ライセンス契約に同意される場合は"I accept the terms of the license agreement"を選択して、[Next >]ボタンをクリックしてください。
同意できない場合は[Cancel]ボタンをクリックしてインストーラを終了させてください。



gnu17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。

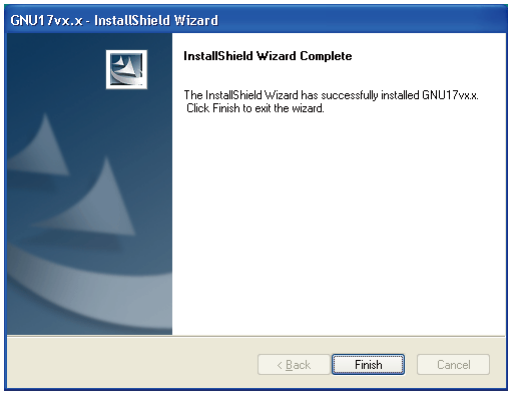
(6) 表示されたインストール先を確認します。
インストール先を変更する場合は、[Browse...]ボタンでフォルダ選択ダイアログボックスを表示させます。インストールするフォルダをダイアログボックス上のリストから選択するか、[Path]テキストボックスにフォルダへのパスを入力して[OK]ボタンをクリックしてください。

(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。

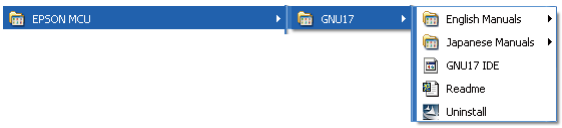


インストールが完了するとコンプリート画面が表示されます。

(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] > [GNU17]メニューが追加されます。



●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）
¥gnu17(gnu17ツールのルートDIR)

readmeVxx.txt	： ツールの内容説明(英語と日本語)
Copying.GNU	： GNU general public license
xgcc.exe	： GNUCコンパイラ(cpp.exe,cc1.exeを使います)
xgcc_filt.exe	： 漢字フィルター（xgcc.exeと同一のもの）
cpp.exe	： プリプロセッサ
cc1.exe	： コンパイラ本体
as.exe	： アセンブラ
ld.exe	： リンカ
objdump.exe	： オブジェクトファイルダンプツール
objcopy.exe	： objファイルを指定形式（Ｓレコードなど)に変換する
gdb.exe	： デバッガ
gdb.exe.manifext	： デバッガのIDEのマニフェストファイル(64ビットOS用)
gdbtk.ini	： gdb初期化ファイル
gnuProf.exe	： プロファイラ実行ファイル
gnuCvrg.exe	： カバレッジ実行ファイル
cygitcl30.dll	： gdb用dll
cygitk30.dll	： gdb用dll
cygtcl80.dll	： gdb用dll
cygtk80.dll	： gdb用dll
tix4180.dll	： gdb用dll
Icd17usb.dll	： gdb用dll
cygwin1.dll	： コンパイラ，binutils等のdll
cygiconv-2.dll	： cygwin1.dll用のdll
cygintl-3.dll	： cygwin1.dll用のdll
cygintl-8.dll	： cygwin1.dll用のdll
cyglsa.dll	： cygwin1.dll用のdll
cyglsa64.dll	： cygwin1.dll用のdll
gnustdin.dll	： シミュレーテッド入力用dll
reset.gdb	： [リセット]ボタンで実行するコマンドファイル
userdefine.gdb	： [ユーザコマンド]ボタンで実行するコマンドファイル
clkmdon.gdb	： [DCLK切替モードON]ボタンで実行するコマンドファイル
clkmdoff.gdb	： [DCLK切替モードOFF]ボタンで実行するコマンドファイル
savebreak.gdb	： ブレークポイントを保存するコマンドファイル
loadbreak.gdb	： ブレークポイントを再設定するコマンドファイル
kill.exe	： 強制ブレーク
c17_cmd_ref_eng.chm	： 旧GDBコマンドリファレンス（英語版）
c17_cmd_ref_jpn.chm	： 旧GDBコマンドリファレンス（日本語版）
make.exe	： make実行
ar.exe	： ライブラリファイルを作成・更新
cp.exe	： ファイルコピー
ps.exe	： プロセス情報取得用ツール
rm.exe	： ファイル削除
sed.exe	： ストリームエディタ
sh.exe	： Bourneシェル
cyggcc_s-1.dll	： sh.exe用のdll
cygreadline7.dll	： sh.exe用のdll
cygncursesw-10.dll	： sh.exe用のdll
ccap.exe	： コンソールキャプチャー
moto2ff.exe	：モトローラ形式ファイルのギャップをFFで埋めるツール
sconv32.exe	：モトローラ形式ファイルをS2レコード形式に変換するツール
DIFF.EXE	： ファイル比較ユーティリティ
vecChecker.exe	： コプロ用ベクタチェッカ
¥include	： ANSICライブラリのヘッダファイル
¥lib	
¥16bit	： 16bitアドレス用ライブラリ

libc.a	： ANSICライブラリ
libgcc.a	： エミュレーションライブラリ
libgccM.a	： エミュレーションライブラリ（乗算のコプロ命令対応版）
libgccMD.a	： エミュレーションライブラリ（乗算/除算/剰余演算のコプロ命令対応版）
libgccMD2.a	： エミュレーションライブラリ（乗算/除算/剰余算のコプロ2対応版）
libstdio.a	： シミュレーテッドIO用ライブラリ
¥24bit	：24bitアドレス用ライブラリ（ファイル構成は、¥16bitと同様）
¥eclipse	
eclipse.exe	： GNU17 IDEの実行ファイル
eclipse.exe.manifest	： IDEのマニフェストファイル（64ビットOS用）
eclipse.ini	： Eclipse設定ファイル
.eclipseproduct	： Eclipseのバージョン
epl-v10.html	： EPLライセンス
notice.html	： ソフトウェア合意書
gnu17_32_trans.ico	： gnu17アイコンファイル
artifacts.xml	： Eclipseアップデートマネージャ用ファイル
¥configuration	： 起動構成ファイル等
¥dropins	： 拡張プラグインフォルダ(空フォルダ)
¥features	： フィーチャー
¥jre	： Java仮想マシン
¥p2	： Eclipseアップデートマネージャフォルダ
¥plugins	： プラグイン
¥readme	： リリースノート
¥essim17	： 組み込みシステムシミュレータ
¥Splint	： 静的コード解析ツールSplintのフォルダ
¥utility	： ユーティリティ
¥tool	： 各種ミドルウェアなどのツール
¥dev	： PAファイル（提出用データ）作成ツール
¥sample	： サンプルファイル
¥mcu_model	： 機種別情報ファイル
¥doc	： マニュアル等ドキュメント

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

●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**GNU17 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、**GNU17 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。

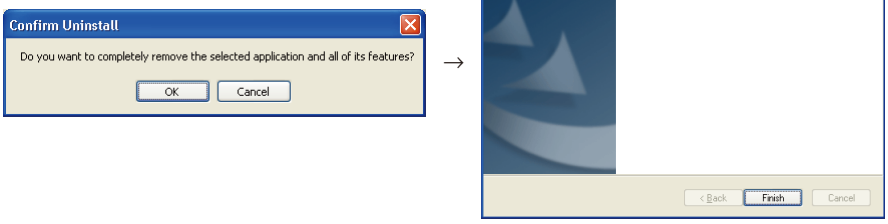
```
C:¥EPSON¥gnu17¥eclipse>eclipse.exe -clean
```

●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。
- ICDとの通信に使用する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**GNU17 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。

●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] > [GNU17]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¥gnu17¥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]>[GNU17]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コンパイラツールはFree Software Foundation, Inc.のGNU 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

GNU17 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¥gnu17¥eclipseフォルダ内のepl-v10.htmlをご覧ください。

S5U1C17001C Manual

3 ソフトウェア開発手順

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。

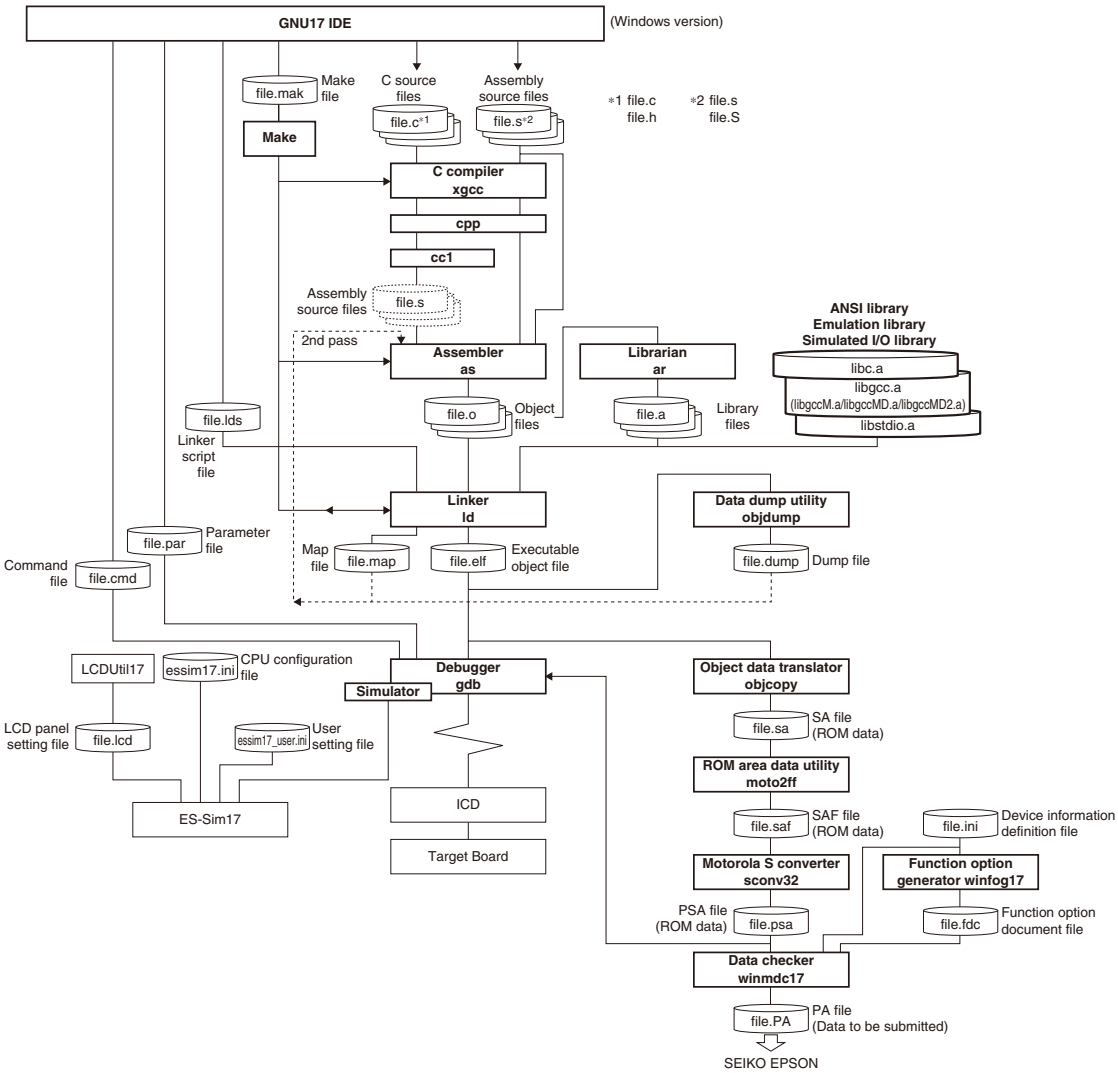


図3.1.1 ソフトウェア開発フロー

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**GNU17 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と、それをSレコード形式に変換したPSAファイル(ROMデータ)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ソースの場合)**
  ソースファイルをCコンパイラ**xgcc**でコンパイルし、リンカ**ld**に入力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**に入力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。
- **Sレコード変換**
  objcopy/moto2ff/sconv32を起動し、elf形式のオブジェクトファイルから、Sレコード形式のPSAファイルを出力します。
  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、SAファイル(ROMデータ)を作成します。
  さらに、SAファイルをmoto2ffおよびsconv32により未使用領域を0xff詰めしたモトローラS2ファイルに変換します。
  これにより生成されたPSAファイルを使用して実機上での最終動作確認を行ってください。

(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およびSレコード形式のPSA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。ICD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S1C17コア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。
デバッガ用の設定と起動は**IDE**から行えます。

(5) PAファイル(提出用データ)の作成

セイコーエプソン工場にてターゲットCPUの内蔵ROM又は内蔵FLASHメモリにユーザプログラムを書き込むサービスをご利用の場合、PAファイル(提出用データ)を作成し、セイコーエプソンへ提出していただく必要があります。動作確認を終えたPSAファイルと、winfog17で作成したFDCファイル(ファンクションオプションドキュメント)を、winmdc17を使用して1つのPAファイルにパックします。このPAファイルをセイコーエプソンへ提出してください。
winfog17とwinmdc17の起動はIDEから行えます。

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S1C17 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。

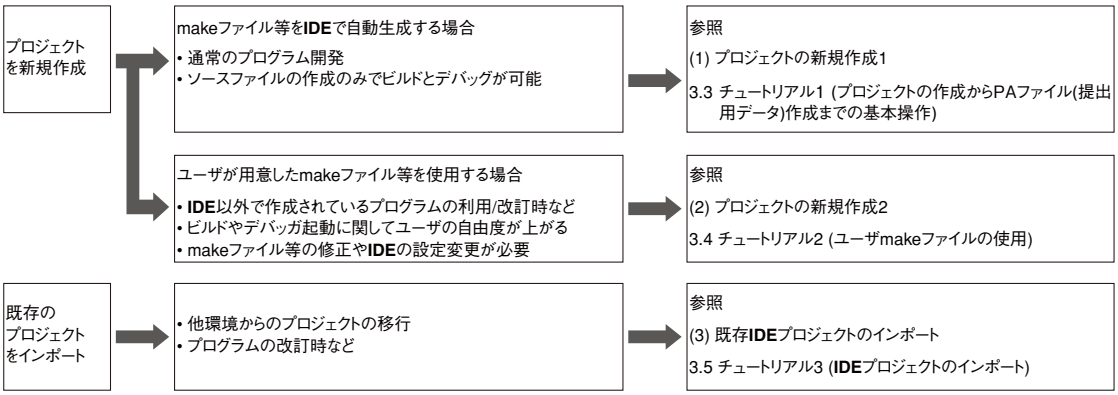


図3.2.1 **IDEによる開発**

(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ガ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。

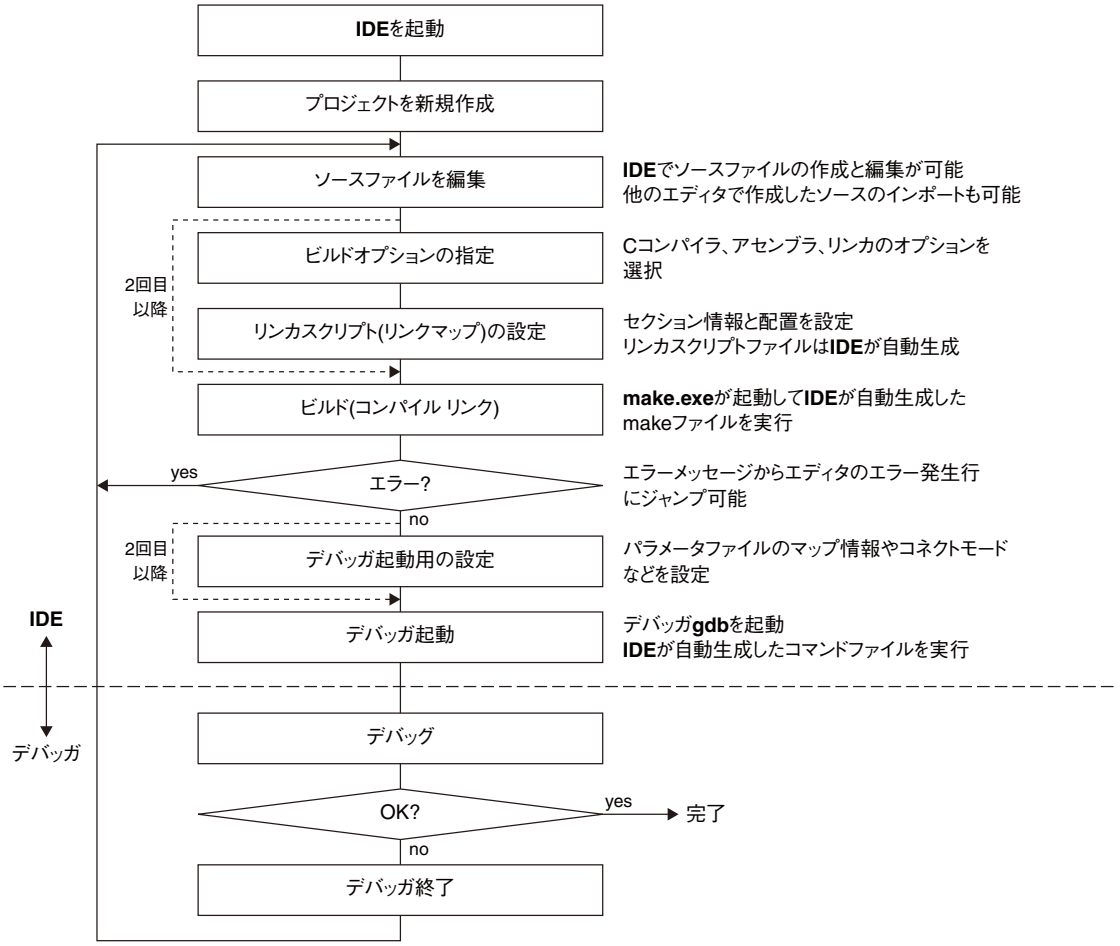


図3.2.2 操作の流れ(makeファイル等をIDEで自動生成)

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(プロジェクトの作成からPAファイル(提出用データ)作成までの基本操作)"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

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は、**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。
基本的な操作の流れは次のとおりです。

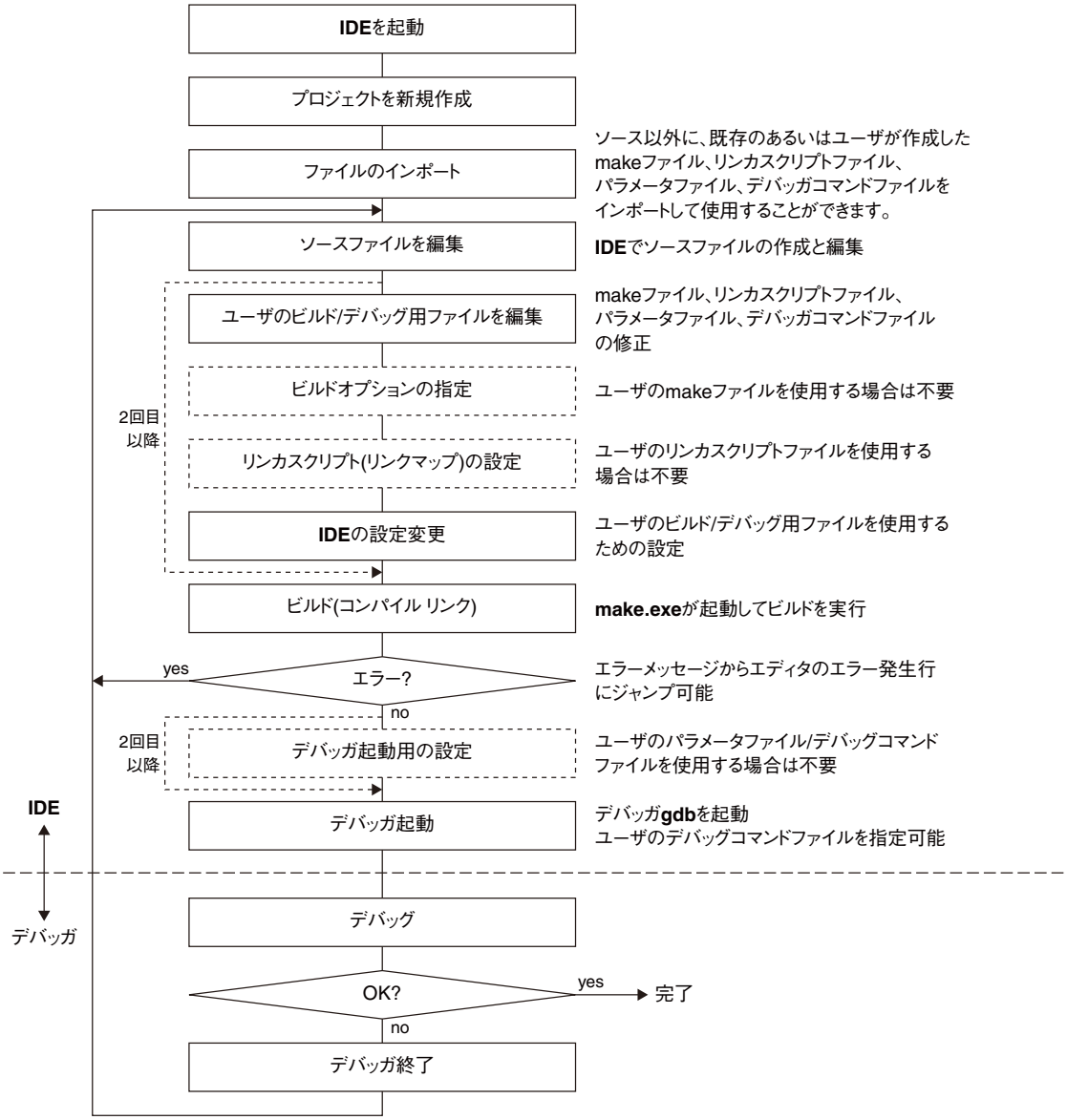


図3.2.3 操作の流れ(ユーザのmakeファイル等を使用)

"3.4 チュートリアル2(ユーザmakeファイルの使用)"では、ユーザの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。

ユーザの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ことが必要です。
基本的な操作の流れは次のとおりです。

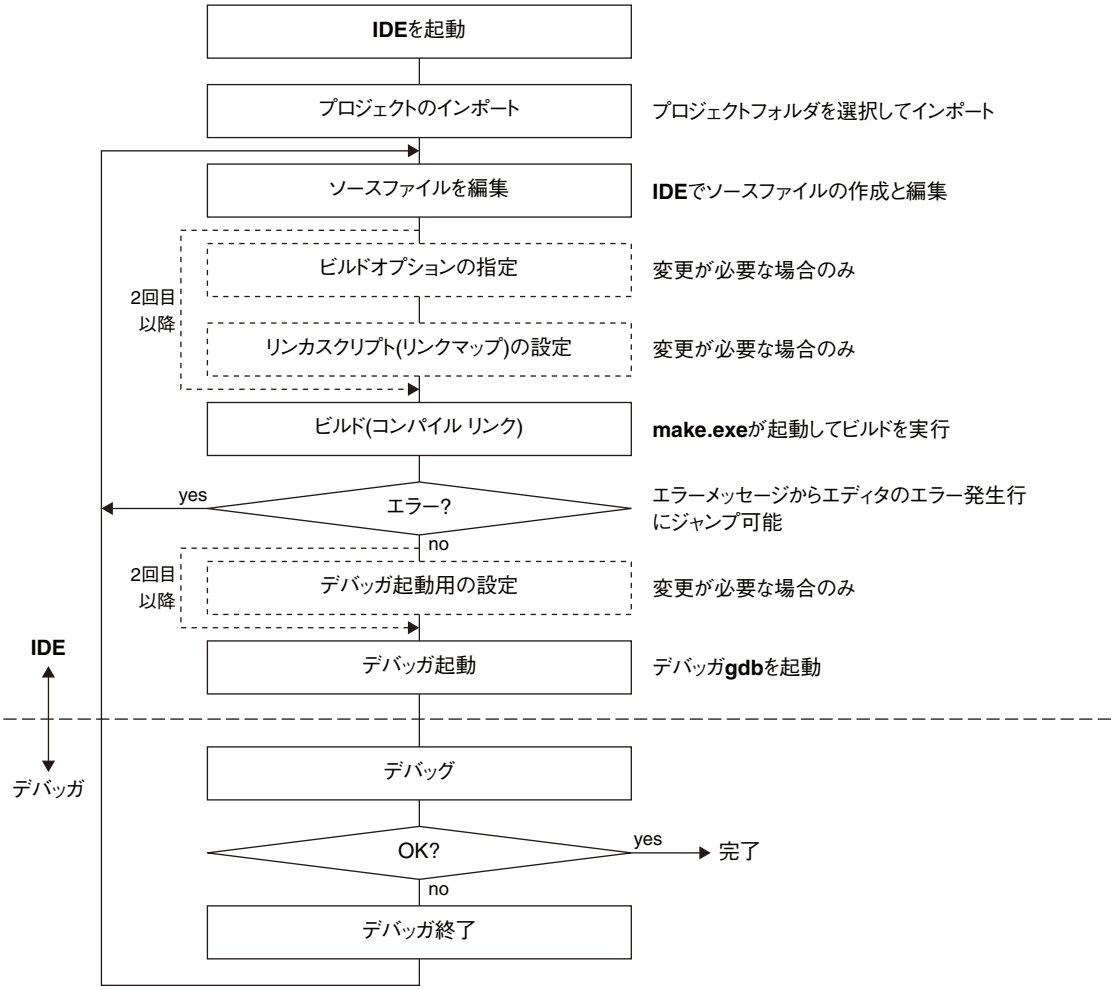


図3.2.4 操作の流れ(IDEプロジェクトのインポート)

"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(プロジェクトの作成からPAファイル(提出用データ)作成までの基本操作)

ここではチュートリアルとして、**IDE**の起動からデバッグ及びPAファイル作成までの基本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¥gnu17¥sample¥S1C17common¥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。

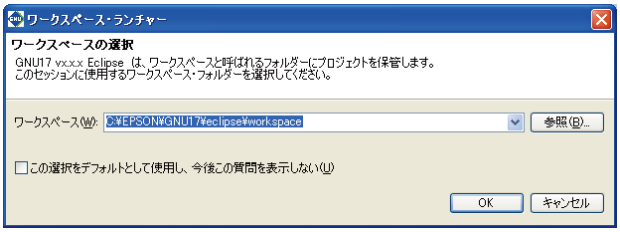
`boot.s`	アセンブラソースファイル
`main.c`	Cソースファイル

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、デバッガ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¥gnu17¥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] > [GNU17] > [GNU17 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。

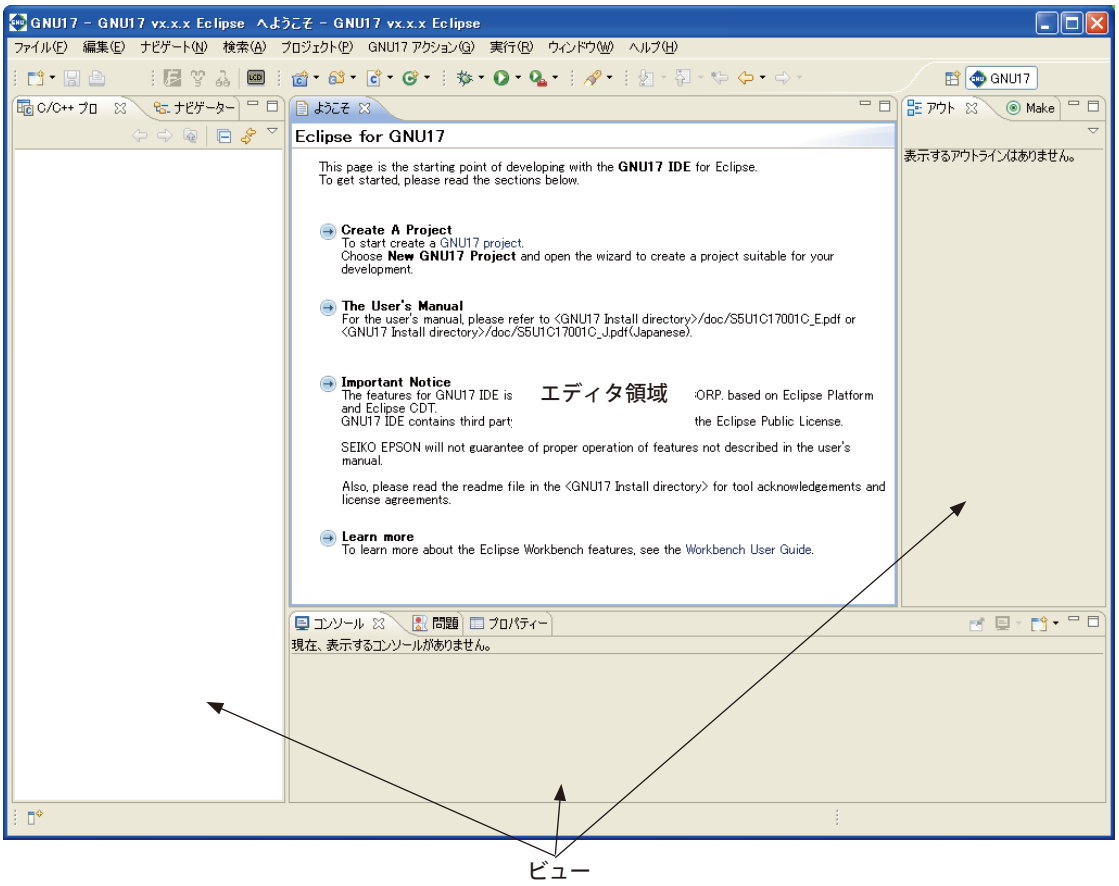


任意のディレクトリを選択または新規作成してワークスペースに設定可能ですが、このチュートリアルでは表示されたデフォルトのワークスペースディレクトリを使用します。

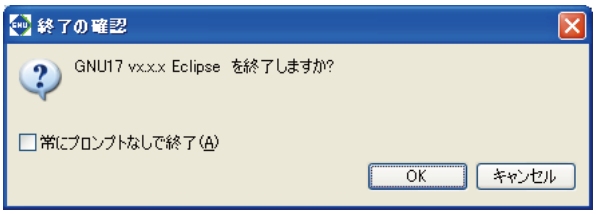
※ ワークスペースディレクトリには、プロジェクトディレクトリ(.projectファイルが含まれるディレクトリ)を指定しないでください。プロジェクトインポート(プロジェクトをワークスペースにコピーがONのとき)に失敗することがあります。

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。



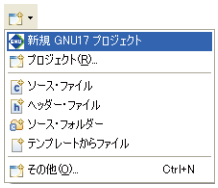
チュートリアルを途中で終了させるには、**IDE**の[ファイル]メニューから[終了]を選択してください。ウィンドウの(閉じる)ボタンでも終了可能です。ただし、この場合は次のダイアログボックスが表示されますので、終了する場合は[OK]ボタンを、終了を取り消す場合は[キャンセル]ボタンをクリックします。



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`、`.gnu17project`、`.project`)が生成されます。

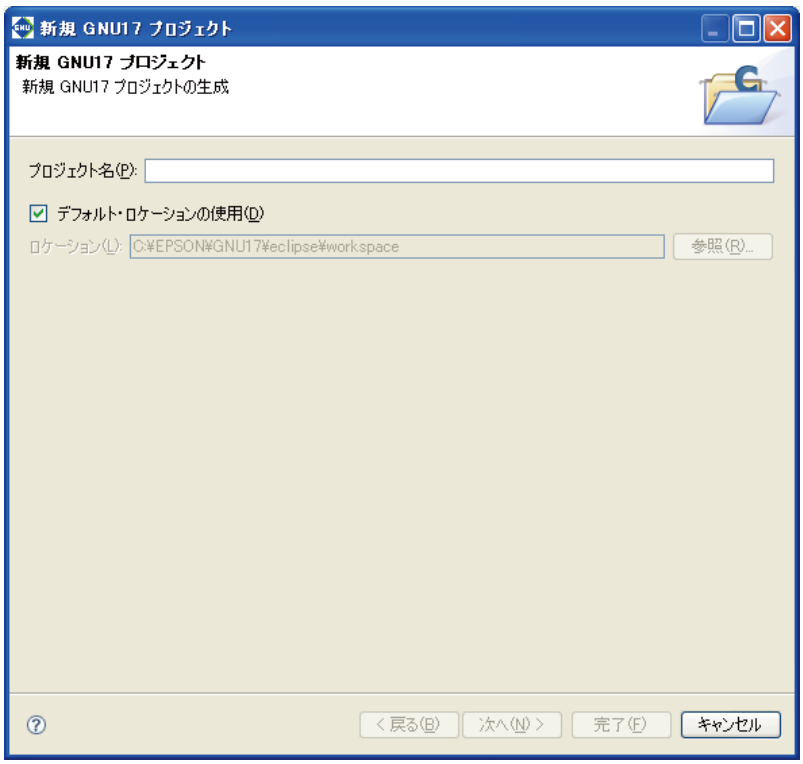
●プロジェクトを新規作成するには



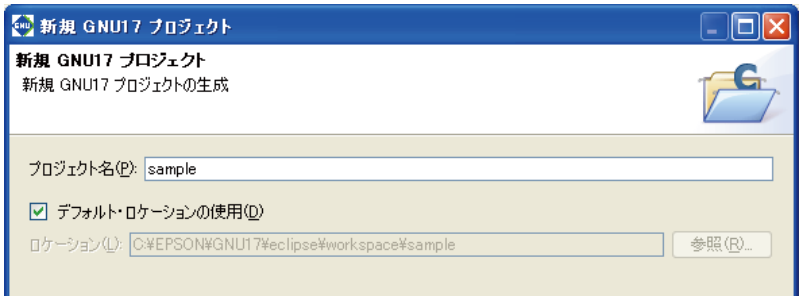
操作3: ツールバーの[新規]プルダウンメニューから[新規 GNU17 プロジェクト]を選択します。

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 GNU17 プロジェクト]が選択できます。

[新規 GNU17 プロジェクト]ウィザードが起動します。



●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、メモリモデル、ベクタテーブルファイルの選択画面に移行します。

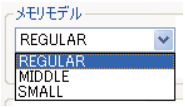
●生成プログラムの指定

操作6: [生成プログラム]ラジオボタンから生成するプログラムを選択します。ここでは"アプリケーション(.elf/.psa)"を選択してください。

●ターゲットCPUの指定

操作7: [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を選択します。ここでは"S1C17"を選択してください。

●メモリモデルの選択



操作8: [メモリモデル]コンボボックスからプロセッサがサポートしているメモリモデルを選択します。ここでは"REGULAR"を選択してください。
REGULAR: 24ビットアドレス空間(16MB)
MIDDLE: 20ビットアドレス空間(1MB)
SMALL: 16ビットアドレス空間(64KB)

●コプロセッサライブラリ選択

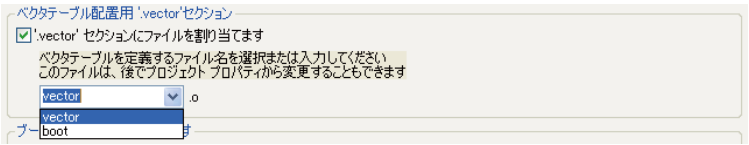
ターゲットCPUに選択した機種によっては、コプロセッサライブラリをリンクするかどうか、およびリンクするライブラリの種類を選択できます。

操作9: [コプロセッサ用ライブラリを使用する]チェックボックスをON/OFFします。
ON： プロジェクト作成時にコプロセッサライブラリlibgccMD2.a(コプロ2乗除算用)、libgccMD.a(乗除算用)もしくはlibgccM.a(乗算用)をリンクする設定を追加します。ONの場合、[コプロセッサ用ライブラリの種類]コンボボックスからリンクするライブラリの種類を選択します(機種によっては選択肢が1つしかないものもあります)。
OFF：プロジェクト作成時に通常のエミュレーションライブラリlibgcc.aをリンクする設定を追加します。

●ベクタテーブルファイルの指定

IDEではトラップベクタを確実にトラップ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カ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。

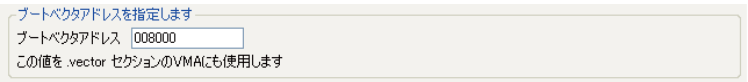
操作10: プルダウンリストから`boot.o`を選択します。



`.vector`セクションの詳細については、"5.7.9 リンカスクリプトの編集"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。
デフォルト設定のブートベクタアドレス値は"008000"です。ここに設定した値は、**IDE**が自動生成するデバッガ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、およびリンカスクリプト内の.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。

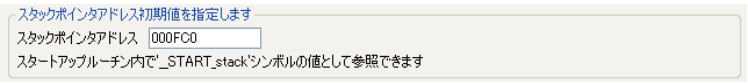


●スタックポインタアドレスの指定

[スタックポインタアドレス]フィールドでは、スタックポインタアドレスを指定します。
デフォルト設定のスタックポインタアドレス値は"000FC0"です。
ここに設定した値は、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。

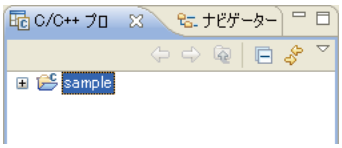
例） ブートルーチン内で以下のように記述できます。

```
boot:
xld.a %sp, __START_stack
```



操作11: [完了]ボタンをクリックします。

[新規 GNU17 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、メモリモデル、コプロライブラリの選択、ベクターテーブルファイル、スタックポインタアドレスの指定は、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、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

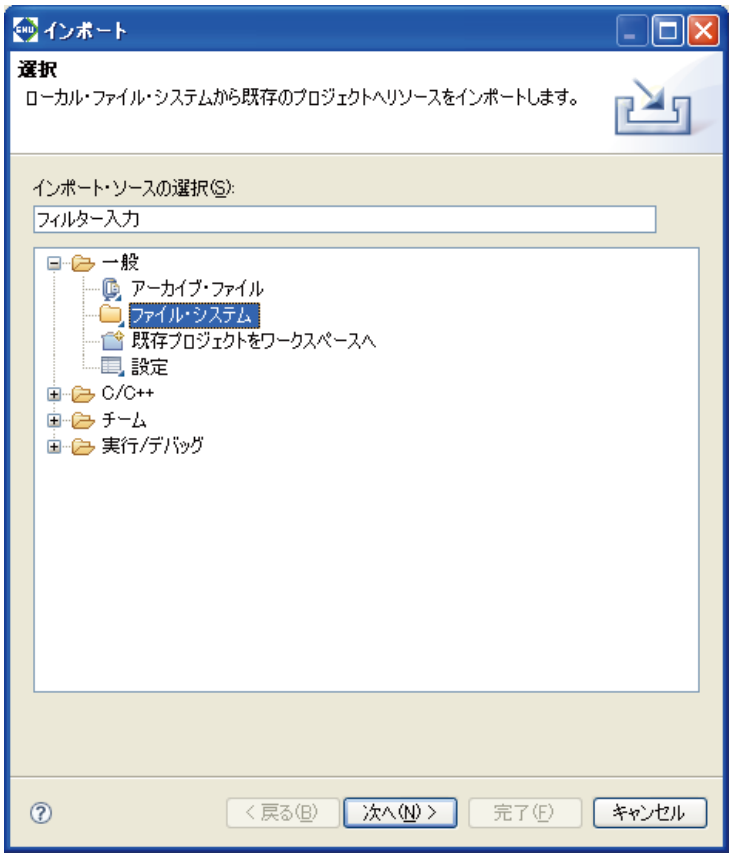
ソースファイルはIDEのエディタまたは汎用のエディタで作成します。また、S1C17 Family用アプリケーションの既存のソースファイルを使用することもできます。
このチュートリアルではサンプルとして用意されたソースファイルを使用します。
IDEによるソースの新規作成方法については"5.5 エディタとソースファイルの編集"を参照してください。

●ソースファイルの追加

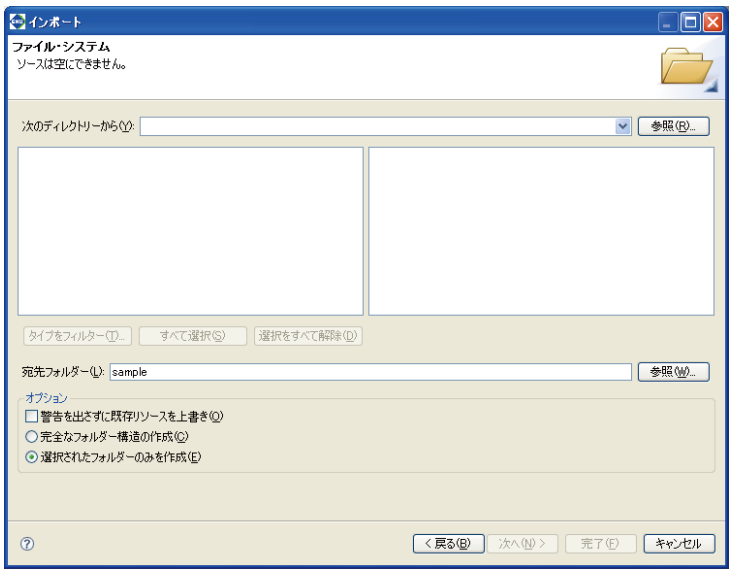
サンプルとして用意されているソースファイルをプロジェクトに読み込みます。

操作12: [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。

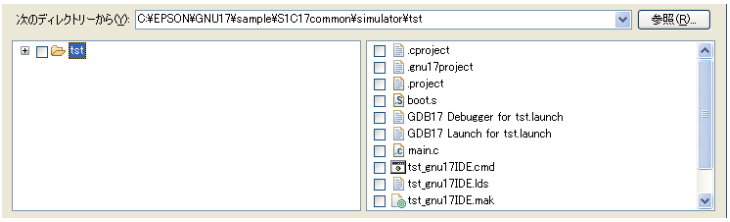


操作13: 表示されている一覧の中から[一般] > [ファイル・システム]を選択し、[次へ>]ボタンをクリックします。



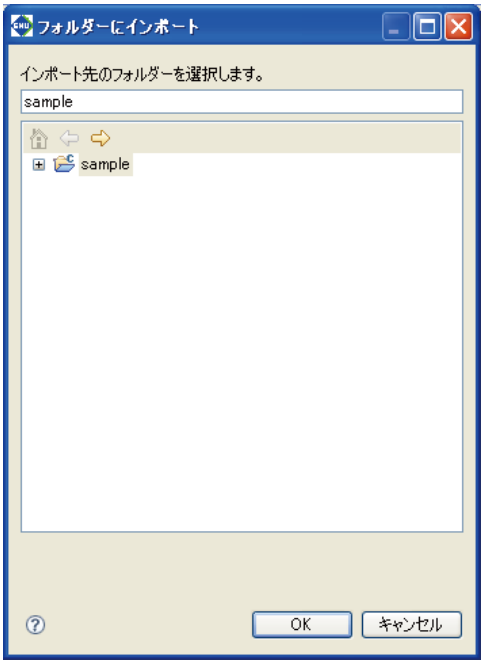
操作14: [次のディレクトリーから:]の[参照...]ボタンをクリックします。[ディレクトリーからインポート]ダイアログボックスが表示されますので、**IDE**をインストールしたドライブ(C)から`¥EPSON¥gnu17¥sample¥S1C17common¥simulator¥tst`ディレクトリを選択し、[OK]ボタンをクリックします。

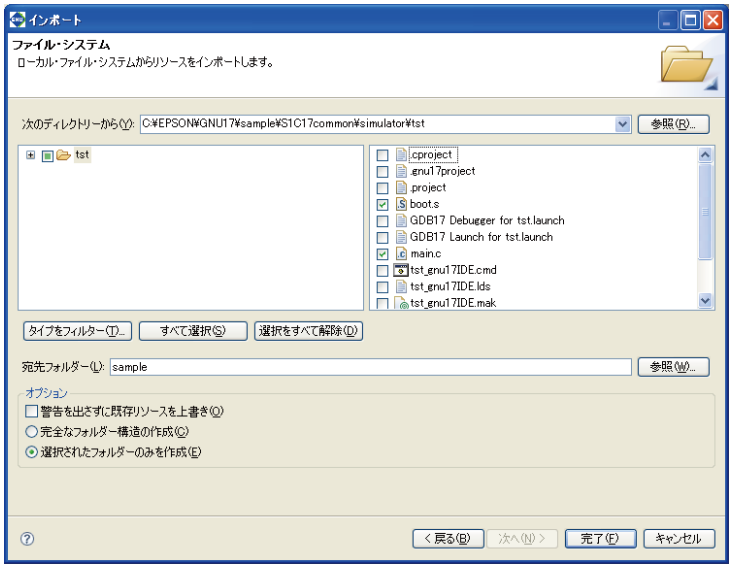
左のリストボックスに選択したディレクトリが、右のリストボックスにディレクトリ内のファイルの一覧が表示されます。



操作15: 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16: [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]ボタンをクリックします。





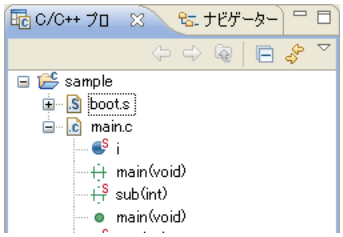
操作17: 上記のとおり表示されていることを確認後、[完了]ボタンをクリックします。

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18: [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19: [C/C++ プロジェクト]ビューの"main.c"の[+]をクリックします。

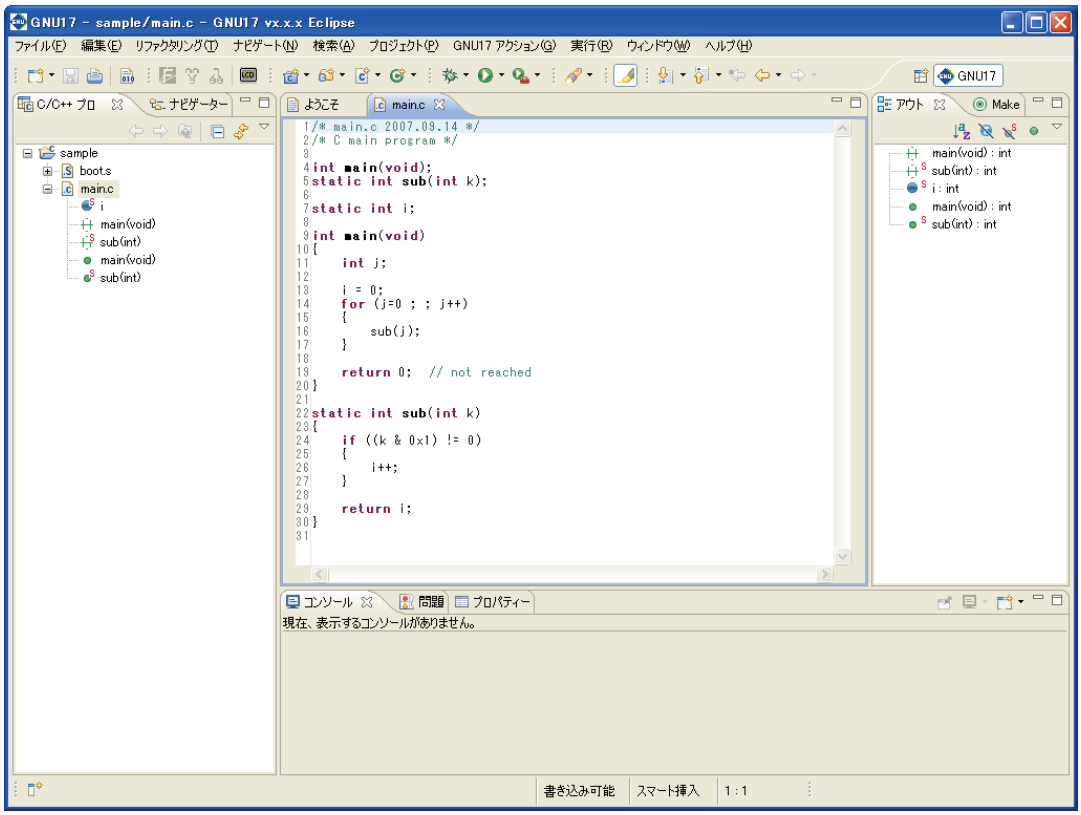


Cソースの場合、そのファイルに定義されているグローバル変数と関数が表示されます。

●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。

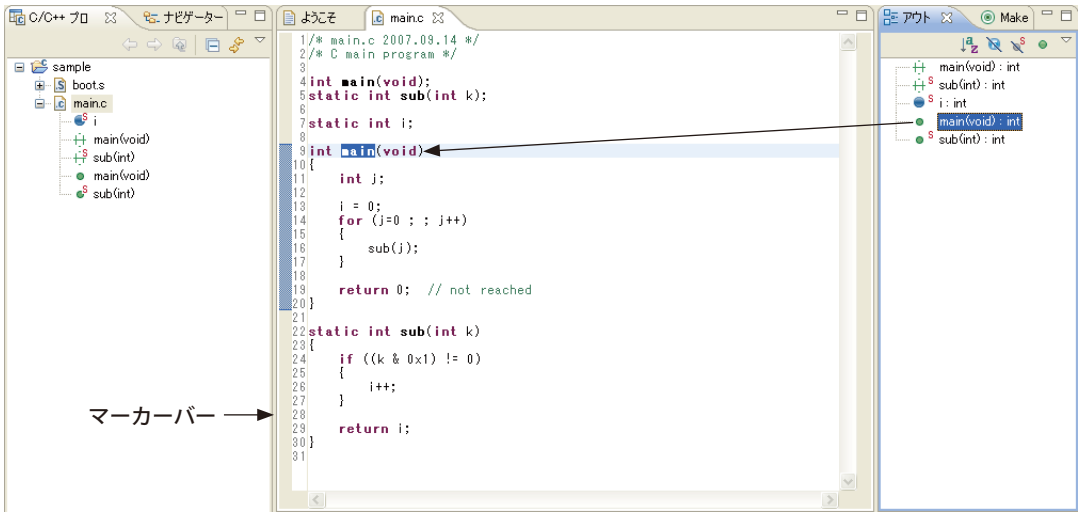
操作20: [C/C++ プロジェクト]ビューの"main.c"をダブルクリックします。



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。

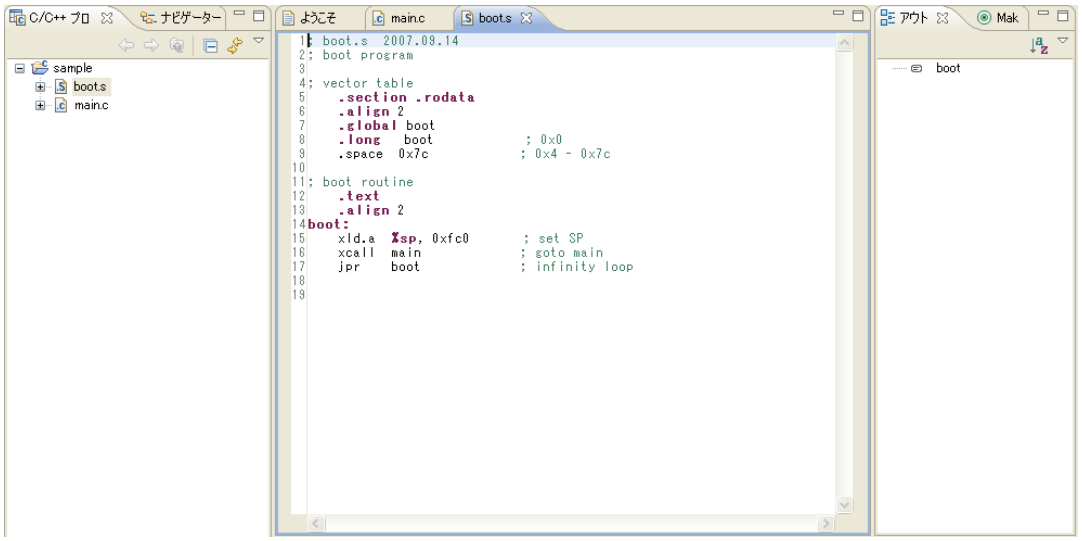
Cソースを表示させた場合、Cの予約語、コメント、文字列が色付きで強調表示されます。

操作21: [アウトライン]ビューの"main"をクリックしてください。



エディタはmain()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。

操作22: [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23: それぞれのソースのエディタタブにある(Close)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カスクリプトの編集

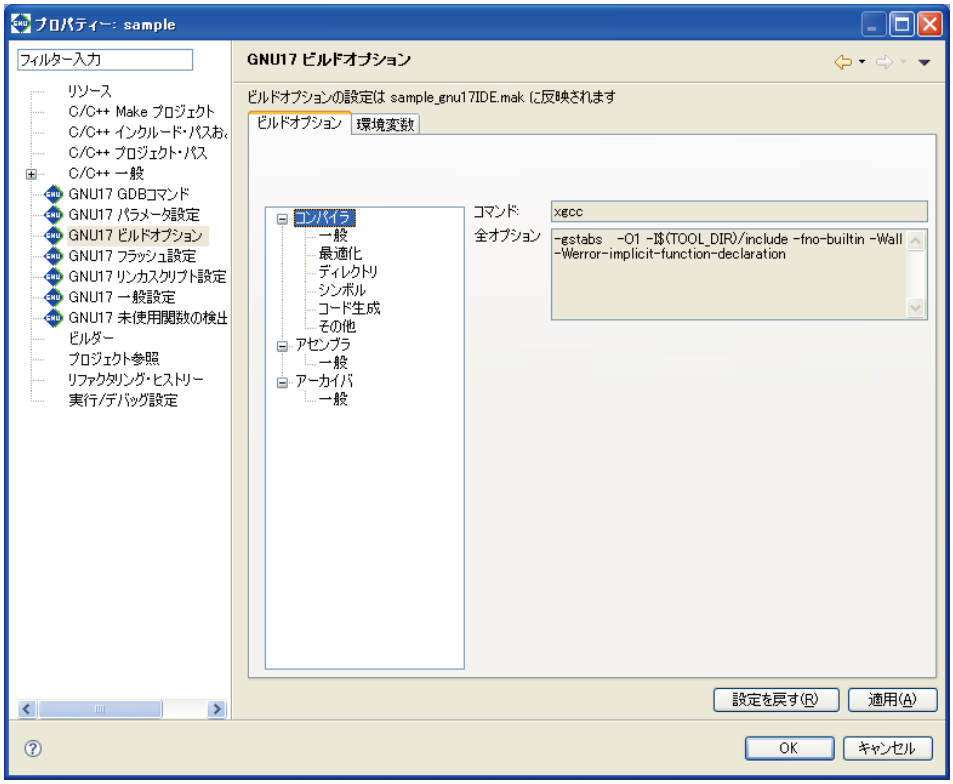
プロジェクトのビルド(実行形式オブジェクトファイルの生成)は**make.exe**によりコンパイラ、アセンブラ、リンカを起動して行います。makeに必要なmakeファイルは**IDE**で自動的に生成されますが、その中に記述するビルドオプション(コンパイラ、アセンブラ、リンカの起動オプション)はあらかじめ設定しておかなければなりません。また、リンクに必要なリンカスクリプトファイルの内容もビルド前に設定しておく必要があります。
以下、それらの設定方法を簡単に説明します。

●ビルドオプションの設定

操作24: [プロジェクト]メニューから[プロパティー]を選択します。[プロパティー]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。

[プロパティー]ダイアログボックスが表示されます。

操作25: 左側にあるプロパティのリストから[GNU17 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。



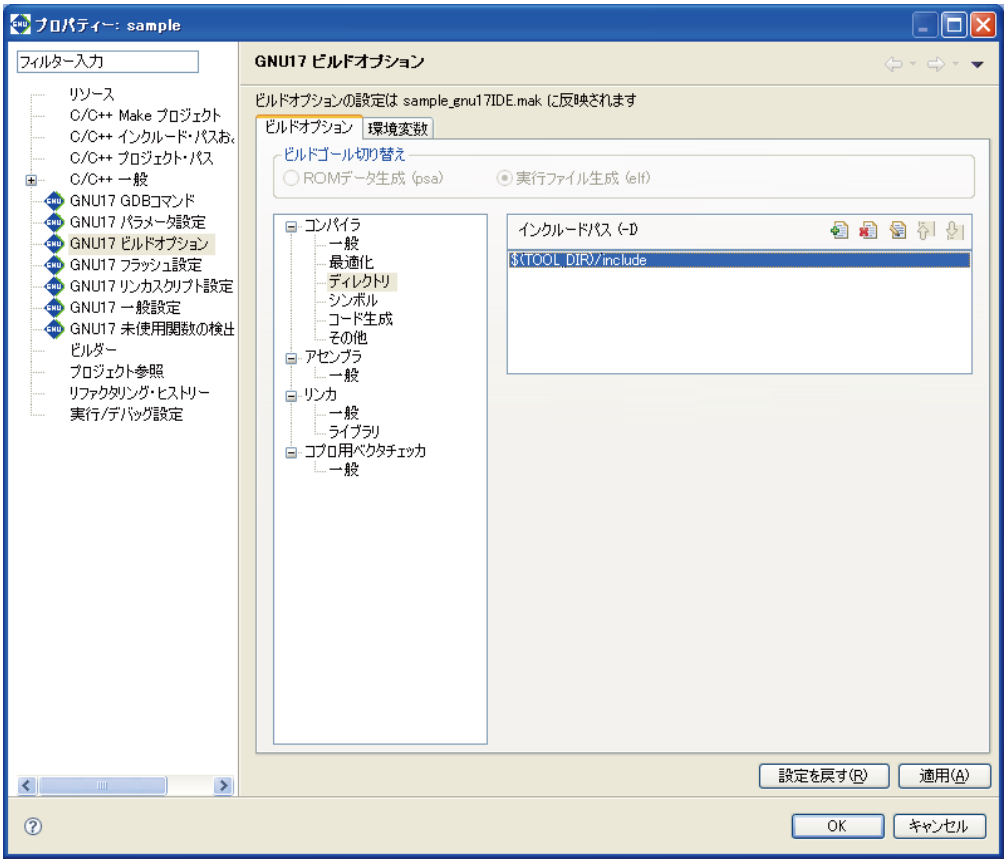
ここで、コンパイラ、アセンブラ、リンカ、ベクタチェッカのコマンドラインオプションを設定できます。
ツリー表示されたツール名(コンパイラ、アセンブラ、リンカ、コプロ用ベクタチェッカ)をクリックして選択すると、現在選択されているオプションが[全オプション]に表示されます。ツリー表示からオプションの分類をクリックして選択すると、個々のオプションを設定できる状態になります。
現在表示されているオプションは、プロジェクトを新規作成した際にデフォルトとして設定された内容です。
オプションの内容については各ツールの章を、オプション選択画面の詳細については"5.7 プログラムのビルド"を参照してください。
こ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26～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26: [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。

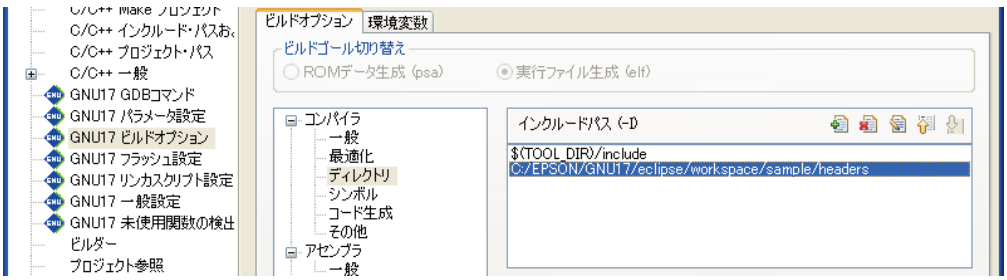
Cコンパイラの-I(インクルードパス指定)オプションを設定するページが表示されます。



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27: [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28: [適用]ボタンで変更内容を確定します。

[GNU17 ビルドオプション]の設定を変更して[適用]ボタンまたは[OK]ボタンをクリックすると、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)を行うためのダイアログボックスが表示されます。ここでは[キャンセル]ボタンをクリックしてください。

以上の設定により、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

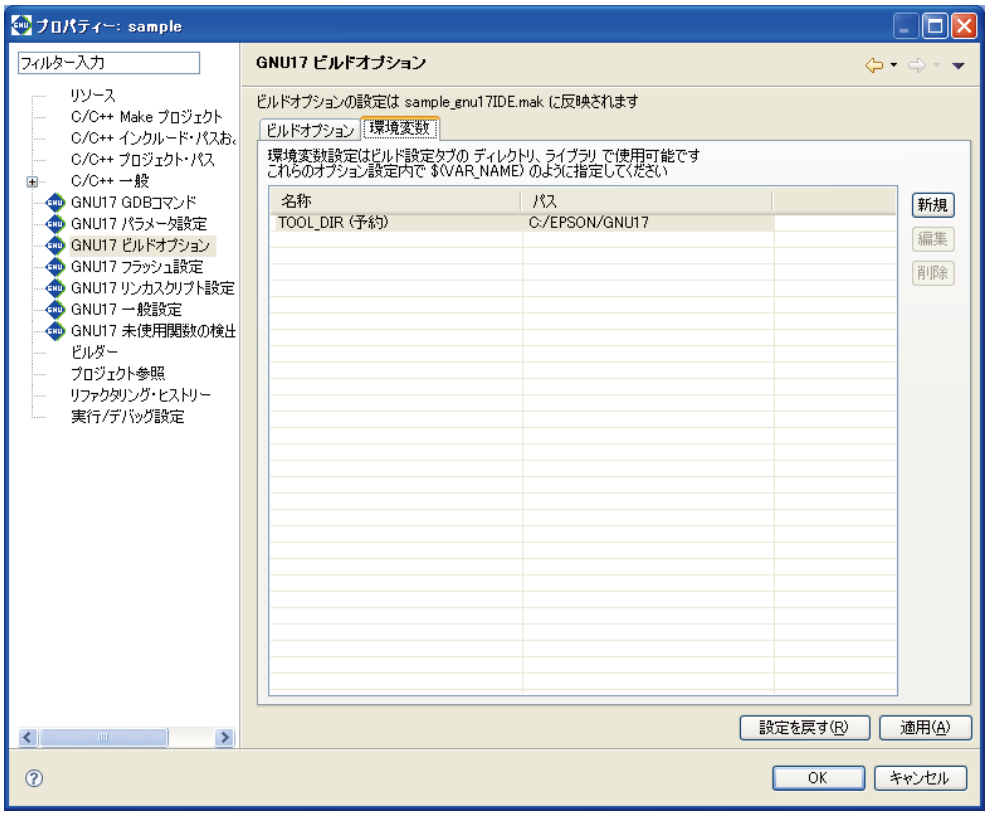
デフォルト設定で、[インクルードパス (-I)]には"$(TOOL_DIR)/include"が表示されています。$(*環境変数*)は、makeファイル(ビルド時に生成)に記述されるマクロです。TOOL_DIRは、gnu17ツールをインストールしたディレクトリへのパスが定義されている環境変数で、定義内容は[環境変数]タブのページで確認できます。

例: gnu17ツールをc:¥EPSON¥gnu17ディレクトリにインストールした場合

```
TOOL_DIR = c:/EPSON/gnu17
```

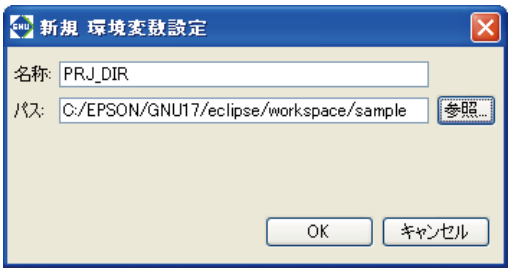
make.exeの実行時は、マクロが()内の環境変数の内容に置き換えられますので、-I$(TOOL_DIR)/includeは-Ic:/EPSON/gnu17/includeの指定となります。

[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29: [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。



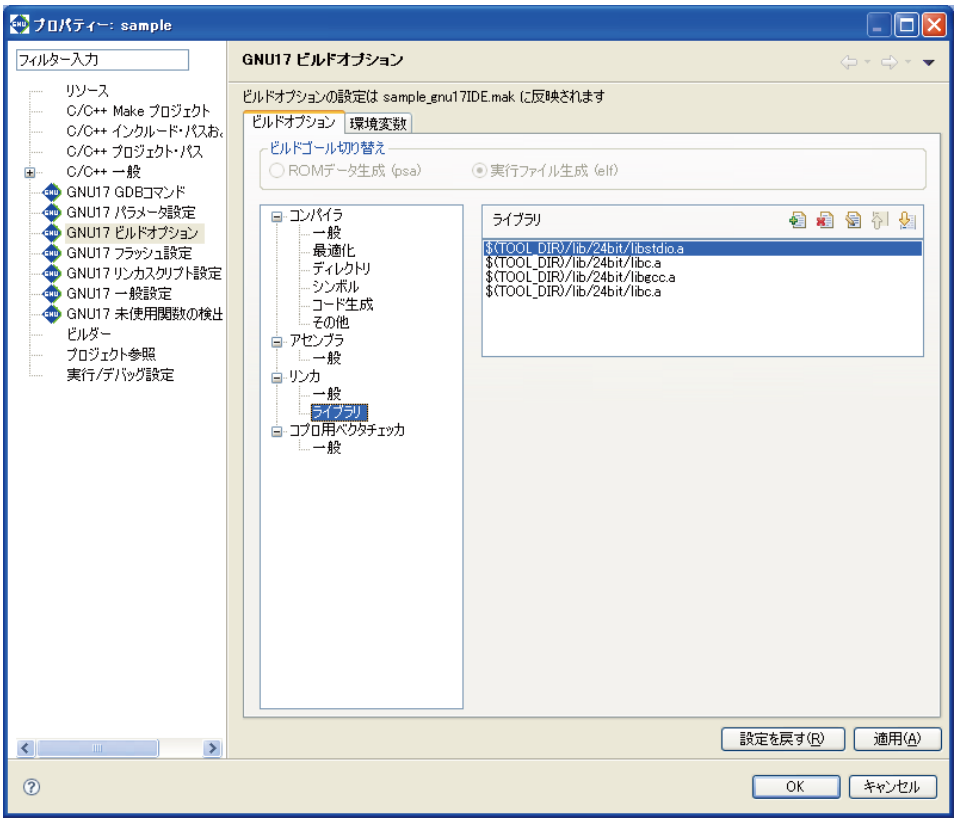
ここで定義した環境変数は、ビルドオプションのインクルードパスおよびライブラリファイルのパスの指定に使用可能です。使用する場合は、上記例のように$ (*環境変数*) のマクロとして入力してください。

●ライブラリファイルの追加

以下、操作30～操作32は参考例ですので、特に操作する必要はありません。

操作30: [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31: [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[フォルダの参照]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32: [適用]ボタンで変更内容を確定します。
[GNU17 ビルドオプション]の設定を変更して[適用]ボタンまたは[OK]ボタンをクリックすると、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)を行うためのダイアログボックスが表示されます。ここでは[キャンセル]ボタンをクリックしてください。

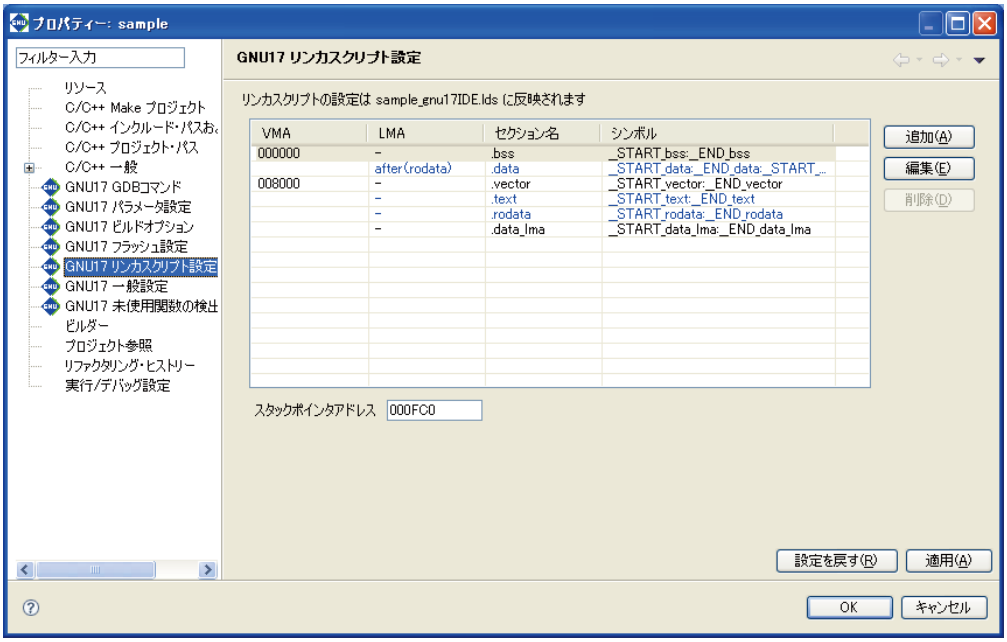
以上の設定により、ここで指定したライブラリもリンクの対象になります。

●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、リンカ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。詳細は、"3.8 セクションとリンク"を参照してください。

操作33: [プロパティー]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[プロパティー]を選択して再度開いてください。

操作34: プロパティのリストから[GNU17 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、.data、.vector、.text、.rodata、.data_lmaの6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)
.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)
.data_lma	.dataの仮想セクション(.dataにある変数の初期値。通常は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(.rodata)"は()内のセクション、この場合は.rodata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。

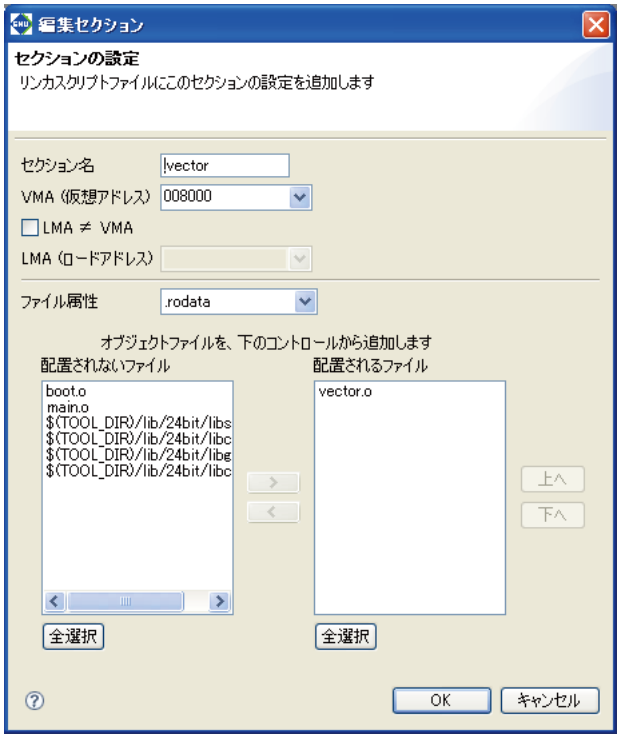
セクション情報は、.vectorのみ黒色、その他のセクションは青色で表示されています。
青色はデフォルトで定義済みの標準セクションを表し、黒色はそれ以外の新たに定義したユーザセクションを表します。
ユーザセクションは、セクション名、セクションの基本属性、配置アドレス、配置するオブジェクトを任意に編集可能です。標準セクションは配置アドレスのみ指定可能で、ユーザセクションに配置された以外のオブジェクトが自動的に配置されるようになっています。

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てみます。

操作35: 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36: [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37: [プロパティー]ダイアログボックスの[OK]ボタンをクリックして、リンカスクリプトの編集を終了します。

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

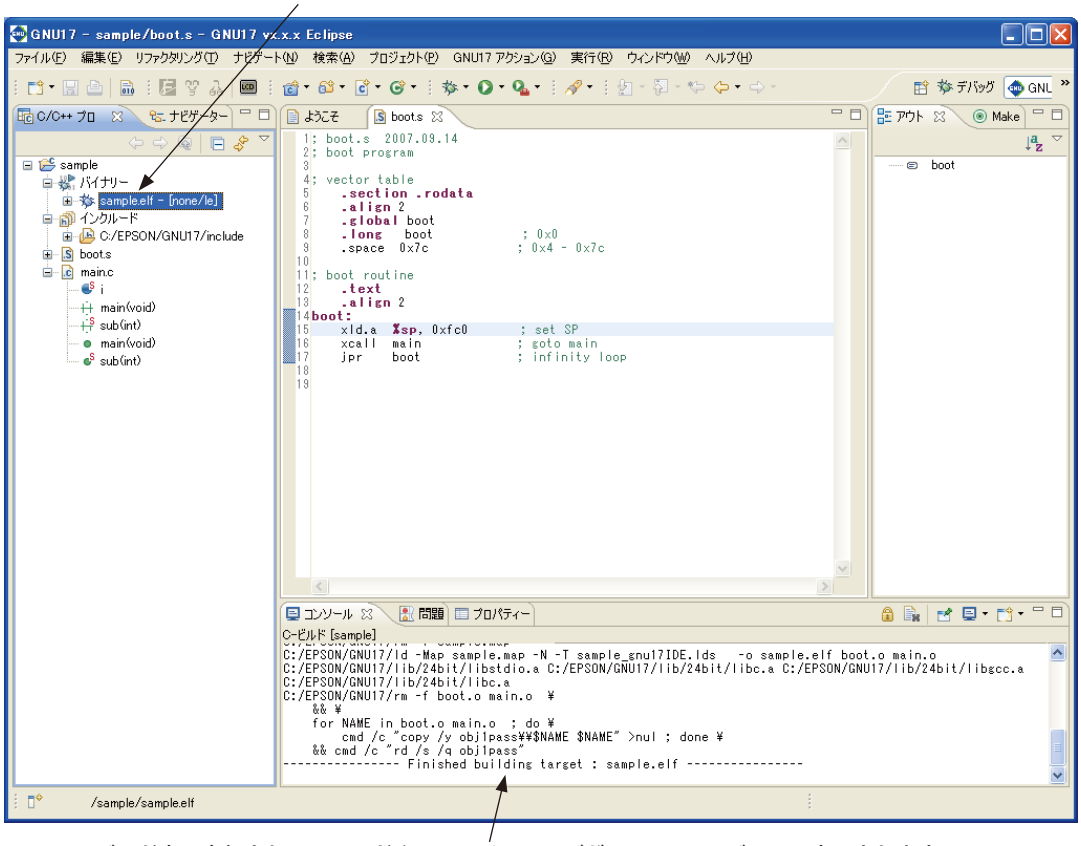
前節までの作業が終わると、プログラムのビルド(コンパイル、アセンブル、リンク)が行えます。

●ビルドを実行するには

操作38: [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39: [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。

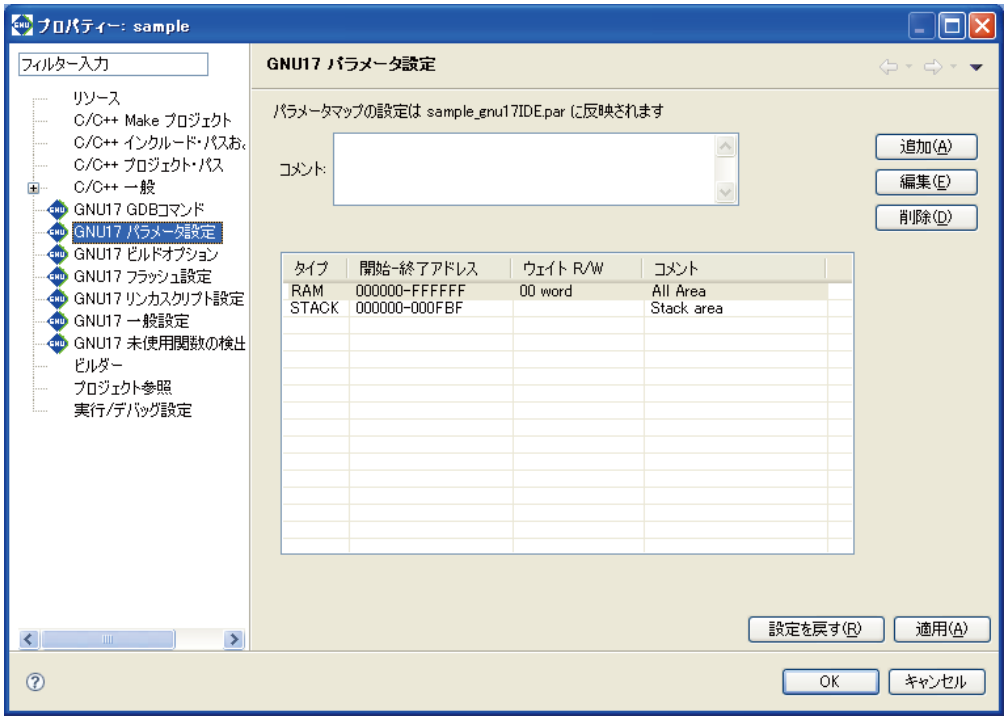
3.3.6 デバッグ

デバッグを行う前にデバッガ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、デバッガ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、必ずデバッガに読み込ませてください。
また、デバッガ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40: [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[プロパティー]を選択します。

操作41: プロパティのリストから[GNU17 パラメータ設定]を選択(クリック)します。

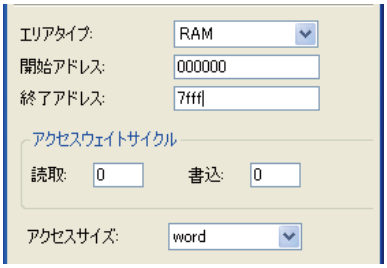


デフォルト設定のエリア情報が2個表示されます。
RAMの情報は、0x0～0xffffff(16Mバイト)をRAM領域として使用することを定義しています。"00 word"とあるのは、読み出しに0サイクル、書き込みに0サイクルのウェイトを挿入し、32ビットでアクセスするデバイスであることを示します(このアクセス条件の設定はシミュレータモードでのみ有効です)。
この他に、RAM内のスタック領域が定義されています。
これは、S1C17コア搭載のS1C17マイクロコンピュータの基本構成です。
これ以外のメモリや外部デバイスを使用する場合は、[追加]ボタンをクリックして追加するエリアの設定を行います。
サンプルプログラムは、特にデフォルトメモリ構成の変更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。以下、操作41～操作47は参考例ですので、特に操作する必要はありません。
ここでは操作の参考として、ROM領域(0x8000 ～ 0x17fff)を追加してみます。ただし、STACKの設定以外は他の領域との重複は認められませんので、先にRAM領域を0x0～0x7fffに変更します。

操作42: リストボックス内のRAM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43: [終了アドレス:]を7ffffに変更します。

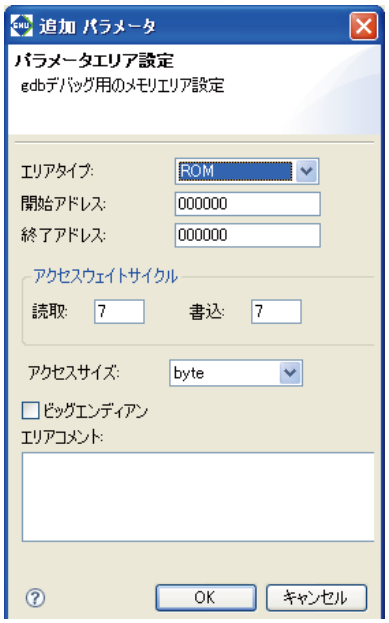


操作44: [OK]ボタンをクリックします。

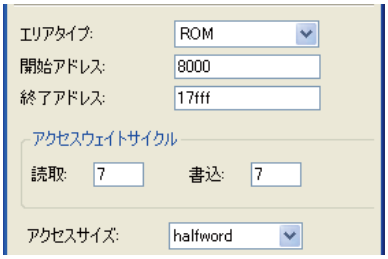
RAM領域の表示が"000000-007FFF"に変更されました。

操作45: [追加]ボタンをクリックします。

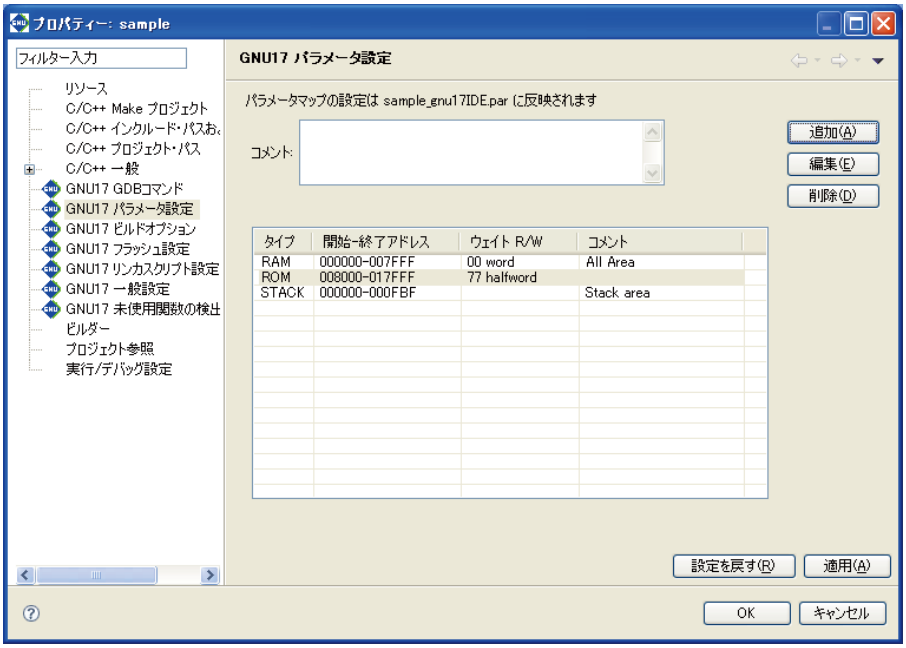
[追加 パラメータ]ダイアログボックスが表示されます。



操作46: [開始アドレス:]テキストボックスに8000を、[終了アドレス:]テキストボックスに17fffを入力します。
[アクセスサイズ:]は"halfword"(16ビット)を選択します。



操作47: [OK]ボタンをクリックします。



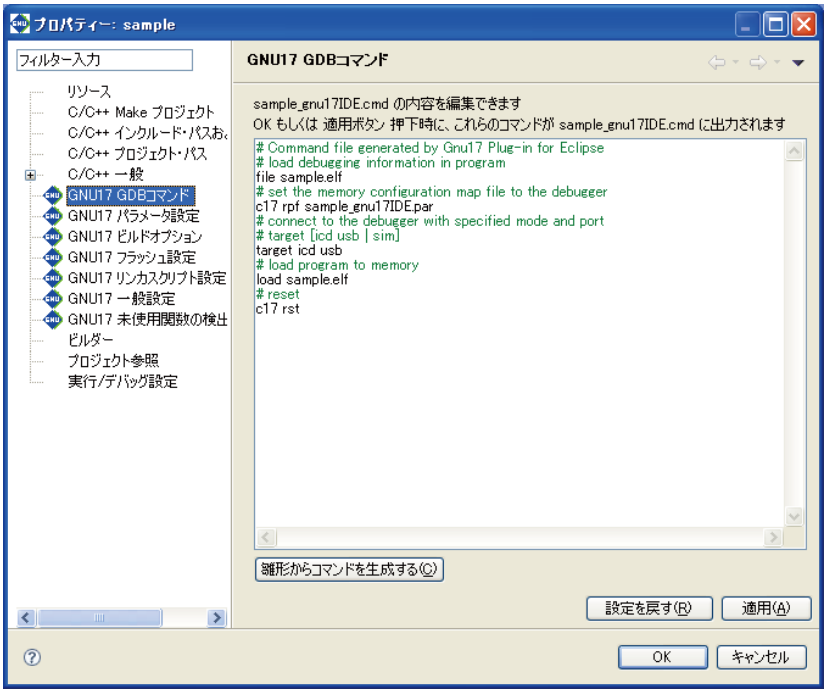
ROM領域が追加されました。

操作48: [適用]ボタンをクリックします。

以上の設定により、デバッガ起動時に`sample_gnu17IDE.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ガに渡されます。

●デバッガ起動オプションの設定

操作49: プロパティのリストから[GNU17 GDBコマンド]を選択(クリック)します。

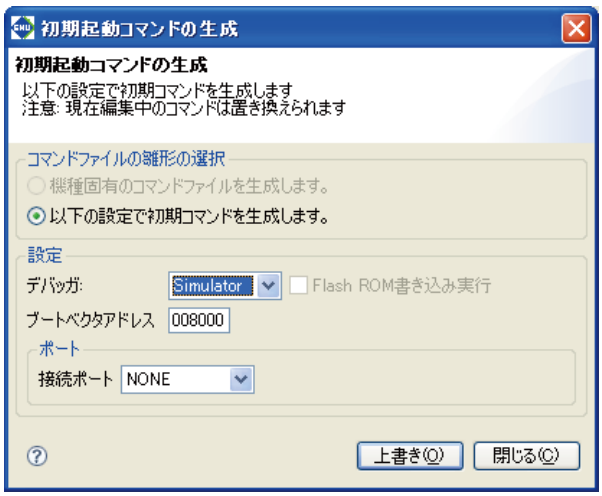


ここには、**IDE**が生成するデバッガ起動用コマンドファイルの内容が表示されます。
デバッガは使用するICDなどに合わせたモードに設定して動作させる必要があります。詳細は、"3.7 デバッグ環境"を参照してください。

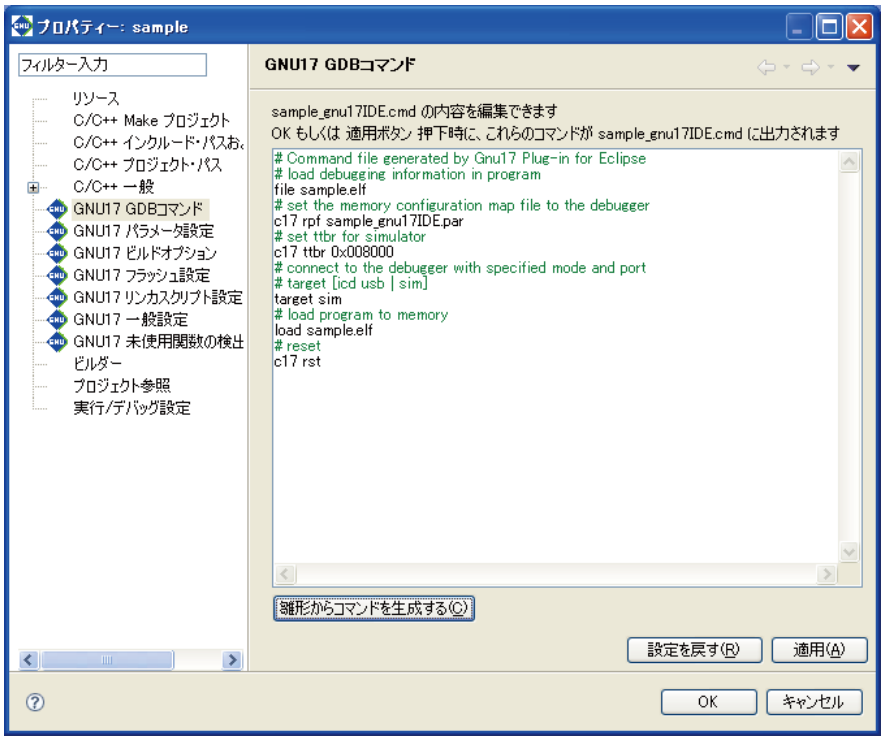
ここでは、外部機器の不要なシミュレータモードに設定してデバッグを行うことにします。
デフォルトで表示されるコマンドファイルの内容は、ICD Miniを使用してデバッグを行うようになっています。
以下の操作でシミュレータモード用に変更できます。

操作50: [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51: [デバッガ:]コンボボックスから"Simulator"を選択します。



操作52: [上書き]ボタンをクリックし、その後表示される[現在のコマンドを上書き]ダイアログボックスでも[OK]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53: [OK]ボタンをクリックします。

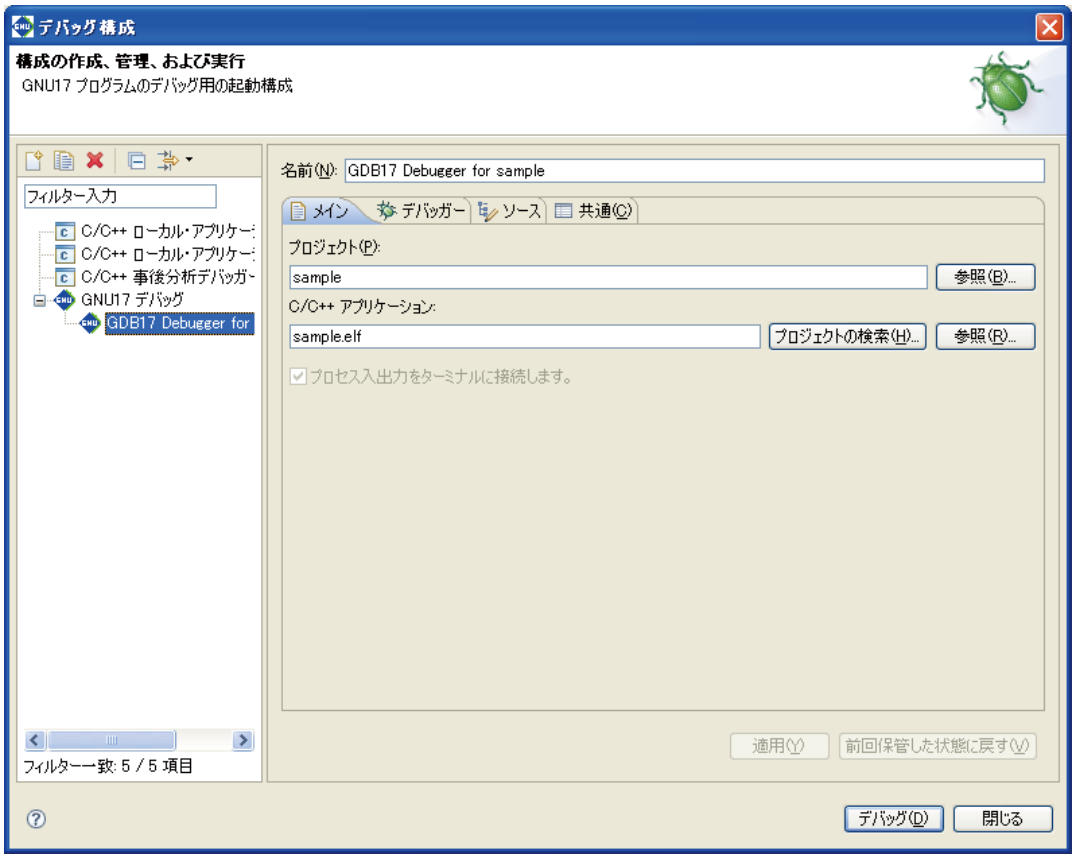
以上の設定により、sample_gnu17IDE.cmdというコマンドファイルが生成され、デバッガに渡されます。

●デバッガの起動

操作54: [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55: リストから[GDB17 Debugger for sample]を選択します。

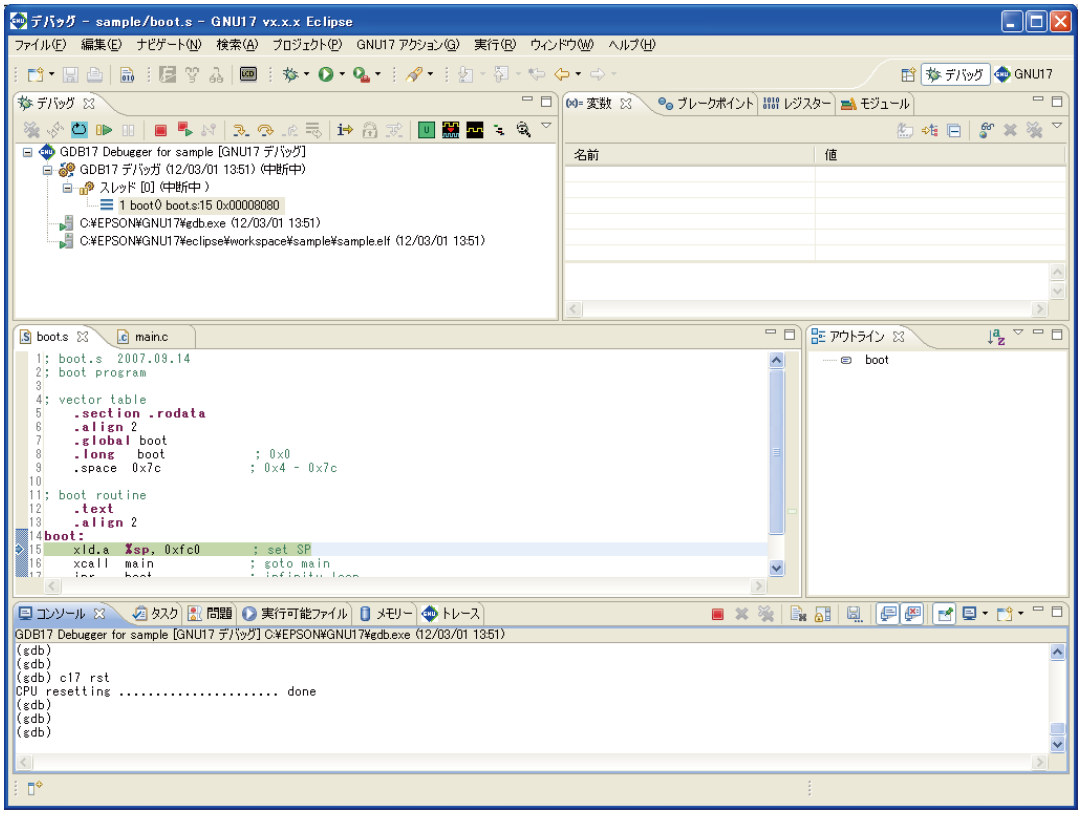


このダイアログボックスでデバッガ**gdb**のコマンドラインを編集することができます。サンプルの実行には特に変更の必要はありませんので、このままの設定でデバッガを起動します。

操作56: [実行]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、[GNU17 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、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

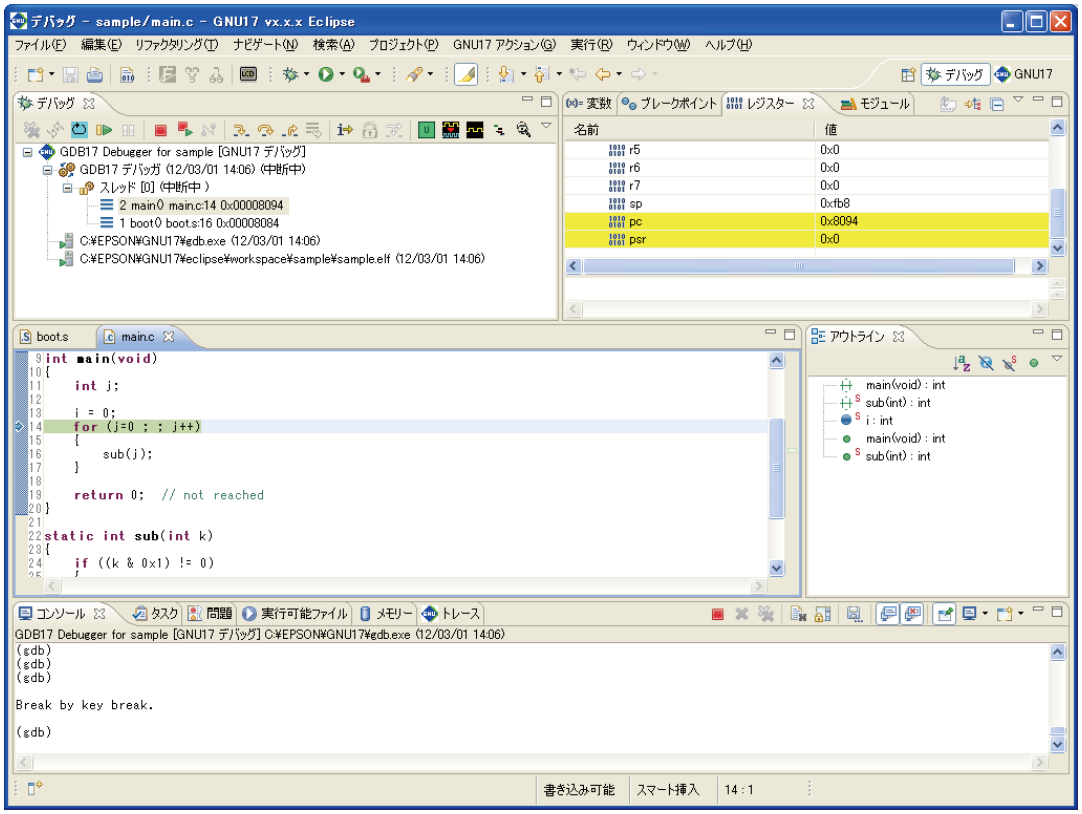
操作57: ツールバーの[再開]ボタンをクリックします。

サンプルプログラムはRAM領域のint変数i(0x0～0x3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58: ツールバーの[中断]ボタンをクリックします。



[ソース]エディター内のfor文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、0x8094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]エディター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。

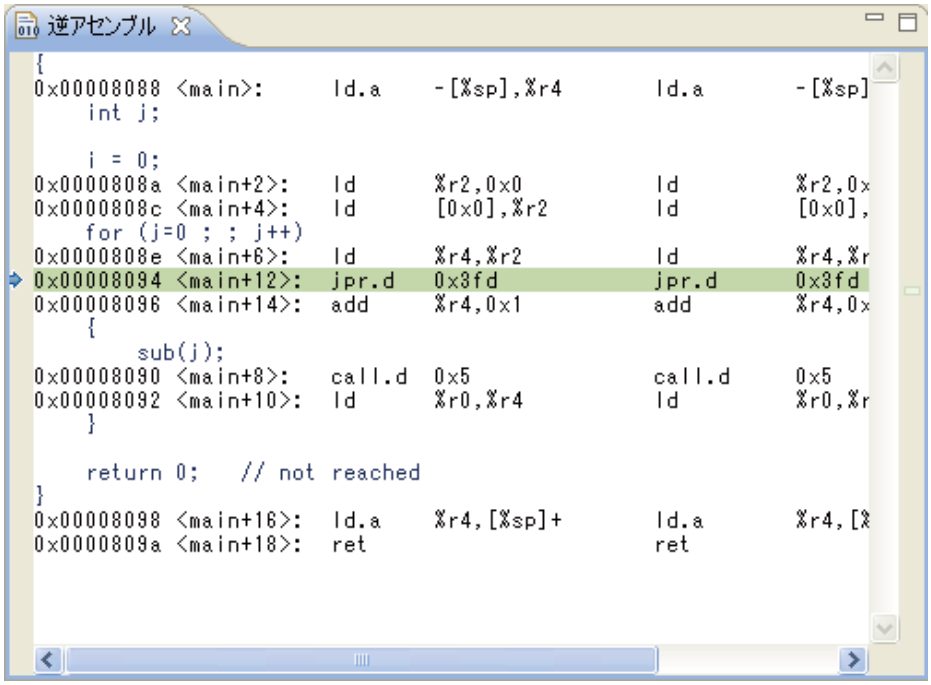
[ソース]エディターに表示されているソースは、[逆アセンブル]ビューに逆アセンブルした形式のプログラムを表示することができます。

●[逆アセンブル]ビューの表示方法

操作59: ツールバーの[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

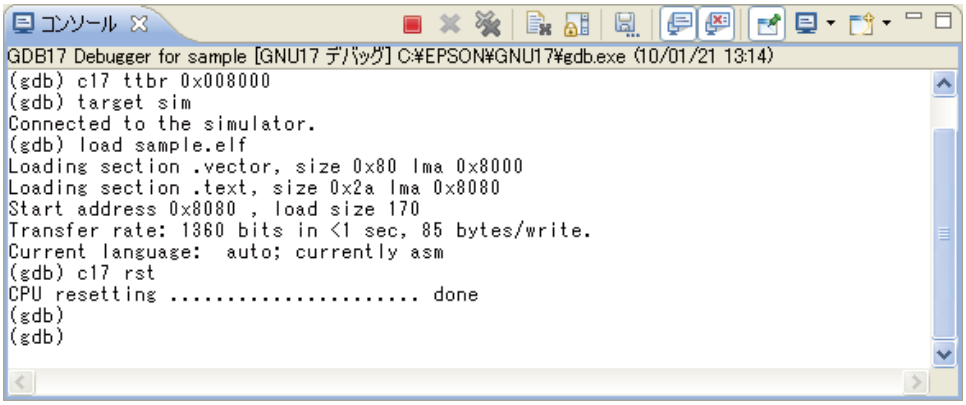
逆アセンブルは、ソース表示の各行ごとに、対応するアセンブラ表示を挿入します。この表示により、プログラムが停止しているソース行とともに、アドレスとその命令コードも[逆アセンブル]ビュー上で分かります。この場合も、Cソースは関数単位の表示となります。



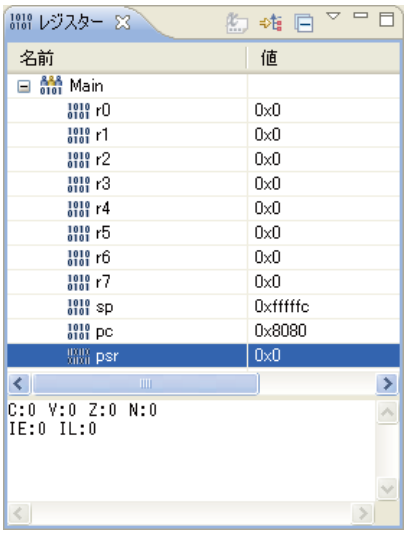
操作60: [ソース]エディターを選択し、ビューを元に戻してください。

ここで、そのほかのビューも見ておきましょう。

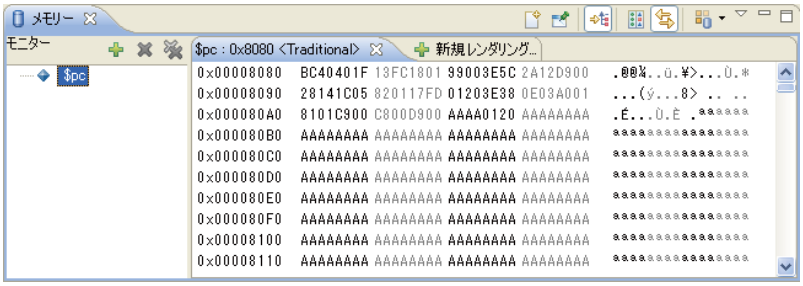
●デバッガのウィンドウについて



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57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。

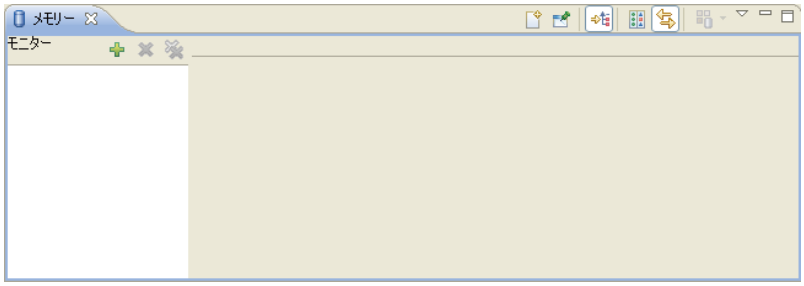


これは[レジスター]ビューです。S1C17コアのレジスタの内容を表示します。ここでレジスタデータの書き換えも行えます。

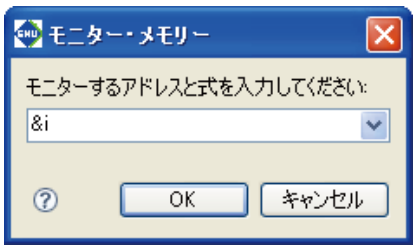


これは[メモリー]ビューです。ターゲットメモリの内容を表示します。ここで、メモリデータの書き換えも行えます。
サンプルプログラムは、先ほどの実行でint変数i(0x0～0x3番地)をインクリメントしました。[メモリー]ビューで変数iの内容を見てみましょう。

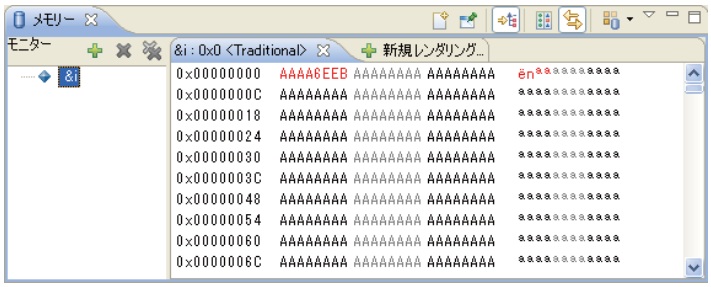
操作61: ツールバーの[ウィンドウ]メニューから[ビューの表示]>[メモリー]を選択します。
[メモリー]ビューが表示されます。



操作62: [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。

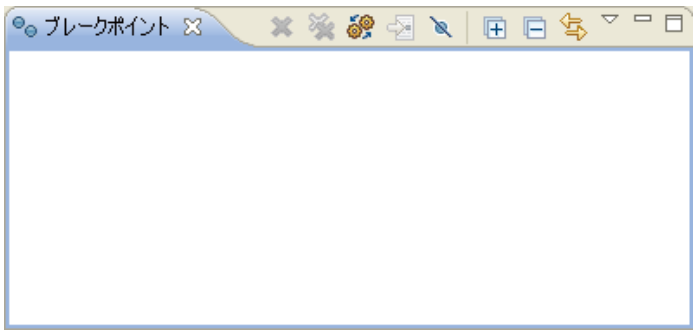


操作63: [モニター・メモリー]ダイアログボックスのテキストボックスに&iまたは0と入力し、[OK]ボタンをクリックします。



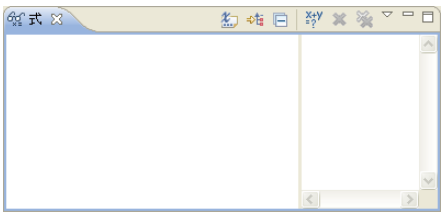
変数i(0x0番地)からメモリの内容が表示されました。これによるとiは0x6eeb(=28395)までカウントされたことを示しています。

操作64: ツールバーの[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



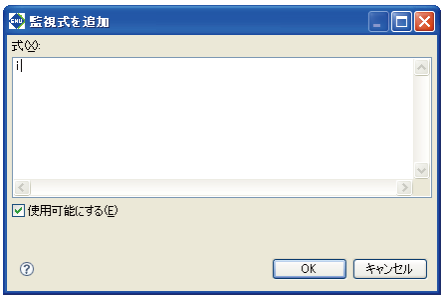
これは[ブレークポイント]ビューです。指定位置でプログラムの実行を停止させるソフトウェアPCブレークポイントの管理に使用します。

操作65: ツールバーの[ウィンドウ]メニューから[ビューの表示]>[式]を選択します。
[式]ビューが表示されます。

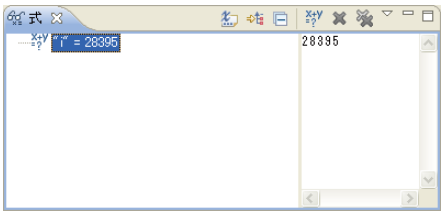


これは[式]ビューです。グローバル変数などの値をモニタするのに使用します。たとえば、先ほど[メモリー]ビューで確認した変数i(iはグローバル変数です)をこのウィンドウでモニタできるようにしてみましょう。

操作66: [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。

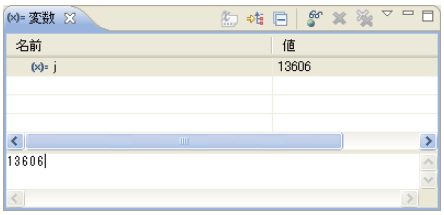


操作67: [監視式を追加]ダイアログボックスのテキストボックスにiと入力し、[OK]ボタンをクリックします。



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68: ツールバーの[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。
[変数]ビューが表示されます。



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69: [ソース]エディター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。

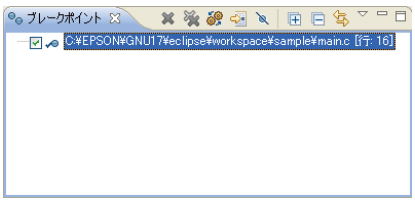
```
13      i = 0;
14      for (j=0 ; ; j++)
15      {
16          sub(j);
17      }
18
19      return 0;  // not reached
20  }
21
```

操作70: [再開]ボタンをクリックします。

操作57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71: ツールバーの[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72: [ソース]エディター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73: [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]エディター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73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74: [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74を繰り返して、[ステップイン]ボタンとの違いを[ソース]エディター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 デバッガ"を参照してください。

デバッガは次の方法で終了します。

●デバッガを終了するには

操作75: [実行]メニューから[終了]を選択します。

デバッガが終了します。IDEのウィンドウに戻したい場合は、ウィンドウ右上にある[「GNU17」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、ICDにターゲットボードを接続してデバッグを行うこともできます。そのモードでのデバッグ方法については"3.7 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76: [ファイル]メニューから[終了]を選択します。

3.3.7 PAファイル(提出用データ)の作成

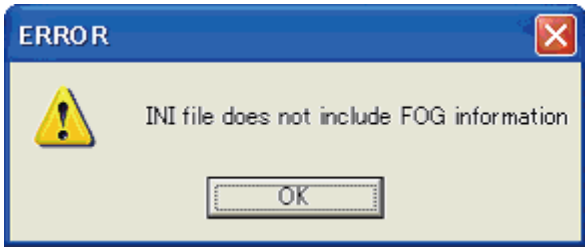
デバッグに使用したPSAファイル(ROMデータ)を元に、PAファイルを作成します。このPAファイルをセイコーエプソンへ提出してください。
(PSAファイルは、プロジェクトプロパティの[ビルドゴール切り替え]で、[ROMデータ生成(psa)]を選択してビルドした場合に生成されます)

PAファイルは以下の手順で作成します。
(1)FDCファイル(ファンクションオプションドキュメント)の作成
(2)FDCファイルとPSAファイルをパックし、PAファイルを生成

●FDCファイルの作成

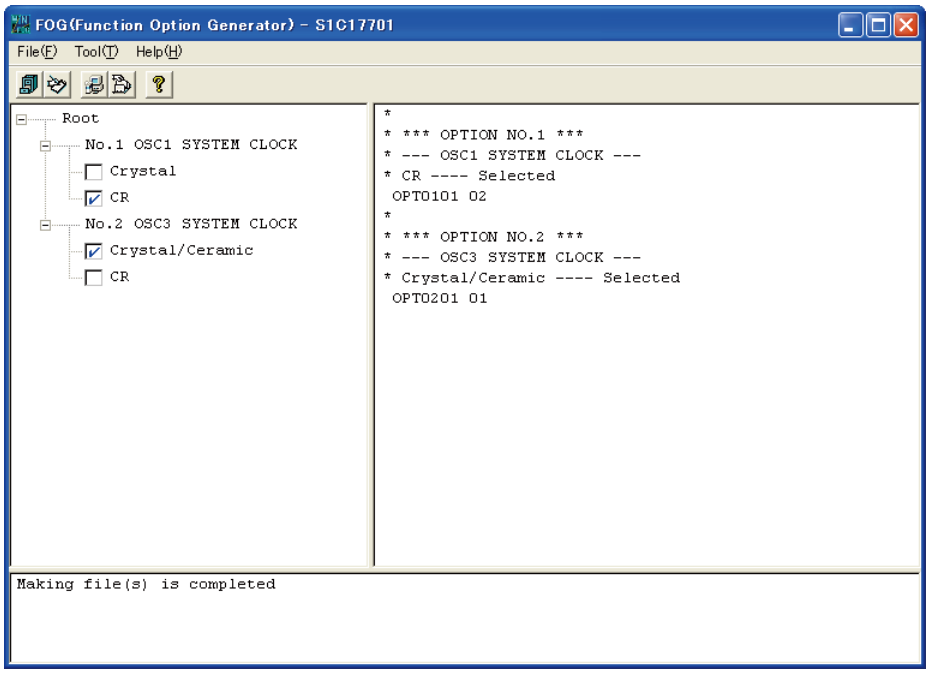
操作77: IDE上でプロジェクトを選択し、ツールバーの[WinFog17を起動]ボタンをクリックします。

ファンクションオプションの選択が不要なターゲットCPUでは、[INI file does not include FOG informaion]ダイアログが表示されます。



この場合は、FDCファイルを作成する必要はありません。ダイアログを閉じ、操作80へ進んでください。

操作78: Winfog17画面でファンクションオプションを選択します。



操作79: [Tool]メニューから[Generate]を選択すると、FDCファイルがプロジェクトディレクトリ下に生成されます。

操作80: Winfog17画面を閉じます。

winfog17.exeの詳細は、その他のツールの章を参照してください。

●FDCファイルとPSAファイルをパック

wimmdc17.exeを使用して、winfog17で生成したFDCファイルと、動作確認を行ったPSAファイルからPAファイルを生成します。

操作81: プロジェクトを選択し、ツールバーの[WinMdc17でパック]ボタンをクリックします。"Pack successfully completed!!"のダイアログが表示され、PAファイルがプロジェクトディレクトリ下に生成されます。

winmdc17.exeの詳細は、その他のツールの章を参照してください。

以上の操作で生成されたPAファイルをセイコーエプソンへ提出してください。

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ファイルとリンカスクリプトファイルを使用してプログラムをビルドし、ユーザのコマンドファイルを使用して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。

使用するファイル

説明には`c:¥EPSON¥gnu17¥sample¥S1C17common¥simulator¥simulatedIO`ディレクトリにある以下のサンプルファイルを使用します。

`vector.c`	ベクタテーブルファイル
`main.c`	Cソースファイル
`mymakefile.mak`	makeファイル
`myldsfile.lds`	リンカスクリプトファイル
`mycmdfile.cmd`	デバッガコマンドファイル(シミュレータモード用)
`myparfile.par`	パラメータファイル

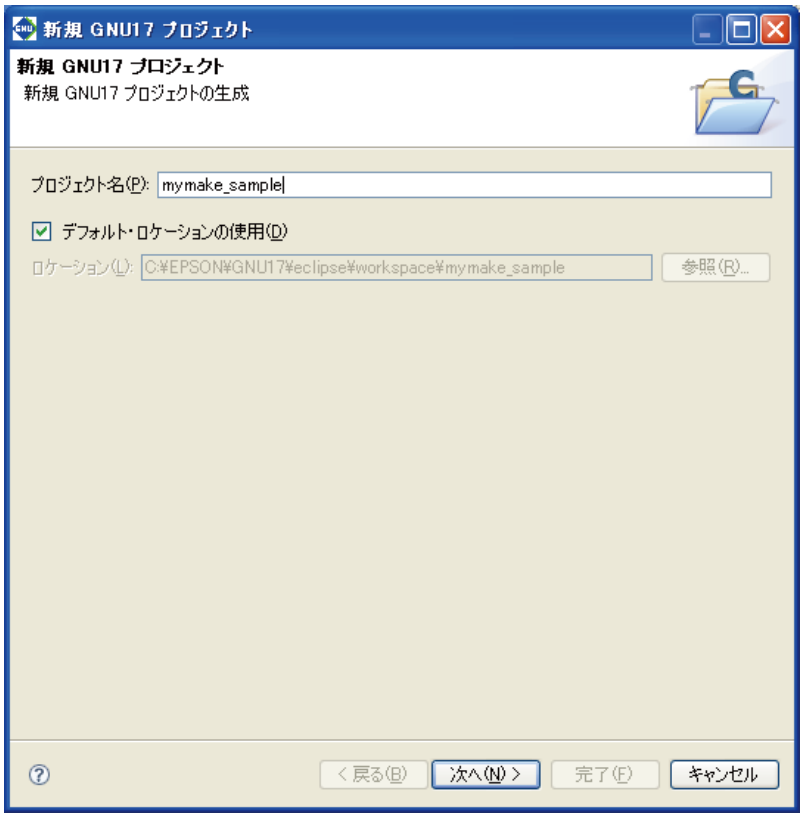
3.4.1 プロジェクトの作成

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 GNU17 プロジェクト]を選択し、[新規 GNU17 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"`mymake_sample`"を入力します。



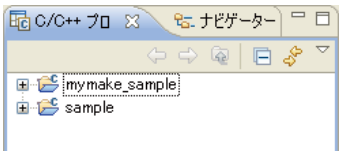
このチュートリアルでは、プロジェクトフォルダをワークスペース(デフォルト)に作成しますので、[デフォルト・ロケーションの使用]チェックボックスは選択したままにしておきます。

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せんので、ターゲットCPU、メモリモデル、.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: [完了]ボタンをクリックして、プロジェクトを作成します。

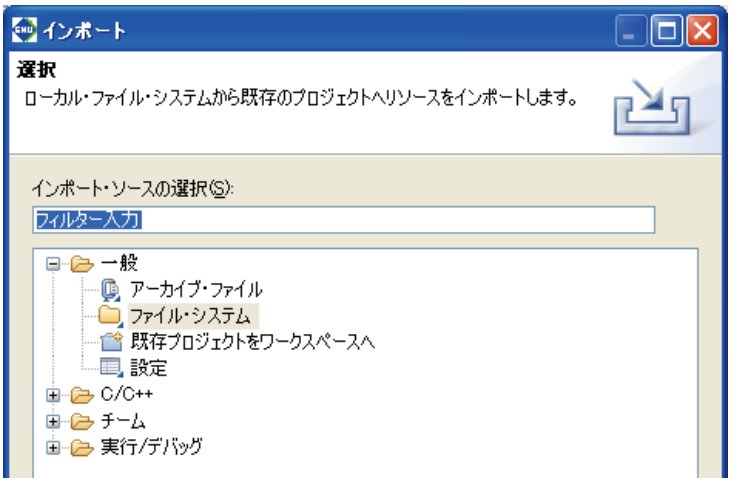


3.4.2 ソースファイルのインポート

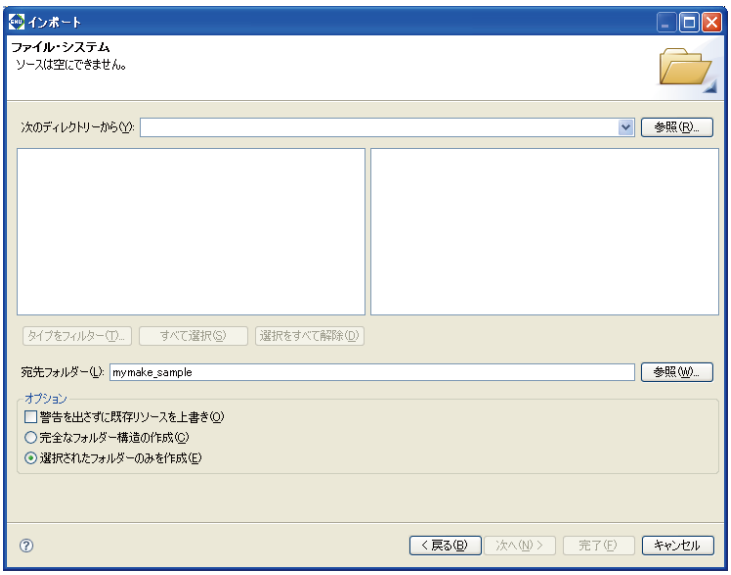
プロジェクトが作成できましたら、次にソースファイルをインポートします。また、同じディレクトリにあるmakeファイルもここでインポートしておくことにします。

操作7: [ナビゲーター]ビューでプロジェクト"mymake_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。

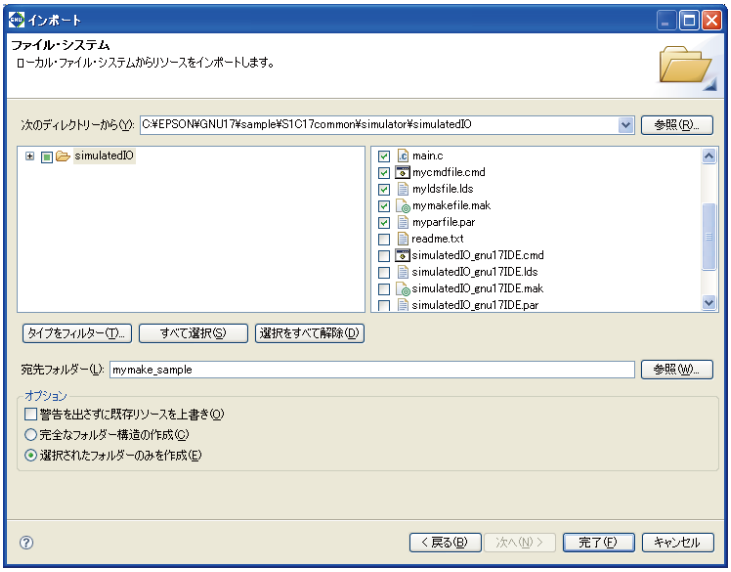


操作9: [次のディレクトリーから:]の[参照...]ボタンで、インポートするファイルがあるC:¥EPSON¥gnu17¥sample¥S1C17common¥simulator¥simulatedIOディレクトリを選択します。

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。

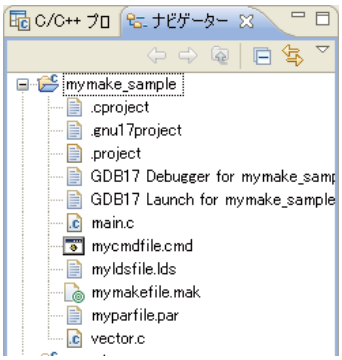
操作10: 右のリストボックスに表示されている以下のファイルのチェックボックスを選択します。

`vector.c`	ベクタテーブルファイル
`main.c`	Cソースファイル
`mymakefile.mak`	makeファイル
`myldsfile.lds`	リンカスクリプトファイル
`mycmdfile.cmd`	デバッガコマンドファイル
`myparfile.par`	パラメータファイル



操作11: [完了]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。

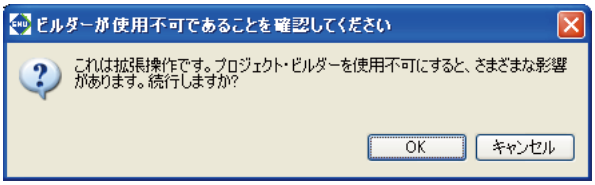


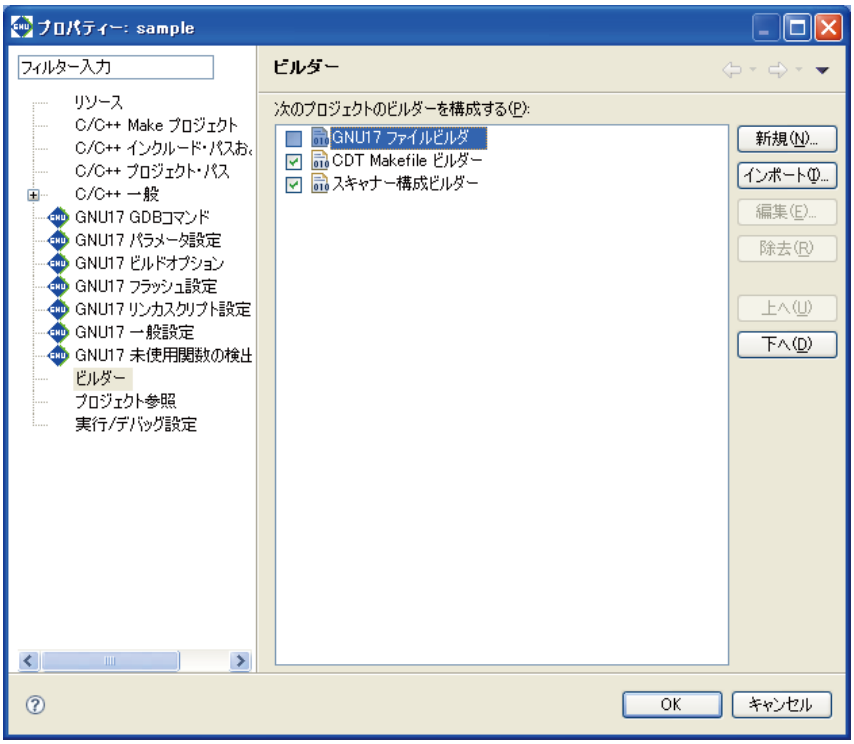
3.4.3 GNU17ファイルビルダの無効化

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、デバッガコマンドファイルを自動生成してビルドやデバッガ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"mymake_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[プロパティー]を選択して[プロパティー]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、[GNU17 ファイルビルダ]のチェックを外します。確認用のダイアログボックスが表示されますので、[OK]ボタンをクリックしてください。





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、デバッガ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。

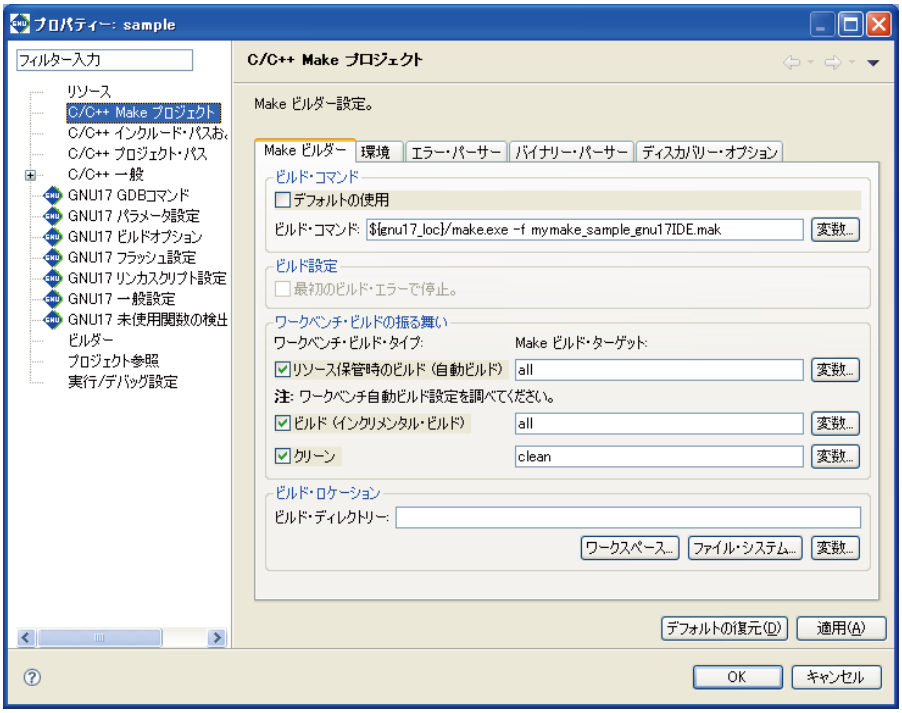
3.4.4 makeファイルの設定と修正

●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mymakefile.mak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"mymake_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[プロパティー]を選択して[プロパティー]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"mymake_sample_gnu17IDE.mak"を"mymakefile.mak"に変更します。

mymakefile.mak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。

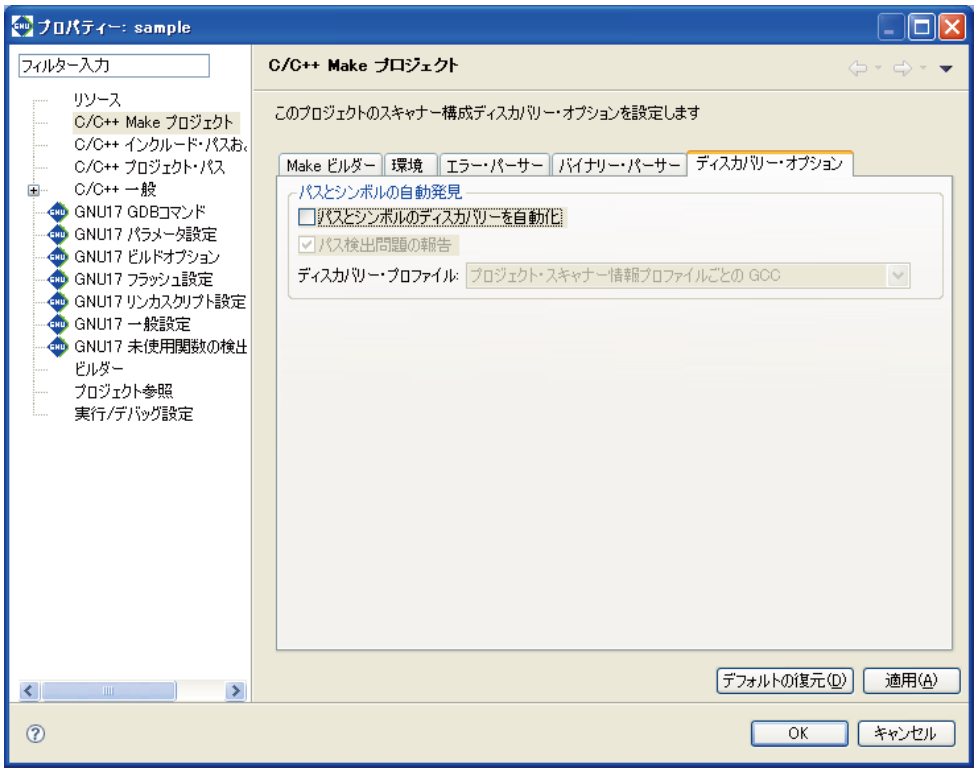
[ビルド(インクリメンタル・ビルド)]
ビルド実行時に呼び出すmakeファイル内のターゲットを指定します。

[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]のチェックをはずします。



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

IDE以外で作成した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。相対パスの記述は、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。

例: 変更前　`SRCDIR=　　..`

　　変更後　**`SRCDIR=/cygdrive/c/EPSON/gnu17/sample/S1C17common/simulator`**

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21: [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"mymake_sample"を選択します。

操作22: [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stdio.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。)

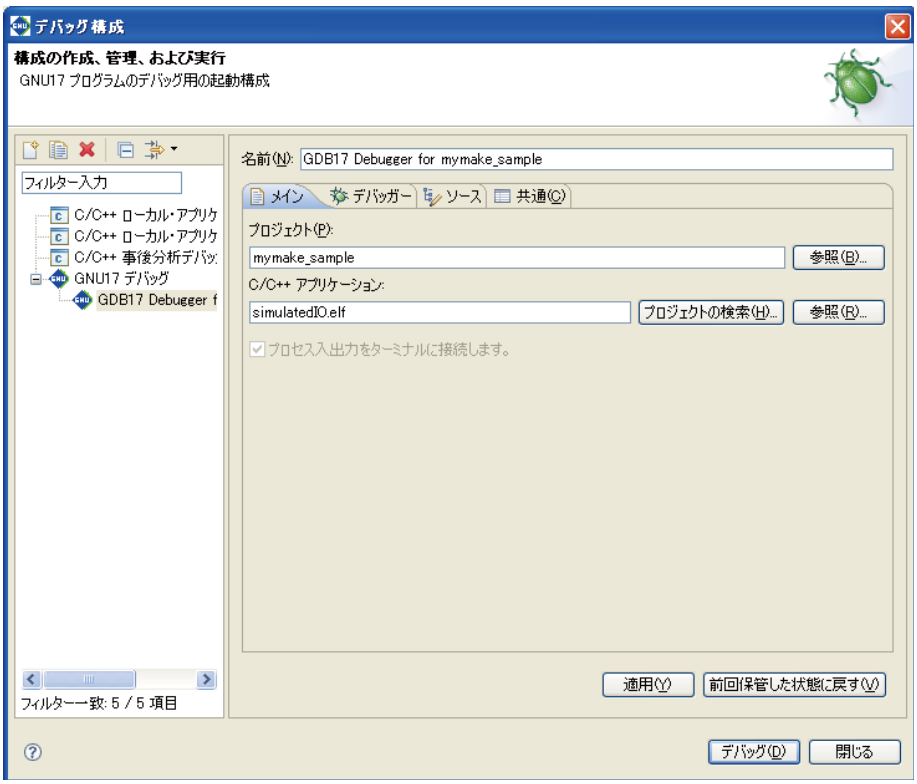
3.4.6 デバッガの起動

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。

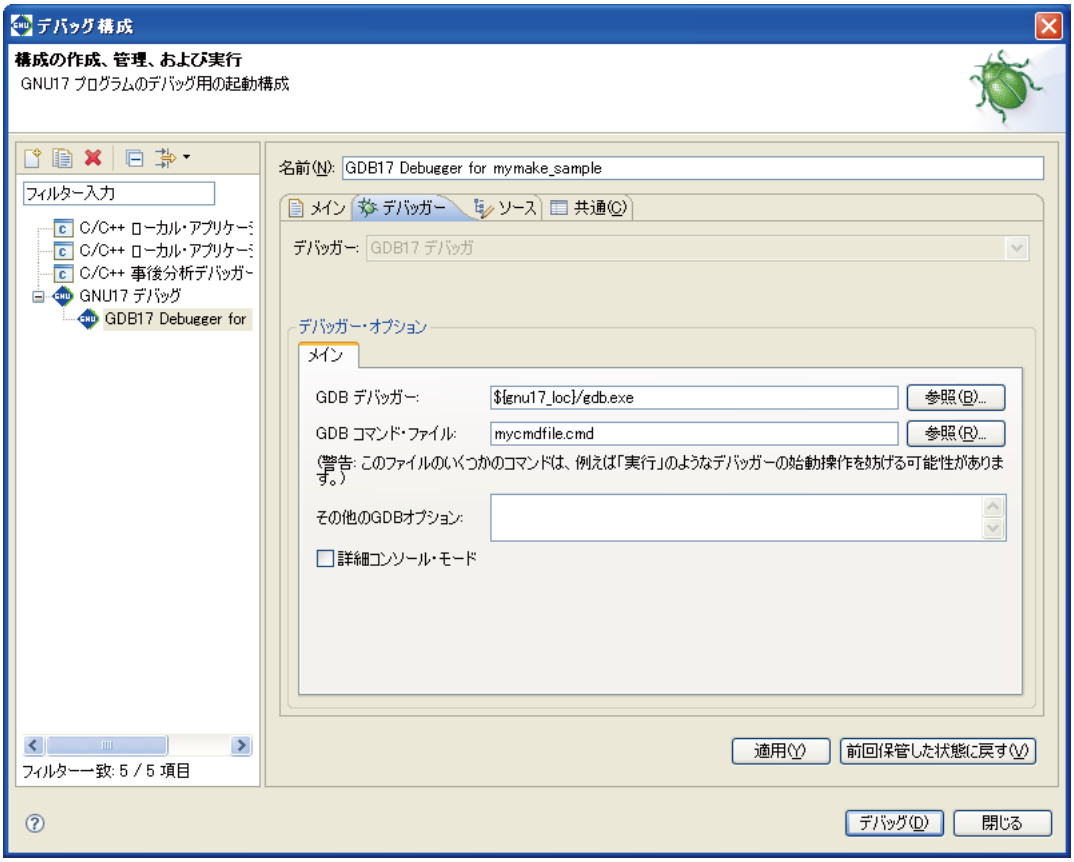
操作23: [実行]メニューから[デバッグの構成...]を選択して、[デバッグ構成]ダイアログボックスを表示させます。

操作24: リストから[GDB17 Debugger for mymake_sample]を選択し、[メイン]タブのページを表示させます。

操作25: [C/C++ アプリケーション:]にあるelfファイルを「simulatedIO.elf」に変更します。



操作26: [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 [mycmdfile.cmd] に変更します。



操作27: [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、mycmdfile.cmdが実行されます。

操作28: 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)

S1C17 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¥gnu17¥sample¥S1C17common¥simulator¥simulatedIO
```

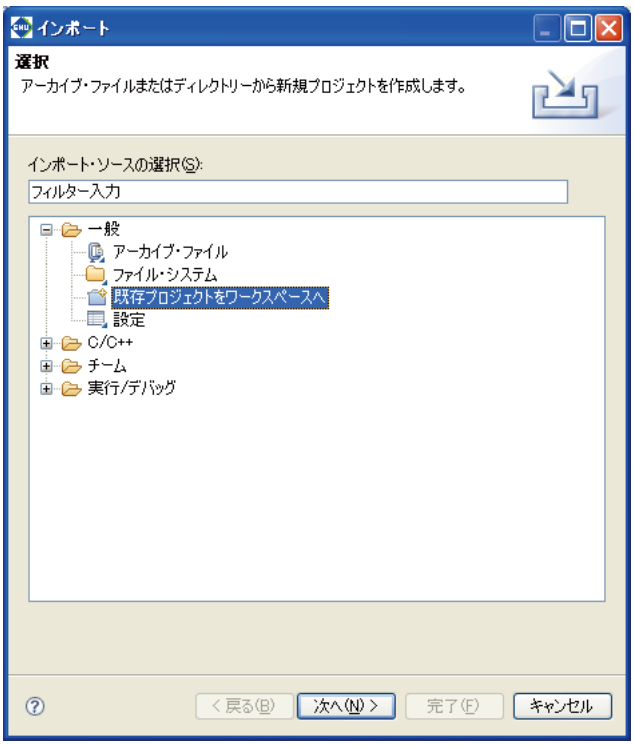
●プロジェクトをインポートするには

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。

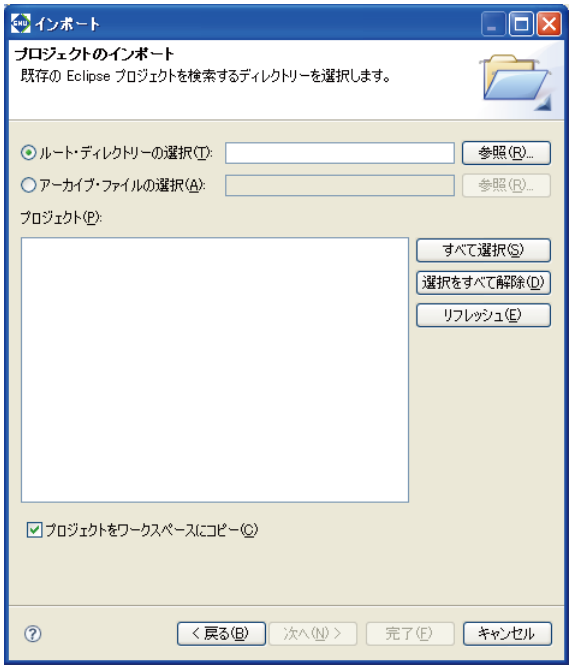
操作1: **IDE**を起動します。

操作2: [ファイル]メニューから[インポート...]を選択します。

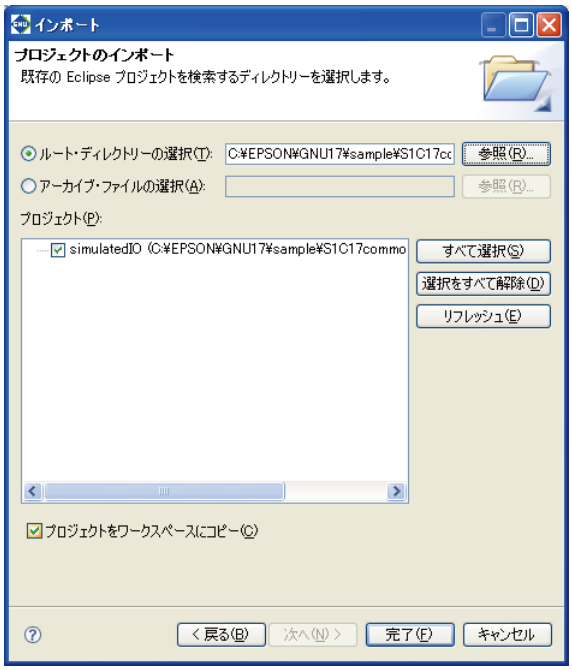
[インポート]ウィザードが起動します。



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。



操作4: [ルート・ディレクトリーの選択]の[参照...]ボタンで、インポートするプロジェクトフォルダ`C:¥EPSON ¥gnu17¥sample¥S1C17common¥simulator¥simulatedIO`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: [完了]ボタンをクリックします。

※ [完了]ボタンクリック後、インポートするプロジェクトのターゲットCPUに対する機種別情報ファイルが存在しない場合、エラーダイアログ表示後、ターゲットCPU選択画面が表示されます。ターゲットCPU選択画面でプロジェクトに合ったターゲットCPUを選択して下さい。詳細は、"5.4.5　既存プロジェクトのインポート"を参照して下さい。

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/.gnu17project/)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¥gnu17`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[プロパティー]ダイアログボックスの[GNU17 ビルドオプション]で変更してください。

3.6 チュートリアル4(ES-Sim17の使用方法)

S5U1C17001Cパッケージには、デバッグ時にターゲット機種の入出力ポートやLCD表示などのハードウェア機能をPC上でシミュレートする組み込みシステムシミュレータ(**ES-Sim17**)が含まれています。**ES-Sim17**はデバッガにより起動しますが、そのためには**IDE**上でいくつかの設定が必要になります。ここでは、**ES-Sim17**を使用するための**IDE**上の操作を説明します。それ以外の操作についてはチュートリアル1～3を参照してください。また、**ES-Sim17**の詳細については、"10.11 組み込みシステムシミュレータ(**ES-Sim17**)"を参照してください。

3.6.1 ES-Sim17を起動するための設定

ES-Sim17をデバッグ時に起動するためには、次の2つの設定が必要です。

1. ターゲットCPUの指定(パラメータファイル)
2. シミュレータモードでのデバッガの起動(デバッガコマンドファイル)

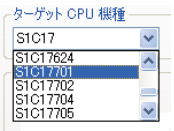
ここでは、新規プロジェクトの作成時に上記の設定を行う、**IDE**上の操作方法を説明します。

● ターゲットCPUの指定

ハードウェア機能をシミュレートするため、使用するターゲットCPUを指定しておく必要があります。プロジェクトを新規作成する場合は、次の手順でターゲットCPUを指定します。

操作1: ツールバーの[新規]プルダウンメニューから[新規 GNU17 プロジェクト]を選択して[新規 GNU17 プロジェクト]ウィザードを起動します。 ウィザードの最初のページで、[プロジェクト名:]ボックスにプロジェクト名を入力し、[次へ>]ボタンをクリックします。

操作2: [ターゲット CPU 機種]コンボボックスから"S1C17701"を選択します。



"S1C17"はS1C17コアのみをシミュレートしますので、デバッグ時に**ES-Sim17**は起動しません。
ES-Sim17を使用せずにシミュレータモードでデバッグを行う場合は、"S1C17"を選択してください。
ICDを使用して(ICD Miniモードで)デバッグを行う場合は、ターゲットCPUの選択にかかわらず**ES-Sim17**は起動しません。
リストに表示される機種は、新機種のリリース等による設定ファイルの変更により追加/削除されます。

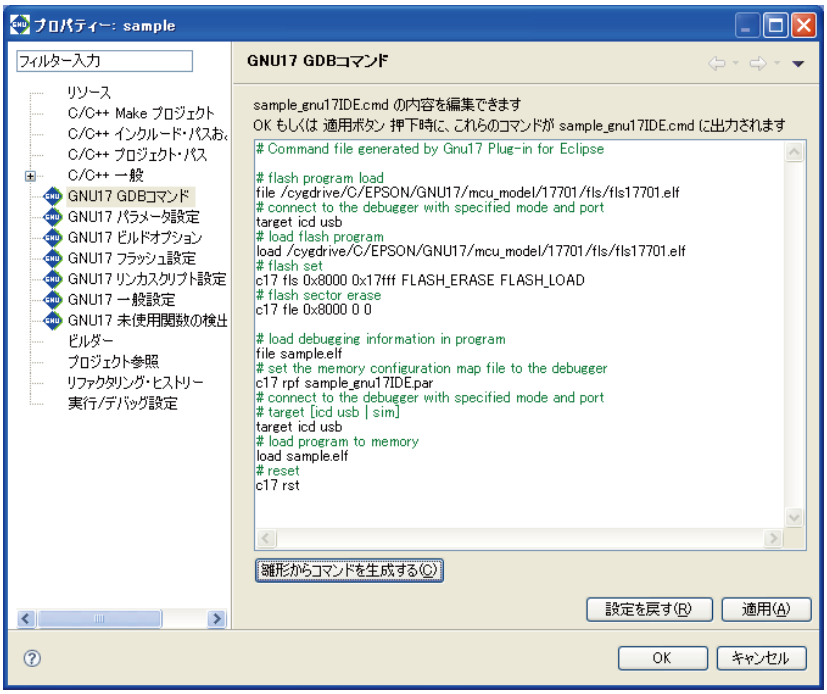
操作3: 必要に応じてメモリモデルとベクタセクションの設定を行い、[完了]ボタンでウィザードを終了します。

この後は、通常と同様にソースの編集とビルドを行います。
ターゲットCPUは[プロパティー]ダイアログボックスの [GNU17 一般設定]ページでも、上記と同様に指定できます。

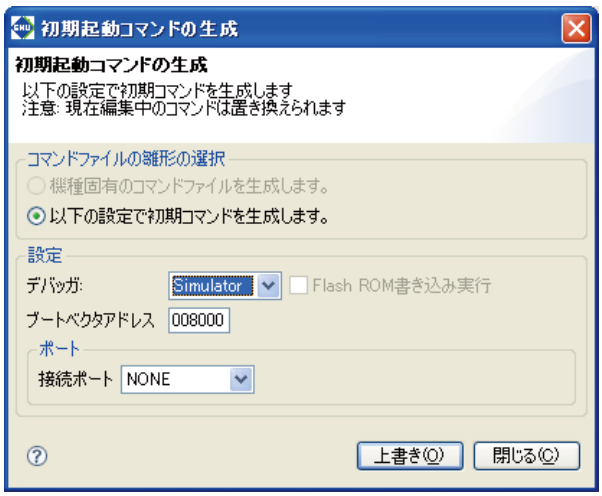
● シミュレータモード用のデバッガコマンドファイルの作成

ES-Sim17はデバッガがシミュレータモード時にのみ動作可能です。デバッガをシミュレータモードで起動させるための**IDE**上での操作方法は次のとおりです。

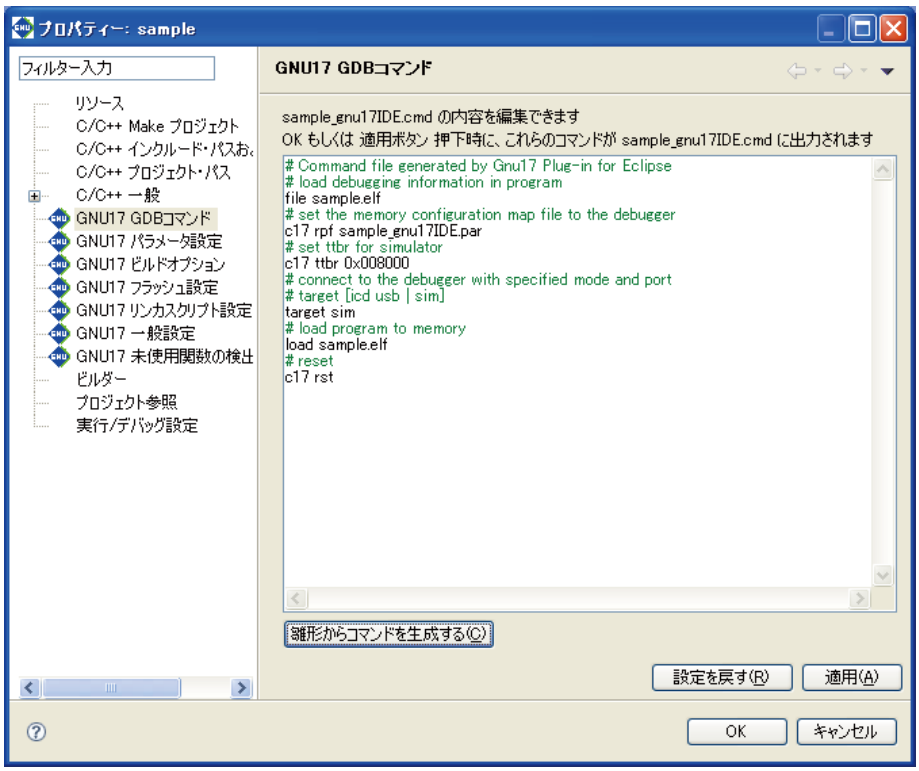
操作4: [プロジェクト]メニューから[プロパティー]を選択して[プロパティー]ダイアログボックスを表示させ、[GNU17 GDBコマンド]のページを開きます。



操作5: [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、[デバッガ:]コンボボックスから"Simulator"を選択します。



操作6: [上書き]ボタンをクリックします。



操作7: [OK]ボタンをクリックします。

以上の操作により、デバッガ起動時に**ES-Sim17**も同時に起動するための設定は完了しました。その他、ソースの編集やビルドにおいては、**ES-Sim17**を起動するための操作は不要です。

3.6.2 既存プロジェクトでES-Sim17を使用する方法

ここでは、S1C17コアをターゲットとして作成されているプロジェクトをインポートし、ターゲット機種を変更し直してデバッガと**ES-Sim17**を起動するまでの手順を説明します。

使用するサンプルプロジェクトフォルダ

```
C:¥EPSON¥gnu17¥sample¥S1C17701¥simulator¥application
```

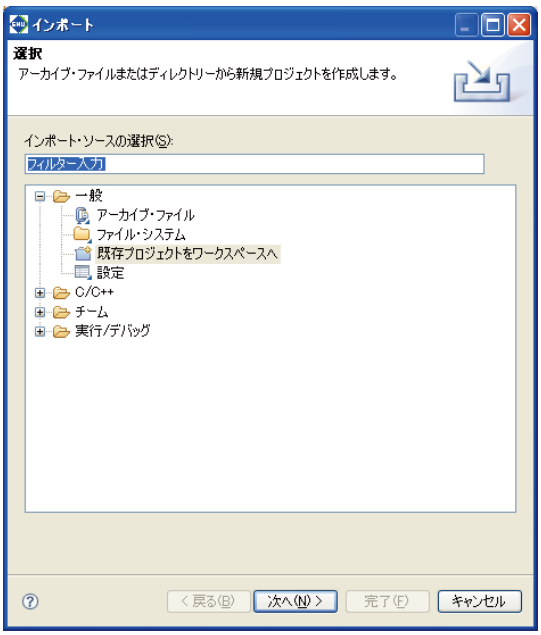
● インポート

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。

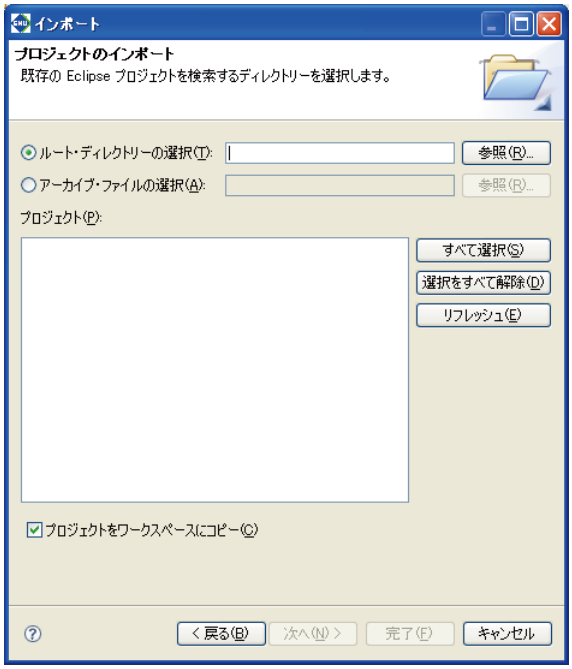
操作1: **IDE**を起動します。

操作2: [ファイル]メニューから[インポート...]を選択します。

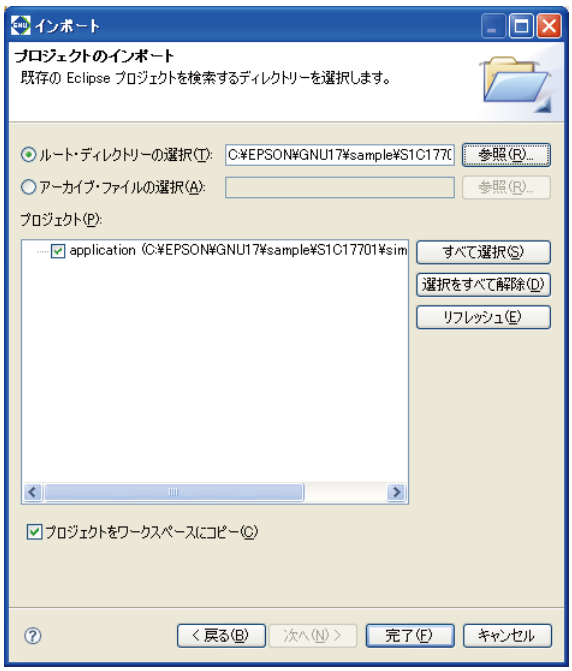
[インポート]ウィザードが起動します。



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。



操作4: [ルート・ディレクトリーの選択]の[参照...]ボタンで、インポートするプロジェクトフォルダ`C:¥EPSON ¥gnu17¥sample¥S1C17701¥simulator¥application`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: [完了]ボタンをクリックします。

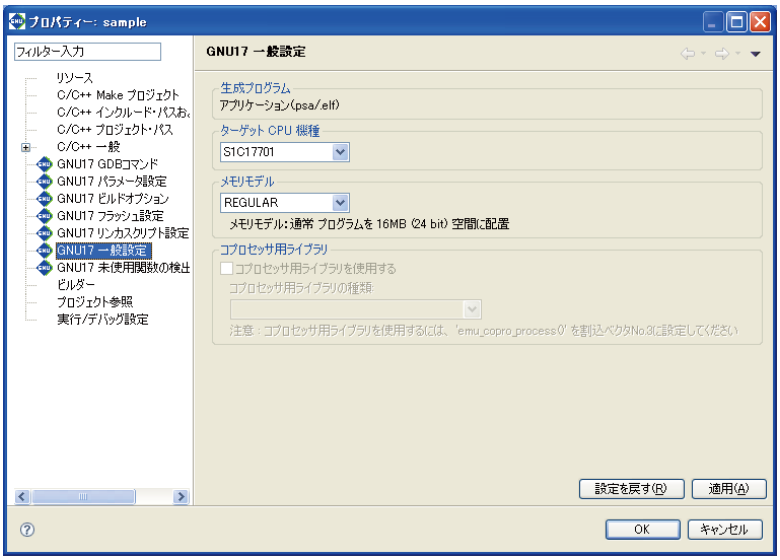
※ [完了]ボタンクリック後、インポートするプロジェクトのターゲットCPUに対する機種別情報ファイルが存在しない場合、エラーダイアログ表示後、ターゲットCPU選択画面が表示されます。ターゲットCPU選択画面でプロジェクトに合ったターゲットCPUを選択して下さい。詳細は、"5.4.5　既存プロジェクトのインポート"を参照して下さい。

これで**IDE**にプロジェクトとしてインポートされました。

● ターゲットCPU(パラメータファイル)の確認

サンプルプロジェクトに、**ES-Sim17**でシミュレートするターゲットプロセッサ"S1C17701"が選択されていることを確認します。

操作7: [プロジェクト]メニューから[プロパティー]を選択して[プロパティー]ダイアログボックスを表示させ、[GNU17 一般設定]のページを開きます。



操作8: [ターゲット CPU 機種]に"S1C17701"が選択されていることを確認します。

操作9: [OK]ボタンをクリックします。

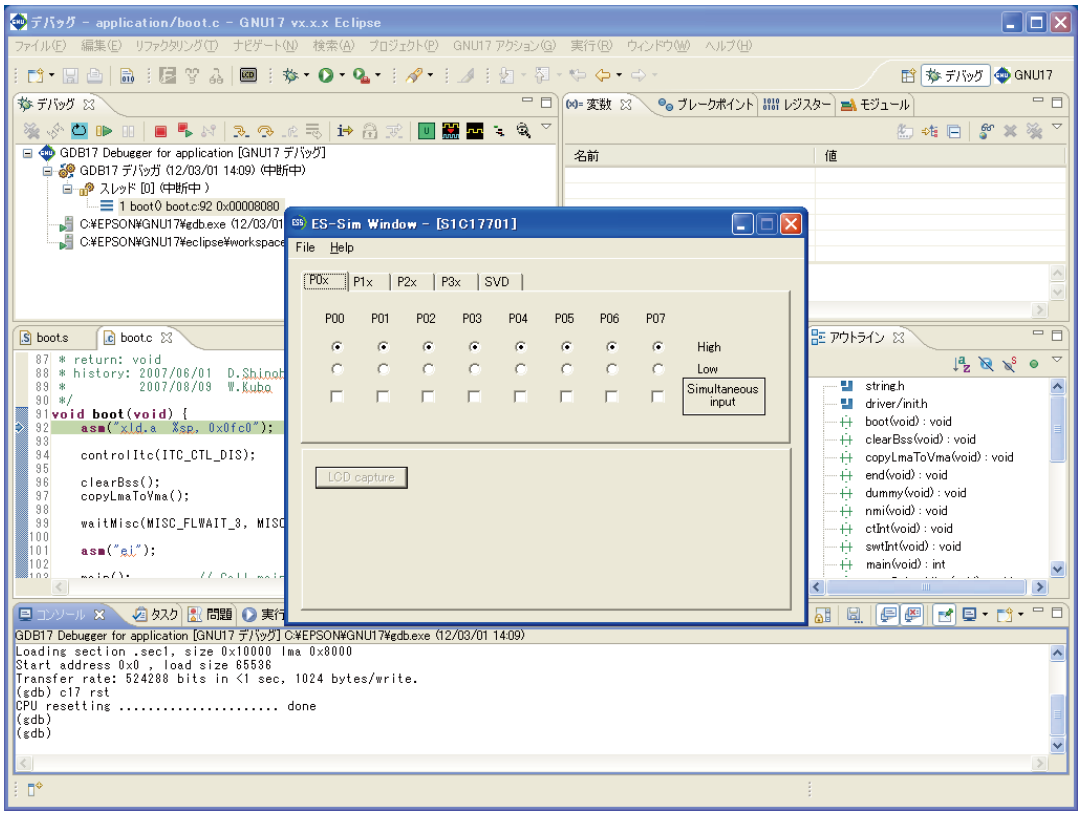
デバッガで**ES-Sim17**を起動するためのターゲット機種名は、デバッガに渡されるパラメータファイルに次のよう記述されます。

```
ESSIM   S1C17701     ("S1C17701"はターゲットプロセッサ名です。)
```

※ ユーザ独自のパラメータファイルを作成している場合は、上記の記述をファイルに書き加えます。

● デバッガ起動

操作10: [実行]メニューから[デバッグの構成...]>[GDB17 Debugger for application]を選択します。



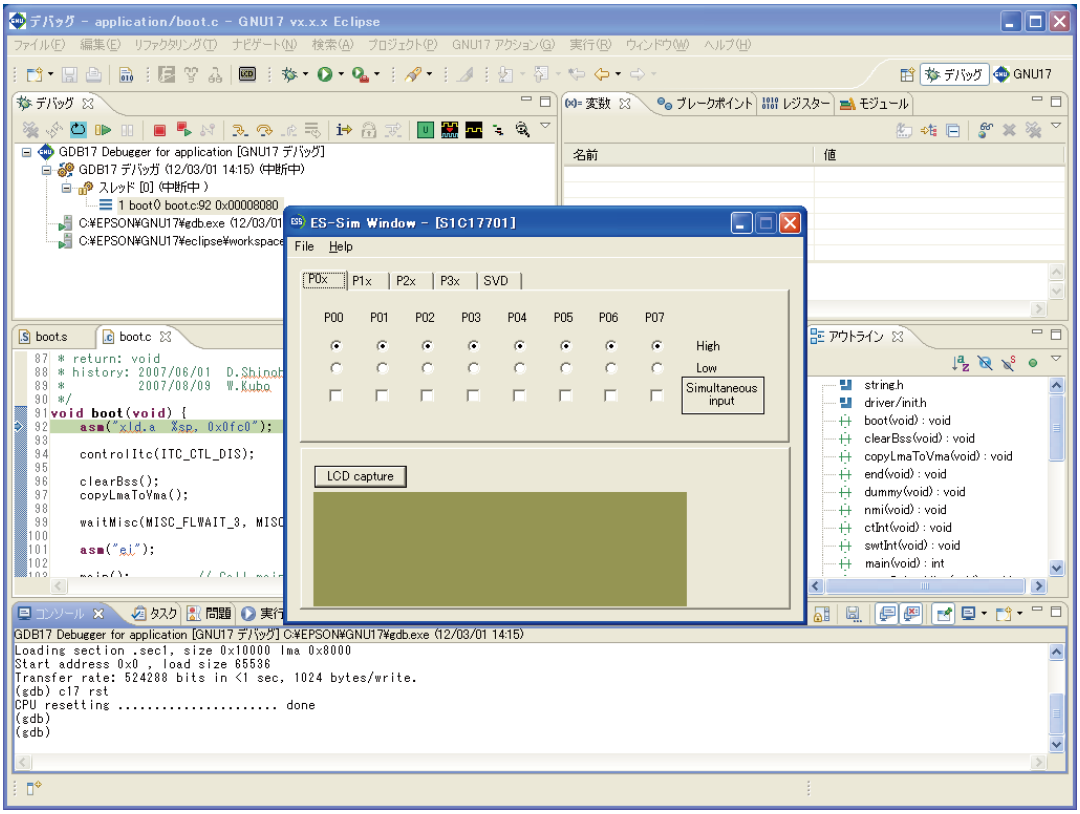
デバッガが起動し、シミュレータモードに設定されると同時に、[ES-Sim]ウィンドウが表示されます。

● LCDシミュレート用ファイルの読み込み

ES-Sim17はLCDUtil17 で作成したLCDファイル(.lcd)を読み込むことで、LCDのシミュレートが可能です。プロジェクトで選択した機種に対応したLCDファイル(.lcd) を読み込んでください。
LCDファイル(.lcd)の作成方法については"12.8 LCDUtil17(LCDパネルカスタマイズツール)"を参照ください。

操作11: [File]メニューから[Load lcd file] を選択します。

ファイルオープンダイアログが出ますので、LCDUtil17で作成したS1C17701用LCDファイル C:¥EPSON¥gnu17¥tool¥LcdUtil17¥sample¥SVT17701.lcd をオープンします。



ES-Sim17の機能と操作方法については、"10.11 組み込みシステムシミュレータ(**ES-Sim17**)"を参照してください。

3.7 デバッグ環境

デバッガは2種類のデバッグ環境に対応しており、**target**コマンドで使用するモードを設定します。

●ICD Miniモード

ICD Mini（S5U1C17001H）またはICDボード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ます。

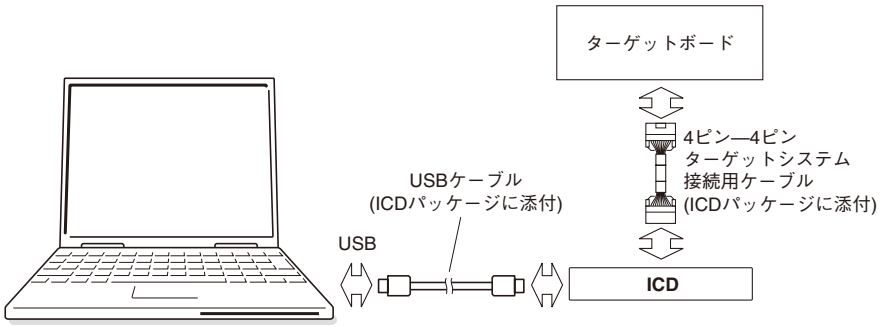


図3.7.1 ICDを使用するデバッグシステム例

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 Mini"を選択して、起動用コマンドファイルを生成します。

ICD Miniモードで起動する場合、ICD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。ICDの取り扱い等については、使用するICDのマニュアルを参照してください。

ICD Miniモードでは、トレース機能を使用することはできません。

注: • **サポートしているOSは、Microsoft Windowsです。**
• **ICD Mini(S5U1C17001H)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¥gnu17¥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、ICD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[デバッガ:]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ではトレース機能が使用可能です。フラッシュライタ機能は使用できません。

3.8 セクションとリンク

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となるセクション管理の概念を説明しておきます。
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コンパイラ**xgcc**では、大きく分けて4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。

.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、"5.7.9 リンカスクリプトの編集"を参照してください。

仮想セクション
LMA設定を行っているセクションに対し、必ず生成されるセクションです。
LMA設定位置に仮想的に表示されるため、セクションとしての役割はありません。
詳細は、"5.7.9 リンカスクリプトの編集"を参照してください。

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)

```
.section .rodata              ; .rodata section

.global BOOT
.align 2
.long BOOT                    ; 0x00 reset
.long UNALIGN                 ; 0x01 unalign
.long EXCEPTION               ; 0x02 nmi
               :
               :
.text                         ; .text section
BOOT:
         xld.a  %sp, 0x3f00   ; set SP
               :
               :
```

(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)

```
/* Linker Script file */

SECTIONS
{
  /* stack pointer symbols */
  __START_stack = 0x000FC0;

  /* location counter */
  . = 0x0;                                                    ...(1)

  /* section information */
  .bss 0x000000 :                                             ...(2)
    {
     __START_bss = . ;                                        ...(3)
     file1.o(.bss)                                            ...(4)
     file2.o(.bss)
     file3.o(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
     }
     __END_bss = . ;                                          ...(5)

  .data __END_bss : AT( __END_rodata )                        ...(6)
    {
     __START_data = . ;
     file1.o(.data)
     file2.o(.data)
     file3.o(.data)
     C:/EPSON/gnu17/lib/24bit/libc.a(.data)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
     C:/EPSON/gnu17/lib/24bit/libc.a(.data)
    }
     __END_data = . ;

  .vector 0x008000 :                                          ...(7)
    {
     __START_vector = . ;
     file1.o(.rodata)
     }
     __END_vector = . ;

  .text __END_vector :                                        ...(8)
    {
     __START_text = . ;
     file1.o(.text)
     file2.o(.text)
     file3.o(.text)
     C:/EPSON/gnu17/lib/24bit/libc.a(.text)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
     C:/EPSON/gnu17/lib/24bit/libc.a(.text)
     }
     __END_text = . ;

  .rodata __END_text :                                        ...(9)
    {
     __START_rodata = . ;
     file2.o(.rodata)
     file3.o(.rodata)
     C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodata)
     C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
     }
     __END_rodata = . ;
```

```
__START_data_lma = __END_rodata ;                    ...(10)
__END_data_lma = __END_rodata + ( __END_data - __START_data );
}
```

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るものとします。

```
file1    .rodataセクションおよび.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) ロケーションカウンタを0x0に設定します。この位置がアドレス0x0となります。この後のアドレスの指定や、セクションの配置に伴ってロケーションカウンタは変更されます。リンカスクリプト中では、'.'によって現在のロケーションカウンタ値を参照することができます。

(2) 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。配置アドレスを0x000000に指定していますので、.bssセクションは0x000000から始まります。

(3) **IDE**で作成されるリンカスクリプトには、このようにセクションの開始アドレスを示すシンボルが定義されます。ここでは.bss出力セクションの開始アドレスを示すシンボルを__START_bssという名称で定義しています。'.'はロケーションカウンタ値を表しますので、シンボル値は0x000000になります。他のセクションでも同様にセクション開始アドレスを定義しています。これらのシンボルはプログラムソース内からもグローバルシンボルとして参照可能です。

(4) セクションに配置するオブジェクトおよび基本セクションの属性を指定します。この指定により、.bss出力セクションにはfile1.o、file2.o、file3.oの順にそれぞれの.bssセクションが配置されます。file2.o内に.bssセクションはありませんので、実際に領域が確保されることはありませんが、この指定自体はエラーとなりません。
また、ライブラリを使用する場合は、例のようにライブラリファイルも記述する必要があります。libc.aが2つ記述されているのは、libgcc.aから手前に記述したlibc.a内のシンボルを参照できないためで(libgcc.aからlibc.a内の関数をコールすることがあり、リンク時にシンボル参照を解決する必要があるため)、コードが重複して配置されることはありません。

(5) (3)と同様に.bss出力セクションの終了アドレスを示すシンボルを__END_bssという名称で定義しています。'.'で示されるロケーションカウンタ値は、__START_bss + (配置される全.bssセクションの合計サイズ)となります。

(6) .data出力セクションを定義します。__END_bssが開始アドレスとして指定されているため、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配置されます。
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__END_rodata(.rodataセクションの直後)に配置することを指定しています。

(7) アドレス0x008000から.vector出力セクションを定義します。ベクタテーブルがfile1.oの.rodataセクションに記述されているため、配置するファイル(セクション属性)はfile1.o(.rodata)のみを指定します。

(8) .vectorセクションの直後に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、この出力セクションに配置されます。

(9) .textセクションの直後に.rodata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.rodataセクションは、この出力セクションに配置されます。file1.oの.rodataセクションは.vectorセクションに配置したため、ここには指定がありません。

(10) .dataセクションの実データが置かれるLMAの開始アドレスと終了アドレスを示すシンボル、__START_data_lmaと__END_data_lmaを定義しています。これらのシンボルはプログラムソース内でデータをLMAからVMAにコピーするために使用可能です。

このスクリプト例により構成されるメモリマップを図3.8.1に示します。

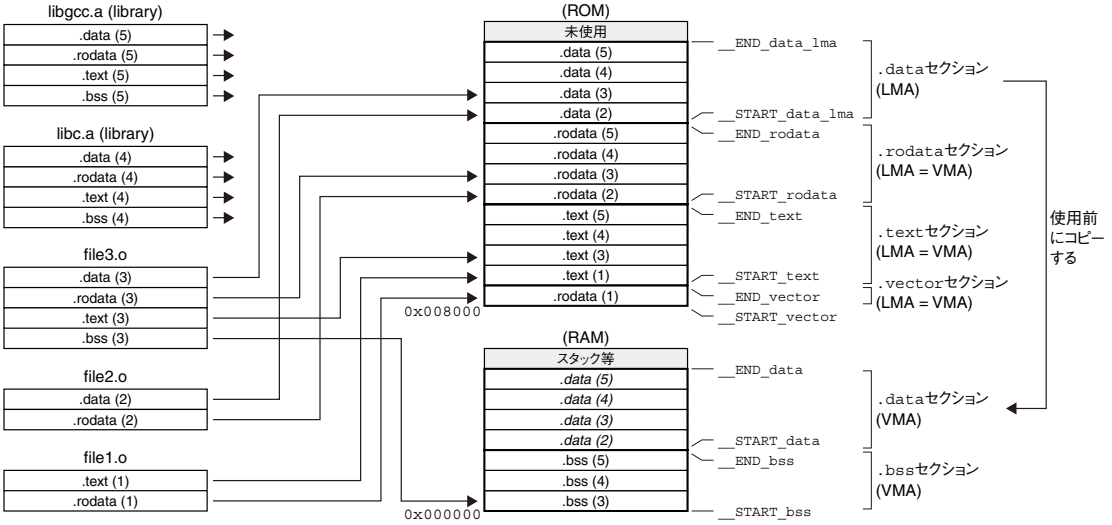


図3.8.1 sample.ldsにより構成されるメモリマップ

このとき、リンカが生成したプログラムは、ROM上のアドレス__START_vectorから__END_data_lmaまでの領域を占めます。同時に、プログラムはRAM上のアドレス__START_bssから__END_dataまでの領域と、スタック(およびヒープ)として割り当てた領域を使用します。
よって、このプログラムおけるROMおよびRAMの使用量は、以下のように求めることができます。

ROM使用量[bytes] = __END_data_lma - __START_vector
RAM使用量[bytes] = __END_data - __START_bss + スタック使用量 (+ヒープ使用量)

__END_data_lmaや__START_vectorといったシンボルの具体的な値は、リンクマップファイルに出力されます。詳細は、"9.4.3 リンクマップ"を参照してください。

S5U1C17001C Manual

4 ソースファイル

4 ソースファイル

この章ではソースファイルを作成する際の規則、文法について解説します。

4.1 ファイル形式とファイル名

ソースファイルは**GNU17 IDE**のエディタ、あるいは汎用のエディタ等で作成してください。

ファイル形式

標準のテキストファイルとしてセーブしてください。

ファイル名

Cソースファイル　　*<ファイル名>*.c
アセンブリソースファイル　*<ファイル名>*.s
*<ファイル名>*は、32文字以内で、以下の英数記号で指定します。
a～z、A～Z、0～9、_(アンダースコア)
なお、S1C17ツールのファイル名には、すべてこの規定が適用されます。

ディレクトリ名

ディレクトリ名もファイル名と同じ英数記号のみが使用可能です。スペースや漢字等は使用しないでください。ディレクトリ名を含むファイル名は64文字以下としてください。

グローバル変数/static変数

グローバル変数/static変数の命名文字数は最大200文字です。
また、グローバル変数/static変数は合わせて32,000個まで使用可能です。

ファイルサイズ

Cソースファイルの最大サイズについては以下を目安としてください。

- 変数、定数、配列等のみのソースファイルは10万行まで使用可能です。
- 実行コード(配列、変数等を含まない)のみのソースファイルは2万行まで使用可能です。ただし、ソースの密度によって使用可能な行数は変動します。
- 変数、定数、配列および実行コードが混在するソースは、上記2項目を参考にしてください。
- コンパイルする環境によっては、上記で挙げた行数は変動します。またリソース不足によりコンパイラが強制終了することがあります。この場合はリソースの豊富な環境でビルトするか、ソースファイルを分割してください(リソースはPCに装着されているRAM容量よりもOSに依存します)。
- Cソースファイルの1行の文字数は、最大512文字です。

アセンブラソースに許可されている行数は最大3万行です。

タブ設定

タブストップは4文字ごとを推奨します。この文字数はIDEがソース表示を行う場合のデフォルトのタブ設定です。

EOF

各ステートメントは必ず改行し、EOFは改行後に付くように(ファイルの最後に独立するように)してください。

4.2 Cソースの文法

本パッケージのCコンパイラ**xgcc**はANSI C準拠のGNU Cコンパイラ(ver. 3.3.2)です。CソースはANSI 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コンパイラ**xgcc**はANSI 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	2	-32768～32767
unsigned int	2	0～65535
long	4	-2147483648～2147483647
unsigned long	4	0～4294967295
pointer	4	0～16777215
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
wchar_t	2	0～65535

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
```

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

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。なお、一部のANSIライブラリ関数は本パッケージでは対応していないため、ANSIライブラリに含まれていません。本パッケージで対応していないANSIライブラリ関数を使用する必要がある場合は、お客さまの責任で関数の実装及び、プロトタイプの宣言を行ってください。ただし、本パッケージで未対応の ANSIライブラリ関数についても、ヘッダファイル内でプロトタイプ宣言のみされているものもありますので、この場合はプロトタイプ宣言をする代わりに、該当するヘッダファイルをインクルードした上で関数の実装を行ってください。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	perror, 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, malloc, calloc, realloc, free, atoi, atol, atof, strtol, strtoul, strtod, abs, labs, div, ldiv, rand, srand, bsearch, qsort	stdlib.h
	setjmp, longjmp	setjmp.h
	time, mktime, gmtime	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, strrchr, 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

エミュレーションライブラリ

ファイル名	関数
libgcc.a (libgccM.a / libgccMD.a / libgccMD2.a)	___subdf3, __adddf3, __addsf3, __ashldi3, __ashlhi3, __ashlsi3, __ashrdi3, __ashrhi3, __ashrsi3, __cmpdi2, __divdf3, __divdi3, __divhi3, __divsf3, __divsi3, __eqdf2, __eqsf2, __extendsfdf2, __fixdfdi, __fixdfsi, __fixsfdi, __fixsfsi, __fixunsdfdi, __fixunsdfsi, __fixunssfdi, __fixunssfsi, __floatdidf, __floatdisf, __floatsidf, __floatsisf, __gedf2, __gesf2, __gtdf2, __gtsf2, __ledf2, __lesf2, __lshrdi3, __lshrhi3, __lshrsi3, __ltdf2, __ltsf2, __moddi3, __modhi3, __modsi3, __muldf3, __muldi3, __mulhi3, __mulsf3, __mulsi3, __nedf2, __negdf2, __negdi2, __negsf2, __nesf2, __subsf3, __truncdfsf2, __ucmpdi2, __udivdi3, __udivhi3, __udivsi3, __umoddi3, __umodhi3, __umodsi3, __cmpsi2, __ucmpsi2

プロトタイプ宣言のみされている関数

関数	対応ヘッダファイル
freopen, tmpfile, tmpnam, remove, rename, fopen, fclose, setbuf, setvbuf, fflush, clearerr, feof, ferror, fseek,fgetpos, fsetpos, ftell, rewind	stdio.h
atexit, getenv, system	stdlib.h
difftime, clock, localtime, asctime, ctime	time.h

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。リンカ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¥
     .long      ADDR_ERR¥n¥
     .long      NMI¥n¥
     .space     4¥n¥
     .long      EINT0¥n¥
     .long      EINT1");
```

例3:
```
BOOT(){
     asm("xld.a  %sp,0x3f00"); /* set SP */
          :
}
```

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:
     ld.a  -[%sp],%r2
     ld    %r2,5
     xld   [int_num],%r2
     ld.a  %r2,[%sp]+
     reti
```

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                                                          4
; boot program                                                    4

#define SP_INI,0x3f00       ; Stack pointer value                6

        .text                                                    2
        .long BOOT          ; BOOT VECTOR                        2
BOOT:                                                            3
        xld.a %sp,SP_INI    ; set SP                             5
        xcall main          ; goto main                          5
        jpr   BOOT          ; infinity loop                      1
  :          :                          :
```

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  %r1,%r2
         ld  %r0,%r1
  ;Error
  BOOT:  ld  %r1,%r2          ld  %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  %r1,%r2
  ;Error
         ld  %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)

S1C17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    %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

S1C17 Familyでは、以下のS1C17コア命令を使用できます。

ld.b	ld.ub	ld	ld.a		
add	add/c	add/nc	add.a	add.a/c	add.a/nc
adc	adc/c	adc/nc	sub	sub/c	sub/nc
sub.a	sub.a/c	sub.a/nc	sbc	sbc/c	sbc/nc
cmp	cmp/c	cmp/nc	cmp.a	cmp.a/c	cmp.a/nc
cmc	cmc/c	cmc/nc			
and	and/c	and/nc	or	or/c	or/nc
xor	xor/c	xor/nc	not	not/c	not/nc
sr	sa	sl	swap		
cv.ab	cv.as	cv.al	cv.la	cv.ls	
jpr	jpr.d	jpa	ipa.d	jrgt	jrgt.d
jrge	jrge.d	jrlt	jrlt.d	jrle	jrle.d
jrugt	jrugt.d	jruge	jruge.d	jrult	jrult.d
jrule	jrule.d	jreq	jreq.d	jrne	jrne.d
call	call.d	calla	calla.d	ret	ret.d
int	intl	reti	reti.d	brk	retd
ext	nop	halt	slp	ei	di
ld.cw	ld.ca	ld.cf			

各命令の詳細についてはS1C17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"、"LD"、"Ld"はすべて"ld"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。

.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
.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

(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:         1label:
_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  %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

S1C17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〜R7	%r0〜%r7または%R0〜%R7

特殊レジスタ	表記
スタックポインタSP	%spまたは%SP
プログラムカウンタPC	%pcまたは%PC

[ ]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: [%r7]　[%r1]+　[%sp+imm7]

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
imm3	3ビット即値	0〜7	0x0〜0x7	0b0〜0b111
imm5	5ビット即値	0〜31	0x0〜0x1f	0b0〜0b1 1111
imm7	7ビット即値	0〜127	0x0〜0x7f	0b0〜0b111 1111
sign7	符号付き7ビット即値	-64〜63	0x0〜0x7f	0b0〜0b111 1111
sign8	符号付き8ビット即値	-128〜127	0x0〜0xff	0b0〜0b1111 1111
sign10	符号付き10ビット即値	-512〜511	0x0〜0x3ff	0b0〜0b11 1111 1111
imm13	13ビット即値	0〜8,191	0x0〜0x1fff	0b0〜0b1 1111 1111 1111
imm16	16ビット即値	0〜65,535	0x0〜0xffff	0b0〜0b1111 1111 1111 1111
sign16	符号付き16ビット即値	-32,768〜32,767	0x0〜0xffff	0b0〜0b1111 1111 1111 1111
imm20	20ビット即値	0〜1,048,575	0x0〜0xfffff	0b0〜0b1111 1111 1111 1111 1111
sign21	符号付き21ビット即値	-1,048,576〜1,048,575	0x0〜0x1fffff	0b0〜0b1 1111 1111 1111 1111 1111
sign23	符号付き23ビット即値	-4194304〜4194303	0x0〜0x7fffff	0b0〜0b111 1111 1111 1111 1111 1111
imm24	24ビット即値	0〜16,777,215	0x0〜0xffffff	0b0〜0b1111 1111 1111 1111 1111 1111
sign24	符号付き24ビット即値	-8,388,608〜8,388,607	0x0〜0xffffff	0b0〜0b1111 1111 1111 1111 1111 1111

(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24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0x00ff00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 = sign10
    ld.a    %rd,LABEL  ← LABEL = sign7
```

アセンブラ**as**では、次のようなディスプレースメント付きの参照も許可します。
LABEL + imm24　LABEL + sign24

```
例: xcall   LABEL+0x10
```

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

```
xadd      xadd.a    xadc      xsub      xsub.a    xsbc      xcmp
xcmp.a    xcmc
sadd      sadd.a    sadc      ssub      ssub.a    ssbc      scmp
scmp.a    scmc
xand      xoor      xxor
sand      soor      sxor
xld       xld.a     xld.b     xld.ub
sld       sld.a     sld.b     sld.ub
xjpr      xjpr.d    xjpa      xjpa.d    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  xcall     xcall.d   xcalla    xcalla.d
sjpr      sjpr.d    sjpa      sjpa.d    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  scall     scall.d   scalla    scalla.d
xld.cw    xld.ca    xld.cf
sld.cw    sld.ca    sld.cf
```

●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[23:10]
                       ; call  LABEL[9:0]

  sld.a  %r1,imm16     ; ext   imm16[15:7]
                       ; ld.a  %r1,imm16[6:0]

  xld.a  %r1,imm24     ; ext   imm24[23:20]
                       ; ext   imm24[19:7]
                       ; ld.a  %r1,imm24[6:0]
```

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.a   %r0,symbol + 0x10        ; R0 ← symbol + 0x10
  xjpa   LABEL + 5               ; LABEL+5のアドレスにジャンプ
```

オペランドを含む拡張命令の詳細については"8.6 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 / マクロ定義
#if-#else-#endif	条件アセンブル

例:
```
#include   "define.h"
#define    NULL    0
#ifdef     TYPE1
           ld      %r0,0
#else
           ld      %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(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) Cコンパイラのデフォルト設定ではアドレス空間が24ビット、`-mshort-offset`オプションのみを指定した場合はアドレス空間が20ビットになります。これに対して`int`型は16ビットのため、`unsigned int/unsigned short`型変数とポインタを含む演算には注意が必要です。
たとえば、以下のコードでは意図した処理を行いません。

```
int* ip_Pt;
unsigned int i = 1;

ip_Pt += (-1)*i;
```

このコードは、`ip_Pt += (-1);`を意図していますが、実際には`ip_Pt += 0xffff;`として処理されます。
アドレス空間が16ビットの場合はこれで正しく処理されますが、24ビット/20ビットの場合には演算結果が不正なアドレスになってしまいます。
デフォルト設定または`-mshort-offset`オプションのみを指定したときにポインタとの演算を行う場合は、`unsigned int/unsigned short`型の変数を避けるか、あるいは次のように定数に接尾語'L' を付けて、`long`型として演算されるようにする必要があります。

```
ip_Pt += (-1L)*i;
```

(5) C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、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、アドレス空間のサイズによって代入可能な整数型が制限されます。

24ビット空間(デフォルト設定)または20ビット空間(`-mshort-offset`のみ指定時)の場合
`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。

16ビット空間の場合(`-mpointer16`指定時)
`short/unsigned short/int/unsigned int`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: 16ビット空間の場合(`-mpointer16`指定時)

1) `short s_Global_Val = (short)boot;`
→ `short`型のグローバル変数`s_Global_Val`の宣言時に、関数名による代入は行えます。

2) `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を行うと、エラーが発生します。

```
error: initializer element is not constant
```

3) `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。

4) `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。

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。

(6) 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(5)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、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。
ただし、アドレス空間のサイズとグローバル/ローカル変数またはグローバル/ローカル配列の型によってはワーニングが発生します。

24ビット空間(デフォルト設定)または20ビット空間(-mshort-offsetのみ指定時)の場合
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。

16ビット空間の場合(-mpointer16指定時)
short/unsigned short/int/unsigned int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: 16ビット空間の場合(-mpointer16指定時)

```
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) short s_Global_Val = (short)fp_Pt;
→ グローバル変数s_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
```

2) 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行えます。

3) 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) short s_Global_Val;

```
void sub()
{
    s_Global_Val = (short)fp_Pt;
}
```

→ 宣言時以外であれば、short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。

(7) 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 l_Val=0x12345678,l_Val_2;

int main()
   {
        l_Val_2 = sub();      // l_Val_2 の値が0x5678になってしまいます。
        return 0;
   }

long sub()
   {
        long l_wk;

        l_wk = l_Val;
        return l_wk;
   }
```

(8) char 以外のポインタを使用して奇数番地のメモリへのリード/ライトを行わないでください。
この場合、アドレス不整例外が発生します。
例:
```
int *ip_Pt;

int sub()
{
   ip_Pt = (int *)0x3;
   (*ip_Pt) = 0x2;

   return 0;                  // ここでアドレス不整例外が発生します。
}
```

(9) 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/-O3)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れらます。

- 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。

S5U1C17001C Manual

5 GNU17 IDE

5 GNU17 IDE

この章では、**GNU17 IDE**の機能と操作方法を説明します。

5.1 概要

5.1.1 特長

GNU17 IDE(以下**IDE**)は、S1C17 Family Cコンパイラパッケージ(S5U1C17001C)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およびアセンブラをサポート**
  GNU準拠の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。
- **未使用関数の検出機能(C言語のみ)**
  ビルド処理時に静的コード解析ツールであるSplintを呼び出し、プロジェクト内で一度も使用されていないC言語の関数を検出することができます。
- **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。また、アプリケーション(*.elf/*.psa)かライブラリ(*.a)のどちらのmakeファイルを作成するか選択することが可能です。
- **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、デバッガ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。
- **デバッガgdbを呼び出すためのランチャ機能**
  ビルド処理終了後、デバッガ**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GNU17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

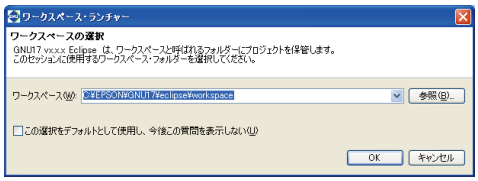
IDEの起動方法は以下のとおりです。

(1) 

c:¥EPSON¥gnu17¥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] > [GNU17] > [GNU17 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。



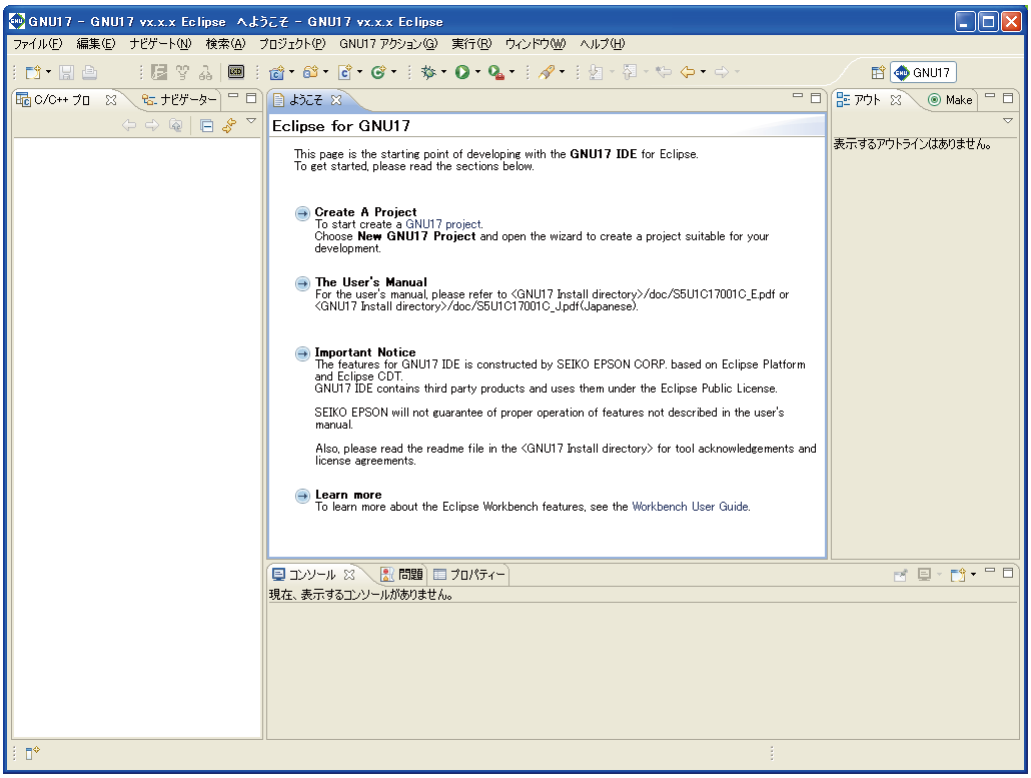
c:¥EPSON¥gnu17¥eclipse¥workspaceがデフォルトディレクトリとして表示されますが、任意のディレクトリを選択または新規作成してワークスペースに設定可能です。[ワークスペース:]コンボボックスにディレクトリ名を入力するか、[参照...]ボタンで表示されるディレクトリ選択ダイアログで選択してください。

[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。

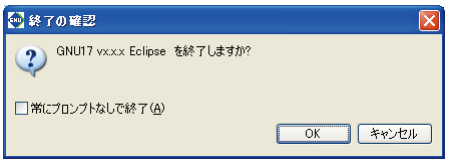
IDEのウィンドウが表示されます。



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。

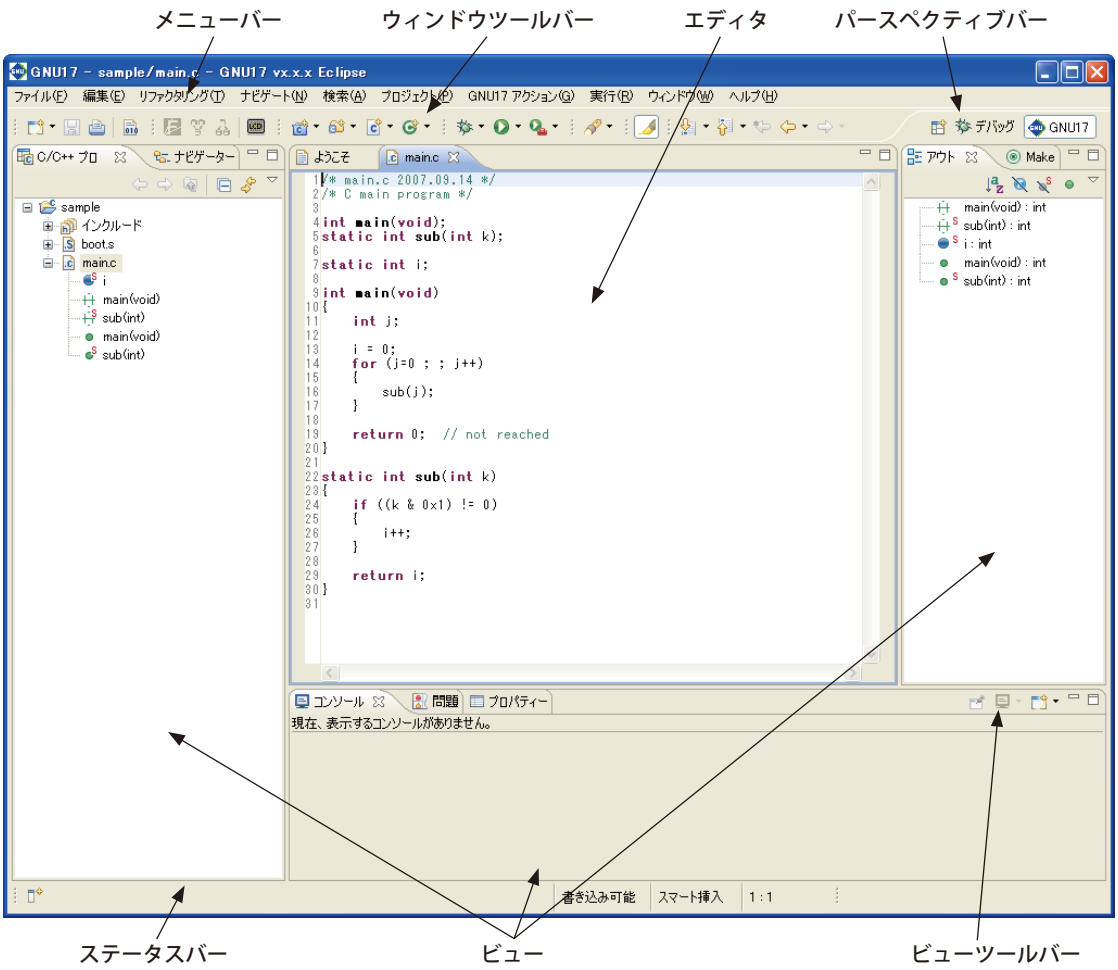
ウィンドウの(閉じる)ボタンでも終了可能です。ただし、この場合は次のダイアログが表示されますので、終了する場合は[OK]ボタンを、終了を取り消す場合は[キャンセル]ボタンをクリックします。



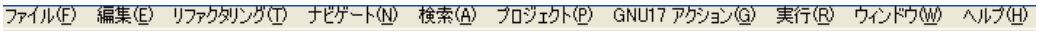
[常にプロンプトなしで終了]をチェックしておくと、以後はこのダイアログが表示されなくなります。

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。

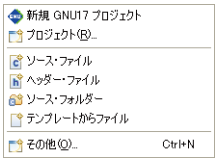


5.3.1 メニューバー



●[ファイル]メニュー

新規 (Alt+Shift+N)



新規 GNU17 プロジェクト
GNU17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[GNU17 プロジェクト]内の[新規 GNU17 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。

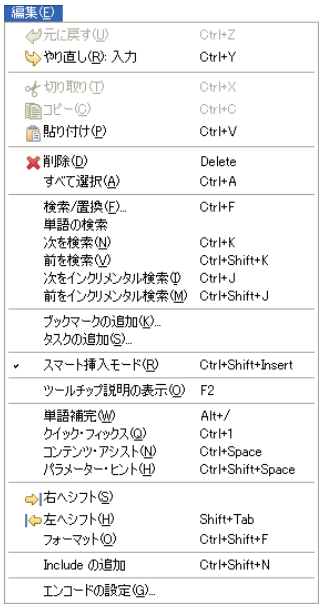
インポート...
既存のプロジェクトやソースファイルを現在のワークスペースやプロジェクトに追加するウィザードが起動します。

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。

プロパティー (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



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。

単語の検索
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。

ツールチップ説明の表示 (F2)
ツールチップ表示中に[F2]キーを押すと、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。

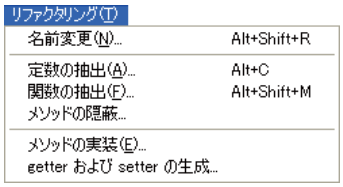
右へシフト
行の先頭をタブ1個分右にシフトします。

左へシフト (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。

●[リファクタリング]メニュー



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。

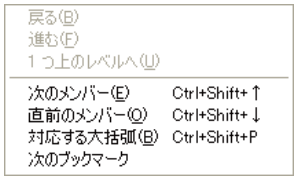
定数の抽出 (Alt+C)
ソースファイルの定数を変数に切り出します。

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

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
[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 つ上のレベルへ
[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+上へ)
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)。

次のブックマーク
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。

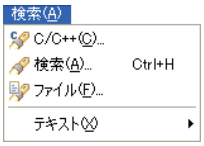
最後の編集位置 (Ctrl+Q)
エディタ内の最後に編集した位置にジャンプします。

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

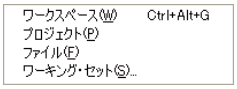


検索... (Ctrl+H)
ファイル検索、C検索を行う[検索]ダイアログを表示します。

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。)

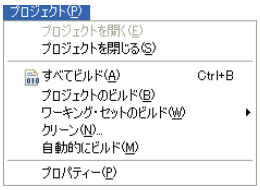
C/C++...
指定の文字列を含むCソースを検索します。([検索]ダイアログのC検索ページを表示します。)

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。

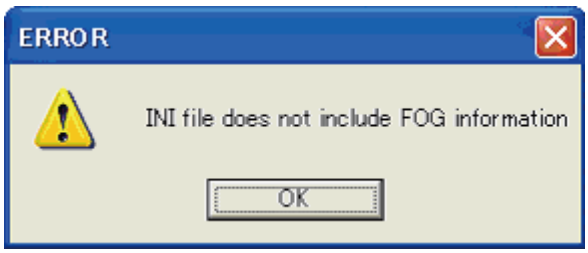
プロパティー
[C/C++ プロジェクト]または[ナビゲーター]ビュー上で選択されているプロジェクトのプロパティを表示/編集する[プロパティー]ダイアログを表示します。

●[GNU17 アクション]メニュー

WinFog17を起動

FDCファイル(ファンクションオプションドキュメント)を作成するwinfog17.exeを起動します。winfog17については"12.9 winfog17.exe"を参照してください。

[WinFog17を起動]ボタンをクリックした時、ファンクションオプションの選択が不要なCPUは、[INI file does not include FOG informaion]ダイアログが表示されます。



その場合は、FDCファイルを作成する必要はありません。ダイアログを閉じ、Winfog17も終了してください。

WinMdc17でパック

winmdc17.exeを起動し、PAファイル(提出用データ)(<プロジェクト名>.PA)を生成します。パックについては"11.2.3 winmdc17 によるpaファイル(提出用データ)の作成"を参照してください。

WinMdc17でアンパック

winmdc17.exeを起動し、PAファイル(<プロジェクト名>.PA)をアンパックします。アンパックについては"11.2.4 paファイル(提出用データ)の分離方法"を参照してください。

LcdUtilityを起動

LCDユーティリティ(LcdUtil17.exe)を起動します。

Flash ROM書き込み

プログラムをデバッグターゲットのFlash ROMへ書き込みます。

Flashセキュリティの解除

Flash ROMのアクセス制限を解除します。

●[実行]メニュー

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。

ヒストリーの実行

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。

ヒストリーのデバッグ

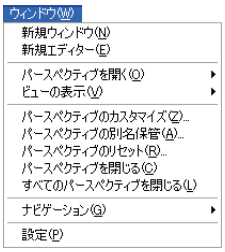
サブメニューに最近起動したデバッグ構成のショートカットを表示します。

デバッグ
サポートしていません。

デバッグの構成...
デバッガ**gdb**の起動構成ダイアログを開きます。

外部ツール
サポートしていません。

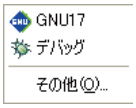
●[ウィンドウ]メニュー



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

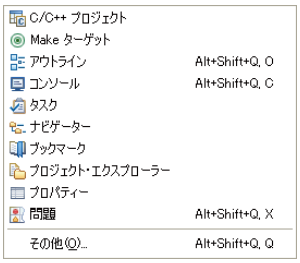


GNU17
GNU17パースペクティブを開きます。

デバッグ
デバッガ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。

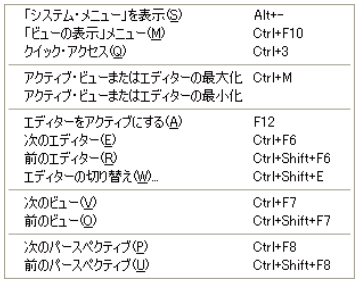
パースペクティブの別名保管...
現在のパースペクティブの設定を別名で保存します。

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 (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。

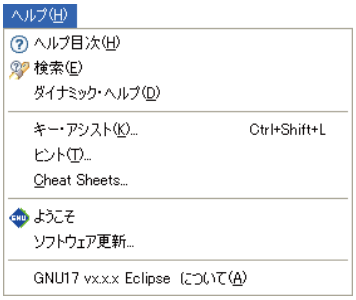
前のビュー (Ctrl+Shift+F7)
アクティブにするビュー(デフォルト: 使用履歴で現在のビューの前に開いたビュー)を選択します。

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。通常は操作しないでください。

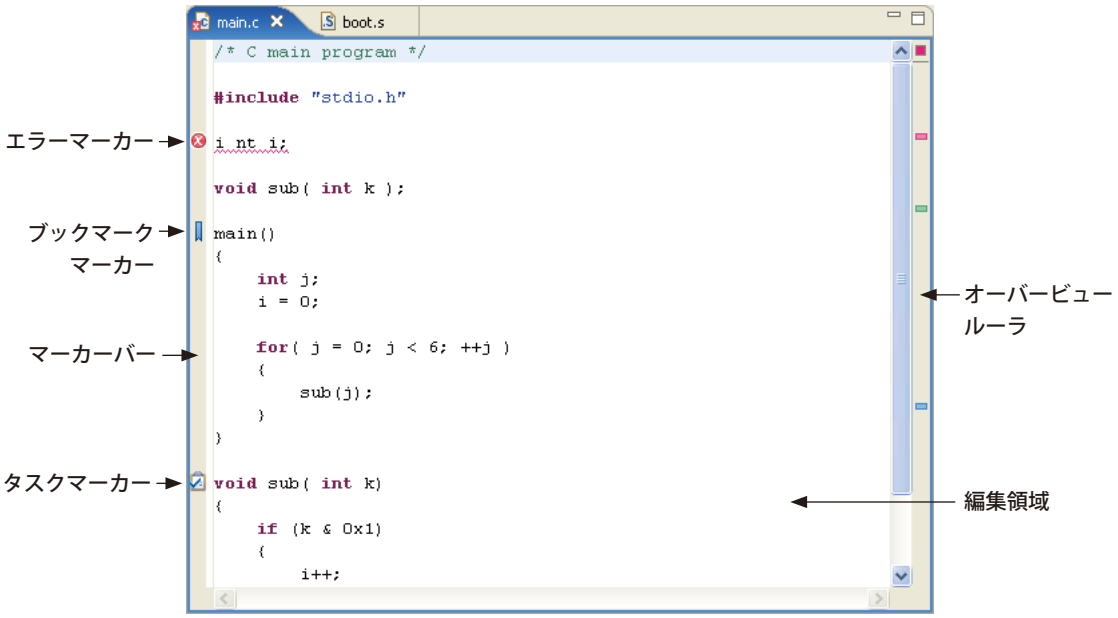
Eclipse for GNU17 V*x.x*について
IDEのバージョンやプラグインの詳細などを表示します。

5.3.2 ウィンドウツールバー

ここには、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]
WinFog17を起動	= [GNU17 アクション]>[WinFog17を起動]
WinMdc17でパック	= [GNU17 アクション]>[WinMdc17でパック]
WinMdc17でアンパック	= [GNU17 アクション]>[WinMdc17でアンパック]
LcdUtilityを起動	= [GNU17 アクション]>[LcdUtilityを起動]
新規 C/C++ プロジェクト	= [ファイル]>[新規]>[新規 GNU17 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++ クラス	未サポート
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。

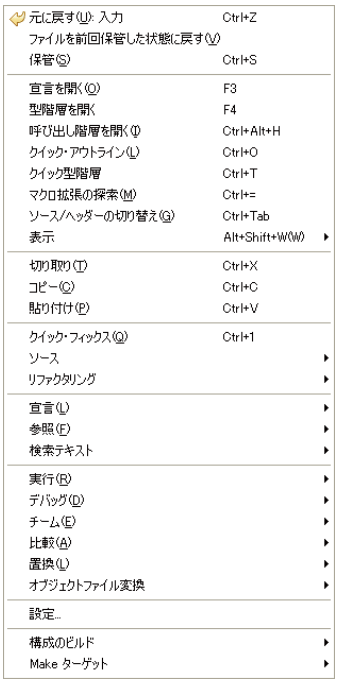
エディタ領域左端のマーカーバーは、エラー発生行、ブックマークやタスクを設定した行などを示すマーカーを表示します。マーカーにマウスポインタを重ねると、エラーの内容やブックマーク名、タスクの説明などが現れます。

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 = [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)。

右へシフト = [編集]>[右へシフト]

左へシフト = [編集]>[左へシフト]

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。

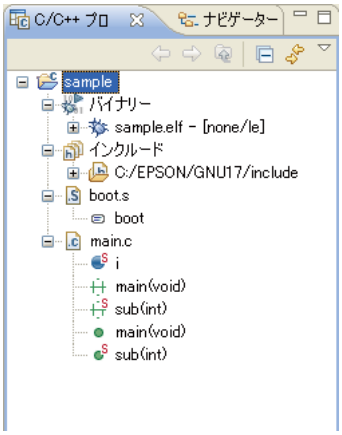
構成のビルド

[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: C/C++プロジェクト ビューのツリー表示では、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おりません。

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

●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。

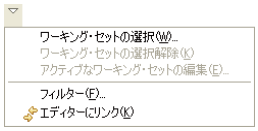
進む
ビュー内の表示を履歴に沿って一つ後に進めます。

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随します。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随させます。

●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...
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。

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、SAファイル(ROMデータ)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。

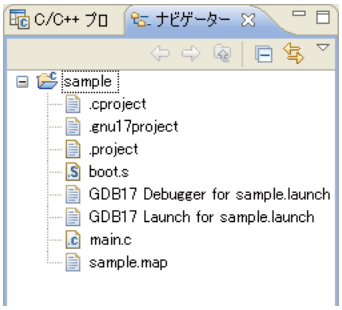
バイナリファイル生成(elfファイル選択時)
選択したelf形式オブジェクトファイルからデバッグ情報等を削除し、バイナリファイルを生成します。呼び出されるコマンドは、`objcopy -I elf32-little -O binary <`*ファイル名*`>.elf <`*ファイル名*`>.bin`です。

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。

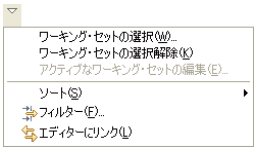
進む
ビュー内の表示を履歴に沿って一つ後に進めます。

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随します。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。

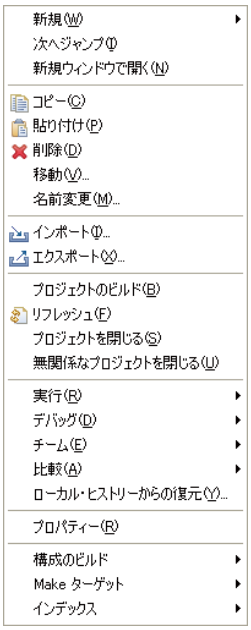
タイプ順
ビューの表示をファイルの種類ごとのアルファベット順に並べます。

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随させます。

●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。

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、SAファイル(ROMデータ)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。

※ Sレコードファイルは<プロジェクト名>.psaの名称で、プロジェクトをビルドした時に生成されます。通常、この操作によるSレコードファイル生成は不要です。

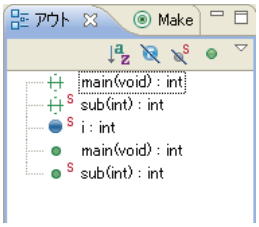
バイナリファイル生成(elfファイル選択時)
選択したelf形式オブジェクトファイルからデバッグ情報等を削除し、バイナリファイルを生成します。呼び出されるコマンドは、`objcopy -I elf32-little -O binary <`*ファイル名*`>.elf <`*ファイル名*`>.bin` です。

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。

5.3.6 [アウトライン]ビュー



エディタで表示中のCソース内に記述されている関数や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

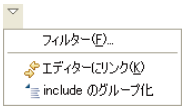
ソート
このボタンを押し込んだ状態にしておくことで、表示内容がアルファベット順に並びます。通常は、エディタ内の出現順に表示されます。

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。

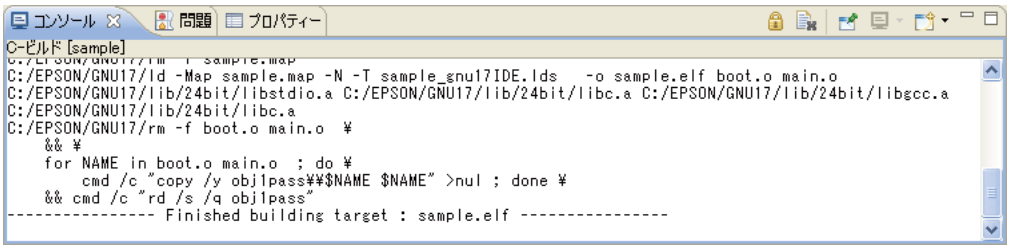
リファクタリング

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GNU17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。

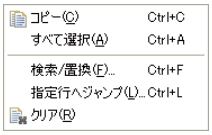
終了
デバッガ起動時などのコンソールに表示されるボタンです。このボタンをクリックすると、コンソールに対応するツール(デバッガ)が処理を中止して閉じます。コンソールは閉じません。ツール側の操作で終了した場合もコンソールは閉じません。

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]

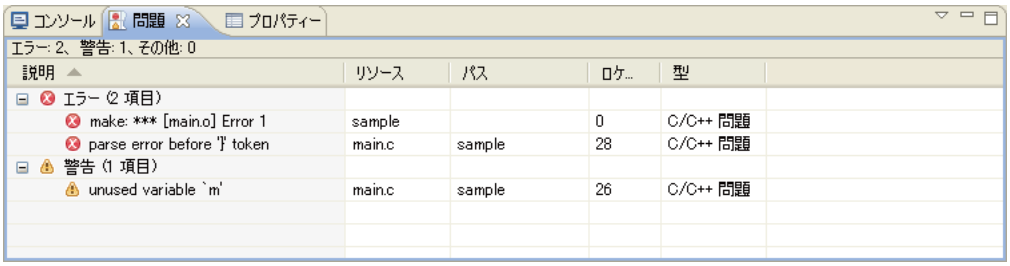
検索/置換... = [編集]>[検索/置換...]

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。

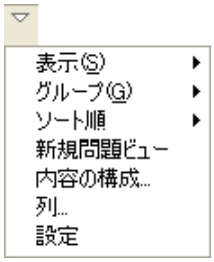
(特に説明のないメニューコマンドについては、メニューバーの説明を参照してください。)

5.3.8 [問題]ビュー



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
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 "5.10.5 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
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]

削除
選択したエラーに対応するエディタ上のエラーマーカーを削除します。

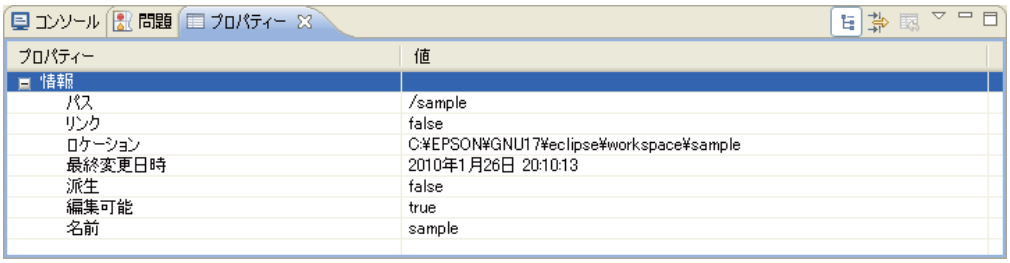
プロパティー
ビュー内で選択されているエラーの情報を表示します。

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。

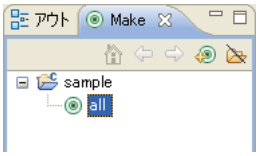
（特に説明のないメニューコマンドについては、メニューバーの説明を参照してください。）

5.3.9 [プロパティー]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。

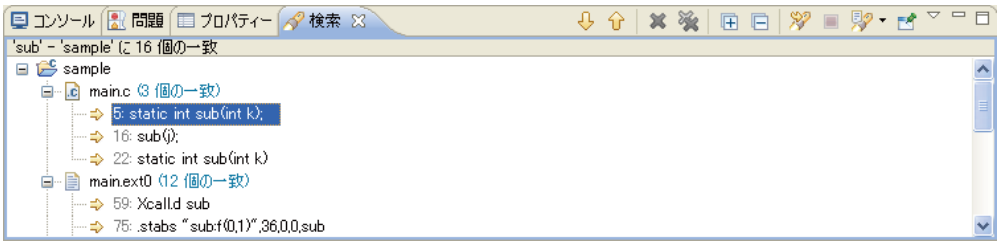
Make ターゲットの編集
選択したターゲットを編集します。

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。

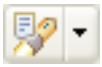
すべての一致を除去
ビュー内の検索結果をすべて削除します。

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。

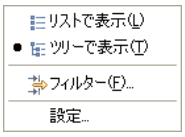
現行検索のキャンセル
進行中の検索動作をキャンセルします。



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

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++ エディター (アセンブリー・エディター)
: Cエディタ(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エディタ(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エディタ(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、SAファイル(ROMデータ)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。

※Sレコードファイルは<プロジェクト名>.psaの名称で、プロジェクトをビルドした時に生成されます。

通常、この操作によるSレコードファイル生成は不要です。

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

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。

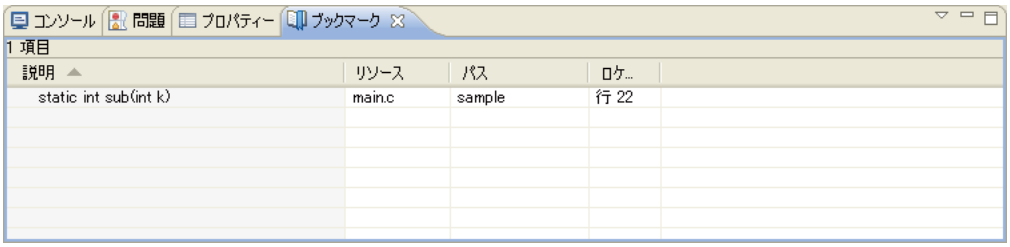
構成のビルド

[Make ターゲットのビルド]で選択するターゲットを定義します。

Make ターゲットのビルド

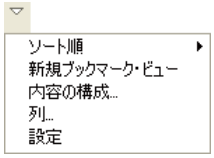
ターゲットを選択して**make.exe**を実行します。

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。

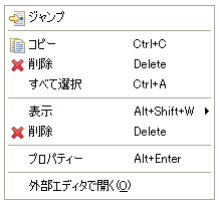
内容の構成...
フィルタ設定を作成、編集します。
ここで条件を設定し、表示させるブックマークの対象や数を制限します。
フィルタ設定については"5.10.5 フィルター "を参照してください。

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]

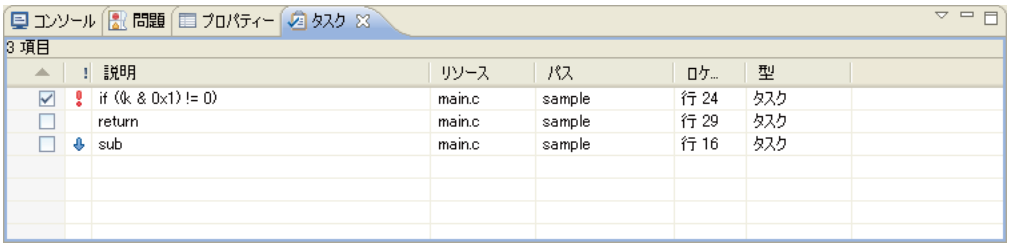
削除 = [編集]>[削除]

すべて選択 = [編集]>[すべて選択]

プロパティー
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。

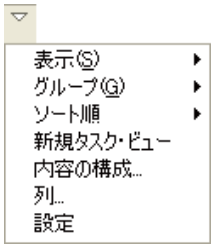
(特に説明のないメニューコマンドについては、メニューバーの説明を参照してください。)

5.3.13 [タスク]ビュー



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
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。

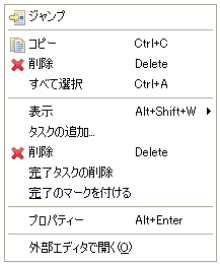
内容の構成...
フィルタ設定を作成、編集します。
ここで条件を設定し、表示させるタスクの対象や数を制限します。フィルタ設定については"5.10.5 フィルター "を参照してください。

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。



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。

プロパティー

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

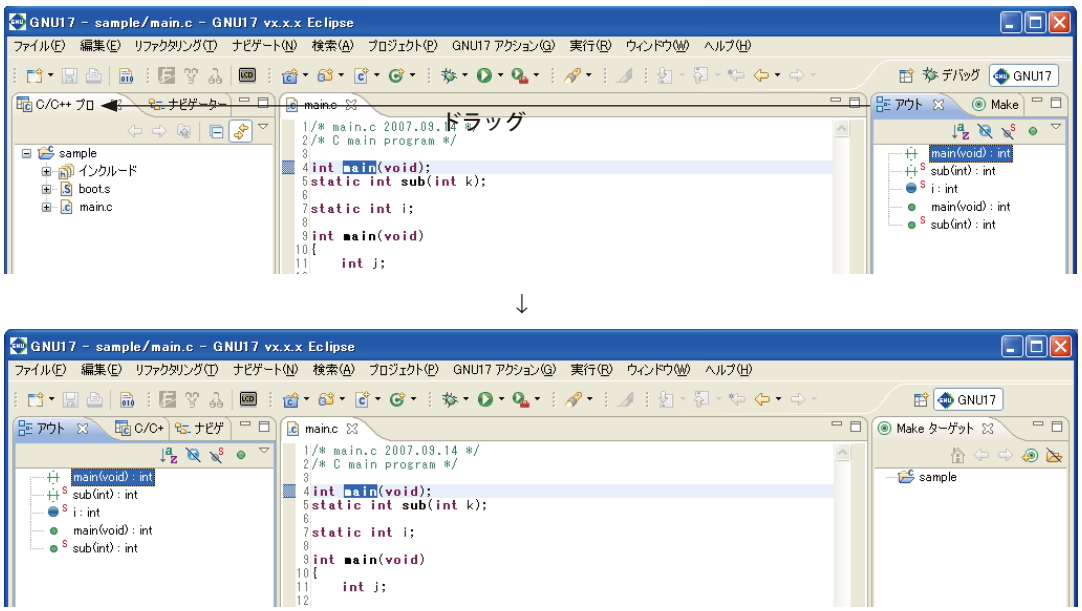
ここでは、すべてのビューに関わるオープン/クローズやレイアウトを変更する操作について説明します。

●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、後述のパースペク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

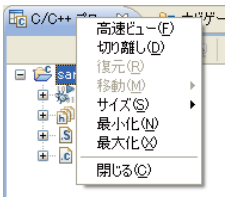
ビューのタブをダブルクリックすると、そのビューを含むペインがIDEウィンドウのサイズまで拡大され、その他のビューは下に隠れます。最大化されたビューのタブをダブルクリックすると、ビューが元のサイズに戻ります。また、各ペインの右上には最大化ボタンがあり、このボタンのクリックでも同様の操作が行えます。最小化ボタンは最大化されたビューを元のサイズに戻します。通常の表示で最小化ボタンをクリックすると、ビューはタブのみの表示となります。


最大化ボタン

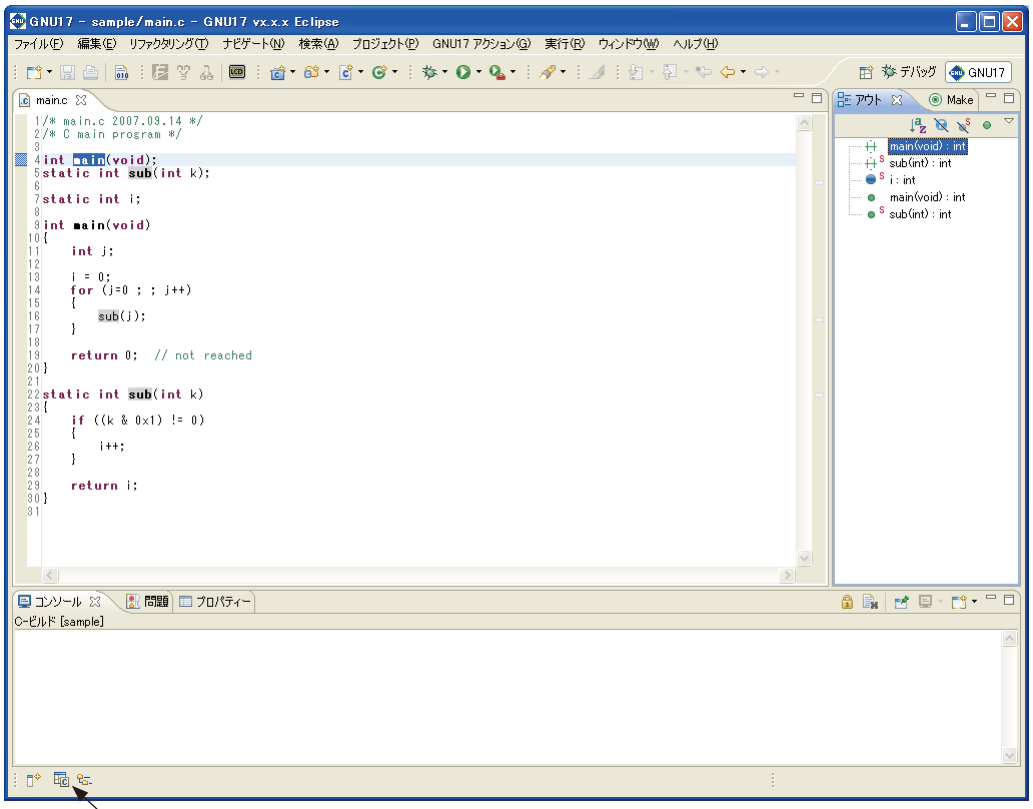
最小化ボタン

●高速ビュー

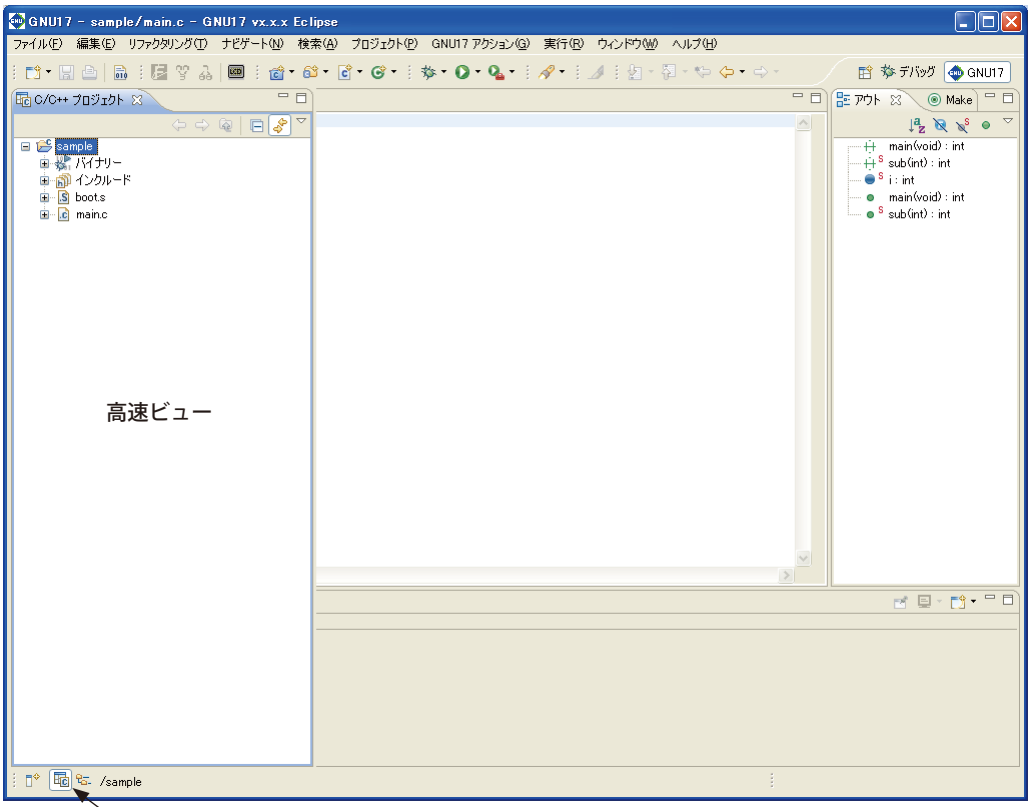
たとえばソースの編集時など、エディタ領域を広げたいことがあります。エディタ領域を最大化する方法もありますが、これとは別に高速ビューを使用する方法があります。これは、とりあえず不要なビューをアイコンにしてウィンドウ左下の高速ビューエリアに置き、そのアイコンをクリックすることでビューが拡大表示される機能です。この方法では、他のビューすべてを隠してしまうことがありません。



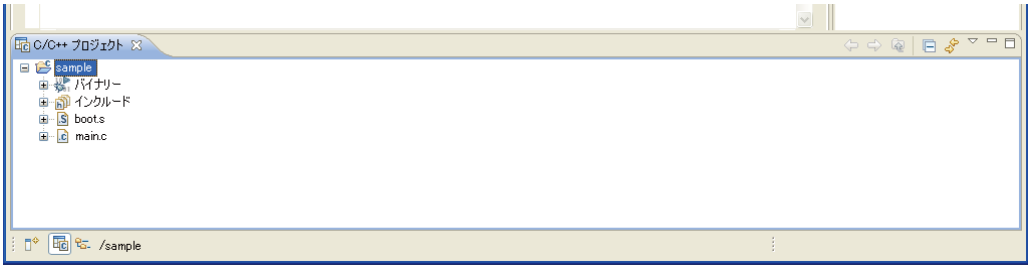
ビューを高速ビューにするには、ビューのタブを右クリックして表示されるコンテキストメニューから[高速ビュー]を選択します。タブを高速ビューエリアまでドラッグ&ドロップして高速ビュー化することも可能です。
たとえば、[C/C++ プロジェクト]と[ナビゲーター]タブのコンテキストメニューから[高速ビュー]を選択してください。それぞれのビューが高速ビュー化され、ビューのアイコンがウィンドウ左下の高速ビューエリアに現れます。左ペイン内のビューが閉じたため、その分エディタや下のビューエリアが拡大されます。

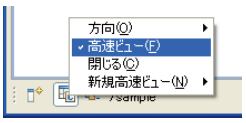


高速ビューを開くには、高速ビューエリアのアイコンをクリックします。閉じるには、それ以外のビューをクリックするか、高速ビューエリアのアイコンまたは開いた高速ビューの最小化ボタンをクリックします。



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。



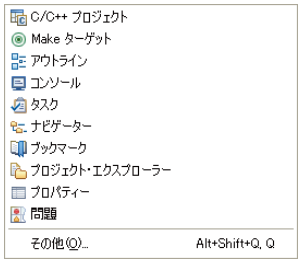


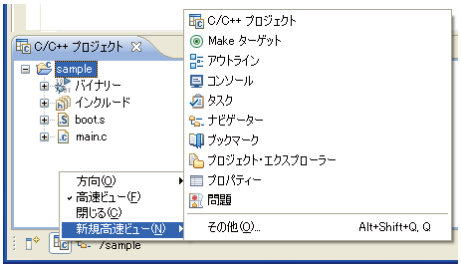
高速ビューを通常のビューに戻すには、高速ビューアイコンのコンテキストメニューから[高速ビュー]を選択します。

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]ボタン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"GNU17"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"GNU17"は、現在GNU17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"GNU17"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`、`.gnu17project`、`.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。

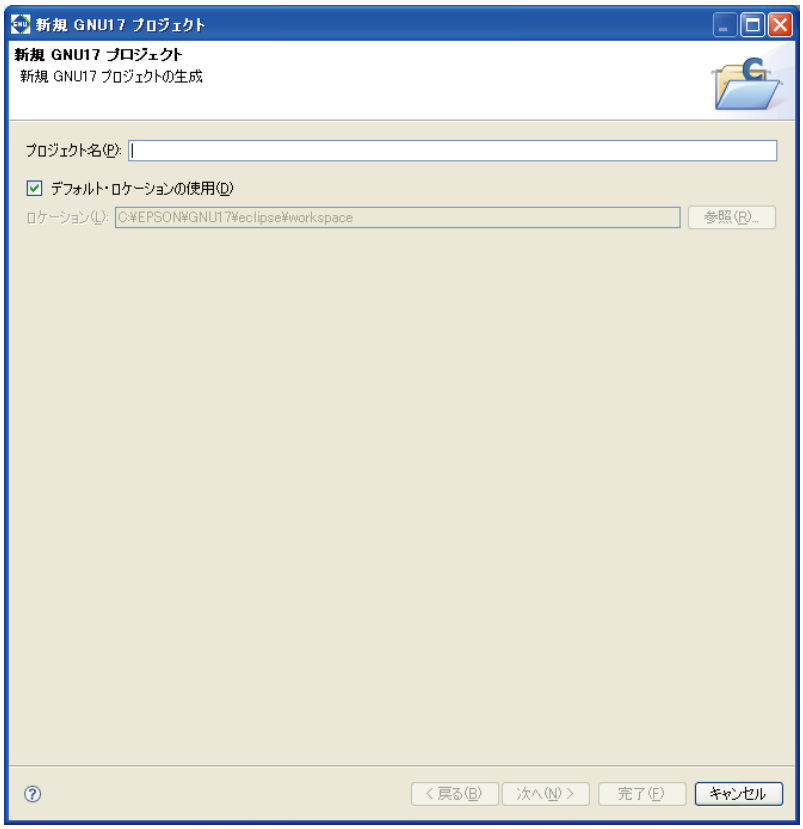
5.4.2 新規プロジェクトの作成

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 GNU17 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 GNU17 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 GNU17 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 GNU17 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。
プロジェクト関連の設定画面に移行します。

(5) [生成プログラム]から生成するプログラムを選択します。

アプケーション(.elf/.psa)：アプリケーション(C17用実行ファイルを作成)開発を行う場合
ライブラリ(.a)：　　ライブラリ(C17用ライブラリファイルを作成)開発を行う場合

ライブラリの場合、(7)[メモリモデル]の設定以外は不要のため表示されません。

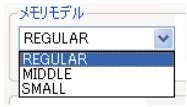
(6) [ターゲット CPU 機種]コンボボックスからターゲットプロセッサの種類を選択します。

リストに表示される機種は、新機種のリリース等による設定ファイルの変更により追加/削除されます。

※ ターゲットCPUは、後からの変更も可能です("5.7.1 GNU17 一般設定の設定"参照)。

※ S1C17選択時は、ビルド時elf実行ファイルまでしか生成されません。

(7) [メモリモデル]コンボボックスからメモリモデルを選択します。



REGULAR: 24ビット(16Mバイト空間を使用)
MIDDLE: 20ビット(1Mバイト空間を使用)
SMALL: 16ビット(64Kバイト空間を使用)

[生成プログラム]でライブラリを選択した場合、MIDDLEは1MB空間のアプリケーションで動作します。メモリモデルは、後からの変更も可能です("5.7.1 GNU17 一般設定の設定"参照)。

(8) コプロセッサ用ライブラリをリンクするかどうか選択します(選択可能な機種のみ)。
プログラムで乗算や除算を行う場合にコプロセッサ命令を使用することができます。コプロセッサ使用に際しては、ベクタテーブルの割り込みベクタ3にemu_copro_process()関数を指定する必要があります。(7.2.6コプロセッサ命令対応参照)
[コプロセッサ用ライブラリを使用する]のチェックボックスをON/OFFします。
ON： プロジェクト作成時にコプロセッサライブラリlibgccMD.a(乗除算用)もしくはlibgccM.a(乗算用)をリンクする設定を追加します。
ONの場合、[コプロセッサ用ライブラリの種類]コンボボックスからリンクするライブラリの種類を選択します(機種によっては選択肢が1つしかないものもあります)。
OFF：プロジェクト作成時に通常のエミュレーションライブラリlibgcc.aをリンクする設定を追加します。
選択できない機種の場合、プロジェクト作成時に通常のエミュレーションライブラリlibgcc.aをリンクする設定を追加します。

(9) .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("5.7.9 リンカスクリプトの編集"参照)。

(10) ブートベクタアドレスを指定します。
[ターゲット CPU 機種]の機種によって設定アドレスが変わります(例　008000や020000など)。ここに設定した値は、**IDE**が自動生成するデバッガ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、およびリンカスクリプト内の.vectorセクションのVMAとして使用されます。

(11) 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選択したターゲットCPUに依存します。デフォルト設定値については、各機種のテクニカルマニュアルを参照してください。
ここに設定した値は、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。

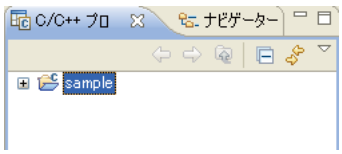
例)　ブートルーチン内で以下のように記述できます。

```
boot:
xld.a %sp, __START_stack
```

(12) [完了]ボタンをクリックします。
[新規 GNU17 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


sample ←開いているプロジェクト


sample B←閉じているプロジェクト

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否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。

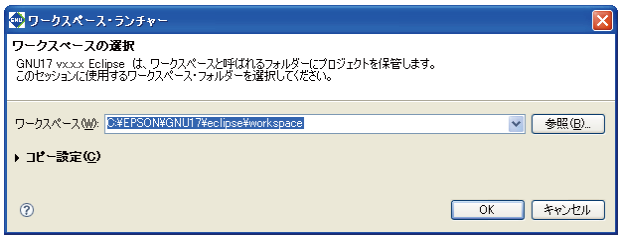
この操作は[C/C++ プロジェクト]または[ナビゲーター]ビューに表示され、閉じているプロジェクトにのみ有効です。現在のワークスペース以外のディレクトリにあるプロジェクトを開くには、ワークスペースをプロジェクトの置かれたディレクトリに切り換えます(5.4.4節参照)。または、現在のワークスペースに既存のプロジェクトをインポートしてください(5.4.5節参照)。

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。

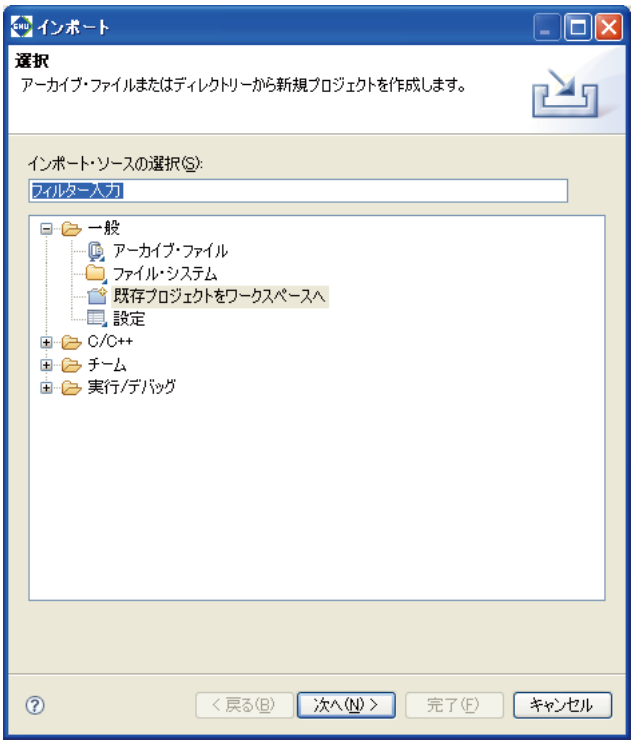
5.4.5 既存プロジェクトのインポート

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(V1.4.0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。

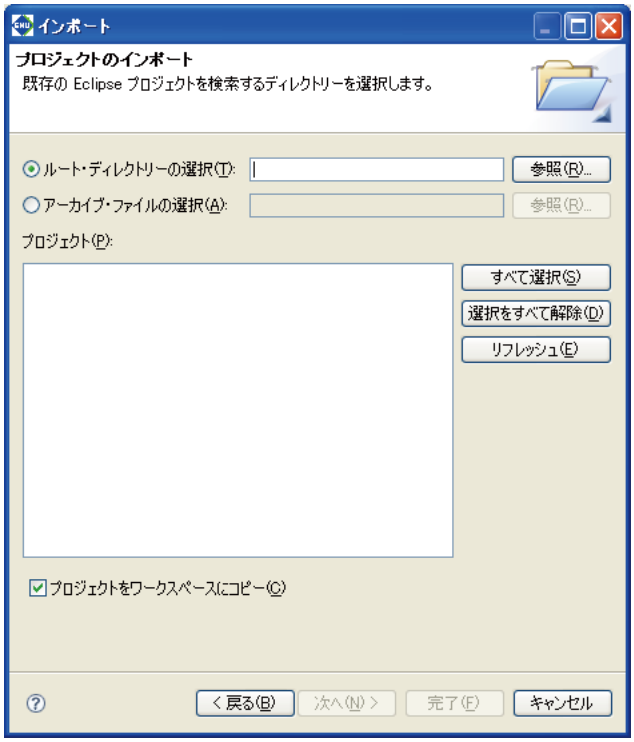
(1)以下のいずれかの操作を行います。

- [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。

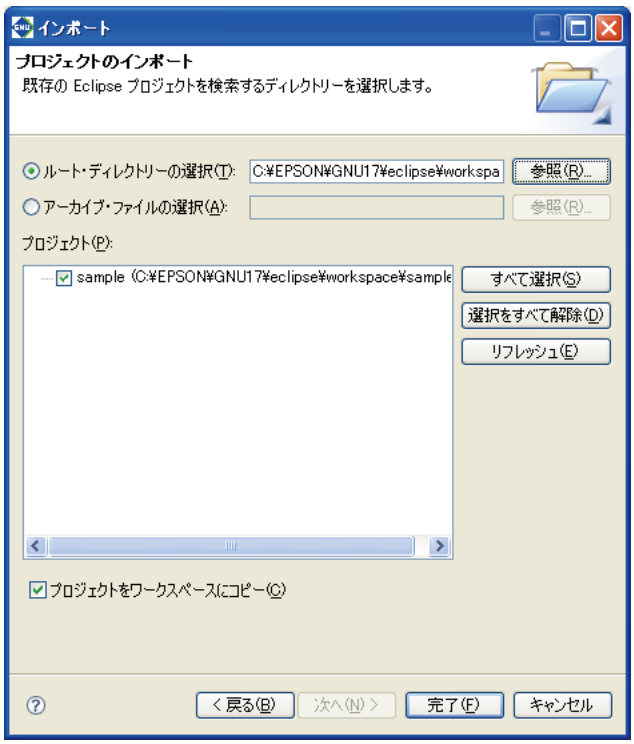
[インポート]ウィザードが起動します。



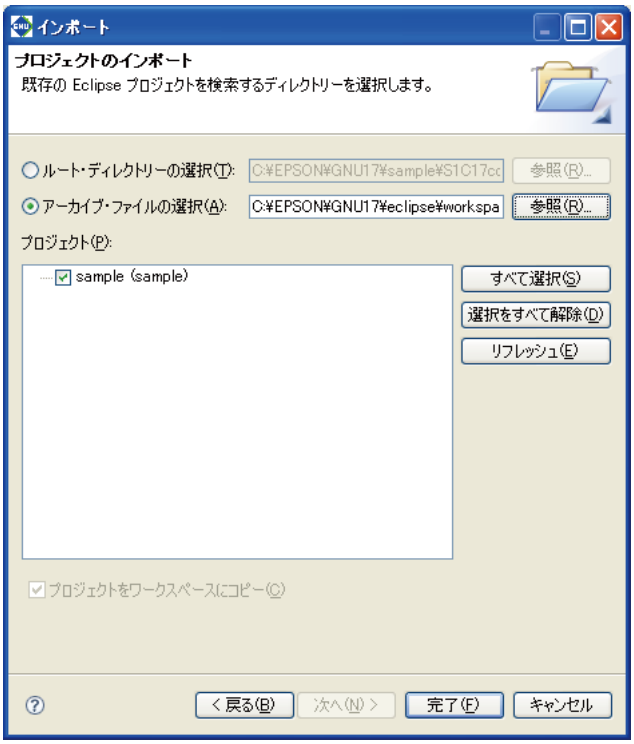
(2)一覧から[既存プロジェクトをワークスペースへ]を選択し、[次へ>]ボタンをクリックします。



(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) [完了]ボタンをクリックします。

※ [完了]ボタンクリック後、インポートするプロジェクトのターゲットCPUに対する機種別情報ファイルが存在しない場合、エラーダイアログ表示後、ターゲットCPU選択画面が表示されます。

[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。

プロジェクトで設定されているターゲットCPUがES-Sim17に対応している場合、essim17_user.iniファイルが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/.gnu17project/)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

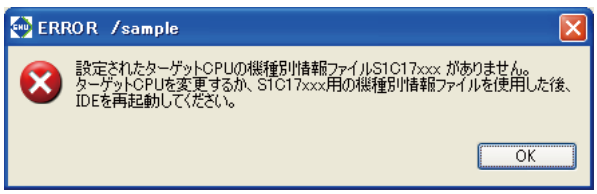
プロジェクトフォルダ内にすべてのリソースが集約されていれば、どこにコピーしても問題はありません。コピー先でも修正なしにビルドが行えます。
オリジナルのプロジェクトフォルダ内から参照している外部ファイルがあった場合、同じディレクトリ構造の環境であれば修正等は必要ありません。チュートリアル2でも説明したように、makeファイルなどをIDEの自動生成ではなく、外部に用意している場合などは、パスの記述に修正が必要になる場合があります。
標準ライブラリやインクルードディレクトリは、ツールをC:¥EPSON¥gnu17など、同じディレクトリにインストールしてあれば問題はありません。ツールディレクトリが異なる環境の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[プロパティー]ダイアログボックスの[GNU17 ビルドオプション]で変更してください。

● ターゲットCPUの機種別情報ファイルが存在しない場合

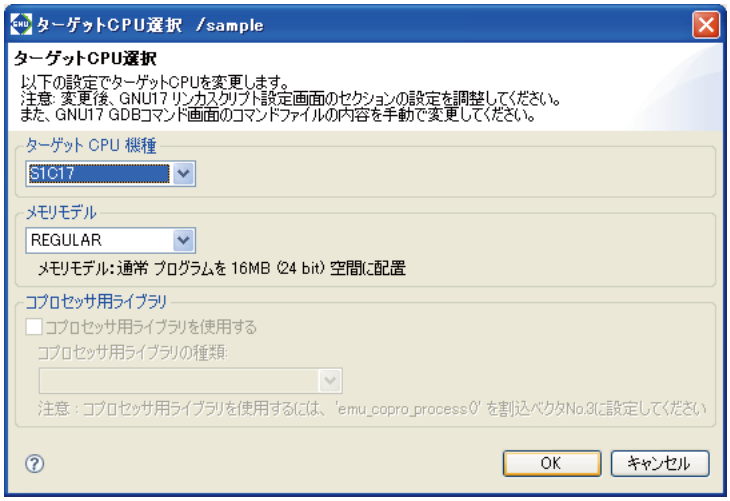
プロジェクトをインポートする時、プロジェクトに設定されたターゲットCPUの機種別情報ファイルが存在しない場合について説明します。
機種別情報ファイルはターゲットCPUに関する設定ファイルです。この機種別情報ファイルが無い場合、以下の手順でプロジェクトのターゲットCPUを変更するか、機種別情報ファイルを入手する必要があります。

[ルート・ディレクトリーの選択]によるインポートの場合

(1) プロジェクトのインポート時にエラーダイアログが表示されます。



(2) エラーダイアログの内容を確認して[OK]ボタンをクリックしてください。(閉じる)ボタンをクリックでも同様です。
[ターゲットCPU選択]ダイアログが表示されます。



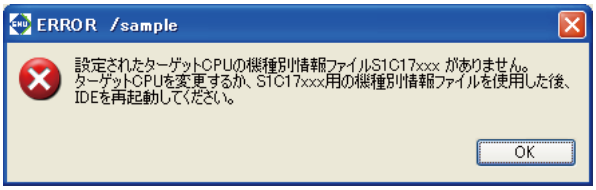
(3) [ターゲットCPU機種]コンボボックスでプロジェクトに合ったターゲットCPUを選択してください。また、[メモリモデル]と[コプロセッサ用ライブラリ]についてもプロジェクトに合わせて選択してください。

(4) [OK]ボタンをクリックしてください。
既存プロジェクトは、ターゲットCPUを変更してインポートされます。
ターゲットCPUを変更してインポートしたくない場合は、[ターゲットCPU選択]ダイアログで[キャンセル]ボタンをクリックしてください。インポート処理は中止されます。
既存の設定のままインポートしたい場合は、対応する機種別情報ファイルを¥gnu17¥ mcu_model以下に追加してください。その後IDEを再起動してからインポートをしてください。
機種別情報ファイルについてはセイコーエプソンのユーザサイトから入手するか、弊社営業窓口にお問い合わせください。
複数のプロジェクトを指定しインポートする際も、一度でも[キャンセル]ボタンをクリックすると、指定された全てのプロジェクトのインポート処理が中止されます。

[アーカイブ・ファイルの選択]によるインポートの場合

[ルート・ディレクトリーの選択]によるインポートの場合と異なり、[ターゲットCPU選択]ダイアログでのターゲットCPU変更はできません。プロジェクトに設定されたターゲットCPUの機種別情報ファイルが存在しない場合でも、プロジェクトを既存の設定のままインポートします。

(1) プロジェクトのインポート処理後にエラーダイアログが表示されます。

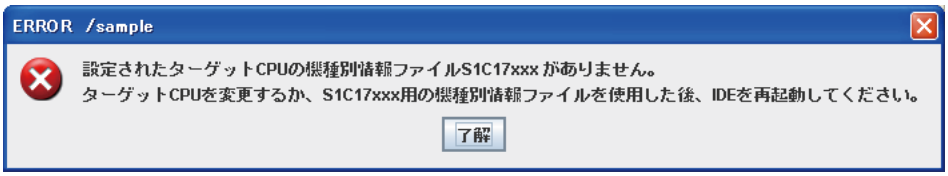


(2) エラーダイアログの内容を確認して[OK]ボタンをクリックしてください。 (閉じる)ボタンをクリックでも同様です。

(3) 以下のいずれかの処理を行ってください。
- [プロパティー]ダイアログの[GNU17 一般設定]でターゲットCPUを変更します。詳細は"5.7.1 GNU17 一般設定の設定"を参照してください。
既存プロジェクトの設定に対応した機種別情報ファイルを¥gnu17¥ mcu_model以下に追加してIDEを再起動します。機種別情報ファイルについてはセイコーエプソンのユーザサイトから入手するか、弊社営業窓口にお問い合わせしてください。

注：・本エラーダイアログは、プロジェクトに設定されたターゲットCPUの機種別情報ファイルが存在しない限り、[プロパティー]ダイアログの以下のページを表示する場合に一度だけ表示されます。
- [GNU17 GDBコマンド]ページ
- [GNU17 パラメータ設定]ページ
- [GNU17 ビルドオプション]ページ
- [GNU17 フラッシュ設定]ページ
- [GNU17 リンカスクリプト設定]ページ
- [GNU17 一般設定]ページ

・プロジェクトに設定されたターゲットCPUの機種別情報ファイルが存在しないまま、ビルドを実行しようとすると下記エラーダイアログが表示されます。



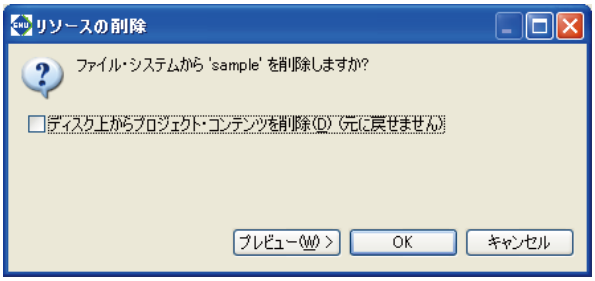
5.4.6 プロジェクトの削除

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。

●プロジェクトの削除ができない場合

[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>*_gnu17IDE.cmd)の名称
  元の*<旧プロジェクト名>*_gnu17IDE.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>*_gnu17IDE.mak)*
  - リンカスクリプトファイル(*<プロジェクト名>*_gnu17IDE.lds)*
  - パラメータファイル(*<プロジェクト名>*_gnu17IDE.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、次回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。*<旧プロジェクト名>*_gnu17IDE.cmdファイルの内容をコピーし、[プロパティー]ダイアログの[GNU17 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メニューから、"プロジェクトのコピー"を行った後に"プロジェクトの貼り付け"の操作を行った場合、上記のようなプロジェクトの設定の自動的な変更は行われませんのでご注意ください。

●プロジェクトの名称変更ができない場合

プロジェクトの名称を変更しようとしたとき、['リソース名の変更'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、プロジェクトの名称変更が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、プロジェクトの名称変更ができなくなります。
この場合は、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、名称変更前のプロジェクトフォルダを削除してください。

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。

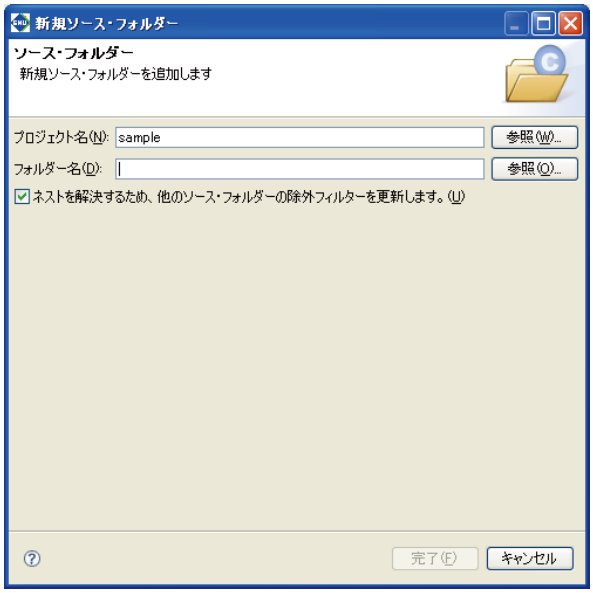
●新規ソースフォルダの作成

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する必要があります。ソースフォルダに格納することで、プロジェクトが自動的にソースファイルをメイクファイルに追加したり、リンカスクリプトファイルを更新してメイクの対象にします。
プロジェクトフォルダは、プロジェクト作成時点でソースフォルダとして扱われます。

(1)以下のいずれかの操作を行います。

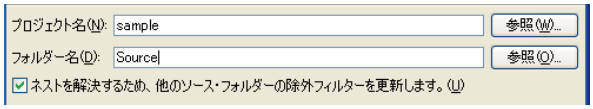
- [ファイル]メニューから[新規]>[ソース・フォルダー]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[ソース・フォルダー]を選択します。
- ツールバーの[新規]ショートカットから[ソース・フォルダー]を選択します。
- ツールバーの[新規 C/C++ ソース・フォルダー]ボタンをクリックします。

[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。

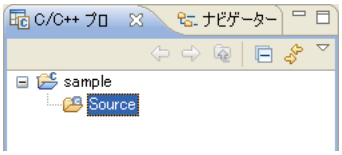
(3)[フォルダー名:]に作成するソースフォルダの名称を入力します。



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、プロジェクトフォルダ直下に配置した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) [完了]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。

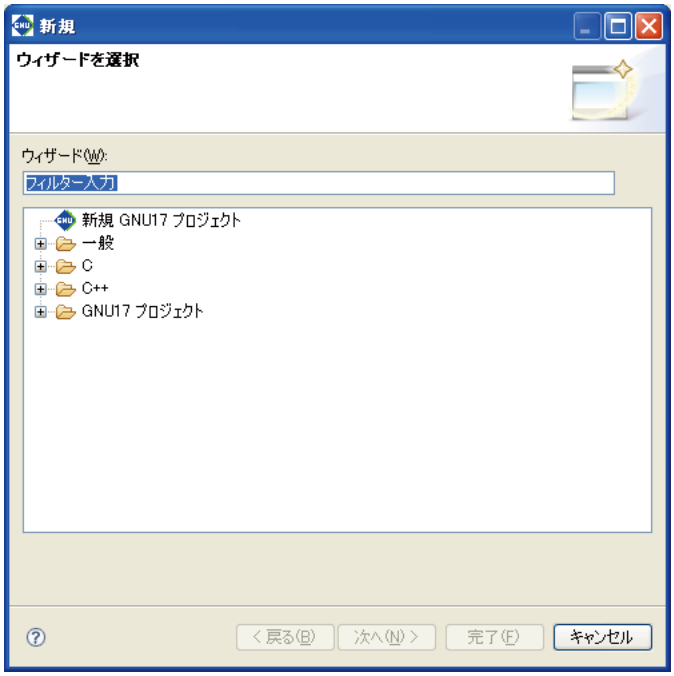
●新規汎用フォルダの作成

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。

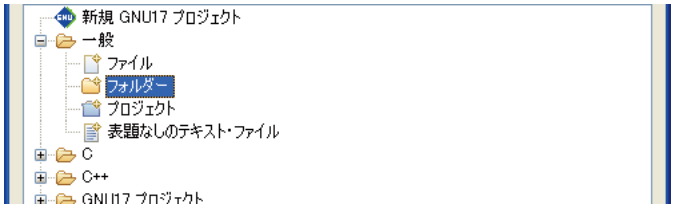
(1) 以下のいずれかの操作を行います。

- [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。

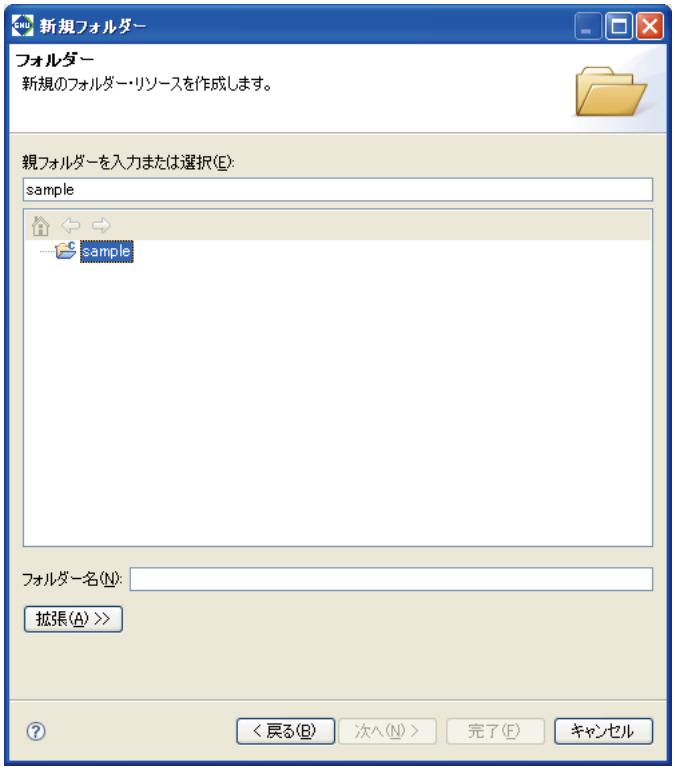
[新規]ダイアログが表示されます。



(2) リストの[一般]から[フォルダー]を選択します。

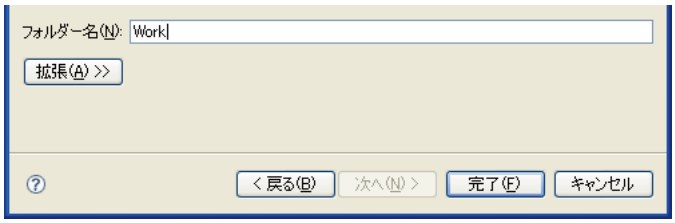


(3) [次へ>]ボタンをクリックします。



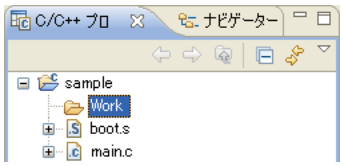
(4) フォルダのツリー表示から、新たなフォルダ(サブディレクトリ)を作成するプロジェクトまたは親フォルダをクリックして選択します。

(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) [完了]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。

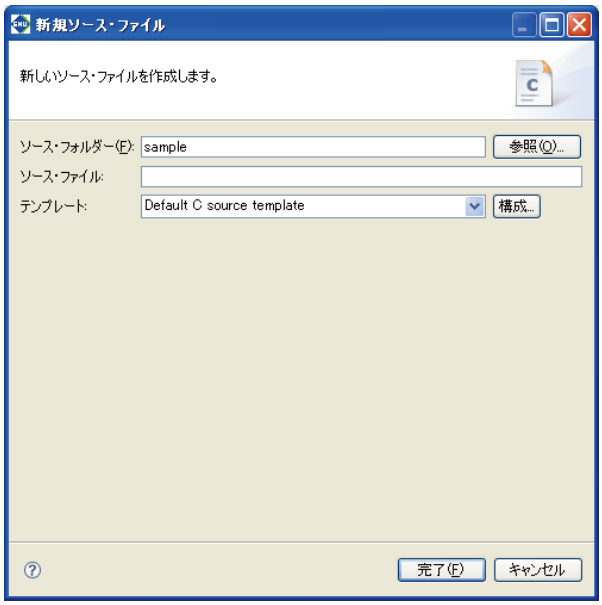
●新規ソースファイル/ヘッダファイルの作成

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。

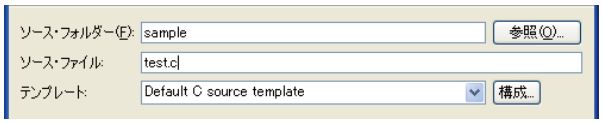
(1)以下のいずれかの操作を行います。

- [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)。

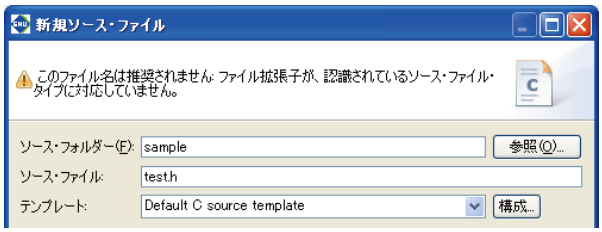
[新規 ソース・ファイル](または[新規 ヘッダー・ファイル])ダイアログが表示されます。



(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。



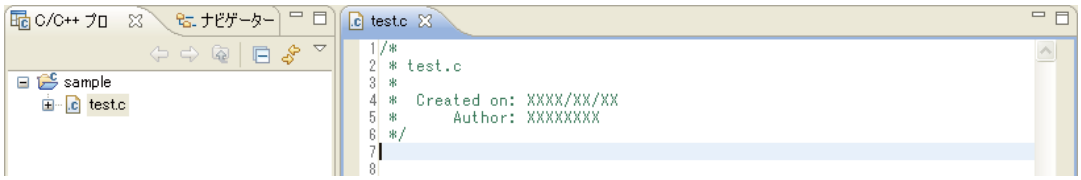
Cソースファイルを作成する場合は拡張子を".c"、アセンブラソースファイルを作成する場合は拡張子を".s"として入力してください。
対応しているソースファイルの拡張子が入力されないと、次のメッセージが表示されます。ただしこの場合でも、その名称でファイルを作成することは可能です。



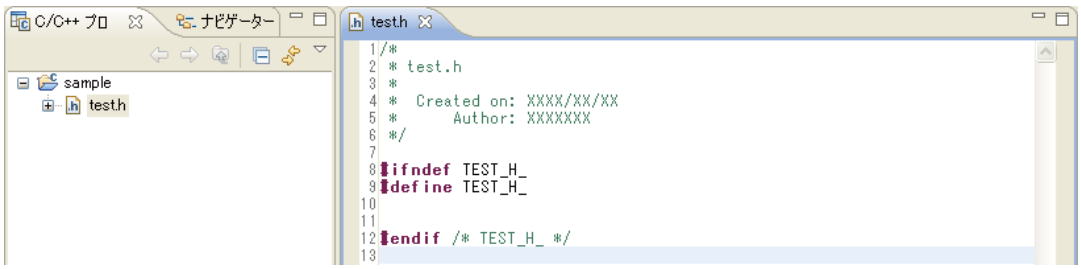
[ソース・フォルダー :]には、現在作業中のプロジェクトフォルダ名が表示されます。他のディレクトリに作成する場合は、パスを入力するか、[参照...]ボタンでディレクトリを選択してください。

(3)[完了]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。



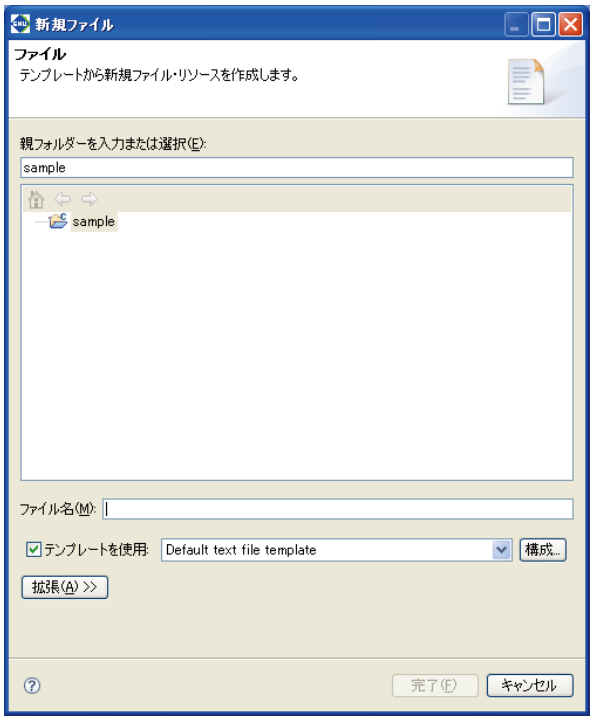
●新規汎用テキストファイルの作成

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。

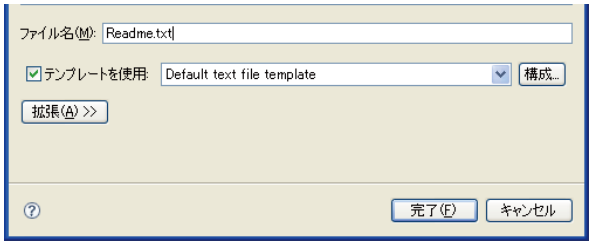
- [ファイル]メニューから[新規]>[テンプレートからファイル]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[新規]>[テンプレートからファイル]を選択します。
- ツールバーの[新規]ショートカットから[テンプレートからファイル]を選択します。
- ツールバーの[新規 C/C++ ソース・ファイル]ショートカットから[テンプレートからファイル]を選択します。

[新規 ファイル]ダイアログが表示されます。



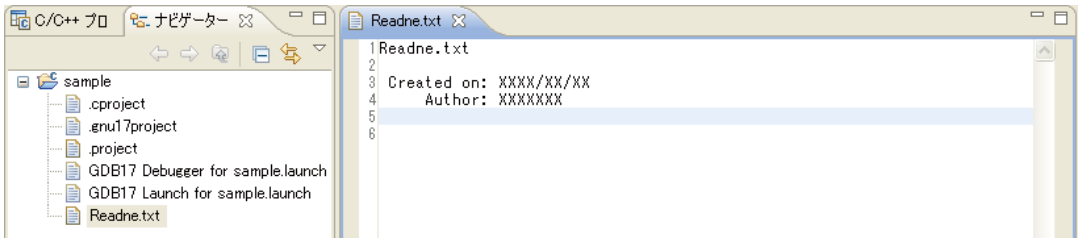
(2) フォルダのツリー表示から、新たなファイルを作成するプロジェクトまたは親フォルダを選択します。

(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) [完了]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。

●既存ファイル、フォルダのインポート

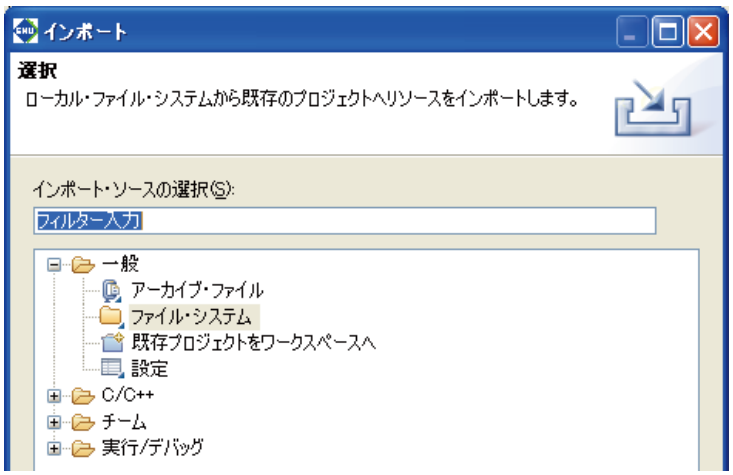
ここでは、既存のファイルやフォルダをプロジェクトに取り込む方法を説明します。ここで説明するインポートは、ファイルシステム上でもファイル/フォルダがプロジェクトディレクトリにコピーされます。既存のソースなどを修正して新たなソースを作成する場合など、オリジナルファイルをそのままの状態に保存しておくことができます。インポートの手順は次のとおりです。

(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。

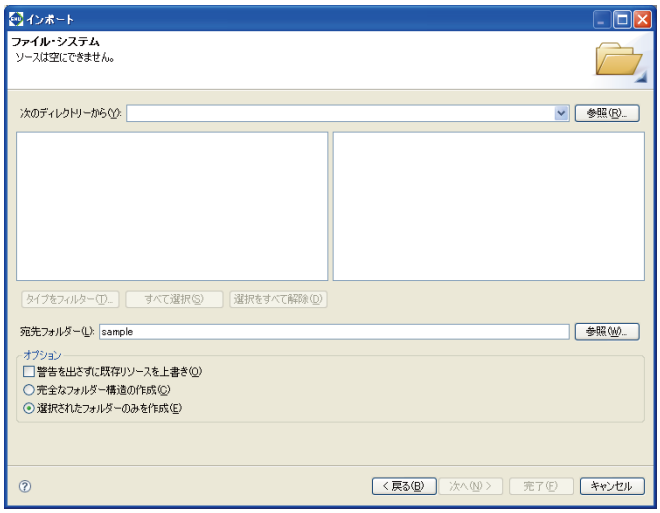
(2) 以下のいずれかの操作を行います。

- [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。

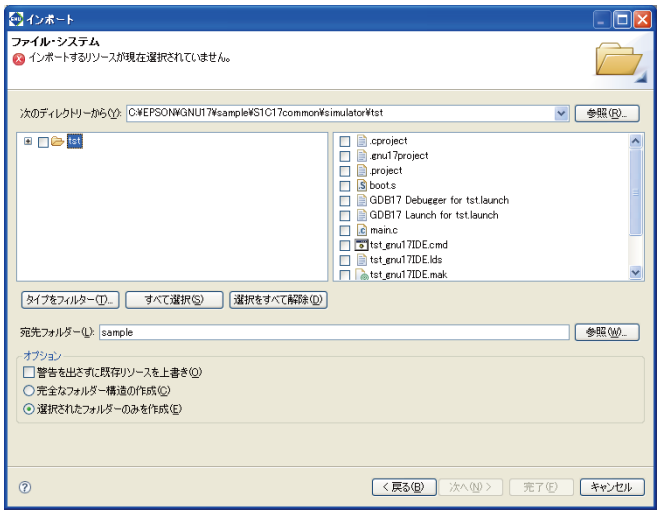
[インポート]ウィザードが起動します。



(3) 一覧から[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。

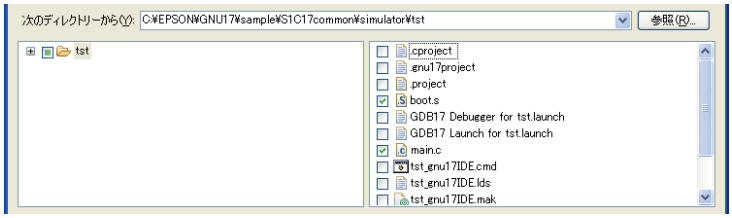


(4) [次のディレクトリーから:]の[参照...]ボタンをクリックして表示されるフォルダ選択ダイアログで、インポートするファイル/フォルダがあるフォルダを選択してください。選択したフォルダへのパスが[次のディレクトリーから:]に入力されます。以前インポート元となったフォルダであれば、[次のディレクトリーから:]の ボタンで表示される履歴から選択することも可能です。

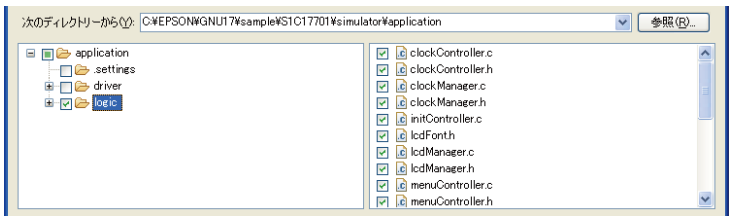


注: ここでは、必ずインポートするファイル/フォルダの親フォルダを選択してください。左側のフォルダリストにルートとして表示されているフォルダ内のファイル/フォルダがインポート対象となります。

(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) [完了]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、5.10.3節を参照してください。

●ソースフォルダの指定

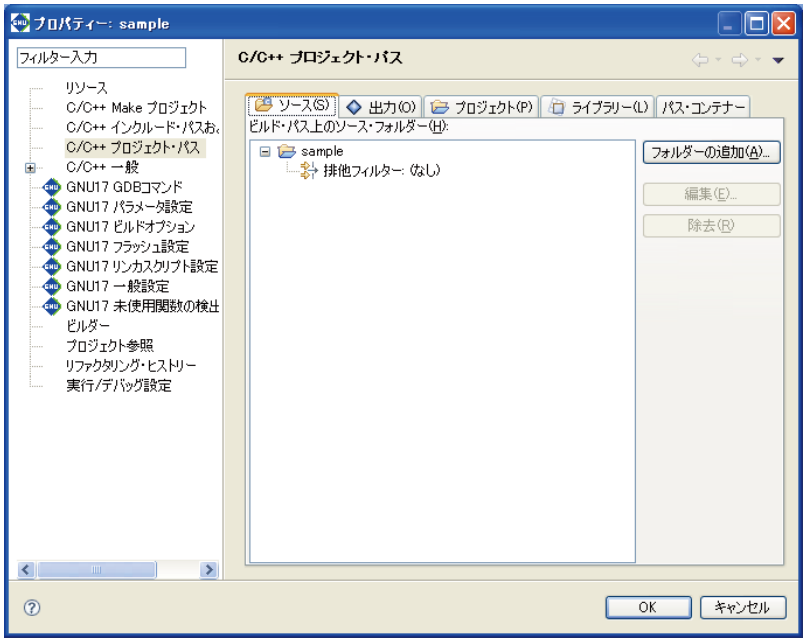
IDEは、プロジェクトフォルダ直下に置かれたソースファイル、およびプロジェクト内のソースフォルダに含まれるソースファイルをmakeファイルに記述してコンパイル/アセンブルの対象にします。プロジェクト内のソースフォルダ以外のフォルダに含まれるソースファイルは初期設定状態ではmakeファイルには記述されません。ただし、プロジェクトのプロパティにソースフォルダとして登録することで、その中のソースファイルもメイクの対象になります。
その手順は次のとおりです。なお、登録するフォルダがすでにプロジェクト内に作成あるいはインポートされているものとします。

(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[プロパティー]を選択します。

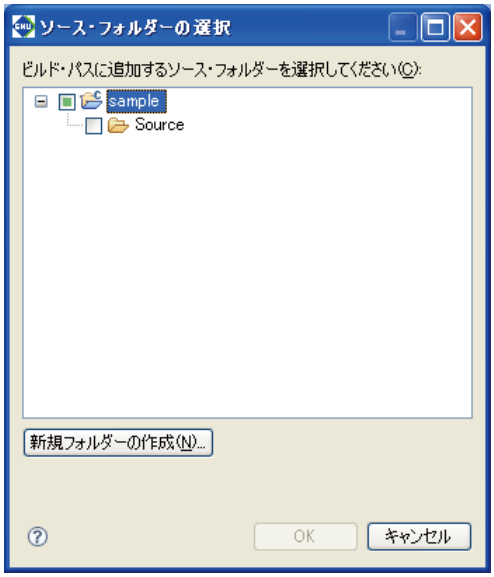
[プロパティー]ダイアログが表示されます。

(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。

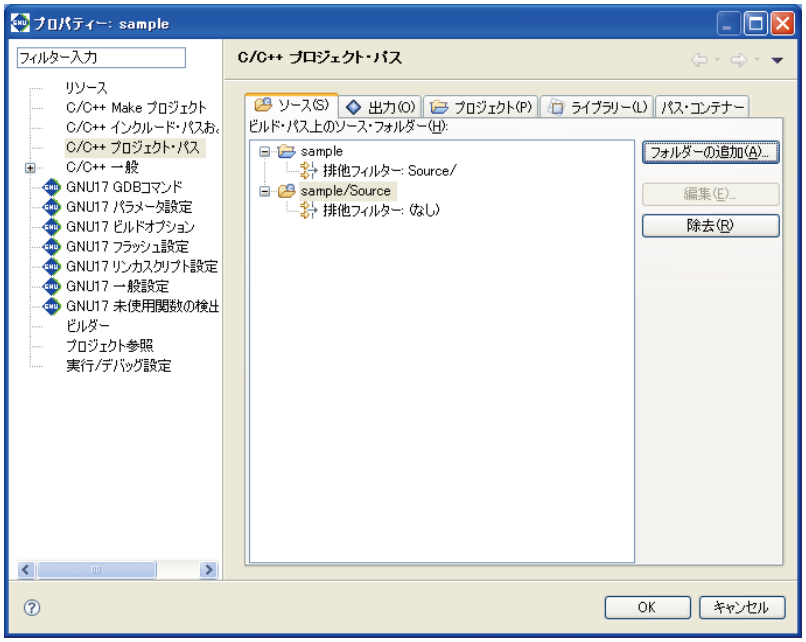


プロジェクトフォルダはデフォルト設定で登録済みです。

(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。



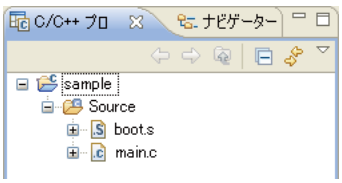
(5) 登録するフォルダを選択し(チェックを付け)、[OK]ボタンをクリックします。



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[プロパティー]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。

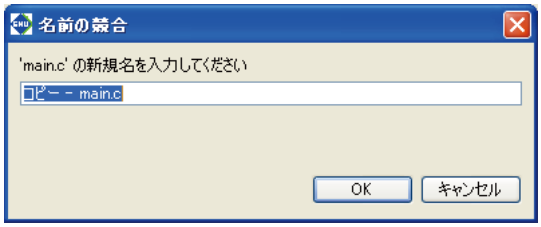
選択したリソースがクリップボードにコピーされます。

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]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

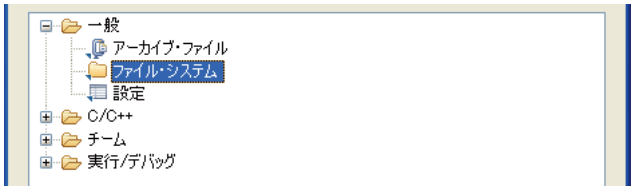
[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビュー内のファイル/フォルダを次の手順でワークスペース外に書き出すことができます。

(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。

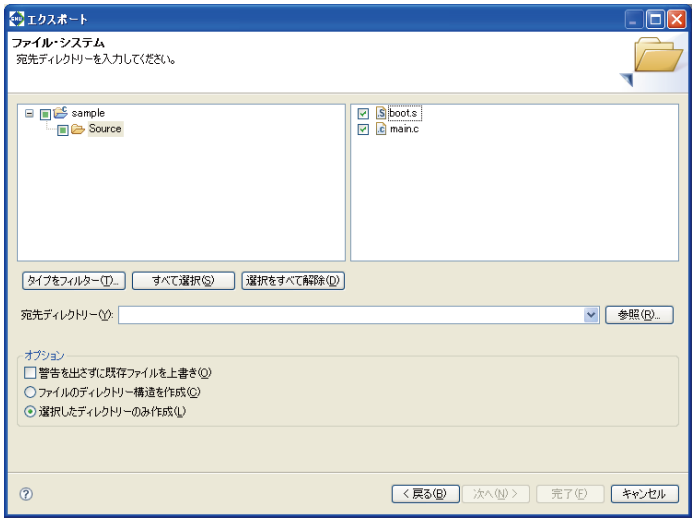
(2)以下のいずれかの操作を行います。
- [ファイル]メニューから[エクスポート...]を選択します。
- [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューのコンテキストメニューから[エクスポート...]を選択します。

[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>]ボタンをクリックします。



(5)[宛先ディレクトリー:]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[宛先ディレクトリー:]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、[宛先ディレクトリー:]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)[完了]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、5.10.4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。

(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]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。

• 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。

● プロジェクト外のファイルへのリンクを張る

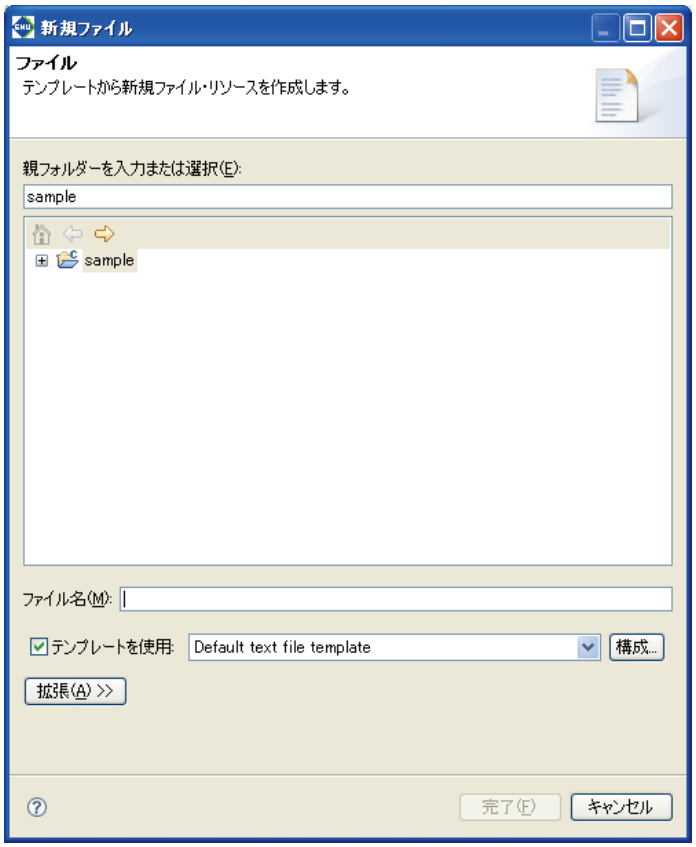
チーム開発中では、プロジェクトフォルダの外にあるソースファイルを参照してビルドしたいことがあります。

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。



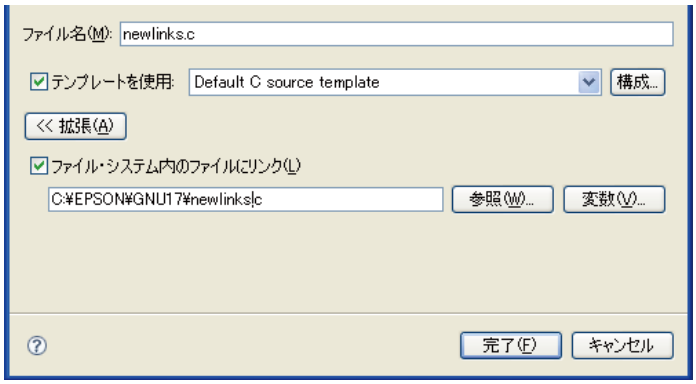
(2) フォルダのツリー表示から、新たなファイルを作成するプロジェクトまたは親フォルダを選択します。

(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) [完了]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]ボタン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

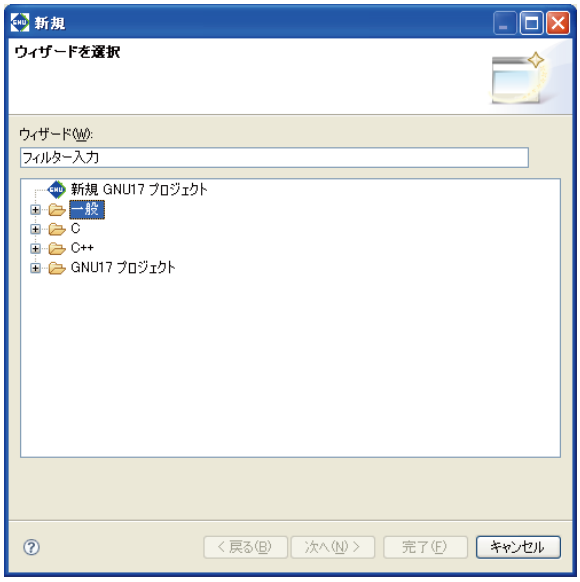
チーム開発中では、プロジェクトフォルダの外にあるソースファイルをフォルダ単位に参照してビルドしたいことがあります。

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。

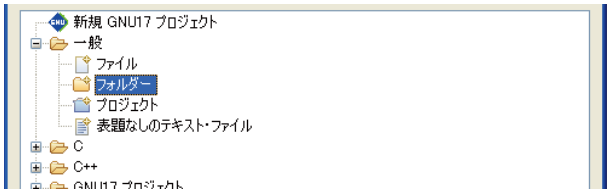
(1) 以下のいずれかの操作を行います。

- [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。

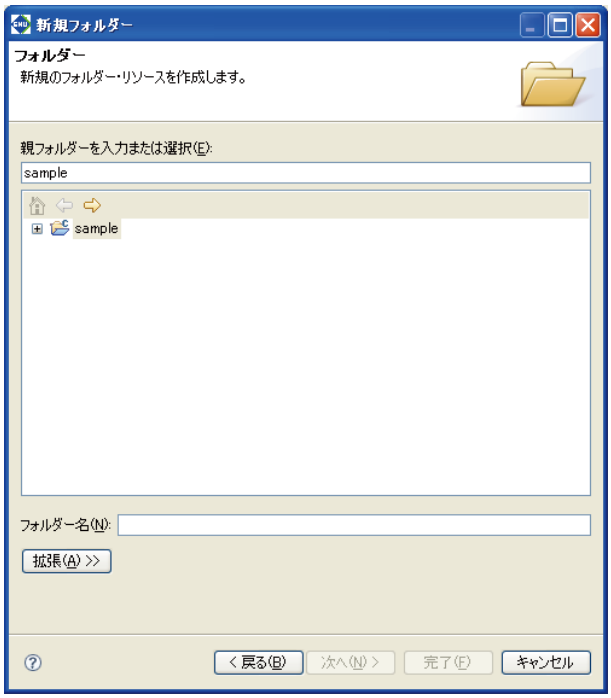
[新規]ダイアログが表示されます。



(2) リストの[一般]から[フォルダー]を選択します。



(3) [次へ>]ボタンをクリックします。



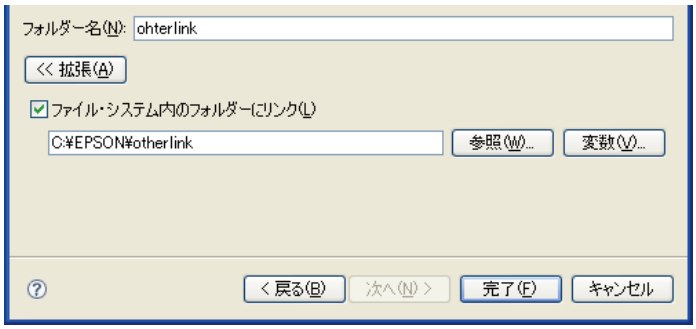
(4) フォルダのツリー表示から、新たなファイルを作成するプロジェクトまたは親フォルダを選択します。

(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ーにリンク]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) [完了]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]ボタン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、

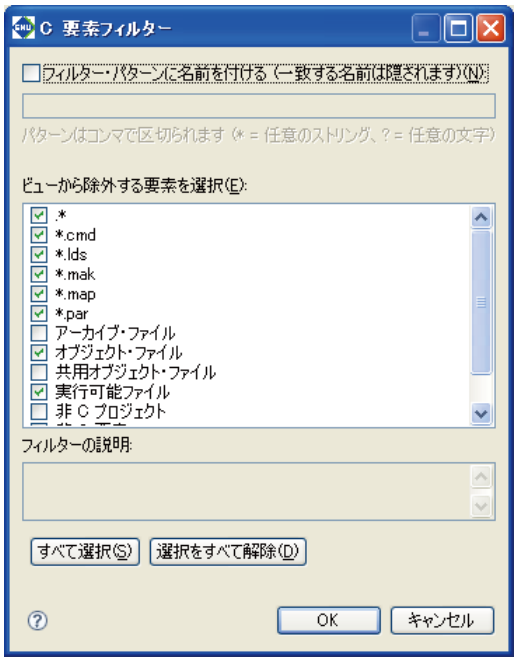
プロジェクトフォルダ直下に生成されます。

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。

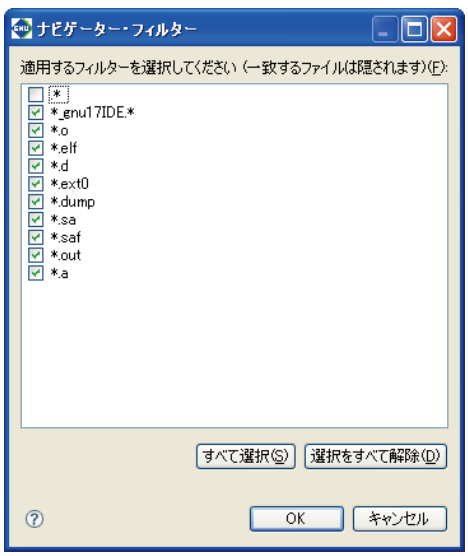


ここで、表示させないファイルの種類を選択します。[すべて選択]はすべてを選択、[選択をすべて解除]はすべての選択を解除します。
また、[フィルター・パターンに名前を付ける:]チェックボックスを選択し、そのテキストボックスに文字パターンを入力することで、その文字パターンに一致するファイルを非表示にすることができます。このパターン指定には、*(文字列)と?(1文字)のワイルドカードが使用できます。
[C/C++ プロジェクト]ビューの初期設定では、以下のファイルが表示されないようになっています。

- .*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)
- *_gnu17IDE.*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)
- *.outに該当するファイル(ベクタチェッカ用ファイル)
- *.aに該当するファイル(ライブラリファイル)

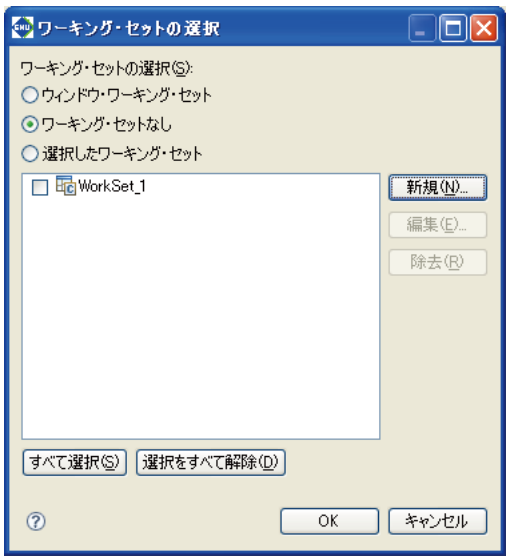
5.4.10 ワーキングセット

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]ボタンをクリックします。

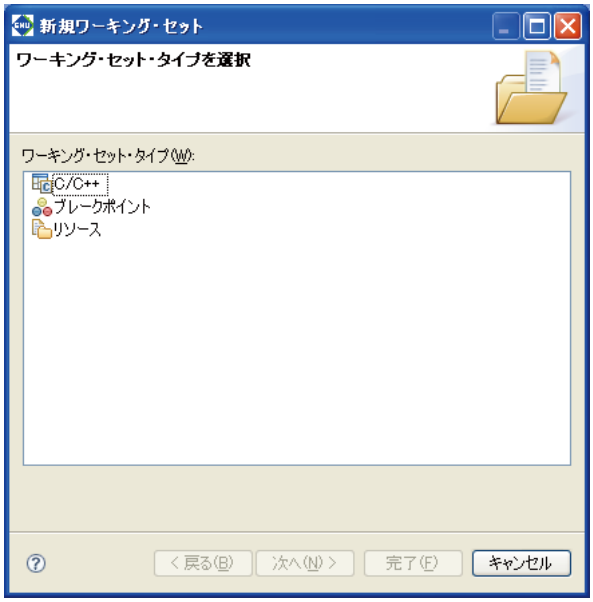
[C/C++ プロジェクト]ビューの表示が、選択したワーキングセットに定義されているリソースのみになります。

[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。

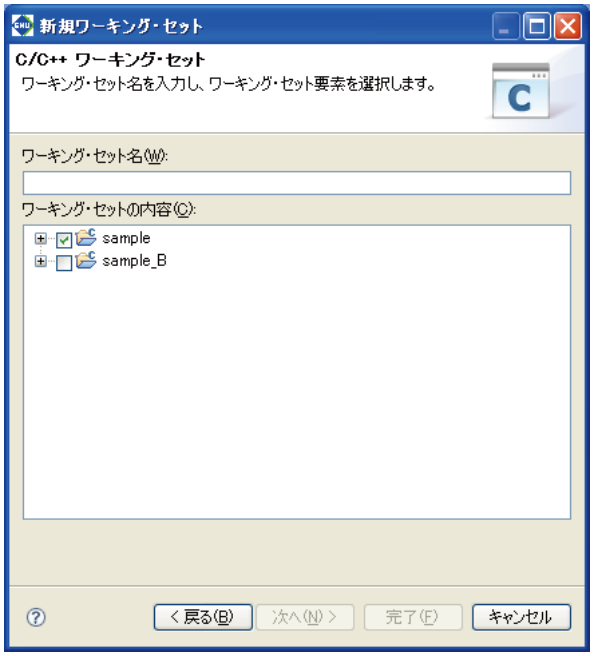
●ワーキングセットの作成

(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。



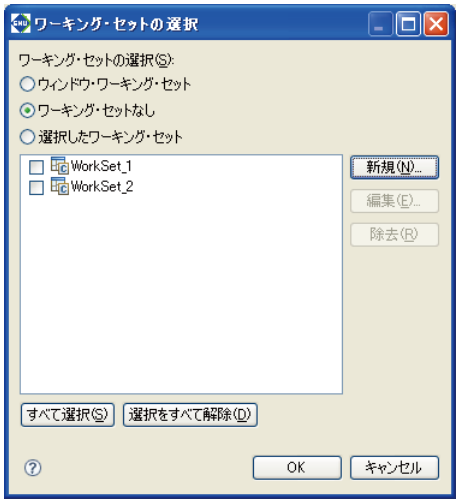
(3) [ワーキング・セット・タイプ:]の一覧から[C/C++]を選択し[次へ>]ボタンをクリックします。



(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) [完了]ボタンをクリックします。



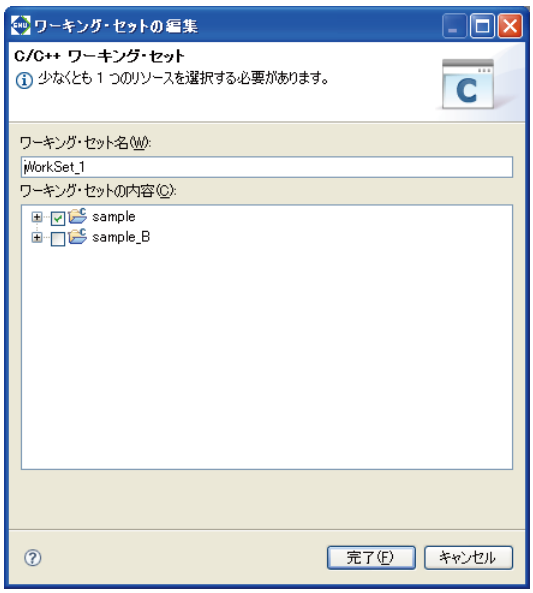
(7) [OK]ボタンをクリックするとビューに適用されます。[キャンセル]ボタンをクリックすると上記の操作が無効になります。

●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...]ボタンをクリックします。
[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、[完了]ボタンをクリックします。

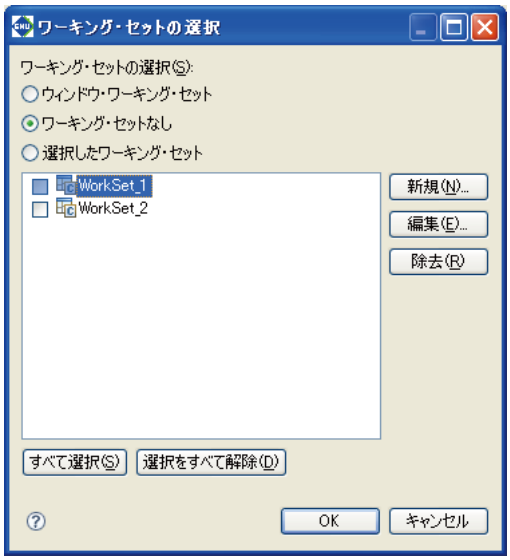
(4) [OK]ボタンをクリックするとビューに適用されます。[キャンセル]ボタンをクリックすると上記の操作が無効になります。

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]ボタンをクリックします。
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。

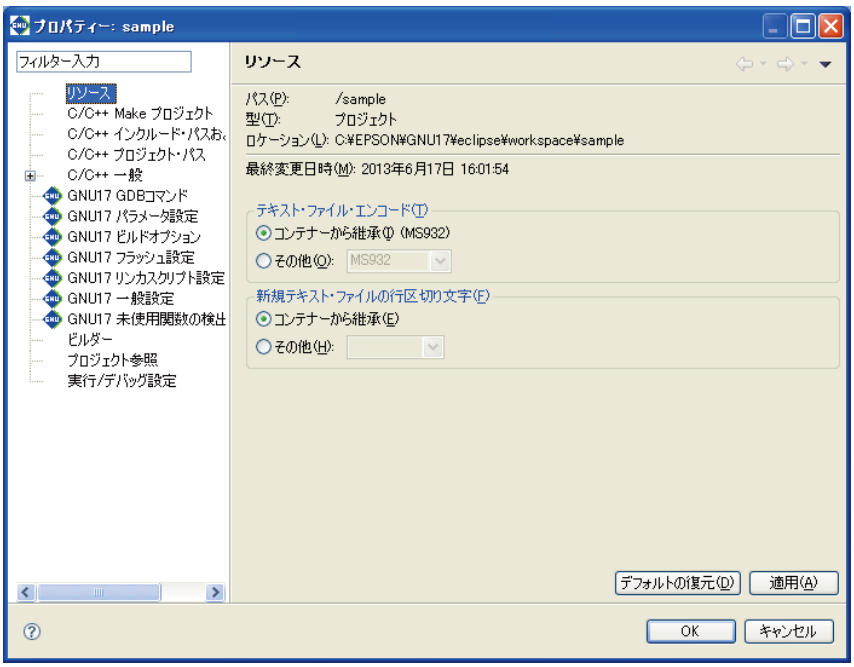
(3) [OK]ボタンをクリックします。
別のワーキングセットを使用する場合は、[OK]ボタンをクリックする前に一覧から選択してください。

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、[プロパティー]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。
[プロパティー]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[プロパティー]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. ビルダー

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、ヘルプメニューのC/C++ Development User Guide > Reference > C/C++ プロパティー > C/C++ Project プロパティー > 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. GNU17 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. GNU17 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。

9. GNU17 一般設定
ターゲットプロセッサ、メモリモデル、コプロセッサ用ライブラリをリンクするかどうか選択します。

10. GNU17 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. GNU17 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。

12. GNU17 フラッシュ設定
フラッシュプロテクトビット、Flashセキュリティパスワードおよびフラッシュ操作プログラムを設定します。

13. GNU17 未使用関数の検出
ビルド処理時にプロジェクト内の未使用関数を検出するかどうか選択します。

14. 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。

15. 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。

16. 実行/デバッグ設定
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エディタ

Cソース(*.c)やヘッダファイル(*.h)の作成/編集に使用します。このエディタでは、Cの予約語、コメント、文字列が強調表示されます。また、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)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では、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おりません。

3. テキストエディタ

上記の以外の形式(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、[プロパティー]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[プロパティー]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の状態となってしまうため、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。

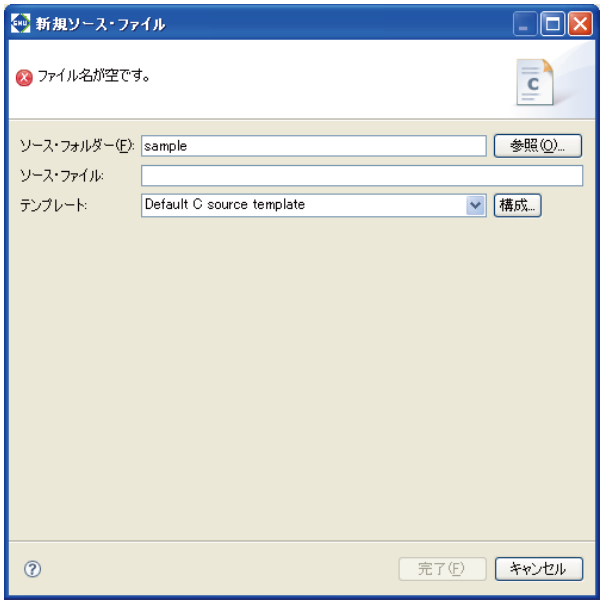
●ソースの新規作成

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)[完了]ボタンをクリックします。

エディタ領域に空白の文書が開きます。

●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] (エディター)	
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] (エディター)	–
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] (エディター)	–
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] (エディター)	–
コピー	[編集]>[コピー] (Ctrl+C)	[コピー] (エディター)	–
貼り付け	[編集]>[貼り付け] (Ctrl+V)	[貼り付け] (エディター)	–
削除	[編集]>[削除] (削除)	–	–
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)	–	–

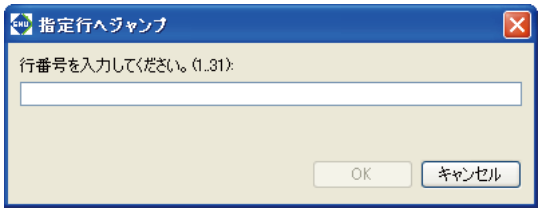
5.5.3 Cソース用編集機能

Cエディタは前述の基本編集機能に加え、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。

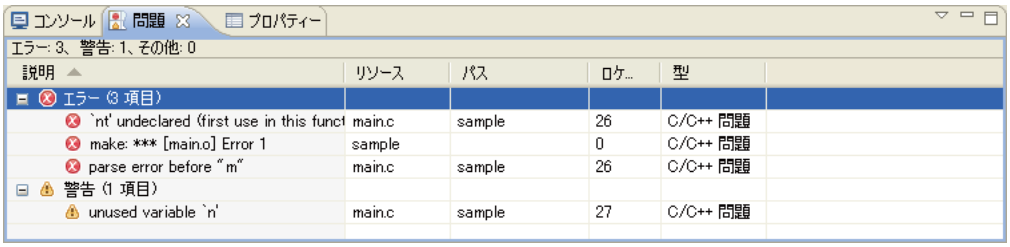
(1)[ナビゲート]メニューから[指定行へジャンプ...]を選択します。
[指定行へジャンプ]ダイアログが表示されます。



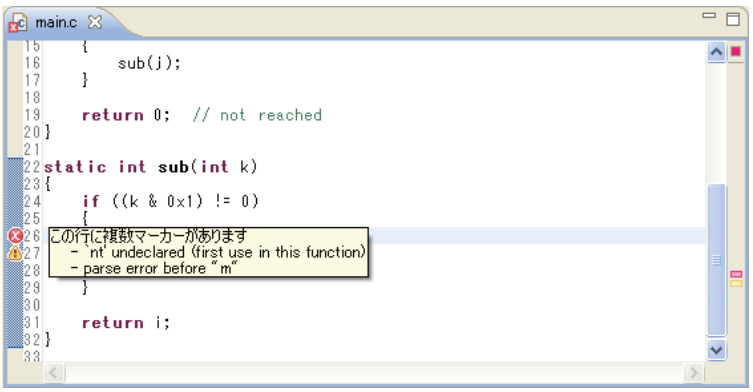
(2)テキストボックスに行番号を入力し、[OK]ボタンをクリックします。
指定行にジャンプします。

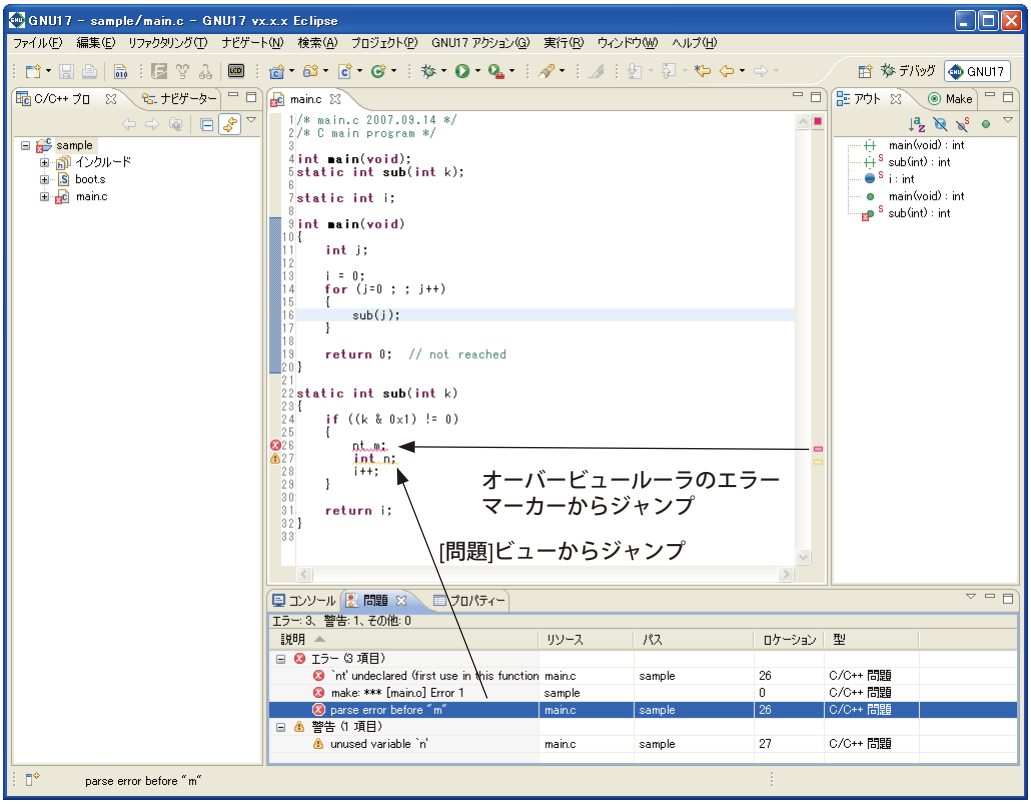
●エラー発生行へのジャンプ

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 (⊗)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ルーラ(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。使用方法は次のとおりです。

(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          */
/* C main program */
          :
      [m]キー入力
          :
/* main.c          */
/* C main program */
          :
    [a] [i] [n]キー入力
          :
/* main.c          */
/* 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          */
/* C main program */
          :
      [ ]キー入力
          :
/* main.c          */
/* 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はありますので、[↑]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[右へシフト]を、左シフトの場合は[左へシフト]を選択します。

例:
```
main()
      {       ←[左へシフト]実行
int j;        ←[右へシフト]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。すでにコメントに設定されている行(//または/*にかかわらず)にも無条件に挿入されます。

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。

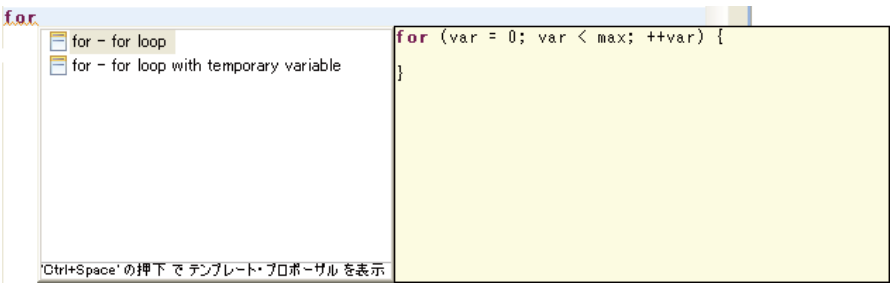
●コンテンツアシスト

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。



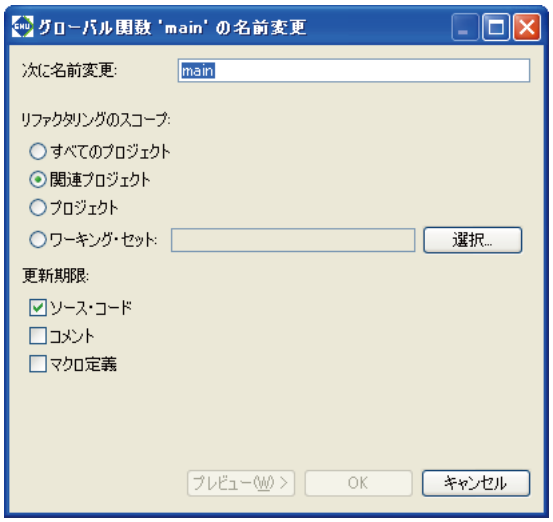
(4)一覧から、使用する予約語またはテンプレートを選択し、ダブルクリックします。
カーソル位置に選択した内容が挿入されます。
テンプレートはループ文や条件文などの定型書式で、文書のアイコンでリストされています。テンプレートの内容は一覧からクリックして選択すると、横に表示されます。汎用のテンプレートはあらかじめ定義されていますが、ユーザ独自のテンプレートを定義することもできます("5.9 **IDE**のカスタマイズ(設定)"の"C/C++ > エディター > テンプレート"参照)。

●リファクタリング

変数、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。

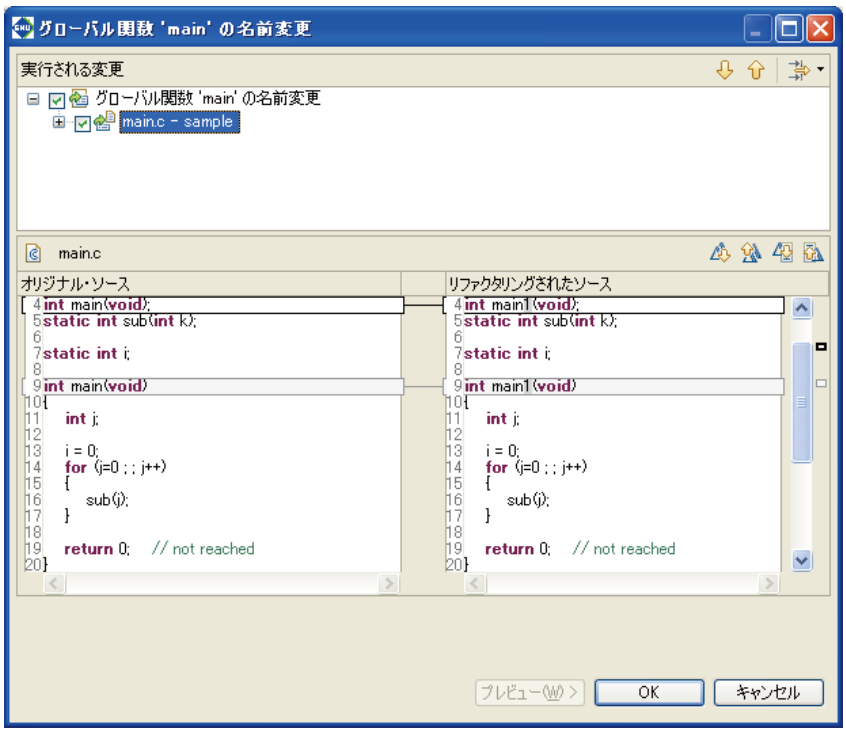


(3) [変更後の名前:]に新しい名称を入力します。

(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。
[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] 現在編集中のプロジェクトに関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。
[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。

(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。



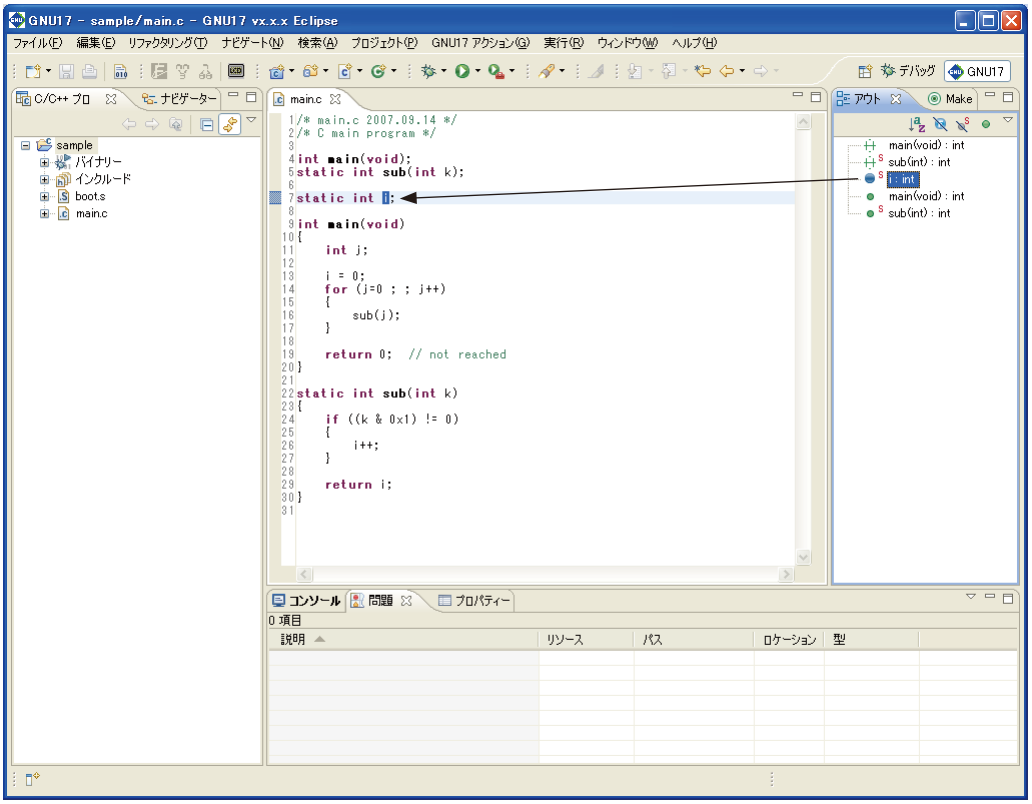
矢印のボタンで、変更箇所を確認することができます。

(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。

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。

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。

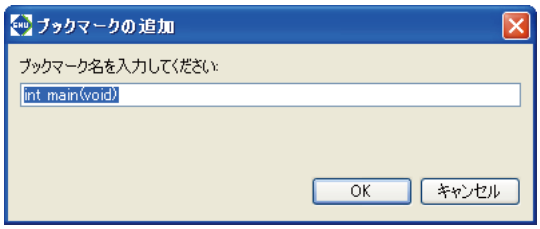
●ブックマークの付加

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。

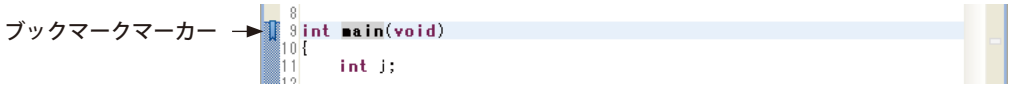
(2) 以下のいずれかの操作を行います。
- [編集]メニューから[ブックマークの追加...]を選択します。
- エディタのマーカーバー(エディタ領域の左端)を右クリックしてコンテキストメニューを表示させ、[ブックマークの追加...]を選択します。

[ブックマークの追加]ダイアログが表示されます。

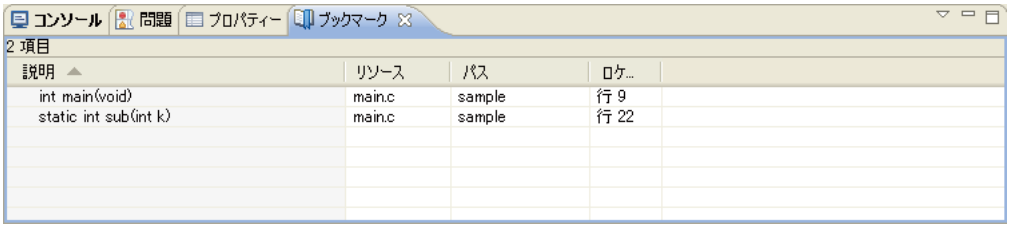


(3) ブックマーク名を設定します。[ブックマーク名を入力してください]に表示された名称をそのまま使用するか、入力し直して[OK]ボタンをクリックします。

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。



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、[説明]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。

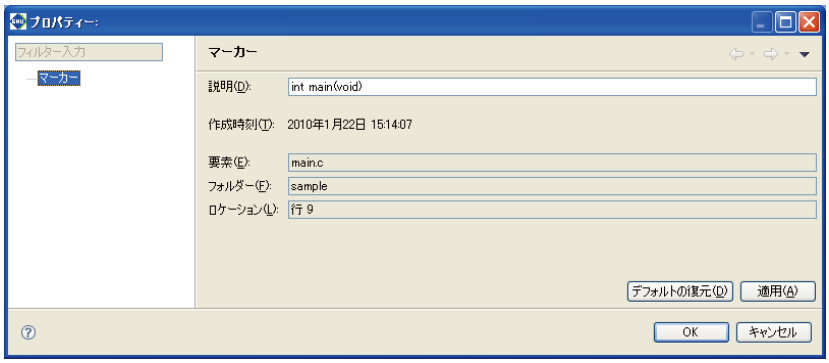
[ブックマーク]ビュー上での削除

(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、[プロパティー]を選択します。[プロパティー]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。

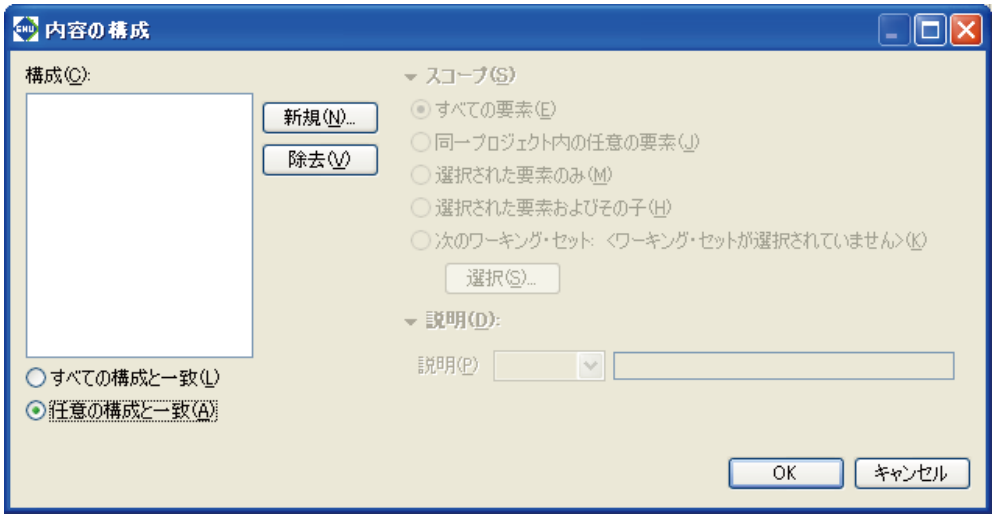
フィルタ

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。

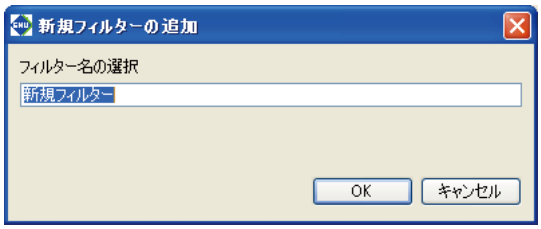
(1) [ブックマーク]ビューをアクティブにします。

(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。

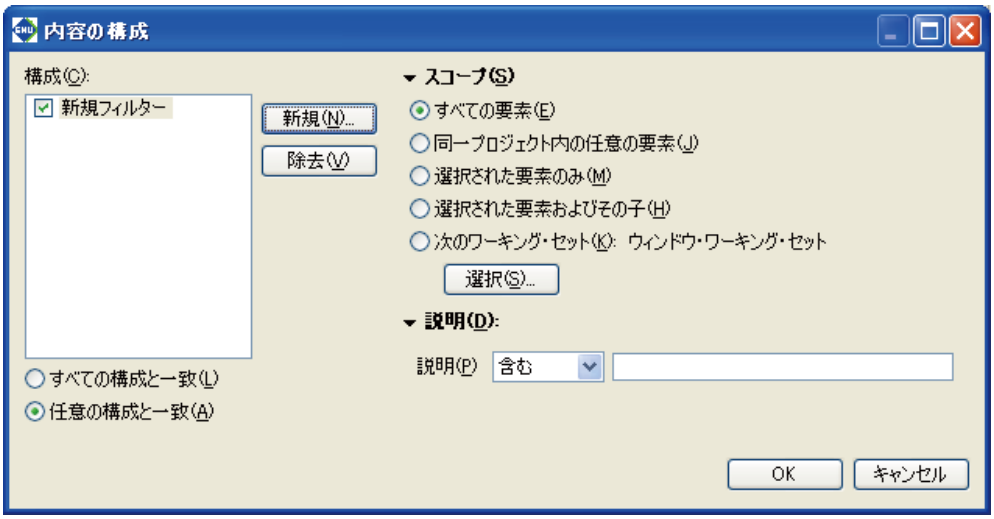


作成するフィルタの名前を入力し、[OK]ボタンをクリックします。

(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。
[説明] - [含む]	上記の範囲指定に加え、[説明]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。
[説明] - [含まない]	上記の範囲指定に加え、[説明]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]ボタンをクリックします。

[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。

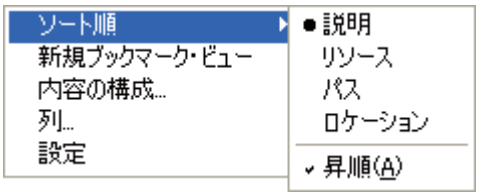
[内容の構成]ダイアログの詳細については5.10.5節を参照してください。

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。

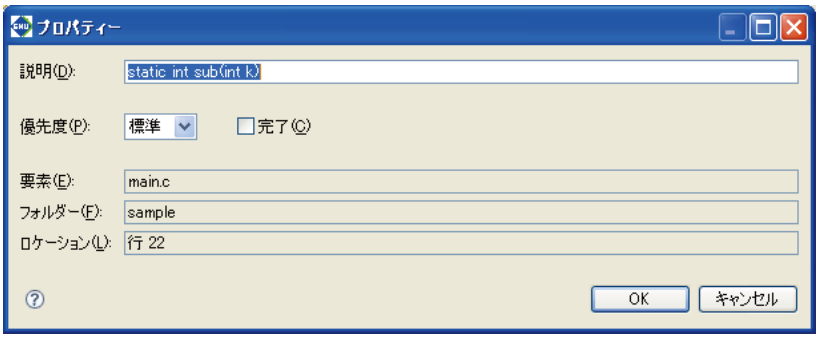
ソース行情報を含める場合

(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。

[プロパティー]ダイアログが表示されます。



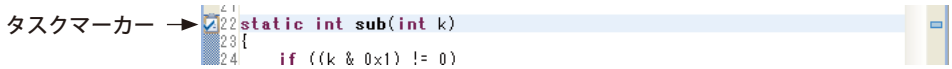
(3)[説明:]にタスクの説明を入力します。

(4)[優先度:]から優先順位(高、標準、低)を選択します。

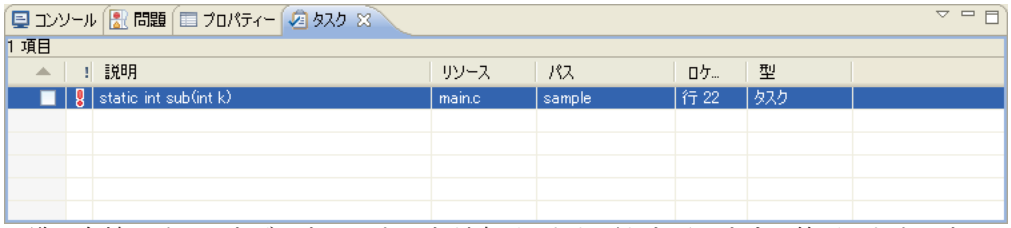
(5)完了したタスクとして作成する場合は[完了]を選択します。

(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。



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。

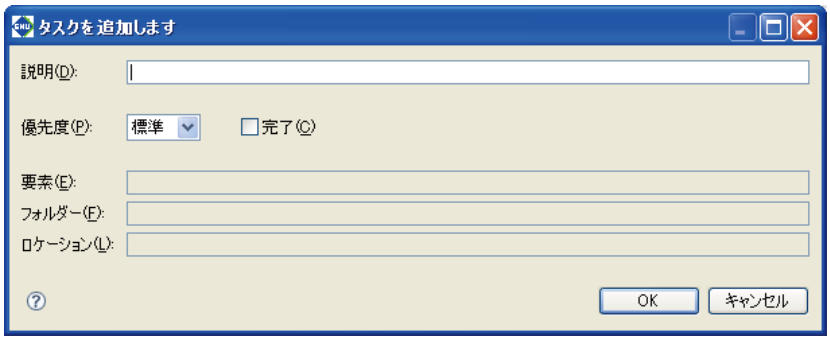
タスクの説明は、[説明]欄をクリックして変更できます。

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)[説明:]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。

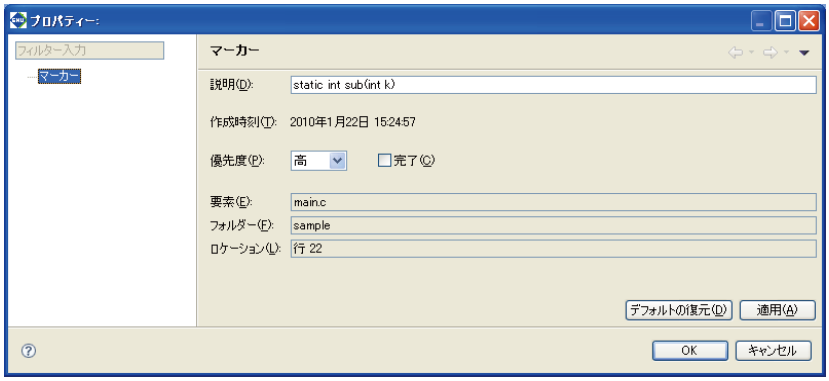
(1)[タスク]ビューのコンテキストメニューから[完了タスクの削除]を選択します。

(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]ボタン、中止する場合は[キャンセル]ボタンをクリックします。

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、[プロパティー]を選択します。[プロパティー]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

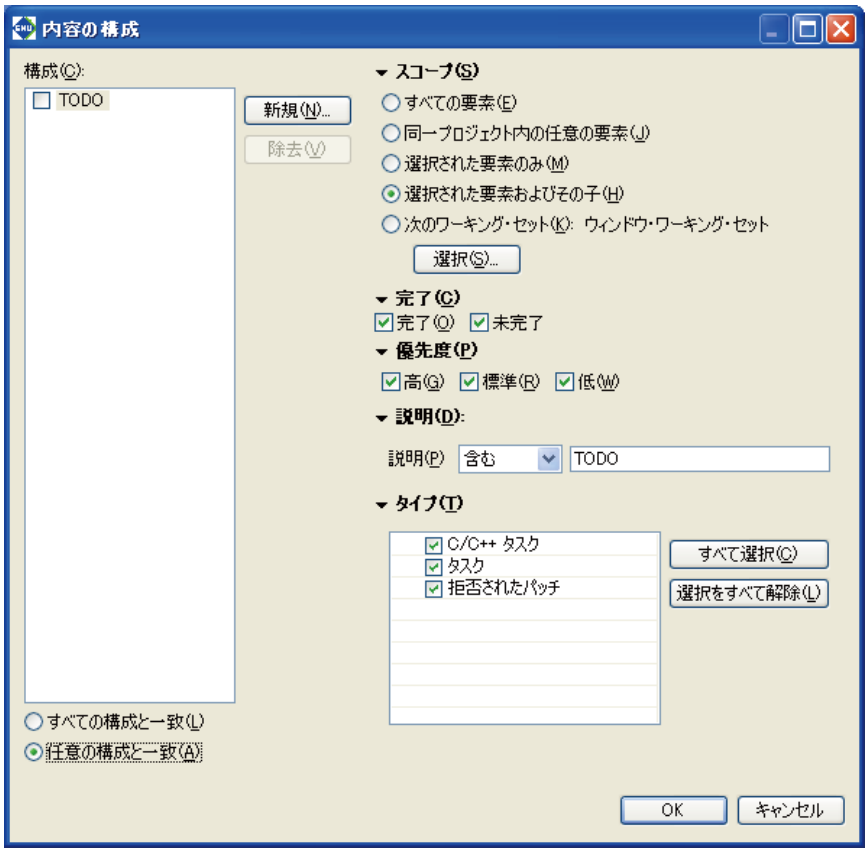
タスクが増え[タスク]ビューの一覧が大きくなりすぎたときのために、一覧の表示を制限したり、項目で並べ替えたりすることができます。

フィルタ

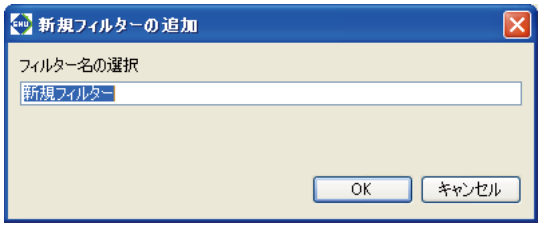
フィルタを使用し、必要なタスク以外を非表示にできます。
また、複数のフィルタを作成し、必要に応じて使い分けることができます。
新規のフィルタは以下の手順で作成します。

(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。



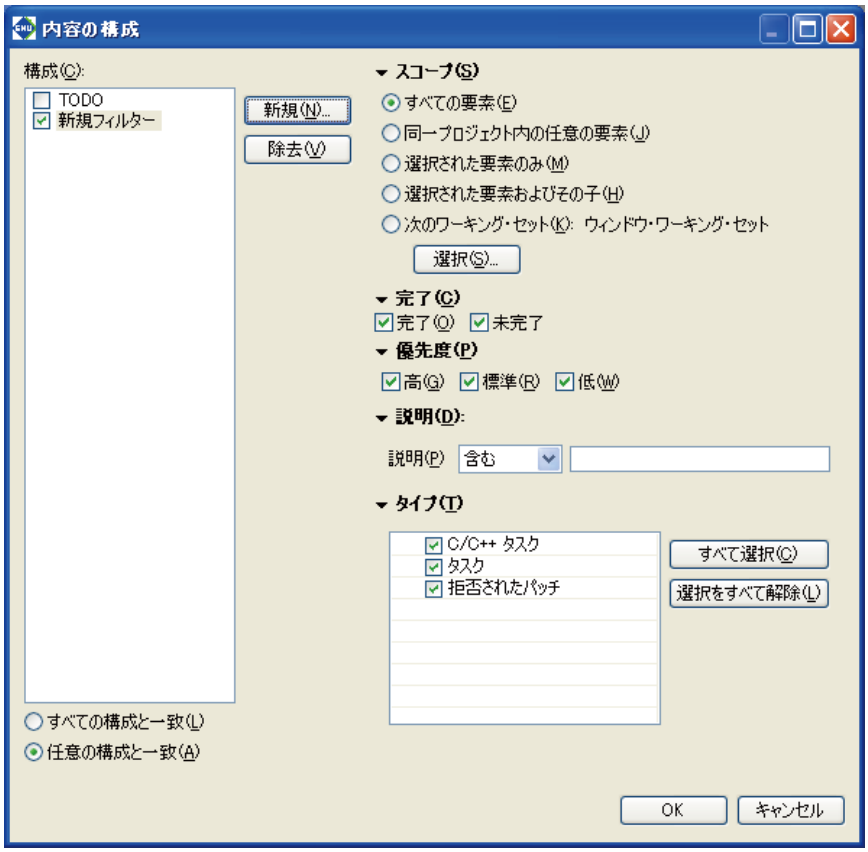
作成するフィルタの名前を入力し、[OK]ボタンをクリックします。

(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、設定されているタスクを表示します。
[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タスクを表示します。
[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タス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タス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] - [完了] / [未完了]	完了/未完了を表示させる場合に選択します。
[優先度] - [高] / [標準] / [低]	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。
[説明] - [含む]	上記の範囲指定に加え、[説明]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。
[説明] - [含まない]	上記の範囲指定に加え、[説明]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]ボタンをクリックします。

[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。

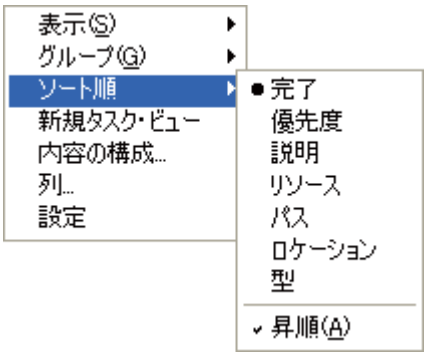
[内容の構成]ダイアログの詳細については5.10.5節を参照してください。

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。

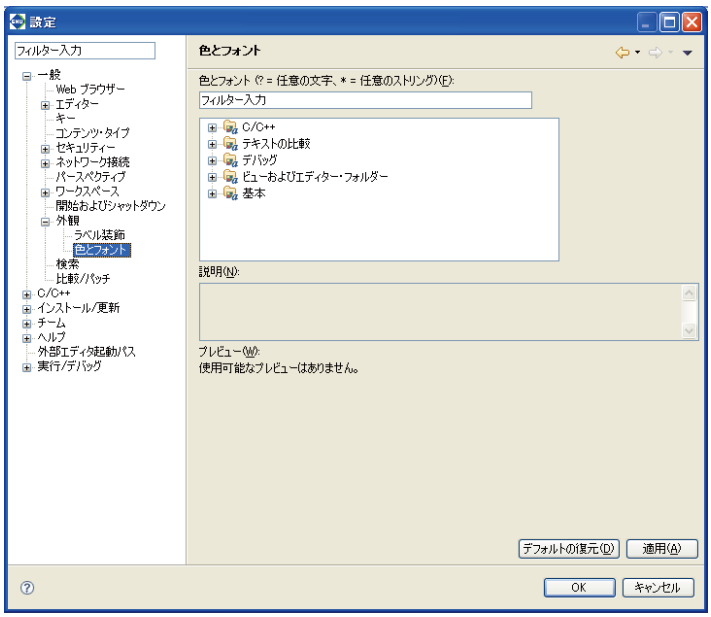
(2) ビューメニュー（▽）から[ソート順]を選択し、一覧のどの項目を優先して並べ替えるかを選択します。また、[昇順]をチェックすると、それぞれの項目を昇順で並べ替え、チェックを外すと降順で並べ替えます。



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係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。

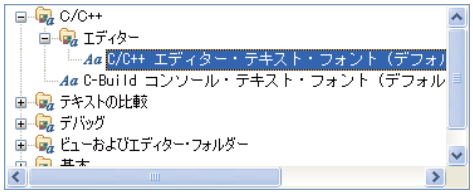
●テキストのフォントと色([一般]>[外観]>[色とフォント])



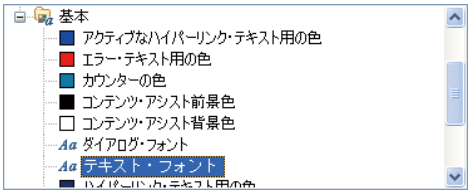
ここで、Cエディタとアセンブラエディタのテキストフォントを変更することができます。

(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。
アセンブラ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エディタフォント

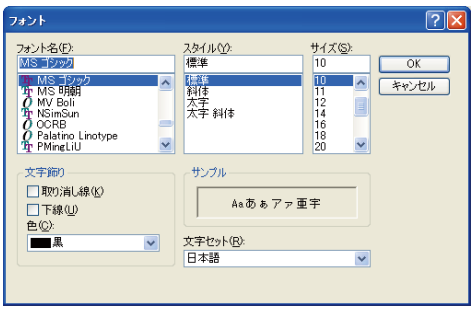


アセンブラエディタフォント



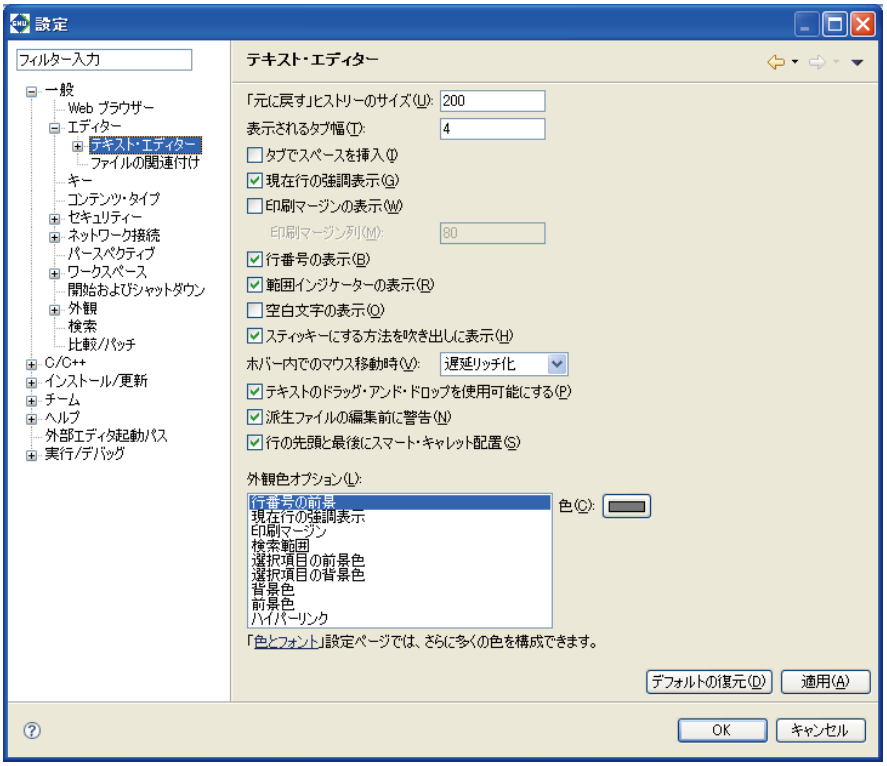
(2) [変更...]ボタンをクリックしてフォント選択ダイアログを表示させ、フォントとスタイルおよび表示色を選択します。

[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]ボタンまたは[OK]ボタン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。

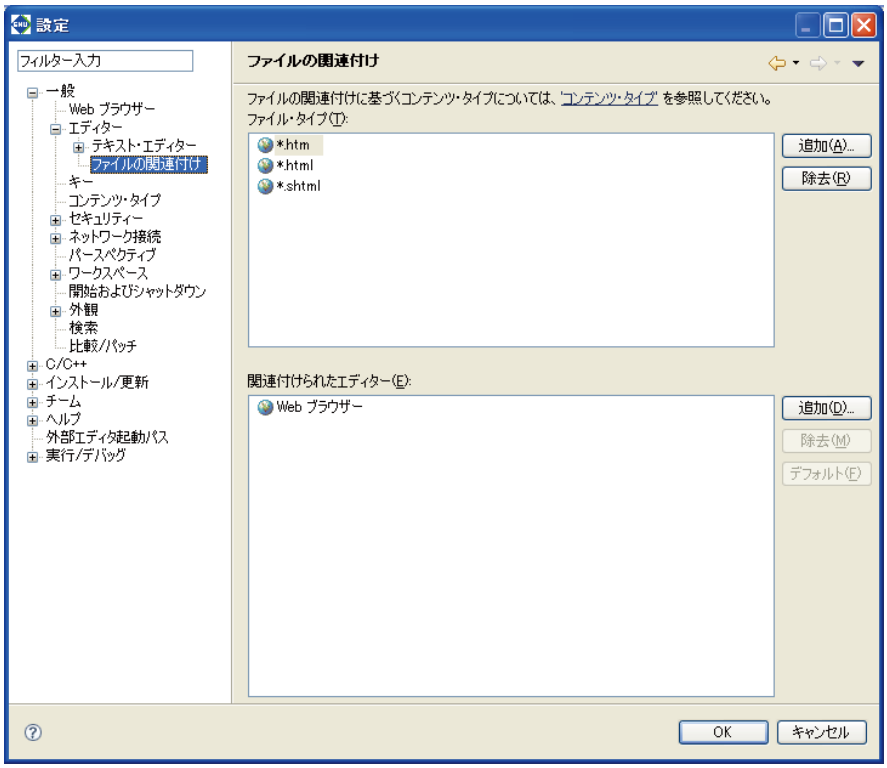
このページでは、このほかに強調表示の設定などが行えます。

5.5.9 外部エディタを使用するには

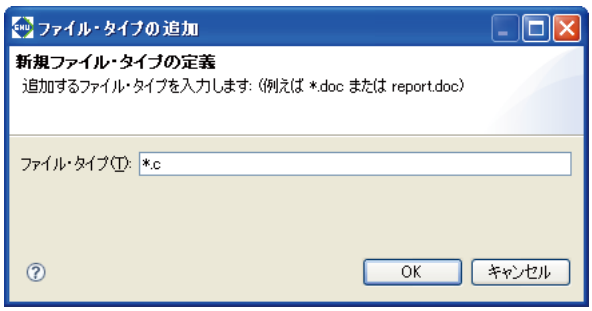
日頃使用しているエディタをIDEに登録して、リソースの編集に使用することができます。
外部エディタを登録する手順は次のとおりです。

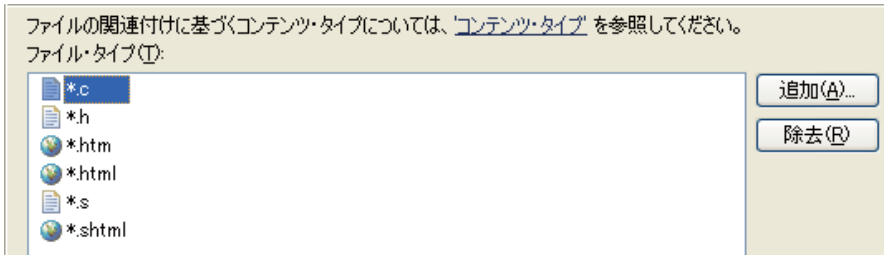
(1) [ウィンドウ]メニューから[設定]を選択します。
[設定]ダイアログが表示されます。

(2) 左側にある設定項目のツリー表示から[一般]>[エディター]>[ファイルの関連付け]を選択します。

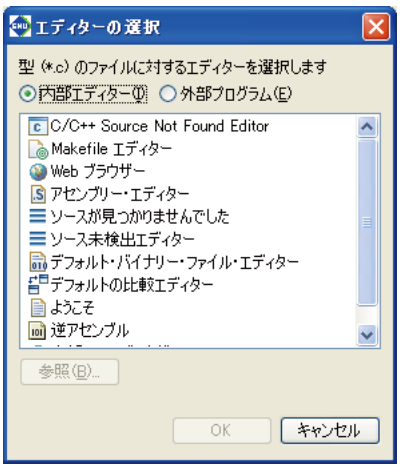


(3) 登録するエディタで編集するファイルの種類(拡張子)を[ファイル・タイプ:]から1つ選択します。
表示されていない場合は[ファイル・タイプ:]の右にある[追加...]ボタンで次のダイアログを表示させ、拡張子(*.c、*.h、*.s)を入力して一覧に加えてください。





(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。

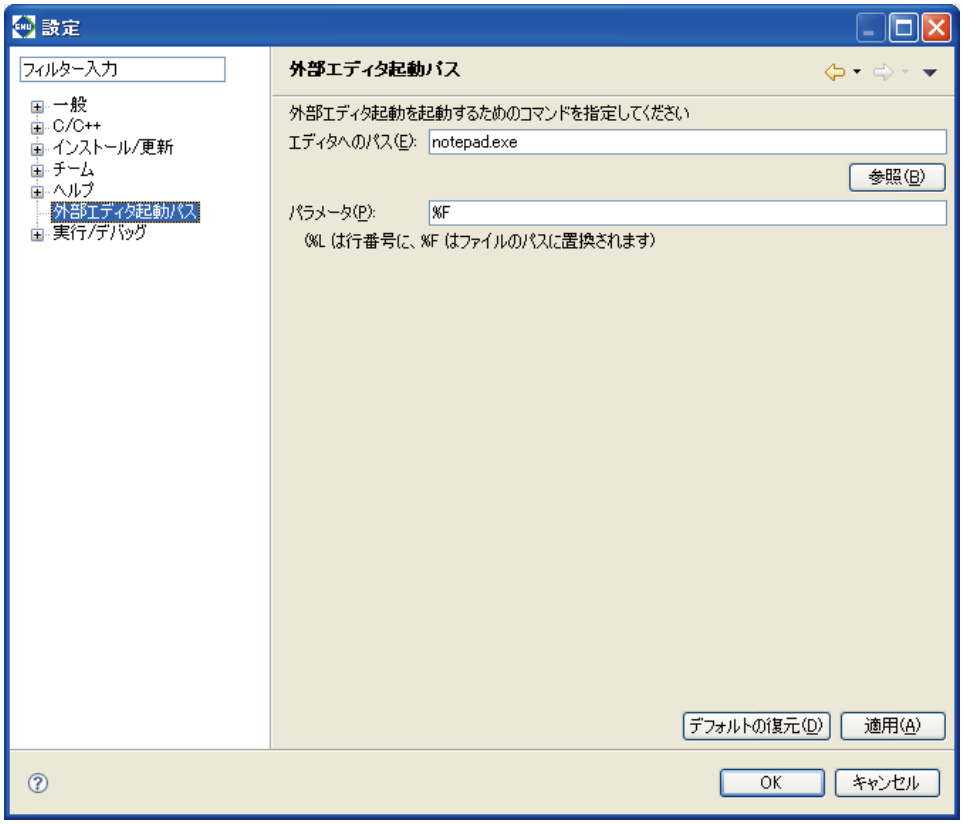
5.5.10 外部エディタを行番号指定で起動するには

●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。



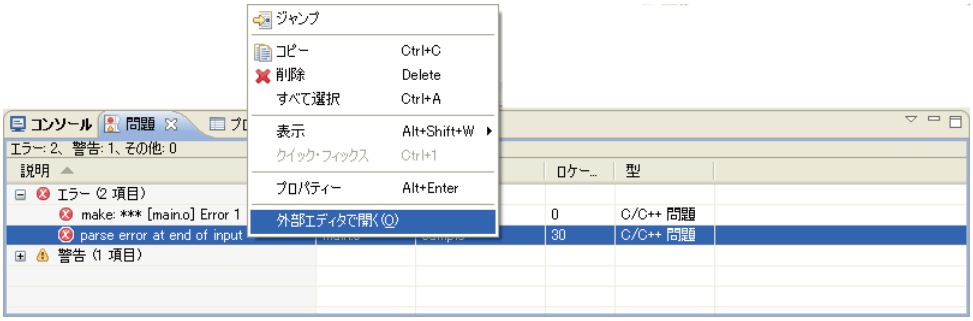
(3)[参照...]ボタンをクリックして、起動したいエディタを選択します。
パス名、ファイル名は、最大255文字まで入力可能です。また、初期状態では"notepad.exe"(メモ帳)が設定されています。

(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。

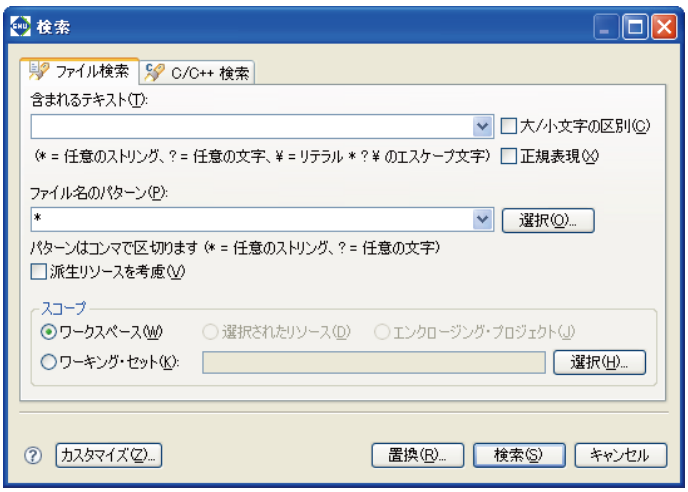
(2) [検索]メニューの[テキスト]を選択し、そのサブメニューから検索範囲(ワークスペース、現在のプロジェクト、現在のファイル、指定のワーキングセット)を選択します。

検索結果は[検索]ビューに表示されます。

5.6.2 ファイル検索

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、また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大文字/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"  "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

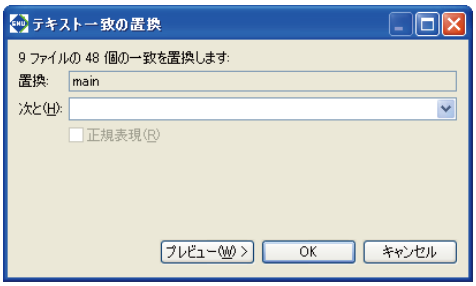
[選択されたリソース] [C/C++ プロジェクト]または[ナビゲーター]ビューで選択されているリソース

[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]

クリックすると[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。

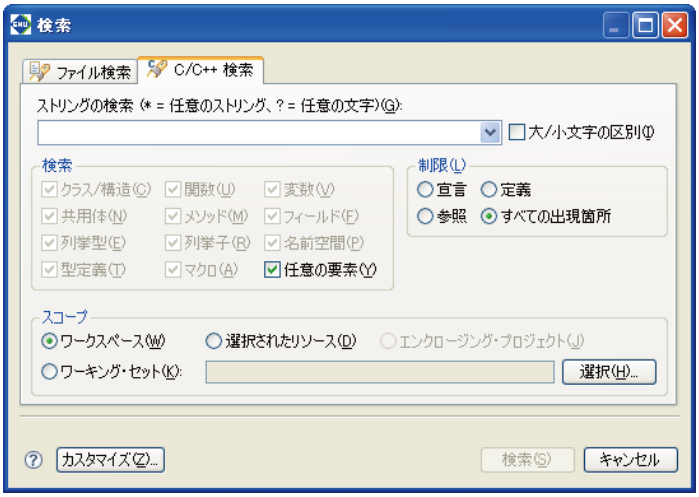
[キャンセル]

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]	構造体
[関数]	グローバル関数(構造体または共用体のメンバ関数を除く)
[変数]	変数(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(構造体または共用体のメンバ)
[フィールド]	フィールド(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(無効)
[型定義]	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。

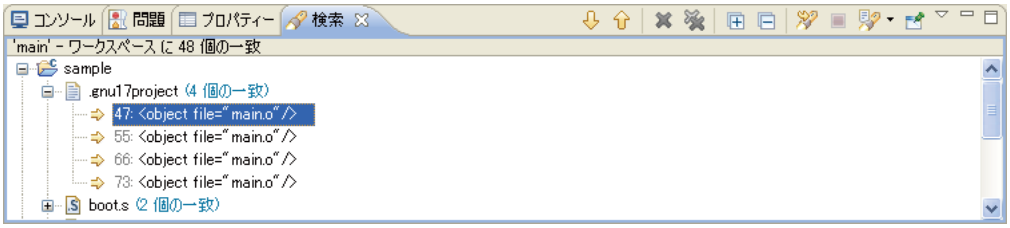
検索結果は[検索]ビューに表示されます。

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

ファイル検索、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。



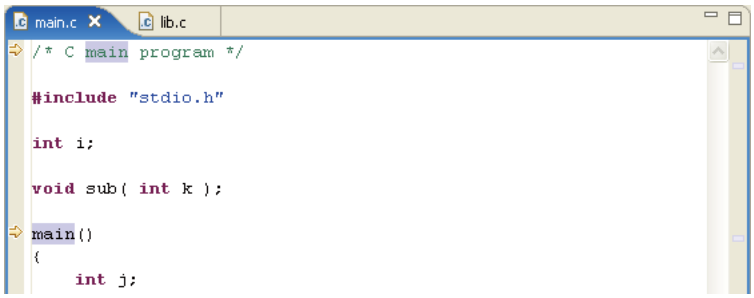
●検索位置の参照

[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、その行のマーカーバーに矢印のマーカーが表示されます。

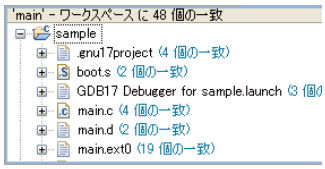


ファイル検索で文字列を空白としてファイル名を検索した場合、検索位置はファイルの先頭となります。ファイル名を検索して[ナビゲーター]ビュー上でファイルを参照したい場合は、[検索]ビュー内のファイル名を選択し、そのコンテキストメニューまたは[ナビゲート]メニューから[表示]>[ナビゲーター]を選択します。[ナビゲーター]ビュー内の該当ファイルが強調表示されます(一覧に表示されている場合のみ)。

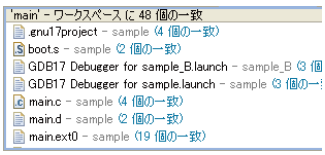
●[検索]ビューの表示形式の変更

[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

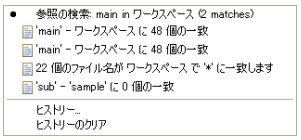


リストで表示



●検索履歴

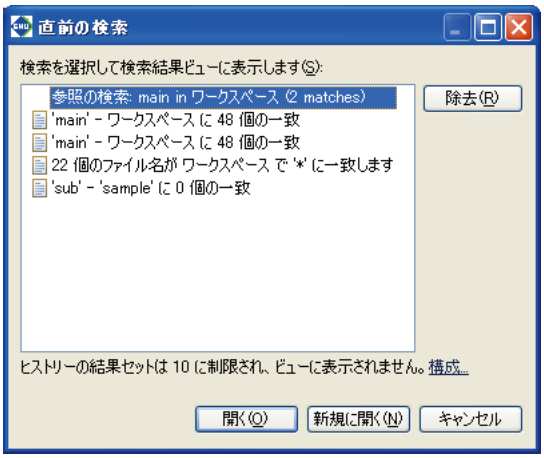
[検索]ビューツールバーの[直前の検索を表示]ショートカットは、これまでに実行したファイル検索、C検索、テキスト検索の一覧を表示します。



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[ヒストリーのクリア]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 GNU17 一般設定の設定

開発するアプリケーションシステムのプロセッサの種類およびメモリ空間のサイズにより、ツールの起動コマンドオプションやリンクするライブラリを切り替えます。そのために、正しいプロセッサの種類とメモリモデル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タイプ及び、メモリモデル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、ターゲットCPUタイプ及び、メモリモデルを変更するプロジェクトを選択します。

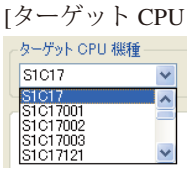
(2) [プロジェクト]メニューまたは上記ビューのコンテキストメニューから[プロパティー]を選択します。
[プロパティー]ダイアログが表示されます。

(3) プロパティの一覧から[GNU17 一般設定]を選択します。

[生成プログラム]にプロジェクトで生成されるプログラムを表示します。(新規プロジェクト作成時に選択したもの)

[ターゲット CPU 機種]と[メモリモデル]に現在選択されているS1C17コア/プロセッサの種類とメモリモデルがそれぞれ表示されます(未選択時のデフォルト設定はS1C17とREGULAR)。

(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。



(5) [メモリモデル]のリストからメモリモデルを選択します。

REGULAR

アドレスサイズ:
24ビット(16Mバイト空間を使用)
アドレス範囲:
0x0 ～ 0xffffff
コンパイラ、アセンブラオプション(makeファイルの記述):
`-mpointer16` 指定なし
`-mshort-offset` 指定なし
ライブラリファイル(makeファイルとリンカスクリプトの記述):
24ビット用ライブラリ

MIDDLE

アドレスサイズ:
20ビット(1Mバイト空間を使用)
アドレス範囲:
0x0 ～ 0xfffff
例えば、プログラムをフラッシュ ROM(開始アドレスが0x80000)にロードする場合、プログラムのサイズの上限は、512KB(0x80000 ～ 0xfffff)となります。このサイズをオーバーするときは、メモリモデルをREGULARに変更してください。
コンパイラ、アセンブラオプション(makeファイルの記述):
`-mpointer16` 指定なし
`-mshort-offset` 指定
ライブラリファイル(makeファイルとリンカスクリプトの記述):
24ビット用ライブラリ

SMALL

アドレスサイズ:
16ビット(64Kバイト空間を使用)
アドレス範囲:
0x0 ～ 0xffff
例えば、プログラムをフラッシュ ROM(開始アドレスが0x8000)にロードする場合、プログラムのサイズの上限は、32KB(0x8000 ～ 0xffff)となります。このサイズをオーバーするときは、メモリモデルをMIDDLEまたは、REGULARに変更してください。
コンパイラ、アセンブラオプション(makeファイルの記述):
`-mpointer16` 指定
`-mshort-offset` 指定
ライブラリファイル(makeファイルとリンカスクリプトの記述):
16ビット用ライブラリ

(6) コプロセッサ用ライブラリをリンクするかどうか選択します(選択可能な機種のみ)。

プログラムで乗算や除算を行う場合にコプロセッサ命令を使用することができます。コプロセッサ使用に際しては、ベクタテーブルの割り込みベクタ3にemu_copro_process()関数を指定する必要があります。(7.2.6コプロセッサ命令対応参照)

[コプロセッサ用ライブラリを使用する]のチェックボックスをON/OFFします。
ON： プロジェクト作成時にコプロセッサライブラリlibgccMD.a(乗除算用)もしくはlibgccM.a(乗算用)をリンクする設定を追加します。
ONの場合、[コプロセッサ用ライブラリの種類]コンボボックスからリンクするライブラリの種類を選択します(機種によっては選択肢が1つしかないものもあります)。
OFF：プロジェクト作成時に通常のエミュレーションライブラリlibgcc.aをリンクする設定を追加します。

選択できない機種の場合、プロジェクト作成時に通常のエミュレーションライブラリlibgcc.aをリンクする設定を追加します。

(7) 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。

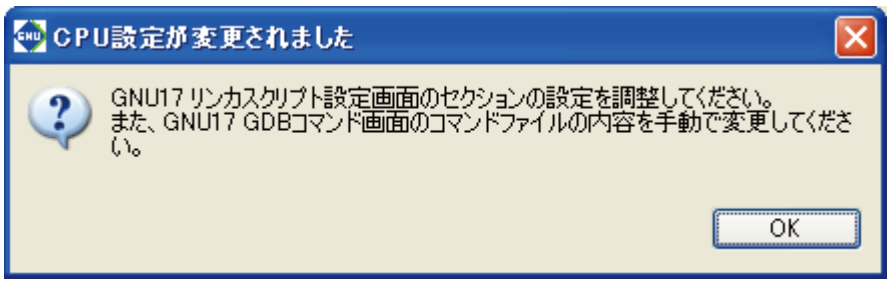
[OK] 変更内容を確定します。設定を変更した場合は、"clean"ビルド(5.7.13節参照)を実行するかどうかを確認するダイアログが表示され、以前の設定で生成されているファイルの削除(およびリビルド)が行えます。
その後、[プロパティー]ダイアログを終了します。

[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。
設定を変更した場合は、"clean"ビルド(5.7.13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。

注：ターゲットCPUを変更した場合、リンカスクリプト設定画面の各セクションの配置とGDBコマンド画面のコマンドファイルの内容を手動で修正する必要がある旨の確認ダイアログが表示されます。



5.7.2 ビルドゴールの設定

GNU17プロジェクトでは、ビルドのゴール(ビルドの最終生成物)を選択できます。

- ROMデータ生成(psa)
  デバッグ終了後の最終成果物としてPSAファイル(ROMデータ)を出力します。
  すべてのプログラム・データは、ROM上に配置されている必要があります。
  ROM上の開き領域は0xFF詰めされたファイルを出力します。
  こちらを選択した場合は、elfファイルも生成されます。
- 実行ファイル生成(elf)
  開発途中のデバッグに使用するプログラムとしてelfファイルを出力します。
  一部のプログラム・データはRAM上に配置してデバッグすることができます。
  こちらを選択した場合は、PSAファイルは生成されません。

デフォルトは、"ROMデータ生成(psa)"が選択されています。ただし、CPUに"S1C17"を選択している場合は、"実行ファイル生成(elf)"固定で変更はできません。

ビルドゴールの設定は、次のように設定します。

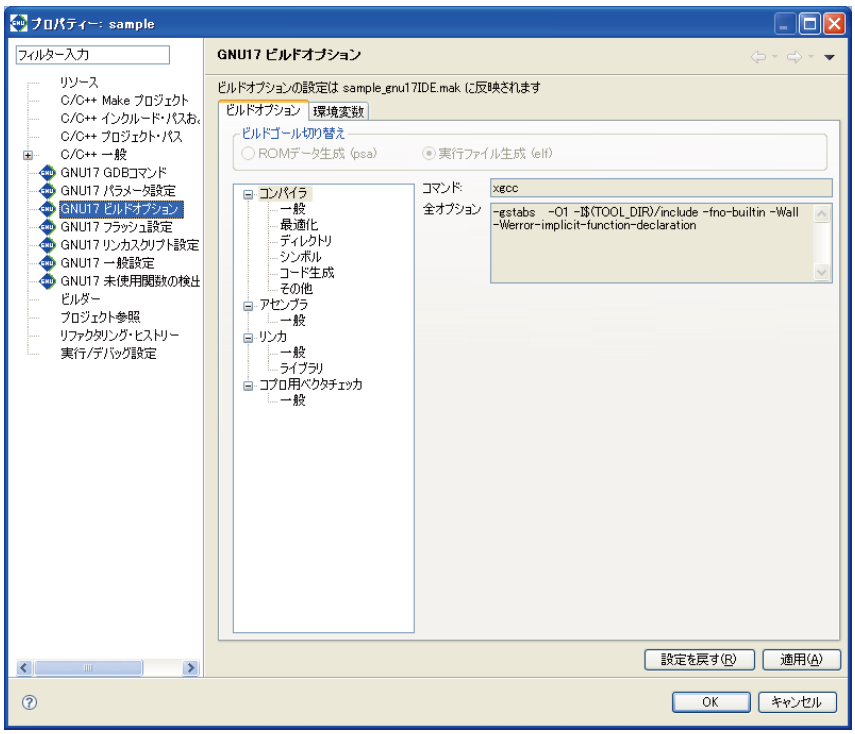
(1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ビルドするプロジェクトを選択します。

(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[プロパティー]を選択します。

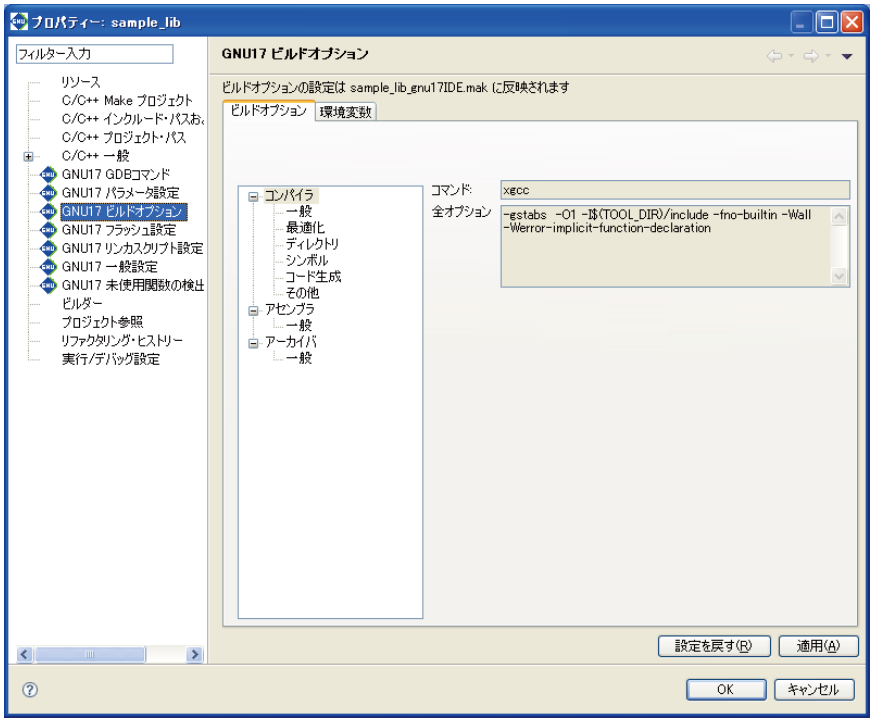
[プロパティー]ダイアログが表示されます。

(3) プロパティの一覧から[GNU17 ビルドオプション]を選択します。

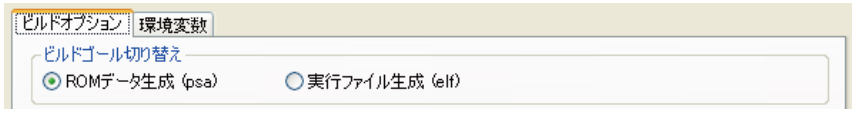
[生成プログラムがアプリケーションの場合]



[生成プログラムがライブラリの場合]



(4) [ビルドゴール切り替え]から[ROMデータ生成(psa)]または[実行ファイル生成(elf)]のいずれかを選択します。



(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(5.7.13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]ボタン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

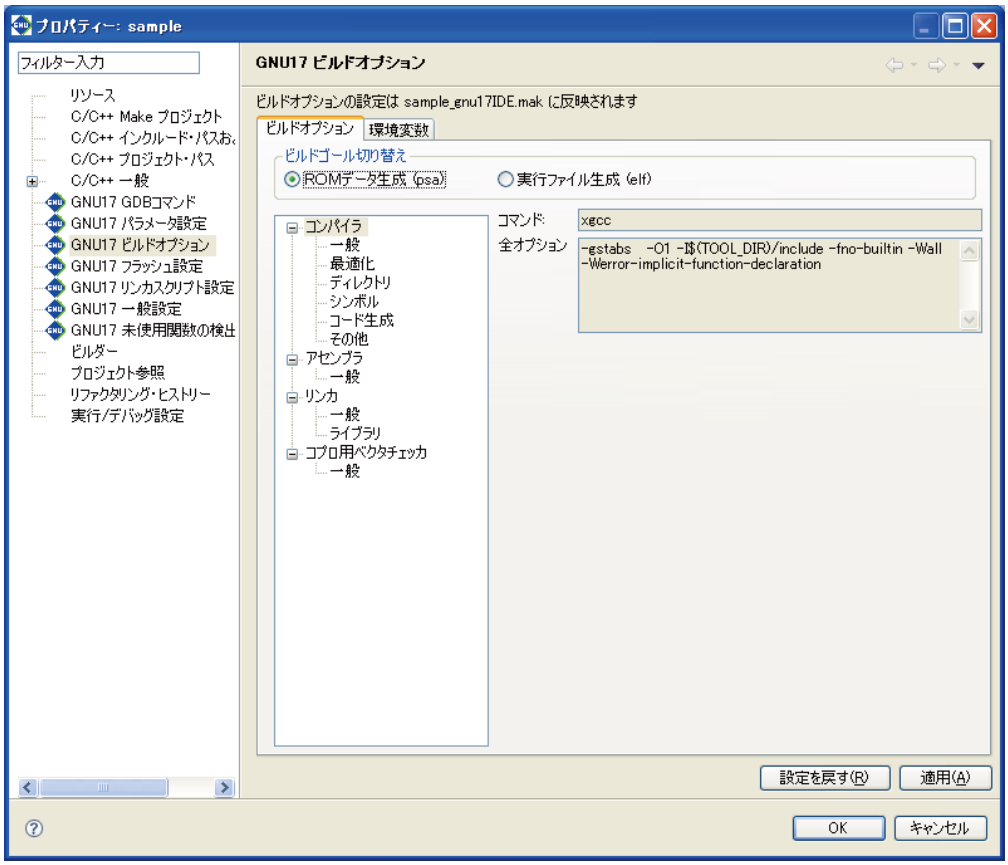
Cコンパイラのコマンドオプションは、次のように設定します。

(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[プロパティー]を選択します。
[プロパティー]ダイアログが表示されます。

(3) プロパティの一覧から[GNU17 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。

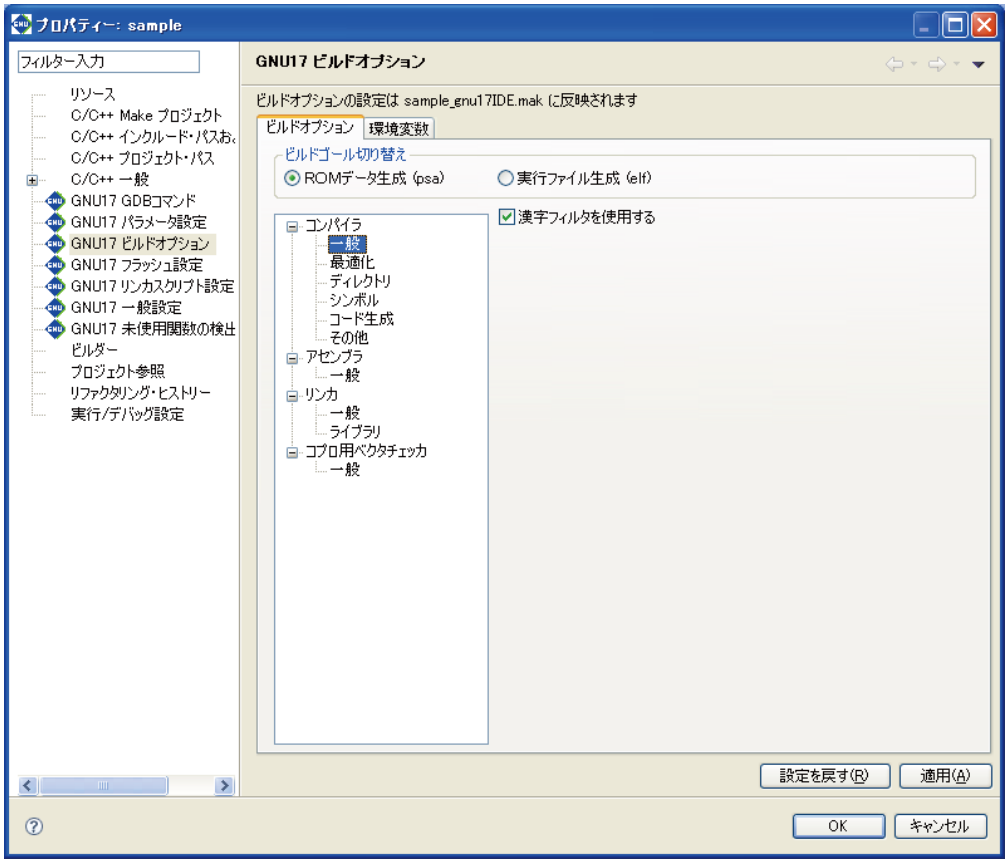
(5) [コンパイラ]のツリーリストからカテゴリを選択し、必要なオプションを設定します。

(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(5.7.13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]ボタン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。

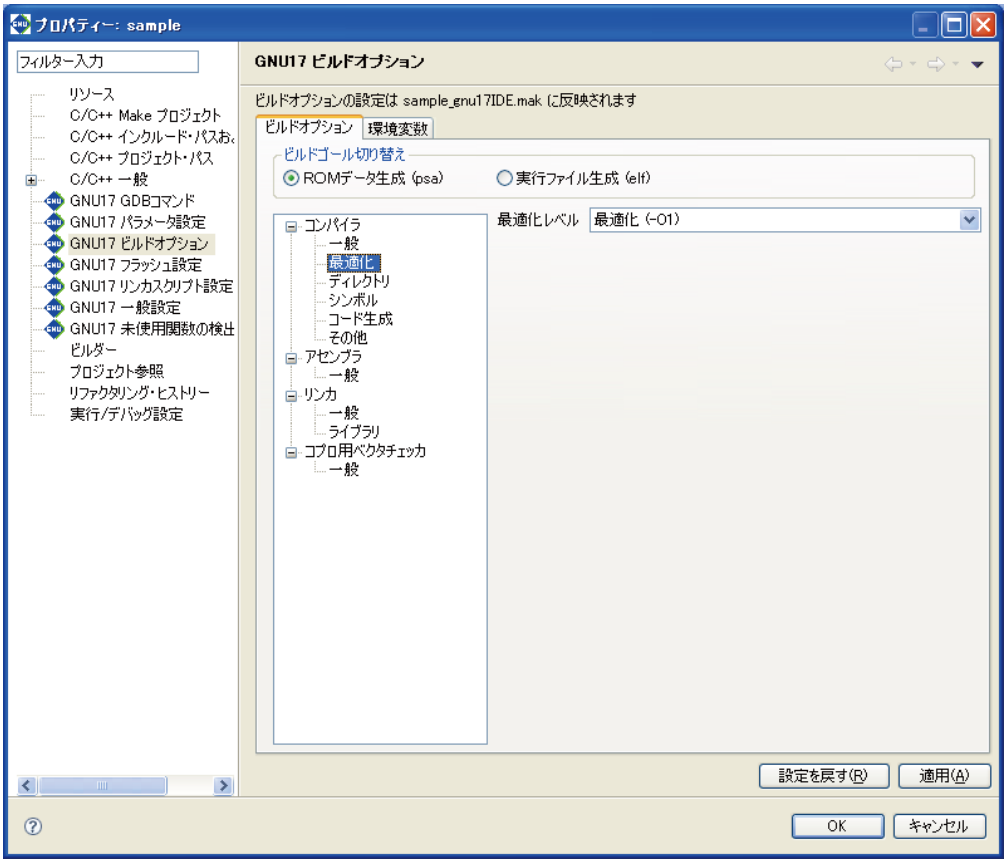
[漢字フィルタを使用する](漢字フィルタの設定)

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)

最適化のレベルを選択します。

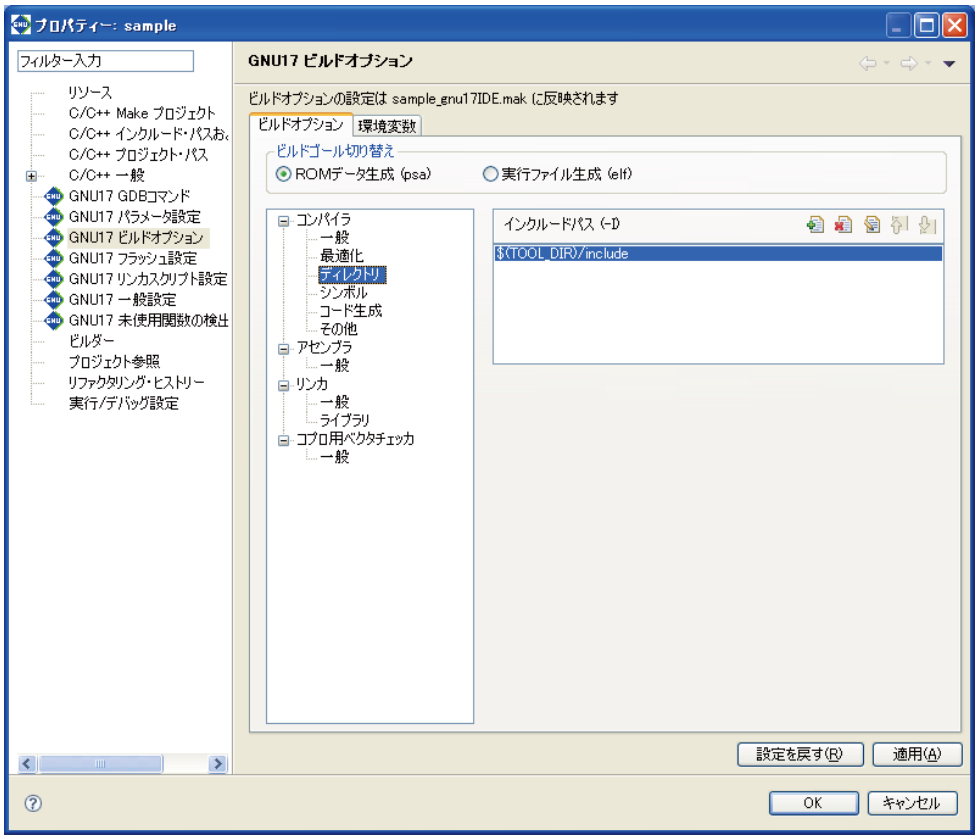
-O0： 最適化を行いません。

-O1： コードのサイズと実行速度の最適化を行います。

-O3： コードの実行速度の最適化を行います。

最適化の詳細については、"6.3.2 コマンドラインオプション"を参照してください。

[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、gnu17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。

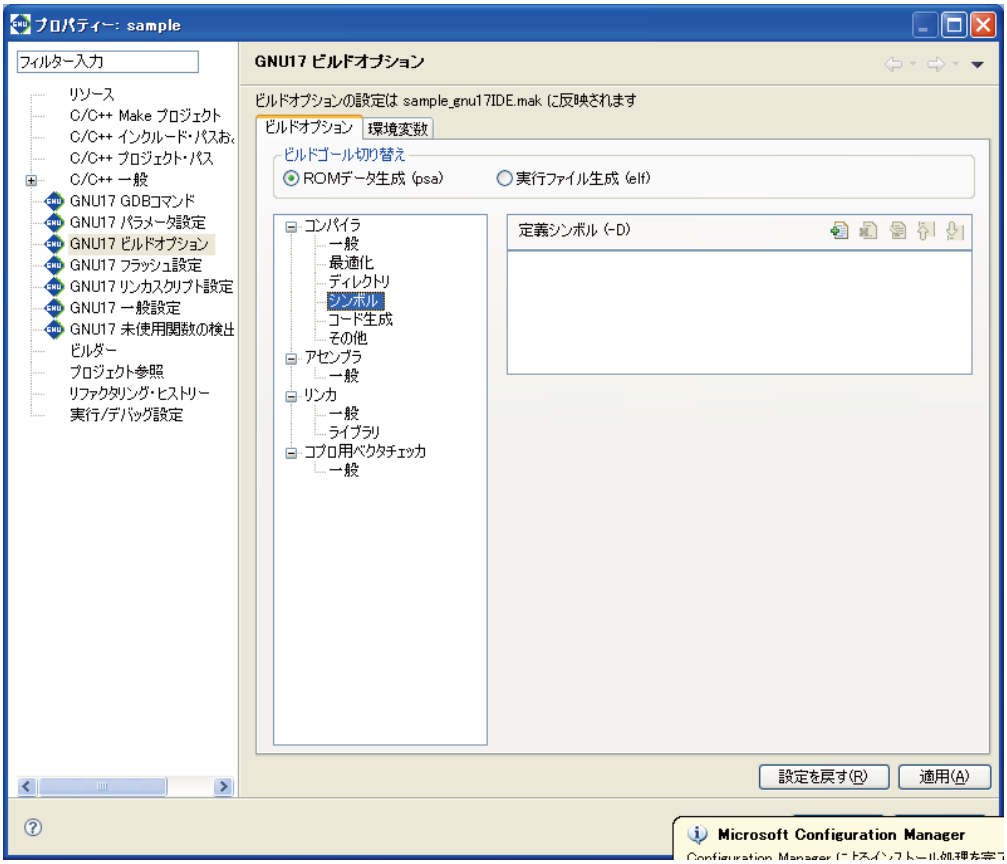
例: gnu17ツールをc:¥EPSON¥gnu17ディレクトリにインストールした場合

```
TOOL_DIR = c:/EPSON/gnu17
```

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/gnu17/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]



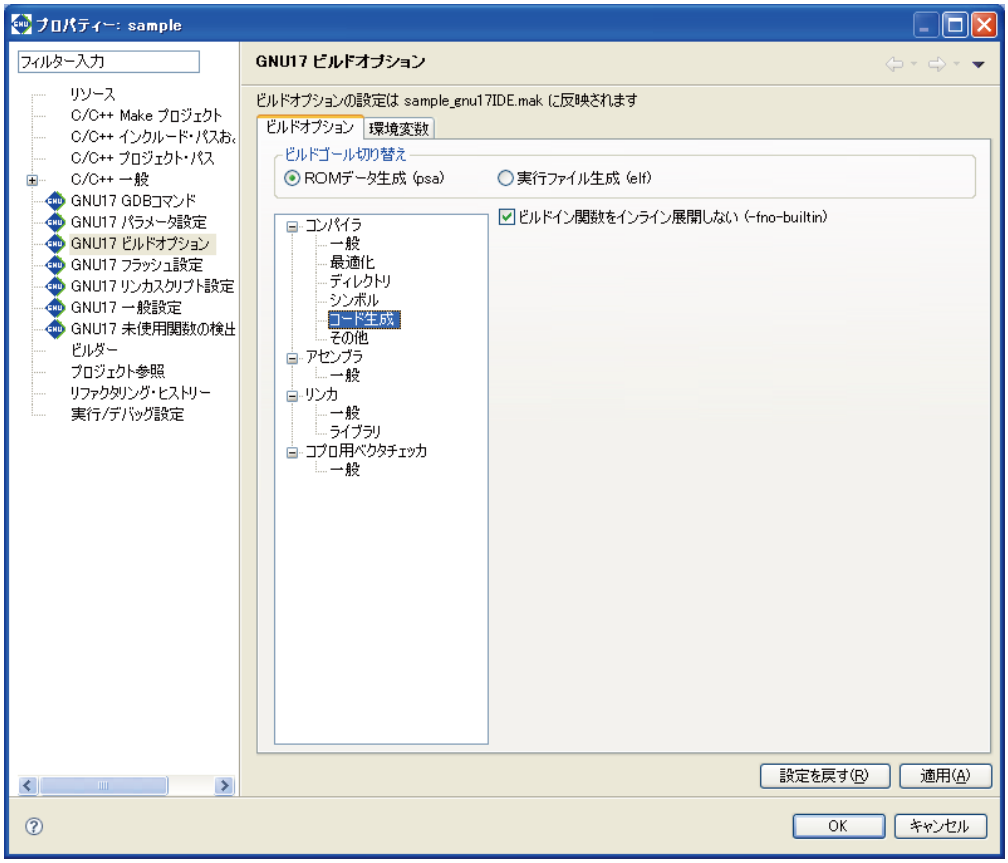
このページではコンパイラのマクロ定義オプションを設定します。

[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。

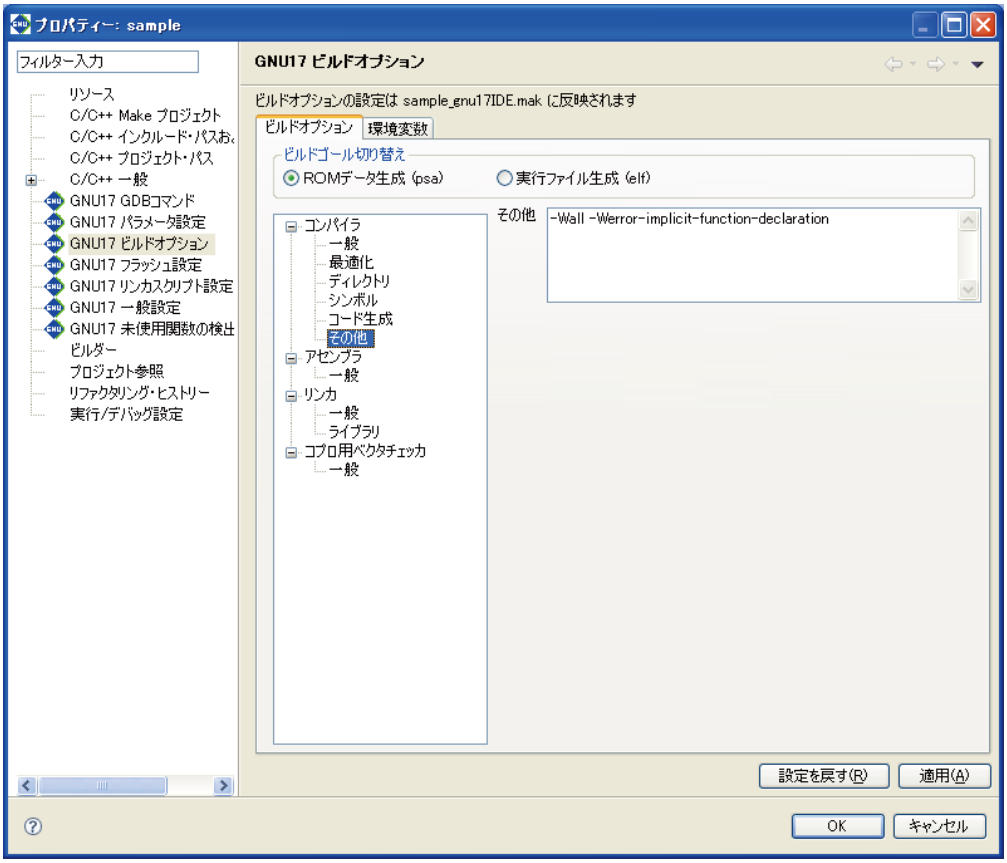
[コード生成]



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。

[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。
各オプションについては、"6.3.2 コマンドラインオプション"を参照してください。

注： IDEではCソースを、-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。

5.7.4 アセンブラオプションの設定

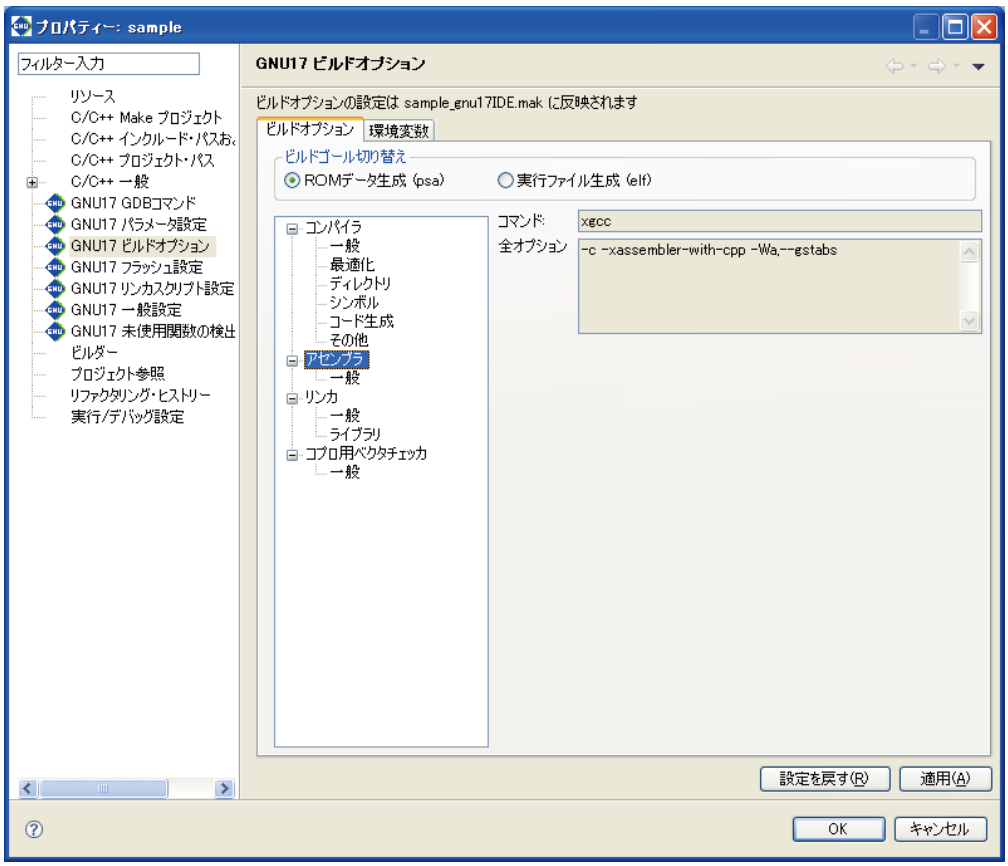
アセンブラのコマンドオプションは、次のように設定します。

(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[プロパティー]を選択します。[プロパティー]ダイアログが表示されます。

(3) プロパティの一覧から[GNU17 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。

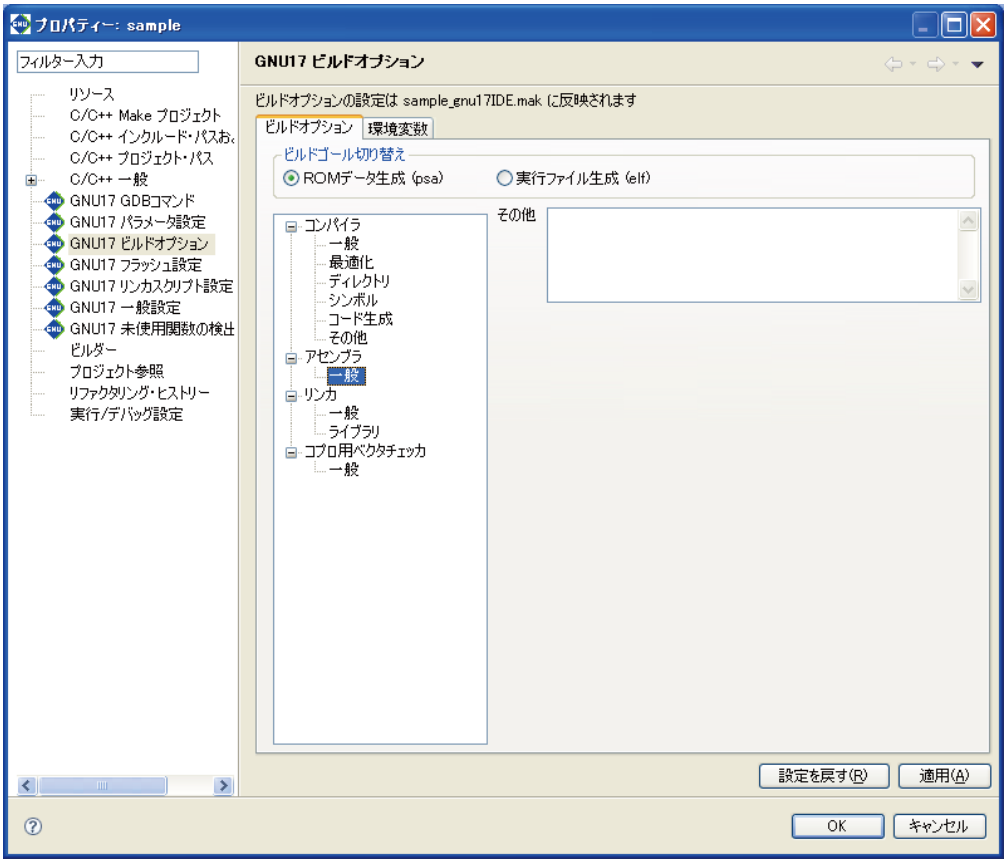
(5) [アセンブラ]のツリーリストから[一般]を選択し、必要なオプションを設定します。

(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(5.7.13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]ボタン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]



アセンブラの-c、-xassembler-with-cpp、-Wa,--gstabsオプション(メモリモデルによっては-mpointer16オプションも)は常に指定されます。
このページではアセンブ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: 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

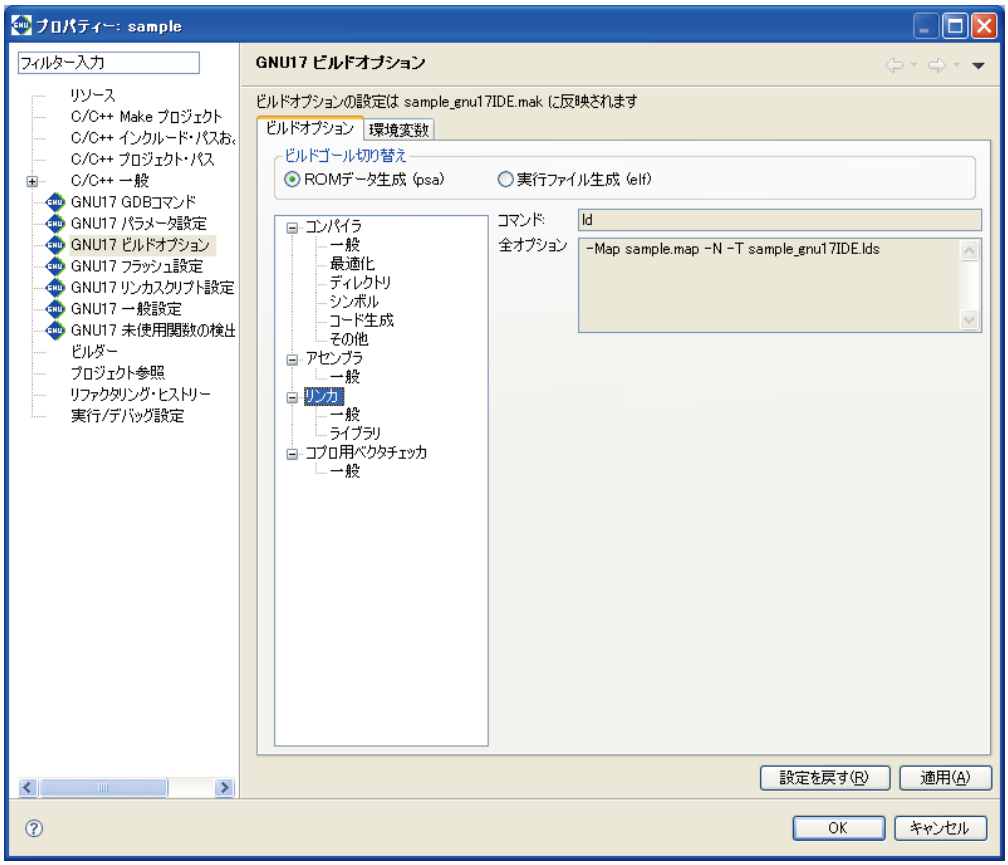
リンカのコマンドオプションは、次のように設定します。
リンカオプションは、生成プログラムがアプリケーション(*.elf/*.psa)の場合のみ設定可能です。

(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、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[プロパティー]を選択します。
[プロパティー]ダイアログが表示されます。

(3) プロパティの一覧から[GNU17 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。

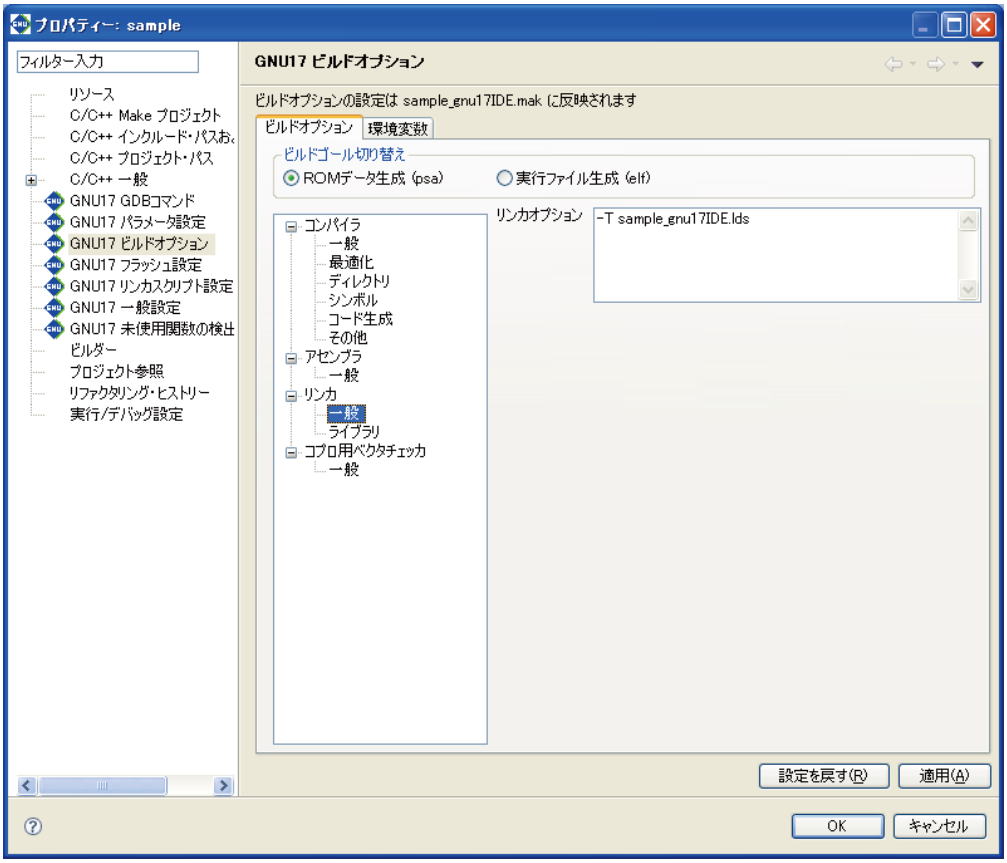
(5) [リンカ]のツリーリストからカテゴリを選択し、必要なオプションを設定します。

(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(5.7.13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]ボタン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。

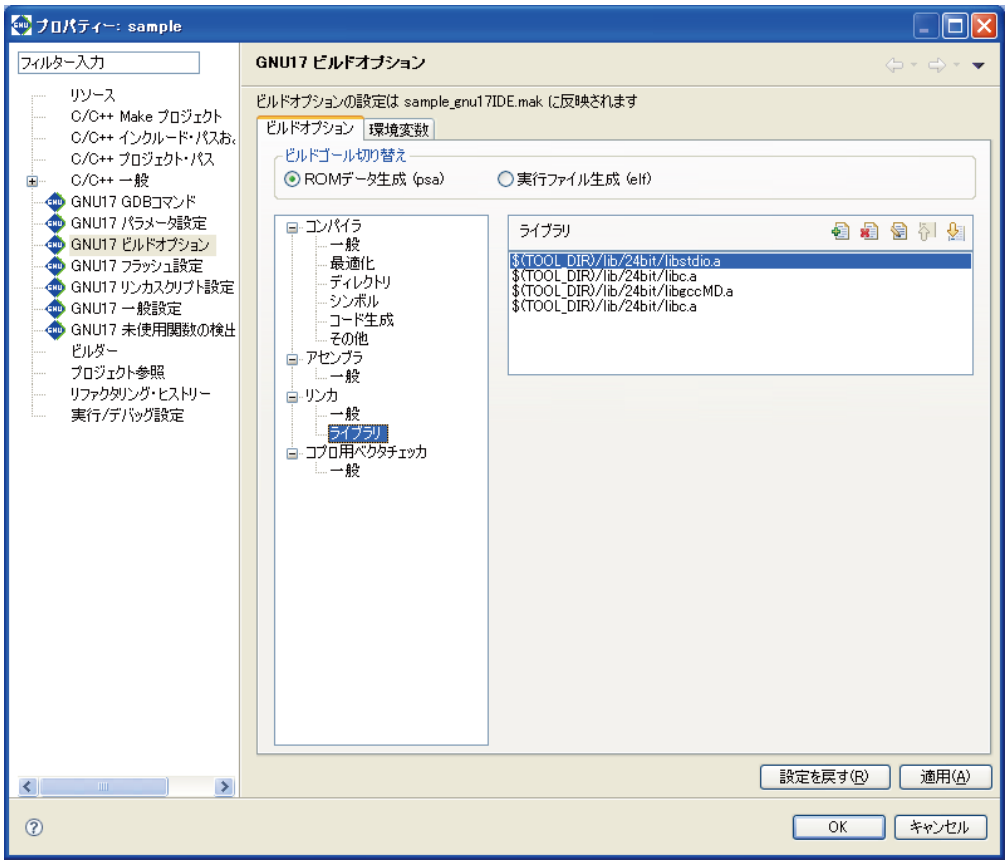
[一般]



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>*_gnu17IDE.lds)
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]

[ライブラリ]のデフォルトは、プロジェクト作成時にコプロセッサ対応ライブラリを使用しないを選択した場合、(libstdio.a, libc.a, libgcc.a, libc.a)となります。プロジェクト作成時にコプロセッサ対応ライブラリを使用するを選択した場合、(libstdio.a, libc.a, libgccM.a or libgccMD.a, ,libc.a)となります。

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、リンカ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。

* デフォルトのライブラリはメモリモデルに合わせ、24ビット版または16ビット版が選択されます。また、libc.aはlibgcc.aとの相互参照を解決するため、2回指定されています。

※$(TOOL_DIR)について

デフォルト設定で、[ライブラリ]には"$(TOOL_DIR)/lib/24bit(16bit)/lib*xxx*.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、gnu17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: gnu17ツールをc:¥EPSON¥gnu17ディレクトリにインストールした場合

```
TOOL_DIR = c:/EPSON/gnu17
```

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/24bit/lib*xxx*.aは-Ic:/EPSON/gnu17/lib/24bit/lib*xxx*.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 アーカイバオプションの設定

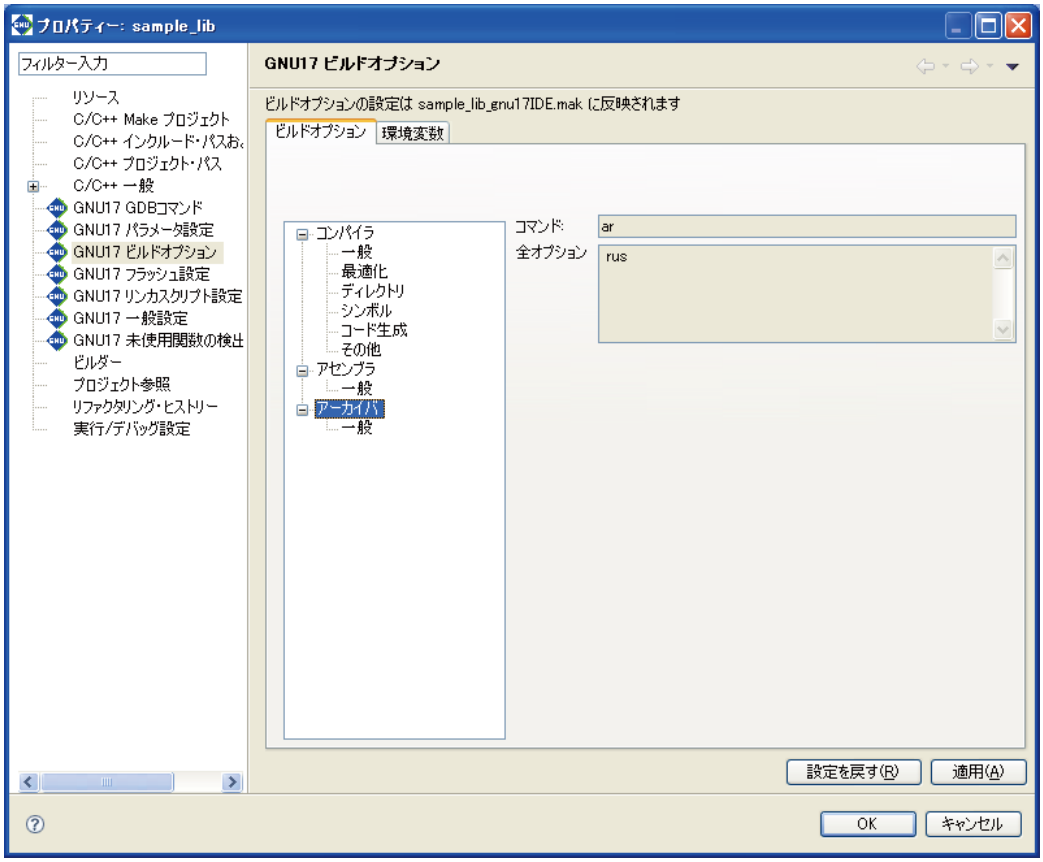
アーカイバのコマンドオプションは、次のように設定します。
アーカイバオプションは、生成プログラムがライブラリ(*.a)の場合のみ設定可能です。

(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、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[プロパティー]を選択します。
[プロパティー]ダイアログが表示されます。

(3) プロパティーの一覧から[GNU17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[アーカイバ]を選択します。



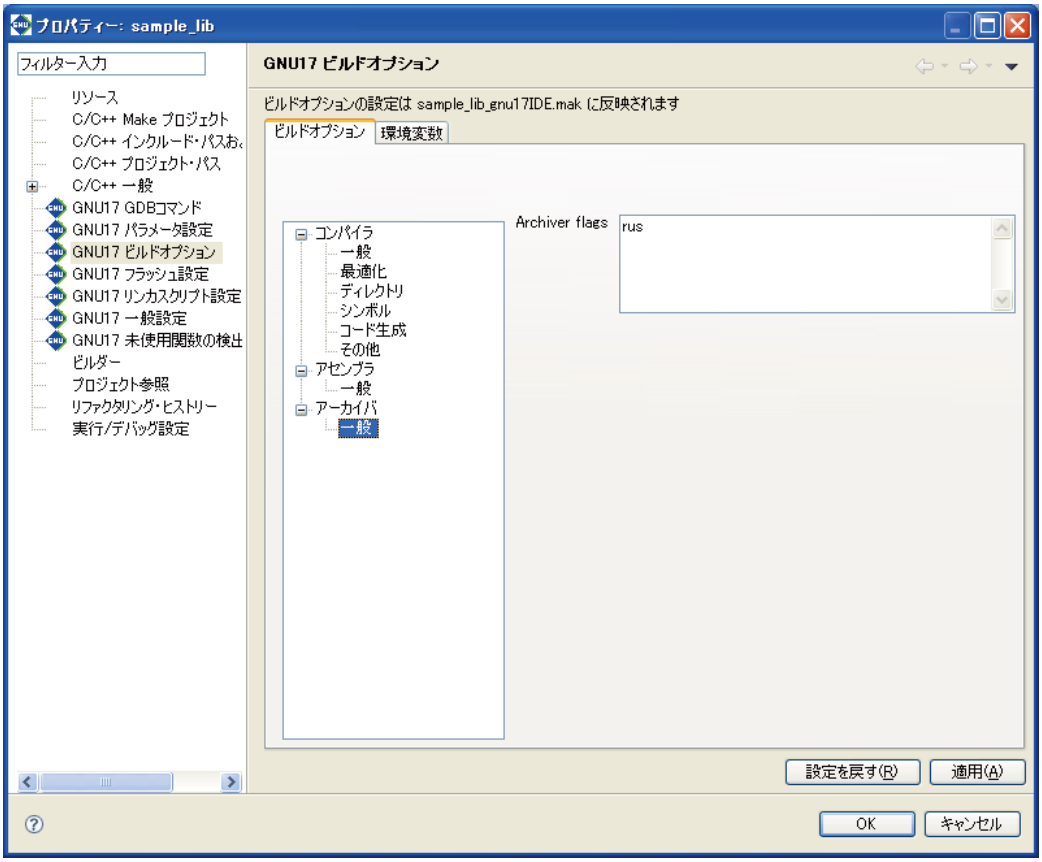
[コマンド:]はアーカイバのプログラム名、[全オプション]は現在設定されているオプションを表示します。

(5) [アーカイバ]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(5.7.13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]ボタン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。

アーカイバ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、"12.5　ar.exe"を参照してください。

[一般]



[アーカイバオプション](デフォルト：rus)
アーカイバオプションを入力します。

5.7.7 ベクタチェッカの設定

ベクタチェッカのオプションを設定します。コプロセッサ用ライブラリを使用する場合に、ベクタテーブルにemu_copro_process割込関数を登録しているかどうか検証するチェッカのON/OFF設定を行います。ベクタチェッカのオプションは、生成プログラムがアプリケーション(*.elf/*.psa)の場合のみ設定可能です。

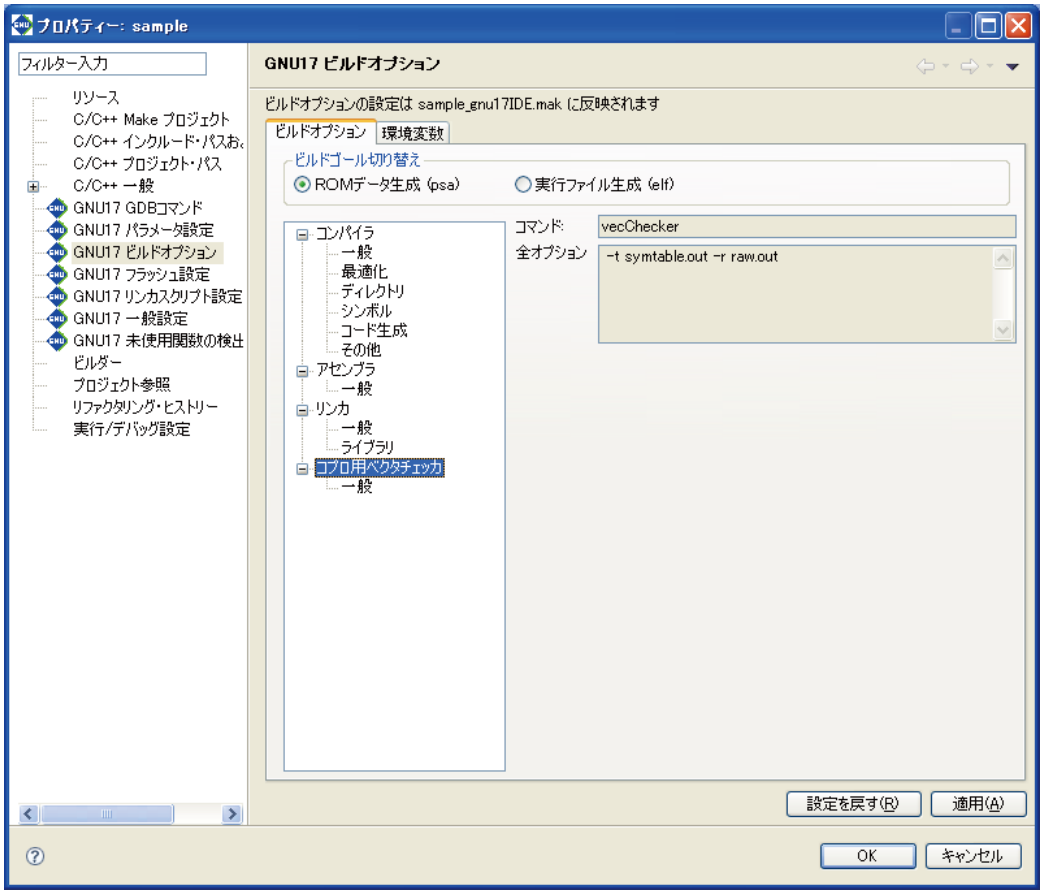
(1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ビルドするプロジェクトを選択します。

(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[プロパティー]を選択します。
[プロパティー]ダイアログが表示されます。

(3) プロパティの一覧から[GNU17 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[コプロ用ベクタチェッカ]を選択します。

[コプロ用ベクタチェッカ]



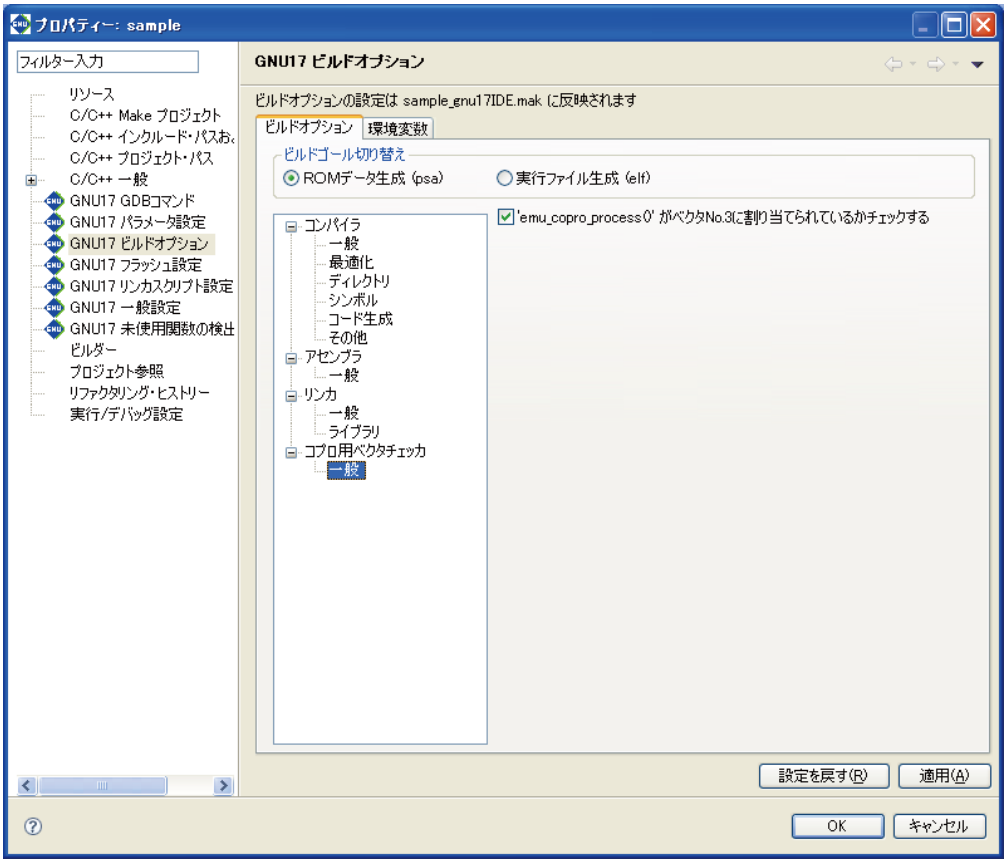
[コマンド:]はベクタチェッカのプログラム名、[全オプション]は現在設定されているオプションを表示します。

(5) [コプロ用ベクタチェッカ]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(5.7.13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]ボタン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。

ベクタチェッカオプションの設定ページを以下に示します。

[一般]



['emu_copro_process()'がベクタNo.3に割り当てられているかチェックする]
(デフォルト：ON　ただしコプロセッサライブラリ使用時)
ベクタテーブルにemu_copro_process割込関数を登録しているかどうか検証するチェッカのON/OFF設定を行います。
ONを選択した場合は、ビルド時にベクタチェッカが起動され、ベクタNo.3にemu_copro_process()割込関数が登録されているかどうかチェックします。
(elf実行ファイル内の.vectorセクション開始アドレスとしてベクタNo.3のデータが検索されます。elfが.vectorセクションを持たない場合はチェックされません。)

ベクタチェッカの-tおよび-sオプションは常に固定で指定されます。

5.7.8 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-17IDE.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。

●生成プログラムがアプリケーション(*.elf/*.psa)の場合

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 Gnu17 Plug-in for Eclipse
# This file should be placed directly under the project folder

# macro definitions for target file                                          (1)
TARGET= sample
GOAL= $(TARGET).psa

# export environment variable
export CYGWIN=nodosfilewarning

# macro definitions for tools                                                (2)
TOOL_DIR= C:/EPSON/GNU17
CC= $(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
CC_KFILT= $(TOOL_DIR)/xgcc_filt
OBJDUMP= $(TOOL_DIR)/objdump
OBJCOPY= $(TOOL_DIR)/objcopy
MOTO2FF= $(TOOL_DIR)/moto2ff
SCONV= $(TOOL_DIR)/sconv32
VECCHECKER= $(TOOL_DIR)/vecChecker

# macro definitions for tool flags                                           (3)
CFLAGS= -B$(TOOL_DIR)/ -gstabs -S  -O1 -I$(TOOL_DIR)/include -fno-builtin -Wall
-Werror-implicit-function-declaration
ASFLAGS= -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs
ASFLAGS_CC=
LDFLAFGS_NOOVERLAP= -Map sample.map -N -T sample_gnu17IDE.lds -c17-overlap-noerr
-c17-memoryover-noerr
LDFLAGS=  -Map sample.map -N -T sample_gnu17IDE.lds
EXTFLAGS= -Wa,-mc17_ext -Wa,$(TARGET).dump -Wa,$(TARGET).map
EXTFLAGS_CC= -mc17_ext $(TARGET).dump $(TARGET).map
OBJDUMPFLAGS= -t
OBJCOPYFLAGS= -I elf32-little -O srec --srec-forceS3
MOTOSTART= 8000
MOTOSIZE= 10000
MOTOPROGSIZE= 10000
SCONVFLAGS= S2
VECCHECKERFLAGS= -t symtable.out -r raw.out
VECCHECKER_ON= false

# macro for switching 2pass or 1pass build                                   (4)
PASS= 2pass

# macro definitions for tool flags
PROTECT_ON= true

# search paths for source files
vpath %.c
vpath %.s
```

```
# macro definitions for object files                                    (5)
OBJS= boot.o ¥
      lib.o ¥
      main.o ¥
      sys.o ¥

# macro definitions for library files
OBJLDS= $(TOOL_DIR)/lib/24bit/libstdio.a ¥
        $(TOOL_DIR)/lib/24bit/libc.a ¥
        $(TOOL_DIR)/lib/24bit/libgcc.a ¥
        $(TOOL_DIR)/lib/24bit/libc.a ¥

# macro definitions for assembly files generated from c source files    (6)
CEXTTEMPS= lib.ext0 ¥
      main.ext0 ¥
      sys.ext0 ¥


# macro definitions for dependency files                                (7)
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                       (8)
DEPCMD_CC= @$(CC) -M -MG $(CFLAGS) $< | $(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                                              (9)
.PHONY : all clean

all : $(GOAL)                                                           (10)

$(TARGET).psa : $(TARGET).elf
# clean psa files
  $(RM) -f $(TARGET).sa $(TARGET).saf $(TARGET).psa
# create psa file from elf
  $(OBJCOPY) $(OBJCOPYFLAGS) $< $(TARGET).sa
  $(MOTO2FF) $(MOTOSTART) $(MOTOSIZE) $(TARGET).sa
  $(SCONV) $(SCONVFLAGS) $(TARGET).saf $(TARGET).psa

# create protected psa file                                             (11)
  ifeq ($(PROTECT_ON), true)
  $(TOOL_DIR)/gdb.exe --nw --command=protect.cmd
  $(SCONV) $(SCONVFLAGS) temp $(TARGET)_ptd.psa
  $(RM) -f temp
  endif
  @cmd /c "echo ---------------- Finished building target : $@ ----------------"

$(TARGET).elf : $(OBJS) sample_gnu17IDE.mak sample_gnu17IDE.lds
ifeq ($(PASS), 1pass)
# 1pass linking
  $(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS)
else
# 1pass linking
  -$(LD) $(LDFLAGS_NOOVERLAP) -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
```

```
# 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" >obj1pass¥¥temp ; 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)
@if [ -s lderr ]; then ¥
  cmd /c "type lderr" ¥
  && $(RM) -f $(TARGET).elf ¥
  && exit 1; ¥
  else $(RM) -f lderr ; ¥
fi
# restore 1pass object files
  $(RM) -f $(OBJS) ¥
  && ¥
  for NAME in $(subst /,¥¥,$(OBJS)) ; do ¥
         cmd /c "copy /y obj1pass¥¥$$NAME $$NAME" >obj1pass¥¥temp ; done ¥
  && cmd /c "rd /s /q obj1pass"
endif

# check copro function in vector                                                    (12)
ifeq ($(VECCHECKER_ON), true)
  $(RM) -f symtable.out raw.out
  $(OBJDUMP) -t $@ > symtable.out
  $(OBJDUMP) -s $@ > raw.out
  $(VECCHECKER) -t symtable.out -r raw.out
endif

  @cmd /c "echo ---------------- Finished building target : $@ -----------------"

## boot.s                                                                           (13)
boot.o : boot.s
  $(AS) $(ASFLAGS) -o $@ $<
  $(DEPCMD_AS)

## lib.c
lib.o : lib.c lib.ext0
  $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
  $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
  $(DEPCMD_CC)

## main.c
main.o : main.c main.ext0
  $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
  $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
  $(DEPCMD_CC)

## sys.c
sys.o : sys.c sys.ext0
  $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
  $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
  $(DEPCMD_CC)


# dependecies for assembled c source files
```

```
lib.ext0 : lib.c
main.ext0 : main.c
sys.ext0 : sys.c

# include dependency files
-include $(DEPS)

# clean files                                                                    (14)
clean :
  $(RM) -f $(OBJS) $(TARGET).elf $(TARGET).map $(DEPS) $(CEXTTEMPS) $(TARGET).dump
lderr $(TARGET).sa $(TARGET).saf $(TARGET).psa $(TARGET)_ptd.psa
  @if [ -e obj1pass ]; then ¥
         cmd /c "rd /s /q obj1pass" ; ¥
  fi
```

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。
最終生成物(.psaもしくは.elf)が、GOALとして定義されます。

(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、S5U1C17001C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([GNU17 ビルドオプション])の選択内容が反映されます。

(4) "OBJS="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。

(5) "OBJLDS="の後には、[GNU17 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"OBJS="と同様に記述されます。

(6) 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。

(7) 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。

(8) 依存リスト内に置く実行コマンドを定義しています。

(9) ターゲットを定義しています。

(10) 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。
実行形式のオブジェクトファイルは2パスメイクにより、拡張命令が最適化されて出力されます。
また、実行形式のオブジェクトファイル(elf)まで生成するか、Sレコード形式のPSAファイル(ROMデータ)まで生成するかは、GOALの設定によります。

(11) フラッシュプロテクトがONになっている場合、プロテクトされた(.psa)ファイルを生成します。

(12) ベクタチェッカの実行コマンドを定義しています。

(13) 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。

(14) 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。

●生成プログラムがライブラリ(*.a)の場合

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 Gnu17 Plug-in for Eclipse
# This file should be placed directly under the project folder

# export environment variable
export CYGWIN=nodosfilewarning

# macro definitions for target file                                          (1)
TARGET= sample
GOAL= $(TARGET).a

# macro definitions for tools                                                (2)
TOOL_DIR= C:/EPSON/GNU17
CC= $(TOOL_DIR)/xgcc
AS= $(TOOL_DIR)/xgcc
AS_CC= $(TOOL_DIR)/as
RM= $(TOOL_DIR)/rm
SED= $(TOOL_DIR)/sed
AR= $(TOOL_DIR)/ar

# macro definitions for tool flags                                           (3)
CFLAGS= -B$(TOOL_DIR)/ -mno-sjis-filt -gstabs -S  -O1 -I$(TOOL_DIR)/include -fno-
builtin -Wall -Werror-implicit-function-declaration
ASFLAGS= -B$(TOOL_DIR)/ -mno-sjis-filt -c -xassembler-with-cpp -Wa,--gstabs
ASFLAGS_CC=
ARFLAGS=  rus

# search paths for source files
vpath %.c
vpath %.s

# macro definitions for object files                                         (4)
OBJS= sub1.o ¥
      sub2.o ¥
      sub3.o ¥

# macro definitions for assembly files generated from c source files         (5)
CEXTTEMPS= sub2.ext0 ¥
           sub3.ext0 ¥


# macro definitions for dependency files                                     (6)
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                            (7)
DEPCMD_CC= @$(CC) -M -MG $(CFLAGS) $< | $(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                                                   (8)
.PHONY : all clean

all : $(GOAL)
```

```
$(TARGET).a : $(OBJS) sample_gnu17IDE.mak                                    (9)
$(AR) $(ARFLAGS) $@ $(OBJS)
 @cmd /c "echo ---------------- Finished building target : $@ ----------------"

## sub1.s                                                                    (10)
sub1.o : sub1.s
 $(AS) $(ASFLAGS) -o $@ $<
 $(DEPCMD_AS)

## sub2.c
sub2.o : sub2.c sub2.ext0
 $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
 $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
 $(DEPCMD_CC)

## sub3.c
sub3.o : sub3.c sub3.ext0
 $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
 $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
 $(DEPCMD_CC)


# dependecies for assembled c source files
sub2.ext0 : sub2.c
sub3.ext0 : sub3.c

# include dependency files
-include $(DEPS)

# clean files                                                               (11)
clean :
 $(RM) -f $(OBJS) $(TARGET).a $(DEPS) $(CEXTTEMPS)
```

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a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、アーカイバ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、S5U1C17001C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、アーカイバ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([GNU17 ビルドオプション])の選択内容が反映されます。

(4) "OBJS="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。

(5) アセンブラソースファイルが記述されます。
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。

(6) 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。

(7) 依存リスト内に置く実行コマンドを定義しています。

(8) ターゲットを定義しています。

(9) 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。

(10) 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。

(11) 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、"12.1 make.exe"を参照してください。

5.7.9 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>*_gnu17IDE.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、"3.8 セクションとリンク"および"9 リンカ"を参照してください。

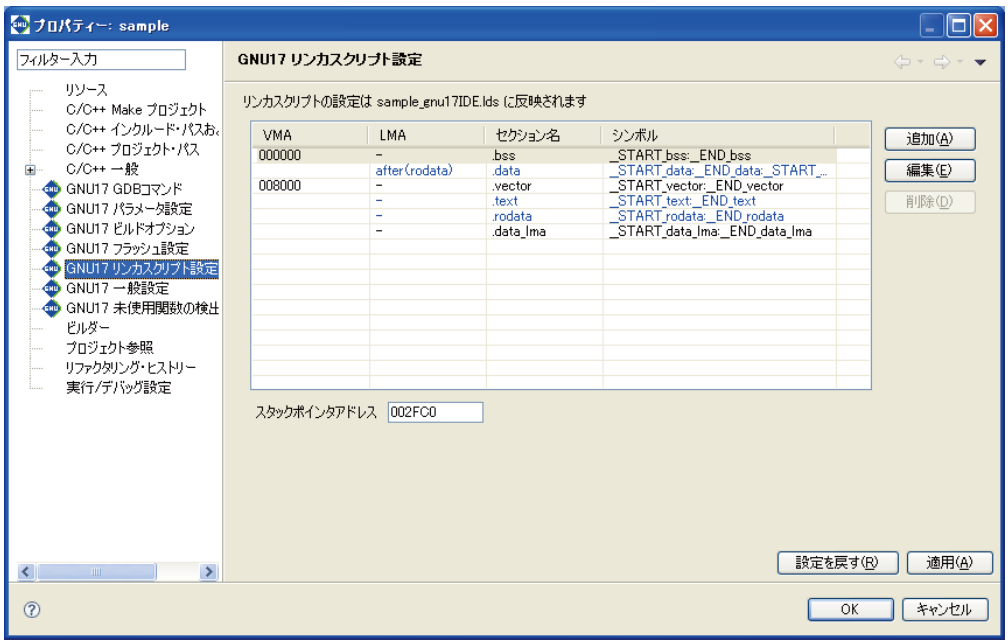
●リンカスクリプト設定ページ

リンカスクリプトは、プロジェクトプロパティの[GNU17 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。

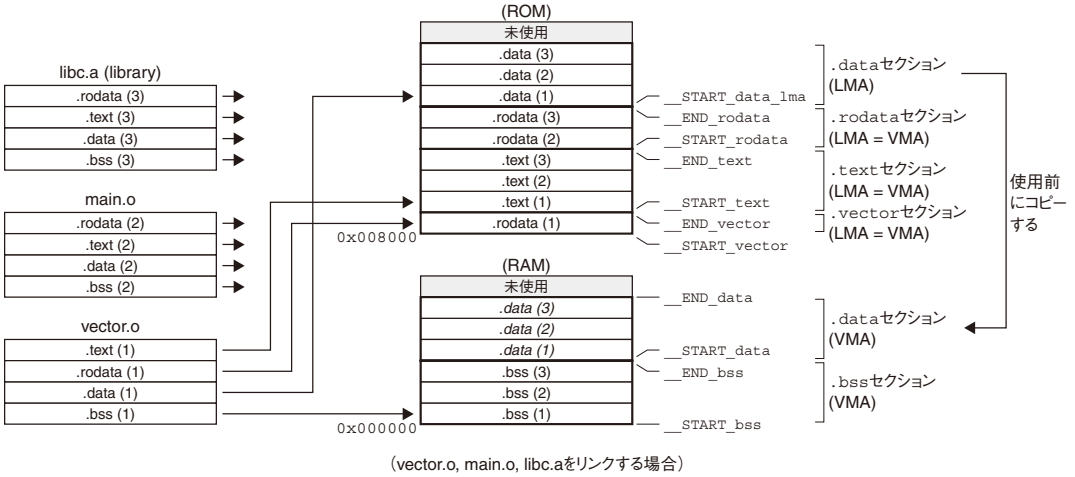
(2) [プロジェクト]メニューまたは上記ビューのコンテキストメニューから[プロパティー]を選択します。[プロパティー]ダイアログが表示されます。

(3) プロパティの一覧から[GNU17 リンカスクリプト設定]を選択します。



注：生成プログラムがライブラリの時は、リンカスクリプトの編集はできません。

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5.7.9.1に示します。



（vector.o, main.o, libc.aをリンクする場合）

図5.7.9.1 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6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）
.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）
.data_lma	.dataの仮想セクション（.dataにある変数の初期値。通常は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されることを表します。

[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(.rodata)"は( )内のセクション、この場合は.rodata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トラップ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選択できます。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。
[ターゲット CPU 機種]の機種によって設定アドレスが変わります(例　008000や020000など)。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ガ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。
また、.vectorセクションに配置している時に[プロパティ]>[GNU17 一般設定]ページにおいて、CPUを変更した場合、そのCPUのベクタアドレスが、.vectorセクションのアドレスに配置されます。

.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ソースの場合(vector.c)

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,              // 0x0    0
        (unsigned long)addr_err,          // 0x4    1
        (unsigned long)nmi,               // 0x8    2
                              :
        (unsigned long)dummy,             // 0x48   18
        (unsigned long)dummy              // 0x4c   19
    };
```

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                  ; 0x0    0
    .long   ADDR_ERR              ; 0x4    1
    .long   NMI                   ; 0x8    2
                  :
    .long   DUMMY                 ; 0x48   18
    .long   DUMMY                 ; 0x4c   19
```

ベクタテーブルが.textセクションに記述されている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) [GNU17 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。

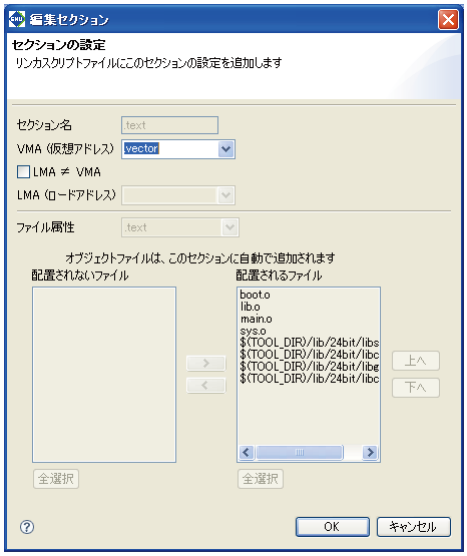
(2) [編集]ボタンをクリックします。
[編集セクション]ダイアログが表示されます。

(3) 下記の説明に従って必要な変更を行い、[OK]ボタンをクリックします。

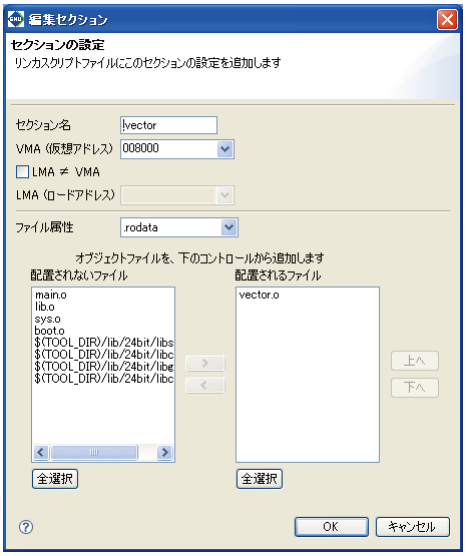
(4) 他のセクションまたはプロパティを変更する場合は[適用]ボタンを、プロパティの設定を終了する場合は[OK]ボタンをクリックします。
[適用]ボタンをクリックする前であれば、[設定を戻す]ボタンによって、変更した内容をこのページを開いたときの状態に戻すことができます。

[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]
ユーザセクション名(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。**
- **セクション名として最後に_lma(大文字/小文字共に)を付けることはできません。**

[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、6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、6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で表示されます。

[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は、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。

[すべて選択]

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)

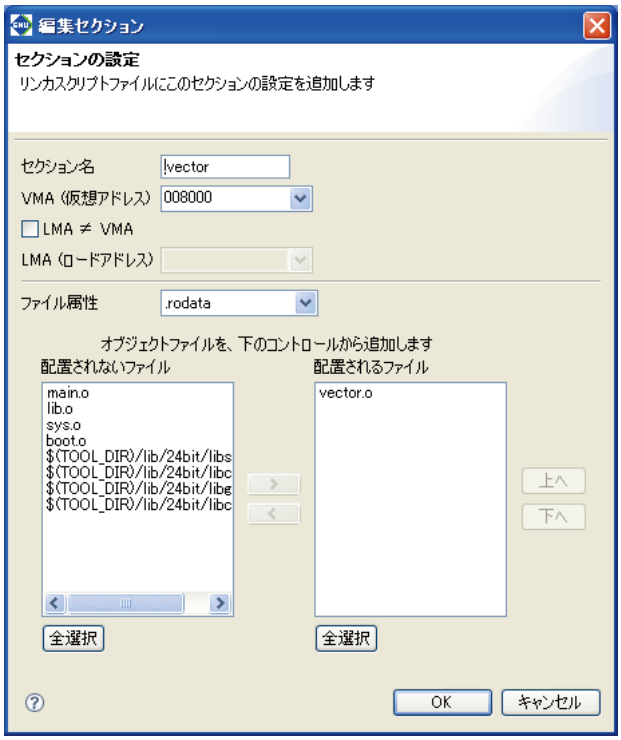
セクション情報の編集例として、新規プロジェクトウィザードのデフォルト設定で作成された.vectorセクション情報を変更します。

セクション属性: .rodata → .text

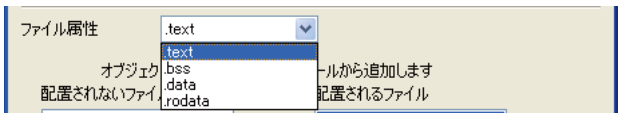
オブジェクト: vector.o → boot.o

(1)プロジェクトプロパティの[GNU17 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。

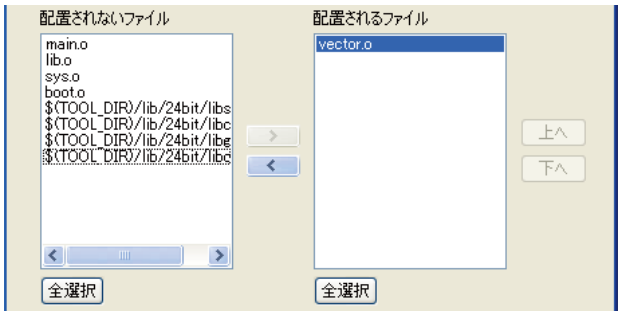
[編集セクション]ダイアログが開きます。



(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。

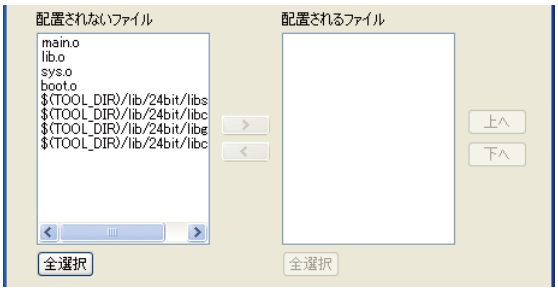


(3)[配置されるファイル]からvector.oを選択し、[<]ボタンをクリックします。

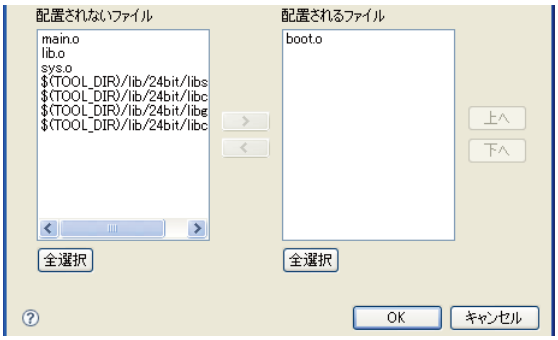


vector.oはソースファイルがないため、[配置されないファイル]には移動せずに削除されます。この操作のみで、vector.oは他のセクション情報からも削除されます。

(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。

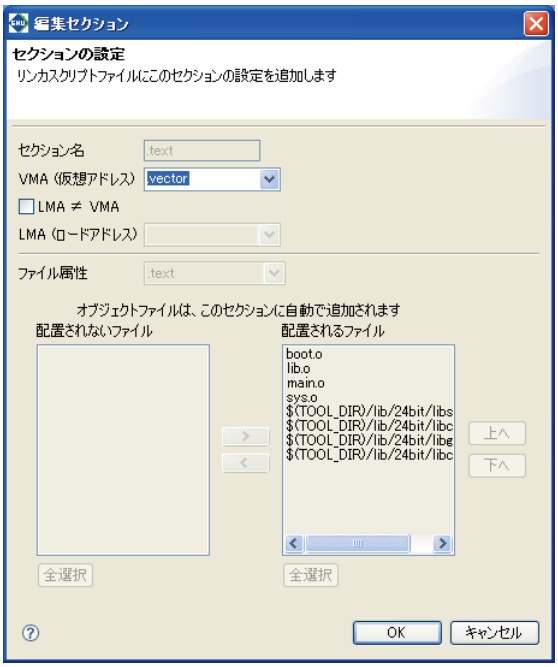


boot.oが[配置されるファイル]に移動します。



(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、リンカ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。

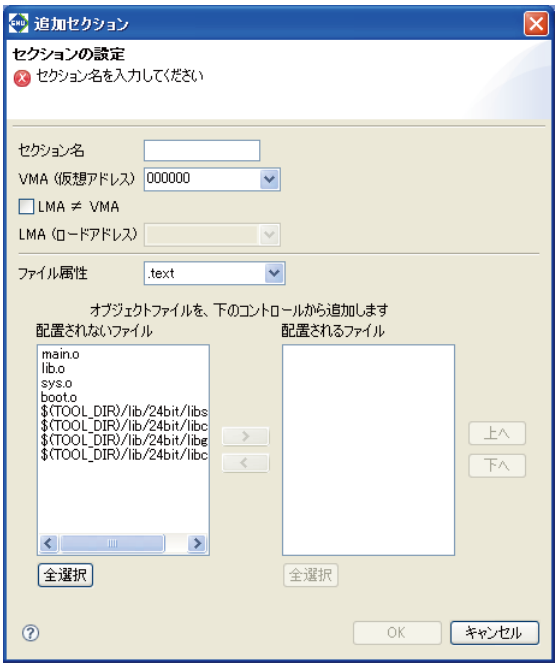
1. セクション属性([ファイル属性])の変更はできません。
2. ファイルリストは参照のみで操作できません。
3. オブジェクトファイルはアルファベットの昇順で追加されます。

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]ボタンをクリックします。

(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]ボタン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) [GNU17 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]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。

(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]ボタン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)] = 000000)
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)] = 000000)
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)000000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)] = 000000)
Section3 ([VMA(仮想アドレス)] = Section2)
Section4 ([VMA(仮想アドレス)] = Section3)

●スタックポインタについて

[GNU17 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。

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 = 0x000FC0;
```

例) ブートルーチン内で以下のように記述できます。

```
boot:
   xld.a %sp, __START_stack
```

●注意事項

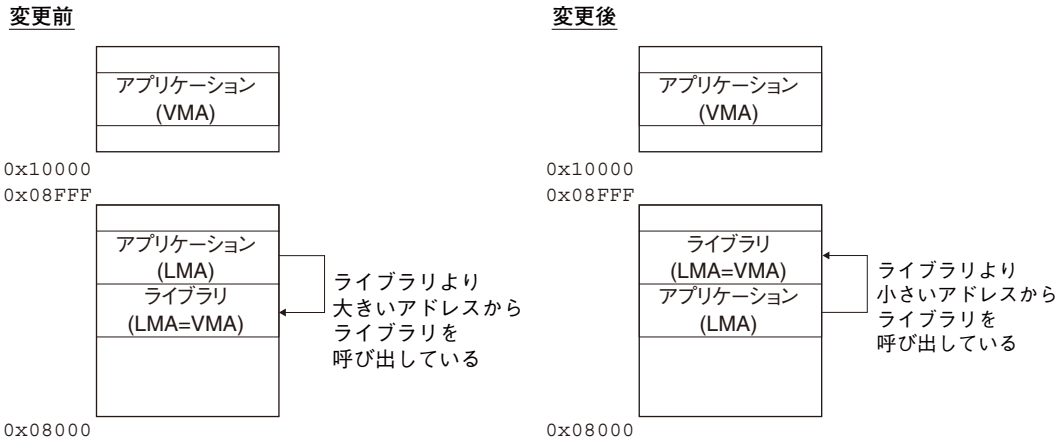
- [編集セクション]ダイアログの[セクション名]、[VMA(仮想アドレス)]、[LMA (ロードアドレス)]に入力可能な文字数は最大255文字です。
- セクション名は"."で始まる必要があります。また、使用可能な文字は、半角英数字、"_"、"."のみです。
- [VMA(仮想アドレス)]、[LMA (ロードアドレス)]にアドレスを入力する場合は、0～9とA～Fのみを使用し、6文字以内としてください。それ以外はアドレスではなくセクション名を入力したもの

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。
- リンカで以下のようなエラーが発生したとき、LMAでライブラリを呼び出すプログラムのアドレスがライブラリのアドレスより大きいアドレスに配置されていることが原因となっている場合があります。

例：`apli.c:107: undefined reference to `libfunc'`
（apli.cで呼び出しているライブラリ関数libfuncが見つからない）

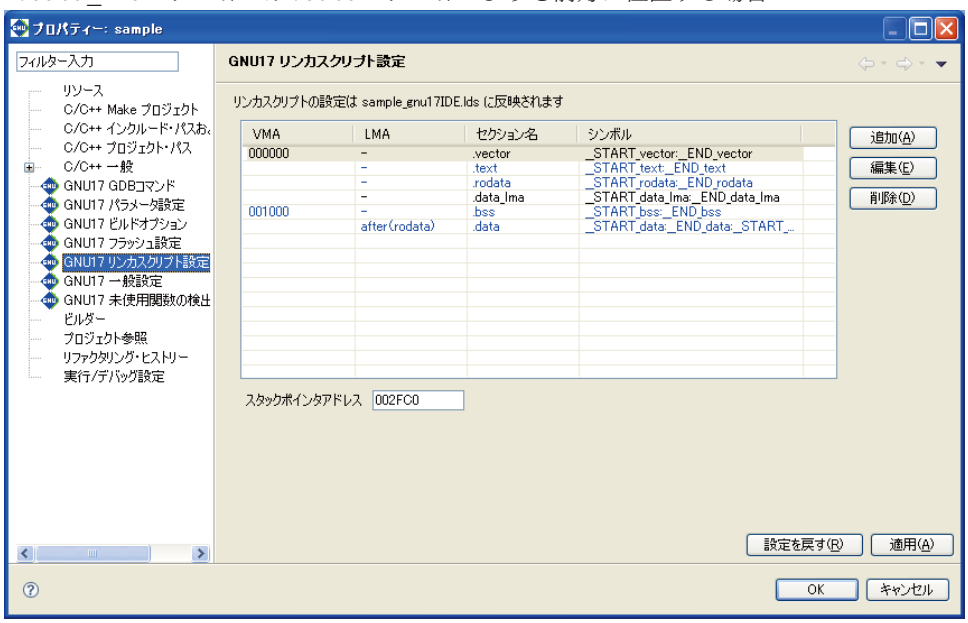
LMAでライブラリを呼び出すプログラムのアドレスがライブラリのアドレスより小さいアドレスに配置するよう変更してください。
以下にLMA配置順の変更例を示します。アプリケーションにはライブラリを呼び出すプログラムが含まれています。



- [新規 GNU17 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。
- セクション名として最後に_lma(大文字/小文字共に)を付けることはできません。

- LMAがVMAより前方に位置する場合、LMAの位置に新たにセクションを追加することができません。

例：.data_lmaセクションが.dataセクションよりも前方に位置する場合

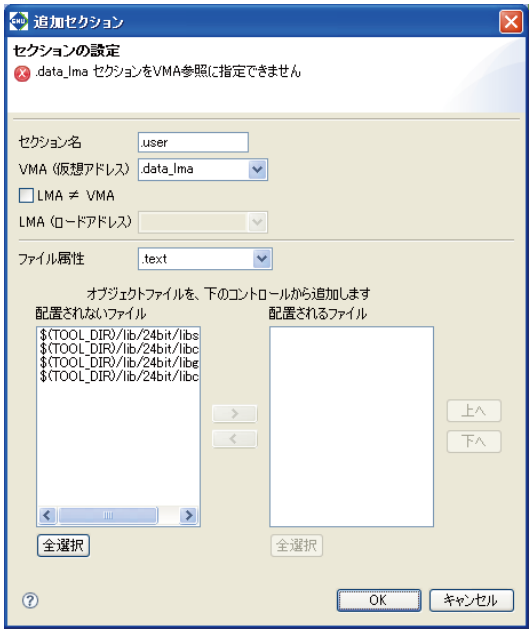


上図のような設定が行われているリンカスクリプト画面において、以下の(1) ～ (3)手順で.userセクションを.data_lmaの後方に追加しようとします。

(1) [追加]ボタンクリックし、[追加セクション]ダイアログを表示します。

(2) セクション名に.user、VMA(仮想アドレス)に.data_lmaを図のように設定します。

(3) [OK]ボタンをクリックします。



上図のようにエラーが表示され、設定することができません。

●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

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

RAMと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。
アドレス0x8000から配置されているROMに、プログラムとデータを図5.7.9.2のように配置します。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。RAMには初期値を持たない変数領域をアドレス0x0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。

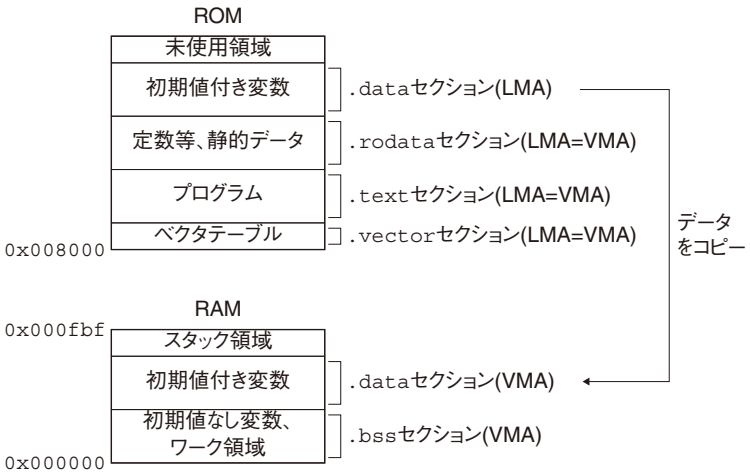


図5.7.9.2 メモリ構成例1

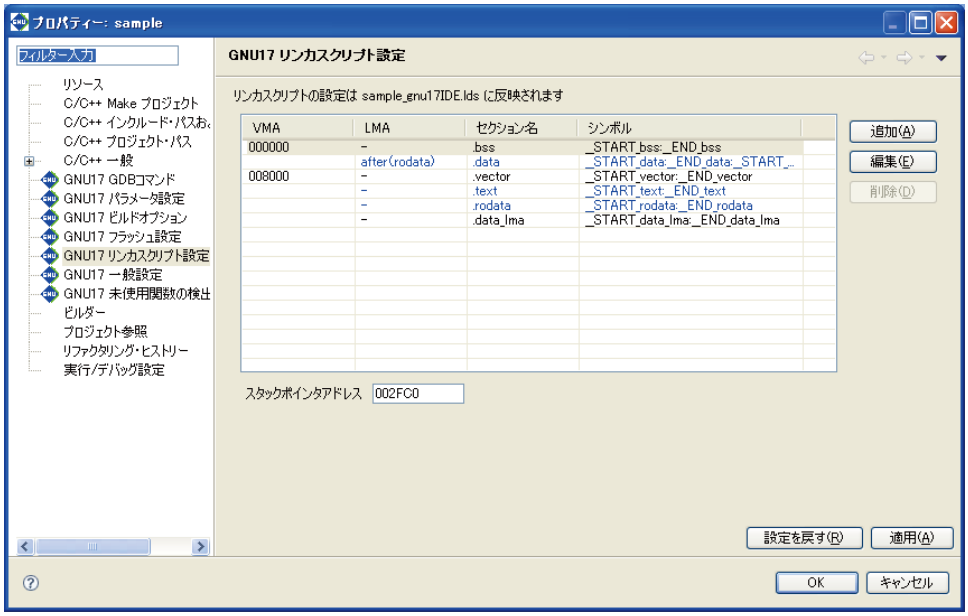
このようなメモリの使い方に対応する最も簡単な方法は、デフォルトのリンカスクリプトファイルを使用することです。
[GNU17 リンカスクリプト設定]ダイアログで設定の追加や修正を行わなくても、このセクション配置が実現できます。

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(0x8000～)に置かれます。

セクションの構成([GNU17 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0x0(RAM)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0x8000(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されるため、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.rodataセクションの後に設定しています。
図5.7.9.2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 Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{

  /* stack pointer symbols */
  __START_stack = 0x000FC0;

  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x000000 :
    {
     __START_bss = . ;
     boot.o(.bss)
     main.o(.bss)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
    }
    __END_bss = . ;

  .data __END_bss : AT( __END_rodata )
    {
```

```
        __START_data = . ;
        boot.o(.data)
        main.o(.data)
        C:/EPSON/gnu17/lib/24bit/libstdio.a(.data)
        C:/EPSON/gnu17/lib/24bit/libc.a(.data)
        C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
        C:/EPSON/gnu17/lib/24bit/libc.a(.data)
       }
       __END_data = . ;

    .vector 0x008000 :
       {
        __START_vector = . ;
        boot.o(.rodata)
       }
       __END_vector = . ;

    .text __END_vector :
       {
        __START_text = . ;
        boot.o(.text)
        main.o(.text)
        C:/EPSON/gnu17/lib/24bit/libstdio.a(.text)
        C:/EPSON/gnu17/lib/24bit/libc.a(.text)
        C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
        C:/EPSON/gnu17/lib/24bit/libc.a(.text)
       }
       __END_text = . ;

    .rodata __END_text :
       {
        __START_rodata = . ;
        boot.o(.rodata)
        main.o(.rodata)
        C:/EPSON/gnu17/lib/24bit/libstdio.a(.rodata)
        C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
        C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodata)
        C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
       }
       __END_rodata = . ;

    __START_data_lma = __END_rodata ;
    __END_data_lma = __END_rodata + ( __END_data - __START_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。

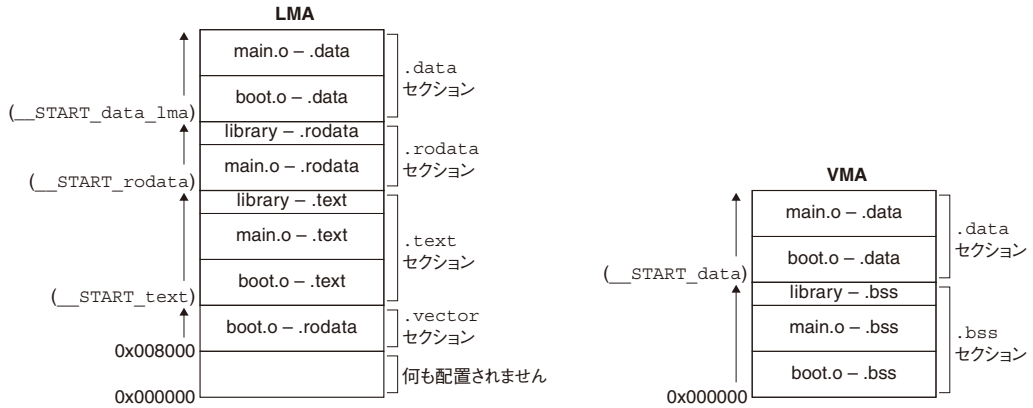


図5.7.9.3 セクション配置例1

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。

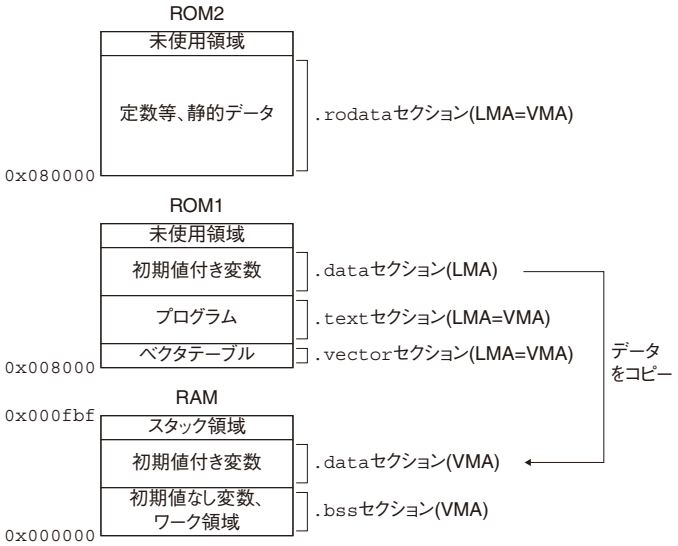


図5.7.9.4 メモリ構成例2

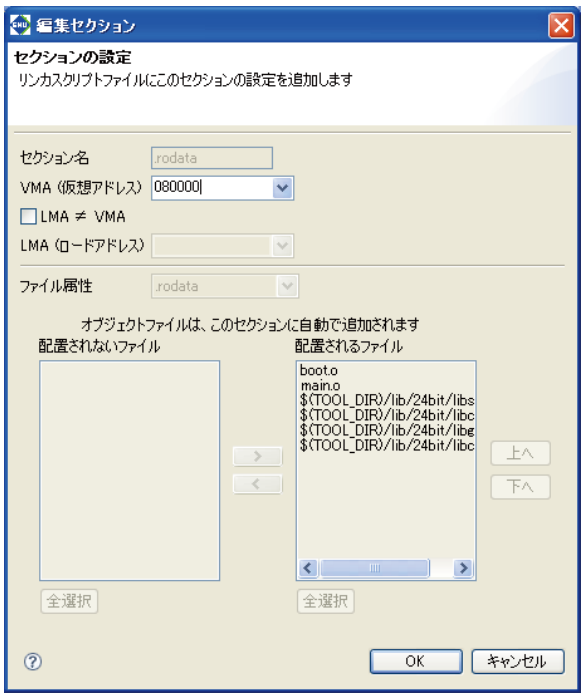
ソースファイルの構成例

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）

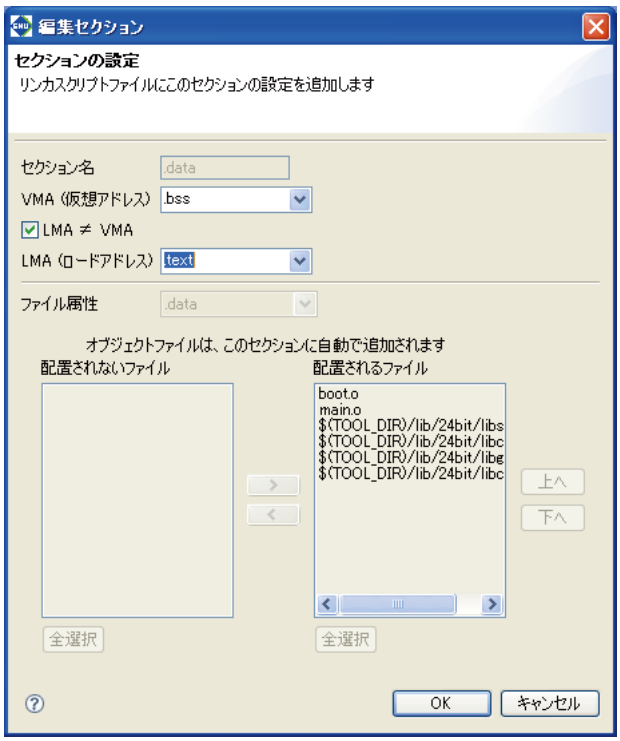
セクションの編集([編集セクション]ダイアログ上の設定内容)

以下のように.rodataセクションと.data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。

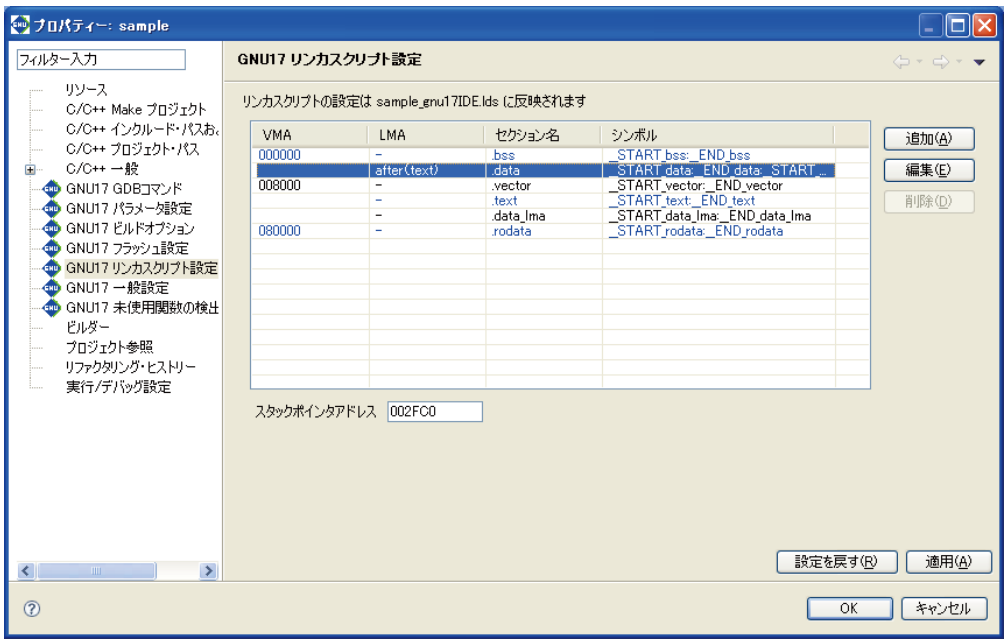
1 .rodataセクションの修正
[VMA(仮想アドレス)]を0x080000に修正します。



2 .dataセクションの修正
[LMA (ロードアドレス)]を".text"に変更します。



セクションの構成([GNU17 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 Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);
```

```
SECTIONS
{
  /* stack pointer symbols */
  __START_stack = 0x000FC0;

  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x000000 :
    {
     __START_bss = . ;
     boot.o(.bss)
     main.o(.bss)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
    }
    __END_bss = . ;

  .data __END_bss : AT( __END_text )
    {
     __START_data = . ;
     boot.o(.data)
     main.o(.data)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.data)
     C:/EPSON/gnu17/lib/24bit/libc.a(.data)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
     C:/EPSON/gnu17/lib/24bit/libc.a(.data)
    }
    __END_data = . ;

  .vector 0x008000 :
    {
     __START_vector = . ;
     boot.o(.rodata)
    }
    __END_vector = . ;

  .text __END_vector :
    {
     __START_text = . ;
     boot.o(.text)
     main.o(.text)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.text)
     C:/EPSON/gnu17/lib/24bit/libc.a(.text)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
     C:/EPSON/gnu17/lib/24bit/libc.a(.text)
    }
  __END_text = . ;

  __START_data_lma = __END_text ;
  __END_data_lma = __END_text + ( __END_data - __START_data );

  .rodata 0x080000 :
    {
     __START_rodata = . ;
     main.o(.rodata)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.rodata)
     C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodata)
     C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
    }
    __END_rodata = . ;

}
```

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。

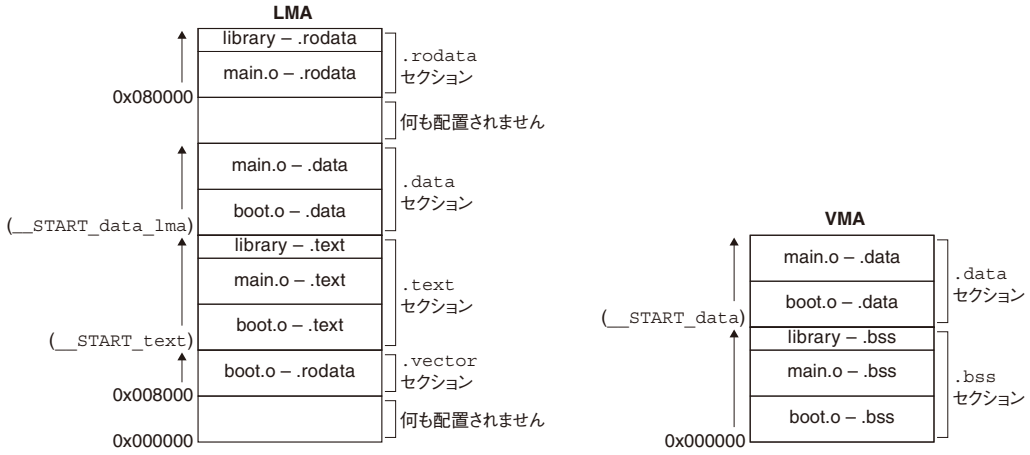


図5.7.9.5 セクション配置例2

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5.7.9.6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。

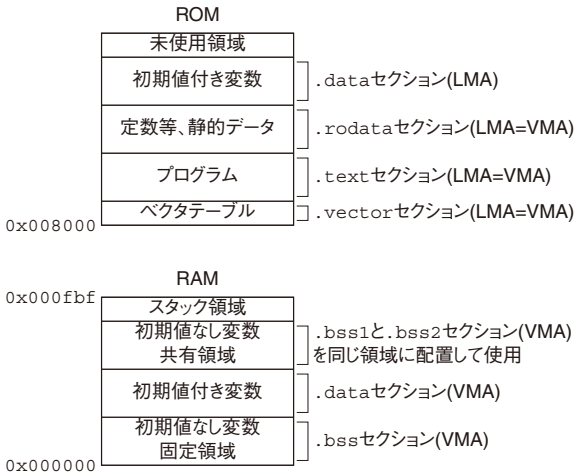


図5.7.9.6 データ領域の共有

ソースファイルの構成例

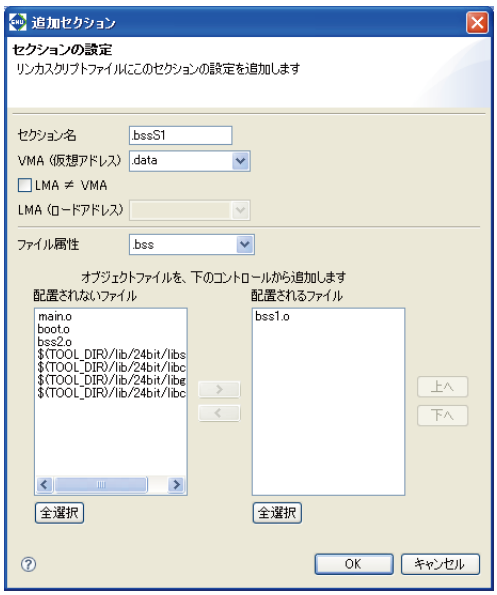
boot.s （ベクタテーブルとスタックの初期化等）
main.c （メインおよびその他の関数）
bss1.c （グローバル変数の定義ファイル1）
bss2.c （グローバル変数の定義ファイル2）

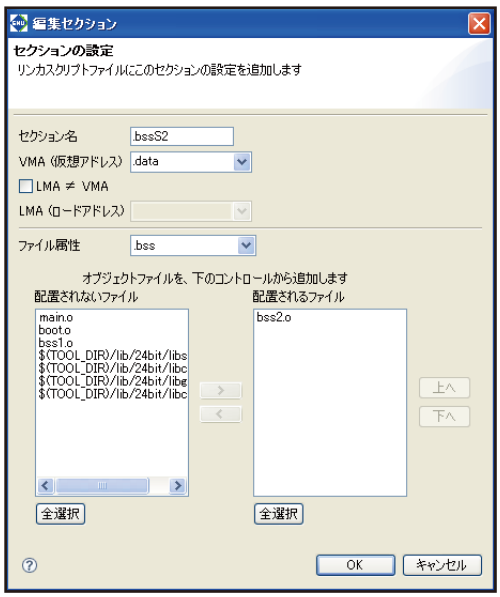
bss1.cとbss2.cには領域を共有する初期値を持たないグローバル変数のみが定義されているものとします。これらのファイルに関数や初期値を持つ変数、あるいは定数が含まれている場合、この例の設定では、.text、.data、.rodataセクションに配置されます。

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)

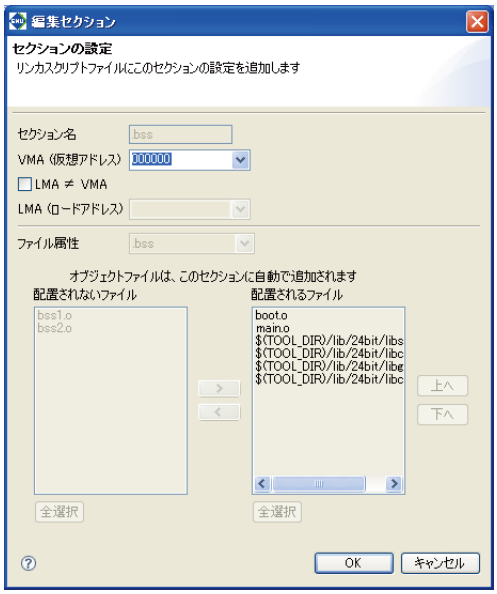
RAM領域を共有する.bssS1、.bssS2セクションを新規に作成します。どちらも.bss属性で.data

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。

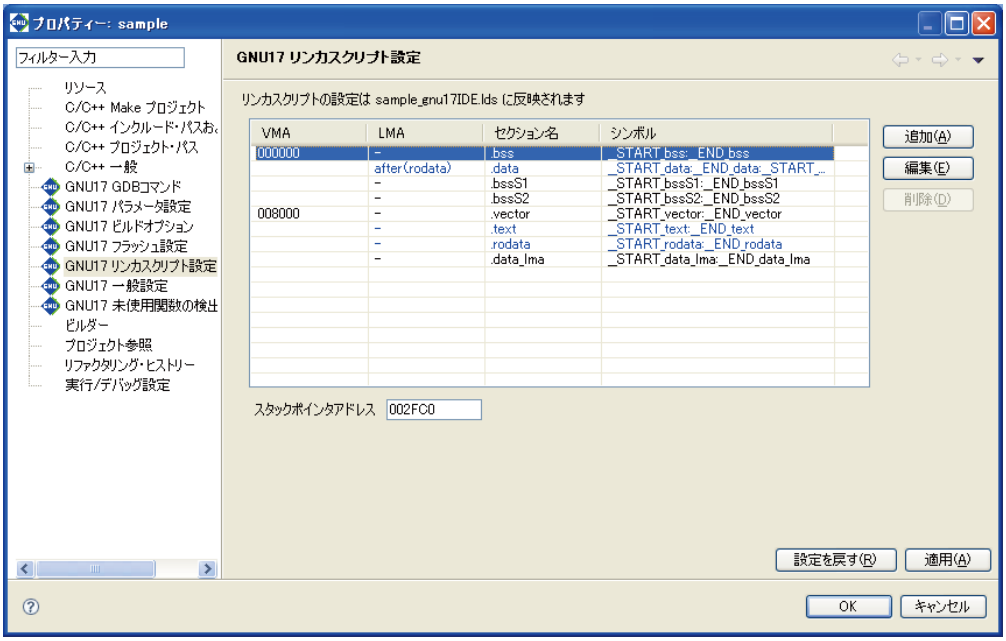




.bssセクション情報は自動更新が設定されていますが、bss1.oとbss2.oは上記のセクションで指定されたため、配置するファイル一覧には含まれません。



セクションの構成([GNU17 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 Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{
  /* stack pointer symbols */
  __START_stack = 0x000FC0;

  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x000000 :
    {
     __START_bss = . ;
     boot.o(.bss)
     main.o(.bss)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
    }
    __END_bss = . ;

  .data __END_bss : AT( __END_rodata )
    {
     __START_data = . ;
     boot.o(.data)
     bss1.o(.data)
     bss2.o(.data)
     main.o(.data)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.data)
     C:/EPSON/gnu17/lib/24bit/libc.a(.data)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
     C:/EPSON/gnu17/lib/24bit/libc.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 0x008000 :
    {
     __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/gnu17/lib/24bit/libstdio.a(.text)
     C:/EPSON/gnu17/lib/24bit/libc.a(.text)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
     C:/EPSON/gnu17/lib/24bit/libc.a(.text)
    }
    __END_text = . ;

  .rodata __END_text :
    {
     __START_rodata = . ;
     bss1.o(.rodata)
     bss2.o(.rodata)
     main.o(.rodata)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.rodata)
     C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodata)
     C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
    }
    __END_rodata = . ;

  __START_data_lma = __END_rodata ;
  __END_data_lma = __END_rodata + ( __END_data - __START_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。

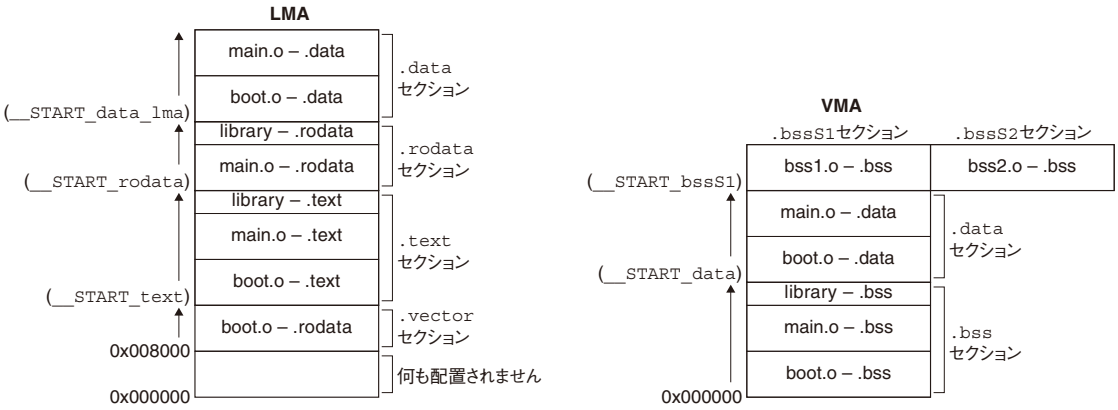


図5.7.9.7 セクション配置例3

.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。

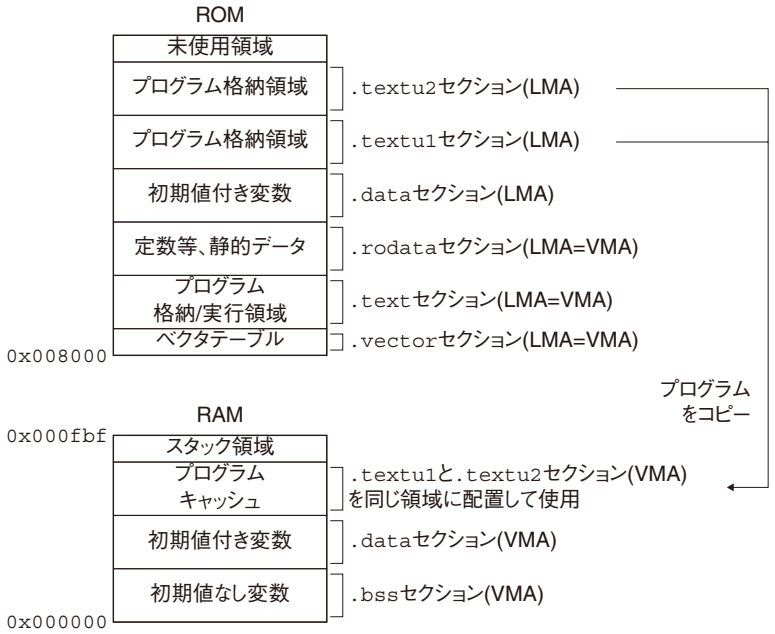


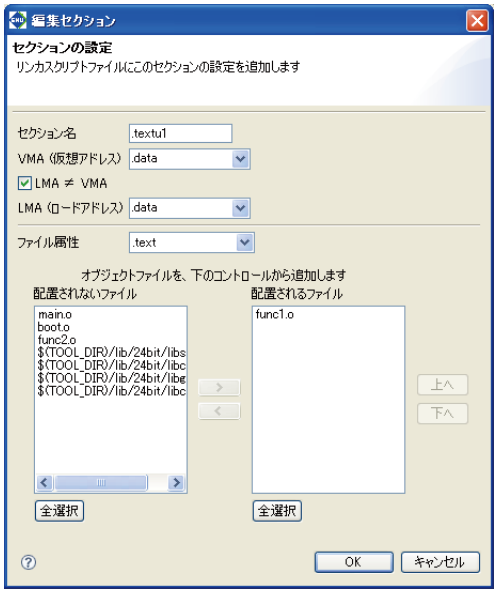
図5.7.9.8 RAM上にプログラム領域を確保

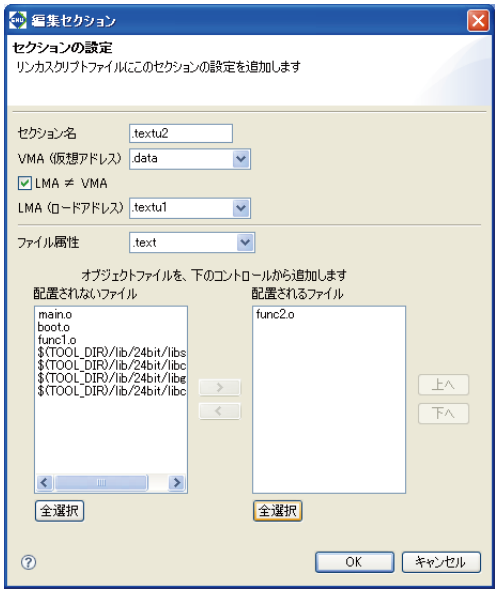
ソースファイルの構成例

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）

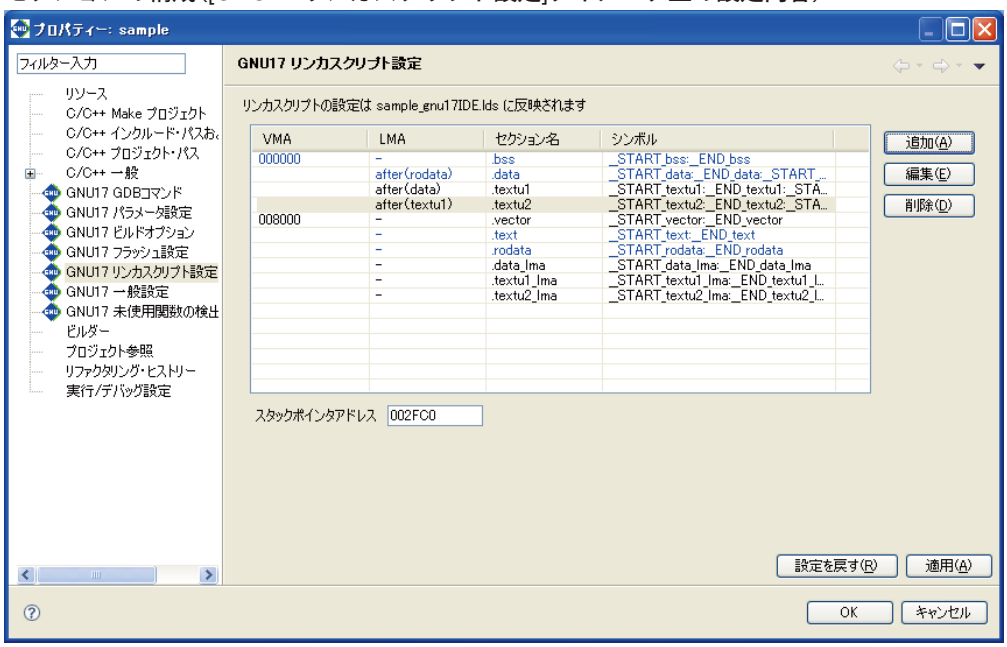
セクションの編集（[追加セクション]ダイアログ上の設定内容）

.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（[GNU17 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 Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{
  /* stack pointer symbols */
  __START_stack = 0x000FC0;

  /* location counter */
  . = 0x0;

  /* section information */
  .bss 0x000000 :
    {
     __START_bss = . ;
     boot.o(.bss)
     func1.o(.bss)
     func2.o(.bss)
     main.o(.bss)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.bss)
     C:/EPSON/gnu17/lib/24bit/libc.a(.bss)
    }
    __END_bss = . ;

  .data __END_bss : AT( __END_rodata )
    {
     __START_data = . ;
     boot.o(.data)
     func1.o(.data)
     func2.o(.data)
     main.o(.data)
     C:/EPSON/gnu17/lib/24bit/libstdio.a(.data)
     C:/EPSON/gnu17/lib/24bit/libc.a(.data)
     C:/EPSON/gnu17/lib/24bit/libgcc.a(.data)
     C:/EPSON/gnu17/lib/24bit/libc.a(.data)
    }
    __END_data = . ;

  .textu1 __END_data : AT( __START_data_lma + SIZEOF( .data ) )
    {
     __START_textu1 = . ;
     func1.o(.text)
    }
    __END_textu1 = . ;

  .textu2 __END_data : AT( __START_textu1_lma + SIZEOF( .textu1 ) )
    {
     __START_textu2 = . ;
     func2.o(.text)
    }
    __END_textu2 = . ;

  .vector 0x008000 :
    {
     __START_vector = . ;
     boot.o(.rodata)
    }
    __END_vector = . ;

  .text __END_vector :
    {
```

```
        __START_text = . ;
        boot.o(.text)
        main.o(.text)
        C:/EPSON/gnu17/lib/24bit/libstdio.a(.text)
        C:/EPSON/gnu17/lib/24bit/libc.a(.text)
        C:/EPSON/gnu17/lib/24bit/libgcc.a(.text)
        C:/EPSON/gnu17/lib/24bit/libc.a(.text)
       }
       __END_text = . ;

     .rodata __END_text :
       {
        __START_rodata = . ;
        func1.o(.rodata)
        func2.o(.rodata)
        main.o(.rodata)
        C:/EPSON/gnu17/lib/24bit/libstdio.a(.rodata)
        C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
        C:/EPSON/gnu17/lib/24bit/libgcc.a(.rodata)
        C:/EPSON/gnu17/lib/24bit/libc.a(.rodata)
       }
       __END_rodata = . ;

     __START_data_lma = __END_rodata ;
     __END_data_lma = __END_rodata + ( __END_data - __START_data );

     __START_textu1_lma = __END_data_lma ;
     __END_textu1_lma = __END_data_lma + ( __END_textu1 - __START_textu1 );

     __START_textu2_lma = __END_textu1_lma ;
     __END_textu2_lma = __END_textu1_lma + ( __END_textu2 - __START_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。

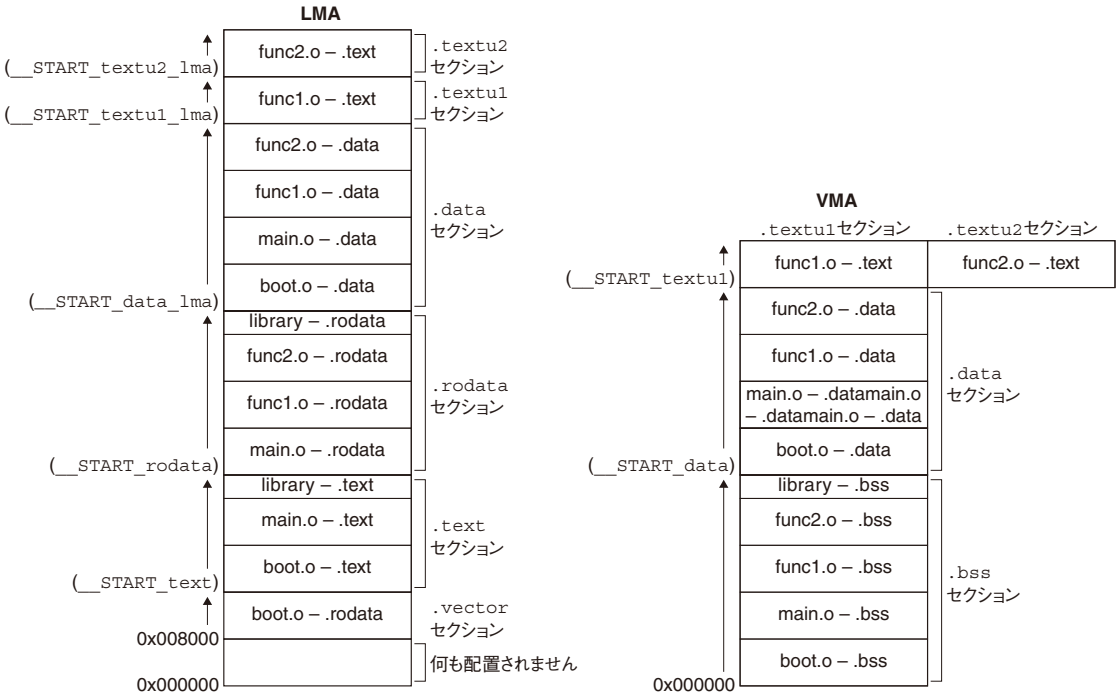


図5.7.9.9 セクション配置例4

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。

5.7.10 フラッシュメモリの設定

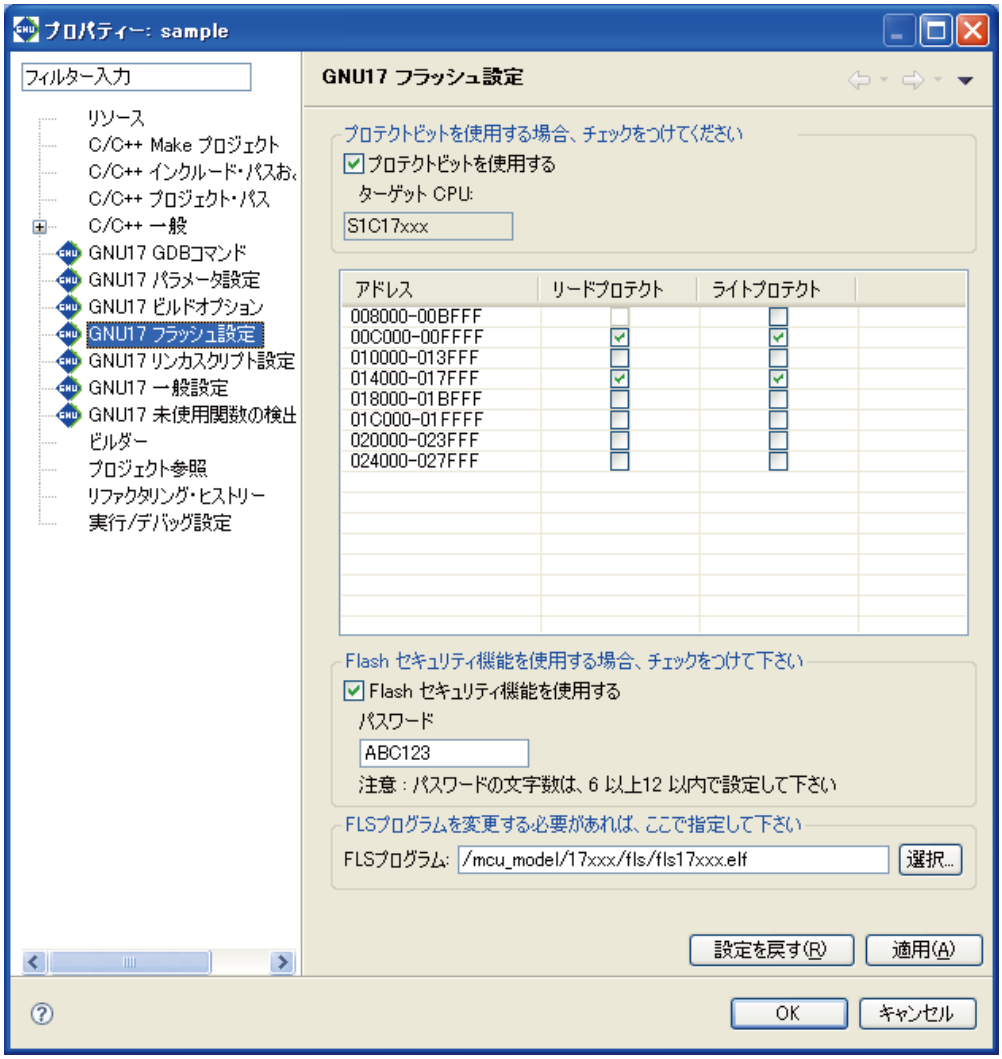
内蔵Flashメモリの内容を保護するため、ライトプロテクトとリードプロテクトの2種類を設定することができます。(機種によってはフラッシュプロテクトを設定できないものがあります。)IDEは、本節で説明する設定によりフラッシュプロテクトが設定されたpsaファイル"<プロジェクト名>_ptd.psa"を生成します。また、FLSプログラムと、(Flashセキュリティ対応機種のとき)Flashセキュリティのパスワードを設定できます。
以下、フラッシュプロテクト、パスワードおよびFLSプログラムの設定手順を説明します。

注：psaファイルを作成するには、[GNU17 ビルドオプション]ページで[ROMデータ生成(psa)]が選択されている必要があります。選択の方法は"5.7.2 ビルドゴールの設定"を参照して下さい。

● フラッシュメモリの設定ページ

フラッシュプロテクトは、プロジェクトのプロパティ [GNU17 フラッシュ設定]ページで設定します。
次の手順でフラッシュメモリ設定を行います。

(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、フラッシュ設定を行う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[プロパティー]を選択します。[プロパティー]ダイアログを表示されます。
(3) プロパティの一覧から[GNU17 フラッシュ設定]を選択します。



[ターゲットCPU]に現在選択されているS1C17コア/プロセッサの種類が表示されます。

(4)フラッシュプロテクトの設定を行うかどうかを選択します。(フラッシュプロテクト設定が可能な機種のみ)

[プロテクトビットを使用する]のチェックボックスをON/OFFします。
ON： ビルド時にフラッシュプロテクト情報が書かれた"<プロジェクト名>_ptd.psa"ファイルを作成します。ONの場合、画面下の表から詳細設定を行うことが可能になります。
OFF： フラッシュプロテクトの設定を行いません。

注：• 機種によってはフラッシュプロテクトの設定ができないものがあります。詳細は各機種のテクニカルマニュアルを参照してください。
• 生成プログラムがライブラリの時は、フラッシュプロテクトの設定はできません。

(5)フラッシュプロテクトの詳細設定を行います。

フラッシュプロテクトの詳細設定について

アドレス：

フラッシュプロテクト対象の開始アドレスと終了アドレスを表示します。

リードプロテクト：

対象アドレスのリードプロテクトをチェックボックスで設定します。
リードプロテクトは、設定したアドレスからのデータの読み出しを禁止します。
ON： プロテクトする
OFF：プロテクトしない

ライトプロテクト：

対象アドレスにライトプロテクトをチェックボックスで設定します。
ライトプロテクトは、設定したアドレスへのデータ書き込みを禁止します。
ON： プロテクトする
OFF：プロテクトしない

(6)Flash セキュリティパスワードの設定を行います。

Flash セキュリティ対応機種の場合、パスワードを設定することができます。
パスワードを設定するには、[Flash セキュリティ機能を使用する]チェックボックスを選択します。
パスワードには、以下の制限があります。
• パスワードに入力できる文字は、0 ～ 9、a ～ z、A ～ Zです。
• パスワードに設定できる文字数は、画面パスワード入力欄下の注意を確認してください。
ここで設定したパスワードは、winmdc17 によるpaファイル(提出用データ)の作成時(11.2.3 winmdc17 によるpaファイル(提出用データ)の作成を参照)、および、[デバッグ]ビューの[Flash セキュリティの解除]ボタンで使用されます。

注：• 機種によってはFlash セキュリティパスワードの設定ができないものがあります。詳細は各機種のテクニカルマニュアルを参照してください。
• 生成プログラムがライブラリの時は、Flash セキュリティパスワードの設定はできません。

(7)FLSプログラムの設定を行います。

Flash ROM書き込みを実行するFLSプログラムを選択することができます。
機種に対応したFLSプログラムがあらかじめ選択されていますので、変更する必要はありません。

(8)変更を確定する場合は[OK]ボタンまたは[適用]ボタンをクリックします。

変更内容の確定を確定します。プログラムのビルドに関する設定を変更した場合は、"clean"ビルド(5.7.13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。このとき、[プロテクトビットを使用する]がONになっていれば、protect.cmdファイルを生成します。すでにファイルがある場合は、上書きします。

[OK]ボタン： [プロパティー]ダイアログを終了します
[適用]ボタン： [プロパティー]ダイアログを終了せず、引き続き編集が行えます

変更内容を破棄して[プロパティー]ダイアログ終了する場合は、[キャンセル]ボタンをクリックします。また、[適用]ボタン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。

注：[GNU17 一般設定]ページで機種変更を行った場合、変更されたフラッシュプロテクト設定、Flashセキュリティ機能設定およびFLSプログラム設定は破棄され、変更後の機種のデフォルト設定となります。

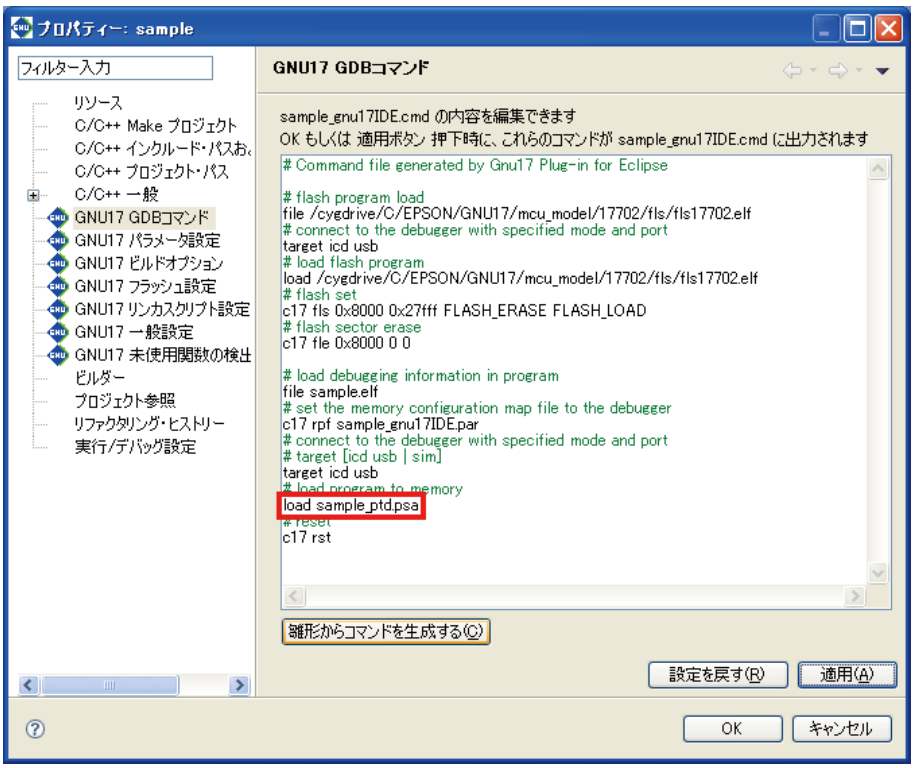
● フラッシュプロテクトされたプログラムについて

フラッシュプロテクトされたプログラムをロード・デバッグ・イレースするには、以下に示すような手順が必要です。ここではS1C17702用「sample」プロジェクトを例に説明します。

psaファイルのロード

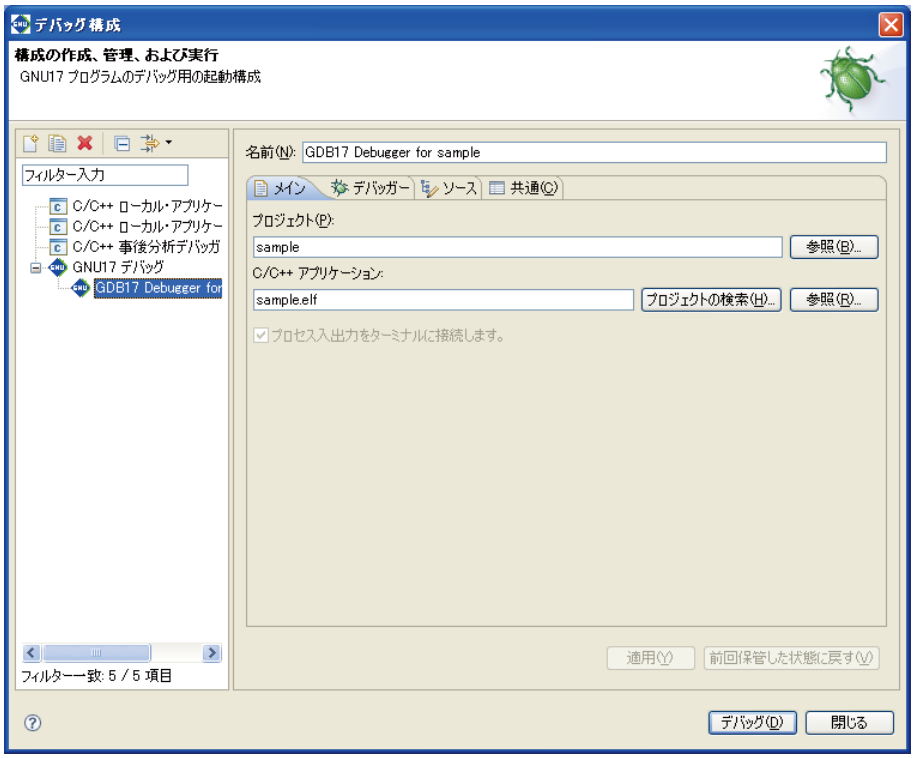
フラッシュプロテクトされたpsaファイルをロードするには、"5.8.2 デバッガ起動コマンドの設定"で説明するコマンドファイルを変更する必要があります。

(1)コマンドファイルの変更



赤枠部分のように「load sample.psa」を「load sample_ptd.psa」に変更します。

(2)psaファイルのロード
psaファイルをフラッシュへロードするために、デバッガーを起動します。
[C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで対象プロジェクトを右クリックします。コンテキストメニューから[デバッグ]>[デバッグの構成]を選択し、[デバッグ構成]ダイアログを表示します。



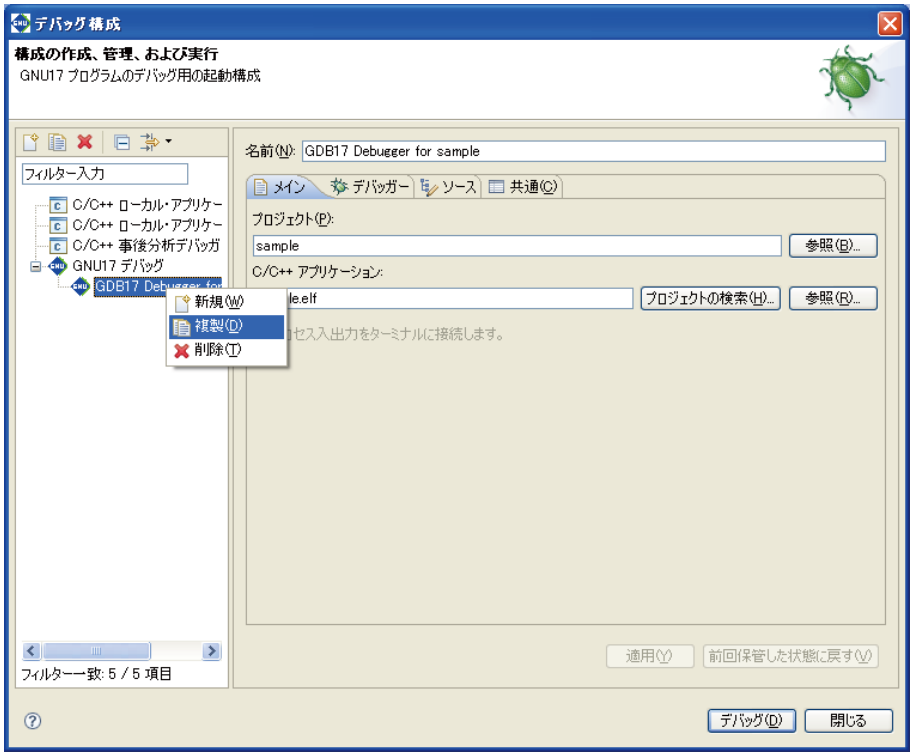
対象プロジェクトのデバッグ構成を選択します。[デバッグ]ボタンをクリックすると、デバッガーが起動します。デバッガーを終了し、ターゲットをリセットすることでプロテクトが反映されます。

psaファイルのデバッグ

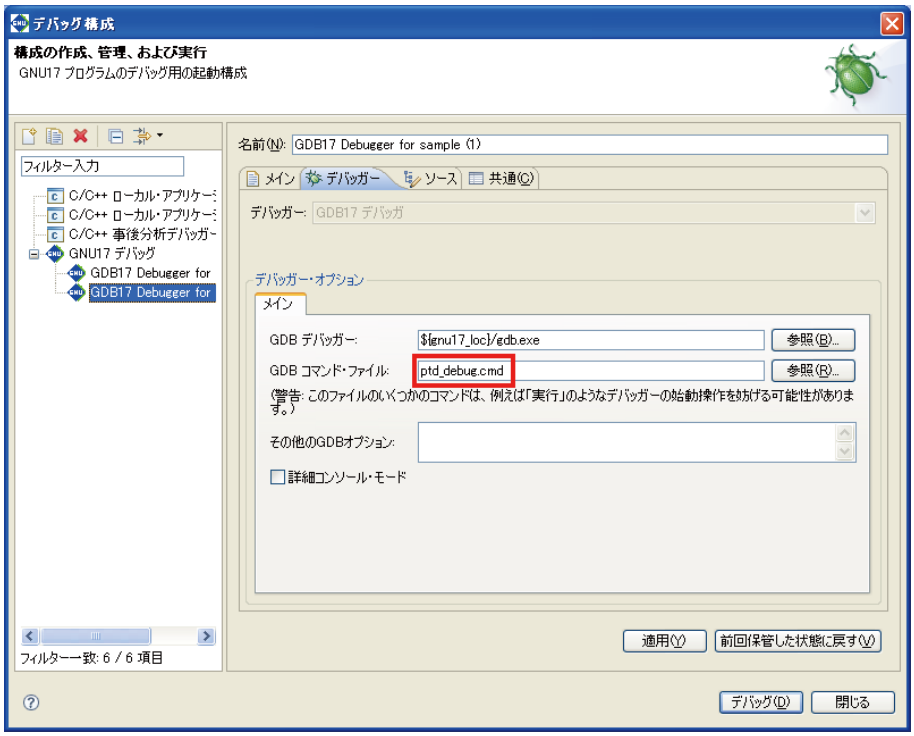
プロテクトされたpsaファイルをデバッグするには以下のような手順が必要です。
(1)コマンドファイルの作成
(作成例)
[雛形からコマンドを生成する]で作成されるコマンドファイル(デバッガ：ICD Mini)を参考に以下の内容を記述したptd_debug.cmdファイルをエディター等で作成します。

```
# load debugging information in program
file sample.elf
# set the memory configuration map file to the debugger
c17 rpf sample_gnu17IDE.par
# connect to the debugger with specified mode and port
# target [icd usb | sim]
target icd usb
# reset
c17 rst
```

(2)デバッグ構成の作成
次にデバッグ構成の作成を行います。
[デバッグ構成]ダイアログを表示します。



対象プロジェクトのデバッグ構成を右クリックし、複製を選択します。
複製されたデバッグ構成について、変更を行います。画面右のタブの中から、デバッガータブを選択します。



GDB コマンド・ファイルを入力あるいは[参照]ボタンをクリックし、赤枠部分のように「sample_gnu17IDE.cmd」を「ptd_debug.cmd」に変更します。

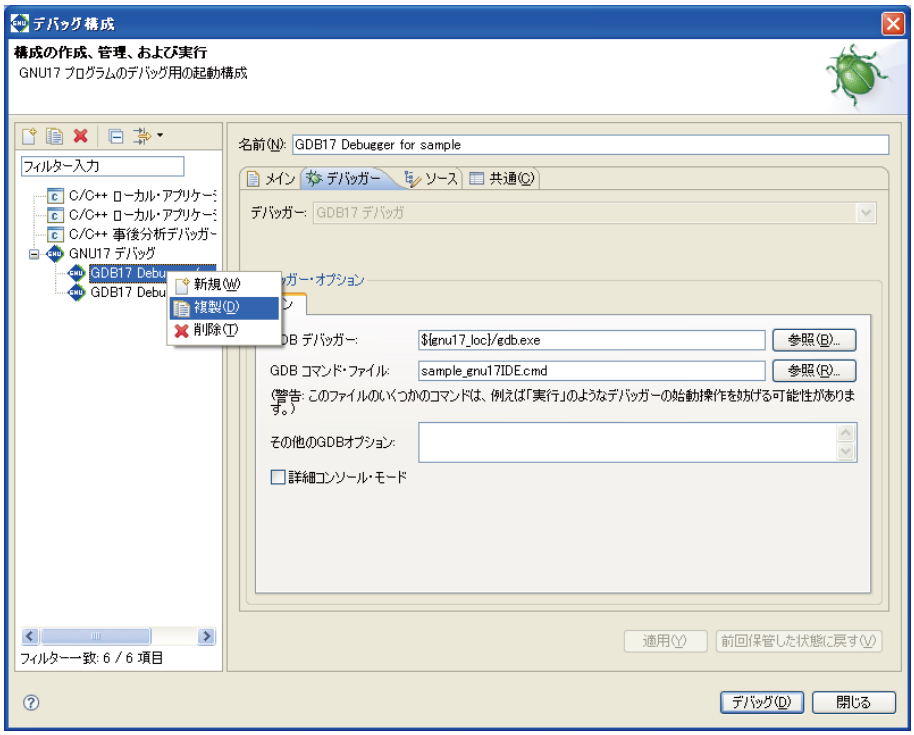
(3)デバッグの実行
複製されたデバッグ構成を選択しデバッグを行います。

プロテクトの解除

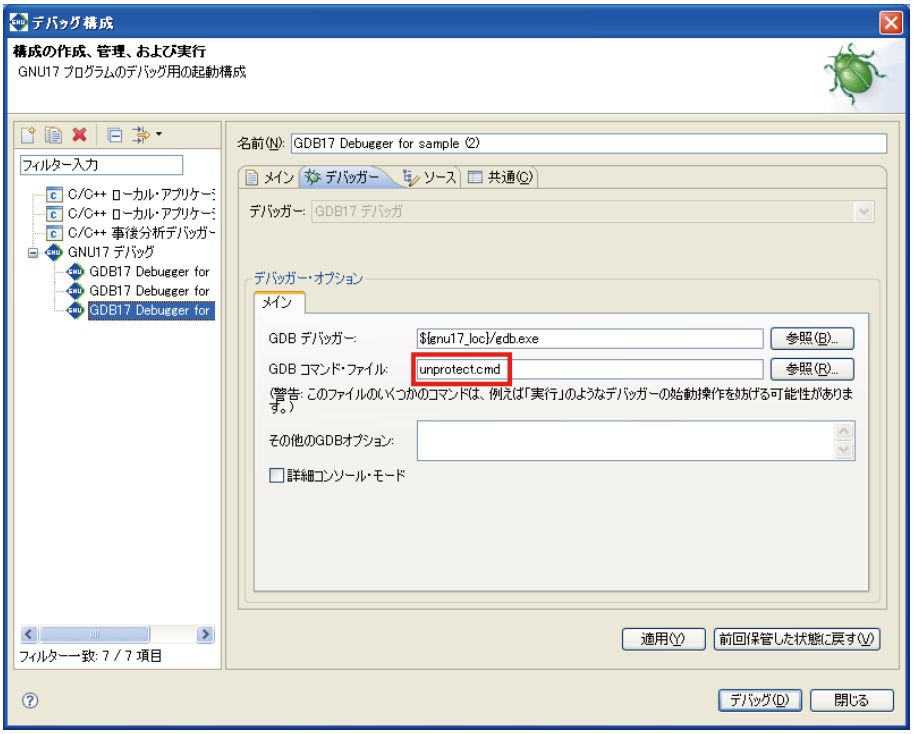
(1)コマンドファイルの作成
(作成例)
[雛形からコマンドを生成する]で作成されるコマンドファイル(デバッガ：ICD Mini)を参考に以下の内容を記述したunprotect.cmdファイルをエディター等で作成します

```
# flash program load
file /cygdrive/C/EPSON/GNU17/mcu_model/17702/fls/fls17702.elf
# connect to the debugger with specified mode and port
target icd usb
# load flash program
load /cygdrive/C/EPSON/GNU17/mcu_model/17702/fls/fls17702.elf
# flash set
c17 fls 0x8000 0x27fff FLASH_ERASE FLASH_LOAD
# flash erase
c17 fle 0x8000 0 0

c17 rstt
```

(2)デバッグ構成の作成
次にデバッグ構成の作成を行います。
デバッグ構成ダイアログを表示します。



対象プロジェクトのデバッグ構成を右クリックし、複製を選択します。
複製されたデバッグ構成について、変更を行います。画面右のタブの中から、デバッガータブを選択します。



GDB コマンド・ファイルを入力あるいは[参照]ボタンをクリックし、赤枠部分のように「sample_gnu17IDE.cmd」を「unprotect.cmd」に変更します。

(3)プロテクトの解除

複製されたデバッグ構成を選択しデバッグします。

デバッガーを終了し、ターゲットをリセットすることでプロテクトの解除が反映されます。

5.7.11 未使用関数の検出

選択したプロジェクト内で一度も使用されていないC言語の関数を検出します。未使用関数の検出には静的コード解析ツールであるSplintを使用しております。未使用関数の検出は、C言語のファイルであるCソース(*.c)やヘッダファイル(*.h)が対象となります(アセンブラのファイルは対象外となります)。

注：Splintは米国のVirginia大学で開発されたオープンソース(GPLライセンス)の静的ソースコード解析ツールです。ソースコードを解析し、関数の呼び出しや脆弱性、コーディングミスなどの解析が可能です。ただし、GNU17では未使用関数の検出機能のみを使用しています。

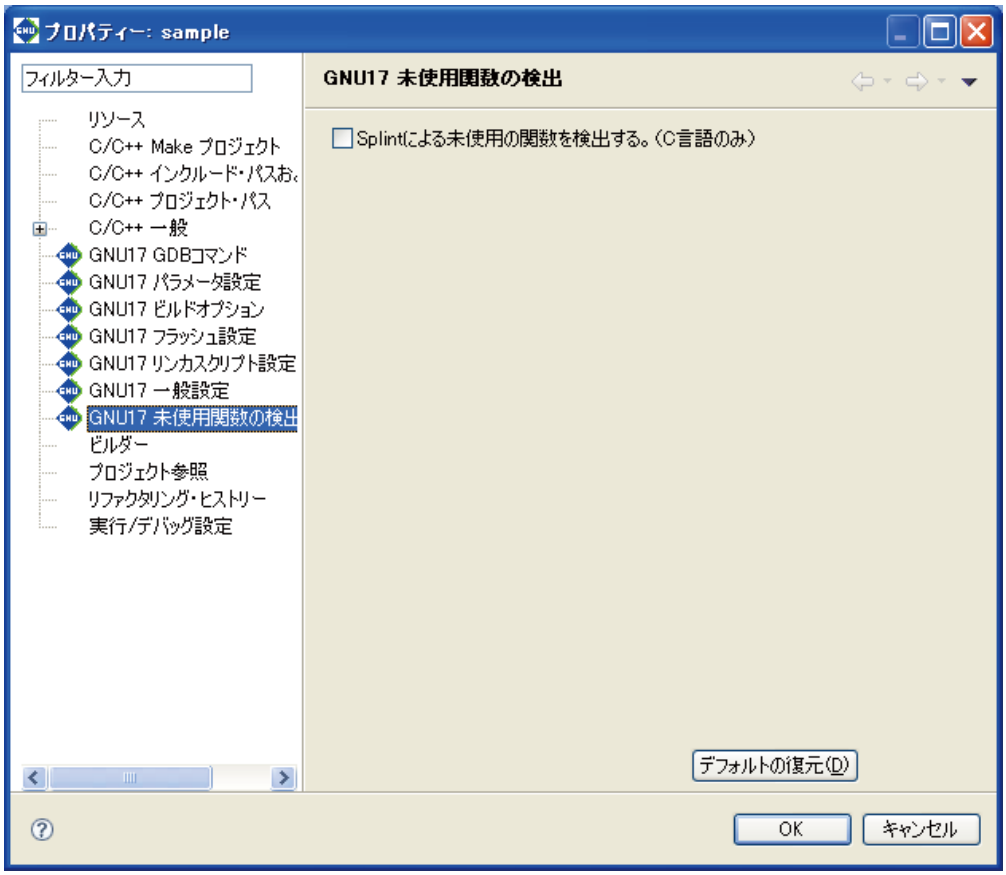
●未使用関数の検出の設定ページ

未使用関数の検出は、プロジェクトプロパティの[GNU17 未使用関数の検出]ページで設定します。次の手順で未使用関数の検出を行います。

(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、未使用関数の検出を行う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[プロパティー]を選択します。[プロパティー]ダイアログが表示されます。

(3) プロパティーの一覧から[GNU17 未使用関数の検出]を選択します。



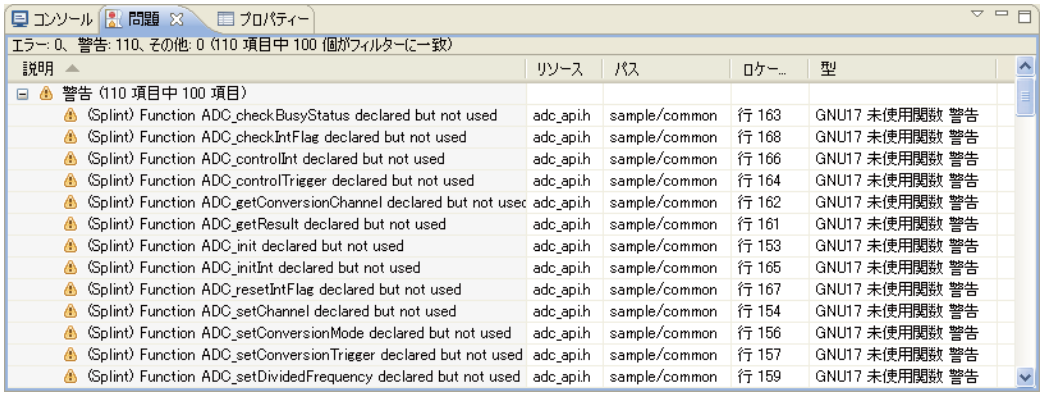
このページでは未使用関数の検出に関するオプションを選択します。

[Splintによる未使用の関数を検出する。(C言語のみ)](デフォルト: OFF)

このオプションを指定するとビルド実行時に未使用関数の検出を行います。

●未使用関数の検出

未使用関数の検出は、ビルド実行時に行います。未使用の関数が検出されると、[問題]ビューに未使用の関数が存在するファイルとその行番号が表示されます。説明の先頭が(Splint)から始まる警告が、本機能により出力された情報となります。



●エラーメッセージ

未使用関数の検出中にエラーが発生した場合、[問題]ビューにエラーが表示されます。表示されるエラーメッセージは次のとおりです。

表5.7.11.1 エラーメッセージ

エラーメッセージ	内容
(Splint)Splint.exeが開けませんでした。	Splintの実行ファイルを開くことができませんでした。
(Splint)解析対象のパスが最大値(255文字)を超えました。	解析対象のCソース(*.c)やヘッダファイル(*.h)のパスが最大値(255文字)を超えました。
(Splint)Splintの実行コマンドが最大値(32767文字)を超えました。	未使用の関数を検出するための実行コマンドが最大値(32767文字)を超えました。
(Splint)警告が10000件を超えました。警告を10000件まで表示します。	未使用の関数の警告が10000件を超えたため、10000件まで警告を表示します。
(Splint)Splintの実行に失敗しました。	Splintの実行に失敗しました。

●制限事項

・ 未使用関数の検出に使用するSplintは、ISO/IEC 9899:1999 - Programming Language C C99(以下、C99と記載)に完全には対応しておりません。そのため、プロジェクトのビルドの実行には成功しますが、未使用関数の検出に失敗して未使用関数の検出を中止する可能性があります(Cコンパイラはc99に対応しております)。次のような場合に未使用関数の検出に失敗します。

例：
```
main()
    {
        int a = 5;
        printf("test");
        int b;          // スコープの途中で変数を宣言
        b = a;
    }
```

上記のようにスコープの途中で変数を宣言したとき、未使用関数の検出に使用しているSplintの実行に失敗します。失敗した際、[問題]ビューに失敗の原因となったファイルとその行番号、説明の先頭が(Splint)から始まる内容が表示されます。

これは未使用関数の検出の失敗であり、ビルドの失敗ではないことに注意してください(ビルドの最終生成物に影響はありません)。

5.7.12 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>*_gnu17IDE.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>*_gnu17IDE.lds)*1
   - パラメータファイル(*<プロジェクト名>*_gnu17IDE.par)*1*2
   - コマンドファイル(*<プロジェクト名>*_gnu17IDE.cmd)*1*2

   *1 生成プログラムがライブラリの時は、これらのファイルは生成されません。
   2 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>*_gnu17IDE.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)
   - Cソースごとのアセンブリファイル(*<Cソース名>*.ext0)
   - 実行形式オブジェクトファイル(*<プロジェクト名>*.elf)
   - Sレコード形式のpsaファイル(*<プロジェクト名>*.psa)
   - 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します。ビルド時に再生成されないようにするには、[プロパティー]ダイアログの[ビルダー]ページで選択されている[GNU17 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

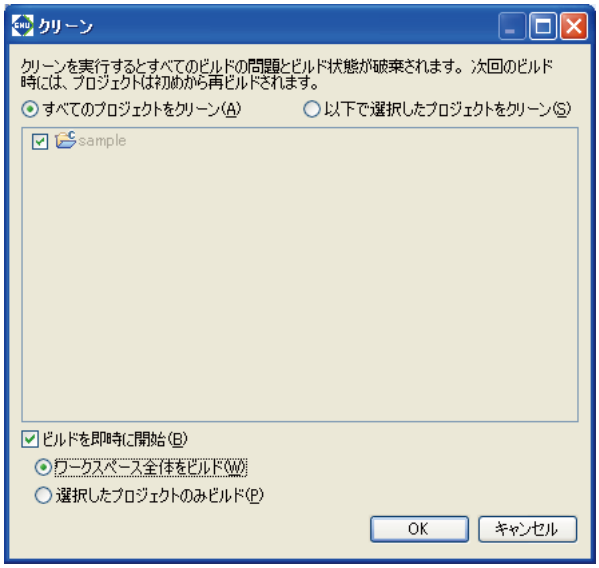
プロジェクト外部にリンクされたソースファイルのオブジェクトファイルは、
プロジェクトフォルダ直下に生成されます。

5.7.13 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にない場合
  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。

5.7.14 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。

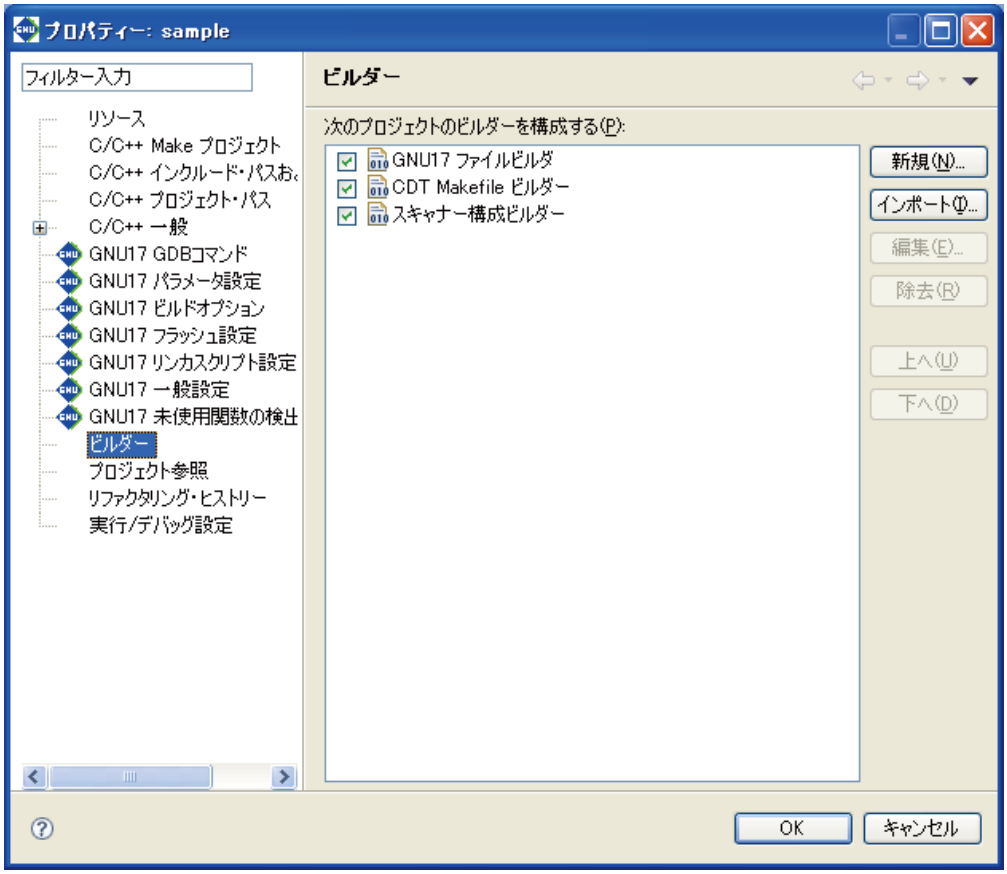
(1) [C/C++ プロジェクト]ビューまたは[ナビゲーター]ビューで、ユーザが作成したmakeファイルでビルドするプロジェクトを選択します。

(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[プロパティー]を選択します。
[プロパティー]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>*_gnu17IDE.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。

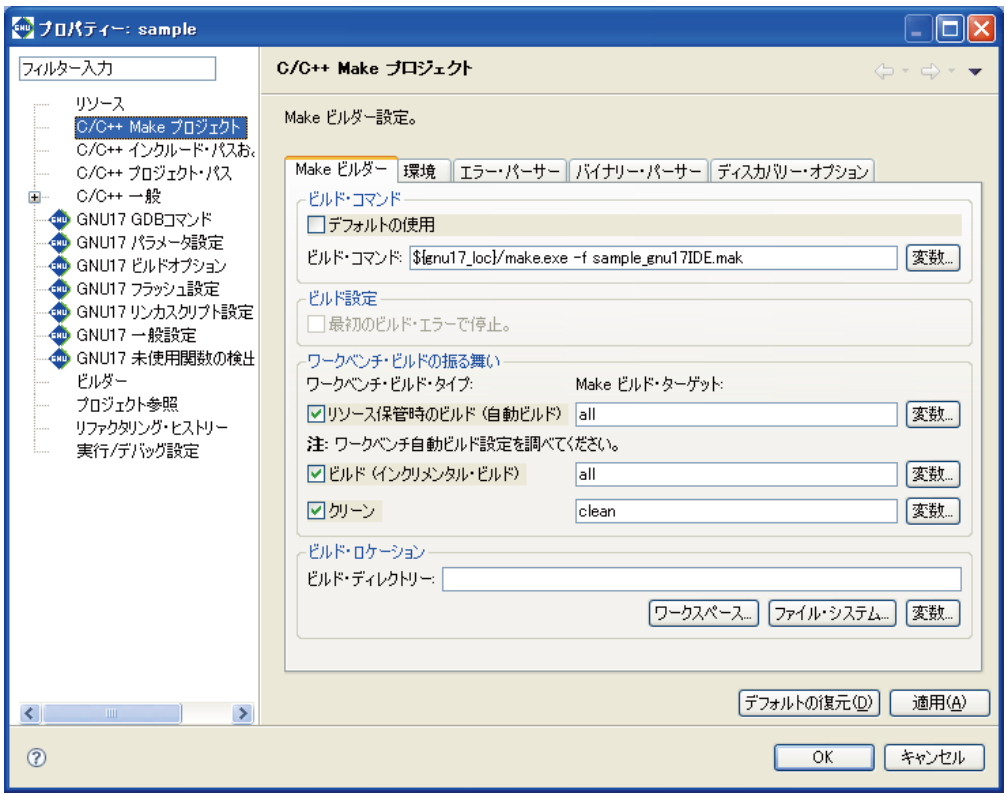
(4) プロパティの一覧から[ビルダー]を選択します。



(5) [GNU17 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>*_gnu17IDE.mak)
- リンカスクリプトファイル(*<プロジェクト名>*_gnu17IDE.lds)
- パラメータファイル(*<プロジェクト名>*_gnu17IDE.par)
- コマンドファイル(*<プロジェクト名>*_gnu17IDE.cmd)

[GNU17 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[GNU17 ファイルビルダ]を選択しておきます。または、[GNU17 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[GNU17 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

```
${gnu17_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[プロパティー]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ブレー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ーク(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)

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、"10.9 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>*_gnu17IDE.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。

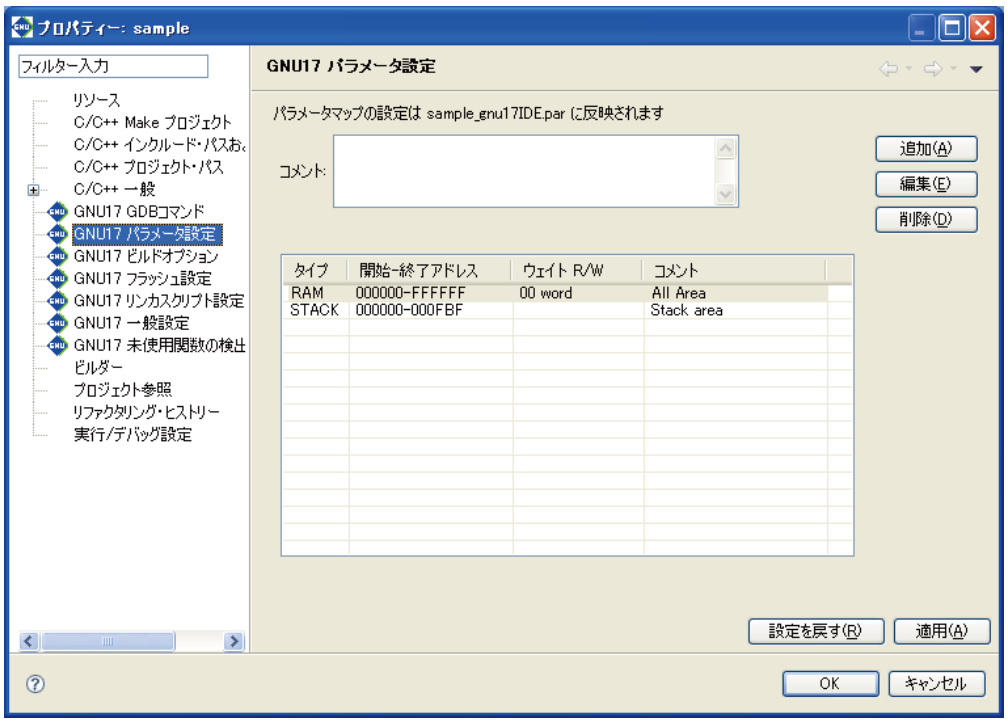
●パラメータ設定ページ

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[GNU17 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[プロパティー]を選択します。[プロパティー]ダイアログが表示されます。

(3) プロパティの一覧から[GNU17 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。

注：生成プログラムがライブラリの時は、パラメータの設定はできません。

[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。

[開始ー終了アドレス]
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"(8ビット)、"halfword"(16ビット)、"word"(32ビット)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。

ターゲットCPUデバイスにS1C17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	エリアコメント
RAM	0x000000-0xFFFFFF	0/0	word	All Area
STACK	0x000000-0x000FBF	–	–	Stack area

S1C17以外のターゲットCPUデバイスを選択した場合は、機種ごとに用意される設定ファイルによりパラメータの初期設定内容が変わります。S1C17702以外のターゲットCPUデバイスの内容については、C:¥EPSON¥gnu17¥mcu_modelの対応する機種フォルダ内の「parameter.txt」を参照してください。

ターゲットCPUデバイスにS1C17702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	エリアコメント
RAM	0x000000-0x002FBF	0/0	Word	Internal RAM
IO	0x004000-0x0043FF	0/0	Byte	Peripheral Area1
IO	0x005000-0x005FFF	0/0	Byte	Peripheral Area2
ROM	0x008000-0x027FFF	1/0	Halfword	ROM(Flash)
RAM	0x080000-0x08055F	1/1	Byte	SRAM(LCD Display)
IO	0xFFFC00-0xFFFFFF	0/0	Byte	Reserved for Core I/O
STACK	0x000000-0x002FBF	–	–	Stack area

パラメータファイルの内容

ターゲットCPUデバイスにS1C17選択時

```
# Parameter file generated by Gnu17 Plug-in for Eclipse

RAM     000000 FFFFFF 00W     # All Area
STACK   000000 000FBF         # Stack area
```

ターゲットCPUデバイスにS1C17702選択時

```
# Parameter file generated by Gnu17 Plug-in for Eclipse
CHIP S1C17702
ESSIM S1C17702

RAM     000000 001FBF 00W     # Internal RAM
IO      004000 0043FF 00B     # Peripheral Area1
IO      005000 005FFF 00B     # Peripheral Area2
ROM     008000 027FFF 10H     # ROM (Flash)
RAM     080000 08055F 11B     # SRAM (LCD Display)
```

```
IO      FFFC00 FFFFFF 00B     # Reserved for Core I/O
STACK   000000 001FBF         # 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) [GNU17 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]ボタンをクリックします。

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]ボタンで終了し、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]ボタン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ブレー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ガで使用されます。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ー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:]、[ビックエンディアン]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

[ビックエンディアン]
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。

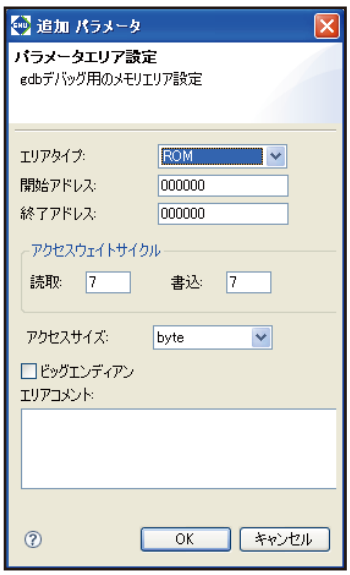
[エリアコメント:]
エリアの内容などをコメントとして入力します。コメントの先頭を示す"#"を入力する必要はありません。

[OK]
ダイアログを閉じ、[GNU17 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]ボタン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]ボタン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。[GNU17 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]ボタンをクリックします。

(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]ボタン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) [GNU17 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]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。

(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]ボタン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。

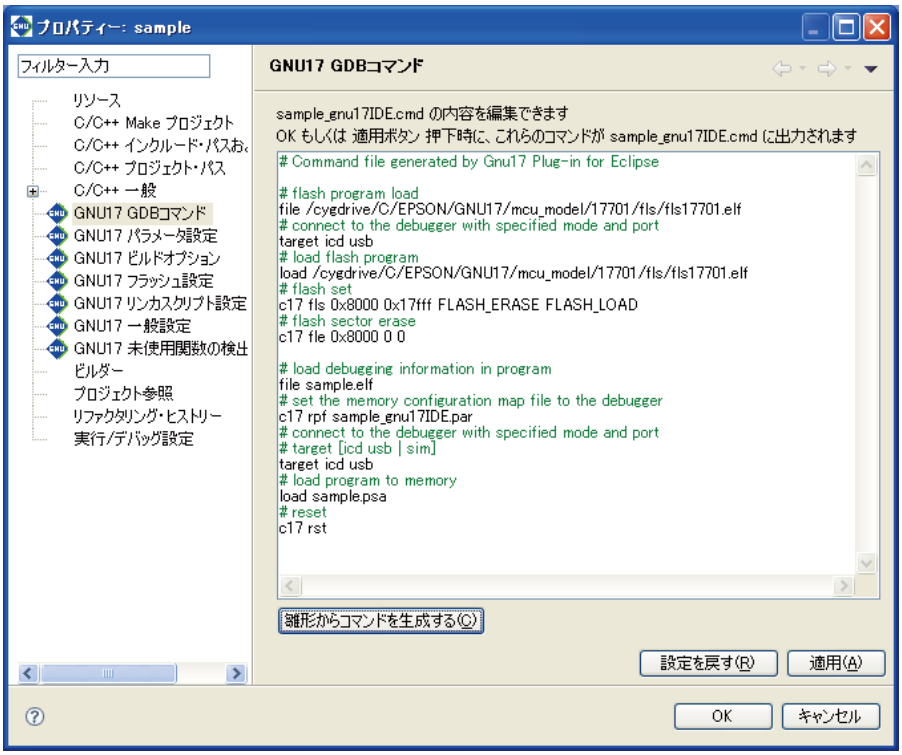
5.8.2 デバッガ起動コマンドの設定

デバッガの起動コマンドは、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[プロパティー]を選択します。[プロパティー]ダイアログが表示されます。

(3) プロパティの一覧から[GNU17 GDBコマンド]を選択します。



このページでデバッガ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>*_gnu17IDE.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。

注：生成プログラムがライブラリの時は、デバッガ起動コマンドファイルの編集はできません。

(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]ボタン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>*_gnu17IDE.cmd)が生成され、デバッガ起動時にデバッガに渡されます。

ICD Miniモード用コマンドファイル(デフォルト)

```
# Command file generated by Gnu17 Plug-in for Eclipse

# flash program load
file /cygdrive/C/EPSON/GNU17/mcu_model/17701/fls/fls17701.elf
# connect to the debugger with specified mode and port
target icd usb
# load flash program
load /cygdrive/C/EPSON/GNU17/mcu_model/17701/fls/fls17701.elf
# flash set
c17 fls 0x8000 0x17fff FLASH_ERASE FLASH_LOAD
# flash erase
c17 fle 0x8000 0 0
```

```
# 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
c17 rpf sample_gnu17IDE.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 | sim]
target icd usb                           ... コネクトモードの設定
# load program to memory
load sample.psa                          ... S レコードファイルの読み込み
# reset
c17 rst                                  ... リセット
```

デフォルト設定では、デバッガをICD Miniモードに設定するコマンドが表示されます。シミュレータ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。

シミュレータモード用コマンドファイル

```
# Command file generated by Gnu17 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
c17 rpf sample_gnu17IDE.par              ... パラメータファイルの読み込み
# set ttbr for simulator
c17 ttbr 0x008000                        ... TTBR の設定
# connect to the debugger with specified mode and port
# target [icd usb | sim]
target sim                               ... コネクトモードの設定
# load program to memory
load sample.psa                          ... S レコードファイルの読み込み
# reset
c17 rst                                  ... リセット
```

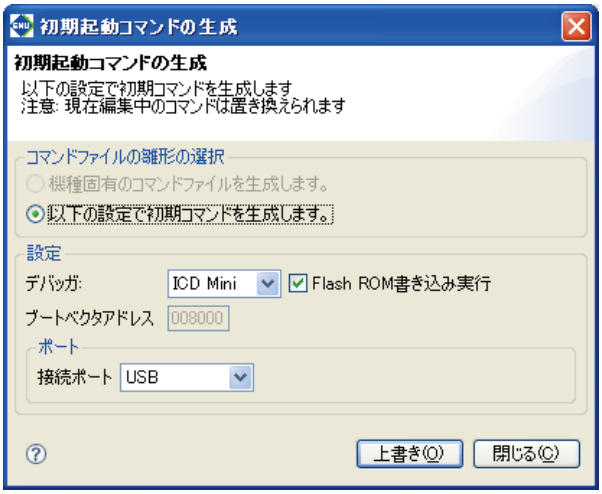
このコマンドファイルはデバッガの起動時に実行され、以下の初期設定を行います。
1. デバッグ情報とパラメータマップ情報の取り込み
2. コネクトモードの設定
3. TTBRの設定(シミュレータモード時のみ)
4. デバッグするプログラムの読み込み
5. PC(プログラムカウンタ)をブートアドレスに設定

これにより、デバッガ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。
デバッガコマンドについては"10 デバッガ"を参照してください。

デバッガ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[プロパティー]ダイアログを開いても、その内容が[GNU17 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、[プロパティー]ダイアログを開いてください。

[初期起動コマンドの生成]



[コマンドファイルの雛形の選択]

コマンドファイルの雛形を選択します。

[機種固有のコマンドファイルを生成します。]

ターゲットCPUの機種情報にあるコマンドファイルの内容を雛形として使用します。
プロジェクトの機種固有のコマンドファイルが存在する場合のみ選択可能となります。
この方法でコマンドファイルを生成する場合、プロジェクトの環境に合わせてコマンドファイルを修正する必要があります。

[以下の設定で初期コマンドを生成します。]

下記の[設定]で指定した内容でコマンドファイルを生成し、それを雛形として使用します。

[設定]

コマンドファイル作成のための設定を行います。

[デバッガ:]

接続するデバッガ(コネクトモード)を選択します。
ICD Mini　ICD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

[Flash ROM書き込み実行]

[デバッガ]にICD Miniを選択した場合に、作成するコマンドファイル中で、ターゲットのFlash ROMへビルドしたプログラムを書き込むか否かを選択します。
チェックした場合、[プロパティー]>[GNU17一般設定]ページで選択されているターゲットCPUと、[プロパティー]>[フラッシュ設定]ページで選択されているFLSプログラムに応じてコマンドが追加されます。

[ブートベクタアドレス]　（シミュレータモード時のみ有効）

シミュレータモードのコマンドファイルを作成する場合に、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。アドレスは256バイト単位で指定する必要があります。**IDE**が生成するシミュレータモード用の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]

[デバッガ:]にICD Miniを選択した場合はUSBが、Simulatorを選択した場合はNONEに固定されますので設定の必要はありません。

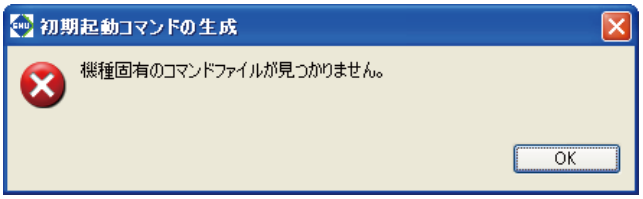
[上書き]

上記の設定内容を[GNU17 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[GNU17 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。

[プロパティー]>[GNU17 一般設定]ページでメモリモデルを変更すると、このダイアログもデフォルト設定(ICD Mini)に戻ります。また、[プロパティー]>[GNU17 一般設定]ページにおいて、CPUを変更した場合、そのCPUのブートベクタアドレスが、[ブートベクタアドレス]に設定されます。

以下のエラーダイアログは、機種情報ファイルでコマンドファイル名が指定されているにも関わらず、コマンドファイルが存在しなかったときに表示されます。
このとき、機種固有のコマンドファイルを生成することはできません。



5.8.3 デバッガの起動方法

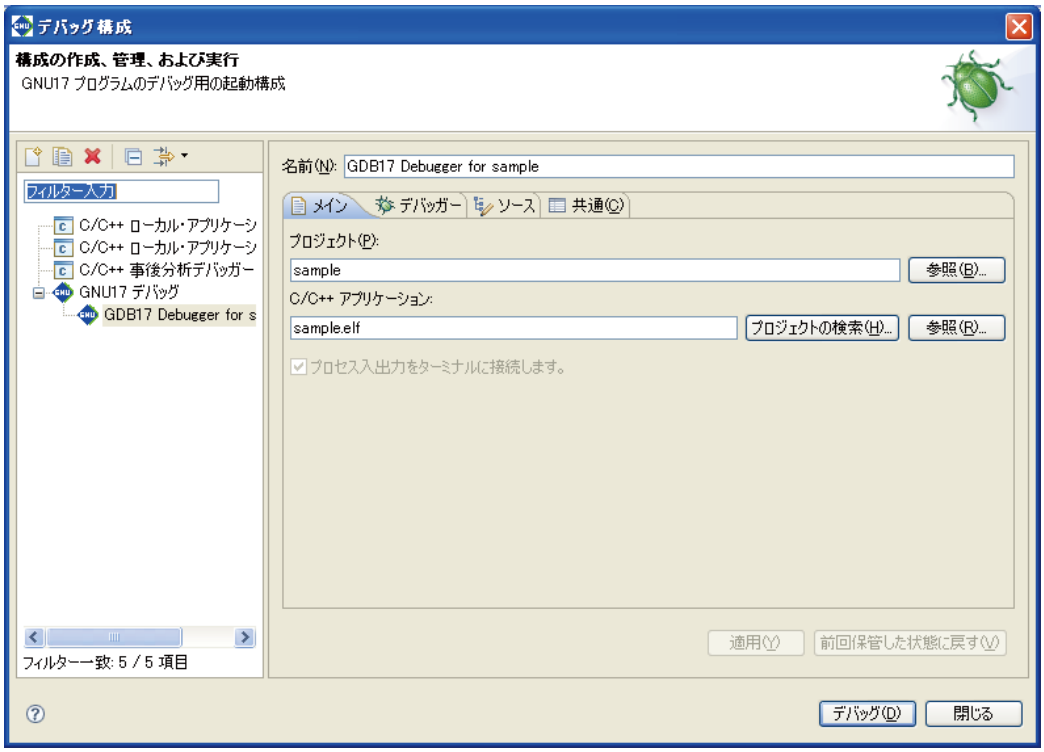
ビルドによって実行ファイル(.elf)が生成され、前節までの準備が完了すると、デバッグを開始できます。
デバッグの開始は、起動構成画面から行います。
起動構成画面では、デバッグを開始するための各種設定を行い、デバッガ(GDB)を起動します

●デバッグ開始

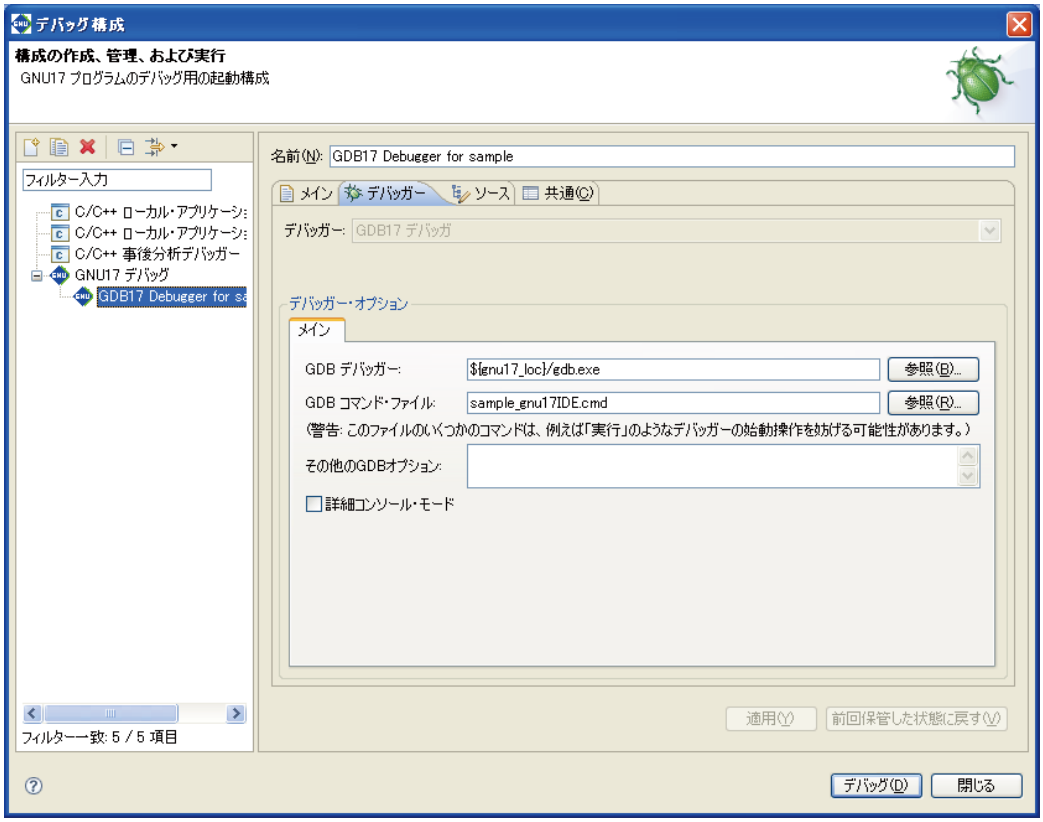
現在のプロジェクトで最初にデバッガを起動する場合の手順は次のとおりです。

(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[GDB17 Debugger for <プロジェクト名>]を選択します。

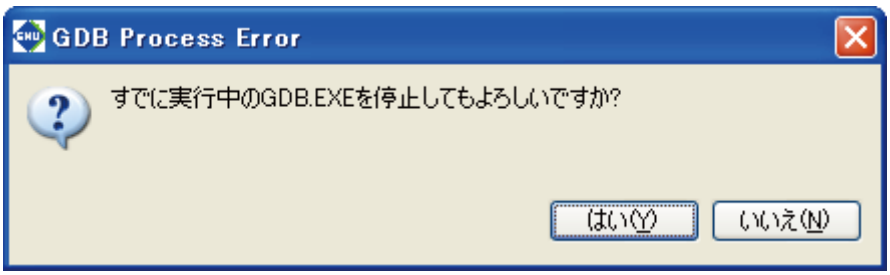


(3) 必要に応じて[デバッガ]タブの[その他のGDBオプション:] (デバッガコマンドラインの引数)を変更します。
IDEが生成したコマンドファイルを使用してデバッガを起動する場合は、変更する必要はありません。



(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。

注：デバッガ起動時、起動オプションに二重起動オプション(`--c17_double_starting`)が無く、すでにデバッガ(`GDB.EXE`)が実行中の場合、実行中のデバッガを停止させるかどうかの確認メッセージを表示します。



[はい]ボタンをクリックすると実行中のデバッガを停止させた後、デバッガを起動します。

●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[ヒストリーのデバッグ]の一覧から前回の[GDB17 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[GNU17 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、[GNU17 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、GNU17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。

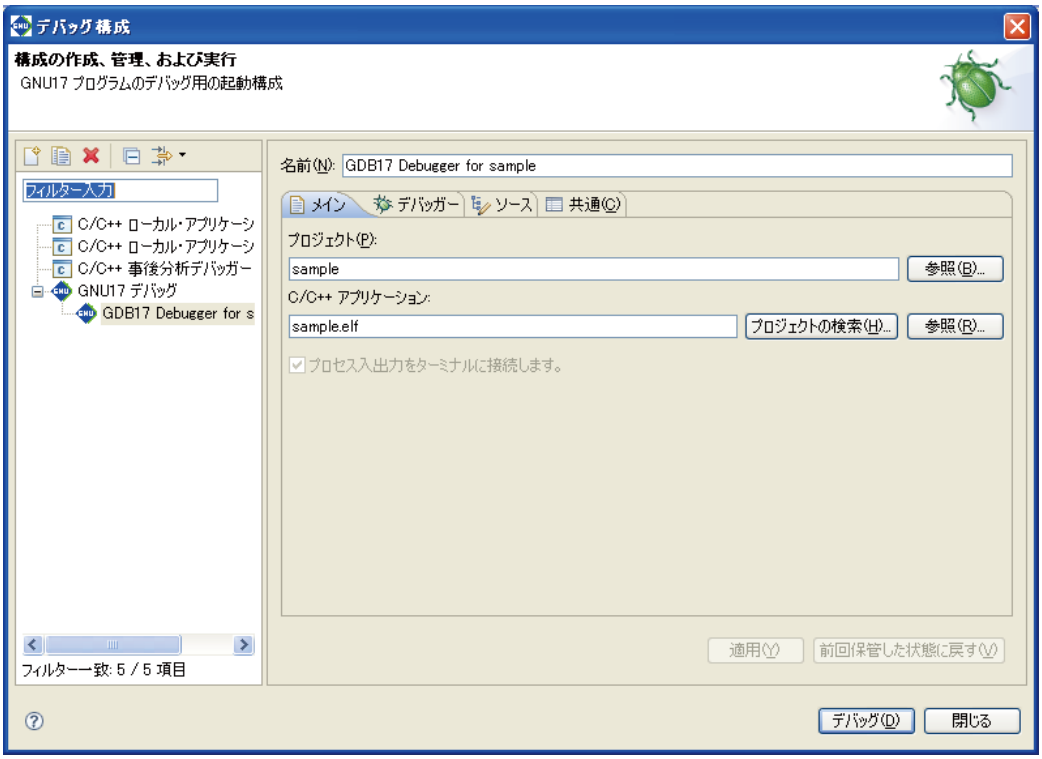


表5.8.3.1 起動構成画面のタブ一覧

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。

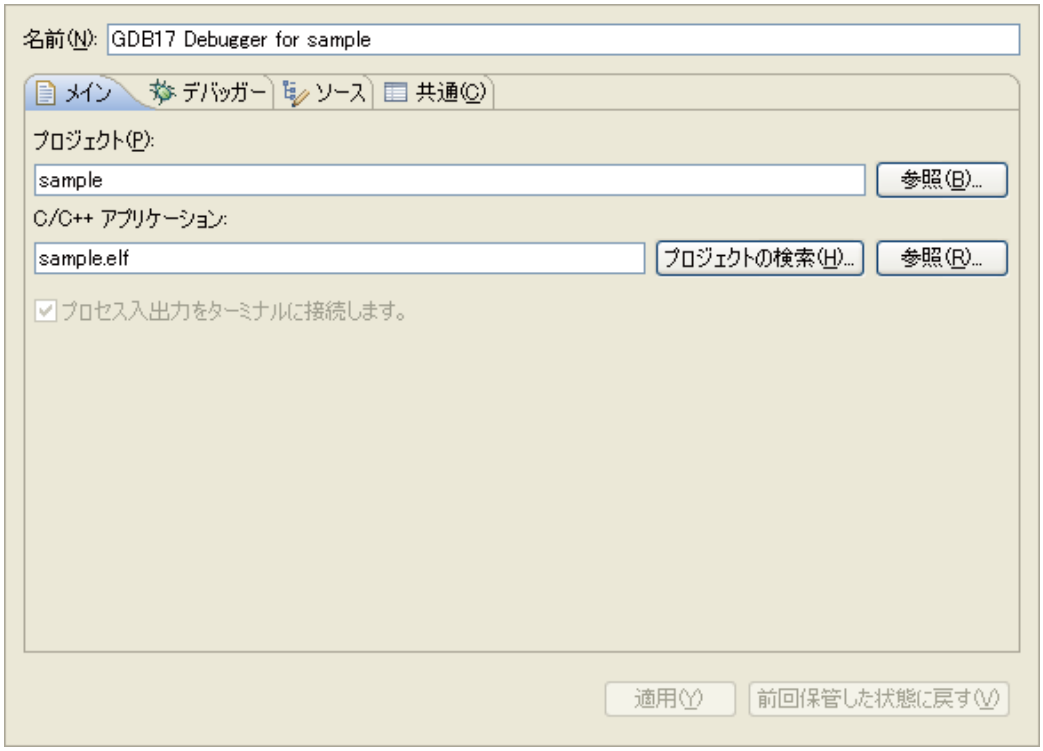


表5.8.3.2 [メイン]タブ

入力項目	内容
Project	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)

[デバッガー]タブ

デバッガ(GDB)へのパス・引数を設定します。

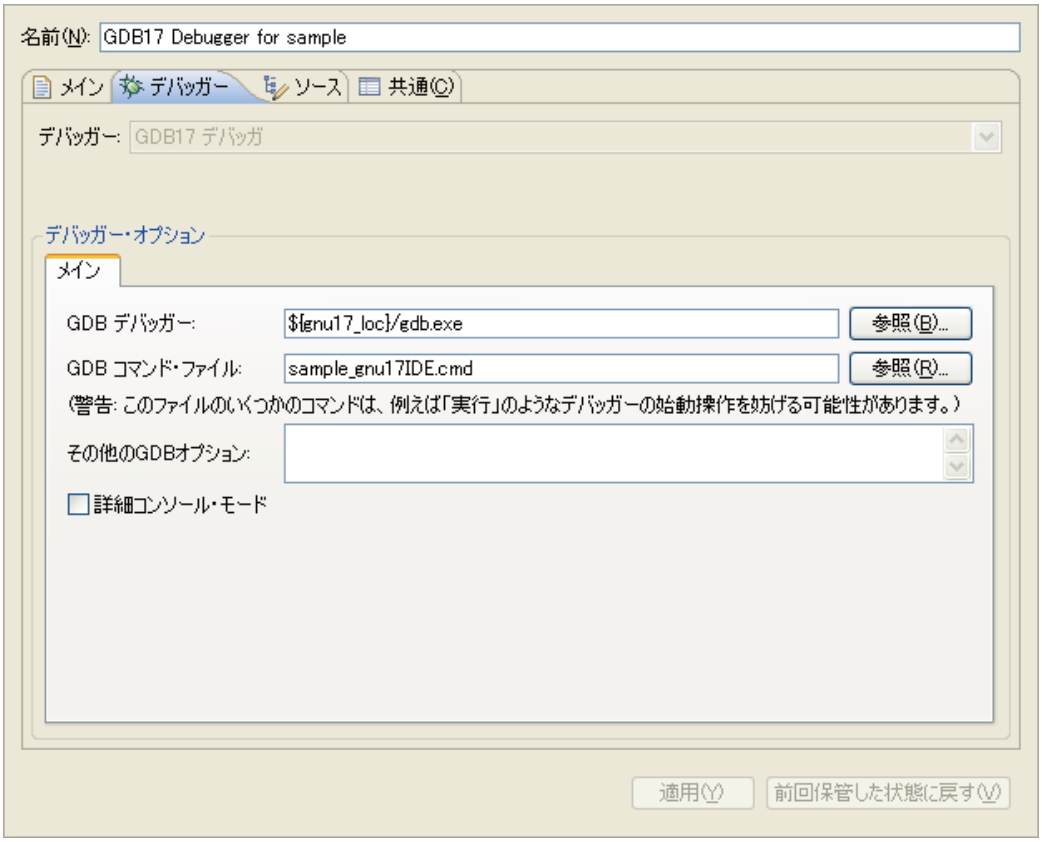


表5.8.3.3 [デバッガー]タブ

入力項目	内容
デバッガー	起動するデバッガの種類を選択します。 GNU17 Debuggerに固定です。
GDBデバッガー	起動するデバッガのパスを表示・入力します。 (デフォルトは${gnu17_loc}/gdb.exe) [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。その場合、デバッガのパスは絶対パス表示となります。
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) --c17_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する。 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。

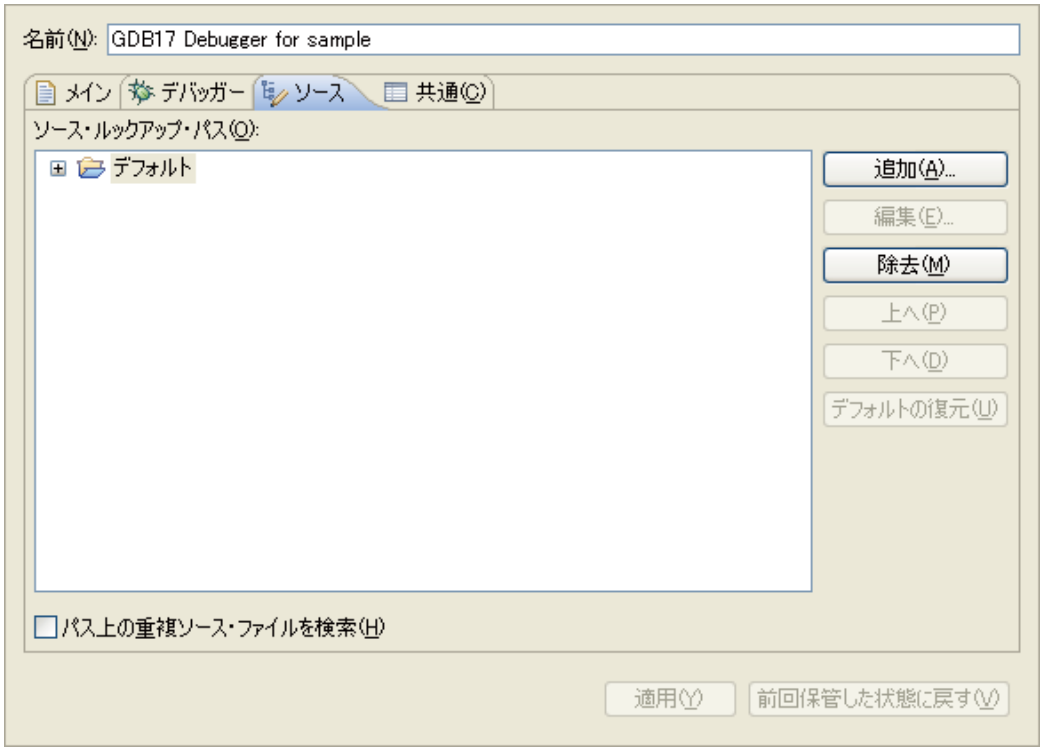


表5.8.3.4 [ソース]タブ

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。

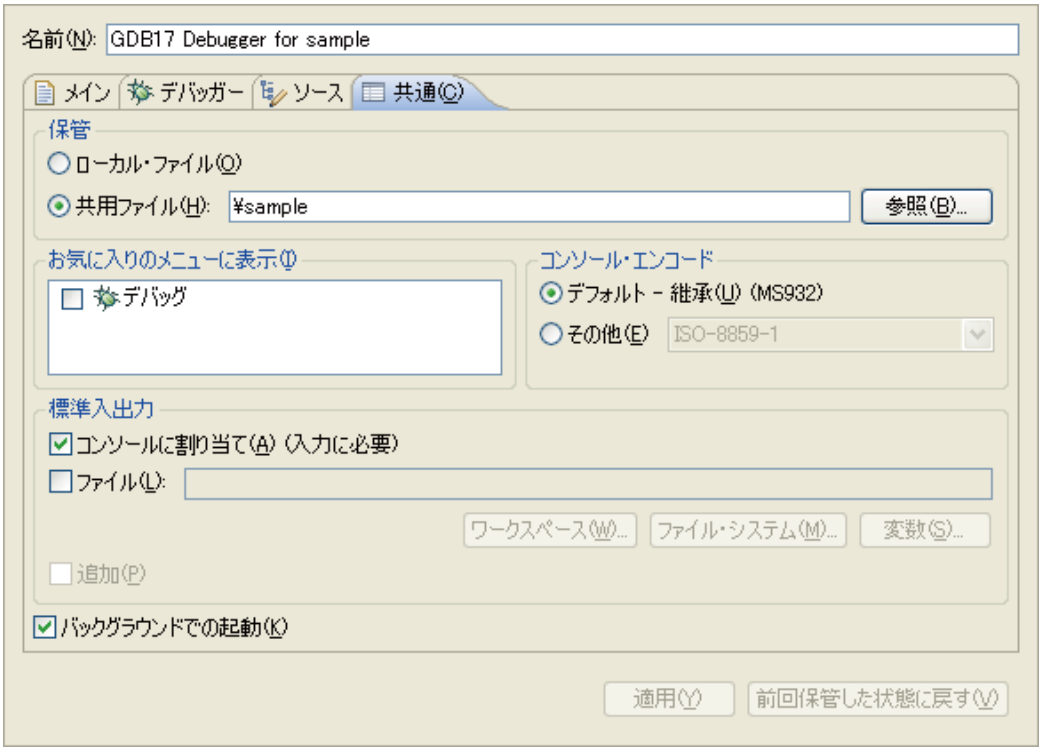


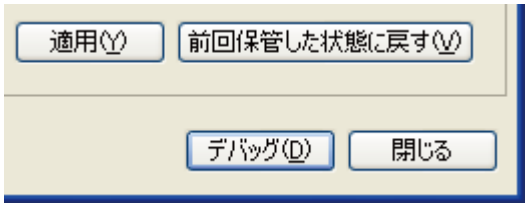
表5.8.3.5 [共通]タブ

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、"GDB17 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]

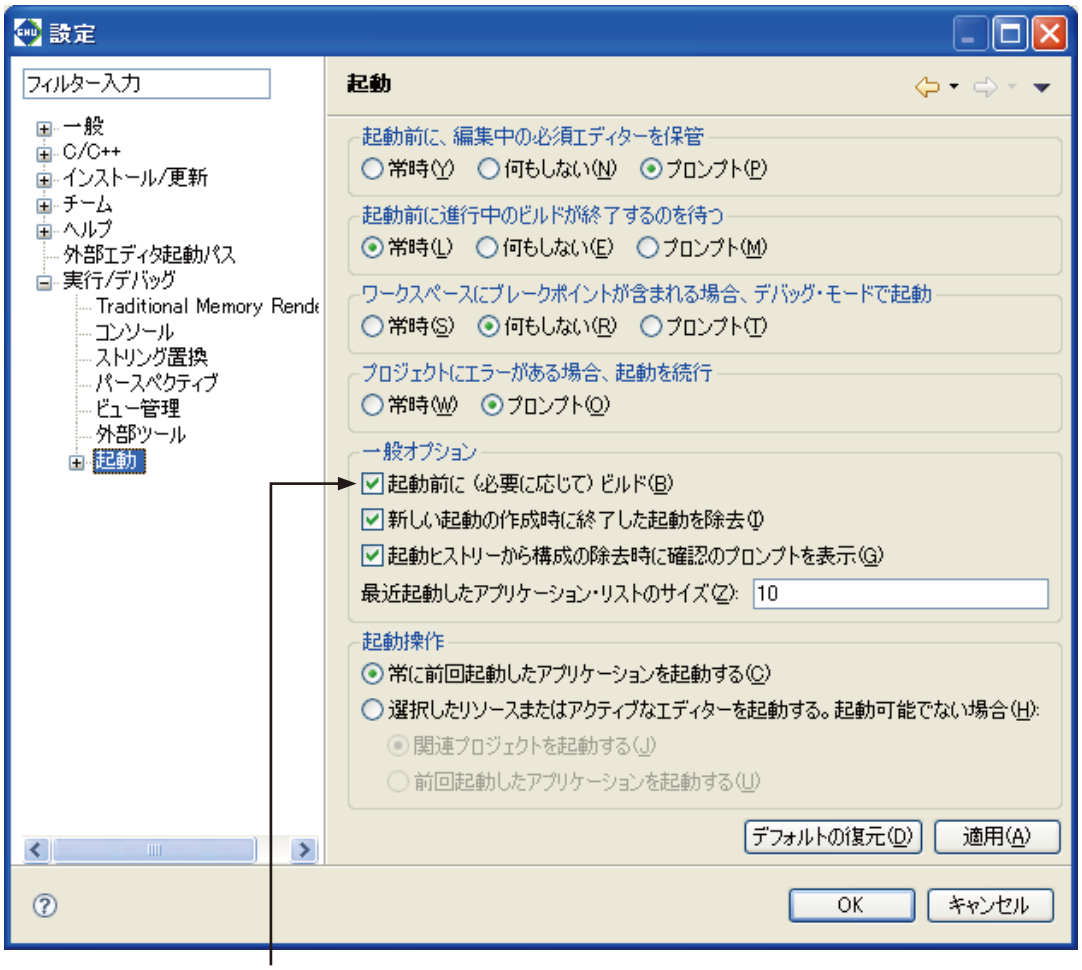
タブ画面の設定を確定し、デバッガを起動します。
初回の起動後は、当ボタンからではなく、ツールバーもしくは[実行]メニューのショートカットから起動できます。
[ブレークポイント]ビュー一覧にブレークポイントがある場合、さらにブレークポイントをデバッガに対して設定します。

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ガを起動します。

●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。

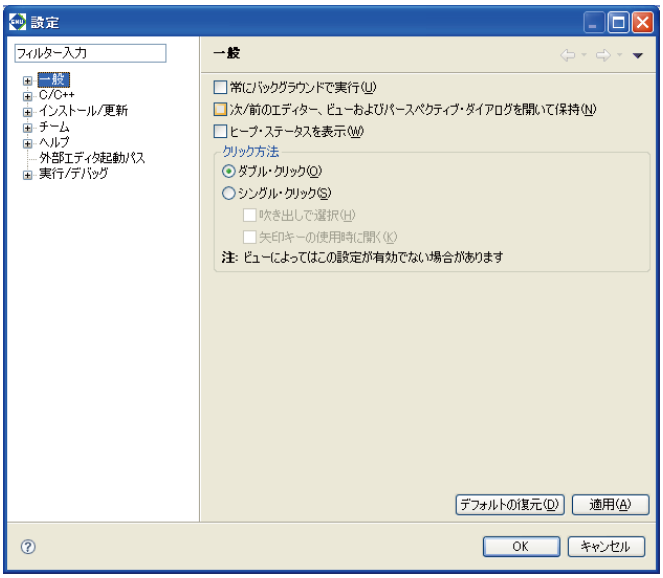
(1) [ウィンドウ]メニューから[設定...]を選択し、[設定]ダイアログを表示させ、[実行/デバッグ]>[起動]のページを表示させます。



(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。

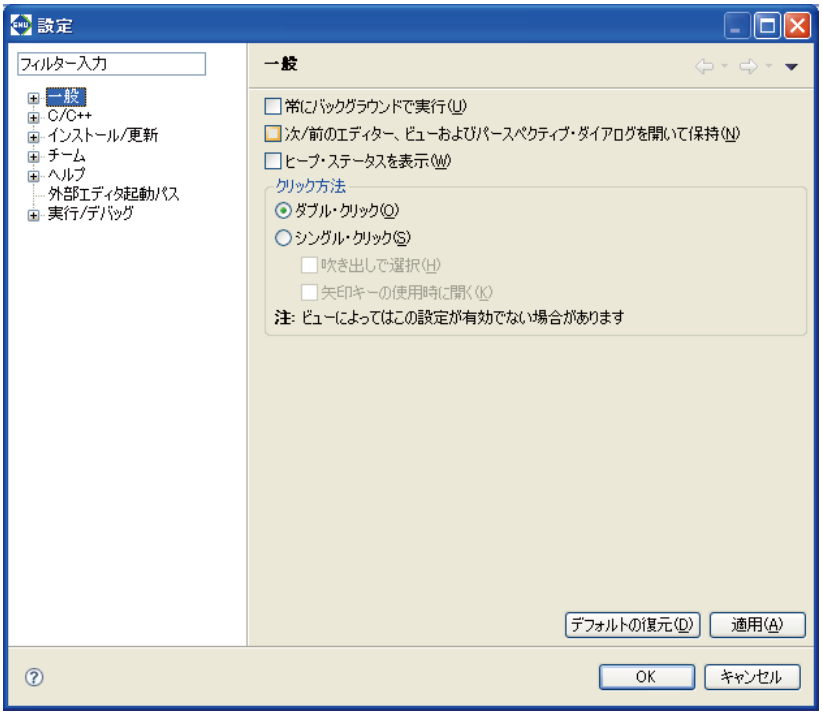
[適用]
そのページで設定した内容を適用します。他の項目を変更する場合は、[適用]ボタンをクリックしてから新たなページに移動してください。

[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。

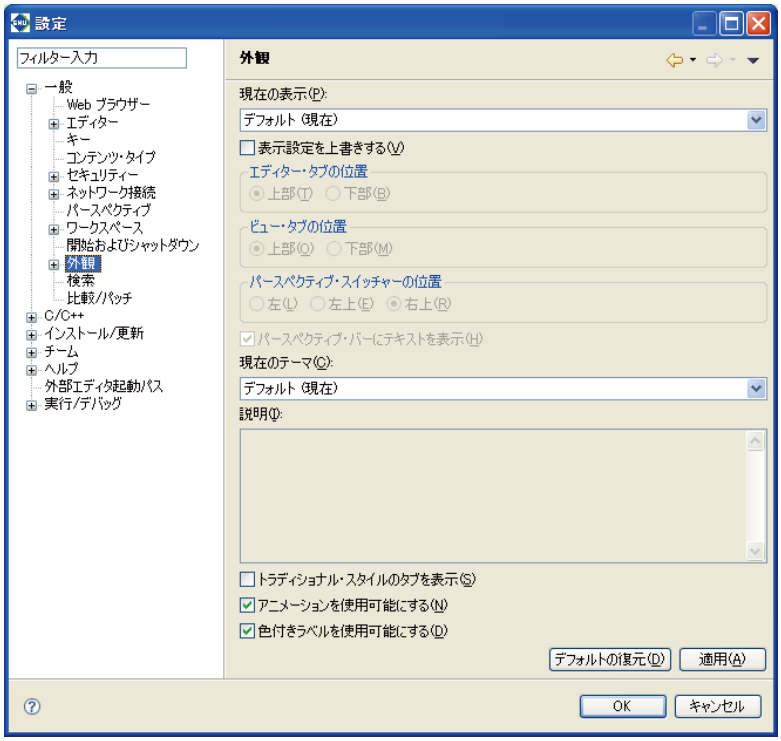
[ダブル・クリック]（デフォルト: ON）
リソースをシングルクリックすると選択され、ダブルクリックすると開きます。

[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 > 外観



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。

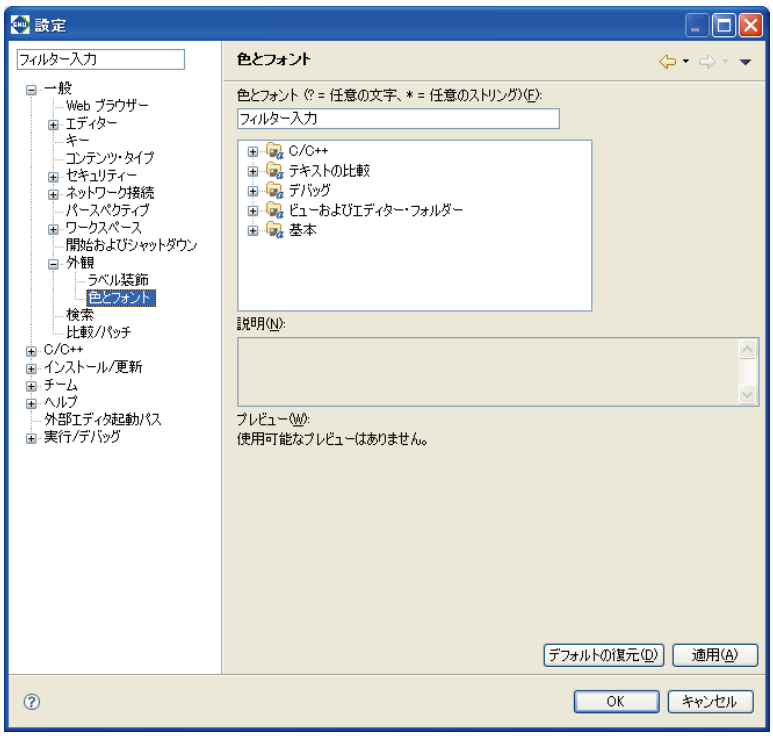
[パースペクティブ・スイッチャーの位置]
パースペクティブバーの表示位置を指定します。
[左] (デフォルト: OFF)
ウィンドウの左端に表示します。
[左上] (デフォルト: OFF)
ウィンドウの左上部(ツールバーの下)に表示します。
[右上](デフォルト: ON)
ウィンドウの右上部(ツールバーの右)に表示します。

[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。

[説明:]
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。

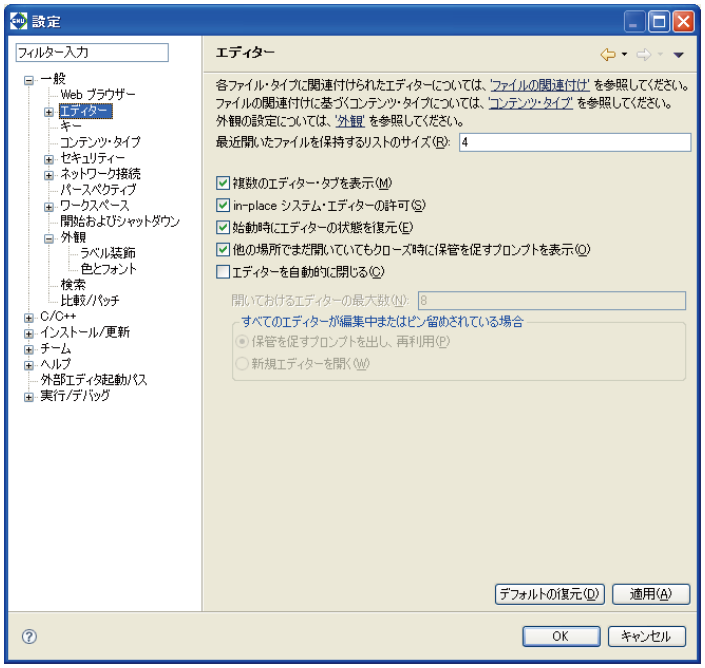
[リセット]
変更したフォントをデフォルト設定に戻します。フォントが変更されている場合にアクティブになります。

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。

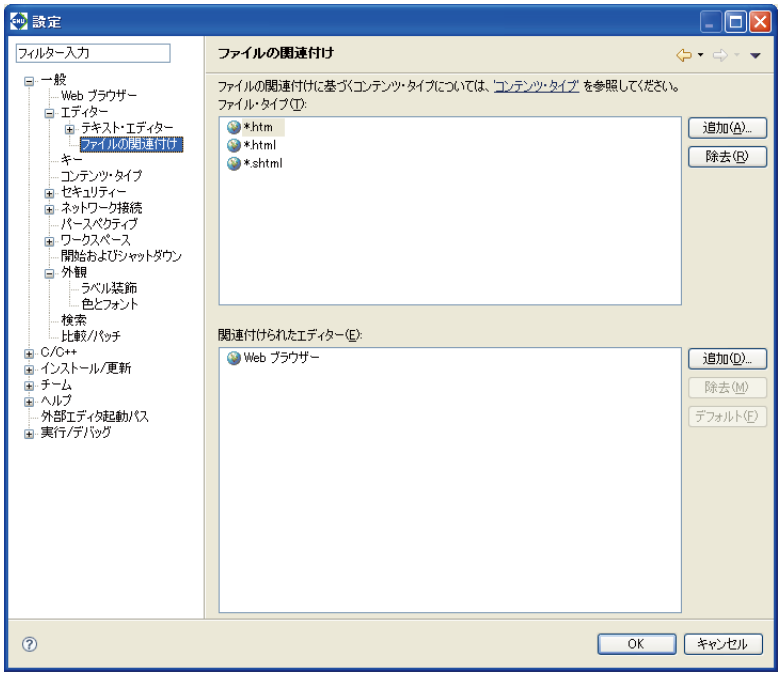
[開いておけるエディターの最大数](デフォルト: 8)
エディタで何個以上のリソースが開かれた場合に上記の自動クローズ機能を有効にするかを指定します。このフィールドは[エディターを自動的に閉じる]が選択された場合にのみ設定可能です。

[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。

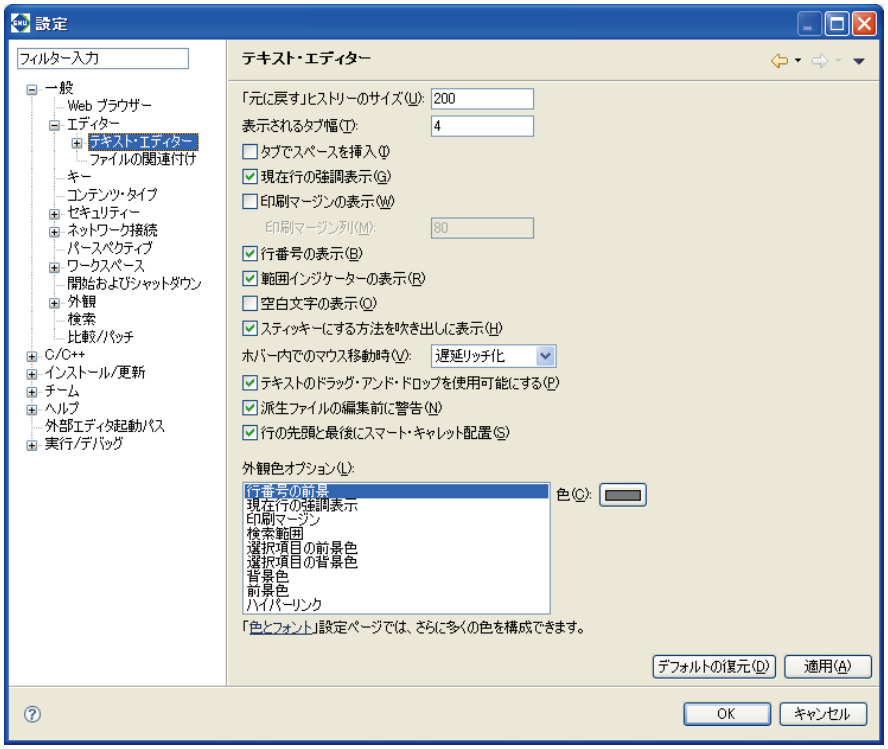
[除去]

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」ヒストリーのサイズ:]（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([Print margin column:]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。

[範囲インジケーターの表示]（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。

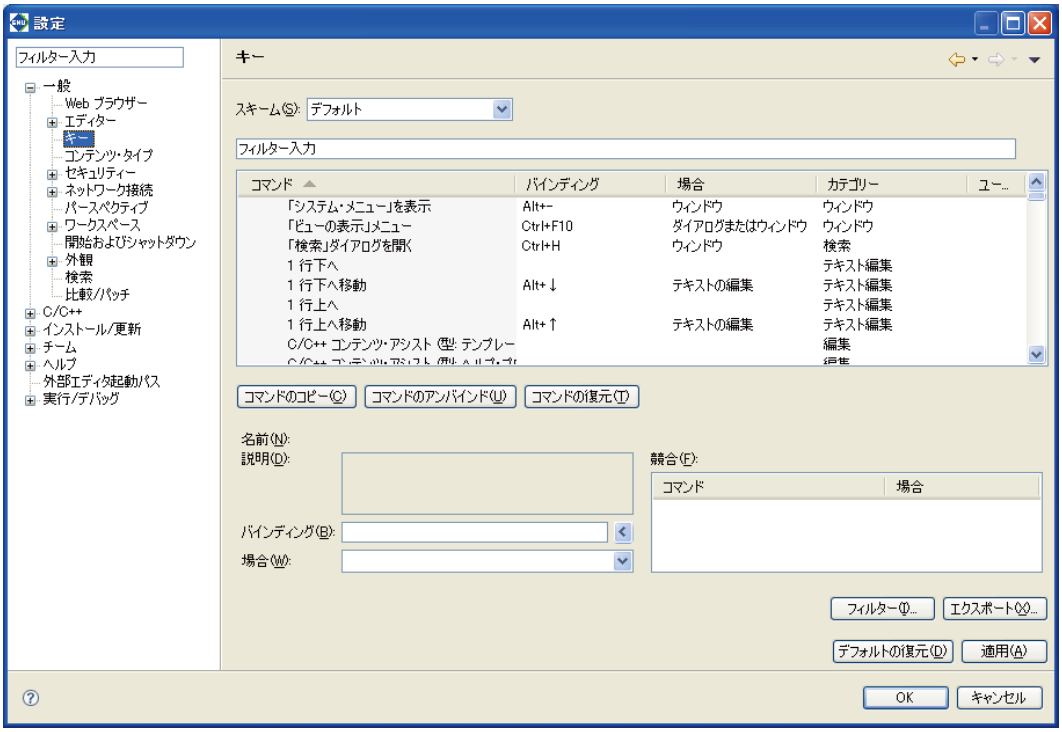
行番号の前景	行番号の文字色
現在行の強調表示	現在行のハイライト色
印刷マージン	プリントマージンの垂直線の色
検索範囲	検索範囲
選択項目の前景色	選択範囲の文字色*

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すことでキーシーケンスを割り当てることができます。

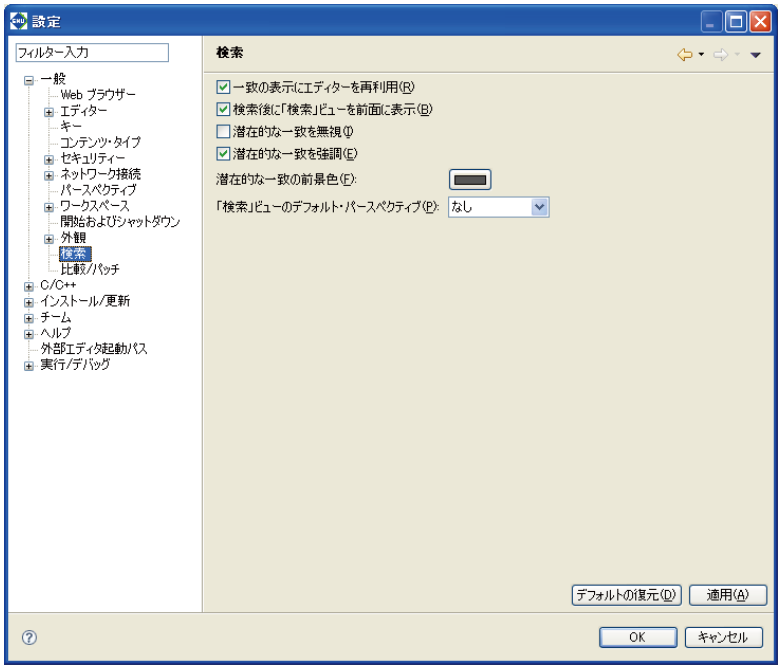
[場合:]

[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



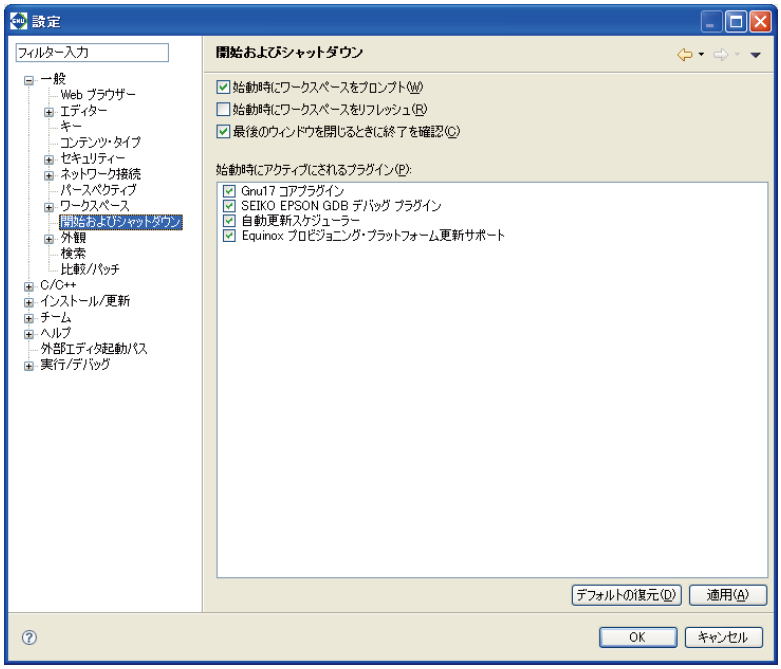
[検索]メニュー /ボタンによる検索に関わる設定を行います。

[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。

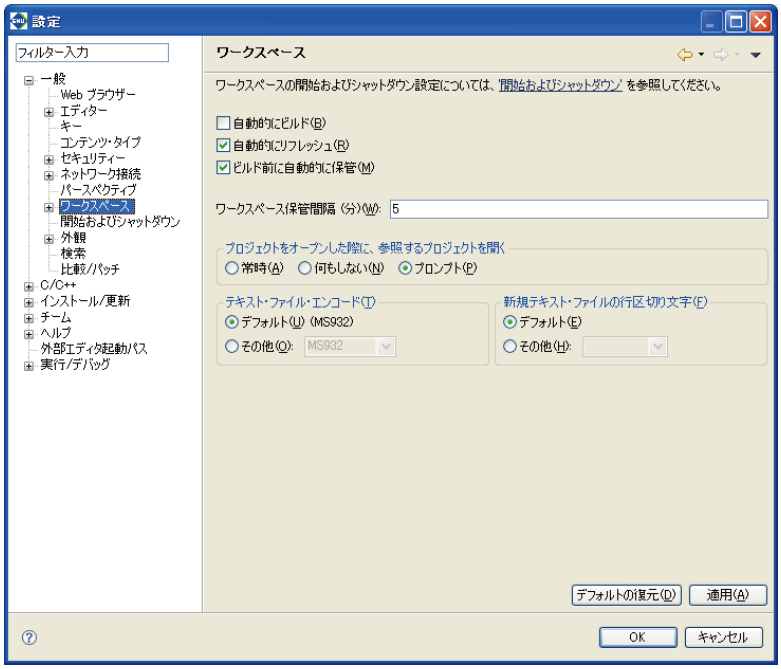
[最後のウィンドウを閉じるときに終了を確認]（デフォルト: ON）

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。

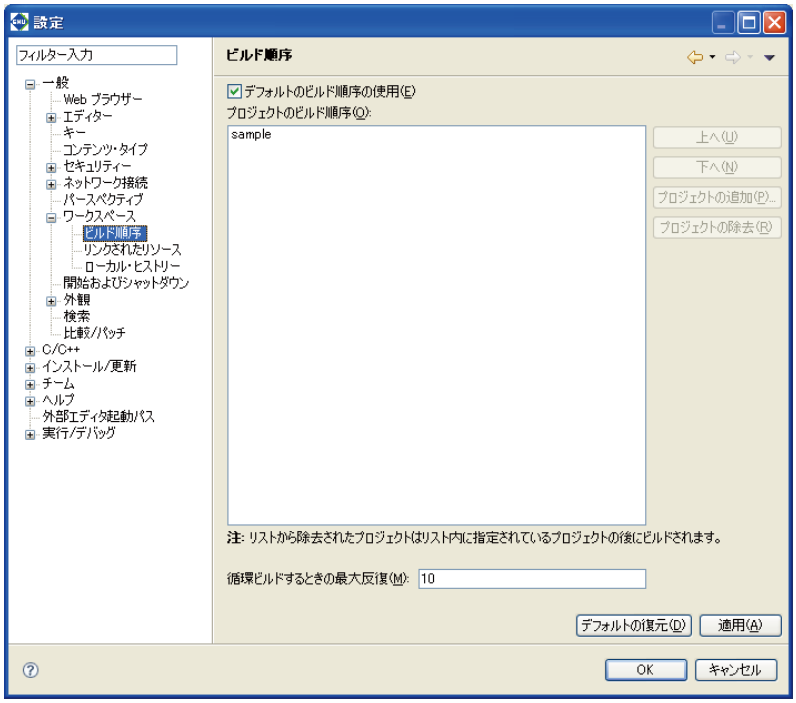
[デフォルト]

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。

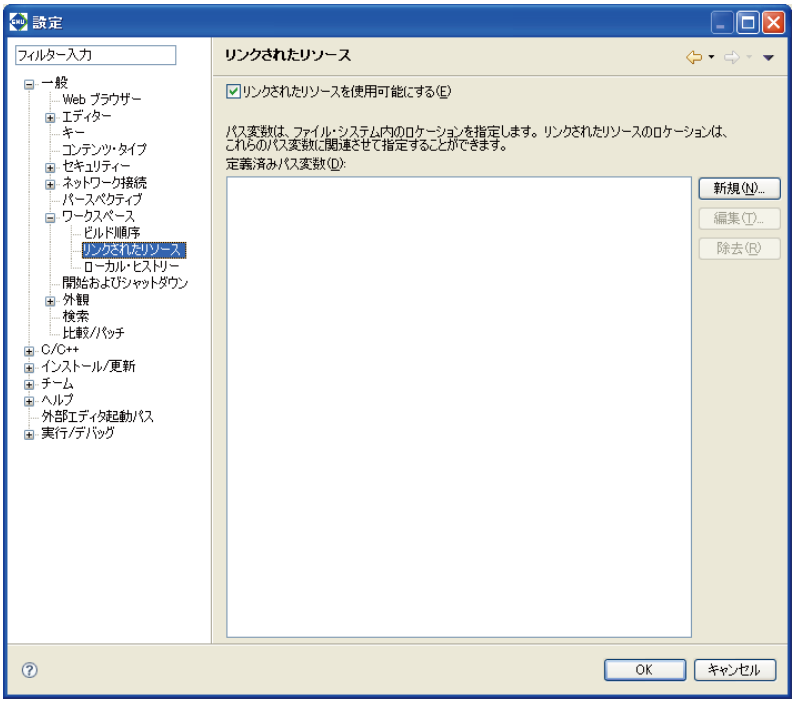
[プロジェクトの追加...]

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



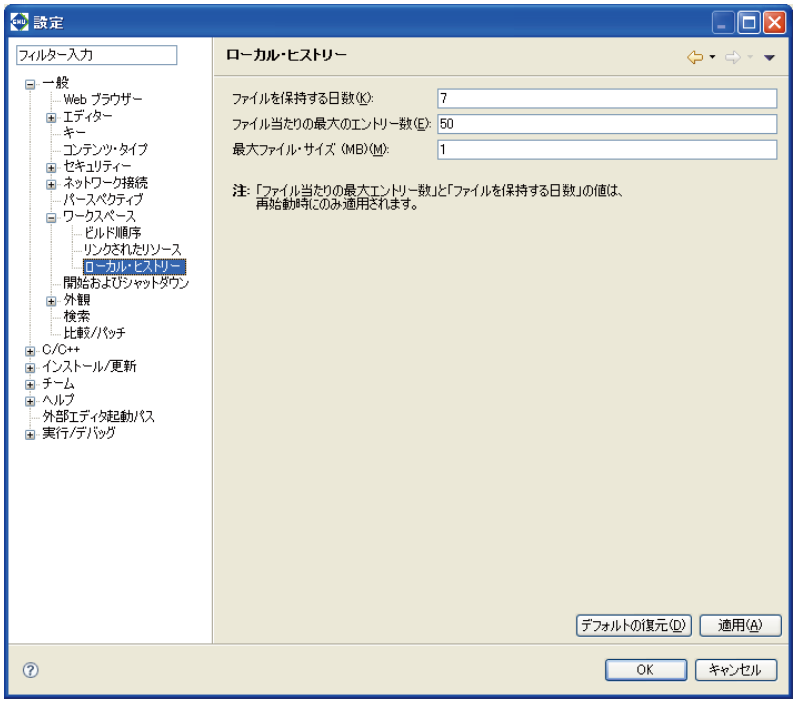
リンクリソースのON/OFFを切り替えます。

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。

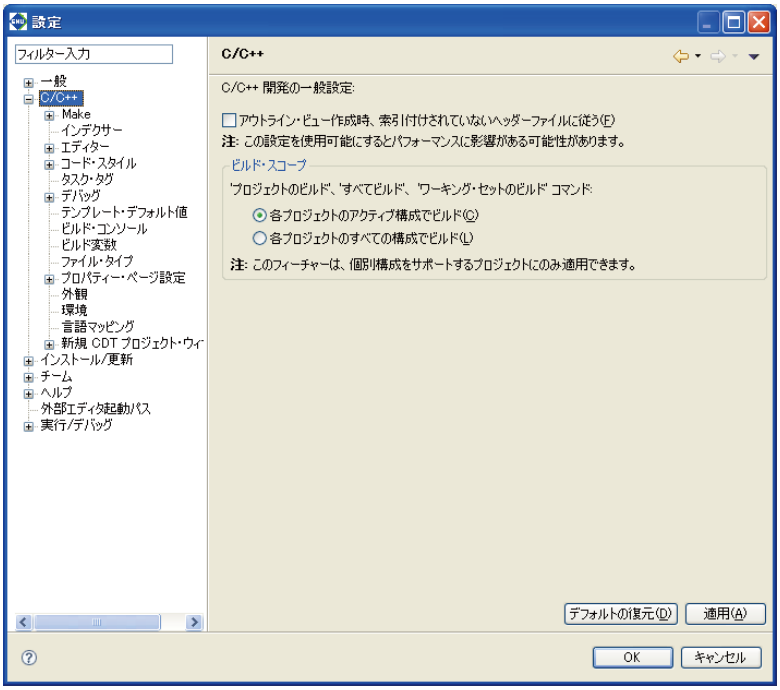
[ファイルを保持する日数:]（デフォルト: 7日）
変更履歴を保持しておく日数を設定します。

[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。

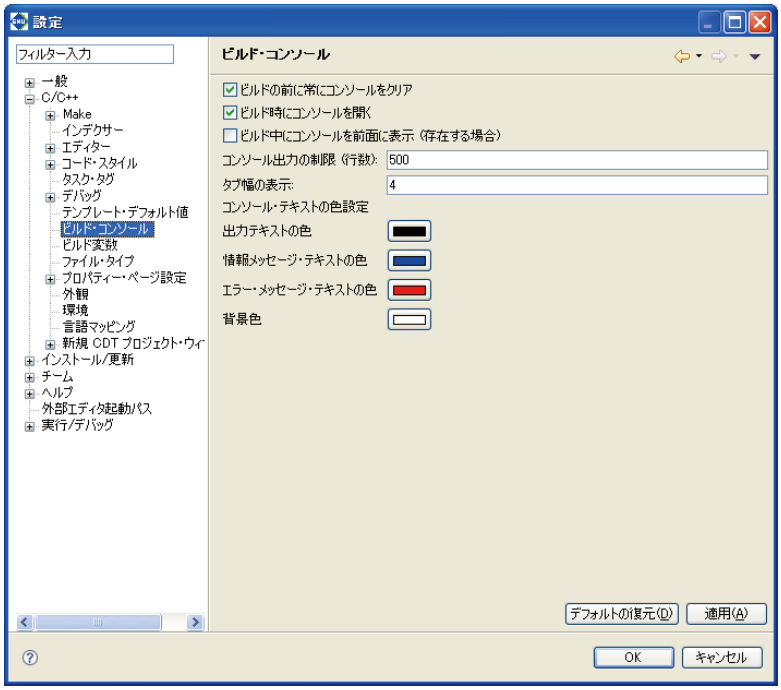
設定した上限を超えた履歴は消去されます。
変更した設定を有効にするには、**IDE**を再起動してください。

●C/C++



この画面は操作しないでください。

●CC++ > 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]

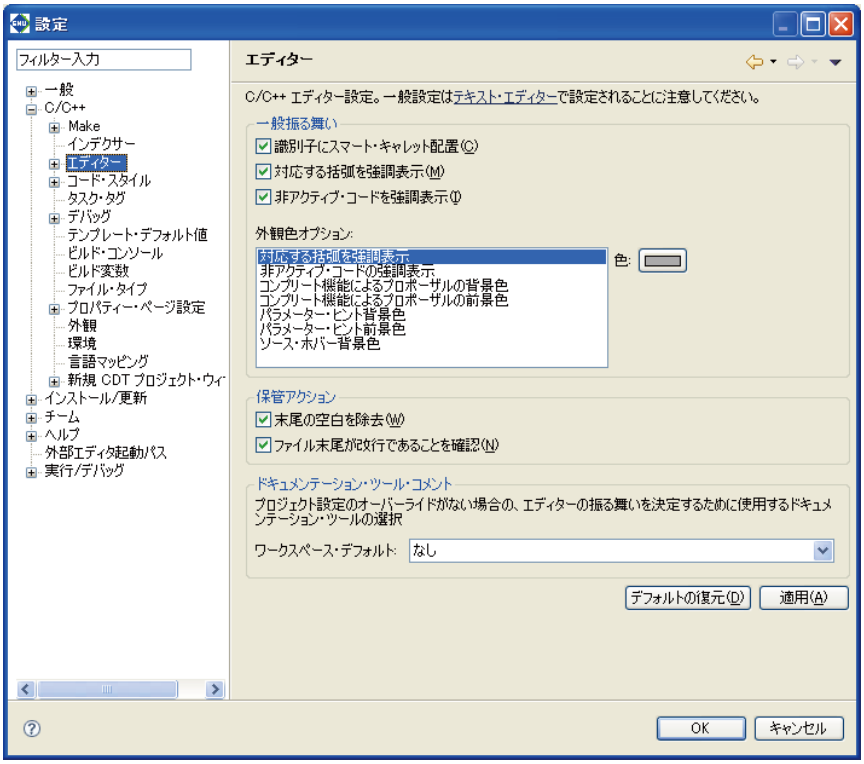
[出力テキストの色]
実行したコマンドラインの表示色です。

[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。



[一般振る舞い]

[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。

[保管アクション]

[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。

[ファイル末尾が改行であることを確認]

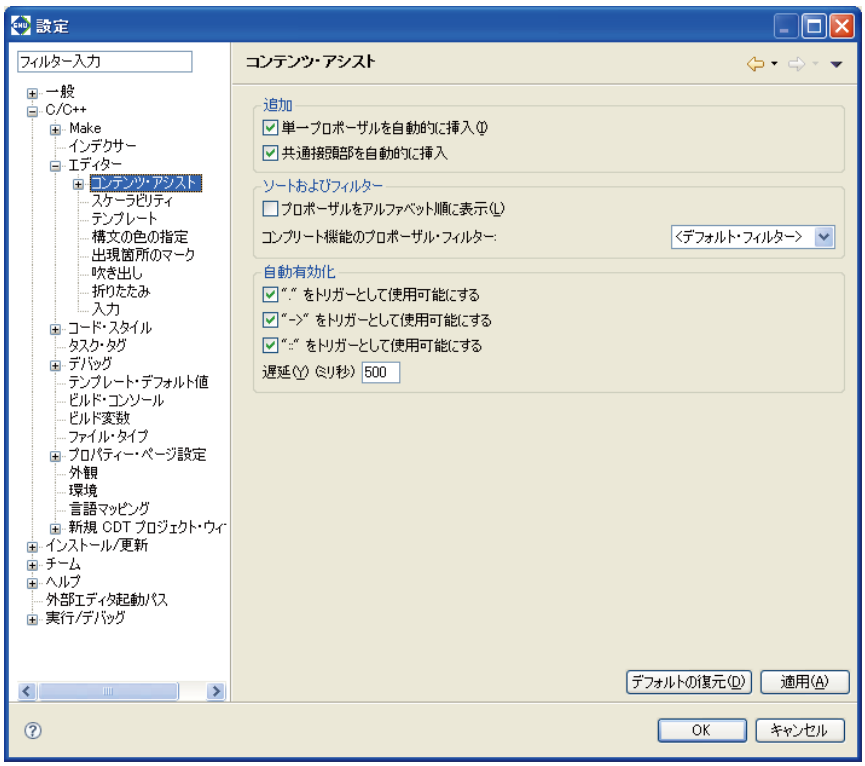
ファイル保存時に最終行が改行で終了するようにします。

[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。

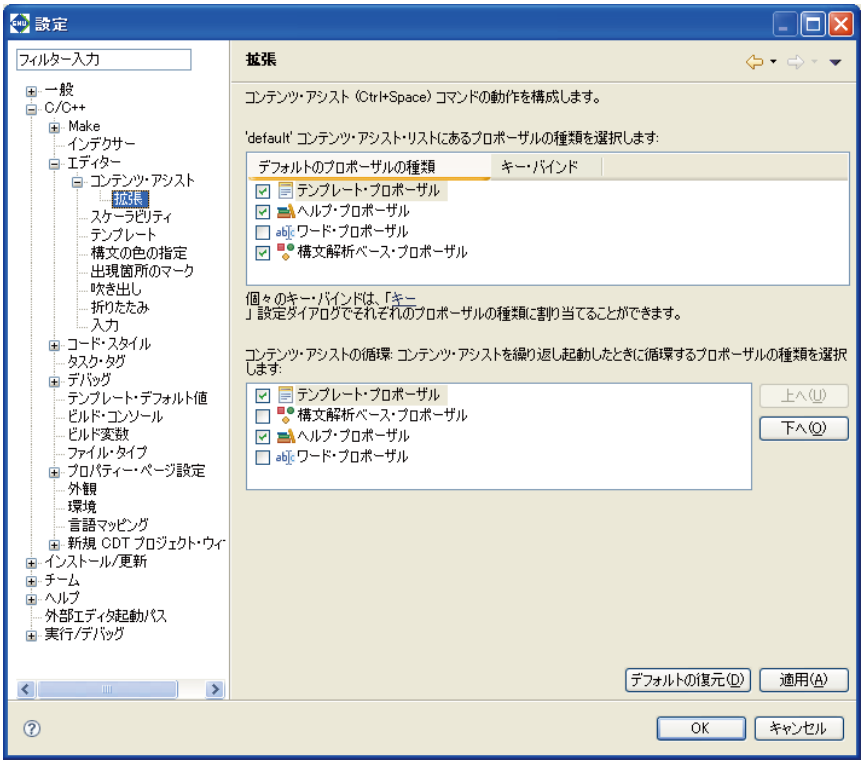
[自動有効化:]

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。

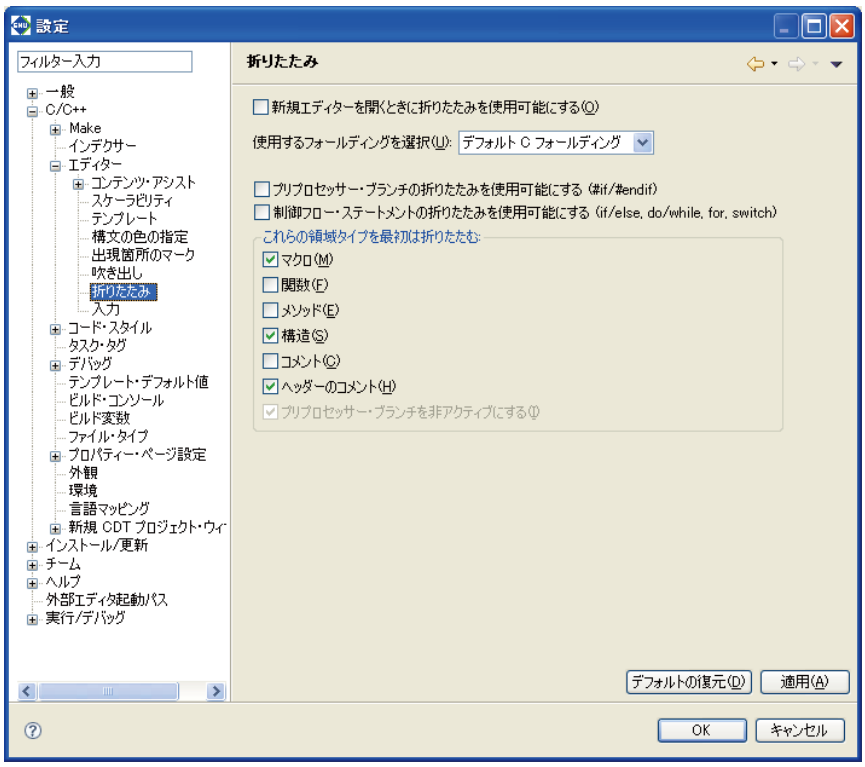
['default' コンテンツ・アシスト・リストにあるプロポーザルの種類を選択します:]

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]ボタンをクリックすることによって、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。

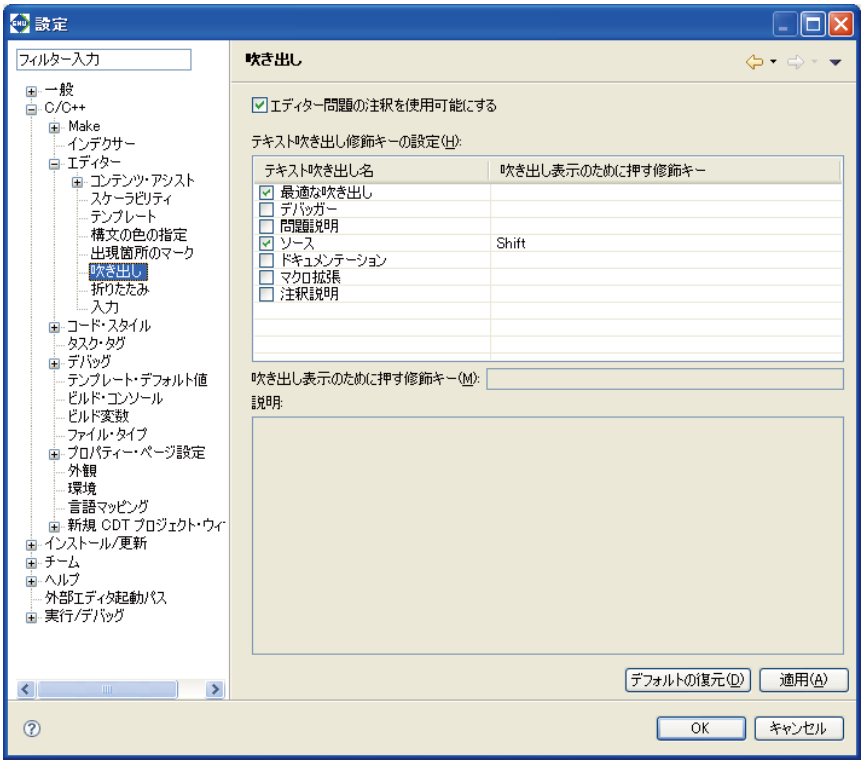
[プリプロセッサー・ブランチの折りたたみを使用可能にする (#if/#endif)](デフォルト：OFF)
　このチェックボックスを選択すると、プリプロセッサ分岐の折りたたみを行います。

[制御フロー・ステートメントの折りたたみを使用可能にする (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
[プロセッサー・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。

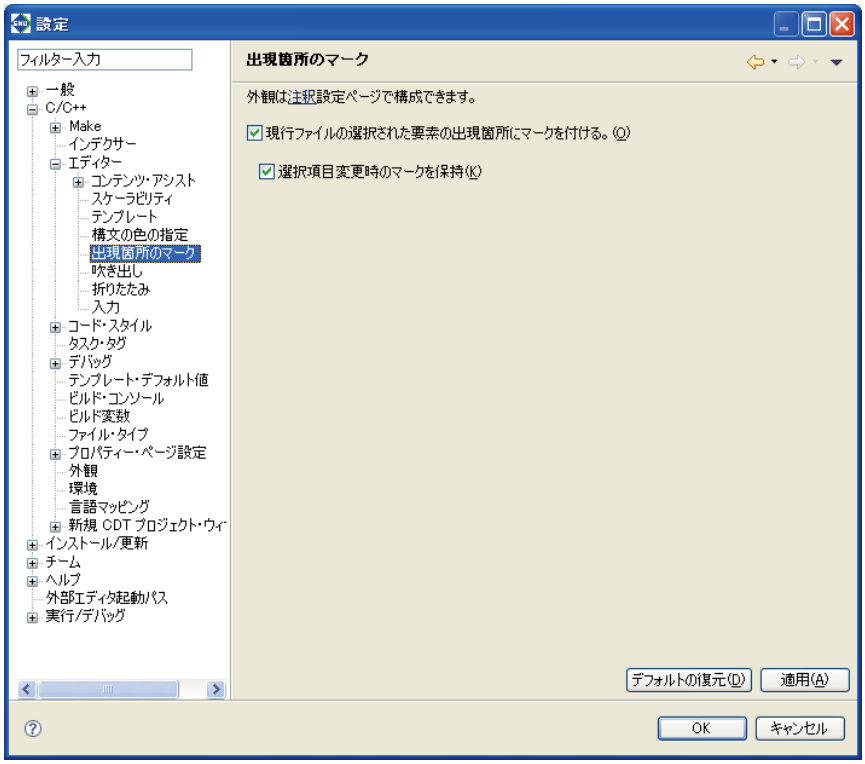
[エディター問題の注釈を使用可能にする](デフォルト：ON)

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



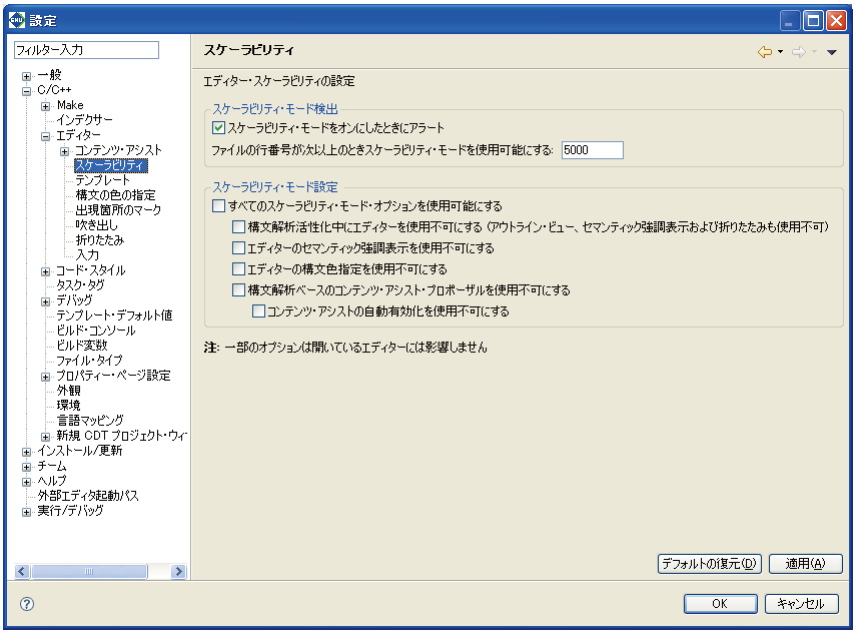
[**現行ファイルの選択された要素の出現箇所にマークを付ける。**](デフォルト：ON)

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を保持**](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。

[スケーラビリティ・モード検出]

[スケーラビリティ・モードをオンにしたときにアラート]
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。

[スケーラビリティ・モード設定]

[すべてのスケーラビリティ・モード・オプションを使用可能にする]
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。

[構文解析活性化中にエディターを使用不可にする]
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)。

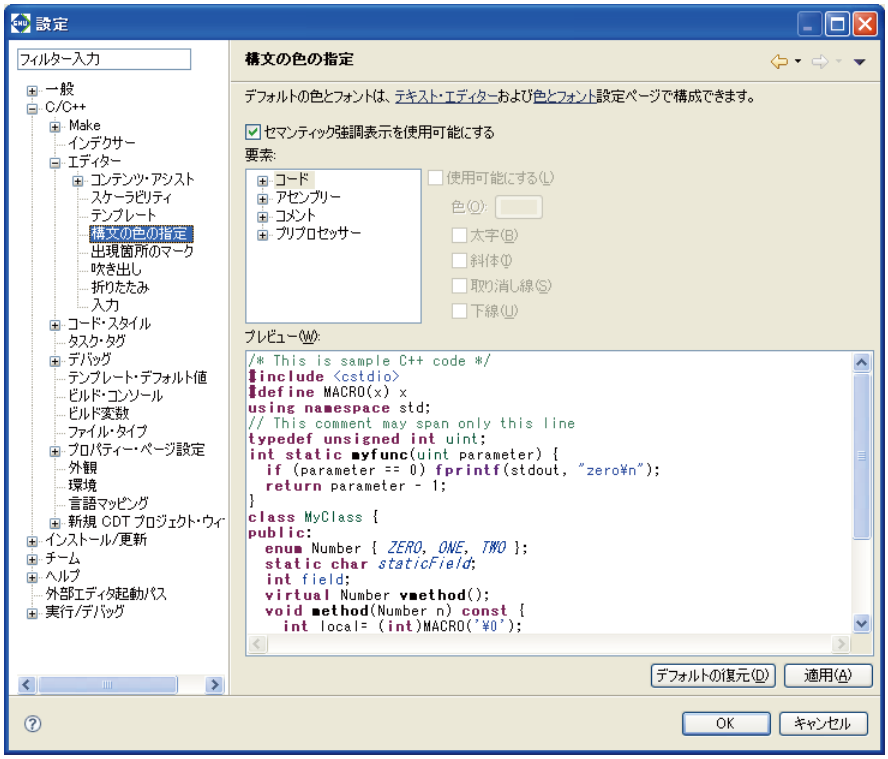
[エディターのセマンティック強調表示を使用不可にする]
エディタの強調表示を無効にします。

[エディターの構文色指定を使用不可にする]
エディタの構文色分けを無効にします。

[構文解析ベースのコンテンツ・アシスト・プロポーザルを使用不可にする]
構文解析ベースのコンテンツアシストを無効にします。

[コンテンツ・アシストの自動有効化を使用不可にする]
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。

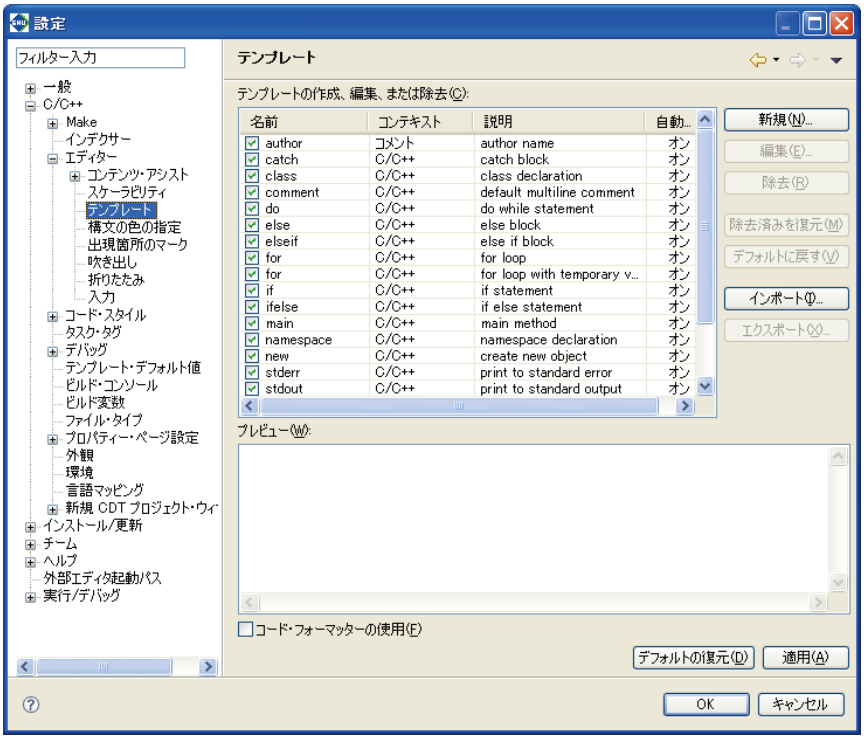
[セマンティック強調表示を使用可能にする](デフォルト：ON)
　このチェックボックスを選択すると、カラーの設定を有効にします。

[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ソース内に記述可能なことを示します。

[説明]

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 :]

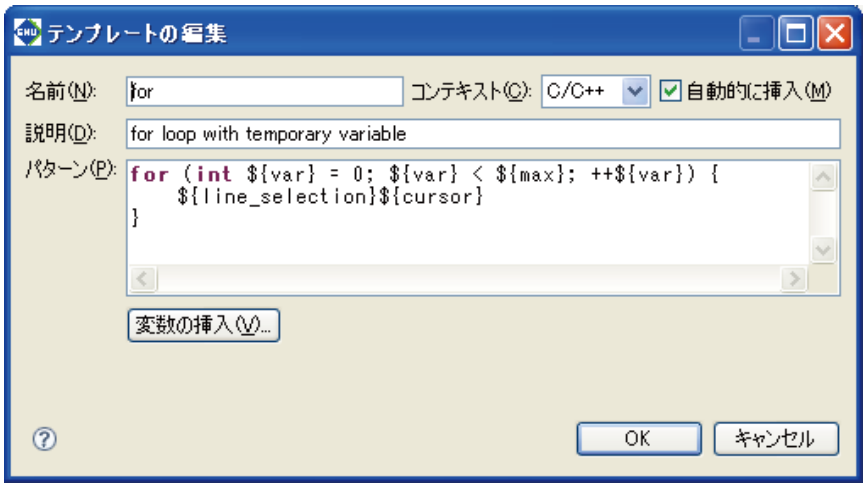
一覧内で選択されているテンプレートの内容を表示します。

[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。

[説明:]
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]ボタンで入力/編集した内容を確定します。[キャンセル]ボタンで作成/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。

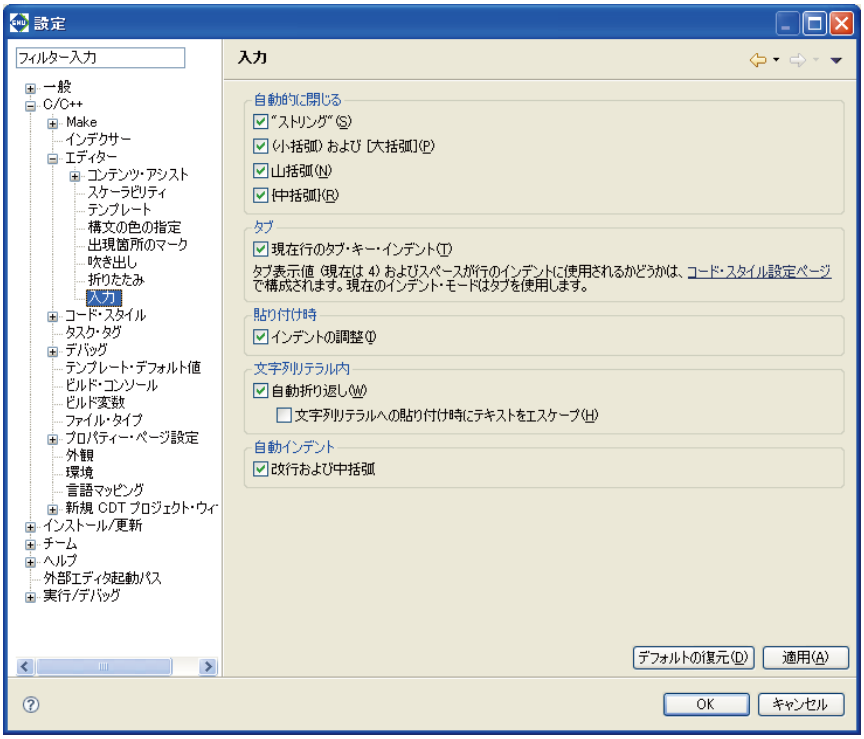
[除去済みを復元]
削除したテンプレートを復帰します。

[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。

[自動インデント]
自動インデントを設定します。

[改行および中括弧](デフォルト：ON)
このチェックボックスを選択すると、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
プロパティー・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]ボタンと[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ー

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[プロパティー]を選択すると表示されます。左のプロパティ一覧に15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。

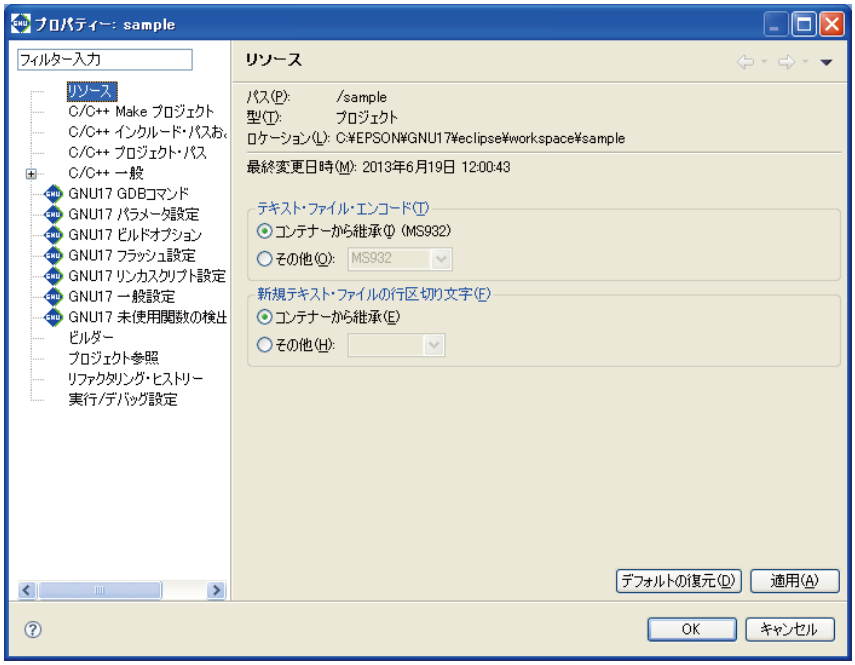
[適用] *
そのページで設定した内容を適用します。他のプロパティを変更する場合は、[適用]ボタンをクリックしてから新たなページに移動してください。

[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。

* プログラムのビルドに関する設定を変更後に[OK]ボタンまたは[適用]ボタンをクリックした場合は、"clean"ビルド(5.7.13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。

[コンテナーから継承 (*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。

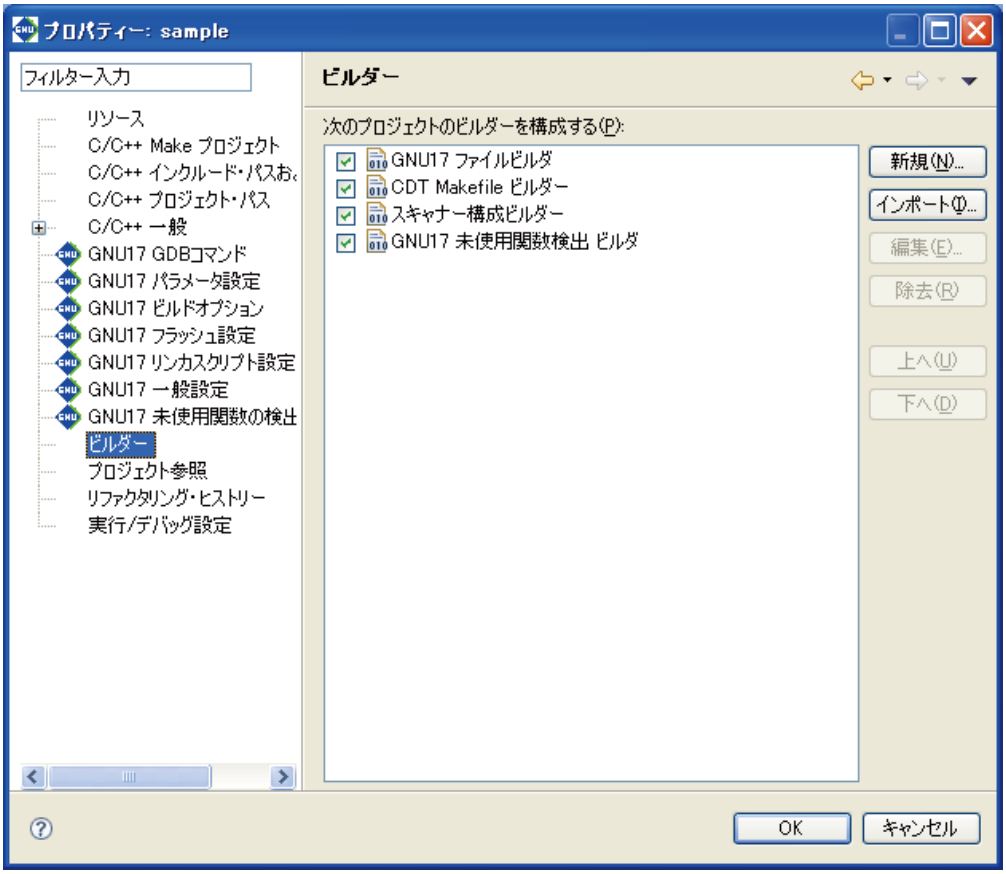
[コンテナーから継承]

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。

[GNU17 ファイルビルダ]

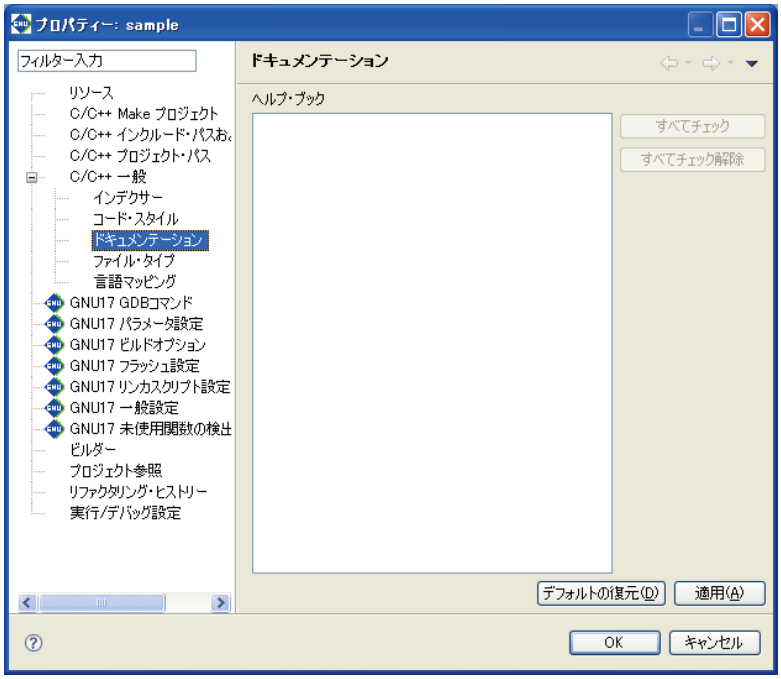
ビルド実行時にmakeファイルなどビルドに必要なファイルの自動生成を行わない場合にチェックを外します("5.7.14 ユーザ作成のmakeファイルを使用するには"参照)。それ以外は、本ページの設定を変更しないでください。

[GNU17 未使用関数検出 ビルダ]

ビルド実行時にプロジェクト内で一度も使用されていないC言語の関数を検出します。プロジェクトプロパティの[GNU17 未使用関数の検出]ページにて[Splintによる未使用の関数を検出する。(C言語のみ)]のチェックを外すことによりこのビルダーを削除します。そのため本ページで[GNU17未使用関数検出 ビルダ]のチェックを外さないでください("5.7.11 未使用関数の検出"参照)。

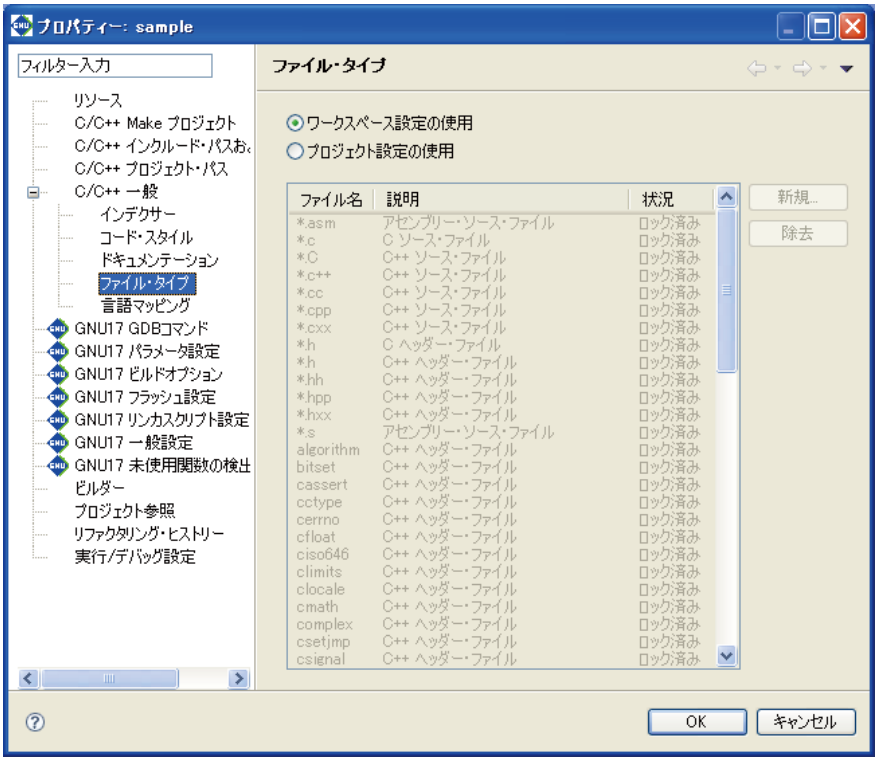
また、このビルダーの位置を移動しないでください。このビルダーの位置を移動すると、未使用関数を検出した結果が表示されない場合があります。

●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。

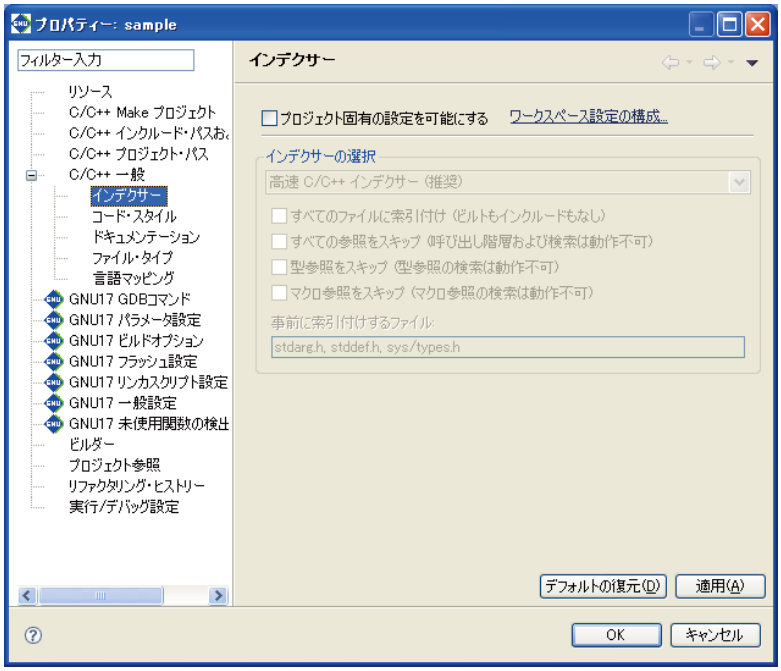
[ワークスペース設定の使用]

[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]

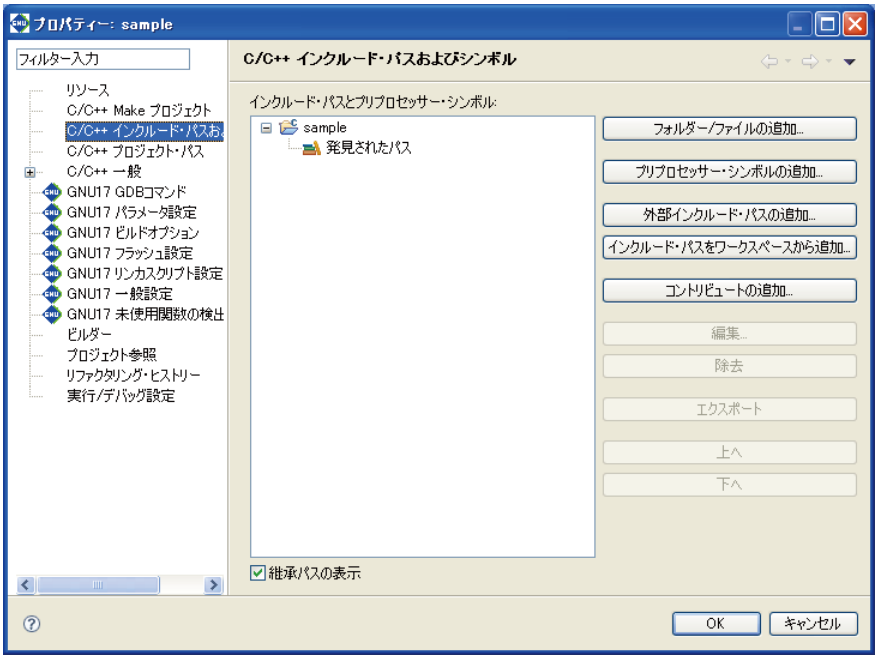
プロジェクト独自のファイル形式セットを使用する場合にラジオボタンを選択します。新しいファイル拡張子は[新規...]ボタンで追加します。不要な拡張子とファイル名は、一覧から選択して[除去]ボタンで削除できます。

●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。このページの設定は変更しないでください。

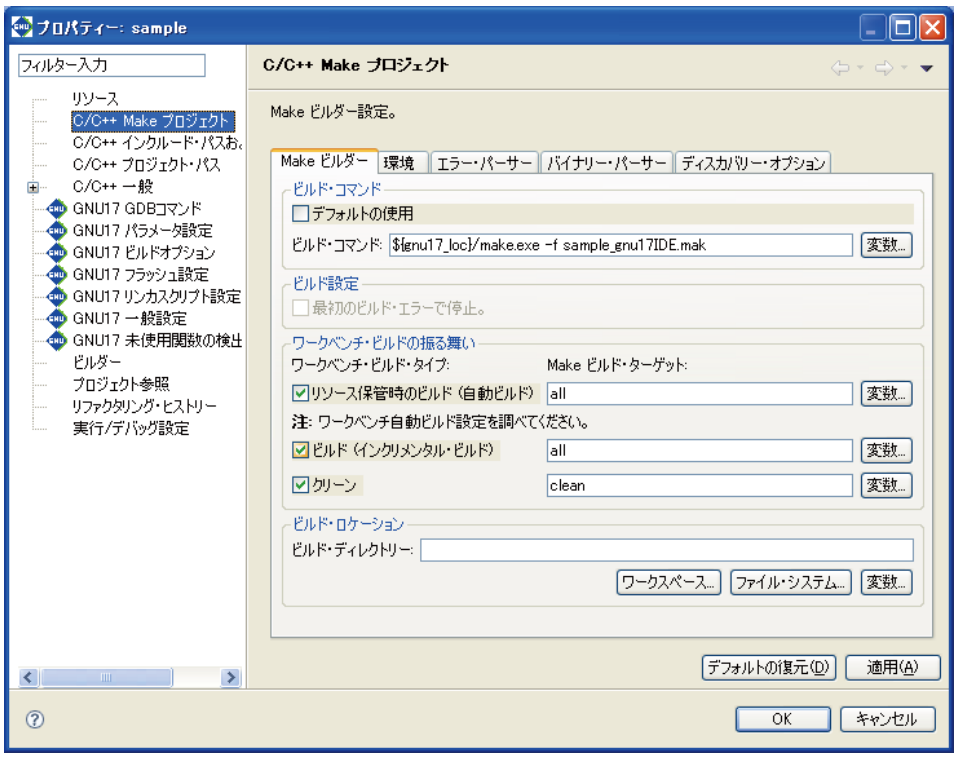
●C/C++ インクルード・パスおよびシンボル



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]

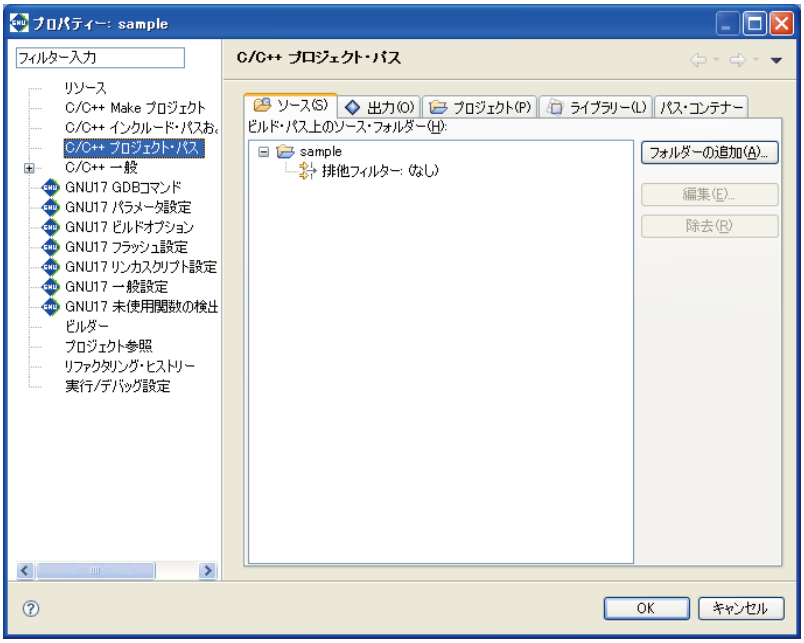
ビルド時の作業ディレクトリを指定するフィールドですが、特に指定する必要はありません。

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、"5.7.14 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



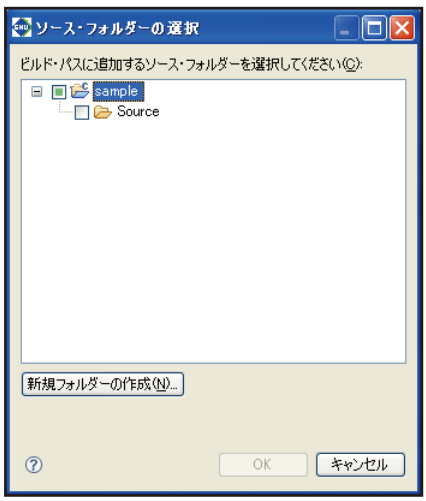
ソースフォルダを設定します。ここで設定したフォルダ内のソースファイルがビルド用のmakeファイル内に記述されます。

[ビルド・パス上のソース・フォルダー :]

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...]

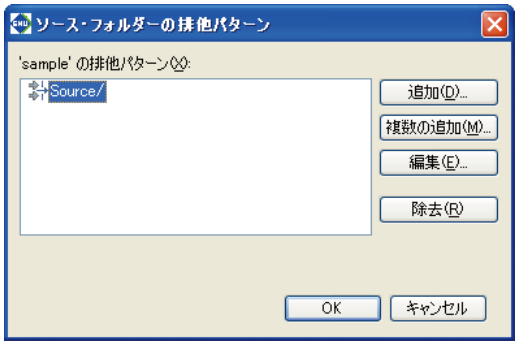
一覧にソースフォルダを追加します。



[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]

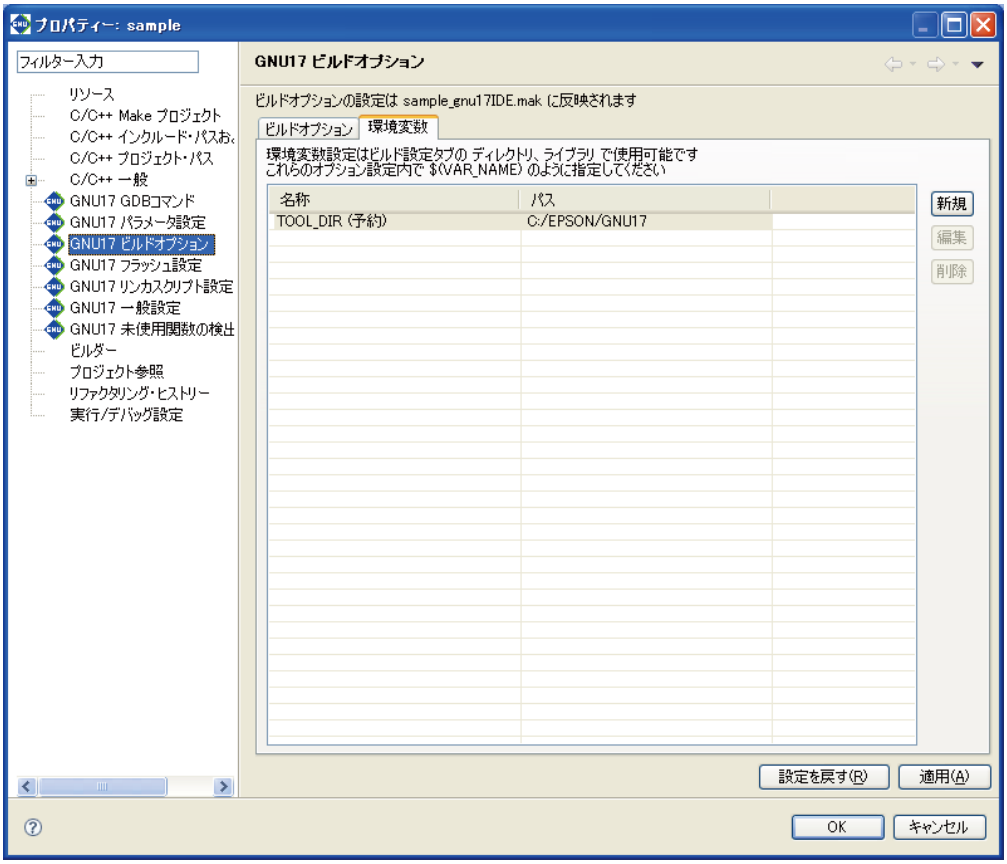
一覧から選択したファイルパターンを削除します。

[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。

●GNU17 ビルドオプション

[環境変数]タブ



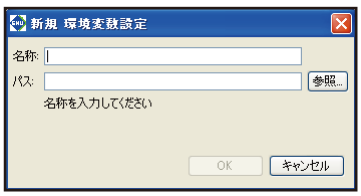
[ビルドオプション]タブの[コンパイラ]>[ディレクトリ]と[リンカ]>[ライブラリ]のパス指定に使用する環境変数を管理します。

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gnu17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]ボタンをクリックします。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]

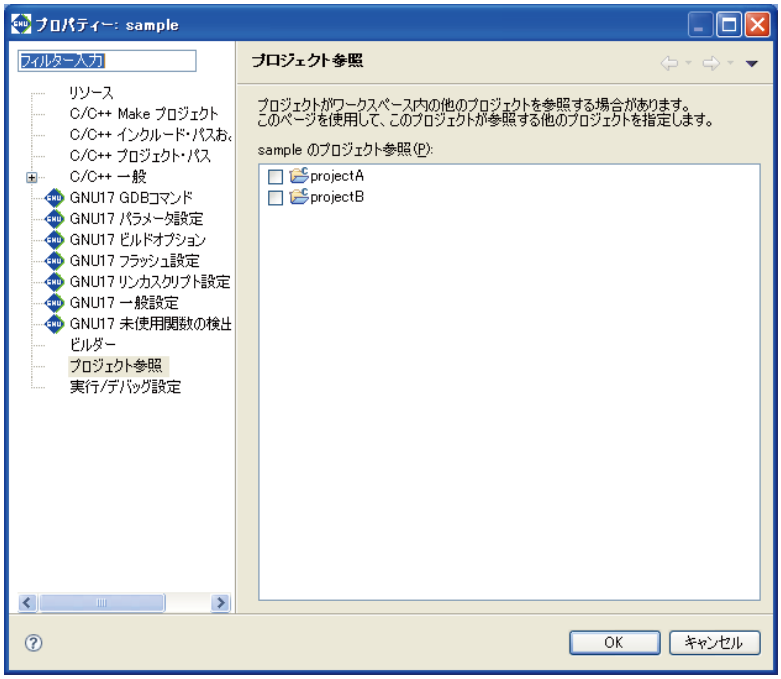
リスト内で選択した環境変数の変数名、パスを変更します。[新規]ボタンと同様のダイアログが表示されます。

[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。**

●プロジェクト参照

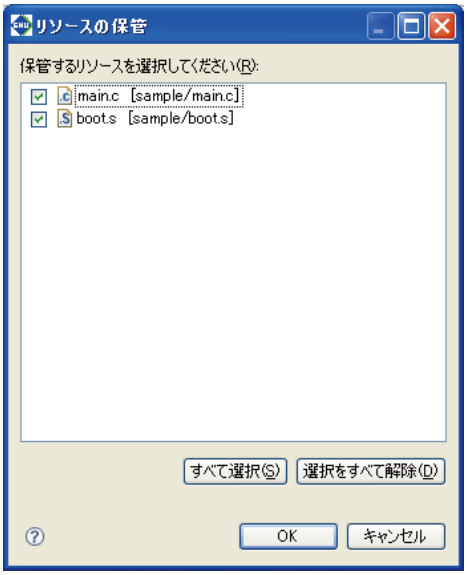


参照するプロジェクトを選択します。

[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。

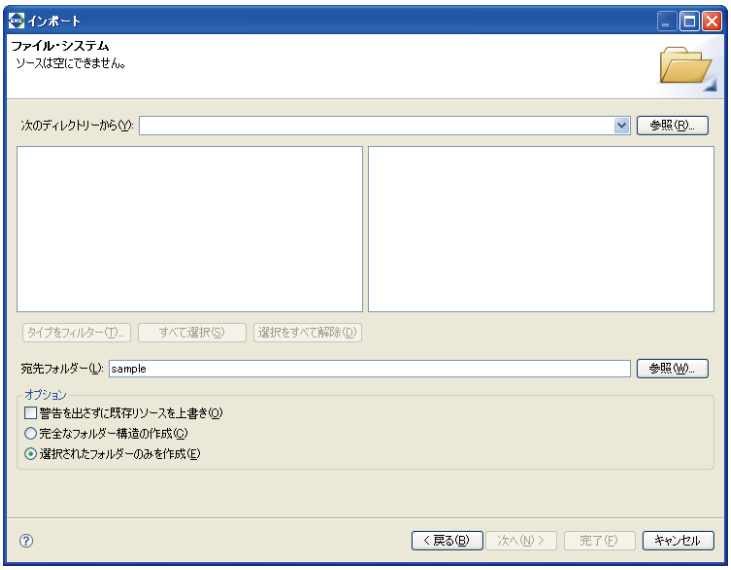
[すべて選択]
すべてのファイルのチェックボックスを選択します。

[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 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[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。

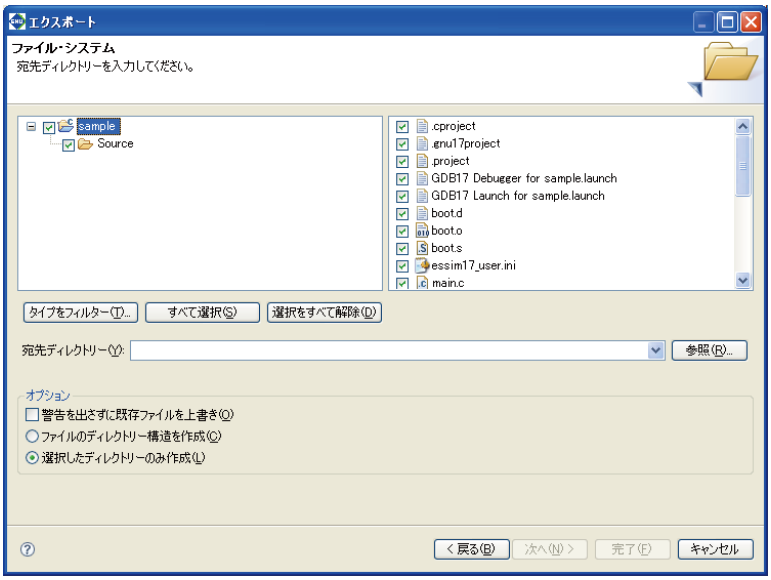
[宛先フォルダー :]
[C/C++ プロジェクト]または[ナビゲーター]ビューで選択したプロジェクト/フォルダが表示されます。他のプロジェクト/フォルダにインポートする場合は、[参照...]ボタンで選択します。

[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。

5.10.4 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[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。

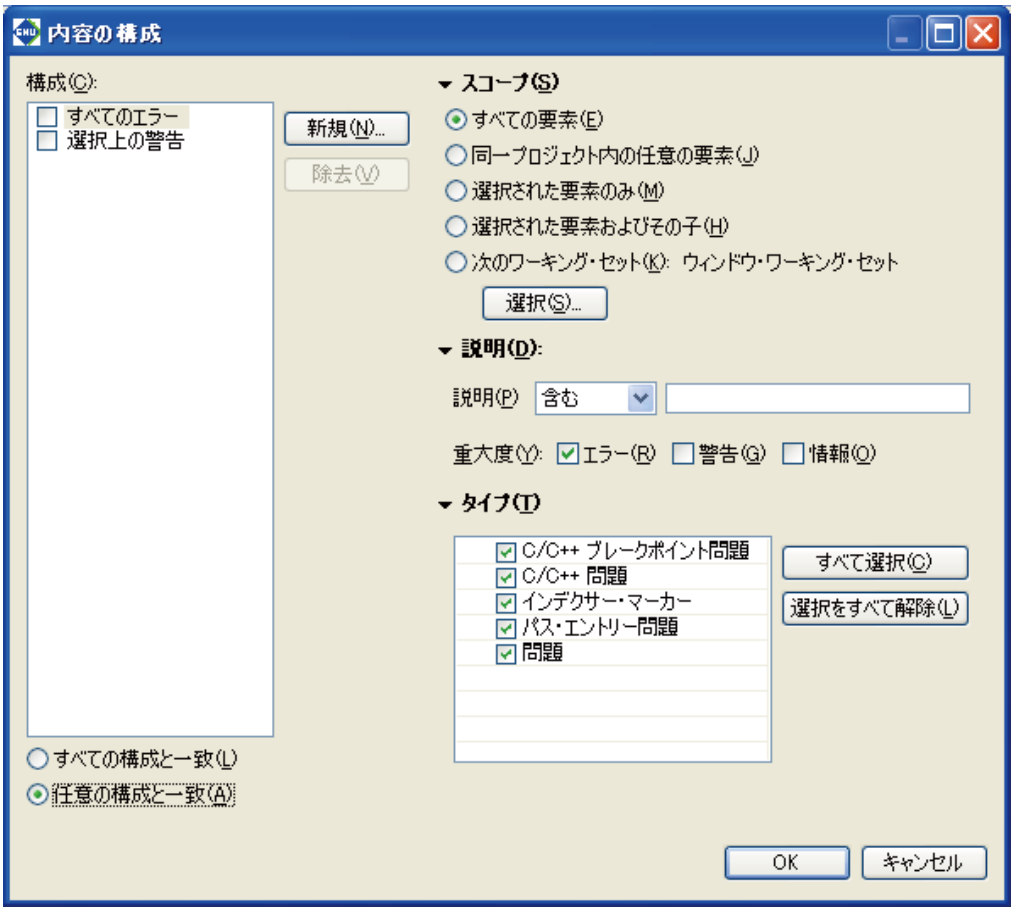
[ファイルのディレクトリー構造を作成]

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。

5.10.5 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[新規フィルターの追加]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]

[User フィルター :]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。

[説明]
含む [説明]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 [説明]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	生成プログラム
.project	IDEプロジェクトファイル	×	○	elf/psa/a
.cproject	IDEプロジェクトファイル(CDT)	×	○	elf/psa/a
.cdtproject	IDEプロジェクトファイル(CDT) GNU17v1.3.0以前に作成されたプロジェクト	×	○	elf/psa/a
.gnu17project	IDEプロジェクトファイル(GNU17)	×	○	elf/psa/a
GDB17 Debugger for *<プロジェクト名>*.launch	GDB起動設定用ファイル	×	○	elf/psa/a
*<プロジェクト名>*_gnu17IDE.cmd	GDBコマンドファイル	○ *1	○	elf/psa
¥.settings	プロジェクト設定格納ディレクトリ	×	○	elf/psa/a
¥.externalToolビルダー	プロジェクト設定格納ディレクトリ(ビルダ)	×	○	elf/psa/a
*<プロジェクト名>*_gnu17IDE.mak	メイクファイル	×	×	elf/psa/a
*<プロジェクト名>*_gnu17IDE.lds	リンカスクリプトファイル	×	×	elf/psa
*<プロジェクト名>*_gnu17IDE.par	パラメータファイル	×	×	elf/psa
<ソースファイル名>.elf	実行ファイル	×	×	elf/psa
<ソースファイル名>.map	マップファイル	×	×	elf/psa
<ソースファイル名>.dump	2passメイク用シンボルファイル	×	×	elf/psa
<ソースファイル名>.d	メイクファイル用依存関係ファイル	×	×	elf/psa/a
<ソースファイル名>.o	オブジェクトファイル	×	×	elf/psa/a
<ソースファイル名>.ext0	2パスメイク用アセンブリファイル	×	×	elf/psa/a
<ソースファイル名>.sa	実行ファイルから生成されたS3ファイル	×	×	elf/psa
<ソースファイル名>.saf	S3ファイルの開き領域を0xFF詰めしたファイル	×	×	elf/psa
<ソースファイル名>.psa	ROMデータ(PSAファイル)	×	×	elf/psa
symtable.out	ベクタチェッカ用シンボルテーブルファイル	×	×	elf/psa
raw.out	ベクタチェッカ用データファイル	×	×	elf/psa
protect.cmd	フラッシュプロテクト設定用コマンドファイル	×	×	elf/psa
*<ソースファイル名>*_ptd.psa	プロテクトされたROMデータ(PSAファイル)	×	×	elf/psa
<ソースファイル名>.a	ライブラリファイル	×	×	a

*1: プロジェクトの[プロパティー]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。

生成プログラムの"elf/psa"は、プロジェクトでアプリケーション(.elf/.psa)選択時に作成するファイルで、"a"は、ライブラリ(.a)選択時に作成するファイルを示します。

S5U1C17001C Manual

6 Cコンパイラ

6 Cコンパイラ

この章ではCコンパイラ**xgcc**の使用方法と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コンパイラの標準機能やCソースの文法については、ANSI 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コンパイラ**xgcc**はCソースファイルをコンパイルし、S1C17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準拠のGNU 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コンパイラは**xgcc.exe**、**cpp.exe**、**cc1.exe**の3つのファイルで構成されています。

xgccはFree Software Foundation, Inc.の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コンパイラの制限事項となっているものは"¥gnu17¥doc¥release_history_j.pdf"及び、"6.7 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

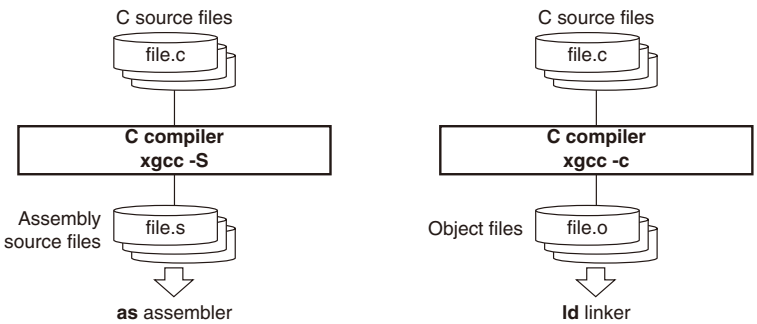


図6.2.1 フローチャート

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。

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ソースファイルを拡張子(.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コンパイラ**xgcc**がサブプログラム(**cpp**や**cc1**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。

-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参照可能です。ただし、ユーザプログラムで同名のマクロを定義することはできません。

マクロ名	内容
__c17	S1C17用にコンパイルされていることを示します。
__INT__	int型のサイズ(16)を示します。
__POINTER24	コンパイラオプション-mpointer16なしでコンパイルされていることを示します。
__POINTER16	コンパイラオプション-mpointer16付きでコンパイルされていることを示します。
__LONG_OFFSET	コンパイラオプション-mshort-offsetなしでコンパイルされていることを示します。
__SHORT_OFFSET	コンパイラオプション-mshort-offset付きコンパイルされていることを示します。

-O0,-O,-O3

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、-O3は速度のみを最適化の対象にしています。
-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。-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ことがあります。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
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により、重複した比較命令を繰り返さないようにします。

-O3
-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。
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。
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、-Osオプションはサポートされていませんので、使用しないでください。

デフォルト: コードの最適化が行われます。

-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**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。

デフォルト: デバッグ情報は出力されません。

-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, cos, exit, 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。

-mpointer16

機能: **16ビット(64KB)データ空間用コードの生成**

説明: データポインタを16ビット(データ空間は64KBに限定)としてコードを生成します。
スタティック変数のポインタを格納するRAMサイズが節約できます。 ただし、これによってスタックを節約することはできません。

デフォルト: 24ビット(16MB)空間にデータを配置可能なコードを生成します。

-mshort-offset

機能: **20ビット(1MB)空間用コードの生成**

説明: データ領域や分岐アドレスを20ビット(1MB空間)の範囲に限定してコードを生成します。
データアクセス/分岐命令の即値拡張をext命令1個に抑制し、コードサイズを削減します。
これにより、データへのアクセスと条件分岐は1MB空間に限定されます。
なお、分岐命令のcallとjprについては、ext命令1個で24ビット範囲の分岐に対応するため、1MBには限定されません。 1MBの空間内にプログラムとデータが収まるアプリケーションで、コードサイズを削減したい場合に有効です。
コンパイラは命令コードを、x拡張命令(xld、xjreqなど)ではなく、s拡張命令(sld、sjreqなど)として出力します。

デフォルト: 24ビット(16MB)空間にデータ/プログラムを配置可能なコードを生成します。

表6.3.2.1 -mpointer16と-mshort-offsetオプションの指定

オプション	データ/プログラム空間		
	24ビット(16MB)*1 データポインタは24ビットとしてコード生成	20ビット(1MB)*1 データポインタは24ビットとしてコード生成	16ビット(64KB)*2 データポインタは16ビットとしてコード生成
	全空間を使用	コードサイズを削減	コードとRAMサイズを削減
-mpointer16	指定しない(デフォルト)	指定しない(デフォルト)	指定する *3
-mshort-offset	指定しない(デフォルト)	指定する	指定する *3

*1 ANSI CおよびエミュレーションライブラリはC24ビット版(¥lib¥24bit)を使用します。

*2 ANSI CおよびエミュレーションライブラリはC16ビット版(¥lib¥16bit)を使用します。

*3 通常、-mpointer16は単独で指定せず、-mshort-offsetとペアで指定してください。

-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型の全ての値に対応する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。
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、"6.5 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。

-Wa,*<オプション>*

機能: **アセンブラオプションの指定**
説明: 指定のオプションがアセンブラに渡されます。複数のオプションを指定することができます。その場合は、`-Wa,`*<オプション>*を必要数入力してください。
デフォルト: アセンブラに渡されるオプションはありません。

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: • 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、必要なリンカのオプションを指定することができません。**
- **`-mpointer16`と`-mshort-offset`オプションは、リンクするすべてのオブジェクトについて、アセンブラの指定も含め、同じ組み合わせで指定してください。異なるオプション指定により生成したオブジェクトは、リンクできない場合があります。**

コンパイル例:

```
xgcc -B$(TOOL_DIR)/ -c -O -gstabs -fno-builtin -I$(TOOL_DIR)/include
-mpointer16 -mshort-offset main.c
```

`-c`オプションが指定されると、xgccの`-mpointer16`オプションは、アセンブラへも渡ります。

アセンブル例:

```
xgcc -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs
-Wa,-mpointer16  boot.s
```

アセンブラソース`boot.s`をプリプロセッサで処理した後、アセンブルします。

6.4 コンパイラの出力

ここでは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コンパイラ**xgcc**はCソースをコンパイルし、以下の内容を出力します。
• S1C17コア命令セットのニーモニック
• 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。

システム制御命令などは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("halt");`

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コンパイラ**xgcc**はANSI 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	2	-32768～32767
unsigned int	2	0～65535
long	4	-2147483648～2147483647
unsigned long	4	0～4294967295
pointer	4	0～16777215
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
wchar_t	2	0～65535

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
```

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

●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型および格納する変数領域に依存します。各型の変数領域における格納位置を表6.4.2.2に示します。

表6.4.2.2 型の種類と格納位置

型	グローバル変数領域	ローカル変数領域
char	1バイト境界	
short	2バイト境界	
int	2バイト境界	
long	4バイト境界	
pointer	4バイト境界	
long long	4バイト境界	
構造体	4バイト境界の型を含まない場合：2バイト境界	サイズが2バイト以下の場合：2バイト境界
	4バイト境界の型を含む場合：4バイト境界	サイズが3バイト以上の場合：4バイト境界
配列	4バイト境界の型を含まない場合：2バイト境界	要素数が1の場合：データ型の格納位置
	4バイト境界の型を含む場合：4バイト境界	要素数が2以上の場合：4バイト境界

構造体内での格納位置はデータ型に依存します。各型の構造体内での格納位置を表6.4.2.3に示します。

表6.4.2.3 型の種類と格納位置

型	格納位置
char	1バイト境界
short	2バイト境界
int	2バイト境界
long	4バイト境界
pointer	4バイト境界
long long	4バイト境界
構造体	4バイト境界の型を含まない場合：2バイト境界
	4バイト境界の型を含む場合：4バイト境界
配列	データ型の格納位置

注：構造体及び配列の直後に配置した変数の格納位置は、上記の内容と一致しない場合があります。

●構造体データ

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      sData;
        char       cArray[3];
        long       lData;
};
```

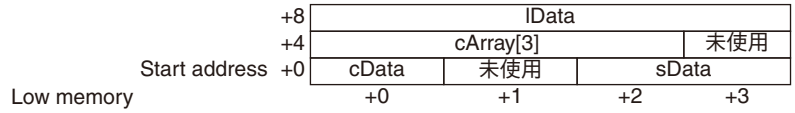


図6.4.2.1 構造体データのメモリ配置例

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。
C言語の規格上、構造体又は共用体のメンバ変数の配置方法は処理系定義で調整することが許されています。
本パッケージのCコンパイラでは、処理系定義として構造体及び共用体のサイズは、必ず偶数バイトになるように調整されます。

●ビットフィールドのアクセス

ビットフィールドをunsigned short型で定義した場合でも、8ビット以下の変数については、コンパイラの最適化処理により、以下のようにバイトサイズでアクセスされます。16ビットアクセスが必要なI/Oレジスタなどをビットフィールドとしてアクセスする場合は注意してください。

プログラム
```
struct IFtag {
     volatile union {
          volatile struct {
               unsigned short  DATA    : 1;        /* 1bit */
               unsigned short  Dummy   : 15;
          } bCTL;
          unsigned short  usCTL;
     } rOUT;
};

struct g_GAtag {
     volatile struct IFtag       g_IF;
};

volatile struct g_GAtag *pg_GAtag;
```

```
main()
{
      pg_GAtag = (struct g_GAtag *)0x8300;
      pg_GAtag->g_IF.rOUT.bCTL.DATA = 1;   (*)
      return;
}
```


生成されるコード 上記(*)部分

```
pg_GAtag->g_IF.rOUT.bCTL.DATA = 1;
```

```
ld.b    %r3,[%r2]      ;バイトアクセス
or      %r3,0x1
ld.b    [%r2],%r3      ; バイトアクセス
```

6.4.3 レジスタ使用法

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	引数渡し用レジスタ(第1ワード) 戻り値格納用レジスタ(8/16ビットデータ、ポインタ、32ビットデータの下位16ビット)
%r1	引数渡し用レジスタ(第2ワード) 戻り値格納用レジスタ(32ビットデータの上位16ビット)
%r2	引数渡し用レジスタ(第3ワード)
%r3	引数渡し用レジスタ(第4ワード)
%r4	関数呼び出し時に値を保存する必要のあるレジスタ
%r5	
%r6	
%r7	

●引数渡し用レジスタ(%r0〜%r3)

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。

%r0 ← 第1引数
%r1 ← 第2引数
%r2 ← 第3引数
%r3 ← 第4引数

32ビット(long)引数は、次のようにレジスタペアに格納します。
%r1(上位16ビット)と%r0(下位16ビット)
%r3(上位16ビット)と%r2(下位16ビット)

例:

- 第1引数がlong、第2引数がlongの場合

```
foo( long lData1, long lData2 );
```

%r0 ← lData1(下位16ビット)
%r1 ← lData1(上位16ビット)
%r2 ← lData2(下位16ビット)
%r3 ← lData2(上位16ビット)

- 第1引数がshort、第2引数がlongの場合

```
foo( short sData, long lData );
```

%r0 ← sData(16ビット)
%r1　未使用
%r2 ← lData(下位16ビット)
%r3 ← lData(上位16ビット)

- 第1引数がlong、第2引数がshort、第3引数がshortの場合

```
foo( long lData, short sData1, short sData2 );
```

%r0 ← lData(下位16ビット)
%r1 ← lData(上位16ビット)
%r2 ← sData1(16ビット)
%r3 ← sData2(16ビット)

- 第1引数がlong、第2引数がポインタ、第3引数がポインタの場合

```
foo( long lData, int *ip_Pt, char *cp_Pt );
```

%r0 ← lData(下位16ビット)
%r1 ← lData(上位16ビット)
%r2 ← ip_Pt(24ビット(REGULAR/MIDDLE MODELの時)、16ビット(SMALL MODELの時))
%r3 ← cp_Pt(24ビット(REGULAR/MIDDLE MODELの時)、16ビット(SMALL MODELの時))

64ビット(long long、double)引数は、スタックに格納して渡します。

戻り値が64ビット(long long、double)の場合、呼び出し時に戻り値領域を確保し、その先頭アドレスを%r0に入れて関数に渡します。

●戻り値格納用レジスタ(%r0, %r1)

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。

- 戻り値が8ビット/16ビットデータ、あるいはポインタ(24ビット)の場合
  %r0 ← 戻り値
  %r1　未使用
- 戻り値が32ビットデータの場合
  %r0 ← 戻り値(下位16ビット)
  %r1 ← 戻り値(上位16ビット)

●関数呼び出し時に値を保存するレジスタ(%r4〜%r7)

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。

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つまでが引数渡し用レジスタ(%r0〜%r3)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。

●構造体引数の扱い

引数が64ビット以下の構造体の場合、引数渡し用レジスタ(%r0〜%r3)を確保できるときは、構造体のメンバの値を引数渡し用レジスタ(%r0〜%r3)を介して渡します。引数渡し用レジスタ(%r0〜%r3)を確保できないときは、構造体のメンバの値をスタックを介して渡します。

引数が64ビットを超える構造体の場合は、構造体のメンバの値をスタックを介して渡します。

6.4.5 スタックフレーム

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。

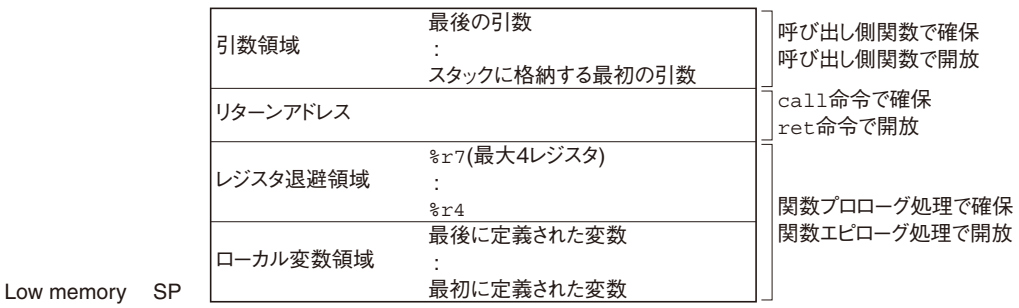


図6.4.5.1 スタックフレーム

●引数領域

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は、スタックフレーム内に領域を確保します。引数はすべて4バイト境界に配置されます。

●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です。

●レジスタ退避領域

%r4～%r7レジスタの中で、呼び出される側の関数が使用しているレジスタがあれば格納されます。
呼び出される側の関数が%r4～%r7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に格納されます。
例: {

```
        char    cData;
        short   sData;
        long    lData;
            :
}
```

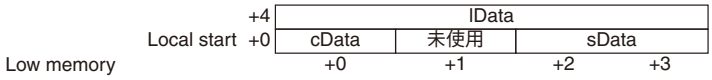


図6.4.5.2 ローカル変数の配置例

なお、最適化の結果、ソースコードの内容によっては変数が宣言された順に配置されないことがあります。
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。

6.4.6 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)については、"4.2 Cソースの文法"を参照してください。

6.4.7 コンパイラの処理系定義

C言語の規格上、構造体又は共用体のメンバ変数の配置方法は処理系定義で調整することが許されています。
本パッケージのCコンパイラでは、処理系定義として構造体及び共用体のサイズは、必ず偶数バイトになるように調整されます。

6.5 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Ver1.5.0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ソースファイル
- 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、[プロパティー]>[GNU17 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Ver1.5.0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

- Ver1.5.0より前のバージョン、もしくはVer1.5.0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。

- Ver1.5.0 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0xfff3が代入されます。

6.6 xgccの機能と注意事項について

- **xgcc**の詳細についてはGNU 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。
- 本パッケージでは__attribute__機能については、__attribute__ ((interrupt_handler));のみに対応しています。その他の__attribute__機能については動作保証対象外です。お客様の責任で使用してください。
  同様に、**xgcc**には__attribute__ ((section ("*セクション名*"))) により、関数及び変数を独自のセクションに割り当てる機能がありますが、この機能も本パッケージでは動作保証しておりません。この機能を使用される場合、セクションの割り当て自体は正常に行われますが、以下の制限があります。
  - ○ 関数を独自のセクションに割り当てた場合、この関数のデバッグ情報が取得できません。
    そのため、ステップやブレークなどによりこの関数内にプログラムカウンタがある場合、ソースエディタ上でのソースコードの表示はできません。
    ただし、逆アセンブルビュー上でのアセンブラ表示はできます。
  - ○ .bss 属性の変数(初期値を持たない変数)を独自のセクションに割り当てることはできません。
    PSAファイルを作成時に以下のエラーが発生します。
    Error: xxx.sa contains data outside of specified range
  - ○ .data 属性の変数(初期値を持つ変数)を独自のセクションに割り当てた場合、
    初期値を ROM から RAM などに転送する処理は、お客さまが作成する必要があります。

  ただし、特定の関数及び変数を任意のアドレスに配置したい場合は上記の__attribute__ ((section ("*セクション名*"))) を指定する代わりに、その関数及び変数のみを含むオブジェクトファイルを生成して独自のセクションに割り当てることによっても実現できます。
- #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。
- 本マニュアルに記載していない機能でかつ、ANSI Cに準拠していない**gcc**の拡張機能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、"¥gnu17¥doc¥release_history_j.pdf"及び、"6.7 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

6.7 既知の問題

以下にGNU17 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
	巨大なサイズの配列（数十万バイト）が定義されているコードをコンパイルすると、以下のコンパイルエラーが発生します。 cc1.exe: out of memory allocating mmmmmmmm bytes after a total of nnnnnnnn bytes
	対応方法
	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ください。
	再現コード
	unsigned char uc_array[] = { 0x00,0x01, ... }; int main() { ※ 配列のサイズは数十万バイト以上です。
	原因
	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ために発生しています。 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もあります。
No.2	不具合の内容
	char型の変数の符号拡張と、他の型との加減算処理が最適化により一つにまとめられることにより、演算結果が正しい値になりません。 この不具合は、以下の条件を全て満たした時に発生します。 ・最初に128(=0x80)以上の値を、char型より大きい型の変数に設定しておきます。 次のこのchar型より大きい型の変数を加減算した結果を、char型の変数に代入します。 最後に、このchar型の変数を加減算した結果をchar型より大きい型の変数に代入します。 このとき、不具合が発生します。 ・いずれかの代入の結果が、0〜127の範囲内である必要があります。
	対応方法
	最適化により符号拡張と加減算が一度に行われないようにするために、char型変数にvolatileをつけて宣言することで回避してください。
	再現コード
	signed int big_type_val ; int main(void) { signed char char_val ; big_type_val = 128 ; char_val = big_type_val - 1 ; ・・・① big_type_val = char_val - 1 ; ・・・② // big_type_valは126になるべきですが、-130になります。
	原因
	最適化によるエラーです。 最適化により、① & ②の２行の処理が一つの処理としてコンパイルされます。 そのため、符号拡張と演算が一度に行われ、正しい値になりません。

No.3

不具合の内容

文字列化演算子のマクロ(#)で定義した漢字の文字列とダブルクォーテーションで囲んだ漢字の文字列同士の比較がstrcmp( )で等しくなりません。
Kanji filter が有効な時にエラーになります。
※ Ver1.5.0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。

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_KFILT=xgcc_filt"を"CC_KFILT=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。

再現コード

```
#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\"" ) ) {      // この比較結果は等しくなるべきですが、そうなりません。
```

原因

Kanji filterは、コンパイル時にシフトJISコードをASCIIコード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るため、等しい文字列になりません。

```
ソースコード                  str("字")            "\"字\""
                                ↓
Kanji filter による変換       str("\x8e\x9a")      "\"\x8e\x9a\""
                                ↓
プリプロセッサ による変換     "\"\\x8e\\x9a\""     "\"\x8e\x9a\""
```

No.4

不具合の内容

以下のコンパイルエラーが発生します。

error: unable to find a register to spill in class

この不具合は、以下の条件を全て満たした時に発生する場合があります。
・REGULARモデルもしくは、MIDDLEモデルでコンパイルすること。
・関数の引数渡しの時に、ポインタ引数が%r3レジスタ経由で渡されること。
・%r3レジスタ経由で渡されたポインタ引数を、関数内で参照すること。

関数の引数渡しのレジスタ割り当てについては、コンパイラパッケージマニュアルの"6.4.3 レジスタ使用法"の"引数渡し用レジスタ(%r0〜%r3)"の箇所を参照してください。

対応方法

エラーの原因となるポインタ引数が%r3経由で引数渡しにされないようにします。
そのためには、パラメータの順序を変更することや、ダミー引数を追加するという方法で実現できます。

再現コード

```
void sub( int arg1, int arg2, int arg3, long *arg4 )
{
        static long long int num ;

        num = *arg4 ;
}
```

※ このケースでは、ポインタarg4が%r3経由で引数渡しされています。
以下のように、例えばダミー引数を追加し、arg4を%r3経由にならないようにすることで、回避できます。

```
void sub( int arg1, int arg2, int arg3, int dummy, long *arg4 )
{
        static long long int num ;

        num = *arg4 ;
}
```

原因

コンパイラの処理中に必要なレジスタを確保できなかったために発生している、コンパイラの内部エラーです。

No.5

不具合の内容

サブルーチン内でのグローバル変数への値の設定が、インライン展開により正しく行われません。
この不具合は、以下の条件を全て満たした時に発生します。
・グローバル変数のアドレスをパラメータとしてサブルーチンに渡すこと。
・サブルーチン内でパラメータであるポインタを経由して、グローバル変数に値を設定すること。
・サブルーチンがインライン展開されていること。
  インライン展開される為には以下のように、いくつかの条件を満たす必要があります。
  - ・inline 宣言をつけて –O1 以上の最適化でコンパイル、もしくは -O3 でコンパイルすること
  - ・サブルーチンの定義部が関数本体よりも前方にあること
  - ・サブルーチンのサイズが小さいこと

実際にインライン展開されているかどうかは、逆アセンブルビュー上からサブルーチンが call 命令されていないことにより確認できます。

対応方法

対応方法①) グローバル変数(以下の例では g2) を volatile をつけて宣言します。
対応方法②) サブルーチンの定義部を関数本体よりも後方に配置するなどによりインライン展開されないようにします。

再現コード

```
int g1, g2;
int i_Val;

void write_at ( int *addr, int off )
{
  addr[off] = 1000;                  // addr[off] は g2 を指しています。
}

int main( void )
{
  g2 = 12;
  write_at ( &g1, &g2 - &g1 );  // この関数がインライン展開されます。
  i_Val = g2;                        // i_Val は 1000 になるべきですが、12になります。
```

原因

最適化によるエラーです。

No.6	不具合の内容
	即値を関数ポインタにキャストしてから関数コールすると、以下のインターナルコンパイルエラーが発生します。 internal compiler error: Segmentation fault
	対応方法
	一度、即値を関数ポインタのグローバル変数に代入してから、グローバル変数を関数コールして下さい。
	再現コード

	typedef void *(*T)(void);
	
	void f(void)
	{
	((T) 10000000)(); // インターナルコンパイルエラーが発生します。
	}
	※このケースでは、以下のように関数ポインタのグローバル変数を関数コールすることで回避できます。
	typedef void *(*T)(void);
	T p_Pt; // 関数ポインタのグローバル変数
	
	void f(void)
	{
	p_Pt = (void *(*)(void))10000000;
	p_Pt();
	}

	原因
	即値から直接、関数コールする処理に不具合があるためです。
No.7	不具合の内容
	パラメータのアドレスを関数ポインタにキャストしてから関数コールすると、不正なアセンブラ命令が生成されます。
	対応方法
	一度、パラメータをグローバル変数に代入してから、グローバル変数のアドレスをキャストして関数コールして下さい。
	再現コード

	void f(int x)
	{
	(*(void (*)())&x)(); // 不正なアセンブラ命令が生成されます。
	}
	※このケースでは、以下のようにグローバル変数のアドレスをキャストして関数コール
	することで回避できます。
	int ip_Pt; // グローバル変数
	
	void f(int x)
	{
	ip_Pt = x;
	(*(void (*)())&ip_Pt)();
	}

	原因
	パラメータのアドレスから直接、関数コールする処理に不具合があるためです。

No.8

不具合の内容

while() 文の条件式内でネストされているサブルーチンとローカル変数との演算が、インライン展開により正しく行われません。
この不具合は、以下の条件を全て満たした時に発生します。
- -O1 よりも強い最適化(gnu17 でサポートしているのは –O3) でコンパイルすること
- while() 文の条件式内のサブルーチンがインライン展開されること
- 9個以上の関数でネストされていること
- while() 文の条件式内で演算されているローカル変数が、while() 文のブロックの中でも同様の演算をされていること

対応方法

while() 文の条件式内で演算されるローカル変数に volatile をつけて宣言して下さい。

再現コード

```
int f( int x )
{
  return ( x + 1 );
}

  int main( void )
{
  int a = 1;

  // while() 文の条件式内で、9個のf() 関数がネストされ、かつローカル変数 a と
  // 演算を行っています。
  while ( (f(f(f(f(f(f(f(f(f(1)))))))))) - a < 10 ){
      a--;                    // 条件式内と同じ減算の処理を行っています。
      exit ( 0 );             // exit( 0 ) が実行されるべきですが、実行されません。
  }
  abort();                    // while() 文のブロックの中に入らず、abort() が
                              // 実行されます。
```

原因

最適化によるエラーです。

S5U1C17001C Manual

7 ライブラリ

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とANSI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。

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、整数の乗除算/シフト、long long型の加減算関数を提供します。

libgccM.a エミュレーションライブラリ(乗算のコプロセッサ命令に対応)
libgcc.aと同じ関数を提供しますが、内部で乗算のコプロセッサ命令を使用しています。乗算のコプロセッサ命令に対応してある機種では、libgcc.aの代わりにこのライブラリをリンクして使用することができます。

libgccMD.a エミュレーションライブラリ(乗算/除算/剰余演算のコプロセッサ命令に対応)
libgcc.aと同じ関数を提供しますが、内部で乗算/除算/剰余演算のコプロセッサ命令を使用しています。乗算/除算/剰余演算のコプロセッサ命令に対応してある機種では、libgcc.aの代わりにこのライブラリをリンクして使用することができます。

libgccMD2.a エミュレーションライブラリ(乗算/除算/剰余演算のコプロセッサ命令に対応)
libgccMD.aと同様に、libgcc.aと同じ関数を提供しますが、対応機種が異なります。

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(7.3.3節参照)と入出力用の下位レベル関数を提供します。

ライブラリは、メモリモデル別に以下のフォルダにインストールされます。

```
¥EPSON
    ¥gnu17
        ¥lib
            ¥24bit              24ビット(16MB)メモリ空間用ライブラリ
                libc.a
                libgcc.a
                libgccM.a
                libgccMD.a
                libgccMD2.a
                libstdio.a
            ¥16bit              16ビット(64KB)メモリ空間用ライブラリ
                libc.a
                libgcc.a
                libgccM.a
                libgccMD.a
                libgccMD2.a
                libstdio.a
```

Cコンパイラおよびアセンブラの-mpointer16オプションを指定したときは、16ビット版の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. libc.a
3. libgcc.a (libgccM.a / libgccMD.a / libgccMD2.a)
4. libc.a　　（2との重複はエラーとはなりません。相互参照が可能です。）

オブジェクトファイル(もしくはライブラリ)は、リンカ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用できませんので、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 libc.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 libc.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
libc.a
```

また、リンカスクリプトファイル(.lds)の先頭のセクションには必ずオブジェクトファイルが含まれるようにして下さい。先頭のセクションにライブラリのみを記述した場合、リンクエラーが発生します。このエラーは、*.makファイル/*.ldsファイル内でライブラリを２回ずつ記述することで回避できます。なお、ライブラリを２回ずつ記述することによりプログラムサイズが増加することはありません。

例: 1. NG

*.ldsファイル

```
/* section information */
.bss 0x0000 :
{
__START_bss = . ;
c:/epson/gnu17/lib/24bit/libstdio.a(.bss)
c:/epson/gnu17/lib/24bit/libc.a(.bss)
c:/epson/gnu17/lib/24bit/libgcc.a(.bss)
c:/epson/gnu17/lib/24bit/libc.a(.bss)
}
__END_bss = . ;

.usr_bss __END_bss :
{
__START_usr_bss = . ;
test.o(.bss)
}
__END_usr_bss = . ;
```

*.ldsファイルの先頭セクションにライブラリのみを記述した場合、test.oが各ライブラリ関数を使

用しようとするとリンクエラーが発生します。

2. OK (1)

*.ldsファイル

```
/* section information */
.bss 0x0000 :
{
__START_bss = . ;
test.o(.bss)
c:/epson/gnu17/lib/24bit/libstdio.a(.bss)
c:/epson/gnu17/lib/24bit/libc.a(.bss)
c:/epson/gnu17/lib/24bit/libgcc.a(.bss)
c:/epson/gnu17/lib/24bit/libc.a(.bss)
}
__END_bss = . ;
```

*.ldsファイルの先頭セクションにオブジェクトファイル(test.o)を記述した場合、test.oは各ライブラリ関数を使用することができます。

3. OK (2)

*.ldsファイル

```
/* section information */
.bss 0x0000 :
{
__START_bss = . ;
c:/epson/gnu17/lib/24bit/libstdio.a(.bss)
c:/epson/gnu17/lib/24bit/libstdio.a(.bss)
c:/epson/gnu17/lib/24bit/libc.a(.bss)
c:/epson/gnu17/lib/24bit/libgcc.a(.bss)
c:/epson/gnu17/lib/24bit/libgcc.a(.bss)
c:/epson/gnu17/lib/24bit/libc.a(.bss)
}
 __END_bss = . ;

.usr_bss __END_bss :
{
__START_usr_bss = . ;
test.o(.bss)
}
__END_usr_bss = . ;
```

*.makファイル

```
OBJLDS= $(TOOL_DIR)/lib/24bit/libstdio.a \
   $(TOOL_DIR)/lib/24bit/libstdio.a \
   $(TOOL_DIR)/lib/24bit/libc.a \
   $(TOOL_DIR)/lib/24bit/libgcc.a \
   $(TOOL_DIR)/lib/24bit/libgcc.a \
   $(TOOL_DIR)/lib/24bit/libc.a \
```

*.ldsファイルの先頭にライブラリのみを記述した場合、各ライブラリを２回ずつ記述することにより、test.oは各ライブラリ関数を使用することができます。

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、整数の乗除算/シフト、long long型の加減算をサポートするエミュレーションライブラリ(libgcc.a / libgccM.a / libgccMD.a / libgccMD2.a)が含まれています。libgcc.aはコプロセッサ命令を使用しないライブラリであり、libgccM.aは乗算のコプロセッサ命令を、libgccMD.aおよびlibgccMD2.aは乗算/除算/剰余演算のコプロセッサ命令を使用しているライブラリです。これらのライブラリは、どれも同一の関数を提供しています。Cコンパイラ**xgcc**は、浮動小数点演算、long long演算、整数演算が行われると、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/スタック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(libgccM.a / libgccMD.a / libgccMD2.a)とlibc.aを指定してください。

ライブラリで使用しているレジスタ

- %r0から%r7を使用しています。
- %r4から%r7は関数の開始時にスタックに保存、終了時に復帰して保護されます。

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	x ← a + b
	__subdf3	減算	x ← a - b
	__muldf3	乗算	x ← a * b
	__divdf3	除算	x ← a / b
	__negdf2	符号反転	x ← -a
単精度浮動小数点演算	__addsf3	加算	x ← a + b
	__subsf3	減算	x ← a - b
	__mulsf3	乗算	x ← a * b
	__divsf3	除算	x ← a / b
	__negsf2	符号反転	x ← -a
型変換	__fixunsdfsi	double → unsigned long	x ← a
	__fixdfsi	double → long	x ← a
	__floatsidf	long → double	x ← a
	__fixunssfsi	float → unsigned long	x ← a
	__fixsfsi	float → long	x ← a
	__floatsisf	long → float	x ← a
	__truncdfsf2	double → float	x ← a
	__extendsfdf2	float → double	x ← a
倍精度浮動小数点比較	__fcmpd	double型比較	PSR変更 ← a - b
	__eqdf2	double型比較 (a=b)	PSR変更 ← a - b, x ← 0 | 1 *
	__nedf2	double型比較 (a≠b)	PSR変更 ← a - b, x ← 1 | 0 *
	__gtdf2	double型比較 (a>b)	PSR変更 ← a - b, x ← 1 | 0 *
	__gedf2	double型比較 (a≥b)	PSR変更 ← a - b, x ← 0 | -1 *
	__ltdf2	double型比較 (a<b)	PSR変更 ← a - b, x ← -1 | 0 *
	__ledf2	double型比較 (a≤b)	PSR変更 ← a - b, x ← 0 | 1 *
単精度浮動小数点比較	__fcmps	float型比較	PSR変更 ← a - b
	__eqsf2	float型比較 (a=b)	PSR変更 ← a - b, x ← 0 | 1 *
	__nesf2	float型比較 (a≠b)	PSR変更 ← a - b, x ← 1 | 0 *
	__gtsf2	float型比較 (a>b)	PSR変更 ← a - b, x ← 1 | 0 *
	__gesf2	float型比較 (a≥b)	PSR変更 ← a - b, x ← 0 | -1 *
	__ltsf2	float型比較 (a<b)	PSR変更 ← a - b, x ← -1 | 0 *
	__lesf2	float型比較 (a≤b)	PSR変更 ← a - b, x ← 0 | 1 *

＊条件成立の場合x=左側の値、条件不成立の場合x=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*(a - b)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。

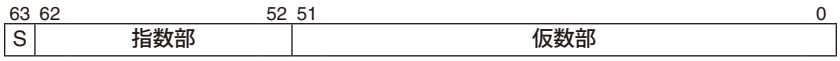
比較結果	C	V	Z	N
op1 > op2	0	0	0	0
op1 = op2	0	0	1	0
op1 < op2	1	0	0	1

●浮動小数点フォーマット

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。



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)

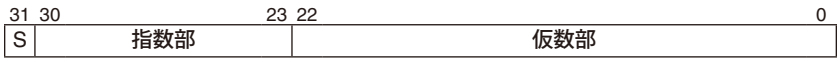
浮動小数点演算の結果は`%r0`が示すアドレスを先頭とする64ビット領域に格納されます。
`double`型の有効範囲、浮動小数点表記と内部データの対応は次のとおりです。

```
+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。



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)

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`%r1`および`%r0`レジスタに格納されます。

`%r1`レジスタ: 符号ビット、指数部および仮数部の上位7ビット(22:16)
`%r0`レジスタ: 仮数部の下位16ビット(15:0)

`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 浮動小数点数処理の処理系定義

以下の処理はC言語の規格上、処理系定義となっており本パッケージのエミューレーションライブラリでは次のように処理しています。

浮動小数点丸め方法

整数型から浮動小数点型への型変換や、浮動小数点型から別の浮動小数点型への型変換、浮動小数点型の演算時において、目的の値が目的の型で表わしうる二つの隣接した値の中間にある場合、どちらの値に丸めるかは処理系定義となります。
本パッケージでは、偶数となるように丸めています。
つまり、丸め前の値のLSB が0 の場合は何も処理せず、LSB が1 の場合は +1 して切り上げます。

浮動小数点型から整数型への変換

浮動小数点型から整数型へ変換する際、小数部分は切り捨てとなります。
小数部分切り捨て後、元の値が目的の型で表現できない場合の挙動は処理系定義となります。

- 単精度/倍精度浮動小数点型から signed / unsigned long への変換
  元の値が+NaNのとき → 目的の型がsignedであれば`0x0`、unsignedであれば`0x80000000`とします。
  元の値が-NaNのとき → `0x0`とします。
  元の値が大きすぎるとき → 目的の型で表現可能な最大値とします。
  元の値が小さすぎるとき → `0x80000000`とします。
- 単精度/倍精度浮動小数点型からsigned/unsigned long longへの変換
  元の値が+NaNのとき → `0x80000000 80000000`とします。
  元の値が-NaNのとき → 目的の型がsignedであれば`0x7fffffff 80000000`、unsignedであれば`0x0`とします。
  元の値が大きすぎるとき → `0xffffffff ffffffff`とします。
  元の値が小さすぎるとき → 目的の型がsignedであれば`0x1`、unsignedであれば`0x0`とします。

浮動小数点型から浮動小数点型への変換

浮動小数点型から別の浮動小数点型へ変換する際、元の値が目的の型で表現できない場合の挙動は処理系定義となります。

- 倍精度浮動小数点型から単精度浮動小数点への変換
  元の値が+NaNのとき → 倍精度浮動小数点の仮数部を2ビット左シフトし、上位32 ビットを取得し、切り捨てた仮数部下位32ビットが`0x0`でなければさらに取得した仮数部32ビットのLSBを1とし、その値と`0x7f900000`を論理和したもの(+NaN)とします。
  元の値が-NaNのとき → 倍精度浮動小数点の仮数部を2ビット左シフトし、上位32ビットを取得し、切り捨てた仮数部下位32ビットが`0x0`でなければさらに取得した仮数部32ビットのLSBを1とし、その値と`0xff900000`を論理和したもの(-NaN)とします。
  元の値が大きすぎるとき → `0x7f800000`(+∞)とします。
  元の値が小さすぎるとき → `0xff800000`(-∞)とします。
  元の値が0に近すぎるとき(0より大) → `0x00000000`(+0)とします。
  元の値が0に近すぎるとき(0より小) → `0x80000000`(-0)とします。
- 単精度浮動小数点型から倍精度浮動小数点への変換
  元の値が+NaN のとき → 単精度浮動小数点の仮数部を2ビット右シフトし、その値と`0x7ff80000 00000000`を論理和したものとします。
  元の値が-NaN のとき → 単精度浮動小数点の仮数部を2ビット右シフトし、その値と`0xfff80000 00000000`を論理和したものとします。

7.2.4 整数演算関数

表7.2.4.1に整数演算関数を示します。

表7.2.4.1 整数演算/関数一覧

分類	関数名	機能	
整数演算	__divsi3	符号付き32ビット整数除算	x ← a / b
	__modsi3	符号付き32ビット整数剰余演算	x ← a % b
	__udivsi3	符号なし32ビット整数除算	x ← a / b
	__umodsi3	符号なし32ビット整数剰余演算	x ← a % b
	__mulsi3	32ビット乗算	x ← a * b
	__divhi3	符号付き16ビット整数除算	x ← a / b
	__modhi3	符号付き16ビット整数剰余演算	x ← a % b
	__udivhi3	符号なし16ビット整数除算	x ← a / b
	__umodhi3	符号なし16ビット整数剰余演算	x ← a % b
	__mulhi3	16ビット乗算	x ← a * b
整数シフト	__ashlsi3	32ビット算術左シフト	x ← a << bビット
	__ashrsi3	32ビット算術右シフト	x ← a >> bビット
	__lshrsi3	32ビット論理右シフト	x ← a >> bビット
	__ashlhi3	16ビット算術左シフト	x ← a << bビット
	__ashrhi3	16ビット算術右シフト	x ← a >> bビット
	__lshrhi3	16ビット論理右シフト	x ← a >> bビット
整数比較	__cmpsi2	比較(long)	x ← 2 | 1 | 0 *1
	__ucmpsi2	比較(unsigned long)	x ← 2 | 1 | 0 *1

7.2.5 long long型演算関数

表7.2.5.1にlong long型演算関数を示します。

表7.2.5.1 long long型演算関数一覧

分類	関数名	機能	
long long型演算	__muldi3	符号付き64ビット乗算	x ← a * b
	__divdi3	符号付き64ビット除算	x ← a / b
	__udivdi3	符号なし64ビット除算	x ← a / b
	__moddi3	符号付き64ビット剰余演算	x ← a % b
	__umoddi3	符号なし64ビット剰余演算	x ← a % b
	__negdi2	符号反転	x ← -a
long long型シフト	__lshrdi3	64ビット論理右シフト	x ← a >> bビット
	__ashldi3	64ビット算術左シフト	x ← a << bビット
	__ashrdi3	64ビット算術右シフト	x ← a >> bビット
型変換	__fixunsdfdi	double → unsigned long long	x ← a
	__fixdfdi	double → long long	x ← a
	__floatdidf	long long → double	x ← a
	__fixunssfdi	float → unsigned long long	x ← a
	__fixsfdi	float → long long	x ← a
	__floatdisf	long long → float	x ← a
long long型比較	__cmpdi2	比較(long long)	x ← 2 | 1 | 0 *1
	__ucmpdi2	比較(unsigned long long)	x ← 2 | 1 | 0 *1

*1 整数比較関数およびlong long比較関数は*op1 - op2*の結果により、以下の値を返します。
op1 > *op2* → 2
op1 = *op2* → 1
op1 < *op2* → 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

7.2.6 コプロセッサ命令対応

S1C17コアはコプロセッサ命令をサポートしています。
コプロセッサ命令に対応したライブラリを使用する場合は、以下のようにベクタテーブルのNo.3に"emu_copro_process" 関数を配置してください。

例： ベクタテーブルの指定方法(vector.c)

```
extern void emu_copro_process(void);
func *const vector[] = {
        VECTOR(boot),                   // 0
        VECTOR(unalign),                // 1
        VECTOR(dummy),                  // 2
        VECTOR(emu_copro_process)       // 3
};
```

また、コプロセッサ命令には対応してある機種と対応してない機種がありますので、注意してください。
IDE からプロジェクトを作成する場合は、コプロセッサ命令に対応した機種を選択した時のみ、コプロセッサ命令に対応したライブラリを指定できるようになっています。
libgccM.aは、乗算のコプロセッサ命令に対応したライブラリです。emu_copro_process関数を呼び出す場合には、10 ～ 20サイクル程度の割り込み禁止区間があります。
libgccMD.aおよびlibgccMD2.aは、乗算/除算/剰余演算のコプロセッサ命令に対応したライブラリです。emu_copro_process関数を呼び出す場合には、15 ～ 40サイクル程度の割り込み禁止区間があります。
割り込み禁止区間の詳細は機種により異なりますので、必要に応じて、お使いの機種でemu_copro_processを動作させてご確認ください。

表7.2.6.1にエミュレーションライブラリ内でコプロセッサ命令を使用している関数を示します。コプロセッサ命令を使用する場合、各関数はemu_copro_processを呼び出しています。

表7.2.6.1 エミュレーションライブラリ内でコプロセッサ命令を使用している関数

関数名	機能	libgcc.a	libgccM.a	libgccMD.a	libgccMD2.a
__mulhi3	16ビット乗算	-	✓	✓	✓
__mulsi3	32ビット乗算	-	✓	✓	✓
__divhi3	符号付き16ビット除算	-	-	✓	✓
__modhi3	符号付き16ビット剰余演算	-	-	✓	✓
__udivhi3	符号なし16ビット除算	-	-	✓	✓
__umodhi3	符号なし16ビット剰余演算	-	-	✓	✓
__divsi3	符号付き32ビット除算	-	-	-	✓
__modsi3	符号付き32ビット剰余演算	-	-	-	✓
__udivsi3	符号なし32ビット除算	-	-	-	✓
__umodsi3	符号なし32ビット剰余演算	-	-	-	✓

7.3 ANSIライブラリ

7.3.1 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、一部のANSIライブラリ関数は本パッケージで対応していないため、ANSIライブラリに含まれていません。“7.3.2 ANSIライブラリ関数一覧”に記載されていないANSIライブラリ関数を使用する場合は、お客さまの責任で関数の実装及び、プロトタイプ宣言を行ってください。
なお、本パッケージで未対応の ANSIライブラリ関数についても、ヘッダファイル内でプロトタイプ宣言のみされているものもありますので、この場合はプロトタイプ宣言をする代わりに、該当するヘッダファイルをインクルードした上で関数の実装を行ってください。
プロトタイプ宣言のみされているANSI ライブラリ関数については“4.2.2 ライブラリ関数とヘッダファイル”の表を参照してください。
ANSIライブラリファイルは`libc.a`で、メモリモデル別に`¥gnu17¥lib¥24bit`、`¥gnu17¥lib¥16bit`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`libgcc.a`(`libgccM.a` / `libgccMD.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`

ライブラリで使用しているレジスタ

- `%r0`から`%r7`を使用しています。
- `%r4`から`%r7`は関数の開始時にスタックに保存、終了時に復帰して保護されます。

7.3.2 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
△ 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7.3.2.1 入出力関数一覧

ヘッダファイル: **`stdio.h`**

関数	機能	リエントラント	備考
`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
`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

関数	機能	リエントラント	備考
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
void **perror**(const char *s);	stdoutへのエラー情報出力	×	グローバル変数stdout, _iobを参照、errnoを変更、read関数をコール
int **fscanf**(FILE *stream, const char *format, ...);	stdinからの書式指定付き入力	×	グローバル変数stdout, _iobを参照、errnoを変更、read関数をコール
int **scanf**(const char *format, ...);	stdinからの書式指定付き入力	×	グローバル変数stdout, _iobを参照、errnoを変更、read関数をコール
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errnoを変更
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	△	グローバル変数stdout, stderr, _iobを参照、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	○	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	△	グローバル変数stdout, stderr, _iobを参照、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	○	write関数をコール

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7.3.4節を参照してください。

●ユーティリティ関数

以下、libc.aに含まれるユーティリティ関数の一覧を示します。

表7.3.2.2 ユーティリティ関数一覧

ヘッダファイル: **stdlib.h**

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
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
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

●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7.3.2.3 非局所分岐関数一覧

ヘッダファイル: **setjmp.h**

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7.3.2.4 日付と時刻関数一覧

ヘッダファイル: **time.h**

関数	機能	リエントラント	備考
struct tm ***gmtime**(const time_t *t);	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
time_t **mktime**(struct tm *tmptr);	標準時間をカレンダー時間に変換	×	ロケール情報と夏時間の設定は反映されません。
time_t **time**(time_t *tptr);	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7.3.2.5 数学関数一覧

ヘッダファイル: **math.h**, **errno.h**, **float.h**, **limits.h**

関数	機能	リエントラント	備考
double **fabs**(double x);	絶対値を返す(double型)	○	
double **ceil**(double x);	double型小数部の切り上げ	×	グローバル変数errnoを変更
double **floor**(double x);	double型小数部の切り捨て	×	グローバル変数errnoを変更
double **fmod**(double x, double y);	double型の剰余計算	×	グローバル変数errnoを変更
double **exp**(double x);	指数計算(e^x)	×	グローバル変数errnoを変更
double **log**(double x);	自然対数の計算	×	グローバル変数errnoを変更
double **log10**(double x);	常用対数の計算	×	グローバル変数errnoを変更
double **frexp**(double x, int *nptr);	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
double **ldexp**(double x, int n);	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
double **modf**(double x, double *nptr);	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
double **pow**(double x, double y);	x^yを計算	×	グローバル変数errnoを変更
double **sqrt**(double x);	平方根を計算	×	グローバル変数errnoを変更
double **sin**(double x);	正弦(サイン)計算	×	グローバル変数errnoを変更
double **cos**(double x);	余弦(コサイン)計算	×	グローバル変数errnoを変更
double **tan**(double x);	正接(タンジェント)計算	×	グローバル変数errnoを変更
double **asin**(double x);	逆正弦(アークサイン)計算	×	グローバル変数errnoを変更
double **acos**(double x);	逆余弦(アークコサイン)計算	×	グローバル変数errnoを変更
double **atan**(double x);	逆正接(アークタンジェント)計算	×	
double **atan2**(double y, double x);	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
double **sinh**(double x);	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
double **cosh**(double x);	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
double **tanh**(double x);	双曲線正接(ハイパボリックタンジェント)計算	×	

●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7.3.2.6 文字関数一覧

ヘッダファイル: **string.h**

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 **strspn**(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 **strcspn**(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

●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7.3.2.7 文字種判定/変換関数一覧

ヘッダファイル: **ctype.h**

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7.3.2.8 可変引数マクロ一覧

ヘッダファイル: **stdarg.h**

マクロ	機能
void **va_start**(va_list ap, type lastarg);	可変個引数集の初期化
type **va_arg**(va_list ap, type);	実引数の値を返す
void **va_end**(va_list ap);	可変個引数関数からの正常復帰

7.3.3 グローバル変数の宣言と初期化

ANSIライブラリ関数は表7.3.3.1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。
libstdio.aを使用したANSIライブラリ初期化の方法については、¥gnu17¥sample¥S1C17common¥simulator¥simulatedIOのサンプルを参照してください。

表7.3.3.1 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

FILE _iob[FOPEN_MAX +1];
FOPEN_MAX=3, stdio.h**にて定義**
標準入出力ストリーム用
ファイル構造体データ
	_iob[*N*]._flg = _UGETN;
	_iob[*N*]._buf = 0;
	_iob[*N*]._fd = *N*;
	(*N*=0~2)
	_iob[0]: stdin**用入力データ**
	_iob[1]: stdout**用出力データ**
	_iob[2]: stderr**用出力データ**
		stdio.h, **smcvals.h**
		fgets, fread, fscanf, getc, getchar, gets, scanf, ungetc, perror, fprintf, fputs, fwrite, printf, putc, putchar, puts, vfprintf, vprintf

FILE *stdin; **標準入出力ファイル構造体** **データ**_iob[0]**へのポインタ**	stdin = &_iob[0];	**stdio.h** fgets, fread, fscanf, getc, getchar, gets, scanf, ungetc
FILE *stdout; **標準入出力ファイル構造体** **データ**_iob[1]**へのポインタ**	stdout = &_iob[1];	**stdio.h** fprintf, fputs, fwrite, printf, putc, putchar, puts, vfprintf, vprintf
FILE *stderr; **標準入出力ファイル構造体** **データ**_iob[2]**へのポインタ**	stderr = &_iob[2];	**stdio.h** fprintf, fputs, fwrite, printf, perror, putc, putchar, puts, vfprintf, vprintf
int errno; **エラー番号を格納する変数**	errno = 0;	**errno.h** perror, fprintf, printf, sprintf, vprintf, vfprintf, fscanf, scanf, sscanf atof, atoi, calloc, div, ldiv, malloc, realloc, strtod, strtol, strtoul acos, asin, atan2, ceil, cos, cosh, exp, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan
unsigned int seed; **乱数の種を格納する変数**	seed = 1;	**stdlib.h** rand, srand
time_t gm_sec; **タイマー関数の70年1月1日** **0時0分0秒からの秒数**	gm_sec = -1;	**time.h** time

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ごとに、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ル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。)

7.3.4 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。
libstdio.aの使用方法については、¥gnu17¥sample¥S1C17common¥simulator¥simulatedIOのサンプルを参照してください。

●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(putcc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。
なお、read関数のみ、もしくはwrite関数のみを使用する場合でも、libstdio.aライブラリに含まれている_init_sys関数、read関数、write関数がリンクされます。

●_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

このページはブランクです。

S5U1C17001C Manual

8 アセンブラ

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

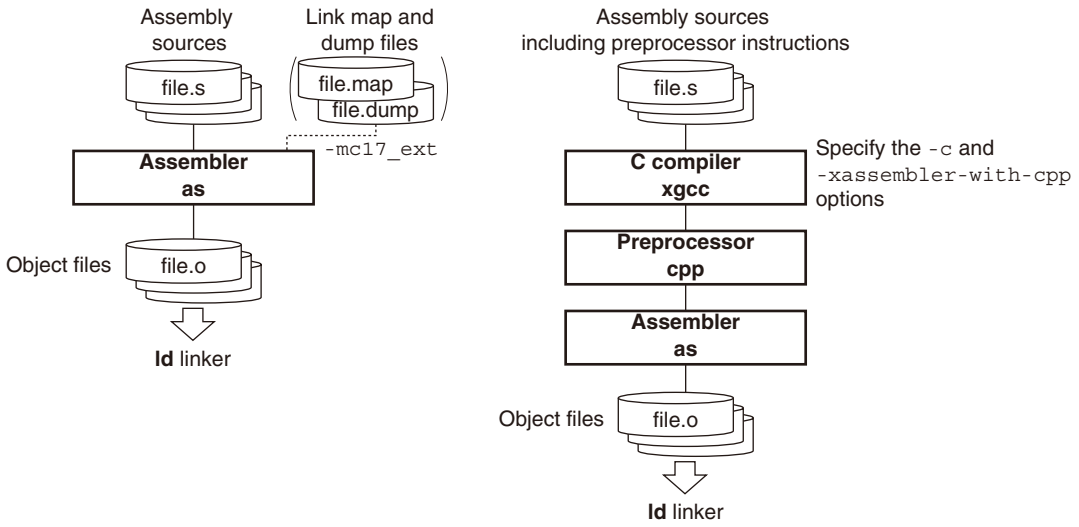


図8.2.1 フローチャート

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）
<ファイル名.ext0>
内容: ソースプログラムを記述したファイルです。通常、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。
IDE上でビルドする場合、<ファイル名.ext0>が入力ファイルとなります。

●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: オブジェクトファイルの配置を示すマップ情報ファイルで、リンカが出力します。
-mc17_extオプション指定時に、グローバルシンボル参照部の最適化に使用されます。

●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。-mc17_extオプション指定時に、ローカル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>*は必ず入力ファイルと同じ名称として、-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: -adhl　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S1C17専用のオプションも用意されています。

-mpointer16

機能:　**16ビットポインタモードの指定**
説明:　16ビットポインタモード(64KBメモリモデル)用のオブジェクトファイルを出力します。
実際には、16ビットポインタモードを判別するためのフラグをセットするのみで、生成されるオブジェクトコードには影響を与えません。
デフォルト: 24ビットポインタモード(16MBまたは1MBメモリモデル)用のオブジェクトファイルを出力します。

-mc17_ext *<ダンプファイル名> <リンクマップファイル名>*

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8.6節および8.7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 -o test.o -adhl 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）

```
        xcall   SYMBOL        ...シンボルを外部参照
                :
        xld.a   %r1,VAR1      ...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

.rodata　　定数を配置する.rodataセクションを宣言
.data　　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
.section .data
```

●機能

(1).section .rodata

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。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　　.rodataセクションを設定します。

(2).section .data

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。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　　.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, .int
4バイト: .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 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 *<4バイトデータ>*[,*<4バイトデータ>* ... ,*<4バイトデータ>*]
.short *<2バイトデータ>*[,*<2バイトデータ>* ... ,*<2バイトデータ>*]
.byte *<1バイトデータ>*[,*<1バイトデータ>* ... ,*<1バイトデータ>*]
.ascii "*<文字列>*"[,"*<文字列>*" ... ,"*<文字列>*"]
.space *<バイト長>*[,*<1バイトデータ>*]

<4バイトデータ>	0x0～0xffffffff
<2バイトデータ>	0x0～0xffff
<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

4バイトデータ、2バイトデータ、1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 .short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*<1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。

*<1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)
```

8.5.5 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
```

8.5.6 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2^n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。

8.5.7 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。

8.5.8 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(24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 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

8.6.1 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
sadd %rd,*imm16*	%rd ← %rd+*imm16*	(1)
sadc %rd,*imm16*	%rd ← %rd+*imm16*+C	(1)
sadd.a %rd,*imm20*	%rd ← %rd+*imm20*	(2)
sadd.a %sp,*imm20*	%sp ← %sp+*imm20*	(2)
ssub %rd,*imm16*	%rd ← %rd-*imm16*	(1)
ssbc %rd,*imm16*	%rd ← %rd-*imm16*-C	(1)
ssub.a %rd,*imm20*	%rd ← %rd-*imm20*	(2)
ssub.a %sp,*imm20*	%sp ← %sp-*imm20*	(2)
xadd %rd,*imm16*	%rd ← %rd+*imm16*	(1)
xadc %rd,*imm16*	%rd ← %rd+*imm16*+C	(1)
xadd.a %rd,*imm24*	%rd ← %rd+*imm24*	(3)
xadd.a %sp,*imm24*	%sp ← %sp+*imm24*	(3)
xsub %rd,*imm16*	%rd ← %rd-*imm16*	(1)
xsbc %rd,*imm16*	%rd ← %rd-*imm16*-C	(1)
xsub.a %rd,*imm24*	%rd ← %rd-*imm24*	(3)
xsub.a %sp,*imm24*	%sp ← %sp-*imm24*	(3)

これらの拡張命令により、加減算で16ビット/20ビット/24ビット即値が直接指定可能となります。
拡張命令に、条件演算オプション(/c、/nc)は指定できません。

●展開後の基本命令

sadd, **xadd**	add命令に展開
sadc, **xadc**	adc命令に展開
sadd.a, **xadd.a**	add.a命令に展開
ssub, **xsub**	sub命令に展開
ssbc, **xsbc**	sbc命令に展開
ssub.a, **xsub.a**	sub.a命令に展開

●展開形式

(1) sOP %rd,*imm16* / xOP %rd,*imm16* (*OP* = add, adc, sub, sbc)

例: xadd %rd,*imm16*

imm16 ≤ 0x7f	0x7f < *imm16*
add %rd,*imm16*(6:0)	ext *imm16*(15:7) add %rd,*imm16*(6:0)

(2) sOP.a %rd,*imm20* / sOP.a %sp,*imm20* (*OP* = add, sub)

例: sadd.a %rd,*imm20*

imm20 ≤ 0x7f	0x7f < *imm20*
add.a %rd,*imm20*(6:0)	ext *imm20*(19:7) add.a %rd,*imm20*(6:0)

(3) xOP.a %rd,*imm24* / xOP.a %sp,*imm24* (*OP* = add, sub)

例: xadd.a %rd,*imm24*

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
add.a %rd,*imm24*(6:0)	ext *imm24*(19:7) add.a %rd,*imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) add.a %rd,*imm24*(6:0)

8.6.2 比較命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
scmp %rd,*imm16*	%rd-*imm16* (PSRのC, V, Z, Nフラグを変更)	(1)
scmc %rd,*imm16*	%rd-*imm16*-C (PSRのC, V, Z, Nフラグを変更)	(1)
scmp.a %rd,*imm20*	%rd-*imm20* (PSRのC, V, Z, Nフラグを変更)	(2)
xcmp %rd,*imm16*	%rd-*imm16* (PSRのC, V, Z, Nフラグを変更)	(1)
xcmc %rd,*imm16*	%rd-*imm16*-C (PSRのC, V, Z, Nフラグを変更)	(1)
xcmp.a %rd,*imm24*	%rd-*imm24* (PSRのC, V, Z, Nフラグを変更)	(3)

これらの拡張命令により、汎用レジスタと16ビット/20ビット/24ビット即値との比較が行えます。
拡張命令に、条件演算オプション(/c、/nc)は指定できません。

●展開後の基本命令

scmp, **xcmp** cmp命令に展開
scmc, **xcmc** cmc命令に展開
scmp.a, **xcmp.a** cmp.a命令に展開

●展開形式

(1) *sOP* %rd,*imm16* / *xOP* %rd,*imm16* (*OP* = cmp, cmc)

例: xcmp %rd,*imm16*

imm16 ≤ 0x7f	0x7f < *imm16*
cmp %rd,*imm16*(6:0)	ext *imm16*(15:7) cmp %rd,*imm16*(6:0)

(2) scmp.a %rd,*imm20*

imm20 ≤ 0x7f	0x7f < *imm20*
cmp.a %rd,*imm20*(6:0)	ext *imm20*(19:7) cmp.a %rd,*imm20*(6:0)

(3) xcmp.a %rd,*imm24*

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
cmp.a %rd,*imm24*(6:0)	ext *imm24*(19:7) cmp.a %rd,*imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) cmp.a %rd,*imm24*(6:0)

8.6.3 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
sand %rd,*imm16*	%rd ← %rd&*imm16*	(1)
soor %rd,*imm16*	%rd ← %rd|*imm16*	(1)
sxor %rd,*imm16*	%rd ← %rd^*imm16*	(1)
snot %rd,*imm16*	%rd ← !*imm16*	(1)
xand %rd,*imm16*	%rd ← %rd&*imm16*	(1)
xoor %rd,*imm16*	%rd ← %rd|*imm16*	(1)
xxor %rd,*imm16*	%rd ← %rd^*imm16*	(1)
xnot %rd,*imm16*	%rd ← !*imm16*	(1)

これらの拡張命令により、論理演算で16ビット即値が直接指定可能となります。
拡張命令に、条件演算オプション(/c、/nc)は指定できません。

●展開後の基本命令

sand, **xand**　and命令に展開
soor, **xoor**　oor命令に展開
sxor, **xxor**　xor命令に展開
snot, **xnot**　not命令に展開

●展開形式

(1) s*OP*　%rd,*imm16* / x*OP*　%rd,*imm16*　(*OP* = and, oor, xor, not)

例: xand　%rd,*imm16*

imm16 ≤ 0x7f	0x7f < *imm16*
and　%rd,*imm16*(6:0)	ext　*imm16*(15:7) and　%rd,*imm16*(6:0)

8.6.4 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
sld.b %rd,[%sp+*imm20*]	%rd ← B[%sp+*imm20*]（**符号拡張**）	(1)
sld.ub %rd,[%sp+*imm20*]	%rd ← B[%sp+*imm20*]（**ゼロ拡張**）	(1)
sld %rd,[%sp+*imm20*]	%rd ← W[%sp+*imm20*]	(1)
sld.a %rd,[%sp+*imm20*]	%rd ← A[%sp+*imm20*](23:0), **無視** ← A[%sp+*imm20*](31:24)	(1)
sld.b [%sp+*imm20*],%rs	B[%sp+*imm20*] ← %rs(7:0)	(1)
sld [%sp+*imm20*],%rs	W[%sp+*imm20*] ← %rs(15:0)	(1)
sld.a [%sp+*imm20*],%rs	A[%sp+*imm20*](23:0) ← %rs(23:0), A[%sp+*imm20*](31:24) ← 0	(1)
xld.b %rd,[%sp+*imm24*]	%rd ← B[%sp+*imm24*]（**符号拡張**）	(2)
xld.ub %rd,[%sp+*imm24*]	%rd ← B[%sp+*imm24*]（**ゼロ拡張**）	(2)
xld %rd,[%sp+*imm24*]	%rd ← W[%sp+*imm24*]	(2)
xld.a %rd,[%sp+*imm24*]	%rd ← A[%sp+*imm24*](23:0), **無視** ← A[%sp+*imm24*](31:24)	(2)
xld.b [%sp+*imm24*],%rs	B[%sp+*imm24*] ← %rs(7:0)	(2)
xld [%sp+*imm24*],%rs	W[%sp+*imm24*] ← %rs(15:0)	(2)
xld.a [%sp+*imm24*],%rs	A[%sp+*imm24*](23:0) ← %rs(23:0), A[%sp+*imm24*](31:24) ← 0	(2)

これらの拡張命令により、20ビット/24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。*imm20*/*imm24*は省略可能です。

●展開後の基本命令

sld.b, **xld.b**　　ld.b命令に展開
sld.ub, **xld.ub**　　ld.ub命令に展開
sld, **xld**　　ld命令に展開
sld.a, **xld.a**　　ld.a命令に展開

●展開形式

imm20、*imm24*を省略した場合、[%sp+0x0]を指定したものとして展開されます。

(1) s*OP* %rd,[%sp+*imm20*]　　(*OP* = ld.b, ld.ub, ld, ld.a)
s*OP* [%sp+*imm20*],%rs　　(*OP* = ld.b, ld, ld.a)
例: sld.a %rd,[%sp+*imm20*]

imm20 ≤ 0x7f	0x7f < *imm20*
ld.a %rd,[%sp+*imm20*(6:0)]	ext *imm20*(19:7) ld.a %rd,[%sp+*imm20*(6:0)]

(2) x*OP* %rd,[%sp+*imm24*]　　(*OP* = ld.b, ld.ub, ld, ld.a)
x*OP* [%sp+*imm24*],%rs　　(*OP* = ld.b, ld, ld.a)
例: xld.a %rd,[%sp+*imm24*]

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
ld.a %rd,[%sp+*imm24*(6:0)]	ext *imm24*(19:7) ld.a %rd,[%sp+*imm24*(6:0)]	ext *imm24*(23:20) ext *imm24*(19:7) ld.a %rd,[%sp+*imm24*(6:0)]

8.6.5 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
sld.b %rd,[*imm20*]	%rd ← B[*imm20*]（**符号拡張**）	(1)
sld.ub %rd,[*imm20*]	%rd ← B[*imm20*]（**ゼロ拡張**）	(1)
sld %rd,[*imm20*]	%rd ← W[*imm20*]	(1)
sld.a %rd,[*imm20*]	%rd ← A[*imm20*](23:0), **無視** ← A[*imm20*](31:24)	(1)
sld.b [*imm20*],%rs	B[*imm20*] ← %rs(7:0)	(1)
sld [*imm20*],%rs	W[*imm20*] ← %rs(15:0)	(1)
sld.a [*imm20*],%rs	A[*imm20*](23:0) ← %rs(23:0), A[*imm20*](31:24) ← 0	(1)
xld.b %rd,[*imm24*]	%rd ← B[*imm24*]（**符号拡張**）	(2)
xld.ub %rd,[*imm24*]	%rd ← B[*imm24*]（**ゼロ拡張**）	(2)
xld %rd,[*imm24*]	%rd ← W[*imm24*]	(2)
xld.a %rd,[*imm24*]	%rd ← A[*imm24*](23:0), **無視** ← A[*imm24*](31:24)	(2)
xld.b [*imm24*],%rs	B[*imm24*] ← %rs(7:0)	(2)
xld [*imm24*],%rs	W[*imm24*] ← %rs(15:0)	(2)
xld.a [*imm24*],%rs	A[*imm24*](23:0) ← %rs(23:0), A[*imm24*](31:24) ← 0	(2)

これらの拡張命令により、20ビット/24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。

●展開後の基本命令

sld.b, **xld.b**　　ld.b命令に展開
sld.ub, **xld.ub**　　ld.ub命令に展開
sld, **xld**　　ld命令に展開
sld.a, **xld.a**　　ld.a命令に展開

●展開形式

(1) **s*OP* %rd,[*imm20*]**　　(*OP* = ld.b, ld.ub, ld, ld.a)
s*OP* [*imm20*],%rs　　(*OP* = ld.b, ld, ld.a)
例: sld.a %rd,[*imm20*]

imm20 ≤ 0x7f	0x7f < *imm20*
ld.a %rd,[*imm20*(6:0)]	ext *imm20*(19:7) ld.a %rd,[*imm20*(6:0)]

(2) **x*OP* %rd,[*imm24*]**　　(*OP* = ld.b, ld.ub, ld, ld.a)
x*OP* [*imm24*],%rs　　(*OP* = ld.b, ld, ld.a)
例: xld.a %rd,[*imm24*]

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
ld.a %rd,[*imm24*(6:0)]	ext *imm24*(19:7) ld.a %rd,[*imm24*(6:0)]	ext *imm24*(23:20) ext *imm24*(19:7) ld.a %rd,[*imm24*(6:0)]

8.6.6 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
sld %rd,*imm16*	%rd ← *imm16*	(1)
sld.a %rd,*imm20*	%rd ← *imm20*	(2)
sld.a %sp,*imm20*	%sp ← *imm20*	(2)
sld %rd,*symbol*±*imm16*	%rd ← *symbol*±*imm16*(15:0)	(4)
sld.a %rd,*symbol*±*imm20*	%rd ← *symbol*±*imm20*(19:0)	(5)
sld.a %sp,*symbol*±*imm20*	%sp ← *symbol*±*imm20*(19:0)	(5)
xld %rd,*imm16*	%rd ← *imm16*	(1)
xld.a %rd,*imm24*	%rd ← *imm24*	(3)
xld.a %sp,*imm24*	%sp ← *imm24*	(3)
xld %rd,*symbol*±*imm16*	%rd ← *symbol*±*imm16*(15:0)	(4)
xld.a %rd,*symbol*±*imm24*	%rd ← *symbol*±*imm24*(23:0)	(6)
xld.a %sp,*symbol*±*imm24*	%sp ← *symbol*±*imm24*(23:0)	(6)

これらの拡張命令により、16ビット/20ビット/24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

sld, **xld**　　ld命令に展開
sld.a, **xld.a**　　ld.a命令に展開

●展開形式

(1) sld %rd,*imm16* / xld %rd,*imm16*

例: xld %rd,*imm16*

imm16 ≤ 0x7f	0x7f < *imm16*
ld %rd,*imm16*(6:0)	ext *imm16*(15:7) ld %rd,*imm16*(6:0)

(2) sld.a %rd,*imm20* / sld.a %sp,*imm20*

例: sld.a %rd,*imm20*

imm20 ≤ 0x7f	0x7f < *imm20*
ld.a %rd,*imm20*(6:0)	ext *imm20*(19:7) ld.a %rd,*imm20*(6:0)

(3) xld.a %rd,*imm24* / xld.a %sp,*imm24*

例: xld.a %rd,*imm24*

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
ld.a %rd,*imm24*(6:0)	ext *imm24*(19:7) ld.a %rd,*imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) ld.a %rd,*imm24*(6:0)

(4) sld %rd,*symbol*±*imm16* / xld %rd,*symbol*±*imm16*

例: sld %rd,*symbol*±*imm16*

無条件
ext (*symbol*±*imm16*)(15:7) ld %rd,(*symbol*±*imm16*)(6:0)

(5) sld.a %rd,*symbol*±*imm20* / sld.a %sp,*symbol*±*imm20*

例: sld.a %rd,*symbol*±*imm20*

無条件
ext (*symbol*±*imm20*)(19:7) ld.a %rd,(*symbol*±*imm20*)(6:0)

(6) xld.a %rd,*symbol±imm24* / xld.a %sp,*symbol±imm24*

例: xld.a %rd,*symbol±imm24*

無条件
ext (*symbol±imm24*)(23:20)
ext (*symbol±imm24*)(19:7)
ld.a %rd,(*symbol±imm24*)(6:0)

8.6.7 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±imm20*	PC相対サブルーチンコール	(1)
sjpr	*label±imm20*	PC相対無条件ジャンプ	(1)
sjreq	*label±imm20*	PC相対条件付きジャンプ	(2)
sjrne	*label±imm20*	PC相対条件付きジャンプ	(2)
sjrgt	*label±imm20*	PC相対条件付きジャンプ	(2)
sjrge	*label±imm20*	PC相対条件付きジャンプ	(2)
sjrlt	*label±imm20*	PC相対条件付きジャンプ	(2)
sjrle	*label±imm20*	PC相対条件付きジャンプ	(2)
sjrugt	*label±imm20*	PC相対条件付きジャンプ	(2)
sjruge	*label±imm20*	PC相対条件付きジャンプ	(2)
sjrult	*label±imm20*	PC相対条件付きジャンプ	(2)
sjrule	*label±imm20*	PC相対条件付きジャンプ	(2)
scalla	*label±imm20*	PC絶対サブルーチンコール	(3)
sjpa	*label±imm20*	PC絶対無条件ジャンプ	(3)
scall	*sign20*	PC相対サブルーチンコール	(4)
sjpr	*sign20*	PC相対無条件ジャンプ	(4)
sjreq	*sign20*	PC相対条件付きジャンプ	(5)
sjrne	*sign20*	PC相対条件付きジャンプ	(5)
sjrgt	*sign20*	PC相対条件付きジャンプ	(5)
sjrge	*sign20*	PC相対条件付きジャンプ	(5)
sjrlt	*sign20*	PC相対条件付きジャンプ	(5)
sjrle	*sign20*	PC相対条件付きジャンプ	(5)
sjrugt	*sign20*	PC相対条件付きジャンプ	(5)
sjruge	*sign20*	PC相対条件付きジャンプ	(5)
sjrult	*sign20*	PC相対条件付きジャンプ	(5)
sjrule	*sign20*	PC相対条件付きジャンプ	(5)
scalla	*imm20*	PC絶対サブルーチンコール	(6)
sjpa	*imm20*	PC絶対無条件ジャンプ	(6)
xcall	*label±imm24*	PC相対サブルーチンコール	(7)
xjpr	*label±imm24*	PC相対無条件ジャンプ	(7)
xjreq	*label±imm24*	PC相対条件付きジャンプ	(8)
xjrne	*label±imm24*	PC相対条件付きジャンプ	(8)
xjrgt	*label±imm24*	PC相対条件付きジャンプ	(8)
xjrge	*label±imm24*	PC相対条件付きジャンプ	(8)
xjrlt	*label±imm24*	PC相対条件付きジャンプ	(8)
xjrle	*label±imm24*	PC相対条件付きジャンプ	(8)
xjrugt	*label±imm24*	PC相対条件付きジャンプ	(8)
xjruge	*label±imm24*	PC相対条件付きジャンプ	(8)
xjrult	*label±imm24*	PC相対条件付きジャンプ	(8)
xjrule	*label±imm24*	PC相対条件付きジャンプ	(8)
xcalla	*label±imm24*	PC絶対サブルーチンコール	(9)
xjpa	*label±imm24*	PC絶対無条件ジャンプ	(9)
xcall	*sign24*	PC相対サブルーチンコール	(10)
xjpr	*sign24*	PC相対無条件ジャンプ	(10)
xjreq	*sign24*	PC相対条件付きジャンプ	(11)
xjrne	*sign24*	PC相対条件付きジャンプ	(11)
xjrgt	*sign24*	PC相対条件付きジャンプ	(11)
xjrge	*sign24*	PC相対条件付きジャンプ	(11)
xjrlt	*sign24*	PC相対条件付きジャンプ	(11)
xjrle	*sign24*	PC相対条件付きジャンプ	(11)
xjrugt	*sign24*	PC相対条件付きジャンプ	(11)
xjruge	*sign24*	PC相対条件付きジャンプ	(11)
xjrult	*sign24*	PC相対条件付きジャンプ	(11)
xjrule	*sign24*	PC相対条件付きジャンプ	(11)
xcalla	*imm24*	PC絶対サブルーチンコール	(12)
xjpa	*imm24*	PC絶対無条件ジャンプ	(12)

これらの拡張命令により、20ビット/24ビット即値、またはラベル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。
拡張命令も、".d"を付加してディレイド分岐命令として使用可能です。
例: xcall.d *sign24*

●展開後の基本命令

scall, **scall.d**, **xcall**, **xcall.d**	call/call.d命令に展開
scalla, **scalla.d**, **xcalla**, **xcalla.d**	calla/calla.d命令に展開
sjpa, **sjpa.d**, **xjpa**, **xjpa.d**	jpa/jpa.d命令に展開
sjpr, **sjpr.d**, **xjpr**, **xjpr.d**	jpr/jpr.d命令に展開
sjreq, **sjreq.d**, **xjreq**, **xjreq.d**	jreq/jreq.d命令に展開
sjrne, **sjrne.d**, **xjrne**, **xjrne.d**	jrne/jrne.d命令に展開
sjrgt, **sjrgt.d**, **xjrgt**, **xjrgt.d**	jrgt/jrgt.d命令に展開
sjrge, **sjrge.d**, **xjrge**, **xjrge.d**	jrge/jrge.d命令に展開
sjrlt, **sjrlt.d**, **xjrlt**, **xjrlt.d**	jrlt/jrlt.d命令に展開
sjrle, **sjrle.d**, **xjrle**, **xjrle.d**	jrle/jrle.d命令に展開
sjrugt, **sjrugt.d**, **xjrugt**, **xjrugt.d**	jrugt/jrugt.d命令に展開
sjruge, **sjruge.d**, **xjruge**, **xjruge.d**	jruge/jruge.d命令に展開
sjrult, **sjrult.d**, **xjrult**, **xjrult.d**	jrult/jrult.d命令に展開
sjrule, **sjrule.d**, **xjrule**, **xjrule.d**	jrule/jrule.d命令に展開

●展開形式

(1) ***sOP label±imm20*** (*OP* = call, call.d, jpr, jpr.d)
例: scall *label±imm20*

無条件
ext (*label±imm20*)(19:12)
call (*label±imm20*)(11:1)

(2) ***sOP label±imm20*** (*OP* = jr*, jr*.d)
例: sjreq *label±imm20*

無条件
ext (*label±imm20*)(19:8)
jreq (*label±imm20*)(7:1)

(3) ***sOP label±imm20*** (*OP* = calla, calla.d, jpa, jpa.d)
例: scalla *label±imm20*

無条件
ext (*label±imm20*)(19:7)
calla (*label±imm20*)(6:0)

(4) ***sOP sign20*** (*OP* = call, call.d, jpr, jpr.d)
例: scall *sign20*

-1024 ≤ *sign20* ≤ 1023	*sign20* < -1024 または 1023 < *sign20*
call *sign20*(11:1)	ext *sign20*(19:12) call *sign20*(11:1)

(5) ***sOP sign20*** (*OP* = jr*, jr*.d)
例: sjreq *sign20*

-128 ≤ *sign20* ≤ 127	*sign20* < -128 または 127 < *sign20*
jreq *sign20*(7:1)	ext *sign20*(19:8) jreq *sign20*(7:1)

(6) sOP imm20 (*OP* = calla, calla.d, jpa, jpa.d)

例: scalla *imm20*

imm20 ≤ 0x7f	0x7f < *imm20*
calla *imm20*(6:0)	ext *imm20*(19:7) calla *imm20*(6:0)

(7) xOP label±imm24 (*OP* = call, call.d, jpr, jpr.d)

例: xcall *label±imm24*

無条件
ext (*label±imm24*)(23:12) call (*label±imm24*)(11:1)

(8) xOP label±imm24 (*OP* = jr*, jr*.d)

例: xjreq *label±imm24*

無条件
ext (*label±imm24*)(23:21) ext (*label±imm24*)(20:8) jreq (*label±imm24*)(7:1)

(9) xOP label±imm24 (*OP* = calla, calla.d, jpa, jpa.d)

例: xcalla *label±imm24*

無条件
ext (*label±imm24*)(23:20) ext (*label±imm24*)(19:7) calla (*label±imm24*)(6:0)

(10) xOP sign24 (*OP* = call, call.d, jpr, jpr.d)

例: xcall *sign24*

-1024 ≤ *sign24* ≤ 1023	*sign24* < -1024 または 1023 < *sign24*
call *sign24*(11:1)	ext *sign24*(23:12) call *sign24*(11:1)

(11) xOP sign24 (*OP* = jr*, jr*.d)

例: xjreq *sign24*

-128 ≤ *sign24* ≤ 127	-1048576 ≤ *sign24* < -128 または 127 < *sign24* ≤ 1048575	*sign24* < -1048576 または 1048575 < *sign24*
jreq *sign24*(7:1)	ext *sign24*(20:8) jreq *sign24*(7:1)	ext *sign24*(23:21) ext *sign24*(20:8) jreq *sign24*(7:1)

(12) xOP imm24 (*OP* = calla, calla.d, jpa, jpa.d)

例: xcalla *imm24*

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
calla *imm24*(6:0)	ext *imm24*(19:7) calla *imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) calla *imm24*(6:0)

8.6.8 コプロセッサ命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
sld.cw %rd,*imm20*	コプロセッサ ← %rd & *imm20*	(1)
sld.ca %rd,*imm20*	コプロセッサ ← %rd & *imm20*, 結果&フラグ取得	(1)
sld.cf %rd,*imm20*	コプロセッサ ← %rd & *imm20*, フラグ取得	(1)
sld.cw %rd,*symbol*±*imm20*	コプロセッサ ← %rd & *symbol*±*imm20*	(2)
sld.ca %rd,*symbol*±*imm20*	コプロセッサ ← %rd & *symbol*±*imm20*, 結果&フラグ取得	(2)
sld.cf %sp,*symbol*±*imm20*	コプロセッサ ← %rd & *symbol*±*imm20*, フラグ取得	(2)
xld.cw %rd,*imm24*	コプロセッサ ← %rd & *imm24*	(3)
xld.ca %rd,*imm24*	コプロセッサ ← %rd & *imm24*, 結果&フラグ取得	(3)
xld.cf %rd,*imm24*	コプロセッサ ← %rd & *imm24*, フラグ取得	(3)
xld.cw %rd,*symbol*±*imm24*	コプロセッサ ← %rd & *symbol*±*imm24*	(4)
xld.ca %rd,*symbol*±*imm24*	コプロセッサ ← %rd & *symbol*±*imm24*, 結果&フラグ取得	(4)
xld.cf %rd,*symbol*±*imm24*	コプロセッサ ← %rd & *symbol*±*imm24*, フラグ取得	(4)

これらの拡張命令により、20ビット/24ビットの即値をコプロセッサに転送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

sld.cw, **xld.cw**　ld.cw命令に展開
sld.ca, **xld.ca**　ld.ca命令に展開
sld.cf, **xld.cf**　ld.cf命令に展開

●展開形式

(1) s*OP* %rd,*imm20*　(*OP* = ld.cw, ld.ca, ld.cf)

例: sld.ca %rd,*imm20*

imm20 ≤ 0x7f	0x7f < *imm20*
ld.ca %rd,*imm20*(6:0)	ext *imm20*(19:7) ld.ca %rd,*imm20*(6:0)

(2) s*OP* %rd,*symbol*±*imm20*　(*OP* = ld.cw, ld.ca, ld.cf)

例: sld.ca %rd,*symbol*±*imm20*

無条件
ext (*symbol*±*imm20*)(19:7) ld.ca %rd,(*symbol*±*imm20*)(6:0)

(3) x*OP* %rd,*imm24*　(*OP* = ld.cw, ld.ca, ld.cf)

例: xld.ca %rd,*imm24*

imm24 ≤ 0x7f	0x7f < *imm24* ≤ 0xfffff	0xfffff < *imm24*
ld.ca %rd,*imm24*(6:0)	ext *imm24*(19:7) ld.ca %rd,*imm24*(6:0)	ext *imm24*(23:20) ext *imm24*(19:7) ld.ca %rd,*imm24*(6:0)

(4) x*OP* %rd,*symbol*±*imm24*　(*OP* = ld.cw, ld.ca, ld.cf)

例: xld.ca %rd,*symbol*±*imm24*

無条件
ext (*symbol*±*imm24*)(23:20) ext (*symbol*±*imm24*)(19:7) ld.ca %rd,(*symbol*±*imm24*)(6:0)

8.6.9 Xext命令

●拡張命令の種類と機能

拡張命令	機能	展開形式
Xext *imm24*	ext 命令に展開	(1)

以下の命令と組み合わせることにより、オフセットとして機能します。

```
Xext  imm24
OP    [%rd], %rs   ==> [%rd+imm24] ← %rs として機能します。

Xext  imm24
OP    %rd, [%rs]   ==> %rd ← [%rs+imm24] として機能します。
```

(*OP* = ld.b, ld.ub, ld, ld.a)

●展開後の基本命令

Xext ext命令に展開

●展開形式

(1) Xext *imm24*

imm24 ≤ 0x1fff	0x1fff < *imm24* ≤ 0xffffff
ext *imm20*(12:0)	ext *imm24*(23:13) ext *imm24*(12:0)

8.7 拡張命令の最適化

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-mc17_extです。

●-mc17_extオプション

-mc17_ext *filename*.dump *filename*.map

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

filename.map　オブジェクトファイルの配置を示すマップ情報ファイルで、リンカが出力します。拡張子は.mapである必要があります。

例:
```
objdump -t test.elf > test.dump
as -mc17_ext test.dump test.map -otest.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-mc17_extオプションで指定されるリンクマップファイルおよび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(-mc17_extオプションなし)
3. リンク(マップファイルの出力が必要)
4. **objdump**でelfファイルからダンプファイルを生成
5. 再アセンブル(-mc17_extオプション付き)　＊2と同じ入出力ファイルを指定

IDEで生成される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

例:
```
sub:
        ret

main:
        xsub.a  %sp,0x4
                :
                :
        xld.b   %r0,[sub]
        add.a   %sp,0x4
        ret
```

シンボル(sub)のアドレスをダンプファイル/マッ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 = 0～0x7f: extなしで展開
シンボルのアドレス = 0x80～0xffffff: ext 1個で展開
シンボルのアドレス = 0xffffff～0xffffffff: ext 2個で展開

最適化2: 算術演算命令の場合

例:
```
sub:
        ret

main:
        xsub.a  %sp,0x4
                :
                :
        xadd    %r1,sub
        add.a   %sp,0x4
        ret
```

シンボル(sub)のアドレスをダンプファイル/マッ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 = 0～0x7f: extなしで展開
シンボルのアドレス = 0x80～0xffff: ext 1個で展開

最適化3: 相対分岐命令の場合(scall/xcall/sjpr/xjpr/sjrxx/xjrxx命令)

例:
```
sub:
        ret

main:
        xsub.a  %sp,0x4
                :
                :
        xcall   sub             ← ADDR1
        add.a   %sp,0x4
        ret
```

分岐先(sub)のアドレスはダンプファイル/マッ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/マッ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。

scall/xcall/sjpr/xjpr命令
分岐距離 = -1024～1022: extなしで展開
分岐距離 = -16,777,216～-1026または1024～16,777,214: ext 1個で展開

sjrxx/xjrxx命令
分岐距離 = -128～126: extなしで展開
分岐距離 = -1,048,576～-130または128～1,048,574: ext 1個で展開
分岐距離 = -16,777,216～-1,048,578または1,048,576～16,777,214: 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
     xjrgt  L1  ;(L1 - 分岐元のアドレス - 2) >> 1 → -2
```

最適化前		最適化後
nop		nop
ext 0x7	← オペランド1(*imm3*)	jrgt 0x7e
ext 0x1fff	← オペランド2(*imm13*)	
jrgt 0x7c	← オペランド3(*sign7*)	

最適化4: 絶対分岐命令の場合(scalla/xcalla/sjpa/xjpa命令)

例:
```
sub:
     ret

main:
     xsub.a  %sp,0x4
             :
             :
     xcalla  sub
     add.a   %sp,0x4
     ret
```

分岐先(sub)のアドレスをダンプファイル/マップファイルから取得します。絶対分岐命令の場合は、分岐先シンボルのアドレス値のみでextの数を決定します。

分岐先アドレス = 0〜0x7f: extなしで展開
分岐先アドレス = 0x80〜0xfffff: ext 1個で展開
分岐先アドレス = 0xfffff〜0xffffff: ext 2個で展開

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:
     ret

main:
     xsub.a  %sp,0x4
             :
             :
     xldb    %r0,[sub+0x2]
     add.a   %sp,0x4
     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。

計算結果 = 0〜0x7f: extなしで展開
計算結果 = 0x80〜0xfffff: ext 1個で展開
計算結果 = 0xfffff〜0xffffff: ext 2個で展開

●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、算術演算命令、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、マップファイルを使用したグローバルシンボルの最適化は行われます。

8.8 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8.8.1 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: junk at end of line: *'XXXXX'*	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: 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。
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. *	`-mc17_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: Cannot find the map file. *	`-mc17_ext`が指定されていますが、マップファイル名が指定されていません。あるいは、指定されたマップファイルが存在しません。
Error: The format of the dump file is invalid. *	`-mc17_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: The format of the map file is invalid. *	`-mc17_ext`で指定されているマップファイルの内容が不正です。
Error: Cannot close the map file. *	`-mc17_ext`で指定されたマップファイル読み込み後、ファイルのクローズに失敗しました。
Error: There are too many characters of one line in dump file. *	`-mc17_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: There are too many characters of one line in map file. *	`-mc17_ext`で指定されたマップファイルの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: Value of XXXX too large for field of AAA bytes at BBB	.textセクションの先頭からXXXXバイト目にあるラベルのアドレスは大きすぎます。このラベルは.rodataセクションの先頭からBBBバイト目にあるAAAバイト型のシンボルテーブルから参照されています。
Error : Failed to register hash symbols in source file. :XXXX	`-mc17_ext`が指定されていますが、ソースファイルのシンボル名の登録に 'XXXX' が失敗しました。
Error : Failed to register hash symbols in dump file. :XXXX	`-mc17_ext`が指定されていますが、ダンプファイルのシンボル名の登録に 'XXXX' が失敗しました。
Error : Failed to register hash symbols in map file. :XXXX	`-mc17_ext`が指定されていますが、マップファイルのシンボル名の登録に 'XXXX' が失敗しました。

* **xgcc経由のアセンブル時は、これらの代わりに次のエラーメッセージを表示して終了します。**

```
xgcc: cannot specify -o with -c or -S and multiple compilations
```

表8.8.2 ワーニングメッセージ

ワーニングメッセージ	内容
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`

8.9 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。
- リンク時の不具合を防止するため、.section擬似命令を記述する場合は必ず.align命令も併記し、セクションの境界を明確にしてください。
  例:

```
            .section .rodata
            .align 2            ;←必須
    .long   data1
    .long   data2
```

- グローバルシンボル名の文字数は最大512文字、定義可能なグローバルシンボルの数は最大32,000個に制限されます。
- アセンブラソースファイル、マップファイル、ダンプファイルの1行の文字数は最大2,047文字(改行文字を除く)に制限されます。これを超えた場合はエラーになります。
- アセンブラ**as**の入力に複数のソースファイルを指定することはできません。指定した場合はエラーになります。
- -oオプションで指定する出力ファイル名(拡張子を除く)は入力ソースファイルと同じ名称にする必要があります。
- 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、-mc17_extオプションによる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、算術演算命令、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、-mc17_extオプションによる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、マップファイルを使用したグローバルシンボルの最適化は行われます。

このページはブランクです。

S5U1C17001C Manual

9 リンカ

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

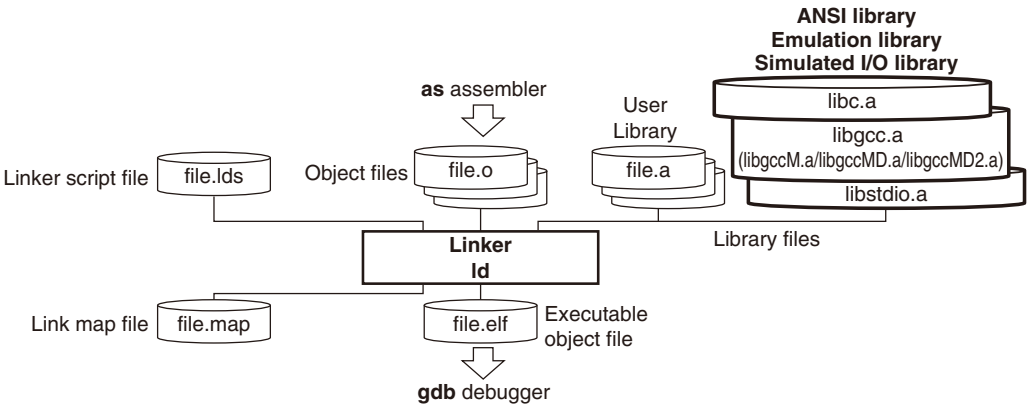


図9.2.1 フローチャート

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、およびユーザのライブラリファイルです。

●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: デバッガ**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。デバッグに必要な情報も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¥24bit¥libc.a ..¥lib¥24bit¥libgcc.a

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 *<リンカ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: デフォルトリンカスクリプト(9.4.1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。

-c17-overlap-noerr

機能: **セクションのアドレス配置の重複を許可**
説明: -c17-overlap-noerrオプションを指定すると、複数のセクションのアドレス配置が重複した場合でもエラーにせず、リンカを正常終了させます。
デフォルト: リンカはセクションのアドレス配置が重複したときエラーにします。

-c17-memoryover-noerr

機能: **メモリ領域オーバーエラーの禁止**
説明: プログラムがメモリ領域を超えてもエラーになりません。
デフォルト: メモリ領域を超えるとエラーになります。

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¥24bit¥libc.a`

9.4 リンク処理

9.4.1 デフォルトリンカスクリプト

-Tオプション未指定時のデフォルトリンカスクリプト

-Tオプションが指定されないと、リンカ**ld**は下記のリンカスクリプトを使用してリンクの処理をします。

```
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);
SECTIONS
{
    /* stack pointer symbols */
    __START_stack = 0x000FC0;

    /* section information */
    .bss 0x0 :
      {
       __START_bss = . ;
       *(.bss)
       __END_bss = . ;
      }
    .data __END_bss : AT( __END_rodata )
      {
       __START_data = . ;
       *(.data)
       __END_data = . ;
      }
    .text 0x8000 :
      {
       __START_vector = . ;
       *(.vector)
       __END_vector = . ;
       . = 0x80 ;
       __START_text = . ;
       *(.text)
       __END_text = . ;
      }
    .rodata __END_text :
      {
       __START_rodata = . ;
       *(.rodata)
       __END_rodata = . ;
      }
    __START_data_lma = __END_rodata ;
    __END_data_lma = __END_rodata + ( __END_data - __START_data );
}
```

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、ベクタテーブル、プログラムコード、および定数データはアドレス0x8000から配置されます。
図9.4.1.1にリンク後のメモリマップを示します。

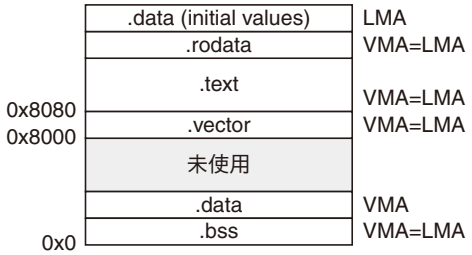


図9.4.1.1 デフォルトリンカスクリプトによるメモリマップ

9.4.2 リンク例

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-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{
   /* stack pointer symbols */
   __START_stack = 0x000FC0;

   /* location counter */
   . = 0x0;

   /* section information */
   .bss 0x000000 :
     {
      __START_bss = . ;
      *(.bss) ;
      __END_bss = . ;
     }

   .data __END_bss : AT( __END_text )
     {
      __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 );
    .rodata 0x008000 :
      {
       __START_rodata = . ;
       *(.rodata) ;
       __END_rodata = . ;
      }
    .text __END_rodata :
      {
       __START_text = . ;
       *(.text) ;
       __END_text = . ;
      }
}
```

図9.4.2.1にセクションの配置を示します。

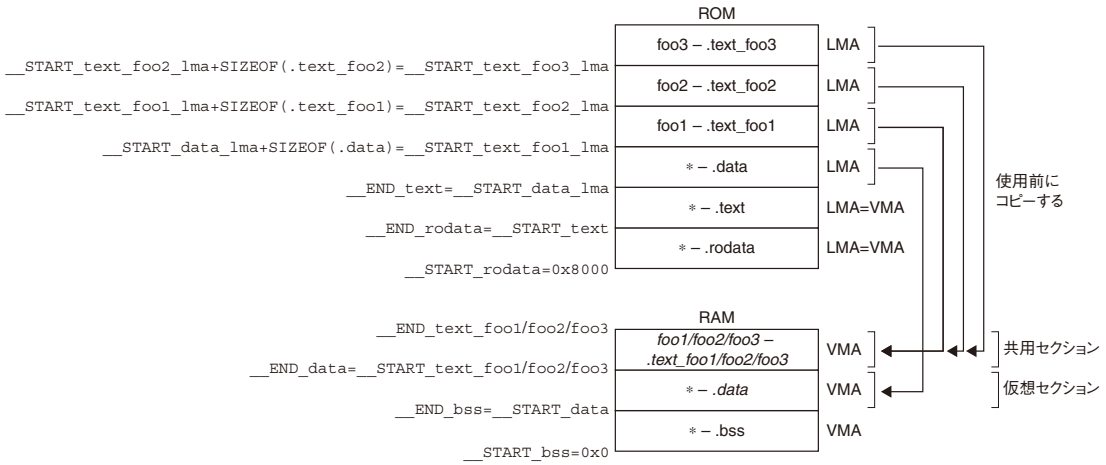


図9.4.2.1 メモリマップ

.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に設定しています。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で実行するように設定しています。.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.rodataセクション(0x8000～)に続く.textセクションに配置され、そのままROM上で実行されます。

このリンク例におけるROMおよびRAMの使用量は、以下のように求めることができます。

ROM使用量[bytes] = __END_text_foo3_lma - __START_rodata
RAM使用量[bytes] = __END_text_foo3 - __START_bss + スタック使用量 (+ヒープ使用量)

__END_text_foo3_lmaなどのシンボルの具体的な値は、リンクマップファイルに出力されます。詳細は、"9.4.3 リンクマップ"を参照してください。

9.4.3 リンクマップ

コマンドラインオプションに-M(または-Map)を指定すると、リンカ**ld**はリンクマップ情報を出力します。
9.4.1に示したデフォルトリンカスクリプトを用いたとき、出力されるリンクマップの例を以下に示します。

Memory Configuration

Name	Origin	Length	Attributes
default	0x00000000	0xffffffff	

Linker script and memory map

```
                0x00000fc0                __START_stack=0xfc0
                0x00000000                .=0x0

.bss            0x00000000        0x0
                0x00000000                __START_bss=.
 *(.bss)
                0x00000000                __END_bss=.

.data           0x00000000        0x0 load address 0x00008026
                0x00000000                __START_data=.
 *(.data)
                0x00000000                __END_data=.

.text           0x00008000       0x26
                0x00008000                __START_vector=.
 *(.vector)
 .vector        0x00008000       0x10 vector.o
                0x00008000                vector
                0x00008010                __END_vector=.
                0x00008010                __START_text=.
 *(.text)
 .text          0x00008010       0x16 vector.o
                0x00008010                boot
                0x0000801a                main
                0x0000801e                dummy
                0x00008022                nmi
                0x00008026                __END_text=.

.rodata         0x00008026        0x0
                0x00008026                __START_rodata=.
 *(.rodata)
                0x00008026                __END_rodata=.
                0x00008026                __START_data_lma=__END_rodata
                0x00008026                __END_data_lma=(__END_rodata+(__END_data-
__START_data))
LOAD vector.o
LOAD C:/EPSON/GNU17/lib/24bit/libstdio.a
LOAD C:/EPSON/GNU17/lib/24bit/libc.a
LOAD C:/EPSON/GNU17/lib/24bit/libgcc.a
LOAD C:/EPSON/GNU17/lib/24bit/libc.a
OUTPUT(testsim.elf elf32-c17)

*** Section .stab will not be loaded to the target ***

.stab           0x00008028      0x1f8
 .stab          0x00008028      0x1f8 vector.o

*** Section .stabstr will not be loaded to the target ***
```

```
.stabstr        0x00008220      0x38e
 .stabstr       0x00008220      0x38e vector.o
                                  0x0 (size before relaxing)
```

"Memory Configuration"として、"Name"列にセクション名が、"Origin"列に開始アドレスが、"Length"列に各セクションのバイト数が、"Attributes"列に開始アドレスに対応するシンボル名が示されています。

リンカスクリプトによる指定に従い、.bssセクションがアドレス0x00000000から始まり、それに続いて.dataセクションが配置されています。さらに.textセクションがアドレス0x00008000から始まり、それに続いて.rodataセクションが配置されています。

シンボル__START_stackは、リンカスクリプトによる指定に従い、アドレス0x00000fc0に配置されています。
その他のシンボル__START_bss, __END_bss, __START_data, __END_data, __START_vector, __END_vector, __START_text, __END_text, __START_rodata, __END_rodata, __START_data_lma, __END_data_lmaについても、リンカ**ld**により配置されたアドレスが"Origin"列に示されています。これらのシンボルを参照することで、プログラム中から各セクションのアドレスやサイズを知ることが可能です。

.stabセクションと.stabstrセクションは、デバッガが参照するデバッグ情報をプログラムに埋め込むために使用されています。これらのセクションの内容は、ターゲットにロードされません。

9.5 エラーメッセージ

エラー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: The offset value of a symbol is over 16bit.	シンボルのアドレスが16ビットアドレス空間を超えています。
Error: The offset value of a symbol is over 24bit.	シンボルのアドレスが24ビットアドレス空間を超えています。
Error: section *XXX* is not within 16bit address.	*XXX*セクションのアドレスが16ビットアドレス空間を超えています。
Error: section *XXX* is not within 24bit address.	*XXX*セクションのアドレスが24ビットアドレス空間を超えています。
Error: Input bject file <objectfile> [included from <archivefile>] is not for C17.	オブジェクトファイルはC17用ではありません。
Error: Input object file <objectfile> is not 16bit nor 24bit address mode.	オブジェクトファイルが16ビットモードでも24ビットモードでもありません。
Error: Cannot link 16bit object <objectfile16> [included from <archivefile16>] with 24bit object <objectfile24> [included from <archivefile24>]	16ビットポインタモードで作成されたオブジェクトファイルと、24ビットポインタモードで作成されたオブジェクトファイルはリンクできません。

9.6 注意事項

- 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)

  ```
        :
  .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) ;
  }
  ```

- 同じ関数を含むライブラリファイル(*.a)をリンクしてもエラーになりません(2重にリンクしません)。同じ関数を含むオブジェクトファイル(*.o)をリンクした場合はエラーになります。

- 変数または変数との演算により、配置されるアドレスが24ビット(0xffffff)を超える場合、もしくは0x0より小さくなる場合はエラーにはならず、アドレスの24ビット以上は0にマスクされます。
  例: xadd.a %r0,symbol-5
  symbolのアドレスが0番地のとき、絶対アドレスは0 - 5 = 0xfffffb(-5)となります。したがって、xadd.a %r0,0xfffffbとして出力コードが生成されます。

S5U1C17001C Manual

10 デバッガ

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
- ICD Mini(S5U1C17001H)またはICDボード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ブレーク機能
- 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(シミュレータモード時)
- 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Stdin/Stdoutの入出力を評価可能なシミュレーテッドI/O機能
- デバッガ上でターゲット機種の入出力ポートやSVD、LCD表示を評価可能な組み込みシステムシミュレータに対応
- ICD Mini(S5U1C17001H)のフラッシュライタ機能に対応

10.2 入出力ファイル

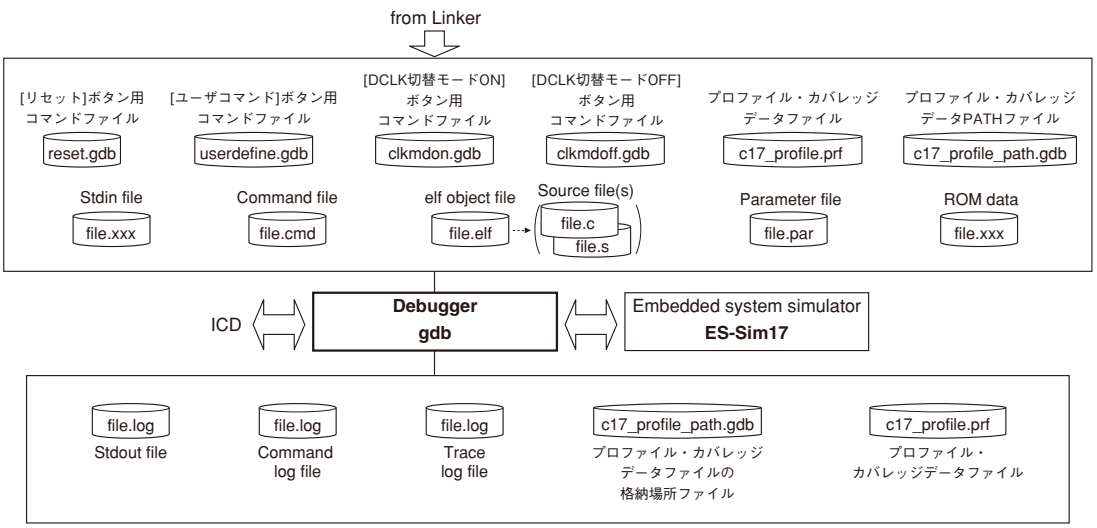


図10.2.1 フローチャート

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: デバッガ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> プロパティー > GNU17 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、"10.9 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 （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

ファイル形式: モトローラ(S1 ～ S3)形式
ファイル名: 任意のファイル名(.psa、.sa、.saf など)
内容: オブジェクトファイル(.elf)から作成された、デバッグ情報を含まないオブジェクトファイルです。デバッグ情報を含まないため、ROMデータではソースレベルのデバッグは行えません。ターゲットのメモリへプログラム/データをロードするloadコマンド、フラッシュライタ機能のc17 fwlp、c17 fwldコマンドで使用します。

●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ガに読み込まれるファイルです。
c17 stdinコマンドにより指定します。

●[リセット]ボタン用コマンドファイル

ファイル形式: テキストファイル
ファイル名: reset.gdb
内容： コマンドファイルと同等。

●[ユーザコマンド]ボタン用コマンドファイル

ファイル形式: テキストファイル
ファイル名: userdefine.gdb
内容： コマンドファイルと同等。

●[DCLK切替モードON]ボタン用コマンドファイル

ファイル形式: テキストファイル
ファイル名: clkmdon.gdb
内容： コマンドファイルと同等。

●[DCLK切替モードOFF]ボタン用コマンドファイル

ファイル形式: テキストファイル
ファイル名: clkmdoff.gdb
内容： コマンドファイルと同等。

●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: c17_profile.prf
内容: `c17 profile`または、`c17 coverage`コマンド実行時に作成される計測結果データ

●プロファイル・カバレッジデータPATHファイル
ファイル形式: テキスト形式
ファイル名: c17_profile_path.gdb
内容: プロファイル・カバレッジデータファイル(c17_profile.prf)の存在するPATH。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。`c17 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。`c17 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。
`c17 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: c17_profile.prf
内容: c17 profileまたは、c17 coverageコマンド実行時に作成される計測結果データ

●プロファイル・カバレッジデータPATHファイル
ファイル形式: テキスト形式
ファイル名: c17_profile_path.gdb
内容: プロファイル・カバレッジデータファイル(c17_profile.prf)の存在するPATH。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。

10.3 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]>[プロパティー]>[GNU17 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[GNU17 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S5U1C17001H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。

--c17_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。

--c17_double_starting
機能: **二重起動の許可**
説明: 1台のPC上で2つ以上のデバッガの起動はデフォルトで禁止されています。
本オプションを指定すると、これが許可されます。

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: c:¥EPSON¥gnu17¥gdb -x sample.cmd --cd=/cygdrive/d/test/sample

注: • 存在しないオプションを指定するとエラーになります。
エラーメッセージ: ··· gdb : unrecognized option '*XXXXXX*'

• 1台のPC上で2つ以上のデバッガを起動することはデフォルトで禁止されています。

10.3.3 終了方法

●[コンソール]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: ICDを使用してデバッグを行っている場合は(ICD Miniモード時)、ICDの電源を切る前に必ずデバッガを終了させてください。デバッガを実行中にICD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。

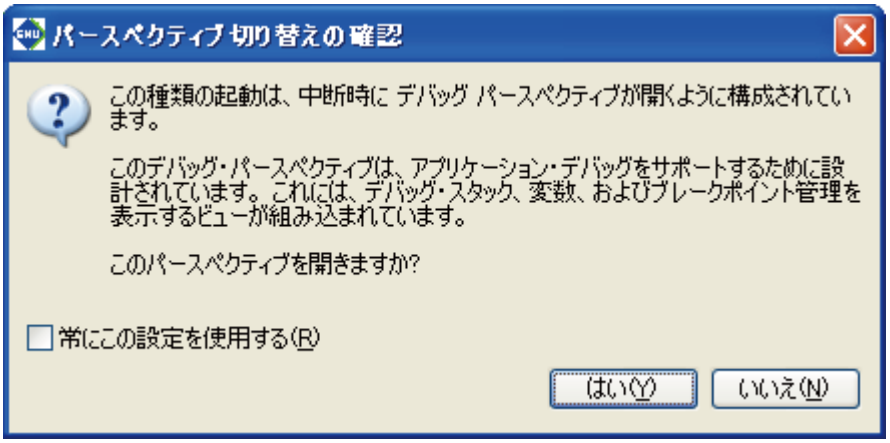
10.4 ウィンドウ

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。

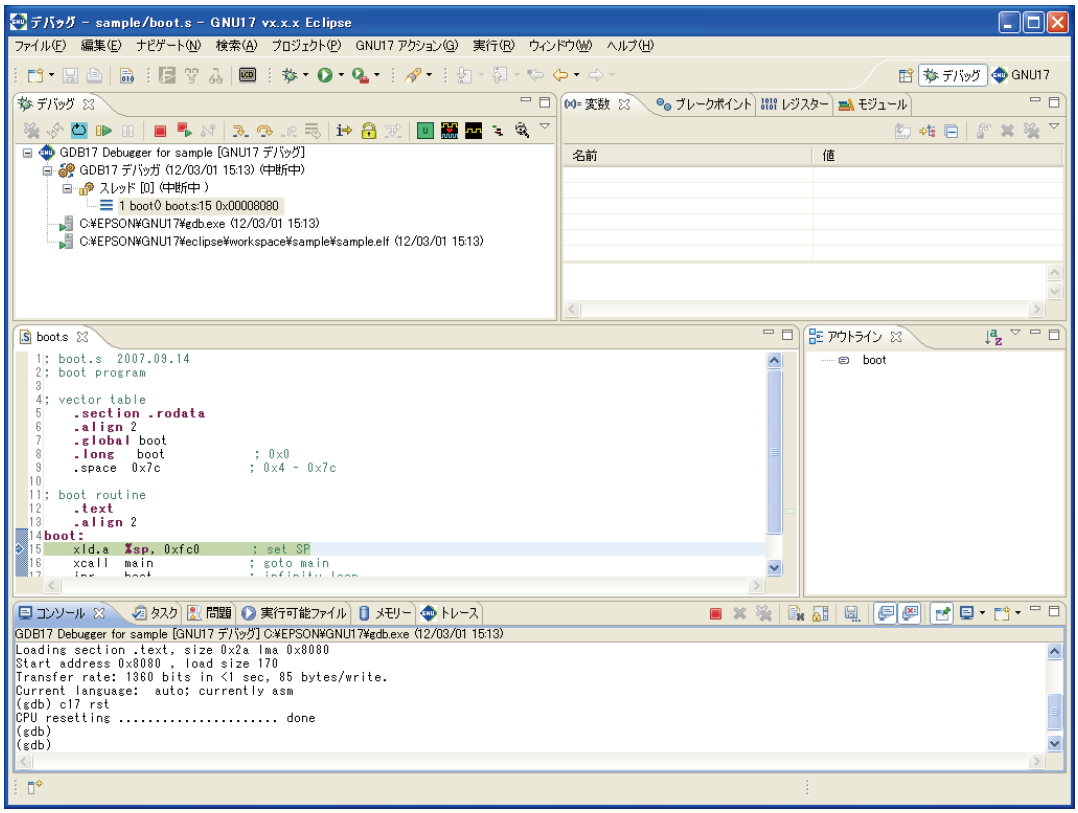


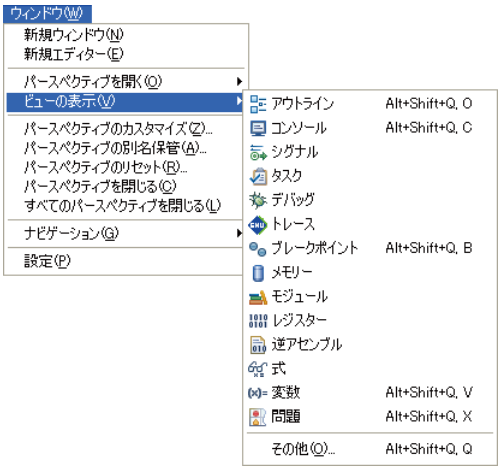
表10.4.1.2.1 デバッグパースペクティブの各ビュー

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。



デバッグ時に使用す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

デバッグ時に使用するビュー	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シミュレーテッド I/O出力
デバッグ	デバッガの開始・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

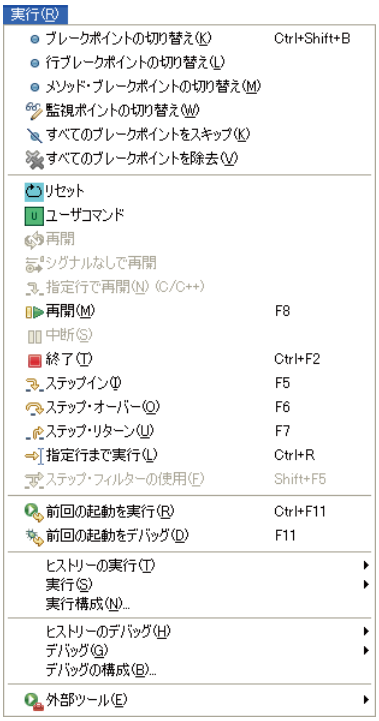


表10.4.1.5.1 ツールバー [実行]メニュー

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	サポートしていません。
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。
リセット	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)
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。
ヒストリーの実行	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。
ヒストリーのデバッグ	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でデバッガの起動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

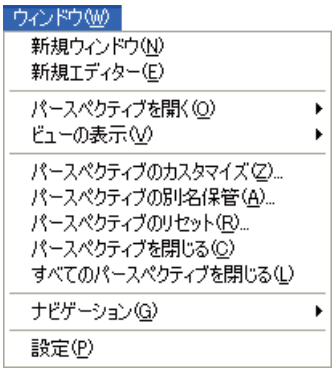


表10.4.1.5.2 ツールバー [ウィンドウ]メニュー

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GNU17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。

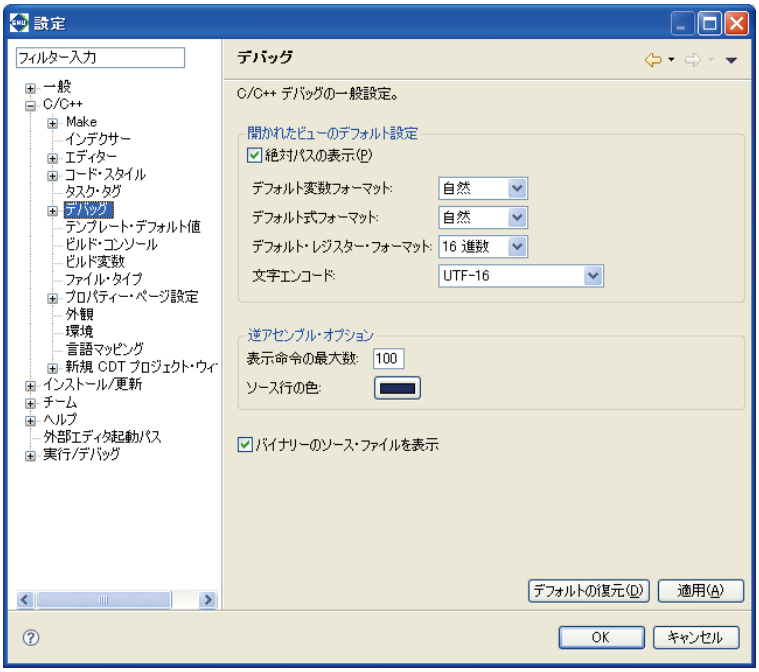


表10.4.1.6.1 [C/C++]>[デバッグ]

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。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。

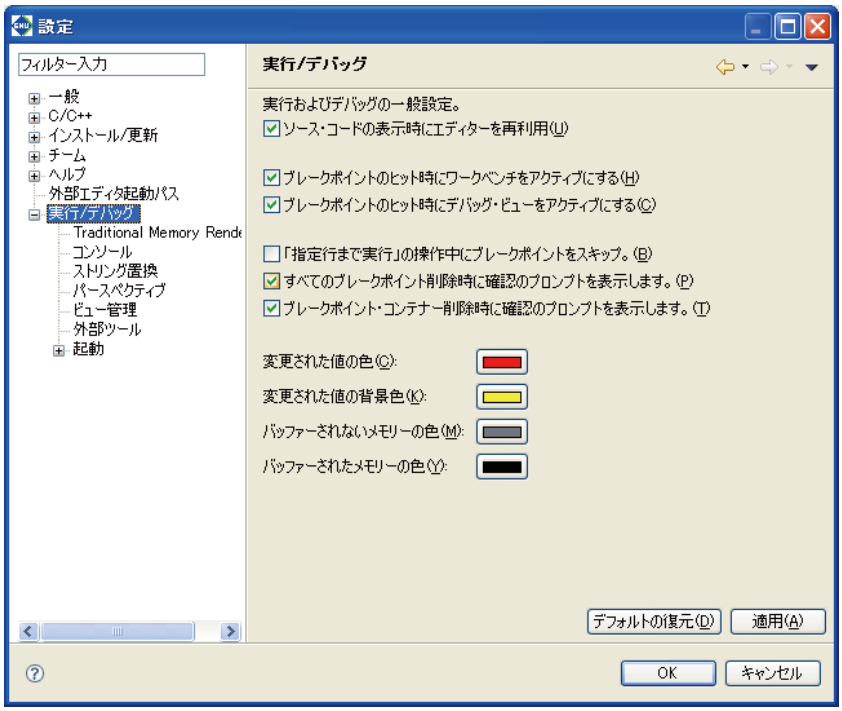


表10.4.1.6.2 [実行/デバッグ]

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 操作しないでください。
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]

デバッガ起動時の挙動に関する設定です。

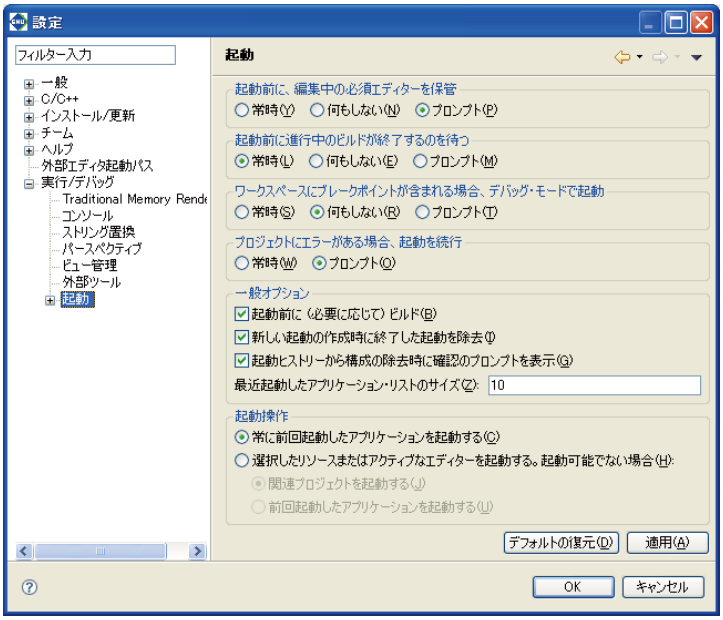


表10.4.1.6.3 [実行/デバッグ]>[起動]

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、編集中で保存していないエディタを保存します。 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)
一般オプション		
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
起動ヒストリー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]の起動ヒストリーの数。 デフォルト：10
起動操作		
常に前回起動したアプリケーションを起動		[F11]キーを押すか[デバッグ]ボタンクリック時、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]キーを押すか[デバッグ]ボタンクリック時、現在選択されているエディタ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 Rendering]

[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。

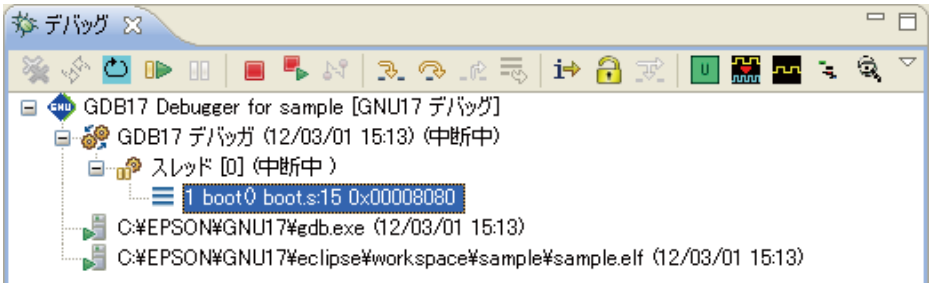
このツリーの以下の設定画面は、操作しないでください。

●[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	終了済みのすべてのデバッグアイコンを削除します。
	再開	サポートしていません。
	リセット	リセットを実行します。
	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	起動中のデバッガ(GDB)を終了後、再起動します。
	切断	サポートしていません
	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)
	フレームにドロップ	サポートしていません。
	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]がアセンブラ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。
	ユーザコマンド	ユーザ定義コマンドを実行します。
	DCLK切替モードON	DCLK(デバッグクロック)切替モードを有効にします。
	DCLK切替モードOFF	DCLK(デバッグクロック)切替モードを無効にします。
	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。
	Flash セキュリティの解除	Flash セキュリティのアクセス制限を解除します。

●コンテキストメニュー

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。
リセット	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。
Flash セキュリティの解除	パスワードが設定されている場合、Flash セキュリティのアクセス制限を解除します。
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。
GDB17 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。

メニュー	機能
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。
プロパティー	サポートしていません。
ユーザコマンド	ユーザ定義コマンドを実行します。
DCLK切替モードON	DCLK(デバッグクロック)切替モードを有効にします。
DCLK切替モードOFF	DCLK(デバッグクロック)切替モードを無効にします。
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

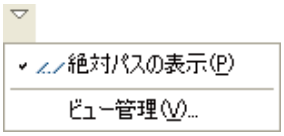


表10.4.2.2.3 ビューメニュー

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。(プロジェクトからの相対パス)
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。

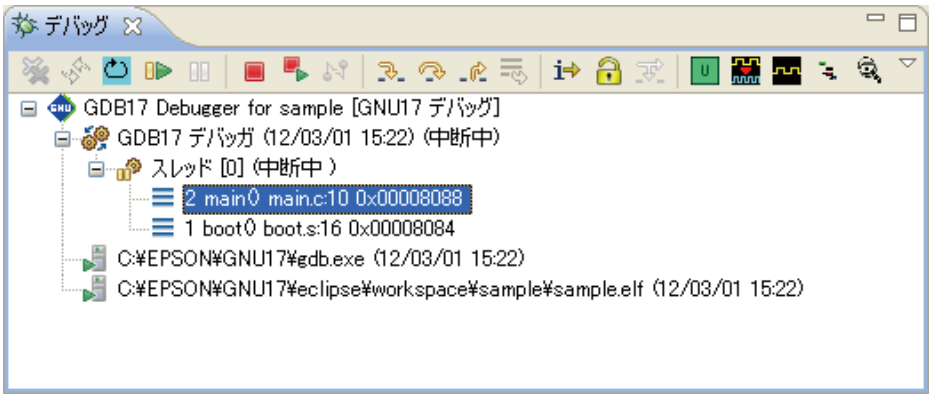


表10.4.2.3.1 表示項目

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。
	デバッガプロセス	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ボタンをクリック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

[リセット]ボタン:

ユーザが編集可能なコマンドファイル(¥gnu17¥reset.gdb)を実行します。
出荷時にはコマンドファイルには

```
c17 rst
```

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

C17レジスタ	初期値
R0-R7	0x0
PC	ブートアドレス
PSR	0x0
SP	0xffffffc

注:コネクトモードによりc17 rstコマンドの動作が異なります。

- ICD Miniモード
  S1C17チップのTTBRのアドレス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
  [プロジェクト]>[プロパティー]>[C17 GDBコマンド]ダイアログの[雛形からコマンドを生成する]ボタンで[初期起動コマンド生成]ダイアログを開き、[ブートベクタアドレス]項目で指定したアドレスによりブートアドレスが決定します。
  コマンドファイル内では、

  c17 ttbr ブートベクタアドレス

  という行が生成されます。

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
- 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。

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。

Flash セキュリティ解除

[Flash セキュリティ解除]ボタン：

Flash セキュリティのアクセス制限を解除します。
ターゲットCPUがFlash セキュリティ対応機種のとき、ボタンは有効となります。
Flash セキュリティのアクセス制限を解除すると、メモリビューでFlash セキュリティメモリを表示することができます。
また、パスワードを設定していない状態ではエラーになります。
パスワードの設定方法については、"5.7.10 フラッシュプロテクトの設定"を参照してください。

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]ボタンをクリック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

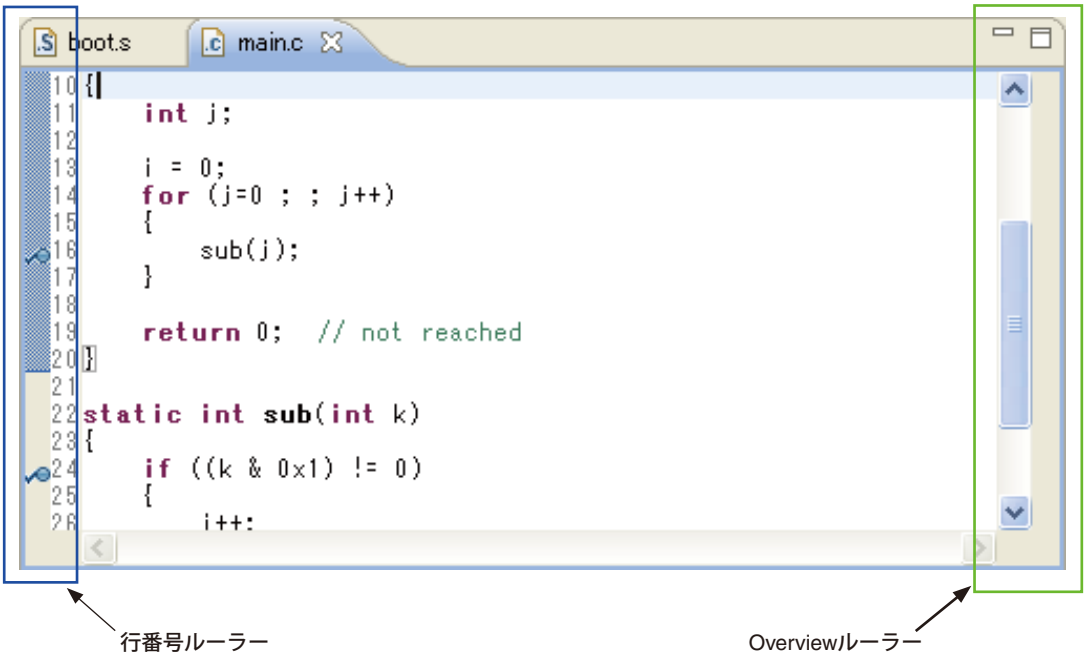
- スタックフレームは最大99個までしか表示されません。
- 以下の場合、スタックフレームが正常に表示できません。
  - RAM上のプログラムをロード後に、この領域の内容を書き換えた場合
  - アセンブラ上で、現PCから上位アドレス方向4096命令までの間でret/ret.d/reti/reti.d命令が無い場合
  - PCリセット直後
- コマンドファイル内や[ユーザコマンド]ボタン([DCLK切替モードON]ボタン、[DCLK切替モードOFF]ボタン)で実行する「コマンドファイル(¥gnu17¥userdefine.gdb、clkmdon.gdb、clkmdoff.gdb)」内に、実行コマンド(next / nexti / finish / until / continueなど)やquitコマンドが含まれており、ブレークポイントにヒットしない場合や無限ループ等で停止しないプログラムの場合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などプログラムが停止するように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GNU17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。

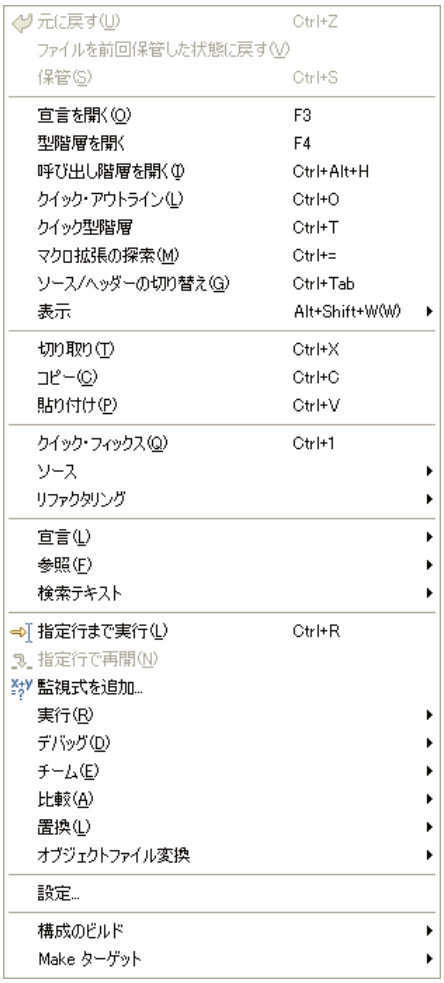


[ソース]エディターは、中央のエディター領域のほかに、左側の行番号ルーラーと右側のOverviewルーラーからなり、行番号ルーラーにブレークポイントを貼ることができます。

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



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ソースコードを表示するには、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

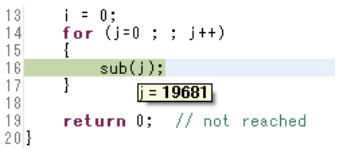
行番号ルーラーにブレークポイントを貼ることができます。
行ルーラーでは、ブレークポイントの状態および種類を表示します。

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。
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。 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。 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。
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。



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。
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ボタンをクリックして閉じることもできます。

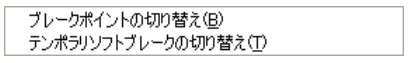
注:エディターでは、通常の編集操作が可能ですが、デバッグ中には編集しないでください。デバッグ中に編集してもデバッグ中の実行ファイルはリロードされませんので、プログラムの停止位置とソースファイルの行位置があわなくなります。

●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。

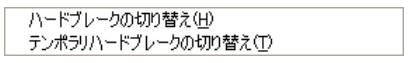
- **行ルーラーをダブルクリック**
  ブレークポイントを貼りたい位置で行ルーラーをダブルクリックすると、ソフトウェアPCブレークポイント が設定されます。設定されている位置で再度ダブルクリックすると、ブレークポイントが解除されます。
  ダブルクリックでブレークポイントを設定できるのは、ソフトウェアPCブレークポイントのみです。ダブルクリックによるブレークポイント解除は、テンポラリソフトウェアPCブレークポイント、ハードウェアPCブレークポイント、テンポラリハードウェアPCブレークポイントでも可能です。

- **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]**

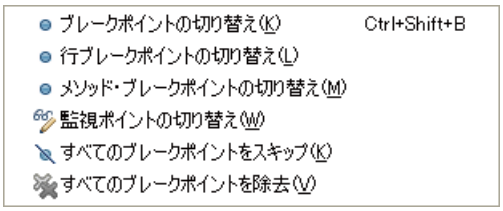


ブレークポイントを貼りたい位置で行ルーラーのコンテキストメニューからハードウェアPCブ

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。

・[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("{" ～ "}"の間)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、関数プロロー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
ブレークポイントは、デバッガ起動前・起動中いずれの時点でも設定可能です。
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。
また、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。
1. デバッガ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
デバッガ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、デバッガに対して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。
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
エラーが発生して使用可能でない場合は、チェックのオーバーレイがつきません。 (未解決)
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。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、"1. デバッガがすでに起動しているときにブレークポイントを貼る" 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。

- 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：機種により最大1 ～ 4カ所
    （SIMモードでは１箇所のみ)

  コマンド入力で設定した分も併せ、これらを超えるとエラーになります。
  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。
  デバッガが起動前であればソースファイル上には設定することができます。
  また、ソフトウェアPCブレークポイントは外部ROM上のアドレスには設定できません。

●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：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場合は有効状態で設定されます。
  無効：ブレークポイントは設定されているが、一時的にブレークポイントで停止させたくない場合に使用します。
- 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。

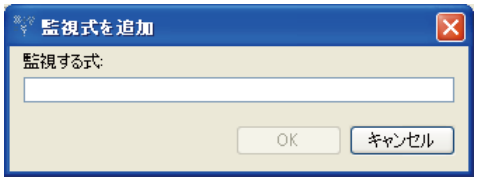
●指定位置までのプログラム実行

[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。

●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]ボタンをクリックすると[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]
変数値バルーン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

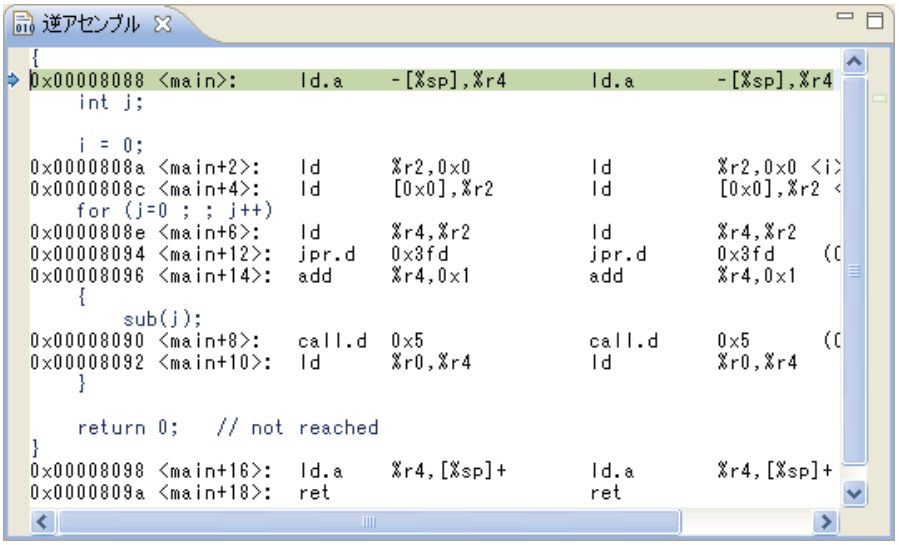
操作	操作内容
[ブレークポイント]ビュー表示	コマンドから設定したブレークポイントは[ブレークポイント]ビューで表示されない場合があります。
[ブレークポイントの切り替え]	ソフトウェアPCブレークポイントを解除する。
[テンポラリソフトブレークの切り替え]	テンポラリソフトウェアPCブレークポイントを解除する。
[ハードブレークの切り替え]	ハードウェアPCブレークポイントを解除する。
[テンポラリハードブレークの切り替え]	テンポラリハードウェアPCブレークポイントを解除する。
[行ブレークポイントの切り替え]	ソフトウェアPCブレークポイントを解除する。
[メソッド・ブレークポイントの切り替え]	関数ブレークポイントを解除する。
[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・プロパティー]	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

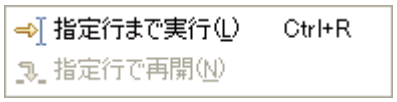


表10.4.4.2.1 コンテキストメニュー

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

デバッガ起動中、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ソースコードを表示するには、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ことができます。

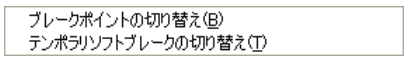
●ブレークポイント設定/解除

[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

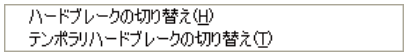
ブレークポイントを貼りたい位置で行ルーラーをダブルクリックすると、ソフトウェアPCブレークポイント  が設定されます。設定されている位置で再度ダブルクリックすると、ブレークポイントが解除されます。
ダブルクリックでブレークポイントを設定できるのは、ソフトウェアPCブレークポイントのみです。ダブルクリックによるブレークポイント解除は、テンポラリソフトウェアPCブレークポイント、ハードウェアPCブレークポイント、テンポラリハードウェアPCブレークポイントでも可能です。

- 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]



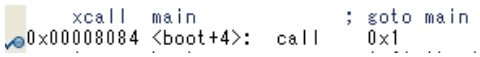
ブレークポイントを貼りたい位置で行ルーラーのコンテキストメニューからソフトウェアPCブレークポイントまたはテンポラリソフトウェアPCブレークポイントが設定できます。
テンポラリソフトウェアPCブレークポイントは、1回のヒットのみ有効なブレークポイントです。
設定されている位置で再度同じメニューを選択すると、ブレークポイントが解除されます。

- 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]



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、デバッガ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ソース行には設定できません。



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 xxx` × 設定不可
- 遅延命令の行(遅延分岐命令の次の行)
  例)
  `jpr.d xxx` ○ 設定可
  `cmp` × 設定不可

ブレークポイントを貼ると、デバッガに対して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。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 ：最大200 ヵ所(テンポラリ含む)
  -ハードウェアPCブレーク ：機種により最大1 ～ 4 ヵ所
  (SIMモードでは 1 箇所のみ)
- 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： 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場合は有効状態で設定されます。

無効： ブレークポイントは設定されているが、一時的にブレークポイントで停止させたくない場合に使用します。

・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・プロパティー

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・プロパティー]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。

"10.4.5.4"の「●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。

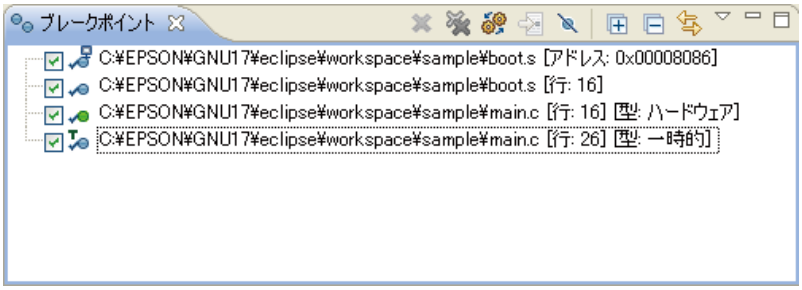
表10.4.4.5.1 制限事項一覧

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。
[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・プロパティー]	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

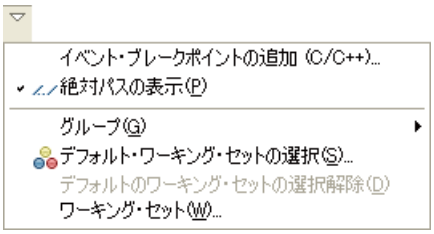


表10.4.5.2.2 ビューメニュー

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

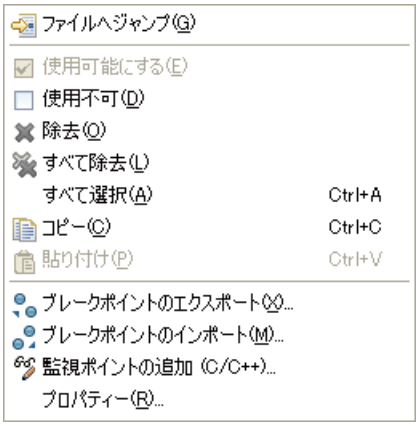


表10.4.5.2.3 コンテキストメニュー

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	ブレークポイントをコピー /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...	サポートしていません
プロパティー ...	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ガ起動時に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 (`info break`コマンドの結果が返るもの)

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 GDB上の状態	ブレークする
	○	×	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	×

注：・ GDB上は、解決されたブレークポイントに対して、有効・無効を設定可能です。
・ GDBがターゲットプログラムを実行したときに、ブレークポイントは実際にターゲットプログラムに設定されます。

●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。

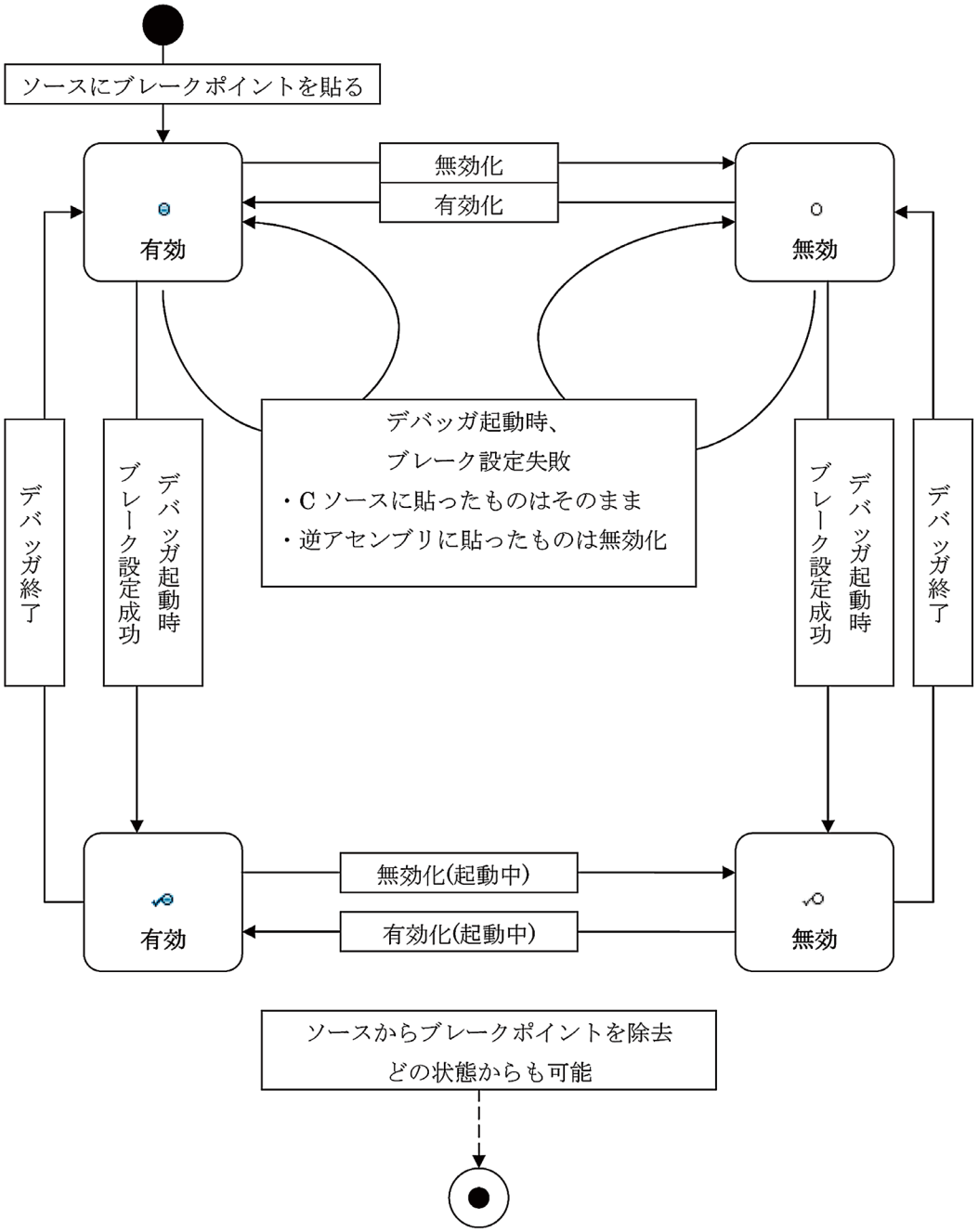


図10.4.5.3.1 ブレークポイントの状態遷移

●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

デバッガが起動していなく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。

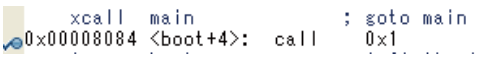
ブレークポイントは、どの行に対しても設定可能です。ただし、コメントや空行に貼った場合に実際に停止する位置は、下方向の直近の命令の位置です。自動的に位置補正はされません。

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

デバッガ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ソース行には設定できません。



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 xxx      × 設定不可
```

- **遅延命令の行（遅延分岐命令の次の行）**

例）

```
jpr.d xxx   ○ 設定可
cmp         × 設定不可
```

- **デバッガ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

デバッガ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、デバッガ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）がつきません。
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なおす必要があります。

・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ーグのあと(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貼れますが、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。 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
ファイル名		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。
設定箇所		ブレークポイントの種類によって、設定箇所の情報を表示します。 プログラムは実行中にこれらの箇所に到達すると、次の命令を実行する直前でブレークします。
	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。

●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。

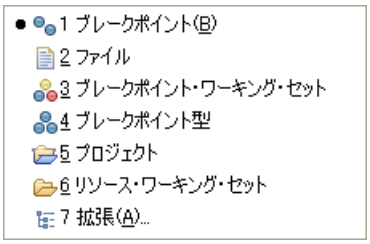


表10.4.5.4.2 グループ表示項目

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：ブレークポイントの一覧表示は、ブレークポイントを貼った順と異なります。

10.4.5.5 操作

●ビューを開く/閉じる

[ブレークポイント]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×ボタンをクリックして閉じることができます。

●ソフトウェア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。

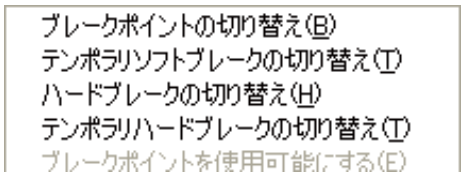
[ソース]エディター

- 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]
- [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]

[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]

例) 行ルーラーのコンテキストメニュー



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：ソフトウェアPCブレークポイントの設定数には上限があります。最大200カ所(テンポラリ含む)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。
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。

●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。機種により最大1 ～ 4カ所まで設定可能です。(SIMモードでは１箇所のみ)
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超えて設定することができます。

●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、ダイアログを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 >[プロパティー]
- [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ー ...]
- [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ー ...]

ツリーの[共通]を選択すると、以下の情報を表示します。

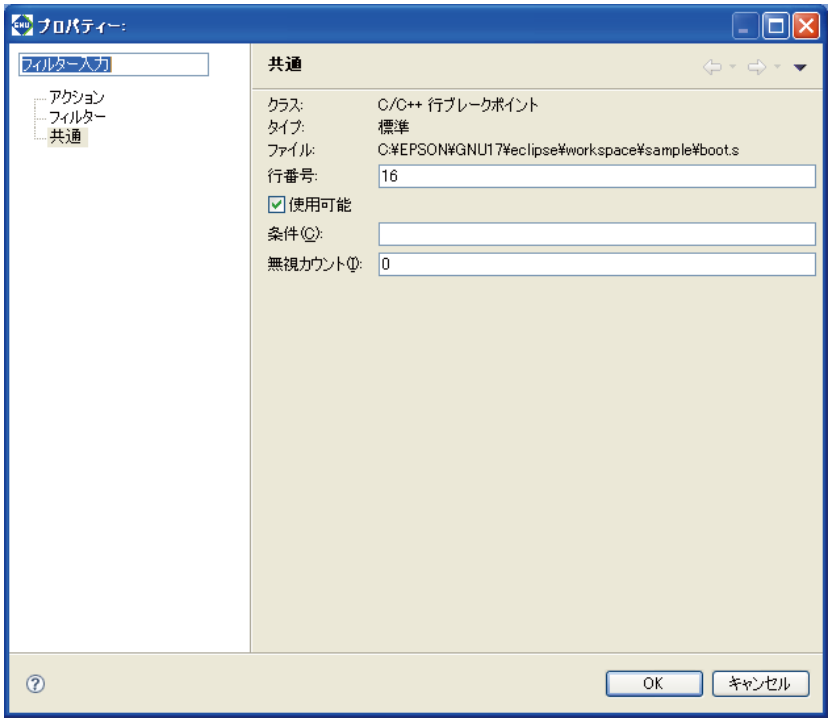


表10.4.5.5.1 ブレークポイントのプロパティー

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) 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)
クラスによる表示	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)
	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)
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

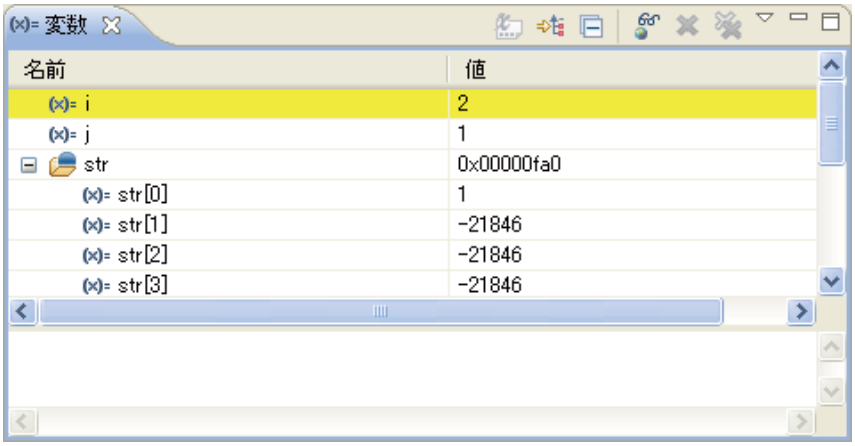
- コマンドからブレークポイント命令を実行した場合、ブレークポイント一覧に正しく反映されません。ブレークポイントは、画面上から設定してください。
- デバッガを終了しても、プロジェクトのブレークポイント情報は[ブレークポイント]ビューに残っています。この状態で、別プロジェクトのデバッガを起動すると、前回のプロジェクトのブレークポイントが設定されます。
  ただし、この場合、[ソース]エディター上にはブレークポイントが表示されません。前回のプロジェクトのブレークポイントが設定されないようにする為には、ツールバーの [すべてのブレークポイントを除去] ボタンなどから前回のブレークポイントの設定を削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ガ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

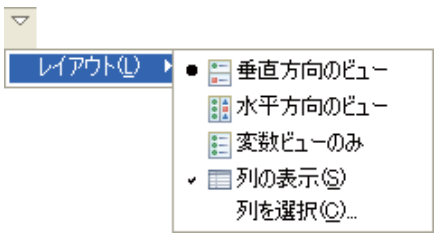


表10.4.6.2.2 ビューメニュー

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

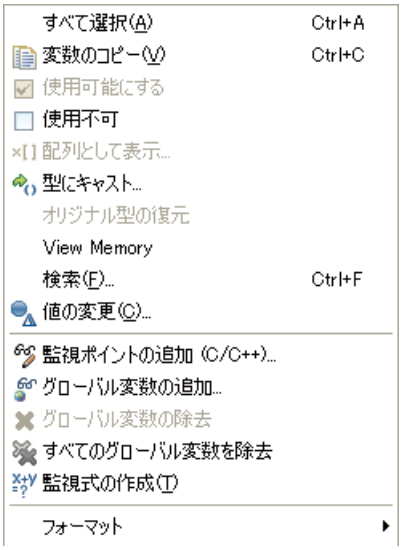


表10.4.6.2.3 コンテキストメニュー

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...		サポートしていません。
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16ビットのため、`iSymbol 0001101001010101` 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(8進数/10進数/16進数)が変更することができます。

注：2進数を設定した場合は、値が正しく表示されません。

●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"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ことができます。

●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]ボタンをクリックします。
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、デバッガ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

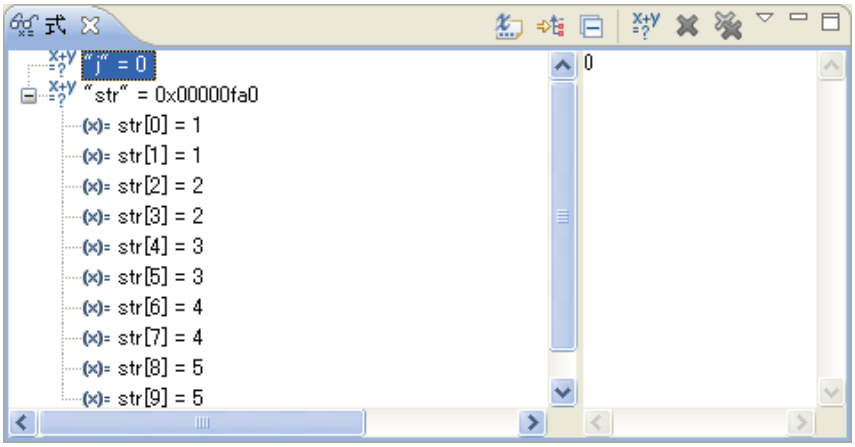
- [デバッグ]ビュー上のスタックフレーム選択時に、ローカル変数が複数個表示されることがあります。これは、以下のいずれかによります。
  - 最適化オプションあり(-O0以外)でビルドした場合
  - ローカル変数がレジスタに割り当てられている場合

  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。
- Cコンパイラの最適化により、レジスタに割り当てられたローカル変数は、値が正しく表示されないことがあります。この場合は一度グローバル変数に代入した上で、グローバル変数の値を[式]ビューから確認してください。

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	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	表示している監視式一覧をたたみます。
	新規監視式を作成	監視式を追加します。
	選択された式の除去	選択中の式を表示から削除します。
	すべての式を除去	すべての式を表示から削除します。

●ビューメニュー

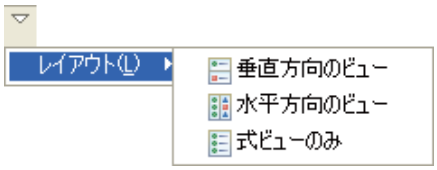


表10.4.7.2.2 ビューメニュー

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)

・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)

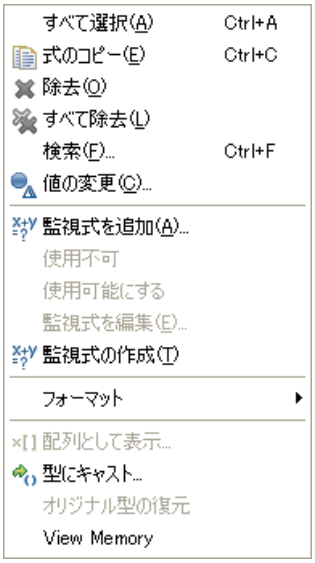


表10.4.7.2.3 コンテキストメニュー

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	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
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16ビットのため、`iSymbol 0001101001010101` 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(8進数/10進数/16進数)が変わります。

注：2進数を設定した場合は、値が正しく表示されません。

●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ことができます。

●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。(このとき、改行は不要です)
  [[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(値更新が行われます。)
  [OK」ボタンをクリックしますと、[式]ビューに式が登録されます。
  [キャンセル]ボタンをクリックしますと、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。(このとき、改行は不要です)
[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(値更新が行われます。)
[OK」ボタンをクリックしますと、[式]ビューに式が登録されます。
[キャンセル]ボタンをクリックしますと、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**を終了したあとで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]ボタンをクリックします。

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10.4.7.3.2 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。

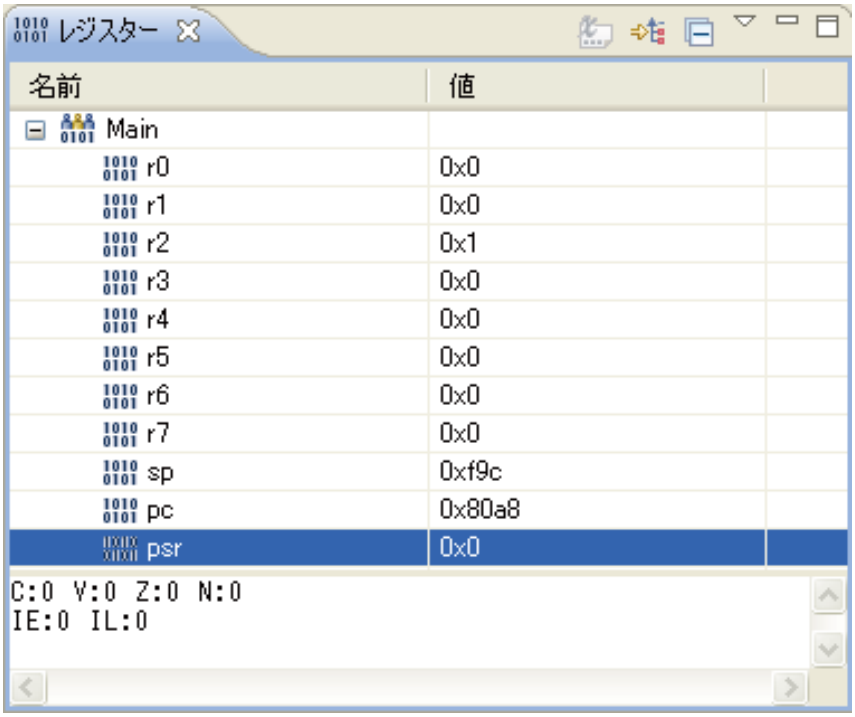
10.4.7.5 制限事項

- [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	サポートしていません。
論理構造の表示	サポートしていません。
すべて縮小表示	表示しているレジスター覧をたたみます。

●ビューメニュー

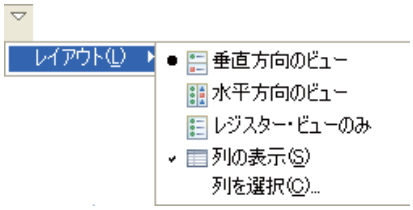


表10.4.8.2.2 ビューメニュー

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。
列を選択...		サポートしていません。

●コンテキストメニュー

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

[レジスター]ビューは、デバッグ中のC17コアに従って以下のCPUレジスタの内容を一覧表示します。

●レジスター覧

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。(表形式=黄色、表形式でないとき=赤)

表10.4.8.3.1 各CPUのレジスター覧

コア	レジスタ種類
C17	r0–r7, psr, sp, pc

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

[レジスター]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、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	レジスタ=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"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。

●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(8進数/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。
レジスター・ビューのみ	詳細ペインを表示しません。

注：2進数を設定した場合は、値が正しく表示されません。

●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
C17	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

各フラグの値の表示方法

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×ボタンをクリックして閉じることができます。

●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10.4.8.4.1 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]ボタンをクリックします。

[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24ビットが有効となり、25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。

[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）

. [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

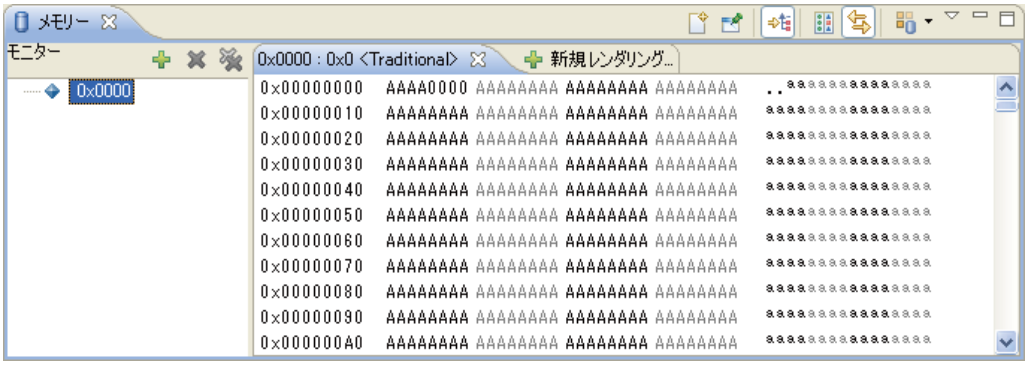
・ レジスタの内容をコマンド等で修正しても表示を更新しません。
・ デバッガ起動直後[レジスター]ビューに初期表示されるレジスタは、ビューの表示領域によって変わります。このとき[すべて選択]の操作を行うと、現在表示されているレジスタのみが選択されます。
一度全レジスタを[レジスター]ビューで表示すると、[すべて選択]の操作ですべてのレジスタが選択されます。

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
新規メモリー・ビュー	新規に[メモリー]ビューを開きます。
メモリー・モニターをピン留め	[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。
メモリー・モニター・ペインを切り替え	メモリモニタ（左）ペインの表示/非表示を切り替えます。
分割ペインを切り替え	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。
メモリーレンダリング・ペインをリンク	サポートしていません。
メモリー・モニターの切り替え	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

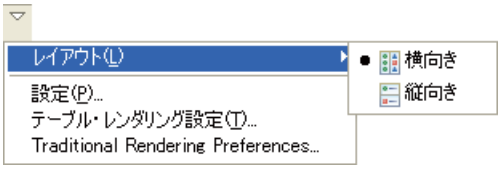


表10.4.9.2.2 ビューメニュー

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

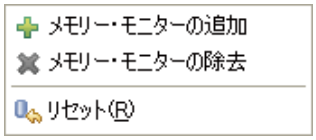


表10.4.9.2.3 コンテキストメニュー(メモリモニタ)

メニュー	機能
メモリー・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。
メモリー・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

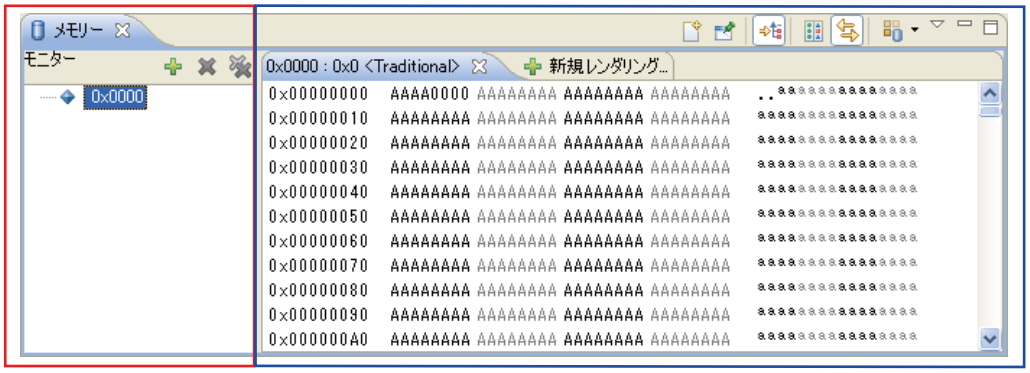
表10.4.9.2.4 コンテキストメニュー(メモリレンダリング)

メニュー		機能
監視ポイントの追加...		サポートしていません。
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(デフォルト:表示)
	Binary	データ部(デフォルト:表示)
	Text	ASCII部(デフォルト:表示)
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	リトルエンディアン(デフォルト)
Text		ASCII部のエンコーディングを切り替えます。
	ISO-8859-1	ISO-8859-1で表示します。(デフォルト)
	US-ASCII	US-ASCIIで表示します。
	UTF-8	UTF-8で表示します。
Cell Size		データ部の列ごとの表示バイト数を切り替えます。
	1	1バイトで表示します。
	2	2バイトで表示します。
	4	4バイトで表示します。(デフォルト)
	8	8バイトで表示します。
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。
	Hex	16進(デフォルト)
	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。（デフォルト）
	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部に分かれます

[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ことができます。

●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✚アイコンをクリックすると[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]ボタンをクリックすると、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ビットが有効となり、33ビット目以上は 無視されます。

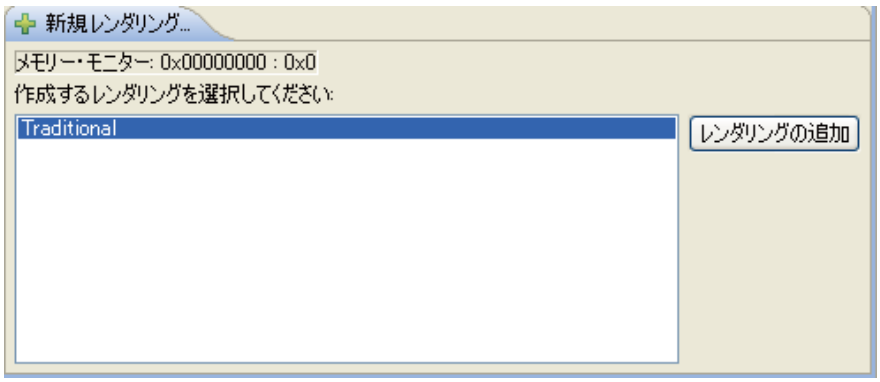
0x1122334455667788と入力した場合は、0x55667788の値を指定したものとされます。

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。

[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]ボタンをクリックします。



[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]をクリックすると該当するメモリ表示を削除できます。

[すべて除去]をクリック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]ボタンをクリックすることで、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。

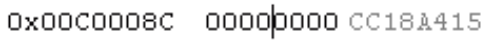
●メモリデータの変更

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。入力は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）

4. 値を確定する場合は、[Enter]キーを押します。
   確定された値は、背景色が変わります。（赤）

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]キーを押します。



値入力時は、以下のキーは使用できません。

- [Backspace]キー
- [Delete]キー
- [Insert]キー

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。デバッガ起動時は前回の設定で[メモリー]ビューを表示します。ただし、PanesとEndianはデフォルトの設定で表示されます。

表10.4.9.4.1 表示形式一覧

表示形式		機能
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（デフォルト:表示）
	Binary	データ部（デフォルト:表示）
	Text	ASCII部（デフォルト:表示）
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	リトルエンディアン（デフォルト）
Text		ASCII部のエンコーディングを切り替えます。
	ISO-8859-1	ISO-8859-1で表示します。（デフォルト）
	US-ASCII	US-ASCIIで表示します。
	UTF-8	UTF-8で表示します。
Cell Size		データ部の列ごとの表示バイト数を切り替えます。
	1	1バイトで表示します。
	2	2バイトで表示します。
	4	4バイトで表示します。（デフォルト）
	8	8バイトで表示します。
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。
	Hex	16進（デフォルト）
	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ます。(デフォルト)
	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]

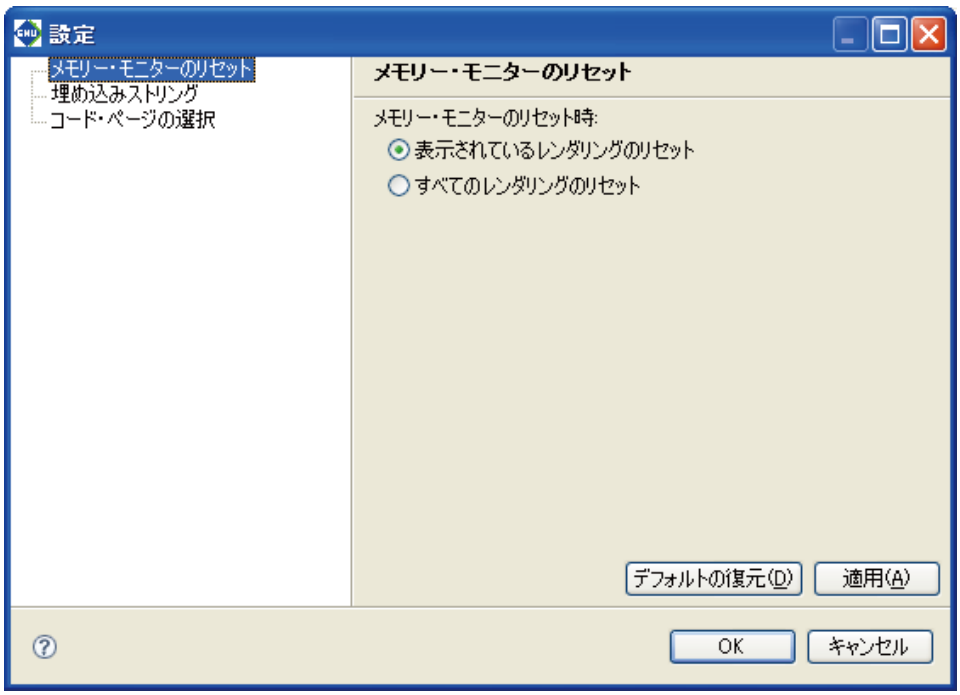


表10.4.9.4.2 [メモリー・モニターのリセット]ダイアログ

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]

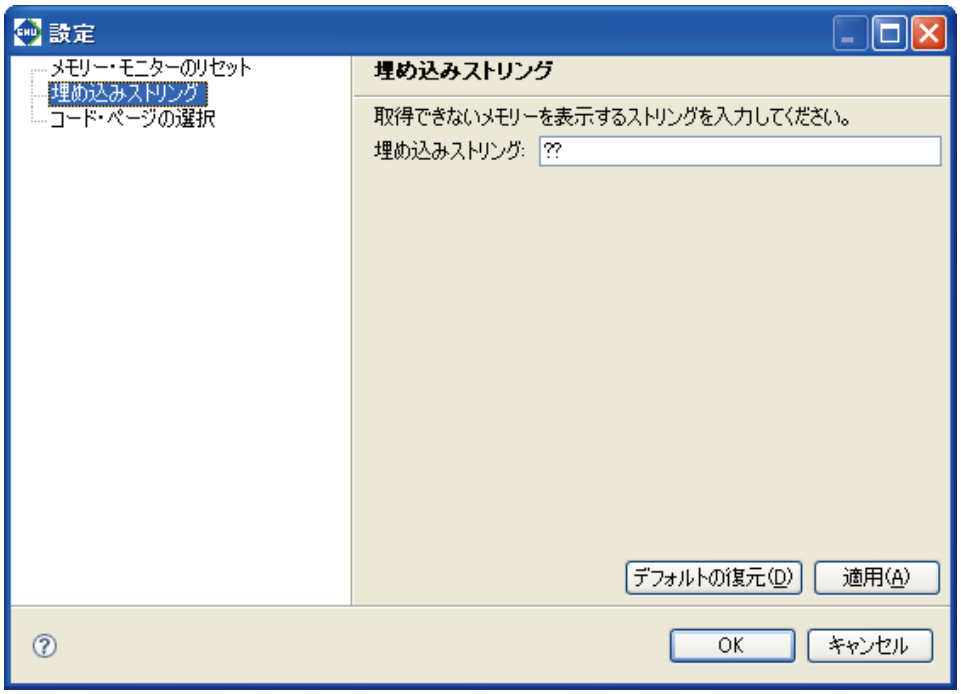


表10.4.9.4.3 [埋め込みストリング]ダイアログ

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 Readering]

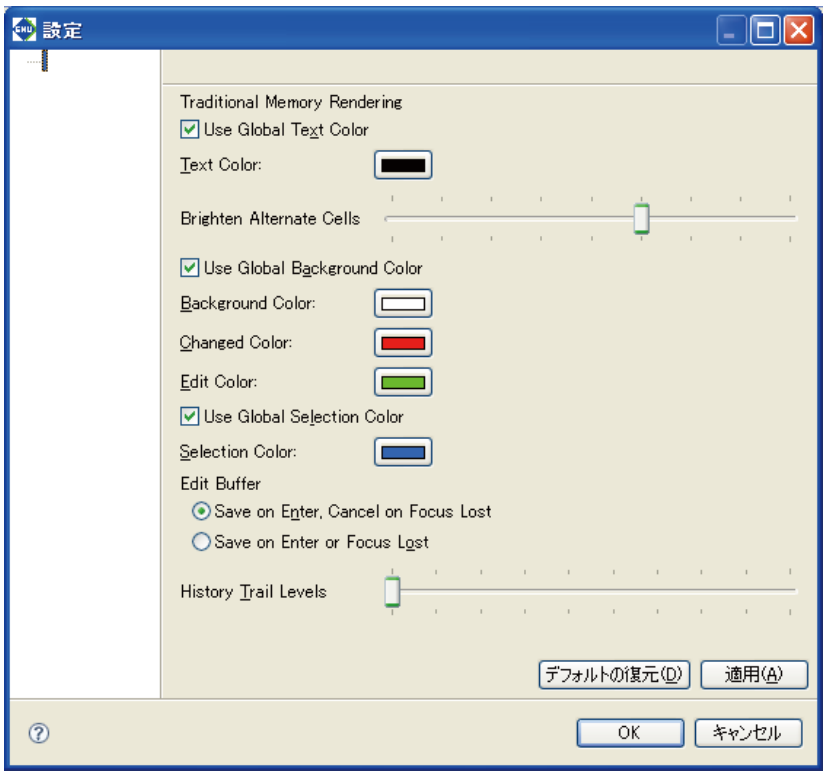


表10.4.9.4.4 [Traditional Memory Readering]ダイアログ

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。

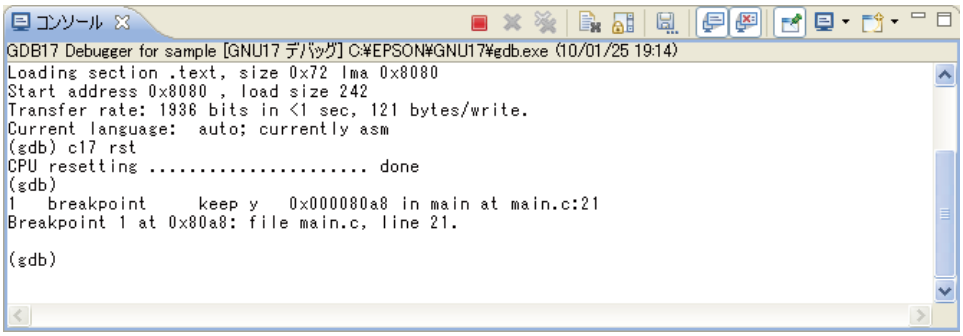
10.4.9.5 制限事項

・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

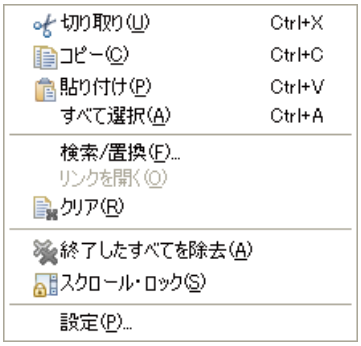


表10.4.10.2.2 コンテキストメニュー

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ことができます。

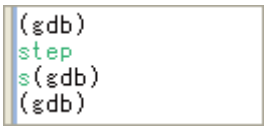
注：[コンソール]ビューを閉じてしまうと、コマンド入力ができなくなります。
[コンソール]ビューを閉じてしまった場合は、[ウィンドウ]>[ビューの表示]>[コンソール]から開き、[デバッグ]ビューのデバッガプロセス(gdb.exe)のアイコンをクリックし、[コンソールのピン留め]ボタンをクリックしてからコンソールをピン留めしてください。
デバッガ起動直後に(gdb)のコンソールが表示されていない場合は、一旦[コンソール]ビューの[コンソールのピン留め]からピン留めをはずし、[デバッグ]ビューのデバッガプロセス(gdb.exe)のアイコンをクリックし、[コンソールのピン留め]ボタンをクリックして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はできません。
・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。)
・コンソールプロンプト"(gdb)"、commandsコマンドのプロンプト">"の前に文字が入った場合は[Enter]キーを押してください。

例1)"(gdb)"プロンプト



例2)">"プロンプト

([Backspace]キーで文字を削除した場合、コンソールが入力できなくなります。この場合、コンソールビュー右上の[終了]ボタン ■ でデバッガを終了してください。)

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
	info breakpoints
ステップ実行関連	step
	stepi
	next
	nexti
	finish
	continue
	until
CPUリセット関連	c17 rst

●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]：

デバッガ(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、終了したデバッガ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]

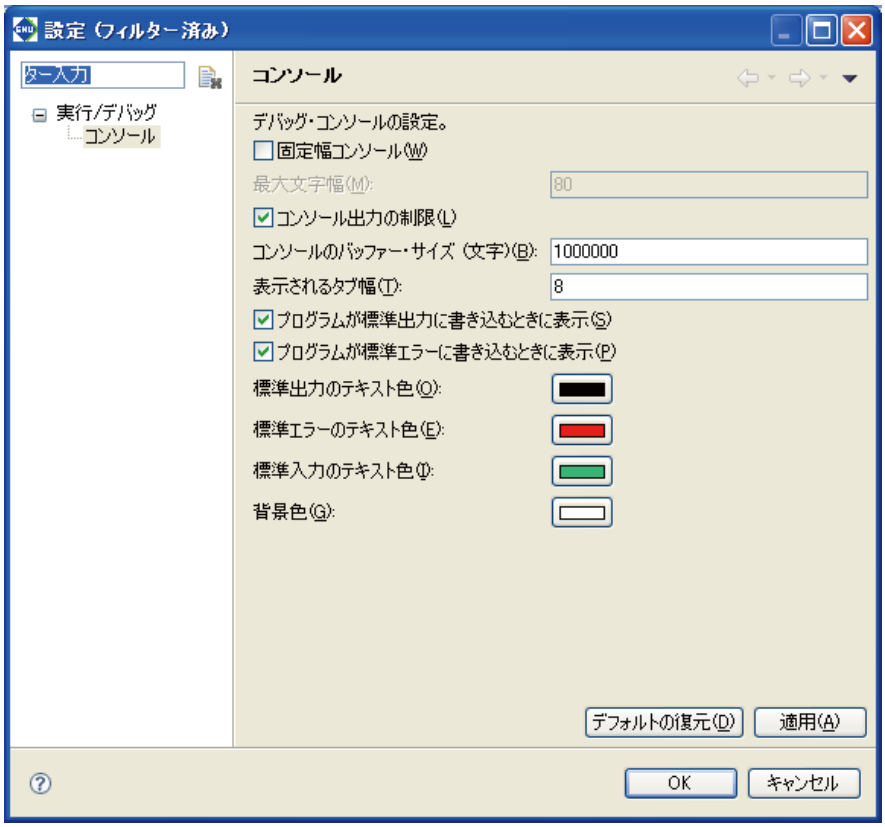


表10.4.10.4.2 [コンソール]設定ダイアログ

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。

設定		設定内容
	コンソールのバッファー・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。

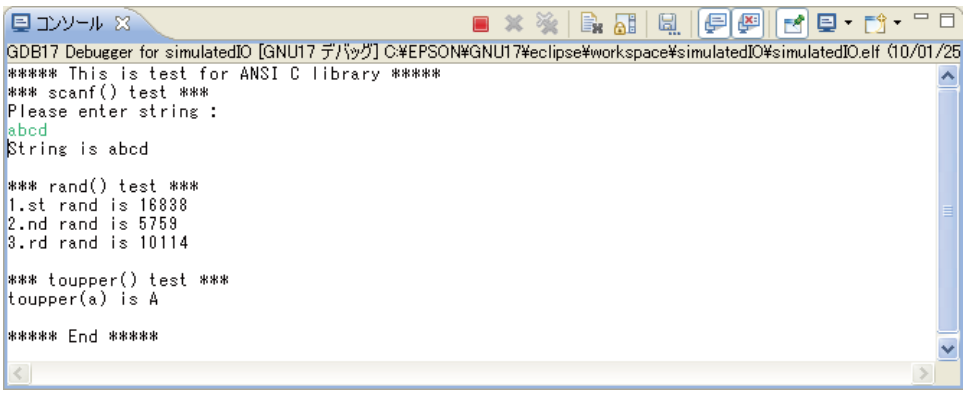
10.4.10.5 制限事項

・最大出力文字数は、[コンソールのバッファー・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(elf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。
シミュレーテッドI/O機能の入力は、[コンソール]ビューにキーボードで入力します。
キーボードからの入力を行いたい場合は、**c17 stdin**コマンドの引数にファイル名を指定しないで下さい。
ファイルを使用した入力を行いたい場合は、**c17 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への出力内容とキーボード入力の内容を表示します。

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]：

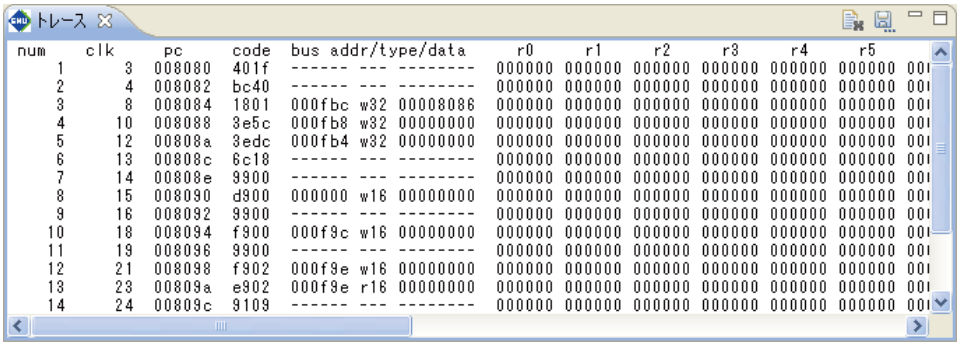
それぞれのボタンを押し込むことで、標準出力もしくは標準エラー出力が書き込まれたときに[コンソール]ビューにフォーカスが移るようになります。

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。

10.4.12 [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

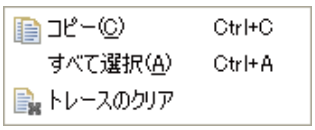


表10.4.12.2.2 コンテキストメニュー

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のときでデバッグ中に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：・ 一度に出力できるトレースデータは、"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。

- ICD Miniモードにはトレース機能はありません。

●PCトレース

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。

- トレース番号
- クロック数
- PC値と命令コード
- バス情報(アドレス、R/Wおよびアクセスサイズとデータⅰ)
- レジスタ値(R0 ～ R7、SP)
- PSR値(IE、IL、CVZN)
- 逆アセンブル内容とソース
- クロック数の積算表示指定(0の場合は命令個別のクロック数表示)

`c17 tm` コマンドを設定時に、上記の内、どの情報を表示するかを設定します。

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ことができます。

●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。

10.5 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(←、→)、[Backspace]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/gnu17/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] - [実行] 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]	
`c17 pwul * *`	[デバッグ]	コンテキストメニュー [Flash セキュリティの解除]	
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]	
`c17 chgclkmd 0 *`	[デバッグ]	コンテキストメニュー [DCLK切替モードON]	
`c17 chgclkmd 1 *`	[デバッグ]	コンテキストメニュー [DCLK切替モードOFF]	
`c17 rst *`	[デバッグ]	[実行] - [リセット] コンテキストメニュー [リセット]	
`c17 profile`	[デバッグ]	コンテキストメニュー [プロファイル]	
`c17 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ポイントの切り替え]	–
`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 [ハードブレークポイントの切り替え]	–
`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ポイントの切り替え]	–
`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]	–

コマンド	ビュー	メニュー /その他	ボタン
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¥gnu17¥userdefine.gdbを実行します。
出荷時のuserdefine.gdb

```
#Edit user command
#c17 rst                (何も実行しません)
```

コンテキストメニュー [DCLK切替モードON]

コマンドファイル¥gnu17¥clkmdon.gdbを実行します。
出荷時のclkmdon.gdb

```
#DCLK(Debug Clock) change mode ON
c17 chgclkmd 0          (DCLK(デバッグクロック)切替モードを有効にします)
```

コンテキストメニュー [DCLK切替モードOFF]

コマンドファイル¥gnu17¥clkmdoff.gdbを実行します。
出荷時のclkmdoff.gdb

```
#DCLK(Debug Clock) change mode OFF
c17 chgclkmd 1          (DCLK(デバッグクロック)切替モードを無効にします)
```

[実行]-[リセット]

コマンドファイル¥gnu17¥reset.gdbを実行します。
出荷時のreset.gdb

```
c17 rst                 (CPUをリセットします)
```

注: 実際のコマンド入力に比べ、パラメータの指定が行えないなどの制限があります。

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

コマンドは1行にひとつのみ記述可能です。
例:

```
c17 rpf c17.par                 メモリマップ情報の設定
file sample.elf                 デバッグ情報の読み込み
target sim                      ターゲットの接続
load                            プログラムのロード
c17 rst                         リセット
c17 stdout 1 WRITE_FLASH WRITE_BUF stdout.txt      stdoutの設定
c17 stdin 1 READ_FLASH READ_BUF stdin.txt          stdinの設定
break _exit                     ソフトウェアPCブレークポイントの設定
cont                            プログラムの実行
c17 stdout 2                    stdoutの解除
c17 stdin 2                     stdinの解除
```

●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**
   デバッガの起動コマンドで-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ガの起動時に実行させることができます。
   例: `c:¥EPSON¥gnu17¥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

--c17_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。--c17_cmwオプションを指定せずにデバッガを起動した場合、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)
c17 log test.log   ロギング開始
        :          ログモード
(gdb)
c17 log            ロギング終了
```

`c17 log`コマンドでロギング開始後は、次に`c17 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2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。

●ICD Miniモード

ICD Mini(S5U1C17001H)またはICDボード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ます。

注：S5U1C17001Hのファームウェアは最新のものをお使いください。

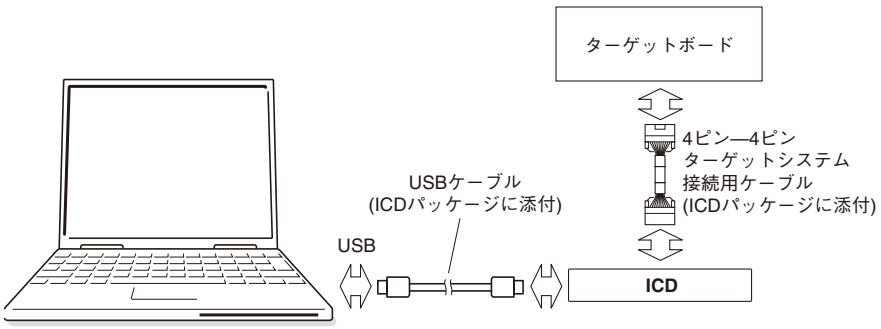


図10.6.1.1 ICDを使用するデバッグシステム例

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 Mini"を選択して、起動用コマンドファイルを生成します。

ICD Miniモードで起動する場合、ICD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。ICDの取り扱い等については、使用するICDのマニュアルを参照してください。
ICD Miniモードでは、トレース機能を使用することはできません。

●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、ICD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[デバッガ:]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ではトレース機能が使用可能です。フラッシュライタ機能は使用できません。

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**c17 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、c17 rpfコマンドは target コマンドの前に実行する必要があります。
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)
c17 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)
c17 rst                     (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。16ビットおよび32ビット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

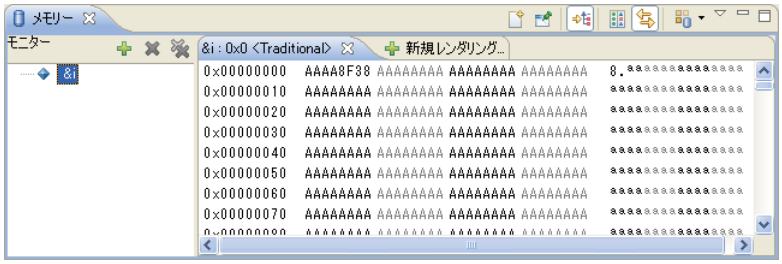
メモリ領域に対しては以下の操作が行えます。アドレスの指定には変数名などのシンボルも使用可能です。また、どの機能も、バイト、16ビット、または32ビット単位の処理に対応しています。

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16×16ビットのメモリダンプ

```
(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
```

[メモリー]ビューにメモリデータを表示させることもできます。[メモリー]ビューの表示と操作については、"10.4.9 [メモリー]ビュー "を参照してください。



データの入力(set {char}、set {short}、set {long}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[メモリー]ビュー上でも行えます。
例: int iを0x5555に設定

```
(gdb)
set {short}&i=0x5555
```

指定領域の書き換え(c17 fb、c17 fh、c17 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個の0x00000001(32ビット)を書き込み

```
(gdb)
c17 fw 0x0 0xf 0x1
Start address = 0x0, End address = 0xc, Fill data = 0x1 .....done
```

指定領域のコピー(c17 mvb、c17 mvh、c17 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)
c17 mvb 0x0 0x7 0x8
Start address = 0x0, End address = 0x7, Destination address = 0x8 .....done
```

メモリ内容の保存(c17 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S1/S2/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)
c17 df 0x80000 0x80103 5 dump.mot
Start address = 0x80000, End address = 0x80103, File type = Motorola-S3
Processing 00080000-00080103 address.
```

ターゲットメモリリードモードの指定(c17 readmdコマンド)　ICD Miniモードのみ

xコマンドやc17 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、**c17 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)
c17 readmd 1
```

●変数一覧

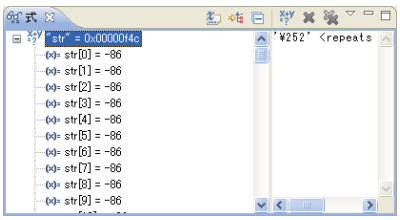
変数の一覧を表示できます。

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
```



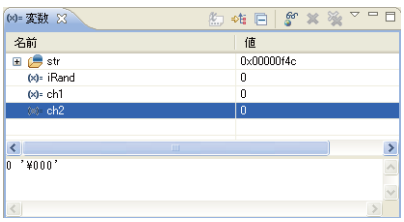
また、[式]ビューにグローバル変数を登録しておくことで、その値をモニタ可能です。[式]ビューの表示と操作については、"10.4.7 [式]ビュー "を参照してください。

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

```
(gdb)
info reg
r0              0xd20           3360
r1              0xaaaaaaa       11184810
r2              0xaaaaaaa       11184810
r3              0xaaaaaaa       11184810
r4              0x690           1680
r5              0xaaaaaaa       11184810
r6              0x0             0
r7              0xaaaaaaa       11184810
sp              0x7f8           2040
pc              0x4090          16528
psr             0x0             0
```

[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
```

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


[再開]ボタン

＊ 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い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)
- 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・オーバー] 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)
・ 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。
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

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。

●実行サイクル/実行時間の測定

ICD Mini、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。

実行カウンタ

ICD Miniは実行時間計測のみ、シミュレータモードはサイクル数計測のみ可能です。
測定結果は**c17 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。
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

デバッガの初期設定では、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c17 clockmdコマンドで設定することができます。リセットモードに設定した場合、実行サイクルカウンタは実行コマンド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(ブレーク)までを測定します。

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c17 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)
- リセットモードでプログラムの実行を開始した場合
- CPUをリセットした場合
- ICD Miniモードでc17 timebrk、step、stepi、next、nexti、finishコマンド実行をした場合

●CPUのリセット

CPUは**c17 rst**コマンドによりリセットされます。このコマンドは[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。

c17 rstコマンドの実行:
(¥gnu17¥reset.gdb)

- [実行]メニューから[リセット]を選択
- [リセット]ボタンをクリック


[リセット]ボタン

注: このメニューコマンド/ボタンは、既定のコマンドファイルを実行します。出荷時の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〜r7: 0x000000
pc: ブートアドレス(トラップテーブルのリセットベクタ)
sp: 0xffffffc
psr: 0x00(IL = 000、IE = 0、CVZN = 0000)

(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。

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）
- 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2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

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。
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1d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。
"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		

• デバッガ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、c17 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。c17 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S1C17コアのオンチップデバッガ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。ハードウェアPCブレークポイントとして設定できるアドレスは機種に依存します。詳細は各機種のテクニカルマニュアルを参照してください。
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。
"10.4.5 [ブレークポイント]ビュー "を参照してください。

注: 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		

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)

ソフトウェアPCブレーク、ハードウェアPC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、デバッガを終了するまで、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
```

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
2   hw breakpoint  del  n   0x00c00038 in sub at main.c:20
```

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。

●[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。

●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、c17 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.
```

10.6.6 トレース機能

シミュレータモードには、プログラムの実行をトレースする機能があります。

注: ICD Miniモードにはトレース機能はありません。

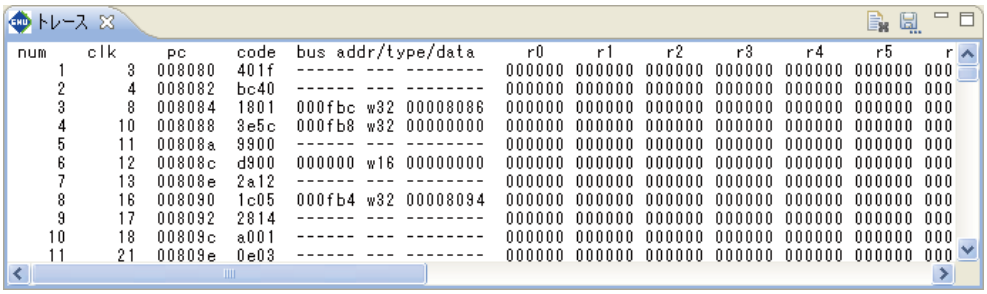
シミュレータモードでは**c17 tm**コマンドでトレース機能のON/OFF、表示方法およびファイル書き出しが指定できます。トレース機能をONすると、1命令実行ごとにトレース結果を表示あるいはファイルに保存します。

例1: 全情報を表示させるトレースモードを設定し、情報を保存するファイルtrace.logを指定

```
(gdb)
c17 tm on 0xff trace.log
```

例2: トレースモードを解除

```
(gdb)
c17 tm off
```



表示されるトレース情報は次のとおりです。

<各行の形式>

num clk pc code bus_addr/type/data r0 r1 r2 r3 r4 r5 r6 r7 sp ie/il/cvzn src_mix

num: 実行命令No.(10進数)
CPUをリセットしてからの実行命令数
clk: 実行クロック数(10進数)
CPUをリセットしてからの実行クロック数
pc: 実行アドレス(16進数)
code: 命令コード(16進数)
bus_addr: 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
r8: バイトデータリード、r16: 16ビットデータリード、r32: 32ビットデータリード
w8: バイトデータライト、w16: 16ビットデータライト、w32: 32ビットデータライト
data: リード/ライトデータ(16進数)
r0～r7: r0～r7レジスタ値(16進数)
sp: spレジスタ値(16進数)
ie: psrのIEビット値
il: psrのILビット値
cvzn: psrのC、V、Z、Nビット値
src_mix: 実行命令の逆アセンブル内容とソースコード

トレース情報はc17 tmコマンドでウィンドウへの表示を選択した場合に[Trace]ウィンドウに表示されます。

注: 表示されるクロック数(clk)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。

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入出力（stdin、stdout）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（c17 stdinコマンド）

c17 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（省略時は[コンソール]ビューからの入力）

これらの条件を設定後、プログラムを連続実行させます。

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)
c17 stdin 1 READ_FLASH READ_BUF input.txt
```

例2: データ入力機能を解除

```
(gdb)
c17 stdin 2
```

入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]ビューを開き、キーボードからデータが入力されるのを待ちます。データを入力して[Enter]キーを押すと、**gdb**は入力されたデータをバッファに書き込み、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入力機能を設定

```
(gdb)
c17 stdin 1 READ_FLASH READ_BUF
```

●stdoutによる出力（c17 stdoutコマンド）

c17 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バッファの内容を指定のファイルに出力します。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: データ出力機能を設定

```
(gdb)
c17 stdout 1 WRITE_FLASH WRITE_BUF output.txt
```

例2: データ出力機能を解除

```
(gdb)
c17 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]ビューを開き、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)
c17 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。
`c17 stdin`、`c17 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。
`c17 stdin`、`c17 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`(7.3.4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`¥gnu17¥sample¥S1C17common¥simulator¥simulatedIO`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

- `c17 stdin`、`c17 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。
- `c17 stdin`、`c17 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。

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ICDを使用するフラッシュライタ機能があります。

●ターゲットボード上のフラッシュメモリ操作

gdbには、S1C17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。
ICD Mini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、各機種の**fls**フォルダ(\gnu17\mcu_model\機種名\fls)内readme_x.txtを参照してください。

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 fls17701.elf              (デバッグ情報の読み込み)
(gdb)
target icd usb                 (ターゲットの接続)
C17 ICD17 debugging
Connecting with target (ID_OK) ..... done
ICD Initializing (ID_INITIALIZE) ... done
Read ICD Version (ID_VER_READ) ..... done
  ICD hardware version ............. 1.0
  ICD software version ............. 1.0
Boot address (ID_DATA_READ) ........ 0x008080
Hardware break MAX ................. 4
Target file is pointer24.
(gdb)
load                           (フラッシュルーチンのロード)
```

ここでロードしたルーチンを用いて、フラッシュの消去、書き込みを行います。

2. フラッシュの設定(c17 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0x8000 ～ 0x17fff、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)
c17 fls 0x8000 0x17fff FLASH_ERASE FLASH_LOAD
```

3. フラッシュの消去(c17 fleコマンド)

フラッシュメモリを全消去、あるいはブロック消去します。
消去されると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0x8000で、全消去する場合

```
(gdb)
c17 fle 0x8000 0 0
```

c17 flsコマンドの次に必ずc17 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、ブロック範囲(上記例の"0 0")に"-1 0"を指定することにより、フラッシュを消去せずに済みます。

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
```

フラッシュへプログラムを書き込む前にステップ１、ステップ２を実行してください。
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c17 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。

●ICD Mini(S5U1C17001H)のフラッシュライタ機能

ICD Mini(S5U1C17001H)はフラッシュライタ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。
ICD Mini(S5U1C17001H)をフラッシュライタとして使用する方法については、"S5U1C17001H Manual(S1C17 Family In-Circuit Debugger)"を参照してください。

フラッシュライタ機能用のコマンドは以下のとおりで、ICD Mini(S5U1C17001H)使用時(ICD Miniモード時)にのみ使用可能です。

注: ICDボードはフラッシュライタ機能に対応していません。

プログラム/データの消去(c17 fweコマンド)

ICD Mini(S5U1C17001H)にロードしたデータ消去/書き込みプログラム、書き込みデータとアドレス情報、またはセキュリティ設定を消去します。

例1: 書き込みデータの消去

```
(gdb)
c17 fwe 0
```

例2: データ消去/書き込みプログラムの消去

```
(gdb)
c17 fwe 1
```

例3: セキュリティ設定の消去

```
(gdb)
c17 fwe 2
```

プログラムのロード(c17 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ICD Mini(S5U1C17001H)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)
c17 fwlp writer.sa 0x90 0xb4
```

データのロード(c17 fwld、c17 fwdcコマンド)

c17 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ICD Mini(S5U1C17001H)にロードします。**c17 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ICD Mini(S5U1C17001H)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: フラッシュの先頭アドレスが0x8000で、全ブロックの消去を指定し、sample.saをロードする場合

```
(gdb)
c17 fwld sample.sa 0 0 0x8000
```

例2: フラッシュの先頭アドレスが0x8000で、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のデータをロードする場合

```
(gdb)
c17 fwdc FLASH_START 0x100000 0 0 0x8000
```

フラッシュライタ情報の表示(c17 fwdコマンド)

ICD Mini(S5U1C17001H)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)
c17 fwd
CPU data address             : xxxxxxxx
Data size                    : xxxxxxxx
Erase start block            : xxxxxxxx
Erase end block              : xxxxxxxx
Erase parameter              : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

CPU program address          : xxxxxxxx
Program size                 : xxxxxxxx
Erase routine entry address  : xxxxxxxx
Write routine entry address  : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

セキュリティの設定(c17 fwpwコマンド)

セキュリティ情報をホストからICD Mini(S5U1C17001H)に設定します。

例: ターゲット機種がC17001、Flashセキュリティのバージョンが M03、パスワードがABCD1234の場合

```
(gdb)
c17 fwpw 17001 M03 ABCD1234
```

10.6.9 ビッグエンディアンへの対応について

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プ情報で指定します。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S1C17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、"10.9 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`, `c17 fb/fh/fw`, `c17 mvb/mvh/mvw`)

コマンドのデータタイプに従ってバイト、16ビット、32ビット単位の読み出し/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、16ビット単位でデータを書き込みます。そのため、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。

10.7 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.
			ICD Mini	SIM	
メモリ操作	**c17 fb**	領域のフィル(バイト単位)	○	○	10-107
	c17 fh	領域のフィル(16ビット単位)	○	○	10-107
	c17 fw	領域のフィル(32ビット単位)	○	○	10-107
	x /b	メモリダンプ(バイト単位)	○	○	10-109
	x /h	メモリダンプ(16ビット単位)	○	○	10-109
	x /w	メモリダンプ(32ビット単位)	○	○	10-109
	set {char}	データ入力(バイト単位)	○	○	10-111
	set {short}	データ入力(16ビット単位)	○	○	10-111
	set {long}	データ入力(32ビット単位)	○	○	10-111
	c17 mvb	領域のコピー (バイト単位)	○	○	10-112
	c17 mvh	領域のコピー (16ビット単位)	○	○	10-112
	c17 mvw	領域のコピー (32ビット単位)	○	○	10-112
	c17 df	メモリ内容の保存	○	○	10-114
	c17 readmd	メモリリードモード	○	–	10-116
レジスタ操作	**info reg**	レジスタ表示	○	○	10-117
	set $	レジスタ変更	○	○	10-118
プログラム実行	**continue**	連続実行	○	○	10-119
	until	テンポラリブレーク付き連続実行	○	○	10-120
	step	ステップ実行(行単位)	○	○	10-122
	stepi	ステップ実行(ニーモニック単位)	○	○	10-122
	next	スキップ付きステップ実行(行単位)	○	○	10-124
	nexti	スキップ付きステップ実行(ニーモニック単位)	○	○	10-124
	finish	関数終了	○	○	10-126
	c17 callmd	ユーザ関数コールモード設定	○	○	10-127
	c17 call	ユーザ関数コール	○	○	10-128
CPUリセット	**c17 rst**	リセット(reset.gdbを実行)	○	○	10-130
	c17 rstt	ターゲットのリセット	○	–	10-131
割り込み	**c17 int**	割り込み	–	○	10-132
	c17 intclear	割り込み解除	–	○	10-133
	c17 int_load	イベントファイルの読み込み	–	○	10-134
ブレーク	**break**	ソフトウェアPCブレーク設定	○	○	10-135
	tbreak	テンポラリソフトウェアPCブレーク設定	○	○	10-135
	hbreak	ハードウェアPCブレーク設定	○	○	10-138
	thbreak	テンポラリハードウェアPCブレーク設定	○	○	10-138
	delete	ブレーク番号によるブレークの解除	○	○	10-141
	clear	ブレーク位置によるブレークの解除	○	○	10-142
	enable	ブレークポイントの有効化	○	○	10-143
	disable	ブレークポイントの無効化	○	○	10-143
	ignore	回数指定付きブレークの無効化	○	○	10-145
	info breakpoints	ブレークポイントリストの表示	○	○	10-146
	c17 timebrk	時間経過後ブレークの設定	○	–	10-147
	commands	ブレーク時に実行するコマンドの設定	○	○	10-148
シンボル情報	**info locals**	ローカルシンボル表示	○	○	10-149
	info var	グローバルシンボル表示	○	○	10-149
	print	シンボル値の変更	○	○	10-150
ファイル読み込み	**file**	デバッグ情報の読み込み	○	○	10-151
	load	プログラムのロード	○	○	10-152
	c17 loadmd	プログラムのロードモード設定	○	–	10-153
マップ情報	**c17 rpf**	マップ情報の設定	○	○	10-154
	c17 map	マップ情報の表示	–	○	10-155
フラッシュメモリ操作	**c17 fls**	フラッシュメモリ設定	○	–	10-156
	c17 fle	フラッシュメモリ消去	○	–	10-157
	c17 flv	フラッシュメモリの書き込み・消去電圧の設定	●*1	–	10-158
	c17 flvs	フラッシュメモリの書き込み・消去電圧の設定の解除	●*1	–	10-159
トレース	**c17 tm**	トレースモード設定	–	○	10-160
シミュレーテッドI/O	**c17 stdin**	データ入力シミュレーション	○	○	10-163
	c17 stdout	データ出力シミュレーション	○	○	10-164

分 類	コマンド	機 能	モード対応		P No.
			ICD Mini	SIM	
フラッシュライタ	**c17 fwe**	プログラム/データ/セキュリティ設定の消去	● *2	–	10-165
	c17 fwlp	プログラムのロード	● *2	–	10-166
	c17 fwld	データのロード	● *2	–	10-167
	c17 fwdc	ターゲットメモリのコピー	● *2	–	10-168
	c17 fwd	フラッシュライタ情報の表示	● *2	–	10-169
	c17 fwpw	セキュリティの設定	● *2	–	10-170
プロファイラ・カバレッジ	**c17 profilemd**	プロファイル・カバレッジモードの設定	–	○	10-171
	c17 profile	プロファイルビューの表示	–	○	10-172
	c17 coverage	カバレッジビューの表示	–	○	10-173
その他	**set output-radix**	変数表示形式の変更	○	○	10-174
	c17 log	ロギング	○	○	10-175
	source	コマンドファイルの実行	○	○	10-176
	c17 clockmd	実行カウンタモード設定	○	○	10-177
	c17 clock	実行カウンタ表示	○	○	10-177
	target	ターゲットの接続	○	○	10-179
	detach	ターゲットの切断	○	○	10-180
	pwd	カレントディレクトリの表示	○	○	10-181
	cd	カレントディレクトリの変更	○	○	10-181
	c17 firmupdate	ICDファームウェア更新	○	–	10-182
	c17 ttbr	TTBRの設定	–	○	10-183
	c17 help	ヘルプ	○	○	10-184
	c17 chgclkmd	DCLK切替モード	○	–	10-186
	c17 pwul	Flash セキュリティのアクセス制限を解除	○	–	10-187
	quit	デバッガ終了	○	○	10-188

モード対応: ○ = 使用可　– = 使用不可　● = 制限付き

*1: ICDmini Ver.2.0(S5U1C17001H2100)用のコマンドです。ICDmini(S5U1C17001H)、ICDボードでは使用できません(エラーが表示されます)。

*2: ICD Mini(S5U1C17001H)用のコマンドです。ICDボードでは使用しても正常に動作しません(エラーも表示されません)。

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[ICD Mini / SIM]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。
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

c17 fb （領域のフィル , バイト単位）
c17 fh （領域のフィル , 16 ビット単位）
c17 fw （領域のフィル , 32 ビット単位） [ICD Mini / SIM]

機 能

c17 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。
c17 fh 指定のメモリ領域を指定の16ビットデータで書き換えます。
c17 fw 指定のメモリ領域を指定の32ビットデータで書き換えます。

書 式

c17 fb ***StartAddr EndAddr Data***
c17 fh ***StartAddr EndAddr Data***
c17 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≦*StartAddr*≦*EndAddr*≦0xffffff、0≦*Data*≦0xff(`c17 fb`)、0≦*Data*≦0xffff(`c17 fh`)、0≦*Data*≦0xffffffff(`c17 fw`)

使用例

■例1

```
(gdb)
c17 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
0x8: 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のバイトデータで書き換えます。

■例2

```
(gdb)
c17 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16ビットデータで書き換えます(リトルエンディアンの場合)。

■例3

```
(gdb)
c17 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32ビットデータで書き換えます(リトルエンディアンの場合)。

注 意

- 16ビットおよび32ビットデータの書き込みは、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)
c17 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32ビットデータ境界アドレスではないため、下位2ビットを00として(16ビット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、32ビットデータアドレス0x0～0x8(バイトデータアドレス0x0～0xb)が0x00000000のデータで書き換えられます。

```
(gdb)
c17 fw 0x0 0x8 0x0
```

- 24ビットを超えるアドレスを指定した場合はエラーとなります。
- データのパラメータは、`c17 fb`は下位8ビット、`c17 fh`は下位16ビット、`c17 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、`c17 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） [ICD Mini / SIM]

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** 16ビット単位
- **w** 32ビット単位(デフォルト)
- **i** 逆アセンブル

Address: 表示開始アドレス(10進数、16進数、シンボル、またはレジスタ名)
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≦*Length*≦2147483647、0≦*Address*≦0xffffff

表 示

メモリ内容は次のように表示されます。(Size:が b/h/wのとき)
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バイトまで表示されます。

逆アセンブルは次のように表示されます。(Size:が iのとき)
Address[<*Symbol*>]: *insn [ext insn]*

Address: 各行のコードのアドレスを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。
insn: *Address*から始まるアセンブラ基本命令が表示されます。
ext insn: *ext*拡張命令があるとき、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。

■例6

```
(gdb) x /10i boot
0x8004 <boot>:     ext   0x20
0x8006 <boot+2>:   ld.a  %sp,0x0    sld.a  %sp,0x1000
0x8008 <boot+4>:   call  0x1        call   0x1 (0x00800C) <main>
0x800a <boot+6>:   nop              nop
0x800c <main>:     ld    %r0,[0x0]  ld     %r0,[0x0] <p1>
0x800e <main+2>:   ld    %r1,[0x2]  ld     %r1,[0x2] <p2>
0x8010 <main+4>:   ext   0x0
0x8012 <main+6>:   ld    %r2,0x64   sld    %r2,0x64
0x8014 <main+8>:   call  0x16 call  0x16   (0x008042) <memcpy>
0x8016 <main+10>:  ld    %r0,[0x0]  ld     %r0,[0x0] <p1>
```

指定した10命令を表示します。

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。
- 24ビットを超えるアドレスを指定した場合はエラーとなります。
- 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。
- Lengthに2147483648以上を指定した場合でもエラーにはなりません。Lengthは2147483647に設定されます。

set { }（データ入力） [ICD Mini / SIM]

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** 16ビット単位（デフォルト）
- **int** 16ビット単位
- **long** 32ビット単位

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≦*Address*≦0xffffff、0≦*Data*≦0xff（set {char}）、0≦*Data*≦0xffff（set {short/int}）、0≦*Data*≦0xffffffff（set {long}）

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16ビットデータ0x5555を書き込みます。

■例3

```
(gdb)
set {long}&i=0x55555555
(gdb)
x /w &i
0x0 <i>: 0x55555555
```

long変数iに32ビットデータ0x55555555を書き込みます。

注 意

- 16ビットおよび32ビットデータの書き込みは、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。
- 24ビットを超えるアドレスを指定した場合はエラーとなります。
- データのパラメータは、set {char}は下位8ビット、set {short}およびset {int}は下位16ビット、set {long}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。

c17 mvb（領域のコピー , バイト単位）
c17 mvh（領域のコピー , 16 ビット単位）
c17 mvw（領域のコピー , 32 ビット単位） [ICD Mini / SIM]

機 能

c17 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。
c17 mvh 指定のメモリ領域の内容を16ビット単位で別の領域にコピーします。
c17 mvw 指定のメモリ領域の内容を32ビット単位で別の領域にコピーします。

書 式

c17 mvb ***SourceStart SourceEnd Destination***
c17 mvh ***SourceStart SourceEnd Destination***
c17 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≦*SourceStart*≦*SourceEnd*≦0xffffff、0≦*Destination*≦0xffffff

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)
c17 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)
c17 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
```

long変数iの内容を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。
- 24ビットを超えるアドレスを指定した場合はエラーとなります。
- c17 mvhとc17 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。
  c17 mvh: アドレスの最下位ビットを0として処理します。
  c17 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0xffffffを超える場合、0xffffff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。

c17 df（メモリ内容の保存） [ICD Mini / SIM]

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

c17 df ***StartAddr EndAddr Type Filename*** [***Append***]

StartAddr: 先頭アドレス（10進数、16進数、またはシンボル）
EndAddr: 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S1ファイル
- **4** モトローラS2ファイル
- **5** 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 = 3,4,5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 = 3,4,5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≦*StartAddr*≦*EndAddr*≦0xffff（モトローラS1ファイルの場合）
0≦*StartAddr*≦*EndAddr*≦0xffffff（バイナリ/テキスト/モトローラS2/S3ファイルの場合）

使用例

■例1

```
(gdb)
c17 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)
c17 df 0x80000 0x80103 5 dump.mot
Start address = 0x80000, End address = 0x80103, File type = Motorola-S3
Processing 00080000-00080103 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。1行が32バイトに満たない場合は、指定されたアドレスの範囲分バイト数を書き出します。
（dump.motの内容）

```
S3250008000094D4BA020FCA086120800961881C6F0AA4D4BA020FCA086120800961881C6F0AA8
S3250008002008D3730A0000000000000000000000008D3730A09CA026139A505610CD309613F
                                        :
S30800080100A4D5BABB
S70500000000FA
```

■例3

```
(gdb)
c17 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)
c17 df 0x1000 0x1fff 5 dump.mot a ; フッタは出力しません。( 最初 )
(gdb)
c17 df 0x3000 0x3fff 5 dump.mot a ; フッタは出力しません。
(gdb)
c17 df 0x5000 0x5fff 5 dump.mot a ; フッタは出力しません。
(gdb)
c17 df 0x7000 0x7fff 5 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

- 24ビットを超えるアドレスを指定した場合はエラーとなります。
- 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。

c17 readmd（メモリリードモード） [ICD Mini]

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8ビッ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8ビット単位でアクセスするときは`ld.b`命令
   メモリを16ビット単位または32ビット単位でアクセスするときは`ld`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。

8ビッ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

`c17 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)
c17 readmd 1
```

境界正確モードに設定します。

注 意

- `c17 readmd`コマンドは、シミュレータモードでは使用できません。
- 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。
  `c17 df`、`x`

10.7.4 レジスタ操作コマンド

info reg（レジスタ表示） [ICD Mini / SIM]

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-r7、sp、pc、psr

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             0xaaaaaa    1184810
(gdb)
info reg pc
pc             0x4090      16528
```

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

```
(gdb)
info reg
r0             0xd20       3360
r1             0xaaaaaa    1184810
r2             0xaaaaaa    1184810
r3             0xaaaaaa    1184810
r4             0x690       1680
r5             0xaaaaaa    1184810
r6             0x0         0
r7             0xaaaaaa    1184810
sp             0x7f8       2040
pc             0xc00030    12582960
psr            0x7f8       2040
```

注 意

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） [ICD Mini / SIM]

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-r7、sp、pc、psr
Value: 設定する24ビットデータ（10進数、16進数、またはシンボル）
条件: 0≦*Value*≦0xffffff

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

- 24ビットを超える値を指定した場合は、下位24ビットのみ有効となります。
- 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） [ICD Mini / SIM]

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継続します。
- 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

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継続します。

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

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

(gdb)
continue
```

注 意

- プログラムの先頭から実行するには、continueの前にc17 rst（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） [ICD Mini / SIM]

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
- 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≦アドレス≦0xffffff

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c17 rst(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、24ビットを超える値を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、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） [ICD Mini / SIM]

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ソースは1行の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

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≦*Count*≦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c17 rst（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。

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） [ICD Mini / SIM]

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ソースは1行の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

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≦*Count*≦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c17 rst(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、[中断]ボタンで解除することもできます。

finish（関数終了） [ICD Mini / SIM]

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ーク条件が成立
- [中断]ボタンのクリック

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。

c17 callmd（ユーザ関数コールモード設定） [ICD Mini / SIM]

機 能

c17 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

c17 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)
c17 callmd 2 call.txt
```

これから実行するc17 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c17 callコマンド実行時にカレントディレクトリに作成され、c17 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。

c17 call（ユーザ関数コール） [ICD Mini / SIM]

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

c17 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)
c17 callmd 1
(gdb)
c17 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

c17 callにより**gdb**は指定のユーザ関数を実行し、%r0から戻り値を受け取ります。戻り値が-1（0xffffff）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c17 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%r0に設定してください。パケットを返せない場合は%r0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"jpa %r1"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
         jpa      %r1             ; back to mini monitor

;        ****28 - ****3f DSIO command area, 24byte
;
;                 ****28          WBUF data1        DSIO output command
;                 ****2c          WBUF data2
;
;                 ****32          RBUF ID           DSIO input command
;                 ****33          RBUF size
;                 ****34          RBUF addr
;                 ****38          RBUF data1
;                 ****3c          RBUF data2

;
; input
;        %r0 : debug ram start address = ****28
;        %r1 : return address( mini Monitor )
; output
;                  ****28 WBUF data1 : gdb output message start address
;
```

```
;               <message format>
;                size    : message size (4 byte)
;                message : String data
;
#define SP_INI  0x1000

    .global ext_test
ext_test:
         ld.a     %r7, %sp             ;
         Xld.a    %r2, SP_INI          ; set SP
         ld.a     %sp, %r2
         ld.a     -[%sp], %r7          ; save SP
         ld.a     -[%sp], %r1          ; save return address
         ld.a     -[%sp], %r0          ; save debug ram start address
         ld.a     %r6, %r0             ; %r0 : debug ram start address = ****28
         add.a    %r6, 0x0c
         ld       %r0, [%r6]+          ; Set Arg1
         ld       %r1, [%r6]+
         ld       %r2, [%r6]+          ; Set Arg2
         ld       %r3, [%r6]+
         ld       %r5, [%r6]+          ; Set Arg3
         ld       %r4, [%r6]+
         sub.a    %sp, 4
         ld       [%sp+0x0], %r5
         ld       [%sp+0x2], %r4
         Xcall    iprint_data          ; enter C program
                                       ; return = %r0: message buffer address
         add.a    %sp, 4
         ld.a     %r6, %r0             ;
         ld.a     %r0, [%sp]+          ; restore debug ram start address
         ld.a     [%r0], %r6           ; set %r0 ( message address ) to WBUF data1
         ld.a     %r1, [%sp]+          ; restore return address
         ld.a     %r7, [%sp]+          ; restore SP
         ld.a     %sp, %r7
         jpa      %r1                  ; back to mini monitor
```

ユーザ関数

```
#include "stdio.h"

void *iprint_data( long arg1, long arg2, long arg3 );

struct {
               long size;
               char buf[0x100];
} tmpbuf;

void *iprint_data( long arg1, long arg2, long arg3 )
{
         tmpbuf.size = sprintf(tmpbuf.buf,"arg1=%d,arg2=%d,arg3=%d",arg1,arg2,arg3);
         return (void*)&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。

10.7.6 CPUリセットコマンド

c17 rst（リセット） [ICD Mini / SIM]

機　能

CPUをリセットします。
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～r7: 0x000000
pc: ブートアドレス（トラップテーブルのリセットベクタ）
sp: 0xffffffc
psr: 0x00（IL = 000、IE = 0、CVZN = 0000）

(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

```
c17 rst
```

使用例

```
(gdb)
c17 rst
```

CPUをリセットします。

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。
- ICD Miniモードで使用する場合、バスやI/Oの状態は保持されます。

c17 rstt（ターゲットリセット） [ICD Mini]

機　能

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

```
c17 rstt
```

使用例

```
(gdb)
c17 rstt
```

ターゲットをリセットします。

注　意

- c17 rsttコマンドはICD Mini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。

10.7.7 割り込みコマンド

c17 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

c17 int [***No Level***]

No:　割り込み番号（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≦*No*≦0x1f、0≦*Level*≦7

使用例

■例1

```
(gdb)
c17 int
```

パラメータを指定しない場合、NMIを発生します。

■例2

```
(gdb)
c17 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

- c17 intコマンドはシミュレータモード時のみ使用可能です。
- 割り込み番号は0～31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0～7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TTBRは有効です。

c17 intclear（割り込み解除） [SIM]

機 能

割り込みの解除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、その割り込みが解除されます。

書 式

c17 intclear [***No***]

No: 割り込み番号（10進数、16進数、またはシンボル）
条件: 0≦*No*≦0x1f

使用例

```
(gdb)
c17 int 3 6
(gdb)
continue
(gdb)
c17 intclear 3
```

割り込み番号3の割り込みを解除します。

注 意

- c17 intclearコマンドはシミュレータモード時のみ使用可能です。
- 割り込み番号は0～31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。

c17 int_load（割り込みイベントファイル読み込み） [SIM]

機 能

割り込みイベントファイルを読み込みます。
プログラムを実行中に、読み込んだファイルに記述された割り込みイベント条件が満たされると、割り込みを発生します。

イベントファイルフォーマット

Address_NMI_INT_Vector_Level_RES // コメント
Address_NMI_INT_Vector_Level_RES
Address_NMI_INT_Vector_Level_RES
Address_NMI_INT_Vector_Level_RES
:
Address_NMI_INT_Vector_Level_RES

Address: イベント発生アドレス（000000～fffffe）
PCがこの値と一致したとき割り込みを発生します。
NMI: 1 = NMI割り込み要求を発生
0 = 要求なし
INT: 1 = 以下の指定による*Vector*と*Level*の割り込み要求を発生
0 = 要求なし
Vector: 割り込みベクタ番号（00～1f、*INT* = 1のとき有効）
Level: 割り込みレベル（0～7、*INT* = 1のとき有効）
RES: 1 = リセット要求
0 = 要求なし

割り込みの優先順位は*RES* > *NMI* > *Vector*の順です。
イベント行の右側に"//"で始まるコメント（英数字）を記述可能です。

書 式

c17 int_load *Filename*

Filename: 割り込みイベントファイル名

使用例

```
(gdb)
c17 int_load event.txt
```

割り込みイベントファイルevent.txtを読み込みます。

イベントファイル例

009002_0_1_10_3_0	PC = 0x9002のとき、割り込みベクタ番号 = 0x10、割り込みレベル = 3の割り込みを発生します。
009f02_0_0_00_0_1	PC = 0x9f02のとき、リセットを発生します。
004030_1_1_1c_7_0	PC = 0x4030のとき、NMIを発生します。その後、割り込み許可されたところで、割り込みベクタ番号 = 0x1c、割り込みレベル = 7の割り込みを発生します。
004030_0_1_1c_7_1	PC = 0x4030のとき、リセットを発生します。*INT* = 0のため、その後で割り込み許可されても割り込みベクタ番号 = 0x1c、割り込みレベル = 7の割り込みは発生しません。

注 意

- c17 int_loadコマンドはシミュレータモード時のみ使用可能です。
- 割り込みイベントファイルの読み込み後にc17 rstコマンドを実行すると、ファイルの最初の行から割り込みイベントが発生するように再設定されます。
- イベントファイルに記述可能な最大イベント数（イベントの行数）は256です。
- イベント行の最大文字数は300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） [ICD Mini / SIM]

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≦アドレス≦0xfffffe

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、デバッガを終了するまで、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
`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

- 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。
  デバッガ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`ld`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、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。また、24ビットを超えるアドレスを指定した場合はエラーとなります。
- 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。
- BRK命令をソースに埋め込んだときの処理

breakコマンドではなく、ユーザアプリケーションソースに埋め込まれたBRK命令を実行した場合には、ソフトブレーク後、自動的にPC+=2を行います（以下の例では①+2で②になる）。C17にはハードブレークが1つしか設定できないため、flashメモリ等ROMで実行する場合、BRK命令をソースに埋め込むことにより複数箇所でブレークが可能になります。

例）

```
 sample.c
 void main()
 {
     •                    ;      <  ここでcontinue
     •
     •
     a = b + 1            ;
     iRet = sub( a )      ;      <  ③
     asm( "brk" )         ;      <  ① brk命令埋め込み
     if ( iRet == 1 ) {   ;      <  ② ここで停止（BRK命令のアドレス+2）
     b -= 2               ;
 }
     •
     •
     •
```

上記例で①にソフトブレーク設定した場合は①で停止します。これは、メモリに埋め込まれたBRK命令が、breakコマンドによるものか、ソースに埋め込まれていたものかが、デバッガには判断できないためです。また、次に実行する前に、ブレークポイント設定を解除する必要があります。①にハードブレーク設定した場合は②で停止します。③をnext後は②で停止します。

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） [ICD Mini / SIM]

機　能

ハードウェアPCブレークポイントを設定します。ハードウェアPCブレークポイントはICDモードでは機種により 1 ～ 4 箇所、SIMモードでは 1 箇所のみ設定可能です。
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。
デバッガから接続したとき、ハードPCブレークの本数は各IBExビットに1ライト、1リードができるかで判断します。実装していないブレーク番号については1が書き込めません(1書き込み、0リード)。

```
0xffffa0
   D0 DM
   D1 SE
   D2 IBE0
   D3 IBE1
   D4 DR
   D5 IBE2          <-追加
   D6 IBE3          <-追加
   D7 IBE4          <-追加

0xffffb0 IBAR0
0xffffb4 IBAR1
0xffffb8 IBAR2           <-追加
0xffffbc IBAR3           <-追加
0xffffd0 IBAR4           <-追加
```

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≦アドレス≦0xfffffe

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、デバッガを終了するまで、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
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

- 有効なハードウェアPCブレークポイントはICDモードでは機種により1～4箇所、SIMモードでは1箇所のみ設定可能です。これ以上に設定する場合は、無効化の状態で設定可能です。テンポラリハードウェア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。
- ニーモニックに展開されない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ld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。
- ハードウェア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。

delete（ブレーク番号によるブレークの解除） [ICD Mini / SIM]

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)
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   breakpoint     keep y   0x00c0003c in sub at main.c:22
```

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   breakpoint     keep y   0x00c0003c in sub at main.c:22
```

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。

clear（ブレーク位置によるブレークの解除） [ICD Mini / SIM]

機　能

設定されているPCブレークポイント(テンポラリブレークを含む)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。

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)

条件: 0≦アドレス≦0xfffffe

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
```

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
```

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
```

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。

注　意

- 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。

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） [ICD Mini / SIM]

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)
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
```

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
```

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)
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
```

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。

ignore（回数指定付きブレークの無効化） [ICD Mini / SIM]

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
```

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。

info breakpoints（ブレークポイントリストの表示） [ICD Mini / SIM]

機 能

設定されているブレー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
2   hw breakpoint  del  n   0x00c00038 in sub at main.c:20
```

Num: ブレーク番号を表示します。
Type: ブレークポイントの種類を表示します。
`breakpoint` ソフトウェアPCブレークポイント
`hw breakpoint` ハードウェアPCブレークポイント
Disp: ブレークヒット後のブレークポイントの状態を表示します。
`keep` ブレークポイントは削除されません。
`del` ブレークポイントが削除されます。テンポラリブレークであることを示します。
Enb: ブレー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: ブレー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。
What: ブレークポイントが設定された位置を表示します。
ブレークポイントは、"`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。

その他、各ブレークポイントごとに、現在までのヒット回数を"breakpoint already hit N times"の形で表示します。

ブレークポイントが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.
```

c17 timebrk（時間経過後ブレークの設定） [ICD Mini]

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

c17 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)
c17 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)
c17 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、シミュレータモードでは使用できません。
- ブレーク時間を設定すると、"c17 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。

commands（ブレーク後に実行するコマンドの設定） [ICD Mini / Sim]

機 能

指定したブレークポイントで停止したときに実行するコマンド(複数行)の設定及び、解除を行います。

書 式

commands [ブレーク番号]

コマンド入力後、プロンプトが“>"に変わります。ここで、ブレーク時に設定するコマンド行を入力します。
コマンド行は、複数行可能で、最後に“end"を入力すると設定を終了します。
設定したコマンド行は、info breakpointコマンドで確認可能です。
ブレーク番号省略時は、直前に設定したブレークポイントの番号が指定されます。

コマンド行を解除する方法：
プロンプトが">"になったら 1 行目に"end"を入力します。

使用例

```
(gdb)
break boot.s:16
(gdb)
commands 1
>x /4b 0x100
>break main
>continue
>x /4b sub
>end
(gdb)continue
Continuing.

Breakpoint 1, boot () at boot.s:16
0x100: 0xaa 0xaa 0xaa 0xaa
Breakpoint 2 at 0x632: file main.c, line 18.

Breakpoint 2, main () at main.c:18
Current language: auto; currentry c

0x658 <sub>:   0x00   0x40   0x25  0x18

(gdb)
```

注 意

- 存在しない数値をブレーク番号に指定するとエラーになります。
- ブレーク番号省略時は、直前に設定したブレークポイントの番号が指定されます。
- コマンド行は最大50行まで動作保障されます。50行を超えてもエラーにはなりません。51行以上はAS ISで使用となります。
- コマンド実行中にエラーが発生した場合は、そこで停止します。
- テンポラリブレークポイント(tbreak、thbreak)でブレークした場合は、コマンド行は実行しません。
- commandsコマンドのネストはできません。コマンド行にcommandsコマンドがあると、ブレーク時にそのcommandsコマンドでのコマンド行が入力できなくなります。

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）　[ICD Mini / SIM]

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） [ICD Mini / SIM]

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≦*Value*≦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0x10000を指定すると、0x0000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） [ICD Mini / SIM]

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)
c17 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)
c17 rst                      (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。
- fileコマンド実行時、指定したファイルの作成時間がソースファイルより新しい場合、ワーニングダイアログ：「source file is more recent than executable」が表示されるので[OK]ボタンをクリックします。
- 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。
- 非サポート(C17用フラグの無い)のelfファイルを指定するとエラーになります。

load（プログラムのロード） [ICD Mini / SIM]

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ROMデータ(モトローラ形式)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)
file sample.elf              (デバッグ情報の読み込み)
(gdb)
c17 rpf sample.par           (マップ情報設定)
(gdb)
target sim                   (ターゲットの接続)
(gdb)
load                         (本コマンド)
(gdb)
c17 rst                      (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。

c17 loadmd（プログラムのロードモード設定） [ICD Mini]

機 能

load命令によるターゲットメモリへのプログラムのロード時に、elf形式のオブジェクトファイルからプログラム/データを読み込む方法を設定します。

書 式

c17 loadmd ***Mode***

Mode: 転送モード設定
- **0** 高速バイト転送モード（デフォルト）
- **1** 低速バイト転送モード

使用例

```
(gdb)
file sample.elf
(gdb)
target icd usb
(gdb)
c17 loadmd 0
(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/データを高速8ビット命令で読み込み、ターゲットメモリにロードします。

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。

10.7.11 マップ情報コマンド

c17 rpf（マップ情報の設定） [ICD Mini/SIM]

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

c17 rpf ***Filename***

Filename: パラメータファイル名（パスも指定可能）

使用例

```
(gdb)
c17 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

- c17 rpfコマンドはシミュレータモード時のみ使用可能です。
- パラメータファイルについては、"10.9 パラメータファイル"を参照してください。
- c17 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)
file sample.elf                 （デバッグ情報の読み込み）
(gdb)
c17 rpf sample.par              （本コマンド）
(gdb)
target sim                      （ターゲットの接続）
(gdb)
load                            （プログラムのロード）
(gdb)
c17 rst                         （リセット）
```

  メモリマップ情報を再設定する場合は、デバッガを一旦終了し、再起動してからc17 rpfコマンドを実行してください。
- デバッガ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）、領域のサイズを小さくしてください。
- 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。
- 本コマンドを実行しない場合とICD接続モード時は、以下のメモリマップ情報に設定されます。
  TTBR　0x8000
  RAM　0x0～0xffffff、R/Wウェイト = 0、16ビットアクセス、リトルエンディアン

c17 map（マップ情報の表示） [SIM]

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

c17 map

表 示

```
(gdb)
c17 map
CPU : C17
Memory map information Type  Wait(r/w) Size  Endian
    00008000           TTBR
    00000000-00ffffff  RAM     0/0     16Bit Little
    00000000-00003eff  STACK
```

CPU: CPUの種類を表示します。
C17: S1C17コア

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。

c17 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)
c17 map
CPU : C17
Memory map information Type  Wait(r/w) Size  Endian
    00008000           TTBR
    00000000-00ffffff  RAM     0/0     16Bit Little
```

注 意

c17 mapコマンドはシミュレータモード時のみ使用可能です。

10.7.12 フラッシュメモリ操作コマンド

c17 fls（フラッシュメモリ設定） [ICD Mini]

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

c17 fls ***StartAddr EndAddr ErasePrg WritePrg [SendSize]***

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）
SendSize: フラッシュロードプログラムを転送するために使用する作業領域のサイズ（10進数、16進数、またはシンボル）
条件: 0≦*StartAddr*≦*EndAddr*≦0xffffff、0≦*ErasePrg*≦0xffffff、0≦*WritePrg*≦0xffffff、0≦*SendSize*≦1010

使用例

```
(gdb)
c17 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。
- SendSizeは、ご使用の機種のフラッシュロードプログラムの使用可能な領域（内蔵RAM）サイズ内に収まるように指定してください。メモリマップなどの詳細は、各機種のテクニカルマニュアルを参照してください。

c17 fle（フラッシュメモリ消去） [ICD Mini]

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

c17 fle ***ControlReg StartBlock EndBlock*** ［***Timer***］

ControlReg: c17 fls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)
c17 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、c17 fls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c17 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。
- 機種によってはフラッシュの消去範囲を任意に設定できないものがあります。その機種では全ブロック消去に設定してください。フラッシュの消去範囲を任意に設定できるかどうかは、各機種のflsフォルダ（¥gnu17¥mcu_model¥機種名¥fls）内readme_x.txtを参照してください。

c17 flv（フラッシュメモリの書き込み・消去電圧の設定） [ICD Mini]

機 能

ICDmini Ver.2.0(S5U1C17001H2100) 専用のコマンドです。
フラッシュプログラミング電源端子があるマイコンの書き込み時、及び消去時の電圧値をICDminiへ設定します。このコマンドは、loadコマンドによるフラッシュマイコンへのデータ書き込み(loadコマンド)、セクタ消去(c17 flsコマンド)の前で使用します。

書 式

c17 flv ***Voltage***

Voltage: フラッシュへの書き込み/消去電圧値(10進数、0.1V単位)
入力形式は「7.0」のように、少数点と「0」は省略できません。
条件: 6.0 ≦ *Voltager* ≦ 8.0

使用例

```
(gdb)
 c17 fls  0x8000  0x1ffff  FLASH_ERASE  FLASH_LOAD
Flash start address = 0x8000, Flash end address = 0x1ffff
Flash erase routine address = 0x100, Flash write routine address = 0x200 .....done
(gdb)
 c17 flv 7.5
Set flash voltage 7.5V
(gdb)
 c17 fle  8000  0  0
(gdb)
 c17 flvs                ※フラッシュ消去後は、必ず実行すること。
Stop output flash voltage.
(gdb)
 c17 flv 7.0
Set flash voltage 7.0V
(gdb)
 load
(gdb)
 c17 flvs                ※フラッシュ書き込み後は、必ず実行すること。
Stop output flash voltage.
```

フラッシュメモリの全領域を 7.5V で消去し、c17 flvs コマンドで設定を解除します。次にフラッシュメモリへプログラムを 7.0V で書き込みます。
最後に c17 flvs コマンドで設定を解除します。

注 意

- 本コマンドを使用するには、フラッシュプログラミング電源端子があるマイコンと、ICDmini Ver.2.0(S5U1C17001H2100) が必要です。(ICDmini が異なるとエラーになります。)
- 設定値は機種によって異なります。各機種のテクニカルマニュアルに記載されている"電気的特性"を参照してください。
- 本コマンドはシミュレータモードでは使用できません。
- フラッシュメモリの書き込み・消去後は、c17 flvsコマンドで書き込み・消去電圧の設定を解除してください。
- loadコマンドが失敗した場合、自動的に電圧供給を解除します。(c17 flvsと同等な処理を行います。)

c17 flvs（フラッシュメモリの書き込み・消去電圧設定の解除） [ICD Mini]

機 能

ICDmini Ver.2.0(S5U1C17001H2100) 専用のコマンドです。
c17 flvコマンドで設定した電圧設定を解除します。
このコマンドは、loadコマンドによるフラッシュマイコンへのデータ書き込み(loadコマンド)、セクタ消去(c17 fleコマンド)の後で使用します。

書 式

c17 flvs

使用例

```
(gdb)
 c17 fls  0x8000  0x1ffff  FLASH_ERASE  FLASH_LOAD
Flash start address = 0x8000, Flash end address = 0x1ffff
Flash erase routine address = 0x100, Flash write routine address = 0x200 .....done
(gdb)
 c17 flv 7.5
Set flash voltage 7.5V
(gdb)
 c17 fle  8000  0  0
(gdb)
 c17 flvs            ※フラッシュ消去後は、必ず実行すること。
Stop output flash voltage.
(gdb)
 c17 flv 7.0
Set flash voltage 7.0V
(gdb)
 load
(gdb)
 c17 flvs            ※フラッシュ書き込み後は、必ず実行すること。
Stop output flash voltage.
```

フラッシュメモリの全領域を7.5Vで消去し、c17 flvsコマンドで設定を解除します。次にフラッシュメモリへプログラムを7.0Vで書き込みます。
最後にc17 flvsコマンドで設定を解除します。

注 意

- 本コマンドを使用するには、フラッシュプログラミング電源端子があるマイコンと、ICDmini Ver.2.0(S5U1C17001H2100) が必要です。(ICDmini が異なるとエラーになります)
- 本コマンドはシミュレータモードでは使用できません。
- フラッシュメモリの書き込み・消去後は、本コマンドを実行してください。

10.7.13 トレースコマンド

c17 tm（トレースモード設定） [SIM]

機 能

以下の選択を行います。

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。

トレース情報の表示項目
トレース情報の中で表示させる項目を選択できます。

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、[トレース]ビュー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

```
c17 tm on Mode [Filename]      （トレースモード設定）
c17 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）
0x00～0xffの範囲で指定します。ビットが1の情報を表示します。
ビット0 トレース番号
ビット1 クロック数
ビット2 PC値と命令コード
ビット3 バス情報（アドレス、R/Wおよびアクセスサイズ、データ）
ビット4 レジスタ値（R0～R7、SP）
ビット5 PSR値（IE、IL、CVZN）
ビット6 逆アセンブル内容とソース
ビット7 クロック数の積算表示指定（0の場合は命令個別のクロック数表示）

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

■例1

```
(gdb)
c17 tm on 0xff trace.log
```

全情報を表示させるトレースモードを設定し、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)
c17 tm off
```

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

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。

各行の形式

num clk pc code bus_addr/type/data r0 r1 r2 r3 r4 r5 r6 r7 sp ie/il/cvzn src_mix

num: 実行命令No.(10進数)
CPUをリセットしてからの実行命令数
clk: 実行クロック数(10進数)
CPUをリセットしてからの実行クロック数
pc: 実行アドレス(16進数)
code: 命令コード(16進数)
bus_addr: 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
r8: バイトデータリード、r16: 16ビットデータリード、r32: 32ビットデータリード
w8: バイトデータライト、w16: 16ビットデータライト、w32: 32ビットデータライト
data: リード/ライトデータ(16進数)
r0～r7: r0～r7レジスタ値(16進数)
sp: spレジスタ値(16進数)
ie: psrのIEビット値
il: psrのILビット値
cvzn: psrのC、V、Z、Nビット値
src_mix: 実行命令の逆アセンブル内容とソースコード

表示例

各行の前半(トレース番号～レジスタ値)

```
num    clk     pc     code  bus addr/type/data     r0     r1     r2     r3     r4     r5     r6     r7
  652   1445  0040dc  9900  ------ --- --------  000094 000000 000000 00ffff 000000 000000 000000 000000
  653   1446  0040de  4000  ------ --- --------  000094 000000 000000 00ffff 000000 000000 000000 000000
  654   1447  0040e0  4000  ------ --- --------  000094 000000 000000 00ffff 000000 000000 000000 000000
  655   1449  0040e2  d900  000000 w16 00000000  000094 000000 000000 00ffff 000000 000000 000000 000000
  656   1450  0040e4  2a12  ------ --- --------  000094 000000 000000 00ffff 000000 000000 000000 000000
  657   1451  0040e6  2814  ------ --- --------  000000 000000 000000 00ffff 000000 000000 000000 000000
  658   1452  0040e8  4000  ------ --- --------  000000 000000 000000 00ffff 000000 000000 000000 000000
  659   1457  0040ea  1805  003ef4 w32 000040ec  000000 000000 000000 00ffff 000000 000000 000000 000000
  660   1458  0040f6  a001  ------ --- --------  000000 000000 000000 00ffff 000000 000000 000000 000000
  661   1459  0040f8  9000  ------ --- --------  000000 000000 000000 00ffff 000000 000000 000000 000000
  662   1462  0040fa  0e0e  ------ --- --------  000000 000000 000000 00ffff 000000 000000 000000 000000
  663   1466  004118  0120  003ef4 r32 000040ec  000000 000000 000000 00ffff 000000 000000 000000 000000
  664   1467  0040ec  8201  ------ --- --------  000000 000000 000000 00ffff 000001 000000 000000 000000
  665   1468  0040ee  9205  ------ --- --------  000000 000000 000000 00ffff 000001 000000 000000 000000
```

各行の後半(SP値～ソースコード)

```
  sp ie/il/cvzn   src mix
003ef8 0 0 0010   ld        %r2,0x0             (main.c)       00012     i = 0;
003ef8 0 0 0010   ext       0x0
003ef8 0 0 0010   ext       0x0
003ef8 0 0 0010   ld        [0x0],%r2
003ef8 0 0 0010   ld        %r4,%r2             (main.c)       00014     for( j = 0; j < 6; ++j )
003ef8 0 0 0010   ld        %r0,%r4             (main.c)       00016         sub(j);
003ef8 0 0 0010   ext       0x0
003ef4 0 0 0010   call      0x5
003ef4 0 0 0010   and       %r0,0x1             (main.c)       00022     if(k & 0x1)
003ef4 0 0 0010   cmp       %r0,0x0
003ef4 0 0 0010   jreq      0xe
003ef8 0 0 0010   ret                           (main.c)       00027 }
003ef8 0 0 0000   add       %r4,0x1             (main.c)       00014     for( j = 0; j < 6; ++j )
003ef8 0 0 1001   cmp       %r4,0x5
```

注 意

- 本コマンドはICD Miniモードでは使用できません。
- 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、"10.9 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(`c17 tm off`を実行)してから再設定してください。

10.7.14 シミュレーテッドI/Oコマンド

c17 stdin（データ入力シミュレーション） [ICD Mini / SIM]

機 能

データをファイルまたは[コンソール]ビューから入力して、プログラムに渡すための設定を行います。
c17 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 シミュレーテッドI/O"を参照してください。

書 式

c17 stdin 1 ***BreakAddr BufferAddr*** ［***Filename***］ （設定）
c17 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
省略すると、[コンソール]ビューからの入力となります。
条件: 0≦*BreakAddr*≦0xffffff、0≦*BufferAddr*≦0xffffff

入力例

■例1

```
(gdb)
c17 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)
c17 stdin 1 READ_FLASH READ_BUF
```

[コンソール]ビューを使用する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、[コンソール]ビューを表示します。ウィンドウにデータを入力して[Enter]キーを押すと、デバッガは入力した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3

```
(gdb)
c17 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

- c17 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c17 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。
- *BreakAddr*と*BuffAddr*は、RAM上のアドレスを指定してください。
*BreakAddr*をROM上に指定すると、デバッガがソフトPCブレーク命令を*BreakAddr*に書き込むため正常に動作しません。

c17 stdout（データ出力シミュレーション） [ICD Mini / SIM]

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。

c17 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 シミュレーテッドI/O"を参照してください。

書 式

c17 stdout 1 ***BreakAddr BufferAddr*** ［***Filename***］ （設定）
c17 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≦*BreakAddr*≦0xffffff、0≦*BufferAddr*≦0xffffff

入力例

■例1

```
(gdb)
c17 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、デバッガ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)
c17 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、デバッガ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)
c17 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

- c17 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c17 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。
- *BreakAddr*と*BuffAddr*は、RAM上のアドレスを指定してください。
*BreakAddr*をROM上に指定すると、デバッガがソフトPCブレーク命令を*BreakAddr*に書き込むため正常に動作しません。

10.7.15 フラッシュライタコマンド

c17 fwe（プログラム / データ / セキュリティ設定の消去） [ICD Mini]

機 能

S5U1C17001Hのフラッシュライタ機能用のコマンドです。
S5U1C17001Hにロードしたデータ消去/書き込みプログラム、書き込みデータとアドレス情報、またはセキュリティ設定を消去します。

書 式

```
c17 fwe 0     （書き込みデータの消去）
c17 fwe 1     （データ消去/書き込みプログラムの消去）
c17 fwe 2     （セキュリティ設定の消去）
```

使用例

■例1

```
(gdb)
c17 fwe 0
```

S5U1C17001H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)
c17 fwe 1
```

S5U1C17001H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。

■例3

```
(gdb)
c17 fwe 2
```

S5U1C17001H内のフラッシュライタ機能用セキュリティ設定を消去します。

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。
- ICD Mini（S5U1C17001H）用のコマンドです。ICDボードでは使用しても正常に動作しません（エラーも表示されません）。
- 機種によってはフラッシュの消去範囲を任意に設定できないものがあります。その機種では全ブロック消去に設定してください。フラッシュの消去範囲を任意に設定できるかどうかは、各機種のflsフォルダ（¥gnu17¥mcu_model¥機種名¥fls）内readme_x.txtを参照してください。

c17 fwlp（プログラムのロード） [ICD Mini]

機 能

S5U1C17001Hのフラッシュライタ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S5U1C17001Hにロードします。

書 式

c17 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。
- フラッシュプログラミング電源端子があるマイコンでは、コメントに書き込み電圧と消去電圧を以下の形式で記載する必要があります。上記端子が無いマイコンでは、本記述を行っても電圧設定はされません。

"-v*EraseVoltage*-*WriteVoltage*" (空白は含みません)

条件: 0≦*EraseEntryAddr*≦0xfffffe（A0 = 0 ）、0≦*WriteEntryAddr*≦0xfffffe（A0 = 0 ）、
0≦コメントサイズ≦127バイト
6.0 ≦ *EraseVoltage* ≦ 8.0V
6.0 ≦ *WriteVoltage* ≦ 8.0V

使用例

■例1

```
(gdb)
c17 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機能用のデータ消去/書き込みプログラムをS5U1C17001H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。

■例2

```
(gdb)
c17 fwlp writer.sa 0x90 0xb4 "-v7.0-7.0"
```

フラッシュプログラミング電源端子があるマイコンの消去電圧を7.0V、書き込み電圧を7.0V に設定しています。

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。
- ICD Mini（S5U1C17001H）用のコマンドです。ICDボードでは使用しても正常に動作しません（エラーも表示されません）。
- データ消去/書き込みプログラムのサイズは8KB以下です。
- *EraseEntryAddr*/*WriteEntryAddr*の値については、mcu_model\機種名\fls 内の"readme_j.txt"を参照してください。
- コメントによるフラッシュプログラミング電源設定をするためには、フラッシュプログラミング電源端子があるマイコンと、ICDmini Ver.2.0(S5U1C17001H2100) が必要です。
設定値は機種によって異なります。各機種のテクニカルマニュアルに記載されている"電気的特性"を参照してください。

c17 fwld（データのロード） [ICD Mini]

機 能

S5U1C17001Hのフラッシュライタ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S5U1C17001Hにロードします。

書 式

c17 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）
EraseEndBlosk: 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≦*EraseStartBlock*≦0xffffffff、0≦*EraseEndBlock*≦0xffffffff
EraseStartBlock = *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≦コメントサイズ≦127バイト

使用例

```
(gdb)
c17 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S5U1C17001Hにロードします。フラッシュ書き込みデータは、モトローラS3形式ファイル（.psa、.sa、.saf など）を指定してください。

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。
- ICD Mini（S5U1C17001H）用のコマンドです。ICDボードでは使用しても正常に動作しません（エラーも表示されません）。
- プログラムデータサイズは4MB以下です。
- 機種によってはフラッシュの消去範囲を任意に設定できないものがあります。その機種では全ブロック消去に設定してください。フラッシュの消去範囲を任意に設定できるかどうかは、各機種のflsフォルダ（¥gnu17¥mcu_model¥機種名¥fls）内readme_x.txtを参照してください。

c17 fwdc（ターゲットメモリのコピー） [ICD Mini]

機 能

S5U1C17001H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S5U1C17001Hにロードします。

書 式

c17 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）
EraseEndBlosk: 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≦*SourceAddr*≦0xfffffe（A0 = 0）、0≦*Size*≦0xfffffe（D0 = 0）、
0≦*EraseStartBlock*≦0xffffffff、0≦*EraseEndBlock*≦0xffffffff
EraseStartBlock = *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≦コメントサイズ≦127バイト

使用例

```
(gdb)
c17 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S5U1C17001Hにコピーします。

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。
- ICD Mini（S5U1C17001H）用のコマンドです。ICDボードでは使用しても正常に動作しません（エラーも表示されません）。
- ICDminiのフラッシュに維持できるデータは4MB以下です。

c17 fwd（フラッシュライタ情報の表示） [ICD Mini]

機 能

S5U1C17001Hのフラッシュライタ機能用のコマンドです。
S5U1C17001H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

c17 fwd

表 示

```
(gdb)
c17 fwd
CPU data address            : xxxxxxxx
Data size                   : xxxxxxxx
Erase start block           : xxxxxxxx
Erase end block             : xxxxxxxx
Erase parameter             : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

CPU program address         : xxxxxxxx
Program size                : xxxxxxxx
Erase routine entry address : xxxxxxxx
Write routine entry address : xxxxxxxx
Comment : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

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、c17 fwldまたはc17 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、c17 fwlpコマンドで設定された内容です。

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。
- ICD Mini（S5U1C17001H）用のコマンドです。ICDボードでは使用しても正常に動作しません（エラーも表示されません）。
- c17 fwpwコマンドによるセキュリティ設定は表示されません。

c17 fwpw（フラッシュライタのセキュリティ設定） [ICD Mini]

機 能

S5U1C17001Hのフラッシュライタ機能用のコマンドです。
Flashセキュリティ対応機種でFlashセキュリティが設定されているとき、S5U1C17001Hの
フラッシュライタ機能がFlashセキュリティを解除するための情報を設定します。

書 式

```
c17 fwpw McuModel Version Password
```

McuModel: 機種名（文字列）
\gnu17\mcu_model以下にある機種別情報ファイルのフォルダ名を指定します。

Version: Flashセキュリティのバージョン（文字列）
例：
M03：Flashセキュリティのバージョン3を表す値

Password: パスワードの値（文字列）。
英数字（A-Z、a-z、0-9）を指定します。
有効な文字数は、Versionの値により変わります。

使用例

```
(gdb)
c17 fwpw 17001 M03 ABCD1234
Load flash security parameters ... done
```

フラッシュライタ機能がFlashセキュリティを解除するための情報をS5U1C17001Hに設定します。
このコマンドに与えるバージョンとパスワードは、c17 pwulコマンドに与える値と同じです。

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。
- ICD Mini(S5U1C17001H）用のコマンドです。ICDボードでは使用しても正常に動作しません（エラーも表示されません）。
- 本コマンドによる設定はc17 fwdコマンドの表示に反映されません。
- 本コマンドによる設定はc17 fwe 2コマンドの実行により消去されます。

10.7.16 プロファイラ・カバレッジコマンド

c17 profilemd（プロファイル・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

c17 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、"C17 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、"C17 command error, invalid parameter."と表示されます。

使用例

```
(gdb)
c17 profilemd 0
Profiler mode is disabled.
(gdb)
c17 profilemd 1
Profiler mode is enabled.
(gdb)
 c17 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
C17 rst
Cont
Break発生
C17 profilemd 0
Cont
Break発生
C17 profilemd 1
（この間の計測データが無い。）
Cont
Break発生
C17 profile ← ここでプロファイル計測は正しく行われません。

c17 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、「10.8　プロファイラ・カバレッジ機能」を参照してください。

書　式

c17 profile

使用例

```
(gdb)
c17 rpf c17.par
(gdb)
file sample.elf
(gdb)
target sim
(gdb)
load
(gdb)
c17 rst
(gdb)
break _exit
(gdb)
cont                ←  計測開始
ここでブレーク。    ←  計測停止
(gdb)
c17 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Profiler Window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。
- c17 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。

c17 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、「10.8 プロファイラ・カバレッジ機能」を参照してください。

書 式

```
c17 coverage
```

使用例

```
(gdb)
c17 rpf c17.par
(gdb)
file sample.elf
(gdb)
target sim
(gdb)
load
(gdb)
c17 rst
(gdb)
break _exit
(gdb)
cont                ←  計測開始
ここでブレーク。    ←  計測停止
(gdb)
c17 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Coverage Window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。
- `c17 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

set output-radix（変数表示形式の変更） [ICD Mini/SIM]

機　能

[ソース]エディター、[式]ビュー、[変数]ビューおよびprintコマンドで変数を表示する形式を変更します。
変更可能な形式は、16進数/10進数/8進数です。
ただし、変数が浮動小数点やポインタの場合は、表示形式は変更しません。
変更した形式は、デバッガ終了時記憶されず、次回GDB起動時には デフォルト(10進数)の表示形式で変数を表示します。

書　式

set output-radix Type

Type: 表示形式
　　16=16進数
　　10=10進数（デフォルト）
　　8=8進数

使用例

```
(gdb)
print i
$1 = -21846
(gdb)
set output-radix 16
(gdb)
print i
$2 = 0xaaaa
(gdb)
set output-radix 8
(gdb)
print i
$3 = 0125252
```

注　意

- 2 進数を設定した場合（`set output-radix 2`）はデバッガの表示が正しく表示されません。

c17 log（ロギング） [ICD Mini / SIM]

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

c17 log *Filename* （ロギング開始）
c17 log （ロギング終了）

Filename: ログファイル名

使用例

■例1

```
(gdb)
c17 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c17 logコマンドの実行まで有効です。

■例2

```
(gdb)
c17 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、c17 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c17 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。c17 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） [ICD Mini / SIM]

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 = src.cmd

```
# load symbol information
file /cygdrive/c/EPSON/gnu17/sample/tst/sample.elf
#connect to the debugger with specified mode and port
target sim
# load to memory
load /cygdrive/c/EPSON/gnu17/sample/tst/sample.elf
# reset
c17 rst
```

#から行の終わりまではコメントとみなされます。

```
(gdb)
source src.cmd
(gdb)
(gdb)
file /cygdrive/c/EPSON/gnu17/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/gnu17/sample/tst/sample.elf
Loading section .text, size 0xbc lma 0xc00000
Start address 0xc00000
Transfer rate: 1504 bits in <1 sec.
(gdb)
(gdb)
c17 rst
CPU 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。
- コマンドファイル内に source コマンドがあるというように source コマンドのネストが可能です。ネスト数には制限はありません。
- コマンドファイル等でif文等の制御命令はサポートしません。

c17 clockmd（実行カウンタモード設定）
c17 clock　（実行カウンタ表示）　[ICD Mini / SIM]

機　能

c17 clockmd: 実行カウンタのモード設定を行います。
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。

c17 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。
ICD Miniモードでは、ICDの実行カウンタ値を時、分、秒、ミリ秒、マイクロ秒で表示します。
シミュレータモードでは、サイクル数を表示します。

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

c17 clockmd ***Mode***　（実行カウンタモード設定）
c17 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）

使用例

■例1（シミュレータモード）

```
(gdb)
c17 clockmd 2
(gdb)
c17 rst
CPU resetting ..................... done
(gdb)
continue
Continuing.

Breakpoint 1, sub (k=0) at main.c:20
(gdb)
c17 clock
           218 cycle
(gdb)
continue
Continuing.

Breakpoint 1, sub (k=1) at main.c:20
(gdb)
c17 clock
           330 cycle
```

積算モードに設定後、リセットコマンド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（ICD Miniモード）

```
(gdb)
c17 clockmd 1
(gdb)
c17 rst
CPU resetting ..................... done
(gdb)
continue

Breakpoint 1, sub (k=0) at main.c:20
(gdb)
c17 clock
           0 hour 0 min 1 sec 23 ms 1.33 us
(gdb)
continue
Continuing.

Breakpoint 1, sub (k=1) at main.c:20
(gdb)
c17 clock
           0 hour 1 min 53 sec 0 ms 0 us
```

こちらの例ではリセットモードに設定後、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。

注　意

- ICD Miniモード時は、continue/untilコマンド実行後にのみ本コマンドが有効となります。
  シミュレータモード時はいつでも有効です。
- 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。
- 時間経過後ブレーク（c17 timebrk）の設定が有効な間は、c17 clockコマンドで実行時間の表示はできません。実行時間を計測する場合は、タイマブレーク設定を無効（c17 timebrk 0）にしてください。
- 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c17 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
  3. CPUをリセットした場合
  4. ICD Miniモードでc17 timebrk、step、stepi、next、nexti、finishコマンド実行をした場合
- IICD Miniモード時の制限
  1. 計測可能な時間は最大6515時間です。
  2. ステップ実行コマンド(step、next等)とステップ・リターン(finish)後の計測表示はできません。
  3. 実行時間が短い場合(3マイクロ秒以下の命令実行時)は、計測できないことがあります。
  4. ICD Mini に搭載される発振子の精度と、デバッグモードに出入りする処理が含まれるため、計測結果は以下の誤差が含まれて表示されます。
     計測結果 ＝ 実際にかかった時間(±50ppm) + デバッグモード出入処理（約40サイクル）
- シミュレータモード時の制限
  シミュレータモードで計測されたサイクル数は、実際のハードウェアのサイクル数と比べ、10%程度の誤差を含みます。

target（ターゲットの接続） [ICD Mini / SIM]

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。
ICD Miniモード: ICD Mini（S5U1C17001H）またはICDボードとUSBインタフェースで接続します。
シミュレータモード: デバッガをシミュレータモードに設定します。

書 式

target ***Type***

Type: ターゲットを指定する以下のシンボル

icd usb	ICDとUSBインタフェースで接続（ICD Miniモード）
icd usb2	ICDとUSBインタフェースで接続（ICD Miniモード） 1台のPCと2台のICDをUSB接続してデバッグするときに使用します。 デバッガは`--c17_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でデバッガを2つ同時に起動できます。
sim	シミュレータを起動（シミュレータモード）

使用例

■例1

```
(gdb)
target sim
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
```

ICD Miniモードに設定します。

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、`c17 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)
file sample.elf          （デバッグ情報の読み込み）
(gdb)
c17 rpf sample.par       （マップ情報設定）
(gdb)
target icd usb           （本コマンド）
(gdb)
load                     （プログラムのロード）
(gdb)
c17 rst                  （リセット）
```

detach（ターゲットの切断） [ICD Mini / SIM]

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
```

ICD Mini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ICD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。

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） [ICD Mini / SIM]

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/gnu17/sample/tst.
(gdb)
cd /cygdrive/c/EPSON/gnu17/sample/ansilib
Working directory /cygdrive/c/EPSON/gnu17/sample/ansilib.
```

カレントディレクトリを確認後c:¥EPSON¥gnu17¥sample¥ansilib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。

c17 firmupdate（ファームウェア更新） [ICD Mini]

機 能

ICD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の電源を入れ直してください。

書 式

c17 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)
c17 firmupdate icd17dmt.sa
```

ICDと接続後、ファームウェアファイルicd17dmt.saでICDのファームウェアを更新します。

注 意

c17 firmupdateコマンドはICD Miniモード時のみ有効です。

c17 ttbr（TTBR設定） [SIM]

機 能

TTBRにアドレスを設定します。
リセットコマンド(`c17 rst`)を実行したときに、TTBRの示すアドレス内の値(リセットベクタ)がPCに設定されます。
パラメータファイルのTTBR行と同じ機能です。

書 式

`c17 ttbr` ***Address***

Address: TTBRに設定するアドレス(10進数、16進数、またはシンボル)
条件: 0≦*Address*≦0xffff00(*Address*の下位8ビットは0x00である必要があります。)

使用例

```
(gdb)
c17 ttbr 0x8000
```

TTBRに0x8000番地を設定します。

注 意

- 本コマンドは、シミュレータモード時のみ使用可能です。
- 本コマンドは、`target`コマンドの前に実行する必要があります。

c17 help（ヘルプ） [ICD Mini / SIM]

機 能

コマンド説明を表示します。

書 式

c17 help [***Command***]
c17 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)
c17 help
group  0: memory ........... c17 fb,c17 fh,c17 fw,x /b,x /h,x /w,set {char},set
                             {short},set {int},c17 mvb,c17 mvh,c17 mvw,c17 df,c17
                             readmd,c17 loadmd
group  1: register ......... info reg,set $
group  2: execution ........ continue,until,step,stepi,next,nexti,finish,c17
                             callmd,c17 call
group  3: CPU reset ........ c17 rst, c17 rstt
group  4: interrupt ........ c17 int,c17 intclear,c17 int_load
group  5: break ............ break,tbreak,hbreak,thbreak,delete,clear,enable,
                             disable,ignore,info breakpoints,c17 timebrk,c17
                             hbreakmd
group  6: symbol ........... info locals,info var,print
group  7: file ............. file,load
group  8: map .............. c17 rpf,c17 map,c17 memwait,c17 ttbr
group  9: flash memory ..... c17 fls,c17 fle
group 10: trace ............ c17 tm
group 11: simulated I/O .... c17 stdin,c17 stdout
group 12: flash writer ..... c17 fwe,c17 fwlp,c17 fwld,c17 fwdc,c17 fwd,c17 fwpw
group 13: profiler.......... c17 profilemd,c17 profile,c17 coverage
group 14: others ........... c17 log,source,c17 clockmd,c17 clock,c17 firmupdate,
                             target,detach,c17 savewatch,c17 loadwatch,c17 chgclkmd,
                             c17 pwul,set output-radix,pwd,cd,c17 help,quit
 Please type "c17 help 1" to show group 1 or type "c17 help c17 fb" to get usage of
command "c17 fb".
```

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)
c17 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
 c17 callmd         Set user function call mode
 c17 call           Call user function
 Please type "c17 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)
c17 help step
step: Single-step,every line                                    [ICD/SIM]

usage: step [Count]
       Count: Number of steps to execute (decimal or hexadecimal)
              One step is assumed if omitted.
       Conditions: 1-0x7fffffff

example:
(gdb)
step
(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)
c17 help c17 rst
c17 rst: Reset                                                  [ICD/SIM]

usage: c17 rst

example:
(gdb)c17 rst
The CPU is reset.
```

C17系のコマンドの詳細説明を表示する場合は、"c17"の後に続けてコマンドを指定します。

注 意

- c17 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"[ICD/SIM]"は、コマンドが有効なモードを示しています。
  ICD: ICD Miniモードで有効(ICD Mini(S5U1C17001H)またはICDボード使用時)
  SIM: シミュレータモードで有効(PCのみでのデバッグ時)

  "[ICD]"のような表示は、そのコマンドがICD Miniモード以外では使用できないことを意味します。

c17 chgclkmd（DCLK 切替モード） [ICD Mini]

機　能

ブレーク時のDCLK(デバッグクロック)切替モードを設定します。
切替有効の場合：ブレーク時に、DCLKが低速クロックの場合に高速クロックに自動的に変更します。プログラムを再開する場合に元のDCLKに戻します。
切替無効の場合：ブレーク時、DCLKは変更しません。

書　式

c17 chgclkmd [***Mode***]

Mode:　0 DCLK 切替モード有効（デフォルト）
1 DCLK切替モード無効
省略した場合は現在のモードを表示します。
Mode = 0の場合："DCLK change mode ON."
Mode = 1の場合："DCLK change mode OFF."

使用例

＜ターゲットCPUがS1C17702の場合＞

```
(gdb)
 continue
    ターゲットプログラム内でOSC1に設定
    プログラム内でブレーク発生      ・・・切替モード有効なのでHSCLKに切り替える
(gdb)
 finish                             ・・・OSC1に戻して実行する
    finish終了                      ・・・HSCLKに切り替える
(gdb)
 c17 chgclkmd 1                     ・・・切替モード無効にして、OSC1に戻す
 DCLK change mode OFF.
(gdb)
 continue                           ・・・切替モード無効なのでOSC1のまま実行する
    プログラム内でブレーク          ・・・切替モード無効なのでクロック切り替えはしない
(gdb)
```

注　意

- c17 chgclkmdコマンドはICD Miniモード時のみ使用可能です。
- c17 chgclkmdコマンドは対応するターゲットCPUのみ使用可能です。
  対応しないターゲットCPUで、本コマンドを実行した場合下記のエラーをコンソールに出力します。
  "C17 command error, command is not supported in present target CPU."
- c17 chgclkmdを使用する場合、c17 rpfコマンドが実行されている必要があります。
- クロックソースの切り替えは、step/stepiコマンドと関数コール以外のnext/nextiコマンドでは動作しません。この場合、クロックの切り替えはしません。
- *Mode* = 0のとき、以下の使い方をするとクロック切換えが正常に動作しなくなることがあります。
  1．ブレーク中のクロック制御レジスタ、クロックソースレジスタの書き換え
  2．ターゲットプログラムのクロック切り替え中にブレーク
  3．ターゲットプログラムのクロック切り替え部分をステップ実行
  以上のような使い方をする場合は*Mode* = 1にしてください。

c17 pwul（Flash セキュリティ・パスワードの解除） [ICD Mini]

機　能

Flash セキュリティ対応機種でパスワードが設定されているとき、パスワードの解除を行います。

書　式

```
c17 pwul Version Password
```

Version: Flash セキュリティのバージョン（文字列）
例：
M03：Flash セキュリティのバージョン３を表す値

Password: パスワードの値。
英数字（A-Z、a-z、0-9）を指定します。有効な文字数は、*Version* の値により変わります。

使用例

```
(gdb)
c17 pwul M03 ABCD1234
Unlock flash security password was setted.
```

設定済のパスワード“ABCD1234”でパスワードの解除を行います。

注　意

- Flash セキュリティ未対応機種のとき、エラーになります。
- 未定義の*Version*を指定するとエラーになり、パスワードの解除はできません。
- *Password*に無効な文字列(英数字以外や無効な文字数)を指定するとエラーになり、パスワードの解除はできません。

quit（デバッガ終了） [ICD Mini / SIM]

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 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します。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します。

10.8.1 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計測
プロファイル計測　　：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"c17 profile"コマンド、[デバッグ]ビューの[プロファイラ]ボタン、またはコンテキストメニューの[プロファイラ]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PATH格納ファイル(c17_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"c17 coverage"コマンド、[デバッグ]ビューの[カバレッジ]ボタン、またはコンテキストメニューの[カバレッジ]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PATH格納ファイル(c17_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)

※ これらのツールは、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されません。
※「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全て含むファイルで、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。

10.8.2 機能一覧

主要な機能一覧を以下に示します。

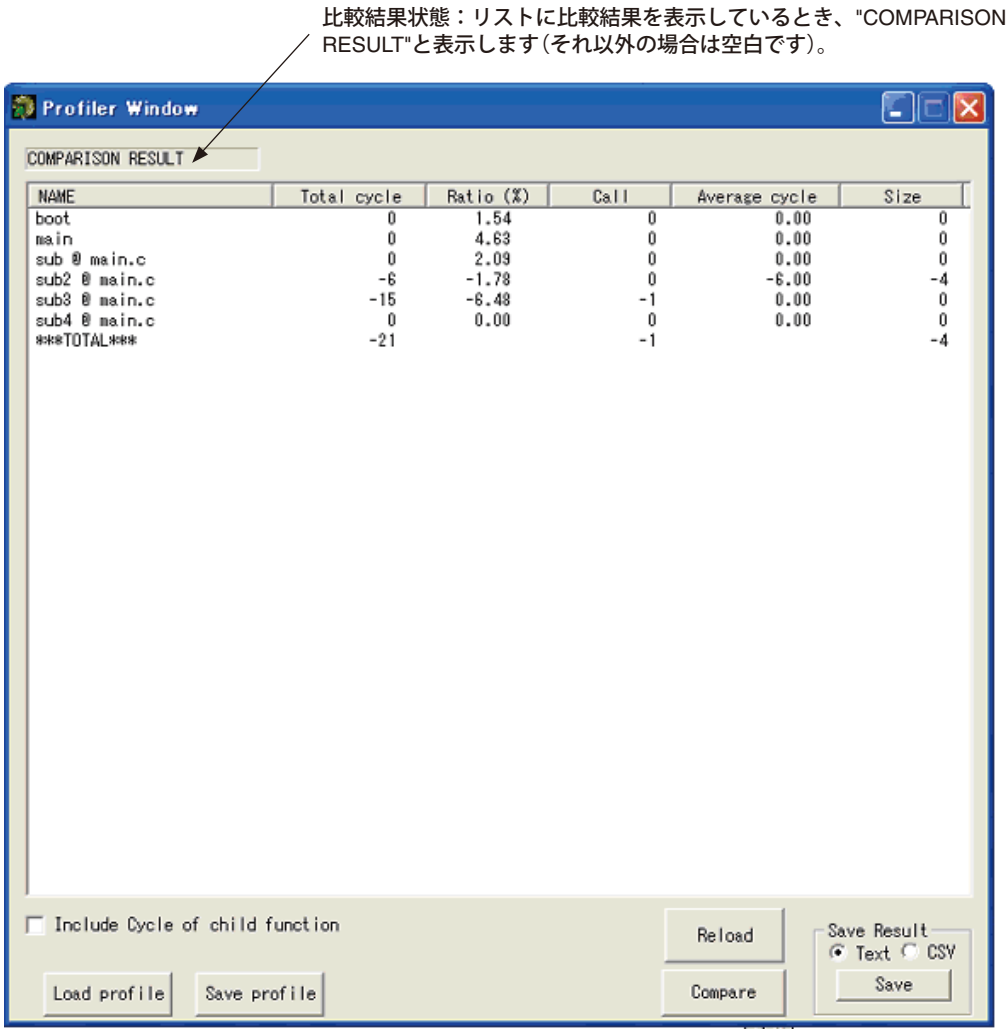
表10.8.2.1 機能一覧

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どうかを指定可能 (c17 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 (c17 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 (c17 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 or 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 or 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

10.8.3 機能詳細

10.8.3.1 プロファイラ・ウィンドウ

"c17 profile"コマンド、[デバッグ]ビューの[プロファイラ]ボタン、またはコンテキストメニューの[プロファイラ]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。計測データが無い場合は、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。



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、計測データが無いとき、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10.8.3.1.1 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

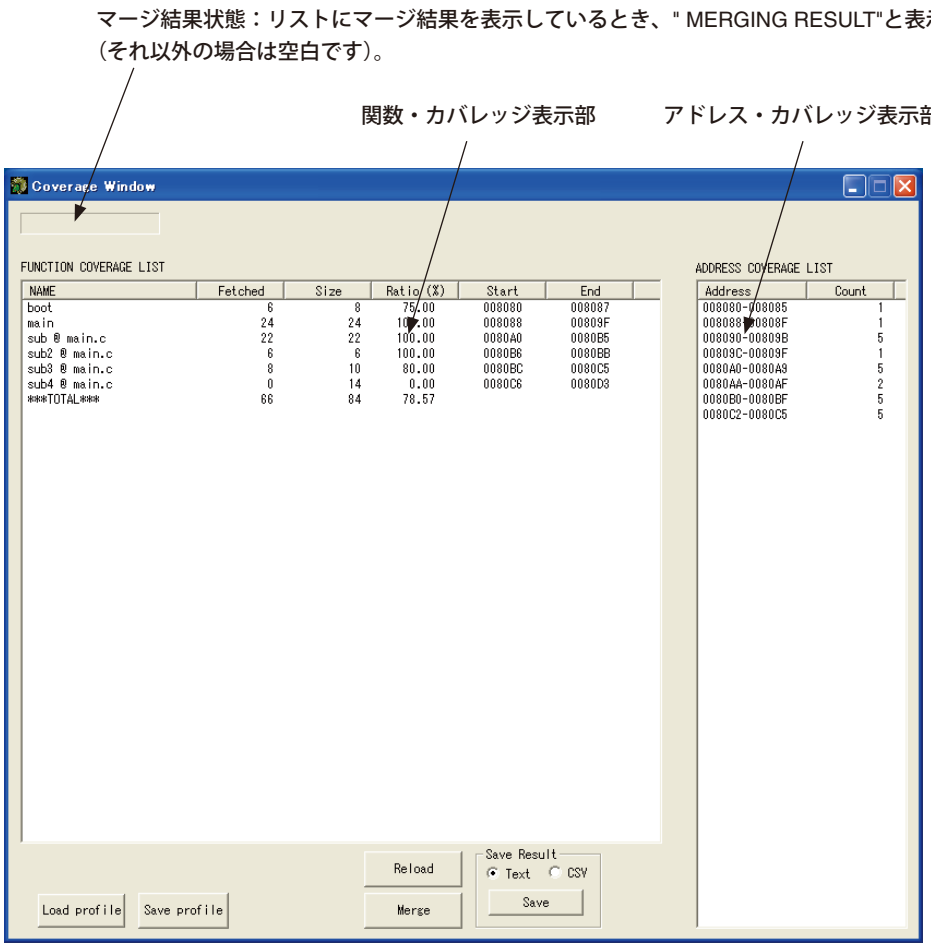
```
--------------------------------------------------------------------------------
Profile result

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 ***
--------------------------------------------------------------------------------
```

10.8.3.2 カバレッジ・ウィンドウ

"c17 coverage"コマンド、[デバッグ]ビューの[カバレッジ]ボタン、またはコンテキストメニューの[カバレッジ]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。計測データが無い場合は、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。



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：%、少数点以下第2位まで表示)
Start： 関数の開始アドレス(16進数、0～FFFFFF)
End： 関数の終了アドレス(16進数、0～FFFFFF)

・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～FFFFFF)
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全て一致しない場合、エラーとなります。また、全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
--------------------------------------------------------------------------------
```

10.8.3.3 プロファイル・ウィンドウとカバレッジ・ウィンドウが表示するエラーメッセージ

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、計測結果はなにも表示しません。
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10.8.3.3.1 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 C17 architecture.	指定したファイルはC17用ではありません。IDが"C17PROF"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 addressed 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宣言が無いなど。「10.8.8.4 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。

10.8.3.4 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、"c17 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"c17 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ラは取れません。
- 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。
- デバッグ情報のない(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。

10.9 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。デバッガ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ブレー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ーク(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)
- リセット時にTTBRを参照
- CPU機種名(ICD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、c17 rpfコマンドの実行によってデバッガに読み込まれます。

(gdb)
c17 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、デバッガ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 GNU17 IDE"を参照してください。

c17 rpfコマンドは、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)
file sample.elf          (デバッグ情報の読み込み)
(gdb)
c17 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)
c17 rst                  (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[プロパティー]ダイアログボックスから[GNU17 パラメータ設定]を選択して作成できます。**IDE**の詳細については、"5 GNU17 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。

```
#gnu17 gdb parameter file                    (1)
CHIP     S1C17701                            (2)
ESSIM    S1C17701                            (3)

RAM      000000 001fff 00W    #IRAM          (4)
IO       040000 04ffff 00H    #IO
RAM      600000 6fffff 11H    #RAM
ROM      c00000 cfffff 55B  B #ROM
STACK    000000 001fff        #STACK         (5)
```

(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) CHIP CPU名

ICDモードでブレーク時にDCLK(デバッグクロック)を高速に切り替える場合に指定します。
指定していない場合、DCLKを高速に切り替える処理は行いません。

(3) ターゲット機種

ES-Sim17によってシミュレートする機種名を指定します。

(4) 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。

BigEndian ビッグエンディアンのデバイスの場合に**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。
ICD Mini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。デバッガ起動時に領域の確保ができないことがあります。**

(5) 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。デバッガ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。

シミュレータモード時

- `c17 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、0x0～0xffffff(16MB)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。また、TTBRのアドレスは0x8000に設定されます。
- 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ト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。

10.10 ステータス/エラーメッセージ

10.10.1 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10.10.1.1 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 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
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
Break by key break.	[中断]ボタンによる強制ブレーク(シミュレータモード)
Break by key break. Program received signal SIGINT, Interrupt.	[中断]ボタンによる強制ブレーク(ICD Miniモード)

10.10.2 エラーメッセージ

表10.10.2.1 エラーメッセージ(アルファベット順)

メッセージ	内容
· · · gdb : unrecognized option '*option*'	起動時オプションに誤りがあります。
A setup of a serial port was not completed.	**gdb**が対応しないICDモードが指定されました。
Address is 24bit over.	指定したアドレスが、24ビットを超えています。S1C17のアドレスは、最大24ビット(0xffffff)です。
Address(0x*#*) is ext or delayed instruction.	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。
C17 command error, command is not supported in present mode.	指定したコマンドは、現在のモード(ICD Miniまたはシミュレータモード)では対応していません。
C17 command error, command is not supported in present target CPU.	指定したコマンドは、現在のターゲットCPUでは対応していません。
C17 command error, command is not supported in ICDmini hardware Ver xx.	ICDminiハードウェアバージョンxxではサポートしていません。
C17 command error, command is not supported in ICDmini firmware Ver xx.	ICDminiファームウェアバージョンxxではサポートしていません。
C17 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。
C17 command error, invalid command.	コマンドに誤りがあります。
C17 command error, no map area.	指定したコマンドの引数のアドレスはパラメータファイルで指定したアドレスの範囲外です。
C17 command error, invalid parameter.	指定したコマンドのパラメータに誤りがあります。
C17 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。
C17 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。
Cannot access memory.	指定した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 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 display clock counter. Now Timer break mode is on. Please timer break mode off.	時間経過後ブレーク有効時に実行カウンタの表示はできません。時間経過後ブレークを解除してから再度計測してください。
Cannot display clock counter. Time measurement should use continue or until command.	実行カウンタの値は、`continue`または`until`コマンドを実行した後でないと表示できません。
Cannot load to no map memory. (0x#-0x#)	パラメータファイルで指定したアドレス範囲外へのファイルロードはできません。
Cannot measure clock timer.	プログラムの実行時間が短すぎて計測できません。
Cannot open file(*file*).	ファイルがオープンできません。

メッセージ	内容
Cannot open ICD17 usb driver.	USBドライブのオープンに失敗しました。
Cannot set hard pc break.	指定したアドレスにハードブレークは設定できません。
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. 1)。
Cannot set soft pc break.	指定したアドレスにソフトブレークは設定できません。
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 soft pc break at ROM area.	読み取り専用メモリに書き込みはできません。
Cannot set same 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 timer. Timer Conditions: 1<=Timer<=300000	指定時間が条件を超えているので、時間経過後ブレークを設定できません。
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 system error(#).	ICDとの通信中に切断してしまいました。
Copy end address max(0x#) overflow.	コピー元の終了アドレスが最大値(0xffffff)を超えています。
Copy start address max(0x#) overflow.	コピー元の開始アドレスが最大値(0xffffff)を超えています。
Coverage Window is already opened.	カバレッジ・ウィンドウが既に開いています。
CPU is running.	ターゲットCPUがRUN状態のため、コマンドを受け付けることができません。
Erase entry address max(0x#) overflow.	フラッシュ消去ルーチンのアドレスが最大値(0xffffff)を超えています。
Flash memory end address max(0x#) overflow.	フラッシュメモリの終了アドレスが、最大値(0xffffff)を超えています。
Flash memory start address max(0x#) overflow.	フラッシュメモリの開始アドレスが、最大値(0xffffff)を超えています。
ICD17 is busy(#).	ICD側がBUSY状態です。
Illegal instruction.	シミュレータモードにおいて、未定義命令を実行しようとした。
Initialization error of ICD17.	ターゲットの初期化に失敗しました。
Invalid ID error(0x#).	**gdb**が未定義通信ID番号を送信しました。(内部エラー)
Invalid format event file (#).	c17 int_load(割り込みイベントコマンド)で指定したイベントファイルのフォーマットエラーがあります。
Invalid parameter file(#: *file*).	パラメータファイルに誤りがあります。
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 is not c17 architecture ELF file.	`file`コマンドで指定したファイルはS5U1C17001Cが対応しているelf形式ファイルではありません。
Load end address max(0x#) overflow.	フラッシュに書き込むプログラムの終了アドレスが最大値(0xffffff)を超えています。
Load motorola file format error.(*file*)	指定したモトローラ形式ファイルにフォーマットエラーがあります。
Load size limit(0x#) overflow.	指定したファイルサイズが最大値を超えています。 • フラッシュメモリ書き込み/消去プログラムの場合 8Kバイト - 1バイト(0x1fff) • フラッシュメモリへ書き込むデータの場合 3Mバイト - 1バイト(0x2fffff) • ファームウェアの場合 8Mバイト - 1バイト(0x7fffff)
Load start address max(0x#) overflow.	フラッシュに書き込むプログラムの開始アドレスが最大値(0xffffff)を超えています。
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。
Script file error: IF nest max(5) over.	スクリプトファイル(*.Spt)でif文のネスト数が５つを超えている。
Script file format error. (Line no.#)	スクリプトファイル(*.Spt)の#行がフォーマットエラーがあります。
Send Size entry address is out of limit (# - #).	ICDに対して、１パケット辺りに送信するデータサイズが有効範囲内でない。
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。
Specified voltage cannot be output.	指定された電圧は出力できませんでした。

メッセージ	内容
Specified voltage is out of range (6.0V – 8.0V).	指定した電圧値が範囲外です。
Target down.	ICDとターゲット間で通信エラーが発生しました。
There is no argument given to this command.	ターゲットの切断に失敗しました。
Too much event(#).	`c17 int_load`(イベントファイル読み込み)コマンドで指定したイベント数が最大値(256)を超えています。
Transmitting failure(#).	ICDとの通信でICDからNAKを受け取りました。
USB communication error(host->ICD17).	ICDへのUSB送信に失敗しました。
USB communication error(ICD17->host).	ICDからのUSB受信に失敗しました。
Write entry address max(0x#) overflow.	フラッシュ書き込みルーチンのアドレスが最大値(0xffffff)を超えています。
Target CPU does not support flash security.	ターゲットCPUはFlash セキュリティに対応していません。
Cannot unlock flash security password. Flash security version is not available.	Flash セキュリティバージョンが無効なため、パスワードの解除ができません。
Cannot unlock flash security password. Password format is not available.	パスワードの書式が無効なため、パスワードの解除ができません。
Target CPU does not support specified flash security version.	未定義のターゲットCPU、または未定義のフラッシュセキュリティバージョンを指定されました。
Password format is not available.	パスワードの書式が無効です。

ICD: ICD Mini（S5U1C17001H）またはICDボード

10.11 組み込みシステムシミュレータ(ES-Sim17)

組み込みシステムシミュレータ(**ES-Sim17**)は、PC上でS1C17チップのハードウェアをシミュレートする機能を提供します。デバッガ**gdb**のシミュレータモードに連動して動作しますので、より実際的なアプリケーションシステムのデバッグがPCのみで行えます。
ES-Sim17の機能は以下のとおりです。
1. 汎用ポートの出力状態を表示するとともに入力状態をシミュレート
2. SVD動作を評価するため、電源電圧値を設定可能
3. ターゲット機種の内蔵LCDドライバによるLCDパネルの表示をシミュレート
   ※ポート数、SVD有り/無し は機種によって決定されます。

操作と表示は、すべて次の[ES-Sim]ウィンドウで行います。

[ES-Sim]ウィンドウ(S1C17701の例)

OSC1クロック動作はリアルタイムに実行可能です。OSC3クロック動作については、"simulator_readme.txt"を参照してください。

注: PC上でのシミュレーションのため、制限事項があります。"10.11.8 制限事項"および"simulator_readme.txt"を参照してください。

10.11.1 入出力ファイル

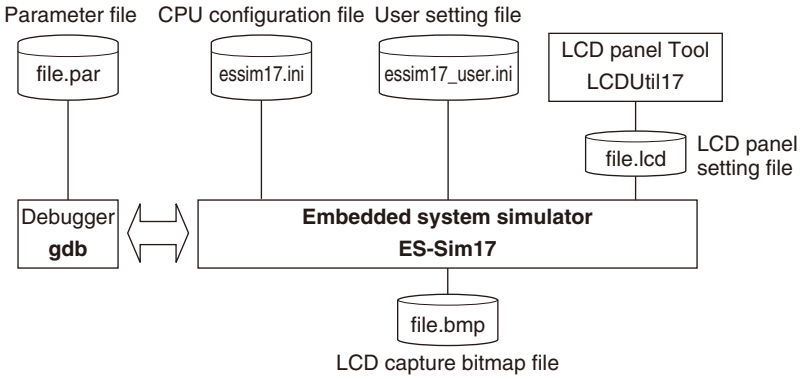


図10.11.1.1 入出力ファイル

●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: デバッガ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。("10.9 パラメータファイル"参照)
ES-Sim17は、デバッガが読み込んでいるパラメータファイルからシミュレートするターゲット機種名を取得します。

CPU構成ファイル

ファイル形式: テキストファイル
ファイル名: essim17.ini(固定)
内容: **ES-Sim17**でシミュレートするターゲット機種のハードウェア構成を記述したファイルです。

注: ES-Sim17が正常に動作しなくなりますので、このファイルは変更しないでください。

ユーザ設定ファイル

ファイル形式: テキストファイル
ファイル名: essim17_user.ini(固定)
内容: ES-Sim17を使用できるCPU機種を選択してプロジェクトを作成した場合、プロジェクトフォルダ内に作成されるファイルです。ユーザが設定可能な値を記述しておきます。

以下の設定が可能です。

- OSC1とOSC3クロックの周波数　Hz単位で記述してください。
- LCDファイルパス　LCDファイルが無い場合は空白にしてください。ES-Sim ウィンドウから設定することも可能です。

```
例: ;;UserSetting                    ';'で始まる行はコメントと見なされます。
   ;oscillator clock [Hz]
   [osc]                             OSCクロック設定フィールド
   osc1=32768                        OSC1クロック周波数 = 32.768kHz
   osc3=4000000                      OSC3クロック周波数 = 4MHz
   ;lcd file path
   [LCD FILE PATH]                   LCDファイルパス設定フィールド
   path=C:¥EPSON¥GNU17¥tools¥LcdUtil17¥sample_lcd¥SVT17701.lcd
                                     LCDファイルパス
```

LCDパネル設定ファイル
ファイル形式: バイナリファイル
ファイル名: <ファイル名>.lcd
内容: LcdUtil17で作成した、ES-Sim17用LCDパネル設定ファイルです。
ES-Sim17でドットマトリクスLCD、セグメントLCDのシミュレートを行えます。

●出力ファイル

LCDキャプチャビットマップファイル
ファイル形式: ビットマップファイル
ファイル名: <*ファイル名*>.bmp
内容: **ES-Sim17**で作成する、シミュレートしたLCD画面のビットマップファイルです。

10.11.2 起動と終了

●ES-Sim17の起動

ES-Sim17は以下の2つの条件が成立した場合にデバッガによって起動します。

1. デバッガが読み込んだパラメータファイルに次の記述が存在する。
   `ESSIM` *<機種名>*

   例: `ESSIM S1C17701`

   IDEが生成するシミュレータモード用のパラメータファイルには、ターゲットプロセッサの選択に従って挿入されます。
2. `target sim`(シミュレータモード設定)コマンドが実行される。

ES-Sim17が起動すると、[ES-Sim]ウィンドウが表示されます。

●ES-Sim17の終了

ES-Sim17が終了するのは、次の2つの場合です。

1. デバッガの`detach`コマンド実行時
2. デバッガの終了時

●[ES-Sim]ウィンドウのオープン/クローズ

[ES-Sim]ウィンドウは[閉じる]ボタンで、閉じることができます(この操作で**ES-Sim17**は終了しません)。
再度開くには、**¥essim17¥EssWnd.exe**を実行します。ただし、**ES-Sim17**が終了していないことが前提です。**ES-Sim17**が終了している場合は、再度`target sim`コマンドを実行する必要があります。
[ES-Sim]ウィンドウは1つのみ開くことができます。すでに開いている場合に再度開く操作を行うと、ウィンドウが最前面に移動しますが、新たなウィンドウは開きません。

10.11.3 メニュー

[File]メニュー

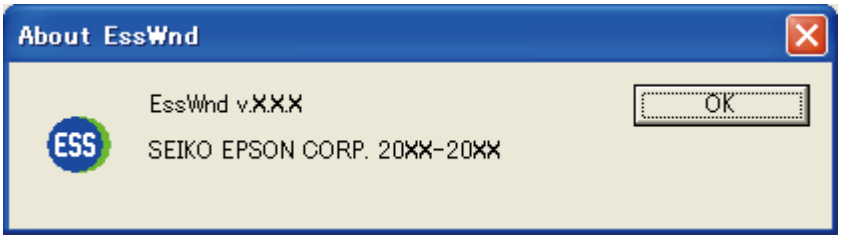
[Load lcd file]

LCDUtil17で作成したLCDファイル(.lcd)を開きます。
LCDUtil17、LCDファイルについては 12.8 LCDUtil17(LCDパネルカスタマイズツール)を参照ください。

[Help]メニュー

[About EssWnd]

ES-Sim Windowのバージョン情報を表示します。

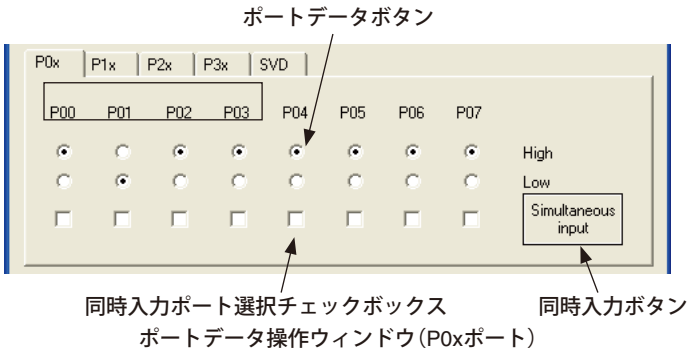


10.11.4 入出力ポートシミュレーション

汎用入力に設定したポートの入力状態を[ES-Sim]ウィンドウで制御することができます。また、汎用出力に設定したポートについては、出力状態が確認できます。

●ポートデータ操作ウィンドウ

操作ウィンドウ選択タブで、操作/表示するポート系列(P0x、P1x、P2x、P3x)を選択します。



ポートデータ操作ウィンドウ(P0xポート)

ES-Sim17は、デバッガを介してPC上のエミュレーションメモリから入出力ポートの機能選択および入出力方向の情報を取得し、ポートデータ操作ウィンドウ上のポートの構成を決定します。

汎用入力に設定されたポートは、ポートデータボタン、同時入力ポート選択チェックボックスが有効になり、入力レベルの設定が可能となります。ウィンドウ上の操作によって入力レベルが変化すると、**ES-Sim17**はデバッガを介してエミュレーションメモリの入力データレジスタを更新します。

汎用出力に設定したポートのポートデータボタンと同時入力ポート選択チェックボックスは、操作が無効なグレー表示となります。ただし、ポートデータボタンは現在の出力状態を示します。汎用出力に設定したポートの出力データレジスタが変更されると、その状態がポートデータボタンに反映されます。

内蔵周辺回路用入出力に設定されたポートのポートデータボタン、同時入力ポート選択チェックボックスは表示されません。

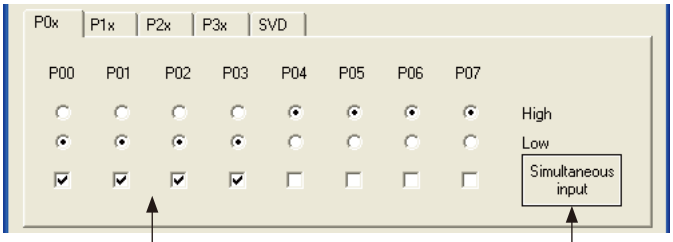
ターゲット機種に存在しないポートのポートデータボタン、同時入力ポート選択チェックボックスは表示されません。

●ポート入力状態の設定

ポートデータボタンでHighまたはLowを選択します。これが現在のポート入力レベルになります。

●複数キーの同時入力

複数のキーを同時に押す操作をシミュレートするには、同時入力ポート選択チェックボックスで同時入力を行うポートをすべて選択します。その状態で同時入力ボタンをクリックすると、該当ポートの入力レベル(ポートデータボタンの設定)が反転します。



(1)同時入力ポートを選択　　(2)クリックで入力レベルが同時に反転

この操作は、現在表示している以外のタブのポート系列にも有効です。同時入力ポート選択ボタンで選択されているポートは、そのタブのページが表示されているか否かにかかわらず、いずれかのページの同時入力ボタンで入力レベルが反転します。

同時入力ポート選択チェックボックスで複数のポートを選択している場合でも、ポートデータボタンで個々のポートの制御は行えます。

●ポートの出力状態

デバッガによるプログラムの実行でポート出力が変化した場合は、その状態が即時ポートデータボタンの表示に反映されます。
なお、汎用出力に設定したポートのポートデータボタンをマウスで操作することはできません。

●P0ポートキー入力リセット

P0ポートの同時入力リセットに対応している機種の場合、ソフトウェアで指定したポートの同時入力によってリセットが行えます。この機能を評価する場合は、上記の複数キーの同時入力と同じ方法を使用してください。

●ポート入力割り込み

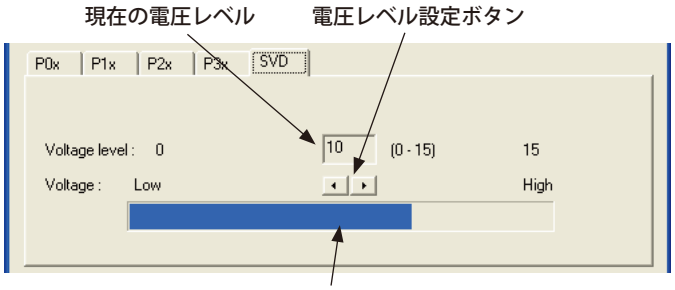
ポートデータ操作ウィンドウ上の入力状態の操作により、ポート入力割り込みを発生させることができます。

10.11.5 SVDシミュレーション

SVD動作を確認するため、電源電圧のレベルを[ES-Sim]ウィンドウで制御することができます。

●SVD操作ウィンドウ

操作ウィンドウ選択タブで、SVDを選択します。



SVD操作ウィンドウ

SVD操作ウィンドウは、電圧レベル15(最高レベル)に初期設定されます。

●電圧レベルの設定

電圧レベル設定ボタンで電圧レベルを0(低)～15(高)の16段階*に設定できます。

* この電圧レベルは、ターゲット機種のSVD比較電圧の有効設定数に相当します。機種により設定可能な電圧レベルが変わる場合があります。

ボタンの操作により、現在の電圧レベルと電圧表示バーが変化します。同時に、エミュレーションメモリ内のSVD制御レジスタに設定されている比較電圧とここで設定した電圧レベルが比較され、結果がSVD検出結果レジスタに書き込まれます。

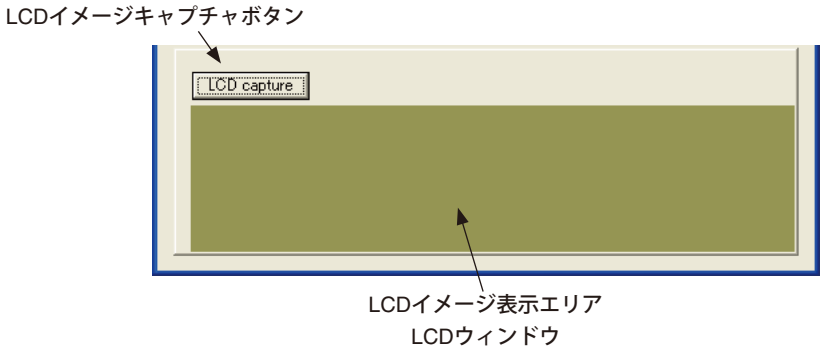
●SVD割り込み

SVD割り込みに対応している機種の場合、このウィンドウで SVD比較電圧より低い電圧レベルを設定することで割り込みを発生可能です。

10.11.6 LCDパネルシミュレーション

LCDドライバの制御と表示メモリの操作に従ったLCDパネルの表示をシミュレートできます。

●LCDウィンドウ



ドットマトリクスタイプのLCDパネルの表示をシミュレートします。
ES-Sim17は32Hzの周期で表示メモリの内容を取得し、このウィンドウを再描画しています。

駆動デューティの設定や表示の制御(表示On/Off、コントラスト、表示エリアの選択など)も反映されます。

●LCD画面の保存

現在、LCDウィンドウに表示されている画面をビットマップデータとしてファイル(.bmp)に保存することができます。
取り込みたい画面になったところで[LCD capture]ボタンをクリックします。ファイル保存ダイアログボックスが表示されますので、保存先の選択とファイル名の入力を行い、ファイルに保存します。
画面データの取り込みは[LCD capture]ボタンをクリックした時点で行われ、保存が完了するまで画面の更新は停止します。
LCDウィンドウにLCDパネル全体が表示されていない場合でも、パネル全体の画像が保存されます。
生成されるファイルはWindows標準のビットマップファイル(.bmp)です。

●制限事項

- 実際のLCDパネルとはドットのサイズやコントラスト、背景色が異なります。
- 実際のLCDパネルとはLCD表示の更新タイミングが異なります。

10.11.7 ES-Sim17のエラーメッセージ

表10.11.7.1 エラーメッセージ(**gdb**の[Console]ウィンドウに表示)

メッセージ	内容
ES-Sim error 01 : Loading the dll file was failed.	**ES-Sim17**のdllファイルが既定の場所にないか、あるいは正常にロードできません。
ES-Sim error 02 : Opening the CPU construction file was failed.	CPU構成ファイルが指定場所にないか、あるいは正常に開けません。
ES-Sim error 03 : Generating the CPU components was failed.	CPUモジュール定義に誤りがあるか、必要なdllがないため、CPUモジュールの生成に失敗しました。
ES-Sim error 04 : Connecting the CPU components was failed.	CPUモジュール定義の接続先に誤りがあるため、CPUモジュールの接続に失敗しました。
ES-Sim error 05 : Opening the "user.ini" was failed.	ユーザ設定ファイルが指定場所にないか、あるいは正常に開けません。
ES-Sim error 06 : Setting of the "user.ini" is invalid.	ユーザ設定ファイルに記述されている設定値が不正です。

表10.11.7.2 エラーメッセージ(ダイアログボックスに表示)

メッセージ	内容
"*path*¥*file*"の保存に失敗しました。	キャプチャ画像の保存に失敗しました。
Failed to open "path¥file". Please confirm the file is fitting with the CPU type, or the file is existing.	LCDファイルのオープンに失敗しました。

10.11.8 制限事項

- 本シミュレータは以下の機種をサポートしています。
  S1C17701、S1C17702、S1C17704、S1C17602、S1C17001
- 外部デバイスは、モノクロドットマトリクスLCDパネル、およびモノクロセグメントLCDパネルをサポートしています。
- 実際のLCDパネルとはドットのサイズやコントラスト、パネルの色が異なります。
- 本シミュレータはインストラクションレベルでシミュレーションを行っています。インストラクションサイクルよりも短いサイクルのシミュレーションを行うことはできません。
- インストラクションサイクルを基に動作クロックをシミュレートしていますので、実際のハードウェアとタイミングが同じではありません。
- 以下の機能はシミュレートできません。
  1. タイマクロックの外部出力、発振クロックの外部出力
  2. UART、I²C、SPI等のデータ送受信機能
  3. ノイズ、チャタリング除去フィルタ機能
- PC上で複数のシミュレータを同時に起動して、シミュレーションを行うことはできません。
- 発振回路の周波数設定値を大きくするとシミュレーションの速度が遅くなります。
- 対応していない回路のI/O制御レジスタは、すべてリード/ライト可能レジスタとなります。また、リセット時に初期化はされません。
- 発振回路、SVD等の回路は安定動作までに時間が必要ですが、シミュレータでは動作開始直後から安定した動作を行います。

※ 最新バージョンのシミュレータの制限事項、および機種に依存する制限事項については"simulator_readme.txt"を参照してください。

S5U1C17001C Manual

11 提出用データの作成

11 提出用データの作成

弊社工場にてターゲットCPUの内蔵ROM又は内蔵FLASHメモリにユーザプログラムを書き込むサービスをご利用の場合、PAファイル(提出用データ)を作成し、弊社へ提出していただく必要があります。PAファイルを作成するには、以下のファイルが必要です。

- FDCファイル(ファンクションオプションドキュメント)
  (ファンクションオプションを選択する必要があるCPUのみ)
- PSAファイル(ROMデータ)

本章では、PAファイルの作成方法について、説明します。

11.1 提出用データ作成ツールの概要

PAファイル(提出用データ)の作成において、以下の2種類のツールを使用します。

1. ファンクションオプションジェネレーター(winfog17.exe)
   本ツールは、ファンクションオプションを選択する必要があるCPUで使用します。
   winfog17はICのマスクパターンを生成するためのFDCファイル(ファンクションオプションドキュメント)を作成するツールです。ウィンドウに表示された選択項目をチェックボックスで選択するだけでファンクションオプションを設定することができます。
   ファンクションオプションを設定する必要がないCPUでは、FDCファイルは不要です。
2. データチェッカ(winmdc17.exe)
   winmdc17は開発が完了したPSAファイル、FDCファイルのデータをチェックし、弊社へ提出するためのPAファイルを作成するツールです

11.2 提出用データの作成手順

図11.2.1にPAファイル(提出用データ)作成のフローを示します

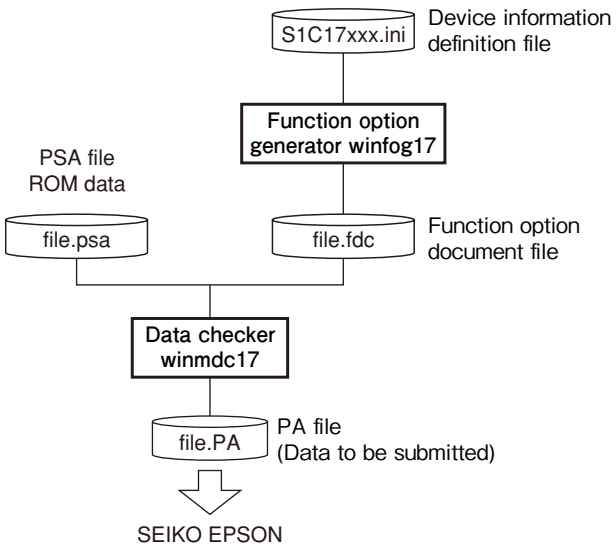


図11.2.1 PAファイル作成フロー

11.2.1 PSAファイル(ROMデータ)の作成

PSAファイル(ROMデータ)は、elf形式オブジェクトファイルと同じファイル名で、拡張子".psa"のモトローラS2形式ファイルです。以下の手順で、PSAファイルを作成します。
なお、生成したPSAファイルを使用して、必ず実機上で動作確認を行ってください。

IDEから自動生成を行う場合

(1) ターゲットCPU機種の選択確認
対象プロジェクト– [プロジェクト] – [プロパティー] – [GNU17一般設定] – [ターゲットCPU機種]が目的の機種であることを確認します。目的の機種が選択されていない場合は、目的の機種を選択してください。また、目的の機種がリストに存在しない場合は、機種別情報ファイル(S5U1C17xxxD4x.zip)をセイコーエプソンのユーザサイトから入手するか、弊社営業窓口にお問い合わせください。

(2) ビルドゴールの切り替え
対象プロジェクト– [プロジェクト] – [プロパティー] – [GNU17ビルドオプション] – [ビルドオプション] – [ビルドゴール切り替え]の[ROMデータ生成(psa)]を選択します。

(3) PSAファイルを生成
対象プロジェクト– [プロジェクト] – [プロジェクトのビルド]からビルドを実行します。対象プロジェクトフォルダ下に、PSAファイルが生成されます。

コマンドラインから手動生成を行う場合

(1) ビルドの実行
対象プロジェクト– [プロジェクト] – [プロジェクトのビルド]からビルドを実行し、elf形式オブジェクトファイルを生成します。

(2) SAファイル(ROMデータ)を生成
以下のコマンドを実行します。objcopy.exe からモトローラ S3 フォーマットの SA ファイルが生成されます。

```
objcopy -I elf32-little -O srec --srec-forceS3 入力ファイル名.elf 出力ファイル名.sa
```

(3) SAFファイル(ROMデータ)を生成
以下のコマンドを実行します。moto2ff.exeを使用してSAファイル内の空きアドレスを0xffの

データで埋めたSAFファイルを生成します。

```
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.sa
```

（出力ファイル名は、入力ファイル名.saf）

(4) PSAファイルを生成

以下のコマンドを実行します。sconv32.exeを使用して、モトローラS2フォーマットに変換し、PSAファイルを生成します。

```
sconv32 S2 入力ファイル名.saf 出力ファイル名.psa
```

ターゲットCPUの内蔵ROM又は内蔵FLASHメモリ領域外にデータが配置されている場合、以下のエラーが発生し、PSAファイルは生成されません。

```
Error: FILENAME contains data outside of specified range (STARTADDR:SIZE)
```


11.2.2 winfog17 によるFDCファイル(ファンクションオプションドキュメント)の作成

本作業は、ファンクションオプションを選択する必要があるCPUで必要となります。
FDCファイル(ファンクションオプションドキュメント)は、ターゲットCPU固有のファンクションオプションを選択したファイルです。選択可能なファンクションオプションについては、ターゲットCPUのテクニカルマニュアルを参照してください。
以下の手順で、winfog17.exeを使用してファンクションオプションを選択し、FDCファイルを生成します。

IDEから自動生成を行う場合

(1) winfog17を起動

対象プロジェクト– [WinFog17を起動] ボタンをクリックします。

図11.2.2.1 [WinFog17を起動]ボタン

ファンクションオプションを選択する必要がないCPUは、[INI file does not include FOG information]とダイアログが表示されます。

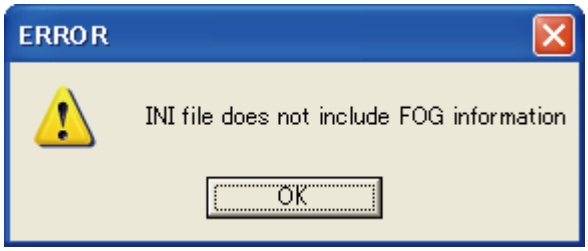


図11.2.2.2 エラー表示

(2) ファンクションオプションを選択

オプションリスト領域のチェックボックスをクリックして必要なオプションを選択します。オプションリスト領域で選択項目を変更すると、選択されたファンクションオプションがファンクションオプション領域に表示されます。

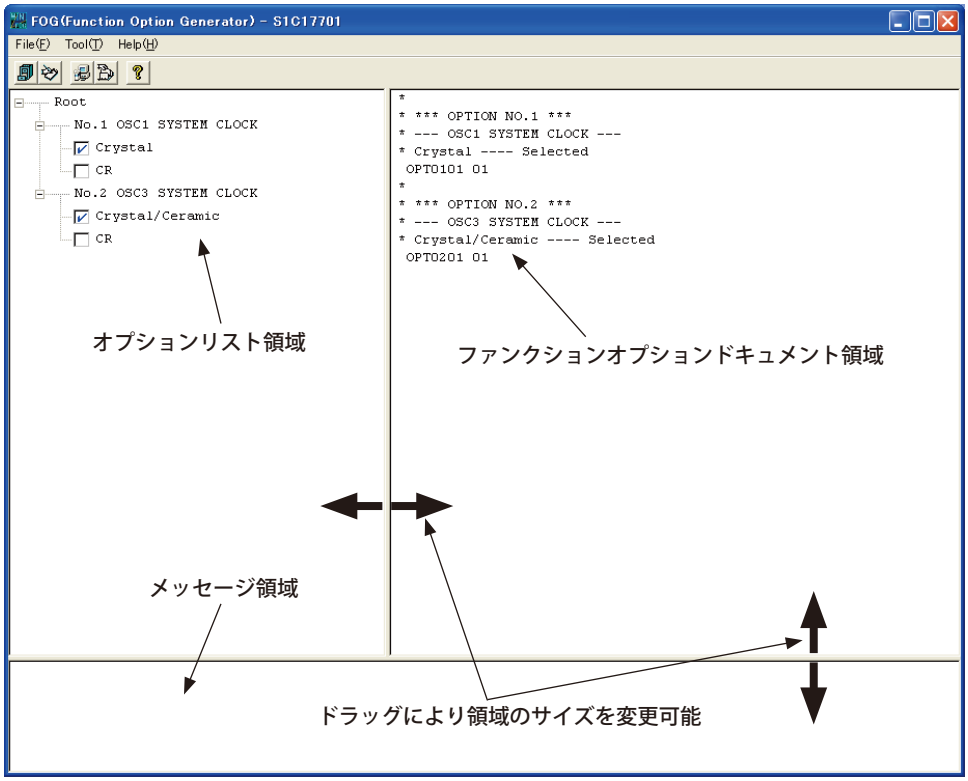


図11.2.2.3 ファンクションオプション表示画面

(3) FDCファイルを生成
winfog17上の[Generate]ボタンをクリック、もしくは[Tool]メニューから[Generate]を選択します。

図11.2.2.4 [Generate]ボタン

ファイル生成が正常に終了した場合は、メッセージ領域に"Making file(s) is completed"が表示されます。
FDCファイルが対象プロジェクトフォルダ下に生成されます。

winfog17.exeから手動生成を行う場合

(1) winfog17を起動
ユーザーフォルダ¥EPSON¥GNU17¥dev¥Binにあるwinfog17.exeをダブルクリックします。

図11.2.2.5 winfog17.exe

前回の実行時に機種情報定義ファイル(S1C17xxx.ini)を読み込んでいる場合は、winfog17起動時に同じファイルを自動的に読み込みます。

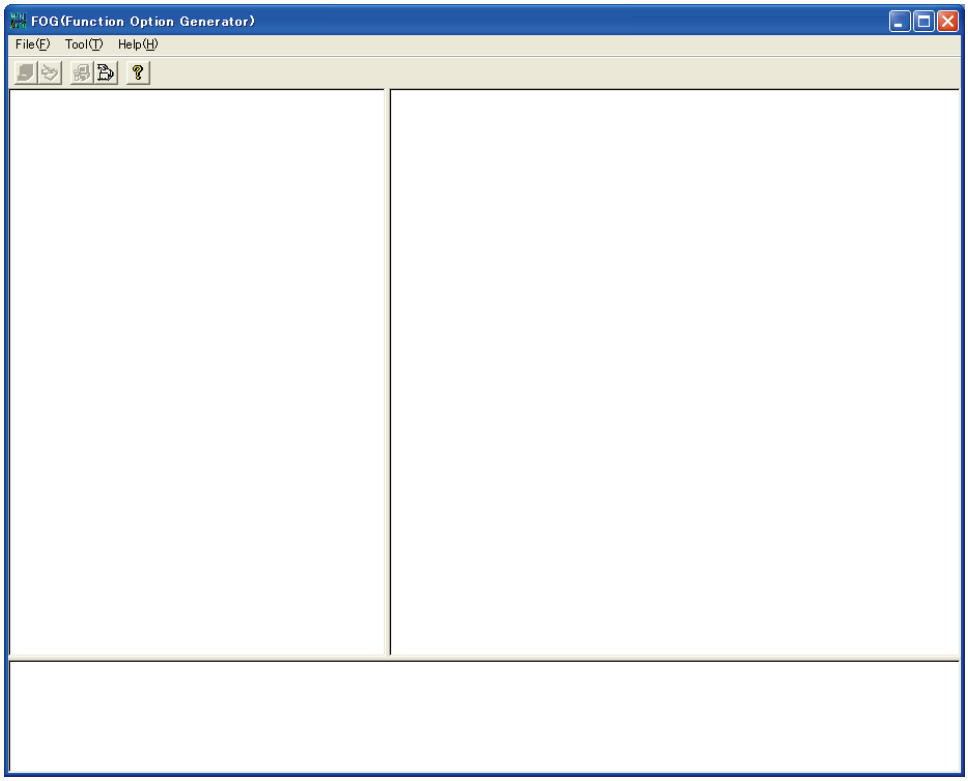


図11.2.2.6 初期画面

(2) S1C17xxx.iniをロード

winfog17.exe上の[Device INI Select]ボタンをクリック、もしくは[Tool]メニューから[Device INI Select]を選択します。

図11.2.2.7 [Device INI Select]ボタン

ダイアログが表示され、テキストボックスにパスを含むファイル名を入力、もしくは[Ref]ボタンをクリックしてターゲットCPUのS1C17xxx.iniをロードします。各機種のS1C17xxx.iniは、以下のフォルダを参照してください。

ユーザーフォルダ¥EPSON¥GNU17¥mcu_model

ターゲットCPUに対応したS1C17xxx.iniが存在しない場合は、営業窓口にお問い合わせください。

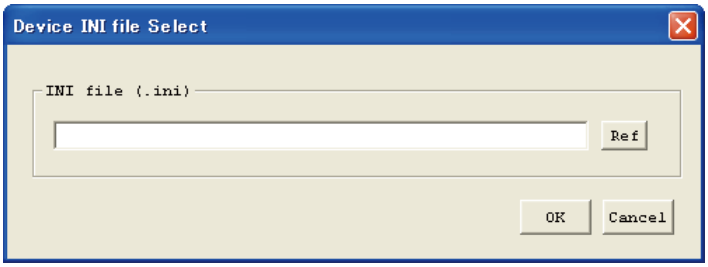


図11.2.2.8 機種情報定義ファイルのロード画面

ファンクションオプションを選択する必要がないCPUについては、[INI file does not include FOG information]とダイアログが表示されます。

(3) ファンクションオプションを選択

オプションリスト領域のチェックボックスをクリックして必要なオプションを選択します。オプションリスト領域で選択項目を変更すると、選択されたファンクションオプションがファンクションオプションドキュメント領域に表示されます。

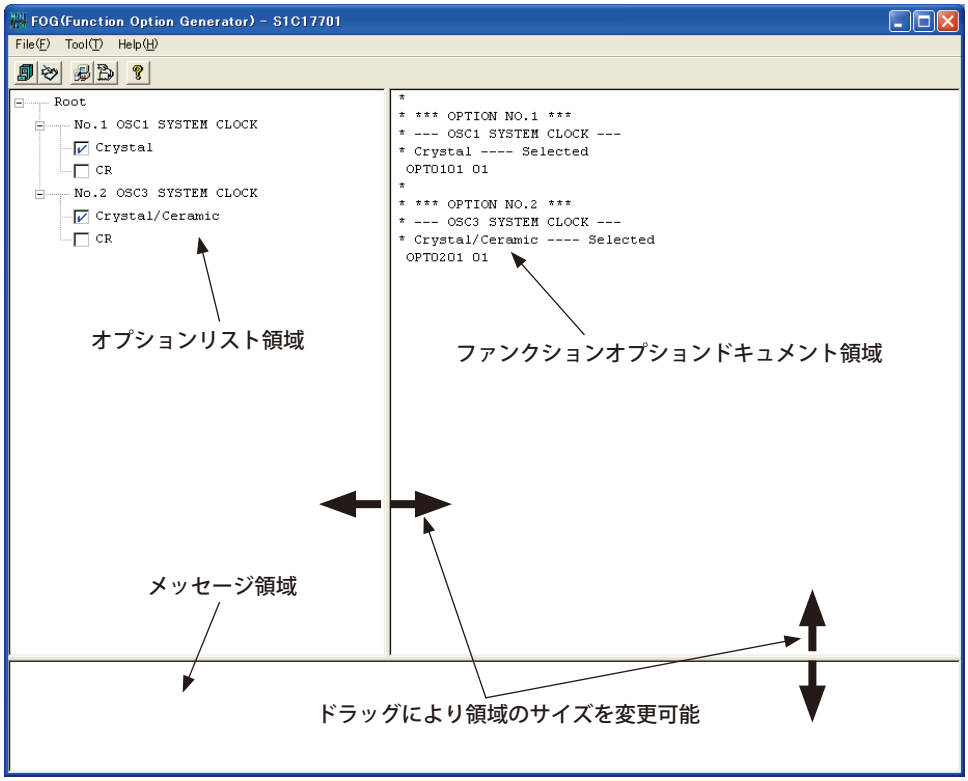


図11.2.2.9 ファンクションオプション表示画面

(4) FDCファイルを生成
Winfog17上の[Generate]ボタンをクリック、もしくは[Tool]メニューから[Generate]を選択します。

図11.2.2.10 [Generate]ボタン

ファイル生成が正常に終了した場合は、メッセージ領域に"Making file(s) is completed"が表示されます。
初期設定では、FDCファイルは以下のフォルダに生成されます。
ユーザーフォルダ¥EPSON¥GNU17¥dev

その他winfog17.exeの機能

上記以外のwinfog17.exeの機能を説明します。下記の機能は必要に合わせてご使用ください。

[file]メニュー

Open

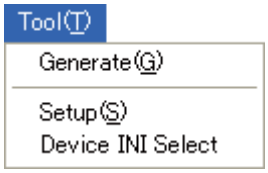
FDCファイルを開きます。既存のファイルを修正する場合に使用します。[Open]ボタンも同機能です。

 [Open]ボタン

End

winfog17を終了します。

[Tool]メニュー

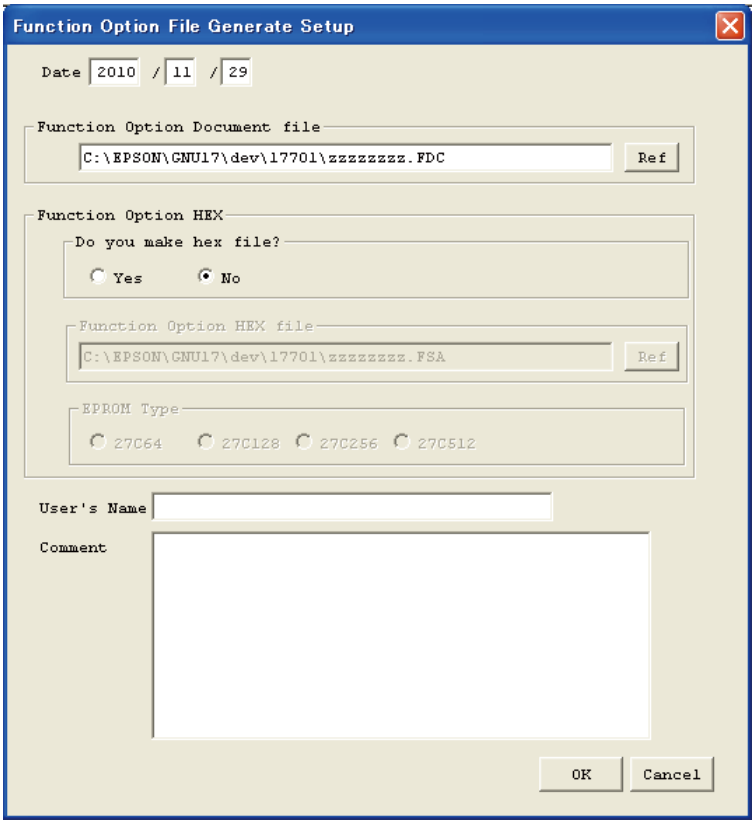


Setup

作成日や出力ファイル名、FDCファイルに含めるコメントなどを設定します。[Setup]ボタンも同機能です。

 [Setup]ボタン

次のダイアログボックスが表示されます。



Date

現在の日付が表示されます。必要に応じて変更してください。

Function Option Document file

作成するFDCファイル名を指定できます。初期設定で表示される名前を修正して使用してください。[Ref]ボタンで他のフォルダも参照できます。

Function Option HEX

S1C17 Familyでは本機能は使用しません。"No"を選択してください。

User's Name

お客さまの会社名を入力します。最大40文字まで入力することができ、それを超えた場合は無視されます。英文字、数字、記号およびスペースが入力可能です。ここに入力した内容は、FDCファイルのUSER'S NAMEフィールドに記録されます。

Comment

コメントを入力します。1行に入力可能な文字数は50文字まで、最大10行まで入力することができます。英文字、数字、記号およびスペースが入力可能です。また、[Enter]キーで改行できます。ここに入力した内容は、FDCファイルのCOMMENTフィールドに記録されます。

上記の必要な項目を入力後、[OK]をクリックすると設定内容が保存され、ダイアログボックスが閉じます。[Cancel]をクリックした場合、現在の設定は変更されずにダイアログボックスが閉じます。

注: • ファイル名の指定には以下の制限があります。

1. パスを含めたファイル名指定の文字数は最大2048文字です。
2. ファイル名（拡張子を除く）は最大15文字、拡張子は最大3文字です。
3. ファイル名の先頭にハイフン(-)は使用できません。また、ディレクトリ名（フォルダ名）、ファイル名、拡張子に、以下の記号の使用を禁止します。
   / : , ; * ? " < > |

• User's NameとCommentに以下の記号は使用できません。
  $ ¥ | `

11.2.3 winmdc17 によるPAファイル(提出用データ)の作成

以下の手順で、winmdc17.exeを使用し、PAファイル(提出用データ)を生成(パック)します。
PAファイルの構成は、以下の2種類あります。

- FDCファイル(ファンクションオプションドキュメント)が存在しない場合
  PSAファイル(ROMデータ)からPAファイルを生成します。
- FDCファイルが存在する場合
  PSAファイルとFDCファイルから、1つのPAファイルを生成します。

IDEから自動生成を行う場合

(1) ご使用の機種がFlash セキュリティ対応でパスワードの設定をしたいときは、プロジェクトプロパティのGNU17 フラッシュ設定画面で「Flash セキュリティ機能を使用する」を有効にして、パスワードを設定してください。

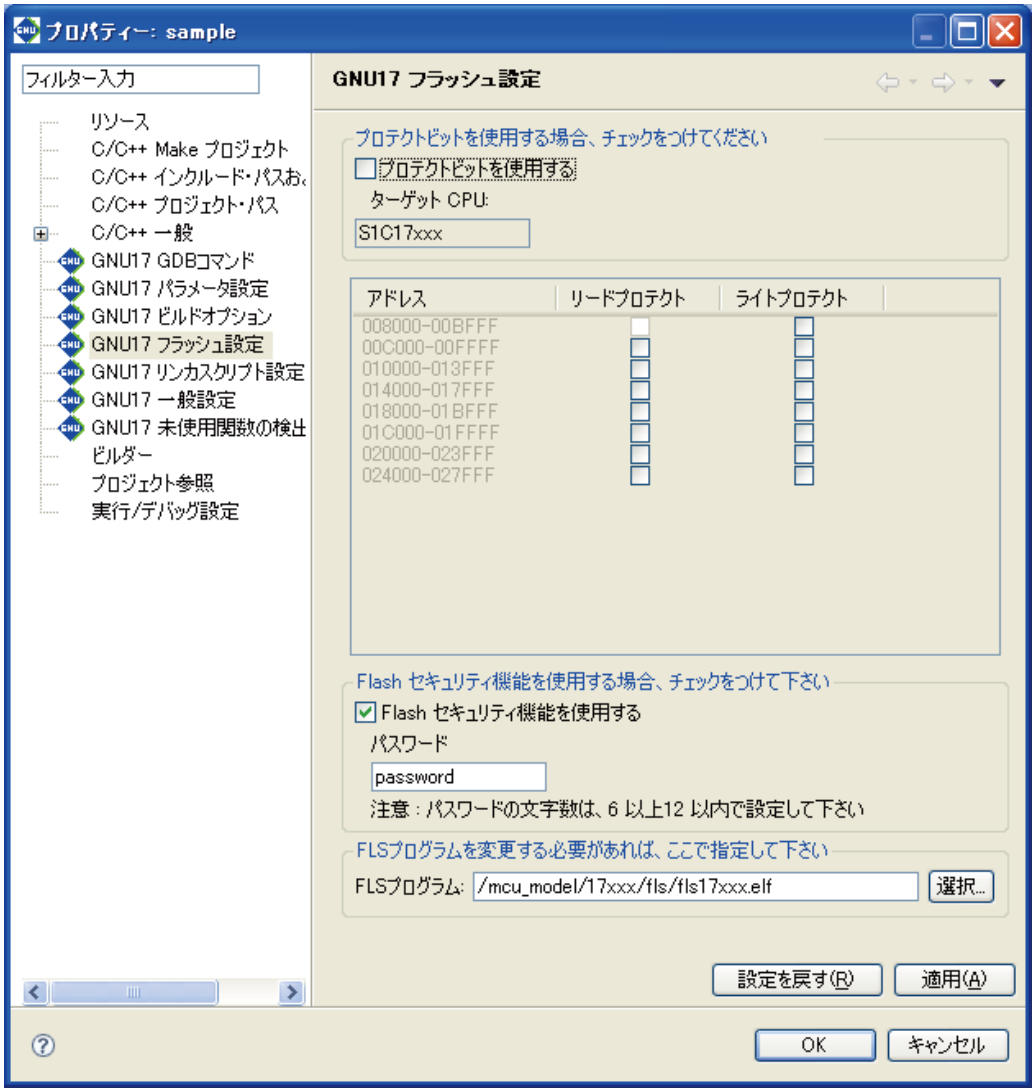


図11.2.3.1 Flash セキュリティパスワードの設定

(2) PAファイルを生成
対象プロジェクト– [WinMdc17でパック] ボタンをクリックし、パックを実行します。

図11.2.3.2 [WinMdc17でパック]ボタン

パックが正常に終了した場合は、以下の画面が表示されます。

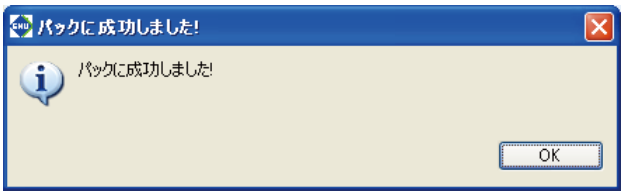


図11.2.3.3 正常終了メッセージ

PAファイルが対象プロジェクトフォルダ下に生成されます。

IDEからwinmdc17.exe から手動生成を行う場合

(1) winmdc17.exeを起動
ユーザーフォルダ¥EPSON¥GNU17¥dev¥Binにあるwinmdc17.exeをダブルクリックします。

図11.2.3.4 winmdc17.exe

前回の実行時に機種情報定義ファイル(S1C17xxx.ini)を読み込んでいる場合は、winmdc17起動時に同じファイルを自動的に読み込みます。

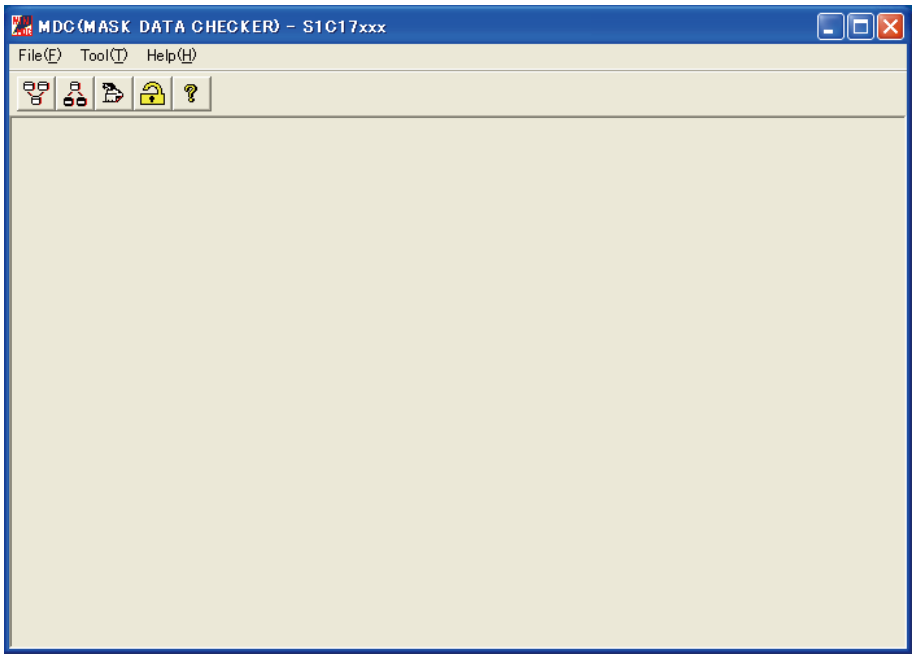


図11.2.3.5 初期画面

(2) S1C17xxx.iniをロード
winmdc17.exe上の[Device INI Select]ボタンをクリック、もしくは[Tool]メニューから[Device INI Select]を選択します。

図11.2.3.6 [Device INI Select]ボタン

ダイアログが表示され、テキストボックスにパスを含むファイル名を入力、もしくは[Ref]ボタンをクリックしてターゲットCPUのS1C17xxx.iniをロードします。各機種のS1C17xxx.iniは、以下のフォルダを参照してください。
ユーザーフォルダ¥EPSON¥GNU17¥mcu_model

ターゲットCPUに対応したS1C17xxx.iniが存在しない場合は、営業窓口にお問い合わせください。

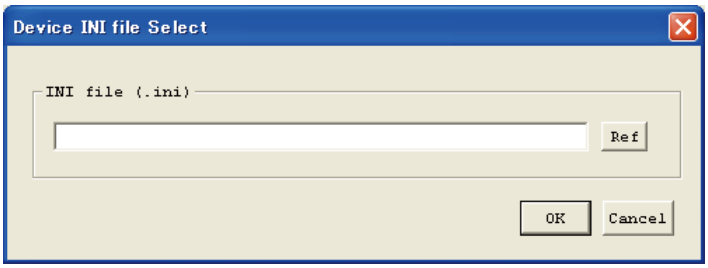


図11.2.3.7 機種情報定義ファイルのロード画面

(3) ご使用の機種がFlash セキュリティ対応でパスワードの設定をしたいときは、winmdc17.exe上の[Password]ボタンをクリック、もしくは[Tool]メニューから[Password]を選択します。

図11.2.3.8 [Password]ボタン

表示されたダイアログに、パスワードを入力してください。

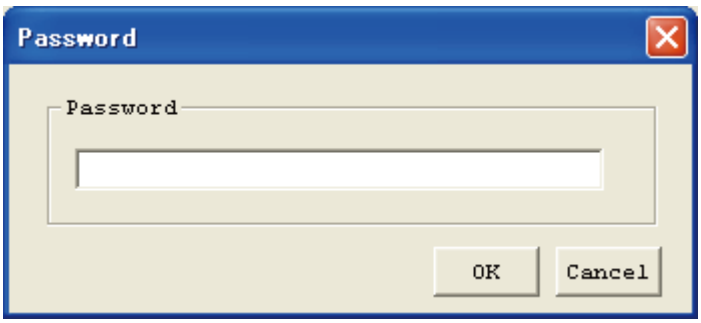


図11.2.3.9 Password設定画面

(4) 入力ファイルの選択
winmdc17.exe上の[Pack]ボタンをクリック、もしくは[Tool]メニューから[Pack]を選択します。

図11.2.3.10 [Pack]ボタン

入力するファイルを選択します。[Pack Input Files]には、S1C17xxx.iniで指定されるファイルがデフォルトのファイル名でリストされます。リストされている全てのファイルに対し、[Ref]ボタンをクリックして、入力するデータファイルを選択してください。

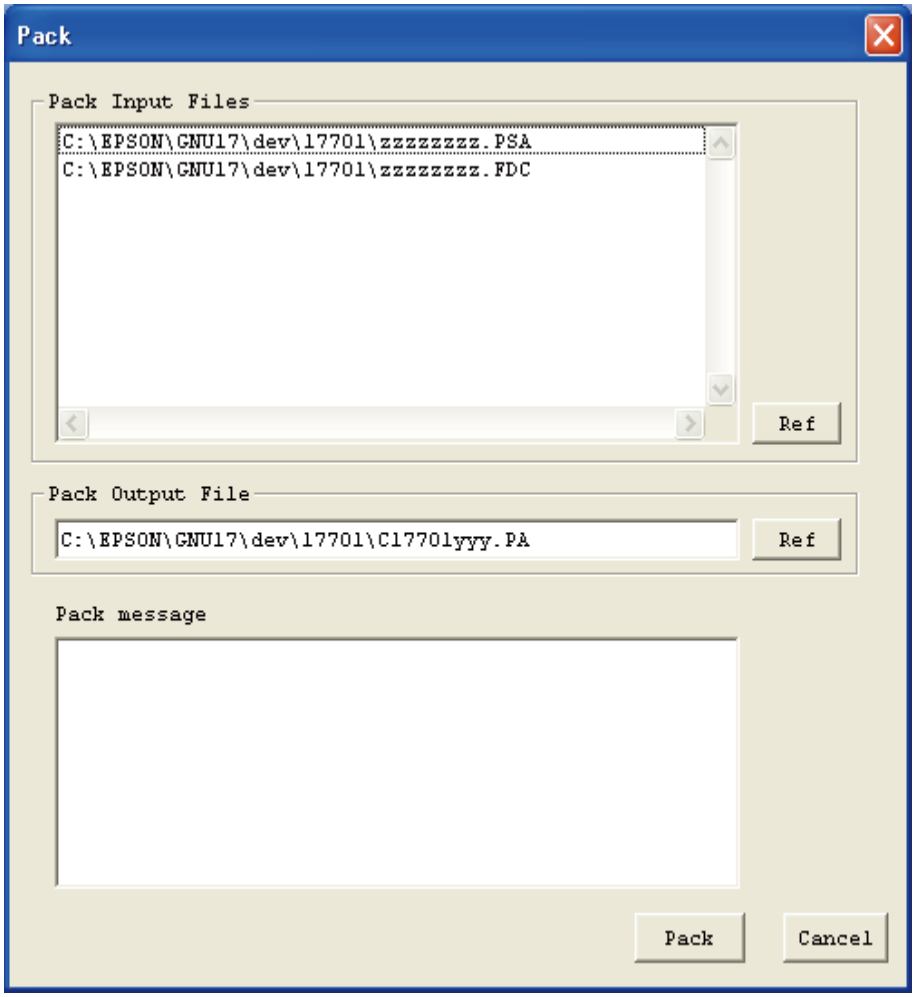


図11.2.3.11 入力ファイル選択画面

[Pack Output File]から、出力するPAファイル名を指定できます。初期状態で表示されているファイル名を修正して使用してください。[Ref]ボタンで他のフォルダも参照できます。出力ファイル名の拡張子は必ず".PA"としてください。

注: ファイル名の指定には以下の制限があります。

1. **パスを含めたファイル名指定の文字数は最大2048文字です。**
2. **ファイル名(拡張子を除く)は最大15文字、拡張子は最大3文字です。**
3. **ファイル名の先頭にハイフン(-)は使用できません。また、ディレクトリ名(フォルダ名)、ファイル名、拡張子に、以下の記号の使用を禁止します。**
   / : , ; * ? " < > |

(5) PAファイルを生成

図11.2.3.8 入力ファイル選択画面の[Pack]ボタンをクリックし、パックを実行します。パックが正常に終了した場合は、[Pack message]領域に"Pack completed !"が表示されます。
初期設定では、PAファイルは以下のフォルダに生成されます。
ユーザーフォルダ¥EPSON¥GNU17¥dev
[Cancel]ボタンをクリックしてダイアログボックスを閉じます。
弊社工場にてターゲットCPUの内蔵ROM又は内蔵FLASHメモリにユーザプログラムを書き込むサービスをご利用の場合、以上の手順で生成したPAファイルを弊社営業窓口へ提出してください。

11.2.4 PAファイル(提出用データ)の分離方法

winmdc17で生成(パック)したPAファイル(提出用データ)をファンクションオプションドキュメントとROMデータに分離(アンパック)することが可能です。

以下の手順で、winmdc17.exeを使用し、PAファイルをUFDファイル(ファンクションオプションドキュメント)とUSAファイル(ROMデータ)に分離します。

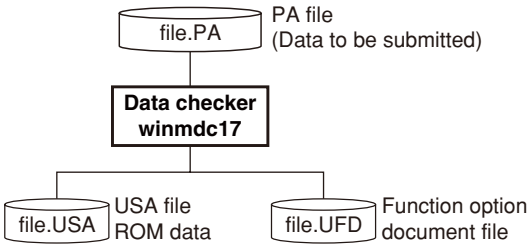


図11.2.4.1 PAファイルのアンパックフロー

IDEから自動で分離を行う場合

(1) PAファイルを分離

対象プロジェクト– [WinMdc17でアンパック] ボタンをクリックし、アンパックを実行します。

図11.2.4.2 [WinMdc17でアンパック]ボタン

アンパックが正常に終了した場合は、以下の画面が表示されます。

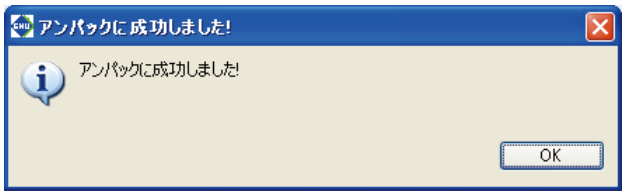


図11.2.4.3 正常終了メッセージ

USAファイルとUFDファイルが対象プロジェクトフォルダ下に生成されます。

winmdc17.exe から手動で分離を行う場合

(1) "winmdc17.exe から手動生成を行う場合"の手順(1) 〜 (2)までを実行します。

(2) 入力ファイルの選択

winmdc17.exe上の[Unpack]ボタンをクリック、もしくは[Tool]メニューから[Unpack]を選択します。

図11.2.4.4 [Unpack]ボタン

アンパックするファイルを選択します。[Packed Input Files]の[Ref]ボタンをクリックし、アンパックするPAファイルを選択してください。

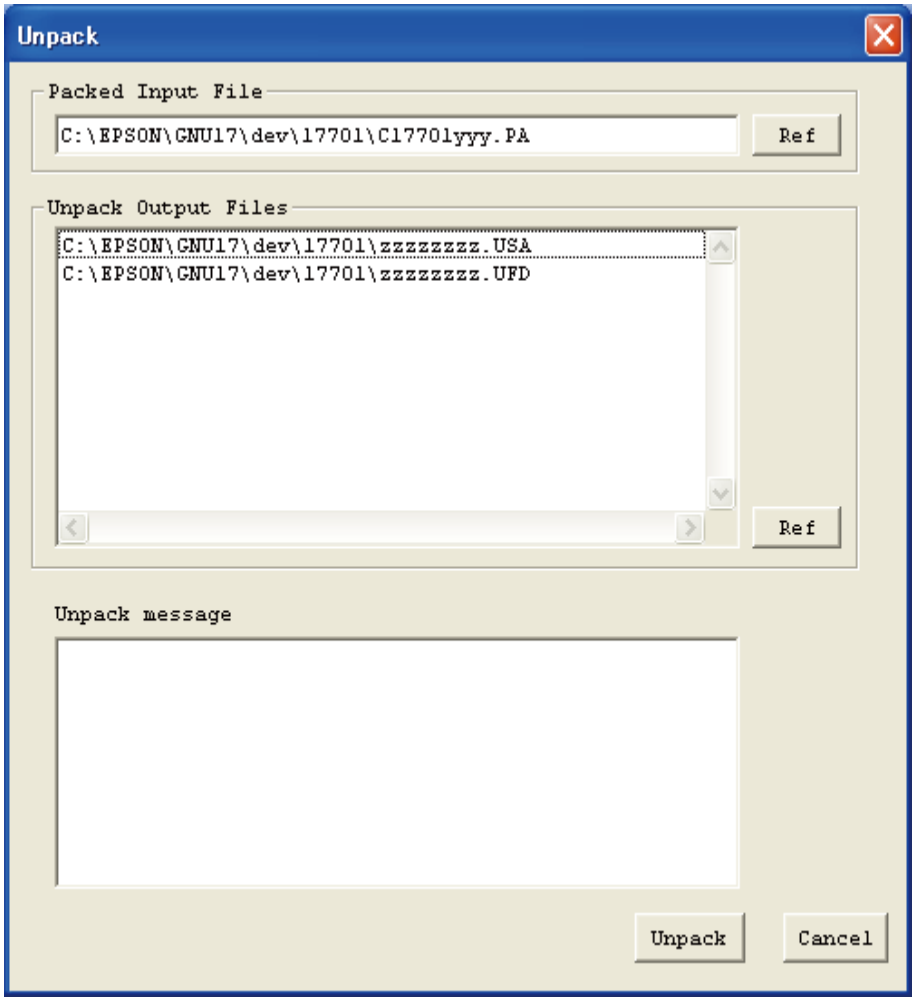


図11.2.4.5 入力ファイル選択画面

[Unpack Output Files]から、出力するファイル名を指定できます。ファイル名をリストされているファイルに対し、[Ref]ボタンをクリックし、出力するフォルダの選択、拡張子付きでファイル名の入力を行ってください。拡張子は変更できません。

(3) PAファイルを分離

入力ファイル選択画面の[Unpack]ボタンをクリックし、アンパックを実行します。アンパックが正常に終了した場合は、[Unpack message]領域に"Unpack completed !"が表示されます。

初期設定では、UFDファイル、USAファイルは以下のフォルダに生成されます。

ユーザーフォルダ¥EPSON¥GNU17¥dev

[Cancel]ボタンをクリックしてダイアログボックスを閉じます。

11.3 提出用データ作成ツールのエラーメッセージ

11.3.1 winfog17のエラーメッセージ

winfog17のエラーメッセージの一覧を示します。表示の"Dialog"はダイアログボックスに表示されるメッセージを、"Message"は[FOG]ウィンドウのメッセージ領域に表示されるメッセージを示します。

表11.3.1.1 エラーメッセージ一覧

メッセージ	説 明	表示
File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。	Dialog
Illegal character	入力禁止文字が入力された。	Dialog
Please input file name	ファイル名が未入力。	Dialog
Can't open File : xxxx	ファイル(xxxx)がオープンできない。 異常時メッセージ(デバッグ中にファイルを削除したときなどに出力される)。	Dialog
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。	Dialog
INI file does not include FOG information	指定した機種情報定義ファイル(.ini)にファンクションオプション情報が含まれていない。	Dialog
Function Option document file is not found	指定したファンクションオプションドキュメントファイルが存在しない。	Dialog
Function Option document file does not match INI file	指定したファンクションオプションドキュメントファイルの内容が機種情報定義ファイル(.ini)と異なる。	Dialog
A lot of parameter	コマンドラインの引数が多すぎる。	Dialog
Making file(s) is completed [xxxx is no data exist]	ファイル作成完了。ただし、作成したファイル(xxxx)にはデータが含まれていない。	Message
Can't open File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)がオープンできない。	Message
Can't write File: xxxx Making file(s) is not completed	Generate実行時、ファイル(xxxx)に書き込みができない。	Message

表11.3.1.2 ワーニングメッセージ

メッセージ	説 明	表示
Are you file update? xxxx is already exist	上書き確認メッセージ (指定したファイルは既に存在する。)	Dialog

11.3.2 winmdc17のエラーメッセージ

winmdc17のエラーメッセージの一覧を示します。表示の"Dialog"はダイアログボックスに表示されるメッセージを、"Message"は[Pack]または[Unpack]ダイアログボックスのメッセージ領域に表示されるメッセージを示します。

表11.3.2.1 I/Oエラーメッセージ一覧

メッセージ	説 明	表示
File name error	ファイル名または拡張子名の文字数が使用可能範囲を超えている。	Dialog
Illegal character	入力禁止文字が入力された。	Dialog
Please input file name	ファイル名が未入力。	Dialog
INI file is not found	指定した機種情報定義ファイル(.ini)が存在しない。	Dialog
INI file does not include MDC information	指定した機種情報定義ファイル(.ini)にMDC情報が含まれていない。	Dialog
Can't open file : xxxx	ファイル(xxxx)がオープンできない。	Dialog
Can't write file: xxxx	ファイル(xxxx)に書き込みができない。	Dialog
Properties file is not found in the same folder as the INI file.	機種情報ファイル(.properties)が機種情報定義ファイル(.ini)と同じフォルダに存在しない。機種情報ファイル(.properties)を機種情報定義ファイル(.ini)と同じフォルダにコピーしてください。	Dialog
Flash security version is not found in the properties file.	Flash セキュリティバージョン情報が、機種情報ファイル(.properties)に無い。	Dialog

メッセージ	説 明	表示
Password length is not found in the properties file.	Flash セキュリティパスワードの有効文字数情報が機種情報ファイル(.properties)に無い。	Dialog

表11.3.2.2 ROMデータエラーメッセージ一覧

メッセージ	説 明	表示
Hex data error: Not S record	データが"S"で始まっていない。	Message
Hex data error: Data is not sequential	データが昇順に並んでいない。	Message
Hex data error: Illegal data	不当なキャラクタがある。	Message
Hex data error: Too many data in one line	1行中のデータ数が多すぎる。	Message
Hex data error: Check sum error	チェックサムが合わない。	Message
Hex data error: ROM capacity over	データ容量が大きい。(データサイズ>ROMサイズ)	Message
Hex data error: Not enough the ROM data	データ容量が少ない。(データサイズ<ROMサイズ)	Message
Hex data error: Illegal start mark	スタートマークが不当である。	Message
Hex data error: Illegal end mark	エンドマークが不当である。	Message
Hex data error: Illegal comment	データの最初の機種名表示が不当である。	Message

表11.3.2.3 ファンクションオプションデータエラーメッセージ一覧

メッセージ	説 明	表示
Option data error : Illegal model name	機種名が不当である。	Message
Option data error : Illegal version	バージョンが不当である。	Message
Option data error : Illegal option number	オプションNo.が不当である。	Message
Option data error : Illegal select number	選択肢No.が不当である。	Message
Option data error : Mask data is not enough	ROMデータが充分でない。	Message
Option data error : Illegal start mark	スタートマークが不当である。	Message
Option data error : Illegal end mark	エンドマークが不当である。	Message

11.4 提出用データ作成ツールの出力例

winfog17.exe、winmdc17.exeから生成されるFDCファイル(ファンクションオプションドキュメント)、PAファイル(提出用データ)のファイルフォーマットを以下に示します。

注: データの構成および内容は機種により異なります。

●FDCファイル例

```
* S1C17xxx_xxKB FUNCTION OPTION DOCUMENT Vx.xx     ←バージョン
*
* FILE NAME zzzzzzzz.FDC                           ←ファイル名（[Setup] で指定）
* USER'S NAME SEIKO EPSON CORPORATION              ←ユーザ名（[Setup] で指定）
* INPUT DATE yyyy/mm/dd                            ←作成年月日（[Setup] で指定）
* COMMENT SAMPLE DATA                              ←コメント（[Setup] で指定）
*
* *** OPTION NO.1 ***                              ←オプション番号
* --- OSC1 SYSTEM CLOCK ---                        ←オプション名
* Crystal(32.768KHz) ---- Selected                 ←選択した仕様
OPT0101 01                                        ←マスクデータ
*
* *** OPTION NO.2 ***
* --- OSC3 SYSTEM CLOCK ---
* CR 200KHz ---- Selected
OPT0201 01
*
* *** OPTION NO.3 ***
* --- INPUT PORT PULL UP RESISTOR ---
* K00 With Resistor ---- Selected
* K01 With Resistor ---- Selected
* K02 With Resistor ---- Selected
* K03 With Resistor ---- Selected
* K04 With Resistor ---- Selected
* K05 With Resistor ---- Selected
* K06 With Resistor ---- Selected
* K07 With Resistor ---- Selected
OPT0301 01
OPT0302 01
OPT0303 01
OPT0304 01
OPT0305 01
OPT0306 01
OPT0307 01
OPT0308 01
*
* *** OPTION NO.4 ***
* --- OUTPUT PORT OUTPUT SPECIFICATION ---
* R00 Complementary ---- Selected
* R01 Complementary ---- Selected
* R02 Complementary ---- Selected
* R03 Complementary ---- Selected
OPT0401 01
OPT0402 01
OPT0403 01
OPT0404 01
*
:
*
* *** OPTION NO.8 ***
* --- SOUND GENERATOR POLARITY ---
* NEGATIVE ---- Selected
OPT0801 01
*EOF                                              ←エンドマーク
```

●PAファイル例

```
*
* S1C17xxx_xxKB xPCS xBITW xBITR MASK DATA VER x.xx       ←バージョン
*
¥ROM1                                          ← ROM データスタートマーク
S1C17xxxyyy PROGRAM ROM                        ←機種名
S224008000.............................
: : : : : "xxxxxxxx.psa"
S804000000FB
¥END                                           ← ROM データエンドマーク
¥OPTION1                                       ←ファンクションオプションスタート
    マーク
* S1C17xxx FUNCTION OPTION DOCUMENT V x.xx   ←機種名 / バージョン
*
* FILE NAME zzzzzzzz.FDC
* USER'S NAME SEIKO EPSON CORPORATION
* INPUT DATE yyyy/mm/dd
* COMMENT SAMPLE DATA
* "xxxxxxxx.fdc"
* *** OPTION NO.1 ***
* --- OSC1 SYSTEM CLOCK ---
* Crystal(32.768KHz) ---- Selected
OPT0101 01
: : : : :
OPTnn01 01
*EOF
¥END                                           ←ファンクションオプションエンドマーク
```

S5U1C17001C Manual

12 その他のツール

12 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。

12.1 make.exe

12.1.1 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make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。

12.1.2 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。

12.1.3 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、make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make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。

S5U1C17001C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。

12.1.4 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。

●生成プログラムがアプリケーション(*.elf/*.psa)の場合

例:

```
# Make file generated by Gnu17 Plug-in for Eclipse
# This file should be placed directly under the project folder

# export environment variable
export CYGWIN=nodosfilewarning

# macro definitions for target file
TARGET= sample
GOAL= $(TARGET).psa

# macro definitions for tools
TOOL_DIR= C:/EPSON/GNU17
CC= $(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
CC_KFILT= $(TOOL_DIR)/xgcc_filt
OBJDUMP= $(TOOL_DIR)/objdump
OBJCOPY= $(TOOL_DIR)/objcopy
MOTO2FF= $(TOOL_DIR)/moto2ff
SCONV= $(TOOL_DIR)/sconv32
VECCHECKER= $(TOOL_DIR)/vecChecker

# macro definitions for tool flags
CFLAGS= -B$(TOOL_DIR)/ -gstabs -S  -O1 -I$(TOOL_DIR)/include -fno-builtin -Wall
-Werror-implicit-function-declaration
ASFLAGS= -B$(TOOL_DIR)/ -c -xassembler-with-cpp -Wa,--gstabs
ASFLAGS_CC=
LDFLAFGS_NOOVERLAP= -Map sample.map -N -T sample_gnu17IDE.lds -c17-overlap-noerr
-c17-memoryover-noerr
LDFLAGS=  -Map sample.map -N -T sample_gnu17IDE.lds
EXTFLAGS= -Wa,-mc17_ext -Wa,$(TARGET).dump -Wa,$(TARGET).map
EXTFLAGS_CC= -mc17_ext $(TARGET).dump $(TARGET).map
OBJDUMPFLAGS= -t
OBJCOPYFLAGS= -I elf32-little -O srec --srec-forceS3
MOTOSTART= 8000
MOTOSIZE= 10000
MOTOPROGSIZE= 10000
SCONVFLAGS= S2
VECCHECKERFLAGS= -t symtable.out -r raw.out
VECCHECKER_ON= false

# macro for switching 2pass or 1pass build
PASS= 2pass

# macro definitions for tool flags
PROTECT_ON= true

# search paths for source files
vpath %.c
vpath %.s

# macro definitions for object files
OBJS= boot.o ¥
```

```
      lib.o ¥
      main.o ¥
      sys.o ¥



# macro definitions for library files
OBJLDS= $(TOOL_DIR)/lib/24bit/libstdio.a ¥
        $(TOOL_DIR)/lib/24bit/libc.a ¥
        $(TOOL_DIR)/lib/24bit/libgcc.a ¥
        $(TOOL_DIR)/lib/24bit/libc.a ¥

# macro definitions for assembly files generated from c source files
CEXTTEMPS= lib.ext0 ¥
      main.ext0 ¥
      sys.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_AS= @$(AS) -M -MG $(ASFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e $(SED_
PTN2) >$(@:%.o=%.d)

# targets and dependencies
.PHONY : all clean

all : $(GOAL)

$(TARGET).psa : $(TARGET).elf
# clean psa files
  $(RM) -f $(TARGET).sa $(TARGET).saf $(TARGET).psa
# create psa file from elf
  $(OBJCOPY) $(OBJCOPYFLAGS) $< $(TARGET).sa
  $(MOTO2FF) $(MOTOSTART) $(MOTOPROGSIZE) $(TARGET).sa
  $(SCONV) $(SCONVFLAGS) $(TARGET).saf $(TARGET).psa

# create protected psa file
  ifeq ($(PROTECT_ON), true)
  $(TOOL_DIR)/gdb.exe --nw --command=protect.cmd
  $(SCONV) $(SCONVFLAGS) temp $(TARGET)_ptd.psa
  $(RM) -f temp
  endif

  @cmd /c "echo ---------------- Finished building target : $@ ----------------"

$(TARGET).elf : $(OBJS) sample_gnu17IDE.mak sample_gnu17IDE.lds
ifeq ($(PASS), 1pass)
# 1pass linking
  $(LD) $(LDFLAGS) -o $@ $(OBJS) $(OBJLDS)
else
# 1pass linking
  -$(LD) $(LDFLAGS_NOOVERLAP) -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
# 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" >obj1pass¥¥temp ; 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)
@if [ -s lderr ]; then ¥
  cmd /c "type lderr" ¥
  && $(RM) -f $(TARGET).elf ¥
  && exit 1; ¥
  else $(RM) -f lderr ; ¥
fi
# restore 1pass object files
  $(RM) -f $(OBJS) ¥
  && ¥
  for NAME in $(subst /,¥¥,$(OBJS)) ; do ¥
        cmd /c "copy /y obj1pass¥¥$$NAME $$NAME" >obj1pass¥¥temp ; done ¥
  && cmd /c "rd /s /q obj1pass"
endif

# check copro function in vector
ifeq ($(VECCHECKER_ON), true)
  $(RM) -f symtable.out raw.out
  $(OBJDUMP) -t $@ > symtable.out
  $(OBJDUMP) -s $@ > raw.out
  $(VECCHECKER) -t symtable.out -r raw.out
endif

  @cmd /c "echo ---------------- Finished building target : $@ ----------------"

## boot.s
boot.o : boot.s
  $(AS) $(ASFLAGS) -o $@ $<
  $(DEPCMD_AS)

## lib.c
lib.o : lib.c lib.ext0
  $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
  $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
  $(DEPCMD_CC)

## main.c
main.o : main.c main.ext0
  $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
  $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
  $(DEPCMD_CC)

## sys.c
sys.o : sys.c sys.ext0
  $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
  $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
  $(DEPCMD_CC)


# dependecies for assembled c source files
```

```
lib.ext0 : lib.c
main.ext0 : main.c
sys.ext0 : sys.c

# include dependency files
-include $(DEPS)

# clean files
clean :
  $(RM) -f $(OBJS) $(TARGET).elf $(TARGET).map $(DEPS) $(CEXTTEMPS) $(TARGET).dump
lderr $(TARGET).sa $(TARGET).saf $(TARGET).psa $(TARGET)_ptd.psa
  @if [ -e obj1pass ]; then ¥
         cmd /c "rd /s /q obj1pass" ; ¥
  fi
```

●生成プログラムがライブラリ(*.a)の場合

例:

```
# Make file generated by Gnu17 Plug-in for Eclipse
# This file should be placed directly under the project folder

# export environment variable
export CYGWIN=nodosfilewarning

# macro definitions for target file
TARGET= sample

# macro definitions for tools
TOOL_DIR= C:/EPSON/GNU17
CC= $(TOOL_DIR)/xgcc
AS= $(TOOL_DIR)/xgcc
AS_CC= $(TOOL_DIR)/as
RM= $(TOOL_DIR)/rm
SED= $(TOOL_DIR)/sed
AR= $(TOOL_DIR)/ar

# macro definitions for tool flags
CFLAGS= -B$(TOOL_DIR)/ -mno-sjis-filt -gstabs -S  -O1 -I$(TOOL_DIR)/include -fno-
builtin -Wall -Werror-implicit-function-declaration
ASFLAGS= -B$(TOOL_DIR)/ -mno-sjis-filt -c -xassembler-with-cpp -Wa,--gstabs
ASFLAGS_CC=
ARFLAGS=  rus

# search paths for source files
vpath %.c
vpath %.s

# macro definitions for object files
OBJS= sub1.o ¥
      sub2.o ¥
      sub3.o ¥

# macro definitions for assembly files generated from c source files
CEXTTEMPS= sub2.ext0 ¥
           sub3.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)
```

```
>$(@:%.o=%.d)
DEPCMD_AS= @$(AS) -M -MG $(ASFLAGS) $< | $(SED) -e $(SED_PTN) | $(SED) -e $(SED_
PTN2) >$(@:%.o=%.d)

# targets and dependencies
.PHONY : all clean

all : $(GOAL)


$(TARGET).a : $(OBJS) sample_gnu17IDE.mak
 $(AR) $(ARFLAGS) $@ $(OBJS)
 @cmd /c "echo ---------------- Finished building target : $@ ----------------"

## sub1.s
sub1.o : sub1.s
 $(AS) $(ASFLAGS) -o $@ $<
 $(DEPCMD_AS)

## sub2.c
sub2.o : sub2.c sub2.ext0
 $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
 $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
 $(DEPCMD_CC)

## sub3.c
sub3.o : sub3.c sub3.ext0
 $(CC) $(CFLAGS) -o $(@:%.o=%.ext0) $<
 $(AS_CC) $(ASFLAGS_CC) -o $@ $(@:%.o=%.ext0)
 $(DEPCMD_CC)


# dependecies for assembled c source files
sub2.ext0 : sub2.c
sub3.ext0 : sub3.c

# include dependency files
-include $(DEPS)

# clean files
clean :
 $(RM) -f $(OBJS) $(TARGET).a $(DEPS) $(CEXTTEMPS)
```

●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/gnu17/sample/tst/boot.o : **c:**/EPSON/gnu17/sample/tst/boot.s
as -o boot.o c:/EPSON/gnu17/sample/tst/boot.s

↓

/cygdrive/c/EPSON/gnu17/sample/tst/boot.o : **/cygdrive/c**/EPSON/gnu17/sample/tst/boot.s
as -o boot.o c:/EPSON/gnu17/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、make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。

12.1.5 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/gnu17
        :
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/gnu17/ld -T sample.lds -Map sample.map -N -o sample.elf boot.o main.o
/cygdrive/c/EPSON/gnu17/lib/libc.a /cygdrive/c/EPSON/gnu17/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。

12.1.6 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

make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 $(OBJS) $(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make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12.1.6.1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。

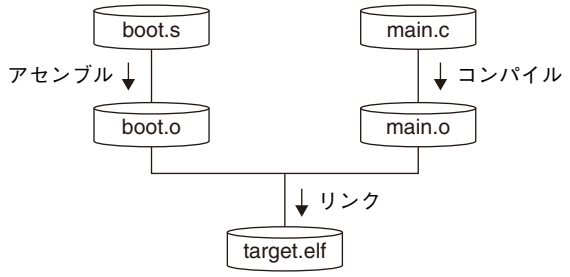


図12.1.6.1 ファイルの依存関係(例)

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make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make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make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makeを実行することができません。
  悪い例:
```
vector2.o : vector.c
    /cygdrive/c/EPSON/gnu17/sed.exe -f .comm/place.sed vector.c > vector2.c
    /cygdrive/c/EPSON/gnu17/xgcc -B/cygdrive/c/EPSON/gnu17/ -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 : vector2.c
    /cygdrive/c/EPSON/gnu17/xgcc -B/cygdrive/c/EPSON/gnu17/ -c vector2.c -o vector2.o

vector2.c : vector.c
    /cygdrive/c/EPSON/gnu17/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。

12.1.7 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                                   サフィックス定義

.c.o :                                                      ┐
  $(CC) $(CFLAGS) -o $(SRC_DIR)/$*.o $(SRC_DIR)/$*.c        │ サフィックスルール
.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)          │
                                                            │ 依存リスト
## 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。make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。

12.1.8 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make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/gnu17
   :
RM= $(TOOL_DIR)/rm
   :
OBJS= boot.o ¥
      main.o ¥
   :
DEPS = $(OBJS:%.o=%.d)
clean:
  $(RM) -f $(OBJS) $(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、make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。

12.1.9 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。

12.1.10 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12.1.10.1 通常メッセージ

メッセージ	内容
make: Nothing to be done for '*TARGET*'.	*TARGET*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.	*FILENAME*はすでに最新の状態に更新されています。make処理は行われませんでした。

表12.1.10.2 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.	*FILENAME2*を作成するための*FILENAME1*が見つからないか、*FILENAME1*を作成する依存リストが定義されていません。make処理は中断されました。
make: *TOOL*: Command not found	*FILENAME*を作成する依存リストに記述された*TOOL*が見つかりませんでした。make処理は中断されます。
make: *** [*FILENAME*] Error 127	
make: *** [*FILENAME*] Error 1	呼び出したツールでエラーが発生しました。make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。make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12.1.10.3 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*)	*FILENAME*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。make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.	

12.1.11 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。
- GNU17v2.2.0より、cygwin関連のdllがバージョンアップされました。それにより、以下の点に注意してください。
  (1) GNU17v2.1.0以前のバージョンのmakeファイル内で記述されていたnulデバイスへの出力(>nul)が廃止されました。
  IDEでmakeファイルを自動生成しない設定になっている場合は、makeファイルからnulデバイスへの出力(>nul)の記述を削除してください。
  makeファイル内でnulデバイスへの出力(>nul)の記述がされている場合、ビルド時にプロジェクトフォルダ内にnulファイルが生成されてしまいます。このnulファイルは通常のファイル削除の操作では、削除することができません。nulファイルを削除する場合は、本パッケージに含まれているrm.exeを使用して、コマンドプロンプト上から以下のように削除してください。
  例：c:¥epson¥gnu17¥rm nul

  (2) makeファイル内にてファイルのパスをDOS形式で指定すると以下のような警告メッセージが表示されます。
  cygwin warning:
  MS-DOS style path detected: xxxxxx
  Preferred POSIX equivalent is: xxxxxx

CYGWIN environment variable option "nodosfilewarning" turns off this warning.
IDEでmakeファイルを自動生成しない設定になっており、かつmakeファイル内でDOS形式のパスを指定する場合は、以下の行をmakeファイルに追加してください。

```
export CYGWIN=nodosfilewarning
```

12.2 ccap.exe

12.2.1 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。

12.2.2 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。

12.2.3 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¥gnu17¥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"
```


12.2.4 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12.2.4.1 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。

12.3 objdump.exe

12.3.1 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。

12.3.2 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。

12.3.3 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¥gnu17¥objdump  -S  test.elf`

12.3.4 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¥gnu17¥>objdump -d main.o
main.o:     file format elf32-c17

Disassembly of section .text:

00000000 <main>:
   0: 3e5c  ld.a    -[%sp],%r4       ld.a       -[%sp],%r4

00000002 <.LBB2>:
   2: 9900  ld      %r2,0x0          ld         %r2,0x0 <main>
   4: 4000  ext     0x0
   6: 4000  ext     0x0
   8: d900  ld      [0x0],%r2       xld         [0x0],%r2 <main>
  .....
```

-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。
デバッガの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¥gnu17¥>objdump -h sample.elf

sample.elf:     file format elf32-c17

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .bss          000000b8  00000000  00000000  000000b4  2**2
                  ALLOC
  1 .data         00000000  000000b8  00004080  00000134  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .rodata       00000080  00004000  00004000  000000b4  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .text         0000023e  00004080  00004080  00000134  2**1
                  CONTENTS, ALLOC, LOAD, CODE
  4 .stab         00000bb8  000042c0  000042c0  00000374  2**2
                  CONTENTS, READONLY, DEBUGGING
  5 .stabstr      000008b3  00004e78  00004e78  00000f2c  2**0
                  CONTENTS, READONLY, DEBUGGING

*** 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¥gnu17¥>objdump -g sample.elf

sample.elf:     file format elf32-c17

boot.s:
/* file boot.s line 47 addr 0x4080 */
/* file boot.s line 48 addr 0x4082 */
/* file boot.s line 49 addr 0x4084 */
/* file boot.s line 50 addr 0x4086 */
/* file boot.s line 53 addr 0x4088 */
  .....
main.c:
typedef int16 int;
typedef int8 char;
typedef int32 long int;
typedef uint16 unsigned int;
typedef uint32 long unsigned int;
  .....
typedef complex float0 complex float;
typedef complex float0 complex double;
typedef complex float0 complex long double;
typedef void *__builtin_va_list;
typedef enum { False, True } _Bool;
int main ()
{ /* 0x40a0 */
  /* file main.c line 10 addr 0x40a0 */
  { /* 0x40a2 */
    register int j /* 0x4 */;
    /* file main.c line 12 addr 0x40a2 */
    /* file main.c line 14 addr 0x40aa */
    /* file main.c line 16 addr 0x40ac */
    /* file main.c line 14 addr 0x40b2 */
  } /* 0x40b8 */
  /* file main.c line 18 addr 0x40b8 */
} /* 0x40bc */
  .....
```

-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¥gnu17¥>objdump -t sample.elf

sample.elf:     file format elf32-c17

SYMBOL TABLE:
00000000 l    d  .bss      00000000
000000b8 l    d  .data     00000000
00004000 l    d  .rodata   00000000
00004080 l    d  .text     00000000
000042c0 l    d  .stab     00000000
00004e78 l    d  .stabstr  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¥gnu17¥>objdump -s sample.elf

sample.elf:     file format elf32-c17

Contents of section .data:
Contents of section .rodata:
 4000 90400000 88400000 80400000 80400000  .@...@...@...@..
 4010 80400000 80400000 80400000 80400000  .@...@...@...@..
 4020 80400000 80400000 80400000 80400000  .@...@...@...@..
  .....
Contents of section .text:
 4080 00000000 0000fc13 00000000 00002801  ..............(.
 4090 7e4000bc 0040b918 00400218 00402010  ~@...@...@...@ .
 40a0 5c3e0099 00400040 00d9122a 14280040  ¥>...@.@...*.(.@
  .....
```

-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。

```
C:¥EPSON¥gnu17¥>objdump -G sample.elf

sample.elf:     file format elf32-c17

Contents of .stab section:

Symnum n_type n_othr n_desc n_value  n_strx String

-1     HdrSym 0      249    000008b3 1
0      SO     0      0      00004080 1      boot.s
1      SLINE  0      47     00004080 0
2      SLINE  0      48     00004082 0
3      SLINE  0      49     00004084 0
4      SLINE  0      50     00004086 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¥gnu17¥>objdump -S sample.elf

sample.elf:     file format elf32-c17

Disassembly of section .text:

00004080 <__START_text>:
.text
.align 1

EXCEPTION:
```

```
    nop
    4080: 0000  nop                       nop
  .....
00004090 <BOOT>:

BOOT:
    xld.a  %sp, 0x3f00
    4090: 407e  ext     0x7e
    4092: bc00  ld.a    %sp,0x0         sld.a       %sp,0x3f00
    xcall  _init_lib
    4094: 4000  ext     0x0
    4096: 18b9  call    0xb9            xcall       0xb9     (0x00420A) <_init_lib>
  .....
000040a0 <main>:

void sub( int k );

main()
{
    40a0: 3e5c  ld.a    -[%sp],%r4       ld.a       -[%sp],%r4

000040a2 <.LBB2>:
  int j;
  i = 0;
    40a2: 9900  ld      %r2,0x0          ld         %r2,0x0 <__START_bss>
    40a4: 4000  ext     0x0
    40a6: 4000  ext     0x0
    40a8: d900  ld      [0x0],%r2       xld         [0x0],%r2 <__START_bss>
  .....
}
    40b8: 3e38  ld.a    %r4,[%sp]+       ld.a       %r4,[%sp]+
    40ba: 0120  ret                      ret
  .....
```

12.3.5 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12.3.5.1 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。

12.3.6 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。

12.4 objcopy.exe

12.4.1 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。
S1C17 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→モトローラS3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。

12.4.2 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

●SAファイル(ROMデータ)

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。

12.4.3 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

S1C17 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。

12.4.4 SAファイル(ROMデータ)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¥gnu17¥>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¥gnu17¥>objcopy -I elf32-little -O srec --srec-forceS3 input.elf
output.sa
```

12.5 ar.exe

12.5.1 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。

12.5.2 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。

12.5.3 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。

12.6 moto2ff.exe

12.6.1 機能

moto2ffは開始アドレスとブロックサイズを指定して、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のデータで埋めた出力ファイルを生成します。
S1C17 Familyのアプリケーション開発においては、objcopyにより生成されたモトローラS3形式ファイルからROMエリアのデータを抽出するために使用します。
ここで生成したROMエリアのデータをsconv32、winmdc17で処理し、最終的にセイコーエプソンに提出するPAファイル(提出用データ)を生成します。PAファイルの生成手順については、"3.3.7 PAファイル(提出用データ)の作成"を参照してください。

注：PAファイルの作成は、必ずプロジェクトプロパティの[ビルドゴール切り替え]で[ROMデータ生成(psa)]を選択し、“3.3.7 PAファイル(提出用データ)の作成”の手順に沿って作成してください。作成の手順や設定を間違えた場合、動作しないPAファイルが作成されることがありますので、注意して作成してください。

12.6.2 入出力ファイル

入力ファイル

●SAファイル(ROMデータ)

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形式のファイルです。

出力ファイル

●SAFファイル(ROMデータ)

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>*.saf
内容: 指定したアドレスのROMデータです。未使用アドレスには0xffが埋め込まれます。

12.6.3 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、以下のエラーメッセージが出力され、、出力ファイルは生成されません。
  エラーメッセージ：
  `Error: FILENAME contains data outside of specified range (STARTADDR:SIZE)`
- 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。

- データの開始アドレスとデータのブロックサイズは、各機種のテクニカルマニュアルを参照し、機種に合わせて正確に入力してください。正しく入力されていない場合、最終的にPAファイルを生成する**winmdc17**の処理でエラーとなります。
- 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。
  `moto2ff : Convert Completed`
- [-f] : 強制出力オプションです。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で指定した範囲外のデータがある場合でも、指定した範囲のみを切り出して0xff埋めを行った出力ファイルを生成します。範囲外のデータがある場合はワーニングメッセージを出力します。

12.6.4 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12.6.4.1 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。
FILENAME contains data outside of specified range ("*STARTADDR*":"*SIZE*")	*FILENAME*には、指定された範囲(*STARTADDR*から*SIZE*)以外のデータも含まれています。出力ファイルは生成されません。

表12.6.4.2 ワーニングメッセージ

ワーニングメッセージ	内容
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。
FILENAME contains data outside of specified range ("*STARTADDR*":"*SIZE*")	*FILENAME*には、指定された範囲(*STARTADDR*から*SIZE*)以外のデータも含まれていますが、強制出力オプション -f によって出力ファイルが生成されます。

12.6.5 SAFファイル(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SAファイル(ROMデータ)を作成後、**moto2ff**によりSAFファイルを作成します。
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¥gnu17¥>moto2ff 8000 10000 input.sa`

このコマンドは`input.sa`内のアドレス0x8000から始まる0x10000バイト分のデータを`input.saf`に出力します。`input.sa`のアドレス0x8000から0x17fff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。
以上の操作により、内蔵ROM用のSAFファイルが作成されます。

この後、ここで生成したSAFファイルを**sconv32**を使用してモトローラS2フォーマットのPSAファイル(ROMデータ)に変換します。そのファイルを使用して実機上での最終動作確認を行ってください。

最終的には、動作確認済みのPSAファイルと**winfog17**で生成したFDCファイル(ファンクションオプションドキュメント)を**winmdc17**で1つのPAファイルにパックし、セイコーエプソンに提出していただきます。

12.7 sconv32.exe

12.7.1 機能

sconv32はモトローラS形式ファイルの変換ツールです。S1C17 Familyのアプリケーション開発においては、**moto2ff**により生成されたモトローラS3形式のSAFファイル(ROMデータ)をモトローラS2形式に変換するために使用します。
ここで変換したPSAファイル(ROMデータ)を使用して実機上での動作確認を行った後、ファイルを**winmdc17**で処理し、最終的にセイコーエプソンに提出するPAファイル(提出用データ)を生成します。PAファイルの生成手順については、"3.3.7 PAファイル(提出用データ)の作成"を参照してください。

12.7.2 入出力ファイル

入力ファイル

●SAF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>*.saf
内容: **moto2ff**で生成したモトローラS3形式のファイルです。

出力ファイル

●PSAファイル

ファイル形式: モトローラS2形式ファイル
ファイル名: *<ファイル名>*.psa
内容: 入力データファイルをモトローラS2形式に変換したファイルです。

12.7.3 起動フォーマット

sconv32 S2 *<入力ファイル> <出力ファイル名>*

S2: モトローラS2形式に変換することを指定するスイッチです。
<入力ファイル名>: **moto2ff**で生成したSAFファイル名を指定します。
<出力ファイル名>: モトローラS2形式に変換後の出力ファイル名を指定します。
拡張子は必ず".psa"としてください。

- 引数の指定がない場合は使用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標準エラーに出力され、出力ファイルは生成されません。
- [Esc]キーにより、変換動作中でも強制停止できます。
- 正常に終了した場合、以下のようなメッセージが出力されます。

```
Sconv32 : Convert Completed.      終了メッセージ              *1
```

*1: 標準出力に出力

12.7.4 エラーメッセージ

sconv32が出力するエラーメッセージを以下に示します。

表12.7.4.1 エラーメッセージ

エラーメッセージ	内容
INVALID SWITCH.	無効なスイッチが指定されました。
COMPLEMENT SWITCH ERROR.	出力ファイルのチェックサム補数の指定に誤りがあります。
S FORMAT TYPE ERROR.	出力ファイルのSフォーマットの指定に誤りがあります。
NO INPUT FILE NAME.	入力ファイルが指定されていません。
NO OUTPUT FILE NAME.	出力ファイルが指定されていません。
INPUT SAME FILE.	入力と出力に同じファイルを指定しました。
CANNOT OPEN SOURCE FILE (*filename*).	入力ファイルが見つからないか、開けません。
CANNOT OPEN DESTINATION FILE (*filename*).	出力ファイルが開けません。
SOURCE RECORD TYPE NOT SUPPORT.	入力ファイルのレコードタイプは未対応です。
ADDRESS LENGTH RANGE OVER.	入力ファイルのアドレスが変換後のS形式のアドレス長を超えています。
OTHER ERROR.	その他のエラーが発生しました。

12.8 LCDUtil17（LCDパネルカスタマイズツール）

12.8.1 概要

LCDパネルカスタマイズツール（以下LcdUtil17と記述）は組み込みシステムシミュレータES-Sim17（v.1.2以降）でモノクロLCDパネルの表示をシミュレートするための、LCDパネルレイアウトおよびCOM/SEGポートの割り付けを記載したLCDファイルを作成します。セグメントLCDのレイアウトはビットマップファイル(.bmp)から作成でき、ES-Sim17上で実際の製品と同様の画面をシミュレートすることができます。また、本ツールはドットマトリクスLCDのレイアウト作成にも対応しています。

12.8.2 入出力ファイル

図12.8.2.1にLcdUtil17の入出力ファイルを示します。

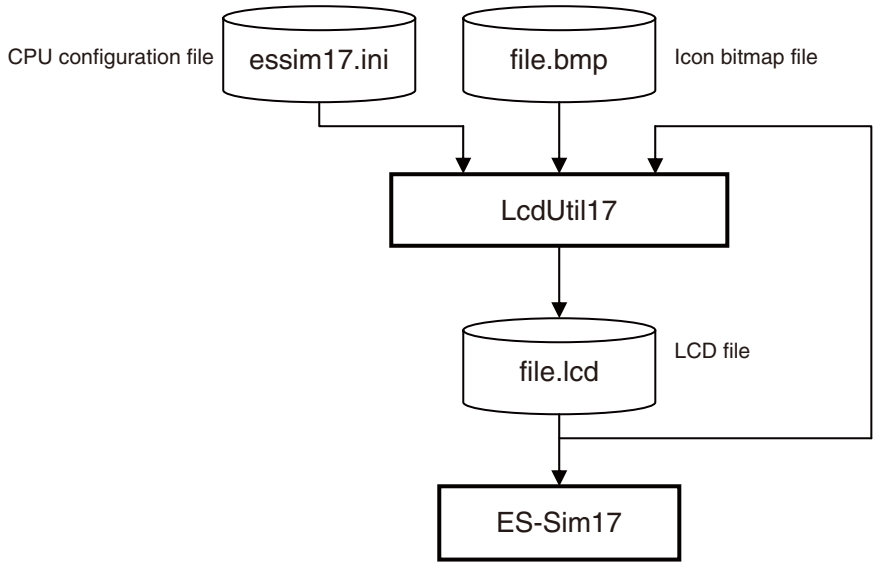


図12.8.2.1　LcdUtil17の入出力ファイル

●CPU構成ファイル（essim17.ini）
シミュレータの機種情報が記録されています。必ずエプソンが提供する設定ファイルを使用してください。ファイルの内容を編集すると、本ツールならびに、ES-Sim17が正常に動作しなくなる恐れがあります。

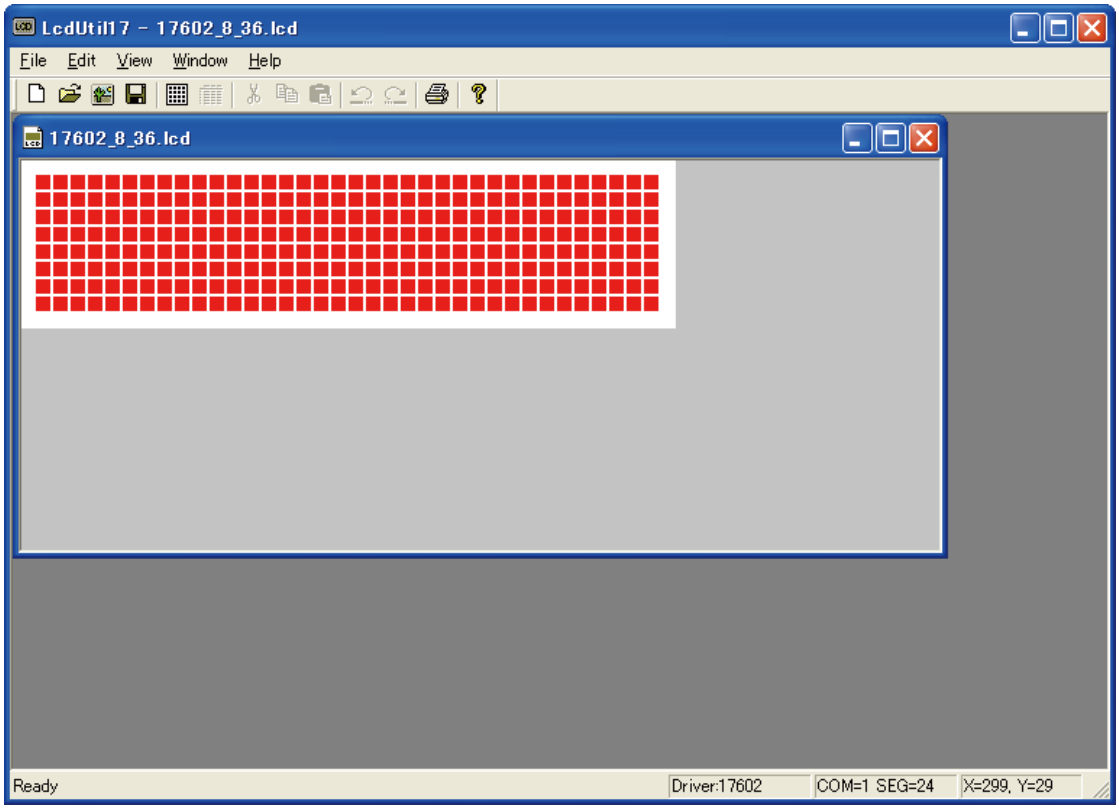
●ビットマップファイル（file_name.bmp）
LCDパネルをイメージした（白黒）ビットマップファイルです。ビットマップファイルを読み込んで、それぞれパーツごとにアイコンとして取り込み、LcdUtil17上でレイアウトを編集することができます。

●LCDファイル
LCDパネルレイアウトとCOM/SEG割り当て情報が記載されたファイルです。このファイルをシミュレータにロードしてLCD表示のシミュレーションを行います。

12.8.3 起動と終了

IDEの[GNU17 アクション]メニューから[LcdUtilityを起動]を選択するか、IDE上のツールバーで  ボタンをクリックするとLcdUtil17が起動します。
終了するには、[File]メニューから[Exit]を選択します。

12.8.4 ウィンドウ



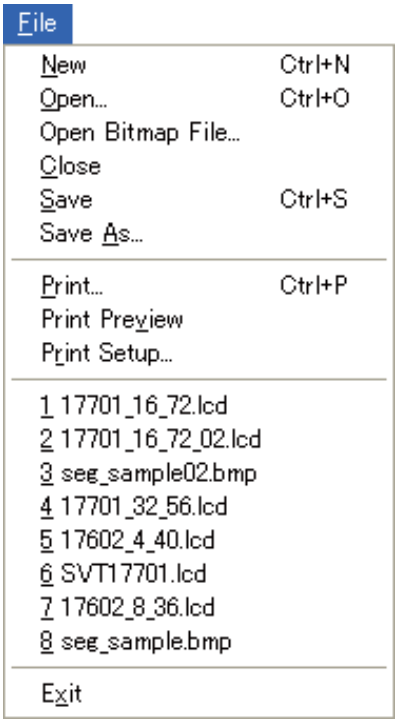
●パネル編集ウィンドウ

ビットマップフ ァイル(.bmp)またはLCDファイル(.lcd)を開くと、このウィンドウに表示されます。このウィンドウ上でLCDパネルのレイアウト、COM/SEGの割り付けを行うことができます。
ウィンドウは同時に複数開くことができ、2つのウィンドウ間でのアイコンおよびドットマトリクスのドラッグ＆ドロップにも対応しています。

12.8.5 メニューとツールバー

12.8.5.1 メニュー

[File]メニュー



[New] ([Ctrl]+[N])
新しいパネル編集ウィンドウを開きます。
[Open…] ([Ctrl]+[O])
LCDファイル(.lcd)を開きます。
[Open Bitmap File…]
ビットマップファイル(.bmp)を開きます。
[Close]
アクティブなパネル編集ウィンドウを閉じます。
[Save] ([Ctrl]+[S])
アクティブなパネル編集ウィンドウの内容をLCDファイル(.lcd)に保存(上書き)します。
[Save As…]
アクティブなパネル編集ウィンドウの内容を別の名称でLCDファイル(.lcd)に保存します。
[Print…] ([Ctrl]+[P])
アクティブなパネル編集ウィンドウのビットマップを印刷します。
[Print Preview]
アクティブなパネル編集ウィンドウの印刷イメージを表示します。
[Print Setup…]
用紙やプリンタを選択するダイアログボックスを表示します。
ファイルリスト
8件まで開いたファイルの履歴を表示し、開くことができます。
[Exit]
LcdUtil17を終了します。

[Edit]メニュー

[Cut] ([Ctrl]+[X])
パネル編集ウィンドウ内で選択されているパーツを、クリップボードへカットします。
[Copy] ([Ctrl]+[C])
パネル編集ウィンドウ内で選択されているパーツを、クリップボードへコピーします。
[Paste] ([Ctrl]+[V])
クリップボードにコピーされているパーツをパネル編集ウィンドウの左上端にペーストします。
[Insert dot matrix] ([Ctrl]+[M])
パネル編集ウィンドウにドットマトリクスを挿入します。ダイアログボックスでサイズなどを編集できます。
[Icon List] ([Ctrl]+[I])
アクティブなパネル編集ウィンドウにあるアイコンの一覧を表示します。この一覧の中でもCOM/SEGの割り付けを行えます。
[Resize LCD]
LCDパネルのサイズを設定します。新規のパネル編集ウィンドウのサイズは640×480です。
[Group Icon]
複数のアイコンを 1 つにグループ化します。
[Release Group]
グループを解除し、個々のアイコンに戻します。

[View]メニュー

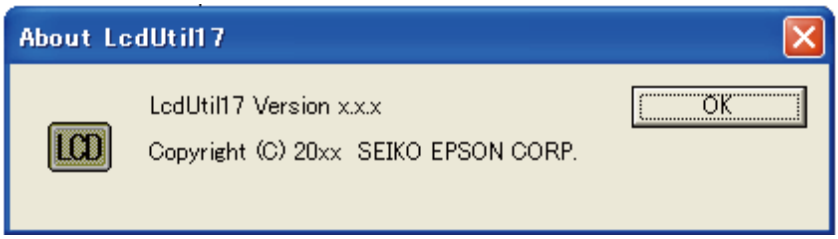
[Toolbar]
ツールバーの表示/非表示を切り替えます。
[Status Bar]
ステータスバーの表示/非表示を切り替えます。

[Window]メニュー

[Cascade]
パネル編集ウィンドウをカスケード表示します。
[Tile]
パネル編集ウィンドウをタイル表示します。
[Arrange Icons]
パネル編集ウィンドウを最小化してウィンドウ下側へ整列させます。
ウィンドウリスト
現在開いているパネル編集ウィンドウ名がリストされます。
ここでパネル編集ウィンドウを選択すると、選択されたパネル編集ウィンドウがアクティブになります。

[Help]メニュー

[About LcdUtil...]
LcdUtil17のバージョン情報を表示します。



12.8.5.2 ツールバーボタン

[New] ボタン
新しいパネル編集ウィンドウを開きます。

[Open] ボタン
LCDファイル(.lcd)を開きます。

[Bitmap] ボタン
ビットマップファイル(.bmp)を開きます。

[Save] ボタン
アクティブなパネル編集ウィンドウの内容をLCDファイル(.lcd)に保存(上書き)します。

[Dot Matrix] ボタン
パネル編集ウィンドウにドットマトリクスを挿入します。

[Icon List] ボタン：[Edit]-[Icon List]
アクティブなパネル編集ウィンドウにあるアイコンの一覧を表示します。

[Cut] ボタン
パネル編集ウィンドウ内で選択されているパーツを、クリップボードへカットします。

[Copy] ボタン
パネル編集ウィンドウ内で選択されているパーツを、クリップボードへコピーします。

[Paste] ボタン
クリップボードにコピーされているパーツをパネル編集ウィンドウの左上端にペーストします。

[Undo] ボタン
3つ前までの操作へ戻ることができます。
Undo可能な操作:アイコン/ドットマトリクスの移動、カット、ペースト、SEG/COM変更、グループ化、グループ化解除

[Redo] ボタン
Undoで戻した操作をやり直すことができます。

[Print] ボタン
アクティブなパネル編集ウィンドウのビットマップを印刷します。

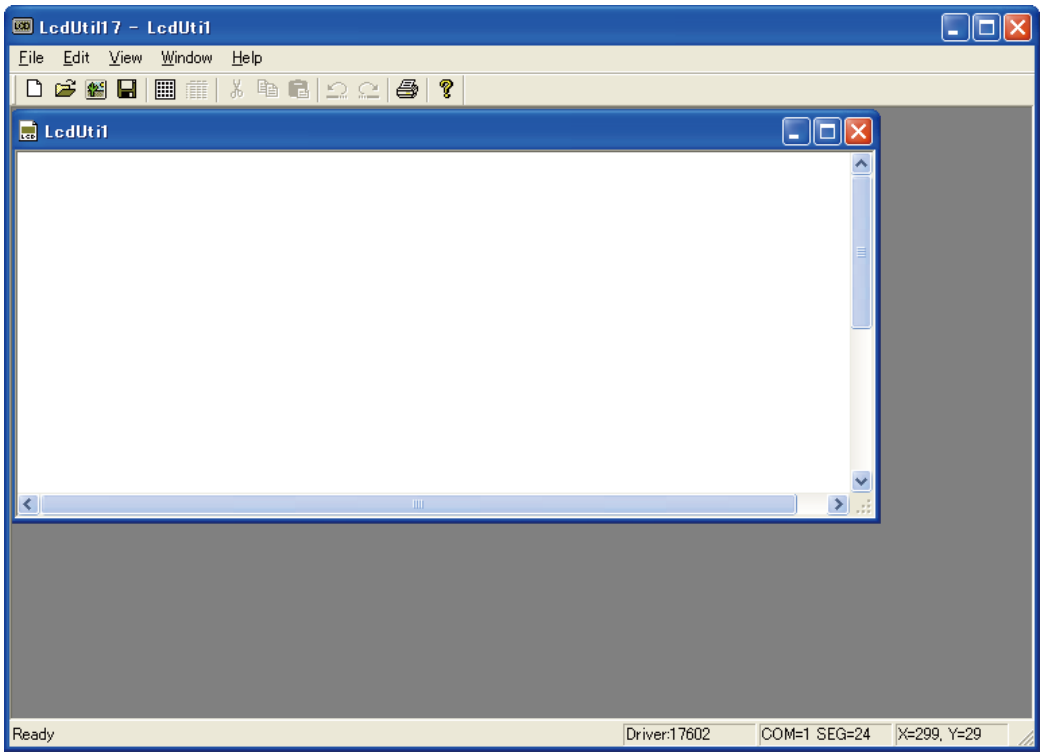
[About] ボタン
LcdUtil17のバージョン情報を表示します。

12.8.6 LCDファイルの作成

パネル編集ウィンドウにはアイコンとドットマトリクスを実際のLCDパネルのようにレイアウトすることができ、COM/SEGの割り付けも行えます。以下、その方法を説明します。

12.8.6.1 ドットマトリクスLCDパネルの作成

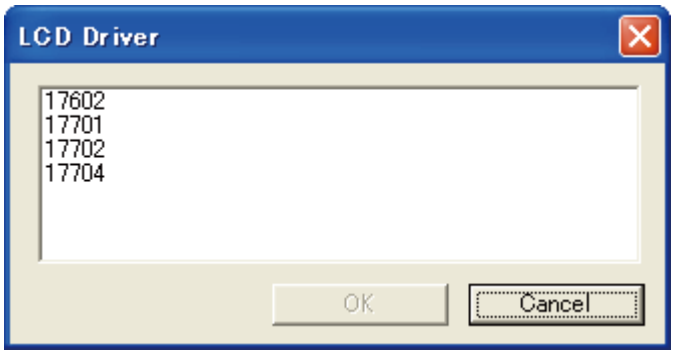
1) [File]メニューから[New]を選択します。
空白のパネル編集ウィンドウが開きます。



2) [Edit]メニューから[Resize LCD]を選択します。
[Resize LCD image]ダイアログボックスが開きます。

ここでLCDパネルのサイズを入力し、[OK]ボタンをクリックします。
新規作成時のサイズは640×480ドットです。
ドットマトリクスのサイズを計算し、LCDパネルのサイズを設定してください。

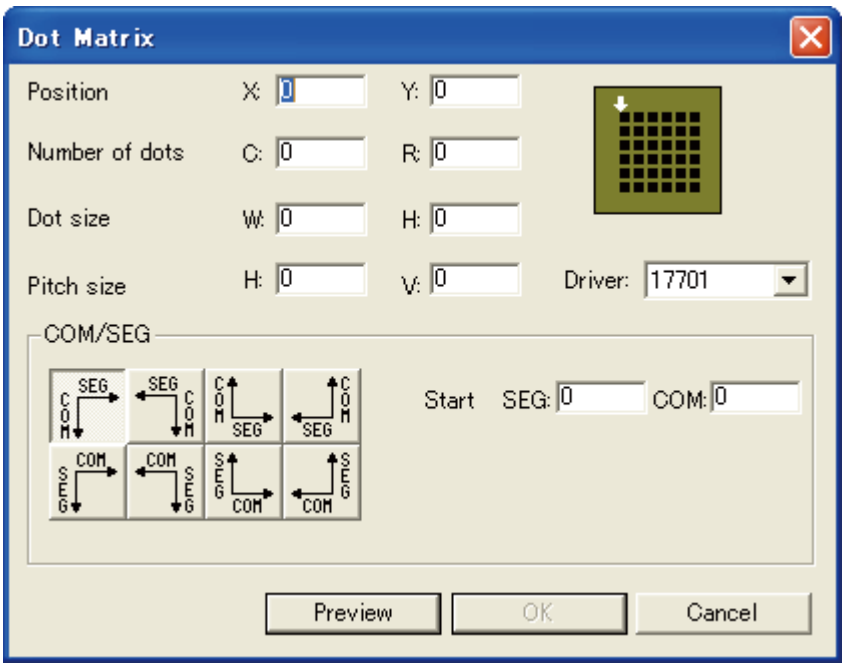
3) [Edit]メニューから[Insert dot matrix]を選択します。
LCDドライバが設定されていない場合は[LCD Driver]ダイアログボックスが表示されます。
リストから開発する機種を選択して[OK]ボタンをクリックします。



※ このダイアログボックスは、LCDドライバが設定されていない場合にのみ表示されます。すでに作成済みのLCDファイルを読み込んだ場合や、IDEでプロジェクトを選択した状態でIDEの[LCDUtilityを起動]ボタンをクリックした場合は、LCDドライバがすでに設定されており、このダイアログボックスは表示されません。

注：LCDドライバは１度設定すると変更することができませんので、選択には注意してください。

4) [Dot matrix]ダイアログボックスが表示されます。



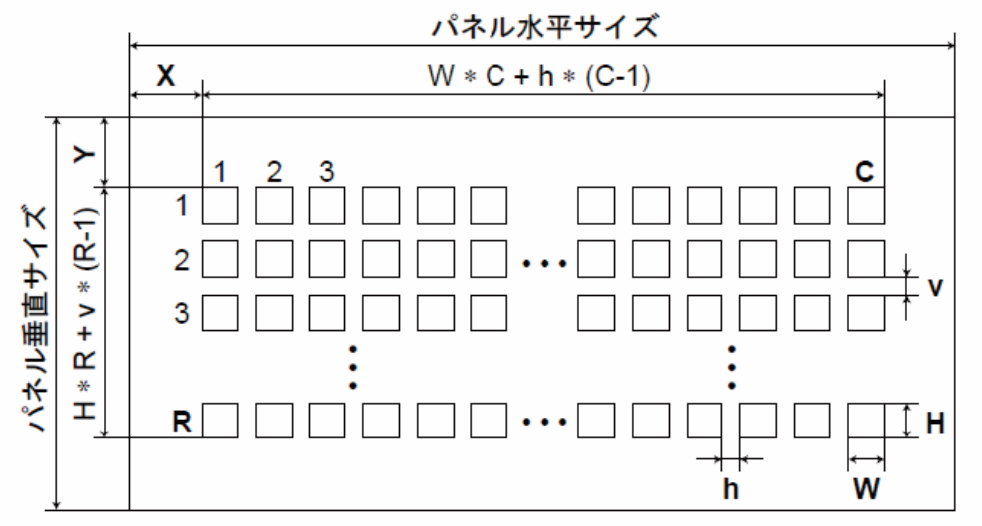


図12.8.6.1.1　ドットマトリクスの設定

[Dot matrix]ダイアログボックスで以下の設定を行ってください。

Position
ドットマトリクスの左上端の座標を指定します。（図のXとY）

Number of dots
ドットマトリクスの水平方向、垂直方向のドット数を指定します。（図のCとR）

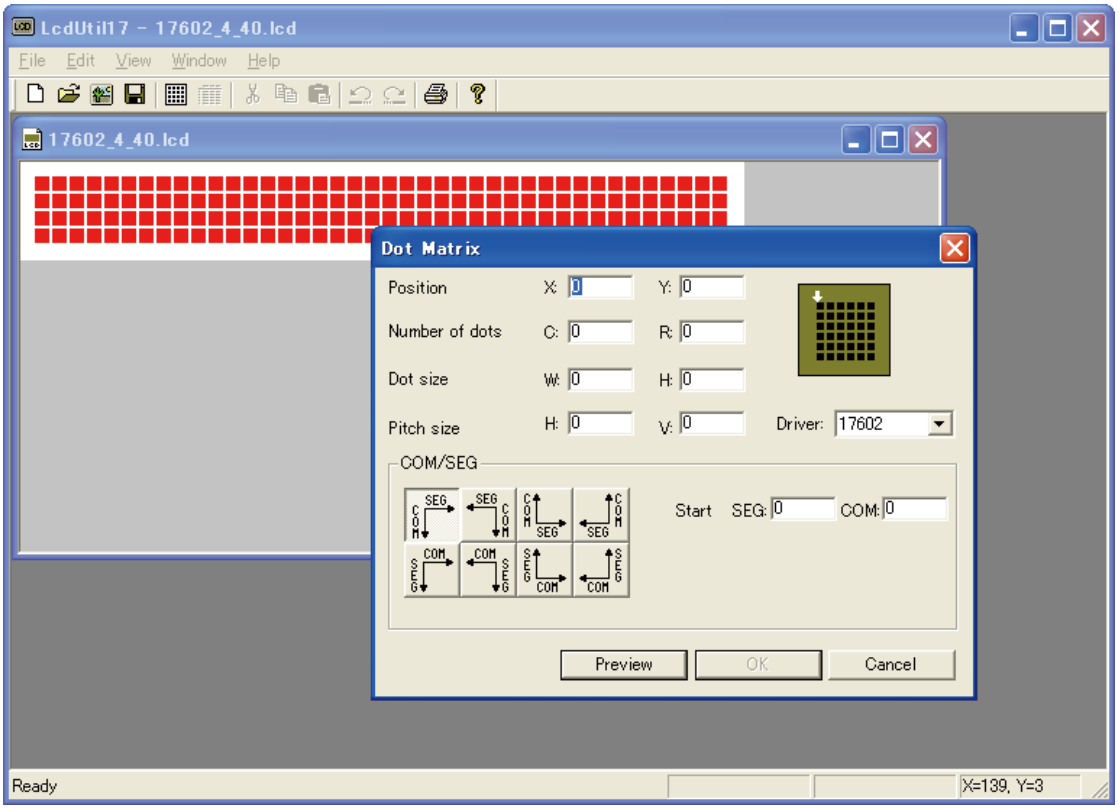
Dot size
ドットマトリクスの１つのドットの大きさをドット数で指定します。（図のWとH）

Pitch size
ドットマトリクスのドットの間隔をドット数で指定します。（図のhとv）

Driver
現在設定されているLCDドライバ名を表示します。変更はできません。
COM/SEG値の上限はLCDドライバで設定されています。

COM/SEG
COM/SEGポートの割り付け方向を選択します。[Start]テキストボックスに指定したCOM/SEGの値から順にポート番号を割り付けます。

上記の各設定を行った後、[Preview]ボタンをクリックすると、パネル編集ウィンドウ上に確認用のドットマトリクスが表示されます。[OK]ボタンをクリックするとこの設定でドットマトリクスが作成されます。



COM/SEGポートを割り付けされたドットマトリクスは赤で表示されます。
ドットマトリクスをダブルクリックすると、[Dot matrix]ダイアログボックスが表され、
ダイアログボックスの設定を変更することができます。

注：ドットマトリクスをコピー＆ペーストした場合、位置とサイズ情報は保持されますが、ポート割り付け情報は破棄されます。割り付け情報のないドットマトリクスは黒で表示されます。

12.8.6.2 セグメントLCDパネルの作成

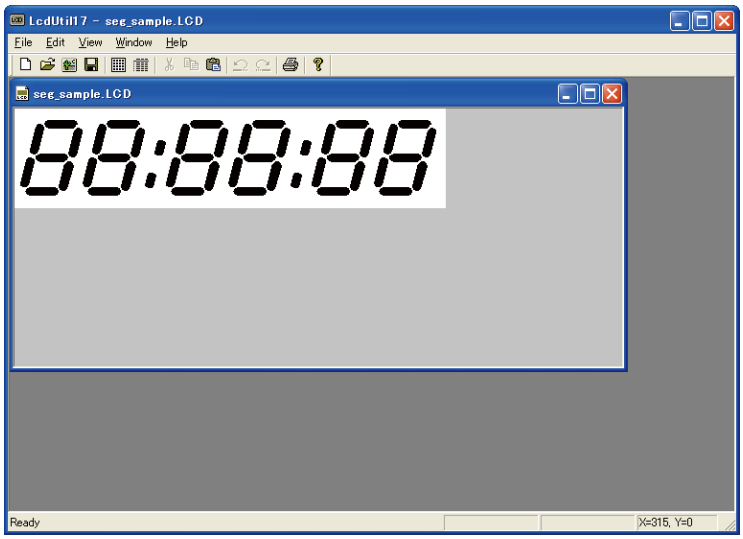
1) セグメントLCD用にアイコンのビットマップファイルを用意します。
ビットマップファイルは一般のペイントソフトで作成できますが、以下の点に注意してください。

色数とファイル形式
背景を白、アイコンを黒で作成し、白黒ビットマップ形式(.bmp)で保存してください。
※カラーのビットマップファイルを読み込むことは可能ですが、正常に2値化されない場合があります。

サイズ
ビットマップファイルのサイズは1280×1024以下で作成してください。

Cファイル数
アイコンのいくつかを別のビットマップファイルに分けて作成し、LCDUtil17上で１つのセグメントLCDパネルにすることができます。ただし、LCDUtil17には簡易的な編集機能しかありませんので、1つのビットマップファイルに作成しておくことを推奨します。

2) ビットマップファイルを読み込みます。
LCDUtil17を起動し、[File]メニューから[Open Bitmap File]を選択します。
作成したビットマップファイルを選択してください。
パネル編集ウィンドウが開き、読み込んだビットマップを表示します。



3) アイコンのレイアウトを編集します。

アイコンの状態
アイコンの状態は以下の３種類があります。
黒：COM/SEG情報　未設定
赤：COM/SEG情報　設定済
青：選択状態

アイコンの編集
選択状態のアイコンに対して、以下の操作ができます

位置変更
マウスによるドラッグで、アイコンの移動ができます。また、別のパネル編集ウィンドウからアイコンをドラッグアンドドロップすることでアイコンの移動が行えます。

カット、コピー
[Edit]メニューまたはツールバーにあるボタンをクリックしてください。

ペースト
[Edit]メニューまたはツールバーにあるボタンをクリックしてください。
アイコンはLCDパネルの左上の位置にペーストされます。

削除
[Delete]キーで削除できます。

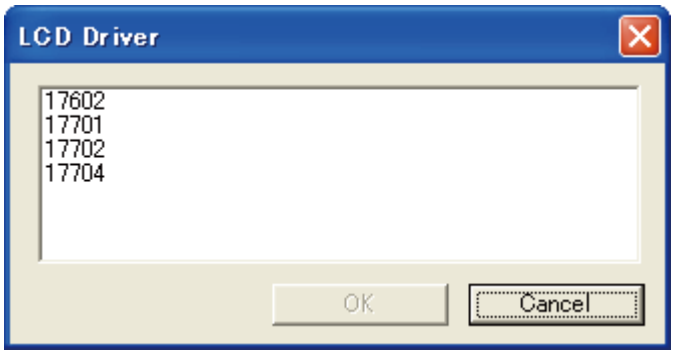
グループ化
[Ctrl]キーを押しながらアイコンをクリックすることで、複数のアイコンを選択できます。
この状態で[Edit]メニューから[Group Icon]を選択すると、選択されたパーツがグループ化され、1つのアイコンとして選択できます。
グループ化を解除するには、[Edit]メニューから[Release Group]を選択してください。

注：グループ化したときのCOM/SEG情報は、最後に選択したアイコンのCOM/SEG情報が反映されます。グループ化解除したときのアイコンのCOM/SEG情報は、グループ化していた時のCOM/SEG情報が残ります。

注：LCDUtil17の編集機能は簡易的なものです。できるだけビットマップ作成時にレイアウトを決定しておくことを推奨します。

4) アイコンにポートの割り付けを行います。
アイコンをダブルクリックします。
[LCD Driver]ダイアログボックスが表示されます。

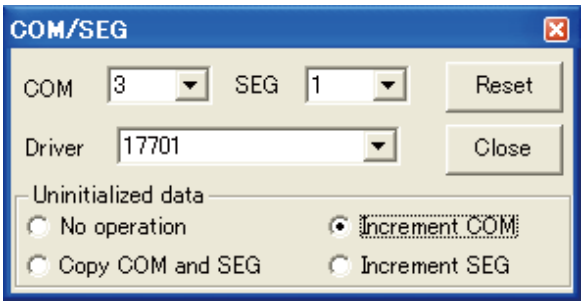
4-1) LCDドライバが設定されていない場合は[LCD Driver]ダイアログボックスが表示されます。
リストから開発する機種を選択して[OK]ボタンをクリックします。



※このダイアログボックスは、LCDドライバが設定されていない場合にのみ表示されます。すでに作成済みのLCDファイルを読み込んだ場合や、IDEでプロジェクトを選択した状態でIDEの[LCDUtilityを起動]ボタンをクリックした場合は、LCDドライバがすでに設定されており、このダイアログボックスは表示されません。

注：LCDドライバは1度設定すると変更することができませんので、選択には注意してください。

4-2) [COM/SEG]ダイアログボックスが表示されます。



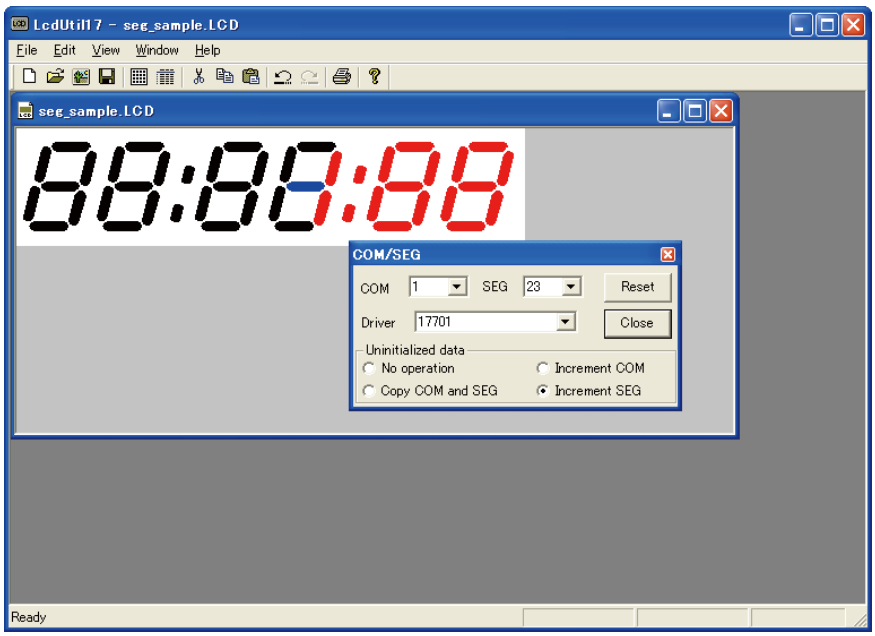
COM、SEG
プルダウンリストから選択します。COM/SEGの割り付けは変更直後に反映されます。

[Reset]ボタン
クリックするとCOM/SEGの内容がクリアされ、COM/SEG未設定状態となります。

Driver
現在設定されているLCDドライバ名を表示します。変更はできません。
COM/SEG値の上限はLCDドライバで設定されています。

Uninitialized data
COM/SEG未設定のアイコンをクリックしたときの設定を決めることができる。

No operation	COM/SEG情報を変更しません。
Copy COM and SEG	表示しているCOM/SEG情報をアイコンに設定します。
Increment COM	表示しているCOM/SEG情報に、COMをインクリメント(+1)して設定します。
Increment SEG	表示しているCOM/SEG情報に、SEGをインクリメント(+1)して設定します。



注：アイコンをコピー＆ペーストした場合や、他のパネル編集ウィンドウから移動したアイコンは、ポート割り付け情報は破棄されます。割り付け情報のないドットマトリクスは黒で表示されます。

5) アイコンリストの表示を行います。
[Edit]メニューから[Icon List]を選択します。
現在アクティブなパネル編集ウィンドウ内のアイコンの一覧が表示されます。

リスト内のアイコンをクリックして選択すると、パネル編集ウィンドウ内のアイコンも青く表示され、どのアイコンに対応しているか分かります。
このウィンドウでもアイコンのCOM/SEG情報を編集することができます。

12.8.7 ショートカットキーリスト

LCDUtil17で使用できるショートカットキーの一覧を示します。

表 12.8.7.1 ショートカットキー一覧

機能	ショートカットキー
コピー	Ctrl ＋ C Ctrl ＋ Insert
カット	Ctrl ＋ X Delete
ペースト	Ctrl ＋ V Shift ＋ Insert
アイコンリスト	Ctrl ＋ I
ドットマトリクス	Ctrl ＋ M
新規作成	Ctrl ＋ N
オープン	Ctrl ＋ O
上書き保存	Ctrl ＋ S
プリント	Ctrl ＋ P
Undo	Ctrl ＋ Z Alt ＋ BS
Redo	Ctrl ＋ Y

12.8.8 ワーニングメッセージ/エラーメッセージ

LCDUtil17で表示されるワーニングメッセージの一覧を示します。

表12.8.8.1　ワーニングメッセージ一覧

No.	新規作成/ファイルオープン時
1	・空のドキュメントの作成に失敗しました。 ・新しいドキュメントを作成できません。
	原因： 100ウィンドウを超えたため、新たにウィンドウを作成できなかった。メモリ不足で新たにウィンドウを作成できなかった。
	対応： ウィンドウ数を99以下にしてください。他のアプリケーションを閉じてメモリを解放してください。
2	予期しないファイル形式です。
	原因： 正常なlcd形式、bmp形式のファイルが読み込まれなかった。サポートしていないドライバのlcdファイルだった。
	対応： LcdUtil17対応のファイルを指定してください。
	値入力時
3	・????から????までの整数を入力してください。 ・????から????までの有効な数値を入力してください。
	原因： 範囲外の値が入力されている。
	対応： 正しい値を入力してください。
4	・整数を入力してください。 ・有効な数字を入力してください。次の値は無効です：空白スペース、小数、0、+、-
	原因： 整数入力のエディットボックスに数字以外が入力されている。
	対応： 正しい値を入力してください。
	ドットマトリクス設定画面([Preview]ボタンクリック時)
5	Invalid ????（???? = position、number of dots、dot size、pitch size、start COM/SEG）
	原因： ???? に不正な値が入っている。
	対応： 正しい値を入力してください。
6	Invalid ??? start number or Invalid number of dots　（??? = COM、SEG）
	原因： 設定されたStart ??? とドット数、COM/SEG割り付け方向では、ドライバのCOM/SEG 数を超える。
	対応： Start ??? とドット数、COM/SEG割り付け方向を、ドライバのCOM/SEG数に合わせてください。
No.	Save as時
7	Some icons/matrixes are out of LCD panel.　Remove them?　[OK][キャンセル]
	原因： LCDパネルからはみ出したアイコン/ドットマトリクスがある状態で、保存しようとした。
	対応： アイコン/ドットマトリクスをはみ出さないようにするか、[OK]ボタンをクリックしてはみ出したアイコン/ドットマトリクスを削除して保存してください。
	アイコン作成時/ファイル読み込み時
8	The number of the icon is exceeding 4096 pieces.
	原因： アイコンの総数が上限の4096を超えた。
	対応： アイコンの総数が4096個を超えないようにしてください。
9	The number of the icon is exceeding 4096 pieces. Loaded the icons to 4096 pieces.
	原因： アイコンの総数が上限の4096を超えた。
	対応： アイコンの総数が4096個を超えないようにしてください。
	マトリクス作成時/ファイル読み込み時
10	The number of the matrix is exceeding 10 pieces.
	原因： ドットマトリクスの総数が上限の10を超えた。
	対応： ドットマトリクスの総数が10個を超えないようにしてください。
	Save時/変更を保存せず閉じるときのSave時

11	This file was not saved.
	原因： はみ出したアイコン/ドットマトリクスの削除をしなかった。
	対応： アイコン/ドットマトリクスがLCDパネルからはみ出した状態で保存はできません。

LCDUtil17で表示されるエラーメッセージを示します。

表12.8.8.2　エラーメッセージ一覧

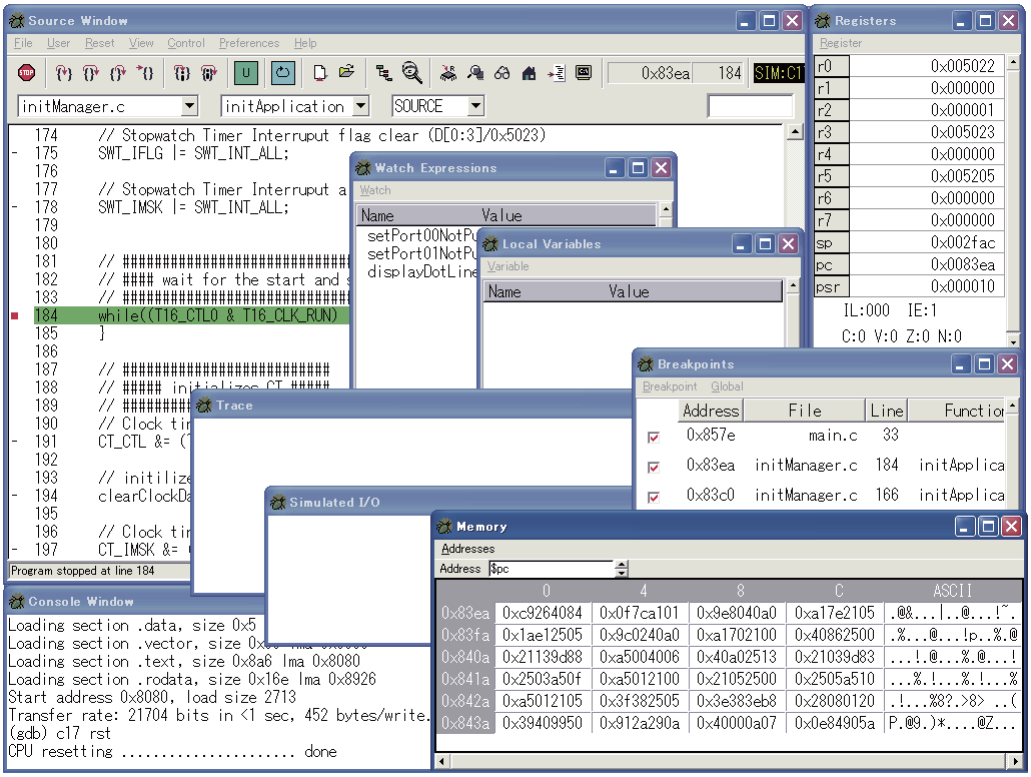
No.	起動時
1	Cannot open bitmap file. The size of the panel is too large.
	原因： ビットマップファイルの縦横サイズが大きすぎる。
	対応： 横1280、縦1024以下のビットマップファイルを使用してください。

12.9 単体フラッシュライタ

ICDminiを単体フラッシュライタとして利用することが可能です。ICDminiを単体フラッシュライタとして利用するには、あらかじめ書き込みたいデータやアドレスを、デバッガを介してICDminiのフラッシュメモリへ書き込みます。そしてICDminiのFlashライタモードで使用することで、デバッガを介さず容易にターゲットへの書き込みを可能にします。
ICDminiのフラッシュメモリへのデータやアドレスの書き込みは、デバッガのコマンドによって実現されます。またFlashライタモードはICDmini本体側面のディップスイッチを設定することで選択することができます。
ICDminiの単体フラッシュライタでは、ERASE、WRITE、VERIFYの３処理が実現されており、ICDmini本体側面のディップスイッチを設定することで実行する処理を選択できます。
上記のすべての設定が終わると、ICDmini上部のRESET/STARTボタンを押すことで上記で選択したフラッシュライタ処理が開始されます。選択された処理の実行状態はICDminiのLEDによって判断できます。
同じデータを再度書く場合は、初期設定は不要でICDmini上部のRESET/STARTボタンを押すだけでフラッシュライタを実行することができます。

単体フラッシュライタのために用いるデバッガコマンドについては、"10.7.15 フラッシュライタコマンド"を参照してください。ICDminiのFlashライタモードと、その単体フラッシュライタの実行手順については、"S5U1C17001HManual"を参照してください。

12.10 旧バージョンのデバッガ



今回のバージョンのS1C17 Family C/C++コンパイラパッケージは、バージョン1.5.0以前のデバッガを起動することが出来ます。起動方法、操作方法などは、バージョン1.5.0のマニュアルを参照してください。バージョン1.5.0のマニュアルはSEIKO EPSONのユーザサイトよりダウンロードしてください。

S1C17 Family C Compiler Package

Quick Reference

EPSON

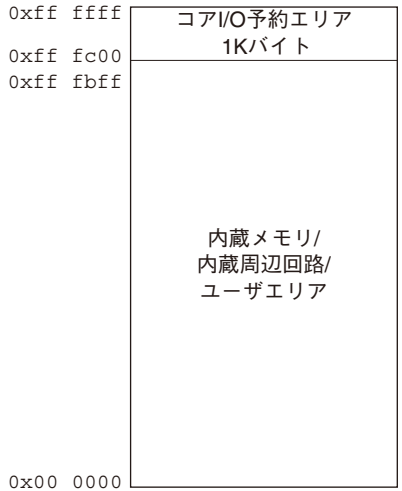
CMOS 16-bit Single Chip Microcomputer
S1C17 Family C Compiler Package

Quick Reference for Development

メモリマップとトラップテーブル (S1C17コア)

S1C17 Core

メモリマップ



トラップテーブル

No.		ベクタアドレス
0 (0x00)	リセット	TTBR + 0x00
1 (0x01)	アドレス不整割り込み	TTBR + 0x04
2 (0x02)	NMI	TTBR + 0x08
3 (0x03)	マスク可能な外部割り込み3	TTBR + 0x0c
:	:	:
31 (0x1f)	マスク可能な外部割り込み31	TTBR + 0x7c

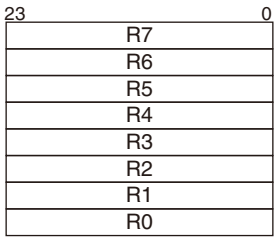
TTBR: トラップテーブル先頭アドレス
(0xffff80番地から読み出し可能)

Reference

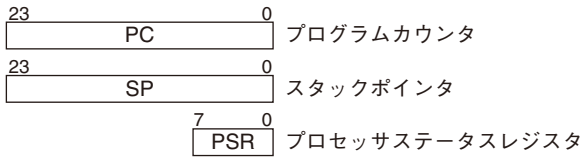
レジスタ (S1C17コア)

S1C17 Core

汎用レジスタ (8)



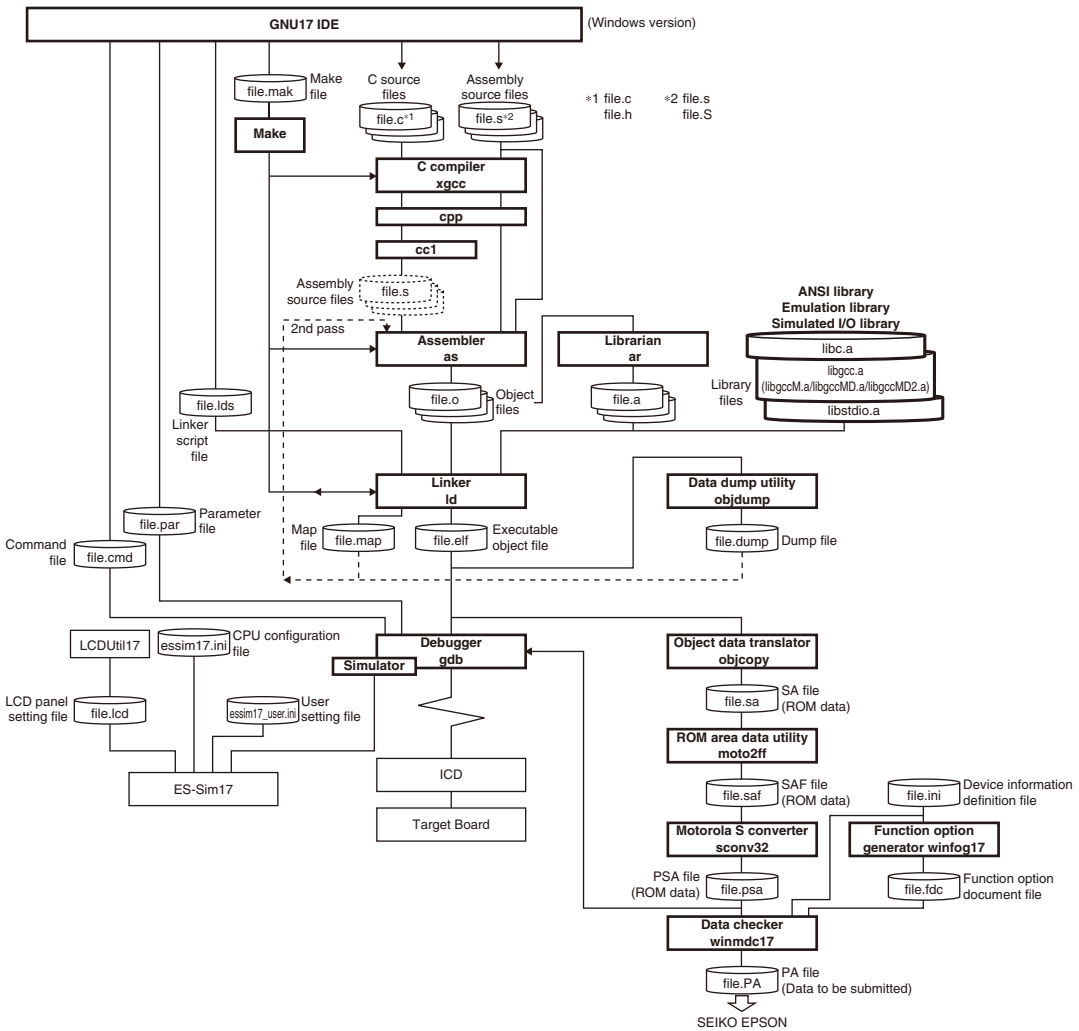
特殊レジスタ (3)



PSR

	7	6	5	4	3	2	1	0
	IL[2:0]			IE	C	V	Z	N
初期値	0	0	0	0	0	0	0	0

IL[2:0]: 割り込みレベル (0~7: 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カスクリプトを編集します。(ユーザ作成のmakeファイルとリンカ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ガ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からデバッガ**gdb**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。

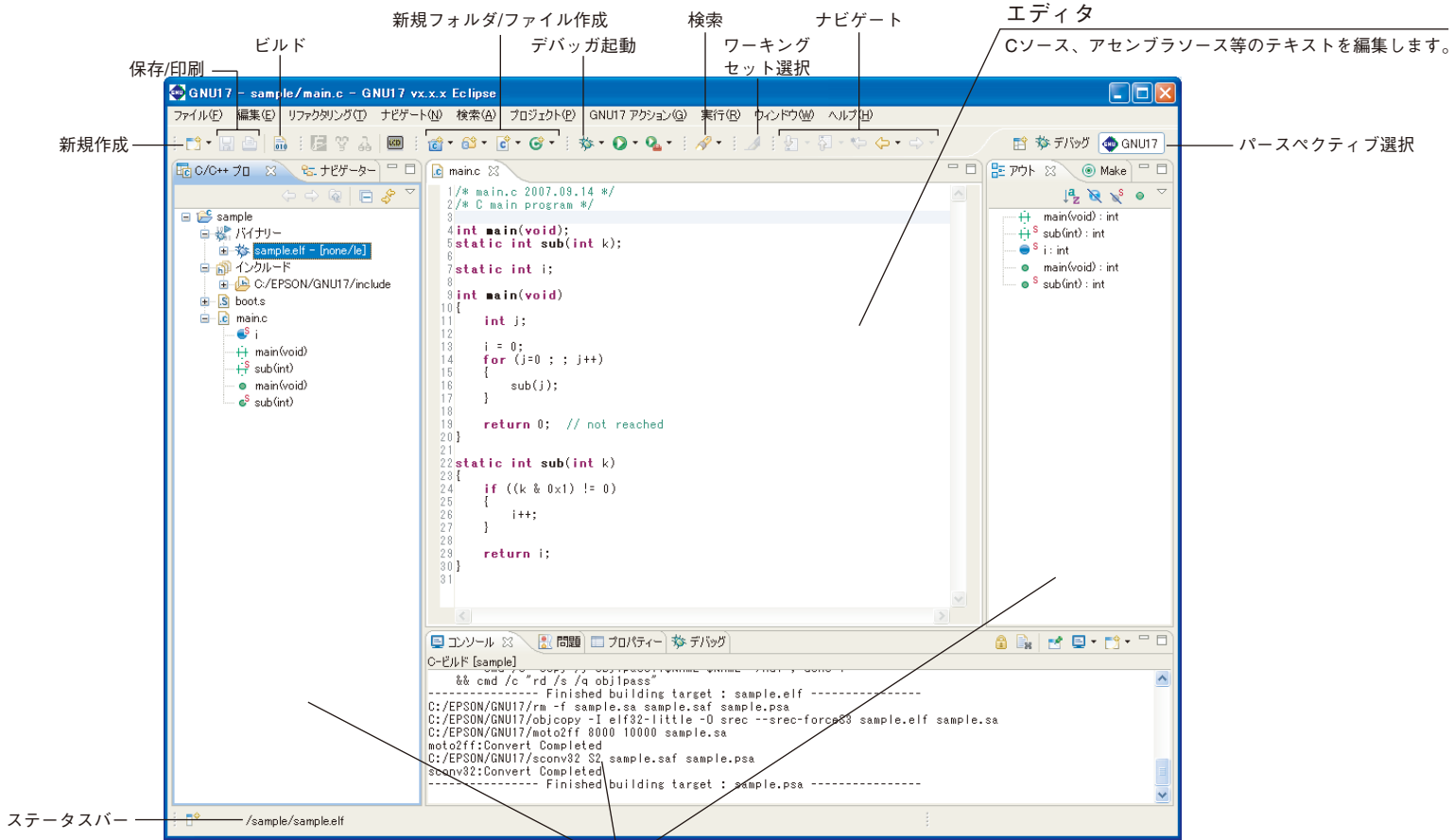
5. PAファイル(提出用データ)の作成
   プログラムの完成後、提出用データを作成します。
   5-1) objcopy、**moto2ff**、**sconv32**を使用して、PSAファイル(ROMデータ)を生成します。
   5-2) winfog17を使用して、FDCファイル(ファンクションオプションドキュメント)を生成します。
   5-3) PSAファイルとFDCファイルを**winmdc17**で1つのPAファイルにパックします。
   5-4) できあがったPAファイルをセイコーエプソンに提出してください。

概要

GNU17 IDEは、S1C17 Family Cコンパイラパッケージ(S5U1C17001C)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
```



Reference

メニューバー

[ファイル]メニュー

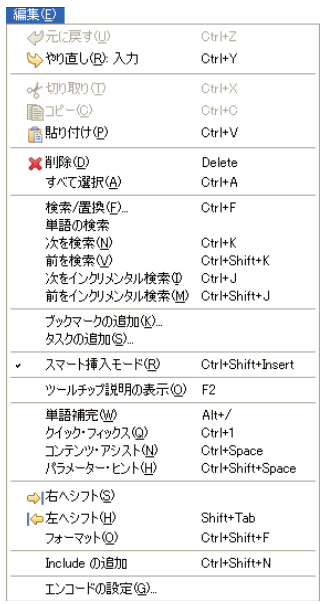
新規 (Alt+Shift+N)　プロジェクト、ファイル、フォルダを新規作成します。
ファイルを開く...　エディタで開くファイルを選択します。
閉じる (Ctrl+W)　エディタ最前面のファイルを閉じます。
すべて閉じる (Ctrl+Shift+W)　エディタ上のすべてのファイルを閉じます。
保管 (Ctrl+S)　エディタ最前面のファイルを保存します。
別名保管 ...　エディタ最前面のファイルを別名で保存します。
すべて保管 (Ctrl+Shift+S)　エディタ上のすべてのファイルを保存します。
前回保管した状態に戻す　エディタ最前面のファイルの編集内容を破棄し、前回の保存内容に戻します。
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。
プロパティー (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。

[編集]メニュー

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。

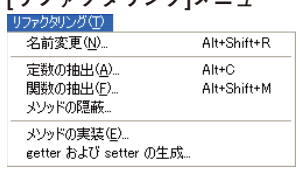
メニューバー

[編集]メニュー



すべて選択 (Ctrl+A)　エディタ最前面の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。
単語の検索　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。
ツールチップ説明の表示 (F2)　[F2] キーを押すと、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・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。

[リファクタリング]メニュー



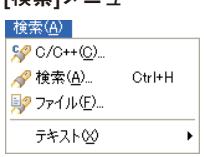
名前変更 ... (Alt+Shift+R)　選択した関数や変数の名称を一括して変更します。
定数の抽出 (Alt+C)　ソースファイルの定数を変数に切り出します。
関数の抽出 (Alt+Shift+M)　ソースファイルのコードの一部を関数に切り出します。

メニューバー

[ナビゲート]メニュー

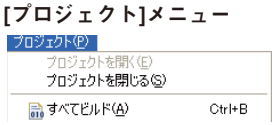
次へジャンプ　[C/C++ プロジェクト]/[ナビゲーター]ビューを、現在選択しているフォルダ内のみを表示するように切り換えます。
ジャンプ　[C/C++ プロジェクト]/[ナビゲーター]ビューの表示を、履歴に沿ってナビゲートします。
宣言を開く (F3)　選択しているオブジェクトの宣言・定義を開きます。
型階層を開く (F4)　選択している変数の型階層を開きます。
呼び出し階層を開く (Ctrl+Alt+H)　選択している関数の呼び出し階層を開きます。
組み込みブラウザーを開く (Ctrl+Alt+I)　選択しているソースファイルのインクルード階層を開きます。
ソース/ヘッダーの切り替え (Ctrl+Tab)　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。
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。

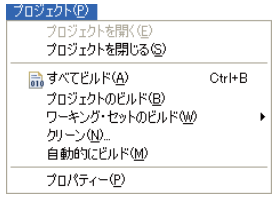
[プロジェクト]メニュー



プロジェクトを開く　[C/C++ プロジェクト]/[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。

メニューバー

[プロジェクト]メニュー

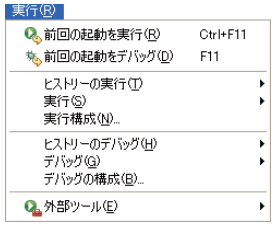


プロジェクトのビルド　[C/C++ プロジェクト]/[ナビゲーター]ビュー上で選択されているプロジェクトのビルドを実行します。
ワーキング・セットのビルド　指定のワーキングセットに含まれるリソースのビルドを実行します。
クリーン...　クリーンまたはリビルドを実行します。
自動的にビルド　オートビルド機能をON/OFFします
プロパティー　[C/C++ プロジェクト]/[ナビゲーター]ビュー上で選択されているプロジェクトのプロパティを表示/編集します。

[GNU17 アクション]メニュー

WinFog17を起動　FDCファイル(ファンクションオプションドキュメント)を作成するwinfog17.exeを起動します。
WinMdc17でパック　winmdc17.exeを起動し、PAファイル(提出用データ)(<プロジェクト名>.pa)を生成します。
WinMdc17でアンパック　winmdc17.exeを起動し、PAファイル(<プロジェクト名>.pa)をアンパックします。
LcdUtilityを起動　LCDユーティリティ（LcdUtil17.exe）を起動します。
Flash ROM書き込み　プログラムをFlash ROMへ書き込みます。
Flashセキュリティの解除　Flashセキュリティを解除します。

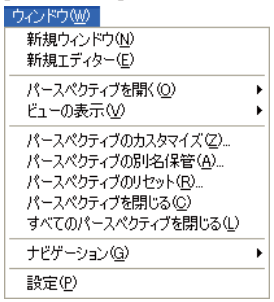
[実行]メニュー



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。
ヒストリーのデバッグ　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。
デバッグの構成...　デバッガgdbの起動構成ダイアログを開きます。

メニューバー

[ウィンドウ]メニュー



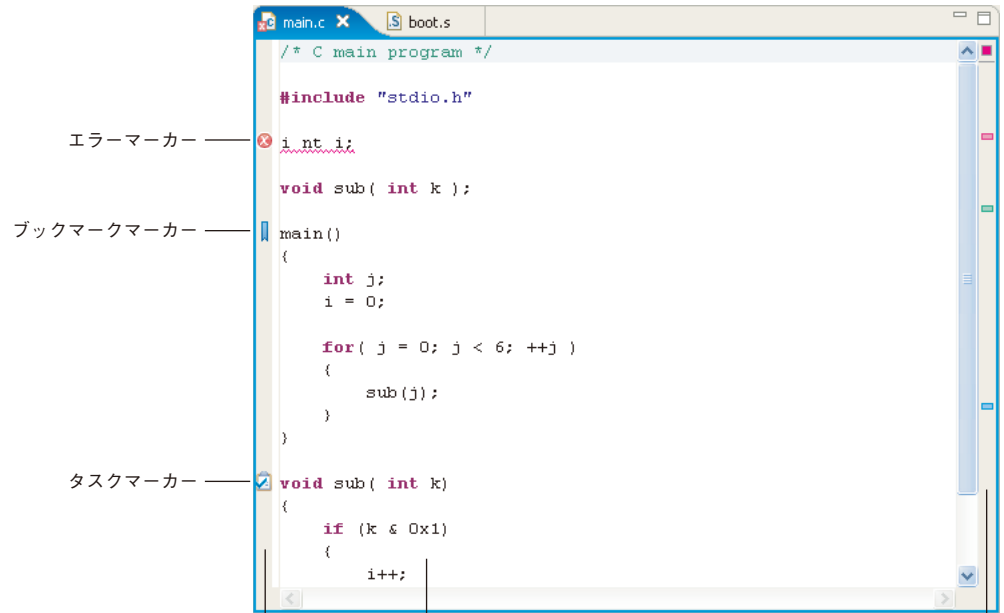
新規ウィンドウ　新しいウィンドウを開きます。
新規エディター　編集中のファイルを新しいエディタタブで開きます。
パースペクティブを開く　パースペクティブを開きます。
ビューの表示　ビューを開きます。
パースペクティブのカスタマイズ...　現在のパースペクティブの設定を変更します。
パースペクティブの別名保管...　現在のパースペクティブの設定を別名で保存します。
パースペクティブのリセット　パースペクティブを初期状態に戻します。
パースペクティブを閉じる　現在のパースペクティブを閉じます。
すべてのパースペクティブを閉じる　読み込まれているすべてのパースペクティブを閉じます。
ナビゲーション　エディタやビューをナビゲートします。
設定...　IDEの環境設定を行う[設定]ダイアログを表示します。

[ヘルプ]メニュー

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 GNU17 V*x.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。

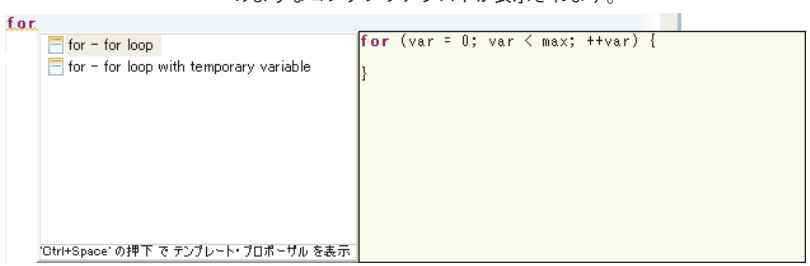


マーカーバー

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。

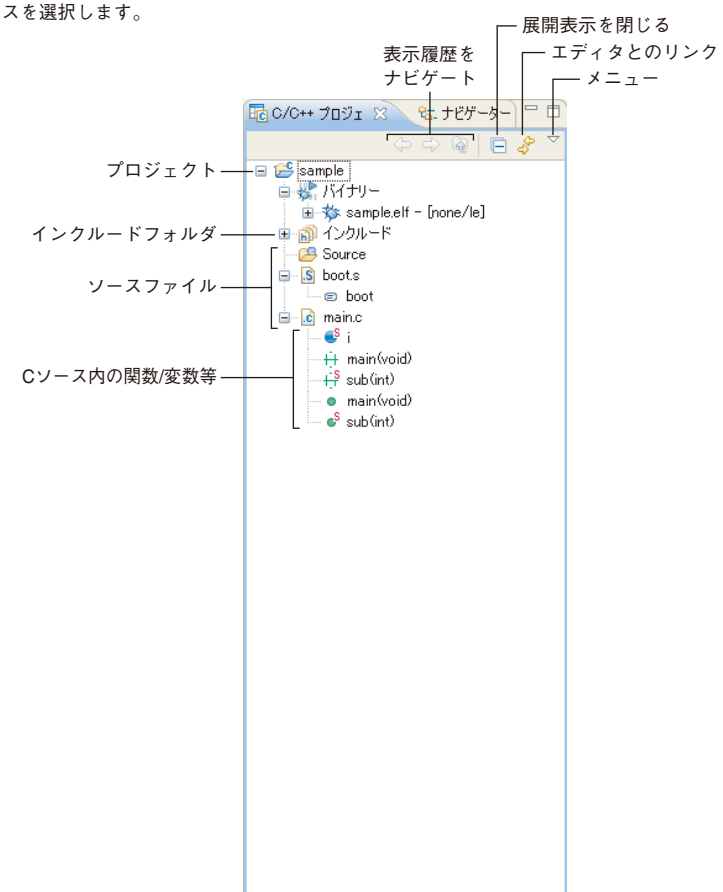


オーバービュールーラ

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。

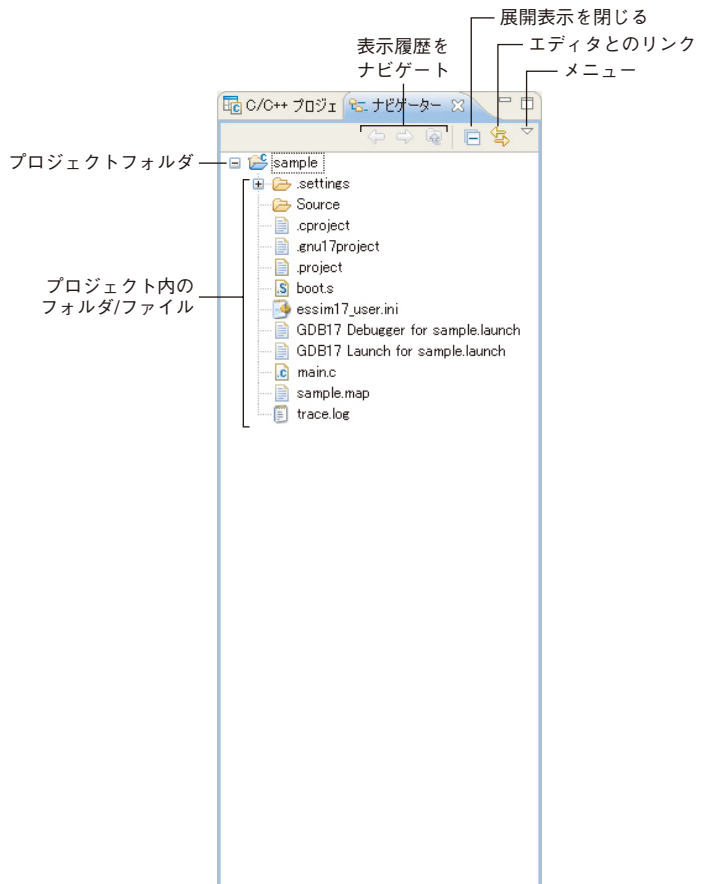
[C/C++ プロジェクト]ビュー

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。



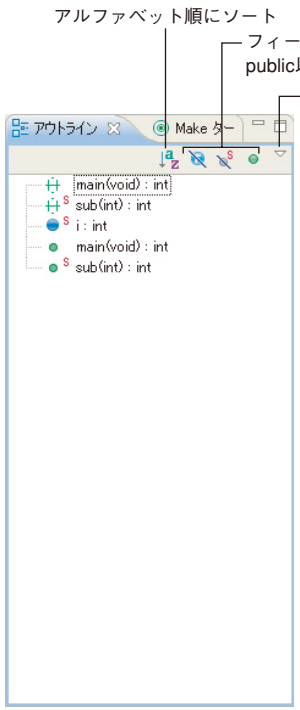
[ナビゲーター]ビュー

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ソース内に記述されている関数や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。

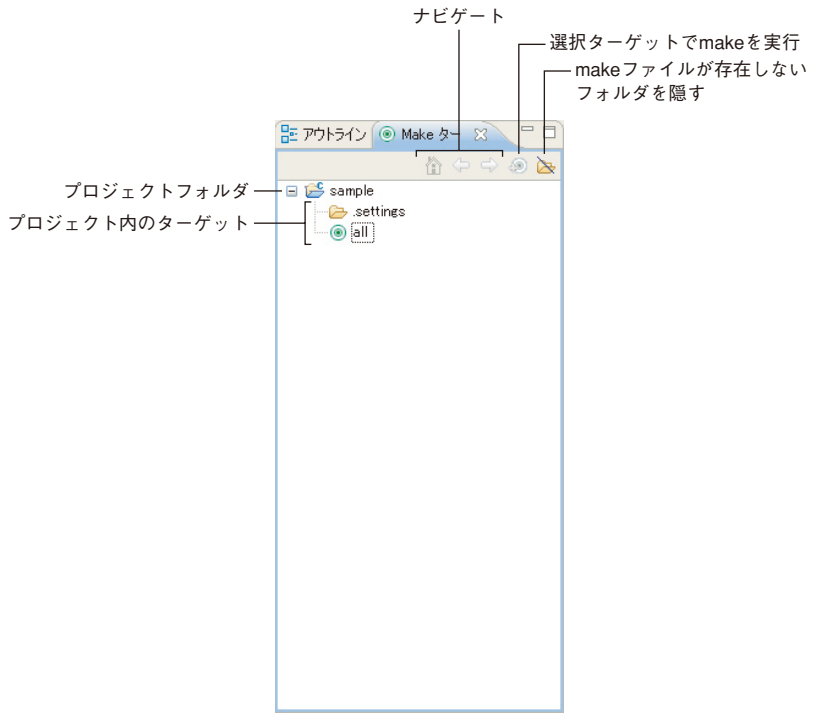


アイコン

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

[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。

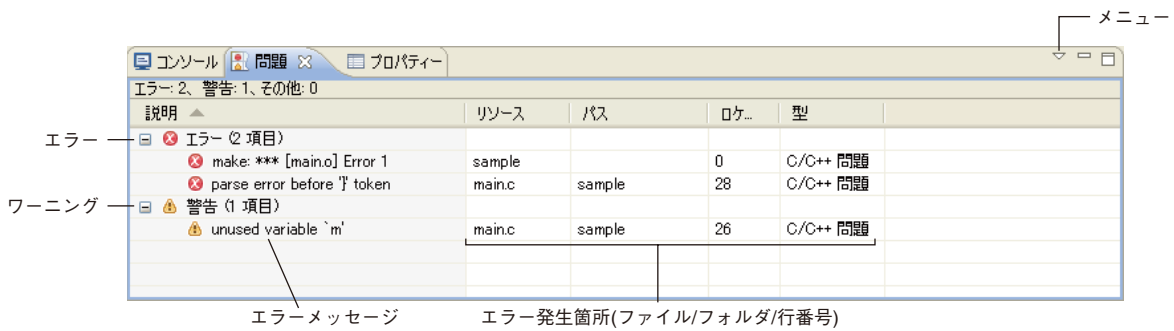


[コンソール]ビュー

実行されたコマンドラインやGNU17ツールが出力するメッセージを表示します。

[問題]ビュー

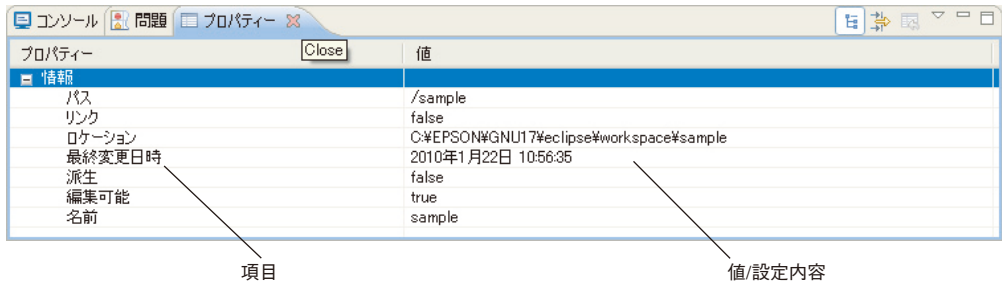
ビルド時に発生したエラーを表示します。ソースファイル内のエラーはエラーメッセージをクリックすることで、エディタ上のエラー発生箇所にジャンプします。



Reference

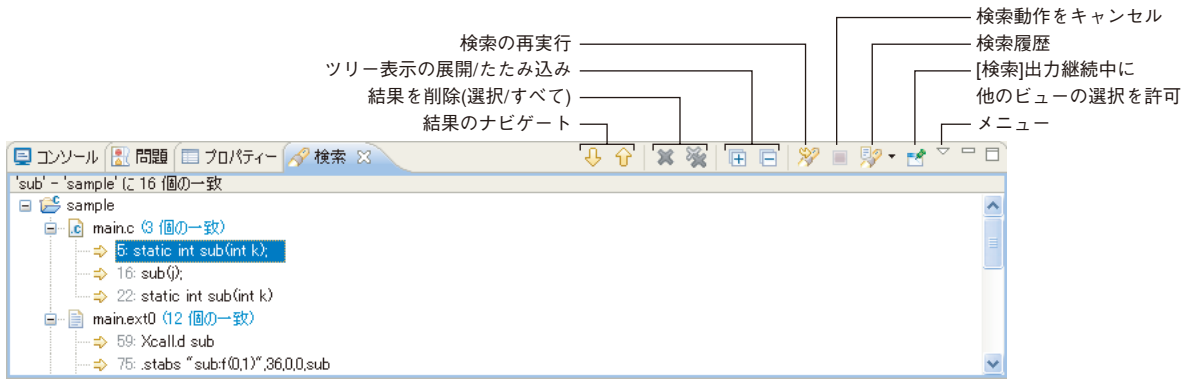
[プロパティー]ビュー

[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。



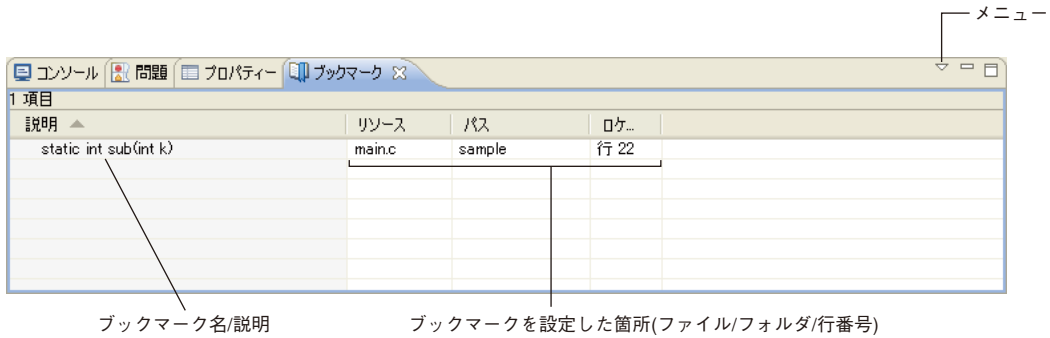
[検索]ビュー

[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。



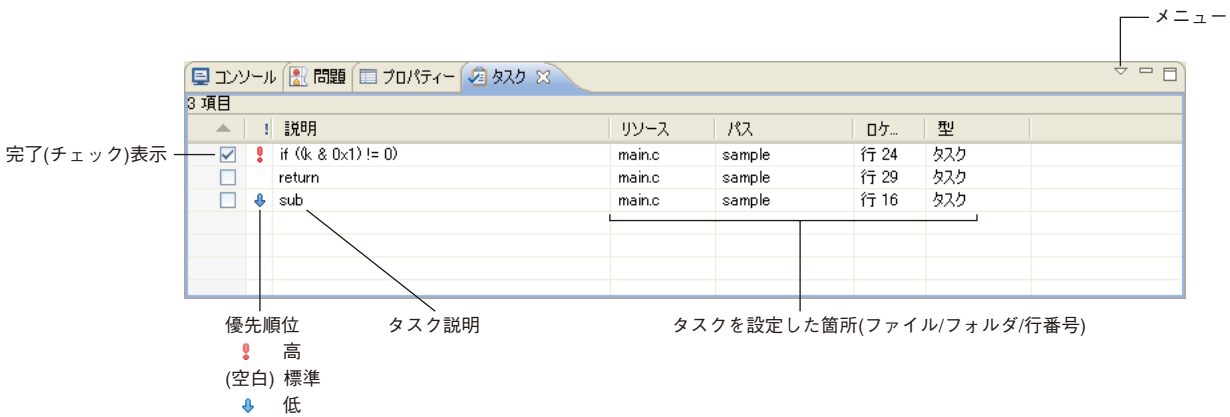
[ブックマーク]ビュー

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。

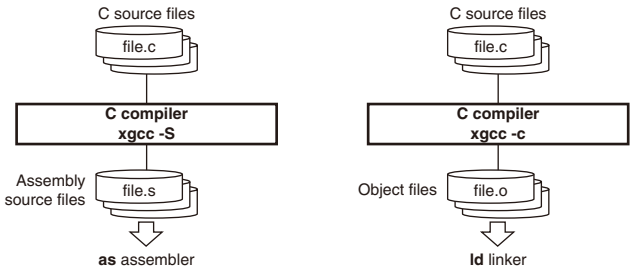


Reference

概要

ANSI Cに準拠したCコンパイラで、GNU 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**を連続して呼び出し、Cソースファイルをコンパイルして S1C17 Family用のアセンブリソースファイルを生成します。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。**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ソースファイル名

例: `xgcc -c -gstabs test.c`

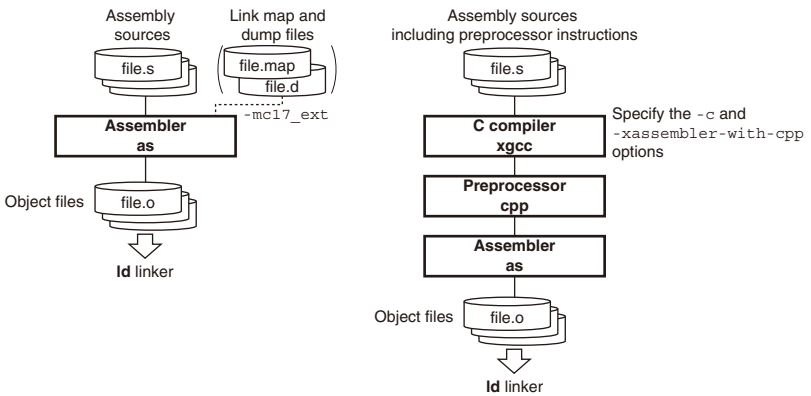
主要なコマンドラインオプション

`-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
`-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,-O3`	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
`-mpointer16`	16ビット(64KB)データ空間用コードの生成
`-mshort-offset`	20ビット(1MB)空間用コードの生成
`-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
`-Wa,`*<option>*	アセンブラオプションの指定

概要

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S1C17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 -o test.o -adhl 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: `-adhl` (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パスを含むデバッグ情報の付加
`-mpointer16`	16ビットポインタモードの指定
`-mc17_ext` *<dumpfile>* *<mapfile>*	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 / マクロ定義
`#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
`.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>*	1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

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。
junk at end of line: '*XXXXX*	オペランド書式のエラーです。
XXXXXX: invalid register name	使用できないレジスタを指定しています。
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。
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.	`-mc17_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。
Cannot find the map file.	`-mc17_ext`が指定されていますが、マップファイル名が指定されていません。あるいは、指定されたマップファイルが存在しません。
The format of the dump file is invalid.	`-mc17_ext`で指定されているダンプファイルの内容が不正です。
The format of the map file is invalid.	`-mc17_ext`で指定されているマップファイルの内容が不正です。
Cannot close the map file.	`-mc17_ext`で指定されたマップファイル読み込み後、ファイルのクローズに失敗しました。
There are too many characters of one line in dump file.	`-mc17_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。
There are too many characters of one line in map file.	`-mc17_ext`で指定されたマップファイルの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: Value of XXXX too large for field of AAA bytes at BBB	`.text`セクションの先頭から XXXXバイト目にあるラベルのアドレスは大きすぎます。このラベルは`.rodata`セクションの先頭からBBBバイト目にあるAAAバイト型のシンボルテーブルから参照されています。
Error : Failed to register hash symbols in source file. :XXXX	`-mc17_ext`が指定されていますが、ソースファイルのシンボル名の登録に 'XXXX' が失敗しました。
Error : Failed to register hash symbols in dump file. :XXXX	`-mc17_ext`が指定されていますが、ダンプファイルのシンボル名の登録に 'XXXX' が失敗しました。
Error : Failed to register hash symbols in map file. :XXXX	`-mc17_ext`が指定されていますが、マップファイルのシンボル名の登録に 'XXXX' が失敗しました。

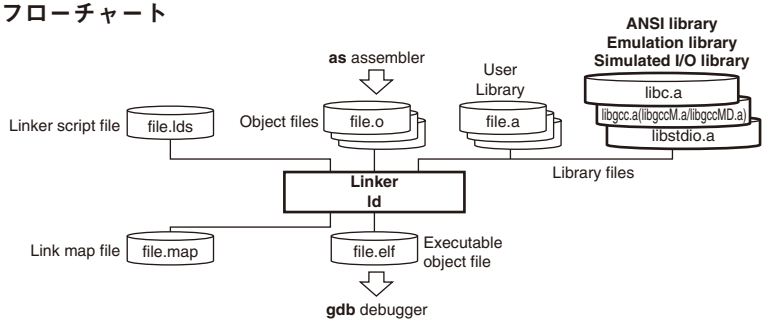
ワーニングメッセージ

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\24bit\libc.a ..\lib\24bit\libgcc.a ..\lib\24bit\libc.a`

主要なコマンドラインオプション

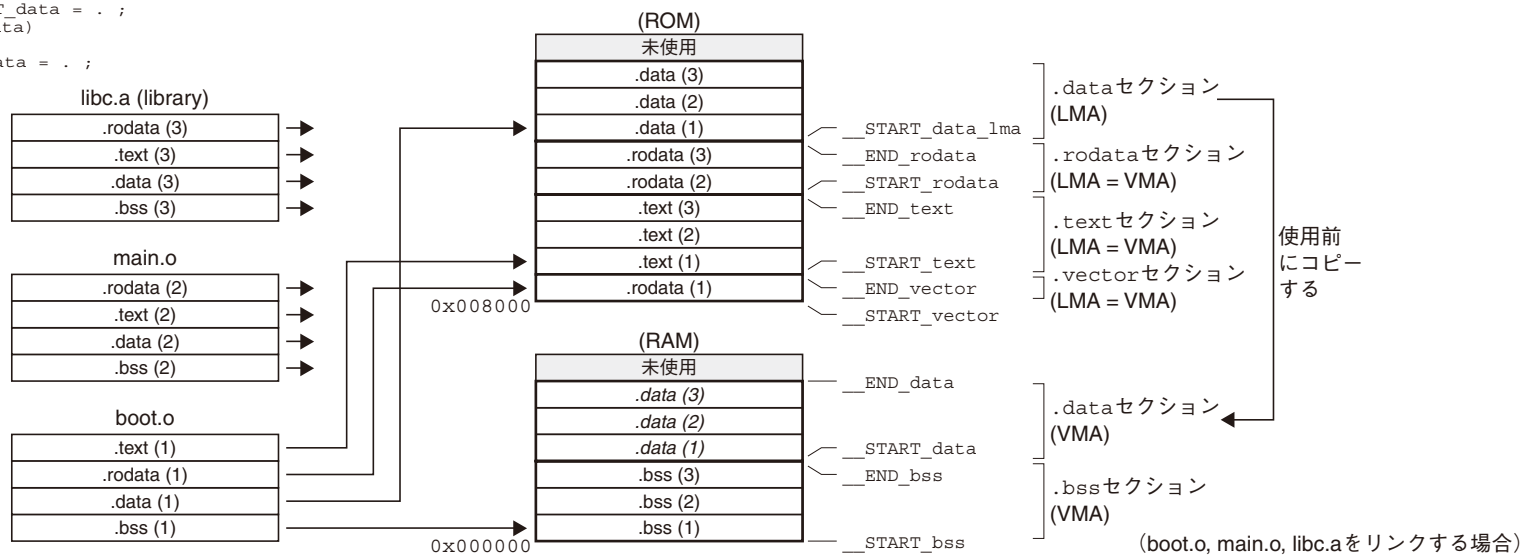
`-o` *<filename>*	出力ファイル名の指定
`-T` *<filename>*	リンカスクリプトファイルの読み込み
`-M`	リンクマップのstdout出力
`-Map` *<filename>*	リンクマップのファイル出力
`-N`	データセグメントのアライメントチェックを禁止
`-c17-overlap-noerr`	セクションのアドレス配置の重複を許可
`-c17-memoryover-noerr`	メモリ領域オーバーエラーの禁止

エラーメッセージ

The offset value of a symbol is over 16bit.	シンボルのアドレスが16ビットアドレス空間を超えています。
The offset value of a symbol is over 24bit.	シンボルのアドレスが24ビットアドレス空間を超えています。
Input object file use both 16bit and 24bit address mode.	24ビットポインタモードで作成されたオブジェクトファイルと、16ビットポインタモードで作成されたオブジェクトファイルをリンクしようとしています。
section *XXX* is not within 16bit address.	*XXX*セクションのアドレスが16ビットアドレス空間を超えています。
section *XXX* is not within 24bit address.	*XXX*セクションのアドレスが24ビットアドレス空間を超えています。

IDEが作成するデフォルトリンカスクリプトファイル

```
/* Linker Script file generated by Gnu17 Plug-in for Eclipse */
OUTPUT_FORMAT("elf32-c17", "elf32-c17", "elf32-c17")
OUTPUT_ARCH(c17)
SEARCH_DIR(.);

SECTIONS
{

 /* stack pointer symbols */
__START_stack = 0x000FC0;

 /* location counter */
 . = 0x0;

 /* section information */
 .bss 0x000000 :
   {
    __START_bss = . ;
    boot.o(.bss)      \
    main.o(.bss)       files(.bss)
    libc.a(.bss)      /
   }
   __END_bss = . ;

 .data __END_bss : AT( __END_rodata )
   {
    __START_data = . ;
    files(.data)
   }
   __END_data = . ;

 .vector 0x008000 :
   {
    __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 = . ;

 __START_data_lma = __END_rodata ;
 __END_data_lma = __END_rodata + ( __END_data - __START_data );
}
```



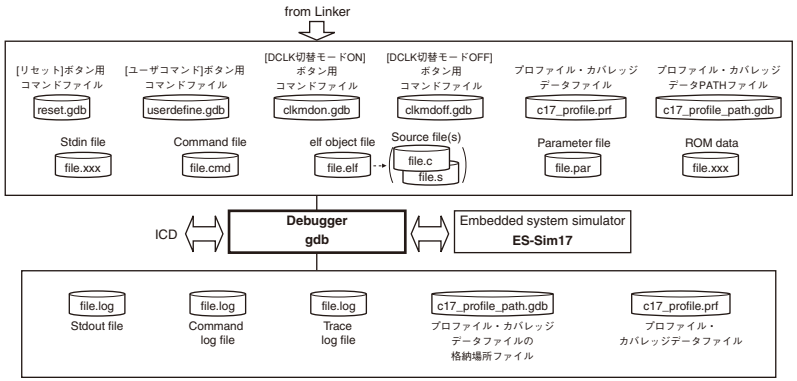
（boot.o, main.o, libc.aをリンクする場合）

Reference

概要

ICD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



起動コマンド

gdb *<options>*

例: `gdb -x sample.cmd --cd=/cygdrive/c/EPSON/gnu17/sample`

コマンドラインオプション

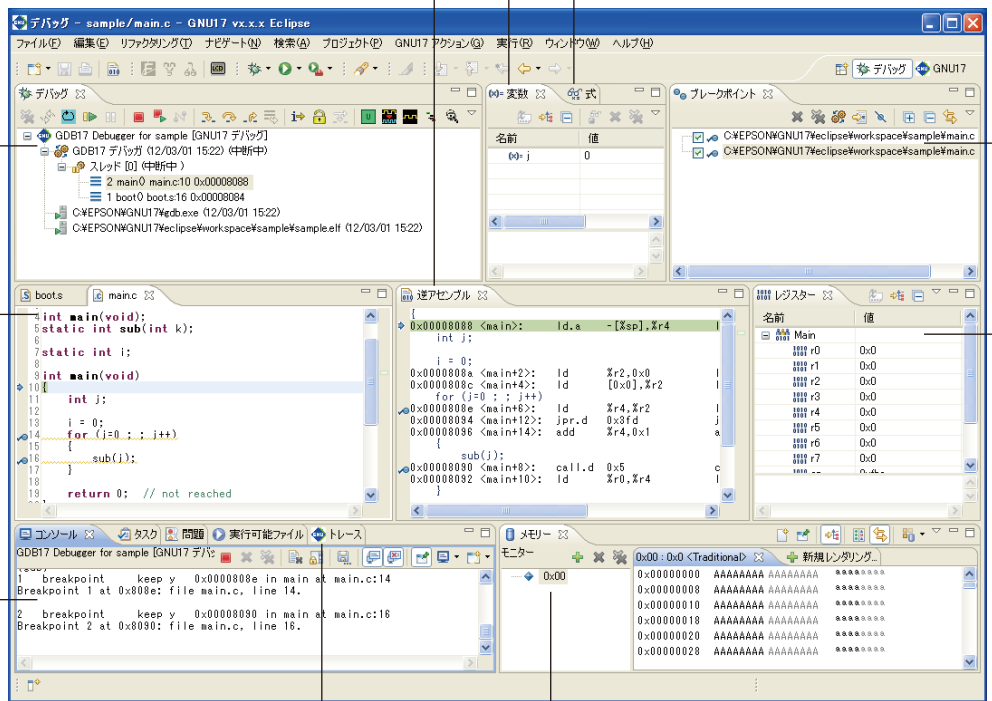
`--command=`*<filename>*	コマンドファイルの指定
`-x` *<filename>*	コマンドファイルの指定
`--c17_cmw=`*<seconds>*	コマンドファイル内のコマンド実行間隔の指定
`--cd=`*<path>*	カレントディレクトリの変更
`--directory=`*<path>*	ソースファイルディレクトリの変更
`--c17_double_starting`	二重起動の許可

デバッグパースペクティブ

[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。

[変数]ビュー
ローカル変数の値をモニタするのに使用します。

[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、その値をモニタするのに使用します。



[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。

[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。

[レジスター]ビュー
CPUレジスタの値の表示と修正に使用します。

[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
メモリ内容の表示と修正に使用します。

Reference

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]: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]: サポートしていません。
[再開]: サポートしていません。
[リセット]: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。
[Flash セキュリティの解除]: パスワードが設定されている場合、Flash セキュリティのアクセス制限を解除します。
[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。
[中断]: 実行を中断します。
[終了]: デバッガ(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]: サポートしていません。
[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]: 終了したデバッガを再起動します。
[GDB17 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。
[プロパティー]: サポートしていません。
[ユーザコマンド]: ユーザ定義コマンドを実行します。
[DCLK切替モードON]:DCLK（デバッグクロック）切替モードを有効にします。
[DCLK切替モードOFF]:DCLK（デバッグクロック）切替モードを無効にします。
[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

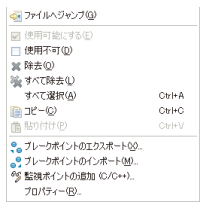
[指定行まで実行]: カ-ソル行の位置まで実行します。
[指定行で再開]: サポートしていません。
[監視式を追加...]: 監視式を登録するダイアログを開きます。
[実行]: サポートしていません。

コンテキストメニュー

[逆アセンブル]ビュー

[指定行まで実行]: カ-ソル行の位置まで実行します。
[指定行で再開]: サポートしていません。

[ブレークポイント]ビュー



[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。
[使用不可]: ブレークポイントを無効化します。
[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]: ブレークポイントを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...]: サポートしていません。
[プロパティー...]: 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。

[変数]ビュー

[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...]: サポートしていません。
[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]: 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...]: 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。レジスタ値が変更できます。

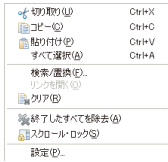
[レジスター]ビュー

[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。
[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。

[メモリー]ビュー

[監視ポイントの追加...]: サポートしていません。
[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]/[シミュレーテッドI/O]ビュー



[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]: 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。

デバッグコマンド

メモリ操作

c17 fb *addr1 addr2 data*	領域のフィル (8ビット単位)	ICD Mini/SIM
c17 fh *addr1 addr2 data*	領域のフィル (16ビット単位)	ICD Mini/SIM
c17 fw *addr1 addr2 data*	領域のフィル (32ビット単位)	ICD Mini/SIM
x /[*length*]**b** [*addr*]	メモリダンプ (8ビット単位)	ICD Mini/SIM
x /[*length*]**h** [*addr*]	メモリダンプ (16ビット単位)	ICD Mini/SIM
x /[*length*]**w** [*addr*]	メモリダンプ (32ビット単位)	ICD Mini/SIM
set {char} *addr=data*	データ入力 (8ビット単位)	ICD Mini/SIM
set {short} *addr=data*	データ入力 (16ビット単位)	ICD Mini/SIM
set {long} *addr=data*	データ入力 (32ビット単位)	ICD Mini/SIM
c17 mvb *addr1 addr2 addr3*	領域のコピー (8ビット単位)	ICD Mini/SIM
c17 mvh *addr1 addr2 addr3*	領域のコピー (16ビット単位)	ICD Mini/SIM
c17 mvw *addr1 addr2 addr3*	領域のコピー (32ビット単位)	ICD Mini/SIM
c17 df *addr1 addr2 type file* [*append*]	メモリ内容の保存	ICD Mini/SIM
c17 readmd *mode*	メモリリードモード	ICD Mini

レジスタ操作

info reg [*register*]	レジスタ表示	ICD Mini/SIM
set $*register=data*	レジスタ変更	ICD Mini/SIM

プログラム実行

continue [*Ignore*]	連続実行	ICD Mini/SIM
until *addr*	テンポラリブレーク付き連続実行	ICD Mini/SIM
step [*count*]	ステップ実行(行単位)	ICD Mini/SIM
stepi [*count*]	ステップ実行(ニーモニック単位)	ICD Mini/SIM
next [*count*]	スキップ付きステップ実行(行単位)	ICD Mini/SIM
nexti [*count*]	スキップ付きステップ実行(ニーモニック単位)	ICD Mini/SIM
finish	関数終了	ICD Mini/SIM
c17 callmd *mode* [*file*]	ユーザ関数コールモードの設定	ICD Mini/SIM
c17 call *func* [*arg1*... [*arg3*]]]	ユーザ関数コール	ICD Mini/SIM

CPUリセット

c17 rst	リセット(reset.gdbを実行)	ICD Mini/SIM
c17 rstt	ターゲットのリセット	ICD Mini

割り込み

c17 int [*intNo. level*]	割り込み発生	SIM
c17 intclear [*intNo.*]	割り込み解除	SIM
c17 int_load [*file*]	イベントファイルの読み込み	SIM

ブレーク

break [*addr*]	ソフトウェアPCブレーク設定	ICD Mini/SIM
tbreak [*addr*]	テンポラリソフトウェアPCブレーク設定	ICD Mini/SIM
hbreak [*addr*]	ハードウェアPCブレーク設定	ICD Mini/SIM
thbreak [*addr*]	テンポラリハードウェアPCブレーク設定	ICD Mini/SIM
delete [*breakNo.*]	ブレーク番号によるブレークの解除	ICD Mini/SIM
clear *addr*	ブレーク位置によるブレークの解除	ICD Mini/SIM
enable [*breakNo.*]	ブレークポイントの有効化	ICD Mini/SIM
disable [*breakNo.*]	ブレークポイントの無効化	ICD Mini/SIM
ignore *breakNo. count*	回数指定付きブレークの無効化	ICD Mini/SIM
info breakpoints	ブレークポイントリストの表示	ICD Mini/SIM
c17 timebrk *timer*	時間経過後ブレークの設定	ICD Mini
commands	ブレーク後に実行するコマンドの設定	ICD Mini/Sim

シンボル情報

info locals	ローカルシンボル表示	ICD Mini/SIM
info var	グローバルシンボル表示	ICD Mini/SIM
print *symbol*[*=value*]	シンボル値の変更	ICD Mini/SIM

ファイル読み込み

file *file*	デバッグ情報の読み込み	ICD Mini/SIM
load [*file*]	プログラムのロード	ICD Mini/SIM
c17 loadmd *mode*	プログラムのロードモード設定	ICD Mini

マップ情報

c17 rpf *file*	マップ情報の設定	ICD Mini/SIM
c17 map	マップ情報の表示	SIM

フラッシュメモリ操作

c17 fls *addr1 addr2 erase write*	フラッシュメモリ設定	ICD Mini
c17 fle *control block1 block2* [*timer*]	フラッシュメモリの消去	ICD Mini
c17 flv *Voltage*	フラッシュメモリの書き込み・消去電圧の設定	ICD Mini
c17 flvs	フラッシュメモリの書き込み・消去電圧設定の解除	ICD Mini

トレース

c17 tm on/off *mode* [*file*]	トレースモードの設定	SIM

シミュレーテッドI/O

c17 stdin 1/2 *break buffer* [*file*]	データ入力シミュレーション	ICD Mini/SIM
c17 stdout 1/2 *break buffer* [*file*]	データ出力シミュレーション	ICD Mini/SIM

フラッシュライタ(S5U1C17001H用)

c17 fwe 0/1/2	プログラム/データ/セキュリティ設定の消去	ICD Mini
c17 fwlp *file erase write* [*comment*]	プログラムのロード	ICD Mini
c17 fwld *file blk1 blk2 par* [*comment*]	データのロード	ICD Mini
c17 fwdc *addr sz blk1 blk2 par* [*comm*]	ターゲットメモリのコピー	ICD Mini
c17 fwd	フラッシュライタ情報の表示	ICD Mini
c17 fwpw *mcu ver pass*	セキュリテイ設定	ICD Mini

デバッグコマンド

プロファイラ・カバレッジ

c17 profilemd	プロファイル・カバレッジモードの設定	SIM
c17 profile	プロファイルウィンドウの表示	SIM
c17 coverage	カバレッジウィンドウの表示	SIM

その他

set output-rad x	変数表示形式の変更	ICD Mini/SIM
c17 log [*file*]	ロギング	ICD Mini/SIM
source *file*	コマンドファイルの実行	ICD Mini/SIM
c17 clockmd *mode*	実行カウンタモード設定	ICD Mini/SIM
c17 clock	実行カウンタ表示	ICD Mini/SIM
target *type*	ターゲットの接続	ICD Mini/SIM
detach	ターゲットの切断	ICD Mini/SIM
pwd	カレントディレクトリの表示	ICD Mini/SIM
cd *directory*	カレントディレクトリの変更	ICD Mini/SIM
c17 firmupdate *file*	ICDファームウェア更新	ICD Mini
c17 ttbr *addr*	TTBR設定	SIM
c17 chgclkmd 0/1	DLCK切替モード	ICD Mini
c17 pwul	Flash セキュリティ・パスワードの解除	ICD Mini
c17 help [*command*/*groupNo.*]	ヘルプ	ICD Mini/SIM
quit	デバッガ終了	ICD Mini/SIM

ステータス/エラーメッセージ

ステータスメッセージ

Breakpoint #, *function* at *file:line*	設定したブレークポイントでブレーク
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
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
Break by key break.	[中断]ボタンによる強制ブレーク (シミュレータモード)
Break by key break. Program received signal SIGINT, Interrupt.	[中断]ボタンによる強制ブレーク (ICDモード)

エラーメッセージ

･･･ gdb : unrecognized option 'option'	起動時オプションに誤りがあります。
A setup of a serial port was not completed.	**gdb**が対応しないICDモードが指定されました。
Address is 24bit over.	指定したアドレスが、24ビットを超えています。S1C17のアドレスは、最大24ビット(0xffffff)です。
Address(0x#) is ext or delayed instruction.	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。
C17 command error, command is not supported in present mode.	指定したコマンドは、現在のモード(ICD Miniまたはシミュレータモード)では対応していません。
C17 command error, command is not supported in present target CPU.	指定したコマンドは、現在のターゲットCPUでは対応していません。
C17 command error, command is not supported in ICDmini hardware Ver xx.	ICDminiハードウェアバージョンxxではサポートしていません。
C17 command error, command is not supported in ICDmini firmware Ver xx.	ICDminiファームウェアバージョンxxではサポートしていません。
C17 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。
C17 command error, invalid command.	コマンドに誤りがあります。
C17 command error, invalid parameter.	指定したコマンドのパラメータに誤りがあります。
C17 command error, no map area.	指定したコマンドの引数のアドレスはパラメータファイルで指定したアドレスの範囲外です
C17 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。
C17 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。
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 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。

ステータス/エラーメッセージ

エラーメッセージ

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 display clock counter. Now Timer break mode is on. Please timer break mode off.	時間経過後ブレーク有効時に実行カウンタの表示はできません。時間経過後ブレークを解除してから再度計測してください。
Cannot display clock counter. Time measurement should use continue or until command.	実行カウンタの値は、`continue`または`until`コマンドを実行した後でないと表示できません。
Cannot load to no map memory. (0x#-0x#)	パラメータファイルで指定したアドレス範囲外へのファイルロードはできません。
Cannot measure clock timer.	プログラムの実行時間が短すぎて計測できません。
Cannot open file(*file*).	ファイルがオープンできません。
Cannot open ICD17 usb driver.	USBドライブのオープンに失敗しました。
Cannot set hard pc break.	指定したアドレスにハードブレークは設定できません。
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. 1)。
Cannot set soft pc break.	指定したアドレスにソフトブレークは設定できません。
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 soft pc break at ROM area.	読み取り専用メモリに書き込みはできません。
Cannot set same 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 timer. Timer Conditions: 1<=Timer<=300000	指定時間が条件を超えているので、時間経過後ブレークを設定できません。
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 system error(#).	ICDとの通信中に切断してしまいました。
Copy end address max(0x#) overflow.	コピー元の終了アドレスが最大値(0xffffff)を超えています。
Copy start address max(0x#) overflow.	コピー元の開始アドレスが最大値(0xffffff)を超えています。
Coverage Window is already opened.	カバレッジ・ウィンドウが既に開いています。
CPU is running.	ターゲットCPUがRUN状態のため、コマンドを受け付けることができません。
Erase entry address max(0x#) overflow.	フラッシュ消去ルーチンのアドレスが最大値(0xffffff)を超えています。
Flash memory end address max(0x#) overflow.	フラッシュメモリの終了アドレスが、最大値(0xffffff)を超えています。
Flash memory start address max(0x#) overflow.	フラッシュメモリの開始アドレスが、最大値(0xffffff)を超えています。
ICD17 is busy(#).	ICD側がBUSY状態です。
Initialization error of ICD17.	ターゲットの初期化に失敗しました。
Invalid ID error(0x#).	**gdb**が未定義通信ID番号を送信しました。(内部エラー)
Invalid format event file (#).	c17 int_load(割り込みイベントコマンド)で指定したイベントファイルのフォーマットエラーがあります。
Invalid parameter file(#: *file*).	パラメータファイルに誤りがあります。
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。

エラーメッセージ

It is not c17 architecture ELF file.	`file`コマンドで指定したファイルはS5U1C17001Cが対応しているelf形式ファイルではありません。
Load end address max(0x#) overflow.	フラッシュに書き込むプログラムの終了アドレスが最大値(0xffffff)を超えています。
Load motorola file format error.(*file*)	指定したモトローラ形式ファイルにフォーマットエラーがあります。
Load size limit(0x#) overflow.	指定したファイルサイズが最大値を超えています。 • フラッシュメモリ書き込み/消去プログラムの場合 8Kバイト - 1バイト(0x1fff) • フラッシュメモリへ書き込むデータの場合 3Mバイト - 1バイト(0x2fffff) • ファームウェアの場合 8Mバイト - 1バイト(0x7fffff)
Load start address max(0x#) overflow.	フラッシュに書き込むプログラムの開始アドレスが最大値(0xffffff)を超えています。
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。
Script file error: IF nest max(5) over.	スクリプトファイル(*.Spt)でif文のネスト数が５つを超えている。
Script file format error. (Line no.#)	スクリプトファイル(*.Spt)の#行がフォーマットエラーがあります。
Send Size entry address is out of limit (# - #).	ICDに対して、1パケット辺りに送信するデータサイズが有効範囲内でない。
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。
Specified voltage cannot be output.	指定した電圧値が範囲外です。
Specified voltage is out of range (6.0V ― 8.0V).	指定した電圧値が範囲外です。
Target down.	ICDとターゲット間で通信エラーが発生しました。
There is no argument given to this command.	ターゲットの切断に失敗しました。
Too much event(#).	`c17 int_load`(イベントファイル読み込み)コマンドで指定したイベント数が最大値(256)を超えています。
Transmitting failure(#).	ICDとの通信でICDからNAKを受け取りました。
USB communication error(host->ICD17).	ICDへのUSB送信に失敗しました。
USB communication error(ICD17->host).	ICDからのUSB受信に失敗しました。
Write entry address max(0x#) overflow.	フラッシュ書き込みルーチンのアドレスが最大値(0xffffff)を超えています。
Target CPU does not support flash security.	ターゲットCPUはFlash セキュリティに対応していません。
Cannot unlock flash security password. Flash security version is not available.	Flash セキュリティバージョンが無効なため、パスワードの解除ができません。
Cannot unlock flash security password. Password format is not available.	パスワードの書式が無効なため、パスワードの解除ができません。
Target CPU does not support specified flash security version.	未定義のターゲットCPU、または未定義のフラッシュセキュリティバージョンを指定されました。
Password format is not available.	パスワードの書式が無効です。

浮動小数点演算関数

Double型演算

__adddf3	加算	x ← a + b
__subdf3	減算	x ← a - b
__muldf3	乗算	x ← a * b
__divdf3	除算	x ← a / b
__negdf2	符号反転	x ← -a

Float型演算

__addsf3	加算	x ← a + b
__subsf3	減算	x ← a - b
__mulsf3	乗算	x ← a * b
__divsf3	除算	x ← a / b
__negsf2	符号反転	x ← -a

型変換

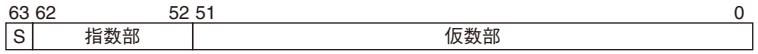
__fixunsdfsi	double → unsigned long	x ← a
__fixdfsi	double → long	x ← a
__floatsidf	long → double	x ← a
__fixunssfsi	float → unsigned long	x ← a
__fixsfsi	float → long	x ← a
__floatsisf	long → float	x ← a
__truncdfsf2	double → float	x ← a
__extendsfdf2	float → double	x ← a

比較

__**df2	double type	Changes %psr and x by a - b **=eq, ne, gt, ge, lt, le
__**sf2	float type	Changes %psr and x by a - %13 **=eq, ne, gt, ge, lt, le

浮動小数点データフォーマット

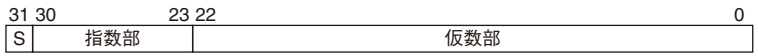
Double型データ



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 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

Float型データ



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 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

整数演算関数

整数演算

__divsi3	符号付き32ビット整数除算	x ← a / b
__modsi3	符号付き32ビット整数剰余演算	x ← a % b
__udivsi3	符号なし32ビット整数除算	x ← a / b
__umodsi3	符号なし32ビット整数剰余演算	x ← a % b
__mulsi3	32ビット乗算	x ← a * b
__divhi3	符号付き16ビット整数除算	x ← a / b
__modhi3	符号付き16ビット整数剰余演算	x ← a % b
__udivhi3	符号なし16ビット整数除算	x ← a / b
__umodhi3	符号なし16ビット整数剰余演算	x ← a % b
__mulhi3	16ビット乗算	x ← a * b

整数シフト

__ashlsi3	32ビット算術左シフト	x ← a << bビット
__ashrsi3	32ビット算術右シフト	x ← a >> bビット
__lshrsi3	32ビット論理右シフト	x ← a >> bビット
__ashlhi3	16ビット算術左シフト	x ← a << bビット
__ashrhi3	16ビット算術右シフト	x ← a >> bビット
__lshrhi3	16ビット論理右シフト	x ← a >> bビット

整数比較

__cmpsi2	比較(long)	x ← 2 | 1 | 0
__ucmpsi2	比較(unsigned long)	x ← 2 | 1 | 0

long long型演算関数

long long型演算

__muldi3	符号付き64ビット乗算	x ← a * b
__divdi3	符号付き64ビット除算	x ← a / b
__udivdi3	符号なし64ビット除算	x ← a / b
__moddi3	符号付き64ビット剰余演算	x ← a % b
__umoddi3	符号なし64ビット剰余演算	x ← a % b
__negdi2	符号反転	x ← -a

long long型シフト

__lshrdi3	64ビット論理右シフト	x ← a >> bビット
__ashldi3	64ビット算術左シフト	x ← a << bビット
__ashrdi3	64ビット算術右シフト	x ← a >> bビット

型変換

__fixunsdfdi	double → unsigned long long	x ← a
__fixdfdi	double → long long	x ← a
__floatdidf	long long → double	x ← a
__fixunssfdi	float → unsigned long long	x ← a
__fixsfdi	float → long long	x ← a
__floatdisf	long long → float	x ← a

long long型比較

__cmpdi2	比較(long long)	x ← 2 | 1 | 0
__ucmpdi2	比較(unsigned long long)	x ← 2 | 1 | 0

入出力関数(ヘッダファイル: stdio.h)

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
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)

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
exit( )	void exit(int status);	
abort( )	void abort(void);	
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

日付と時刻関数(ヘッダファイル: time.h)

gmtime( )	struct tm *gmtime(const time_t *t);	
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)

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。

数学関数(ヘッダファイル: math.h, errno.h, float.h, limits.h)

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)

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)

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)

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 *str, 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);

∗1 グローバル変数の宣言と初期化

FILE _iob[FOPEN_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)

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~R7)、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~R7)、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~R7)、またはレジスタの内容
%sp, sp:	スタックポインタ(SP)またはその内容
%pc, pc:	プログラムカウンタ(PC)またはその内容

メモリ/アドレス/メモリデータ

[%rb], [%sp]:	レジスタ間接アドレッシング指定
[%rb]+, [%sp]+:	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
[%rb]-, [%sp]-:	ポストデクリメント付きレジスタ間接アドレッシング指定
-[%rb], -[%sp]:	プリデクリメント付きレジスタ間接アドレッシング指定
[%sp+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
[imm7]:	即値によるメモリアドレス指定
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
W[XXX]:	XXXで指定される16ビット境界アドレス、またはそのアドレスにストアされているワードデータ
A[XXX]:	XXXで指定される32ビット境界アドレス、またはそのアドレスにストアされている24ビットデータもしくは32ビットデータ

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

フラグ

IL:	割り込みレベル
IE:	割り込みイネーブルフラグ
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)、リセット(0)、または変更なし
1:	セット(1)
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S1C17命令セットの基本命令の他に、拡張命令(sで始まる命令および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。

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
符号付き8ビットデータ転送	ld.b	%rd, %rs	rd(7:0)←rs(7:0), rd(15:8)←rs(7), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]+	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0, rb(23:0)←rb(23:0)+1	–	–	–	–	–	–	○
		%rd, [%rb]-	rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0, rb(23:0)←rb(23:0)-1	–	–	–	–	–	–	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-1, rd(7:0)←B[rb], rd(15:8)←B[rb](7), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%sp+imm7]	rd(7:0)←B[sp+imm7], rd(15:8)←B[sp+imm7](7), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [imm7]	rd(7:0)←B[imm7], rd(15:8)←B[imm7](7), rd(23:16)←0	–	–	–	–	–	–	○
		[%rb], %rs	B[rb]←rs(7:0)	–	–	–	–	–	–	○
		[%rb]+, %rs	B[rb]←rs(7:0), rb(23:0)←rb(23:0)+1	–	–	–	–	–	–	○
		[%rb]-, %rs	B[rb]←rs(7:0), rb(23:0)←rb(23:0)-1	–	–	–	–	–	–	○
		-[%rb], %rs	rb(23:0)←rb(23:0)-1, B[rb]←rs(7:0)	–	–	–	–	–	–	○
		[%sp+imm7], %rs	B[sp+imm7]←rs(7:0)	–	–	–	–	–	–	○
		[imm7], %rs	B[imm7]←rs(7:0)	–	–	–	–	–	–	○
	sld.b	%rd, [%sp+imm20]	%rd←B[%sp+imm20](符号拡張)	–	–	–	–	–	–	–
		%rd, [imm20]	%rd←B[imm20](符号拡張)	–	–	–	–	–	–	–
		[%sp+imm20], %rs	B[%sp+imm20]←%rs(7:0)	–	–	–	–	–	–	–
		[imm20], %rs	B[imm20]←%rs(7:0)	–	–	–	–	–	–	–
	xld.b	%rd, [%sp+imm24]	%rd←B[%sp+imm24](符号拡張)	–	–	–	–	–	–	–
		%rd, [imm24]	%rd←B[imm24](符号拡張)	–	–	–	–	–	–	–
		[%sp+imm24], %rs	B[%sp+imm24]←%rs(7:0)	–	–	–	–	–	–	–
		[imm24], %rs	B[imm24]←%rs(7:0)	–	–	–	–	–	–	–
符号なし8ビットデータ転送	ld.ub	%rd, %rs	rd(7:0)←rs(7:0), rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]+	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0, rb(23:0)←rb(23:0)+1	–	–	–	–	–	–	○
		%rd, [%rb]-	rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0, rb(23:0)←rb(23:0)-1	–	–	–	–	–	–	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-1, rd(7:0)←B[rb], rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%sp+imm7]	rd(7:0)←B[sp+imm7], rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [imm7]	rd(7:0)←B[imm7], rd(15:8)←0, rd(23:16)←0	–	–	–	–	–	–	○
	sld.ub	%rd, [%sp+imm20]	%rd←B[%sp+imm20](ゼロ拡張)	–	–	–	–	–	–	–
		%rd, [imm20]	%rd←B[imm20](ゼロ拡張)	–	–	–	–	–	–	–
	xld.ub	%rd, [%sp+imm24]	%rd←B[%sp+imm24](ゼロ拡張)	–	–	–	–	–	–	–
		%rd, [imm24]	%rd←B[imm24](ゼロ拡張)	–	–	–	–	–	–	–

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
16ビット データ転送	ld	%rd, %rs	rd(15:0)←rs(15:0), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, sign7	rd(6:0)←sign7(6:0), rd(15:7)←sign7(6), rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(15:0)←W[rb], rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%rb]+	rd(15:0)←W[rb], rd(23:16)←0, rb(23:0)←rb(23:0)+2	–	–	–	–	–	–	○
		%rd, [%rb]-	rd(15:0)←W[rb], rd(23:16)←0, rb(23:0)←rb(23:0)-2	–	–	–	–	–	–	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-2, rd(15:0)←W[rb], rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [%sp+imm7]	rd(15:0)←W[sp+imm7], rd(23:16)←0	–	–	–	–	–	–	○
		%rd, [imm7]	rd(15:0)←W[imm7], rd(23:16)←0	–	–	–	–	–	–	○
		[%rb], %rs	W[rb]←rs(15:0)	–	–	–	–	–	–	○
		[%rb]+, %rs	W[rb]←rs(15:0), rb(23:0)←rb(23:0)+2	–	–	–	–	–	–	○
		[%rb]-, %rs	W[rb]←rs(15:0), rb(23:0)←rb(23:0)-2	–	–	–	–	–	–	○
		-[%rb], %rs	rb(23:0)←rb(23:0)-2, W[rb]←rs(15:0)	–	–	–	–	–	–	○
		[%sp+imm7], %rs	W[sp+imm7]←rs(15:0)	–	–	–	–	–	–	○
		[imm7], %rs	W[imm7]←rs(15:0)	–	–	–	–	–	–	○
	sld	%rd, imm16	%rd←imm16	–	–	–	–	–	–	–
		%rd, symbol±imm16	%rd←symbol±imm16(15:0)	–	–	–	–	–	–	–
		%rd, [%sp+imm20]	%rd←W[%sp+imm20]	–	–	–	–	–	–	–
		%rd, [imm20]	%rd←W[imm20]	–	–	–	–	–	–	–
		[%sp+imm20], %rs	W[%sp+imm20]←%rs(15:0)	–	–	–	–	–	–	–
		[imm20], %rs	W[imm20]←%rs(15:0)	–	–	–	–	–	–	–
	xld	%rd, imm16	%rd←imm16	–	–	–	–	–	–	–
		%rd, symbol±imm16	%rd←symbol±imm16(15:0)	–	–	–	–	–	–	–
		%rd, [%sp+imm24]	%rd←W[%sp+imm24]	–	–	–	–	–	–	–
		%rd, [imm24]	%rd←W[imm24]	–	–	–	–	–	–	–
		[%sp+imm24], %rs	W[%sp+imm24]←%rs(15:0)	–	–	–	–	–	–	–
		[imm24], %rs	W[imm24]←%rs(15:0)	–	–	–	–	–	–	–
32ビット データ転送	ld.a	%rd, %rs	rd(23:0)←rs(23:0)	–	–	–	–	–	–	○
		%rd, imm7	rd(6:0)←imm7(6:0), rd(23:7)←0	–	–	–	–	–	–	○
		%rd, [%rb]	rd(23:0)←A[rb](23:0), 無視←A[rb](31:24)	–	–	–	–	–	–	○
		%rd, [%rb]+	rd(23:0)←A[rb](23:0), 無視←A[rb](31:24), rb(23:0)←rb(23:0)+4	–	–	–	–	–	–	○
		%rd, [%rb]-	rd(23:0)←A[rb](23:0), 無視←A[rb](31:24), rb(23:0)←rb(23:0)-4	–	–	–	–	–	–	○
		%rd, -[%rb]	rb(23:0)←rb(23:0)-4, rd(23:0)←A[rb](23:0), 無視←A[rb](31:24)	–	–	–	–	–	–	○
		%rd, [%sp+imm7]	rd(23:0)←A[sp+imm7](23:0), 無視←A[sp+imm7](31:24)	–	–	–	–	–	–	○
		%rd, [imm7]	rd(23:0)←A[imm7](23:0), 無視←A[imm7](31:24)	–	–	–	–	–	–	○

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
32ビット データ転送	ld.a	[%rb], %rs	A[rb](23:0)←rs(23:0), A[rb](31:24)←0	–	–	–	–	–	–	○
		[%rb]+, %rs	A[rb](23:0)←rs(23:0), A[rb](31:24)←0, rb(23:0)←rb(23:0)+4	–	–	–	–	–	–	○
		[%rb]-, %rs	A[rb](23:0)←rs(23:0), A[rb](31:24)←0, rb(23:0)←rb(23:0)-4	–	–	–	–	–	–	○
		-[%rb], %rs	rb(23:0)←rb(23:0)-4, A[rb](23:0)←rs(23:0), A[rb](31:24)←0	–	–	–	–	–	–	○
		[%sp+imm7], %rs	A[sp+imm7](23:0)←rs(23:0), A[sp+imm7](31:24)←0	–	–	–	–	–	–	○
		[imm7], %rs	A[imm7](23:0)←rs(23:0), A[imm7](31:24)←0	–	–	–	–	–	–	○
		%rd, %sp	rd(23:2)←sp(23:2), rd(1:0)←0	–	–	–	–	–	–	○
		%rd, %pc	rd(23:0)←pc(23:0)+2	–	–	–	–	–	–	○
		%rd, [%sp]	rd(23:0)←A[sp](23:0), 無視←A[sp](31:24)	–	–	–	–	–	–	○
		%rd, [%sp]+	rd(23:0)←A[sp](23:0), 無視←A[sp](31:24), sp(23:0)←sp(23:0)+4	–	–	–	–	–	–	○
		%rd, [%sp]-	rd(23:0)←A[sp](23:0), 無視←A[sp](31:24), sp(23:0)←sp(23:0)-4	–	–	–	–	–	–	○
		%rd, -[%sp]	sp(23:0)←sp(23:0)-4, rd(23:0)←A[sp](23:0), 無視←A[sp](31:24)	–	–	–	–	–	–	○
		[%sp], %rs	A[sp](23:0)←rs(23:0), A[sp](31:24)←0	–	–	–	–	–	–	○
		[%sp]+, %rs	A[sp](23:0)←rs(23:0), A[sp](31:24)←0, sp(23:0)←sp(23:0)+4	–	–	–	–	–	–	○
		[%sp]-, %rs	A[sp](23:0)←rs(23:0), A[sp](31:24)←0, sp(23:0)←sp(23:0)-4	–	–	–	–	–	–	○
		-[%sp], %rs	sp(23:0)←sp(23:0)-4, A[sp](23:0)←rs(23:0), A[sp](31:24)←0	–	–	–	–	–	–	○
		%sp, %rs	sp(23:2)←rs(23:2)	–	–	–	–	–	–	○
		%sp, imm7	sp(6:2)←imm7(6:2), sp(23:7)←0	–	–	–	–	–	–	○
	sld.a	%rd, imm20	%rd←imm20	–	–	–	–	–	–	–
		%sp, imm20	%sp←imm20	–	–	–	–	–	–	–
		%rd, symbol±imm20	%rd←symbol±imm20(19:0)	–	–	–	–	–	–	–
		%sp, symbol±imm20	%sp←symbol±imm20(19:0)	–	–	–	–	–	–	–
		%rd, [%sp+imm20]	%rd←A[%sp+imm20](23:0), 無視←A[%sp+imm20](31:24)	–	–	–	–	–	–	–
		%rd, [imm20]	%rd←A[imm20](23:0), 無視←A[imm20](31:24)	–	–	–	–	–	–	–
		[%sp+imm20], %rs	A[%sp+imm20](23:0)←%rs(23:0), A[%sp+imm20](31:24)←0	–	–	–	–	–	–	–
		[imm20], %rs	A[imm20](23:0)←%rs(23:0), A[imm20](31:24)←0	–	–	–	–	–	–	–
	xld.a	%rd, imm24	%rd←imm24	–	–	–	–	–	–	–
		%sp, imm24	%sp←imm24	–	–	–	–	–	–	–
		%rd, symbol±imm24	%rd←symbol±imm24(23:0)	–	–	–	–	–	–	–
		%sp, symbol±imm24	%sp←symbol±imm24(23:0)	–	–	–	–	–	–	–
		%rd, [%sp+imm24]	%rd←A[%sp+imm24](23:0), 無視←A[%sp+imm24](31:24)	–	–	–	–	–	–	–
		%rd, [imm24]	%rd←A[imm24](23:0), 無視←A[imm24](31:24)	–	–	–	–	–	–	–
		[%sp+imm24], %rs	A[%sp+imm24](23:0)←%rs(23:0), A[imm24](31:24)←0	–	–	–	–	–	–	–
		[imm24], %rs	A[imm24](23:0)←%rs(23:0), A[%sp+imm24](31:24)←0	–	–	–	–	–	–	–

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
算術演算	add	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0	–	–	↔	↔	↔	↔	○
	add/c	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	add/nc	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	add	*%rd, imm7*	rd(15:0)←rd(15:0)+imm7(ゼロ拡張), rd(23:16)←0	–	–	↔	↔	↔	↔	○
	sadd	%rd, imm16	rd(15:0)←rd(15:0)+imm16, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	xadd	%rd, imm16	rd(15:0)←rd(15:0)+imm16, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	add.a	%rd, %rs	rd(23:0)←rd(23:0)+rs(23:0)	–	–	–	–	–	–	○
	add.a/c	%rd, %rs	rd(23:0)←rd(23:0)+rs(23:0) if C = 1 (nop if C = 0)	–	–	–	–	–	–	○
	add.a/nc	%rd, %rs	rd(23:0)←rd(23:0)+rs(23:0) if C = 0 (nop if C = 1)	–	–	–	–	–	–	○
	add.a	%sp, %rs	sp(23:0)←sp(23:0)+rs(23:0)	–	–	–	–	–	–	○
		%rd, imm7	rd(23:0)←rd(23:0)+imm7(ゼロ拡張)	–	–	–	–	–	–	○
		%sp, imm7	sp(23:0)←sp(23:0)+imm7(ゼロ拡張)	–	–	–	–	–	–	○
	sadd.a	%rd, imm20	rd(23:0)←rd(23:0)+imm20(ゼロ拡張)	–	–	–	–	–	–	–
		%sp, imm20	sp(23:0)←sp(23:0)+imm20(ゼロ拡張)	–	–	–	–	–	–	–
	xadd.a	%rd, imm24	rd(23:0)←rd(23:0)+imm24	–	–	–	–	–	–	–
		%sp, imm24	sp(23:0)←sp(23:0)+imm24	–	–	–	–	–	–	–
	adc	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0	–	–	↔	↔	↔	↔	○
	adc/c	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	adc/nc	%rd, %rs	rd(15:0)←rd(15:0)+rs(15:0)+C, rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	adc	*%rd, imm7*	rd(15:0)←rd(15:0)+imm7(ゼロ拡張)+C, rd(23:16)←0	–	–	↔	↔	↔	↔	○
	sadc	%rd, imm16	rd(15:0)←rd(15:0)+imm16+C, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	xadc	%rd, imm16	rd(15:0)←rd(15:0)+imm16+C, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	sub	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0	–	–	↔	↔	↔	↔	○
	sub/c	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	sub/nc	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	sub	*%rd, imm7*	rd(15:0)←rd(15:0)-imm7(ゼロ拡張), rd(23:16)←0	–	–	↔	↔	↔	↔	○
	ssub	%rd, imm16	rd(15:0)←rd(15:0)-imm16, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	xsub	%rd, imm16	rd(15:0)←rd(15:0)-imm16, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	sub.a	%rd, %rs	rd(23:0)←rd(23:0)-rs(23:0)	–	–	–	–	–	–	○
	sub.a/c	%rd, %rs	rd(23:0)←rd(23:0)-rs(23:0) if C = 1 (nop if C = 0)	–	–	–	–	–	–	○
	sub.a/nc	%rd, %rs	rd(23:0)←rd(23:0)-rs(23:0) if C = 0 (nop if C = 1)	–	–	–	–	–	–	○
	sub.a	%sp, %rs	sp(23:0)←sp(23:0)-rs(23:0)	–	–	–	–	–	–	○
		%rd, imm7	rd(23:0)←rd(23:0)-imm7(ゼロ拡張)	–	–	–	–	–	–	○
		%sp, imm7	sp(23:0)←sp(23:0)-imm7(ゼロ拡張)	–	–	–	–	–	–	○

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
算術演算	ssub.a	%rd, imm20	rd(23:0)←rd(23:0)-imm20(ゼロ拡張)	–	–	–	–	–	–	–
		%sp, imm20	sp(23:0)←sp(23:0)-imm20(ゼロ拡張)	–	–	–	–	–	–	–
	xsub.a	%rd, imm24	rd(23:0)←rd(23:0)-imm24	–	–	–	–	–	–	–
		%sp, imm24	sp(23:0)←sp(23:0)-imm24	–	–	–	–	–	–	–
	sbc	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0	–	–	↔	↔	↔	↔	○
	sbc/c	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	sbc/nc	%rd, %rs	rd(15:0)←rd(15:0)-rs(15:0)-C, rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	sbc	*%rd, imm7*	rd(15:0)←rd(15:0)-imm7(ゼロ拡張)-C, rd(23:16)←0	–	–	↔	↔	↔	↔	○
	ssbc	%rd, imm16	rd(15:0)←rd(15:0)-imm16-C, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	xsbc	%rd, imm16	rd(15:0)←rd(15:0)-imm16-C, rd(23:16)←0	–	–	↔	↔	↔	↔	–
	cmp	%rd, %rs	rd(15:0)-rs(15:0)	–	–	↔	↔	↔	↔	○
	cmp/c	%rd, %rs	rd(15:0)-rs(15:0) if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	cmp/nc	%rd, %rs	rd(15:0)-rs(15:0) if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	cmp	*%rd, sign7*	rd(15:0)-sign7(符号拡張)	–	–	↔	↔	↔	↔	○
	scmp	%rd, imm16	rd(15:0)-imm16	–	–	↔	↔	↔	↔	–
	xcmp	%rd, imm16	rd(15:0)-imm16	–	–	↔	↔	↔	↔	–
	cmp.a	%rd, %rs	d(23:0)-rs(23:0)	–	–	↔	–	↔	–	○
	cmp.a/c	%rd, %rs	rd(23:0)-rs(23:0) if C = 1 (nop if C = 0)	–	–	–	–	↔	–	○
	cmp.a/nc	%rd, %rs	rd(23:0)-rs(23:0) if C = 0 (nop if C = 1)	–	–	–	–	↔	–	○
	cmp.a	*%rd, imm7*	rd(23:0)-imm7(ゼロ拡張)	–	–	↔	–	↔	–	○
	scmp.a	%rd, imm20	rd(23:0)-imm20(ゼロ拡張)	–	–	↔	–	↔	–	–
	xcmp.a	%rd, imm24	rd(23:0)-imm24	–	–	↔	–	↔	–	–
	cmc	%rd, %rs	rd(15:0)-rs(15:0)-C	–	–	↔	↔	↔	↔	○
	cmc/c	%rd, %rs	rd(15:0)-rs(15:0)-C if C = 1 (nop if C = 0)	–	–	–	↔	↔	↔	○
	cmc/nc	%rd, %rs	rd(15:0)-rs(15:0)-C if C = 0 (nop if C = 1)	–	–	–	↔	↔	↔	○
	cmc	*%rd, sign7*	rd(15:0)-sign7(符号拡張)-C	–	–	↔	↔	↔	↔	○
	scmc	%rd, imm16	rd(15:0)-imm16-C	–	–	↔	↔	↔	↔	–
	xcmc	%rd, imm16	rd(15:0)-imm16-C	–	–	↔	↔	↔	↔	–
論理演算	and	%rd, %rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0	–	–	–	0	↔	↔	○
	and/c	%rd, %rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	0	↔	↔	○
	and/nc	%rd, %rs	rd(15:0)←rd(15:0)&rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	0	↔	↔	○
	and	*%rd, sign7*	rd(15:0)←rd(15:0)&sign7(符号拡張), rd(23:16)←0	–	–	–	0	↔	↔	○
	sand	%rd, imm16	rd(15:0)←rd(15:0)&imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xand	%rd, imm16	rd(15:0)←rd(15:0)&imm16, rd(23:16)←0	–	–	–	0	↔	↔	–

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
論理演算	or	%rd, %rs	d(15:0)←rd(15:0) ∣ rs(15:0), rd(23:16)←0	–	–	–	0	↔	↔	○
	or/c	%rd, %rs	rd(15:0)←rd(15:0) ∣ rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	0	↔	↔	○
	or/nc	%rd, %rs	rd(15:0)←rd(15:0) ∣ rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	0	↔	↔	○
	or	*%rd, sign7*	rd(15:0)←rd(15:0) ∣ sign7(符号拡張), rd(23:16)←0	–	–	–	0	↔	↔	○
	soor	%rd, imm16	rd(15:0)←rd(15:0) ∣ imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xoor	%rd, imm16	rd(15:0)←rd(15:0) ∣ imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xor	%rd, %rs	rd(15:0)←rd(15:0)^rs(15:0), rd(23:16)←0	–	–	–	0	↔	↔	○
	xor/c	%rd, %rs	rd(15:0)←rd(15:0)^rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	0	↔	↔	○
	xor/nc	%rd, %rs	rd(15:0)←rd(15:0)^rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	0	↔	↔	○
	xor	*%rd, sign7*	rd(15:0)←rd(15:0)^sign7(符号拡張), rd(23:16)←0	–	–	–	0	↔	↔	○
	sxor	%rd, imm16	rd(15:0)←rd(15:0)^imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xxor	%rd, imm16	rd(15:0)←rd(15:0)^imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	not	%rd, %rs	rd(15:0)←!rs(15:0), rd(23:16)←0	–	–	–	0	↔	↔	○
	not/c	%rd, %rs	rd(15:0)←!rs(15:0), rd(23:16)←0 if C = 1 (nop if C = 0)	–	–	–	0	↔	↔	○
	not/nc	%rd, %rs	rd(15:0)←!rs(15:0), rd(23:16)←0 if C = 0 (nop if C = 1)	–	–	–	0	↔	↔	○
	not	*%rd, sign7*	rd(15:0)←!sign7(符号拡張), rd(23:16)←0	–	–	–	0	↔	↔	○
	snot	%rd, imm16	rd(15:0)←!imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
	xnot	%rd, imm16	rd(15:0)←!imm16, rd(23:16)←0	–	–	–	0	↔	↔	–
分岐	jpr / jpr.d	%rb	pc←pc+2+rb	–	–	–	–	–	–	–
		sign10	pc←pc+2+sign11; sign11={sign10,0}	–	–	–	–	–	–	–
	sjpr / sjpr.d	label±imm20	pc←label±imm20	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20	–	–	–	–	–	–	–
	xjpr / xjpr.d	label±imm24	pc←label±imm24	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24	–	–	–	–	–	–	–
	jpa / jpa.d	%rb	pc←rb	–	–	–	–	–	–	–
		imm7	pc←imm7	–	–	–	–	–	–	–
	sjpa / sjpa.d	label±imm20	pc←label±imm20	–	–	–	–	–	–	–
		imm20	pc←imm20	–	–	–	–	–	–	–
	xjpa / xjpa.d	label±imm24	pc←label±imm24	–	–	–	–	–	–	–
		imm24	pc←imm24	–	–	–	–	–	–	–
	jrgt / jrgt.d	*sign7*	pc←pc+2+sign8 if !Z&!(N^V) is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrgt / sjrgt.d	label±imm20	pc←label±imm20 if !Z&!(N^V) is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !Z&!(N^V) is true	–	–	–	–	–	–	–
	xjrgt / xjrgt.d	label±imm24	pc←label±imm24 if !Z&!(N^V) is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !Z&!(N^V) is true	–	–	–	–	–	–	–

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
分岐	*jrge / jrge.d*	*sign7*	pc←pc+2+sign8 if !(N^V) is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrge / sjrge.d	label±imm20	pc←label±imm20 if !(N^V) is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !(N^V) is true	–	–	–	–	–	–	–
	xjrge / xjrge.d	label±imm24	pc←label±imm24 if !(N^V) is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !(N^V) is true	–	–	–	–	–	–	–
	jrlt / jrlt.d	*sign7*	pc←pc+2+sign8 if N^V is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrlt / sjrlt.d	label±imm20	pc←label±imm20 if N^V is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if N^V is true	–	–	–	–	–	–	–
	xjrlt / xjrlt.d	label±imm24	pc←label±imm24 if N^V is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if N^V is true	–	–	–	–	–	–	–
	jrle / jrle.d	*sign7*	pc←pc+2+sign8 if Z	(N^V) is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrle / sjrle.d	label±imm20	pc←label±imm20 if Z	(N^V) is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if Z	(N^V) is true	–	–	–	–	–	–	–
	xjrle / xjrle.d	label±imm24	pc←label±imm24 if Z	(N^V) is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if Z	(N^V) is true	–	–	–	–	–	–	–
	jrugt / jrugt.d	*sign7*	pc←pc+2+sign8 if !Z&!C is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrugt / sjrugt.d	label±imm20	pc←label±imm20 if !Z&!C is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !Z&!C is true	–	–	–	–	–	–	–
	xjrugt / xjrugt.d	label±imm24	pc←label±imm24 if !Z&!C is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !Z&!C is true	–	–	–	–	–	–	–
	jruge / jruge.d	*sign7*	pc←pc+2+sign8 if !C is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjruge / sjruge.d	label±imm20	pc←label±imm20 if !C is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !C is true	–	–	–	–	–	–	–
	xjruge / xjruge.d	label±imm24	pc←label±imm24 if !C is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !C is true	–	–	–	–	–	–	–
	jrult / jrult.d	*sign7*	pc←pc+2+sign8 if C is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrult / sjrult.d	label±imm20	pc←label±imm20 if C is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if C is true	–	–	–	–	–	–	–
	xjrult / xjrult.d	label±imm24	pc←label±imm24 if C is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if C is true	–	–	–	–	–	–	–
	jrule / jrule.d	*sign7*	pc←pc+2+sign8 if Z	C is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrule / sjrule.d	label±imm20	pc←label±imm20 if Z	C is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if Z	C is true	–	–	–	–	–	–	–
	xjrule / xjrule.d	label±imm24	pc←label±imm24 if Z	C is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if Z	C is true	–	–	–	–	–	–	–

備　考

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
分岐	*jreq / jreq.d*	*sign7*	pc←pc+2+sign8 if Z is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjreq / sjreq.d	label±imm20	pc←label±imm20 if Z is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if Z is true	–	–	–	–	–	–	–
	xjreq / xjreq.d	label±imm24	pc←label±imm24 if Z is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if Z is true	–	–	–	–	–	–	–
	jrne / jrne.d	*sign7*	pc←pc+2+sign8 if !Z is true; sign8={sign7,0}	–	–	–	–	–	–	–
	sjrne / sjrne.d	label±imm20	pc←label±imm20 if !Z is true	–	–	–	–	–	–	–
		sign20	pc←pc+2+sign20 if !Z is true	–	–	–	–	–	–	–
	xjrne / xjrne.d	label±imm24	pc←label±imm24 if !Z is true	–	–	–	–	–	–	–
		sign24	pc←pc+2+sign24 if !Z is true	–	–	–	–	–	–	–
	call / call.d	%rb	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+rb	–	–	–	–	–	–	–
		sign10	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+sign11; sign11={sign10,0}	–	–	–	–	–	–	–
	scall / scall.d	label±imm20	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←label±imm20	–	–	–	–	–	–	–
		sign20	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+sign20	–	–	–	–	–	–	–
	xcall / xcall.d	label±imm24	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←label±imm24	–	–	–	–	–	–	–
		sign24	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←pc+2+sign24	–	–	–	–	–	–	–
	calla / calla.d	%rb	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←rb	–	–	–	–	–	–	–
		imm7	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←imm7	–	–	–	–	–	–	–
	scalla / scalla.d	label±imm20	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←label±imm20	–	–	–	–	–	–	–
		imm20	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←imm20	–	–	–	–	–	–	–
	xcalla / xcalla.d	label±imm24	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←label±imm24	–	–	–	–	–	–	–
		imm24	sp←sp-4, A[sp]←pc+2(d=0)/4(d=1), pc←imm24	–	–	–	–	–	–	–
	ret / ret.d		pc←A[sp](23:0), sp←sp+4	–	–	–	–	–	–	–
	int	imm5	sp←sp-4, A[sp]←{psr, pc+2}, pc←ベクタ(TTBR+imm5×4)	–	0	–	–	–	–	–
	intl	imm5, imm3	sp←sp-4, A[sp]←{psr, pc+2}, pc←ベクタ(TTBR+imm5×4), psr(IL)←imm3	↔	0	–	–	–	–	–
	reti / reti.d		{psr, pc}←A[sp], sp←sp+4	↔	↔	↔	↔	↔	↔	–
	brk		A[DBRAM]←{psr, pc+2}, A[DBRAM+4]←r0, pc←0xfffc00	–	0	–	–	–	–	–
	retd		r0←A[DBRAM+4](23:0), {psr, pc}←A[DBRAM]	↔	↔	↔	↔	↔	↔	–
シフト&スワップ	sr	%rd, %rs	右論理シフト: rd(15:0)←rd(15:0)>>rs(15:0), rd(23:16)←0, MSB←0 (*1)	–	–	↔	–	↔	↔	○
		%rd, imm7	右論理シフト: rd(15:0)←rd(15:0)>>imm7, rd(23:16)←0, MSB←0 (*1)	–	–	↔	–	↔	↔	○
	sa	%rd, %rs	右算術シフト: rd(15:0)←rd(15:0)>>rs(15:0), rd(23:16)←0, MSB←符号 (*1)	–	–	↔	–	↔	↔	○
		%rd, imm7	右算術シフト: rd(15:0)←rd(15:0)>>imm7, rd(23:16)←0, MSB←符号 (*1)	–	–	↔	–	↔	↔	○
	sl	%rd, %rs	左論理シフト: rd(15:0)←rd(15:0)<<rs(15:0), rd(23:16)←0, LSB←0 (*1)	–	–	↔	–	↔	↔	○
		%rd, imm7	左論理シフト: rd(15:0)←rd(15:0)<<imm7, rd(23:16)←0, LSB←0 (*1)	–	–	↔	–	↔	↔	○
	swap	%rd, %rs	rd(15:8)←rs(7:0), rd(7:0)←rs(15:8), rd(23:16)←0	–	–	–	–	–	–	○

備　考

*1)　シフト量: rs/imm7 = 0〜3の場合は0〜3ビット, rs/imm7 = 4〜7の場合は4ビット, rs/imm7 = 8以上の場合は8ビット

分類	ニーモニック		機　能	フラグ						D
	オペコード	オペランド		IL	IE	C	V	Z	N	
コンバージョン	cv.ab	%rd, %rs	rd(23:8)←rs(7), rd(7:0)←rs(7:0)	–	–	–	–	–	–	○
	cv.as	%rd, %rs	rd(23:16)←rs(15), rd(15:0)←rs(15:0)	–	–	–	–	–	–	○
	cv.al	%rd, %rs	rd(23:16)←rs(7:0), rd(15:0)←rd(15:0)	–	–	–	–	–	–	○
	cv.la	%rd, %rs	rd(23:8)←0, rd(7:0)←rs(23:16)	–	–	–	–	–	–	○
	cv.ls	%rd, %rs	rd(23:16)←0, rd(15:0)←rs(15)	–	–	–	–	–	–	○
即値拡張	ext	imm13	次の命令の即値またはオペランドを拡張	–	–	–	–	–	–	–
システム制御	nop		ノーオペレーション	–	–	–	–	–	–	○
	halt		HALT	–	–	–	–	–	–	–
	slp		SLEEP	–	–	–	–	–	–	–
	ei		psr(IE)←1	–	1	–	–	–	–	○
	di		psr(IE)←0	–	0	–	–	–	–	○
コプロセッサ	ld.cw	%rd, %rs	co_dout0←rd, co_dout1←rs	–	–	–	–	–	–	○
		%rd, imm7	co_dout0←rd, co_dout1←imm7	–	–	–	–	–	–	○
	sld.cw	%rd, imm20	co_dout0←rd, co_dout1←imm20	–	–	–	–	–	–	–
		%rd, symbol±imm20	co_dout0←rd, co_dout1←symbol±imm20	–	–	–	–	–	–	–
	xld.cw	%rd, imm24	co_dout0←rd, co_dout1←imm24	–	–	–	–	–	–	–
		%rd, symbol±imm24	co_dout0←rd, co_dout1←symbol±imm24	–	–	–	–	–	–	–
	ld.ca	%rd, %rs	co_dout0←rd, co_dout1←rs, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	○
		%rd, imm7	co_dout0←rd, co_dout1←imm7, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	○
	sld.ca	%rd, imm20	co_dout0←rd, co_dout1←imm20, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
		%rd, symbol±imm20	co_dout0←rd, co_dout1←symbol±imm20, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
	xld.ca	%rd, imm24	co_dout0←rd, co_dout1←imm24, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
		%rd, symbol±imm24	co_dout0←rd, co_dout1←symbol±imm24, rd←co_din, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
	ld.cf	%rd, %rs	co_dout0←rd, co_dout1←rs, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	○
		%rd, imm7	co_dout0←rd, co_dout1←imm7, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	○
	sld.cf	%rd, imm20	co_dout0←rd, co_dout1←imm20, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
		%rd, symbol±imm20	co_dout0←rd, co_dout1←symbol±imm20, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
	xld.cf	%rd, imm24	co_dout0←rd, co_dout1←imm24, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–
		%rd, symbol±imm24	co_dout0←rd, co_dout1←symbol±imm24, psr(C, V, Z, N)←co_cvzn	–	–	↔	↔	↔	↔	–

備　考

拡張命令の展開形式 (1)

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
sld.b **sld.ub** **sld** **sld.a**	**%rd, [%sp+imm20]** 例) sld.b %rd, [%sp+imm20]	imm20≦0x7f ld.b %rd, [%sp+imm20(6:0)]	0x7f<imm20 ext imm20(19:7) ld.b %rd, [%sp+imm20(6:0)]	—
	%rd, [imm20] 例) sld %rd, [imm20]	imm20≦0x7f ld %rd, [imm20(6:0)]	0x7f<imm20 ext imm20(19:7) ld %rd, [imm20(6:0)]	—
sld.b **sld** **sld.a**	**[%sp+imm20], %rs** 例) sld.b [%sp+imm20], %rs	imm20≦0x7f ld.b [%sp+imm20(6:0)], %rs	0x7f<imm20 ext imm20(19:7) ld.b [%sp+imm20(6:0)], %rs	—
	[imm20], %rs 例) sld [imm20], %rs	imm20≦0x7f ld [imm20(6:0)], %rs	0x7f<imm20 ext imm20(19:7) ld [imm20(6:0)], %rs	—
sld	**%rd, imm16** 例) sld %rd, imm16	imm16≦0x7f ld %rd, imm16(6:0)	0x7f<imm16 ext imm16(15:7) ld %rd, imm16(6:0)	—
	%rd, symbol±imm16 例) sld %rd, symbol+imm16	無条件 ext (symbol+imm16)(15:7) ld %rd, (symbol+imm16)(6:0)	—	—
sld.a	**%rd, imm20** 例) sld.a %rd, imm20	imm20≦0x7f ld.a %rd, imm20(6:0)	0x7f<imm20 ext imm20(19:7) ld.a %rd, imm20(6:0)	—
	%sp, imm20 例) sld.a %sp, imm20	imm20≦0x7f ld.a %sp, imm20(6:0)	0x7f<imm20 ext imm20(19:7) ld.a %sp, imm20(6:0)	—

備　考

拡張命令の展開形式 (2)

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
sld.a	**%rd, symbol±imm20** 例) sld.a %rd, symbol+imm20	無条件 ext (symbol+imm20)(19:7) ld.a %rd, (symbol+imm20)(6:0)	—	—
	%sp, symbol±imm20 例) sld.a %sp, symbol-imm20	無条件 ext (symbol-imm20)(19:7) ld.a %sp, (symbol-imm20)(6:0)	—	—
xld.b **xld.ub** **xld** **xld.a**	**%rd, [%sp+imm24]** 例) xld.b %rd, [%sp+imm24]	imm24≤0x7f ld.b %rd, [%sp+imm24(6:0)]	0x7f<imm24≤0xfffff ext imm24(19:7) ld.b %rd, [%sp+imm24(6:0)]	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) ld.b %rd, [%sp+imm24(6:0)]
	%rd, [imm24] 例) xld %rd, [imm24]	imm24≤0x7f ld %rd, [imm24(6:0)]	0x7f<imm24≤0xfffff ext imm24(19:7) ld %rd, [imm24(6:0)]	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) ld %rd, [imm24(6:0)]
xld.b **xld** **xld.a**	**[%sp+imm24], %rs** 例) xld.b [%sp+imm24], %rs	imm24≤0x7f ld.b [%sp+imm24(6:0)], %rs	0x7f<imm24≤0xfffff ext imm24(19:7) ld.b [%sp+imm24(6:0)], %rs	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) ld.b [%sp+imm24(6:0)], %rs
	[imm24], %rs 例) xld [imm24], %rs	imm24≤0x7f ld [imm24(6:0)], %rs	0x7f<imm24≤0xfffff ext imm24(19:7) ld [imm24(6:0)], %rs	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) ld [imm24(6:0)], %rs
xld	**%rd, imm16** 例) xld %rd, imm16	imm16≤0x7f ld %rd, imm16(6:0)	0x7f<imm16 ext imm16(15:7) ld %rd, imm16(6:0)	—
	%rd, symbol±imm16 例) xld %rd, symbol+imm16	無条件 ext (symbol+imm16)(15:7) ld %rd, (symbol+imm16)(6:0)	—	—

備　考

拡張命令の展開形式 (3)

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
xld.a	**%rd, imm24** 例) xld.a %rd, imm24	imm24≤0x7f ld.a %rd, imm24(6:0)	0x7f<imm24≤0xfffff ext imm24(19:7) ld.a %rd, imm24(6:0)	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) ld.a %rd, imm24(6:0)
	%sp, imm24 例) xld.a %sp, imm24	imm24≤0x7f ld.a %sp, imm24(6:0)	0x7f<imm24≤0xfffff ext imm24(19:7) ld.a %sp, imm24(6:0)	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) ld.a %sp, imm24(6:0)
	%rd, symbol±imm24 例) xld.a %rd, symbol+imm24	無条件 ext (symbol+imm24)(23:20) ext (symbol+imm24)(19:7) ld.a %rd, (symbol+imm24)(6:0)	— 	—
	%sp, symbol±imm24 例) xld.a %sp, symbol-imm24	無条件 ext (symbol-imm24)(23:20) ext (symbol-imm24)(19:7) ld.a %sp, (symbol-imm24)(6:0)	— 	—
sadd **sadc** **ssub** **ssbc**	**%rd, imm16** 例) sadd %rd, imm16	imm16≤0x7f add %rd, imm16(6:0)	0x7f<imm16 ext imm16(15:7) add %rd, imm16(6:0)	—
sadd.a **ssub.a**	**%rd, imm20** 例) ssub.a %rd, imm20	imm20≤0x7f sub.a %rd, imm20(6:0)	0x7f<imm20 ext imm20(19:7) sub.a %rd, imm20(6:0)	—
	%sp, imm20 例) sadd.a %sp, imm20	imm20≤0x7f add.a %sp, imm20(6:0)	0x7f<imm20 ext imm20(19:7) add.a %sp, imm20(6:0)	—
xadd **xadc** **xsub** **xsbc**	**%rd, imm16** 例) xadc %rd, imm16	imm16≤0x7f adc %rd, imm16(6:0)	0x7f<imm16 ext imm16(15:7) adc %rd, imm16(6:0)	—

備　考

拡張命令の展開形式 (4)

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
xadd.a **xsub.a**	**%rd, imm24** 例) xsub.a %rd, imm24	imm24≤0x7f sub.a %rd, imm24(6:0)	0x7f<imm24≤0xfffff ext imm24(19:7) sub.a %rd, imm24(6:0)	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) sub.a %rd, imm24(6:0)
	%sp, imm24 例) xadd.a %sp, imm24	imm24≤0x7f add.a %sp, imm24(6:0)	0x7f<imm24≤0xfffff ext imm24(19:7) add.a %sp, imm24(6:0)	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) add.a %sp, imm24(6:0)
scmp **scmc**	**%rd, imm16** 例) scmp %rd, imm16	imm16≤0x7f cmp %rd, imm16(6:0)	0x7f<imm16 ext imm16(15:7) cmp %rd, imm16(6:0)	—
scmp.a	**%rd, imm20** 例) scmp.a %rd, imm20	imm20≤0x7f cmp.a %rd, imm20(6:0)	0x7f<imm20 ext imm20(19:7) cmp.a %rd, imm20(6:0)	—
xcmp **xcmc**	**%rd, imm16** 例) xcmc %rd, imm16	imm16≤0x7f cmc %rd, imm16(6:0)	0x7f<imm16 ext imm16(15:7) cmc %rd, imm16(6:0)	—
xcmp.a	**%rd, imm24** 例) xcmp.a %rd, imm24	imm24≤0x7f cmp.a %rd, imm24(6:0)	0x7f<imm24≤0xfffff ext imm24(19:7) cmp.a %rd, imm24(6:0)	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) cmp.a %rd, imm24(6:0)
sand **soor** **sxor** **snot**	**%rd, imm16** 例) sand %rd, imm16	imm16≤0x7f and %rd, imm16(6:0)	0x7f<imm16 ext imm16(15:7) and %rd, imm16(6:0)	—
xand **xoor** **xxor** **xnot**	**%rd, imm16** 例) xoor %rd, imm16	imm16≤0x7f or %rd, imm16(6:0)	0x7f<imm16 ext imm16(15:7) or %rd, imm16(6:0)	—

備　考

拡張命令の展開形式 (5)

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
scall **scall.d** **sjpr** **sjpr.d**	**label±imm20** 例) scall label+imm20	無条件 ext (label+imm20)(19:12) call (label+imm20)(11:1)	—	—
	sign20 例) sjpr sign20	-1024≤sign20≤1023 jpr sign20(11:1)	sign20<-1024 or 1023<sign20 ext sign20(19:12) jpr sign20(11:1)	—
sjr∗1 **sjr**∗1**.d**	**label±imm20** 例) sjreq label+imm20	無条件 ext (labe+imm20)(19:8) jreq (label+imm20)(7:1)	—	—
	sign20 例) sjrne sign20	-128≤sign20≤127 jrne sign20(7:1)	sign20<-128 or 127<sign20 ext sign20(19:8) jrne sign20(7:1)	—
scalla **scalla.d** **sjpa** **sjpa.d**	**label±imm20** 例) scalla label+imm20	無条件 ext (label+imm20)(19:7) calla (label+imm20)(6:0)	—	—
	imm20 例) sjpa imm20	imm20≤0x7f jpa imm20(6:0)	0x7f<imm20 ext imm20(19:7) jpa imm20(6:0)	—
xcall **xcall.d** **xjpr** **xjpr.d**	**label±imm24** 例) xcall label+imm24	無条件 ext (label+imm24)(23:12) call (label+imm24)(11:1)	—	—
	sign24 例) xjpr sign24	-1024≤sign24≤1023 jpr sign24(11:1)	sign24<-1024 or 1023<sign24 ext sign24(23:12) jpr sign24(11:1)	—

備　考

∗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

拡張命令の展開形式 (6)

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
xjr∗1 **xjr**∗1**.d**	**label±imm24** 例) xjreq label+imm24	無条件 ext (label+imm24)(23:21) ext (label+imm24)(20:8) jreq (label+imm24)(7:1)	— 	—
	sign24 例) xjrne sign24	-128≤sign24≤127 jrne sign24(7:1)	-1048576≤sign24<-128 or 127<sign24≤1048575 ext sign24(20:8) jrne sign24(7:1)	sign24<-1048576 or 1048575<sign24 ext sign24(23:21) ext sign24(20:8) jrne sign24(7:1)
xcalla **xcalla.d** **xjpa** **xjpa.d**	**label±imm24** 例) xcalla label+imm24	無条件 ext (label+imm24)(23:20) ext (label+imm24)(19:7) calla (label+imm24)(6:0)	— 	—
	imm24 例) xjpa imm24	imm24≤0x7f jpa imm24(6:0)	0x7f<imm24≤0xfffff ext imm24(19:7) jpa imm24(6:0)	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) jpa imm24(6:0)
sld.cw **sld.ca** **sld.cf**	**%rd, imm20** 例) sld.cw %rd, imm20	imm20≤0x7f ld.cw %rd, imm20(6:0)	0x7f<imm20 ext imm20(19:7) ld.cw %rd, imm20(6:0)	—
	%rd, symbol±imm20 例) sld.ca %rd, symbol+imm20	無条件 ext (symbol+imm20)(19:7) ld.ca %rd, (symbol+imm20)(6:0)	— 	—
xld.cw **xld.ca** **xld.cf**	**%rd, imm24** 例) xld.cw %rd, imm24	imm24≤0x7f ld.cw %rd, imm24(6:0)	0x7f<imm24≤0xfffff ext imm24(19:7) ld.cw %rd, imm24(6:0)	0xfffff<imm24 ext imm24(23:20) ext imm24(19:7) ld.cw %rd, imm24(6:0)
	%rd, symbol±imm24 例) xld.ca %rd, symbol+imm24	無条件 ext (symbol+imm24)(23:20) ext (symbol+imm24)(19:7) ld.ca %rd, (symbol+imm24)(6:0)	— 	—

備　考

∗1)　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

セイコーエプソン株式会社
マイクロデバイス事業部　デバイス営業部

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：411086411
2007年 9月作成 ⓁB
2011年 2月改訂 Ⓓ
2012年 4月改訂 Ⓓ
2013年 7月改訂 Ⓓ
2015年 9月改訂 Ⓓ