EPSON

CMOS 16-BIT SINGLE CHIP MICROCONTROLLER
(S1C17 Family C コンパイラパッケージ Ver. 3.2)

GNU17 Ver.3.2
チュートリアル

セイコーエプソン株式会社

Rev.1.1

本資料のご使用につきましては、次の点にご留意願います。

本資料の内容については、予告なく変更することがあります。

1. 本資料の一部、または全部を弊社に無断で転載、または、複製など他の目的に使用することは堅くお断りします。
2. 弊社製品のご購入およびご使用にあたりましては、事前に弊社営業窓口で最新の情報をご確認いただきますとともに、弊社ホームページなどを通じて公開される最新情報に常にご注意ください。
3. 本資料に掲載されている応用回路、プログラム、使用方法などはあくまでも参考情報です。お客様の機器・システムの設計において、応用回路、プログラム、使用方法などを使用する場合には、お客様の責任において行ってください。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ならびに損害の発生に対し、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。
4. 弊社は常に品質、信頼性の向上に努めていますが、一般的に半導体製品は誤作動または故障する場合があります。弊社製品のご使用にあたりましては、弊社製品の誤作動や故障により生命・身体に危害を及ぼすこと又は財産が侵害されることのないように、お客様の責任において、お客様のハードウエア、ソフトウエア、システムに必要な安全設計を行うようお願いします。なお、設計および使用に際しては、弊社製品に関する最新の情報(本資料、仕様書、データシート、マニュアル、弊社ホームページなど)をご確認いただき、それに従ってください。また、上記資料などに掲載されている製品データ、図、表などに示す技術的な内容、プログラム、アルゴリズムその他応用回路例などの情報を使用する場合は、お客様の製品単独およびシステム全体で十分に評価を行い、お客様の責任において適用可否の判断をお願いします。
5. 弊社は、正確さを期すために慎重に本資料およびプログラムを作成しておりますが、本資料およびプログラムに掲載されている情報に誤りがないことを保証するものではありません。万一、本資料およびプログラムに掲載されている情報の誤りによってお客様に損害が生じた場合においても、弊社は一切その責任を負いかねます。
6. 弊社製品の分解、解析、リバースエンジニアリング、改造、改変、翻案、複製などは堅くお断りします。
7. 弊社製品は、一般的な電子機器(事務機器、通信機器、計測機器、家電製品など)に使用されること(一般用途)、および本資料に個別に掲載または弊社が個別に指定する用途に使用されること(指定用途)を意図して設計、開発、製造されています。これら一般用途および指定用途以外の用途(特別な品質、信頼性が要求され、その誤動作や故障により生命・身体に危害を及ぼす恐れ、膨大な財産侵害を引き起こす恐れ、もしくは社会に深刻な影響を及ぼす恐れのある用途。以下、特定用途といいます)に使用されることを意図していません。お客様に置かれましては、弊社製品を一般用途および指定用途に使用されることを推奨いたします。もし特定用途で弊社製品のご使用およびご購入を希望される場合、弊社はお客様が弊社製品を使用されることへの商品性、適合性、安全性について、明示的・黙示的に関わらずいかなる保証を行うものではありません。お客様が特定用途での弊社製品の使用を希望される場合は、弊社営業窓口まで事前にご連絡の上、承諾を得てください。
   【特定用途(例)】
   宇宙機器(人工衛星・ロケットなど) / 輸送車両並びにその制御機器(自動車・航空機・列車・船舶など)
   医療機器/ 海底中継機器 / 発電所制御機器 / 防災・防犯装置 / 交通用機器 / 金融関連機器
   上記と同等の信頼性を必要とする用途。詳細は、弊社営業窓口までお問い合わせください。
8. 本資料に掲載されている弊社製品および当該技術を国内外の法令および規制により製造・使用・販売が禁止されている機器・システムに使用することはできません。また、弊社製品および当該技術を大量破壊兵器等の開発および軍事利用の目的その他軍事用途等に使用しないでください。弊社製品または当該技術を輸出または海外に提供する場合は、「外国為替及び外国為替法」、「米国輸出管理規則(EAR)」、その他輸出関連法令を遵守し、係る法令の定めるところにより必要な手続きを行ってください。
9. お客様が本資料に掲載されている諸条件に反したことに起因して生じたいかなる損害(直接・間接を問わず)に関して、弊社は一切その責任を負いかねます。
10. お客様が弊社製品を第三者に譲渡、貸与などをしたことにより、損害が発生した場合、弊社は一切その責任を負いかねます。
11. 本資料についての詳細に関するお問合せ、その他お気付きの点などがありましたら、弊社営業窓口までご連絡ください。
12. 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。

評価ボード・キット、開発ツールご使用上の注意事項

1. 弊社評価ボード・キット、開発ツールは、お客様での技術的評価、動作の確認および開発のみに用いられることを想定し設計されています。それらの技術評価・開発等の目的以外には使用しないでください。本品は、完成品に対する設計品質に適合していません。
2. 弊社評価ボード・キット、開発ツールは、電子エンジニア向けであり、消費者向け製品ではありません。お客様において、適切な使用と安全に配慮願います。弊社は、本品を用いることで発生する損害や火災に対し、いかなる責も負いかねます。通常の使用においても、異常がある場合は使用を中止してください。
3. 弊社評価ボード・キット、開発ツールに用いられる部品は、予告なく変更されることがあります。

Rev. j1.2, 2023.4

はじめに

本書は S1C17 Family マイクロコントローラを使用する製品を開発される設計者、プログラマを対象としたチュートリアルで、S1C17 Family のソフトウェア開発ツール“S1C17 Family C コンパイラパッケージ”による組み込みプログラムの開発手順を説明します。

参照文書

本書に記載されていない内容については、必要に応じて以下のマニュアルや書籍を参照してください。

参照項目	参照文書
C 言語(ANSI C 準拠)および C ソースプログラムの作成方法	ANSI C を解説した一般の書籍
GNU17 IDE(Eclipse IDE for C/C++ Devel- opers Package)の基本的な操作方法	Eclipse IDE for C/C++ Developers Package を解説した一般の書籍
GNU C、binutil、および GNU リンカ(ld)用リンカスクリプト	GNU ツールのマニュアル
Windows の基本的な操作方法	Windows のマニュアル
S1C17 Family の命令セット	S1C17 Family S1C17 コアマニュアル
S1C17 Family の開発ツール	S5U1C17001C マニュアルおよびハードウェア開発ツールマニュアル
S1C17 Family マイクロコントローラ	S1C17xxx テクニカルマニュアル

動作環境、インストール方法

S1C17 Family C コンパイラパッケージの動作環境、インストール方法、インストール後のフォルダ構成については、パッケージに添付の Readme ファイルを参照してください。

日本語および各国語への対応

GNU17 IDE は Eclipse IDE for C/C++ Developers Package を元に作成されており、インストールされるワークベンチのユーザインタフェースはすべて英語で表示されます。日本語で表示させたい場合は、日本語化言語パックを導入してください。ほかの言語で表示させたい場合も同様です。GNU17 IDE のメニューから Babel プロジェクトの言語パックをインストールする例を Appendix B に掲載しています。本書はオリジナル(英語)の表示のまま説明します。

目次

1. ソフトウェア開発フロー

S1C17 Family C コンパイラパッケージ(S5U1C17001C)は、S1C17 Family マイクロコントローラに搭載するソフトウェアを開発するための統合開発環境(GNU17 IDE)を提供します。GNU17 IDE を使用して、ソースファイルの作成からコンパイル/アセンブル、リンク、デバッグ、最終的な提出データファイルの作成まで行うことができます。本書は、基本的なソフトウェア開発の流れを、IDE の操作手順と共に解説します。

1.1 ソフトウェア開発ツールの構成

図 1.1.1 に、S1C17 Family C コンパイラパッケージ(S5U1C17001C)に含まれるソフトウェア開発ツールの構成を示します。

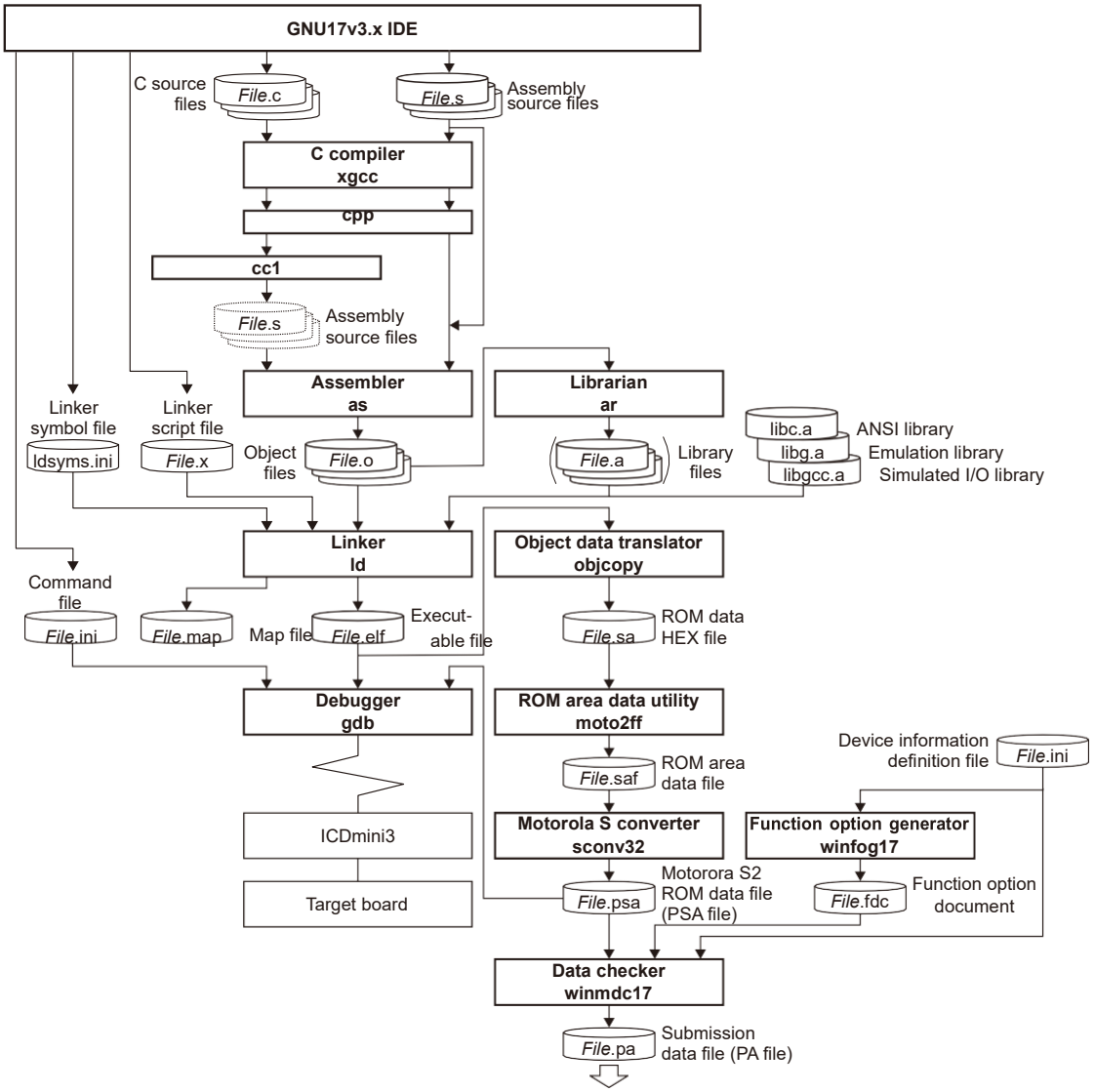


図 1.1.1　ソフトウェア開発ツールとファイルの構成

1.2 GNU17 IDE によるソフトウェア開発

ここでは、GNU17 IDE(以下 IDE)によるソフトウェア開発手順の概要を説明します。実際の操作方法については、別途チュートリアルの章を用意しています。
IDE による基本的なソフトウェア開発手順を図 1.2.1 に示します。

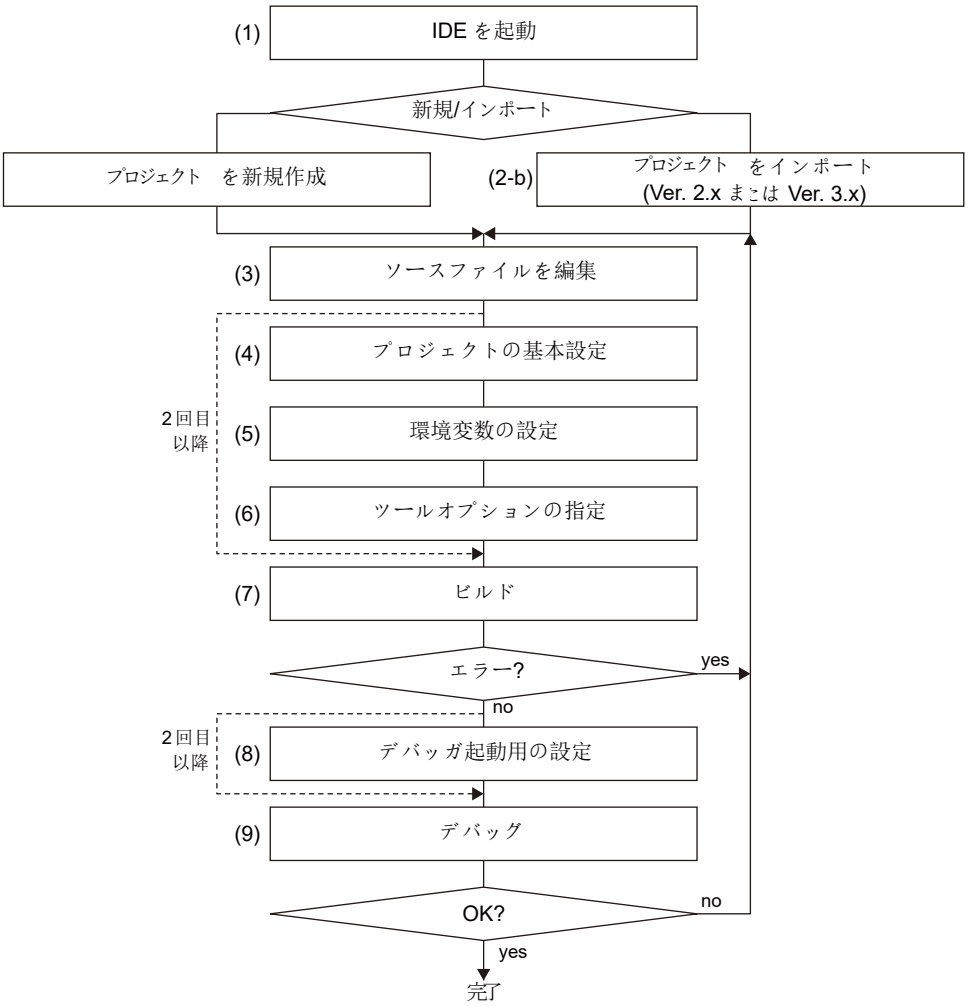


図 1.2.1 IDE によるソフトウェア開発手順

(1) IDEを起動

IDE は、Eclipse に S1C17 Family 用のプラグインを追加した統合開発環境ソフトウェアです。1 つのワークベンチ上でソースの作成からデバッグ、セイコーエプソンに提出する PA ファイルの作成までを行うことができます。
始めに、Windows 上で IDE を起動します。以下の処理/操作は IDE で行います。

(2) プロジェクトを新規作成/インポート

IDE によるソフトウェア開発では、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。

(2-a) プロジェクトを新規作成
利用可能なプロジェクトがない場合、新規プロジェクトを作成します。

(2-b) プロジェクトをインポート
すでに IDE でプロジェクトが作成されている場合、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改訂の作業などを行うことができます。

注: GNU17 Ver. 3.x では、新たに割り込みベクタテーブルやブート処理用のライブラリファイル(起動処理ライブラリ)が追加され、基本的にこのライブラリを使用してビルドが行われます。このため、通常(GNU17 Ver. 3.x)のプロジェクトのインポート機能に加え、このライブラリファイルを持たない GNU17 Ver. 2.x で作成されたプロジェクトをインポートするための機能も用意されています。詳細は後述のチュートリアルで説明します。

(3) ソースプログラムの編集

ソースファイルをプロジェクトに追加します。
ソースファイルは IDE のエディタで作成/編集可能です。既存のソースファイルや汎用のエディタ等で作成したソースファイルをインポートして編集することもできます。
注: GNU17 Ver. 2.x のプロジェクトをインポートした場合は、GNU17 Ver. 3.x の新機能に合わせ、ソースの編集が必要になります。詳細は後述のチュートリアルで説明します。

(4) プロジェクトの基本設定

プログラムタイプ(実行形式ファイル/ライブラリファイル)、ターゲット CPU、メモリモデル(16MB/64KB)、GCC バージョンを設定します。この設定により、下記の(5)と(6)のほとんどの項目が自動設定されます。

(5) 環境変数の設定

必要に応じ、リンクするユーザライブラリやターゲット CPU のコプロセッサの種類を指定します。

(6) ツールオプションの指定

必要に応じ、C コンパイラ、アセンブラ、およびリンカの起動オプション、リンクに使用するリンカスクリプトファイルの指定や編集を行います。
(4)～(6)はビルドに必要な情報を設定するもので、1 度設定してしまえば、2 回目以降のビルド時はスキップできます。

(7) ビルド

ソースプログラムの作成/編集とビルド用の設定が終了後、ビルドを実行します。以下の処理が順次実行され、デバッグ可能な elf 形式のオブジェクトファイルや提出用データファイル(PA ファイル)が生成されます。

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生成された複数のオブジェクトファイルをリンカ ld によって 1 つにまとめ、デバッグに必要な情報なども含み ROM 上に配置して実行可能な elf 形式のオブジェクトファイルを生成します。

S レコード変換
objcopy、moto2ff、sconv32 を順次起動し、elf 形式のオブジェクトファイルから ROM データを抽出してモトローラ S2 形式の(使用領域に 0xff が埋め込まれた)PSA ファイルを生成します。
実機上での最終動作確認には、この PSA ファイルを使用します。

提出用データファイル(PA ファイル)の生成

winmdc17 を起動して、PSA ファイルからセイコーエプソンに提出する PA ファイルを生成します。このファイルは、MCU の内蔵 ROM/フラッシュメモリへのユーザプログラム書き込みをセイコーエプソンに依頼する場合に必要です。その際は、動作確認を終えた PSA ファイルから生成された PA ファイルを提出してください。

ファンクションオプションを持つ MCU の場合は、winfog17 と winmdc17 を別途起動して PA ファイルを生成する必要があります。その詳細は個別機種の“テクニカルマニュアル”および“S5U1C17001C マニュアル”を参照してください。

(8) デバッガ起動用の設定

デバッガ起動用のコマンドファイルを指定し、必要に応じてその内容を編集します。

この作業はデバッガの起動に必要な情報を設定するもので、1 度設定してしまえば、2 回目以降のデバッグ時はスキップできます。

(9) 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または 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。ICDmini 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 Family マイクロコントローラ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。

(10) ライブラリファイルの生成

上記のツール以外に、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 マイクロコントローラを使用したアプリケーションの開発に有効です。

2. チュートリアル 1（プロジェクトの作成からデバッグまでの基本操作）

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、“S5U1C17001C マニュアル”を参照してください。

使用するファイル

説明は“C:¥EPSON¥GNU17V3¥sample¥tutorial”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。

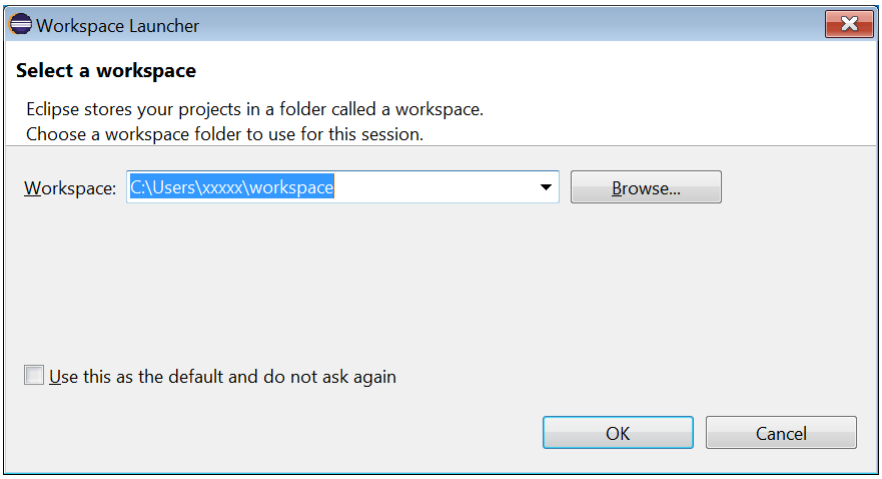
¥src¥tutorial.c
¥src¥sub.c
¥inc¥sub.h

以下、こ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、画面表示例は使用環境などにより異なる場合があります。

2.1 IDE の起動

操作 1: Windows のスタートメニューから[EPSON MCU] > [GNU17V3] > [GNU17V3 IDE]を選択して(あるいは“C:¥EPSON¥GNU17V3¥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て)、IDE を起動させます。

Eclipse のスプラッシュに続き、[Workspace Launcher]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。

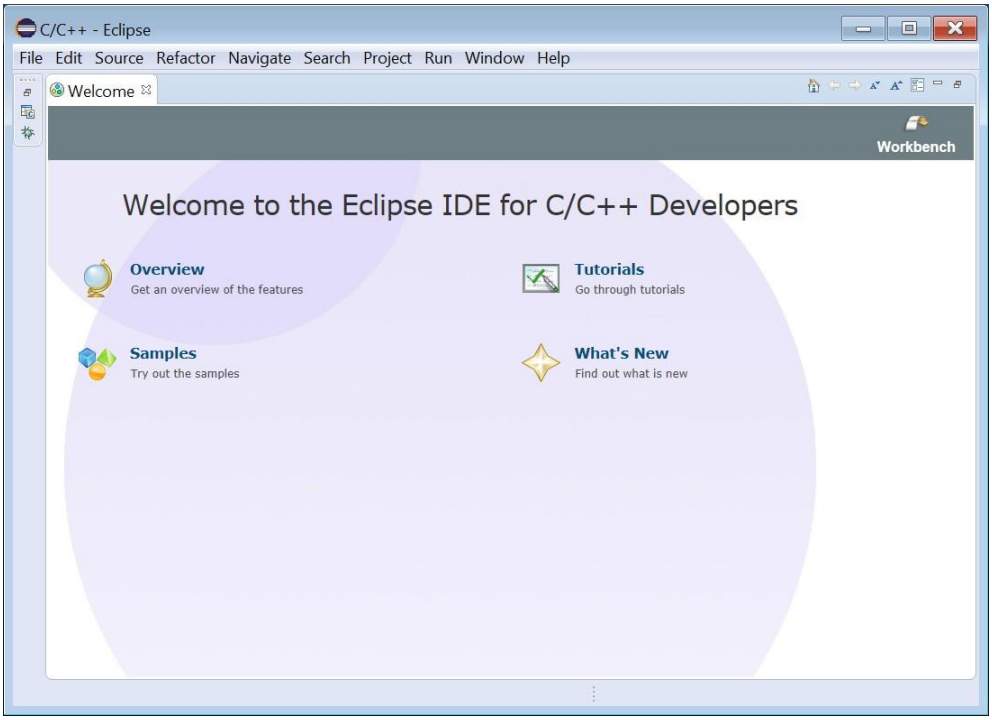


任意のディレクトリを選択または新規作成してワークスペースに設定可能ですが、このチュートリアルでは表示されたデフォルトのワークスペースディレクトリを使用します。

注: 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([Copy projects into workspace]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。

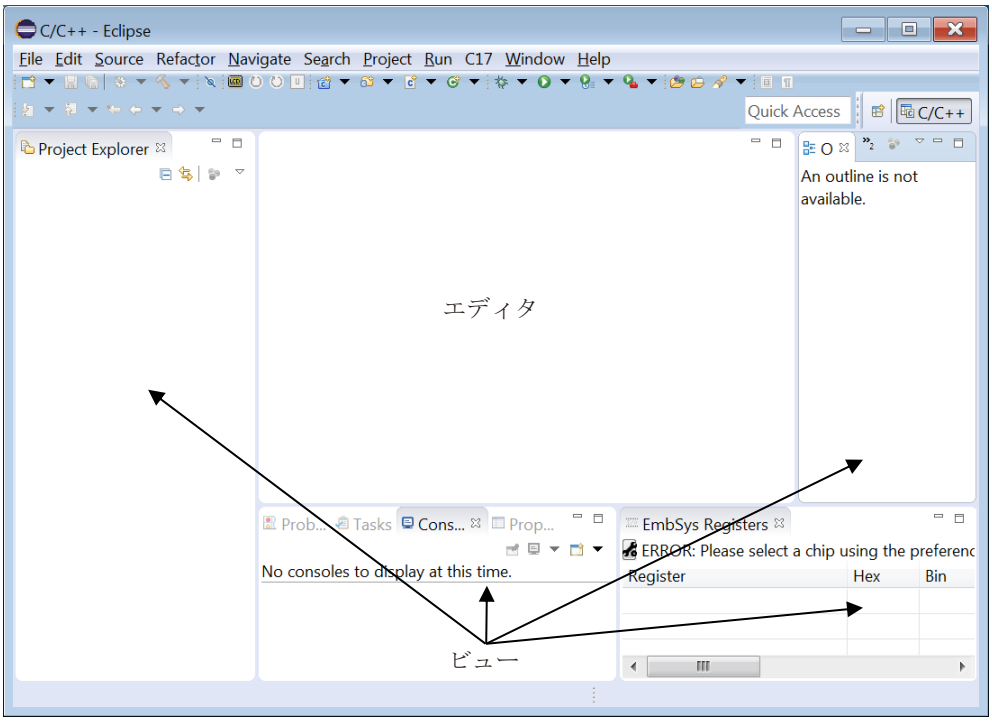
IDE ウィンドウが表示されます。



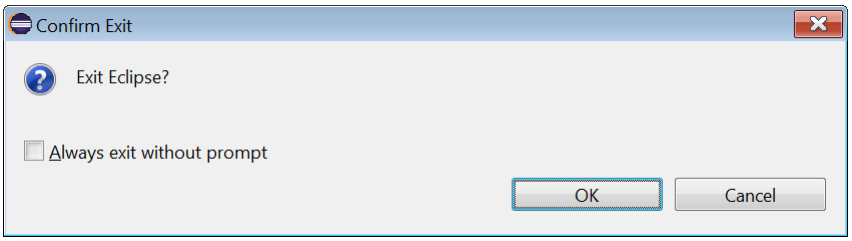
最初に IDE を起動した際には、[Welcome]ページが表示されます。

操作 3: [Welcome]タブの⊠(Close)ボタンをクリックして、[Welcome]ページを閉じます。

*次回以降、IDE 起動時に[Welcome]ページは表示されません。なお、[Help] > [Welcome]の選択により、再表示が可能です。



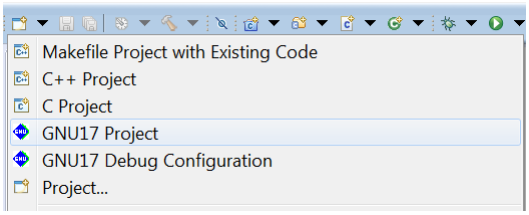
*チュートリアルを途中で終了させるには、IDE の[File]メニューから[Exit]を選択してください。
ウィンドウ (閉じる)ボタンでも終了可能です。次のダイアログボックスが表示された場合、終 了するには[OK]ボタンを、終了を取り消すには[Cancel]ボタンをクリックします。



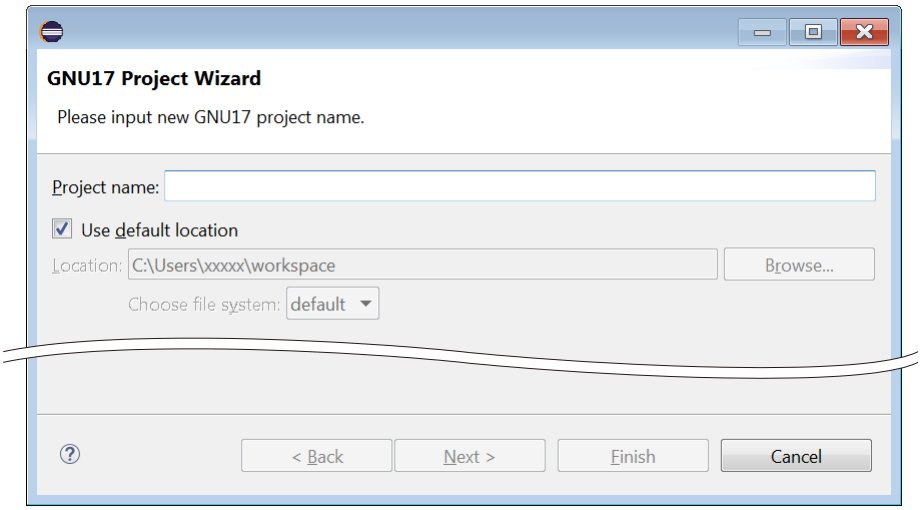
2.2 プロジェクトの作成

アプリケーション開発においては、ソースファイルやヘッダファイルなど、複数の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、プロジェクトごとに指定したプロジェクト名を持つフォルダが作られます。

プロジェクトを新規作成するには
操作 1: ツールバー (New)ドロップダウンリスト*から[GNU17 Project]を選択します。
*その他、[File]メニュー > [New]、ツールバー (New C/C++ Project)ドロップダウンリストなどからも選択可能

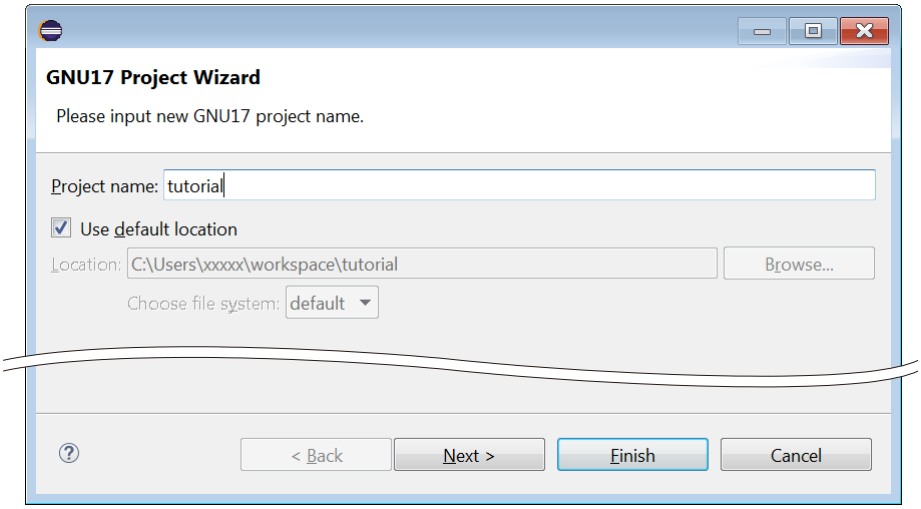


[GNU17 Project]ウィザードが起動します。



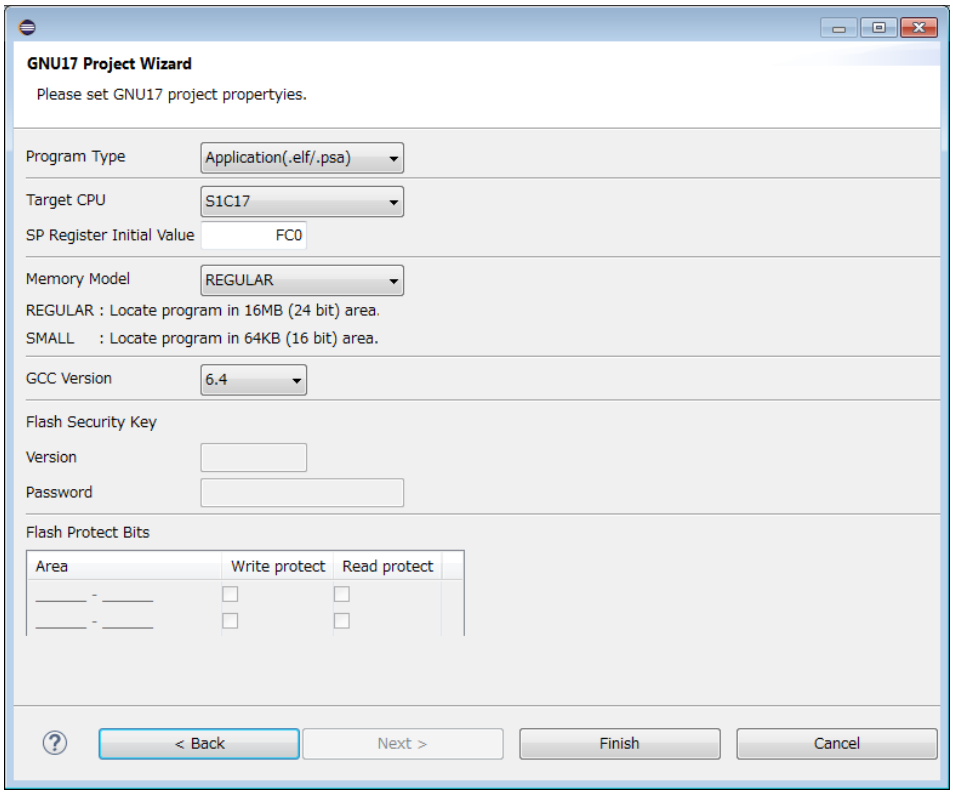
プロジェクト名の指定

操作 2: [Project name:]にプロジェクト名“tutorial”を入力します。



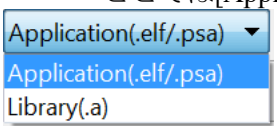
[Use default location]チェックボックスを選択しておくと、IDE 起動時に指定したワークスペースディレクトリ内に“tutorial”というプロジェクトフォルダが生成されます。
また、プロジェクトのビルドにより生成される実行形式のオブジェクトファイル(.elf/.psa)にはここで指定した名前が付けられます。

操作 3: [Next >]ボタンをクリックします。
プロジェクトプロパティ設定画面に移行します。



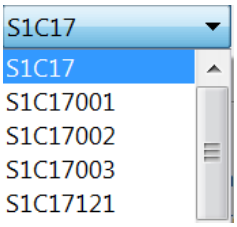
プログラムタイプの選択

操作 4: [Program Type]のドロップダウンリストから、生成するプログラムファイルの種類を選択します。ここでは[Application (.elf/.psa)]を選択してください。



ターゲット CPU の指定

操作 5: [Target CPU]のドロップダウンリストから、アプリケーションで使用する S1C17 MCU を選択します。ここでは[S1C17] (機種依存なし)を選択してください。

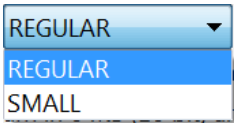


SP レジスタ初期値の設定

操作 6: [SP Register Initial Value]に、SP レジスタ(スタックポインタ)の初期値を設定します。通常はここに表示されているとおり、選択されているターゲット CPU の RAM 容量に従って適切な値に設定されますので、特に変更する必要はありません。

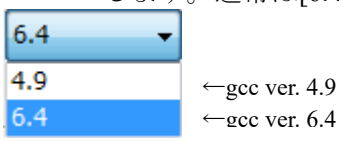
メモリモデルの選択

操作 7: [Memory Model]のドロップダウンリストから、アプリケーションで使用する S1C17 MCU 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。



GCC バージョンの選択

操作 8: [GCC Version]のドロップダウンリストから、使用する C コンパイラ“gcc”のバージョンを選択します。通常は[6.4]のままとしてください。



Flash セキュリティキーの設定

操作 9: ターゲット CPU が 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場合は、Flash セキュリティを解除するためのパスワードを[Password]に入力します。このチュートリアルでは設定する必要はありません。

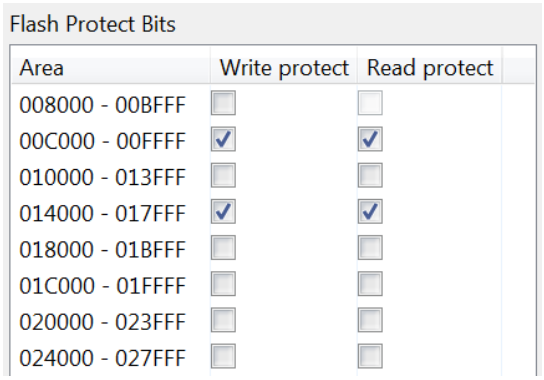
ここに設定したパスワードは、winmdc17 による提出用データファイル(PA ファイル)の生成時、およびデバッガ起動用コマンドファイル中で“c17 pwul”コマンドの引数として使用されます。
[Version]は選択したターゲット CPU により、自動的に設定されます。Flash セキュリティに対応していないターゲット CPU が選択されている場合、[Version]と[Password]は入力禁止状態になります。

Flash プロテクトビットの設定

操作 10: [Flash Protect Bits]フィールドで、プロテクトする Flash 領域を選択します。このチュートリアルでは設定の必要はありません。

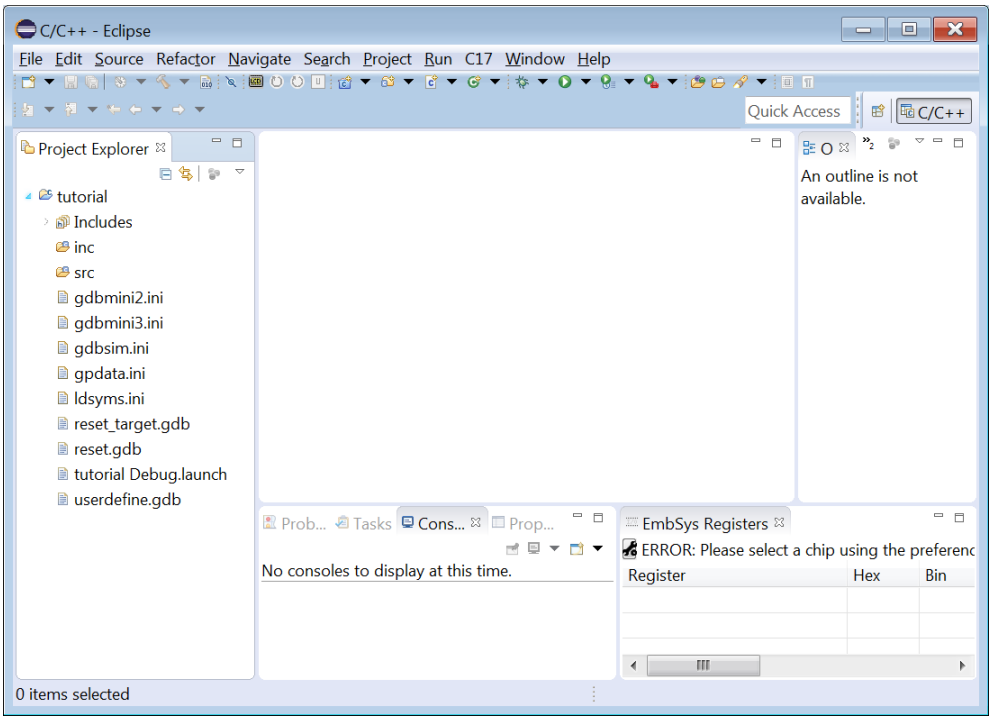
プロテクトビットに対応したターゲット CPU を選択すると、[Area]フィールドには Flash プロテクト対象領域の開始アドレスと終了アドレスが表示されます。
右側の[Write protect]または[Read Protect]チェックボックスを選択すると、ライトプロテクトまたはリードプロテクト(もしくは両方)が有効になり、その領域に対するデータ書き込みまたは読み出し操作が禁止されます。書き込み/読み出しを許可する領域は、チェックボックスを非選択状態にしておきます。
デバッグを行う Flash 領域はチェックボックスを両方共非選択にしておきます。

例: 0xc000～0xffff および 0x14000～0x17fff のアドレス範囲にリード/ライトプロテクトを設定



リード/ライトプロテクトを設定した場合、IDE は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。

操作 11: [Finish]ボタンをクリックします。



[GNU17 Project]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、SP レジスタ初期値、メモリモデル、GCC バージョン Flash セキュリティキー、Flash プロテクトビットの設定は、後から変更が可能です。

生成されたフォルダ/ファイル

作成された“tutorial”プロジェクトフォルダには以下のフォルダ/ファイルが生成されます。

tutorial	プロジェクトフォルダ
Includes	インクルードパス
C:/EPSON/GNU17V3/gcc6/include	gcc ver. 6.4 用ヘッダファイルへのパス
tutorial/inc	tutorial プロジェクトヘッダファイルへのパス
inc	tutorial プロジェクト用ヘッダファイル格納用フォルダ
src	tutorial プロジェクト用ソースファイル格納用フォルダ
gdbmini2.ini	デバッガ起動用コマンドファイル(ICDmini(ICDmini2)モード用)
gdbmini3.ini	デバッガ起動用コマンドファイル(ICDmini(ICDmini3)モード用)
gdbsim.ini	デバッガ起動用コマンドファイル(シミュレータモード用) gpdata.ini
gpdata	オプションファイル(Gang Programmer 用)
ldsyms.ini	リンカシンボルファイル
reset_target.gdb	GDB コマンドファイル(ターゲットリセット用)
reset.gdb	GDB コマンドファイル(CPU リセット用)
tutorial Debug.launch	デバッガ起動設定用ファイル
userdefine.gdb	GDB コマンドファイル(ユーザ定義用)

2.3 ソースファイルの作成と追加

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。

2.3.1 ソースファイルの作成

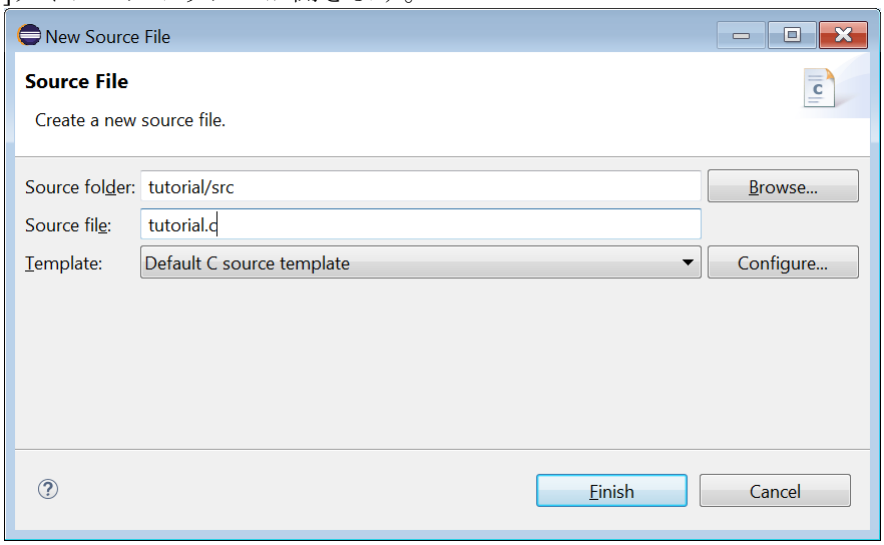
ソースファイル/ヘッダファイルは IDE のエディタまたは汎用のエディタで作成します。また、S1C17 Family 用アプリケーションの既存のソースファイルを使用することもできます。
このチュートリアルではサンプルとして用意されたソースファイルを使用しますので、ソースを新規に作成する必要はありません。参考として、操作 1〜2 に新規作成手順を示します(ここでの操作は不要です)。

ソース/ヘッダファイルを新規作成するには

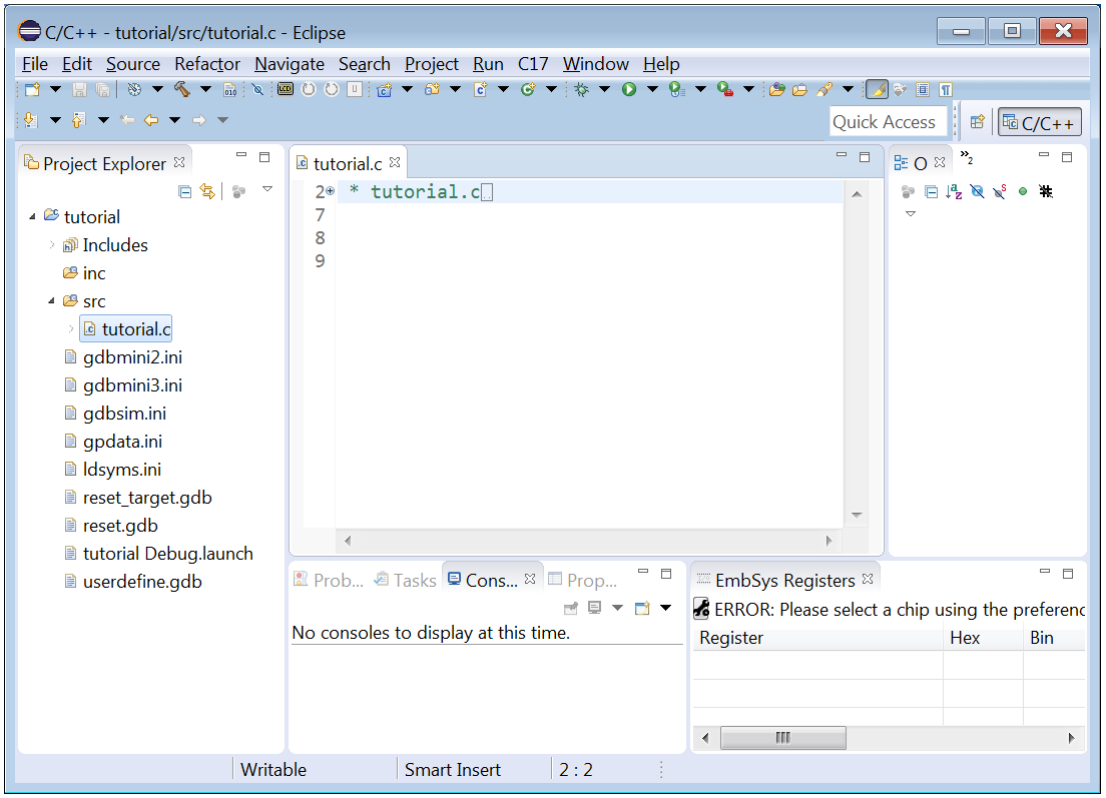
操作 1: [Project Explorer]ビューの“tutorial” > “src”を選択し、ツールバー (New)ドロップダウンリスト*から[Source File]を選択します。(ヘッダファイルを作成する場合は“tutorial” > “inc” を選択し、[New] > [Header File]を選択します。)

*その他、[File]メニュー > [New]、ツールバー (New C/C++ Source File)ドロップダウンリストなどからも選択可能

[New Source File]ダイアログボックスが開きます。



操作 2: [Source file:]([Header file:])に作成するソースファイル名(ヘッダファイル名)を入力し、[Finish] ボタンをクリックします。



選択または指定したフォルダ内に指定名称のファイルが生成されます。また、エディタのタブに作成したファイルの名称が表示され、テキストの入力が可能な状態になります。

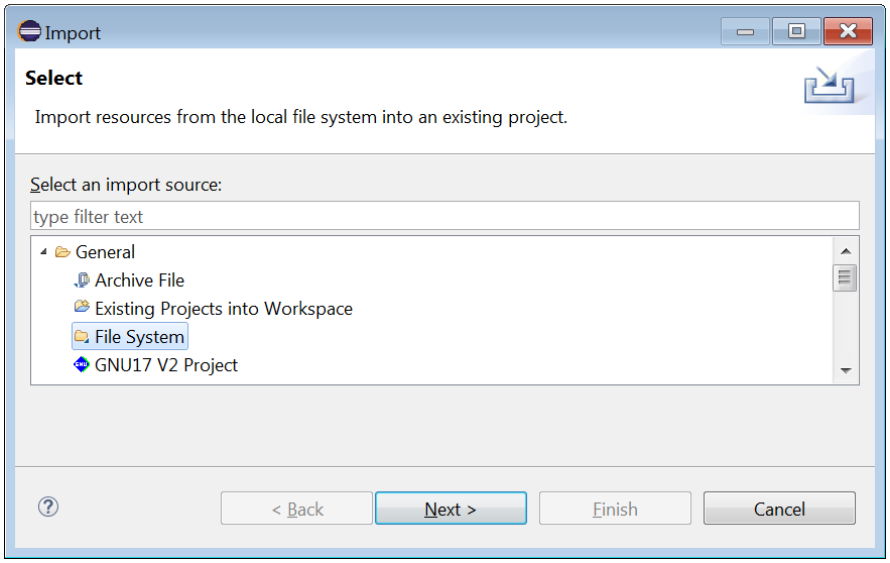
2.3.2 ソースファイルのインポート

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。

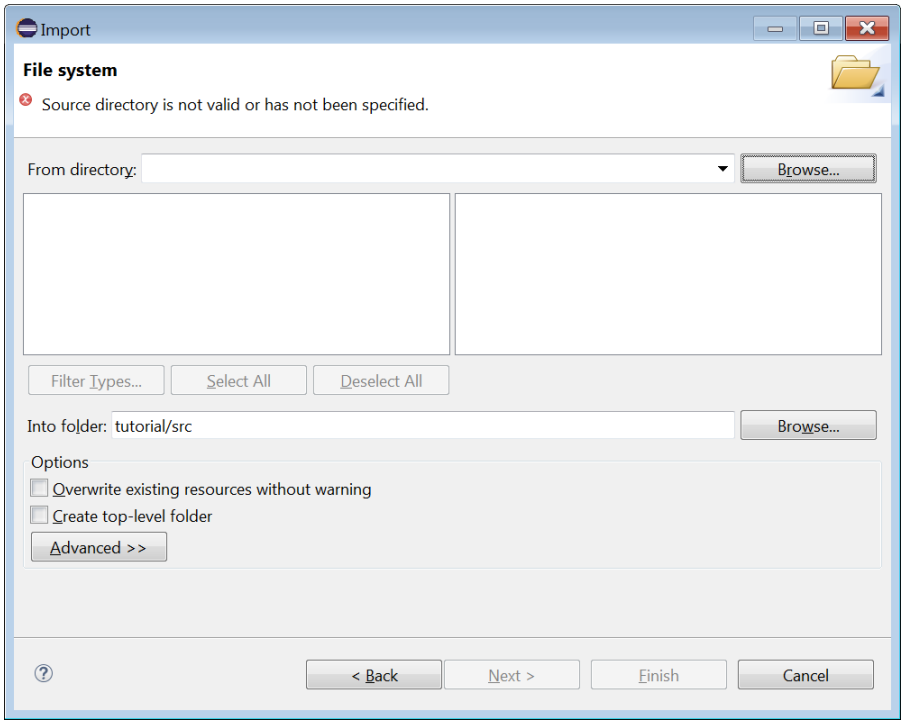
ソース/ヘッダファイルをインポートするには

操作 3: [Project Explorer]ビューの“tutorial”>“src”を選択し、[File]メニューから[Import...]を選択します。

[Import]ウィザードが起動します。

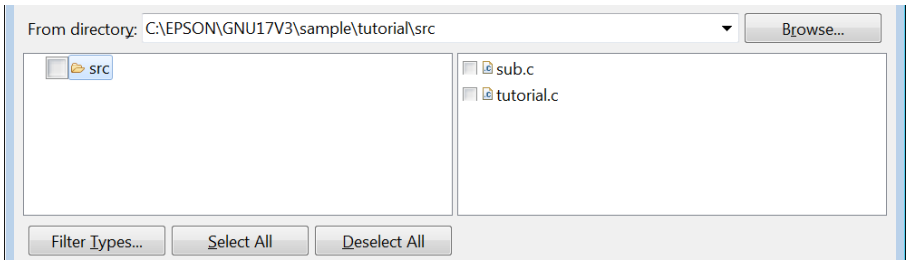


操作 4: 表示されている一覧の中から[General] > [File System]を選択し、[Next >]ボタンをクリックします。

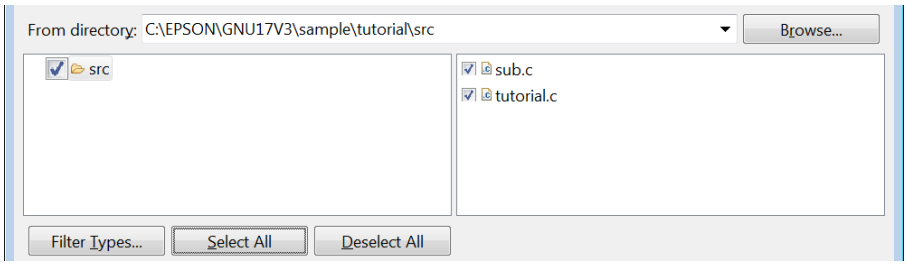


操作 5: [From directory:]の[Browse...]ボタンをクリックします。[Import from directory]ダイアログボックスが表示されますので、IDE をインストールしたドライブ(C)から“¥EPSON¥GNU17V3¥sample¥tutorial¥src”ディレクトリを選択し、[OK]ボタンをクリックします。

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。

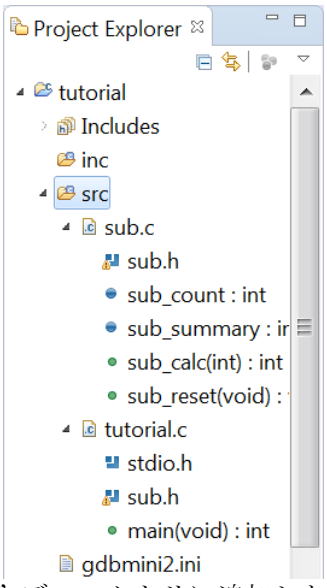


操作 6: [Select All]ボタンをクリックして、ソースファイルを選択します。



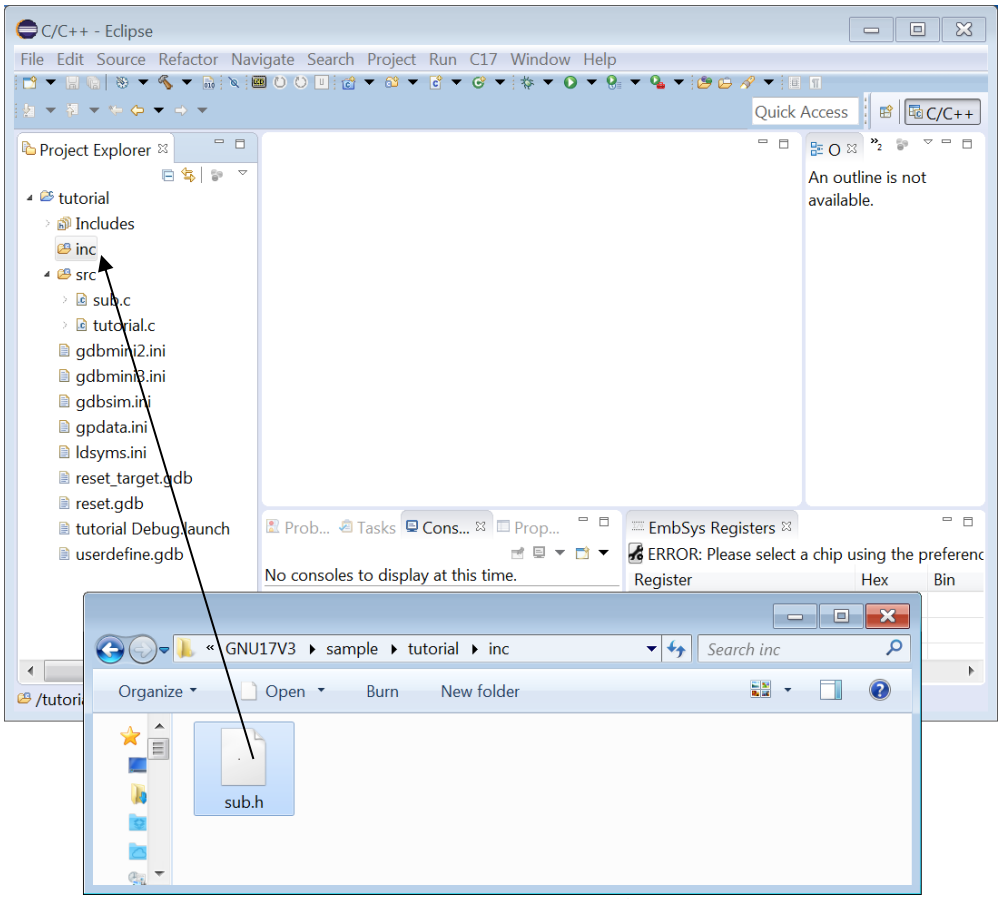
操作 7: 上記のとおり表示されていることを確認後、[Finish]ボタンをクリックします。
上記の操作により、“sub.c”と“tutorial.c”がプロジェクトに追加されました。

操作 8: [Project Explorer]ビューの“tutorial” ＞ “src” ＞ “sub.c/tutorial.c”を展開します。

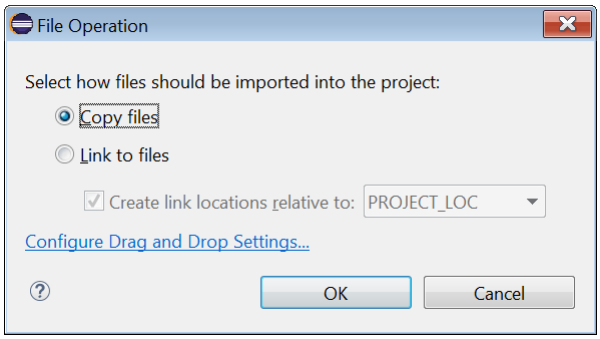


[Project Explorer]ビューの“tutorial¥src”ディレクトリに追加したソースファイルと、そのファイルにインクルードされているヘッダファイル、定義されているグローバル変数と関数が表示されます。

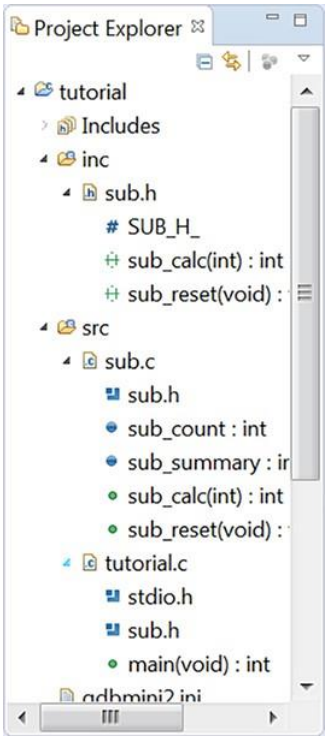
操作 9: ヘッダファイル“C:¥EPSON¥GNU17V3¥sample¥tutorial¥inc¥sub.h”をインポートしてください。次のように Windows のエクスプローラーから IDE の[Project Explorer]ビューへドラッグアンドドロップすることでもインポートできます。(“tutorial”プロジェクトの“inc”フォルダへインポート)



ファイルをドラッグアンドドロップすると、[File Operation]ダイアログボックスが表示されます。



操作 10: [Copy files]を選択し、[OK]ボタンをクリックします。

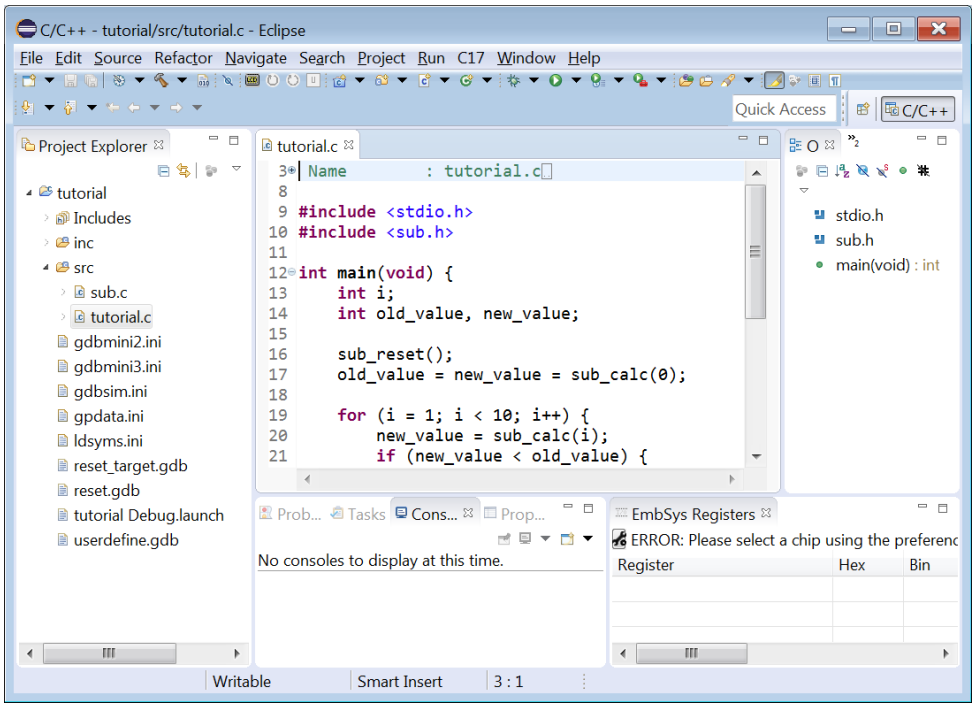


2.3.3 ソースファイルの表示と編集

新規作成したソースファイル、および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。

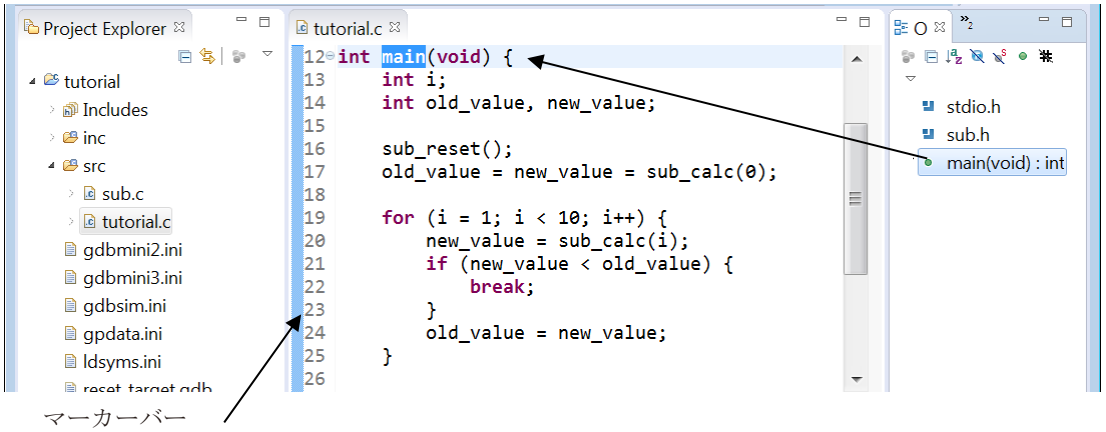
ファイルを編集するには

操作 11: [Project Explorer]ビューの“tutorial.c”をダブルクリックします。



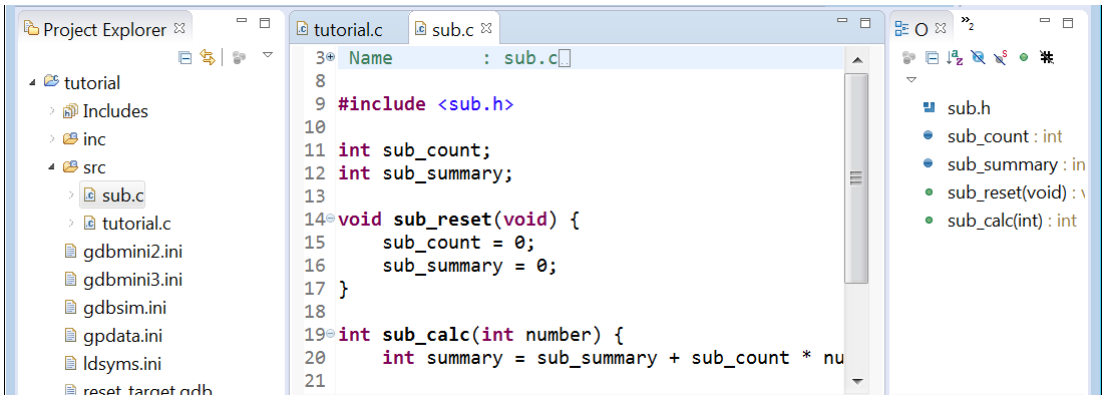
“tutorial.c”の内容がエディタ上に表示されます。ここで汎用エディタと同様にソースの修正が行えます。エディタの機能と操作方法については、Eclipse の Help や、Eclipse IDE for C/C++ Developers Package を解説した一般の書籍を参照してください。
C ソースを表示させた場合、C の予約語、コメント、文字列が色付きで強調表示されます。
また、選択したファイルを通常使用している汎用エディタで開くように設定することも可能です。

操作 12: [Outline]ビューの“main(void): int”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 13: [Project Explorer]ビューの“sub.c”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。

操作 14: 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。

プログラムの起動処理

本パッケージには、ベクタテーブルと main 関数の前後にコールされるブートおよび終了のための関数が記述された起動処理ライブラリ“crt0.o”が含まれており、特に指定しない場合は、このファイルがリンクされます。
RAM と標準ライブラリの初期化、割り込み許可は、起動処理ライブラリが実施していますので、main 関数に実装する必要はありません。起動処理内容を変更する場合は、起動処理ライブラリの仕 様に則り独自の処理を定義してください。

詳細は、“S1C17001C マニュアル”の IDE の章とライブラリの章を参照してください。また、本書の“3.2 GNU17 Ver. 2.x プロジェクトのインポート(“crt0.o”を使用する場合)”に“crt0.o”に合わせたソースの修正について説明されていますので、そちらも参考にしてください。

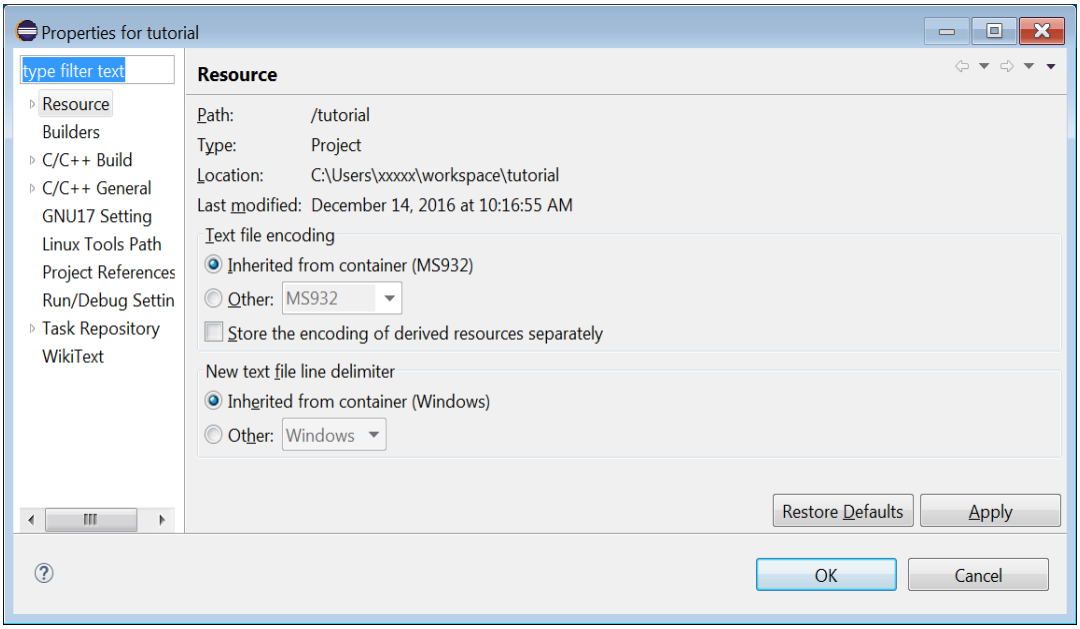
2.4 プロジェクトの基本設定

IDE がプロジェクトをターゲットシステムに合わせて正しくビルドするために、各種の情報(プロパティ)を必要とします。そのほとんどを簡単に設定するために、基本項目をまとめたページが用意されています。
“プロジェクトの作成”の節に示したとおり、プロジェクトの新規作成時に[GNU17 Project]ウィザードで基本項目を設定します。また、各プロジェクトの[Properties]ダイアログボックス上で、後からの変更も可能です。ここでは、[Properties]ダイアログボックスによる設定方法を説明します。各項目の詳細については、“5U1C17001C マニュアル”の“3.4.1 GNU17 用プロジェクトプロパティの設定”を参照してください。

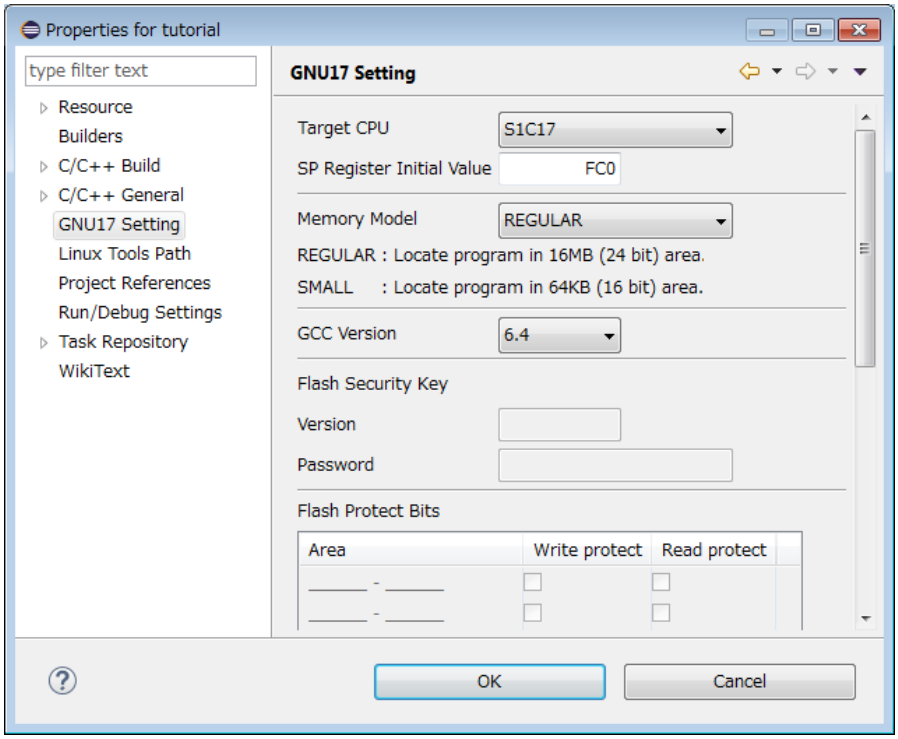
基本設定ページを表示させるには

操作 1: [Project Explorer]ビューの“tutorial”を選択した状態で、[Project]メニュー (または右クリックで表示されるコンテキストメニュー)から[Properties]を選択します。

[Properties]ダイアログボックスが開きます。



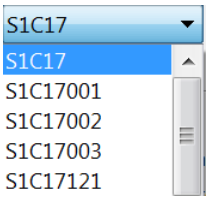
操作 2: プロパティのリストから[GNU17 Setting]を選択します。



基本項目の変更

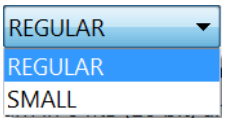
操作 3: [Target CPU]、[Memory Model]、[GCC Version]のドロップダウンリストから変更後の内容を選択します。このチュートリアルでは変更不要です。

ターゲット CPU の選択



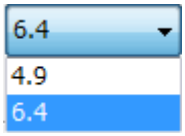
使用する機種を選択します。
機種に依存しないプログラムを作成する場合は、“S1C17”を選択します。

メモリモデルの選択



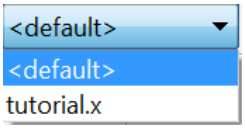
使用する機種が対応するアドレス空間を選択します。
←24 ビットアドレス空間 (16MB)
←16 ビットアドレス空間 (64KB)

GCC バージョンの選択



C コンパイラのバージョンを選択します。
←gcc ver. 4.9
←gcc ver. 6.4

リンカスクリプトの選択



このドロップダウンリストは、プロジェクトフォルダ内にリンカスクリプトファイルが存在する場合にのみ表示され、使用するリンカスクリプトファイルを選択することができます。ここでリンカスクリプトファイルを選択すると、リンカ実行時に-T オプションが指定されます。
<default>を選択すると、リンカに組み込まれている標準的なリンカスクリプトが使用されます。“tutorial”フォルダにはリンカスクリプトファイルが存在しないため、このドロップダウンリストは表示されません。

*リンカスクリプトは、リンカがプログラムやデータをメモリ上の指定位置に配置するために必要な情報です。デフォルトのリンカスクリプトではアプリケーションが構築できない場合は、独自のリンカスクリプトファイルを作成する必要があります。リンカスクリプトの内容と、ウィザードを使用したリンカスクリプトファイルの作成方法については、“Appendix A セクションとリンカスクリプト”で説明します。

[GNU17 Setting]ページでは以上の項目に加え、SP レジスタ初期値、Flash セキュリティキー、Flash プロテクトビットの設定が行えます(“プロジェクトの作成”の節参照)。

操作 4: プロパティ設定を終了する(ダイアログボックスを閉じる)場合は[OK]ボタンを、設定を継続する場合は[Apply]ボタンをクリックします。

2.5 プロジェクトの詳細設定

前節に述べたプロジェクトの基本設定により、ほとんどのプロパティは自動設定されます。それらを含め、すべてのプロパティが[Properties]ダイアログボックスで設定できるようになっていますので、ここでは、それらの設定ページを見ておきます。
なお、チュートリアルでは設定の変更は不要です。

2.5.1 環境変数の設定

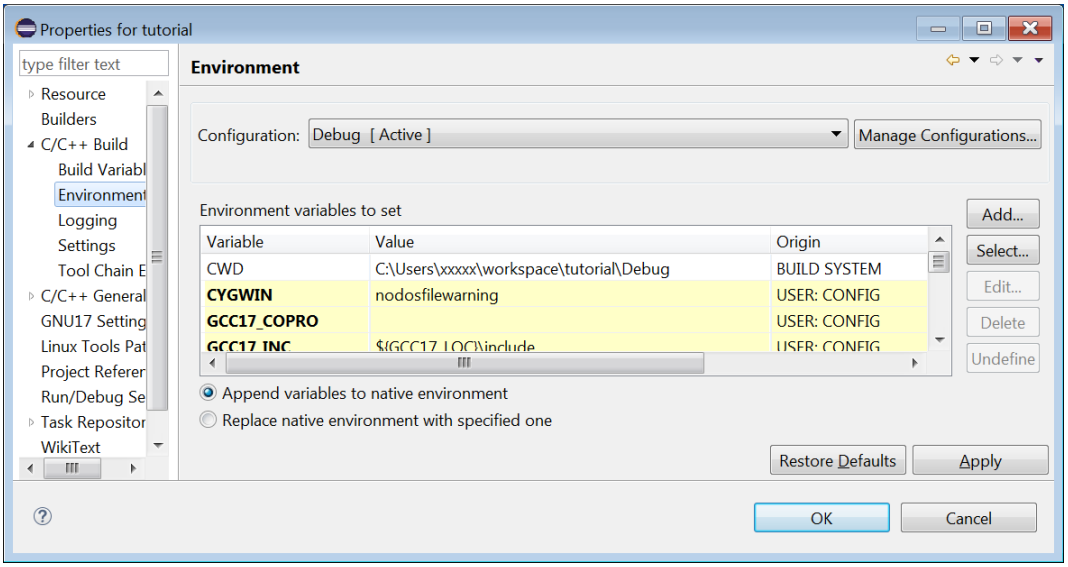
IDE には、プロジェクトのビルドに必要な情報を設定しておく環境変数が設けられています。以下、その設定手順を説明します。環境変数の詳細は、“S5U1C17001C マニュアル”の“3.4.2 環境変数の設定”を参照してください。

環境変数設定用のページを表示させるには

操作 1: [Project Explorer]ビューの“tutorial”を選択した状態で、[Project]メニュー (または右クリックで表示されるコンテキストメニュー)から[Properties]を選択します。

[Properties]ダイアログボックスが開きます。

操作 2: プロパティのリストから[C/C++ Build] > [Environment]を選択します。

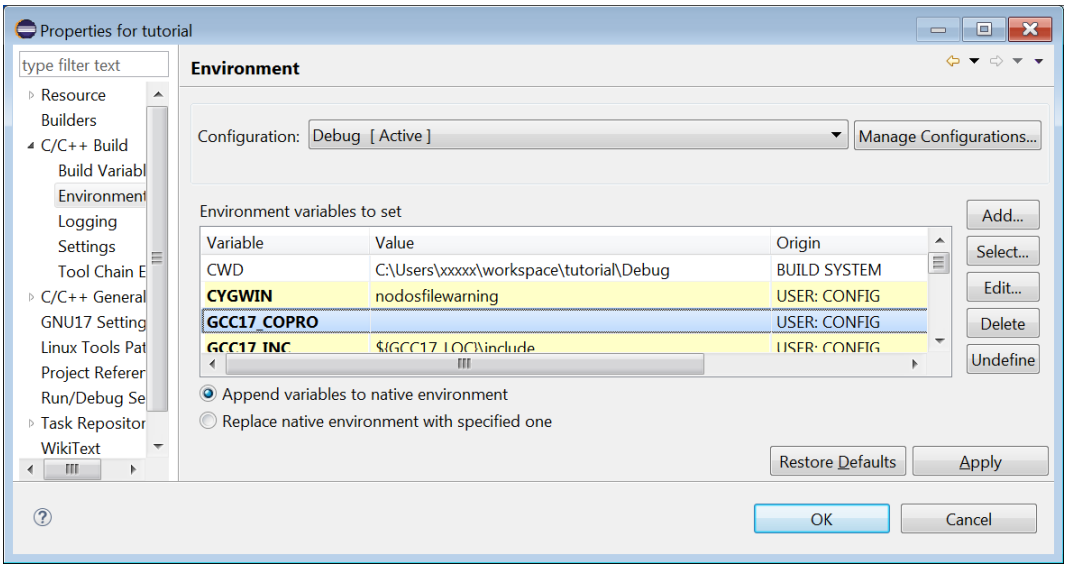


ほとんどの環境変数が、プロジェクトの基本設定に従って設定されています。このページで設定が必要となる可能性のある変数は、“GCC17_COPRO”(コプロセッサの設定)と“GCC17_USER_LIBS”(ユーザライブラリの指定)の 2 つです。

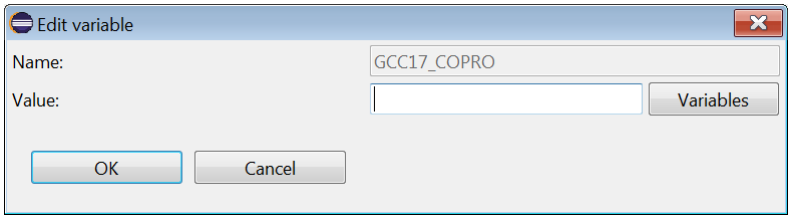
コプロセッサの設定(GCC17_COPRO)

基本設定の[Target CPU]で個別の機種名を選択した場合は、“GCC17_COPRO”も機種に合わせ正しく設定されますので、ここで変更する必要はありません。[Target CPU]で“S1C17”を選択し、特定のコプロセッサに対応したプログラムを作成する場合は、以下の手順でこの環境変数を設定します。

操作 3: 環境変数“GCC17_COPRO”を選択し、[Edit...]ボタンをクリックします。



[Edit variable]ダイアログボックスが表示されます。



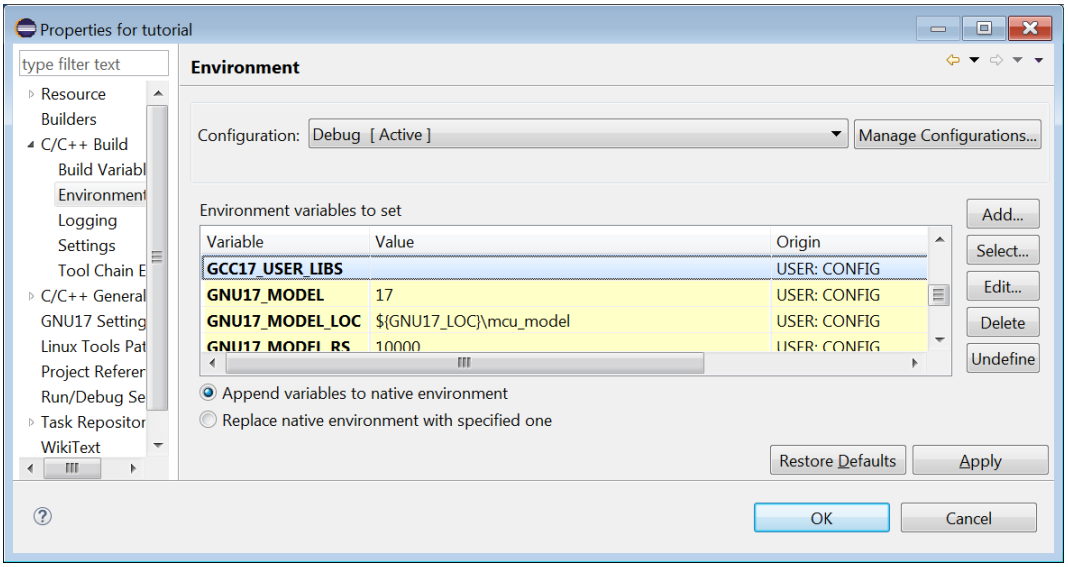
操作 4: 搭載しているコプロセッサを示す以下の文字を[Value:]に入力して、[OK]ボタンをクリックします。

表 2.5.1.1 GCC17_COPRO 設定値

コプロセッサの種類	設定値
なし	空欄(入力なし)
乗算コプロセッサ	¥¥M
COPRO	¥¥MD
COPRO2	¥¥MD2

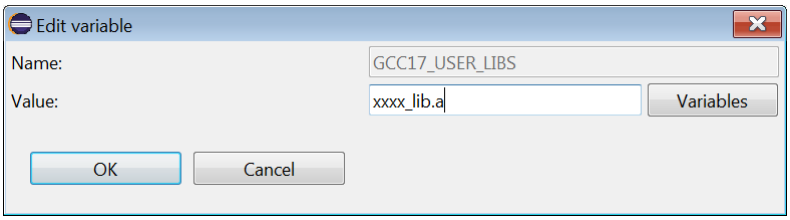
ユーザライブラリの指定(GCC17_USER_LIBS)
リンクするユーザライブラリがある場合は、この環境変数にそのフィル名を登録します。
また、ライブラリファイルは、“(プロジェクト)¥Debug”フォルダにコピーしておく必要があります。

操作 5: 環境変数“GCC17_USER_LIBS”を選択し、[Edit...]ボタンをクリックします。



[Edit variable]ダイアログボックスが表示されます。

操作 6: ライブラリファイル名を[Value:]に入力して、[OK]ボタンをクリックします。



複数のファイルはセミコロンで区切って指定します。

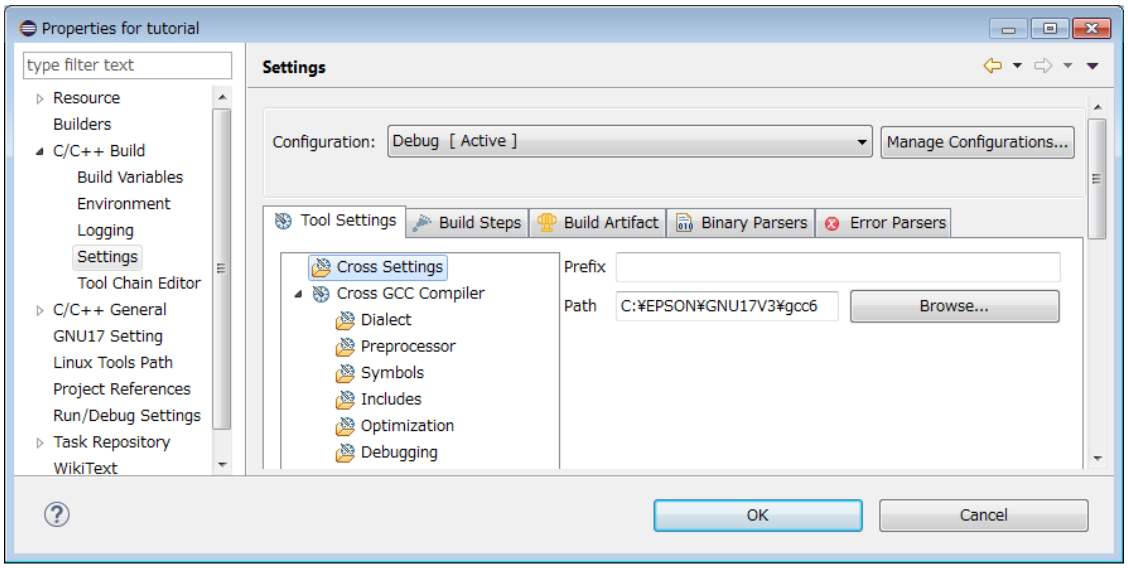
2.5.2 ツールオプションの指定

ビルドツール(C コンパイラ、アセンブラ、リンカ)には標準的なコマンドオプションが設定されていますが、必要に応じて追加や修正が可能です。ここでは、それらの設定ページを見ておきます。
なお、チュートリアルではコンパイラの最適化オプションを除き、設定の変更は不要です。
各ツールのコマンドオプションの詳細は、“S5U1C17001C マニュアル”の 3.4.3 節～3.4.6 節を参照してください。

ツールオプション設定用のページを表示させるには

操作 7: [Project Explorer]ビューの“tutorial”を選択した状態で、[Project]メニュー (または右クリックで表示されるコンテキストメニュー)から[Properties]を選択します。
[Properties]ダイアログボックスが開きます。

操作 8: プロパティのリストから[C/C++ Build] > [Setting]を選択し、[Tool Settings]タブのページを表示させます。



コンパイラパスの設定

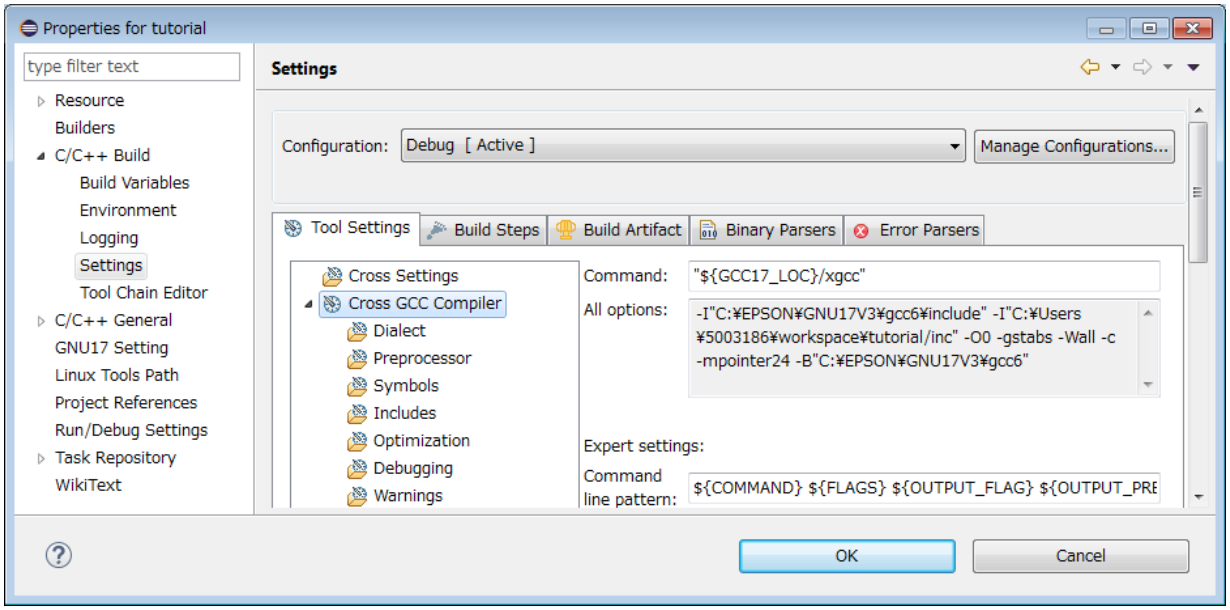
使用する C コンパイラおよびその他のツールが存在するディレクトリを設定しておくことができます。通常は、基本設定の[GCC Version]の選択に従って正しく設定されますので、ここで変更する必要はありません。インストール後にツールのディレクトリを変更した場合などは、次のように設定し直してください。

操作 9: [Tool Settings]ページの設定リストから[Cross Setting]を選択します。

操作 10: 上記画面のように、[Path]にツールのディレクトリをフルパスで入力するか、[Browse...]ボタンで選択します。

コンパイラオプションの設定

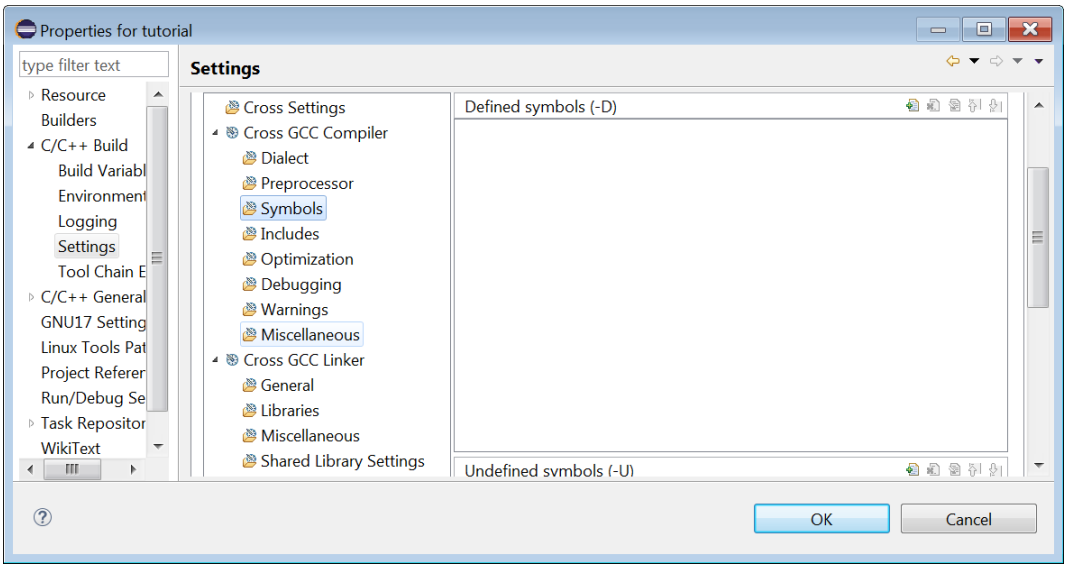
操作 11: [Tool Settings]ページの設定リストから[Cross GCC Compiler]を選択します。



現在設定されている C コンパイラのコマンドオプションが、[All options:]に表示されます。
これを変更するには、以下のように設定リストからオプションを選択してオプション変更ページに移動します。

マクロ定義オプション(-D)

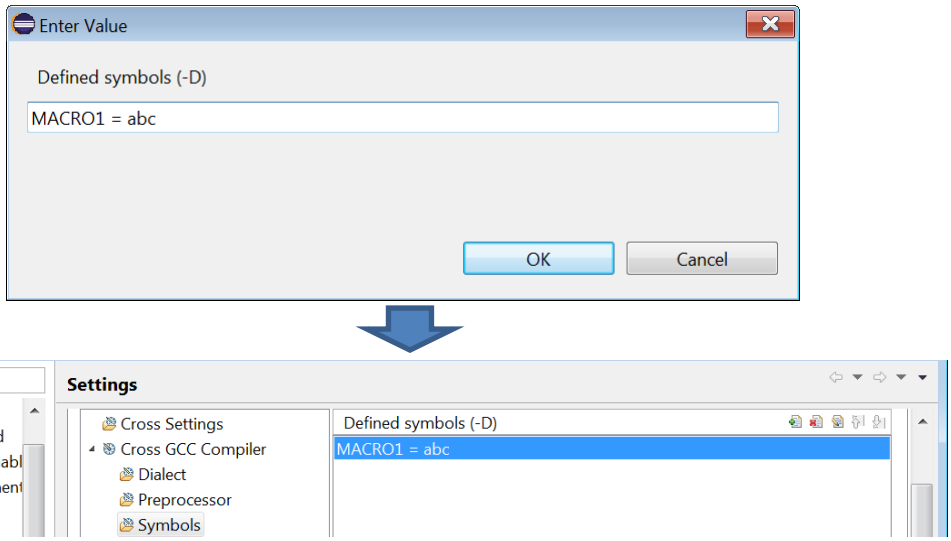
操作 12: [Cross GCC Compiler]下のリストから[Symbols]を選択します。



このページでは、マクロ名を定義(-D オプションを指定)できます。

操作 13: (Add...)ボタンをクリックして[Enter Value]ダイアログボックスを表示させ、定義するマクロ名を次の形式で[Defined symbol (-D)]に入力します(“-D”は入力不要)。入力後、[OK]ボタンをクリックします。

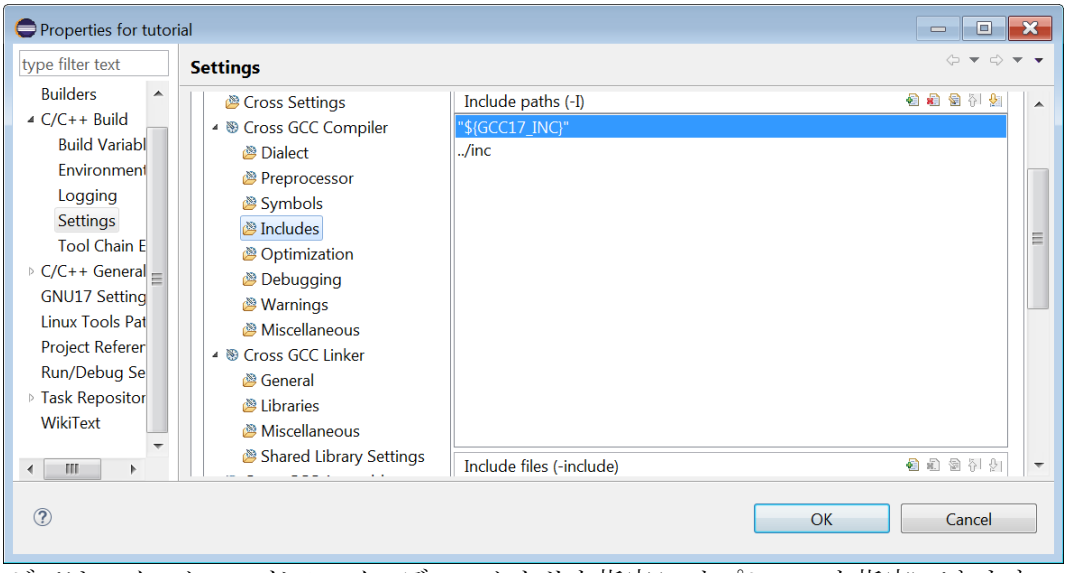
1. <マクロ名>　(“<マクロ名>＝1”として定義されます。)
2. <マクロ名>＝<置き換え文字>



複数のマクロ名を定義する場合は、マクロの数だけ操作 13 を繰り返してください。初期状態では、マクロ名は定義されていません。

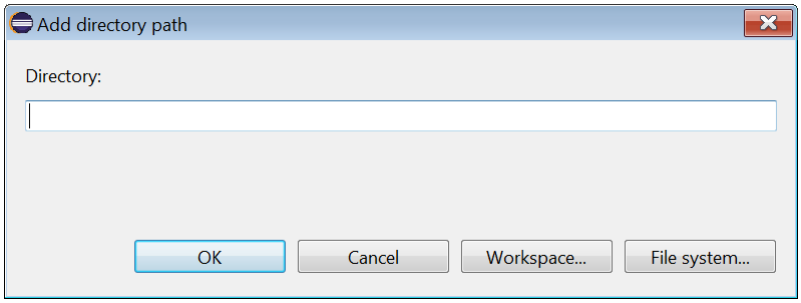
インクルードファイルディレクトリ指定オプション(-I)

操作 14: [Cross GCC Compiler]下のリストから[Includes]を選択します。



このページでは、インクルードファイルディレクトリを指定(-I オプションを指定)できます。

操作 15: (Add...)ボタンをクリックして[Add directory path]ダイアログボックスを表示させます。



操作 16: ワークスペース上のディレクトリは[Workspace...]ボタンを、それ以外のディレクトリは[File system...]ボタンをクリックしてフォルダ選択ダイアログボックスを開き、追加するインクルードフォルダを選択します。

初期状態では、“${GCC17_INC}”(C:¥GNU17V3¥gcc6¥include)と“(プロジェクト)¥inc”ディレクトリが指定されています。

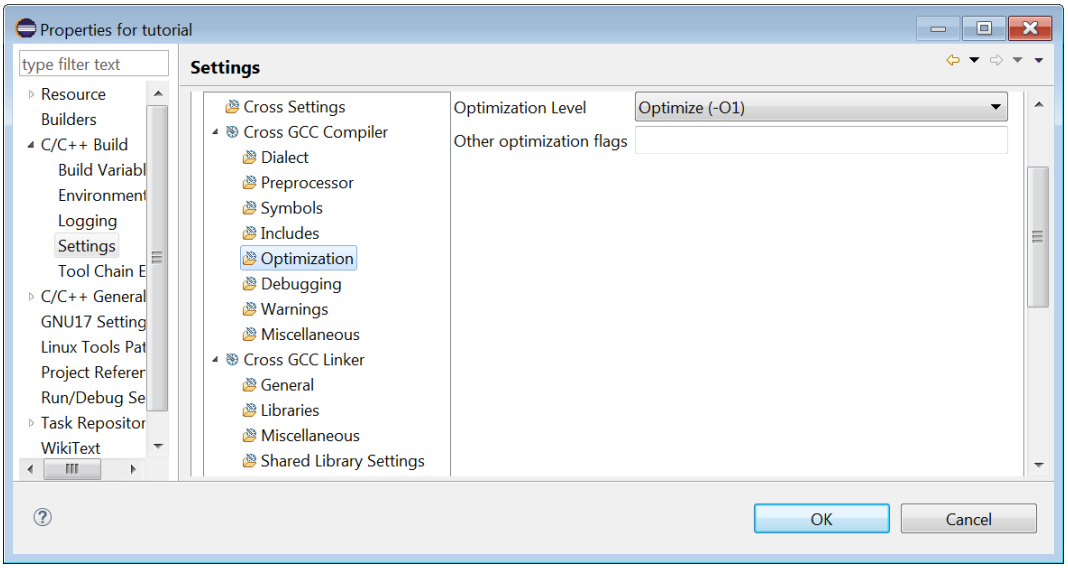
パスを指定するマクロと環境変数について

デフォルト設定の“${GCC17_INC}”は ANSI C ライブラリディレクトリへのパスを表すマクロです。GCC17_INC は[Environment]のページで“${GCC17_LOC}/include”と定義されている環境変数です。同様に、GCC17_LOC には 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ます。

例: GNU17 ツールを C:¥EPSON¥GNU17V3 ディレクトリにインストールした場合
GCC17_LOC = C:¥EPSON¥GNU17V3¥gcc6
GCC17_INC = C:¥EPSON¥GNU17V3¥gcc6¥include

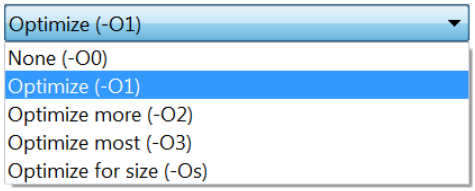
最適化オプション(-O)

操作 17: [Cross GCC Compiler]下のリストから[Optimization]を選択します。



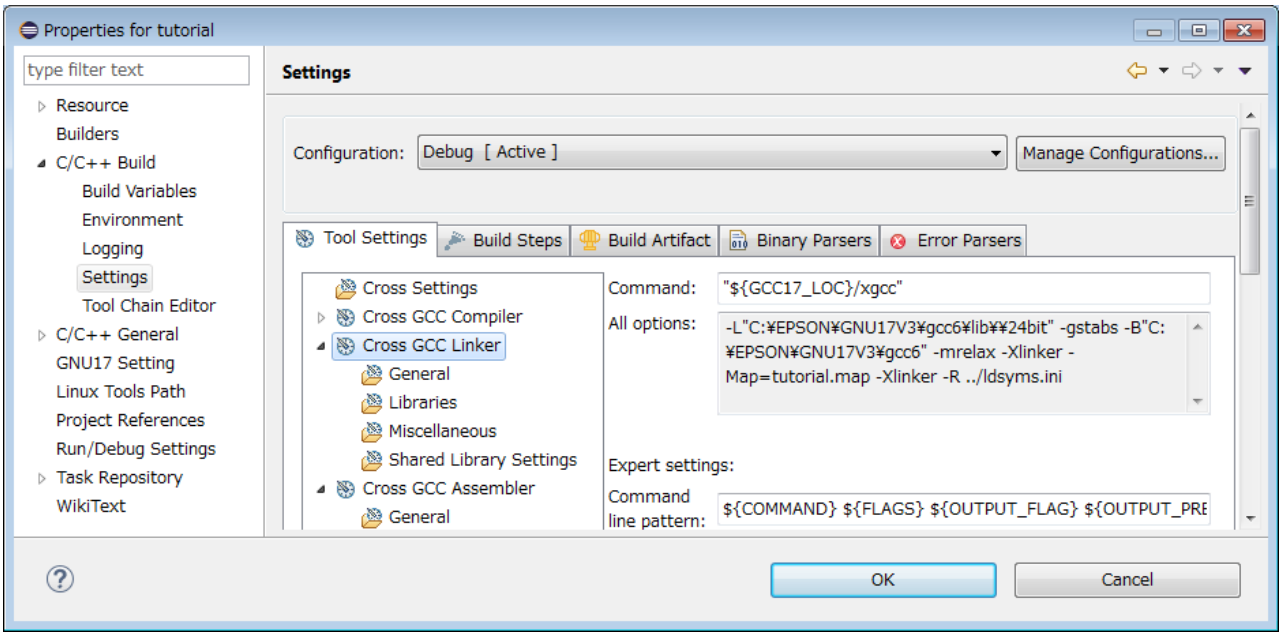
このページでは、最適化レベルが指定できます。初期状態では、-O1 が指定されています。

操作 18: [Optimization Level]ドロップダウンリストから最適化レベルを選択します。本チュートリアルでは、デバッグを容易にするため、-O0(最適化なし)を選択してください。



リンカオプションの設定

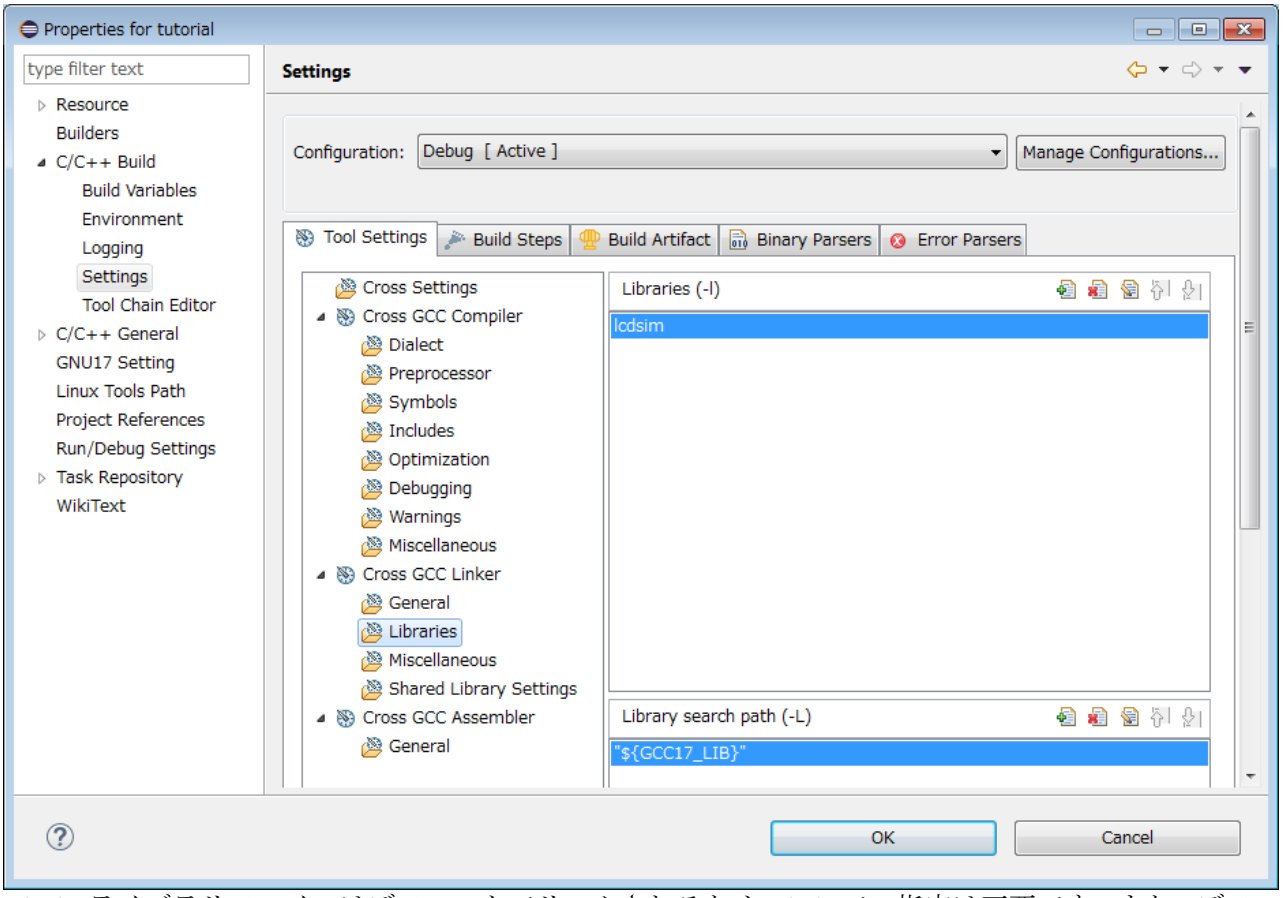
操作 19: [Tool Settings]ページの設定リストから[Cross GCC Linker]を選択します。



現在設定されているリンカのコマンドオプションが、[All options:]に表示されます。
これを変更するには、以下のように設定リストからオプションを選択してオプション変更ページに移動します。

ライブラリの指定

操作 20: [Cross GCC Linker]下のリストから[Libraries]を選択します。



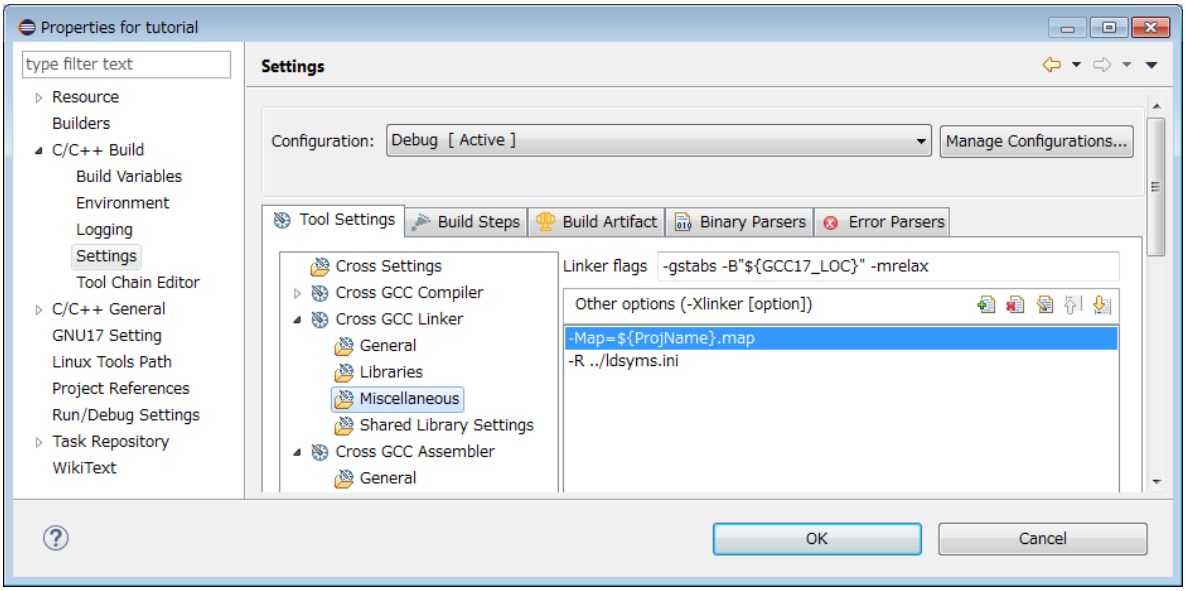
S1C17 ライブラリファイルはデフォルトでリンクされるため、ここでの指定は不要です。また、デフォルトで LCD パネルシミュレータライブラリ(lcdsim)がリンクされています。ユーザライブラリは環境変数として設定してください。

操作 21: (Add...)ボタンをクリックして[Enter Value]ダイアログボックスを表示させ、ライブラリ名(libxxx.a の xxx の部分のみ)を[Libraries (-l)]に入力します。入力後、[OK]ボタンをクリックします。

[Library search path (-L)]には、ライブラリを検索するパスを設定します。あらかじめ、${GCC17_ LIB}というマクロで ANSI C ライブラリとエミュレーションライブラリのディレクトリが登録されていますので、通常は変更の必要はありません。

その他のリンカコマンドオプションの指定

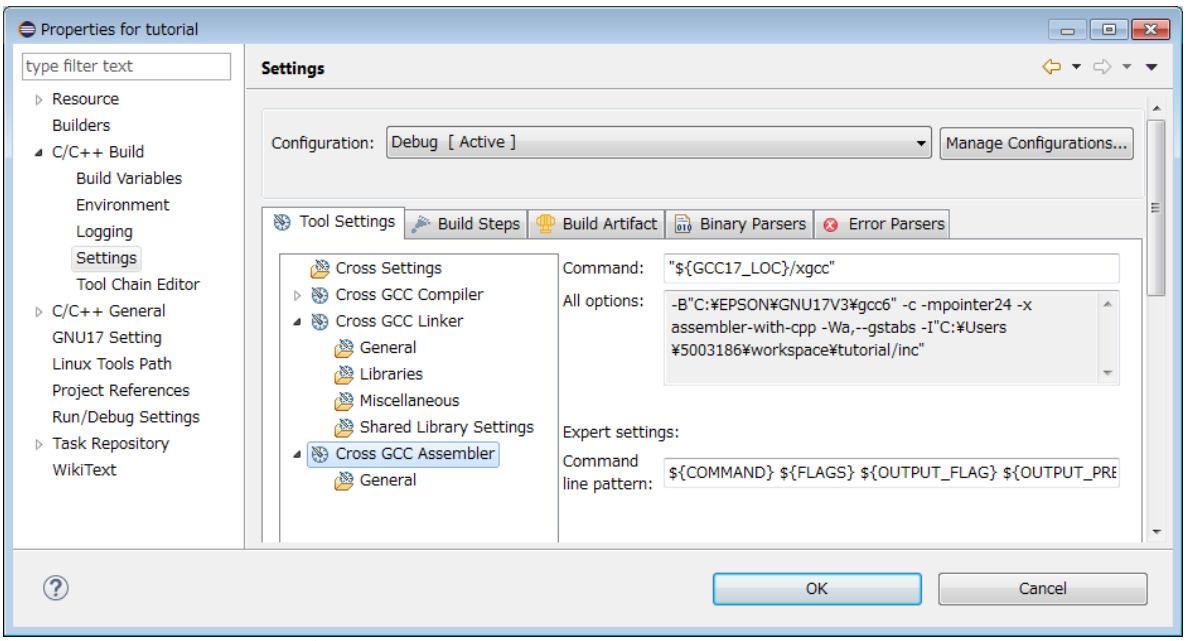
操作 22: [Cross GCC Linker]下のリストから[Miscellaneous]を選択します。



ここで、リンカのコマンドオプションが指定できます。あらかじめ、マップファイル((プロジェクト名).map)を生成するオプションが登録されています。

アセンブラオプションの設定

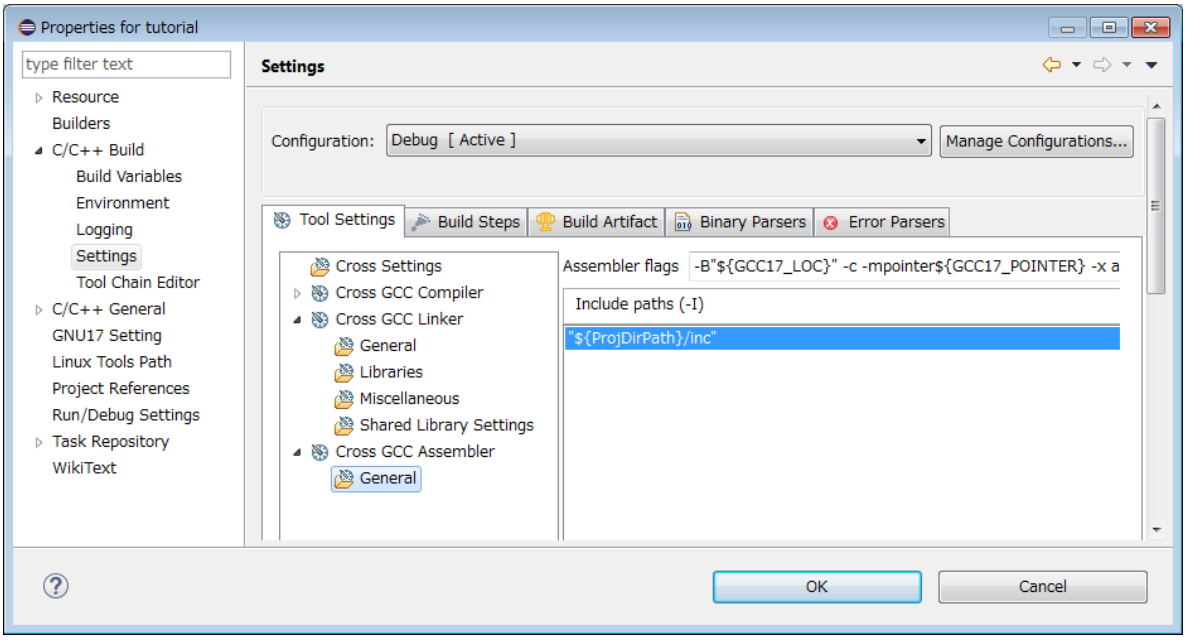
操作 23: [Tool Settings]ページの設定リストから[Cross GCC Assembler]を選択します。



現在設定されているアセンブラのコマンドオプション*が、[All options:]に表示されます。
これを変更するには、以下のように設定リストからオプションを選択してオプション変更ページに移動します。

*IDE はアセンブラソースを、“-c -xassembler-with-cpp”オプションを指定し、C プリプロセッサを通してアセンブルします。

操作 24: [Cross GCC Assembler]の[General]を選択します。



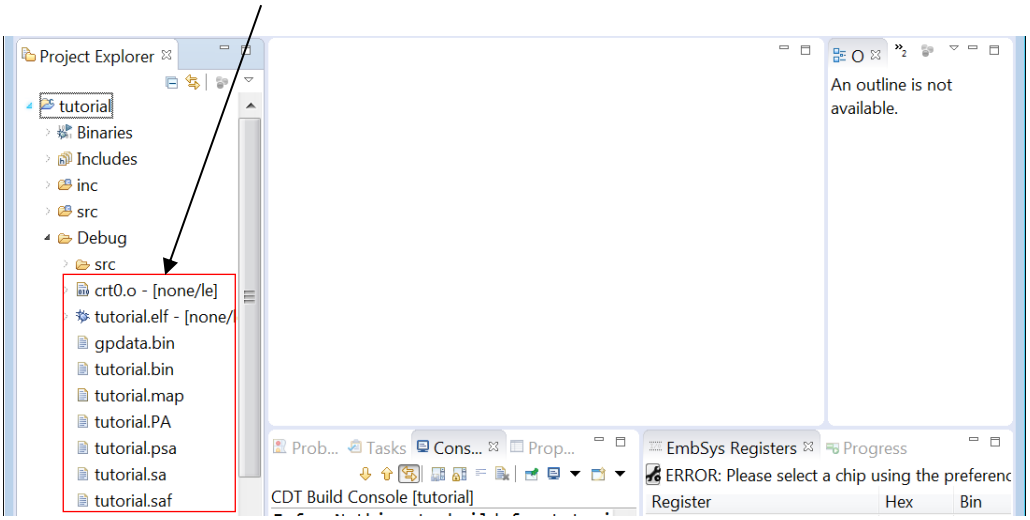
操作 25: [Assembler flags]に必要なオプションを入力し、[OK]ボタンをクリックします。
以上で、ツールオプションの設定が完了します。

2.6 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 1: [Project Explorer]ビューの“tutorial”を選択した状態で、[Project]メニュー (または右クリックで表示されるコンテキストメニュー)から、[Build Project]を選択します。

この操作によりビルド処理に加え、リンカ以降のツールも実行され、実行形式のオブジェクトファイル“tutorial.elf”や提出用の“tutorial.PA”など、必要なファイルがすべて生成されます。



*1 [Project]メニューから[Build All]を選択して、ワークスペース内のすべてのプロジェクトを一括してビルドすることもできます。

*2 一旦ビルドを実行した後に、ソースやヘッダファイルを変更せずに再度ビルドを実行しても、オブジェクトファイルなどの出力ファイルは作り直されません。この場合に、出力ファイルを再生成させるには、オブジェクトファイルを消去するクリーンという処理が必要です。詳細は、“S5U1C17001C マニュアル”の“3.5.3 クリーンとリビルド”を参照してください。

2.7 デバッグ

ビルドにより“tutorial.elf”が生成されると、そのファイルをデバッガに読み込ませてプログラムのデバッグが行えます。以下、その基本的な操作方法を見ていきます。

2.7.1 デバッグ環境(ICDmini モードとシミュレータモード)

IDE は 2 種類のデバッグ環境(ICDmini モードとシミュレータモード)に対応しています。

ICDmini モード

ICDmini モードでは、PC 上のデバッガに ICDmini(S5U1C17001H*)を介して実際の MCU を搭載したターゲットシステムを接続し、デバッグを行います。ターゲットプログラムはターゲットシステム上の MCU によって実行されますので、ソフトウェアのデバッグに加え、ハードウェアの動作チェックも同時に行えます。

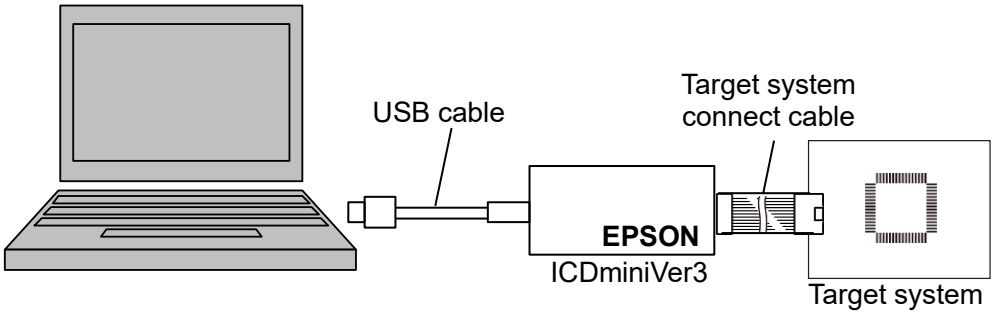


図 2.7.1.1 ICDminiVer3 を使用するデバッグシステム例

注: ICDmini モードでデバッグを行う場合、ターゲット CPU(プロジェクトの基本設定)を正しく選択しておく必要があります。

シミュレータモード

シミュレータモードは、PC 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、ICDmini に依存した機能は使用できません。

本チュートリアルでは、シミュレータモードでのデバッグ操作を説明します。デバッグ環境の詳細は、“S5U1C17001C マニュアル”を参照してください。

2.7.2 デバッグの準備(GDB コマンドファイルの選択/編集)

デバッグを開始するには、デバッガの起動時に実行する GDB コマンドファイルを指定しておく必要があります。デバッガのコマンドについては、“S5U1C17001C マニュアル”の“8.5 コマンドリファレンス”を参照してください。

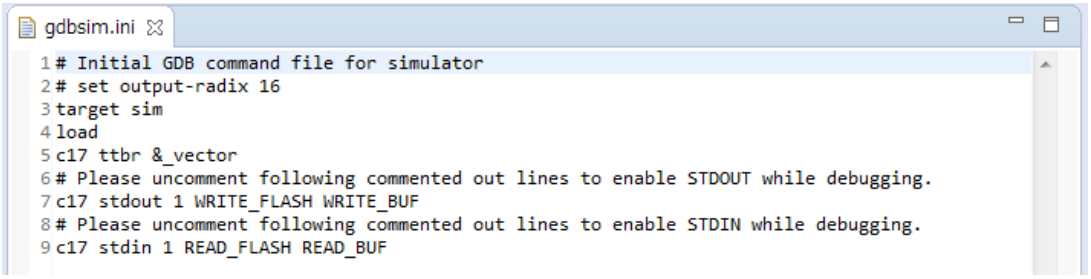
自動生成 GDB コマンドファイル

プロジェクトを新規作成すると、以下の 3 つのデバッガ起動用 GDB コマンドファイルがプロジェクトフォルダ内に生成されます。

gdbmini2.ini	ICDmini モード用 GDB コマンドファイル(ICDmini Ver. 1～2 用)
gdbmini3.ini	ICDmini モード用 GDB コマンドファイル(ICDmini Ver. 3 用)
gdbsim.ini	シミュレータモード用 GDB コマンドファイル

これらのファイルの内容は、エディタに表示させて確認できます。

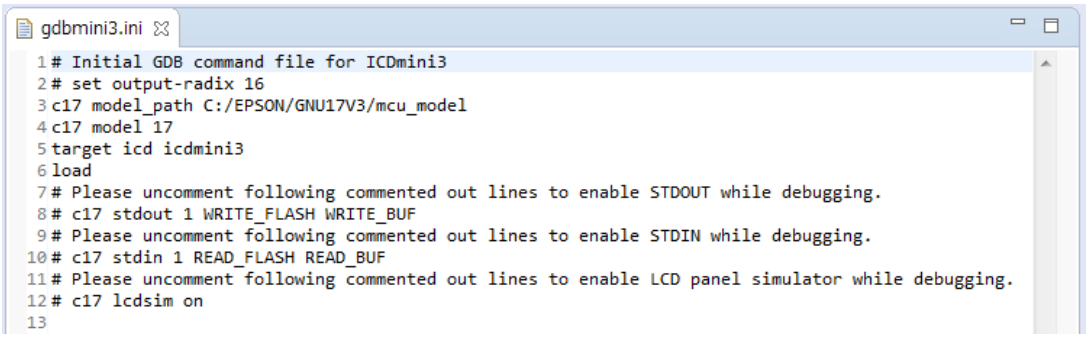
操作 1: [Project Explorer]ビューの“tutorial”プロジェクト内にある“gdbsim.ini”をダブルクリックします。



シミュレータモード用 GDB コマンドファイルの内容がエディタに表示されます。この GDB コマンドファイルにより実行される内容は以下のとおりです。

1. デバッガをシミュレータモードに設定
2. プログラムをターゲット(シミュレータ)のメモリにロード
3. CPU の TTBR レジスタを設定
4. 標準入出力機能(stdout/stdin)をイネーブル

操作 2: [Project Explorer]ビューの“tutorial”プロジェクト内にある“gdbmini3.ini”をダブルクリックします。



ICDmini モード用 GDB コマンドファイルの内容がエディタに表示されます。この GDB コマンドファイルにより実行される内容は以下のとおりです。

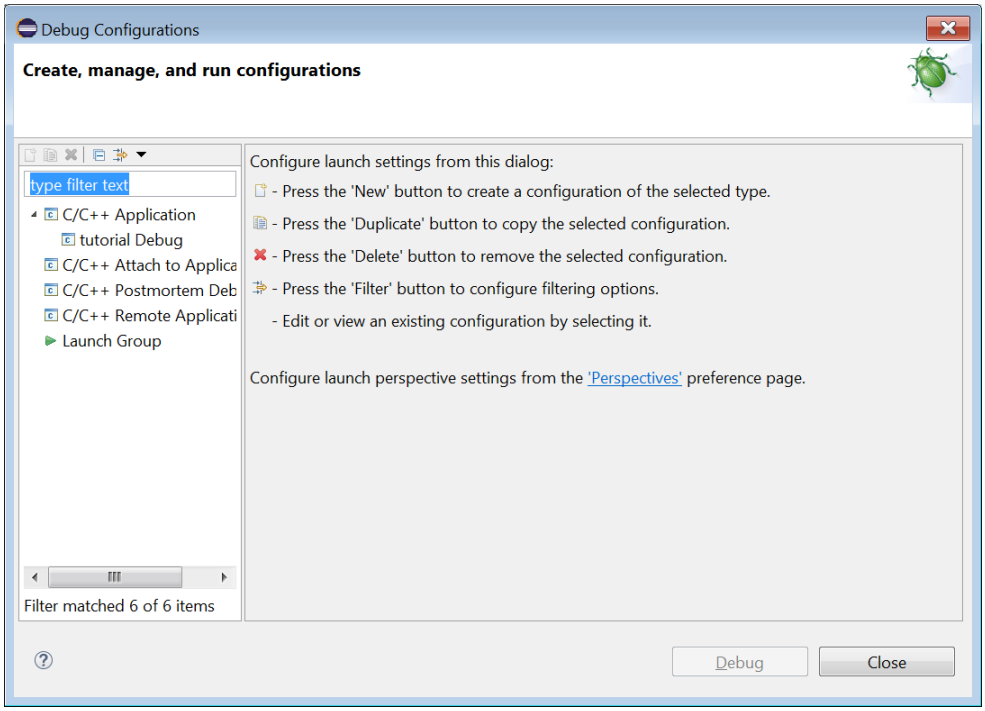
1. ターゲット(MCU)の指定
2. デバッガを ICDmini モードに設定
3. プログラムをターゲットのメモリにロード

GDB コマンドファイルの選択

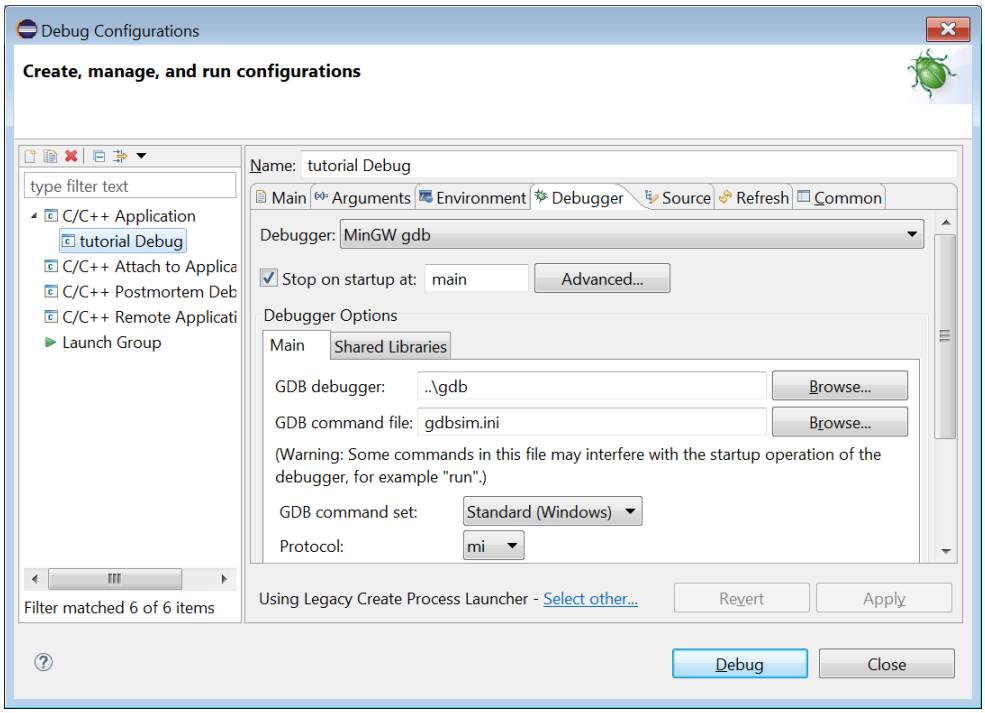
チュートリアルでは“gdbsim.ini”を使用してデバッグを行います。デバッガの起動時にこの GDB コマンドファイルを実行させるには、あらかじめ次のように設定しておきます。

操作 3: [Project Explorer]ビューの“tutorial”を選択し、ツールバー (Debug)ドロップダウンリスト*から[Debug Configurations...]を選択します。
*その他、[Run]メニューからも[Debug Configurations...]を選択可能

[Debug Configurations]ダイアログボックスが開きます。



操作 4: リストから[C/C++ Application] > [tutorial Debug]を選択し、[Debugger]タブのページを表示させます。



操作 5: 使用する GDB コマンドファイルを[Browse...]ボタンで選択し、[GDB command file:]に設定します。このチュートリアルでは、“gdbsim.ini”のままにしておきます。
ユーザが作成した GDB コマンドファイルも、ここで選択可能です。

GDB コマンドファイルの編集

シミュレータモードの場合、特に“gdbsim.ini”を変更する必要はありません。ICDmini モードの場合は、必要に応じて以下のアンダーライン部をエディタで変更してください。

```
# Initial GDB command file for ICDmini3
# set output-radix 16
c17 model_path C:/EPSON/GNU17V3/mcu_model          …(1)
c17 model 17xxx                                    …(2)
target icd icdmini3                                …(3)
load
# Please uncomment following commented out lines to enable STDOUT while debugging.
# c17 stdout 1 WRITE_FLASH WRITE_BUF               …(4)
# Please uncomment following commented out lines to enable STDIN while debugging.
# c17 stdin 1 READ_FLASH READ_BUF
# Please uncomment following commented out lines to enable LCD panel simulator while debugging.
# c17 lcdsim on                                    …(5)
```

(1) c17 model_path

機種別情報ファイルが格納されているディレクトリへのパスを指定します。インストール時のままであれば、変更する必要はありません。

(2) c17 model

ターゲット機種名を指定します。IDE は、プロジェクト新規作成時またはプロジェクトの基本設定により指定されている機種名をここに書き込んで GDB コマンドファイルを作成します。通常、変更する必要はありませんが、デバッグに必要な場合は変更可能です。

なお、ICD の TARGET VCC IN 端子が未接続の場合、機種名に続けて@NOVCCIN と Detail オプションを指定してください。指定可能な Detail オプションは機種別情報ファイルの付属文書(fls/fls17*_readme.txt)で確認できます。

シミュレータモードでデバッグする場合にターゲット機種名が指定され、それが対応機種であれば、デバッガと共に周辺回路シミュレータ(ES-Sim17)も起動します。

(3) target

デバッグモードを指定します。ICDmini モードを指定した場合は、ターゲットシステムと接続まで行います。デバッグ環境に合った自動生成 GDB コマンドファイルを使用することで適切に設定されますので、通常は変更する必要はありません。

target sim	シミュレータモード gdbsim.ini に記述されています。
target icd icdmini3	ICDmini モード(ICDmini Ver. 3 を使用する場合) gdbmini3.ini に記述されています。
target icd icdmini2	ICDmini モード(ICDmini Ver. 1.0/1.1/2.0 を使用する場合) gdbmini2.ini に記述されています。

(4) c17 stdoutおよびc17 stdin

これらの行のコメントを解除(#を削除)することで、プログラム中の stdout への出力を IDE の[Console]ビューに表示させることができます。また、専用の入力ウィンドウからプログラム中の stdin へ入力ができます。

(5) c17 lcdsim

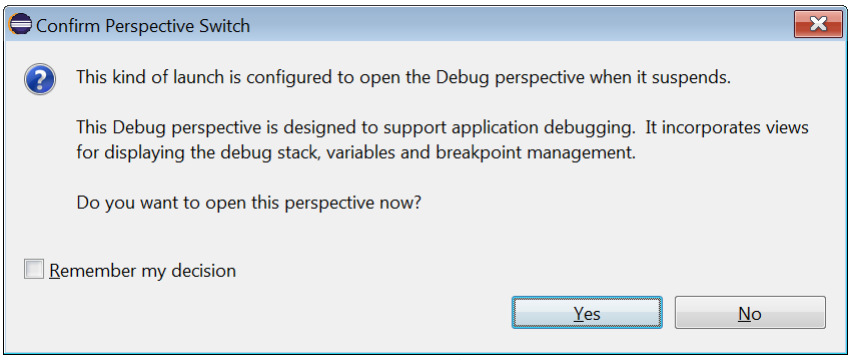
この行のコメントを解除(#を削除)することで、LCD パネルシミュレータ機能を使用することができます。

デバッガの起動時に実行させたいコマンドがあれば、選択した GDB コマンドファイルに追加しておきます。

2.7.3 デバッガの起動

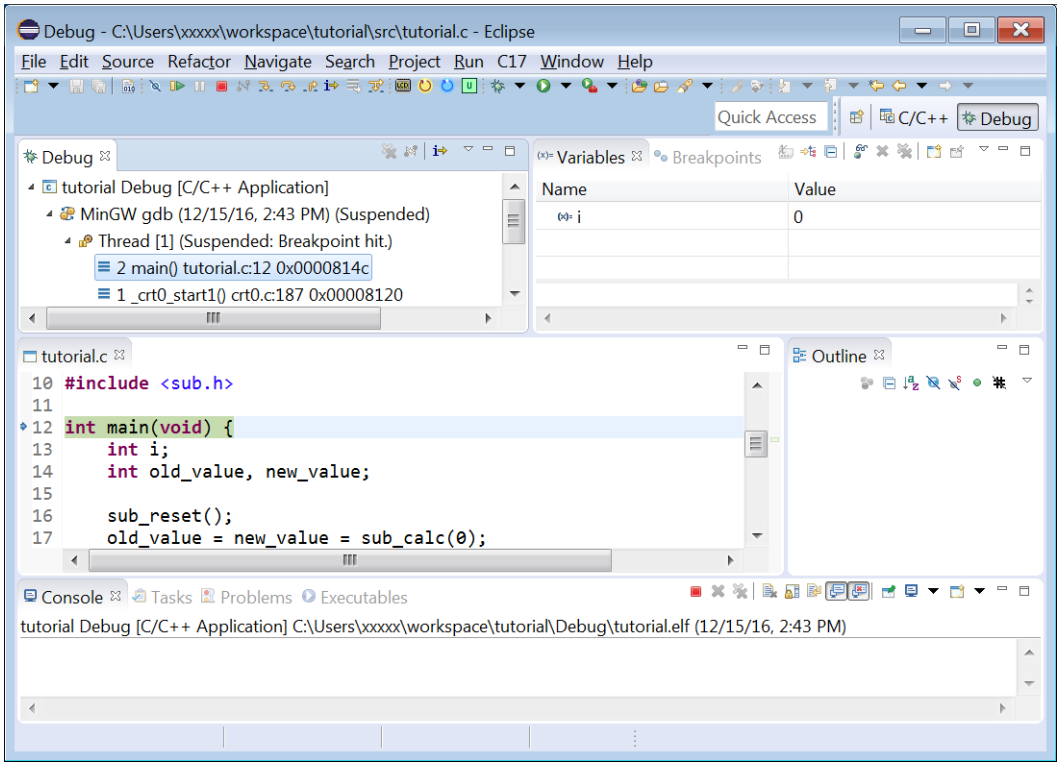
デバッガを起動するには

操作 6: [Debug Configurations]ダイアログボックスのリストから[C/C++ Application] > [tutorial Debug]を選択し、[Debug]ボタンをクリックします。[Confirm Perspective Switch]ダイアログボックスが表示された場合は[Yes]ボタンをクリックします。



*このダイアログボックスは、IDE のパースペクティブ(ビューの構成)が[C/C++]から[Debug]に変更されることを確認するために表示されます。[Remember my decision]チェックボックスを選択しておくことで、次回からは表示されなくなります。

デバッガが起動し、ウィンドウが[Debug]パースペクティブに切り換わります。



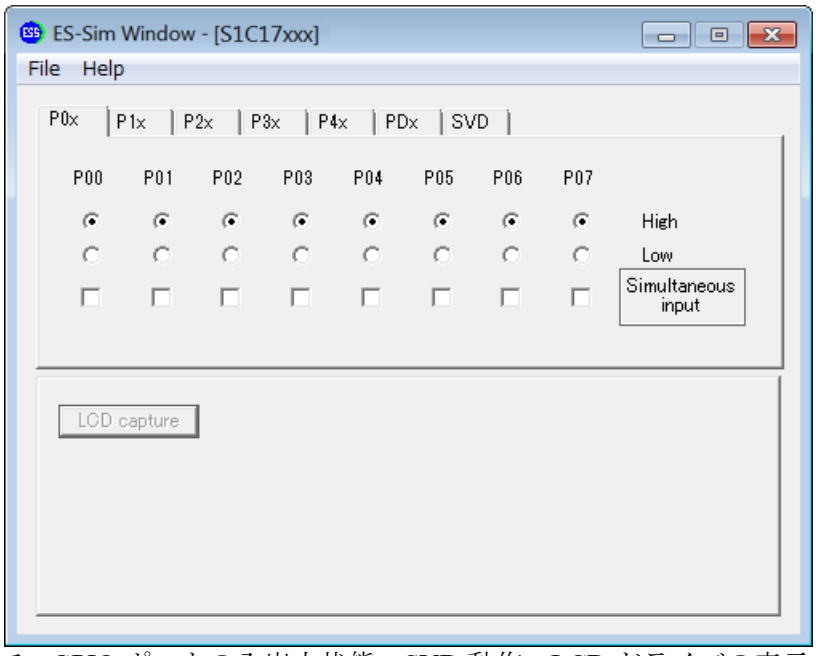
GDB コマンドファイルによってオブジェクトファイル“tutorial.elf”がロードされ、ブート処理を実行後、main 関数の位置(関数の先頭)でブレークしています。
[Debug]ビューの“2 main() tutorial.c: 12 0x0000814c”は、ブレーク位置を示しています。
エディタの“tutorial.c”では、12 行目が緑でハイライト表示され、停止位置を示しています。

*main 関数の位置でブレークしたのは、[Debug Configurations]ダイアログボックスの[Debug]タブのページに次の指定があるためです。

このチェックを外すと、プログラムは起動するとブレークせずに動作し続けます。また、main 関数以外を指定することもできます。

周辺回路シミュレータ

周辺回路シミュレータに対応したターゲット CPU を選択し、シミュレータモードでデバッガを起動すると、周辺回路シミュレータ(ES-Sim17)も起動します。



このウィンドウ内で、GPIO ポートの入出力状態、SVD 動作、LCD ドライバの表示をシミュレート可能です。詳細は、S5U1C17001C マニュアルの“周辺回路シミュレータ(ES-Sim17)”を参照してください。周辺回路シミュレータを起動可能な機種は、機種情報フォルダ(GNU17V3/mcu_model/17xxx)内に CPU 構成ファイル“essim17”が含まれています。

2.7.4 デバッガのツールバーボタン概要

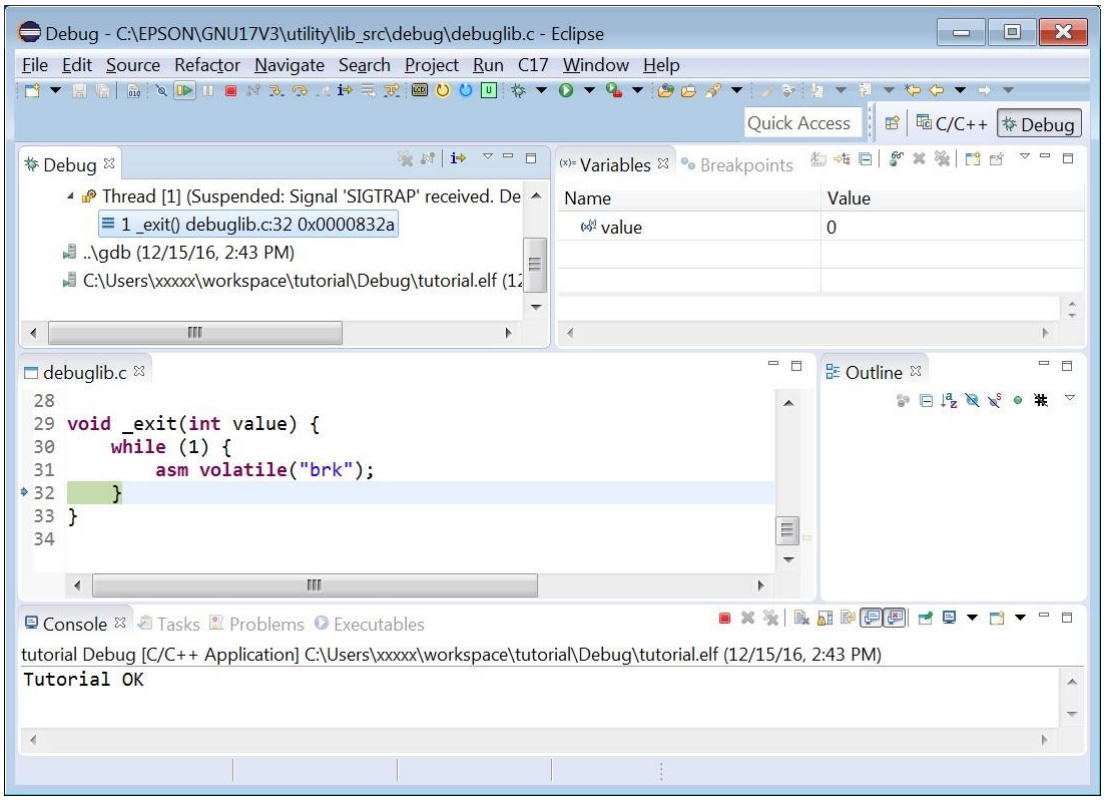
表 2.7.4.1 デバッガのツールバーに表示されているボタン

アイコン	名前	機能
	[Skip All Breakpoints]	設定したブレークポイントを無視します。
	[Resume]	プログラムを実行します。
	[Suspend]	実行中のプログラムを一時停止します。デバッグモード内では、MCU 内部情報の参照、編集、各種設定が可能です。
	[Terminate]	GDB デバッガを終了します
	[Step Into]	プログラムを 1 命令実行します。
	[Step Over]	サブ関数も 1 命令として、実行します。
	[Step Return]	現在の関数を抜けるまでプログラムを実行します。
	[Instraction Stepping Mode]	アクティブにすると、[Step *]がアセンブラ命令単位になります。
	[Launch LCDUtility]	LCDUtil17 ウィンドウを開きます。
	[Reset]	プログラムの先頭に移動し、汎用レジスタを初期化します。
	[Reset Target]	ハード的にリセットします。
	[User Commnad]	ユーザが定義したデバッグコマンドを実行します。 定義は、”userdefine.gdb”ファイルで行います。
	[Debug]	デバッガを起動します。

2.7.5 プログラムの実行

プログラムを実行させるには

操作 7: ツールバー (Resume)ボタンをクリック(または[Run]メニュー ＞[Resume]を選択)します。サンプルプログラムは現在の位置から実行を開始します。1 段から 9 段までの四角錐数(球で四角錐を構成するのに必要な球の数)を計算し、最後に[Console]ビューに“Tutorial OK!”を表示して終了します。



プログラムは main 関数の実行終了後にコールされた_exit 関数中の brk 命令で停止しているため、エディタにはそのソースと停止位置が表示されます。

実行中のプログラムを停止させるには

実際のソフトウェア開発時に実行中のプログラムを停止させるには、次のように操作してください。

操作 8: 再開可能な状態で一次停止させるには、ツールバー (Suspend)ボタンをクリック(または[Run]メニュー ＞[Suspend]を選択)します。

この停止位置から実行を再開するには (Resume)ボタンをクリックします。

操作 9: 現在のデバッグセッションを終了するには、ツールバー (Terminate)ボタンをクリック(または[Run]メニュー ＞ [Terminate]を選択)します。

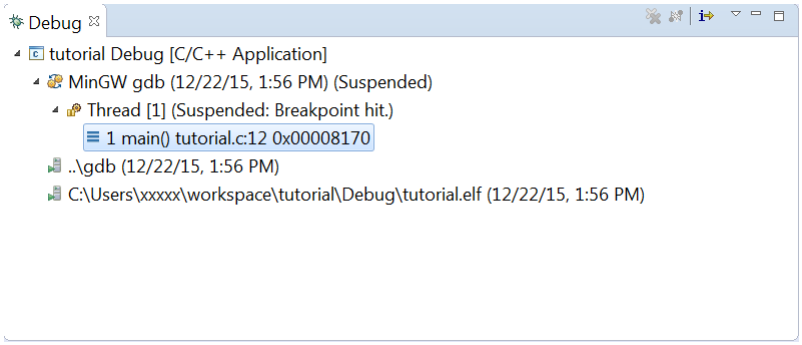
この状態から再度デバッグを始めるには (Debug)ボタンをクリック(または[Run]メニュー ＞[Debug]を選択)します。

2.7.6 デバッガのビュー

ここで、デバッグで使用する主なビューを見ておきます。

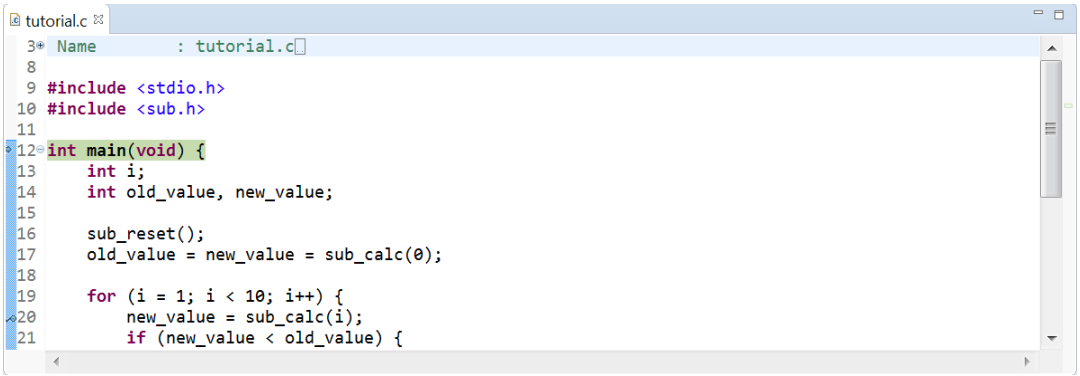
操作 10: 表示されていないビューは、[Window]メニュー ＞[Show View]サブメニューから選択して表示させることができます。

[Debug]ビュー



このビューにはデバッグ情報が表示されます。
上の例では、デバッガ gdb に“tutorial.elf”がロードされており、実行がブレークポイントのヒットにより、“tutorial.c”の main 関数の位置(アドレス 0x8170)で中断していることを示しています。

エディタビュー



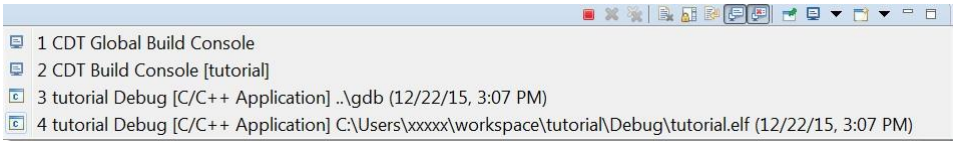
現在実行中あるいはブレーク中のソースを表示します。ブレーク中の現在位置は緑でハイライト表示されます。
左端のマーカーバーに表示されている矢印は現在のプログラムカウンタが指し示している行、はその行にブレークポイントが設定されていることを示します。

[Console]ビュー

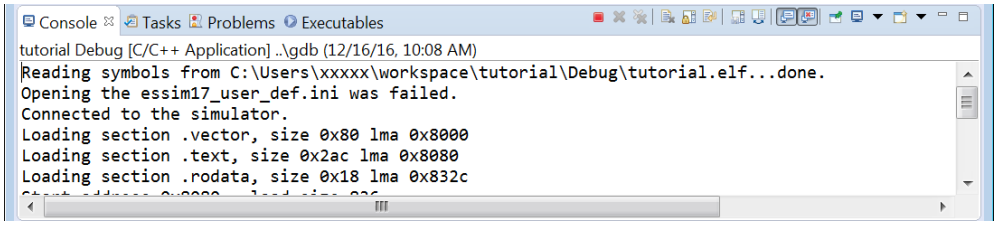
デバッガのメッセージを表示したり、ターゲットプログラムの標準入出力として使用します。

◆ デバッガのメッセージを表示させるには

操作 11: (Display Selected Console)ドロップダウンリストから、“tutorial Debug [C/C++ Application] ..¥gdb”を選択します。



デバッガのメッセージ表示例

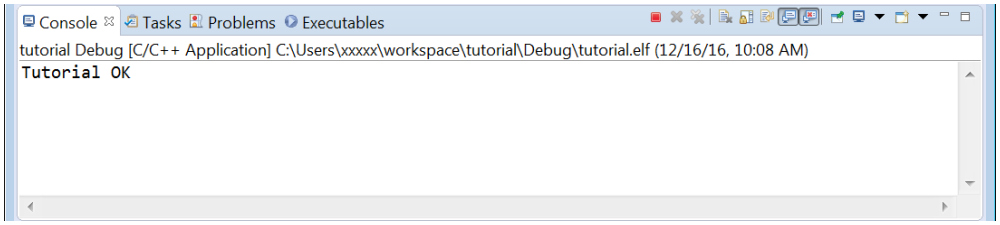


*デバッガのコンソールでは、デバッグコマンドを入力して実行させることもできます。(gdb)のプロンプトは表示されませんので、最終行に実行するコマンドを入力し、[Enter]キーを押してください。

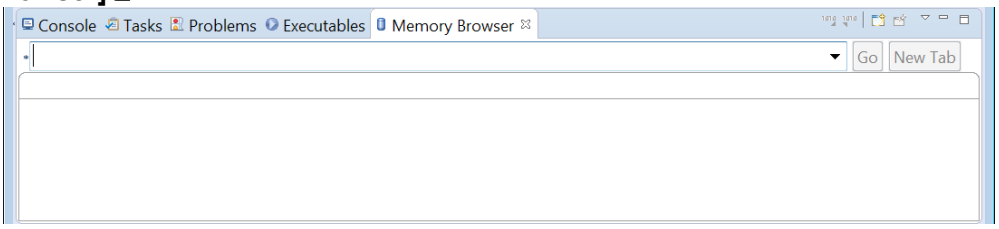
◆ ターゲットプログラムの入出力に使用するには

操作 12: (Display Selected Console)ドロップダウンリストから、“tutorial Debug [C/C++ Application] C:¥ ... ¥tutorial¥Debug¥tutorial.elf”を選択します。

ターゲットプログラムの出力表示例



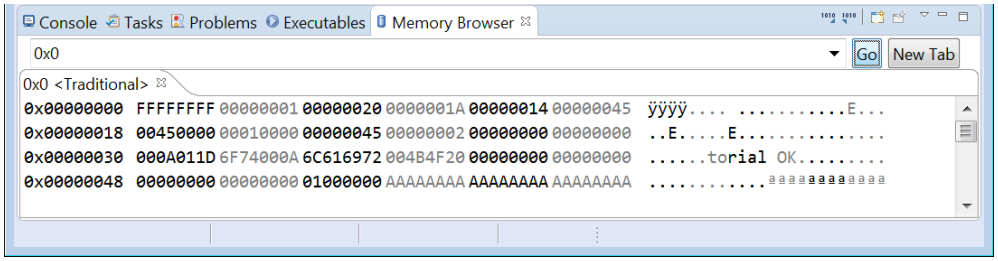
[Memory Browser]ビュー



指定したターゲットメモリの内容を表示します。

◆ 表示するメモリ領域を指定するには

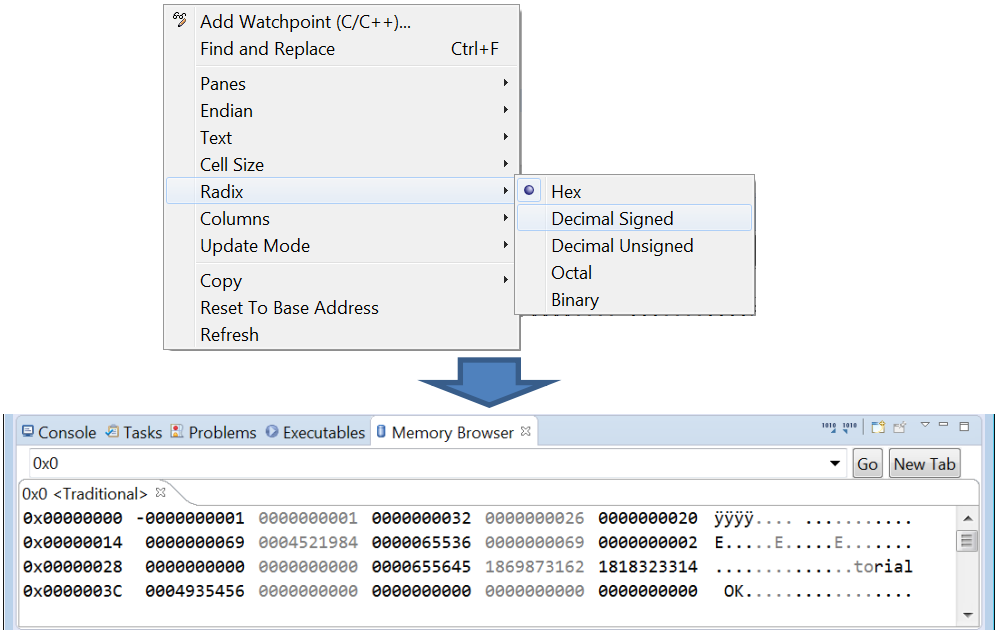
操作 13: テキストボックスにモニタするメモリのアドレスか、変数名を入力して[Go]ボタンをクリックします(または[Enter]キーを押します)。



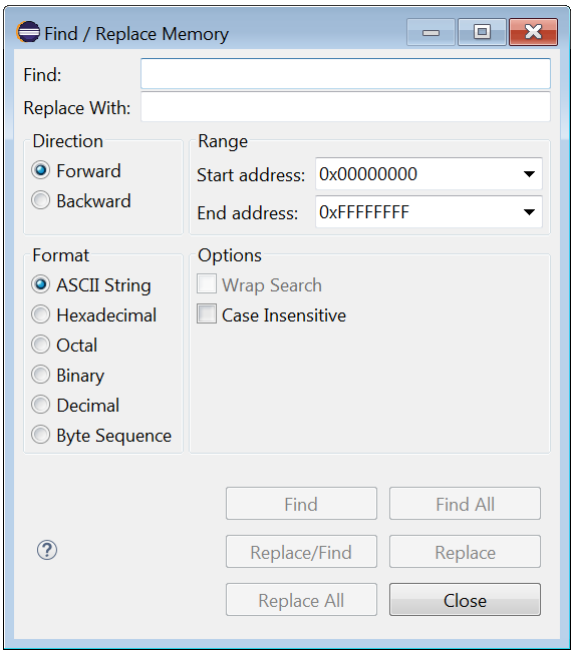
メモリの内容が 16 進ダンプ形式で指定アドレスから表示されます。また、メモリの値をクリックして数値をキー入力することにより、内容を変更することができます。
[New Tab]ボタンをクリックすると、新たなタブページが開きますので、複数の領域を参照する場合に使用します。

◆ 表示形式を変更するには

操作 14: ビュー内を右クリックしてコンテキストメニューを表示させ、所望の表示形式を選択します。



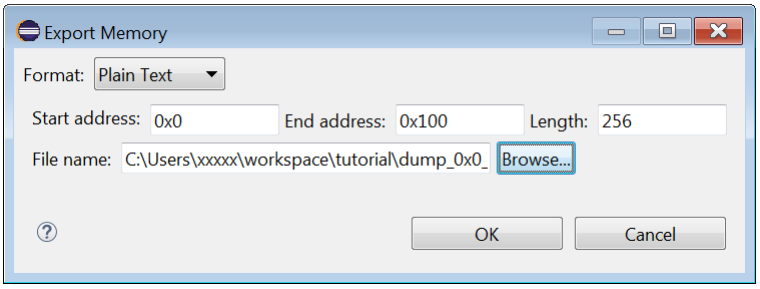
この例では、HEX 表示を符号付き 10 進数に変更しています。このほかにも、テキストのエンコード形式や、列数、各列のバイト数などが選択可能です。
また、[Find and Replace]を選択して、指定の数値/文字列の検索/置き換えも行えます。



◆ メモリデータをファイルに書き出すには

メモリの指定領域をファイルに書き出すことができます。

操作 15: (Export)ボタンをクリックして[Export Memory]ダイアログを表示させます。



操作 16: 出力形式(テキスト、バイナリ、S レコード)、アドレス範囲、ファイル名を設定して、[OK] ボタンをクリックします。

出力例:

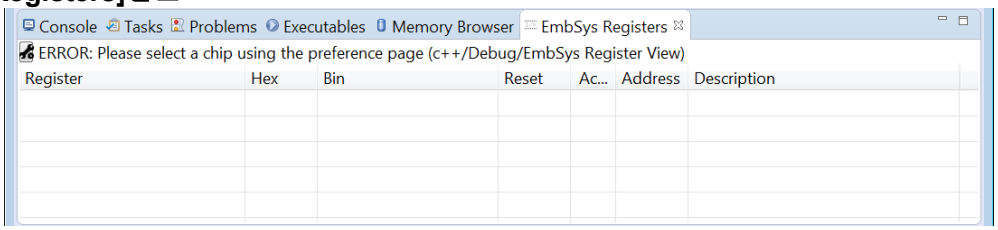
テキスト

```
FFFFFFFF  01000000  20000000  1A000000  14000000
45000000  00004500  00000100  45000000  02000000
00000000  00000000  0A001D01  0A00746F  7269616C
204F4B00  00000000  00000000  00000000  00000000
00000001  AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
:         :         :         :         :
```

S レコード

```
S315000000000FFFFFFFF01000000200000001A000000B3
S3150000001014000000450000000000450000000001003B
S3150000002045000000020000000000000000000000083
S315000000300A001D010A00746F7269616C204F4B0043
S31500000040000000000000000000000000000000000AA
S31500000050000000001AAAAAAAAAAAAAAAAAAAAAAAA1
:       :       :       :       :
```

[EmbSys Registers]ビュー



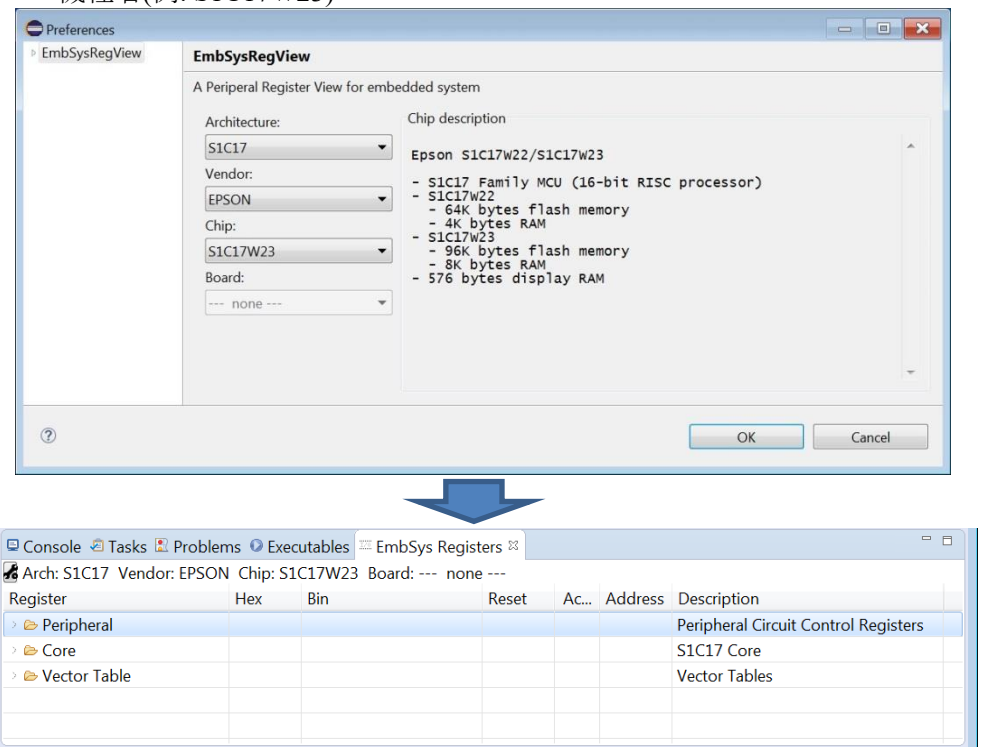
メモリにマッピングされたコア/周辺回路制御レジスタやベクタテーブルの内容を表示します。

◆ 機種を設定するには

このビューに表示をさせるには、機種を選択する必要があります。

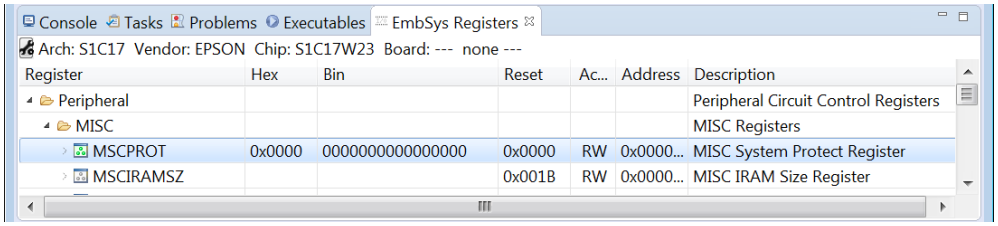
操作 17: ボタンをクリックして[Preferences]ダイアログボックスを表示させ、各項目を以下のように設定します。設定後、[OK]をクリックしてダイアログボックスを閉じます。

Architecture: S1C17
Vendor: EPSON
Chip: 機種名(例: S1C17W23)



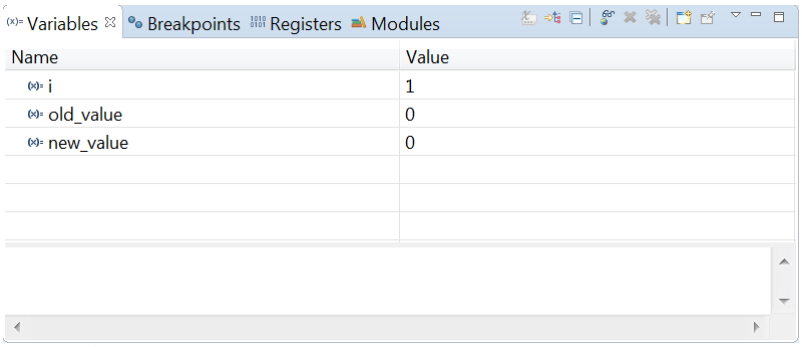
◆ 現在のレジスタ値を表示させるには

操作 18: [Peripheral] > [(周辺回路)]を展開して確認するレジスタを表示させます。レジスタ名の左のアイコンをクリックするとレジスタ値の表示が有効となり、現在の値が 16 進数および 2 進数で表示されます。



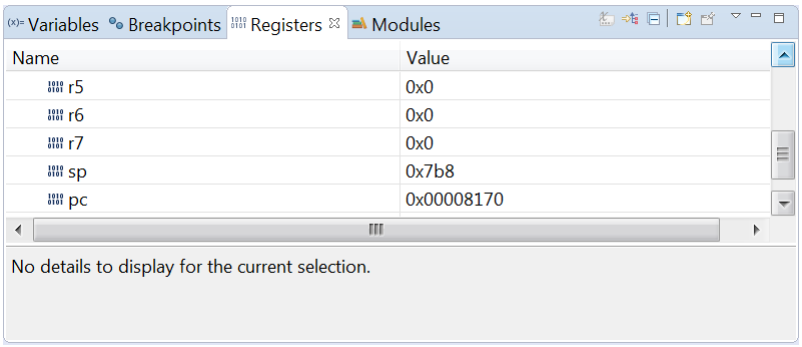
プログラムの実行に伴いレジスタ値が変更されると、実行を中断した時点で表示が更新されます。また、表示された値をクリックして、レジスタ値を変更することもできます。

[Variables]ビュー



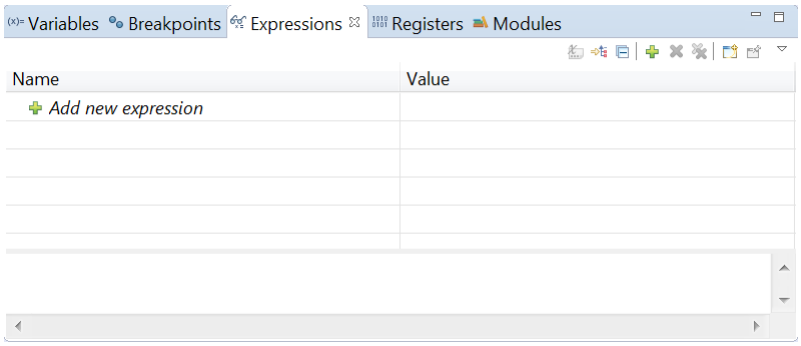
ターゲットプログラムがブレークした位置からのスコープ内にある変数の名称と値が表示されます。
[Value]の値を選択して数値をキー入力することにより、変数値を変更することができます。

[Registers]ビュー



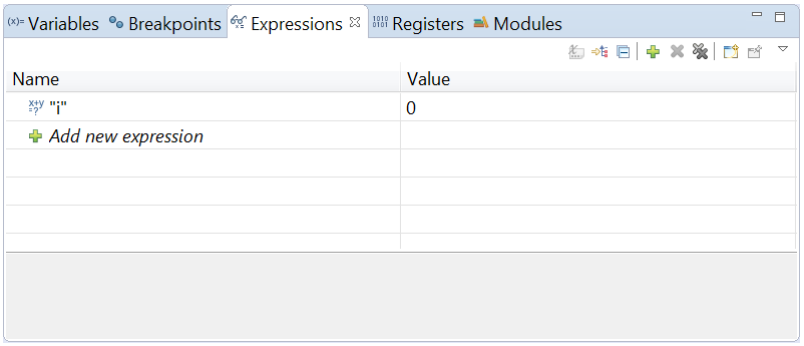
ターゲットプログラムがブレークした時点の CPU レジスタ値が表示されます。
[Value]の値を選択して数値をキー入力することにより、レジスタ値を変更することができます。

[Expressions]ビュー



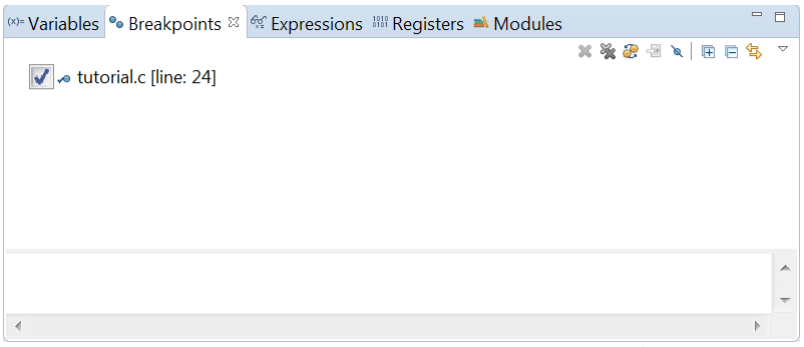
モニタしたい変数をここに登録して、ブレーク時の値を確認することができます。

操作 19: 式を登録するには、[+ Add new expression]をクリックしてモニタする変数名を入力します。



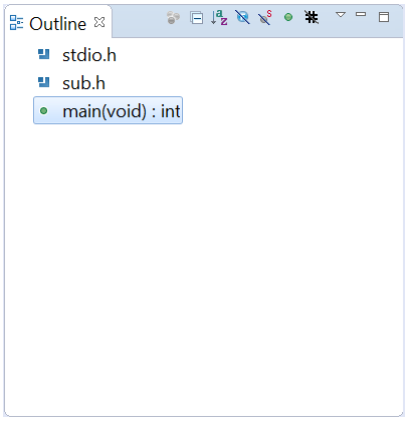
注:　ターゲットプログラムが変数のスコープ外に移動すると、エラーとなり値は表示されません。

[Breakpoints]ビュー



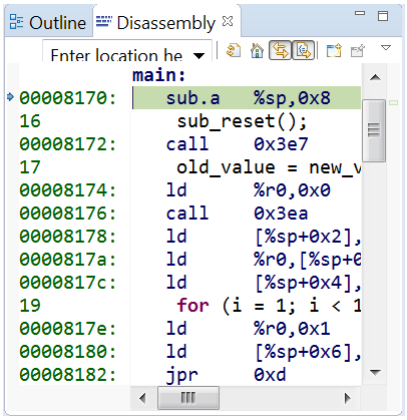
設定されているブレークポイントを表示します。チェックを外すと、ブレークポイントは無効となります(再びチェックを付けて有効にするまで、ブレークしません)。ブレークポイントの設定方法については、次節で説明します。

[Outline]ビュー



エディタが表示しているファイル内の構造(インクルードファイルや関数名)が表示されます。表示されている要素をクリックして、エディタの表示をすばやく希望の位置に移動させることができます。

[Disassembly]ビュー



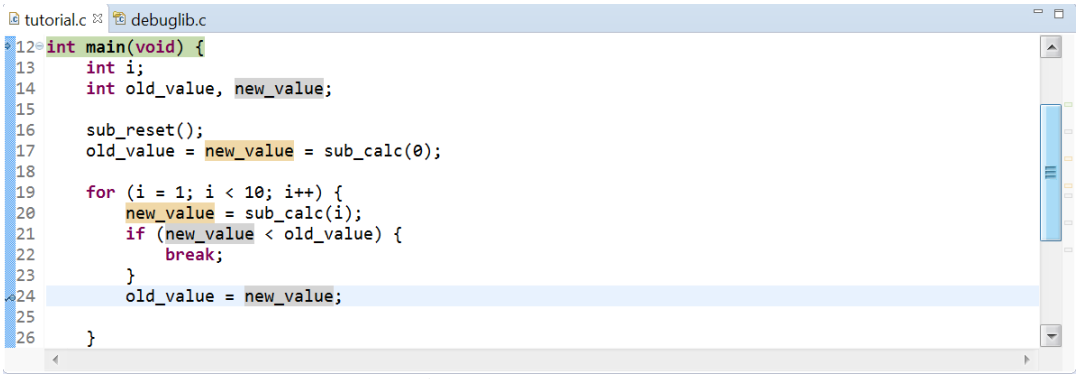
現在ブレークしている位置から、C ソースを逆アセンブルして表示します。
左端のマーカーバーに表示されている矢印は現在のプログラムカウンタが指し示している行、はその行にブレークポイントが設定されていることを示します。
このビューの中で、ブレークポイントの設定や、ステップ実行を行うことができます。

2.7.7 ブレークポイントの指定

指定した位置でプログラムの実行を中断させ、その時点の変数やレジスタの状態を確認することができます。このためには次のようにブレークポイントを追加します。

ブレークポイントを追加するには

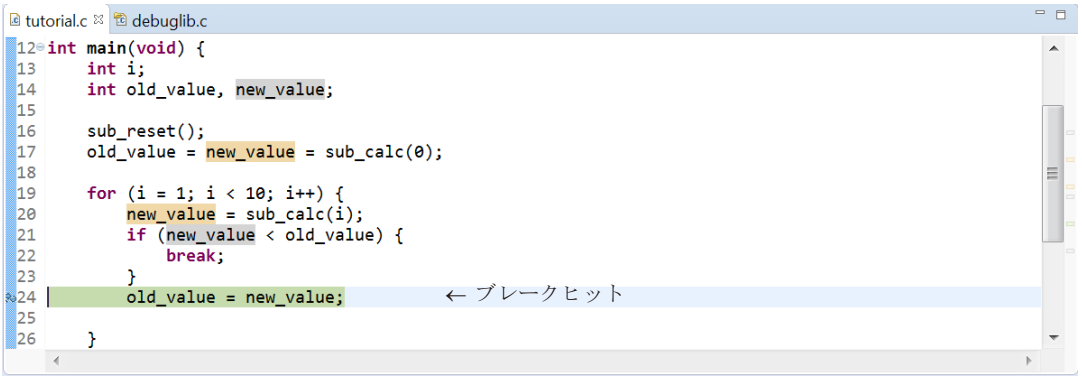
操作 20: エディタ内で“tutorial.c”の行番号“24”の数字上またはその左側のマーカーバーをダブルクリックします。



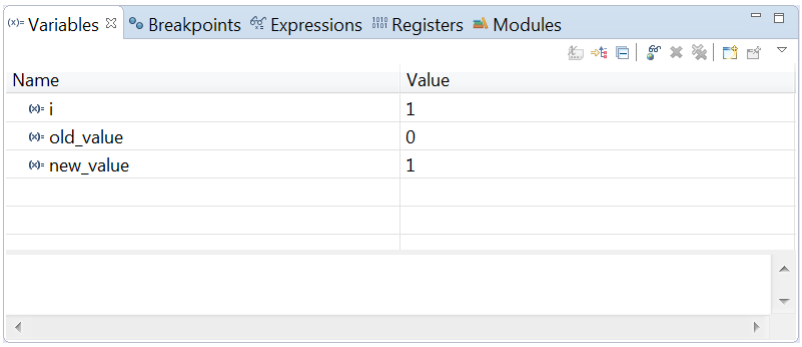
マーカーバーにが表示され、この行がブレークポイントに設定されたことを示します。この状態で、プログラムを実行してみます。

操作 21: プログラムの最初から実行させるために、ツールバー (Debug)ボタンをクリックします。

操作 22: ツールバー (Resume)ボタンをクリック(または[Run]メニュー> [Resume]を選択)します。
サンプルプログラムが実行を開始し、“tutorial.c”の 24 行目で停止します。



ここで[Variables]ビューを見ると、変数が次のように設定されています。



sub_calc 関数によって “i = 1” (1 段目)の四角錐数が計算され、“new_value” (結果)が 1 になっています。まだ “old_value” に代入されていませんので、プログラムは 24 行目の実行前に停止したことが分かります。

操作 23: (Resume)ボタンでプログラムの実行を繰り返してください。
[Variables]ビューの変数値は次のように変化し、正しく計算されていることが分かります。

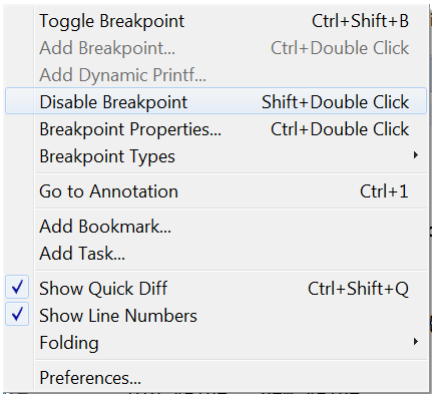
i = 1, new_value = 1　　(1 段の四角錐数 = 1)
i = 2, new_value = 5　　(2 段の四角錐数 = 5)
i = 3, new_value = 14　　(3 段の四角錐数 = 14)
:　　:
i = 9, new_value = 285　　(9 段の四角錐数 = 285)

ブレークポイントを無効にするには

設定を残したまま、ブレークポイントを無効にするには、次のように操作します。

◆ エディタ内の操作

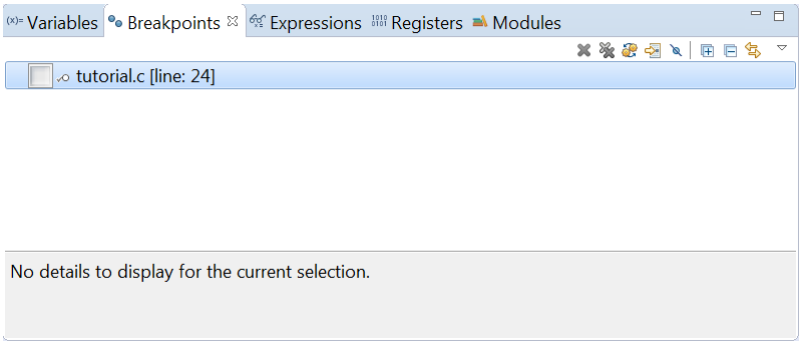
操作 24: 行番号上または上を右クリックし、表示されたコンテキストメニューから[Disable Breakpoint]を選択します。



ROM 上で同時に有効にできるブレークポイントの数には限りがあります。不要なブレークポイントは無効にしてください。
ブレークポイントが無効に設定されると、エディタ内のマーカーは に変わります。
*再度有効にするには、同様の操作でコンテキストメニューから[Enable Breakpoint]を選択します。

◆ [Breakpoints]ビュー内の操作

操作 25: “tutorial.c [Line: 24]”のチェックボックスをクリックしてチェックを外します。



再度有効にするには、同様の操作でチェックを付けてください。

2.7.8 ステップ実行

これまでは、プログラムを連続実行させましたが、1 行ずつステップ実行させ、動作を確認することができます。

操作 28: プログラムの最初から実行させるために、ツールバー (Debug)ボタンをクリックします。

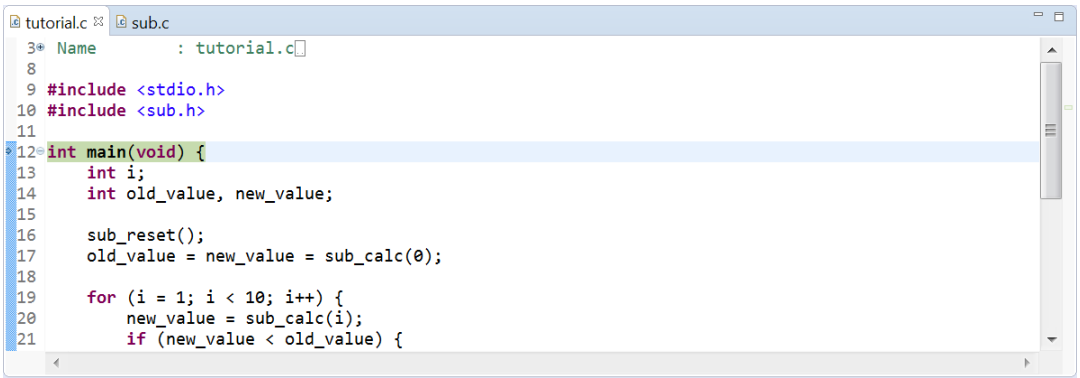
C ソースをステップ実行させるには

◆ コールした関数内もステップ実行させる場合

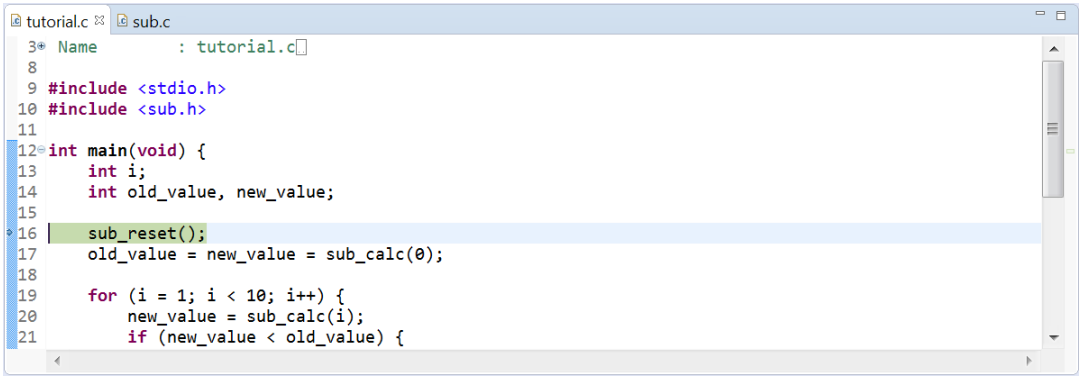
操作 29: ツールバー (Step Into)ボタンをクリックします。

ボタンをクリックするたびに、緑でハイライト表示された行が実行され、ハイライト表示が次に実行される行に移ります。

1.

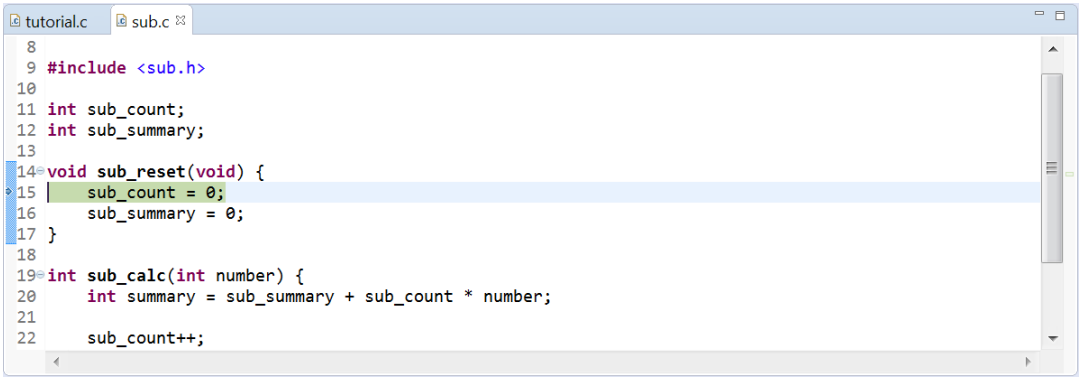


2.



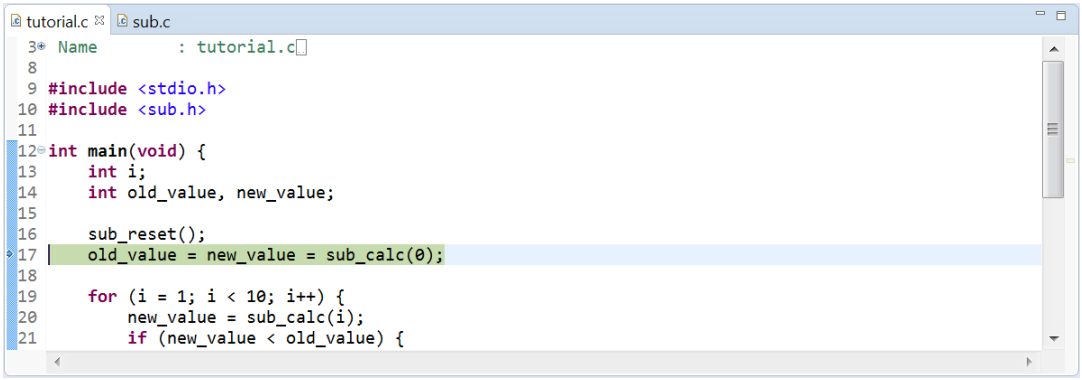
[Step Into]では sub_reset 関数や sub_calc 関数がコールされると、その内部もステップ実行します。

3.



操作 30: 関数内 (Step Return)ボタンをクリックします。
この操作により、実行中の関数内の残りを連続実行して、呼び出し元に 1 ステップで戻ることができます (Step Return)ボタンは関数内に入ることで使用可能になります。

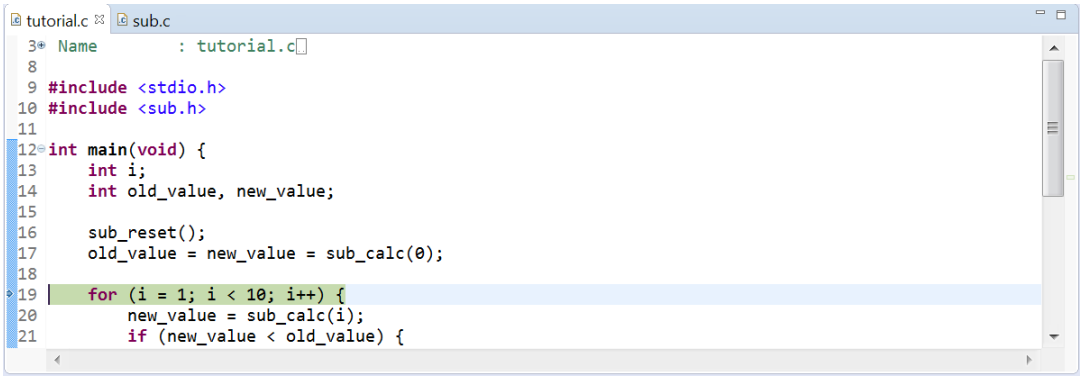
4.



◆ コールした関数内を連続実行させる場合

操作 31: ツールバー (Step Over)ボタンをクリックします。
[Step Into]と同様に 1 行ずつステップ実行します。ただし、このステップ実行では、関数のコールからリターンまでを 1 ステップとして連続実行します。これにより、デバッグが不要な関数の内部をスキップすることができます。

5.



ニーモニック命令単位でステップ実行させるには

操作 32: ツールバー (Instruction Stepping Mode)ボタンをクリックして ON 状態にします。
[Disassembly]ビューがアクティブになり、以降のステップ実行は[Disassembly]ビュー内のニーモニック命令単位で行われます。
C ソースのステップ実行に戻すには (Instruction Stepping Mode)ボタンをクリックして OFF 状態にしてください。
ステップ実行の操作は、前述の C ソースの場合と同様です。

[Step Into]	コールされたサブルーチン内も含め、1 命令ずつステップ実行
[Step Over]	1 命令ずつステップ実行。ただし、コールされたサブルーチン内は連続実行
[Step Return]	サブルーチン内の残りの命令を連続実行して呼び出し元にリターン

2.7.9 リセット

メニューの選択により、ターゲット CPU やターゲットボードをリセットすることができます。

ターゲット CPU をリセットするには

操作 33: [C17]メニューから[Reset]を選択します。
GDB コマンドファイル reset.gdb が実行され、ターゲット CPU をリセットします。

ターゲットボードをリセットするには

操作 34: [C17]メニューから[Reaet Target]を選択します。
GDB コマンドファイル reset_target.gdb が実行され、ターゲットボードをリセットします。この機能は、ICDmini モード時のみ有効です。また、ターゲットボードには ICDmini からリセット信号を入力する端子を設けておく必要があります。

2.7.10 C17 独自のデバッグ機能

[C17]メニューは、前節に示したリセット実行機能に加え、以下の機能を提供します。

1. **Launch LcdUtility**

   周辺回路シミュレータで使用する LCD パネル画面を設計するユーティリティ“LcdUtil17”を起動します。“LcdUtil17”の詳細は、“S5U1C17001C マニュアル”の“10.8 LCDUtil1（7 LCD パネルカスタマイズツール）”を参照してください。

2. **User Command**

   GDB コマンドファイル userdefine.gdb を実行します。このファイルは、プロジェクトの新規作成時にプロジェクトフォルダ内に生成されます。内容は、エディタを使用してユーザが自由に編集することができます。

3. Debug Command

以下のデバッグコマンドをサブメニューから選択して実行することができます。

c17 rst	CPU をリセットします。
c17 rstt	ターゲットボードにリセット信号を出力します。(ICDmini モードのみ)
c17 int	指定番号の割り込みを発生します。(シミュレータモードのみ)
c17 intclear	指定番号の割り込みを解除ます。(シミュレータモードのみ)
c17 tm	トレース機能を設定します。(シミュレータモードのみ)
c17 chgclkmd	ブレーク時に DCLK を高速クロックに切り換えるか設定します。(ICDmini モードのみ)
c17 flv	ICDmini Ver.2.0 の Flash プログラミング電圧を設定します。(ICDmini モードのみ)
c17 flvs	ICDmini Ver.2.0 に設定した Flash プログラミング電圧を解除します。(ICDmini モードのみ)

各コマンドの詳細は、“S5U1C17001C マニュアル”の“8.5 コマンドリファレンス”を参照してください。

2.7.11 デバッガの終了

操作 35: ツールバー (Terminate)ボタンをクリックします。
[C/C++]パースペクティブに戻るには[C/C++]ボタンをクリックします。IDE を終了するには、[File]メニューから[Exit]を選択してください。

3. リュートリアル 2（既存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 を参照してください。

3.1 GNU17 Ver.3.x プロジェクトのインポート

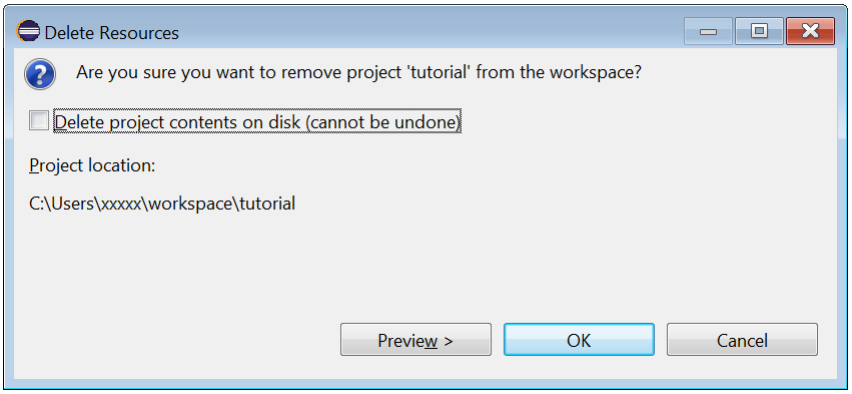
GNU17 Ver. 3.x で作成したプロジェクトは、そのままインポート可能です。

使用するサンプルプロジェクトフォルダ
C:¥EPSON¥GNU17V3¥sample¥tutorial

ワークスペースのプロジェクトフォルダは、IDE 上での同名プロジェクトのインポート操作によって上書きすることはできません。このため、まずチュートリアル 1 によって作成した tutorial プロジェクトを削除しておきます。

操作 1: IDE を起動します。
チュートリアル 1 を実行していない場合、次のプロジェクト削除の操作(操作 2～3)は必要ありません。

プロジェクトを削除するには
操作 2: [Project Explorer]ビューの“tutorial”を選択した状態で、[Delete]キーを押すか、あるいは[Edit]メニュー (または右クリックで表示されるコンテキストメニュー)から[Delete]を選択します。
[Delete Resources]ダイアログボックスが開きます。

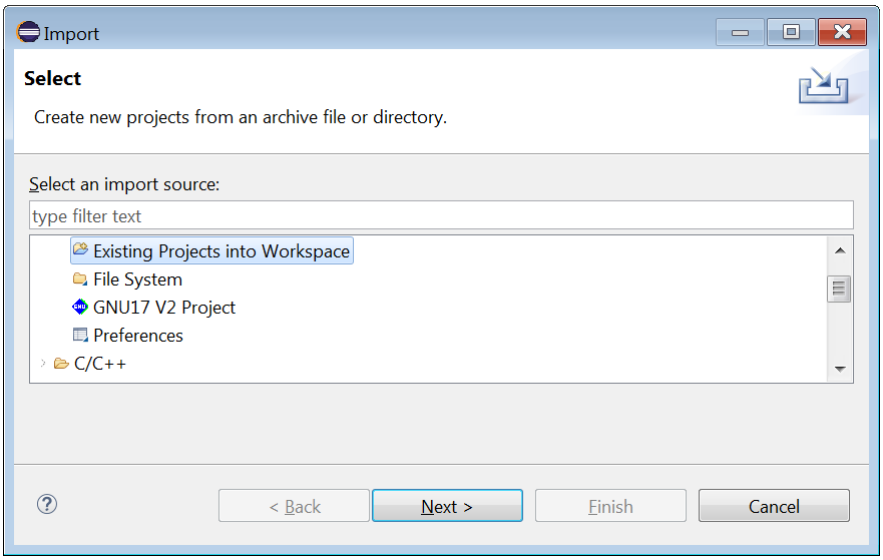


操作 3: [Delete project contents on disk (cannot be undone)]チェックボックスを選択し、[OK]ボタンをクリックします。

* [Delete project contents on disk (cannot be undone)]チェックボックスはデフォルトでは選択されていません。その状態で[OK]ボタンをクリックすると、tutorial プロジェクトは[Project Explorer]ビューから削除されますが、ディスク上のプロジェクトフォルダは削除されません。後から再度インポートすることができます。
このチェックボックスを選択して[OK]ボタンをクリックすると、tutorial プロジェクトは[Project Explorer]ビューに加え、ディスクのワークスペースディレクトリからも削除され、復活させることはできなくなります。

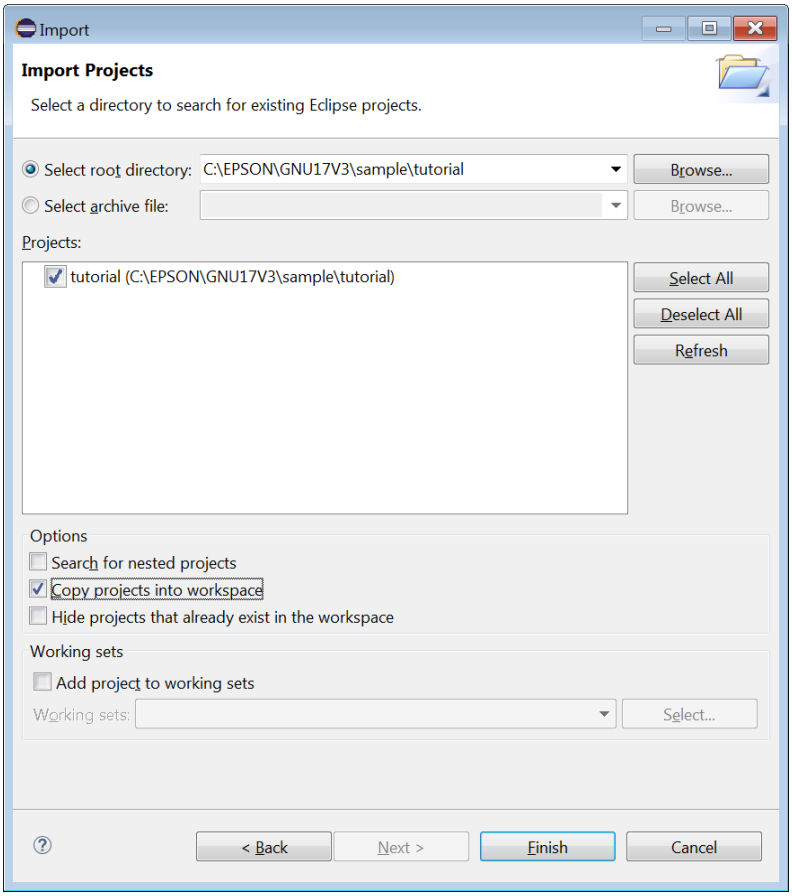
GNU17 Ver. 3.x プロジェクトをインポートするには

操作 4: IDE を起動し、[File]メニューから[Import...]を選択します。
[Import]ウィザードが起動します。



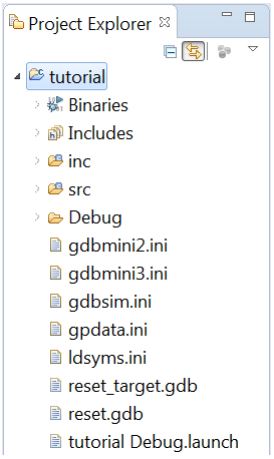
操作 5: 表示されている一覧の中から[General] > [Existing Projects into Workspace]を選択し、[Next >]ボタンをクリックします。

操作 6: [Select root directory:]の[Browse...]ボタンで、インポートするプロジェクトフォルダ“C:¥EPSON¥GNU17V3¥sample¥tutorial”を選択します。

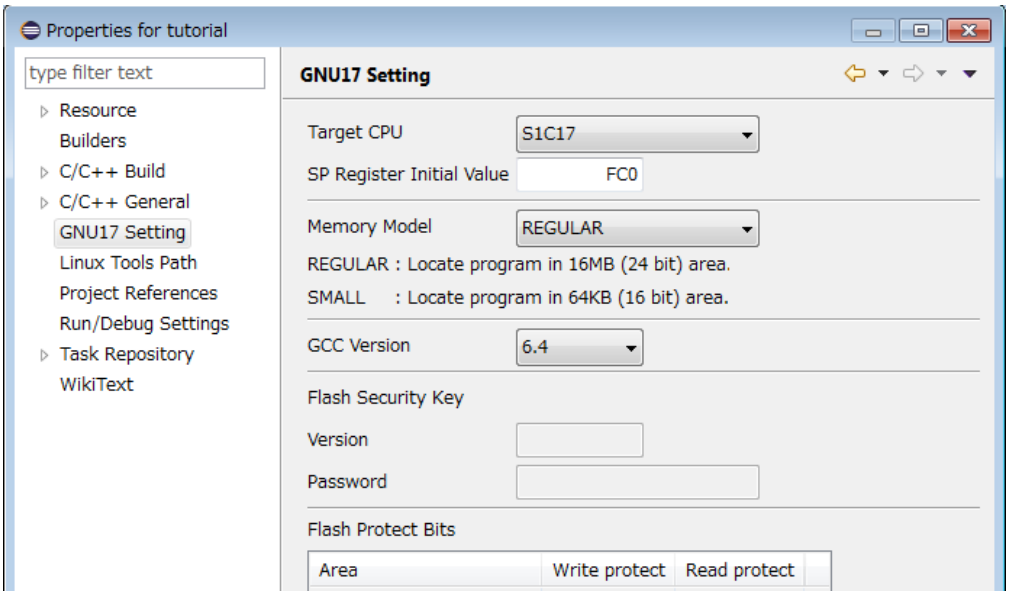


[Copy projects into workspace]チェックボックスを選択しておくことにより、プロジェクトのコピーがワークスペースディレクトリに作成され、オリジナルが変更されることはありません。

操作 7: [Finish]ボタンをクリックします。
[Project Explorer]ビューに、インポートしたプロジェクトが表示されます。



操作 8: [Project Explorer]ビューの“tutorial”を選択した状態で、[Project]メニュー (または右クリックで表示されるコンテキストメニュー)から[Properties]を選択します。[Properties]ダイアログボックスが開きますので、プロパティのリストから[GNU17 Setting]を選択します。



ここで、プロジェクトの基本設定を確認し、必要であれば設定し直してください。

GCC バージョンの変更

操作 9: 必要に応じて[GCC Version]のドロップダウンリストから、使用する C コンパイラ“gcc”のバージョンを選択します。通常は[6.4]のままとしてください。

*GCC バージョンを変更した場合は、[Properties]上で以下の設定が反映されているか確認してください。

1) C/C++ Build->Environment->GCC17-LOC->Value

<4.9 使用時>	<6.4 使用時>
${GNU17_LOC}¥gcc4	${GNU17_LOC}¥gcc6

2) C/C++ Build->Settings->Tool Settings->Cross Settings->Path

<4.9 使用時>	<6.4 使用時>
C:¥EPSON¥GNU17V3¥gcc4	C:¥EPSON¥GNU17V3¥gcc6

3) C/C++ Build->Settings->Tool Settings->Cross GCC Compiler->Optimization->Optimization Level

<4.9 使用時>	<6.4 使用時>
-O0/-O1/-O2/-O3/-Os	-O0/-O1/-Os

4) C/C++ Build->Settings->Tool Settings->Cross GCC Compiler->Debugging->Other debugging flags

<4.9 使用時>	<6.4 使用時>
-gstabs	-g

3.2 GNU17 Ver. 2.x プロジェクトのインポート

GNU17 Ver. 2.x で作成したプロジェクトを GNU17 Ver. 3.x にインポートし、ビルドするための手順を示します。ここでは、GNU17 Ver. 3.x の機能である起動処理ライブラリ“crt0.o”を使用しません。以下の作業が必要になります。

1. リンカスクリプトファイルの準備
2. リンカオプションの設定
3. ソースコードの変更
4. GDBコマンドファイルの修正

以下、これらの作業手順を見ていきます。

サンプルプロジェクトの準備

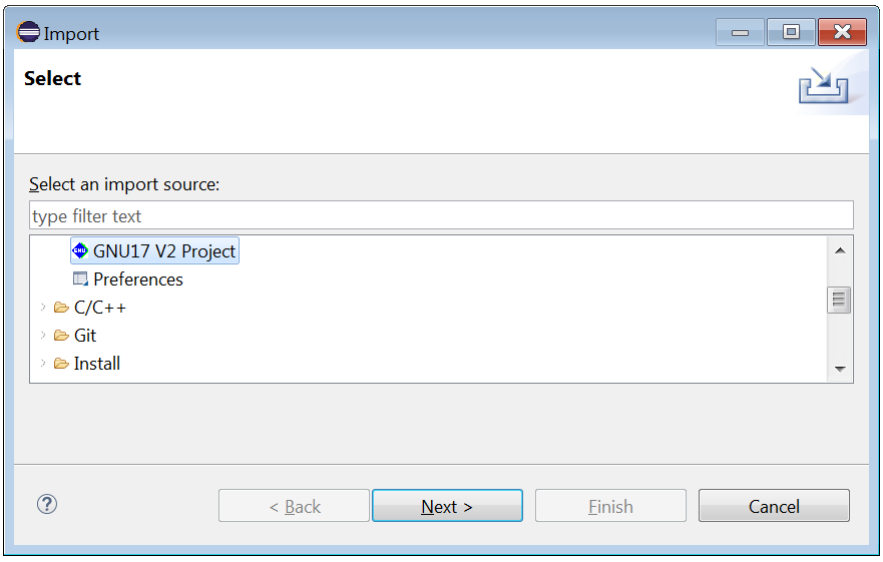
本チュートリアルで使用するサンプルプロジェクト“tutorial_v2import”は、“C:¥EPSON¥GNU17V3¥ sample”フォルダに ZIP ファイルとして用意されていますので、始めにこのファイルを展開してください。

操作 1: sample フォルダ内にある“tutorial_v2import.zip”を右クリックしてコンテキストメニューを表示させ、[すべて展開(T)...]を選択してファイルを展開します。表示されたダイアログでは、下記のフォルダを展開先に設定してください。
C:¥EPSON¥GNU17V3¥sample¥tutorial_v2import

すでにこのプロジェクトがインポートされている場合は、“tutorial_v2import”プロジェクトを、ディスク上のプロジェクトフォルダも含め､削除してください(3.1 節の“プロジェクトを削除するには”参照)。

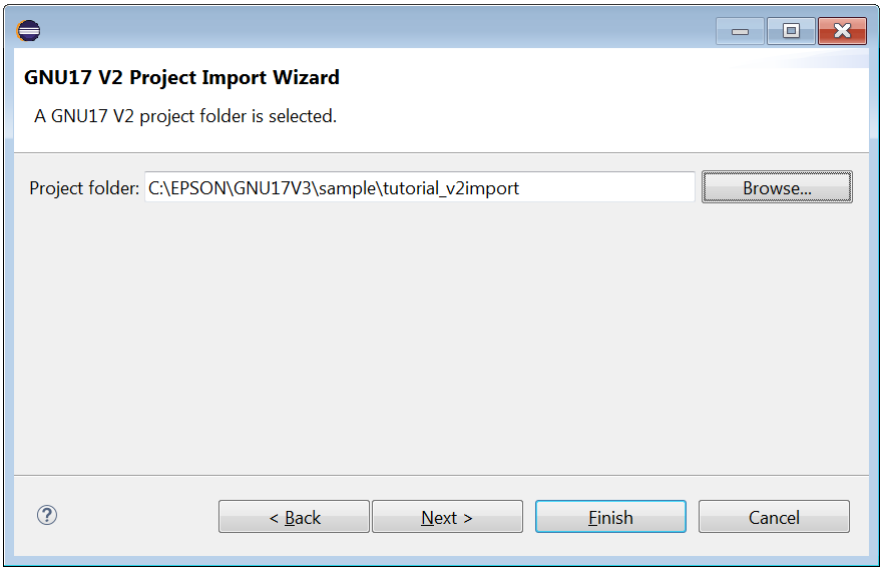
GNU17 Ver. 2.x プロジェクトのインポート

操作 2: IDE を起動し、[File]メニューから[Import...]を選択します。
[Import]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[General] > [GNU17 V2 Project]を選択し、[Next >]ボタンをクリックします。

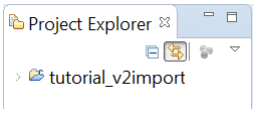
操作 4: [Project folder:]の[Browse...]ボタンで、インポートするプロジェクトフォルダ“C:¥EPSON¥GNU17V3¥sample¥tutorial_v2import”を選択します。



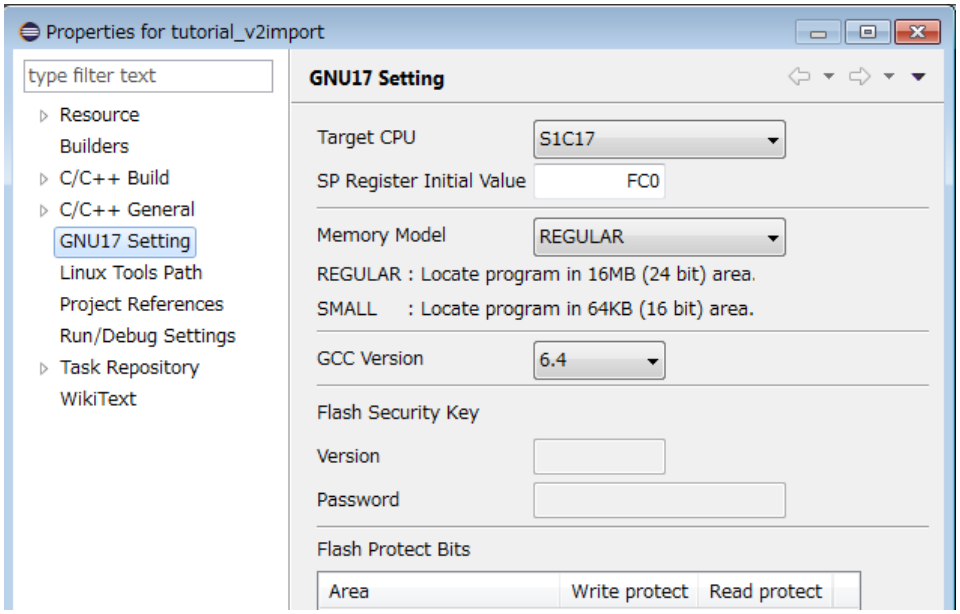
操作 5: [Finish]ボタンをクリックします。

操作 6: “ImportV2Note”(C:¥EPSON¥GNU17V3¥doc¥ImportV2Note.txt)が表示されますので、一読後に閉じてください。

[Project Explorer]ビューに、インポートしたプロジェクトが表示されます。



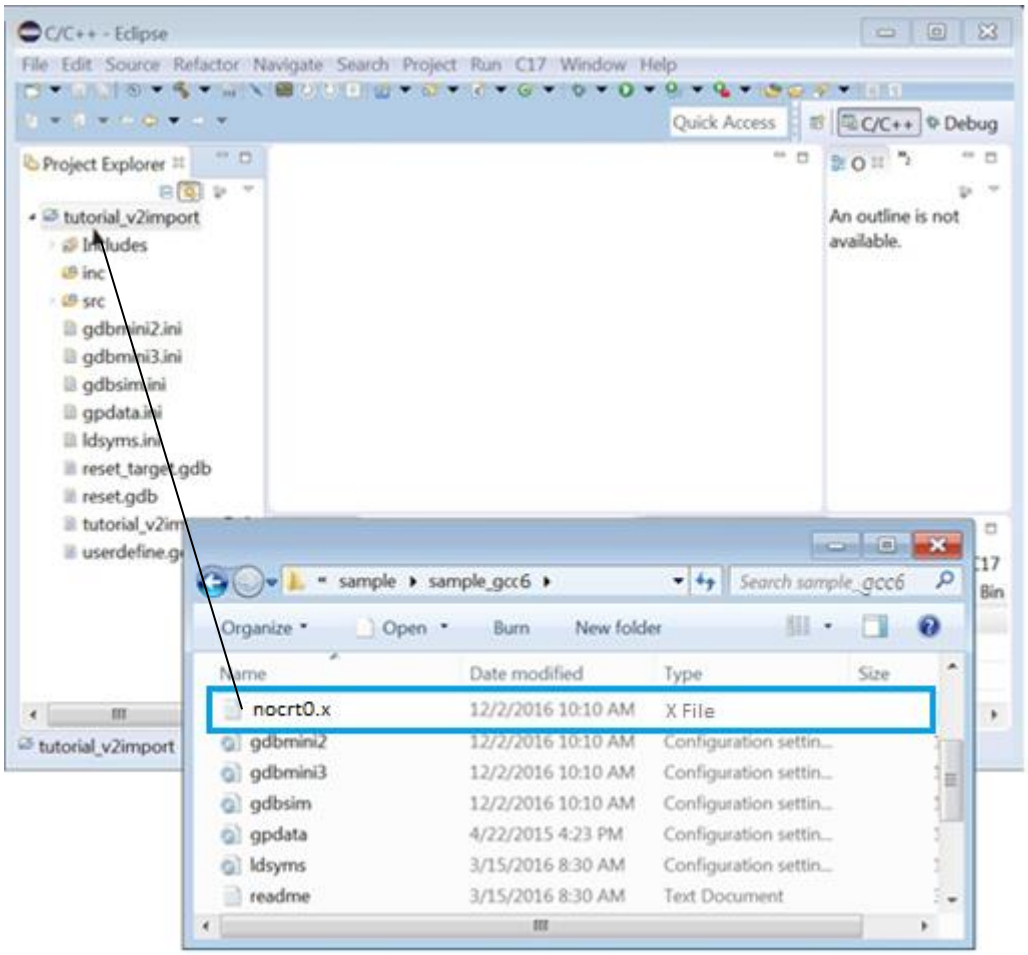
操作 7: [Project Explorer]ビューの“tutorial_v2import”を選択した状態で、[Project]メニュー (または右クリックで表示されるコンテキストメニュー)から[Properties]を選択します。[Properties]ダイアログボックスが開きますので、プロパティのリストから[GNU17 Setting]を選択します。



ここで、プロジェクトの基本設定を確認し、必要であれば設定し直してください。

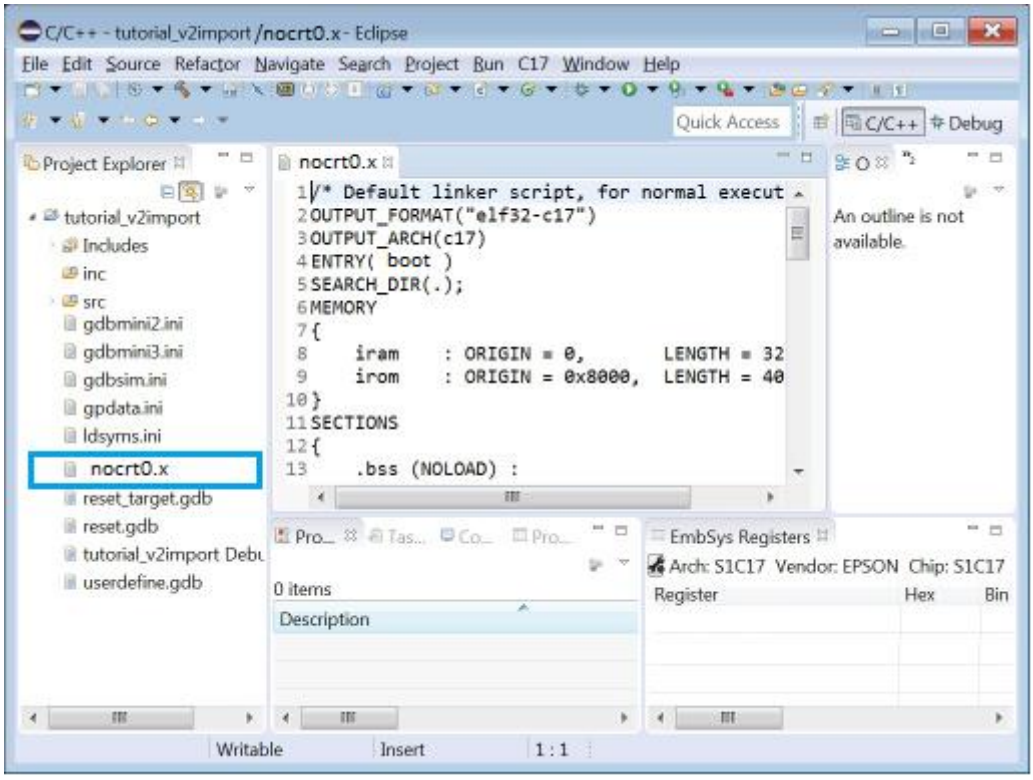
リンカスクリプトファイルの準備

操作 8:“C:¥EPSON¥GNU17V3¥sample¥nocrt0_gcc6”フォルダか ら“nocrt0.x”を[Project Explorer]ビューの“tutorial_v2import”プロジェクトにドラッグアンドドロップします。[File Operation]ダイアログボックスが表示されますので、[Copy files]を選択して[OK]ボタンをクリックしてください。



このリンカスクリプトファイルを修正して使用します。以下のように、ベクタテーブルが定義されているオブジェクトファイルの指定を“boot.o“からユーザが作成したファイル(本サンプルでは vector.o)に修正します。

操作 9: [Project Explorer]ビューの“nocrt0.x”をダブルクリックしてエディタで開きます。



操作 10: 以下の 3 カ所を修正後、ファイルをセーブします。

1. 起動処理ルーチンの設定
   ENTRY(boot) → ENTRY(boot)
   ※本例では変更不要です。

2. ベクタテーブルの設定
   .vector：
   {
   PROVIDE (START_vector = .) ;
   KEEP (*boot.o(.rodata)) → KEEP (*vector.o(.rodata))
   } > irom

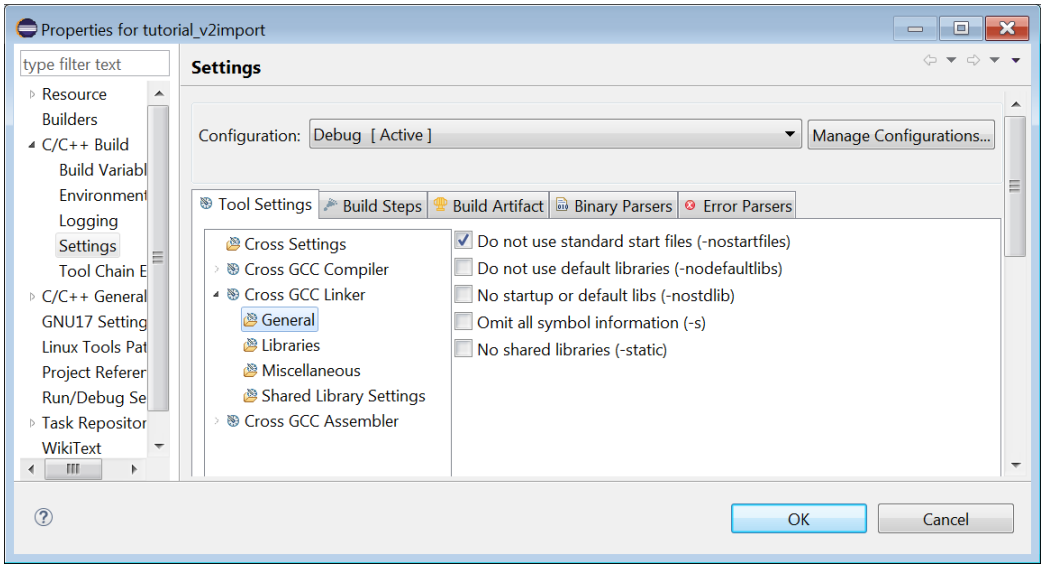
3. .rodataセクションからベクタテーブルを除外
   .rodata：
   {
   PROVIDE (START_rodata = .) ;
   *(EXCLUDE_FILE (*boot.o) .rodata) → *(EXCLUDE_FILE (*vector.o) .rodata)
   (.rodata.)
   PROVIDE (END_rodata = .) ;
   } > irom

リンカオプションの設定

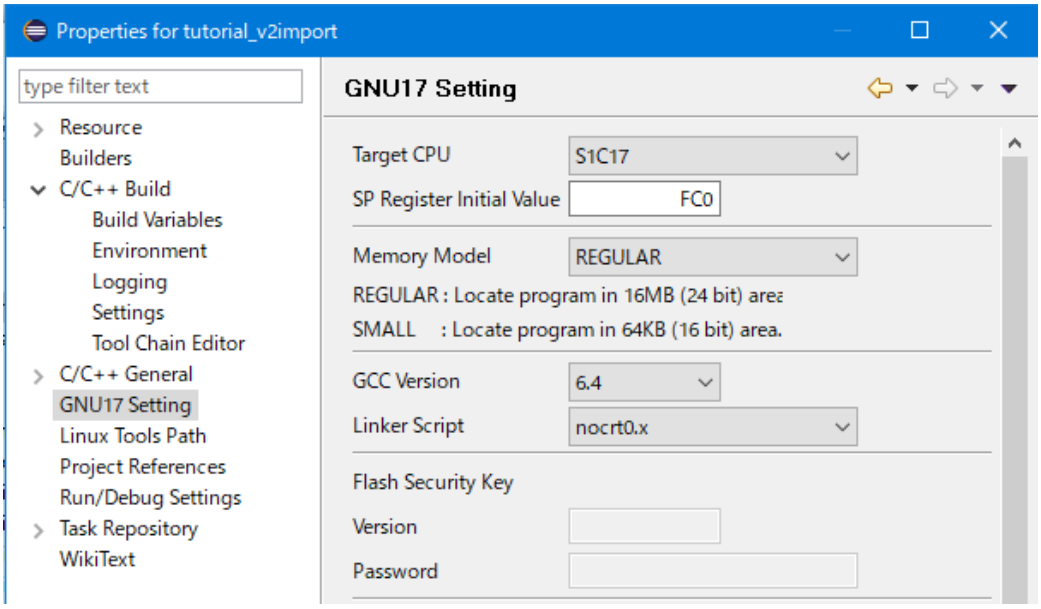
準備したリンカスクリプトファイルが使用されるように、リンカオプションを設定します。

操作 11: [Project Explorer]ビューの“tutorial_v2import”を選択した状態で、[Project]メニュー (または右クリックで表示されるコンテキストメニュー)から[Properties]を選択し、[Properties]ダイアログボックスを開きます。[C/C++ Build] > [Setting]を選択し、[Tool Settings]タブのページを表示させます。

操作 12: [Tool Settings]ページの設定リストから[Cross GCC Linker] > [General]を選択し、表示されたページで[Do not use standard start files (-nostartfiles)]を選択します。



操作 13: [Properties]ダイアログボックスの[GNU17 Setting]ページを開き、[Linker Script]ドロップダウンリストから“nocrt0.x”を選択します。



操作 14: [OK]ボタンをクリックして[Properties]ダイアログボックスを閉じます。

ソースコードの変更

標準入出力機能などを使うために、起動処理ルーチン（本例の場合は、vector.c の“void boot(void)”）などで以下を記述している場合は、ソースコードを変更します。

1. _init_sys();
   GNU17 Ver. 3.x 上で“_init_sys();”は、定義されていないため、削除する。

2. _init_lib();

GNU17 Ver. 3.x 上で“__init_lib();”は、定義されていないため、以下の①②を追加する。

① “#include <libstdio.h>”を削除し、以下を追加する。

```
#include <smcvals.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

② “_init_lib”関数を追加する。

```
static void _init_lib(void);
…
int errno;
FILE _iob[FOPEN_MAX + 1];
FILE *stdin;
FILE *stdout;
FILE *stderr;
unsigned long seed;
time_t gm_sec;
…
static void _init_lib(void) {
    /* initalize for general */
    errno = 0;                      /* clear error number */

    /* initalize for io stream in io.lib */
    _iob[0]._flg = _UGETN;          /* initialize stdin stream */
    _iob[0]._buf = 0;
    _iob[0]._fd = 0;

    _iob[1]._flg = _UGETN;          /* initialize stdout stream */
    _iob[1]._buf = 0;
    _iob[1]._fd = 1;

    _iob[2]._flg = _UGETN;          /* initialize stderr stream */
    _iob[2]._buf = 0;
    _iob[2]._fd = 2;

    stdin = &_iob[0];               /* initialize each file pointer */
    stdout = &_iob[1];
    stderr = &_iob[2];

    /* initialize for others in lib.lib */
    seed = 1;                       /* initialize random seed */
    gm_sec = -1;                    /* initialize time */
}
```

3. _exit();

GNU17 Ver. 3.x 上で“_exit()”は、定義されているため、以下の①-③を変更する。

① “void _exit(void);”を削除し、以下を追加する。

```
extern void _exit(int);
```

② “void _exit(void)”を削除する。

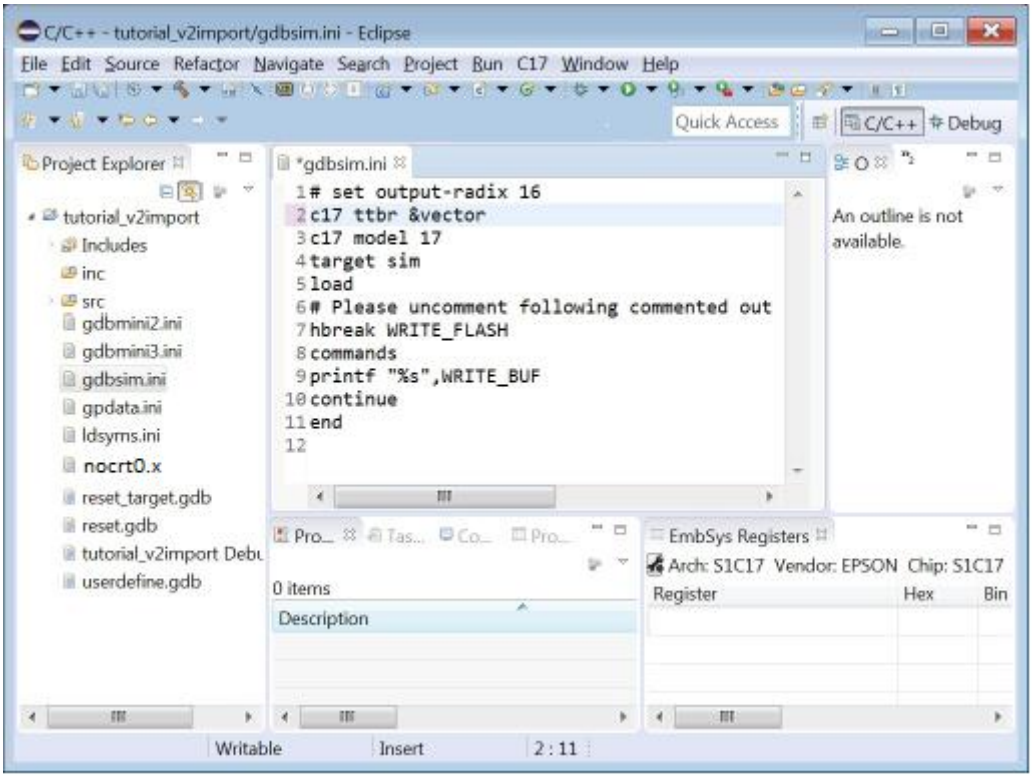
③ “_exit();”に引数を追加する。

```
void boot(void)
{
    …
    _exit(0);        // In last, go to exit in sys.c
}
```

GDB コマンドファイルの修正

ベクタテーブルアドレスの指定をユーザの定義(本サンプルでは vector)に修正します。

操作 15: [Project Explorer]ビューの“gdbsim.ini”をダブルクリックしてエディタで開き、“c17 ttbr &_vector”を“c17 ttbr &vector”に修正後、ファイルをセーブします。



以上で、必要な修正が完了しました。この後は、通常の手順でビルドとデバッグを行います。

Appendix A セクションとリンカスクリプト

A.1　セクション

ここでは、ソースファイルの作成およびリンクの際に必要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(フラッシュメモリ)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。初期値は 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。
ベクタテーブルは“.rodata”属性を持ちますが、指定の位置に配置する必要があるため、“.vector”という専用の出力セクションが指定されています。GNU17 Ver.3.x では、標準でリンクされる起動処理用ライブラリ“crt0.o”がベクタテーブルを含んでおり、これが“.vector”セクションに配置されるようになっています。

(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。

A.2　リンカスクリプト

リンカスクリプトは、各セクションの格納位置や実行位置を指定し、リンクを制御するために使用されます。特に指定のない場合はリンカに組み込まれたデフォルトのリンカスクリプトが使用されますが、独自に作成した外部のリンカスクリプトファイルを使用することもできます。
標準的なリンカスクリプトの例を以下に示します。デフォルトのリンカスクリプトは、コマンドプロンプトから“ld --verbose”を実行することにより表示できます。

標準的なリンカスクリプトの例

```
OUTPUT_FORMAT(“elf32-c17”)                                                                    (1)
OUTPUT_ARCH(c17)
ENTRY(_start)
SEARCH_DIR(.);
MEMORY                                                                                        (2)
{
        iram : ORIGIN = 0, LENGTH = 32K
        irom : ORIGIN = 0x8000, LENGTH = 4064K
}
SECTIONS                                                                                      (3)
{
        .bss (NOLOAD) :
        {
                PROVIDE (    START_bss = .) ;
                *(.bss)
                *(.bss.*)
                *(COMMON)
                PROVIDE (    END_bss = .) ;
        } > iram
        .vector :
        {
                PROVIDE ( START_vector = .) ; KEEP (*crt0.o(.rodata)) PROVIDE (    END_vector = .) ;
        } > irom
        .text :
        {
                PROVIDE (    START_text = .) ;
                *(.text.*)
                *(.text)
                PROVIDE (    END_text = .) ;
        } > irom
        .data :
        {
                PROVIDE (    START_data = .) ;
                *(.data)
                *(.data.*)
                PROVIDE (    END_data = .) ;
        } > iram AT > irom
        .rodata :
        {
                PROVIDE (    START_rodata = .) ;
                *(EXCLUDE_FILE (*crt0.o ) .rodata)
                *(.rodata.*)
                PROVIDE (    END_rodata = .) ;
        } > irom
        PROVIDE (    START_data_lma = LOADADDR(.data));
        PROVIDE ( END_data_lma = LOADADDR(.data) + SIZEOF (.data));
        PROVIDE (    START_stack = 0x0007C0);
}
```

(1)　OUTPUT_FORMAT～SEARCH_DIR

OUTPUT_FORMAT(“elf32-c17”) 出力ファイルの形式
OUTPUT_ARCH(c17)　ターゲットのアーキテクチャ
ENTRY(_start)　エントリーポイント(プログラム開始アドレス)
“_start”は起動処理用ライブラリ“crt0.o”内のブート処理関数のアドレスを指します。リンカのコマンドラインで指定されている場合は、そちらの指定が優先されます。
SEARCH_DIR(.);　ライブラリ検索パス
リンカのコマンドラインで指定されている場合は、そちらの指定が優先されます。

この部分は固定の内容です。外部のリンカスクリプトファイルを作成する場合も、同じように記述してください。

(2)　MEMORY

ターゲットのメモリ構成を設定します。“iram”と“irom”は領域名称で、それぞれ内蔵 RAM、内蔵 ROM(フラッシュ)を表します。GNU17 では属性の記述は省略しています。“ORIGIN”は先頭アドレス、“LENGTH”は容量を指定します。
デフォルトのリンカスクリプトは、S1C17 Family マイクロコントローラに多い、内蔵 RAM が 0 番地に、内蔵 ROM が 0x8000 番地に配置されたメモリ構成に合わせた記述となっています。
外部のリンカスクリプトファイルを作成する場合は、ここにターゲット機種のメモリ構成を記述してください。ターゲットのメモリ構成に合わせ、“iram”と“irom”以外の名称も使用可能です。

(3)　SECTIONS

セクションをメモリの先頭方向から配置順に定義します。定義の基本形は次のとおりです。

```
出力セクション名 [ (セクションタイプ) ] :
{
        PROVIDE (シンボル名 = .) ;
        *(入力セクション名)
        :
        PROVIDE (シンボル名 = .) ;
} > VMA 領域名 [ AT > LMA 領域名 ]                    [ ]は省略可能
```

これは、メモリ“VMA 領域名”の中に“出力セクション”を設定し、その中にリンクするオブジェクトファイルの中に定義されている“入力セクション”のデータをファイルの指定順に連続して配置する、　という定義です。
“*(入力セクション名)”の‘*’はワイルドカードで、リンクするすべてのオブジェクトファイルに含まれる“入力セクション”を指定しています。
“AT > LMA 領域名”はデータを格納しておく領域を実行領域とは別に指定するもので、たとえば“.data” セクションのように実体を ROM(LMA 領域)に置き、実行時は RAM(VMA 領域)にコピーしてアクセスするような場合に指定します。VMA は“Virtual Memory Address”、LMA は“Load Memory Address”を意味します。“AT > LMA 領域名”を省略した場合は、“VMA 領域” = “LMA 領域”となります。
“PROVIDE (シンボル名 = .) ;”は、‘.’(ロケーションカウンタの値 = 記載されている位置のアドレス)を“シンボル名”で定義します。たとえば、“PROVIDE (__START_bss = .)　;”は内蔵 RAM の先頭アドレスを“　START_bss”というシンボルとして定義し、“PROVIDE (　 END_bss = .) ;”は、入力セクションをすべて配置し終えた“.bss”セクションの終了アドレスを“__END_bss”というシンボルとして定義します。これらのシンボルはターゲットプログラム中で使用可能です。ソースファイルの中で同じシンボルが定義されている場合は、ソースファイルの定義が有効です。
その他、例で使用しているスクリプトを以下に示します。

.bss (NOLOAD)
NOLOAD はセクションタイプを指定するキーワードで、実体のない(リンクによって実際のコードやデータが格納されない)セクションを表します。

KEEP (*crt0.o(.rodata))
KEEP は、指定の入力セクションを必ず出力セクションに含めるためのキーワードです。この例では、“crt0.o” の “.rodata” セクション(ベクタテーブル)が “.vector” セクションとして出力されます。

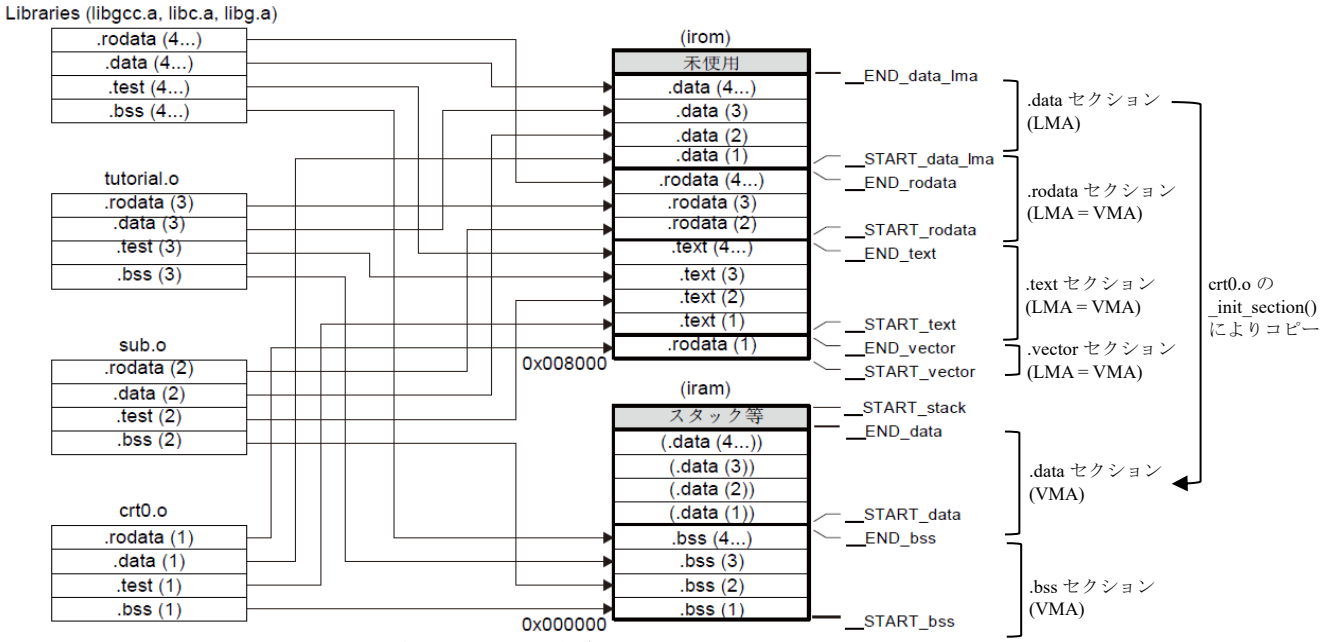
***(EXCLUDE_FILE (*crt0.o) .rodata)**
EXCLUDE_FILE は、指定した入力セクションを出力セクションに含めないようにするキーワードです。この例では、“.vector” セクションに出力した “crt0.o” の “.rodata” セクションを “.rodata” セクションへの出力から除外します。

PROVIDE (　START_data_lma = LOADADDR(.data));
LOADADDR は格納領域の先頭アドレスを表します。この例では、“.data” セクションが格納される ROM 領域の先頭アドレスを “　START_data_lma” というシンボルとして定義しています。

PROVIDE (　END_data_lma = LOADADDR(.data) + SIZEOF (.data));
SIZEOF は指定したセクションのサイズを取得するキーワードです。この例では、“.data” セクションが格納される ROM 領域の終了アドレスを “__END_data_lma” というシンボルとして定義しています。

PROVIDE (　START_stack = 0x0007C0);
スタックポインタの初期値を “ START_stack” というシンボルとして定義しています。ソースファイルの中で次の例のように定義することで、リンカスクリプトの指定は無視され、ソースファイルの指定が有効になります。
asm(“.global START_stack”);
asm(“.set　START_stack, 0x1fc0”);

このリンカスクリプト例では、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 A.2.1 にチュートリアル 1 で使用したファイル構成とした場合の、リンク後のメモリマップを示します。



(この図には、オブジェクトに存在しないセクションも記載されています。)
図 A.2.1 リンカスクリプト例によるメモリマップ

プログラムは、ROM 上のアドレス“__START_vector”から“__END_data_lma”までの領域に格納され、RAM 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。
したがって、このプログラムおける ROM および RAM の使用量は、次のように求めることができます。
　ROM 使用量[バイト] = END_data_lma － START_vector
　RAM 使用量[バイト] = END_data － START_bss ＋ スタック使用量(+ ヒープ使用量)
各セクションで定義されたシンボルの値は、リンクマップファイルに出力されます。例として、“tutorial¥Debug¥tutorial.map”を参照してください。

A.3　リンカスクリプト例

ここでは、リンカスクリプトの記述例を示します。また、以下のフォルダに各種サンプルプログラムを用意しています。サンプルプログラムのリンカスクリプトファイルも合わせて参照してください。
C:¥EPSON¥GNU17V3¥sample

前提として、以下が定義されていることとします。
・対象プロジェクト内に追加するセクションの定義がされている。
・対象リンカスクリプトファイル内に、MEMORY として以下が定義されている。

```
MEMORY
{
        iram            : ORIGIN = 0,           LENGTH = xxK
        irom            : ORIGIN = 0x8000,      LENGTH = xxK
}
```

● 例：対象データを指定アドレスに置く
プログラム内にデータを設定する。

```
const unsigned char __attribute__ ((section (".testdata"))) checkerLineBit[16] = {
        0x00, 0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F
};
```

対象データを 0xB000 番地から配置する。

```
.testdata (0xB000) :                    →0xB000番地を指定
{
    *(.testdata);
    . = ALIGN(0x800);                   →ロケーションカウンタを2KB境界に設定
} > irom = 0xff                         →iromに配置し空き領域を0xffffでFILLする
```

● 例：対象プログラム“test.c”の実態をFlashに置き、RAMで実行させる

```
…
.text :
    {
    PROVIDE (__START_text = .) ;
    *(.text.*)
    *(EXCLUDE_FILE (*test.o ) .text)        →.text セクションから test.o を除外する
    ….
    PROVIDE (__END_text = .) ;
} > irom
…
.rodata :
{
    PROVIDE (__START_rodata = .) ;
    ….
```

```
        *(EXCLUDE_FILE (*test.o) .rodata)          →.rodata セクションから test.o を除外する
        *(.rodata.*)
        PROVIDE (__END_rodata = .) ;
} > irom
…
.test_text :                                       →.test_text セクションを追加
{
        __START_test_text = . ;
        *test.o(.text);                            →属性.text として test.o を配置
        __END_test_text = . ;
} > iram AT > irom                                 →実態はiromに置き、iramで実行を指定

.test_rodata :                                     →.test_rodata セクションを追加
{
        __START_test_rodata = . ;
        *test.o(.rodata);                          →属性.rodataとしてtest.oを配置
        __END_test_rodata = . ;
} > iram AT > irom                                 →実態はiromに置き、iramで実行を指定

__START_test_text_lma = LOADADDR(.test_text);
__START_test_rodata_lma = LOADADDR(.test_rodata);
                          →__START_test_xxxx_lmaに追加したセクションの先頭アドレスを指定
```

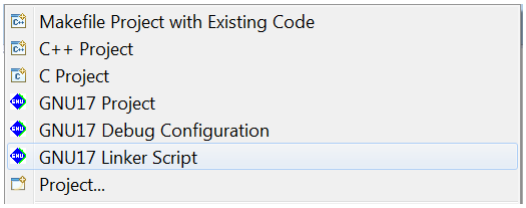
A.4　リンカスクリプト生成ウィザード

本 IDE には、リンカスクリプトファイルを作成するためのリンカスクリプト生成ウィザードが組み込まれています。

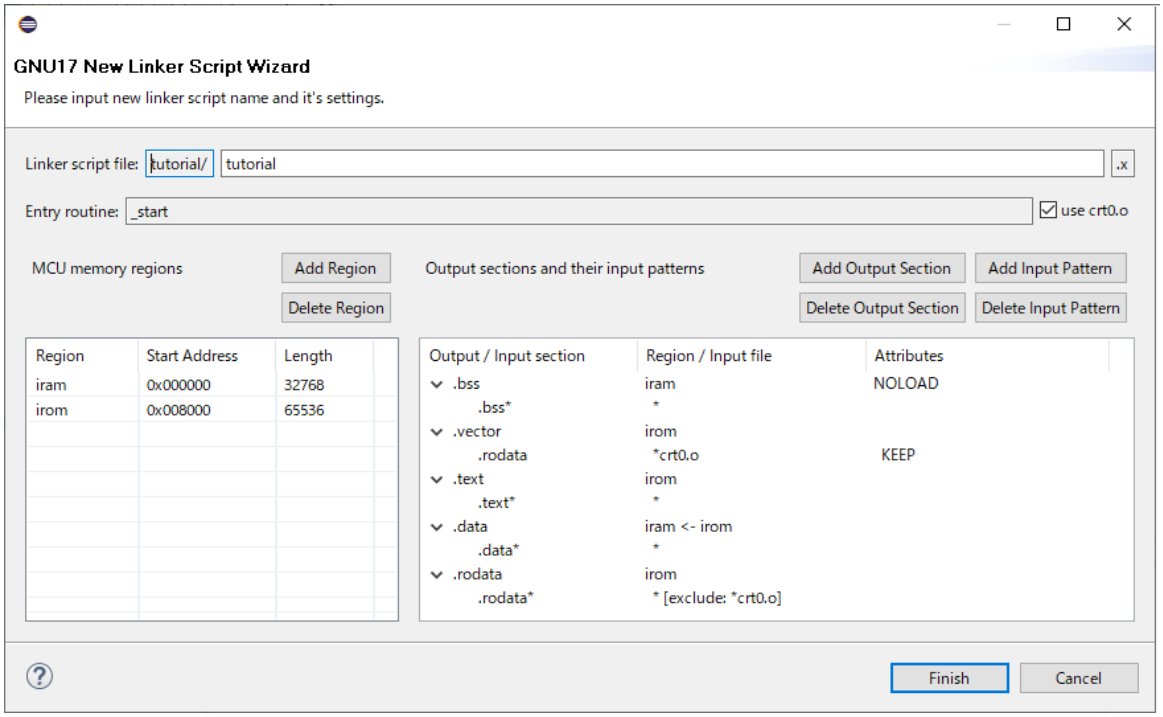
リンカスクリプトファイルを新規作成するには

操作 1:ツールバー(New)ドロップダウンリスト*から[GNU17 Linker Script]を選択します。

*その他、[File]メニュー > [New]、ツールバー(New　C/C++　Project)ドロップダウンリストなどからも選択可能



[GNU17 New Linker Script]ウィザードが起動します。起動時には、デフォルトの設定がされています。



操作 2:以下に説明する項目を設定し、[Finish]ボタンをクリックします。
作成したリンカスクリプトファイルは、プロジェクトの[Properties]ダイアログボックスの[GNU17 Setting]ページで選択してください。

設定項目

(1)　Linker script file:
リンカスクリプトファイル名を入力します。リンカスクリプトファイルは、現在選択されているプロジェクトフォルダ内に“(指定ファイル名).x”として生成されます。

(2)　Entry routine:
ENTRY コマンドで設定するプログラム開始アドレスを入力します。
起動処理用ライブラリ“crt0.o”を使用する場合は、[use crt0.o]チェックボックスを選択し、“_start”を設定します。

(3)　MCU memory regions
MEMORY コマンドで記述するメモリ構成(領域名称、開始アドレス、容量)を入力します。以下の構成がデフォルトで定義されています。

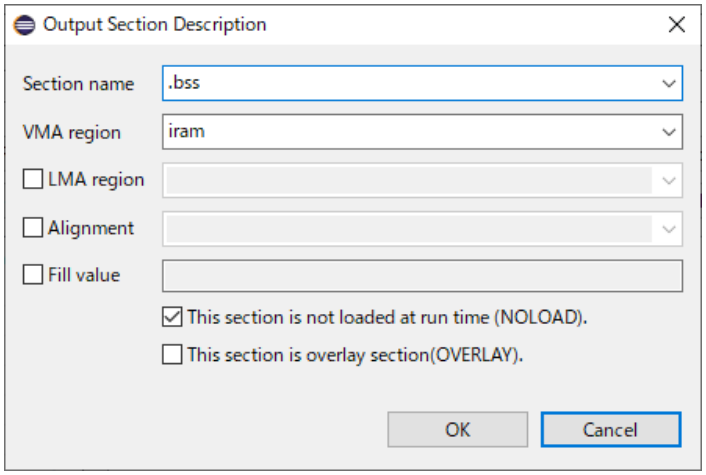
iram : ORIGIN = 0, LENGTH = 32K
irom : ORIGIN = 0x8000, LENGTH = 64K

iram と irom の ORIGIN と LENGTH を変更する場合は、表示されている[Start Address]と[Length]の数値をクリックして選択し、新たな数値を入力します。
新たなメモリ領域を追加するには、[Add Region]ボタンをクリックします。メモリ領域のリストに“regionX”という名称の領域が追加されますので、この内容を修正してください。

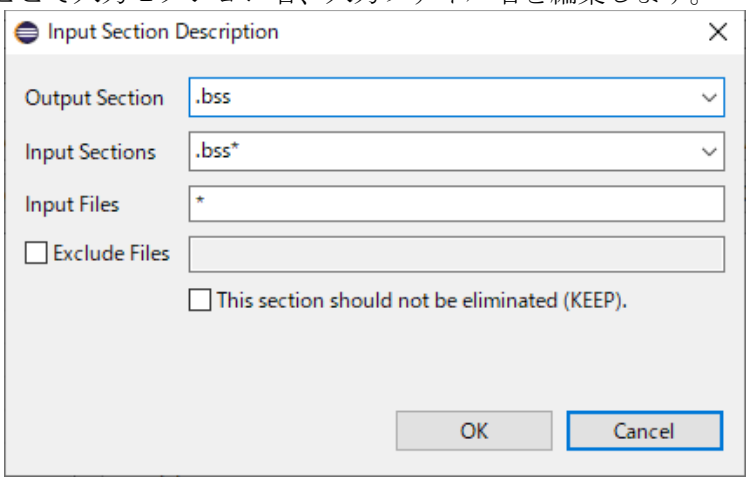
(4)　Output sections and their input patterns
SECTIONS コマンドで記述する出力セクションと入力セクションを入力します。

◆　出力セクションおよび入力セクションを変更する場合

・対象の出力セクション名をダブルクリックすると[Output Section Description]ダイアログボックスが表示されますので、ここで出力セクション名、VMA 領域名、LMA 領域名(オプション)、アライメント(オプション)、埋め込み値(オプション)を編集します。

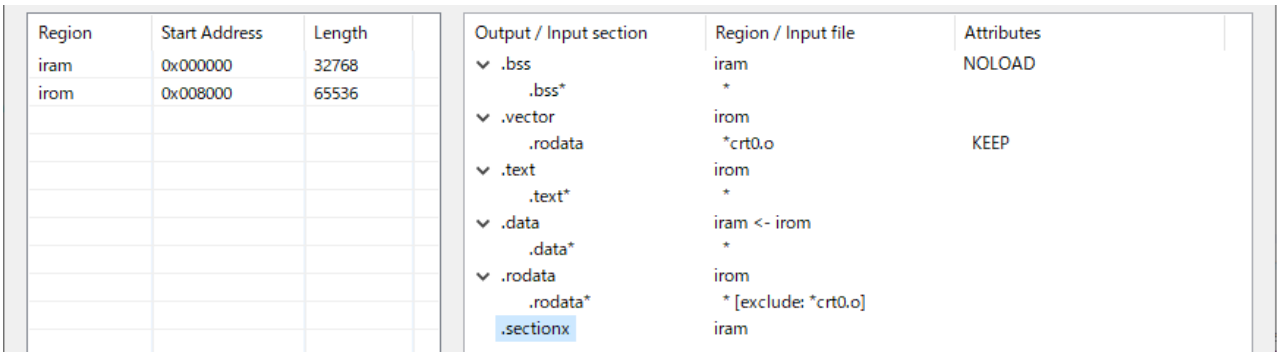


・対象の入力セクション名をダブルクリックすると[Input Section Description]ダイアログボックスが表示されますので、ここで入力セクション名、入力ファイル名を編集します。

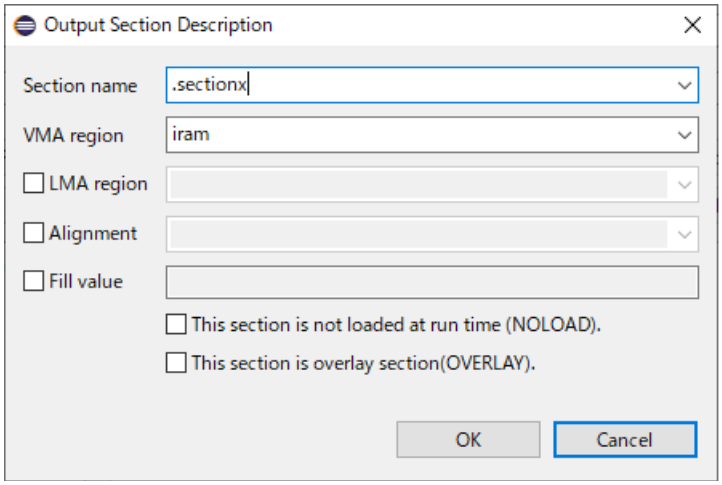


◆　出力セクションおよび入力セクションを追加する場合

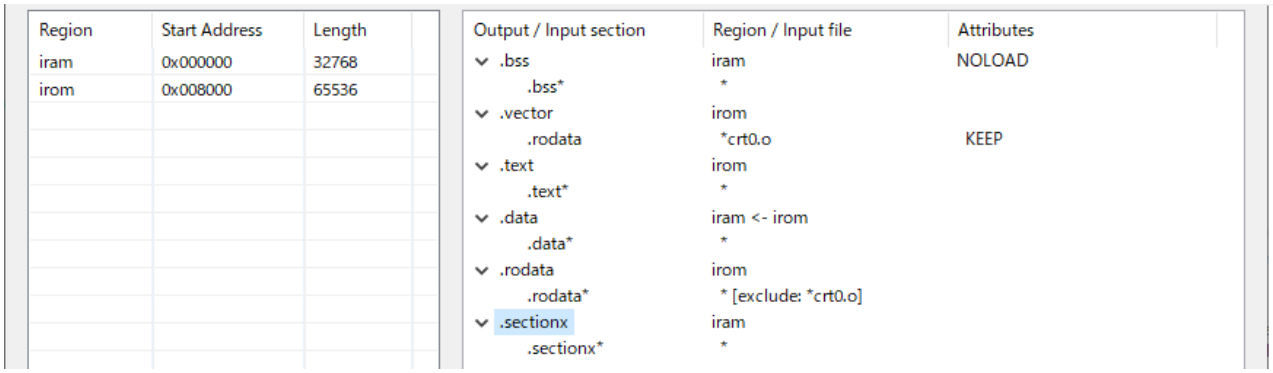
1.[Add Output Section]ボタンをクリックします。出力セクション名 ＝ “.sectionX”、VMA 領域名 ＝ “iram” というセクション定義が追加されます。



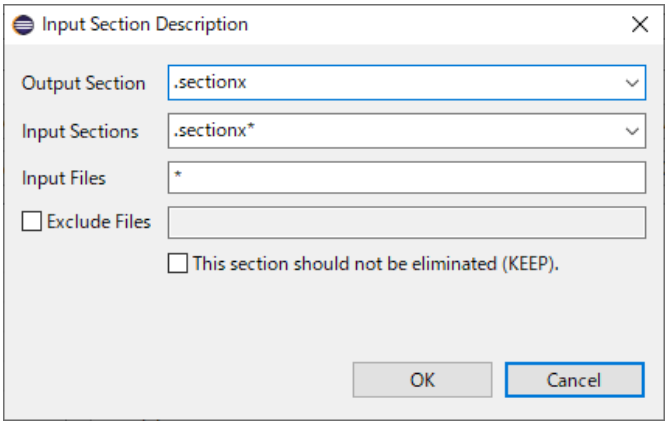
2. “.sectionX” をダブルクリックすると[Output Section Description]ダイアログボックスが表示されますので、ここで出力セクション名、VMA 領域名、LMA 領域名(オプション)、アライメント(オプション)、埋め込み値(オプション)を編集します。
また、出力セクションの NOLOAD 属性および OVAERLAY 属性は、チェックボックスで選択して定義します。



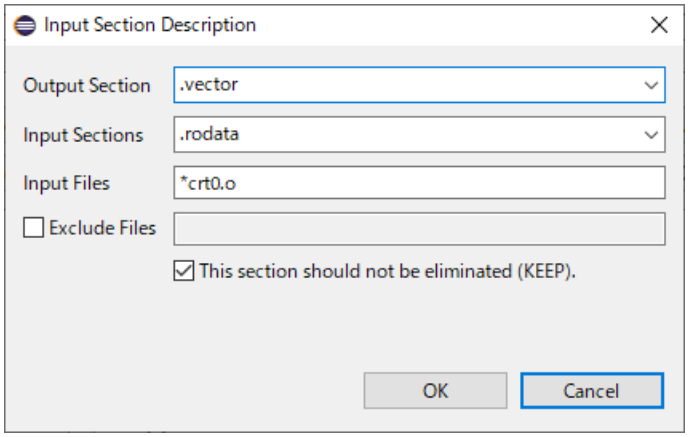
3.出力セクション名を選択後、[Add Input Pattern]ボタンをクリックすると、出力セクションに対する入力セクションが追加されます。



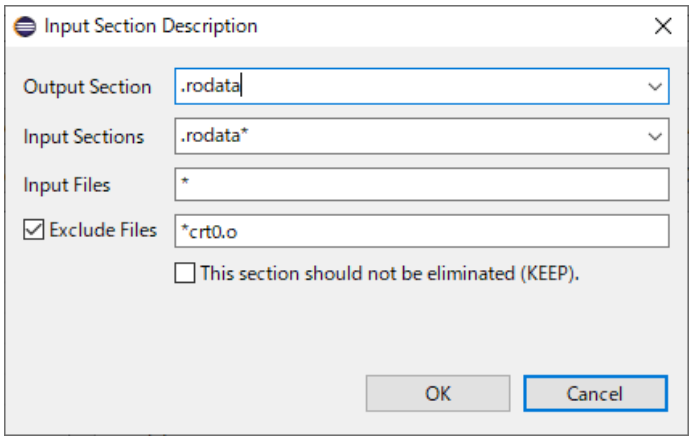
4.入力セクション名をダブルクリックすると[Input Section Description]ダイアログボックスが表示されますので、ここで入力セクション名、入力ファイル名を編集します。



KEEP や EXCLUDE_FILE の定義は以下のように設定します。



“KEEP (*crt0.o(.rodata))” 定義



“*(EXCLUDE_FILE (*crt0.o) .rodata)” 定義

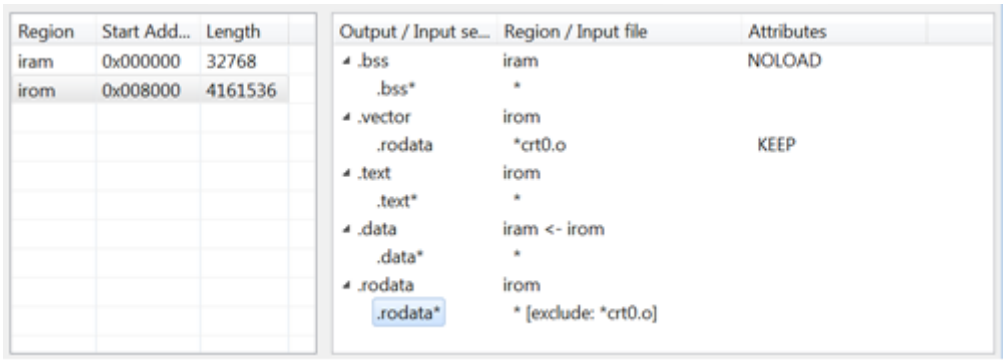
以上の操作を、定義するセクション数分繰り返します。
ウィザードで定義できない内容については、生成されたリンカスクリプトファイルをエディタで開き、編集してください。

◆　出力セクションおよび入力セクションを削除する場合

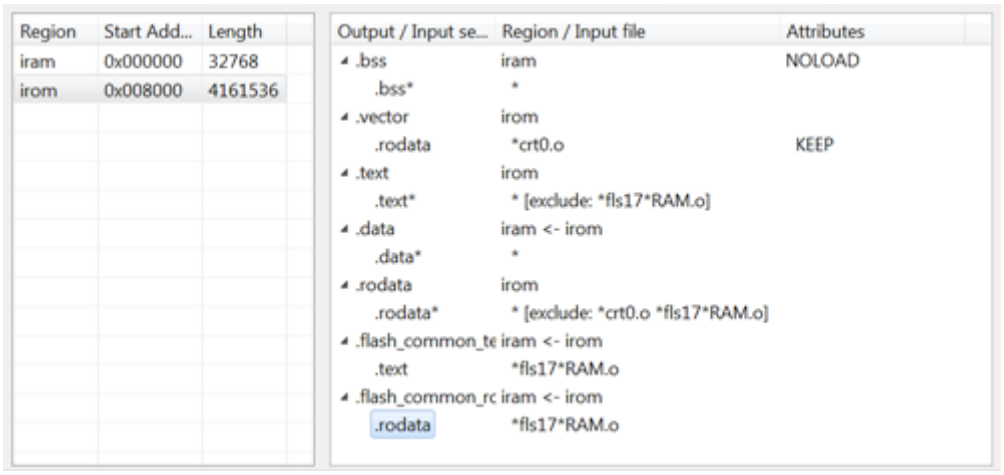
・対象の出力セクション名を選択後、[Delete Output Section]ボタンをクリックすると、出力セクションが削除されます。

・対象の入力セクション名を選択後、[Delete Input Pattern]ボタンをクリックすると、出力セクションに対する入力セクションが削除されます。

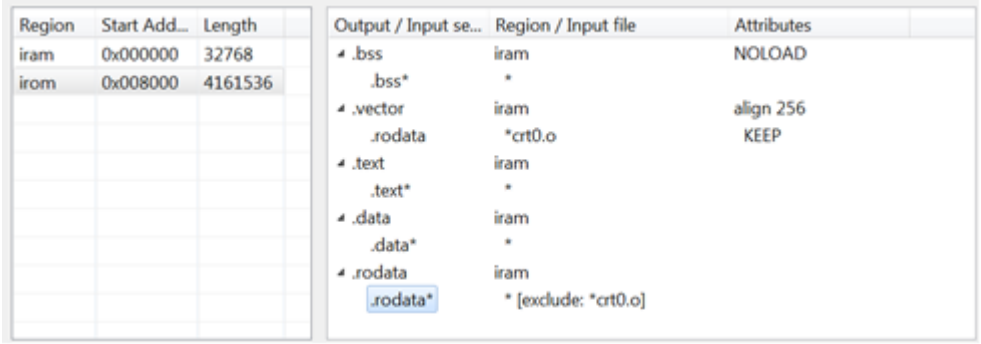
以下に[GNU17 New Linker Script]ウィザードの設定例を示します。



“A.2 リンカスクリプト”に記載の標準的なリンカスクリプト相当の定義例



自己書き換えライブラリ使用時のリンカスクリプト定義例



RAM 実行時のリンカスクリプト定義例

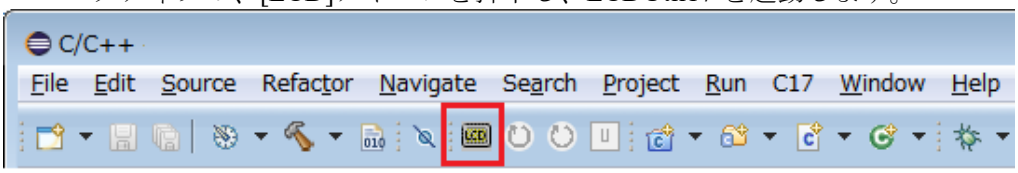
Appendix B LCD パネルシミュレータ

本デバッガには、LCD パネルシミュレータ機能が搭載されています。LCD パネルシミュレータは、PC 上で LCD パネルの表示をシミュレートする機能を提供します。この機能は、プログラムを実機上で動作させ、LCD パネルの表示部分のみ PC 上でシミュレートします。そのため、LCD パネルが搭載されていない実機に対し、PC 上で LCD パネル表示の確認が行えます。

B.1　LCD パネルカスタマイズツール(LCDUtil17)での LCD パネル作成方法

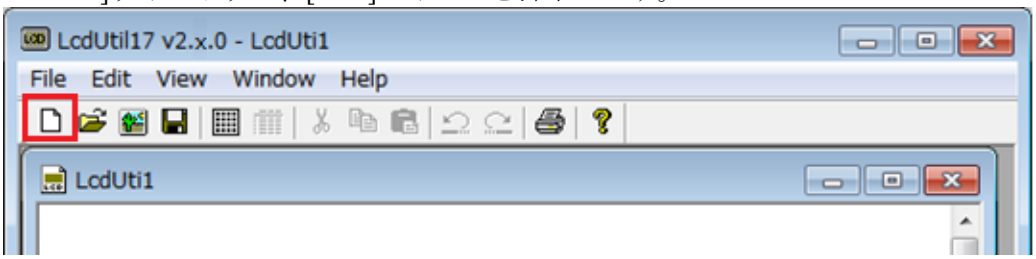
LCD パネルシミュレータで使用するシミュレート用の LCD パネルを、LCD パネルカスタマイズツール(LCDUtil17)を使って作成します。

[GNU17]パースペクティブの、[LCD]アイコンを押下し、LCDUtil17 を起動します。

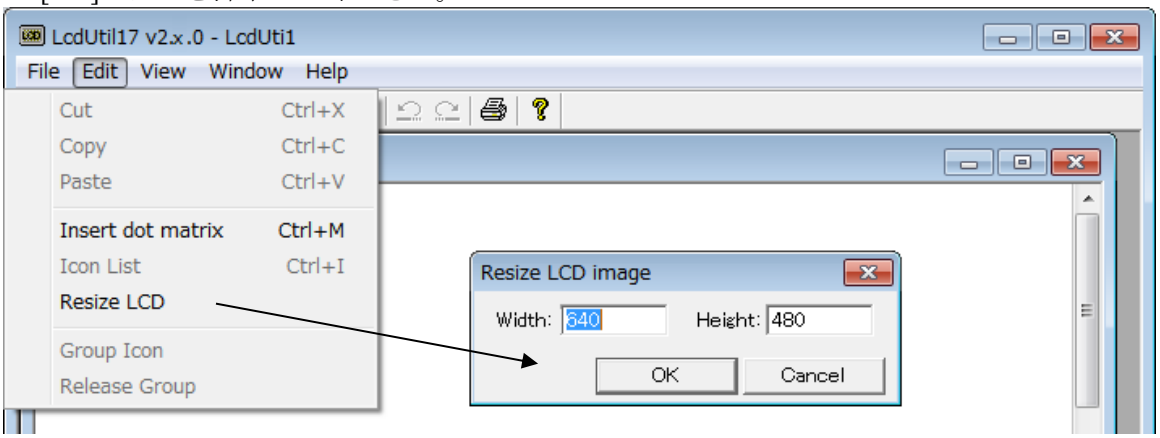


ドットマトリクス用の LCD パネルを作成する

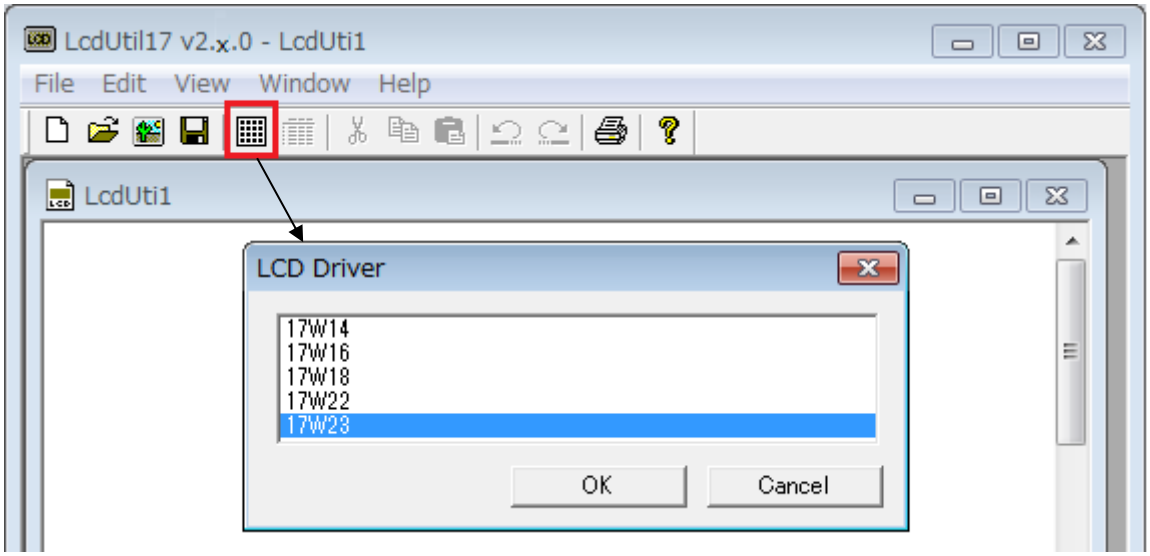
操作 1: [LcdUtil17]ウインドウで、[New]アイコンを押下します。



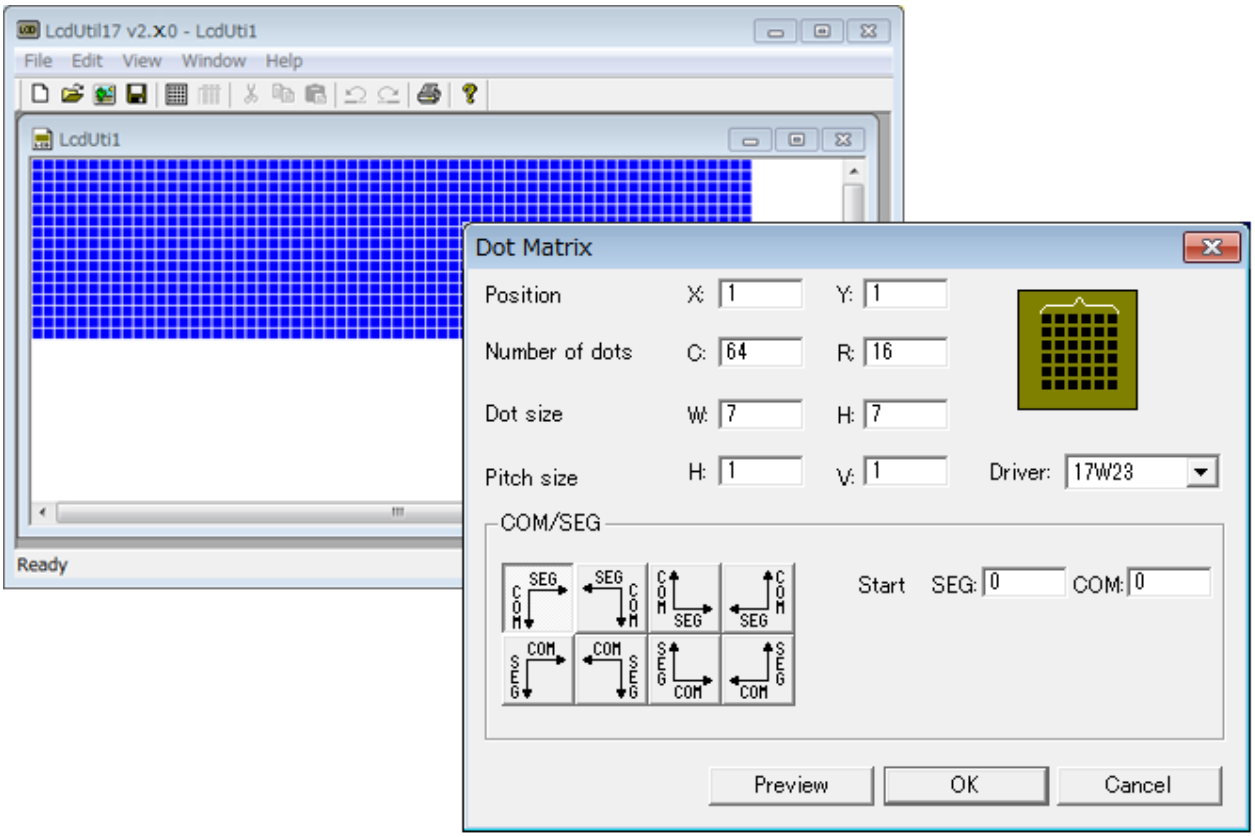
操作 2: [Edit]>[Resize LCD]で、[Resize LCD image]ウィンドウを開き、LCD パネルのサイズを入力し、[OK]ボタンを押下してください。



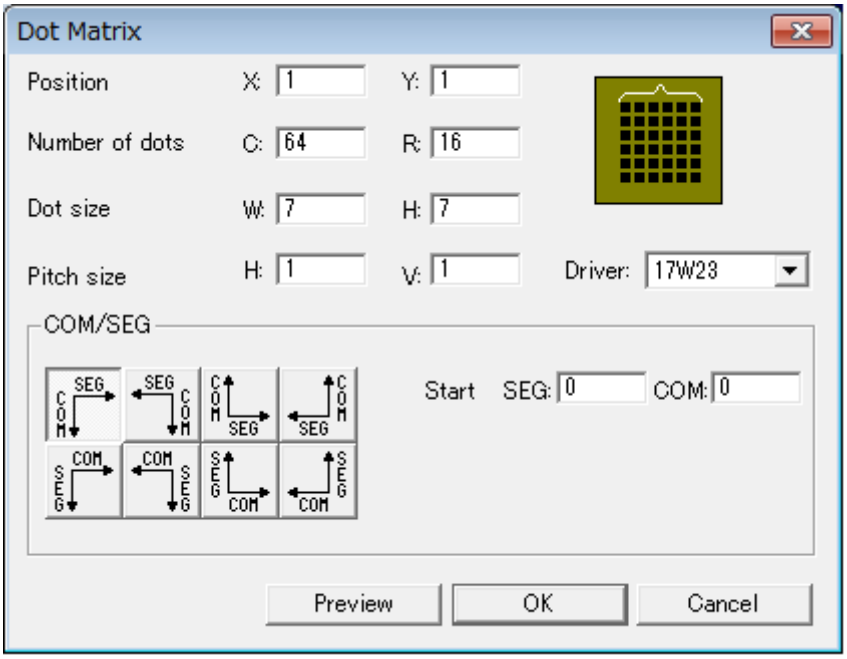
操作 3: [Dot Matrix]アイコンを押し、機種を選択した後、[OK]ボタンを押下してください。



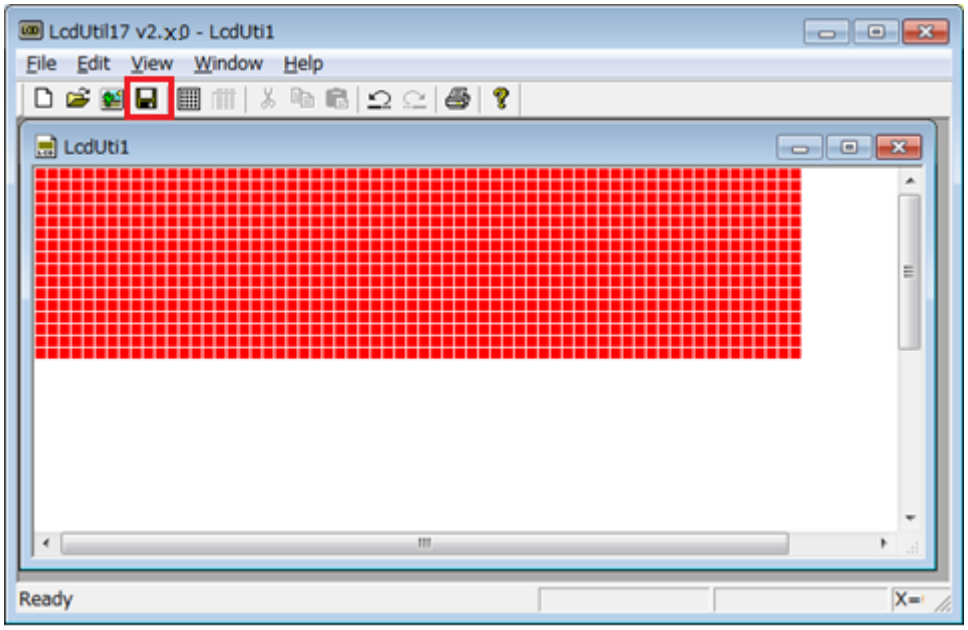
操作 4: LCD パネルの詳細なレイアウト設定を行います。
[Dot Matrix]画面の右側に表示されるレイアウトを参考にしながら設定後、[Preview]ボタンを押下してください。



操作 5: Preview 内容でよければ、[OK]ボタンを押して確定してください。

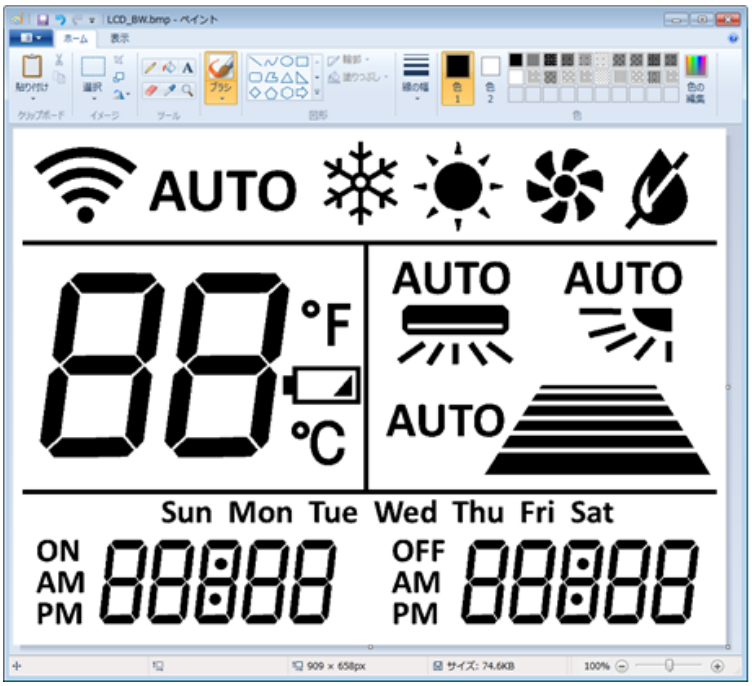


操作 6: [Save]アイコンで、作成した LCD パネルをファイル(*.lcd)に保存します。

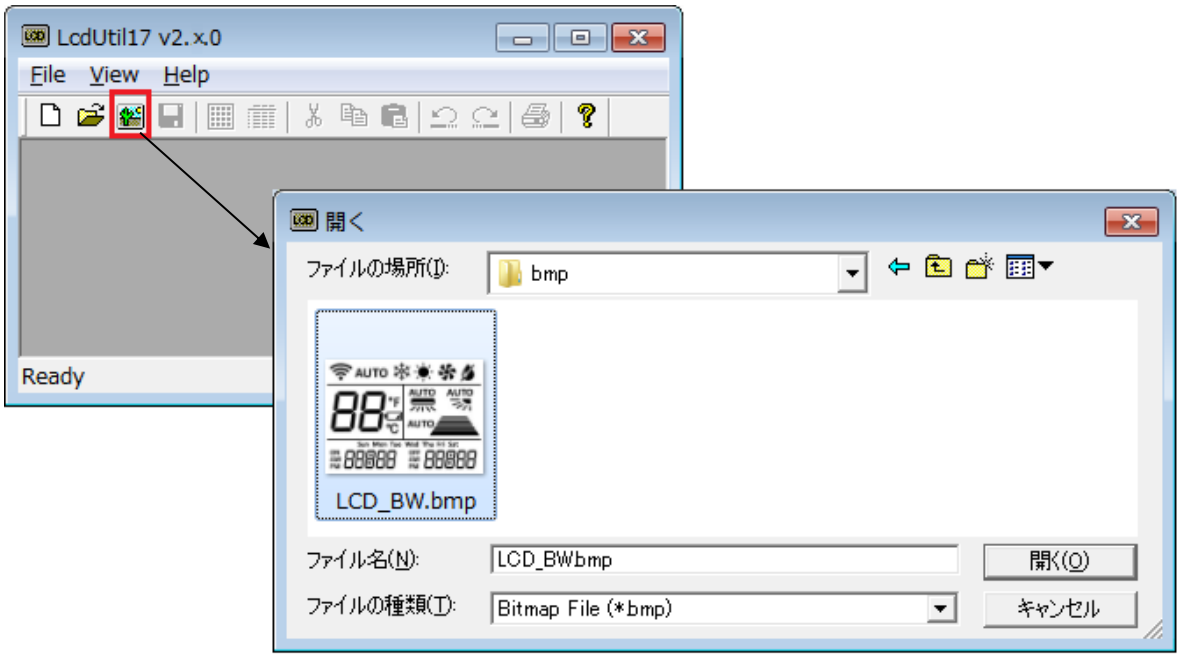


セグメント用の LCD パネルを作成する

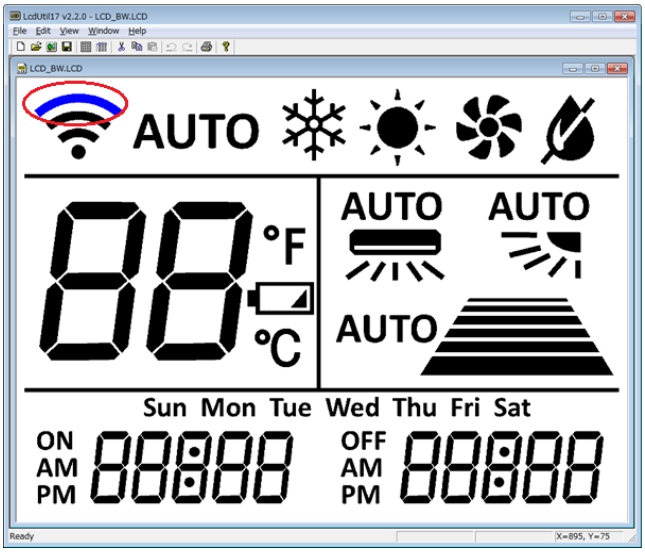
操作 1: ペイントツール等で、モノクロビットマップファイル（＊.bmp）を作成します。
白黒の 2 色のみを使用してください。
注：別セグメントとして登録する各キャラクタは、十分な隙間をあけて配置してください。
近すぎると 1 つのセグメントとして認識されます。



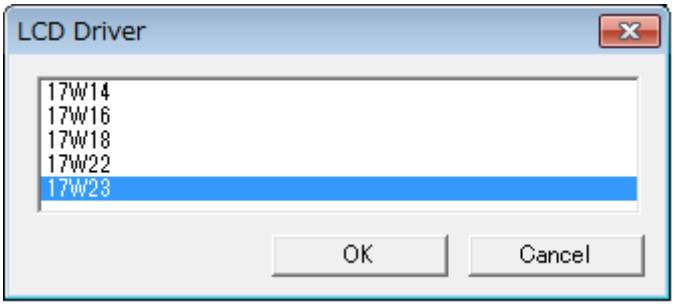
操作 2: [LcdUtil17]ウインドウで、[Bitmap]アイコンを押下し、ビットマップファイルを読み込んでください。



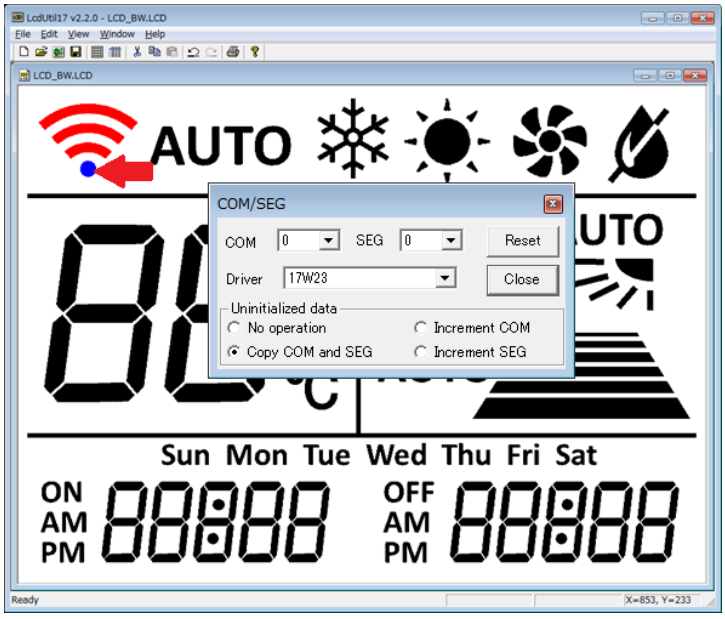
操作 3: いずれかのキャラクタをマウスカーソルで選択後、ダブルクリックしてください。



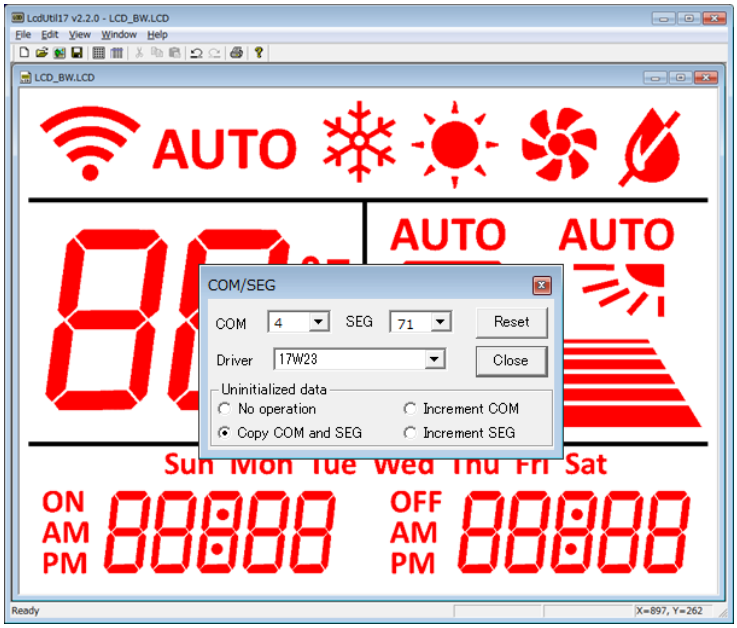
操作 4: [LCD Driver]ウィンドウが開いたら、機種を選択し [OK]ボタンを押下します。



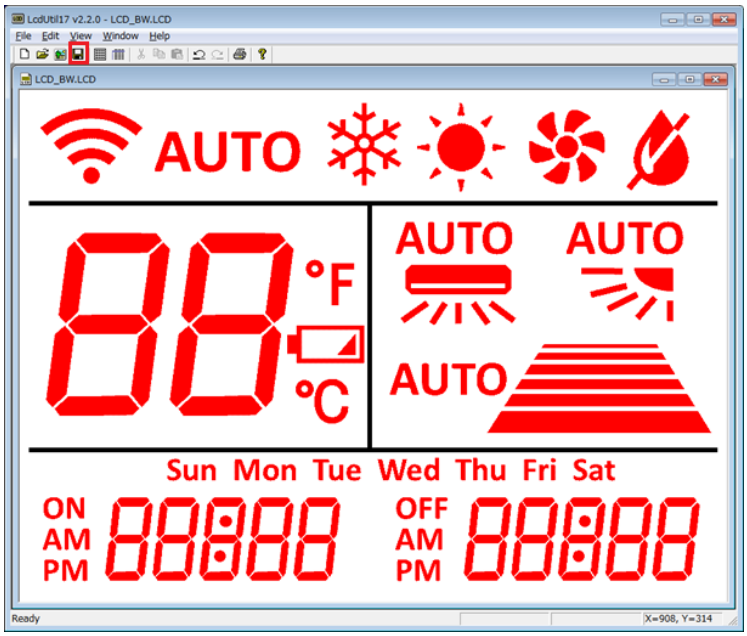
操作 5: 各キャラクタを選択し、割り当てる COM/SEG を設定します。
（複数のキャラクタに、同じ COM/SEG を設定することも可能です。）



操作 6: すべての設定が終わったら、[Close]ボタンで[COM/SEG]ウィンドウを閉じます。



操作 7: [Save]アイコンで、作成した LCD パネルをファイル(*.lcd)に保存します。



B.2　LCD パネルシミュレータを使ったシミュレート方法

LCD パネルカスタマイズツール(LCDUtil17)で作成した LCD パネルのファイル(.lcd)を使って、LCD パネルシミュレータで LCD パネルのシミュレートをします。本機能は、実機を併用するため、デバッグモードは ICD モードで動作します。

事前準備

操作 1: シミュレート用の LCD パネルを、LCD パネルカスタマイズツール(LCDUtil17)を使って作成します。作成については、前章を参照してください。

操作 2: 機種情報フォルダ(GNU17V3/mcu_model/17xxx)内に、必要ファイルが含まれていることを確認します。

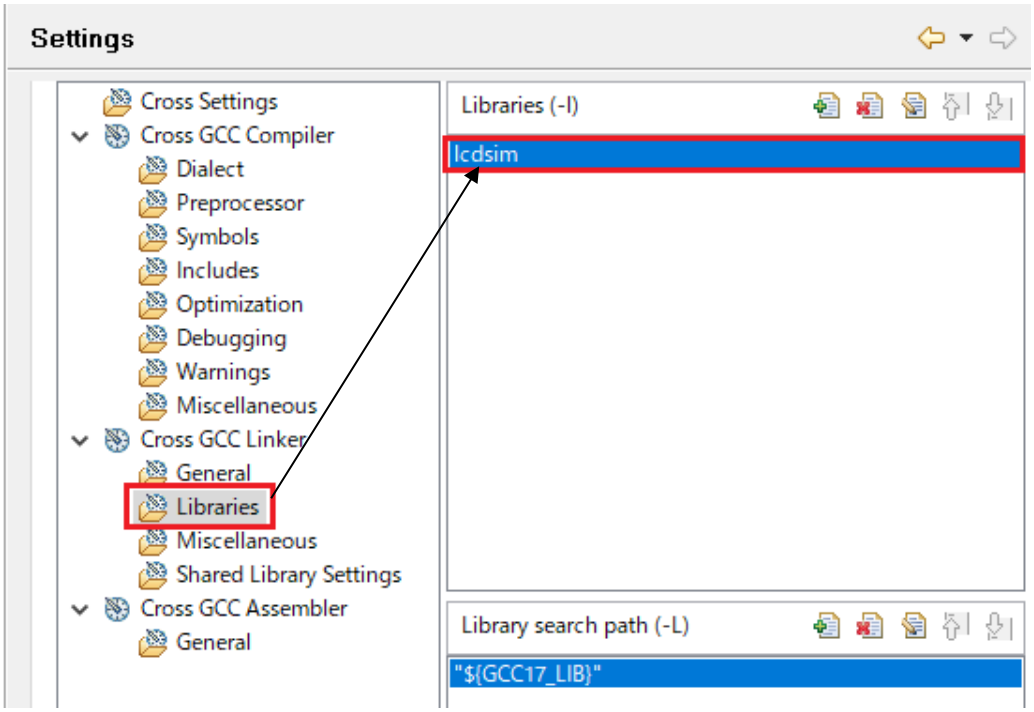
・essim17.ini
・essim17_user_def.ini
・lcdDisplaySim17xxx.dll

操作 3: GDB コマンドファイルを編集します。
デバッグモードは ICD モードを使用します。使用するデバッグ環境に合わせ gdbmini2.ini もしくは gdbmini3.ini のいずれかを編集してくださいここでは gdbmini3.ini を使用した例を示します。

```
# Initial GDB command file for ICDmini3
# set output-radix 16
c17 model_path C:/EPSON/GNU17V3/mcu_model
c17 model 17xxx
target icd icdmini3
load
# Please uncomment following commented out lines to enable STDOUT while debugging.
# c17 stdout 1 WRITE_FLASH WRITE_BUF
# Please uncomment following commented out lines to enable STDIN while debugging.
# c17 stdin 1 READ_FLASH READ_BUF
# Please uncomment following commented out lines to enable LCD panel simulator while debugging.
c17 lcdsim on                                   …c17 lcdsim コマンドを有効にします。
```

ユーザプログラムを編集

操作 4: ライブラリを追加します。
[Project Explorer]ビューの対象プロジェクトを選択した状態で、[Project]メニュー(または右クリックで表示されるコンテキストメニュー)から[Properties]ダイアログ＞C/C++ Build＞Settings＞[Tool Settings]＞[Cross GCC Linker]＞[Libraries] に LCD パネルシミュレータライブラリ“lcdsim”を追加してください。



GNU17V3.2 以降を使用して作成したプロジェクトは、標準で登録されています。

操作 5: 対象のプログラムファイルに、LCD パネルシミュレータライブラリのインクルードファイル(lcdsim.h)を追記します。

```
#include "lcdsim.h"
```

操作 6: LCD パネルシミュレータ表示更新関数(lcdupdate();)を、LCD ドライバ制御レジスタの操作後に追記します。追記をすることで、LCD ドライバ制御レジスタに書かれている内容を、LCD パネルシミュレータへ反映します。以下は、S1C17W22 を使用した例です。

```
例：　LCD24DSP.DSPC = 0x2　　// LCD 全点灯
      lcdsimUpdate();　　// LCD ウィンドウの表示更新を行います
      LCD24DSP.DSPC = 0x0　　// LCD 全消灯
      lcdsimUpdate();　　// LCD ウィンドウの表示更新を行います
      ※LCD24DSP.DSPC ビットは、LCD の表示状態を制御するレジスタです。
```

注: 最終的にターゲットシステム上に LCD パネルが搭載し動作確認を行うために、プロジェクトを最終状態にする場合は、LCD パネルシミュレータライブラリのインクルード命令、LCD パネルシミュレータ表示更新関数はプログラム中から削除してください。

LCD パネルシミュレータの起動

操作 7: PC と ICDmini(S5U1C17001H*)およびターゲットシステムを接続します。以下は、ICDminiV3 を使用した例です。

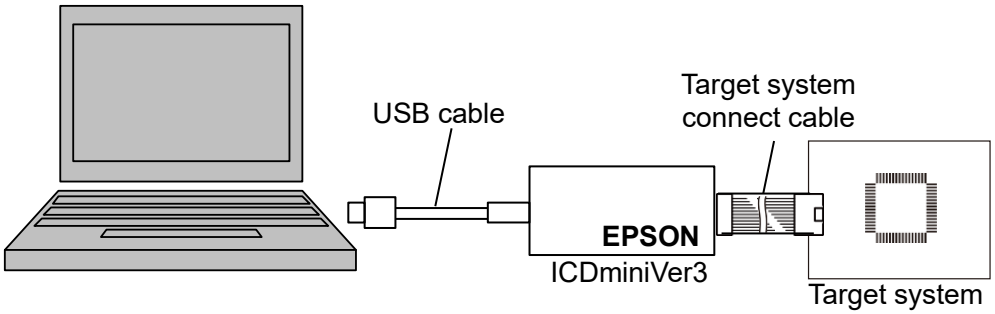
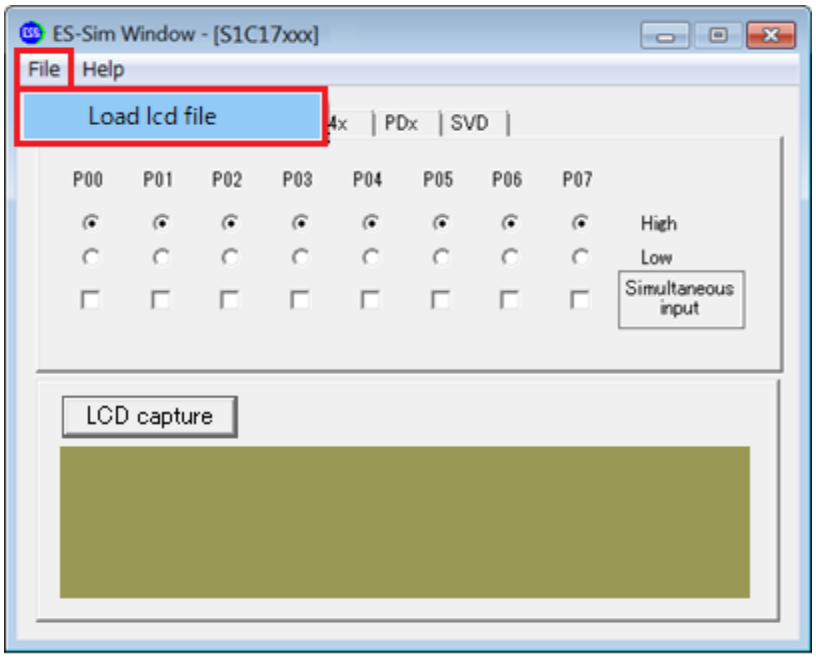


図 B.2.1 ICDminiVer3 を使用するデバッグシステム例

操作 8: [Debug Configurations]ダイアログボックスから、デバッガを起動します。
デバッガの詳細な起動方法については、”2.7 デバッガ”の章をご参照ください。

操作 9: デバッガの起動とともに、周辺回路シミュレータ(ES-Sim17)ウィンドウが立ち上がります。
ES-Sim17 ウィンドウの[File]>[Load lcd file] から、LCDUtil17 で作成した LCD ファイル(.lcd)を開きます。



本パッケージには、サンプルの.lcd ファイルが同梱されています。参考にこちらを使用することも可能です。
C:¥EPSON¥GNU17V3¥utility¥LcdUtil17¥sample

操作 10: プログラムを実行します。
ツールバー (Resume)ボタンをクリック(または[Run]メニュー ＞[Resume]を選択)します。
LCD パネルシミュレータ表示更新関数が実行されると、ES-Sim17 ウィンドウに LCD パネルのシミュレーションが表示されます。

Appendix C ローカライズ(参考)

GNU17 IDE は Eclipse IDE for C/C++ Developers Package を元に作成されており、インストールされるワークベンチのユーザインタフェースはすべて英語で表示されます。他言語で表示させたい場合は、言語パックを導入してください。
以下に、GNU17 IDE に Babel プロジェクトの言語パックを適用する例を示します。

操作 1: 以下の Babel プロジェクトのアーカイブサイトにアクセスします。
http://archive.eclipse.org/technology/babel/babel_language_packs/R0.15.1/mars/mars.php
注：上記 URL は変更されている可能性があります。
Eclipse の Web サイトより、最新の URL を確認してください。

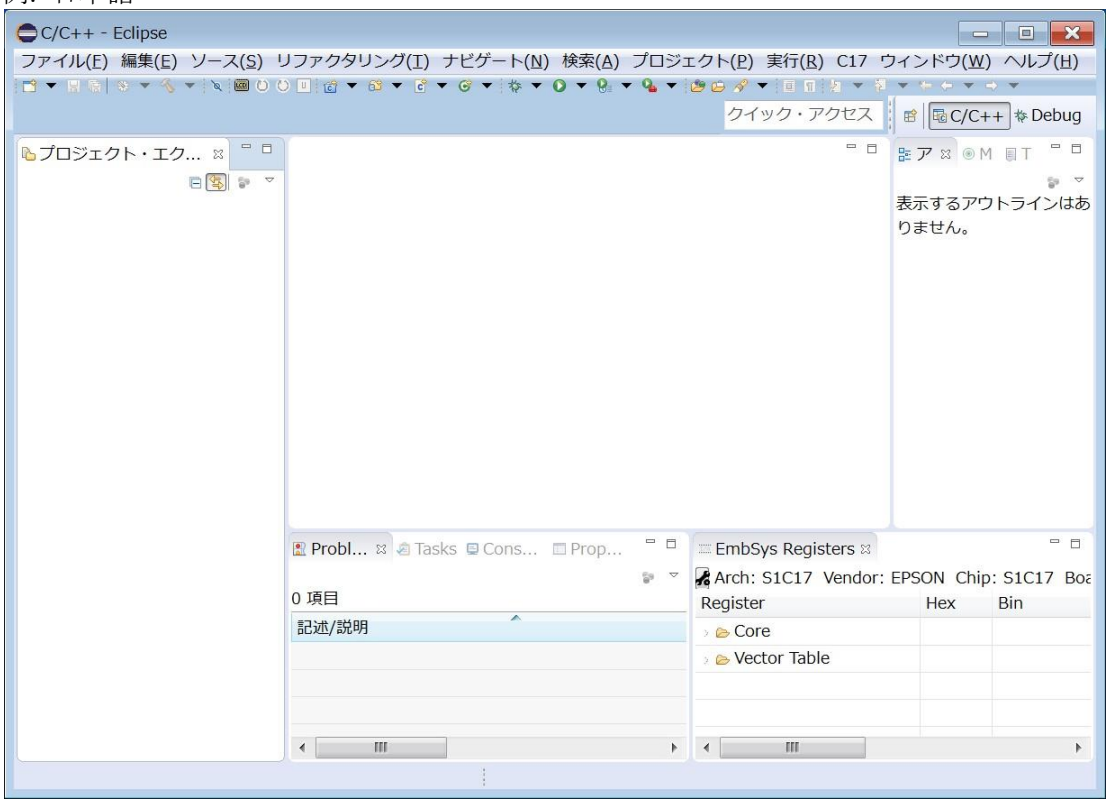
操作 2: 希望する言語のプラグインをダウンロードします。ここでは日本語化を例に、以下の 2 つのファイルをダウンロードします。
・BabelLanguagePack-eclipse-ja_4.5.0.v20171231064042.zip
・BabelLanguagePack-tools.cdt-ja_4.5.0.v20171231064042.zip

操作 3: ダウンロードしたファイルを解凍し、eclipse 以下を直接次のフォルダにコピーします。
C:¥EPSON¥GNU17V3¥eclipse

操作 4: IDE を再起動し、変更を適応してください。初回の起動は遅いのでご注意ください。

注：適応されない場合、Windows コマンドプロンプトから以下を指定して起動すると反映されます。
C:¥EPSON¥GNU17V3¥eclipse>eclipse -clean

再起動後、メニューなどがインストールした言語で表示されます。
例: 日本語



言語を指定して IDE を起動するには

言語パックを導入後に英語に戻す、あるいは複数の言語パックを導入後に特定の言語で起動させるには、以下のような方法があります。

1. 起動コマンドで指定

単発的に特定の言語に設定するには、コマンドプロンプトから“eclipse.exe -nl (言語)”の形式で IDE を起動します。

例: 英語

eclipse.exe -nl en

複数の言語を使い分けたい場合は、言語数分の eclipse.exe のショートカットを作成し、そのプロパティの[リンク先]に上記の-nl オプションを追加してください。

例: C:¥EPSON¥GNU17V3¥eclipse¥eclipse.exe -nl en

2. eclipse.ini で指定

通常使用する言語は、C:¥EPSON¥GNU17V3¥eclipse ディレクトリ内の eclipse.ini ファイルに以下の言語指定を追加してください。

-Duser.language=(言語)

-Duser.country=(国)

例: 英語(米国)

-Duser.language=en

-Duser.country=US

改訂履歴表

付－1

Rev. No.	日付	ページ	種別	改訂内容（旧内容を含む） および改訂理由
Rev 1.0	2019/09/09	All	新規	新規制定
Rev 1.1	2023/04/11	P57-P63	改訂	3.2 GNU17Ver.2.x プロジェクトのインポートを変更
		P68-P69	改訂	A.3 リンカスクリプト例を追加

セイコーエプソン株式会社

営業本部　ＭＤ営業部

東京　〒160-8801 東京都新宿区新宿 4-1-6 JR 新宿ミライナタワー

大阪　〒530-6122　大阪市北区中之島 3-3-23　中之島ダイビル 22F

ドキュメントコード：413880501

2019 年 06 月 作成

2023 年 04 月 改訂