

Eclipse+OpenJTAG +OpenOCD で ARM シリーズ開発環境構築

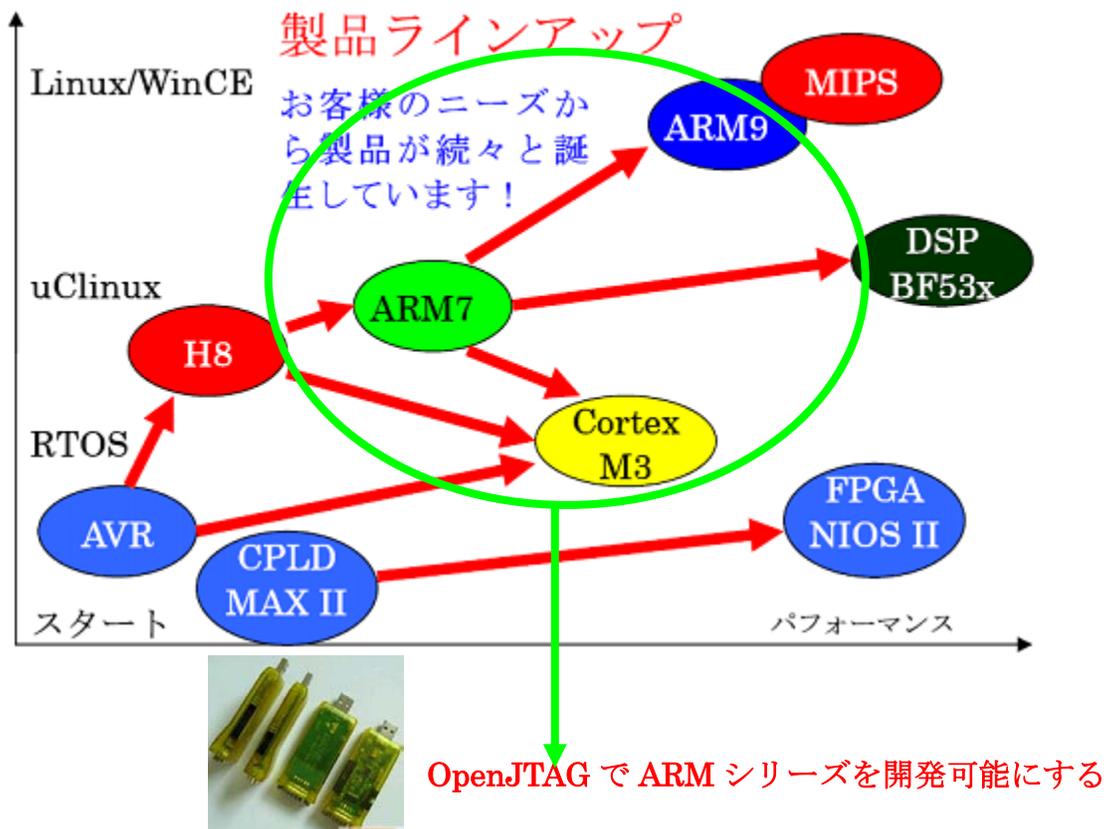
株式会社日昇テクノロジー

<http://www.csun.co.jp>

info@csun.co.jp

Ver1.3

2010/11/22



[copyright@2009](http://www.csun.co.jp)

※ この文書の情報は、事前の通知なく変更される可能性があります。
※ (株)日昇テクノロジーの書面による許可のない複製は、いかなる形態においても厳重に禁じられています。



修正履歴

NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2009/07/31
2	Ver1.1	誤った環境構築用、ワーク用のパスを訂正	2010/07/17
3	Ver1.2	P58～P63 の説明画面の訂正	2010/07/20
4	Ver1.3	フォルダ名誤記訂正 ARM EABI「arm-none-eabi-gcc」確認コマンド誤記訂正 5.2章 Zylind Embedded CDT のインストール手順変更	2010/11/22



第一章	背景	4
第二章	ARM シリーズ開発の仕組み・イメージ	5
第三章	用意するもの	6
第四章	インストール手順	7
4.1	OpenJTAG のドライバをインストールする	7
4.2	ソフトウェアをインストールする	10
4.2.1	「arm-none-eabi.exe」をインストール	11
4.2.2	「openocd」をインストール	15
4.2.3	「yagarto」をインストール	19
4.2.4	「yagarto-tools」をインストール	23
4.2.5	Cygwin のダウンロード&インストール	26
4.2.6	「jre」をインストール	29
4.2.7	「Eclipse」をインストール	29
4.3	ソフトウェアインストール後の動作確認	30
4.3.1	OpenOCD の確認	30
4.3.2	コンパイラ確認	31
第五章	Eclipse の設定	35
5.1	Eclipse を起動する	35
5.2	Eclipse プラグイン(Zylin Embedded GDT)インストール	37
5.3	ビルドの設定	38
5.4	OpenOCD の設定	39
5.5	Cygwin のソース・ルックアップ・パスの設定	43
5.6	Eclipse デバッグ用のコマンド及びショットカットキー一覧	46
第六章	ARM シリーズデバッグ手順	47
6.1	ARM7 の LPC2148	47
6.1.1	LPC2148 ボード購入 URL	47
6.1.2	ハードウェア動作確認	47
6.1.3	LPC2148 用のサンプル「LED」をデバッグ	48
6.2	ARM7 の LPC2388	69
6.2.1	LPC2388 ボード購入 URL	69
6.2.2	ハードウェア動作確認	69
6.2.3	LPC2388 用のサンプル「LED」をデバッグ	71
6.3	ARM Cotex-M3 の STM32F103	85
6.3.1	STM32F103 ボード購入 URL	85
6.3.2	ハードウェア動作確認	85
6.3.3	LED サンプルデバッグ	86
6.4	ARM9 の MINI2440	101
6.4.1	MINI2440 を購入	101
6.4.2	ハードウェア動作確認	101
6.4.3	LED サンプルデバッグ	102
6.4.4	u-boot サンプルデバッグ	128



第一章 背景

近年、ARM プロセッサが急速に広まっています。さらに、ARM9, ARM11, ARM-M, Cortex といった新たなアーキテクチャが次々と発表されています。これらの新しいプロセッサでは、従来から存在したARMが拡張され、さらにARM9とARM11が追加されました。

本書では、従来から広く利用されているGNU クロス開発ツールを、これらの最新アーキテクチャ/命令セットへ対応させます。最新版のGNU ツールを実際を使って、対応したクロス開発環境を作ります。

組込アプリを開発する際に、2点は非常に重要です。一つは統合開発環境(IDE)となり、もう一つはデバッグ環境です。いろんな統合開発環境ツールがありますが、無料で使用できるオープンソースEclipseプラットフォームはJava アプリを含めて、C/C++アプリを開発用の全世界共通のIDEです、今後、ますます多くの使用者が増えていきます。なお、デバッグツールはオープンソースOpen-On-Chip Debugger(OpenOCD)という便利ツールがあります。

Open-On-Chip Debugger(OpenOCD)は、システムプログラミングにおけるデバッグと埋め込まれた対象装置がないかどうかテストされる境界走査を提供することを目指します。

OpenOCDのGDB と接続するためにOpen JTAG コネクトを使う必要があります。ただし、パラレルJTAGの転送スピードが遅いので、本書はカスタマイズ化のOpenJTAGを利用し、Eclipse、OpenOCDなどオープンソースとあわせて、ARMシリーズ開発を行います、例えば、ARM基板からダウンロード、書き込み、デバッグを行います。

GDBは以下の基板をデバッグできます。

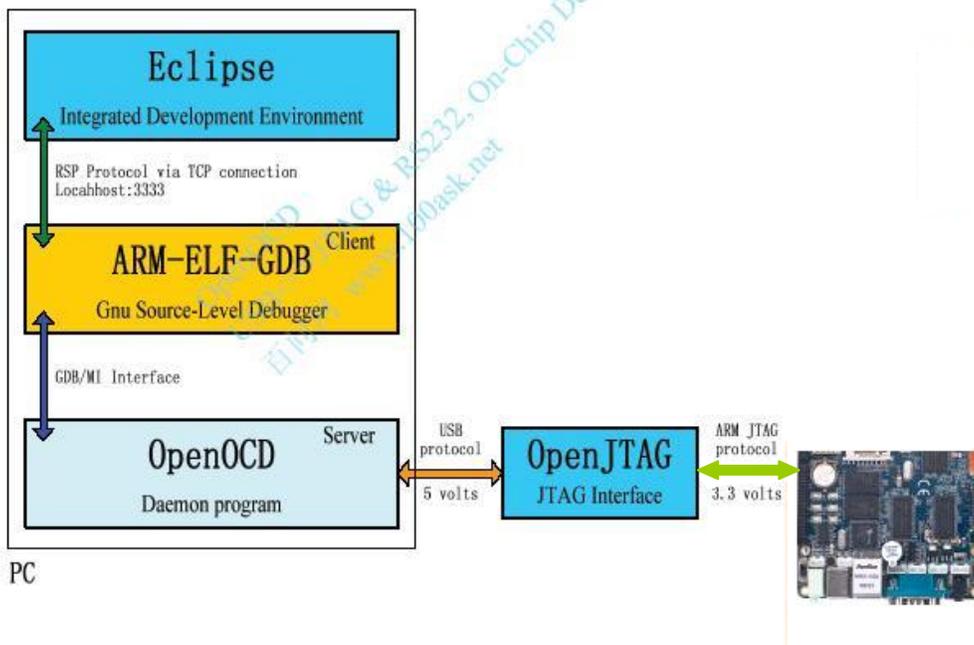
- ① ARM7(ARM7TDMIとARM720t)
- ② ARM9、(ARM920t、ARM922t、 ARM926ej-s、ARM966e-s)
- ③ XScale(PXA25x、IXP42x)
- ④ Cortex-M3(LM3とST STM32)

第二章 ARM シリーズ開発の仕組み・イメージ

組込開発のよいプラットフォームは基本的に4つモジュールを含めます。

- ① 統合開発環境: IDE (Integrated Development Environment)、例えばEclipse
- ② クロスコンパイラツールチェーン (Cross Compiler tools chain)
- ③ リモートデバッグツール (例: OpenOCD、Open On-Chip Debugger)
- ④ 開発用のパソコンとARM基板のコネクタ (JTAG)

全体のイメージは下記の図です。





第三章 用意するもの

以下は Windows 用です。

3.1 ソフトウェアダウンロード URL:

弊社のサーバーから全て必要のものを一括ダウンロードできます。

<http://www.dragonwake.com/download/open-jtag/openJTAGIDE.zip>

中身に、下記内容を含めます。

ソフトウェアリスト(名前をクリックしてからツールもそれぞれダウンロードできます)

* 解凍先: お好きな場所を選べられても OK(例: D:¥tmp)

[01.arm-2009q1-161-arm-none-eabi.exe](#)

[02.openocd-0.1.0.msi](#)

[03.yagarto-bu-2.19.1_gcc-4.3.3-c-c++_nl-1.17.0_gi-6.8.50_20090311.exe](#)

[04.yagarto-tools-20070303-setup.exe](#)

[05.Cygwin](#)

[06.jre-6u7-windows-i586-p.exe](#)

[07.eclipse-cpp-galileo-win32.zip](#)

3.2 OpenJTAG 用のデバイス(jtag デバugg、弊社で販売しているもの)

3.2.1 Open-JTAG の購入 URL:

<http://www.csun.co.jp/SHOP/200905191.html>

3.2.2 デバイスダウンロード URL:

<http://www.dragonwake.com/download/open-jtag/open-jtag-driver.zip>

①usb-driver(OpenJTAG 用の USB ドライバ)

* 解凍後: D:¥tmp¥open-jtag-driver

3.3 本書使用サンプルソースコード

http://www.dragonwake.com/download/open-jtag/gcc_openjtag_eclipse_arm_example.zip

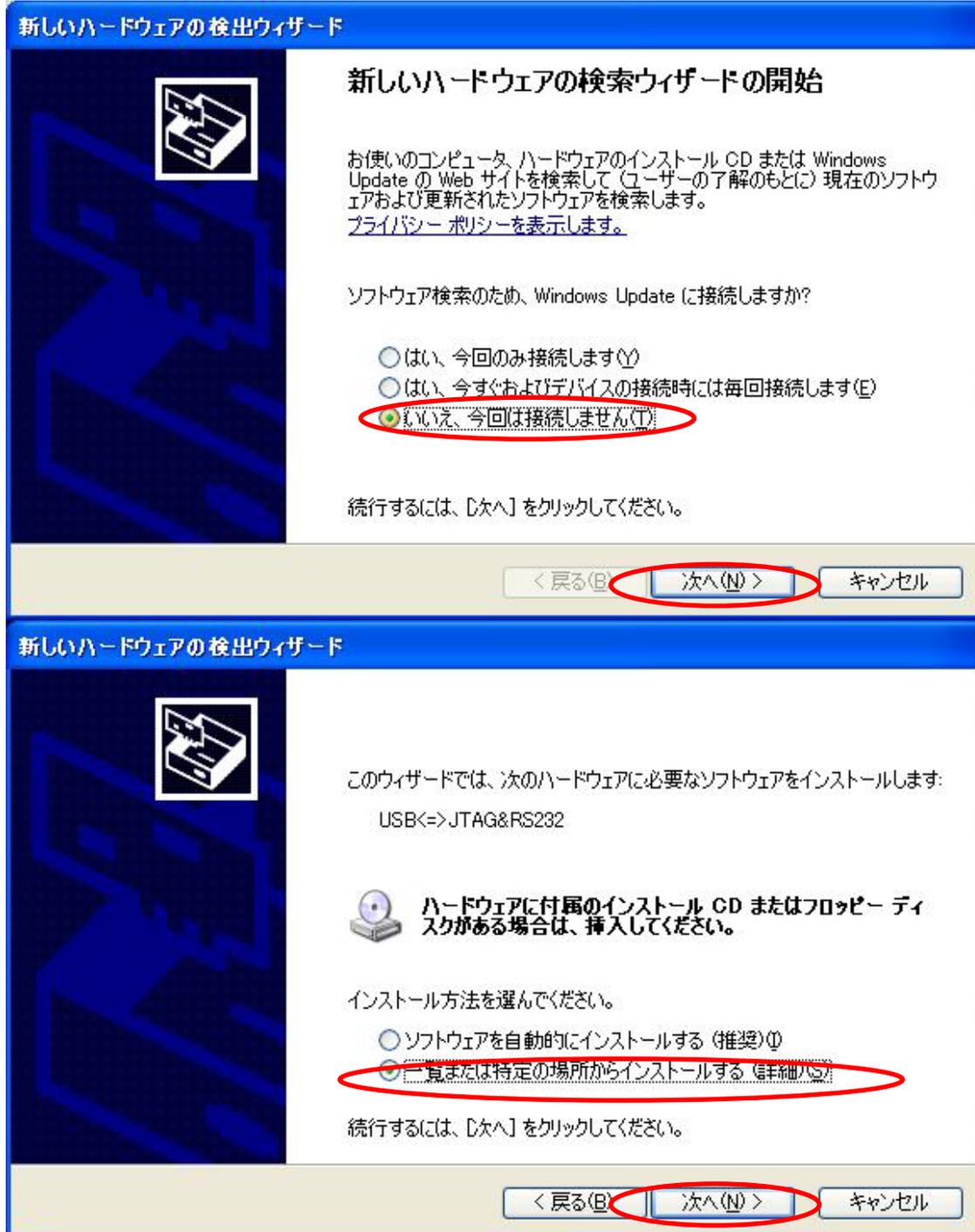
*解凍後:(D:¥OpenJTAG¥ gcc_openjtag_eclipse_arm_example)

1. 「config」: OpenOCD 用の設定ファイル
2. 「lpc2148」: LPC2148 ボード用のサンプル
3. 「lpc2388」: LPC2388 ボード用のサンプル
4. 「stm32」: stm32FX103 ボード用のサンプル
5. 「mini2440」: mini2440 ボード用のサンプル

第四章 インストール手順

4.1 OpenJTAG のドライバをインストールする

OpenJTAG をパソコンの USB ポートに挿入して、下の通りにドライバをインストールしてください。



新しいハードウェアの検出ウィザード

新しいハードウェアの検索ウィザードの開始

お使いのコンピュータ、ハードウェアのインストール CD または Windows Update の Web サイトを検索して (ユーザーの了解のもとに) 現在のソフトウェアおよび更新されたソフトウェアを検索します。
プライバシー ポリシーを表示します。

ソフトウェア検索のため、Windows Update に接続しますか?

はい、今回のみ接続します (Y)

はい、今すぐおよびデバイスの接続時には毎回接続します (E)

いいえ、今回は接続しません (N)

続行するには、[次へ] をクリックしてください。

< 戻る (B) **次へ (N) >** キャンセル

新しいハードウェアの検出ウィザード

このウィザードでは、次のハードウェアに必要なソフトウェアをインストールします。

USB<=>JTAG&RS232

 **ハードウェアに付属のインストール CD またはフロッピー ディスクがある場合は、挿入してください。**

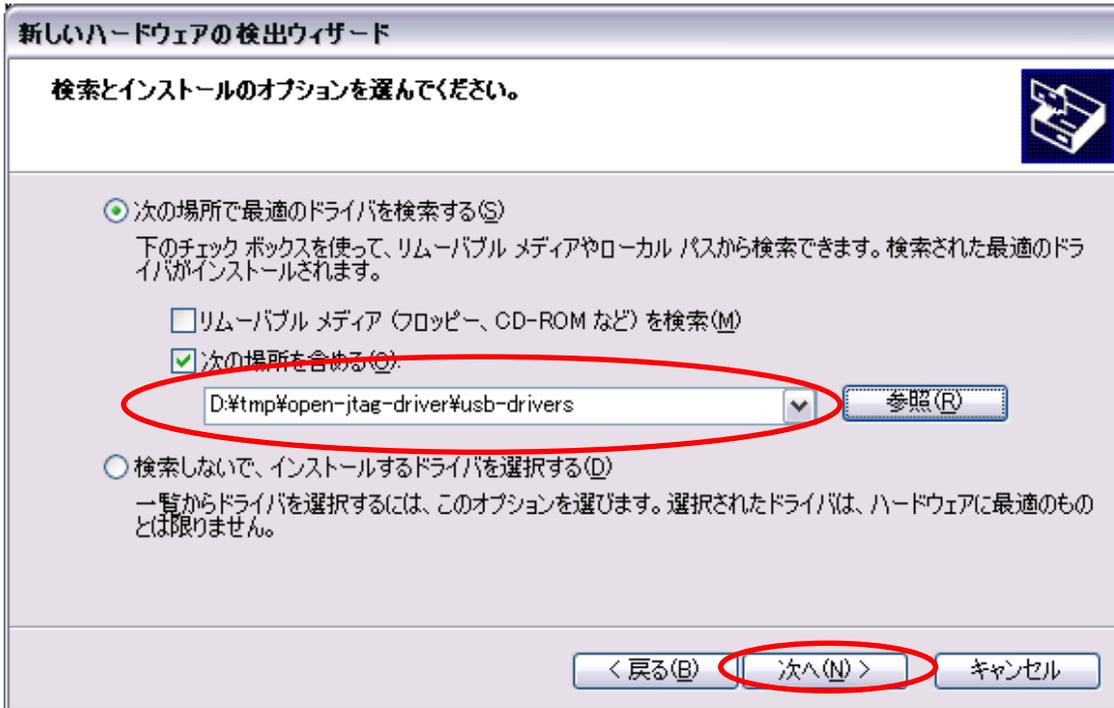
インストール方法を選んでください。

ソフトウェアを自動的にインストールする (推奨) (A)

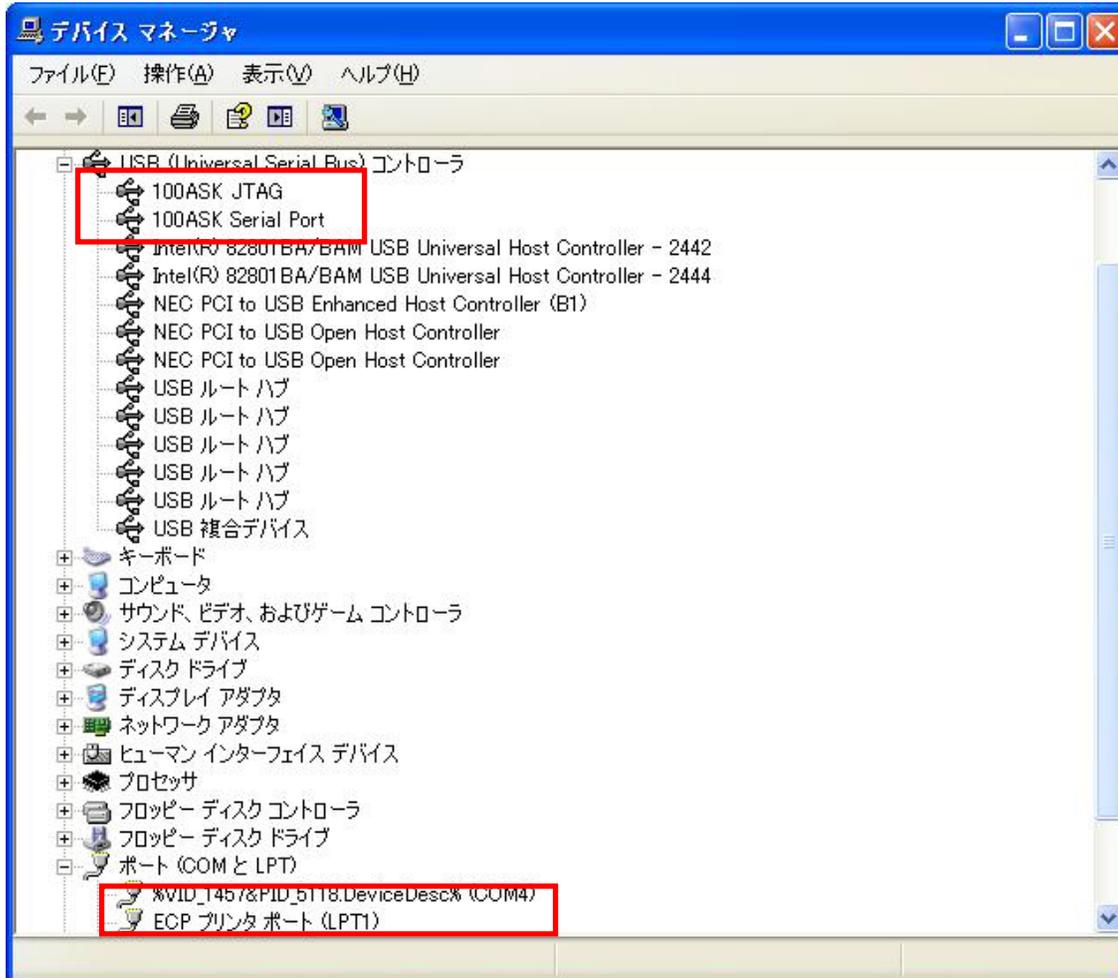
任意または特定の場所からインストールする (詳細) (S)

続行するには、[次へ] をクリックしてください。

< 戻る (B) **次へ (N) >** キャンセル



USBドライバのインストールは3回があります。インストール完了すると、デバイスマネージャで三つのデバイスが見えます。



※ OpenJTAG は USB シリアルポートとして使えます。



4.2 ソフトウェアをインストールする

順番で以下のソフトウェアをインストールしてください

01.arm-2009q1-161-arm-none-eabi.exe(EABI を持ってコンパイラ)

02.openocd-0.1.0.msi(jtag デバッグ用ソフト)

03.yagarto-bu-2.19.1_gcc-4.3.3-c-c++_nl-1.17.0_gi-6.8.50_20090311.exe(gcc)

04.yagarto-tools-20070303-setup.exe(各種ユーティリティ)

05.Cygwin

06.jre-6u7-windows-i586-p.exe(Java)

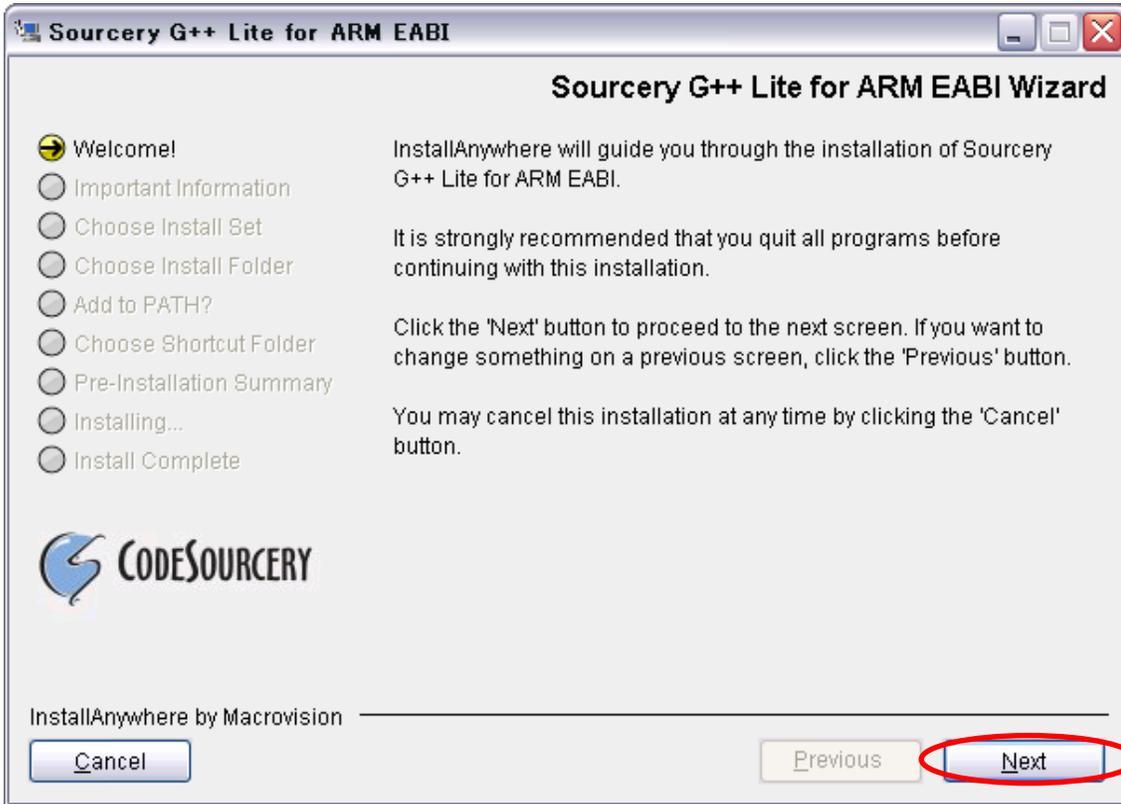
07.eclipse-cpp-galileo-win32.zip(Eclipse)

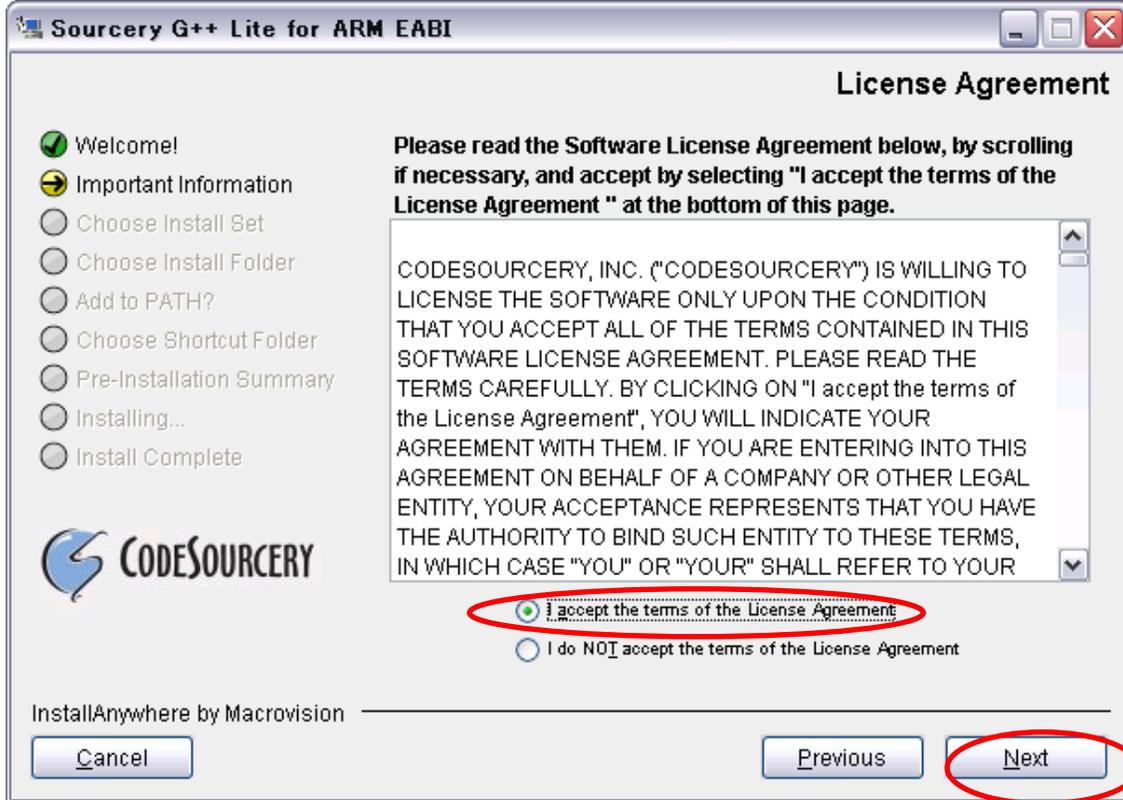
**メモ:①もしバージョン 1.4.2 以上の JRE を既にパソコンにインストールされたら、
06 番の JRE のインストールが不要です。**

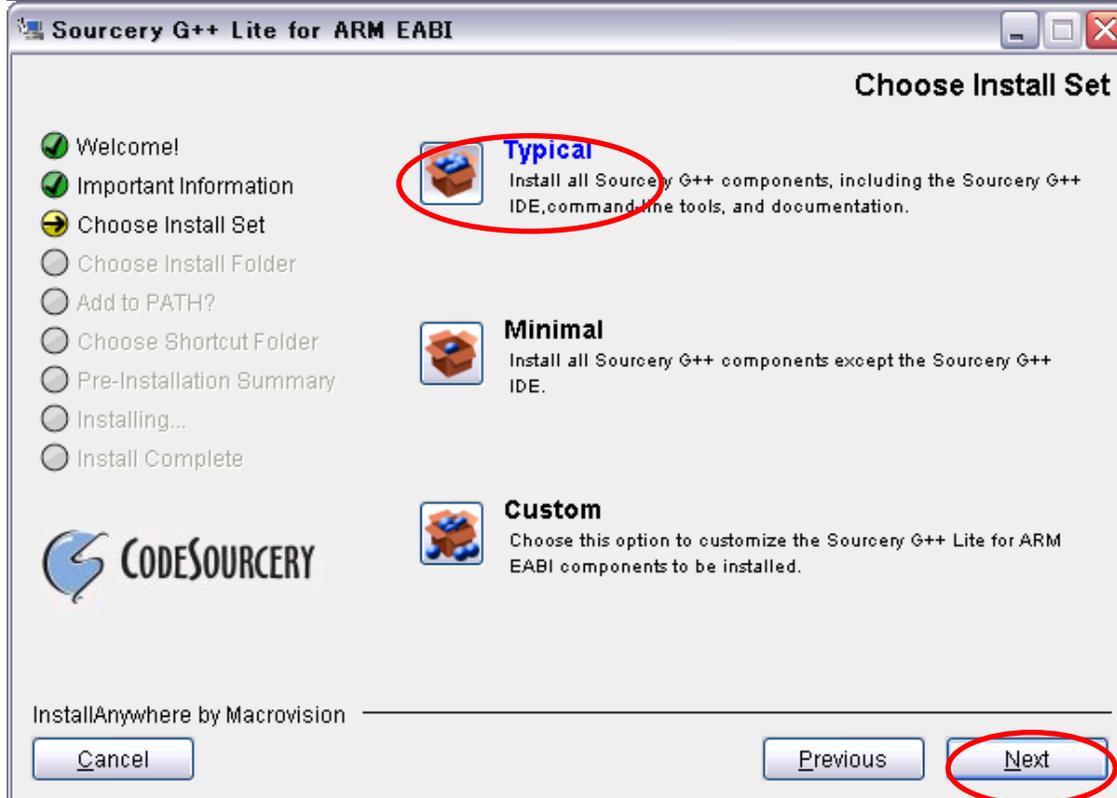
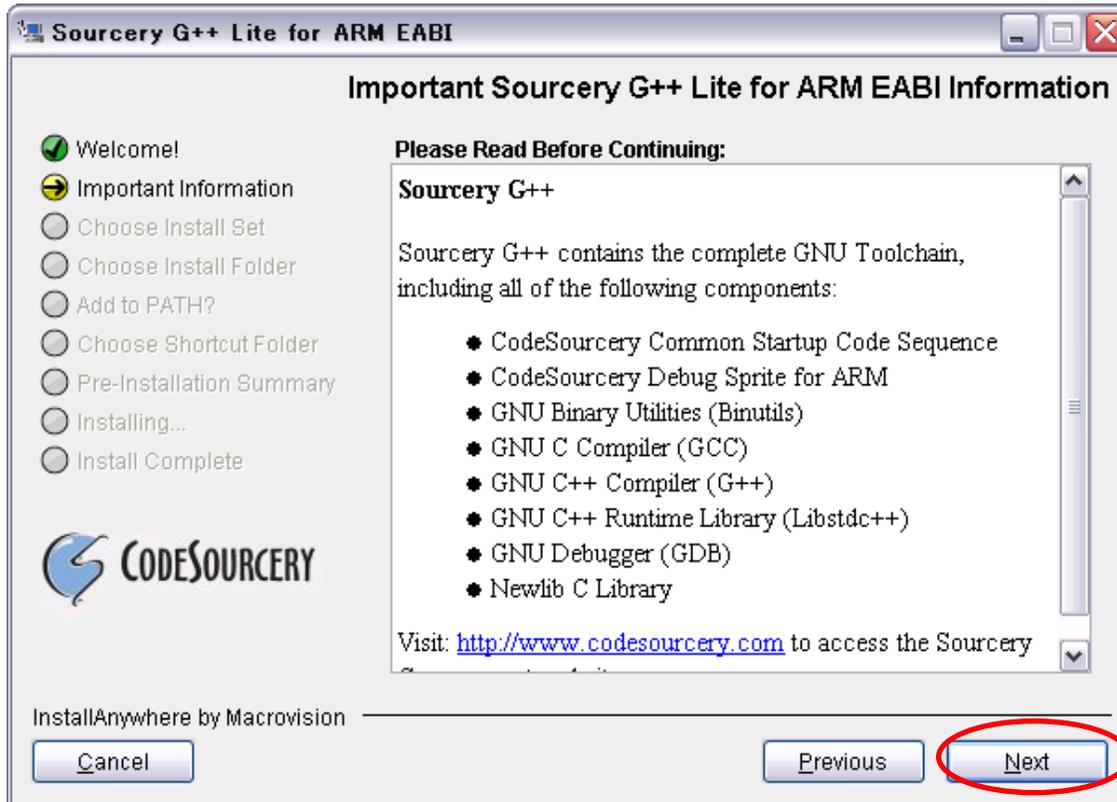
②05 番の eclipse を解凍して OK、インストール不要

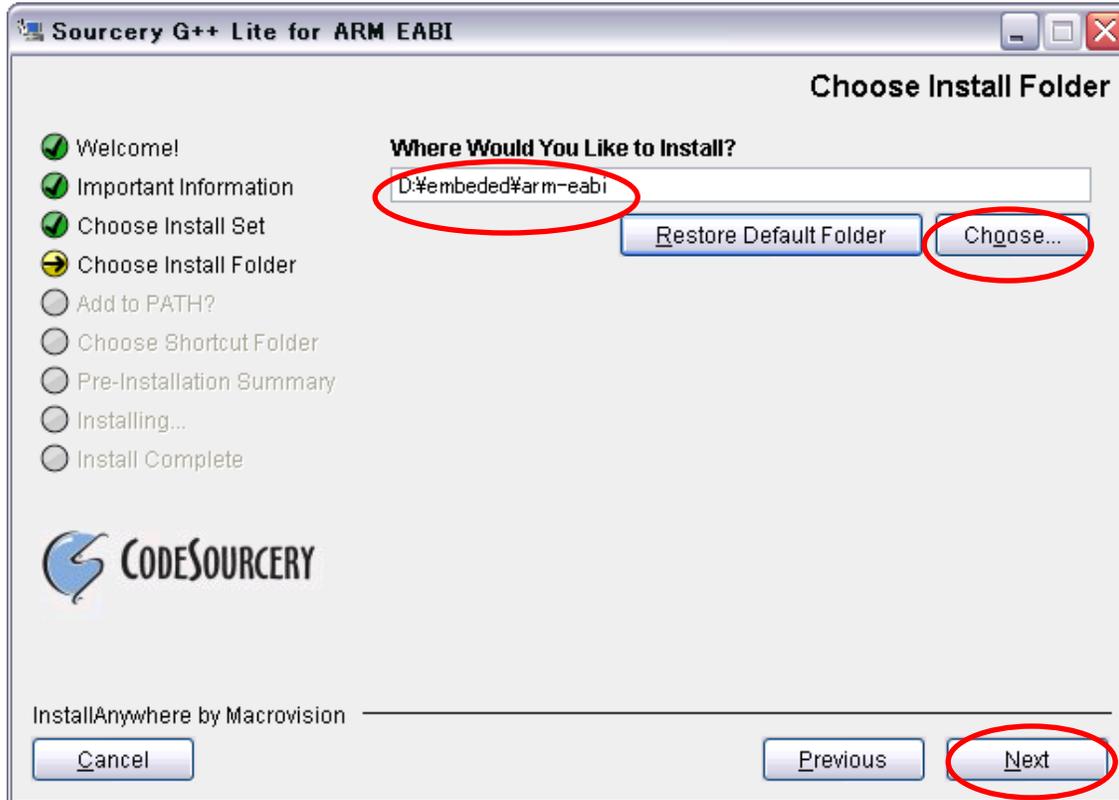
ダウンロードしたインストールファイルは「D:%tmp¥openJTAGIDE」に格納されます、本書に全てソフトウェアは「D:%embedded」にインストールされます。

4.2.1 「arm-none-eabi.exe」をインストール





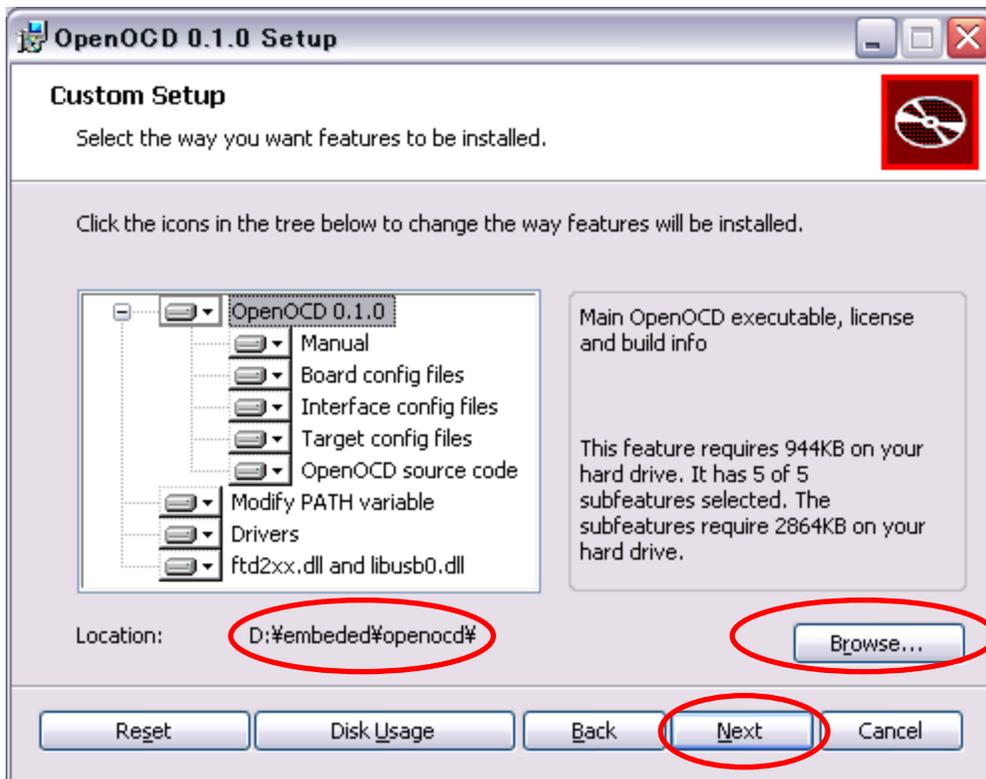
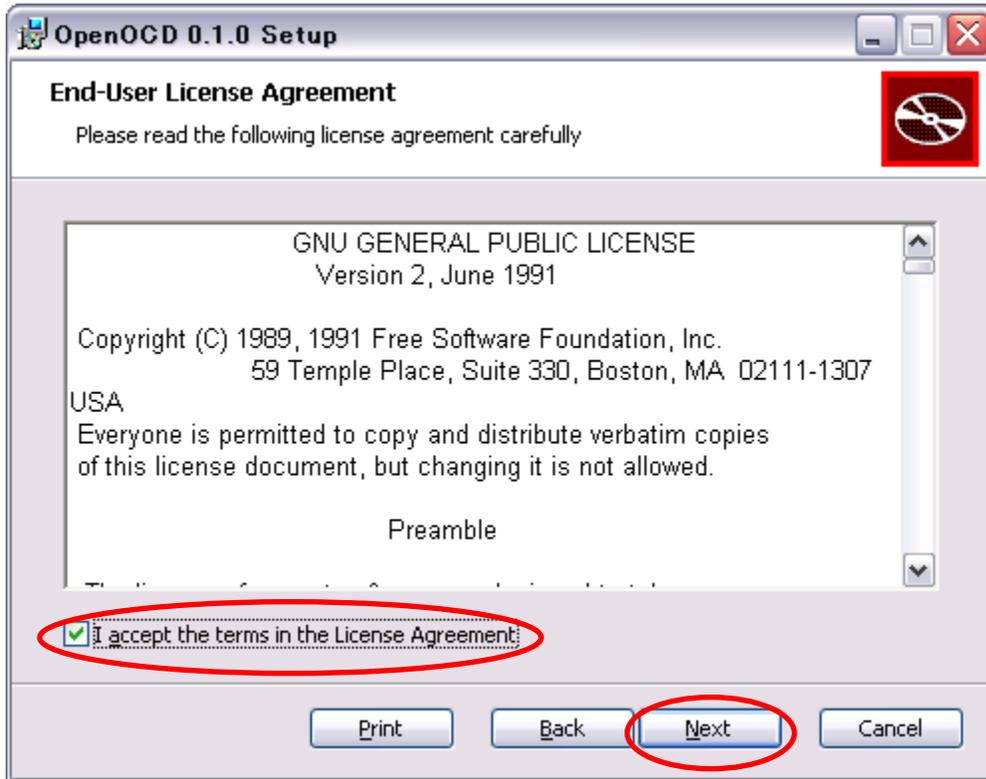


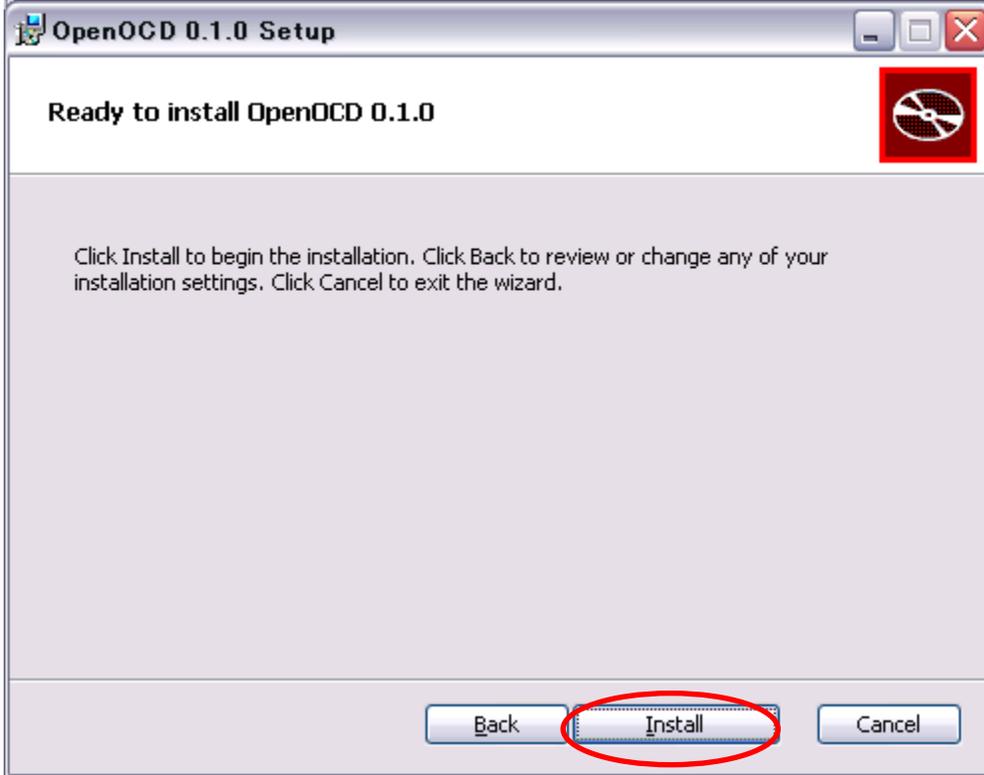


「Next」をクリックすると、インストールが始まります、インストール完了までに何分かかります、完了までにお待ちください。

4.2.2 「openocd」をインストール







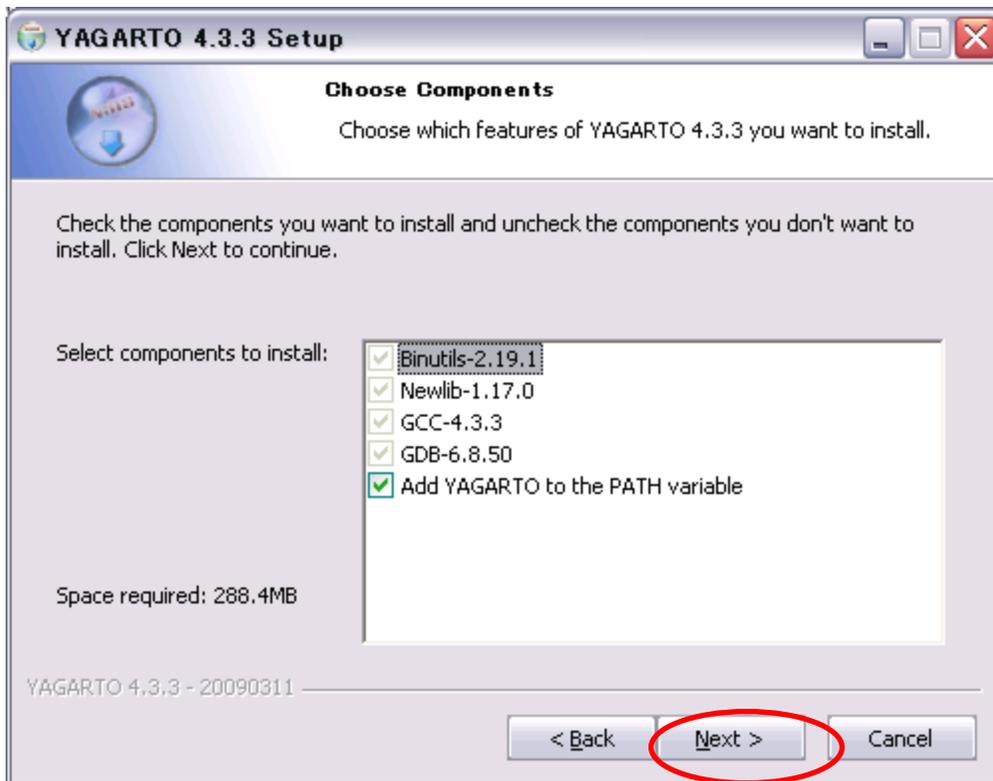
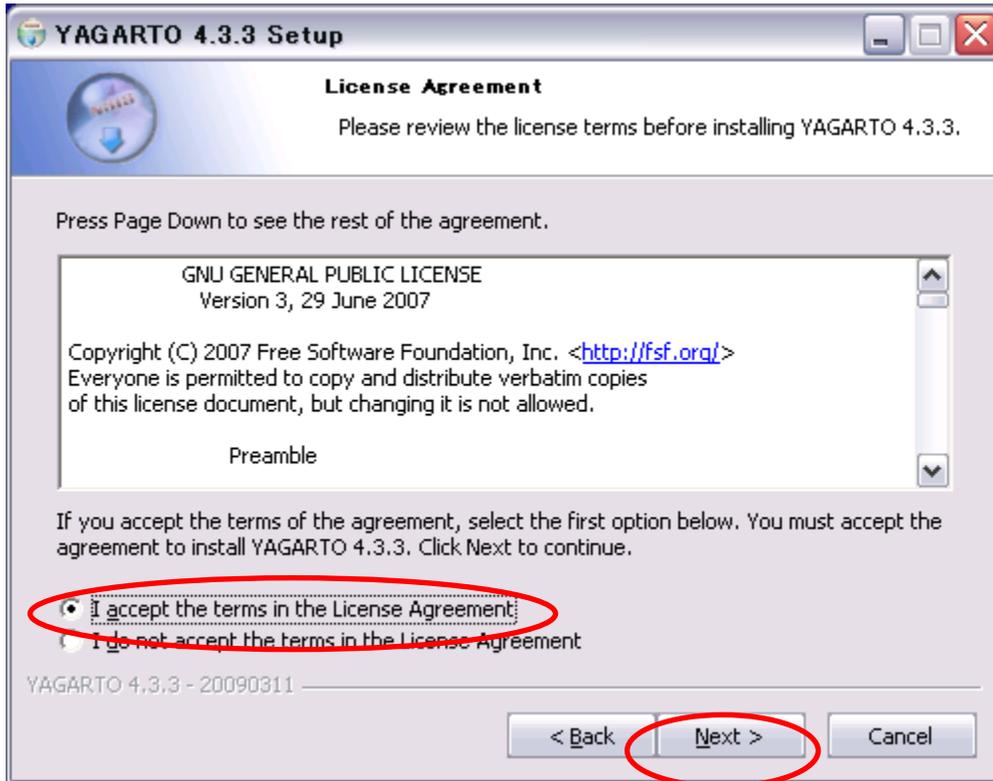
「Install」をクリックすると、インストールが始まります、インストール完了までに何分がかかります、完了までにお待ちください。

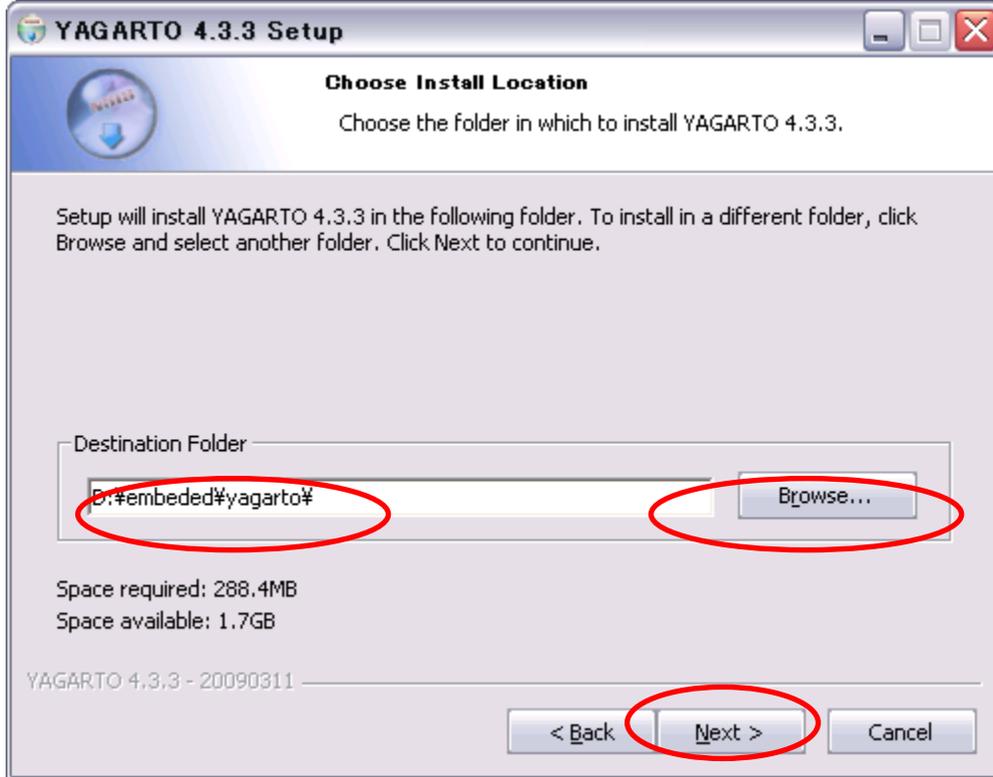
「Finish」をクリックし、OpenOCD のインストールが完了です。



4.2.3 「yagarto」をインストール



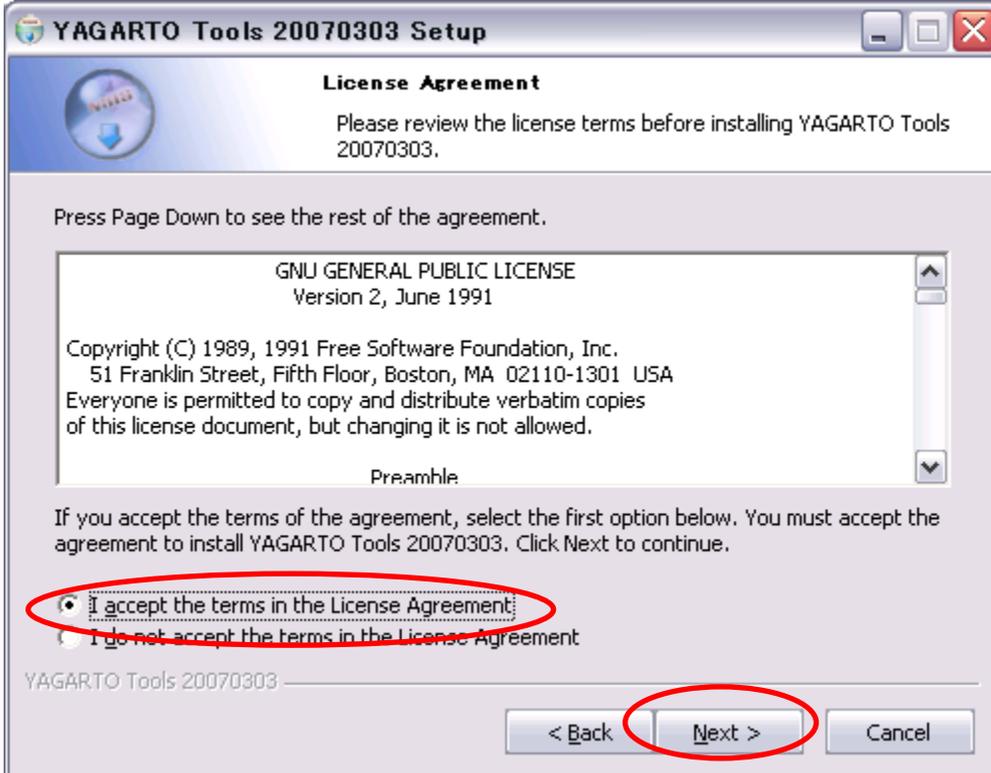


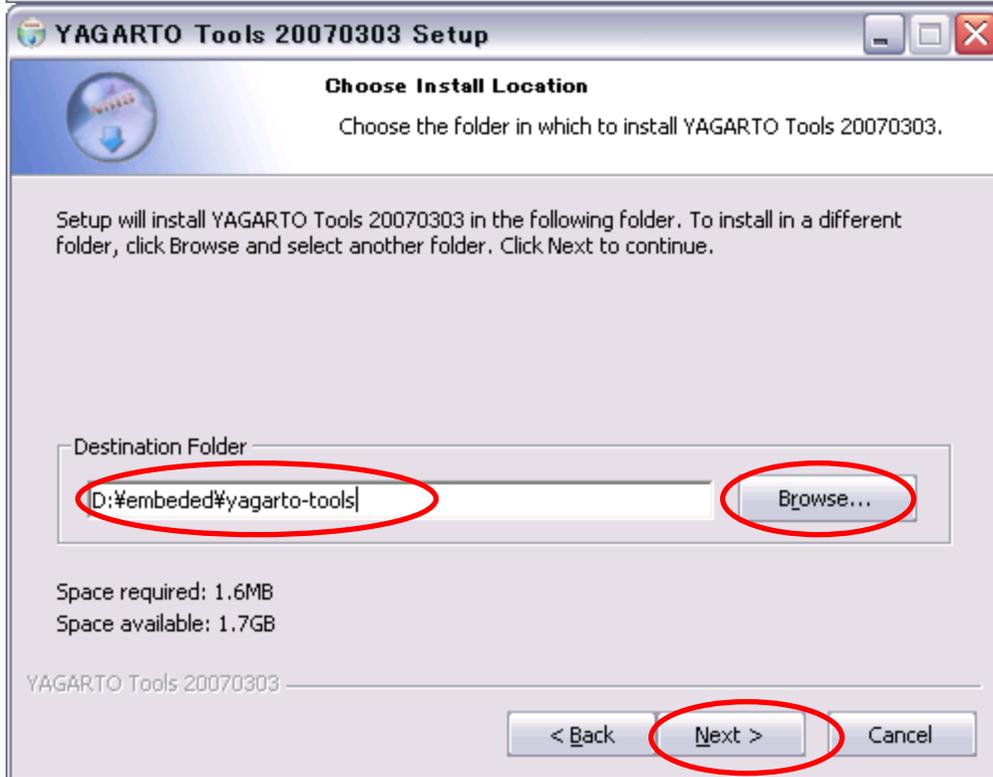


「Install」を押下、インストール完了まで何分がかかります。



4.2.4 「yagarto-tools」をインストール





「Install」を押下、インストール完了まで何分がかかります。



4.2.5 Cygwin のダウンロード&インストール

1 インストール

a) setup.exe の DL & 起動

<http://cygwin.com/>より **setpu.exe** をダウンロードし起動してください。

直接ダウンロードしたファイルを起動しても OK。

以下に注意すべき画面を中心に解説します(より詳しい手順は Cygwin のページを参照してください)。

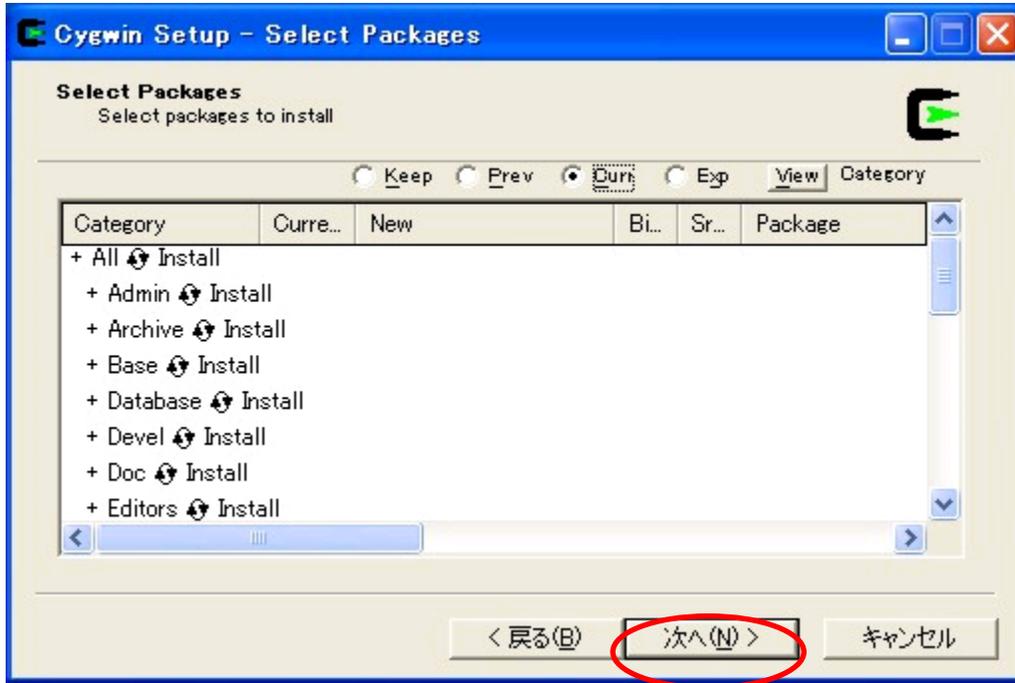
b) テキストファイル形式の選択

以下の「Default Text File Type」に「**Unix**」を選びます(テキストファイルが UNIX 形式(改行が LF)となります)。また Cygwin をインストールするマシンをサーバーとして利用したい場合には、「Install For」で「**All Users**」を選びます。



c) パッケージ選択

パッケージ選択の画面にて Category の「All」を「Install」に変更する。この変更によって全パッケージがインストールされることになる



上記の設定が終われば「次へ」をクリックすれば、ダウンロード&インストールが始まる。



2 インストール後の設定

a) ホームディレクトリの作成

通常、デスクトップにできた Cygwin アイコンをクリックすると、/usr/username がホームディレクトリになるが、Windows の環境変数 HOME が設定されていれば、Cygwin においてもそこがホームディレクトリとなる。この問題を避けるために、コントロールパネルの「システム」を起動し、「詳細設定」タブの下にある「環境変数」をクリックする。そしてユーザ環境変数に HOME 変数が設定されていれば削除する。

ここでデスクトップの Cygwin アイコンをクリックすればホームディレクトリが自動的に作成される。

次にホームディレクトリの下に各種設定ファイルを編集していく。利用する漢字コードによって設定方法が異なるので、以下では SJIS 環境を説明する。設定するファイルは、.bash_profile、.bashrc、.inputrc、.vimrc の4つで(全てのファイルはピリオドから始まるファイルで隠しファイルとなっており、“ls -a”にて確認できる)、以下に示した通りに設定する。既に存在するファイルについては追加し、存在しないファイルについては新規作成する。またファイルの編集は [TeraPad](#) などのエディタを利用すれば便利。注意点として、新規ファイルを保存する際には、「ファイル」→「漢字・改行コードを指定保存」を選び、改行コードに「LF」を選んで保存してください。

b) SJIS 環境に関わる4つファイルの設定

i).bash_profile

```
export TERM=vt100
export LANG=ja_JP.SJIS
export LESSCHARSET=japanese-sjis
```

ii).bashrc

```
alias ls='ls --show-control-chars --color=auto'
alias ct='cygterm &'
function ie {
echo /cygdrive/c/Program Files/Internet Explorer/IEXPLORE "$(cygpath -w $PWD)¥¥$1"
/cygdrive/c/Program Files/Internet Explorer/IEXPLORE "$(cygpath -w $PWD)¥¥$1" &
}
```

上記の ie 関数を設定しておくことにより、XML や XMLtable ファイルをコマンド上から起動可能になる。
例) ie dat.xml #dat.xml を Internet Explorer を利用して表示する。

iii).inputrc

```
set convert-meta off
set input-meta on
set output-meta on
```

iv).vimrc

```
set encoding=japan
set fileencodings=sjis
```



4.2.6 「jre」をインストール

一般のソフトウェアと同じようにインストールします。もしバージョン 1.4.2 以上の JRE を既にパソコンにインストールされたら、インストールが不要です。

「06.jre-6u7-windows-i586-p.exe」あるいはダウンロードした JRE インストールファイルをクリックし、「D:¥Embedded¥JRE」にインストールする

4.2.7 「Eclipse」をインストール

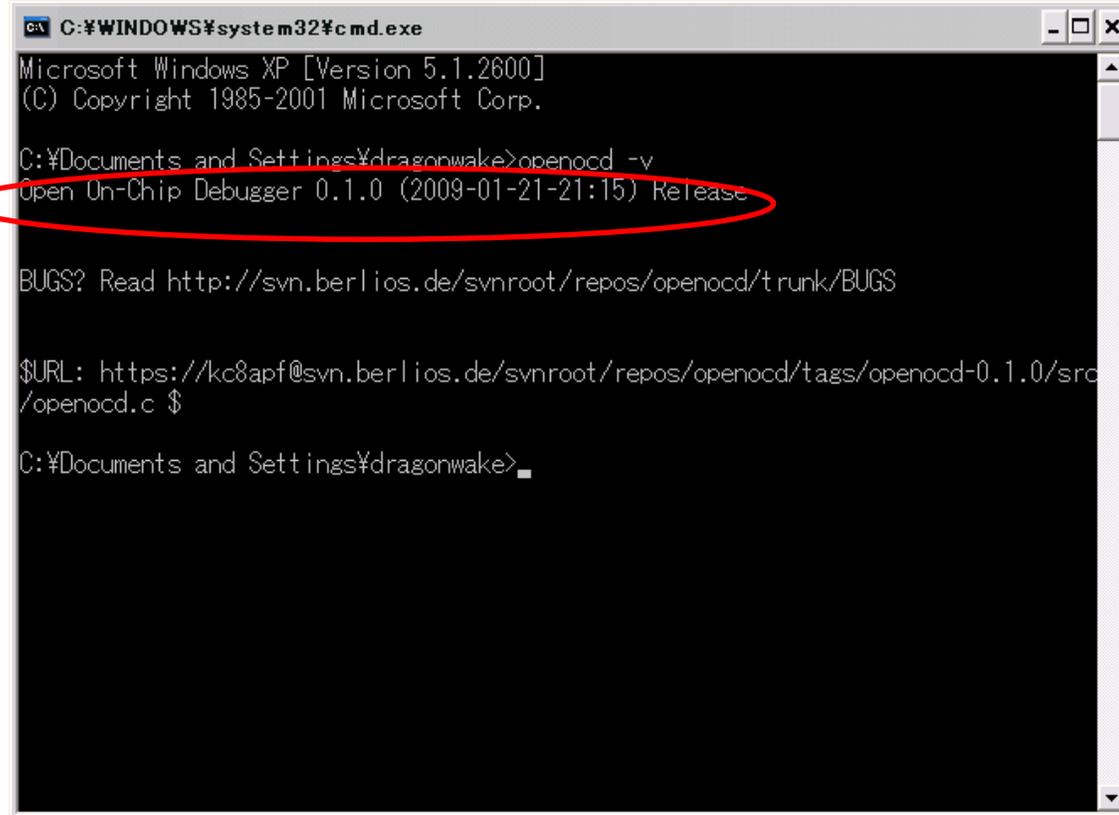
「07.eclipse-cpp-galileo-win32.zip」あるいはダウンロードしたファイルを「D:¥Embedded」解凍

4.3 ソフトウェアインストール後の動作確認

確認方法:「スタート」→「ファイル名を指定して実行」→「cmd」を入力

4.3.1 OpenOCD の確認

確認コマンド:openocd -v



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dragonwake>openocd -v
Open On-Chip Debugger 0.1.0 (2009-01-21-21:15) Release

BUGS? Read http://svn.berlios.de/svnroot/repos/openocd/trunk/BUGS

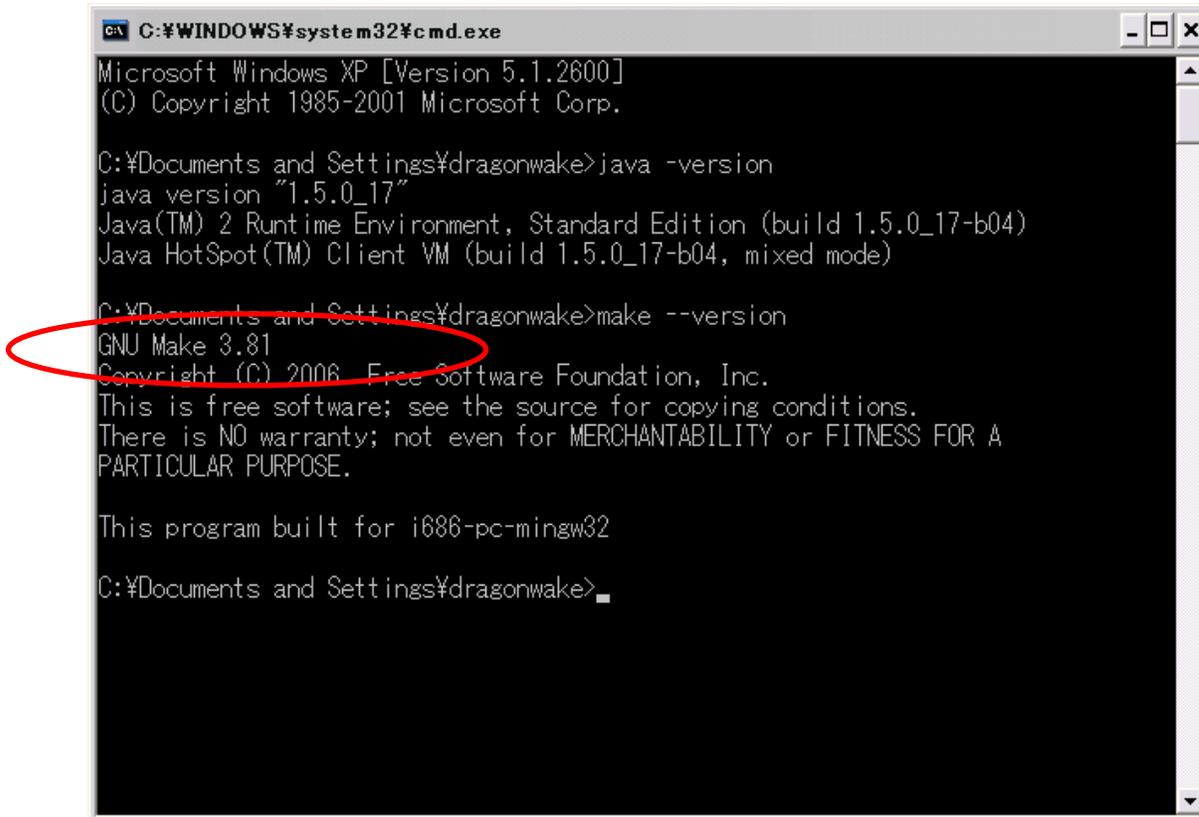
$URL: https://kc8apf@svn.berlios.de/svnroot/repos/openocd/tags/openocd-0.1.0/src
/openocd.c $

C:\Documents and Settings\dragonwake>
```

4.3.2 コンパイラ確認

1:GCC コンパイラ「make」確認

確認コマンド:make --version



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dragonwake>java -version
java version "1.5.0_17"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_17-b04)
Java HotSpot(TM) Client VM (build 1.5.0_17-b04, mixed mode)

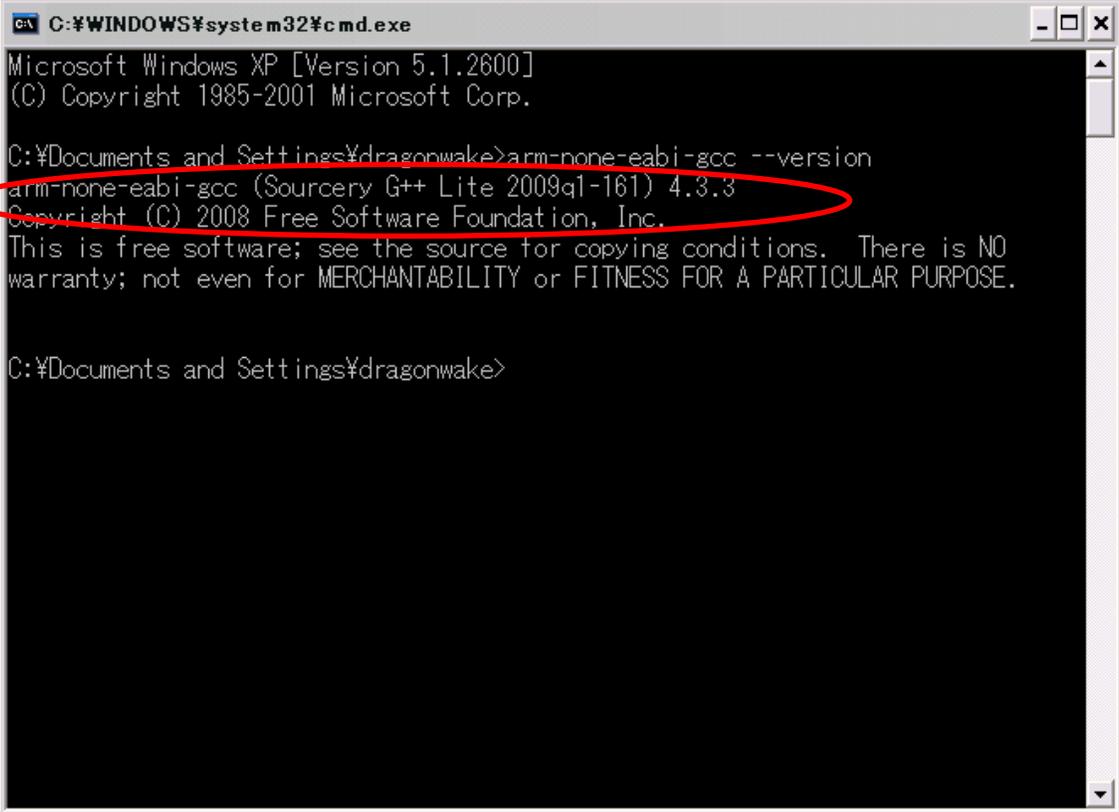
C:\Documents and Settings\dragonwake>make --version
GNU Make 3.81
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

This program built for i686-pc-mingw32

C:\Documents and Settings\dragonwake>
```

2: ARM EABI「arm-none-eabi-gcc」確認

確認コマンド: arm-none-eabi-gcc --version



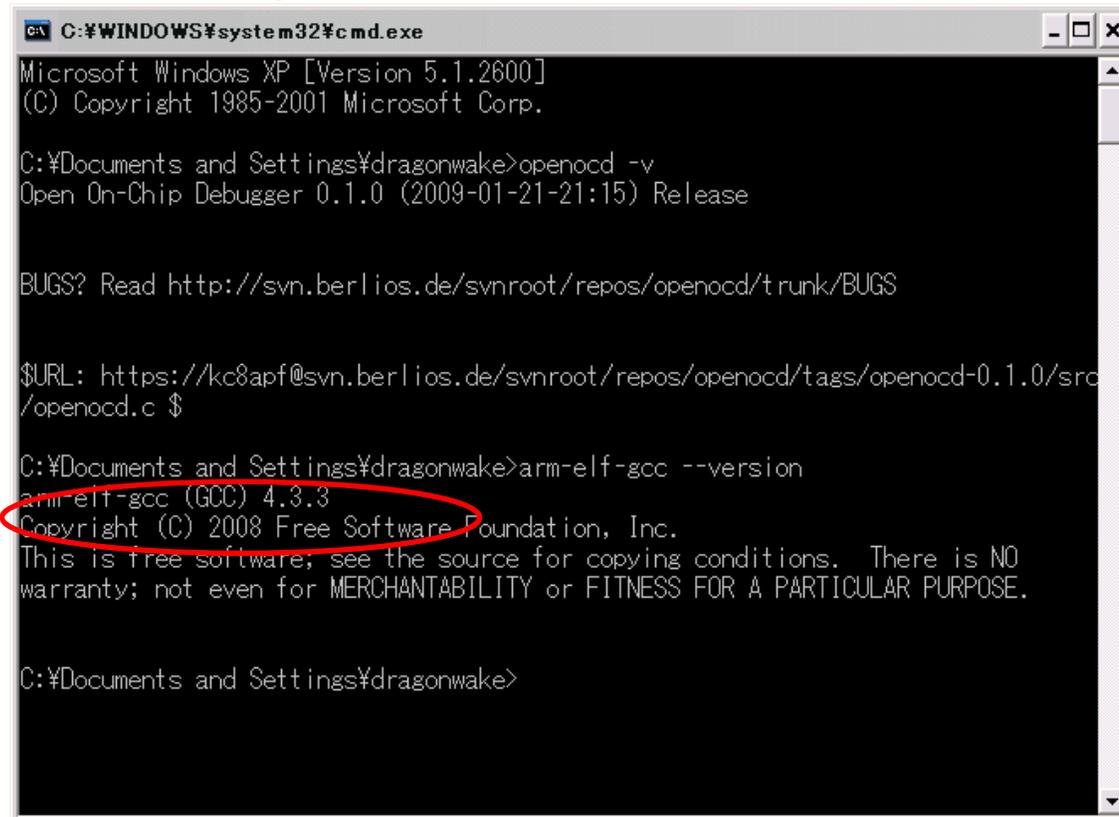
```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dragonwake>arm-none-eabi-gcc --version
arm-none-eabi-gcc (Sourcery G++ Lite 2009q1-161) 4.3.3
Copyright (C) 2008 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C:\Documents and Settings\dragonwake>
```

3: ARM コンパイラ「arm-elf-gcc」確認

確認コマンド: arm-elf-gcc --version



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dragonwake>openocd -v
Open On-Chip Debugger 0.1.0 (2009-01-21-21:15) Release

BUGS? Read http://svn.berlios.de/svnroot/repos/openocd/trunk/BUGS

$URL: https://kc8apf@svn.berlios.de/svnroot/repos/openocd/tags/openocd-0.1.0/src
/openocd.c $

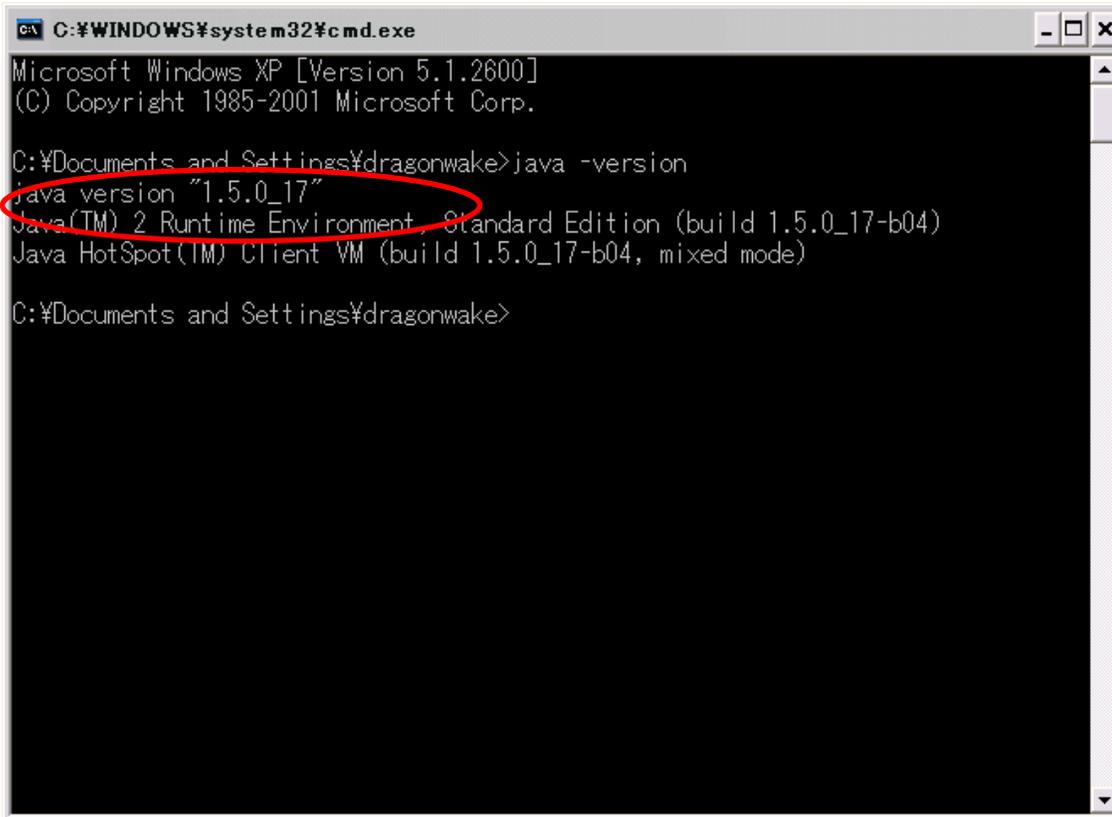
C:\Documents and Settings\dragonwake>arm-elf-gcc --version
arm-elf-gcc (GCC) 4.3.3
Copyright (C) 2008 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C:\Documents and Settings\dragonwake>
```

4.JRE バージョン確認:

確認コマンド:

java -version



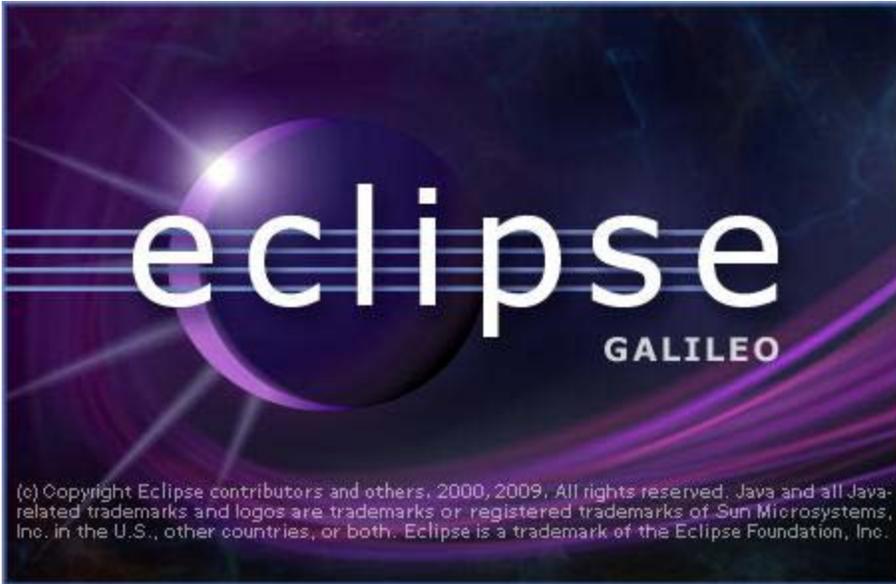
```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dragonwake>java -version
java version "1.5.0_17"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_17-b04)
Java HotSpot(TM) Client VM (build 1.5.0_17-b04, mixed mode)

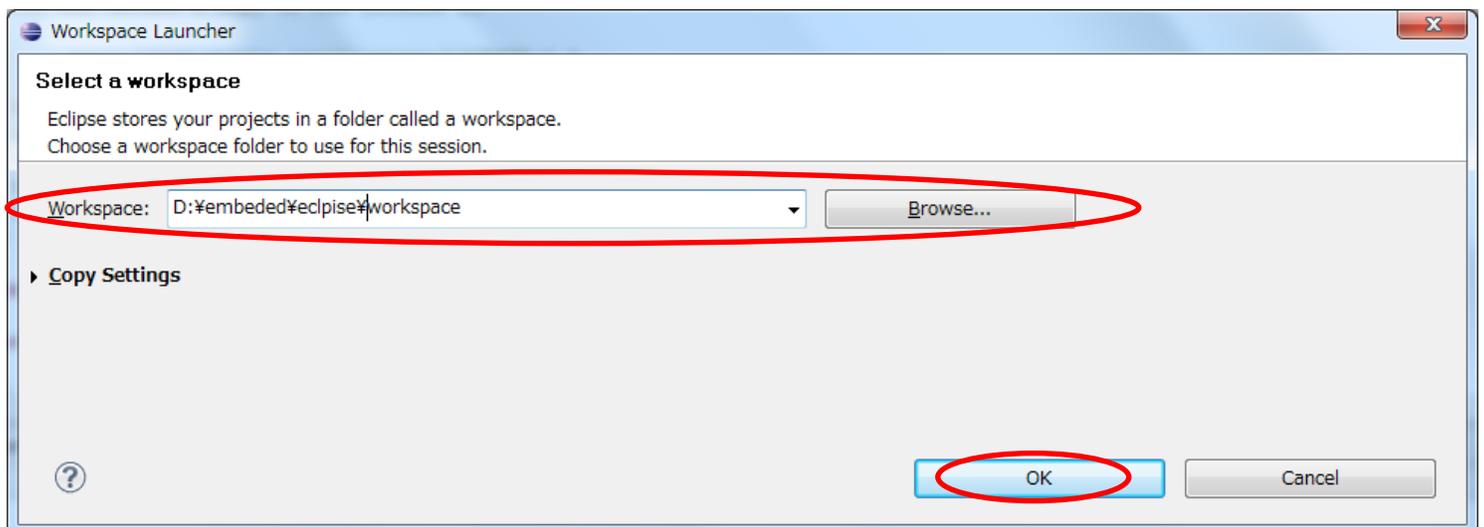
C:\Documents and Settings\dragonwake>
```

第五章 Eclipse の設定

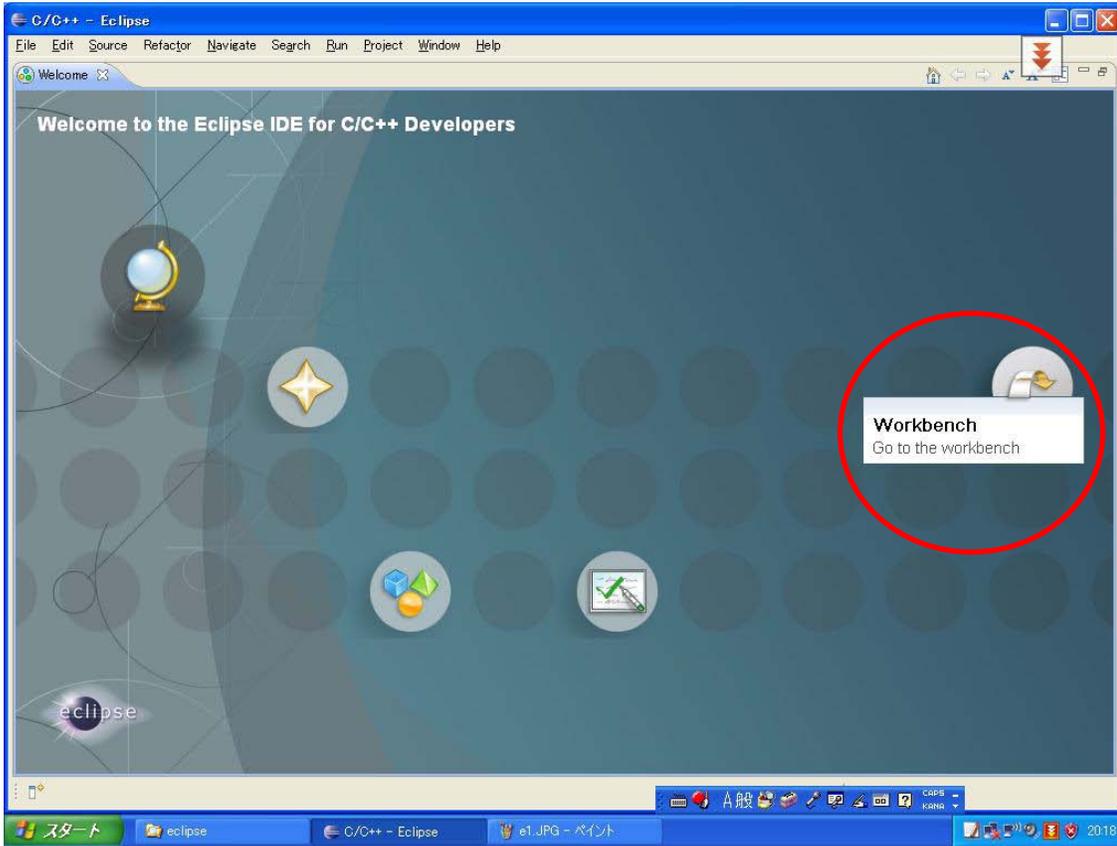
5.1 Eclipse を起動する



最初に Workspace の場所を聞いてきます。デフォルトは名前が長いので、“D:¥embedded¥eclipse¥workspace”に変更しました。



画面の Workbench をクリックします。



注意: サンプルを簡単に導入できるように、ダウンロードして前章で解凍してできた「D:\OpenJTAG\gcc_openjtag_eclipse_arm_example」の中の全てのフォルダーを「D:\embedded\eclipse\workspace」へコピーしてください。



5.2 Eclipse プラグイン(Zylin Embedded CDT)インストール

- * Zylin Embedded CDT の 4.81 バージョンは Zylin のホームページから取り下げますので、弊社から圧縮ファイルをダウンロードしてから Eclipse にコピーしてください。
- * Zylin Embedded CDT の最新版では検証していないため、インストールしないでください。

Zylin Embedded CDT ダウンロード URL:

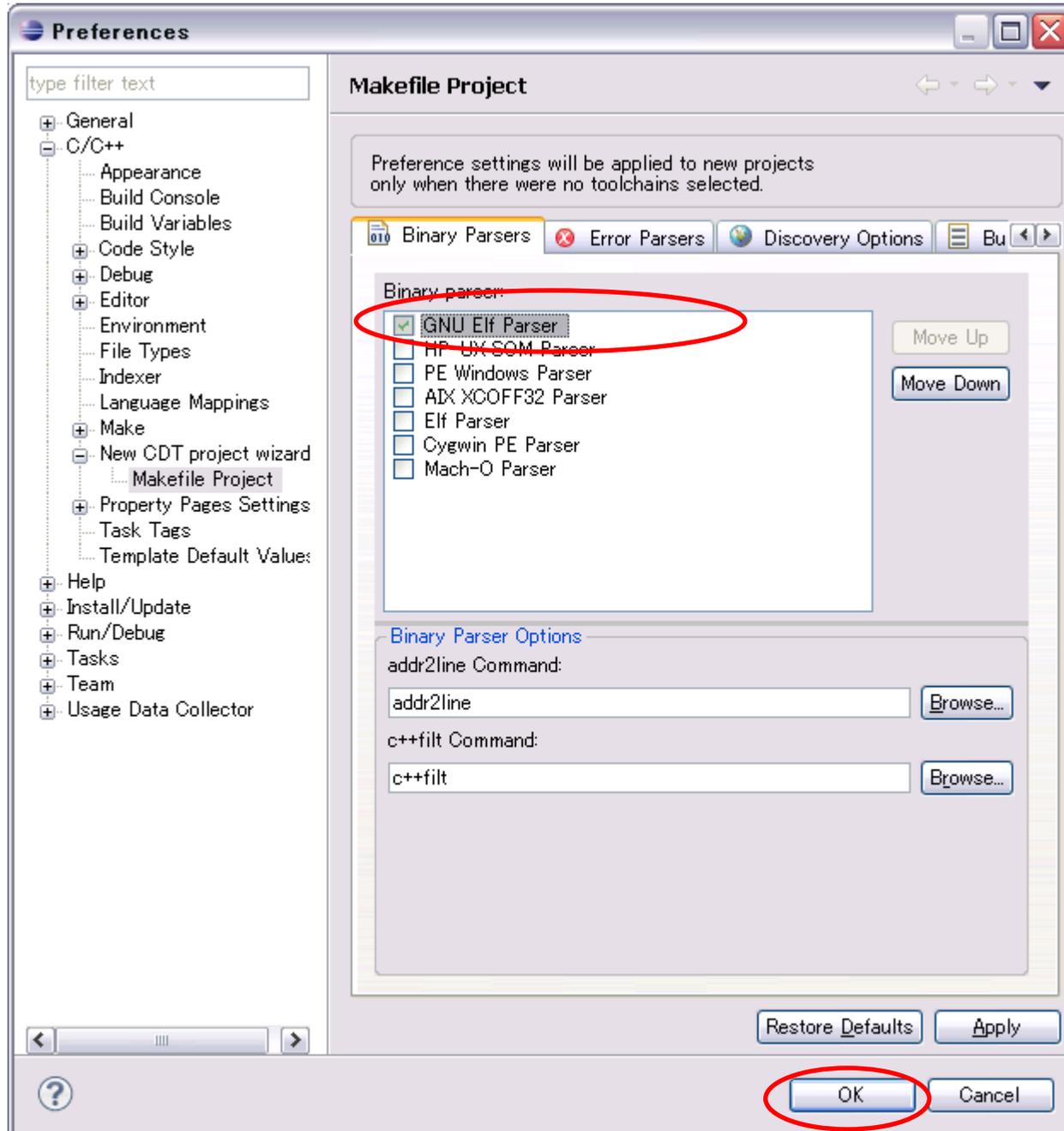
http://www.dragonwake.com/download/open-jtag/com.zylin.embeddedcdt_4.8.1.zip

上記ファイルを解凍し「Eclipse インストール先」の「plugins」というフォルダーにコピーしてください。コピーが完了したら、Eclipse を再起動させます。

5.3 ビルドの設定

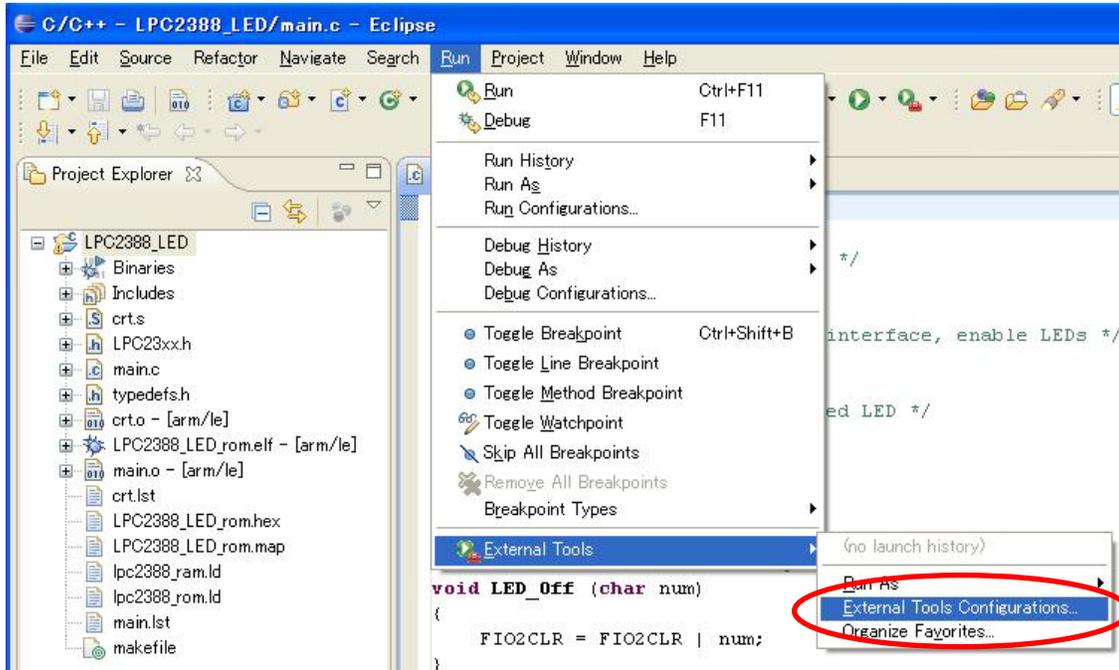
Eclipse の“Window”→“Preferences”を選択し、

Preferences の“C/C++ Build”→“Settings”を選択し“Binary Parsers”タブの“GNU Elf Parser”にチェックを入れて OK ボタンを押します

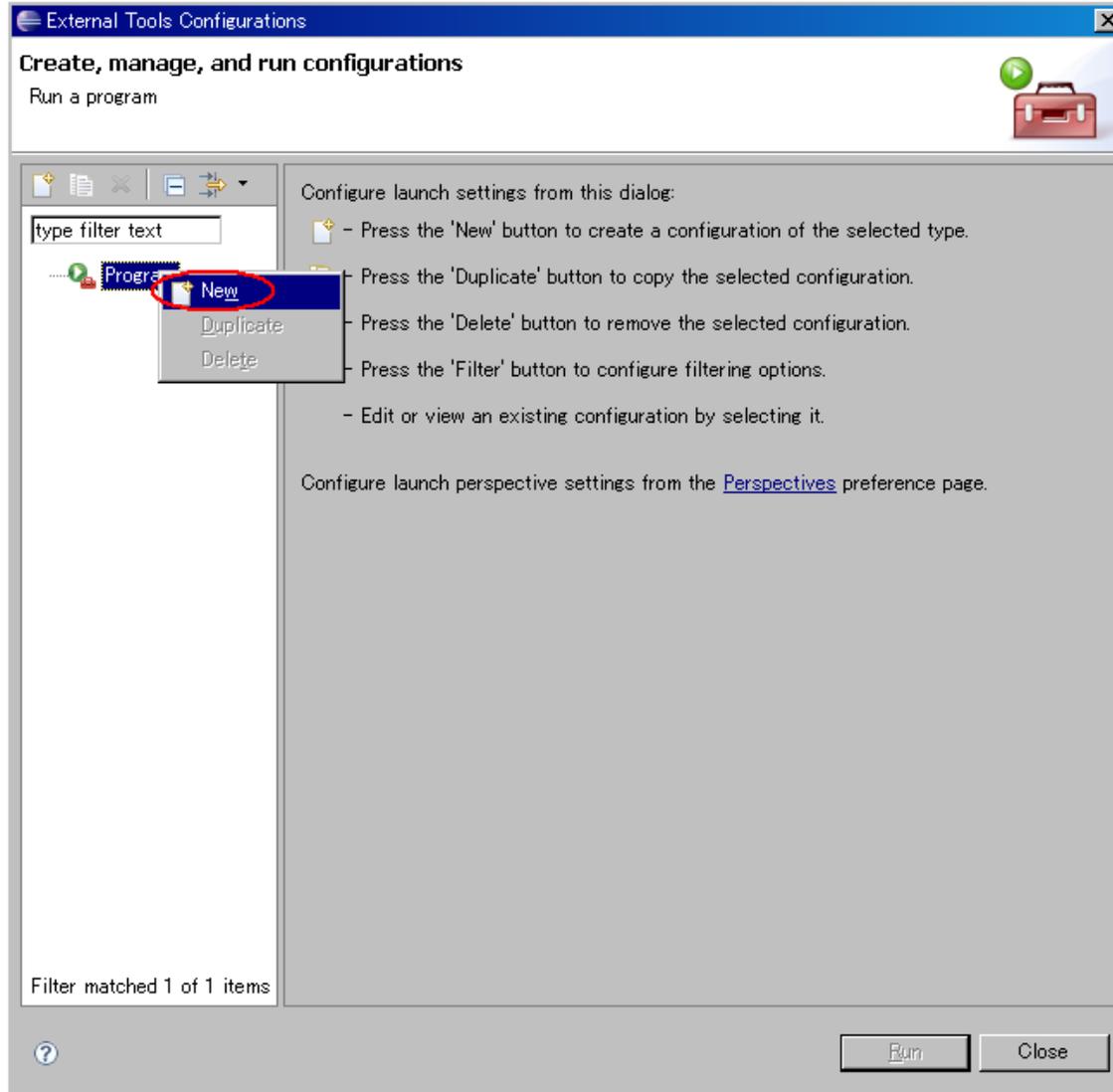


5.4 OpenOCD の設定

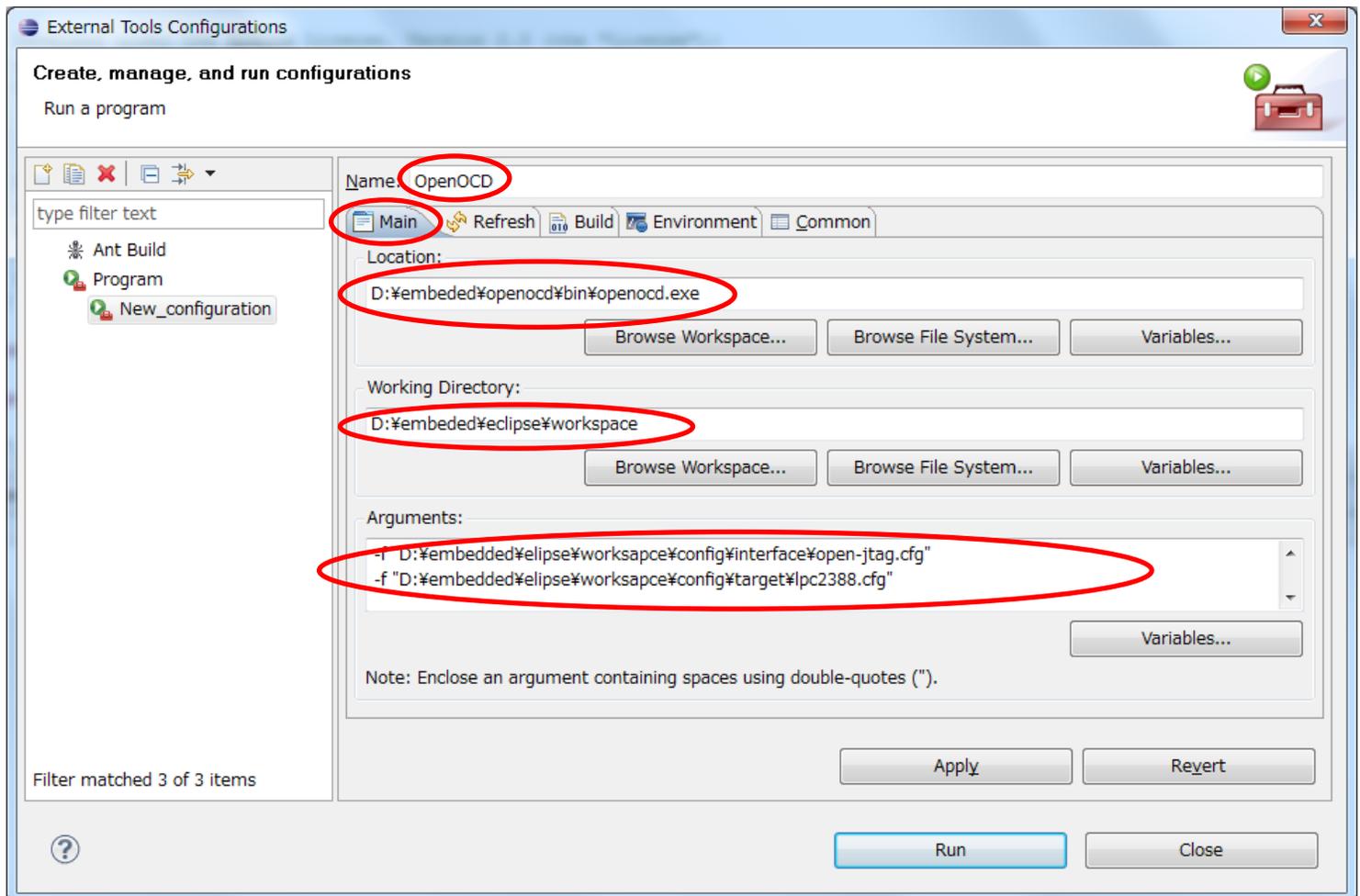
1. Eclipse の“Run”→“External Tools.”→“External Tools Configurations...”を選択します。

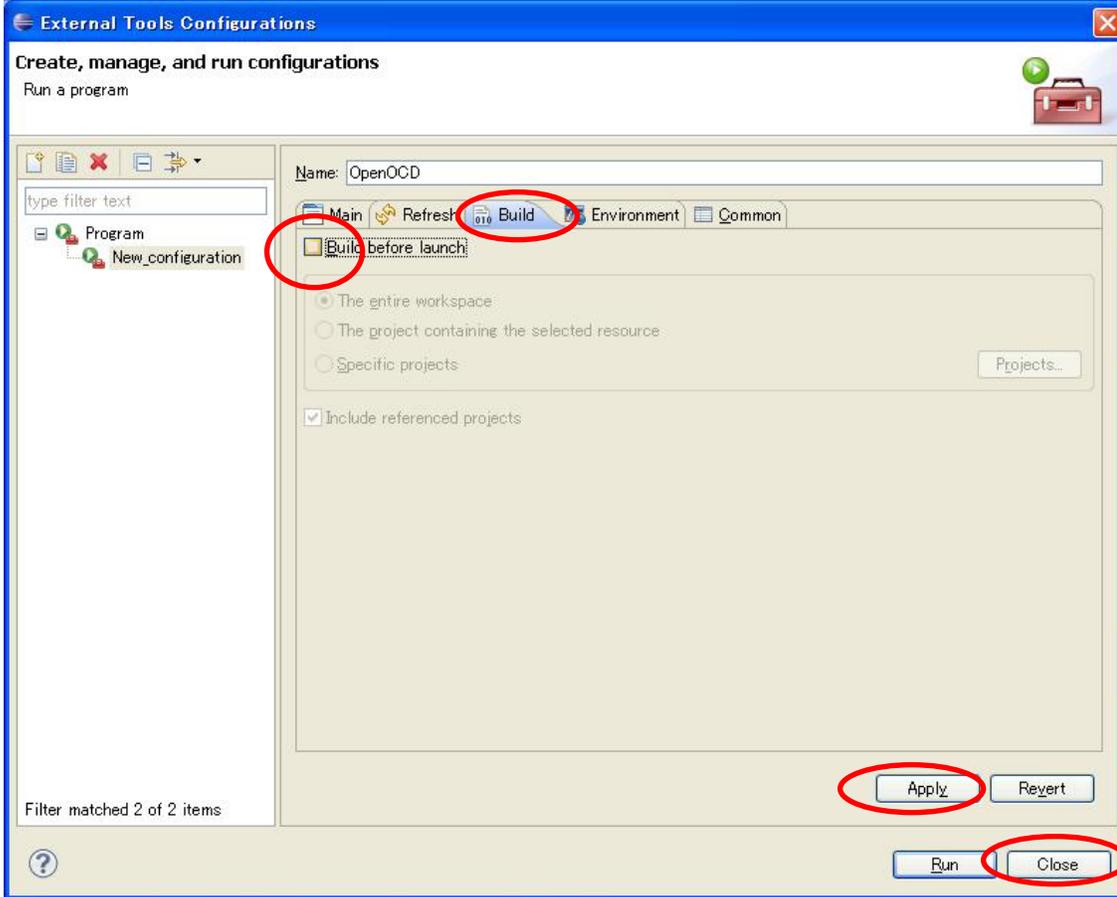


2. External Tools Configurations の“Program”を右クリックし、“New”を選択します。



3. Main タブの”Name”に適当な名前を入力してください、例えば、”OpenOCD”。
”Location:”に”D:¥embedded¥openocd¥bin¥openocd.exe”、
”Working Directory:”に” D:¥embedded¥eclipse¥workspace ”
”Arguments:”に-f ”D:¥embedded¥eclipse¥worksapce¥config¥interface¥open-jtag.cfg”
-f ”D:¥embedded¥eclipse¥worksapce¥config¥target¥lpc2388.cfg”と入力します。
* 基板種類 (lpc2148,lpc2388,mini2440 など)により、Arguments の設定も異なる。



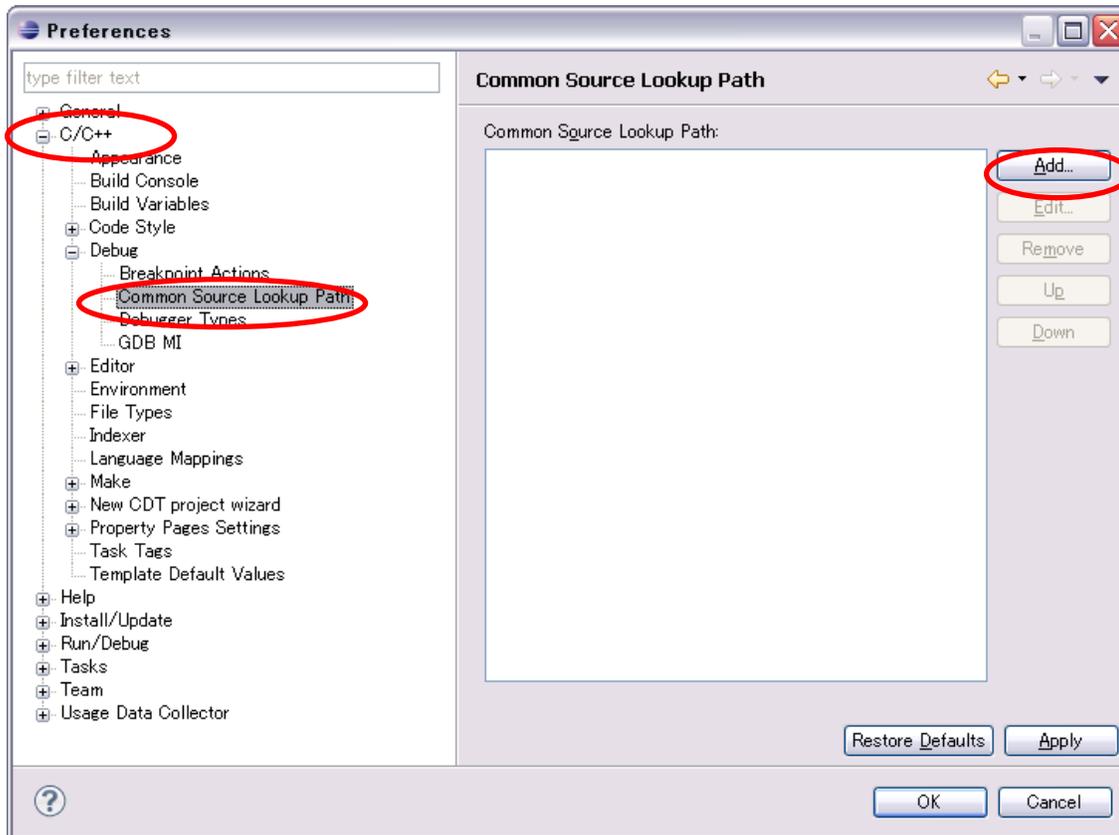


Build タブをクリックし“Build before launch”にチェックを外れます。全てを入力し終えたら“Apply”ボタンを押し、“Close”ボタンを押します。

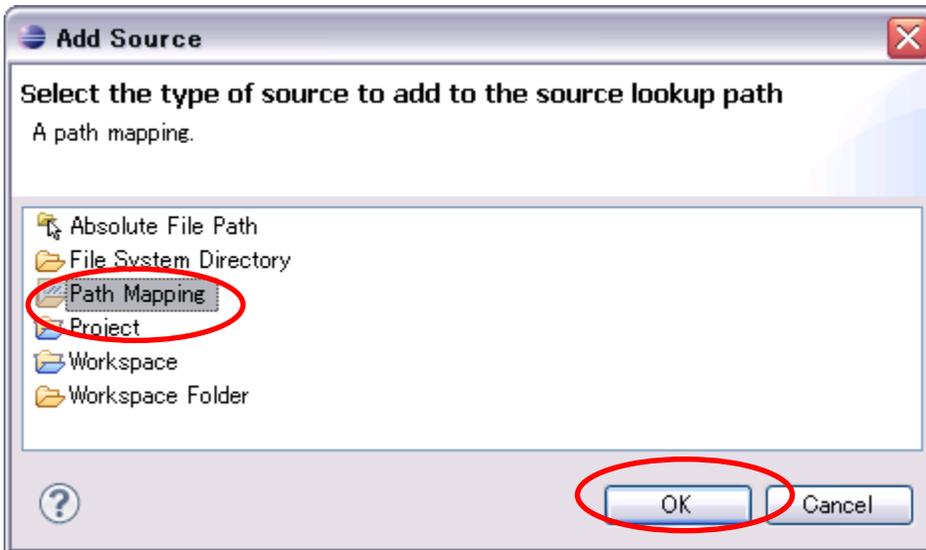
5.5 Cygwin のソース・ルックアップ・パスの設定

GDB でデバッグを開始すると「Can't find a source file at "/cygdrive/c/project/hoge/hoge.cpp"」なんて怒られる。Cygwin のパスを eclipse が認識できていない様子。この場合は、ルックアップ・パスの設定が必要。

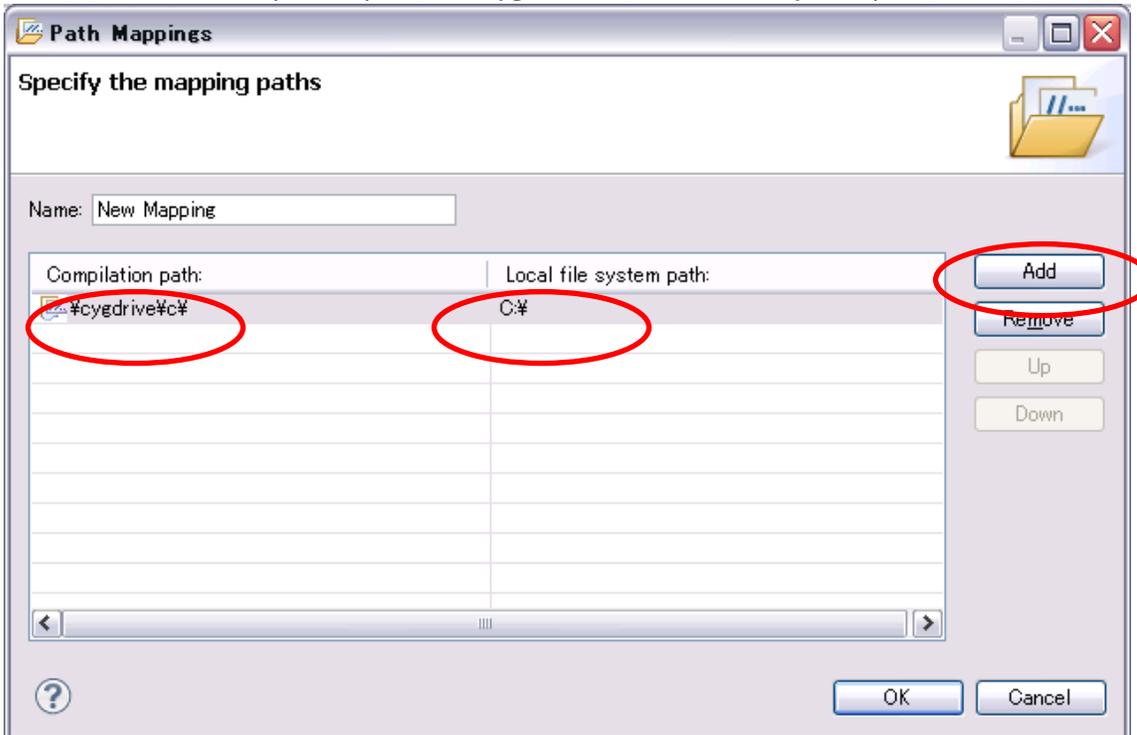
eclipse のメニュー[ウィンドウ]→[設定]で設定ダイアログを開き、[C/C++]→[Debug]→[Common Source Lookup path]にルックアップ設定を追加する。



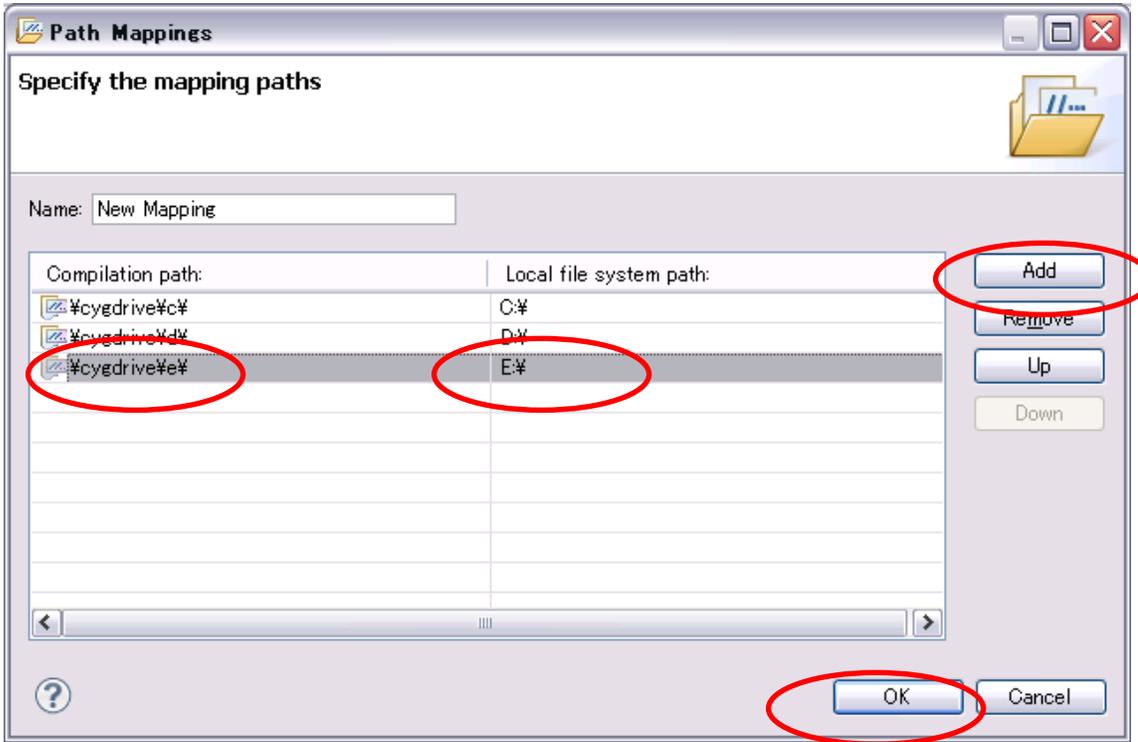
「Path Mapping」を選択、「OK」を押下



「Add」をクリック、「Compilation path」に「/cygdrive/c/」、「Local file system path」に「c:/」を指定する。

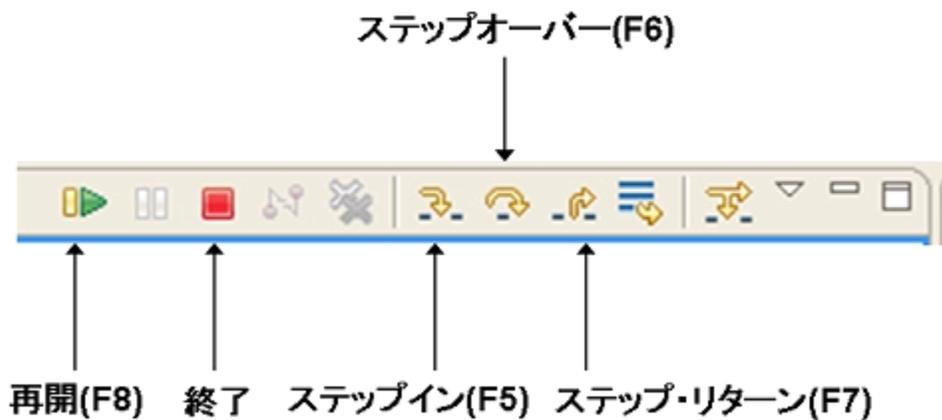


「Add」をクリック、他のローカルドライブを同じように追加する。最後、「OK」ボタンを押下



5.6 Eclipse デバッグ用のコマンド及びショートカットキー一覧

詳しくは Eclipse に関わるドキュメントを参照



ステップ実行において良く使われる操作の一覧を以下に示します。

操作名	ショートカットキー
再開	F8
ステップイン	F5
ステップオーバー	F6
ステップ・リターン	F7

ステップ実行とは関係ありませんが、前回起動したクラスを再度実行したりデバッグする場合は、以下のショートカットキーが便利です。

操作名	ショートカットキー
前回の起動を実行	Ctrl + F11
前回の起動をデバッグ	F11

第六章 ARM シリーズデバッグ手順

* 本書に使われる ARM シリーズは全て弊社の製品となります。

詳しくは弊社の通販サイト: <http://www.csun.co.jp> までにご参照ください。

第四章「インストール手順」、第五章「Eclipse の設定」に従って行っていれば、ARM シリーズ、例えば、ARM7 (LPC2148、LPC2388)、Cortex-M3 (STM32F103)、ARM9 (MINI2440) のサンプルを導入のうえ、デバッグ手順を説明させていただきます。

6.1 ARM7 の LPC2148

6.1.1 LPC2148 ボード購入 URL

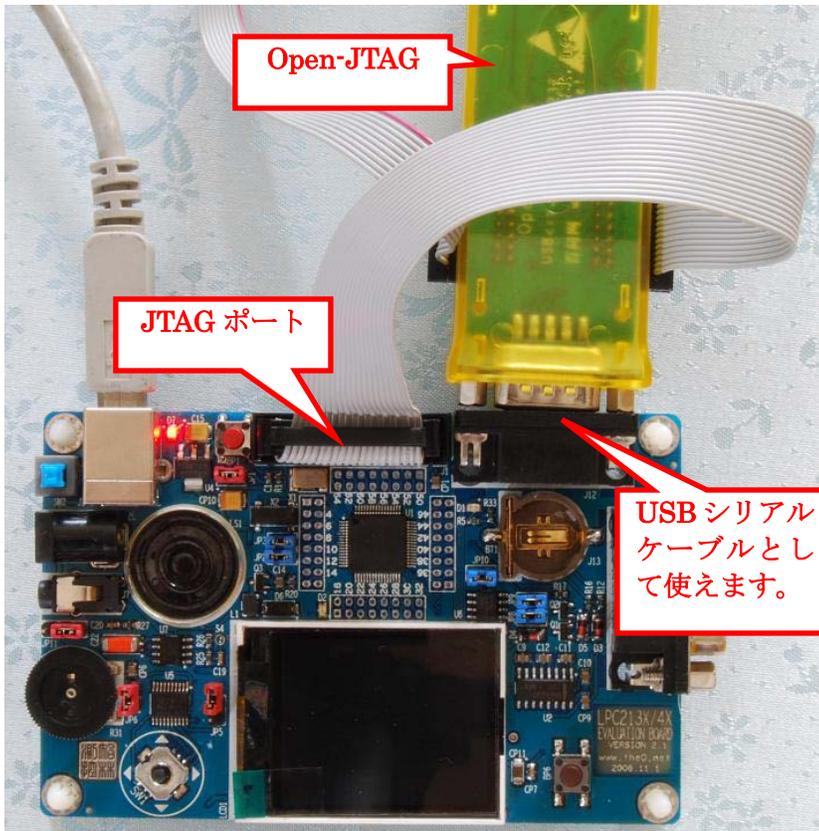
<http://www.csun.co.jp/SHOP/2009071209.html>

ボード URL から LPC2148 マニュアルを参照し、Open-JTAG に接続してください。

LPC2148 マニュアルダウンロード URL :

http://www.dragonwake.com/download/LPC2148/LPC2148_manual.pdf

6.1.2 ハードウェア動作確認



1. OpenJTAG をパソコンの USB ポートに挿入する
2. JTAG ケーブルで OpenJTAG と LPC2148 ボードを繋ぐ
3. LPC2148 ボードに電源を入れる
4. 下記のコマンドを入力します。

```
openocd -f "D:\embedded\eclipse\workspace\config\interface\open-jtag.cfg" -f
```

```
"D:\embedded\eclipse\workspace\config\target\lpc2148.cfg"
```

はじめの-f は OpenJTAG のコンフィグファイルを使います。二番目の-f は lpc2148 のコンフィグファイルを使います。

6.1.3 LPC2148 用のサンプル「LED」をデバッグ

サンプルダウンロード URL :

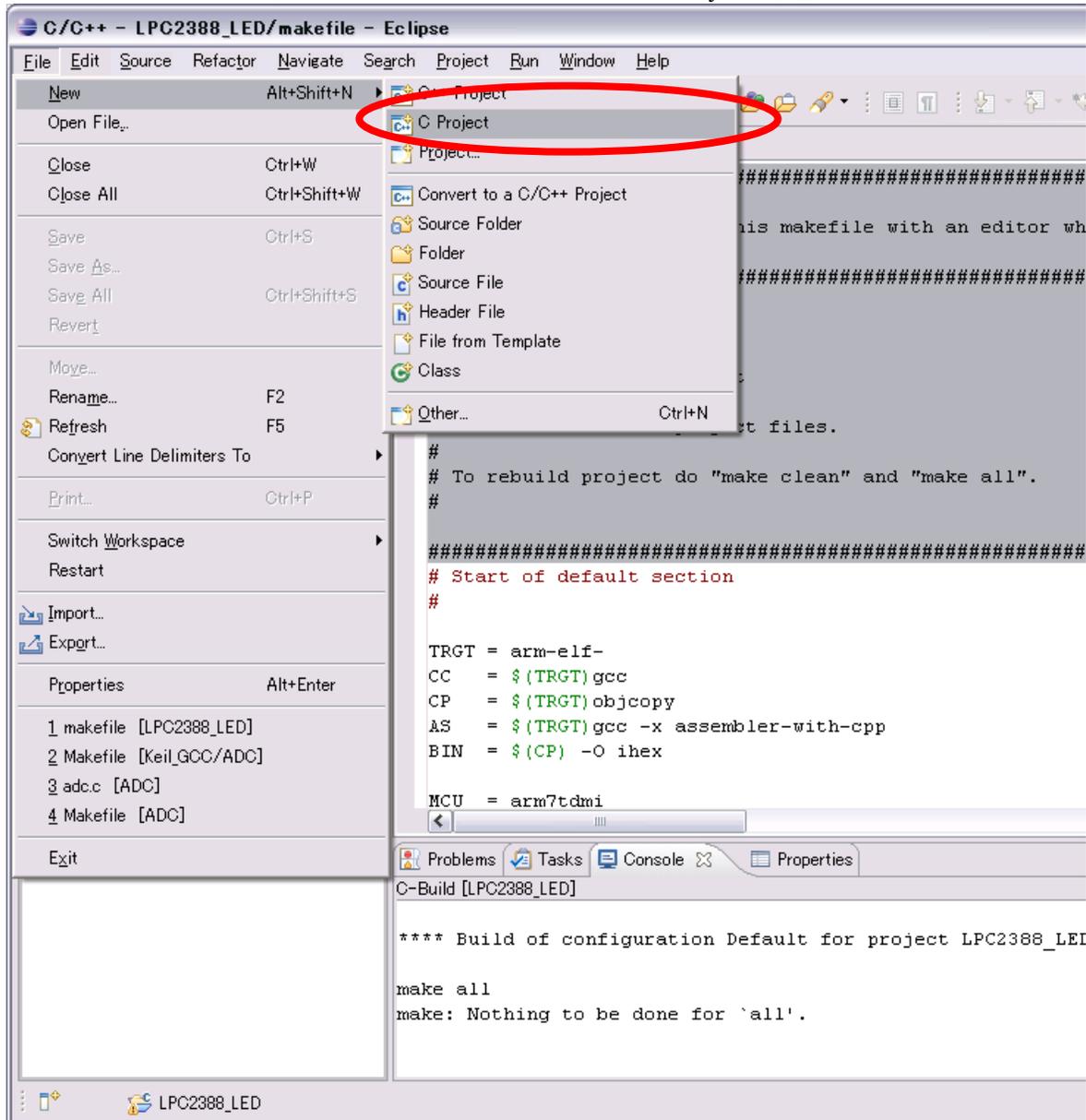
<http://www.dragonwake.com/download/LPC2148/Example-2148GCC.rar>

*サンプルの導入について、二つ方法があります、一つは新規プロジェクトを作成してからソースを導入、もう一つはダウンロードプロジェクトをインポートします。

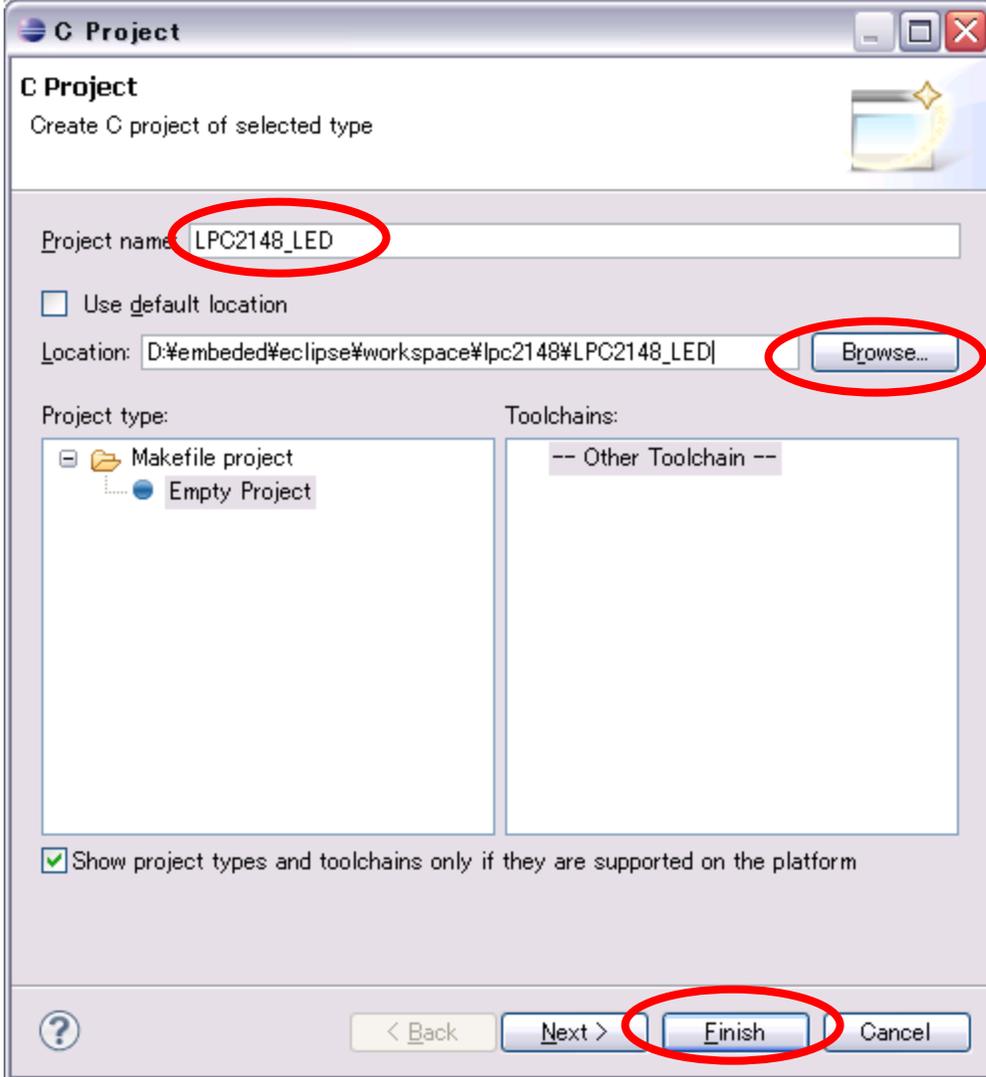
*ここに新規プロジェクトの作成として説明

1. プロジェクト「LPC2148_LED」を作る

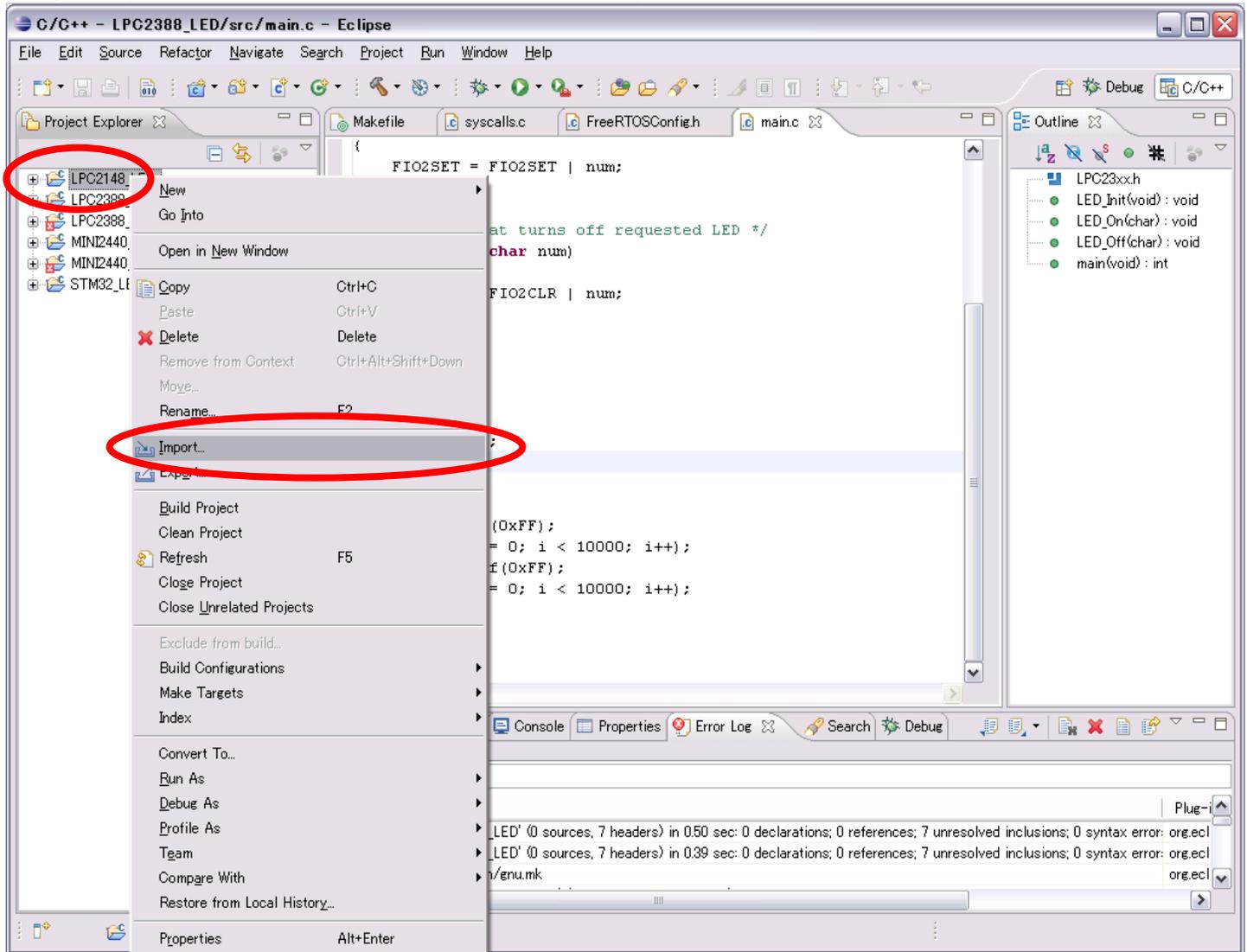
a)新規プロジェクトを作成するため"File"→"New"→"C Project"を選択します



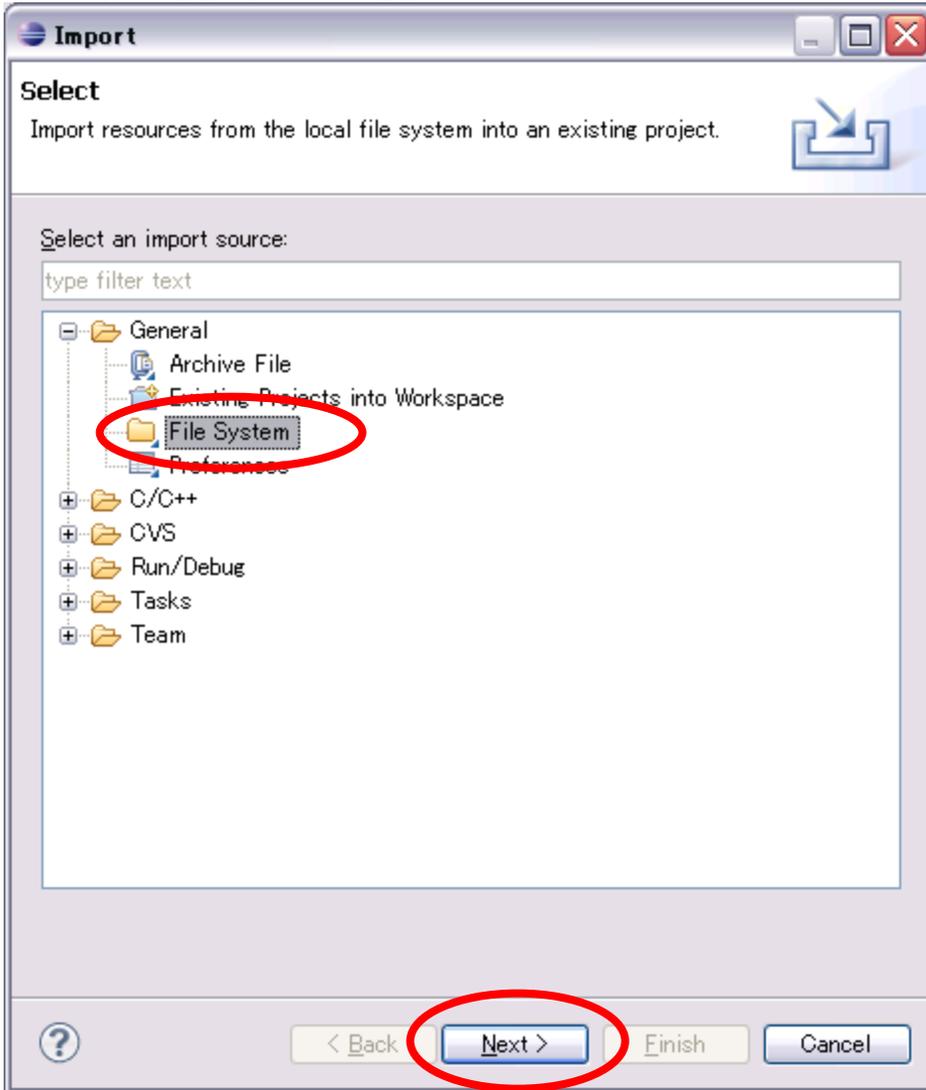
b) 適当なプロジェクト名前(LPC2148_LED)を入力し Finish ボタンを押します。



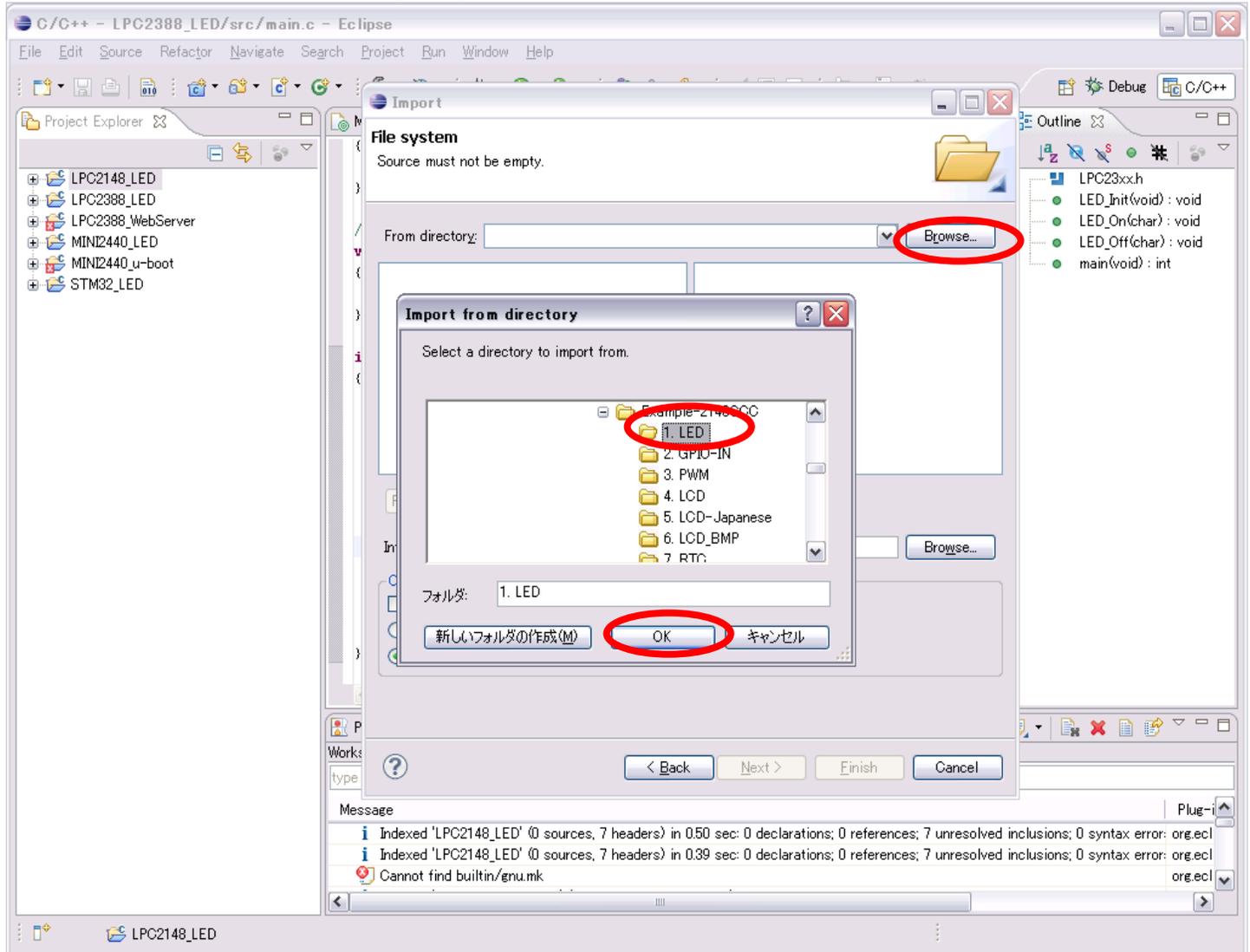
c) サンプルソースを作成されたプロジェクトにインポート
Project Explorer の「LPC2148_LED」を選べ、右クリックし、「Import」を押下



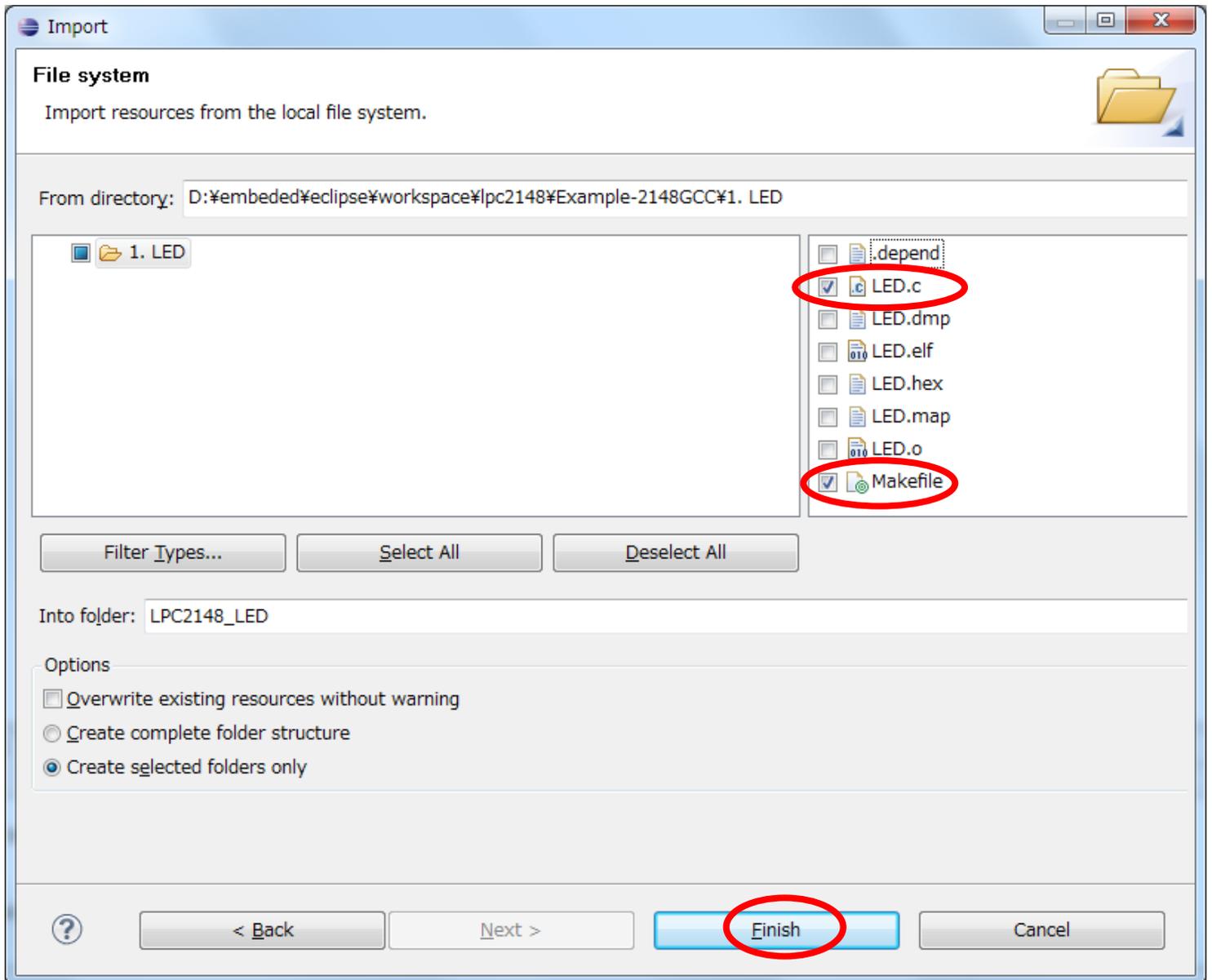
d)「File System」を選択し、「Next」をクリック



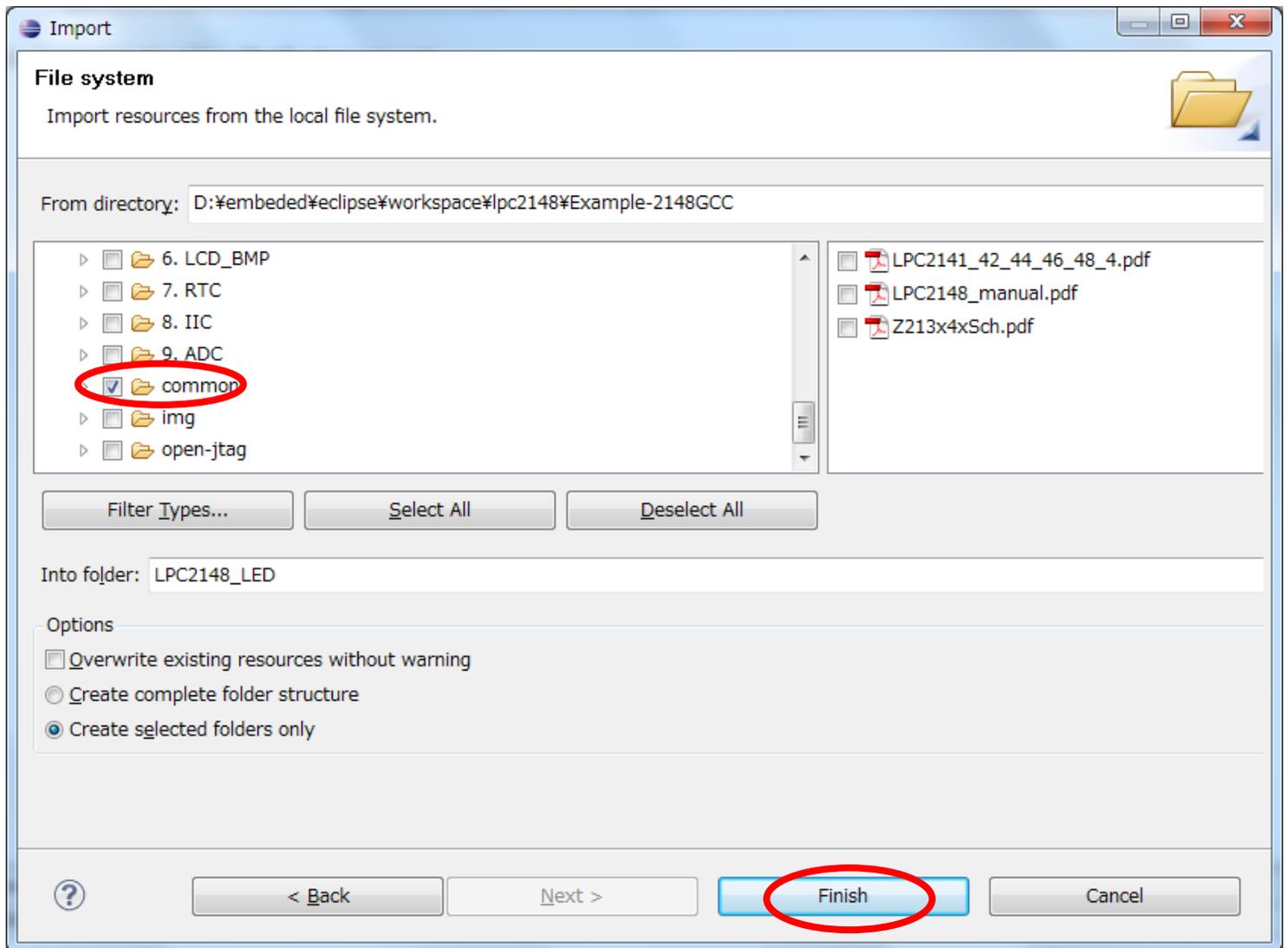
e) ダウンロードしたサンプル LPC2148_LED を選択



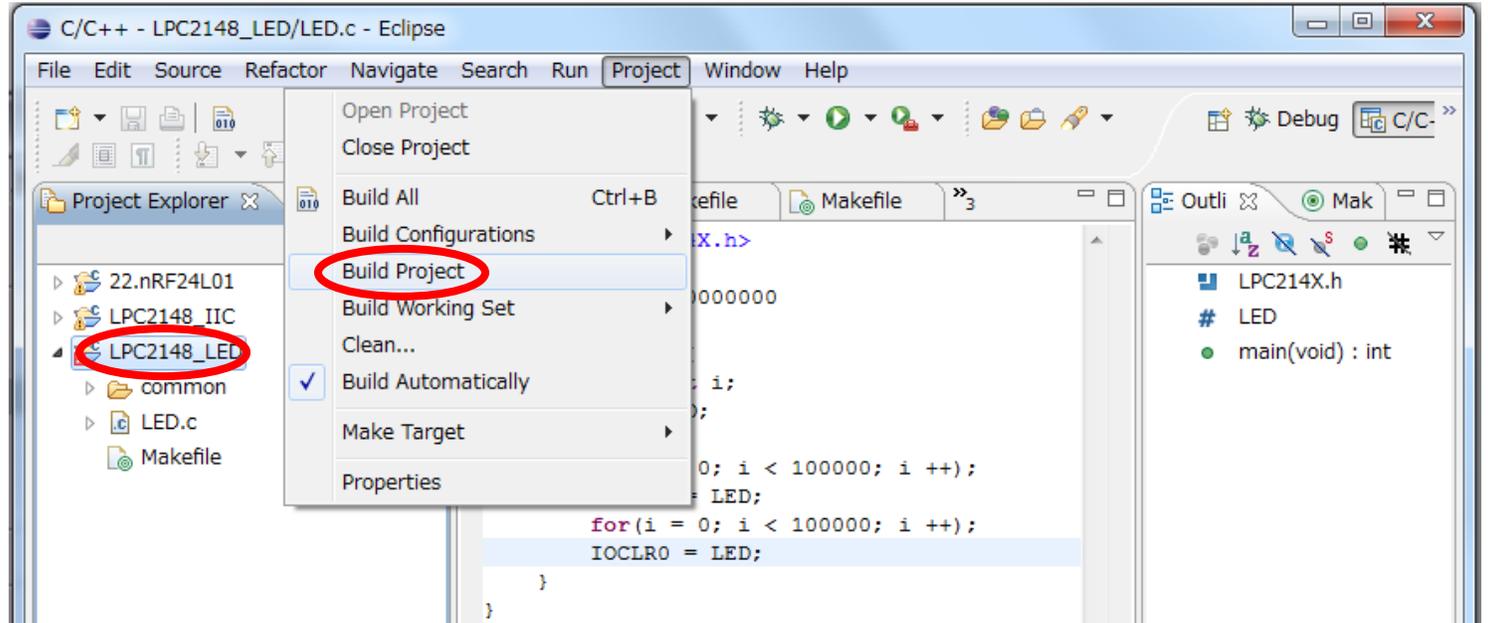
f) ソースファイルと Makefile を導入



g) 共通のソースも導入(他の操作は上記 c)、d)と同じ)

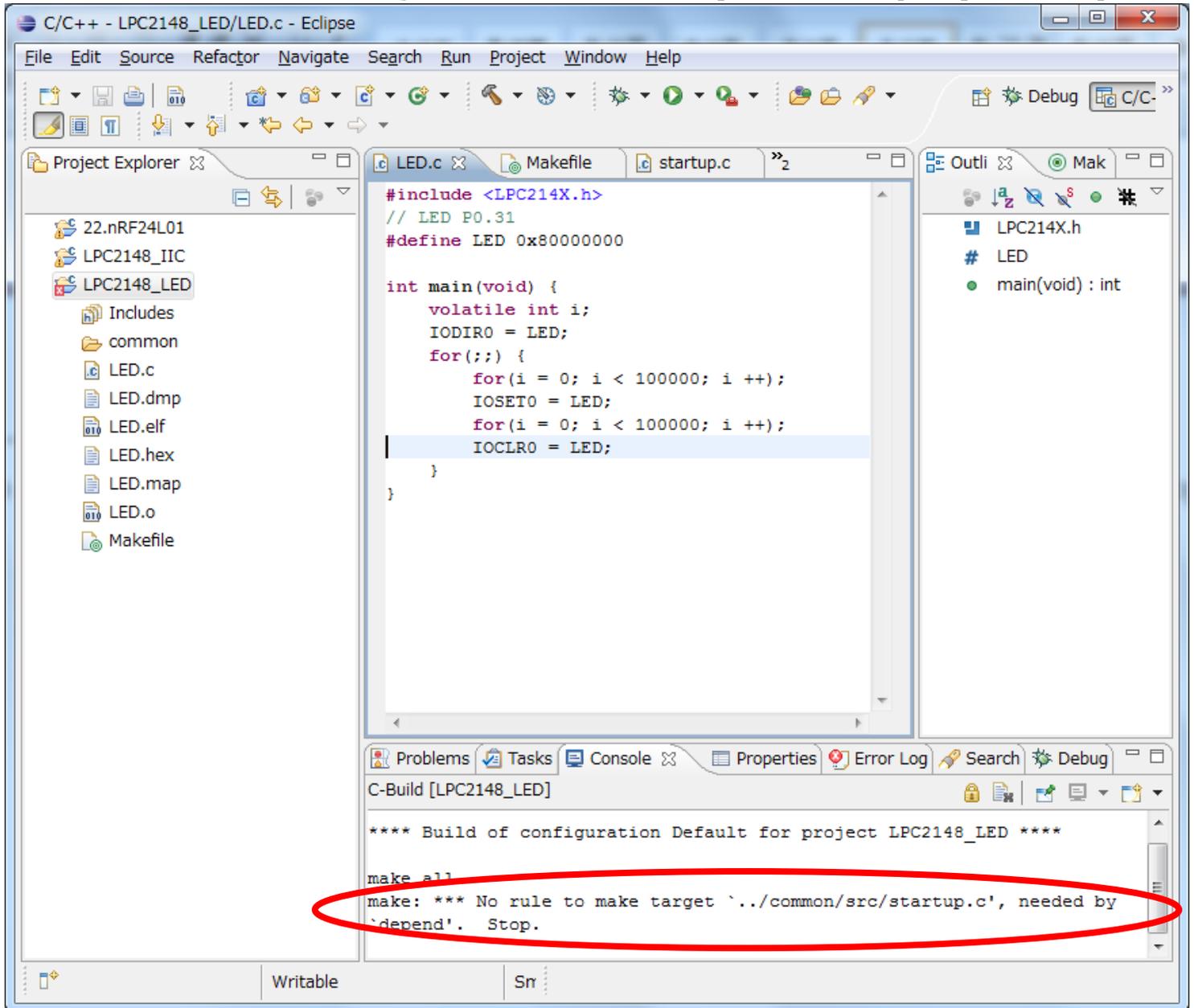


h) サンプル LPC2148_LED をビルド



i)ビルドエラーメッセージ (エラーメッセージがなければ、i) を飛ばして j)を直接参照)

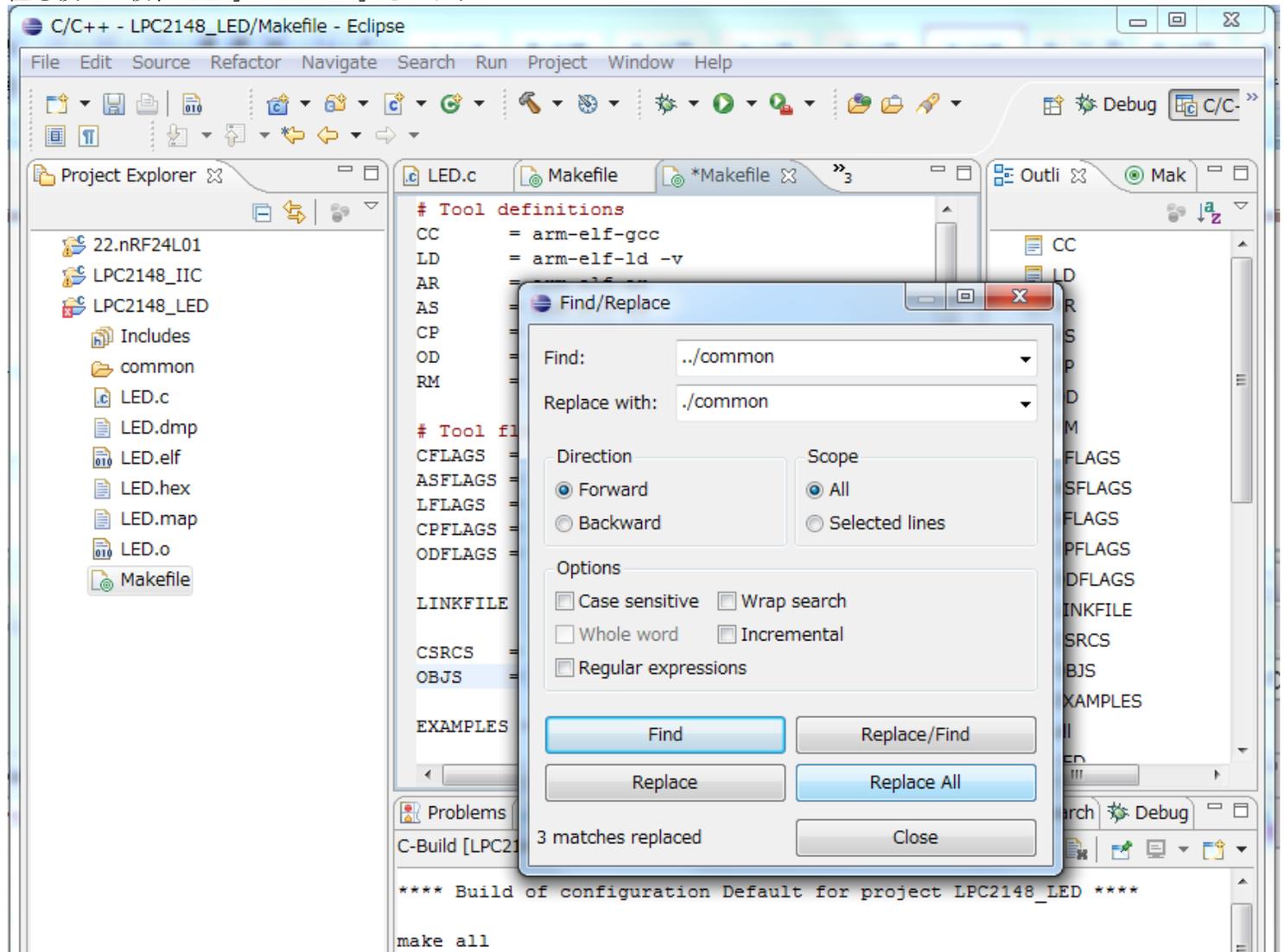
「make: *** No rule to make target `../common/src/startup.c', needed by `depend'. Stop.」



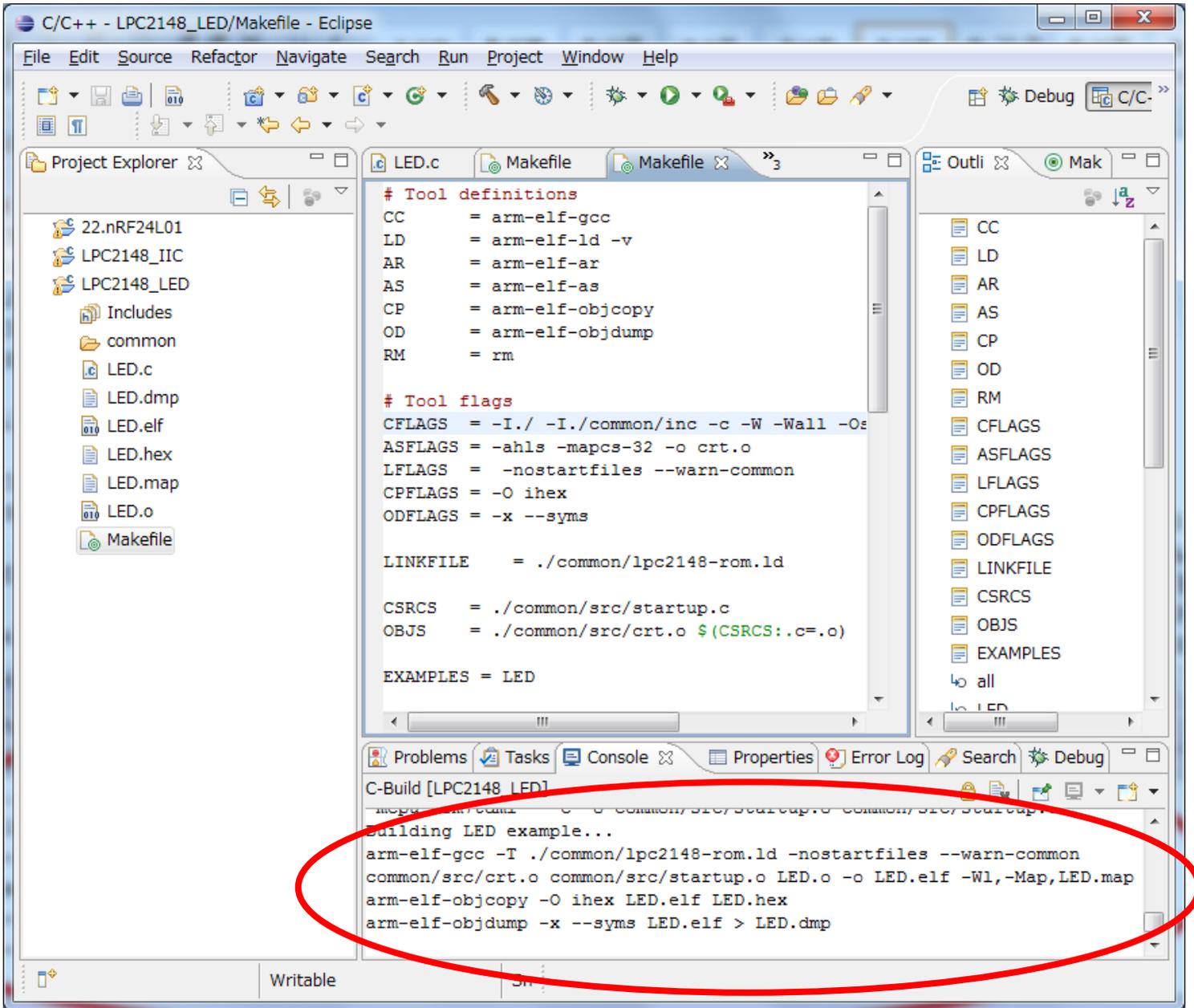
j) 「common」 こういうソースのパスを「Makefile」に修正

「Edit」 → 「Find/Replace」

「Find」 : 「../common」 を入力、「Replace with」 : 「./common」 を入力、「Replace All」 をクリック
置き換えの後、「File」 → 「Save」 をクリック



k)修正後、再度ビルド（赤丸に囲まれるメッセージが出てきれば、ビルドが成功ということを判明）



```
C/C++ - LPC2148_LED/Makefile - Eclipse
File Edit Source Refactor Navigate Search Run Project Window Help
Project Explorer LED.c Makefile Makefile x 3 Outli Mak
22.nRF24L01
LPC2148_IIC
LPC2148_LED
Includes
common
LED.c
LED.dmp
LED.elf
LED.hex
LED.map
LED.o
Makefile

# Tool definitions
CC      = arm-elf-gcc
LD      = arm-elf-ld -v
AR      = arm-elf-ar
AS      = arm-elf-as
CP      = arm-elf-objcopy
OD      = arm-elf-objdump
RM      = rm

# Tool flags
CFLAGS  = -I./ -I./common/inc -c -W -Wall -Oe
ASFLAGS = -ahls -mapcs-32 -o crt.o
LFLAGS  = -nostartfiles --warn-common
CPFLAGS = -O ihex
ODFLAGS = -x --syms

LINKFILE = ./common/lpc2148-rom.ld

CSRCS   = ./common/src/startup.c
OBS     = ./common/src/crt.o $(CSRCS:.c=.o)

EXAMPLES = LED

C-Build [LPC2148_LED]
building LED example...
arm-elf-gcc -T ./common/lpc2148-rom.ld -nostartfiles --warn-common
common/src/crt.o common/src/startup.o LED.o -o LED.elf -Wl,-Map,LED.map
arm-elf-objcopy -O ihex LED.elf LED.hex
arm-elf-objdump -x --syms LED.elf > LED.dmp
```

2. openocd 設定の修正

一般の設定は「5.4 OpenOCD の設定」を参照

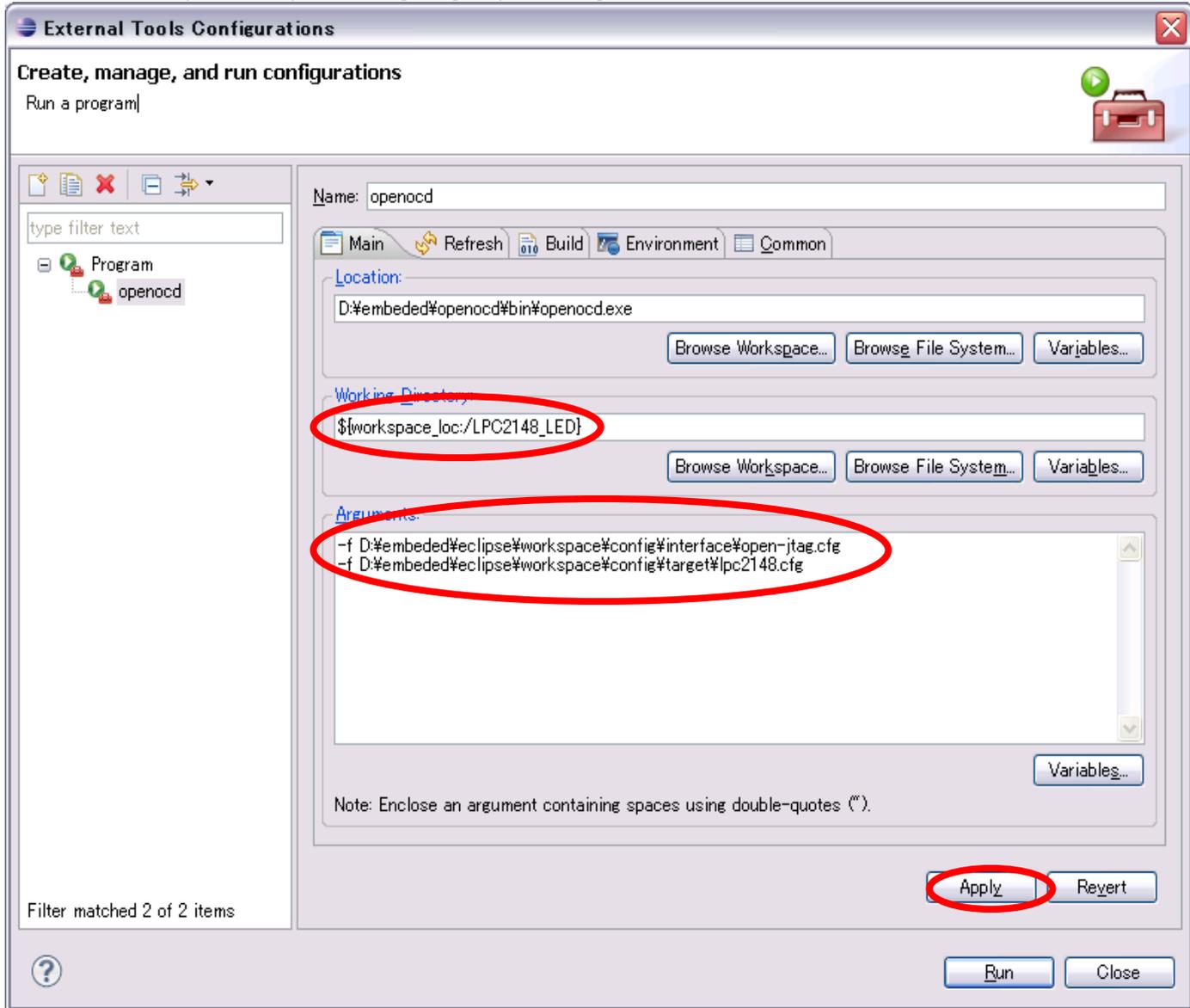
a) 「Working Directory」を「LPC2148_LED」に変更

b) 「Arguments」を下記のように変更

-f "D:\%embedded%\eclipse%\workspace%\config%\interface%\open-jtag.cfg"

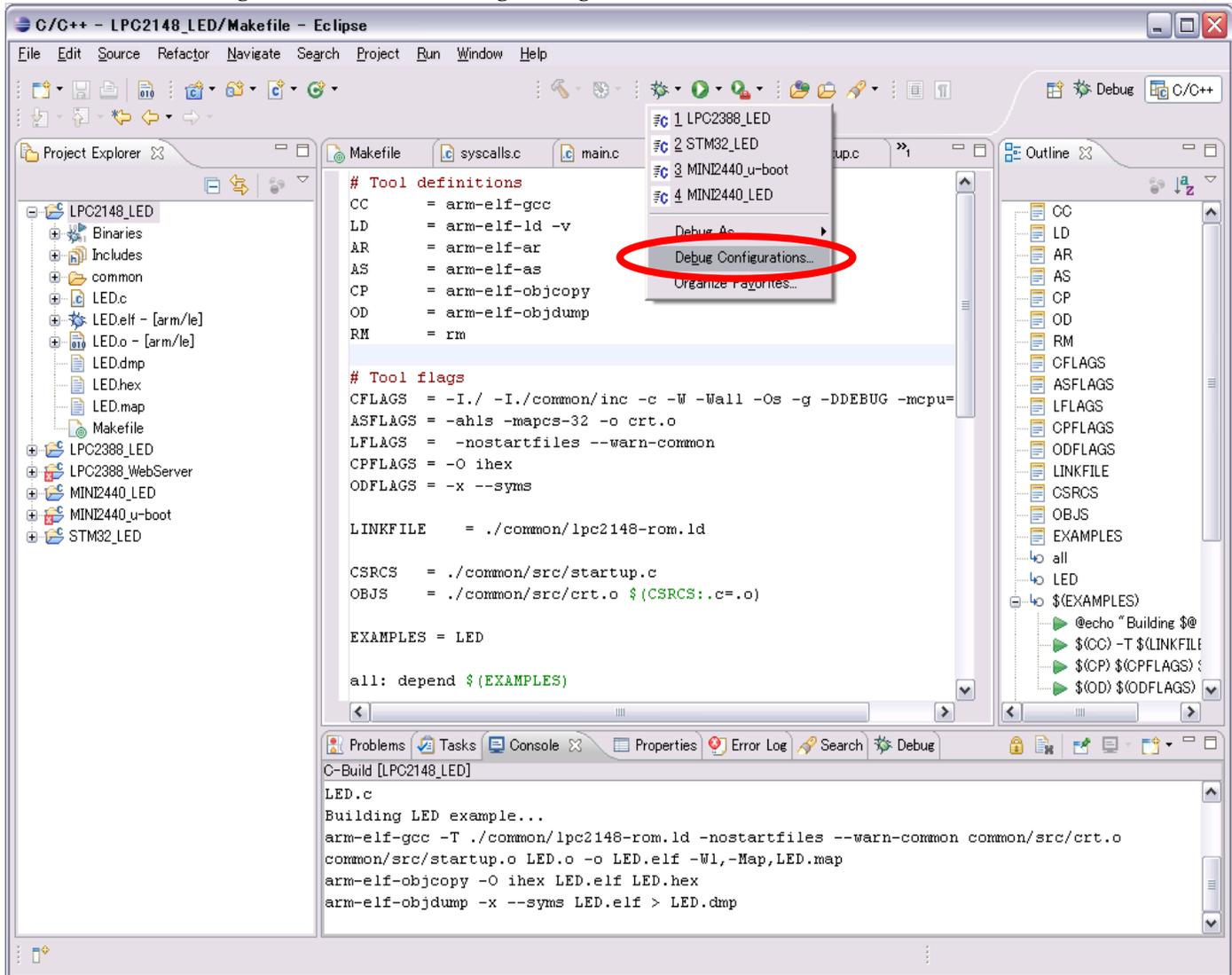
"D:\%embedded%\eclipse%\workspace%\config%\target%\lpc2148.cfg"

-f

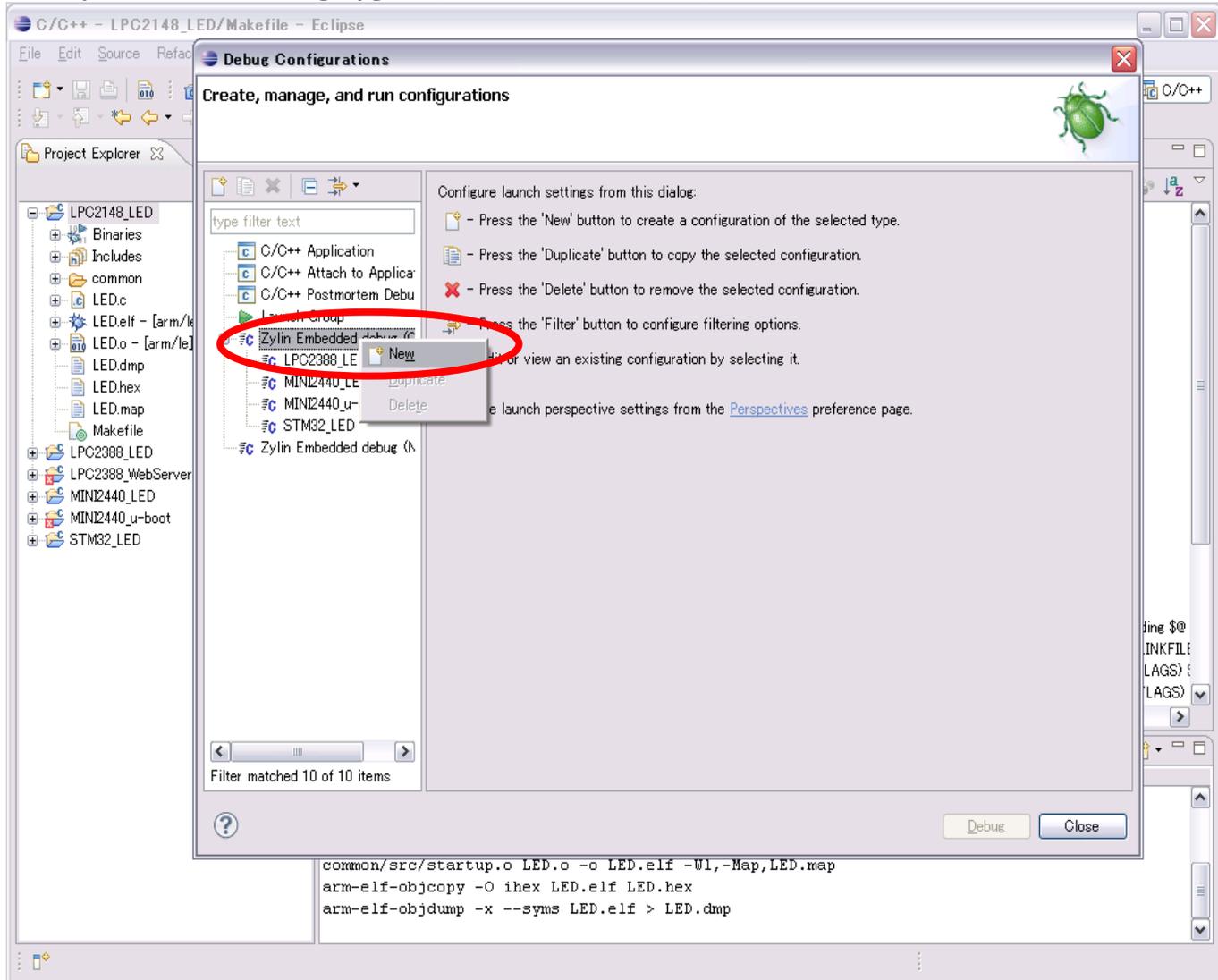


3. LPC2148_LED用のGDBを追加

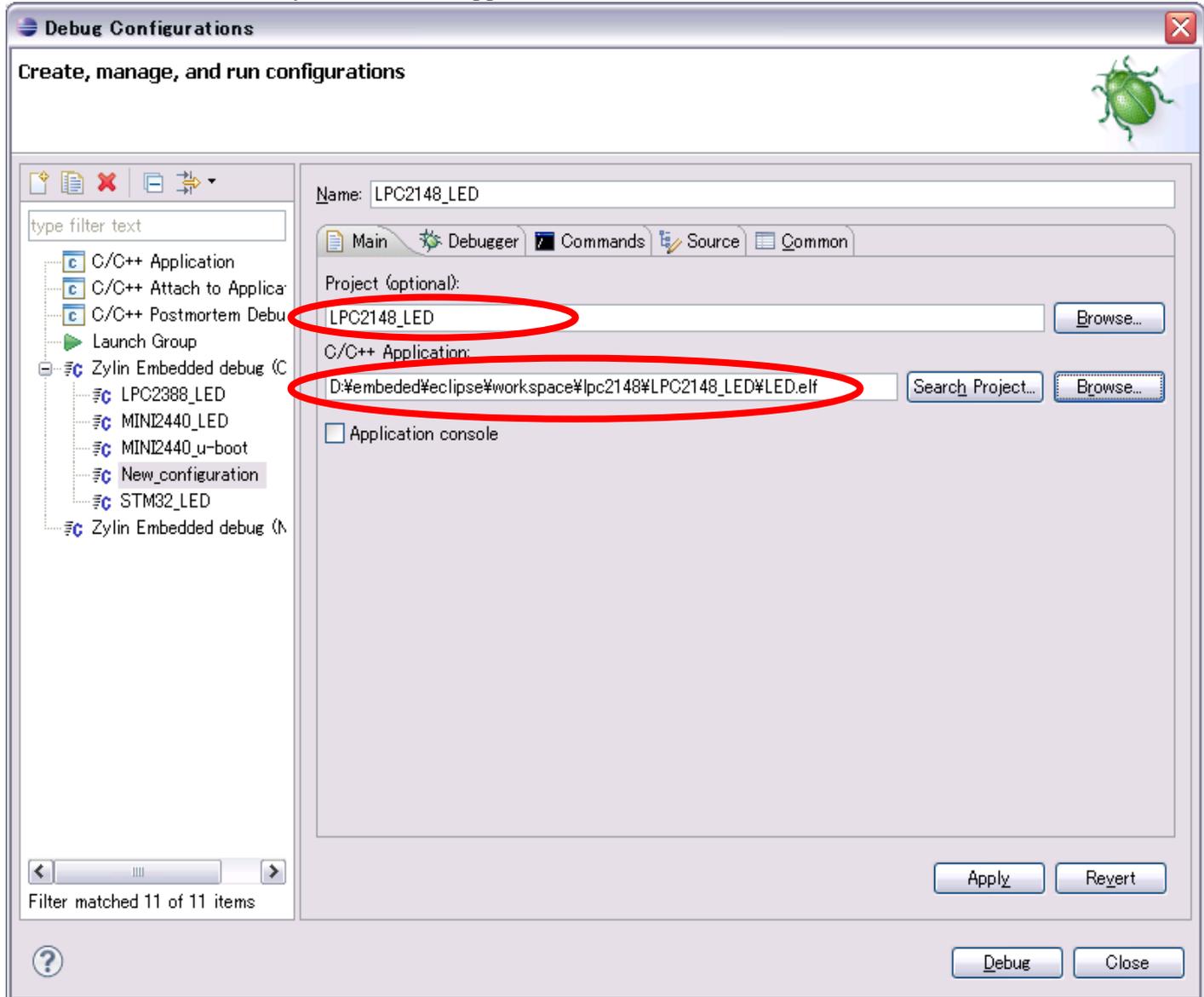
a)アイコン「Debug」をクリック、「Debug Configuration」を押下



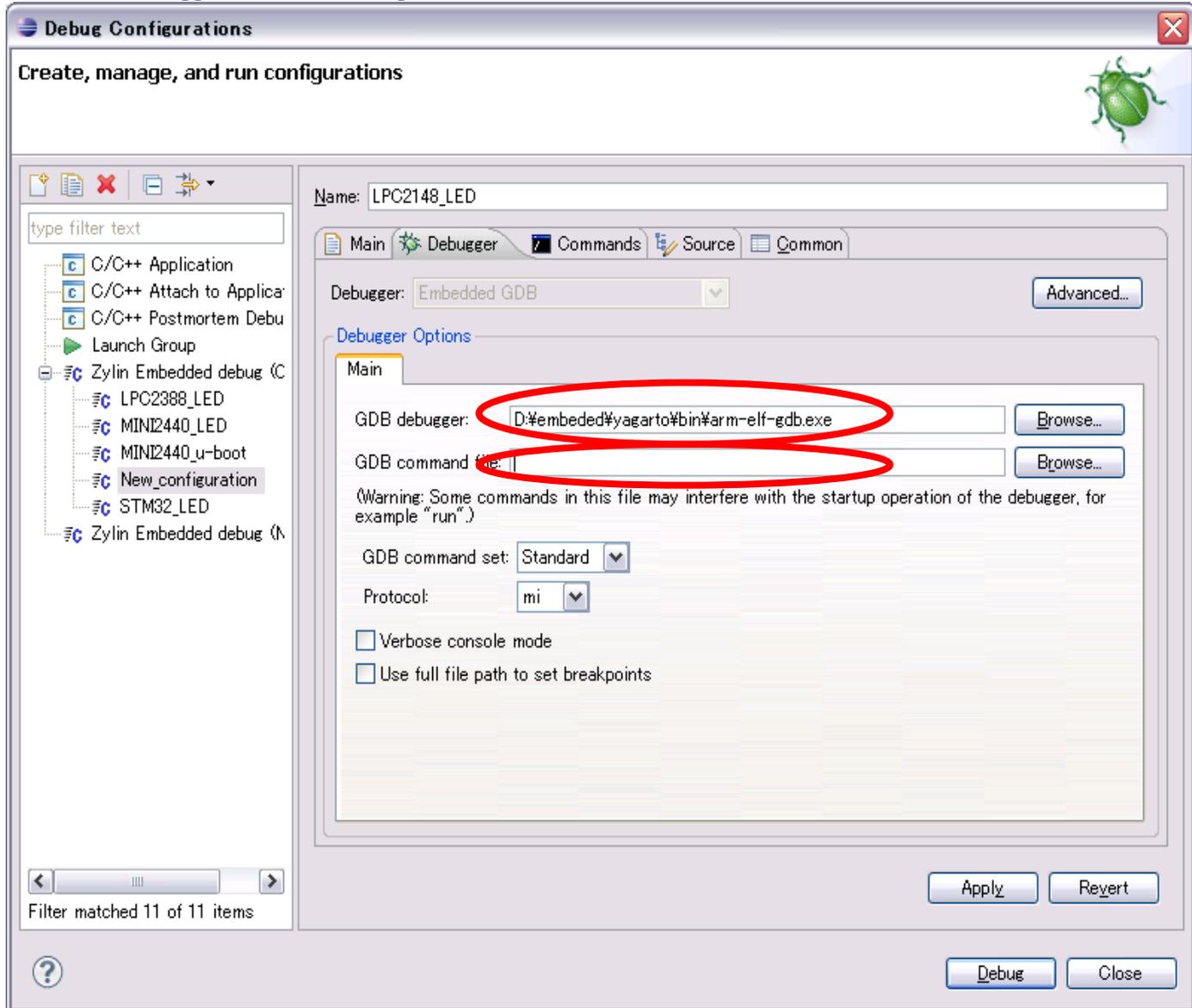
b) 「Zylin Embedded debug(Cygwin)」を右クリック、「New」を選択



c) 「Name」を入力、「Project」、「C/C++ Application」(elfファイル指定)を指定



d) 「GDB debugger」を「arm-elf-gdb.exe」の場所に指定



e) 「Commands」を入力 (//の右はコメントです、Eclipse にコピーしないでください。)

```
target remote localhost:3333 // ローカルポート「3333」と接続 (OpenOCDと接続)
```

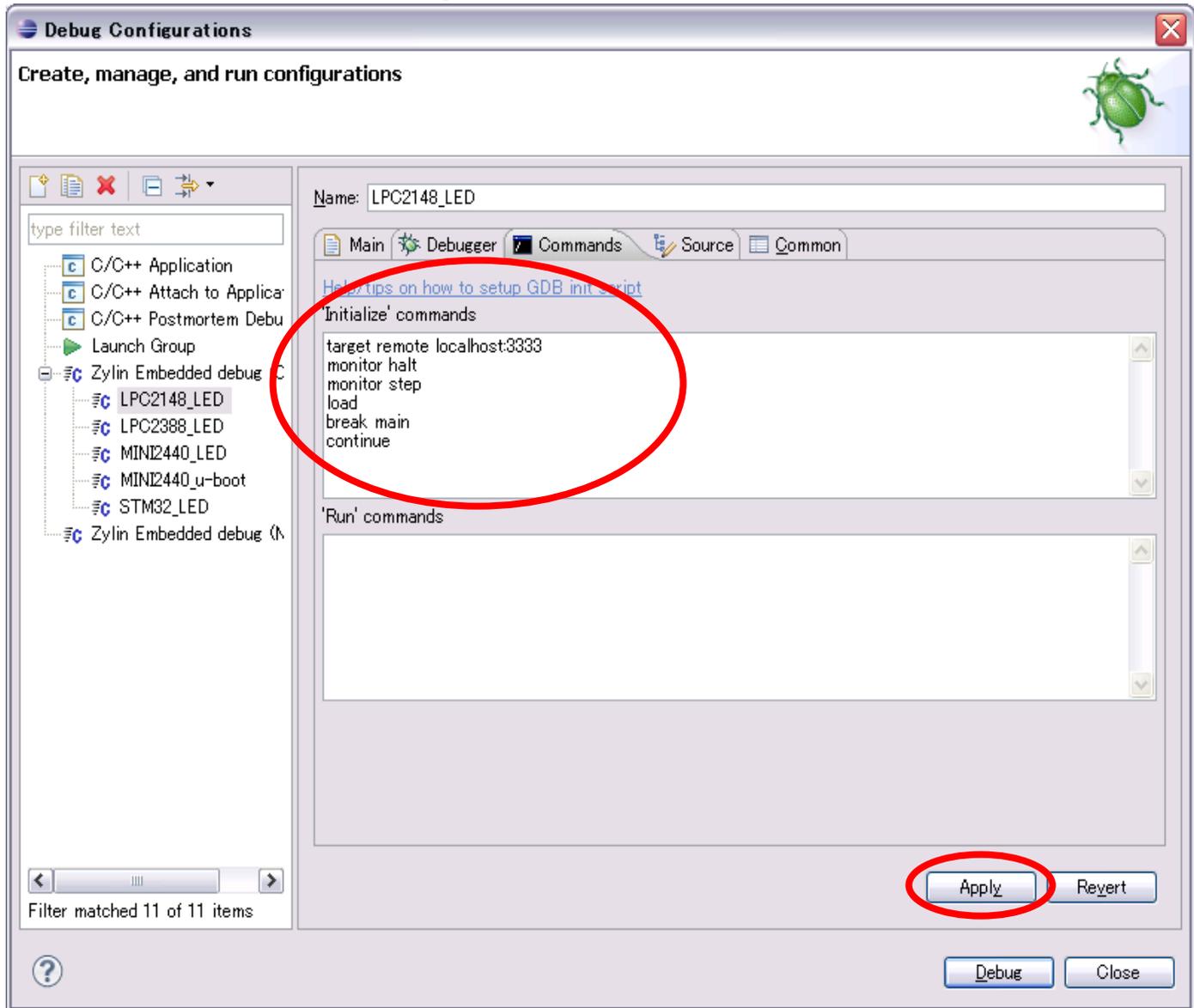
```
monitor halt//ボードの実行を中断させる
```

```
monitor step//ステップで実行するように
```

```
load // leds_elfをロード, 「elf」というフォーマットのファイルにアドレスが含まれる
```

```
break main // 「main」関数にブレークポイントを設定
```

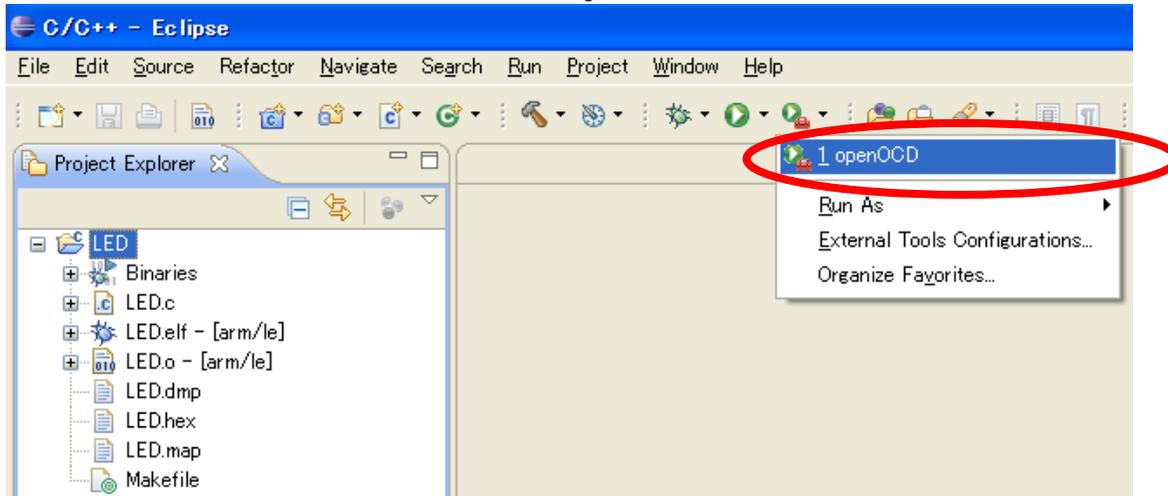
```
continue // プログラムを実行させて、「main」にとまってステップでデバッグ可能
```



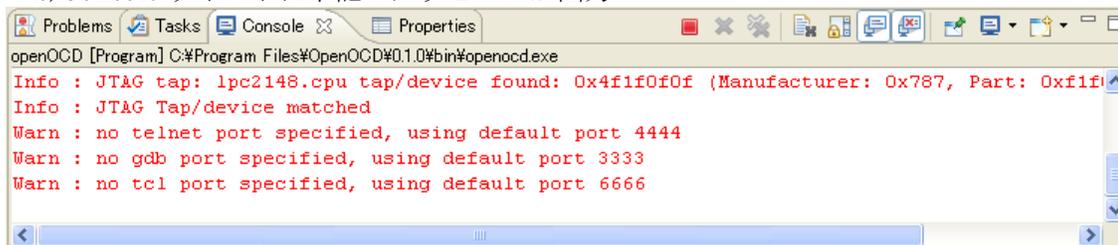
4. デバッグ

a) OpenOCD を起動

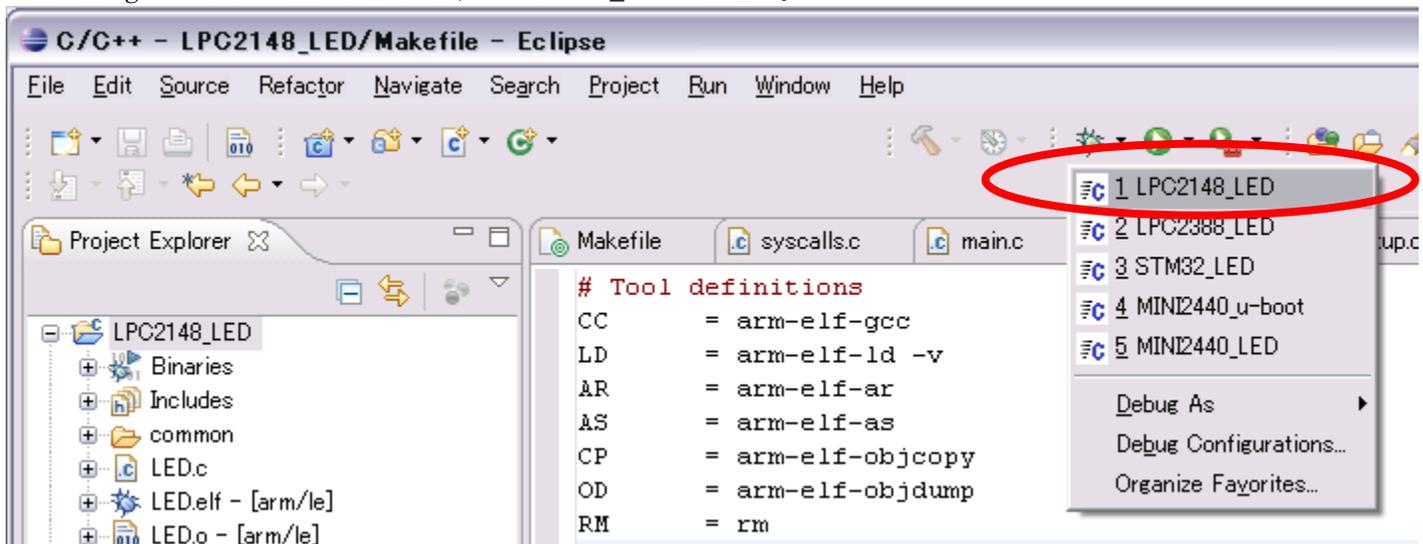
External Tools の▼ボタンをクリックし、OpenOCD を選択



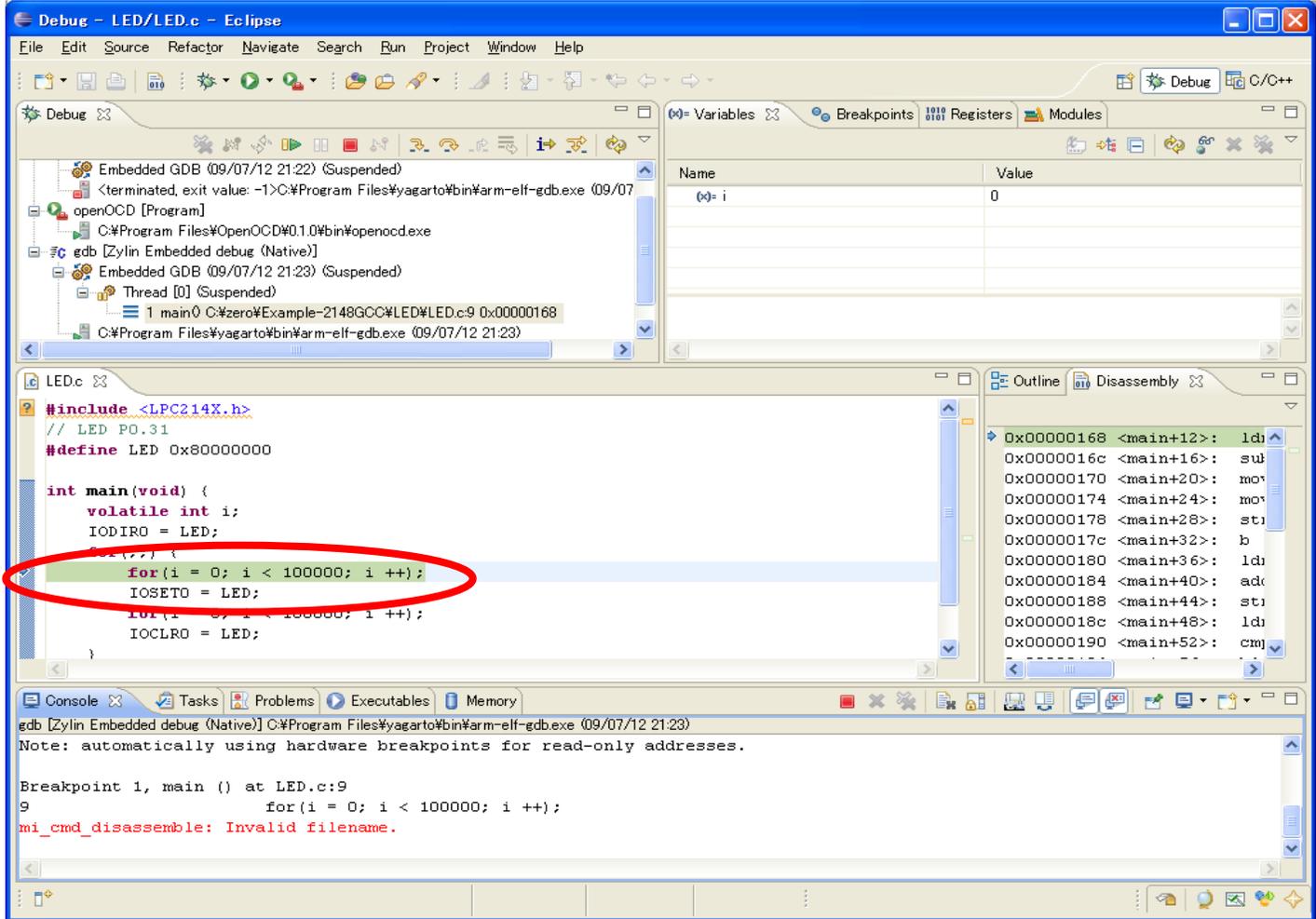
b) Console ウィンドに下記のメッセージが出力



c) Debug の▼ボタンをクリックし、"LPC2148_LED"を選択。



d) 「main」関数に止まる



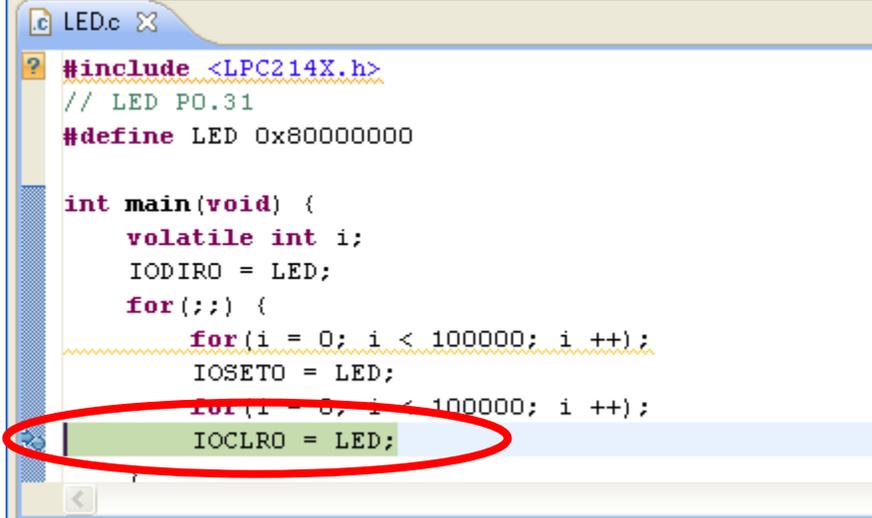
The screenshot shows the Eclipse IDE interface with the following components:

- Debugger View:** Shows the execution stack with 'main' at the top, indicating the program has stopped at the specified location.
- Variables View:** Shows a single variable 'i' with a value of 0.
- Source Editor:** Displays the C code for 'LED.c'. The line `for(i = 0; i < 100000; i ++);` is highlighted with a red circle.
- Disassembly View:** Shows the assembly instructions corresponding to the current line of code.
- Console:** Displays the following text:

```
gdb [Zylin Embedded debug (Native)] C:\Program Files\yagarto\bin\arm-elf-gdb.exe (09/07/12 21:23)
Note: automatically using hardware breakpoints for read-only addresses.

Breakpoint 1, main () at LED.c:9
9          for(i = 0; i < 100000; i ++);
mi_cmd_disassemble: Invalid filename.
```

e) ブレークポイントでプログラムが中断した状態から、次のブレークポイントまで実行させたり、1行ずつ実行させたりできます。コードを繰り返して実行することにより、LED ランプが点滅します。



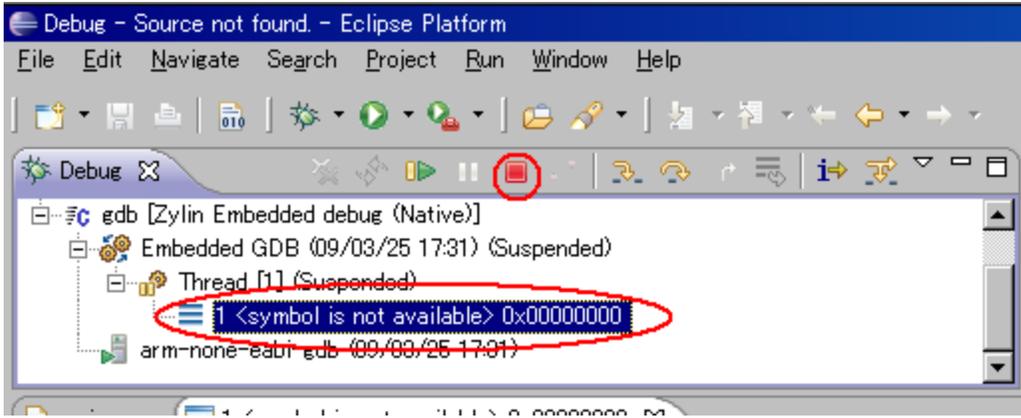
```
LEDc
#include <LPC214X.h>
// LED PO.31
#define LED 0x80000000

int main(void) {
    volatile int i;
    IODIRO = LED;
    for(;;) {
        for(i = 0; i < 100000; i ++);
        IOSETO = LED;
        for(i = 0; i < 100000; i ++);
        IOCLR0 = LED;
    }
}
```

5. デバッグ終了

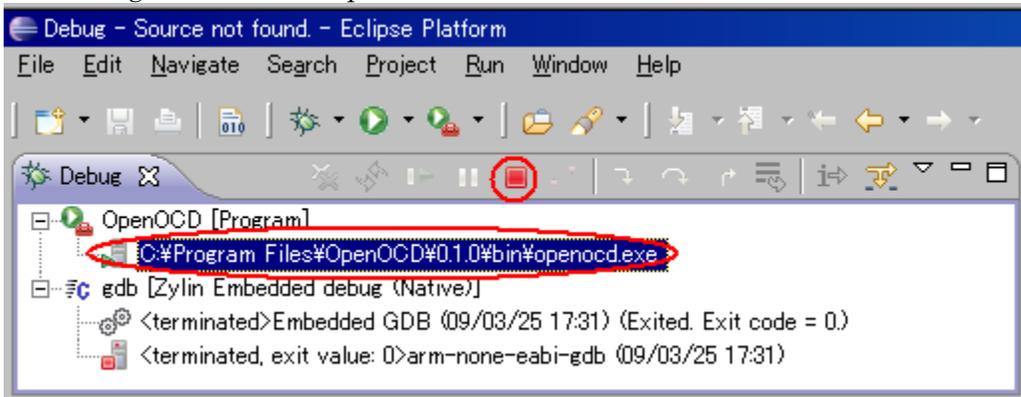
a) gdb の停止

Debug ウィンドウの gdb の Thread を選択し、停止ボタンと押します



b) OpenOCD の停止

Debug ウィンドウの OpenOCD の Thread を選択し、停止ボタンと押します



c) 電源停止

ターゲットの電源を停止

d) OpenJTAG をターゲットから取り外す

e) 上記が面倒であれば Eclipse を終了しターゲットの電源停止、open-JTAG を取り外しても OK です。

6.2 ARM7 の LPC2388

6.2.1 LPC2388 ボード購入 URL

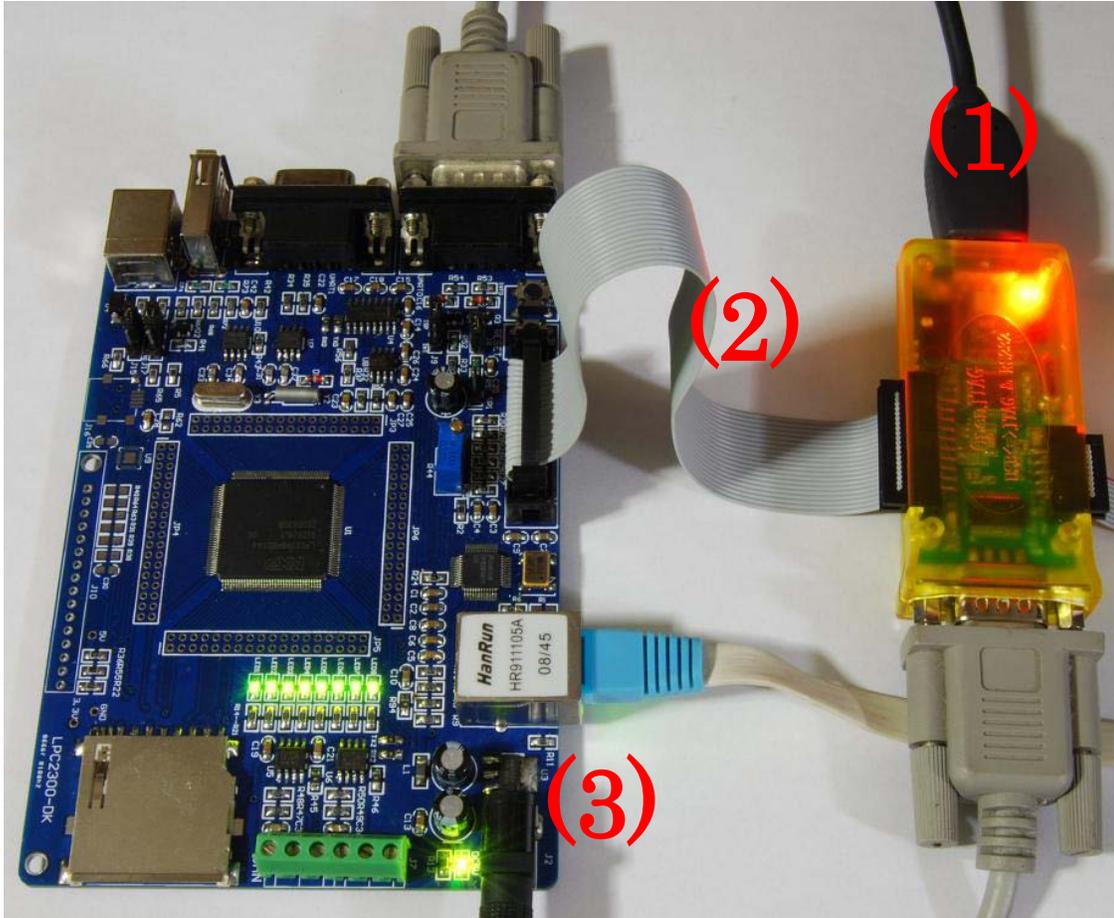
<http://www.csun.co.jp/SHOP/200903021.html>

ボード URL から LPC2388 マニュアルを参照し、Open-JTAG に接続してください。

LPC2388 マニュアルダウンロード URL :

http://www.dragonwake.com/download/LPC2388/LPC2388_manual.pdf

6.2.2 ハードウェア動作確認



- (1). OpenJTAG をパソコンの USB ポートに挿入する
- (2). JTAG ケーブルで OpenJTAG と LPC2388 ボードを繋ぐ
- (3). LPC2388 ボードに電源を入れる
- (4). 下記のコマンドを入力します。

```
openocd -f "D:\¥embedded¥eclipse¥workspace¥config¥interface¥open-jtag.cfg" -f "D:\¥embedded¥eclipse¥workspace¥config¥target¥lpc2388.cfg"
```

※ はじめの-fは OpenJTAG のコンフィグファイルを使います。二番目の-fは lpc2388 のコンフィグファイルを使います。

```
C:\> C:\WINDOWS\system32\cmd.exe - openocd -f D:\embedded\eclipse\workspace\config...
BUGS? Read http://svn.berlios.de/svnroot/repos/openocd/trunk/BUGS

$URL: https://kc8apf@svn.berlios.de/svnroot/repos/openocd/tags/openocd-0.1.0/src
/openocd.c $
Can't find D:\embedded\eclipse\workspace\config\interface\open-jtag.cfg-f

C:\Documents and Settings\dragonwake>openocd -f D:\embedded\eclipse\workspace\con
fig\interface\open-jtag.cfg -f D:\embedded\eclipse\workspace\config\target\lpc238
8.cfg
Open On-Chip Debugger 0.1.0 (2009-01-21-21:15) Release

BUGS? Read http://svn.berlios.de/svnroot/repos/openocd/trunk/BUGS

$URL: https://kc8apf@svn.berlios.de/svnroot/repos/openocd/tags/openocd-0.1.0/src
/openocd.c $
jtag_speed: 0
500 kHz
Info : JTAG tap: lpc2388.cpu tap/device found: 0x4f1f0f0f (Manufacturer: 0x787,
Part: 0xf1f0, Version: 0x4)
Info : JTAG Tap/device matched
Warn : EmbeddedICE version 7 detected. EmbeddedICE handling might be broken
```

画面のように"Info : JTAG Tap/device matched"と表示されればOKです
(この時点で ARM LPC2388 と通信が来ています)

(6). その後 CTRL+C を押してデバッグを中止します。

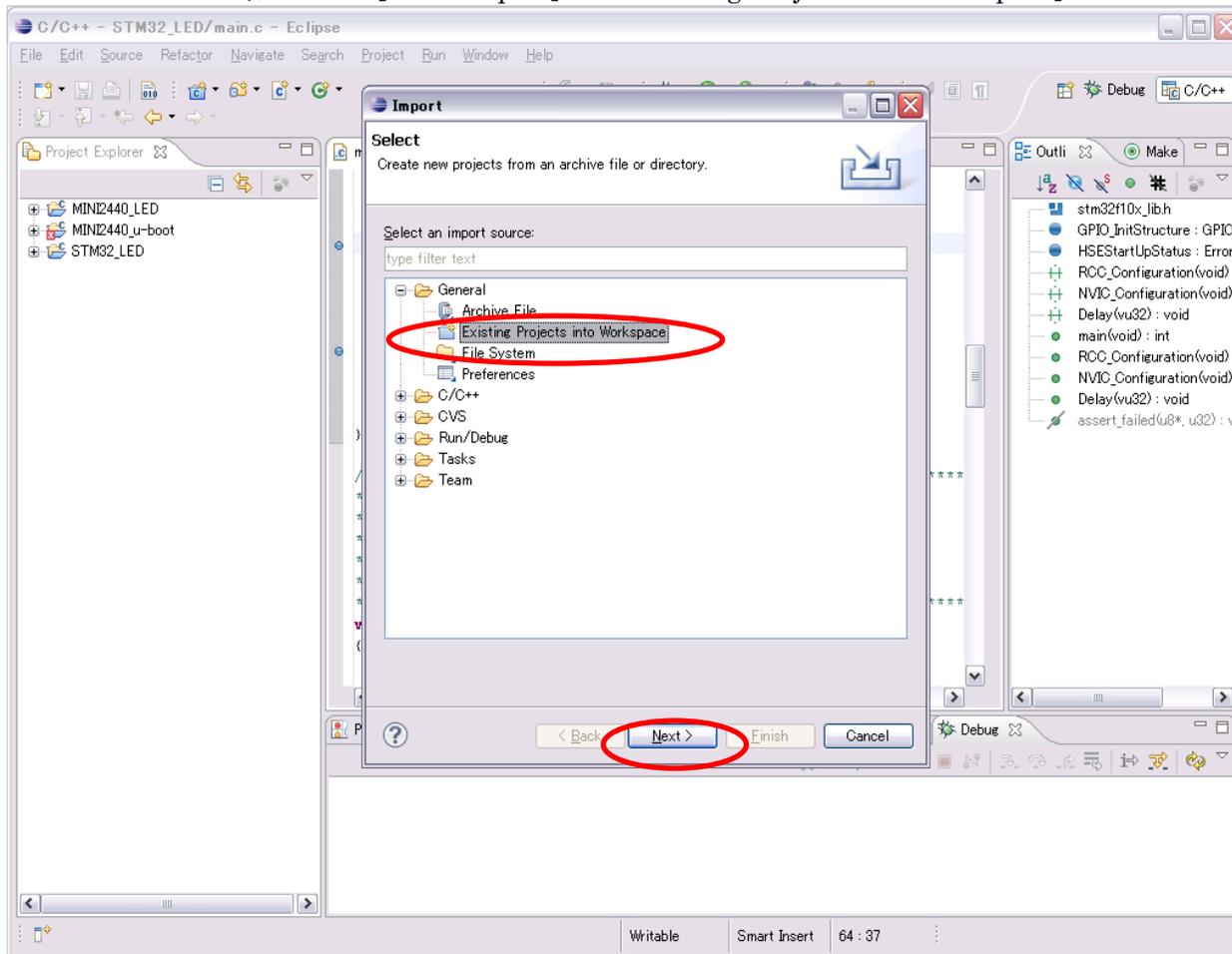
6.2.3 LPC2388 用のサンプル「LED」をデバッグ

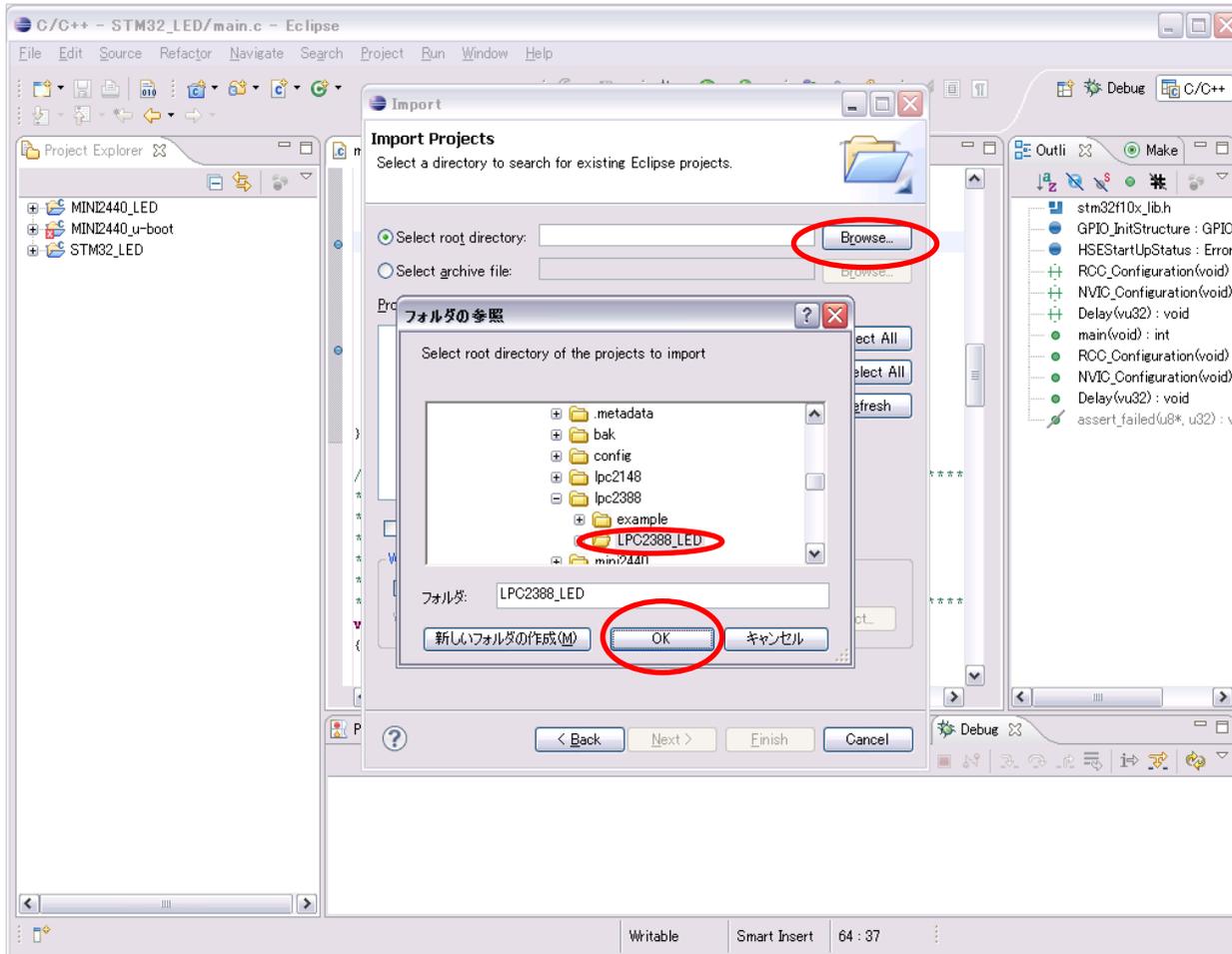
*サンプルの導入について、二つ方法があります、一つは新規プロジェクトを作成してからソースを導入、もう一つはダウンロードプロジェクトをインポートします。

*新規プロジェクトの作成方法は「6.1.3 LPC2148 用のサンプル「LED」をデバッグ」を参照

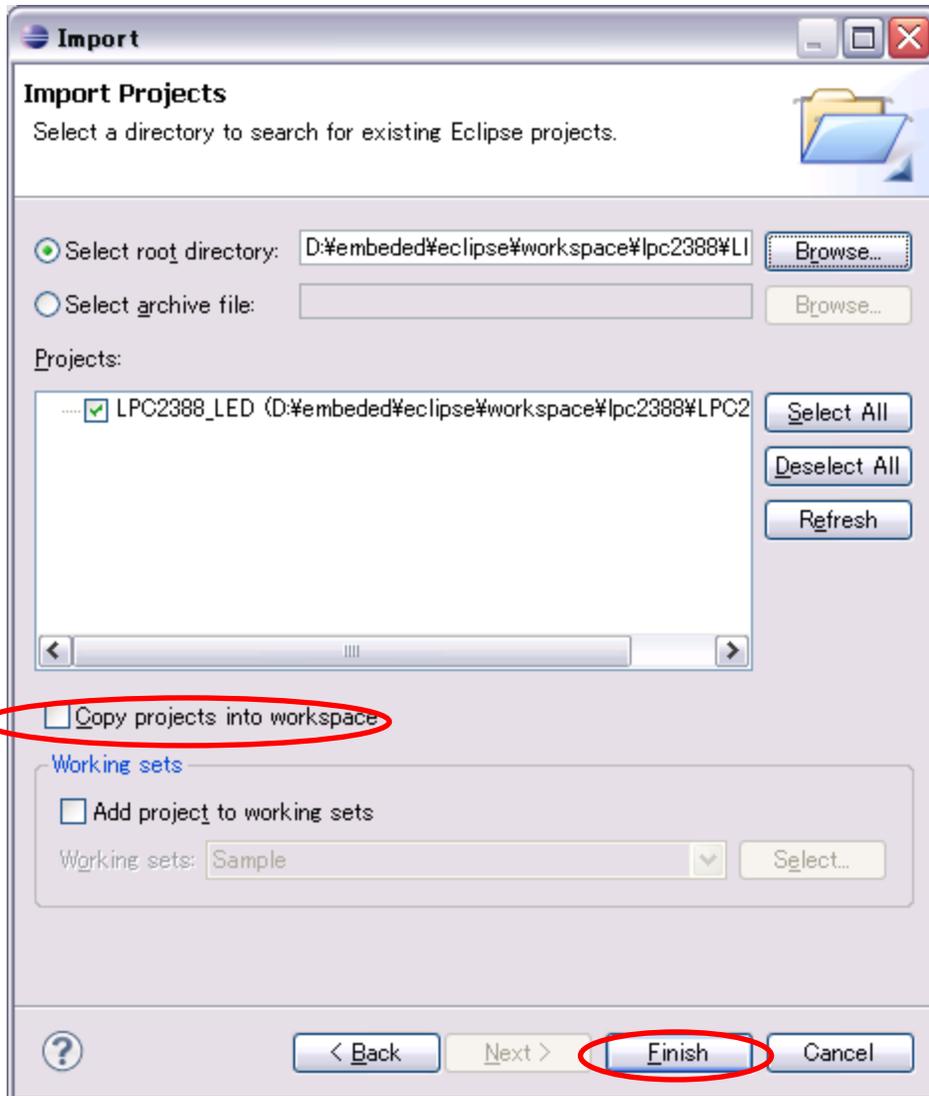
*ここに既存プロジェクトのインポートとして説明

1. プロジェクト導入：「File」→「Import」→「Existing Projects into workspace」



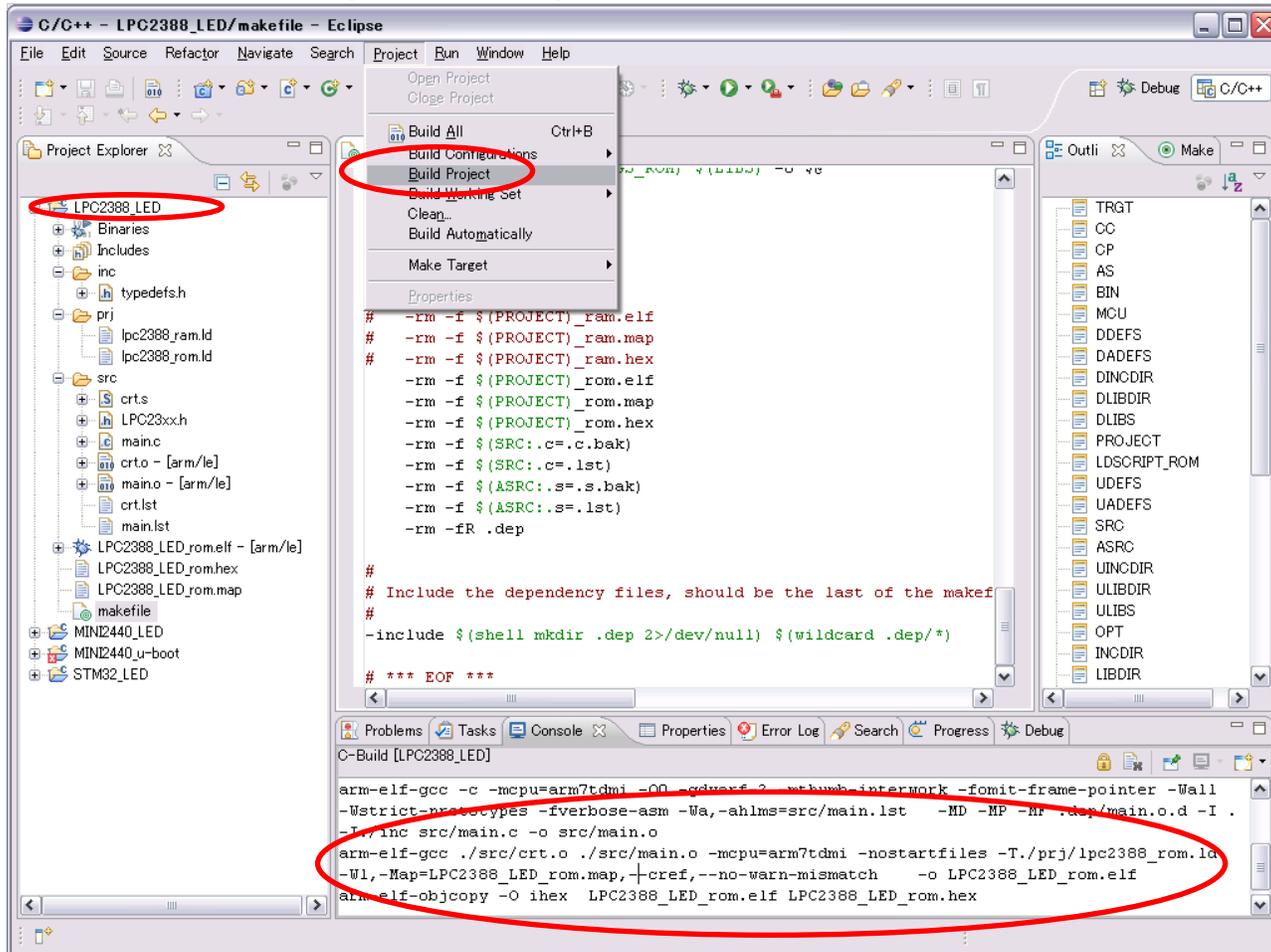


既にダウンロードしたプロジェクトを Workspace に入れていますので、ここに「Copy projects into workspace」を外します。Workspace に入っていない場合、「Copy projects into workspace」をチェックしてください。



2. 導入済みのプロジェクトをコンパイル

「LPC2388_LED」を選択、「Project」→「Build Project」を押下



コンパイルが成功すれば、実行ファイルLPC2388_LED_rom.elfとLPC2388_LED_rom.hexを生成されます。

3. openocd 設定の修正

一般の設定は「5.4 OpenOCD の設定」を参照

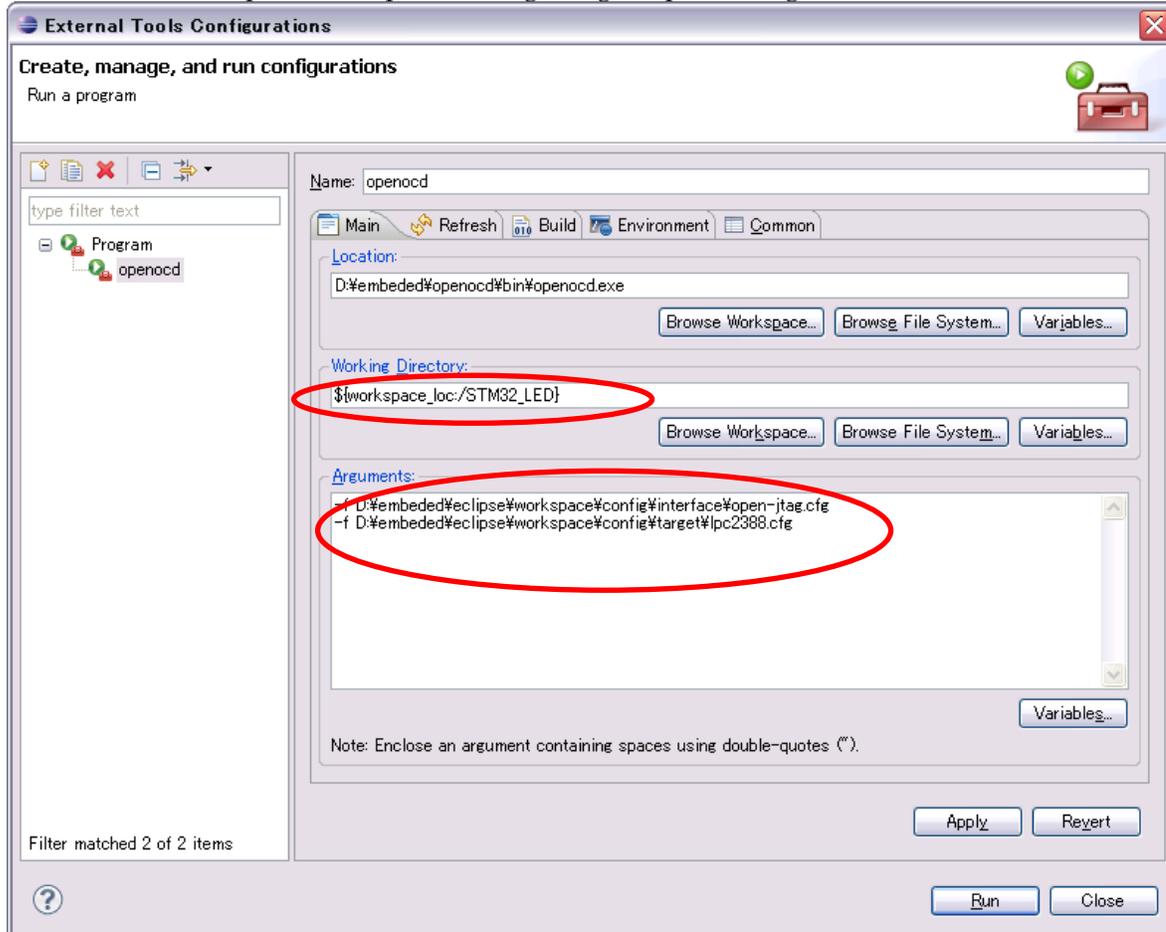
a) 「Working Directory」を「LPC2388_LED」に変更

b) 「Arguments」を下記のように変更

-f "D:\¥embedded¥eclipse¥workspace¥config¥interface¥open-jtag.cfg"

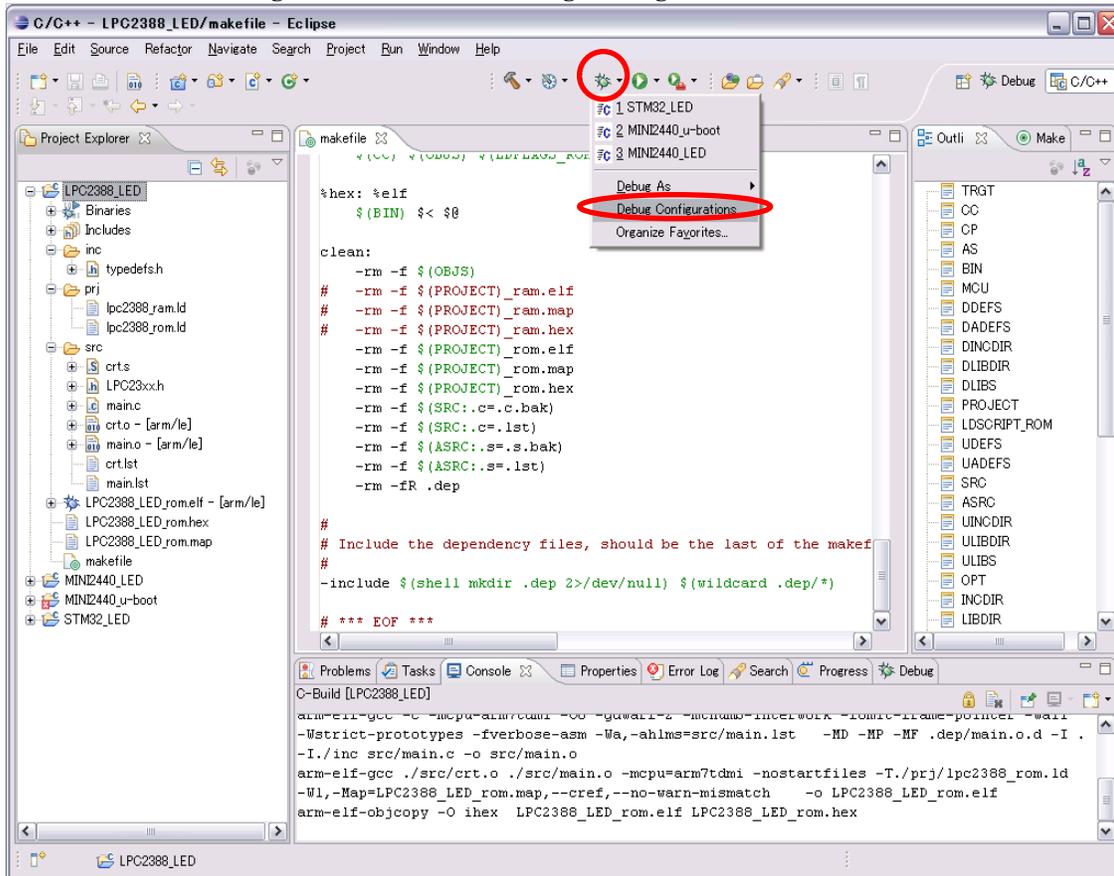
-f

"D:\¥embedded¥eclipse¥workspace¥config¥target¥lpc2388.cfg"

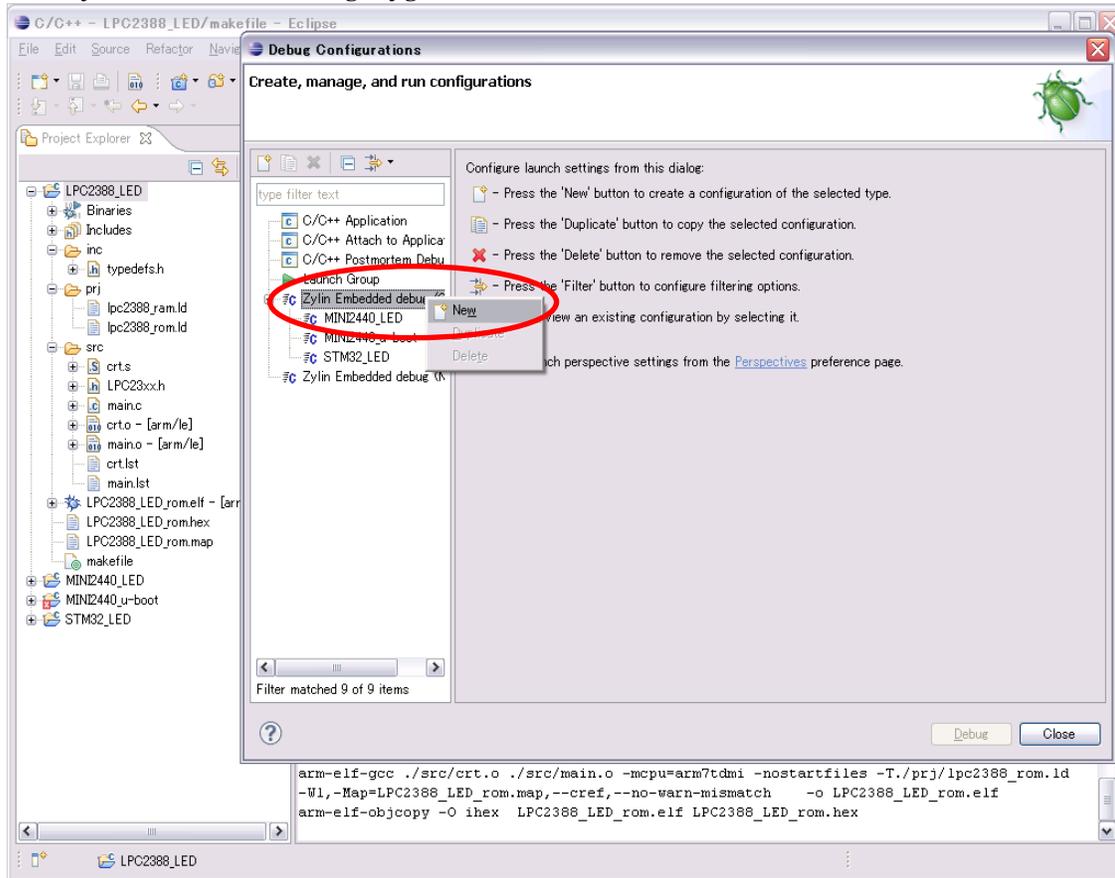


4. LPC2388_LED用のGDBを追加

①アイコン「Debug」をクリック、「Debug Configuration」を押下



② 「Zylin Embedded debug(Cygwin)」 を右クリック、「New」を選択



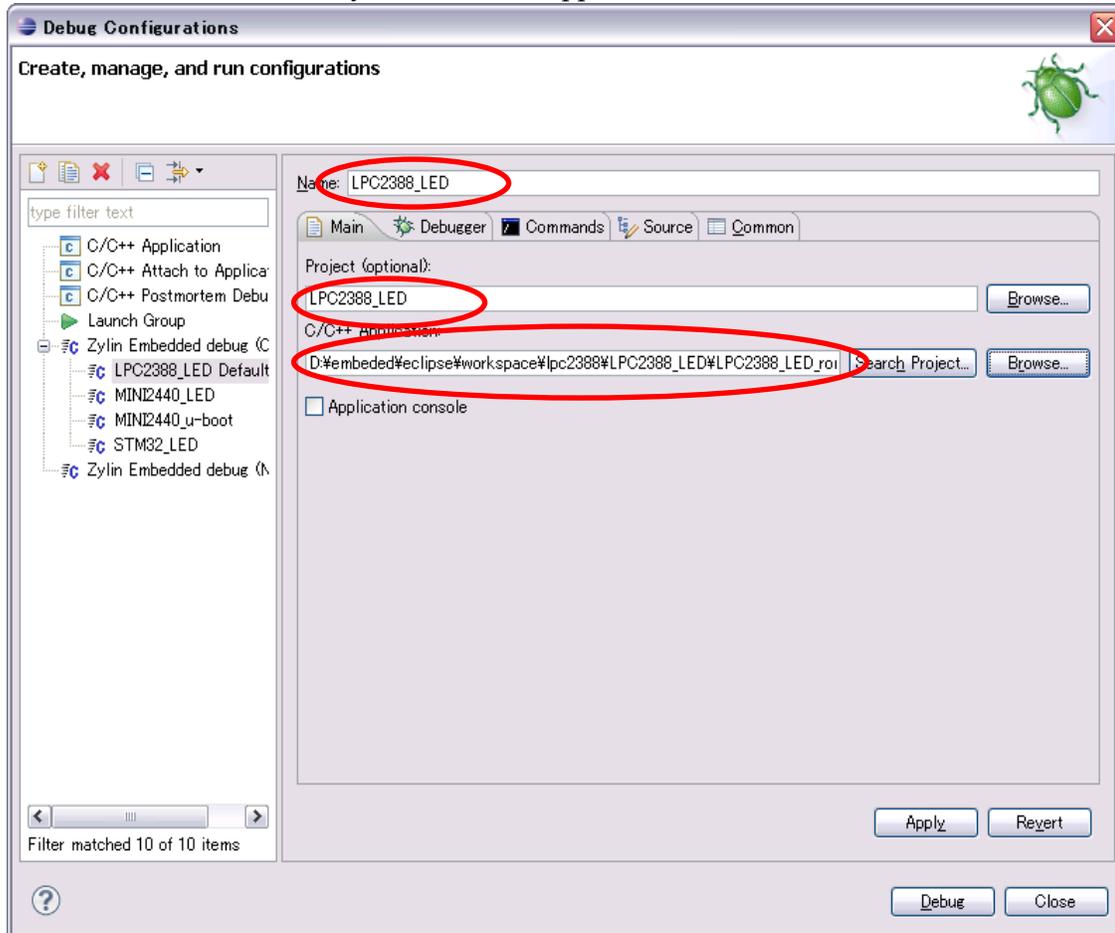
The screenshot shows the Eclipse IDE interface. The main window is titled "C/C++ - LPC2388_LED/makefile - Eclipse". The "Debug Configurations" dialog is open, displaying a list of configurations. The configuration "Zylin Embedded debug" is selected, and the "New" button is highlighted with a red circle. The Project Explorer on the left shows the project structure for "LPC2388_LED".

Configure launch settings from this dialog:

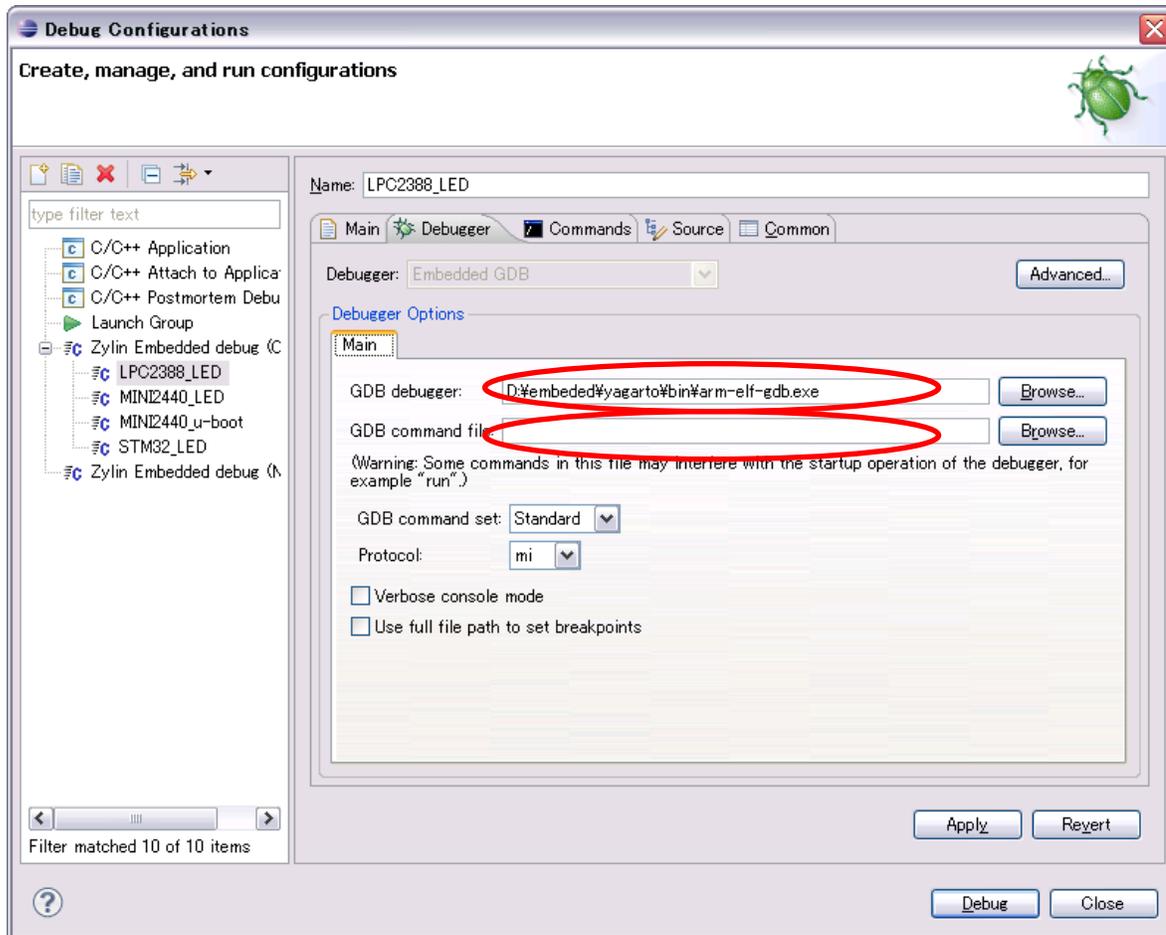
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Press the 'Filter' button to configure filtering options.
- Press the 'Filter' button to configure filtering options.
- Press the 'Filter' button to configure filtering options.

```
arm-elf-gcc ./src/crt.o ./src/main.o -mcpu=arm7tdmi -nostartfiles -T./prj/lpc2388_rom.ld -Wl,-Map=LPC2388_LED_rom.map,--cref,--no-warn-mismatch -o LPC2388_LED_rom.elf arm-elf-objcopy -O ihex LPC2388_LED_rom.elf LPC2388_LED_rom.hex
```

③ 「Name」を入力、「Project」、「C/C++ Application」(elfファイル指定)を指定



- ④ 「GDB debugger」を「arm-elf-gdb.exe」の場所に指定



⑤ 「Commands」 を入力

target remote localhost:3333 // ローカルポート「3333」と接続 (OpenOCDと接続)

monitor halt//ボードの実行を中断させる

monitor step//ステップで実行するように

load // leds_elfをロード, 「elf」というフォーマットのファイルにアドレスが含まれる

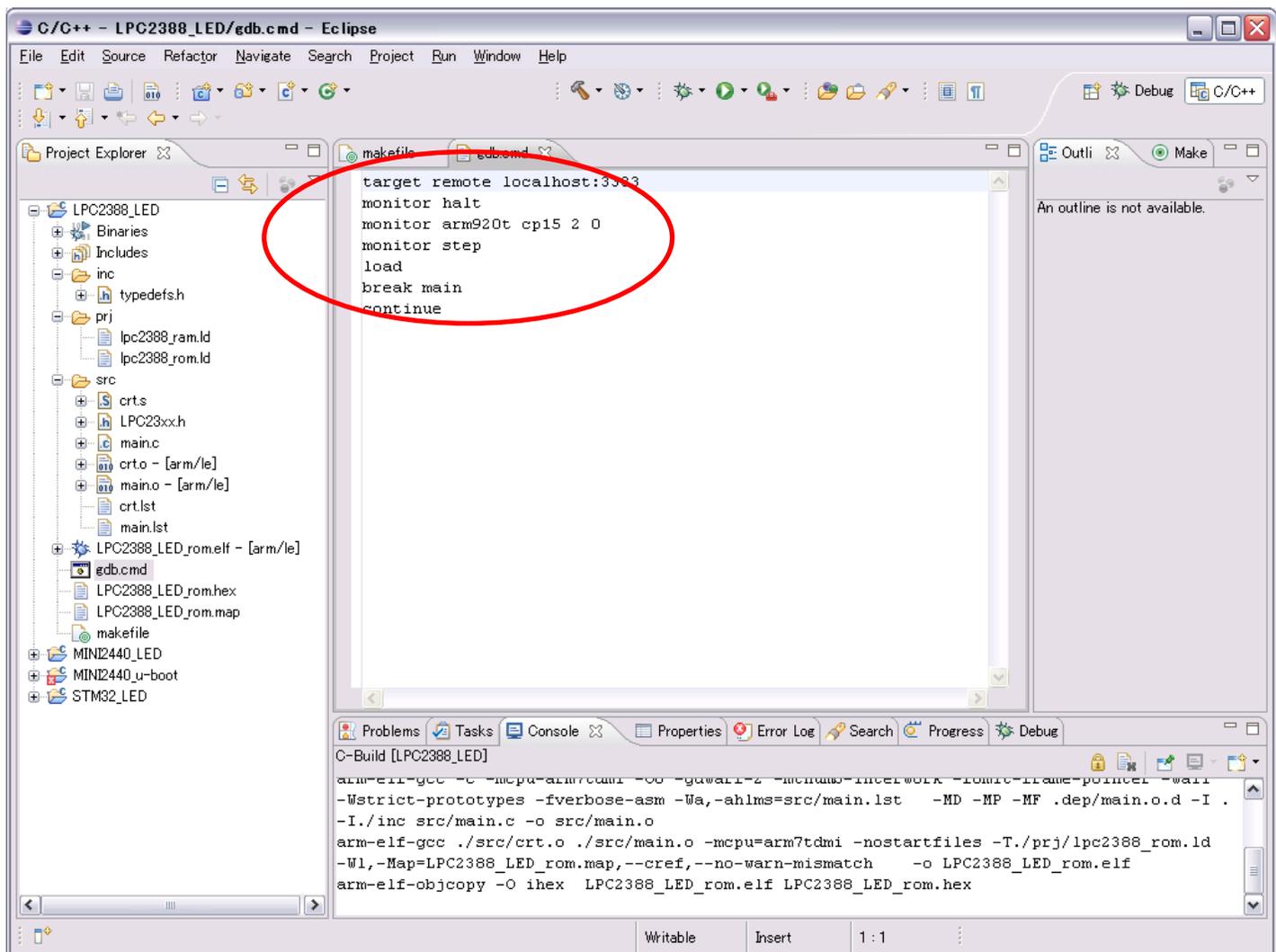
break main // 「main」関数にブレークポイントを設定

continue // プログラムを実行させて、「main」にとまってステップでデバッグ可能

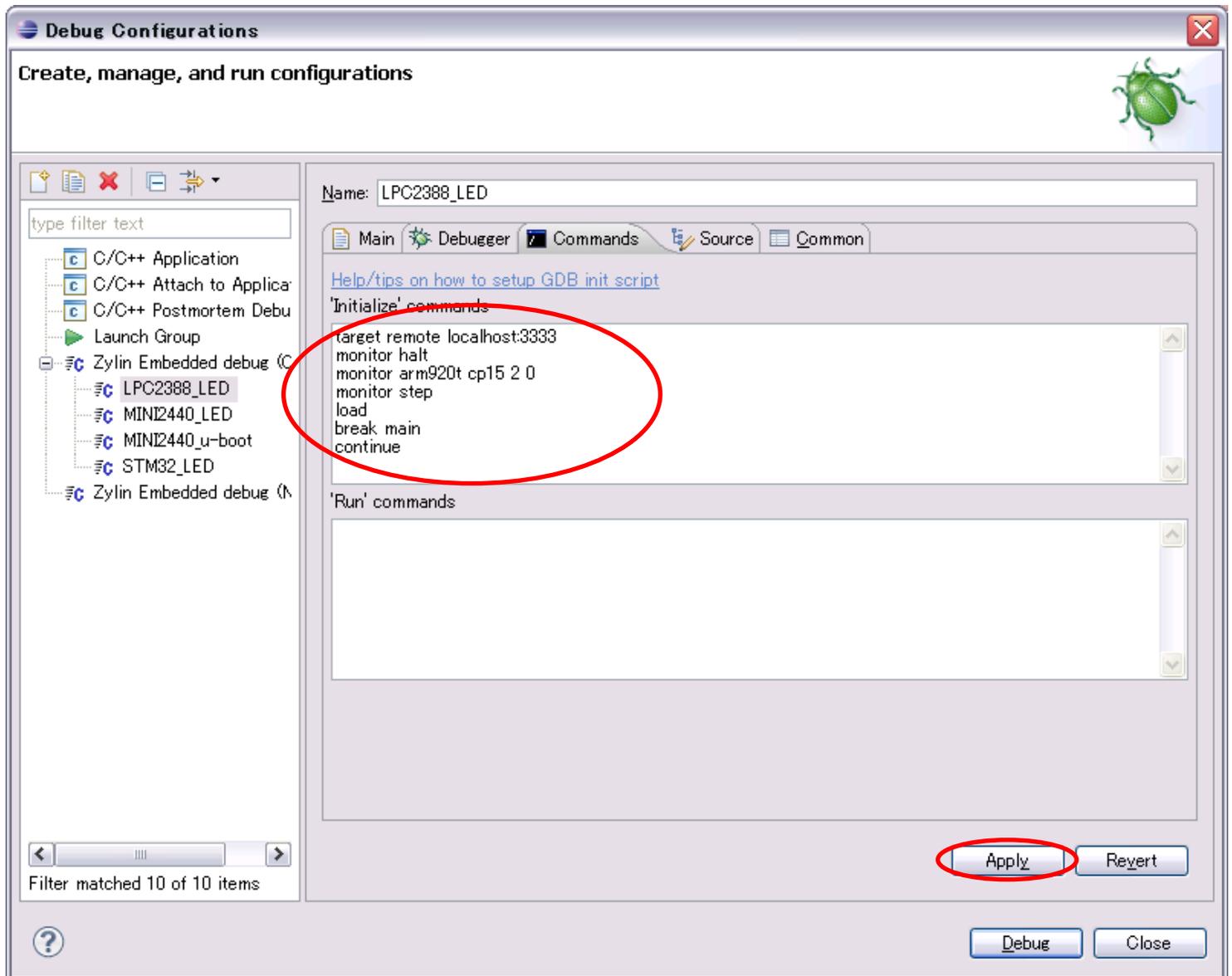
「LPC2388_LED」プロジェクトの配下、「gdb.cmd」ファイルの内容をそのままコピーしても OK

「gdb.cmd」を右クリック、「Open With」→「Text Editor」

***ほかの設定を保存するため、コピー途中で「Apply」ボタンを押して適用します。**

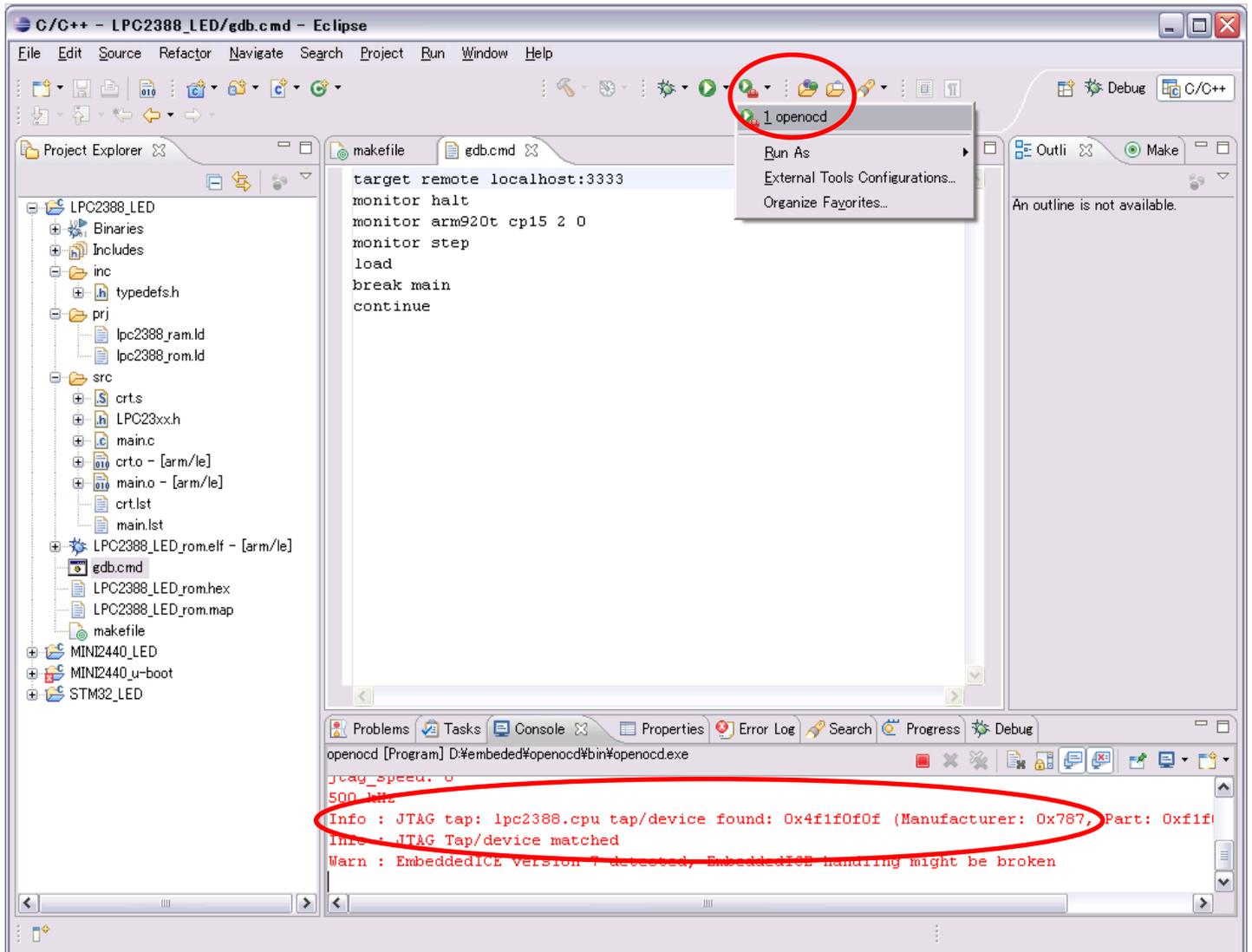


上記のコピー内容を下記「Commands」に貼り付け、「Apply」ボタンを押下



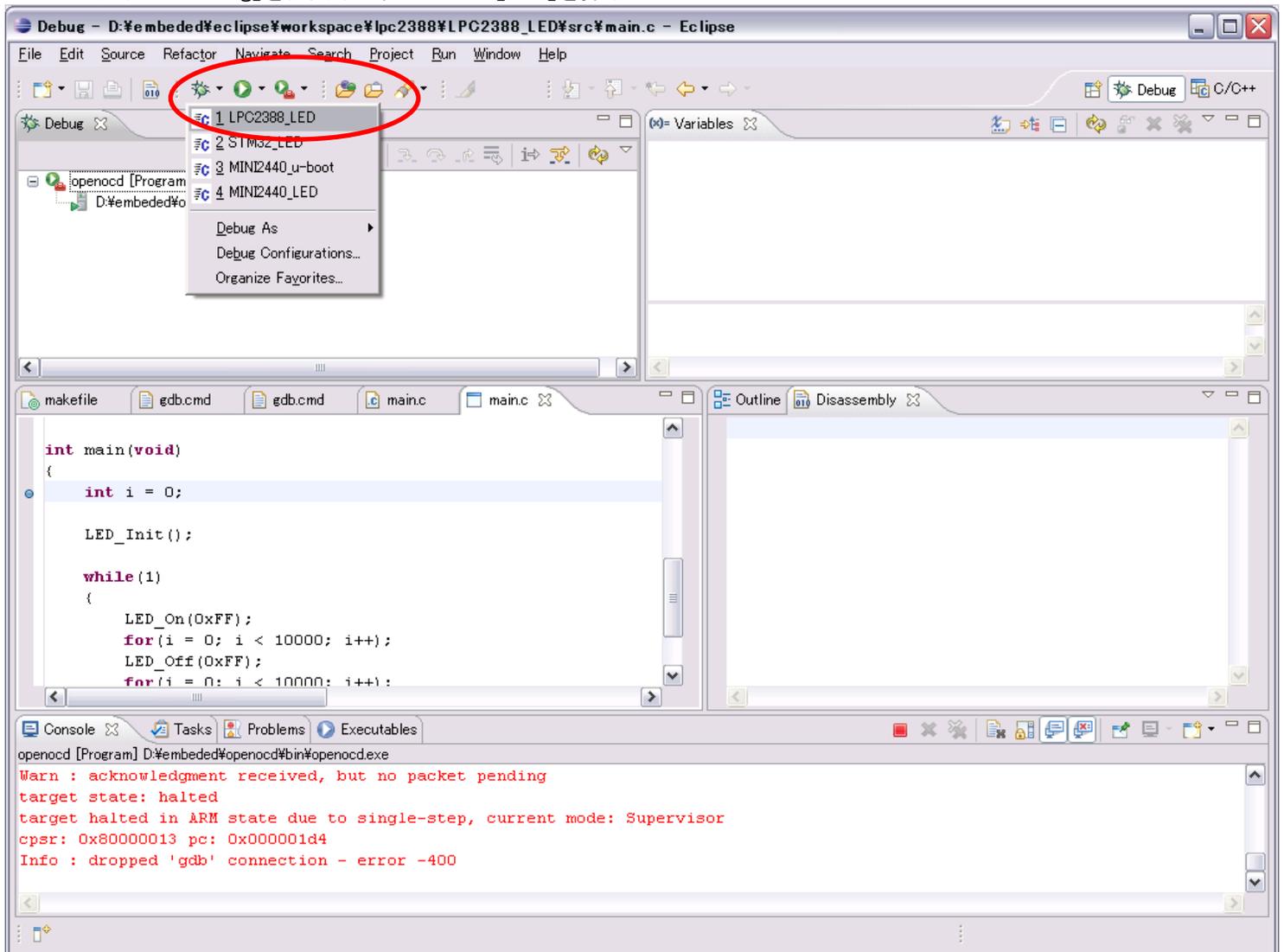
5. デバッグ

① openocd 起動

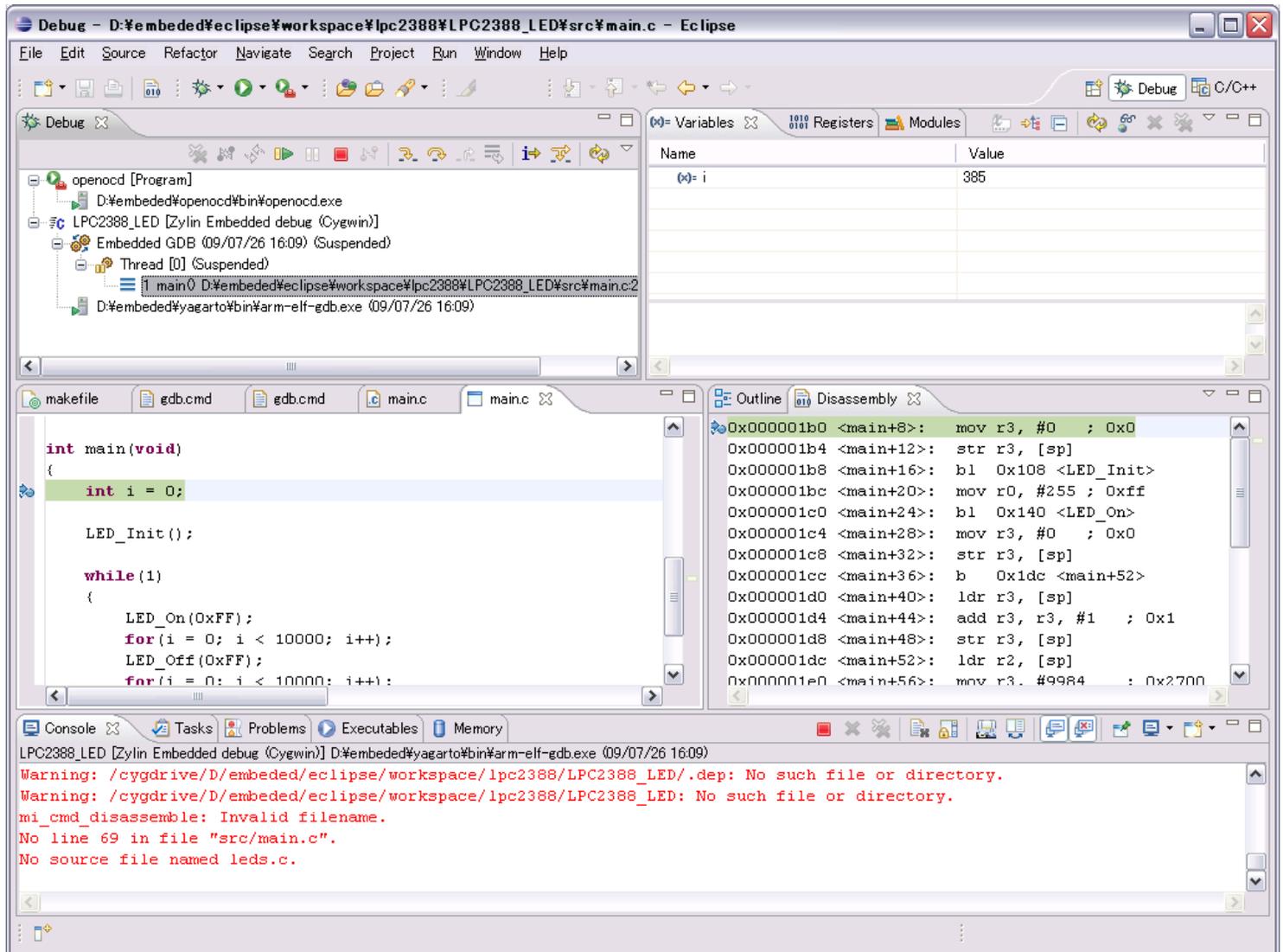


② gdb 起動

アイコン「Debug」をクリックし、「LPC2388_LED」を押下



③ 関数「main」に止まり、ショットカットキー「F6」でステップでデバッグできる



The screenshot shows the Eclipse IDE interface with the following components:

- Debugger View:** Shows the execution stack with 'main' selected. The variable 'i' is shown with a value of 385.
- Source Editor:** Displays the C code for 'main.c'. The line 'int i = 0;' is highlighted, indicating the current execution point.
- Disassembly View:** Shows the assembly code for the current function, starting with 'mov r3, #0 ; 0x0'.
- Console:** Displays the following output:

```
LPC2388_LED [Zylin Embedded debug (Cygwin)] D:\embedded\yagarto\bin\arm-elf-gdb.exe (09/07/26 16:09)
Warning: /cygdrive/D/embedded/eclipse/workspace/lpc2388/LPC2388_LED/.dep: No such file or directory.
Warning: /cygdrive/D/embedded/eclipse/workspace/lpc2388/LPC2388_LED: No such file or directory.
mi_cmd_disassemble: Invalid filename.
No line 69 in file "src/main.c".
No source file named leds.c.
```

6.3 ARM Cortex-M3 の STM32F103

6.3.1 STM32F103 ボード購入 URL

<http://www.csun.co.jp/SHOP/200903019.html>

ボード URL から STM32F103 マニュアルを参照し、Open-JTAG に接続してください。

ボード URL から LPC2388 マニュアルを参照し、Open-JTAG に接続してください。

STM32F103 ボードマニュアルダウンロード URL :

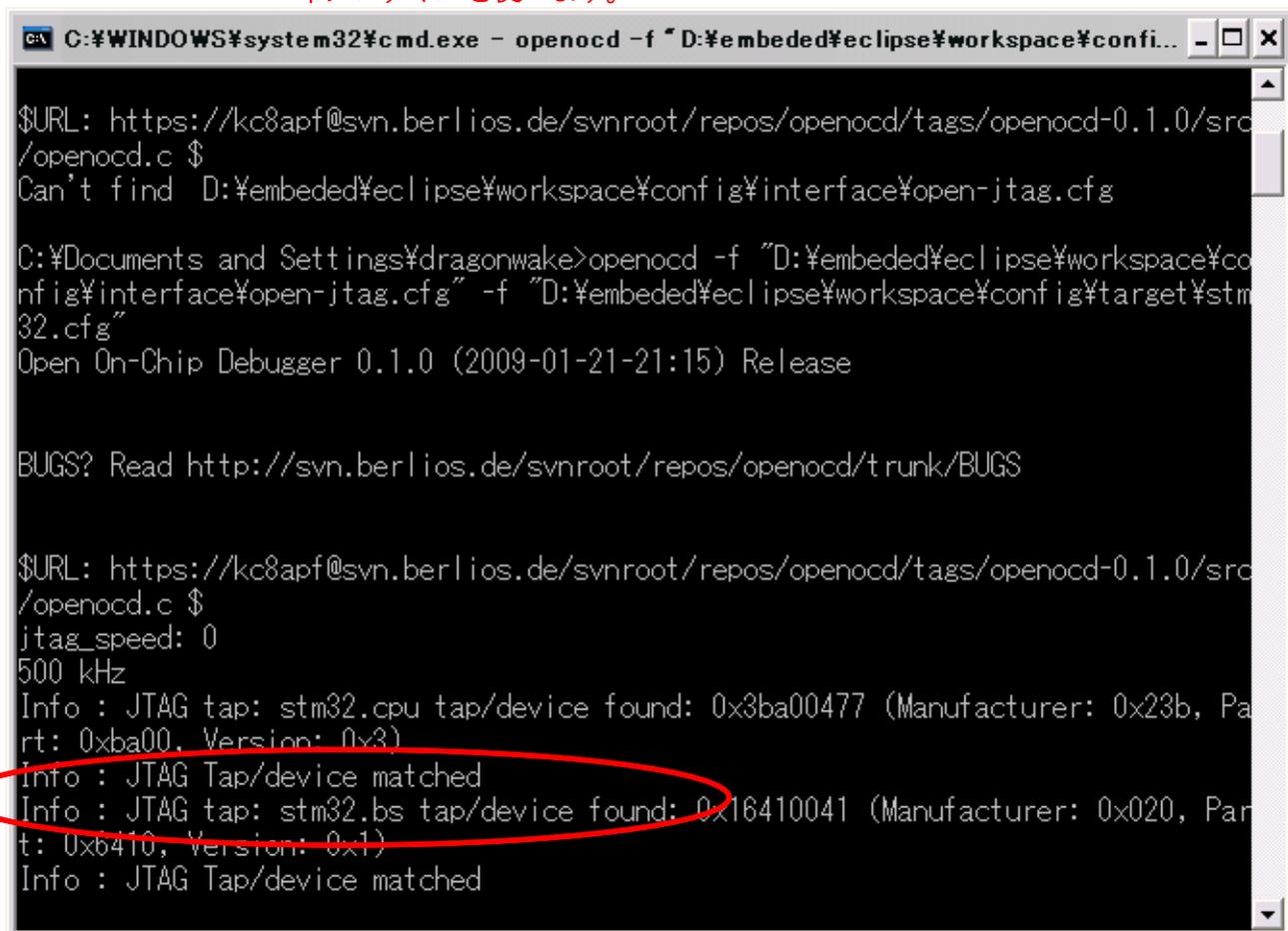
http://www.dragonwake.com/download/download-stm32/STM32_manual.pdf

6.3.2 ハードウェア動作確認

- (1). OpenJTAG をパソコンの USB ポートに挿入する
- (2). JTAG ケーブルで OpenJTAG と STM32 ボードを繋ぐ
- (3). STM32 ボードに電源を入れる
- (4). 下記のコマンドを入力します。

```
openocd -f "D:\embedded\eclipse\workspace\config\interface\open-jtag.cfg" -f "D:\embedded\eclipse\workspace\config\target\stm32.cfg"
```

※ はじめの-fは OpenJTAG のコンフィグファイルを使います。二番目の-fは stm32 のコンフィグファイルを使います。



```
C:\WINDOWS\system32\cmd.exe - openocd -f "D:\embedded\eclipse\workspace\confi...  
$URL: https://kc8apf@svn.berlios.de/svnroot/repos/openocd/tags/openocd-0.1.0/src  
/openocd.c $  
Can't find D:\embedded\eclipse\workspace\config\interface\open-jtag.cfg  
  
C:\Documents and Settings\dragonwake>openocd -f "D:\embedded\eclipse\workspace\co  
nfig\interface\open-jtag.cfg" -f "D:\embedded\eclipse\workspace\config\target\stm  
32.cfg"  
Open On-Chip Debugger 0.1.0 (2009-01-21-21:15) Release  
  
BUGS? Read http://svn.berlios.de/svnroot/repos/openocd/trunk/BUGS  
  
$URL: https://kc8apf@svn.berlios.de/svnroot/repos/openocd/tags/openocd-0.1.0/src  
/openocd.c $  
jtag_speed: 0  
500 kHz  
Info : JTAG tap: stm32.cpu tap/device found: 0x3ba00477 (Manufacturer: 0x23b, Pa  
rt: 0xba00, Version: 0x3)  
Info : JTAG Tap/device matched  
Info : JTAG tap: stm32.bs tap/device found: 0x16410041 (Manufacturer: 0x020, Par  
t: 0xb410, Version: 0x1)  
Info : JTAG Tap/device matched
```

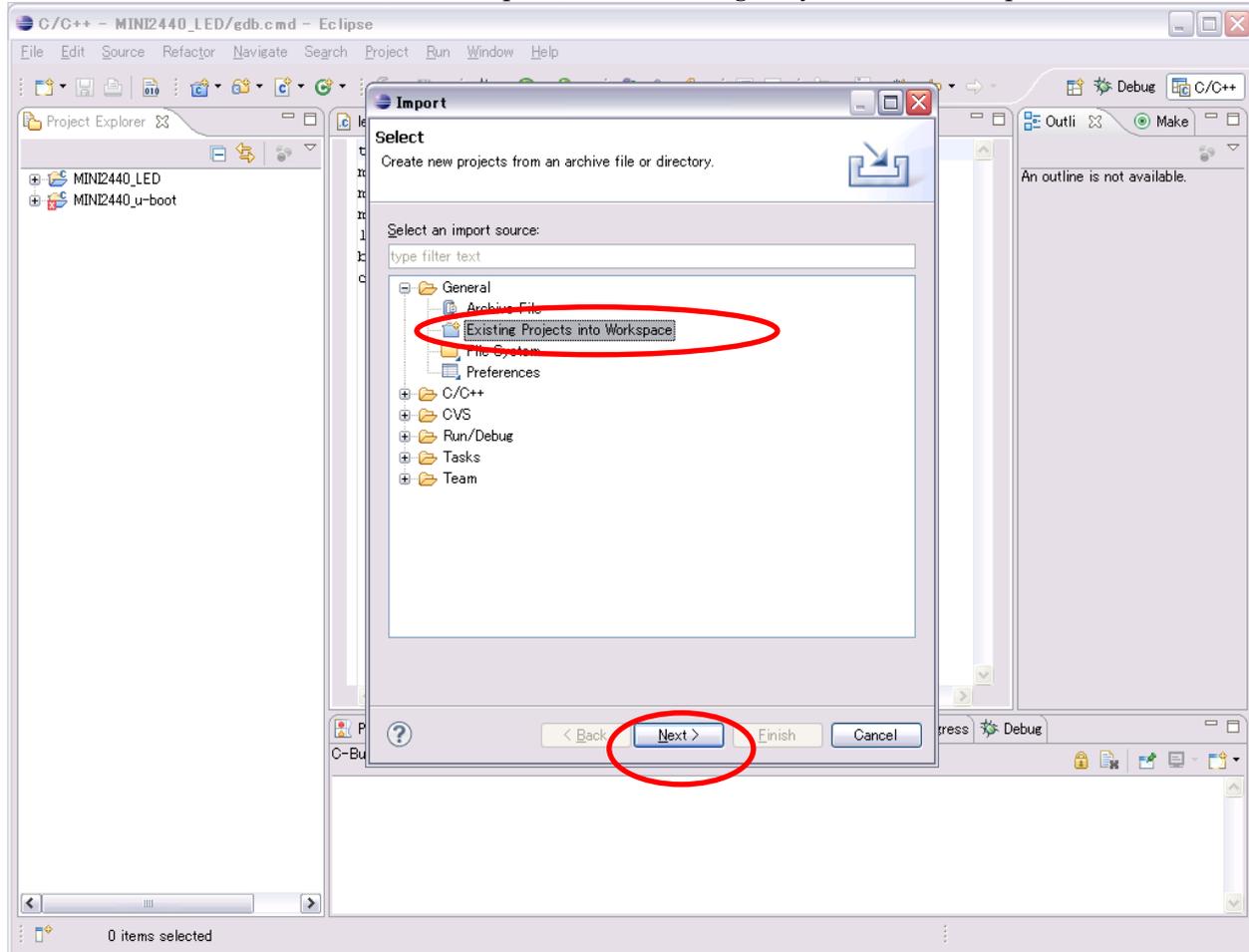
6.3.3 LED サンプルデバッグ

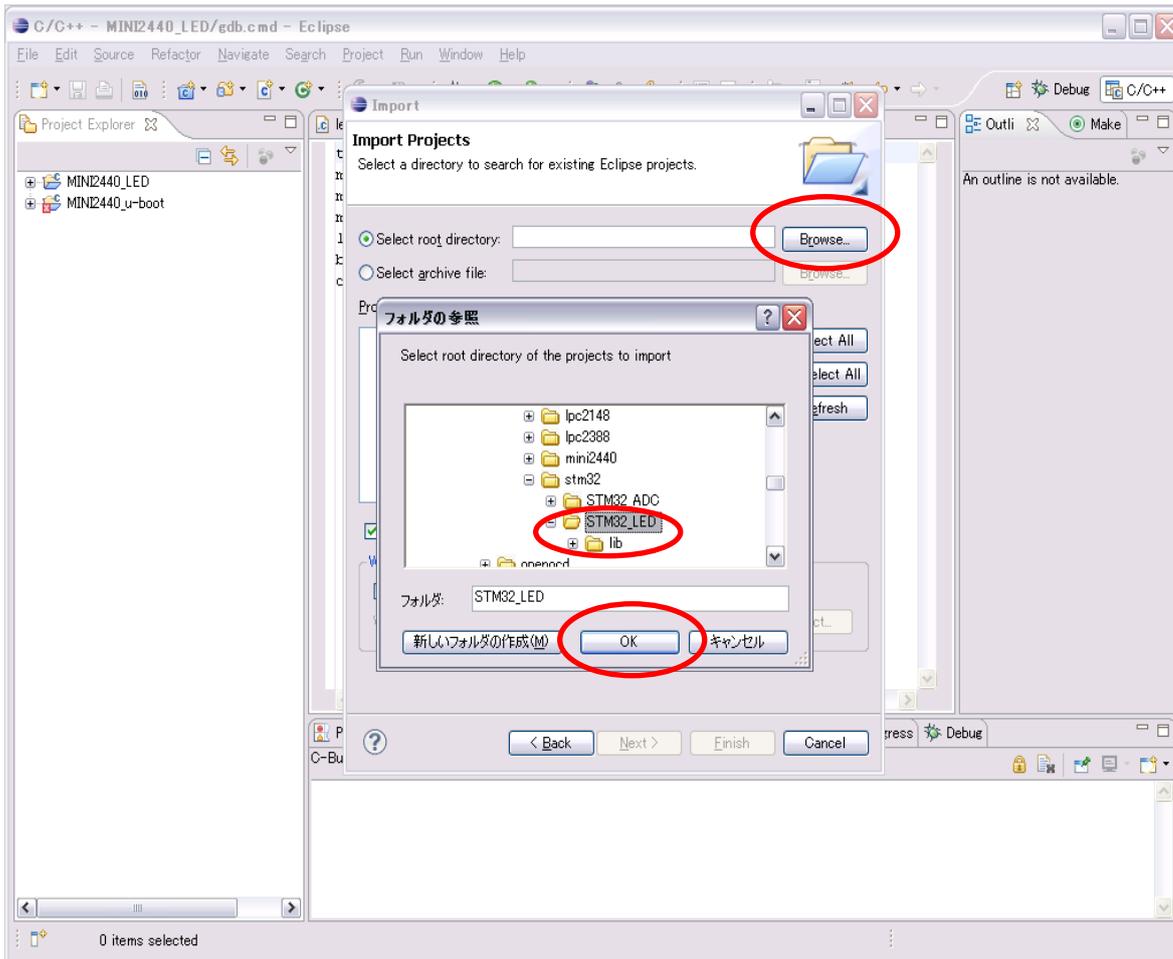
*サンプルの導入について、二つ方法があります、一つは新規プロジェクトを作成してからソースを導入、もう一つはダウンロードプロジェクトをインポートします。

*新規プロジェクトの作成方法は「6.1.3 LPC2148 用のサンプル「LED」をデバッグ」を参照

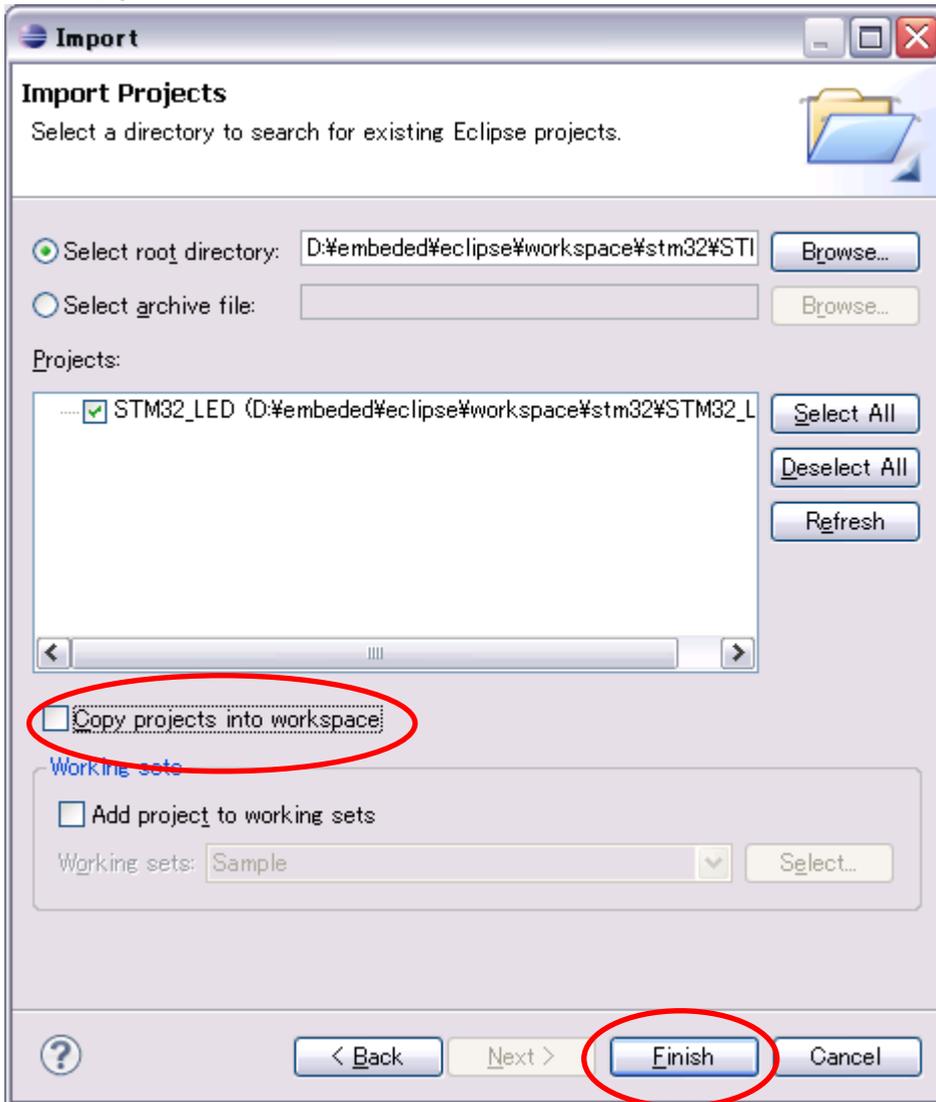
*ここに既存プロジェクトのインポートとして説明

1. プロジェクト導入 : 「File」 → 「Import」 → 「Existing Projects into workspace」



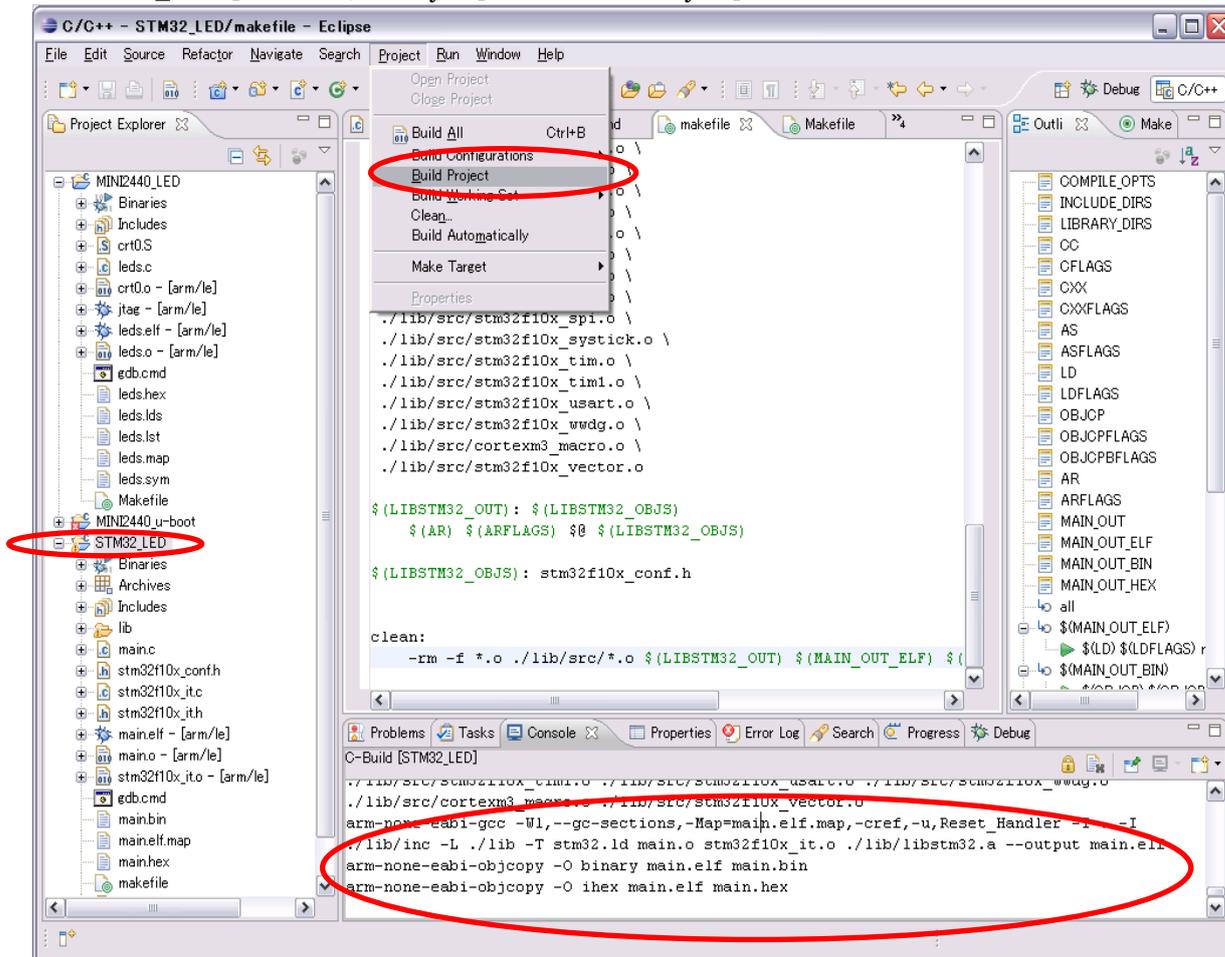


既にダウンロードしたプロジェクトを Wokspace に入れていますので、ここに「Copy projects into workspace」を外します。



2. 導入済みのプロジェクトをコンパイル

「STM32_LED」を選択、「Project」→「Build Project」を押下



3. openocd 設定の修正

一般の設定は「5.4 OpenOCD の設定」を参照

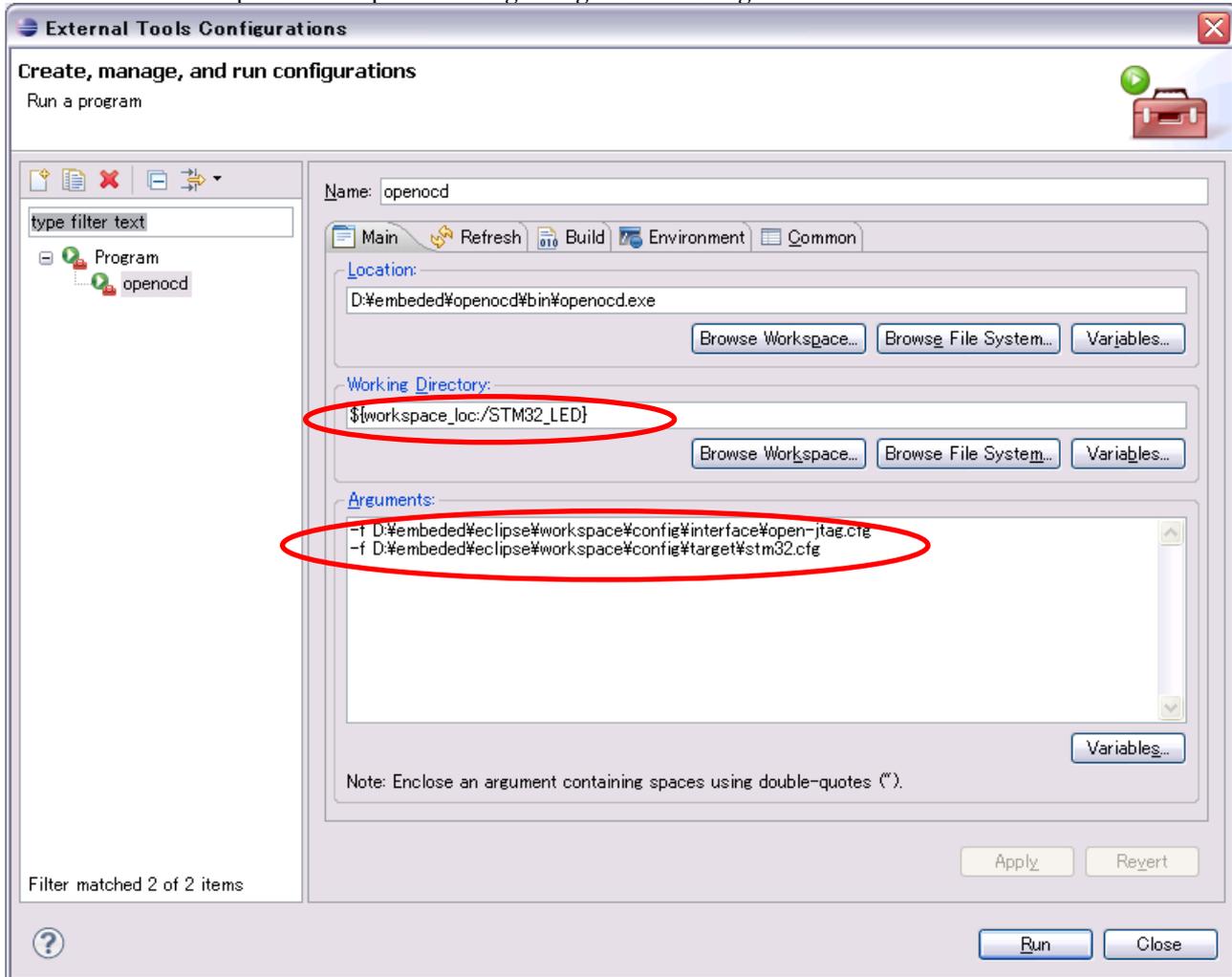
a) 「Working Directory」を「STM32_LED」に変更

b) 「Arguments」を下記のように変更

-f "D:¥embedded¥eclipse¥workspace¥config¥interface¥open-jtag.cfg"

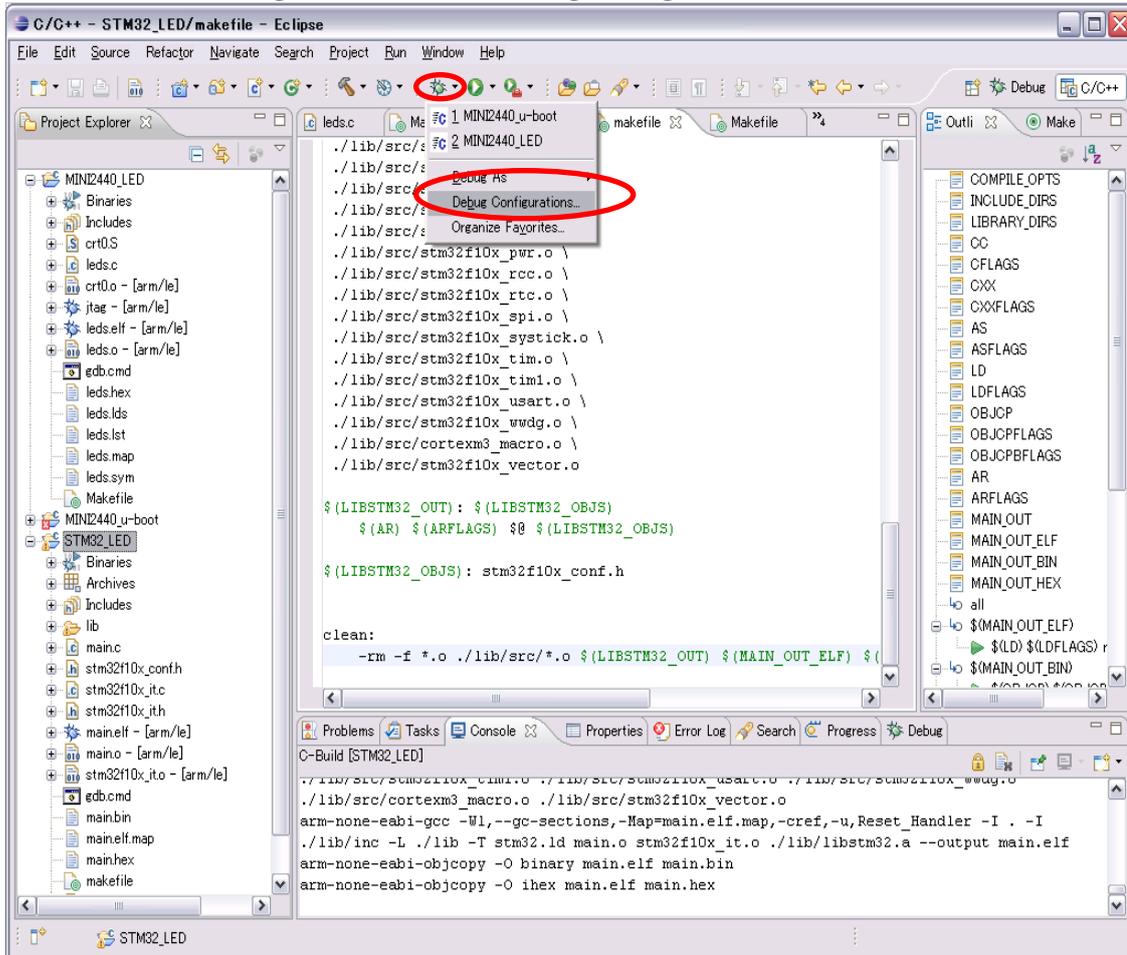
-f

"D:¥embedded¥eclipse¥workspace¥config¥target¥stm32.cfg"

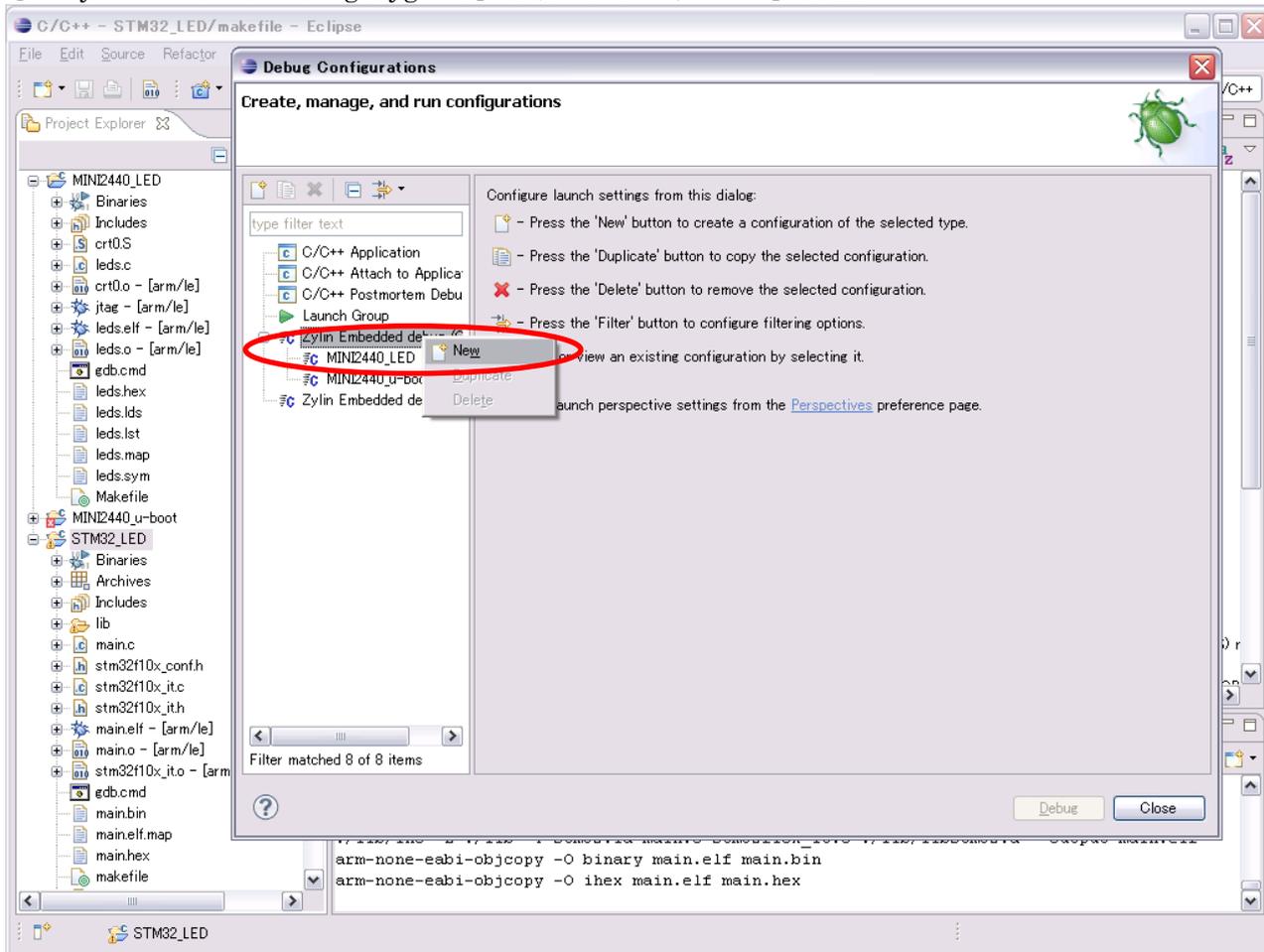


4. STM32_LED用のGDBを追加

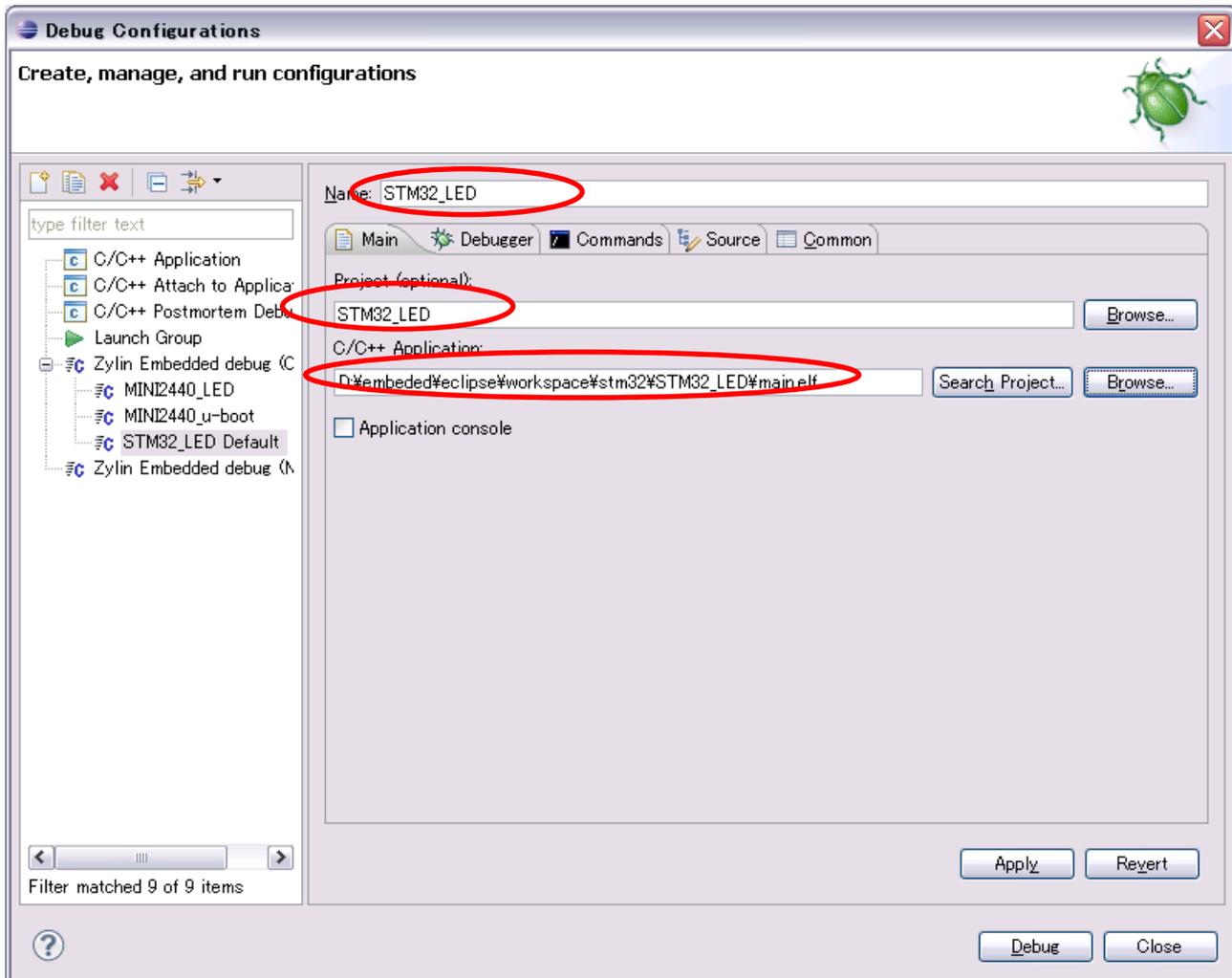
①アイコン「Debug」をクリック、「Debug Configuration」を押下



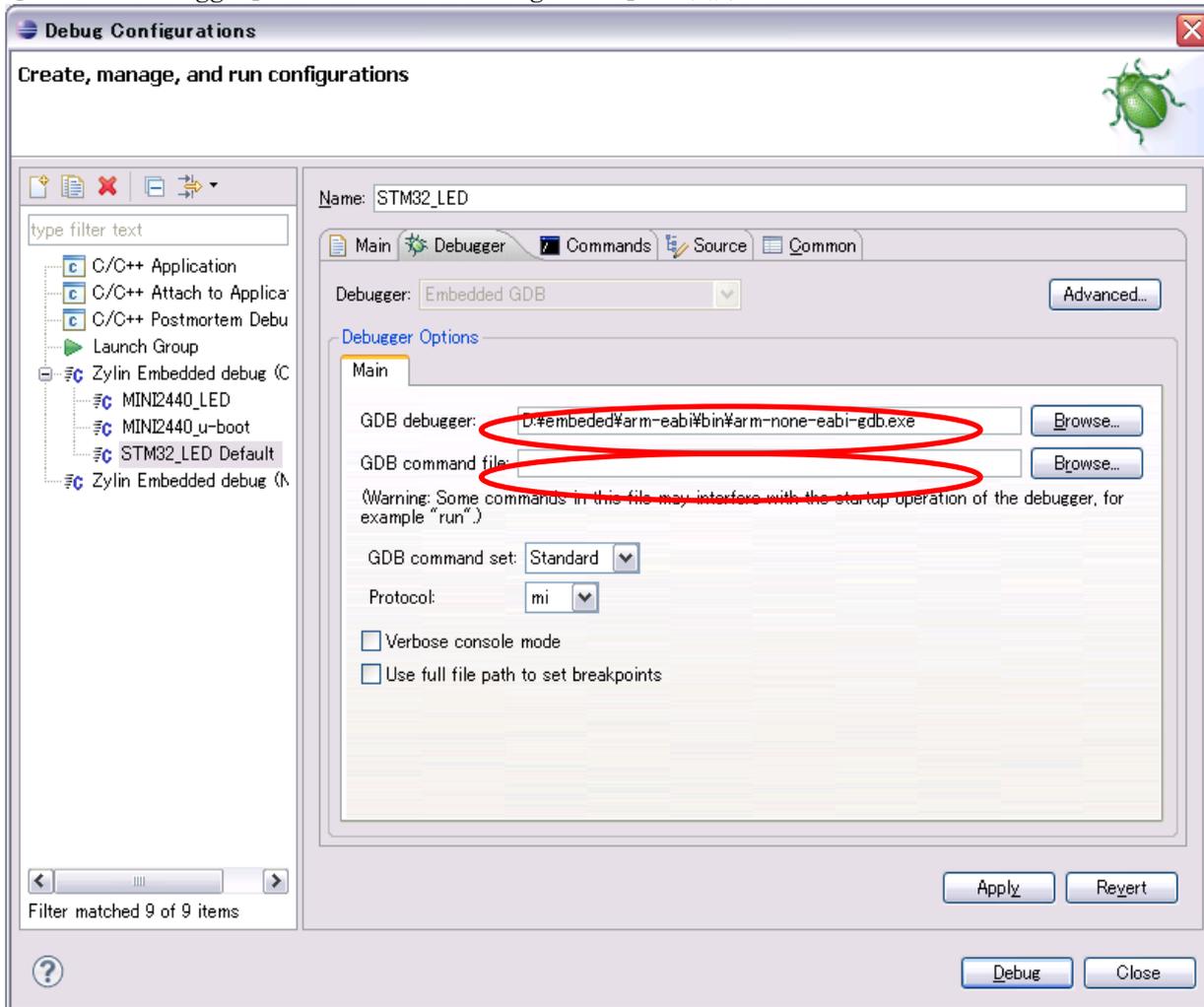
② 「Zylin Embedded debug(Cygwin)」を右クリック、「New」を選択



③ 「Name」を入力、「Project」、「C/C++ Application」を指定



④ 「GDB debugger」を「arm-none-eabi-gdb.exe」の場所に指定



⑤ 「Commands」を入力

target remote localhost:3333 // ローカルポート「3333」と接続 (OpenOCDと接続)

monitor halt//ボードの実行を中断させる

monitor step//ステップで実行するように

load // leds_elfをロード, 「elf」というフォーマットのファイルにアドレスが含まれる

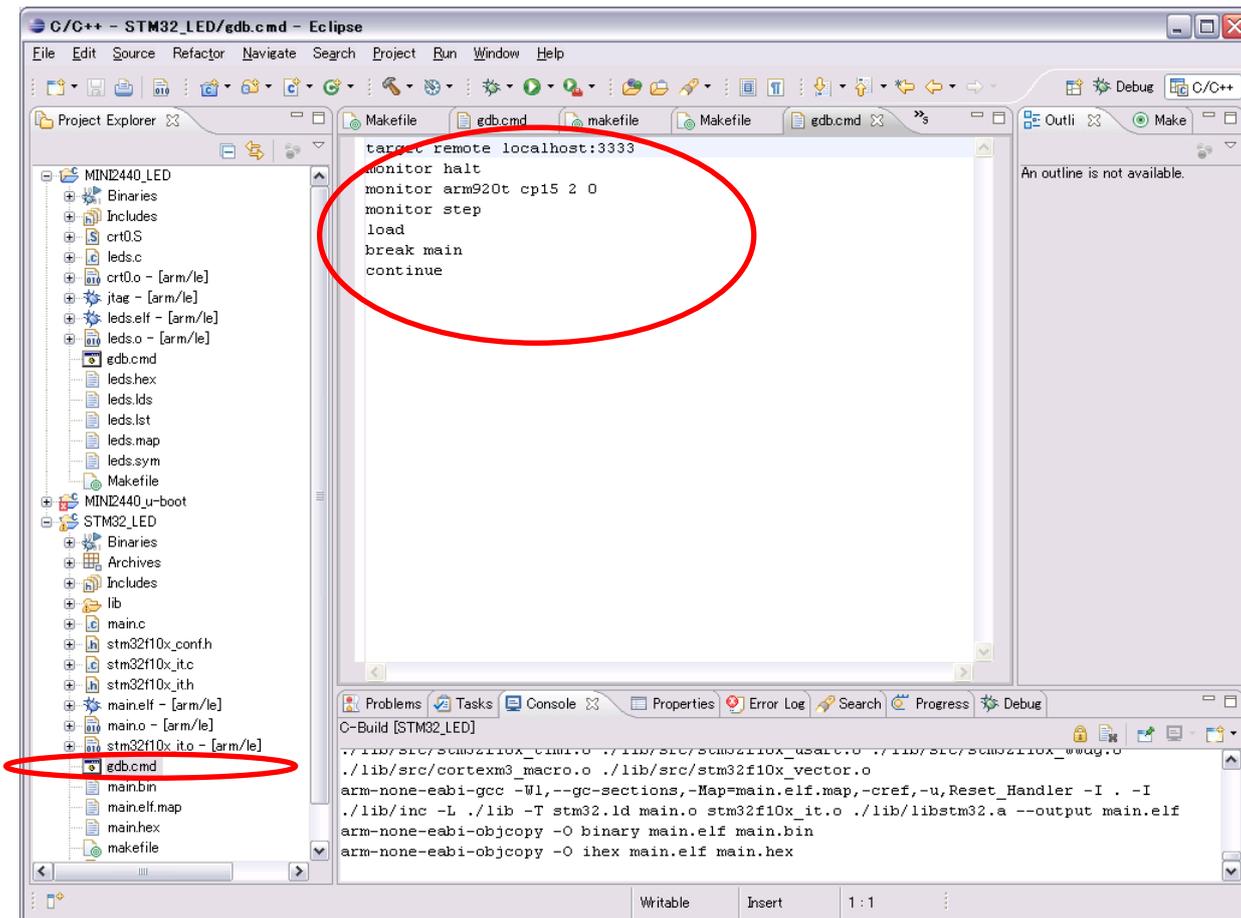
break main // 「main」関数にブレークポイントを設定

continue // プログラムを実行させて、「main」にとまってステップでデバッグ可能

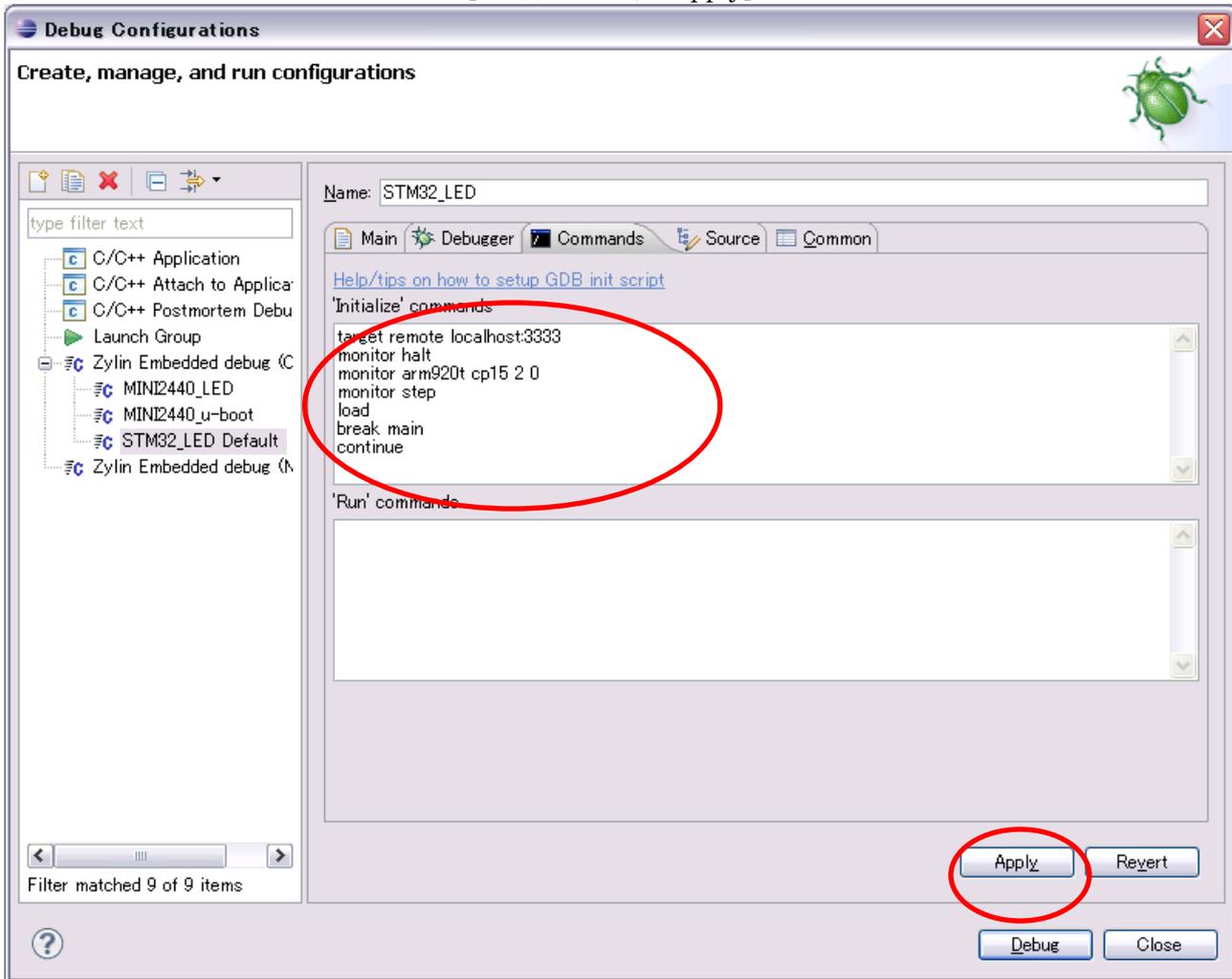
「STM32_LED」プロジェクトの配下、「gdb.cmd」ファイルの内容をそのままコピーしても OK

「gdb.cmd」を右クリック、「Open With」→「Text Editor」

*ほかの設定を保存するため、コピー途中で「Apply」ボタンを押して適用します。

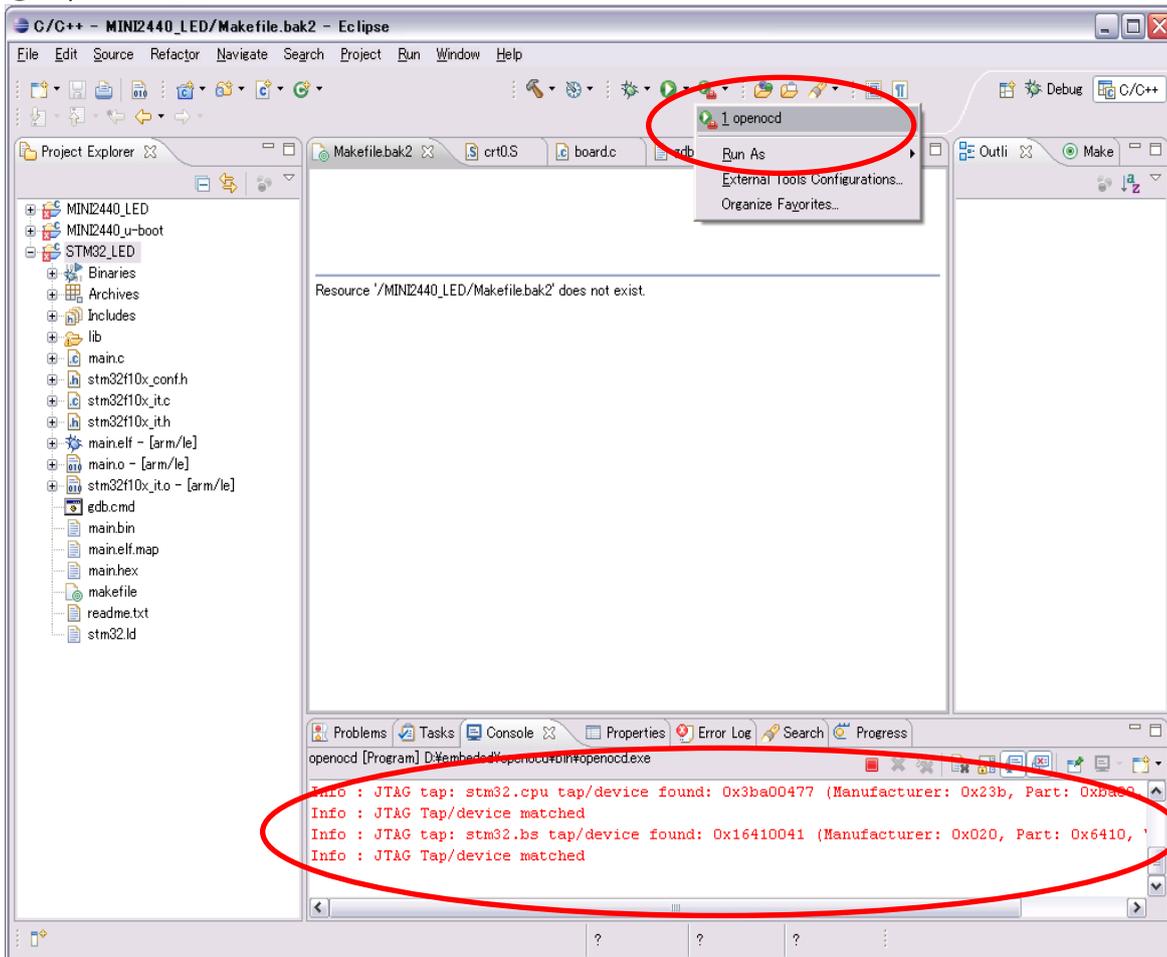


上記のコピー内容を下記「Commands」に貼り付け、「Apply」ボタンを押下

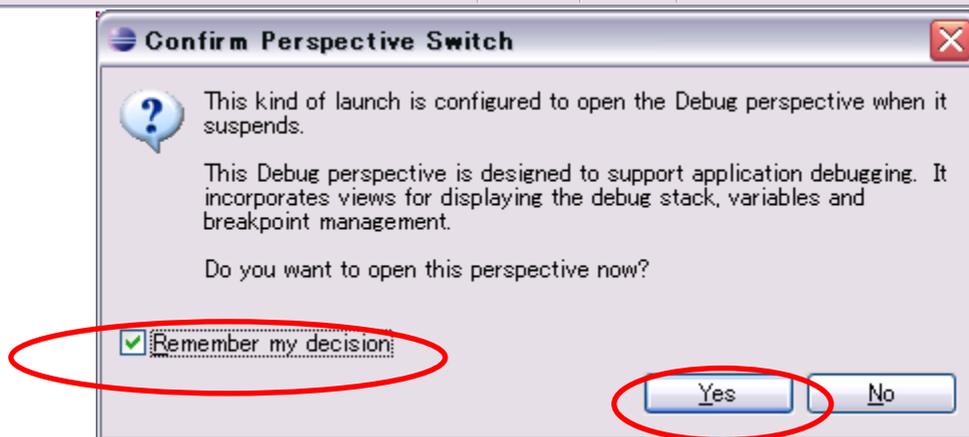
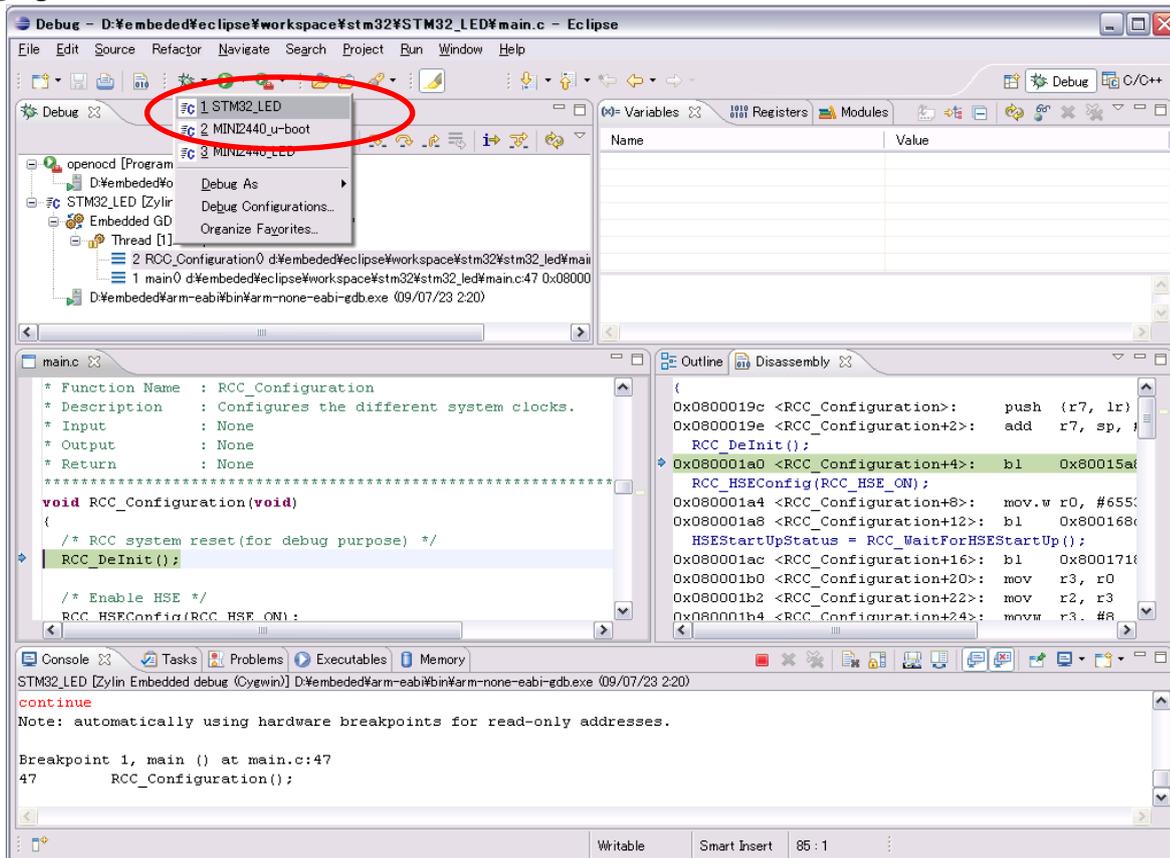


5. デバッグ

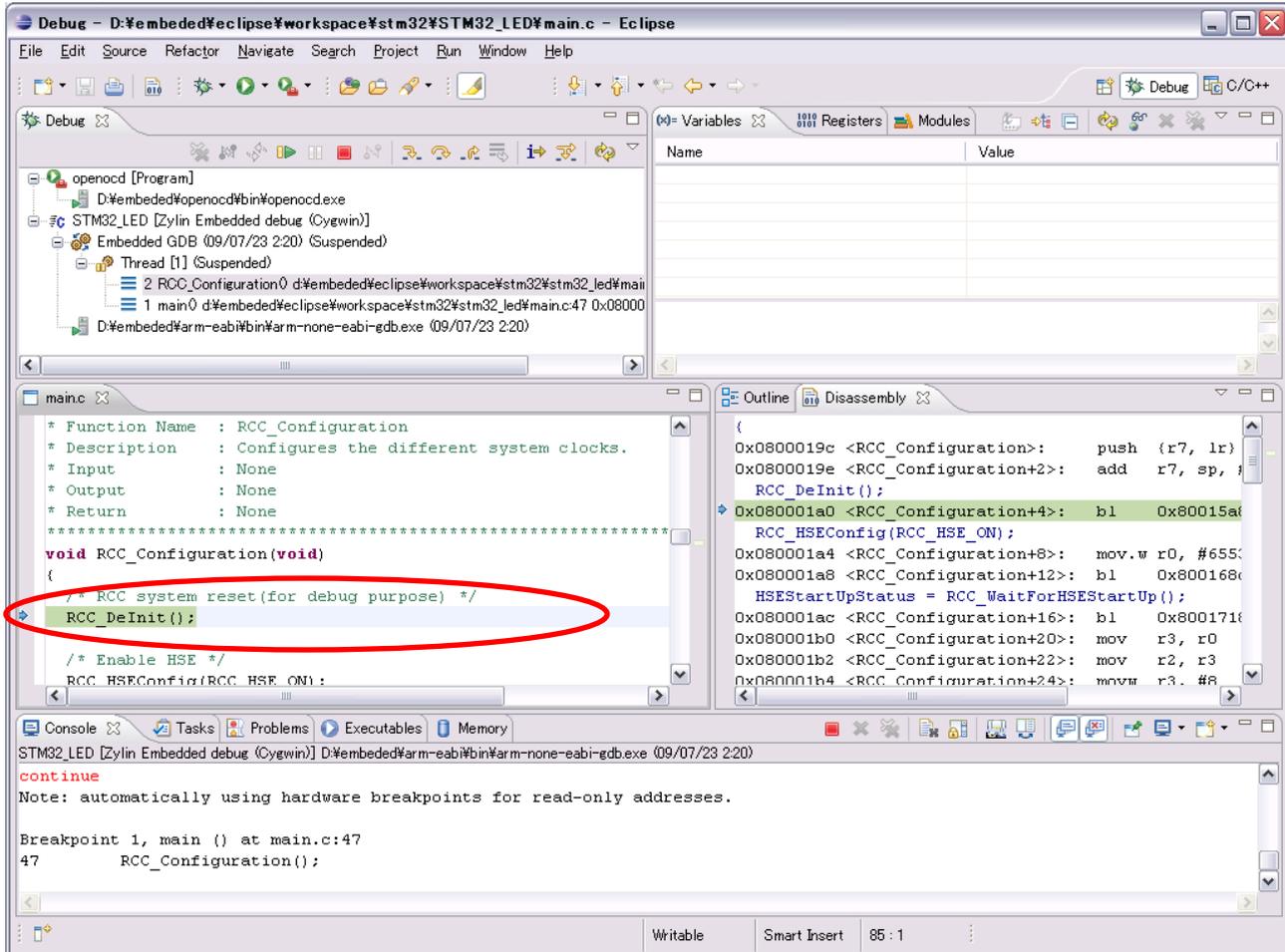
① openocd 起動



② gdb 起動



③ 関数「main」に止まり、ショットカットキー「F6」でステップでデバッグできる



The screenshot shows the Eclipse IDE interface during a debug session. The main window displays the source code for the `RCC_Configuration` function, with the `RCC_DeInit();` line highlighted in blue and circled in red. The console window at the bottom shows the program output, including the `continue` command and a breakpoint hit at `main.c:47`.

```
* Function Name : RCC_Configuration
* Description : Configures the different system clocks.
* Input : None
* Output : None
* Return : None
*****
void RCC_Configuration(void)
{
    /* RCC system reset(for debug purpose) */
    RCC_DeInit();
    /* Enable HSE */
    RCC_HSEConfig(RCC_HSE_ON);

```

```
{
0x0800019c <RCC_Configuration>: push {r7, lr}
0x0800019e <RCC_Configuration+2>: add r7, sp, #4
RCC_DeInit();
0x080001a0 <RCC_Configuration+4>: b1 0x80015a0
RCC_HSEConfig(RCC_HSE_ON);
0x080001a4 <RCC_Configuration+8>: mov.w r0, #65535
0x080001a8 <RCC_Configuration+12>: b1 0x8001680
HSEStartUpStatus = RCC_WaitForHSEStartUp();
0x080001ac <RCC_Configuration+16>: b1 0x8001710
0x080001b0 <RCC_Configuration+20>: mov r3, r0
0x080001b2 <RCC_Configuration+22>: mov r2, r3
0x080001b4 <RCC_Configuration+24>: movw r3, #8

```

```
STM32_LED [Zylin Embedded debug (Cygwin)] D:\embedded\arm-eabi\bin\arm-none-eabi-gdb.exe (09/07/23 2:20)
continue
Note: automatically using hardware breakpoints for read-only addresses.

Breakpoint 1, main () at main.c:47
47      RCC_Configuration();

```

④ ステップでデバッグ

*関数「main」のコーディング「GPIO_SetBits(GPIOB, GPIO_Pin_5);」を実行前、
要は、LED 点灯前に、STM32 ボードの LED が消している



ショットカットキー「F6」を押しながら、関数「main」のコーディング「GPIO_SetBits(GPIOB, GPIO_Pin_5);」
を実行後、STM32 ボードの LED が点灯している



6.4 ARM9 の MINI2440

6.4.1 MINI2440 を購入

MINI2440 ボード購入 URL : (LCD3.5 とのセット)

<http://www.csun.co.jp/SHOP/200812021.html>

LPC2148 マニュアルダウンロード URL :

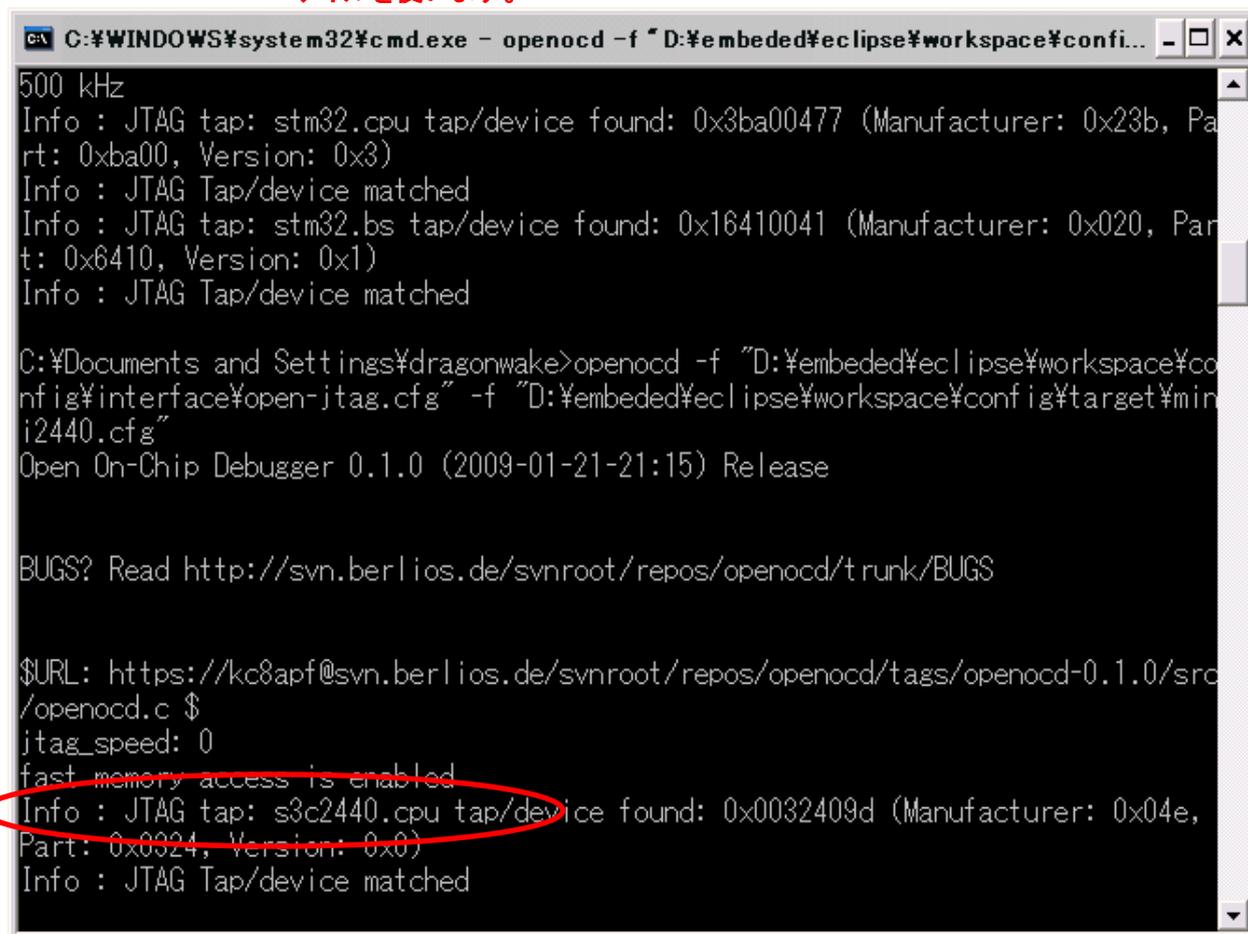
<http://www.dragonwake.com/download/arm9-download/linux-2.6.29/MINI2440-linux-2.6.29.pdf>

6.4.2 ハードウェア動作確認

1. OpenJTAG をパソコンの USB ポートに挿入する
2. JTAG ケーブルで OpenJTAG と MINI2440 ボードを繋ぐ
3. MINI2440 ボードに電源を入れる
4. 下記のコマンドを入力します。

```
openocd -f "D:\embedded\eclipse\workspace\config\interface\open-jtag.cfg" -f "D:\embedded\eclipse\workspace\config\target\mini2440.cfg"
```

※ はじめの-f は OpenJTAG のコンフィグファイルを使います。二番目の-f は mini2440 のコンフィグファイルを使います。



```
C:\WINDOWS\system32\cmd.exe - openocd -f "D:\embedded\eclipse\workspace\config\interface\open-jtag.cfg" -f "D:\embedded\eclipse\workspace\config\target\mini2440.cfg"
500 kHz
Info : JTAG tap: stm32.cpu tap/device found: 0x3ba00477 (Manufacturer: 0x23b, Part: 0xba00, Version: 0x3)
Info : JTAG Tap/device matched
Info : JTAG tap: stm32.bs tap/device found: 0x16410041 (Manufacturer: 0x020, Part: 0x6410, Version: 0x1)
Info : JTAG Tap/device matched

C:\Documents and Settings\dragonwake>openocd -f "D:\embedded\eclipse\workspace\config\interface\open-jtag.cfg" -f "D:\embedded\eclipse\workspace\config\target\mini2440.cfg"
Open On-Chip Debugger 0.1.0 (2009-01-21-21:15) Release

BUGS? Read http://svn.berlios.de/svnroot/repos/openocd/trunk/BUGS

$URL: https://kc8apf@svn.berlios.de/svnroot/repos/openocd/tags/openocd-0.1.0/src/openocd.c $
jtag_speed: 0
fast memory access is enabled
Info : JTAG tap: s3c2440.cpu tap/device found: 0x0032409d (Manufacturer: 0x04e, Part: 0x0024, Version: 0x0)
Info : JTAG Tap/device matched
```

6.4.3 LED サンプルデバッグ

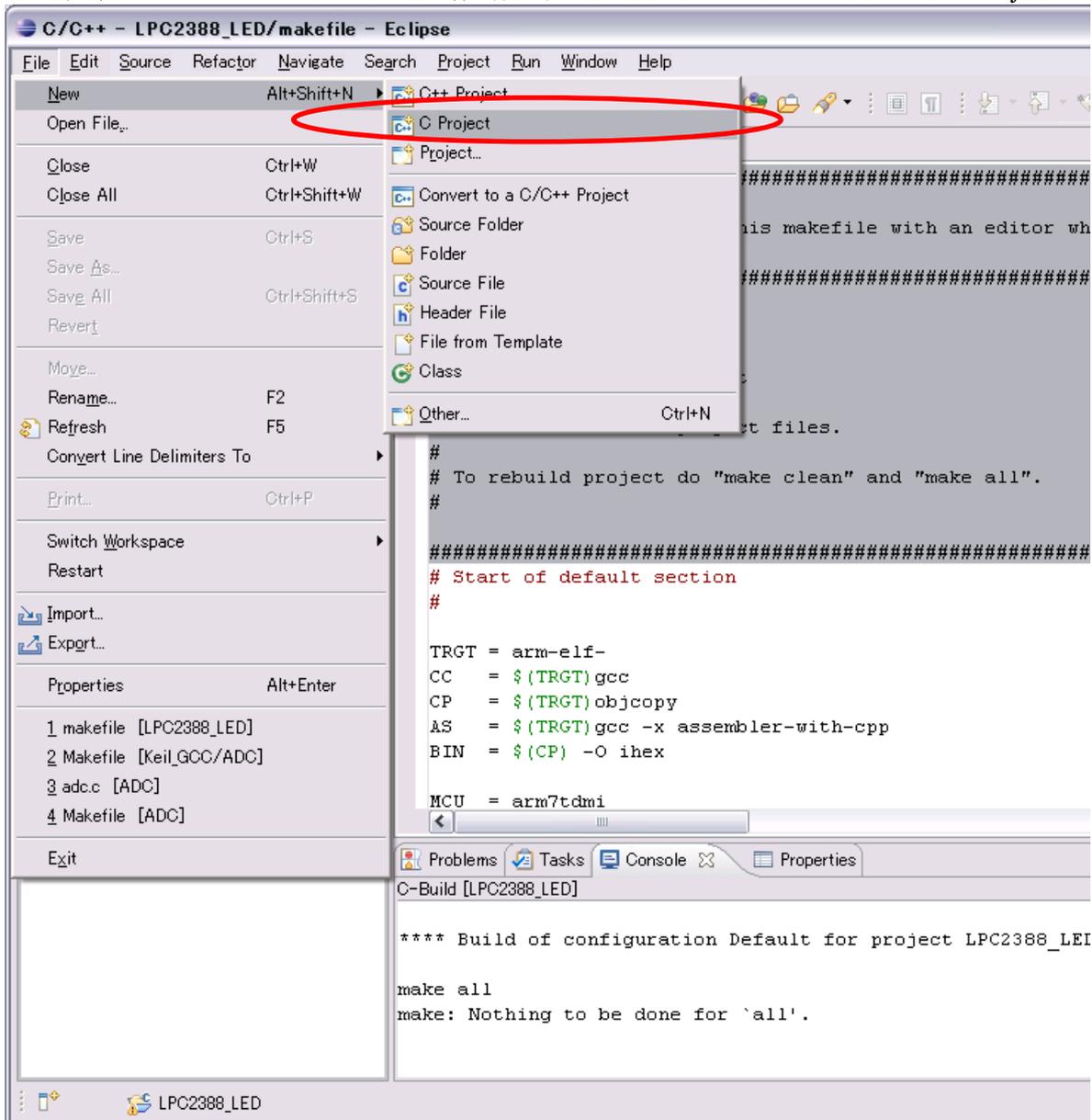
1. サンプルの導入とコンパイル :

サンプルの導入について、二つ方法があります、一つは新規プロジェクトを作成してからソースを導入、もう一つはダウンロードプロジェクトをインポートします。

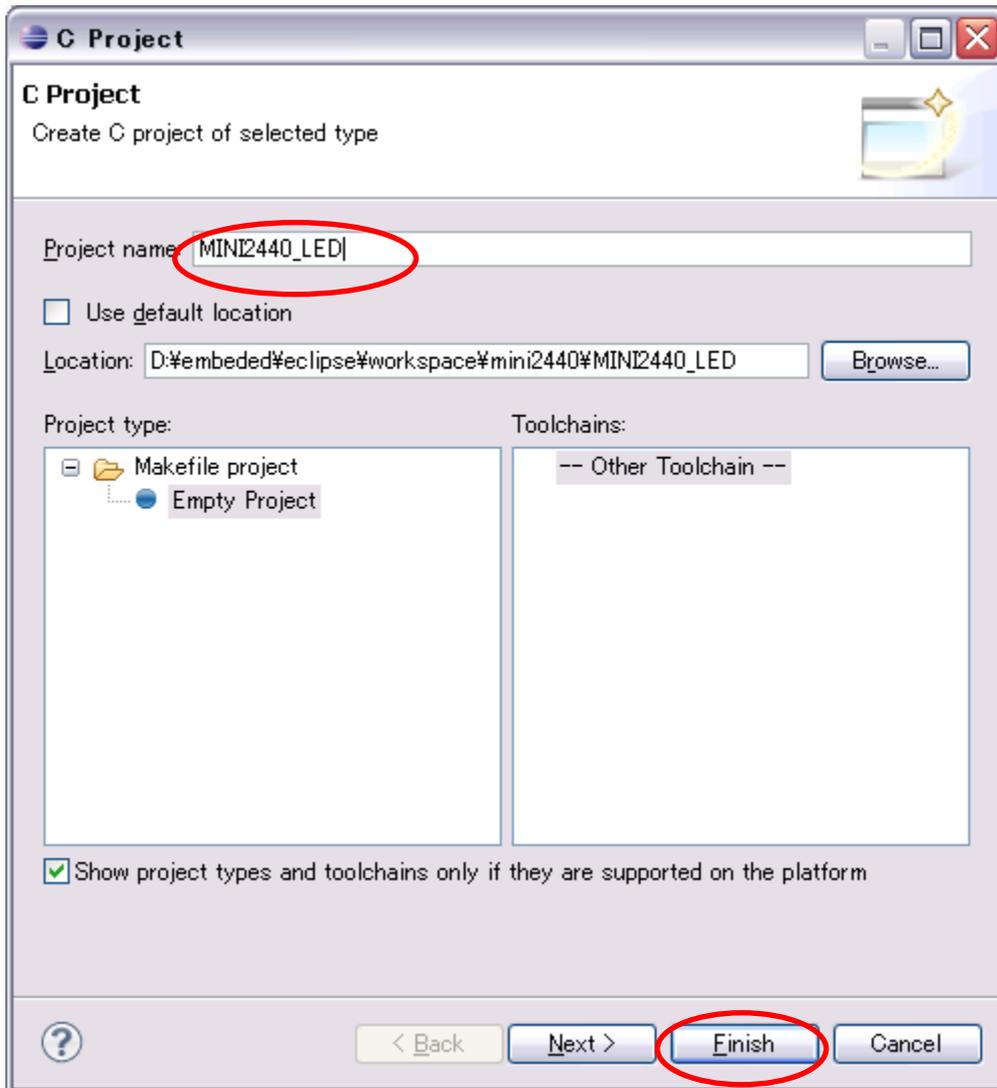
*ここに新規プロジェクトの作成方法として説明

①プロジェクト「MINI2440_LED」を作る

a) 新規プロジェクトを作成するため "File"→"New"→"C Project" を選択します

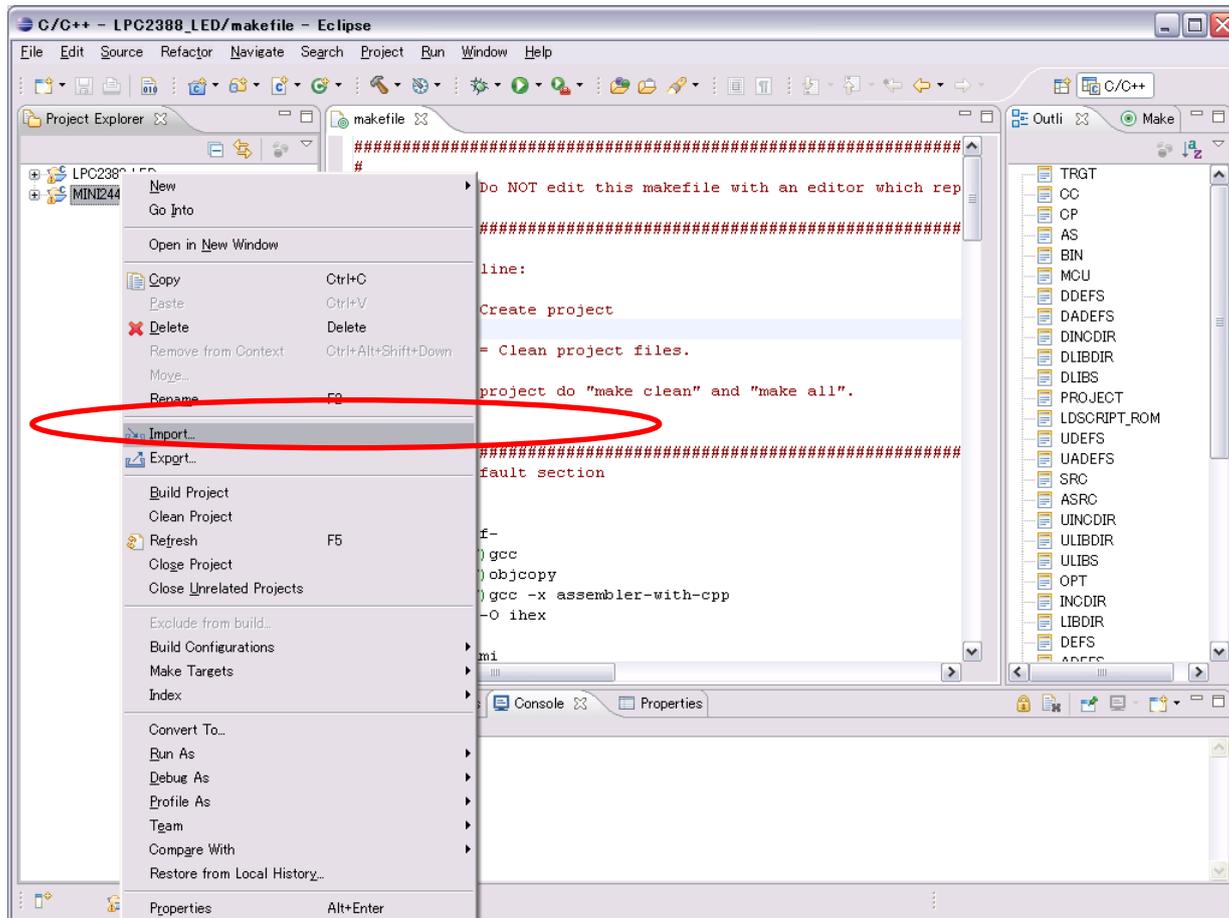


b) 適当なプロジェクト名前(MINI2440_LED)を入力し Finish ボタンを押します。

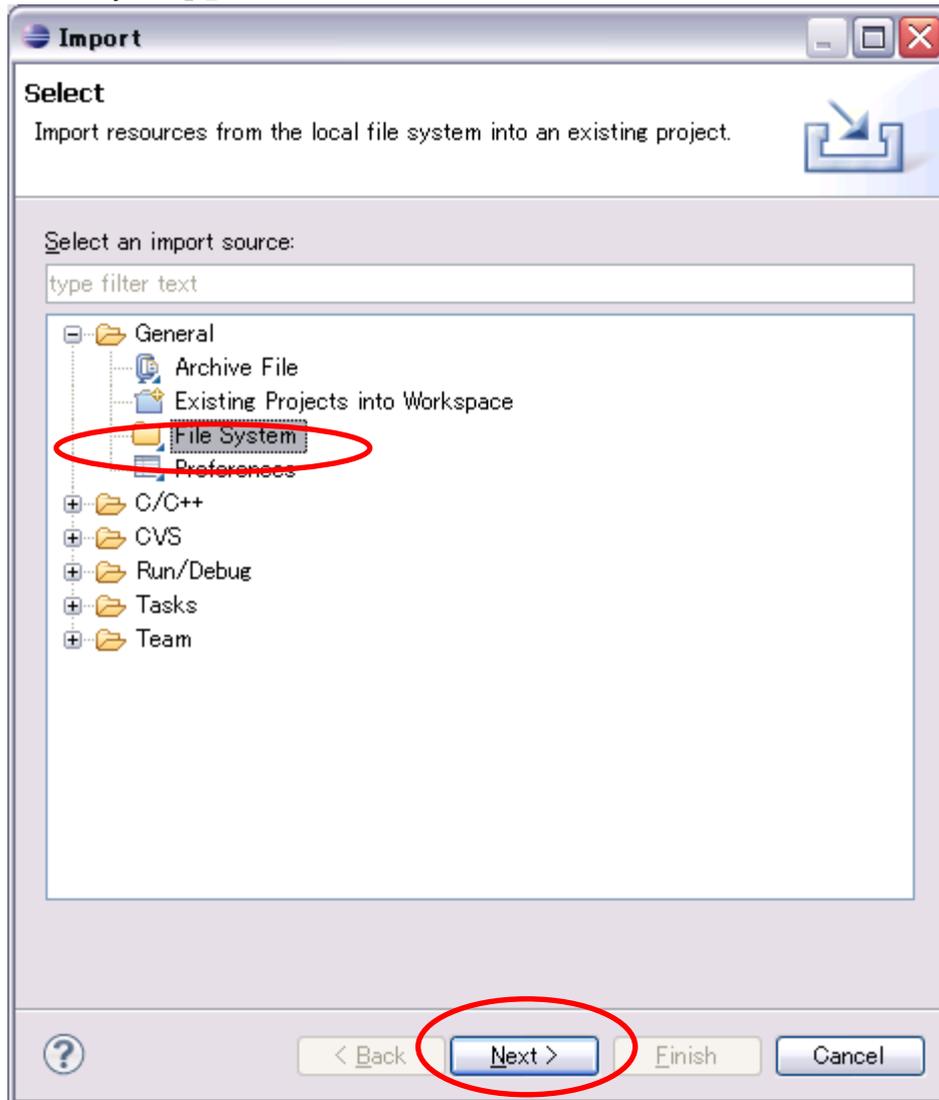


c) サンプルソースを作成されたプロジェクトにインポート

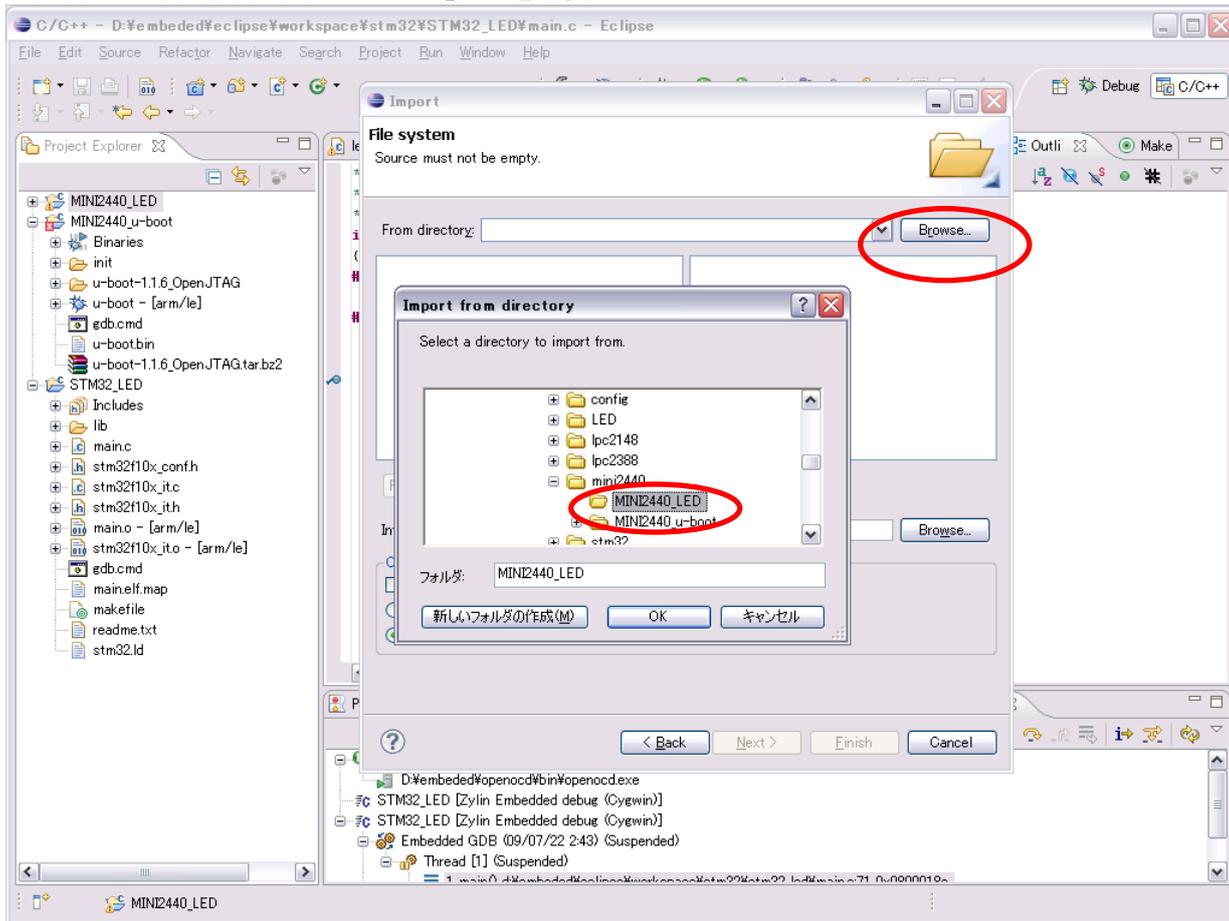
Project Explorer の「MINI2440_LED」を選べ、右クリックし、「Import」を押下



d)「File System」をクリック

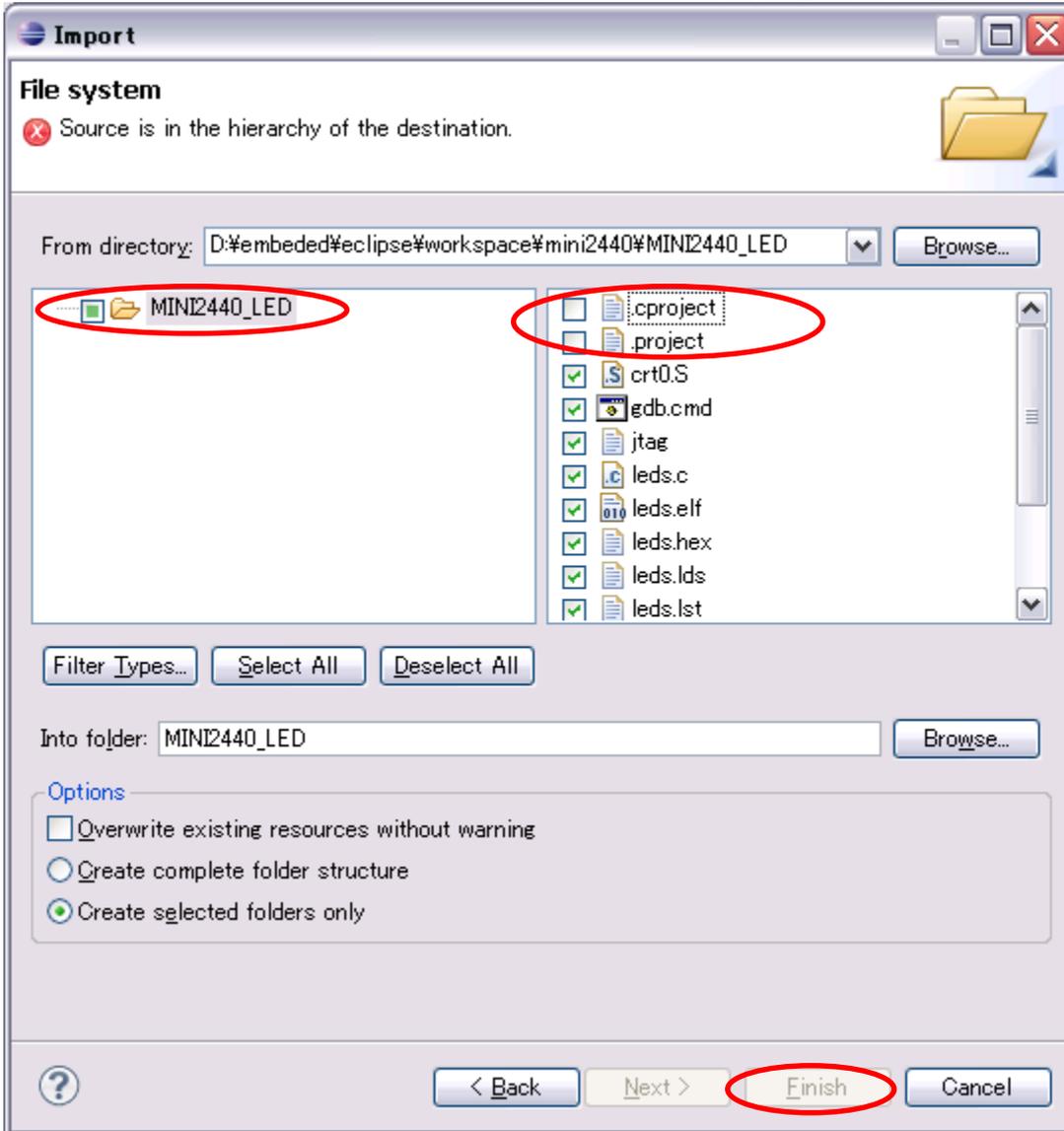


e) ダウンロードしたサンプル MINI2440_LED を選択

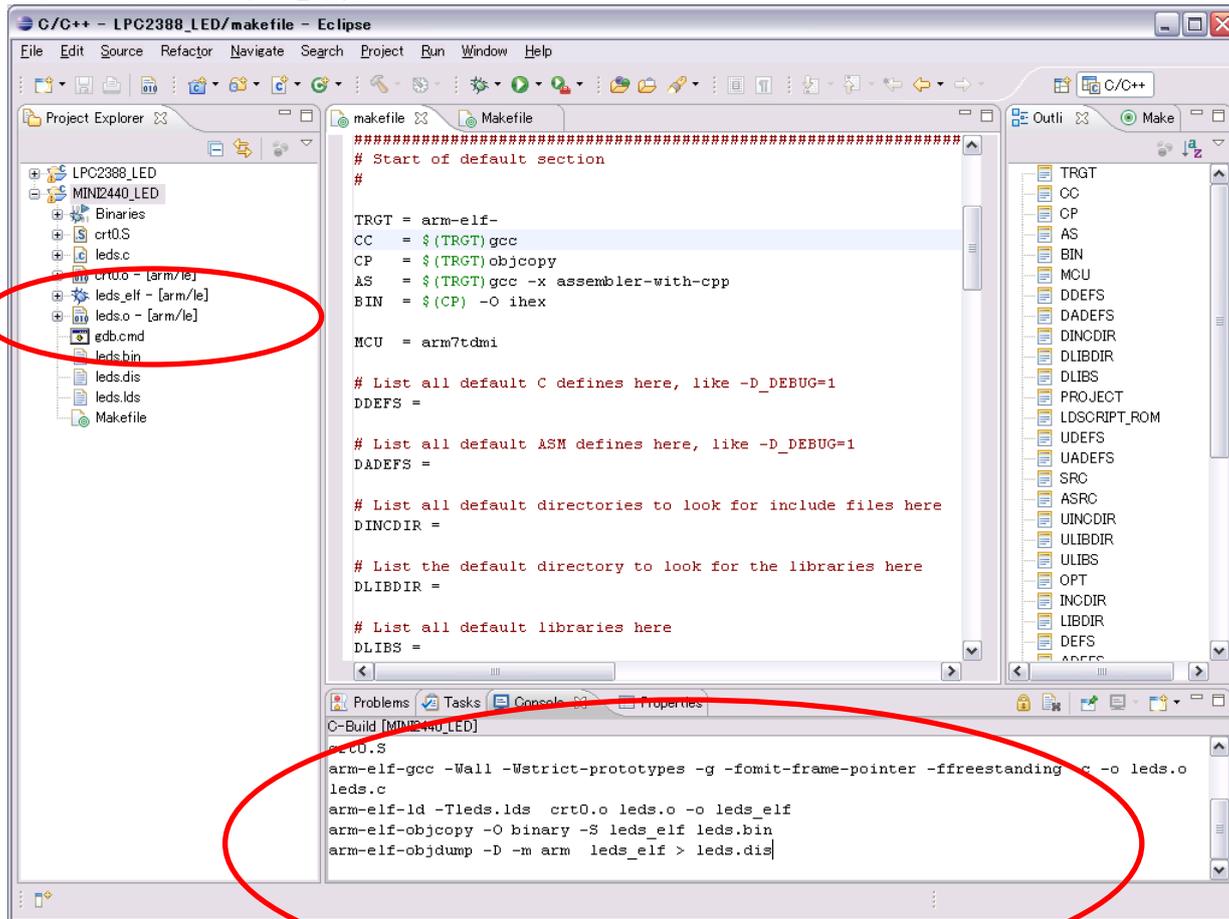


f) 選択したソースを導入

(新規プロジェクトなので、既存プロジェクトの設定ファイル「.cproject」と「.project」のチェックを外す)



h)ビルドが成功かどうかを確認



2. サンプル MINI2440_LED をデバッグ

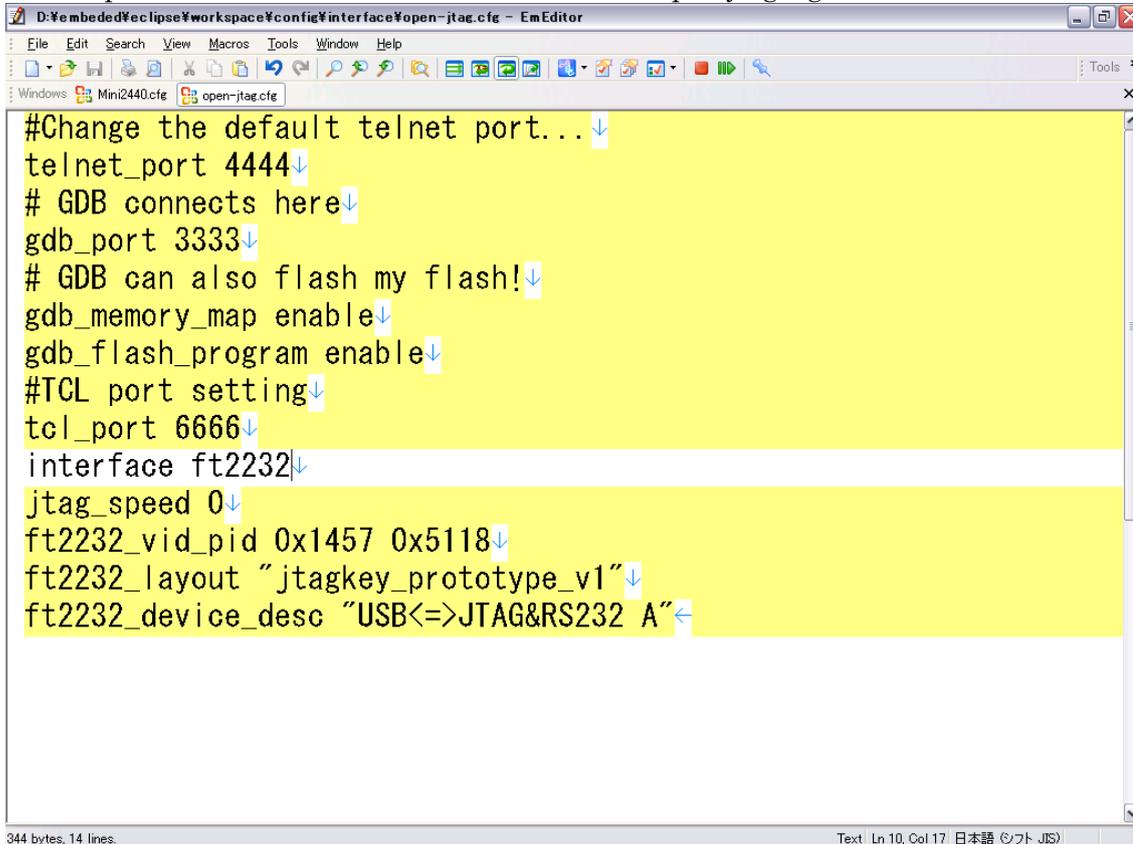
①MINI2440 用の OpenOCD デバッグパラメータを設定

* OpenOCD の設定は「5.4 OpenOCD の設定」(P23~P26)を参照

a) MINI2440 用のコンフィギュレーションファイル内容を確認

「config/interface」の中: JTAG の種類によるコンフィギュレーションファイル

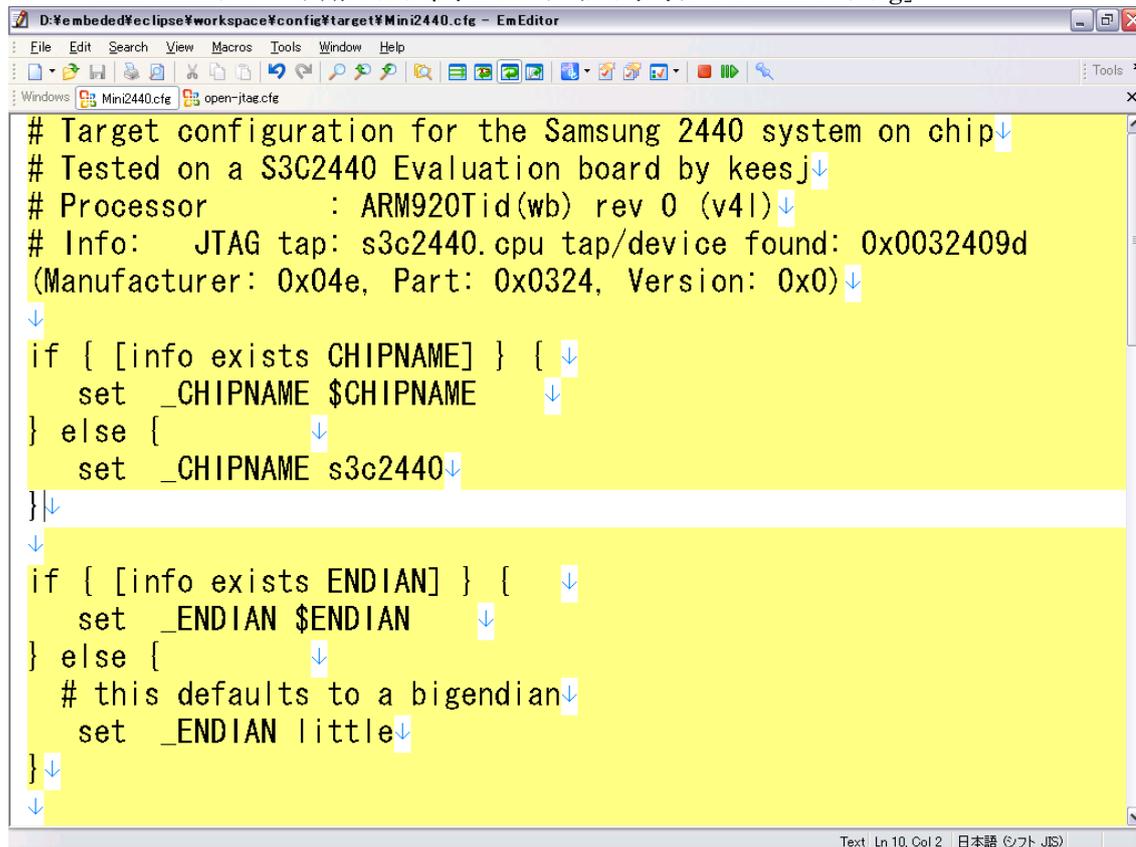
Open-JTAG 用のコンフィギュレーションファイル:「open-jtag.cfg」



```
D:\embeded\workspace\config\interface\open-jtag.cfg - EmEditor
File Edit Search View Macros Tools Window Help
Windows Mini2440.cfg open-jtag.cfg
#Change the default telnet port...
telnet_port 4444
# GDB connects here
gdb_port 3333
# GDB can also flash my flash!
gdb_memory_map enable
gdb_flash_program enable
#TCL port setting
tcl_port 6666
interface ft2232
jtag_speed 0
ft2232_vid_pid 0x1457 0x5118
ft2232_layout "jtagkey_prototype_v1"
ft2232_device_desc "USB<=>JTAG&RS232 A"
```

344 bytes, 14 lines. Text | Ln 10, Col 17 | 日本語 (ソフト JIS)

- b) config/target」の中：CHIP に関わるコンフィギュレーションファイル
MINI2440 ボード用のコンフィギュレーションファイル:「Mini2440.cfg」

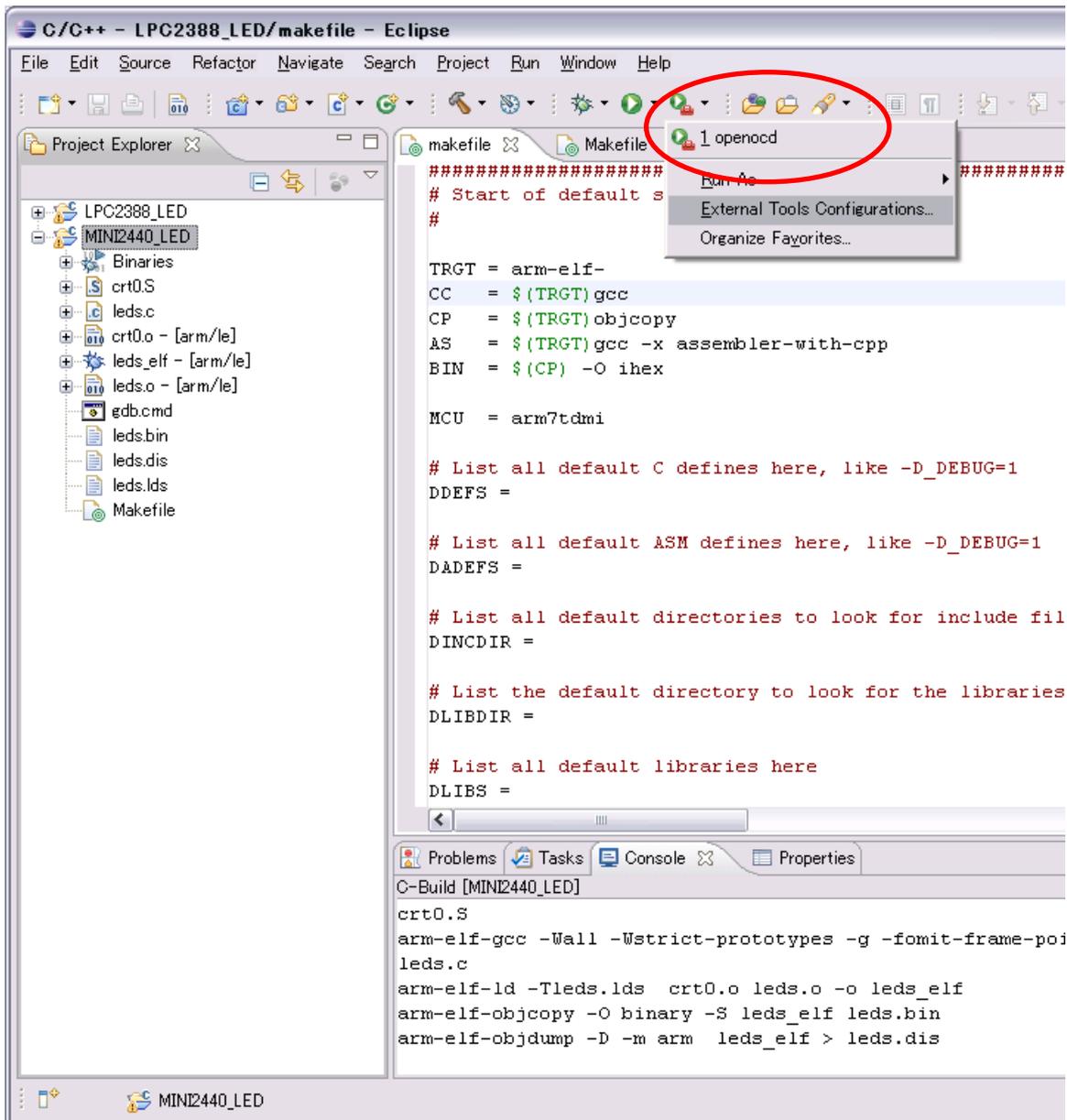


```
# Target configuration for the Samsung 2440 system on chip
# Tested on a S3C2440 Evaluation board by keesj
# Processor      : ARM920Tid(wb) rev 0 (v4l)
# Info:  JTAG tap: s3c2440.cpu tap/device found: 0x0032409d
# (Manufacturer: 0x04e, Part: 0x0324, Version: 0x0)

if { [info exists CHIPNAME] } {
    set _CHIPNAME $CHIPNAME
} else {
    set _CHIPNAME s3c2440
}

if { [info exists ENDIAN] } {
    set _ENDIAN $ENDIAN
} else {
    # this defaults to a bigendian
    set _ENDIAN little
}
```

c) OpenOCD の設定画面を呼び出す

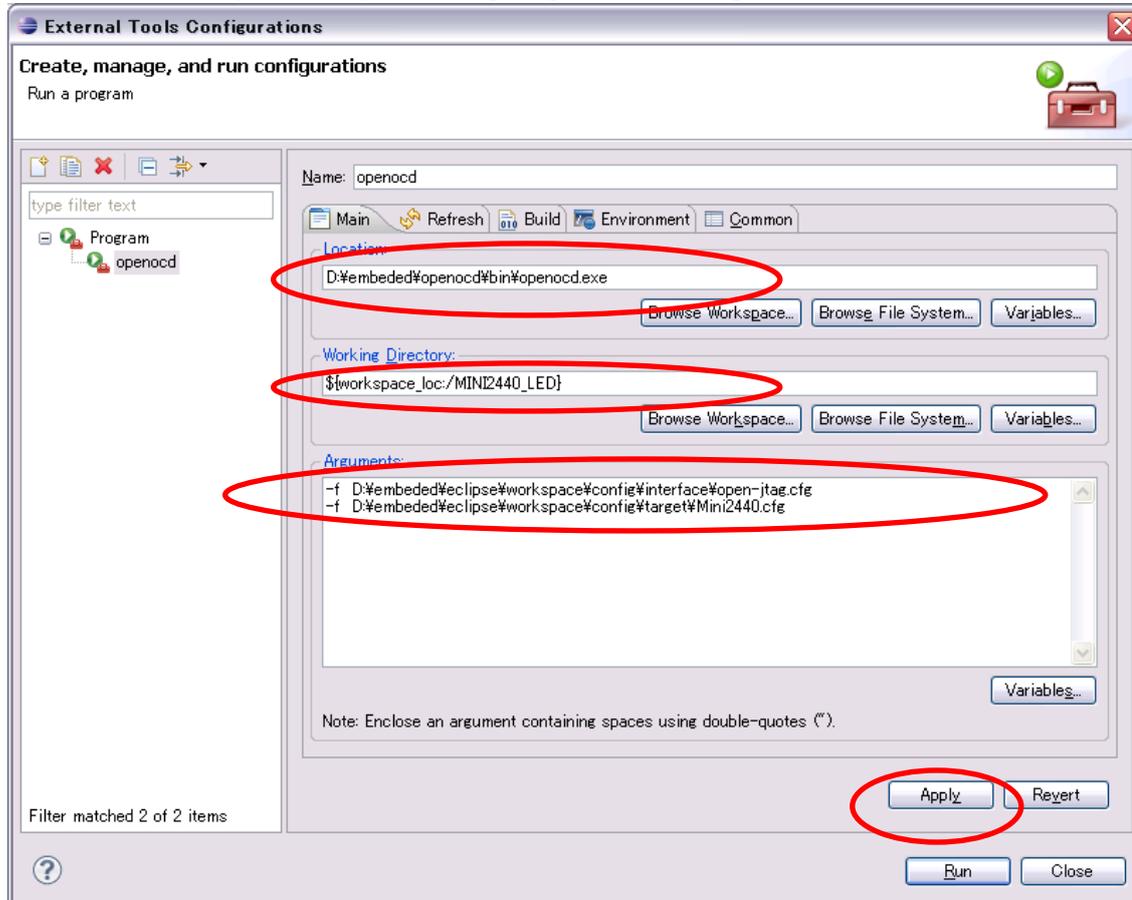


d) 「Location」、「Working Directory」、「Arguments」を設定

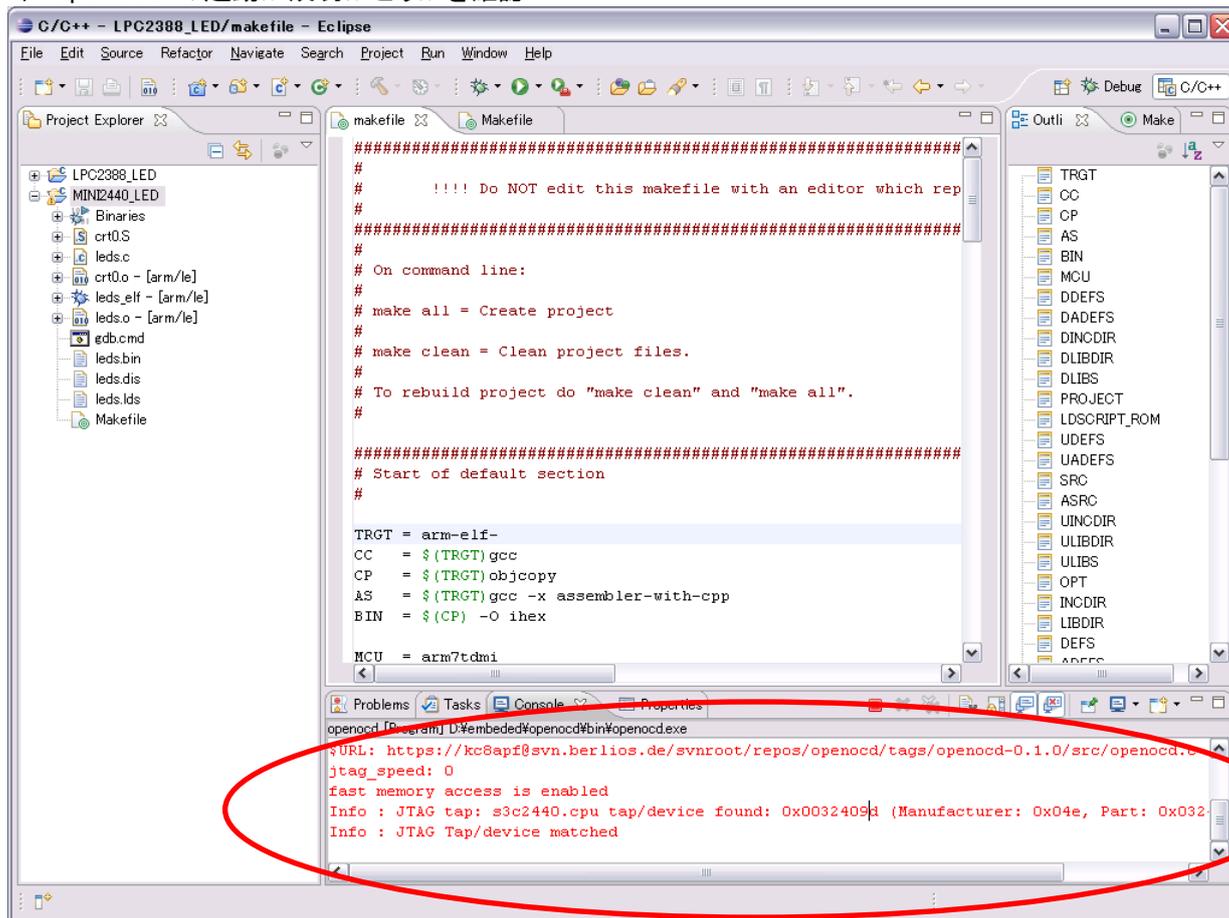
* 特に「Arguments」に MINI2440 用のコンフィギュレーションファイルを設定必要

-f D:\%embedded%\eclipse\workspace\config\interface\open-jtag.cfg

-f D:\%embedded%\eclipse\workspace\config\target\Mini2440.cfg

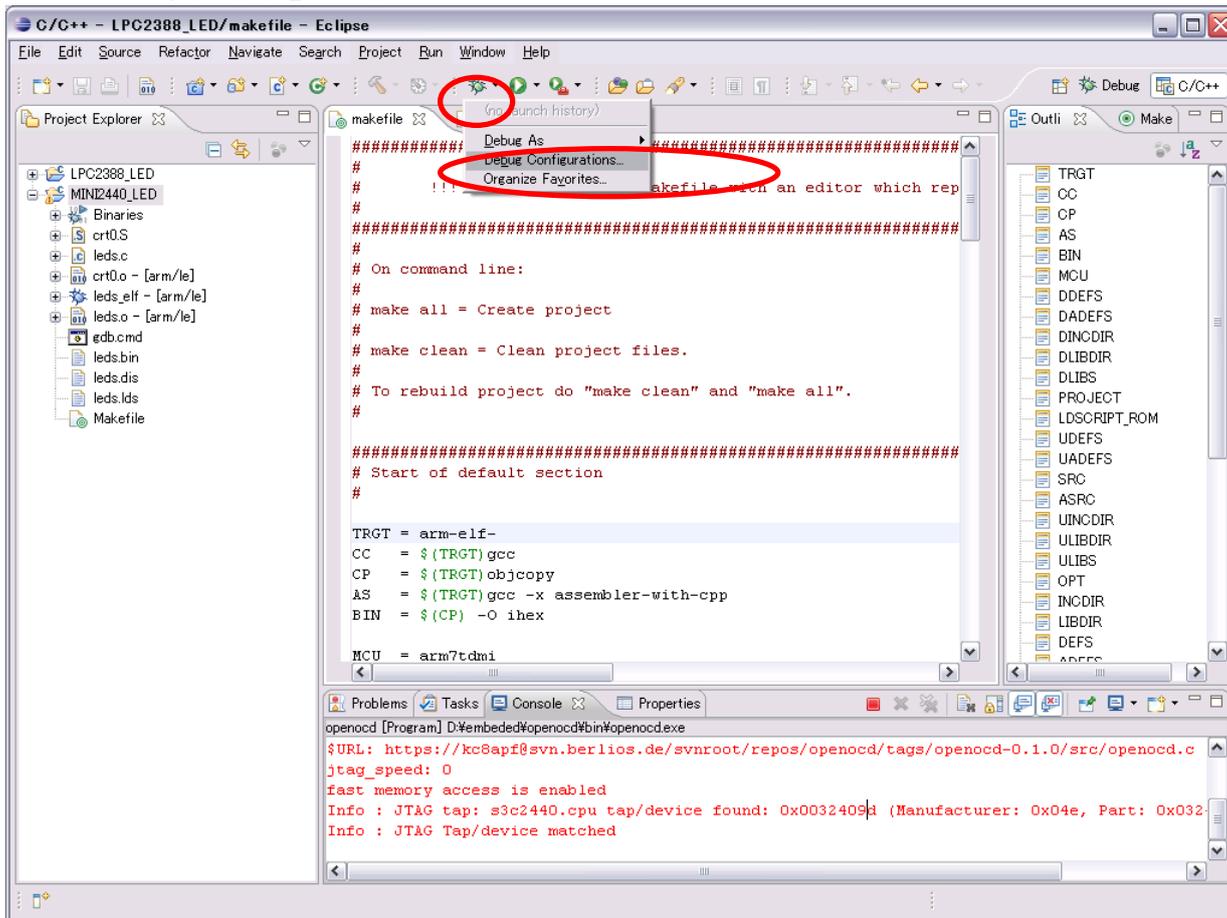


e) OpenOCD の起動が成功かどうかを確認

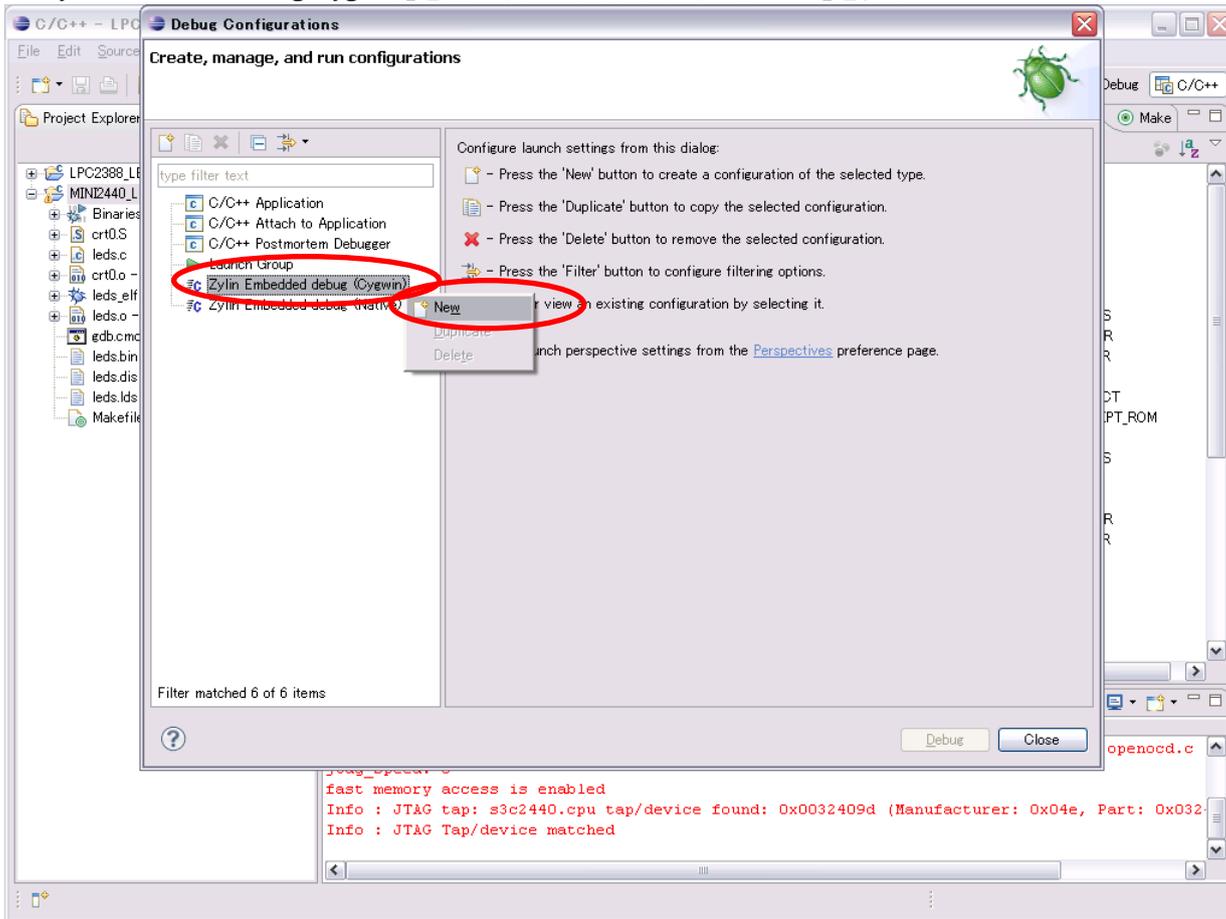


②MINI2440 用のデバッグを設定

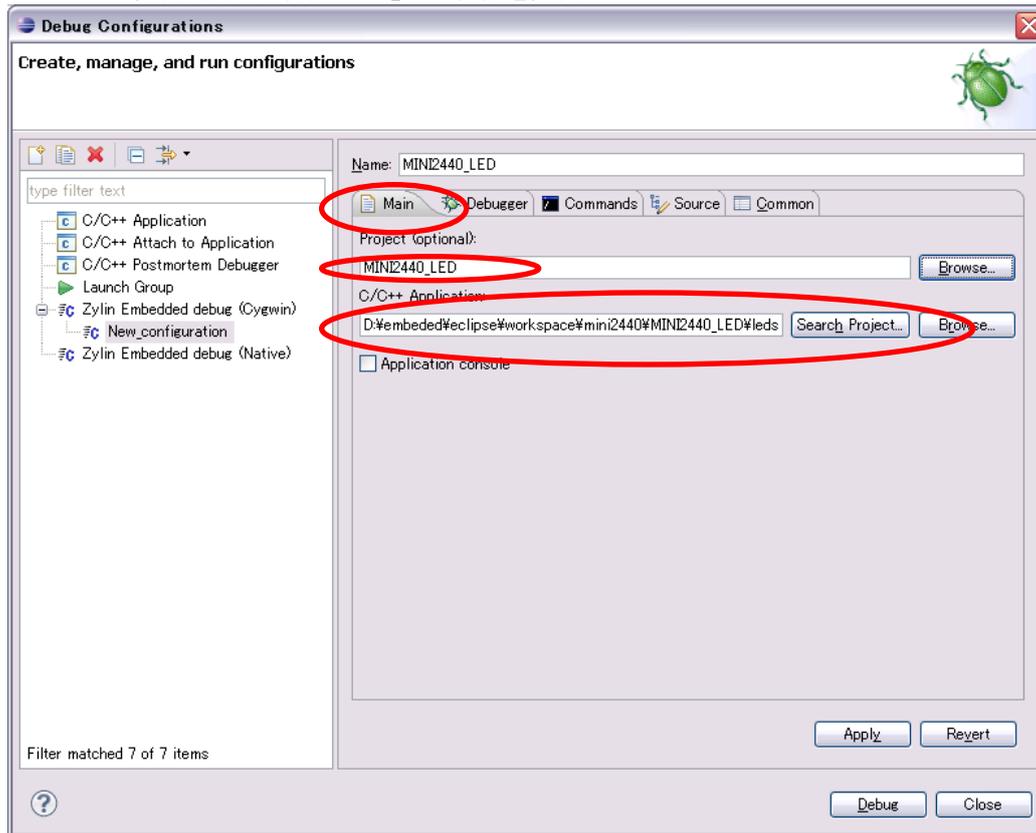
a)デバッグ設定画面を呼び出す



b)「Zylin Embedded debug(Cygwin)」を選べ、右にクリックしてから「New」を押下



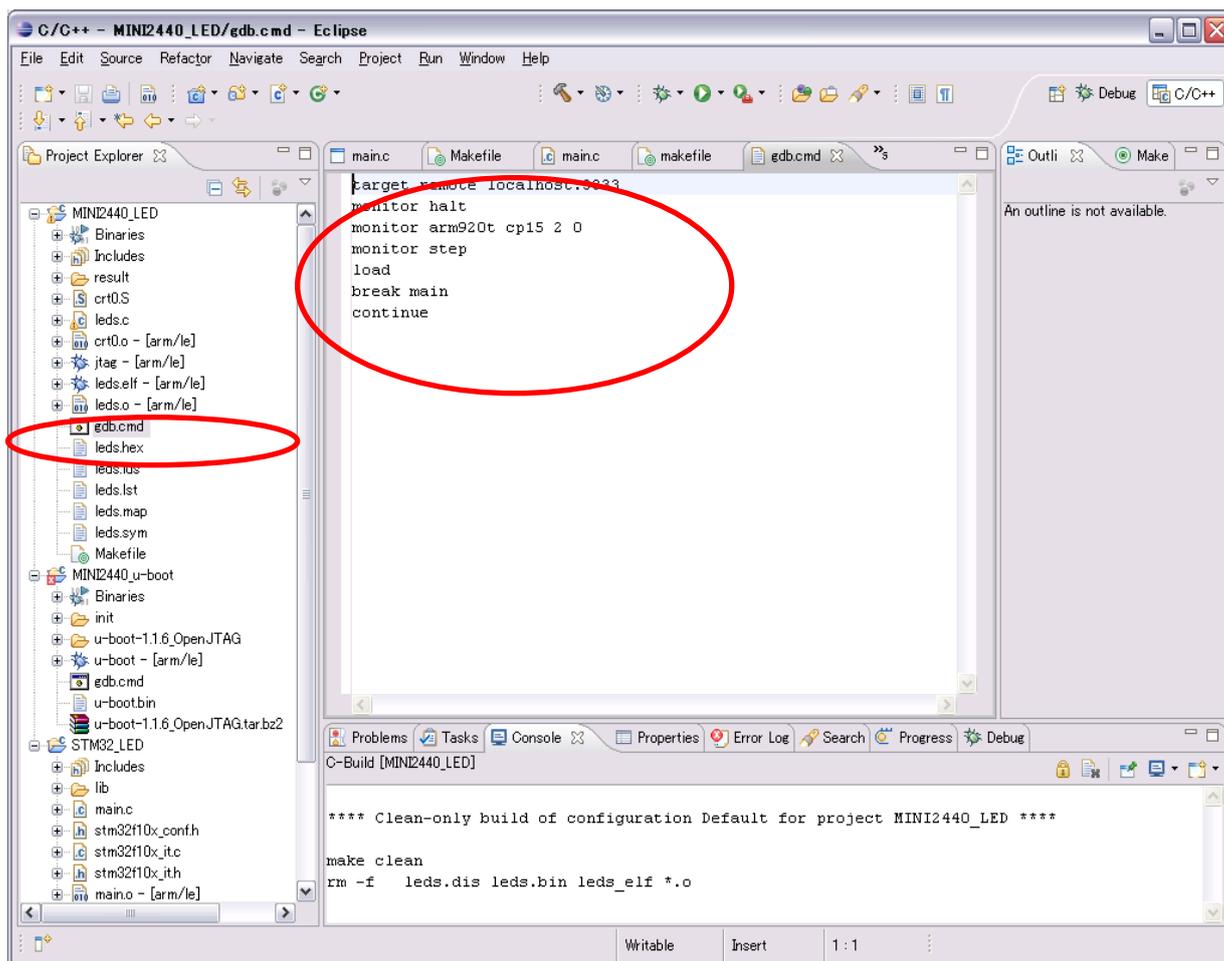
c)デバッグ設定画面の中、「Main」タブ項目を設定



d)「Commonds」タブ項目を設定(//の後にはコマンドのコメントとなり、設定内容に入れない)

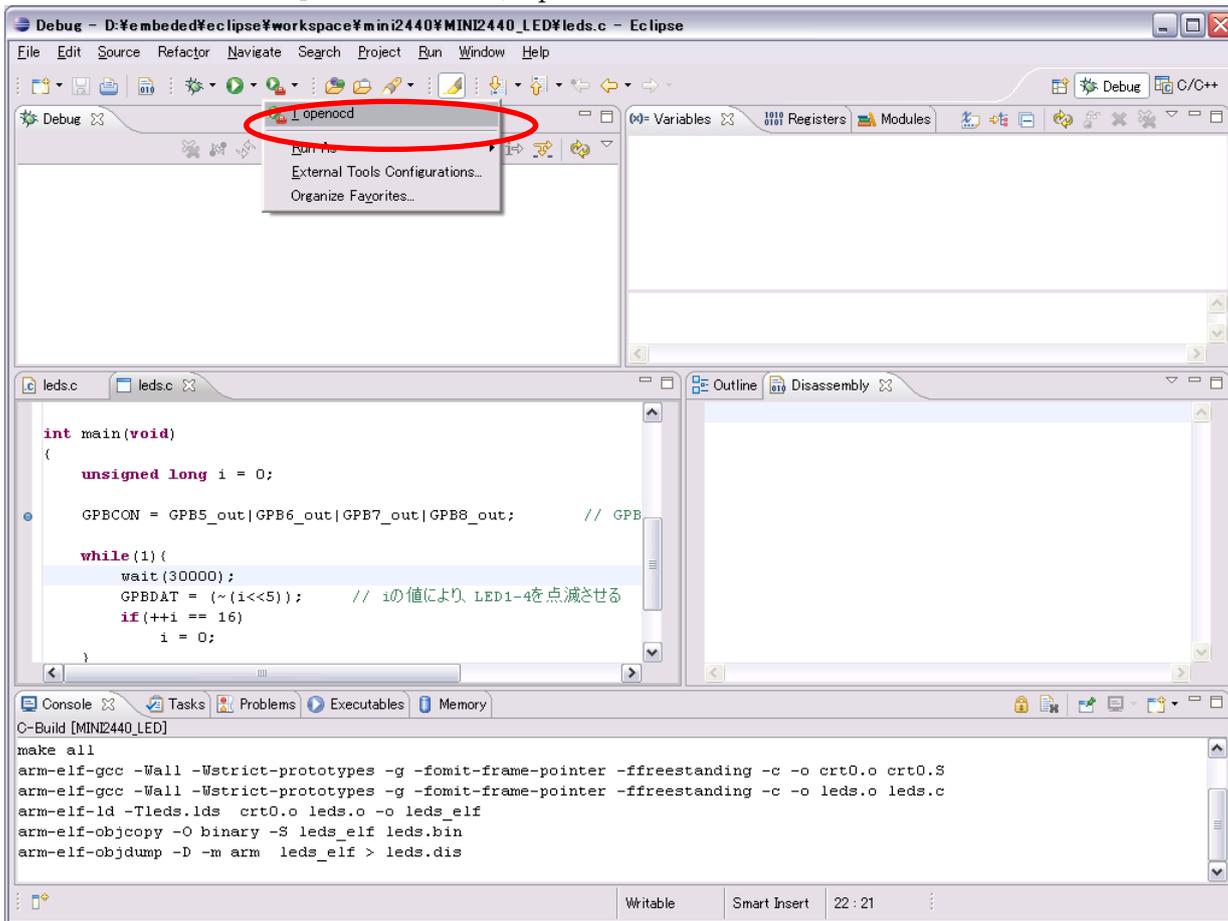
```
target remote localhost:3333 // ローカルポート「3333」と接続 (OpenOCDと接続)
monitor halt//ボードの実行を中断させる
monitor arm920t cp15 2 0 // MMU機能をクローズ
monitor step//ステップで実行するように
load // leds_elfをロード, 「elf」というフォーマットのファイルにアドレスが含まれる
break main // 「main」関数にブレークポイントを設定
continue // プログラムを実行させて、「main」にとまってステップでデバッグ可能
```

「MINI2440_LED」プロジェクトの配下、「gdb.cmd」ファイルの内容をそのままコピーしても OK
「gdb.cmd」を右クリック、「Open With」→「Text Editor」
***ほかの設定を保存するため、コピー途中で「Apply」ボタンを押して適用します。**

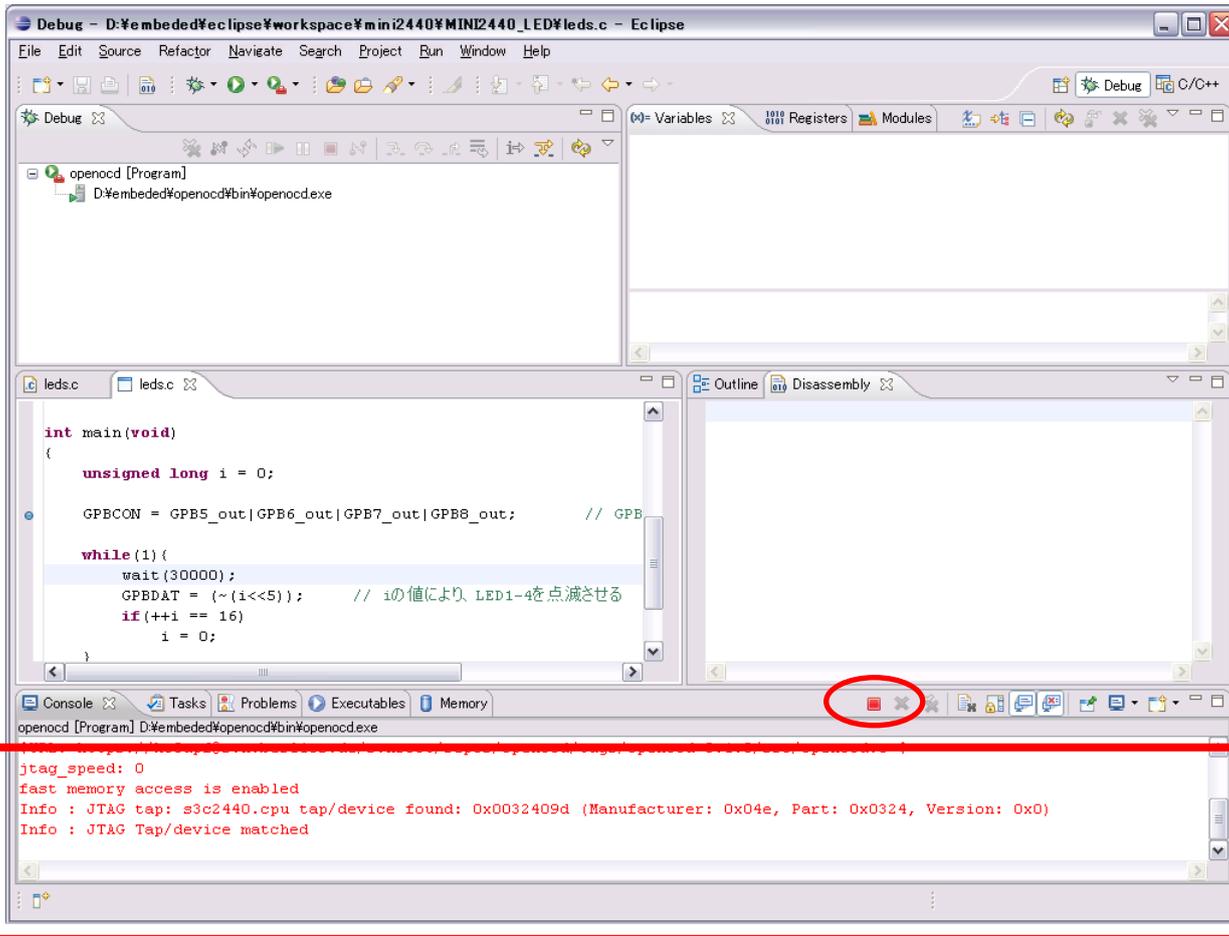


③ OpenOCD を起動させる

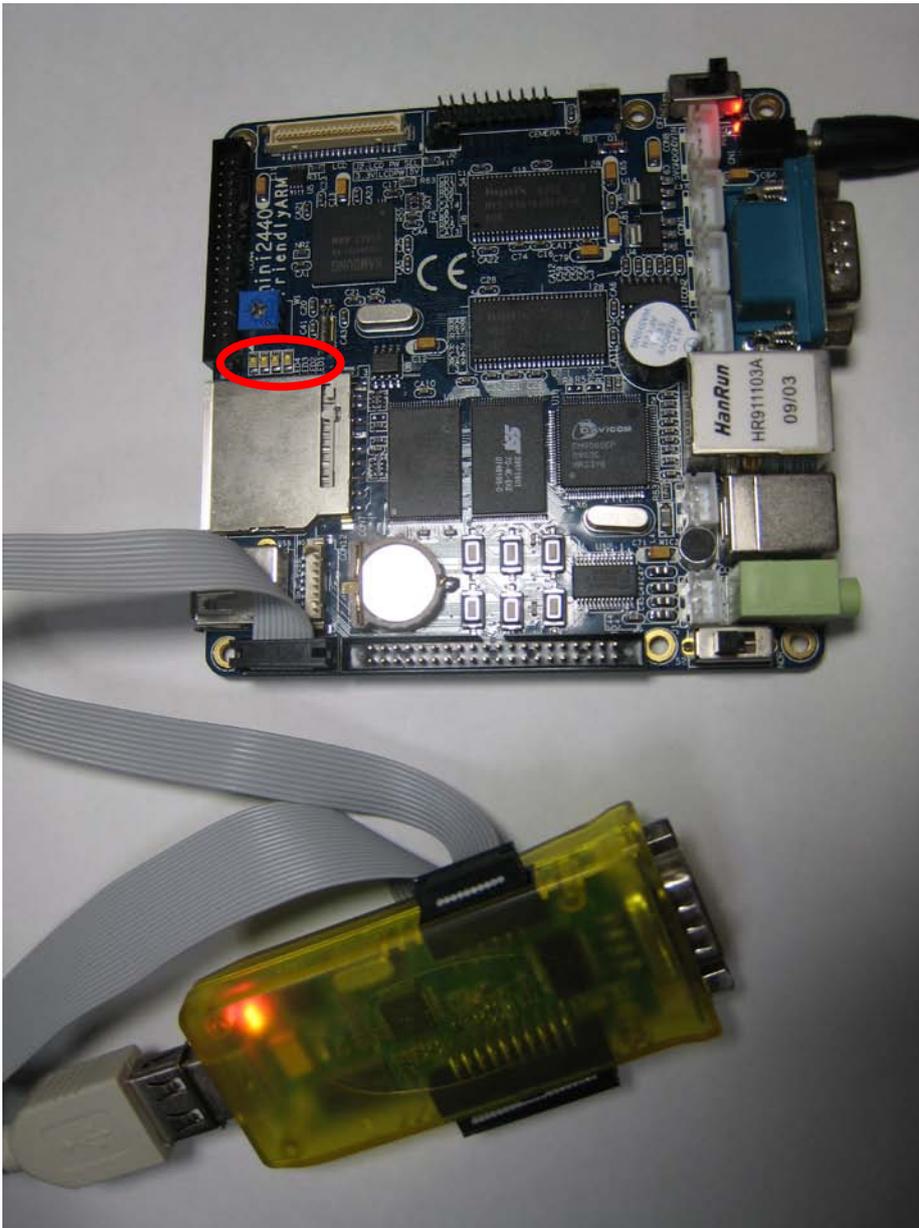
a) 「External Tools」 をクリックし、openocd を押下



b) 成功に起動したことを確認



- ④ Debug ツール「gdb」が起動前、ARM9 ボードの様子
a) 4LED が全て消している



⑤ Debug ツール「gdb」を起動させる

The screenshot shows the Eclipse IDE interface during a debug session. The top toolbar has a red circle around the 'Debug' button. The 'Debug' console on the right shows the variable 'i' with a value of 3. The source code editor shows the following code:

```
int main(void)
{
    unsigned long i = 0;

    GPBCON = GPB5_out|GPB6_out|GPB7_out|GPB8_out;    // GPB

    while (1){
        wait(30000);
        GPBDAT = ~(i<<5);    // iの値により、LED1-4を点滅させる
        if(++i == 16)
            i = 0;
    }
}
```

The disassembly window on the right shows the following assembly code:

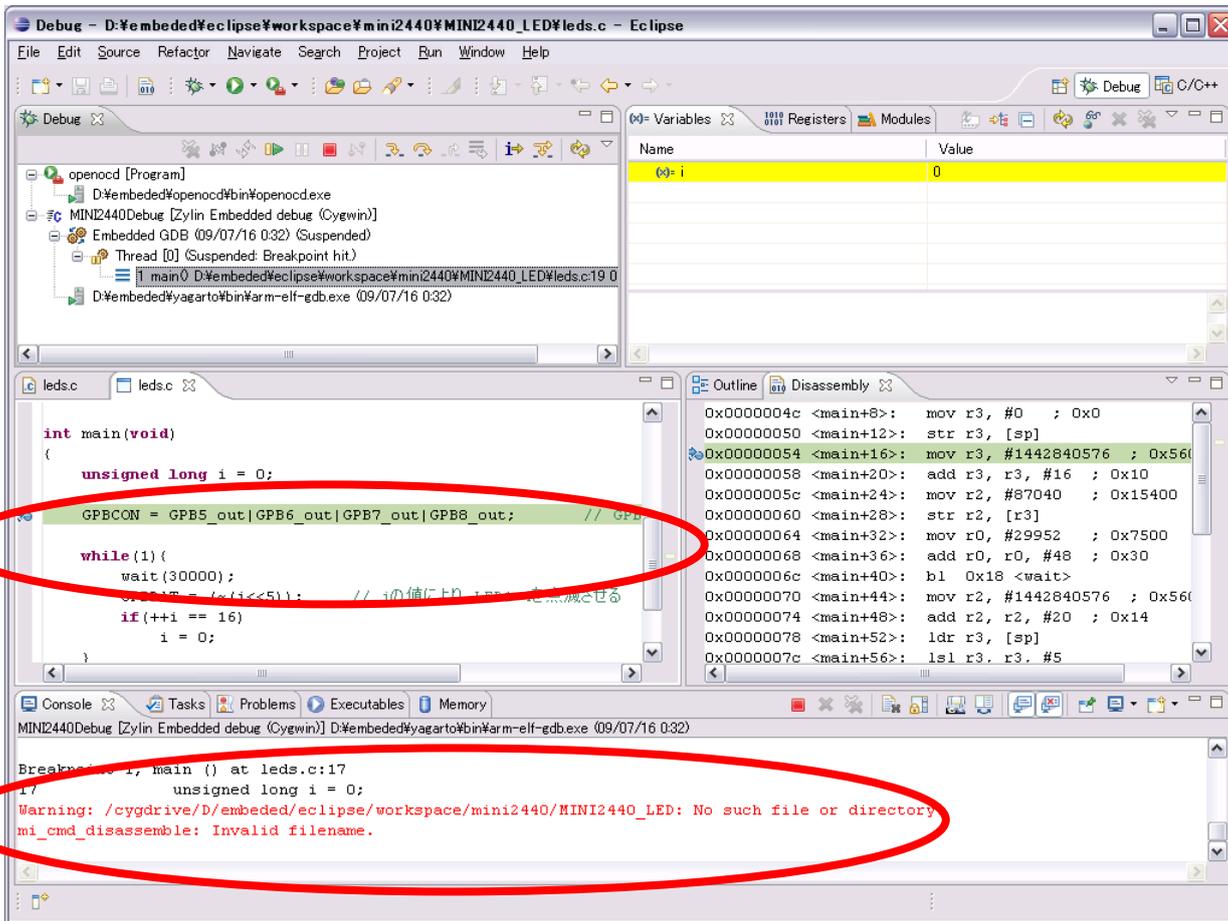
```
0x00000058 <main+20>: add r3, r3, #16 ; 0x10
0x0000005c <main+24>: mov r2, #87040 ; 0x15400
0x00000060 <main+28>: str r2, [r3]
0x00000064 <main+32>: mov r0, #29952 ; 0x7500
0x00000068 <main+36>: add r0, r0, #48 ; 0x30
0x0000006c <main+40>: bl 0x18 <wait>
0x00000070 <main+44>: mov r2, #1442840576 ; 0x56000000
0x00000074 <main+48>: add r2, r2, #20 ; 0x14
0x00000078 <main+52>: ldr r3, [sp]
0x0000007c <main+56>: lsl r3, r3, #5
0x00000080 <main+60>: mvn r3, r3
0x00000084 <main+64>: str r3, [r2]
0x00000088 <main+68>: ldr r3, [sp]
```

The console output shows:

```
MINI2440Debug [Zylin Embedded debug (Cygwin)] D:\embedded\yagarto\bin\arm-elf-gdb.exe (09/07/16 0:32)

Breakpoint 1, main () at led.s.c:17
17      unsigned long i = 0;
Warning: /cygdrive/D/embedded/eclipse/workspace/mini2440/MINI2440_LED: No such file or directory.
mi_cmd_disassemble: Invalid filename.
```

⑥ ウィンドに関数「main」の一番最初に止まっている

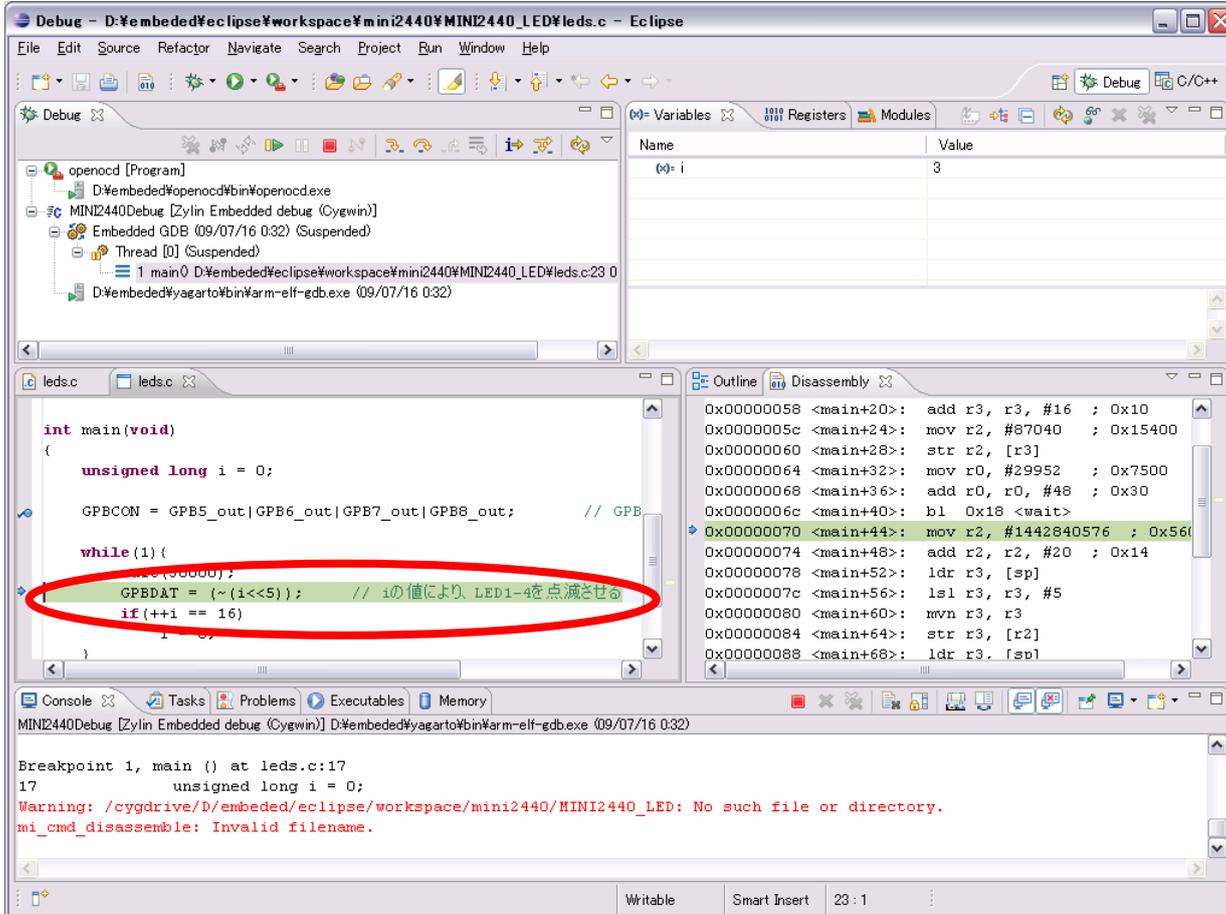


⑦ ステップ実行して Debug (ショットカットキーF6)

a) ブレークポイントでプログラムが中断した状態から、次のブレークポイントまで実行させたり、1行ずつ実行させたりできます。

コード「GPBDAT = (~i << 5)」を繰り返して実行することにより、

LEDランプが1つずつ点滅



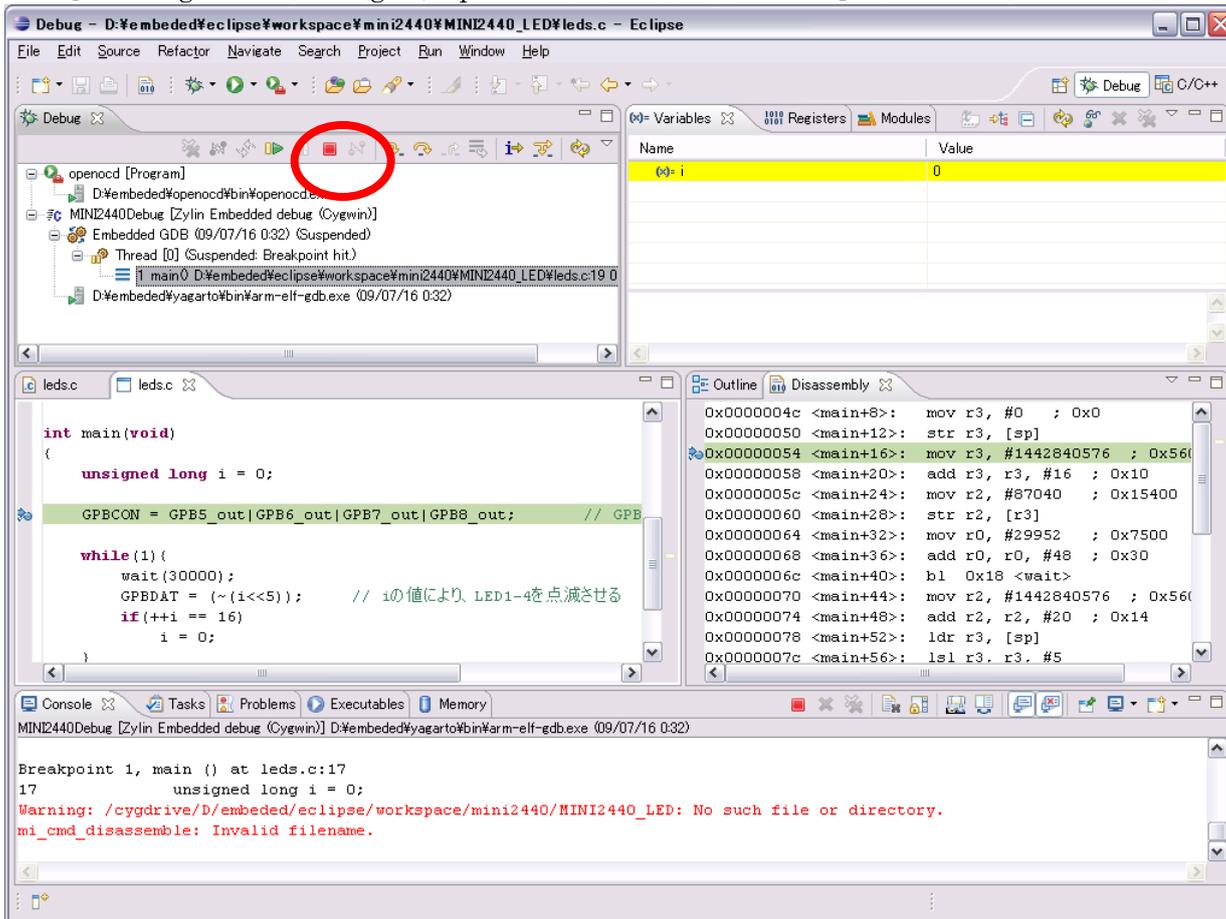
b)Debug 途中の ARM9 ボードの様子の1(一番右の LED ランプが点灯)



c)Debug 途中の ARM9 ボードの様子(右から 2 番目の LED ランプが点灯)



⑧ Debug を停止する (gdb、openocd をそれぞれ選択し「■」をクリック)



The screenshot shows the Eclipse IDE interface during a debug session. The top toolbar contains several icons, with a red square icon (representing 'Stop') circled in red. The main window is divided into several panes:

- Debug Console:** Shows the execution flow, including 'openocd [Program]', 'MINI2440Debug [Zylin Embedded debug (Cygwin)]', 'Embedded GDB (09/07/16 0:32) (Suspended)', and 'Thread [0] (Suspended: Breakpoint hit)'. The current thread is 'main() D:\embedded\eclipse\workspace\mini2440\MINI2440_LED\leds.c:19 0'.
- Variables:** A table showing the current state of variables:

Name	Value
i	0
- Source Editor:** Displays the C code for 'leds.c'. The current line is highlighted: `GPBCON = GPB5_out|GPB6_out|GPB7_out|GPB8_out; // GPB`. The code includes a `while` loop that sets `GPBDAT` and increments `i`.
- Disassembly:** Shows the assembly code for the current instruction: `0x00000054 <main+16>: mov r3, #1442840576 ; 0x56`.
- Console:** Shows the output of the debug session, including a breakpoint hit at line 17 and a warning: `Warning: /cygdrive/D/embedded/eclipse/workspace/mini2440/MINI2440_LED: No such file or directory.`

6.4.4 u-boot サンプルデバッグ

1. サンプルの導入

①プロジェクト「MINI2440_u-boot」を作る

作成方法は「4.4.3 LED サンプルデバッグ」の「①プロジェクト「MINI2440_LED」を作る」を参照

メモ:

a)u-boot のソースコードを Windows 上にコンパイルできないので、コンパイル後のファイルを提供

b)自分のソースをコンパイル場合、下記の 2 点修正が必要

```
#define CONFIG_SKIP_LOWLEVEL_INIT 1 /* SDRAMの初期化を行わない */
```

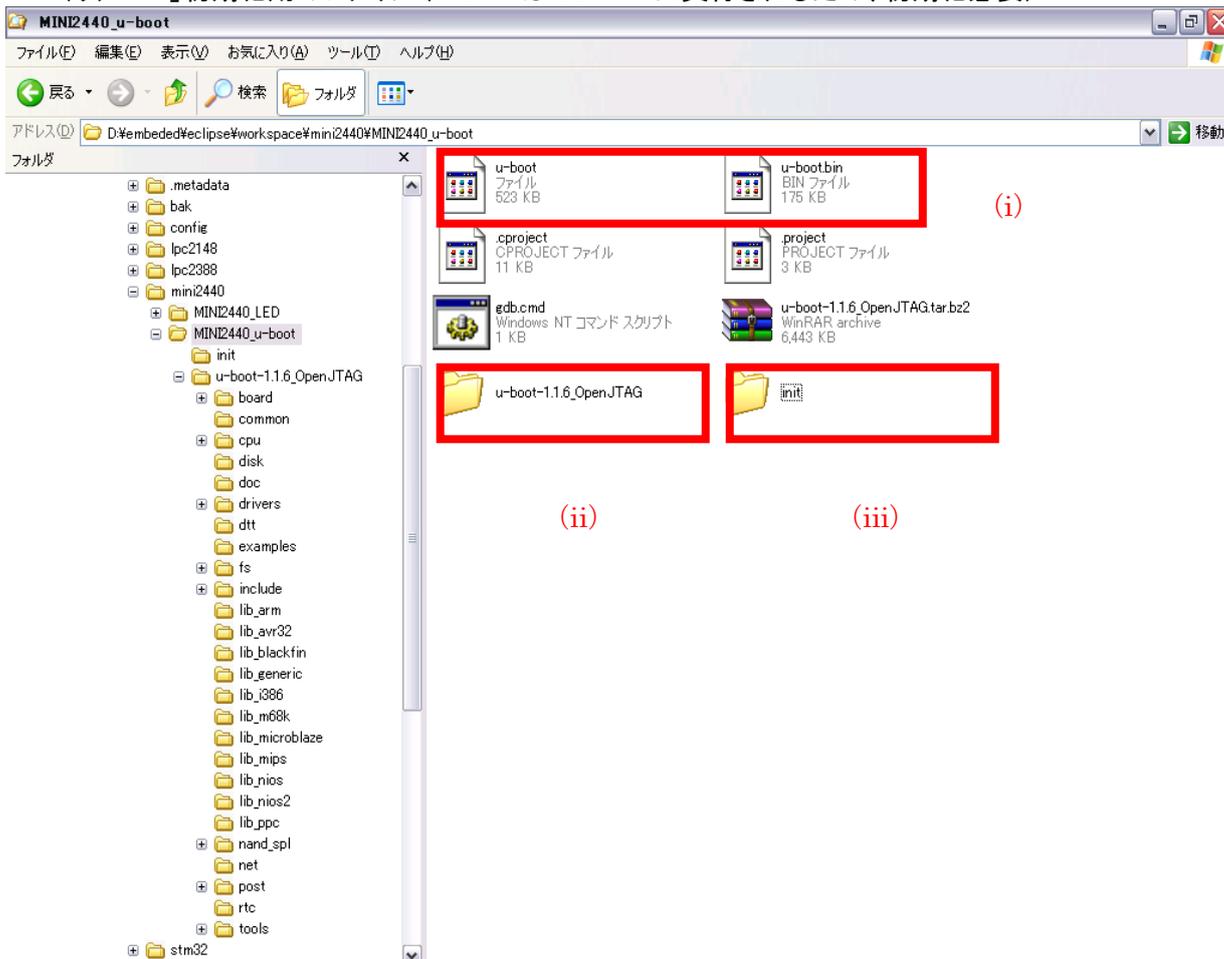
```
#define CONFIG_SKIP_RELOCATE_UBOOT 1 /*実行アドレスをロードされるため、再配置必要なし*/
```

作成されたプロジェクトフォルダ構成:

(ア) コンパイル後のファイル:u-boot は elf ファイルとなる

(イ) デバッグ用の u-boot ソース

(ウ) 「init」初期化用のファイル(u-boot は SDRAM に実行されるため、初期化必要)

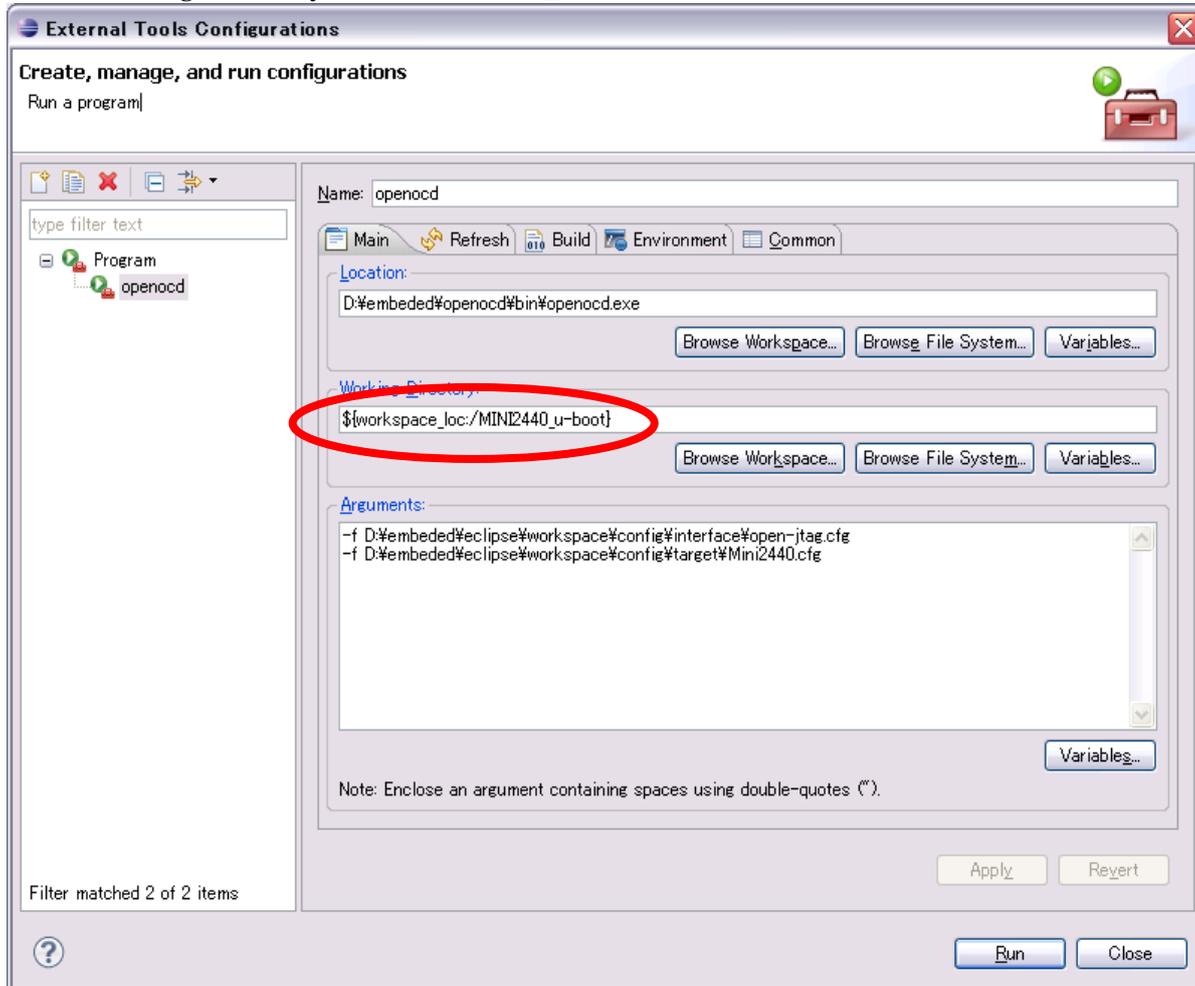


2. サンプル MINI2440_u-boot をデバッグ

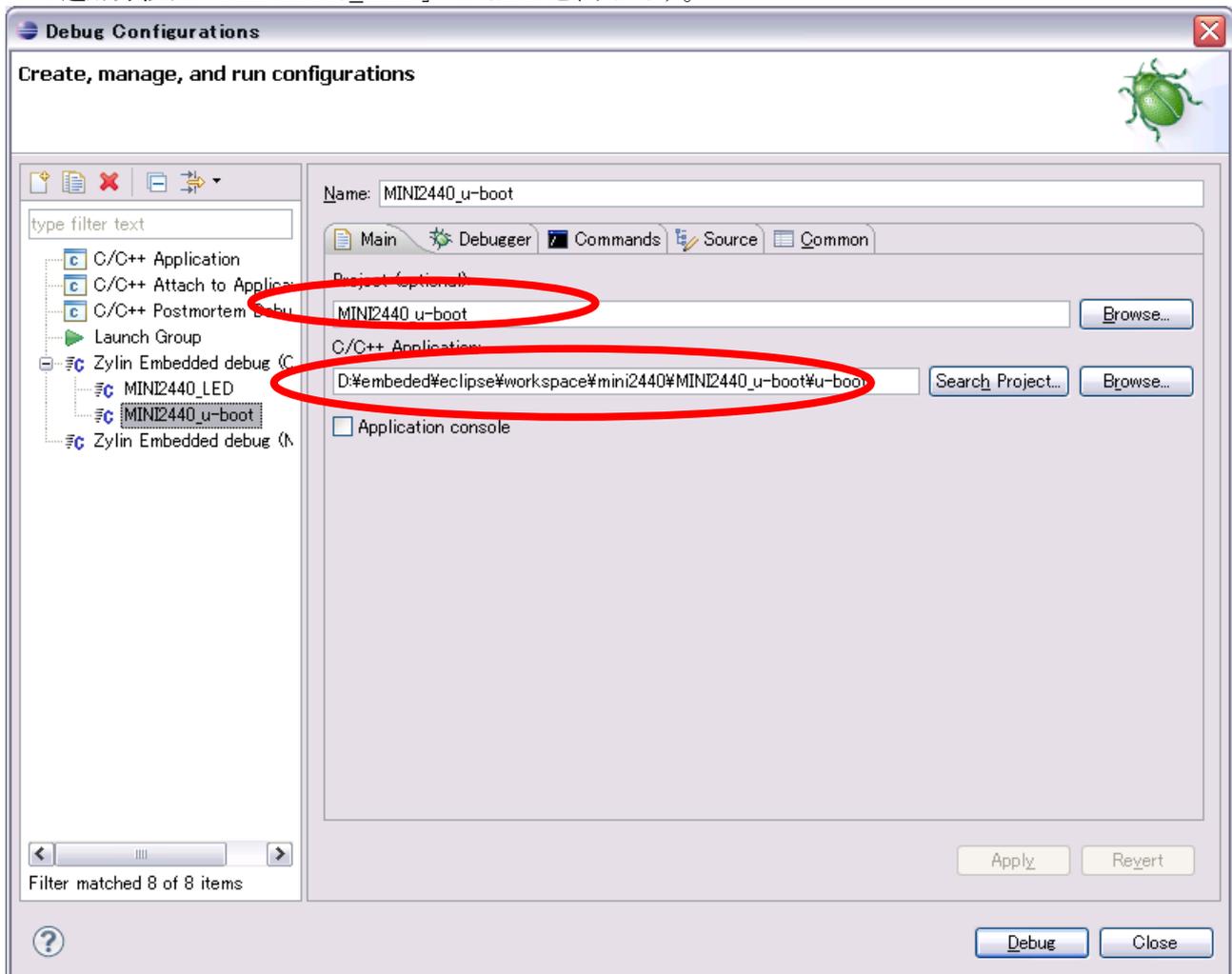
他の設定は MINI2440_LED と同じですので、違い設定のみを説明

① openocd 設定

「Working Directory」を「MINI2440_u-boot」に変更

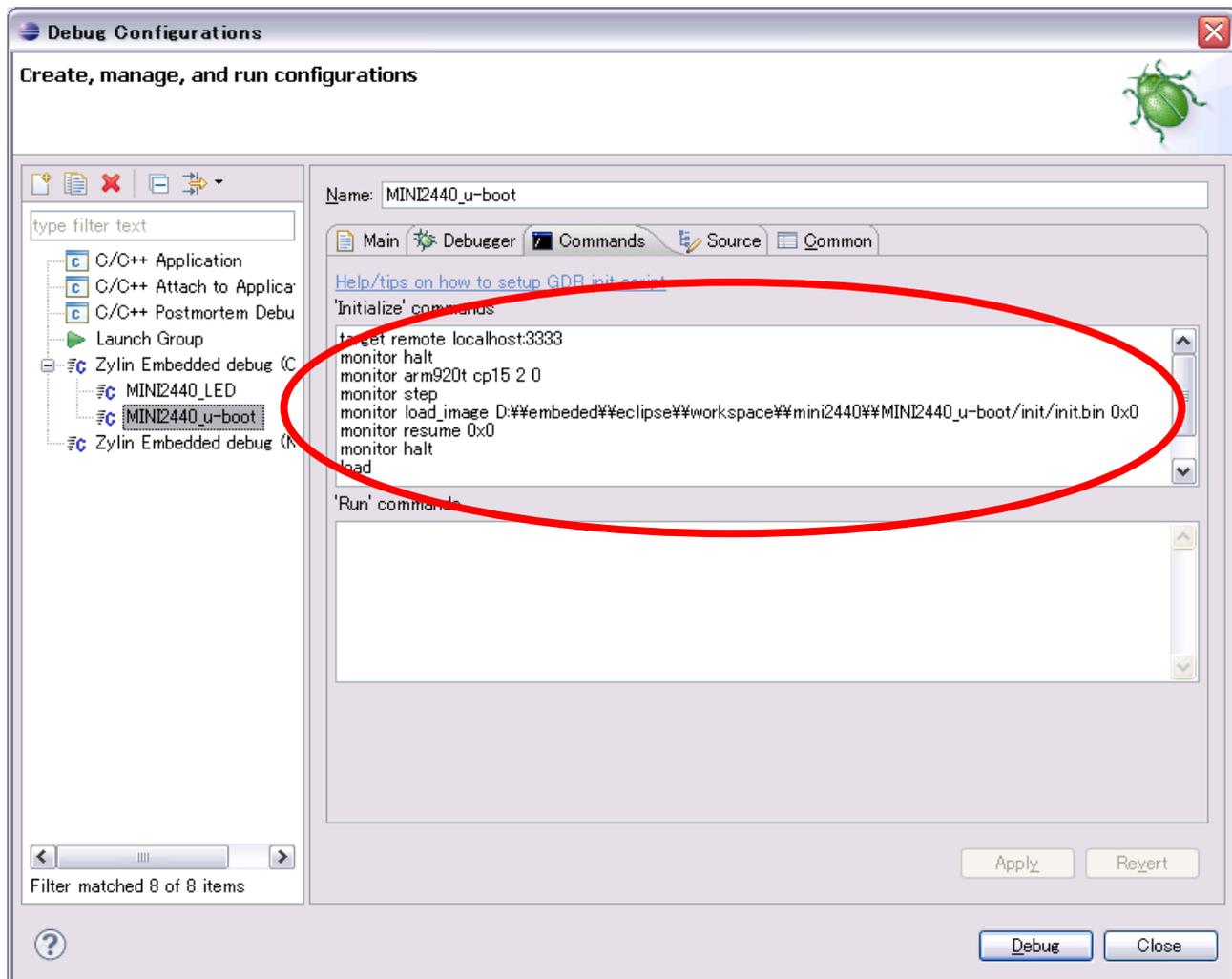


- ② MINI2440_u-boot 用の GDB を追加
*追加方法は「MINI2440_LED」の GDB と同じです。



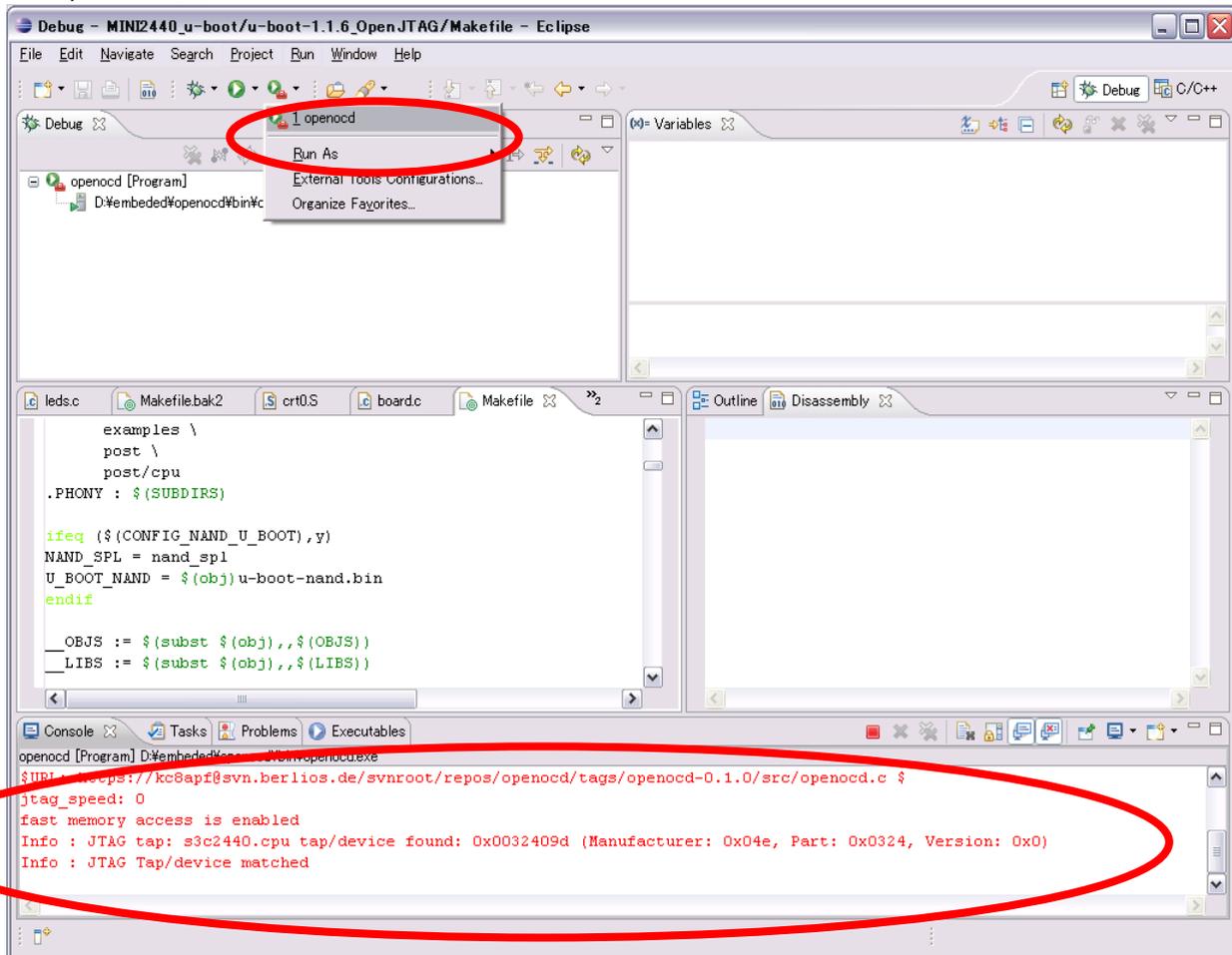
③デバッグ用のコマンドを設定(//はコメント内容となり、実際の設定に入れなくてください。)

```
target remote localhost:3333 // ローカルポート「3333」と接続 (OpenOCDと接続)
monitor halt//ボードの実行を中断させる
monitor arm920t cp15 2 0 // MMU 機能をクローズ
monitor step//ステップで実行するように設定
monitor load_image D:¥¥embedded¥¥eclipse¥¥workspace¥¥mini2440¥¥MINI2440_u-boot/init/init.bin 0x0
//u-boot は SDRAM に実行されるため、初期化実行モジュールをロード(実際の init パスを入力)
monitor resume 0x0//初期化モジュールを実施
monitor halt//ボードの実行を中断させる
load//u-boot をロード
break start_armboot//「start_armboot」関数にブレークポイントを設定
continue// プログラムを実行させて、「start_armboot」にとまってステップでデバッグ可能
```



④デバッグ

a) openocd 起動



b) gdb 起動

関数「start_armboot」に止まり、ショットカットキー「F6」でステップでデバッグできる

The screenshot displays the Eclipse IDE interface for debugging the MIN2440_u-boot project. The 'Variables' window shows the following data:

Name	Value
init_func_ptr	0x0000000c
size	12

The 'Code' window shows the assembly for the 'start_armboot' function, with a red circle highlighting the following instructions:

```
0x33f81540 <start_armboot>: push {r4, r5, r6},
0x33f81544 <start_armboot+4>: sub sp, sp, #68,
gd = (gd_t*) (_armboot_start - CFG_MALLOC_LEN -
0x33f81544 <start_armboot+4>: ldr r5, [pc, #4],
0x33f81548 <start_armboot+8>: ldr r3, [r5],
0x33f8154c <start_armboot+12>: sub r3, r3, #196,
0x33f81550 <start_armboot+16>: sub r3, r3, #36,
0x33f81558 <start_armboot+24>: mov r8, r3,
memset ((void*)gd, 0, sizeof(gd_t)),
0x33f81558 <start_armboot+24>: mov r1, #0 ; (
0x33f81560 <start_armboot+32>: mov r2, #36 ; (
0x33f81564 <start_armboot+36>: mov r0, r3,
0x33f81568 <start_armboot+40>: bl 0x33f970e0 <
```

The 'Console' window shows the following warning messages:

```
MINI2440_u-boot [Zylin Embedded debug (Cygwin)] D:\embedded\yagarto\bin\arm-elf-gdb.exe (09/07/22 1:23)
Warning: /cygdrive/d/embedded/eclipse/workspace/mini2440/MINI2440_u-boot/u-boot-1.1.6_OpenJTAG/board: No such file or direct
Warning: /cygdrive/d/embedded/eclipse/workspace/mini2440/MINI2440_u-boot/u-boot-1.1.6_OpenJTAG: No such file or directory.
Warning: /cygdrive/d/embedded/eclipse/workspace/mini2440/MINI2440_u-boot/init: No such file or directory.
Warning: /cygdrive/d/embedded/eclipse/workspace/mini2440/MINI2440_u-boot: No such file or directory.
No source file named leds.c.
```

c) ステップでデバッグ

コーディング「hang()」を実行したら、MINI2440 ボードからベルを鳴る

The screenshot displays the Eclipse IDE interface during a debug session. The top toolbar includes standard IDE actions and a 'Debug' button. The 'Debug' toolbar shows the current state: 'Thread [0] (Suspended)' and 'start_armboot() board.c:248 0x33f81544'. The 'Variables' window shows the following data:

Name	Value
init_fnc_ptr	0x0000000c
size	12

The 'Code' window shows the following C code snippet, with the `hang();` line circled in red:

```
for (init_fnc_ptr = init_sequence; *init_fnc_ptr; ++init_fnc_ptr) {  
    if (!(*init_fnc_ptr)()) != 0) {  
        hang();  
    }  
}  
  
#ifndef CFG_NO_FLASH  
/* configure available FLASH banks */  
size = flash_init ();  
display_flash_config (size);  
#endif /* CFG_NO_FLASH */
```

The 'Disassembly' window shows the following ARM assembly code:

```
0x33f81540 <start_armboot>: push {r4, r5, r6,  
0x33f81554 <start_armboot+20>: sub sp, sp, #68  
    gd = (gd_t*)( armboot_start - CFG_MALLOC_LEN -  
0x33f81544 <start_armboot+4>: ldr r5, [pc, #43]  
0x33f81548 <start_armboot+8>: ldr r3, [r5]  
0x33f8154c <start_armboot+12>: sub r3, r3, #196  
0x33f81550 <start_armboot+16>: sub r3, r3, #36  
0x33f81558 <start_armboot+24>: mov r8, r3  
    memset ((void*)gd, 0, sizeof (gd_t));  
0x33f8155c <start_armboot+28>: mov r1, #0 ; (  
0x33f81560 <start_armboot+32>: mov r2, #36 ; (  
0x33f81564 <start_armboot+36>: mov r0, r3  
0x33f81568 <start_armboot+40>: bl 0x33f970e0
```

The 'Console' window shows the following warning messages:

```
MINI2440_u-boot [Zylin Embedded debug (Cygwin)] D:\embedded\yagarto\bin\arm-elf-gdb.exe (09/07/22 1:23)  
Warning: /cygdrive/D/embedded/eclipse/workspace/mini2440/MINI2440_u-boot/u-boot-1.1.6_OpenJTAG/board: No such file or direct  
Warning: /cygdrive/D/embedded/eclipse/workspace/mini2440/MINI2440_u-boot/u-boot-1.1.6_OpenJTAG: No such file or directory.  
Warning: /cygdrive/D/embedded/eclipse/workspace/mini2440/MINI2440_u-boot/init: No such file or directory.  
Warning: /cygdrive/D/embedded/eclipse/workspace/mini2440/MINI2440_u-boot: No such file or directory.  
No source file named leds.c.
```