BeagleBone Black Eclipse and GDB

From TinCanTools

GDB, the GNU Project Debugger (http://linux.die.net/man/1/gdb) is a debugging tool provided with the GNU Compiler Collection (GCC). GDB allows you to stop and start a running program, examine its functioning, and make changes.

The Eclipse IDE (http://www.eclipse.org) is a multi-language integrated development environment which can be used to develop, install, and debug embedded applications.

This guide will walk you through installing Eclipse and configuring it for embedded development, installing a GCC toolchain to compile programs for ARM processors, and finally compiling and debugging a simple program in GDB. These instructions were written for Ubuntu 12.04 LTS and the Texas Instruments BeagleBone Black.

The BeagleBone Black comes pre-loaded with Angstrom Linux, but this guide will walk you through compiling, loading, and debugging a "bare metal" program. Users wishing to develop programs that run within the BeagleBone's operating system may not find much help here, but plenty of useful guides are available elsewhere. (The BeagleBone's Linux install includes tutorial information to help you get started. To access it, just connect the BeagleBone to a USB port on your PC and go to http://192.168.7.2 in Firefox or Chrome.)

Contents

Install the ARM EABI Toolchain

To compile programs for ARM processors you will need a cross-compiler and toolchain. This guide will compile and install the ARM EABI toolchain via a script created by James Snyder (https://github.com/jsnyder/arm-eabi-toolchain).

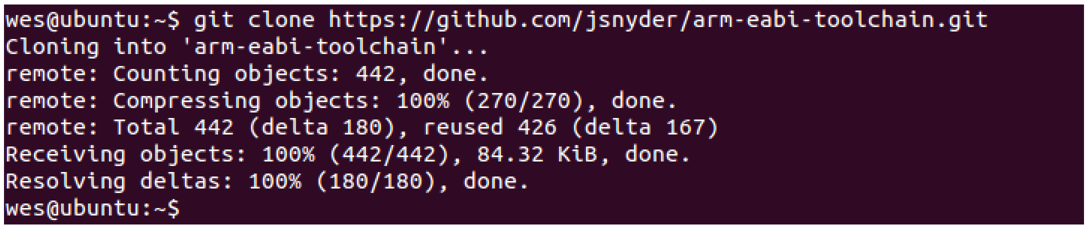
Step 1: Download the Script via Git.

Git is version-control software used for managing code and assets during software development. Here we use it to download Snyder's script. In a terminal window, type:

```
sudo apt-get install git
```

When the install finishes, type:

```
cd ~
git clone https://github.com/jsnyder/arm-eabi-toolchain.git
```



```
wes@ubuntu:~$ git clone https://github.com/jsnyder/arm-eabi-toolchain.git
Cloning into 'arm-eabi-toolchain'...
remote: Counting objects: 442, done.
remote: Compressing objects: 100% (270/270), done.
remote: Total 442 (delta 180), reused 426 (delta 167)
Receiving objects: 100% (442/442), 84.32 KiB, done.
Resolving deltas: 100% (180/180), done.
wes@ubuntu:~$
```

This will create a folder called **arm-eabi-toolchain** in your Home directory, containing the makefile script we need.

Step 2: Install Dependencies.

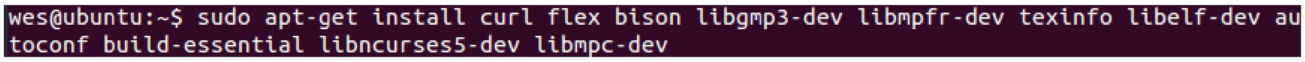
The makefile requires several utilities and libraries to compile and install correctly. In a terminal window type (all on one line):

```
sudo apt-get install curl flex bison libgmp3-dev libmpfr-dev texinfo
 libelf-dev autoconf build-essential libncurses5-dev libmpc-dev
```

WARNING: Don't try to copy and paste the text above into your terminal window! If you want to copy and paste so you don't miss anything, use the line below:

```
sudo apt-get install curl flex bison libgmp3-dev libmpfr-dev texinfo libelf-dev autoconf build-essential libncurses5-dev libmpc-dev
```

Select the text in your browser, press **Ctrl+C**, then right-click on the terminal window and select **Paste**. Press Enter to install.



```
wes@ubuntu:~$ sudo apt-get install curl flex bison libgmp3-dev libmpfr-dev texinfo libelf-dev au
toconf build-essential libncurses5-dev libmpc-dev
```

You can find this list of dependencies in Snyder's readme file, installed at **/home/USERNAME/arm-eabi-toolchain/readme.md**.

Step 3: Install libexpat.

Before we compile the toolchain we need to install one more library. In the terminal window, type:

```
sudo apt-get install libexpat-dev
```

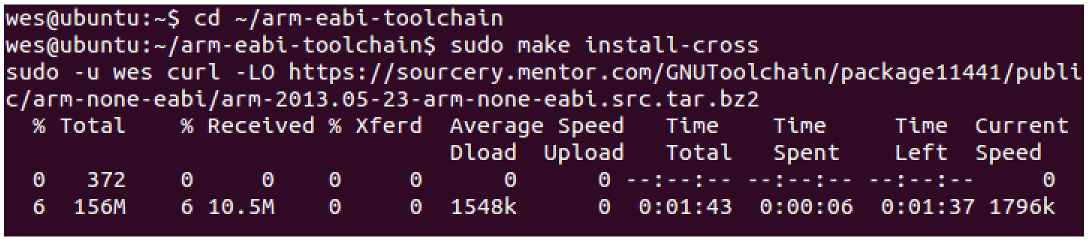
Without this library, the toolchain will compile successfully but the debugger won't be configured to load software to the correct place in the board's memory. You may see an error stating that **libexpat-dev** has an unmet dependency: the wrong version of another library **libexpat1**. In that case, use this instead to install with the correct version of libexpat1:

```
sudo apt-get install libexpat1=2.0.1-7.2ubuntu1 libexpat-dev
```

Step 4: Execute the Makefile.

Navigate to **/home/USERNAME/arm-eabi-toolchain** and execute the makefile with *sudo make install-cross*:

```
cd ~/arm-eabi-toolchain
sudo make install-cross
```



```
wes@ubuntu:~$ cd ~/arm-eabi-toolchain
wes@ubuntu:~/arm-eabi-toolchain$ sudo make install-cross
sudo -u wes curl -LO https://sourcery.mentor.com/GNUToolchain/package11441/publi
c/arm-none-eabi/arm-2013.05-23-arm-none-eabi.src.tar.bz2
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0   372    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  6  156M    6 10.5M    0     0  1548k      0  0:01:43  0:00:06  0:01:37 1796k
```

The script downloads the source code for the toolchain, compiles them, and installs.

Step 5: Wait.

The makefile will take a long time to run - possibly upwards of an hour. When it finishes, check back over the last several lines in the terminal window to make sure there were no errors. If everything went correctly, you should now have a folder in your Home directory called **arm-cs-tools**.

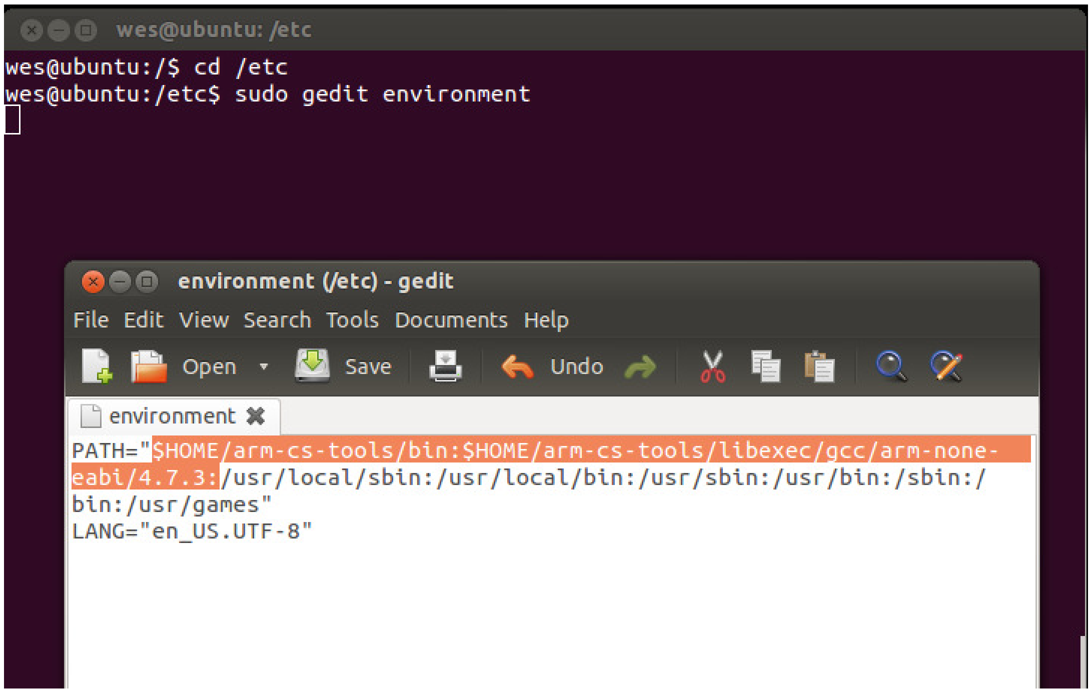
Step 6: Add the Cross-Compiler Tools to PATH.

In a terminal window, navigate to **/etc** and open the file called **environment** in a text editor. We'll use gedit, the text editor installed by default on Ubuntu.

```
cd /etc
sudo gedit environment
```

The first line of **environment** is a list of directories Ubuntu uses to search for executables. Add the following text to the beginning of the list, right after the open-quote:

```
$HOME/arm-cs-tools/bin:$HOME/arm-cs-tools/libexec/gcc/arm-none-eabi/4.7.3:
```


wes@ubuntu: /etc
environment (/etc) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
environment

```
wes@ubuntu:/$ cd /etc
wes@ubuntu:/etc$ sudo gedit environment
```

```
PATH="$HOME/arm-cs-tools/bin:$HOME/arm-cs-tools/libexec/gcc/arm-none-eabi/4.7.3:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games"
LANG="en_US.UTF-8"
```

Save the file and exit. Now back in the terminal window, update your PATH by typing:

```
source /etc/environment
```

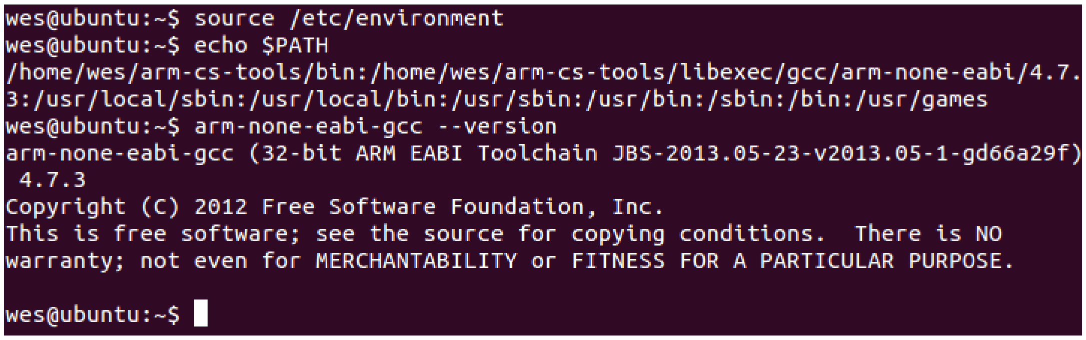
The directories containing the cross-compiler executables should now be in your PATH. You can verify by typing:

```
echo $PATH
```

To make sure Ubuntu can find the cross-compiler, type:

```
arm-none-eabi-gcc --version
```

You should see a printout with version information.



```
wes@ubuntu:~$ source /etc/environment
wes@ubuntu:~$ echo $PATH
/home/wes/arm-cs-tools/bin:/home/wes/arm-cs-tools/libexec/gcc/arm-none-eabi/4.7.
3:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
wes@ubuntu:~$ arm-none-eabi-gcc --version
arm-none-eabi-gcc (32-bit ARM EABI Toolchain JBS-2013.05-23-v2013.05-1-gd66a29f)
 4.7.3
Copyright (C) 2012 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.

wes@ubuntu:~$ 
```

Remember this Later...

You may get a "Command not found" error when trying to run part of the toolchain. When you restart your computer, your path may change. You can fix this by resetting PATH from the file you edited earlier:

```
source /etc/environment
```

Also, the *sudo* command changes your PATH. It won't matter for the rest of this guide, but if you ever need to access part of the toolchain with *sudo* you can do so like this:

```
sudo env PATH=$PATH command_here
```

Install and Configure Eclipse

Step 1: Install Java.

Eclipse can be used to develop programs for multiple languages, but the Eclipse software itself runs in Java. If you're not sure whether you have Java installed already, open a terminal window and type:

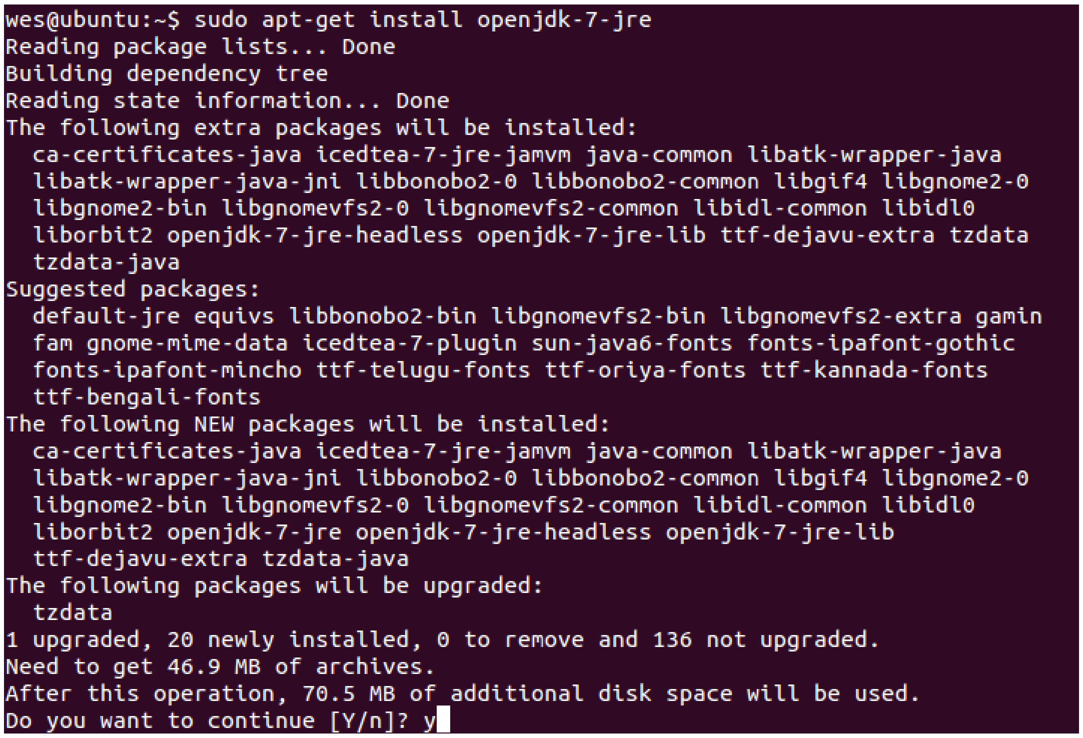
```
java --version
```

If you have a Java Runtime Environment (JRE) installed, you will see the name of the distribution and a version number. *Note for users unfamiliar with Java development: The Java Runtime Environment (JRE) and the Java plugin that runs applets in your web browser are two different things. What we need is the JRE.*

To install the JRE, type in a terminal window:

```
sudo apt-get install openjdk-7-jre
```

Press 'Enter' on your keyboard, then 'y' when prompted.



```
wes@ubuntu:~$ sudo apt-get install openjdk-7-jre
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  ca-certificates-java icedtea-7-jre-jamvm java-common libatk-wrapper-java
  libatk-wrapper-java-jni libbonobo2-0 libbonobo2-common libgif4 libgnome2-0
  libgnome2-bin libgnomevfs2-0 libgnomevfs2-common libidl-common libidl0
  liborbit2 openjdk-7-jre-headless openjdk-7-jre-lib ttf-dejavu-extra tzdata
  tzdata-java
Suggested packages:
  default-jre equivs libbonobo2-bin libgnomevfs2-bin libgnomevfs2-extra gamin
  fam gnome-mime-data icedtea-7-plugin sun-java6-fonts fonts-ipafont-gothic
  fonts-ipafont-mincho ttf-telugu-fonts ttf-oriya-fonts ttf-kannada-fonts
  ttf-bengali-fonts
The following NEW packages will be installed:
  ca-certificates-java icedtea-7-jre-jamvm java-common libatk-wrapper-java
  libatk-wrapper-java-jni libbonobo2-0 libbonobo2-common libgif4 libgnome2-0
  libgnome2-bin libgnomevfs2-0 libgnomevfs2-common libidl-common libidl0
  liborbit2 openjdk-7-jre openjdk-7-jre-headless openjdk-7-jre-lib
  ttf-dejavu-extra tzdata-java
The following packages will be upgraded:
  tzdata
1 upgraded, 20 newly installed, 0 to remove and 136 not upgraded.
Need to get 46.9 MB of archives.
After this operation, 70.5 MB of additional disk space will be used.
Do you want to continue [Y/n]? y
```

Once installation completes you can verify that everything installed correctly by typing **java --version** again.

```
java --version
```

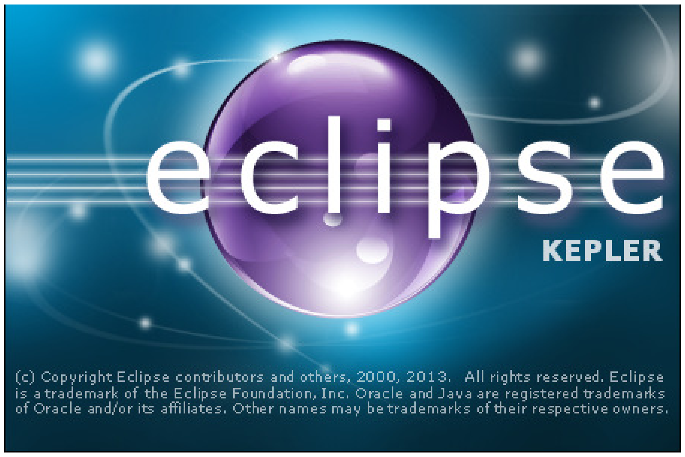
Step 2: Download Eclipse.

Go to http://www.eclipse.org/downloads in your browser. Download the **Eclipse IDE for C/C++ Developers.** There are different versions for 32- and

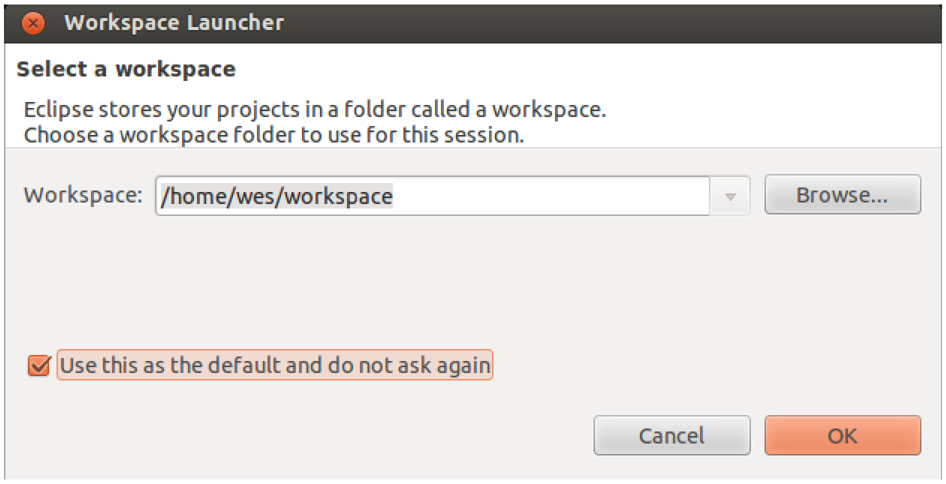
64-bit Linux; download the one appropriate to your computer's operating system.

Open the archive and extract it to your **Home folder**.

Doubleclick the **eclipse** executable to start the IDE. You should see this loading screen:


eclipse
KEPLER
(c) Copyright Eclipse contributors and others, 2000, 2013. All rights reserved. Eclipse is a trademark of the Eclipse Foundation, Inc. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Next Eclipse will prompt you to select a workspace:


Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

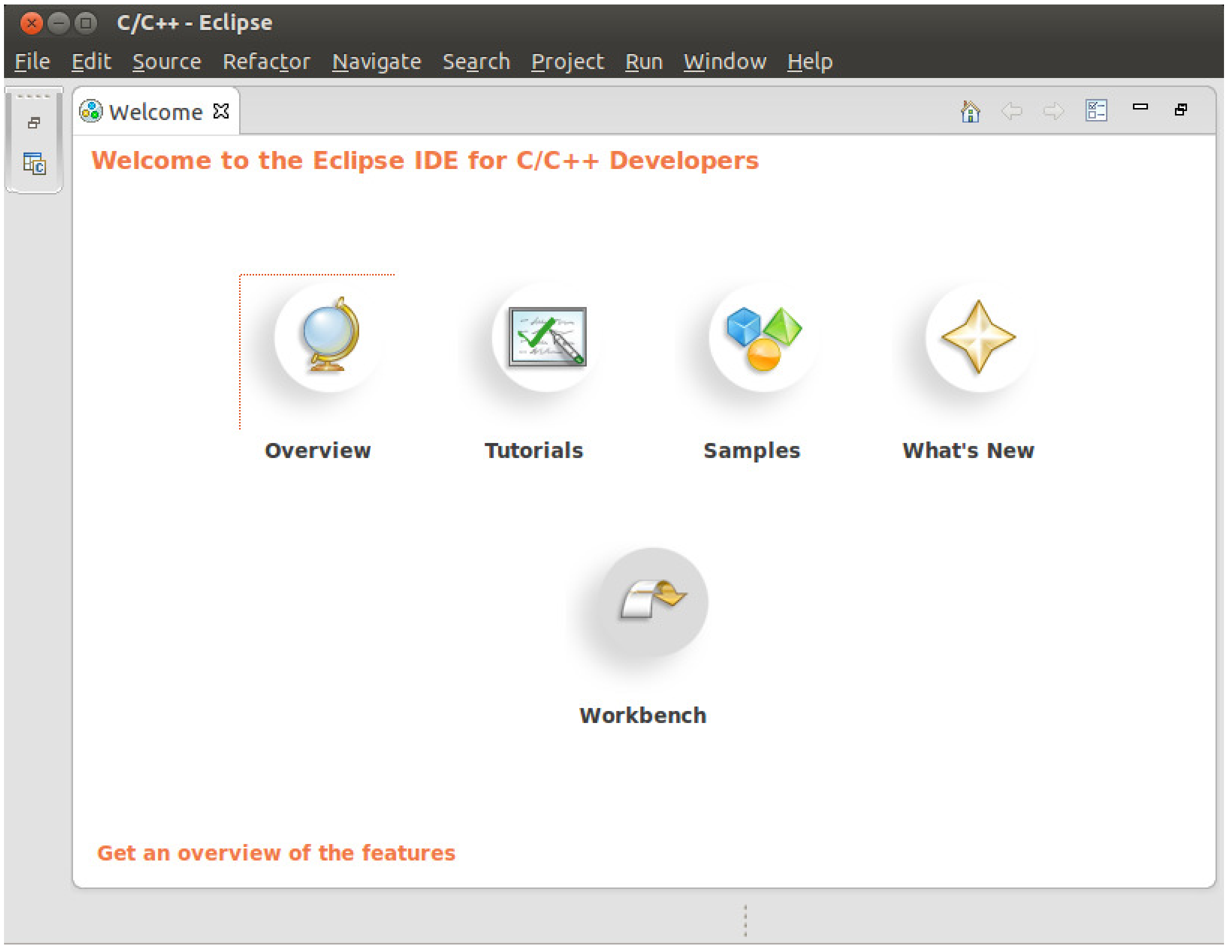
Workspace: /home/wes/workspace

[✓] Use this as the default and do not ask again

This will create a new directory to store your Eclipse projects. This guide will assume you use the default **/home/USERNAME/workspace**. Click OK.

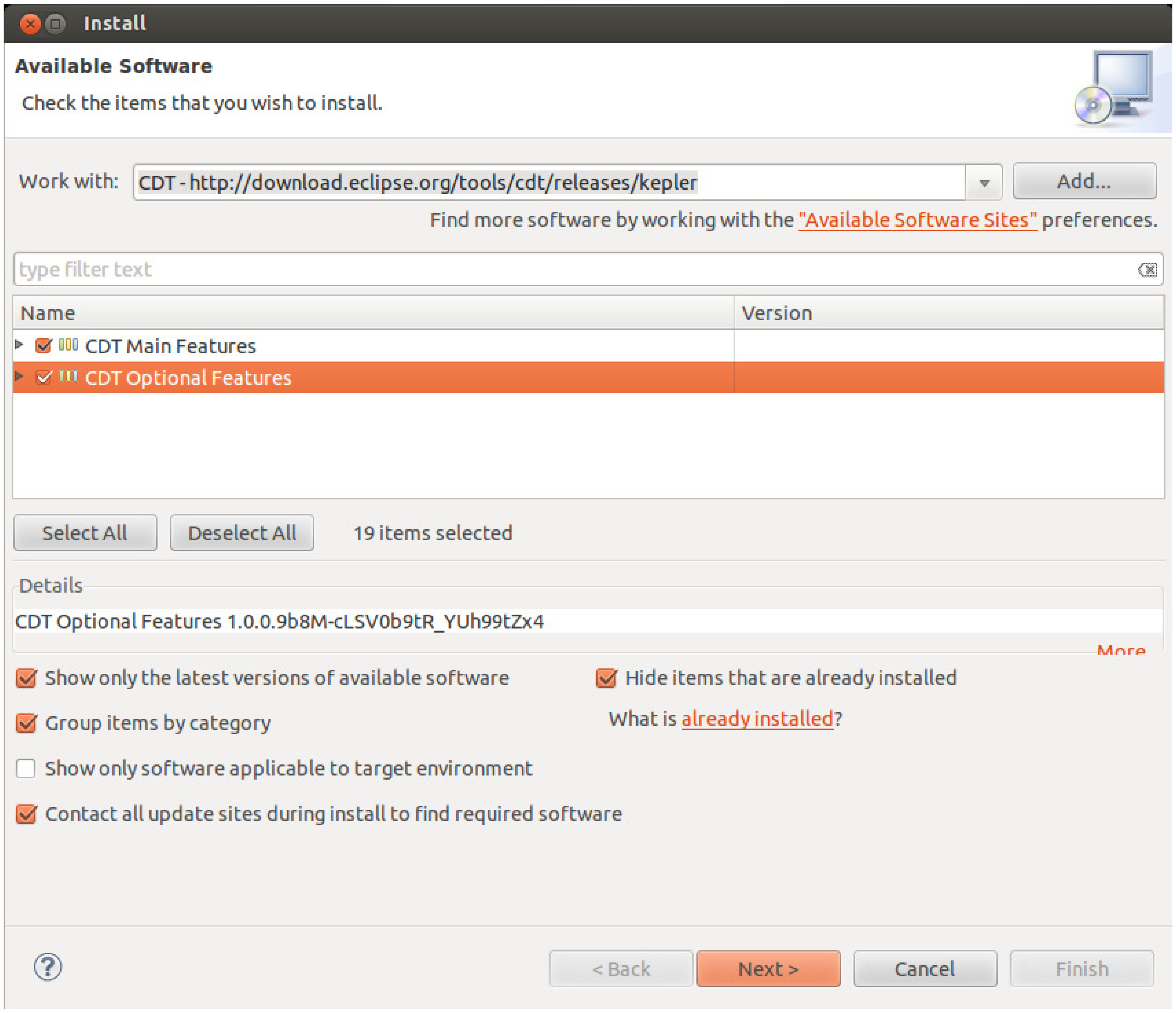
Step 3: Install the Eclipse CDT Plugin.

You should now be at the Eclipse welcome screen.


C/C++ - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
Welcome to the Eclipse IDE for C/C++ Developers
Overview
Tutorials
Samples
What's New
Workbench
Get an overview of the features

At the top of the window, go to **Help > Install New Software...** to bring up the Install window. In the text bar at the top of the window, enter the following URL:

```
http://download.eclipse.org/tools/cdt/releases/kepler
```


Install
Available Software
Check the items that you wish to install.
Work with:
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
Select All
Deselect All
Details
More
What is already installed?
< Back
Next >
Cancel
Finish

Work with: CDT - http://download.eclipse.org/tools/cdt/releases/kepler

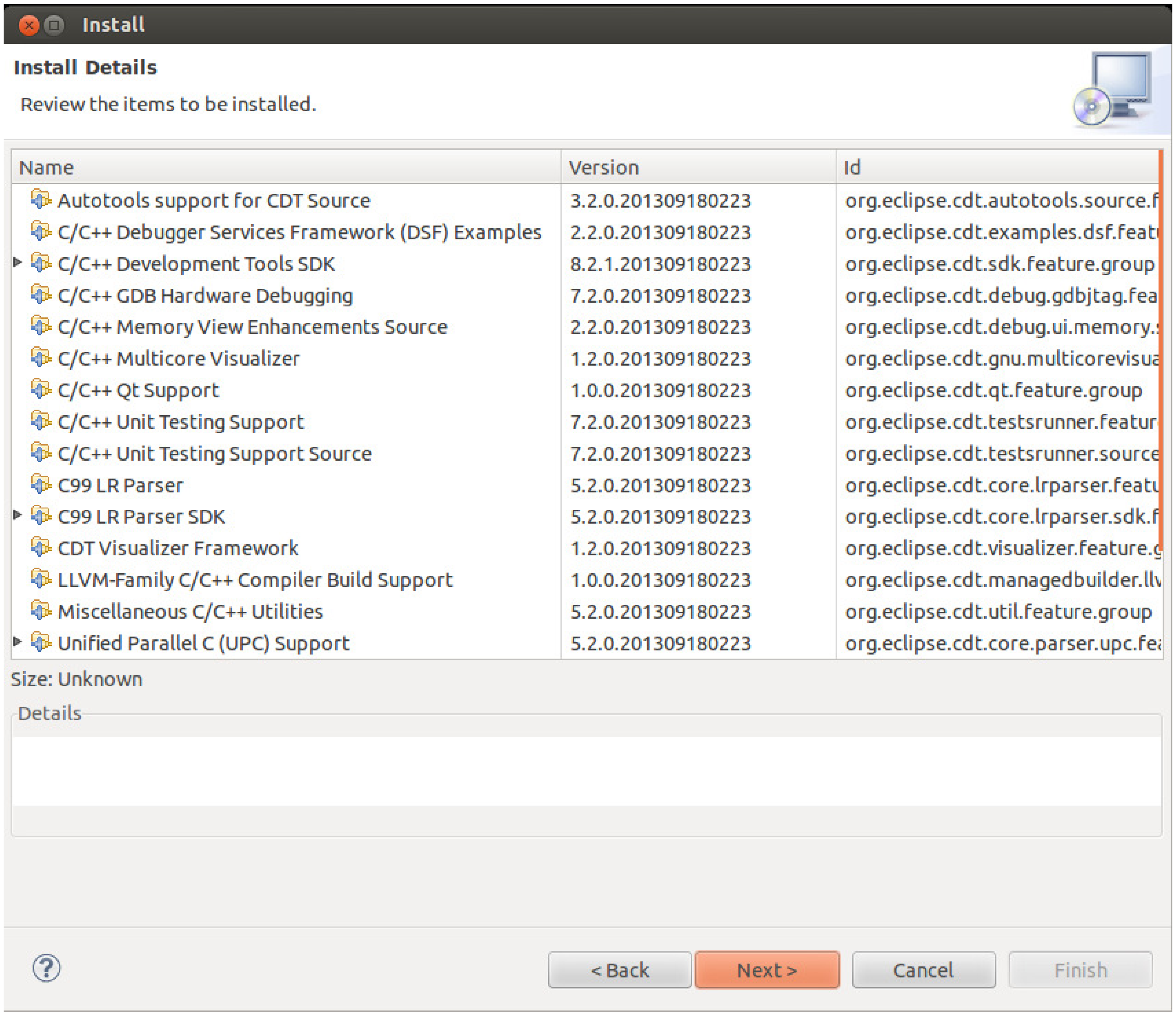
Name	Version
✓ CDT Main Features	
✓ CDT Optional Features	

19 items selected

CDT Optional Features 1.0.0.9b8M-cLSV0b9tR_YUh99tZx4

- [✓] Show only the latest versions of available software
- [✓] Hide items that are already installed
- [✓] Group items by category
- [ ] Show only software applicable to target environment
- [✓] Contact all update sites during install to find required software

You should see two options: **CDT Main Features** and **CDT Optional Features**. Click the checkboxes next to each and click Next to go to a list of features to be installed:



Install

Install Details

Review the items to be installed.

Name	Version	Id
Autotools support for CDT Source	3.2.0.201309180223	org.eclipse.cdt.autotools.source.f
C/C++ Debugger Services Framework (DSF) Examples	2.2.0.201309180223	org.eclipse.cdt.examples.dsf.feat
C/C++ Development Tools SDK	8.2.1.201309180223	org.eclipse.cdt.sdk.feature.group
C/C++ GDB Hardware Debugging	7.2.0.201309180223	org.eclipse.cdt.debug.gdbjtag.fea
C/C++ Memory View Enhancements Source	2.2.0.201309180223	org.eclipse.cdt.debug.ui.memory.:
C/C++ Multicore Visualizer	1.2.0.201309180223	org.eclipse.cdt.gnu.multicorevisua
C/C++ Qt Support	1.0.0.201309180223	org.eclipse.cdt.qt.feature.group
C/C++ Unit Testing Support	7.2.0.201309180223	org.eclipse.cdt.testsrunner.featur
C/C++ Unit Testing Support Source	7.2.0.201309180223	org.eclipse.cdt.testsrunner.source
C99 LR Parser	5.2.0.201309180223	org.eclipse.cdt.core.lrparser.featu
C99 LR Parser SDK	5.2.0.201309180223	org.eclipse.cdt.core.lrparser.sdk.f
CDT Visualizer Framework	1.2.0.201309180223	org.eclipse.cdt.visualizer.feature.g
LLVM-Family C/C++ Compiler Build Support	1.0.0.201309180223	org.eclipse.cdt.managedbuilder.llv
Miscellaneous C/C++ Utilities	5.2.0.201309180223	org.eclipse.cdt.util.feature.group
Unified Parallel C (UPC) Support	5.2.0.201309180223	org.eclipse.cdt.core.parser.upc.fea

Size: Unknown

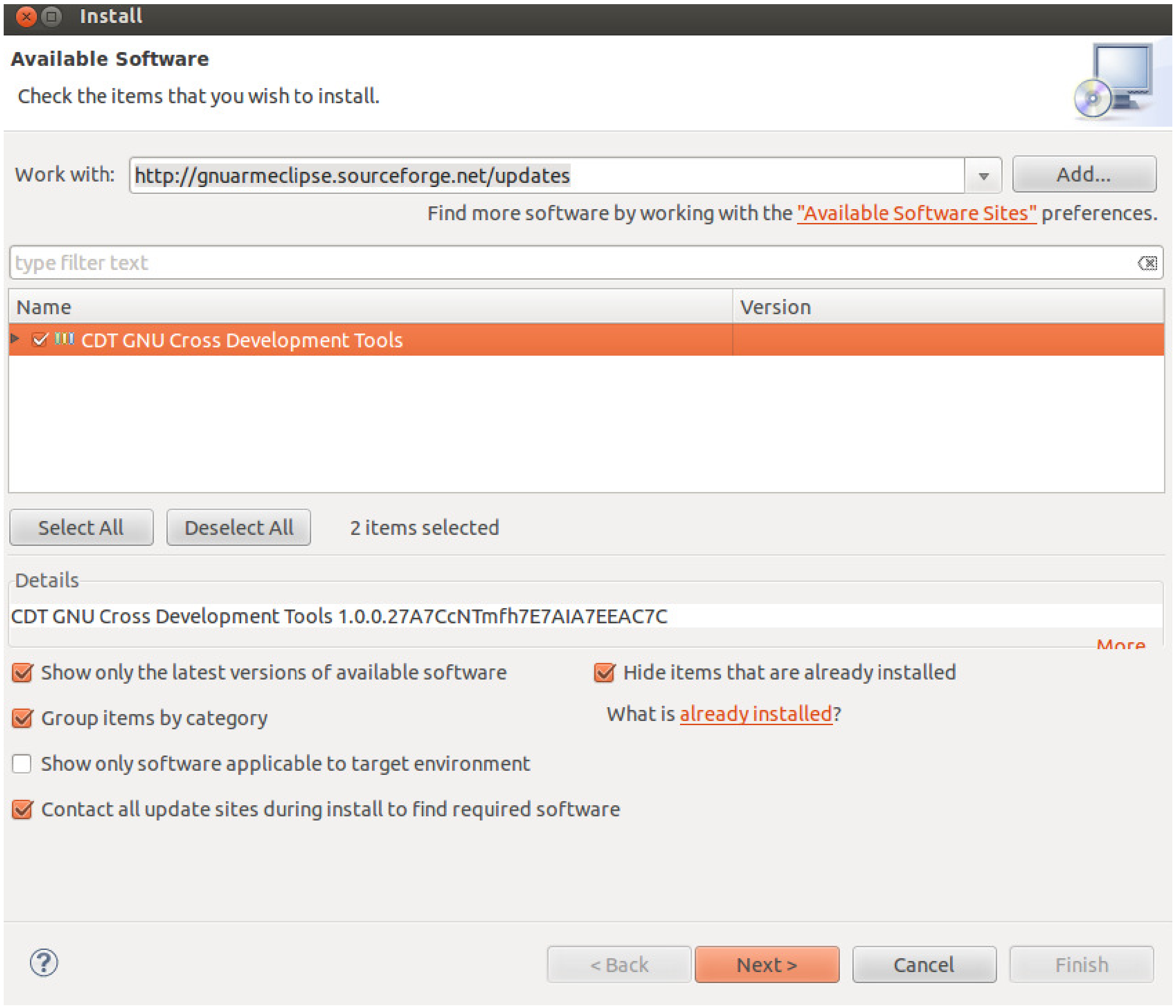
Details

Click Next again to install.

Step 4: Install Cross Development Tools.

In the Eclipse window, go again to **Help > Install New Software...** This time in the text bar at the top of the window, enter:

```
http://gnuarmeclipse.sourceforge.net/updates
```


Install
Available Software
Check the items that you wish to install.
Work with:
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
Select All
Deselect All
Details
More
What is already installed?
< Back
Next >
Cancel
Finish

Work with: http://gnuarmeclipse.sourceforge.net/updates

☑ CDT GNU Cross Development Tools

2 items selected

CDT GNU Cross Development Tools 1.0.0.27A7CcNTmfh7E7AIA7EEAC7C

- ☑ Show only the latest versions of available software
- ☑ Hide items that are already installed
- ☑ Group items by category
- ☐ Show only software applicable to target environment
- ☑ Contact all update sites during install to find required software

Check the box net to **CDT GNU Cross Development Tools**, hit Next, and then Next again to install.

Create a Simple Eclipse Project

Step 1: Download bbone_ledblink.

The bbone_ledblink program will repeatedly blink the outermost blue LED on the BeagleBone Black. Download it here:

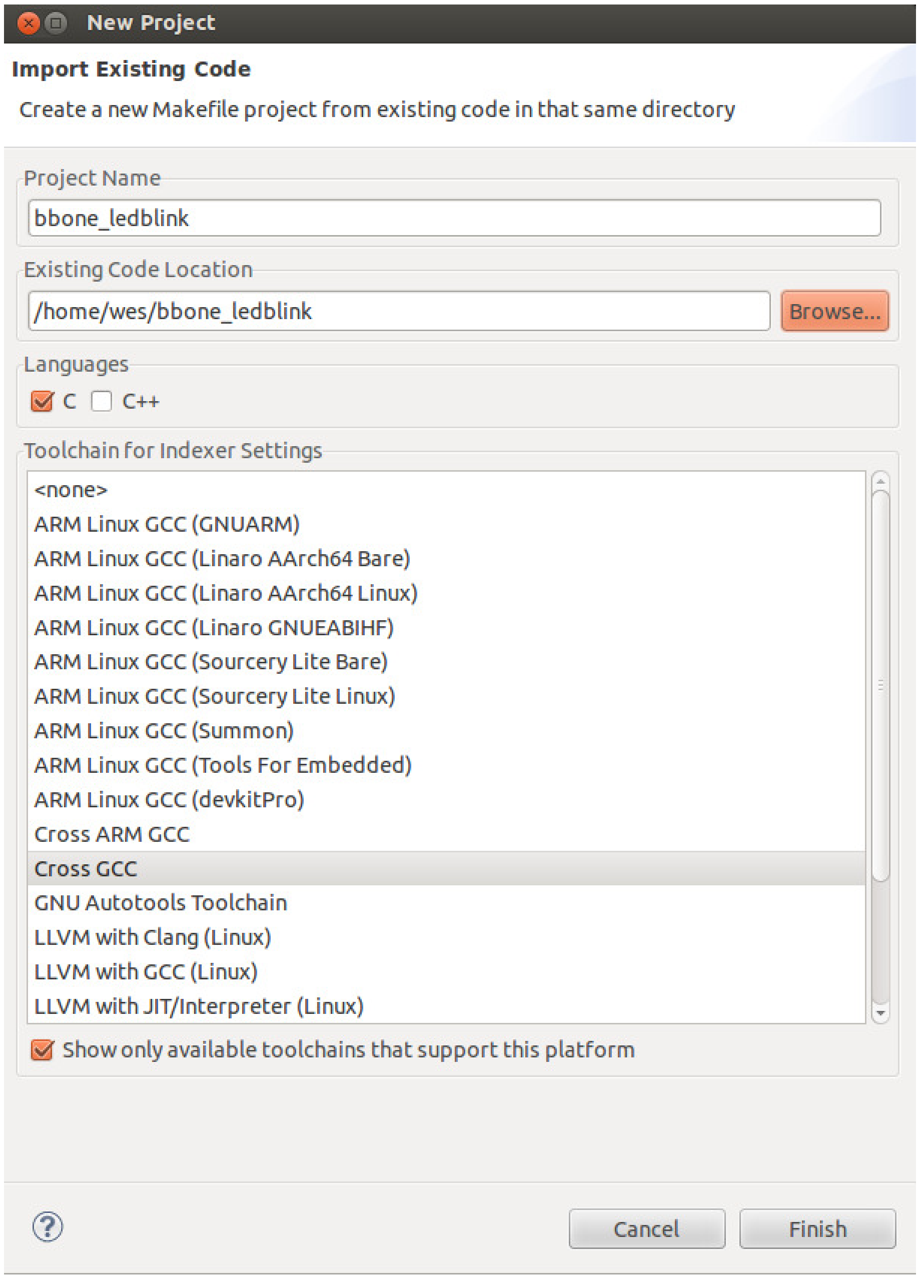
Download bbone_ledblink

Open the archive and extract the contents to your Home folder.

Step 2: Create a New Eclipse Project.

In Eclipse, look at the menu at the top of the screen. Select **New > Makefile Project with Existing Code**. The New Project window should appear. Configure your new project as follows:

- Give the project a name.
- In Existing Code Location, browse to **/home/USERNAME/bbone_ledblink** and click OK.
- Check **C** and uncheck **C++** under Languages.
- Under Toolchain for Indexer Settings, select **Cross GCC.**



Import Existing Code

Create a new Makefile project from existing code in that same directory

Project Name: bbone_ledblink

Existing Code Location: /home/wes/bbone_ledblink

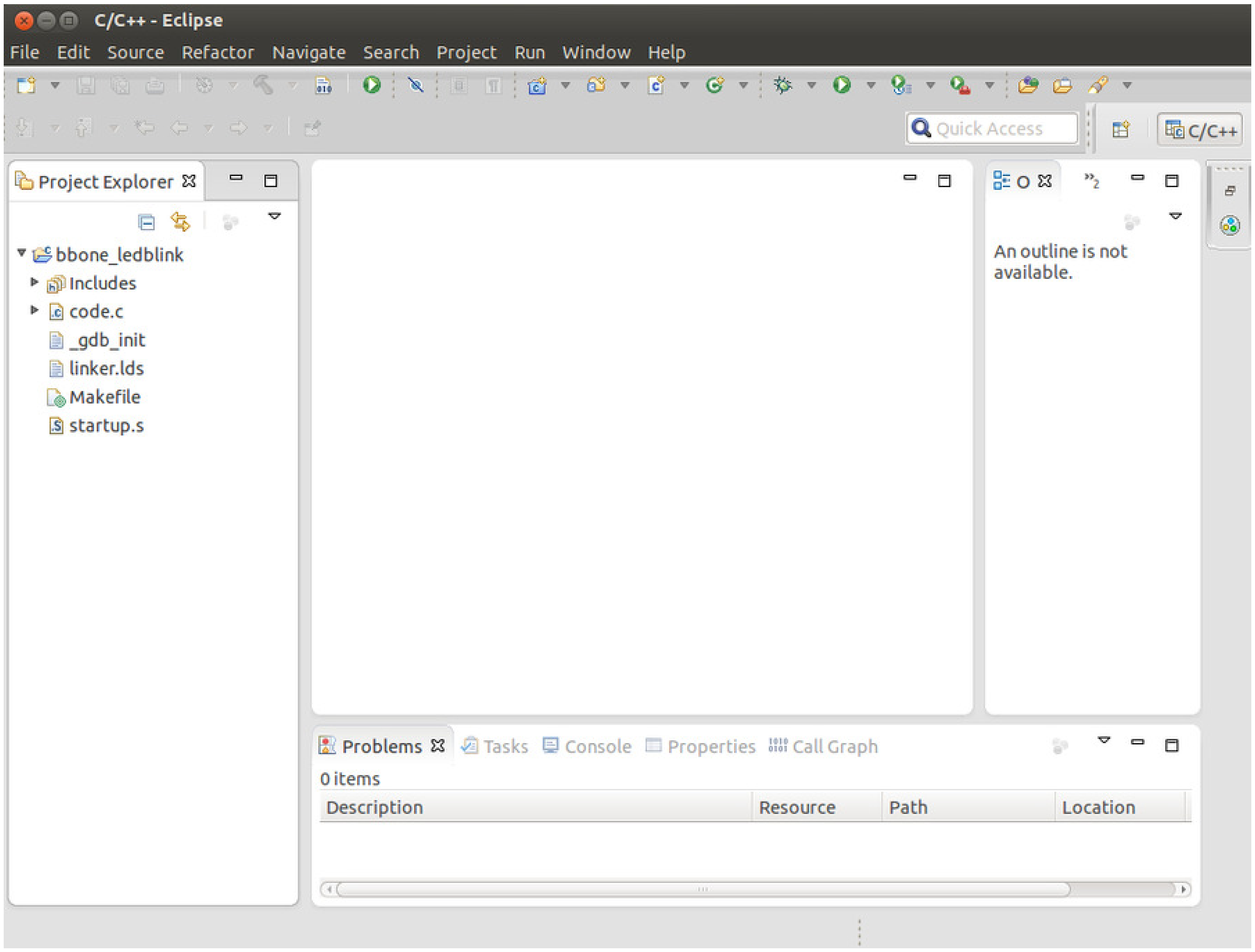
Languages: [✓ C] C++

Toolchain for Indexer Settings:

<none>
ARM Linux GCC (GNUARM)
ARM Linux GCC (Linaro AArch64 Bare)
ARM Linux GCC (Linaro AArch64 Linux)
ARM Linux GCC (Linaro GNUEABIHF)
ARM Linux GCC (Sourcery Lite Bare)
ARM Linux GCC (Sourcery Lite Linux)
ARM Linux GCC (Summon)
ARM Linux GCC (Tools For Embedded)
ARM Linux GCC (devkitPro)
Cross ARM GCC
[✓ Cross GCC]
GNU Autotools Toolchain
LLVM with Clang (Linux)
LLVM with GCC (Linux)
LLVM with JIT/Interpreter (Linux)

[✓ Show only available toolchains that support this platform]

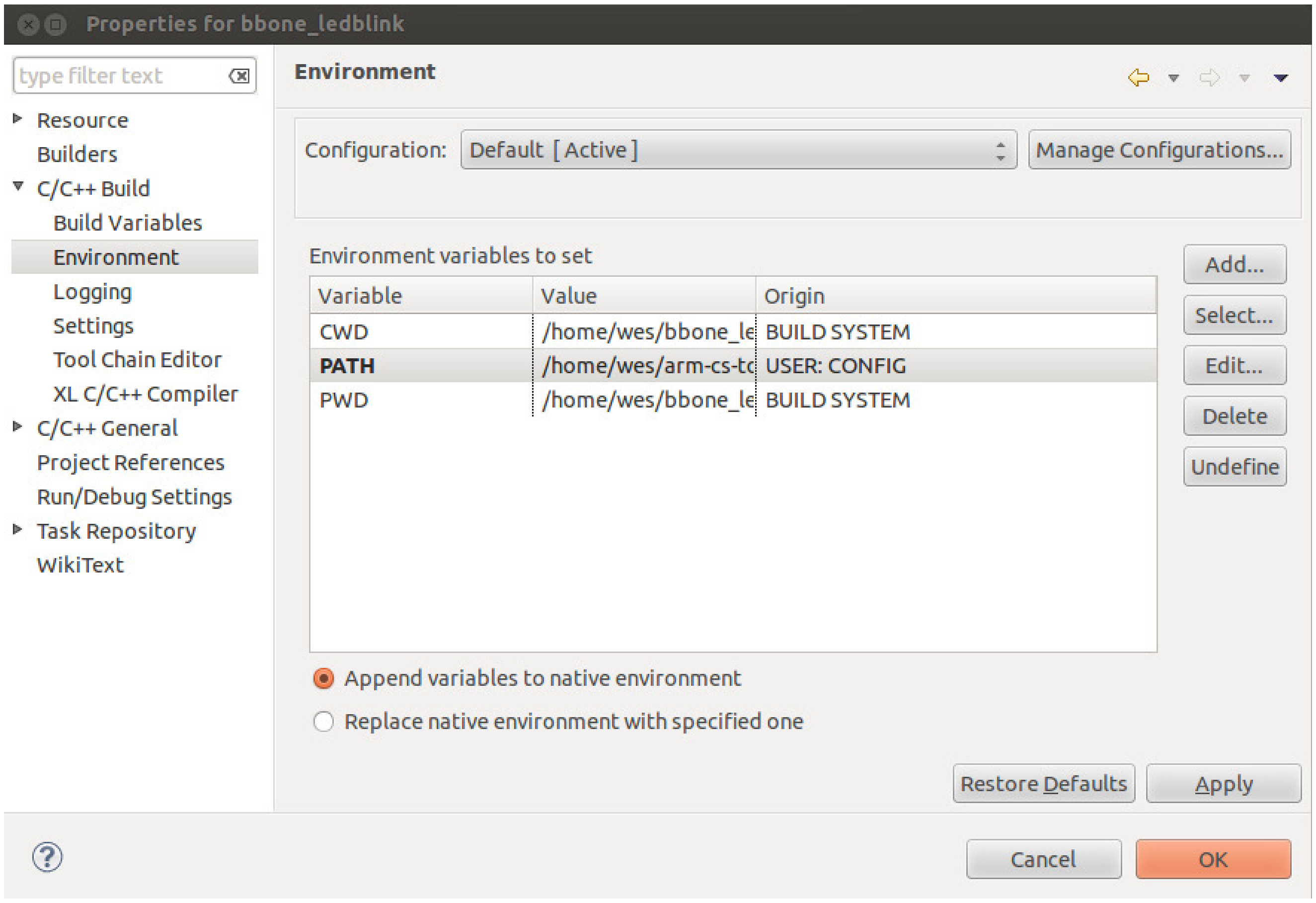
Click Finish to create your project. You can now view your project heirarchy in the Eclipse Project Explorer window (on the left in the picture below).



Step 3: Set your Project's PATH Variable.

Earlier, we set the PATH variable so your operating system could find the ARM EABI toolchain executables. When Eclipse creates a new project it imports your PATH and uses it by default. Just one problem: Eclipse can't parse the $HOME variable in your PATH, so you'll have to edit your project settings.

In the Project Explorer window, left-click on the top level of your project to select it. Then in the menu at the top, go to **Project > Properties**. Expand **C/C++ Build** and select **Environment.**


Properties for bbone_ledblink

Environment

Configuration: Default [Active]

Environment variables to set

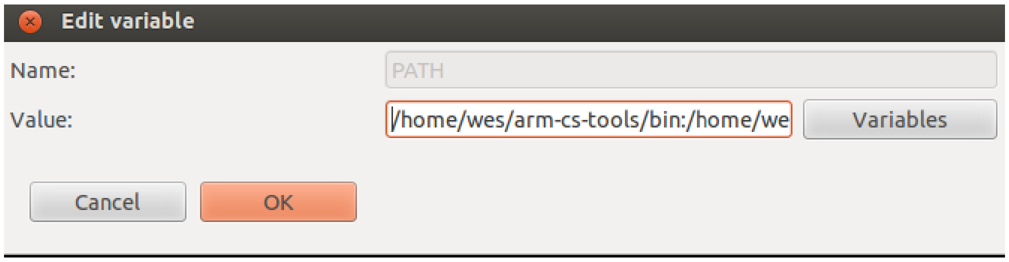
Variable	Value	Origin
CWD	/home/wes/bbone_le	BUILD SYSTEM
PATH	/home/wes/arm-cs-tc	USER: CONFIG
PWD	/home/wes/bbone_le	BUILD SYSTEM

[✓ Append variables to native environment] Replace native environment with specified one

Select **PATH** and click Edit to bring up the Edit Variable window. Add the following text to the beginning of the text box:

```
/home/USERNAME/arm-cs-tools/bin:/home/USERNAME/arm-cs-tools/libexec/gcc/arm-none-eabi/4.7.3:
```

...but replace *USERNAME* with your Linux username. Don't use **$HOME** or the tilde (~) symbol in place of /home/USERNAME.


Edit variable

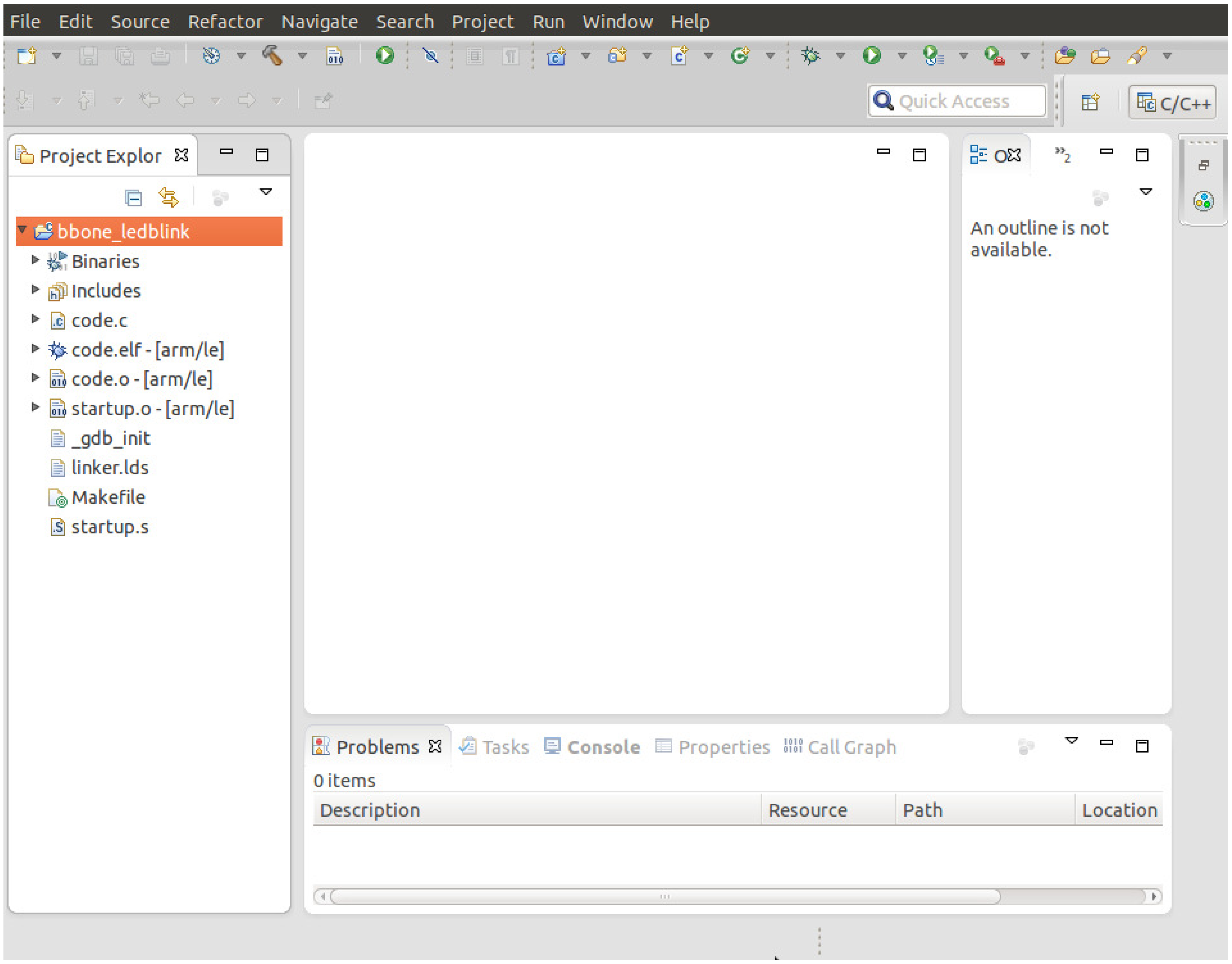
Name: PATH

Value: /home/wes/arm-cs-tools/bin:/home/we

Click OK. Then back in the Properties window, click Apply then OK.

Step 4: Build your Project.

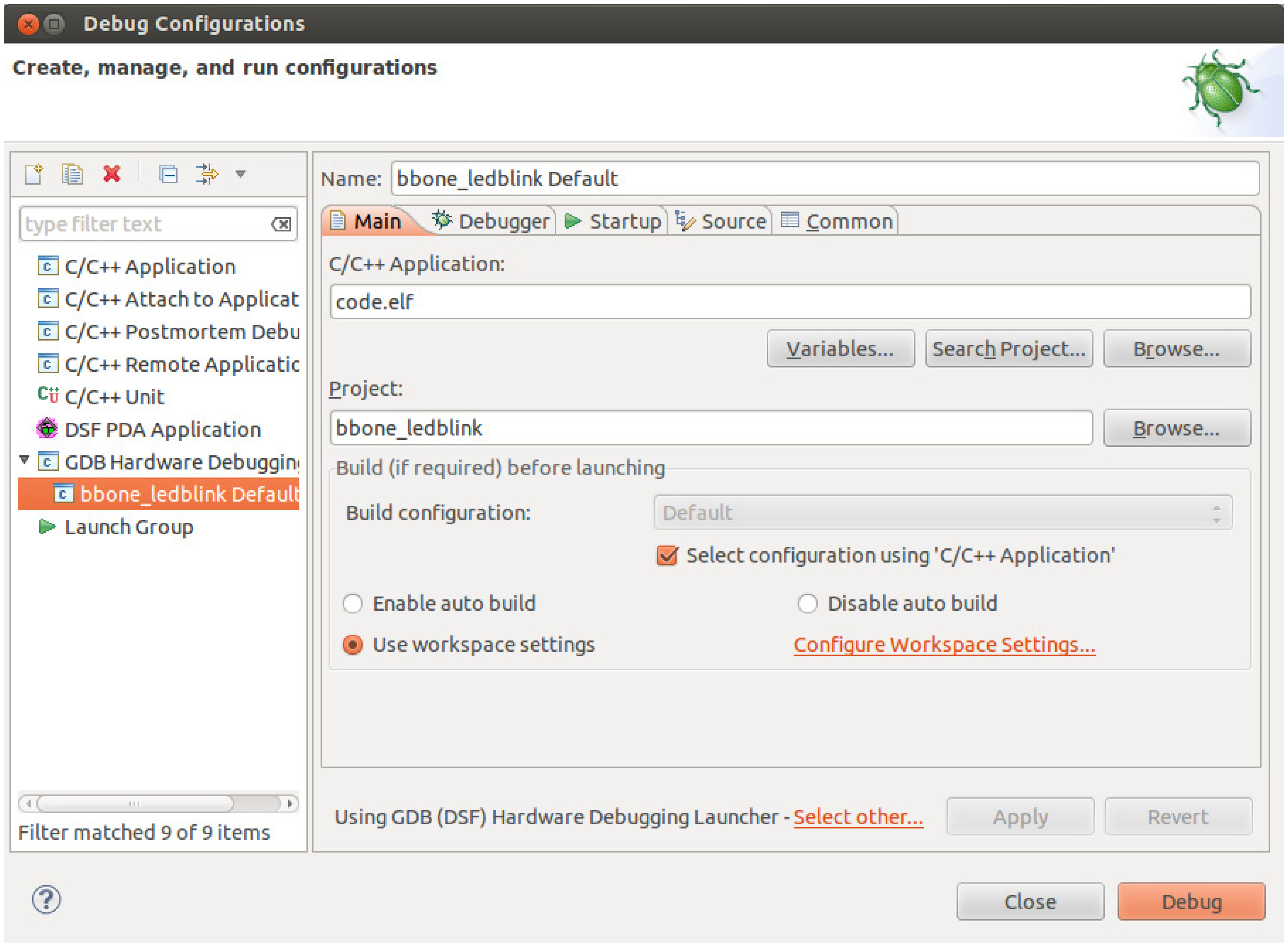
In the top menu, go to **Project > Build All**. If the project compiles successfully you should see new object files appear in the Project Explorer.



Step 5: Set your Project's Debug Configuration.

Back on the Eclipse main screen, click on the top level of your project again to select it. Then in the top menu go to **Run > Debug Configurations**. A new window should appear. In the menu on the left, double-click **GDB Hardware Debugging** and select **bbone_ledblink Default** (or whatever you named your project).

To the right, look under **Project:** The text field should show the name of your project. If it doesn't, click **Browse...** and select your project. Then under **C/C++ Application**, make sure it lists the name of your executable, **code.elf.** If it doesn't, click **Search Project...** and select **code.elf.**



Name: bbone_ledblink Default

C/C++ Application: code.elf

Project: bbone_ledblink

Build (if required) before launching

Build configuration: Default

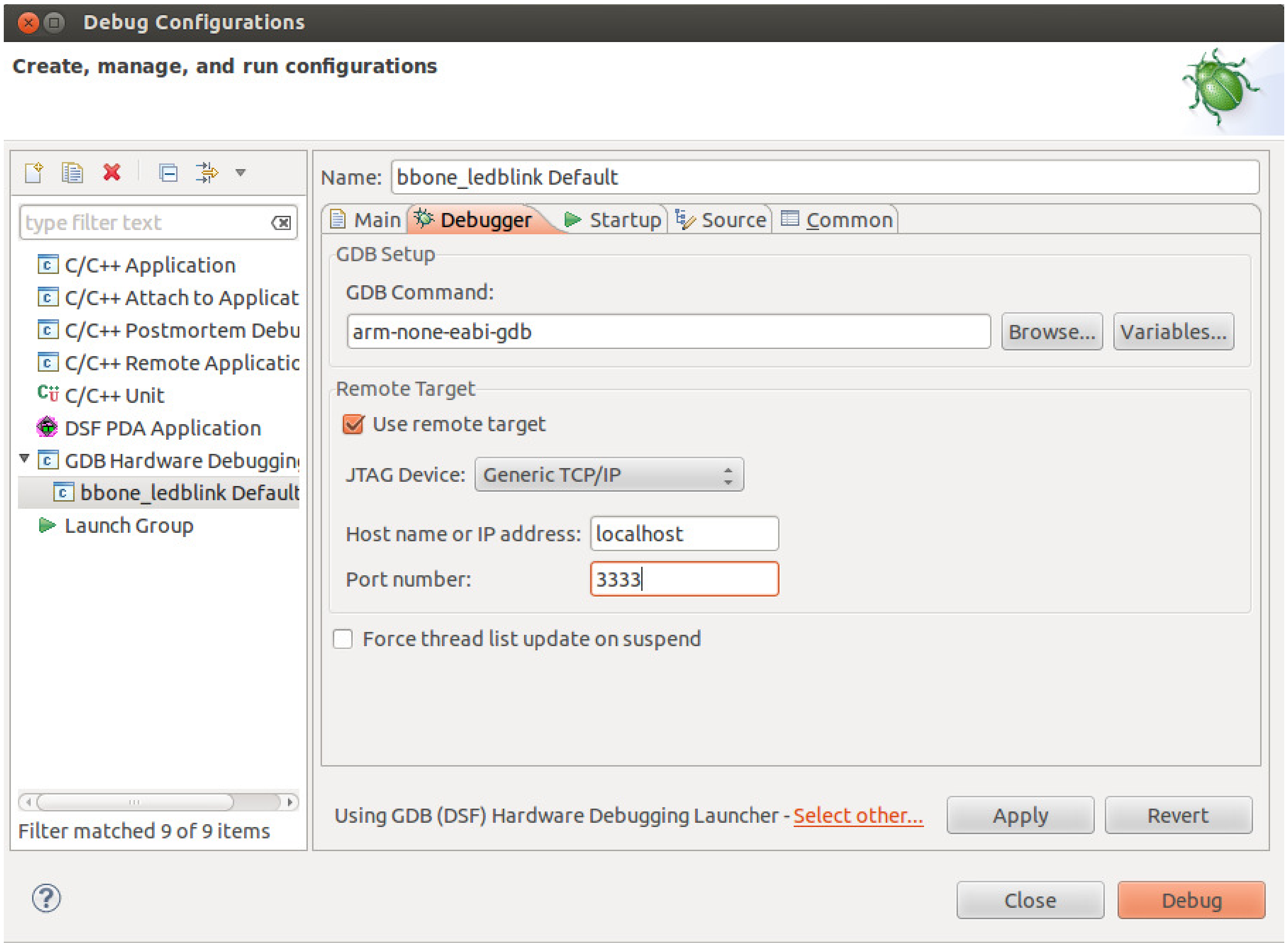
[✓] Select configuration using 'C/C++ Application'

Enable auto build / Disable auto build / [✓ Use workspace settings]

Using GDB (DSF) Hardware Debugging Launcher

Next, click on the **Debugger** tab.

- Change *GDB Command* to **arm-none-eabi-gdb**. (It should say *gdb* by default.)
- Change *Port number* to **3333**.


Debug Configurations

Create, manage, and run configurations

Name: bbone_ledblink Default

Main | Debugger | Startup | Source | Common

GDB Setup

GDB Command: arm-none-eabi-gdb

Remote Target

☑ Use remote target

JTAG Device: Generic TCP/IP

Host name or IP address: localhost

Port number: 3333

☐ Force thread list update on suspend

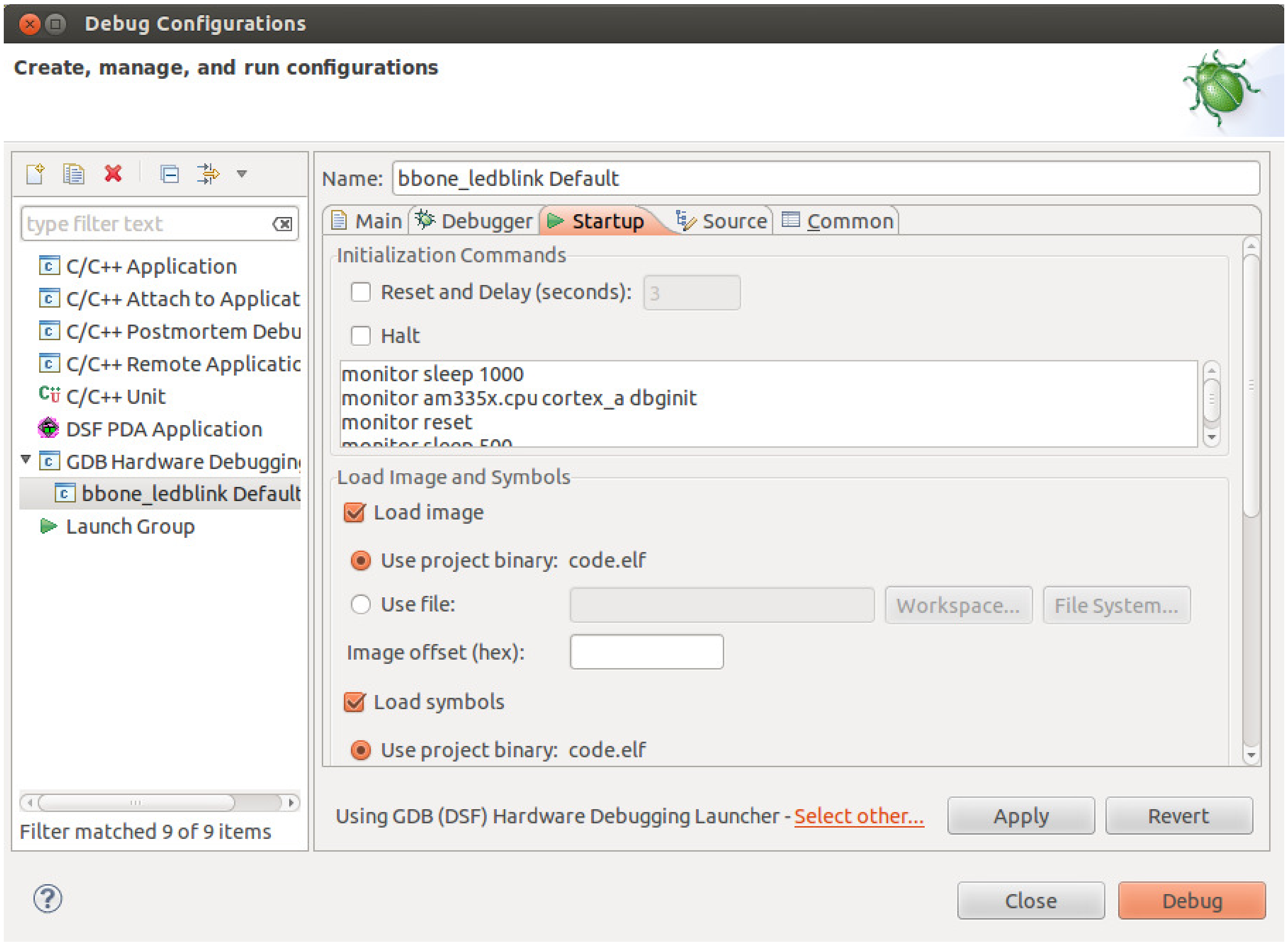
Using GDB (DSF) Hardware Debugging Launcher - Select other...

Now go to the **Startup** tab.

- Uncheck *Reset and Delay*.
- Uncheck *Halt*.
- In the textbox beneath *Reset and Delay* and *Halt*, type the following:

```
monitor sleep 1000
monitor am335x.cpu cortex_a dbginit
monitor reset
monitor sleep 500
monitor halt
monitor sleep 500
```

- Make sure the settings under *Load Image and Symbols* look like the image below. (You shouldn't need to change anything.)


Debug Configurations

Create, manage, and run configurations

Name: bbone_ledblink Default

Startup

Initialization Commands

Reset and Delay (seconds): 3

Halt

```
monitor sleep 1000
monitor am335x.cpu cortex_a dbginit
monitor reset
monitor sleep 500
```

Load Image and Symbols

[✓] Load image

[● Use project binary: code.elf]

Use file:

Image offset (hex):

[✓] Load symbols

[● Use project binary: code.elf]

Using GDB (DSF) Hardware Debugging Launcher - Select other...

Filter matched 9 of 9 items

Click the Apply button, then click Close.

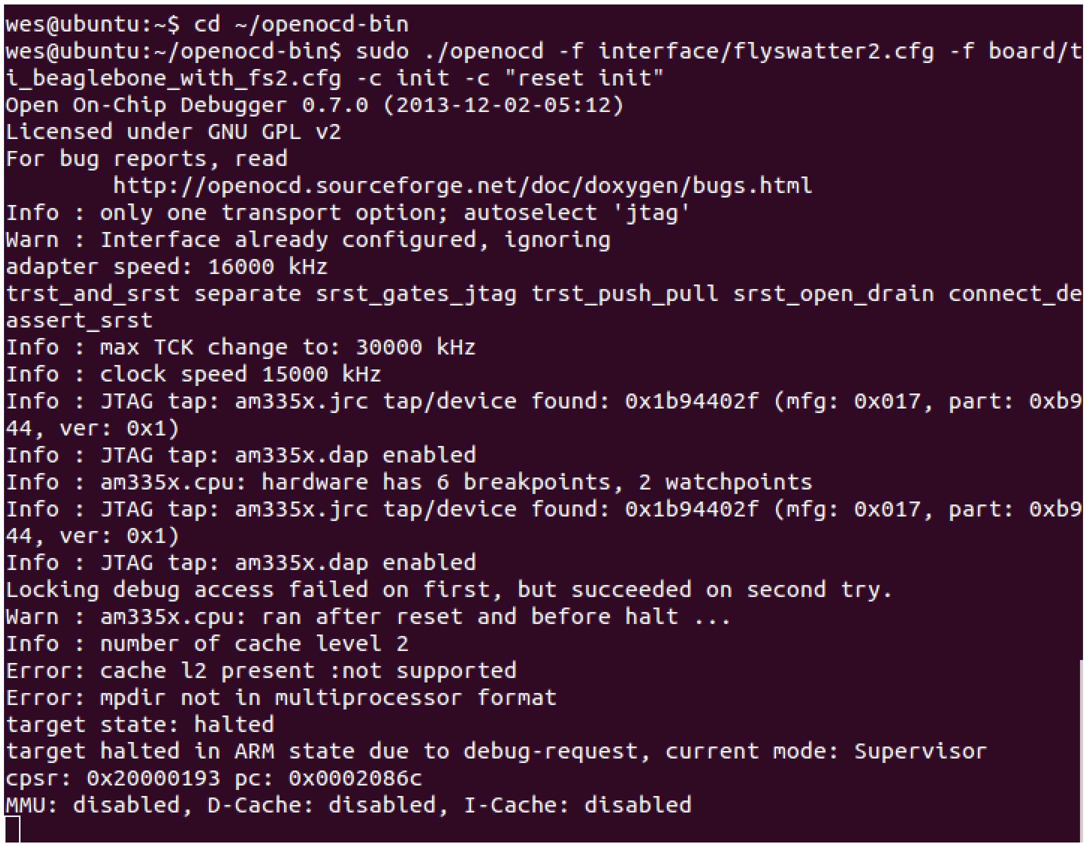
Load and Debug the Program with GDB

For this step, you will need a Flyswatter2, OpenOCD and a configuration file for the BeagleBone Black. If you haven't already installed OpenOCD and downloaded ti_beaglebone_with_fs2.cfg, go to the Flyswatter2 BeagleBone Black How To and follow the instructions there.

Step 1: Start OpenOCD.

Connect your Flyswatter2 and BeagleBone Black. Open a terminal window and run OpenOCD as described in the Flyswatter2 BeagleBone Black How To.

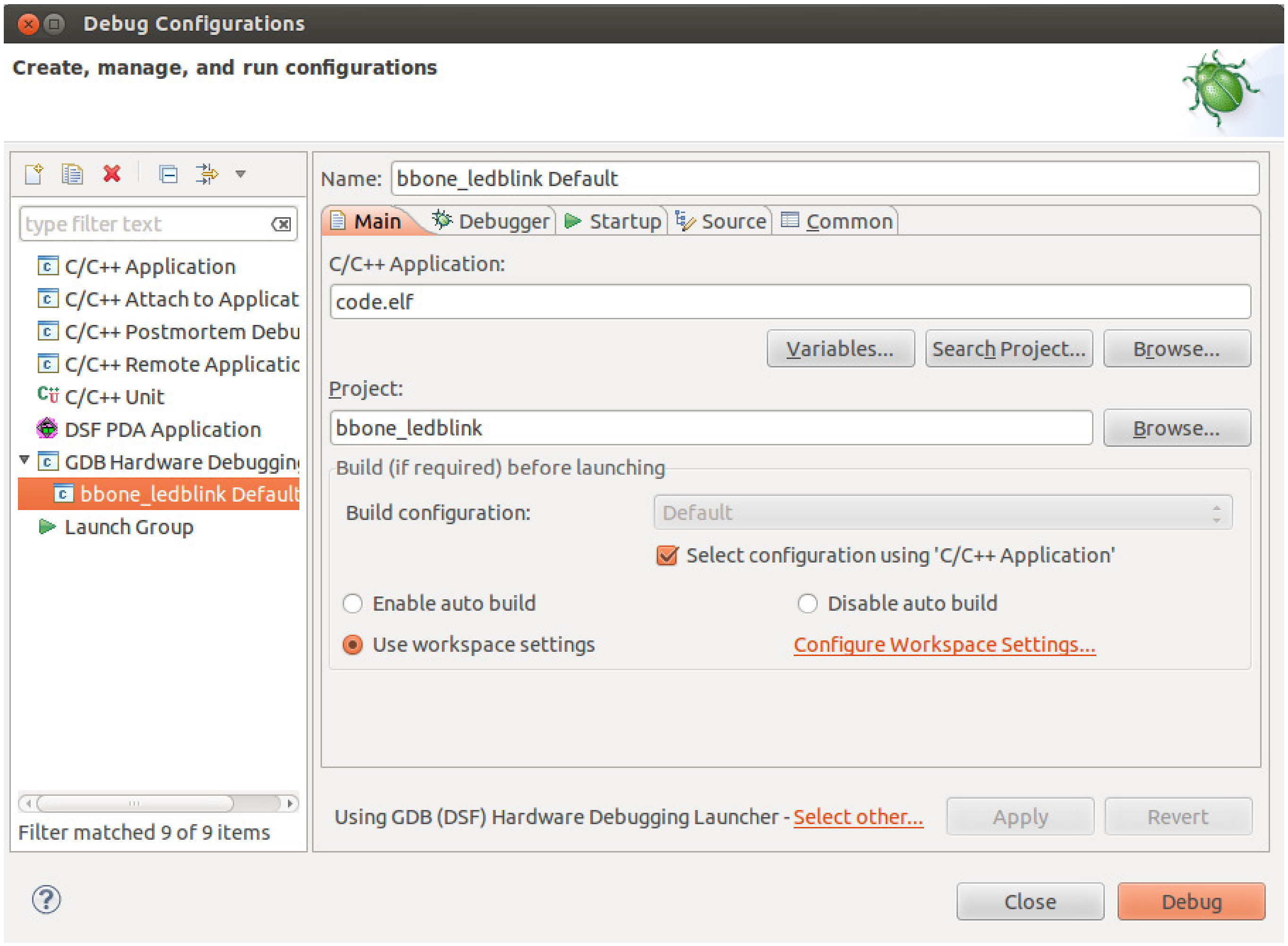
```
cd ~/openocd-bin
sudo ./openocd -f interface/flyswatter2.cfg -f target/ti_beaglebone_with_fs2.cfg -c init -c "reset init"
```



```
wes@ubuntu:~$ cd ~/openocd-bin
wes@ubuntu:~/openocd-bin$ sudo ./openocd -f interface/flyswatter2.cfg -f board/t
i_beaglebone_with_fs2.cfg -c init -c "reset init"
Open On-Chip Debugger 0.7.0 (2013-12-02-05:12)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.sourceforge.net/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
Warn : Interface already configured, ignoring
adapter speed: 16000 kHz
trst_and_srst separate srst_gates_jtag trst_push_pull srst_open_drain connect_de
assert_srst
Info : max TCK change to: 30000 kHz
Info : clock speed 15000 kHz
Info : JTAG tap: am335x.jrc tap/device found: 0x1b94402f (mfg: 0x017, part: 0xb9
44, ver: 0x1)
Info : JTAG tap: am335x.dap enabled
Info : am335x.cpu: hardware has 6 breakpoints, 2 watchpoints
Info : JTAG tap: am335x.jrc tap/device found: 0x1b94402f (mfg: 0x017, part: 0xb9
44, ver: 0x1)
Info : JTAG tap: am335x.dap enabled
Locking debug access failed on first, but succeeded on second try.
Warn : am335x.cpu: ran after reset and before halt ...
Info : number of cache level 2
Error: cache l2 present :not supported
Error: mpdir not in multiprocessor format
target state: halted
target halted in ARM state due to debug-request, current mode: Supervisor
cpsr: 0x20000193 pc: 0x0002086c
MMU: disabled, D-Cache: disabled, I-Cache: disabled
```

Step 2: Start GDB and Load the Program.

Back in Eclipse, select the top level of your project. In the top menu go to **Run > Debug Configurations.** Expand the **GDB Hardware Debugging Tab** and select **bbone_ledblink Default**. You should now be back where you were when you set your debug settings earlier.



Name: bbone_ledblink Default

C/C++ Application: code.elf

Project: bbone_ledblink

Build (if required) before launching

Build configuration: Default

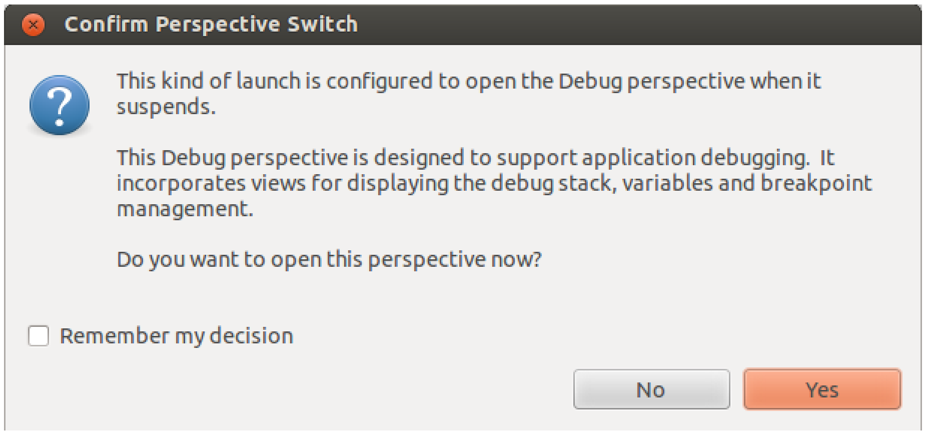
[✓] Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

● Use workspace settings

Using GDB (DSF) Hardware Debugging Launcher

This time, click Debug. You will be prompted to switch to a Debug perspective. Click Yes.



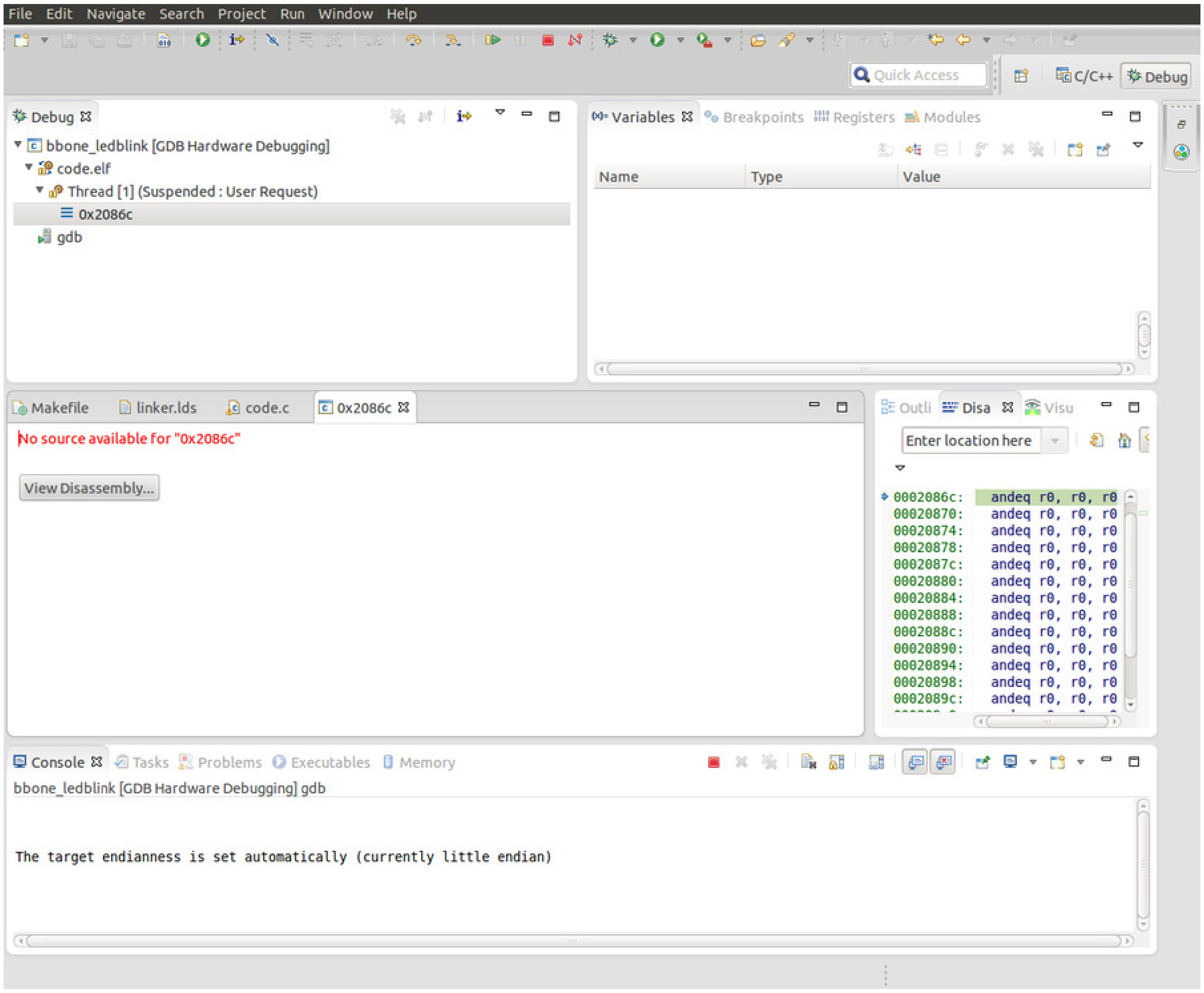
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☐ Remember my decision

This will change Eclipse's layout to show debugging information. (Later you can get back to the old perspective from the top menu: *Window* > *Open Perspective* > *C/C++*.) The Debug perspective looks something like this:



No source available for "0x2086c"

```
0002086c:   andeq r0, r0, r0
00020870:   andeq r0, r0, r0
00020874:   andeq r0, r0, r0
00020878:   andeq r0, r0, r0
0002087c:   andeq r0, r0, r0
00020880:   andeq r0, r0, r0
00020884:   andeq r0, r0, r0
00020888:   andeq r0, r0, r0
0002088c:   andeq r0, r0, r0
00020890:   andeq r0, r0, r0
00020894:   andeq r0, r0, r0
00020898:   andeq r0, r0, r0
0002089c:   andeq r0, r0, r0
```

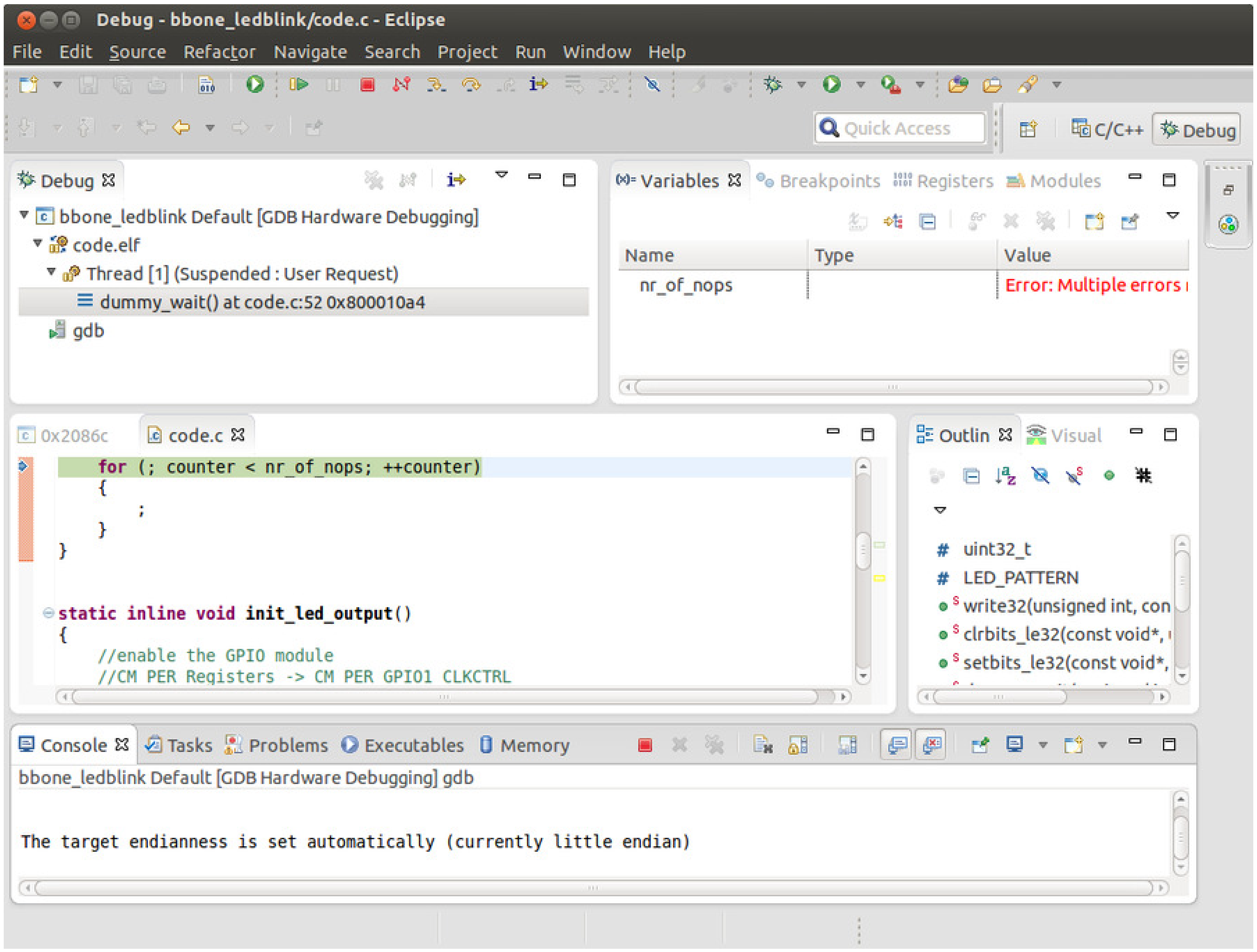
bbone_ledblink [GDB Hardware Debugging] gdb

```
The target endianness is set automatically (currently little endian)
```

You may see an error like the one above - *No source available for "0x2086c"*. If you do, don't worry. It should go away when we resume the program in the next step.

Step 3: Resume the Program.

The LEDblink program should now be in your board's flash memory. However, nothing should be happening yet. In the Eclipse top menu, go to **Run > Resume**. You should see the rightmost blue LED on the Beaglebone flash one long pulse, then one short pulse, repeating about once per second. If you saw the *No source available...* error before, it should now be replaced with a view of your code:



```
bbone_ledblink Default [GDB Hardware Debugging]
  code.elf
    Thread [1] (Suspended : User Request)
      dummy_wait() at code.c:52 0x800010a4
  gdb
```

Name	Type	Value
nr_of_nops		Error: Multiple errors

```
    for (; counter < nr_of_nops; ++counter)
    {
        ;
    }
}

static inline void init_led_output()
{
    //enable the GPIO module
    //CM PER Registers -> CM PER GPIO1 CLKCTRL
```

```
# uint32_t
# LED_PATTERN
write32(unsigned int, con
clrbits_le32(const void*,
setbits_le32(const void*,
```

```
bbone_ledblink Default [GDB Hardware Debugging] gdb

The target endianness is set automatically (currently little endian)
```

If you see the *No source available...* error again, end your session (top menu bar, **Run > Disconnect**) and try again. If you still see *No source available...*, end your session again, but this time exit OpenOCD and unplug the BeagleBone and the Flyswatter2. Plug them back in, and repeat Steps 1-3 above.

Using GDB in Eclipse

Eclipse provides a GUI wrapper around the most common GDB commands. Normally you would run GDB commands from a terminal window, but you can perform several GDB actions through Eclipse.

Suspend and Resume the Program

To pause the program, use the top menu, **Run > Suspend**. Try this and you should see the LEDs on the BeagleBone stop flashing. To resume, use **Run > Resume** or press **F8** on your keyboard.

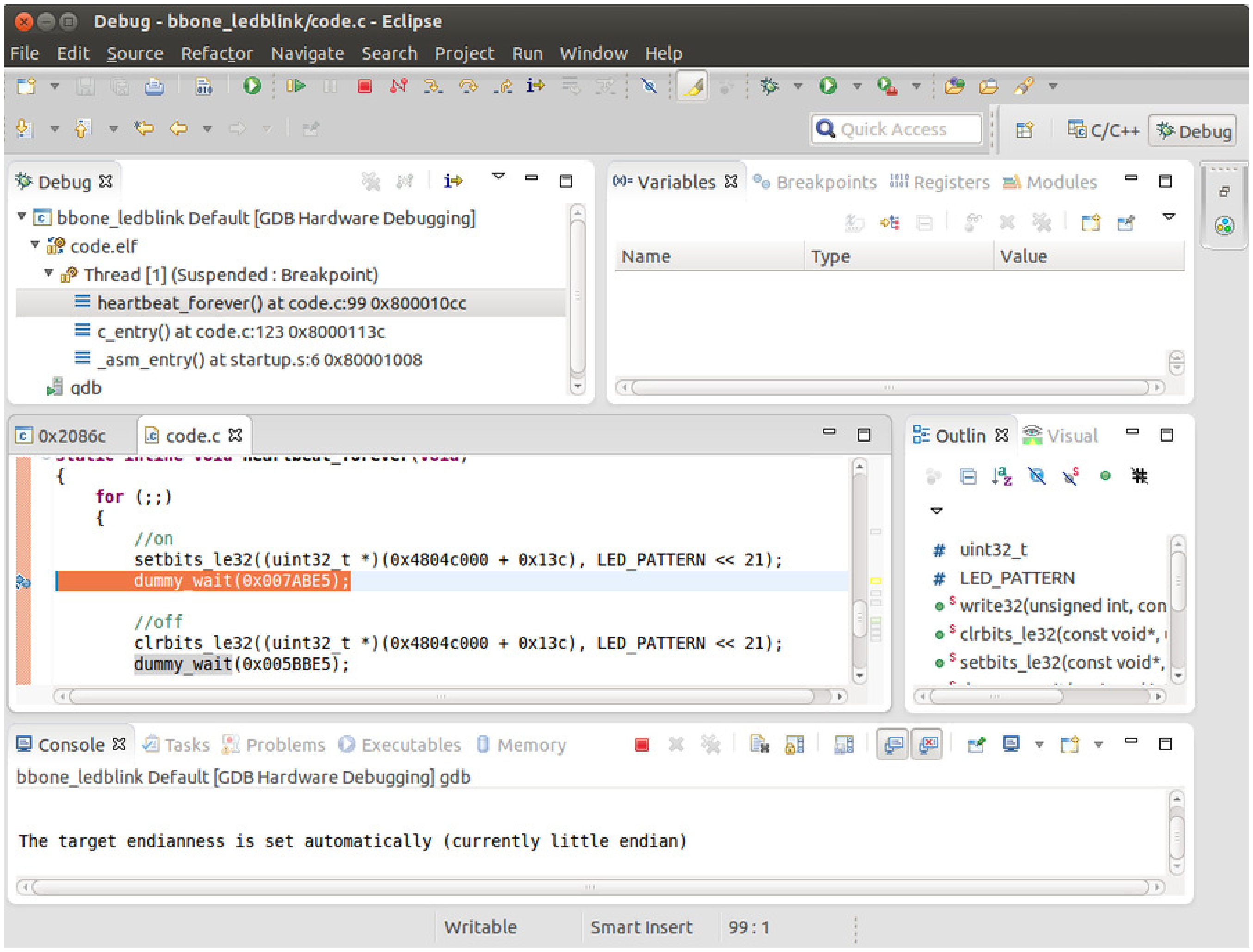
Insert a Breakpoint

A breakpoint suspends the program each time it reaches a particular line of code. Breakpoints allow you to read the contents of memory before the program can change it, or find errors by verifying that the program runs correctly up to a particular point.

First suspend the program as above (**Run > Suspend**). In the code view for *code.c* at the center left of the Eclipse screen, click to place your mouse cursor in line 99. Line 99 should read:

```
dummy_wait(0x007ABE5);
```

In the top menu, select **Run > Toggle Breakpoint**. Alternatively you can press **Shift+Ctrl+B** on your keyboard. A blue dot should appear to the left of Line 99.



```
bbone_ledblink Default [GDB Hardware Debugging]
  code.elf
    Thread [1] (Suspended : Breakpoint)
      heartbeat_forever() at code.c:99 0x800010cc
      c_entry() at code.c:123 0x8000113c
      _asm_entry() at startup.s:6 0x80001008
  gdb
```

```
{
    for (;;)
    {
        //on
        setbits_le32((uint32_t *)(0x4804c000 + 0x13c), LED_PATTERN << 21);
        dummy_wait(0x007ABE5);

        //off
        clrbits_le32((uint32_t *)(0x4804c000 + 0x13c), LED_PATTERN << 21);
        dummy_wait(0x005BBE5);
```

```
# uint32_t
# LED_PATTERN
● s write32(unsigned int, con
● s clrbits_le32(const void*,
● s setbits_le32(const void*,
```

```
bbone_ledblink Default [GDB Hardware Debugging] gdb

The target endianness is set automatically (currently little endian)
```

Resume the program (**Run > Resume** or **F8**). The program will run until it reaches the breakpoint and then suspend automatically. You should see the LED on the BeagleBone flash on and stay on. Because this breakpoint is in a function that is called repeatedly from a loop that never terminates, you can keep repeating this indefinitely. Each time you resume the program, the light should turn off, then quickly turn on and off again, then come on and stay on.

To remove the breakpoint, select **Run > Toggle Breakpoint** or press **F8** again. Do that now.

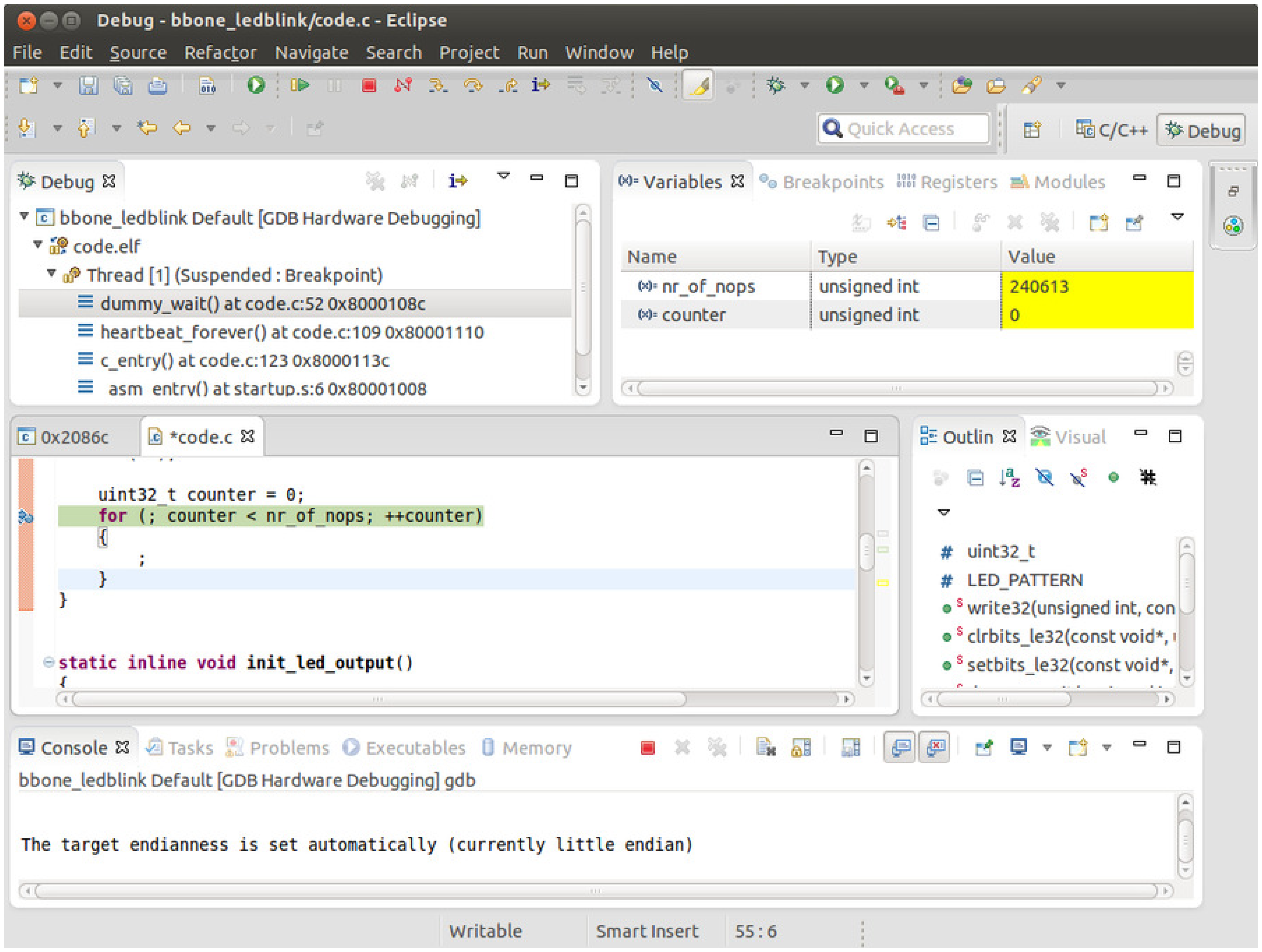
Check the Value of a Variable

While the program is suspended, Eclipse shows you the current value of variables. Insert a new breakpoint at Line 52. Just as before, suspend the program (**Run > Suspend**), place your cursor in Line 52 and select **Run > Toggle Breakpoint** or press **F8**. Line 52 is a *for* loop that forces LEDblink to delay between flashes:

```
for (; counter < nr_of_nops; ++counter)
```

After you insert the breakpoint at line 52, resume the program with **Run > Resume** or **F8**. The program will now break each time the *dummy_wait()* function is called. In the upper right, you can see the values of two variables:

- **nr_of_ops**, which controls how many passes through the loop to wait. You can see that the program calls **dummy_wait()** with different values for long or short flashes and long or short pauses.
- **counter**, which should always be zero. We're just starting the for loop, so it hasn't yet incremented.



```
bbone_ledblink Default [GDB Hardware Debugging]
  code.elf
    Thread [1] (Suspended : Breakpoint)
      dummy_wait() at code.c:52 0x8000108c
      heartbeat_forever() at code.c:109 0x80001110
      c_entry() at code.c:123 0x8000113c
      asm_entry() at startup.s:6 0x80001008
```

Name	Type	Value
nr_of_nops	unsigned int	240613
counter	unsigned int	0

```
    uint32_t counter = 0;
    for (; counter < nr_of_nops; ++counter)
    {
        ;
    }
}

static inline void init_led_output()
{
```

```
# uint32_t
# LED_PATTERN
write32(unsigned int, con
clrbits_le32(const void*, 
setbits_le32(const void*,
```

```
bbone_ledblink Default [GDB Hardware Debugging] gdb

The target endianness is set automatically (currently little endian)
```

Check the Value of a Memory Address

Unfortunately we can't use the Eclipse GUI to read the values that turn the LEDs on and off. However, we can get those values from memory directly using GDB text commands. Remove the breakpoint at Line 52 and put a new one back at Line 99 where we had it earlier (*dummy_wait(0x007ABE5);* in the *heartbeat_forever()* function). Resume the program so that it runs to the breakpoint.

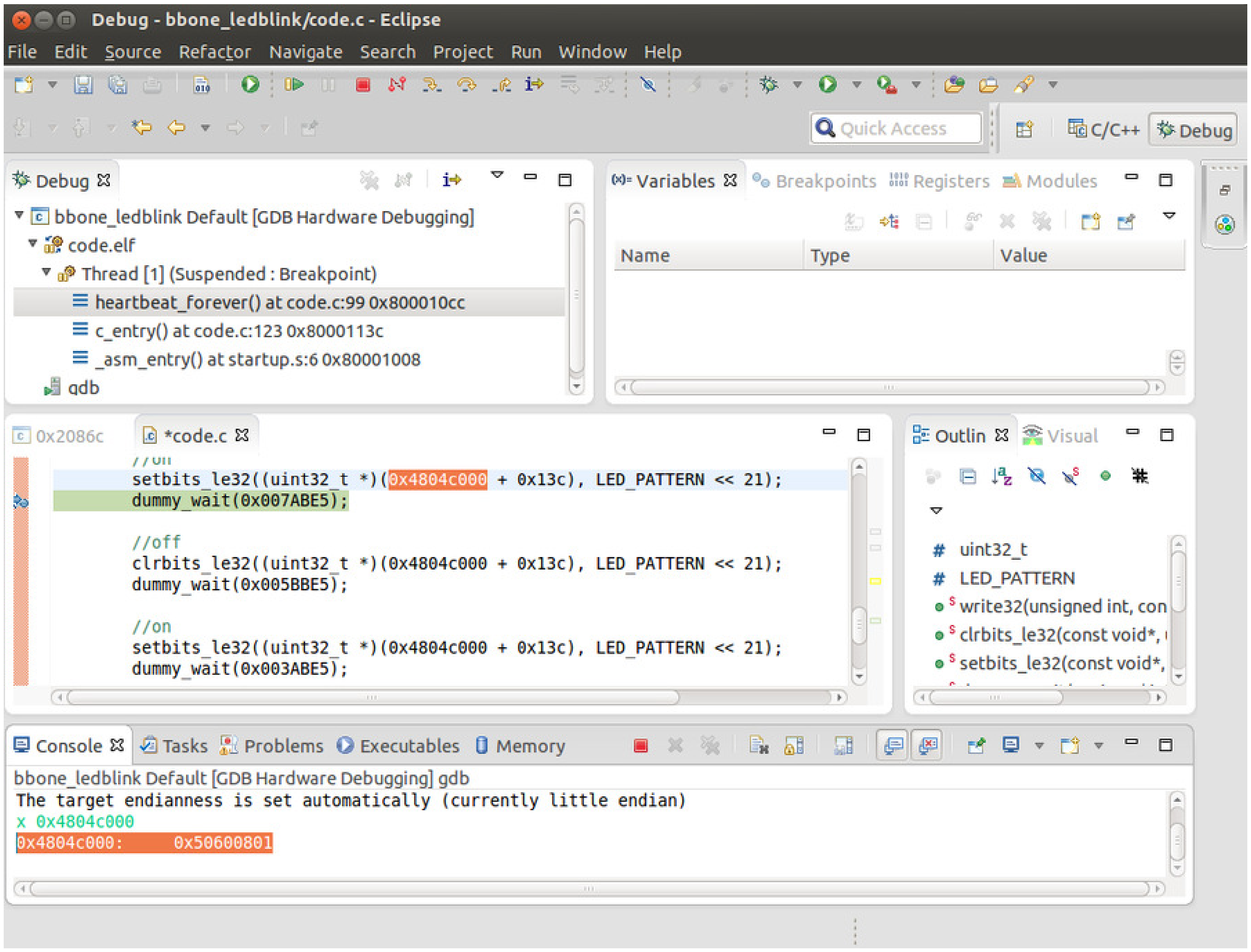
Move your mouse over **0x4804c000** in the line just above the breakpoint. This value is a memory address, and you should see a small window pop up showing you the value stored at that address.

To check the current value of *0x4804c000*, click on the **Console** window at the bottom of the screen. Your mouse cursor should appear. Move the mouse cursor to the last line and type:

```
x 0xE0028024
```

The Console window should print out:

```
0x4804c000:     0x50600801
```



```
bbone_ledblink Default [GDB Hardware Debugging]
  code.elf
    Thread [1] (Suspended : Breakpoint)
      heartbeat_forever() at code.c:99 0x800010cc
      c_entry() at code.c:123 0x8000113c
      _asm_entry() at startup.s:6 0x80001008
  gdb
```

```
        //on
        setbits_le32((uint32_t *)(0x4804c000 + 0x13c), LED_PATTERN << 21);
        dummy_wait(0x007ABE5);

        //off
        clrbits_le32((uint32_t *)(0x4804c000 + 0x13c), LED_PATTERN << 21);
        dummy_wait(0x005BBE5);

        //on
        setbits_le32((uint32_t *)(0x4804c000 + 0x13c), LED_PATTERN << 21);
        dummy_wait(0x003ABE5);
```

```
# uint32_t
# LED_PATTERN
write32(unsigned int, con
clrbits_le32(const void*,
setbits_le32(const void*,
```

```
bbone_ledblink Default [GDB Hardware Debugging] gdb
The target endianness is set automatically (currently little endian)
x 0x4804c000
0x4804c000:     0x50600801
```

Run Other Text Commands

You can run other GDB text commands from the Console window. You can find a full list of GDB commands at http://web.cecs.pdx.edu/~jrb/cs201/lectures/handouts/gdbcomm.txt .

End your Debugging Session

To stop debugging, go to **Run > Disconnect** in the top menu. You can leave Debug perspective and return to the C/C++ perspective (the window layout you saw when you first loaded your project) with **Window > Open Perspective > C/C++**.

Retrieved from "http://www.tincantools.com/w/index.php?title=BeagleBone_Black_Eclipse_and_GDB&oldid=2753"

- This page was last modified on 10 December 2013, at 13:30.