Armstrap Documentation

Release 0.0.1

Charles Armstrap

Mar 20, 2017

Contents

1 Introduction **3**

2 Hardware Overview **5**
2.1 Armstrap Eagle 5

3 Getting Started with C/C++ Development Tools for Armstrap Boards, Eclipse Edition **9**
3.1 Overview 9
3.2 Download and Installation 9
3.3 Consolidate all Downloaded Parts into a Single Folder 9
3.4 Configuring C/C++ Development Tools for Armstrap boards, Eclipse Edition, for First Use 10
3.5 Creating a C/C++ Project 12
3.6 Creating a Build of a C/C++ Project 17
3.7 Downloading and Debugging Code 28

4 Armstrap Naming and Versioning Convension **39**
4.1 Format 39
4.2 Rules 40
4.3 Remarks 40
4.4 Workflow Example 40
4.5 Naming Examples 41

5 Indices and tables **43**

Contents:

CHAPTER 1

Introduction

Armstrap a community of engineers and makers determined to help make ARM prototyping easy and fun. We focus on ease-of-use, helping real-world engineers bootstrap interactive objects or environments.

Armstrap boards are open-source electronics prototyping platforms centered on the powerful STM32F4 ARM processor. Each board can be built by hand or purchased preassembled; the software can be downloaded for free. The hardware reference designs (CAD files) are available under an MIT license, you are free to adapt them to your needs.

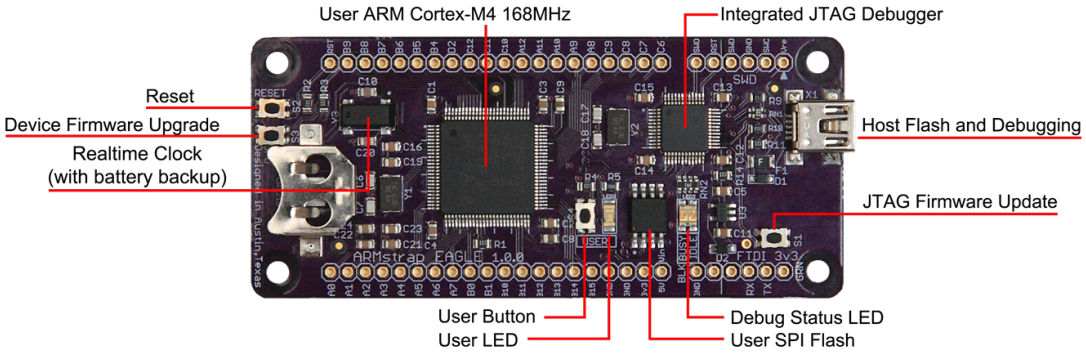
Our Goals:

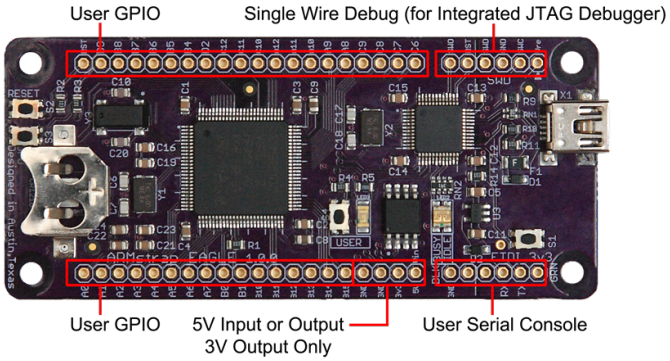
- To provide an easy-to-use ARM development experience on Microsoft Windows, Apple Macs and Ubuntu Linux platforms.
- To provide open-source ARM development that enables real-world engineers to bootstrap any project.
- To have fun!

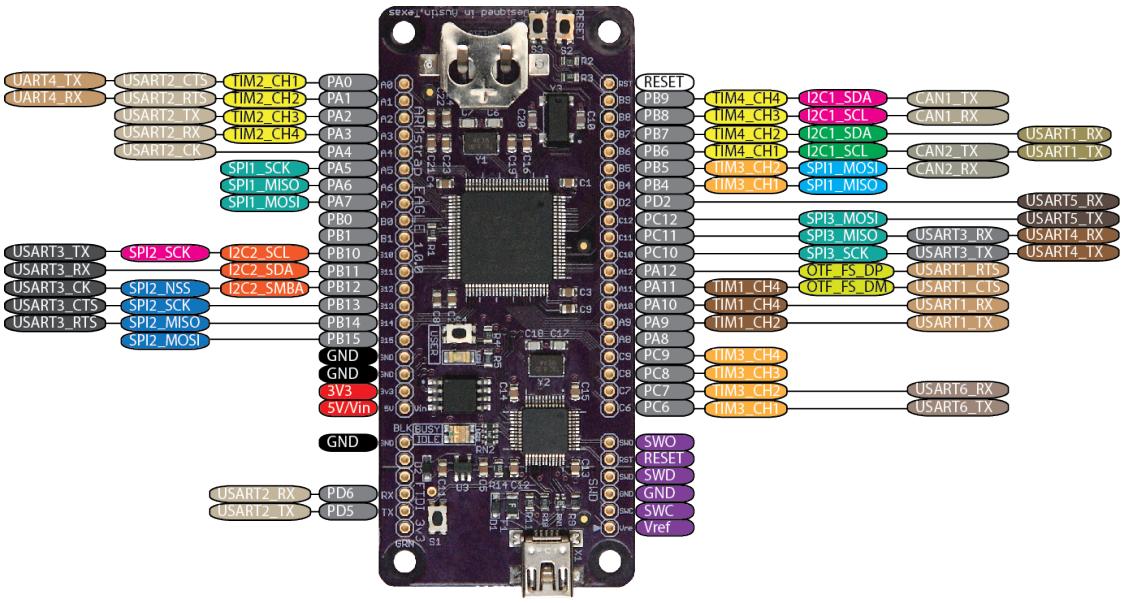
CHAPTER 2

Hardware Overview

Armstrap Eagle







Shipment Includes:

- One Armstrap Eagle Board

Standard Features

- 168MHz STM32F4 Cortex-M4 Microcontroller
- 512KB - 2048KB on-chip flash
- 192KB - 256KB RAM
- 2MB - 8MB SPI flash for easy data and configuration storage
- On-board JTAG controller for easy flashing and debugging (with no code-size limitations)
- 35 digital input/output pins
- Ready-to-use realtime clock with backup battery
- Debug over a standard USB cable
- Serial Console for runtime communication and diagnostics
- Reset and Device Firmware Upgrade button

- User-programmable LED and button
- Powered via the debug USB connector or by the 5V +Vin line
- Breadboard friendly for quick prototyping
- Flexible development options (develop with one version and deploy with another)
- Schematics and board layout file are open source (via MIT License) allowing you to build and remix the board, either personally or commercially.

Armstrap Eagle 512 Features

- 168MHz STM32F407VET6 Cortex-M4 Microcontroller
- 512KB on-chip flash
- 192KB RAM
- 2MB SPI flash

Armstrap Eagle 1024 Features

- 168MHz STM32F417VGT6 Cortex-M4 Microcontroller
- 1024KB on-chip flash
- 192KB RAM
- 4MB SPI flash

Armstrap Eagle 2048 Features

- 168MHz STM32F427VIT6 Cortex-M4 Microcontroller
- 2048KB on-chip flash
- 256KB RAM
- 8MB SPI flash

Requires (sold separately):

- Mini-USB Cable for flashing and debugging the onboard ARM chip
- CR1216 battery for realtime clock backup
- FTDI TTL Serial Cable for serial console access (TTL-232R-3V3)
- GPIO through-hole header soldering
- A Microsoft Windows, Apple Mac or Ubuntu Linux computer for project development

CHAPTER 3

Getting Started with C/C++ Development Tools for Armstrap Boards, Eclipse Edition

Overview

This guide outlines how to create C/C++ projects using the C/C++ Development Tools for Armstrap boards using Eclipse, build an ELF executable from your project source code, run and debug the executable on your Armstrap target.

While this specific document shows screenshots for Apple Mac OSX, the steps are verified to work on both Microsoft Windows and Ubuntu Linux machines as well.

Download and Installation

1. Install Java (Java SE 6 or greater is recommended), which you can download at http://www.java.com/getjava.
2. Download "Eclipse IDE for C/C++ Developers" from http://www.eclipse.org/downloads
3. Download "GNU Tools for ARM Embedded Processors" from https://launchpad.net/gcc-arm-embedded
4. Download "Armstrap blinky examples" from https://s3.amazonaws.com/armstrap-public/examples/armstrap_blinkyexamples_1.0.0.zip

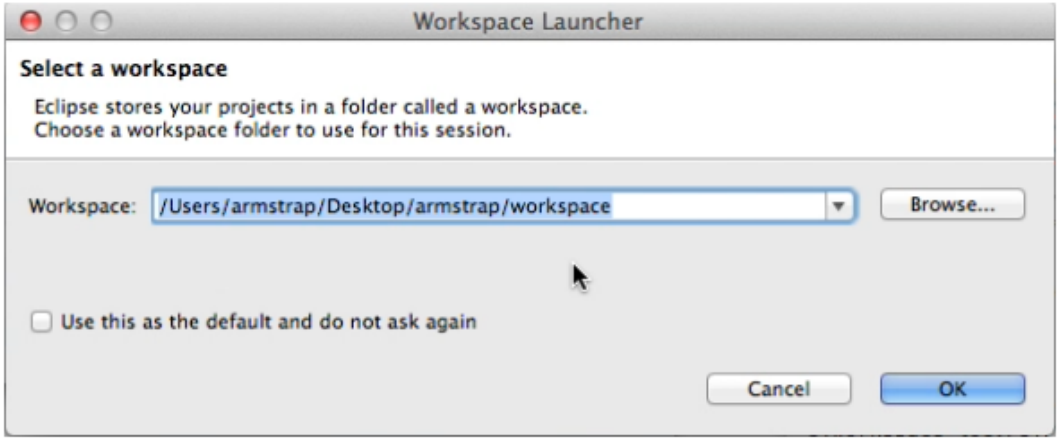
Consolidate all Downloaded Parts into a Single Folder

1. Create a folder on your Desktop called *armstrap*
2. Extract your downloaded "Eclipse IDE for C/C++ Developers" into *<user>/Desktop/armstrap/eclipse*
3. Extract your downloaded "GNU Tools for ARM Embedded Processors" into *<user>/Desktop/armstrap/gcc-arm*
4. Extract your downloaded "Armstrap blinky examples" into *<user>/Desktop/armstrap/workspace*

Configuring C/C++ Development Tools for Armstrap boards, Eclipse Edition, for First Use

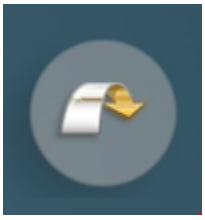
Complete the following steps to configure C/C++ Development Tools for Armstrap boards, Eclipse Edition, for first use:

1. Launch Eclipse by clicking on the *<user>/Desktop/armstrap/eclipse/eclipse* executable
2. When prompted, select the *<user>/Desktop/armstrap/workspace* folder in which to store Eclipse projects and click **OK**.

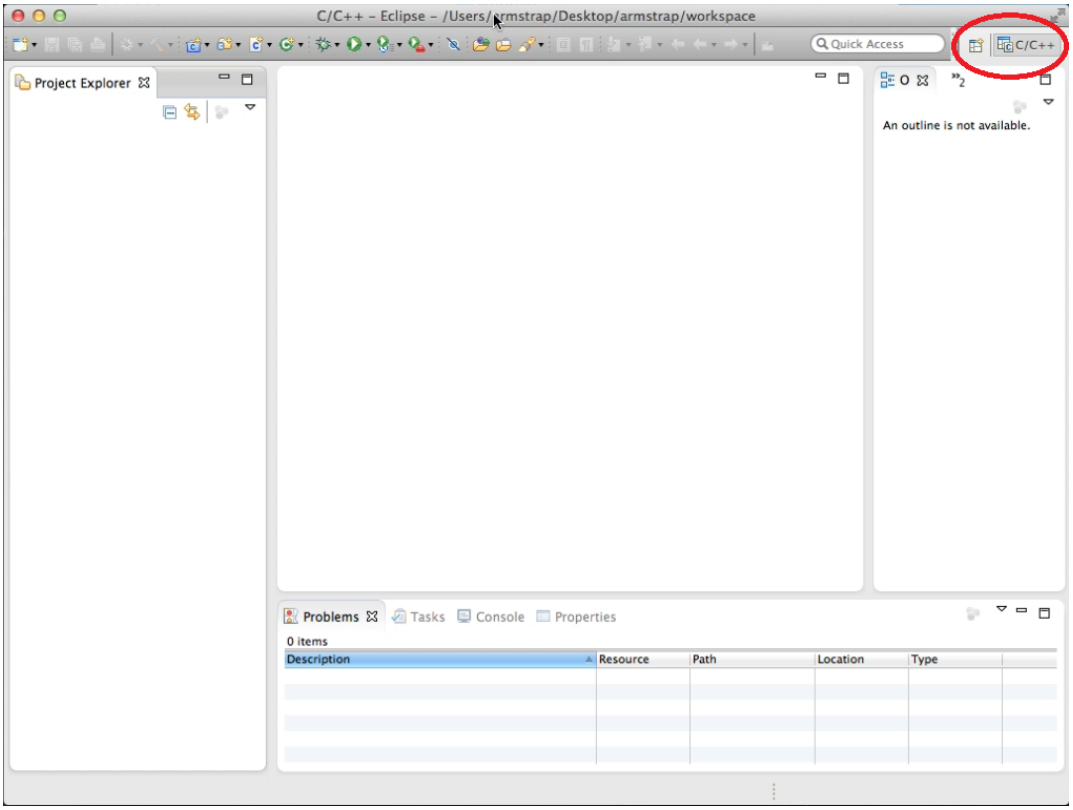


> **Tip:** Enable **Use this as the default and do not ask again** to save a project folder as your default workspace.

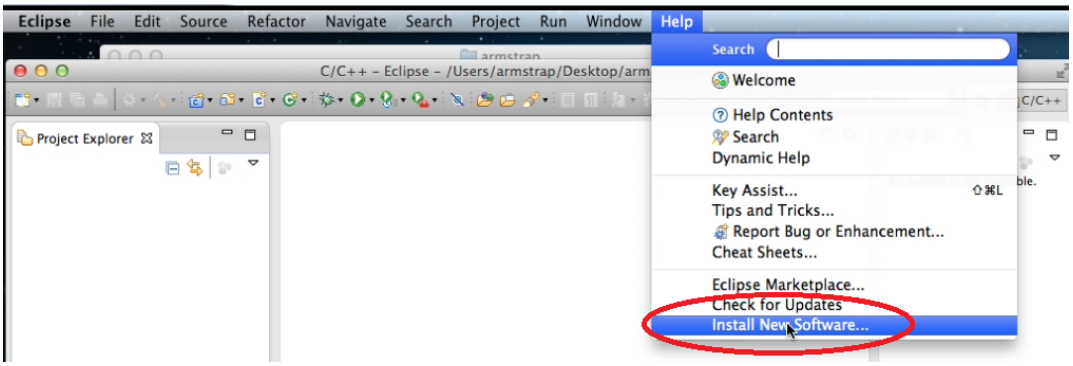
3. In the Eclipse welcome screen, select the **Workbench** icon on the far right to open the workbench view.



4. Eclipse highlights the active perspective on the perspectives bar, as shown in the following image. The first time you use Eclipse the workbench view opens in the C/C++ perspective.

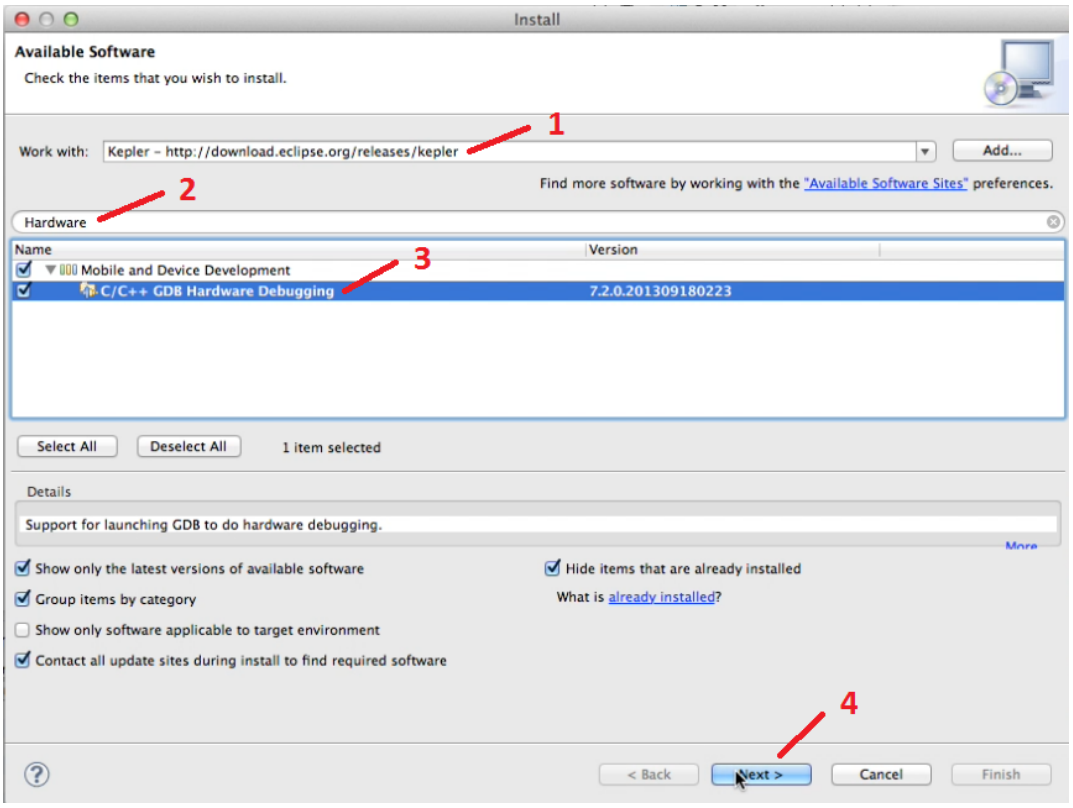


5. Debugging Armstrap requires the "C/C++ GDB Hardware Debugging" plugin. To install the plugin, select the **Help>>Install New Software...** menu item



6. Configure the plugin installation

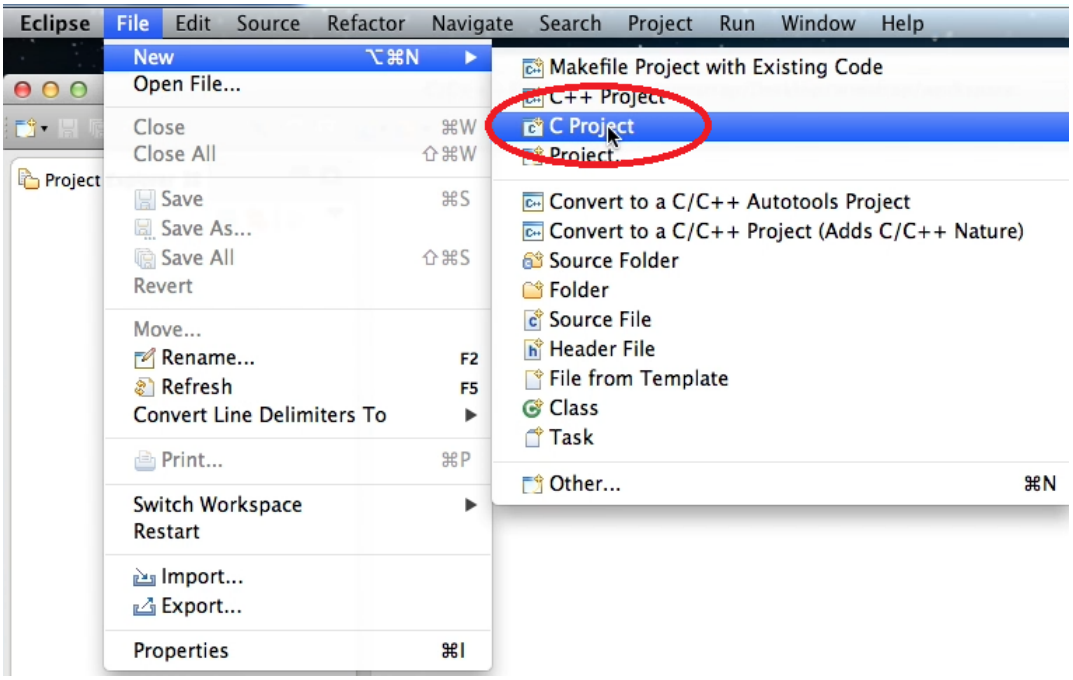
   - In the **Work with:** drop-down, select the version of Eclipse you downloaded (for example "Kepler"). As seen in label mark 1 in picture.
   - In the search field, type *Hardware* as seen in label mark 2 in picture.
   - Click the check-box to select the **C/C++ GDB Hardware Debugging** plugin as seen in label mark 3 in picture.
   - Click the **Next** button, as seen in label mark 4 in picture and accept the licensing agreement to complete the installation.
   - You will need to restart Eclipse when the plugin installation is complete.



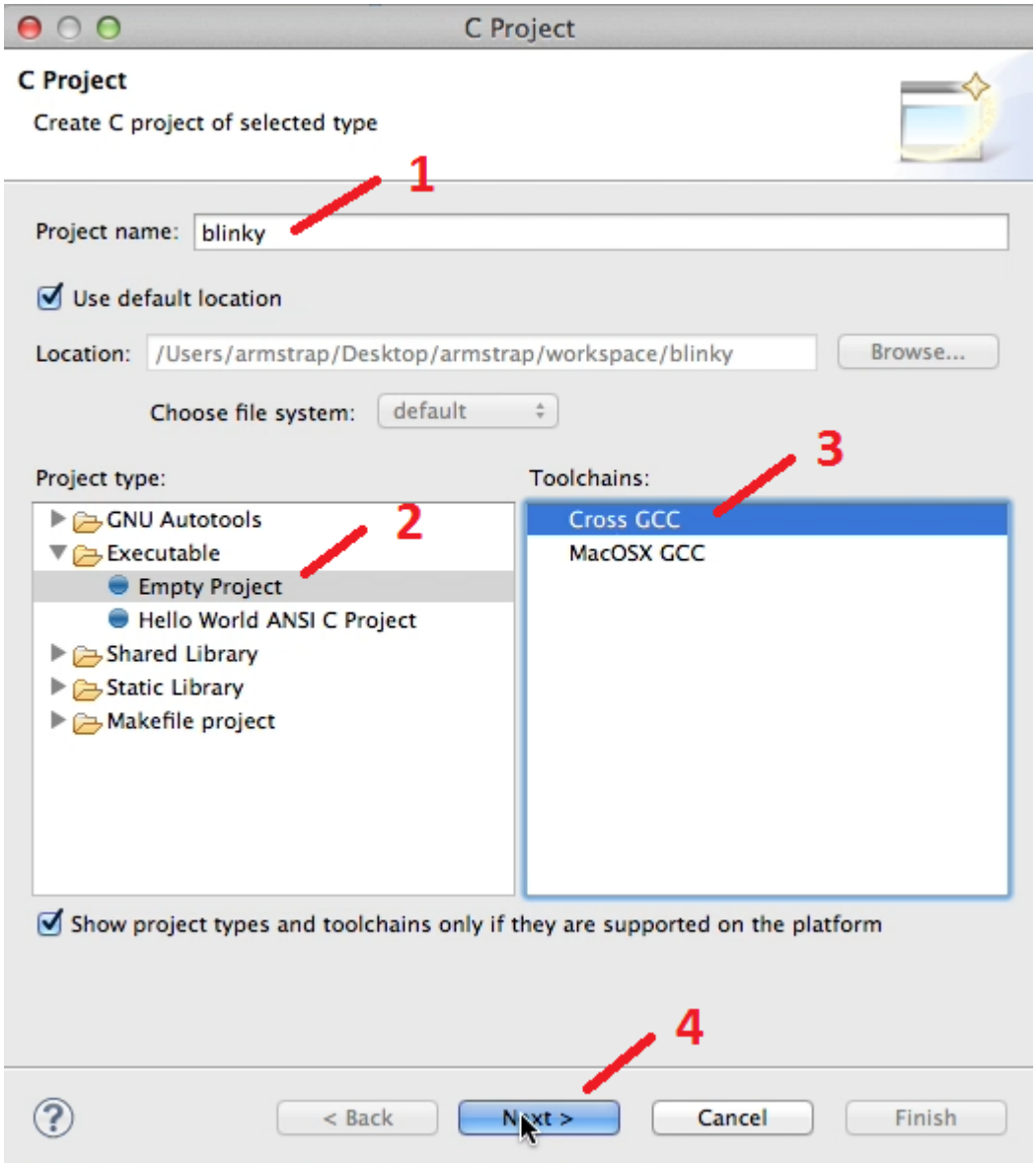
Creating a C/C++ Project

Complete the following steps to create a C or C++ project in C/C++ Development Tools for Armstrap boards

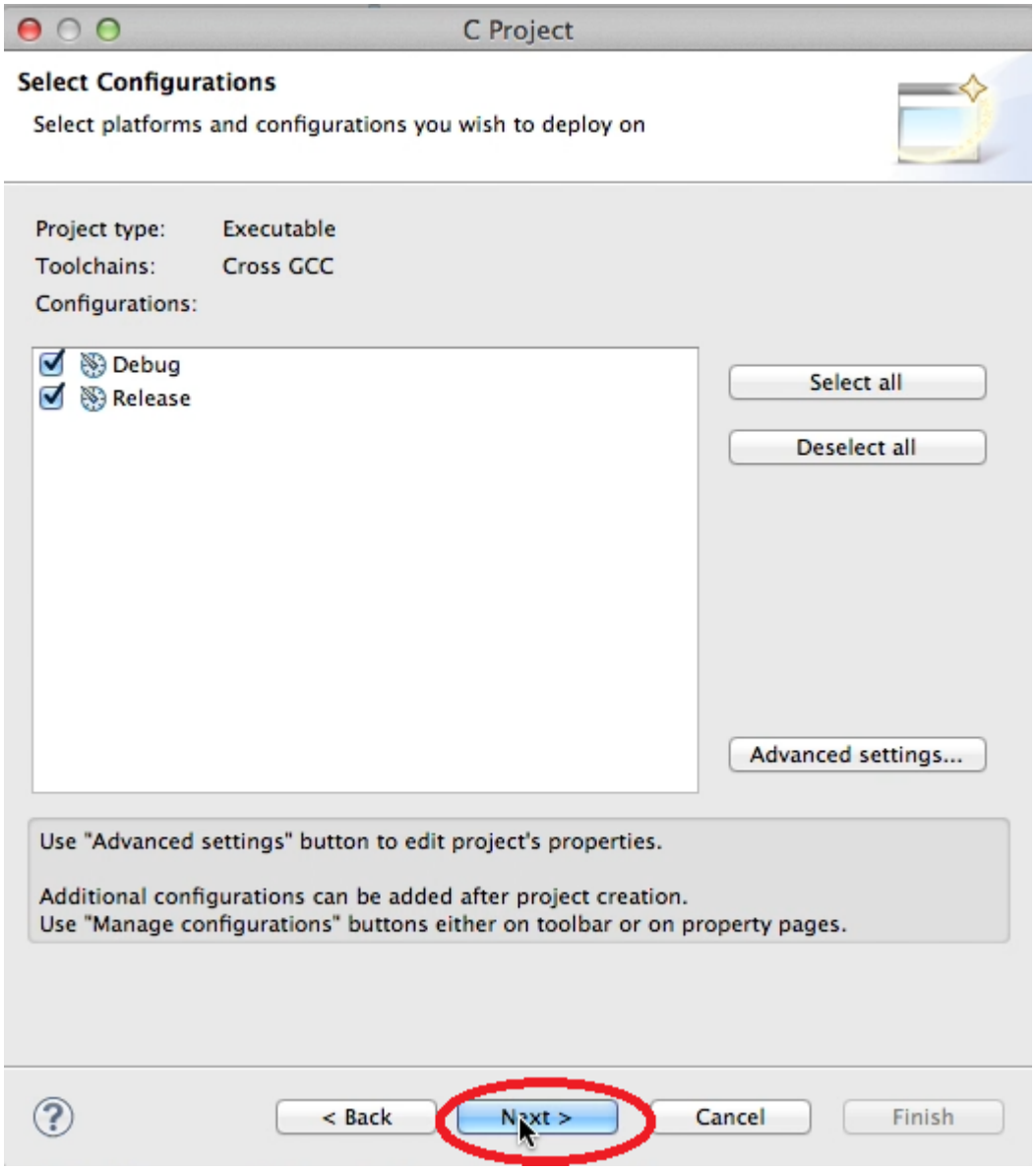
1. Switch to the C/C++ perspective.
2. Select **File>>New>>C Project** to open the New Project Wizard.



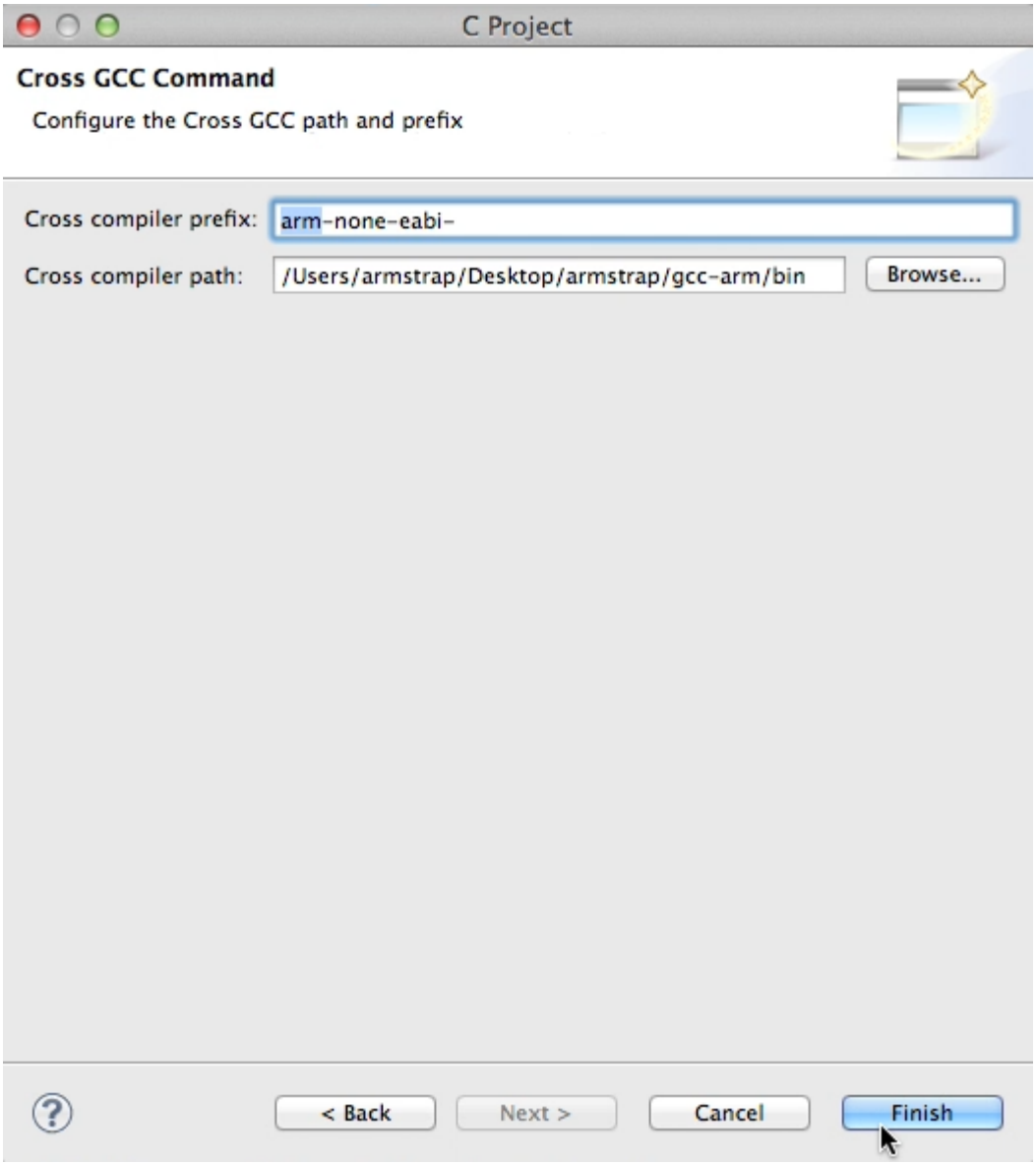
3. Configure the C Project
   - Give the project the name *blinky*, as seen in label mark 1 in picture.
   - Under **Project Type**, select the **Empty Project** option, as seen in label mark 2 in picture.
   - Under **Toolchains**, select **Cross GCC** option, as seen in label mark 3 in picture.



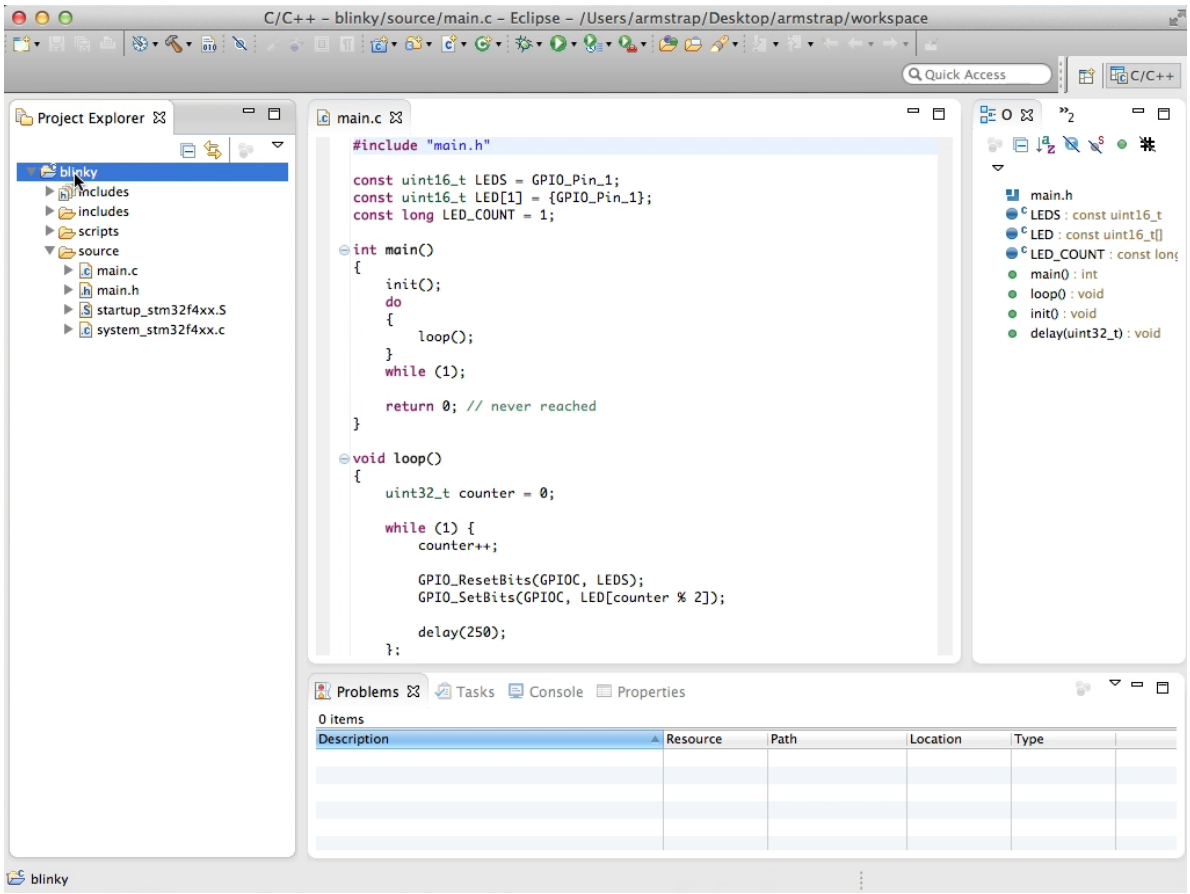
4. Click **Next** to open the **Select Configuration** page.
5. Enable **Debug** to configure the project to allow debugging your executable, and/or enable **Release** to configure the project to allow building a smaller, faster executable optimized for release. Note: For purposes of this tutorial, ensure you enable **Debug**.



6. Click **Next** to open the **Cross GCC Command** page.
7. In the **Cross compiler prefix** text box, enter *arm-none-eabi-*, including the hyphen (-) at the end, to specify the correct compiler for Armstrap targets.
8. In the Cross compiler path text box, browse to the location of the *<user>/Desktop/armstrap/gcc-arm/bin* directory to specify the location of the compiler.



9. Click **Finish** to create your project and return to the workbench view.
10. Verify your project source code appears in the Project Explorer. If it does not, you may have to hit the **F5** key to refresh.

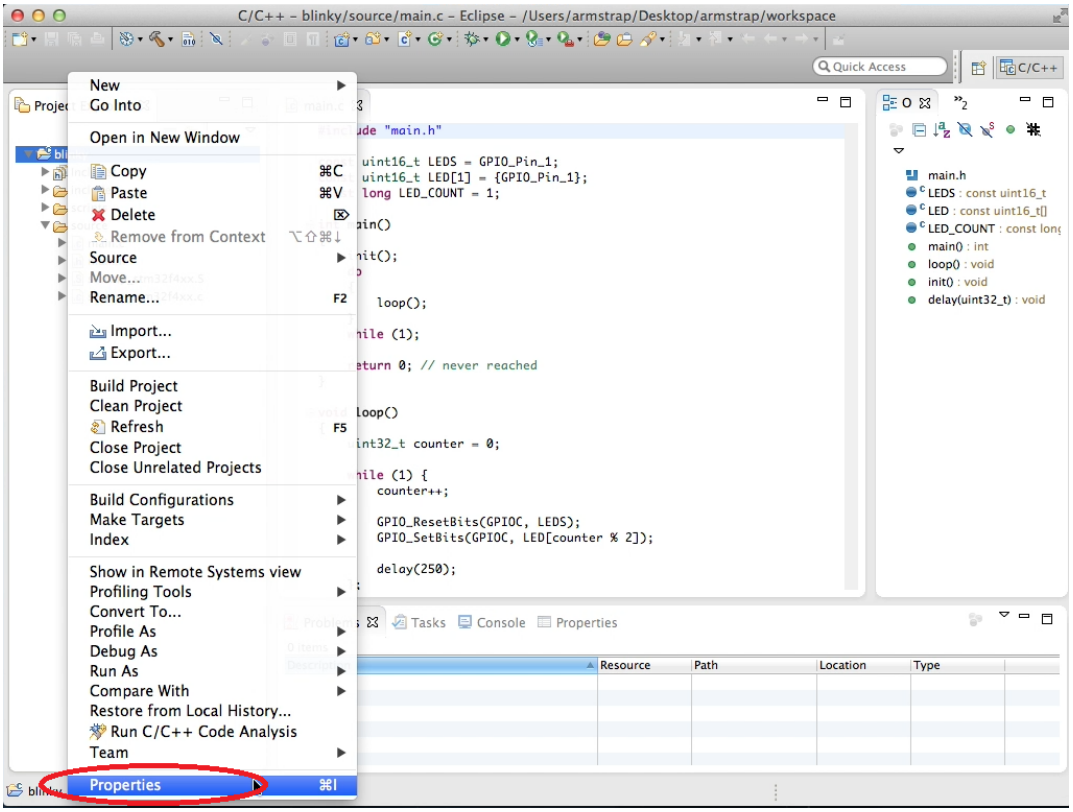


In the next section of this tutorial, you create an executable build of your project to enable it to run.

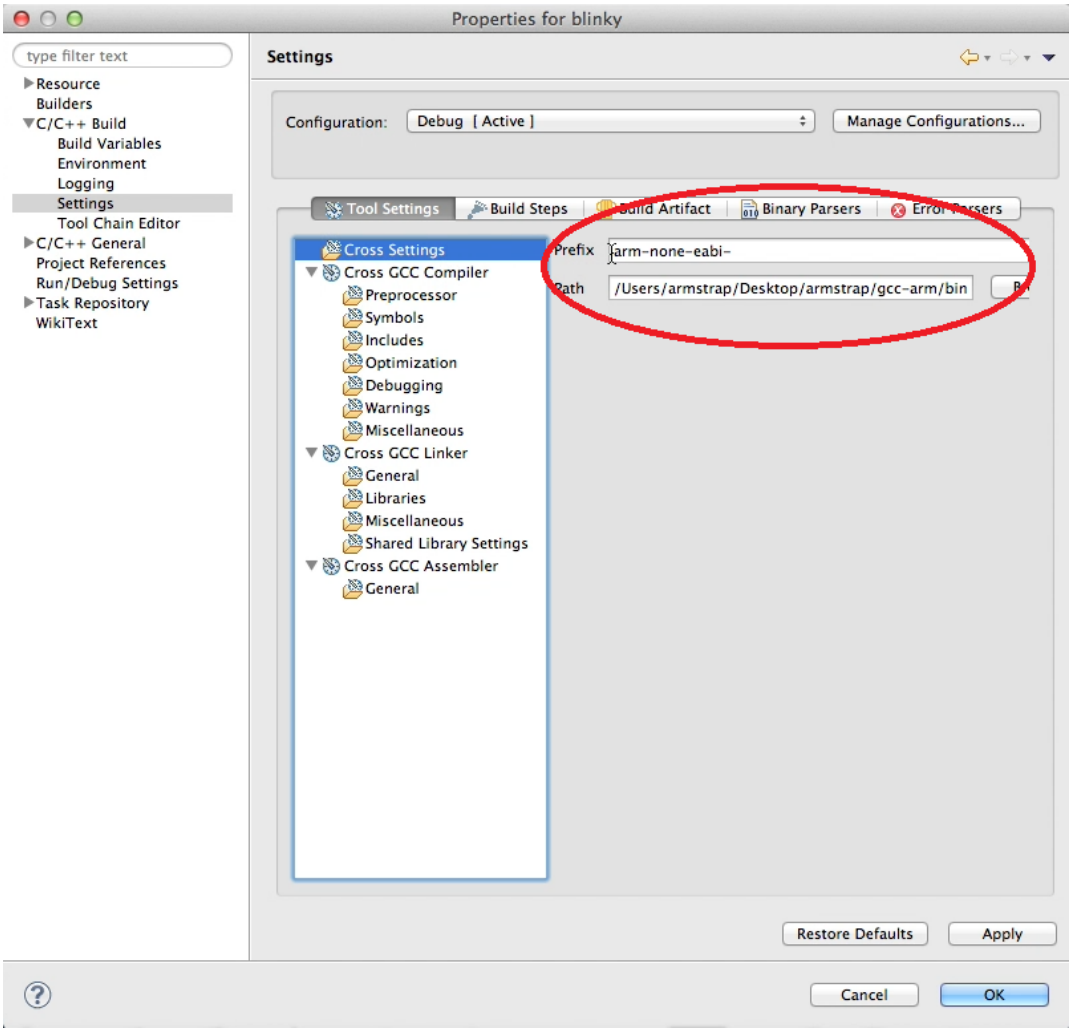
Creating a Build of a C/C++ Project

Before you can run your project, you need to test that your source code compiles by creating an executable build of your project. Complete the following steps to create an executable build of a C/C++ project:

1. Switch to the C/C++ perspective.
2. Right-click (or Ctrl-click on a Mac) your project in the **Project Explorer** tab and select **Properties**.

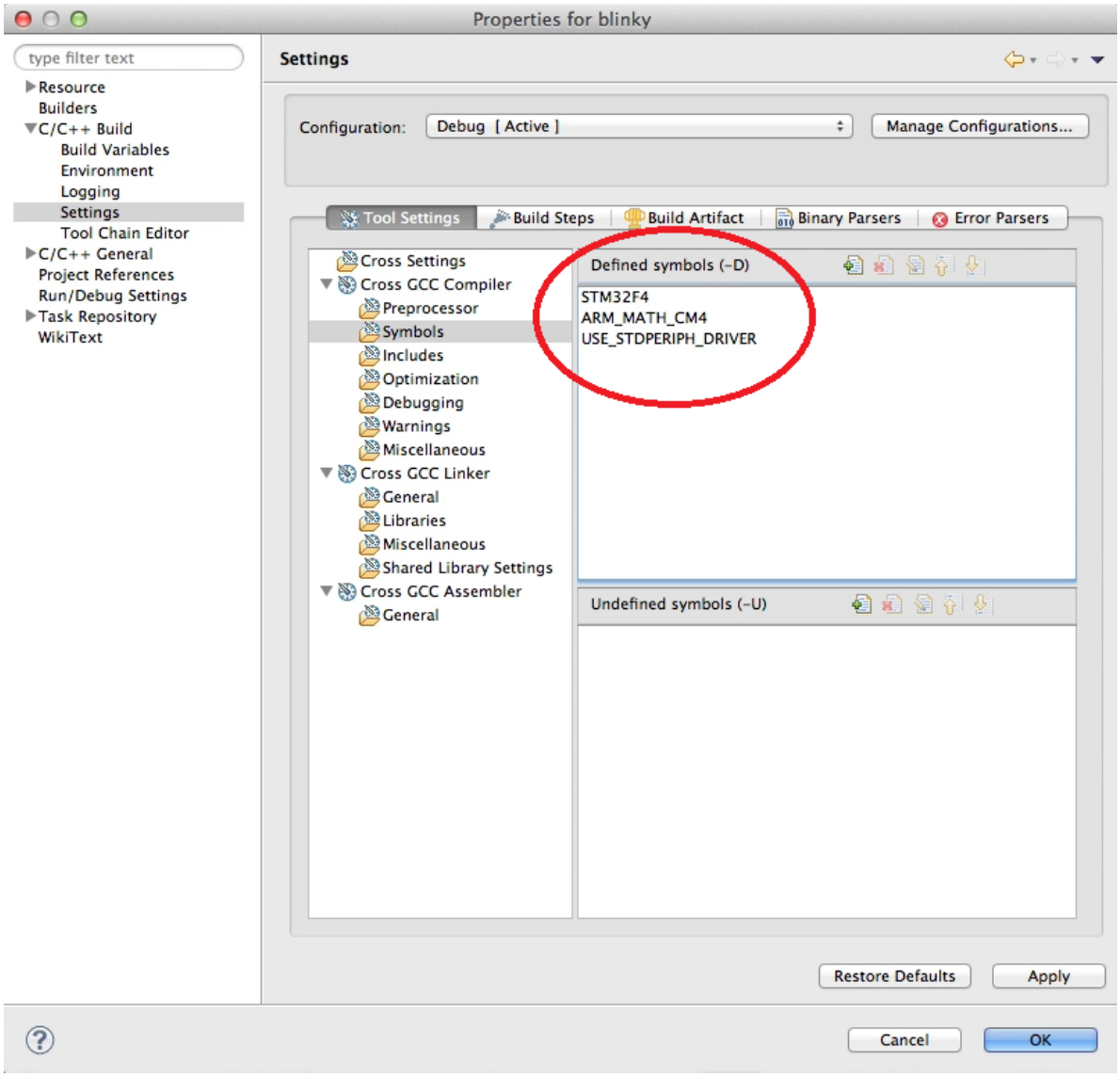


3. Select **C/C++ Build>>Settings** in the left pane of the **Properties** dialog box. Verify that **Cross Settings>>Tool Settings>>Prefix** is set to *arm-none-eabi-* and **Cross Settings>>Tool Settings>>Path** is set to the bin path to your compiler toolchain *<user>/Desktop/armstrap/gcc-arm/bin*



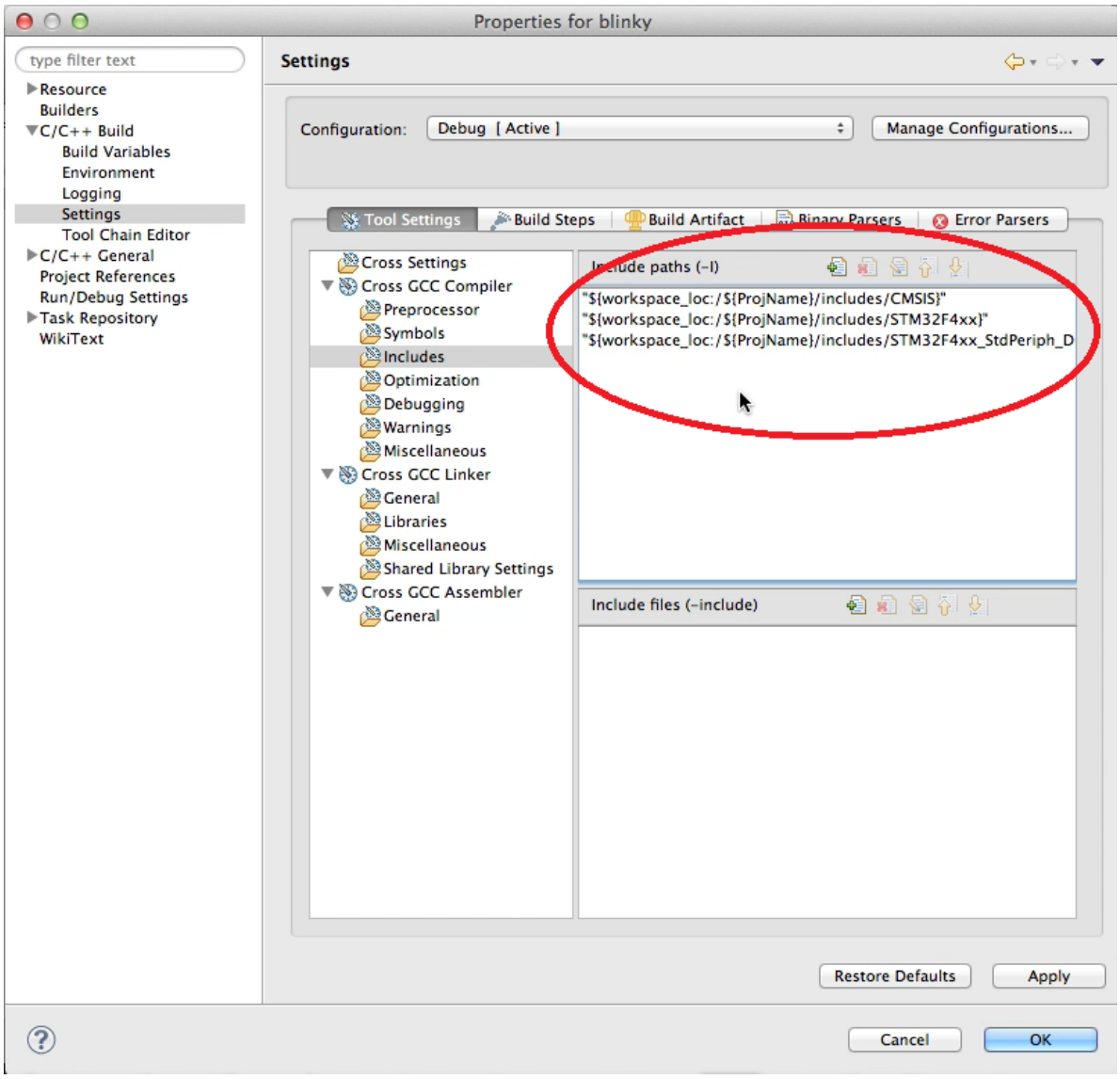
4. Under **Cross GCC Compiler>>Symbols>>Defined symbols**, enter

```
STM32F4
ARM_MATH_CM4
USE_STDPERIPH_DRIVER
```



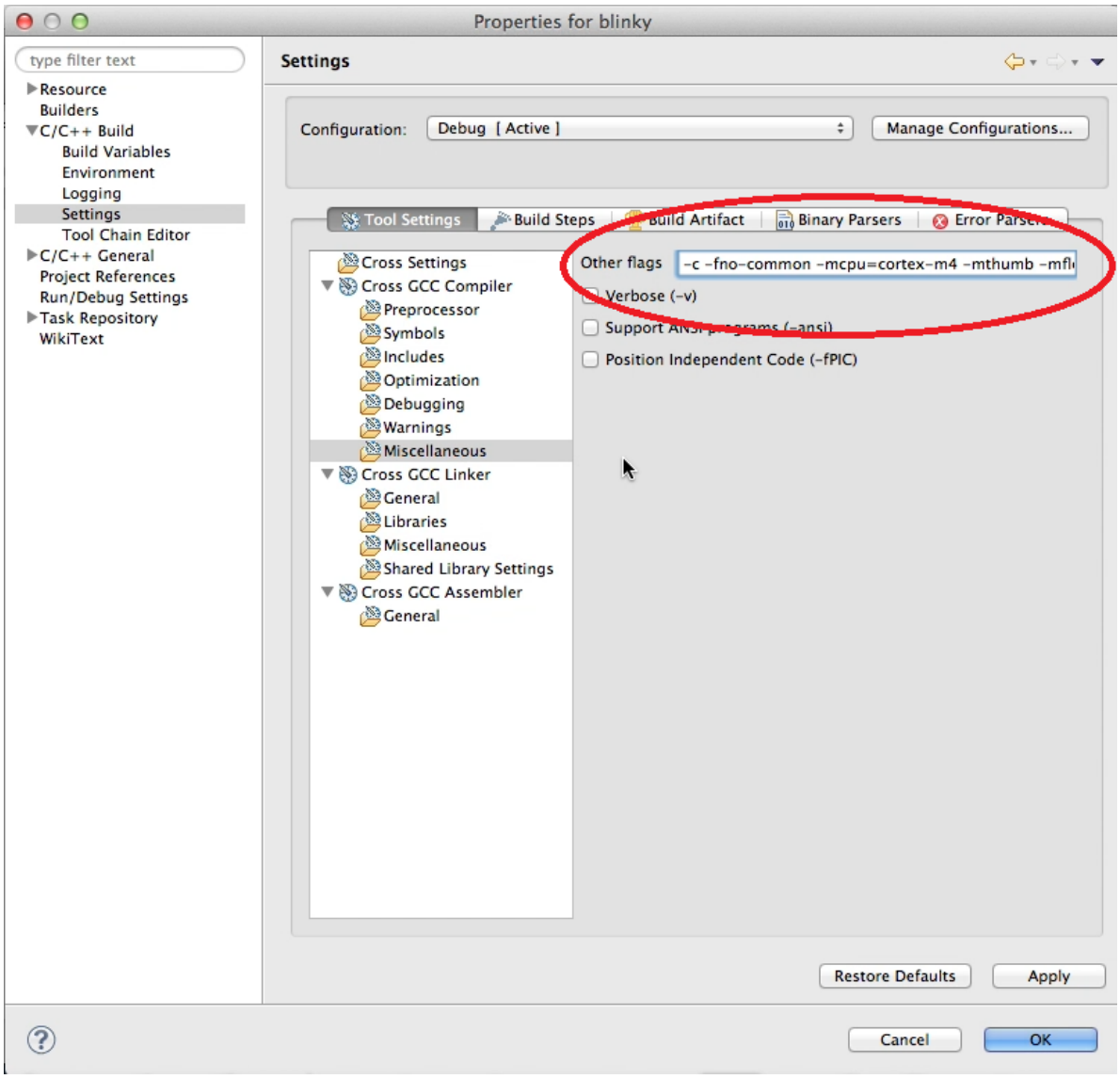
5. Under **Cross GCC Compiler>>Includes>>Include paths**, enter

```
"${workspace_loc:/${ProjName}/includes/CMSIS}"
"${workspace_loc:/${ProjName}/includes/STM32F4xx}"
"${workspace_loc:/${ProjName}/includes/STM32F4xx_StdPeriph_Driver/inc}"
```



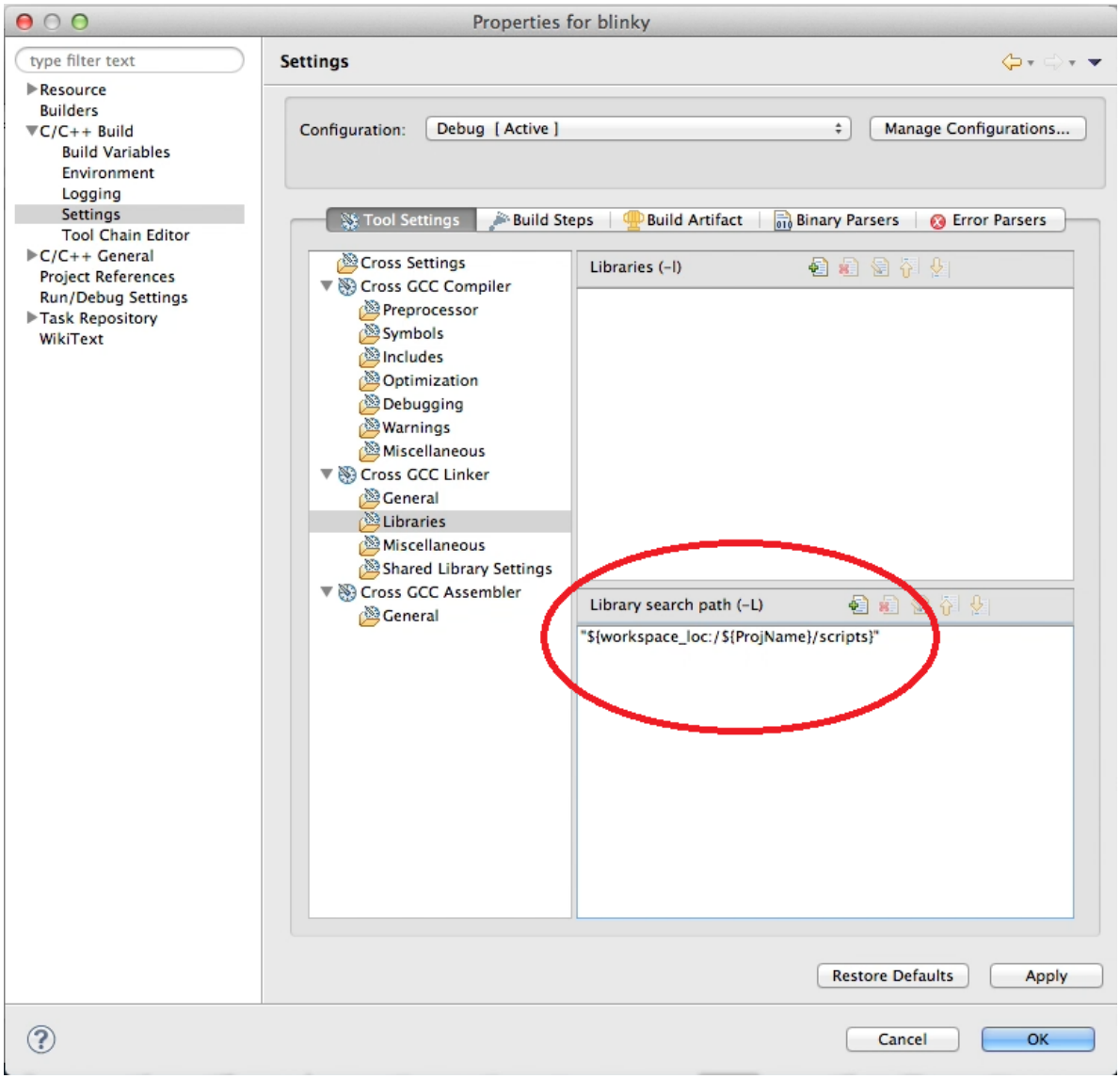
6. Under **Cross GCC Compiler>>Miscellaneous>>Other flags**, enter

```
-c -fno-common -mcpu=cortex-m4 -mthumb -mfloat-abi=hard -mfpu=fpv4-sp-d16 -MD
```



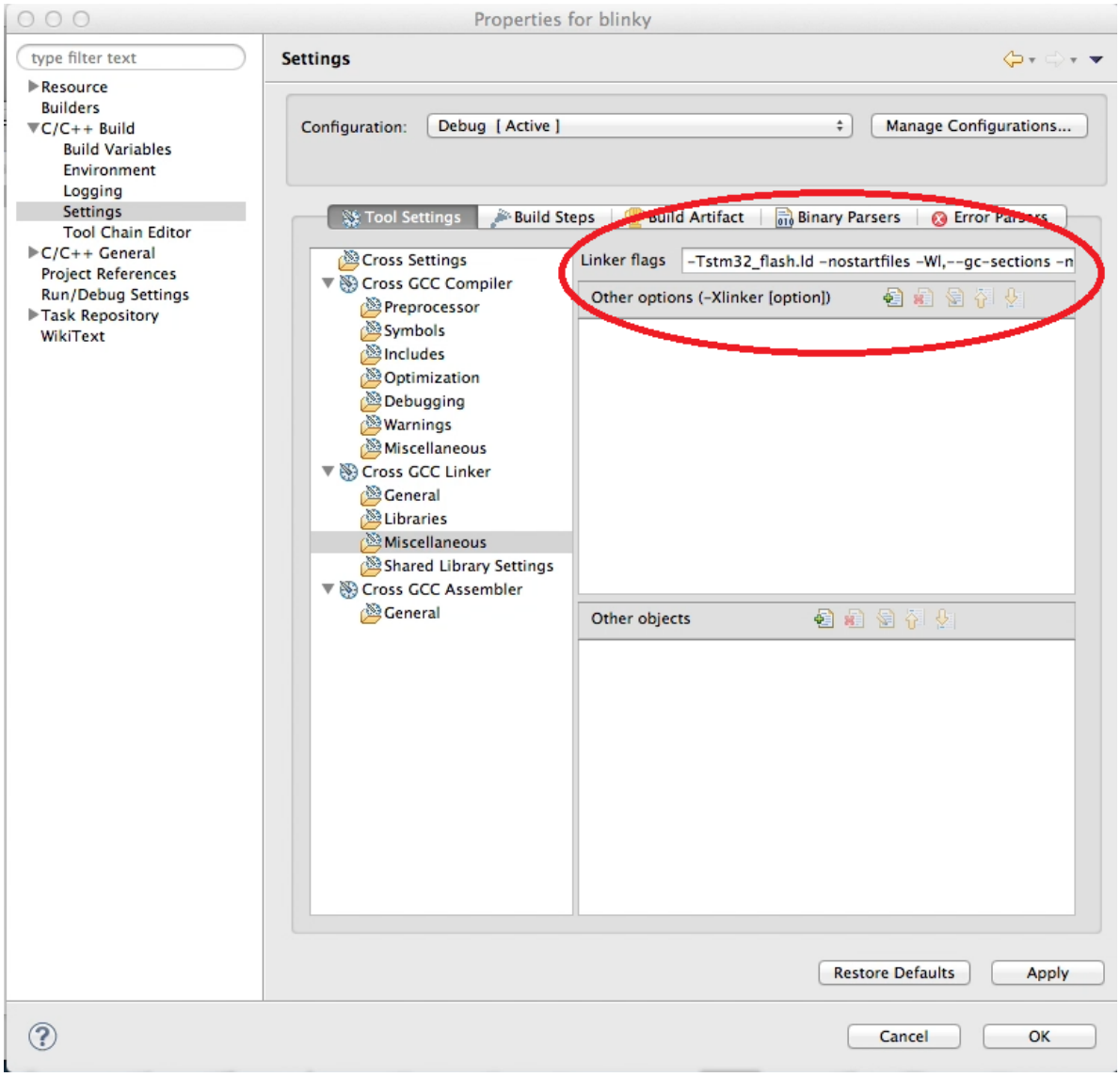
7. Under **Cross GCC Linker>>Libraries>>Library search path**, enter

```
"${workspace_loc:/${ProjName}/scripts}"
```



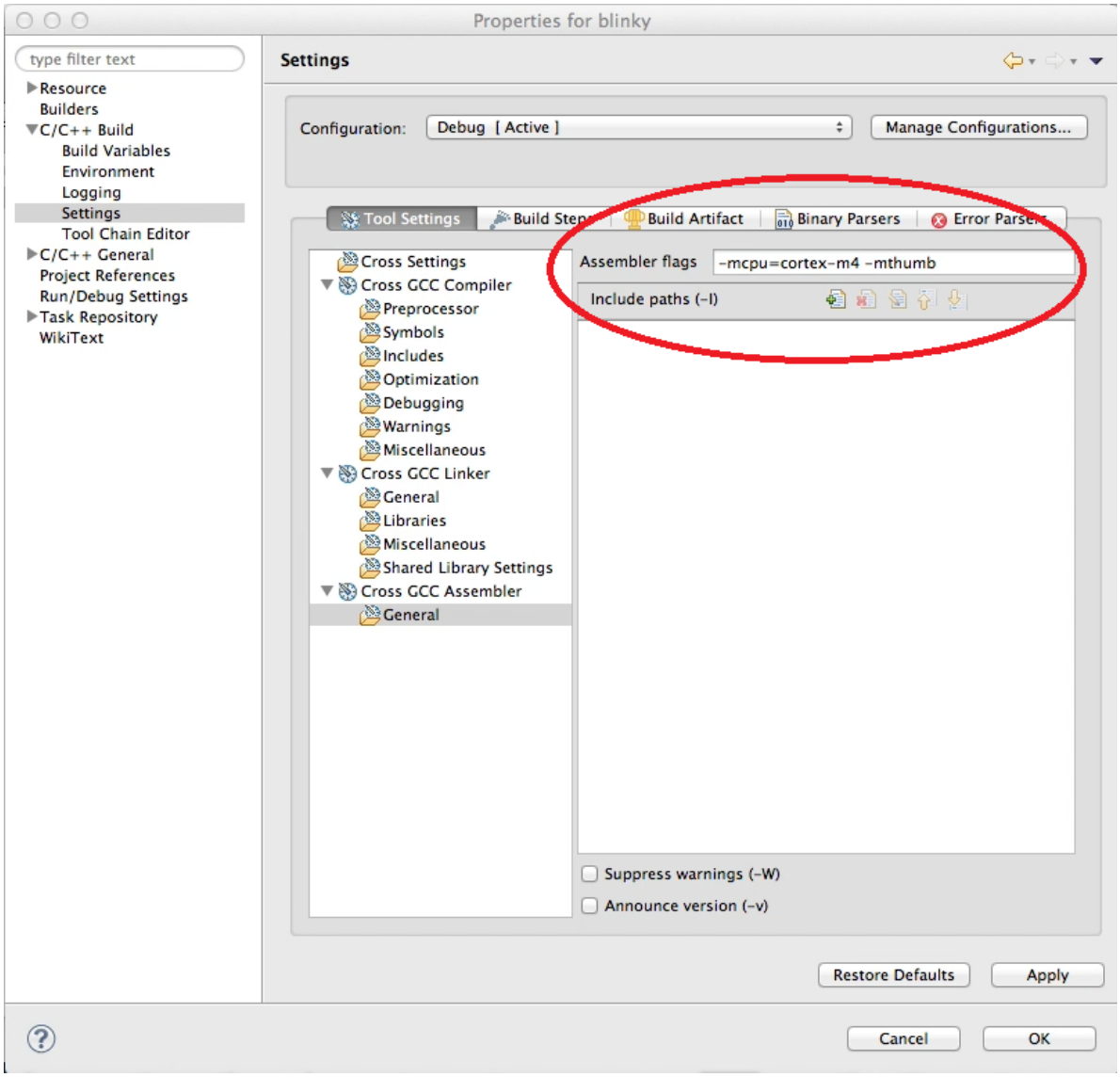
8. Under **Cross GCC Linker>>Miscellaneous>>Linker flags**, enter

```
-Tstm32_flash.ld -nostartfiles -Wl,--gc-sections -mthumb -mcpu=cortex-m4 -mthumb -
↪mfloat-abi=hard -mfpu=fpv4-sp-d16
```



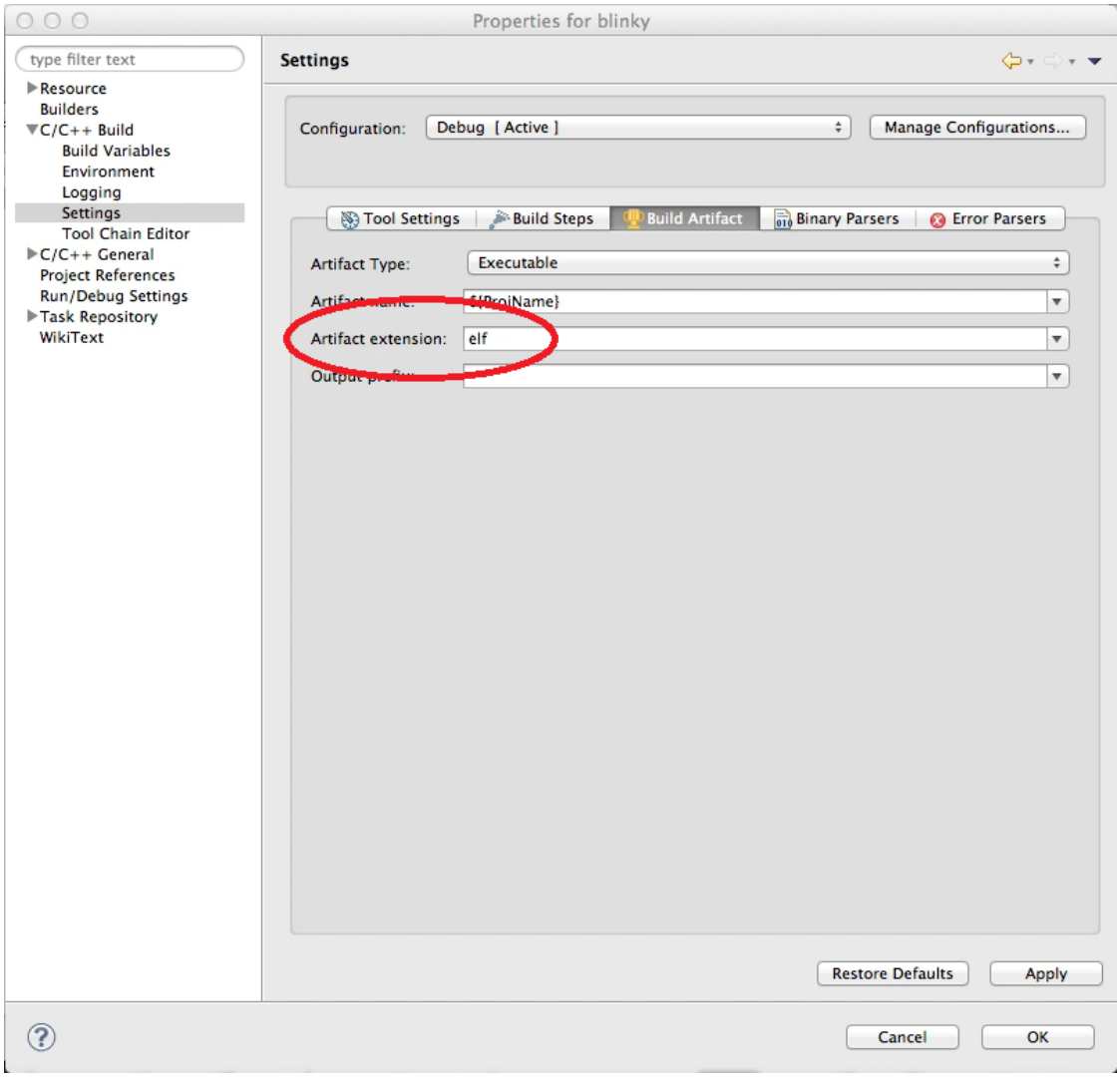
9. Under **Cross GCC Assembler>General>>Assembler flags**, enter

```
-mcpu=cortex-m4 -mthumb
```

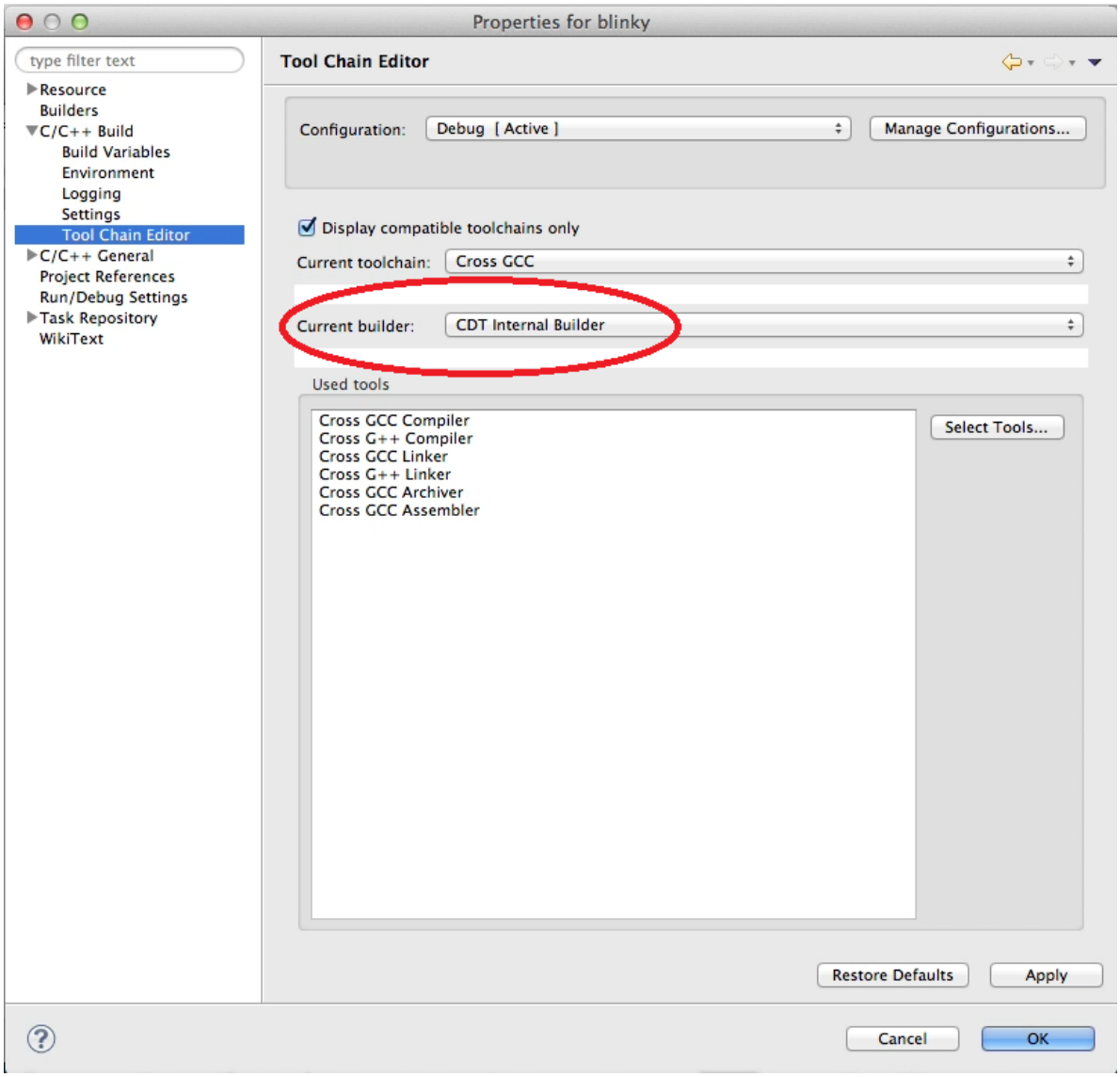


10. In the **Build Artifacts** tab, under **Artifact extension**, enter

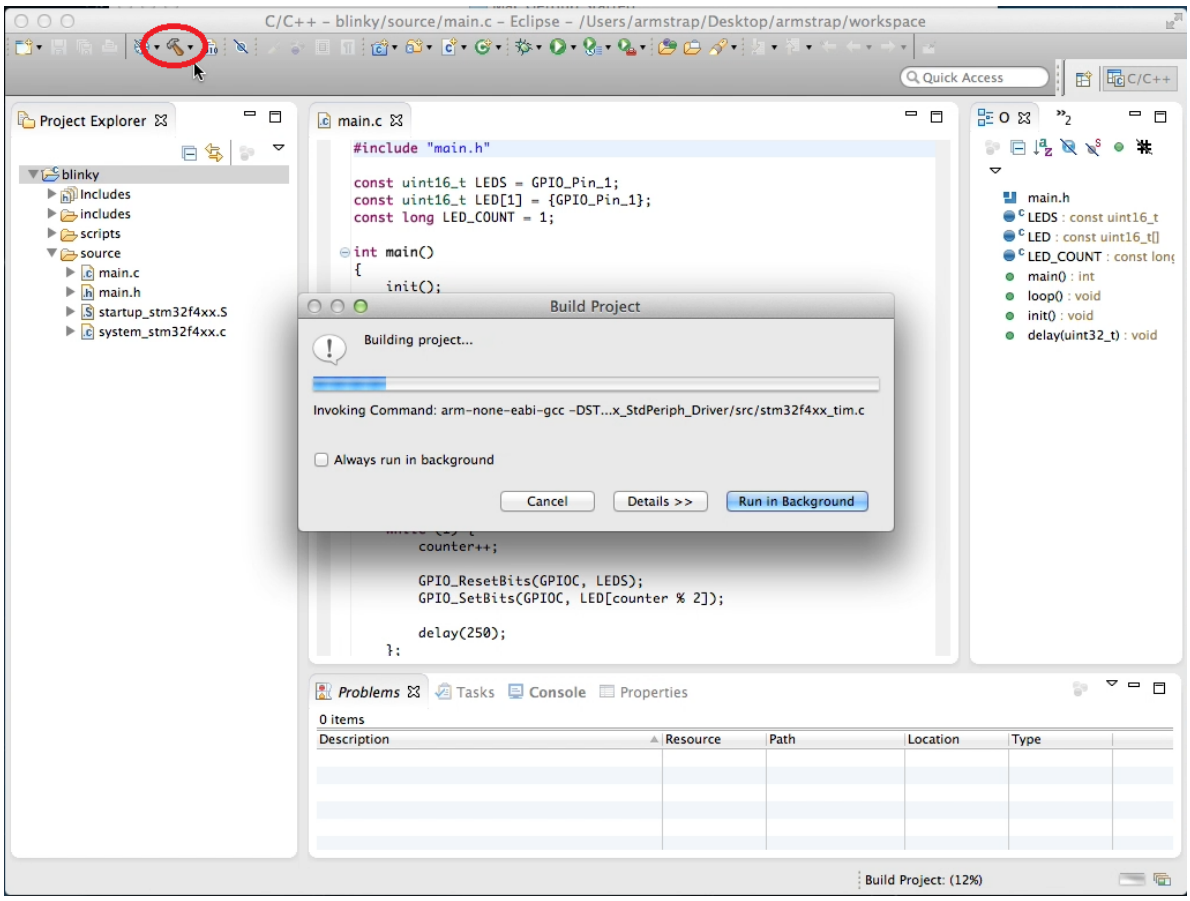
```
elf
```



11. Select **C/C++ Build>>Tool Chain Editor** in the left pane of the **Properties** dialog box. Set **Current builder** to *CDT Internal Builder*



12. Click **Apply** and then **OK** to close the **Properties** dialog box.
13. Click the build icon in the toolbar or select **Project>>Build Project** in the workbench view to create an elf executable of your project. Verify your project builds successfully.



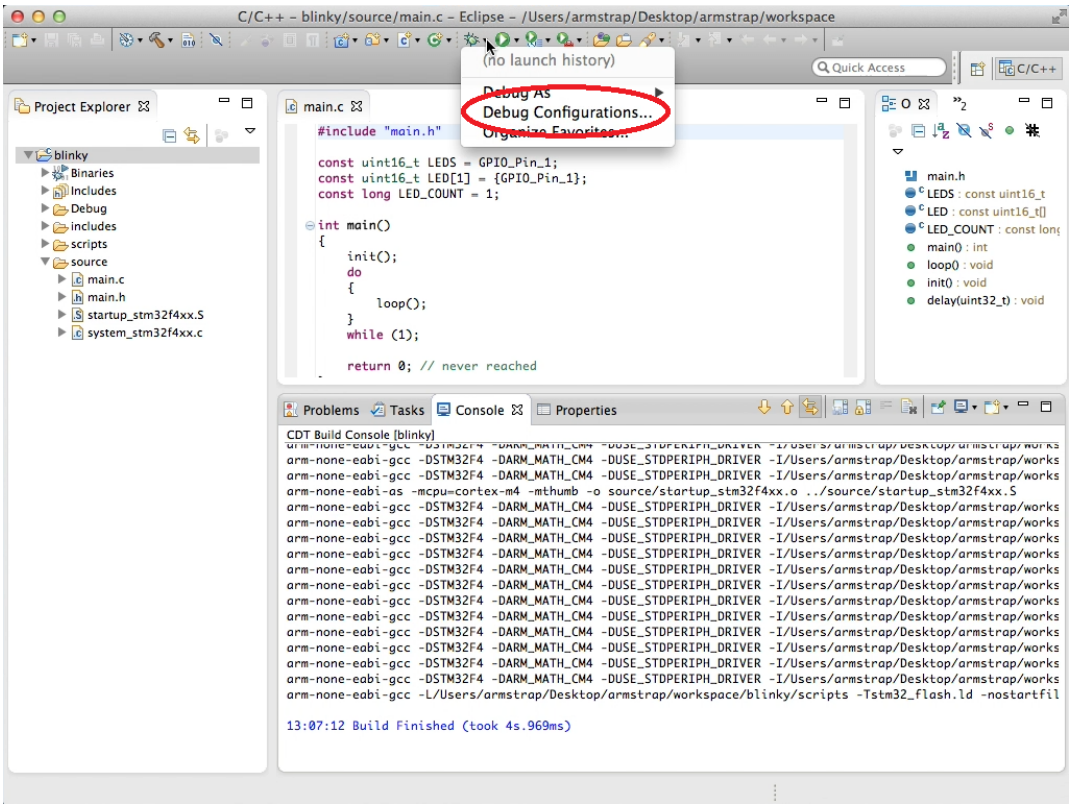
14. The Console tab displays Build Finished if the build completes successfully, as shown in the following image.

In the next section of this tutorial, you prepare to run and debug the ELF executable on your Armstrap target.

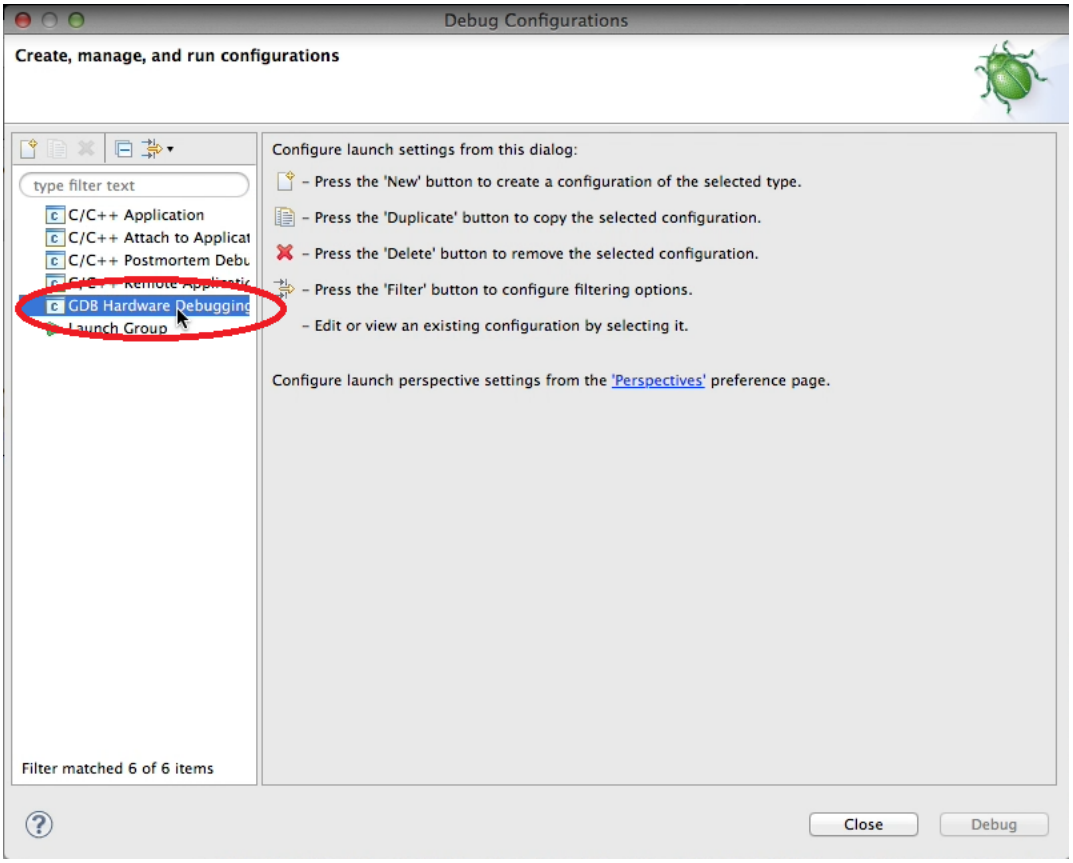
Downloading and Debugging Code

Before you can run the ELF executable you created in the previous section on your Armstrap target, you need to create a Debug Configuration.

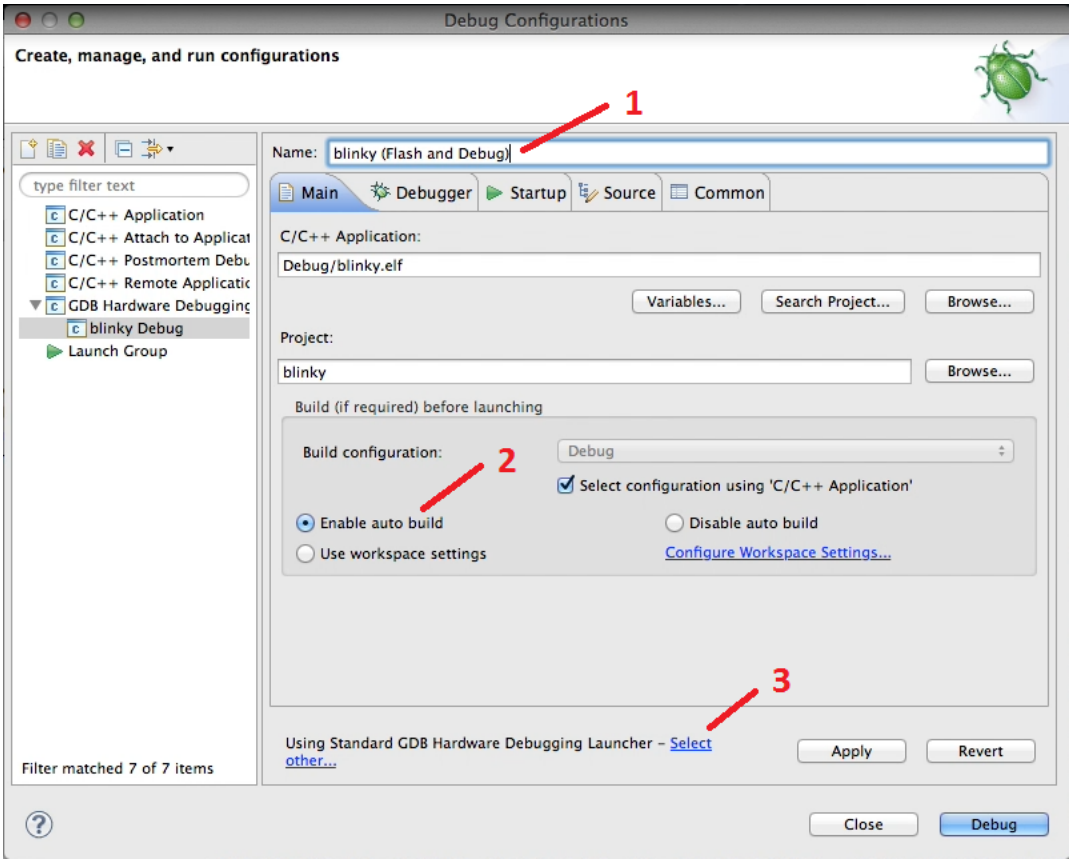
1. In the C/C++ perspective, select the **Debug Configurations...** in the debug drop-down



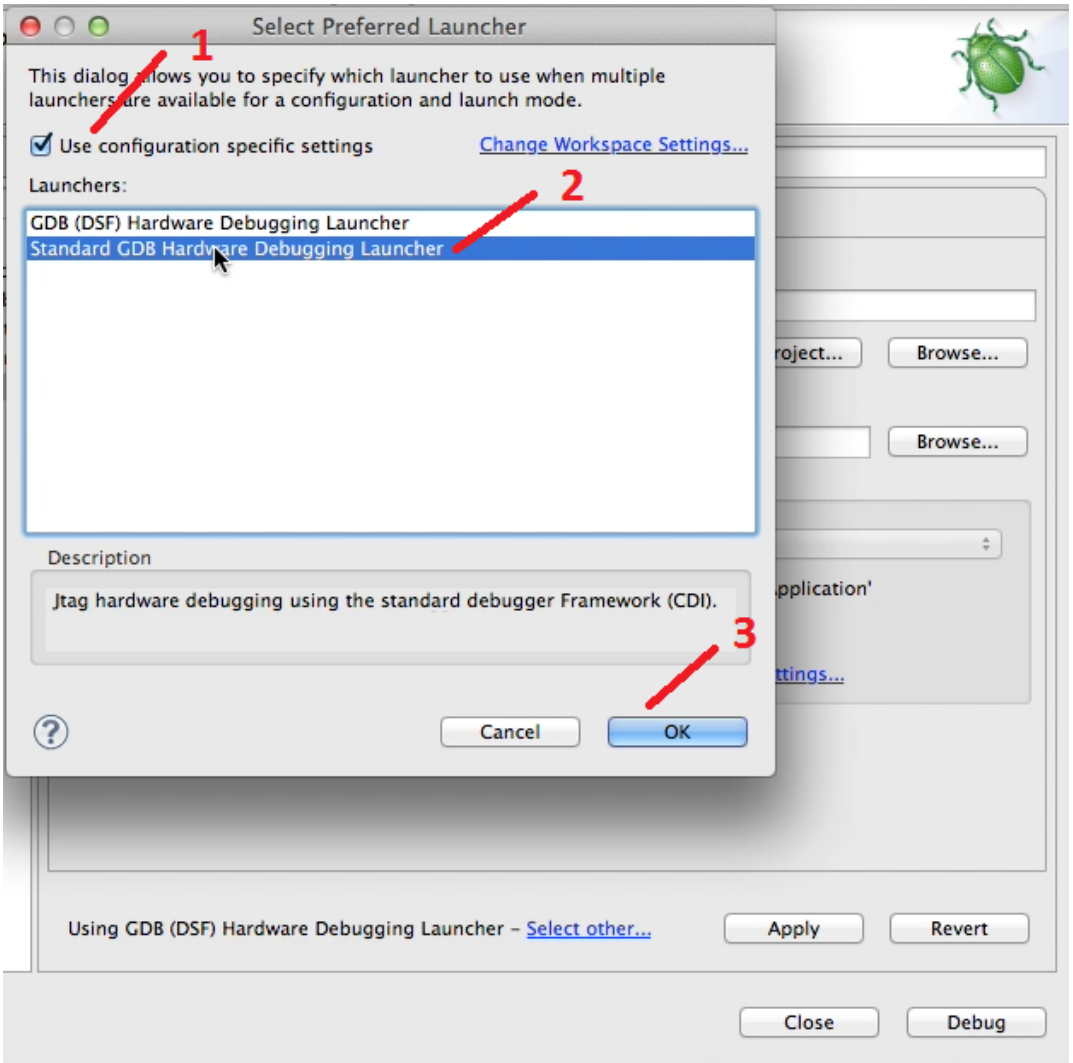
2. Double-click the **GDB Hardware Debugging** to create a new debug configuration. The debug configuration should be populated with settings from the current project.



3. Change the debug configuration name to *blinky (Flash and Debug)*, as seen in label mark 1 in picture. Click **Enable auto build** in the **Build configuration** section to enable builds to automatically happen (if needed) when the debug button is pressed, as seen in label mark 2. Click the **Select other...** link, as seen in label mark 3, to configure the GDB Hardware Debugging Launcher.

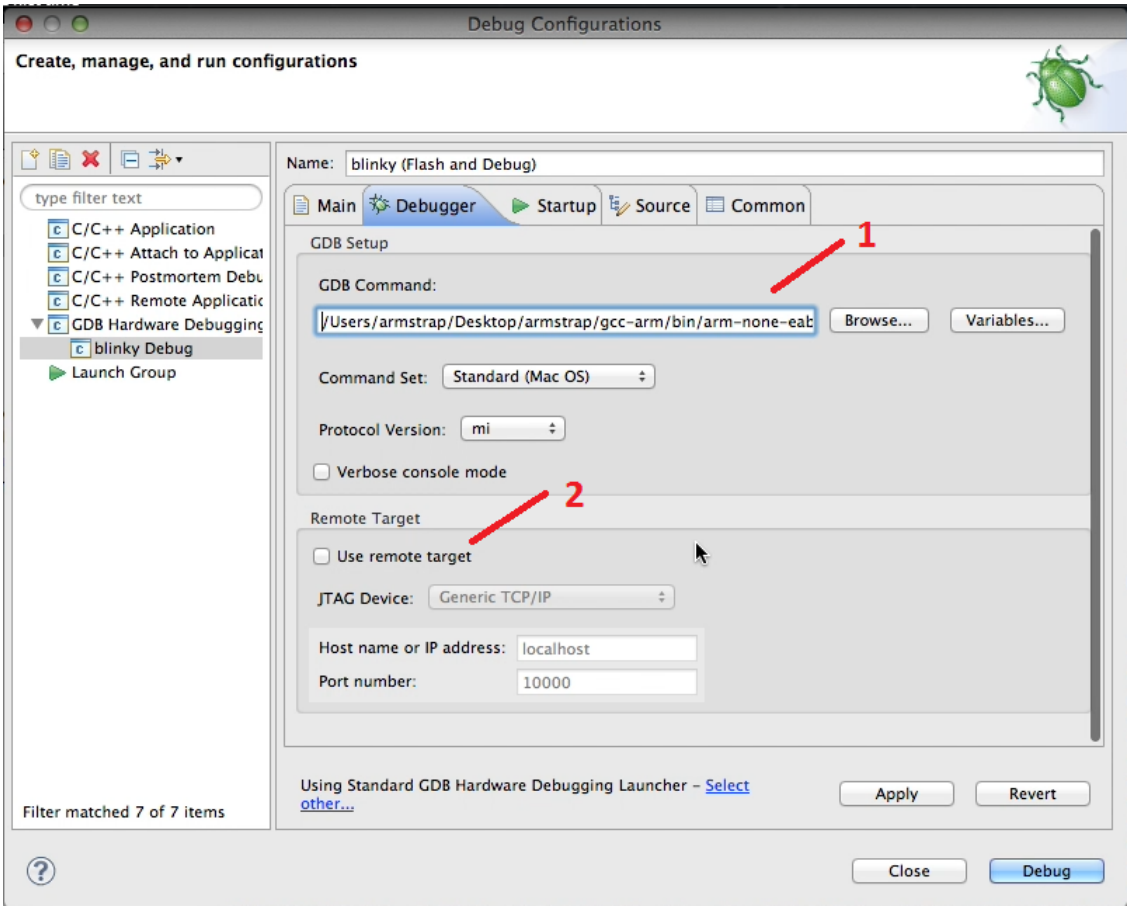


4. Check **User configuration specific settings** option as seen in label mark 1 in picture. Select **Standard GDB Hardware Debugging Launcher** in the list of Launchers. Click the **OK** button to complete the GDB launcher configuration.



5. **In the Debugger tab,**

   - Under, **GDB Setup>>GDB Command**, enter the full path to the location of *arm-none-eabi-gdb* that was downloaded with GNU Tools for ARM Embedded Processors. This should be *<user>/Desktop/armstrap/gcc-arm/bin/arm-none-eabi-gdb*, as seen in label mark 1 in picture
   - Under **Remote Target**, uncheck **Use remote target**, as seen in label mark 2.



6. In the **Startup** tab, under the Initialization Commands:
   - Uncheck **Reset and Delay (seconds)** option
   - Check **Halt** option
   - For Apple Mac OSX machines, enter the following start-up script

```
target extended /dev/tty.usbmodem7B4078B1
monitor swdp_scan
attach 1
monitor vector_catch disable hard
set mem inaccessible-by-default off
set print pretty
```

   - For Ubuntu Linux machines, enter the following start-up script

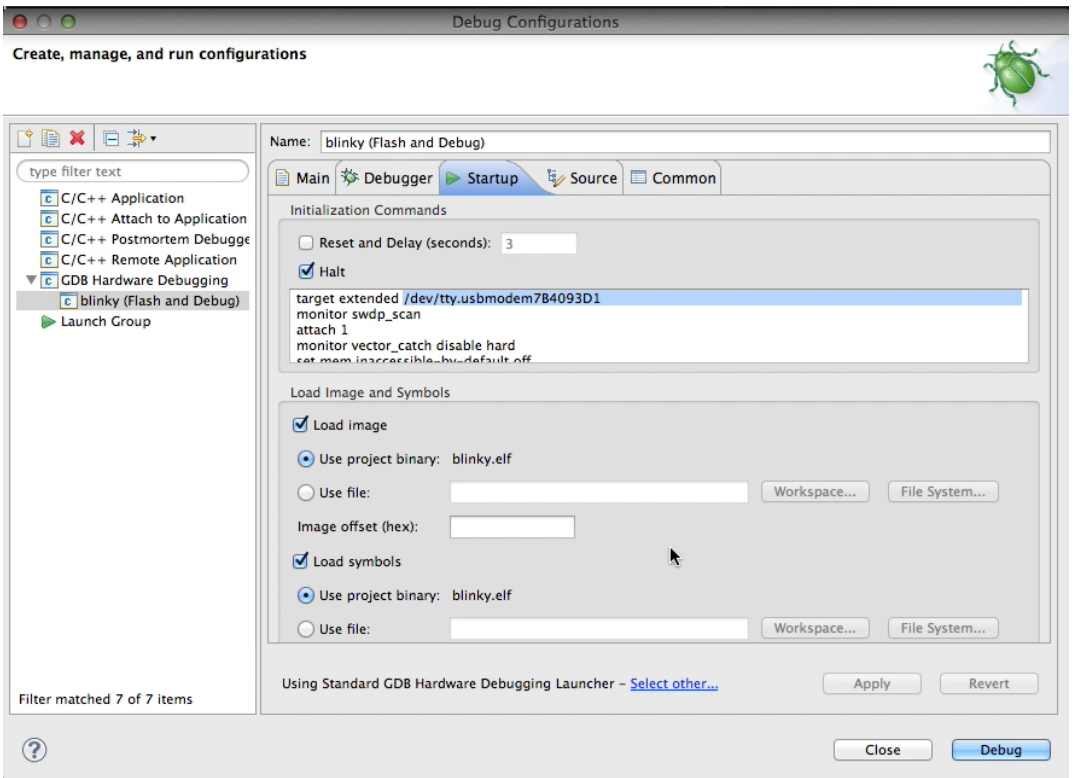
```
target extended-remote /dev/ttyACM0
mon swdp_scan
attach 1
monitor vector_catch disable hard
set mem inaccessible-by-default off
set print pretty
```

   - For Microsoft Windows machines, enter the following start-up script

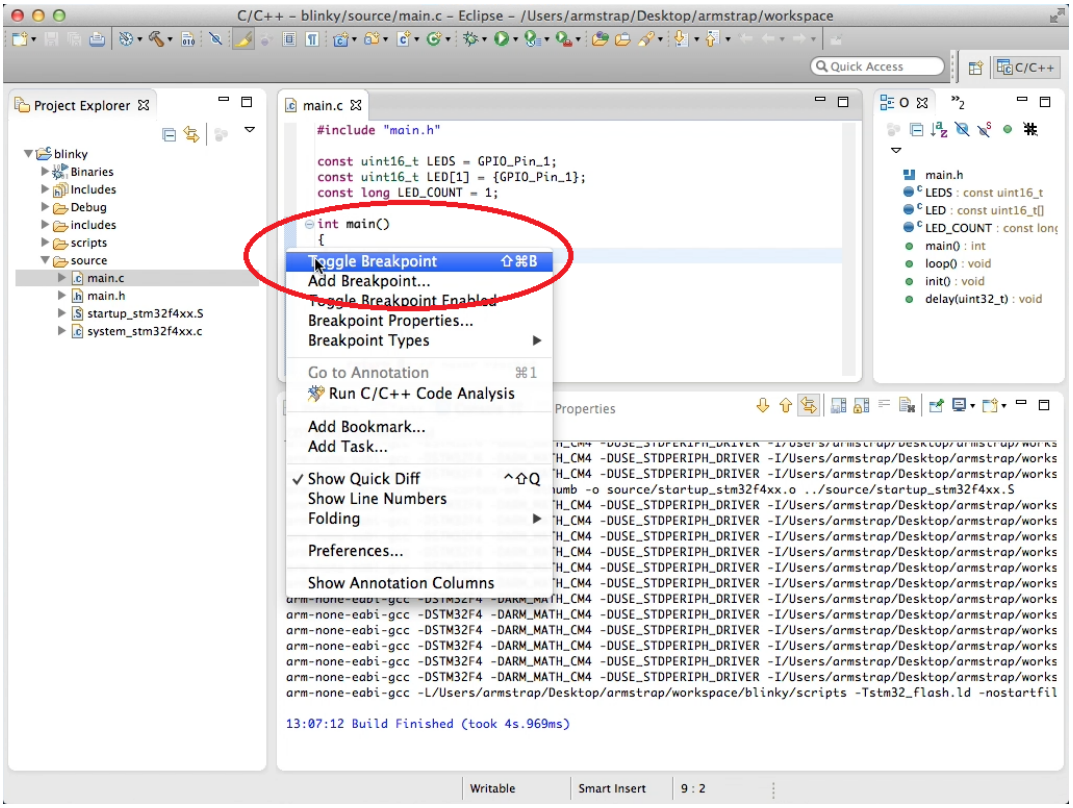
```
target extended-remote \\.\COM2
mon swdp_scan
```

```
attach 1
monitor vector_catch disable hard
set mem inaccessible-by-default off
set print pretty
```

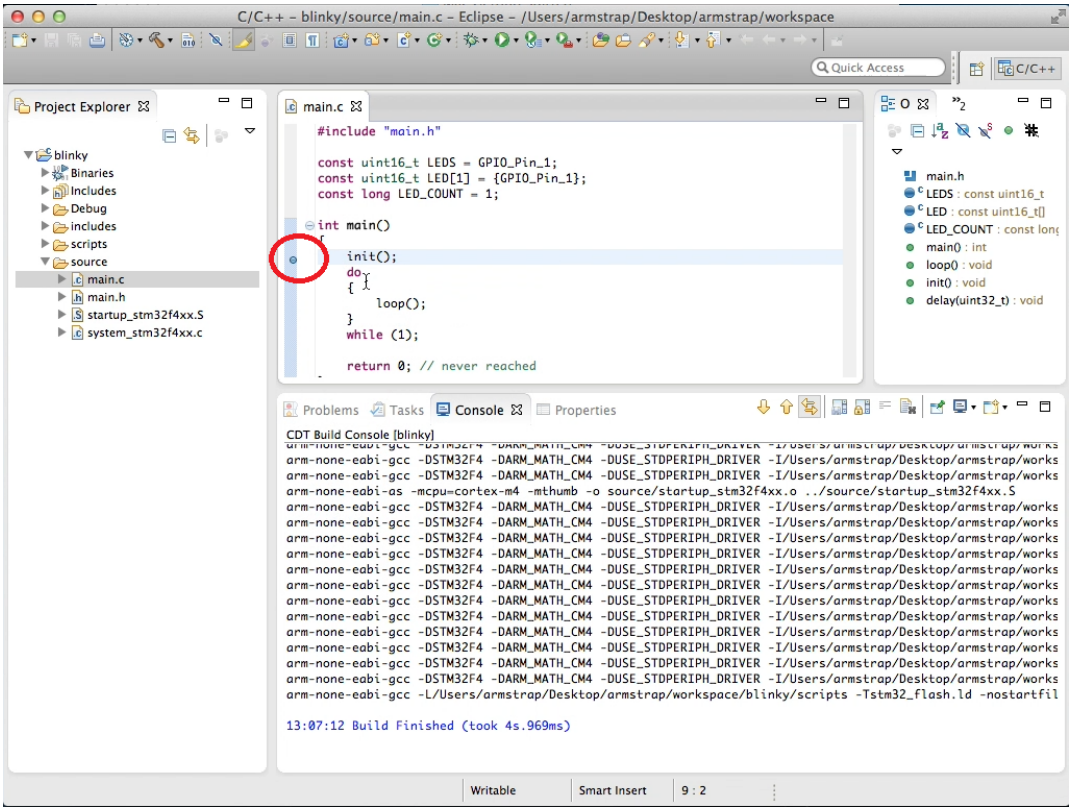
- Check **Load image** option and **Use project binary**
- Check **Load symbols** option and **User project binary**



7. Click the **Apply** button and the **Close** button to return to the C/++ perspective.
8. Open *main.c* from in the project find the first line inside the main() function. In this project, the first line is a call to init(). Right-click (or Ctrl-click on a Mac) on the margin to open a menu-item and select the **Toggle Breakpoint** menu option to set a breakpoint.

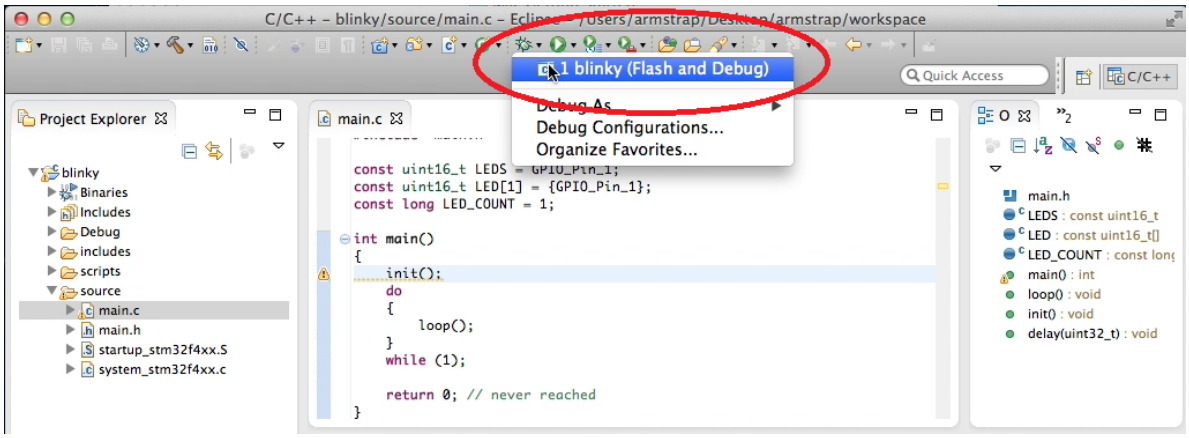


9. Verify the breakpoint is set by visually inspecting a blue dot in the margin.

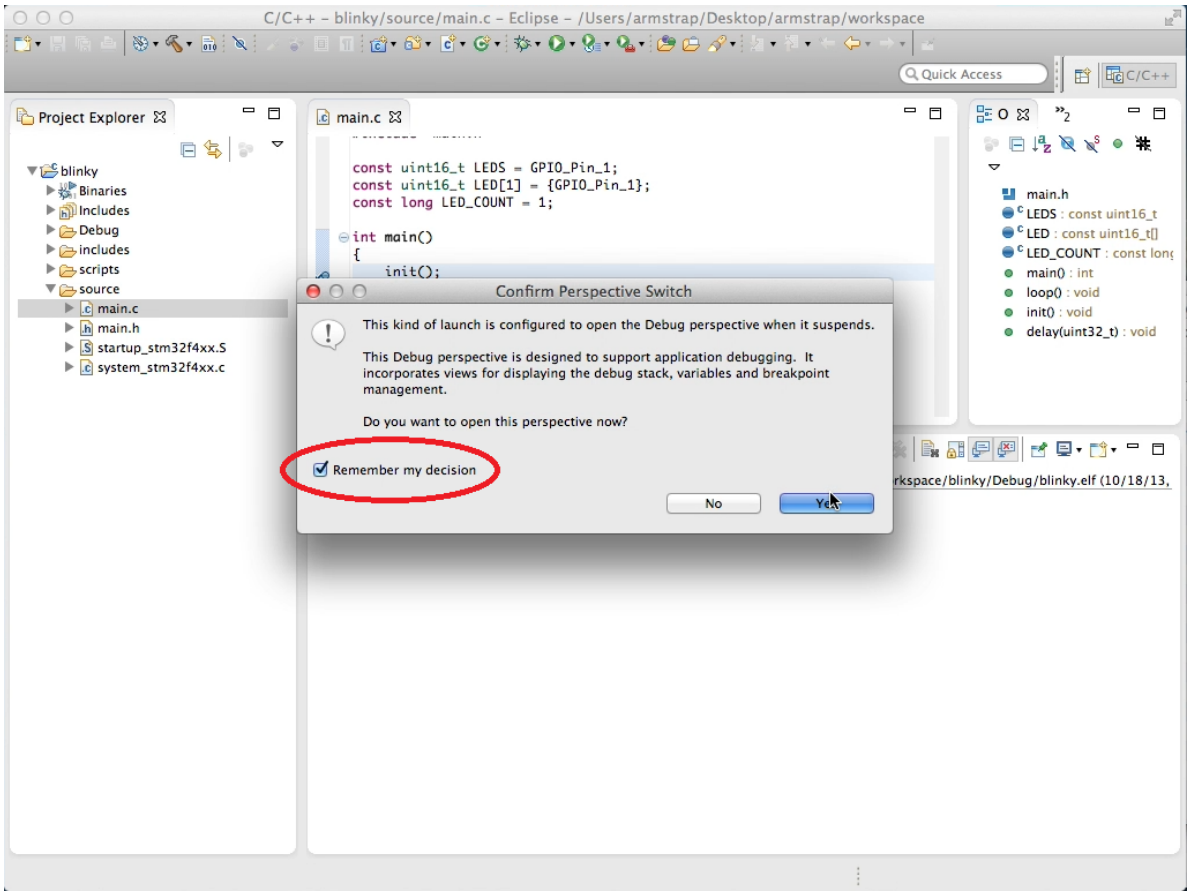


10. Click the debug toolbar and select your debug configuration to start flashing and debugging your Armstrap

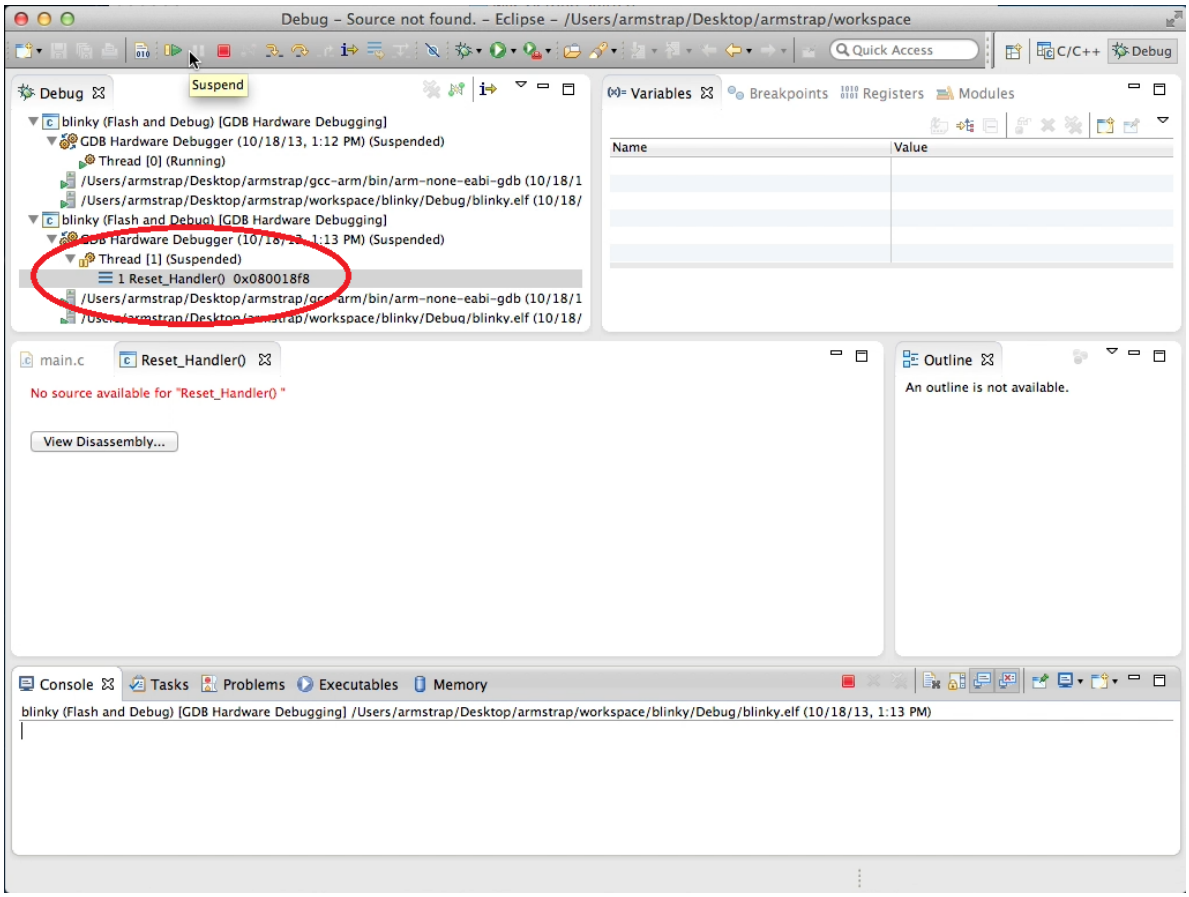
board.



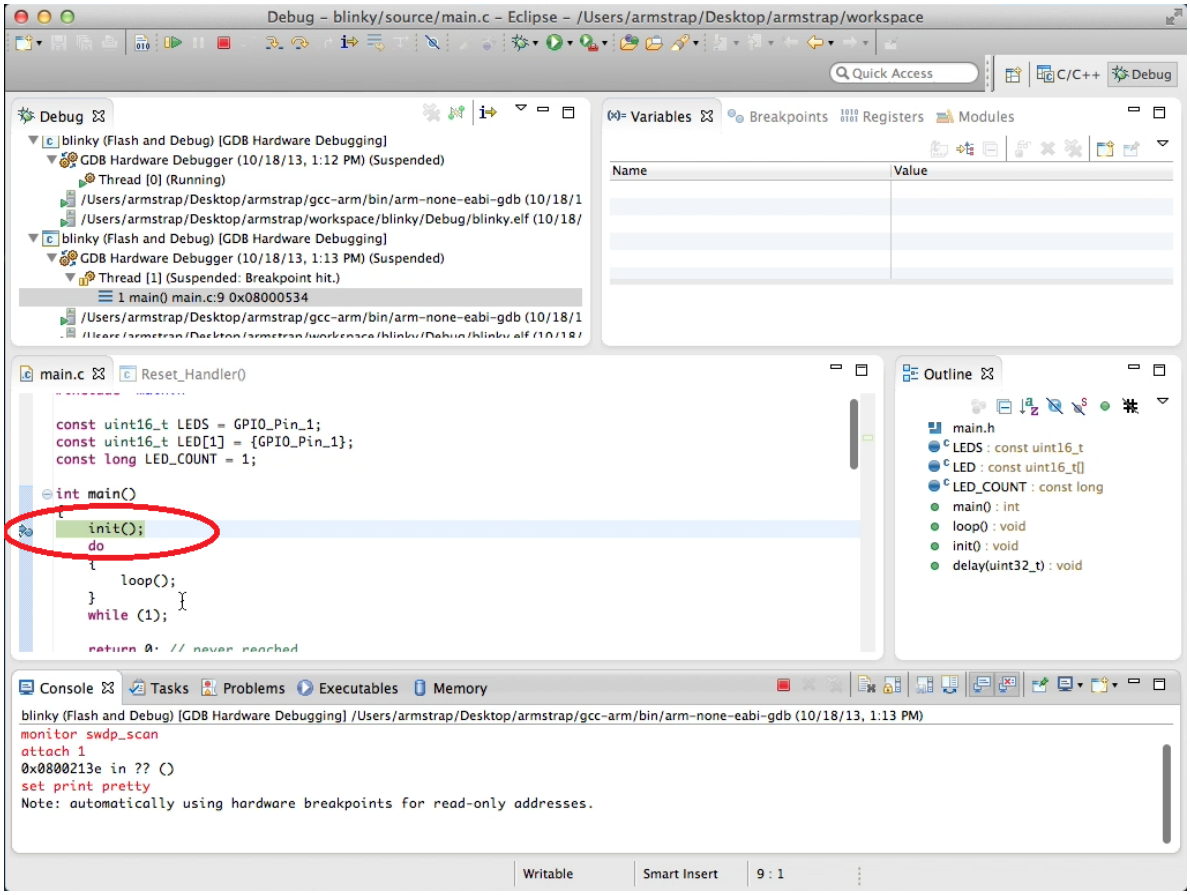
11. If this is the first time you are debugging, you may be presented with a confirmation dialog to confirm the perspective switch. Check the **Remember my decision** option and click the **OK** button.



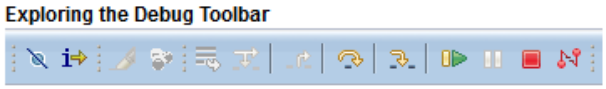
12. By default, Eclipse will halt on the first line of code, usually the *Reset_Handler*. Click on the **F8** key or the green **Play** button to continue.



13. The execution should stop at your breakpoint (as seen below) and you should be able to debug your target.



Exploring the Debug Toolbar



The Debug toolbar includes the following buttons:

Instruction Stepping Mode Enables instruction stepping mode to examine a program as it steps into disassembled code.

Drop to Frame Re-enters the selected stack frame in the Debug view.

Use Step Filters Enables step filters in the Debug view.

Step Return Continues execution to the end of the current routine, then follows execution to the caller of the routine.

Step Over Executes the current line, following execution inside a routine.

Step Into Executes the current line, including any routines, and proceeds to the next statement.

Resume Resumes execution of the currently suspended debug target.

Suspend Halts execution of the currently selected thread in a debug target.

Terminate Ends the selected debug session and/or process.

Disconnect Detaches the debugger from the selected process.

CHAPTER 4

Armstrap Naming and Versioning Convension

Format

Use the simple file naming convention when creating boards:

```
<author>_<board-name>_<version>.<extension>
```

<author>=([0-9][a-z]\-)*

- lowercase-alpha-numeric string which can contain dash characters
- all uppercase characters convert to lowercase characters
- space characters convert to dash '-' characters
- invalid characters are ommited
- usually a company name, username or website domain name

<board-name>=([0-9][a-z]\-)*

- lowercase-alpha-numeric string which can contain dash characters
- all uppercase characters convert to lowercase characters
- space characters convert to dash '-' characters
- invalid characters are ommited
- usually the name of the product

<version>=[0-9]*.[0-9]*.[0-9]*

- three numbers with a period separator
- <major-version>.<minor-version>.<micro-version>

<extension>=(\.brd|\.sch)

- the given extension of the PCB editor program (*brd* and *sch* are the extension for Cadsoft EAGLE files)

Rules

- There are exactly two underscore '_' characters (aka delimeters) in the filename
- **author** must changes when a board forks from one owner/company to another owner/company
- **board-name** usually never changes but can change at time of forking to preserve original 'author'
- **version** must change when a board is submitted to manufacturing
- **version** does not change when **author** changes on a newly forked board
- **board-name** and **version** must exist on the board silkscreen layer (top or bottom).
- *[optional but highly recommended]* **author** must exist on the board silkscreen layer (top or bottom)
- **major-version is incremented when:**
  - the size (dimensions) of the board changes
  - the interface (connectors) to the board change
- **minor-version is incremented when:**
  - any SMT chip is added
  - any SMT chip is removed
  - any SMT has moved
- **micro-version is incremented when:**
  - the silkscreen layer is modified
  - a trace is modified
- **major-version** changing resets **minor-version** and **micro-version** to zero
- **minor-version** changing resets **micro-version** to zero

Remarks

- **major-version** changing usually requires a solder-wave/selective-solder re-tooling, re-stenciling and is the most expensive change.
- **minor-version** changing usually requires no solder-wave/selective-solder re-tooling, but requires re-stenciling.
- **micro-version** changing usually requires no solder-wave/selective-solder re-tooling, no re-stenciling and is generally the cheapest change

Workflow Example

The Armstrap Eagle Board:

```
armstrap_eagle_1.0.0.brd
```

Note: armstrap-org_eagle_1.0.0.brd is also acceptible

The company "VOV Technology" forks the board, keeps armstrap branding but adds its own company logo:

```
vovtech_armstrap-eagle_1.0.0.brd
```

Note: vovtech-com_armstrap-eagle_1.0.0.brd is also acceptible

The company "VOV Technology" later changes changes a chip on the board but maintains the same board size and interface:

```
vovtech_armstrap-eagle_1.1.0.brd
```

The company "VOV Technology" later discovers a silkscreen naming problem and makes a minor change to the silkscreen layer:

```
vovtech_armstrap-eagle_1.1.1.brd
```

The Armstrap community intergrates VOV's changes, removed the VOV branding:

```
armstrap_eagle_1.1.0.brd
```

Community member 'John Smith' forks the Armstrap Eagle board into his own source code repository:

```
john-smith_armstrap-eagle_1.1.0.brd
```

Naming Examples

- Valid Board Names:

```
adafruit-com_mintyboost_3.0.0.brd
sparkfun_weather-shield_1.0.0.brd
netduino-com_netduino_plus_2.0.1.brd
```

- Invalid Board Names:

```
mark's_arduino-motor-sheild_2.1.0.brd           // author contains invalid␣
↪apostrophe character
arduino-bluetooth-module_1.0.0.brd              // does not have exactly two␣
↪underscore '_' characters
supermechanical_twine_1.0.brd                   // missing micro-version
SparkFun.com_Current-Sensor-Breakout_2.1.0.brd  // author and board name must be␣
↪lowercase, invalid '.' character in author
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search