# Configuring Ubuntu to Code for the OmniFlash or OmniEP

## Table of Contents

## Introduction

This document describes the steps for setting up Ubuntu to develop and code for JK Micro's OmniFlash and OmniEP embedded devices (See http://www.jkmicro.com).

## Assumptions

It is assumed the reader has a working Ubuntu operating system. For instructions on how to set this up, please see the document entitled "Installing and configuring Ubuntu Linux.docx". It goes over in detail exactly how to get up and running in Ubuntu Linux. It is also assumed that the following packages have been installed via the Synaptic Package Manager.

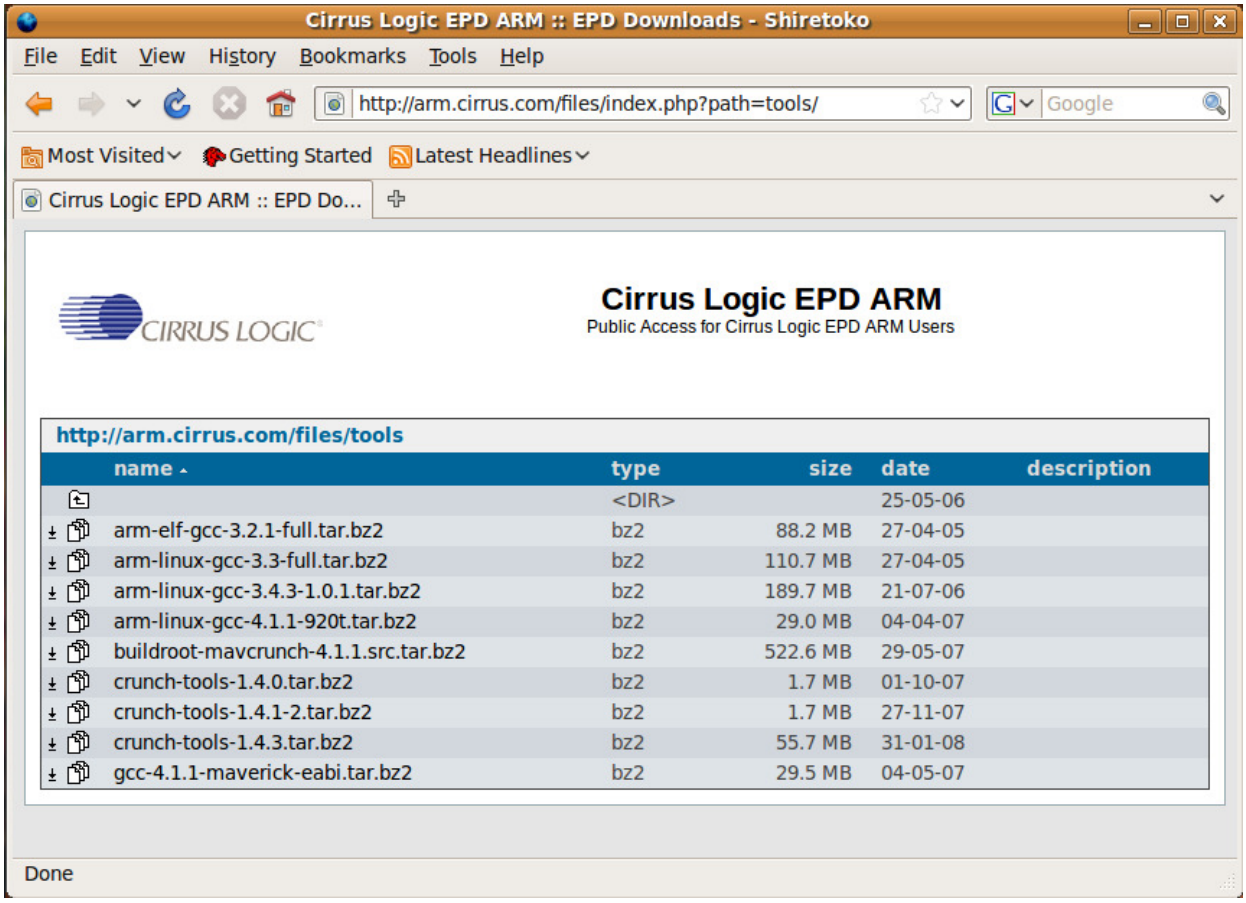
1.) **codeblocks**
2.) **codeblocks-contrib**
3.) **g++**
4.) **cutecom**
5.) **lrzsz**
6.) **Putty** and/or **gtkterm**
7.) **xutils-dev** (if you want to build from command line and use makedepend)

## Getting Started

We need to download the cross compiler for the OmniFlash and OmniEP as well as configure Code::Blocks to compile our programs.

## Getting the Cross Compiler for ARM

We can download the cross compiler from Cirrus Logic at this website:
http://arm.cirrus.com/files/index.php?path=tools/

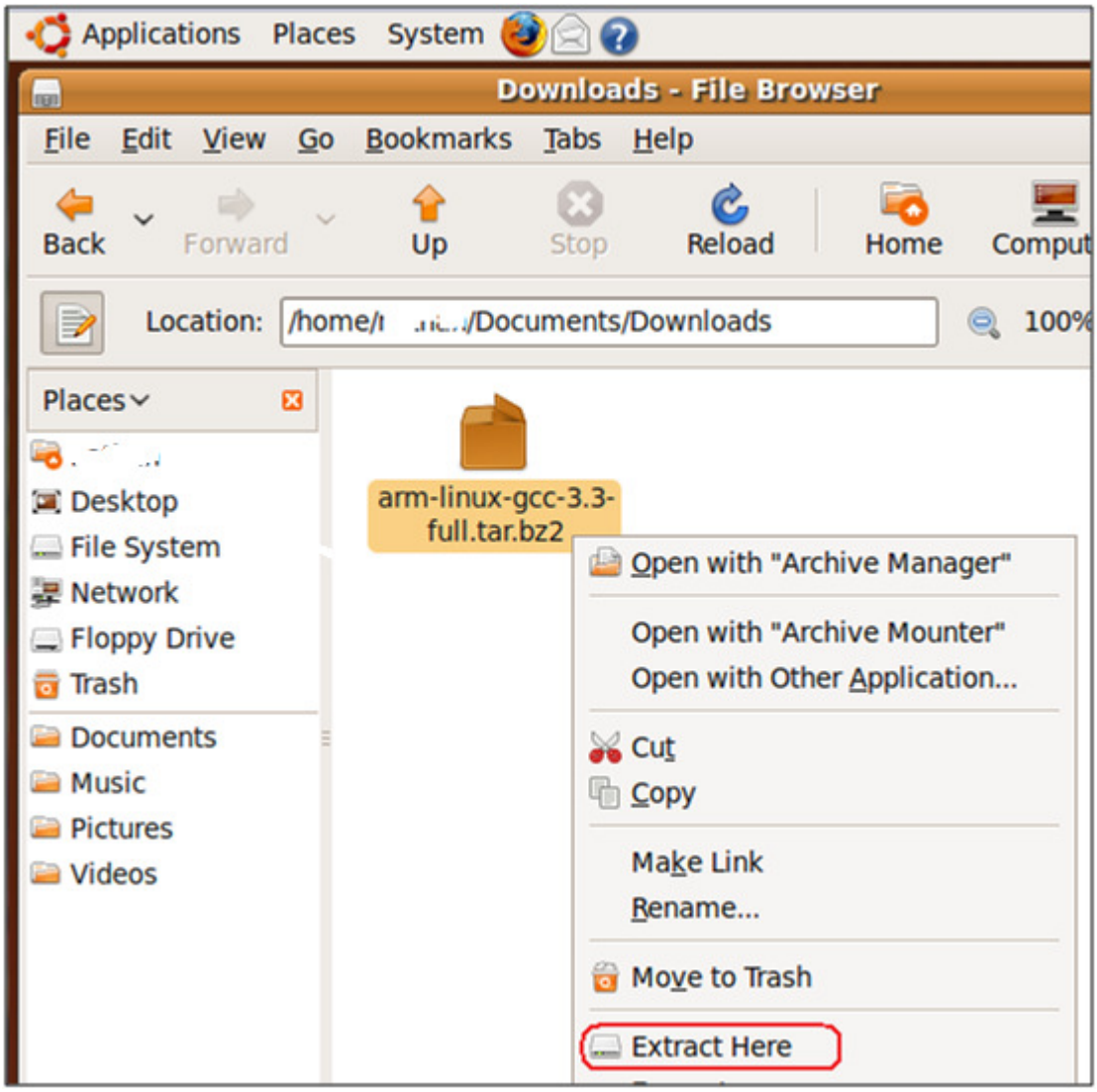



For the **OmniFlash**, download the "**arm-linux-gcc-3.3-full.tar.bz2**" file. For the **OmniEP**, download the file "**arm-linux-gcc-4.1.1-920t.tar.bz2**". I haven't played around too much with compiling for the OmniFlash with the 3.43 or 4.1.1 compiler yet. It may work.

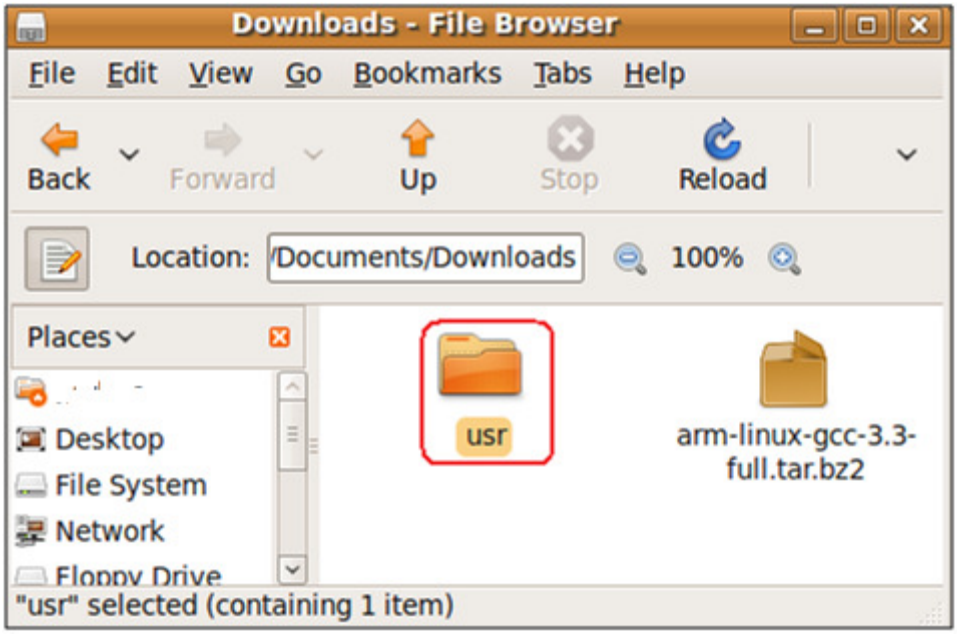
This document will cover the installation steps for the OmniFlash. The steps for OmniEP are the same, just with a different compiler (the 4.1.1 version).

Save the file somewhere on your system. It defaults to your desktop. You can change this in the options of Firefox.

## Extracting the contents of the compressed file



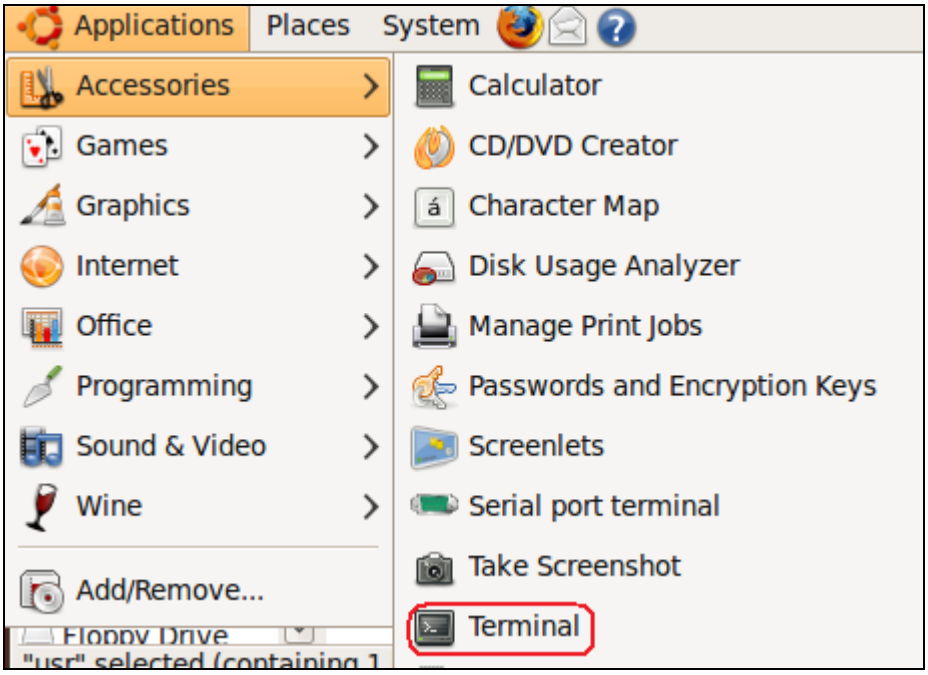

Navigate to where you saved your file. Right-click on it and click Extract here.



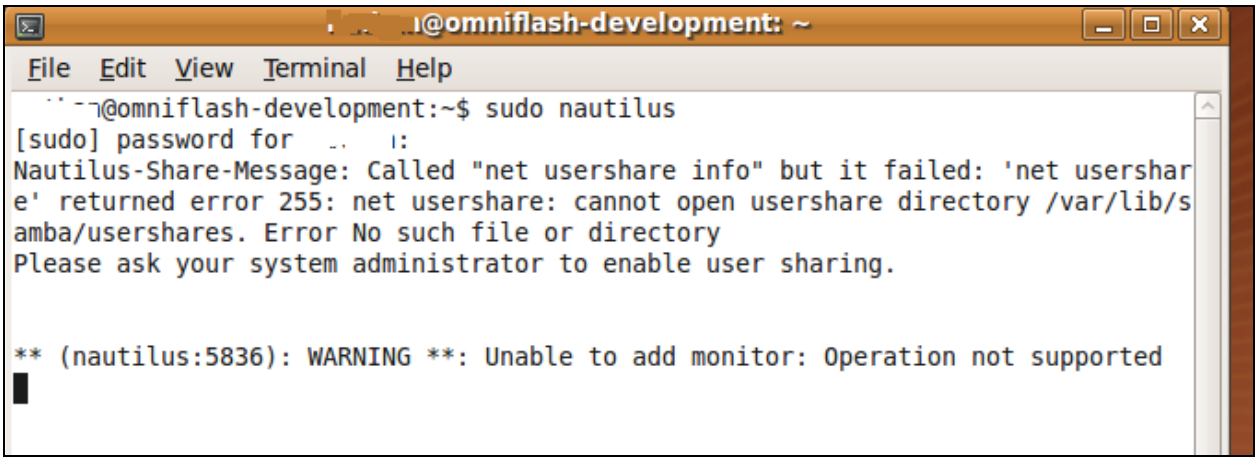

Once extracted, you should have a folder like the one above called usr.

## Copying the compiler into the proper place

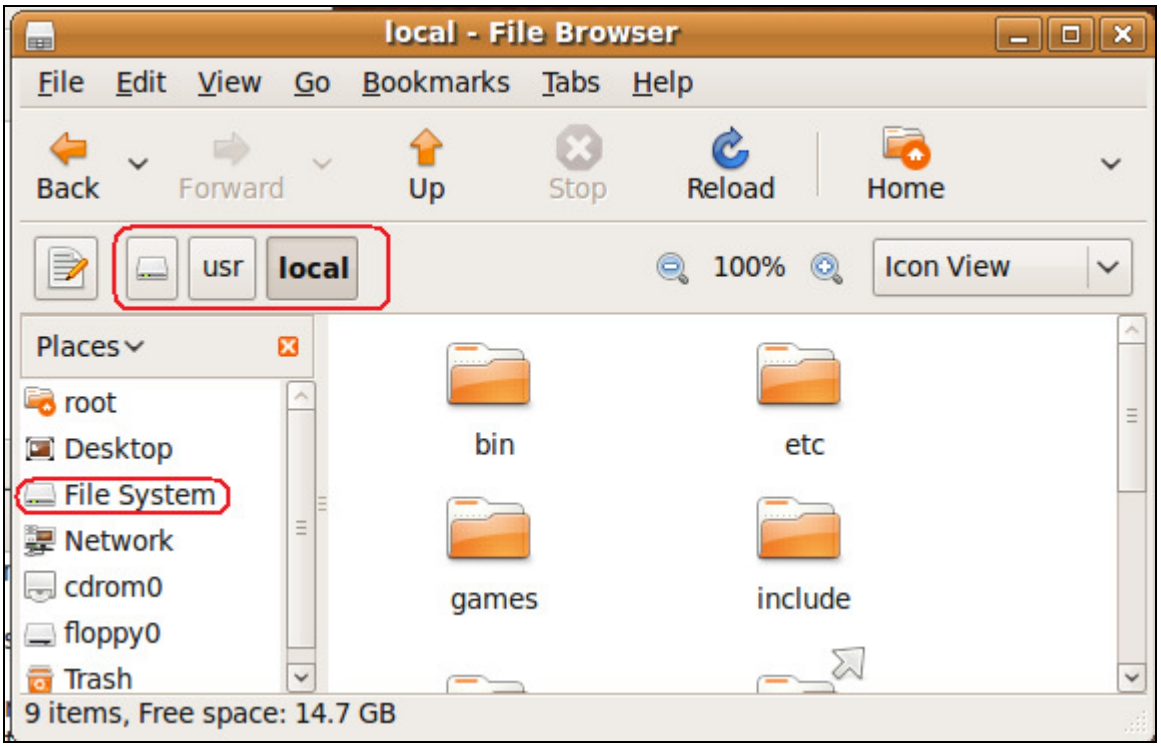
We need to copy the contents to a location where we can access it globally.




Click on Applications->Terminal to get a terminal session.



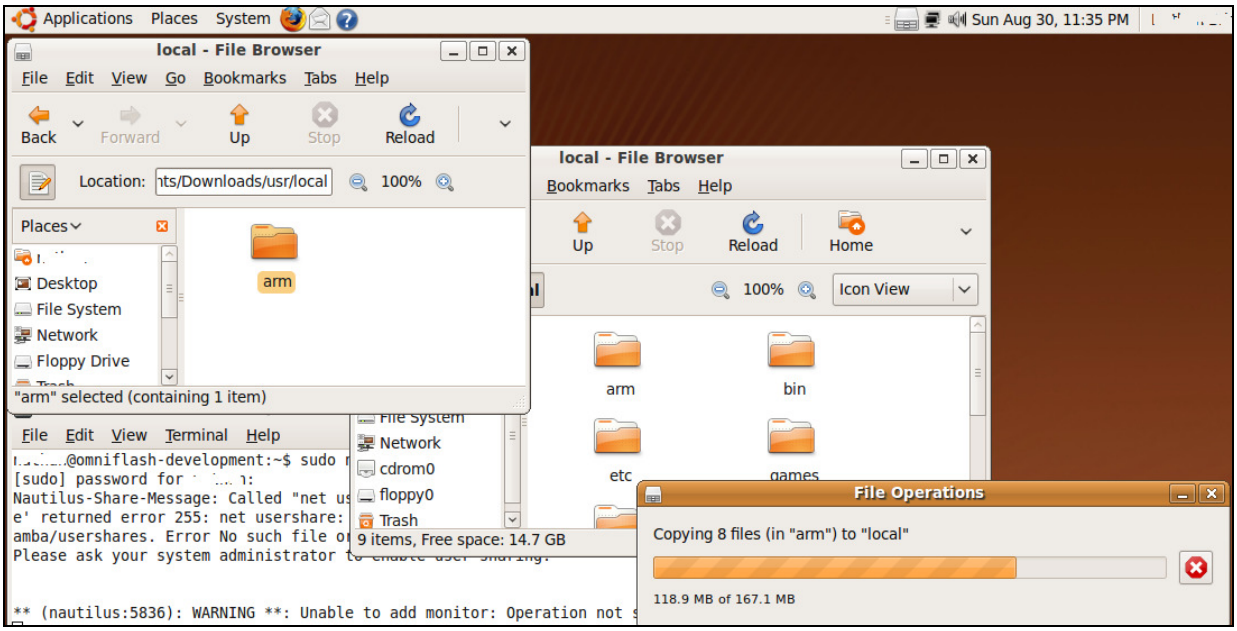

Type **sudo nautilus** to bring up a file explorer that has root access. It is important that it has root access so we can copy files to a protected folder.



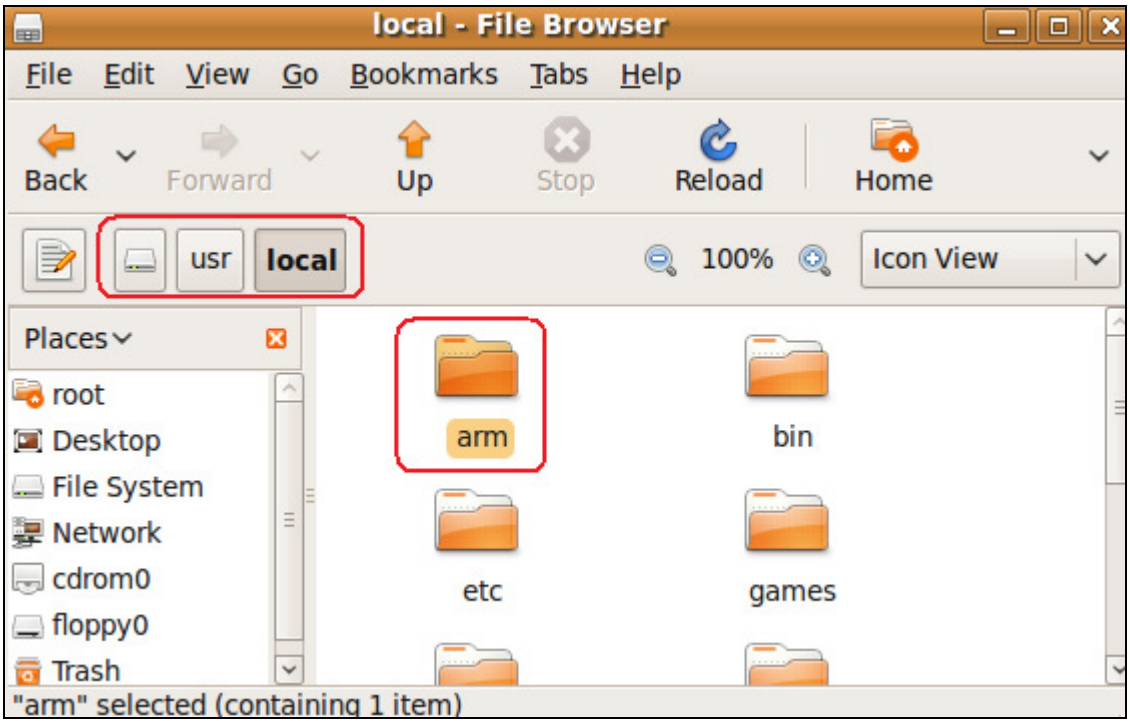

From the new File Browser that just opened up, navigate to the /usr/local folder.

In the other folder where we extracted the compiler to, navigate inside the **usr** folder and then inside the **local** folder. You fill find an **arm** folder.

Copy the **arm** folder (by dragging it over) into **the /usr/local** folder.



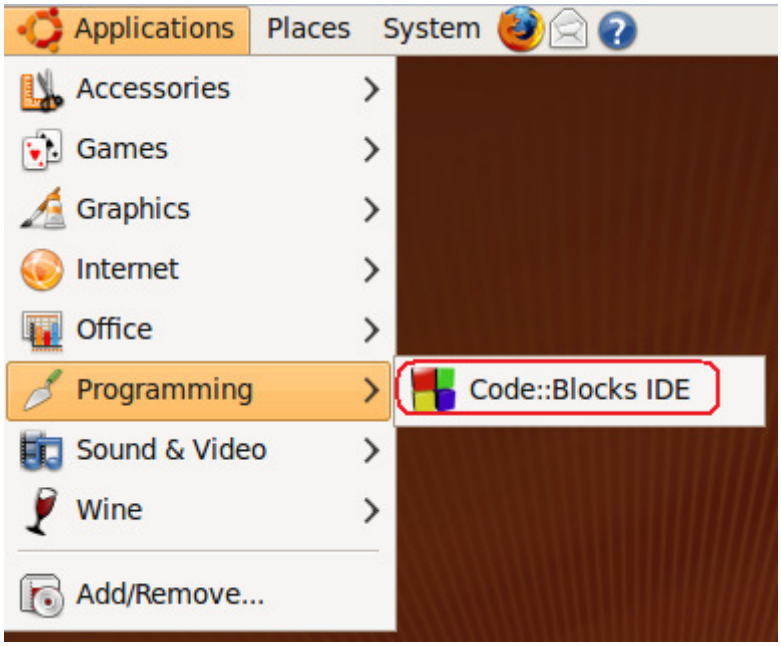

Wait for the copy process to complete.



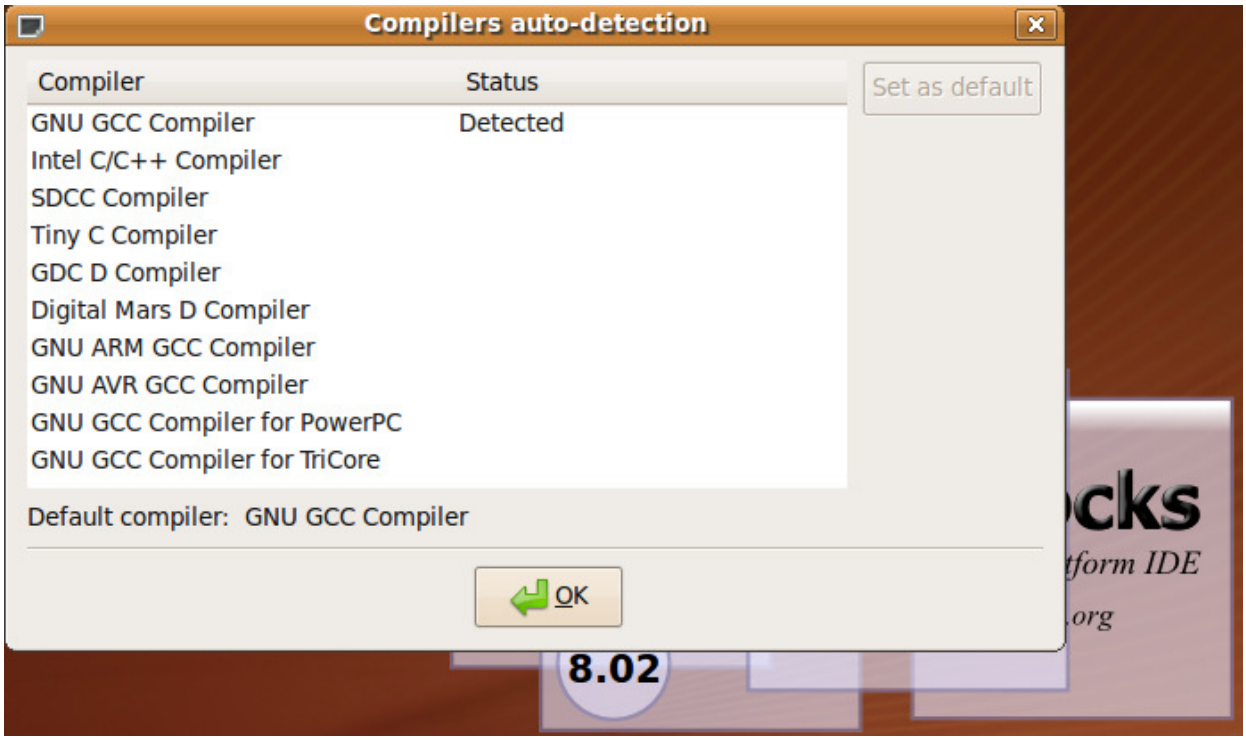

You should now have an arm folder under /usr/local. This is the cross compiler which we will use to set up Code::Blocks.

## Now close all open windows.

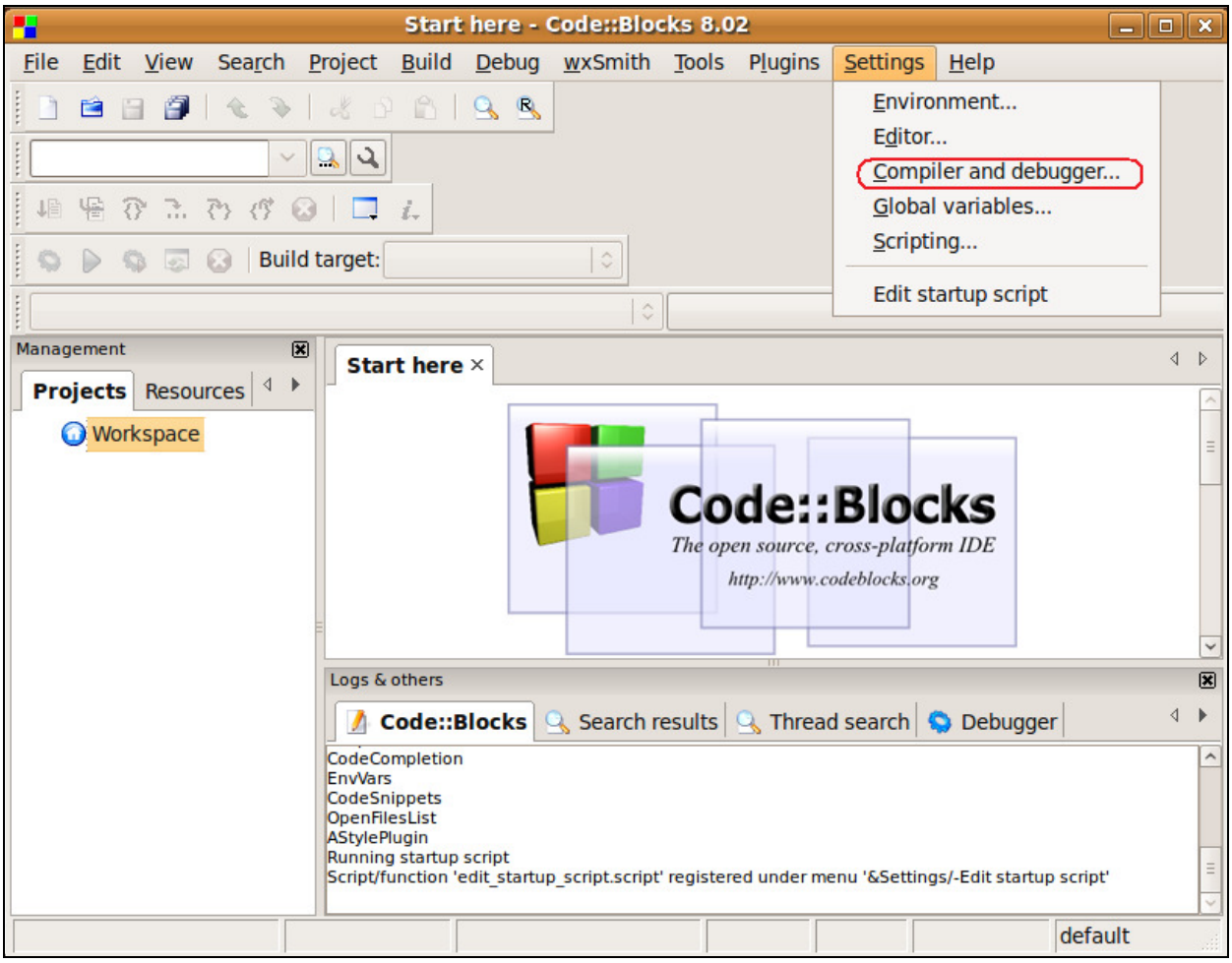
## Configuring CodeBlocks to use the Cross Compiler



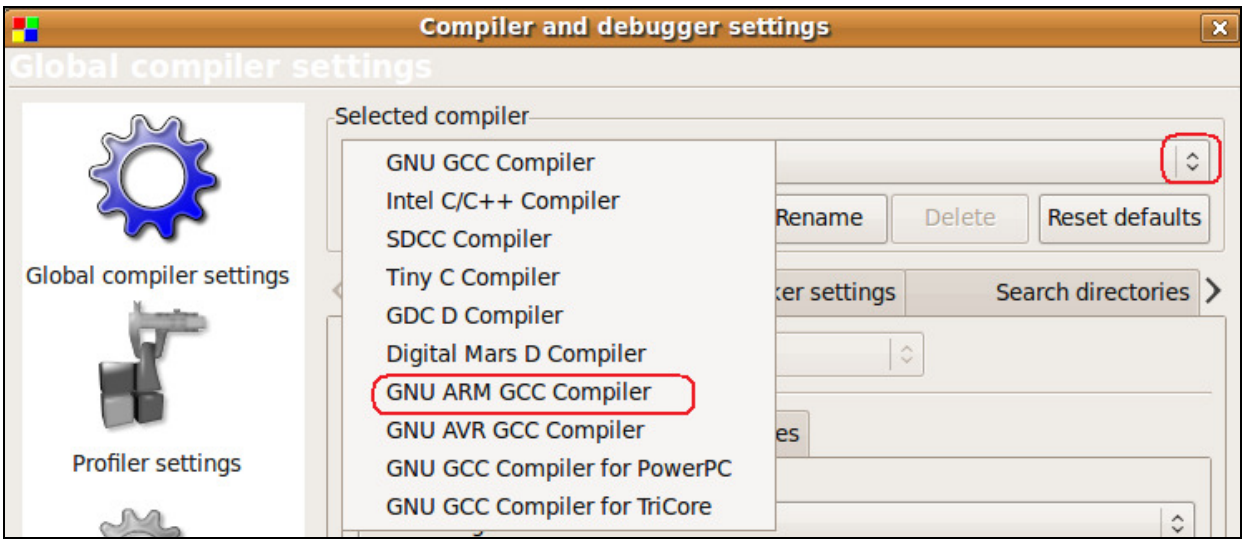

Launch the CodeBlocks IDE.



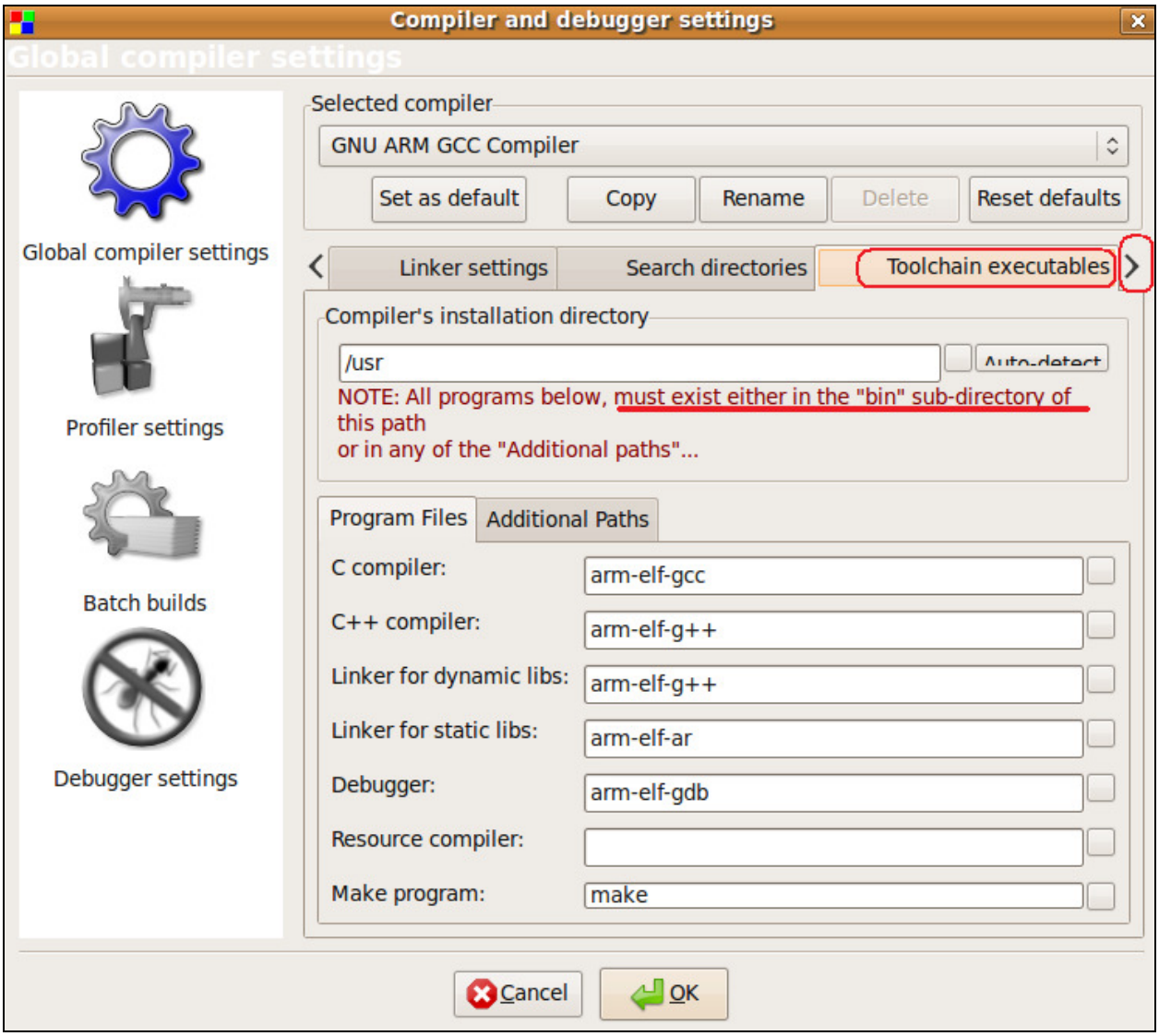

Click OK to this window. It only shows up the first time. We will configure the ARM GCC compiler.



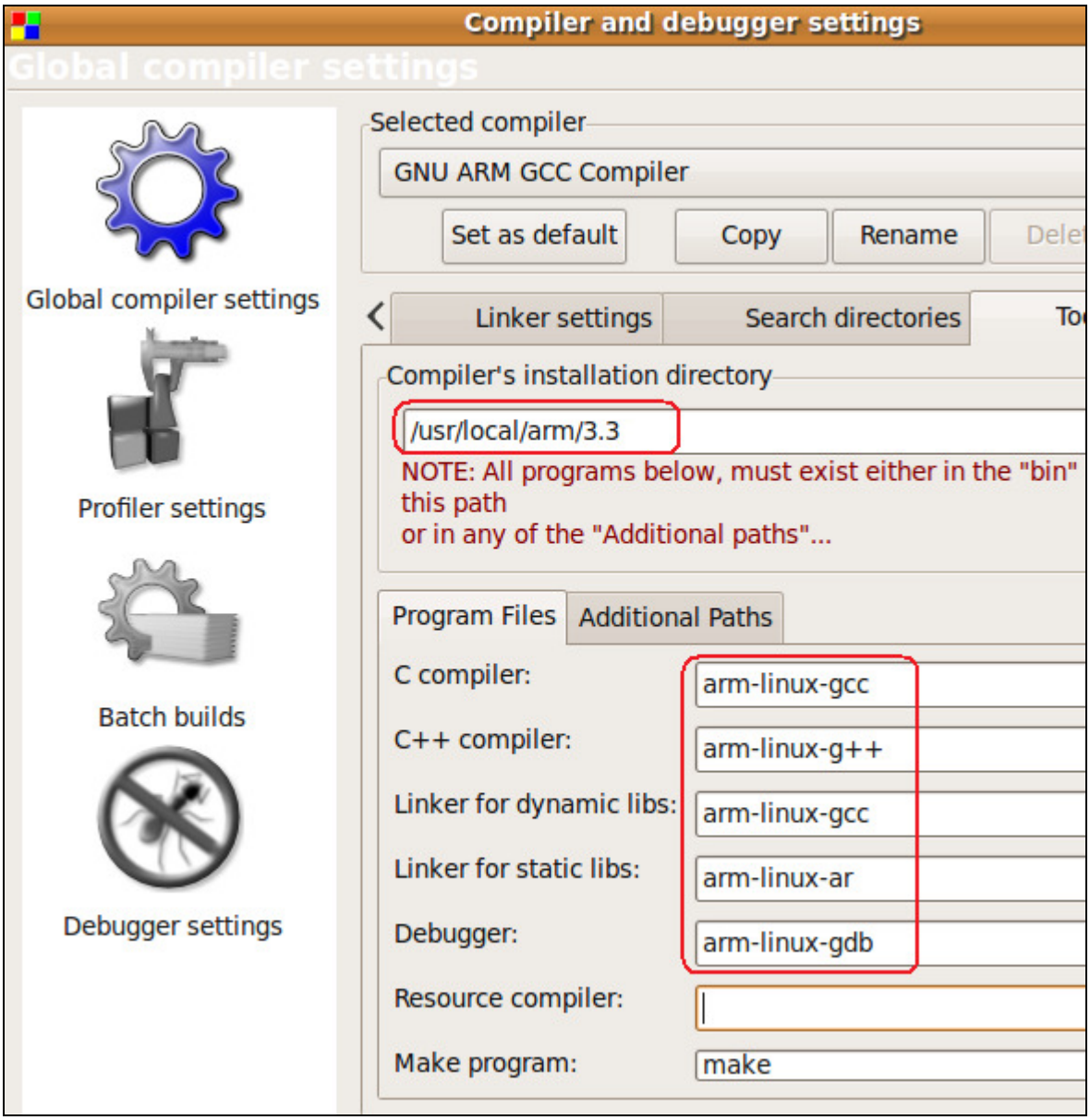

Click on Settings -> Compiler and debugger...




Click the pull down menu for the compilers and choose the GNU ARM GCC Compiler



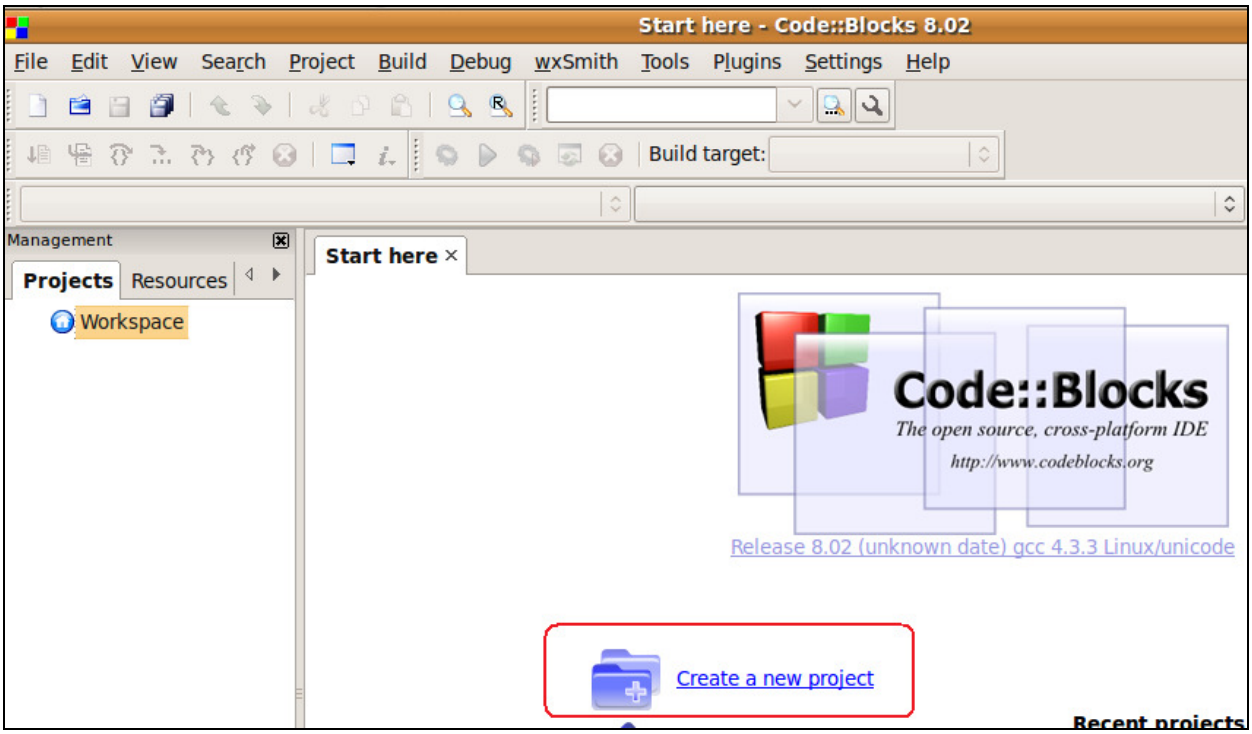

Click the arrow to the right of the tabs until you come to the Toolchain executables. The defaults listed are wrong. We need to specify what to use.



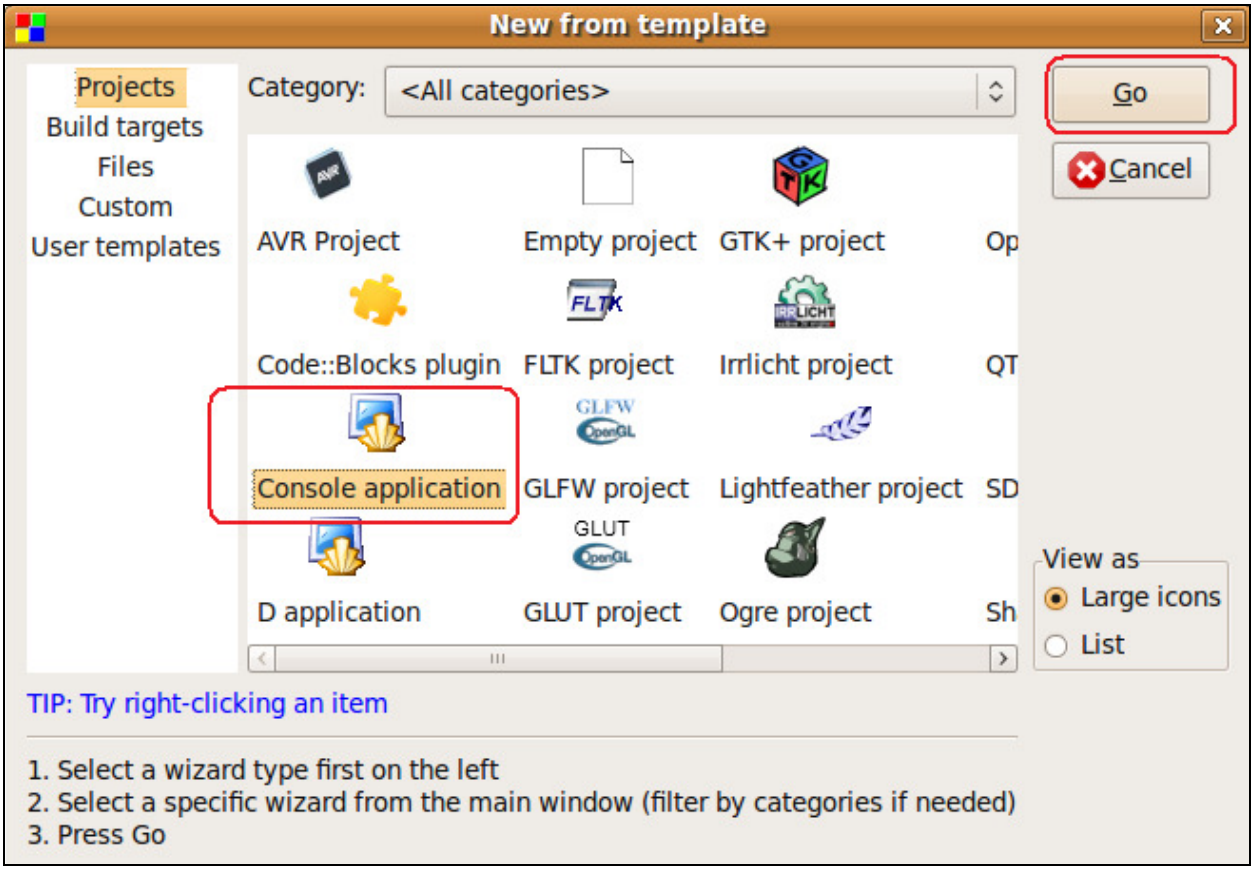

Change the compiler's installation directory to **/usr/local/arm/3.3** . If you installed a different compiler, select it's directory MINUS the bin directory. The program appends the bin directory automatically to the path you enter here. The compiler used for dynamic libs is not a mistake. We cannot use the C++ compiler for libraries on this platform for some reason. It doesn't work (or I haven't found out how to make it work yet).

There are lots of other settings for Code Blocks. I recommend you go through each screen and set it up to your liking.

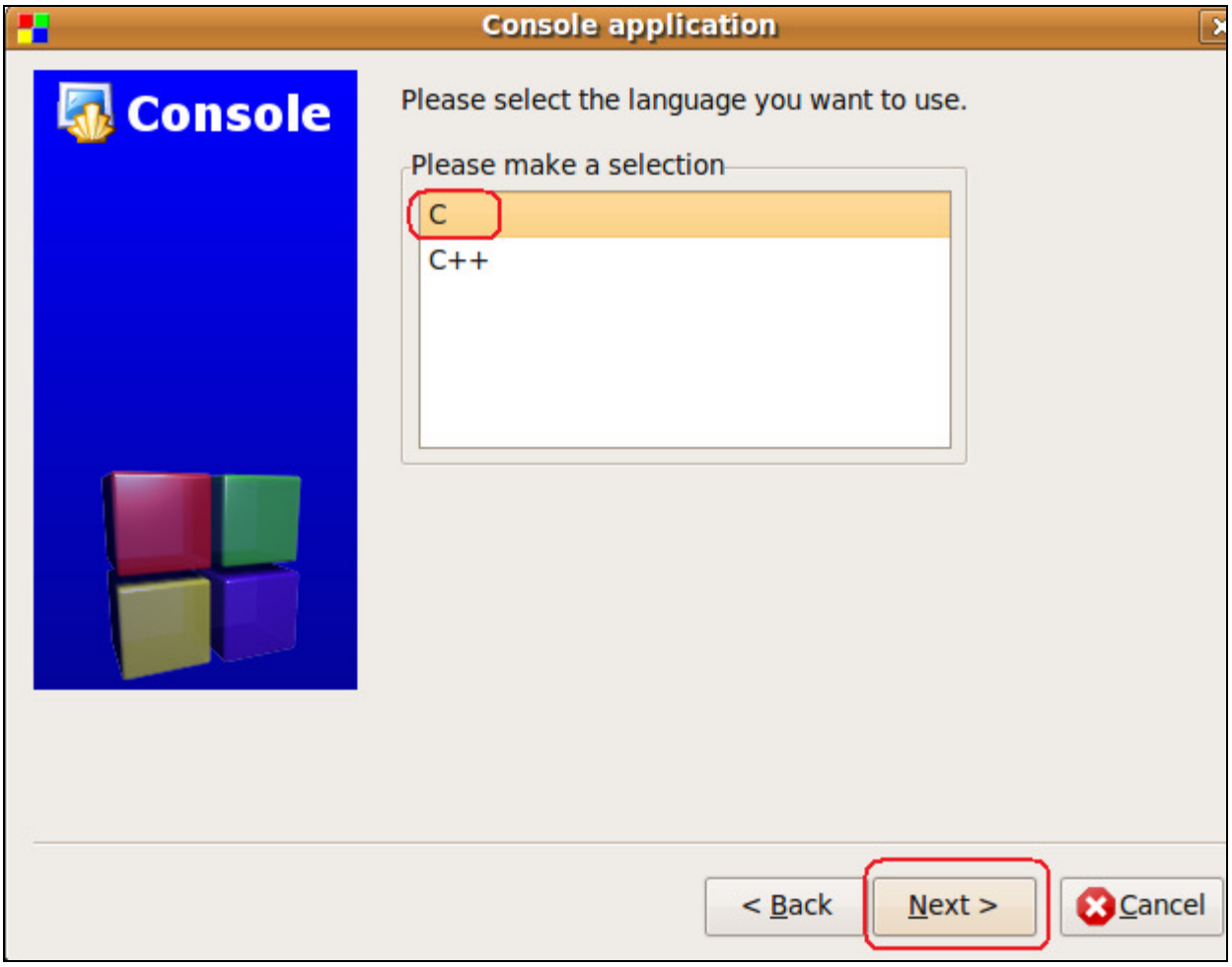
## Starting a new Project in CodeBlocks



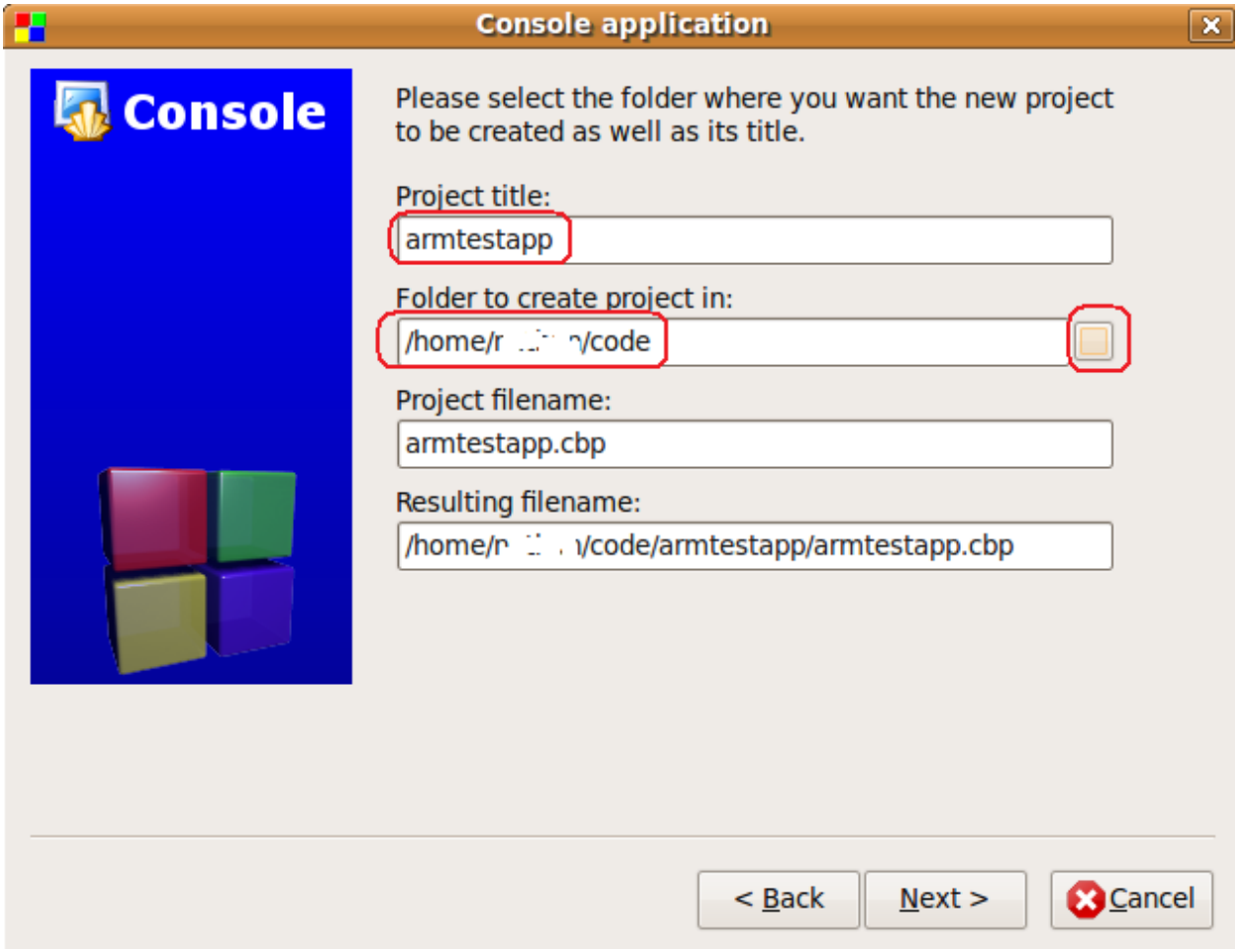

From the main window of CodeBlocks, click the Create a new project button.



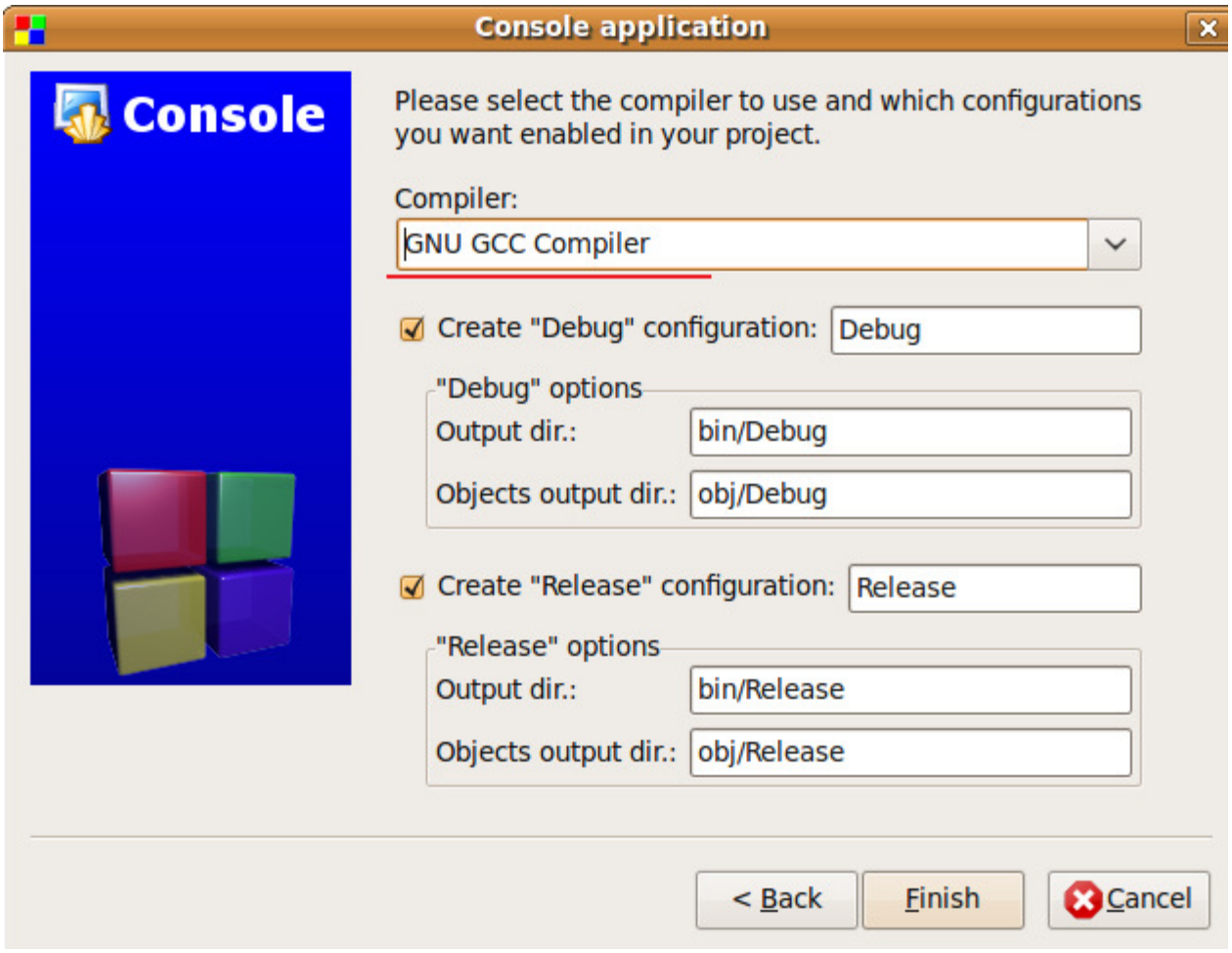

Click the Console application and click GO.




Click the C language and Next.

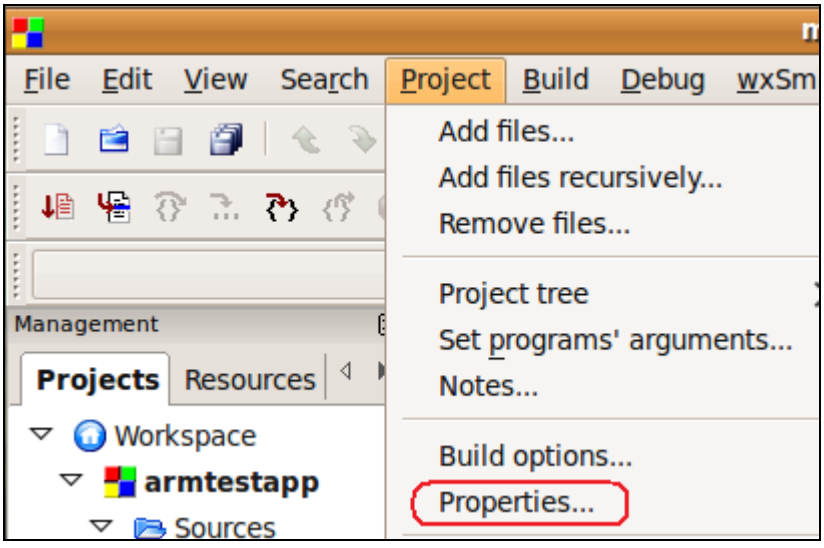



Give your project a name. Click the ellipses button next to the folder to create your project in and navigate to a folder you want to work in. Verify the resulting filename and click Next.

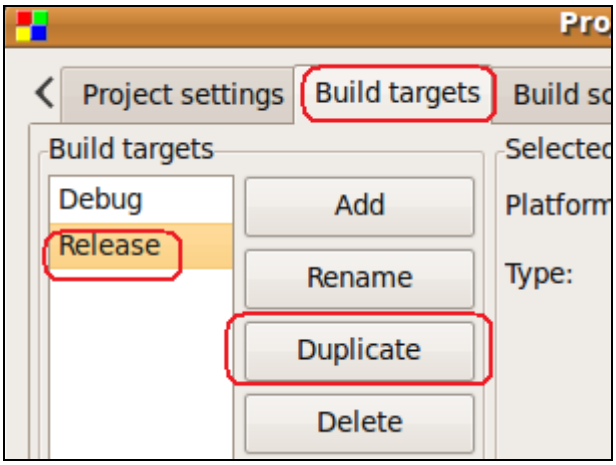



For now, accept the defaults. We will fill in the ARM compiler section later. Click Finish.

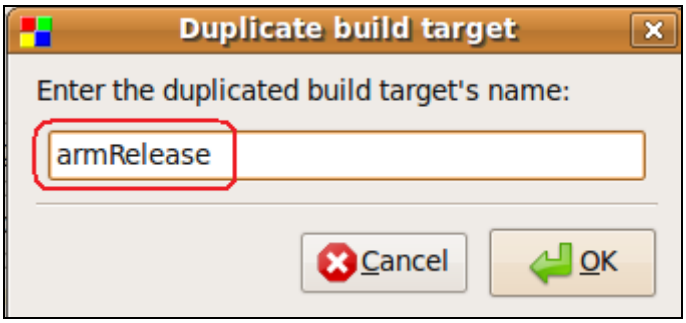
## Adding a new Build Target for the ARM processor



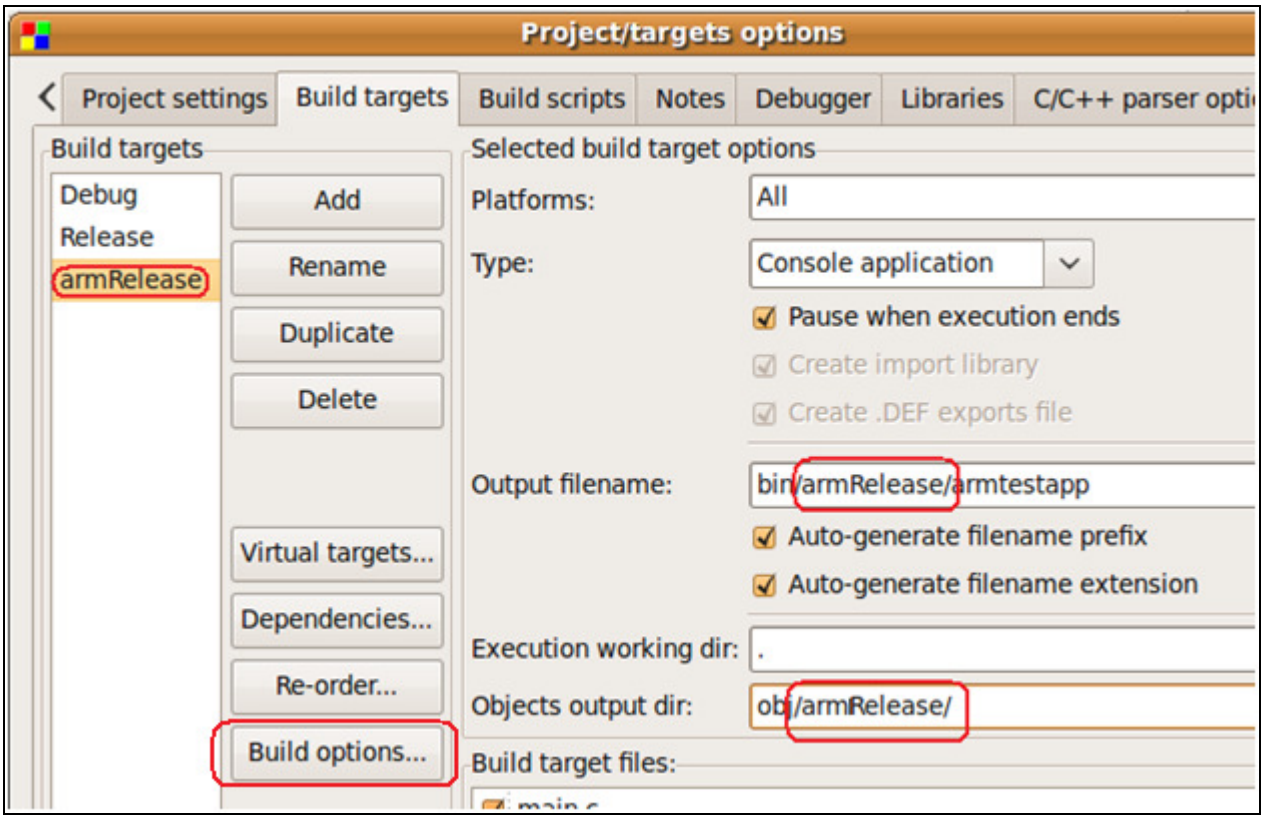

Click Project -> Properties...




Click the Build targets tab. Select the Release target and click Duplicate.

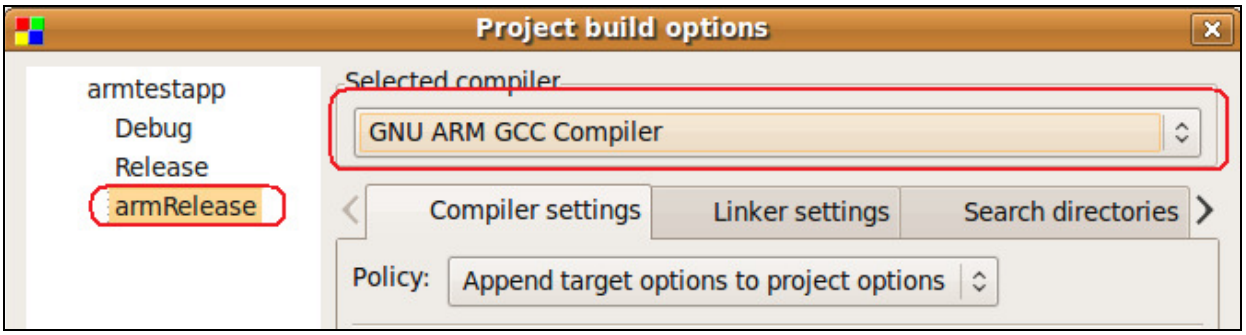



Give it a name. I chose armRelease.




Make sure you have the armRelease target specified from the Build targets. Now add this name to the "Output filename" and "Objects output dir" or you will overwrite your regular release objects.

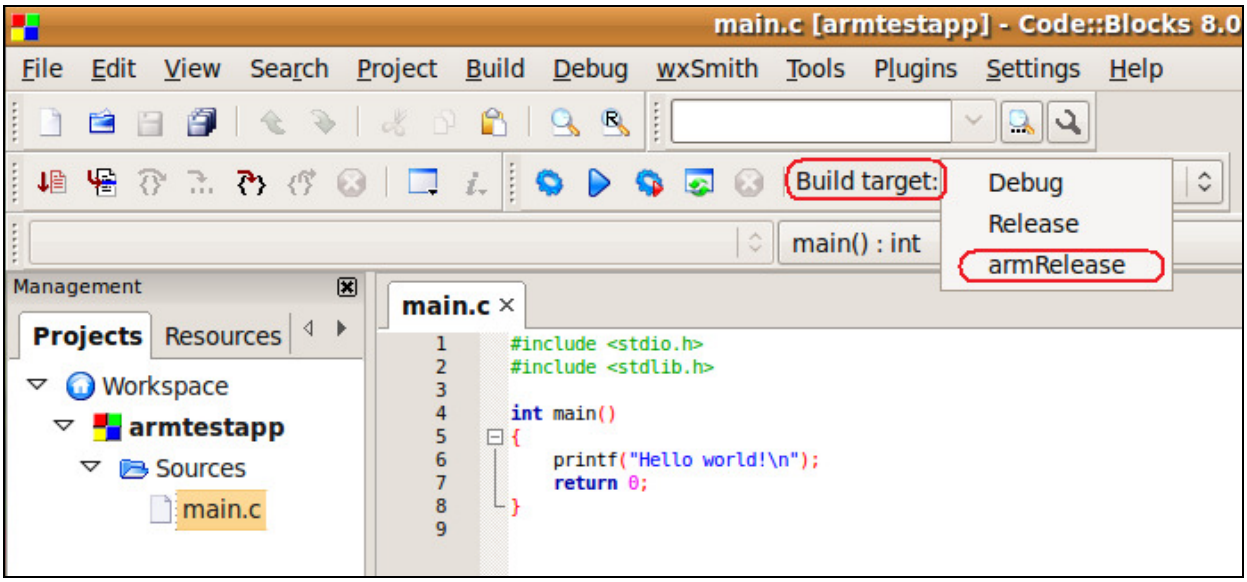
Now click the Build options... button.



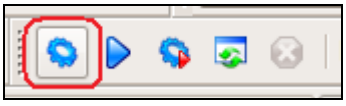

Select the armRelease target and then change the compiler to the ARM GCC Compiler.
Click **OK** to save the settings and the **OK** again.

## Testing the Compiler

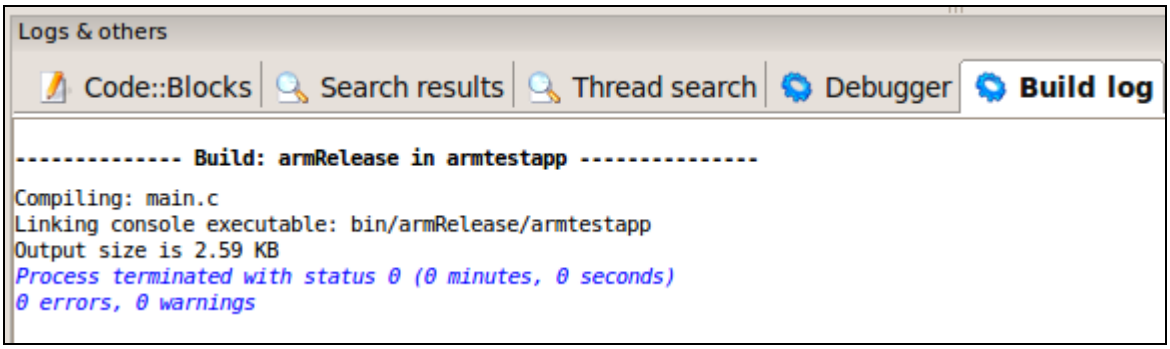
Now let's test our new compiler out.




Change the Built target type to our armRelease target we just set up.



Click the Build button (or the Rebuild all button).




Check for success down in the Build Log.

Congratulations! You successfully configured the CodeBlocks editor to compile for the OmniFlash ARM processor.

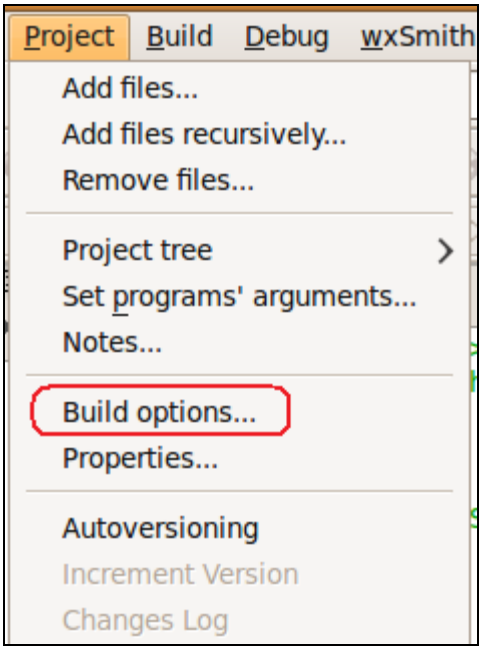
## Adding #defines to a project

Sometimes you need to add conditional #defines to your project. For example, some code statements may work on one platform, but not on others.

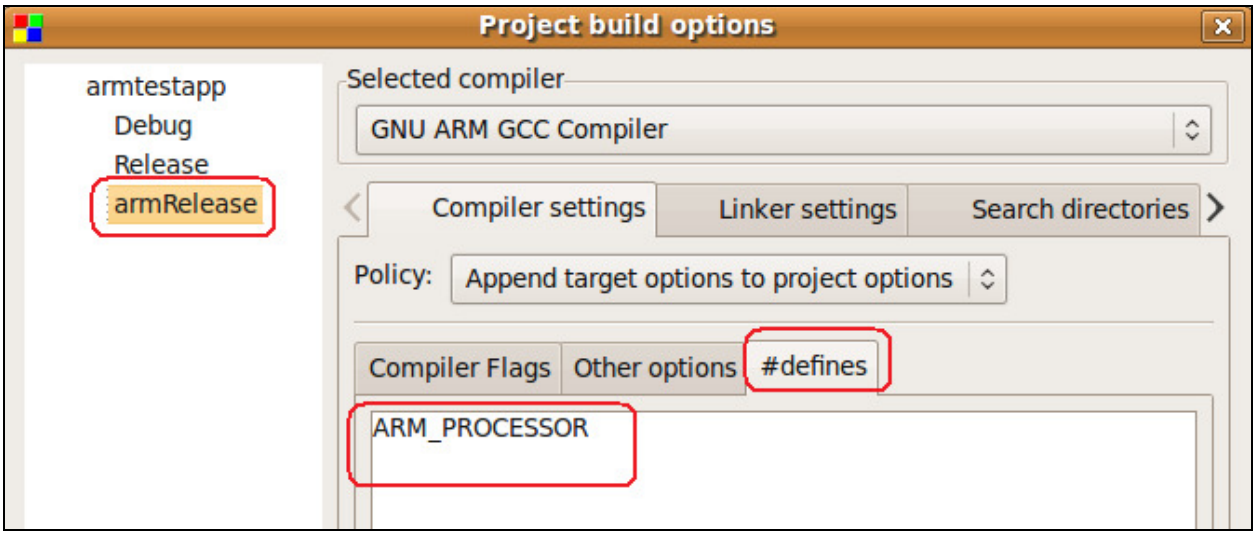
main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int val = 0;
7
8  #ifdef ARM_PROCESSOR
9      printf("Hello ARM World!\n");
10     val = 1;
11 #else
12     printf("Hello PC World!\n");
13     val = 2;
14 #endif
15     printf("The value of val is %d\n", val);
16     return 0;
17 }
```

Take for example the following code snippet. We need to define **ARM_PROCESSOR** when we build for the arm.




Click on Project -> Build options...



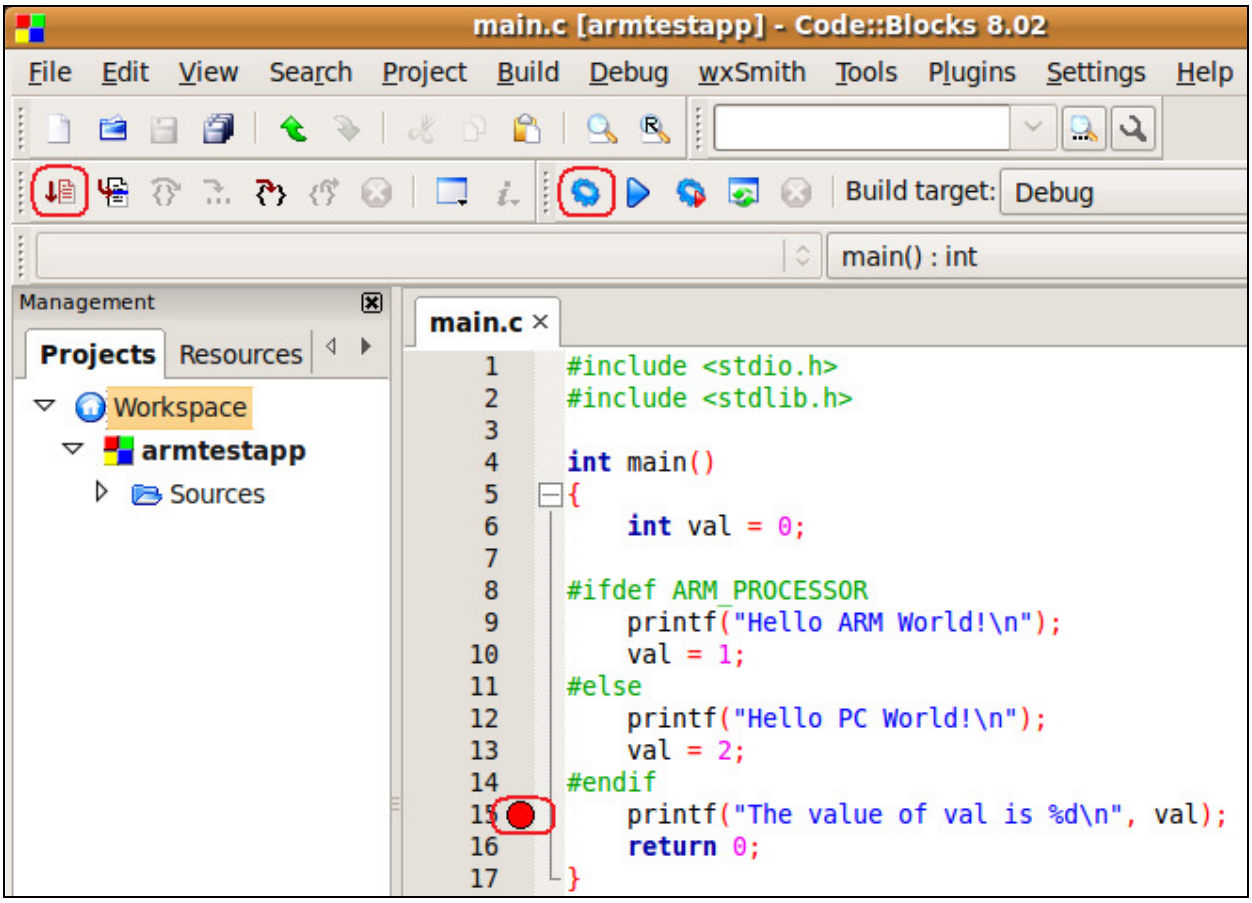

From the left, click on the armRelease target. Then click the #defines tab and add your defines. If you have more than one define to add, add them each on separate lines.

## Testing our application locally

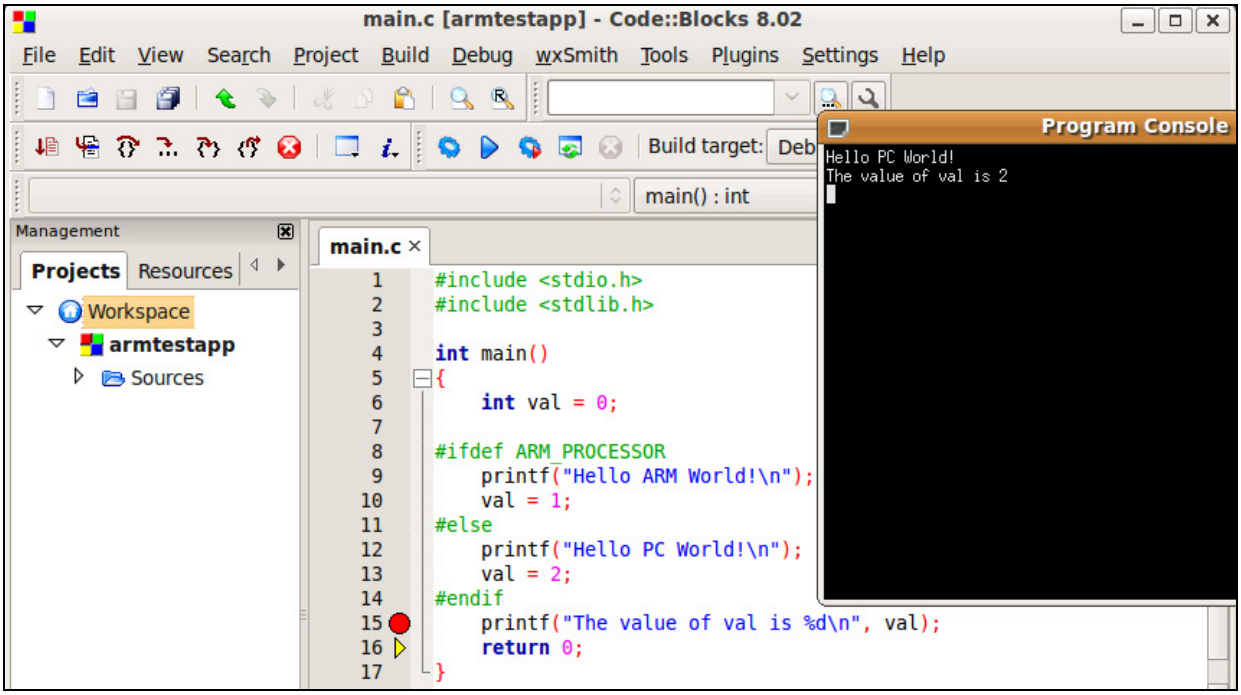
Before we run the application on the ARM processor, we can test and debug our application on the Linux system and work out the kinks.


Change the build target to Debug.
Compile the program by either clicking the Build icon or the Rebuild All icon.

To set a breakpoint, click just to the right of the line number.

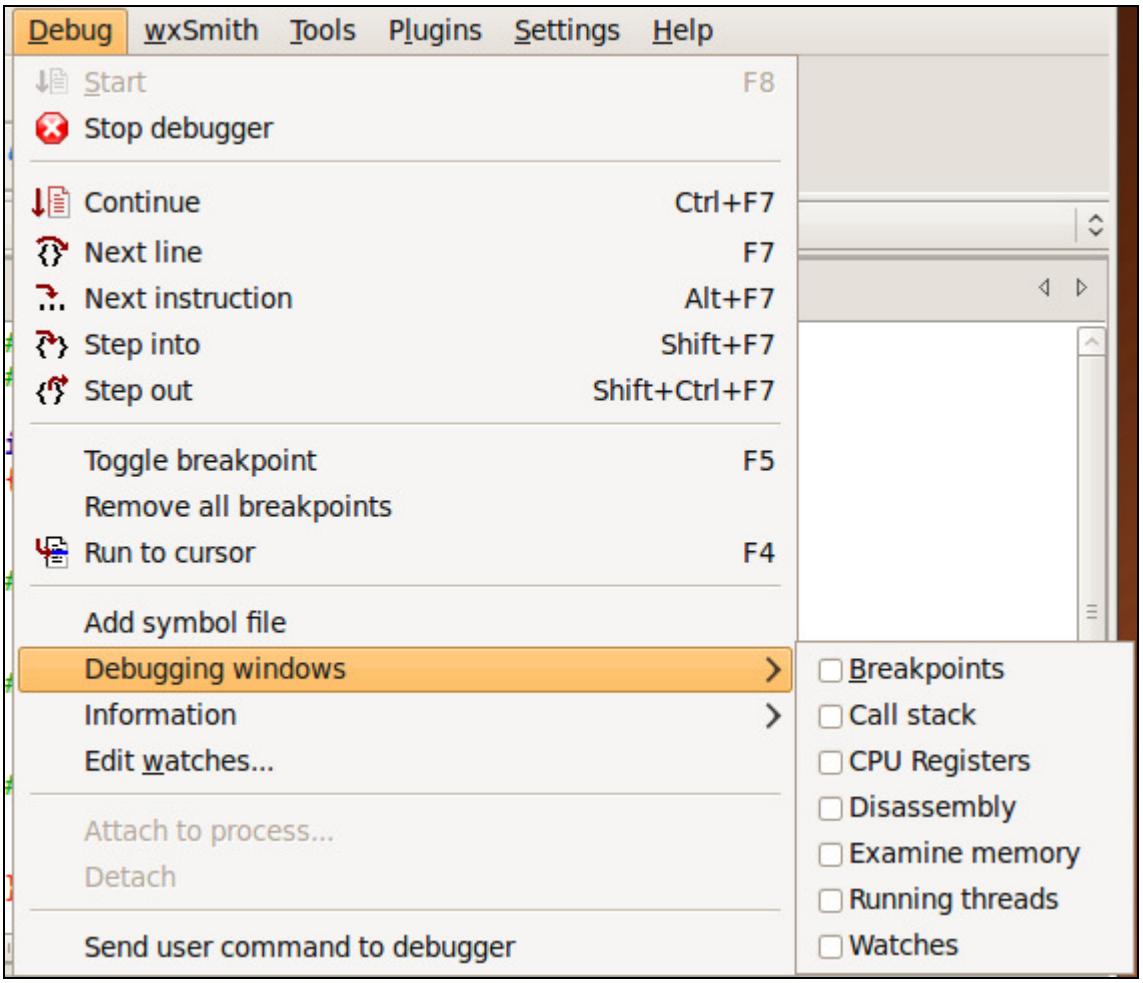



This picture shows setting a breakpoint, building the application, and then running it in debug.

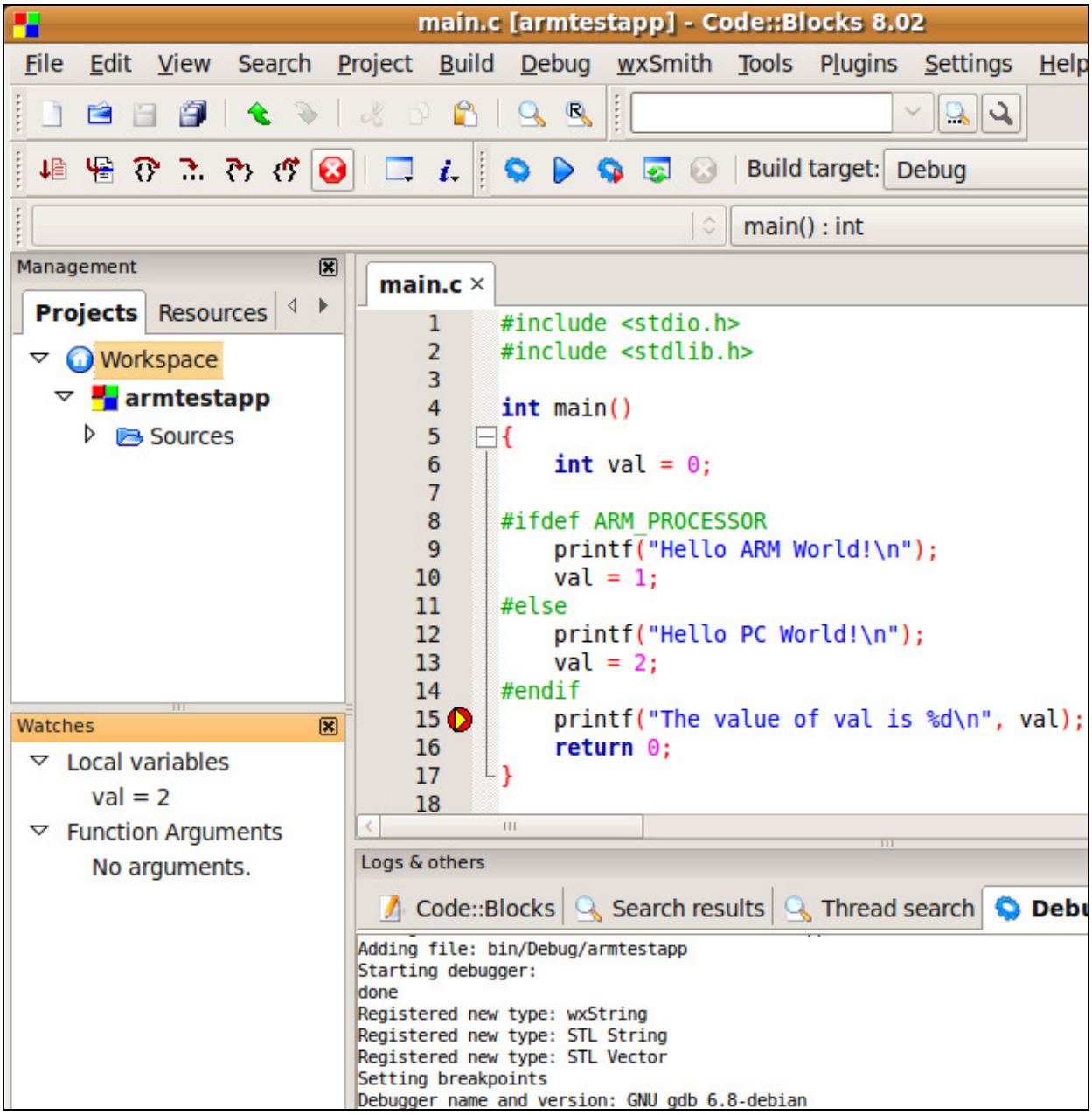


Here is our debugger output.

To enable the watch window and any other debugging information,




Click Debug -> Debugging windows -> Watches (or any other window). Then you can drag the window(s) and dock them into your main window somewhere.

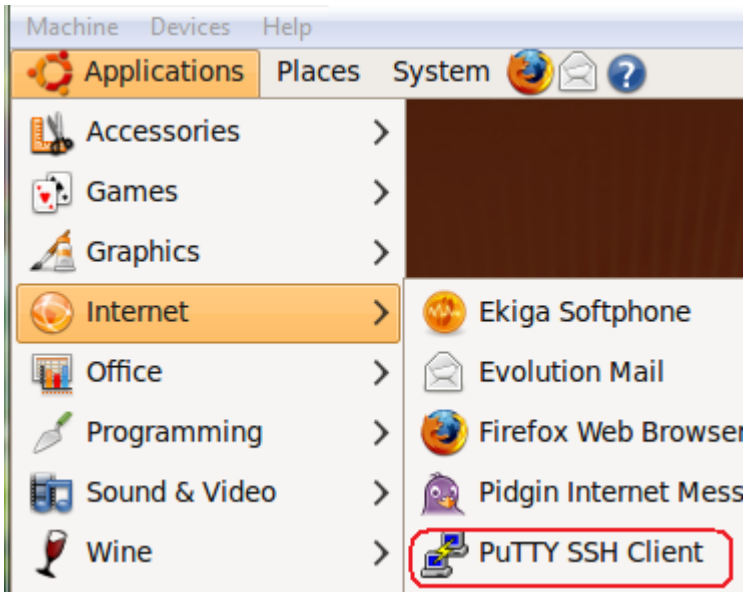


Example workspace with Watches turned on.

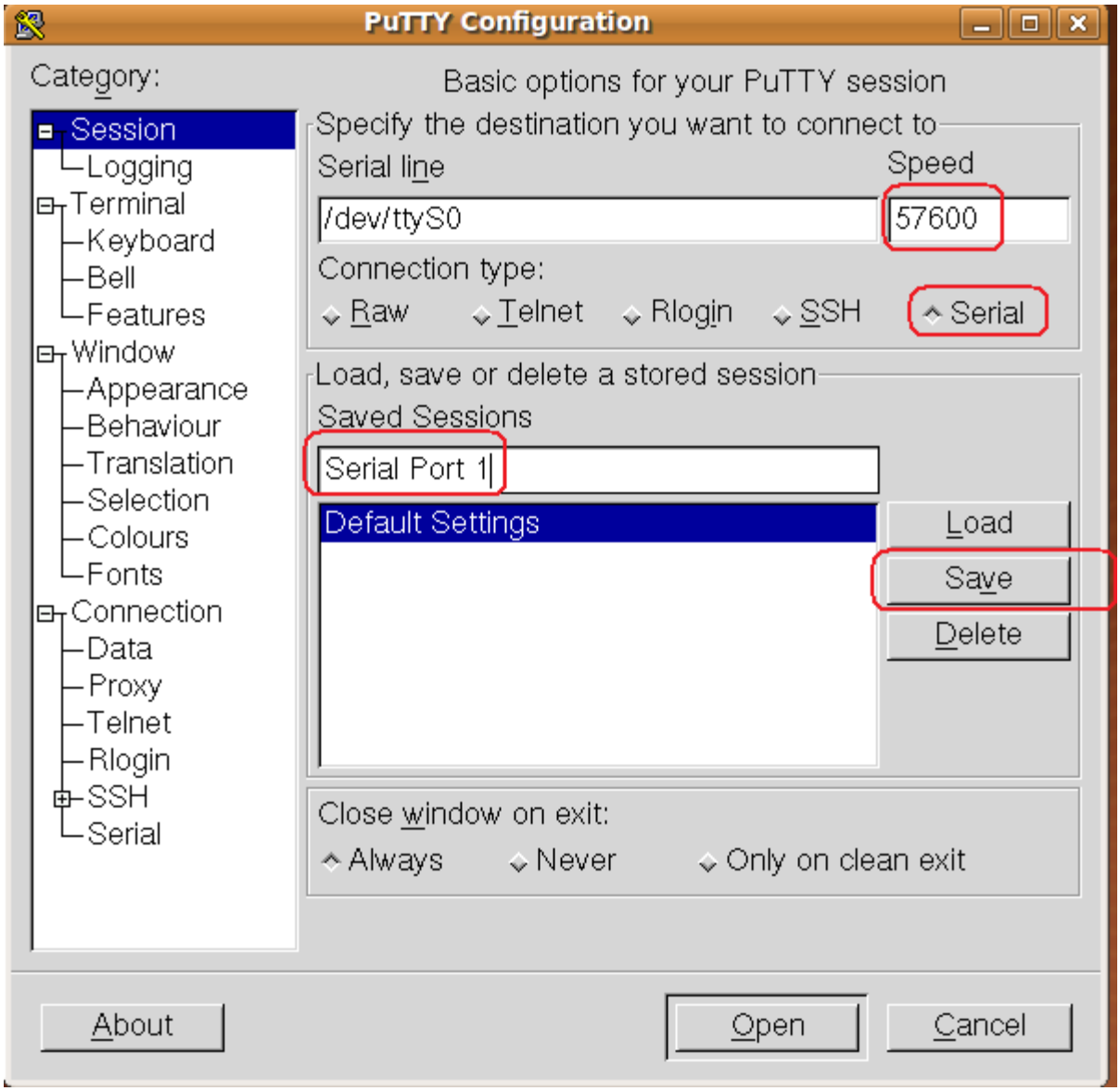
## Running our new program on the ARM Processor

The next thing we need to do is configure a couple tools so we can communicate and program the OmniFlash ARM processor.

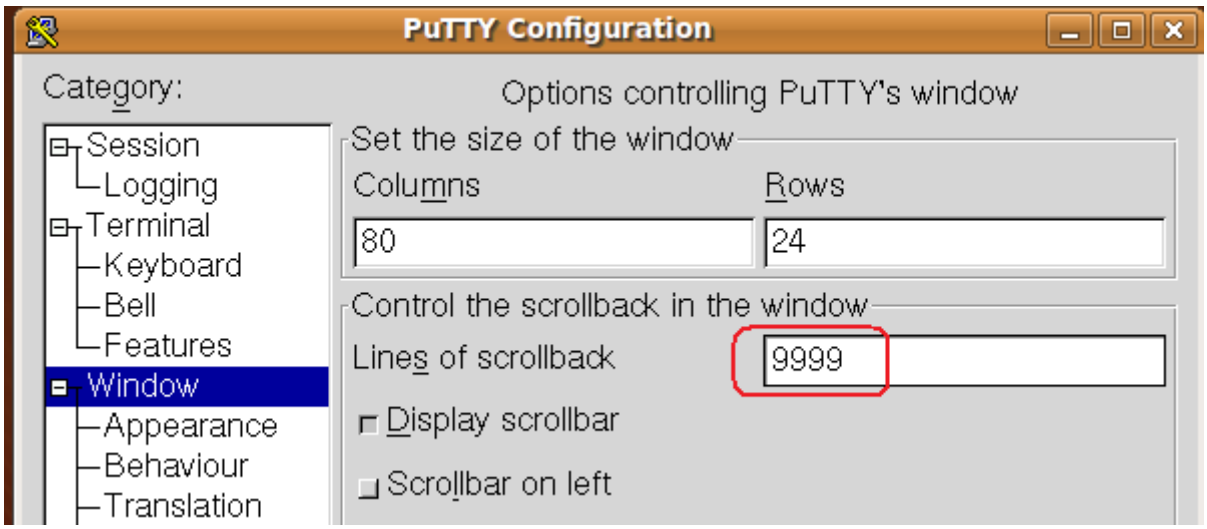
## Serial Port Configuration - PuTTY



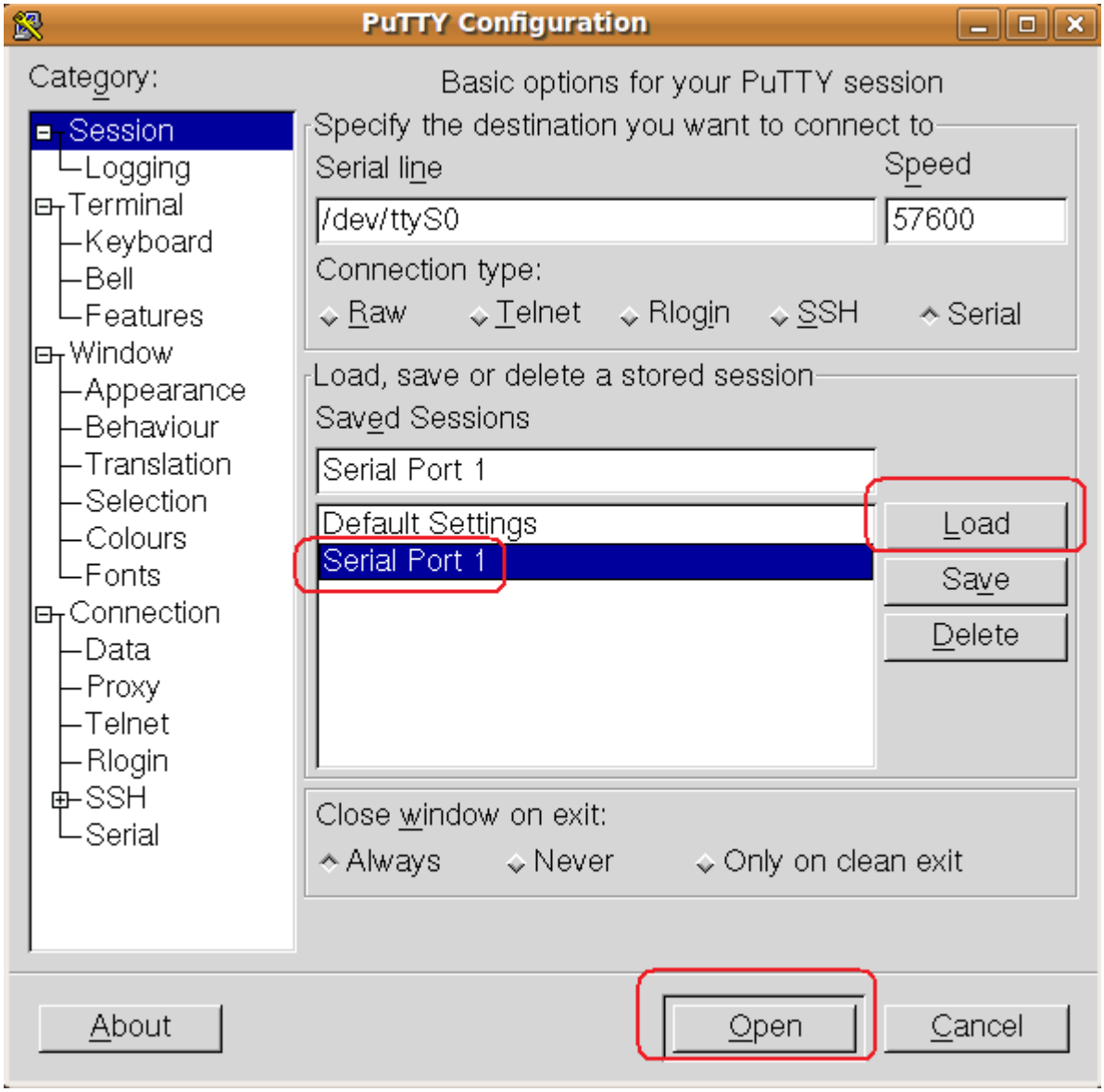

Start up PuTTY. (You can use gtkterm also. For the sake of this document, I will only cover PuTTY).




Click the option in the upper right-hand section of the screen. Change the speed to 57600. Give the session a name and then click Save.

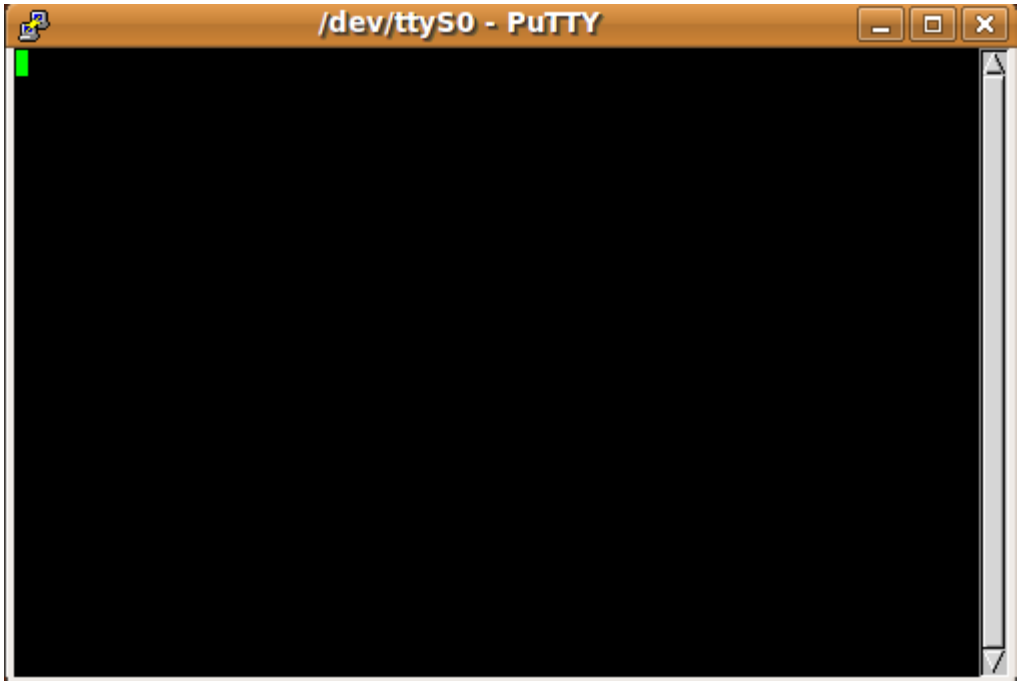



If you want more scroll back lines, load the profile for Serial Port 1 and then change the Window settings for scroll back lines. Then go back to the Session option and save it again.

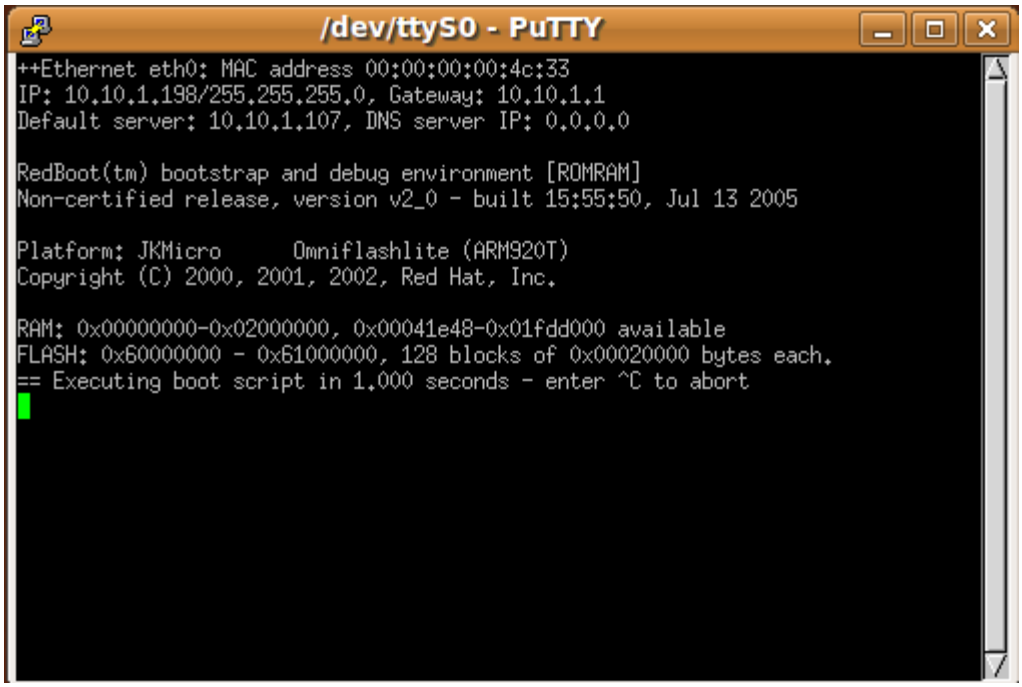



To open the connection, click the one on the list you want to open, click the Load button, then click the Open button.

## Connection Verification - PuTTy



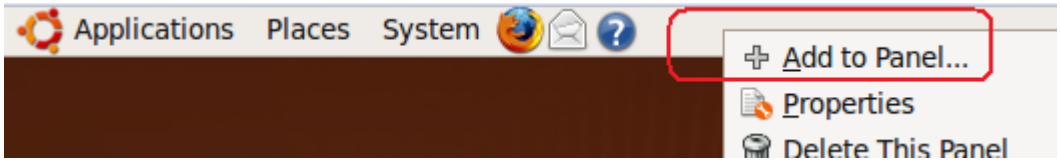

Once we click Open we get the following window. We are ready to turn plug in the OmniFlash and power it up. Connect the serial cable to the OmniFlash and turn on power.




You should get output from the OmniFlash. If you don't get any output, verify you have the OmniFlash plugged in correctly. You may want to try a Windows COM port program like Hyperterminal or Tera Term (http://www.ayera.com/teraterm/) and verify you have everything connected and working. Note, when the Virtual Machine is running, you will NOT be able to use the serial port from Windows.

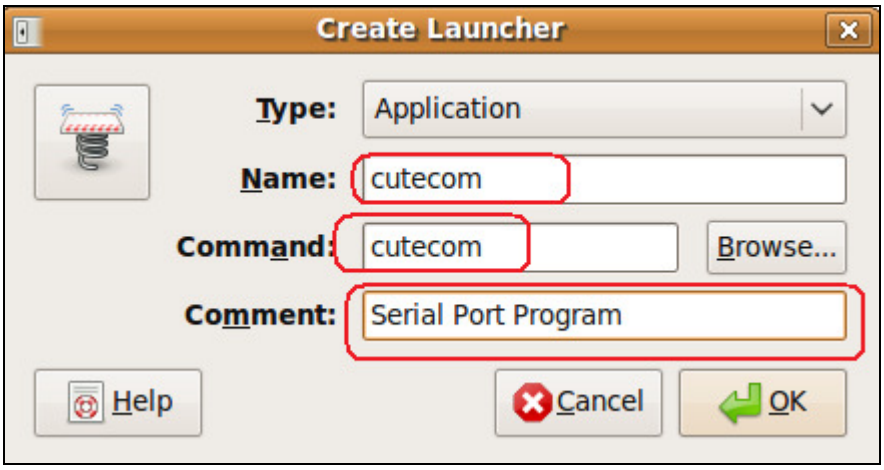
## Serial Port Configuration - CuteCom

Next we need to configure the program used to send software via X-modem protocol to the OmniFlash. This program is called cutecom. Since a quick launch icon was not created for cutecom, we need to create one.



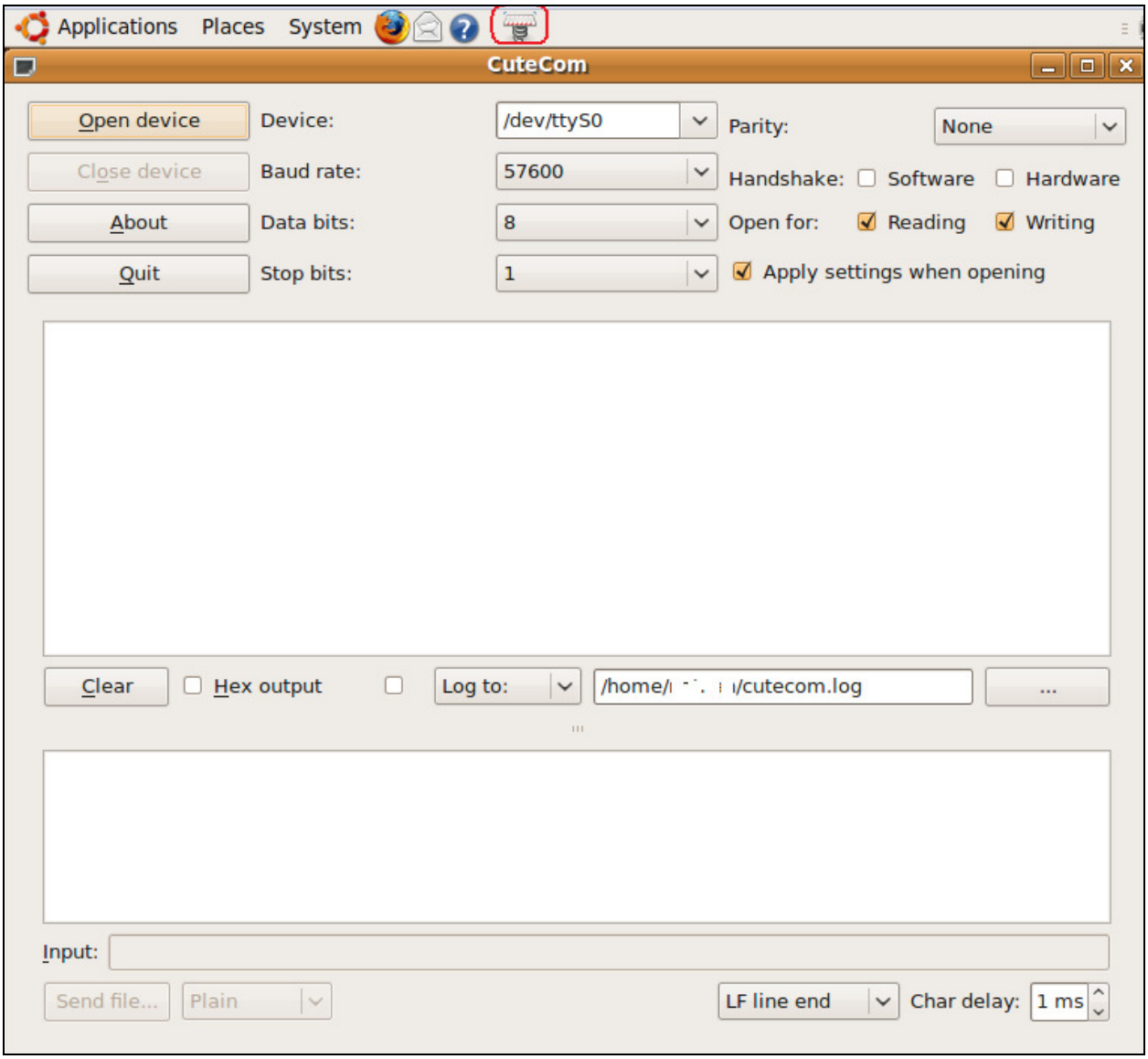

Right-click on the toolbar past the question mark and select Add to Panel...


Choose Custom Application Launcher.




The Command is the most important here. Type cutecom. Give it a name and a comment and press OK. Then close the Add to Panel window.

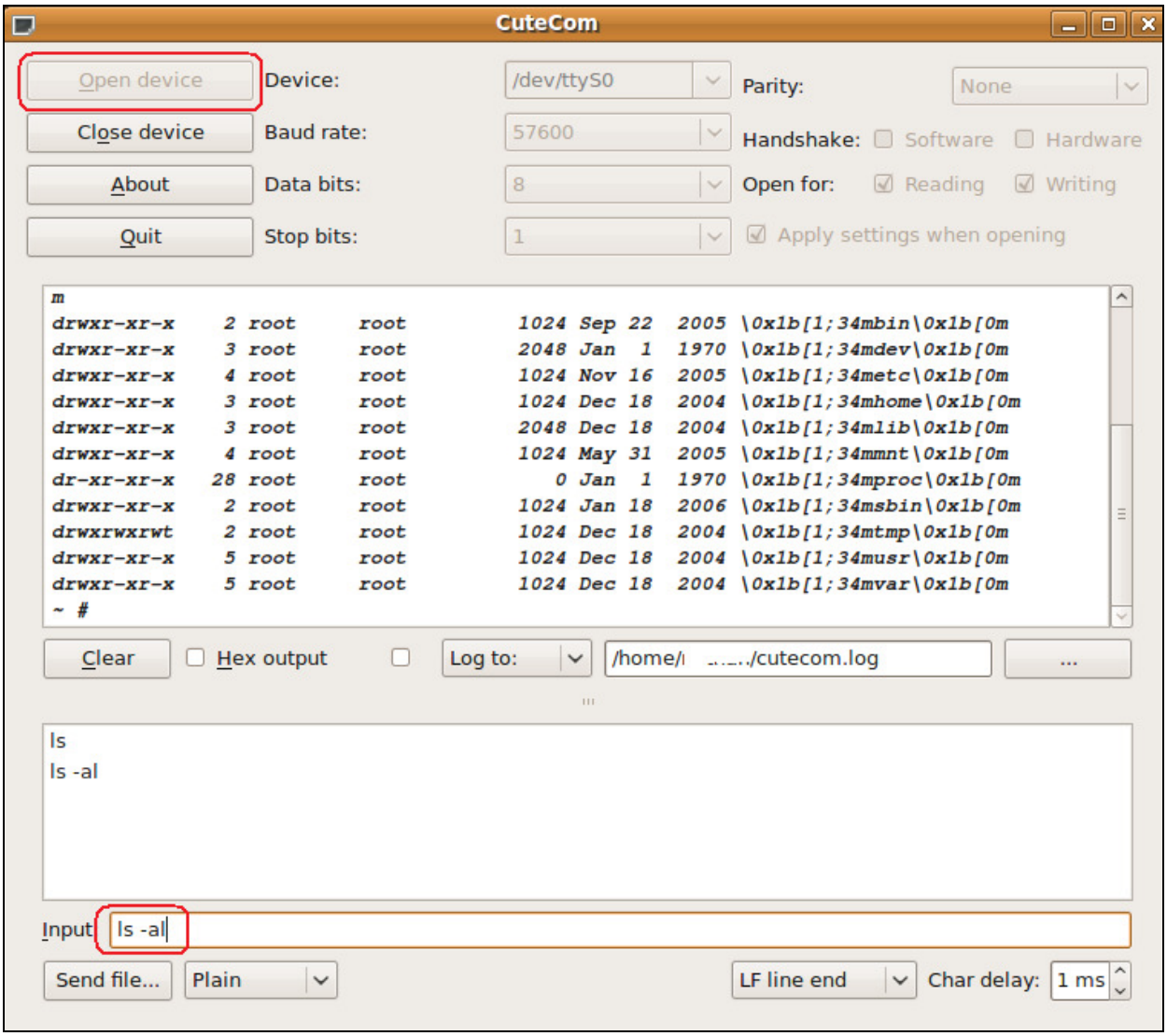
## Testing connection with CuteCom




**Note: Make sure you close PuTTY before you do this as only one program can be connected to the serial port at a time.**

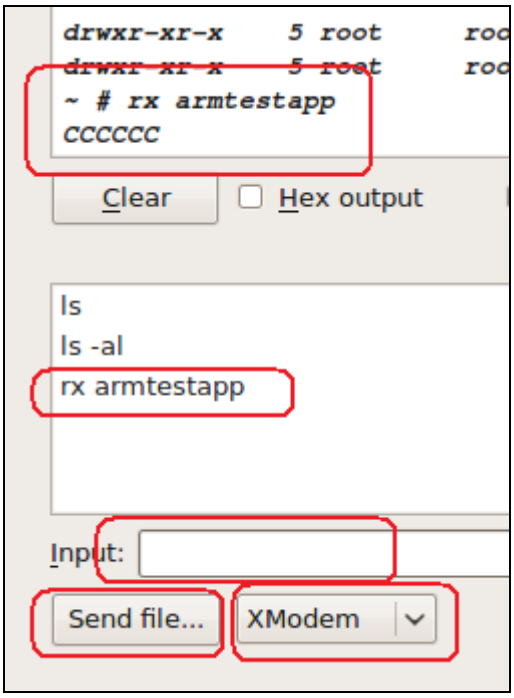
We now have a new icon at the top of our screen. Click it and test it to see that it works.
Verify that the Device selected in ttyS0 and the baud rate is 57600.

The way this program works is we Open the device (for reading and writing) when we need to communicate with the OmniFlash. We can only send one command at a time on the Input line. Output is shown in the big white square.



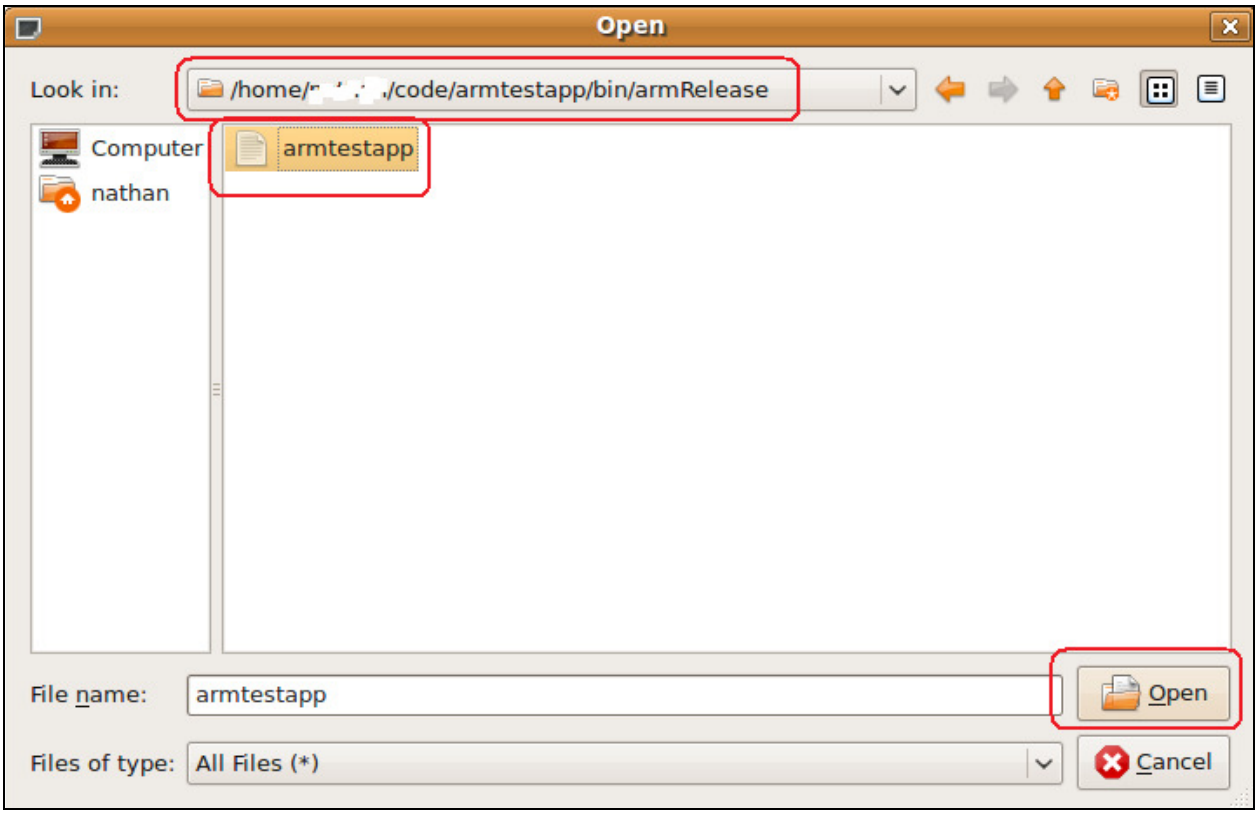

On the input line, we can type commands. Pressing Enter will send them. Once we send a command, we can double-click it from the lower window list.

## Sending a program via CuteCom



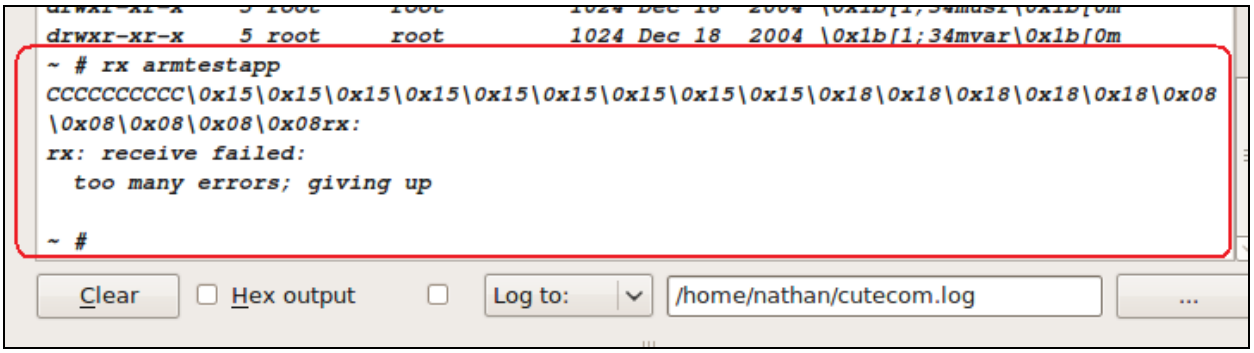

The first thing we do is type "rx armtesetapp" or whatever the name of the program we want to receive on the ARM processor. We then press ENTER. This sends the receive X-Modem command to the ARM processor. The ARM processor starts querying for the file to be received. You will see at the top of the screen shot that the ARM received our command and the CCCC lets us know it is retrying to receive the file. Next we change the file type to XModem and click the Send file... button.

Note: We can only save files to **/mnt/FlashMemory** if we want them to persist when we reboot. For this test, we won't be writing to this location. We will send it to the root file system which will be erased when we reboot the OmniFlash.

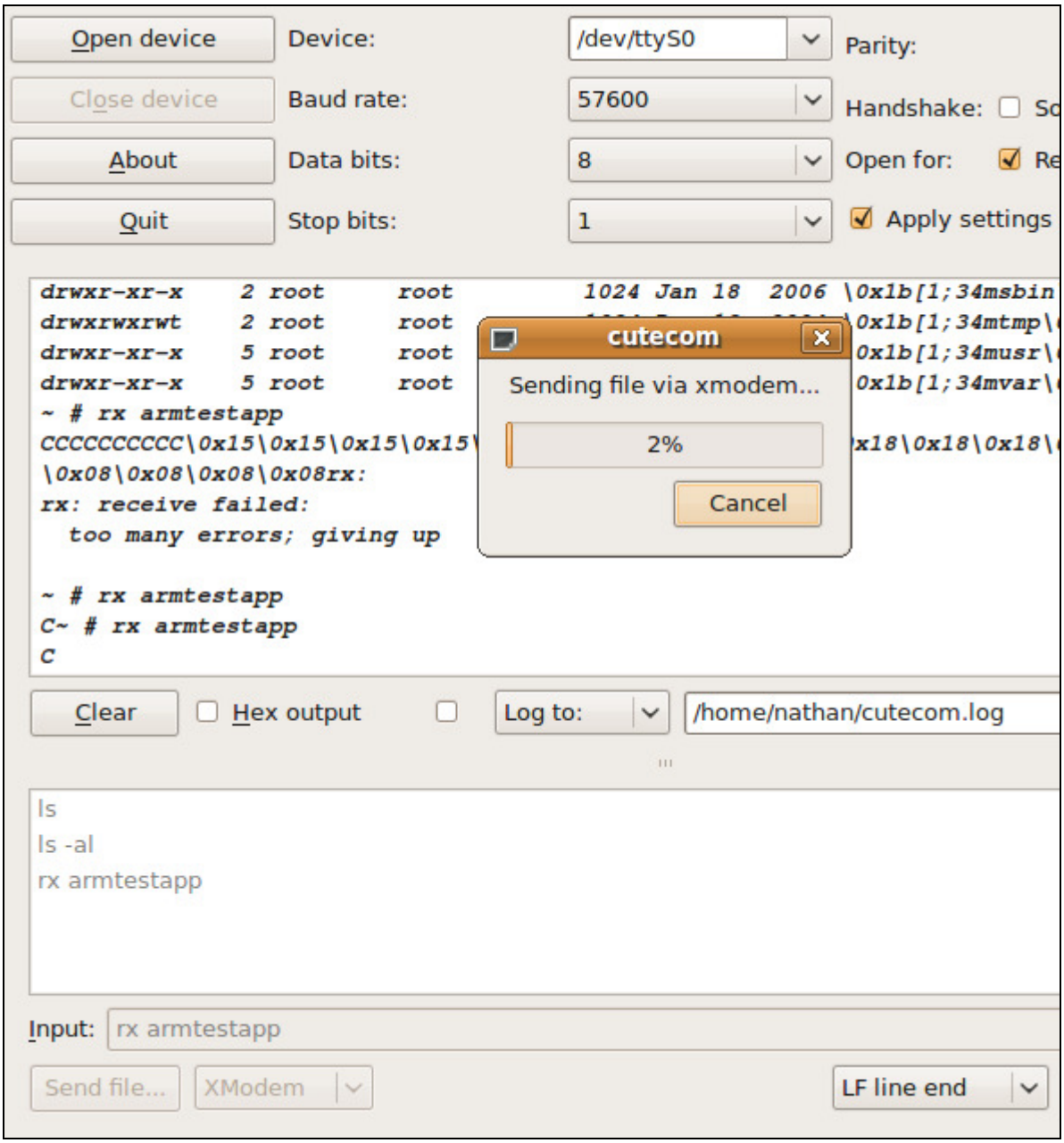



Navigate to where the program we built earlier resides. Look in the armRelease folder for the application.

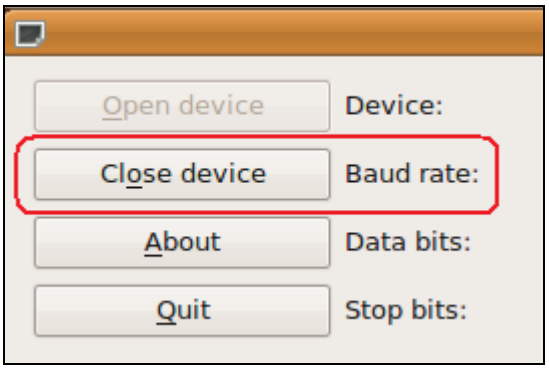
Click Open to send the file.




If the ARM processor times out, you will have to send it again.

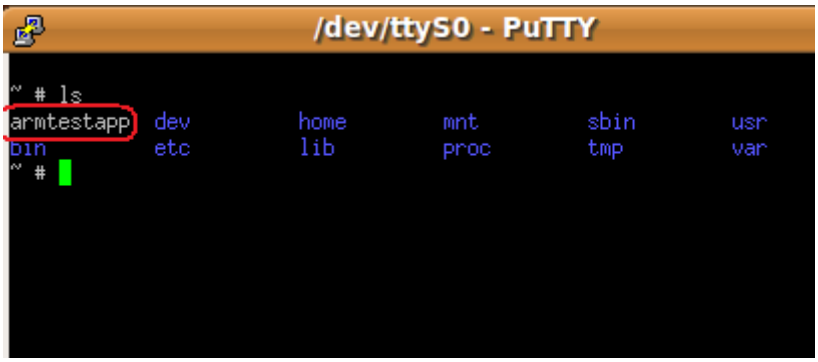



Here, I double-clicked on the "rx armtestapp" text which sent it to the ARM processor. Then I clicked the "Send file..." button and selected my test application. You can see that the file is being transferred to the ARM processor.




Click the "Close device button after you have sent the file.

## Verifying the program was received by the OmniFlash




Now launch PuTTY to connect to the OmniFlash again. Press enter to get a prompt back. Type "ls" and you should see the file we just transmitted.

## Launching the program


Before we can run the application, we need to set "execute" permissions on the file.

Type "**chmod +x armtestapp**" to make it an executable

Now try running it and see what happens.


Type "**./armtestapp**" to run it. The "./" tells the operating system to look in the current directory (since it isn't in the path).

Notice we ran the ARM specific code and not the PC specific version.

There you have it. A fully graphic environment to code and test your application for your OmniFlash. And we didn't have to write a single makefile to compile for the ARM processor.

This is a bit of a juggling act in that you have to switch back and forth between two applications remembering to close PuTTY when you use CuteCom and remembering to close the connection on CuteCom before you can use PuTTY. But at least you don't have to type all of this from the command line.

## Final Notes

### Note about writing to /mnt/FlashMemory.

If you write your programs to /mnt/FlashMemory, be sure to type "sync" to write it to flash memory after you chmod the program or anytime you write to the Flash memory system. If you don't and you lose power or reboot your device, you may corrupt the Flash Memory and you will have to repair it.