REFERENCE: http://www.michaelhleonard.com/cross-compile-for-beaglebone-black/

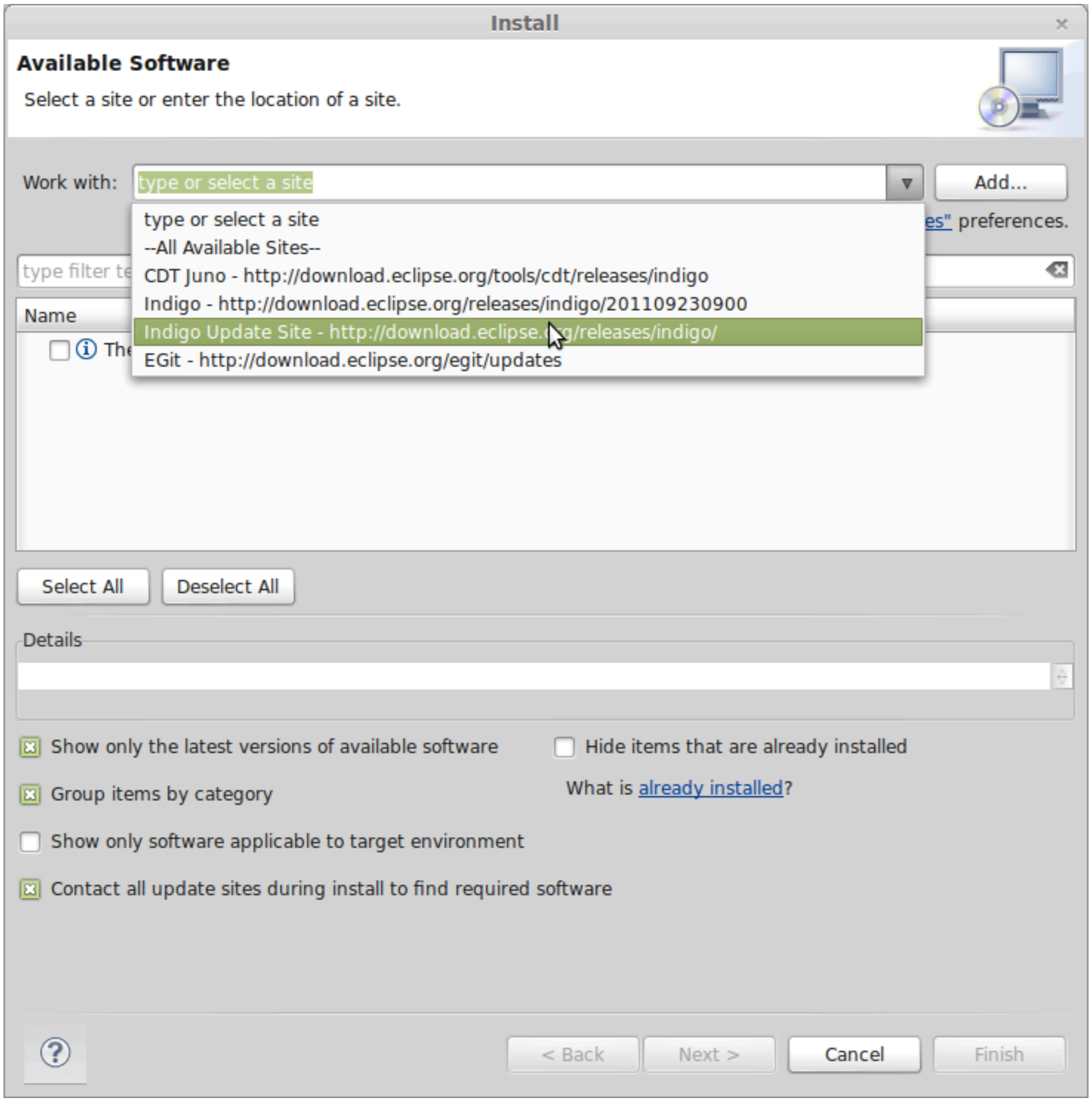
Install Eclipse, C/C++ Tools, and Remote Tools (Debian) - Lab 5

Open up the Terminal and enter:

```
sudo apt-get install eclipse eclipse-cdt g++ gcc
```

This command installs the main Eclipse package, the Eclipse C/C++ development tools (CDT), the GNU C++ Compiler/Linker (G++), and the GNU Compiler Collection (GCC). You most likely already have G++ and GCC installed, and they aren't even really necessary for this example, but it never hurts to verify.

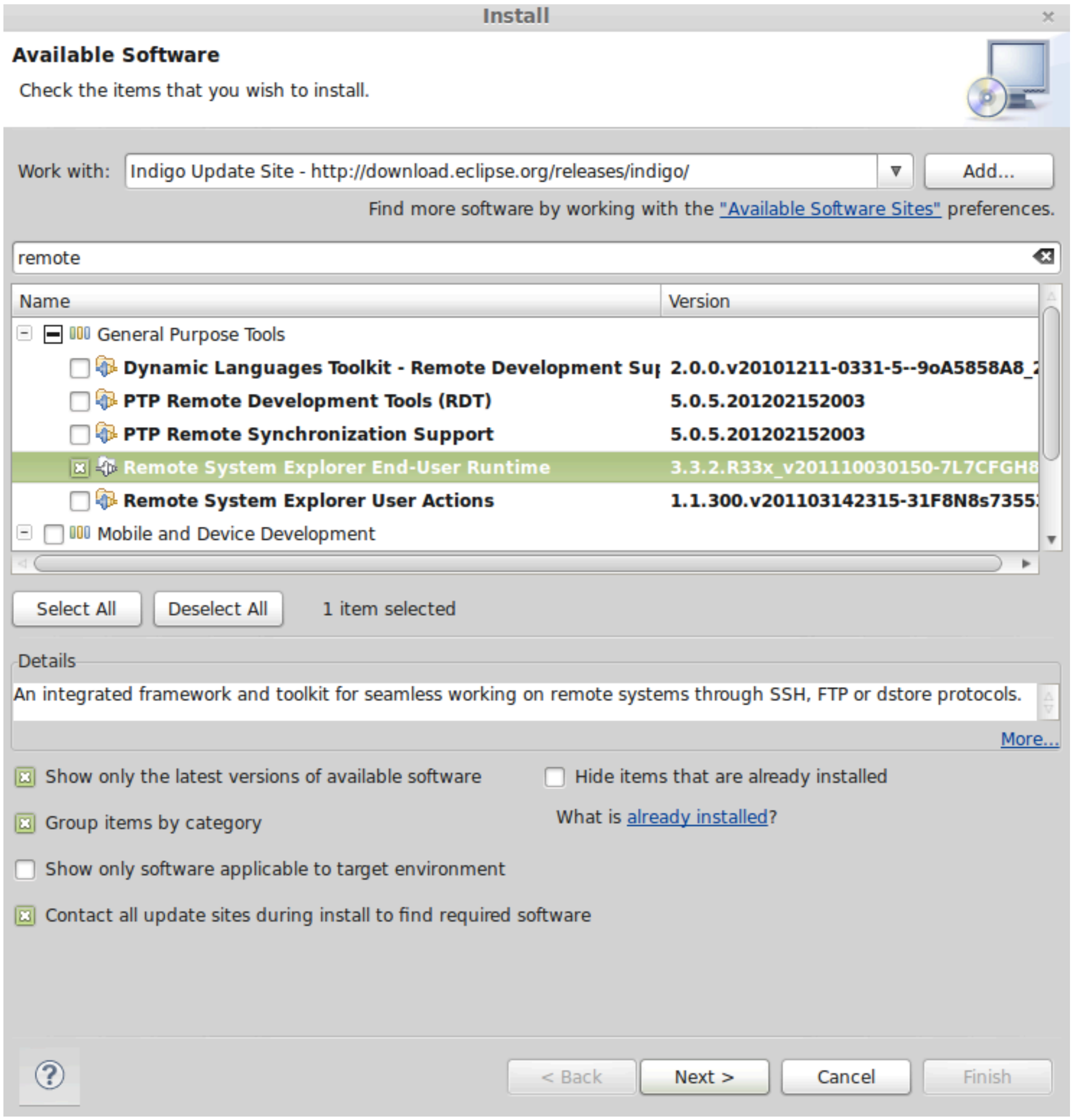
After this installation finishes you need to open up Eclipse and install the "Remote System Explorer" (RSE) plugin. It is possible that RSE is included in the default Eclipse installation. To check this, go to Window -> Open Perspective -> Other… and choose "Remote System Explorer" from the Open Perspective dialog to open the RSE perspective. If it is not installed, navigate to Help -> Install New Software… Where the label says "Work with:" click on the down arrow to show a dropdown box. Select the update site for your version of Eclipse. For example my version of Eclipse is Indigo, so I choose the Indigo Update Site.


Install
Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
type or select a site
--All Available Sites--
CDT Juno - http://download.eclipse.org/tools/cdt/releases/indigo
Indigo - http://download.eclipse.org/releases/indigo/201109230900
Indigo Update Site - http://download.eclipse.org/releases/indigo/
EGit - http://download.eclipse.org/egit/updates
es" preferences.
type filter te
Name
Select All
Deselect All
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Cancel
Finish

Selecting the Indigo Update Site

Then move your cursor to the search box below and enter “remote”. After a short wait you will be presented with a list of packages, one of which will be the “Remote System Explorer End-User Runtime” select this box. You will also want to scroll a bit further down under the “Mobile and Device Development” section and select the “C/C++ Remote Launch” plugin. After you have selected these packages, proceed with the wizard to finish the install. So again, be sure to install:

- Remote System Explorer End-User Runtime
- C/C++ Remote Launch


Install

Available Software

Check the items that you wish to install.

Work with: Indigo Update Site - http://download.eclipse.org/releases/indigo/

Find more software by working with the "Available Software Sites" preferences.

remote

Name	Version
[–] General Purpose Tools	
[ ] **Dynamic Languages Toolkit - Remote Development Su**	**2.0.0.v20101211-0331-5--9oA5858A8_**
[ ] **PTP Remote Development Tools (RDT)**	**5.0.5.201202152003**
[ ] **PTP Remote Synchronization Support**	**5.0.5.201202152003**
[x] **Remote System Explorer End-User Runtime**	**3.3.2.R33x_v201110030150-7L7CFGH**
[ ] **Remote System Explorer User Actions**	**1.1.300.v201103142315-31F8N8s7355**
[ ] Mobile and Device Development	

1 item selected

Details

An integrated framework and toolkit for seamless working on remote systems through SSH, FTP or dstore protocols.

[x] Show only the latest versions of available software

[ ] Hide items that are already installed

[x] Group items by category

What is already installed?

[ ] Show only software applicable to target environment

[x] Contact all update sites during install to find required software

Installing Remote System Explorer

Now that your basic Eclipse environment is set-up, we can move on to installing the BeagleBone Specific functionality.

Install Toolchain for BeagleBone Black

Plug your BeagleBone Black into your computer using the provided Mini-USB cable. If you are developing from a virtual machine make sure that the BeagleBone Black is mounted on your virtual machine and not on your host machine.

After the BeagleBone Black is mounted it should appear on your desktop as a removable drive. Navigate to this drive and double-click on START.htm. This will run the install scripts and verify that the BeagleBone Black is operating as expected. After a few seconds you should see the boxes for Step 1 and Step 2 turn green and earn a check-mark. This means that you are ready to move on to installation.

To install the proper toolchain for the BeagleBone Black we will pull the gcc-arm-linux-gnueabi package from the apt repository. This is included in the default Debian repositories so you can simply type…

```
sudo apt-get install gcc-arm-linux-gnueabi
```

After a short wait the ARM toolchain should install successfully and you are well on your way to being able to cross-compile for BeagleBone Black. The next step is to set up your Eclipse environment to make this process as automated as possible.

Note (5/11/2015): It looks like some newer versions of Linux have separated the GCC and G++ toolchains and you will likely need both. Install G++ with:

```
sudo apt-get install g++-arm-linux-gnueabi
```

Create and Configure a Project to Build Using this Toolchain

This is where the magic happens, once we complete this section you should be able to create your own project that will compile on your local computer and then run on your BeagleBone Black with the push of a single button. Let's begin with something simple, a classic "Hello, World" application.

Creating a C++ Project

Navigate your cursor to the "Project Explorer" view. This is usually the left side-panel but if it isn't there then you can go to Window -> Show View -> Project Explorer. With your cursor positioned inside this view, right click and select New -> C++ Project.

A window will appear asking you to name your project and choose some project settings. You can name your project whatever you like, I will be naming mine "Test".

Then select Executable -> Hello World C++ Project. If you have multiple Toolchain options just choose one, it doesn't matter since we will be changing this later.

At the next screen you will be asked to set up some of the template options, you can leave them as-is or change them to your liking. I will be changing the Hello World message to Hello BeagleBone. After you are done with this screen you can go to the next and skip that one, then select the “Finish” button.

After a moment Eclipse will finish preparing the project and you will be presented with the familiar “Hello World” program. Just to make sure there are no weird errors, go ahead and click the build button in the toolbar (it looks like a hammer). In your console at the bottom you should see a few compiler commands and it should end with a build summary. If you get errors at this point then there is most likely something wrong with your Eclipse installation, I recommend doing a few Google searches for help, then re-installing Eclipse if you can’t find a solution. If all went well then you can move on to the next step.

Configure the ARM Toolchain

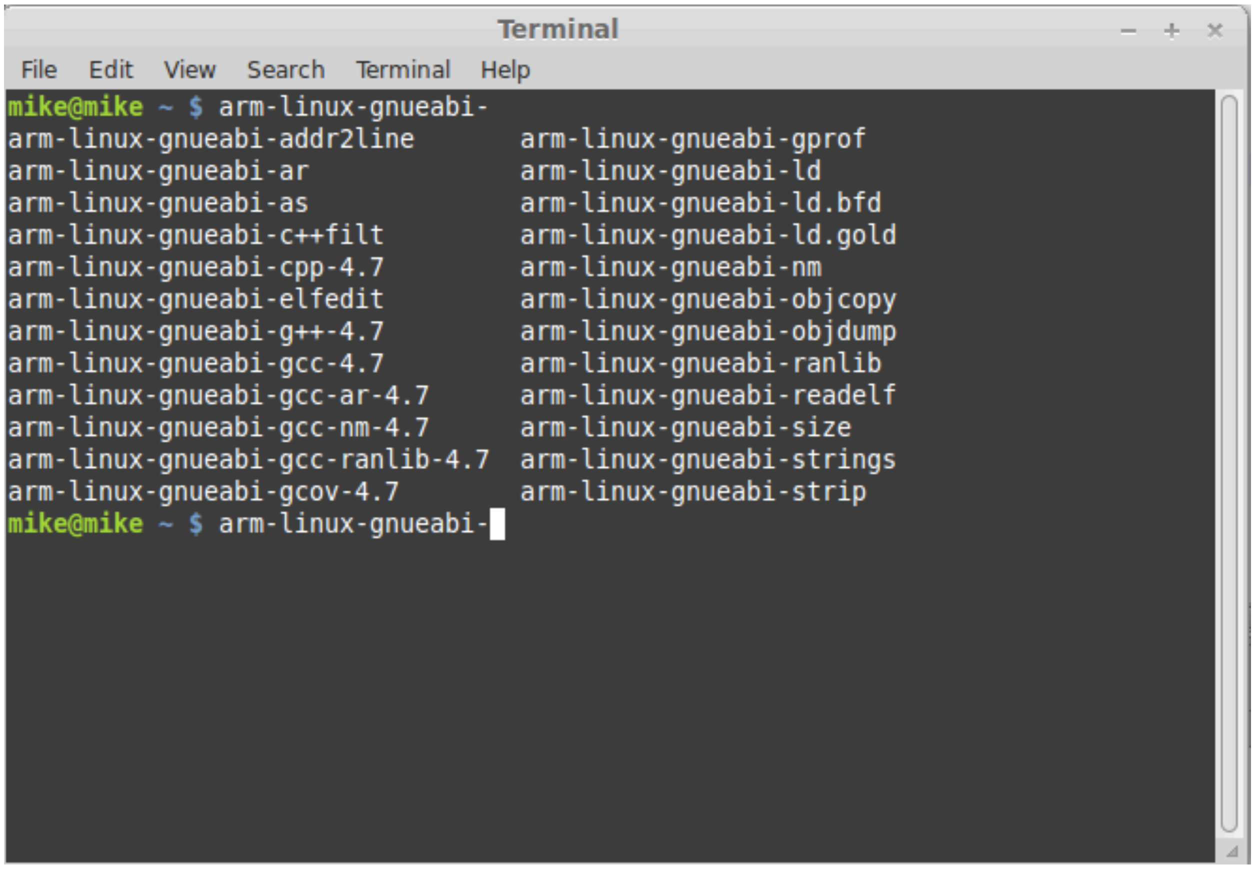
If you took a good look through your last build log you will notice that Eclipse used GCC and G++ to compile your program. This is normally fine, but in our case we want to deploy to an ARM architecture so we need to reconfigure our project to build using the toolchain we downloaded earlier.

To do this you can right click on your project name in the “Project Explorer” and select “Properties” from the menu. This will bring up a new window where you can modify the build settings of your project.

On the left sidebar of the properties window expand the “C/C++ Build” category and select “Settings”. This is where you can change your compilers, linker, and assembler, otherwise known as your toolchain. Make sure your active tab is “Tool Settings”, we will begin by changing your GCC Compiler. For my current set-up I change my compiler to:

```
arm-linux-gnueabi-gcc-4.7
```

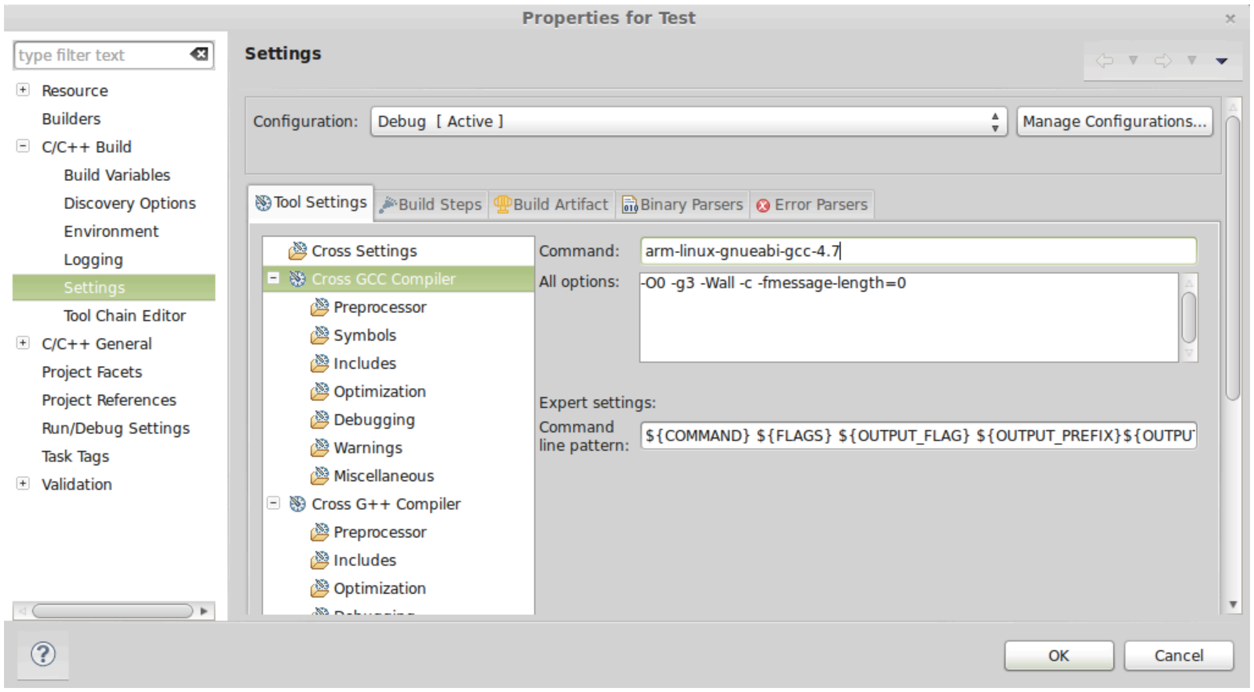
But this could be different for you, I recommend opening up a terminal window and typing arm-linux then hitting tab a few times to get a list of suggestions. This will show you what options you have available. To avoid messing up, I would just copy and paste from this list. This is what it looks like on my system.



```
mike@mike ~ $ arm-linux-gnueabi-
arm-linux-gnueabi-addr2line       arm-linux-gnueabi-gprof
arm-linux-gnueabi-ar              arm-linux-gnueabi-ld
arm-linux-gnueabi-as              arm-linux-gnueabi-ld.bfd
arm-linux-gnueabi-c++filt         arm-linux-gnueabi-ld.gold
arm-linux-gnueabi-cpp-4.7         arm-linux-gnueabi-nm
arm-linux-gnueabi-elfedit         arm-linux-gnueabi-objcopy
arm-linux-gnueabi-g++-4.7         arm-linux-gnueabi-objdump
arm-linux-gnueabi-gcc-4.7         arm-linux-gnueabi-ranlib
arm-linux-gnueabi-gcc-ar-4.7      arm-linux-gnueabi-readelf
arm-linux-gnueabi-gcc-nm-4.7      arm-linux-gnueabi-size
arm-linux-gnueabi-gcc-ranlib-4.7  arm-linux-gnueabi-strings
arm-linux-gnueabi-gcov-4.7        arm-linux-gnueabi-strip
mike@mike ~ $ arm-linux-gnueabi-
```

Showing available ARM compilers

You will now go through all four options and change them to their appropriate ARM equivalents. My GCC compiler command looks like this after changing it:



Configuration: Debug [Active]

Command: arm-linux-gnueabi-gcc-4.7

All options: -O0 -g3 -Wall -c -fmessage-length=0

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPU

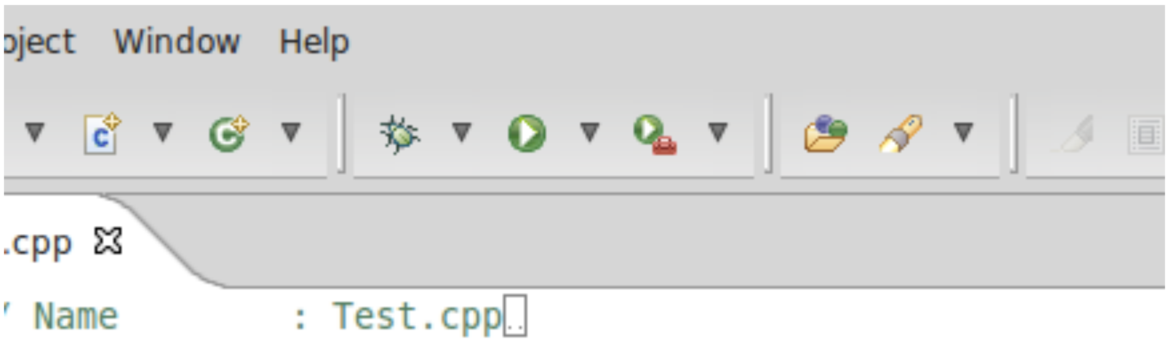
ARM GCC Compiler Command

Now you will need to go ahead and update the commands for the G++ compiler, G++ linker, and the GCC assembler. Since this is fairly repetitive I won’t show each step, but note that (at the time of writing) the assembler is the only command that does not have a 4.7 after it. If you need help feel free to post in the comments below.

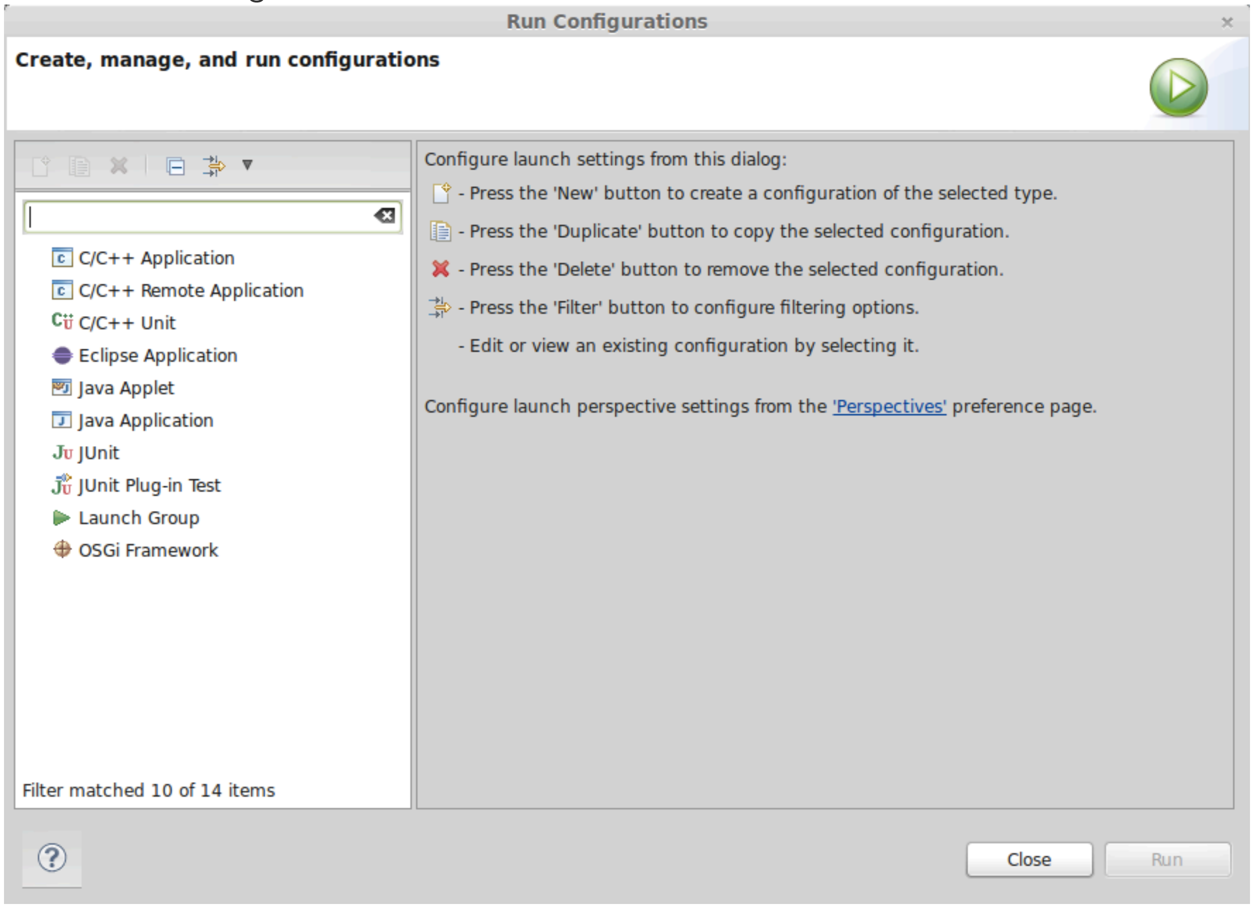
After you have updated your toolchain commands press “OK” and you will return to the main Eclipse view. To verify that everything worked as you hoped go ahead and build your project again. If it compiles without errors then congratulations, the hardest part is over and you have a binary that can run on your BeagleBone Black. This next step is just to make everything a bit easier. When we are done, you will be able to hit the run button and Eclipse will *automagically* compile your program, transfer it to the BeagleBone Black, and run it on the BeagleBone Black.

Set up Remote Deployment

Now that we have a project set up and know that it compiles, let’s set it up to run on the BeagleBone Black. You will start by clicking on the down facing arrow next to the run button, as shown below.



When you click on this arrow you will be presented with a dropdown menu, choose the “Run configurations…” option and you will see a new window that looks something like this.

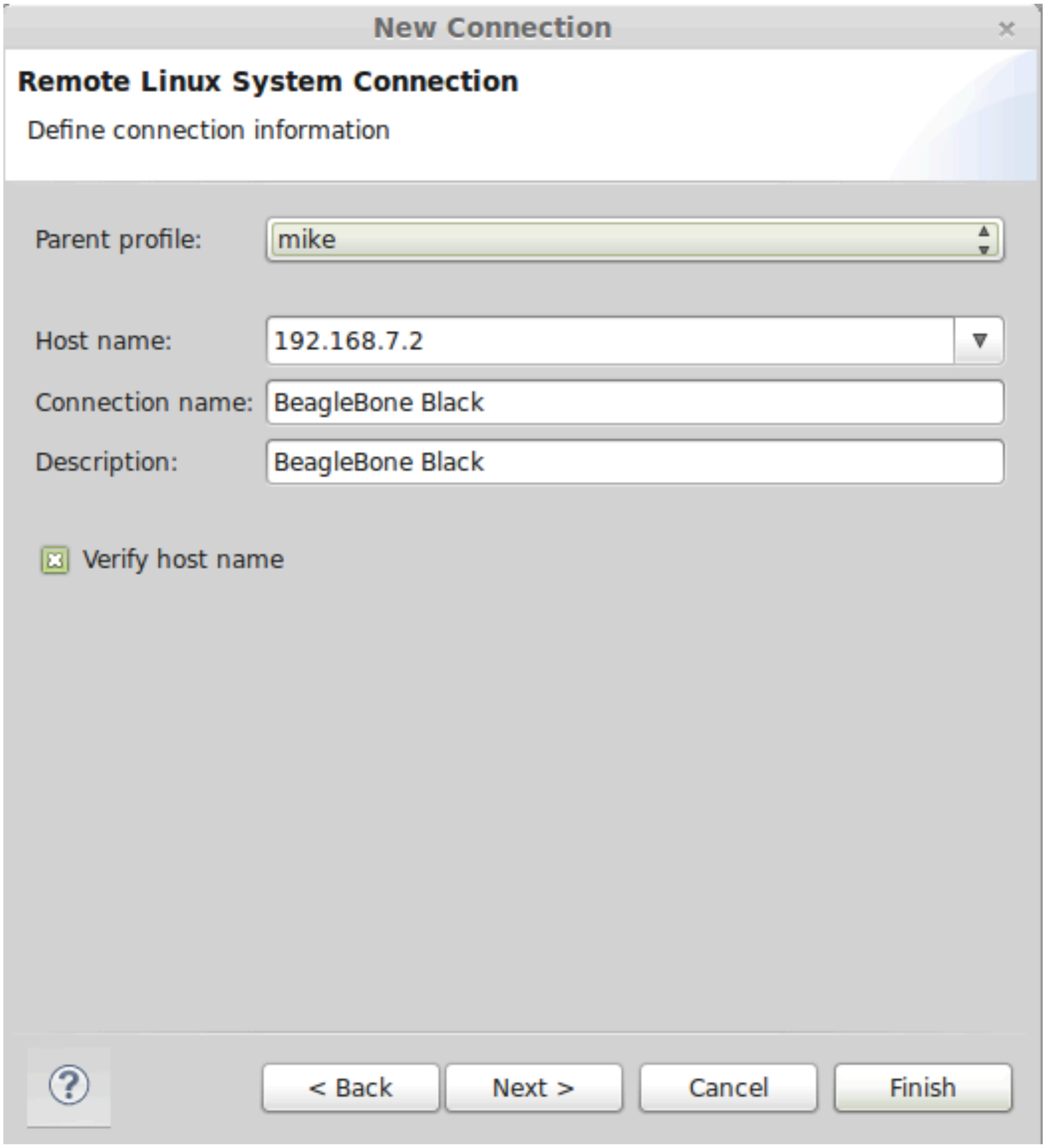


Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

You can see in the left sidebar that there is an option for “C/C++ Remote Application” if you do not see this on your system then you did not install all of the necessary plugins. Specifically, you forgot the C/C++ Remote Launch” plugin, return to the Eclipse marketplace and install this plugin. If you do see this option, then you are ready to move on to setting up your connection.
To create a new run configuration just double-click on the “C/C++ Remote Application” item, after a moment Eclipse will have created a new run

configuration and you should see a window that looks something like this.



Parent profile: mike

Host name: 192.168.7.2

Connection name: BeagleBone Black

Description: BeagleBone Black

[✓ Verify host name]

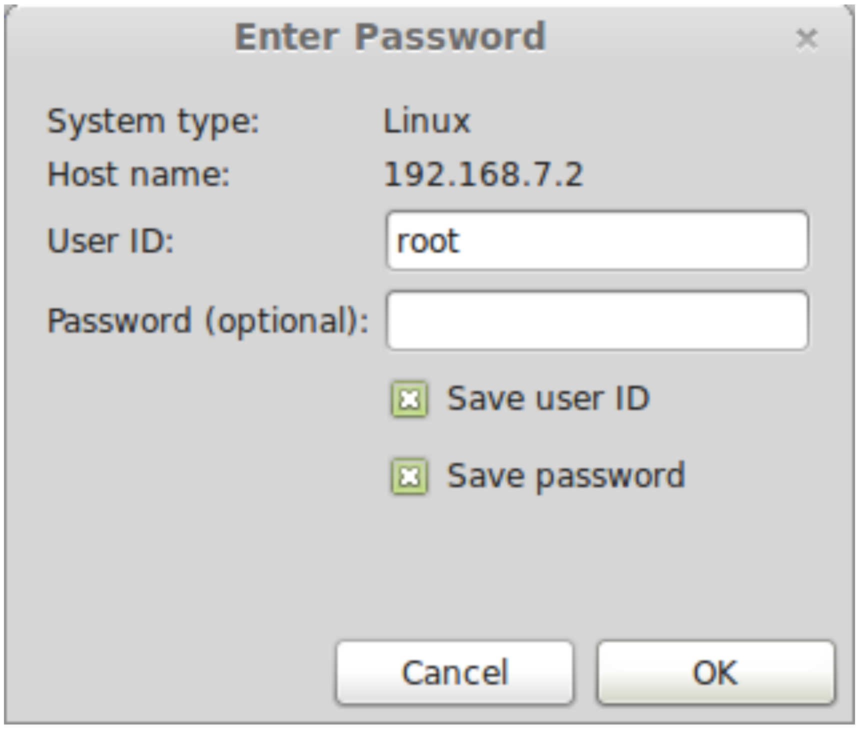
You now have the skeleton of a remote run configuration but need to inform Eclipse of which device you would like to deploy to. In order to do this you can click on the “New…” button to create a new connection which will present you with the new connection window. You will first be asked to define the remote system type, for the BeagleBone Black you will choose “Linux” but as you can see there are many other methods of connected to devices.

On the next screen you will tell Eclipse what IP address to connect to as well as what SSH profile to use. Setting up the SSH profile is outside of the scope of this tutorial, but Eclipse should be able to handle that for you.
For my BeagleBone Black (and I believe all of them) connected by USB the IP address is 192.168.7.2. Enter this number in the "Host Name" section. If you BeagleBone Black has a different IP you will want to use that instead.
Finally, you can give the connection an easy to remember name and description, or you can leave these fields blank, that's up to you. My completed connection information is as follows.

Click the next button and proceed to defining your connection protocols. These next four prompts allow you to define what protocols eclipse should use to communicate with your BeagleBone Black's subsystem. You will choose the following options:

1. ssh.files
2. processes.shell.linux
3. ssh.shells
4. ssh.terminals

To verify that your connection settings are correct, go to the Remote System Explorer view and right click on your new connection. In the dropdown menu and select "Connect…" after this you will be prompted for a user name and password. This should be the user name and password for the account you want to log-in to the BeagleBone Black with. If you haven't set up any non-default accounts or know that you will need root access you should just log in with the default root account by using the following credentials.



System type: Linux
Host name: 192.168.7.2
User ID: root
Password (optional):
[✓] Save user ID
[✓] Save password

Hit OK and you now have a working connection to the BeagleBone Black through Eclipse, you can even use Eclipse as a remote terminal now, but we won't get in to that right now. The next and final step is to configure your remote paths and to give your program execute permissions. If you don't understand what that means, don't worry, I am about to walk you through it. Return to the "Run Configurations" window you need to begin by setting the remote path of your executable. Since you haven't yet uploaded an executable to the BBB you need to define this as the program name in your root directory. So for example, in my case the program is called "Test" and my root directory is /home/root/ this means that my remote path will be:

```
/home/root/Test
```

In general you will define this as:

```
/home/root/{ProjectName}
```

Finally, you will need to grant your program execute permissions. This is essentially just a flag that tells the BeagleBone Black that it's okay to run the file through the processor. You can do this by using the chmod command with the +x flag. For my project it looks like this:

```
chmod +x /home/root/Test
```

The final run configuration is:

Apply your configuration using the "Apply" button, then run your project and check your console, you should see an output that looks something like this…

```
root@beaglebone:~# echo $PWD'>'
/home/root>
root@beaglebone:~# chmod +x /home/root/Test;/home/root/
Test;exit
Hello BeagleBone
logout
```

This shows the commands Eclipse used to run the application and shows a successful “Hello BeagleBone” message. There are any number of things that could go wrong in these steps so if you run into a problem feel free to comment below, I will do my best to help you and update this tutorial to be more clear.

```
*************************************************************************************
*************************
```

Test.c

```
#include <stdio.h>
int main(void)
{
  printf("Hello Beaglebone!\n");
  return 0;
}
```

```
************************************Another
Example*************************************************
/*******************************************************
 * Example code for user LEDs on the new beaglebone
 * black running the Linux Angstrom distribution
 *******************************************************
 * Instructions:
 * -Compile and run from the root directory
 * -For the older (white) beaglebone, change
 *  "beaglebone:green:usr0" to "beaglebone::usr0"
 *
 * Code adapted from:
 * - Derek Molloy, "Beaglebone: C/C++ Programming
 * Introduction for ARM Embedded Linux Development
 * using Eclipse CDT" video tutorial,
 * (link: www.youtube.com/watch?v=vFv_-ykLppo)
 * -  Mark A. Yoder, EBC Exercise 10 Flashing an LED
 * (link: "elinux.org/EBC_Exercise_10_Flashing_an_LED)
 *****************************************************/

#include <stdio.h>
#include <unistd.h>

using namespace std;

int main(int argc, char** argv) {

  FILE *LEDHandle = NULL;
  char *LEDBrightness = "/sys/class/leds/beaglebone:green:usr0/
brightness";
  printf("\nStarting simple LED blink program\n");

  while(1){

    if((LEDHandle = fopen(LEDBrightness, "r+")) != NULL){
      fwrite("1", sizeof(char), 1, LEDHandle);
      fclose(LEDHandle);
    }

    sleep(1);

    if((LEDHandle = fopen(LEDBrightness, "r+")) != NULL){
      fwrite("0", sizeof(char), 1, LEDHandle);
      fclose(LEDHandle);
    }

    sleep(1);
  }
  return 0;

}

*************************************************************************************
*************************
```

REFERENCE: http://learnbuildshare.wordpress.com/2013/05/29/beaglebone-black-digital-ouput/

BeagleBone Black – Digital output using C++

Posted on May 29, 2013 by learnbuildshare
Did some more digging after my last post and came up with C++ example for digital output on the BBB. I used it to drive an LED. This example controls GPIO1_28 (pin 12 on the P9 header). All the pins are listed on the user manual. Note that this pin is treated as pin 60 in the code. This is how it's set up in Linux. Here's how to convert. GPIOn_m is n x 32 + m. So GPIO1_28 is 32 x 1 + 28 = 60. There are four steps to the process of using a GPIO pin. First, you export the pin, i.e. tell the OS that you want to use it for something. Second, you set the direction, e.g. "out" in this case. Third, you do what you want with the pin. In this example we toggle it between high and low. Fourth, when you are done with the pin, you must unexport it. Doing any of the above four steps with pins is like writing to a file. That's why you need to check if the file is accessible right before every file write. And right after the file write, you need to close the file handle. It seems that this needs to be done after every file write. If you blink an LED 50 times, you close the handle after each high/low event.

Lastly, a note on the hardware side of things. The BBB is a 3.3V board and is NOT 5V tolerant. The recommended source/sink current is only about 4-6 mA. So if you're driving an LED, use a transistor, say, a PN2222A. Use a 10K resistor on the base, and 100 ohm on the collector. For this setup, connect pin 7 on the P9 header (SYS_5V) to the LED. This is 5V from the USB. Connect the ground to pin 45 (GND) on the P9 header. Be sure to check the data sheet for your transistor. Different variations of the same transistors have different pinouts. I nearly connected mine all wrong while looking at the connections in one video I was following. Turned out I had a different 2222 transistor with different pinouts.

```
/*******************************************************
* Example code for digital output on a GPIO pin
* on the new beaglebone black running the
* Linux Angstrom distribution.
********************************************************
*
* Instructions:
```

```
* -Compile and run from the root directory
* -Windows users can use WinSCP to transfer
* code files to the beaglebone. This will
* allow you to edit code in your favorite
* editor on Windows
* -It is recommended that you do not try to
* source more than 4 - 6 mA. Use a transistor
* e.g. the PN2222A to source higher currents,
* such as for driving an LED
*
* Code adapted from:
* - Derek Molloy, "Beaglebone: C/C++ Programming
* Introduction for ARM Embedded Linux Development
* using Eclipse CDT" video tutorial,
* (link: <a href="http://www.youtube.com/watch?v=SaIpz00lE84"
title="http://www.youtube.com/watch?v=SaIpz00lE84">http://
www.youtube.com/watch?v=SaIpz00lE84</a>)
* -  Mark A. Yoder, EBC Exercise 10 Flashing an LED
* (link: http://www.elinux.org/EBC_Exercise_10_Flashing_an_LED)
*
*******************************************************/

#include <stdio.h>
#include <unistd.h>
#include <string.h>

using namespace std;

int main(int argc, char** argv) {

    int GPIOPin=60, /* GPIO1_28 or pin 12 on the P9 header */ times=10;

    printf("\nStarting GPIO output program\n");
    FILE *myOutputHandle = NULL;
    char setValue[4], GPIOString[4], GPIOValue[64], GPIODirection[64];
    sprintf(GPIOString, "%d", GPIOPin);
    sprintf(GPIOValue, "/sys/class/gpio/gpio%d/value", GPIOPin);
    sprintf(GPIODirection, "/sys/class/gpio/gpio%d/direction", GPIOPin);
```

```
// Export the pin
if ((myOutputHandle = fopen("/sys/class/gpio/export", "ab")) == NULL){
    printf("Unable to export GPIO pin\n");
    return 1;
}
strcpy(setValue, GPIOString);
fwrite(&setValue, sizeof(char), 2, myOutputHandle);
fclose(myOutputHandle);

// Set direction of the pin to an output
if ((myOutputHandle = fopen(GPIODirection, "rb+")) == NULL){
    printf("Unable to open direction handle\n");
    return 1;
}
strcpy(setValue,"out");
fwrite(&setValue, sizeof(char), 3, myOutputHandle);
fclose(myOutputHandle);

for(int i=0; i<times; i++){
    // Set output to high
    if ((myOutputHandle = fopen(GPIOValue, "rb+")) == NULL){
        printf("Unable to open value handle\n");
        return 1;
    }
    strcpy(setValue, "1"); // Set value high
    fwrite(&setValue, sizeof(char), 1, myOutputHandle);
    fclose(myOutputHandle);
    sleep(1); // wait for 1 sec

    // Set output to low
    if ((myOutputHandle = fopen(GPIOValue, "rb+")) == NULL){
        printf("Unable to open value handle\n");
        return 1;
    }
    strcpy(setValue, "0"); // Set value low
    fwrite(&setValue, sizeof(char), 1, myOutputHandle);
    fclose(myOutputHandle);
```

```
        sleep(1); // wait for 1 sec

    }

    // Unexport the pin
    if ((myOutputHandle = fopen("/sys/class/gpio/unexport", "ab")) == NULL)
{
        printf("Unable to unexport GPIO pin\n");
        return 1;
    }
    strcpy(setValue, GPIOString);
    fwrite(&setValue, sizeof(char), 2, myOutputHandle);
    fclose(myOutputHandle);
    printf("\nCompleted GPIO output program\n");

    return 0;
}
```

****

References

This was a complex article and took quite a bit of research on my part. I came across several other articles that would cover a part of the process but not the whole flow, so I decided to combine all these articles as well as my own personal experiences into a unified place. I hope you have been able to learn everything you came for but if not feel free to comment below or check out these articles I read along the way.

fortune datko - Cross-compiling applications for the BeagleBone

Jan Axelson - Using Eclipse to Cross-Compile Applications for Embedded Systems

Derek Malloy - Beaglebone: C/C++ Programming Introduction for ARM Embedded Linux Development using Eclipse CDT