OpenBoard-phyCORE-AM335x

Application Devlopment Kit (ADK)

SOM Product No: PCM-051

Carrier Board Product No: PCM-950

Release 1.0
January,2013

In this manual copyrighted products are not explicitly indicated. The absence of the trademark (™) and copyright (©) symbols does not imply that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, PHYTEC Embedded Pvt. Ltd. assumes no responsibility for any inaccuracies. PHYTEC Embedded Pvt. Ltd neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. PHYTEC Embedded Pvt. Ltd reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages that might result.

Additionally, PHYTEC Embedded Pvt. Ltd offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. PHYTEC Embedded Pvt. Ltd further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

	India	Europe	North America
Address:	PHYTEC Embedded Pvt. Ltd. Koramangala 8th block, Bangalore -95 India	PHYTEC Technologie Holding AG Robert-Koch-Str. 39 55129 Mainz GERMANY	PHYTEC America LLC 203 Parfitt Way SW, Suite G100 Bainbridge Island, WA 98110 USA
Ordering Information:	+91-80-41307589 Sales@phytec.in	+49 (800) 0749832 order@phytec.de	1 (800) 278-9913 sales@phytec.com
Web Site:	http://www.phytec.in	http://www.phytec.de	http://www.phytec.com

Table of Contents

Embedded Linux Appliation Development for OpenBoard-AM335x

In this Manual we are going to describe how to use this Open Board for application development. Our first chapter is dealing with the instalation of eclipse and how a user can developed his own application which would run on open board easilly with the help of eclipse. In our second chapter we will describe how to write an application using console terminal. After going through this manual you will not only familiar with Open board as well as you will also have an idea how to use eclipse and console terminal for application development.

1. Application development using Eclipse IDE

With the help of example projects, we will teach you how to work with eclipse. First we take a look on the C programming language. At the end of this chapter we explain how to debug your written programs when running on the target.

1.1. Eclipse IDE Installaion

First of all get the latest eclipse i.e, **Eclipse IDE for C/C++ Developers** and install it.

Eclipse IDE for C/C++ Developers

http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/junosr1

Eclipse IDE for C/C++ Developers for Windows 32-bit

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/juno/SR1/eclipse-cpp-juno-SR1-win32.zip

Note: Eclipse required java also so we need to install the java.

http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html

Eclipse IDE for C/C++ Developers for Windows 64-bit

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/juno/SR1/eclipse-cpp-juno-SR1-win32-x86_64.zip

Eclipse IDE for C/C++ Developers for Linux 32-bit

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/juno/SR1/eclipse-cpp-juno-SR1-linux-gtk.tar.gz

Note: Eclipse required java also so we need to install the java using this command.

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

Eclipse IDE for C/C++ Developers for Linux 64-bit

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/juno/SR1/eclipse-cpp-juno-SR1-linux-gtk-x86_64.tar.gz

Eclipse IDE for C/C++ Developers for Mac OS X(Cocoa 32)

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/juno/SR1/eclipse-cpp-juno-SR1-macosx-cocoa.tar.gz

Eclipse IDE for C/C++ Developers for Mac OS X(Cocoa 64)

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/juno/SR1/eclipse-cpp-juno-SR1-macosx-cocoa-x86_64.tar.gz

1.2. Eclipse IDE Configuration for OpenBoard-AM335x

1.2.1. Host Setup

Toolchain: For Compiling the Application we need the toolchain which you can easilly download from the below link.

For Linux:

ftp://ftp.phytec.de/pub/Products/India/OpenBoard-AM335x/Linux/latest/tools/toolchain/arm-cortexa8-linux-gnueabi.tar.bz2

Untar the toolchain first the prefix of the toolchain is **arm-cortexa8-linux-gnueabi-**

For Window:

http://sourcery.mentor.com/public/gnu_toolchain/arm-none-linux-gnueabi/arm-2012.09-64-arm-none-linux-gnueabi.exe

Install the toolchain the prefix of the toolchain is **arm-none-linux-gnueabi-** and default path is ***C:\Program Files\CodeSourcery\Sourcery_CodeBench_Lite_for_ARM_GNU_Linux\bin***

1.2.2. Target Setup

Connect the power adaptor, serial cable, ethernet cables to the open board.
Boot the open board.

1.2.1.1. Set the ip for the open board.

To see all the interfaces present on the open board

ifconfig -a

After this we got the particular interface then we have to configure it.

ifconfig eth0 192.168.1.11 up

where eth0 is the interface name.

Again check whether it is configured or not

ifconfig -a

Then we have to set the gateway

route add default gw 192.168.1.1

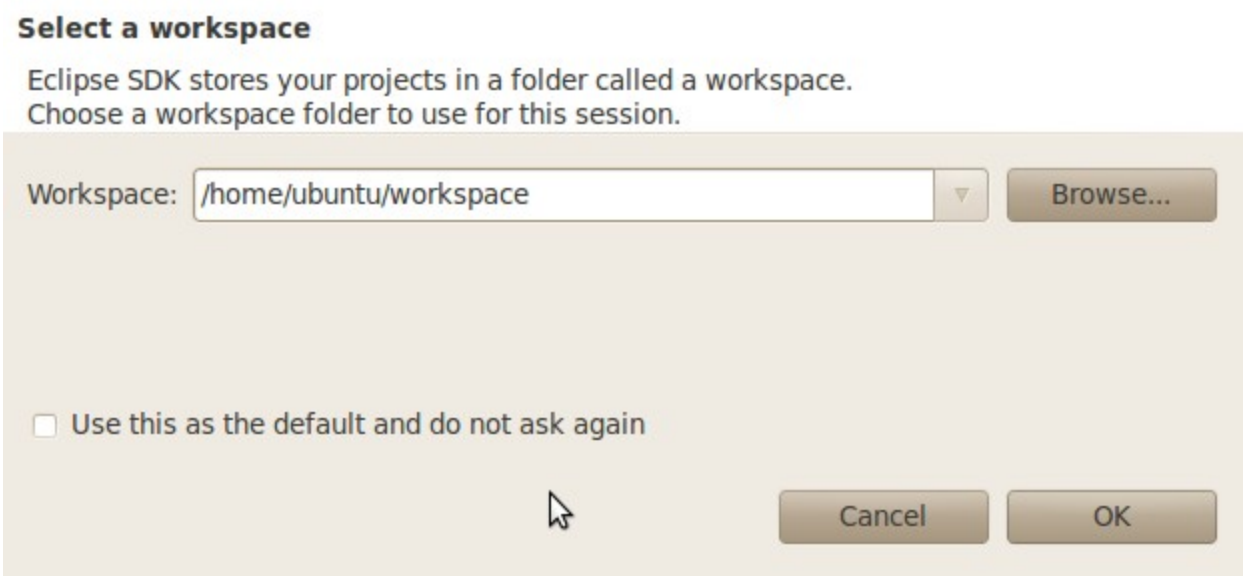
To see the change in the gateway.

route

1.3. Creating a New Project

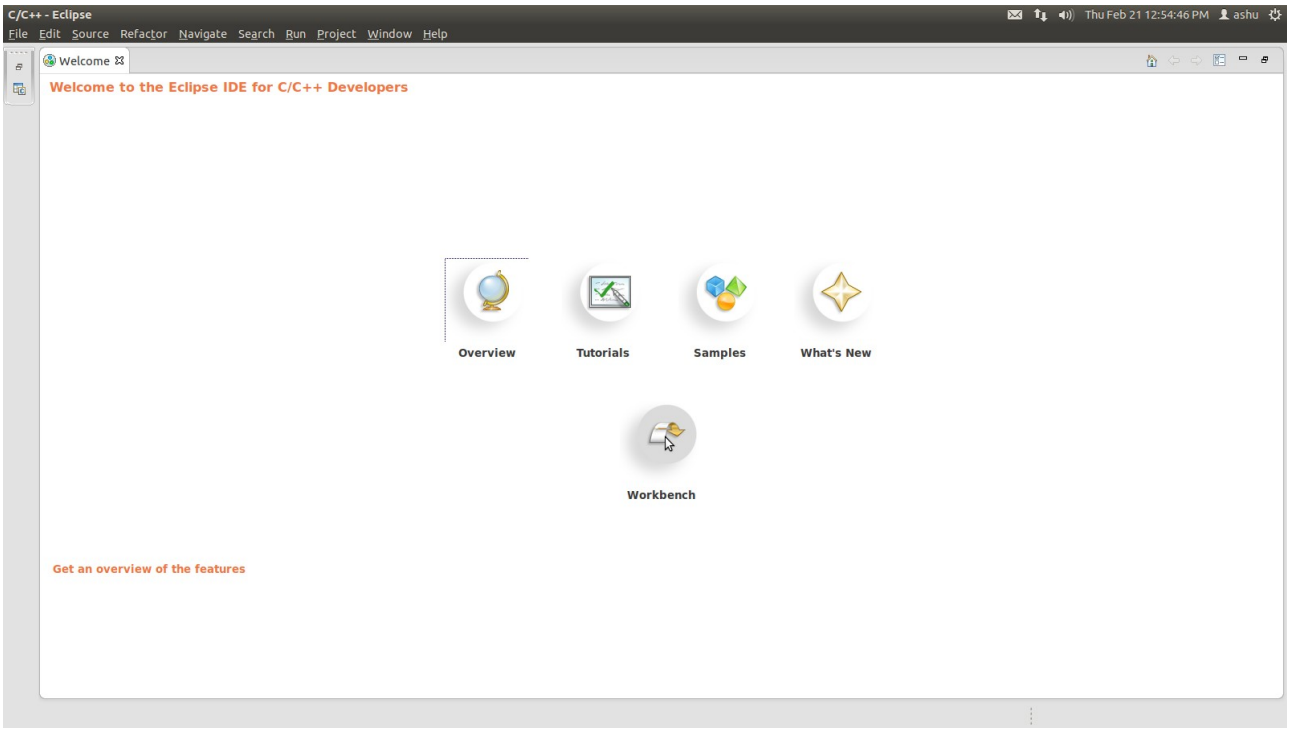
In this section you will learn how to create a new project with Eclipse and how to configure the project for use with the GNU C/C++ cross development toolchain. Click the Eclipse icon to start the application. You can find this icon where you have extracted the Eclipse IDE for C/C++ Developers .

1.3.1. How to open eclipse



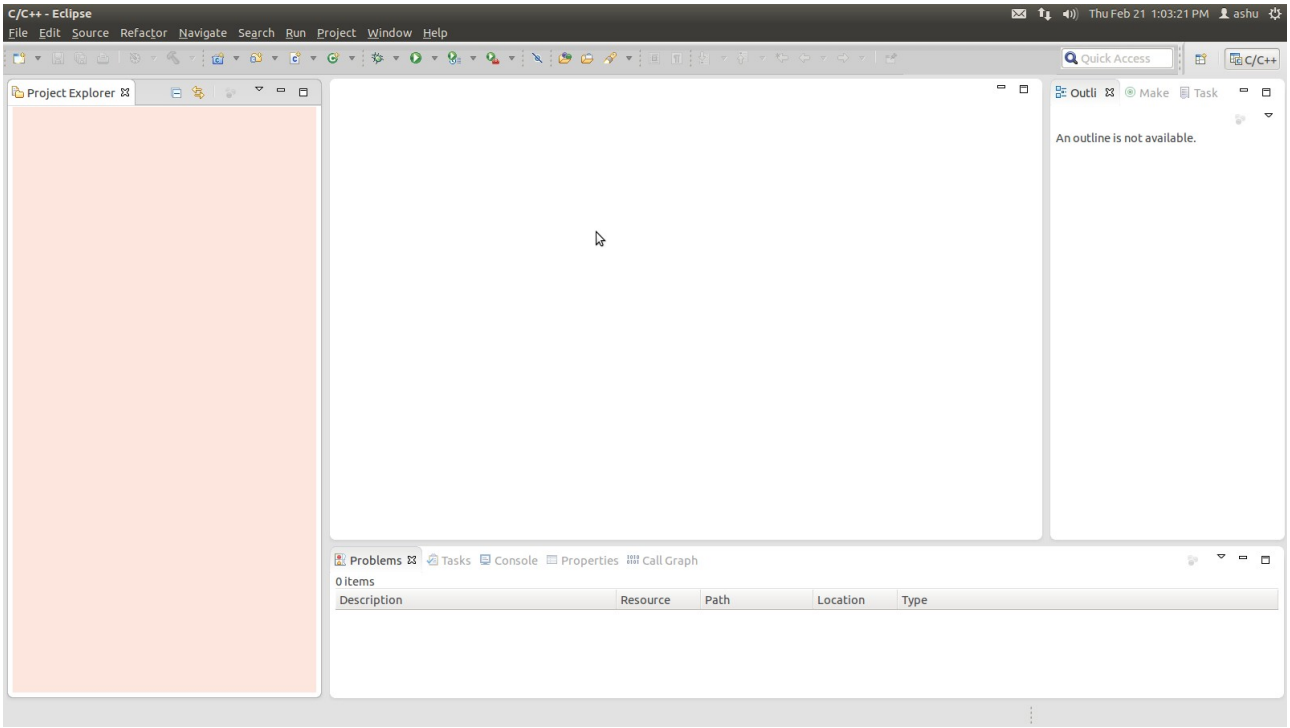
Confirm the workspace directory with OK

1.3.2. Welcome to Eclipse



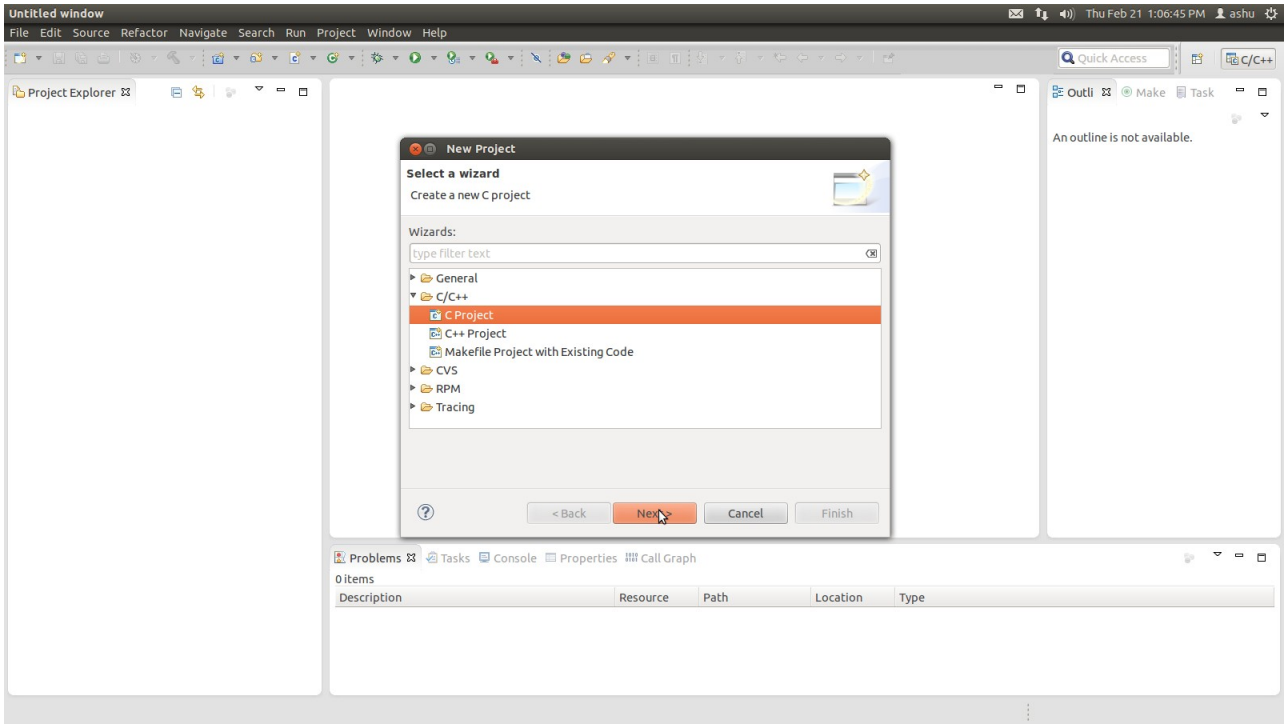
Close the "Welcome to Eclipse" screen by clicking on the "Go to the workbench" button

1.3.3. Open a new project



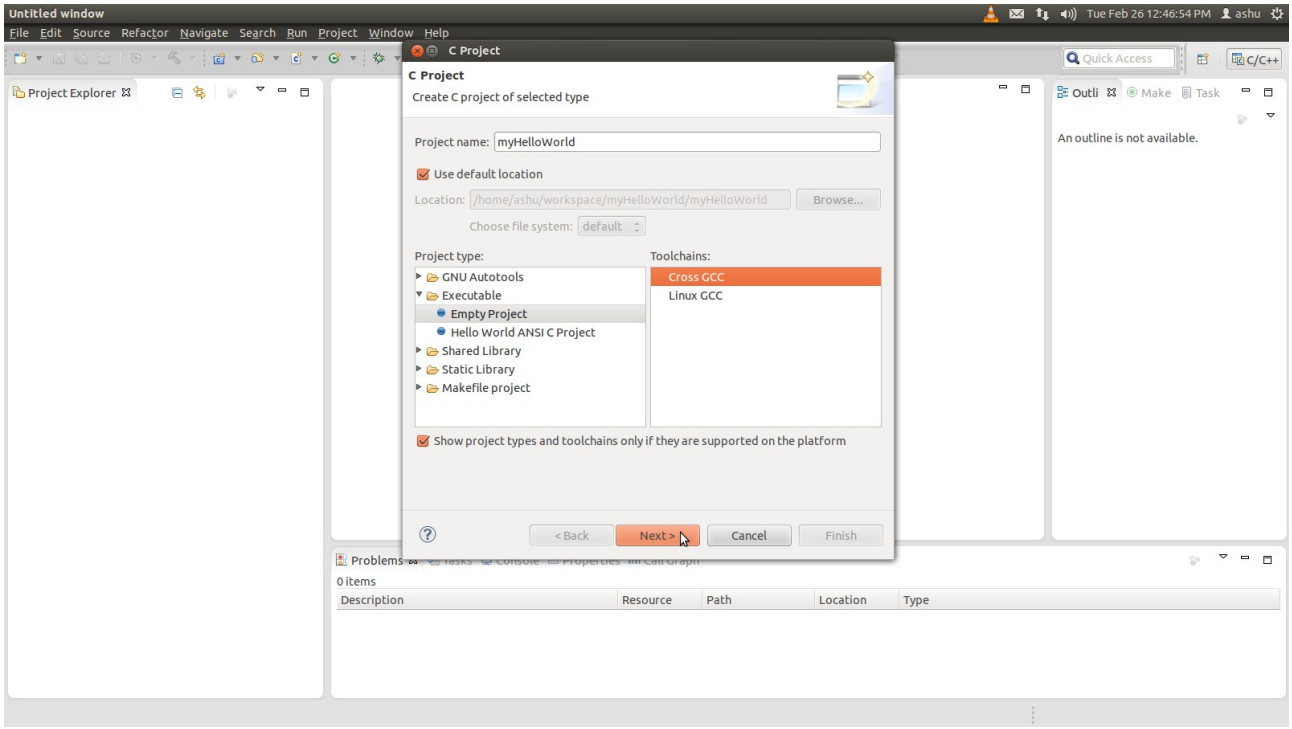
• Select File ► New ► Project from the menu bar
A new dialog opens.

1.3.4. Select C Project



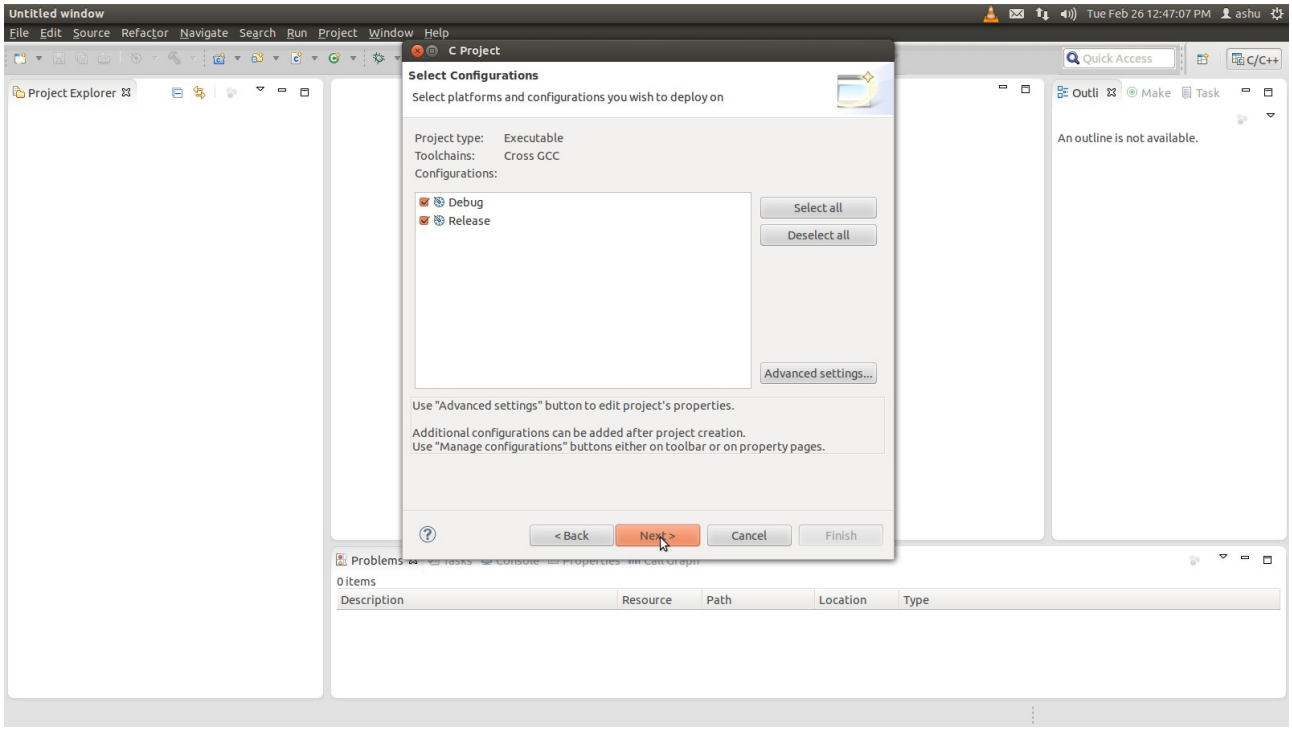
Select C Project and click Next

1.3.5. Select the Cross GCC



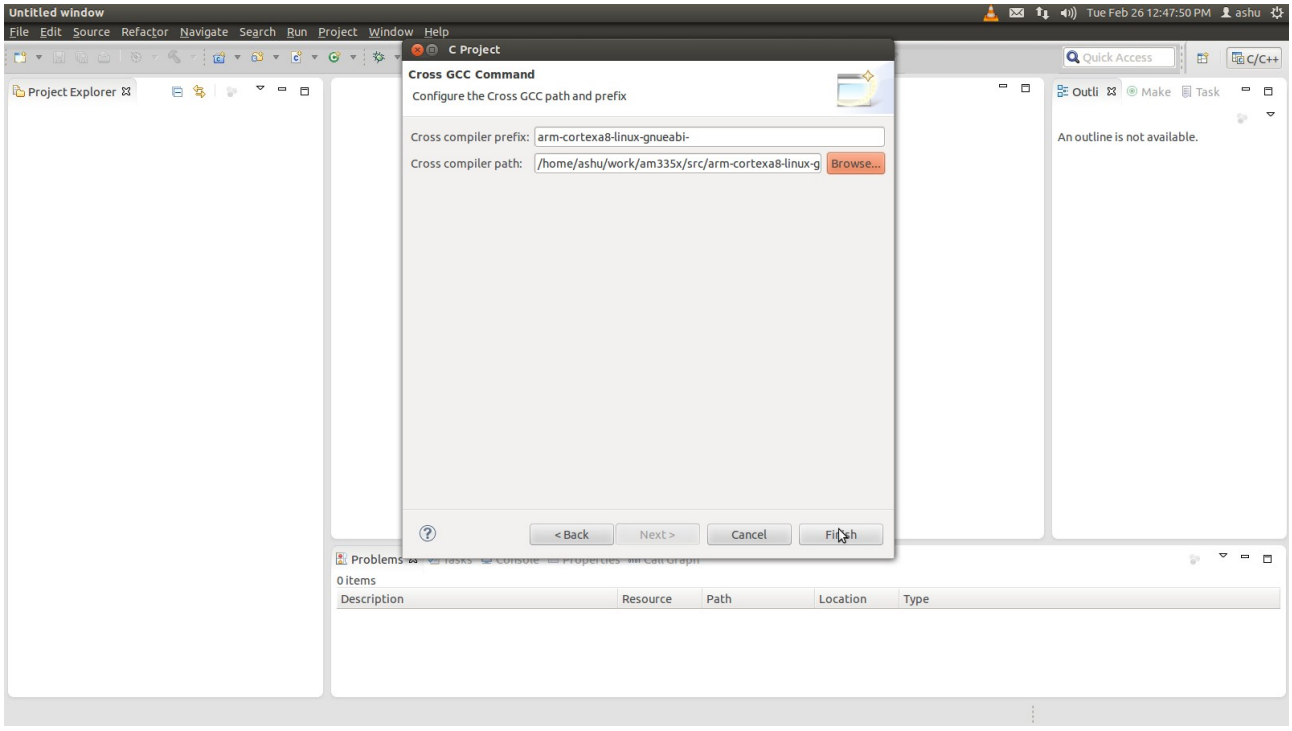
Enter the project name myHelloWorld and Toolchain as Cross GCC then click Next

1.3.6. Select Debug Facilities



Click Next

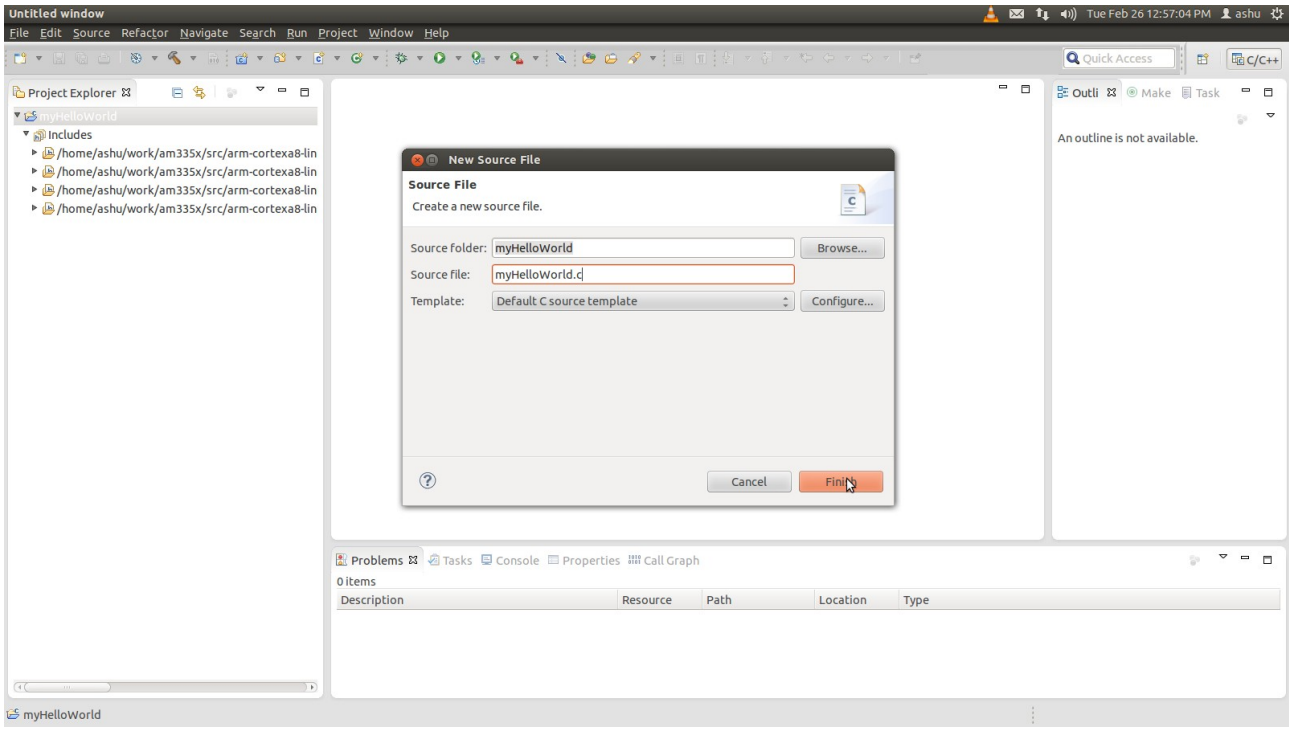
1.3.7 Toolchain Prefix & Path



Select the Cross Compiler Prefix as **arm-cortexa8-linux-gnueabi-**
and Cross Compiler Path as <path of toolchain bin>

Note: for window you have to select the **arm-none-linux-gnueabi-** and the appropriate path of the toolchain.

1.3.8. Open new C source file



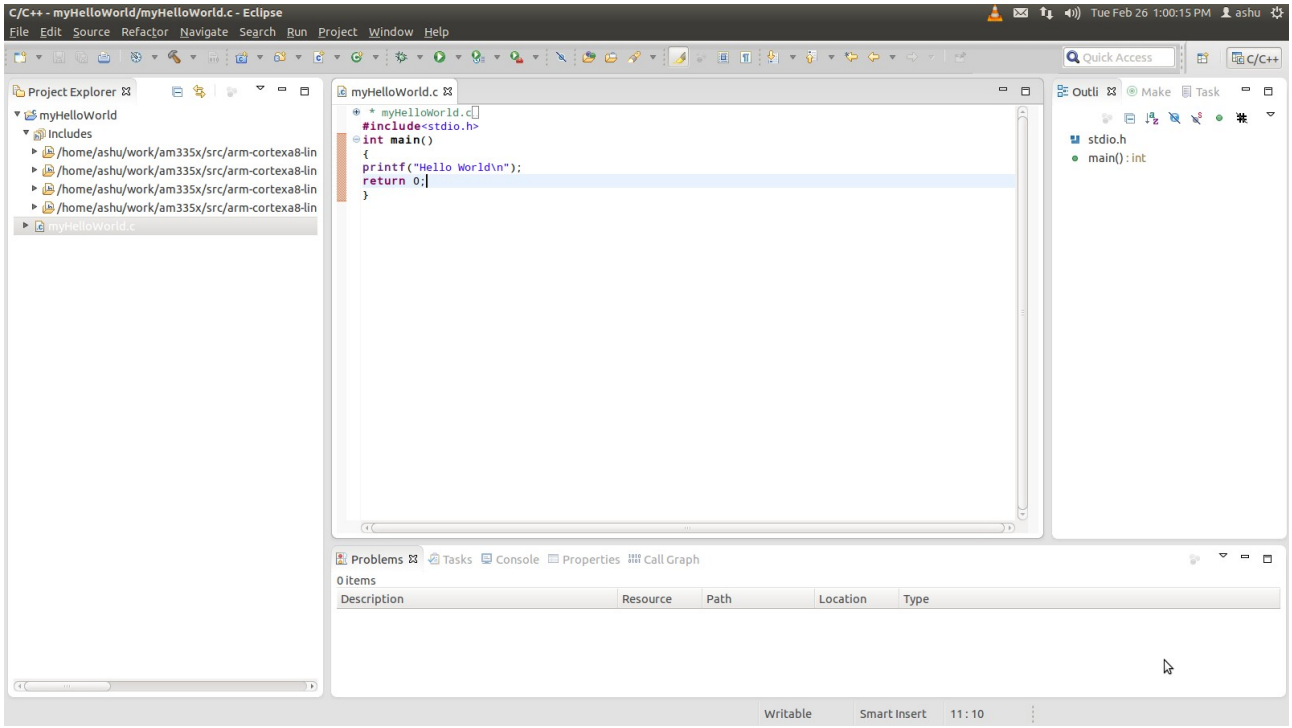
You will see the C/C++ IDE with the myHelloWorld project.

Right-click on HelloWorld project

Select File ► New ► Source file from the menu bar

In Source file write myHelloWorld.c and click on finish

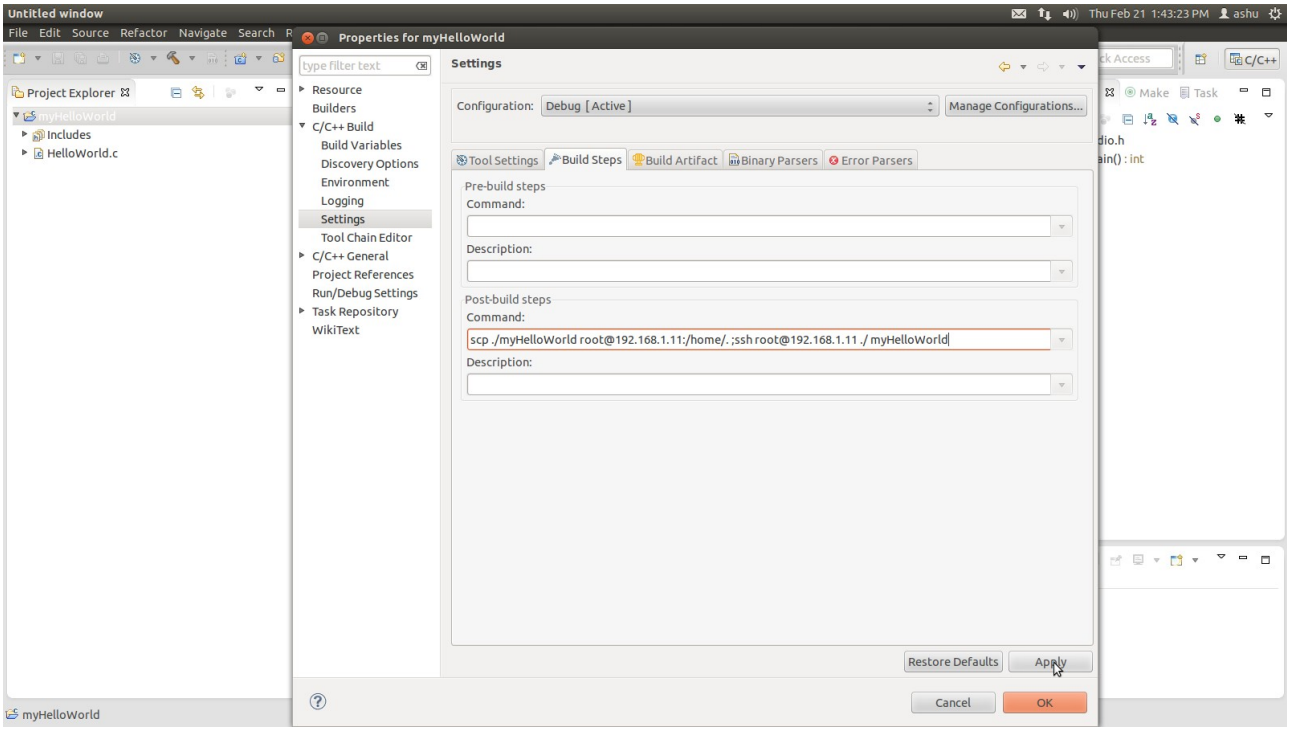
1.3.9. Write simple Hello Application



Write a simple Hello Application in C.

To compile your project for the Open Board-AM335X instead, you will have to use the GNU C/C++ cross compiler.

1.3.10. Modify Post build steps



Right-click the myHelloWorld project and chose Properties

The Properties dialog appears.

Select C/C++ Build
----> Setting
------->Select the Build Steps tab

Enter the following command in the Post-build steps Command input field:

scp ./myHelloWorld root@192.168.1.11:/home/. ;ssh root@192.168.1.11 ./myHelloWorld

Note: If you are using Window machine then there is no use of writing the above line it will through error so you have to use Winscp or directly copy the binary into target board using pen drive or sd card.

You can download the winscp from the below link:

http://download.winscp.net/download/files/201302271159a5e0d9e193373cacc9998a2df283e19a/winscp514setup.exe

Click Apply

Click OK

Select Project ► Build project from the menu bar

The project will be built.

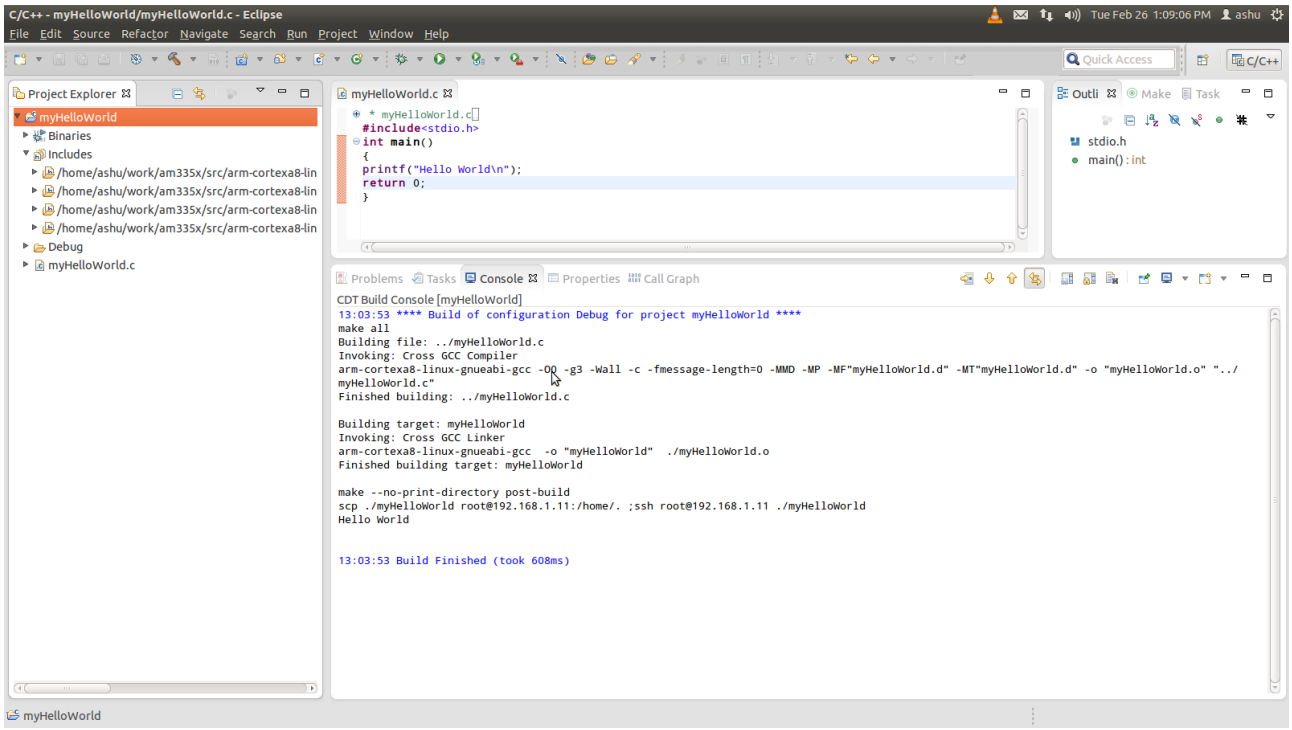
Select the Console tab.

if no errors occur while building the project, you will see the following output:

Note: If you are using Window machine then you need the make utils using te below link you can download it.

ftp://ftp.equation.com/make/32/make.exe

1.3.11. Finally Build the project



1.4. Changing the Demo Application

Now we will extend the myHelloWorld application. The extended myHelloWorld application will write an output to the first serial interface as well as to the standard output.

Open Eclipse if it is not opened yet

Double-click HelloWorld.c in the myHelloWorld project

First include the following two additional header files:

```
#include <unistd.h>
#include <fcntl.h>
```

Then add the function write_tty(), which writes n bytes to the first serial interface

(which, on the Open-Board-AM335x, is connected to the system console /dev/console):

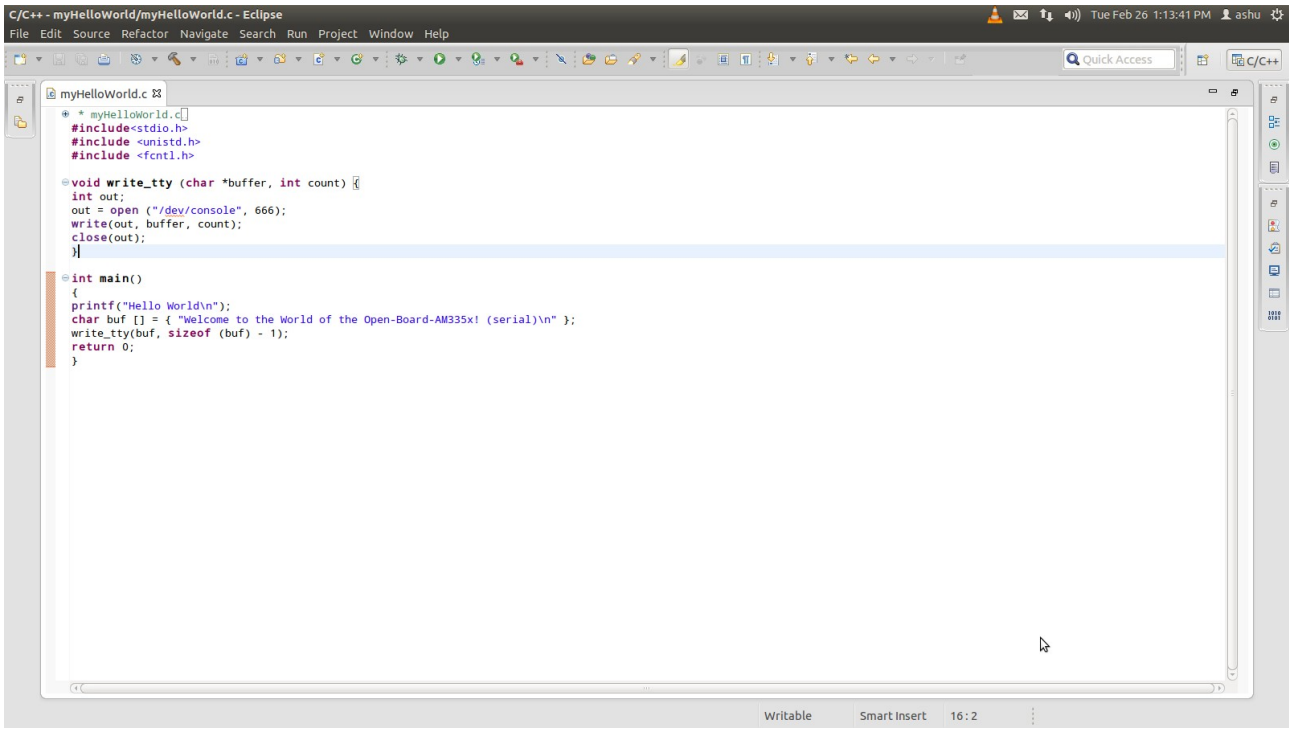
```
void write_tty (char *buffer, int count) {
int out;
out = open ("/dev/console", 666);
write(out, buffer, count);
close(out);
}
```

Enter the following two lines in the main() function to declare the buffer and call the

write_tty() function.

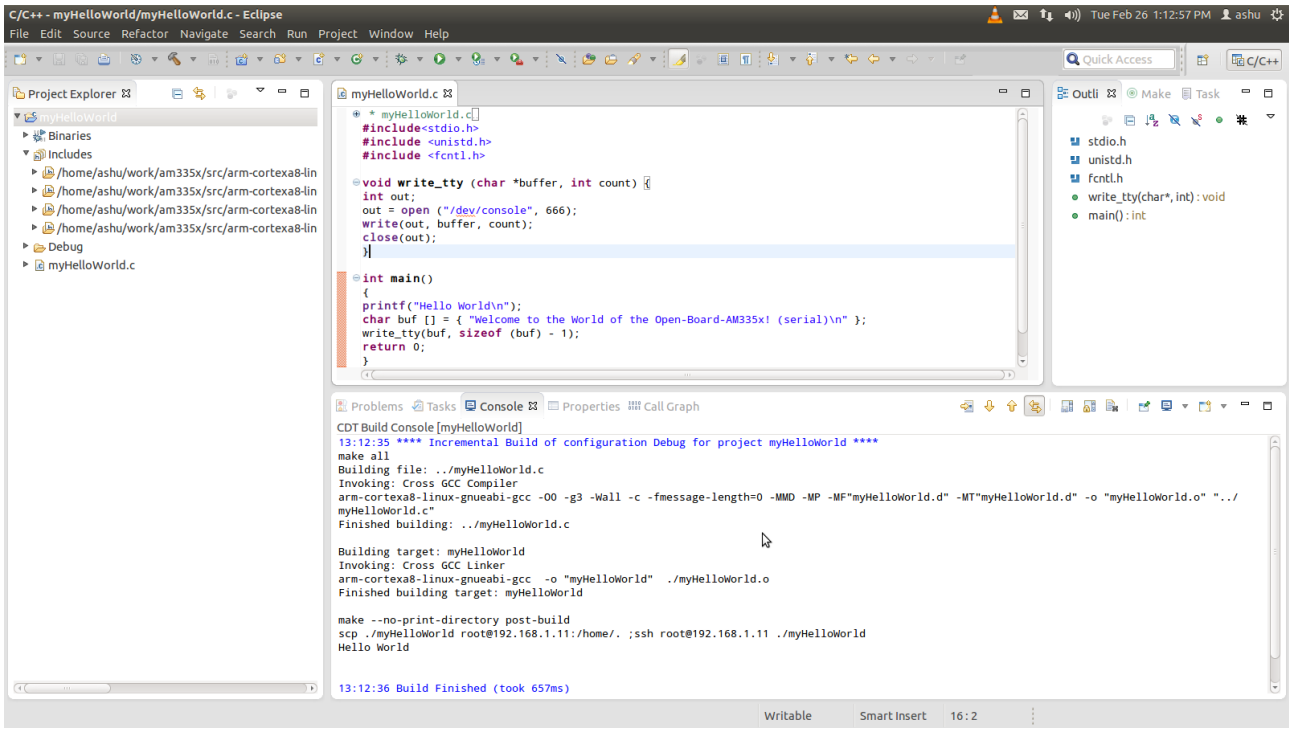
```
char buf [] = { "Welcome to the World of the Open-Board-AM335x! (serial)\n" };
write_tty(buf, sizeof (buf) - 1);
```

In the next screenshot you can see the complete program.



Save your program after changing the code.

The application will be compiled, built, copied to the target and executed.



1.4.1. Open Target Board using Minicom

Open the terminal using minicom

minicom -D /dev/ttyXX

If you are not logged in, enter root and press Enter
then type ls to see all the file.

ls

Type ./myHelloWorld to start the application

./myHelloWorld

Hello World

Welcome to the World of the Open-Board-AM335x! (serial)

close minicom.

In this section you have changed an existing application. You also learned how to access

the serial interface. First you called the function open() on the device /dev/console.

The return value of this function was a file descriptor. With the file descriptor you called

the function write() to send n bytes to the device /dev/console. After that, the file descriptor was closed with the function close().

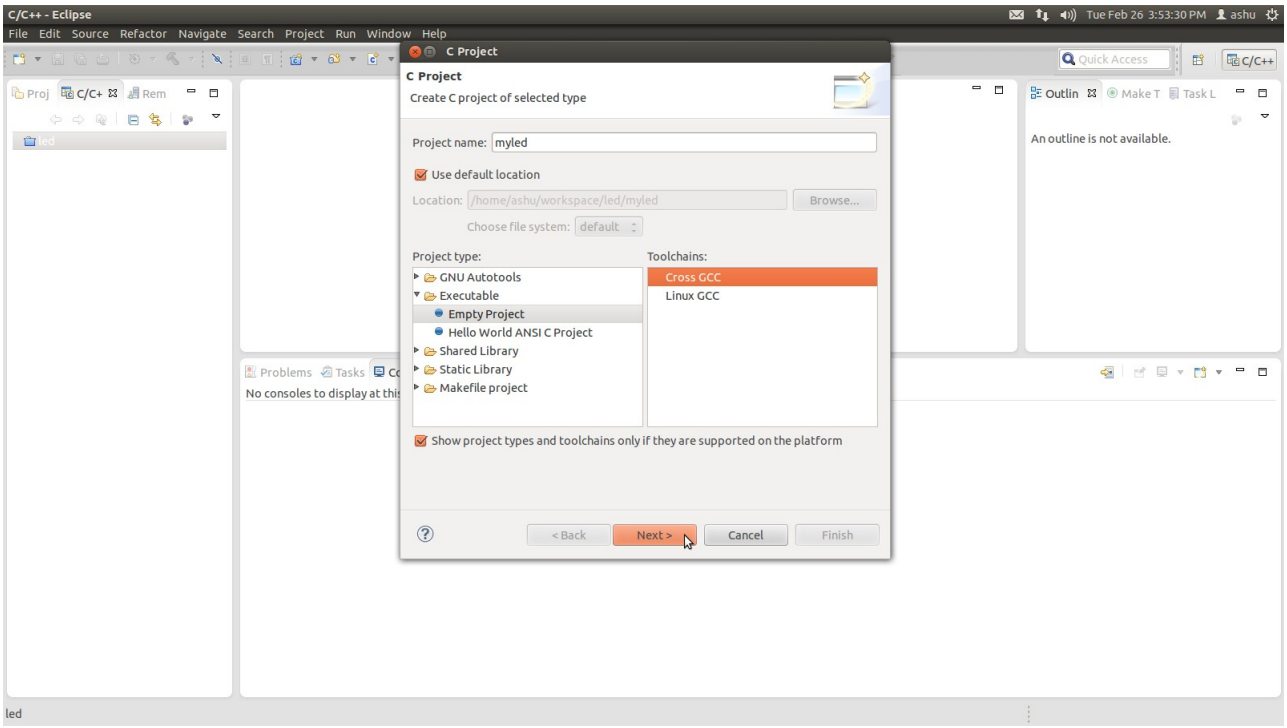
1.5. Led Blinking Application

In this section we are dealing with some hardware such as led on th e Open Board. For writing an application for led first follows Step 1.3.1. to Step -1.3.4. as discuss above. After then

1.5.1. Select the Cross GCC

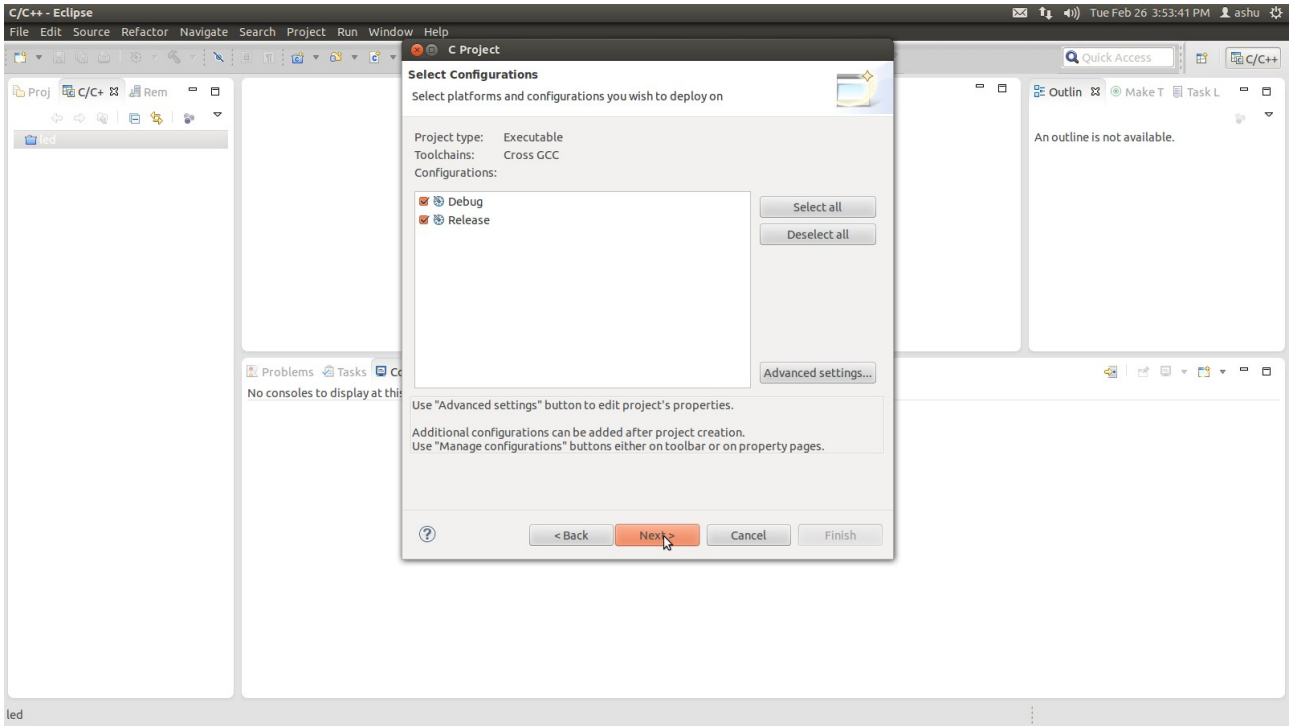
• Select File ► New ► C Project from the menu bar

A new dialog opens.



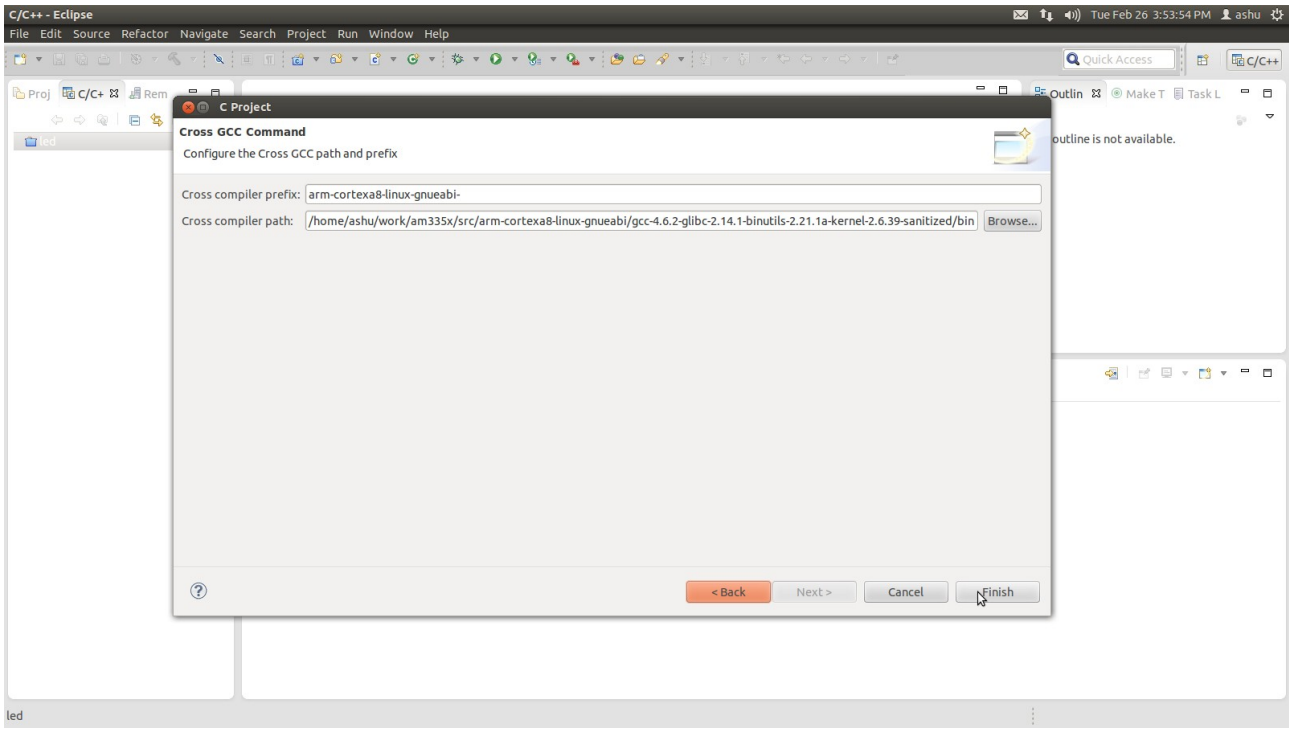
Enter the project name myled and Toolchain as Cross GCC then click Next

1.5.2. Select Debug Facilities



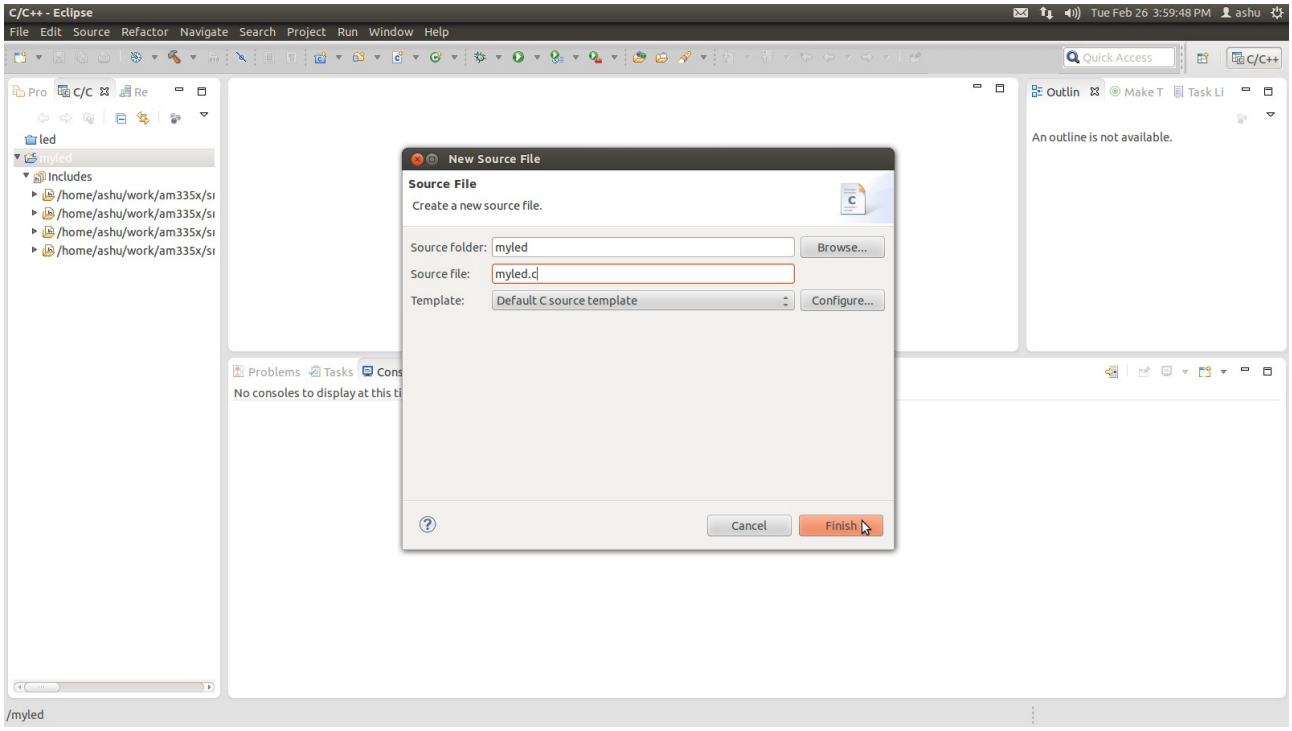
Click Next

1.5.3. Toolchain Prefix & Path



Select the Cross Compiler Prefix as **arm-cortexa8-linux-gnueabi-**
and Cross Compiler Path as <path of toolchain bin>

1.5.4. Open new C source file



You will see the C/C++ IDE with the led project.

Right-click on led project

Select File ► New ► Source file from the menu bar

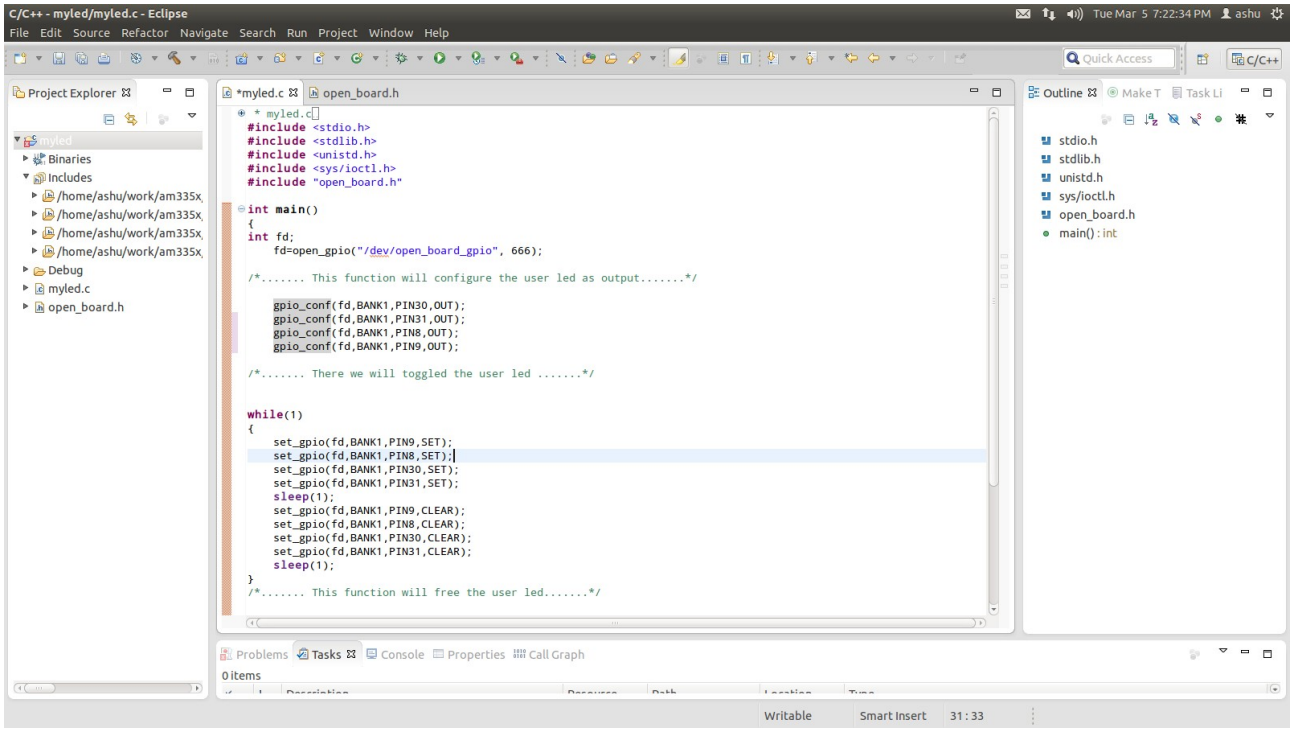
In Source file write myled.c and click on finish

1.5.5. Write Led Application

Then write a led application as:

```
/*
 * myled.c
 *
 *  Created on: 26-Feb-2013
 *      Author: Ashutosh
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include "open_board.h"

int main()
{
int fd;
        fd=open_gpio("/dev/open_board_gpio", 666);

/*..... This function will configure the user led as output.....*/

        gpio_conf(fd,BANK1,PIN30,OUT);
        gpio_conf(fd,BANK1,PIN31,OUT);
        gpio_conf(fd,BANK1,PIN8,OUT);
        gpio_conf(fd,BANK1,PIN9,OUT);

/*..... There we will toggled the user led .....*/

while(1)
{
        set_gpio(fd,BANK1,PIN9,SET);
        set_gpio(fd,BANK1,PIN8,SET);
        set_gpio(fd,BANK1,PIN30,SET);
        set_gpio(fd,BANK1,PIN31,SET);
        sleep(1);
        set_gpio(fd,BANK1,PIN9,CLEAR);
        set_gpio(fd,BANK1,PIN8,CLEAR);
        set_gpio(fd,BANK1,PIN30,CLEAR);
        set_gpio(fd,BANK1,PIN31,CLEAR);
        sleep(1);
}
/*..... This function will free the user led.....*/

      gpio_free(fd,BANK1,PIN30);
      gpio_free(fd,BANK1,PIN31);
      gpio_free(fd,BANK1,PIN8);
      gpio_free(fd,BANK1,PIN9);

/*..... This function will close the node.....*/

      close_gpio(fd);
```

```
return 0;
}
```



1.5.6. Write Open Board Header

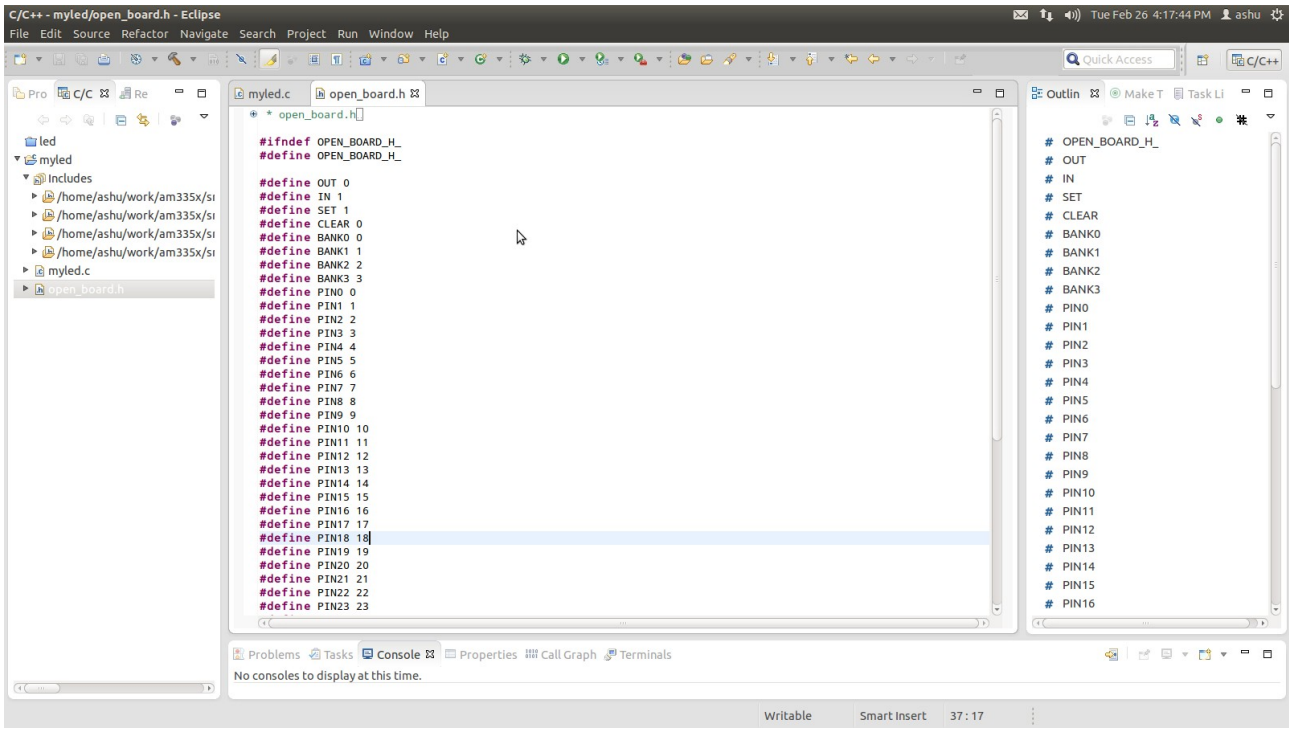
Select File ► New ► Header file from the menu bar

In Header file write open_board.h and click on finish

Then write the Header file as.

```
/*
 * open_board.h
 *
 *  Created on: 26-Feb-2013
 *      Author: Ashutosh
 */

#ifndef OPEN_BOARD_H_
#define OPEN_BOARD_H_
#define OUT 0
#define IN 1
#define SET 1
#define CLEAR 0
#define BANK0 0
#define BANK1 1
#define BANK2 2
```

```
#define BANK3 3
#define PIN0 0
#define PIN1 1
#define PIN2 2
#define PIN3 3
#define PIN4 4
#define PIN5 5
#define PIN6 6
#define PIN7 7
#define PIN8 8
#define PIN9 9
#define PIN10 10
#define PIN11 11
#define PIN12 12
#define PIN13 13
#define PIN14 14
#define PIN15 15
#define PIN16 16
#define PIN17 17
#define PIN18 18
#define PIN19 19
#define PIN20 20
#define PIN21 21
#define PIN22 22
#define PIN23 23
#define PIN24 24
#define PIN25 25
#define PIN26 26
#define PIN27 27
#define PIN28 28
#define PIN29 29
#define PIN30 30
#define PIN31 31
int open_gpio(char *node, int per);
int gpio_conf(int fd, int bank, int pin, int dir);
int gpio_intr_conf(int fd, int bank, int pin, int dir);
int gpio_free(int fd, int bank, int pin);
int gpio_intr_free(int fd, int bank, int pin);
int set_gpio(int fd, int bank, int pin, int value);
int close_gpio(int fd);
unsigned int read_interupt(int fd, int bank, int pin);

#endif /* OPEN_BOARD_H_ */
```



1.5.7. Modify the Library Path

These above functions are in the shared library so we are going to add our shared library to this project.

you can download led and user button library from Phytec FTP using this link

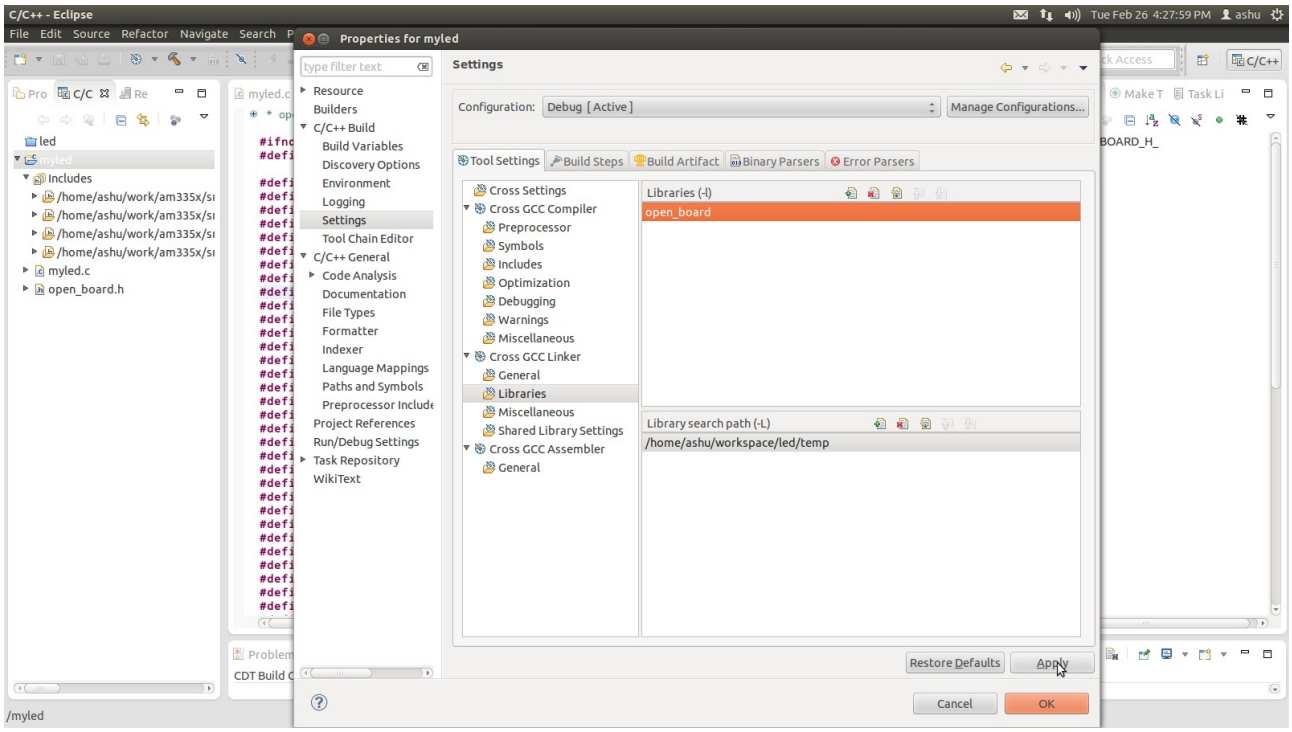
ftp://ftp.phytec.de/pub/Products/India/OpenBoard-AM335x/Linux/latest/src/apps/apps.tar.gz

untar it and go to the led app directory where you find the library file, this is your library path.

Right-click the led project and chose Properties

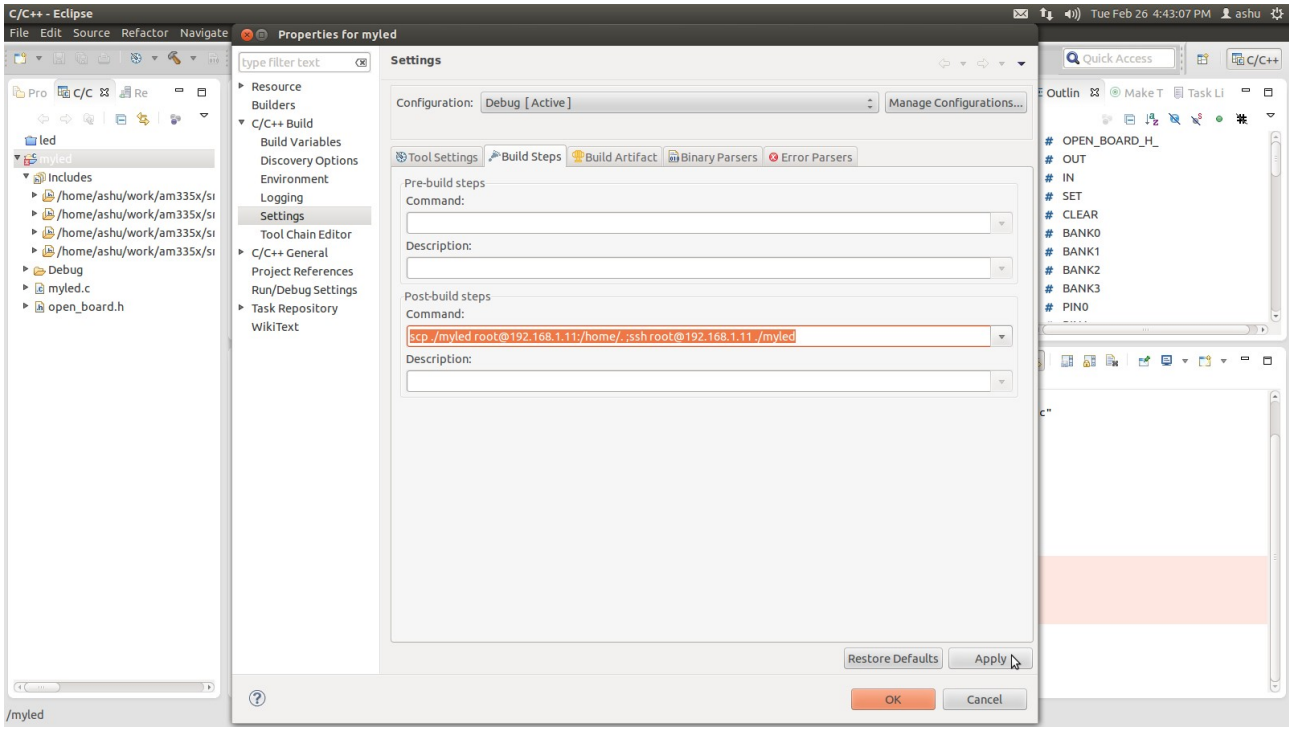
The Properties dialog appears.

Select C/C++ Build
----> Setting
-------> tools setting
-----------> Cross GCC Linker
---------------> Libraries
There add new Libraries (-l) as open_board
and Library search path (-L) as <location of your library >



1.5.8. Modify Post build steps

Select the Build Steps tab



Enter the following command in the Post-build steps Command input field:

scp ./myled root@192.168.1.11:/home/. ;ssh root@192.168.1.11 ./myled

Click Apply and then Click OK

Before building your application we need the drivers for led which is already in the Open Board root file in home directory:

/home/led/driver.ko

we have to insert this driver before using the application so for this you have a remote access of the open board.

1.5.9. Remote System Access using Eclipse

For Windows :

You have to set the address manually

connect ethernet cable

goto network connections

right click on "Local area connection"

-->properties

-->under general tab

double click on "Internet Protocol(TCP/IP)"

change the parameters

For Linux :

You have to set the address manually

connect ethernet cable

To see all the interfaces present on the open board

ifconfig -a

After this we got the particular interface then we have to configure it.

ifconfig eth0 192.168.1.12 up

where eth0 is the interface name.

Again check whether it is configured or not

ifconfig -a

Then we have to set the gateway

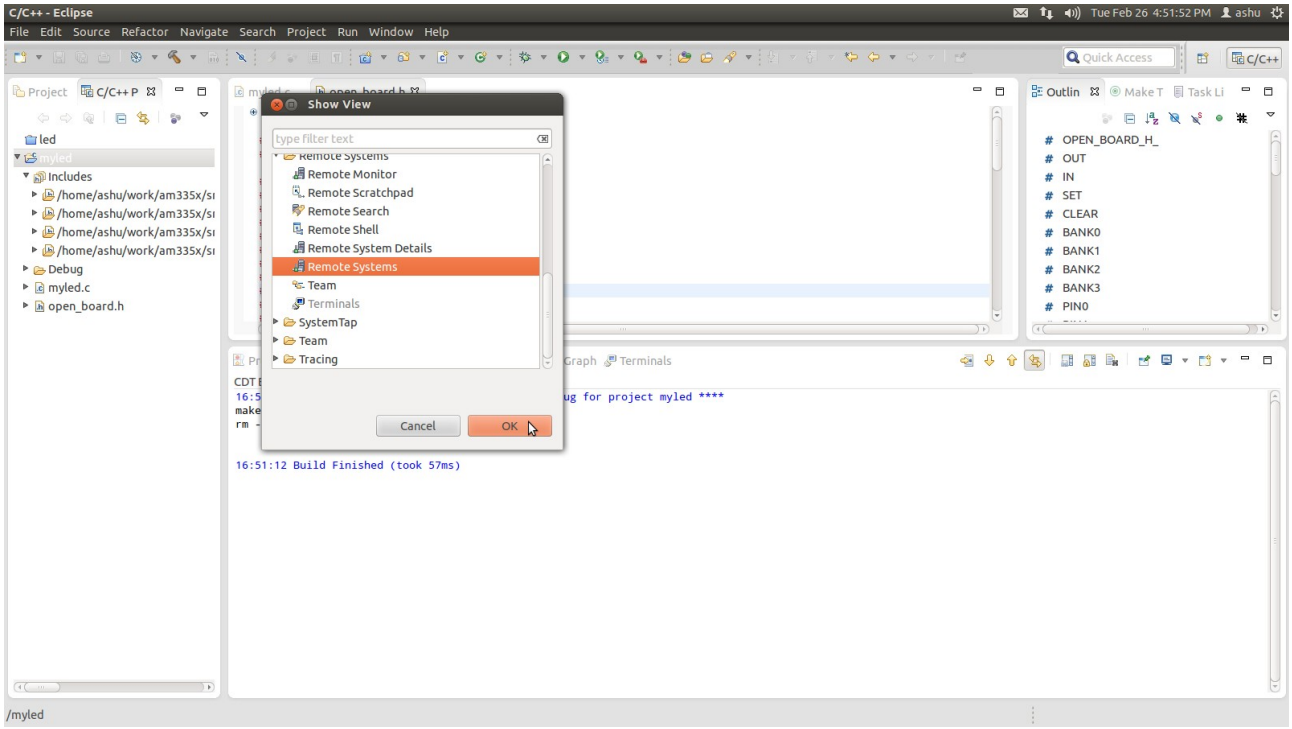
route add default gw 192.168.1.1

To see the change in the gateway.

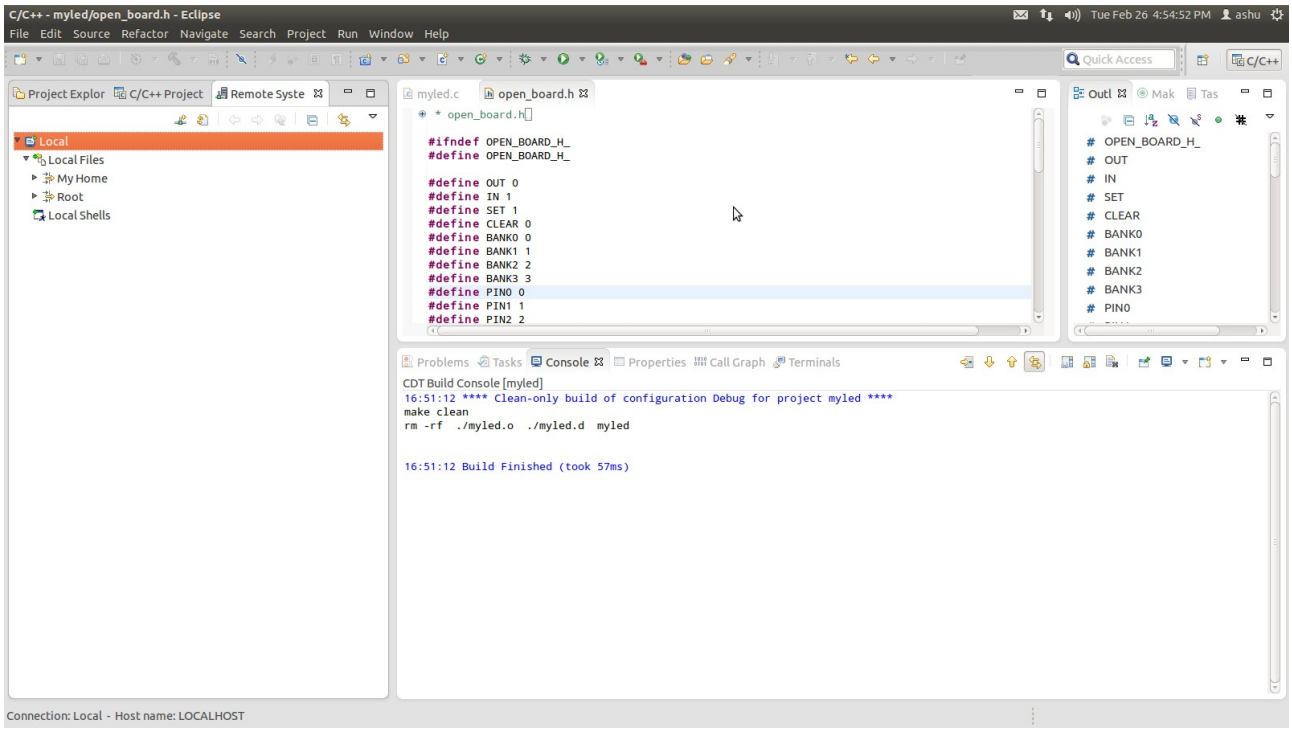
route

Left-click the Window tab.

Show view
----> others..
-------> Remote Systems
and ok



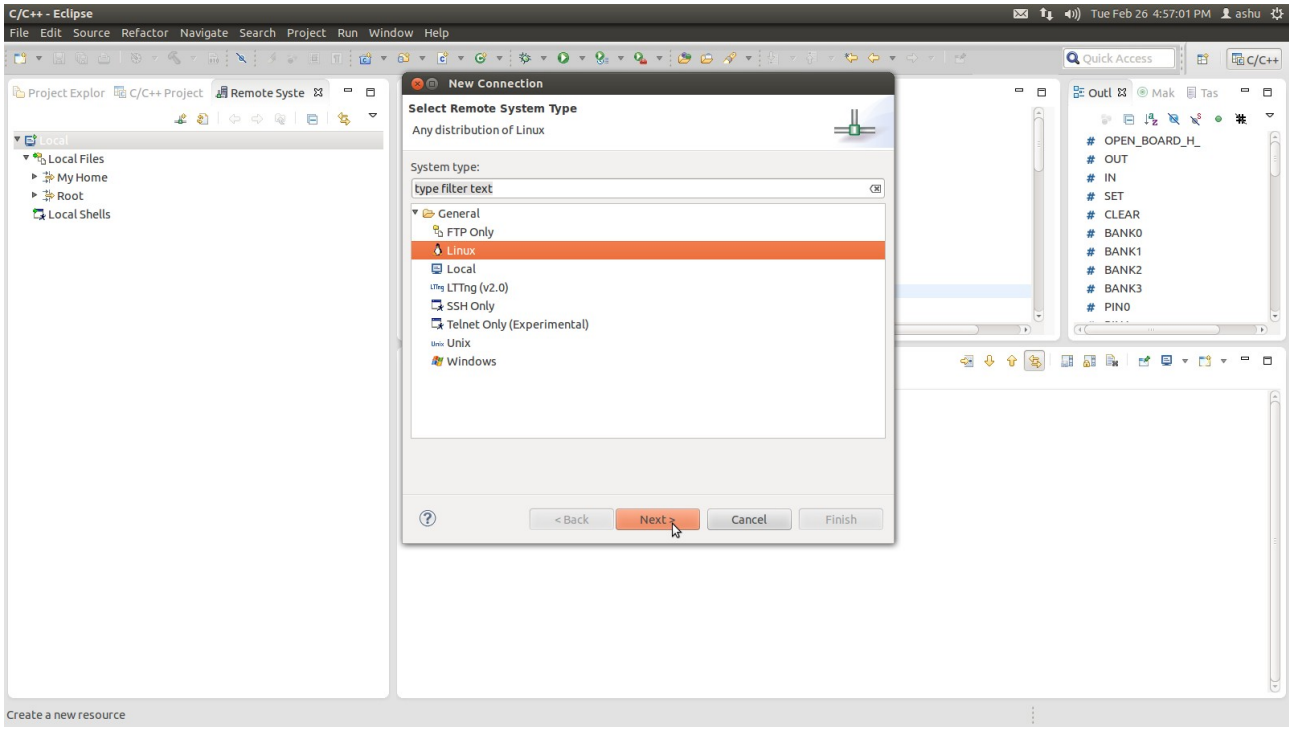
Now you are able to see the remote system page.



1.5.10. Create New Connection for Remote System login

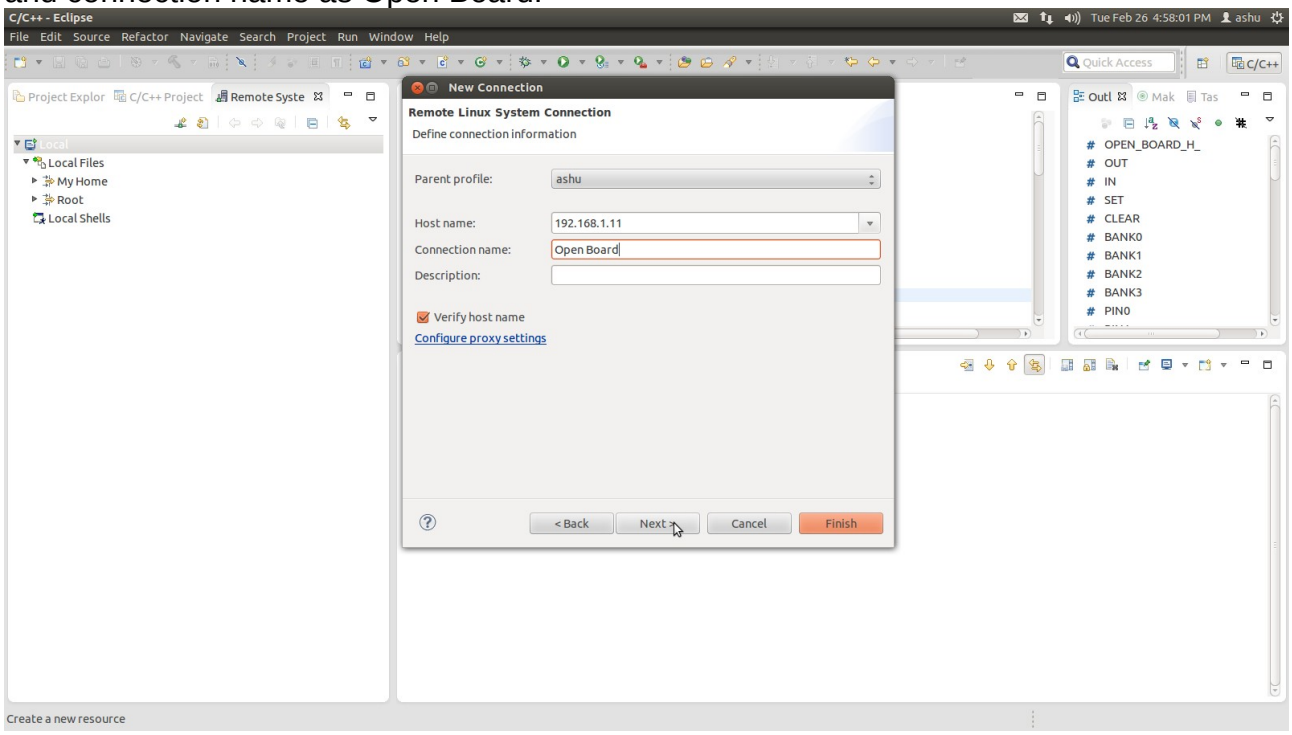
Right Click on Local select new connection

select linux

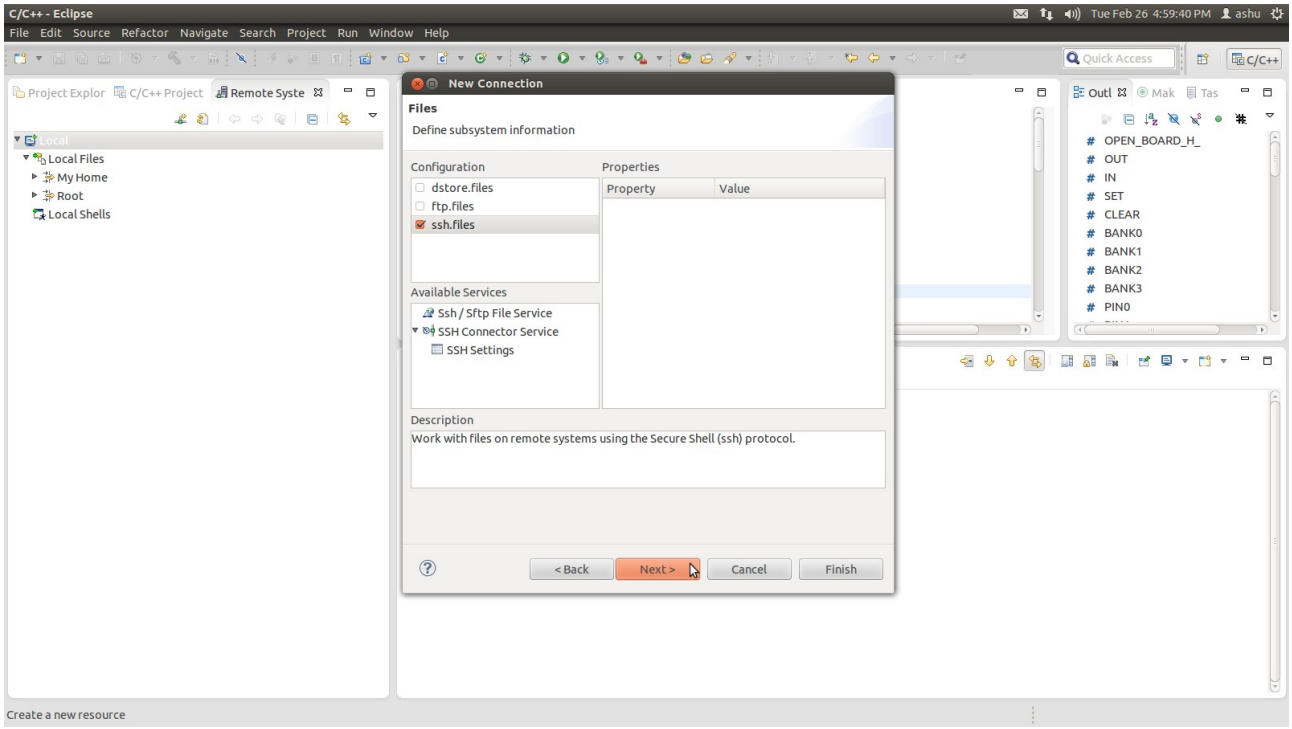


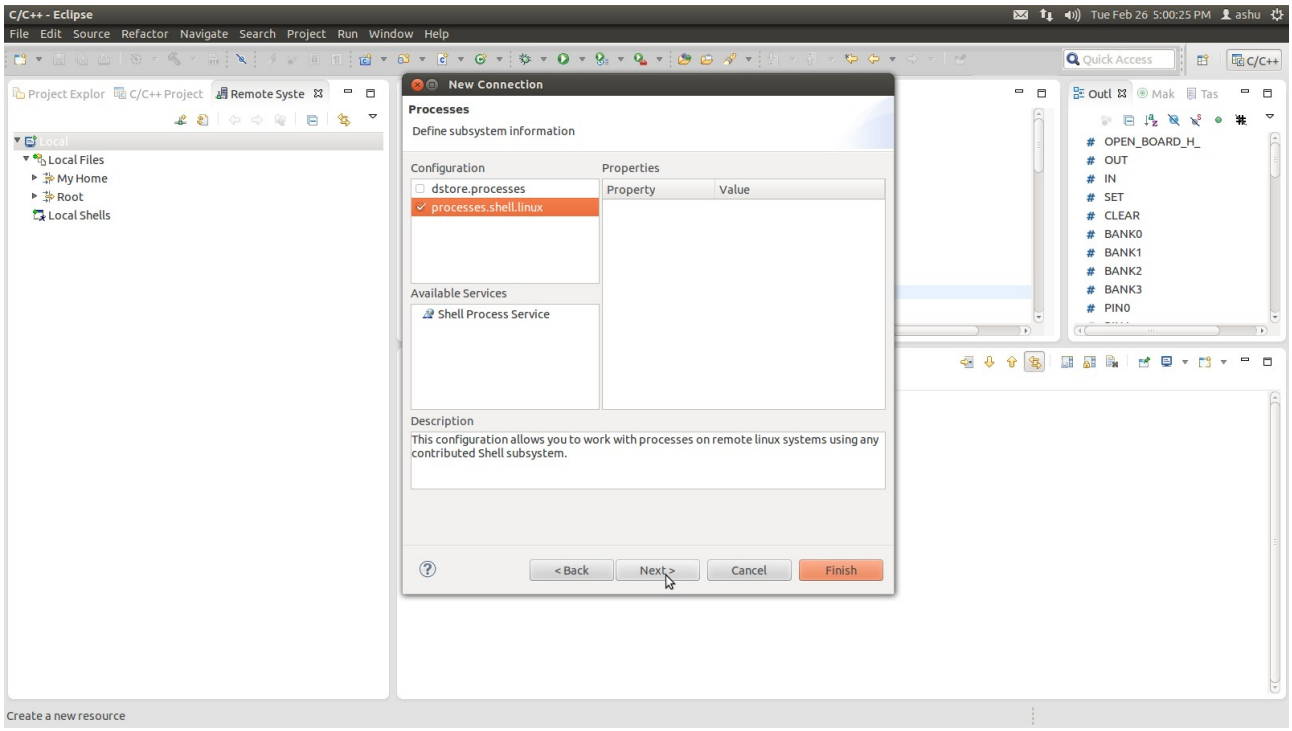
1.5.11. Set the Host Name and IP

Then write Host name as 192.168.1.11
and connection name as Open Board.

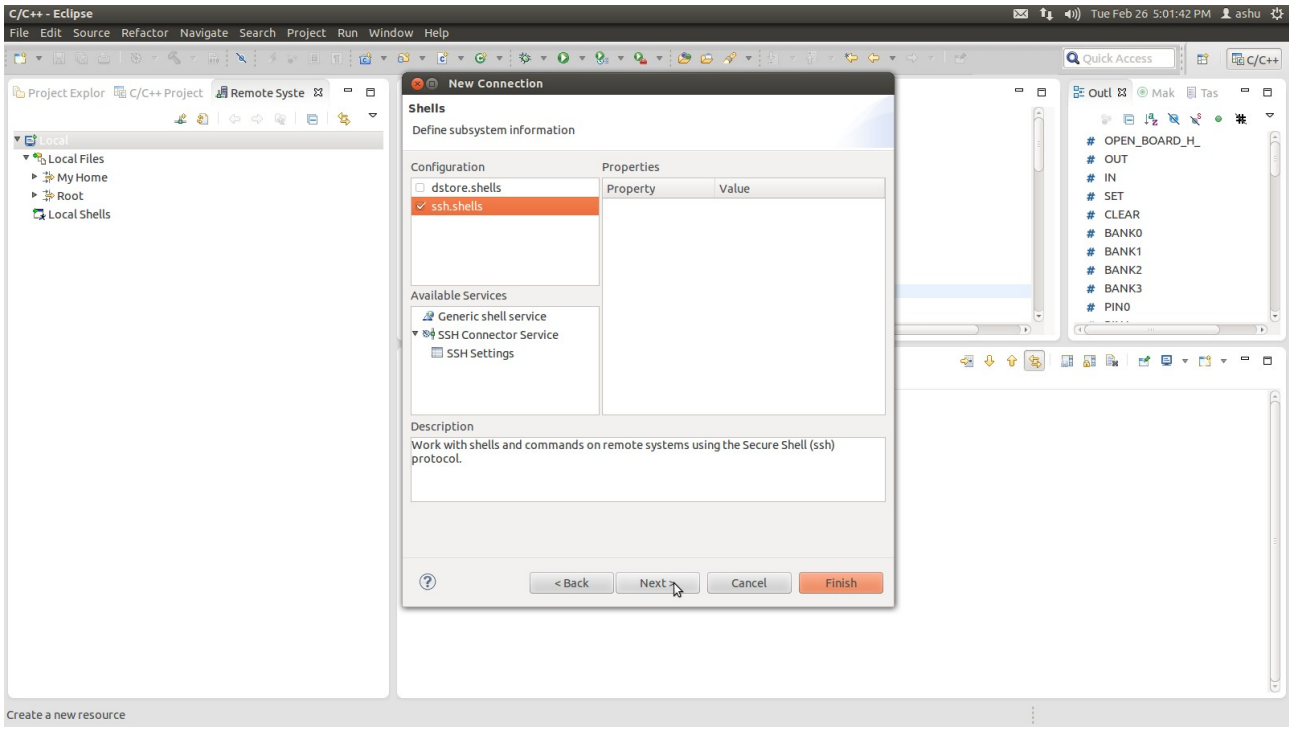


Select ssh.files

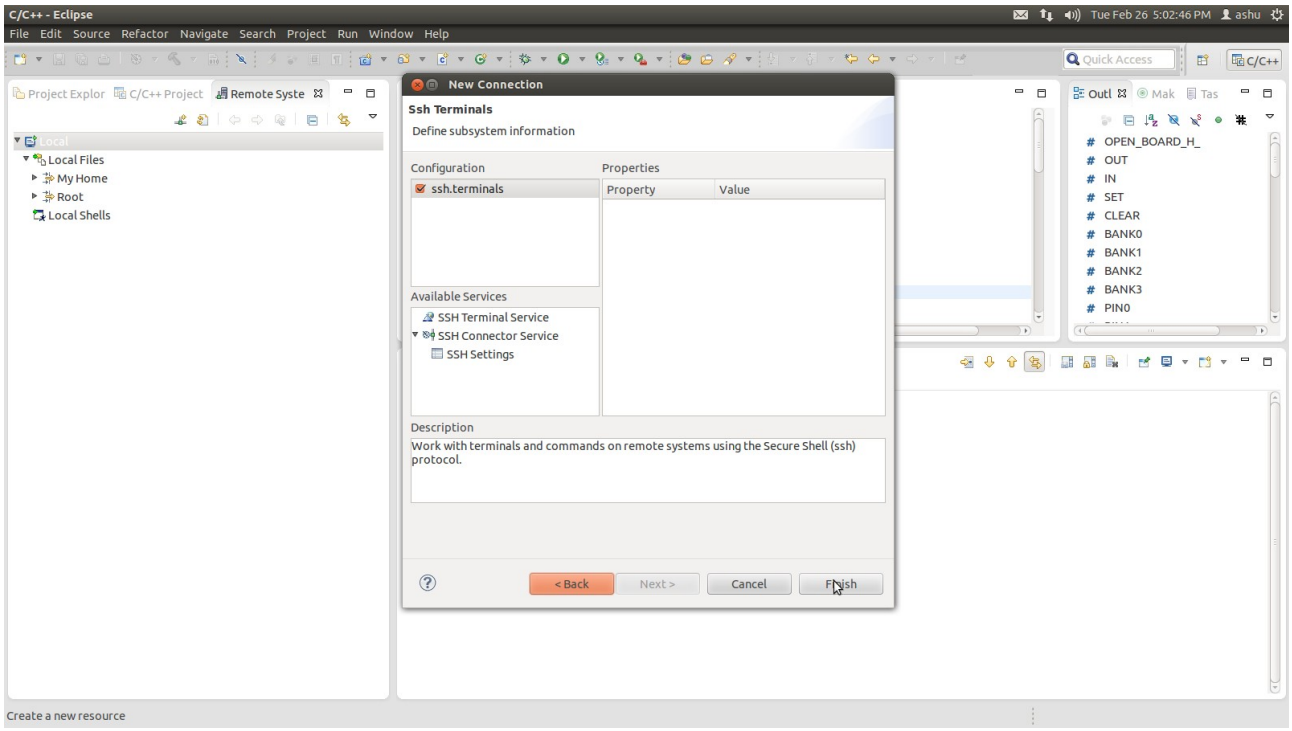




select processes.shell.linux and next

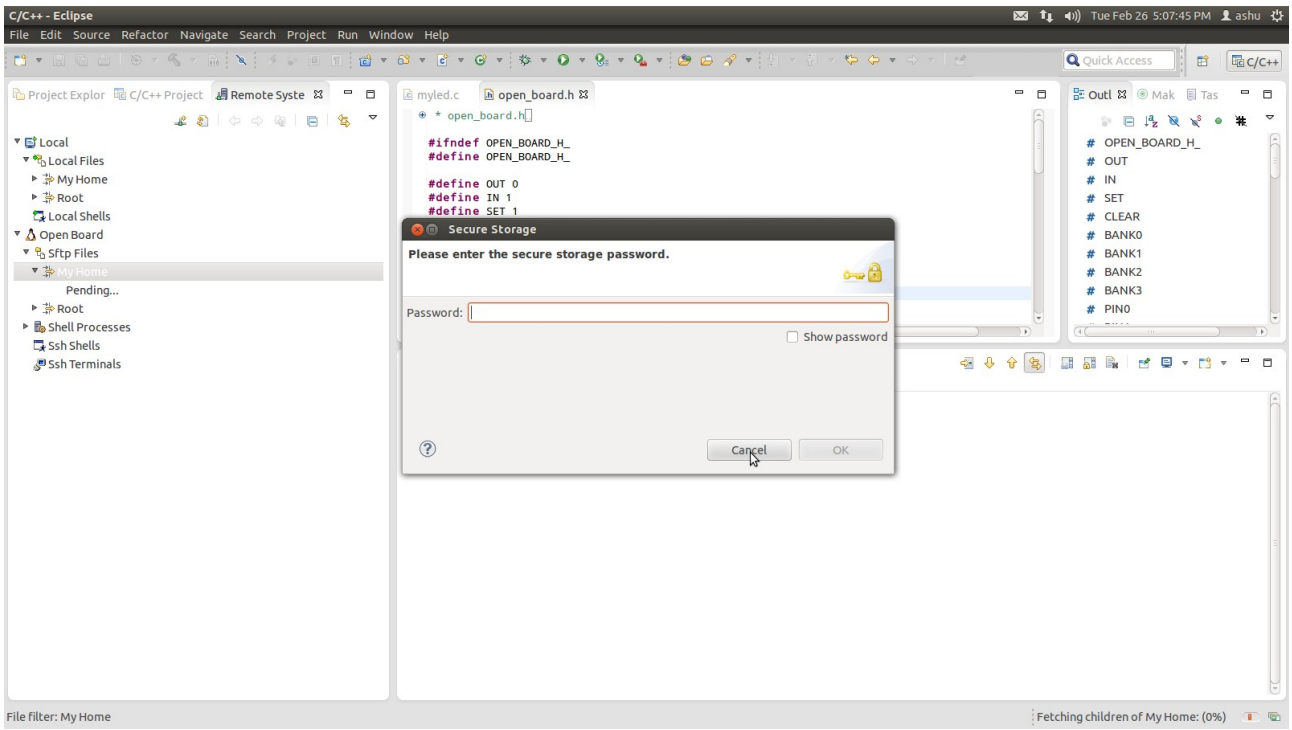


select ssh.shells and next

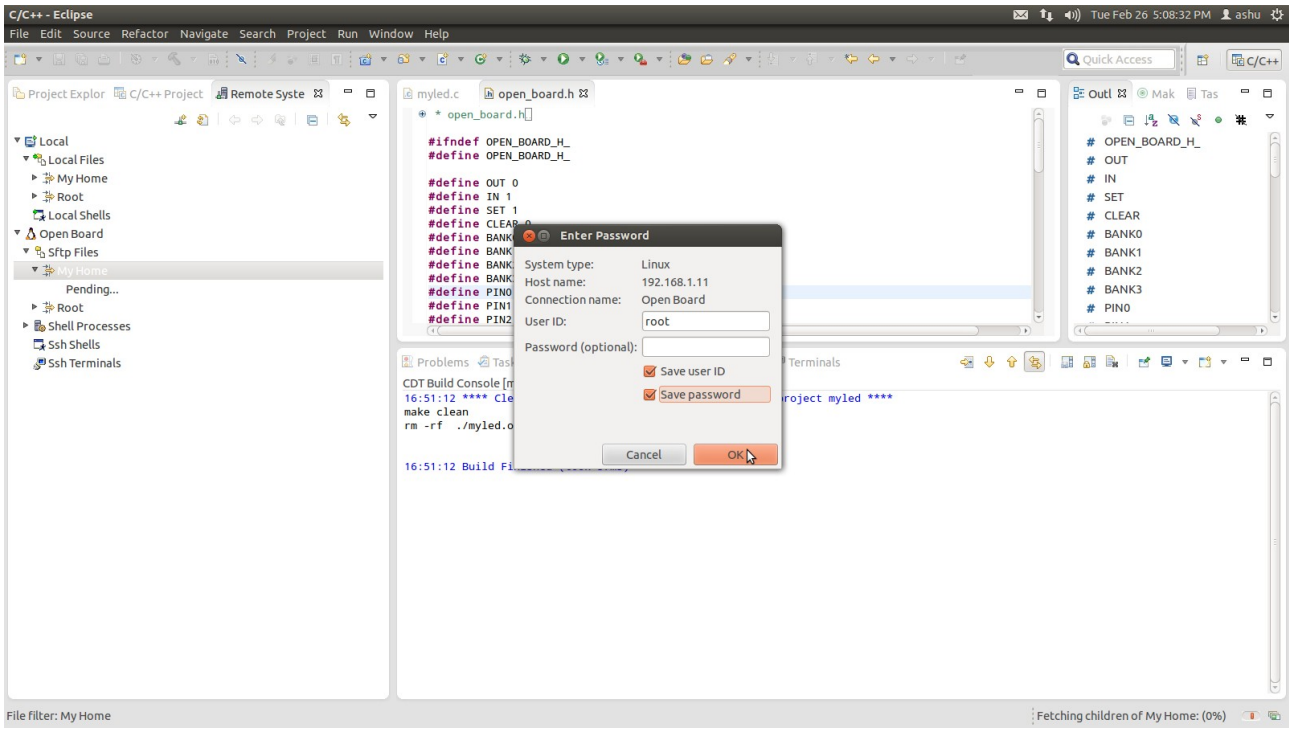


select ssh.terminals and finish

Now we successfully create the connection.
Click on the open board
My Home

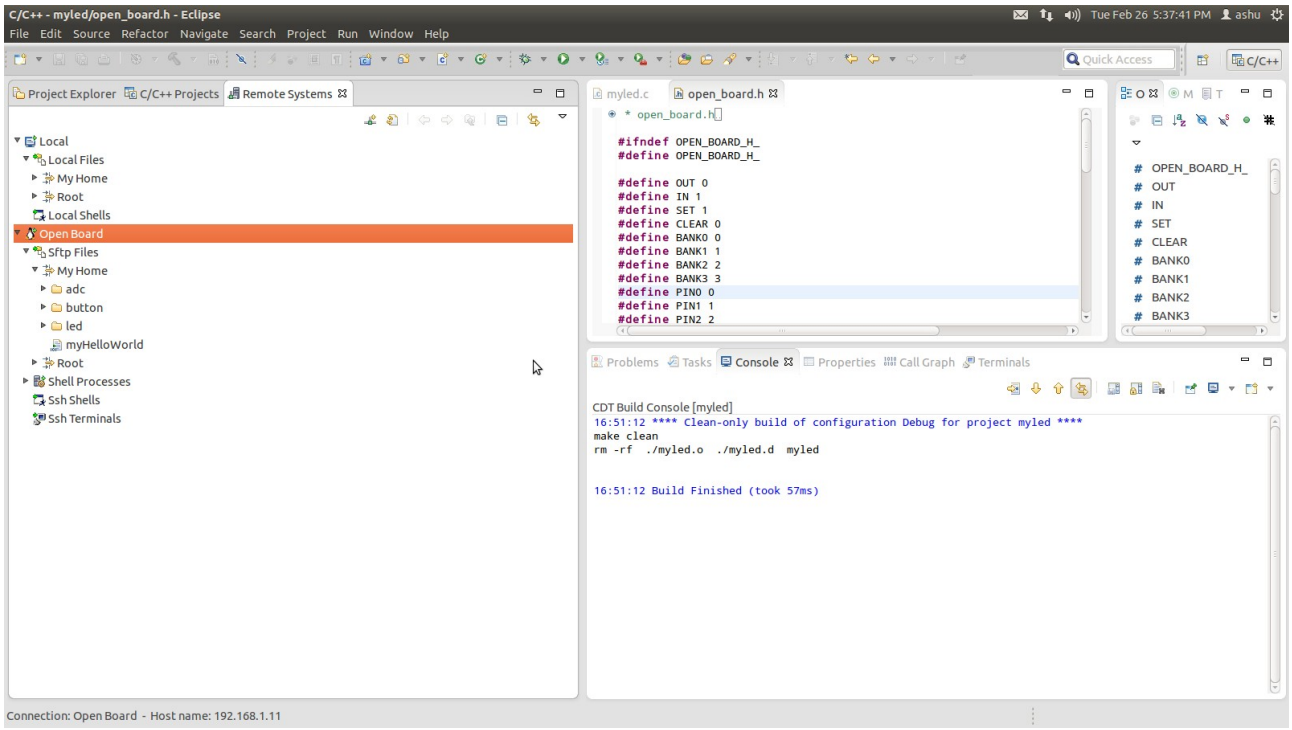


Then a secure Storage tab is opened just cancelled it.



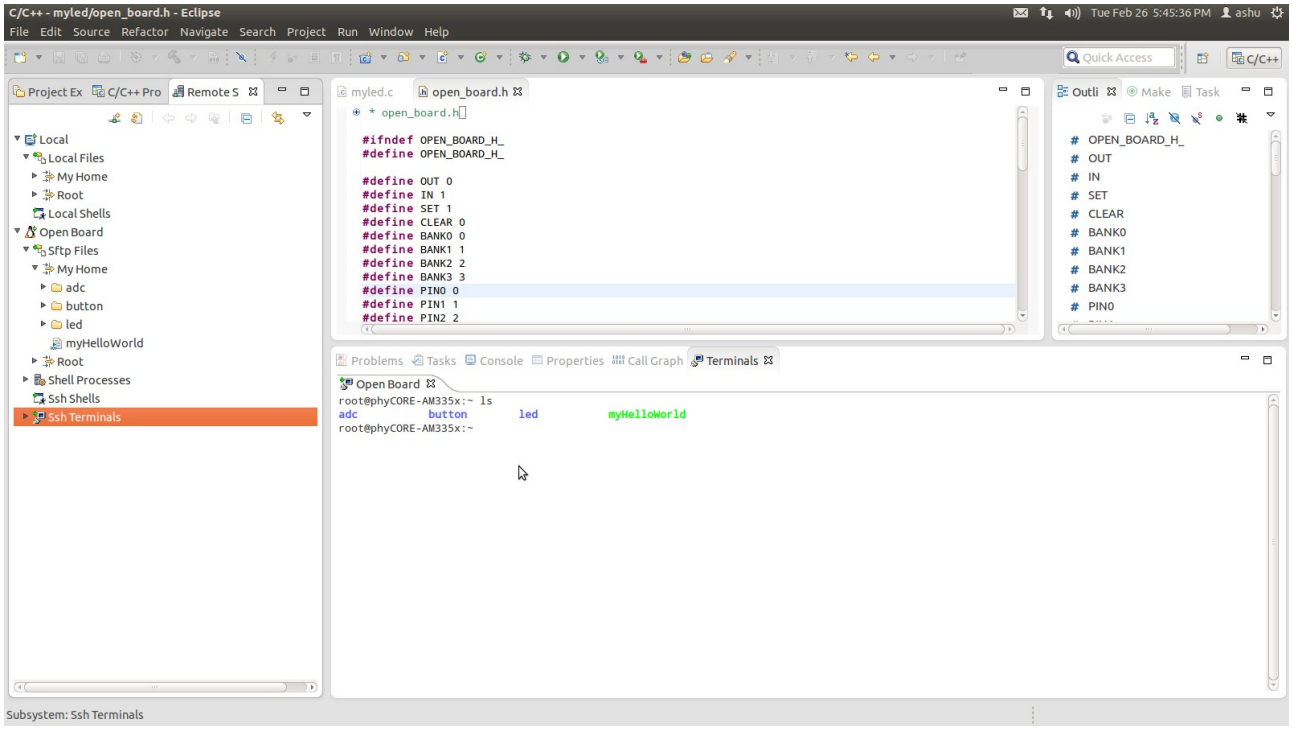
Type User ID as **root**
leave password as blank. Then press OK.

1.5.12. Launch the Remote Terminal



Now we can see all the contents of open board.

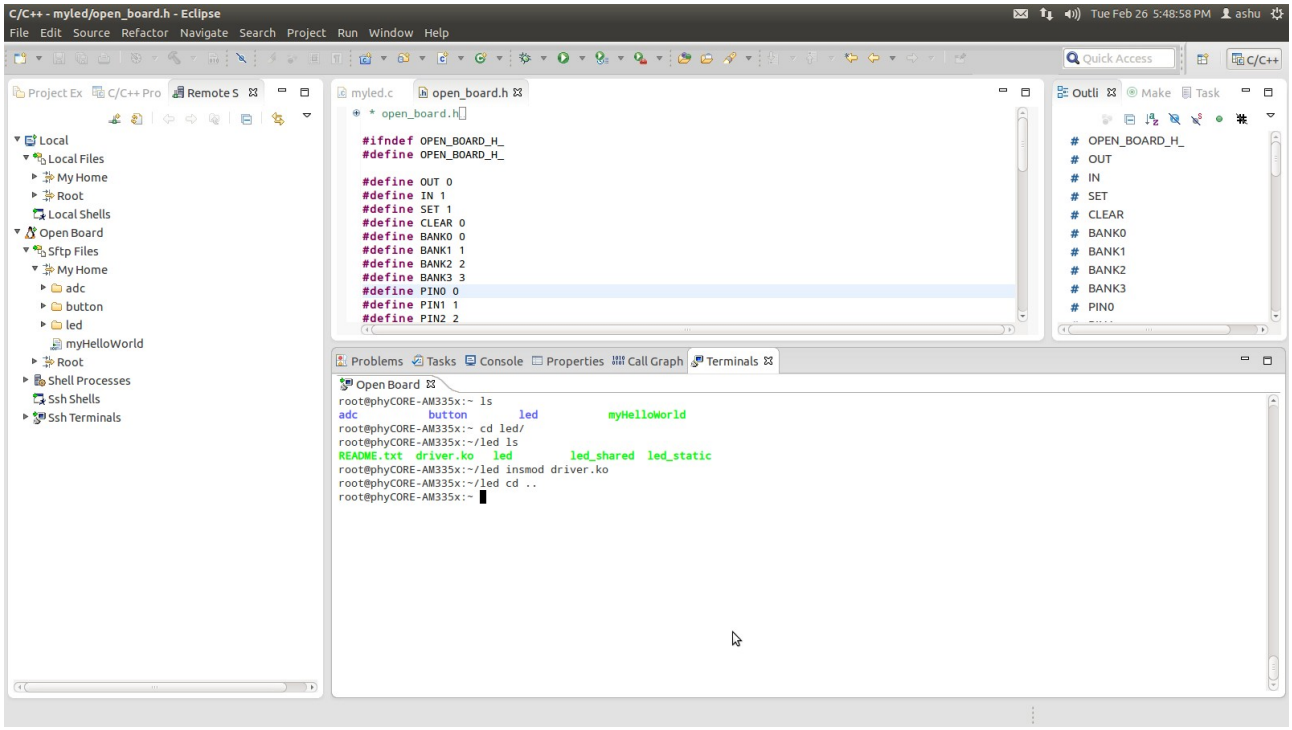
If you want to open the Open Board terminal just Right click on Ssh Terminal and click on the Launch Terminal.



1.5.13. Insert the driver using the Remote Terminal

Go to the led directory and insert the driver into the kernel space.

```
# cd led
# ls
# insmod driver.ko
```



Since the driver is inserted in the kernel space now we can run our application

1.5.13. Finally Build the project

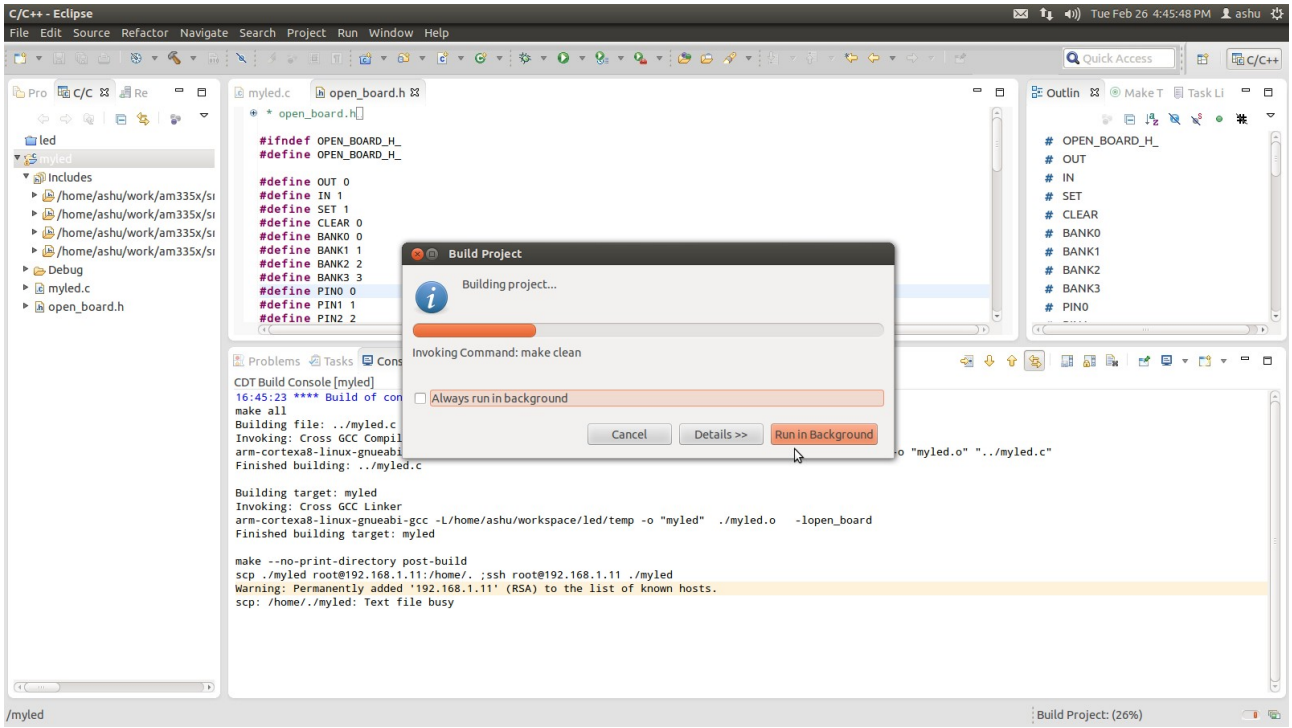
Select Project ► Build project from the menu bar

The project will be built.

Select the Console tab.

if no errors occur while building the project, you will see the following output:

It will ask for a confirmation the type **yes** to allow.



All 4 led's starts blinking.

1.6. Debugging an example project

In this chapter you will learn using the GNU debugger GDB on the host for remote debugging in conjunction with the GDB server on the target. GDB is the symbolic debugger of the GNU project and is arguably the most important debugging tool for any Linux system.
First you will start the GDB server on the target. Then you will configure the Eclipse platform and start the GNU debugger out of Eclipse using the Debug view.
The CDT extends the standard Eclipse Debug view with functions for debugging C/C++ code. The Debug view allows you to manage the debugging and running of a program in the workbench. Using the Debug view you will be able to set breakpoints/watchpoints in the code and trace variables and registers. The Debug view displays the stack frame for the threads of each target you are debugging. Each thread in your program appears as a node in the tree, and the Debug view displays the process for each target you are running.
The GDB client is running on the host and is used to control the GDB server on the target, which in turn controls the application running on the target. GDB client and GDB server can communicate over a TCP/IP network connection as well as via a serial interface. In this Quickstart we will only describe debugging via TCP/IP.

1.6.1. Starting the GDB server on the target

In this passage you will learn how to start the GDB server on the target. The GDB server will be used to start and control the myHelloWorld program.
To debug a program with GDB, the program needs extended debugging symbols.
This has already been added while building the program.
Open Minicom
minicom -D /dev/ttyXX
Type root and press Enter

Start the GDB server:

gdbserver 192.168.1.11:10000 myled

You have started the GDB server on the target. The GDB server is now waiting for connections on TCP port 10000.

1.6.2. Configuring and starting the debugger in Eclipse

In this passage you will learn how to configure your project settings to use Eclipse with the GNU debugger. After the configuration of your project settings, the GNU debugger will start and connect to the GDB server on the target.

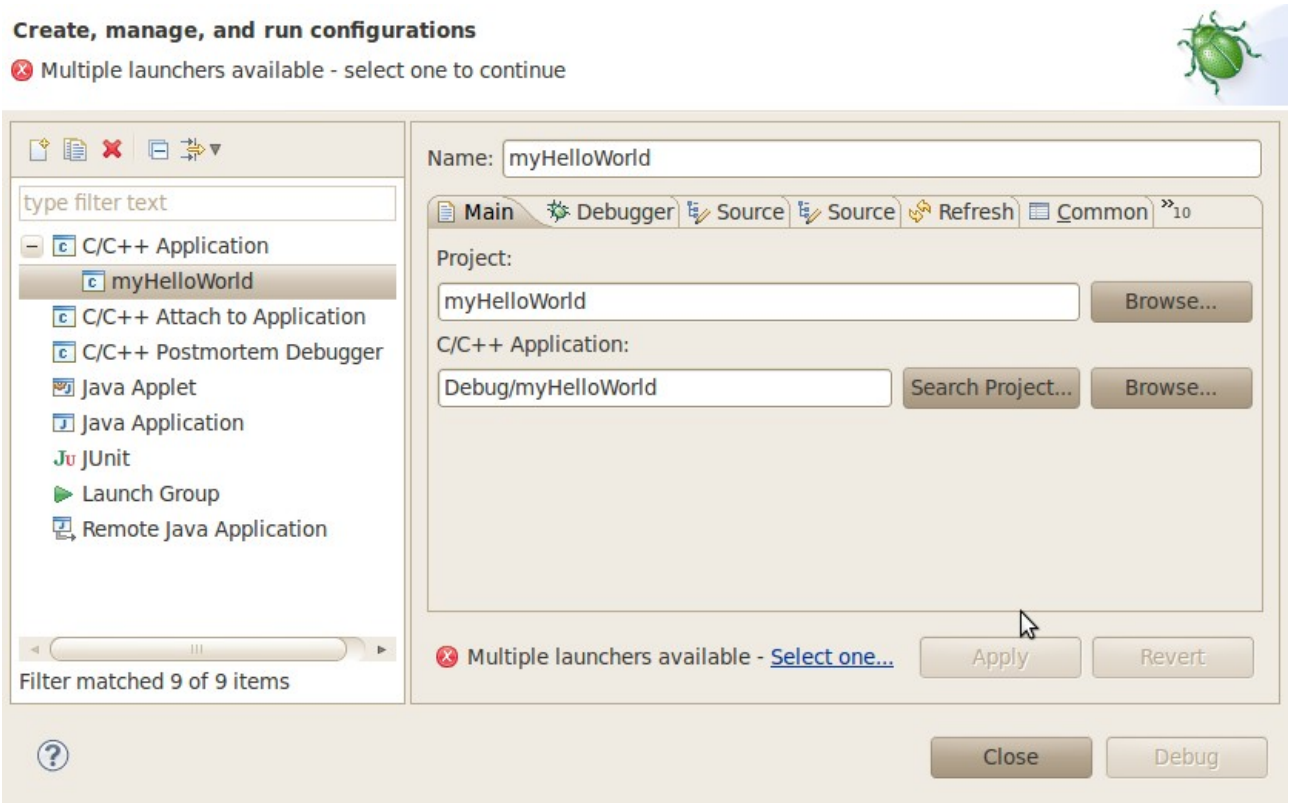
• Start Eclipse if the application is not started yet

• Right-click on the myHelloWorld project in the Navigator window

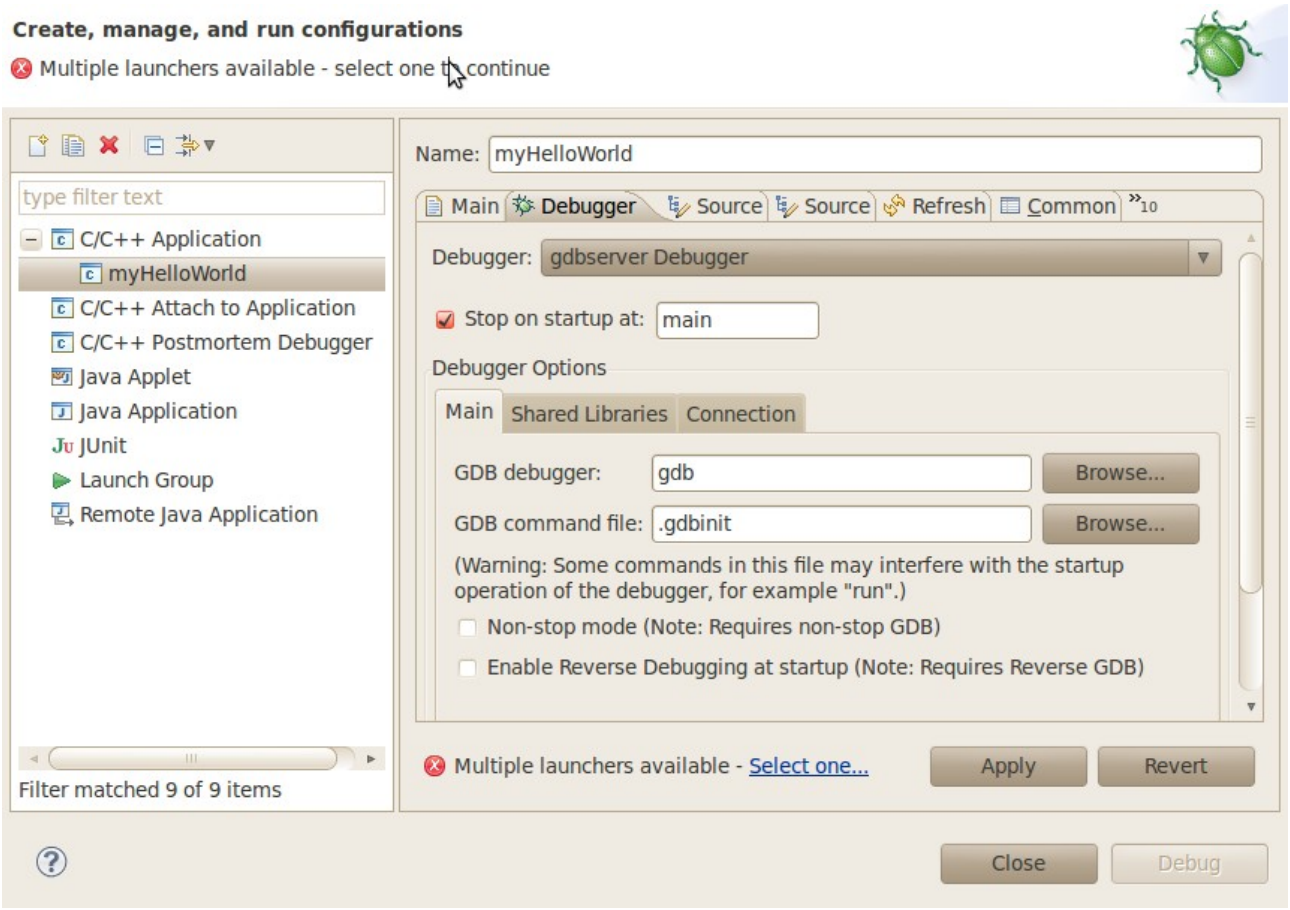
• Select Debug As ► Debug Configurations

A dialog to create, manage and run applications appears.

• Select myHelloWorld under C/C++ Application



Select the Debugger tab



Select gdbserver Debugger from the Debugger drop-down box

Click the Browse button right beside the GDB debugger input field.

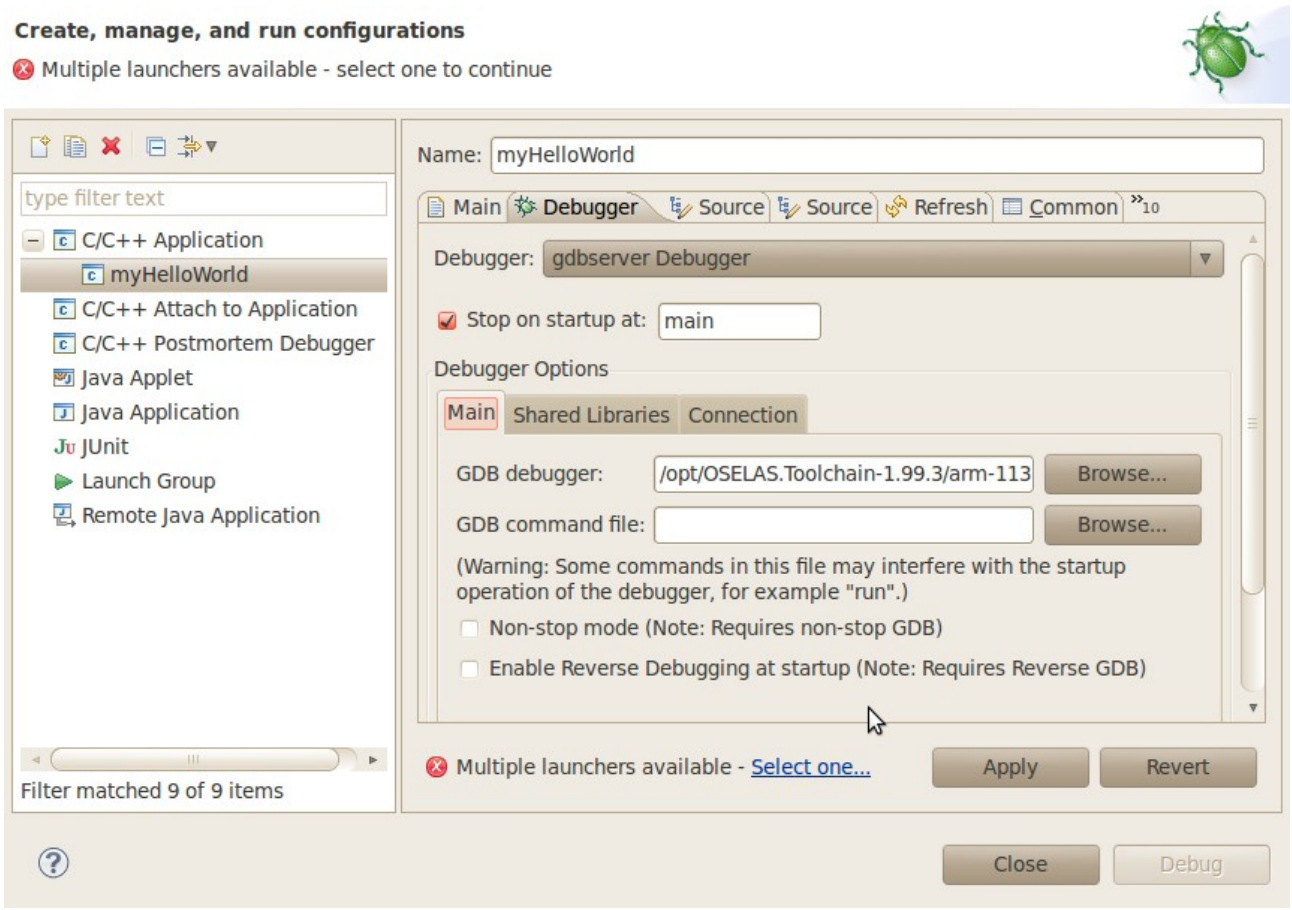
A new dialog opens to choose the GDB executable.

Click on File System

Navigate to the directory <Path of the Toolchain>/bin

Select the file arm-cortexa8-linux-gnueabi-gdb

Click OK

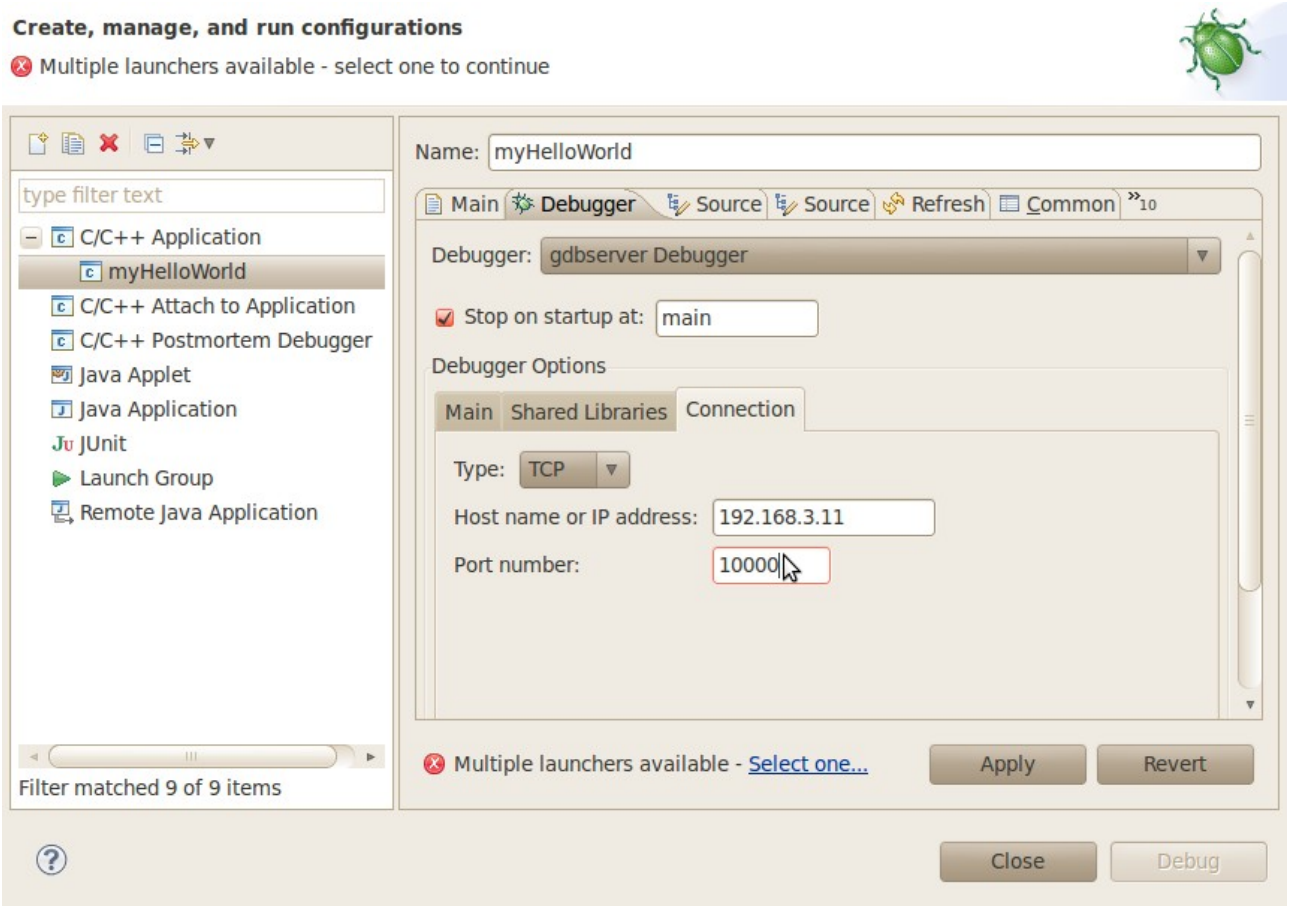


Keep the GDB command file field empty

Select the Connection tab and select TCP in the drop-down box

Enter 192.168.1.11 (the target's IP address) in the Host name input field.

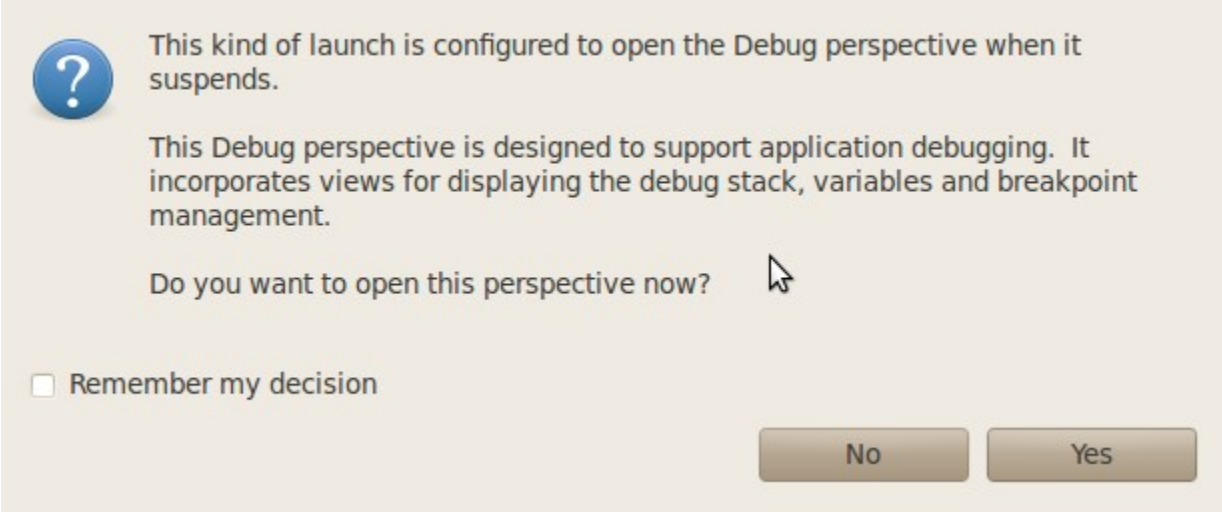
The host's GDB will connect to this IP address to communicate with the target's GDB server



Click Apply

Click Debug

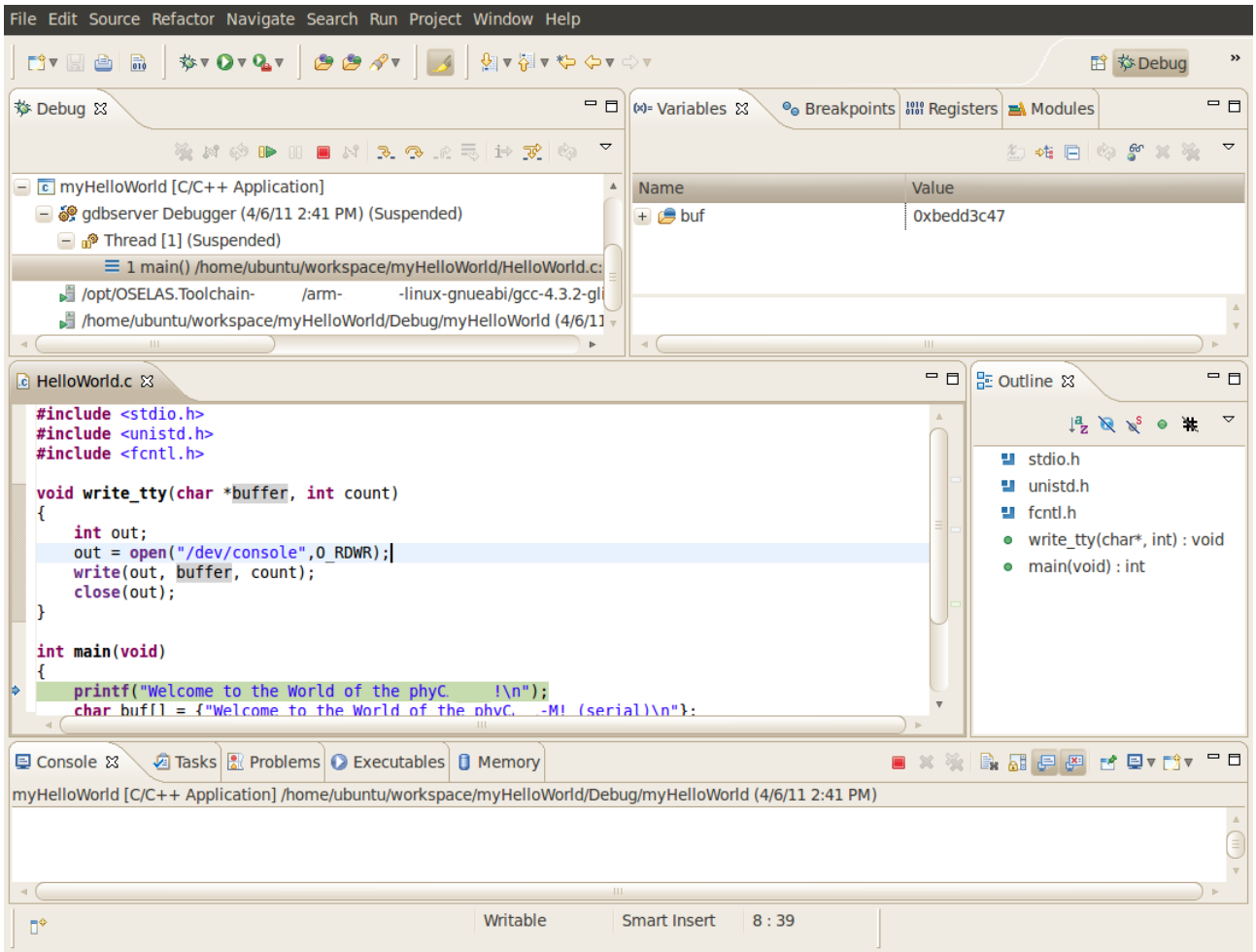
A new dialog appears.



Select Yes to switch to the Debug perspective

The debug perspective opens and the debugger stops automatically at the first line.

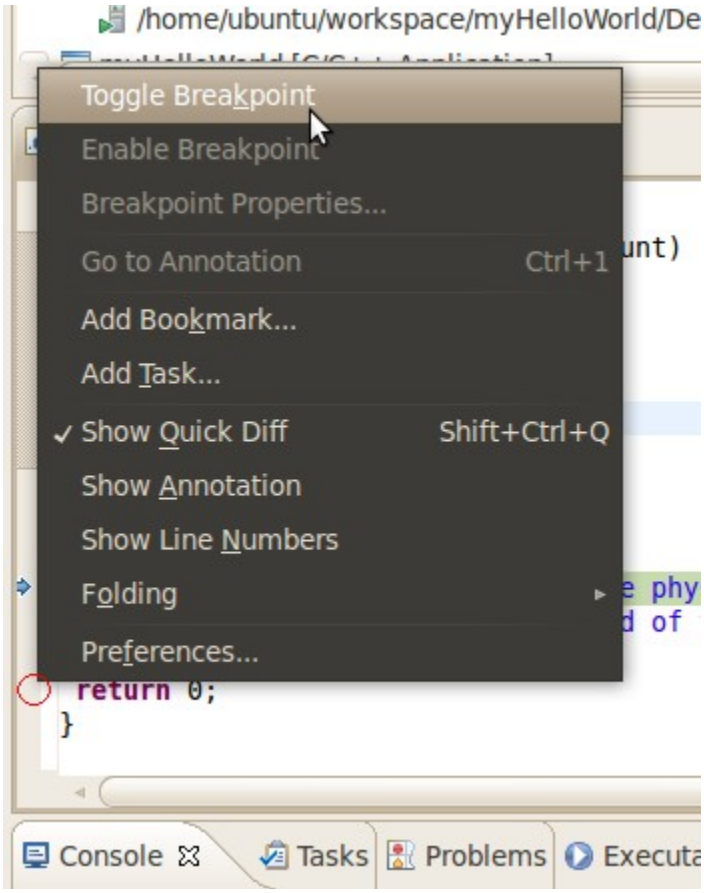
The host's GDB is now connected to the GDB server on the target.



You have configured your project for remote debugging. You have started the GNU debugger in Eclipse and connected the host's GDB with the target's GDB server. You can now start to debug the project.

1.6.3. Setting a Breakpoint

Now you will set a breakpoint in your program. The breakpoint will be set on the last line of the function main(). If you resume the application, the debugger will stop on this line.



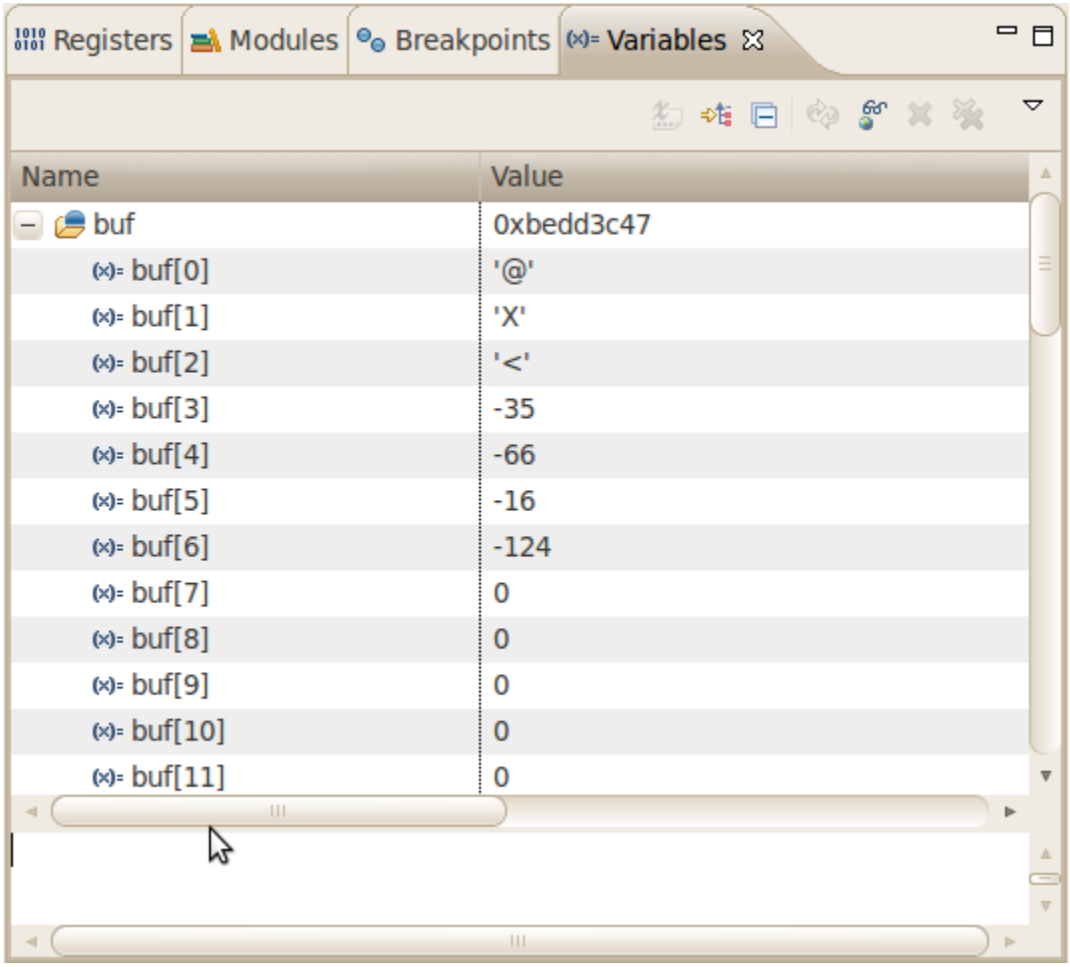
Select the last line in main()

Right-click into the small grey border on the left-hand side and select Toggle Breakpoint to set a new breakpoint

1.6.4. Stepping and Watching Variable Contents

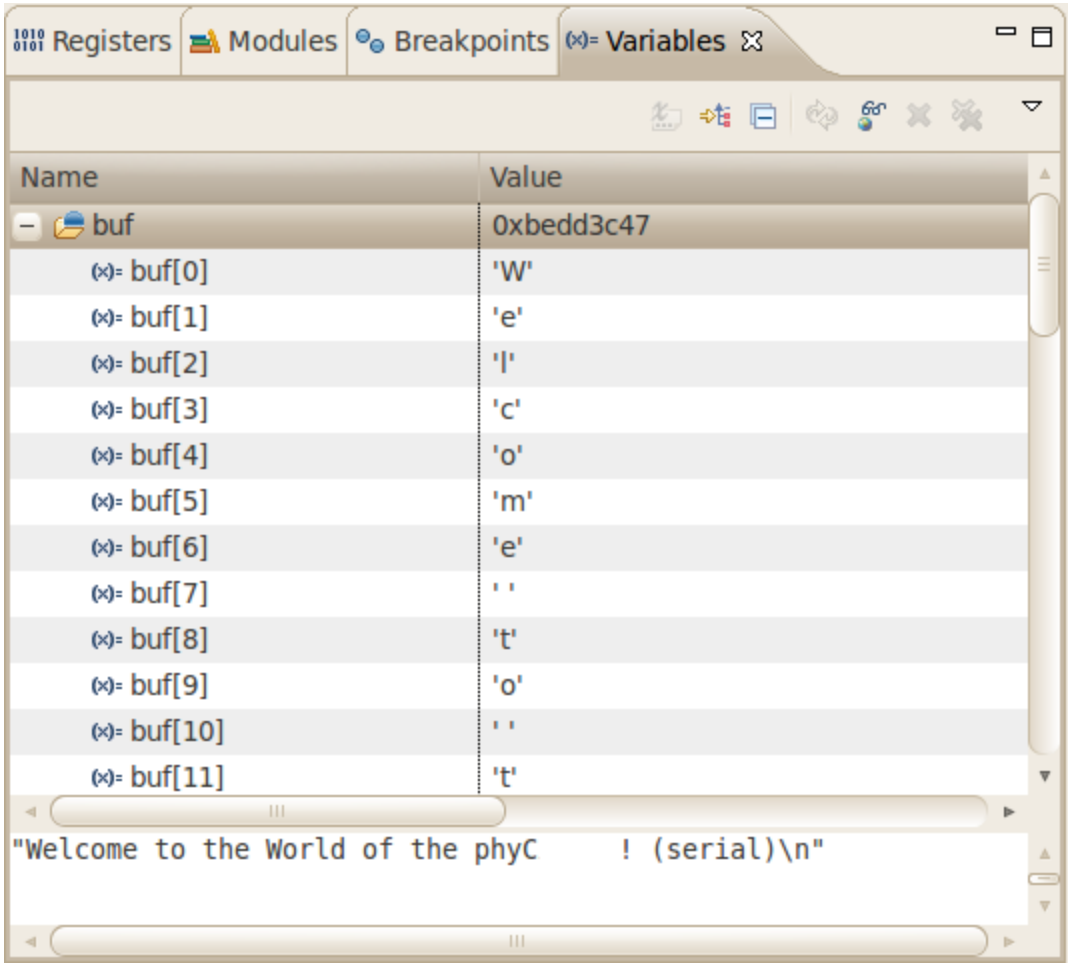
In this part you will step through the example project with the debugger. You will also learn how to check the content of a variable.

Expand buf in the Variables window



Click the Step Over button in the Debug window to step to the next line

You will see the content of the buf variable in the Variables window.

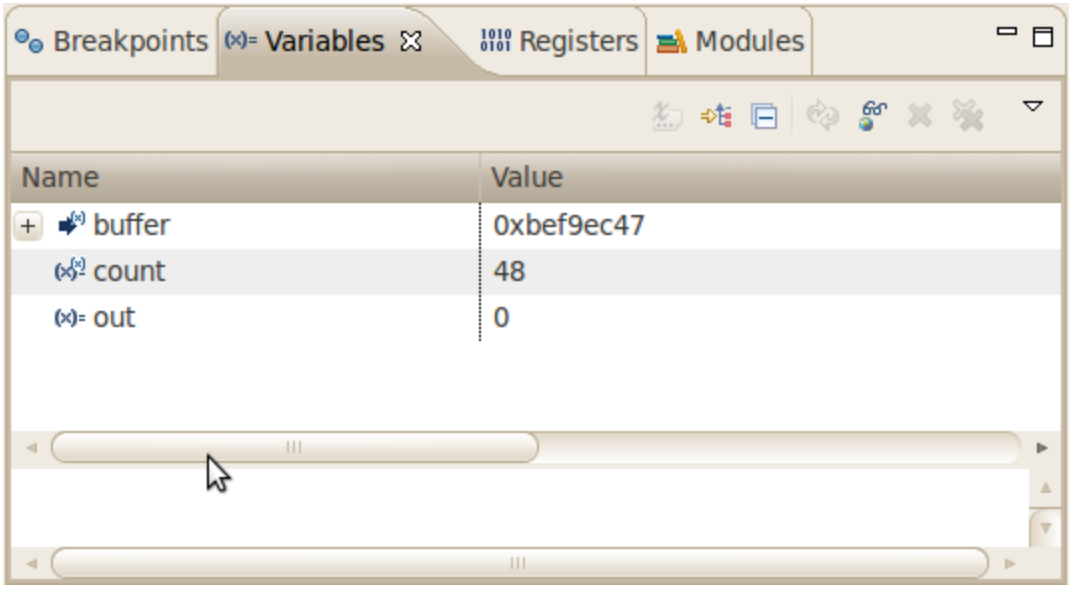


Click on the variable buf

Then click the button Step into to enter the function write_tty()

The debugger stops in write_tty().

You will see the following variable window:



Click on the variable buffer

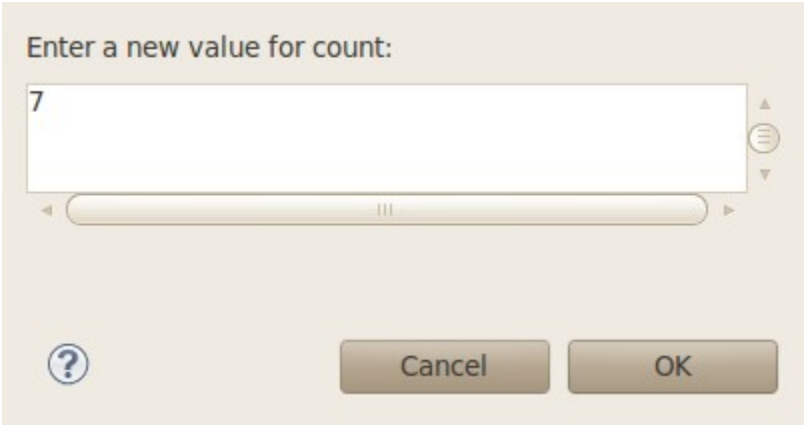
You will probably see a different address on the buffer pointer. Remember what address is shown in your case; you will need this address later.

1.6.5. Stepping and Watching Variable Contents

In this section you will change the value of a variable. At the end of this part you will see the effect of this change.

Select the count variable in the Variables window

Right-click on count and select Change Value



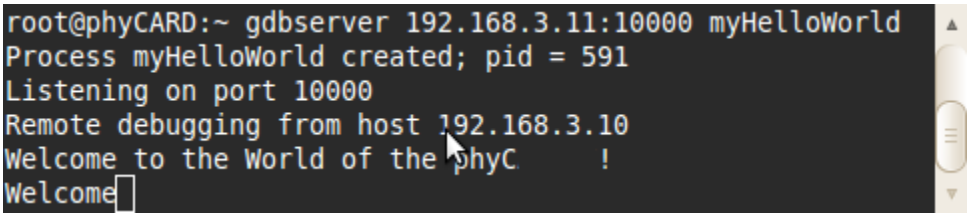
Change the value of count to 7 and click OK

Open Minicom if the application is not already opened

Go back to Eclipse

Click the Step Overbutton two times

Change to Minicom



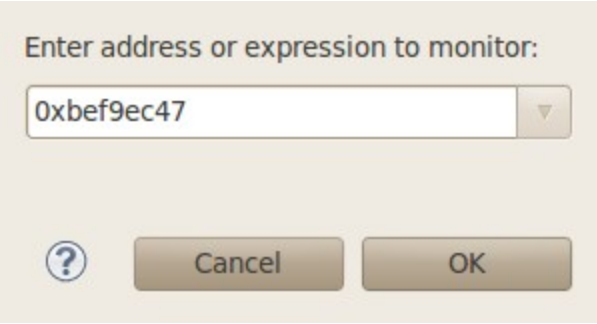
You will see the output Welcome in the Microcom window. This shows when changing the counter variable's value to 7 only the first seven characters of the buffer are output, instead of the whole sentence.

1.6.6. Using the Memory Monitor

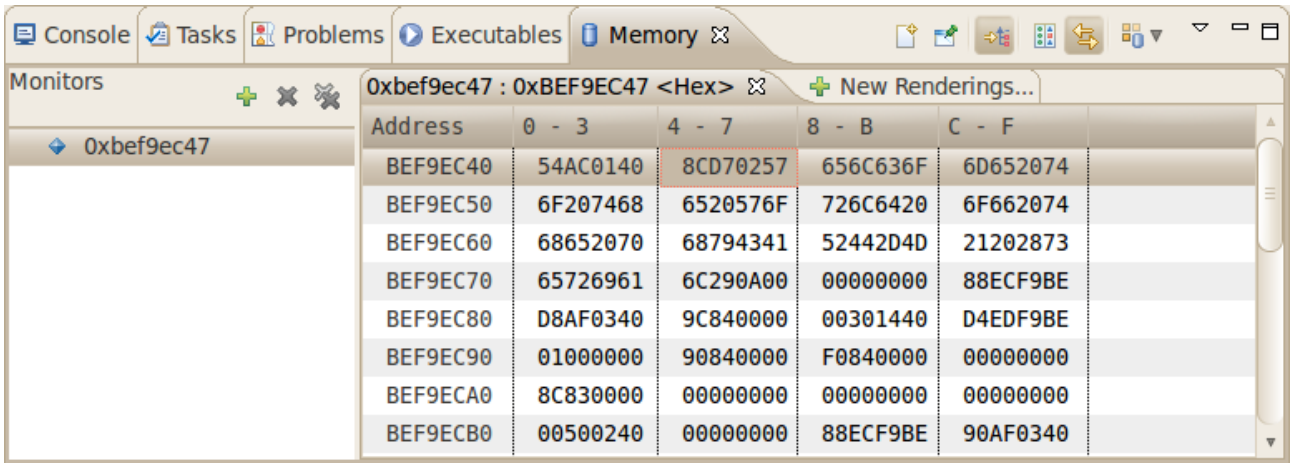
In the last section of this chapter you will use the memory monitor to control the content at a memory address.

Select the Memory tab

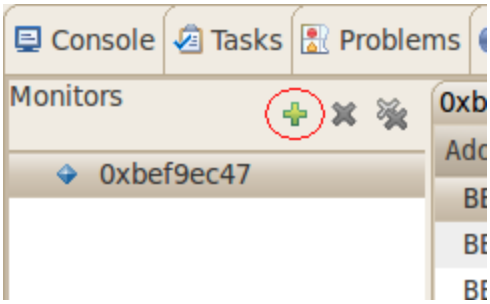
Click Add Memory Monitor



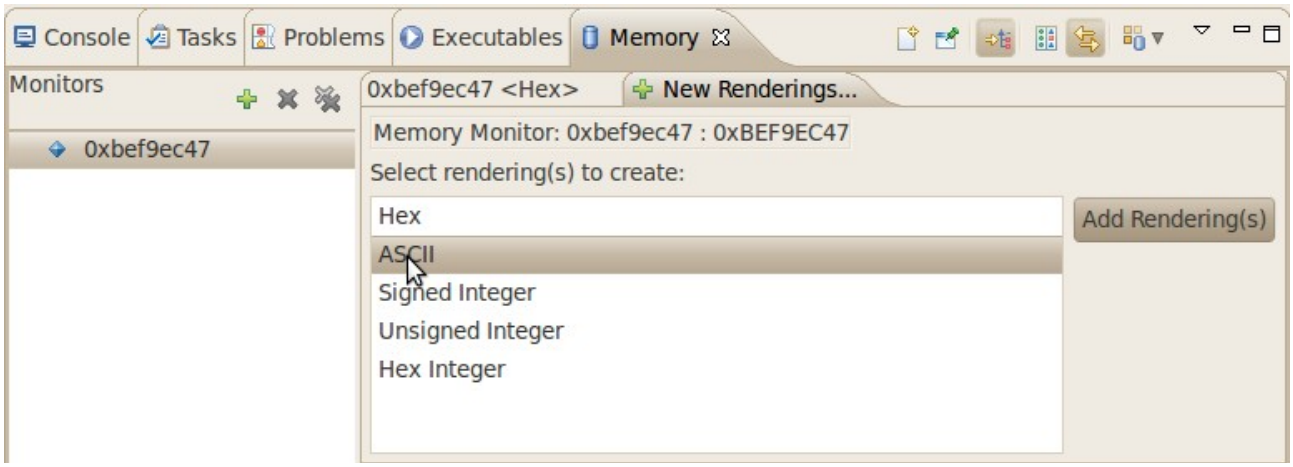
Enter the address of buffer and click OK. Remember that the variable's address might differ from your system.



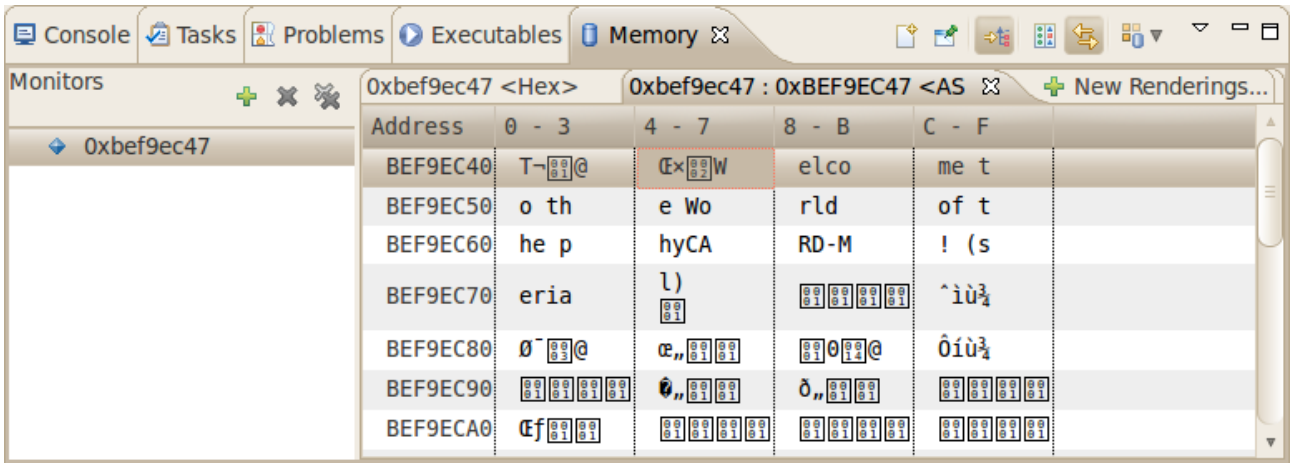
Change the window size



Click Add Rendering

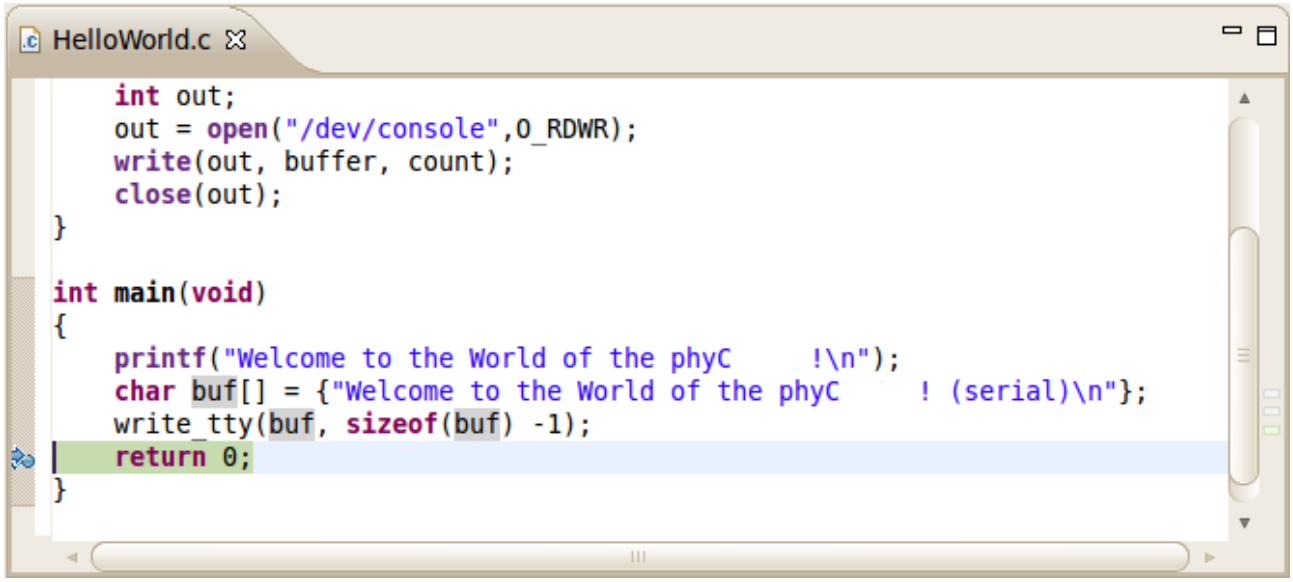


Select ASCII and click OK



You can see the contents of the variable buffer at the address 0xbee13ca7 (or whatever address is used on your system).

Now click the Resume button from the menu bar



The debugger stops at the breakpoint in the last line of main().

Click the Resume button to end the application

2. Application development using Console Terminal

You can download the demo application from the phytec ftp using below link.

After downloading just extract the file.

tar -xvf apps.tar.gz
cd final

Before doing any thing please read the RAEDME.txt
We have to resolve some dependence as given below:
ftp://ftp.phytec.de/pub/Products/India/OpenBoard-AM335x/Linux/latest/src/apps/apps.tar.gz

HOST SIDE:

Step-1: set the toolchain path edit the .bashrc file

vi ~/.bashrc

Step-2: Add the path of your toolchain at the last of this file and close the terminal

export PATH=$PATH:<the path of toolchain bin> (eg. go to toolchain/bin and type pwd)

Step-3: Export the path of your kernel source location (eg. go to kernel source and type pwd)

export K_SRC=<path of your kernel source>

Step-4: Run the Makefile

make

Finally you got all binary in related bin directory.

2.1. User LED'S

The Open Board-AM335x includes 4 LEDs listed as LED1, LED2, LED3, LED4 on the board.

2.1.1. Pre-Requists

User Led driver should be enabled in Kernel below is configuarion details. It is already enable in our linux uImage. If you want to do it manually then.

Type make menuconfig in linux kernel source

System Type --->
TI OMAP2/3/4 Specific Features --->
[*] AM335X based phyCORE AM335X
Handle AM3359 Openboard Muxing --->
[*]Led driver

2.1.2.Compiling User Led Application

2.1.2.1. Host Side

First go to the application directory and then to the led directory.

cd final/led

Export the kernel source directory path, architecture, cross compiler

export K_SRC=/home/phytec/work//linux-3.2 (This is the path of your kernel source)
export ARCH=arm
export CROSS_COMPILE=arm-cortexa8-linux-gnueabi-

After exporting these things just simply type make

make

It will create a bin directory where driver.ko (driver to be inserted) and led_shared, led_static (application to be executed) is created.

2.1.2.2. Target Side

Copy the contents of the bin directory into the target using SD-CARD, SSH, TFTP, USB.

SSH

scp driver.ko led_shared root@192.168.1.11:/home

Insert the driver.ko into the kernel space using insmod

insmod driver.ko

Then run the Application

./led_shared
It will blink the 4 leds 5 times.

2.2. User BUTTON'S

The Open Board-AM335x includes 4 BUTTON'S listed as BTN1, BTN2, BTN3, BTN4 on the board.

2.2.1. Pre-Requists

User Button driver should be enabled in Kernel below is configuarion details. It is already enable in our linux uImage. If you want to do it manually then.

Go to the kernel directory and type make menuconfig

```
# make menuconfig

System Type  --->
        TI OMAP2/3/4 Specific Features  --->
                [*] AM335X based phyCORE AM335X
                        Handle AM3359 Openboard Muxing  --->
                                [*] User Buttons driver
```

2.2.2. Compiling User Switch Application

2.2.2.1. Host Side

First go to the application directory and then to the button directory.

```
# cd final/button
```

Export the kernel source directory path, architecture, cross compiler

```
# export K_SRC=/home/phytec/work//linux-3.2 (This is the path of your kernel source)
# export ARCH=arm
# export CROSS_COMPILE=arm-cortexa8-linux-gnueabi-
```

After exporting these things just simply type make

```
# make
```

It will create a bin directory where driver.ko (driver to be inserted) and button_shared, button_static (application to be executed) is created.

2.2.2.2. Target Side

Copy the contents of the bin directory into the target using SD-CARD, TFTP, USB, etc.

Insert the driver.ko into the kernel space using insmod

```
# insmod driver.ko
```

Then run the Application

```
#  ./button_shared
```

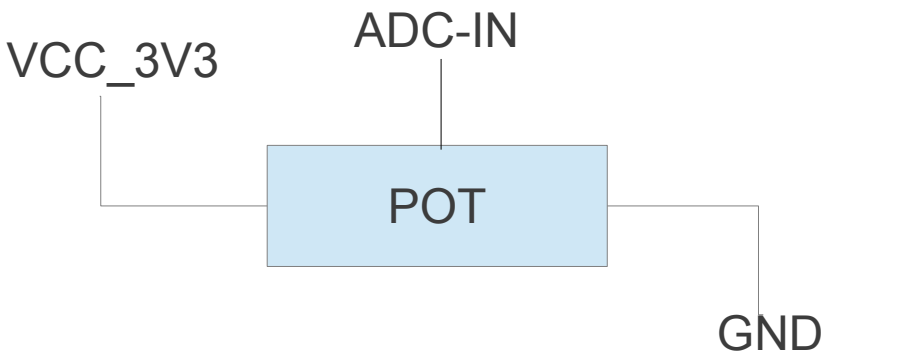
It will blink the leds depends on the buttons pressed.

2.3. ADC

The Open Board-AM335x provided additional ADC expansion for ADC Cannels there we have 8 channel. You find it out in the board at connector number X-50.

2.3.1. Pre-Requists

For testing the ADC you can use an POT which looks like



We can connect VCC_3V3 to X50.2
GND to X50.11 and ADC-IN to AIN0-AIN7

In the above figure we are using a 10 k Potentiometer

2.3.2. Compiling User ADC Application

2.3.2.1. Host Side

First go to the application directory and then to the adc directory.

cd final/adc

Export the kernel source directory path, architecture, cross compiler

export K_SRC=/home/phytec/work//linux-3.2 (This is the path of your kernel source)
export ARCH=arm
export CROSS_COMPILE=arm-cortexa8-linux-gnueabi-

After exporting these things just simply type make

make

It will create a bin directory where adc_am3359.ko (driver to be inserted) and adc (application to be executed) is created.

2.3.2.2. Target Side

Copy the contents of the bin directory into the target using SD-CARD, TFTP, USB, etc.

Step-1: Insert the driver file adc_am3359.ko

```
#  insmod adc_am3359.ko
```

Step-2: Run the adc user app to print the adc value

```
Syntax: adc <channel_num>      // channel_num 0 to 7

# ./adc 1
```

Step-3: Change the value of POT and observer the values printed on the screen.

PHYTEC

Get the dialog going ...
... and stay in touch

India

PHYTEC Embedded Ltd.
#9/1 1st Floor, 3rd Main
8th Block, Opp. Police Station
Kormangala, Bangalore-560095
Tel.: +91-8041307589
www.phytec.in

Germany

PHYTEC Messtechnik GmbH
Robert-Koch-Straße 39
D-55129 Mainz
Tel.: +49 6131 9221-32
Fax: +49 6131 9221-33
www.phytec.de
www.phytec.eu

America
PHYTEC America LLC
203 Parfitt Way SW, Suite G100
Bainbridge Island, WA 98110
Tel.: +1 206 780-9047
Fax: +1 206 780-9135
www.phytec.com

France
PHYTEC France SARL
17, place St. Etienne
F-72140 Sillé le Guillaume
Tel.: +33 2 43 29 22 33
Fax: +33 2 43 29 22 34
www.phytec.fr

.....We are looking forward to hearing from you!.....