

# Install and Use the Sourcery G++ Lite Toolchain with Eclipse in Windows for Cross Development with ARM Single Board Computers

William Wang

[ww2@andrew.cmu.edu](mailto:ww2@andrew.cmu.edu)

Class 2011  
School of Computer Science  
Carnegie Mellon University

## Table of Contents

1	Abstract.....	1
2	Downloading and installing the Sourcery G++ Lite toolchain .....	1
3	Downloading and installing the Eclipse IDE .....	2
3.1	Creating and Building an Example Project.....	5
3.2	Mounting a Windows folder from an ARM Single Board Computer.....	12
3.2.1	Creating a Shared Folder Under Windows Vista.....	12
3.2.2	Creating a Shared Folder Under Windows XP .....	14
3.2.3	Terminal Access to the ARM SBC .....	15
3.2.4	Mounting the shared folder .....	19
4	Appendix I: Remote debugging with gdbserver and Eclipse in Windows .....	23
5	Appendix II: Internet Access While on Multiple Networks in Windows .....	28

## 1 Abstract

This tutorial is for those who wish to participate in cross development for ARM single board computers using a Windows machine. Windows is a good choice for those unfamiliar with Linux or prefer a graphical IDE.

## 2 Downloading and installing the Sourcery G++ Lite toolchain

Download Sourcery G++ Lite ARM toolchain from <http://www.codesourcery.com/sgpp/lite/arm/portal/release858>. The Windows Installer is recommended for download. This tutorial will focus on the installer only.

Run the installation file and accept the agreement. Make changes to the default settings as necessary. If multiple users on a single Windows machine will be participating in development, it is recommended to change Modify PATH for current user to Modify PATH for all users. This is so that one can run the Sourcery G++ executables in any directory without having to specify a path. Figure 1 shows the screen which modifies this option.

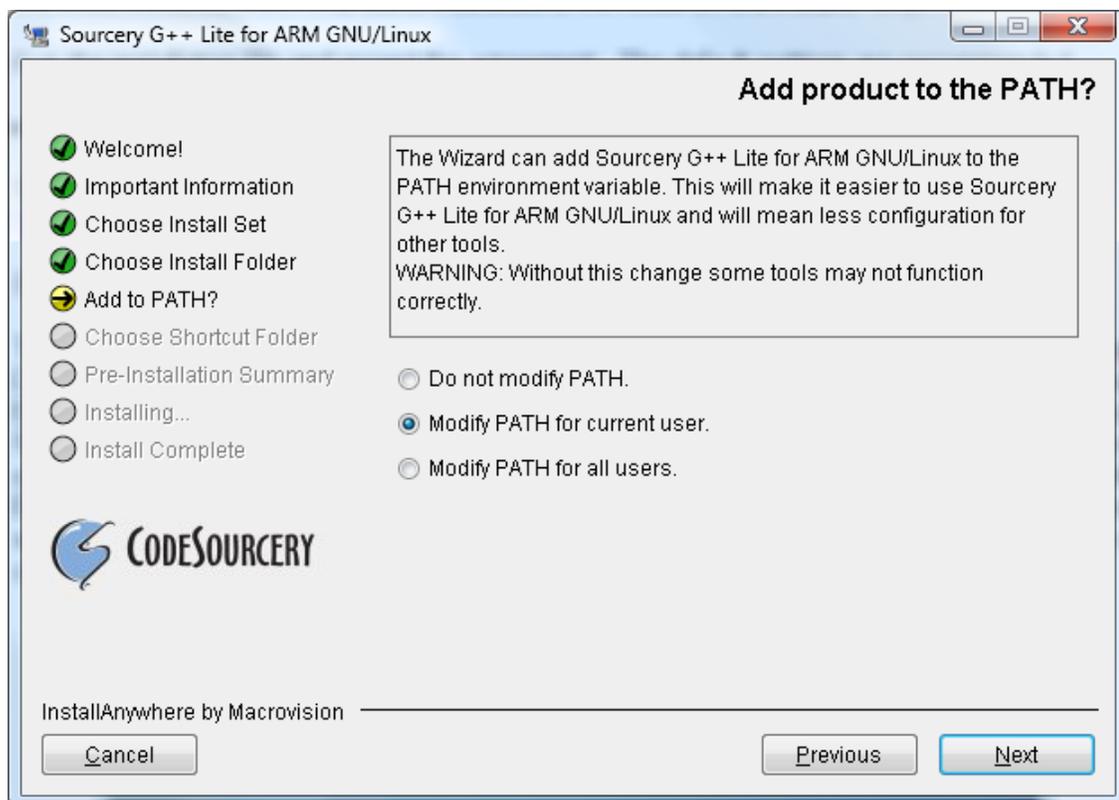


Figure 1, Modifying the PATH

### 3 Downloading and installing the Eclipse IDE

Please go to <http://java.sun.com/javase/downloads/index.jsp> and download and install a JDK.

The Eclipse IDE is an open source graphical IDE, which is the recommended IDE for cross development. Download it from <http://www.eclipse.org/downloads/>. Either the Eclipse IDE for C/C++ Developers or Eclipse Classic should be downloaded. If only C/C++ development is planned for use with Eclipse, the former should be installed. Otherwise, install the Classic version. Installation of Eclipse is as simple as extracting the eclipse folder to the desired location. Click eclipse.exe to start Eclipse. On the first run, Eclipse will ask for a location for its workspace. Choose a desired location and check the box to set the workspace as default. On future runs, Eclipse will no longer ask for a workspace location if the box is checked. A window like below in Figure 2 will be shown.

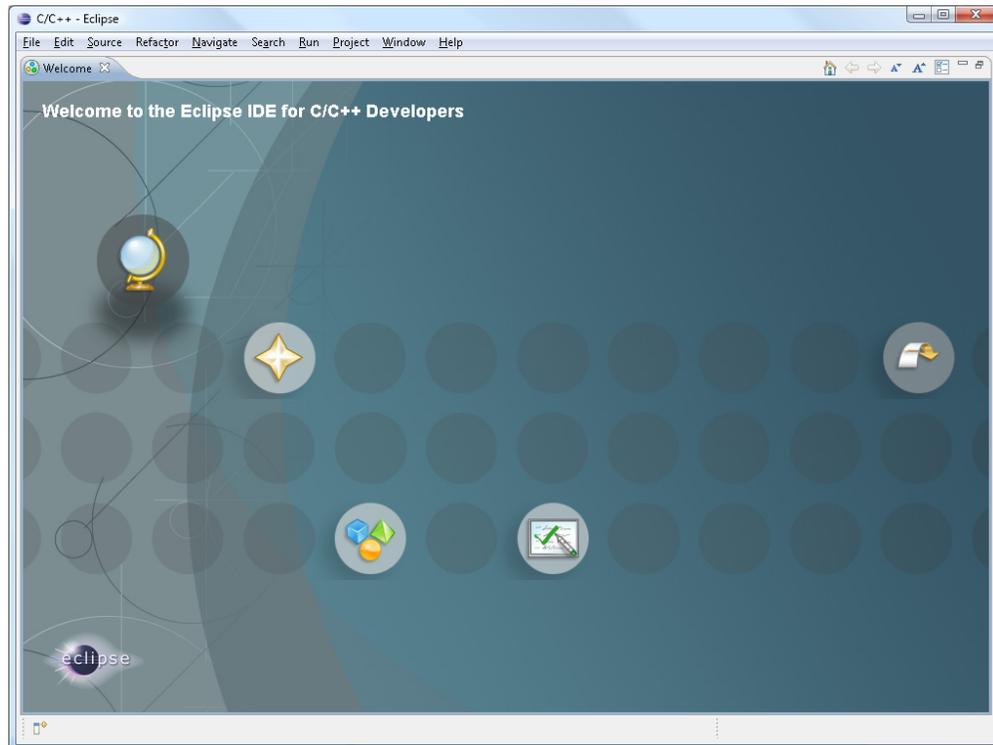
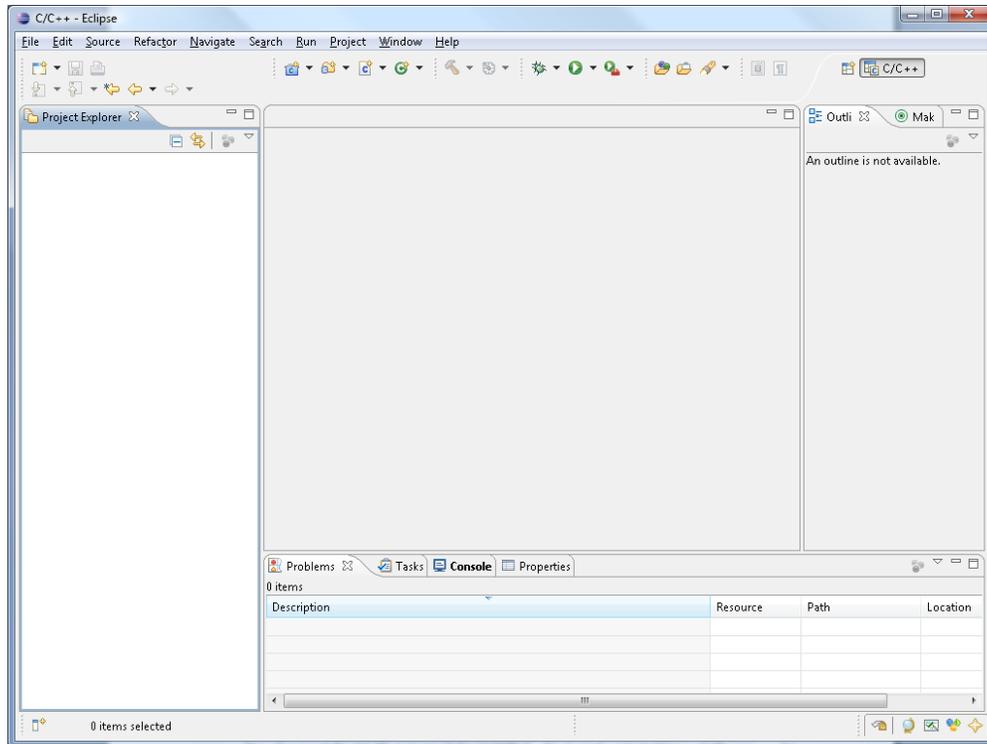


Figure 2, The Welcome Screen

Click the rightmost icon to reach the workbench which is shown below in Figure 3. The welcome screen will no longer appear in future executions of Eclipse and it will go directly to the workbench.



**Figure 3, The workbench**

If Eclipse IDE for C/C++ Developers is not the version installed, the CDT (C/C++ Development Tools) Plug-in must be installed. Go to Help->Install New Software.... As it shows in Figure 4, in the drop-down menu, choose Galileo. A list of categories will appear. Expand the Programming Languages category. Locate and check Eclipse C/C++ Development Tool. Click Next and then Finish to install the plug-in. Restart Eclipse if asked.

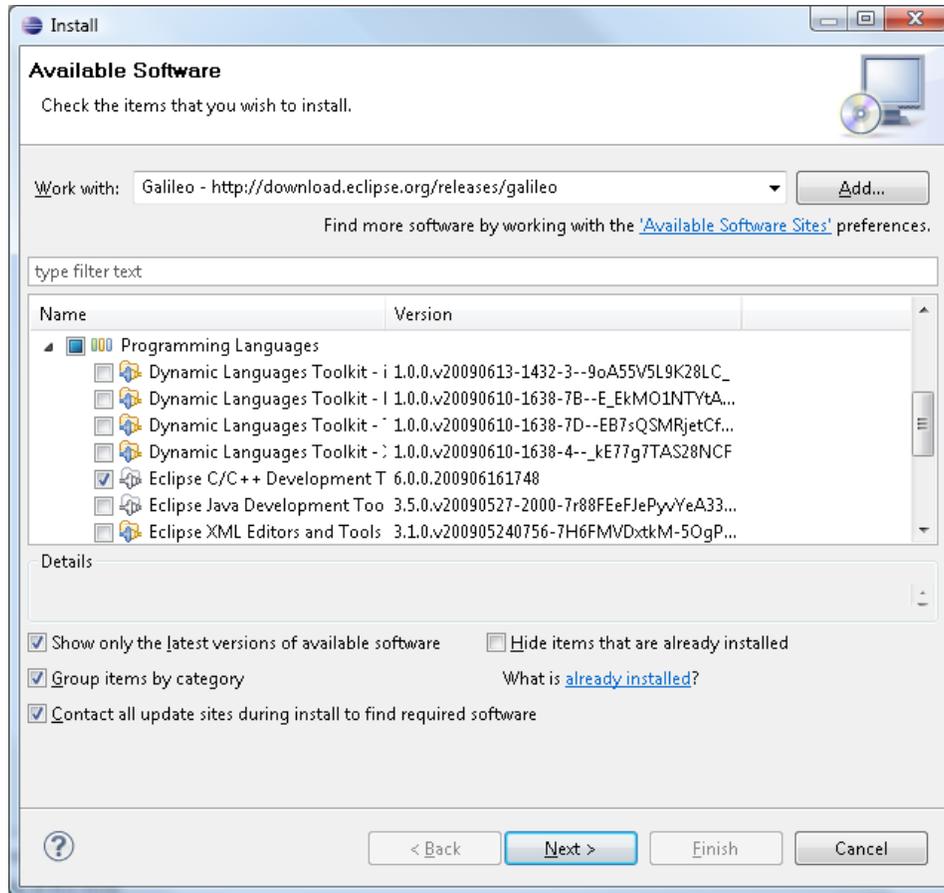


Figure 4, Installing Plug-ins

Now Eclipse can be used for cross development for the ARM platform.

### 3.1 Creating and Building an Example Project.

To start a new C/C++ project, go to File->New->C Project or File->New->C++ Project. For project type, expand *Makefile project* and choose *Empty Project*. For the Toolchain, choose *--Other Toolchain--*. An example of this screen is shown in Figure 5. Choose a project name and click Finish.

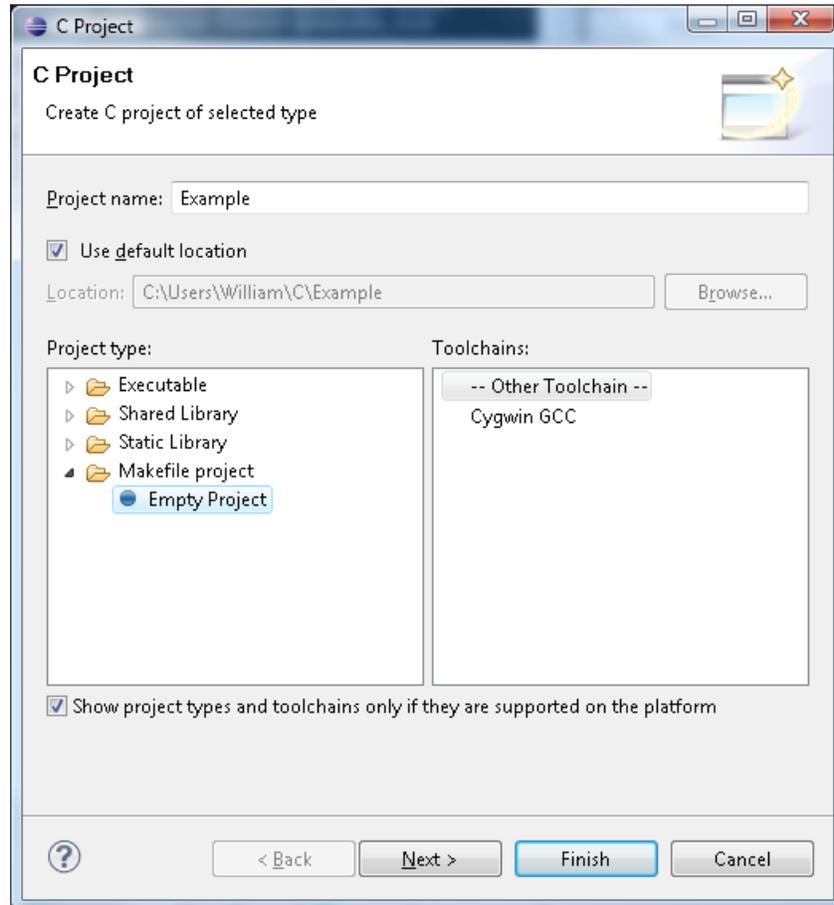
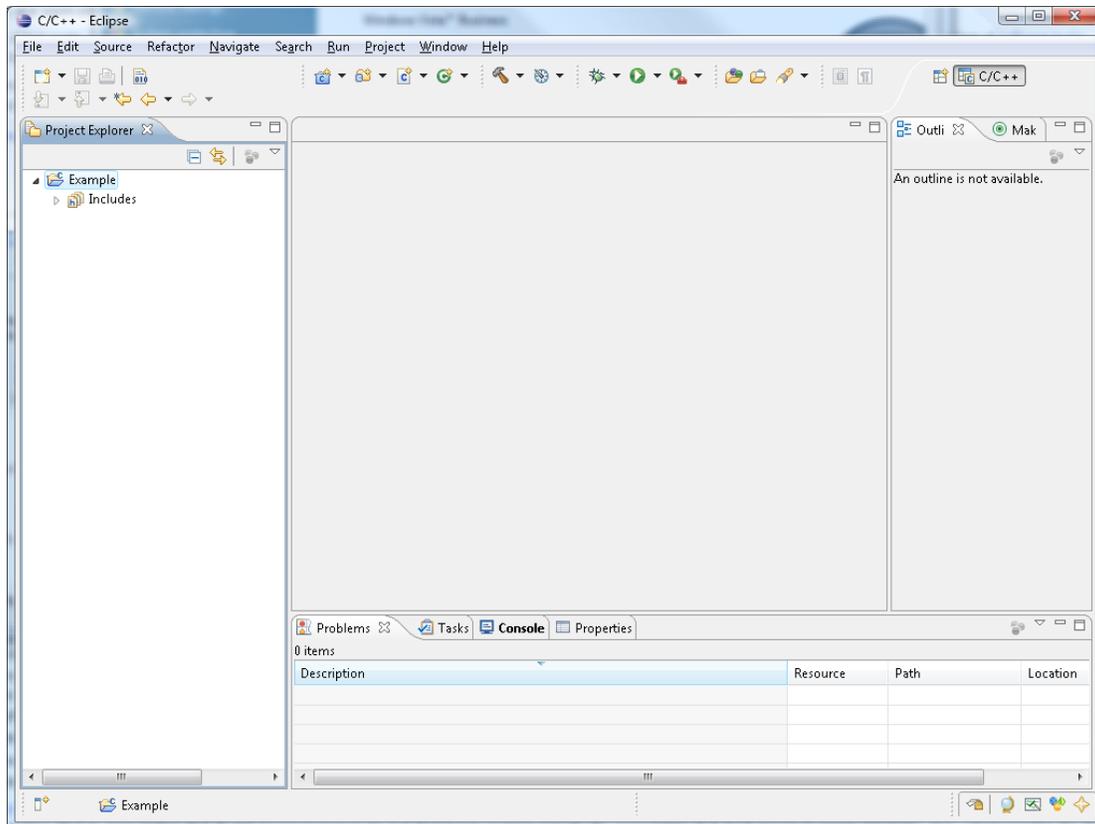


Figure 5, Creating a new project

The workbench should look similar to the one in Figure 6. If the Includes folder does not appear under the Example project folder when expanded, do not worry. In general, the normal case is that Example should be empty. The Includes folder as shown in the figure is created automatically if Eclipse detects that Cygwin GCC libraries are installed on the computer.



**Figure 6, Example Project**

The tutorial example project will be to create a program that reads a number from the command line and prints the factorial of that number or -1 if the number is negative or would overflow a 32-bit integer. A total of 4 files will be created for this project. They are as follows:

- main.c – The source file that will contain main()
- factorial.h – The header file that will provide macro constants and function prototype for the factorial function
- factorial.c – A source which will contain an implementation of the factorial function
- makefile – The file needed to build the project, which will be later explained in greater detail.

To create a new file in the project, right click the Example project folder and go to New->File, New->Header File, or New->Source File (for makefiles, .h files, and .c files, respectively) depending on the desired file type. Figure 7 shows the creation of main.c. Click Finish to create the new file.

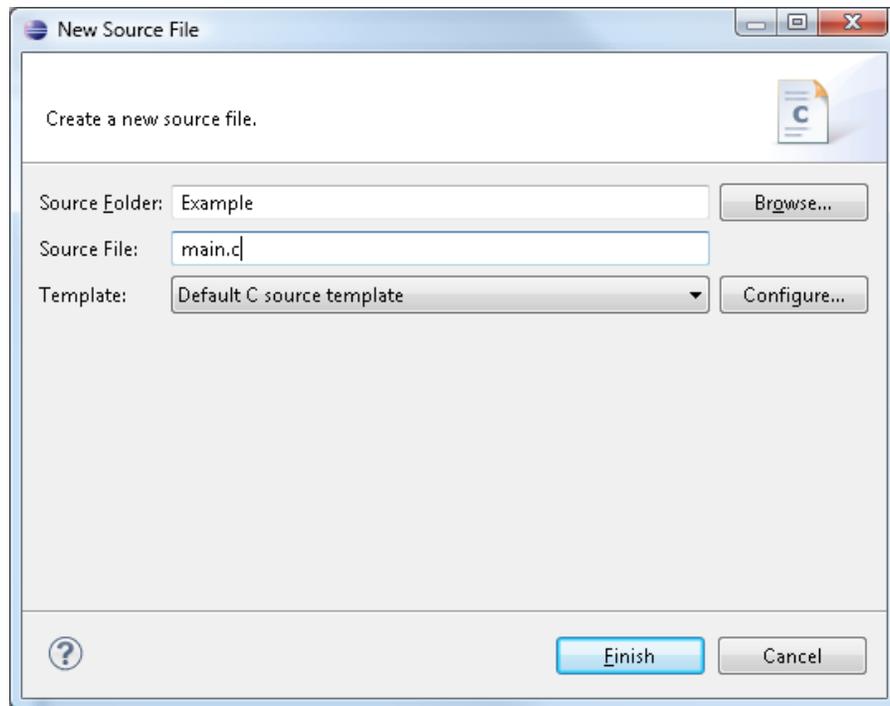
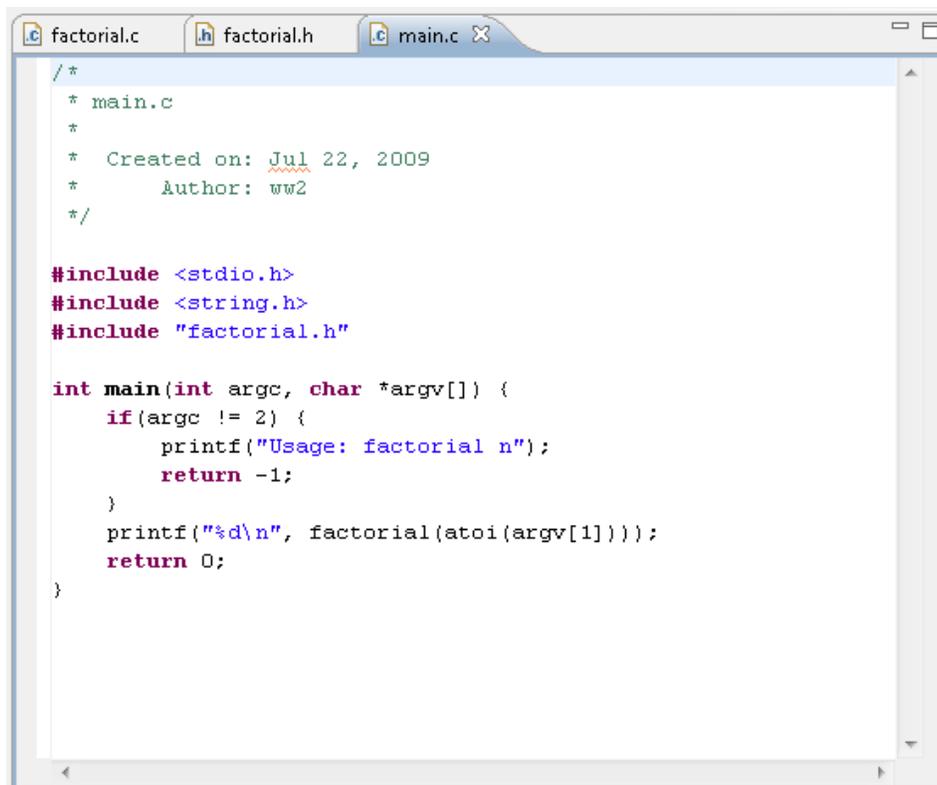
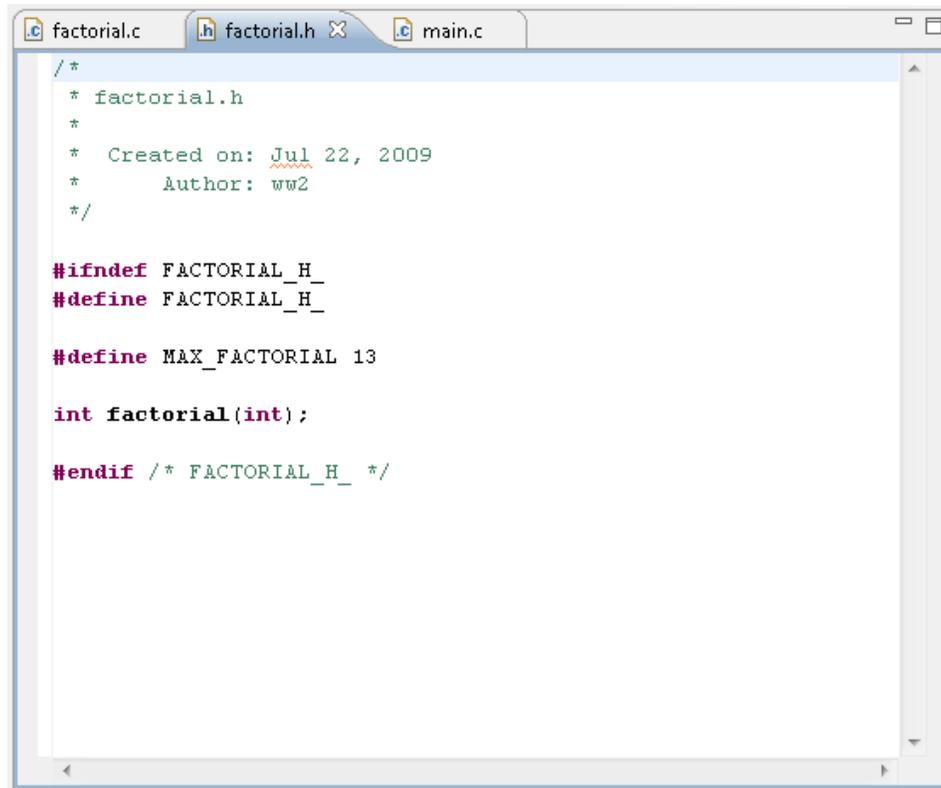


Figure 7, Creating main.c

Figures 8, 9, 10 show what the source code for what main.c, factorial.h, and factorial.c should look like.

Figure 8, main.c





```
.c factorial.c | .h factorial.h | .c main.c
/*
 * factorial.h
 *
 * Created on: Jul 22, 2009
 * Author: ww2
 */

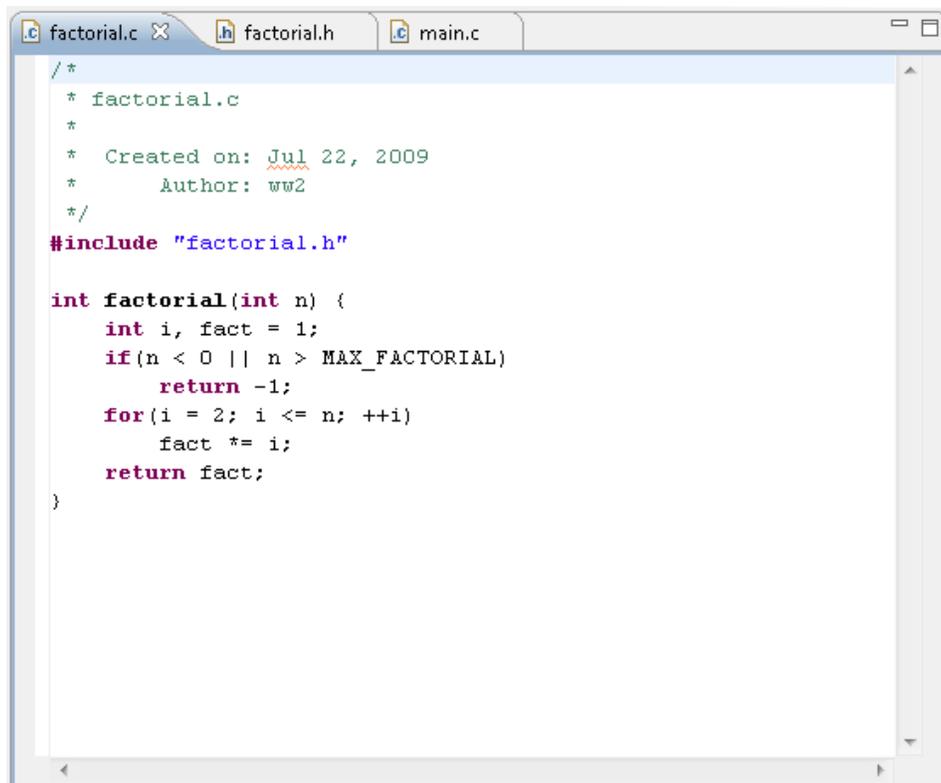
#ifndef FACTORIAL_H_
#define FACTORIAL_H_

#define MAX_FACTORIAL 13

int factorial(int);

#endif /* FACTORIAL_H_ */
```

Figure 9, factorial.h



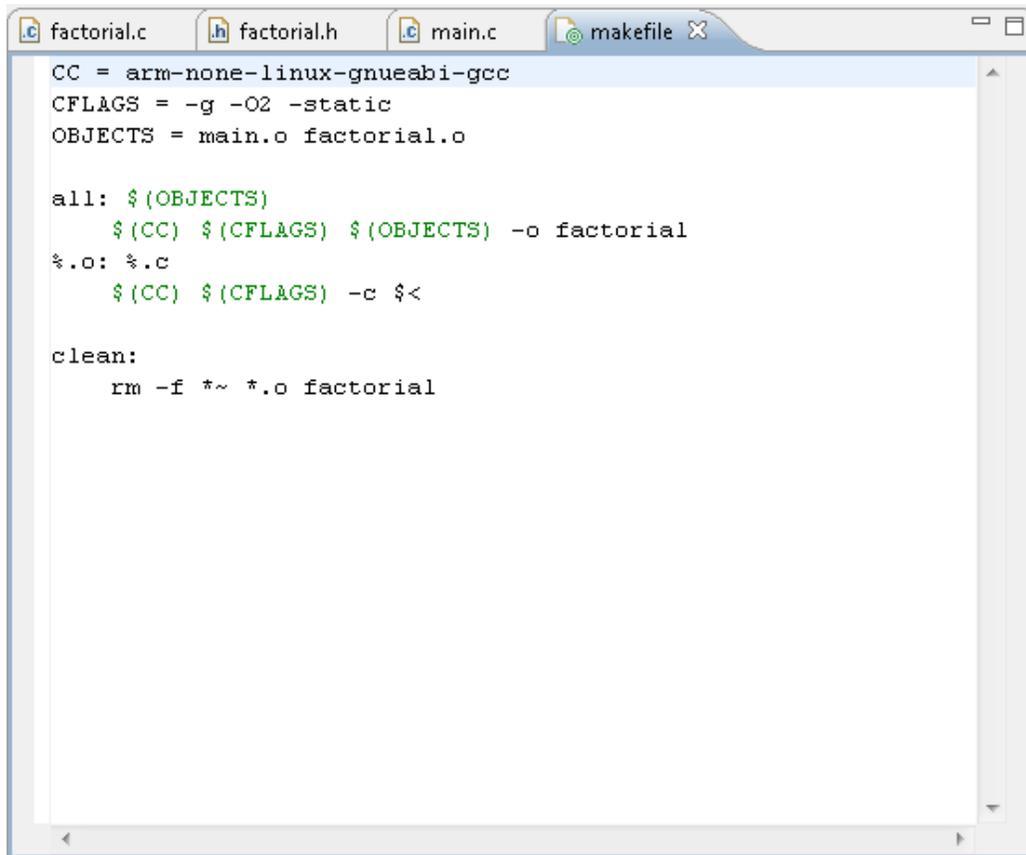
```
.c factorial.c | .h factorial.h | .c main.c
/*
 * factorial.c
 *
 * Created on: Jul 22, 2009
 * Author: ww2
 */
#include "factorial.h"

int factorial(int n) {
    int i, fact = 1;
    if(n < 0 || n > MAX_FACTORIAL)
        return -1;
    for(i = 2; i <= n; ++i)
        fact *= i;
    return fact;
}
```

Figure 10, factorial.c

After writing these files, it should now be possible to compile them into a program. This can be done in two ways. One way is to just call the compiler, `arm-none-linux-gnueabi-gcc`, directly. The other way is to use a makefile. However, for larger projects, calling the compiler directly becomes confusing and complicated. Makefiles significantly simplify the compiling process and thus will be the method explained in this tutorial.

**make** is a utility for automatically building executable programs and libraries by using files called makefiles which specify how to derive the target program from its dependencies. Figure 11 shows an example makefile to build the factorial program



```
CC = arm-none-linux-gnueabi-gcc
CFLAGS = -g -O2 -static
OBJECTS = main.o factorial.o

all: $(OBJECTS)
    $(CC) $(CFLAGS) $(OBJECTS) -o factorial
%.o: %.c
    $(CC) $(CFLAGS) -c $<

clean:
    rm -f *~ *.o factorial
```

Figure 11, makefile

In a makefile, one defines a set of targets, the set of dependencies for that target, and the commands that should be run to build the target. The three targets in the makefile illustrated in Figure 11 are *all*, *%.o*, and *clean*. Anything to the right of the colon is a dependency. Below the rule is the list of commands to build the target. This list of commands **must be TAB indented**. The makefile in Figure 11 only has one command per target, but there can be more.

One can also define macros like in the example. For instance, `OBJECTS` is a macro which lists the object files that are required to build the complete program. The example also shows use of some of the internal macros that **make** recognizes such as `CC` and `CFLAGS` and changes them. `CC` is a macro that contains a string for the C compiler. It normally defaults to a compiler called `cc`, but for programs to run on the ARM SBC, the compiler must be `arm-none-linux-gnueabi-gcc`. The makefile in Figure 11 has this change.

CFLAGS is a macro used to store the set of options or flags that the C compiler should use in compiling. The flags themselves will be discussed later. \$< is a macro that is defined to be the single dependency of the current running target.

The example also shows the use of inference rules. % is a wild card character that matches zero or more characters. The rule, %.o: %.c, infers that an object file can be built by its corresponding source file.

In Eclipse, *all* and *clean* are special targets. When building normally, Eclipse calls make on the *all* target. When cleaning up the project, Eclipse calls make on the *clean* target.

Out of the flags shown in the CFLAGS macro, only `-static` is necessary. The purpose of the `-g` flag is to allow for the code to be run through a debugger such as GDB and is thus recommended, but not necessary. `-O2` is a flag for optimization purposes. The `-static` flag tells the compiler to not use dynamic linking to shared libraries. Normally, programs are compiled such that they make references to libraries located elsewhere. At runtime, the dynamic linker is called which then loads the relevant subroutines into the program. Part of the reason that this is done is to cut down the space taken up by the program. However, the ARM single board computer neither has the correct libraries nor has a usable dynamic linker because it does not have the space to store it. Therefore, programs must be statically linked or have the relevant subroutines loaded at compile time. Although this makes the program larger, it is an insignificant increase. The `-o` flag is for naming the output file. In Figure 11, the program will be named *factorial* when it is compiled. The `-c` flag tells the compiler to assemble the source files, but not to link them together.

Automatic building is the default setting, but should be turned off. Before building the program, click Project on the menu bar and uncheck Build Automatically as shown in Figure 12. If this is not done, the project will be built automatically even after cleaning. This could potentially cause confusion because cleaning implies that all the files should be removed, but building will cause them to reappear again. Also, sometimes building should be purposely held off until later because it could potentially take a long time for larger projects, slowing down the development process.

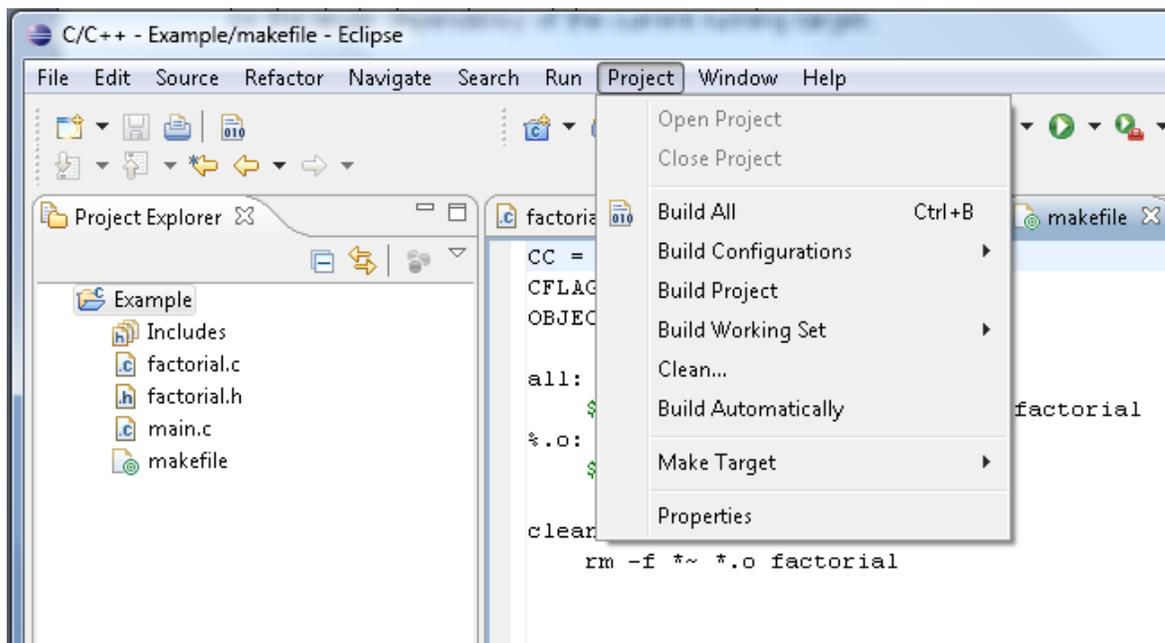
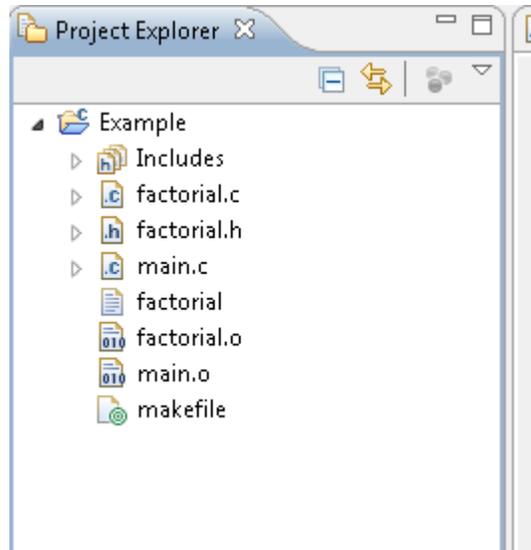


Figure 12, Uncheck Build Automatically

Now either click Build All in the same menu as shown in Figure 12 or right click Example and click Build Project. If everything was done correctly, then the Project Explorer should be similar to the one in Figure 13.



**Figure 13, After building**

It can be seen that the object files as well as the *factorial* program were created. Now this program can be loaded onto the ARM single board computer and it should be able to run. The next tutorial will teach how to connect the Windows machine with the ARM SBC.

## 3.2 Mounting a Windows folder from an ARM Single Board Computer

Once programs have been developed and written on Windows for the ARM SBC, they can be tested by being executed on the SBC. Therefore, the SBC needs some way to access the compile programs. A way to do this is to have Windows share a folder and have the SBC mount this folder as part of its file system. The steps are as follows.

### 3.2.1 Creating a Shared Folder Under Windows Vista

Right click on the folder to be shared. In the menu click Share... as in Figure 14.

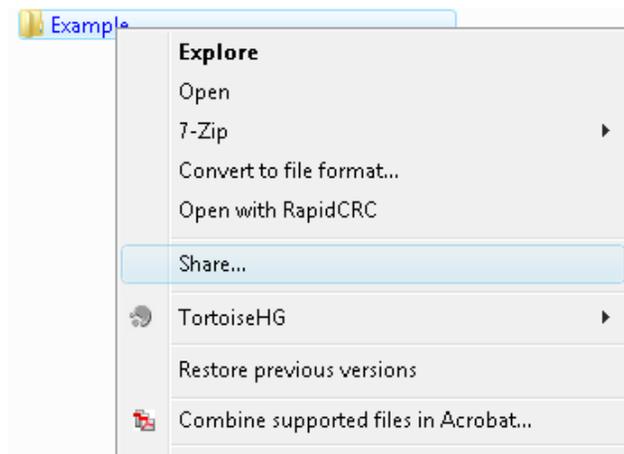


Figure 14, The right click menu

The window in Figure 15 will appear

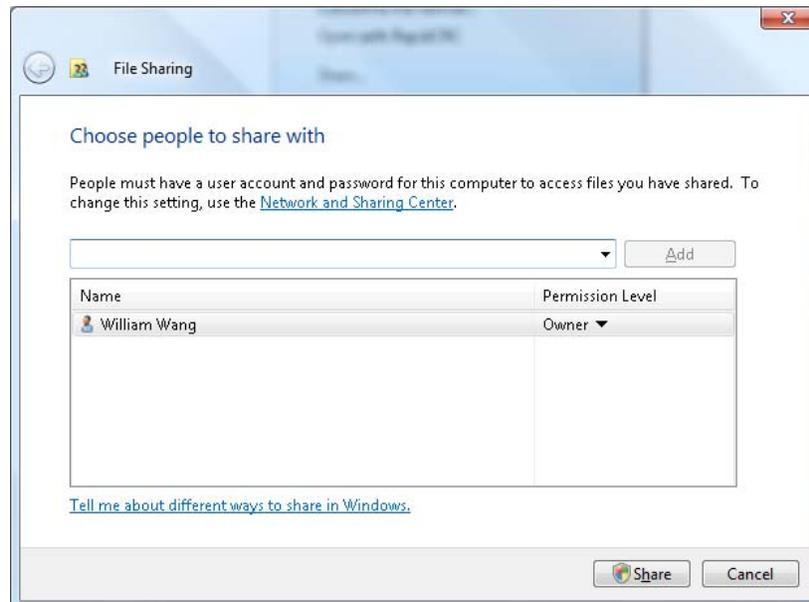


Figure 15, File Sharing

If only the folder creator will use the folder, then click Share to share the folder. Otherwise click the arrow on the drop down menu and choose other users and click Add to add them. Change their permission levels as necessary. There are three permission levels:

- Reader – Users can display the contents of the folder, open files, display attributes, and run programs.
- Contributor – Users have all the rights of Reader, plus the ability to create new folders and files within the shared folder and change or delete any files they have contributed to.
- Co-owner – Users have all of the rights of Contributor, plus the ability to change or delete any files.

### 3.2.2 Creating a Shared Folder Under Windows XP

Right click on the folder to be shared. In the menu, click Sharing and Security... as in Figure 16.

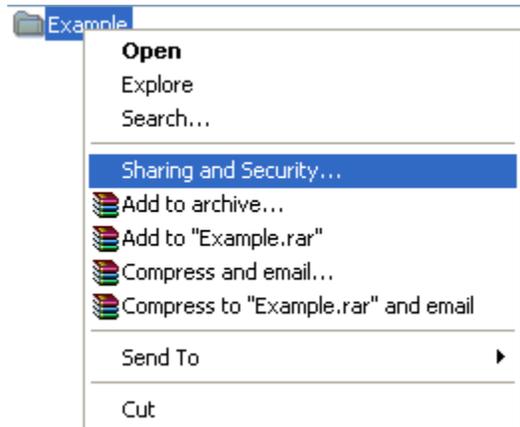


Figure 16, The right click menu

The window in figure 17 will appear.

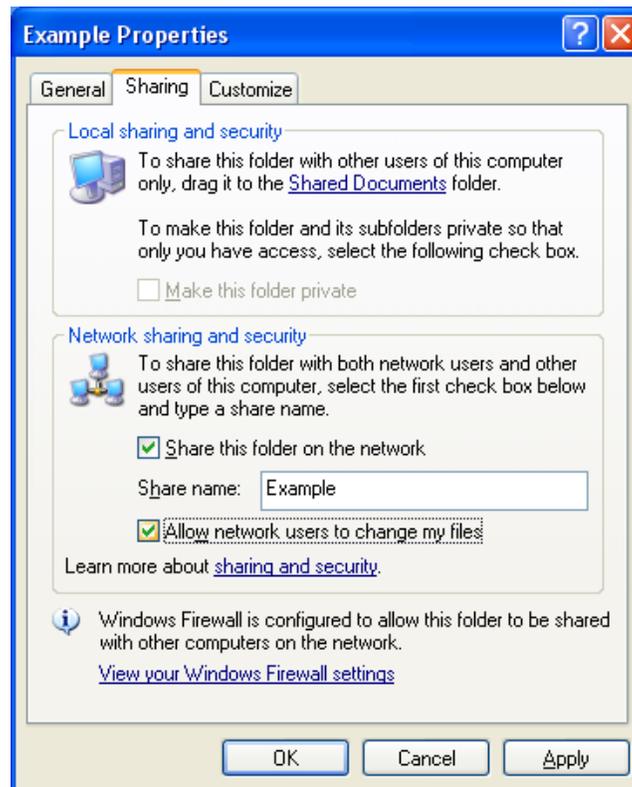


Figure 17, Folder Properties

Check the two boxes as shown in the figure and click Apply. Now this folder is shared.

### 3.2.3 Terminal Access to the ARM SBC

There are two ways to get a terminal window on the single board computer, through serial port or through telnet. A free client that can do both and more is PuTTY, which can be downloaded at <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>. Putty.exe is only one that is needed. When PuTTY is opened, the window in Figure 18 will appear.

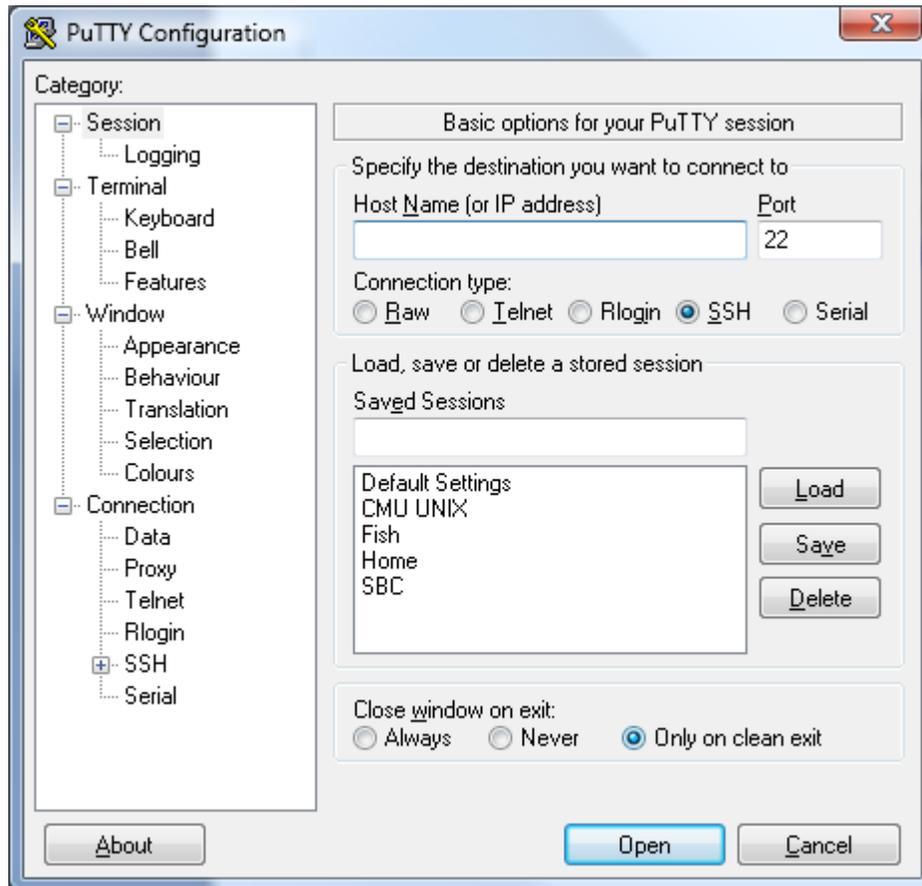


Figure 18, PuTTY

PuTTY can save sessions so that the process of configuration does not have to be done every time. Just put a name in the text field under Saved Sessions and click Save. Click load from the list of sessions and the configuration will be loaded automatically or double click the desired session and a terminal window will open immediately.

### 3.2.3.1 Access ARM SBC via Serial Port

For connecting through the serial port, connect a null modem cable between one of the serial ports of the Windows computer and the serial port of the SBC. Then click Serial in the tree view and set the options as shown in Figure 19. These are the options needed for this particular kind of ARM single board computer.

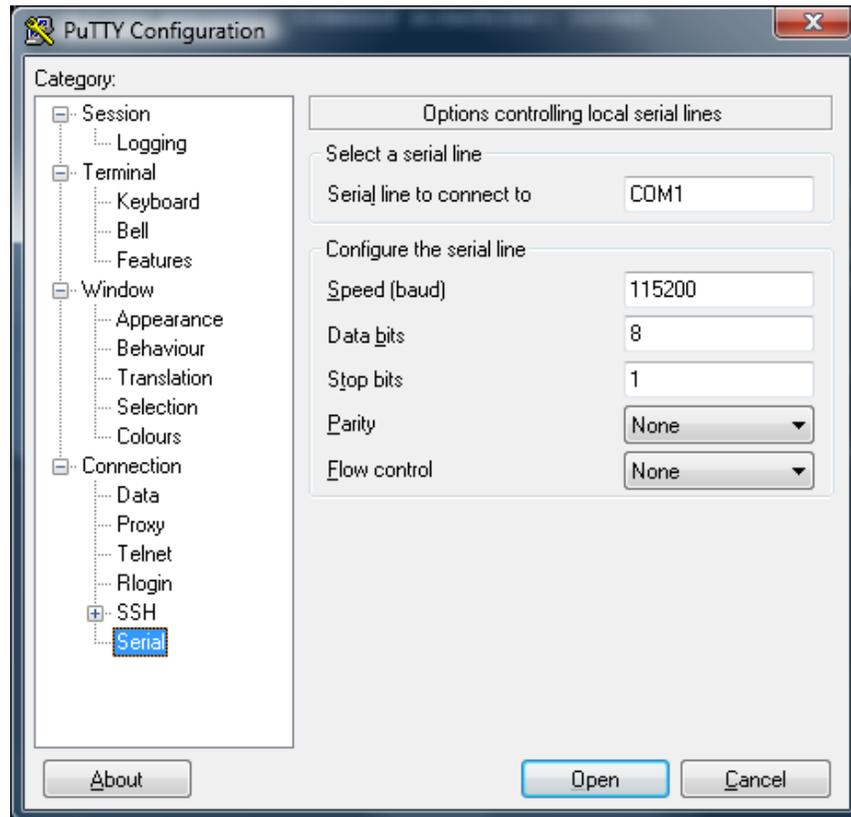


Figure 19, Serial configuration

Then click Session in the tree view to return to the screen in Figure 18 and then click the Serial radio button. It should then look like Figure 20.

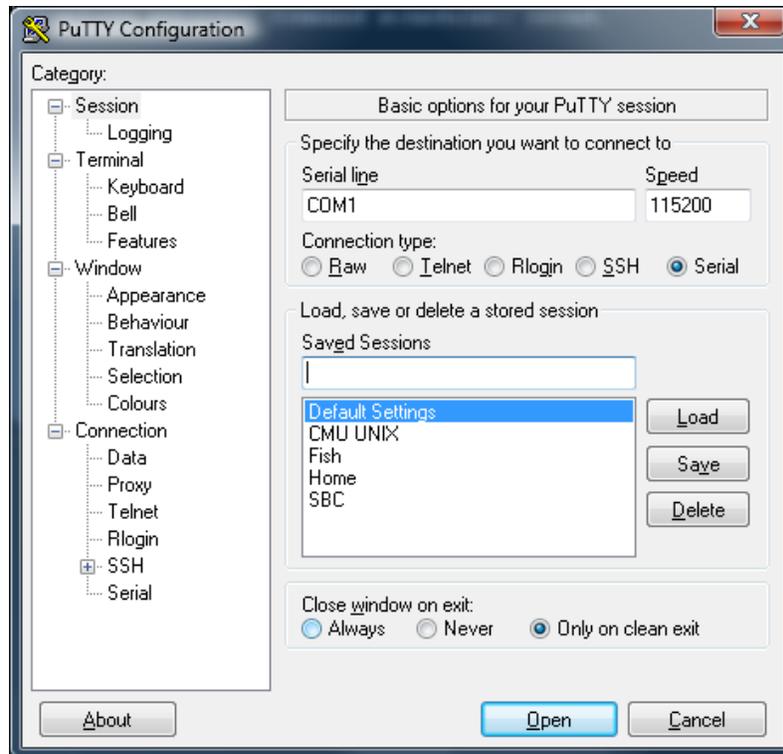


Figure 20, Serial session

Choose the correct serial line (e.g. COM1, COM2, etc), save the session if desired, and then click Open. The terminal window will show up like in Figure 21. When asked for a login, put root. There is no password.

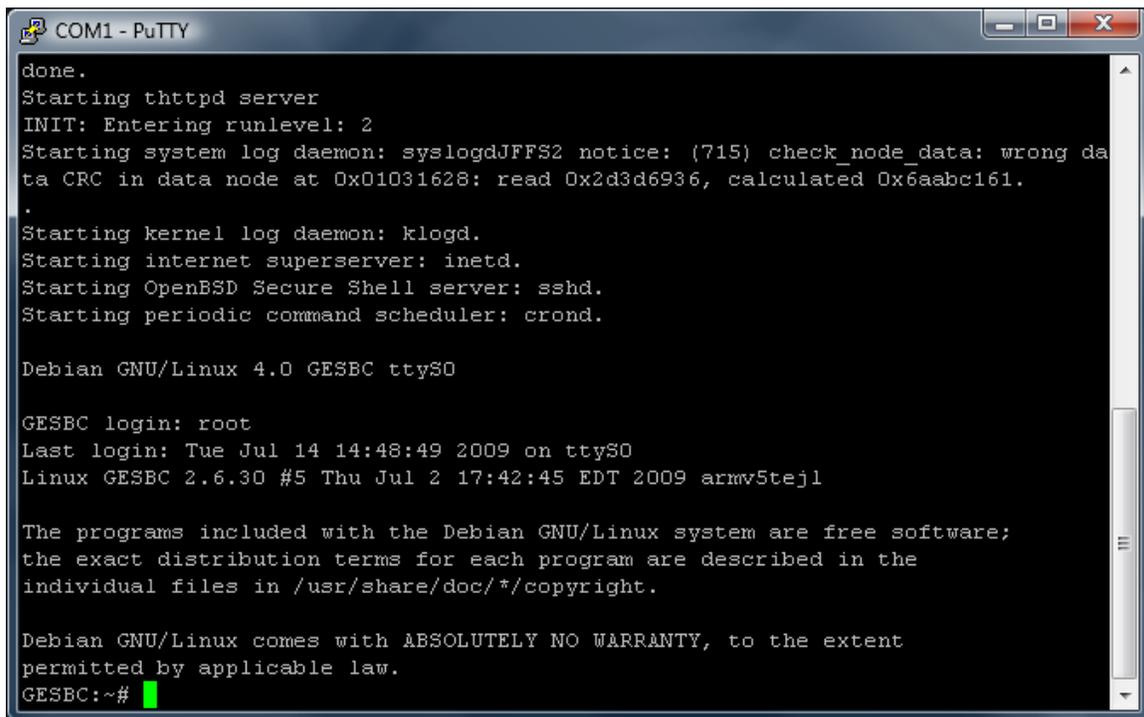


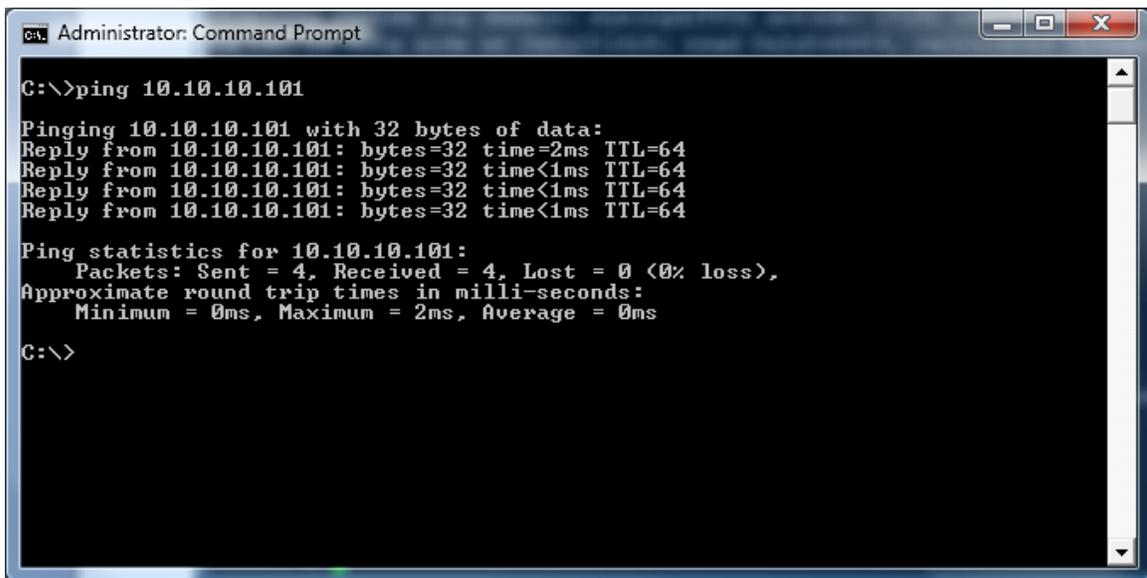
Figure 21, The terminal window

If nothing shows up, press Enter and the login prompt should appear. The text in Figure 21 may not always appear

Note: If pressing Enter does not work, then it may be possible that another process or device is using the same serial line as the SBC. This is especially true of laptops which normally do not have serial ports. Please check the serial line properties. If there is something already using the line, either change the serial port that the SBC uses or move that process or device to a different line. This tutorial will not go in depth on this problem as this is not the focus and there are too many possibilities for the cause of this problem which may need different methods to fix.

### 3.2.3.2 Access ARM SBC via Telnet

For connecting using telnet, the SBC needs an IP address and needs to be on a network which the Windows computer can access. Assuming there is no other way to connect to the SBC, a router is necessary. The SBC is DHCP enabled and thus any router which has its DHCP server enabled can give it an IP address. Connect both the Windows computer and the SBC to the same computer. Wait for some time so that the computers can get their IP addresses. Find the computer in the router's DHCP client table. It will be normally the one with no name. Copy down the associated address and make sure the Windows computer can detect the SBC by using the ping command in the Command Prompt. An example is shown in Figure 22.



```
Administrator: Command Prompt
C:\>ping 10.10.10.101
Pinging 10.10.10.101 with 32 bytes of data:
Reply from 10.10.10.101: bytes=32 time=2ms TTL=64
Reply from 10.10.10.101: bytes=32 time<1ms TTL=64
Reply from 10.10.10.101: bytes=32 time<1ms TTL=64
Reply from 10.10.10.101: bytes=32 time<1ms TTL=64
Ping statistics for 10.10.10.101:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 2ms, Average = 0ms
C:\>
```

Figure 22, The ping command

If above results when pinging to the SBC, then the Windows computer should be able to connect. Then in PuTTY, click the Telnet radio button and type in the address, save if the session if desired, and click Open. See Figure 23 for an example.

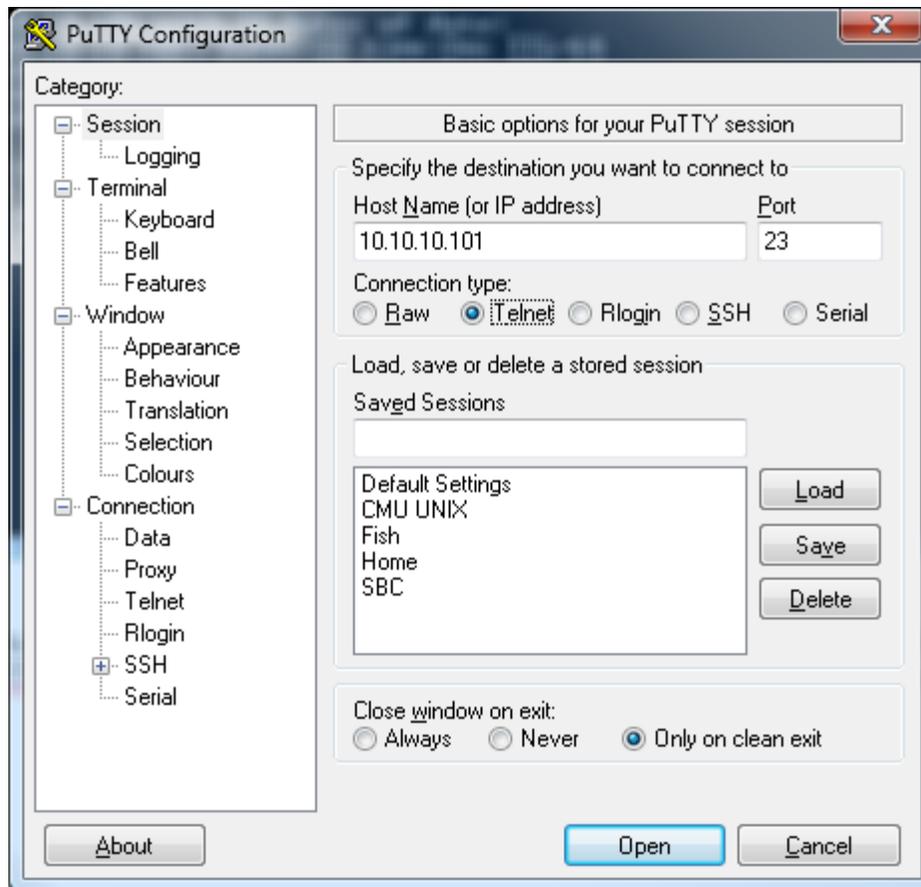


Figure 23, Telnet configuration

The terminal will take some time to load and then the login prompt should appear similarly to the one for the serial connection. Again, the login name is root. There is no password.

### 3.2.4 Mounting the shared folder

Now that a shared folder (refer to step 1) has been created and the Windows machine can connect to the console on the ARM SBC, mounting is the final step to start running programs compiled on the Windows machine.

For mounting, the Common Internet Network Interface (CIFS) will be used. As the name suggests, networking will be required and thus the Windows machine needs an IP address that the SBC can access. This can be done simply by putting the Windows machine and the SBC on the same network. If connecting to a terminal was achieved through telnet, this is already done. If it was achieved through serial port, there are two ways to do it. One requires a crossover cable and is somewhat complicated. The other way is to follow the instructions for setting up the telnet connection. Connect both machines to the router and make sure they have connectivity. Now use either the DHCP table of the router or the ipconfig command in the Command Prompt to find the IP address of the Windows machine. An example run of this command is shown in Figure 24.

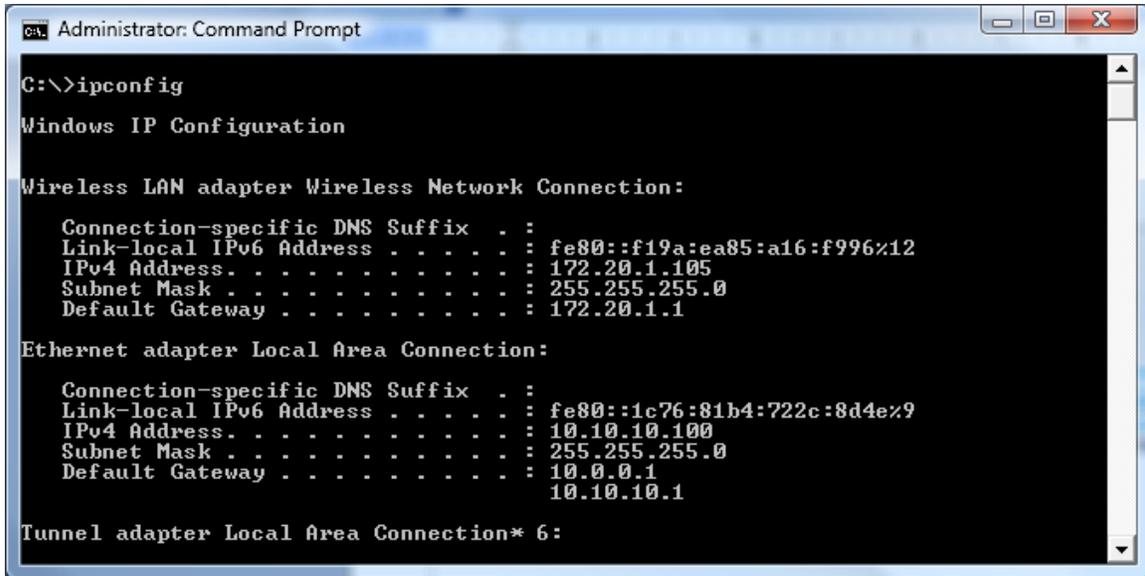


Figure 24, ipconfig

Copy down the IP address located in the network that the SBC is in. Now in the SBC, type the below command if on Vista:

```
mount -t cifs //<IP address>/<share name> -o username=<username>,password=<password> /mnt
```

Where <IP address> is the IP address of the Windows machine, <share name> is the path to the folder in Vista (e.g. C:\Users\Example, share name is Users/Example) or the share name chosen in Figure 17 in XP, <username> is the username of the user who wishes to access this folder, <password> is the user's password. The password is in clear text. If this is not secure enough, then type in

```
mount -t cifs //<IP address>/<share name> -o username=<username> /mnt
```

The terminal will then prompt for a password in hidden text. If the username is not specified like below:

What this command does is that it mounts the folder in the /mnt directory. Thus the files can be accessed by going to that directory on the SBC. A different directory can be used if desired. The /mnt directory is meant for temporary mounts. To unmount the folder, use the below command:

```
umount /mnt
```

Now for an example, the factorial program created in the Sourcery G++ Lite toolchain tutorial will be placed into an example shared folder. The folder will be mounted by the SBC and the code will be executed. Then the folder will be unmounted.

First, place the factorial binary into the shared folder as shown in Figure 25.

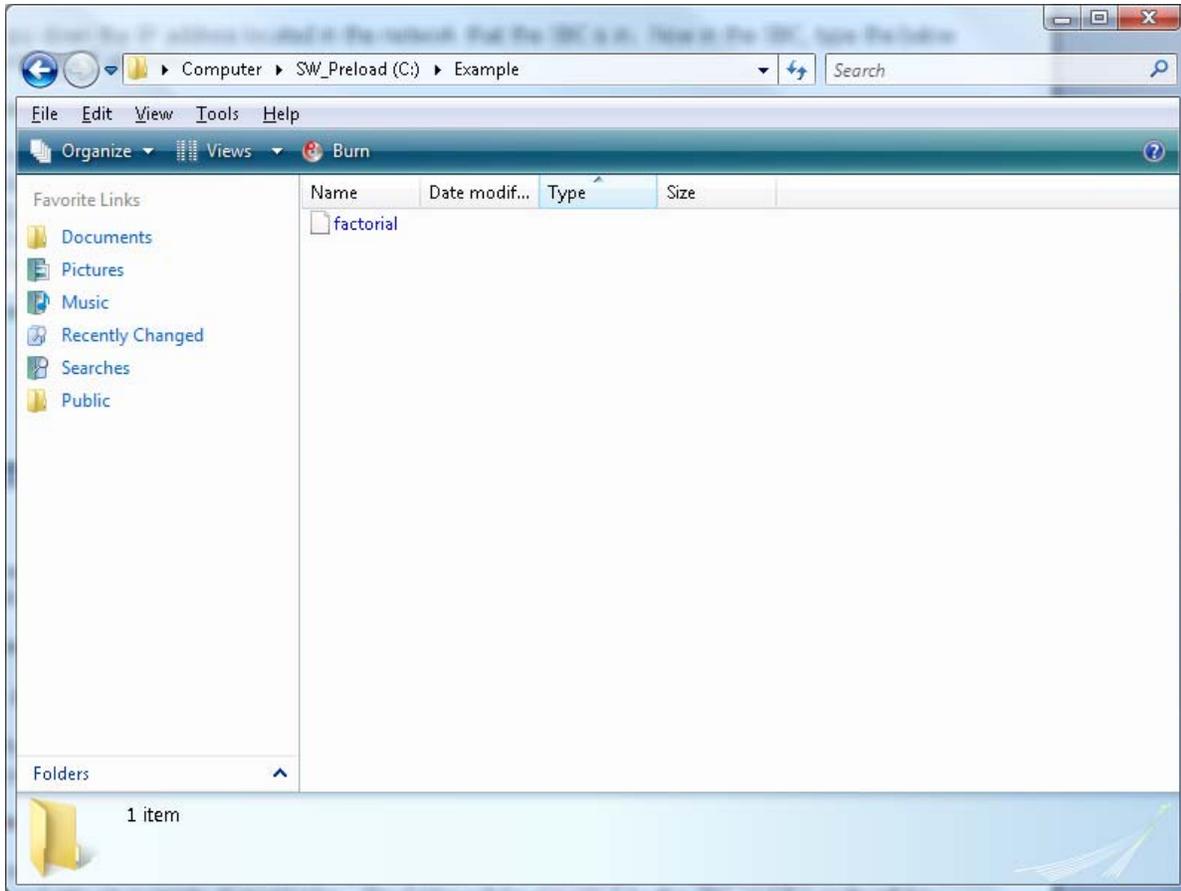


Figure 25. *factorial* in the shared folder

Then, open a terminal window and enter in the mount command, change to the /mnt directory, execute the program, and *umount*. Figure 26, shows an example terminal run from Telnet.

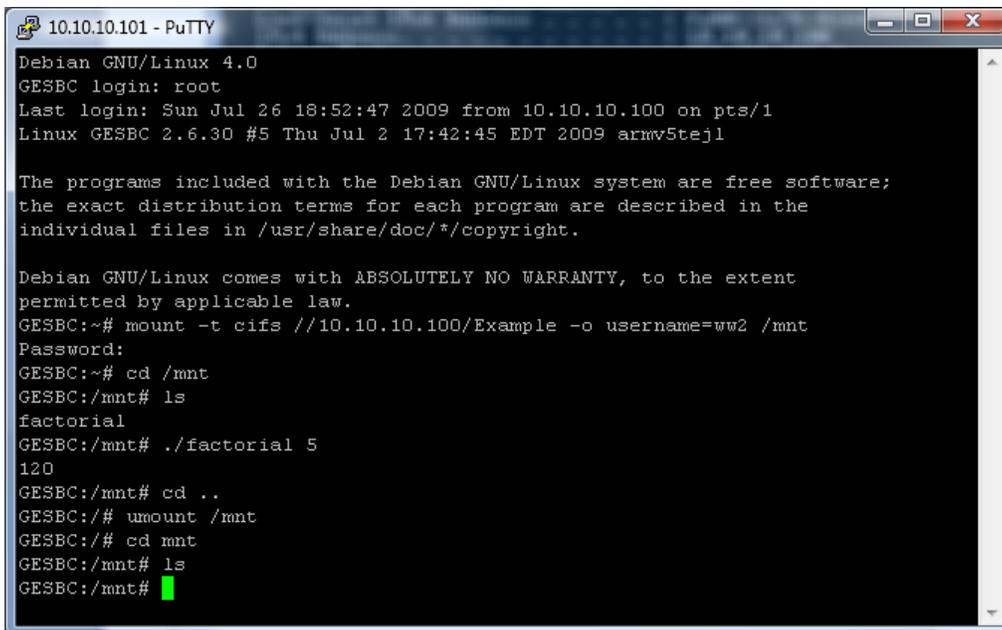
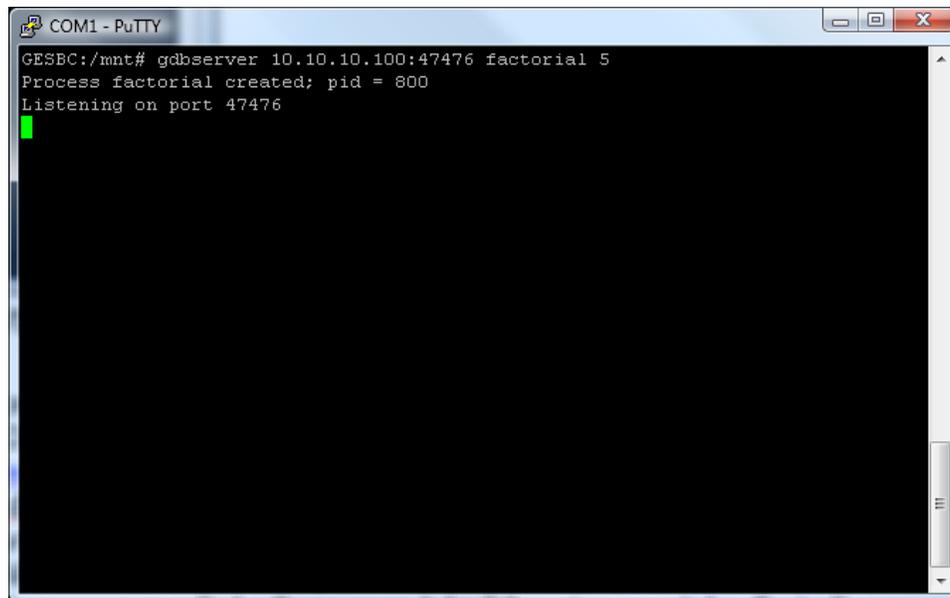


Figure 26, Example run

There is an extra command used in this example called *ls*. *ls* is a command to list all the files and folders in a directory and is very similar to *dir* in a Windows Command Prompt. *ls* was used in this example to demonstrate that mounting and unmounting were successful. After mounting, the factorial program was listed. After unmounting, the directory became empty. One thing to note is that the program was called using *./factorial* (i.e. the absolute path) rather than just simply *factorial*. Unlike Windows, Linux does NOT check for the program in the current directory. It only checks for programs in the directories listed in its *PATH* variable. Another thing to note is that unmounting cannot occur while still in the directory. Hence, this is why the directory was changed to the parent directory before calling *umount*. Now, one can run programs compiled on a Windows machine using a cross development tool on the ARM SBC.

## 4 Appendix I: Remote debugging with *gdbserver* and Eclipse in Windows

As stated above in section 3.1, if a program was compiled with the `-g` option, then it is possible to debug it with a debugger such as *gdb* (GNU Debugger). To learn more about *gdb* and the various functions it provides, please go to [http://sourceware.org/gdb/current/onlinedocs/gdb\\_toc.html](http://sourceware.org/gdb/current/onlinedocs/gdb_toc.html). The Sourcery G++ Lite comes with its own version of *gdb* called *arm-none-linux-gnueabi-gdb* which can be used to debug programs using this toolchain. Unfortunately, as Windows does not run on an ARM processor architecture, it cannot natively run ARM code. To solve this problem, it is possible to have an ARM based computer (such as the SBC) serve the program through a program called *gdbserver* and do debugging remotely. Basically how this works is that a debugger remotely connects to the ARM computer and has the ARM computer perform the execution of the program for the debugger rather than its host computer. Figure 27 shows an example run of *gdbserver*.

A screenshot of a terminal window titled "COM1 - PuTTY". The terminal shows the following text: "GESEBC:/mnt# gdbserver 10.10.10.100:47476 factorial 5", "Process factorial created: pid = 800", and "Listening on port 47476". A green cursor is visible on the line "Listening on port 47476".

```
COM1 - PuTTY
GESEBC:/mnt# gdbserver 10.10.10.100:47476 factorial 5
Process factorial created: pid = 800
Listening on port 47476
```

Figure 27, *gdbserver*

*gdbserver* takes at least 2 arguments. The first argument it takes is the location of *gdb*. This can be a combination of an IP address and a port as shown above, or a different port (such as a serial port). The second argument is the program to serve. Any arguments that follow are used by the program being debugged. As it shows, *gdbserver* waits and listens for a connection from an instance of *gdb*. One thing to note is that *gdbserver* cannot be stopped through the means of pressing CTRL+C. *gdbserver* terminates either by ending a debugging session or using a command called *kill* through another terminal.

Eclipse can be used in tandem with *gdb* to provide a graphical debugging environment and can also interact with *gdbserver*. To set this up, in Eclipse, go to Run->Run Configurations (or Debug Configurations) to see the screen in Figure 28.

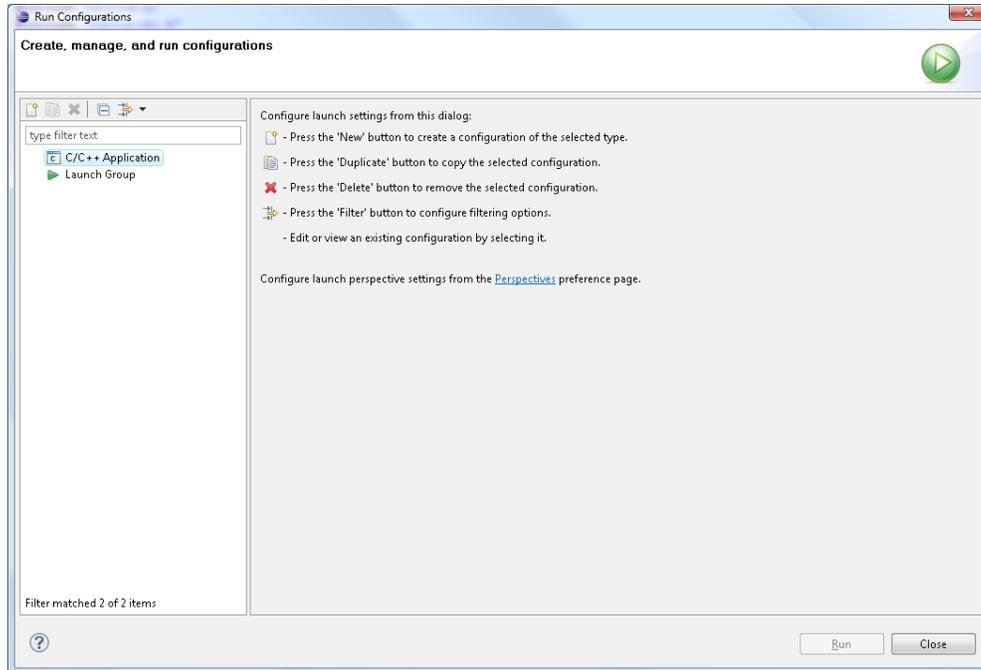


Figure 28, Run Configurations

Double-click C/C++ Application to create a new configuration. Give it a name. Note that the program has to be on both the ARM computer and the machine running Eclipse. This is because the symbol table and various other bits of debugging information are loaded from the local copy of the program. Thus, choose the project to debug under the Main tab. Then choose the program to debug. In the case that the program's file extension is not .exe, the program must be found through Browse... Otherwise, the program can be found using Search Project... Figure 29 shows how to set it up for the example created in section 3.1.

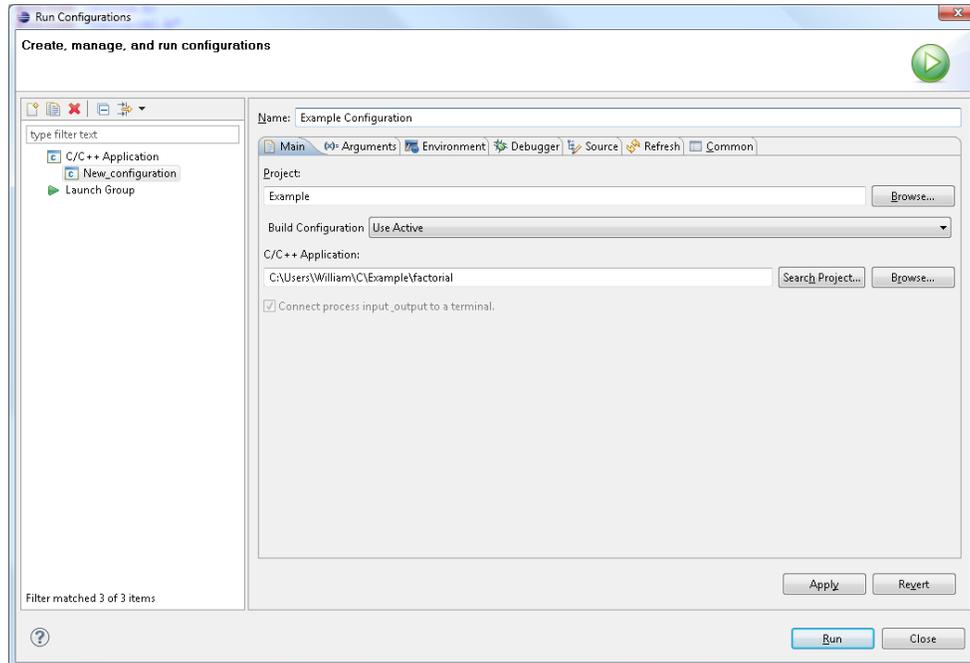
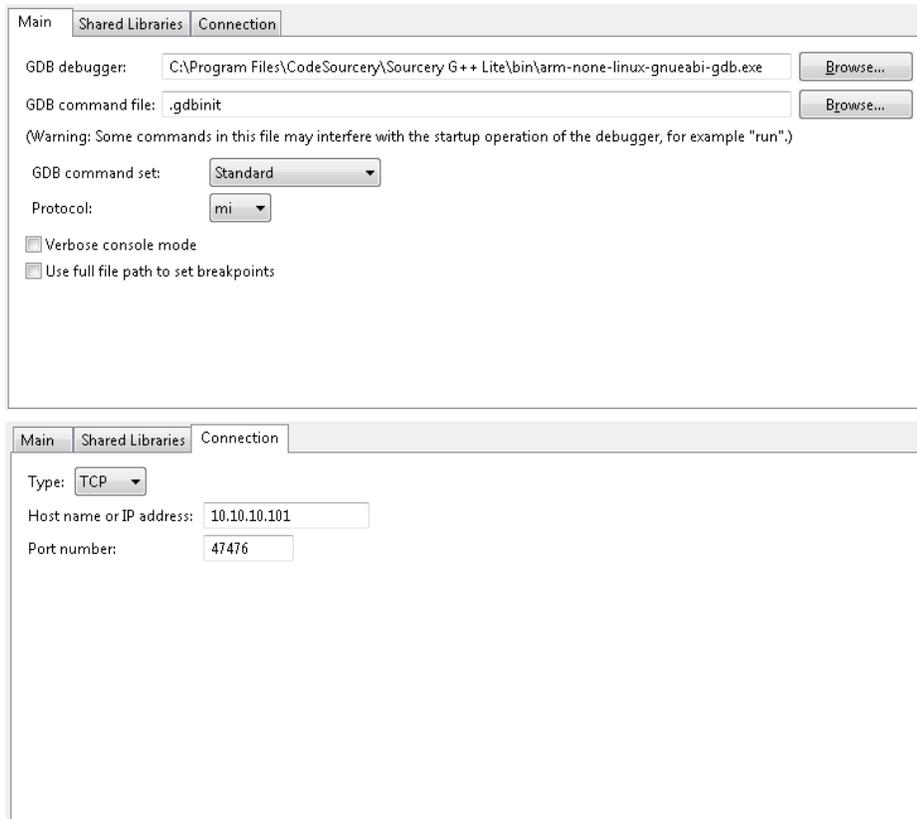


Figure 29, Configuring the Main tab

Now click the Debugger tab. Change the option in the dropdown menu to *gdbserver* Debugger. Choose a function to stop at if desired (if unchecked, the program will attempt to run to completion). Change the debugger to arm-none-linux-gnueabi-gdb. The Shared Libraries tab is for adding the shared libraries that gdb should use. The assumption is that no shared libraries are used and thus the Shared Libraries tab will be skipped. Move to the Connection tab. Change the connection type to the one being used. If using TCP, type in the IP address and port number of the computer where *gdbserver* is executing. Click Apply afterwards to save the configuration. Figure 30 shows an example of how the Main and Connection tabs under the Debugger tab should look.



**Figure 30, Configuring the Debugger tab**

If *gdbserver* is running, then it is now possible to start the remote debugging process. If in Debug Configurations, choose the correct debug configuration and click Debug. Otherwise, click the arrow right of the **“bug icon”** in the menu bar at the top and choose the correct debug configuration (refer to Figure 6). The message box in Figure 31 should pop up.

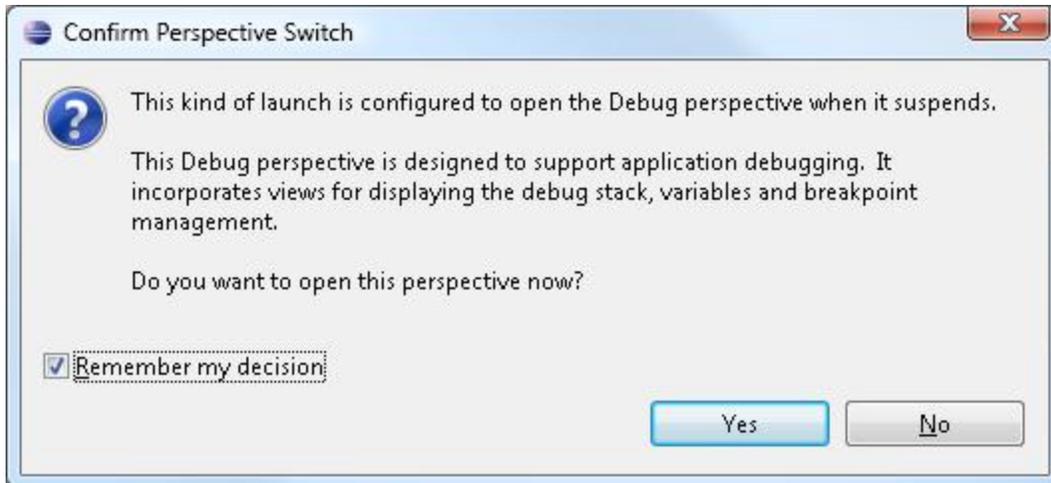


Figure 31, Changing Perspective

Eclipse asks if a switch should be made to the Debug perspective. Debug perspective gives many useful tools such as a register viewer and a memory viewer. It is recommended to click Yes and check the box to remember the decision. If Yes is clicked, the screen will change to something similar to the one in Figure 32.

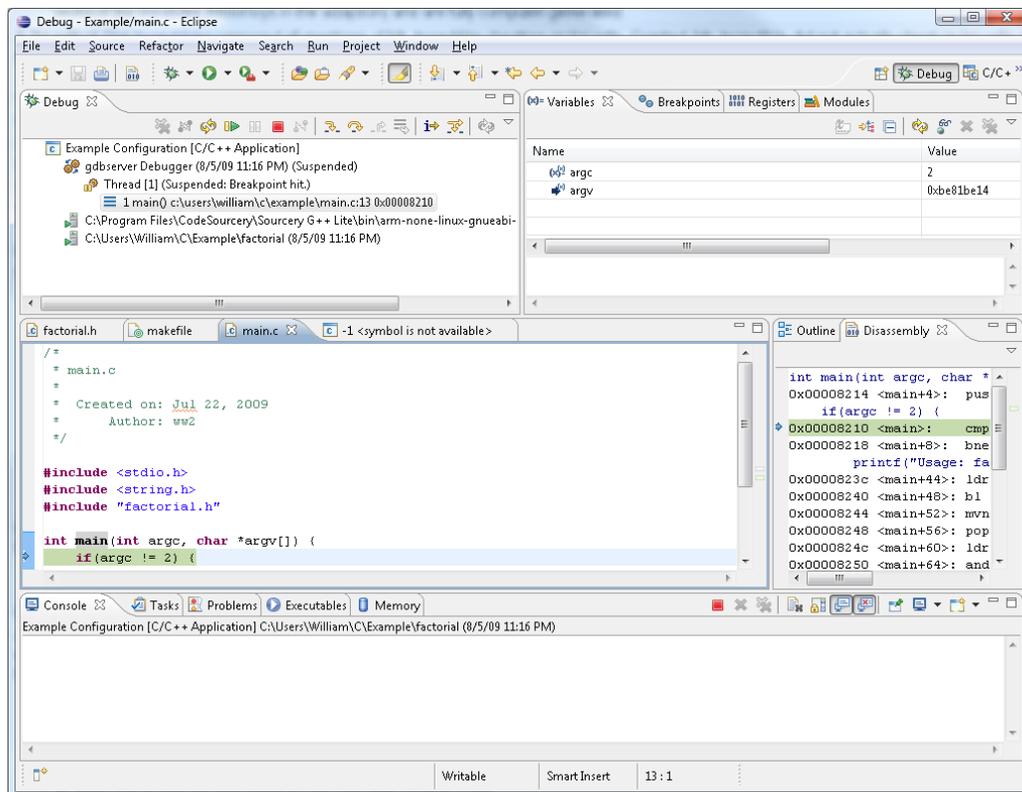


Figure 32, Debug Perspective

As shown in Figure 32, other than the tool mentioned above, there is a disassembly of the code where one can choose to move to different lines or run up to specific lines of assembly. The variables that are in scope are also shown. In this configuration a breakpoint was set at main. More breakpoints can also be set as well by double clicking the space on the left of the code. In the upper left hand corner under

the Debug tab is another place that can be used to step through the code. To have the code step until the next breakpoint, either click the Pause/Play icon or press F8. Once the code has terminated, the screen should look similar to Figure 33.

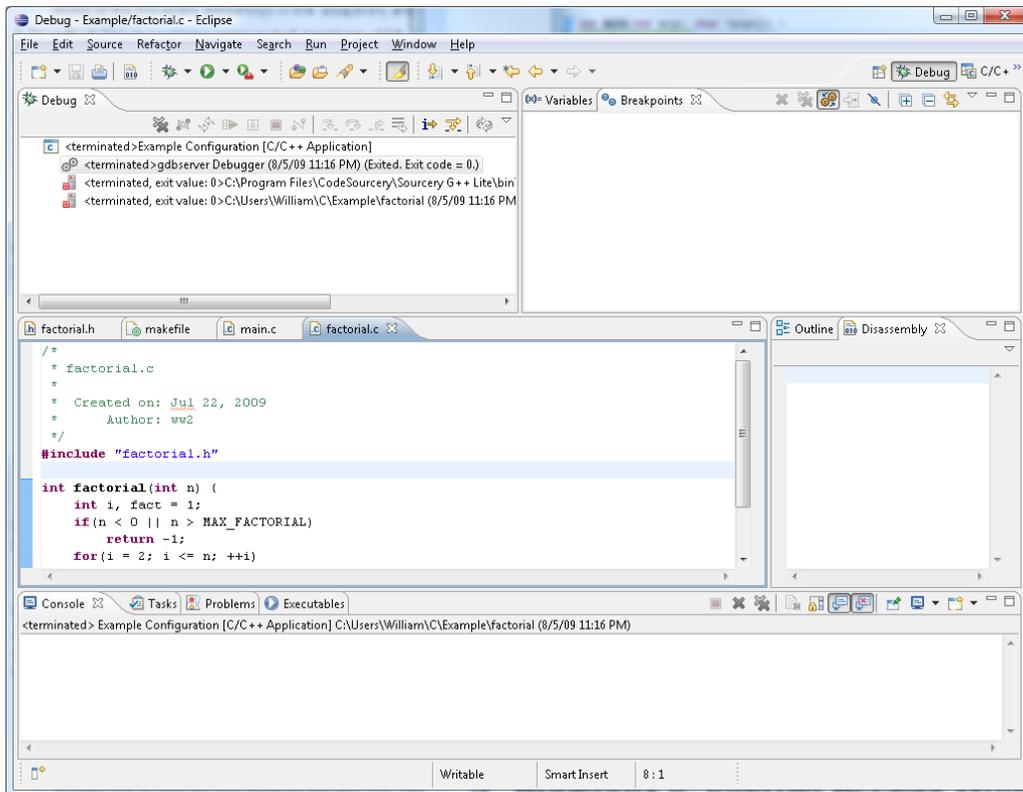


Figure 33, Program Termination

If one looks at the terminal session, it should look similar to Figure 34.

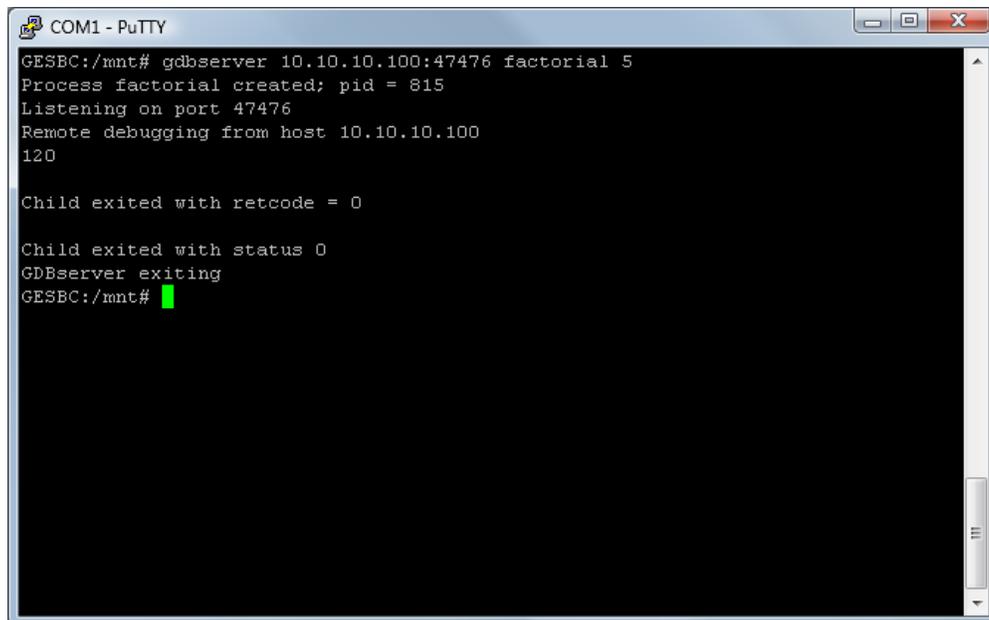


Figure 34, gdbserver Termination

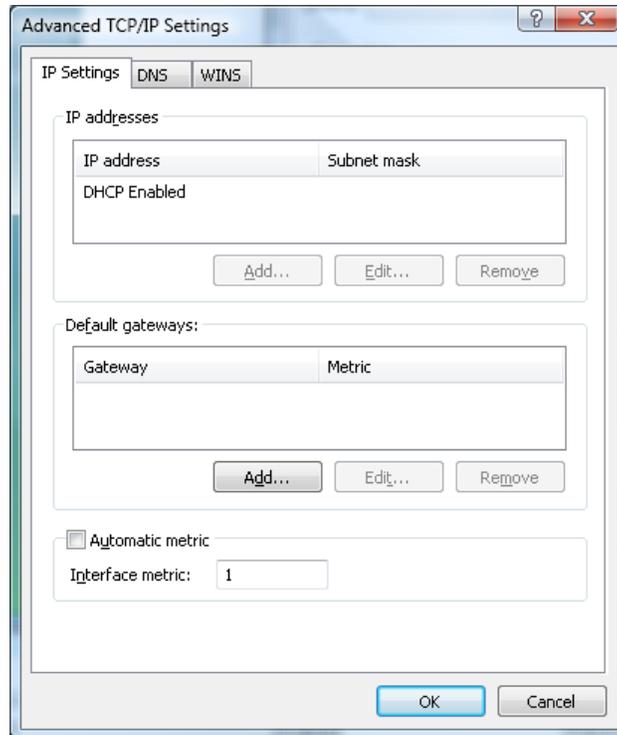
As shown in the figures, the printing to the console occurs on the ARM computer and not the console in Eclipse. Also shown is that gdbserver has terminated. To debug the program again, run gdbserver again.

One final note about the `-g` option. As stated before, this flag is needed for use with `gdb`. This is because this option tells the compiler to add extra information needed to debug effectively. However, also noted above is the fact that there needs to be two copies of the program, one on the machine running Eclipse and one on the ARM machine. `gdb` only needs to look at the local copy to get the debugging information and thus the copy on the ARM machine does not need it. If it is necessary to conserve space on the ARM machine, compile a copy of the program without the `-g` option and put that copy on the ARM machine. This copy will not have the debugging information and therefore a bit smaller.

## 5 Appendix II: Internet Access While on Multiple Networks in Windows

The mounting tutorial asks one to place the SBC on the same network as the Windows machine. This can be as simple as connecting the SBC to router that is currently being used by Windows machine. However, sometimes it is easier to have the Windows machine connect to two different networks at the same time, one that it normally is connected to and a network specifically for communication between the Windows machine and the SBC. Such is the case is when the Windows machines is connected wirelessly to a router for its Internet connection and the router is located relatively far away from the computer. It may be easier to get another router in the same area as the computer and connect the two on this router. However, this could cause a problem with the Windows' machine's Internet connection. When on multiple networks, Windows only tries to find Internet access on the network with the lowest metric or the network that Windows believes to be "closest." The problem is that Windows automatically assigns a lower metric to a wired network as compared to a wireless one and thus the Internet connection on the wireless network is not used.

To deal with this problem, metrics have to be manually set, specifically the network with the Internet connection should be set with a very small metric. To do this, go to the properties of the network device with an Internet connection. Click on TCP/IP (TCP/IPv4 on Vista) in the list and click Properties. Then in the General tab, click Advanced... At the bottom of the IP Settings tab, uncheck Automatic metric and put a small number as the metric (1 is a good choice). Figure 35 shows the window.



**Figure 35, Setting the metric**

After setting this metric, the computer should now have Internet access. In general, the metric should not need to be changed. It is recommended to check Automatic metric when this set up is not being used.