EE414 Embedded Systems
Lab 1. Cross Development Environment

Due Demo & Report: 11:59 PM, March 24 Wed. via KLMS.

1. Purpose

Get acquainted with the cross development system:

Target board: Beaglebone Black with Linux (in microSD).

Host computer: PC with Linux, cross compiler, NFS, and minicom.

2. Problem Statement

Problem 1 (An Embedded Application Program)

Construct a cross-development system on a Lab (or your own) PC. Edit and cross compile an example embedded application program: *Compute a sine function using Taylor series*. Download to the embedded system, debug, and run.

3. Technical Backgrounds

A. Hardware connection

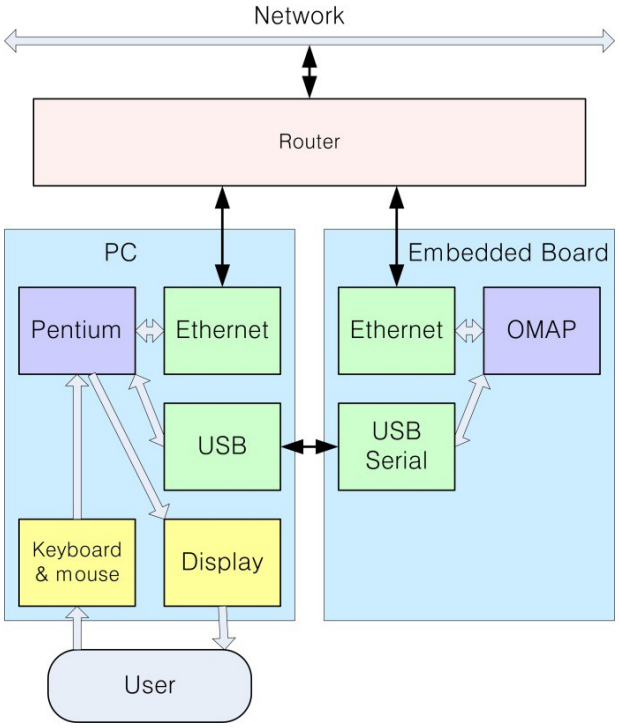


Fig. 1.1 Block Diagram for Lab 1

B. Router

A router contains capability to connect multiple computers to the network with single IP port. Moreover, it has capability of assigning floating IPs to slave computers using DHCP. Hence, for the upper connection from the router to KAIST network, we use an IP for KAIST network, such as 143.248.aaa.bbb. If you are using your own router, public IP will be used. For the lower connections from the router to computer and embedded board, the router can be programmed to assign IPs in a specific range, e.g., from 192.168.100.2 to 192.168.100.254. The router occupies its own IP address, e.g., 192.168.100.1.

Note that the floating IPs should start with 192.168, according to international agreement on IPv4.

C. Embedded board

Embedded board is a single board computer which contains CPU, Memory, and several input/output interfaces. The embedded board which will be used throughout this laboratory named "*Beaglebone Black*" is shown in Fig. 1.2. It contains an AM3358 processor from Texas Instruments, 512 MB RAM, 4GB 8-bit eMMC on-board flash storage, microSD card slot, as well as USB and Ethernet I/O interfaces. Also two 46-pin bus connectors are provided at top and bottom sides of the board, which enables developer to add additional I/O interface boards according to specific requirements. By adding suitable software to this hardware, a development engineer can implement an embedded system for a specific product. For more details, see [1].

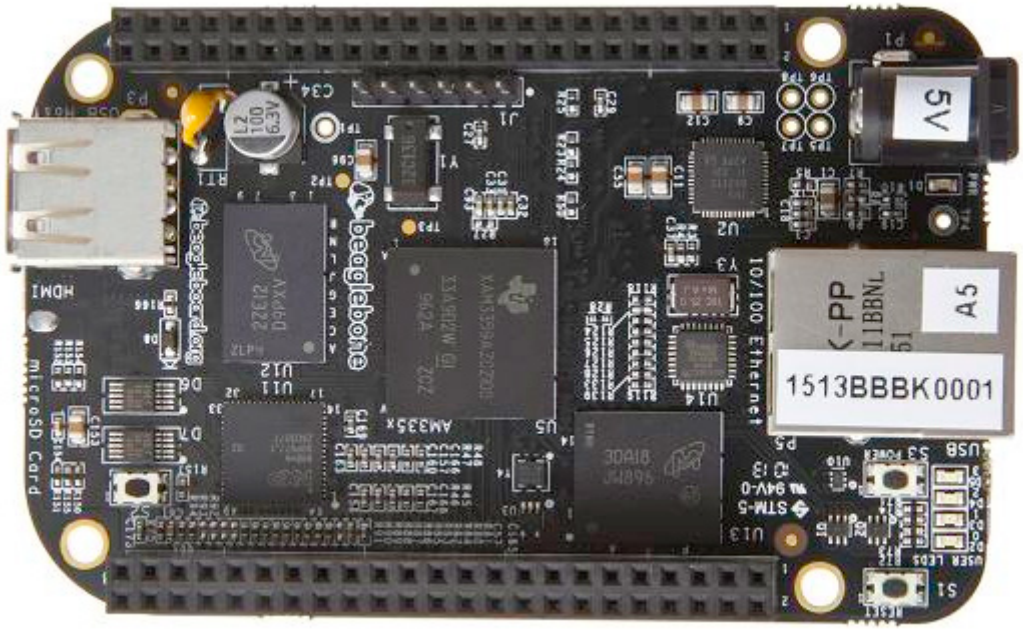


Fig. 1.2 Beaglebone Black embedded board.

A. Ubuntu on PC

Hierarchy

Operating system > Linux > Ubuntu.

What is Operating System? [http://en.wikipedia.org/wiki/Operating_system]

An **operating system** (**OS**) is a set of software that manages computer hardware resources and provides common services for computer programs. The operating system is a vital component of the system software in a computer system. Application programs require an operating system to function.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and will frequently make a system call to an OS function or be interrupted by it. Operating systems can be found on almost any device that contains a computer—from cellular phones and video game consoles to supercomputers and web servers.

Examples of popular modern operating systems include Android, BSD, iOS, Linux, Mac OS X, Microsoft Windows, Windows Phone, and IBM z/OS. All these, except Windows and z/OS, share roots in UNIX.

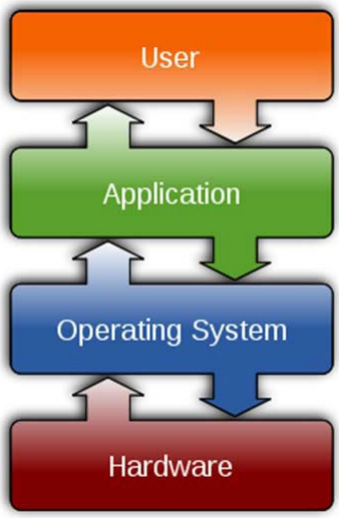


Fig 1.3 Operating System

Common features of OS

- Process management
- Interrupts
- Memory management
- File system
- Device drivers
- Networking (TCP/IP, UDP)
- Security (Process/Memory protection)
- I/O

Ubuntu [http://www.ubuntu.com/ubuntu]

Super-fast, easy to use and free, the Ubuntu Linux operating system powers millions of desktops, netbooks and servers around the world. Ubuntu does everything you need it to. It'll work with your existing PC files, printers, cameras and MP3 players. And it comes with thousands of free apps.

D. Cross development system

Software for embedded system includes all the software required for developing embedded systems. According to a classification of user and developer, the followings are included in the embedded system software.

- User: Software running on the embedded system
- Developer: Software development tools for cross development environment

Software development tools for developer include the following software:

- Editor: Edit source files
- Compiler: Translates into object files
- Linker: Links object files and libraries
- Debugger: Step-by-step execution and status check

Stand-alone system and embedded system are classified as follows.

- Stand-alone system
  - PC and Workstation
  - Self-contained for general purposes
    - Hardware: CPU, Memory, general-purpose user interface, Disk,
    - Software: Operating system, application program, editor, compiler, linker
  - Native compiler
    - The computer in which compiler is running and the computer in which the compiled program is running are the same.
    - Example: Compiler (VC++) in the PC are used to compile a program to run on the same PC.

- Embedded system
  - Self-contained for a specific purpose
    - Limited hardware: CPU, memory, specific I/O, Flash or ROM
    - Software: Embedded OS, application program
  - Cross compiler
    - The computer in which compiler is running and the computer in which the compiled program is running are different.
    - Example: A program to be run on embedded board is compiled by a cross-compiler running on the development PC. Cross-compiled program is stored in the PC, and should be transferred (or downloaded) to the embedded system.

A system with tools for developing software programs to be run on an embedded board (or target board) is called *cross development system*. A compute containing disk, keyboard, display is required, and usually personal computer can be used as a cross-development system. Installing Linux in a PC, and also installing cross development software such as editor, compiler, linker, and debugger are required. Since the compiler

installed is a cross-compiler which generates ARM binary program instead of Pentium binary, the development system is called cross-development system.

Connection of the target board and host computer is basically performed by a serial interface. Suitable software like terminal emulator is required to drive the serial communication.

Cross-compiler [http://en.wikipedia.org/wiki/Cross_compiler]

A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is run. Cross compiler tools are used to generate executables for embedded system or multiple platforms. It is used to compile for a platform upon which it is not feasible to do the compiling, like microcontrollers that don't support an operating system. It has become more common to use this tool for paravirtualization where a system may have one or more platforms in use. Not targeted by this definition are source to source translators, which are often mistakenly called cross compilers.

Uses of cross compilers: The fundamental use of a cross compiler is to separate the build environment from target environment. This is useful in a number of situations:

- Embedded computers where a device has extremely limited resources. For example, a microwave oven will have an extremely small computer to read its touchpad and door sensor, provide output to a digital display and speaker, and to control the machinery for cooking food. This computer will not be powerful enough to run a compiler, a file system, or a development environment. Since debugging and testing may also require more resources than are available on an embedded system, cross-compilation can be less involved and less prone to errors than native compilation.

E. An Example Embedded Application program

Embedded application program is a program to be run on the embedded board under embedded Linux OS. Usually application program can be programmed and compiled using assembler, C, or C++ languages. We use C program in this lab. An example embedded application program is shown in Example 1.1.

Example 1.1 (An example embedded application program helloes with file helloes.c)

```
#include <stdio.h>

void main()
{
        printf("Hello, application program for embedded system!\ n");
}
```

F. System Software

Application program such as HELLO_ES is edited and cross-compiled on the development PC, downloaded (secure copied) to the embedded board, and finally it can be run on the embedded board.

Note that the outputs from printf() are displayed on one borrowed terminal in the PC windows display.

Software	Development PC	Embedded Board
App	-	Helloes
Terminal	Terminal	Terminal with minicom
NFS	NFS Server	NFS client
Secure copy	Scp send	Scp receive
Compiler	Cross-compiler for ARM	-
Editor	gedit	Vi / nano
OS	Linux	Linux

Secure Copy SCP [http://www.hypexr.org/linux_scp_help.php]

scp allows files to be copied to, from, or between different hosts. It uses **ssh** for data transfer and provides the same authentication and same level of security as **ssh**.

Examples

Copy the file "foobar.txt" from a remote host to the local host

$ scp your_username@remotehost.edu:foobar.txt /some/local/directory

Copy the file "foobar.txt" from the local host to a remote host

$ scp foobar.txt your_username@remotehost.edu:/some/remote/directory

Copy the directory "foo" from the local host to a remote host's directory "bar"

$ scp -r foo your_username@remotehost.edu:/some/remote/directory/bar

NFS [http://en.wikipedia.org/wiki/Network_File_System]

Network File System (**NFS**) is a distributed file system protocol originally developed by Sun Microsystems in 1984, allowing a user on a client computer to access files over a network in a manner similar to how local storage is accessed. NFS, like many other protocols, builds on the Open Network Computing Remote Procedure Call (ONC RPC) system. The Network File System is an open standard defined in RFCs, allowing anyone to implement the protocol.

G. Taylor Series

[http://en.wikipedia.org/wiki/Taylor_series]

In mathematics, a **Taylor series** is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point.

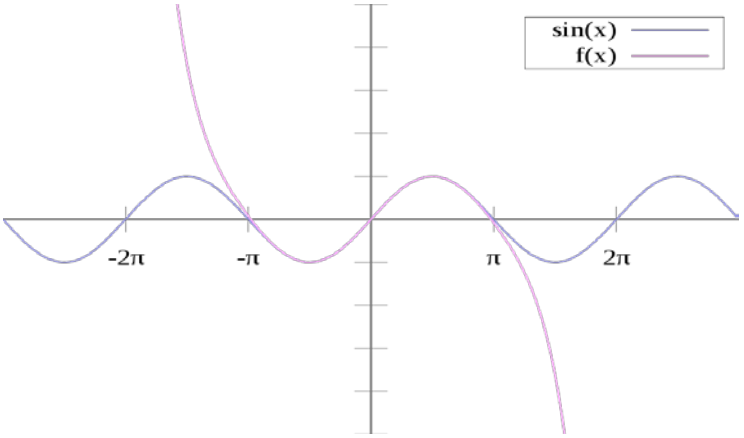
The concept of a Taylor series was formally introduced by the English mathematician Brook Taylor in 1715. If the Taylor series is centered at zero, then that series is also called a **Maclaurin series**, named after the Scottish mathematician Colin Maclaurin, who made extensive use of this special case of Taylor series in the 18th century.

It is common practice to approximate a function by using a finite number of terms of its Taylor series. Taylor's theorem gives quantitative estimates on the error in this approximation. Any finite number of initial terms of the Taylor series of a function is called a Taylor polynomial. The Taylor series of a function is the limit of that function's Taylor polynomials, provided that the limit exists. A function may not be equal to its Taylor series, even if its Taylor series converges at every point. A function that is equal to its Taylor series in an open interval (or a disc in the complex plane) is known as an analytic function.

Pictured on the right is an accurate approximation of $\sin(x)$ around the point $x = 0$. The pink curve is a

polynomial of degree seven:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}.$$



A skeleton application program taylor_ce.c will be provided.

4. Design and Preparation

1. Read and understand (at least try to understand) the Previous Section on Technical backgrounds.
2. Prepare the host PC if you are using your own PC. Partition the hard disk and install *Linux Ubuntu 16.04* in the PC (PC Ubuntu). See Appendix A for required procedures.
3. Read and understand the provided skeleton application program taylor_ce.c. Check for compile and logical errors.

   Taylor_ce.c is composed of two functions:

   Main(): Main program with five steps:

   1) Get user input,
   2) compute sine with math library,
   3) compute sine with Taylor series with calling factorial(n),
   4) compute error, and
   5) print output.

   Factorial(n): Computes factorial of a number.

5. Lab Procedure

Overview: Steps and Hardware

Step 1. Test Ubuntu on PC	PC
Step 2. Install Debian on Beaglebone	PC, SD writer
Step 3. Configure Beaglebone Ubuntu	PC, Beaglebone, Router
Step 4. Install cross compiler on PC and test	PC, Beaglebone, Router
Step 5. NFS in PC and Bone	PC, Beaglebone, Router
Step 6. Test Makefile	PC, Beaglebone, Router
Step 7. Test application with gdb	PC, Beaglebone, Router

Note.
If you are going to use your own PC or Notebook, refer Appendix A for dual boot installation.

Step 1. Test Ubuntu on PC

1.1 Turn on the development PC Ubuntu.

If the computer is configured as multi-boot, select Ubuntu 16.04. If multiple kernel versions exist, you need to specify the most recent kernel version. Actually, use arrow up/down keys to select "4.15.0-136-generic" or similar.

After a while, login window will appear. Type in your user-name and your password. Ubuntu main window will appear.

1.2 Launch and test several useful windows.

In the left column, left click "Home Folder" icon (with file-drawer shape) to launch a file browser. You can navigate disks, directories, and files.

In the left column, left click "Terminal" icon (black box with ">_") to launch terminal window. It is the basic text user interface. You can type in commands via keyboard, and the results are displayed as texts. You can launch multiple terminals to perform multiple jobs. Right click "Terminal" and then select "New Terminal".

In the left column, left click "Text editor" icon (pencil on paper) to launch text editor "gedit". It is a basic interactive text editor to enter program source codes or documents in text.

1.3 Check the kernel version of Ubuntu

In one terminal, type

```
$ uname -r
```

Check that it gives

```
4.15.0-136-generic                // Or similar
```

Hence we check that the kernel version of PC Ubuntu.

*Notice that we use bold courier new font (such as "**`uname -r`**") to denote the content that should be typed, and courier new font (such as "`4.15.0-60-generic`") for responses from the computer.*

1.4 Check network connection.

In one terminal, type

```
$ ifconfig
```

It usually gives:

```
eth1      Link encap:Ethernet  HWaddr b4:2e:99:4e:0b:0d
          inet addr:192.168.0.2 ……
lo        Link encap:Local Loopback
          inet addr:127.0.0.1 ……
```

Check that the "ethN" has correct IP address such as 192.168.0.2 as above.
The "lo" means local loopback, which should show IP of 127.0.0.1.

Test ping to gateway.
In the above case, the gateway is 192.168.0.1 (The last number in IP changed to 1). "-c 5" means 5 trials.

```
$ ping 192.168.0.1 -c 5
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_req=1 ttl=64 time=0.543 ms
……
--- 192.168.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 5997ms
……
```

Test web browser.
In the left column, left click "FireFox Web Browser" icon to launch Internet browser FireFox. You can search Internet using FireFox.

Linux Basics for future reference is given in the Appendix B for your convenience.

Step 2. Install Ubuntu on Beaglebone microSD

2.1 Get Prebuilt image

Visit the following Website:

BeagleBoard.org Latest Firmware Images
http://beagleboard.org/latest-images

Find the following:
Older Debian Images

- *Debian 9.5 2018-10-07 4GB SD IoT* image for BeagleBone, BeagleBone Black, BeagleBone Black Wireless, BeagleBone Blue, SeeedStudio BeagleBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, element14 BeagleBone Black Industrial, Arrow BeagleBone Black Industrial and Mentorel BeagleBone uSomIQ - more info - bmap – sha256sum:52363c654b7a1187656b08c5686af7564c956d6c60c7df5bf4af098f7df395e0

Download
Debian 9.5 2018-10-07 4GB SD IoT
to ~/Downloads.

Check file

```
$ ls -la
-rw-rw-r-- 1 smile smile 514594496  9 월  4 21:37 bone-debian-9.5-iot-armhf-2018-10-07-4gb.img.xz
```

Check sha256sum

```
$ sha256sum bone-debian-9.5-iot-armhf-2018-10-07-4gb.img.xz
52363c654b7a1187656b08c5686af7564c956d6c60c7df5bf4af098f7df395e0  bone-debian-9.5-iot-armhf-2018-10-07-4gb.img.xz
```

Ensure that 5236… equals the value in the last line of Debian 9.5 2018-10-07 4GB SD IoT image:
sha256sum:52363c654b7a1187656b08c5686af7564c956d6c60c7df5bf4af098f7df395e0

Unpack image.xz
Just right-click the file in File Browser. Select "Extract Here".

Check files

```
$ ls -la
-rw-rw-r-- 1 smile smile 3565158400  9 월  4 21:43 bone-debian-9.5-iot-armhf-2018-10-07-4gb.img
-rw-rw-r-- 1 smile smile  514594496  9 월  4 21:37 bone-debian-9.5-iot-armhf-2018-10-07-4gb.img.xz
```

Compare file size of .img and .img.xz.

2.2 Write Debian image to uSD

Insert a blank 16GB microSD card into SD card reader. Connect the SD reader to PC USB port.
Check microSD device name.

```
$ df
/dev/sdb1           75754    5726     70028   8% /media/smile/boot
/dev/sdb2         7660836  739892   6536648  11% /media/smile/rootfs
```

It shows /dev/sdb, which should appear only after you connect the reader. To remove ambiguity, it is safe to remove other USB medias before start.
Note that the device name (such as sdb) may be different from computer to computer (such as sdc, sda).

Unmount sdbN: Be sure to attach N as shown with "df" command.

```
$ umount /dev/sdb1
$ umount /dev/sdb2
```

It should be repeated for every sdbN shown with "df" command. If it's a fresh microSD card, only one would appear.

Run to write image to microSD

```
$ sudo dd if=bone-debian-9.5-iot-armhf-2018-10-07-4gb.img of=/dev/sdb
bs=4M status=progress
[sudo] password for smile:
```

Note: **In of, don't use /dev/sdb1, but use /dev/sdb**
Wait……: It requires time to copy ~4GB with speed ~4.5MB/s.
Sometimes, the shown speed is extremely high (like ~100MB/s). This is mostly due to buffering. ***Please*** wait until the execution completely ends.

Remove your flash media when the command completes (you may need to wait a few extra seconds for it to finish).

2.3 Prepare PC for connection using minicom

Install minicom

```
$ sudo apt-get install minicom
```

Check device name for connection

```
$ ls -la /dev/ttyACM*
crw-rw---- 1 root dialout 188, 0  9 월  5 21:41 /dev/ttyACM0
```

Note that this device name appears only after connecting Beaglebone to PC. Now proceed even if this command doesn't show ttyACM*

Configure minicom ("-s" means setup)

```
$ sudo minicom -s
```

The following window will appear on the terminal.

```
+-----[configuration]---------+
| Filenames and paths         |
| File transfer protocols     |
| Serial port setup           |
| Modem and dialing           |
| Screen and keyboard         |
| Save setup as dfl           |
| Save setup as..             |
```

```
          | Exit                        |
          | Exit from Minicom           |
          +-----------------------------+
```

Use arrow keys to highlight 'Serial port setup', and click 'Enter' key.
The following sub-window will popped up:

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyACM0                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
|
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : Yes                                       |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

Type 'A', and then set the Serial Device as '/dev/ttyACM0'. Type 'Enter' key.
Type 'E' and set speed as "115200 8N1".
Type 'F' several times to set the Hardware Flow Control to 'No'.
Type 'G' several times to set the Software Flow Control' to 'No'.
Press 'Enter' key to exit the sub-window.

Use arrow keys to highlight "Save setup as dfl". Press 'Enter' key.
"Configuration saved" sub-window will appear.

Use arrow keys to highlight "Exit", and then press 'Enter' key.

Note that this setup needs to be done only once after PC Ubuntu install.

2.4 Prepare Beaglebone

Disconnect USB cable of SD reader from PC USB.

Remove microSD from SD reader, and then insert it to Beaglebone.

2.5 Start Beaglebone

Connect power adaptor (5V 2A), and Ethernet to Beaglebone, and power on the power adaptor.
After power on, connect USB cable from Beaglebone to PC.

Note. It seems that power adaptor is required for sufficient current (2A). USB cable only cannot provide sufficient current.

Check that the blue LED near +5V connector is on.
Then check that four LEDs (near Ethernet connector) are changing (somewhat randomly).
Eventually, check that the outermost LED among four LEDs is blinking.

Connect to Beaglebone console from PC using minicom

```
$ sudo minicom -w
```

Boot messages will appear, and then shows login prompt as follows:

```
Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Nov 15 2018, 20:18:47.
Port /dev/ttyACM0, 12:03:15

Press CTRL-A Z for help on special keys


Debian GNU/Linux 9 beaglebone ttyGS0

BeagleBoard.org Debian Image 2018-10-07

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

beaglebone login:
```

Login as user "debian" with password "temppwd".

```
debian@beaglebone:~$
```

Does this prompt appear?

Check kernel version

```
# uname -r
4.14.71-ti-r80
```

Debian 9.5 2018-10-07 4GB SD IoT is successfully installed in uSD!

Notice that we use "#" for prompts in Beaglebone Debian, "$" for prompts in PC Ubuntu.

Note.

PC Ubuntu 16.04 kernel	*4.15.0-136-generic*
Beaglebone Debian 9.5 kernel	*4.14.71-ti-r80*

Step 3. Configure Beaglebone Debian

3.1 Add superuser with passwd

Add superuser: User with all abilities.

```
# sudo passwd root
[sudo] password for debian:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Become superuser

```
# su
Password:           // Enter Root password
```

Check who is logged in

```
# whoami
root
```

You can see the directory "/root" by "ls -la /"

```
# ls -la /root
total 24
drwx------  4 root root 4096 Oct  7  2018 .
drwxr-xr-x 21 root root 4096 Oct  7  2018 ..
-rw-r--r--  1 root root  570 Jan 31  2010 .bashrc
drwxr-xr-x  2 root root 4096 Oct  7  2018 .c9
drwxr-xr-x  4 root root 4096 Oct  7  2018 .node-red
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
```

Logout and then login as root (with new root password).

3.2 Add user - yourself

Add user (for later user login)

```
# su
# adduser njahn
Adding user `njahn' ...
Adding new group `njahn' (1001) ...
Adding new user `njahn' (1001) with group 'njahn' ...
Creating home directory `/home/njahn' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for njahn
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
```

Check /home directory

```
# ls /home
debian  njahn
```

Two users named "debian" and "njahn" does exist. Become superuser

```
# su
Password:           // Enter Root password
```

Edit sudoers to use "sudo" via visudo editor

```
# visudo -f /etc/sudoers
```

Insert "[user] ALL=(ALL:ALL) ALL" below "root ALL=(ALL:ALL) ALL"

```
...
# User privilege specification
root    ALL=(ALL:ALL) ALL
```

```
njahn ALL=(ALL:ALL) ALL
...
```

3.3 Check network configuration.

Check network configuration.

```
# ifconfig
eth0: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
  inet 192.168.0.4 netmask 255.255.255.0 broadcast 192.168.0.255
  inet6 fe80::218:31ff:fe8e:2d1d prefixlen 64 scopeid 0x20<link>
  ether 00:18:31:8e:2d:1d  txqueuelen 1000  (Ethernet)…
```

Now you can login to BeagleBone in other terminal window:

```
$ ssh njahn@192.168.0.4
The authenticity of host '192.168.0.4 (192.168.0.4)' can't be established.
ECDSA key fingerprint is SHA256:JPmS6I8wOd/lJ7nLltOAxc5FTcYYhoXhUlHQPeJpkDk.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.4' (ECDSA) to the list of known hosts.
njahn@192.168.0.4's password:            // Enter user password
```

Now you are double-logged in to Debian: Two users (actually identical) are logged in to Debian.
You can issue commands to each terminal.

Note.
The following message may appear:

```
$ ssh njahn@192.168.0.4
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:UkqvWqxfVh+zaYQW966WVAZbtPNu+LQWYtbS5fjH4W4.
Please contact your system administrator.
Add correct host key in /home/smile/.ssh/known_hosts to get rid of this
message.
Offending ECDSA key in /home/smile/.ssh/known_hosts:2 remove with:
  ssh-keygen -f "/home/smile/.ssh/known_hosts" -R 192.168.0.4
ECDSA host key for 192.168.0.4 has changed and you have requested strict
checking.
Host key verification failed.
```

In this case, issue command on PC window:

```
$ ssh-keygen -f "/home/njahn/.ssh/known_hosts" -R 192.168.0.4
```

Then ssh again.

3.4 Update packages

Ubuntu software packages are updated frequently. You need to catch up.

```
# sudo apt-get update
```

It takes a while.....

Check disk usage

```
# df
Filesystem     1K-blocks   Used Available Use% Mounted on
……
/dev/mmcblk0p1   3357264 2015488   1151520  64% /
```

64% of microSD is used. Enough spaces are left.

3.9 Turn off Bone

In order to save microSD content: do

```
# sudo shutdown -h now
```

Important: Do this every time to turn off the computer!

Step 4. Install Cross-compiler and Test

4.1 Check existence of cross compiler for Beaglebone in PC Ubuntu

Check existence of cross-compiler

```
$ arm-linux-gnueabihf-gcc --version
arm-linux-gnueabihf-gcc (Ubuntu/Linaro 5.4.0-6ubuntu1~16.04.9) 5.4.0
20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

If none, install cross-gcc

```
$ sudo apt-get install gcc-arm-linux-gnueabihf
```

If your linux is a fresh ubuntu 16.04, it will automatically install 5.4.0 (which is the latest version available for 16.04)

Check the location of cross-gcc

```
$ sudo find / -name arm-linux-gnueabihf-gcc -print
/usr/bin/arm-linux-gnueabihf-gcc
```

Check version of native gcc on PC

```
$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.11) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

If none, install gcc

```
$ sudo apt-get install gcc
```

If your linux is a fresh ubuntu 16.04, it will automatically install 5.4.0 (which is the latest version available for 16.04)

We get the same gcc version 5.4.0: Good!

Note.

If you have gcc/arm-gcc installed already, you'll have to manage multiple gcc versions. Try 'update-alternatives' command.

4.2 Test cross-compiler on PC

Make a working directory

```
$ mkdir -p ~/Embedded/lab1/a_testgcc
$ cd ~/Embedded/lab1/a_testgcc
```

Edit helloes.c using gedit (or vi if you prefer)

```
$ gedit helloes.c   OR
$ vi helloes.c
```

Type in as follows:

```
#include <stdio.h>

void main()
{
       printf("Hello, application for embedded system!\n");
}
```

Native compile

```
$ gcc -o helloes_nc helloes.c
```

Check output of helloes_nc

```
$ ls
helloes.c helloes_nc
```

Run (of course, on PC)

```
$ ./helloes_nc
Hello, application for embedded system!
```

Now, cross compile the same program.

```
$ arm-linux-gnueabihf-gcc -o helloes_cc helloes.c
```

NOTE: The name of cross-compiler is "arm-linux-gnueabihf-gcc", NOT "arm-linux-gnueabi-gcc" nor "arm-angstrom-linux-gnueabi-gcc".

Check output of helloes_cc

```
$ ls
helloes.c  helloes_cc  helloes_nc
```

Try to run on PC

```
$ ./helloes_cc
bash: ./helloes_cc: cannot execute binary file: Exec format error
```

4.3 Check if scp from PC works.

Test scp from PC to Bone root

```
$ scp helloes_cc root@192.168.0.4:/root
root@192.168.0.4's password:
helloes_cc                                    100% 8248     8.1KB/s   00:00
```

Test scp from PC to Bone debian

```
$ scp helloes_cc njahn@192.168.0.4:/home/njahn       OR
$ scp helloes_cc njahn@192.168.0.4:~
njahn@192.168.0.4's password:
helloes_cc                                    100% 8248     8.1KB/s   00:00
```

Sometimes, scp does not work due to security and firewall. You can proceed anyway.

4.4 Run on Bone

Check if copied to Beaglebone.
Log in to Beaglebone.
Check

```
# ls /home/njahn
helloes_cc
```

Run

```
# cd /home/njahn
# ./helloes_cc
Hello, application for embedded system!
```

Step 5. NFS in Bone-Ubuntu & PC-Ubuntu

A. Set up NFS server on PC Ubuntu.

5.1 Install NFS server on PC

Type

```
$ sudo apt-get install nfs-kernel-server
```

5.2 Configure NFS

Check the resultant IP. For example,
PC(smile): eth1 192.168.0.2
Bone(njahn): eth0 192.168.0.4

Edit /etc/exports.

```
$ sudo vi /etc/exports
```

Append the following:

```
# nfs from Beaglebone - Embedded
# PC-Directory Allowed-IP-to-nfs-PC (Options)
/home/smile 192.168.0.4(rw,sync,no_root_squash,no_subtree_check)
```

Whenever we modify /etc/exports, we must run

```
$ sudo exportfs -a
```

afterwards to make the changes effective.

5.3 Start PC NFS server

To start the NFS server, you can run the following command at a terminal prompt:

```
$ sudo /etc/init.d/nfs-kernel-server start
```

[ok] Starting nfs-kernel-server (via systemctl): nfs-kernel-server.service.

Or, use service command

```
$ sudo service nfs-kernel-server restart
```

which would finish without any output message.

Check if nfs started.

```
$ ps -aux | grep nfsd
```

You must be able to see multiple nfsd process by the root.

B. Set NFS client on Beaglebone Ubuntu

5.4 Install nfs client in Bone

```
# sudo apt-get install nfs-common
```

5.5 Make the mount point

Make the mount point

```
# mkdir ~/nfs_client
```

There should be no files or subdirectories in the directory.

5.6 Start Beaglebone nfs client

```
# cd ~
# sudo mount 192.168.0.2:/home/smile/Embedded ~/nfs_client
```

NOTE. If no response, ping Beaglebone from PC may help.

```
$ ping 192.168.0.4
```

5.7 For later use, edit ~/bone_nfs_client.sh using vi.

```
# vi ~/bone_nfs_client.sh
```

Type in as follows:

```
#! /bin/bash
# Mount nfs_client in BeagleBone
mount 192.168.0.2:/home/smile/Embedded ~/nfs_client
```

Later, to start nfs client, you can simply type

```
# sudo bash ~/bone_nfs_client.sh
```

Note.

If your router assigns different IPs every time, it's strongly recommended to set the fixed DHCP address, or just mount manually with dynamic address every time.

5.8 Go to nfs directory

```
# cd ~/nfs_client
# ls
lab1
```

You can see PC-Ubuntu directory ~/Embedded!

5.9 Run helloes_cc using nfs

Go to the directory for hello_es.

```
# cd ~/nfs_client/lab1/a_testgcc
# ls
helloes.c helloes_nc helloes_cc
```

You can run "helloes_cc" without scp.

```
# ./helloes_cc
Hello, application program for embedded system!
```

Step 6. Test Makefile

Alternatively you can use Makefile, which does not require typing compile commands many times.

6.1 Make a working directory on PC.

```
$ mkdir -p ~/Embedded/lab1/b_makegcc
$ cd ~/Embedded/lab1/b_makegcc
```

Copy source file

```
$ cp ../a_testgcc/helloes.c .
```

Note the third word of ".", which means the current directory.

6.2 Edit Makefile

Edit "Makefile" as follows:

```
#      Embedded Systems Lab1 makegcc Makefile

# Source file: helloes.c

# Do everything
all:   nc cc

# Native compile
nc:
       gcc -o helloes helloes.c

# Cross compile for Bone Ubuntu
cc:
```

```
	arm-linux-gnueabihf-gcc -o helloes.x helloes.c

# Clean up
clean:
	rm -f helloes helloes.x
```

6.3 Use Makefile

Native compile

```
$ make nc
```

Cross compile

```
$ make cc
```

Compile all

```
$ make all
```

Clear all previously compiled files.

```
$ make clean
```

Now you can run helloes.x on Beaglebone using nfs.

Step 7. Debug Application on PC

Program: Sine computation using Taylor series

Objective

Edit, compile, run, and debug the program performing the following:
Compute sine 30 degree using Taylor series up to n terms (n=1, 2, …, 5)
and also compute errors (math library and Taylor series with n terms).

Working directory

Make a working directory

```
$ cd lab1
$ mkdir c_taylor
$ cd c_taylor
```

We are going use three source files

taylor_ce.c	Given source program with many compile errors
taylor_re.c	Modified source program without compile error, but output error
taylor_ok.c	Debugged final source program with good output.

7.1 Compile taylor_ce

Initial program taylor_ce.c
Composed of factorial function and main function.

Edit Makefile

```
#	Embedded Systems Lab1 c_Taylor Makefile

# Source file: taylor_ce.c, taylor_re.c, taylor_ok.c

# Do everything
all:	nce nre nok cok
```

```
# Native compile
# Note: Be sure to add '-lm' to include math library.
nce:
	gcc -o taylor_ce_pc taylor_ce.c -lm

nre:
	gcc -o taylor_re_pc taylor_re.c -lm

nok:
	gcc -g -O0 -o taylor_ok_pc taylor_ok.c -lm

# Cross compile
cok:
	 arm-linux-gnueabihf-gcc -o taylor_ok taylor_ok.c -lm
```

Make

```
$ make nce
```

You can see many compile errors.
File taylor_ce_pc cannot be made.

7.2 Edit, run, and debug taylor_re.c

Copy taylo_ce.c to taylor_re.c.
Edit taylor_re.c to eliminate compile errors (many times if necessary).
Now "make nre" should return no error.

Run taylor_re_pc

```
$ ./taylor_re_pc
Enter angle in degrees: 30
sin(30)= -0.988032 using Math library.
sin(30) using Taylor series:
1 terms: sin(30)= 1.4013e-45, error= 0.988032
2 terms: sin(30)= 0, error= 0
3 terms: sin(30)= 5.88088e-39, error= 5.87965e-39
4 terms: sin(30)= 0, error= 0
5 terms: sin(30)= 5.6729e+14, error= 5.67305e+14
```

Output of sin(30) gives wrong result. Why?

Debug 1.
After some browsing, you can find that the argument of sine should be in radian, not degree!
In taylor_re.c, you need to change degree to radian, for example:

```
double pi = 3.14;
double dtr = pi/180.;
double inrad;
...
inrad = indeg*dtr;  // after scanf("%d", &indeg);
```

Test run

```
$ ./taylor_re_pc
Enter angle in degrees: 30
sin(30)= 0.49977 using Math library.
sin(30) using Taylor series:
1 terms: sin(30)= 2.61667, error= 5.88083e-39
```

```
2 terms: sin(30)= 0.0955529, error= 0
3 terms: sin(30)= 0.000981368, error= 5.87965e-39
4 terms: sin(30)= 4.26627e-06, error= 0
5 terms: sin(30)= 8.12534e-09, error= 1.11974e-11
```

Still a little error in sin(30) with Math library.

Debug 2.

In taylor_re.c, you set exact value of pi, for example:

```
double pi = atan(1.)*4.;
```

Test Run

```
$ ./taylor_re_pc
```

Enter angle in degrees: 30
sin(30)= 0.5 using Math library.
sin(30) using Taylor series:
1 terms: sin(30)= 2.61799, error= 5.88083e-39
2 terms: sin(30)= 0.0956984, error= 0
3 terms: sin(30)= 0.00098386, error= 5.87965e-39
4 terms: sin(30)= 4.28144e-06, error= 0
5 terms: sin(30)= 8.15126e-09, error= -5.80303e-28

Now math library gives exact result, but Taylor series doesn't.

7.3 Use debugger with taylor_ok.c

Now it is the time to use debugger.
Copy taylor_re.c to taylor_ok.c.
In makefile, we used "**gcc -g -O0":**

"-g" is an option include debug information.
"-O0" is recommended: No optimization by compiler.

After "make nok", you can check the difference in file size:

```
$ ls -la
……
-rwxrwxr-x 1 smile smile  8808  9 월  5 21:52 taylor_re_pc
-rwxrwxr-x 1 smile smile 10400  9 월  5 22:01 taylor_ok_pc
```

Gdb 1.

Now run taylor_ok_pc with gdb

```
$ gdb taylor_ok_pc
```

Gdb command "l" lists a part of the source program

```
(gdb) l
8
9
10      #define TERMS 5
11
12      int factorial(int m)
13      {
14              int facto = 1;
```

```
15            for (int i=2; i<=m; ++i)
16                  facto = facto*i;
17
```

After several "l" commands, you can find the line to debug:

```
52                          facto = factorial(pwr);
53                          // Compute sine by Taylor series
54                          s[i-1] += angpwr/facto;
55                  }
```

Gdb command "b 54" sets a breakpoint at line 54 (At s[i-1] += …)

```
(gdb) b 54
Breakpoint 1 at 0x400850: file taylor_ok.c, line 54.
```

Run with Gdb:

```
(gdb) r
Starting program: /home/smile/Embedded/lab1/c_taylor/taylor_ok_pc
Enter angle in degrees: 30
sin(30)= 0.5 using Math library.
```

In runs until the end of program!(If you are lucky(?), it will stop.)

This means that loop for I is not executed. Why?

You may find line 45 something wrong.

Correct line 45 from

```
for (i=1; j<=n; ++i) {
```

to

```
for (i=1; i<=n; ++i) {
```

Gdb 2.

There is an index error: Correct line 54 from

```
s[i-1] += angpwr/facto;
```

to

```
s[n-1] += angpwr/facto;
```

Gdb 3.

s[n] term is not initialized: Add clear s[n] term in line 44:

```
s[n-1] = 0.0;       // Clear sum
```

Gdb more…

Correct this error and other remaining bugs using gdb.

The correct result is as follows:

```
$ ./taylor_ok_pc
Enter angle in degrees: 30
sin(30)= 0.5 using Math library.
sin(30) using Taylor series:
1 terms: sin(30)= 0.523599, error= 0.0235988
2 terms: sin(30)= 0.499674, error= -0.000325821
3 terms: sin(30)= 0.500002, error= 2.13259e-06
4 terms: sin(30)= 0.5, error= -8.13098e-09
5 terms: sin(30)= 0.5, error= 2.02799e-11
```

You can see that Taylor series of sine with 5 terms gives value with error less than 10^(-10)!

7.4 Cross compile and run on Beaglebone

Now you can cross-compile with

```
$ make cok
```

Then you can run taylor_ok on Beaglebone, and check the result.

If it is OK with multiple angles, you are ready to take a demo video.

6. Demonstration

You have to take a short video, showing cross-compiled taylor_ok at the step 7.4 works well in the Beaglebone board.

Video may include at least 1 correct sin(30) result (with small errors), and must include validate indication that taylor_ok is ran on the Beaglebone board (like the login-prompt, or ID/device-name at the console).

Make sure the video is no longer than 1 minute.

7. Report

Upload it with the demo video and the taylor_ok.c code, via KLMS.

Each student should prepare his own report.

The report contains:

- Purpose
- Experiment sequence
- Experimental results
- Discussion
- References

Discussion for the following question should be included in the report.

8. References

[1] Getting Started with Beaglebone and Beaglebone Black, http://www.beagleboard.org/Getting%20Started

[2] Beaglebone Rev. A5. System Reference Manual, http://circuitco.com/support/index.php?title=Beaglebone#Rev_A5.

NOTE. The version of Beaglebone boards in the Lab is Rev. A5. Be sure to download the correct version.

[3] "Embedded Linux Primer", C. Hallinan, Prentice Hall.

Appendices for Lab1

A. Ubuntu 16.04 install on Windows PC

1. Plan disk partitions

Basically you need at least five partitions: two for Windows and three for Linux.

Partitions for Windows suggested: Two partitions.

- Partition C: For system (Windows and System application software such as Office)
- Partition D: For user program.

In this case, it is easy to back-up: You can only back up partition D.

Partitions for Linux suggested: Three partitions.

- Partition root (/): For system (Linux and System application software). At least 30 GB.
- Partition swap: For swap area. Twice the size of the main memory.
- Partition home (/home): For user program. At least 30 GB.

A. Partition for Windows

You can use Disk Management in Windows Control panel as follows:

Windows Start – Control Panel – System & Security – Management Tools – Computer Management - Storage – Disk Management

Allocate partitions as follows:

- C: partition for Windows System
- D: partition for Windows User

Leave disk space for Linux partitions (minimum 112 GB recommended).

B. Partition for Linux

Will be performed when installing Ubuntu.

2. Download Ubuntu 16.04: 64-bit version

Use Windows PC.
Visit Ubuntu download page [http://www.ubuntu.com/download/desktop].
Click "Download Ubuntu", and then click "Download".

Save the file "Ubuntu-16.04.1-desktop-amd64.iso" or similar (about 1.4 GB) to your hard disk.

3. Burn Ubuntu DVD or USB from this image.

Use DVD or USB to install Ubuntu.

DVD: http://www.ubuntu.com/download/desktop/burn-a-dvd-on-windows

USB: http://www.ubuntu.com/download/desktop/create-a-usb-stick-on-windows

I found that USB is much more convenient.

4. Install Ubuntu16.04 for dual boot

Install Ubuntu via DVD or USB.
Refer http://www.ubuntu.com/download/desktop/install-ubuntu-desktop

Additional Notes on Installation.
Step 1. You may need to change boot sequence of your BIOS. After power on PC, press the F12 to bring up the Boot menu, and change boot priority adequately.

After a while, Ubuntu logo will be displayed at the center of display, and then
"Install – Welcome" window appears.
In the left column, select "English"
In the right column, select "Install Ubuntu".

Step 3. Allocate Drive Space.
IMPORTANT: In the 'Install – Installation Type" window, Select *"Something Else"* in order to make Linux partitions.
Click "Add", and create a new partition. You should make three partitions as follows:

/dev/sdb2	ext4 file /	32 GB or more	
/dev/sdb3	swap		Twice of main memory size. 16 GB
/dev/sdb4	ext4 file	/home	64 GB or more.

Appendix B. Linux Basics

- **1. Getting Started**
- # login: *username* or *root*
- # password: *user_password* or *root_password*
- # logout
- # shutdown –h now ; Shutdown the computer

- **2. Basic commands**
- # date ; Display date and time

Ex) Wed Sep 1 12:12:29 EDT 2012

- # who ; List users currently logged in
- # man command ; Display manual of the command
- # pwd ; Print the complete pathname of the current directory
- # cd /usr/src/linux ; Change directory to /usr/src/linux

- **3. File manipulation**
- # ls [-la] ; List files in the current directory
- # cat filename ; Prints the file with filename
- # cp source_file dest_file ; Copy source_file to dest_file

Ex) # cp file /dev/ttyS0 ; Copy file to COM1

- # rm junk_file ; Remove junk_file
- # mv old_file to new_file ; Rename the old_file to new_file

- **4. Manipulating directories**
- # mkdir new_dir ; Make a new_dir directory
- # rmdir old_dir ; Delete the old_dir directory
- # mv old_dir new_dir ; Rename old_dir directory to new_dir
- # cd new_dir ; Change directory to new_dir

Ex) # cd .. ; Change to upper directory
Ex) # cd / ; Change to root directory

- **5. System inquiries**
- # ps ; List active processes with process_id
- # kill -9 process_id ; Kill the process with process_id
- # du ; Disk usage of the current directory
- # df ; Display file system usage
- # su ; Become the superuser (root)

Ex) # password: *root_password*

- # exit ; Become a normal user

- **6. Editing files with vi**
- # vi file.c ; Visual edit file.c
- # Ctrl-F, Ctrl-B : Move forward/backward a full screen
- # space, backspace, return ; Move cursor right/left/next_line
- # i… esc ; Insert characters before cursor (until escape)
- # a… esc ; Insert characters after cursor (until escape)
- # o… esc ; Insert line by line after the current line
- # O… esc ; Insert line by line before the current line
- # x ; Delete the current character
- # dw ; Delete the current word
- # dd ; Delete the current line
- # r file ; Read the file
- # s/old/new/g ; Substitute old to new globally
- # :q ; Quit without saving
- # :wq ; Quit after saving

Alternatively, you may use "nano" editor, which is WYSIWYG.

- **7. Compile and run**
- # mkdir /embedded/test
- # cd /embedded/test
- # vi hello.c
- # gcc –o hello hello.c ; Cross-compile and link the program to produce hello.
- # ./hello ; Run hello
- Hello, Embedded board! ; Output: print a string on the console

- **8. Make command**
- # vi Makefile

main.o average.o: defs.h
average: main.o average.o
gcc –o average main.o average.o -lm

- # vi defs.h
- # vi main.c
- # vi average.c
- # make average.o ; Compile average.c to average.o
- # make average ; Compile main.c to main.o
- ; Link main.o, average.o, and lib into average
- ./average ; Run average

FIN