Lab 1. Cross Development Environment

I. Purpose

Suitable Linux software cross development system based on a PC will be constructed and tested. This cross development system will be used for developing a specific embedded system. We test various aspects: PC Ubuntu on PC, Debian on Beaglebone, cross-compile, NFS, module build, and reaction time measurement application.

II. Problem Statement

Problem 1. Cross Development Environment and an Application.
Construct a cross-development system on a PC for application and module programs development, and implement a reaction timer application program.

We start from the simplest and perform step-by-step improvements.

Problem 1A. An Example of Embedded Application Program.
Construct a cross-development system on a Lab PC (or your own notebook PC). Edit and cross compile an example embedded application program. Download to the embedded system or use NFS, and then run.

Problem 1B. An Example of Embedded Module Program.
Construct a module program development environment on a PC. Edit and cross compile an example embedded module program. Download to the embedded system or use NFS, and then run.

Problem 1C. Design of Reaction Timer Application.
Implement a program which measures your response time from message display to key input.

i. Computer displays a message at random time (2 to 5 s) with a lowercase alphabet 'f' or 'j' (random):
   "Type 'f' key without Enter: " or
   "Type 'j' key without Enter: "
ii. You type the designated key without Enter key as soon as possible.
iii. Computer computes your reaction time from i to ii in ms, and compares the required key and the response key.
iv. Computer reports:
   "Correct answer 'f' in ab.cde ms." or
   "Wrong answer 'j' in ab.cde ms."

Of course, quotation marks do not appear on terminal.
Test on PC to debug, and then run on Beaglebone.

III. Technical Backgrounds

FIRST WEEK
A1. Hardware Components of Cross Development Environment

1. Hardware connection

A block diagram describing hardware connection is shown in Fig. 1.1.

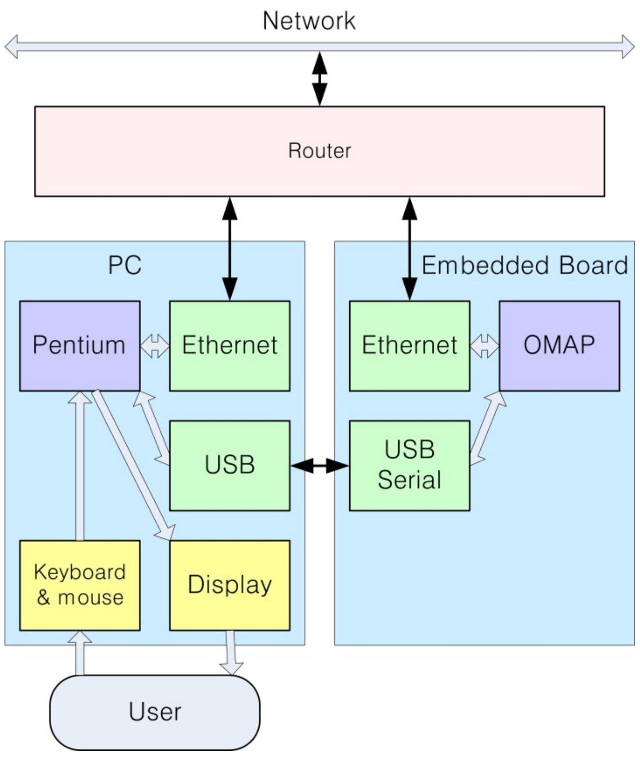

Fig. 1.1. Block Diagram with hardware connection for Lab 1.

2. Router

In this Lab, PC and Embedded board are connected via router as shown in Fig. 1.1. A router contains capability to connect multiple computers to the network with single IP (Internet Protocol) port. Moreover, it has capability of assigning floating IPs to slave computers using DHCP (Dynamic Host Configuration Protocol).

Upper connection

For the upper connection from the router to KAIST network, we use a registered static IP with four bytes for KAIST network, such as 143.248.aaa.bbb (both aaa and bbb bytes ranging from 1 to 255, according to IP v4 (Internet Protocol version 4)). Note that this is a human-readable form of four bytes IP address. It is represented with 4 bytes (or 32 bits) data inside computer. Note that the registered KAIST network IP should start with 143.248.

Lower connections

For the lower connections from the router to computer and embedded board, the router can be programmed to assign personal IPs in a specific range, e.g., from 192.168.100.2 to 192.168.100.254. The third data '100' can be any number between 0 and 255. It is set by router configuration. The router occupies its own personal IP address, e.g., 192.168.100.1. Connected computers will get IP from 192.168.100.2 to 192.168.100.244. The IP X.X.X.255 is assigned as broadcast IP. Note that the floating personal IPs should start with 192.168, according to international agreement on IPv4.

3. 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*" is shown in Fig. 1.2. Size? The same as a credit-card. Capability? Similar to Android phone.

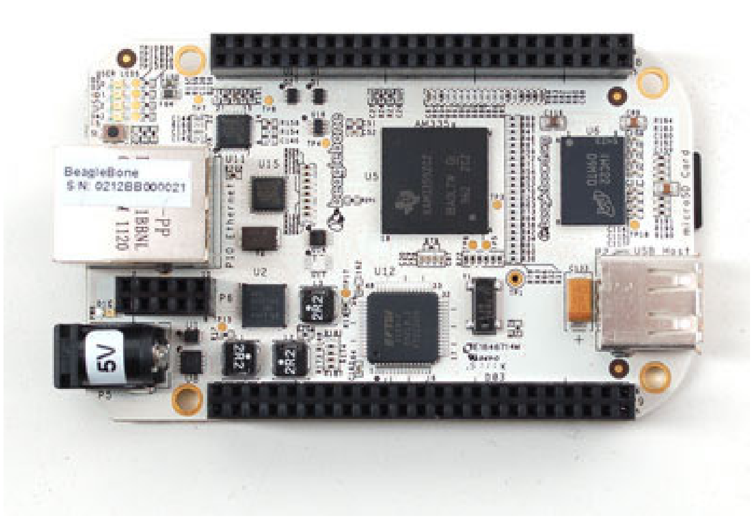

Fig. 1.2. Beaglebone embedded board.

Components

It contains an AM3359 processor from Texas Instruments (biggest chip at the center), 256 MB RAM, 32 KB EEPROM (inside AM3359!), and I/O interfaces with microSD card slot (right bottom), as well as USB host (right lower side), USB client (left upper bottom side), and wired Ethernet (left center). Also two 46-pin (2x23 configuration) connectors P8 and P9 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 Manual [1]. Also visit http://beagleboard.org/getting-started.

Power

The Beaglebone is powered either by USB (5 V, 1 A) or Power adapter (5 V, 2 A). Desktop computers usually provide USB power of +5V up to 1A, which is sufficient to driver the Beaglebone. However, some notebook computers provide USB power up to 500 mA, which is not sufficient for Beaglebone. All peripherals on AM3359 use 3.3 V, and core of AM3359 uses 1.8 V.

CAUTION. WHILE OPERATION, DO NOT TOUCH THE MICROSD! The microSD will be ejected by spring with power on, and the board and microSD can be damaged!

A2. Software Components

4. 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. See Fig. 1.3.

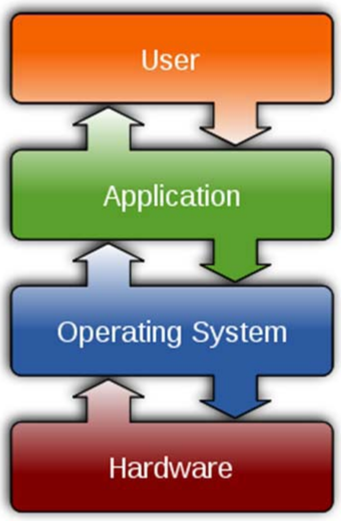

Fig 1.3 Operating System.

Functions of OS

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.

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.

5. Debian on Beaglebone

Check the Web page:
BeagleBoard.org Latest Firmware Images
http://beagleboard.org/latest-images

You can see that
Recommended Debian Images
Wheezy for Beaglebone…
Debian 7.9. …
We use this Debian Wheezy 7.9 OS on Beaglebone.

6. Cross Development System

Cross development system is a software development system for developing software which is to be executed in the target embedded board. As a suitable hardware, a general-purpose computer is necessary including a keyboard and display for user input/output, and also a disk for program storage. Since vast amount of PCs (Personal Computers) are widely available, a PC can be utilized as a development system (sometime called a host computer or a host development system). Suitable cross-development development software should be installed and utilized, which includes an editor, a compiler, a linker, and a debugger. The compiler here does not generate Pentium binary codes, but generates ARM Cortex binary codes. Hence this compiler is called a cross compiler, and the system is called a cross-development system. Software for development system is often called development tools.

(1) Editor
In order to enter source program, we need a suitable editor.

(2) Assembler
An assembler performs a conversion function from an assembly language program to a machine language, the latter is often called an object code. Good news: We do not require assembly language in this lab.

(3) C compiler
A C compiler performs a conversion function from a C language program to a machine language program (or an object code). The gcc compiler has good performance, and we are going to use it. We use it as the cross-compiler.

(4) Linker

Compiled user program composed of several files should be linked together along with library, to produce an executable code (which can be run on the target system, not on PC).

Overall flow of the development process is illustrated in Fig. 1.4.

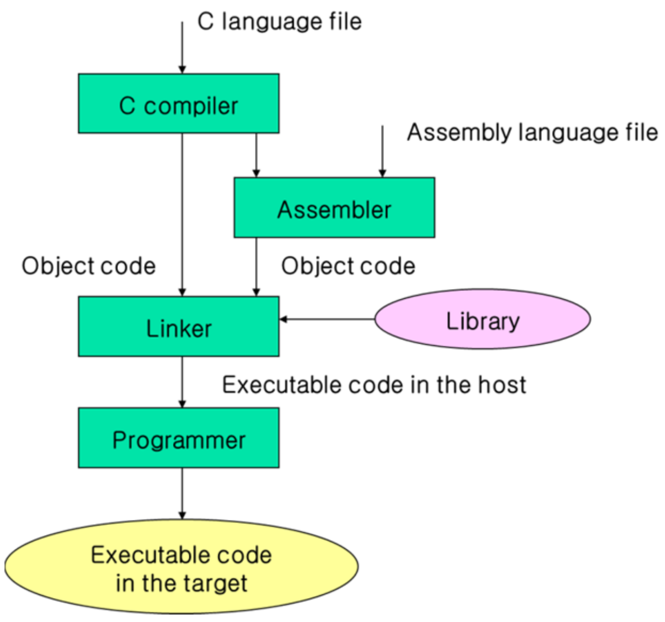

Fig. 1.4. Program Cross-Development Process.

7. Application program

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. A simple C program hello_es.c is shown in Example 1A.

Example 1A. A simple example application program.

```
#include <stdio.h>

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

This program prints a message using printf() function, which is a C library function to print variables, string, etc, defined in stdio.h. The printf() function is located inside C library.

Note that we can use the same program for both PC and Beaglebone, if the program is hardware independent. This program can be native-compiled on the development PC, and it can be run on PC. Also this program can be cross-compiled on the development PC, downloaded to the embedded board, and finally it can be run on the embedded board.

8. System Software

Application program such as hello_es.c 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	Hello_es_pc	Hello_es
Terminal	Terminal	Terminal with screen or minicom command
NFS	NFS Server	NFS client
Secure copy	Scp send	Scp receive
Compiler	Cross-compiler for ARM	-
Editor	gedit	-
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
```

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
```

Note. Some scp commands does not work due to security reasons: Firewall may prevent scp action.

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.

B. Module Build

9. Kernel

Embedded Linux operating system is composed of kernel and file system. The kernel is a core part of the Linux operating system, and the file system is a collection of required files. An application program is executed in the user address space, while the operating system is executed in the system address space. The operating system is protected from unexpected writing from the user program.

An application program can use a system service via system call. For example, in order to read a block of data from the hard disk, application program can use system calls like open(), read(), and close(). Disk driver program containing the actual open(), read(), and close() is implemented as a module program running in the system address space – the same address space as the kernel with equal priority.

10. Module in Linux

What Is A Kernel Module?

Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system.

Module commands and functions

Module program requires a specific syntax. Two functions named init_module() and cleanup_module() should be included in the module program.

- Init_module() function is called when the module is started, and hence it is used for initialization of the module.
- Cleanup_module() is called when the module is finished, and hence it is used to rearrange before stopping the module.

Functions callable from user application program may be included inside a module.

Command to install a module is the *insmod* command. In order to list the current modules installed, *lsmod* command can be used. Finally the command to remove a module is *rmmod* command.

A simple module program is shown in Example 1B. In order to develop a module program, *an environment which is the same as the underlying kernel* is ultimately required. In addition, header files and the ".config" file in the module program should be the same as those in the kernel program.

Example 1B. A simple embedded module program hellomod.c

```
#include <linux/module.h>     /* Needed by all modules */

static int __init hellomod_init(void)      // init_module()
{
       printk("Hello, hellomod!\n");
       return 0;
}

static void __exit hellomod_exit(void)    // cleanup_module()
{
       printk("Goodbye, hellomod!\n");
}

module_init(hellomod_init);
module_exit(hellomod_exit);

MODULE_AUTHOR("Christoper Hallinan");
MODULE_DESCRIPTION("Hello Module Example");
MODULE_LICENSE("GPL");
```

First you need to cross-compile to hellomod_mod.ko and download to the embedded board.

You can start this module program by

```
        # insmod hellomod_mod.ko
```

This module executes init_module() function, and prints a message. You can check by

```
        # dmesg | tail -20
        ……
        Hello, hellomod!
```

Note that bold fonts (except '#' symbol) represent the command to type, and normal fonts represent responses from computer.

Notice that init_module() function is executed, and a message is output using printk() function. Then this module resides beside Kernel. You can check by

```
        # lsmod
        hellomod_mod.ko
```

You can terminate this module by

```
        # rmmod hellomod_mod.ko
```

Then this module executes cleanup_module() function, and prints another message. You can check by

```
        # dmesg | tail -20
        ……
        Goodbye, hellomod!
```

For more understanding modules in Linux, read Reference [4].

SECOND WEEK
C. Reaction Timer

11. Debugger on Linux

Gdb – The GNU debugger – is most widely used.

Read and test yourself:
How to Debug C Program using gdb in 6 Simple Steps
http://www.thegeekstuff.com/2010/03/debug-c-program-using-gdb

12. Random number generation on Linux

rand(3) - Linux man page
http://linux.die.net/man/3/rand

The **rand**() function returns a pseudo-random integer in the range 0 to **RAND_MAX** inclusive (i.e., the mathematical range [0, **RAND_MAX**]).

The **srand**() function sets its argument as the seed for a new sequence of pseudo-random integers to be returned by **rand**(). These sequences are repeatable by calling **srand**() with the same seed value.

You can use this rand() function to generate a random lowercase letter or random time to wait.
Note that you need to include <stdlib.h> in order to use rand().

test_rand.c: Test random number
For hands-on experience, we test rand() function with test_rand.c as follows.

Algorithm
```
0. Include required headers:
        stdio.h, stdlib.h
0'. Declare variables with suitable types:
        int k;                          // Loop index
        double Lower, Upper, Range;     // Lower/Upper and Range for random number
        double rn[100];                 // Random number array
1. Set the range of output as [Lower, Upper] (Say, 1 to 10 s)
    Set Range = Upper – Lower;
2. Loop 100 times using for statements i=0, …, 99:
     Generate a random number, adjust range to [Lower, Upper], and store to rn[i]
   End loop
3. Compute average, min, and max values of rn[i], i=0, …, 99.
4. Print the result.
        Ex: Rand(1 to 10) Avg= 5.58478, Min= 1.07472, Max= 9.92739
```

C Program:
```c
/*
        Program test_rand.c

        Test function rand()

        Programmed by   Byung Kook Kim, Feb. 1, 2017
*/

// Declare include hader files
#include <stdio.h>
#include <stdlib.h>             // rand(), srand()
#include <time.h>               // time()


int main(void)
{
  // Declare variables to be used
  int k;                // Loop index
  double Lower, Upper, Range;    // Random number range
  double rn[100];       // Random number array

  // 1. Init variables
  Lower = 1.;
  Upper = 10.;
  Range = Upper - Lower;

  // 2. Loop 100 times
  for (k=0; k<100; ++k) {
        // Generate a random number [Lower, Upper] to rn[k]
        rn[k] = Lower + Range*rand()/RAND_MAX;
  }

  // 3. Compute avg, min, and max of rm[k]
  double Min = 1e9;
  double Max = -1e9;
  double sum = 0.;
  for (k=0; k<100; ++k) {
        sum += rn[k];
        if (rn[k] < Min) Min = rn[k];
        if (rn[k] > Max) Max = rn[k];
  }
  double Avg = sum/100.;

  // 4. Print result
  printf("Rand(1 to 10) Avg= %g, Min= %g, Max= %g\n", Avg, Min, Max);

  return 0;
}
```

How can you make rand() as random? It should generate different random numbers at each run.
Hint: You can use srand(arg) with a suitable arg.

13. Time on Ubuntu & Debian

Time – Overview of time and timers
https://linux.die.net/man/7/time

Real time and process time

Real time is defined as time measured from some fixed point, either from a standard point in the past (see the description of the Epoch and calendar time below), or from some point (e.g., the start) in the life of a process (elapsed time).

Process time is defined as the amount of CPU time used by a process. This is sometimes divided into user and system components. User CPU time is the time spent executing code in user mode. System CPU time is the time spent by the kernel executing in system mode on behalf of the process (e.g., executing system calls). The **time**(1) command can be used to determine the amount of CPU time consumed during the execution of a program. A program can determine the amount of CPU time it has consumed using **times**(2), **getrusage**(2), or **clock**(3).

The Hardware Clock

Most computers have a (battery-powered) hardware clock which the kernel reads at boot time in order to initialize the software clock. For further details, see **rtc**(4) and **hwclock**(8).

ioctl(fd, RTC_request, param): Interface driver for real-time clocks, from year to seconds: Yymmdd.hhmmss.
System clock: Software clock by kernel. For ***gettimeofday***() and ***time***().

time() returns Epoch time (since Jan 1 1970) in seconds.
ftime() returns the current time in secs and milli-secs.
gettimeofday() gives microseconds.
clock_gettime() gives nanoseconds, but is not yet widely available.

times()

times() returns user time & system time, and also the same for children. Units in clock ticks. We should know the clock resolution.

clock()

clock(): Determine processor time used by the program. Usually in us.

clock_gettime()

The best clock for performance measurement is system-wide real-time clock. In Linux, you can use clock_gettime function with CLOCK_MONOTONIC or CLOCK_MONOTONIC_RAW. Those will give you nanosecond precision.

gettimeofday()

Elapsed time, in s & us.

```
#include <sys/time.h>
       int gettimeofday(struct timeval *tv, struct timezone *tz);
       int settimeofday(const struct timeval *tv, const struct timezone *tz);
       The functions gettimeofday() and settimeofday() can get and set the
       time as well as a timezone.  The tv argument is a struct timeval (as
       specified in <sys/time.h>):

           struct timeval {
               time_t      tv_sec;     /* seconds */
               suseconds_t tv_usec;    /* microseconds */
           };

       and gives the number of seconds and microseconds since the Epoch (see
       time(2)).  The tz argument is a struct timezone:

           struct timezone {
               int tz_minuteswest;     /* minutes west of Greenwich */
               int tz_dsttime;         /* type of DST correction */
           };

       If either tv or tz is NULL, the corresponding structure is not set or
       returned.  (However, compilation warnings will result if tv is NULL.)
```

Summary

times()	Not useful in ticks.
Clock() = gettimeofday()	Elapsed time in us.
Getrusage()	User/system time in us.
clock_gettime()	Elapsed time. ns.

You can use gettimeofday() to obtain the execution time in us resolution.

Note that Debian is also a Linux distribution, but it shows a little difference from Ubuntu.
Check timer functions on Debian, and select available and accurate timer function.

14. Linux sleep functions

sleep sleep for the specified number of seconds

```
    #include <unistd.h>
     unsigned int sleep(unsigned int seconds);
```

usleep suspend execution for microsecond intervals

```
     #include <unistd.h>
     int usleep(useconds_t usec);
```

nanosleep high resolution sleep

```
    #include <time.h>
     int nanosleep(const struct timespec *req, struct timespec *rem);
        struct timespec {
            time_t tv_sec;         /* seconds */
            long    tv_nsec;        /* nanoseconds */
         };
```

Use usleep – simple, accurate up to us.

test_gettimeofday_usleep.c: Test gettimeofday and usleep.

Algorithm

```
0. Include required headers:
        stdio.h, time.h, unistd.h
0′. Declare variables with suitable types:
        int wait_us; struct timeval start_time, stop_time; double elap_time_ms; ...
1. Init variables:
        wait_us = 2000000;       // for 2 s
2. Loop three times:
        Get start_time using gettimeofday()
        Just sleep wait_us (in us) using usleep()   (or do some computational work)
        Get end_time using gettimeofday()
        Compute elap_time_ms from (end_time – start_time) and converting to ms.
        Print elap_time_ms.
```

C program:

```
/*
        Program test_gettimeofday_usleep.c

        Test Linux function: gettimeofday() and usleep()

        Programmed by   Byung Kook Kim, Feb. 1, 2017.
```

```
*/

#include <stdio.h>
#include <unistd.h>                      // usleep()
#include <time.h>                        // clock_gettime()
#include <sys/time.h>           // gettimeofday()


int main(void)
{
  int k;                                         // Loop index
  int wait_us;                                   // Wait time in us

  struct timeval start_tv, stop_tv;              // For gettimeofday() in s, us
  struct timezone tz;
  double elap_time_ms;                   // Elapsed time in ms

  // 1. Init variables
  wait_us = 2000000UL;          // Wait time of 2 s in us

  // 2. Loop three times
  for (k=0; k<3; ++k) {
        // Get Start time
        gettimeofday(&start_tv, &tz);    // Elapsed time in us

        // Wait 2 s using usleep()
        usleep(wait_us);

        // Get Stop time
        gettimeofday(&stop_tv, &tz);

        // Compute response time in ms
        elap_time_ms = (stop_tv.tv_sec - start_tv.tv_sec)*1e3
                     + (stop_tv.tv_usec - start_tv.tv_usec)*1e-3;

        // Print Response time
        printf("Elapsed_time= %g ms.\n", elap_time_ms);

  }     // End of for k

  return 0;
}
```

15. Raw key input

What is equivalent to getch() & getche() in Linux?
http://stackoverflow.com/questions/7469139/what-is-equivalent-to-getch-getche-in-linux

Good example codes. Use this in Lab 1 with modification!
Basically you have to turn off canonical mode (and echo mode to suppress echoing).

Modified code is getche.c, which is shown below.
Check getche.c and compare to the above example code.

getche.c:

```
/*
    Function getch() amnd getche() in Linux

    Modified from
http://stackoverflow.com/questions/7469139/what-is-equivalent-to-getch-getche-in-linux
```

```
    Global old_termios and new_termios for efficient key inputs.
*/

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

static struct termios old_tio;
static struct termios new_tio;

// char = getch() Reads 1 character without echo
char getch(void)
{
  char ch = 0;

  tcgetattr(0, &old_tio);                // Grab old_tio terminal i/o setting
  new_tio = old_tio;                    // Copy to new_tio
  new_tio.c_lflag &= ~ICANON;    // disable buffered i/o
  new_tio.c_lflag &= ~ECHO;      // Set echo mode off
  if (tcsetattr(0, TCSANOW, &new_tio) < 0)  perror("tcsetattr ~ICANON");
                                         // Set new_tio terminal i/o setting

  if (read(0, &ch, 1) < 0)  perror ("read()");   // Read one character

  if (tcsetattr(0, TCSADRAIN, &old_tio) < 0)  perror ("tcsetattr ICANON");
                                         // Restore old terminal i/o setting
  return ch;
}

// char = getche() Reads 1 character with echo
char getche(void)
{
  char ch = 0;

  tcgetattr(0, &old_tio);                // Grab old_tio terminal i/o setting
  new_tio = old_tio;                    // Copy to new_tio
  new_tio.c_lflag &= ~ICANON;    // disable buffered i/o
  new_tio.c_lflag &= ECHO;              // Set echo mode on
  if (tcsetattr(0, TCSANOW, &new_tio) < 0)  perror("tcsetattr ~ICANON");
                                         // Set new_tio terminal i/o setting

  if (read(0, &ch, 1) < 0)  perror ("read()");   // Read one character

  if (tcsetattr(0, TCSADRAIN, &old_tio) < 0)  perror ("tcsetattr ICANON");
                                         // Restore old terminal i/o setting
  return ch;
}
```

Compile application with multiple source files.

You have two source files: Test_getch.c and getche.c. How can you compile successfully?

Test_getch.c requires syntax as shown below:

```
// Include required header files
……
// Define function prototype
// Inform the compiler about arguments and return values: conform to actual function in
getche.c
char getch(void);
char getche(void);

int main(void)    // Function type of main() is defined also.
{
    char c;       // Set variable of return value from getch()
    ……
    c = getch();  // Call getch() and obtain return value of char c
    ……
    Return 0;
}
```

Cross compile with two source files (`Test_getch.c and getche.c`):

```
        $ arm-linux-gnueabihf-gcc -o Test_getch Test_getch.c getche.c
```

IV. Equipment and Parts

1. Lab equipment

1	Router
1	IBM PC with Windows and Linux (Dual boot)
1	Beaglebone set: Beaglebone embedded board, USB cable, and Ethernet cable
1	4 or more GB microSD card.
1	SD card reader

2. Electronic parts

NONE.

V. Design

Pre-report of first week

Read the lab text thoroughly, read supplementary materials. Understand what is required. The following items are required for a pre-report.

A. Describe the purpose and procedure of cross-development.

B. Difference of application program and module program.

Pre-report of second week

C. Read and summarize how to use gdb.

Read a good tutorial "How to Debug C Program using gdb in 6 Simple Steps", which can be found at the web http://www.thegeekstuff.com/2010/03/debug-c-program-using-gdb.
Exercise the process. Summarize how to use gdb.

D. Write component test program for Problem 1C.

Prepare test_getch.c with given getche.c

Suggestions:
The following is a suggestion for design. You can build your own design instead.

Suggested Algorithm of test_getch.c

```
0. Include required headers and declare variables.
1. Print information
        "Test getch(): Please type 16 letters sequentially.₩n"
2. Loop 16 times
        Get a key without Enter using getch()
        Print the key three times with formats "%c, %d, %2xh; " as follows:
                printf("%c, %d, %02xh; ", c, c, c);
3. Print "₩n".
```

In order to output immediately, use fflush().

E. Write an application program for Problem 1C – Reaction Timer.

The following is a suggestion of algorithm. You can build your own design instead.

```
Loop three times
        a. Random generate time T (3 sec)                rand()
             Wait random time T+2 sec.                   usleep()
        b. Random generate char x ('f' or 'j')           rand()
        c. Output a string "Type the 'x' character:"    // 'x' is 'f' or 'j'
        d. Get Start time                                gettimeofday()
        e. Wait until user hits a key – Input in raw mode and reply.
        f. Get Stop time                                 gettimeofday()
        g. Compute Response time = Stop time – Start time (in ms),
        h. Print Key correctness and response time.
```

Compile with multiple files: reaction_timer.c and getche.c, since getche.c will be reused later.

VI. Lab Procedures

Lab Procedure Summary

Prepare
Step 0. Install Ubuntu as Dual boot OS [Optional for your own PC].
Step 1.Test Ubuntu PC.
Step 2. Install BeagleBone Ubuntu on microSD.

Problem 1A.

Step 3. Test Cross-compile on PC Ubuntu.
Step 4. Test NFS.

Problem 1B.
Step 5.Prepare Module Build.
Step 6. MODULE Build.

Problem 1C.
Step 7. Exercise how to use gdb.
Step 8. Test Reaction timer components.
Step 9. Test Reaction timer App.

First Week

PREPARE

Step 0. Install Ubuntu as Dual boot OS [Optional for your own PC].

Optional, but recommended

You can use your own PC as the cross development PC. In this case, you need to install dual OS – Windows and Linux. We recommend this method: You can use it at any time!

If you decided to use your own PC, install Ubuntu to your computer, in addition to Windows. Partition the hard disk and install Linux Ubuntu 16.04 64 bit LTS (Long-Term Support) in the PC. See Appendix A for required procedure.

Note that computers in the Lab are dual OS installed also: Windows and Linux Ubuntu 16.04 64-bit LTS (Long-Term Support).

Step 1. Test operation of development PC with Ubuntu

10. Hardware

PC with Ubuntu: Lab computer is already installed as dual boot (Windows and Ubuntu).
Peripherals: Keyboard and mouse. Ethernet connection via Router.
See Fig. 1.1.

11. Get your user-name and password from TA.

Also get root password from TA, which will be used later.

12. Turn on the development PC.

If it is configured as multi-boot, select Ubuntu 16.04.
You need to specify the kernel version of Ubuntu 16.04.
Actually, use arrow up/down keys to select "Ubuntu”.
After a while, login window will appear at left-center of the display. Type in your user-name and password.

Ubuntu main window will appear.

NOTE. A Lab PC is shared by several student groups, and hence other student group is registered to the Lab PC at other times.

13. Launch several useful windows.

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

In the left column, left click "Terminal" icon 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 to launch text editor "gedit". It is a basic interactive text editor to enter program source codes or documents in text.

14. Check the kernel version of Ubuntu 16.04

In one terminal, type

```
$ uname -r
4.4.0-57-generic
```

Or similar. Hence we check that the kernel version of Ubuntu 16.04 is 4.4.0-57-generic, or similar.

Note that the symbol '$' means command prompt. In Ubuntu, it shows "bkkim@bkkim-LPC3:~$" or similar. You need to type in the command line after the '$' symbol: Do not type '$'! The second line `4.4.0-57-generic`" is the response to your command line from Ubuntu: Hence do not type: Just check.

15. Check network connection.

In one terminal, type

```
$ ifconfig
eth1      Link encap:Ethernet  HWaddr 00:1b:21:0c:00:94
          inet addr:192.168.100.12 ……
lo        Link encap:Local Loopback
          inet addr:127.0.0.1 ……
```

Check that the "ethN" has correct IP address such as 192.168.100.12 as above. This is the PC IP. 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 192.168.100.1 (The last number in IP changed to 1). "-c 5" means 5 trials.

```
 $ ping 192.168.100.1 -c 5
PING 192.168.100.1 (192.168.100.1) 56(84) bytes of data.
64 bytes from 192.168.100.1: icmp_req=1 ttl=64 time=9.04 ms
```

```
64 bytes from 192.168.100.1: icmp_req=1 ttl=64 time=0.543 ms
……
--- 192.168.100.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.434/9.049/5.613/2.068 ms
```

16. Test web browser.

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

Step 2. Install Beaglebone Debian on microSD

20. Hardware

PC with Ubuntu
Keyboard, mouse, and display. Ethernet connection via Router.
Beaglebone with Ubuntu
Ethernet connection via Router. Power from USB cable.
See Fig. 1.1.

21. Download proven Debian OS image

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

Download "Debian Wheezy 7.9" to ~/Downloads and check

```
$ ls -la ~/Downloads
-rw-rw-r--  1 bkkim bkkim 626164520  1월  5 10:04 bone-debian-7.9-lxde-4gb-armhf-2015-11-12-4gb.img.xz
```

Size is 626 MB.

Note. You may tempt to download the newest Debian Jessie 8.6. However, it is revealed to have problem in WiFi setting later. Hence we use stable (a little old) version of Debian Wheezy 7.9.

Check sha256sum

```
$ cd ~/Downloads
$ sha256sum bone-debian-7.9-lxde-4gb-armhf-2015-11-12-4gb.img.xz
f6e67ba01ff69d20f2c655f5e429c3e6c2398123bcd3d8d548460c597275d277  bone-debian-7.9-lxde-4gb-armhf-2015-11-12-4gb.img.xz
```

How can you type the filename without mistype?
Auto-completion: Assuming that no other file starts with "bo…", you can just type

```
sha256sum bo [TAB]
```

The remaining part of the filename will be filled automatically by OS.

Check matchto Web sha256sum:

sha256sum: f6e67ba01ff69d20f2c655f5e429c3e6c2398123bcd3d8d548460c597275d277

Copy to a suitable directory such as ~/CDE/Debian/7.9

Unpack .....image.xz (Option “-k” keeps the source file):

```
$ cd ~/CDE/Debian/7.9
$ unxz -k  bone-debian-7.9-lxde-4gb-armhf-2015-11-12-4gb.img.xz
```

Check sizes

```
$ ls -la
-rw-rw-r-- 1 bkkim bkkim 3565158400  1월 19 17:48 bone-debian-7.9-lxde-4gb-armhf-2015-11-12-4gb.img
-rw-rw-r-- 1 bkkim bkkim  626164520  1월 19 17:48 bone-debian-7.9-lxde-4gb-armhf-2015-11-12-4gb.img.xz
```

Notice that the size of output file X.img is increased to 3.56 GB.

22. Write Debian image to uSD

Refer Installation/FromImgFiles - Command Line Interface
https://help.ubuntu.com/community/Installation/FromImgFiles

Insert blank 4 GB uSD card into SD card reader.
Connect the SD card reader to PC via USB cable.
Check uSD device name.

```
$ df
/dev/sdc1         75754    5726     70028   8% /media/boot
/dev/sdc2       7660836  739892   6536648  11% /media/rootfs
```

It shows device name of microSD as /dev/sdc. Depending on system, it may show other device name with format /dev/sdN, where N can be any lowercase alphabet. Remember this device name for later use.

Unmount sdc 1 & 2:

```
$ umount /dev/sdc1
$ umount /dev/sdc2
```

Go to the directory of downloaded Debian.
Run dd as follows:

```
$ sudo dd if=/path/to/downloaded.img of=/dev/devicenode bs=1M
```

Wait (about 12 min)......

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

Check File Browser that /dev/sdc is unmounted (disappeared).
You can also check in the terminal with 'df' command, and check /dev/sdc? are disappeared.

23. Prepare PC for Bone console

We are going to use "Bone" and "Beaglebone" interchangeably.
In one PC terminal window, install minicom.

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

Setup minicom to use as "console terminal" (major I/O terminal) for Beaglebone.
Configure minicom with

```
$ sudo minicom -s -w
Enter the root password.
```

The "Configuration" menu will appear:

```
+-----[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        |
+--------------------------+
```

Select the third row submenu labeled "Serial port setup" with up/down arrow keys and then press the enter key.
Type 'A' to select Serial Device. Change to '/dev/ttyUSB0'. (Do not type "'")
Type 'F' to select Hardware Flow Control. It will be changed to 'No'.
Press Enter to quit this submenu.

Select "Save setup as dfl". 'Configuration saved' sub-window will appear.
Select "Exit". Exit minicom by typing Ctrl+A, Q, and then Y.

Afterwards, when you start the minicom, no more setting is required. Hence type (without "-s": No setup)

```
$ sudo minicom   -w
```

24. Start Beaglebone

Disconnect USB cable of SD reader.
Remove uSD from SD reader, and then insert it to Beaglebone.

Power on the Beaglebone.
Connect 5V-DC, Ethernet, and then USB.
It seems that 5V_DC is required for sufficient current (2A). USB cannot provide sufficient current.

In the console window for Beaglebone, issue minicom command:

```
 $ sudo minicom -w
 Welcome to minicom 2.7
 ……
default username:password is [debian:temppwd]

beaglebone login:
```

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

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

Does this prompt appear?

NOTE. If your keyboard input does not respond, check that the Hardware Flow Control in minicom setup is set to 'NO'.

Note. To exit Minicom, press 'Ctrl-A' to get a message bar at the bottom and then press 'X'.

25. Check Beaglebone Debian kernel version

Check Kernel version

```
# uname -r
3.8.13-bone79
```

Debian Wheezy 7.9 is successfully installed in uSD!
The symbol '#' denotes Bone Ubuntu prompt. It is usually shown as "ubuntu@arm:~$" or similar.

Note that PC and Bone use different kernel and OS version as follows:

	Linux version	Kernel version
PC Ubuntu	Ubuntu16.04.3	4.4.0-57-generic
Beaglebone Debian	Debian Wheezy 7.9	3.8.13-bone79

26. Configure Beaglebone Debian

Add superuser with passwd.

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

Become superuser

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

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

```
# ls -la /root
total 28
drwx------  5 root root 4096 Nov 12 19:40 .
drwxr-xr-x 22 root root 4096 Nov 12 19:41 ..
-rw-r--r--  1 root root  570 Jan 31  2010 .bashrc
drwxr-xr-x  8 root root 4096 Nov 12 19:27 .c9
drwxr-xr-x  3 root root 4096 Nov 12 19:24 .cache
drwxr-xr-x  3 root root 4096 Nov 12 19:27 .node-gyp
-rw-r--r--  1 root root  140 Nov 19  2007 .profile
```

Add user (as superuser, for later user login)

```
# adduser bkkim
Adding user `bkkim' ...
Adding new group `bkkim' (1003) ...
Adding new user `bkkim' (1001) with group `bkkim' ...
Creating home directory `/home/bkkim' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for bkkim
Enter the new value, or press ENTER for the default
       Full Name []: Byung Kook Kim
       Room Number []:
       Work Phone []:
       Home Phone []:
       Other []:
Is the information correct? [Y/n] Y
```

Check /home directory

```
# ls /home
bkkim  debian
```

Two users exist with names "debian" and "bkkim".

Make bkkim as sudoer: Enables "sudo apt-get ..." as user.

```
 # su
 # adduser bkkim sudo
Adding user `bkkim' to group `sudo' ...
Adding user bkkim to group sudo
Done.
```

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

```
# logout
```

At the "login:" prompt, type in "root" and then root-password.

You can exit su by "exit".

```
# exit
```

27. Check network

Check network configuration with ifconfig:

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr d4:94:a1:90:f4:a7
  inet addr:192.168.100.18  Bcast:192.168.100.255  Mask:255.255.255.0
```

Note 1. ---

In Begalbebone with Debian, catching IP takes time. You may have to wait one minute or more to obtain IP.

Note 2. ---

If you can't get an IP address:

1. On your Beaglebone, type the following command

   vi /etc/network/interfaces

2. Modify the following line within the file as follows

   "iface eth0 inet static" => "iface eth0 inet dhcp"

3. Save and exit the file.

4. Type the following command

   /etc/init.d/networking restart

Once you complete the steps above, your Beaglebone will get an IP address automatically.

Now you can rlogin (remote login) to Beaglebone in other terminal window:

```
$ ssh bkkim@192.168.100.18
bkkim@192.168.100.18's password:
```

rlogin or ssh can be terminated by typing "logout".

Note. Log in to root may not work: due to security reason. Direct rlogin as root to a remote computer can be

very dangerous (for hackers).
Hence we have to do rlogin to Ubuntu, and then become a superuser by ‵su′, which requires two passwords.

Note. rlogin or ssh can be terminated by typing "logout".

28. Update package

Update packages (as root)

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

It takes a while..... (20 minutes or more).
For further test without waiting, Proceed in another terminal using ssh.

Check disk usage

```
 # df
Filesystem     1K-blocks    Used Available Use% Mounted on
rootfs           3263536 2049752   1044672  67% /
```

67% seems OK.

29. When shut down a beagle board...

When you shut down a beagle board, please write the following:

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

or your beagle board might be broken.

Problem 1A
Step 3. Test Cross-compile on PC Ubuntu

31. Install cross-compiler for ARM in the PC.

In PC, check if a cross-compiler already installed.

```
$ arm-linux-gnueabihf-gcc --version
arm-linux-gnueabihf-gcc (Ubuntu/Linaro 4.8.2-16ubuntu4) 4.8.2
Copyright (C) 2013 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 the cross-compiler does not exist, install:

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

32. Make a working directory

In PC, make a working directory named 'DesignLab/1_CrossDevEnv/a_GetStarted'.

```
$ mkdir -p ~/DesignLab/1_CrossDevEnv/a_GetStarted
$ cd ~/DesignLab/1_CrossDevEnv/a_GetStarted
```

Note. For the last command, you can simply type c, d, D, [Tab], 1, [Tab], a, [Tab], and [Enter] to get the same result: Less labor!

33. Edit the example program.

Edit the example 1A program hello_es.c using gedit. Launch the gedit window:

```
$ gedit hello_es.c
```

Then type in the example 1A program. Check if it is correctly saved.

```
$ ls
hello_es.c
```

34. Cross-compile the program

Cross-compile:

```
$ arm-linux-gnueabihf-gcc -o hello_es hello_es.c
```

Check resultant "hello_es"

```
$ ls -la
total 20
drwxrwxr-x 2 bkkim bkkim 4096 Dec  5 12:49 .
drwxrwxr-x 3 bkkim bkkim 4096 Dec  5 12:06 ..
-rw-rw-r-- 1 bkkim bkkim  115 Dec  5 12:12 hello_es.c
-rwxrwxr-x 1 bkkim bkkim 7796 Dec  5 12:48 hello_es
```

35. Use makefile

Alternatively you can use Makefile. Edit "Makefile" as follows:

```
#       Embedded Systems Lab1 Testgcc Makefile

# Source file: hello_es.c

# Do everything
all:    nc cc

# Native compile
nc:
        gcc -o hello_es_pc hello_es.c

# Cross compile for Bone Ubuntu
cc:
        arm-linux-gnueabihf-gcc -o hello_es hello_es.c

# Clean up
clean:
        rm -f hello_es~~-~~_pc hello_es
```

Since we compile many times (until all errors has gone…), using Makefile is much more convenient.

Use Makefile:

Clear all previously compiled files.

```
$ make clean
$ ls
hello_es.c  Makefile
```

Compile all

```
$ make all                          // Or just type "make"
gcc -o hello_es_pc hello_es.c

arm-linux-gnueabihf-gcc -o hello_es hello_es.c
$ ls
hello_es  hello_es.c  hello_es_pc  Makefile
```

Two files are generated: hello_es & hello_es_pc.

36. Try to run on PC

Try

```
$ ./hello_es_pc
Hello, application program for embedded system!
```

Try

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

Is it correct? The result is as expected: It is not native-compiled to run on PC. Instead it is cross-compiled and intended to run on embedded board with ARM CPU! In contrast, hello_es_pc can run on PC.

37. Download

Power on Bone.
Connect Ethernet cable.
Connect USB cable: Power on BeagleBone
Open BeagleBone Console on PC

```
$ sudo minicom -w
```

Login as a normal user

Make a working directory on Bone (With a user, not root)

```
# mkdir -p ~/test_scp
# cd ~/test_scp
```

Copy from PC terminal

```
$ scp hello_es bkkim@192.168.100.18:/home/bkkim/test_scp
```

```
        bkkim@192.168.100.18's password:
```

Check on Bone console:

```
        # cd ~/test_scp
        # ls
        hello_es
```

Or copy from BeagleBone console

```
        # scp
bkkim@192.168.100.3:/home/bkkim/DesignLab/1_CrossDevEnv/a_GetStarted/hello_es_pc .
```

Don't forget the last space and '.'!

Note 1. Sometimes, error occurs with:

ssh: connect to host 192.168.100.17 port 22: Connection refused

In this case, install openssh-server on PC:

```
        $ sudo apt-get install openssh-server
```

Note 2. If the following message appears when copying from PC terminal:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

Then perform on PC

```
        $ mv ~/.ssh ~/.ssh.old
```

Note 3. Due to security reason, some scp command does not function.

38. Run on Beaglebone

Try

```
        # ./hello_es_pc
        -bash: ./hello_es_pc: cannot execute binary file
```

Try

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

Step 4. Setup NFS

Set up NFS server on PC Ubuntu.

https://help.ubuntu.com/lts/serverguide/network-file-system.html

41. Install NFS server on PC

Installl nfs server on PC.

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

42. Configure NFS

Remember IPs for PC and Bone, for example,

PC: eth1 192.168.100.12
Bone: eth0 192.168.100.18

Edit /etc/exports to INCLUDE hosts allowed to connect (i.e., IP of Beaglebone).

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

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check)
hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#

# nfs for Bone Ubuntu - RoboCam
/home/bkkim 192.168.100.18(rw,sync,no_root_squash,no_subtree_check)
```

Whenever we modify /etc/exports, we must run 'exportfs' to make the changes effective afterwards.

```
        $ sudo exportfs -a
```

43. 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
```

Check if nfs started.

```
        $ ps -aux | grep nfs
        …...
        root      3005  0.0  0.0      0     0 ?       S    17:51   0:00 [nfsd]
        …...
```

Set NFS client on Beaglebone Debian (as user)

44. Install nfs client in Bone

Install nfs-client software

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

Wait......

45. Make the mount point

```
# mkdir ~/nfs_client
```

There should be no files or subdirectories in the ~/nfs_client directory.

46. Start Beaglebone nfs client

```
# cd ~
# sudo mount 192.168.100.12:/home/bkkim/RoboCam ~/nfs_client
```

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

```
$ ping 192.168.100.18
```

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

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

Type in as follows:

```
#! /bin/sh
# bone_nfs_client.sh
# Mount nfs_client on Bone
sudo mount 192.168.100.12:/home/bkkim ~/nfs_client
echo "Mount nfs_client in Bone Ubuntu ~/nfs_client"
```

Change mode of bone-nfs-dl to 775: It makes bone_nfs_client.sh executable.

```
# cd ~
# chmod 775 bone_nfs_client.sh             // or
# chmod a+x bone_nfs_client.sh
```

Later, you can simply type to start nfs client:

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

48. Go to nfs directory

```
# cd ~/nfs_client
# ls
DesignLab …
```

You can see PC-Ubuntu directory ~/RoboCam via nfs!

49. Run hello_es using nfs

Go to the directory for hello_es. You can run "hello_es" without scp.

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

Success?

Problem 1B
Step 5. Prepare Module Build

We assume that Kernel source can be downloaded from the Lecture web.

51. Get Kernel source from Web for RoboCam.

Get Beaglebone compressed Kernel source

```
Kernel3.8.13-bone79_linux.tgz
```

from the Lecture Web server.

52. Install Kernel source

```
$ cd ~/SW/Bone/Ubuntu_1304/Kernel3813-bone37/linux-dev
$ tar -xvzf Kernel3.8.13-bone79_linux.tgz
```

Directory KERNEL will be generated.

```
$ ls linux
arch … kernel … Makefile … vmlinux …
```

53. Configure cross compile environment

Edit sh/cde_bd_k3813_bone79 as follows:

```
#!/bin/bash
### Shell script cde_bd_k3813-bone79
### Setup cross-compile environment for Bone-Debian with Kernel 3.8.13-bone79
### Usage: source sh/cde_bd_k3813-bone79

## Line starting with '#' means comment line.

## Set MACHINE
MACHINE=beaglebone

## Set SYSROOTSDIR & STAGEDIR
SYSROOTSDIR=/usr
STAGEDIR=${SYSROOTSDIR}

## Set CROSSBINDIR (where cross compiler exists)
CROSSBINDIR=/usr/bin

## Set KERNELDIR (where the Linux kernel source is located)
## NOTE: This path to KernelDir should be exact.
```

```
export KERNELDIR==/home/bkkim/CDE/Debian/7.9/Kernel/linux

## Set PATH
PATH=${CROSSBINDIR}:${PATH}

unset CFLAGS CPPFLAGS CXXFLAGS LDFLAGS MACHINE

export ARCH="arm"
export CROSS_COMPILE="arm-linux-gnueabihf-"
export CC="arm-linux-gnueabihf-gcc"
export LD="arm-linux-gnueabihf-ld"
export STRIP="arm-linux-gnueabihf-strip"

echo "Set cross-development environment for Beaglebone Debian (3.8.13-bone79
kernel)."
```

Note that KERNELDIR should match your computer. Correct if necessary.

Set the cross-compile environment

```
$ source sh/cde_bd_k3813_bone79
  Setting cross-development environment for Beaglebone Debian (3.8.13-bone79
kernel).
```

Note that this command should be repeated after power on the development PC. Type the same command in every terminal.

Step 6. MODULE COMPILE

61. Make another working directory

Make a suitable working directory different from Problem 1A, since we need another Makefile in a new directory.

```
$ mkdir -p ~/DesignLab/1_CrossDevEnv/b_ModuleBuild
$ cd ~/DesignLab/1_CrossDevEnv/b_ModuleBuild
```

62. Edit Makefile

Edit Makefile as follows.

```
# Bone-Debian cross-compile module makefile

ifneq ($(KERNELRELEASE),)
    obj-m := hellomod.o
else
    SUBDIRS := $(shell pwd)

default:
ifeq ($(strip $(KERNELDIR)),)
       $(error "KERNELDIR is undefined!")
else
       $(MAKE) -C $(KERNELDIR) M=$(SUBDIRS) modules
endif
```

```
clean:
	rm -rf *~ *.ko *.o *.mod.c modules.order Module.symvers .pwm* .tmp_versions

endif
```

63. Make

```
$ make clean
$ ls
HelloMod.c  Makefile

$ make
make -C /home/bkkim/CDE/Debian/7.9/Kernel/linux
M=/home/bkkim/Lab/DesignLab17/1_CrossDevEnv/b_ModuleBuild modules
make[1]: Entering directory '/home/bkkim/CDE/Debian/7.9/Kernel/linux'
  CC [M]  /home/bkkim/Lab/DesignLab17/1_CrossDevEnv/b_ModuleBuild/hellomod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/bkkim/Lab/DesignLab17/1_CrossDevEnv/b_ModuleBuild/hellomod.mod.o
  LD [M]  /home/bkkim/Lab/DesignLab17/1_CrossDevEnv/b_ModuleBuild/hellomod.ko
make[1]: Leaving directory '/home/bkkim/CDE/Debian/7.9/Kernel/linux'
```

64. Insert module on Beaglebone

Now, go to the Beaglebone terminal with 'sudo minicom –w'.

Start nfs as user.

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

Go to the working directory

```
# cd ~/nfs_client/1_CrossDevEnv/b_ModuleBuild
```

Become superuser

```
# su
```

Insert module HelloMod.ko on Bone as root

Insert module

```
# insmod hellomod.ko
```

Check if inserted correctly

```
# lsmod
Module                  Size  Used by
hellomod                775  0
…...
```

Check output

```
        # dmesg | tail
        …...
        [ 1656.516056] Hello, hellomod!
```

Check modinfo

```
       # modinfo hellomod.ko
       filename:      /home/bkkim/Lab/RoboCam16_64/0_SetUp/HelloMod/HelloMod.ko
       license:       GPL
       description:   Hello Module Example
       author:        Christoper Hallinan
       srcversion:    CEADE72078B65B98567B954
       depends:
       vermagic:      3.8.13 SMP mod_unload modversions ARMv7 thumb2 p2v8
```

65. Remove the module

Remove module

```
        # rmmod hellomod
```

Check output

```
        # dmesg | tail
        …...
        [ 1789.645019] Goodbye, hellomod!
```

If successful, the experiment for the first week is done. Congratulations!
No need to demo to TA this week.

Second Week

PROBLEM 1C

Step 7. Exercise Gnu Debugger gdb

Read and test yourself:
How to Debug C Program using gdb in 6 Simple Steps
http://www.thegeekstuff.com/2010/03/debug-c-program-using-gdb

Step 8. Test Reaction Timer Components

81. Edit Makefile

Make a working directory

```
        $ mkdir -p ~/DesignLab/1_CrossDevEnv/c_ReactionTimer
```

Edit Makefile to compile Programs:

Test_rand, test_gettimeofday_usleep, test_getch, and reaction_timer.

For each program, native compile first to test on PC, and then cross compile to test on Bone.

82. Test test_rand_pc on PC

Edit prepared test_rand.c

Compile using Makefile
Re-edit if compile error

Run and test using Gdb.

If your result of multiple run shows

```
$ ./test_rand_pc
Rand(1 to 10) Avg= 5.92142, Min= 1.14671, Max= 9.99032
$ ./test_rand_pc
Rand(1 to 10) Avg= 5.92142, Min= 1.14671, Max= 9.99032
$ ./test_rand_pc
Rand(1 to 10) Avg= 5.92142, Min= 1.14671, Max= 9.99032
```

They produce the same result for each run: Not random!

Change your program to show different results of multiple run

```
$ ./test_rand_pc
Rand(1 to 10) Avg= 5.74366, Min= 1.06545, Max= 9.83656
$ ./test_rand_pc
Rand(1 to 10) Avg= 5.643, Min= 1.05605, Max= 9.93379
$ ./test_rand_pc
Rand(1 to 10) Avg= 5.62338, Min= 1.22069, Max= 9.95474
```

They should produce different random results!

83. Test test_gettimeofday_usleep_pc on PC

Proceed similar to 82.

Typical result:

```
$ ./test_gettimeofday_usleep_pc
Elapsed_time= 2000.21 ms.
Elapsed_time= 2000.24 ms.
Elapsed_time= 2000.24 ms.
```

You can notice a little jitter in Elapsed_time.

84. Test test_getch_pc on PC

Proceed similar to 82.

Typical result:

```
$ ./test_getch_pc
Test getch(): Please type 16 letters sequentially.
a, 97, 61h; b, 98, 62h; c, 99, 63h; d, 100, 64h; e, 101, 65h; f, 102, 66h; g, 103,
67h; h, 104, 68h; i, 105, 69h; j, 106, 6ah; 1, 49, 31h; 2, 50, 32h; 3, 51, 33h; 4,
52, 34h; 5, 53, 35h; 6, 54, 36h;
```

Can you obtain good results for lowercase letters?
How about upper case letters? Numbers? F1 to F12?
How about ESC key? Ctrl+C key?

Now test on Bone.

85. Test test_rand on Bone

86. Test test_gettimeofday_usleep on Bone

87. Test test_getch on Bone

Step 9. Test reaction_timer

91. Test reaction_timer on PC

Can use the same directory: c_ReactionTimer.
Edit ReactionTimer.c

Native compile

```
$ gcc -g -o reaction_timer_pc reaction_timer.c getche.c
```

Run (with Gdb Iif required)

```
$ ./reaction_timer_pc
Type the 'j' character: f
  Incorrect response f in 1331.02 ms.
Type the 'f' character: f
  Correct response f in 1056.13 ms.
Type the 'j' character: j
  Correct response j in 513.711 ms.
```

92. Test reaction_timer on Bone

Proceed similar to 91.

If successful, demonstrate ReactionTimer on Bone to TA!

VII. Final Report

Each student should prepare his own report, which means that the discussion should be different for each student in the same group.

The final report contains:

- Purpose
- Experiment sequence
- Experimental results
- Discussion: Should be unique! Different even for each member of the same team.
- References

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

1) What is the difference of development system and target system?

2) What is the difference of application program and module program?

3) For application programs on Beaglebone, is it possible to native-compile on Beaglebone? Discuss advantages and disadvantages of native-compile and cross-compile in this case.

4) Why we test on PC, and then test on Bone? Is it waste of time? What are limitations in this method?

5) Set your own topic to discuss related to Lab 1, and explain summary of your search result.

VIII. References

[1] Beaglebone Rev A6 System Reference Manual,
http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SRM.pdf

[2] AM3359 Datasheet, AM335x ARM Cortex-A8 Microprocessors (MPUs) (Rev. J),
http://www.ti.com/lit/ds/symlink/am3359.pdf

[3] Technical Reference Manual - AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual (Rev. O), http://www.ti.com/lit/ug/spruh73o/spruh73o.pdf

[4] Alesandro Rubini, "Linux Device Drivers", 3rd edition. http://free-electrons.com/doc/books/ldd3.pdf

VIII. Appendices

Appendix A. Ubuntu 16.04 Install on Windows PC

1. Use Disk management in Control panel

In Windows 7 or 10, start the Control panel.

Start > Control panel.

Click to

System and Security > Management Tools > Computer management > Disk management

Regardless of your disk use MBR or GPT, at least five partitions are suggested as follows:

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 (Logical partitions for MBR).

- Partition root (/): For system (Linux and System application software). At least 10 GB.
- Partition swap: For swap area. Twice the size of main memory, e.g. 8 GB for 4 GB RAM.
- Partition home (/home): For user program. At least 20 GB.

The three Linux partitions are reserved with suitable sizes. Actual designation of partitions will be performed in installing Ubuntu.

2. Download Ubuntu 16.04 64 bit

2A. Download on Windows PC - Method A

Visit Ubunut.com http://www.ubuntu.com

In the Ubuntu home page, click "Desktop" in the top orange menu stripe.

In "Ubuntu for desktops" page, click "Download Ubuntu".

In "Download Ubuntu Desktop" page, see "Ubuntu 16.04 LTS (Long Term Support)"

Select 'Choose your flavour' as "64-bit –- recommended".

Click "Download"

In "Help shape..." page, you may contribute to Ubuntu.

Click "Not now, take me to the download".

In pop-up window "Opening... ",

check the filename: "ubuntu-16.04-desktop-amd64.iso". // Not "...i386.iso"
Click "Save File", and then click "OK".
It will be downloaded to ~/Downloads: Takes a long time...

2B. Download on Windows PC - Method B (Much faster)

Instead, you can use KAIST Mirror site.
Visit ftp://ftp.kaist.ac.kr.
Select "ubuntu-cd".
Select "16.04".
Select "ubuntu-16.04-desktop-amd64.iso". // Not "...i386.iso"

In pop-up window "Opening... ",
check the filename: "ubuntu-16.04-desktop-amd64.iso". // Not "...i386.iso"
Click "Save File", and then click "OK".
It will be downloaded to ~/Downloads: Takes a shorter time!

3. Make a bootable media

3A. How to burn a DVD on Windows

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

3B. How to create a bootable USB stick on Windows

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

4. Install

Refer Install Ubuntu 16.04 LTS
http://www.ubuntu.com/download/desktop/install-ubuntu-desktop

1. Using DVD or USB?
2. Prepare to install Ubuntu
3. Setup Wireless

4. Allocate drive space
Select "Something else".
Allocate three partitions as follows:

/	Format	Root partition, 10 GB or more
Swap		Twice the memory size
/home	Format	Home partition, 20 GB or more

5. Begin the installation

6. Select your location
7. Select your preferred keyboard layout.
8. Enter your login and password details.
9. Learn more about Ubuntu while the system installs...
10. That's it!

Appendix B. Linux Basics

Getting Started

▪ # login: *username* or *root*	
▪ # password: *user_password* or *root_password*	
▪ # logout	
▪ # sudo shutdown –h now	; Shutdown the computer

Basic commands

▪ # date	; Display date and time
▪ Ex) Wed Sep 1 12:12:29 EDT 2004	
▪ # 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

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

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

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

▪ 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

▪ 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