Lab1 Procedure:
Embedded Board and Development Environment Setup

Preparation

- IBM PC with Linux
- Beaglebone set: Beaglebone embedded board, USB cable, 1 ethernet cable, 4(or More) GB microSD card (with debian Linux image in it)
- getche.c, getche.h source code (prob. 1c)

Problems

Pre Lab

- Basic settings for Beaglebone Black

Prob 1A. Cross-development system

- Test Cross-compile on PC Ubuntu
- Test NFS

Prob 1B. Exercise1: Timed loop program

- Write a program that performs countdown using a timed loop
- Use standard library

Prob 1C. Exercise2: User input program

- Use a preferred text editor
- Write a timer program that takes in user input and prints out the elapsed time.
- Compile source code with multiple files with CMake

Lab Contents

Preparation

1. Turn on the development PC.
2. Launch several useful windows.
3. Check network connection.
4. ~~Download proven Debian OS image~~
5. ~~Write Debian image to uSD~~
6. Start Beaglebone
7. Configure Beaglebone Debian
8. Check network for Beaglebone
9. Update package
10. When shutting down a beagle board...

Problem 1A. Cross-development system

1. Install cross-compiler for ARM in the PC.
2. Make a working directory
3. Write a hello world program.
4. Compile the program
5. Cross-compile the program
6. Try to run on PC
7. Download and run on Beaglebone
8. Install NFS server on PC
9. Configure NFS
10. Start PC NFS server
11. Install nfs client in Beaglebone
12. Make the mount point on Beaglebone
13. Start Beaglebone nfs client
14. For later use, edit ~/bone_nfs_client.sh using nano
15. Go to nfs directory
16. Run hello_es using nfs

Problem 1B. Exercise1: Timed loop program

1. Write a timed loop program in PC using Chrono
2. Compile the program and run on PC
3. Cross-compile the program and run on Beaglebone

Problem 1C. Exercise2: User input program

1. Install VSCode (or preferred text editor / IDE)
2. Install CMake
3. Configure the source code
4. Write a timed loop program in PC
5. Use debugger
6. Cross-compile the program and run on Beaglebone

Lab Procedure

Preparation

1. Turn on the development PC.

First of all, get a root password from TA, which will be used later.
If it is configured as multi-boot, select Ubuntu 16.04.
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.

2. 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 a **terminal** window. It is the basic text user interface. Or, you can open a terminal with shortcut **“ctrl+alt+t”**

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. You can also open a gedit window in a terminal by typing “gedit”

```
$ gedit
```

Note that you don’t need to type “$”. This just means that this is a terminal(bash) command

3. 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.x.x 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
1964 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
```

This means that you are able to use the internet.

~~4. Download proven Debian OS image~~

Important note : The TA will provide a micro SD card with the Debian OS installed, so you don't need to do this section.

Visit BeagleBoard.org Latest Firmware Images (http://beagleboard.org/latest-images)
Download the latest version of image to ~/Downloads and check

Command “ls” shows all folder/files in the folder.

```
$ ls -la ~/Downloads

-rw-rw-r--  1 kh11kim kh11kim 598798048 12월  1 16:55
bone-debian-10.3-iot-armhf-2020-04-06-4gb.img.xz
```

Check sha256sum

```
$ cd ~/Downloads
$ sha256sum bone-debian-10.3-iot-armhf-2020-04-06-4gb.img.xz
22448ba28d0d58e25e875aac3b4e91eaef82e2d11c9d2c43d948ed60708f7434
 bone-debian-10.3-iot-armhf-2020-04-06-4gb.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]
bone-debian-7.9-The remaining part of the filename will be filled automatically by OS.
```

Check match to Web sha256sum:
sha256sum: 22448ba28d0d58e25e875aac3b4e91eaef82e2d11c9d2c43d948ed60708f7434
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
```

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

5. ~~Write Debian image to uSD~~

Important note : The TA will provide a micro SD card with the Debian OS installed, so you don't need to do this section.

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
```

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.

Follow the instruction in this link (https://beagleboard.org/getting-started)

Tip. You may download and execute balenaEtcher .Appimage file
When downloaded, you should make the file executable.

```
$ ls -al
-rw-rw-r--  1 kh11kim kh11kim  94162624  2월  9 15:34 balenaEtcher-1.14.3-x64.AppImage
….
$ chmod +x balenaEtcher-1.14.3-x64.AppImage
$ ls -al
-rw-rw-r-x  1 kh11kim kh11kim  94162624  2월  9 15:34 balenaEtcher-1.14.3-x64.AppImage
```

You can see the file become executable. Let's open it.

```
$ ./balenaEtcher-1.14.3-x64.AppImage
```

Now you can follow the remaining process accordingly.
Remove your flash media when the process completes.

6. Start Beaglebone with ssh command

~~Disconnect USB cable of SD reader.~~
~~Remove uSD from SD reader, and then~~ insert microSD(=uSD) to Beaglebone.

Connect 5V-DC, Ethernet, and then USB.
Important!. To boot up the Beaglebone from the OS in uSD, push and hold the boot button until the lights are turned on.
(It seems that 5V_DC is required for sufficient current (2A). USB cannot provide sufficient current.)

If connected, a new network adaptor will showed.
(Refer to https://beagleboard.org/static/START.htm)

```
$ ifconfig
…
enx0cb2b7ca173a: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

```
        inet 192.168.7.1  netmask 255.255.255.252  broadcast 192.168.6.3
…
```

Note that the local network(via USB) ip for Beaglebone is fixed as 192.168.7.2.
We'll connect to Beaglebone using secure shell command(ssh, https://en.wikipedia.org/wiki/Secure_Shell).
The first userid and password is "debian", and "temppwd"

```
$ ssh debian@192.168.7.2
The authenticity of host '192.168.7.2 (192.168.7.2)' can't be established.
ECDSA key fingerprint is SHA256:+7PFvmGldXtlAvA25EoPXC/rMdu9Bu1PIi47+KNgoA4.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
$ yes
debian@192.168.7.2's password:
$ temppwd
…
debian@beaglebone:~ $
```

Tip. You can't see your typing when you are inputting the password.

Check kernel version of beaglebone.

```
# uname -a
Linux beaglebone 4.14.71-ti-r80 #1 SMP PREEMPT Fri Oct 5 23:50:11 UTC 2018 armv7l GNU/Linux
```

Note. From here on, I'll use # for beaglebone shell.

7. Configure Beaglebone Debian

Add superuser with passwd.

```
# sudo passwd root
```

Become superuser (administrator in windows)

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

Add user (as superuser, for later user login)

```
# adduser {yourid}

Adding user `yourid' …
Adding new group `yourid' (1003) …
Adding new user `yourid' (1001) with group `yourid' …
Creating home directory `/home/yourid' …
Copying files from `/etc/skel' …
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for yourid
Enter the new value, or press ENTER for the default
```

Full Name []: {Your name}
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n] Y

Check /home directory

```
# ls /home
yourid    debian
```

Two users exist with names “debian” and “yourid”.

Make yourid as sudoer: Enables “sudo apt-get ...” as user.

```
# su
# adduser yourid sudo
Adding user `yourid' to group `sudo' …
Adding user yourid 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
```

8. Check network for Beaglebone

Check network configuration with ifconfig:

```
# ifconfig
eth0: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet 192.168.xx.xxx  netmask 255.255.255.0  broadcast 192.168.50.255
```

The number after “inet” is the temporary ip.

Note. If you can’t get an IP address:----------

1. On your Beaglebone, type the following command
   nano /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.

---

9. Update package

Update packages (as root)

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

It takes a while..... (20 minutes or more).

10. When shutting 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. Cross-development system

17. 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 g++-arm-linux-gnueabihf
```

18. 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. -p option makes directory with other parent directories
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!

19. Write a hello world program.

Edit the example 1A program hello.cpp using gedit. Launch the gedit window:

```
$ gedit helloworld.cpp
```

Then type in the example 1A program (The code is on Lab1 Experimental Guide, but you can write your own code). Check if it is correctly saved.

```
$ ls
helloworld.cpp
```

20. Compile the program

Compile:

```
$ g++ -o helloworld helloworld.cpp
```

Note that -o option is for output name.
Check resultant “helloworld”

```
$ ls -la
```

Execute it!

```
$ ./helloworld
Hello, world!
```

21. Cross-compile the program and try to run on PC

The command is exactly same except [g++ → arm-linux-gnueabihf-g++]

Cross-compile:

```
$ arm-linux-gnueabihf-g++ -o helloworld helloworld.cpp
```

Execute it!

```
$ ./helloworld
bash: ./helloworld : 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!
You can check this by typing

```
$ file helloworld
helloworld: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter….
```

22. Download it via scp (secure copy) and run on Beaglebone

Power on Beaglebone.
Connect USB cable: Power on BeagleBone
Open another terminal for Beaglebone and login as a normal user (your id)
Make a working directory on Bone (With a user, not root)

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

Copy from PC terminal

```
$ scp helloworld {yourid}@192.168.100.17:/home/{yourid}/test_scp
{yourid}@192.168.100.17's password:
```

Check on Beaglebone console:

```
# cd ~/test_scp
# ls
helloworld
```

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
```

Try

```
# ./helloworld
Hello, world!
```

23. Install NFS server on PC

Installl nfs server on PC.

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

24. Configure NFS

Remember IPs for PC and Bone, for example,

PC: eth1 192.168.100.12
Beaglebone: eth0 192.168.100.18

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

```
$ sudo nano /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 - Robot Manipulator
/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
```

25. 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
```

26. Install nfs client in Beaglebone

Install nfs-client software

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

Wait.....

27. Make the mount point on Beaglebone

```
# mkdir ~/nfs_client
```

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

28. Start Beaglebone nfs client

command: sudo mount {pc_ip}:{pc_directory} {beaglebone_directory}

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

Check your PC IP!

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

```
$ ping 192.168.100.18
```

29. For later use, edit ~/bone_nfs_client.sh using nano

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"
```

Make bone_nfs_client.sh executable.

```
# cd ~
# chmod +x bone_nfs_client.sh
```

Later, you can simply type to start nfs client:

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

30. Go to nfs directory

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

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

31. Run hello_es using nfs

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

```
# ./hello_es
Hello, world!
```

Success?

Problem 1B. Exercise1: Timed loop program

32. Write a counter program that uses chrono library.
    (Designlab/1_CrossDevEnv/b_Counter/counter.cpp)

We’ll use the <chrono> c++ library to deal with the time. I’ll show you three basic examples of how to use it.

Include header in your .cpp file using

```
#include <chrono>
```

Current time point can be obtained by

```
std::chrono::system_clock::time_point now;   // make a time point variable “now”
now = std::chrono::system_clock::now();       // assign current time into the variable
```

Elapsed time between two time points:

```
#include <chrono>

int main(int argc, char *argv[])
{
   std::chrono::system_clock::time_point start, end;
   start = std::chrono::system_clock::now();       // assign current time: start

   … some process…

   end = std::chrono::system_clock::now();       // assign current time: end
   auto elapsed_us = std::chrono::duration_cast<std::chrono::microseconds>(start - end);
   std::cout << elapsed_us.count() << std::endl;  //print out the elapsed time in microsecond
   return 0;
}
```

We recommend that you fully understand the code above before solving the problem 1.b.

Problem 1.b: Please write a countdown program that outputs the following:

```
$ ./countdown
Countdown !
10
9
8
7
6
5
4
3
2
1
0
End
```

The recommended pseudocode for timed loop is as follows:

1. make a time point “prev” and save current time
2. in a while loop,
    a. calculate elapsed time between current time and prev
    b. if elapsed time >= 1 sec, print out counter and assign current time as prev
    c. if counter is less than 0, break

There are countless ways to implement this countdown timer, but for the sake of continuity with the following lab session, please follow this pseudocode.

33. Compile the program and run on PC

```
$ g++ -o counter counter.cpp
$ ./counter
```

34. Cross-compile the program and run on Beaglebone

Please refer to the previous steps! If you are done, show it to the TA

Problem 1C. Exercise2: User input program

35. Install VSCode (or preferred text editor / IDE)

Download link(https://code.visualstudio.com/download)

If you have another preferred IDE/text editor, you can also use it. In this class, we will be using VSCode as the basis for conducting the lessons.

To open current directory in Code, just type

```
$ code .
```

. (dot) means the current path.

In the left tab of VSCode, click on "Extensions" and download the following extensions: C/C++, C/C++ extension pack, CMake, CMake tools.

36. Install CMake

CMake is a cross-platform build management tool that automates the process of compiling and linking software projects on different platforms and with different compilers, making it easier for developers to build, test, and deploy their code.

Install CMake

```
$ sudo apt install cmake
```

37. Configure VSCode CMake

First, make new folder.
(DesignLab/1_CrossDevEnv/c_Timer/)

Open VSCode at the folder

```
$ cd DesignLab/1_CrossDevEnv/c_Timer
$ code .
```

In the Command Palette (ctrl+shift+p), click on "CMake: Quick start".
Enter the name of the new project “timer” and select executable.
Three items will automatically be generated inside the folder.

- Build (folder):
  This is a temporary storage place for the files needed for compilation (it can be deleted without any problem). The compiled executable file is also generated in this folder.
- CMakeLists.txt:
  CMake is a macro that automatically manages the build process. All the configurations for this are stored in a file called CMakeLists.txt.
- main.cpp:
  You can write your own code here!

At first, the main.cpp file contains the hello world code. To build this, select [CMake: Build] from the command palette (ctrl+shift+p).

After the build is complete, an executable will be generated in the build folder. Let's run it.

```
$ ./build/timer
```

38. Use debugger

We will use the debugger of VSCode. The debugger makes it easy to catch bugs in the code I wrote. First, copy the code below to main.cpp.

```
#include <iostream>

int main(int, char**) {
    std::cout << "1\n";
    std::cout << "2\n";
    std::cout << "3\n";
    std::cout << "4\n";
    std::cout << "5\n";
    std::cout << "6\n";
    std::cout << "7\n";
    std::cout << "8\n";
    std::cout << "9\n";
    std::cout << "10\n";
    return 0;
}
```

The leftmost side of the text editor window displays line numbers, and by clicking next to it, you can create a breakpoint. Breakpoints pause the program's execution temporarily for debugging purposes.

Then, go to the command palette and run "CMake: Debug". During debugging, you can continue, step through line by line, unit run (execute a function at once), and restart.

A detailed explanation will be conducted during the TA's class time.

39. Configure the source code

Typically, source code is put in the src folder and header files are put in the include folder. Here, the header is a file with the .h extension that helps to prevent duplicated declarations of the same variable when used in multiple .cpp files.

The given files, getche.cpp and getche.h, contain functions getch() and getche() that receive keyboard inputs and store them in variables. Let's run an example in main.cpp that uses these functions.

- Create src and include folders.
- Put getche.h in include and getche.cpp and main.cpp in src folder.
- Write main.cpp as follows.

```
#include <iostream>
#include "getche.h"

int main(int, char**) {
    char a = getch();      //This function pauses the process until enter is pressed.
    std::cout << a << std::endl;
    return 0;
}
```

- Modify CMakeLists.txt to inform CMake about the file structure of the source code.

```
cmake_minimum_required(VERSION 3.0.0)
project(timer VERSION 0.1.0)

include(CTest)
enable_testing()

include_directories(include)
add_executable(timer src/main.cpp src/getche.cpp)

set(CPACK_PROJECT_NAME ${PROJECT_NAME})
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
include(CPack)
```

- Perform the command palette CMake: Build.

40. Write a timer program in PC

We'll use getch() function to receive user input. The difference between getch and getche is that getch does not display keyboard input. Please write a timer program that outputs the following:

```
Start! Press 's' to stop
```

```
f: Wrong input            //if wrong input(f), print this
q: Wrong input            //if wrong input(q), print this
Elapsed time(sec): 4.8924   //if s, print this and elapsed time in second (float)
```

41. Cross-compile the program and run on Beaglebone

Cross-compile in VSCode is pretty easy.
In the taskbar of VSCode below, there is a part that sets the compiler (example: GCC 9.4.0 x86_64-linux-gnu).

Click it and change it to GCC 9.4.0 arm-linux-gnueabihf. Then, perform the build using the CMake: Build command, just like before, to generate a ARM binary.