Lab1 Experiment Guide:
Embedded Board and Development Environment Setup

Objectives

The goal of this lab is to establish and test a cross-compiling system for developing embedded systems on a PC-based environment. We test various aspects: Ubuntu Linux for PC, Debian Linux for BeagleBone, C code build system (CMake), and cross-compile, NFS, and GUI tool (VSCode).

- Get familiar with using Linux with Bash shell
- Complete basic configuration of Linux using SSH (network, NFS)
- Understand the process of compiling and cross-compiling C code
- Practice basic debugging.

Problem statement

The content of this lab is to establish a cross-development system and implement and run an embedded application example using C language. We will also implement a simple timer application.

Problem 1A.

Set up a cross-development system on the lab PC (or personal laptop PC). Write and cross-compile a simple "Hello, world!" code. Use NFS and SSH to download the program to the embedded system and run it.

Problem 1B.

Countdown program: print the countdown from 10 to 0 using a timed loop of 500Hz. (The detailed requirements of the program is described on the Lab Procedure document)

Problem 1C.

Reaction timer program: measure the time interval between two keyboard inputs. The requirements for the program are as follows:

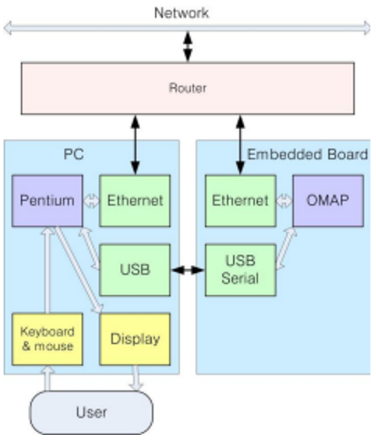
- When the program starts, the computer will output the following message: "Type 's' key to start timer."
- Pressing the 's' key will start the timer, otherwise, it will not be performed.
- When the timer starts, the following message will be output: "Type 'e' key to end timer."
- Pressing the 'e' key will end the timer. When the timer ends, the timer's execution time and loop counter will be output.

Backgrounds

Hardware components of Cross Development Environment

1. Hardware connection

In this experiment, the PC and embedded board are connected through a router as shown in the following picture.



2. Router

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).

1. **Upper connection**

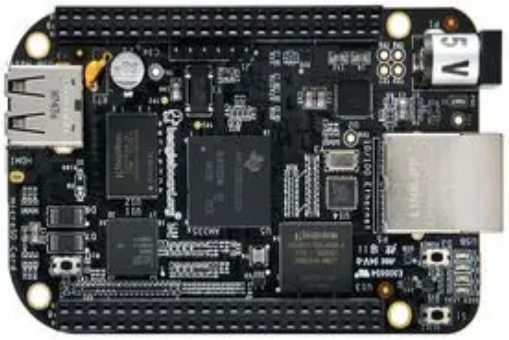
For the upper connection from the router to KAIST network, we use a registered static IP with four bytes for KAIST network, such as 143.248.aaa.bbb (both aaa and bbb bytes ranging from 1 to 255, according to IPv4 (Internet Protocol version 4)). Note that this is a human-readable form of four bytes IP 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.

2. **Lower connection**

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.



- **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, 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!

Software components of Cross Development Environment

1. Operating system (OS)

- **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.

- 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

2. Linux OS

- Ubuntu on PC [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.

- Debian on Beaglebone

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

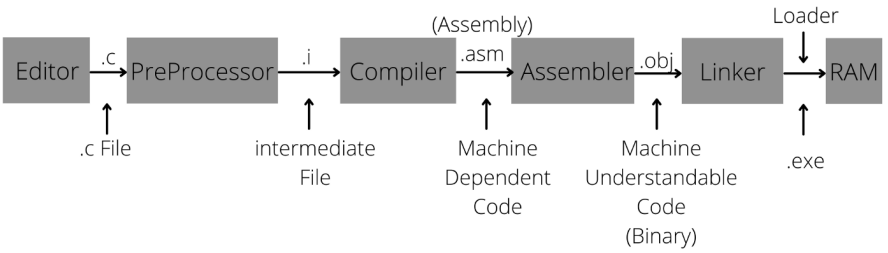
Install the latest debian image (See lab procedure section).

3. 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. Note that 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.

Overall flow of the development process is illustrated below.



- Editor: In order to edit source program, we need a suitable text editor to write a code.
- Preprocessor: The preprocessor takes the source code as input and performs macro substitution, conditional compilation, and inclusion of header files. It generates a preprocessed code file.
- 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.
- 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.
- 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).

4. 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 is shown below.

```
#include <iostream>
void main()
{
        std::cout << "Hello, world! << std::endl;
}
```

This program prints a message using cout, which is an object to print variables, string, etc, defined in iostream. 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.

5. How to send binary files to Beaglebone (scp, NFS)

The following will be used to move the executable files compiled from the PC to the target system (beaglebone).

- 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**(**https://en.wikipedia.org/wiki/Secure_Shell**)** for data transfer and provides the same authentication and same level of security as **ssh**.

```
# Copy the file “foobar.txt” from the local host to a remote host
# scp {filename} {username}@{remote_ip_address}:{file_location}

# Copy the file "foobar.txt" from a remote host to the local host:
# scp {username}@{remote_ip_address}:{file_location} {host_file_location}

# For directory, add option “-r” (example: scp -r …)
```

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]

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.

6. Debugger

- When only the command line interface is available (like Embedded board)

Gdb – The GNU debugger – is most widely used.
Refer to the following document: How to Debug C Program using gdb in 6 Simple Steps (http://www.thegeekstuff.com/2010/03/debug-c-program-using-gdb)

- When the GUI is available, you can use any of the following text editors or IDEs(integrated development environments)

IDE: Visual studio, Eclipse, Xcode, CLion, …

Text editor: **Visual studio code**, Atom, Sublime text, Vim, …

In this session, This lab session will teach you the basic debugger usage of visual studio code.

7. Overview of time and timers

- Real time and process time

Real time is defined as time measured from some fixed point, either from a standard point in the past , or from some point in the life of a process (elapsed time). We'll check the elapsed time using the program in Problem 1.C.

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). We'll check the process time of a function on the next lab session (Lab2, GPIO).

- 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.

- Chrono library

The <chrono> library is a part of C++ that helps programmers work with time. It has tools for measuring how long things take and for figuring out what time it is now. The main parts of the library are duration and time_point. A duration is how long something takes, like a second or a minute. A time_point is a specific moment in time, like right now. The library also has different types of clocks that can help measure time. Some clocks are more precise than others.

The most important parts of the <chrono> library are **duration** and **time_point**. A duration is like a length of time, such as one second or two minutes. It can be used to measure how long things take. A time_point is a specific moment in time, like right now. It can be used to figure out how much time has passed between two different moments.

Some useful functions for our experiment:

- **std::chrono::duration_cast**: Converts a duration to a different duration with a different tick period.
- **std::chrono::system_clock**: A clock that provides access to the current time according to the system-wide clock.

8. 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 will be provided as "**getche.c"**.

9. Compile application with multiple source files

Assume you have two source files and one header: getche.c, getche.h and test_getch.c(your source code will use getch() function in getche.c). How can you compile successfully?

Cross compile with multiple source files and headers:

```
# gcc -o {output file} {source files} -I{header include path}
$ arm-linux-gnueabihf-gcc -o Test_getch Test_getch.c getche.c -Iinclude
```

Or, you can use CMake for doing this automatically. We'll give you a very basics of CMake setting for our experiments. For more details, please refer to other explanations or tutorials, like this one (https://riptutorial.com/cmake)

Preparation

List of components

- Router and an ethernet cable
- IBM PC with Linux
- Beaglebone set: Beaglebone embedded board, USB cable, 1 ethernet cable, 4(or More) GB microSD card, microSD card reader
- getche.c, getche.h (week2)

Lab Procedures

- See the documentation “EE405 Lab1 Procedure.”

Final Report

Each student must write a final report for each experiment. Especially, if the content of the discussion is similar to that of other students, it will result in a deduction or a score of 0. The final report must include the following items:

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

The discussion questions for Lab1 experiment are as follows:

1. What is the difference between the development system and the target system?
2. Is it possible to native-compile on BeagleBone? Describe the advantages and disadvantages of native-compiling and cross-compiling.
3. Can the timed loop in the countdown program ensure a pre-defined time interval (hard real-time)? If not, what is the reason and what are the ways to ensure it?
4. Verifying your code is very important for developing programs rapidly. What methods did you use to get your code to work on the BeagleBone, and are there any alternatives?
5. Choose your own discussion topic related to Lab 1 and provide an answer to it.

References

[1] Beaglebone System Reference Manual, https://cdn-shop.adafruit.com/datasheets/BBB_SRM.pdf

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

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