MV 4412 Android 4.0 Compilation

Microvision Co., Ltd.

Document Information

Version	1.0
File Name	MV4412 Android Compilation.doc
Date	2012. 7. 12
Satus	Working

Revision History

Date	Version	Update Descriptions	Editor
2012. 7. 12.	V1.0	First Edition	Microvision

Contents

1. Package for Development

The following packages are in the directory /SRC/Android in the CD:

File	Description	Version
u-boot-mv4412-dev.tar.gz	Bootloader	
mv4412-kernel-3.0.15.tar.gz	Kernel	3.0.15
ics-mv4412-4.0.3_0705.tar.gz	ICE Cream Sandwich	4.0.3
arm-2009q3-67-arm-none-linux-gnueabi.bin	q3-compiler	Q3 67

Development Environment

We use Ubuntu 10.04.1 for the test environment of Linux.

Other OS or version of Ubuntu could make problem in the compilation process.

Tool chain

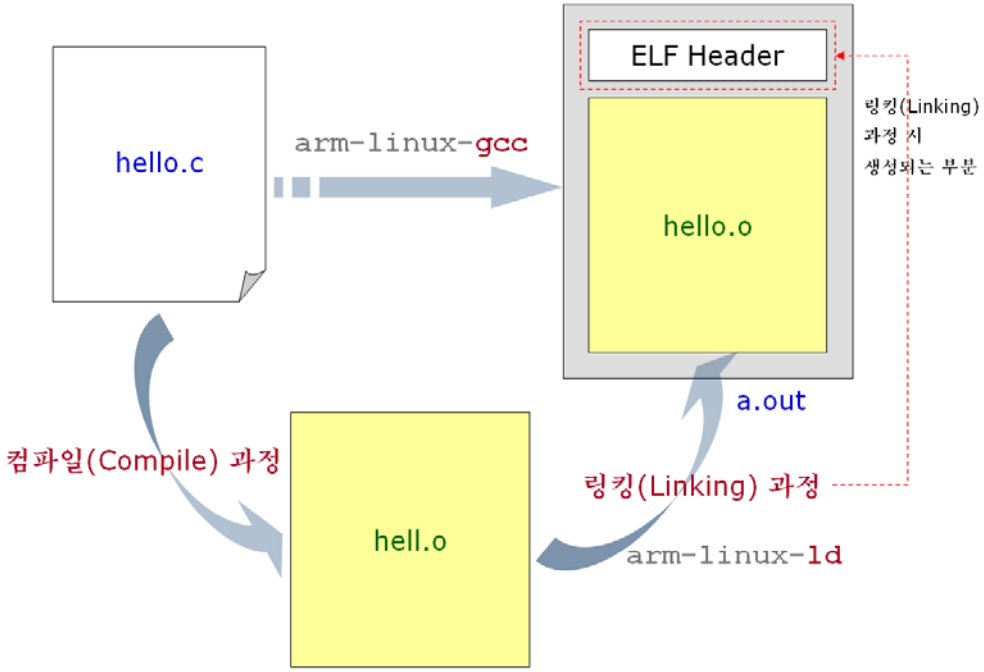
- This Android4.0.3 BSP uses Q3 Compiler for compilation.

(How to install Q3 is described on page 12.)

2. GCC (Tool chain) Setup

In order to develop MV-4412 Linux BSP, a tool that can compile the boot loader and kernel sources and also make the desired output files, such as ELF or BIN files, is required. All of these tools combined makes up the “Tool chain”.

In order to get the desired files from compiling the source code, you need a system library and utilities in addition to the compiler. Please refer to the following diagram to aid your understanding:



<Source Compilation and Linking Process>

Referring to the diagram above, the compilation process has two parts: compiling process and linking process. Each of the processes are explained as such:

Compiling Process:

-Involves a preprocessing process, such as #include, #define

-Checks the source code for syntax errors. If no errors are found, the actual compiler called cc1 in the gcc compiler compiles the hello.c source code and generates the object file, hello.o

링킹(Linking) 과정

-The compiler compiles hello.o and makes the file “Relocatable ELF” which cannot run by itself. Therefore, in order to make the “relocatable ELF” file run by itself, we need to bring in information from the linker before executing the file. Such information includes what CPU Core(Architecture) is and how the program code is loaded in the memory (RAM).

In addition, the Id Linker is referenced from the hello.c source code. The linker automatically adds the call routine of the system library low-level functions, such as open(), read(), and write(). These low-level functions are located in the hello.c source code. This enables the transition from User level to Kernel level.

In order to compile the source, we learned that a compiler and system library are needed. The remaining process is to use the utility to turn the executable ELF file into the executable file. The following are some of the major utilities:

- (arm-linux) objcopy

The a.out file, which an output from the diagram on page 5, can be executed after Linux has been booted. The Loader in the Linux loads the data from the a.out file to the memory, then CPU runs the program, with reference to the ELF header in the a.out file.

System softwares like Boot Loader and Kernel are different from a.out in that it cannot get help from the Loader. Therefore, they cannot run as executable ELF files. As a result, they must be made into binary files, which remove the ELF header files. The utility that must be used here is the (arm-linux) objcopy.

- (arm-linux) nm

This is the utility that shows the Symbol Table from the compiled a.out file.

- (arm-linux) strip

When the compiled a.out file size is too big, this utility is used to reduce the size. The Tool chain required to compile the source code refers to the development environment, which includes cross compiler, system library, and other related utilities. All of these are compressed under the file name 4.3.1-eabi-

armv6.tar.bz2 . Next, we will explain how to install and test the Tool chains. 4.3.1-eabi-armv6.tar.bz2 is used to develop MV-4412-LCDLinux BSP as well.

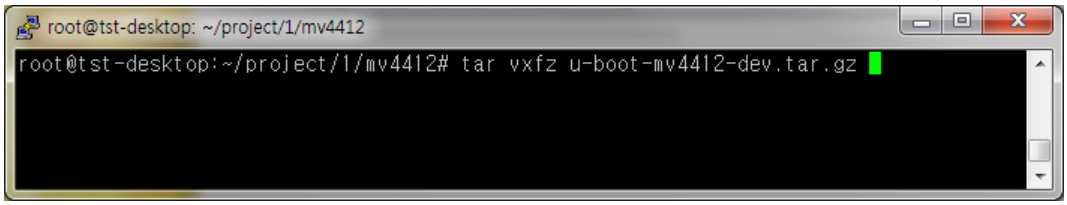
3. Bootloader Setup

3.1. u-boot Environment Setup

Generally, the Embedded Linux BSP is composed of 3 image files:
Embedded Linux BSP = Boot Loader + Kernel + File System
Boot Loader is the program necessary to load the kernel to the memory.

Enter in the following for file decompression:

tar vxfz u-boot-mv4412-dev.tar.gz



As shown below using the “vi” editor, open the file “mv4412.h” and you will find the basic environment at its default. (ex: TFTP, CPU clock, DDR Program Counter)

vi include/configs/mv4412.h

mv4412.h Content

The prompt name on the mv-4412 boot board after booting the new bootloader program:

#define CFG_PROMPT " MV4412 # " /* Monitor Command Prompt */

CPU Clock Configuration (Remove the commend):

#define CONFIG_CLK_ARM_1200_APLL_1200

DRAM Program Counter address for downloading

/* DRAM Base */

#define CONFIG_SYS_SDRAM_BASE 0x40000000

3.2. U-boot Compilation

Install arm-2009q3-67-arm-none-linux-gnueabi.bin which Is in CD→/SRC/Android2.2/q3-compiler

When installing Q3, you must install it on a Linux PC environment, not the console because it is installed by using GUI window.

Installing procedures:

./arm-2009q3-67-arm-none-linux-gnueabi.bin

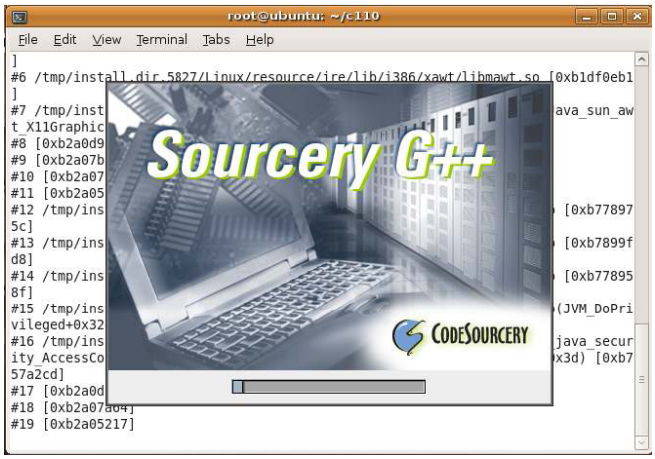
When this message displays "% sudo depkg-reconfigure -plow dash" follow the steps below:

sudo dpkg-reconfigure -plow dash

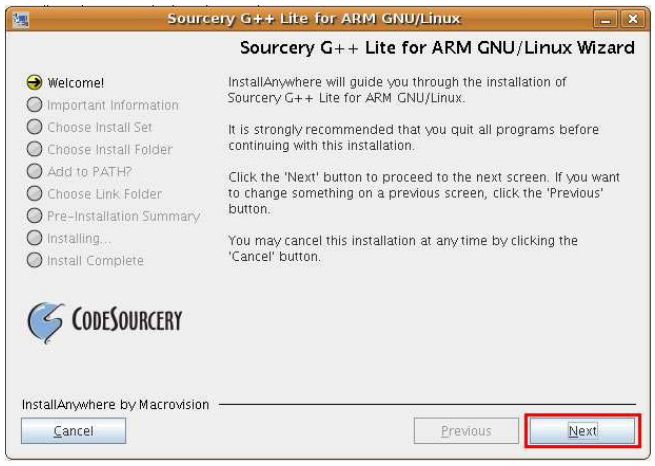
When a [yes/no] screen pops up, click "No" and enter in the command as shown below:

./sudo sh arm-2009q3-67-arm-none-linux-gnueabi.bin

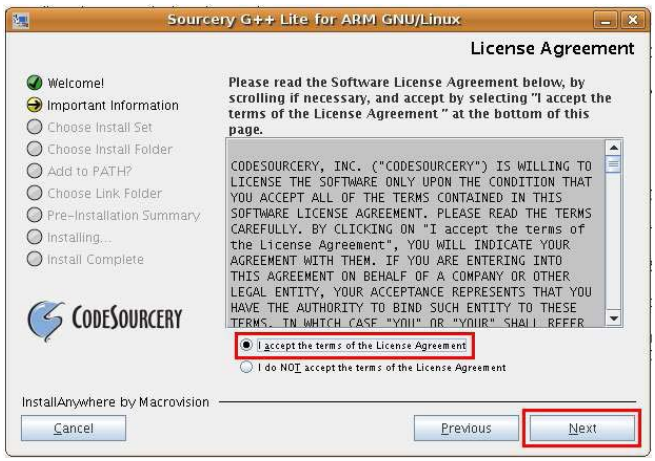
Below is a picture of the loading process:



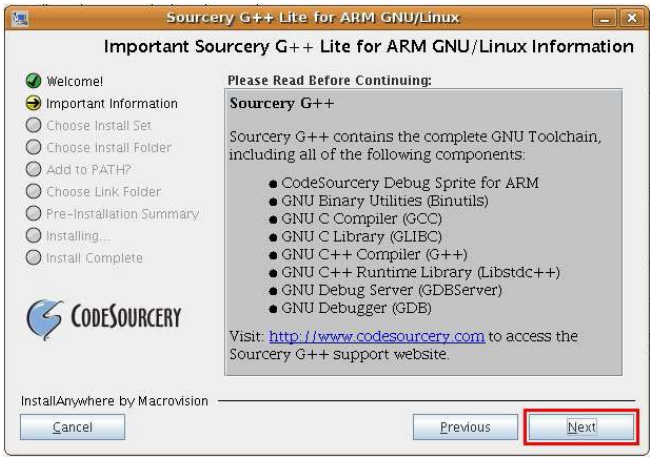
Click “Next”



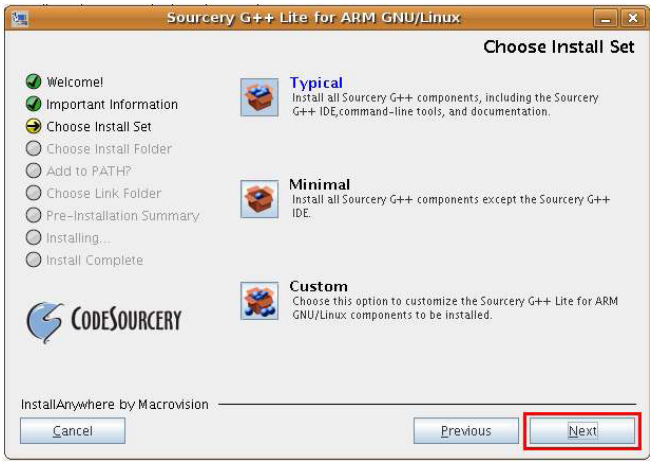
Agree to the terms of the License Agreement then click “Next”



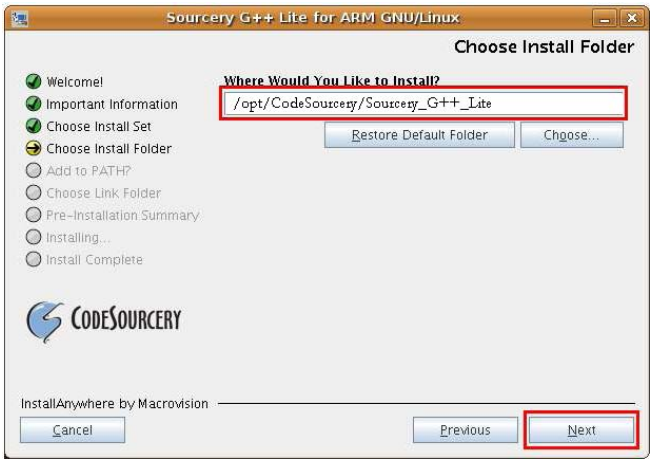
Click “Next”



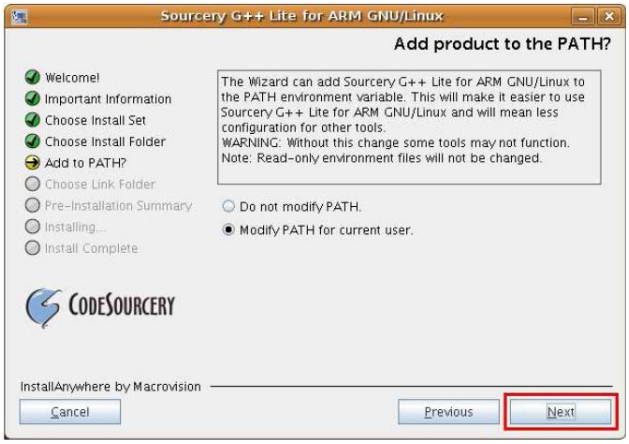
Click “Next”



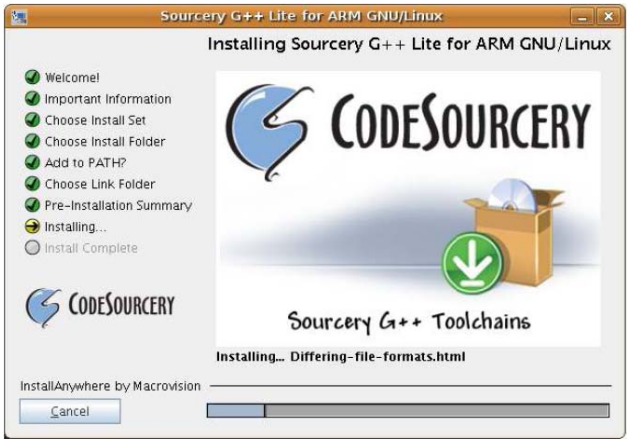
Click “Next”



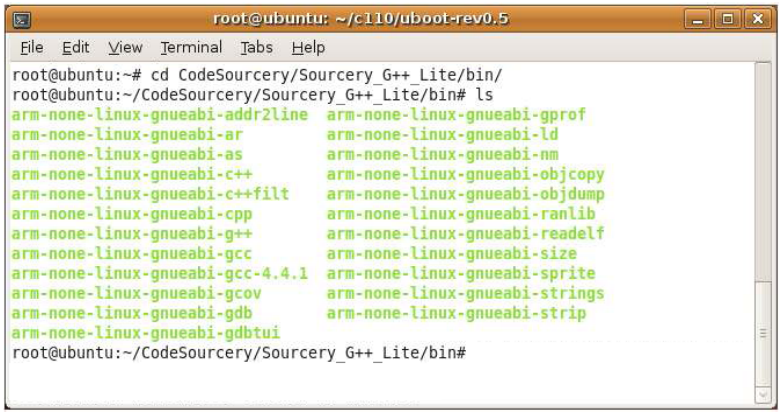
Click “Next”



A series of “Next’s” will lead to the screen as shown below. When the installation is complete, the Shell prompt will run automatically.



When the installation is complete, you can check the Q3 library which has been installed in “/root/CodeSourcery/Sourcery_G++_Lite/bin”



To compile the Boot loader, we will edit the build.sh.

Previously we used the Makefile for the batch job of compilation but we will use the shell script to execute them at one time.

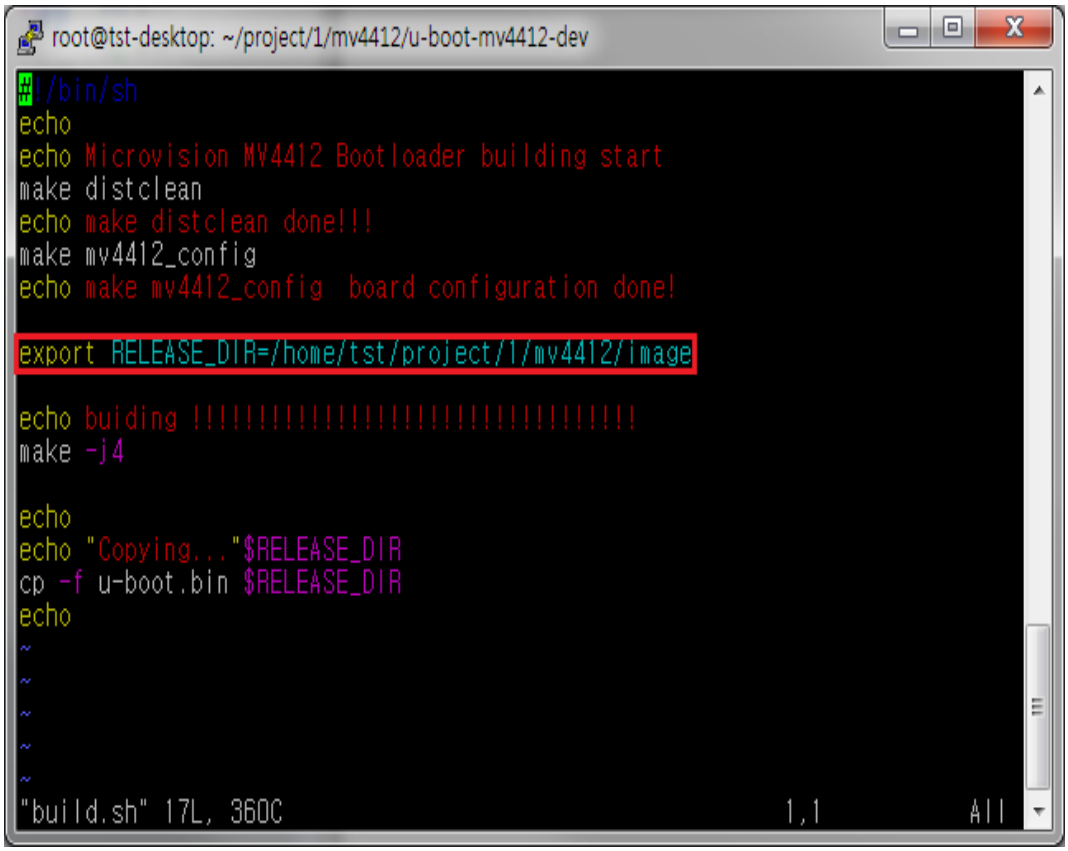
Use the “vi” editor to open build.sh.

vi build.sh

Set the folder path which the compiled image will be copied after compilation.

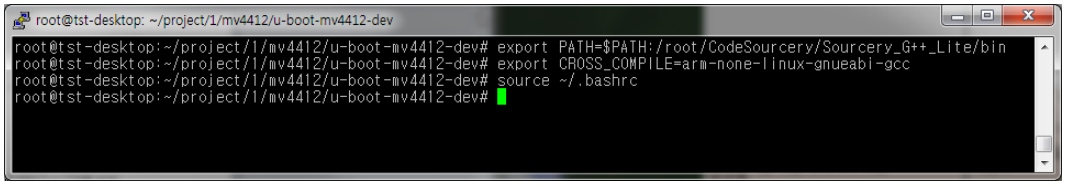
(We set the path of our own purpose.)

export RELEASE_DIR=/home/tst/project/1/mv4412/image



Previously the compiler path is set in the Makefile but MV4412-LCD sets the path using export to continue the compilation after setting the path.

export PATH=$PATH:/root/CodeSourcery/Sourcery_G++_Lite/bin
export CROSS_COMPILE=arm-none-linux-gnueabi-gcc
source ~/.bashrc

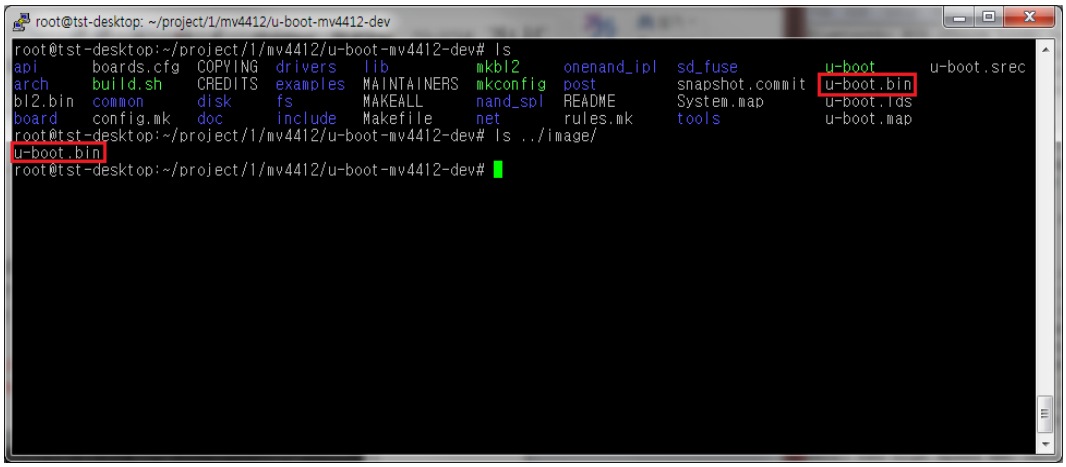


After the path is set as above, execute “./build.sh” .

./build.sh

When compilation is complete, u-boot.bin file is generated in /u-boot-mv4412-dev.

You can find the built image in the same directory which is supposed to setup for it.

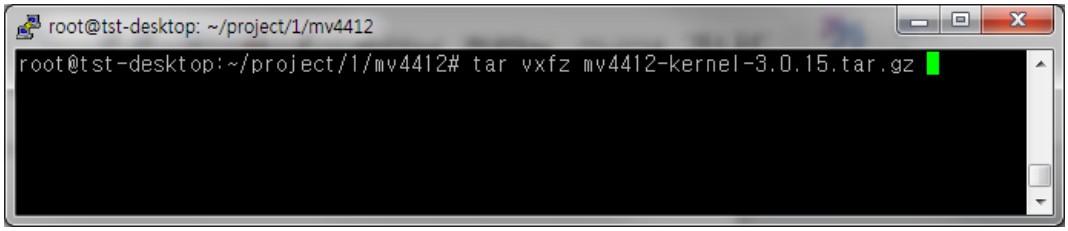


4. Kernel Setup

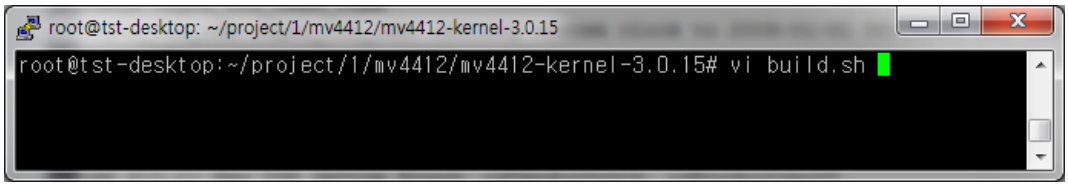
4.1. How to Compile

Enter in the following commands for decompression:

tar vxfz mv4412-kernel-3.0.15.tar.gz

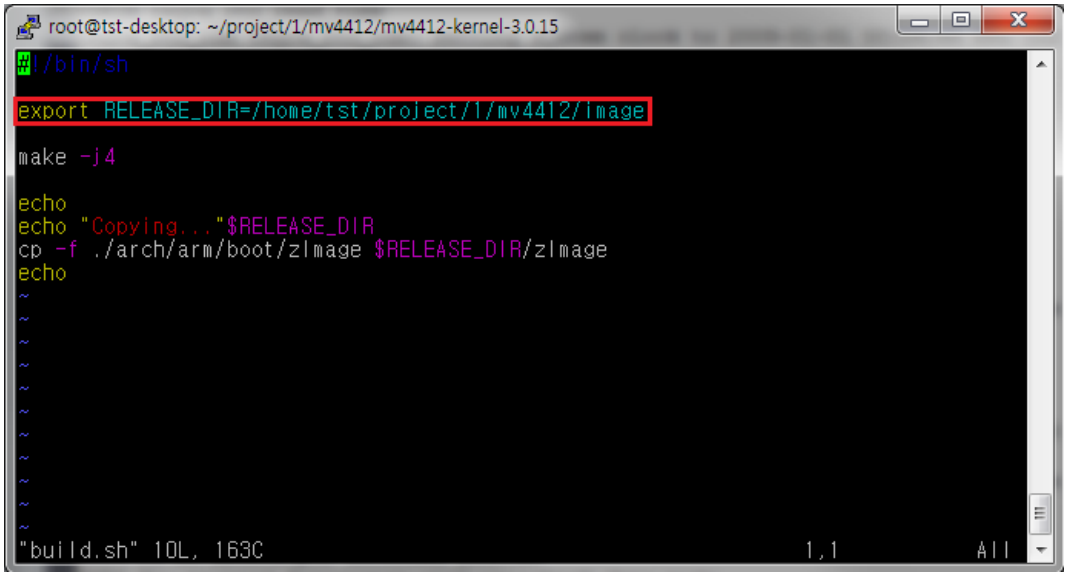


vi build.sh



Set the folder path for being copied with the built image.

Change the folder path just created.



Kernel Source Tree Struncture

MV-4412 Kernel Source Tree	
Documentation/	Technical documents such as Linux HOWTO document

arch/	Provides the sources related to the CPU Core (Architecture) by Linux The sources related to MV-4412 are in arm/→ mach-s5pv310/.
crypto/	Security algorithm supported by Linux
drivers/	Various device drivers supported by Linux
fs/	The file system source supported by Linux
include/	The header file required to compile the Linux source
init/	The sources that are executed when Linux kernel is first initialized. The start_kernel() function, which is called when the Linux kernel is decompressed and first initialized, is located in the main.c file
ipc/	Communication (IPC) algorithms between processes supported by Linux (IPC, Message Queue, Semaphore, Shared Memory)
kernel/	Original sources for Linux kernel. Sources for process management supported by Linux or for interrupt process
lib/	Library required to compile Linux kernel sources
mm/	Memory management algorithms supported by Linux
net/	TCP/IP protocol algorithm supported by Linux
scripts/	ncurses is the script for the display screen (menuconfig) required for set up when the Linux kernel is compiled. The TK script is for GUI setup based on X Window.

Kernel Building Steps

Basically there are two main steps for building the Linux Kernel Image (zImage): Kernel configuration and Kernel Source Compilation.

Kernel Configuration

Kernel generally refers to Operating System(OS) and there are various functions. So Linux helps to configure each of the many functions. The most important part is the CPU configuration. Besides the CPU configuration, the default number needs to be adjusted(plus or minus) depending on hardware features. Another important aspect is that the supporting option needs to be set to the kernel modules, since the insmod command can recognize the device driver's module files (*.o).

Kernel Source Compilation

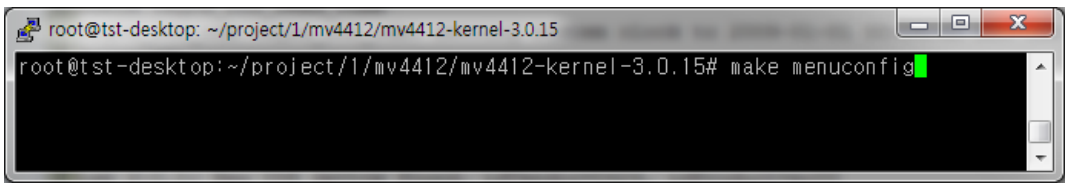
The compilation method after the Kernel configuration is finished includes using the make utility, such as when compiling other source codes. However, the Kernel cannot directly use the Executable ELF, which is the output file from compiling and linking. As a result, it needs to be made into a binary file type by using (arm-linux) objcopy, which removes the ELF header. The final goal is to make the zImage file by decompressing the kernel image using gzip. The compilation command must be "make zimage".

Below is a summary of the buildup process of the Linux kernel image (zImage):

Kernel source→ Configuration →Compiling and linking → objcopy & gzip → zImage

Put in the following commands for compilation to execute the kernel environment setup:

make menuconfig

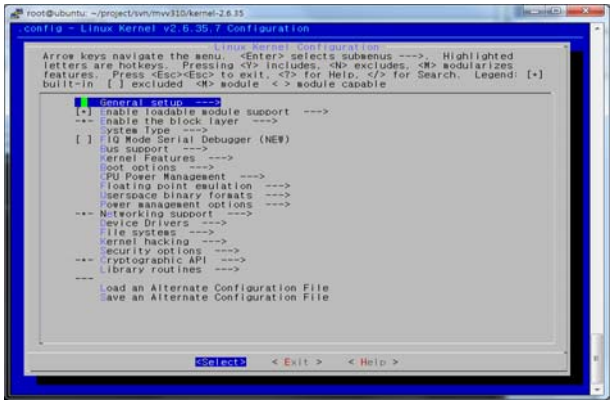


Besides “make menuconfig”, there are kernel commands such as “make config” and “make xconfig”. But we use “make menuconfig” because it is the most popular command which has the same interface such as console(monitor) or telnet terminal with the simple arrow keys.

Once each setup menu is set, it can’t be updated each time. So there is an option to save the configuration into the separate file in the bottom of the screen as the menu “Save Configuration to an Alternate File”. On the other hand to read out the saved configuration, use “Load” command to read the “mv4412_defconfig” file from the kernel source directory, “arch/arm/configs/”.

.

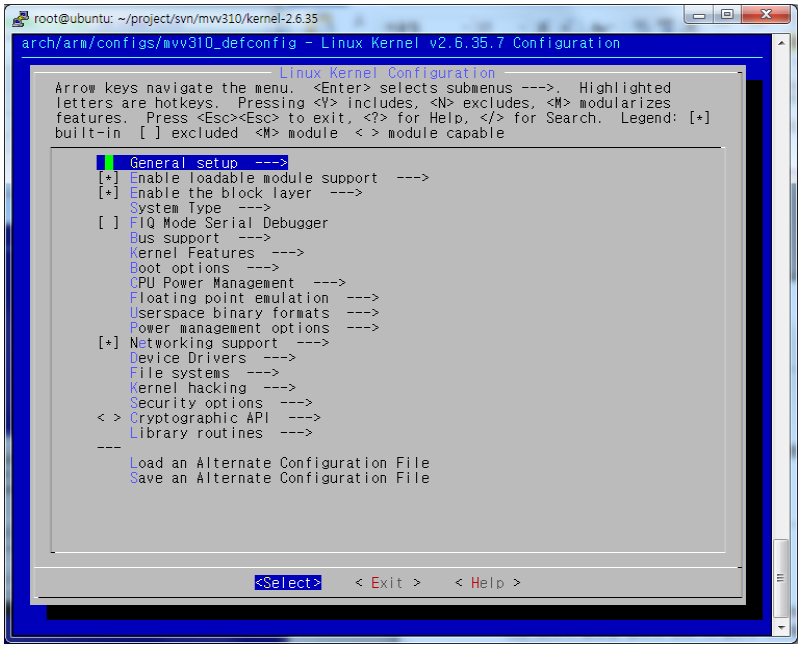
So select the menu “Load an Alternate Configuration File” in the bottom menu of the screen and input as below screen.



Load “arch/arm/configs/ mv4412_defconfig”.

Now all the setup is finished for ending up the kernel configuration (make menuconfig). One thing to be cautious is to save the configuration when the process is finished. The configuration is stored in the filename “config” in the linux kernel source directory. After the kernel

configuration, this file should be in the right place because it is to be verified in the step of “make dep”. Please select “<Yes>” when it’s asked if the configuration is saved or not?.
After checking loading, exit.

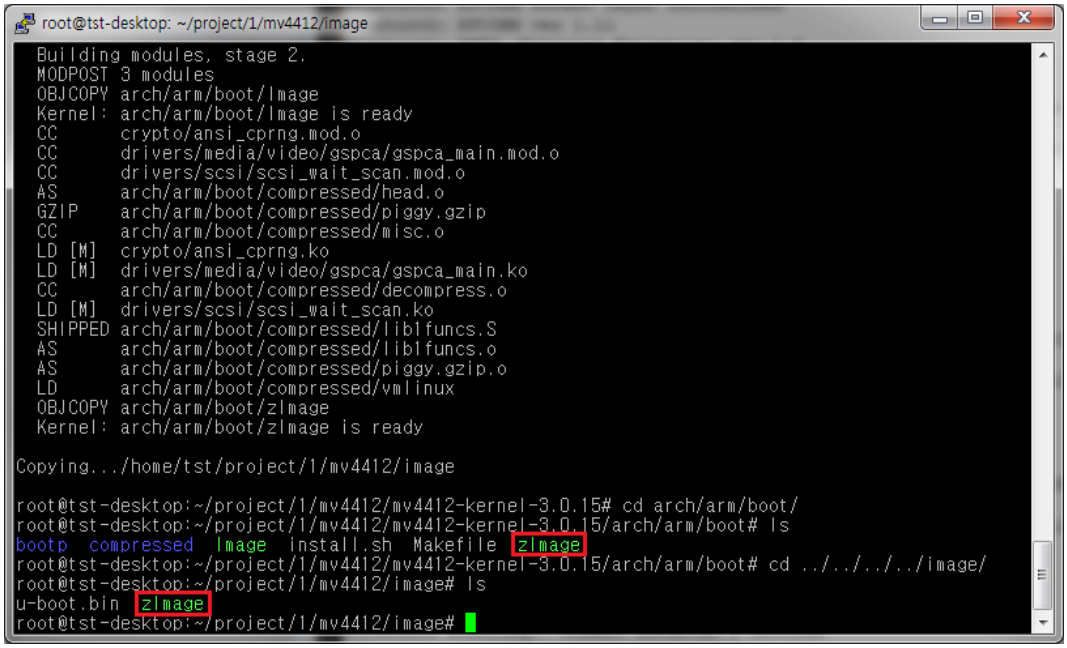


The process of making linux kernel image(zImage) is divided into compilation, linking and file type conversion(ELF -> BIN) by Binutil(objcopy) and file compression(gzip). But for the convenience of development, these processes are in the “make” with the rule of Makefile.

Kernel is made by shell script like u-boot for the compilation in the better way.
Set the compiler path using the below command.

export PATH=$PATH:/root/CodeSourcery/Sourcery_G++_Lite/bin
export CROSS_COMPILE=arm-none-linux-gnueabi-gcc
source ~/.bashrc

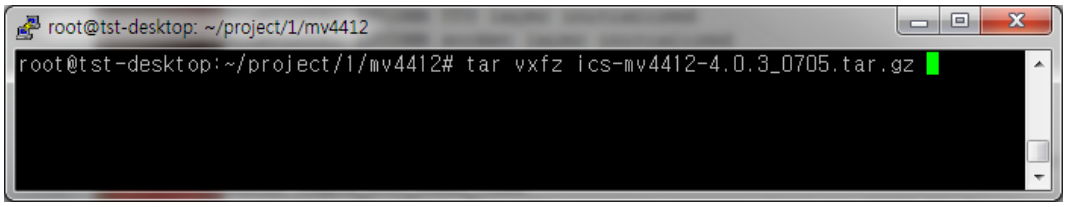
We can see the zImage is built in the folder “arch/arm/boot” after the compilation and can see the copied image successfully located in the directory.


root@tst-desktop: ~/project/1/mv4412/image
Building modules, stage 2.
MODPOST 3 modules
OBJCOPY arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
CC crypto/ansi_cprng.mod.o
CC drivers/media/video/gspca/gspca_main.mod.o
CC drivers/scsi/scsi_wait_scan.mod.o
AS arch/arm/boot/compressed/head.o
GZIP arch/arm/boot/compressed/piggy.gzip
CC arch/arm/boot/compressed/misc.o
LD [M] crypto/ansi_cprng.ko
LD [M] drivers/media/video/gspca/gspca_main.ko
CC arch/arm/boot/compressed/decompress.o
LD [M] drivers/scsi/scsi_wait_scan.ko
SHIPPED arch/arm/boot/compressed/lib1funcs.S
AS arch/arm/boot/compressed/lib1funcs.o
AS arch/arm/boot/compressed/piggy.gzip.o
LD arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
Copying.../home/tst/project/1/mv4412/image
root@tst-desktop:~/project/1/mv4412/mv4412-kernel-3.0.15# cd arch/arm/boot/
root@tst-desktop:~/project/1/mv4412/mv4412-kernel-3.0.15/arch/arm/boot# ls
bootp compressed Image install.sh Makefile zImage
root@tst-desktop:~/project/1/mv4412/mv4412-kernel-3.0.15/arch/arm/boot# cd ../../../../image/
root@tst-desktop:~/project/1/mv4412/image# ls
u-boot.bin zImage
root@tst-desktop:~/project/1/mv4412/image#

5. ICE Cream Sandwich Compilation

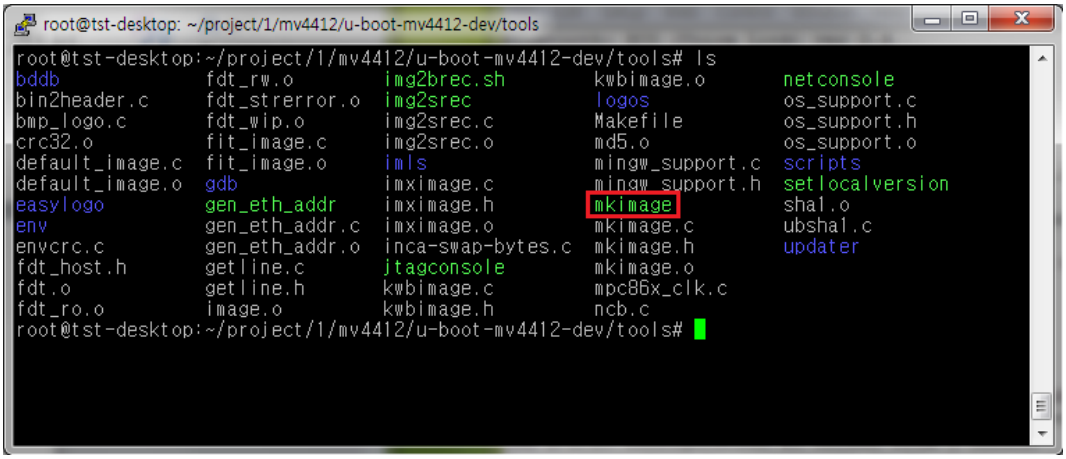
Enter in the following command to uncompress the file.

tar vxfz ics-mv4412-4.0.3_0705.tar.gz



(Caution)

Before compilation, “mkimage” in the image-compiled /u-boot-mv4412-dev/tools must be exported in order to generate the ramdisk properly. Enter in the following command for export:



For developer’s convenience, we created a bin folder inside a temporary folder:

export PATH=$PATH:/home/tst/project/1/mv4412/u-boot-mv4412-dev/tools

source ~/.bashrc

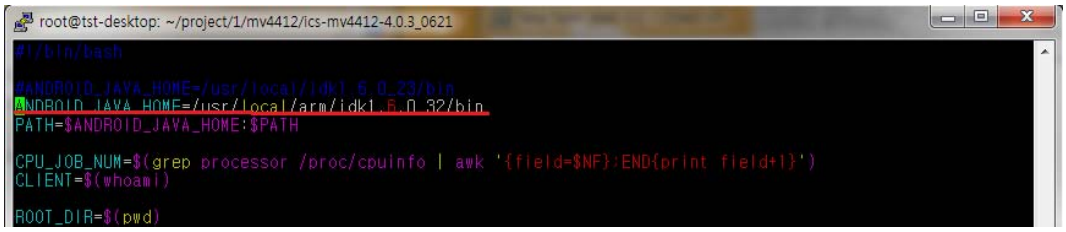
To compile the ICE Cream Sandwich, we need JDK ver1.6. Install it by running the execution

file in the SRC/Android/JDK in the GUI environment.

http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u26-download-400750.html

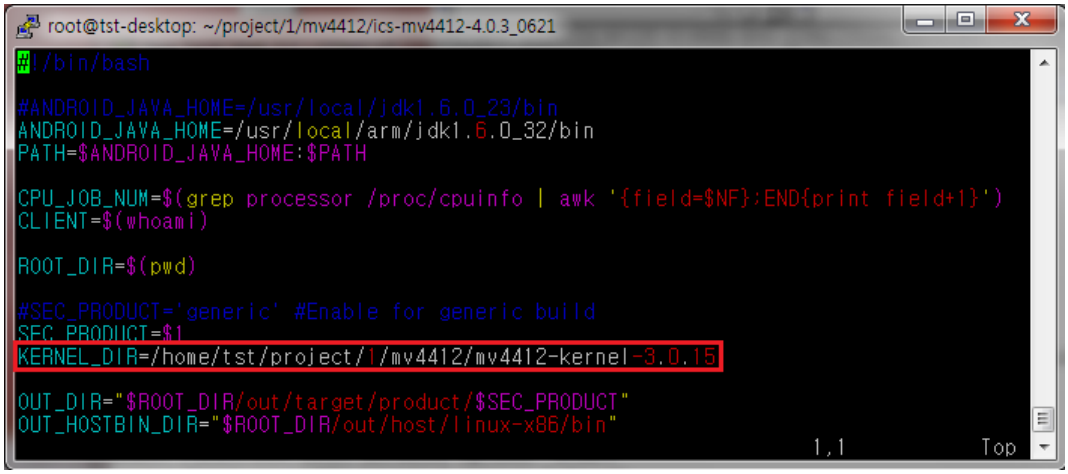
vi build_android.sh

Move to the path to the folder which is installed with JDK.



Kernel Path Setup

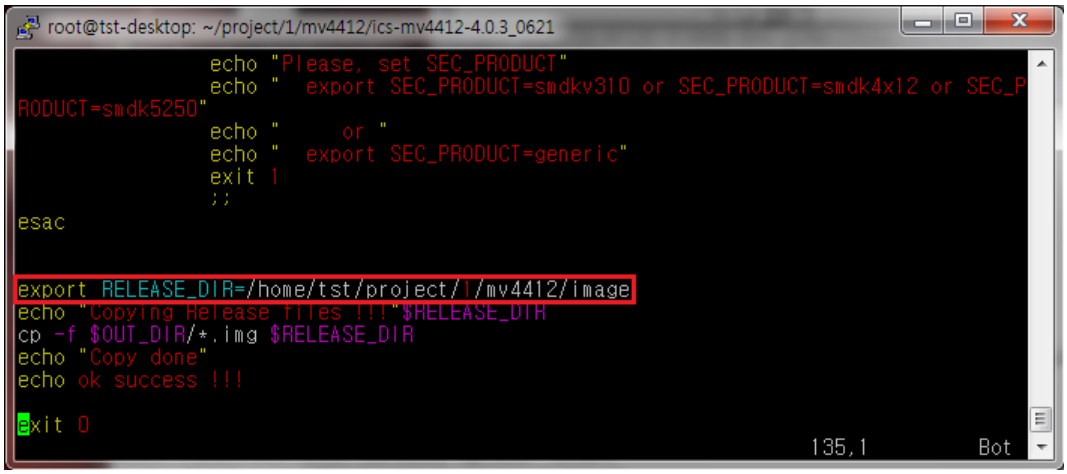
(Move to the path to the kernel folder)



KERNEL_DIR=/home/tst/project/1/mv4412/mv4412-kernel-3.0.15

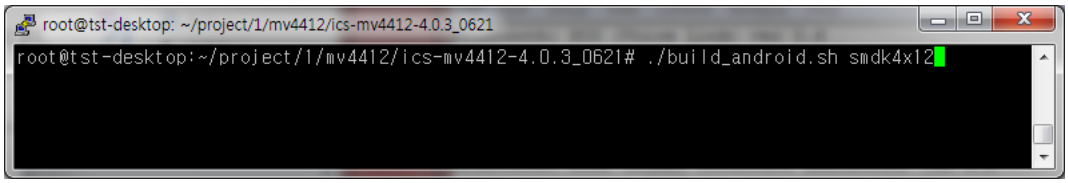
Move to the folder path where the compiled image is located.

(Move to the folder created)

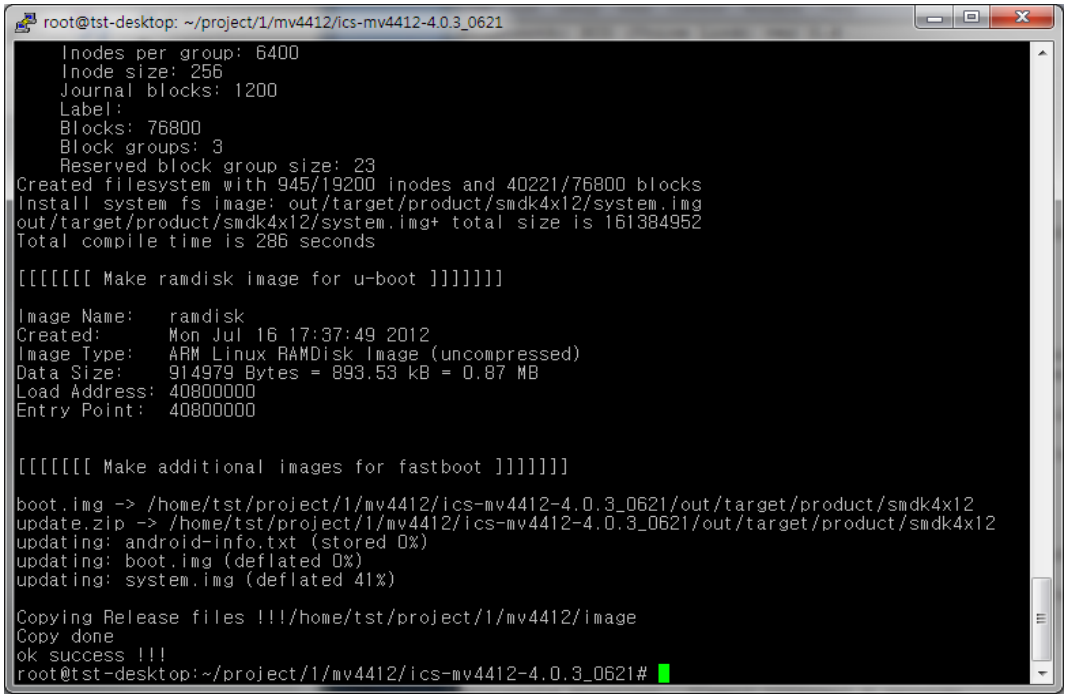


Do the compilation with the below command.

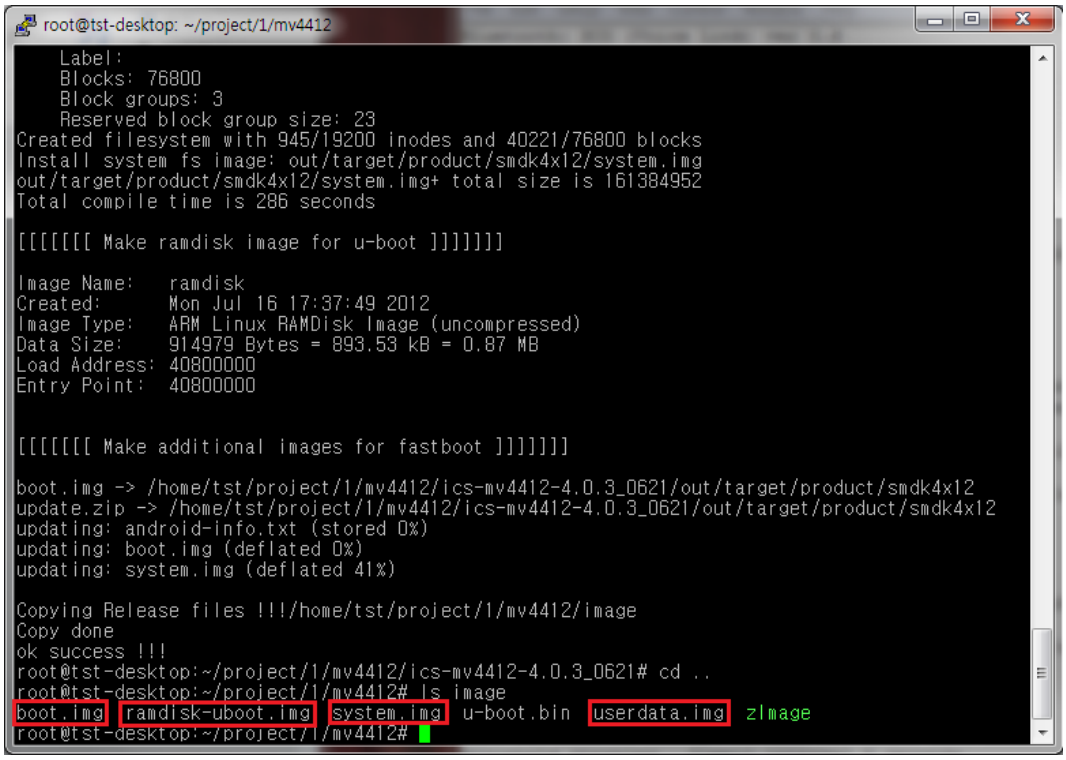
./build_android.sh smdk4x12



Screen with the successful compilation



If the build is complete, we can see the image in the folder where the image is to be located.


root@tst-desktop: ~/project/1/mv4412
Label:
Blocks: 76800
Block groups: 3
Reserved block group size: 23
Created filesystem with 945/19200 inodes and 40221/76800 blocks
Install system fs image: out/target/product/smdk4x12/system.img
out/target/product/smdk4x12/system.img+ total size is 161384952
Total compile time is 286 seconds
[[[[[[[Make ramdisk image for u-boot]]]]]]]
Image Name: ramdisk
Created: Mon Jul 16 17:37:49 2012
Image Type: ARM Linux RAMDisk Image (uncompressed)
Data Size: 914979 Bytes = 893.53 kB = 0.87 MB
Load Address: 40800000
Entry Point: 40800000
[[[[[[[Make additional images for fastboot]]]]]]]
boot.img -> /home/tst/project/1/mv4412/ics-mv4412-4.0.3_0621/out/target/product/smdk4x12
update.zip -> /home/tst/project/1/mv4412/ics-mv4412-4.0.3_0621/out/target/product/smdk4x12
updating: android-info.txt (stored 0%)
updating: boot.img (deflated 0%)
updating: system.img (deflated 41%)
Copying Release files !!!/home/tst/project/1/mv4412/image
Copy done
ok success !!!
root@tst-desktop:~/project/1/mv4412/ics-mv4412-4.0.3_0621# cd ..
root@tst-desktop:~/project/1/mv4412# ls image
boot.img ramdisk-uboot.img system.img u-boot.bin userdata.img zImage
root@tst-desktop:~/project/1/mv4412#