

MV V310 Android 4.0 Compilation



Microvision Co., Ltd.

Document Information

Version	1.0
File Name	MVV310 Android Compilation.doc
Date	2012. 4. 17
Satus	Working

Revision History

Date	Version	Update Descriptions	Editor
2012. 4. 17.	V1.0	First Edition	Microvision

Contents

1. Package for Development 4/25
2. GCC Setup 5/25
 - 2.1 Decompression 8/25
 - 2.2 GCC Environment PATH Setup 8/25
3. Bootloader Setup 10/25
 - 3.1. u-boot Environment Setup 10/25
 - 3.2. u-boot Compilation 11/25
4. Kernel Setup 17/25
 - 4.1. . Compilation 17/25
5. ICE Cream Sandwich Compilation 23/25

1. Package for Development

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

파일	설명	버전
V310_u-boot.tar.gz	Bootloader	
V310_kernel_3_0_15.tar.gz	Kernel	3.0.15
V310_ics.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

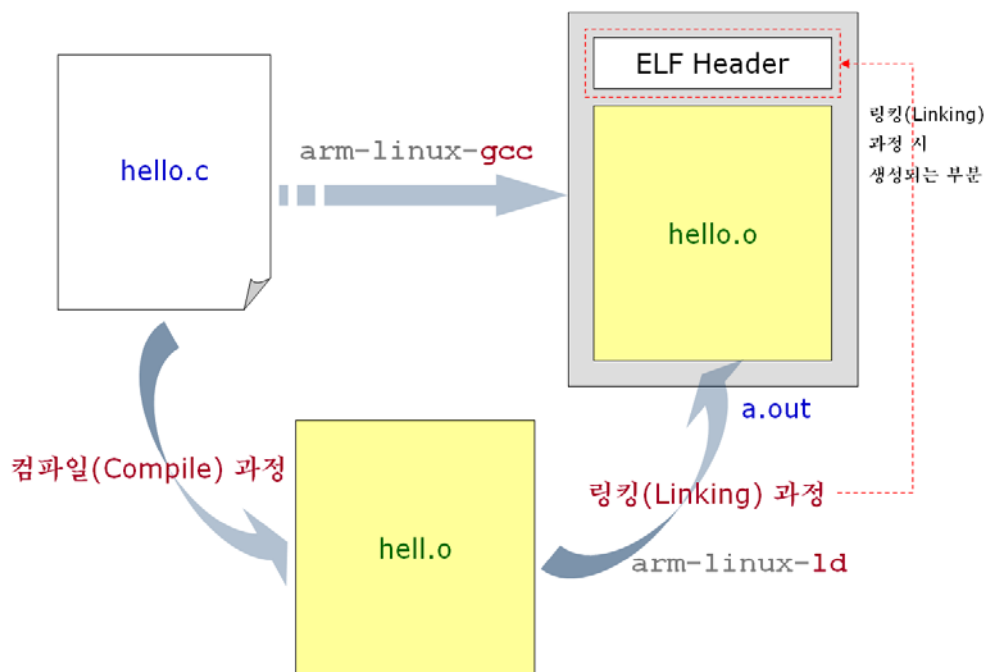
- This Android4.0.3 BSP uses Q3 Compiler for compilation.

(How to install Q3 is described on page 11.)

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

Linking

-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 ld 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 is 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-

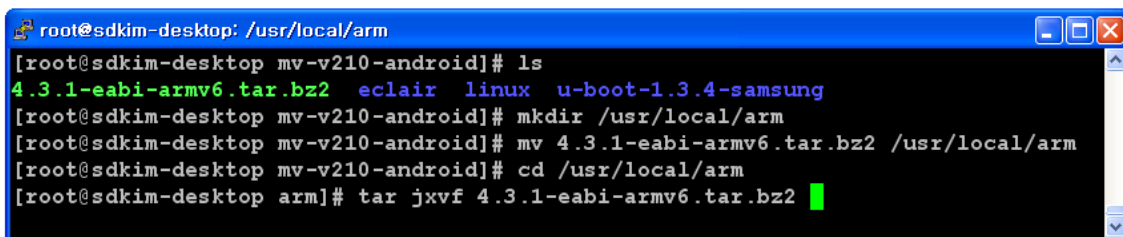
armv6.tar.bz2 . Next, we will explain how to install and test the Tool chains.

4.3.1-eabi-armv6.tar.bz2 is used to develop MV-4412-LCDLinux BSP as well.

2.1 Decompression

After copying the files to Linux server through Samba or FTP, use the following steps to start the decompression process:

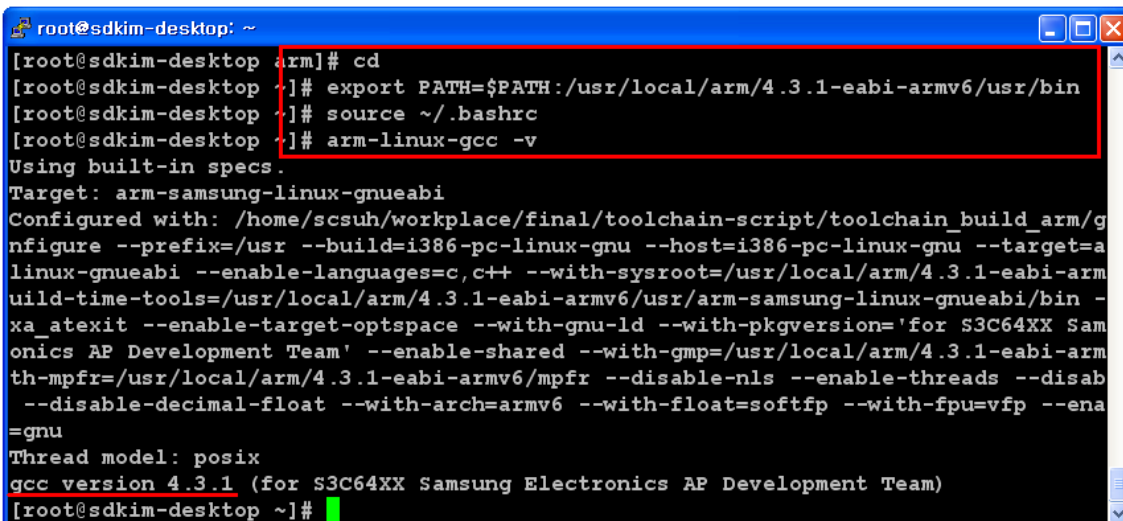
```
# mkdir /usr/local/arm
# mv 4.3.1-eabi-armv6.tar.bz2 /usr/local/arm
# cd /usr/local/arm
# tar jxvf 4.3.1-eabi-armv6.tar.bz2
```



```
root@sdkim-desktop: /usr/local/arm
[root@sdkim-desktop mv-v210-android]# ls
4.3.1-eabi-armv6.tar.bz2  eclair  linux  u-boot-1.3.4-samsung
[root@sdkim-desktop mv-v210-android]# mkdir /usr/local/arm
[root@sdkim-desktop mv-v210-android]# mv 4.3.1-eabi-armv6.tar.bz2 /usr/local/arm
[root@sdkim-desktop mv-v210-android]# cd /usr/local/arm
[root@sdkim-desktop arm]# tar jxvf 4.3.1-eabi-armv6.tar.bz2
```

2.2 GCC Environment PATH Setup

```
# export PATH=$PATH:/usr/local/arm/4.3.1-eabi-armv6/usr/bin
# source ~/.bashrc : Environment Setup
# arm-linux-gcc -v : Check the GCC version
```



```
root@sdkim-desktop: ~
[root@sdkim-desktop arm]# cd
[root@sdkim-desktop ~]# export PATH=$PATH:/usr/local/arm/4.3.1-eabi-armv6/usr/bin
[root@sdkim-desktop ~]# source ~/.bashrc
[root@sdkim-desktop ~]# arm-linux-gcc -v
Using built-in specs.
Target: arm-samsung-linux-gnueabi
Configured with: /home/scsuh/workplace/final/toolchain-script/toolchain_build_arm/gnfigure --prefix=/usr --build=i386-pc-linux-gnu --host=i386-pc-linux-gnu --target=arm-linux-gnueabi --enable-languages=c,c++ --with-sysroot=/usr/local/arm/4.3.1-eabi-armv6 --with-headers=/usr/local/arm/4.3.1-eabi-armv6/usr/arm-samsung-linux-gnueabi/bin --with-libc=libc --with-newlib --with-gmp=/usr/local/arm/4.3.1-eabi-armv6/usr/local/arm/4.3.1-eabi-armv6/mpfr --disable-nls --enable-threads --disable-decimal-float --with-arch=armv6 --with-fpu=vfp --enable-gnu
Thread model: posix
gcc version 4.3.1 (for S3C64XX Samsung Electronics AP Development Team)
[root@sdkim-desktop ~]#
```

< Steps to modify the PATH variables for Tool chain >

In order to recognize the currently-running Shell (bash), we must run the command “source”.

If there are no problems, use the “which” command to check where the installed command is located in the PATH. If the PATH is displayed correctly as shown in the diagram on page 5, the command has been executed successfully. For reference, you can use the “--version” option to check that the currently installed arm-linux-gcc has a version of 4.3.1.

3. Bootloader Setup

3.1. u-boot Environment Setup

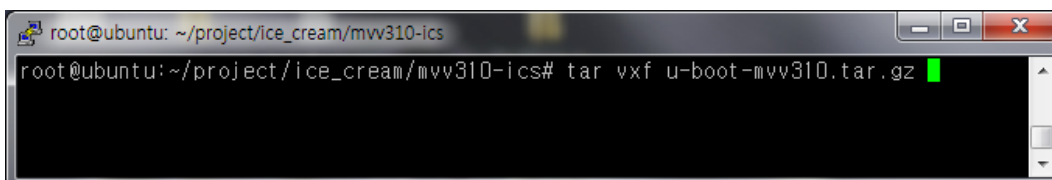
Generally, the Embedded Linux BSP is composed of 3 image files:

Embedded Linux BSP = Boot Loader + Kernel + File System

Boot Loader is the program necessary to load the kernel to the memory.

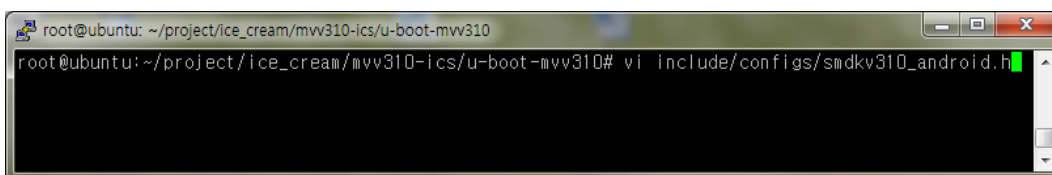
Enter in the following for file decompression:

```
# tar vxf u-boot-mvv310.tar.gz
```



As shown below using the "vi" editor, open the file "[smdkv310_android.h](#)" and you will find the basic environment at its default. (ex: TFTP, CPU clock, DDR Program Counter)

```
# vi include/configs/smdkv310_android.h
```



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

```
#define CFG_PROMPT                "MVV310_ICS # "
```

CPU Clock Configuration (Remove the comment):

```
#define CONFIG_CLK_1000_400_200
```

DRAM Program Counter address for downloading

```
//#define CFG_UBOOT_BASE            0xc3e00000
```

```
//#else
```

```
#define CFG_UBOOT_BASE            0x43e00000
```


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
```

A terminal window titled 'root@ubuntu: ~/c110' showing the following commands and output:

```
test@ubuntu:~$ sudo -s
[sudo] password for test:
root@ubuntu:~# cd c110/
root@ubuntu:~/c110# ls
arm-2009q3-67-arm-none-linux-gnueabi.bin  uboot-rev0.5_c110.tar.gz
uboot-rev0.5
root@ubuntu:~/c110# ./arm-2009q3-67-arm-none-linux-gnueabi.bin
```

The command `./arm-2009q3-67-arm-none-linux-gnueabi.bin` is highlighted with a red box.

When this message displays “% [sudo dpkg-reconfigure -plow dash](#)” follow the steps below:

```
# sudo dpkg-reconfigure -plow dash
```

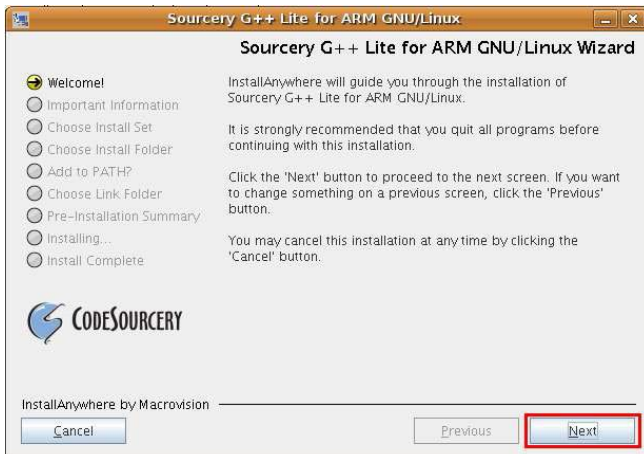
When a [\[yes/no\]](#) screen pops up, click “No” and enter in the command as shown below:

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

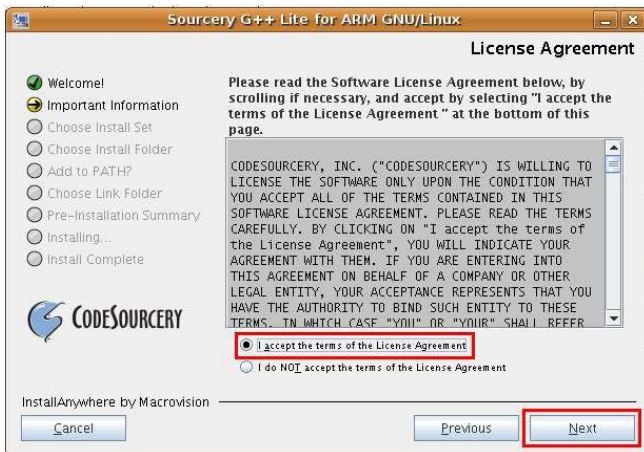
Below is a picture of the loading process:



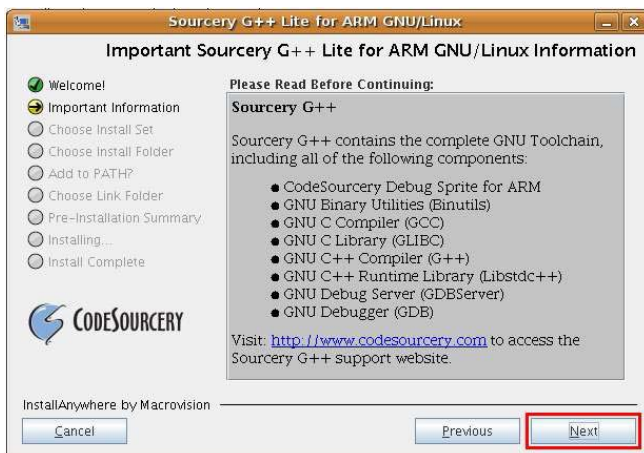
Click "Next"



Agree to the terms of the License Agreement then click "Next"



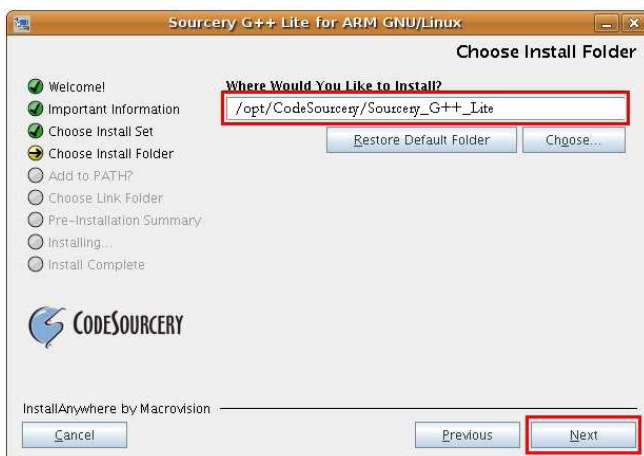
Click "Next"



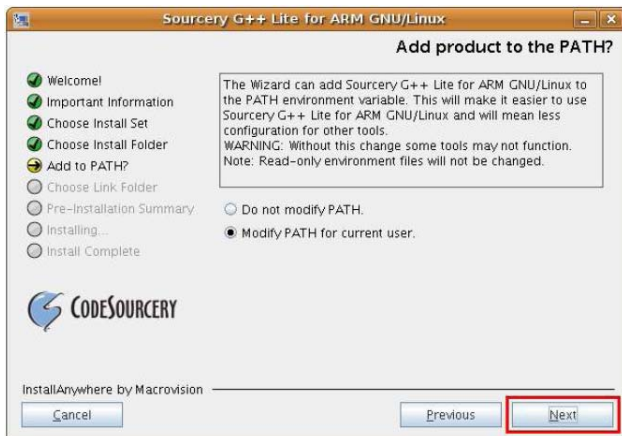
Click "Next"



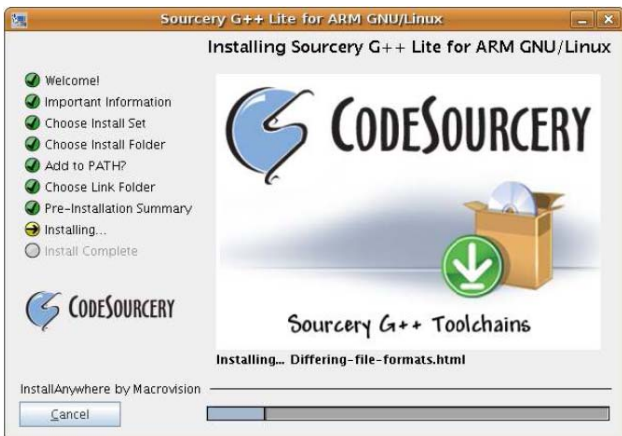
Click "Next"



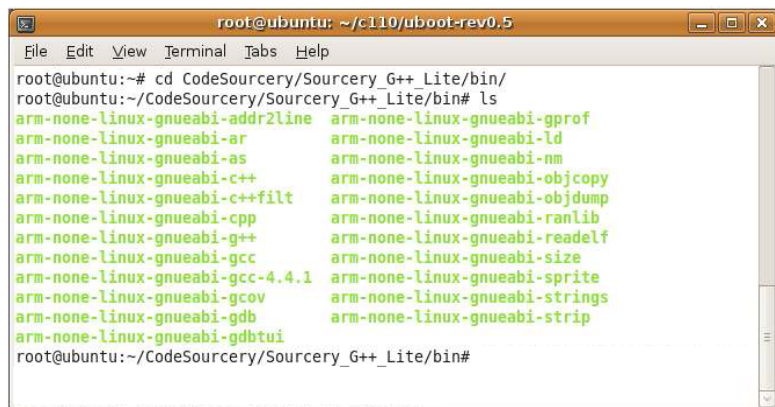
Click “Next”



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”`

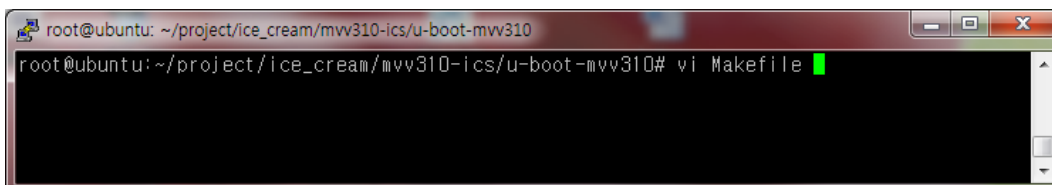


To compile the Boot loader, we will edit the build.sh.

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

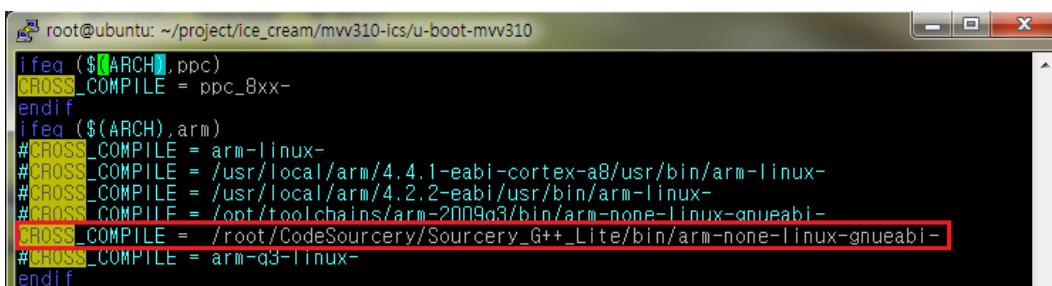
Use the “vi” editor to open Makefiles

vi Makefile



```
root@ubuntu: ~/project/ice_cream/mvv310-ics/u-boot-mvv310
root@ubuntu:~/project/ice_cream/mvv310-ics/u-boot-mvv310# vi Makefile
```

Add “/opt/CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-“ to (\$(ARCH), arm)



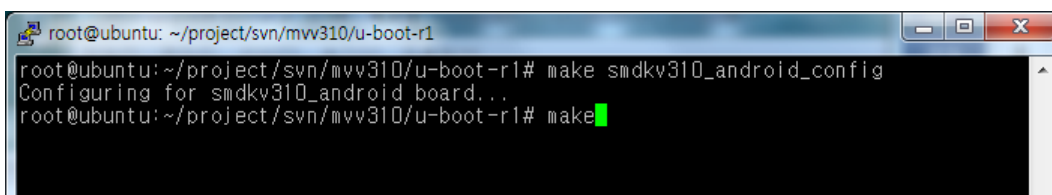
```
root@ubuntu: ~/project/ice_cream/mvv310-ics/u-boot-mvv310
ifeq ($(ARCH),ppc)
CROSS_COMPILE = ppc_8xx-
endif
ifeq ($(ARCH),arm)
#CROSS_COMPILE = arm-linux-
#CROSS_COMPILE = /usr/local/arm/4.4.1-eabi-cortex-a8/usr/bin/arm-linux-
#CROSS_COMPILE = /usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-
#CROSS_COMPILE = /opt/toolchains/arm-2009q3/bin/arm-none-linux-gnueabi-
CROSS_COMPILE = /root/CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-
#CROSS_COMPILE = arm-q3-linux-
endif
```

Compilation Steps: (after compilation, type in command # [make distclean](#))

[make clobber](#)

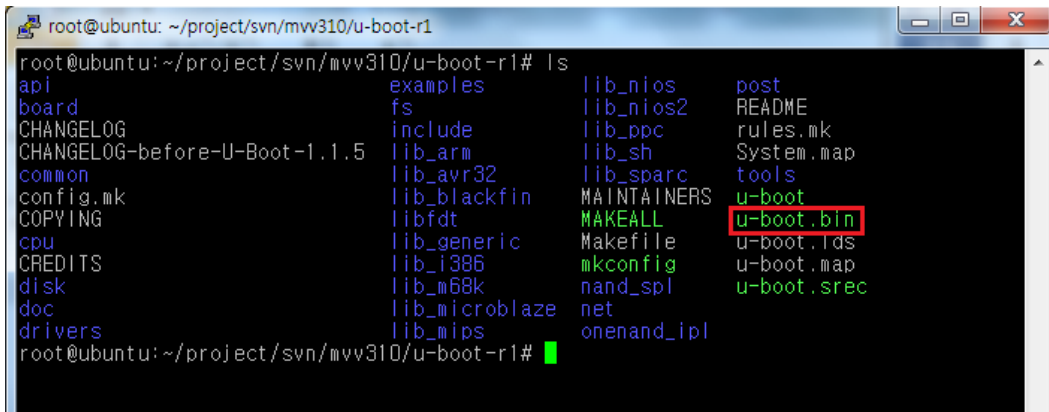
[make smdkv310_config](#)

[make](#)



```
root@ubuntu: ~/project/svn/mvv310/u-boot-r1
root@ubuntu:~/project/svn/mvv310/u-boot-r1# make smdkv310_android_config
Configuring for smdkv310_android board...
root@ubuntu:~/project/svn/mvv310/u-boot-r1# make
```

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



```
root@ubuntu: ~/project/svn/mvv310/u-boot-r1
root@ubuntu:~/project/svn/mvv310/u-boot-r1# ls
api                examples          lib_nios          post
board             fs               lib_nios2        README
CHANGELOG         include          lib_ppc          rules.mk
CHANGELOG-before-U-Boot-1.1.5 lib_arm          lib_sh           System.map
common            lib_avr32        lib_sparc        tools
config.mk         lib_blackfin    MAINTAINERS     u-boot
COPYING           libfdt           MAKEALL          u-boot.bin
cpu               lib_generic     Makefile         u-boot.lds
CREDITS           lib_i386        mkconfig         u-boot.map
disk              lib_m68k        nand_spl        u-boot.srec
doc               lib_microblaze  net
drivers           lib_mips        onenand_ipi
root@ubuntu:~/project/svn/mvv310/u-boot-r1#
```

4. Kernel Setup

4.1. How to Compile

Kernel Source Tree Structure

MV-V210 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-V210 are in arm/→machs5pv210/.
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

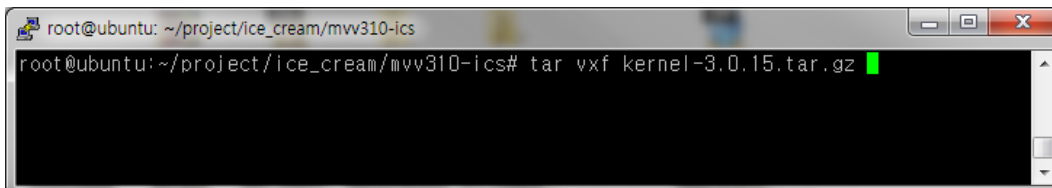
The compilation method after the Kernel configuration is finished includes using the make utility, such as when compiling other source codes. However, the Kernel cannot directly use the Executable ELF, which is the output file from compiling and linking. As a result, it needs to be made into a binary 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

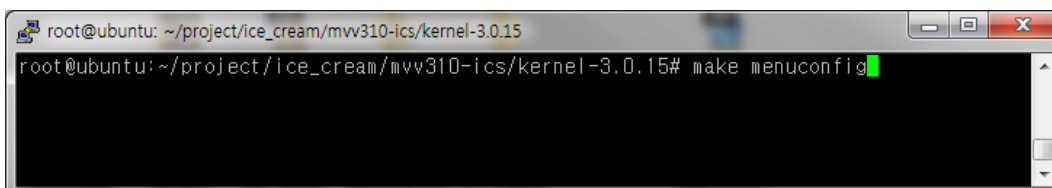
Enter in the following commands for decompression:

```
# tar vxf kernel-3.0.15.tar.gz
```

A terminal window titled 'root@ubuntu: ~/project/ice_cream/mv310-ics' showing the command 'tar vxf kernel-3.0.15.tar.gz' being executed. The command prompt is 'root@ubuntu:~/project/ice_cream/mv310-ics#' and the output is a green cursor.

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

```
# make menuconfig
```

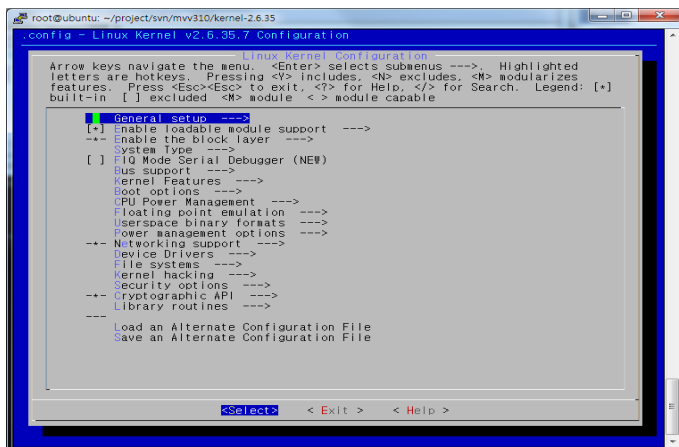
A terminal window titled 'root@ubuntu: ~/project/ice_cream/mv310-ics/kernel-3.0.15' showing the command 'make menuconfig' being executed. The command prompt is 'root@ubuntu:~/project/ice_cream/mv310-ics/kernel-3.0.15#' and the output is a green cursor.

Besides make menuconfig, there are Kernel setting commands such as make config and make xconfig but the most popular one is the make menuconfig which is simple UI(User Interface) to use with the arrow keys known as the console(monitor) or telnet terminal is used for the Kernel Configuration.

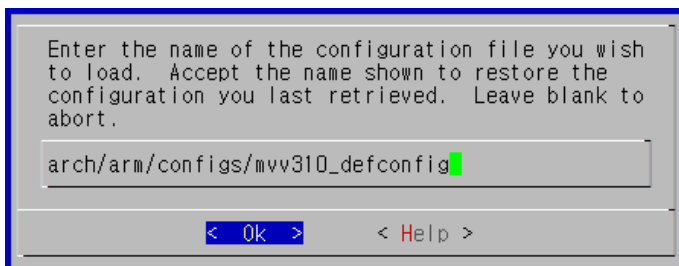
If all the content of the setting menu is set, it doesn't have to be newly set in each time. So to save the previous configuration to a separate file, there is an option in the menu down below as "Save Configuration to an Alternate File". In opposite, previous setup configuration can be reloaded, Load and Kernel Configuration can be made by reading the file from "[mv210_defconfig](#)" which is saved at arch/arm/configs/ which is Kernel Source directory.

Next, select the "[Load an Alternate Configuration File](#)" menu on the bottom section of the make menuconfig screen, and enter in the following:

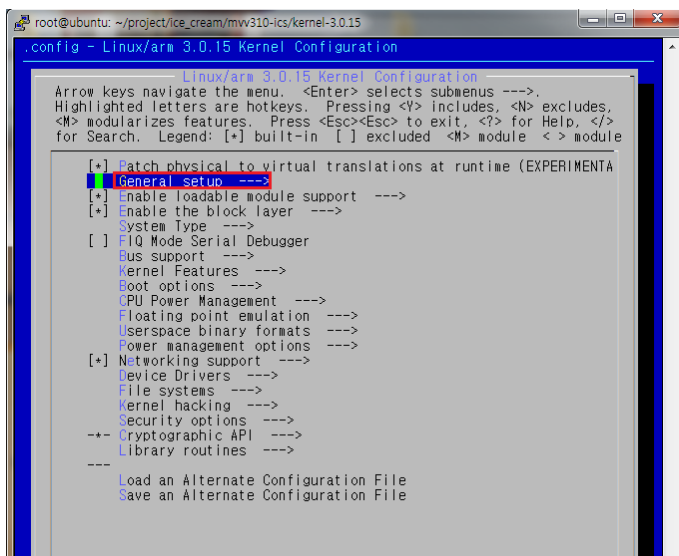
“Load an Alternate Configuration File”



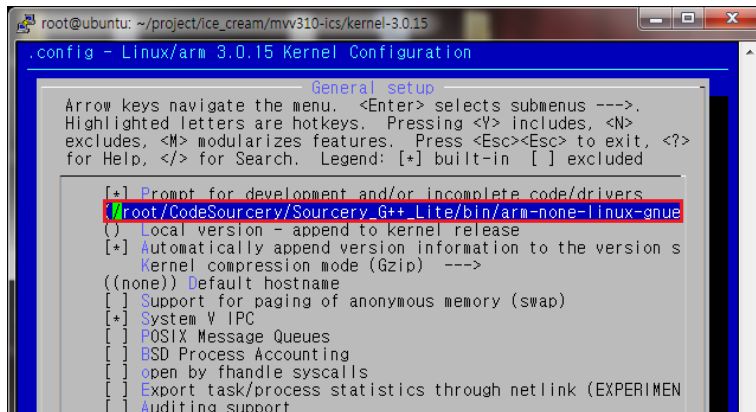
Load “arch/arm/configs/mv210_defconfig”



General setup

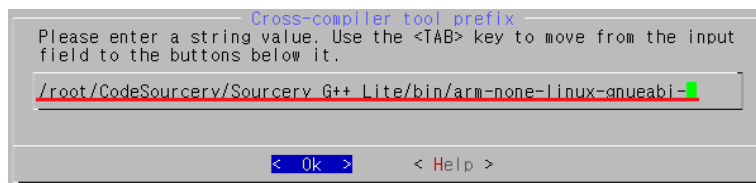


“arm-none-linux-gnueabi-“



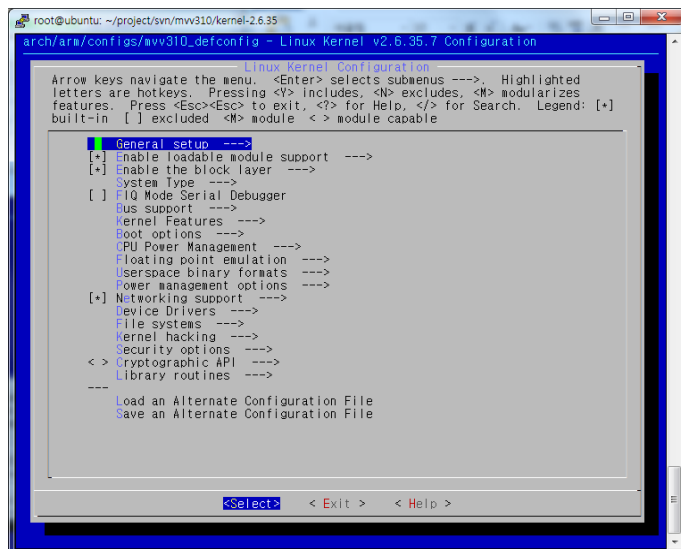
Add the installed Q3 path.

(/root/CodeSourcery/Sorcery G++ Lite/bin/arm-none-linux-gnueabi-)



The kernel configuration(make menuconfig) must be saved after the setup is complete. The kernel configuration is saved under the file name “config” under the kernel source directory. The reason “config” needs to be saved is that it will be checked during the “make dep” step, which is a crucial step for the compilation process. If a window asking to save pops up, make sure to answer “yes”.

After loading is complete, exit.



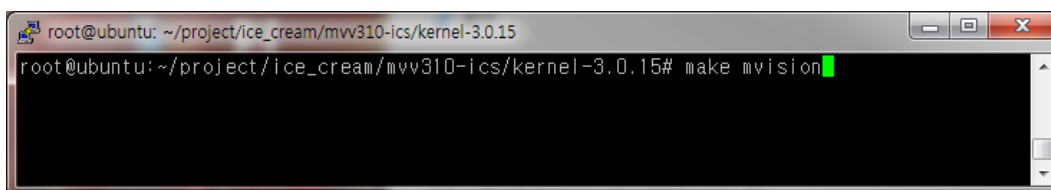
The Linux kernel image (zImage) making process is divided into compiling, linking, file type changing (ELF→BIN) by Binutil(objcopy), and file decompression (gzip). All of these combined make up the command “make” under Makefile.

Compile using the “make” command:

```
# make mvv310_defconfig
```

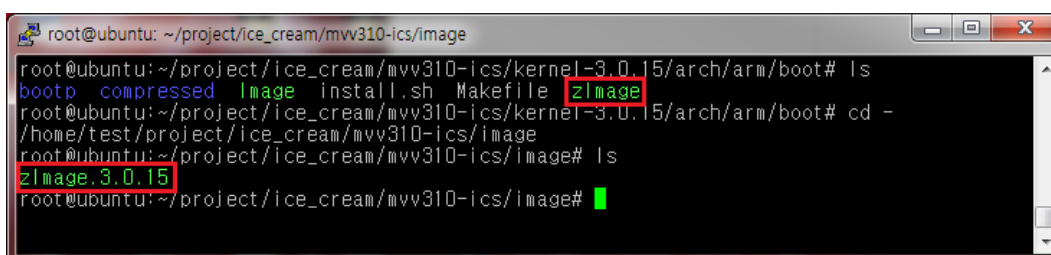
```
# make modules
```

```
# make mvision
```



```
root@ubuntu: ~/project/ice_cream/mvv310-ics/kernel-3.0.15
root@ubuntu: ~/project/ice_cream/mvv310-ics/kernel-3.0.15# make mvision
```

zImage file is created inside linux/arch/arm/boot when the compilation is complete:

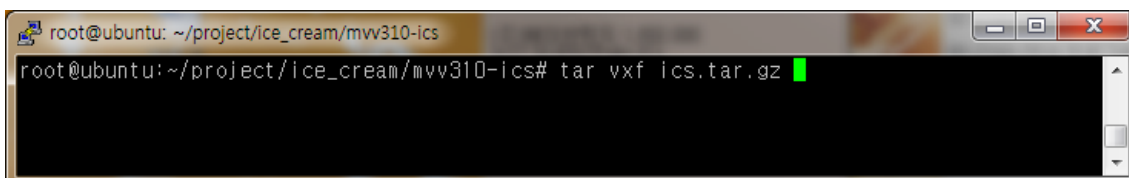


```
root@ubuntu: ~/project/ice_cream/mvv310-ics/image
root@ubuntu: ~/project/ice_cream/mvv310-ics/kernel-3.0.15/arch/arm/boot# ls
bootp compressed Image install.sh Makefile zImage
root@ubuntu: ~/project/ice_cream/mvv310-ics/kernel-3.0.15/arch/arm/boot# cd -
~/home/test/project/ice_cream/mvv310-ics/image
root@ubuntu: ~/project/ice_cream/mvv310-ics/image# ls
zImage.3.0.15
root@ubuntu: ~/project/ice_cream/mvv310-ics/image#
```

5. ICE Cream Sandwich Compilation

Enter in the following command to uncompress the file.

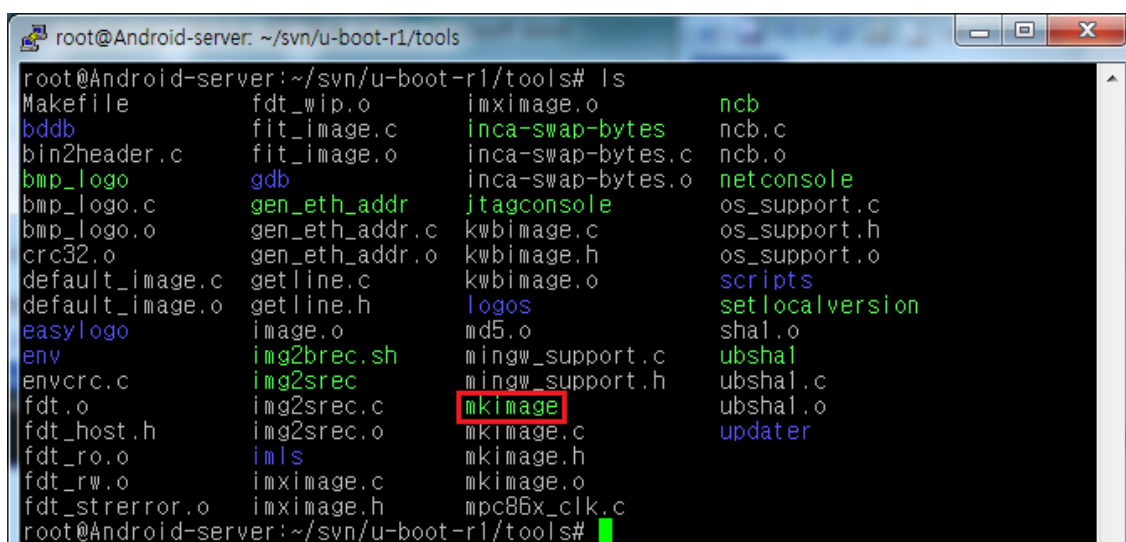
```
# tar xvf ics.tar.gz
```



```
root@ubuntu: ~/project/ice_cream/mvv310-ics
root@ubuntu:~/project/ice_cream/mvv310-ics# tar xvf ics.tar.gz
```

(Caution)

Before compilation, “mkimage” in the image-compiled /u-boot-mvv310-dev/tools must be exported in order to generate the ramdisk properly. Enter in the following command for export:

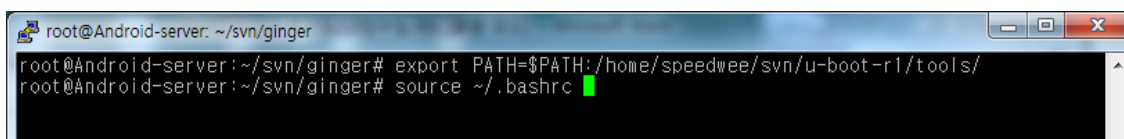


```
root@Android-server: ~/svn/u-boot-r1/tools
root@Android-server:~/svn/u-boot-r1/tools# ls
Makefile          fdt_wip.o          imximage.o        ncb
bddb              fit_image.c        inca-swap-bytes   ncb.c
bin2header.c     fit_image.o        inca-swap-bytes.c ncb.o
bmp_logo         gdb                inca-swap-bytes.o netconsole
bmp_logo.c       gen_eth_addr       jtagconsole       os_support.c
bmp_logo.o       gen_eth_addr.c     kwbimage.c        os_support.h
crc32.o          gen_eth_addr.o     kwbimage.h        os_support.o
default_image.c  get_line.c         kwbimage.o        scripts
default_image.o  get_line.h         logos              set_localversion
easylogo         image.o            md5.o              shal.o
env              img2brec.sh        mingw_support.c   ubshal
envcrc.c         img2srec           mingw_support.h   ubshal.c
fdt.o            img2srec.c         mkimage           ubshal.o
fdt_host.h       img2srec.o         mkimage.c         updater
fdt_ro.o         i_mls              mkimage.h
fdt_rw.o         imximage.c         mkimage.o
fdt_strerror.o  imximage.h         mpc85x_clk.c
```

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

```
# export PATH=$PATH:/home/speedwee/svn/u-boot-mvv310/tools/
```

```
# source ~/.bashrc
```

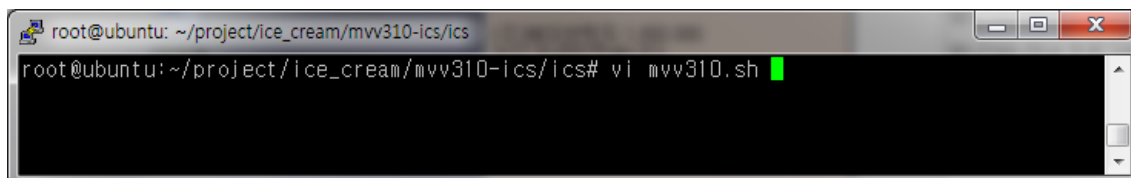


```
root@Android-server: ~/svn/ginger
root@Android-server:~/svn/ginger# export PATH=$PATH:/home/speedwee/svn/u-boot-r1/tools/
root@Android-server:~/svn/ginger# source ~/.bashrc
```

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

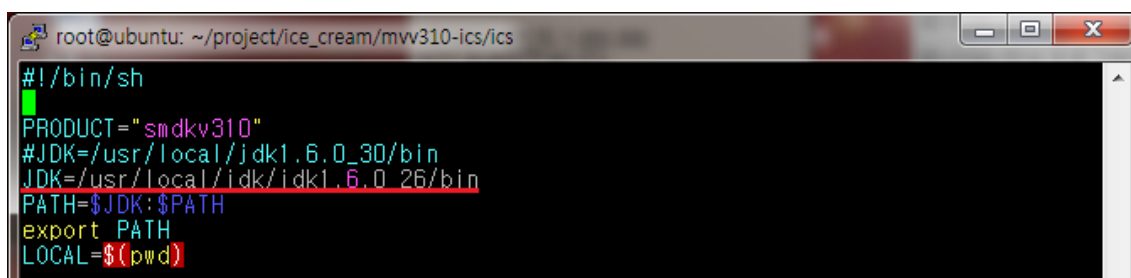
<http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u26-download-400750.html>

#vi mvv310.sh



```
root@ubuntu: ~/project/ice_cream/mvv310-ics/ics
root@ubuntu:~/project/ice_cream/mvv310-ics/ics# vi mvv310.sh
```

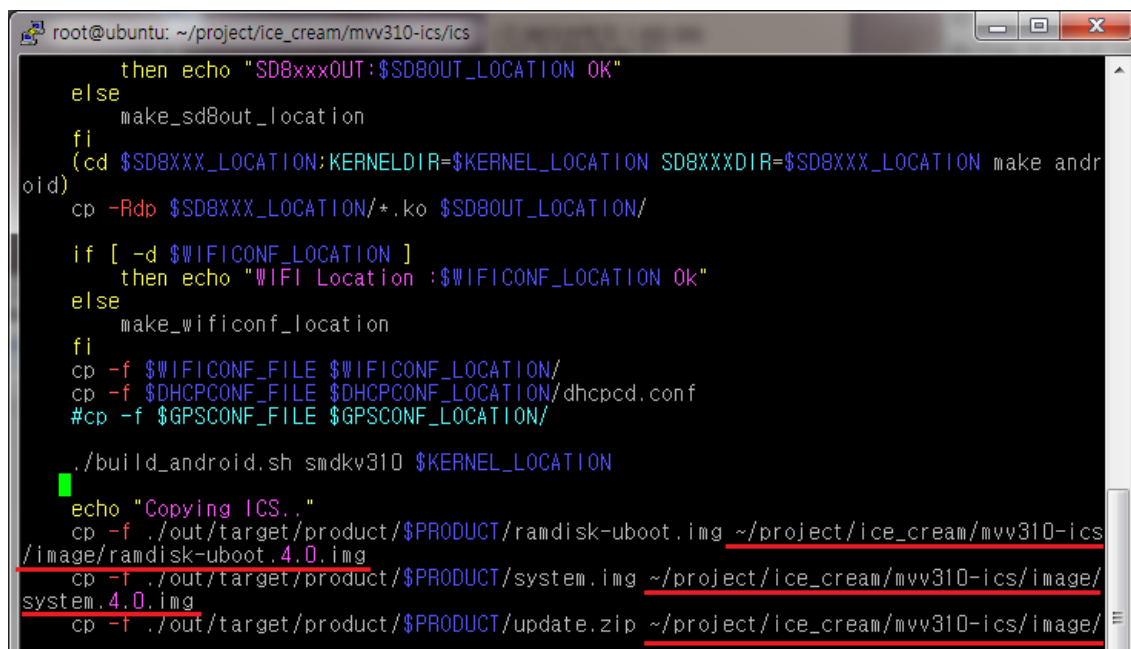
Move to the path to the folder which is installed with JDK.



```
#!/bin/sh
PRODUCT="smdkv310"
#JDK=/usr/local/jdk1.6.0_30/bin
JDK=/usr/local/jdk/jdk1.6.0_26/bin
PATH=$JDK:$PATH
export PATH
LOCAL=$(pwd)
```

Kernel Path Setup

(Move to the path to the kernel folder)



```
then echo "SDBxxxOUT:$SDBOUT_LOCATION OK"
else
make_sdBout_location
fi
(cd $SDBXXX_LOCATION:KERNELDIR=$KERNEL_LOCATION SDBXXXDIR=$SDBXXX_LOCATION make android)
cp -Rdp $SDBXXX_LOCATION/+.ko $SDBOUT_LOCATION/

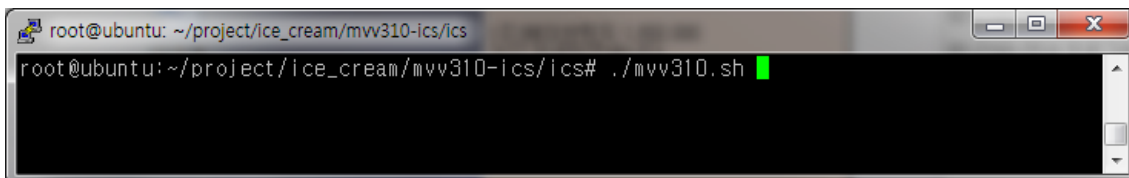
if [ -d $WIFICONF_LOCATION ]
then echo "WIFI Location :$WIFICONF_LOCATION Ok"
else
make_wificonf_location
fi
cp -f $WIFICONF_FILE $WIFICONF_LOCATION/
cp -f $DHCPCONF_FILE $DHCPCONF_LOCATION/dhcpd.conf
#cp -f $GPSCONF_FILE $GPSCONF_LOCATION/

./build_android.sh smdkv310 $KERNEL_LOCATION

echo "Copying ICS.."
cp -f ./out/target/product/$PRODUCT/ramdisk-uboot.img ~/project/ice_cream/mvv310-ics/image/ramdisk-uboot.4.0.img
cp -f ./out/target/product/$PRODUCT/system.img ~/project/ice_cream/mvv310-ics/image/system.4.0.img
cp -f ./out/target/product/$PRODUCT/update.zip ~/project/ice_cream/mvv310-ics/image/
```

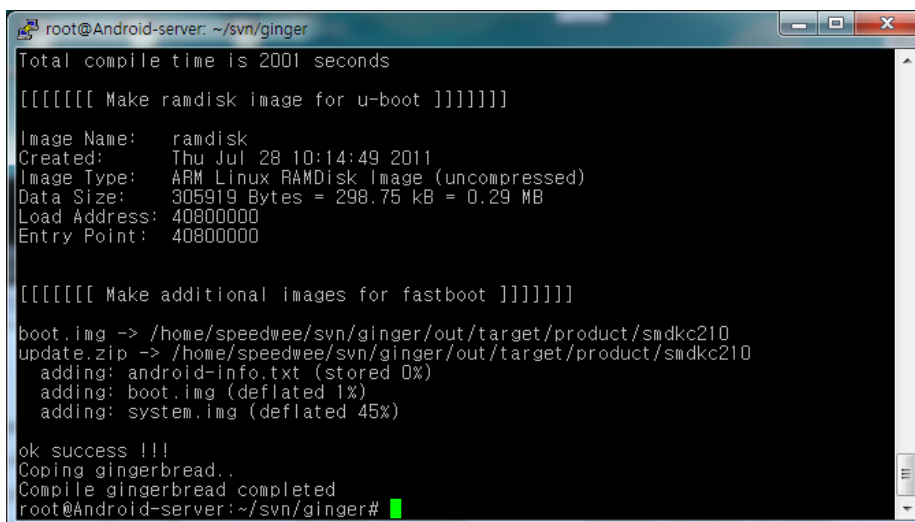
Do the compilation with the below command.

```
# ./mvv310.sh
```



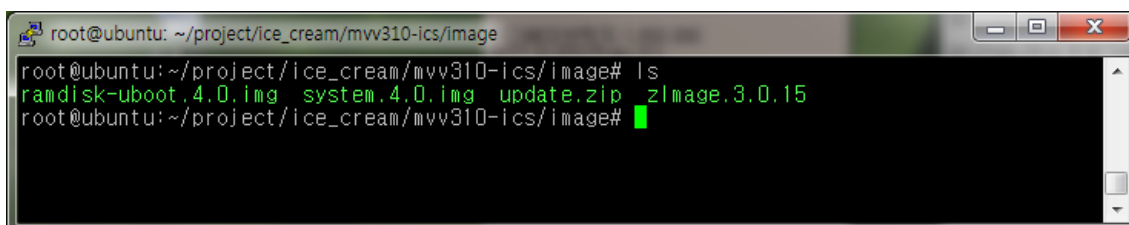
```
root@ubuntu: ~/project/ice_cream/mvv310-ics/ics
root@ubuntu:~/project/ice_cream/mvv310-ics/ics# ./mvv310.sh
```

Screen with the successful compilation



```
root@Android-server: ~/svn/ginger
Total compile time is 2001 seconds
[[[[[[[[ Make ramdisk image for u-boot ]]]]]]]
Image Name: ramdisk
Created: Thu Jul 28 10:14:49 2011
Image Type: ARM Linux RAMDisk Image (uncompressed)
Data Size: 305919 Bytes = 298.75 kB = 0.29 MB
Load Address: 40800000
Entry Point: 40800000
[[[[[[[[ Make additional images for fastboot ]]]]]]]
boot.img -> /home/speedwee/svn/ginger/out/target/product/smdkc210
update.zip -> /home/speedwee/svn/ginger/out/target/product/smdkc210
adding: android-info.txt (stored 0%)
adding: boot.img (deflated 1%)
adding: system.img (deflated 45%)
ok success !!!
Coping gingerbread..
Compile gingerbread completed
root@Android-server:~/svn/ginger#
```

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



```
root@ubuntu: ~/project/ice_cream/mvv310-ics/image
root@ubuntu:~/project/ice_cream/mvv310-ics/image# ls
ramdisk-uboot.4.0.img system.4.0.img update.zip zImage.3.0.15
root@ubuntu:~/project/ice_cream/mvv310-ics/image#
```