MV V210 Linux Compilation

Microvision Co., Ltd.

Document Information

Version	1.3
File Name	MVV210 Linux Compilation.doc
Date	2010. 9. 13
Satus	Working

Revision History

Date	Version	Update Descriptions	Editor
2010. 6. 14.	V1.0	First Edition	Jongill Wee

Published by MicroVision Co.,Ltd.

(☎) +82-2-3283-0101, (✉) sale@microvision.co.kr

http://www.microvision.co.kr, http://www.mvtool.co.kr

Room #610, Hanshin IT Tower 235, Guro3-dong, Guro-gu, Seoul, Korea.

Contents

1. Package for Development

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

File	Description	Version
mv-v210_linux_uboot.tar.gz	Bootloader	1.3.4
kernel_2.6.32_v210.tar.gz	Kernel	2.6.32
mv-v210_rootfs.tar.gz	File System	ramdisk
4.3.1-eabi-armv6.tar.gz	GCC Compiler	4.3.1
arm-2009q3-67-arm-none-linux-gnueabi.bin	q3-compiler	Q3 67

Tool chain

This Linux 2.6.32 BSP compiles Bootloader and the Kernel uses Q3 Compiler for compilation. (How to install Q3 is described on page 12.)

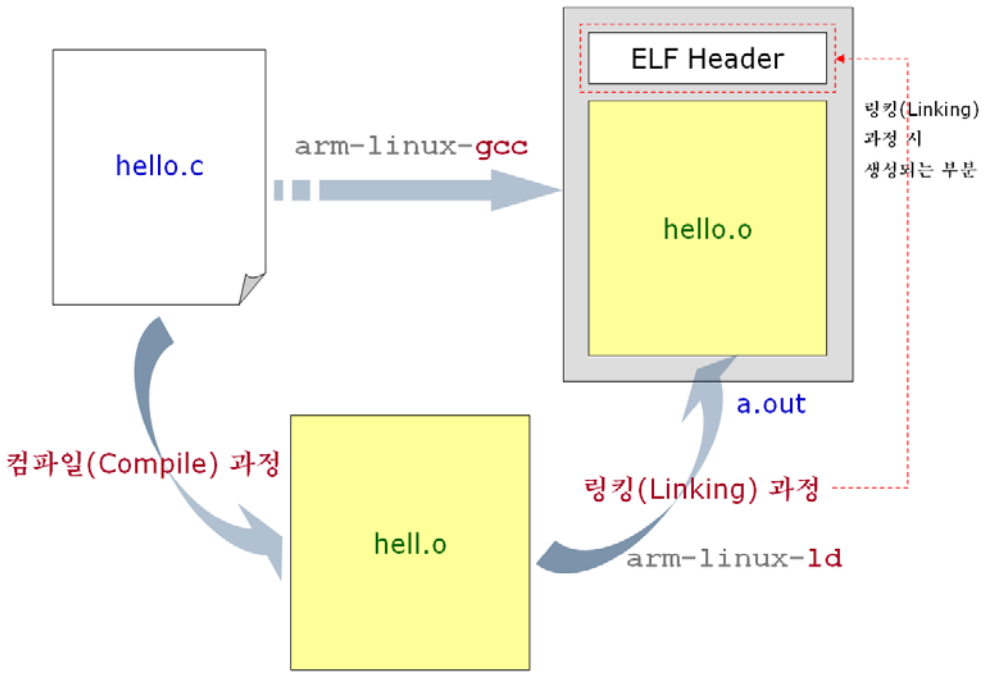
The reason for including GCC 4.3.1 is to use it for busy box or other purposes.

MV-V210 IO MAP		
NAME	Virtual offset	Physical
VIC[0:3]	0x00000000	0xF2000000
SYSCON	0x00100000	0xE0100000
GPIO	0x00200000	0xE0200000
TIMER	0x00300000	0xE2500000
WATCHDOG TIMER	0x00400000	0xE2700000
PSHOLD	0x00500000	0xE010E81C
LCD-FIMD	0x00600000	0xF8000000
CHIPID	0x00700000	0xE0000000
SROMC	0x00800000	0xE8000000
DMC0	0x00A00000	0xF0000000
DMC1	0x00B00000	0xF1400000
USB OTG	0x00E00000	0xEC000000
OTGSFR	0x00F00000	0xEC100000
UART	0x01000000	0xE2900000
SYSTEM TIMER	0x01200000	0xE2600000
NAND	0x01400000	0xB0E00000
AUDSS	0x01600000	0xEEE10000

2. GCC (Tool chain) Setup

In order to develop MV-V210 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 Process:

-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-V210-LCDLinux BSP as well.

2.1 Decompression

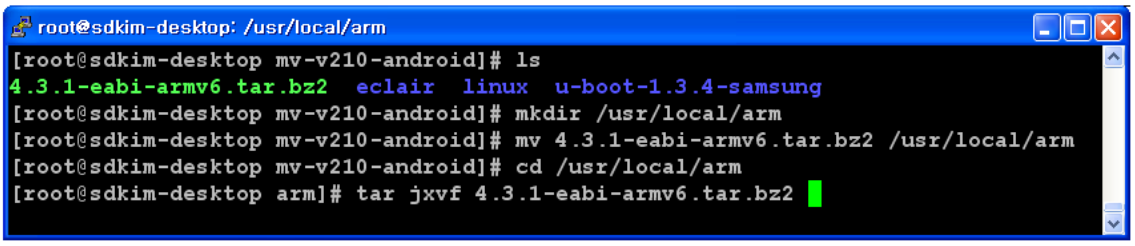
After copying the files to Linux server through Samba or FTP, use the following steps to start the decompression process:

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

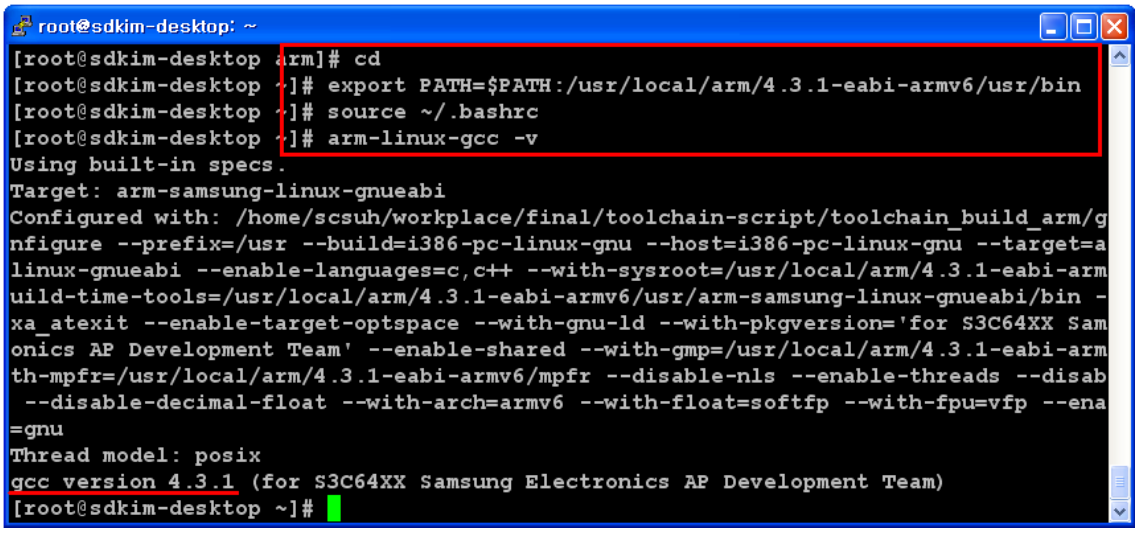


2.2 GCC Environment PATH Setup

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



< Steps to modify the PATH variables for Tool chain >

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

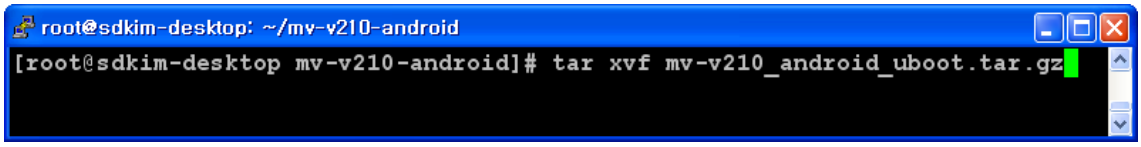
If there are no problems, use the “which” command to check where the installed command is located in the PATH. If the PATH is displayed correctly as shown in the diagram on page 5, the command has been executed successfully. For reference, you can use the “--version” option to check that the currently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

3. Bootloader Setup

3.1. u-boot Environment Setup

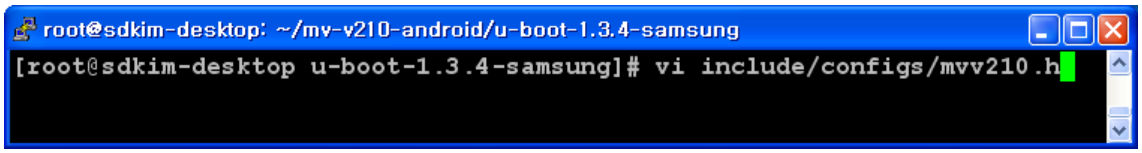
Generally, the Embedded Linux BSP is composed of 3 image files:
Embedded Linux BSP = Boot Loader + Kernel + File System
Boot Loader is the program necessary to load the kernel to the memory.

Enter in the following for file decompress:
tar xvf mv-v210_android_uboot.tar.gz



As shown below using the “vi” editor, open the file “mvv210.h” and you will find the basic environment at its default. (ex: TFTP, CPU clock, DDR Program Counter)

vi include/configs/mvv210.h



mvv210.h Content

TFTP IP Setup:

```
#define CONFIG_NETMASK    255.255.255.0        <- Subnet Mask
#define CONFIG_IPADDR          192.168.0.20          <- mv-v210 IP
#define CONFIG_SERVERIP        192.168.0.10          <- Linux TFTP Server IP
#define CONFIG_GATEWAYIP       192.168.0.1           <- Gate Way IP
```

Change the prompt name on the mv-v210 boot board after booting the new bootloader program:

```
#define CFG_PROMPT             "MV-V210 # "    /* Monitor Command Prompt        */
```

CPU Clock Configuration(Remove the commend):
#define CONFIG_CLK_800_160_166_133

DDR Program Counter address required for downloading:
#define CFG_UBOOT_BASE 0xc3e00000
#else
#define CFG_UBOOT_BASE 0x23e00000

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.

Installing procedures:

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

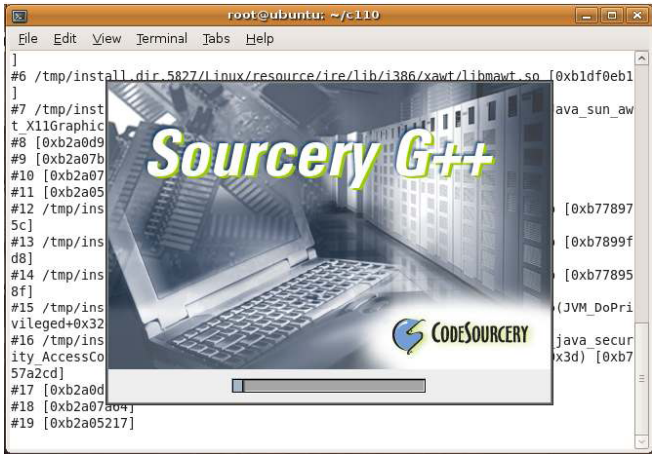
When this message displays “% sudo depkg-reconfigure -plow dash” follow the steps below:

sudo dpkg-reconfigure -plow dash

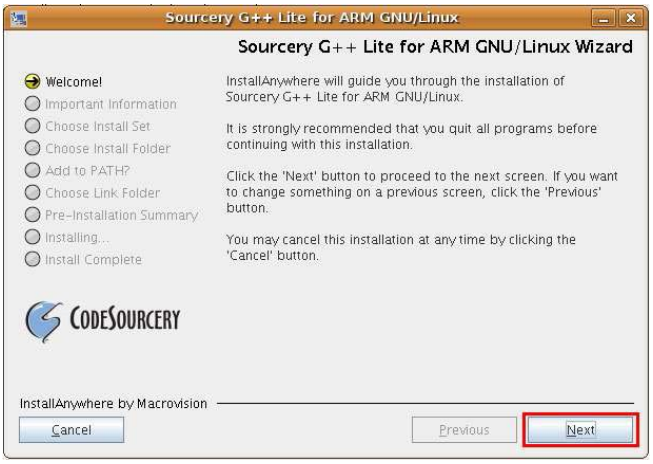
When a [yes/no] screen pops up, click “No” and enter in the command as shown below:

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

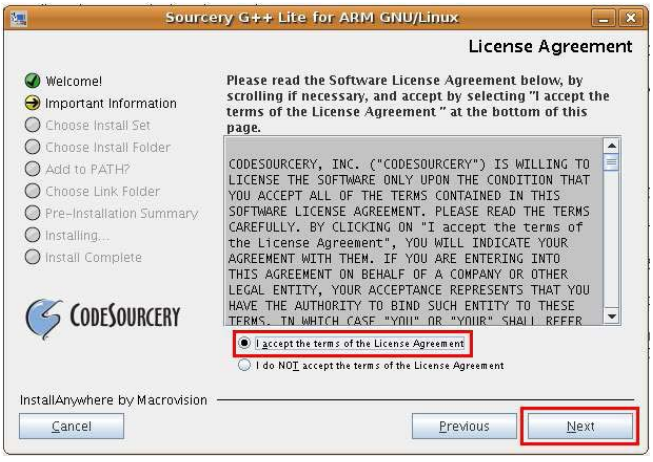
Below is a picture of the loading process:



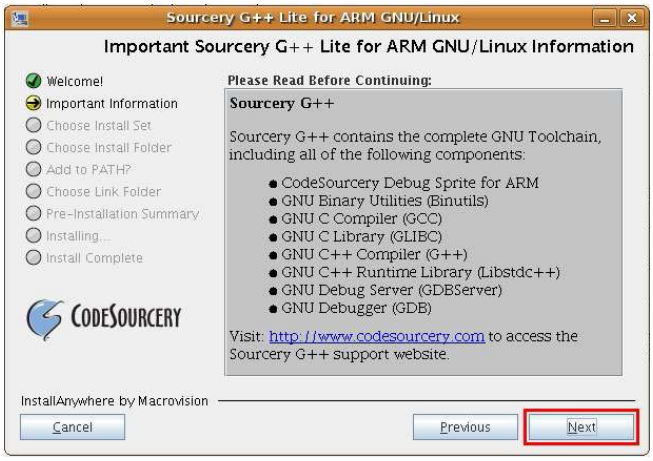
Click “Next”



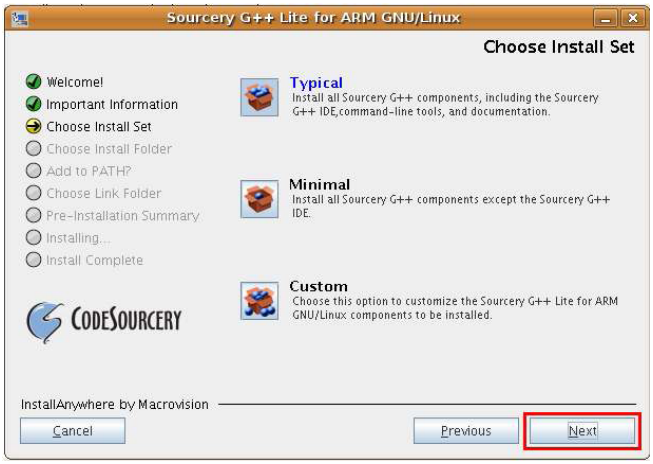
Agree to the terms of the License Agreement then click “Next”



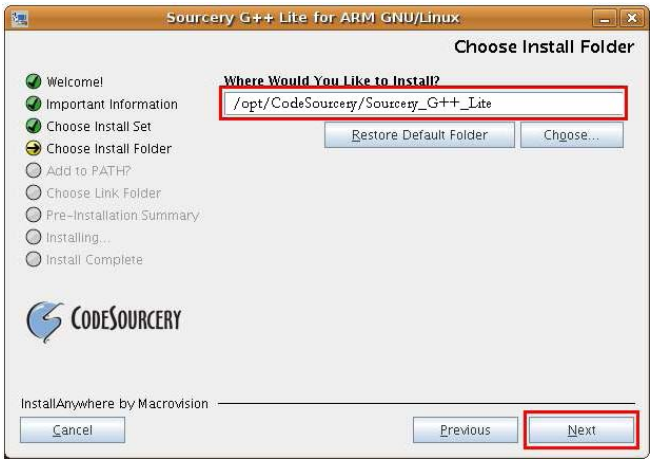
Click “Next”



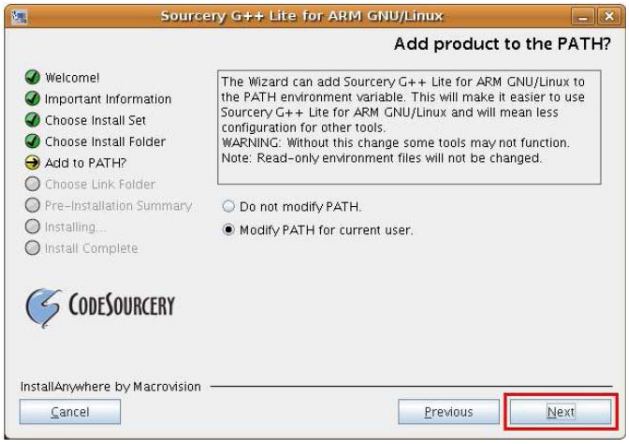
Click “Next”



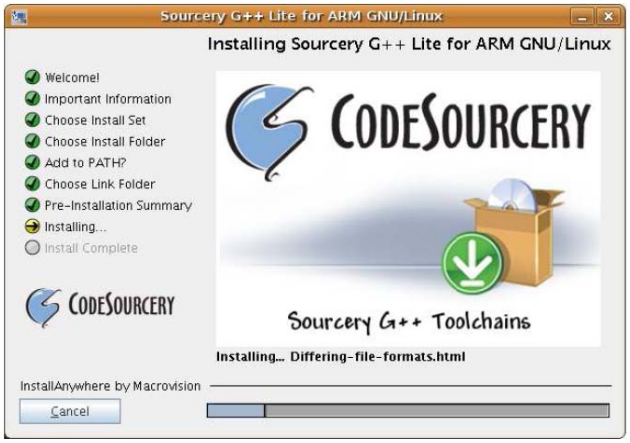
Click “Next”



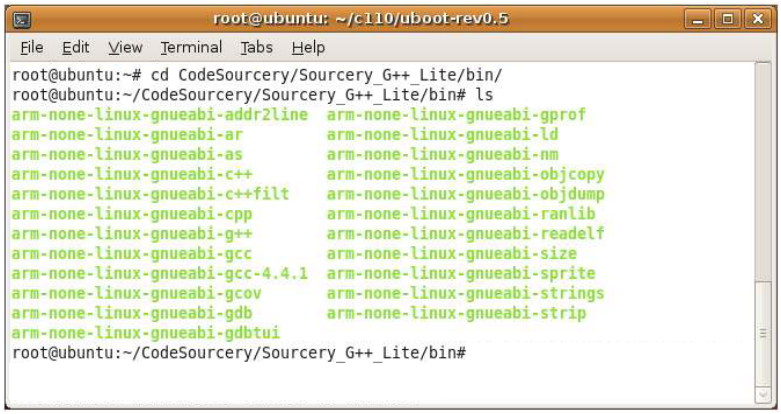
Click “Next”



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



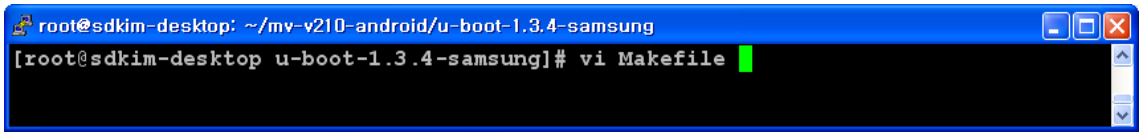
When the installation is complete, you can check the Q3 library which has been installed in “/opt/CodeSourcery/Sourcery_G++_Lite/bin”.



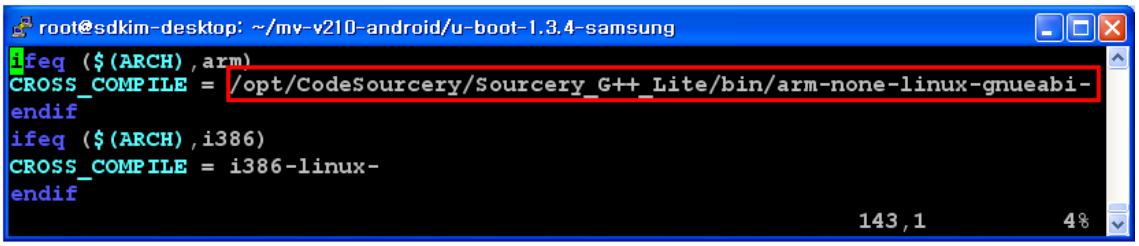
Now we will work on Makefile for the Boot Loader compilation.

Use the “vi” editor to open Makefile.

vi Makefile



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

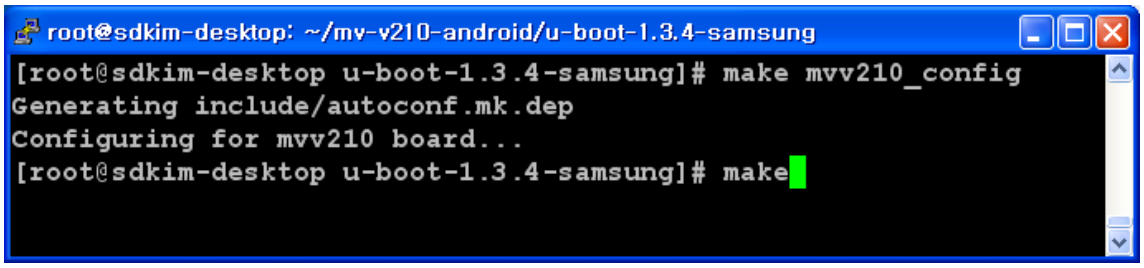


Compilation Steps: (after compilation, type in command #make distclean)

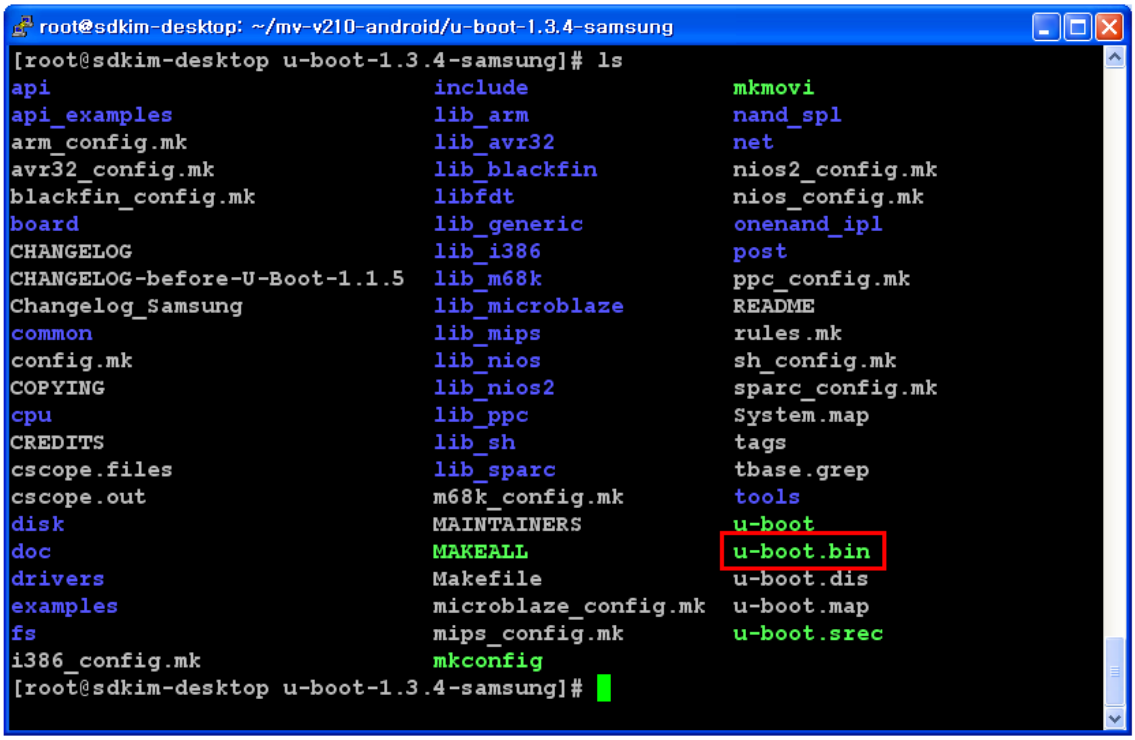
make clobber

make mvv210_config

make



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



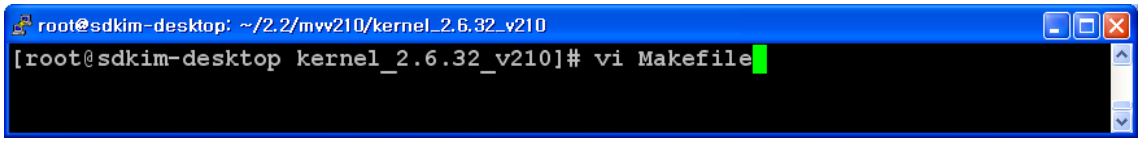
4. Kernel Setup

4.1. How to Compile

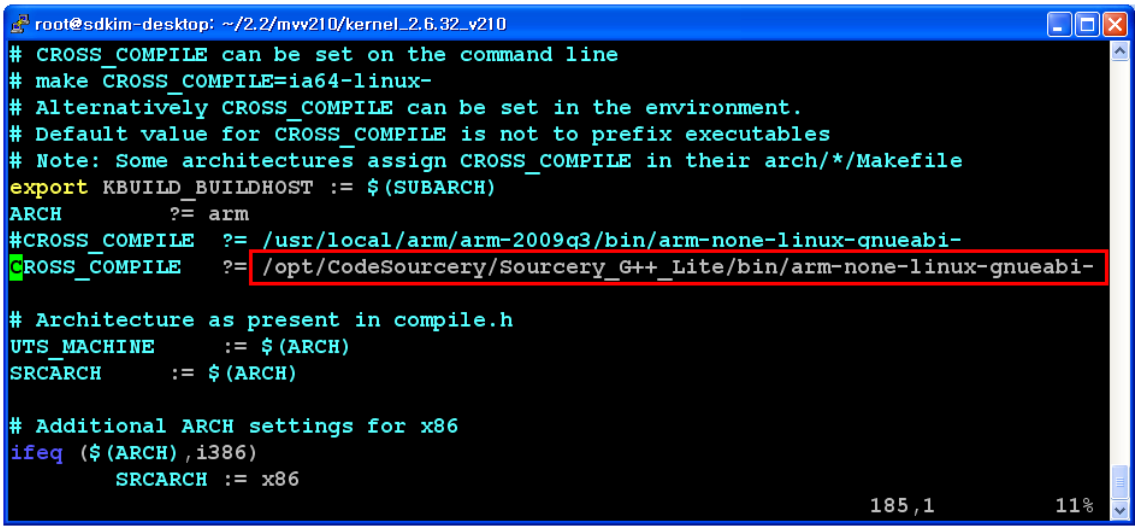
Enter in the following commands for decompression:

tar xvf kernel_2.6.32_v210.tar.gz

vi Makefile



Add the installed Q3 path.



/opt/CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-

Save after checking the path.

Kernel Source Tree Structure

MV-V210 Kernel Source Tree	
Location	**Description of content**
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

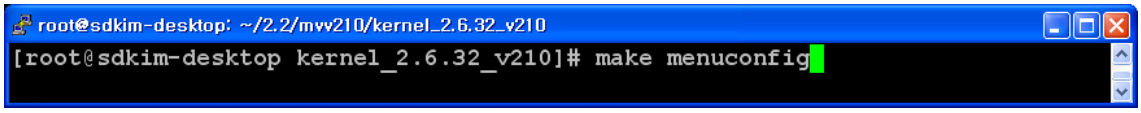
The compilation method after the Kernel configuration is finished includes using the make utility, such as when compiling other source codes. However, the Kernel cannot directly use the Executable ELF, which is the output file from compiling and linking. As a result, it needs to be made into a binary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

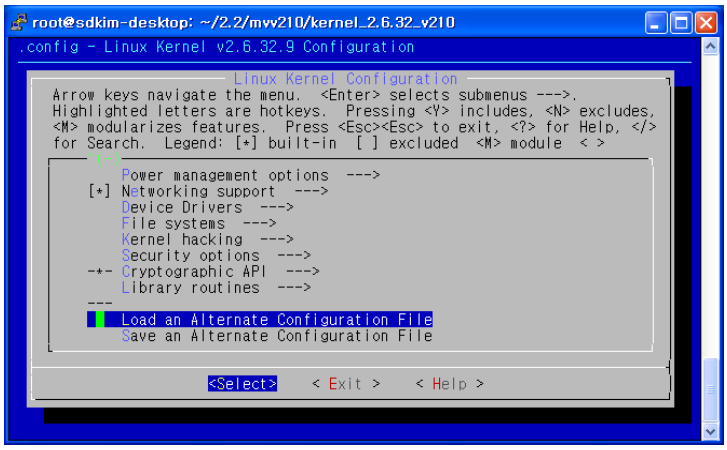
Put in the following commands for compilation to execute the kernel environment setup:

make menuconfig

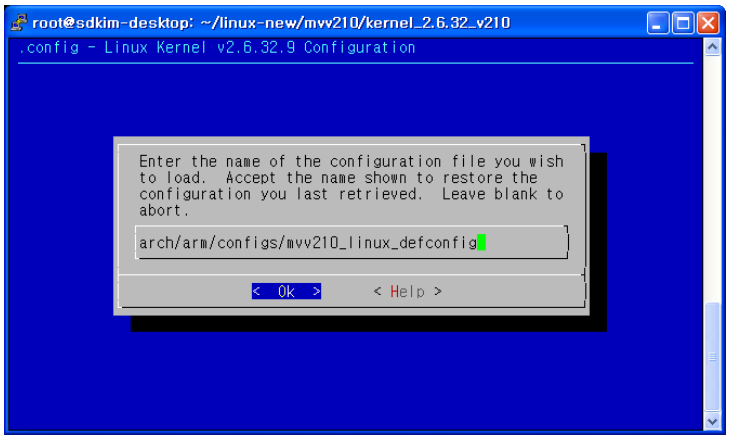


There is a "Save Configuration to an Alternate File" menu. On the other hand, you can also load the configuration file. Load "mvv210_linux_defconfig" which is under the kernel source directory arch/arm/configs/ .

Next, select the “Load an Alternate Configuration File” menu on the bottom section of the make menuconfig screen, and enter in the following:

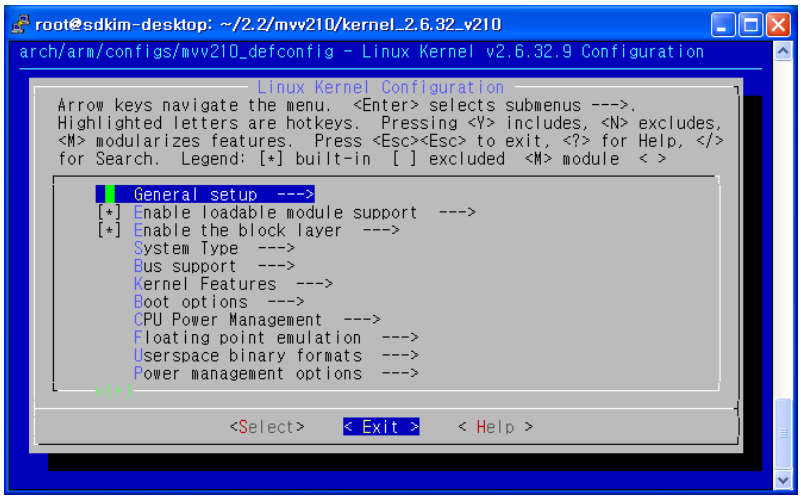


Load “arch/arm/configs/mvv210_defconfig”



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”.

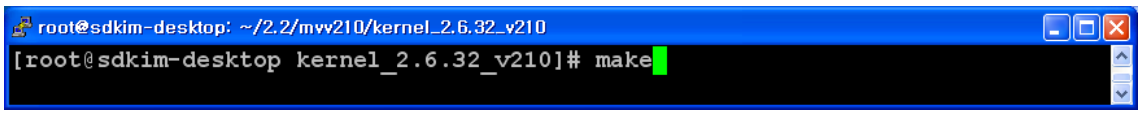
After loading is complete, exit.



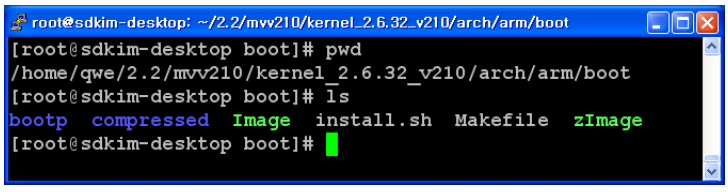
The Linux kernel image (zImage) making process is divided into compiling, linking, file type changing (ELF→BIN) by Binutil(objcopy), and file decompression (gzip). All of these combined make up the command “make” under Makefile.

Compile using the “make” command:

make



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



5. File System

In order to run the user application in Linux, a file system is required. There are three different embedded file systems:

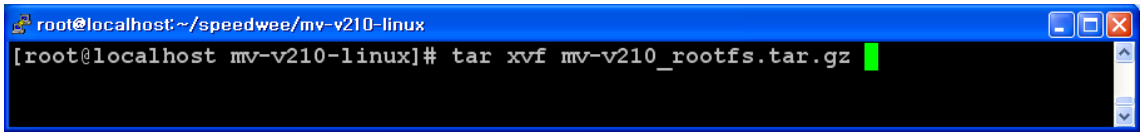
Embedded File System Comparisons:

Type	Memory	Pros	Cons
RAMDISK	RAM (SDRAM)	File system is stored in memory(RAM). The process speed is fast because the application can be read directly from the memory.	Because a part of memory(RAM) is used for DISK(file system), fewer applications can be executed.
JFFS2, YAFFS	FLASH Memory	Due to the buildup of file system in the flash memory, more applications can be executed.	The speed is slower than RAMDISK because the application program must be loaded from the flash memory to memory(RAM)..
Cramfs	ROM (Read Only)	The file system image file size smaller due to higher compression rate.	The application environment is inconvenient because it is Read Only.

MV-V210 uses RAMDISK.

Enter in the following commands for decompression:
(the decompression must be made in the “root” account)

tar xvf mv-v210_rootfs.tar.gz



(CAUTION)

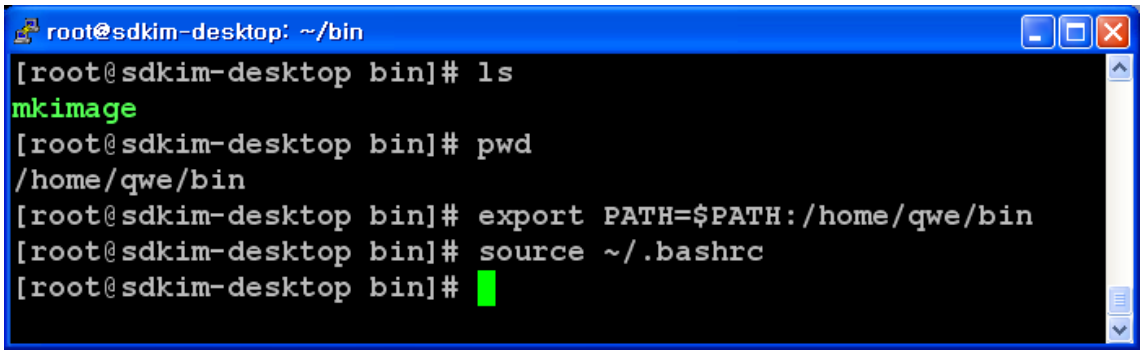
Before compilation, "mkimage" in the image-compiled /u-boot-1.3.4-samsung/tools must be exported in order to generate the ramdisk properly. Enter in the following command for export:

Example: # export PATH=$PATH:/home/u-boot-1.3.4-samsung/tools/tools

For convenience, we created a bin folder inside the qwe folder:

export PATH=$PATH:/home/qwe/bin

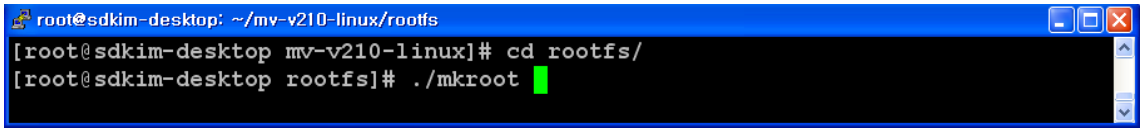
source ~/.bashrc



Generate the ramdisk using the following command:

cd rootfs

./mkroot



Successfully built image

As shown below, the generation of image linux-ramdisk.gz may be checked :