Ingenic Linux Development Guide

Revision: 0.2
Date: Aug 2009

北京君正集成电路有限公司
Ingenic Semiconductor Co. Ltd

Ingenic Linux Development Guide

Release history

Date	Revision	Change
2009.08.17	0.2	Added the contents of jz4750d (jz4755), introduce the method to burn the whole system using SDcard, deleted the content of jdi.
2009.02.02	0.1	Created

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenic for the current documentation and errata.

Ingenic Semiconductor Inc.

Room 108, Information Center Block A
Zhongguancun Software Park
8 Dongbeiwang west Road, Haidian District
Beijing China, 100193

Tel: 86-10-82826661
Fax: 86-10-86865845
Http: //www.ingenic.cn

CONTENT

1 Overview

This manual is intended for engineers who want to develop Linux drivers and applications on the XBurst Platform. By reading this manual, you can learn how to setup the development environment and how to develop your drivers and applications.

First of all, you want to prepare following resources:

1） A Linux development host to install the toolchain and source code. Ubuntu or Fedora is recommended.
2） A development board based on the JZ series CPU.
3） Debugging tools including a serial cable, a power line.
4） MIPS cross toolchain: we provide a toolchain with gcc-4.1.2 and glibc-2.6.1.
5） The Bootloader source: we provide u-boot-1.1.6.
6） The Linux kernel source: we provide Linux 2.6.24.3.
7） A rootfs

2 Prepare the Development Environment

The first thing you want to do is to prepare a Linux host PC for your development. We will install the cross toolchain and the source codes on the PC host, and will configure it to start the TFTP and NFS services. Here we tested it on an Ubuntu 7.10 host.

2.1 Install the cross toolchain

You can download the toolchain from the homepage of Ingenic (http://www.ingenic.cn). The toolchain was named mipseltools-gcc412-glibc261.tar.bz2. Then you can follow these steps to install it:

```
$ mkdir -p /opt
$ cd /opt
$ tar xjf mipseltools-gcc412-glibc261.tar.bz2
$ export PATH=/opt/mipseltools-gcc412-glibc261/bin:$PATH
```

Then you can test the toolchain:

```
$ mipsel-linux-gcc -o helloworld helloworld.c
```

The cross compiler should be ok now.

2.2 Start the TFTP and NFS services

Then we want to start the TFTP and NFS services.

Modify /etc/xinetd.d/tftp:

```
service tftp
{
    socket_type     = dgram
    protocol        = udp
    wait            = yes
    user            = root
```

```
    server          = /usr/sbin/in.tftpd
    server_args     = -s /tftpboot/
    disable         = no
    per_source      = 11
    cps             = 100 2
    flags           = IPv4
}
```

After save it, start the service:

```
$ sudo /etc/init.d/xinetd restart
```

Modify /etc/exports:

```
/nfsroot     *(rw,sync,no_root_squash)
```

Then start the service:

```
$ sudo exportfs -r
$ sudo /etc/init.d/nfs-kernel-server restart
```

After the cross toolchain, TFTP and NFS services have been setup and started successfully, you can start your work now. Please read following sections for details.

3 U-Boot Development

U-Boot is the bootloader to load and boot the Linux kernel. We have ported u-boot version 1.1.6 on the XBurst platform. It supports to load the Linux kernel from Ethernet via the TFTP or NFS service. It also supports various device drivers, including NAND flash, LCD, MMC/SD card etc.

3.1 Get the U-Boot Source

You can get the u-boot source from Ingenic homepage (http://www.ingenic.cn/). The source contains two packages, one is the original source tarball called u-boot-1.1.6.tar.bz2, the other is the patch from Ingenic called u-boot-1.1.6-jz-yyyymmdd.patch.gz ("yyyymmdd" represents the patch release date).

Copy the source packages to the Linux host and extract them to your work directory:

```
# tar -xjf u-boot-1.1.6.tar.bz2
# cd u-boot-1.1.6
# gzip -cd ../u-boot-1.1.6-jz-yyyymmdd.patch.gz | patch -p1
```

3.2 Configure and Compile U-Boot

The U-Boot source supports several boards, they use different configuration file for compiling. Here is a simple list:

JZ4730 PMPV2 board (Boot from NOR flash):

```
$ make pmpv2_config
$ make
```

JZ4730 PMPV2 board (Boot from NAND flash):

```
$ make pmpv2_nand_config
$ make
```

JZ4740 PAVO board (Boot from NAND flash):

```
$ make pavo_nand_config
$ make
```

JZ4750 APUS board (Boot from NAND flash):

```
$ make apus_nand_config
$ make
```

5) JZ4750 APUS board (Boot from SD card):

```
$ make apus_msc_config
$ make
```

3.3 Burn U-Boot Binary to the Target Board

After making, you can get a binary called u-boot-nand.bin (or u-boot.bin or u-boot-msc.bin, depending on the board configuration). Then you may use USB-Boot tool to burn the binary to the target board. Next is a simple command guide for you. Detailed usage can be referred to the USB-Boot manual.

1) Burning u-boot-nand.bin via USB Boot tool

```
USBBoot :> boot 0
USBBoot :> nerase 0 8 0 0
USBBoot :> nprog 0 u-boot-nand.bin 0 0 -n
```

After that, you can boot your target board now. If things go well, you should see the print message on the serial console.

3.4 U-Boot command

Below is a brief description of some commonly used U-Boot command:

"Help" command: The command to view all the orders, which "help command" see the specific command format.
"printenv" command: This command is used to view environment variables.
"setenv" command: This command sets the environment variable.
"askenv" command: This command sets the environment variable.
"saveenv" command: The command is used to save the environment variable.
"bootp" command: This command dynamically get IP.
"tftpboot" command: This command through the TFTP protocol to download files from the network to run.
"nfs" command: This command through the NFS protocol to download files from the network to run.

"bootm" command: This command is from the memory to run u-boot image.
"go" command: This command is from the memory to run the application.
"boot" command: This command runs the specified command bootcmd environment variable.
"reset" command: This command reset CPU.
"md" command: Display memory data.
"mw" command: Modify memory data.
"cp" command: Memory copy command.

NOR Flash command:

"protect" command: NOR Flash write protection enable / disable command.
"erase" command: NOR Flash Erase command.

NAND Flash command:

"nand info": see NAND information.
"nand erase": NAND Flash Erase command.
"nand read": NAND Flash read command.
"nand write": NAND Flash Write Command.
"nand bad": NAND Flash bad block information.

MMC / SD Card command:

"mmcinit": MMC / SD initialization command.
"fatls mmc 0 /": See MMC / SD directory and file commands.
"fatload mmc 0 address file": read MMC / SD card file command.

Below is a brief description of some commonly used U-Boot environment variables:

"ipaddr": development board IP address.
"serverip": the server IP address.
"bootfile": u-boot startup files.
"bootcmd": u-boot startup command.
"bootdelay": u-boot startup delay.
"bootargs": u-boot boot parameters (ie linux command-line parameters)
"ethaddr": the network MAC address.

3.5 An example of using U-Boot

Power to the target board and start, you can see the output of information from the serial port terminal, and then press any key to bypass the automatic startup process into the U-Boot command interface. In the U-Boot command interface to run "help" command, you can see U-Boot support all commands, run "help command" see the specific format and use the command, run "printenv"

command to see all current environment variables, For example:

```
APUS # help tftpboot
APUS # printenv
```

At this time, you can configure U-Boot from the network via TFTP to download the kernel to run and mount NFS network file system. Please refer to the following steps to configure: (This assumes your server IP address is 192.168.1.4, and the server has started a TFTP and NFS services; Linux core services in the TFTP directory / tftpboot, the file is named uImage; Network File the system path / nfsroot/root26; apus development board of the IP-192.168.2.84, MAC address of any given, there is 00:2 a: c6: 2c: ab: f1)

```
APUS # setenv ipaddr 192.168.2.84
APUS # setenv serverip 192.168.1.4
APUS # setenv ethaddr 00:2 a: c6: 2c: ab: f1
APUS # askenv bootcmd
bootcmd: tftpboot; bootm
APUS # setenv bootargs mem = 64M console = ttyS3, 57600n8 ip = dhcp nfsroot
= 192.168.1.4: / nfsroot/root26 rw
APUS # setenv bootfile uImage
APUS # saveenv
APUS # boot
```

"Saveenv" command will save the current configuration to Flash, the system does not require re-configured when the next restart. "Boot" command will start the runtime environment variable "bootcmd" the definition of command. U-Boot via TFTP downloads the first Linux core files uImage to the target board SDRAM, and then run "bootm" command to start a Linux core. After Linux starting, via NFS to mount to nfsroot/root26 of 192.168.1.4 as the root file system.

3.6 U-Boot Image Type

U-Boot's "boot" command to support and guide a specific type of image file, these files contain U-Boot supporting the file header information. These files are usually defined in the first operating system image file types (such as Linux, VxWorks, Solaries, etc.), CPU architecture (eg ARM, MIPS, X86), compression type (gzip, bzip2, non-compressed), download the program entry address, the image name and timestamp.

Usually the Linux core compiled in various formats, such as vmlinux, zImage and so on, these can not be used. Then need to use U-Boot provides the tools mkimage to generate U-Boot supported Linux core uImage, the command format is as follows:

tools / mkimage-A arch-O os-T type-C comp-a addr-e ep-n name-d data_file image

Of which:

-A ==> set architecture to 'arch'
-O ==> set operating system to 'os'
-T ==> set image type to 'type'
-C ==> set compression type 'comp'
-a ==> set load address to 'addr' (hex)
-e ==> set entry point to 'ep' (hex)
-n ==> set image name to 'name'
-d ==> use image data from 'datafile'

Use mkimage generate uImage the following steps:

Compiler generates a standard "vmlinux" kernel file (ELF binary format)
The "vmlinux" converted to the original binary file:

```
$ mipsel-linux-objcopy -o binary –R .note –R .comment -S vmlinux linux.bin
```

Compressed binary file:

```
$ gzip -9 linux.bin
```

mkimage packaged with U-Boot for the format of files:

```
$ mkimage.exe -A mips -o linux-T kernel -C gzip \
-a 0x80100000 -e 0x8029a040 -n "Linux Kernel Image" \
-d linux.bin.gz uImage
```

UImage is generated; use the "mkimage-l image" command look at the image file which contains the header file information, as follows:

```
$ mkimage -l uImage
Image Name: Linux Kernel Image
Created: Mon Feb 5 12:20:02 2007
Image Type: MIPS Linux Kernel Image (gzip compressed)
Data Size: 765734 Bytes = 747.79 kB = 0.73 MB
Load Address: 0x80100000
Entry Point: 0x8029A040
```

4 Linux Kernel and Drivers

We support the Linux 2.6 kernel on the XBurst platform. This section will describe how to configure and make the kernel.

4.1 Get the Linux Kernel Source

You can get the Linux kernel source from Ingenic homepage (http://www.ingenic.cn/). The source contains two packages, one is the original source tarball called linux-2.6.24.3.tar.bz2, the other is the patch from Ingenic called linux-2.6.24.3-jz-yyyymmdd.patch.gz ("yyyymmdd" represents the patch release date).

Copy the source packages to the Linux host and extract them to your work directory:

```
$ tar xjf linux-2.6.24.3.tar.bz2
$ cd linux-2.6.24.3
$ gzip -cd ../linux-2.6.24.3-jz-yyyymmdd.patch.gz | patch -p1
```

The kernel source tree is:

- arch/mips/:
  - kernel/
  - mm/
  - lib/
  - jz4730/: files related to the JZ4730 processor
    - *.c: JZ4730 common files
    - board-pmp.c: PMP board specific file
  - jz4740/: files related to the JZ4740 processor
    - *.c: JZ4740 common files
    - board-pavo: PAVO board specific
  - jz4750/: files related to the JZ4750 processor
    - *.c: JZ4750 common files
    - board-apus: APUS board specific files
  - ramdisk/
  - boot/: various image files including zImage, uImage etc.
  - Kconfig
  - Makefile
  - configs/
    - pmp_defconfig: JZ4730-PMP default configuration
    - pavo_defconfig: JZ4740-PAVO default configuration

        - apus_defconfig: JZ4750-APUS default configuration
- include/asm-mips/:
    - jzsoc.h: common header for JZSOC
    - mach-jz4730/: headers for JZ4730 processor
    - mach-jz4740/: headers for JZ4740 processor
    - mach-jz4750/: headers for JZ4750 processor
- sound
    - oss/
        - jz_i2s.c
        - jzcodec.c
        - ak4642en.c
        - jz_ac97.c
    - soc/jz4740/: ALSA for JZ4740
        - jz4740-i2s.c
        - jz4740-ac97.c
        - jz4740-pcm.c
    - soc/codecs/：ALSA CODEC driver
        - jzcodec.c
- kernel/
- mm/
- lib/
- init/
- ipc/
- net/
- fs/
    - jffs2/: JFFS/JFFS2
    - yaffs2/: YAFFS/YAFFS2
        - utils/: mkyaffs2image utility
    - ubifs/：UBIFS
- drivers/
    - block/
    - char/
    - char/jzchar/: JZSOC specific drivers
    - cpufreq/
    - input/
    - mmc/
    - mtd/
        - ubi/
        - mtd-utils/: MTD and UBI utilities, such as flash_eraseall、nandwrite_mlc、ubimkvol etc.
    - net/
    - serial/
    - ssi/

- usb/
- usb/gadget/
  - jz4740_udc.c
  - jz4730_udc.c
  - file_storage.c
- video/
  - jzlcd.c: framebuffer driver for JZ4730 and JZ4740
  - jz4750_lcd.c: framebuffer driver for JZ4750

4.2 Configure and Compile the Linux Kernel

Select a default configuration for you board:

```
$ make pmp_defconfig       (JZ4730 PMP board)
$ make pavo_defconfig      (JZ4740 PAVO board)
$ make apus_defconfig      (JZ4750 APUS board)
```

Run next command to change the configuration:

```
$ make xconfig
```

Then type next command to compile the kernel:

```
$ make uImage
```

This will generate a binary file called uImage at arch/mips/boot/ for u-boot. Then you can download and boot the kernel from u-boot.

5 Linux root file system

After the Linux kernel compilation, it must have the root file system (root file system) to make a Linux system run. This section will briefly introduce functions of the root file system, and how to create the root file system according to their own system's needs. You can also go to the website to download our production release of the root file system, use the super-user privileges can be used to test after decompression.

5.1 The contents of the file system

Root file system is used to store required applications, script, configuration files during the system running, and so on, root file system usually contains the following directories:

1. bin /, sbin /: systems executable programs and tools
2. lib /: dynamic Runtime
3. etc /: system configuration information and the startup script rcS
4. usr /: user executable program

5.2 Compile Busybox

Due to storage space constraints of embedded systems, this makes strict limitations to the size of the program. Especially pay more attention to the production of root file system,. The use of BusyBox can greatly simplify the production of the root file system. After compiling BusyBox and installation, there is only one binary executable file busybox, which implements almost all commonly used and the necessary applications (such as init, shell, getty, ls, cp, etc.), and these applications are in the form of symbolic link . For users, the metods of the command of implementation do not be changed; the command line call will be passed as a parameter to busybox to complete the corresponding function. Produced root file system by the use of BusyBox can either save a lot of space, but also save a lot of cross-compiling work.

Please visit the official website of BusyBox (http://www.busybox.net) to download the source package.Take busybox-1.8.2 as an example to explain the process to compile and install BusyBox.

Please install mipsel-linux-gcc compiler tools (gcc-4.1.2 and glibc-2.3.6) before compiling. The process of compiling BusyBox is similar to compiling Linux kernel, as follows:

```
# make defconfig
# make xconfig
# make ARCH = mips CROSS_COMPILE = mipsel-linux -
# make ARCH = mips CROSS_COMPILE = mipsel-linux-install
```

```
# cp-afr _install / * / nfsroot/root26 /
```

5.3 Compile and config udev

Device file /dev/ under the Linux 2.6 is created by udev. Udev manage the device files dynamically through the information which is provided by sysfs file system, including the creation and delete of device files and so on.

The following steps are how to compile udev:

```
1) Download udev Source
2) cd udev-117
3) make CROSS_COMPILE = mipsel-linux-DESTDIR = / nfsroot/root26
4) make CROSS_COMPILE = mipsel-linux-DESTDIR = / nfsroot/root26 install
```

Root file system configuration is as follows:

/ etc / init.d / rcS: This file contains the command to start udev

```
# Mount filesystems
/ bin / mount-t proc / proc / proc
/ bin / mount-t sysfs sysfs / sys
/ bin / mount-t tmpfs tmpfs / dev

# Create necessary devices
/ bin / mknod / dev / null c 1 3
/ bin / mkdir / dev / pts
/ bin / mount-t devpts devpts / dev / pts
/ bin / mknod / dev / audio c 14 4
/ bin / mknod / dev / ts c 10 16
/ bin / mknod / dev / rtc c 10 135

echo "Starting udevd ..."
/ sbin / udevd - daemon
/ sbin / udevstart

/ etc / udev / udev.conf: udev configuration file
/ etc / udev / rules.d / *. rules: udev's rule document
/ etc / udev / script / *. sh: udev script files
/ sbin / hotplug: hotplug scripts
/ usr / cpufreqd / *: cpufreqd library files and configuration files
/ usr / alsa / *: ALSA tools and library files
/ usr / tslib / *: tslib library file and test procedures
```

5.4 Create initramfs

Refer to the following steps to create a Linux 2.6 kernel's initramfs, assume that your root file system is located in the /rootfs directory.

First of all, create a cpio format image files:

```
# cd / rootfs
# find. | Cpio-c-o | gzip -9> .. / rootfs.cpio.gz
```

Then, reconfigure the kernel, select the following options:

```
[General setup] → [initial RAM filesystem and RAM disk (initramfs / initrd)
support]
Initramfs source file (s): / rootfs.cpio.gz
```

Recompile the kernel, according to the following configure command-line to restart the kernel:

```
root = /dev/ram0 rw rdinit =/sbin/init
```

6 How to build a complete system via using sdcard?

Jz475X processor can support boot from the sd card, that is the case in the absence of nand flash, using mmc / sd cards you can build a complete system.

First of all, rebuild the sdcard's partition table.

```
sd card partition:
0-4M u-boot
4-8M kernel
8 -? fs (the same as the hard disk, you can create any number of partitions)
```

When the kernel is configured, you should choose JZ4750 Boot from SD / Multimedia Card Interfaces support, the pre-8M will be protected in this kerenl where we have to rebuild the partition table of sdcard (using fdisk / dev/mmcblk0). In other words mbr sector offset of sdcard is from 0th to 16384th sector, and then formatted into mkfs.ext2. Process is as follows:

```
umount /mnt/mmc
fdisk /dev/mmcblk0
'p'  //check how many partitions
'd' // delete partion
'n' // bulid a new partition
'p' // build the primary partition
number: 1
        enter
        enter
        w  //write data

mkfs.ext2 /dev/mmcblk0p1  //format as ext2

mount -t ext2 /dev/mmcblk0p1 /mnt/mmc  // if you format as ext2, you
should use ext2 as parameter
```

Secondly, the sdcard should be mount, and then copy all directories and files of the root file system to sdcard. Use our usb-boot burming tool or sdcard buring tool to burn the u-boot and kenerl to 0 and 8192 sector respectively. As follows:

(1) Copy all directories and files of the root file system to sdcard:

Users can download tar package of the root file system on ingenic's website(root.tar), boot the target board through the network, put this package on any directory like /home/zhzhao/.

```
mount -t /dev/mmcblk0p1 /mnt/mmc
cd /mnt/mmc
tar xvf /home/zhzhao/root.tar
```

2) burn u-boot and kernel:

If it is compiled for apus-board from sdcard to start u-boot:

```
make apus_msc_config
make
```

If it is compiled cetus-board from sdcard to start u-boot:

```
make cetus_msc_config
make
```

U-boot and kernel with sdcard tool or use usb tool to burn. if you use usb tool, the command as follows:

```
sdprog 0 u-boot-mcs.bin
sdrpog 8192 uImage
```

Finally, specify the parameters in the u-boot.

For apus-board:

```
bootargs mem=64M console=ttyS3,57600n8 ip=off root=/dev/mmcblk0p1 rw
bootcmd msc read 0x80600000 0x400000 0x200000;bootm
```

For cetus-board:

```
bootargs mem=64M console=ttyS1,57600n8 ip=off root=/dev/mmcblk0p1 rw
bootcmd msc read 0x80600000 0x400000 0x200000;bootm
```

Note: Start u-boot of cetus-board from sdcard, you should press SW2 first, and then press reset.

Start u-boot of apus board from sdcard, you should hold down the SW6+SW7, and then press reset.

As mentioned above, the complete system was built up.

Use this sdcard as a master, and use our tools to read the root file system partition as a binary file, assuming that the file system size is less than 100M, we have to read the contents of 100M, then the sector length is 204800 = (100 * 1024 * 1024) / 512 command as follows:

```
sdread 16384 204800 aa.bin
```

It would also read out the partition table, use burning tool to burn to the other sdcards. Commands are as follows:

```
sdprog 0 u-boot-mcs.bin
sdrpog 8192 uImage
sdprog 16384 aa.bin
```

7 Test Linux kernel and drivers

This section briefly introduces how to test the ingenic Linux 2.6.24.3 device driver, verify the driver and the hardware module whether is working correctly or not.

7.1 Run Linux kernel

We can guide the program U-Boot to boot Linux kernel and mount the root file system to test the kernel whether are normal or not. First, configure and compile the Linux kernel, uImage will be genertated, run it through the U-Boot boot. If Linux can run normally and mount the root file system successfully, which shows Linux kernel run properly.

Common Linux command-line parameters:

mem = xxM set memory size
console = tty0 console output is set to tty0
console = ttyS0, 57600n8 set the console output to serial port
ttyS0, baud rate 57600,8 of data bits,
no parity bit
ip = dhcp IP address via DHCP service to obtain
ip = 192.168.2.84 set a static IP to 192.168.2.84
nfsroot = 192.168.1.4: / nfsroot/root26 rw
set up a network root file system with read and write permissions
root = / dev/mtdblock2 rw root file system settings / dev/mtdblock2, read and write permissions
rootfstype = yaffs2 root file system type YAFFS2

Ingenic Linux add some command-line parameters additionally:

ts_debug enabled touch-screen test pattern

7.2 Test Linux device driver

o **Test LCD display**

Select the right LCD screen model of target board when you config Linux, A picture of the output of LCD panels should be able to be seen. If not, please check the hardware connection of the LCD screen .

o **Test the audio playback and recording**

MP3 Player Test:

```
# madplay test.mp3
```

Recording and playback recording test：

```
# vrec -S -s 48000 -b 16 sample1
# vplay -S -s 48000 -b 16 sample1
```

o Test Video Player

Run mplayer on the optimized Jz4730：

```
# mplayer30 -vf scale=480:272 binhe.avi
```

Run mplayer on the optimized Jz4740:

```
# mplayer40 -vf scale=480:272 binhe.264
```

o **Test the file system JFFS2 and YAFFS2**

When configure Linux, select MTD device and JFFS2/YAFFS file system. After booting Linux, test the file system JFFS2 base on MTD and YAFFS2:

First of all, view the MTD partition table's information:

```
# cat /proc/mtd
```

Then, format the MTD partition：

```
# flash_eraseall -j /dev/mtd3     JFFS2 file system formatted
# flash_eraseall /dev/mtd4    YAFFS2 file system formatted
```

Then, mount and test the file system:

```
# mount -t jffs2 /dev/mtdblock3 /mnt/mtdblock3  JFFS2 partition loaded
# mount -t yaffs2 /dev/mtdblock3 /mnt/mtdblock4   YAFFS2 partition
loaded
# cp test /mnt/mtdblock4
# umount /mnt/mtdblock4
```

If this works, note the file system has been working.

o **Test MMC / SD Card**

After starting Linux normally, insert the MMC/SD card into the SD card connector, Linux will automatically detect the card inserted and read out the partition table's information, then you can mount and operate SD card:

```
# mount -t vfat /dev/mmcblk0p1 /mnt/mmc
# ls /mnt/mmc
```

o **Touch Screen Test**

After start-up Linux, run the following command to calibrate Touch screen:

```
# ts_calibrate
```

Run the following command to test the touch screen:

```
# ts_test
```

o **Testing USB Host**

Plug USB devices (such as U disk) to the USB Host port, Linux should be able to detect when a device is inserted, then you can mount and operation of U disk:

```
# mount -t vfat /dev/sda1 /mnt/usb
# ls /mnt/usb
```

o **Test USB Device**

First of all, make modules to compile and generate module of USB Device Drive: jz4740_udc.ko and g_file_storage.ko; and then load the driver in accordance with the following methods:

```
# insmod jz4740_udc.ko
# insmod g_file_storage.ko file=/dev/mtdblock5,/dev/mmcblk0
```

Then we mount the two U disk partitions, one is based on NAND Flash partition mtdblock5, the other is based on the SD card partition. Now connect PC and target board USB Device port with the USB cable, PC should be able to identify the insertion of the device, and identify as U disk device.

o **Test Power Management**

a. Test System sleep, wake-up and shutdown
Run the following command, the system will enter a sleep state, wake up after you press power button.

```
# echo 1 > /proc/sys/pm/suspend
```

Run the following command, the system will shut down after you press power button to restart.

```
# echo 1 > /proc/sys/pm/hibernate
```

b. Test Dynamic Frequency Management

When you compile kernel, select userspace as the default governor in the CPU Frequency scaling options. This can control the cpufreq driver through the sys file system interface after booting the target board. If not mount sys file system, then the implement is needed.

```
# mount –t sysfs sysfs /sys
```

When compiling the kernel, if the default option is governor not a userspace, then execute a command to change the governor as userspace:

```
# echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Use the following command to see the current system frequency (Unit: KHz):

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

In addition, a more detailed system frequency can be obtained via the following command; the command has nothing to do with the cpufreq:

```
# cat /proc/jz/cgm
CPPCR         : 0x1b000520
CPCCR         : 0x406c4442
PLL           : ON
m:n:o         : 56:2:1
C:H:M:P       : 3:6:6:6
PLL Freq      : 336.00 MHz
CCLK          : 112.00 MHz
HCLK          :  56.00 MHz
MCLK          :  56.00 MHz
PCLK          :  56.00 MHz
LCDCLK        :  25.84 MHz
PIXCLK        : 9081.08 KHz
I2SCLK        :  12.00 MHz
USBCLK        :  12.00 MHz
  MSCCLK        :  24.00 MHz
```

```
EXTALCLK      :  12.00 MHz
RTCCLK        :   0.03 MHz
```

Look at the current minimum system frequency which can be set:

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

Look at the current maximum system frequency which can be set:

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

Set the current system frequency (the frequency unit is KHz, in order to set to 112MHz, for example):

```
# echo 112000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

When compiling the kernel, check CPU frequency translation statistics details in the CPU Frequency scaling options, you can use the three commands to see frequency conversion's records. First, use the following command to obtain a history of frequency conversion's table:

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/stats/trans_table
From  :    To
      :    42000    56000    84000    112000    168000    336000
 42000:        0        0        0         0         0         3
 56000:        0        0        0         0         0         0
 84000:        0        0        0         0         0         0
112000:        0        0        0         0         0         0
168000:        0        0        0         0         0         0
336000:        3        0        0         0         0         0
```

The following command can be use to get the system's work time at various frequencies, hours of work, unit time is the same with unit of jiffies, generally it is 10ms (is decided by Timer frequency of Kernel type when compile kernel). The following example, the system works in the 42000KHz, the work time is 23676 * 10ms.

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state
42000 23676
56000 0
84000 0
112000 0
168000 0
336000 61957
```

Use the following commands available to get the total number of frequency

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/stats/total_trans
6
```

8 Linux 2.6 audio driver

The audio driver of Linux 2.6 including ALSA and OSS can be used alone, but also can be used simultaneously, the followings are their usage.

8.1 OSS Audio Driver

OSS audio driver is in the linux-2.6.24.3/sound/oss directory. Currently ak4642en and internal codec driver of jz4740 are provide, the relevant code are jz_ac97.c, jz_i2s.c, ak4642en.c and jzcodec.c.

When using the OSS audio driver, in the make xconfig pop-up window, select a codec of sound /Open Sound System/Obsolete OSS drivers/jz On-Chip I2S driver, and then make uImage to compile the kernel.

8.2 ALSA audio driver

ALSA audio driver is in the linux-2.6.24.3/sound/soc which offers jz4740 and its internal codec driver, the relevant code is in the directory jz4740 and codecs of soc, jz4740 directory includes I2S support, codec directory includes the internal codec support.
When using the ALSA audio driver, after make xconfig pop-up window, select the relevant option of sound / Advanced Linux Sound Architecture / System on Chip audio support, and is also optional on the support of OSS emulate the options, and then make uImage to compile the kernel.
ALSA library support is needed when executing ALSA application program, ALSA library places in the /usr/alsa/lib; ALSA utils test procedures placed in the / usr / alsa / bin directory.
ALSA configuration file is located in /usr/alsa/share/alsa.conf, whose path is decided by environment variable ALSA_CONFIG_PATH in /etc/profile.
Sound card's configuration file places in the /usr/alsa/etc/asound.conf, it specifies the name of some equipment, frequency conversion and mixing plugin; the document's path is decided by the ALSA configuration file.

8.3 ALSA audio test

Refer to the following command to test the ALSA:

Set playback gain (0 to 3):

```
# Amixer set Master 1
```

View playback gain:

```
# amixer get Master
```

Set Recording Gain (0 to 31):

```
# amixer set Line 10
```

View Recording Gain:

```
# amixer get Line
```

Test record:

```
# arecord -d 20 -c 2 -t wav -r 8000 -f "Signed 16 bit Little
Endian" ./test.wav
```

-d: 20 Miao
-c: 2 Ge channel
-t: File Types
-r: Frequency
-f: 16bits

Test playback:

```
# aplay ./test.wav
```

Or

```
# aplay -D primary ./test.wav
```

Reference asound.conf which specified primary

Test Frequency:

```
# aplay -D rate_44k ./test.wav
```

Reference asound.conf which specified rate_44k

Test Mix:

```
# aplay -D plug:dmix_44k ./test.wav &
```

Re-implementation

```
# aplay -D plug: dmix_44k ./test.wav
```

Reference asound.conf which specified dmix_44k

9 NAND Flash File System

In the consumer electronics, NAND Flash is more and more widely used. Ingenic processor JZ47XX integrated NAND Flash controller, NAND Flash supported and a direct connection with NAND Flash. At the same time, Ingenic Linux 2.6 kernel also provides a support for NAND Flash file system; this section describes how to build the file system based on NAND flash on the Linux 2.6.

9.1 NAND Flash Drive

Linux needs to configure CONFIG_MTD, CONFIG_MTD_PARTITIONS, CONFIG_MTD_CHAR, CONFIG_MTD_BLOCK, CONFIG_MTD_NAND as y when compiling Linux, and in accordance with the ingenic processor type, configure CONFIG_MTD_NAND_JZ4730, CONFIG_MTD_NAND_JZ4740 or CONFIG_MTD_NAND_JZ4750 as y.

JZ4730 processor integrates hardware HAMMING ECC algorithm which can detect 2 bits error bits in the data of 256 bytes, and can correct 1 bit error bit; JZ4740 integrated hardware REED-SOLOMON ECC algorithm can correct at least 4 bits error of the data of 512 bytes.

JZ4750 integrated hardware 4-bit and 8-bit BCH ECC algorithm, the former corrected 4 bits error within 1016 bytes, the last corrected 8 bits errors within the 1010 bytes. In the driver, we let them correct 4 or 8 bits errors in each (512 + n) bytes, where n = (the part number of bytes before the ECC stored in the oob / eccsteps), eccsteps = pagesize / 512. The JZ4750's NAND driver supports the oob area (used to store information of the file system) of the verification (only support when oob with data were read out). In addition, it is strongly recommended for JZ4750, NAND way using DMA read and write operations, compile linux to let CONFIG_MTD_NAND_DMA = y, which can greatly reduce the processor burden, and improve system efficiency. Propose that CONFIG_MTD_NAND_DMABUF = n, otherwise, multiple copies of a buffer affects the speed of NAND read and write.

JZ4740, and JZ4750 support multi-chip NAND choice, chip choice CS1_N, CS2_N, CS3_N and CS4_N can access NAND, driver default CS1_N, if use the other chip choice, make CONFIG_MTD_NAND_CS2, CONFIG_MTD_NAND_CS3 or CONFIG_MTD_NAND_CS4 = y when compiling. JZ4730 only supports the pick CS3_N on NAND, but you can use address decoding to link multi-chip NAND, and make some changes to support in the driver.

On the market, currently, a lot of NANDs support multi-plane to read and write operations, conducting multi-plane write operation, NAND can synchronize to read and write the corresponding page of two planes, which greatly reduce the waiting time to write, significantly speed up the NAND's write speed. In addition to JZ4730, Ingenic NAND driver has supported the multi-plane to write. For the multi-plane read, the current ingenic NAND driver approach is to read the page content of two separate planes, multi-plane on the read speed does not increase. While some

NAND support for special multi-plane read command, but even use it, it will not speed up the speed of read, because the time mainly consume outside bus's NAND read, rather than waiting for NAND internal busy ready. To use multi-plane properties of NAND, need to make CONFIG_MTD_NAND_MULTI_PLANE = y at compile-time. If the selected NAND does not support the multi-plane properties, it is best to make CONFIG_MTD_NAND_MULTI_PLANE = n, so as to avoid mistaking identification in support of its multi-plane characteristics of NAND.

Ingenic has changed linux MTD system as 64bit, in order to support size of NAND which is more than 2GB. And support the 4KB pagesize of NAND.

9.2 NAND Flash File System Type

Linux use the MTD (Memory Technology Devices) driver to manage the NAND Flash device. Because the JFFS2 file system achieved NFTL (NAND Flash Translation Layer) function, these two file systems can be directly built on the MTD NAND Flash device.

Ingenic also modified mtdblock block device driver to support NAND Flash file system. The revised mtdblock block device driver increase the logical block addresses to physical block address mapping, bad block management, and loss of balance and other functions. In this way, users can build EXT2, EXT3 and FAT file systems on mtdblock block device of NAND Flash and so.

9.3 MTD partition

Prior to the use of NAND Flash, define partition first. MTD partition was defined in linux-2.6.24.3/drivers/mtd/nand / under jz4730_nand.c, jz4740_nand.c or jz4750_nand.c file. Users need to define the partition information of physical block of NAND Flash according to their own system.

Ingenic added three members based on standard information: use_planes, cpu_mode and mtdblock_jz_invalid. Use_planes which is used to decide whether to use multi-plane partition read and write operations or not, the value 0 means do not use multi-plane, that value 1 means use. If the selected NAND flash does not support the multi-plane operation, then the value of items use_planes meaningless. At present, jz4730 does not support multi-plane operation, so that the settings do not work. If the selected NAND does not support the multi-plane, or all the partitions do not use multi-plane, it is best to configure the linux compile option, let CONFIG_MTD_NAND_MULTI_PLANE = n.

cpu_mode is used to specify the partition using the cpu mode or dma mode.

mtdblock_jz_invalid is used to indicate the partition whether to use the file system was provided conversion layer mtdblock_jz.c by Ingenic, such as YAFFS2 partition do not need to use it, mtdblock_jz_invalid will be set to 1, then mount YAFFS2 partition, it will not do something to scan, not to be allocated a number of unnecessary memory; VFAT partition need to use mtdblock_jz.c, set mtdblock_jz_invalid to 0, otherwise use the partition will be reported exception. If you set CONFIG_ALLOCATE_MTDBLOCK_JZ_EARLY = y, and that the partition to use dma mode, then when the NAND driver was initialized, contiguous physically NAND block cache will be given to all

partition which use a partition of mtdblock_jz.c, which is particularly applicable to U disk to use VFAT partition.

If the partition uses cpu mode, there is no need to apply for the cache in advance, because at this time does not require contiguous physical memory.

Here we use 2GB NAND Flah in jz4750 as an example to introduce to the readers how to define your own partition:

```
Parameters:
Page size: 4096 Bytes
Block size: 512 KB
Pages per block: 128
Row address cycles: 3
Total size: 2GB
Total blocks: 4096
```

```
NAND partition table information:
```

Table 1 2GB NAND Flash partition table

partition	The start of physical block	The end of the physical block	Size of partition	Description of partition	Whether to use multi-plane or not	Whether to use Cpu mode or not	Whether to use Mtdblock-jz or not
MTD BLOCK 0	0	7	4MB	MTD partition 0 (bootloader)	Not use	Not use	Not use
MTD BLOCK 1	8	15	4MB	MTD partition 1 (kernel)	Not use	Not use	Not use
MTD BLOCK 2	16	1023	504MB	MTD partition 2 (rootfs)	Not use	Not use	Not use
MTD BLOCK 3	1024	2047	512MB	MTD partition 3 (data)	Use	Not use	Not use
MTD BLOCK 4	2048	4095	1GB	MTD partition 4 (vfat)	Not use	Not use	use

MTD partition was defined in the linux-2.6.24.3/drivers/mtd/nand/jz4750_nand.c, as follows:

```
static struct mtd_partition partition_info[] = {
    {name:"NAND BOOT partition",
```

```
    offset:0 * 0x100000,
    size:4 * 0x100000,
    cpu_mode: 0,
    use_planes: 0,
    mtdblock_jz_invalid: 1},
   {name:"NAND KERNEL partition",
    offset:4 * 0x100000,
    size:4 * 0x100000,
    cpu_mode: 0,
    use_planes: 0,
    mtdblock_jz_invalid: 1},
   {name:"NAND ROOTFS partition",
    offset:8 * 0x100000,
    size:504 * 0x100000,
    cpu_mode: 0,
    use_planes: 0,
    mtdblock_jz_invalid: 1},
   {name:"NAND DATA partition",
    offset:512 * 0x100000,
    size:512 * 0x100000,
    cpu_mode: 0,
    use_planes: 1,
    mtdblock_jz_invalid: 1},
   {name:"NAND VFAT partition",
    offset:1024 * 0x100000,
    size:1024 * 0x100000,
    cpu_mode: 0,
    use_planes: 1,
    mtdblock_jz_invalid: 0},
   };
```

9.4 Create YAFFS2 File System

Becaise YAFFS2 completed NAND Flash management functions such as address mapping, bad block management and so on, so you can directly create YAFFS2 file system on the MTD partition.

Create YAFFS2 file system in the MTD partition are as follows:

```
# flash_eraseall /dev/mtd3
# mount –t yaffs2 /dev/mtdblock3 /mnt/mtdblock3
# cp test /mnt/mtdblock3
# umount /mnt/mtdblock3
```

You can also use mkyaffs2image tool to generate YAFFS2 image, and then use the command

nandwrite_mlc write MTD partition, as follows:

On the host PC, run the following command to generate yaffs2 image：

```
# mkyaffs2image 1 /rootfs/ rootfs.yaffs2
```

On target board:

```
# flash_eraseall /dev/mtd2
# nandwrite_mlc -a -o /dev/mtd2 rootfs.yaffs2
```

Above flash_eraseall and nandwrite_mlc tools's source is located in `linux/drivers/mtd/mtd-utils/`, Ingenic make some changes to support more than MLC NAND of 2GB. nandwrite_mlc also be used for SLC NAND.

Run the command mkyaffs2image on the host PC to generate YAFFS2 image, such as:

```
$ mkyaffs2image 2 /rootfs/ rootfs.yaffs2
```

Use mkyaffs2image on PC, the source located in linux/fs/yaffs2/utils/.

```
usage: mkyaffs2image layout# source image_file [convert]
    layout#     NAND OOB layout:
                    0 - nand_oob_raw, no used,
                    1 - nand_oob_64, for 2KB pagesize,
                    2 - nand_oob_128, for 2KB pagesize using multiple planes or 4KB
                        pagesize,
                    3 - nand_oob_256, for 4KB pagesize using multiple planes
    source          the directory tree or file to be converted
    image_file  the output file to hold the image
    'convert'   make a big-endian img on a little-endian machine. BROKEN !
```

9.5 Creating FAT and EXT2 file system

Related command example is as follows:

```
# flash_eraseall /dev/mtd3
# mkfs.vfat /dev/mtdblock3
# mount -t vfat /dev/mtdblock3 /mnt/mtdblock3
# cp test /mnt/mtdblock3
# umount /mnt/mtdblock3

# flash_eraseall /dev/mtd3
# mkfs.ext2 /dev/mtdblock3
```

```
# mount -t ext2 /dev/mtdblock3 /mnt/mtdblock3
# cp test /mnt/mtdblock3
# umount /mnt/mtdblock3
```

9.6 UBI Equipment

Linux 2.6 kernel introduced the UBI (Unsorted Block Images) to manage Flash. Create UBIFS file system on the basis of the UBI. Also in order to be able to directly create EXT2, and FAT file system on the UBI, added UBI BLOCK device layer driver (ubiblk.c and bdev.c).

UBI layer accomplish the following functions:

• Bad block management
• loss of balance
• the logical to physical block address mapping
• Volume information storage
• Device Information

Enable UBI, UBIFS and UBI BLOCK device drivers:

In the Linux configuration menu, find the "Device Drivers" -> "Memory Technology Device (MTD) support" -> "UBI - Unsorted block images", select "Enable UBI" and "Common interface to block layer for UBI 'translation layers' "and" Emulate block devices ". UBI and UBI BLOCK either compiled in modules also can be directly compiled into the kernel.

In the Linux configuration menu, find the "File systems" -> "Miscellaneous filesystems", then select "UBIFS file sysem support". UBIFS either compiled in modules also can be directly compiled into the kernel.

Run "make modules" to compile generation module, run "make modules_install INSTALL_MOD_PATH = / nfsroot/root26" to install the module to the root file system directory.

Here we only introduce the use and test methods of UBIFS and UBI BLOCK, the more technical details, please refer to the following homepage:

UBI Homepage: http://www.linux-mtd.infradead.org/doc/ubi.html
UBIFS Home: http://www.linux-mtd.infradead.org/doc/ubifs.html

9.6.1 UBIFS

It is assumed that you have compiled ubi and ubifs drive, and has been installed to the root file

system / lib / modules directory, you can use it the following:

After the system started, MTD partition table is as follows：

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00400000 00040000 "NAND BOOT partition"
mtd1: 00400000 00040000 "NAND KERNEL partition"
mtd2: 07800000 00040000 "NAND ROOTFS partition"
mtd3: 08000000 00040000 "NAND DATA1 partition"
mtd4: 10000000 00040000 "NAND DATA2 partition"
mtd5: 20000000 00040000 "NAND VFAT partition"
```

Take mtd5 as a example to introduce how to use UBI and UBIFS.
First, format this partition:

```
# flash_eraseall /dev/mtd5
```

Second, take mtd5 as a ubi device, load ubi driver:

```
# modprobe ubi mtd=5
UBI: empty MTD device detected
UBI: create volume table (copy #1)
UBI: create volume table (copy #2)
UBI: attached mtd5 to ubi0
UBI: MTD device name:            "NAND VFAT partition"
UBI: MTD device size:            512 MiB
UBI: physical eraseblock size:   262144 bytes (256 KiB)
UBI: logical eraseblock size:    258048 bytes
UBI: number of good PEBs:        2048
UBI: number of bad PEBs:         0
UBI: smallest flash I/O unit:    2048
UBI: VID header offset:          2048 (aligned 2048)
UBI: data offset:                4096
UBI: max. allowed volumes:       128
UBI: wear-leveling threshold:    4096
UBI: number of internal volumes: 1
UBI: number of user volumes:     0
UBI: available PEBs:             2024
UBI: total number of reserved PEBs: 24
UBI: number of PEBs reserved for bad PEB handling: 20
UBI: max/mean erase counter: 0/0
UBI: background thread "ubi_bgt0d" started, PID 754
```

Here we must note that the blue tag. Indicates that this UBI device has a total used 2024 physical blocks, retain 24 physical blocks. In this UBI device to establish the total size of all the volumes are: 2024 * logical eraseblock size (here for 258048bytes) that the
2024 * 258048 / 1024 / 1024 = 498.09375 M to take the integer part, that is 498M

Then, in the ubi device to create ubi volume can create up to 128 volumes, where we have created two, the volume's name are "ubifs" and "vfat"：

```
# ubimkvol /dev/ubi0 -s 200MiB –N ubifs
# ubimkvol /dev/ubi0 -s 298MiB –N vfat
```

After ubifs driver loaded, ubifs can be used directly.

```
# modprobe ubifs
```

Load ubifs partition, ubi0:ubifs are one method to visit ubifs:

```
# mount –t ubifs ubi0:ubifs /mnt/ubifs
UBIFS: default file-system created
UBIFS: mounted UBI device 0, volume 0
UBIFS: minimal I/O unit size:   2048 bytes
UBIFS: logical eraseblock size: 258048 bytes (252 KiB)
UBIFS: file system size:        207212544 bytes (202356 KiB, 197 MiB,
803 LEBs)
UBIFS: journal size:            10321920 bytes (10080 KiB, 9 MiB, 40
LEBs)
UBIFS: data journal heads:      1
UBIFS: default compressor:      LZO
```

Note: The following description of the ubi block equipment, after creating volume, the first need to uninstall ubi module and re-add and then load the ubi block device driver modules.

9.6.2 UBI Block Device

It is assumed that you have compiled ubi and block device drivers, and has installed to the root file system / lib / modules directory, please refer to the use of the following methods:

Boot into the Linux kernel command line interface, in the command-line interface, in accordance with the following steps to test the UBI block device.

```
# flash_eraseall /dev/mtd5           -- format mtd partition
# modprobe ubi mtd=5                    -- load UBI driver
# ubimkvol /dev/ubi0 -s 200MiB -N vfat  -- create UBI volume 0
# ubimkvol /dev/ubi0 -s 298MiB -N ext2  -- create UBI volume 1
# rmmod ubi                          -- load UBI driver
# modprobe ubi mtd=5                   -- re-load UBI driver
# modprobe ubiblk               -- load UBI block device driver
```

Now , you can check the two generated device of ubiblock in /dev.

```
# ls –l /dev/ubiblock*
brw-r-----    1 root     root      254,    0 Jan  1 00:06 /dev/ubiblock0
brw-r-----    1 root     root      254,    1 Jan  1 00:06 /dev/ubiblock1
```

Now, you can format the ubiblock device, and copy files, as follows:

```
# mkfs.vfat /dev/ubiblock0
# mount -t vfat /dev/ubiblock0 /mnt/ubiblock0
# mkfs.ext2 /dev/ubiblock1
# mount -t ext2 /dev/ubiblock1 /mnt/ubiblock1
```

After the system restarts, only need to follow the following steps to load the driver which can use ubiblock device, / deV/ubiblock0 is our way to access the block device volume:

```
# modprobe ubi mtd=5
# modprobe ubiblk
# mount -t vfat /dev/ubiblock0 /mnt/ubiblock0
# mount -t ext2 /dev/ubiblock1 /mnt/ubiblock1
```

9.6.3 Production UBI image file

Use mkfs.ubifs tools to generate the image file which contains a variety of formats volume. After writing the image file to nand, the volumes and files may be used directly. NOTE: The compilation and use of mkfs.ubifs need to install the lzo library (lzo-2.02.tar.gz), go to the following address to download resources, the specific operations inside the mkfs.ubifs / README document.

Download：ftp://ftp.ingenic.cn/3sw/01linux/07utils/linux-nand-utils.tar.gz

Suppose producing such an image file, name of ubi.img, which consists of two volumes ubifs and vfat, respectively ubifs format and vfat format. Concrete steps are as follows:

First, use mkfs.ubifs tools to create ubifs Volume Mirror ubifs.img0:

```
$ export LD_LIBRARY_PATH=/opt/lzo/lib:$LD_LIBRARY_PATH    //operate on pc
$ ./mkfs.ubifs -r /nfsroot/root26/ -m 2048 -e 258048 -c 813 -o ubifs.img0
```

```
-c specifies the volume where the maximum effective capacity for the number
of logical blocks
-e logical block size (bytes)
```

Use ubinfo tools to get such data on the developed board

Then, produce vfat volume mirror vfat.img0 in Linux systems, this image size is 60M:

```
$ dd if=/dev/zero of=vfat.img0 bs=1M count=60  //operate on pc
#losetup /dev/loop0 vfat.img0        //operate on pavo board
#mkfs.vfat /dev/loop0
#mount -t vfat /dev/loop0 /mnt/udisk
#cp /mnt/mmc/* /mnt/udisk/      //copy files which are needed in the
image
#umount  /mnt/udisk
#losetup -d /dev/loop0
losetup: : No such device or address
```

With ubirefimg: format ubifs.img0 and vfat.img0 into a LEB document ubifs.img and vfat.img:

```
$. / ubirefimg ubifs.img0 ubifs.img
$. / ubirefimg vfat.img0 vfat.img
```

Then use ubinize to make ubifs.img and vfat.img to ubi.img. Prepare a cfg configuration file, as follows:

```
# cat ubinize.cfg
[ubifs]
mode=ubi
image=ubifs.img
vol_id=0
vol_size=200MiB
vol_type=dynamic
vol_name=ubifs
vol_alignment=1
vol_flag=autoresize

[vfat]
mode=ubi
image=vfat.img
vol_id=1
vol_size=298MiB
vol_type=dynamic
vol_name=vfat
vol_alignment=1
vol_flag=autoresize
```

Then run:

```
# ./ubinize -o ubi.img ubinize.cfg -p 262144 -m 2048
```

At this point, image files ubi.img has now been completed.

And then format the partition, use nandwrite_mlc tools to write ubi.img to mtd5 on nand flash.

```
# flash_eraseall /dev/mtd5
# nandwrite_mlc -a -m -q /dev/mtd5 ubi.img
```

After loading ubi-driven equipment, ubi can be used, load ubi driver:

```
# modprobe ubi mtd=5
# modprobe ubifs
# modporbe ubiblk
# mount -t ubifs ubi0:ubifs /mnt/ubifs/
# mount -t vfat /dev/ubiblock1 /mnt/ubiblock1/
# ls /mnt/ ubiblock1/
```

You will find volume image file which is pre-included vfat volume is already there.

9.6.4 Update the volume with the contents of ubiupdatevol

First delete all the contents on vfat volume

```
# rm /mnt/ubiblock1/* -rf
# df
Filesystem           1k-blocks      Used Available Use% Mounted on
tmpfs                    30224        64     30160   0% /dev
ubi0:ubifs              197536     48172    149364  24% /mnt/ubifs
/dev/ubiblock1           61302         0     61302   0%
/mnt/ubiblock1

# umount /mnt/ubiblock1
```

Update vfat volumes with ubiupdatevol (below vfat.img through ubirefimg formatted image files).

```
# ubiupdatevol /dev/ubi0_1 vfat.img
# mount -t vfat /dev/ubiblock1 /mnt/ubiblock1
# df
Filesystem           1k-blocks      Used Available Use% Mounted on
tmpfs                    30224        64     30160   0% /dev
ubi0:ubifs              197536     48172    149364  24% /mnt/ubifs
```

```
/dev/ubiblock1           61302     12716     48586  21%
/mnt/ubiblock1
# ls /mnt/ubiblock1
bingheshiji.avi  binhe.avi
```

Update Successfully.

9.6.5 Create a root file system with UBIFS

UBIFS can be used to create the root file system, some needs should be done:

1) UBI, and UBIFS compiled into the kernel code
2) Set UBI associated MTD partition in the Linux command-line parameters: "ubi.mtd = 5"
3) Specify the Linux command-line arguments UBI volumes as root file system: such as "root = ubi0: ubifs rootfstype = ubifs"

The following are examples:
If you have already created a ubifs and vfat with UBI empty volumes, please refer to the following steps. If you have written ubifs the image file to nand, and included ubifs in root file system, may omit steps 1, 2.

1) To set the linux command-line arguments with a "ubi.mtd = 5" to start the kernel to mount the root file system with NFS;
2) At system startup, follow the following steps to mount UBI partition, copy files, etc. :

```
#mount –t ubifs ubi0:ubifs /mnt/ubifs
# cp –afr /test-root/* /mnt/ubifs
# umount /mnt/ubifs
```

3) Restart the system, set the linux command-line parameters as following:

```
ubi.mtd = 5 root = ubi0: ubifs rootfstype = ubifs rw
```

10 Linux Power Management

Because battery life of handheld embedded devices is short, operating power consumption and standby power consumption should be reduced. Ingenic Linux implements power management function, dynamic frequency occurred when system is running, and system will go into sleep mode when it is not used in a long time, Dynamic Frequency Management (cpufreq) can allow the system to change frequency within the scope of about 50MHz to 400MHz, and when the system is in sleep mode, current can be reduced to below 2mA.

10.1 Dynamic Frequency Management

Ingenic Linux2.6 has implemented the dynamic frequency management; CPU's work frequency can be changed via communication between users' layer and management function of kernel. JZ47xx dynamic frequency management-related code is located in the code tree arch/mips/jz47xx / directory cpufreq.c. in the Linux kernel.

Linux dynamic Frequency Management cpufreq provides the dynamic frequency function of operating system level, also need to develop and implement user-level strategies.

To use a background process cpufreqd at the user level, first need to configure the linux kernel compile option, select CPU Frequency scaling option, within which there are a number of sub-options, including the governor's choice and whether to enable CPUfreq debugging or not. cpufreq has five governor (described in 10.1.3), make cpufreqd run normally, at least need to select the performance governor; in the testing phase, if you want to see the system frequency information, then you need to enable CPUfreq boot parameter with debugging. If you have selected Enable CPUfreq debugging, you can add linux boot parameter with loglevel = 8 cpufreq.debug = <value> in the u-boot, you can see the cpufreq information of corresponding level of operating. <value> can be 1,2,4, it can be to their or (3,5,6,7), the specific meanings are as follows:
1 to activate CPUfreq core debugging,
2 to activate CPUfreq drivers debugging (which is and JZ47xx related debugging), and
4 to activate CPUfreq governor debugging

Background process cpufreqd need to use three libraries, respectively cpufreqd-2.2.1, cpufrequtils-002 and sysfsutils-2.1.0, these three libraries are free open-source GNU software, we made some changes to cpufreqd-2.2.1, cpufrequtils - 002, so require you to download from our web site contains the three library source package cpufreqd.tar.gz. The following will introduce cpufreqd cross-compilation and installation process and the method of installation and run on the target board.

10.1.1 Cross-compiling and installation on host

First, extract the source package:

```
$ tar -xzvf cpufreqd.tar.gz
$ cd cpufreqd
```

The following access the subdirectory of cpufreqd-2.2.1, cpufrequtils-002 and sysfsutils-2.1.0 respectively, then cross-compile and install.

10.1.1.1 Compile sysfsutils-2.1.0

Entry sysfsutils-2.1.0 directory:

```
$ cd sysfsutils-2.1.0
```

Then run the following command to start to compile and install:

```
$ ./configure --prefix=<sysfsuitls_install_dir>
--host=mipsel-linux
$ make && make install
```

On the host, in the installation path <sysfsuitls_install_dir> will generate the following documents (*. la is a libtool-generated temporary file):

```
|-- bin
|     |-- dlist_test
|     |-- get_device
|     |-- get_driver
|     |-- get_module
|     `-- systool
|-- include
|     `-- sysfs
|           |-- dlist.h
|           `-- libsysfs.h
|-- lib
|     |-- libsysfs.a
|     |-- libsysfs.la
|     |-- libsysfs.so -> libsysfs.so.2.0.1
|     |-- libsysfs.so.2 -> libsysfs.so.2.0.1
|     `-- libsysfs.so.2.0.1
`-- man
```

```
    `-- man1
        `-- systool.1
```

10.1.1.2 Compile cpufrequtils-002

Entry cpufrequtils-002 directory:

```
$ cd ../cpufrequtils-002
```

Makefile need to re-assign two variables:

```
sysfsuitls_install_dir = <sysfsuitls_install_dir>
DESTDIR = <cpufrequtils_destdir>
```

<sysfsuitls_install_dir> is path of installation of sysfsuitls, <cpufrequtils_destdir> is cpufrequtils library prefix of installation path, DESTDIR can be empty, cpufrequtils's installation path is $ (DESTDIR) / usr.
Compile and run the following command to start installation:

```
$ make && make install
```

Make install process would be an error message:

```
/ usr / bin / install: cannot stat `.libs / libcpufreq.lai ': No such
file or directory
make: *** [install-lib] Error 1
```

The solution will be that cpopy .libs / libcpufreq.la as .libs/libcpufreq.lai, and change installed = no as installed = yes, then make install. In the installation path $(DESTDIR)/usr will generate the following documents:

```
|-- bin
|    |-- cpufreq-info
|    `-- cpufreq-set
|-- include
|    `-- cpufreq.h
|-- lib
|    |-- libcpufreq.a
|    `-- libcpufreq.la
|-- man
|    `-- man1
|         |-- cpufreq-info.1
|         `-- cpufreq-set.1
```

```
`-- share
    `-- locale
      |-- de
      |   `-- LC_MESSAGES
      |       `-- cpufrequtils.mo
      |-- fr
      |   `-- LC_MESSAGES
      |       `-- cpufrequtils.mo
      `-- it
          `-- LC_MESSAGES
              `-- cpufrequtils.mo
```

10.1.1.3 Complie cpufreqd-2.2.1

Entry cpufreqd-2.2.1 directory:

```
$ cd ../ cpufreqd-2.2.1
```

Define the three path variables in the configure-ingenic.sh configure script

```
cpufreqd_install_dir=  #It's the directory where you want to
install cpufreqd
cpufrequtils_destdir=  #It's the same as the DESTDIR in Makefile of
cpufrequtils-002
sysfsuitls_install_dir=  #It's the directory where you installed
sysfsutils
```

Then run the script:

```
$ ./configure-ingenic.sh
```

The script generates Makefile and config.h, need to modify the path of generated config.h, these paths are the installation path on the target board (refer on the following the path established):

```
/* Define this to configuration dir location */
#define CPUFREQD_CONFDIR "/etc/"

/* Define this to plugins dir location */
#define CPUFREQD_LIBDIR "/usr/lib/"

/* Define this to local state dir location */
```

```
#define CPUFREQD_STATEDIR "/var/"
```

And then start to compile and install:

```
$ make && make install
```

Under the installation path <cpufreqd_install_dir> will generate the following documents:

```
|-- bin
|     |-- cpufreqd-get
|     `-- cpufreqd-set
|-- etc
|     `-- cpufreqd.conf
|-- lib
|     |-- cpufreqd_cpu.la
|     |-- cpufreqd_cpu.so
|     |-- cpufreqd_governor_parameters.la
|     |-- cpufreqd_governor_parameters.so
|     |-- cpufreqd_programs.la
|     `-- cpufreqd_programs.so
|-- sbin
|     `-- cpufreqd
`-- share
      `-- man
            |-- man1
            |     |-- cpufreqd-get.1
            |     `-- cpufreqd-set.1
            |-- man5
            |     `-- cpufreqd.conf.5
            `-- man8
                  `-- cpufreqd.8
```

10.1.2 Install the cpufreqd on the target board

Only need to install sysfsutils-2.1.0 and cpufreqd-2.2.1. cpufrequtils-002 has connected to the cpufreqd-2.2.1 statically in the cross-compiling phrase.

a. Install sysfsutils-2.1.0.

Copy <sysfsuitls_install_dir>/lib/libsysfs.so, libsysfs.so.2 and libsysfs.so.2.0.1 to /lib/or/usr/. If you want to copy to the other special <dir>, then implement the command befor running cpufreqd required:

```
# export LD_LIBRARY_PATH=<dir>:$LD_LIBRARY_PATH
```

b. Install cpufreqd-2.2.1.
Copy the host's <cpufreqd_install_dir>/lib/cpufreqd_cpu.so, cpufreqd_governor_parameters.so, cpufreqd_programs.so to CPUFREQD_LIBDIR (defined in config.h) path on the target board;
Copy the host's <cpufreqd_install_dir>/sbin/cpufreqd to /sbin/or/usr/sbin/ on the target board;
Copy the host's <cpufreqd_install_dir>/bin/cpufreqd-get and cpufreqd-set to /bin/or/usr/bin/ on the target board;
Copy the host's <cpufreqd_install_dir>/etc/cpufreqd.conf to CPUFREQD_CONFDIR (defined in config.h) path on the target board.

10.1.3 Run cpufreqd on the target board

First, modify the configuration file cpufreqd.conf according to system's need, this profile determines the cpufreqd behavior. Its complete description can be found in <cpufreqd_install_dir>/share/man/man5/cpufreqd.conf.5.
cpufreqd.conf divided into three parts: General, Profile, and Rule.
General part of the example：

```
[General]
pidfile = /var/cpufreqd.pid
poll_interval = 2
suspend_interval = 10
verbosity = 0
# enable_remote = 1
[/General]
```

PID of coufreqd is stored in the Pidfile, its path is <CPUFREQD_STATEDIR>/cpufreqd.pid, CPUFREQD_STATEDIR was defined in the cross-compile-time by the cpufreqd-2.2.1/config.h of host(see 7.1.1.3), CPUFREQD_STATEDIR = "/var/" in the example. poll_interval (units of seconds, you can set to a decimal) is time interval in the read system status, and read status each time, will base on the current state of each Rule to score for the Rule, the rule who gets the high score will implement the corresponding Profile.
Parts of the sample of Profile(which can have more than one Profile):

[Profile]
name = Performance High
minfreq = 100%
maxfreq = 100%
policy = performance
[/ Profile]

[Profile]
name = Powersave Low
minfreq = 42000
maxfreq = 42000
policy = performance
[/ Profile]

minfreq and maxfreq define the allowed frequency range, you can use the absolute frequency with the default unit KHz, for example minfreq = 42000 indicated that the minimum frequency is 42MHz; Also can use the maximum frequency allowed by kernel(decided by u-boot's start frequency) is expressed as a percentage, if the u-boot startup frequency is 336MHz, then the maxfreq = 100% indicates the highest frequency is 336MHz. Kernel determines the frequency table of this system will only run on a specific set of frequencies. There are two ways to see the frequency table. The first is to look at the initialization function jz47xx_cpufreq_driver_init() of linux kernel code arch/mips/jz47xx/cpufreq.c; The second is to compile the kernel under the premise of choice debug cpufreq, after in the u-boot of the linux boot parameters bootargs with loglevel = 8 cpufreq.debug = 1,see the frequency table in the kernel boot process.

The parts of the sample of Rule (which can have more than one Rule):

[Rule]
name = CPU not busy
cpu_interval = 0-15
profile = Powersave Low
[/ Rule]

[Rule]
name = Movie Watcher
programs = mplayer, madplay
cpu_interval = 0-100
profile = Performance High
[/ Rule]

The name is the Rule's name; profile is the name of Profile which will be implemented with the highest scores. There are two conditions of Rule One is the scope of cpu occupancy, cpu_interval, the second is the current running programs. For example cpu_interval = 0-15, said cpu occupancy rate is 0% ~ 15%, programs = mplayer, madplay procedures that are currently running mplayer or madplay.
Set up cpufreqd.conf, run cpufreqd and then you can start cpufreqd at the background (ensure that the file system has been mount):

```
# cpufreqd
```

If you want to manually select profile of cpufreqd implementation, you should set enable_remote = 1 in the General column of cpufreqd.conf. and add -m as a option, that is

```
# cpufreqd –m
```

To implement the n profile via command cpufreqd–set n

If you want to see the information of operation of cpufreqd, you can add -V option, a total of 0 to 7 eight message level. To print out the score of any Rule, you can do:

```
# cpufreqd –V 6
```

Add-D option allows cpufreqd to run in the foreground and print out the operating information. Added --help option to see help information:

```
# cpufreqd –help
```

10.2 Sleep and wake-up management

The system is idle a long time, you can let it into the sleep mode. In this mode, the most of the modules of system are placed in low-power mode. JZ47xx sleep and wake-up management-related code is located in the code tree arch/mips/jz47xx directory pm.c.

The use of sleep and wake-up management need to configure linux kernel compile option, select Legacy Power Management API (DEPRECATED) (PM_LEGACY) in the Power Management support options .

Users can execute the following command to enter sleep mode:

```
#  echo 1 > /proc/sys/pm/suspend
```

10.3 Switch Management

Switch Management’s implementation mechanisms are: the short press power key will make system entry sleep mode, then wake up the system to press power key again; long press power key, system will shutdown, and then restart when press the power key. JZ47xx sleep and wake-up management-related code is located in the drivers/char/jzchar/poweroff.c.

The use of switch machine management needs to select Device Drivers-> Character devices-> JZSOC char device support-> JZ board poweroff support.

11 Wireless Device Configuration

Ingenic processor JZ47xx provides rich peripheral interfaces that can support multiple interfaces for wireless devices. The supported WIFI devices are currenly:

Interface	Device	OS	Max Throughput	Productor
USB	VT6656	celinux040503 linux-2.6.24.3	3Mbit/s	VIA
USB	ZD1211	celinux040503 linux-2.6.24.3	5.5Mbit/s	Atheros
SDIO	AR6001X	linux-2.6.24.3	16Mbit/s	Atheros
SDIO	88W8686	linux-2.6.24.3	16Mbit/s	Marvell
SPI	88W8686	linux-2.6.24.3	10Mbit/s	Marvell

11.1 Kernel Configuration

To support the wireless network equipment, linux kernel must select the "Wirelesss Extensions".

In the Linux configuration menu, find the "Networking" -> "wirelesss" -> "Wirelesss Extensions", select "Wirelesss Extensions".

Version of a wireless network interface of Linux-2.6.24.3 (WIRELESS_EXT) is 22.

SDIO interface wifi driver and the MMC / SD card driver are not compatible. If you want to use the wifi card of the SDIO interface ,in the kernel configuration options you need to remove MMC / SD driver .

11.2 Wlan driver module loaded

Ingenic platform currently supports a wireless network driver as a module loaded. Ingenic provids compiled modules, does not provide source code.
Users can get source code from the wifi vendors, as well as getting Ingenic platform’s patch, recompile to use.

11.2.1 Load VT6656 Driver:

```
# /sbin/insmod vntwusb.ko
```

11.2.2 Load ZD1211 driver:

```
# /sbin/insmod zd1211b.ko
```

11.2.3 Load GSPI8686 Drive:

GSPI8686 driver divided two main layers: IO layer (gspi.ko), wlan control level (gspi8686.ko).

```
# insmod gspi.ko clkdiv=4 spi_irq_pin=127 chipselect=0
# insmod gspi8686.ko helper_name=/lib/FwImage/helper_gspi.bin
fw_name=/lib/FwImage/gspi8686.bin
```

==========

gspi.ko: module parameters:

clkdiv: SPI clock divider factor. Range [0,15].
spi clock = (pllout/(clkdiv +1))/4
If the system pll clock is 360M Hz, clkdiv = 5, then
spi clock = (360/(5 +1))/4MHz =15 MHz

spi_irq_pin: interrupt signal line. GSPI8686 with the host communication needs this interrupt signal.
PAVO development board use the GPIO127 as the interrupt signal, spi_irq_pin = 127.
Users can allocate GPIO as an interrupt signal, if GPIO100 is allocated as the interrupt signal, set spi_irq_pin = 100.

chipselect: SPI device number. JZ4730, JZ4740 processor' SPI controller support 2 slave devices, CE0 and CE2.
PAVO development board uses CE0 and GSPI8686 connection, set the chipselect = 0.
Users can choose CE0 or CE2 connection GSPI8686 according to the actual situation. If you choose CE2, then set chipselect = 2.

==========

11.2.4 Load SD8686 driver:

SD8686 driver has 2 main layers: IO layer (sdio.ko), wlan control level (sd8686.ko).

```
# insmod sdio.ko gpio_sd_vcc_en_n = 113 gpio_sd_cd_n = 110
# insmod sd8686.ko helper_name =. / FwImage / helper_sd.bin fw_name
=. / FwImage/sd8686.bin
```

========== ==========

sdio.ko module parameters:

gpio_sd_vcc_en_n: MMC / SD power enbale pin, Pavo board use GPIO 113 (default)

gpio_sd_cd_n: MMC / SD card detect pin, Pavo board use GPIO 110 (default)

In your own board, if you use GPIO 100 as MMC / SD power enable pin,
GPIO 101 as MMC / SD card detect pin, set gpio_sd_vcc_en_n = 100 gpio_sd_cd_n = 101

========== ==========

11.2.5 Load AR6000 driver：

SD8686 driver has four main levels: IO layer (sdio_jz_hcd.ko), SDIO bus layer (sdio_busdriver.ko), SDIO protocol layer (sdio_lib.ko), wlan protocol layer (ar6000.ko).

```
# /sbin/insmod sdio_lib.ko
# /sbin/insmod sdio_busdriver.ko
# /sbin/insmod sdio_jz_hcd.ko builtin_card=0 debuglevel=3
gpio_sd_vcc_en_n=113 gpio_sd_cd_n=110
# /sbin/insmod ar6000.ko
```

==========

sdio_jz_hcd.ko: module parameters:

builtin_card: 0 (default), card is not built in the board.

1, card is built in the board.

gpio_sd_vcc_en_n: MMC / SD power enbale pin, Pavo board use GPIO 113 (default)

gpio_sd_cd_n: MMC / SD card detect pin, Pavo board use GPIO 110 (default)

In your own board, if you use GPIO 100 as MMC / SD power enable pin,

GPIO 101 as MMC / SD card detect pin, set gpio_sd_vcc_en_n = 100 gpio_sd_cd_n = 101

==========

11.3 Wireless-tools, wireless network configuration tool

Wireless tools are a set of wireless network configuration tool. Wireless tools are an open source project at:

http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

Here are commonly used commands:

iwconfig : query or configure the wireless network card parameters

iwconfig : query parameters of the wireless network card which not be shown

iwpriv : query or configure the specific parameter of network card

See wireless card eth1 information:

```
# iwconfig eth1
eth1 MRVL-GSPI8686 ESSID: "Ingenic1"
          Mode: Managed Frequency: 2.437 GHz Access Point: 00:15: E9:
DF: BB: 45
          Bit Rate: 54 Mb / s Tx-Power = 13 dBm
          Retry limit: 8 RTS thr = 2347 B Fragment thr = 2346 B
          Encryption key :****-****-** Security mode: open
```

```
          Power Management: off
          Link Quality: 0 / 10 Signal level: -33 dBm Noise level: -89
dBm
          Rx invalid nwid: 0 Rx invalid crypt: 0 Rx invalid frag: 41093
          Tx excessive retries: 0 Invalid misc: 0 Missed beacon: 0
```

iwlist Parameters:

```
# iwlist
Usage: iwlist [interface] scanning [essid NNN] [last]
               [interface] frequency
               [interface] channel
               [interface] bitrate
               [interface] rate
               [interface] encryption
               [interface] keys
               [interface] power
               [interface] txpower
               [interface] retry
               [interface] ap
               [interface] accesspoints
               [interface] peers
               [interface] event
               [interface] auth
               [interface] wpakeys
               [interface] genie
               [interface] modulation
```

Search around the AP:

```
# iwlist eth1 scan
eth1 Scan completed:
          Cell 01 - Address: 00:02:6 F: 05: BA: CC
                    ESSID: "Ingenic"
                    Mode: Managed
                    Frequency: 2.412 GHz (Channel 1)
                    Quality: 0 / 10 Signal level =- 45 dBm Noise level
=- 96 dBm
                    Encryption key: on
                    Bit Rates: 11 Mb / s
          Cell 02 - Address: 00:14:6 C: CF: B6: 8C
                      ESSID: "Ingenic-test"
                      Mode: Managed
```

```
                    Frequency: 2.437 GHz (Channel 6)
                    Quality: 0 / 10 Signal level =- 76 dBm Noise level
=- 96 dBm
                    Encryption key: on
Bit Rates: 54 Mb / s
```

11.4 Steps to configure a wireless network

After successfully loaded wireless network card driver, configure the wireless network, follow these steps：

```
# ifconfig eth1 192.168.1.111    // set the local IP (the same
                                 network segment with the AP)
# iwconfig eth1 mode managed       // set a wireless network model
# iwconfig eth1 key 1234567890 key on    // set the network password
# iwconfig eth1 essid AP-name            // set the AP Name
```

Now configuration is completed, you can connect ap:

```
# ping 192.168.1.1 / / test the connection with the AP
```

11.5 Iperf network test tools：

Iperf is a small network test tool that can test network throughput between two hosts.

Server-side：

```
# iperf-s-t 10
------------------------------------------------ ----------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------ ----------
[4] local 192.168.1.20 port 5001 connected with 192.168.1.123 port 38571
[4] 0.0-10.0 sec 2.15 MBytes 1.81 Mbits / sec
```

Client：

```
# iperf-i 10-t 100-c 192.168.1.52-d
------------------------------------------------ ----------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
```

```
------------------------------------------------------------ ----------
------------------------------------------------------------ ----------
Client connecting to 192.168.1.52, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------ ----------
[5] local 192.168.1.20 port 40410 connected with 192.168.1.52 port 5001
[4] local 192.168.1.20 port 5001 connected with 192.168.1.52 port 54116
[4] 0.0-10.0 sec 5.25 MBytes 4.40 Mbits / sec
[5] 0.0-10.0 sec 3.73 MBytes 3.13 Mbits / sec
[5] 10.0-20.0 sec 3.52 MBytes 2.96 Mbits / sec
[4] 10.0-20.0 sec 5.23 MBytes 4.38 Mbits / sec
[5] 20.0-30.0 sec 3.22 MBytes 2.70 Mbits / sec
```