君正 **Linux** 开发指南

Revision: 0.1
Date: Apr 2011

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

君正 **Linux** 开发指南

Release history

Date	Revision	Change
2011.04.14	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.

北京君正集成电路有限公司

北京市海淀区东北旺西路 8 号中关村软件园一号楼
信息中心 A 座 108 室，100193
Tel: 86-10-82826661
Fax: 86-10-82825845
Http: //www.ingenic.cn

内容

1 概述

君正处理器是高集成度、高性能和低功耗的 32 位 RISC 处理器，带有 MMU 和数据及指令 Cache，以及丰富的外围设备，可以运行 Linux 操作系统。本文将向读者介绍基于君正处理器平台进行 Linux 内核和应用开发的过程和方法，引导开发人员快速进行 Linux 开发。包括建立交叉编译环境，引导程序 U-Boot，Linux 2.6 内核和驱动，Linux 电源管理，Linux 无线网络驱动和配置，Linux GUI 移植和应用开发等。

为了构建基于君正处理器的 Linux 2.6 的开发平台，您需要准备以下资源：

1） Linux 开发主机，用来安装交叉编译器和相关源码；
2） 基于君正处理器的开发板，包括串口线、电源线、以及用于烧录的 USB Device 线；
3） USB Boot 1.4b 烧录工具；
4） MIPS 交叉编译工具链：君正提供基于 GCC-4.1.2 和 GLIBC-2.6.1 的工具链；
5） 引导程序 U-Boot 源码：君正提供 u-boot-1.1.6 的源码和补丁；
6） Linux 2.6 核心源码：君正提供 linux 2.6.31.3 核心的源码和补丁；
7） 根文件系统：君正提供参考的根文件系统，基于 glibc 2.6.1 动态库，支持 udev 和 hotplug 等；
8） GUI 开发环境：用户自己选择，如 GTK/QTE/MiniGUI 等。

2 建立开发环境

在开始 Linux 开发之前，您需要准备一台 PC 来安装交叉编译器，放置源代码和根文件系统，启动 TFTP 和 NFS 服务等。我们使用的测试主机安装了 Ubuntu 7.10。

2.1 安装交叉编译工具链

在君正处理器平台上进行 Linux 2.6 内核开发之前，首先需要安装好 MIPS 的交叉编译工具连。针对 Linux 2.6 内核开发，君正提供基于 GNU GCC 4.1.2 和 GLIBC-2. 6.1 的 MIPS 交叉编译工具链，需要安装在 Linux 主机上。

在 Linux 主机上安装 MIPS 交叉编译工具的步骤如下：

首先，创建一工作目录，并把工具链包解压到该目录下，比如：

```
$ mkdir –p /opt
$ cd /opt
$ tar xzf mipseltools-gcc412-glibc261.tar.gz
```

其次，更新 PATH 环境变量：

```
$ export PATH=/opt/mipseltools-gcc412-glibc261/bin:$PATH
```

按照上面步骤建立好交叉编译环境后，可以编译简单的 helloworld.c 测试一下：

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

如果编译通过，说明刚刚安装的交叉编译工具链可以工作了。

2.2 启动 TFTP 和 NFS 服务

TFTP 是用来下载远程文件的网络传输协议，引导程序 U-Boot 支持 TFTP 下载功能。在开发阶段，如果主机启动了 TFTP 服务，引导程序 U-Boot 就可以通过 TFTP 下载 Linux 核心到目标板运行，这样将极大的方便用户进行开发。 因此，建议用户在用来开发的 Linux 或者 Windows 主机上启动 TFTP 服务。Linux 服务器上启动 TFTP 服务的步骤如下：

修改文件/etc/xinetd.d/tftp，主要是设置 TFTP 服务器的根目录和开启服务。修改后的文件如下：

```
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
}
```

以上指定 TFTP 服务的根目录为/tftpboot。如果该目录没有，创建它并启动 TFTP 服务，方法如下：

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

这样，TFTP 服务就启动了，你可以使用 TFTP 客户端程序进行测试。

网络文件系统（NFS）能够在不同的系统间实现文件的共享。在启动 Linux 内核后，内核可以通过 NFS 挂载根文件系统，以方便应用开发和调试。在开发阶段，您需要配置 Linux 开发主机启动 NFS 服务，并安装 ROOT 文件系统到 NFS 目录下。Linux 服务器上启动 NFS 服务的步骤如下：

创建和编辑文件/etc/exports 为：

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

设定好后可以使用以下命令启动NFS:

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

到这里，基本的开发环境已经准备好了，下面就可以开始具体的开发工作了。

3 U-Boot 的开发和使用

Linux 内核需要 U-Boot 来引导。U-Boot 是为嵌入式平台提供的开放源代码的引导程序，它提供串行口、以太网等多种下载方式，提供 NOR 和 NAND 闪存和环境变量管理等功能，支持网络协议栈、JFFS2/EXT2/FAT 文件系统，同时还支持多种设备驱动如 MMC/SD 卡、USB 设备、LCD 驱动等。

君正移植了 U-Boot 1.1.6 版本，这里将向读者介绍 U-Boot 的移植、开发和使用方法。

注意：此处只是举一些配置的例子，与实际情况可能有所差异。具体使用时，要根据实际的硬件进行配置。参考文档 linux_nand_flash_guide_CN.pdf

3.1 U-Boot 源码

首先，您需要从君正公司主页(http://www.ingenic.cn/)下载U-Boot的源码包u-boot.1.1.6.tar.bz2 和最新补丁（u-boot-1.1.6-jz-yyyymmdd.patch.gz，yyyymmdd表示日期）。

接下来，把 U-Boot 源码包解压到工作目录下，如果有最新补丁，把补丁打到其目录，举例如下：

```
# 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
```

到这里，您已经准备好 U-Boot 的源码。

u-boot-1.1.6 目录结构如下：

- cpu：CPU 相关文件，其中的子目录都是以 U-Boot 支持的 CPU 命名的。君正 CPU 相关的代码都位于 cpu/mips/目录下，主要文件包括：

  - start.S MIPS 内核启动代码
  - cpu.c CPU 其它相关代码，如 TLB 和 CACHE 操作等
  - jz4750.c JZ4750 相关代码，如系统 timer、PLL 和 SDRAM 的初始化等
  - jz_serial.c 串行口 UART 驱动程序
  - jz_eth.c 以太网底层驱动程序
  - jz_i2c.c I2C 接口驱动程序
  - jz_lcd.c LCD 控制器驱动程序
  - jz4750_lcd.c jz4750 LCD 控制器驱动程序
  - jz_mmc.c MMC/SD 卡驱动程序
  - jz4750_nand.c JZ4750 NAND flash 驱动

  注：jz4750d 的代码与 jz4750 是共用的，通过宏 CONFIG_JZ4750D 控制。

- board：开发板相关文件，包括代码的链接脚本文件 u-boot.lds 和地址分配文件 config.mk、以及开发板的初始化代码等。基于 JZ4750 的平台有 APUS 等，主要文件包括：

  - flash.c 开发板 NOR Flash 驱动程序
  - Makefile makefile 文件
  - config.mk 地址分配文件
  - u-boot.lds U-Boot 链接脚本文件
  - u-boot-nand.lds NAND U-Boot 链接脚本文件

- common：与体系结构无关的文件，包含各种 U-Boot 通用命令的文件
- disk：disk 驱动的分区处理代码
- doc：相关文档
- drivers：通用设备驱动程序，如各种网卡驱动、CFI 标准 Flash 驱动、USB Device 驱动等。
- fs：各种文件系统的驱动，如 EXT2、FAT、JFFS2、CRAMFS 等
- include：各种头文件，包含体系相关的定义和开发板的配置文件等。
  - include/asm-mips/jz4750.h JZ4750 相关的头文件定义。
  - include/asm-mips/jz4750d.h JZ4755 相关的头文件定义。
  - include/configs/apus.h 基于 JZ4750 的开发平台 APUS 的配置文件。
  - include/configs/cetus.h 基于 JZ4755 的开发平台 CETUS 的配置文件。

- lib_generic：所有体系通用的文件
- lib_mips：MIPS 体系通用的文件
- lib_arm：ARM 体系通用的文件
- nand_spl：NAND SPL (Secondary Program Loader)代码
- msc_spl：MMC/SD SPL (Secondary Program Loader)代码
- net：网络相关的代码
- tools：创建 S-Record 和 U-Boot 映像的工具，如 mkimage

3.2 配置和编译 U-Boot

配置和编译 U-Boot 的过程很简单。不过，在开始之前，您需要确定开发板的名称，并确定 CPU 是从 NAND Flash 还是从 NOR Flash 启动。现在发布的版本支持君正多个系列的 CPU 和开发平台，每个平台都有对应的配置文件。具体参考以下例子：

1) JZ4750 APUS 平台，Nor Flash 启动：
编译 Uboot 之前，先把 apus.h 中的“#define CONFIG_LOAD_UBOOT”屏蔽掉，然后

```
$ make apus_spi_config
$ make
```

最后，把生成的 u-boot-spi.bin 烧录到 spi nor 的 0 地址，把 zImage 烧写到 spi 的 256KB 处（这是由

apus.h 中的#define CFG_SPI_ZIMAGE_OFFS (256<<10) 决定的）。

2) JZ4750 APUS 平台，NAND Flash 启动：
对于 JZ4750, 在 NAND 的 block 0 的前 16KB 放置了 nand_spl，从 CFG_NAND_U_BOOT_OFFS 开始放置 u-boot.bin。u-boot-nand.bin 由 nand_spl 和 u-boot.bin 组合而成。 JZ4750 支持各种页大小的 NAND(512B, 2KB 和 4KB)，bootrom 通过读取 nand_spl 的前 12 个 byte,来获取 NAND 的总线宽度、页大小和行地址长度。
编译之前需要在 include/configs/apus.h 中，设置好相关 NAND 参数， 例如对于型号 K9G8G08U0M 的 NAND：

```
#define CFG_NAND_BW8                1
#define CFG_NAND_PAGE_SIZE          4096
#define CFG_NAND_ROW_CYCLE          3
#define CFG_NAND_BLOCK_SIZE         (512 << 10)
#define CFG_NAND_BADBLOCK_PAGE      127
#define CFG_NAND_BCH_BIT            8
#define CFG_NAND_ECC_POS            24
//#define CFG_NAND_BCH_WITH_OOB
```

其中， CFG_NAND_BCH_BIT 表示使用哪种 ECC 算法 ，有 4bit BCH 和 8bit BCH 两种选择。CFG_NAND_BCH_BIT 和 CFG_NAND_ECC_POS 的设置需要与烧录器的相应设置一致。

CFG_NAND_BCH_WITH_OOB 表示 u-boot 对 NAND 的 BCH ECC 编解码方法是否和 linux kernel 一致。如果未定义 CFG_NAND_BCH_WITH_OOB，那么 u-boot 只对 NAND data 区域进行编解码，和 linux 不一致。如果定义了 CFG_NAND_BCH_WITH_OOB，u-boot 将对 NAND data 区域和 oob 区域编解码，和 linux 一致，此时 u-boot 可以读取在 linux 下用 nandwrite_mlc 命令烧录的 uImage，但被烧录的 uImage 要用 mkyaffs2image 做特殊的处理，具体说明请参见君正网站上的手册 Linux NAND Flash Guide 的 1.3.3.3 “JZ4750 在 linux 下烧录 uImage”。

然后，

```
$ make apus_nand_config
$ make
```

编译后生成 u-boot-nand.bin。

3) JZ4750 APUS 平台，SD Card 启动：
如果想在 u-boot 中使用 NAND 相关的命令，需要在 include/configs/apus.h 中，设置好一个 NAND 参数，比如：

```
#define CFG_NAND_IS_SHARE           1
```

CFG_NAND_IS_SHARE 设为 1，表示 NAND 和 SDRAM 共用总线。对于支持 1.8V mobile SDRAM 的 JZ4750，两者是不共用总线的，须将其设为 0。在 nand boot 的情况下，不必关心该参数的值，因为 bootrom 在读取 nand_spl 时会自动判断总线是否共享。

然后，

```
$ make apus_msc_config
$ make
```

编译后生成 u-boot-msc.bin。

4) JZ4755 CETUS 平台，NAND Flash 启动：
对于 JZ4755（JZ4750d），编译之前需要在 include/configs/cetus.h 中，设置好相关 NAND 参数， 例如对于型号 K9GAG08U0M 的 NAND：

```
   #define CFG_NAND_BW8                    1
   #define CFG_NAND_PAGE_SIZE        4096
   #define CFG_NAND_ROW_CYCLE         3
   #define CFG_NAND_BLOCK_SIZE  (512 << 10)
   #define CFG_NAND_BADBLOCK_PAGE     0
   #define CFG_NAND_BCH_BIT            8
#define CFG_NAND_ECC_POS           24
```

然后，确认你的 SDRAM 的大小，如果是 64M，则 include/configs/cetus.h 中设置

```
#define CONFIG_NR_DRAM_BANKS       1
```

如果是 128M 则设置

```
#define CONFIG_NR_DRAM_BANKS       2
```

最后，

```
$ make cetus_nand_config
$ make
```

编译后生成 u-boot-nand.bin。

3.3 烧录 U-Boot 到目标板

以上编译好的二进制映像需要烧录到目标板的 Flash 上，CPU 在上电后会从 Flash 读取指令并启动 u-boot。可以通过 USB Boot 工具烧录，详细的烧录方法请参考这个工具的使用文档。这里我们举个例子说明一下：

USB Boot 工具烧录 u-boot-nand.bin

USB Boot 工具仅适用于支持 USB 引导的君正处理器平台。

首先要安装好 USB Boot 工具的主机驱动和应用。连接好 USB 电缆，启动目标板从 USB 引导。在确认驱动正确加载和识别目标板后，运行 USB Boot 应用 USB_Boot.exe 并开始烧录。

在烧录前要确认 USB Boot 配置文件的参数设置是否正确，在确认正确后就可以开始烧录了：

```
   USBBoot :> boot 0
```

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

烧录完成后，启动目标板，这时您应该可以在串口终端看到 U-Boot 运行的打印信息。

3.4 U-Boot 命令

下面简要介绍一些常用的 U-Boot 命令：

“help”命令：该命令查看所有命令，其中“help command”查看具体命令的格式。
“printenv”命令：该命令查看环境变量。
“setenv”命令：该命令设置环境变量。
“askenv”命令：该命令设置环境变量。
“saveenv”命令：该命令保存环境变量。
“bootp”命令：该命令动态获取 IP。
“tftpboot”命令：该命令通过 TFTP 协议从网络下载文件运行。
“nfs”命令：该命令通过 NFS 协议从网络下载文件运行。
“bootm”命令：该命令从 memory 运行 u-boot 映像。
“go”命令：该命令从 memory 运行应用程序。
“boot”命令：该命令运行 bootcmd 环境变量指定的命令。
“reset”命令：该命令复位 CPU。
“md”命令：显示内存数据。
“mw”命令：修改内存数据。
“cp”命令：内存拷贝命令。

NOR Flash 命令：

“protect”命令：NOR Flash 写保护使能/禁止命令。
“erase”命令：NOR Flash 擦除命令。

NAND Flash 命令：

“nand info”：查看 NAND 信息。
“nand erase”：NAND Flash 擦除命令。
“nand read”：NAND Flash 读命令。
“nand write”：NAND Flash 写命令。
“nand bad”：NAND Flash 坏块信息。

MMC/SD 卡命令：

“mmcinit”：MMC/SD 初始化命令。
“fatls mmc 0 /”：查看 MMC/SD 目录和文件命令。
“fatload mmc 0 address file”：读 MMC/SD 卡文件命令。

下面简要介绍一些常用的 U-Boot 环境变量：

“ipaddr”：开发板 IP 地址。
“serverip”：服务器 IP 地址。
“bootfile”：u-boot 启动文件。
“bootcmd”：u-boot 启动命令。
“bootdelay”：u-boot 启动延时。
“bootargs”：u-boot 启动参数（即 linux 命令行参数）
“ethaddr”：网络 MAC 地址。

3.5 U-Boot 使用举例

给目标板上电并启动后，在串行口终端可以看到 U-Boot 的输出信息，这时按任意键跳过自动启动过程进入到 U-Boot 命令界面。在 U-Boot 命令界面里运行“help”命令，可以看到 U-Boot 支持的所有命令，运行“help command”查看具体命令的格式和使用方法，运行“printenv”命令察看当前的所有环境变量，比如：

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

这时，您可以配置 U-Boot 从网络通过 TFTP 下载 Linux 核心运行，并挂载 NFS 网络文件系统。请参考下面的步骤来进行配置：（这里假设您的服务器 IP 地址为 192.168.1.4，并且服务器已经启动了 TFTP 和 NFS 服务；Linux 核心位于 TFTP 服务的目录/tftpboot 下，文件名为 uImage；网络文件系统路径为/nfsroot/root26；apus 开发板的 IP 为 192.168.2.84，MAC 地址任意给定，这里为 00:2a:c6:2c:ab:f1）

```
APUS # setenv ipaddr 192.168.2.84
APUS # setenv serverip 192.168.1.4
APUS # setenv ethaddr 00:2a: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”命令将保存当前配置到 Flash 上，下次重起系统时就不需要再配置了。“boot”命令将启动运行环境变量“bootcmd”定义的命令。这里，U-Boot 首先通过 TFTP 下载 Linux 核心文件 uImage 到目标板的 SDRAM，然后运行“bootm”命令启动 Linux 核心。Linux 启动后将通过 NFS 挂载 192.168.1.4 服务器上的/nfsroot/root26 为根文件系统。

3.6 U-Boot 映像类型

U-Boot 的“boot”命令支持引导特定类型的映像文件，这些文件都包含 U-Boot 所支持的文件头信息。这些文件头通常定义了映像文件的操作系统类型（如 Linux、VxWorks、Solaries 等），CPU 的体系结构（如 ARM、MIPS、X86）、压缩类型（gzip、bzip2、非压缩）、下载地址、程序入口地址、映像名称和时间戳等。

通常编译的各种格式的 Linux 核心如 vmlinux、zImage 等在 U-Boot 上并不能使用。这时需要使用 U-Boot 提供的工具 mkimage 生成 U-Boot 支持的 Linux 核心 uImage，其命令格式如下所示：

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

其中：

```
-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'
```

使用 mkimage 生成 uImage 的步骤如下：

- 编译生成一个标准的“vmlinux”内核文件（ELF 的二进制格式）
- 把“vmlinux”转换为原始的二进制文件：

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

- 压缩二进制文件：

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

- 用 mkimage 打包为 U-Boot 的格式文件：

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

生成 uImage 后，使用“mkimage –l image”命令察看映像文件包含的头文件信息，如下所示：

```
$ 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 内核和驱动

君正移植了 Linux 2.6.31.3 内核和设备驱动程序，可以直接运行在君正处理器的各种开发平台上。本文将具体描述君正 Linux 2.6.31.3 内核和驱动程序。

君正 Linux 核心支持基于 JZ4750(d)的各种开发平台。其中基于 Jz4720 的开发平台有 virgo，基于 Jz4750 的开发平台有 apus，基于 Jz4750d(Jz4755)的开发平台有 cetus。

4.1 Linux 源码的目录结构

Linux 2.6 内核的源码和补丁可以直接从君正公司主页上（http://www.ingenic.cn/）下载。

首先，在一工作目录下解开 Linux 2.6.31.3 的源码：

```
$ tar xjf linux-2.6.31.3.tar.bz2
```

然后，把 linux 2.6.31.3 的补丁打到源代码目录下：

```
$ cd linux-2.6.31.3
$ gzip -cd ../linux-2.6.31.3-jz-yyyymmdd.patch.gz | patch –p1
```

请使用君正最新发布的内核补丁。

Linux-2.6.31.3 内核源代码的目录结构如下：

- arch/mips/：MIPS 体系相关目录和文件
  - kernel/：MIPS 内核相关文件
  - mm/：MIPS 内存管理相关文件
  - lib/：MIPS 公用库函数
  - jz4750/：JZ4750 处理器相关目录和文件
    - *.c： JZ4750 处理器通用文件
    - board-apus：APUS 开发板相关文件
  - jz4750d/：JZ4750d 处理器相关目录和文件
    - *.c： JZ4750d 处理器通用文件
    - board-cetus：CETUS 开发板相关文件
  - ramdisk/：initrd 相关文件
  - boot/：uImage 生成目录
  - Kconfig：MIPS 体系配置文件
  - Makefile：MIPS 通用 makefile
  - configs/：平台缺省配置文件

        - apus_defconfig：JZ4750-APUS 开发板缺省配置
        - cetus_defconfig：JZ4750D-CETUS 开发板缺省配置
- include/asm-mips/：MIPS 体系相关头文件
    - jzsoc.h：JZSOC 通用头文件
    - mach-jz4750/：JZ4750 处理器相关头文件
    - mach-jz4750d/：JZ4750D 处理器相关头文件
- sound
    - oss/：OSS 音频驱动
        - jz_i2s.c：I2S 通用驱动
        - jzcodec.c：JZ CODEC 驱动
        - ak4642en.c：AK4642EN CODEC 驱动
        - jz_ac97.c：AC97 通用驱动
    - soc/jz4745/：JZ4750 ALSA 驱动
        - jz4750-i2s.c
        - jz4750-ac97.c
        - jz4750-pcm.c
    - soc/codecs/：ALSA CODEC 驱动
        - jzcodec.c
- kernel：Linux 通用内核文件
- mm/：Linux 通用内存管理文件
- lib/：Linux 通用库函数
- init/：Linux 初始化函数
- ipc/：Linux 进程间通信函数
- net/：网络相关文件
- fs/：文件系统相关文件
    - jffs2/： JFFS/JFFS2 文件系统
    - yaffs2/: YAFFS/YAFFS2 文件系统
        - utils/：mkyaffs2image 工具
    - ubifs/：UBIFS 文件系统
- drivers/：设备驱动目录
    - block/：块设备驱动
    - char/：字符设备驱动
    - char/jzchar/：JZSOC 字符设备驱动
    - cpufreq：cpufreq 驱动
    - input/：输入设备驱动
    - mmc/：MMC/SD 卡驱动
    - mtd/：MTD 设备驱动
        - ubi/：UBI 驱动
        - mtd-utils/：MTD 和 UBI 工具，如 flash_eraseall、nandwrite_mlc、ubimkvol 等
    - net/：网络设备驱动
    - serial/：UART 驱动
    - ssi/：同步串行接口驱动

- usb/：USB host 驱动
- usb/gadget： USB device 驱动
  - jz4750_udc.c
  - file_storage.c
- video/：LCD framebuffer 驱动
  - jzlcd.c
  - jz4750_lcd.c

4.2 配置和编译 Linux

首先，选择目标板的配置：

```
$ make apus_defconfig      (JZ4750 APUS 平台)
$ make cetus_defconfig     (JZ4755 CETUS 平台)
```

如果要修改配置，运行以下命令：

```
$ make menuconfig
```

最后编译 Linux 核心：

```
$ make uImage
```

命令‘make uImage’编译生成 U-Boot 可以引导的二进制映像 uImage，位于 linux-2.6.31.3/arch/mips/boot/目录下。
编译过程中需要用到命令 uudecode 和 mkimage。如果找不到 uudecode，请为 PC 安装软件包 sharutils-4.6.1-2，如果找不到 mkimage，请在 u-boot-1.1.6 编译后，在 u-boot-1.1.6/tools/ 获取。

4.3 由 U-Boot 启动 Linux 内核

为便于开发和调试，您可以配置 U-Boot 使其通过 TFTP 服务下载 Linux 内核到开发板 SDRAM 中，并从 SDRAM 启动 Linux 内核，同时配置 Linux 命令行参数使其通过 NFS 挂载根文件系统。

如果 Linux 能正常启动并挂载成功根文件系统，说明 Linux 内核运行正常。

Linux 常用命令行参数：

mem=xxM	设置内存容量大小
console=tty0	设置控制台输出为 tty0
console=ttyS0,57600n8	设置控制台输出为串口 ttyS0，波特率为 57600，8 个数据位，无校验位
ip=dhcp	IP 地址通过 DHCP 服务获取
ip=192.168.2.84	设置静态 IP 为 192.168.2.84

nfsroot=192.168.1.4:/nfsroot/root26 rw
设置网络根文件系统，具有读写权限
root=/dev/mtdblock2 rw 设置根文件系统为/dev/mtdblock2，具有读写权限
rootfstype=yaffs2 根文件系统类型为 YAFFS2
ethaddr=00:a1:22:b2:dc:38 设置以太网 MAC 地址

4.4 测试 Linux 内核和驱动

在启动 Linux 内核后，您可以通过一些方法和命令测试各驱动功能是否正常，具体如下：

- **测试 LCD 显示**

配置 Linux 时选择好目标板 LCD 屏的型号，在启动 Linux 时就应该能看到 LCD 屏幕有图案输出。如果没有，请检查内核驱动配置，并检查 LCD 屏的硬件连接是否正常。

- **测试音频播放和录音**

MP3 播放测试：

```
# madplay test.mp3
```

录音和播放录音测试：

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

- **测试视频播放**

测试经过优化的在 Jz4750 上运行的 mplayer:

```
# mplayer50 –vf scale=480:272 binhe.264
```

- **测试 JFFS2 和 YAFFS2 文件系统**

配置 Linux 时选择 MTD 设备和 JFFS2/YAFFS 文件系统，启动 Linux 后就可以测试基于 MTD 的文件系统 JFFS2 和 YAFFS2：

首先，查看 MTD 分区表信息：

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

接着，格式化 MTD 分区：

```
# flash_eraseall –j /dev/mtd3      JFFS2 文件系统格式化
# flash_eraseall /dev/mtd4     YAFFS2 文件系统格式化
```

然后，挂载和测试文件系统：

```
# mount –t jffs2 /dev/mtdblock3 /mnt/mtdblock3      JFFS2 分区挂载
# mount –t yaffs2 /dev/mtdblock3 /mnt/mtdblock4     YAFFS2 分区挂载
# cp test /mnt/mtdblock4
# umount /mnt/mtdblock4
```

如果正常，说明文件系统已经工作。

- **测试 MMC/SD 卡**

在 Linux 正常启动后，插入 MMC 或 SD 卡到 SD 卡座，Linux 就能自动探测到卡的插入并读出分区表信息，这时就可以挂载和操作 SD 卡了：

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

- **测试触摸屏**

启动 Linux 后，运行下面命令校正触摸屏：

```
# ts_calibrate
```

运行下面命令测试触摸屏：

```
# ts_test
```

- **测试 USB Host**

插入 USB 设备（如 U 盘）到 USB Host 端口，Linux 应能探测到设备的插入，这时可以挂载和操作 U 盘了：

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

- **测试 USB Device**

首先，以模块方式编译生成 USB Device 驱动：jz4740_udc.ko 和 g_file_storage.ko；然后按照下面方法加载驱动：

```
JZ4750 JZ4750D
# insmod jz4740_udc.ko
# insmod g_file_storage.ko file=/dev/mtdblock4,/dev/mmcblk0
```

这时我们挂载了两个 U 盘分区，一个是基于 NAND Flash 的分区 mtdblock4，另一个是基于 SD 卡的分区。现在用 USB 电缆连接 PC 和目标板的 USB Device 端口，PC 应该能识别到设备的插入并识别为 U 盘设备。

o **测试电源管理**

a. 测试系统睡眠、唤醒和关机

运行下面命令，系统将进入睡眠状态，在按 power 键之后唤醒。

JZ4750:

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

运行下面命令，系统将关机，在按 power 键之后重启。

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

b. 测试动态变频管理

在编译内核时，在 CPU Frequency scaling 选项中选择 userspace 为默认的 governor。这样在目标板启动之后可以通过 sys 文件系统接口来控制 cpufreq 驱动。在目标板启动之后，如果没有 mount sys 文件系统，那么需要执行

```
# mount –t sysfs sysfs /sys
```

如果编译内核时选择的默认 governor 不是 userspace，而是 performance，那么需要执行命令把 governor 改为 userspace：

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

然后通过下面命令可以看当前系统频率(单位:KHz)：

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

另外，更详细的系统频率可由下面命令获得，该命令与 cpufreq 无关：

```
# 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
```

看当前可设置的最低系统频率:

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

看当前可设置的最高系统频率:

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

设置当前系统频率(频率单位是 KHz，以设为 112MHz 为例):

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

在编译内核时，在 CPU Frequency scaling 选项中勾选 CPU frequency translation statistics details，可以使用三个命令看变频记录。首先，通过下面命令可获得变频的历史记录表:

```
# 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
```

通过下面命令便可获得系统在各个频率点的工作时间，单位时间和 jiffies 的单位相同，一般是 10ms（由编译内核时的 Kernel type 中的 Timer frequency 决定）。如下例中系统在 42000KHz 的工作时间是 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
```

下面命令可得到总的变频次数

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

4.5 Linux 2.6 音频驱动

Linux 2.6 的音频驱动包括 ALSA 和 OSS 两种架构，这里介绍它们的实现和用法。

4.5.1 OSS 音频驱动

OSS 音频驱动在 linux-2.6.31.3/sound/oss 目录下。目前提供了 ak4642en 和 jz4750 内部 codec 的驱动，相关代码是 jz4750_i2s.c、jz4750_dlv.c、jz_ac97.c、jz_i2s.c、ak4642en.c 和 jzcodec.c。

当使用 OSS audio 驱动时，在 make xconfig 弹出的窗口中选择 sound/Open Sound System/Obsolete OSS drivers/jz On-Chip I2S driver 中的一种 codec，选择之后，再执行 make uImage 编译内核。

4.5.2 ALSA 音频驱动

ALSA 音频驱动在 linux-2.6.31.3/sound/soc 目录下，目前提供了 jz4750 及其内部 codec 的驱动，相关代码在 soc 目录下的 jz4750 和 codecs 中。其中 jz4750 目录下包括对 I2S 的支持，codec 目录下包括对内部 codec 的支持。

当使用 ALSA 音频驱动时，在 make xconfig 弹出的窗口中选择 sound/Advanced Linux Sound Architecture/System on Chip audio support 下选择相关选项，也可选上对 OSS emulate 的支持的选项，再执行 make uImage 编译内核。

ALSA 应用程序执行时，需要 ALSA 库的支持，ALSA 库放在/usr/alsa/lib 目录；ALSA utils 的测试程序放在/usr/alsa/bin 目录。

ALSA 的配置文件放在/usr/alsa/share/alsa.conf，它的路境由/etc/profile 中的环境变量 ALSA_CONFIG_PATH 指定。

声卡的配置文件放在/usr/alsa/etc/asound.conf 中，指定了一些设备名、频率转换和混音 plugin；该文

件的路境由 ALSA 的配置文件决定。

ALSA 音频驱动测试方法如下：

设置放音增益（0 to 3）：

amixer set Master 1

查看放音增益：

amixer get Master

设置录音增益（0 to 31）：

amixer set Line 10

查看录音增益：

amixer get Line

测试录音：

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

-d : 20 秒
-c : 2 个声道
-t ： 文件类型
-r : 频率
-f : 16bits

测试放音：

aplay ./test.wav
或
aplay －D primary ./test.wav

参考 asound.conf 中指定的 primary

测试变频：

aplay -D rate_44k ./test.wav

参考 asound.conf 中指定的 rate_44k

测试混音：

aplay -D plug:dmix_44k ./test.wav &
再执行
aplay -D plug:dmix_44k ./test.wav

参考 asound.conf 中指定的 dmix_44k

5 Linux 根文件系统

Linux 内核编译好之后，还必须有根文件系统（root filesystem）才能使一个 Linux 系统正常运行。本节将简要介绍根文件系统的功能，以及如何根据自己系统的需求来制作根文件系统。您也可以到网站上下载我们制作发布的根文件系统，使用超级用户权限解压后可用作测试。

5.1 根文件系统的内容

根文件系统用于存放系统运行期间所需的应用程序、脚本、配置文件等，通常包含下面目录：

- bin/、sbin/：系统可执行程序和工具
- lib/：动态运行库
- etc/：系统配置信息和启动脚本 rcS
- usr/：用户可执行程序

5.2 编译 Busybox

嵌入式系统由于存储空间的限制，使得其对程序大小有严格的限制。特别是在制作根文件系统时，更要注意。而使用 BusyBox 就可以大大的简化根文件系统的制作。编译安装后的 BusyBox 只有一个二进制可执行文件 busybox，它实现了几乎所有常用、必须的应用程序（比如 init，shell，getty，ls，cp 等等），而这些应用程序都以符号链接的形式存在。对用户来说，执行命令的方法并没有改变，命令行调用会作为一个参数传给 busybox，即可完成相应的功能。使用 BusyBox 制作的根文件系统既可以节省大量的空间，还节省了大量的交叉编译的工作。

请登陆 BusyBox 的官方网站（http://www.busybox.net）下载起源码包。下面以 busybox-1.8.2 版本为例说明编译和安装 BusyBox 的过程。

请在编译之前安装好 mipsel-linux-gcc 编译器工具（gcc-4.1.2 和 glibc-2.3.6）。编译 BusyBox 的过程类似于编译 Linux 内核的过程，如下：

```
# 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 编译和配置 udev

Linux 2.6 下设备文件/dev/的创建通过 udev 来实现。udev 通过 sysfs 文件系统提供的信息，动态地对设

备文件进行管理，包括设备文件的创建、删除等。

编译 udev 的步骤如下：

1） 下载 udev 源码
2） cd udev-117
3） make CROSS_COMPILE=mipsel-linux- DESTDIR=/nfsroot/root26
4） make CROSS_COMPILE=mipsel-linux- DESTDIR=/nfsroot/root26 install

根文件系统配置如下：

/etc/init.d/rcS：此文件包含启动 udev 的命令

```
# 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 的配置文件
/etc/udev/rules.d/*.rules：udev 的 rule 文件
/etc/udev/script/*.sh：udev 的脚本文件
/sbin/hotplug：hotplug 脚本
/usr/cpufreqd/*：cpufreqd 库文件和配置文件
/usr/alsa/*：ALSA工具和库文件
/usr/tslib/*：tslib 库文件和测试程序

5.4 创建 initramfs

参考下面步骤创建Linux 2.6内核的initramfs，这里假设您的根文件系统位于/rootfs目录下。

首先，创建cpio格式的映像文件：

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

接着，重新配置内核，选定下面选项：

[General setup] → [initial RAM filesystem and RAM disk (initramfs/initrd) support]

Initramfs source file(s): /rootfs.cpio.gz

重新编译内核，按照下面方法配置命令行参数重启内核即可：

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

6 如何在只有 sd 卡的情况下构建一个完整的系统

◆ 生成 u-boot-msc.bin 或 mbr-u-boot.bin 及烧录

对于 jz475x 系列在烧录时，本身 uboot 就需要烧录在第 0 个扇区，因此需要在 mbr 的开始加一条跳转指令，跳到第一个扇区执行。

若要使用 mbr-u-boot.bin 在编译之前我们需要先确定分区的数目，位置，大小，类型等信息，这些信息主要是在 apus.h 中来定义。默认的我们定义了四个主分区：

```
#define JZ_MBR_TABLE          /* configure the MBR below if JZ_MBR_TABLE defined*/
#define LINUX_FS_TYPE   0x83
#define VFAT_FS_TYPE    0x0B
/*======== Partition table ============ */
#define MBR_P1_OFFSET  PTN_SYSTEM_OFFSET
#define MBR_P1_SIZE    PTN_SYSTEM_SIZE
#define MBR_P1_TYPE    LINUX_FS_TYPE

#define MBR_P2_OFFSET  PTN_CACHE_OFFSET
#define MBR_P2_SIZE    PTN_CACHE_SIZE
#define MBR_P2_TYPE    LINUX_FS_TYPE

#define MBR_P3_OFFSET  0x3a400000
#define MBR_P3_SIZE    0x2000000
#define MBR_P3_TYPE    LINUX_FS_TYPE

#define MBR_P4_OFFSET  0x40000000
#define MBR_P4_SIZE    0xa8c00000
#define MBR_P4_TYPE    VFAT_FS_TYPE
```

MBR_PN_OFFSET 对应第 N 个分区相对 MBR 字节数，因为我们的 MBR 在第 0 扇区，所以 MBR_PN_OFFSET/512（一个扇区 512Byte）就是是烧录文件系统时，烧录工具的开始烧录位置。

MBR_PN_SIZE 对应第 N 个分区的的字节数。

MBR_PN_TYPE 对应第 N 个分区的数据类型，有 linux 和 vfat 两中选择

注意事项：对于 475x 系列，使用本方案时要注意以下几点

1：在第 0 个扇区的前 8 个字节写上：80 00 00 10 00 00 00 00 （uboot 生成的 mbr 中默认已添加）

这是一条跳转指令相当于

b 0x200

nop

2：以 apus 为例，u-boot 需要做的修改是：

根目录下的 Makefile apus_msc_config TEXT_BASE 要改为，0x80100200

msc_spl/board/apus/config.mk TEXT_BASE 要改为，0x80000200

3： kernel 需要做的确认是：

drivers/mmc/card/block.c

若有

brq.cmd.arg = req->sector + 16384;

直接改为

brq.cmd.arg = req->sector;

不再需要 8M 偏移

◆ 内核配置及烧录

1：下载 linux-2.6.31.3 源码包

2：make distclean

3：make apus_defconfig

4:：make xconfig (make menuconfig)

Device Driver

---- MMC/SD/SDIO card support

---- File system at MMC/SD support

---- JZ SOC Multimedia/SD/SDIO host controller 0 support

----- 选择 1/4/8 bit 数据线

----- 去掉 sdio support

----- 按照需求选择其他控制器及对应的数据线位数。

---- Memory Technology Device (MTD) support （可去）

File system

---- The Extended 4(ext4) filesystem

---- Yaffs2 Filesystems

---- YAFFS2 file system support （可去）

5：设置数据传输方式：

修改 driver/mmc/host/jzmmc/include/jz_msc_host.h 文件。

#define JZ_MSC_USE_DMA 1 //使用 dma 方式传输

#define JZ_MSC_USE_PIO 1 .//使用 pio 方式传输

#define USE_DMA_DESC 1 //使用 dma 的 descripter 方式传输

6：烧录，修改烧录工具中的 tool_cfg/LinuxFileCfg.ini 文件。设置如下：

[File2]
FileName=uImage
StartPage=8192 //从第 8192 个扇区开始烧录
NandOption=2

7：其他，对于 MSC0 内核中默认是把扇区 1-16384（8M）设置为只读，在访问卡时请通过分区(例如 mmcblk0p1)来访问,不要直接访问 mmcblk0.因为扇区 1-16384 保护了，所以我们在分区的时候，第一个扇区的 offset 要在 8M 之后。

◆ Uboot 参数设置（将 fs 放到第一个分区）：

setenv bootargs mem=64M console=ttyS3,57600n8 ip=off root=/dev/mmcblk0p1 rw
setenv bootcmd 'msc read 0x80600000 0x400000 0x300000;bootm'

◆ 文件系统制作及烧录

2：制作 ext4.img

genext2fs 是专门用来生成 ext2 文件系统镜像文件的工具，生成的 ext2.img 可以用 tune2fs 工具转化成 ext4 文件系统镜像。脚本如下：

```
#!/bin/sh
if [ $# -eq 1 ]; then
imagename=$1
GENEXT2FS=./genext2fs
TMPDIR=out/nfsroot/root/home/lltang/root26
OUTPUT=out$PWD/$imagename.img
#####from data to system####20101202##

echo $TMPDIR $OUTPUT

num_blocks=`du -sk $TMPDIR | tail -n1 | awk '{print $1;}'`
# add 1%
extra=`expr $num_blocks / 5`
reserve=10
[ $extra -lt $reserve ] && extra=$reserve
num_blocks=`expr $num_blocks + $extra`
num_inodes=`find $TMPDIR | wc -l`
echo $GENEXT2FS" -d "$TMPDIR" -b "$num_blocks" -I "$num_inodes"    "$OUTPUT
$GENEXT2FS    -d $TMPDIR -b $num_blocks -I $num_inodes $OUTPUT
e2fsck -fy $OUTPUT
tune2fs -j $OUTPUT
#tune2fs -L $imagename $OUTPUT
tune2fs -O extents,uninit_bg,dir_index $OUTPUT
e2fsck -fy $OUTPUT
fsck.ext4 -fy $OUTPUT
else
echo "./mkext2img.sh product_name imagename"
```

fi

TMPDIR 指向文件系统所在目录。

保存脚本为 mkext4fs.sh，执行

./mkext4fs ext4.img　生成 EXT4 镜像。

File ext4.img 查看镜像类型为 EXT4。

在此脚本中 extra 是为此文件系统预留的 block 数，这个值可根据需要变化，但是如果太小，可能会在执行 genext2fs 时报空间不足错误。

如果没有 genext2fs 和 tune2fs(在 e2fsprogs 包中可以获得)，可以到网站上自己下载，编译生成。

3：将文件系统镜像烧录到 mbr.bin 中设置的位置。

7 测试 Linux 内核和驱动

本节简单介绍如何测试君正 Linux 2.6.31.3 设备驱动程序，以验证驱动和硬件模块是否正常工作。

7.1 运行 Linux 内核

我们可以通过引导程序 U-Boot 启动 Linux 内核并挂载根文件系统来测试内核是否正常。首先配置和编译 Linux 内核，生成 uImage，通过 U-Boot 引导运行。如果 Linux 能正常启动并挂载成功根文件系统，说明 Linux 内核运行正常。

Linux 常用命令行参数：

mem=xxM	设置内存容量大小
console=tty0	设置控制台输出为 tty0
console=ttyS0,57600n8	设置控制台输出为串口 ttyS0，波特率为 57600，8 个数据位，无校验位
ip=dhcp	IP 地址通过 DHCP 服务获取
ip=192.168.2.84	设置静态 IP 为 192.168.2.84
nfsroot=192.168.1.4:/nfsroot/root26 rw	设置网络根文件系统，具有读写权限
root=/dev/mtdblock2 rw	设置根文件系统为/dev/mtdblock2，具有读写权限
rootfstype=yaffs2	根文件系统类型为 YAFFS2

君正 Linux 新增加的一些命令行参数有：

ts_debug	使能触摸屏测试模式

7.2 测试 Linux 设备驱动

- **测试 LCD 显示**

配置 Linux 时选择好目标板 LCD 屏的型号，在启动 Linux 时就应该能看到 LCD 屏幕有图案输出。如果没有，请检查 LCD 屏的硬件连接。

- **测试音频播放和录音**

MP3 播放测试：

```
# madplay test.mp3
```

录音和播放录音测试：

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

o **测试视频播放**

测试经过优化的在 Jz4750 上运行的 mplayer:

```
# mplayer50 –vf scale=480:272 binhe.264
```

测试经过优化的在 Jz4750D 上运行的 mplayer:

```
# mplayer50D –vf scale=480:272 binhe.264
```

o **测试 JFFS2 和 YAFFS2 文件系统**

配置 Linux 时选择 MTD 设备和 JFFS2/YAFFS 文件系统，启动 Linux 后就可以测试基于 MTD 的文件系统 JFFS2 和 YAFFS2：

首先，查看 MTD 分区表信息：

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

接着，格式化 MTD 分区：

```
# flash_eraseall –j /dev/mtd3      JFFS2 文件系统格式化
# flash_eraseall /dev/mtd4     YAFFS2 文件系统格式化
```

然后，挂载和测试文件系统：

```
# mount –t jffs2 /dev/mtdblock3 /mnt/mtdblock3      JFFS2 分区挂载
# mount –t yaffs2 /dev/mtdblock3 /mnt/mtdblock4     YAFFS2 分区挂载
# cp test /mnt/mtdblock4
# umount /mnt/mtdblock4
```

如果正常，说明文件系统已经工作。

o **测试 MMC/SD 卡**

在 Linux 正常启动后，插入 MMC 或 SD 卡到 SD 卡座，Linux 就能自动探测到卡的插入并读出分区表信息，这时就可以挂载和操作 SD 卡了：

```
# mount –t vfat /dev/mmcblk0p1 /mnt/mmc
```

```
# ls /mnt/mmc
```

o **测试触摸屏**

启动 Linux 后，运行下面命令校正触摸屏：

```
# ts_calibrate
```

运行下面命令测试触摸屏：

```
# ts_test
```

o **测试 USB Host**

插入 USB 设备（如 U 盘）到 USB Host 端口，Linux 应能探测到设备的插入，这时可以挂载和操作 U 盘了：

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

o **测试 USB Device**

首先，以模块方式编译生成 USB Device 驱动：jz4740_udc.ko 和 g_file_storage.ko；然后按照下面方法加载驱动：

```
# insmod jz4750_udc.ko
# insmod g_file_storage.ko file=/dev/mtdblock5,/dev/mmcblk0
```

这时我们挂载了两个 U 盘分区，一个是基于 NAND Flash 的分区 mtdblock5，另一个是基于 SD 卡的分区。现在用 USB 电缆连接 PC 和目标板的 USB Device 端口，PC 应该能识别到设备的插入并识别为 U 盘设备。

o **测试电源管理**

a. 测试系统睡眠、唤醒和关机
运行下面命令，系统将进入睡眠状态，在按 power 键之后唤醒。

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

运行下面命令，系统将关机，在按 power 键之后重启。

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

b. 测试动态变频管理

在编译内核时，在 CPU Frequency scaling 选项中选择 userspace 为默认的 governor。这样在目标板启动之后可以通过 sys 文件系统接口来控制 cpufreq 驱动。在目标板启动之后，如果没有 mount sys 文件系统，那么需要执行

```
# mount –t sysfs sysfs /sys
```

如果编译内核时选择的默认 governor 不是 userspace，而是 performance，那么需要执行命令把 governor 改为 userspace：

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

然后通过下面命令可以看当前系统频率(单位:KHz)：

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

另外，更详细的系统频率可由下面命令获得，该命令与 cpufreq 无关：

```
# 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
```

看当前可设置的最低系统频率:

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

看当前可设置的最高系统频率：

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

设置当前系统频率(频率单位是 KHz，以设为 112MHz 为例)：

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

在编译内核时，在 CPU Frequency scaling 选项中勾选 CPU frequency translation statistics details，可以使用三个命令看变频记录。首先，通过下面命令可获得变频的历史记录表：

```
# 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
```

通过下面命令便可获得系统在各个频率点的工作时间，单位时间和 jiffies 的单位相同，一般是 10ms（由编译内核时的 Kernel type 中的 Timer frequency 决定）。如下例中系统在 42000KHz 的工作时间是 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
```

下面命令可得到总的变频次数

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

8 Linux 2.6 音频驱动

Linux 2.6 的音频驱动包括 ALSA 和 OSS，可以单独使用，也可以同时使用，下面介绍它们的用法。

8.1 OSS 音频驱动

OSS 音频驱动在 linux-2.6.31.3/sound/oss 目录下。目前提供了 jz4750 内部 codec 的驱动，相关代码是 jz4750_i2s.c、jz4750_dlv.c、jz_ac97.c、jz_i2s.c、ak4642en.c 和 jzcodec.c。

当使用 OSS audio 驱动时，在 make xconfig 弹出的窗口中选择 sound/Open Sound System/Obsolete OSS drivers/jz On-Chip I2S driver 中的一种 codec，选择之后，再执行 make uImage 编译内核。

8.2 ALSA 音频驱动

ALSA 音频驱动在 linux-2.6.31.3/sound/soc 目录下，目前提供了 jz4750 及其内部 codec 的驱动，相关代码在 soc 目录下的 jz4750 和 codecs 中。其中 jz4750 目录下包括对 I2S 的支持，codec 目录下包括对内部 codec 的支持。

当使用 ALSA 音频驱动时，在 make xconfig 弹出的窗口中选择 sound/Advanced Linux Sound Architecture/System on Chip audio support 下选择相关选项，也可选上对 OSS emulate 的支持的选项，再执行 make uImage 编译内核。

ALSA 应用程序执行时，需要 ALSA 库的支持，ALSA 库放在/usr/alsa/lib 目录；ALSA utils 的测试程序放在/usr/alsa/bin 目录。

ALSA 的配置文件放在/usr/alsa/share/alsa.conf，它的路境由/etc/profile 中的环境变量 ALSA_CONFIG_PATH 指定。

声卡的配置文件放在/usr/alsa/etc/asound.conf 中，指定了一些设备名、频率转换和混音 plugin；该文件的路境由 ALSA 的配置文件决定。

8.3 ALSA 音频测试

参考下面几个命令来测试 ALSA：

设置放音增益（0 to 3）：

amixer set Master 1

查看放音增益：

amixer get Master

设置录音增益（0 to 31）：

amixer set Line 10

查看录音增益：

amixer get Line

测试录音：

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

-d : 20 秒
-c : 2 个声道
-t ： 文件类型
-r： 频率
-f : 16bits

测试放音：

aplay ./test.wav
或
aplay －D primary ./test.wav
参考 asound.conf 中指定的 primary

测试变频：

aplay -D rate_44k ./test.wav
参考 asound.conf 中指定的 rate_44k

测试混音：

aplay -D plug:dmix_44k ./test.wav &
再执行
aplay -D plug:dmix_44k ./test.wav
参考 asound.conf 中指定的 dmix_44k

9 NAND Flash 文件系统

在消费类电子产品中，NAND Flash 得到越来越广泛地应用。君正处理器 JZ47XX 集成了 NAND Flash 控制器，支持与 NAND Flash 的直接连接。同时，君正 Linux 2.6 内核也提供了对 NAND Flash 文件系统的支持，本节将介绍在 Linux 2.6 上如何构建基于 NAND Flash 的文件系统。

9.1 NAND Flash 驱动

在编译 linux 时需要配置 CONFIG_MTD、CONFIG_MTD_PARTITIONS、CONFIG_MTD_CHAR、CONFIG_MTD_BLOCK、CONFIG_MTD_NAND 为 y，并根据使用的君正处理器类型，配置 CONFIG_MTD_NAND_JZ4750 为 y。

JZ4750 集成了硬件 4-bit 和 8-bit BCH ECC 算法，前者可纠正 1016 字节内的 4 bits 错误，后者可纠正 1010 字节内的 8 bits 错误。在驱动中，我们让两者在每(512+n)个 bytes 中纠 4 bits 或 8 bits 错误，其中 n= (oob 中存放 ECC 之前的部分的字节数 / eccsteps)， eccsteps = pagesize / 512。这样， JZ4750 的 NAND 驱动就支持了 oob 区域（用来存放文件系统信息的部分）的校验（仅在 oob 紧接着 data 一起读出的时候支持）。另外，强烈建议对于 JZ4750，让 NAND 采用 DMA 的方式进行读写操作，编译 linux 使 CONFIG_MTD_NAND_DMA=y， 这可以大大减轻处理器负担，提高系统效率。建议让 CONFIG_MTD_NAND_DMABUF=n，否则，将多一次 buffer 拷贝，影响 NAND 读写速度。

JZ4750 支持 NAND 多片选，片选 CS1_N、CS2_N、CS3_N 和 CS4_N 都可以接 NAND，驱动中默认使用 CS1_N，如果还要使用其它片选，就在编译时使相应的 CONFIG_MTD_NAND_CS2、CONFIG_MTD_NAND_CS3 或 CONFIG_MTD_NAND_CS4=y。

目前市场上很多 NAND 都支持 multi-plane 读写操作，在进行 multi-plane 写操作时，NAND 内部可以同步对位于两个 plane 中的相应 page 进行写操作，大大缩短了写等待时间，显著加快 NAND 写速度。。对于 multi-plane 读操作，君正目前的 NAND 驱动采用的方法是分别读取两个 plane 中的 page 内容，因此 multi-plane 对读速度没有提升。虽然有些 NAND 支持专门的 multi-plane 读命令，但就算使用它，也不会加快多少读速，因为读时间主要消耗在外部总线对 NAND 读取上，而非等待 NAND 内部的 busy ready。要在驱动中使用 NAND 的 multi-plane 特性，编译时需要让 CONFIG_MTD_NAND_MULTI_PLANE=y。如果所选用的 NAND 不支持 multi-plane 特性，最好让 CONFIG_MTD_NAND_MULTI_PLANE=n，以免误把其识别为支持 multi-plane 特性的 NAND。

君正已将 linux MTD 系统转换为 64bit，以支持大于 2GB 的 NAND。并支持了 4KB page size 的 NAND。

9.2 NAND Flash 文件系统类型

Linux 使用 MTD（Memory Technology Devices）驱动来管理 NAND Flash 设备。因为 JFFS2 和 YAFFS2 文件系统自己实现了 NFTL（NAND Flash Translation Layer）功能， 所以在 MTD NAND Flash 设备上可以直接构建这两种文件系统。

君正还修改了 mtdblock 块设备驱动以支持 NAND Flash 的文件系统。修改后的 mtdblock 块设备驱动增加了逻辑块地址到物理块地址的映射、坏块管理、损耗均衡等功能。 这样，用户就可以在 NAND Flash 的 mtdblock 块设备上构建 EXT2、EXT3 和 FAT 等文件系统。

9.3 MTD 分区

在使用 NAND Flash 之前，首先要定义好 NAND Flash 的分区。MTD 分区的定义在 linux-2.6.31.3/drivers/mtd/nand/下的 jz4750_nand.c 文件里。用户需要根据自己的系统分别定义 NAND Flash 物理块的分区信息。

君正在标准分区信息的基础上增加了三个成员：use_planes，cpu_mode 和 mtdblock_jz_invalid。
其中 use_planes 用来决定该分区是否使用 multi-plane 读写操作，其值为 0 表示不使用 multi-plane，为 1 表示使用。如果所选用的 NAND Flash 不支持 multi-plane 操作，那么 use_planes 项的值无意义。如果所选用的 NAND 不支持 multi-plane， 或所有分区都不使用 multi-plane，那么最好在配置 linux 编译选项时，使 CONFIG_MTD_NAND_MULTI_PLANE=n。
cpu_mode 用来指定该分区使用 cpu 模式还是 dma 模式。
mtdblock_jz_invalid 用来指示该分区是否使用君正提供的文件系统转换层 mtdblock_jz.c，比如 YAFFS2 分区不需要使用它，将 mtdblock_jz_invalid 设为 1，那么在 mount YAFFS2 分区时，就不会做一些不必要的扫描，不会分配一些不必要的内存；VFAT 分区需要使用 mtdblock_jz.c，要将 mtdblock_jz_invalid 设为 0，否则使用该分区时会报异常。如果设 CONFIG_ALLOCATE_MTDBLOCK_JZ_EARLY=y，并且该分区使用 dma 模式，那么在 NAND 驱动初始化时，会给所有使用了 mtdblock_jz.c 的分区提前分配一个物理上连续的 NAND block 的缓存，这特别适用于会被当作 U 盘使用的 VFAT 分区，因为如果在加载 U 盘时才申请 1 个 block 的连续缓存，可能因为系统内存碎片太多而无法申请到。但这种提前申请的缓存无法释放。如果该分区使用 cpu 模式，就没有必要提前申请缓存，因为此时不需要连续的物理内存，一般都能申请到。

下面我们以在 jz4750 下使用 2GB 的 NAND Flash 为例来向读者介绍如何定义自己的分区：

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

NAND 分区表信息：

表 1　　2GB NAND Flash 的分区表示例

分区	起始物理块	结束物理块	分区大小	分区描述	是否使用 multi-plane	是否使用 cpu 方式	是否使用 mtdblock-jz
MTD BLOCK 0	0	7	4MB	MTD 分区 0 (bootloader)	不使用	不使用	不使用
MTD BLOCK 1	8	15	4MB	MTD 分区 1 (kernel)	不使用	不使用	不使用
MTD BLOCK 2	16	1023	504MB	MTD 分区 2 (rootfs)	不使用	不使用	不使用
MTD BLOCK 3	1024	2047	512MB	MTD 分区 3 (data)	使用	不使用	不使用
MTD BLOCK 4	2048	4095	1GB	MTD 分区 4 (vfat)	使用	不使用	使用

MTD 分区定义于文件 linux-2.6.31.3/drivers/mtd/nand/jz4750_nand.c 中，如下所示：

```
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},
  };
```

下面我们以在 jz4750 下使用 64MB 的 NAND Flash 为例来向读者介绍如何定义自己的分区：

参数：
Page size: 512 Bytes
Block size: 16 KB
Pages per block: 32
Row address cycles: 3
Total size: 64MB
Total blocks: 4096

NAND 分区表信息：

表 2　　64MB NAND Flash 的分区表示例

分区	起始物理块	结束物理块	分区大小	分区描述	是否使用 multi-plane	是否使用 cpu 方式	是否使用 mtdblock-jz
MTD BLOCK 0	0	255	4MB	MTD 分区 0 (bootloader)	不使用	不使用	不使用
MTD BLOCK 1	256	511	4MB	MTD 分区 1 (kernel)	不使用	不使用	不使用
MTD BLOCK 2	512	1535	16MB	MTD 分区 2 (rootfs)	不使用	不使用	不使用
MTD BLOCK 3	1536	2559	16MB	MTD 分区 3 (data)	使用	不使用	不使用
MTD BLOCK 4	2560	4095	24MB	MTD 分区 4 (vfat)	使用	不使用	使用

MTD 分区定义于文件 linux-2.6.31.3/drivers/mtd/nand/jz4750_nand.c 中，如下所示：

```
static struct mtd_partition partition_info[] = {
    {name:"NAND BOOT partition",
     offset:0 * 0x4000,
     size:256 * 0x4000,
     cpu_mode: 0,
     use_planes: 0,
     mtdblock_jz_invalid: 1},
    {name:"NAND KERNEL partition",
     offset:256 * 0x4000,
     size:256 * 0x4000,
     cpu_mode: 0,
     use_planes: 0,
       mtdblock_jz_invalid: 1},
```

```
    {name:"NAND ROOTFS partition",
     offset:512 * 0x4000,
     size:1024 * 0x4000,
     cpu_mode: 0,
     use_planes: 0,
     mtdblock_jz_invalid: 1},
    {name:"NAND DATA partition",
     offset:1536 * 0x4000,
     size:1024 * 0x4000,
     cpu_mode: 0,
     use_planes: 1,
     mtdblock_jz_invalid: 1},
    {name:"NAND VFAT partition",
     offset:2560 * 0x4000,
     size:1536 * 0x4000,
     cpu_mode: 0,
     use_planes: 1,
     mtdblock_jz_invalid: 0},
   };
```

注意：在 jz4750 下使用 64MB 的 NAND Flash 或 2GB 的 NAND Flash 分区需要手动修改宏定义来选择。

9.4 MTD Block 层

君正自己实现了“MTD Block”层。给出一个选项，在 NAND flash 上执行通常的文件系统，如 FAT 和 ext2。用于 VFAT 的分区可以在 u-boot 的 bootargs 中设置，这样内核就知道其它分区不基于 MTD Block 层，它们的加载速度更快。例如，如果 mtdblock5 用于 VFAT，可以设置 bootargs mem=64M console=ttyS1,57600n8 ip=dhcp root=/dev/mtdblock2 rw mtdblk=5，不基于 MTD Block 层的 mtdblock2 的加载速度更快。

“MTD Block”层的要素包括：

1. 块单元管理 （地址映射和块缓存操作）
2. 损耗均衡
3. 坏块管理
4. 写验证使能
5. ECC 算法的多种选择 （硬件 Hamming ECC，Reed-Solomon ECC，软件 Hamming ECC）

涉及“MTD Block”层的内核配置有：

* CONFIG_MTD_OOB_COPIES: 定义每个 block 关键的 oob 数据有多少拷贝。因为 page 数据可以通过 ECC 算法校正，但是 oob 数据不可以，我们希望通过这种方式确保 oob 数据正确。mtdblock-jz 转换层驱动使用 block 模式操作 NAND flash。它为每个 block 产生 oobinfo 数据的几个拷贝，这样即使其中一个出错，我们也能得到一个正确的拷贝。

* CONFIG_MTD_MTDBLOCK_WRITE_VERIFY_ENABLE: 定义它来使能写验证函数，在写操作的同时读回数据进行验证。

使用“MTD Block”层时采用以下步骤：

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

注意：

可以定义多个 VFAT 分区，所有 VFAT 分区都使用上述配置。

每个 VFAT 分区都在 RAM 中有自己的块缓存。通常，当访问地址超出块边界时触发块缓存刷新操作。最后一个块缓存通常在设备关闭的时候刷新到 NAND 设备中（eg: umount /mnt/vfat; use system call close(fd)）。

突然掉电而没有刷新最后一个块缓存将导致 VFAT 分区丢失最重要的数据，这些数据记录着文件系统管理信息，如 FAT table，inode 等。

为避免这些事情发生，需要立刻刷新块缓存。请在写操作完成后手动刷新块缓存。

MTD block 层驱动提供 ioctl 触发刷新块缓存操作。

eg:

```
# cp * /mnt/vfat
# sync          ; this step is necessary to flush the FS cache
# flushcache /dev/mtdblock3 ; this step is necessary to flush the NFTL block cache
```

9.5 YAFFS1 文件系统对硬件 ECC 的支持

对于(512+16)Bytes page 大小的 Nand Flash，内核可以自动选择 YAFFS1 作为其文件系统。君正增加了 YAFFS1 对硬件 ECC 的支持。为此，我们重新定义了硬件 ECC 方式下 yaffs_Spare 结构体，通过 CONFIG_MTD_HW_BCH_ECC 宏来区分软件 ECC 和硬件 ECC。

定义方式如下：

```
#ifdef CONFIG_MTD_HW_BCH_ECC
typedef struct {
    __u8 tagByte0;
    __u8 tagByte1;
    __u8 tagByte2;
    __u8 tagByte3;
    __u8 pageStatus;   /* set to 0 to delete the chunk */
  __u8 blockStatus;
  __u8 tagByte4;
    __u8 tagByte5;
    __u8 tagByte6;
      __u8 ecc[7];
  } yaffs_Spare;
```

```
#else
typedef struct {
    __u8 tagByte0;
    __u8 tagByte1;
    __u8 tagByte2;
    __u8 tagByte3;
    __u8 pageStatus;   /* set to 0 to delete the chunk */
    __u8 blockStatus;
    __u8 tagByte4;
    __u8 tagByte5;
    __u8 ecc1[3];
    __u8 tagByte6;
    __u8 tagByte7;
    __u8 ecc2[3];
} yaffs_Spare;
#endif
```

在 nand_oob_16 定义中，使用 oob 最后 7 个字节存放硬件 ECC 数据，其它字节存放文件系统信息。

```
static struct nand_ecclayout nand_oob_16 = {
    .eccbytes = 7,
    .eccpos = {9, 10, 11, 12, 13, 14, 15},
    .oobfree = {
        {.offset = 0,
         .length = 9}}
};
```

同时，在 mkyaffsimage.c 文件 write_chunk 函数中，在写 oob 数据时将上述结构体中 ecc 数据置为 0xff，由硬件进行 ECC 校验。

9.6 创建 YAFFS2 文件系统

由于 YAFFS2 完成了对 NAND Flash 的管理功能如地址映射、坏块管理等， 因此可以在 MTD 分区上直接创建 YAFFS2 文件系统。

在 MTD 分区上创建 YAFFS2 文件系统的方法如下：

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

还可以用 mkyaffs2image 工具生成 YAFFS2 映像，然后用 nandwrite_mlc 命令写到 MTD 分区上，如下：

在主机 PC 上运行下面命令生成 yaffs2 映像：

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

在目标板上：

```
# flash_eraseall /dev/mtd3
# nandwrite_mlc -a -o /dev/mtd2 rootfs.yaffs2
```

上面的 flash_eraseall 和 nandwrite_mlc 工具的源码位于 linux/drivers/mtd/mtd-utils/ 下，君正对源码进行了一些修改，以支持大于 2GB 的 MLC NAND。nandwrite_mlc 也可用于 SLC NAND。

在主机 PC 上运行命令 mkyaffs2image 生成 YAFFS2 映像，比如：

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

mkyaffs2image 工具在 PC 上使用，源码位于 linux/fs/yaffs2/utils/ 下。

```
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 !
```

烧录image_file要注意烧录工具ECCPOS的设置。目前最新驱动中JZ4750的ECCPOS设置为24，此值等于文件 linux-2.6/drivers/mtd/nand/nand_base.c 里的函数 nand_scan_tail() 中 chip->ecc.layout->eccpos[0]的值。

君正也对其做了修改以支持大于 2KB page size 和 multi-plane 的 MLC NAND。如果 mkyaffs2image 工具要处理的/rootfs/ 大小是 pagesize*N,，那么生成的 yaffs2 image 文件大小是 (pagesize + oobsize) * N，其中 oob 区域存放了每一个 page 的 yaffs2 文件系统信息(16 bytes)，以及这些信息的 ECC 校验码，ECC 校验可以采用 hamming （只能纠 1 bit 错误，用于 SLC NAND）或 reed-solomn ECC（能纠 2-10 bits 错误,用于 MLC NAND)。但 jz4750 不需要这种 ECC 校验,因为 yaffs2 文件系统调用 mtd->read_oob() 读取一个 page 的 NAND 内容时， MTD 层中，BCH 算法已经对 oob 区域进行了校验，不需要再在 yaffs2 层做校验。使用 BCH 时， eccpos 固定设为 24， 并且，为了便于 usb boot 烧录校验方便，mkyaffs2image 工具会使生成的映像文件中 oob 区域内 eccpos 以后的数据都设为 0xff。

所以，mkyaffs2image 工具会根据 linux 编译选项不同生成不同的 image。比如当内核 CONFIG_YAFFS_ECC_RS=y， 也即配置 yaffs2 使用 reed-solomn ECC 算法对 oob 区域进行校验时，mkyaffs2image 工具生成的 image 的 oob 区域就要存放相应的 reed-solomn ECC。当 CONFIG_YAFFS_ECC_HAMMING=y, image 的 oob 区域存放 Hamming ECC。当配置 CONFIG_MTD_HW_BCH_ECC=y, image 的 oob 区域内 eccpos（=24）以后的数据都设为 0xff。

让 mkyaffs2image 工具获得 linux 编译选项的方法是，每当编译 linux 改变 NAND ECC 种类或 yaffs2 中对 oob 使用哪种 ECC 的方法时，就进入 fs/yaffs2/utils/ 目录，重新 make 生成新的 mkyaffs2image 工具。

在使用 jz4750 的 BCH 算法时，为了使 u-boot 能正常读取在 linux 下烧录的 uImage，需要对 uImage 进行处理变为 uImage.oob, 即在每个 NAND pagesize 内容之后，添加一个 oobsize 的区域（其中的内容可以任意）。君正对 mkyaffs2image 工具进行了修改使其能对文件做这样的处理，对于 4KB 页 NAND:

```
$ mkyaffs2image 2 uImage uImage.oob
```

9.7 创建 FAT 和 EXT2 文件系统

在 MTD 基础上，君正公司移植了 mtdblock-jz 块设备驱动，负责对 NAND Flash 进行管理，如逻辑到物理块地址映射、坏块管理、损耗均衡等。这样就可以在 mtdblock 块设备上创建 FAT 和 EXT2 等文件系统。
在 cmdline（通过 u-boot 的 bootargs 传递进来）中，可以通过 mtdblk 字符串来通知 linux 对哪个或哪几个分区使用 mtdblock-jz 块设备驱动，比如 mtdblk=5，表示只有 mtdblock5 使用块设备驱动；mtdblk=2,4-6，表示 mtdblock2, mtdblock4, mtdblock5, mtdblock6 都使用块设备驱动。如果不指定，那么默认所有分区都使用块设备驱动，这样的坏处是会影响其它不使用 mtdblock-jz 块设备驱动的分区的 mount 速度，比如 yaffs2 分区。

相关命令示例如下：

```
# 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
```

9.8 UBI 设备

Linux 2.6 内核引入了 UBI (Unsorted Block Images)来对 Flash 进行管理。在 UBI 的基础上可以直接创建 UBIFS 文件系统。另外为了能在 UBI 上直接创建 EXT2 和 FAT 等文件系统，在 UBI 之上添加了 UBI BLOCK 设备层驱动（ubiblk.c 和 bdev.c）。

UBI 层主要完成以下功能：

- 坏块管理
- 损耗均衡
- 逻辑到物理块地址的映射
- Volume 信息存储
- Device 信息

使能 UBI，UBIFS 及 UBI BLOCK 设备驱动：

在 Linux 配置菜单，找到"Device Drivers" -> "Memory Technology Device (MTD) support" -> "UBI - Unsorted block images"，选定"Enable UBI"和"Common interface to block layer for UBI 'translation layers'"以及"Emulate block devices"。UBI 和 UBI BLOCK 既可以按模块方式编译，也可以直接编译到内核里。

在 Linux 配置菜单，找到”File systems” -> "Miscellaneous filesystems"，然后选定”UBIFS file sysem support”。UBIFS 既可以按模块方式编译，也可以直接编译到内核里。

若以模块方式编译，请运行“make modules”编译生成模块，运行“make modules_install INSTALL_MOD_PATH=/nfsroot/root26”安装模块到根文件系统目录下。

这里我们只介绍 UBIFS 和 UBI BLOCK 设备的使用和测试方法，关于更多技术细节，请参考以下主页：

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

9.8.1 UBIFS

这里假设你已经按照 module 方式编译好了 ubi 和 ubifs 驱动，并且已经安装到根文件系统的/lib/modules 目录下，下面就可以使用它了：

假设系统起来后，MTD 分区表如下：

```
# 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"
```

以 mtd5 为例来说明如何使用 UBI 和 UBIFS。

首先，格式化该分区：

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

接着，将 mtd5 作为一个 ubi 设备，加载 ubi 驱动：

```
# 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
```

这里要注意的是蓝色标记的部分。表明这个 UBI 设备总共有 2024 个可用物理块，保留 24 个物理块。在这个 UBI 设备上建立的所有卷的总大小为：
2024*logical eraseblock size(在这里为 258048bytes)　即
2024*258048 /1024 /1024 = 498.09375 M 取整数部分，即 498M

然后，在这个 ubi 设备上创建 ubi 卷，最多可创建 128 个卷，这里我们创建两个，卷的名字分别为”ubifs”和”vfat”：

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

加载 ubifs 驱动后可直接使用 ubifs:

```
# modprobe ubifs
```

挂载 ubifs 分区，ubi0:ubifs 是我们访问 ubifs 卷的一种方式:

```
# 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
```

注意：接下来介绍的 ubi block 设备，在创建完卷以后，需要先将 ubi 模块卸载，重新添加后再加载 ubi block 设备驱动模块。注意区分 ubifs 和 ubi block 操作流程的差异。

9.8.2 UBI Block 设备

这里假设你已经按照 module 方式编译好了 ubi 和块设备驱动，并且已经安装到根文件系统的/lib/modules 目录下，请参考以下方法使用：

启动内核进入 Linux 命令行界面，在命令行界面按照以下步骤测试 UBI 块设备。

```
# flash_eraseall /dev/mtd5                -- 格式化 MTD 分区
# modprobe ubi mtd=5                      -- 加载 UBI 驱动
# ubimkvol /dev/ubi0 -s 200MiB  -N vfat   -- 创建 UBI 卷 0
# ubimkvol /dev/ubi0 -s 298MiB  -N ext2   -- 创建 UBI 卷 1
# rmmod ubi                               -- 卸载 UBI 驱动
# modprobe ubi mtd=5                      -- 重新加载 UBI 驱动
# modprobe ubiblk                         -- 加载 UBI 的块设备驱动
```

运行以上三步是为了创建 ubiblock 设备，这时可以查看到/dev 目录下生成了两个 ubiblock 设备：

```
# 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
```

这时就可以格式化 ubiblock 设备，并拷贝文件了，如下：

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

系统重新启动之后，只需要按照下面步骤加载驱动就可以使用 ubiblock 设备，/dev/ubiblock0 是我们访问块设备卷的一种方式：

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

9.8.3 制作 UBI 镜像文件

使用 mkfs.ubifs 工具可以将包括多种格式的卷生成镜像文件，将镜像文件写到 nand 上后，可直接使用包含的卷及文件。注意：编译和使用 mkfs.ubifs 需要安装 lzo 库（lzo-2.02.tar.gz）,请到下述地址下载资源，具体操作请参考资源里的 mkfs.ubifs/README 文档。

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

假设制作这样一个镜像文件，名字为 ubi.img，里面包括两个卷 ubifs 和 vfat，分别为 ubifs 格式和 vfat 格式。具体步骤如下：

首先用 mkfs.ubifs 工具制作 ubifs 卷镜像 ubifs.img0：

```
$ export LD_LIBRARY_PATH=/opt/lzo/lib:$LD_LIBRARY_PATH    //在 PC 上操作
$ ./mkfs.ubifs  -r /nfsroot/root26/ -m 2048 -e 258048 -c 813 -o ubifs.img0
```

-c 指定这个卷的最大有效容量为多少个逻辑块
-e 逻辑块容量（字节为单位）
可以开发板上用 ubinfo 工具得知这些数据。

接着在 Linux 系统制作 vfat 卷镜像 vfat.img0， 此处镜像大小为 60M：

```
$ dd if=/dev/zero of=vfat.img0 bs=1M count=60  //在 PC 上操作
#losetup /dev/loop0 vfat.img0          //在 PAVO 板上操作
#mkfs.vfat /dev/loop0
#mount -t vfat /dev/loop0 /mnt/udisk
#cp /mnt/mmc/* /mnt/udisk/        //拷贝需要包含在镜像内的文件
#umount  /mnt/udisk
```

```
#losetup -d /dev/loop0
losetup: : No such device or address
```

用 ubirefimg: 将镜象文件 ubifs.img0 和 vfat.img0 格式化成带 LEB 逻辑块号镜象文件 ubifs.img 和 vfat.img：

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

然后用 ubinize 通过将 ubifs.img 和 vfat.img 生成一个镜像 ubi.img。准备一个 cfg 配置文件，内容如下：

```
# 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
```

然后运行：

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

至此，镜像文件 ubi.img 已经制作完成。

然后格式化分区， 用 nandwrite_mlc 工具 将 ubi.img 写到 nand flash 的 mtd5 上。

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

加载 ubi 驱动后即可使用 ubi 设备，加载 ubi 驱动：

```
# 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/
```

你会发现预先包含进 vfat 卷镜像的文件已然存在。

9.8.4 用 ubiupdatevol 更新卷的内容

下面介绍如何用 ubiupdatevol 去更新已知卷的内容，首先删除 vfat 卷上的所有内容

```
# 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
```

用 ubiupdatevol 更新 vfat 卷(下面的 vfat.img 是通过 ubirefimg 格式化过的镜像文件)

```
# 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
```

更新成功。

9.8.5 用 UBIFS 创建根文件系统

UBIFS 可以用来创建根文件系统，这时需要做：

1） UBI 和 UBIFS 编译到内核代码里
2） 在 Linux 命令行参数里设定与 UBI 关联的 MTD 分区：如”ubi.mtd=5”
3） 指定 Linux 命令行参数挂载 UBI 卷为根文件系统：如”root=ubi0:ubifs rootfstype=ubifs”

下面举例说明：
如果你已经按照创建了 ubifs 和 vfat 两个 UBI 空卷，请参考以下步骤。如果已经将 ubifs 的镜像文件写入 nand，且 ubifs 中已包含 root 文件系统，可略去步骤 1，2。

1） 设定 linux 命令行参数带”ubi.mtd=5”启动内核，以 NFS 挂载根文件系统；
2） 系统启动后，按照下面步骤挂载 UBI 分区、拷贝文件等：

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

3） 重新启动系统，设定 linux 命令行参数如下：ubi.mtd=5 root=ubi0:ubifs rootfstype=ubifs rw
OK

10 Linux 电源管理

手持嵌入式设备由于电池寿命短，所以需要减少运行功耗和待机功耗。君正 Linux 实现了电源管理功能，可以在系统运行时动态变频，并在系统长时间不使用时进入睡眠模式。其中，动态变频管理（cpufreq）可以允许系统在大约 50MHz 到 400MHz 范围内变频运行，而系统在睡眠模式下的电流可以降低到 2mA 以下。

10.1 动态变频管理

Linux2.6 实现了君正处理器的动态变频管理，通过用户层制定策略与内核提供管理功能交互，可以动态地改变 CPU 的工作频率。Linux 内核中与 JZ47xx 动态变频管理相关的代码位于代码树的 arch/mips/jz47xx/ 目录下的 cpufreq.c。

Linux 动态变频管理 cpufreq 提供了操作系统级别的变频功能，同时需要用户层来制定和执行策略。cpufreq 后台进程 cpufreqd 就是用来监测系统的运行状况，并根据不同的状况设定 CPU 的工作频率的。

要想在用户层使用后台进程cpufreqd，首先需要在配置linux内核编译选项时选择CPU Frequency scaling 功能选项, 其内有一些子选项，包含governor的选择和是否Enable CPUfreq debugging。 cpufreq有五种governor（将在10.1.3中介绍），要使cpufreqd正常运行，至少需要选择performance governor；在测试阶段往往希望看到系统变频信息，那么需要Enable CPUfreq debugging。在Enable CPUfreq debugging的前提下，在u-boot的linux启动参数bootargs加上loglevel=8 cpufreq.debug=<value>，就可以看到cpufreq的相应级别的运行信息。 <value>的值可以是 1、2、4，也可以是它们的或（3、5、6、7），具体意义如下：
1 to activate CPUfreq core debugging,
2 to activate CPUfreq drivers debugging（这是和 JZ47xx 相关的调试）, and
4 to activate CPUfreq governor debugging

后台进程 cpufreqd 需要用到三个库，分别是 cpufreqd-2.2.1、cpufrequtils-002 和 sysfsutils-2.1.0，这三个库都是 GNU 免费开源软件，我们对 cpufreqd-2.2.1、cpufrequtils-002 进行了一些修改，所以需要您从我们的网站上下载包含了这三个库的源码包 cpufreqd.tar.gz。下面将介绍 cpufreqd 在主机上的交叉编译和安装过程以及在目标板上安装和运行方法。

10.1.1 在主机上的交叉编译和安装

首先解压源码包：

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

下面需要分别进入三个库 cpufreqd-2.2.1、cpufrequtils-002 和 sysfsutils-2.1.0 的子目录,对这三个库进行

交叉编译和安装。

10.1.1.1 编译 sysfsutils-2.1.0

进入 sysfsutils-2.1.0 目录:

```
$ cd sysfsutils-2.1.0
```

然后执行下列命令开始编译和安装：

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

在主机上的安装路径<sysfsuitls_install_dir>下将生成下列文件(*.la 是 libtool 生成的临时文件)：

```
|-- 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 编译 cpufrequtils-002

进入 cpufrequtils-002 目录:

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

需要对 Makefile 中的两个变量重新赋值:

sysfsuitls_install_dir = <sysfsuitls_install_dir>
DESTDIR = <cpufrequtils_destdir>

其中<sysfsuitls_install_dir>为上文所述 sysfsuitls 的安装路径，<cpufrequtils_destdir>为自定义的 cpufrequtils 库的安装路径的前缀，DESTDIR 可以为空，cpufrequtils 的安装路径为$(DESTDIR)/usr.
执行下列命令开始编译和安装：

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

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

解决方法是将 .libs/libcpufreq.la 拷贝为.libs/libcpufreq.lai，并将内部的 installed=no 改为 installed=yes，然后再 make install。在安装路径$(DESTDIR)/usr 下将生成下列文件：

```
|-- 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 4 编译 cpufreqd-2.2.1

进入 cpufreqd-2.2.1 目录:

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

使用 configure-ingenic.sh 配置脚本对编译和安装过程进行配置，首先需要在该脚本中定义下面三个路径变量：

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

然后运行该脚本：

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

该脚本会生成 Makefile 和 config.h，需要修改生成的 config.h 中的路径，这些路径为 cpufreqd 在目标板上运行时所需要的相关文件的路径。默认路径的前缀为编译时所设的在主机上的安装路径，需要改为在目标板上的相应路径(以下所设路径仅供参考)：

/* 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/"

然后开始编译和安装：

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

在安装路径<cpufreqd_install_dir>下将生成下列文件：

|-- 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 在目标板上安装 cpufreqd

只需要安装 sysfsutils-2.1.0 和 cpufreqd-2.2.1。cpufrequtils-002 在交叉编译阶段已静态连接到 cpufreqd-2.2.1。

a. 安装 sysfsutils-2.1.0.
将主机<sysfsuitls_install_dir>/lib/下的 libsysfs.so、libsysfs.so.2、libsysfs.so.2.0.1 三个文件拷贝到目标板的/lib/或/usr/lib/下。如果想拷贝到其他特殊的<dir>,则以后在运行 cpufreqd 前需要执行命令：

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

b. 安装 cpufreqd-2.2.1.
将主机<cpufreqd_install_dir>/lib/下的 cpufreqd_cpu.so、cpufreqd_governor_parameters.so、cpufreqd_programs.so 三个文件拷贝到目标板上的 CPUFREQD_LIBDIR(在 config.h 中定义)路径下；
将主机<cpufreqd_install_dir>/sbin/cpufreqd 拷贝到目标板的/sbin/或/usr/sbin/下；
将主机<cpufreqd_install_dir>/bin/下的 cpufreqd-get 和 cpufreqd-set 拷贝到目标板的/bin/或/usr/bin/下；
将主机<cpufreqd_install_dir>/etc/cpufreqd.conf 拷贝到目标板上的 CPUFREQD_CONFDIR(在 config.h 中定义)路径下.

10.1.3 在目标板上运行 cpufreqd

首先根据系统需要来修改 cpufreqd 的配置文件 cpufreqd.conf，该配置文件决定了 cpufreqd 的行为.
Cpufreqd 及内核中的 cpufreq 驱动是针对桌面电脑开发的，其中很多和桌面相关的接口比如 acpi、apm 等我们没有用到，下面针对我们使用到的功能来介绍 cpufreqd.conf 的写法。它的完整介绍可参看<cpufreqd_install_dir>/share/man/man5/cpufreqd.conf.5。
cpufreqd.conf 分三个部分：General、Profile 和 Rule。
General 部分的示例：

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

pidfile 中存放了 cpufreqd 的 PID, 它的路径是 <CPUFREQD_STATEDIR>/cpufreqd.pid，CPUFREQD_STATEDIR 在交叉编译时由主机中的 cpufreqd-2.2.1/config.h（见 7.1.1.3）定义，示例中 CPUFREQD_STATEDIR="/var/"。poll_interval(单位是秒，可以设为小数)是 cpufreqd 读取系统状态的时间间隔，每一次读取状态后，都会根据当前状态来给各个 Rule 打分，得分最高的 Rule 对应的 Profile 将被执行。名为 Powersave Low 的 Profile 的执行时间长达 suspend_interval（单位是秒，应该设为整数）时，系统将自动进入睡眠状态。verbosity 指示 syslogd 后台进程记录 cpufreqd 进程的运行信息等级。

Profile 部分的示例（可以有多个 Profile）：

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

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

其中，minfreq 和 maxfreq 定义了该 Profile 允许的频率范围，可以用绝对频率表示，单位默认是 KHz，比如 minfreq=42000 表示最低频率为 42MHz；也可以用内核允许的最高频率（由 u-boot 的启动频率决定)的百分比表示，如果 u-boot 的启动频率为 336MHz，那么 maxfreq=100%表示最高频率为 336MHz. 内核中的频率表决定了系统只能运行在一系列特定频率上。有两种方法可以看到该频率表，一是直接看 linux 内核代码 arch/mips/jz47xx/cpufreq.c 中的初始化函数 jz47xx_cpufreq_driver_init()；二是在编译内核选择 debug cpufreq 的前提下，在 u-boot 的 linux 启动参数 bootargs 加上 loglevel=8 cpufreq.debug=1 后，在 kernel 启动过程中看该频率表。

policy 条目用来选择该 Profile 采用的内核 cpufreq 驱动中的 governor，内核 cpufreq 驱动支持五种 governor,分别是 performance、powersave、userspace、ondemand、conservative。performance 会让系统运行在指定频率范围[minfreq, maxfreq]内的最高频率，powersave 会让系统运行在 [minfreq,

maxfreq]内的最低频率。userspace 允许用户通过 echo 命令在[minfreq, maxfreq]内设置当前频率，通过 cat 命令查看当前频率（详细介绍见 5.2 中的“测试电源管理”）。ondemand 和 consecutive 会让内核根据 cpu 占用率自动在[minfreq, maxfreq]内选择合适的频率，consecutive 比 ondemand 变频速度平缓些。具体可用哪些 governor 是 linux 内核的 cpufreq 编译选项决定的.

我们只需要 performance governor 就可以让 cpufreqd 正常运行. 如果希望系统在空闲时间超过 suspend_interval 秒后自动睡眠，那么必须设置 name=Powersave Low 的 Profile。

Rule 部分的示例（可以有多个 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]
```

其中 name 是该 Rule 的名字，profile 是该 Rule 得分最高之后将要执行的 Profile 的 name. Rule 的条件有两种，一是 cpu 的占有率范围 cpu_interval，二是当前运行的程序 programs. 比如 cpu_interval=0-15 表示 cpu 占用率为 0%~15%，programs=mplayer,madplay 表示当前运行的程序有 mplayer 或 madplay. 每个 Rule 的满分由自身条件的个数决定，有 n 个条件的 Rule 的满分是(100+n)%，比如上面所列的名为“CPU not busy”的 Rule 满分为 101%，名为“Movie Watcher”的 Rule 满分为 102%。如果存在两个 Rule 都为满分时，分数较高的 Movie Watcher 将被执行。如果两个 Rule 得分相同，那么位置靠前的优先执行。

设置好 cpufreqd.conf,之后，执行命令 cpufreqd 即可启动 cpufreqd 在后台运行(要保证已经 mount sys 文件系统)：

```
# cpufreqd
```

如果想手动选择 cpufreqd 执行的 Profile, 需要在 cpufreqd.conf 的 General 栏中设置 enable_remote=1. 并在 cpufreqd 命令后加上-m 选项，即

```
# cpufreqd –m
```

这样就可以通过命令 cpufreqd-get 打印出 cpufreqd.conf 中所有可用的 Profile，通过命令 cpufreqd-set n, 来选择执行第 n 条 profile.

如果想看到 cpufreqd 的运行信息，那么可以加-V 选项，共有 0 到 7 八个信息级别，0 的信息最少，7 的最多。比如要想打印出各个 Rule 的得分,可以执行：

```
# cpufreqd –V 6
```

加 –D 选项可以让 cpufreqd 在前台运行并打印出运行信息。
加--help 选项可以看帮助信息：

```
# cpufreqd –help
```

10.2 睡眠和唤醒管理

系统如果长时间处于闲置状态时，可以让其进入睡眠模式。在这种模式下，系统大部分模块都置于低功耗模式，SDRAM 处于自刷新模式并保存程序运行的现场，只保留 RTC 时钟工作以唤醒系统。Linux 内核中与 JZ47xx 睡眠和唤醒管理相关的代码是位于代码树 arch/mips/jz47xx 目录下的 pm.c。

使用睡眠和唤醒管理需要在配置 linux 内核编译选项时选择 Power Management support 选项下的 Legacy Power Management API (DEPRECATED) (PM_LEGACY)。

JZ47xx 处理器睡眠和唤醒的流程如下：

1） 系统根据运行状态(cpufreqd、命令行或按键)进入睡眠模式；
2） 操作系统根据 PM（power management）机制调用各个设备驱动的 suspend 函数让设备进入睡眠模式；
3） 操作系统保存 CPU 现场状态（该步骤仅限于 JZ4750 由芯片自身自动保存）；
4） 将 JZ47xx 内部模拟设备关闭，将 GPIO 置为合适的状态，JZ47xx 执行睡眠指令进入睡眠模式，并让 SDRAM 进入自刷新模式；
5） 通过按键或者其它中断唤醒 JZ47xx；
6） CPU 被唤醒并 reset，根据 reset 状态寄存器判断当前为 hibernate 唤醒复位，CPU 跳到 resume 函数恢复睡眠前的系统状态（该步骤仅限于 JZ4750 不会 reset，而是直接继续执行睡眠指令之后的语句）；
7） 将 JZ47xx 内部模拟设备的开关状态和 GPIO 状态恢复到睡眠之前的状态，并调用各个设备驱动的 resume 函数让设备恢复工作；
8） 系统正常运行。

用户可以通过执行下面命令进入睡眠模式：

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

10.3 开关机管理

开关机管理实现的机制是：短按 power key 系统睡眠，再按 power key 系统唤醒；长按 power key 系统关机，再按 power key 系统重启。Linux 内核中与 JZ47xx 睡眠和唤醒管理相关的代码是 drivers/char/jzchar/poweroff.c。

使用开关机管理需要在配置 linux 内核编译选项时选择 Device Drivers->Character devices->JZSOC char device support->JZ board poweroff support。

开关机管理中让系统睡眠和关机的具体操作默认分别是由 poweroff.c 中的宏 DO_SUSPEND 和 DO_SHUTDOWN_SYSTEM 实现，这种方式简单快速。另外可以在 poweroff.c 中
#define USE_SUSPEND_HOTPLUG
就可以通过调用脚本（位于/sbin/hotplug）实现睡眠和关机，这种方式可以实现在睡眠和关机之前做各种涉及用户空间程序的准备工作，但调用脚本的速度较慢，特别是播放视频时调用脚本会有更大的延时。

10.4 wm8310 电源管理芯片管理

由于市场的发展和需求，为了实现省电并且便于控制，有些系统中可能会用到单独的电源管理的芯片，此处以 wm831x 系列芯片中的 wm8310 芯片在 aquila 板上的应用为例，做简单的介绍：

1. wm8310 电源管理芯片的特性：

   wm8310 是一个单独的用于电源管理的芯片，它由四个可编程的 DC-DC 转换器，11 个 LDO 稳压器组成，它可以提供的电压范围如下：

   - 2 x DC-DC 动态降压转换器 (0.6V - 1.8V, 1.2A, DVS)
   - 1 x DC-DC 动态降压转换器 (0.85V - 3.4V, 1A)
   - 1 x DC-DC 升压转换器 (up to 20V @ 40mA)
   - 1 x LDO 稳压器 (0.9V - 3.3V, 300mA, 2Ω)
   - 2 x LDO 稳压器 (0.9V - 3.3V, 200mA, 2Ω)
   - 3 x LDO 稳压器 (0.9V - 3.3V, 100mA, 4Ω)
   - 2 x 高性能LDO 稳压 (1.0V - 3.5V, 200mA, 1Ω)
   - 2 x 高性能LDO 稳压 (1.0V - 3.5V, 150mA, 2Ω)
   - 1 x 'always on' 稳压 (0.8V - 1.55V, up to 10mA)

   其中每个LDO稳压器的输出电压和DC-DC转换器的电压都可以单独的通过设置寄存器的值来控制。可以通过i2c 和 SPI 两种方式连接到其它的控制芯片中（aquila 板是通过i2c 连接的）。主要适用于便携式媒体播放器，便携式导航设备，基于闪存的便携式音频设备，电子图书，电子游戏设备，手持录音机等。

2. wm8310 相关的驱动程序；

   drivers/mfd/wm831x-core.c
   drivers/mfd/wm831x-otp.c
   drivers/mfd/wm831x-irq.c
   drivers/regulator/wm831x-ldo.c
   drivers/regulator/wm831x-dcdc.c
   drivers/regulator/wm831x-isink.c
   drivers/power/wm831x_power.c
   drivers/power/wm831x_backup.c

drivers/input/misc/wm831x-on.c
drivers/rtc/ rtc-wm831x.c
drivers/gpio/wm831x-gpio.c
drivers/watchdog/wm831x_wdt.c
drivers/video/backlight/wm831x_bl.c
drivers/hwmon/wm831x-hwmon.c
drivers/leds/ leds-wm831x-status.c

3. make xconfig，在 Device Drivers 下增加如下选项：

i2c support => I2C Hardware Bus support =>JZ47XX I2C Interface support
Multifunction device drivers => Support Wolfson Microelectronics WM831x PMICs
Power supply class support => WM831x backup battery chargersupport
=> WM831x PMU support
Voltage and Current Regulator Support
=> Wolfson MicroelcronicsWM831x PMIC regulator driver
Input device support => Miscellaneous devices => Wm831x on pin
Real Time Clock => wm831x Rtc support
注意：

(1) 这里有些驱动在 aquila 板中没有使用，如：wm831x_wdt.c，wm831x_bl.c 等，故 xconfig 中没有选择，也就没有编译.

(2) 有些驱动是有依赖关系的，由于 aquila 板中 wm8310 管理芯片是通过 i2c 和 cpu 进行通信的，所以所有的驱动依赖于 i2c 驱动，故在 xconfig 中要先选择上 i2c 驱动（即 xconfig 中的 JZ47XX I2C Interface support 选项）；另外，xconfig 中的 Support Wolfson Microelectronics WM831x PMICs 选项也要先选择上，它对应于驱动程序：wm831x-core.c，wm831x-otp.c， wm831x-irq.c ，因为，其它的驱动如：WM831x PMU support（对应于 wm831x_power.c）等，依赖于 xconfig 中的 Support Wolfson Microelectronics WM831x PMICs 选项，否则，有些选项可能会找不到。

(3) wm831x Rtc support 选项依赖于 Wm831x on pin 选项，故要先选 Wm831x on pin 选项，wm831x Rtc support 选项才能出现；

4. 在其它驱动中使用该驱动的接口；

int wm8310_ldo_enable(int enable)；
int wm8310_ldo_disable(int disable)；
包含头文件：#include <include/linux/mfd/wm831x/core.h>
#include <asm/jzsoc.h>
如打开或关闭某路 ldo 电压，可以用：
打开某路电压 ：wm8310_ldo_enable（WM8310_BT_WIFI_VPA）
关闭某路电压 ：wm8310_ldo_disable(WM8310_BT_WIFI_VPA)
（1）参数可从相应的电路图中得到，其中 WM8310_XXXX 是要打开或者关闭的某路 ldo 的 id ，值为 0~9，如：ldo1 对应 id : 0, ldo2 对应 id : 1, 依此类推，ldo10 对应 id : 9 。在 aquila 板中，可以传参数如下：
WM8310_BT_WIFI_VPA
WM8310_CIM_VCORE
WM8310_GPS_VDD

WM8310_LCD_VDD
WM8310_LCD_VDDA
WM8310_VCC_MSCI
WM8310_BT_WIFI_VCORE
WM8310_BT_WIFI_VCC
WM8310_CIM_AVDD
WM8310_CIM_DVDD

其中，WM8310_XXXX 中的 XXXX 对应于电路图中要打开或关闭的电源 pin ，这些 pin 分别与 ldo1 ~ ldo10 的电源输出 pin 相连接 ；

注意：在 aquila 板中，ldo11 是 always-on 的不要控制其开和关；

(2) 可以通过返回值校验调用是否成功，两个接口，成功均返回 0 ，失败则返回非 0 ；

5. 上面的驱动加载成功后，会在 sys/ 文件系统中注册相关的接口：

(1) /sys/bus/platform/devices/wm831x-ldo.0 ~ wm831x-ldo.5
/sys/bus/platform/devices/wm831x-aldo.6 ~ wm831x-aldo.9
/sys/bus/platform/devices/wm831x-alive-ldo.10

这些接口中提供了该 ldo 的相关属性，可以进入上边这些文件夹中的 regulator:regulator.* 文件夹，从里面可以用 cat 命令查看该 ldo 的属性值；

name : wm831x-ldo1 ~ wm831x-ldo11

type : 现在都为 voltage

microvolts，min_microvolts，max_microvolts ：都是该 ldo 的设置电压（因为衡压，所以它们的值都相等）

state ：disable or enable ,代表该 ldo 的开关状态；

(2) /sys/class/power_supply

这个接口提供了电源供电管理，也是上层使用的主要接口。

以 /sys/class/power_supply/battery/ 下的属性为例，可以用 cat 查看：

cat	type	battery , usb or mains
cat	status	显示充电类型（chargeing or not charging）
cat	technology	显示电池的工艺技术；
cat	health	显示电池的健康状态；
cat	capacity	显示电池的剩余电量的百分比；
cat	voltage_now	显示电池的当前电压；
cat	voltage_max	显示电池能够供电的最高电压：4.2 v；
cat	voltage_min_design	显示电池能够供电的最低电压：3.6v；
cat	batt_temp	显示电池的温度 ，暂不支持；
cat	batt_vol	显示电池的电压 ；

6. 上层中可以通过 sys 文件系统来使用电源管理模块，以 android 为例，如下；

(1) 在 android 层，对电池的管理主要是通过 servers 来实现的；

在 frameworks/base/services/java/com/android/server/ BatteryService.java 中，定义了 BatteryService 类，

在 frameworks/base/services/java/com/android/server/ SystemService.java 中创建

BatteryService 对象，并将该实例化的 BatteryService 添加到 system 中。

BatteryService 类中实现了监听文件系统/sys/class/power_supply 下的 uevent ， 而这些 uevent 的是由 linux 内核实现的自动上报。当 BatteryService 监听到有 /sys/class/power_supply 中上报的 uevent 后，就会调用该类的 update() 方法。而 update() 方法首先通过 native_update 调用

android_server_BatteryService_update() 函数，从 sysfs 文件系统中读取相关状态，得到电池相关的状态信息。读取文件系统的 android_server_BatteryService_update() 函数在 /frameworks/base/services/jni/com_android_server_BatteryService.cpp 中实现。

需要读取的状态如下：

"/sys/class/power_supply/ac/online"	指示充电器是否插入
"/sys/class/power_supply/usb/online"	指示 usb 是否插入
"/sys/class/power_supply/battery/status"	指示是否进行充电
"/sys/class/power_supply/battery/health"	指示电池的健康状态
"/sys/class/power_supply/battery/present"	指示电池连接是否正常
"/sys/class/power_supply/battery/capacity"	指示电池的剩余电量百分比
"/sys/class/power_supply/battery/batt_vol"	指示电池的当前电压
"/sys/class/power_supply/battery/batt_temp"	指示电池的温度，暂不支持
"/sys/class/power_supply/battery/technology"	电池的工艺，如：锂电池

然后 update 根据读到的状态更新 BatteryService 的成员变量，并广播一个 Intent 来通知其它关注电源状态的组件。

关注电源状态的组件接收 Intent 来更新相关的信息。

11 无线设备配置

互联网已经融入人们的生活，人们希望能够随时随地接入互联网查看新闻、收发邮件，对互联网的依赖日益增强。随着无线技术的日渐成熟，无线网络覆盖范围不断扩大，机场、旅馆、办公大楼、城市的公共场所几乎都有覆盖，方便了人们接入互联网。

君正处理器 JZ47xx 提供了丰富的外设接口，可以支持多种接口的无线设备。目前支持的 WIFI 设备有：

Interface	Device	OS	Max Throughput	Productor
USB	VT6656	celinux040503 linux-2.6.31.3	3Mbit/s	VIA
USB	ZD1211	celinux040503 linux-2.6.31.3	5.5Mbit/s	Atheros
SDIO	AR6001X	linux-2.6.31.3	16Mbit/s	Atheros
SDIO	88W8686	linux-2.6.31.3	16Mbit/s	Marvell
SPI	88W8686	linux-2.6.31.3	10Mbit/s	Marvell

11.1 内核配置

要支持无线网络设备, linux kernel 需选择"Wirelesss Extensions"。

在 Linux 配置菜单，找到" Networking" -> "wirelesss" -> "Wirelesss Extensions"，选择"Wirelesss Extensions"。

Linux-2.6.31.3 的无线网络接口版本（WIRELESS_EXT）为 22。

目前 SDIO 接口 wifi 驱动与 MMC/SD 卡驱动不兼容。 如果要使用 SDIO 接口的 wifi 卡，在内核配置选项中需要将 MMC/SD 驱动去掉。

11.2 wlan 模块驱动加载

君正平台目前支持的无线网络驱动是以模块方式加载的。用户可以从 wifi 厂商得到源码，以及君正平台补丁，重新编译使用。

11.2.1 加载 VT6656 驱动：

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

11.2.2 加载 ZD1211 驱动：

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

11.2.3 加载 GSPI8686 驱动：

GSPI8686 驱动分 2 个主层次：IO 层（gspi.ko），wlan 控制层（gspi8686.ko）。

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

```
==============================================================
gspi.ko 模块参数:
clkdiv: SPI时钟分频系数。取值范围[0,15].
    spi clock = (pllout/(clkdiv+1))/4
    假如系统pll时钟360M Hz, clkdiv=5, then
    spi clock = (360/(5+1))/4 MHz = 15 MHz
gspi_irq_pin: 中断信号线。GSPI8686与主机通信需要此中断信号。
    PAVO开发板使用GPIO127作为中断信号，spi_irq_pin=127.
    用户可以根据实际情况分配GPIO作为此中断信号，如果分配GPIO100作为此中断信号，
    则设置spi_irq_pin=100.
chipselect: SPI设备号。JZ4750处理器的SPI控制器支持2个slave设备，CE0和CE2。
    PAVO开发板使用CE0与GSPI8686连接，所以设置chipselect=0。
    用户可以根据实际情况选择CE0或CE2连接GSPI8686。
    如果选择CE2，则设置chipselect=2。
Helper_name,fw_name: 具体路径为你存放厂家提供的Firmware Image的路径，用户根据实际情况
    使之指向正确路径
==============================================================
```

11.2.4 加载 SD8686 驱动：

SD8686 驱动分 2 个主层次：IO 层（sdio.ko），wlan 控制层（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 模块参数:
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 加载 AR6000 驱动：

SD8686 驱动分 4 个主层次：IO 层（sdio_jz_hcd.ko），SDIO 总线层（sdio_busdriver.ko），SDIO 协议层（sdio_lib.ko），wlan 协议层（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 模块参数:
    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, 无线网络配置工具

Wireless tools 是一组无线网络配置工具。Wireless tools 是的一个开源项目，网址:
http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

下面介绍常用的命令：

iwconfig	查询或配置无线网卡参数
iwlist	查询 iwconfig 没有显示的无线网卡参数
iwpriv	查询或配置无线网卡厂商提供的网卡特定参数

查看无线网卡 eth1 信息：

```
# 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 参数

```
# 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
```

搜索周围的 AP：

```
# iwlist eth1 scan
eth1        Scan completed :
            Cell 01 - Address: 00:02:6F: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: 00:14:6C: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 配置无线网络步骤

成功加载无线网卡驱动后，配置无线网络，步骤如下：

```
# ifconfig eth1 192.168.1.111          // 设置本机 IP（与 AP 同一个网段）
# iwconfig eth1 mode managed           // 设置无线网络模式
```

```
# iwconfig eth1 key 1234567890 key on     // 设置网络密码
# iwconfig eth1 essid AP-name             // 设置 AP 名称
```

此时配置完毕，可以连接 ap：

```
# ping 192.168.1.1                        // 测试与 AP 的连接
```

11.5 iperf 网络测试工具：

Iperf 是一款小巧的网络测试工具,可以测试两台主机之间的网络吞吐量.

Server 端：

```
# 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
```

客户端：

```
# 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
```