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 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
  - jz4770.c JZ4770 相关代码，如系统 timer、PLL 的初始化等
  - jz4770_ddr.c DDR2 控制器初始化代码，一般情况下您不需要改动这里的代码，DDR 的参数配置在相应的板级配置文件中包含(如 pisces.h)
  - jz_serial.c 串行口 UART 驱动程序
  - jz4770_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
  - jz_mmc.c MMC/SD 卡驱动程序
  - jz4770_nand.c JZ4770 NAND flash 驱动

- 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。
- 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/jz4770.h JZ4770 相关的头文件定义。
  - include/configs/pisces.h 基于 JZ4770 的开发平台 PISCES 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 MSC 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：

3.2.1 JZ4770 PISCES 平台， Nand Flash 启动：

对于 JZ4770, 编译之前需要在 include/configs/pisces.h 中，设置好相关 ECC 参数：

```
#define CFG_NAND_BCH_BIT        8
#define CFG_NAND_ECC_POS        24
```

另外，在 include/asm-mips/jz_mem_nand_configs/NAND_K9GAG08U0D.h 中确认 nand flash 的参数：

```
#define CFG_NAND_BW8            1
#define CFG_NAND_PAGE_SIZE      4096
#define CFG_NAND_OOB_SIZE       218
#define CFG_NAND_ROW_CYCLE      3
#define CFG_NAND_BLOCK_SIZE     (512 << 10)
#define CFG_NAND_BADBLOCK_PAGE  127
```

然后，确认你的 DDR2 的大小，如果是 256M，则 include/asm/ jz_mem_nand_configs/ DDR2_H5PS1G63EFR-G7C.h 中设置

最后，

```
$ make pisces_nand_config
```

```
$ make
```

编译后生成 u-boot-nand.bin。

3.2.2 JZ4770 PISCES 平台， EMMC 启动：

```
$ make pisces_msc_config
$ make
```

编译后生成 u-boot-msc.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，目前除了 JZ4730，其它君正处理器都支持 USB 引导。

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，具体请参照烧录工具文档。

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：

```
PISCES # help tftpboot
PISCES # 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, pisces 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）

```
PISCES # setenv ipaddr 192.168.2.84
PISCES # setenv serverip 192.168.1.4
PISCES # setenv ethaddr 00:2a:c6:2c:ab:f1
PISCES # askenv bootcmd
bootcmd: tftpboot;bootm
PISCES # setenv bootargs mem=256M console=ttyS1,57600n8 ip=dhcp
nfsroot=192.168.1.4:/nfsroot/root26 rw
PISCES # setenv bootfile uImage
PISCES # saveenv
PISCES # 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。本文将具体描述基于 JZ4770 的 pisces 平台相关的君正 Linux 2.6.31.3 内核和驱动程序。

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
  - jz4770/：JZ4770 处理器相关目录和文件
    - *.c： JZ4770 处理器通用文件
    - board-pisces：PISCES 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
    - pisces_defconfig：JZ4770-PISCES 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-jz4770/：JZ4770 处理器相关头文件
- sound
  - oss/：OSS 音频驱动
    - jz4770_i2s.c：I2S 通用驱动

        - jz4770_dlv.c：JZ CODEC 驱动
- 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
    - 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、mkubifs 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/musb: USB otg 驱动
        - musb*.c
        - jz4760.c

    - usb/gadget： USB device gadget 驱动
        - file_storage.c
    - video/：LCD framebuffer 驱动
        - jz4760_lcd.c

4.2 配置和编译 Linux

首先，选择目标板的配置：

```
$ make pisces_defconfig(JZ4770 PISCES 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。

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=tty1	设置控制台输出为 tty1
console=ttyS1,57600n8	设置控制台输出为串口 ttyS1，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
ubi.mtd=2	加载 MTD device 2
root= ubi0:ubifs rw	设置根文件系统为 ubi0:ubifs，具有读写权限
rootfstype=ubifs	根文件系统类型为 UBIFS
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**

```
# mplayer  binhe.264
```

- **测试 JFFS2 和 UBIFS 文件系统**

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

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        UBIFS 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
# ubiattach /dev/ubi_ctrl –m 4
# ubimkvol /dev/ubi1 –s 200MiB –N ubivol –n 2
# mount –t ubifs ubi1:ubivol /mnt/ubifs            UBIFS 分区挂载
# cp test /mnt/ubifs
# umount /mnt/ubifs
```

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(OHCI)**

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 OTG**

otg 驱动支持三种模式：device、host 和 otg(即能做 host 又能做 device)相关的配置在：

```
Device Drivers  ->
    USB support ->
        Inventra Highspeed Dual Role Controller
        *** Ingenic OTG USB support ***
        Driver Mode (USB Host、USB Peripheral、Both host and peripheral)
```

a. 测试 device 模式：

选择 Driver Mode 为 device 模式，以模块方式编译生成 g_file_storage.ko，按照如下方法加载模块：

```
# 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。

b. 测试 host 模式：

选择 Driver Mode 为 USB Host;插入 OTG 线(A 型)，就可以像测试 USB Host(OHCI)测试 U 盘设备

c. 测试 otg 模式：

选择 Driver Mode 为 Both host and peripheral;同时支持 device 和 host 模式的功能

注意： 1. 使用 OTG 模式时需要根据板极硬件情况修改 drviers//usb/musb/jz4760.c 中的 GPIO_OTG_ID_PIN 相关定义；OTG 的 host 功能正常使用前也要加载相应的 g_file_storage.ko 等 gadget 模块

2. 如果要使用 hotplug 功能需要在内核菜单 otg 选项的同级目录下添加以下支持：

Support Ingenic USB Device Controller Hotplug

同时根据板极硬件情况修改 divers/usb/musb/vbus_hotplug.c 中的 OTG_HOTPLUG_PIN 定义

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 mem > sys/power/state
```

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 jz4770 内部 codec 的驱动，相关代码是 jz4770_i2s.c、jz4770_dlv.c。

当使用 OSS audio 驱动时，在 make xconfig 弹出的窗口中选择 Scound card support/Open Sound System/jz On-Chip I2S driver、I2S codec type(Internal On-Chip codec on Jz4770)之后，再执行 make uImage 编译内核。

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

现在我们可以不使用 NAND,在 SD 卡上构建一个完整的系统，在 sd boot 时我们用的都是 msc0，对于 476x 系列，msc0 有两组 gpio 定义，只有 GPIO A 组的可以用作 boot,这组 gpio 只是定义了 4 位数据线，如果想使用 8 位数据线，需要将 GPIO E 组的 DAT4-DAT7 接过来，硬件上需要做相应的修改，同时在初始化 gpio 时，也要将 E 组的那 4 根 DATA 线初始化，在我们发布的 patch 中，默认的都是用的 4bit 数据传输，如果想用 8bit 数据传输，请修改 uboot 和 kernel 中的 gpio 定义，硬件上也做相应的处理。

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

6.1.1：对于 jz476x 系列请先确认使用的 eMMC 是遵循 sd 协议还是 mmc 协议，根据卡的类型修改 pisces.h，默认我们使用的是 SD 卡：

```
#ifdef CONFIG_MSC_U_BOOT
#define CONFIG_MSC_TYPE_SD
#endif
```

如果使用的是 MMC 卡则

```
#ifdef CONFIG_MSC_U_BOOT
//#define CONFIG_MSC_TYPE_SD
#endif
```

此宏在 msc_boot_jz4770.c 中的使用。

6.2.2：u-boot-msc.bin 和 mbr-u-boot.bin 的主要区别是:mbr-u-boot.bin 是在 u-boot-msc.bin 的基础上加上 512Byte 的 mbr。

如何生成 mbr-u-boot.bin 呢？以 pisces 为例，打开主 Makefile, 在 pisces_msc_config 中添加：@echo "CONFIG_MBR_UBOOT = y" >> $(obj)include/config.mk

如果想生成 u-boot-msc.bin，那么就把这句话去掉。

对于 jz476x 系列再烧录时，u-boot-msc.bin 是烧录在第 1 个扇区，mbr-u-boot.bin 是烧录在第 0 个扇区。

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 pisces.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

◆ 内核配置及烧录

1：下载 linux-2.6.31.3 源码包

2：make distclean

3：make pisces_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
        -----  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  （去掉!!!）

File system
    ----  The Extended 4(ext4) filesystem
    ----  Miscellaneous filesystems
        ---- UBIFS file system support  （去掉）
```

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=256M console=ttyS1,57600n8 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

dd if=/dev/zero of=xxx.img bs=1M count=256
> mkfs.ext4 xxx.img
> mount -o loop xxx.img /mnt/loop
> cp rootfs/* /mnt/loop/ -rf
> sync
> umount /mnt/loop

3：将文件系统镜像烧录到 mbr.bin 中设置的位置。

7 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 JZ4770 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。

7.1 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 及 CONFIG_MTD_NAND_JZ4770 为 y。

JZ4770 集成了硬件 4-bit 、8-bit、12-bit、16-bit、20-bit 及 24-bit BCH ECC 算法，分别可纠正最多 8191-bit 内的 4-bit 、8-bit、12-bit、16-bit、20-bit 及 24-bit 错误。在驱动中，我们让每 512-bytes 做一次校验。

另外，强烈建议对于 JZ4770，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。

JZ4770 支持 NAND 多片选，片选 CS1_N、CS2_N、CS3_N、CS4_N、CS5_N 和 CS6_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 、 CONFIG_MTD_NAND_CS4 、 CONFIG_MTD_NAND_CS5 或 CONFIG_MTD_NAND_CS6=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 和 8KB page size 的 NAND。

7.2 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 UBI & UBIFS 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种文件系统。

7.3 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 jz4770_nand_ubi.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 UBIFS 分区不需要使用它，将 mtdblock_jz_invalid 设为 1，那么在 mount UBIFS 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 jz4770 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 0	0	7	4MB	MTD 分区 0 (bootloader)	不使用	不使用	不使用
MTD 1	8	15	4MB	MTD 分区 1 (kernel)	不使用	不使用	不使用

MTD 2	16	1023	504MB	MTD 分区 2 (rootfs)	不使用	不使用	不使用
MTD 3	1024	2047	512MB	MTD 分区 3 (data)	使用	不使用	不使用
MTD 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/jz4770_nand_ubi.c 中，如下所示：

```
static struct mtd_partition partition_info[] = {
    {name:"NAND BOOT partition",
     offset:0 * 0x100000,
     real_size:4 * 0x100000,
     cpu_mode: 0,
     use_planes: 0,
     mtdblock_jz_invalid: 1},
    {name:"NAND KERNEL partition",
     offset:4 * 0x100000,
     real_size:4 * 0x100000,
     cpu_mode: 0,
     use_planes: 0,
     mtdblock_jz_invalid: 1},
    {name:"NAND ROOTFS partition",
     offset:8 * 0x100000,
     real_size:504 * 0x100000,
     cpu_mode: 0,
     use_planes: 0,
     mtdblock_jz_invalid: 1},
    {name:"NAND DATA partition",
     offset:512 * 0x100000,
     real_size:512 * 0x100000,
     cpu_mode: 0,
     use_planes: 1,
     mtdblock_jz_invalid: 1},
    {name:"NAND VFAT partition",
     offset:1024 * 0x100000,
     real_size:1024 * 0x100000,
     cpu_mode: 0,
     use_planes: 1,
     mtdblock_jz_invalid: 0},
   };
```

7.4 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：

1. 坏块管理
2. 损耗均衡
3. 逻辑到物理块地址的映射
4. Volume 信息存储
5. Device 信息

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

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。

注：以下描述均以 Samsung K9GAG08U0D（4KiB * 128 * 4096）为例。

7.4.1 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 00080000 "NAND BOOT partition"
  mtd1: 00400000 00080000 "NAND KERNEL partition"
  mtd2: 1f800000 00080000 "NAND ROOTFS partition"
```

```
mtd3: 20000000 00080000 "NAND DATA partition"
mtd4: 40000000 00080000 "NAND VFAT partition"
```

以 mtd3 为例来说明如何使用 UBI 和 UBIFS。

首先，格式化该分区：

```
# flash_eraseall /dev/mtd3
```

接着，将 mtd3 作为一个 ubi 设备，加载 ubi 驱动：

```
# modprobe ubi mtd=3
UBI: attaching mtd3 to ubi0
UBI: kmalloc instead of vmalloc is used.
UBI: physical eraseblock size:   524288 bytes (512 KiB)
UBI: logical eraseblock size:    516096 bytes
UBI: smallest flash I/O unit:    4096
UBI: VID header offset:          4096 (aligned 4096)
UBI: data offset:                8192
UBI: empty MTD device detected
UBI: create volume table (copy #1)
UBI: create volume table (copy #2)
UBI: attached mtd3 to ubi0
UBI: MTD device name:            "NAND VFAT partition"
UBI: MTD device size:            512 MiB
UBI: number of good PEBs:        1024
UBI: number of bad PEBs:         0
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:             1010
UBI: total number of reserved PEBs: 14
UBI: number of PEBs reserved for bad PEB handling: 10
UBI: max/mean erase counter: 0/0
UBI: image sequence number: 0
UBI: background thread "ubi_bgt0d" started, PID 66
```

这里要注意的是蓝色标记的部分。表明这个 UBI 设备总共有 1024 个可用物理块，保留 14 个物理块。在这个 UBI 设备上建立的所有卷的总大小为：

1010*logical eraseblock size(在这里为 516096bytes)　即

1010*516096 /1024 /1024 = 497.109375 MiB 取整数部分，即 496MiB

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，名字分别为”ubifs”和”vfat”：

```
# ubimkvol /dev/ubi0 -s 200MiB  -N ubifs
# ubimkvol /dev/ubi0 -s 200MiB  -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: mounted UBI device 0, volume 0, name "ubifs"
UBIFS: file system size:   205406208 bytes (200592 KiB, 195 MiB, 398 LEBs)
UBIFS: journal size:       10321920 bytes (10080 KiB, 9 MiB, 20 LEBs)
UBIFS: media format:       w4/r0 (latest is w4/r0)
UBIFS: default compressor: lzo
UBIFS: reserved for root:  4952683 bytes (4836 KiB)
```

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。

7.4.2 UBI Block 设备

这里假设你已经按照 module 方式编译好了 ubi 和 ubi 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/mtd3                -- 格式化 MTD 分区
# modprobe ubi mtd=3                      -- 加载 UBI 驱动
# ubimkvol /dev/ubi0 -s 200MiB  -N vfat   -- 创建 UBI 卷 0
# ubimkvol /dev/ubi0 -s 200MiB  -N ext2   -- 创建 UBI 卷 1
# rmmod ubi                               -- 卸载 UBI 驱动
# modprobe ubi mtd=3                      -- 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=3
# modprobe ubiblk
# mount –t vfat /dev/ubiblock0 /mnt/ubiblock0
# mount –t ext2 /dev/ubiblock1 /mnt/ubiblock1
```

7.4.3 使用 UBIFS 做根文件系统

制作根文件系统镜像

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

针对以下 MTD 分区：

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00400000 00080000 "NAND BOOT partition"
mtd1: 00400000 00080000 "NAND KERNEL partition"
mtd2: 7f800000 00080000 "NAND UBI partition"
```

我们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，假设 ubifs 卷占用 504MiB，vfat 卷占用 1.5GiB。具体步骤如下：

首先准备 UBI 工具：
在 fs/ubifs/mkfs.ubifs/路径下编译 mkfs.ubifs 工具，
在 drivers/mtd/mtd-utils/ubi-utils/new-utils/路径下编译在 PC 机和开发板上使用的工具，具体操作可参考其中的 readme。

使用 mkfs.ubifs 工具制作 ubifs 卷镜像 ubifs.img0：

```
$ export LD_LIBRARY_PATH=`pwd`/lzo/lib:$LD_LIBRARY_PATH   //在 PC 上操作
$ ./mkfs.ubifs  -r /nfsroot/root26/ -m 4096 -e 516096 -c 1008 -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）
-m min. I/O unit

接着在 Linux 系统制作 vfat 卷镜像 vfat.img0， 此处镜像大小为 30M：

```
$ dd if=/dev/zero of=vfat.img0 bs=1M count=30 //以下操作在 PC 机上完成
$ losetup /dev/loop0 vfat.img0                //需要 root 权限
$ mkfs.vfat /dev/loop0
$ mount -t vfat /dev/loop0 /mnt/udisk         //optional
$ cp /mnt/mmc/* /mnt/udisk/                   //optional
$ umount  /mnt/udisk                          //optional
$ losetup -d /dev/loop0
```

在 PC 机上使用 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
```

然后在 PC 机用 ubinize 通过将 ubifs.img 和 vfat.img 生成一个镜像 ubi.img。准备一个配置文件 ubi.ini，内容如下：

```
$ cat ubi.ini
[ubifs]
mode=ubi
image=ubifs.img
vol_id=0
vol_size=488MiB
vol_type=dynamic
vol_name=ubifs
vol_alignment=1
vol_flag=autoresize

[vfat]
mode=ubi
image=vfat.img
vol_id=1
vol_size=1488MiB
vol_type=dynamic
vol_name=vfat
  vol_alignment=1
  vol_flag=autoresize
```

注：由于 UBI 会占用少量空间存储管理信息，故 vol_size 要小于实际空间大小。vol_size 的具体值按如下方式计算：

```
vol_size = ((mtdsize / blocksize) * 99% - 4 ) * (ppb - 2) * pagesize
```

其中，

```
mtdsize      -   rootfs 所在的 mtd 分区大小
blocksize    -   nand flash block size
ppb          -   page num per block
pagesize     -   nand flash page size
```

UBI 要占用 PEB 总数的 1%用于 bad block handling，要保留 2 个 PEB 用于 layout volume，保留 1 个 PEB 用于 wear-leveling，保留 1 个 PEB 用于 atomic LEB change，而且每个 PEB 内要保留起始的 2 个 page 用于存储 ubi header。

然后运行：

```
$ ubinize -o ubi.img ubi.ini -p 524288 -m 4096
```

-p 物理块容量（字节为单位）

至此，镜像文件 ubi.img 已经制作完成。

注：以上制作 ubifs image 的过程比较繁琐，我们已经将其整理成一个 shell 脚本，位于 drivers/mtd/mtd-utils/ubi-utils/new-utils 路径下，请将其放在 linux-2.6.31.3/内核目录下运行：

```
ylyuan@ubuntu:~/linux-2.6.31.3$ ./mkubifs
Usage:
    mkubifs /nfsroot/root26 pagesize ppb leb_num ubifs.img

    pagesize:    min I/O size, same as pagesize
    ppb:         page num per block
    leb_num:     logical erase block number
    ubifs.img:   output file name

Example:
    mkubifs /nfsroot/root26 4096 128 1008 ubifs.img
```

烧录、启动

使用 USBBoot 烧录工具以-n 方式将制作好的 ubi.img 烧录到 nand flash 的 mtd2 上：

```
USBBoot :> nprog 2048 ubi.img 0 0 -n
```

配置 u-boot 启动参数：

```
setenv bootargs mem=256M console=ttyS1,57600n8 ubi.mtd=2 root=ubi0:ubifs rootfstype=ubifs rw
```

配置内核，将 UBI、UBI Block 设备及 UBIFS 编译进 kernel，并将 uImage 烧录至 mtd1。

系统起来后，将能够在/dev 下看到以下设备：

```
# ls -l /dev/ubi*
crw-rw----    1 root     root      253,   0 Oct  1 15:20 /dev/ubi0
crw-rw----    1 root     root      253,   1 Oct  1 15:20 /dev/ubi0_0
crw-rw----    1 root     root      253,   2 Oct  1 15:20 /dev/ubi0_1
crw-rw----    1 root     root       10,  63 Oct  1 15:20 /dev/ubi_ctrl
brw-r-----    1 root     root      254,   0 Oct  1 15:20 /dev/ubiblock0
brw-r-----    1 root     root      254,   1 Oct  1 15:20 /dev/ubiblock1
```

其中，/dev/ubiblock0 和/dev/ubiblock0 分别对应 ubi 设备中的 ubifs 卷和 vfat 卷。

接着，进行以下操作：

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

我们能够看到已 mount 的 ubiblock1：

```
# df -h
Filesystem                Size      Used Available Use% Mounted on
ubi0:ubifs              444.4M     12.8M    431.6M   3% /
tmpfs                   124.3M     44.0K    124.3M   0% /dev
/dev/ubiblock1            1.4G      4.0K      1.4G   0% /mnt/ubiblock1
```

开机优化

由于系统启动时会扫描 rootfs 所在的整个 ubi device，所以按照以上描述在实现一个较大的 ubi device 并在其中建立两个 volume 的方法会使得开机速度较慢。
为了提高开机速度，我们可以实现两个 ubi device，其中一个较小的 ubi 设备建立 rootfs 卷，另外一个较大的 ubi 设备建立 vfat 卷。以下只解释与上述方法的不同之处。

假设有以下 MTD 分区：

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00400000 00080000 "NAND BOOT partition"
mtd1: 00400000 00080000 "NAND KERNEL partition"
mtd2: 1f800000 00080000 "NAND UBIFS partition"
mtd3: 60000000 00080000 "NAND VFAT partition"
```

准备两个*.ini 配置文件：

```
  $ cat ubifs.ini
  [ubifs]
```

```
mode=ubi
image=ubifs.img
vol_id=0
vol_size=488MiB
vol_type=dynamic
vol_name=ubifs
vol_alignment=1
vol_flag=autoresize

$ cat vfat.ini
[vfat]
mode=ubi
image=vfat.img
vol_id=2
vol_size=1488MiB
vol_type=dynamic
vol_name=vfat
vol_alignment=1
vol_flag=autoresize
```

注：为了能够使用 UBI Block 设备，请指定不同的 vol_id。

分别制作 ubi 镜像：

```
$ ubinize -o ubi.img ubifs.ini -p 524288 -m 4096
$ ubinize -o fat.img vfat.ini -p 524288 -m 4096
```

然后，使用烧录工具将 ubi.img 和 fat.img 分别烧录至 mtd2 和 mtd3：

```
USBBoot :> nprog 2048 ubi.img 0 0 -n
USBBoot :> nprog 131072 fat.img 0 0 -n
```

系统启动后，由于 vfat 卷所在的 ubi 设备尚未加载，只能看到以下设备：

```
# ls -l /dev/ubi*
crw-rw----    1 root     root      253,   0 Oct  1 12:24 /dev/ubi0
crw-rw----    1 root     root      253,   1 Oct  1 12:24 /dev/ubi0_0
crw-rw----    1 root     root       10,  63 Oct  1 12:24 /dev/ubi_ctrl
brw-r-----    1 root     root      254,   0 Oct  1 12:24 /dev/ubiblock0
```

加载 vfat 卷所在的 ubi 设备：

```
# ubiattach /dev/ubi_ctrl -m 3
UBI: attaching mtd3 to ubi1
UBI: kmalloc instead of vmalloc is used.
UBI: physical eraseblock size:   524288 bytes (512 KiB)
UBI: logical eraseblock size:    516096 bytes
```

```
UBI: smallest flash I/O unit:    4096
UBI: VID header offset:          4096 (aligned 4096)
UBI: data offset:                8192
UBI: attached mtd3 to ubi1
UBI: MTD device name:            "NAND VFAT partition"
UBI: MTD device size:            1536 MiB
UBI: number of good PEBs:        3072
UBI: number of bad PEBs:         0
UBI: max. allowed volumes:       128
UBI: wear-leveling threshold:    4096
UBI: number of internal volumes: 1
UBI: number of user volumes:     1
UBI: available PEBs:             62
UBI: total number of reserved PEBs: 3010
UBI: number of PEBs reserved for bad PEB handling: 30
UBI: max/mean erase counter: 5/1
UBI: image sequence number: 0
UBI: background thread "ubi_bgt1d" started, PID 74
UBI device number 1, total 3072 LEBs (1585446912 bytes, 1.5 GiB), available 62 LEBs (31997952
bytes, 30.5 MiB), LEB size 516096 bytes (504.0 KiB)
```

此时，就能够看到与此设备相关的设备文件：

```
# ls -l /dev/ubi*
crw-rw----    1 root     root      253,   0 Oct  1 12:24 /dev/ubi0
crw-rw----    1 root     root      253,   1 Oct  1 12:24 /dev/ubi0_0
crw-rw----    1 root     root      252,   0 Oct  1 12:24 /dev/ubi1
crw-rw----    1 root     root      252,   3 Oct  1 12:24 /dev/ubi1_2
crw-rw----    1 root     root       10,  63 Oct  1 12:24 /dev/ubi_ctrl
brw-r-----    1 root     root      254,   0 Oct  1 12:24 /dev/ubiblock0
brw-r-----    1 root     root      254,   2 Oct  1 12:24 /dev/ubiblock2
```

接着，进行如下操作：

```
# mkfs.vfat /dev/ubiblock2
# mount -t vfat /dev/ubiblock2 /mnt/ubiblock2/
# df -h
Filesystem                Size      Used Available Use% Mounted on
ubi0:ubifs              444.4M     13.0M    431.4M   3% /
tmpfs                   124.3M     48.0K    124.3M   0% /dev
/dev/ubiblock2            1.4G      4.0K      1.4G   0% /mnt/ubiblock2
```

此时可以向/mnt/ubiblock2 中拷贝文件进行测试。

当然，可以不烧录 fat.img，直接创建 vfat 卷，如下：

```
# flash_eraseall /dev/mtd3                    --擦除 mtd3
# ubiattach /dev/ubi_ctrl -m 3                --attach mtd3
# ubimkvol /dev/ubi1 -s 1GiB -N vfat -n 2     --创建 vfat 卷
# ubidetach /dev/ubi_ctrl -m 3                --detach mtd3
# ubiattach /dev/ubi_ctrl -m 3                --再次 attach mtd3
# mkfs.vfat /dev/ubiblock2                    --格式化为 vfat
# mount -t vfat /dev/ubiblock2 /mnt/ubiblock2/
```

8 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。

8.1 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，DRAM 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。

内核配置：

为了支持休眠唤配，你需要在 make menuconfig(或 make xconfig 等)的 Power management options 菜单中选择 Power Management support 和 Suspend to RAM and standby。默认的开发板配置是选择了这两项的

选择了上述配置后，您可以执行以下命令使系统进入休眠

```
echo mem >/sys/power/state
```

一般情况下休眠都是由用户态的管理程序发起的，因此我们不介绍内核的休眠入口函数的，有兴趣的用户可以自行研究内核代码，相关的代码位于 kernel/power/以及 drivers/base/power 目录下。

代码说明：

1、与 CPU 本身相关的代码是位于代码树 arch/mips/jz47xx 目录下的 pm.c，主要逻辑位于 jz_pm_do_sleep 函数中，因为这里主要是 CPU 相关的代码，建议用户不要自行修改，但有一个例外是 RTC 的唤醒功能，在我们发布的内核中只是一个参考实现，用户可能需要修改以符合自己的需求

2、与板级相关的代码位于 arch/mips/jz47xx/board-yyy.c 中，这里主要的代码位于 config_gpio_on_sleep 中，当 CPU 进入休眠时，所有的 GPIO 都必须处于一个明确的状态，不能是悬浮的，我们的 GPIO 目前有**输入带上拉(input_pull)，输入不带上拉(input_nopull)，输出高电平，输出低电平四种状态**，在 config_gpio_on_sleep 中，您只需配置输入带上拉、输出高电平、输出低电平以及哪些 pin 脚不能操作，**程序会根据您的配置计算哪些 pin 是需要输入不带上拉的，具体请参见 config_gpio_on_sleep 的代码。**

关于 no_operation 的 pin: 休眠时，有些 pin 可能是需要保持之前的状态的，比如说如果您不想让 SD 卡进入休眠，那么休眠时就不能操作 SD 卡相关的 GPIO，否则 3.3V 等不供电的话，SD 卡会进入一种不确定的状态，导致系统唤醒后 SD 卡不可用，在这种情况下，您需要将 SD 卡相关的 pin 记入 no_operation 中

3、与各个模块相关的 suspend/resume 方法，比如音频模块的休眠唤醒，SD 卡驱动的休眠唤醒等，都是在各自的内核模块处理的

建议：在各个内核模块中，你可能也会拉高或是拉低某些 pin 以关闭某些电源等，虽然您在这些模块里操作了这些 pin，我们仍然建议您在 board-yyy.c 的 config_gpio_on_sleep 中对这些 pin 再进行

一次正确的设置，你应该将 config_gpio_on_sleep 当作一个休眠时集中管理 gpio 的地方，**但这两者不是冲突的，不是说某个 GPIO 在 config_gpio_on_sleep 中设置了，在模块的 suspend/resume 中就不需要操作**

决定哪些 GPIO pin 可以唤醒系统：

您还可以在在 arch/mips/jz47xx/board-yyy.c 设置哪些 GPIO pin 可用作唤醒 pin，这是通过 wakeup_key 数据实现的，我们的默认实现里只支持了上升沿及下降沿的触发方式，如果你需要电平触发，你需要自行进行修改。

使用电源管理芯片：

您可以使用电源管理芯片进行电源管理，因为可能和具体使用的芯片相关，在此我们不进行描述

9 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
SDIO	Atheros AR6102 及 AR6302	Linux-2.6.31.3	AR6102: 10Mbps AR6302: 截止文档发布日未进行测试	Atheros

注：我们发布的内核的 SDIO 驱动对 boardcom 的 wifi 支持不是很好，因为他们的驱动对 DMA 不友好，buffer 起始地址以及长度都不对齐，而且还可能有从 stack 上分配内核的嫌疑，这会给 DMA 传输以及 cache 刷新带来影响。

Atheros 正在试图将他们的驱动加入到主干内核，您可以从 linux 官方代码树中的 drivers/staging/目录下获取他们的最新驱动，可以从以下页面下载到最新的 firmware:

http://wireless.kernel.org/en/users/Drivers/ath6kl

要注意的是这个驱动只支持 AR6003 系列，比如 AR6302，如果需要其他系列的驱动，请联系 Atheros 的代码

9.1 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"和"Common routines for IEEE802.11 drivers"。

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。

9.2 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。

9.2.1 加载 VT6656 驱动：

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

9.3 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
```

9.4 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
```

9.5 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
```