君正®

JZ4780_Linux3.0.8_开发指南

Date: Sep. 2013

北京君正集成电路股份有限公司
Ingenic Semiconductor Co., Ltd.

君正®

JZ4780_Linux3.0.8_开发指南

Release history

Date	Revision	Change
July. 2013	1.01	First release
Sep. 2013	1.02	1、添加了制作 Ramdisk 的详细过程，以及配置 Linux 内核支持 Ramdisk. 2、删除在制作文件系统拷贝时，误加的/符号 3、更新 git 下载网址及公司地址

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.

北京君正集成电路股份有限公司
地址：北京市海淀区东北旺西路中关村软件园二期君正总部大楼
电话:(86-10)56345000
传真:(86-10)56345001
Http: //www.ingenic.cn

目录

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 3.x 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 3.x 的 Grus 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的源码；
2） 基于君正 4780 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） USBBurn Tool 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 mipseltools-gcc412-glibc261 工具链；
5） 引导程序 U-Boot 源码：君正提供 uboot 的源码和补丁；
6） Linux 3.x 核心源码：君正提供 linux 3.x 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

2 建立开发环境

在开始 Linux 开发之前，您需要准备一台 PC 机来下载 uboot、kernel 等源码，安装交叉编译器，启动 TFTP 和 NFS 服务等。

我们使用的测试主机安装了 Ubuntu 12.04。

2.1 下载源码

目前，Linux 的 ingenic Git 仓库包括 binaries，documents，sources 三大部分，该仓库涵盖了 kernel，bootloader，rootfs，applications，工具链，二进制文件，常用工具，文档等。

1) 下载 repo

a) `$ mkdir <your work dir>`

b) `$ cd <your work dir>`

c) `$ wget http://git.ingenic.cn:8082/bj/repo`

d) `$ chmod +x repo`

2) 下载 Linux 的 ingenic Git 仓库

公共下载：

```
$./repo init -u http://git.ingenic.cn/linux/manifest -m outside.xml
$./repo sync
```

2.2 设置交叉编译环境

在君正处理器平台上进行 Linux 3.x 内核开发之前，首先需要安装好 MIPS 的交叉编译工具链。针对 Linux 3.x 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，Linux 的 ingenic Git 仓库 binaries/toolchain/mipseltools-gcc412/glibc261 目录下。

安装交叉编译工具链只需设置下环境变量即可，有两种方法：

1) export 设置 PATH 环境变量（每次打开终端均需设置，比较麻烦）

```
$ export PATH=仓库路径/ binaries/toolchain/mipseltools-gcc412-glibc261
/bin:$PATH
```

2) 在~目录下.bashrc 文件中设置（只需设置一次，但设完之后需重启终端）

```
$ export PATH=仓库路径/ binaries/toolchain/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。

2.3 启动 TFTP 和 NFS 服务

2.3.1 设置 TFTP 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：

1) 安装 tftp 的服务端和客户端：

```
sudo apt-get install tftpd-hpa ； 安装 tftp 服务端
sudo apt-get install tftp-hpa ； 安装 tftp 客户端
```

2) 安装 xinetd：

```
sudo apt-get install xinetd
```

3) 安装完 xinetd 应该会建立目录/etc/xinetd.d，在此目录中添加文件：tftpd，主要是设置 TFTP 服务器的根目录和开启服务。文件内容如下：™

```
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
log_on_success     += PID HOST DURATION
log_on_failure     += HOST
}
```

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。

2.3.2 设置 NFS 服务

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：

1) 安装 nfs：

```
sudo apt-get install nfs-kernel-server
```

2) 创建和编辑文件/etc/exports 为：

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

3) 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
```

这样 nfs 服务也就设置好了。

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/EXT4/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 uboot-grus 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

3.1 U-Boot 源码结构

uboot-grus 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。君正系列的 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
- jz4780.c　　JZ4780 相关代码，如系统 timer、PLL 的初始化等
- jz4780_ddr.c　　DDR 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 grus.h)
- jz_serial.c　　串行口 UART 驱动程序
- jz4780_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

- jz4780_nand.c JZ4780 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 EXT4、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/jz4780.h JZ4780 相关的头文件定义
- include/configs/grus.h 基于 JZ4780 GRUS 开发平台的配置文件

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

以 MSC 启动为例，对于 JZ4780 grus 平台其默认的 U-Boot 启动代码及相关的文件主要位于 msc_spl/board/grus 目录下，主要文件包括：

- start.S grus 平台 MIPS 内核启动代码
- u-boot.lds grus 平台的代码链接脚本文件
- config.mk grus 平台的地址分配文件
- cpu.c grus 平台 CPU 其它相关代码，如 TLB 和 CACHE 操作等
- jz4780.c JZ4780 grus 平台相关代码，如系统 timer、PLL 的初始化等
- jz4780_ddr.c DDR 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 grus.h)
- jz_serial.c 串行口 UART 驱动程序
- jz_ddr3_init.c DDR3 控制器初始化代码，一般情况下您不需要改动这里的代码，DDR3 参数配置在相应的板级配置文件中包含(如 grus.h)
- msc_boot_jz4780.c grus 平台 U-Boot 的初始化代码

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。

3.2.1 JZ4780 grus 平台，Nand Flash 启动：

对于 JZ4780，编译之前需要在 include/configs/grus.h 中，确认使用 Nand 型号并设置好相关 ECC 参数：

```
#define CFG_NAND_ECC_POS            24
```

另外，在 include/asm-mips/jz_mem_nand_configs/ 目录下对应 Nand Flash 型号的头文件中，确认配置参数：

如若 grus.h 中指定了 NAND_K9GAG08U0D.h，确认配置参数：

```
#define CFG_NAND_BCH_BIT            8
#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      0
```

然后，确认你的 DDR3 的大小，并在 include/asm/ jz_mem_nand_configs/ 下的对应的 DDR 型号文件中配置，如在 DDR3_H5TQ2G63BFR.h 中设置

最后，

```
$ make distclean
$ make grus_nand_config
$ make
```

编译后生成 u-boot-nand.bin。

3.2.2 JZ4780 GRUS 平台，EMMC 启动

```
$ make distclean
$ make grus_msc_config
$ make
```

编译后生成 mbr-uboot-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 USBBurn Tool 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：

USBBurn Tool 工具烧录 u-boot-nand.bin

USBBurn Tool 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 USBBurn Tool 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，运行应用程序 USBBurn Tool.exe 并开始烧录。

在烧录前要确认 USBBurn Tool 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：help nand

“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。

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。

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：

```
GRUS# help tftpboot
GRUS# 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 192.168.3.56，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, grus 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）

```
GRUS# setenv ipaddr 192.168.2.84
GRUS# setenv serverip 192.168.3.56
GRUS# setenv ethaddr 00:2a:c6:2c:ab:f1
GRUS# askenv bootcmd
bootcmd: tftpboot;bootm
GRUS# setenv bootargs mem=256M console=ttyS3,57600n8 ip=dhcp
nfsroot=192.168.3.56:/nfsroot/root26 rw
GRUS # setenv bootfile uImage
GRUS # saveenv
GRUS# boot
```

“saveenv”命令将保存当前配置到 Flash 上，下次重启系统时就不需要再配置了。“boot”命令是启动运行环境变量“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 192.168.3.56 服务器上的/nfsroot/root26 为根文件系统。

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 -A mips -O linux -T kernel -C gzip \
-a 0x80010000 -e 0x804ecc30 -n "Linux-3.x" \
    -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-3.X
Created:      Tue Jan  15 14:49:37 2013
Image Type:   MIPS Linux Kernel Image (gzip compressed)
Data Size:    2994480 Bytes = 2924.30 kB = 2.86 MB
Load Address: 0x80010000
Entry Point:  0x804ecc30
```

4 Linux 内核和驱动

君正移植了 Linux 3.x 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 JZ4780 的 grus 平台相关的君正 Linux 3.x 内核和驱动程序。

4.1 Linux 源码的目录结构

Linux-3.x 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
- xburst/soc-4780/：JZ4780 处理器相关目录和文件
  - board/grus/： JZ4780 处理器 GRUS 平台通用处理文件
  - common/： JZ4780 处理器通用处理文件
  - include /： 各种头文件，包含 JZ4780 相关的定义和开发板的配置文件等

boot/compressed/：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
- grus_linux_defconfig : build for grus_linux

include/asm-generic/：MIPS 体系相关各种头文件
sound
- oss/jzsound/：君正处理器 OSS 音频驱动
- devices/codecs：编解码器驱动
- interface/： 常用接口驱动

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
- cpufreq： cpufreq 驱动
- input/： 输入设备驱动

keyboard - mouse - touchscreen 等各种输入设备驱动
mmc/： MMC/SD 卡驱动
mtd/： MTD 设备驱动
ubi/： UBI 驱动
net/： 网络设备驱动
tty/serial/： UART 驱动
spi/： 同步串行接口驱动
usb/host： USB host 驱动
usb/otg： USB otg 驱动
usb/musb： USB musb 驱动
musb*.c
usb/gadget： USB device gadget 驱动
file_storage.c
video/jz4780_fb： LCD framebuffer 驱动
misc/jz_cim： Camera 驱动

4.2 配置和编译 Linux

首先，选择目标板的配置：

```
$ make grus_linux_defconfig
```
(JZ4780 grus 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-3.x/arch/mips/boot/compressed/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=256M@0x0 mem=256M@0x30000000 设置内存容量大小

console=ttyS3,57600n8 设置控制台输出为串口 ttyS3，波特率为 57600，8 个数据位，无校验位

root= /dev/ndsystem rw 设置根文件系统为/dev/ndsystem，具有读写权限

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：

4.4.1 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。(注：用电源供电，不能用 USB 供电)

1) 测试音频播放和录音

MP3 播放测试：

```
# ./mplayer  test.mp3
```

录音和播放录音测试：使用 sndkit

```
# sndkit -R -t wav -o 30 -w test.wav -L    //line in 录音
# sndkit -R -t wav -o 30 -w test.wav      // mic 录音
# sndkit -P -t wav -o 30 -p test.wav -L    // 耳机播放
```

2) 测试视频播放

```
# ./mplayer  binhe.264
```

4.4.2 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：

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

4.4.3 测试 USB

1) 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，这时会自动挂载 USB 设备，提示如：

---------automount /dev/sda1 to /mnt/usb_sda1----------

参考提示 ls /mnt/usb_sdb1，看是否出现您 USB 设备里的文件，如出现，则测试 OK；

直接拔掉 USB 设备，会自动卸载此 USB 设备，出现提示，如：

--------autoumount /dev/sda1 from /mnt/usb_sda1 --------

2) 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：

make menuconfig，确保 File-backed Storage Gadget 选中。

Device Drivers --->

  [*] USB support --->

    <*> USB Gadget Support --->

<*> USB Gadget Drivers (File-backed Storage Gadget) --->
(X) File-backed Storage Gadget

```
# mkfs.vfat /dev/ndmisc
# echo /dev/ndmisc > /sys/devices/platform/jz4780-dwc2/dwc2/gadget
/gadget-lun0/file
# echo 0 > /sys/devices/platform/jz4780-dwc2/dwc2/gadget/gadget-lun0
/file
```

a) 测试 device 模式：

插入 usb 数据线，在开发板终端输入：

```
# mkfs.vfat /dev/ndmisc
# echo /dev/ndmisc > /sys/devices/platform/jz4780-dwc2/dwc2/gadget
/gadget-lun0/file
```

看是否挂在到电脑上，若挂载上，以主机为 linux 系统为例，如挂载路径为/media/512D-F658,

在主机端:`touch 1.txt; sync; sudo umount /media/512D-F658`

回到开发板终端输入：

```
# echo 0 > /sys/devices/platform/jz4780-dwc2/dwc2/gadget/gadget-lun0
/file
# mount/dev/ndmisc   /mnt/; cd /mnt; ls 1.txt
```

若看到 1.txt，说明 otg 作为 device 测试成功，输入 umount /mnt/ 卸载，拔掉 usb 数据线。

b) 测试 host 模式：

将 USB 插入插入 OTG 线(A 型)USB 口，将 OTG 线 usb 口另一端插入开发板 usb 接口。

会出现自动挂载提示，如：

---------automount /dev/sda1 to /mnt/usb_sda1------------

照着提示 ls /mnt/usb_sda1，看看是否出现您 u 盘里的文件；若有，说明 otg 作为 host 端测试 OK；

拔掉 OTG 线，也会出现自动卸载提示，如：

--------autoumount /dev/sda1 from /mnt/usb_sda1 --------

4.4.4 测试电源管理

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 wakeup 键之后唤醒。

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

4.5 Linux 3.x 音频驱动

Linux 3.x 的音频驱动包括 ALSA 和 OSS 两种架构，这里简要介绍下 OSS 驱动。

OSS 音频驱动在 linux-3.x/sound/oss/jzsound/devices/codecs 目录下。目前提供了 jz4780 内部 codec 的驱动，相关代码是 jz4780_codec.c、jz4780_codec.h 、jz4780_route_conf.c 等。

其他相关代码有 linux-3.x/sound/oss/jzsound/devices/ 目录下的 xb47xx_i2s.c 、xb47xx_pcm.c 等

当使用 OSS audio 驱动时，在 make menuconfig 弹出的窗口中进行配置

Device Drivers -□

<*> Scound card support
<*> Open Sound System(DEPRECATED)
<*> Open Sound System of xburst
[*] jz On-Chip I2S driver
[*] xburst internal codec
[*] JZ4780 internal codec

然后，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）下载其源码包。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-2.6.1）。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 3.x 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。

5.3.1 编译 udev 的步骤

下载 udev 源码
cd udev-117/
make CROSS_COMPILE=mipsel-linux- DESTDIR=/nfsroot/root26
make CROSS_COMPILE=mipsel-linux- DESTDIR=/nfsroot/root26 install

5.3.2 根文件系统配置

/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/ts c 10 16
/bin/mknod /dev/rtc c 10 135

# comment next when you use alsa
/bin/mknod /dev/audio c 14 4

echo "Starting udevd ..."
/usr/jz_mips/bin/udevd --daemon
/usr/jz_mips/bin/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 3.x 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 -H -newc | 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

[*] Support initial ramdisks compressed using gzip

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

在上面制作的 rootfs 的基础上：

1）制作一个大小为 8M 的镜像文件

```
$cd ~
$dd if=/dev/zero of=initrd.img bs=1k count=8192
```

(initrd.img 为 8M 的大小)

2）格式化这个镜像文件为 ext2

```
$mkfs.ext2 -F initrd.img
```

3）在 mount 目录下创建 initrd 目录作为挂载点

```
$sudo mkdir /mnt/initrd
```

4）将创建的镜像文件挂载到/mnt/initrd 下

注意这里的 initrd.img 不要和 ubuntu 根目录下的 initrd.img 弄混了，同时 initrd.img 不能存放在 rootfs 目录中

```
$sudo mount -t ext2 -o loop initrd.img  /mnt/initrd
```

5)将我们的文件系统复制到 initrd.img 中,将测试好的文件系统的内容全部拷贝到/mnt/initrd 目录下面

```
$sudo cp /source/rootfs/*  /mnt/initrd -a
```

6）卸载 initrd

```
$sudo umount /mnt/initrd
```

7）压缩 initrd.img 为 initrd.img.gz 并拷贝到/tftpboot 目录下

```
$gzip --best -c initrd.img > initrd.img.gz
```

8）配置内核支持 ramdisk

Device Drivers

SCSI device support--->

<*>SCSI disk support

Block devices--->

<*>RAM disk support

(1)Default number of RAM disks

(8192)Default RAM disk size(kbytes) (修改为 8M)

General setup--->

[*]Initial RAM filesystem and RAM disk(initramfs/initrd) support

[*]System V IPC

重新编译内核。

6 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 JZ4780，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。

6.1 生成 u-boot-msc.bin 或 mbr-uboot-msc.bin 及烧录

6.1.1 SD 卡协议探测

对于 jz4780 系列请先确认使用的 eMMC 是遵循 sd 协议还是 mmc 协议即使用 sd_found() 还是 mmc_found()，根据卡的类型修改做相应的修改，默认我们使用的是 mmc 探测：

通过对函数 mmc_cmd()进行不同传参并由其返回值确定我们使用的是 sd 探测还是 mmc 探测，其代码如下所示：

```
resp = mmc_cmd(55, 0, 0x1, MSC_CMDAT_RESPONSE_R1);
if(resp[5] == 0x37){
    resp = mmc_cmd(41, 0x40ff8000, 0x3, MSC_CMDAT_RESPONSE_R3);
if(resp[5] == 0x3f){
    sd_found();
}else
    mmc_found();
}else
mmc_found();
```

以上代码在 msc_spl/ msc_boot_jz4780.c 中的使用。

6.1.2 u-boot-msc.bin 和 mbr-uboot-msc.bin 的主要区别

mbr-uboot-msc.bin 是在 u-boot-msc.bin 的基础上加上 512Byte 的 mbr。

如何生成 mbr-uboot-msc.bin 呢？以 grus 为例，打开主 Makefile, 在 grus_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 jz4780 系列再烧录时，u-boot-msc.bin 是烧录在第 1 个扇区，mbr-uboot-msc.bin 是烧录在第 0 个扇区。

若要使用 mbr-uboot-msc.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 grus.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_OFFSE
#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

6.2 内核配置及烧录

6.2.1 下载 linux-3.x 源码包

```
# make distclean
# make grus_linux_defconfig
# make xconfig (make menuconfig)
```

Device Driver

<*>(MMC/SD/SDIO card support

<*>Ingenic(XBurst) JZ4780 MMC/SD Card Controller(MSC) support

[*] MSCx

Z4780 MSCx function pins select (GPIO E, Data width 4 bit) --->

-----选择 1/4/8 bit 数据线以及 GPIO A or D or E

-----按照需求选择其控制器(MSCX)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

< >UFS file system support (read only) (NEW)

6.2.2 设置数据传输方式：

修改 driver/mmc/host/jz4780_mmc.c、jz4780_mmc.h。

jz4780_mmc.c : static void jzmmc_data_start() 函数中相关代码

```
#define        is_pio_mode(host)          (host->flags & (1 << JZMMC_USE_PIO))
if (is_pio_mode(host)) {
                pio_trans_start(host, data);
            } else {
                jzmmc_dma_start(host, data);
        }
```

6.2.3 烧录

修改烧录工具中的 ini/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
```

6.2.4 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=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

6.3 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/
> cp rootfs/* /mnt/ -rf
> sync
> umount /mnt/
```

6.4 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 JZ4780 集成了 NAND Flash 控制器，支持与 NAND Flash 的直接连接。同时，君正 Linux 3.0.8 内核也提供了对 NAND Flash 文件系统的支持，本节将介绍在 Linux 3.0.8 上如何构建基于 NAND Flash 的文件系统。

7.1 NAND Flash 驱动

编译配置：

Make menuconfig/xconfig

Device Drivers

<*> NAND support -→

[*] Support NAND Flash device on JZ4780

select real or fake nand driver for test (use real nand driver test) -→

(X) use real nand driver test //真实驱动

( ) use fake nand driver test //仿真驱动，不操作实际硬件

[ ] Support JZ4780 8bit sram setting

[*] Support JZ4780 function setting

-*- NAND_COMMON

[*] NAND_CS1 //片选，需要根据硬件原理图配置，一片 Nand 一般配置为 CS1

[ ] NAND_CS2

[ ] NAND_CS3

[ ] NAND_CS4

[ ] NAND_CS5

[ ] NAND_CS6

8 bit or 16 bit NAND bus width (8 bit) -→ //bus 宽度，一般选择 8bit

(X) 8 bit

( ) 16 bit

512 byte or 1024 byte NAND ecc size (1024 byte) -→ //每次做 ecc 校验的数据长度，一般选择 1024

( ) default

( ) 512 byte

(X) 1024 byte

ECC type (Select hardware BCH) -→ //一般选择 select hardware BCH

( ) Select software BCH

(X) Select hardware BCH

4 bit or 8 bit ... 64 bit BCH ecc (auto) -→ //根据 Nand 手册进行配置 (X) auto

( ) 4 bit

.....

( ) 64 bit

The Nand output driver (low driver) -→ //驱动能力选项，与 Nand 和 CPU 之间的距离成反比

(X) low driver

( ) normal driver

( ) high driver

[*] Use DMA mode //使用 DMA 模式传输，否则使用 CPU 模式传输

[ ] Use Toggle NAND

[ ] Mul Parts //Nand 管理策略选项，可不选

其他未提及选项，一律不选

7.2 NAND Flash 文件系统类型

支持 ext2/3/4 vfat 等文件系统，可根据需求格式化为所需文件系统类型。

Nand 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-3.x/arch/mips/xburst/soc-4780/board/grus/下的 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。

下面我们以在 jz4780 下使用 2GB 的 NAND Flash 为例来向读者介绍如何定义自己的分区：

MICRON_MT29F32G08CBACAWP NAND 参数：

```
Page size: 4096 Bytes
Block size: 1024 KB
Pages per block: 256
Row address cycles: 3
Total size: 4GB
Total blocks: 4096
```

详细参数定义在 kernel/drivers/nand/driver/core/raw_nand/nand_ids.c 下：

```
{"MICRON_MT29F32G08CBACAWP",    0x2C68,        0x00A94A04,     2,1,
10,   5,   15, 15,  100,  60,  200,    20,   100,   70,   0,     4096,
1024*1024,      224,   3,  100,4096,  1024,   24, 8,  0,  MICRON_NAND,
0x03,       0x02,       0x01,       0x00,          0x04},
```

NAND 分区表信息：

表7-1 GRUS 4G NAND Flash的主要分区表示例

分区	起始位置	分区大小	描述	是否使用 use_planes	是否使用 cpu_mode
ndxboot	0x100000	8 * 0x100000	uboot分区	0	dma

ndboot	8 * 0x100000	16 * 0x100000	内核分区	0	dma
ndsystem	64 * 0x100000	512 * 0x100000	系统分区	0	dma

NAND 分区定义于文件 arch/mips/xburst/soc-4780/board/grus/nand.c 中，如下所示：

```
//此配置需根据您的 nand 型号和实际需要配置。
#define ECCBIT 24
static struct platform_nand_partition partition_info[] = {
{
name:"ndxboot",
offset:0 * 0x100000LL,
size:8 * 0x100000LL,
mode:SPL_MANAGER,
eccbit:ECCBIT,
use_planes:ONE_PLANE,
part_attrib:PART_XBOOT,
ex_partition:{{0},{0},{0},{0}}
},
{
name:"ndboot",
offset:8 * 0x100000LL,
size:16 * 0x100000LL,
mode:DIRECT_MANAGER,
eccbit:ECCBIT,
use_planes:ONE_PLANE,
part_attrib:PART_KERNEL,
ex_partition:{{0},{0},{0},{0}}
},
{
name:"ndrecovery",
offset:24 * 0x100000LL,
size:16 * 0x100000LL,
mode:DIRECT_MANAGER,
eccbit:ECCBIT,
use_planes:ONE_PLANE,
part_attrib:PART_KERNEL,
ex_partition:{{0},{0},{0},{0}}
},
{
name:"ndapanic",
offset:40 * 0x100000LL,
size:4 * 0x100000LL,
```

```
mode:DIRECT_MANAGER,
eccbit:ECCBIT,
use_planes:ONE_PLANE,
part_attrib:PART_MISC,
ex_partition:{{0},{0},{0},{0}}
   },
{
name:"ndsystem",
offset:64 * 0x100000LL,
size:512 * 0x100000LL,
mode:ZONE_MANAGER,
eccbit:ECCBIT,
use_planes:ONE_PLANE,
part_attrib:PART_SYSTEM,
ex_partition:{{0},{0},{0},{0}}
},
{
name:"ndcache",
offset:576 * 0x100000LL,
size:128 * 0x100000LL,
mode:ZONE_MANAGER,
eccbit:ECCBIT,
use_planes:ONE_PLANE,
part_attrib:PART_MISC,
ex_partition:{{0},{0},{0},{0}}
},
{
name:"ndextern",
offset:704 * 0x100000LL,
size:3392 * 0x100000LL,
mode:ZONE_MANAGER,
eccbit:ECCBIT,
use_planes:ONE_PLANE,
part_attrib:PART_MISC,
ex_partition:
{
    {
    name:"nddata",
    offset:704 * 0x100000LL,
    size:1024 * 0x100000LL,
    },
    {
    name:"ndmisc",
```

```
        offset:1728 * 0x100000LL,
        size:2366 * 0x100000LL,
        }
    }
    }
    };
```

7.3 使用 ext4 做根文件系统

制作根文件系统镜像

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

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，通过电源管理控制器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。

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 Debug support 和 Suspend to RAM and standby。默认的开发板配置是选择了以下几个选项：

Power management options

[*] Suspend to RAM and standby

[*] Wake lock

[*] Wake lock stats

[*] Userspace wake locks

[*] Early suspend

[*] Run-time PM core functionality

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：

与 CPU 本身相关的代码是位于代码树 arch/mips/xburst/soc-4780/common/目录下的 pm.c，主要逻辑位于 jz4780_pm_enter 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 时钟的唤醒功能，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

与板级相关的代码位于 arch/mips/xburst/soc-4780/board/grus/中，这里主要的代码位于 pm.c 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 pm.c 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\pm.c 的代码。

关于 GCC_IGNORE 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 GCC_IGNORE 中

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/pm.c 中对这些 pin 再进行一次正确的设置，你应该将 pm.c 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 pm.c 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/xburst/soc-4780/board/pm.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：

表 9-1：wifi 支持的设备

Interface	Device	OS	Max Throughput	Productor
USB	VT6656	celinux040503 linux-3.0.8	3Mbit/s	VIA
SDIO	Atheros AR6102及 AR6302	Linux-3.0.8	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”，默认选择"Wirelesss Extensions sysfs files” 和”Common routines for IEEE802.11 drivers”。

Linux-3.0.8 的无线网络接口版本（WIRELESS_EXT）为 22。

9.2 wpa_supplicant，无线网络配置工具

查看是否有 wlan0 节点：busybox ifconfig -a

启动 wlan0 节点：busybox ifconfig wlan0 up

扫描网络：wlarm_android scan

显示扫描结果，可以获取加密方式：wlarm_android scanresults

例如：

扫描时获取的 ingenic_M 的信息如下：

SSID: "ingenic_M"

Mode: Managed RSSI: -50 dBm SNR: 0 dB noise: -92 dBm Flags: RSSI on-channel Channel: 3

BSSID: 00:21:91:EE:E4:08 Capability: ESS WEP ShortPre ShortSlot

Supported Rates: [1(b) 2(b) 5.5(b) 11(b) 6 9 12 18 24 36 48 54]

WPA:

multicast cipher: AES-CCMP

unicast ciphers(1): AES-CCMP

AKM Suites(1): WPA-PSK

No WPA Capabilities advertised

扫描时获取的 ingenic_Y 的信息如下：

SSID: "ingenic_Y"

Mode: Managed RSSI: -39 dBm SNR: 0 dB noise: -92 dBm Flags: RSSI on-channel Channel: 11

BSSID: 5C:63:BF:3B:95:2C Capability: ESS ShortPre ShortSlot

Supported Rates: [1(b) 2(b) 5.5(b) 11(b) 6 9 12 18 24 36 48 54]

其中 ingenic_Y 是没密码的，ingenic_M 是有密码的

没有密码,可直接用 wlarm_android 连接

```
wlarm_android join ingenic_Y     //ssid : ingenic_Y
```

有密码的用 wpa_supplicant 工具

```
wpa_supplicant -B -Dnl80211 -i wlan0 -c /etc/wpa_supplicant.conf
```

wpa_supplicant.conf 内容如下：

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
eapol_version=1
ap_scan=2
fast_reauth=1
network={
ssid="ingenic_M"
scan_ssid=1
```

key_mgmt=WPA-PSK
proto=WPA
pairwise=CCMP
group=CCMP
psk="password"
}

其中 key_mgmt 参数和上面扫描结果中的 AKM Suites 对应，pairwise、group 和上面的 multicast cipher、unicast ciphers 只写 CCMP 即可

ingenic_M 表示路由器名称，psk 表示密码

查看网络连接状态

wlarm_android status

如果连接上了，就会打印出扫描时打印的信息

连接上之后设置本地的 IP 信息：ifconfig wlan0 192.168.1.23 netmask 255.255.255.0 up

用 wget 命令可以访问到网络

10 以太网配置

10.1 以太网内核配置

JZ4780 grus 支持 DM9000 网卡。

10.2 配置 IP

busybox ifconfig –a

busybox ifconfig eth0 192.168.2.68 netmask 255.255.248.0 up

ping 192.168.2.67

若能 ping 通，说明以太网连接 OK