# 君正®

# X1000 软件开发手册

Version: 1.0
Date: 2016.03.28

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

君正@

# Linux X1000 软件开发手册

**Release history**

| Date | Revision | Revision History |
|---|---|---|
| 2016-03-28 | 1.0 | - First released |

## Disclaimer



**Ingenic Semiconductor Co., Ltd.**
**Ingenic Headquarters, East Bldg. 14, Courtyard #10, Xibeiwang East Road, Haidian Dist., Beijing, China, 100193**
**Tel: 86-10-56345000**
**Fax: 86-10-56345001**
**Http: //www.ingenic.cn**

# 1. 前言

## 1.1. 文档目的及背景

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 内核的配置方法和开发过程，,引导开发人员快速进行 Linux 开发。本文档为君正内核 3.10 版本开发文档，基于芯片 X1000，不针对具体开发板，文中如有涉及具体开发板型号，是为了说明方便。

在阅读该文档前，需要具备以下基本技能：

1.会使用 Linux 系统进行开发，最好是 ubuntu。

2.知道嵌入式开发基本流程。如 uboot，linux，文件系统制作等。

阅读该文档，会提供以下帮助：

1.帮助理解君正 BSP 基本组成。（uboot,linux,文件系统）

2.提供基于君正开发平台创建自己的应用程序方法。

3.提供应用程序访问驱动的基本测试用例。

# 2. 搭建开发环境

在发布 SDK 时，可以使用君正提供的开发平台，该平台包含了 uboot 源码，kernel 源码，交叉工具链和一些测试程序等，基于该平台，可以方便第三方库的添加，方便应用程序开发。详细参考文档《Manhattan platform 编译系统简介.pdf》完成开发环境搭建。

# 3. Uboot 配置和使用

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。

## 3.1. uboot 编译

在进行此步骤前，请确保已经正确配置好交叉编译环境。

针对不同开发板的配置，uboot 的编译配置也不相同，在发布的 uboot 中，编译配置由开发板型号[BOARD_NAME]，内核镜像格式[IMAGE_FMT]和启动方式[BOOT]组成，格式如下

[BOARD_NAME]_[IMAGE_FMT]_[BOOT]

具体配置在 uboot/boards.cfg 文件中，可以通过以下命令快速查看开发板支持的编译配置

$ cat boards.cfg | grep [BOARD_NAME]

根据以上方法，找到对应开发板的编译选项按照以下方式进行编译：

$ make distclean

$ make [BOARD_NAME]_[IMAGE_FMT]_[BOOT]

例如开发板 phoenix 的编译配置如下：

phoenix_v10_uImage_msc0　　支持 sd 卡启动 uImage 的配置

phoenix_v10_uImage_sfc_nor　　支持 nor flash 启动 uImage 的配置

$ make distclean

$ make phoenix_v10_uImage_msc0

例如开发板 halley2 的编译配置如下：

halley2_v10_uImage_sfc_nor支持 nor flahs 启动 uImage 的配置

$ make distclean

$ make halley2_v10_uImage_sfc_nor

编译完成后会在当前目录下生成 u-boot-with-spl.bin 文件。即最终烧录所需的 uboot 文件。

## 3.2. uboot 常用命令

打开调试串口，在 uboot 启动过程中，敲击任意按键，打断 uboot 引导镜像过程，进入 uboot shell 环境，uboot 常用命令如下：

“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。
“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。
“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。

sfc nor 命令：
:“sfcnor read” 从 spi nor flash 中读取数据到内存。
“sfcnor write” 从内存中写数据到 spi nor flash。
“sfcnor erase” spi nor flash 擦出。

Sd 卡命令：
“mmc read” 从 sd 卡中读取数据。
“mmc write” 写数据到 sd 卡。

### 3.3. uboot 通过 SPI flash 加载内核镜像

1. 一种方法，可以在 uboot 运行的时候，进入 uboot shell，修改 bootcmd 变量如下：
   # set bootcmd ‘sfcnor read 0x40000 0x300000 0x80800000 ;bootm 0x80800000’
   # saveenv
2. 另一种方法，修改 uboot 源码，include/configs/[BOARD_NAME].h
   例如修改 include/configs/halley2.h 文件中的
   #define CONFIG_BOOTCOMMAND "sfcnor read 0x40000 0x300000 0x80800000 ;bootm 0x80800000"

注意需要编译支持 Nor flash 启动的 uboot。

### 3.4. uboot 通过 tftp 加载内核镜像

1. PC 服务器端环境配置
   $ sudo apt-get install tftpd-hpa
   $ sudo service tftpd-hpa start
   在 /var/lib/tftpboot 下的文件都可以通过 TFTP 协议下载，可以通过 tftp 客户端程序测试 tftp 服务器是否可以访问。
   $ sudo apt-get install tftp-hpa #安装客户端程序
   $tftp 127.0.0.1
   tftp> get test.txt

2. 开发板端配置

进入 uboot 环境，根据网络情况设置参数，将需要的 uImage 放入上述服务器文件夹下执行以下命令将 uImage 下载到内存 0x80800000 位置，并启动 tftp 服务，根据实际的 ip 情况，设置 uboot 各个环境变量如下：

$set ipaddr 192.168.4.145

$set serverip 192.168.4.146

$tftp 0x80800000 uImage

$bootm 0x80800000

## 3.5. uboot 挂载文件系统

在进行开发时，可以将文件系统存放在不同的介质中，比如 nfs 网络文件系统，spi nor flash jffs2 文件系统， SD 卡 ext4 文件系统等等。针对不同的文件系统，uboot 需要向内核传递参数，即设置 bootargs 环境变量，在内核启动的时候，会根据参数去挂载相应的文件系统。

### 3.5.1. 挂载 Jffs2 文件系统

# set bootargs ‘console=ttyS2,115200n8 mem=31M@0x0
ip=off init=/linuxrc rootfstype=jffs2 root=/dev/mtdblock2 rw’
以上命令在同一行。
# saveenv

### 3.5.2. 挂载网络文件系统

#### 3.5.2.1. PC 端配置

挂载网络文件系统，需要在 PC 端安装 NFS 服务器，并且已经将其导出。以 ubuntu 为例，设置如下：

安装 nfs 服务器：

$ sudo apt-get install nfs-kernel-server

导出/home/user/nfs_root 文件夹

$ sudo vi /etc/exports

添加以下行：

/home/user/nfs_root *(rw,insecure,no_root_squash,no_subtree_check)

启动 nfs 服务

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

测试 nfs 服务

mount 127.0.0.1:/home/user/nfs_root/ /mnt

如果 mnt 下的文件和/home/user/nfs_root 目录下文件一致说明 nfs 服务器安装成功

#### 3.5.2.2. 开发板端配置

具体 ip 地址以实际开发环境为准。注意内核需要选中 nfs 网络文件系统支持，才可以通过网络挂载文件系统。

# set bootargs ‘console=ttyS2,115200n8 mem=31M@0x0
ip=192.168.4.145:192.168.4.1:192.168.4.1:255.255.255.0
nfsroot=192.168.4.146:/home/user/nfs_root rw’

以上命令在同一行。

# saveenv

# 4. Linux 内核驱动和应用

本章主要介绍内核的各个模块的驱动和相应的用户空间测试方法。通过阅读本章节，旨在对引导内核编译，对内核的各个驱动有基本的了解，对相应的测试程序有一定的了解。

以下介绍的模块，在发布的内核源码中不一定全部包含，可以根据需要，按照文档的说明，自己添加相关驱动的编译配置。

本章节不限定在某个固定的平台，但是为了描述方便，会以 halley2 nor 开发板为例进行介绍。

## 4.1. 内核配置和编译

在君正发布的 BSP 中，会根据发布的开发板型号，组成 defconfig。内核的 defconfig 一般组成格式如下：

[board_name]_[media_type]_linux_defconfig

[board_name] ： 发布开发板名。

[media_type]: 一般是开发板所使用的存储介质名。例如，nor,spinand 等。

具体使用哪一个作为开发板的默认配置，按照发布为准。

在 arch/mips/configs/目录下可以找到相应的配置文件。

例如，针对 halley2_nor_v10 的开发板，内核提供的默认配置为

halley2_nor_v10_linux_defconfig

在 PC 开发环境下执行

```
$ make halley2_nor_v10_linux_defconfig
$ make uImage
```

会生成 arch/mips/boot/uImage

可以在 defconfig 的基础上，通过 make menuconfig 添加自己的驱动模块，或为内核添加其它的特性。例如：

```
$ make halley2_nor_v10_linux_defconfig
$ make menuconfig
```

## 4.2. Xburst 板级介绍

在发布的内核版本中，针对不同的芯片型号，会在 arch/mips/xburst 目录下进行添加，该目录基本介绍如下：

```
common/                 #所有芯片公共部分
core/                   # xburst 核心文件
Kconfig
lib/
Makefile
Platform
soc-4775/               #4775 系列板级
soc-m200/               #m200 系列板级
soc-x1000/              #x1000 系列板级
```

以 halley2 开发板为例，其板级定义在

soc-x1000/chip-x1000/halley2

该目录重要文件介绍如下：

```
├── common
│   ├── 43438_bt_power_control.c        #蓝牙电源管理
│   ├── 43438_wlan_device.c
│   ├── 43438_wlan_power_control.c      #wifi 电源管理
│   ├── board_base.c                    # 各个 platform_device 注册
│   ├── board_base.h
│   ├── cim_board.c                     # camera 定义
│   ├── i2c_bus.c                       # 内核 i2c 设备描述
│   ├── keyboard_gpio.c
│   ├── lcd                             # 不同 LCD 屏幕参数配置
│   │   ├── lcd-kfm701a21_1a.c
│   │   └── lcd-truly_tft240240_2_e.c
│   ├── misc.c
│   ├── mmc.c                           # sd 控制器设备描述
│   ├── spi_bus.c                       # 内核 spi 设备描述
│   └── ssv6051_wlan_power_control.c
├── halley2_v10
│   ├── board.h                         #所有使用的 GPIO 定义。
```

## 4.3. GPIO 模块

X1000 的 gpio 控制器有五组，A，B，C，D，Z。

支持输入输出和设备复用功能。内核的 gpio 驱动程序是基于 gpio 子系统架构编写的。应用程序可以使用 gpio_demo 进行测试。内核驱动可以在内核空间使用，也可以通过导出 gpio sys 节点到用户空间，在用户空间进行操作。

### 4.3.1. 内核空间

#### 4.3.1.1. 文件介绍

gpio 一般在进行开发板设计的时候就已经固定好了，有的 gpio 只能作为设备复用功能管脚，有的 gpio 作为普通的输入输出和中断检测功能，对于固定设备复用的功能管脚在以下文件中定义：

arch/mips/xburst/soc-x1000/include/mach/platform.h

在 arch/mips/xburst/soc-x1000/common/platform.c 会根据驱动配置，选中相应的设备功能管脚。

内核的 gpio 驱动基于 gpio 子系统实现，所以其它驱动程序可以通过内核提供的 libgpio 接口，很方便的进行 gpio 控制，例如 gpio_request_one, gpio_get_value,gpio_set_value 等。

gpio 驱动文件所在位置：

arch/mips/xburst/soc-x1000/common/gpio.c

#### 4.3.1.2. 编译配置

内核通过配置 CONFIG_GPIOLIB 选项可以使用 gpio 功能，默认必须选上。

CONFIG_GPIOLIB:
 Symbol: GPIOLIB [=y]
 Type : boolean
 Prompt: GPIO Support
  Location:
  -> Device Drivers
  Defined at drivers/gpio/Kconfig:38
  Depends on: ARCH_WANT_OPTIONAL_GPIOLIB [=n] || ARCH_REQUIRE_GPIOLIB [=y]

通过配置 CONFIG_GPIO_SYSFS 选项，可以将 gpio 导出到用户节点/sys/class/gpio 下，对该节点下的文件操作，可以控制 gpio 输入输出。

Symbol: GPIO_SYSFS [=y]
Type : boolean
Prompt: /sys/class/gpio/... (sysfs interface)
  Location:
  -> Device Drivers
  -> GPIO Support (GPIOLIB [=y])
  Defined at drivers/gpio/Kconfig:69

Depends on: GPIOLIB [=y] && SYSFS [=y]

### 4.3.2. 用户空间

在内核导出 gpio 节点的前提下，可以操作/sys/class/gpio 节点，控制 gpio 输入输出。

/sys/class/gpio/

"export" ... 用户空间可以通过写其编号到这个文件，要求内核导出一个 GPIO 的控制到用户空间。

例如: 如果内核代码没有申请 GPIO #19,"echo 19 > export" 将会为 GPIO #19 创建一个 "gpio19" 节点。

"unexport" ... 导出到用户空间的逆操作。

例如: "echo 19 > unexport" 将会移除使用"export"文件导出的 "gpio19" 节点。

GPIO 信号的路径类似 /sys/class/gpio/gpio42/ (对于 GPIO #42 来说)，并有如下的读/写属性:

/sys/class/gpio/gpioN/

"direction" ... 读取得到 "in" 或 "out"。这个值通常运行写入。

写入"out" 时,其引脚的默认输出为低电平。为了确保无故障运行, "low" 或 "high" 的电平值应该写入 GPIO 的配置，作为初始输出值。注意:如果内核不支持改变 GPIO 的方向，或者在导出时内核代码没有明确允许用户空间可以重新配置 GPIO 方向，那么这个属性将不存在。

"value" ... 读取得到 0 (低电平) 或 1 (高电平)。如果 GPIO 配置为输出,这个值允许写操作。任何非零值都以高电平看待。

如果引脚可以配置为中断信号，且如果已经配置了产生中断的模式（见"edge"的描述），你可以对这个文件使用轮询操作(poll(2))，且轮询操作会在任何中断触发时返回。如果你使用轮询操作(poll(2))，请在 events 中设置 POLLPRI 和 POLLERR。如果你使用轮询操作(select(2))，请在 exceptfds 设置你期望的文件描述符。在轮询操作(poll(2))返回之后，既可以通过 lseek(2)操作读取 sysfs 文件的开始部分，也可以关闭这个文件并重新打开它来读取数据。

"edge" ... 读取得到“none”、“rising”、“falling”或者“both”。
将这些字符串写入这个文件可以选择沿触发模式，会使得轮询操作(select(2))在"value"文件中返回。

这个文件仅有在这个引脚可以配置为可产生中断输入引脚时，才存在。

"active_low" ... 读取得到 0 (假) 或 1 (真)。写入任何非零值可以翻转这个属性的(读写)值。已存在或之后通过"edge"属性设置了"rising"

## 4.4. I2C 模块

X1000 的 i2c 控制器为 SMB，SMB 支持的接口有 i2c,支持 100 Kb/s 和 400 Kb/s。

I2C 接口可连接至 pmu, camera，通过 i2c 接口进行配置。内核的 i2c 驱动程序是基于 i2c 子系统构编写的。应用程序可以使用 i2c_demo 进行测试。可以在内核空间通过 i2c 驱动操作 i2c 接口，也可以在用户空间通过 i2c_msg 操作 i2c 接口。

### 4.4.1. 内核空间

#### 4.4.1.1. 文件介绍

内核驱动基于 i2c 驱动架构,所有的硬件资源在板级定义。

按照 i2c 驱动编写规范，需要提供 i2c_board_info。

以下介绍驱动对应在内核的中的路径。

板级资源定义路径:

arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/i2c_bus.c

这里是各种使用 i2c 的设备，比如 camera， pmu。详情见本文档相应章节。

与 X1000 I2C 相关的驱动所在目录和文件说明,忽略目录中存在的其他文件。

driver/i2c/

```
├──── algos
├──── busses
│    ├──── i2c-v12-jz.c
├──── i2c-core.c
├──── i2c-dev.c
├──── i2c-smbus.c
```

#### 4.4.1.2. 编译配置

一般发布的软件版本中会默认配置 i2c 驱动,如果需要自己更改 i2c 的编译选项,可以通过以下方式进行配置。

配置以下选项:

CONFIG_I2C_V12_JZ

Symbol: I2C_V12_JZ [=y]

Type : tristate

Prompt: Ingenic SoC based on Xburst arch's I2C controler Driver support

Location:

-> Device Drivers

-> I2C support (I2C [=y])

-> I2C Hardware Bus support

Defined at drivers/i2c/busses/Kconfig:855

Depends on: I2C [=y] && HAS_IOMEM [=y] && MACH_XBURST [=y]

如果需要在用户控件编写 i2c 设备驱动，需要将 i2c 设备节点导出到用户空间 dev/下，配置以下选项：

CONFIG_I2C_CHARDEV:
Symbol: I2C_CHARDEV [=n]
Type : tristate
Prompt: I2C device interface
Location:
->Device Drivers
-> I2C support (I2C [=y])
Defined at drivers/i2c/Kconfig:38
Depends on: I2C [=y]

### 4.4.2. 用户空间

在 packages/example/Demo/i2c_demo 下，提供了 i2c 操作的接口。
通过填写 i2c_msg 结构体，通过 ioctl 在用户空间进行 i2c 设备的读写操作。
基于 i2c 应用程序编写，仅供参考：

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "i2c.h"
#include "I2cAdapter.h"
#include "i2c-dev.h"
class PmuDevice{
public:
    int PmuSmbReadByte(unsigned char reg, unsigned char slave_addr);
};
    I2cAdapter Phoenix(1);

int PmuDevice:: PmuSmbReadByte(unsigned char reg, unsigned char slave_addr)
{
    unsigned char msgbuf0[1];
    unsigned char msgbuf1[1];
    unsigned char cmd=reg;
    struct i2c_msg msg[2];
            msg[0].addr = slave_addr,
            msg[0].flags = 0,
            msg[0].len = 1,
            msg[0].buf = &cmd,
            msg[1].addr = slave_addr,
            msg[1].flags = I2C_M_RD,
            msg[1].len = 1,
```

```
                msg[1].buf = msgbuf1,
        Phoenix.I2cTransfer(msg, 2);
        printf("msgbuf1=%x\n", msgbuf1[0]);
}
int main()
{
        PmuDevice Pmu;
        unsigned char slave_addr = Phoenix.I2cSetSlave(0x32);
        Pmu.PmuSmbReadByte(0x2c, slave_addr);
}
```

## 4.5. SPI 模块

X1000 的 spi 控制器为 SSI，SSI 支持的接口有 Microwire，SSP，SPI。SPI 接口可连接至 spi nor，支持 spi 读写功能。内核的 spi 驱动程序是基于 spi 子系统架构编写的。应用程序可以使用 spi_demo 进行测试。Spi 总线下，可以挂普通的 char 设备，也可以挂 mtd 设备。可以在内核空间通过 spi 驱动操作 spi 接口，也可以在用户空间通过 spi_ioc_transfer 操作 spi 接口。

### 4.5.1. 内核空间

#### 4.5.1.1. 文件介绍

内核驱动基于 spi 驱动架构,所有的硬件资源在板级定义。

按照 spi 驱动编写规范，需要提供 spi_board_info。

以下介绍驱动对应在内核中的路径

板级资源定义路径:

arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/spi_bus.c，这个文件下面是 spi 的设备。

与 X1000 spi 相关的驱动所在目录和文件说明,忽略目录中存在的其他文件。

drivers/spi/

├── jz_spi.c

├── jz_spi.h

├── spi-bitbang.c

├── spi.c

├── spidev.c

#### 4.5.1.2. 编译配置

一般发布的软件版本中会默认配置 spi 驱动,如果需要自己更改 spi 的编译选项,可以通过以下方式进行配置。
在工作电脑上执行:
$ make menuconfig

配置以下选项:

(1)CONFIG_JZ_SPI:
SPI driver for Ingenic JZ series SoCs
Symbol: JZ_SPI [=y]
Type : tristate
Prompt: Ingenic JZ series SPI driver
Location:
-> Device Drivers
-> SPI support (SPI [=y])
Defined at drivers/spi/Kconfig:96
Depends on: SPI [=y] && SPI_MASTER [=y] && MACH_XBURST [=y]
Selects: SPI_BITBANG [=y]

配置以下选项，可以把 spi 的设备节点导出到/dev 下，供用户空间使用。
(2)CONFIG_SPI_SPIDEV:
Symbol: SPI_SPIDEV [=y]
Type : tristate
Prompt: User mode SPI device driver support
Location:
-> Device Drivers
-> SPI support (SPI [=y])
Defined at drivers/spi/Kconfig:613
Depends on: SPI [=y] && SPI_MASTER [=y]

### 4.5.2. 用户空间

在 packages/example/Demo/spi_demo 下，提供了 spi 操作的接口。
通过填写 spi_ioc_transfer 结构体，调用 ioctl，对 spi nor 设备进行读写操作。

```
#include <stdio.h>
#include <string.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include "spidev.h"
#include "spi.h"

class NorDev{
    public:
        int InitSpi(unsigned short mode, unsigned short bits, unsigned int speed);
};

SpiDev spinorflash(0,0);

int NorDev::InitSpi(unsigned short mode, unsigned short bits, unsigned int speed)
```

```
{
    spinorflash.SpiSetMode(mode);
    spinorflash.SpiSetBitsPerWord(bits);
    spinorflash.SpiSetMaxSpeed(speed);
}

int main()
{
    unsigned short mode = 0;
    unsigned short bits = 8;
    unsigned int speed = 1000000;
    unsigned int delay = 500;
    unsigned char send_buf[1] = {0x9f,};
    unsigned char recv_buf[3] = {0};

    NorDev nor;

    nor.InitSpi(mode, bits, speed);
    spinorflash.SpiMessageTransfer(send_buf, recv_buf, sizeof(send_buf) + sizeof(recv_buf));

    unsigned int id = (recv_buf[1] << 16) | (recv_buf[2] << 8) | recv_buf[3];
    printf("recv_buf[0]=%x      recv_buf[1]=%x    recv_buf[2]=%x    recv_buf[3]=%x\n",recv_buf[0],
recv_buf[1], recv_buf[2], recv_buf[3]);
    printf("id=%06x\n", id);
}
```

## 4.6. Audio 模块

X1000 Audio 音频控制器为 AIC，AIC 支持的接口有 I2S，SPDIF。I2S 接口可连接至内部 Codec，也可连接至外部 Codec。DMIC 控制器支持 DMIC 录音。PCM 控制器支持播放和录音，一般用作蓝牙通话方面。内核的音频驱动程序是基于 alsa 音频驱动架构编写的。应用程序可以使用 alsa-lib 库编程。

### 4.6.1. 内核空间

#### 4.6.1.1. 文件介绍

内核驱动基于 alsa 驱动架构，所有的硬件资源在板级定义。按照 alsa 音频驱动编写规范，实现声卡驱动需要提供 struct snd_soc_card，和声卡驱动用于传输数据的 snd_soc_platform_driver，以及不同的播放或录音设备所需要的 snd_soc_component_driver，以下介绍驱动对应在内核的中的路径

板级资源定义路径：
arch/mips/xburst/soc-x1000/common/platform.c

与 X1000 音频相关的驱动所在目录和文件说明，忽略目录中存在的其他文件。
sound/soc/ingenic/

```
.
├── asoc-board
│   ├── phoenix_icdc.c
│   ├── phoenix_spdif.c
├── asoc-v13
│   ├── asoc-aic-v13.h
│   ├── asoc-dma-dmic.c          #使用 dmic 语音唤醒时 snd_soc_platform_driver 实现。
│   ├── asoc-dma-v13.c           #通用 snd_soc_platform_driver 实现。
│   ├── asoc-dma-v13.h
│   ├── asoc-dmic-module.c       #使用 dmic 语音唤醒的 snd_soc_component_driver 实现。
│   ├── asoc-dmic-v13.c          #不使用 dmic 语音唤醒 snd_soc_component_driver 实现。
│   ├── asoc-dmic-v13.h
│   ├── asoc-i2s-v13.c           #i2s 控制器的 snd_soc_component_driver
│   ├── asoc-pcm-v13.c
│   ├── asoc-pcm-v13.h           #pcm 控制器的 snd_soc_component_driver 实现
│   └── asoc-spdif-v13.c
├── icodec
│   ├── dmic_dump.c
│   ├── icdc_d3.c                #内部 codec 操作具体实现
│   ├── icdc_d3.h
│   ├── pcm_dump.c
│   └── spdif_dump.c
```

#### 4.6.1.2. 编译配置

一般发布的软件版本中会默认配置音频驱动，如果需要自己更改音频的编译选项，可以通过以下方式进行配置。

在工作电脑上执行：

$ make menuconfig

配置以下选项：

CONFIG_SND_ASOC_INGENIC:

Say 'Y' to enable Alsa drivers of xburst.

Symbol: SND_ASOC_INGENIC [=y]
Type : tristate
Prompt: ASoC support for Ingenic
Location:
-> Device Drivers
-> Sound card support (SOUND [=y])
-> Advanced Linux Sound Architecture (SND [=y])
-> ALSA for SoC audio support (SND_SOC [=y])

### 4.6.2. 用户空间

开源项目 alsa-project,为基于 alsa 编写的音频驱动程序提供了用户空间访问的库 alsa-lib 和基于 alsa-lib 编写的 alsa-utils 工具，可以用来测试音频的播放和录音功能，具体使用方法可参考下文。

可访问 http://www.alsa-project.org 获取关于 alsa 的资料。

alsa 库整体占用资源较多，考虑 X1000 内存资源，可使用 tinyalsa 库替换 alsa 库，实现录音和播放功能。

#### 4.6.2.1. alsa 库

通过交叉编译 alsa-lib 和 alsa-utils，可以生成 amixer, aplay 程序，amixer 用于配置复杂的混音功能，通道切换等，aplay 用于播放 wav 格式音频文件，将 aplay 重命名为 arecord，使用 arecord 即可实现录音功能。

##### 4.6.2.1.1. alsa 工具使用

amixer, aplay, arecord 的使用方法如下：

假设播放的音频文件为 play.wav，录音生成的文件为 record.wav，在开发板上按照以下操作。

(1) 播放音频文件

$ aplay play.wav

(2) 通过 amic 录音

$ arecord -D hw:0,0 -c 2 -f S16_LE -r 44100 -d 10 record.wav

单独使用该命令可以在当前目录下生成 10 秒的录音文件 record.wav。一般在执行 arecord 命令之前，会通过 amixer 设置录音的通道和参数，命令如下：

$ amixer cset numid=17,iface=MIXER,name='ADC Mux' 0
$ amixer cset numid=4,iface=MIXER,name='Digital Capture Volume' 20
$ amixer cset numid=6,iface=MIXER,name='Mic Volume' 3

(3) 通过 LINE-IN 录音

X1000 内部 codec 的 LINE-IN 功能，与 amic 通路一模一样，使用 amic 的录音方式就可以实现 LINE-IN 功能

$ arecord -D hw:0,0 -c 2 -f S16_LE -r 44100 -d 10 record.wav

(4) 通过 dmic 录音

通过 dmic 录 8K 采样率的双通道音频数据：

$ arecord -D hw:0,2 -c 2 -f S16_LE -r 8000 -d 10 record.wav

（5）混响

1.ADC 数据叠加上DAC 通路的数据流(录音时将放音通路的数据做为背景)：

amixer cset numid=6,iface=MIXER,name='Mic Volume' 2
amixer cset numid=4,iface=MIXER,name='Digital Capture Volume' 30
amixer cset numid=17,iface=MIXER,name='ADC Mux' 0
amixer cset numid=12,iface=MIXER,name='ADC Mode Mux' 1
amixer cset numid=11,iface=MIXER,name='AIADC Mux' 2
amixer cset numid=10,iface=MIXER,name='ADC MIXER Mux' 2
amixer cset numid=9,iface=MIXER,name='mixer Enable' 1

2.DAC 数据叠加上ADC 通路的数据流(放音时将录音通路的数据做为背景)：

amixer cset numid=6,iface=MIXER,name='Mic Volume' 4
amixer cset numid=16,iface=MIXER,name='DAC_MERCURY VMux' 0
amixer cset numid=15,iface=MIXER,name='MERCURY AIDAC MIXER Mux' 2
amixer cset numid=14,iface=MIXER,name='DAC Mode Mux' 1
amixer cset numid=13,iface=MIXER,name='MERCURY AIDAC Mux' 2
amixer cset numid=9,iface=MIXER,name='mixer Enable' 1

录音播放：arecord -D hw:0,0 -c 2 -f S16_LE -r 8000 -d 10 a.wav&
aplay a.wav &

(6)audio 测试工具说明：

1.arecord 录音

-D 参数用于指定音频设备PCM

hw 的第一个参数用来指定声卡号，第二个参数用于指定设备号

-c 用于指定声道数

-f 用于指定数据格式

-r 用于指定采样频率

-d 用于指定录音时间

--help 获取帮助

2.aplay 放音

参数设置与 arecord 一致。

#### 4.6.2.1.2. 基于 alsa 库应用程序编写

alsa 为标准的音频架构，开源网站 http://www.alsa-project.org 有详细的教程可供参考，这里从网站摘取部分源码，仅供参考。

一个典型的应用程序一般按照以下流程，调用 alsa lib 接口。

```
open_the_device();
set_the_parameters_of_the_device();
while (!done) {
        /* one or both of these */
        receive_audio_data_from_the_device();
  deliver_audio_data_to_the_device();
}
close the device
```

以下提供一个简单的播放程序源码和录音程序源码，基本上使用了 alsa-lib 的常用 API。

一个简单的基于 alsa-lib 的播放程序，配置为立体声，16bit,，44.1KHZ。

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

main (int argc, char *argv[])
{
        int i;
        int err;
        short buf[128];
```

```
        snd_pcm_t *playback_handle;
        snd_pcm_hw_params_t *hw_params;

        if ((err = snd_pcm_open (&playback_handle, argv[1], SND_PCM_STREAM_PLAYBACK,
0)) < 0) {
                fprintf (stderr, "cannot open audio device %s (%s)\n",
                        argv[1],
                        snd_strerror (err));
                exit (1);
        }

        if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
                fprintf (stderr, "cannot allocate hardware parameter structure (%s)\n",
                        snd_strerror (err));
                exit (1);
        }

        if ((err = snd_pcm_hw_params_any (playback_handle, hw_params)) < 0) {
                fprintf (stderr, "cannot initialize hardware parameter structure (%s)\n",
                        snd_strerror (err));
                exit (1);
        }

        if ((err = snd_pcm_hw_params_set_access (playback_handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
                fprintf (stderr, "cannot set access type (%s)\n",
                        snd_strerror (err));
                exit (1);
        }

        if ((err = snd_pcm_hw_params_set_format (playback_handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
                fprintf (stderr, "cannot set sample format (%s)\n",
                        snd_strerror (err));
                exit (1);
        }

        if ((err = snd_pcm_hw_params_set_rate_near (playback_handle, hw_params, 44100, 0)) < 0)
{
                fprintf (stderr, "cannot set sample rate (%s)\n",
                        snd_strerror (err));
                exit (1);
        }
```

```
        if ((err = snd_pcm_hw_params_set_channels (playback_handle, hw_params, 2)) < 0) {
            fprintf (stderr, "cannot set channel count (%s)\n",
                    snd_strerror (err));
            exit (1);
        }

        if ((err = snd_pcm_hw_params (playback_handle, hw_params)) < 0) {
            fprintf (stderr, "cannot set parameters (%s)\n",
                    snd_strerror (err));
            exit (1);
        }

        snd_pcm_hw_params_free (hw_params);

        if ((err = snd_pcm_prepare (playback_handle)) < 0) {
            fprintf (stderr, "cannot prepare audio interface for use (%s)\n",
                    snd_strerror (err));
            exit (1);
        }

        for (i = 0; i < 10; ++i) {
            if ((err = snd_pcm_writei (playback_handle, buf, 128)) != 128) {
                fprintf (stderr, "write to audio interface failed (%s)\n",
                        snd_strerror (err));
                exit (1);
            }
        }

        snd_pcm_close (playback_handle);
        exit (0);
    }
```

一个简单的录音程序示例：

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <alsa/asoundlib.h>

    main (int argc, char *argv[])
    {
        int i;
        int err;
        short buf[128];
```

```
        snd_pcm_t *capture_handle;
        snd_pcm_hw_params_t *hw_params;

        if ((err = snd_pcm_open (&capture_handle, argv[1], SND_PCM_STREAM_CAPTURE, 0)) <
0) {
                fprintf (stderr, "cannot open audio device %s (%s)\n",
                        argv[1],
                        snd_strerror (err));
                exit (1);
        }

        if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
                fprintf (stderr, "cannot allocate hardware parameter structure (%s)\n",
                        snd_strerror (err));
                exit (1);
        }

        if ((err = snd_pcm_hw_params_any (capture_handle, hw_params)) < 0) {
                fprintf (stderr, "cannot initialize hardware parameter structure (%s)\n",
                        snd_strerror (err));
                exit (1);
        }

        if      ((err      =      snd_pcm_hw_params_set_access      (capture_handle,      hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
                fprintf (stderr, "cannot set access type (%s)\n",
                        snd_strerror (err));
                exit (1);
        }

        if      ((err      =      snd_pcm_hw_params_set_format      (capture_handle,      hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
                fprintf (stderr, "cannot set sample format (%s)\n",
                        snd_strerror (err));
                exit (1);
        }

        if ((err = snd_pcm_hw_params_set_rate_near (capture_handle, hw_params, 44100, 0)) < 0) {
                fprintf (stderr, "cannot set sample rate (%s)\n",
                        snd_strerror (err));
                exit (1);
        }

        if ((err = snd_pcm_hw_params_set_channels (capture_handle, hw_params, 2)) < 0) {
```

```
            fprintf (stderr, "cannot set channel count (%s)\n",
                    snd_strerror (err));
            exit (1);
        }

        if ((err = snd_pcm_hw_params (capture_handle, hw_params)) < 0) {
            fprintf (stderr, "cannot set parameters (%s)\n",
                    snd_strerror (err));
            exit (1);
        }

        snd_pcm_hw_params_free (hw_params);

        if ((err = snd_pcm_prepare (capture_handle)) < 0) {
            fprintf (stderr, "cannot prepare audio interface for use (%s)\n",
                    snd_strerror (err));
            exit (1);
        }

        for (i = 0; i < 10; ++i) {
            if ((err = snd_pcm_readi (capture_handle, buf, 128)) != 128) {
                fprintf (stderr, "read from audio interface failed (%s)\n",
                        snd_strerror (err));
                exit (1);
            }
        }

        snd_pcm_close (capture_handle);
        exit (0);
    }
```

#### 4.6.2.2. tinyalsa 库

##### 4.6.2.2.1. 源码介绍

Tinyalsa 是上层基于 alsa 接口实现的简易 alsa 库，能够进行播放和录音，库的体积较小，适合在内存资源比较紧张的系统上运行。

tinlyalsa 编译后的库的大小：libtinyalsa.so 约为 30kB。

Tinyalsa 可以使用以下命令从 github 上下载：

git clone https://github.com/tinyalsa/tinyalsa.git

交叉编译 tinyalsa 后，会生成 tinymix，tinypcminfo，tinyplay，tinycap 可执行程序和 libtinyalsa.so 库，通过这些执行程序，可以进行录音和播放测试。例如：

录音：

tinycap record.wav -D 0 -d 2 -c 2 -r 8000 -b 16

播放：

tinyplay play.wav

##### 4.6.2.2.2. 基于 tinyalsa 应用程序编写

与 alsa 库的应用程序编写流程一样，都是按照

1. 打开设备文件
2. 配置参数
3. 读数据或写数据
4. 结束

具体 API 使用方法可以参考源文件 tinymix.c,tinypcminfo.c,tinyplay.c,tinycap.c。

以下贴出代码片段，详细源码请查看上述源文件。

```
void play_sample(FILE *file, unsigned int card, unsigned int device, unsigned int channels,
                 unsigned int rate, unsigned int bits, unsigned int period_size,
                 unsigned int period_count)
{
    struct pcm_config config;
    struct pcm *pcm;
    char *buffer;
    int size;
    int num_read;

    memset(&config, 0, sizeof(config));
    config.channels = channels;
    config.rate = rate;
    config.period_size = period_size;
    config.period_count = period_count;
    if (bits == 32)
```

```
            config.format = PCM_FORMAT_S32_LE;
        else if (bits == 16)
            config.format = PCM_FORMAT_S16_LE;
        config.start_threshold = 0;
        config.stop_threshold = 0;
        config.silence_threshold = 0;

        if (!sample_is_playable(card, device, channels, rate, bits, period_size, period_count)) {
            return;
        }

        pcm = pcm_open(card, device, PCM_OUT, &config);
        if (!pcm || !pcm_is_ready(pcm)) {
            fprintf(stderr, "Unable to open PCM device %u (%s)\n",
                        device, pcm_get_error(pcm));
            return;
        }

        size = pcm_frames_to_bytes(pcm, pcm_get_buffer_size(pcm));
        buffer = malloc(size);
        if (!buffer) {
            fprintf(stderr, "Unable to allocate %d bytes\n", size);
            free(buffer);
            pcm_close(pcm);
            return;
        }

        printf("Playing sample: %u ch, %u hz, %u bit\n", channels, rate, bits);

        /* catch ctrl-c to shutdown cleanly */
        signal(SIGINT, stream_close);

        do {
            num_read = fread(buffer, 1, size, file);
            if (num_read > 0) {
                if (pcm_write(pcm, buffer, num_read)) {
                    fprintf(stderr, "Error playing sample\n");
                    break;
                }
            }
        } while (!close && num_read > 0);
        free(buffer);
        pcm_close(pcm);
    }
```

## 4.7. Camera 模块(cim)

X1000 camera 摄像头控制器为 cim，x1000 平台使用的接口为 dvp，8 位数据并行传输，硬件上用 hsync，vsynv，pclk，等控制信号控制 camear 数据的传输，cim 控制器支持输入格式为 yuv422，输出格式支持 yuv422.，内核的 camera 控制器驱动 和 sensor 驱动是基于 v4l2 驱动架构编写的。应用程序可以使用 cimutils 进行测试。

### 4.7.1. 内核空间

#### 4.7.1.1. 文件介绍

内核驱动基于 v4l2 驱动架构，所有的硬件资源在板级定义。按照 v4l2 视屏驱动编写规范，实现视屏驱动提供的控制接口 v4l2_ioctl_ops，和视屏驱动用于传输数据的 v4l2_subdev_core_ops 和 v4l2_subdev_video_ops。

板级资源定义路径：

arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/cim_board.c 在 arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/cim_board.c 中实现 soc_camera_link 结构体和注册设备到 platform 平台，

I2c 注册文件路径：

arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/i2c_bus.c

在 arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/i2c_bus.c 定义 sensor 的名称，以及 i2c 设备地址。

与 X1000camera 相关的驱动所在目录和文件说明，忽略目录中存在的其他文件。

```
drivers/media/platform/soc_camera/
    ├──── jz_camera_v13.c #控制器驱动的实现
    ├──── soc_camera.c    #soc_camera 子系统的实现

drivers/media/i2c/soc_camera/
    ├──── ov5640.c        #sensor 驱动的实现
    ├──── gc2155.c        #sensor 驱动的实现
```

#### 4.7.1.2. 编译配置

一般发布的软件版本中会默认不会配置 camera 驱动，如果需要自己添加 camera 的编译选项，可以通过以下方式进行配置。

在工作电脑上执行：

$ make menuconfig

配置以下选项：

Symbol: VIDEO_JZ_CIM_HOST_V13 [=y] #选择 x1000 camera 的控制器驱动

Type : tristate

Prompt: ingenic cim driver used on camera x1000

Location:

-> Device Drivers

-> Multimedia support (MEDIA_SUPPORT [=y])

-> V4L platform devices (V4L_PLATFORM_DRIVERS [=y])

-> SoC camera support (SOC_CAMERA [=y])

Symbol: SOC_CAMERA_GC2155 [=y] #选择具体 camera 型号。

Type : tristate

Prompt: gc2155 camera support

Location:

-> Device Drivers

- > Multimedia support (MEDIA_SUPPORT [=y])

-> Sensors used on soc_camera driver

Symbol: JZ_IMEM [=y] #为 camera use_ptr 模式提供 vpu 编码所需要的 memory

Type : boolean

Prompt: jz imem

Location:

-> Machine selection

-> SOC type (SOC_TYPE [=y])

CONFIG_JZ_VPU: # 使用 vpu（硬件编码）作为编码的方式

VPU support.

Symbol: JZ_VPU [=y]

Type : boolean

Prompt: JZ VPU driver

Location:

-> Device Drivers

-> Graphics support

Symbol: I2C0_V12_JZ [=y] #选择 i2c0 控制器

Type : tristate

Prompt: JZ_v12 i2c controler 0 Interface support

Location:

-> Device Drivers

-> I2C support (I2C [=y])

-> I2C Hardware Bus support

Symbol: I2C0_SPEED [=100]　　#设置 i2c0 控制器为 100KHz
Type　: integer
Range : [100 400]
Prompt: Jz_v12 i2c0 speed in KHZ
Location:
-> Device Drivers
-> I2C support (I2C [=y])
-> I2C Hardware Bus support

### 4.7.2. 用户空间

#### 4.7.2.1. Camera 应用的使用

在应用层我们可以用 cimutils 来测试 camera 是否正常。cimutils 可以拍照和预览，拍照可保存成为 jpeg，bmp，和 raw 格式的文件，预览可以把 camera 获取到的数据显示到显示屏上。Cimutils 应用可以动态设置拍照的大小和格式以及保存的格式和名字，目前 cimutils 支持 user 和 mmap 两种模式。

例如：

user_ptr:

./cimutils -I 0 -C -x 640　-y 480 -v　　#拍照命令
./cimutils -I 0 -P -w 320　-h 240 -v　　#预览命令

Mmap:

./cimutils -I 0 -C -x 640　-y 480　　#拍照命令
./cimutils -I 0 -P -w 320　-h 240　　#预览命令

## 4.8. SD 卡模块

### 4.8.1. 内核空间

1. 板级配置文件：
   arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/mmc.c
   其中：
   struct jzmmc_platform_data tf_pdata 描述 sd 卡设备。
   struct jzmmc_platform_data sdio_pdata 描述 sdio 类设备，一般指 sdio-wifi。
2. 电路连接：
   sd 卡接到 MSC0，sdio-wifi 接到 MSC1。

1. 驱动程序文件：
   drivers/mmc/host/jzmmc_v12.c
驱动配置：

Symbol: JZMMC_V12 [=y]
Type : tristate
Prompt: Ingenic(XBurst) MMC/SD Card Controller(MSC) v1.2 support
  Location:
    -> Device Drivers
      -> MMC/SD/SDIO card support (MMC [=y])
  Defined at drivers/mmc/host/Kconfig:7
  Depends on: MMC [=y] && (SOC_M200 [=n] || SOC_X1000 [=y])

### 4.8.2. 用户空间

MSC 支持 SD 设备，SDIO 设备，MMC 设备，以 SD 卡设备为例。SD 卡作为一个标准的块设备驱动程序，在 dev/ 下会产生/dev/mmcblk*p*类似的设备节点。使用 mount 命令能够将分区 mount 到目录
例如：
$ mount -t ext4 /dev/mmcblk0p7 /mnt

如果 mount 失败，一般是由于内核没有支持 ext4 文件系统选项，或者分区不是 ext4 文件系统类型。

## 4.9. LCD 模块

X1000 SLCD Controller 支持 16/18/24 bit 6800/8080 并行接口，9 bit6800/8080 2 次并行接口，8 bit 6800/8080 1/2/3 次并行接口，8/16/18/24 bit 串行接口。

### 4.9.1. 内核空间

#### 4.9.1.1. 文件介绍

内核驱动基于 framebuffer 驱动架构，所有的硬件资源在板级定义。按照 framebuffer 驱动编写规范，实现 LCD 驱动需要提供 struct fb_info，以及用作显示的 platform_driver 以下介绍驱动对应在内核的中的路径：

板级资源定义路径：

Phoenix 开发板的板级：

arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/lcd/

Halley2 开发板的板级：

arch/mips/xburst/soc-x1000/chip-x1000/halley2/common/lcd/

与 X1000 LCD 相关的驱动所在目录和文件说明，忽略目录中存在的其他文件。

driver/video/jz_fb_v13/

├── regs.h #包含 lcdc 的所有寄存器。

├── jz_fb.h #定义了 struct jzfb，包含驱动过程中的所有参数，放到 fb_info->par；定义了 ioctl 以及 ioctl 相关的结构体。

├── jz_fb.c #驱动实现的文件。

#### 4.9.1.2. 编译配置

一般发布的软件版本中会默认配置 LCD 驱动，如果需要自己更改 LCD 屏幕，可以通过以下方式进行配置。

在工作电脑上执行：

$ make menuconfig

配置以下选项：

以 slcd truly240240 为例

配置

CONFIG_LCD_TRULY_TFT240240_2_E

Location:

-> Device Drivers

-> Graphics support

-> Backlight & LCD device support (BACKLIGHT_LCD_SUPPORT [=y])

-> Lowlevel LCD controls (LCD_CLASS_DEVICE [=y])

如需添加新的 LCD 屏幕，需要在板级添加相关文件和配置，以 SLCD truly240240 为例：

arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/lcd/

添加 lcd-truly_tft240240_2_e.c 文件，填写屏幕参数；

Makefile 中添加 obj-$(CONFIG_LCD_TRULY_TFT240240_2_E) += lcd-truly_tft240240_2_e.o

driver/video/backlight/

添加 truly_tft240240_2_e.c 文件，配置 gpio、电源配置等；

Kconfig 中添加：

```
config LCD_TRULY_TFT240240_2_E
    tristate "SLCD TRULY TFT240240-2-E with control IC st7789s (240x240)"

    depends on BACKLIGHT_CLASS_DEVICE

    default n
```

### 4.9.2. 用户空间

#### 4.9.2.1. 操作方法

可以使用 ioctl 直接对设备进行操作，也可以使用提供的 libfb.so，实现显示操作。

#### 4.9.2.2. 通过调用 ioctl 实现对 LCD 的控制

Open /dev/fb0 设备节点

| | |
|---|---|
| JZFB_ENABLE | 使能 LCD 控制器 |
| JZFB_ENABLE_FG0 | 使能 fg0 |
| FBIOGET_VSCREENINFO | 获取 framebuffer 可变参数 |
| JZFB_SET_FG_SIZE | 设置 fg0 显示大小 |
| JZFB_SET_FG_POS | 设置 fg0 显示起始位置 |
| JZFB_SET_FG_ALPHA | 设置 fg0 透明度 |
| JZFB_GET_FG_SIZE | 获取 fg0 显示大小 |
| JZFB_GET_FG_POS | 获取 fg0 显示起始位置 |
| JZFB_GET_FG_ALPHA | 获取 fg0 透明度 |

mmap 获取 framebuffer 地址，填充显示图像

FBIOPAN_DISPLAY 显示图像

#### 4.9.2.3. 使用 libfb.so

包含 libfb.h 头文件；

```
jzfb_init(void)                    打开设备节点，获取 var
jzfb_cfg(struct jzfb_par *fg0_par)        配置 fg0 显示参数,用于单层显示
jzfb_draw(void *buf)                显示单层图像
jzfb_deinit(void)                  关闭设备节点，释放内存

struct jzfb_par {
        unsigned int pos_x;             //显示起始位置坐标 x
        unsigned int pos_y;             //显示起始位置坐标 y
        unsigned int size_w;                //显示图像宽度
        usnigned int size_h;            //显示图像高度
        unsigned int alpha_enable;      //是否显示透明度
        unsigned int alpha_value;           //透明度数值  0x00--0xff
};
```

#### 4.9.2.4. 使用 libfb.so 的简单的应用程序

```
#include "libfb.h"
#include <stdio.h>
#include <malloc.h>

static void jzfb_display_v_color_bar(struct jzfb_par *fg0_par, void *buf_0);

int main()
{
    printf("Test Osd.\n");
    struct jzfb_par *fg0_par;
    unsigned char *buf_0;

    fg0_par = (struct jzfb_par*)malloc(sizeof(struct jzfb_par));

    fg0_par->pos_x = 0;
    fg0_par->pos_y = 0;
    fg0_par->size_w = 360;
    fg0_par->size_h = 600;
    fg0_par->alpha_enable = 1;
    fg0_par->alpha_value = 0xff;

    buf_0 = (unsigned char*)malloc(fg0_par->size_w * fg0_par->size_h * 4);

    jzfb_init();
    jzfb_display_v_color_bar(fg0_par, (void*)buf_0);

    jzfb_cfg(fg0_par);
```

```
    jzfb_draw(buf_0);
    usleep(2000000);

    jzfb_deinit();
    free(fg0_par);
    return 0;
}

static void jzfb_display_v_color_bar(struct jzfb_par *fg0_par, void *buf_0)
{
    int i,j;
    int w, h;
    int bpp;
    unsigned int *p32;
    unsigned short *p16;

    p32 = (unsigned int *)buf_0;
    p16 = (unsigned short *)buf_0;

    bpp = 16;

    w = fg0_par->size_w;
    h = fg0_par->size_h;

    for (i = 0; i < h; i++) {
        for (j = 0; j < w; j++) {
            short c16;
            int c32;
            switch ((j / 10) % 4) {
            case 0:
                c16 = 0xF800;
                c32 = 0x00FF0000;
                break;
            case 1:
                c16 = 0x07C0;
                c32 = 0x0000FF00;
                break;
            case 2:
                c16 = 0x001F;
                c32 = 0x000000FF;
                break;
            default:
                c16 = 0xFFFF;
                c32 = 0xFFFFFFFF;
```

```
                    break;
                }
                switch (bpp) {
                case 18:
                case 24:
                case 32:
                    *p32++ = c32;
                    break;
                default:
                    *p16 ++ = c16;
                }
            }
        }

        return ;
}
```

## 4.10. VPU 模块

Vpu 是视屏处理单元，在 x1000 中用到了 vpu 中的 jpeg 模块，JPEG 模块是一个 JPEG 编码单元，用来将 yuv 数据转换成 jpeg 格式。目前 jpeg 支持输入数据格式为 yuv422，输出格式 jpeg，目前支持 3 个级别 jpeg 的格式输出，分别是低，中，高。

### 4.10.1. 内核空间

板级资源定义路径：
arch/mips/xburst/soc-x1000/common/platform.c 申请资源
arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/board_base.c 注册资源

与 X1000 vpu 相关的驱动所在目录和文件说明，忽略目录中存在的其他文件。
drivers/video/jz_vpu/
|——jz_vpu_v13.c #vpu 驱动的具体实现
|——jz_vpu_v13.h
在发布版本中默认不会添加 vpu 驱动，如果想要自己改变 vpu 的编译配置可以通过以下进行配置：

```
CONFIG_JZ_VPU:
VPU support.
Symbol: JZ_VPU [=y]
Type    : boolean
Prompt: JZ VPU driver
  Location:
    -> Device Drivers
      -> Graphics support
```

### 4.10.2. 用户空间

在用户空间，vpu 主要功能是把摄像头或者其他方式输入的 yuv422 格式的数据进行编码，转化为 jpeg 格式。在 development/libs/jpg_api/jpeg_encode2.c 中提供相应的接口：

```
/*jpeg 初始化函数，用来初始化 vpu*/
1. void* jz_jpeg_encode_init(int width,int height)
{
    struct jz_jpeg_encode *jz_jpeg;
    jz_jpeg = malloc(sizeof(struct jz_jpeg_encode));
    jz_jpeg->vpu = vpu_init();
    if(!jz_jpeg->vpu){
        fprintf(stderr,"VPU init failure!\n");
    }
```

```
        jz_jpeg->yuyv_info.des_va = (unsigned int)JZMalloc(128,sizeof(int));

        if(!jz_jpeg->yuyv_info.des_va)
            fprintf(stderr,"Alloc jz_jpeg->yuyv_info.des_va memory failure!\n");
        jz_jpeg->yuyv_info.width = width;
        jz_jpeg->yuyv_info.height = height;
        jz_jpeg->yuyv_info.BitStreamBuf = JZMalloc(128,width * height *2 /10);
        if(!jz_jpeg->yuyv_info.BitStreamBuf)
            fprintf(stderr,"Alloc jz_jpeg->yuyv_info.BitStreamBuf memory failure!\n");
        return (void *)jz_jpeg;
    }
    /*反初始化*/
    2. void jz_jpeg_encode_deinit(void *handle){
            struct jz_jpeg_encode *jz_jpeg = (struct jz_jpeg_encode *)handle;
            vpu_exit(jz_jpeg->vpu);
            free(jz_jpeg);
            jz47_free_alloc_mem();
    }

    /*将 yuv422 格式的数据转换为 jpeg 格式的数据，并保存为 xxx.jpg 文件*/
    3. int yuv422_to_jpeg(void *handle,unsigned char *input_image, FILE *fp, int width, int
height, int quality)
    {
            struct jz_jpeg_encode *jz_jpeg = (struct jz_jpeg_encode *)handle;
            if(!handle)
            {
                fprintf(stderr,"jz_put_jpeg_yuv420p_memor:handle isn't init or init failed!\n");
                    return -1;
            }
            jz_jpeg->yuyv_info.width = width;
            jz_jpeg->yuyv_info.height = height;
            jz_jpeg->yuyv_info.ql_sel = quality;
            jz_jpeg->yuyv_info.buf = input_image;

jz_jpeg->yuyv_info.des_pa=(unsignedint)get_phy_addr(jz_jpeg->yuyv_info.des_va);

    /* 2: jpge strcut init */
            jpge_struct_init(jz_jpeg);
            jz_jpeg->bs_size =
    jz_start_hw_compress(jz_jpeg->vpu,jz_jpeg->yuyv_info.des_va,jz_jpeg->yuyv_info.des_pa);
            gen_image(&jz_jpeg->yuyv_info, fp, jz_jpeg->bs_size);
            return 0;
```

## 4.11. USB 模块

USB，是英文 Universal Serial Bus（通用串行总线）的缩写，而其中文简称为“通串线”，是一个外部总线标准，用于规范电脑与外部设备的连接和通讯。是应用在 PC 领域的接口技术。USB 接口支持设备的即插即用和热插拔功能。USB 是在 1994 年底由英特尔、康柏、IBM、Microsoft 等多家公司联合提出的。

### 4.11.1. 内核空间

资源相关文件路径：

A) 板级资源定义路径:arch/mips/xburst/soc-x1000/common/platform.c

B) 平台设备注册文件路径：

arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/ board_base.c

C）GPIO 申请文件路径：

在“arch/mips/xburst/soc-x1000/chip-x1000/phoenix/common/misc.c”中申请 userusb 的 dete pin,ID pin,bus pin

usb 控制器驱动的目录如下：

```
drivers/usb/dwc2/
|——dwc2
    |——dwc2_jz.c        #usb 控制器实现
    |——core.c           #usb core 层在控制器中的具体实现
    |——gadget.c         #usb 一些工具在控制器的具体实现
    |——rh.c
    |——debugfs.c          #debugfs 虚拟文件系统的实现
    |——ep0.c
    |——host-ddma.c
```

在发布版本中会默认添加 usb 驱动，usb 分为 host 端和 device 端，如果想要自己改变 usb 的编译配置可以通过以下进行配置：

### 4.11.2. usb-host

usb-host 端又可以分为 usb-mass_storage 和 usb-camera，下面会议此介绍：

#### 4.11.2.1. usb-mass_storage:

在电脑端输入 make menuconfig，然后配置：

Device Drives

->USB support

->DesignWare USB2 DRD Core Support

->Driver Mode

->Both host and device.

同时在本级目录下选上 USB Mass Storage support，然后配置：

->DeviceDrivers
-> SCSI device support
->SCSI device support

现在 usb-host-mass_storage 功能就配置成功。当有 u 盘插入时会有如下提示：

```
[    22.092290] jz-dwc2 jz-dwc2: set vbus on(on) for host mode
[    22.202174] USB connect
[    22.902209] usb 1-1: new high speed USB device number 2 using dwc2
[    23.328262] usb 1-1: New USB device found, idVendor=0930, idProduct=6545
[    23.342090] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[    23.356745] usb 1-1: Product: DataTraveler 2.0
[    23.365858] usb 1-1: Manufacturer: Kingston
[    23.374570] usb 1-1: SerialNumber: C86000BDBA09EF60CA285106
[    23.412410] scsi0 : usb-storage 1-1:1.0
[    24.475824] scsi 0:0:0:0: Direct-Access     Kingston DataTraveler 2.0 PMAP PQ: 0 ANSI: 4
[    25.751099] sd 0:0:0:0: [sda] 30497664 512-byte logical blocks: (15.6 GB/14.5 GiB)
[    25.768695] sd 0:0:0:0: [sda] Write Protect is off
[    25.779135] sd 0:0:0:0: [sda] No Caching mode page present
[    25.790501] sd 0:0:0:0: [sda] Assuming drive cache: write through
[    25.807171] sd 0:0:0:0: [sda] No Caching mode page present
[    25.832513] sd 0:0:0:0: [sda] Assuming drive cache: write throug
[    25.879083] sda:sdal
[    25.895075] sd 0:0:0:0: [sda] No Caching mode page present
[    25.932372] sd 0:0:0:0: [sda] Assuming drive cache: write through
[    25.964595] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

#### 4.11.2.2. usb-camera

再上述的基础上，配置：

-> Device Drivers
-> Multimedia support
-> Media USB Adapters
-> USB Video Class

就是可以使用 usb-camera。

### 4.11.3. usb-device

目前 gadget 功能使用 android gadget ，android gadget 目前支持 mass_storge, adb，rndis 功能。

Device Drivers
->USB support
->USB Gadget support
->USB Gadget Drivers (Android Gadget)

下面分别说明上述三个功能如可开启：

#### 4.11.3.1. adb & mass_storage

在文件系统中输入下面命令:

```
cd /sys/class/android_usb/android0
    echo 0 > enable
    echo 18d1 > idVendor
    echo d002 > idProduct
    echo mass_storage,adb > functions
    echo 1 > enable
    /sbin/adbd &
```

现在 usb 支持了 adb 和 mass_storage 功能。具体验证方法：

验证 adb：

在电脑端输入 adb shell 出现/sys/devices/virtual/android_usb/android0 # 说明 adb 开启成功。

验证 mass_storage:

1. dd if=/dev/zero of=fat32.img bs=1k count=2048 #制作一个大小为 2M 的空文件
2. mkfs.vfat fat32.img #把这个文件格式化为 vfat 格式
3. echo fat32.img > suy/device/platform/jz-dwc2/dwc2/lun0/file

如果成功会在电脑端弹出 u 盘的盘符。

#### 4.11.3.2. rndis 功能

```
cd /sys/class/android_usb/android0
echo 0 > enable
echo 18d1 > idVendor
echo d002 > idProduct
echo rndis > functions
echo 1 > enable
```

配置完后同时在电脑端和设备端出现 usb0 网络端点，可用 ifconfig -a 查看，把这两个 usb0 配置成两个不同的 ip，可用 ping 命令来检测是否成功。

## 4.12. 蓝牙-WIFI 模块

蓝牙、wifi 芯片使用的是 BCM43341，发布的版本默认提供蓝牙 wifi 配置，这里只说明测试方法。

### 4.12.1. 蓝牙

使用步骤详解:

1. 开发板执行 bt_enable

```
# bt_enable
[   34.997129] restore_pin_status is not defined
port --baudrate --------------- 5
=========read from uart_fd===(null)=Done setting line discpline
Broadcom firmware initialized.


#
```

2. 开发板执行 sdptool add OPUSH

```
#
# sdptool add OPUSH
OBEX Object Push service registered
#
```

3. 链接设备开启蓝牙，搜索 BlueZ，点击该设备进行配对
注： 配对成功，开发板有如下相应提示

```
#
# Confirmation request of 232480 for device /org/bluez/144/hci0/dev_B4_30_52_EE_0B_8E
#
```

4. 开发板执行 obex_test -b local 9

```
# obex_test -b local 9
Using Bluetooth RFCOMM transport
OBEX Interactive test client/server.
>
```

5. 链接设备准备好要传送的文件

6. 开发板执行 s （准备传输文件）

链接设备要在规定时间内将文件分享到 BlueZ 蓝牙

```
# sdptool add OPUSH
OBEX Object Push service registered
# Confirmation request of 359585 for device /org/bluez/144/hci0/dev_B4_30_52_EE_0B_8E
# obex_test -b local 9
Using Bluetooth RFCOMM transport
OBEX Interactive test client/server.
> s
Timeout while doing OBEX_HandleInput()
```

注：如果分享文件超时会有如下提示：

```
# obex_test -b local 9
Using Bluetooth RFCOMM transport
OBEX Interactive test client/server.
> s
Timeout while doing OBEX_HandleInput()
> s
Server register error! (Bluetooth)
```

注：分享超时，可重新执行第 6 步即可

7. 开发板执行 s（开始传输）

```
# sdptool add OPUSH
OBEX Object Push service registered
# Confirmation request of 359585 for device /org/bluez/144/hci0/dev_B4_30_52_EE_0B_8E
# obex_test -b local 9
Using Bluetooth RFCOMM transport
OBEX Interactive test client/server.
> s
Timeout while doing OBEX_HandleInput()
> s
connect_server()
connect_server() Skipped header c0
Server request finished!
server_done() Command (00) has now finished
OBEX_HandleInput() returned 12
OBEX_HandleInput() returned 990
Unknown event 0b!
Made some progress...
OBEX_HandleInput() returned 1024
Made some progress...
OBEX_HandleInput() returned 954
```

重启休眠注意事项:

1. 重启后， 开发板不会保存蓝牙设置，要使用蓝牙，需要将按上述步骤重新设置一遍.

2. 休眠后重启后， 要先杀死下图中的进程， 再重新设置一遍

```
  134 root     brcm_patchram_plus --enable_hci --baudrate 3000000 --no2bytes --
```

注意：
在多次链接蓝牙时，或链接蓝牙错误时，会建立多个重复的上述进程。要将这些进程清理干净，才能再次使用蓝牙！

### 4.12.2. WIFI

**Wifi 使用操作步骤**

1. 链接设备链接 wifi，
   开启 AirKissDemo 应用程序
   按提示输入 wifi 密码

2. 开发板执行 airkiss

```
# airkiss
killall: udhcpc: no process killed
killall: wpa_supplicant: no process killed
[  900.361412] BCM:
[  900.361412] Dongle Host Driver, version 1.141.66
[  900.361412] Compiled in drivers/net/wireless/bcmdhd on Apr  8 2016 at 17:47:47
[  900.376280] BCM:wl_android_wifi_on in
[  900.380068] BCM:wifi_platform_set_power = 1
[  900.384920] BCM:========== WLAN placed in RESET ON  ========
[  900.612576] sdio_reset_comm():
[  900.641229] BCM:F1 signature OK, socitype:0x1 chip:0xa9a6 rev:0x0 pkg:0x4
[  900.649431] BCM:DHD: dongle ram size is set to 524288(orig 524288) at 0x0
[  900.780699] BCM:dhdsdio_write_vars: Download, Upload and compare of NVRAM succeeded.
[  900.838309] BCM:dhd_bus_init: enable 0x06, ready 0x06 (waited 0us)
[  900.846643] BCM:wifi_platform_get_mac_addr
[  900.852523] BCM:dhd_get_concurrent_capabilites: Get P2P failed (error=-23)
[  900.860231] BCM:Firmware up: op_mode=0x0001, MAC=00:90:4c:c5:12:38
[  900.872060] BCM:dhd_preinit_ioctls pspretend_threshold for HostAPD failed  -23
[  900.885051] BCM:Firmware version = wl0: Jun 12 2015 17:06:46 version 7.10.323.47.cn2.x7 FWID 01-dcca911a es4.c3.n3.a2
[  900.897361] BCM:dhd_wlfc_hostreorder_init(): successful bdcv2 tlv signaling, 64
Easy setup target library v3.2.0
```

当出现“Easy setup target library v3.2.0”时，表示开发板 wifi 准备成功，此时可以执行下一步操作。

3. 链接设备按提示点击发送
开发板提示：

```
ssid: JZ_MD
password: 1JZmdingenic2
/etc/wpa_supplicant.conf create successfully!
random: 0x67
/etc/airkiss_random.conf create successfully!
[ 1020.587477] BCM:[CFG80211-ERROR) wl_notify_scan_status : BCM:scan is not ready
scan gets no result(ret: -1, count: 0).
[ 1023.605034] BCM:[CFG80211-ERROR) wl_notify_scan_status : BCM:scan is not ready
scan gets no result(ret: -1, count: 0).
[ 1026.613089] BCM:[CFG80211-ERROR) wl_notify_scan_status : BCM:scan is not ready
scan gets no result(ret: -1, count: 0).
security: wpa2# killall: udhcpc: no process killed
killall: wpa_supplicant: no process killed
Successfully initialized wpa_supplicant
[ 1029.797524] BCM:[CFG80211-ERROR) wl_cfg80211_connect : BCM:Connectting withff:ff:ff:ff:ff:ff channel (0) ssid "JZ_MD", len (5)

udhcpc (v1.22.1) started
Sending discover...
[ 1030.905184] BCM:wl_bss_connect_done succeeded with 8c:0c:90:d8:47:c8
[ 1030.985464] BCM:wl_bss_connect_done succeeded with 8c:0c:90:d8:47:c8
Sending discover...
Sending select for 10.10.50.35...
Lease of 10.10.50.35 obtained, lease time 86400
deleting routers
adding dns 192.168.1.2
```

当出现“adding dns 192.168.1.2”时，表示开发板接受 wifi 密钥完成，此时可以执行下一步操作。

注：Ssid，password 为 wifi 的用户名和密码。其被保存在/etc/wpa_supplicant.conf 文件中

```
# cat /etc/wpa_supplicant.conf

network={
scan_ssid=1
ssid="JZ_MD"
psk="1JZmdingenic2"
priority=1
}
```

4. 开发板执行 ping www.baidu.com， 验证 wifi 是否连接成功

```
# ping www.baidu.com
PING www.baidu.com (220.181.111.188): 56 data bytes
64 bytes from 220.181.111.188: seq=0 ttl=54 time=241.266 ms
64 bytes from 220.181.111.188: seq=1 ttl=54 time=19.682 ms
64 bytes from 220.181.111.188: seq=2 ttl=54 time=19.649 ms
```

**wifi 重启休眠共用等注意事项**

1. 重启，休眠唤醒后开发板会根据/etc/wpa_supplicant.conf 自动连接 wifi。

执行 wifi 功能的进程：

```
 108 root     wpa_supplicant -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf -B
```

2. Wifi 和 Lan 不可同时使用

Wifi 链接时会配置 wlan0， Lan 链接时会配置 eth0，两者造成冲突。

如在 wifi 链接成功后，需要使用 LAN，需要删除/etc/wpa_supplicant.conf（其会导致 wpa_supplicant -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf 不执行），即可。

## 4.13. SECURITY 加密模块

AES-RSA 的驱动名称是“jz_security”
（1）设备节点：/dev/jz_security
使用如下的方式来操作驱动：
open(/dev/jz-security, ...) → ioctl(xxx...) → ioctl(xxx...) → ...→ ioctl(xxx) -->close(device)
（2）驱动文件路径：
“kernel/drivers/misc/jz_security/”

### 4.13.1. 驱动配置

Device Drivers
-->Misc devices
-->JZ SECURITY Driver(AES && RSA)

### 4.13.2. IOCTL 命令定义

#define SECURITY_INTERNAL_CHANGE_KEY (0xffff0010)
/*设置 AES-KEY*/
#define SECURITY_INTERNAL_AES (0xffff0020)
/*AES 加密或解密*/
#define SECURITY_RSA (0xffff0030)
/*RSA 加密或解密*/

### 4.13.3. 驱动结构体描述

以下驱动结构体定义路径为“kernel/tools/security-test”。
（1）定义 AES-KEY 结构如下：

```
struct rsa_aes_packet {
unsigned short oklen; //old AES-KEY len (unit:word)
unsigned short nklen; // new AES-KEYlen (unit:word)
unsigned int * okey; // old AES-KEYaddr
unsigned int * nkey;// new AES-KEYaddr
};
```

也可以用数组来存储 AES-KEY，具体定义方式可见下例：
unsigned int user_key[9]=

```
{
    0x00040004,
    /*bit31~16: old key length ,bit15~bit0:new key length*/
    0x1,0x2,0x3,0x4, /*old AES-KEY*/
    0x4, 0x5,0x6,0x7, /*new AES-KEY*/
}
```

注：初始化 AES-KEY 时，需将“old AES-KEY”“new AES-KEY”设置成相同的初值。

（2）RSA 参数结构

用户使用 RSA 加解密“AES-KEY”,描述 RSA 的结构体为“struct rsa_param”，具体结构如下：

```
struct rsa_param {
unsigned int in_len;        //input data length (units:word)
unsigned int key_len;     //private or public key length(units:word)
unsigned int n_len;            //n length(units:word)
unsigned int out_len;      //ouput length(units:word)
unsigned int *input;            //input data buffer
unsigned int *key;         /*Ku or KR*/ buffer
unsigned int *n;           /*N*/buffer
unsigned int *output;      //encrypted data or decrypted data buffer
unsigned int mode;         //mode: 1,encrypt 0,decrypt
};
```

（3）CHANGE-KEY 结构体

```
struct change_key_param {
int len;                          //rsa_enc_data len in bytes.
int *rsa_enc_data;                //okey_len,nkey_len,okey,nkey
int *n_ku_kr;                     //NKU or NKR buffer, for rsa decrypt (62 words)
int init_mode;                    //init key 1;init key, 0:change key
};
```

其中 rsa_enc_data 是上述结构体“struct rsa_aes_packet”经过 rac 加密后的数据，上述加密数据的 NKU 或 NKR。

(4)AES 加解密参数

目前我们只支持 ECB 模式，AES-KEY 大小为 128-bit，支持每次四 word 的 AES 加密解密 。

```
struct aes_param {
unsigned int in_len;              //input data length (units:word)
unsigned int key_len;             //private or public key length(units:word)
unsigned int n_len;               //n length(units:word)
unsigned int out_len;             //ouput length(units:word)
unsigned int *input;              //input data buffer
unsigned int *key;                /*Ku or KR*/ buffer
unsigned int *n;                  /*N*/buffer
unsigned int *output;             //encrypted data or decrypted data buffer
unsigned int mode;                //mode: 1,encrypt 0,decrypt
};
```

### 4.13.4. 编程指导

一：打开设备并初始化 AES 控制器。
二：使用正确格式的 AES-KEY，并进行 RSA 加密。
三：使用步骤 1 中的结果，设置 AES-KEY 到 CPU。
四：执行 AES 加密或解密。
若想改变 AES-KEY，请执行步骤二，步骤三, 在步骤二中应注意“1”为初始 key 值，“0”为改变后的 key 值。

### 4.13.5. 用户 API

（1）rsa 加解密

```
int do_rsa(unsigned int fd,       //file node
unsigned int orig_aes_len,        // original AES_KEY length
unsigned int * rsa_key,           //KU or KR
unsigned int * n,
unsigned int * input,             // input_data
unsigned int *output,            //encrypted or decrypted data
unsigned int mode) ;              //mode: 0:encryption 1:decryption
```

（2）设置 AES-KEY 或改变 AES-KEY

```
int setup_aes_key(int fd,         //file node
unsigned int *key,                //encrypted AES-KEY by using RSA
int len,                          // length of key
unsigned int *nku_kr,             //NKU ok NKR(62 words)
int init_mode) ;                  //init_mode: 1:init code , 0:change code
```

（3）AES 加密解密

```
int do_aes(int fd,     //file node
unsigned int *input,     //input data
unsigned int in_len,     //input and output data length(unit:word)
unsigned int *output,    //output data(encrypted data or decrypted data)
unsigned int mode);      //mode :0,encryption 1:decryption
```

## 4.14. 语音唤醒

Voice trigger(语音休眠唤醒)主要是利用 linux 系统和君正处理器支持休眠唤醒等特点，使君正方案达到更好省电效果。

Voice trigger 的代码可划分为两部分:第一部分为语音识别测试的代码;第二部分为语音休眠唤醒的代码。

文件描述：

语音休眠唤醒固件代码：drivers/char/voice_wakeup_v13/

语音识别测试驱动代码：drivers/char/jz_wakeup_v13.c

提供给其他应用的 DMIC 驱动：drivers/char/jz_dmic_v13.c

测试用例代码：tools/wakeup-test/wakup.c

测试用例所需的语音比对文件：tools/wakeup-test/ivModel_v21.irf

linux 系统休眠后的入口文件：arch/mips/xburst/soc-x1000/common/pm_p0.c

### 4.14.1. Voice trigger 驱动配置方法

1.在“drivers/char/voice_wakeup_v13/wakeup_module/”目录下，首先执行 make clean,然后执行./mkmodule.sh

配置编译选项：

Symbol: WAKEUP_MODULE_V13 [=y]
Type : boolean
Prompt: Ingenic Voice Wakeup Module
Location:
-> Device Drivers
-> Character devices
Defined at drivers/char/Kconfig:24
Depends on: SOC_X1000 [=y]

上述选项配置完成后执行以下命令进行编译：

$ make uImage

编译完成后会在”phoenix/platform/kernel/arch/mips/boot/”目录下产生 voice trigger 需要的 uImage。

### 4.14.2. 验证方法

将内核中 tools/wakeup-test/wakeup 目录下的 wakeup 及 ivModel_v21.irf 拷贝到文件系统。在串口端输入以下命令（后台运行）：

$./wakeup ivModel_v21.irf &

```
# ./wakeup ivModel_v21.irf &
# open file[ivModel_v21.irf], ok!
open file[/dev/jz-wakeup] ok![ 2019.532480] enable wakeup function

open file[/sys/class/jz-wakeup/jz-wakeup/wakeup] ok!
read don[ 2019.546428] module open open_cnt = 1
e!!!!
#### begin read !!!!!
```

出现上述打印后对着开发板上的音频模块音频输入“灵犀灵犀”，串口出现如下打印：

```
WKUPOK[ 2026.691348] sys wakeup ok!--wakeup_timer_handler():158, wakeup_pending:1
[ 2026.705626] [Voice Wakeup] wakeup by voice.
#################ret:9, wakeup_ok
sh: input: not found
#########read ok!, wakeup ok!
#### begin read !!!!!
```

之后使开发板进入休眠状态 echo mem > /sys/power/state,休眠之后通过音频“灵犀灵犀”唤醒系统。

# 5. 烧录工具使用

烧录工具能够将 uboot,uimage 和文件系统烧录到 flash 中，具体使用方法参考文档《USBCloner-1.0 the burn tool instructions_CN.pdf》