# 君正®

# T31 开发指南

Date: May.2020

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

君正®
SDK 开发指南


Release history

| Date | Revision | Change |
|---|---|---|
| July.2019 | 1.0 | First release |
| October.2019 | 1.1 | Second release |
| January.2020 | 1.2 | Third release |
| May.2020 | 1.3 | Fourth release |

Disclaimer



北京君正集成电路股份有限公司

地址: 北京市海淀区西北旺东路 10 号院东区 14 号楼君正大厦
电话:(86-10)56345000
传真:(86-10)56345001
Http: //www.ingenic.cn

# 目录

# 1 开发资源介绍

## 1.1 SDK 介绍

ISVP SDK，即软件开发工具包，包括 API 库、开源源码、文档、Samples 等等。开发者可以通过 SDK 快速的开展产品功能开发。以下是 ISVP SDK 的内容概览图：

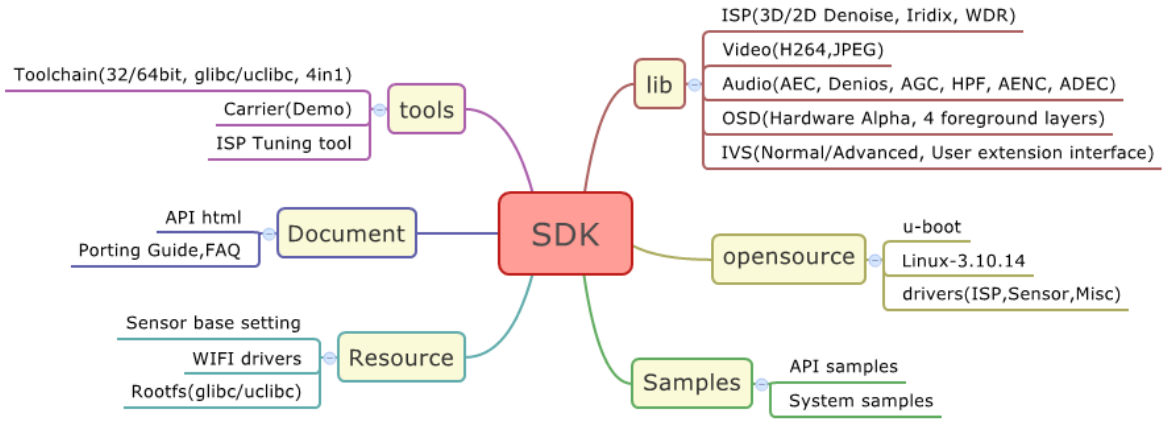



图 1-1 SDK 组成结构

## 1.2 SDK 层次结构

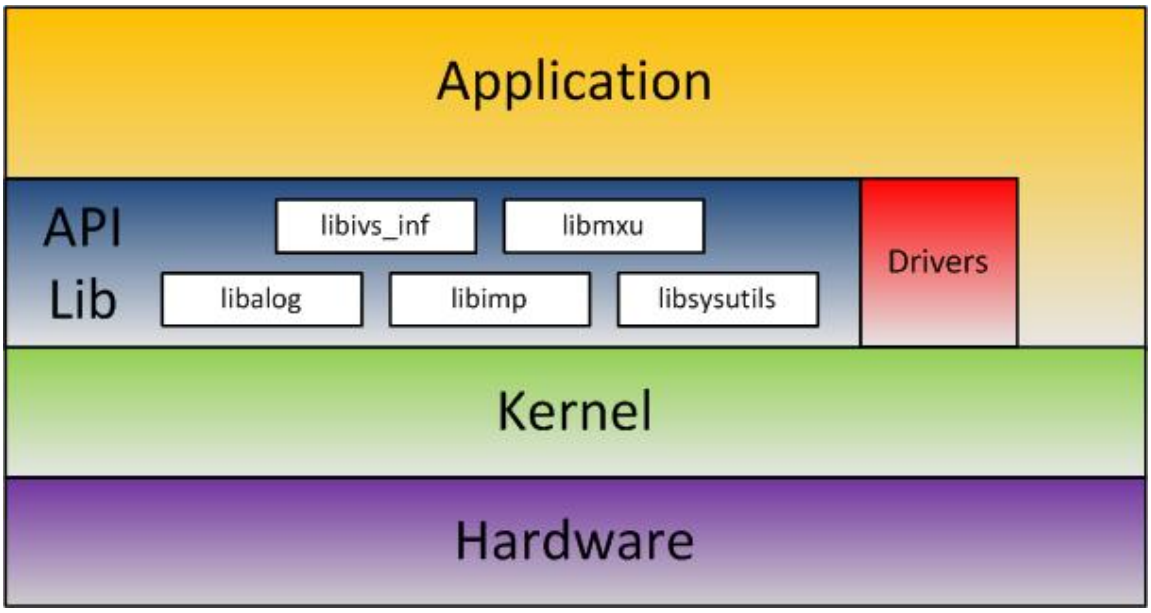
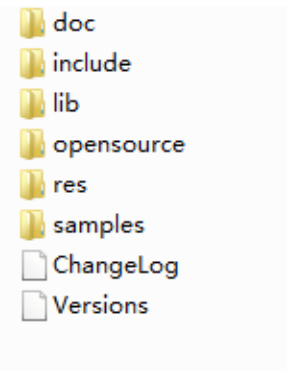



图 1-2 SDK 层次机构

- Hardware: 硬件层，完成 I/O 等具体的硬件功能。
- Linux Kernel: 内核层。完成基础的系统功能，定义硬件资源
- drivers: ko 模块驱动，可通过 driver 进行硬件操作
- API lib: 接口库，实现硬件功能的抽象，方便于应用层的开发。API 库主要有五部分：
  - libimp: 多媒体功能库。如 H264 编码，JPEG 编码，IVS 和 Audio 等
  - libsysutils: 系统功能库。如重启，设置系统时间和电池功能等
  - libalog: ISVP-SDK 的 log 实现库
  - libivs_inf: IVS 算法库，包括越线检测，周界防范等
  - libmxu: 128 位 mxu 加速指令算子库
- Application: 应用层。实现功能逻辑等
- Application 推荐使用 SDK 库提供的 API 及配合 drivers 进行开发。对于一些特殊的功能需求，也可以直接调用内核接口进行开发

## 1.3 SDK LIB




doc：指导文档
include：相关头文件
lib：相关库文件
opensource：kernel、uboot、drivers、busybox 等
res：文件镜像、效果文件等
samples：相关应用代码

# 2 开发环境搭建

## 2.1 为什么需要搭建开发环境

由于嵌入式单板的资源有限，不能在单板上运行开发和调试工具，通常需要交叉编译调试的方式进行开发和调试，即“宿主机＋目标机”的形式。宿主机和目标机一般采用串口连接显示交互信息，网口连接传输文件。

但宿主机和目标机的处理器一般不相同。宿主机需要建立适合于目标机的交叉编译环境。程序在宿主机上经过“编译－连接－定位”得到可执行文件。通过一定的方法将可执行文件烧写到目标机中，然后在目标机上运行。

## 2.2 安装 Linux 服务器

建议选择常用的 Linux 发行版，便于寻找各类技术资源。例如：

RedHat 较新的发行版如 RedHat Fedora Core 系列和 Redhat Enterprise Linux、Red。Hat 3.4.4-2。RedHat 较老的发行版如 RedHat 9.0 等。

推荐使用较新版本，以方便获取各类资源，如 Fedora Core 系列、Ubuntu12 版本以上。

## 2.3 交叉工具链

Toochain 即交叉编译工具链，是 Linux Host 机上用来编译和调试嵌入式设备程序的一系列工具的集合。ISVP 中的 Toolchain 版本信息如下：

1) gcc 版本：4.7.2
2) libc 版本：
   A. glibc 版本：2.16
   B. uclibc 版本：0.9.33.2-npt

### 2.3.1 如何安装 Toolchain

安装流程：

第一步：

安装 7z 解压工具$ `sudo apt-get install p7zip`

第二步：

根据 Host 机 CPU 位宽选择 mips-gcc472-glibc216-32bit.7z 或者 mips-gcc472-glibc216-64bit.7z 进行解压。例如：

```
$ 7z x mips-gcc472-glibc216-64bit.7z
```

第三步：

通过 export PATH=xxxx:$PATH 命令，将 toolchain 下的 bin 目录添加到 PATH 环境变量中或者在~/.bashrc 中加上下面一句永久改变

```
$ vim ~/.bashrc
$ export PATH=/opt/mips-gcc472-glibc216-64bit/bin:$PATH
```

第四步：

测试 toolchain 可执行：

```
$ mips-linux-gnu-gcc --version
mips-linux-gnu-gcc (Ingenic 2015.02) 4.7.2
Copyright (C) 2012 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is
NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

若出现如上信息则可确认 toolchain 安装正确。

### 2.3.2 交叉工具包

交叉工具链是一个软件安装包，是有很多工具的集合。有我们常用的 gcc 编译工具、objdump 反汇编工具、as 汇编器、ld 链接器、size 查看二进制文件大小等等。

```
mips-linux-uclibc-gnu-addr2line   mips-linux-uclibc-gnu-gcc-5.4.0   mips-linux-uclibc-gnu-ldd
mips-linux-uclibc-gnu-ar          mips-linux-uclibc-gnu-gcc-ar      mips-linux-uclibc-gnu-nm
mips-linux-uclibc-gnu-as          mips-linux-uclibc-gnu-gcc-nm      mips-linux-uclibc-gnu-objcopy
mips-linux-uclibc-gnu-c++         mips-linux-uclibc-gnu-gcc-ranlib  mips-linux-uclibc-gnu-objdump
mips-linux-uclibc-gnu-c++filt     mips-linux-uclibc-gnu-gcov        mips-linux-uclibc-gnu-prelink
mips-linux-uclibc-gnu-cpp         mips-linux-uclibc-gnu-gcov-tool   mips-linux-uclibc-gnu-ranlib
mips-linux-uclibc-gnu-elfedit     mips-linux-uclibc-gnu-gdb         mips-linux-uclibc-gnu-readelf
mips-linux-uclibc-gnu-execstack   mips-linux-uclibc-gnu-gfortran    mips-linux-uclibc-gnu-size
mips-linux-uclibc-gnu-g++         mips-linux-uclibc-gnu-gprof       mips-linux-uclibc-gnu-strings
mips-linux-uclibc-gnu-gcc         mips-linux-uclibc-gnu-ld          mips-linux-uclibc-gnu-strip
```

# 3 T31 必读

## 3.1 Uboot 编译

- Uboot 编译流程：

u-boot 可单独编译，不依赖其他代码。T31 u-boot 的板机配置文件位于 include/configs/isvp_t31.h。默认编译配置文件介绍如表 3-1。

表 3-1 T31 芯片对应 uboot 编译文件

| 芯片型号 | 编译命令 | 说明 |
|---|---|---|
| T31N | make isvp_t31_sfcnor | 表示编译 norflash 启动的 uboot，针对 T31N 芯片 |
| T31L | make isvp_t31_sfcnor_lite | 表示编译 norflash 启动的 uboot，针对 T31L 芯片 |
| T31X | make isvp_t31_sfcnor_ddr128M | 表示编译 norflash 启动的 uboot，针对 T31X 芯片 |
| T31A | make isvp_t31a_sfcnor_ddr128M | 表示编译 norflash 启动的 uboot，针对 T31A 芯片 |
| T31N | make isvp_t31_msc0 | 表示编译 SD 卡启动的 uboot，针针对 T31N 芯片 |
| T31L | make isvp_t31_msc0_lite | 表示编译 SD 卡启动的 uboot，针针对 T31L 芯片 |
| T31X | make isvp_t31_msc0_ddr128M | 表示编译 SD 卡启动的 uboot，针针对 T31X 芯片 |
| T31A | make isvp_t31a_msc0_ddr128M | 表示编译 SD 卡启动的 uboot，针针对 T31A 芯片 |

编译步骤：

**第一步：** `$ make distclean` 清除旧配置

**第二步：** `$ make isvp_t31_xxx` 根据对应芯片类型(芯片型号查看《芯片版本说明》)，编译对应的 uboot，生成对应的 u-boot-with-spl.bin

- Uboot 配置文件中常见修改的地方：

1) CONFIG_BOOTARGS，主要修改点是内核启动以后的内存配置，分区大小配置。（注：mem 表示内核启动以后保留内存，rmem 表示预留给 SDK 的内存（包括 ISP 模块的内存）；两者相加为芯片真实内存大小；具体大小可参考代码）。

2) CONFIG_BOOTCOMMAND，配置 uboot 启动执行的命令。例如：norflash 启动模式下添加 sd 卡启动的命令，

`"sf probe;sf read 0x80600000 0x40000 0x280000; bootm 0x80600000"` 改为 `"mmc read 0x80600000 0x1800 0x3000; bootm 0x80600000"`。

3) CONFIG_BOOTDELAY，配置 uboot 的等待时间。

4) 需要添加新的 norflash 芯片的支持。

5) uboot 中添加密码功能：

修改配置文件，修改 isvp_t31.h 中添加如下内容：

`#define CONFIG_AUTOBOOT_KEYED` // 必配

`#define CONFIG_AUTOBOOT_STOP_STR “123456”` //必配，uboot 设置的密码。

`#define CONFIG_AUTOBOOT_PROMPT “Press xxx in %d second”` // bootdelay,选配，uboot 提示信息。

`#define CONFIG_AUTOBOOT_DELAY_STR “linux”` //选配，

uboot 提示信息代码的具体实现在 common/main.c 中 `abortboot_keyed(int bootdelay)`；可以根据自己的需要具体改动。

6) SD 卡升级问题

在 isvp_t31.h 中添加 `#define CONFIG_AUTO_UPDATE` 定义。具体代码实现在 common/cmd_sdupdate.c 中。需要注意点：`LOAD_ADDR` 表示把 SD 卡上的相应内存加载到内存的位置。程序中默认设置为 0x82000000，由于这个地址位于 uboot 的堆上，常见 uboot 的堆大小的设置在 isvp_t31.h 中 `CONFIG_SYS_MALLOC_LEN` 宏的配置；所以这个地址能够被使用的大小将受到堆大小的限制，还有 uboot 代码中 malloc 空间的限制。（注意：需要读取较大文件的时候，可以适当增大 `CONFIG_SYS_MALLOC_LEN`）

7) 编译出的 uboot 大于限制的大小处理方法

uboot 代码中默认限制 spl 部分为 26Kbytes，uboot 部分为 214Kbytes；一共 240Kbytes 大小的限制。如果 uboot 编译生成的 u-boot-with-spl.bin 文件大于 240Kbytes，则无法启动。

解决方案一：

增加 uboot 的限制；修改 isvp_t31.h 中 `CONFIG_SYS_MONITOR_LEN` 宏的定义；同时修改 `CONFIG_BOOTARGS` 变量中 boot 分区的大小及以后分区的偏移地址。

解决方案二：

如果生成的 u-boot-with-spl.bin 超出 240Kbytes 较少可以采取压缩 uboot 的方式。在 isvp_t31.h 中 修改`#undef CONFIG_SPL_LZOP` 为 `#define CONFIG_SPL_LZOP`；然后重新编译烧录的文件名 u-boot-lzo-with-spl.bin

8) uboot 网络问题

默认的 isvp 配置是包含以太网部分的代码，如果产品在 uboot 阶段不需要 TFTP 下载或者 NFS 挂载，可以把以太网部分代码裁剪掉，以便减小 uboot。

具体操作：打开 isvp_T31.h 配置文件，把`#define CONFIG_CMD_NET` 宏注掉即可。

## 3.2 Kernel 编译

kernel 可单独编译，不依赖其他代码。以 isvp 板级编译为例，进入 kernel 源码目录。在 arch/mips/configs/ 文件夹下存放了内核的板机配置文件；T31 芯片板级文件为 isvp_swan_defconfig；

**第一步：**使用相对配置好的板级文件 `$ make isvp_swan_defconfig`

**第二步：**根据需求选择性编译 `$ make menuconfig`

**第三步：**编译内核文件 `$ make uImage`

如果报错，执行 `$ make distclean`

然后从第一步重新开始。

注意：对于 kernel 之外的 ko 编译，依赖 kernel，且 kernel 必须经过编译。每当 kernel 更改之后，可能会出现 ko 无法 insmod 的情况，或者可以 insmod 但会出现未知错误。这是因为 kernel 重新编译后，函数 Symbol 表有变化，需要重新编译 ko driver 以正确 link 函数。此时需要重新编译 ko。

内核默认的 config 留有一定余量，而实际产品往往需要进行内核裁减以释放更多的空间。

以下是几个常用可供裁减的选项及说明：

1) CONFIG_NETWORK_FILESYSTEMS：网络文件系统，一般用来方便开发，但会占用较多空间，也可用 tftp 进行替代。若不支持 NFS，可以 Disable

2) CONFIG_KALLSYMS &CONFIG_KALLSYMS_ALL：内核函数符号表，会占用较多空间，在 panic 时的函数栈可以显示出函数名。当内核稳定后，可以考虑 Disable 此功能。但建议完全稳定之前使能此功能。

3) 其他文件系统。一般文件系统会占用较多空间，开发者可根据需求对文件系统的选项进行裁减，比如 ext 文件系统等

4) 和产品定义无关的模块可以裁掉，比如 USB，以太网，TF 卡等等。

注：内核的深度裁减有一定的技术难度，开发者尽量在深入了解配置的情况下再进行深度裁减。

## 3.3 T31 需要加载的驱动

| 驱动名称 | 驱动介绍 |
|---|---|
| tx-isp-t31.ko | ISP 驱动 |
| sensor_xxxx_t31.ko | Sensor 驱动 |
| avpu.ko | 视频编码驱动 |
| sinfo.ko | Sinfo 探测驱动 |
| audio.ko | 音频驱动 |

## 3.4 T31 与其他 T 系列芯片使用区别及注意事项

T31 与 T30/T21 使用上的主要区别有：

1) FrameSource 模块增加 fcrop 参数，实现前级裁剪功能。T30/T21 的流程是缩

放--->裁剪，T31 的流程是裁剪--->缩放-->裁剪。（具体流程请参照 6.3.1 节）

2) 图像编码参数不同，T31 编码通道参数与 T30/T21 的参数比较有变化，在原有的编码器属性 encAttr 和码率控制器属性 rcAttr 基础上增加 gop 属性 gopAttr。（具体变化请参照 7.5.5 节）

# 4 文件系统的制作

## 4.1 根文件系统介绍

Linux 的目录结构的最顶层是一个被称为“.”的根目录。系统加载 Linux 内核之后，就会挂载一个设备到根目录上。存在于这个设备中的文件系统被称为根文件系统。所有的系统命令、系统配置以及其他文件系统的挂载点都位于这个根文件系统中。

根文件系统通常存放于内存和 Flash 中，或是基于网络的文件系统。根文件系统中存放了嵌入式系统使用的所有应用程序、库以及其他需要用到的服务。下图列出了根文件系统的顶层目录。

```
.                    //根目录
├─── bin             //基本命令的可执行文件
├─── dev             //设备文件
├─── etc             //系统配置文件，包括启动文件
├─── lib             //基本库文件，例如 C 库和内核模块 ko
├─── mnt             //临时文件系统挂载点
├─── opt             //添加的软件包
├─── proc            //内核以及进程信息的虚拟文件系统
├─── root            //root 用户目录
├─── sbin            //系统管理的可执行程序
├─── sys             //系统设备和文件层次结构
├─── system          //可读写系统文件(jffs2)
├─── tmp             //临时文件
├─── usr             //包含一些有用的文件
└─── var             //存放系统日志或一些服务程序的临时文件
```

## 4.2 利用 Busybox 制作根文件系统

利用 busybox 制作根文件系统需要先获取 busybox 源代码，然后配置、编译和安装 busybox，操作成功后开始制作根文件系统。

### 4.2.1 获取 Busybox

成功安装 SDK 后，busybox 完整源代码就存放在 /opensource/busybox/目录中。要获取 busybox 源代码也可以从网站 http://www.busybox.net 下载相同版本的源码。

### 4.2.2 配置和编译 Busybox

busybox 与 kernel 的配置方法类似，需要先 make defconfig 再 make，之后 make install 会默认把安装文件生成在 busybox/_install 目录下。SDK 提供的 busybox 中有配置好的 defconfig 文件。

文件系统有 glibc 与 uclibc 两种选择，glibc 的特点是支持功能全面，但是占用存储稍多；uclibc 的特点是占用存储空间小于 glibc，但是支持功能比 glibc 稍少。开发者应该根据自己实际情况选用适合的 libc 方式。

对于一般的应用场景，推荐客户使用 uclibc 搭建系统。

```
isvp_uclibc_defconfig              --uclibc 默认 config
isvp_uclibc_mini_defconfig         --uclibc 默认裁剪 config
isvp_glibc_defconfig               --glibc 默认 config
isvp_glibc_mini_defconfig          --glibc 默认裁剪 config
```

如果用户想修改配置文件，可以执行 make menuconfig 自行配置。busybox 的配置界面和内核配置相似，其功能选项容易理解，可以根据自己的需求选择配置。

编译具体操作如下：

```
make isvp_uclibc_defconfig
make ; make install
```

## 4.3 制作根文件系统

成功安装 SDK 后，在 res/rootfs_xxx/目录中存放已制作好的根文件系统，可以跳过文件系统的制作，直接使用 SDK 中提供的文件系统。如果用户如果有特殊需求，可在 busybox 的基础上制作根文件系统。

制作根文件系统的具体操作步骤如下

**第一步：**建立根文件系统文件 rootfs

```
# mkdir rootfs
# cd rootfs
# cp -R opensource/busybox/_intsall/* .
# mkdir etc dev lib tmp var mnt system proc
```

**第二步：**配置 etc、lib、dev 等目录的必需文件(可以参考 SDK 中的文件系统)。

etc 目录可参考系统/etc 下的文件。其中最主要的文件包括 inittab、fstab、init.d/rcS 文件等，这些文件最好从 busybox 的 examples 目录下拷贝过来，根据需要

自行修改。

dev 目录下的设备文件，可以直接从系统中拷贝过来或者使用 mknod 命令生成需要的设备文件。拷贝文件时请使用 cp –R file。

lib 目录是存放应用程序所需要的库文件，根据应用程序需要拷贝相应的库文件。

### 4.3.1 文件系统的选择

对于 NorFlash Based 系统，存储空间一般较小，因此会选用压缩文件系统。推荐以下两种文件系统：

A. squashfs：只读文件系统，压缩率高

B. jffs2：可读写文件系统，可选择压缩方式

推荐的系统搭建的方案是----系统 rootfs 以及不需要经常修改的系统分区采用 squashfs 文件系统，而配置分区 system 等需要经常读写的分区采用 jffs2 文件系统。

### 4.3.2 Squashfs 文件系统

squashfs 文件系统是一套基于 Linux 内核使用的压缩只读文件系统，压缩率高。squashfs 具有如下特点：

- 数据(data)，节点(inode)和目录(directories)都被压缩
- 保存了全部的 32 位 UID/GIDS 和文件的创建时间
- 最大支持 4G 文件系统
- 检测并删除重复文件

squashfs 制作方法为：

```
#   ./mksquashfs rootfs ./rootfs.squashfs.img -b 64K –comp xz
```

其中，mksquashfs 为制作 squashfs 文件系统工具可网上下载；rootfs 是之前已经制作好的根文件系统，rootfs.squashfs.img 是生成的 squashfs 文件系统映像文件。-b 64K 指定 squashfs 文件系统的块大小为 64K（决定于实际 spi flash 块大小）。-comp 指定文件系统压缩方式为 xz。请根据实际情况修改参数。

### 4.3.3 jffs2 文件系统

jffs2 是 RedHat 的 David Woodhouse 在 jffs2 基础上改进的文件系统，是用于微型嵌入式设备的原始闪存芯片的实际文件系统。jffs2 文件系统是日志结构化的可读写的文件系统。jffs2 的优缺点如下：

**优点:**

使用了压缩的文件格式。最重要的特性是可读写操作。

**缺点:**

jffs2 文件系统挂载时需要扫描整个 jffs2 文件系统，因此当 jffs2 文件系统分区增大时，挂载时间也会相应的变长。使用 jffs2 格式可能带来少量的 Flash 空间的浪费。这主要是由于日志文件的过度开销和用于回收系统的无用存储单元，浪费的空间大小大致是若干个数据段 jffs2 的另一缺点是当文件系统已满或接近满时，jffs2 运行速度会迅速降低。这是因为垃圾收集的问题。

jffs2 的制作方法为：

```
# ./mkfs.jffs2 -d ./rootfs -l -e 0x8000 -s 0x40000 -o rootfs.jffs2.img
```

其中，mkfs.jffs2 工具可以从互联网中下载。rootfs 为之前已经制作好的根文件系统；rootfs.jffs2.img 为输出的镜像文件。

参数说明：

```
-d 指定根文件系统; -l little-endian 小端模式;
-e flash 的块大小; -o 输出镜像文件; -s 页大小;
```

注意：jffs2 制作某个分区为 jffs2 文件，分区大小必须为 nor erase_size 的整数倍，这个是官方驱动的要求。

## 4.4 Demo rootfs 简单说明

ISVP 的 Demo 分区方式为：

```
A. u-boot:256K
B. uImage:2560K
C. rootfs:2048K
D. system:3328K
```

在 ISVP 的文件系统中，有两个指定的文件路径：

```
/etc/sensor/ ——此目录下存放 Sensor 的效果 bin 文件
/etc/webrtc_profile.ini ——回音消除参数文件
```

不同的产品可能以上两处会有不同，而且产品升级时也可能升级相关的参数文件，因此建议将以上两处做软连接到可读写或者可更新的路径下。

Demo 的 rootfs 在启动后，会探测 system 分区是否可用，如果不可用，会进行格式化，并创建相关的目录结构。因此，第一次制作 Demo 系统，可以将整片 Flash 擦除，并烧录 u-boot， uImage，rootfs 即可；system 分区会自动创建。

system 分区的结构如下：

```
system
├─── bin   --此目录下放置应用程序
├─── etc
│     └─── sensor   --/etc/sensor 软链接到这里
│           └─── jxh42.bin  --效果文件
├─── init
│     └─── app_init.sh    --开机执行初始化脚本
└─── lib
     ├─── firmware   --/lib/firmware 软链接到这里
     └─── modules    --/lib/modules 软链接到这里
          ├─── sensor_jxh42.ko   --sensor 驱动
          └─── tx-isp.ko    --ISP 驱动
```

注：Demo rootfs 可作为参考，开发者可以根据产品的实际情况定义 rootfs 方案。

## 4.5 应用程序编译

### 4.5.1 编译注意

应用程序编译注意有以下几点：

1) 关于 glibc 和 uclibc 编译的区分方法请参考《Toolchain 使用说明》

2) 关于 API 库的链接顺序：[IVS 库] [mxu 库] [libimp/libsysutils] [libalog]

3) 由于 libimp 中依赖 C++库，因此需要使用 mips-linux-gnu-g++进行链接，若使用 gcc 链接，需要手动添加 `LD_FLAGS+=stdc++`

4) 如何优化 elf 文件的大小：

A. 编译等级选择 O2：`C_FLAGS/CXX_FLAGS += -O2`

B. `DFLAG += -Wl,-gc-sections`，不链接不必要的段。

C. elf 文件执行 mips-linux-gnu-strip，链接后删除不必要的段。

5) 若系统中有多个 elf 文件需要链接库文件，可使用动态链接的方式，若只有一个文件链接库文件，请使用静态链接的方式（注，libimp 相关功能在系统中只能存在一份实例）。在调试时选择动态链接的方式可以方便的进行 debug 及问题反馈。

### 4.5.2 运行应用程序

要运行编译好的应用程序，首先需要将其添加到目标机中，然后完成以下工作：

将应用程序和需要的库文件（如果有）等添加到目标机的根文件系统相应的目录中。通常将应用程序放到/bin 目录里，库文件放到/lib 目录里，配置文件则放到/etc 目录里。

制作包含新应用程序的根文件系统。

注意：如果执行应用程序，需要读写文件系统操作。请选择 squashfs、jffs2 文件系统。如果新添加的应用程序需要系统启动后自动运行，请在编写文件系统时，编辑/etc/init.d/rcS 文件，添加需要启动的应用程序路径。

# 5 系统启动与烧录

## 5.1 系统启动

一个全新的板子里面什么都没有(准确的说芯片里面有固化的 Bootloader)，只是一个冷冰冰的机器，怎么让这个机器变得有灵魂呢？看下面介绍。

### 5.1.1 Uboot 启动

#### 5.1.1.1 SD 卡启动

SD 卡插上读卡器就可以在 PC 上操作，所以把 SD 卡分出一个新分区，把编译好的 uboot 拷贝到这个分区上的固定偏移位置上，然后指导 Bootloader 指向 SD 卡的这个位置，启动 uboot（SD 卡启动卡制作查看附录 9.1）。

#### 5.1.1.2 Nor/Nand 启动

nor/nand 是板子上的存储介质，本质上和上面 SD 卡里的存储介质类似，但这些存储器是焊接到板子上的，不能直接拿下来操作，不方便操作，但这些闪存是 EEPROM 掉电也不丢失数据，所以通过把 uboot 拷贝到 nor/nand 闪存中，再次启动直接读取 flash 中的 uboot 分区，就不需要每次都重新烧录了。

## 5.2 系统烧录

### 5.2.1 TFTP 传输与烧录

1) 目标

将文件通过 tftp 方式，从 PC 端，下载到 Uboot 的内存中，然后写到 flash 中。

2) 前提

硬件：开发板上网卡；板子通过网线连接到路由器或者交换机上，PC 也连到该路

由器或者交换机上，并且处于同一网段上；

软件：PC 端安装并设置好 TFTP 服务，把相应的 u-boot 等文件放到 TFTP 根目录下；uboot 中支持网卡驱动并且支持相应的 TFTP 操作。

3) 操作

在 uboot 中，执行 `tftp mem_addr file_name`；可以将文件 file_name 传送到 Uboot 的内存地址 mem_addr 中了。

4）传输实例

通过 tftp load 文件到内存；

```
$ tftp 0x80600000 u-boot-with-spl.bin
$ tftp 0x80640000 uImage
$ tftp 0x808c0000 root-uclibc-1.1.squashfs
```

分别把 u-boot-with-spl.bin、uImage、root-uclibc-1.1.squashfs 下载到内存的 0x806000000、0x80640000、0x808c0000 处。

5) 烧录到 Nor

把下载到内存上的文件，写到开发板上的 nor flash 上，重启后可以直接从 nor 上直接读取文件，而不必每次手动 load 文件到内存上。烧录命令如下：

```
$ sf probe;sf erase 0x0 0x1000000;sf write 0x80600000 0x0 0x1000000
```

- `sf probe` 在使用 sf read、sf write 之前,一定要调用 sf probe
- `sf erase` 擦除指定位置，指定长度的 flash 内容，擦除后内容全 1；
- `sf write` 写内存数据到 flash 上。

### 5.2.2 SD 卡传输与烧录

1) 目标

将某个文件写到内存中，然后写到 flash 上。

2) 前提

硬件：开发板上有 SD 卡槽；有读卡器把 SD 卡插到 PC 上拷贝数据。

软件： uboot 中支持 SD 卡驱动。

3) 操作

在 uboot 中，执行 `fatls mmc 0` 查看 SD 卡中的文件

`fatload mmc 0 mem_addr file_name` 可以将文件 file_name 传送到 Uboot 的内存地址 mem_addr 中了。

4）传输实例

通过 tftp load 文件到内存；

```
$ fatload mmc 0 0x80600000 u-boot-with-spl.bin
$ fatload mmc 0 0x80640000 uImage
$ fatload mmc 0 0x808c0000 root-uclibc-1.1.squashfs
```

分别把 u-boot-with-spl.bin、uImage、root-uclibc-1.1.squashfs 下载到内存的 0x806000000、0x80640000、0x808c0000 处。

5) 烧录到 Nor

把下载到内存上的文件，写到开发板上的 nor flash 上，重启后可以直接从 nor 上直接读取文件，而不必每次手动 load 文件到内存上。烧录命令如下：

```
$ sf probe;sf erase 0x0 0x1000000;sf write 0x80600000 0x0 0x1000000
```

- `sf probe` 在使用 sf read、sf write 之前,一定要调用 sf probe
- `sf erase` 擦除指定位置，指定长度的 flash 内容，擦除后内容全 1；
- `sf write` 写内存数据到 flash 上。

## 5.3 串口连接

开发板串口负责交互信息传输，内核打印会通过串口发送到显示屏上，开发板通过串口接收键盘的输入信息。常用的串口工具有 putty、Xshell 等软件

PC 端串口与开发板串口的连接，需要配置对应的参数，如串口号，波特率，奇偶校验，数据位，停止位等。比如配置串口号为 com6，波特率为 115200，8 位数据位，一位停止位，无奇偶校验。

## 5.4 Uboot 配置

uboot 最终目的是启动加载内核。内核的启动需要 uboot 的引导并把相应的环境变量传到 kernel，此外还需要挂载一个根文件系统，存放一些内核配置信息和执行一些命令操作。

### 5.4.1 Uboot 命令学习

- reset：重新启动嵌入式系统。
- printenv：打印当前系统环境变量。
- setenv：设置环境变量，格式：setenv name value，表示将 name 变量设置成 value 值；如果没有这个参数，表示删除该变量。
- saveenv：保存环境变量到 flash 中。
- sleep：延迟执行，格式：sleep N，可以延迟 N 秒钟执行。
- run：执行环境变量中的命令，格式：run var，可以跟几个环境变量名。
- cp：在内存中复制数据块，格式：cp source target count，第一个参数是源地址，第二个参数是目的地址，第三个参数是复制数目。
- tftpboot：通过 tftp 协议下载文件到内存，格式 tftp target source。
- fatls：显示 SD 卡的文件，格式 fatls mmc 0 。
- bootm：可以引导启动存储在内存中的程序映像。格式：bootm addr1 addr2，第一个参数是程序映像的地址，第二个参数一般是 ramdisk 地址。

详细介绍可以在 uboot 命令行输入 help 查看。

### 5.4.2 Uboot 环境变量

板子上电启动，进入 uboot 命令行（读秒时按下 Enter 键）；通过 uboot 命令 printenv 可以查看 uboot 的环境变量：

```
isvp_t31# printenv
bootargs=console=ttyS1,115200n8 32M@0x0 rmem=32M@0x2000000 init=/linuxrc
rootfstype=squashfs root=/dev/mtdblock2 rw
mtdparts=jz_sfc:256k(boot),2560k(kernel),2048k(root),-(appfs)
bootcmd=sf probe;sf read 0x80600000 0x40000 0x280000; bootm 0x80600000
bootdelay=1
ipaddr=193.169.4.151
serverip=193.169.4.2
```

参数介绍：

- `setenv` 是用来设置环境变量的命令
- `bootargs` 内核通过 bootargs 找到文件系统
- `console=ttyS1,115200n8` 设置串口号为 ttyS1，波特率为 115200
- `mem=32M@0x0 rmem=32M@0x2000000` 分配内存
- `init=/linuxrc` 系统首先运行/linuxrc 文件
- `rootfstype=squashfs` 使用压缩只读文件系统 squashfs
- `root=/dev/mtdblock2 rw` 告诉内核根文件从 mtd 分区 3 挂载，可读写
- `mtdparts=jz_sfc:256k(boot),2560k(kernel),2048k(root),-(appfs)` mtd 分区大小
- `bootcmd` 启动命令
- `sf probe;sf read 0x80600000 0x40000 0x280000; bootm 0x80600000` 从 nor 启
- `ipaddr 193.169.4.151` 板子的 IP
- `serverip=193.169.4.2` 服务器 IP 地址

### 5.4.3 更改 mtd 分区大小

mtd 分区大小，在 uboot 中以命令行参数的形式传入 kernel，配置文件在 include/configs/isvp_t20.h 中，对于 8M Flash，找到 CONFIG_SFC_NOR，参考平台定义其中：

```
mtdparts=jz_sfc:256k(boot),2560k(kernel),2048k(rootfs),-(appfs)
```

就是 MTD 的分区情况，共 4 块分区，uboot 256k，kernel 2.5M，rootfs(squashfs)2M。-(appfs)具有自适应功能，当 Flash 为 8M 时，appfs 分区会自动分配成 3.25M；当 Flash 为 16M 时会自动分配成 11.25M。如果想修改，请保持总大小不变的情况下，按如上格式修改。一般 uboot、kernel、rootfs 大小固定不变，建议用户不要修改，可以调整 appfs 分区大小。

### 5.4.4 更改 rmem 内存大小

rmem 大小同样是在 include/configs/isvp_t20.h 中修改，参考平台定义如下

```
#define BOOTARGS_COMMON "console=ttyS1,115200n8 mem=32M@0x0
rmem=32M@0x2000000
```

其中：`mem=32M@0x0 rmem=32M@0x2000000` 就是对 mem 的相关设置

`mem=32M@0x0`，从 0 地址开始给系统分配 32M 内存，

`rmem=32M@0x2000000`，从 0x2000000（32M）地址开始给 SDK 分配 32M；

T31 内嵌一块 64M 的内存，如要修改 rmem，请在总大小 64M 不变情况下按格式

进行修改。示例：

假设 rmem 减小 4M，mem 增大 4M

如果想减小 rmem，那么 mem 应该增大。所以 mem 大小为 36M，从 0 地址开始。rmem 为 28M，地址应该从 36M 开始。。

```
    #define BOOTARGS_COMMON "console=ttyS1,115200n8 mem=36M@0x0
rmem=28M@0x2400000"
```

# 6 系统资源使用与调试

## 6.1 ISP_Sensor

ISP 通过一系列数字图像处理算法完成对数字图像的效果处理。主要包括 3A、坏点校正、去噪、强光抑制、背光补偿、色彩增强、镜头阴影校正等处理。

### 6.1.1 ISP 驱动

1) 进入驱动文件夹/opensource/drivers/isp-t31/tx-isp-t31；首先修改 Makefile 中 `ISVP_ENV_KERNEL_DIR` 宏定义，使之能够索引到正确的 kernel 路径。

2) 然后进行执行 `make clean; make`

3) 最后生成的 tx-isp-t31.ko 拷贝到系统中。

4) ISP 驱动注册时提供 module_param 参数有 isp_clk 参数，其参考配置：例如 3M@25fps 使用 125Mhz，

```
$ insmod tx-isp-t31.ko isp_clk=150000000
```

Sensor 分辨率、帧率与 ISP 时钟对应如下：

```
2M@30fps ----> 100000000 (100Mhz)
3M@25fps ----> 150000000 (150Mhz)
4M@25/30fps,5M@25fps -----> 200000000 (200Mhz)
```

如果按照以上配置，ISP 仍有报错，如 fifo overflow 等，可以直接将 ISP 时钟提到 200Mhz，看是否还有此错误，如果错误消失，可以在中间找一个合适的值。

### 6.1.2 Sensor 驱动

1) sensor 驱动依赖于 kernel 和 ISP 驱动，所以在编译 sensor 驱动之前 kernel 和 ISP 驱动必须已经编译完成。

2) 进入具体的 sensor 驱动文件夹，首先修改 Makefile 中 `ISVP_ENV_KERNEL_DIR` 和 `ISP_DRIVER_DIR` 宏定义，使之能够索引到正确的 kernel 和 ISP 驱动路径。

3) 然后进行执行 `make clean;make`

4) 最后将生成的 sensor_xxx.ko 拷贝到系统中。

5) sensor 驱动注册时提供了多个 module_param 可配置参数，如果产品硬件遵循参考设计 insmod 时无需跟参数，使用默认值即可。

A. reset，power_down 的配置：加载模块时加入参数，例如：

```
$ insmod sensor_xx.ko reset_gpio=18 pwdn_gpio=20
```

其中，gpio 的值为 GPIO 编号，规则为：num = 32 * n + bit，例如：PA18 的 GPIO 编号为 18，PC20 的 GPIO 编号为 84

B. DVP 数据端口配置：加载模块时加入参数，例如：

```
$ insmod sensor_xx.ko sensor_gpio_func=1
```

其中 sensor_gpio_func 为 DVP Port 的配置选项，0：PA Low-10bit，1：PA High-10bit，2：PA 12bit。

C. Sensor 的数据接口有多种，目前支持 DVP 以及 MIPI CSI-2 两种接口；但有的 senosr 可能同时支持两种数据接口，为了区分加入了模块参数 data_interface，例如：

```
$ insmod sensor_xx.ko data_interface=1
```

其中，data_inteface 为 sensor 数据接口配置选项 1：MIPI　　2：DVP

D. 基于降低功耗的考量，在 sensor 驱动中加入了配置最高帧率的参数 sensor_max_fps。一般 sensor 驱动中支持 30/25fps 与 15fps 的切换。例如：

```
$ insmod sensor_xx.ko sensor_max_fps=15
```

其中，sensor_max_fps 为 sensor 帧率配置选项。15:配置为 15fps， 25: 配置为 25fps，默认为 30/25fps 模式。

### 6.1.3 Avpu 视频编码驱动

1) 进入驱动文件夹 /opensource/drivers/avpu/；首先修改 Makefile 中 `ISVP_ENV_KERNEL_DIR` 宏定义，使之能够索引到正确的 kernel 路径，视频编码驱动只依赖内核，所以在编译 avpu 驱动之前 kernel 必须已经编译完成。

2) 然后进行执行 `make clean;make`

3) 最后生成的 avpu.ko 拷贝到系统中。

4) avpu 驱动注册时提供了多个 module_param 可配置参数，如果产品硬件遵循参考设计 insmod 时无需跟参数，使用默认值即可。例如：

```
$ insmod avpu.ko clk_name='vpll' avpu_clk=400000000
```

其中 clk_name 是 vpu 选择使用的时钟源，avpu_clk 用于设置 avpu 的时钟频率。

### 6.1.4 Sensor 探测识别驱动

使用情况同一个开发平台支持多个 sensor 的情况。可以调用该驱动先探测 sensor 型号，然后加载正确的 sensor 驱动进行后续操作。

Sensor 识别驱动位于/opensource/drivers/misc/sensor_info 目录下，用户可以通过 ioctl 或者 proc 接口进行 Sensor 型号的查询。使用方法可参考例子 sample_sinfo.c。

### 6.1.5 图像效果文件

不同的 Sensor 以及镜头可能需要不同的效果参数配置，配置文件位置为：/etc/sensor 目录下，文件名为[sensor]-t31.bin，例如：ov9732-t31.bin。如果没有这个文件图像颜色等可能不正常。实际产品中效果 bin 文件往往需要随版本迭代更新，因此

/etc/sensor 目录需要有读写权限。可供参考的方法之一是将/etc/sensor 目录做成一个软链接，链接到一个 rw 的分区中，这样就可以在版本更新时单独更新 bin 文件了。

### 6.1.6 Sample 的使用

Sample 程序位于/samples/libimp-samples，里面是依赖 SDK 库的应用程序，包括图片的抓取、图片格式的转换、智能检测、录音放音、回应消除等应用程序。设置好 Makefile，直接在 sdk 目录下 make 即可以编译。用户可以参考提供的 sample 程序，编写自己相应工程代码。

完成编译后，可以在目录下找到对应的可执行程序；拷贝到开发板上测试即可。详细操作查看下表；

表 5-1 sample 功能

| 应用文件 | 功能 | 执行命令 | 执行结果 |
|---|---|---|---|
| sample-Ai.c | 录音 | ./sample-Ai | 生成录音文件 ai_record.pcm |
| sample-Ao.c | 音频文件播放 | ./sample-Ao | 播放采样率 16K 的音频文件 ao_paly.pcm |
| sample-Ai-AEC.c | 音频回声消除 | ./sample-Ai-AEC | 程序运行后，生成了 test_for_play.pcm 和 test_aec_record.pcm 两个录音文件 |
| sample-Ai-Ref.c | 验证音频获取回声消除算法参考帧 | ./sample-Ai-Ref | 程序运行后，生成了 test_for_play.pcm 和 test_aec_record.pcm 两个录音文件 |
| sample-dmic.c | 数字麦克阵列录音 | ./sample-dmic | 程序运行后，录音生成 dmic0_record.pcm ，dmic1_record.pcm，分别对应麦克阵列中的 MIC0，MIC1 录制的声音数据 |
| sample-Capture-nv12.c | 抓取 NV12 图片 | ./sample-Capture-nv12 | 会在/tmp 下面产生抓取的 NV12 的图片 |
| sample-Decoder-jpeg.c | 把 jpeg 解码成 yuv | ./ sample-Decoder-jpeg | 会在/tmp 下面生成 yuv 文件 |
| sample-Encoder-jpeg.c | 把 NV12 图片编码成 | ./ sample-Encoder-jpeg | 把 NV12 图片编码成 jpeg 图片，并保 |

| | | | |
|---|---|---|---|
| | jpeg 图片 | | 存在/tmp 下面 |
| sample-Framesource.c | 保存对应格式的视屏 | ./ sample-Framesource | 保存对应格式的视屏（nv12 ,bgr0,raw） |

## 6.2 Audio

音频分为内部 codec 和外部 codec，内部 codec 又分数字 mic 和模拟 mic。

### 6.2.1 内核配置

在内核源码根目录下执行 make menuconfig 命令进入配置界面，按需求进行下面配置。

**1) Amic 配置(默认已配置)**

```
Device Drivers --->
     <*>Sound card support --->
          <*>Open Sound System --->
               [*]Open Sound System of Xburst --->
                    [*]Compile jzsound driver into ko
                    [*]  Jz On-Chip I2S driver
                    [*]  xburst internal coedc  --->
                            [*]  t10 internal codec
```

**2) Dmic 配置**

```
Device Drivers --->
     <*>Sound card support --->
          <*>Open Sound System --->
               [*]Open Sound System of Xburst --->
                    [*]Compile jzsound driver into ko
                    [*]  Jz On-Chip I2S driver
                    [*]  xburst internal coedc  --->
                           [*] t10 internal codec
                    [*]  jz soc t31 dmic enable
```

**3) 外部 Codec ES8374 配置**

```
Device Drivers --->
     <*>Sound card support --->
          <*>Open Sound System --->
               [*]Open Sound System of Xburst --->
```

```
[*]Compile jzsound driver into ko
[*]  Jz On-Chip I2S driver
[*]  xburst internal coedc  --->
      [*] t10 internal codec
[*]  jz soc t31 dmic enable
[*]  xburst external codec  --->
       <*> JZ_v12 i2c controler 2 Interface support
```

### 6.2.2 Audio 驱动

驱动代码位于/opensource/drivers/audio/oss2；在编译前需要确认 Makefile 中的 kernel 路径正确性。驱动参数使用如下。

1) Audio 驱动加载时提供了参数 spk_gpio 用于配置功放的使能，驱动默认配置的是 PB31；如果不需要驱动控制该引脚配置该参数为 spk_gpio=-1。

```
# insmod audio.ko  spk_gpio=-1
```

2) Audio 支持同时使用 Amic 和 Dmic 功能；默认关闭。如果使用需要内核同时配置上 Amic 和 Dmic。Audio 同时使用 Amic 和 Dmic 时，可以通过设置参数 dmic_amic_sync=1 来确保 amic 和 dmic 的数据时间差固定。

```
# insmod audio.ko  dmic_amic_sync=1
```

3) Codec 支持输入 ALC 功能，用于自动控制输入增益；默认开启。驱动加载时提供参数 alc_mode 设置 ALC 开启和关闭。该参数配置成 0 时，关闭输入 ALC 功能，使用手动模式配置输入增益。
ALC 开启后，使用 IMP_AI_SetAlcGain 接口设置输入相对增益。ALC 关闭后，使用 IMP_AI_SetGain 接口手动设置输入增益(具体 API 查看文档 7.6 节)。

```
# insmod audio.ko  alc_mode=0
```

## 6.3 GPIO

GPIO 是“General Purpose Input/Output”的简称，具有输入输出，功能复用的功能。开发者在开发硬件驱动时往往需要操作 GPIO，以控制外设硬件。

ISVP 使用 Linux 标准的 GPIOLIB 接口。GPIOLIB 提供了统一的 GPIO 申请/释放/设置/获取接口，按照 GPIOLIB 的设定，需要在 Kernel space 进行调用。如果是 User space 需要操作 GPIO，有两种方法可以选择：

A. 通过 sysfs GPIO 接口进行操作

B. 应用程序调用相关的驱动，驱动中实现 GPIO 的设置。

### 6.3.1 头文件及 API

GPIOLIB 的头文件为：include/linux/gpio.h

在驱动程序中加入头文件引用：

```
#include <linux/gpio.h>
```

API 在头文件 include/asm-generic/gpio.h 中定义，例如：

```
int gpio_request(unsigned gpio, const char *label);
void gpio_free(unsigned gpio);
int gpio_direction_input(unsigned gpio);
int gpio_direction_output(unsigned gpio, int value);
int gpio_get_value(unsigned int gpio);
void gpio_set_value(unsigned int gpio, int value);
int gpio_to_irq(unsigned int gpio);
```

等等。详细的文档说明可参考 kernel/Documentation/gpio.txt；参考代码 Linux kernel 中标准驱动的 GPIO 操作均使用标准的 GPIOLIB，比如 Software I2C，Fixed regulator，以及中断等。

## 6.3.2 sysfs GPIO

sysfs GPIO 是 Linux 标准的用户空间操作 GPIO 的接口。用户可用过命令行或者应用程序直接设置 GPIO 的输入/输出，高低电平等属性。一般情况下，GPIO 调试或者简单的 GPIO 应用（比如 IR-Cut 操作），可通过 sysfs GPIO 接口进行快速开发。

### 6.3.2.1 内核选项

在内核源码根目录下执行$ make menuconfig 命令进入配置界面，选中以下选项：

```
Device Drivers  --->
      -*- GPIO Support  --->
          [*]   /sys/class/gpio/... (sysfs interface)
```

一般情况下，内核的默认配置已经勾选了此选项。

### 6.3.2.2 sysfs GPIO 的申请与释放

在操作 sysfs GPIO 之前需要对其进行申请。值得注意的是，由于申请 sysfs GPIO 会在内核 request_gpio，因此在内核中已经申请过的 GPIO 在 sysfs GPIO 再次申请会失败。

申请/释放 GPIO 方法如下：

```
$ cd /sys/class/gpio
$ echo [gpio_num] > export         #申请 GPIO
$ echo [gpio_num] > unexport       #释放 GPIO
```

注：gpio_num 即 GPIO 号。计算公式为：

PA(n) = 0 * 32 + n

PB(n) = 1 * 32 + n

PC(n) = 2 * 32 + n

...

例如：申请 PB(10) = 1 * 32 + 10 = 42

```
$ echo 42 > export
```

申请后在/sys/class/gpio 目录下即会出现 gpio42 目录。

```
$ echo 42 > unexport
```

释放后 gpio42 目录也会消失。释放后的 GPIO 状态并不会恢复，会保持申请时的状态（电平等）。

#### 6.3.2.3 设置输入/输出方式

在申请 GPIO 后，进入 gpioN 目录，例如 gpio42，进行如下操作：

```
$ echo out > direction            #设置 PB10 为输出模式
$ echo in > direction             #设置 PB10 为输入模式
```

#### 6.3.2.4 设置有效电平

gpioN 目录下有 active_low 节点，表示当前 GPIO 的有限电平，默认为 0，其意义为，当输入/输出 value 为 0 时，GPIO 为低电平，当输入/输出 value 为 1 时，GPIO 为高电平。同样的，当 active_low 为 1 时，当输入/输出 value 为 0 时，GPIO 为高电平，当输入/输出 value 为 1 时，GPIO 为低电平。

也就是说，GPIO 的真实电平=value^active_low。

```
$ echo 0 > active_low    #value 是 0,表示低电平。value 是 1,表示高电平
$ echo 1 > active_low    #value 是 1,表示低电平。value 是 0,表示高电平
```

#### 6.3.2.5 输入/输出

gpioN 目录下有 value 节点，表示 gpioN 的电平：当 GPIO 为输入模式时，读取到 value 的值异或 active_low 即为 GPIO 的电平；当 GPIO 为输出模式时，写入到 value 的值异或 active_low 即为 GPIO 的输出电平。

```
$ cat value               #读取电平（输入模式）
$ echo 0 > value          #设置电平（输出模式）
```

## 6.4 Watchdog

1) 内核配置

在内核源码根目录下执行 make menuconfig 命令进入配置界面，

按下面方式配置

```
Symbol: JZ_WDT [=y]
Type  : tristate
Prompt: Ingenic jz SoC hardware watchdog
    Location:
      -> Device Drivers
        -> Watchdog Timer Support (WATCHDOG [=y])
```

2) 应用例程参考

如果客户使用 Watchdog，请参考 /samples/libsysutils-samples/sample-wdt.c

客户常遇到的问题，没有调用 wdt_disable()；就直接 close()；这样会导致关不掉从而出现错误！

## 6.5 Exfat

1) 内核选项

如果客户需要支持 exfat 文件系统，把/opensource/drivers/misc/exfat-nofuse 下面

的驱动编译加载即可，内核使用默认的就可以了，不需要额外的内核配置。

不提供设备上格式化 exfat 的工具。

## 6.6 ADC

1) 内核配置

在内核源码根目录下执行 make menuconfig 命令进入配置界面，

按下面方式配置

```
Device Drivers  --->
Multifunction device drivers  --->
<*> Support for the XBurst SADC core
<*> Support for the XBurst SADC AUX
```

2) 应用例程参考

参考代码在 sample_adc.c

T31 ADC 的基准电压为 1.8V。

注意：参考代码中 ADC_PATH 表示选择哪个 ADC；STD_VAL_VOLATAGE 表示参考电压，单位为 mv！

## 6.7 Motor

### 6.7.1 驱动介绍

电机的 sample 驱动是使用 GPIO 和定时器来控制四相八拍步进电机的转动。驱动支持两个电机同时使用。参考代码/opensource/drivers/misc/下面；sample_motor 目录为不限位开关的电机驱动； sample_motor2 为带有限位开关的电机驱动；sample_motor3 为带细分器的电机驱动。

电机的控制单位是"步"。在实际产品中，两个电机受到结构的限制有最大转动角度。驱动中定义了两个坐标点(0,0),(vmaxstep,hmaxstep)分别对应两个电机转动的结构限制点。vmaxstep 和 hmaxstep 两个参数可以在加载电机驱动时以参数的形式设置，跟产品的结构转动限制和齿轮转速比有关。

### 6.7.2 接口介绍

表 5-2 Motor API

| 函数 | 功能 |
| --- | --- |
| MOTOR_RESET | RESET 是使用电机前必须设置的。RESET 操作会控制电机先运动到(0，0)坐标点，然后运动到(vmaxstep/2，hmaxstep/2)坐标中心点。上电前，每个电机所处结构的位置可能都不一样，RESET 操作就是把两个电机都初始化到(vmaxstep/2，hmaxstep/2)坐标中心点。 |
| MOTOR_MOVE | MOVE 操作是控制两个电机转动一定的步数，参数 x， |

| | |
|---|---|
| | y 对应两个方向，是相对坐标。 |
| MOTOR_GET_STATUS | GET_STATUS 接口可以得到目前电机的状态和坐标，这个坐标是绝对坐标。 |
| MOTOR_SPEED | 电机的转速范围是 100-900。值越大，速度越快。一般设置 400/500。 |
| MOTOR_GOBACK | 控制电机转动到(vmaxstep/2，hmaxstep/2)坐标中心点。 |
| MOTOR_STOP | STOP 接口控制电机停止。 |
| MOTOR_CRUIS | CRUISE 接口控制两个电机以最大行程巡航。 |

### 6.7.3 驱动适配

根据硬件设计，驱动需要修改 GPIO 配置。在文件 motor.h 中。例如：

```
#define HORIZONTAL_ST1_GPIO     GPIO_PB(22) /**< Phase A */
#define HORIZONTAL_ST2_GPIO     GPIO_PB(21) /**< Phase B */
#define HORIZONTAL_ST3_GPIO     GPIO_PB(20) /**< Phase C */
#define HORIZONTAL_ST4_GPIO     GPIO_PB(19) /**< Phase D */

#define VERTICAL_ST1_GPIO       GPIO_PC(7)
#define VERTICAL_ST2_GPIO       GPIO_PC(6)
#define VERTICAL_ST3_GPIO       GPIO_PC(5)
#define VERTICAL_ST4_GPIO       GPIO_PC(4)
```

注意：产品使用的 gpio 不一样，这点需要认真对比。驱动使用 TCU 定时器，如果客户使用 PWM 驱动，请确认 PWM 驱动使用的 TCU 通道，以防和电机使用 TCU 产生冲突！

## 6.8 PWM

1) 内核配置

在内核源码根目录下执行 make menuconfig 命令进入配置界面，不使用内核自带的驱动程序，只需配置如下选项即可。

```
Device Drivers  --->
[*] Pulse-Width Modulation (PWM) Support  --->
[]   JZ PWM driver (NEW)
*** JZ PWM function pin select ***
```

2) 驱动参考

参考代码 /opensource/drivers/misc/sample_pwm 中。具体使用方法查看驱动目录下的 README 本章节适用的驱动版本为“H20180309a”

注意点：使用相关 PWM 时会使用到 TCU 模块；如果产品同时使用电机，请确认是否和电机驱动中的 TCU 是否冲突！

设置的 polarity 表示有效电平的极性，duty 表示有效电平的占空。例如：polarity = 0，duty=0：表示低电平为有效电平，同时有效电平的占空为 0，此时应该输出的是高电

平。

客户参考代码为 test_pwm.c；其中设置周期的单位为 纳秒！

# 6.9 SPI

## 6.9.1 内核配置

在内核源码根目录下执行 make menuconfig 命令后进入配置界面，

按下面方式配置：

```
Device Drivers  --->
[*] SPI support  --->
<*>   Ingenic JZ series SPI driver
[*]    Ingenic SoC SSI controller 0 for SPI Host driver
-*-      JZ SSI0 controller function pins select
```

## 6.9.2 添加一个 SPI Flash

uboot 和 kernel 需要同步添加，

1) 对于 uboot

打开 drivers/spi/jz_spi.h ；找到 jz_spi_support_table 数组，把新加的 flash 按格式放进去，可以 copy 一份任意现有的 flash 配置，主要修改 name、id_manufactory、sector_size、size，sector_size 是 flash 的擦大小，建议设成 64K，提高擦除速度，如果不支持 64K 请设成 32K。

以添加一个 8M flash EN25QH64 为例：

```
    {
        .name           = “EN25QH64”,
        .id_manufactory = 0x1c7017,
        .page_size       = 256,
        .sector_size      = (32 * 1024),
        .addr_size = 3,
        .size           = (8 * 1024 * 1024),
        .quad_mode = {
            .dummy_byte = 8,
            .RDSR_CMD = CMD_RDSR_1,
            .WRSR_CMD = CMD_WRSR_1,
            .RDSR_DATE = 0x2,//the data is write the spi status register for
QE bit
    .RD_DATE_SIZE = 1,.WRSR_DATE = 0x2,//this bit should be the flash QUAD
mode enable
            .WD_DATE_SIZE = 1,
            .cmd_read = CMD_QUAD_READ,
    #ifdef CONFIG_JZ_SFC
```

```
        .sfc_mode = TRAN_SPI_QUAD,
#endif
    },
},
```

如果要使用四线模式，请根据 spec 定义设置 quad_mode 参数。

还需要查看 drivers/mtd/spi/spi_flash.c 中关于 flash 厂商号是否存在，如果不存在需要自己添加。

2) 对于 kernel

```
$ vi arch/mips/xburst/soc-t31/chip-t31/isvp/common/spi_bus.c
```

找到 spi_nor_pdata 定义添加方法与 uboot 类似，只是参数名称不太一样，其中 id 对应 uboot 中 id_manufactory，erasesize 对应 uboot 中 sector_size，chipsize 对应 uboot 中 size，pagesieze 和 sectorsize 可以不改。

```
  {
    .name          = “EN25QH64”,
    .pagesize      = 256,
    .sectorsize    = 4 * 1024,
    .chipsize      = 8192 * 1024,
    .erasesize     = 32 * 1024,//4 * 1024,
    .id            = 0x1c7017,
    .block_info    = flash_block_info,
    .num_block_info = ARRAY_SIZE(flash_block_info),
    .addrsize      = 3,
    .pp_maxbusy    = 3,           /* 3ms */
    .se_maxbusy    = 400,         /* 400ms */
    .ce_maxbusy    = 8 * 10000,    /* 80s */
    .st_regnum     = 3,
#ifdef CONFIG_SPI_QUAD
    .quad_mode = &flash_quad_mode[0],
#endif
},
```

## 6.10 I2C

1) 内核配置

在内核源码根目录下执行 make menuconfig 命令进入配置界面，选中以下选项：

```
Device Drivers  --->
<*> I2C support  --->
[*]   Autoselect pertinent helper modules
I2C Hardware Bus support  --->
<*> JZ_v12 i2c controler 0 Interface support
JZ_v12 i2c controler 0 function
pins select (GPIO_PA12 & GPIO_PA13)  --->
```

根据需求选择硬件 i2c 或者软件 i2c；硬件 i2c 是确定的引脚，软件 i2c gpio 可以在/kernel/arch/mips/xburst/soc-t31/chip-t31/isvp/Swan/board.h 中修改。

2) 驱动编译

在/opensource/drivers/i2c/sample_eeprom 目录下有使用内核 i2c 驱动的 at24.c 例子，在编译直接需要确认 Makefile 中的 kernel 路径正确性。可以实现 i2c 读写内存。

## 6.11 SLCD

1) 内核配置

```
Device Drivers  --->
<*> Multimedia support  --->
Graphics support  --->
<*> Support for frame buffer devices  --->
```

2) 驱动编译

驱动代码位于/opensource/drivers/sample_slcd；在编译前需要确认 Makefile 中的 kernel 路径正确性和 lcd_device 目录中的初始化 GPIO 需要和硬件对应。

A. /lcd-device_truly240320.c

```
#define GPIO_LCD_CS             GPIO_PB(20)
#define GPIO_LCD_RD             GPIO_PB(19)      //背光
#define GPIO_BL_PWR_EN          GPIO_PC(20)      //背光使能
#define GPIO_LCD_RST            GPIO_PB(22)
```

B. /lcd-driver_truly240320.c

```
jzgpio_set_func(GPIO_PORT_B, GPIO_FUNC_3, 0x3f<<6 | 0x3<<13 | 0x3<<15 | 0x7<<20);          //设置 slcd 控制器的功能引脚
```

注意：T21 平台上 SLCD 使用的时钟是 AHB1，并且默认是没打开的；T31 上使用的时钟是 AHB0，默认是已经打开了。

## 6.12 Wifi

目前 wifi 模组主要存在两种接口方式：USB 和 SDIO； T 系列芯片两种接口都支持。现在主流 wifi 芯片调试完成的型号如下：

A. USB 接口：MT7601，RTL8188，RTL8723BU.

B. SDIO 接口：AP6212，AP6212A，AP6181，RTL8189FS，RTL8189ES

### 6.12.1 Wifi 内核配置

不同 wifi 芯片内核的配置存在不同的地方；下面将逐步介绍以上 wifi 的内核配置部分。

#### 6.12.1.1 MT7601

```
Device Drivers--->
[*] Network device support --->
```

```
[*] Wireless LAN --->
<*> MT7601_util STA support
```

#### 6.12.1.2 RTL8188/RTL8723BU

使用内核默认配置即可支持。

#### 6.12.1.3 AP6212/AP6212A

这两款 wifi 内核默认不支持，需要打上相应的 patch，打上 patch 后按如下配置内核。

**第一步**：先选择 BCM 驱动；

```
Device Drivers--->
Misc devices --->
<*> BCM module power control core driver
```

**第二步**：选择 AP6212，AP6212A 驱动

```
Device Drivers--->
[*] Network device support --->
[*] Wireless LAN --->
<*> Broadcom bcm43438 wireless cards support
Interrupt type (In-Band Interrupt)
```

**第三步**：选择 MMC1 的驱动

```
Device Drivers  --->
<*> MMC/SD/SDIO card support  --->
[*]    JZMMC_V12 MMC1
```

#### 6.12.1.4 AP6181

**第一步**：先选择 BCM 驱动；

```
Device Drivers--->
Misc devices --->
<*> BCM module power control core driver
```

**第二步**：选择 AP6181 驱动

```
Device Drivers--->
[*] Network device support --->
[*] Wireless LAN --->
<*>Broadcom 43341 wireless cards support
```

**第三步**：选择 MMC1 的驱动

```
Device Drivers  --->
<*> MMC/SD/SDIO card support  --->
[*]    JZMMC_V12 MMC1
```

#### 6.12.1.5 RTL8189FS/RTL8189ES

这两款 wifi 只需在内核中把 MMC1 驱动选上即可。

```
Device Drivers  --->
<*> MMC/SD/SDIO card support  --->
```

```
[*]     JZMMC_V12 MMC1
```

### 6.12.2 Wifi 驱动编译

1) USB 接口的 wifi 驱动，只需要指定编译工具链和 kernel 路径，然后直接编译即可。

以 RTL8723BU 为例：打开 Makefile，参照添加 `CONFIG_PLATFORM_INGENIC = y`；然后参照现有板机继续添加：

```
1300 ifeq ($(CONFIG_PLATFORM_INGENIC), y)
1301 EXTRA_CFLAGS += -DCONFIG_LITTLE_ENDIAN -DCONFIG_MINIMAL_MEMORY_USAGE
1302 ARCH ?= mips
1303 CROSS_COMPILE ?= mips-linux-gnu-
1304 KSRC ?= /home_d/ywhan/work/isvp/opensource/kernel
1305
1306 ifeq ($(CONFIG_SDIO_HCI), y)
1307 EXTRA_CFLAGS += -DCONFIG_PLATFORM_OPS
1308 _PLATFORM_FILES += platform/platform_ingenic_sdio.o
1309 endif
1310 endif
```

2) SDIO 接口的 wifi 驱动，首先需要在 platform 文件夹下面添加 power_en 引脚控制代码 platform_ingenic_sdio.c，该文件主要定义了 WIFI 模组的 power_en 引脚的控制，代码默认引脚为 PB(30)。以 RTL8189ES 为例：

A. 首先添加 platform_ingenic_sdio.c，确定硬件对应的 power_en 引脚和软件是否匹配。

```
17 #define GPIO_WIFI_WAKEUP        GPIO_PC(17)
18 #define GPIO_WIFI_RST_N         GPIO_PC(16)
19 #define SDIO_WIFI_POWER         GPIO_PB(30)
20 #define WLAN_SDIO_INDEX         1
21
```

B. 然后打开 Makefile，参照添加 `CONFIG_PLATFORM_INGENIC = y` 然后参照现有板机继续添加

```
1315 ifeq ($(CONFIG_PLATFORM_INGENIC), y)
1316 EXTRA_CFLAGS += -DCONFIG_LITTLE_ENDIAN -DCONFIG_MINIMAL_MEMORY_USAGE
1317 #EXTRA_CFLAGS += -DCONFIG_IOCTL_CFG80211 -DRTW_USE_CFG80211_STA_EVENT
1318 ARCH ?= mips
1319 CROSS_COMPILE ?= mips-linux-gnu-
1320 KSRC ?= /home_d/ywhan/work/isvp/opensource/kernel
1321
1322 ifeq ($(CONFIG_SDIO_HCI), y)
1323 EXTRA_CFLAGS += -DCONFIG_PLATFORM_OPS
1324 _PLATFORM_FILES += platform/platform_ingenic_sdio.o
1325 endif
1326 endif
```

注：AP6212，AP6212A，AP6181 驱动已经集成到内核不需要单独编译。

### 6.12.3 Wifi 启动操作流程

加载 KO， 成功会生成 wlan0 节点，通过 ifconfig -a 可以看到。

**第一步：** `$ ifconfig wlan0 up` 启动 WIFI（可选操作）

**第二步：** `$ wpa_supplicant -D wext -i wlan0 -c wpa_supplicant.conf –B`

（wpa_supplicant 应用软件的参考附录 9.2）

**第三步：** 连接 WIFI：自动配置 IP；`$ udhcpc -i wlan0` 获取 IP 地址，网关等信息（注意 udhcpc.script 的路径匹配）。

或者手动配置 IP；

```
$ ifconfig wlan0 193.169.4.75
$ route add default gw 193.169.4.1
$ echo "nameserver 193.169.1.57" > /etc/resolv.conf
```

### 6.12.4 提高 SDIO wifi 工作时钟

针对需要提高 SDIO wifi 工作时钟的操作;一般使用默认的 24Mclk 即可满足工作，假如需要提高 wifi 的工作时钟，以 T31 芯片为例参考步骤如下：

**第一步：** 进入内核文件 ,打开 arch/mips/xburst/soc-t31 /chip-t31/isvp/common/mmc.c ；修改 sdio_pdata 结构体中 capacity 变量，在原值基础上添加 MMC_CAP_MMC_HIGHSPEED。

**第二步：** 通过 make menuconfig 进入 MMC1 的配置，可以将频率提高到 36M 或者 48M。

### 6.12.5 针对 BCM wifi 的额外说明

BCM WIFI 模块驱动在内核中实现，内核针对 BCM 模块封装了四个板级电源管理接口函数，

```
bcm_wlan_power_on(),
bcm_wlan_power_off(),
bcm_manual_detect(), bcm_customer_wlan_get_oob_irq();
```

用于实现对 WIFI 芯片的上电、下电，sdio 的侦测和复位操作，对应操作中使用的 GPIO 也做了宏定义，客户如有变动，只需要需改相应的宏定义即可。T31 芯片 GPIO 定义在 arch/mips/xburst/soc_t31/chip_t31/isvp/Swan/board.h

```
/* ***************GPIO WIFI START*************** */
#define WL_WAKE_HOST    GPIO_PB(8)
#define WL_REG_EN   GPIO_PC(9)
#define WL_MMC_NUM  1 //sdio use MMC1
#define WLAN_PWR_EN (-1)
#define BCM_PWR_EN  (-1)
#define WL_32K_EN   GPIO_PB(18)
/* ***************GPIO WIFI END***************** */
```

# 6.13 USB

## 6.13.1 USB 工作模式配置

USB 设备模式和 USB 主机模式。在 USB 设备模式下，外部 USB 硬件充当 USB 主机，开发板上的 USB 属于从机设备；USB 主机模式正好相反，开发板属于主机，外部 USB 设备属于从机设备。

### 6.13.1.1 Device Only 模式

```
Device Drivers --->
[*] USB support --->
<*> DesignWare USB2 DRD Core Support
Driver Mode(Device Mode Only) --->
```

### 6.13.1.2 Host Only 模式

```
Device Drivers --->
[*] USB support --->
<*> DesignWare USB2 DRD Core Support
Driver Mode (Host Mode Only) --->
```

### 6.13.1.3 Both 模式

```
Device Drivers --->
[*] USB support --->
<*> DesignWare USB2 DRD Core Support
Driver Mode (Both Host and Device) --->
```

### 6.13.1.4 模式的选择

设置为 both 模式时，host 与 devic 切换有两种方式：

A. 硬件使用 ID PIN 自动切换
B. 软件手动设置

```
//set host mode:
$ devmem 0x10000040 32 0x0b000096
// set device mode:
$ devmem 0x10000040 32 0x0b800096
$ devmem 0x13500000 32 0x001100cc
```

## 6.13.2 USB 接口配置

由于 USB 外设种类繁多，内核默认配置不能完全兼顾，用户可以根据需要自行添加。

### 6.13.2.1 U 盘配置

**第一步：**进入配置界面 make menuconfig 配置如下

```
a)   Device Driver --->
              SCSI device support --->
                  <*> SCSI device support
                  <*> SCSI disk support
b)     Device Driver --->
   [*] USB support  --->
       <*> Support for Host-side USB
       <*> USB Mass Storage support
       <*> DesignWare USB2 DRD Core Support
           Driver Mode (Host Mode Only)
```

**第二步：**编译内核 make uImage 并烧录到开发板上；

**第三步：**插上 U 盘，内核会打印相应提示信息。

#### 6.13.2.2 USB 转串口

**第一步：**进入配置界面 make menuconfig 配置如下

```
a) Device Driver --->
[*] USB support  --->
<*> Support for Host-side USB
<*> USB Modem (CDC ACM) support
<*> USB Serial Converter support  --->
              <*> USB Prolific 2303 Single Port Serial Driver //选择串口芯片，2303 的芯片配置为例。
```

**第二步：**编译内核 make uImage 并烧录到开发板上；

**第三步：**插上串口线，内核会打印相应提示信息。

#### 6.13.2.3 USB 硬盘的配置

**第一步：**进入配置界面 make menuconfig 配置如下

```
a) Device Driver --->
SCSI device support --->
<*> SCSI device support
<*> SCSI disk support
 [*] SCSI low-level drivers (NEW)  --->
b) Device Driver --->
[*] USB support  --->
<*> Support for Host-side USB
<*> USB Mass Storage support
<*> DesignWare USB2 DRD Core Support
Driver Mode (Host Mode Only)
c) [*] Enable the block layer  --->
[*]  Support for large (2TB+) block devices and files
```

**第二步：**编译内核 make uImage 并烧录到开发板上；

**第三步：**连接硬盘，启动系统，内核会打印相应提示信息。

#### 6.13.2.4 USB 以太网卡

**第一步：**进入配置界面 make menuconfig 配置如下

```
a) Eth Config
Device Drivers  --->
[*] Network device support  --->
USB Network Adapters  --->
<*> Multi-purpose USB Networking Framework
<*>   ASIX AX88xxx Based USB 2.0 Ethernet Adapters (NEW)
b) USB Mode Config
   Device Drivers  --->
       [*] USB support  --->
           <*> DesignWare USB2 DRD Core Support
               Driver Mode (Host Mode Only)  --->
```

**第二步：**编译内核 make uImage 并烧录到开发板上；

**第三步：**启动系统，查看相应信息。

#### 6.13.2.5 USB RNDIS 配置

USB RNDIS（远程网络驱动接口规范），产品为 USB 从设备。

**第一步：**进入配置界面 make menuconfig 配置如下

```
a) Device Driver --->
[*] USB support  --->
<*> Support for Host-side USB
<*> USB Gadget Support  --->
<*>   USB Gadget Drivers (Ethernet Gadget
(with CDC Ethernet support))  --->
[*] RNDIS support (NEW)
b)  Device Driver --->
[*] USB support  --->
<*> DesignWare USB2 DRD Core Support
Driver Mode (Device Mode Only)  --->
```

**第二步：**编译内核 make uImage 并烧录到开发板上；

**第三步：**等待进入系统以后

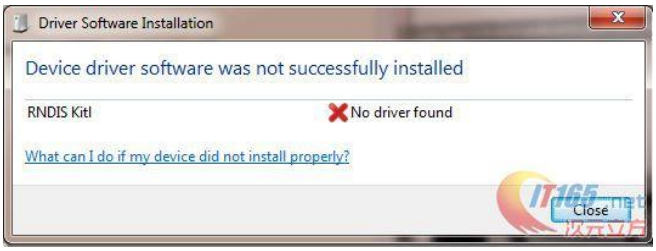
```
$ echo connect >/sys/devices/platform/jz-dwc2/dwc2/udc/dwc2/soft_connect
$ ifconfig usb0 193.169.4.xx up
$ route add default gw 193.169.4.1
```

**第四步：**PC 端配置

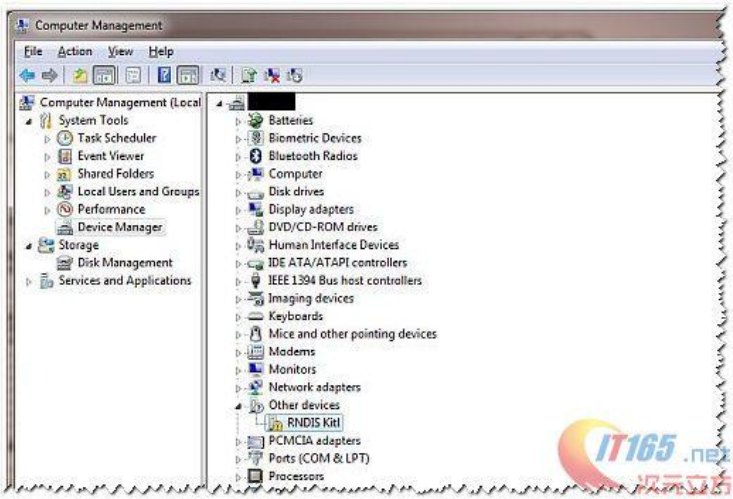
1) RNDIS（Remote Network Driver Interface Specification）也叫远端网络驱动接口协议，设备通过 USB 方式同主机连接，模拟网络连接以便用于下载和调试工作。RNDIS 驱动是 Windows7 的一部分，遗憾的是如果默认安装（插上符合 RNDIS 的设备时）一般均会安装失败（以后可能会修正此问题）。可通过如下方法重新安装 RNDIS 驱动。

2) 设备同计算机连接时，操作系统会自动搜索并安装 RNDIS 驱动，不过，片刻之

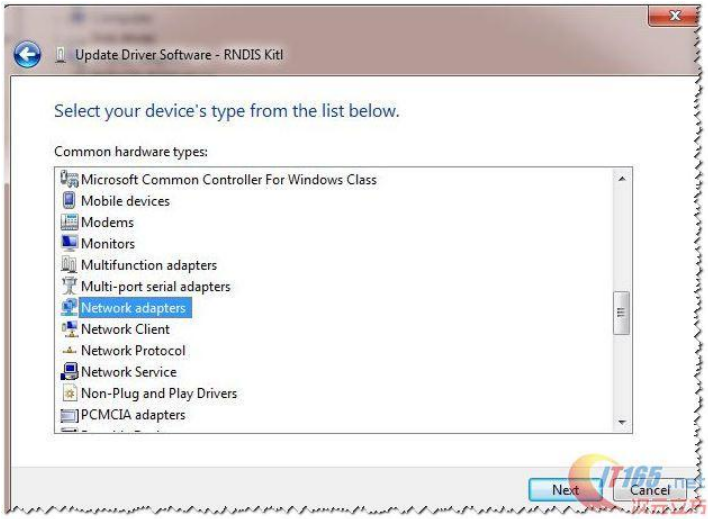
后您会发现安装失败。



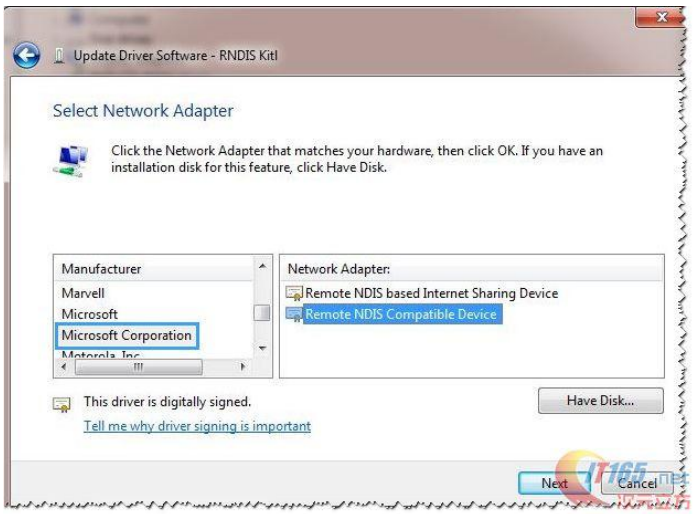

3) 右键点击桌面“计算机”图标，选择“管理”——“设备管理”，可以看到“RNDIS Kitl”设备，并且处于驱动未安装状态。




4) 右键点击该设备，选择更新驱动软件，在如何搜索设备软件提示窗口中，选择“浏览我的计算机”。选择从设备列表中选择“网络适配器”。

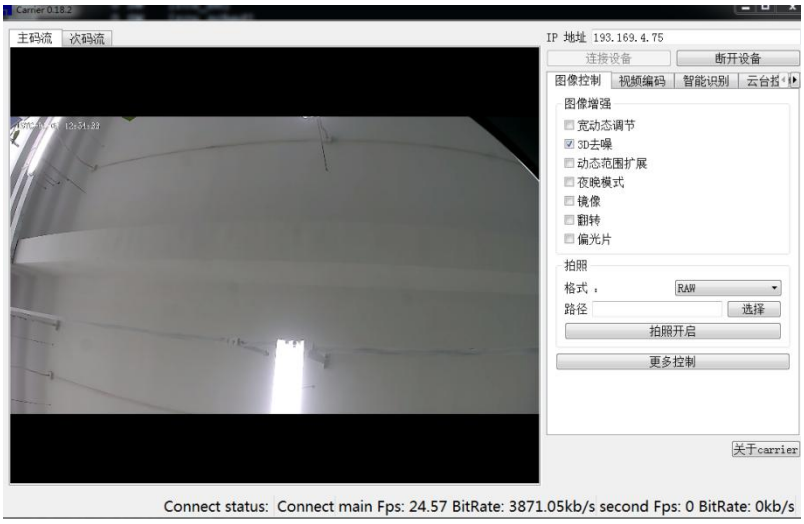



5) 在网络适配器窗口的制造商列表中选择微软公司（Microsoft Corporation），右侧列表中选择远端 NDIS 兼容设备。




6) 点击下一步并等待安装结束，RNDIS Kitl 设备将会安装成功


7) 然后可以执行 carrier tool 工具查看图像。




8) Win10 RNDIS

安装一个驱动 kindle_rndis.inf_amd64-v1.0.0.1.zip ， 在 SDK 中的 /tools/pc 目

录有驱动安装包。

**第一步：**在需要操作的电脑上，解压安装包；

**第二步：**找到解压文件中有个 5-runasadmin_register-CA-cer.cmd ，以管理员身份安装程序；

**第三步：**在我的电脑 管理->设备管理器，在右边目录中有“端口（COM 和 LPT）”，右键单击“串行 USB 设备（COM3）”->更新驱动程序软件...

**第四步：**在控制面板中“网络和共享中心”>更改适配器设置中看到一个新的网卡 Kindle USB RNDIS 设备（已启用 USBNetwork）。分配一个静态 IP（例如 193.169.4.xx），可以 ping 设备 ip。

## 6.14 4G

### 6.14.1 PPP

**第一步：**内核中需要修改的代码

主要修改下面几个文件的代码；参考附录 9.4.1 PPP 模式

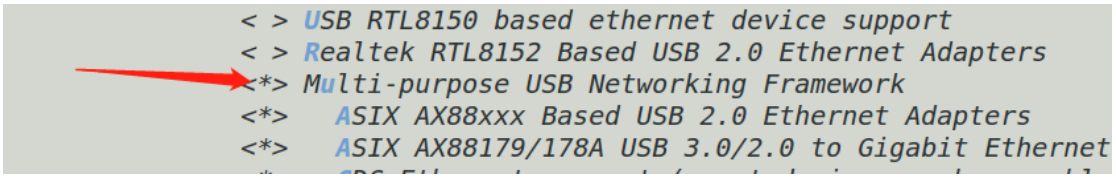
```
modified:   drivers/net/usb/qmi_wwan.c
modified:   drivers/usb/serial/option.c
modified:   drivers/usb/serial/qcserial.c
modified:   drivers/usb/serial/usb_wwan.c
```

**第二步：**内核配置（编译烧录）

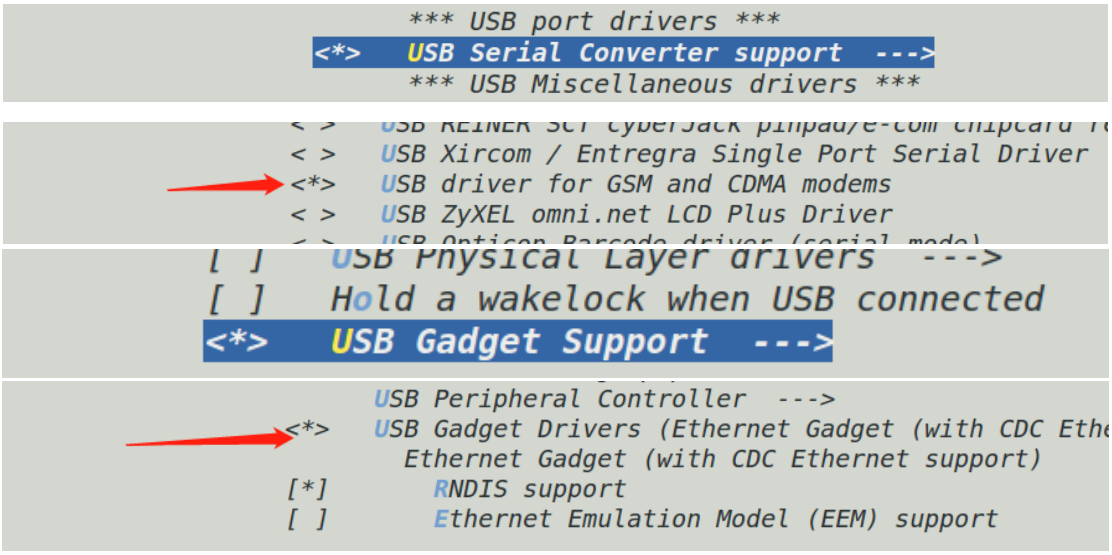
A. network 部分修改

```
<*>   PPP (point-to-point protocol) support
<*>     PPP BSD-Compress compression
<*>     PPP Deflate compression
[*]     PPP filtering
<*>     PPP MPPE compression (encryption)
[*]     PPP multilink support
<*>     PPP over Ethernet
< >     PPP on L2TP Access Concentrator
< >     PPP on PPTP Network Server
<*>     PPP support for async serial ports
<*>     PPP support for sync tty ports
< >   SLIP (serial line) support
```

```
<*>     PPP support for sync tty ports
< >   SLIP (serial line) support
      USB Network Adapters  --->
[*]   Wireless LAN  --->
```




B. usb 部分修改




**第三步：**文件系统的修改

A. 请再文件系统的 /etc 目录下通过创建 ppp 软连接；可参考如下操作

```
cd /etc
ln -s ../system/etc/ppp ppp
```

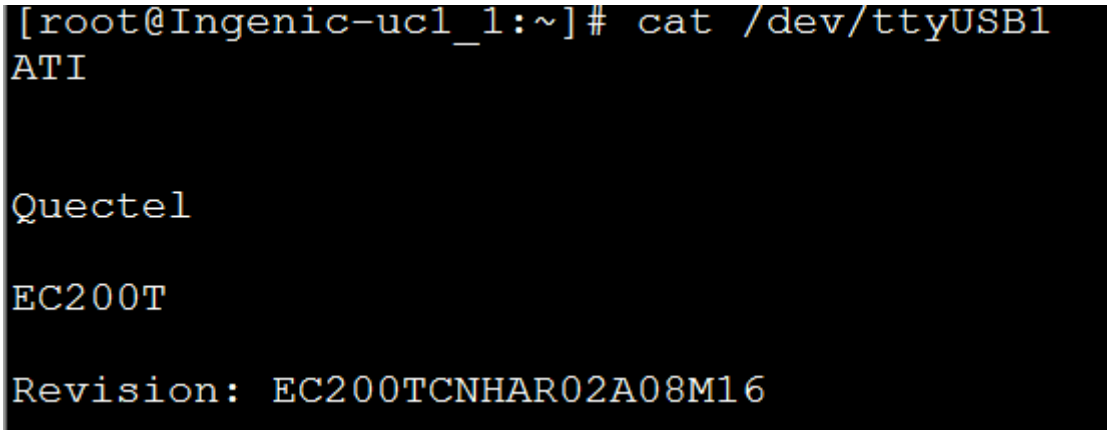
B. 制作 squashfs 文件系统。

linux-ppp-scripts 文件夹里面的 lib 库拷贝到文件系统中， sbin 下的拷贝到系统的 bin 下面；其它脚本参考 readme 使用方法一，拷贝到 /etc/ppp/peers 文件夹下面；ip-up 需要拷贝到 /etc/ppp 目录下！

**第四步：**系统调试

A. 优先查看是否生成了 /dev/ttyUSB*

B. 执行 `echo -e "ATI\r\n" > /dev/ttyUSB1; cat /dev/ttyUSB1`




出现上图所示；表明可以和 4G 模块通信了！

C. 执行命令 pppd call quectel-ppp &

```
Script chat -s -v -f /etc/ppp/peers/quectel-chat-connect finished (pid 299), status = 0x0
Serial connection established.
using channel 1
Using interface ppp0
Connect: ppp0 <--> /dev/ttyUSB2
sent [LCP ConfReq id=0x1 <asyncmap 0x0> <magic 0x97e58dc8> <pcomp> <accomp>]
rcvd [LCP ConfReq id=0x1 <asyncmap 0x0> <auth pap> <magic 0x33bafefe> <pcomp> <accomp>]
sent [LCP ConfAck id=0x1 <asyncmap 0x0> <auth pap> <magic 0x33bafefe> <pcomp> <accomp>]
rcvd [LCP ConfAck id=0x1 <asyncmap 0x0> <magic 0x97e58dc8> <pcomp> <accomp>]
sent [PAP AuthReq id=0x1 user="test" password=<hidden>]
rcvd [PAP AuthAck id=0x1 "" 00]
PAP authentication succeeded
sent [IPCP ConfReq id=0x1 <addr 0.0.0.0> <ms-dns1 0.0.0.0> <ms-dns2 0.0.0.0>]
rcvd [IPCP ConfReq id=0x2]
sent [IPCP ConfNak id=0x2 <addr 0.0.0.0>]
rcvd [IPCP ConfNak id=0x1 <addr 100.185.143.235> <ms-dns1 202.102.213.68> <ms-dns2 61.132.163.68>]
sent [IPCP ConfReq id=0x2 <addr 100.185.143.235> <ms-dns1 202.102.213.68> <ms-dns2 61.132.163.68>]
rcvd [IPCP ConfReq id=0x3]
sent [IPCP ConfAck id=0x3]
rcvd [IPCP ConfAck id=0x2 <addr 100.185.143.235> <ms-dns1 202.102.213.68> <ms-dns2 61.132.163.68>]
Could not determine remote IP address: defaulting to 10.64.64.64
local  IP address 100.185.143.235
remote IP address 10.64.64.64
primary   DNS address 202.102.213.68
secondary DNS address 61.132.163.68

[root@Ingenic-uc1_1:~]# ping www.baidu.com
ping: bad address 'www.baidu.com'
[root@Ingenic-uc1_1:~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         0.0.0.0         0.0.0.0         U     0      0        0 ppp0
10.64.64.64     0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
[root@Ingenic-uc1_1:~]#
```

出现上图打印说明配网成功！

D. 检测 /etc/resolv.conf 是否有配置 DNS，如果有就可以进行下一步

```
[root@Ingenic-uc1_1:~]# cat /etc/resolv.conf
nameserver 61.132.163.68
nameserver 202.102.213.68
```

E. ping www.baidu.com 测试

```
[root@Ingenic-uc1_1:~]# ping www.baidu.com
PING www.baidu.com (14.215.177.38): 56 data bytes
64 bytes from 14.215.177.38: seq=0 ttl=54 time=38.556 ms
64 bytes from 14.215.177.38: seq=1 ttl=54 time=42.371 ms
64 bytes from 14.215.177.38: seq=2 ttl=54 time=37.308 ms
64 bytes from 14.215.177.38: seq=3 ttl=54 time=37.153 ms
```

## 6.14.2 GobNet

**第一步：**在配置 gobnet 时需要先在内核中修改以下代码：

```
A. /drivers/net/usb/Makefile (添加下面两行)
+     obj-y += GobiNet.o
+     GobiNet-objs := GobiUSBNet.o QMIDevice.o QMI.o
B. /drivers/net/usb/qmi_wwan.c  (注释下面一行代码)
-     //{QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
C. /drivers/usb/serial/qcserial.c  (注释下面一行代码)
-     //{USB_DEVICE(0x05c6, 0x9215)},   /* Acer Gobi 2000 Modem device
(VP413) */
```

**第二步：**内核配置（编译烧录）

```
[*] Device Drivers →
-*- Network device support →
```

```
    USB Network Adapters →
        {*} Multi-purpose USB Networking Framework
```

**第三步：**烧录完内核后执行 quectel-CM &即可

### 6.14.3 WWAN

**第一步：**在配置 gobnet 时需要先在内核中修改以下代码：参考附录 9.4.2 Wwan 模式

**第二步：**内核配置（编译烧录）

```
 [*] Device Drivers →
-*- Network device support →
    USB Network Adapters →
        {*} Multi-purpose USB Networking Framework
            <*> QMI WWAN driver for Qualcomm MSM based 3G and LTE modems
```

**第三步：**烧录完内核后执行 quectel-CM &即可

# 7 ISVP-SDK 功能介绍

## 7.1 ISVP-SDK

ISVP-SDK 是由北京君正集成电路股份有限公司开发，集成了系统控制模块、视频源处理模块、图像信号处理单元、视频编码、视频解码、音频模块、OSD 模块、CSC 模块和 IVS 智能分析等模块。具有 OSD 图片叠加（时间戳、Logo 和覆盖等）、CSC 图片格式转换、人脸识别、人形识别、越线检测等功能。组成如图 1 所示。

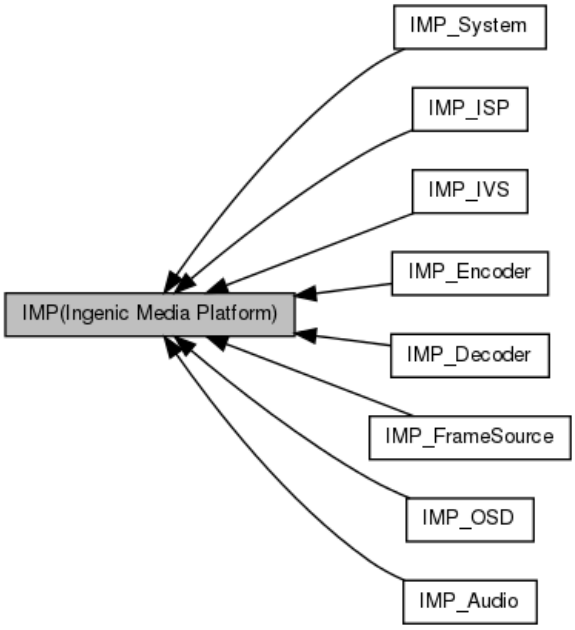



图 6-1 Ingenic Media

### 7.1.1 ISVP-SDK 简介

ISVP SDK，即软件开发工具包，包括 API 库、开源源码、文档、Samples 等等。开发者可以通过 SDK 快速的开展产品功能开发。

#### 7.1.1.1 应用库

1) imp 库

System——实现模块间的数据和系统控制功能

ISP——图像信号处理单元。包括 WDR，3D-Denoise 和 Iridix 等功能

FrameSource——视频源。输出 ISP 处理后的原始视频

Encoder——编码器，实现 H264 和 JPEG 编码

Audio——音频模块，包括录放音，音量调节，音频编码，回音消除等功能

OSD——视频叠加，实现字符点阵叠加的功能

IVS——智能分析，包括 IVS 通道管理，移动侦测功能

2) sysutils 库

su_base——基础功能，包括关机、重启、获取 RTC 时间等

su_cipher——数据加解密模块

su_adc——数模转换数据读取接口

su_misc——其他功能，包括获取按键事件等

#### 7.1.1.2 开源代码

1) u-boot 与 Linux 内核

u-boot 基于 2013.07 版本，支持常用的网络、Nor Flash 操作以及 TF 卡、Fat 文件系统操作

Linux Kernel 版本为 3.10.14，支持常用的总线驱动。开发者可直接编译使用或者进行拓展开发

2) drivers

ISP & Sensor driver. 支持 Sensor 探测识别

EEPROM driver, Timer driver 等系统驱动

## 7.2 IMP_System 系统控制模块

系统控制模块，包括 IMP 的基础功能以及模块间绑定的相关功能

### 7.2.1 相关概念

系统控制主要实现连接各模块，定义数据流的功能。以下是一些重要概念：

#### 7.2.1.1 Device

Device 是完成某一（类）功能的集合。如 FrameSource 完成视频源数据的输出，Encoder 完成视频编码或者图像编码的功能。这里的 FrameSource 和 Encoder 就是 Device 的概念。

Device 只是一个集合的概念，并不是具体的数据流节点。

#### 7.2.1.2 Group

Group 是一路数据输入的最小单位。一个 Device 可以有多个 Group，每个 Group 只能接收一路数据输入，Group 可以有多路输出。

Group 也是具体”功能“的容器，可以详见 Channel 部分的解释。

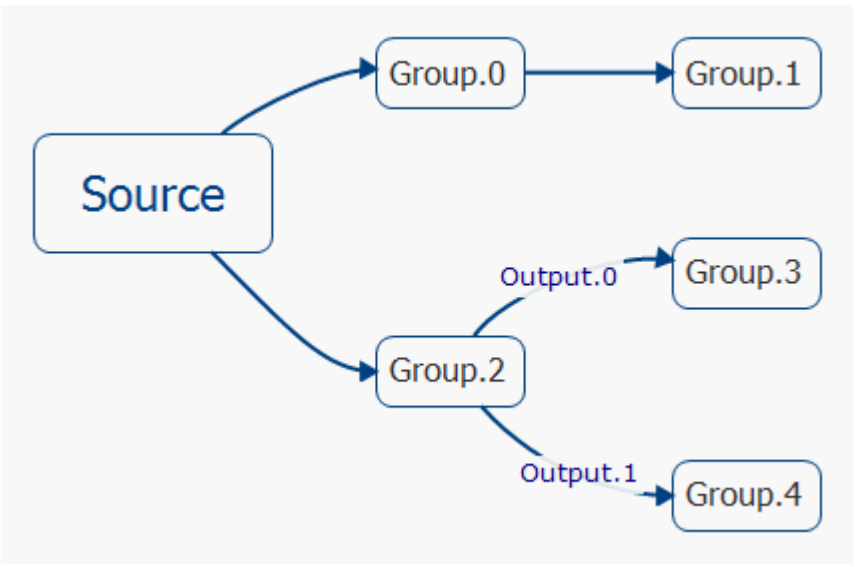



图 6-2 数据流

#### 7.2.1.3 Output

Output 是一个 Group 的一路数据输出的最小单位。一个 Group 可以有多个 Output，每个 Output 只能输出一路数据。

#### 7.2.1.4 Cell

Cell 指包含了 Device、Group、Output 信息的集合。以 IMPCell 的数据结构呈现，Cell 主要用来 Bind（bind）。根据 Device、Group、Output 的定义，Output 作为数据输出的节点，而 Group 作为数据输入的节点。

在 Bind 时数据输出节点的 Cell 索引到输出的 Output，数据输入节点的 Cell 索引到输入的 Group（因此作为数据输入的 Cell，Output 是一个无意义值）。

#### 7.2.1.5 Channel

Channel 通常指一个单一功能的单元，Channel 在 Create 时(实例化)时被指定具体功能。例如：

对于 Encoder，一个 Channel 完成一路 H264 编码或者 JPEG 编码的功能，具体的编码功能(类型，参数)在通道创建时指定；

对于 IVS，一个 Channel 完成一个具体的算法的功能，具体的算法类型参数在通道创建时指定；

对于 OSD，有一个同 Channel 类似的概念 Region，一个 Region 是一个具体的叠加区域，可以是 PIC(图像)，COVER(遮挡)等；

对于 FrameSource，一个 Channel 输出一路原始图像，FrameSource 的 Channel 实际上就是 Group；

Channel 作为功能单元，通常需要 Register 到 Group 中(FrameSource 除外)，才能接收到数据。Channel 注册到 Group 中后，会得到 Group 输入的数据。不同 Device 的 Group 可 Register 的 Channel 数也不同。

### 7.2.2 模块绑定（Bing）

两个 Group 经过 Bind 连接后，源 Group 的数据会自动发到目的 Group。

由于 Group 是数据输入的最小单元，Output 是数据输出的最小单元，因此 IMP_System_Bind(IMPCell *srcCell, IMPCell *dstCell) 的两个参数中 srcCell 的 deviceID，groupID，outputID 是有效的而 dstCell 仅 deviceID 和 groupID 有效，outputID 作为数据输入是无意义的。

下图是一个简单 Bind 的例子。

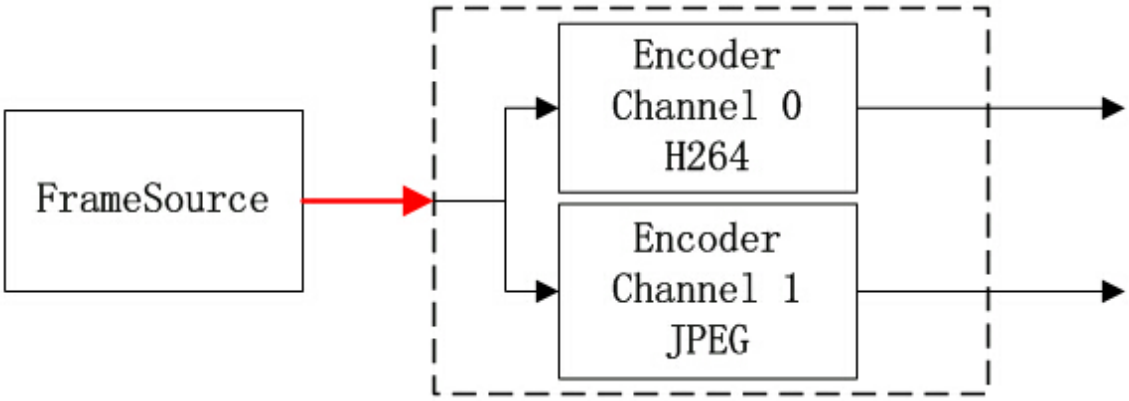



图 6-3 模块绑定

在上图中，实现了 FrameSource 的一路输出 Bind 到 Encoder 的一个 Group。在 Encoder Group 中注册了两个 Channel，因此 Encoder Group 有 H264 和 JPEG 两路输出。

#### 7.2.2.1 典型双路码流

Bind 将系统的数据流串接起来，根据不同的产品功能需求，Bind 的策略也可能不同。以下是典型双路码流产品应用 Bind 的示意图：

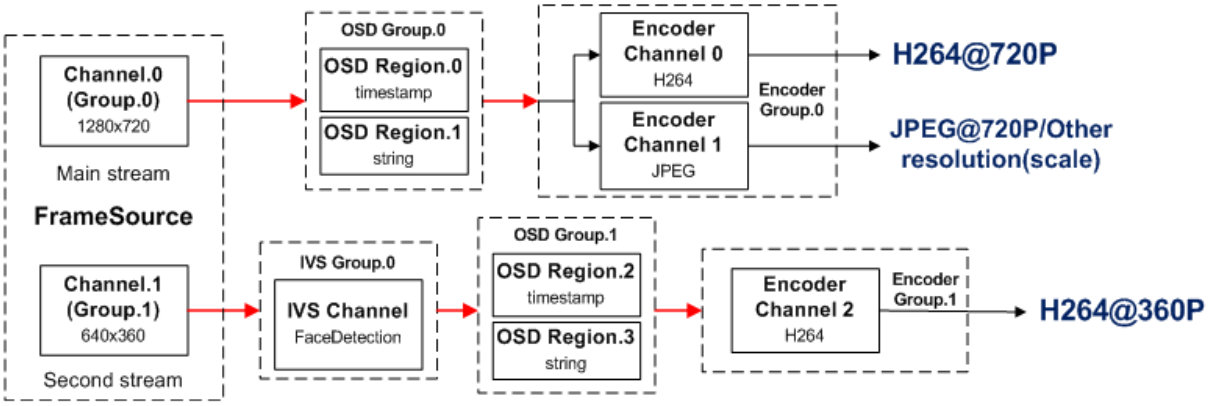



图 6-4 典型双码流 bind 示意

上图中，FrameSource 有两路输出，分别是 Channel 0 输出的主码流(1280x720)和 Channel1 输出的从码流(640x360)。

1) 主码流：

FrameSource 的 `Channel.0` Bind `OSD Group.0`；`OSD Group.0` Bind `Encoder Group.0`。其中：

- OSD Group.0 注册了两个 Region，分别用来显示时间戳和字符串信息
- Encoder Group.0 注册了两个 Channel，分别进行 H264 编码和 JPEG 编码。

其中 JPEG 编码通道的图像大小如果不等于输入设置（FrameSource 的 Channel0），那么就会进行缩放(Software at T10)，达到任意分辨率抓拍的目的。

2) 从码流：

```
    FrameSource 的 Channel.1  Bind  IVS Group.0 ; IVS Group.0  Bind OSD
Group.1 ;
    OSD Group.1  Bind  Encoder Group.1 其中：
```

- IVS Group.0 注册了一个 Channel，用来进行移动侦测
- OSD Group.1 注册了两个 Region，分别用来显示时间戳和字符串信息
- Encoder Group.1 注册了一个 Channel，进行 H264 编码

这里值得注意的一点是，IVS Bind 在 OSD 之前，是因为 OSD 的时间戳可能造成 IVS 移动侦测的误判。

3) 配置 Bind 的时的注意点：

A. 建议所有的 Bind 的操作在系统初始化时进行。

B. 在 FrameSource 使能后 Bind 和 UnBind 操作不能动态调用，需要 Disable FrameSource 后才可进行 UnBind。

C. DestroyGroup 要在 UnBind 之后才能进行。

#### 7.2.2.2 Bind 成树状结构

Bind 可以呈树状结构，下图是一个例子：

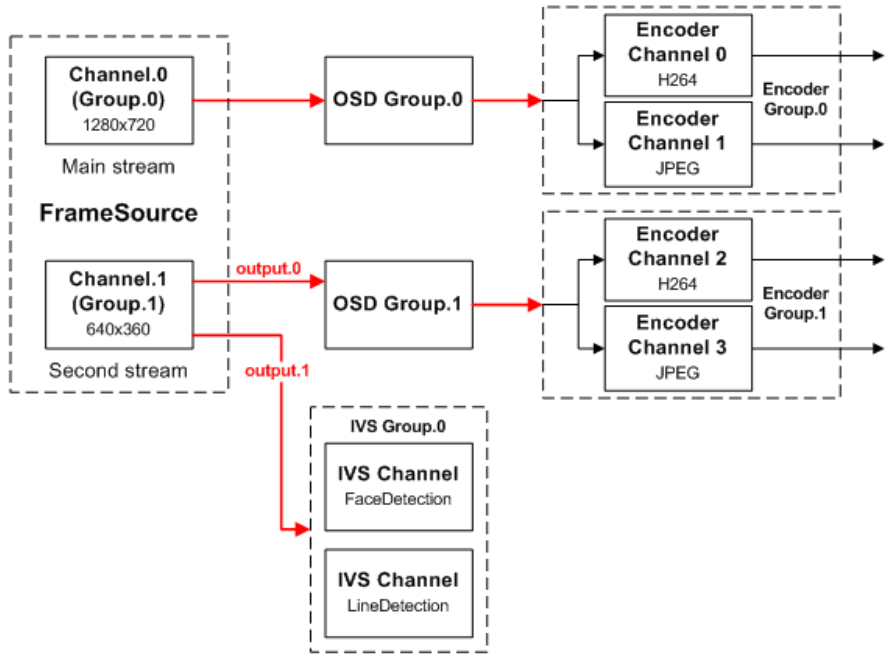



图 6-5 bind 成树状图

上图中，FrameSource 的 Channel 1(Group.1)后端分别 Bind 了两个 Group，分别从 Output.0 和 Output.1 输出数据。本例中这样 Bind 的好处是，IVS Group 可以与 OSD Group.1 并行工作。

上图可以这样进行描述，FrameSource 的 Channel 1(Group.1)后端分别 Bind 了两

个 Group，分别从 Output.0 和 Output.1 输出数据。而 Output.1 的数据是从 Output.0 拷贝而来。这样可以使 IVS Group 可以与 OSD Group.1 并行工作。

注意：

此例中的 Bind 方式可能对普通移动侦测造成影响，因此普通移动侦测不建议采用这种方式。

## 7.2.3 IMP_System 相关 API 介绍

### 7.2.3.1 系统初始化函数

| | |
|---|---|
| 函数原型 | int System_Init(void) |
| 功能介绍 | IMP 系统初始化 |
| 参数介绍 | 无 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 此 API 调用后会初始化基础的数据结构，但不会初始化硬件单元 |
| 注意 | 在 IMP 的任何操作之前必须先调用此接口进行初始化 |

### 7.2.3.2 退出系统函数

| | |
|---|---|
| 函数原型 | int System_Enit(void) |
| 功能介绍 | IMP 系统去初始化 |
| 参数介绍 | 无 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 此函数调用后会释放 IMP 所有的内存以及句柄，并关闭硬件单元 |
| 注意 | 在调用此 API 后，若要再次使用 IMP 则需重新进行 IMP 系统初始化 |

### 7.2.3.3 Bind 绑定

| | |
|---|---|
| 函数原型 | int IMP_System_Bind (IMPCell *srcCell, IMPCell *dstCell) |
| 功能介绍 | 绑定源 Cell 和目的 Cell |
| 参数介绍 | [in] srcCell 源 Cell 指针<br>[in] dstCell 目的 Cell 指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 根据 Device、Group 和 Output 的概念，每个 Device 可能有多个 Group，每个 Group 可能有多个 Output， Group 作为 Device 的输入接口，而 Output 作为 Device 的输出接口.因此绑定实际上是将输出 Device 的某 个 Output 连接到输入 Device 的某个 Group 上.<br>绑定关系成功后，源 Cell(Output)产生的数据会自动传送到目的 Cell(Group) |
| 注意 | 无 |

#### 7.2.3.4 解除 Bind 绑定

| | |
|---|---|
| 函数原型 | int IMP_System_UnBind (IMPCell *srcCell, IMPCell *dstCell) |
| 功能介绍 | 解除源 Cell 和目的 Cell 的绑定 |
| 参数介绍 | [in] srcCell 源 Cell 指针.<br>[in] dstCell 目的 Cell 指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

## 7.3 IMP_FrameSource 视频源模块

视频源，是 IMP 系统的图像数据源，可设置图像的分辨率、裁减、缩放等属性，以及后端降噪功能。

### 7.3.1 模块介绍

视频源，是 IMP 系统的图像数据源，可设置图像的分辨率、裁减、缩放等属性，以及后端降噪功能。

FrameSource 是一个数据流相关概念，可以设置图像分辨率，格式等，并向后端提供原始图像。

FrameSource 的结构如下图：

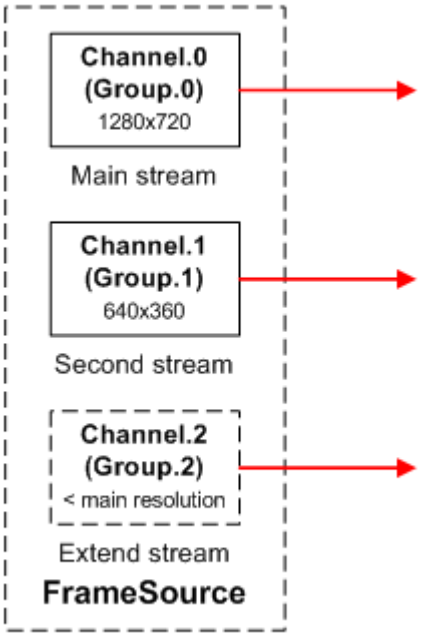



图 6-5 FrameSource

如左图所示，FrameSource 有三种（主码流、次码流和扩展码流）输出，三种输出均可用来编码，其中：

Channel 0 一般作为高清视频流

Channel 1 一般作为标清视频流，或者 IVS 智能算法的数据源

Channel 2 是拓展通道，在特殊应用下使用，一般不建议使用

其实还有一个 Channel 3 是拓展通道，同 Channel 2，所以在这里就没有画出来，在特殊应用下使用，一般情况下不建议使用

注意：在这里扩展通道种的码流其实是从主码流（Channel 0）或次码流（Channel 1）拷贝过来的，作用就是把拷贝过来的视频进行一些其他的处理，来达到一些主码流或次码流不方便做的功能（比如 CSC）或并行运行。但是扩展通道的码流尺寸一般不建议大于或小于所拷贝码流通道的尺寸，这样会额外消耗 CPU，较低 CPU

效率。

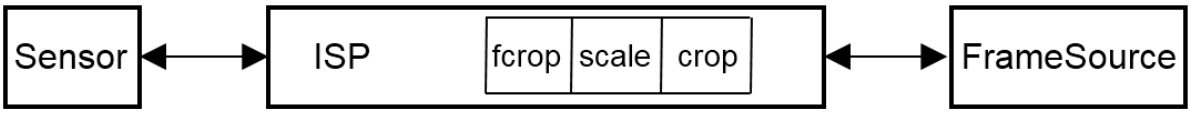



图 6-6 Framesource 获得流程

FrameSource 是 ISP 出图功能的软件抽象。Sensor 输出的一定分辨率的图像经过 fcrop、scaler 和 crop 三级处理后输出。fcrop 是前级裁剪功能，scaler 是缩放功能，crop 是后级裁剪功能。这些功能在 FrameSource 通道参数中都有对应。注意：T30/T21 的区别是没有 fcrop 功能。

### 7.3.2 FrameSource 来源示意图

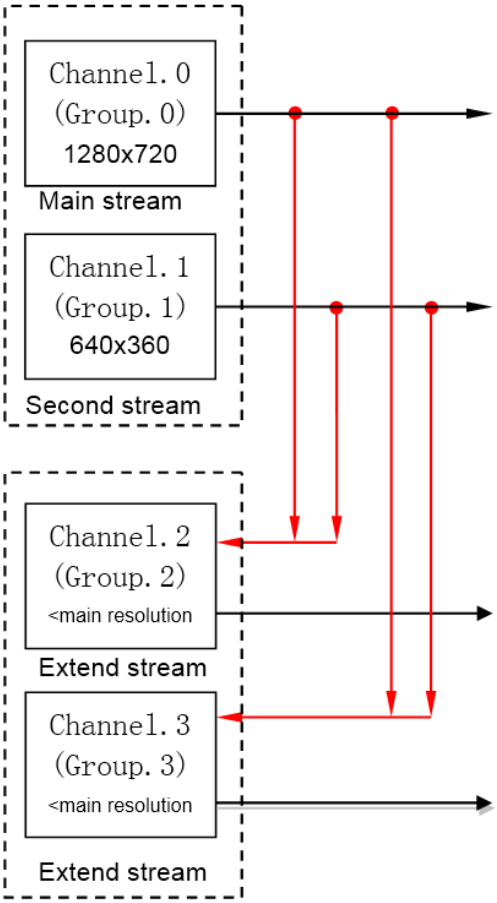



图 6-7 FrameSource 来源示意图

注：红色箭头代表视频源来源（从哪个通道进行拷贝）。黑色箭头代表图像输出。

对于 T31 的 IPU 来说总共有 5 个视屏源输出通道，分别是三个物理通道和两个扩展通道（扩展通道不建议使用）。

下表面为码流表：

表 6-1 码流表

| 码流 | 最大分辨率 | 说明 |
|---|---|---|

| | | |
|---|---|---|
| Main stream | 2048p(2592x2048) | 第一路码流 |
| Second stream | 1080p(1920x1080) | 第二路码流 |
| Third stream | 1080p(1920x1080) | 第三路码流 |

主码流输出的视频源最大分辨率是 2048p(2592x2048)，第二路和第三路码流输出视频源默认的大小是 1080p(1920x1080)，这三个物理通道是被 VIC 直接提供视频源的，而两个扩展通道是需要从三路物理通道拷贝视频源，然后进行相应的操作；可以通过下面的三个函数进行操作：

1) 设置从 chnNum 通道可获取的图像的数目。

```
int IMP_FrameSource_SetFrameDepth(int chnNum, int depth)
```

2) 设置获得的图像数目，这个获得是从刚才设置获取数目里面去拿。

```
int IMP_FrameSource_GetFrameDepth(int chnNum, int *depth)
```

3) 设置获取图像

```
int IMP_FrameSource_GetFrame(int chnNum, IMPFrameInfo **frame)
```

对于使用扩展通道的用户，建议设置输出图像的大小为 2048p(2592x2048)或者 1080p(1920x1080)，这正是主码流通道和次码流通道输出的图像大小，不建议其他大小，因为需要把拷贝的图像进行缩放或者扩大，这样会占用 CPU，降低 CPU 效率。

## 7.3.3 IMP_FrameSource 相关 API 介绍

### 7.3.3.1 设置图像深度

| | |
|---|---|
| 函数原型 | int IMP_FrameSource_SetFrameDepth (int chnNum, int depth) |
| 功能介绍 | 设置可获取的图像最大深度 |
| 参数介绍 | [in] chnNum 通道的编号<br>[in] depth 设置可获取的图像最大深度值 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 1.此接口用于设置某一通道缓存的视频图像帧数。当用户设置缓存多帧视频图像时，用户可以获取到一定数目的连续图像数据。<br>2.若指定 depth 为 0，表示不需要系统为该通道缓存图像，故用户获取不到该通道图像数据。系统默认不为通道缓存图像，即 depth 默认为 0。<br>3.系统将自动更新最旧的图像数据，保证用户一旦开始获取，就可获取到最近最新的图像。<br>4.系统因获取不到图像而自动停止缓存新的图像，用户也不能获取新的图像。因此建议用户保证获取和释放接口配对使用。<br>5.系统将自动更新用户仍未获取的最旧的图像数据，保证缓存的图像队列为最近最新的图像。由于用户不能保证获取速度，导致获取的可能不是连续的图像。<br>6.此函数可以调用位置，没有要求。但是只能使用一次。 |
| 注意 | 无 |

这个函数是指设置能够从通道里获得的图像数目。

#### 7.3.3.2 获取图像最大深度

| 函数原型 | int IMP_FrameSource_GetFrameDepth (int chnNum, int *depth) |
|---|---|
| 功能介绍 | 获取的图像最大深度 |
| 参数介绍 | [in] chnNum 通道的编号<br>[out] depth 获取的图像最大深度值 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

这个函数与上面的函数关系非常深，是指从某个通道获得图像的数目。

#### 7.3.3.3 获取图像

| 函数原型 | int IMP_FrameSource_GetFrame (int chnNum, IMPFrameInfo **frame) |
|---|---|
| 功能介绍 | 获取的图像 |
| 参数介绍 | [in] chnNum 通道的编号<br>[out] frame 获取的图像 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 1.此接口可以获取指定通道的视频图像信息。图像信息主要包括：图像的宽度、高度、像素格式以及图片数据起始地址。<br>2.此接口需在通道已启用后才有效。<br>3.支持多次获取后再释放，但建议获取和释放接口配对使用。<br>4.该接口默认超时时间为 2s，即 2s 内仍未获取到图像，则超时返回 |
| 注意 | 无 |

其实这个函数就是从某个通道获取图像数据。

## 7.4 IMP_ISP 图像信号处理单元

图像信号处理单元。主要包含图像效果设置、模式切换以及 Sensor 的注册添加删除等操作。

### 7.4.1 详细描述

图像信号处理单元。主要包含图像效果设置、模式切换以及 Sensor 的注册添加删除等操作。

ISP 模块与数据流无关，不需要进行 Bind，仅作用于效果参数设置及 Sensor 控制。

#### 7.4.1.1 ISP 模块使能

1) 创建 ISP 模块。
2) 添加一个 sensor，在此操作之前 sensor 驱动已经添加到内核。
3) 使能 sensor，现在 sensor 开始传输图像。

4) 使能 ISP tuning，然后才能调用 ISP 调试接口。
5) 效果调试。

#### 7.4.1.2 ISP 模块卸载步骤

1) 关闭 ISP。
2) 关闭 sensor，现在 sensor 停止传输图像；在此之前 FrameSource 必须全部关闭。
3) 删除 sensor，在此操作之前 sensor 必须关闭。
4) 清理 ISP 模块，在此操作之前所有的 sensor 都必须被删除。
5) ISP 综合扩展功能，色彩模式选择。

### 7.4.2 ISP 支持的模式

#### 7.4.2.1 支持的色彩模式

**表 6-2 ISP 支持的模式**

| | |
|---|---|
| IMPISP_COLORFX_MODE_AUTO | 自动模式 |
| IMPISP_COLORFX_MODE_BW | 黑白模式 |
| IMPISP_COLORFX_MODE_SEPIA | 棕褐色模式 |
| IMPISP_COLORFX_MODE_NEGATIVE | 反向色模式 |
| IMPISP_COLORFX_MODE_VIVID | 鲜艳模式 |

#### 7.4.2.2 ISP 综合扩展功能

ISP 综合扩展功能，场景模式选择。

**表 6-3 场景模式**

| | |
|---|---|
| IMPISP_SCENE_MODE_AUTO | 自动模式 |
| IMPISP_SCENE_MODE_BEACH_SNOW | 海滩雪景模式 |
| IMPISP_SCENE_MODE_CANDLE_LIGHT | 蜡烛灯光模式 |
| IMPISP_SCENE_MODE_DAWN_DUSK | 傍晚模式 |
| IMPISP_SCENE_MODE_FALL_COLORS | 秋天场景模式 |
| IMPISP_SCENE_MODE_FIREWORKS | 焰火场景模式 |
| IMPISP_SCENE_MODE_LANDSCAPE | 风景模式 |
| IMPISP_SCENE_MODE_NIGHT | 夜晚模式 |
| IMPISP_SCENE_MODE_PARTY_INDOOR | 室内晚会模式 |
| IMPISP_SCENE_MODE_SPORTS | 运动模式 |
| IMPISP_SCENE_MODE_SUNSET | 日落模式 |
| IMPISP_SCENE_MODE_TEXT | 文本模式 |
| IMPISP_SCENE_MODE_NIGHT_PORTRAIT | 夜间肖像模式 |

#### 7.4.2.3 白平衡模式

表 6-4 白平衡模式

| | |
|---|---|
| ISP_CORE_WB_MODE_AUTO | 自动模式 |
| ISP_CORE_WB_MODE_MANUAL | 手动模式 |
| ISP_CORE_WB_MODE_DAY_LIGHT | 晴天 |
| ISP_CORE_WB_MODE_CLOUDY | 阴天 |
| ISP_CORE_WB_MODE_INCANDESCENT | 白炽灯 |
| ISP_CORE_WB_MODE_FLOURESCENT | 荧光灯 |
| ISP_CORE_WB_MODE_TWILIGHT | 黄昏 |
| ISP_CORE_WB_MODE_SHADE | 阴影 |
| ISP_CORE_WB_MODE_WARM_FLOURESCENT | 暖色荧光灯 |

### 7.4.3 ISP 相关 API 介绍

#### 7.4.3.1 开启 ISP

| | |
|---|---|
| 函数原型 | int IMP_ISP_Open(void) |
| 功能介绍 | 打开 ISP 模块 |
| 参数介绍 | 无 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 创建 ISP 模块，准备向 ISP 添加 sensor，并开启 ISP 效果调试功能。 |
| 注意 | 这个函数必须在添加 sensor 之前被调用。 |

#### 7.4.3.2 关闭 ISP

| | |
|---|---|
| 函数原型 | int IMP_ISP_Close(void) |
| 功能介绍 | 关闭 ISP 模块 |
| 参数介绍 | 无 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | ISP 模块，ISP 模块不再工作。 |
| 注意 | 在使用这个函数之前，必须保证所有 FrameSource 和效果调试功能已经关闭，所有 sensor 都已被卸载。 |

#### 7.4.3.3 添加一个 sensor

| | |
|---|---|
| 函数原型 | int IMP_ISP_AddSensor(IMPSensorInfo *pinfo) |
| 功能介绍 | 添加一个 sensor，用于向 ISP 模块提供数据源 |

| 参数介绍 | [in] pinfo 需要添加 sensor 的信息指针 |
|---|---|
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 添加一个摄像头，用于提供图像。 |
| 注意 | 在使用这个函数之前，必须保证摄像头驱动已经注册进内核 |

#### 7.4.3.4 删除一个 sensor

| 函数原型 | int IMP_ISP_DelSensor(IMPSensorInfo *pinfo) |
|---|---|
| 功能介绍 | 删除一个 sensor |
| 参数介绍 | [in] pinfo 需要删除 sensor 的信息指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在使用这个函数之前，必须保证摄像头已经停止工作，即调用了 IMP_ISP_DisableSensor 函数 |

#### 7.4.3.5 使能一个 Sensor

| 函数原型 | int IMP_ISP_EnableSensor(void) |
|---|---|
| 功能介绍 | 使能一个 sensor |
| 参数介绍 | 无 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 使能一个摄像头，使之开始传输图像，这样 FrameSource 才能输出图像，同时 ISP 才能进行效果调试。 |
| 注意 | 在使用这个函数之前，必须保证摄像头已经被添加进 ISP 模块 |

#### 7.4.3.6 失能一个 Sensor

| 函数原型 | int IMP_ISP_DisableSensor(void) |
|---|---|
| 功能介绍 | 不使能一个 sensor |
| 参数介绍 | 无 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 不使能一个摄像头，使之停止传输图像，这样 FrameSource 无法输出图像，同时 ISP 也不能进行效果调试。 |
| 注意 | 在使用这个函数之前，必须保证所有 FrameSource 都已停止输出图像，同时效果调试也在不使能态 |

#### 7.4.3.7 设置 ISP 图像上下反转效果

| 函数原型 | int IMP_ISP_Tuning_SetISPVflip(IMPISPTuningOpsMode mode) |
|---|---|
| 功能介绍 | 设置 ISP 图像上下反转效果功能是否使能 |
| 参数介绍 | [in] mode 是否使能图像上下反转 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在使用这个函数之前，IMP_ISP_EnableTuning 已被调用。 |

#### 7.4.3.8 获取 ISP 图像上下反转效果

| 函数原型 | int IMP_ISP_Tuning_GetISPVflip(IMPISPTuningOpsMode *mode) |
|---|---|
| 功能介绍 | 获取 ISP 图像上下反转效果功能的操作状态 |
| 参数介绍 | [in] pmode 操作参数指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在使用这个函数之前，IMP_ISP_EnableTuning 已被调用 |

#### 7.4.3.9 设置 ISP 工作模式

| 函数原型 | int IMP_ISP_Tuning_SetISPRunningMode(IMPISPRunningMode mode) |
|---|---|
| 功能介绍 | 设置 ISP 工作模式，正常模式或夜视模式；默认为正常模式 |
| 参数介绍 | [in] mode 运行模式参数 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在使用这个函数之前，IMP_ISP_EnableTuning 已被调用 |

#### 7.4.3.10 获取 ISP 工作模式

| 函数原型 | int IMP_ISP_Tuning_GetISPRunningMode(IMPISPRunningMode *mode) |
|---|---|
| 功能介绍 | 获取 ISP 工作模式，正常模式或夜视模式 |
| 参数介绍 | [in] pmode 操作参数指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在使用这个函数之前，IMP_ISP_EnableTuning 已被调用 |

#### 7.4.3.11 设置摄像头输出帧率

| | |
|---|---|
| 函数原型 | int IMP_ISP_Tuning_SetSensorFPS(uint32_t fps_num,uint32_t fps_den) |
| 功能介绍 | 设置摄像头输出帧率 |
| 参数介绍 | [in] fps_num 设定帧率的分子参数<br>[in] fps_den 设定帧率的分母参数 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在使用这个函数之前，必须保证 IMP_ISP_EnableSensor 和 IMP_ISP_EnableTuning 已被调用。<br>这个默认输出的帧率是 25fps,没有特殊使用情况，不建议去改变输出帧率 |

#### 7.4.3.12 获取摄像头输出帧率

| | |
|---|---|
| 函数原型 | int IMP_ISP_Tuning_GetSensorFPS(uint32_t *fps_num,uint32_t *fps_den) |
| 功能介绍 | 获取摄像头输出帧率 |
| 参数介绍 | [in] fps_num 获取帧率分子参数的指针<br>[in] fps_den 获取帧率分母参数的指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在使用这个函数之前，必须保证 IMP_ISP_EnableSensor 和 IMP_ISP_EnableTuning 已被调用。 在使能帧通道开始传输数据之前必须先调用此函数获取摄像头默认帧率。 这个在默认情况下获取到的帧率是 25fps，这是默认的设置 |

#### 7.4.3.13 使能 WDR 模式

| | |
|---|---|
| 函数原型 | int IMP_ISP_WDR_ENABLE(IMPISPTuningOpsMode mode) |
| 功能介绍 | 使能 ISP WDR |
| 参数介绍 | [in] mode ISP WDR 模式 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 需要在 IMP_ISP_OPEN 以及 IMP_ISP_AddSensor 之间使能 WDR 模式。<br>//step1：创建一个 ISP 模块<br>IMP_ISP_Open();<br>//step2：使能 WDR |

```
    if (gconf_wdr != 0) {
        IMPISPTuningOpsMode mode;
        mode = IMPISP_TUNING_OPS_MODE_ENABLE;
        IMP_ISP_WDR_ENABLE(mode);
        }
    //step3：添加一个 sensor
    IMP_ISP_AddSensor(&sensor_info);
        }
```

## 7.5 IMP_Encode 视频编码模块

视频编码（H264, H265, JPEG）模块，包含编码通道管理，编码参数设置等功能。

### 7.5.1 模块介绍

Encoder 模块内部结构如下如：

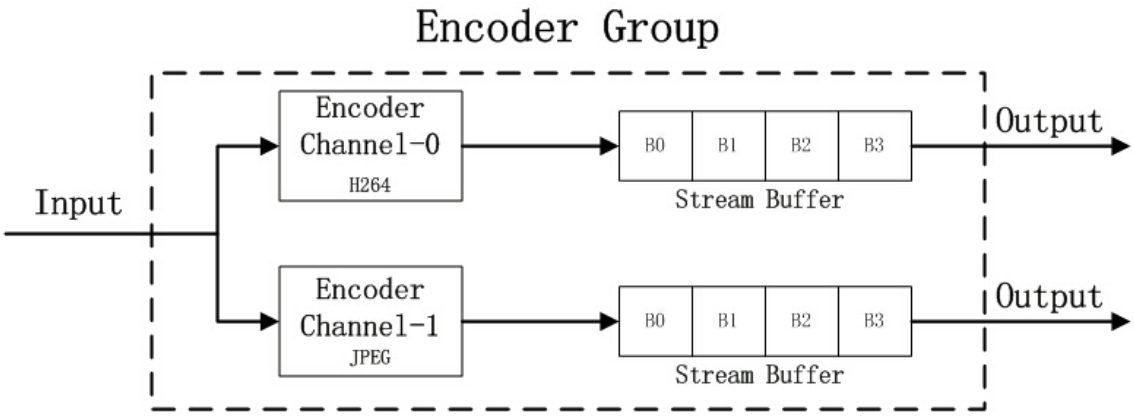



图 6-8 Encode 模块内部结

如上图所示，编码模块由若干个 Group 组成，每个 Group 由编码 Channel 组成。每个编码 Channel 附带一个输出码流缓冲区，这个缓冲区由多个 buffer 组成。

目前 T31 上支持 6 个 Group，每个 Group 由 2 个 Channel 组成。最多使用 6 个 Channel。

### 7.5.2 编码 Channel

一个编码 Channel 可以完成一种协议的编码。目前每个 Group 有两个 Channel，可以添加一个 H264 编码 Channel 与一个 JPEG 编码 Channel。

### 7.5.3 码率控制

#### 7.5.3.1 CBR

CBR（Constent Bit Rate）恒定比特率，即在码率统计时间内编码码率恒定。以 H.264

编码为例，用户可设置 maxQp，minQp，bitrate 等。maxQp，minQp 用于控制图像的质量范围，bitrate 用于钳位码率统计时间内的平均编码码率。

当编码码率大于恒定码率时，图像 QP 会逐步向 maxQp 调整，当编码码率远小于恒定码率时，图像 QP 会逐步向 minQp 调整。

当图像 QP 达到 maxQp 时，QP 被钳位到最大值，bitrate 的钳位效果失效，编码码率有可能会超出 bitrate。

当图像 QP 达到 minQp 时，QP 被钳位到最小值，此时编码的码率已经达到最大值，而且图像质量最好。

#### 7.5.3.2 VBR

VBR 可变比特率。

#### 7.5.3.3 CAPPED_VBR

Use variable bitrate rate controller based on ma PSNR target

#### 7.5.3.4 FixQP

Fix Qp 固定 Qp 值。在码率统计时间内，编码图像所有宏块 Qp 值相同，采用用户设定的图像 Qp 值。

### 7.5.4 IMP_Encoder 相关 API 介绍

#### 7.5.4.1 创建编码 Group

| 函数原型 | int IMP_Encoder_CreateGroup(int encGroup) |
|---|---|
| 功能介绍 | 创建编码 Group |
| 参数介绍 | [in] encGroup Group 号,取值范围:[0, NR_MAX_ENC_GROUPS - 1] |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 一路 Group 仅支持一路分辨率，不同分辨率需启动新的 Group。一路 Group 同时支持一路 H264 和一路 JPEG 抓拍 |
| 注意 | 如果指定的 Group 已经存在，则返回失败 |

#### 7.5.4.2 销毁编码 Group

| 函数原型 | int IMP_Encoder_DestroyGroup(int encGroup) |
|---|---|
| 功能介绍 | 销毁编码 Grouop |
| 参数介绍 | [in] encGroup Group 号,取值范围:[0,NR_MAX_ENC_GROUPS - 1] |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 销毁 Group 时，必须保证 Group 为空，即没有任何 Channel 在 Group 中注册，或注册到 Group 中 |
| 注意 | 销毁并不存在的 Group，则返回失败 |

#### 7.5.4.3 创建编码 Channel

| | |
|---|---|
| 函数原型 | int IMP_Encoder_CreateChn(int encChn, const IMPEncoderCHNAttr *attr) |
| 功能介绍 | 创建编码 Channel |
| 参数介绍 | [in] encChn编码 Channel 号,取值范围:[0, NR_MAX_ENC_CHN - 1]<br>[in] attr 编码 Channel 属性指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 编码 Channel 属性由两部分组成，编码器属性和码率控制属性 。<br>编码器属性首先需要选择编码协议，然后分别对各种协议对应的属性进行赋值。<br>第一次创建的编码 channel 时帧率码率等参数必须设置为最大。 |
| 注意 | 无 |

#### 7.5.4.4 销毁编码 Channel

| | |
|---|---|
| 函数原型 | int IMP_Encoder_DestroyChn(int encChn) |
| 功能介绍 | 销毁编码 Channel |
| 参数介绍 | [in] encChn编码 Channel 号,取值范围:[0,NR_MAX_ENC_CHN - 1] |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 销毁并不存在的 Channel，则返回失败 。<br>销毁前必须保证 Channel 已经从 Group 反注册，否则返回失败 。 |

#### 7.5.4.5 获取编码 Channel 属性

| | |
|---|---|
| 函数原型 | int IMP_Encoder_GetChnAttr(int encChn, IMPEncoderCHNAttr *const attr) |
| 功能介绍 | 获取编码 Channel 的属性 |
| 参数介绍 | [in] encChn编码 Channel 号,取值范围: [0,NR_MAX_ENC_CHN - 1]<br>[in] attr 编码 Channel 属性 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.5.4.6 注册编码 Channel 到 Group

| | |
|---|---|
| 函数原型 | int IMP_Encoder_RegisterChn(int encGroup, int encChn) |
| 功能介绍 | 注册编码 Channel 到 Group |
| 参数介绍 | [in] encGroup 编码 Group 号，取值 |

| | |
|---|---|
| | 围:[0,NR_MAX_ENC_GROUPS-1]<br>[in] encChn 编码 Channel 号,取值范围:[0, NR_MAX_ENC_CHN - 1] |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 注册并不存在的 Channel，则返回失败 。<br>注册 Channel 到不存在的 Group，否则返回失败 。<br>同一个编码 Channel 只能注册到一个 Group，如果该 Channel 已经注册到某个 Group，则返回失败。<br>如果一个 Group 已经被注册，那么这个 Group 就不能再被其他的 Channel 注册，除非之前注册关系被解除。 |

#### 7.5.4.7 开启编码 Channel 接收图像

| | |
|---|---|
| 函数原型 | int IMP_Encoder_StartRecvPic( int encChn) |
| 功能介绍 | 开启编码 Channel 接收图像 |
| 参数介绍 | [in] encChn编码 Channel 号,取值范围: [0,NR_MAX_ENC_CHN - 1] |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 开启编码 Channel 接收图像后才能开始编码 |
| 注意 | 如果 Channel 未创建，则返回失败 。<br>如果 Channel 没有注册到 Group，则返回失败 |

#### 7.5.4.8 停止编码 Channel 接收图像

| | |
|---|---|
| 函数原型 | int IMP_Encoder_StopRecvPic( int encChn) |
| 功能介绍 | 停止编码 Channel 接收图像 |
| 参数介绍 | [in] encChn编码 Channel 号,取值范围: [0,NR_MAX_ENC_CHN - 1] |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 此接口并不判断当前是否停止接收，即允许重复停止接收不返回错误 。<br>调用此接口仅停止接收原始数据编码，码流 buffer 并不会被消除。 |
| 注意 | 如果 Channel 未创建，则返回失败 。<br>如果 Channel 没有注册到 Group，则返回失败 。 |

#### 7.5.4.9 获取编码的码流

| | |
|---|---|
| 函数原型 | int IMP_Encoder_GetStream( int encChn, IMPEncoderStream *stream, bool blockFlag) |
| 功能介绍 | 获取编码的码流 |

| | |
|---|---|
| 参数介绍 | [in] encChn编码 Channel 号,取值范围: [0, NR_MAX_ENC_CHN - 1]<br>[in] stream 码流结构体指针<br>[in] blockFlag 是否使用阻塞方式获取，0：非阻塞，1：阻塞 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 每次获取一帧码流的数据。<br>如果用户长时间不获取码流,码流缓冲区就会满。一个编码 Channel 如果发生码流缓冲区满,就会把后 面接收的图像丢掉,直到用户获取码流,从而有足够的码流缓冲可以用于编码时,才开始继续编码。建议用户 获取码流接口调用与释放码流的接口调用成对出现,且尽快释放码流,防止出现由于用户态获取码流,释放不 及时而导致的码流 buffer 满,停止编码。<br>对于 H264 类型码流，一次调用成功获取一帧的码流，这帧码流可能包含多个包。<br>对于 JPEG 类型码流，一次调用成功获取一帧的码流，这帧码流只包含一个包，这一帧包含了 JPEG 图片文件的完整信息。 |
| 注意 | 无 |

#### 7.5.4.10 释放码流缓存

| | |
|---|---|
| 函数原型 | int IMP_Encoder_ReleaseStream( int encChn, IMPEncoderStream *stream) |
| 功能介绍 | 释放码流缓存 |
| 参数介绍 | [in] encChn编码 Channel 号,取值范围: [0,NR_MAX_ENC_CHN - 1]<br>[in] stream 码流结构体指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 此接口应当和 IMP_Encoder_GetStream 配对起来使用，<br>用户获取码流后必须及时释放已经获取的码流缓存，否则可能会导致码流 buffer 满，影响编码器编码。<br>并且用户必须按先获取先 释放的顺序释放已经获取的码流缓存；<br>在编码 Channel 反注册后，所有未释放的码流包均无效，不能再使用或者释放这部分无效的码流缓存。 |
| 注意 | 如果 pstStream 为 NULL,则返回失败；如果 Channel 未创建，则返回失败；释放无效的码流会返回失败。 |

#### 7.5.4.11 设置码流缓存 Buffer 数

| | |
|---|---|
| 函数原型 | int IMP_Encoder_SetMaxStreamCnt( int encChn, int nrMaxStream) |
| 功能介绍 | 设置码流缓存 Buffer 个数 |
| 参数介绍 | [in] encChn 编码 Channel 号,取值范围: [0,NR_MAX_ENC_CHN - 1]<br>[in] nrMaxStream 码流 Buffer 数,取值范 |

| | |
|---|---|
| | 围:[1,NR_MAX_ENC_CHN_STREAM] |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 由于码流缓存 Buffer 个数在通道创建时就已经固定，因此次 API 需要在通道创建之前调用。<br>若通道创建之前不调用此 API 设置码流缓存 Buffer 个数，则 H264 通道默认个数为 5，JPEG 通道默认个数为 1。 |
| 注意 | 无 |

#### 7.5.4.12 获取码流缓存 Buffer 数

| | |
|---|---|
| 函数原型 | int IMP_Encoder_ReleaseStream( int encChn, int *nrMaxStream) |
| 功能介绍 | 获取码流 Buffer 数 |
| 参数介绍 | [in] encChn 编码 Channel 号,取值范围:[0,NR_MAX_ENC_CHN - 1]<br>[out] nrMaxStream 码流 Buffer 数变量指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.5.4.13 设置编码默认参数

| | |
|---|---|
| 函数原型 | int IMP_Encoder_SetDefaultParam(IMPEncoderChnAttr *chnAttr, IMPEncoderProfile profile, IMPEncoderRcMode rcMode, uint16_t uWidth, uint16_t uHeight, uint32_t frmRateNum, uint32_t frmRateDen, uint32_t uGopLength, int iInitialQP, uint32_t uTargetBitRate) |
| 功能介绍 | 一键设置默认编码通道参数 |
| 参数介绍 | [in] profile, rcMode 编码模式，uWidth,uHeigth 编码分辨率，frmRateNum 帧率分子，frmRateDen 帧率分母，uGopLength GOP 长度，iInitialQP 初始化 QP，uTargetBitRate 目标码率<br>[out] chnAttr 编码通常参数 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

### 7.5.5 编码参数变化说明

T31 编码通道参数与其他平台的参数变化比较多，再原有的编码器属性 encAttr 和码率控制器属性 rcAttr 基础上增加 gop 属性 gopAttr。下面介绍一下这三种属性的说明和与原来平台的参数差异

#### 7.5.5.1 encAttr

encAttr 属性中删除了原来的 enType，bufSize，crop，userData；变化的有 eProfile，其类型有 AVC_MAIN，AVC_HIGH，AVC_BASELINE，HEVC_MAIN，JPEG；增加的有 ePicFormat 编码源数据的格式（400,420,422），eEncOptions 编码附加选项，eEncTools 编码附加工具。

#### 7.5.5.2 rcAttr

rcAttr 属性中删除了原来的 maxGop，attrFrmUsed，attrDenoise，attrHSkip，编码模式 attrRcMode 去掉原来的 smart 模式，增加了 CappedVbr 模式，其中 cbr 模式删除了原来的 iBiasLvl，frmQPStep，gopQPStep，adaptiveMode，gopRelation，增加了 iInitialQP 初始化 QP，iIPDelta I 帧和 P 帧的 qp 差（设为-1 编码器自动设置），iPBDelta P 帧和 B 帧的 qp 差（设为-1 编码器自动设置），eRcOptions 码率控制附加选项，uMaxPictureSize 指定最大帧的尺寸；vbr 删除了原来的 staticTime，iBiasLvl，changePos，qualityLvl，frmQPStep，gopQPStep，gopRelation，增加的参数和 cbr 相同；CappedVbr 在 vbr 的基础上加入 uMaxPSNR 图像最大 psnr 值。

#### 7.5.5.3 gopAttr

Gop 属性参数主要有 uGopCtrlMode gop 模式的设置，有 DEFAULT 和 PYRAMIDAL 两种，DEFAULT 如下示意图：

**DEFAULT_GOP** : Basic GOP settings :

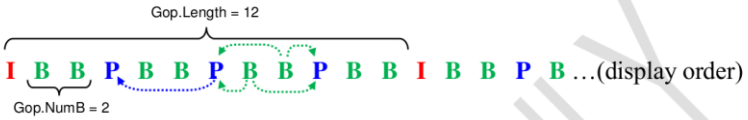



图 6-9 DEFAULT

PYRAMIDAL 如下示意图：

PYRAMIDAL_GOP : Advanced Gop pattern with hierarchical B frame. The size of the hierarchy depends on the Gop.NumB parameter.

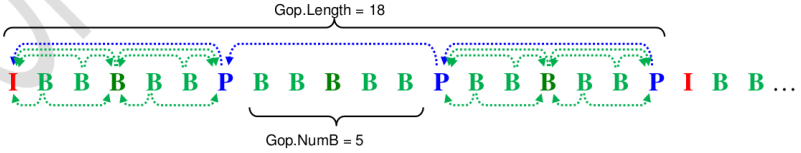



图 6-10 PYRAMIDAL

uGopLength gop 的长度；uNumB gop 中的 B 帧个数；uFreqIDR 指定两个 I 帧之间的帧数（取决于 gop 长度）；bEnableLT 使能长参考帧；uFreqLT 以帧数为单位指定长期参考图片刷新频率，前提使能长参考帧；bLTRC 将长期参考帧作为第二个参考候选（以额外带宽为代价增加静态场景的质量）。

## 7.6 IMP_Audio 音频模块

音频模块，包含录放音、音频编解码、音量及增益设置、回声消除、自动增益等功

能。

## 7.6.1 模块介绍

音频功能包含音频输入，音频输出，回音消除，音频编码和音频解码 5 个模块。

其中音频输入和音频输出存在设备和通道的概念。其中一个 MIC 认为是一个 Device，而一个 MIC 可以有多路 Channel 输入。同样的一个 SPK 认为是一个放音 Device，而一个 SPK 也可以有多路 Channel 输出。当前版本的音频 API 一个 Device 只支持一个 Channel。

回音消除位于音频输入接口中，具体说明在功能描述中体现。

音频编码当前音频 API 中支持 PT_G711A、PT_G711U 和 PT_G726 格式音频编码，如需要增加新的编码方式，需要注册编码器。

音频解码当前音频 API 中支持 PT_G711A、PT_G711U 和 PT_G726 格式音频解码，如需要增加新的解码方式，需要注册解码器。

## 7.6.2 功能描述

以下是对每个模块的具体说明。

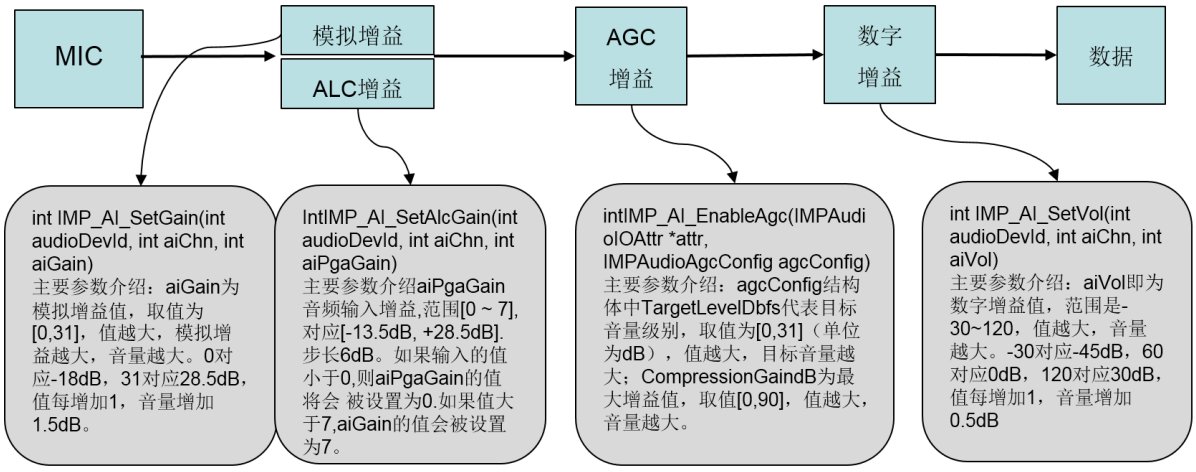
### 7.6.2.1 音频输入

音频输入 Device ID 对应关系， 0：对应数字 MIC 1：对应模拟 MIC

音频输入 Channel 当前 API 只支持 1 个通道。

音频输入的音量设置包含模拟增益 gain，agc 增益，数字增益 vol，ALC 增益。其中模拟增益 gain 和 ALC 增益两者是互斥的关系，通过驱动参数 alc_mode 选择使用模拟增益 gain 或者 ALC 增益。

增益控制流程如下图所示：



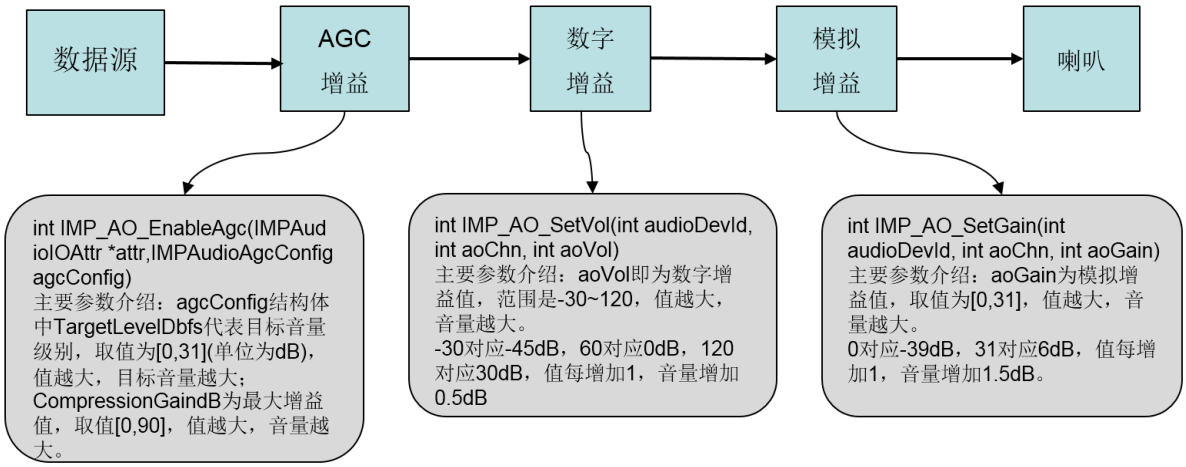

### 7.6.2.2 音频输出

音频输出 Device ID 对应关系， 0：对应默认 SPK 1：对应其他 SPK

音频输出 Channel 当前 API 只支持 1 个通道。

音频输入的音量设置包含模拟增益 gain，agc 增益，数字增益 vol。

增益控制流程如下图所示：




#### 7.6.2.3 音频编码

音频编码目前音频 API 支持 PT_G711A、PT_G711U 和 PT_G726 格式音频编码，如果需要增加新的编码方式，需要调用 IMP_AENC_RegisterEncoder 接口进行注册编码器。

#### 7.6.2.4 音频解码

音频解码目前音频 API 支持 PT_G711A、PT_G711U 和 PT_G726 格式音频解码，如果需要增加新的解码方式，需要调用 IMP_ADEC_RegisterDecoder 接口进行注册解码器。

### 7.6.3 相关变量的定义

#### 7.6.3.1 解码方式

**表 6-5 解码方式**

| | |
|---|---|
| ADEC_MODE_PACK | Pack 方式解码 |
| ADEC_MODE_STREAM | Stream 方式解码 |

#### 7.6.3.2 音频采样频率

**表 6-6 音频采样频率**

| | |
|---|---|
| AUDIO_SAMPLE_RATE_8000 | 8KHz 采样率 |
| AUDIO_SAMPLE_RATE_16000 | 16KHz 采样率 |
| AUDIO_SAMPLE_RATE_24000 | 24KHz 采样率 |
| AUDIO_SAMPLE_RATE_44100 | 44.1KHz 采样率 |
| AUDIO_SAMPLE_RATE_48000 | 48KHz 采样率 |
| AUDIO_SAMPLE_RATE_96000 | 96KHz 采样率 |

#### 7.6.3.3 音频声道模式

表 6-7 音频声道模式

| | |
|---|---|
| AUDIO_SOUND_MODE_MONO | 单声道 |
| AUDIO_SOUND_MODE_STEREO | 双声道 |

#### 7.6.3.4 噪声抑制级别

表 6-8 噪声抑制级别

| | |
|---|---|
| NS_LOW | 低等级级别噪声抑制 |
| NS_MODERATE | 中等级级别噪声抑制 |
| NS_HIGH | 高等级级别噪声抑制 |
| NS_VERYHIGH | 最高等级级别噪声抑制 |

以上表格是在音频中用到的相关枚举定义。

### 7.6.4 IMP_Audio 相关 API 介绍

#### 7.6.4.1 设置音频输入设备属性

| | |
|---|---|
| 函数原型 | int IMP_AI_SetPubAttr(int audioDevId, IMPAudioIOAttr *attr) |
| 功能介绍 | 设置音频输入设备属性 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] attr 音频设备属性指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 需要在 IMP_AI_Enable 前调用 |

#### 7.6.4.2 获取音频输入设备属性

| | |
|---|---|
| 函数原型 | int IMP_AI_GetPubAttr(int audioDevId, IMPAudioIOAttr *attr) |
| 功能介绍 | 获取音频输入设备属性 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[out] attr 音频设备属性指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.3 启用音频输入设备

| | |
|---|---|
| 函数原型 | int IMP_AI_Enable(int audioDevId) |
| 功能介绍 | 启用音频输入设备 |

| | |
|---|---|
| 参数介绍 | [in] audioDevId 音频设备号 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在调用此函数前必须调用 IMP_AI_SetPubAttr( |

#### 7.6.4.4 禁用音频输入设备

| | |
|---|---|
| 函数原型 | int IMP_AI_Disable(int audioDevId) |
| 功能介绍 | 禁用音频输入设备 |
| 参数介绍 | [in] audioDevId 音频设备号 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 与 IMP_AI_Enable 配套使用，在系统休眠前必须执行 IMP_AI_Disable |

#### 7.6.4.5 启用音频输入通道

| | |
|---|---|
| 函数原型 | int IMP_AI_EnableChn(int audioDevId, int aiChn) |
| 功能介绍 | 启用音频输入通道 |
| 参数介绍 | [in] audioDevId 音频设备号<br>[in] aiChn 音频输入通道号 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 必须先使能 device |

#### 7.6.4.6 禁用音频输入通道

| | |
|---|---|
| 函数原型 | int IMP_AI_DisableChn(int audioDevId, int aiChn) |
| 功能介绍 | 禁用音频输入通道 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] aiChn 音频输入通道号 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.7 Polling 音频流缓存

| | |
|---|---|
| 函数原型 | int IMP_AI_PollingFrame(int audioDevId, int aiChn, unsigned int |

| | |
|---|---|
| | timeout_ms) |
| 功能介绍 | Polling 音频流缓存 |
| 参数介绍 | [in] audioDevId 音频设备号<br>[in] aiChn 音频输入通道号<br>[in] timeout_ms Polling 超时时间 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在使用 IMP_AI_GetFrame 之前使用该接口，当该接口调用成功之后表示音频数据已经准备完毕，可以使用 IMP_AI_GetFrame 获取音频数据 |

#### 7.6.4.8 获取音频帧

| | |
|---|---|
| 函数原型 | int IMP_AI_GetFrame(int audioDevId, int aiChn, IMPAudioFrame *frm, IMPBlock block) |
| 功能介绍 | 获取音频帧 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] aiChn 音频输入通道号<br>[out] frm 音频帧结构体指针<br>[in] block 阻塞/非阻塞标识 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.9 获取音频帧

| | |
|---|---|
| 函数原型 | Int IMP_AI_ReleaseFrame(int audioDevId, int aiChn, IMPAudioFrame *frm) |
| 功能介绍 | 释放音频帧 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] aiChn 音频输入通道号<br>[in] frm 音频帧结构体指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.10 设置音频输入通道参数

| | |
|---|---|
| 函数原型 | int IMP_AI_SetChnParam(int audioDevId, int aiChn, IMPAudioIChnParam *chnParam) |
| 功能介绍 | 设置音频输入通道参数 |

| | |
|---|---|
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] aiChn 音频输入通道号.<br>[in] chnParam 音频通道参数. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在 IMP_AI_EnableChn 前调用 |

#### 7.6.4.11 获取音频输入通道参数

| | |
|---|---|
| 函数原型 | int IMP_AI_GetChnParam (int audioDevId, int aiChn, IMPAudioIChnParam *chnParam) |
| 功能介绍 | 获取音频输入通道参数 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] aiChn 音频输入通道号.<br>[out] chnParam 音频通道参数. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.12 设置音频 Amic 通道增益值

| | |
|---|---|
| 函数原型 | int IMP_AI_SetAlcGain(int audioDevId, int aiChn, int aiPgaGain) |
| 功能介绍 | 设置音频 Amic 通道增益值，在模拟 mic ALC 功能开启时有效。 |
| 参数介绍 | [in] audioDevId 音频输入设备号.<br>[in] aiChn 音频输入通道号.<br>[out] aiPgaGain 音频输入增益,范围[0 ~ 7]. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | aiPgaGain 的范围为[0 ~ 7],如果输入的值小于 0,则 aiPgaGain 的值将会\n<br>被设置为 0.如果值大于 7,aiGain 的值会被设置为 7. |

#### 7.6.4.13 获取音频 Amic 通道增益值

| | |
|---|---|
| 函数原型 | int IMP_AI_GetAlcGain(int audioDevId, int aiChn, int *aiPgaGain) |
| 功能介绍 | 获取音频 Amic 通道增益值，在模拟 mic ALC 功能开启时有效。 |
| 参数介绍 | [in] audioDevId 音频输入设备号.<br>[in] aiChn 音频输入通道号.<br>[out] aiPgaGain 音频输入增益. |

| | |
|---|---|
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.14 设置音频输入通道音量

| | |
|---|---|
| 函数原型 | int IMP_AI_SetVol(int audioDevId, int aiChn, int aiVol) |
| 功能介绍 | 设置音频输入音量 |
| 参数介绍 | [in] aiDevId 音频输入设备号.<br>[in] aiChn 音频输入通道号.<br>[in] aiVol 音频输入音量大小(数字增益值). |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 音量的取值范围为[-30 ~ 120]. -30 代表静音,120 表示将声音放大 30dB,步长 0.5dB。其中 60 是音量设置的一个临界点，在这个值上软件不对音量做增加或减小，当音量值小于 60 时，每下降 1，音量减小 0.5dB；当音量值大于 60 时，上增加 1，音量增加 0.5dB。 |
| 注意 | 如果输入的 aiVol 超过了[-30 ~ 120]的范围,小于-30 的将会取-30,大于 120 的取 120 |

#### 7.6.4.15 获取音频输入通道音量

| | |
|---|---|
| 函数原型 | int IMP_AI_GetVol(int audioDevId, int aiChn, int aiVol) |
| 功能介绍 | 获取音频输入音量 |
| 参数介绍 | [in] aiDevId 音频输入设备号.<br>[in] aiChn 音频输入通道号.<br>[out] vol 音频输入通道音量. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.16 设置音频输出设备属性

| | |
|---|---|
| 函数原型 | int IMP_AO_SetPubAttr(int audioDevId, IMPAudioIOAttr *attr) |
| 功能介绍 | 设置音频输出设备属性 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] attr 音频输出设备属性指针. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |

| 注意 | 无 |
|---|---|

#### 7.6.4.17 获取音频输出设备属性

| 函数原型 | int IMP_AO_GetPubAttr(int audioDevId, IMPAudioIOAttr *attr) |
|---|---|
| 功能介绍 | 获取音频输入输出设备属性 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[out] attr 音频输出设备属性指针. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.18 启用音频输出设备

| 函数原型 | int IMP_AO_Enable(int audioDevId) |
|---|---|
| 功能介绍 | 启用音频输出设备 |
| 参数介绍 | [in] audioDevId 音频设备号 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 在使能之前必须先调用 IMP_AO_SetPubAttr |

#### 7.6.4.19 禁用音频输出设备

| 函数原型 | int IMP_AO_Disable(int audioDevId) |
|---|---|
| 功能介绍 | 禁用音频输出设备、 |
| 参数介绍 | [in] audioDevId 音频设备号 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.20 启用音频输出通道

| 函数原型 | int IMP_AO_EnableChn(int audioDevId, int aoChn) |
|---|---|
| 功能介绍 | 启用音频输出通道 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] aoChn 音频输出通道号. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |

| 注意 | 无 |
|---|---|

#### 7.6.4.21 禁用音频输出通道

| 函数原型 | int IMP_AO_DisableChn(int audioDevId, int aoChn) |
|---|---|
| 功能介绍 | 禁用音频输出通道 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] aoChn 音频输出通道号. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.6.4.22 设置音频输出通道音量

| 函数原型 | int IMP_AO_SetVol(int audioDevId, int aoChn, int aoVol) |
|---|---|
| 功能介绍 | 设置音频输出通道音量 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] aoChn 音频输出通道号.<br>[in] aoVol 音频输出音量. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 音量的取值范围为[-30 ~ 120]. -30 代表静音,120 表示将声音放大 30dB,步长 0.5dB。其中 60 是音量设置的一个临界点，在这个值上软件不对音量做增加或减小，当音量值小于 60 时，每下降 1，音量减小 0.5dB；当音量值大于 60 时，上增加 1，音量增加 0.5dB。 |
| 注意 | 如果输入的 aoVol 超过了[-30 ~ 120]的范围,小于-30 的将会取-30,大于 120 的取 120 |

#### 7.6.4.23 获取音频输出通道音量

| 函数原型 | int IMP_AO_GetVol(int audioDevId, int aoChn, int aoVol) |
|---|---|
| 功能介绍 | 获取音频输出通道音量 |
| 参数介绍 | [in] audioDevId 音频设备号.<br>[in] aoChn 音频输出通道号.<br>[out] aoVol 音频输出音量. |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

# 7.7 IMP_OSD 模块

OSD 模块，可在视频流上叠加图片、位图、直线、矩形框。

## 7.7.1 模块介绍

OSD 的全称是 On-Screen Display。模块的功能就是在每张片源中叠加线条、图片等信息。

## 7.7.2 相关概念

### 7.7.2.1 Region

Region 即是叠加区域，在 API 中简称 Rgn。每个 Region 具有一定的图像信息，可以经过 OSD 模块叠加后，与背景图像合为一张图片。对于图片的叠加，还可以实现 Alpha 效果。关于各种叠加类型的详细介绍请参考 osd_region_type。

### 7.7.2.2 Region type

Region 有几种类型，分别为：

```
1. OSD_REG_LINE:  直线
2. OSD_REG_RECT:  矩形框
3. OSD_REG_BITMAP:    位图
4. OSD_REG_COVER:     遮挡
5. OSD_REG_PIC:       图片
```

其中，位图与图片的区别是，位图仅进行像素的单色覆盖，而图片是 RGBA 图像的 Alpha 叠加。

## 7.7.3 模块功能

OSD 模块支持线条、矩形框、位图叠加、矩形遮挡和图片叠加。 线条、矩形框和位图采用软件实现；矩形遮挡和图片叠加采用硬件实现。

## 7.7.4 模块使用

OSD 的使用一般有以下几个步骤：
**第一步**：创建 OSD 组
**第二步**：绑定 OSD 组到系统中
**第三步**：创建 OSD 区域
**第四步**：注册 OSD 区域到 OSD 组中
**第五步**：设置 OSD 组区域属性和区域属性
**第六步**：设置 OSD 功能开关
简单的来说 IPU 最多 4 层 OSD 叠加功能，背景图片来自 OSD 的背景通道，输入数据类型可以选择从以下 4 类：ARGB、RGB32、NV12、NV21。对于 OSD 来说输出与输入图像数据类型应该是相同的类型。

当 OSD 的操作时，应注意下列事项：

1) 每个通道可以独立设置的参数。

3) OSD 只能支持整个背景图片叠加到 OSD 或重叠照片 OSD，如果背景图片是形式 OSD 背景频道，重叠的照片到 OSD 的模式将节省带宽，因为只有重叠的图片由 DMA 获取。

3) 4 个 OSD 通道也可以支持 RGB 向 YUV 和 YUV 向 RGB 转换。

4) 4 个 OSD 通道可以支持掩码功能，透明度选择功能，您可以从以下几种方式中选择透明度，如：pixel alpha、global alpha 和 (pixel alpha)*(global alpha)。

5) 如果所有 4 个 OSD 通道被禁用，在不改变数据值情况下，分离通道可以作为背景数据移动。

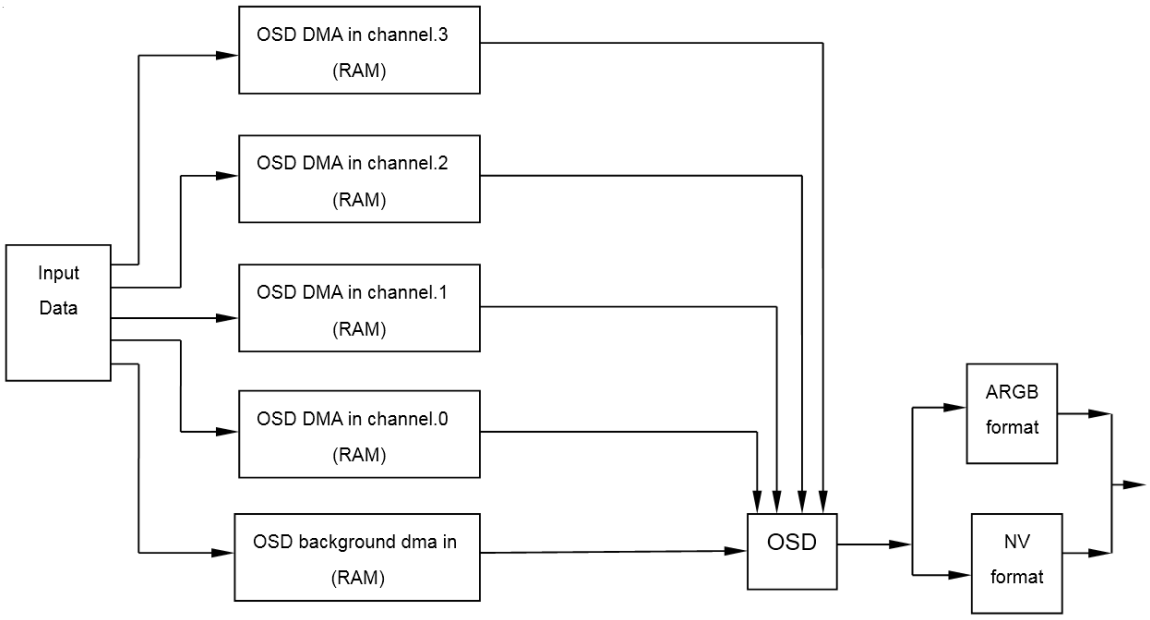



图 6-11 OSD 叠加原理

由上图可知，OSD 是 4 个 OSD 通道和一个背景通道还有 OSD 叠加器组成。实现原理是通过背景通道输入背景图片和 OSD 通道输入要叠加图片，然后通过 OSD 器实现图片的叠加，然后通过图片格式输出到 FIFO。然后再输出到 DMA（out）中，进行输出。

叠加的原理是在背景图片上画出一个区域，然后把要叠加的图片放在这个区域上，针对于这一层其他区域可以通过设置透明度，把这一层其他区域设置为透明。然后就实现了叠加功能。针对 T31 相应的芯片，支持 4 层叠加，就是 4 个叠加通道一起叠加。

图 6-12 OSD 叠加效果

## 7.7.5 IMP_OSD 相关 API 介绍

### 7.7.5.1 设置区域属性

| | |
|---|---|
| 函数原型 | int IMP_OSD_SetRgnAttr(IMPRgnHandle handle, IMPOSDRgnAttr *prAttr) |
| 功能介绍 | 设置区域属性 |
| 参数介绍 | [in] handle 区域句柄，IMP_OSD_CreateRgn 的返回值<br>[in] prAttr OSD 区域属性 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 调用此 API 时要求对应的区域已经创建 |
| 注意 | 无 |

### 7.7.5.2 获取区域属性

| | |
|---|---|
| 函数原型 | int IMP_OSD_GetRgnAttr(IMPRgnHandle handle, IMPOSDRgnAttr *prAttr) |
| 功能介绍 | 获取区域属性 |
| 参数介绍 | [in] handle 区域句柄，IMP_OSD_CreateRgn 的返回值<br>[out] prAttr OSD 区域属性 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 调用此 API 时要求对应的区域已经创建 |
| 注意 | 无 |

#### 7.7.5.3 更新区域数据属性

| | |
|---|---|
| 函数原型 | int IMP_OSD_UpdateRgnAttrData(IMPRgnHandle handle, IMPOSDRgnAttrData *prAttrData) |
| 功能介绍 | 更新区域数据属性，只针对 OSD_REG_BITMAP 和 OSD_REG_PIC 的区域类型 |
| 参数介绍 | [in] handle 区域句柄，IMP_OSD_CreateRgn 的返回值<br>[in] prAttrData OSD 区域数据属性 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 调用此 API 时要求对应的区域已经创建且区域属性已经设置成 OSD_REG_BITMAP 或 OSD_REG_PIC |
| 注意 | 无 |

#### 7.7.5.4 设置 OSD 组区域属性

| | |
|---|---|
| 函数原型 | int IMP_OSD_SetGrpRgnAttr(IMPRgnHandle handle, int grpNum, IMPOSDGrpRgnAttr *pgrAttr) |
| 功能介绍 | 设置 OSD 组区域属性 |
| 参数介绍 | [in] handle 区域句柄，IMP_OSD_CreateRgn 的返回值<br>[in] grpNum OSD 组号<br>[in] pgrAttr OSD 组区域属性 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 调用此 API 时要求对应的 OSD 组已经创建,区域已经创建并注册 |
| 注意 | 无 |

#### 7.7.5.5 获取 OSD 组区域属性

| | |
|---|---|
| 函数原型 | int IMP_OSD_GetGrpRgnAttr(IMPRgnHandle handle, int grpNum, IMPOSDGrpRgnAttr *pgrAttr) |
| 功能介绍 | 获取 OSD 组区域属性 |
| 参数介绍 | [in] handle 区域句柄，IMP_OSD_CreateRgn 的返回值<br>[in] grpNum OSD 组号<br>[out] pgrAttr OSD 组区域属性 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 调用此 API 时要求对应的 OSD 组已经创建,区域已经创建并注册 |
| 注意 | 无 |

## 7.8 IMP_CSC 图片格式转换

CSC 可以实现把输入的图片转化成硬件支持转化的图像 HSV、NV12、NV21、RGB32、和 ARGB 模式中任意一种格式。

### 7.8.1 模块介绍

CSC 的全称是 color space conversion。实现的功能是把输入的图片格式转换成硬件支持的任意一种格式。

### 7.8.2 CSC 支持的转化

表 6-8 CSC 支持的格式转化

| 输入格式 | 输出格式 | Destination |
|---|---|---|
| ARGB | ARGB/RGB/NV12/NV21/HSV | DDR |
| RGB | ARGB/RGB/NV12/NV21/HSV | DDR |
| NV12 | ARGB/RGB/NV12/NV21/HSV | DDR |
| NV21 | ARGB/RGB/NV12/NV21/HSV | DDR |

### 7.8.3 CSC 转化流程

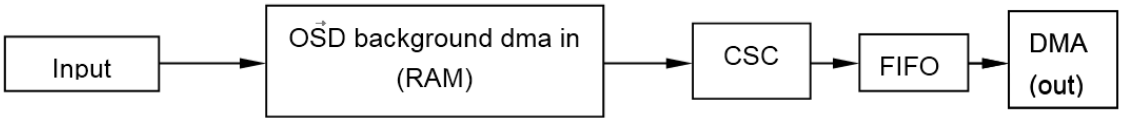



图 6-13 CSC 转化流程

CSC 就是实现把输入的图片格式转化成另外一种硬件支持的图片格式。具体操作是输入一张图片到背景通道，然后会经过 CSC 进行图片的格式的转化。最后通过 FIFO 输出到 DMA 进行输出。在 OSD 的四个通道也包含的有 CSC 功能，是把要叠加的图片转换成跟背景图片格式一样的格式。

所以一般只承认 CSC 只有一个通道，但是如果把 OSD 的 4 个通道算上就有 5 个具有 CSC 功能的通道。

在这里要说一下，SDK 给出的 sample 在进行 CSC 功能的时候，使用的是通道 2 扩展通道，图片或者视频是从主码流或者次码流拷贝的输入到扩展通道 2 里面，然后在对这些数据进行相应的操作。

## 7.9 IMP_IVS 智能分析

IVS 智能分析通用接口 API。

### 7.9.1 模块介绍

IMP IVS 通过 IVS 通用接口 API 调用实例化的 IMPIVSInterface 以将智能分析算法嵌入到 SDK 中来分析 SDK 中的 frame 图像。

IMPIVSInterface 为通用算法接口，具体算法通过实现此接口并将其传给 IMP IVS 达到在 SDK 中运行具体算法的目的。

一个 channel 有且仅为单个算法实例的载体，必须将具体实现的通用算法接口传给具体的 channel 才能在 SDK 中运行算法。

IMPIVSInterface 成员 param 为成员函数 init 的参数。

IMP_IVS 会在传给成员函数 ProcessAsync 参数的 frame 时对其进行外部加锁，ProcessAsync 必须在使用完 frame 后调用 IMP_IVS_ReleaseData 释放 frame，以免死锁。

除上述通过绑定在数据流中的 IMP IVS 实现算法调用之外，还提供一种非绑定的方式实现算法调用，即获取 framesouce channel 的 frame 图像，直接调用 IMPIVSInterface 的成员函数实现算法处理

### 7.9.2 使用方法

#### 7.9.2.1 绑定模式的算法使用方法

以移动侦测算法为例，函数的具体实现见 sample-move_c.c 文件，步骤如下：

**第一步：**初始化系统，可以直接调用范例中的 sample_system_init()函数。整个应用程序只能初始化系统一次，若之前初始化了，这儿不需要再初始化。

**第二步：**初始化 framesource 若算法所使用的 framesource 通道已创建，直接使用已经创建好的通道即可。若算法所使用的 framesource 通道未创建，可以调用范列中的 sample_framesource_init(IVS_FS_CHN, &fs_chn_attr)进行创建。

**第三步：**创建 ivs 具体算法通道组。多个算法可以共用一个通道组，也可以分别使用通道组 sample_ivs_move_init()。

**第四步：**绑定算法通道组和 framesource 通道组。

**第五步：**启动 framesource 和算法。建议算法通道号和算法编号一致，以便可以直接对应当前操作哪一个算法。

**第六步：**获取算法结果 Polling 结果、获取结果和释放结果必须严格对应，不能中间有中断；只有 Polling 结果正确返回，获取到的结果才会被更新，否则获取到的结果无法预知。

**第七步：**释放资源。

#### 7.9.2.2 非绑定模式的算法使用方法

以移动侦测算法为例，函数的具体实现见 sample-Framesource-algo.c 文件，步骤如下：

**第一步：**初始化系统，可以直接调用范例中的 sample_system_init()函数。

**第二步：**初始化 framesource 若算法所使用的 framesource 通道已创建，直接使用已经创建好的通道即可。若算法所使用的 framesource 通道未创建，可以调用范列中的 sample_framesource_init(IVS_FS_CHN, &fs_chn_attr)进行创建。

**第三步：**初始化 interface 句柄，调用 interface->init()初始化算法，调用 IMP_FrameSource_SetFrameDepth()设置帧缓存。

**第四步：**循环调用 IMP_FrameSource_GetFrame 获取 frame，interface->preProcessSync 算法预处理，interface->processAsync 算法处理过程，interface->getResult 获取算法结果，interface->releaseResult 释放算法结果，IMP_FrameSource_ReleaseFrame 释放 frame。

**第五步：**IMP_FrameSource_SetFrameDepth 缓存深度回 0，释放缓存，反初始化 interface 句柄，反初始化 framesouce 和系统。

## 7.9.3 IMP_IVS 相关 API 介绍

### 7.9.3.1 通道开始接收图像

| | |
|---|---|
| 函数原型 | int IMP_IVS_StartRecPic(int ChnNum) |
| 功能介绍 | 通道开始接收图像 |
| 参数介绍 | [in] ChnNum 通道号 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 通道号为 Chnnum 的 IVS 功能通道开始接收图像做智能分析 |
| 注意 | 无 |

### 7.9.3.2 通道停止接收图像

| | |
|---|---|
| 函数原型 | Int IMP_IVS_StopRecPic(int ChnNum) |
| 功能介绍 | 通道停止接收图像 |
| 参数介绍 | [in] ChnNum 通道号 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 通道号为 Chnnum 的 IVS 功能通道停止接收图像，暂停智能分析 |
| 注意 | 无 |

### 7.9.3.3 获得 IVS 功能的智能分析结果

| | |
|---|---|
| 函数原型 | int IMP_IVS_GetResult(int ChnNum, void **result) |
| 功能介绍 | 获得 IVS 功能计算出的智能分析结果 |
| 参数介绍 | [in] ChnNum IVS 功能对应的通道号<br>[in] result IVS 功能对应的通道号输出的结果，返回此通道对应的智能分析算法的结果指针，外部客户无需分配空间 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 根据不同 IVS 功能绑定的通道,输出其对应的结果 |
| 注意 | 无 |

#### 7.9.3.4 释放 IVS 功能的智能分析结果资源

| | |
|---|---|
| 函数原型 | int IMP_IVS_ReleaseResult(int ChnNum, void *result) |
| 功能介绍 | 释放 IVS 功能计算出的结果资源 |
| 参数介绍 | [in] GrpNum 通道组号<br>[in] ChnNum IVS 功能对应的通道号<br>[in] result IVS 功能对应的通道号输出的结果 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 根据不同 IVS 功能绑定的通道,释放其输出的结果资源 |
| 注意 | 无 |

#### 7.9.3.5 设置通道算法参数

| | |
|---|---|
| 函数原型 | int IMP_IVS_SetParam(int chnNum, void *param) |
| 功能介绍 | 设置通道算法参数 |
| 参数介绍 | [in] ChnNum IVS 功能对应的通道号<br>[in] param 算法参数虚拟地址指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

#### 7.9.3.6 获取通道算法参数

| | |
|---|---|
| 函数原型 | int IMP_IVS_GetParam(int chnNum, void *param) |
| 功能介绍 | 设置通道算法参数<br>参数： |
| 参数介绍 | [in] ChnNum IVS 功能对应的通道号<br>[in] param 算法参数虚拟地址指针 |
| 返回值 | 0：成功<br>非 0：失败 |
| 备注 | 无 |
| 注意 | 无 |

# 8 FAQ（常见问题解答）

Q-1001: 关于 uclibc 的 Toolchain

**A**: ISVP 的 Toolchain 是 multilib 的，即同时包含 glibc 和 uclibc 的链接库。因此 glibc 的应用和 uclibc 的应用均适用此 Toolchain 进行编译。关于使用方法请参考 《2.3 Toolchain 使用说明》 和 《4.5 应用程序编译方法》。

Q-1002: T10 启动失败或者外设（TF 卡等）初始化异常

**A**: T10 由于 SMIC 工艺的问题，在电源完全掉电的情况下，会从 IO 口漏电到 T10 内的各电源网络。因此如果板子在没有插电源的情况下，插入串口，会漏电到各个外设，漏电压可能处在逻辑不稳定态，可能导致外设的逻辑出错，进入 Invalid State。这时上电，这些外设可能工作异常，比如 TF 卡初始化失败。因此需要注意，在调试时，尽量先上电，再插串口。

Q-1003: rmmod 出错

**A**: 移除 ko 时出现错误：

rmmod: can't change directory to '/lib/modules': No such file or directory，或者 rmmod: can't change directory to '3.10.14-00310-g4e55a44': No such file or directory。新版本的 busybox 依赖/lib/modules/[uname]文件夹，一个参考的办法是在 lib/目录下创建一个 modules 的软连接，再使用 SDK 中 tools/device/script/mk_ko_dir.sh，在设备启动后自动创建/lib/modules/3.10.14-00310-g4e55a44/这样的文件夹。

Q-1004: watchdog close 文件句柄时提示'watchdog watchdog0:watchdog dit not stop'

**A**:wdt 驱动支持 Magic Close，在 close 前需要往 wdt 中写入一个大写字符 'V'，否则 close 时不关闭 wdt（会超时重启），这样能防止系统崩溃导致的意外退出。

代码片段：

```
int fd,retval;
int timeout = 0;
char a = 'V';
fd = open("/dev/watchdog",O_WRONLY);
```

```
if(fd < 0){
    perror("/dev/watchdog");
    return -1;
}
if (argc > 1) {
    timeout = atoi(argv[1]);
    retval =  ioctl(fd, WDIOC_SETTIMEOUT,&timeout);
    if (retval == -1) {
        perror("WDIOC_SETTIMEOUT ioctl");
        exit(errno);
    }
}
for(;;){
    printf("feed wdt \n");
    retval =  ioctl(fd, WDIOC_KEEPALIVE);
    if (retval == -1) {
        perror("WDIOC_KEEPALIVE ioctl");
        exit(errno);
    }
    sleep(4);
}
if (write(fd,&a,1) != 1);
    printf("write error\n");

close(fd);
```

Q-1005: flash 操作出错，类似打印"jffs2:Node totlen on flash (oxffffffff) != totlen from node ref(0x00000001) "

**A**: 这个可能可擦除大小有关，为了提高软件的兼容性，把 flash 的擦除大小统一改成了 32K，但是有的 flash 不支持 32K 擦除，此时如果软件仍把擦除大小设成 32K，就会出现问题。

已知有此问题的 flash: MX25L6406E，与它有相同 id 的 MX25L6433F 是支持 32K 擦除的，用 MX25L6433F 是没问题的。 客户可考虑更换 flash 类型或修改 flash 擦除大小。

修改路径：
T10-->arch/mips/xburst/soc-t20/chip-t20/isvp/common/spi_bus.c
T20-->arch/mips/xburst/soc-t20/chip-t20/isvp/common/spi_bus.c

修改 erasesize，32 * 1024 改为 64 * 1024

```
{
```

```
        .name           = “MX25L64XXX”,
        .pagesize       = 256,
        .sectorsize     = 4 * 1024,
        .chipsize       = 8192 * 1024,
        .erasesize      = 32 * 1024,
        .id             = 0xc22017,

        .block_info     = flash_block_info,
        .num_block_info = ARRAY_SIZE(flash_block_info),

        .addrsize       = 3,
        .pp_maxbusy     = 3,            /* 3ms */
        .se_maxbusy     = 400,          /* 400ms */
        .ce_maxbusy     = 8 * 10000,    /* 80s */

        .st_regnum      = 3,
    #ifdef CONFIG_SPI_QUAD
        .quad_mode = &flash_quad_mode[0],
    #endif
    },
```

Q-1006: isp 驱动报错”Sensor outputs bigger resolution than that ISP device can't deal with”

**A**: T10 ISP 支持的最大分辨率是 1280*1024，T20 ISP 支持的最大分辨率是 2048*1536，T31 ISP 支持的最大分辨率是 2592x2048 出现这个问题的原因可能有：

1、设置输出的分辨率超过了 ISP 支持的最大分辨率，这种情况需要客户设置正确的分辨率参数

2、可能把在 T10 内核上编译的 tx-isp.ko 用在了 T20 上。这种情况需要客户在 T20 的重新编译 isp 驱动，需要 make clean 一下，先编译 isp 驱动再编译 sensor 驱动。同时建议客户 T10 和 T20 切换时，所有驱动最好用对应的内核都重新编译一下。

Q-2001: 图像显示不正常

A: 如果是颜色不正常，可能的原因有：

1、Sensor Data 线对齐方式设置错误。需要根据 SCH 的设计，insmod 时加入参数，例如：

`$ insmod sensor_jxh42.ko sensor_gpio_func=1` ，其中参数值的意义为：

0: GPIO-A 低 10bit

1: GPIO-A 高 10bit

2: GPIO-A12bit

3: GPIO-A 低 8bit

4: GPIO-A 高 8bit

2、图像偏绿的原因可能是没有在文件系统中放置 bin 文件(/etc/sensor/xxx.bin)，在此情况下会有内核打印：**ISP: open %s file for isp calibrate read failed**。或者 Sensor bin 文件与 ISP driver 不匹配，会出现如下打印：

```
#########################################################################
#### The version of ov9750.bin doesn't match with driver! ####
#### The version of ov9750.bin is xxx, the driver is yyy ####
#########################################################################
```

3、图像偏红的原因可能是没有加 IR-Cut
图像花屏的问题可能是 Sensor DVP 线(pclk, sync)的驱动能力不足

B. 如果是夜视图像不正常，需要检查：
1、是否使用对应的夜视 setting
2、是否将 IR-CUT 切换到白片
3、红外灯是否打开
C. 如果没有图像：
1、检查硬件的连接是否良好，可以更换排线或者 sensor 板试一下
2、是否加载对应 sensor 的驱动 ko
3、分辨率、裁剪、缩放等配置是否正确
D. 图像条纹：
1、检查 senosr 初始化 setting 是否正确
2、检查 sensor 板温度是否正常
3、senosr 的 AVDD 以及 DVDD 是否正确，AGAIN 是否设置过大
D. 图像闪烁：
1、判断是工频干扰的条纹还是整帧闪烁
2、检查 sensor 驱动中配置的 delay time 是否正确
3、检查 sensor 驱动中 gain 的转换算法是否正确

Q-2002: 关于设置帧率接口，帧率如何定义？Sensor FPS 和 Encoder FPS 有什么区别或联系？

**A**: ISVP 中的帧率为分数形式，由两个数据组成：Num 和 Den，其中 Num 是分子（被除数），Den 是分母（除数），因此，FPS=Num/Den，例如：对于 FPS=25，Num=25，Den=1，对于 FPS=12.5，Num=25，Den=2。此外，应确保 Num 和 Den 的最大公约数为 1。

关于 Sensor FPS 和 Encoder FPS 有什么区别或联系：

Sensor 的 FPS 为系统的输入帧率，通过配置 Sensor 的 VTS 参数实现。一般在夜视时需要降低 Sensor 帧率，以获得在较低 Gain 下足够的曝光。Encoder（H264）的 FPS 为输出的 H264 视频流帧率，降低 Encoder FPS 可有效降低码率。Encoder FPS 通过丢帧实现，因此 Encoder FPS **<=** Sensor FPS，值得注意的是，当改变 Sensor FPS 时，Encoder FPS 会被同时自动改变，而改变 Encoder FPS 时，Sensor FPS 不会自动改变。

改变 Encoder 的帧率也可以实现两路编码不同帧率。

Q-2003: 如何修改 Sensor 默认帧率

**A**: 开启 ISP 数据流的操作流程如下：

```
IMP_ISP_Open();
IMP_ISP_AddSensor();
IMP_ISP_EnableSensor();
IMP_ISP_EnableTuning();
/**
* 如果想改变 ISP 原始输出帧率，这时请调用 IMP_ISP_Tuning_SetSensorFPS();
 * 如果不改变 ISP 原始输出帧率，不进行操作。
 * /
FrameSourceInit()
IMP_FrameSource_CreateChn()
IMP_FrameSource_EnableChn()
```

Q-2004: T10 如何进行三路 H264 编码

**A**: 有两种方式实现三路 H264 编码。

在创建一个新的 Encoder Group，Bind 在主码流或次码流的 OSD 后面，注册一个 Encoder Channel 到这个 Group，设定编码分辨率，即可获得第三路码流。三路码流编码帧率受硬件限制，请合理分配

2、创建 FrameSource 拓展通道，Bind OSD 以及 Encoder Group。拓展通道的 FrameSource 分辨率比主码流小，FrameSource 与 Encoder 设置为同一分辨率即可。拓展通道为软件实现（T10），需要占用一定的 CPU load 以及内存资源。

Q-2005: T10 次码流如何进行 D1 分辨率编码

**A**: T10 ISP 次码流输出最大为 640x480，如果需要 D1 分辨率的编码，可以将 Encoder 通道的分辨率设置为 D1 即可，Encoder 与前一级 Group 分辨率不一致时会进行 Scale。

Q-3001: 音频的一些参数设置注意事项

**A**: 主要包括以下几点：

1、接口 IMP_AI_SetVol(int audioDevId, int aiChn, int aiVol)注意事项：

aiVol 的取值范围为 -30 ～ 120，数值每增加 1，音量增加 0.5dB。其中 aiVol = 60 是软件设置音量的一个临界点，在这个点上软件不对音量做增加或减小，aiVol < 60 时，每下降 1，音量减小 0.5dB；aiVol > 60 时，上增加 1，音量增加 0.5dB

2、接口 IMP_AI_SetGain(int audioDevId, int aiChn, int aiGain) 和 IMP*AO* SetGain(int audioDevId, int aoChn, int aoGain)：

硬件增益接口，该接口的增益设置情况根据不同的芯片会有所不同，请参考对应的 SDK 头文件

软件设置音量接口和硬件设置增益接口应该相互协调使用，否则会造成音质破损或者音质太小

关于声音的噪声抑制，自动增益，高通滤波的效果只有在回声消除功能没有打开时才会有效，否则的话则打开无效。回声消除中自带自带这些功能，无需另行开启

5、请保证送入 SDK 的音频数据是 10ms 音频数据的整数倍（EXP：8K 采样率，送入的数据为：8000 × 2 / 100 = 160byte 的整数倍）

# 9 附录

## 9.1 制作启动卡

对于第一次启动板子，需要从 SD 卡启动，然后将编译好的 uboot 烧录到 flash 中。以后就可以直接从 flash 中启动了。

### 9.1.1 制作新分区

格式化 sd 卡，并将卡启动 uboot 烧录到 SD 卡

此步骤只需执行一次，若不更新卡起 uboot 则不需要再次执行此步骤。经过此步骤处理的 SD 卡可当作正常卡使用

将 SD 卡插入电脑

注意!!!：确定插入你的电脑上 TF 卡对应的设备节点文件是不是 sdb(有可能是 sdc 或者 sdd，可以通过插拔 SD 卡确认)；如果对应错了可能会破坏电脑的硬盘文件系统。

A. 卸载 sd 卡

```
umount /media/user/*
```

B. 将 SD 卡重新分区

```
# fdisk /dev/sdb
> o  --创建一个新的分区表
> n  --添加一个分区 n
--此时会提示：
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p):
Using default response p
--这里回车默认是 p，创建主分区
```

```
分区号 (1-4，默认为 1)：
将使用默认值 1
--这里回车默认创建 1 号分区
起始 sector (2048-15130623，默认为 2048)：
将使用默认值 2048
Last sector, +扇区 or +size{K,M,G} (2048-15130623，默认为 15130623)：
将使用默认值 15130623
--2048 号扇区在 1MB 的位置，给 uboot 空出了空间
最后一步，输入 w，向 SD 卡中写入分区表
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.
```

到这里分区表已创建完成

D．执行 sync，拔卡，重新插卡，PC 将重新识别分区

E．格式化 sd 卡为通用的 VFat 文件系统：

```
# mkfs.vfat /dev/sdb1        //这个里重新分区的是 sdb1
```

F．**编译卡起 u-boot**，选择对应的 uboot 类型(3.1 uboot 编译)，将生成的 u-boot-with-spl.bin 文件烧录进 sd 卡的 17KB 偏移处(注意是 dev/sdb 分区，不是 dev/sdb1)：

```
# dd if=u-boot-with-spl.bin of=/dev/sdb bs=1024 seek=17
```

到此，可以用来烧录的 SD 卡制作完成。下面可以使用这个启动卡启动板子。

### 9.1.2 SD 卡启动

1) bootload 默认最先从读 nor 读取启动信息，如果 nor 原本烧写过 uboot，现在想要从卡启，需要做如下操作：

   A. 先断电，然后同时按住 reset 键和 boot 键，上电等 2s；

   B. 然后，先松开 reset 键，等 2s，再松开 boot 键。

2) 使用 sd 卡启动开发板，开机前几秒迅速按下回车键，进入 uboot 命令行；

A．首先，将需要烧录的文件读到内存中，以 u-boot 为例：

B．把内存先全部清为全 f

```
isvp# mw.b 0x80600000 0xff 0x1000000  --第三个参数是大小，这里清了 16MB
```

C．ls SD 卡中的文件

```
isvp# fatls mmc 0
   222944   u-boot-with-spl.bin
   2368879   uimage
5 file(s), 0 dir(s)
```

D．load 文件到内存(这里烧录的是 nor 启 uboot)

```
isvp# fatload mmc 0 0x80600000 u-boot-with-spl.bin
```

E．然后，将内存中的数据写入 Nor

```
isvp# sf probe
isvp# sf erase 0x0 0x40000  --擦除 uboot 分区大小 256K
isvp# sf write 0x80600000 0x0 0x40000  --将 uboot 写入 Nor 0x0~0x40000 处
```

F. 此时已经把 uboot 烧录到 flash 中，下次启动可以直接从 flash 中的 uboot 启动。

## 9.2 调试说明

### 9.2.1 ISP 如何抓 RAW 图

当我们需要获取 RAW 原始图像数据时，我们可以在开发板上直接输出命令抓取图像数据。

命令使用帮助：

```
$ echo help > proc/jz/isp/isp-w02
```

抓取图像：

```
$ echo snapraw [savenum] > /proc/jz/isp/isp-w02    //savenum 抓取的图像数
```

保存图像：

```
$ echo snapraw [savenum] > /proc/jz/isp/isp-w02    //savenum 保存的图像数
```

最终保存的图像数据存放在 /tmp 目录下

### 9.2.2 如何抓取 Log

出现问题时，需要抓取 kernel 以及 lib 的 log。

kernel log 抓取命令：

```
$ dmesg > /tmp/time.dmesg。
```

lib log 抓取方法：

```
$ logcat -c time > /tmp/time.log
```

命令大概运行 3 秒后 ctrl+c 终止程序（logcat 命令在 sdk tools 目录下）。

### 9.2.3 读写 eth phy 寄存器

PHY 是 IEEE802.3 中定义的一个标准模块，可以对 PHY 的行为、状态进行管理和控制，而具体管理和控制动作是通过读写 PHY 内部的寄存器实现的。

Uboot 环境下：

读格式：ethphy read addr

```
isvp_t31# ethphy read 0
```

写格式：ethphy write addr data

```
isvp_t31# ethphy write 0 0
```

Kernel 环境下：

读格式：reg:phy 地址-phy 寄存器地址(phy 地址默认是 0，phy 寄存器地址根据标准手册查看)

```
$ echo r reg:0-0 > /proc/jz/mdio/cmd
```

写格式：reg:phy 地址-phy 寄存器地址-写入数据

```
$ echo w reg:0-0-0 > /proc/jz/mdio/cmd
```

### 9.2.4 RMEM 查看与设置

1) 内存的使用说明

在 uboot 的 cmdline 里面有下面内存配置

```
mem=42012K@0x0 ispmem=8100K@0x2907000 rmem=15424K@0x30F0000
mem 是分配给 linux 系统使用的内存。
rmem 是 SDK 保留使用的内存，使用该内存的地方包括，ISP dmaout 的轮转 buf、编码模块的输入输出 buf，OSD 模块的叠加图片。
```

2) rmem 内存大小如何确定

```
A. rmem 的内存可以使用 SDK 中的 rmem 计算器 excel 表格来计算得到。
B. 先把 rmem 设置大一下，先让系统跑起来，然后执行 impdbg --sys;logcat 可以看到 rmem 实际使用的内存和剩余内存，然后把剩余内存从 rmem 拿出来给 mem 使用。
```

3) rmem 内存的优化

OSD 内存占用默认是 1M，可以根据实际产品功能调小，例如 512K。可以通过下面接口设置 OSD 的 rmem 的内存占用。OSD 占用的内存大小是叠加最大图片的 width*height*4。

内存优化调用(默认是 1M)

```
extern "C" {
//设置 OSD 模块使用大小
extern int IMP_OSD_SetPoolSize(int size);
}
初始化 SDK 之前调用
IMP_OSD_SetPoolSize(512*1024);
```

## 9.2.5 如何保留&恢复现场

对于 Nor Flash，可以把 Flash 内容保存下来。方法如下：

```
$ dd if=/dev/mtd0 of=/tmp/mtd0
$ dd if=/dev/mtd1 of=/tmp/mtd1
$ dd if=/dev/mtd2 of=/tmp/mtd2
$ dd if=/dev/mtd3 of=/tmp/mtd3
$ cd tmp
$ cat mtd0 mtd1 mtd2 mtd3 > flash.bin
$ rm mtd0 mtd1 mtd2 mtd3
把 flash.bin 取出即可
或者：
$ cat /dev/mtd0 /dev/mtd1 /dev/mtd2 /dev/mtd3 > flash.bin
```

在需要恢复现场时，在 u-boot 中把 Nor Flash 完全擦除，再把 flash.bin 完全写入即可。

## 9.2.6 Coredump 使用方法

### 9.2.6.1 coredump 介绍

当程序运行的过程中异常终止或者崩溃，操作系统会将程序当时的内存状态记录下来，保存在一个文件中，这种行为就叫做 core dump。可以认为 core dump 是“内存快照”，但实际上，除了内存信息之外，还有些关键的程序运行状态也会同时 dump 下来，例如寄存器信息（包括程序指针、栈指针等）、内存管理信息、其他处理器和操作系统状

态和信息。core dump 对于编程人员诊断和调试程序是非常有帮助的，因为对于有些程序错误是很难重现的，例如指针异常，而 core dump 文件可以再现程序出错时的情景。

#### 9.2.6.2 开启 coredump

可以使用命令 ulimit 开启，也可以在程序中通过 setrlimit 系统调用开启。

在终端输入命令 `$ ulimit -c` 输出结果为 0，说明默认是关闭 coredump 的，即当程序异常终止时，不会生成 coredump 文件。可以使用 `$ ulimit -c unlimited` 来开启 coredump 功能，并且不限制 coredump 文件大小；如果需要限制文件大小，将 unlimited 改成你想生成 core 文件的大小，注意单位是 KB。

#### 9.2.6.3 调试 core 文件

调试使用 PC 上的 gdb 查看程序挂在位置。但调试开发板程序需要使用交叉编译工具链中的 gdb 命令

glibc:mips-linux-gnu-gdb

uclibc: mips-linux-uclibc-gnu-gdb

使用 mips-linux-***-gdb [program] [core] 查看 core 文件，其中 program 是可执行程序文件名，core 是生成的 core 文件名。例如：

```
$ mips-linux-uclibc-gnu-gdb ./build core-build
```

### 9.2.7 gdbserver 使用方法

gdbserver 是嵌入式开发调试的主要工具，依赖开发板上的 gdbserver 程序以及交叉编译工具链中的 mips-linux-gnu-gdb 命令。gdbserver 需要调试的 PC 和开发板网络相通。

本文仅介绍 gdbserver 的开启及连接方法，相关命令遵循标准 gdbserver 命令，详细内容请在互联网查询。

gdb 以及 gdbserver 命令位置：

A. gdb 在 toolchain 文件夹下，位于 bin/mips-linux-gnu-gdb

B. gdbserver 位于 toolchain 文件夹下：

glibc: mips-linux-gnu/libc/usr/lib/bin/gdbserver

uclibc: mips-linux-gnu/libc/uclibc/usr/lib/bin/gdbserver

1) PC 运行 gdb

```
$ mips-linux-gnu-gdb [PC 端应用程序路径]
```

例如：

```
$ mips-linux-gnu-gdb sample-Encoder-h264
GNU gdb (Ingenic 2015.02) 7.4.50.20120716-cvs
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type “show
copying”and “show warranty”for details.
This GDB was configured as “--host=i686-pc-linux-gnu --target=mips-
```

```
linux-gnu".
    For bug reporting instructions, please see:
    <http://www.gnu.org/software/gdb/bugs/>...
    Reading symbols from sample-Encoder-h264...done.
    (gdb)
```

注：

A. 如果出现<code>(no debugging symbols found)</code>的提示，可能是因为应用程序 strip 过了（PC 端），删除了 debug 需要的段，无法进行 gdb 调试

B. 开发板上程序是可以 strip 的，Symbol 的读取在 PC 端 gdb 工具完成。这也就是 gdbserver 相对与 local gdb 以及 core dump 的优势——开发板端可以 strip——因为前两者都需要在开发板端 load symbol，因此应用程序会变得非常大，无法放在 flash 中

2) 设置动态库搜索路径

应用程序往往动态链接 libc、ld 等库，当运行于动态库中 break 时（比如 memcpy）往往无法跟踪到当前 pc 所在位置。这是因为动态库的加载地址是不固定的。因此需要在 PC 端 gdb 上指定动态库加载路径。方法是：

```
    set solib-search-path /opt/mips-gcc472-glibc216-64bit/mips-linux-gnu/libc/lib/:/opt/mips-gcc472-glibc216-64bit/mips-linux-gnu/lib
```

多个路径之间通过“：”来隔开。注：上述例子中仅指定了 toolchain 中 glibc 的 libc 系列库和 libstdc++等主要外部库。

A. glibc 的动态库在 toolchain 文件夹下的相对目录为：mips-linux-gnu/libc/lib 与 mips-linux-gnu/lib

B. uclibc 的动态库在 toolchain 文件夹下的相对目录为：
mips-linux-gnu/libc/uclibc/lib 与 mips-linux-gnu/lib/uclibc

3) 开发板启动 gdbserver

```
    $ gdbserver [PC 机 IP:端口，与步骤 2 中端口需一致] [应用程序路径]
```

例如：

```
    $ gdbserver 192.168.1.100:1234 /tmp/sample-Encoder-h264
```

4) 链接开发板

```
    $ target remote [开发板 IP:端口]
```

例如：

```
    $ target remote 192.168.1.101:1234
```

5) gdbserver 连通，开发板会出现如下信息：

```
    (gdb) target remote 192.168.1.101:1234
    Remote debugging using 192.168.1.101:1234
    Reading symbols from /opt/mips-gcc472-glibc216-64bit/mips-linux-gnu/libc/lib/ld.so.1...done.
    Loaded symbols for /opt/mips-gcc472-glibc216-64bit/mips-linux-gnu/libc/lib/ld.so.1
    0x77fc7070 in __start () from /opt/mips-gcc472-glibc216-64bit/mips-linux-gnu/libc/lib/ld.so.1
```

至此，即可在 PC 端运行 gdb 命令。

注：与 gdbserver 与 local gdb 不同，启动程序的命令为“c”(continue)。

### 9.2.8 IMPSDK 调试命令

1) Rmem

rmem 的内存可以使用 SDK 中的 rmem 计算器 excel 表格来计算得到；SDK remem 内存使用信息可以使用下面命令查看：

```
impdbg --sys;logcat
```

2) 查看数据流处理统计

update_cnt 是每个 group 模块线程的处理次数统计 可以通过下面命令来获取 10 秒间隔的统计来计算处理帧率。

```
[root@IPC:tmp]# ./impdbg --sys;sleep 10;./impdbg --sys
tree item: 3
Framesource-0       update_cnt=81540(qframecnt=81538, dqframecnt=81540, sem_msg_cnt=16, sem_cnt=0)
OSD-0               update_cnt=81540(sem_msg_cnt=16, sem_cnt=0)
Encoder-0           update_cnt=81540(sem_msg_cnt=16, sem_cnt=0)
---Framesource-0-----OSD-0------------Encoder-0
tree item: 4
Framesource-1       update_cnt=108828(qframecnt=108823, dqframecnt=108823, sem_msg_cnt=16, sem_cnt=0)
IVS-0               update_cnt=108828(sem_msg_cnt=16, sem_cnt=0)
OSD-1               update_cnt=108828(sem_msg_cnt=16, sem_cnt=0)
Encoder-1           update_cnt=108828(sem_msg_cnt=16, sem_cnt=0)
---Framesource-1-----IVS-0------------OSD-1------------Encoder-1
```

3) 查看编码信息统计

可以通过下面命令获取编码 group，channel 信息，编码类型，码流控制等参数 tf 信息，编码通道码流 buf 成功被上次取走的码流统计 df 信息，编码通道码流 buf 慢导致的丢帧数统计。

```
[root@IPC:tmp]# ./impdbg --enc_info;sleep 10;./impdbg --enc_info
GROUP 0
        CHANNEL 0    1920x 1080     START H265 tf:69198      df:6050 encdur:3343131
    ------------------------------------------------
    ch->index = 0
    rcAttr->outFrmRate->frmRateNum = 25(0x19) offset:size = 0:4
    rcAttr->outFrmRate->frmRateDen = 1(0x1) offset:size = 4:4
    rcAttr->maxGop = 50(0x32) offset:size = 8:4
    rcAttr->rcMode->rcMode = 3(0x3) offset:size = 12:4
    rcAttr->rcMode->Smart->maxQp = 45(0x2d) offset:size = 16:4
    rcAttr->rcMode->Smart->minQp = 15(0xf) offset:size = 20:4
    rcAttr->rcMode->Smart->staticTime = 2(0x2) offset:size = 24:4
    rcAttr->rcMode->Smart->maxBitRate = 2048(0x800) offset:size = 28:4
    rcAttr->rcMode->Smart->iBiasLvl = -3(0xfffffffd) offset:size = 32:4
```

```
    rcAttr->rcMode->Smart->changePos = 80(0x50) offset:size = 36:4
    rcAttr->rcMode->Smart->qualityLvl = 2(0x2) offset:size = 40:4
    rcAttr->rcMode->Smart->frmQPStep = 3(0x3) offset:size = 44:4
    rcAttr->rcMode->Smart->gopQPStep = 15(0xf) offset:size = 48:4
    rcAttr->rcMode->Smart->flucLvl = 2(0x2) offset:size = 52:4
    rcAttr->frmUsed->enable = 0(0x0) offset:size = 56:1
    rcAttr->frmUsed->frmUsedMode = 0(0x0) offset:size = 60:4
    rcAttr->frmUsed->frmUsedTimes = 0(0x0) offset:size = 64:4
    rcAttr->denoise->enable = 0(0x0) offset:size = 68:1
    rcAttr->denoise->dnType = 0(0x0) offset:size = 72:4
    rcAttr->denoise->dnIQp = 0(0x0) offset:size = 76:4
    rcAttr->denoise->dnPQp = 0(0x0) offset:size = 80:4
    rcAttr->hSkip->hSkipAttr->skipType = 0(0x0) offset:size = 84:4
    rcAttr->hSkip->hSkipAttr->m = 49(0x31) offset:size = 88:4
    rcAttr->hSkip->hSkipAttr->n = 1(0x1) offset:size = 92:4
    rcAttr->hSkip->hSkipAttr->maxSameSceneCnt = 1(0x1) offset:size = 96:4
    rcAttr->hSkip->hSkipAttr->bEnableScenecut = 0(0x0) offset:size = 100:4
    rcAttr->hSkip->hSkipAttr->bBlackEnhance = 0(0x0) offset:size = 104:4
    rcAttr->hSkip->maxHSkipType = 1(0x1) offset:size = 108:4
    GROUP 1
           CHANNEL 1     640x  480     START H265 tf:83485      df:1506
encdur:3343132
    ------------------------------------------------
    ch->index = 1
    rcAttr->outFrmRate->frmRateNum = 25(0x19) offset:size = 0:4
    rcAttr->outFrmRate->frmRateDen = 1(0x1) offset:size = 4:4
    rcAttr->maxGop = 50(0x32) offset:size = 8:4
    rcAttr->rcMode->rcMode = 3(0x3) offset:size = 12:4
    rcAttr->rcMode->Smart->maxQp = 45(0x2d) offset:size = 16:4
    rcAttr->rcMode->Smart->minQp = 15(0xf) offset:size = 20:4
    rcAttr->rcMode->Smart->staticTime = 2(0x2) offset:size = 24:4
    rcAttr->rcMode->Smart->maxBitRate = 546(0x222) offset:size = 28:4
    rcAttr->rcMode->Smart->iBiasLvl = -3(0xfffffffd) offset:size = 32:4
    rcAttr->rcMode->Smart->changePos = 80(0x50) offset:size = 36:4
    rcAttr->rcMode->Smart->qualityLvl = 2(0x2) offset:size = 40:4
    rcAttr->rcMode->Smart->frmQPStep = 3(0x3) offset:size = 44:4
    rcAttr->rcMode->Smart->gopQPStep = 15(0xf) offset:size = 48:4
    rcAttr->rcMode->Smart->flucLvl = 2(0x2) offset:size = 52:4
    rcAttr->frmUsed->enable = 0(0x0) offset:size = 56:1
    rcAttr->frmUsed->frmUsedMode = 0(0x0) offset:size = 60:4
    rcAttr->frmUsed->frmUsedTimes = 0(0x0) offset:size = 64:4
    rcAttr->denoise->enable = 0(0x0) offset:size = 68:1
```

```
    rcAttr->denoise->dnType = 0(0x0) offset:size = 72:4
    rcAttr->denoise->dnIQp = 0(0x0) offset:size = 76:4
    rcAttr->denoise->dnPQp = 0(0x0) offset:size = 80:4
    rcAttr->hSkip->hSkipAttr->skipType = 0(0x0) offset:size = 84:4
    rcAttr->hSkip->hSkipAttr->m = 49(0x31) offset:size = 88:4
    rcAttr->hSkip->hSkipAttr->n = 1(0x1) offset:size = 92:4
    rcAttr->hSkip->hSkipAttr->maxSameSceneCnt = 1(0x1) offset:size = 96:4
    rcAttr->hSkip->hSkipAttr->bEnableScenecut = 0(0x0) offset:size = 100:4
    rcAttr->hSkip->hSkipAttr->bBlackEnhance = 0(0x0) offset:size = 104:4
    rcAttr->hSkip->maxHSkipType = 1(0x1) offset:size = 108:4

    GROUP 0
           CHANNEL 0    1920x 1080     START H265 tf:69319      df:6050
encdur:3353252
    ------------------------------------------------
    ch->index = 0
    rcAttr->outFrmRate->frmRateNum = 25(0x19) offset:size = 0:4
    rcAttr->outFrmRate->frmRateDen = 1(0x1) offset:size = 4:4
    rcAttr->maxGop = 50(0x32) offset:size = 8:4
    rcAttr->rcMode->rcMode = 3(0x3) offset:size = 12:4
    rcAttr->rcMode->Smart->maxQp = 45(0x2d) offset:size = 16:4
    rcAttr->rcMode->Smart->minQp = 15(0xf) offset:size = 20:4
    rcAttr->rcMode->Smart->staticTime = 2(0x2) offset:size = 24:4
    rcAttr->rcMode->Smart->maxBitRate = 2048(0x800) offset:size = 28:4
    rcAttr->rcMode->Smart->iBiasLvl = -3(0xfffffffd) offset:size = 32:4
    rcAttr->rcMode->Smart->changePos = 80(0x50) offset:size = 36:4
    rcAttr->rcMode->Smart->qualityLvl = 2(0x2) offset:size = 40:4
    rcAttr->rcMode->Smart->frmQPStep = 3(0x3) offset:size = 44:4
    rcAttr->rcMode->Smart->gopQPStep = 15(0xf) offset:size = 48:4
    rcAttr->rcMode->Smart->flucLvl = 2(0x2) offset:size = 52:4
    rcAttr->frmUsed->enable = 0(0x0) offset:size = 56:1
    rcAttr->frmUsed->frmUsedMode = 0(0x0) offset:size = 60:4
    rcAttr->frmUsed->frmUsedTimes = 0(0x0) offset:size = 64:4
    rcAttr->denoise->enable = 0(0x0) offset:size = 68:1
    rcAttr->denoise->dnType = 0(0x0) offset:size = 72:4
    rcAttr->denoise->dnIQp = 0(0x0) offset:size = 76:4
    rcAttr->denoise->dnPQp = 0(0x0) offset:size = 80:4
    rcAttr->hSkip->hSkipAttr->skipType = 0(0x0) offset:size = 84:4
    rcAttr->hSkip->hSkipAttr->m = 49(0x31) offset:size = 88:4
    rcAttr->hSkip->hSkipAttr->n = 1(0x1) offset:size = 92:4
    rcAttr->hSkip->hSkipAttr->maxSameSceneCnt = 1(0x1) offset:size = 96:4
    rcAttr->hSkip->hSkipAttr->bEnableScenecut = 0(0x0) offset:size = 100:4
```

```
    rcAttr->hSkip->hSkipAttr->bBlackEnhance = 0(0x0) offset:size = 104:4
    rcAttr->hSkip->maxHSkipType = 1(0x1) offset:size = 108:4
    GROUP 1
           CHANNEL 1     640x  480     START H265 tf:83738      df:1506
encdur:3353253
    ------------------------------------------------
    ch->index = 1
    rcAttr->outFrmRate->frmRateNum = 25(0x19) offset:size = 0:4
    rcAttr->outFrmRate->frmRateDen = 1(0x1) offset:size = 4:4
    rcAttr->maxGop = 50(0x32) offset:size = 8:4
    rcAttr->rcMode->rcMode = 3(0x3) offset:size = 12:4
    rcAttr->rcMode->Smart->maxQp = 45(0x2d) offset:size = 16:4
    rcAttr->rcMode->Smart->minQp = 15(0xf) offset:size = 20:4
    rcAttr->rcMode->Smart->staticTime = 2(0x2) offset:size = 24:4
    rcAttr->rcMode->Smart->maxBitRate = 546(0x222) offset:size = 28:4
    rcAttr->rcMode->Smart->iBiasLvl = -3(0xfffffffd) offset:size = 32:4
    rcAttr->rcMode->Smart->changePos = 80(0x50) offset:size = 36:4
    rcAttr->rcMode->Smart->qualityLvl = 2(0x2) offset:size = 40:4
    rcAttr->rcMode->Smart->frmQPStep = 3(0x3) offset:size = 44:4
    rcAttr->rcMode->Smart->gopQPStep = 15(0xf) offset:size = 48:4
    rcAttr->rcMode->Smart->flucLvl = 2(0x2) offset:size = 52:4
    rcAttr->frmUsed->enable = 0(0x0) offset:size = 56:1
    rcAttr->frmUsed->frmUsedMode = 0(0x0) offset:size = 60:4
    rcAttr->frmUsed->frmUsedTimes = 0(0x0) offset:size = 64:4
    rcAttr->denoise->enable = 0(0x0) offset:size = 68:1
    rcAttr->denoise->dnType = 0(0x0) offset:size = 72:4
    rcAttr->denoise->dnIQp = 0(0x0) offset:size = 76:4
    rcAttr->denoise->dnPQp = 0(0x0) offset:size = 80:4
    rcAttr->hSkip->hSkipAttr->skipType = 0(0x0) offset:size = 84:4
    rcAttr->hSkip->hSkipAttr->m = 49(0x31) offset:size = 88:4
    rcAttr->hSkip->hSkipAttr->n = 1(0x1) offset:size = 92:4
    rcAttr->hSkip->hSkipAttr->maxSameSceneCnt = 1(0x1) offset:size = 96:4
    rcAttr->hSkip->hSkipAttr->bEnableScenecut = 0(0x0) offset:size = 100:4
    rcAttr->hSkip->hSkipAttr->bBlackEnhance = 0(0x0) offset:size = 104:4
    rcAttr->hSkip->maxHSkipType = 1(0x1) offset:size = 108:4
```

## 9.3 wpa_supplicant 使用方法

wpa_supplicant 及 wpa_cli 使用方法。

### 9.3.1 概述

wpa_supplicant 是一个连接、配置 WIFI 的工具。它主要包含两个程序：wpa_supplicant 与 wpa_cli。二者的关系就是 server 与 client 的关系。通常情况下，可以通过 wpa_cli 来进行 WIFI 的配置与连接，如果有特殊的需要，可以编写应用程序直接调用 wpa_supplicant 的接口直接开发。

本文主要讲述如何通过 wpa_cli 进行 WIFI 的配置与连接。

### 9.3.2 使用方法

1） 启动 wpa_supplicant 应用

```
$ wpa_supplicant -D nl80211 -i wlan0 -c /etc/wpa_supplicant.conf -B
```

注意：/etc/wpa_supplicant.conf 文件里，添加下面代码。

```
ctrl_interface=/var/run/wpa_supplicant
update_config=1
```

2）启动 wpa_cli 应用

```
$ wpa_cli -i wlan0 scan              搜索附近 wifi 网络
$ wpa_cli -i wlan0 scan_result       打印搜索 wifi 网络结果
$ wpa_cli -i wlan0 add_network       添加一个网络连接
如果要连接加密方式是：[WPA-PSK-CCMP+TKIP][WPA2-PSK-CCMP+TKIP][ESS] (wpa 加密)
wifi 名称是：wifi_name
wifi 密码是：wifi_psk
$ wpa_cli -i wlan0 set_network 0 ssid '"wifi_name"'
$ wpa_cli -i wlan0 set_network 0 psk '"wifi_psk"'
$ wpa_cli -i wlan0 enable_network 0

如果要连接加密方式是：[WEP][ESS] (wep 加密)
wifi 名称是：wifi_name
wifi 密码是：wifi_psk
$ wpa_cli -i wlan0 set_network 0 ssid '"wpa_name"'
$ wpa_cli -i wlan0 set_network 0 key_mgmt NONE
$ wpa_cli -i wlan0 set_network 0 wep_key0 '"wap_psk"'
$ wpa_cli -i wlan0 enable_network 0

如果要连接加密方式是：[ESS] (无加密)
wifi 名称是：wifi_name
$ wpa_cli -i wlan0 set_network 0 ssid '"wifi_name"'
$ wpa_cli -i wlan0 set_network 0 key_mgmt NONE
$ wpa_cli -i wlan0 enable_network 0
```

3） 分配 ip,netmask,gateway,dns

```
$ udhcpc -i wlan0
```

执行完毕，就可以连接网络了。

4） 保存连接

```
$ wpa_cli -i wlan0 save_config
```

5）断开连接

```
$ wpa_cli -i wlan0 disable_network 0
```

6）连接已有的连接

```
$ wpa_cli -i wlan0 list_network              列举所有保存的连接
$ wpa_cli -i wlan0 select_network 0          连接第 1 个保存的连接
$ wpa_cli -i wlan0 enable_network 0          使能第 1 个保存的连接
```

7）断开 wifi

```
$ ifconfig wlan0 down
$ killall udhcpc
$ killall wpa_supplicant
```

### 9.3.3 wpa_wifi_tool 使用方法

wpa_wifi_tool 是基于 wpa_supplicant 及 wpa_cli 的一个用于快速设置 wifi 的工具，方便调试时连接 wifi 使用。使用者无需运行步骤 2 中复杂的命令，即可实现 wifi 设置和连接等功能。

注：此工具仅作为调试工具使用，实际 wifi 功能开发推荐使用 wpa_cli 或者直接调用 wpa_supplicant 实现。

使用方法：

A. 运行 wpa_wifi_tool

B. 输入 help 进行命令查看

C. s 进行 SSID 扫描

D. c[n]进行 wifi 连接，连接时若为新的 SSID 则需输入密码，若为已保存的 SSID 则可以使用保存过的密码或者重新输入密码

E. e 退出工具

其他使用方法请参考 help

## 9.4 4G 代码参考

### 9.4.1 PPP 模式

A drivers/net/usb/qmi_wwan.c

```
diff --git a/drivers/net/usb/qmi_wwan.c b/drivers/net/usb/qmi_wwan.c
index 5645921..f8f0020 100644
--- a/drivers/net/usb/qmi_wwan.c
+++ b/drivers/net/usb/qmi_wwan.c
@@ -614,7 +614,7 @@ static const struct usb_device_id products[] = {
```

```
          {QMI_GOBI_DEVICE(0x05c6, 0x9225)},      /* Sony Gobi 2000 Modem
device (N0279, VU730) */
          {QMI_GOBI_DEVICE(0x05c6, 0x9245)},      /* Samsung Gobi 2000 Modem
device (VL176) */
          {QMI_GOBI_DEVICE(0x03f0, 0x251d)},      /* HP Gobi 2000 Modem
device (VP412) */
    -     {QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
    +//    {QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
          {QMI_GOBI_DEVICE(0x05c6, 0x9265)},      /* Asus Gobi 2000 Modem
device (VR305) */
          {QMI_GOBI_DEVICE(0x05c6, 0x9235)},      /* Top Global Gobi 2000
Modem device (VR306) */
          {QMI_GOBI_DEVICE(0x05c6, 0x9275)},      /* iRex Technologies Gobi
2000 Modem device (VR307) */
```

B drivers/usb/serial/option.c

```
   --- a/drivers/usb/serial/option.c
   +++ b/drivers/usb/serial/option.c
   @@ -530,6 +530,22 @@ static const struct option_blacklist_info
telit_le920_blacklist = {
    };

    static const struct usb_device_id option_ids[] = {
   +#if 1 //Added by Quectel
   +      { USB_DEVICE(0x05C6, 0x9090) }, /* Quectel UC15 */
   +      { USB_DEVICE(0x05C6, 0x9003) }, /* Quectel UC20 */
   +      { USB_DEVICE(0x2C7C, 0x0125) }, /* Quectel EC25 */
   +      { USB_DEVICE(0x2C7C, 0x0121) }, /* Quectel EC21 */
   +      { USB_DEVICE(0x05C6, 0x9215) }, /* Quectel EC20 */
   +      { USB_DEVICE(0x2C7C, 0x0191) }, /* Quectel EG91 */
   +      { USB_DEVICE(0x2C7C, 0x0195) }, /* Quectel EG95 */
   +      { USB_DEVICE(0x2C7C, 0x0306) }, /* Quectel EG06/EP06/EM06 */
   +      { USB_DEVICE(0x2C7C, 0x0296) }, /* Quectel BG96 */
   +      { USB_DEVICE(0x2C7C, 0x0435) }, /* Quectel AG35 */
   +      { USB_DEVICE(0x2C7C, 0x0512) }, /* Quectel EG12/EG18 */
   +      { USB_DEVICE(0x2C7C, 0x6026) }, /* Quectel EC200 */
   +      { USB_DEVICE(0x2C7C, 0x6120) }, /* Quectel UC200 */
   +      { USB_DEVICE(0x2C7C, 0x6000) }, /* Quectel EC200/UC200 */
   +#endif
          { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_COLT) },
```

```
          { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA) },
          { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA_LIGHT) },
   @@ -1377,6 +1393,9 @@ static struct usb_serial_driver
option_1port_device = {
    #ifdef CONFIG_PM
          .suspend           = usb_wwan_suspend,
          .resume            = usb_wwan_resume,
   +#if 1 //add by Quectel
   +      .reset_resume = usb_wwan_resume,
   +#endif
    #endif
    };

   @@ -1421,6 +1440,27 @@ static int option_probe(struct usb_serial
*serial,
                                &serial->interface->cur_altsetting->desc;
          struct usb_device_descriptor *dev_desc =
&serial->dev->descriptor;

   +#if 1 //Added by Quectel
   +      //Quectel UC20's interface 4 can be used as USB Network device
   +      if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) &&
serial->dev->descriptor.idProduct ==
   +                    cpu_to_le16(0x9003) &&
serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
   +              return -ENODEV;
   +      //Quectel EC20's interface 4 can be used as USB Network device
   +      if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) &&
serial->dev->descriptor.idProduct ==
   +                    cpu_to_le16(0x9215) &&
serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
   +              return -ENODEV;
   +      if (serial->dev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
   +              __u16 idProduct =
le16_to_cpu(serial->dev->descriptor.idProduct);
   +
   +              //Quectel EC200&UC200's interface 0 can be used as USB
Network device (ecm, rndis)
   +              if
(serial->interface->cur_altsetting->desc.bInterfaceClass != 0xFF)
   +                    return -ENODEV;
   +              //Quectel
```

```
EC25&EC21&EG91&EG95&EG06&EP06&EM06&BG96&AG35&EG12&EG18's interface 4 can be
used as USB network device (qmi,ecm,mbim)
    +              if ((idProduct != 0x6026 && idProduct != 0x6126) &&
serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
    +                      return -ENODEV;
    +      }
    +#endif
    +
           /* Never bind to the CD-Rom emulation interface */
           if (iface_desc->bInterfaceClass == 0x08)
                  return -ENODEV;
```

C drivers/usb/serial/qcserial.c

```
    --- a/drivers/usb/serial/qcserial.c
    +++ b/drivers/usb/serial/qcserial.c
    @@ -53,7 +53,7 @@ static const struct usb_device_id id_table[] = {
           {DEVICE_G1K(0x05c6, 0x9202)},   /* Generic Gobi Modem device */
           {DEVICE_G1K(0x05c6, 0x9203)},   /* Generic Gobi Modem device */
           {DEVICE_G1K(0x05c6, 0x9222)},   /* Generic Gobi Modem device */
    -      {DEVICE_G1K(0x05c6, 0x9008)},   /* Generic Gobi QDL device */
    +//     {DEVICE_G1K(0x05c6, 0x9008)},   /* Generic Gobi QDL device */
           {DEVICE_G1K(0x05c6, 0x9009)},   /* Generic Gobi Modem device */
           {DEVICE_G1K(0x05c6, 0x9201)},   /* Generic Gobi QDL device */
           {DEVICE_G1K(0x05c6, 0x9221)},   /* Generic Gobi QDL device */
```

D drivers/net/usb/qmi_wwan.c

```
    --- a/drivers/usb/serial/usb_wwan.c
    +++ b/drivers/usb/serial/usb_wwan.c
    @@ -456,8 +456,22 @@ static struct urb *usb_wwan_setup_urb(struct
usb_serial_port *port,

           /* Fill URB using supplied data. */
           usb_fill_bulk_urb(urb, serial->dev,
    -                        usb_sndbulkpipe(serial->dev, endpoint) | dir,
    -                        buf, len, callback, ctx);
    +                       usb_sndbulkpipe(serial->dev, endpoint) | dir,
    +                       buf, len, callback, ctx);
    +
    +#if 1 //Added by Quectel for Zero Packet
    +      if (dir == USB_DIR_OUT) {
    +             struct usb_device_descriptor *desc =
&serial->dev->descriptor;
```

```
    +               if (desc->idVendor == cpu_to_le16(0x05C6) &&
desc->idProduct == cpu_to_le16(0x9090))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +               if (desc->idVendor == cpu_to_le16(0x05C6) &&
desc->idProduct == cpu_to_le16(0x9003))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +               if (desc->idVendor == cpu_to_le16(0x05C6) &&
desc->idProduct == cpu_to_le16(0x9215))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +               if (desc->idVendor == cpu_to_le16(0x2C7C))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +       }
    +#endif

            return urb;
     }
```

### 9.4.2 Wwan 模式

A drivers/net/usb/qmi_wwan.c 添加如下代码

```
    --- a/drivers/net/usb/qmi_wwan.c
    +++ b/drivers/net/usb/qmi_wwan.c
    @@ -20,6 +20,72 @@
     #include <linux/usb/usbnet.h>
     #include <linux/usb/cdc-wdm.h>

    +#if 1 //Added by Quectel
    +#include <linux/etherdevice.h>
    +struct sk_buff *qmi_wwan_tx_fixup(struct usbnet *dev, struct sk_buff
*skb, gfp_t flags)
    +{
    +       if (dev->udev->descriptor.idVendor != cpu_to_le16(0x2C7C))
    +               return skb;
    +       //Skip Ethernet header from message
    +       if (dev->net->hard_header_len == 0)
    +               return skb;
    +       else
    +               skb_reset_mac_header(skb);
    +       if (skb_pull(skb, ETH_HLEN)) {
    +               return skb;
    +       } else {
```

```
    +               dev_err(&dev->intf->dev, "Packet Dropped ");
    +       }
    +       // Filter the packet out, release it
    +       dev_kfree_skb_any(skb);
    +       return NULL;
    +}
    +
    +#include <linux/version.h>
    +#if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))
    +static int qmi_wwan_rx_fixup(struct usbnet *dev, struct sk_buff *skb)
    +{
    +       __be16 proto;
    +       if (dev->udev->descriptor.idVendor != cpu_to_le16(0x2C7C))
    +               return 1;
    +       /* This check is no longer done by usbnet */
    +       if (skb->len < dev->net->hard_header_len)
    +               return 0;
    +       switch (skb->data[0] & 0xf0) {
    +               case 0x40:
    +                       proto = htons(ETH_P_IP);
    +                       break;
    +               case 0x60:
    +                       proto = htons(ETH_P_IPV6);
    +                       break;
    +               case 0x00:
    +                       if (is_multicast_ether_addr(skb->data))
    +                               return 1;
    +                       /* possibly bogus destination - rewrite just in
case */
    +                       skb_reset_mac_header(skb);
    +                       goto fix_dest;
    +               default:
    +                       /* pass along other packets without modifications
*/
    +                       return 1;
    +       }
    +       if (skb_headroom(skb) < ETH_HLEN)
    +               return 0;
    +       skb_push(skb, ETH_HLEN);
    +       skb_reset_mac_header(skb);
    +       eth_hdr(skb)->h_proto = proto;
    +       memset(eth_hdr(skb)->h_source, 0, ETH_ALEN);
```

```
    +fix_dest:
    +       memcpy(eth_hdr(skb)->h_dest, dev->net->dev_addr, ETH_ALEN);
    +       return 1;
    +}
    +/* very simplistic detection of IPv4 or IPv6 headers */
    +static bool possibly_iphdr(const char *data)
    +{
    +       return (data[0] & 0xd0) == 0x40;
    +}
    +#endif
    +#endif
    +
     /* This driver supports wwan (3G/LTE/?) devices using a vendor
      * specific management protocol called Qualcomm MSM Interface (QMI) -
      * in addition to the more common AT commands over serial interface
    @@ -332,6 +398,33 @@ next_desc:
                    dev->net->dev_addr[0] &= 0xbf;  /* clear "IP" bit */
            }
            dev->net->netdev_ops = &qmi_wwan_netdev_ops;
    +
    +#if 1 //Added by Quectel
    +       if (dev->udev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
    +               if (intf->cur_altsetting->desc.bInterfaceClass != 0xff) {
    +                       dev_dbg(&intf->dev, "Quectel module not qmi_wwan
mode! please check 'at+qcfg=\"usbnet\"'\n");
    +                       return -ENODEV;
    +               }
    +               dev_dbg(&intf->dev, "Quectel
EC25&EC21&EG91&EG95&EG06&EP06&EM06&EG12&EP12&EM12&EG16&EG18&BG96&AG35 work
on RawIP mode\n");
    +               dev->net->flags |= IFF_NOARP;
    +#if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))
    +               /* make MAC addr easily distinguishable from an IP header
*/
    +               if (possibly_iphdr(dev->net->dev_addr)) {
    +                       dev->net->dev_addr[0] |= 0x02; /* set local
assignment bit */
    +                       dev->net->dev_addr[0] &= 0xbf; /* clear "IP" bit
*/
    +               }
    +#endif
    +               usb_control_msg(
```

```
    +                               interface_to_usbdev(intf),
    +                               usb_sndctrlpipe(interface_to_usbdev(intf),
0),
    +                               0x22, //USB_CDC_REQ_SET_CONTROL_LINE_STATE
    +                               0x21, //USB_DIR_OUT | USB_TYPE_CLASS |
USB_RECIP_INTERFACE
    +                               1, //active CDC DTR
    +
intf->cur_altsetting->desc.bInterfaceNumber,
    +                               NULL, 0, 100);
    +       }
    +#endif
    +
     err:
           return status;
     }
    @@ -416,6 +509,10 @@ static const struct driver_info    qmi_wwan_info = {
           .unbind         = qmi_wwan_unbind,
           .manage_power   = qmi_wwan_manage_power,
           .rx_fixup       = qmi_wwan_rx_fixup,
    +#if 1 //Added by Quectel
    +       .tx_fixup = qmi_wwan_tx_fixup,
    +       .rx_fixup = qmi_wwan_rx_fixup,
    +#endif
     };

     #define HUAWEI_VENDOR_ID       0x12D1
    @@ -434,6 +531,31 @@ static const struct driver_info    qmi_wwan_info = {
           QMI_FIXED_INTF(vend, prod, 0)

     static const struct usb_device_id products[] = {
    +#if 1 //Added by Quectel
    +#ifndef QMI_FIXED_INTF
    +       /* map QMI/wwan function by a fixed interface number */
    +#define QMI_FIXED_INTF(vend, prod, num) \
    +       .match_flags = USB_DEVICE_ID_MATCH_DEVICE |
    +       USB_DEVICE_ID_MATCH_INT_INFO, \
    +               .idVendor = vend, \
    +               .idProduct = prod, \
    +               .bInterfaceClass = 0xff, \
    +               .bInterfaceSubClass = 0xff, \
    +               .bInterfaceProtocol = 0xff, \
```

```
    +             .driver_info = (unsigned long)&qmi_wwan_force_int##num,
    +#endif
    +       { QMI_FIXED_INTF(0x05C6, 0x9003, 4) }, /* Quectel UC20 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0125, 4) }, /* Quectel EC25 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0121, 4) }, /* Quectel EC21 */
    +       { QMI_FIXED_INTF(0x05C6, 0x9215, 4) }, /* Quectel EC20 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0191, 4) }, /* Quectel EG91 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0195, 4) }, /* Quectel EG95 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0306, 4) }, /* Quectel EG06/EP06/EM06
*/
    +       { QMI_FIXED_INTF(0x2C7C, 0x0512, 4) }, /* Quectel
EG12/EP12/EM12/EG16/EG18 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0296, 4) }, /* Quectel BG96 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0435, 4) }, /* Quectel AG35 */
    +#endif
    +
            /* 1. CDC ECM like devices match on the control interface */
            {       /* Huawei E392, E398 and possibly others sharing both
device id and more... */
                    USB_VENDOR_AND_INTERFACE_INFO(HUAWEI_VENDOR_ID,
USB_CLASS_VENDOR_SPEC, 1, 9),
    @@ -614,7 +736,7 @@ static const struct usb_device_id products[] = {
            {QMI_GOBI_DEVICE(0x05c6, 0x9225)},      /* Sony Gobi 2000 Modem
device (N0279, VU730) */
            {QMI_GOBI_DEVICE(0x05c6, 0x9245)},      /* Samsung Gobi 2000 Modem
device (VL176) */
            {QMI_GOBI_DEVICE(0x03f0, 0x251d)},      /* HP Gobi 2000 Modem
device (VP412) */
    -       {QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
    +//     {QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
            {QMI_GOBI_DEVICE(0x05c6, 0x9265)},      /* Asus Gobi 2000 Modem
device (VR305) */
            {QMI_GOBI_DEVICE(0x05c6, 0x9235)},      /* Top Global Gobi 2000
Modem device (VR306) */
            {QMI_GOBI_DEVICE(0x05c6, 0x9275)},      /* iRex Technologies Gobi
2000 Modem device (VR307) */
```

B drivers/net/usb/qmi_wwan.c

```
     --- a/drivers/usb/serial/option.c
    +++ b/drivers/usb/serial/option.c
```

```
   @@ -530,6 +530,19 @@ static const struct option_blacklist_info
telit_le920_blacklist = {
    };

    static const struct usb_device_id option_ids[] = {
   +#if 1 //Added by Quectel
   +	{ USB_DEVICE(0x05C6, 0x9090) }, /* Quectel UC15 */
   +	{ USB_DEVICE(0x05C6, 0x9003) }, /* Quectel UC20 */
   +	{ USB_DEVICE(0x2C7C, 0x0125) }, /* Quectel EC25 */
   +	{ USB_DEVICE(0x2C7C, 0x0121) }, /* Quectel EC21 */
   +	{ USB_DEVICE(0x05C6, 0x9215) }, /* Quectel EC20 */
   +	{ USB_DEVICE(0x2C7C, 0x0191) }, /* Quectel EG91 */
   +	{ USB_DEVICE(0x2C7C, 0x0195) }, /* Quectel EG95 */
   +	{ USB_DEVICE(0x2C7C, 0x0306) }, /* Quectel EG06/EP06/EM06 */
   +	{ USB_DEVICE(0x2C7C, 0x0512) }, /* Quectel
EG12/EP12/EM12/EG16/EG18 */
   +	{ USB_DEVICE(0x2C7C, 0x0296) }, /* Quectel BG96 */
   +	{ USB_DEVICE(0x2C7C, 0x0435) }, /* Quectel AG35 */
   +#endif
	{ USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_COLT) },
	{ USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA) },
	{ USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA_LIGHT) },
   @@ -1245,7 +1258,7 @@ static const struct usb_device_id option_ids[] = {
	{ USB_DEVICE(CINTERION_VENDOR_ID, CINTERION_PRODUCT_AHXX) },
	{ USB_DEVICE(CINTERION_VENDOR_ID, CINTERION_PRODUCT_PLXX),
		.driver_info = (kernel_ulong_t)&net_intf4_blacklist },
   -	{ USB_DEVICE(CINTERION_VENDOR_ID, CINTERION_PRODUCT_HC28_MDM) },
   +	{ USB_DEVICE(CINTERION_VENDOR_ID, CINTERION_PRODUCT_HC28_MDM) },
	{ USB_DEVICE(CINTERION_VENDOR_ID,
CINTERION_PRODUCT_HC28_MDMNET) },
	{ USB_DEVICE(SIEMENS_VENDOR_ID, CINTERION_PRODUCT_HC25_MDM) },
	{ USB_DEVICE(SIEMENS_VENDOR_ID, CINTERION_PRODUCT_HC25_MDMNET) },
   @@ -1377,6 +1390,9 @@ static struct usb_serial_driver
option_1port_device = {
    #ifdef CONFIG_PM
	.suspend           = usb_wwan_suspend,
	.resume            = usb_wwan_resume,
   +#if 1 //Added by Quectel
   +	.reset_resume = usb_wwan_resume,
   +#endif
    #endif
    };
```

```
   @@ -1444,6 +1460,22 @@ static int option_probe(struct usb_serial
*serial,
              iface_desc->bInterfaceClass != USB_CLASS_CDC_DATA)
                  return -ENODEV;

   +#if 1 //Added by Quectel
   +      //Quectel UC20's interface 4 can be used as USB network device
   +      if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6)
   +                      && serial->dev->descriptor.idProduct ==
cpu_to_le16(0x9003)
   +                      &&
serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
   +              return -ENODEV;
   +      //Quectel EC20's interface 4 can be used as USB network device
   +      if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6)
   +                      && serial->dev->descriptor.idProduct ==
cpu_to_le16(0x9215)
   +                      &&
serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
   +              return -ENODEV;
   +      //Quectel EC25&EC21&EG91&EG95&EG06&EP06&EM06&BG96/AG35's
interface 4 can be used as USB network device
   +      if (serial->dev->descriptor.idVendor == cpu_to_le16(0x2C7C)
   +                      &&
serial->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
   +              return -ENODEV;
   +#endif
           /* Store device id so we can use it during attach. */
           usb_set_serial_data(serial, (void *)id);
```

C drivers/usb/serial/qcserial.c

```
   --- a/drivers/usb/serial/qcserial.c
   +++ b/drivers/usb/serial/qcserial.c
   @@ -78,7 +78,7 @@ static const struct usb_device_id id_table[] = {
           {USB_DEVICE(0x03f0, 0x241d)},   /* HP Gobi 2000 QDL device
(VP412) */
           {USB_DEVICE(0x03f0, 0x251d)},   /* HP Gobi 2000 Modem device
(VP412) */
           {USB_DEVICE(0x05c6, 0x9214)},   /* Acer Gobi 2000 QDL device
(VP413) */
   -       {USB_DEVICE(0x05c6, 0x9215)},   /* Acer Gobi 2000 Modem device
```

```
(VP413) */
    +//     {USB_DEVICE(0x05c6, 0x9215)},   /* Acer Gobi 2000 Modem device
(VP413) */
           {USB_DEVICE(0x05c6, 0x9264)},   /* Asus Gobi 2000 QDL device
(VR305) */
           {USB_DEVICE(0x05c6, 0x9265)},   /* Asus Gobi 2000 Modem device
(VR305) */
           {USB_DEVICE(0x05c6, 0x9234)},   /* Top Global Gobi 2000 QDL
device (VR306) */
```

D drivers/usb/serial/usb_wwan.c

```
    --- a/drivers/usb/serial/usb_wwan.c
    +++ b/drivers/usb/serial/usb_wwan.c
    @@ -459,6 +459,20 @@ static struct urb *usb_wwan_setup_urb(struct
usb_serial_port *port,
                             usb_sndbulkpipe(serial->dev, endpoint) | dir,
                             buf, len, callback, ctx);

    +#if 1 //Added by Quectel for zero packet
    +       if (dir == USB_DIR_OUT) {
    +               struct usb_device_descriptor *desc =
&serial->dev->descriptor;
    +               if (desc->idVendor == cpu_to_le16(0x05C6) &&
desc->idProduct == cpu_to_le16(0x9090))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +               if (desc->idVendor == cpu_to_le16(0x05C6) &&
desc->idProduct == cpu_to_le16(0x9003))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +               if (desc->idVendor == cpu_to_le16(0x05C6) &&
desc->idProduct == cpu_to_le16(0x9215))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +               if (desc->idVendor == cpu_to_le16(0x2C7C))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +       }
    +#endif
    +
            return urb;
    }
```

# 9.5 Carrier 使用说明

## 9.5.1 简介

Carrier 是 Ingenic 开发的一个实时图像浏览工具，可用来在线更改图像效果及编码参数等 Carrier 之称取自 Star Craft ，为航母之意。Carrier 由两部分构成，PC 端的应用程序 Carrier 以及 device 端的 carrier-server。Client 和 Server 的通信采取单独的方式进行通信，不兼容 Onvif 等 NVR 协议。

Features :

A. 双路码流控制

B. 抓图以及录像

C. 图像效果调节: Sensor 帧率设置，图像参数设置

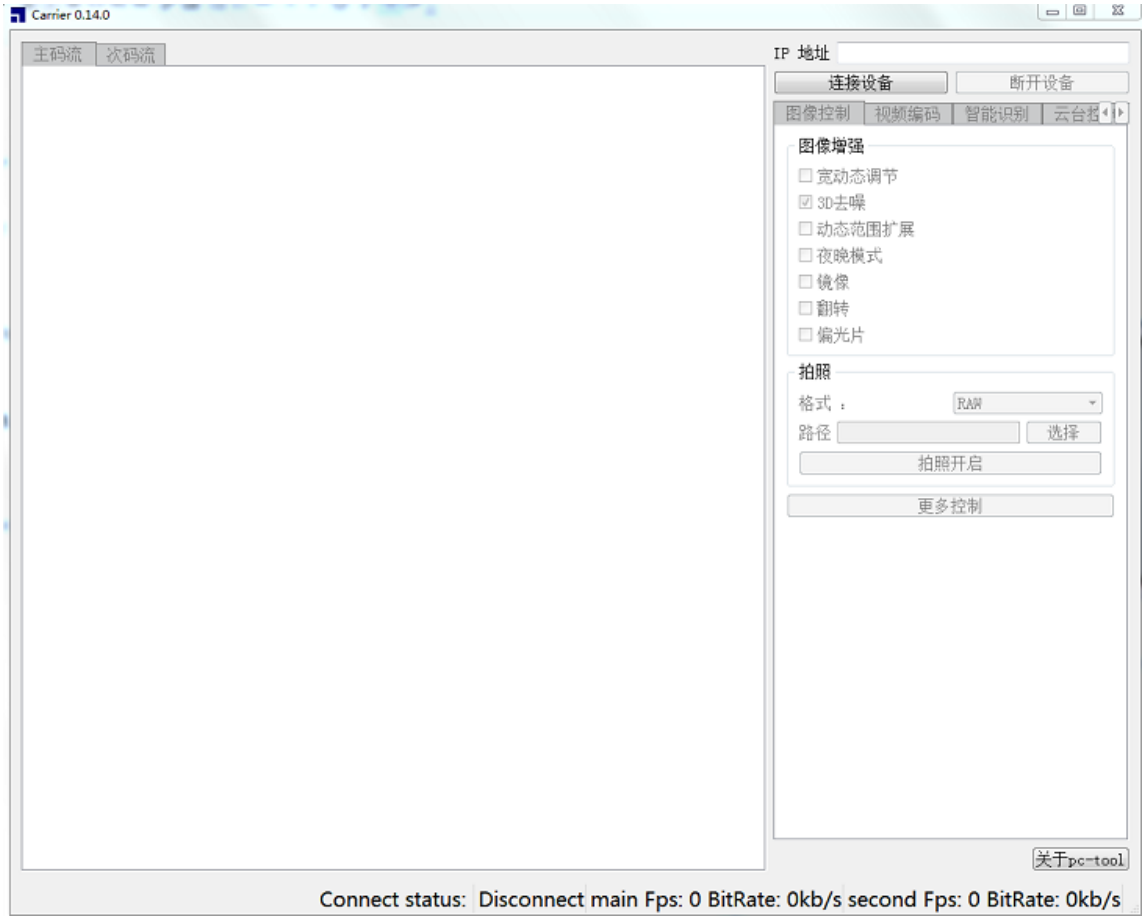
D. 编码参数调节: CBR 、 VBR 和 FixQP 的所有编码参数

E. IVS :人脸人形混合检测、人脸识别、移动检测、人形检测、越线检测、周界防范、云台控制

## 9.5.2 使用方法

1) 进入 Carrier 目录，双击 Carrier.exe ，启动程序


打开之后，会看到主程序界面，如下




2) 连接开发板

进入开发板的串口命令行，依次执行以下命令:

```
~# ifconfig eth0 <ipaddr>
~# route add default gw 192.168.x.x
~# insmod /lib/modules/tx-isp.ko
~# insmod /lib/modules/sensor_xxx.ko
~# carrier-server --st=xxx
```

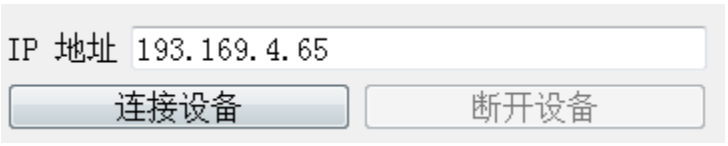
A. 配置 ip 地址，确定和运行 carrier 的 pc 可以通信

B. 配置 route ，以正确显示 RTSP 地址。若未设置 route，RTSP Url 可能显示不正确，但用户连接时输入正确的 IP 依然可以正常连接

C. 加载模块，需要加载 tx-isp.ko 以及 sensor_xxx.ko 。如果使用的摄像头是 ov9750，运行 `insmod /lib/modules/sensor_ov9750.ko` ；如果使用的摄像头是 ov9732，运行 `insmod /lib/modules/sensor_ov9732.ko`

D. 运行 carrier-sever 程序。如果使用的摄像头是 ov9750，运行 `carrier-server --st=ov9750` ；如果使用的摄像头是 ov9732，运行 `carrier-server-static --st=ov9732`

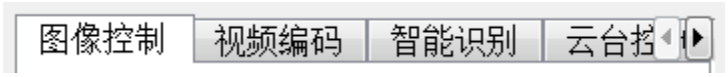
在确定开发板 carrier-server 进程开启之后，在 Carrier.exe 界面中,输入开发板的 IP 地址,点击连接设备，稍等片刻，即可看到实时视频。



3) 配置选项

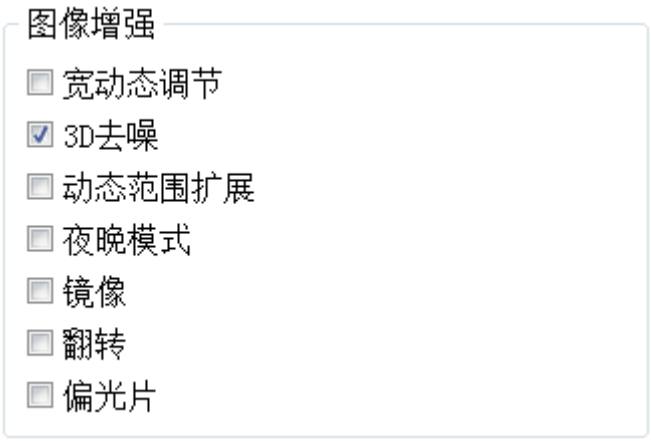
现在就可以设置开发板的各种配置。

A. 将菜单选中图像控制

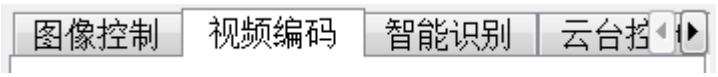



B. 可以通过下拉菜单，抓取 RAW，NV12，YUYV422，UYVY422 或者 RGB565 格式图片，抓图功能界面如下：


C. 图像增强功能界面如下：




D. 将菜单选中视频编码

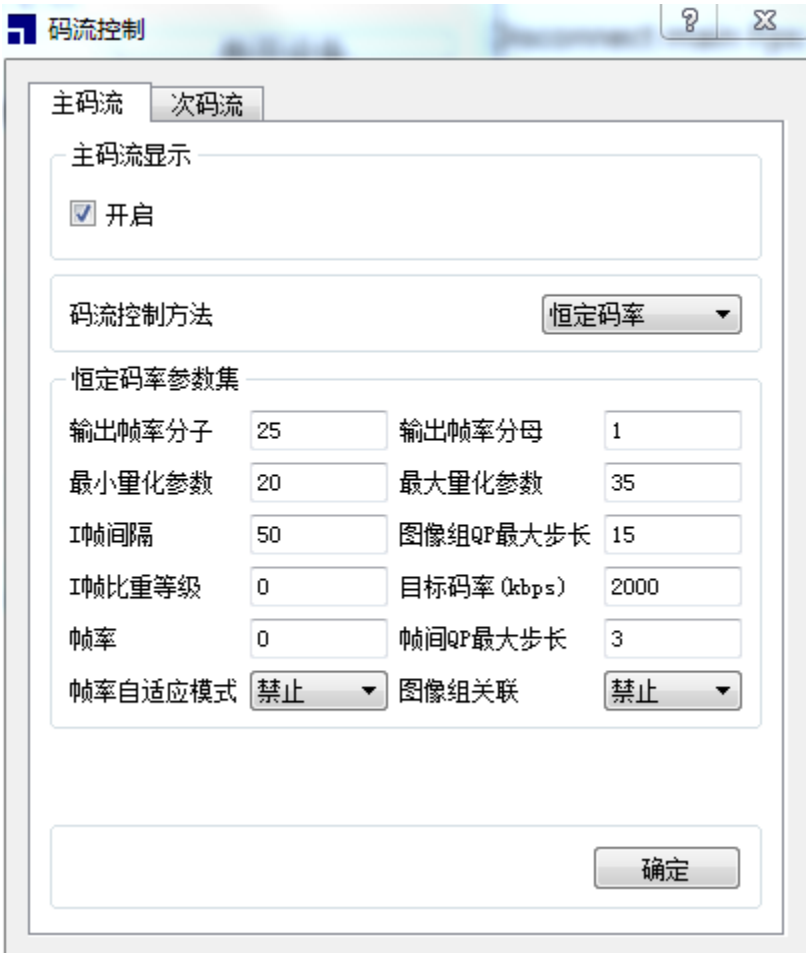



E. 录像功能界面如下：

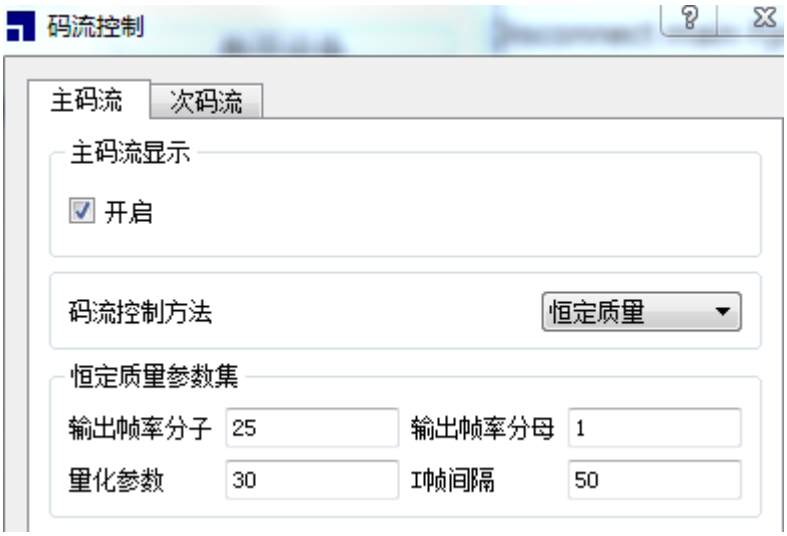
码流设置需要先点击更多控制:


F. 设置主码流的码流控制:

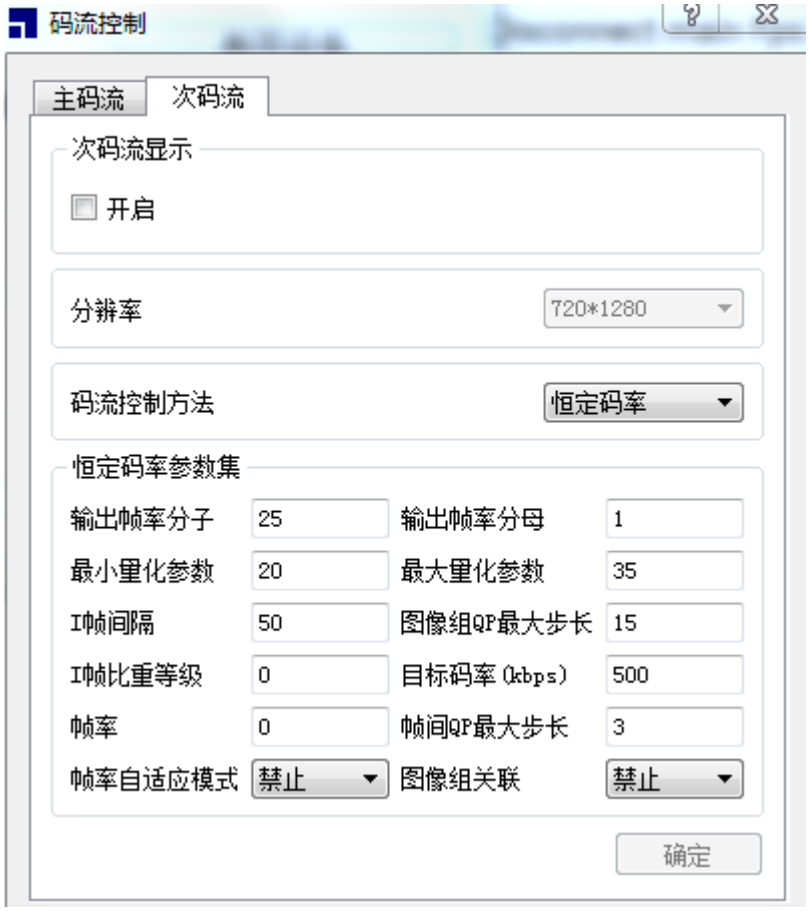
恒定码率控制:



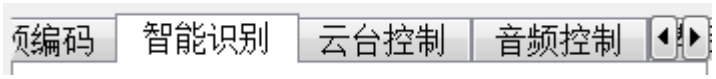

恒定质量控制:




G. 设置次码流的码流控制，比主码流多了分辨率可调接口:

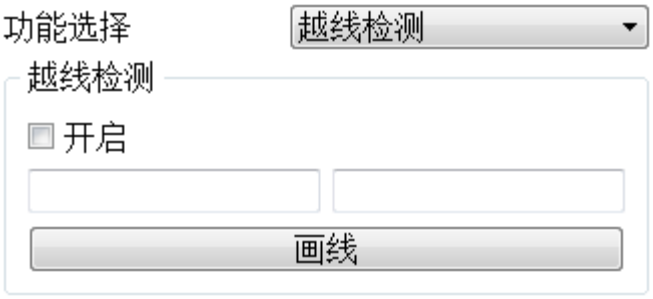



H. 将菜单选中智能识别




I. 开启越线检测功能。在视频显示界面，鼠标左键感兴趣的两点坐标，系统自动连

成一条直线。如果越线，就会报警。



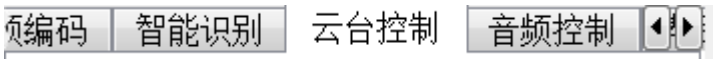

J. 开启人脸检测功能。自动检测人脸。


K. 开启移动检测功能。当检测到运动物体，就会报警。

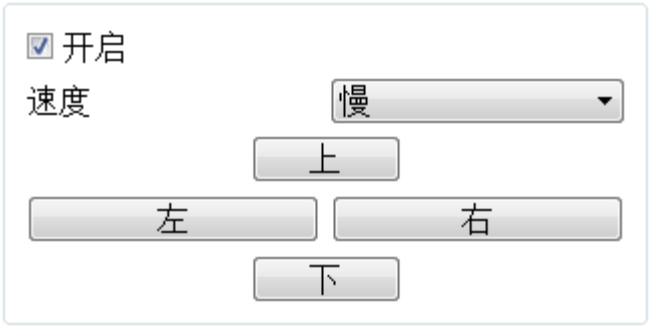

L. 开启人形检测功能。自动检测行人。


M. 开启周界防范功能。在视频显示界面，鼠标左键感兴趣的四点坐标，系统自动围成一个区域。如果进入该区域，就会报警。

N. 将菜单选中云台控制




O. 开启云台功能，设置速度和方向，云台就会旋转。




## 9.6 版本说明

ISVP_Devkit 版本升级说明

ISVP 版本升级（主要以软件资源及 SDK 升级为主）是一项持续的工作，新的版本会提供新的功能、修复问题、以及系统优化。

ISVP 软件系统主要包含：uboot，kernel，drivers，lib，sensor bin，rootfs，toochain 几个部分，对于每次 Devkit 更新，这几个部分间可能存在一定的依赖关系，需要同步升级，否则可能会出现错误。

版本升级有以下注意事项：

A. 开发人员需要认真阅读 ChangeLog，了解有哪些更新。每次的 ChangeLog 会注释“务必更新”和“推荐更新”的资源

B. 一般情况下建议 drivers(ISP, Sensor)、sensor bin、与 lib 同时更新

C. kernel 更新较少，但是如果需要更新 kernel，需要完全重新编译所有 ko，其中 ISP driver 和 Sensor driver 的编译顺序是先编译前者后编译后者

D. ISP driver 与 Sensor driver 及 Sensor bin 文件三者有匹配关系，不匹配可能造成图像异常

E. API 更新需要完全替换 lib 与头文件，头文件与 lib 不匹配可能造成程序崩溃

新的版本也可能会带来新的问题，ISVP 的开发团队会尽力保证新版本的正确性。若开发者发现了 Bug，请及时向君正的技术支持人员反馈，描述现象与复现情况等细节，ISVP 的开发团队会第一时间进行处理。

调试过程中需要问题，需要与技术支持人员进行沟通反馈。正确的提供 Log 以及记录问题，可以有效提高解决问题的效率。主要有以下几点：

A. 正确的描述现象，复现方法，复现概率

B. 提供 Log 以及截图

C. 提供 SDK 版本信息（或提供完整的 logcat）

D. 找到复现规律