



# **Melis RTOS**

## **系统开发指南**

**版本号: 0.1**  
**发布日期: 2020.02.27**

## 版本历史

版本号	日期	制/修订人	内容描述
0.1	2020.02.27	PDC-PSW	创建



## 目 录

<b>1 Melis3.0 用户开发手册</b>	<b>1</b>
1.1 系统架构 . . . . .	1
1.2 系统 IO 和内存映射 . . . . .	2
1.3 物理/虚拟地址转换 . . . . .	3
1.4 目录介绍 . . . . .	3
1.4.1 source 目录 . . . . .	3
1.4.1.1 ekernel 目录 . . . . .	3
1.5 编译与打包 . . . . .	4
1.6 编译环境命令介绍 . . . . .	5
<b>2 系统启动介绍</b>	<b>6</b>
2.1 Flash Layout . . . . .	6
2.2 启动流程图 . . . . .	7
<b>3 驱动开发介绍</b>	<b>8</b>
3.1 hal 层开发注意事项 . . . . .	8
3.2 src 层开发注意事项 . . . . .	8
3.3 驱动测试注意事项 . . . . .	9
3.4 SPINOR 驱动开发举例分析 . . . . .	9
3.4.1 Makefile 编写 . . . . .	9
3.4.1.1 编译选项添加 . . . . .	9
3.4.1.2 编译文件添加 . . . . .	10
3.4.1.3 添加依赖库 . . . . .	10
3.4.2 添加配置选项 . . . . .	10
3.4.3 hal 层开发 . . . . .	11
3.4.4 src 层开发 . . . . .	12
3.4.5 spinor 驱动的测试 . . . . .	12
<b>4 常用调试命令</b>	<b>14</b>
4.1 系统相关 . . . . .	14
4.1.1 help . . . . .	14
4.1.2 list_device . . . . .	14
4.1.3 list_timer . . . . .	14
4.1.4 list_msgqueue . . . . .	14
4.1.5 list_maibox . . . . .	14
4.1.6 list_mutex . . . . .	15
4.1.7 list_event . . . . .	15
4.1.8 list_sem . . . . .	15
4.1.9 list_thread . . . . .	15
4.2 调试相关 . . . . .	15
4.2.1 top . . . . .	15
4.2.2 mmlk . . . . .	15

4.2.3 p . . . . .	16
4.2.4 m . . . . .	16
4.2.5 fork . . . . .	16
4.2.6 backtrace . . . . .	16
4.3 烧录相关 . . . . .	16
4.3.1 reboot . . . . .	16
4.4 文件系统 IO 相关 . . . . .	17
4.4.1 echo . . . . .	17
4.4.2 cd . . . . .	17
4.4.3 pwd . . . . .	17
4.4.4 ls . . . . .	17
4.4.5 rm . . . . .	17
4.4.6 mkdir . . . . .	17
4.4.7 cat . . . . .	18
4.4.8 hexdump . . . . .	18
4.4.9 grep . . . . .	18
4.4.10 touch . . . . .	18
4.4.11 tail . . . . .	18
4.5 adb 使用 . . . . .	18
4.5.1 启动小机端 adbd 服务 . . . . .	18
4.5.2 查看设备列表 . . . . .	19
4.5.3 adb 终端登录 . . . . .	19
4.6 使用 Neon . . . . .	19
4.7 网络测试 . . . . .	20
4.7.1 配置 . . . . .	20
4.7.1.1 驱动配置 . . . . .	20
4.7.1.2 协议配置 . . . . .	20
4.7.1.3 应用配置 . . . . .	21
4.7.1.4 基础通信测试配置 . . . . .	21
4.7.2 测试 . . . . .	21
4.7.2.1 网络回环测试 . . . . .	21
4.7.2.2 socket 基础测试 . . . . .	22
4.7.2.3 网络基础功能测试 . . . . .	22
4.7.2.4 网络应用测试 . . . . .	23
4.8 文件系统测试 . . . . .	24
4.8.1 插入 SD 卡，自动挂载文件系统 . . . . .	24
4.8.2 读写压测 . . . . .	24
4.9 G2D 测试用例 . . . . .	25
<b>5 卡刷机/升级/量产功能支持</b>	<b>27</b>
<b>6 IPC 测试用例</b>	<b>28</b>
6.1 ISP 数据采集/处理用例 . . . . .	28

6.2 抓图测试用例 . . . . .	30
6.3 抓视频存卡用例 . . . . .	30



## 插 图

1-1 硬件框架图 . . . . .	1
1-2 系统 IO 框架 . . . . .	2
1-3 物理/虚拟地址转换示意图 . . . . .	3
2-1 启动流程图 . . . . .	7
3-1 menuconfig 配置图 . . . . .	11
4-1 adbd 命令运行效果图 . . . . .	19
4-2 adb devices 命令运行效果图 . . . . .	19
4-3 adb shell 命令运行效果图 . . . . .	19
4-4 函数代码用法图 . . . . .	20
4-5 loop_server 命令运行效果图 . . . . .	21
4-6 loop_client 命令运行效果图 . . . . .	22
4-7 插入 SD 卡运行效果图 . . . . .	24
4-8 rwcheck -d /sdmmc 命令运行效果图 . . . . .	25
4-9 g2d_test 命令运行效果图 . . . . .	26
6-1 ISP 数据采集命令运行效果图 . . . . .	29

# 1 Melis3.0 用户开发手册

## 1.1 系统架构

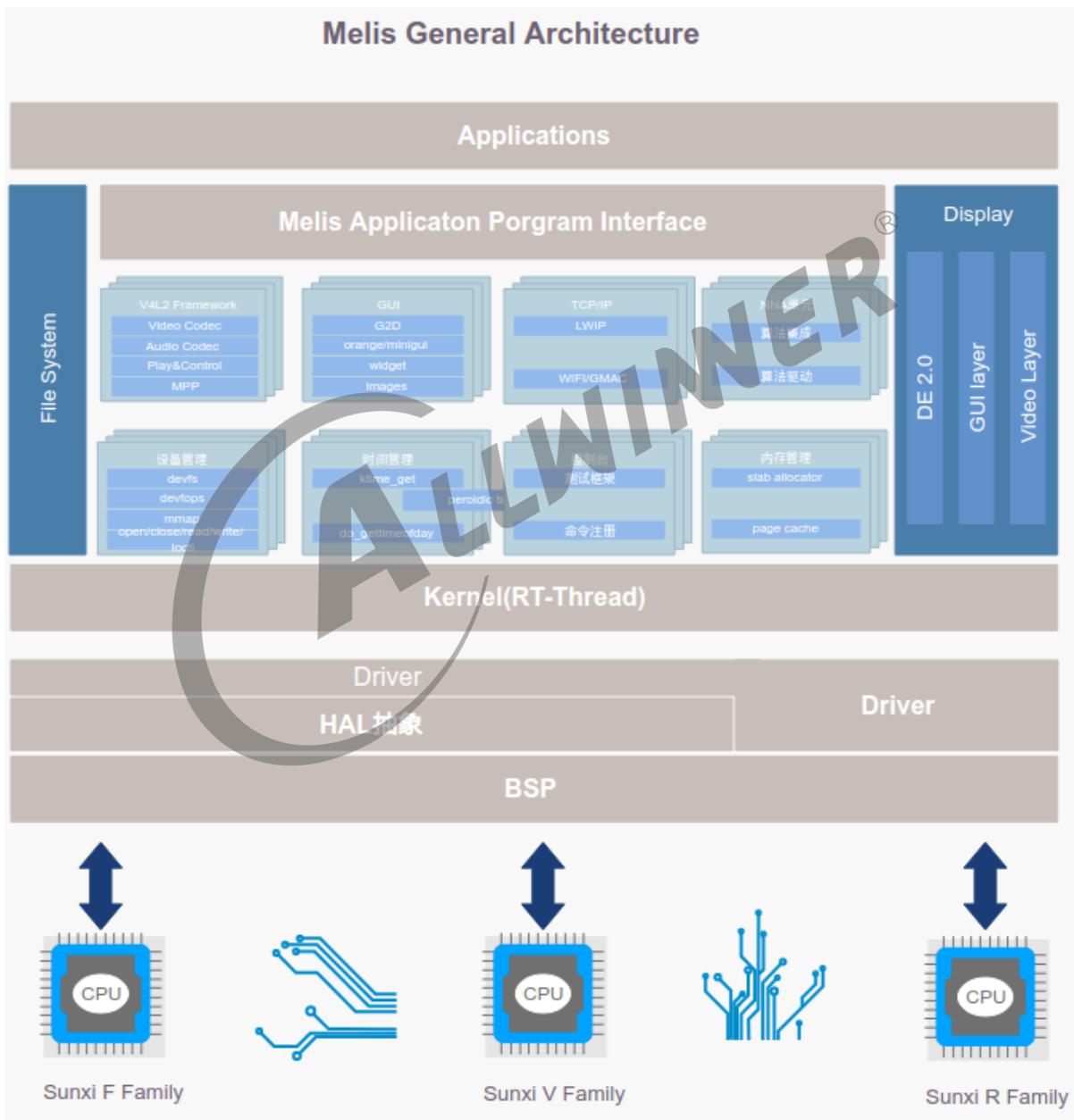


图 1-1: 硬件框架图

## 1.2 系统 IO 和内存映射

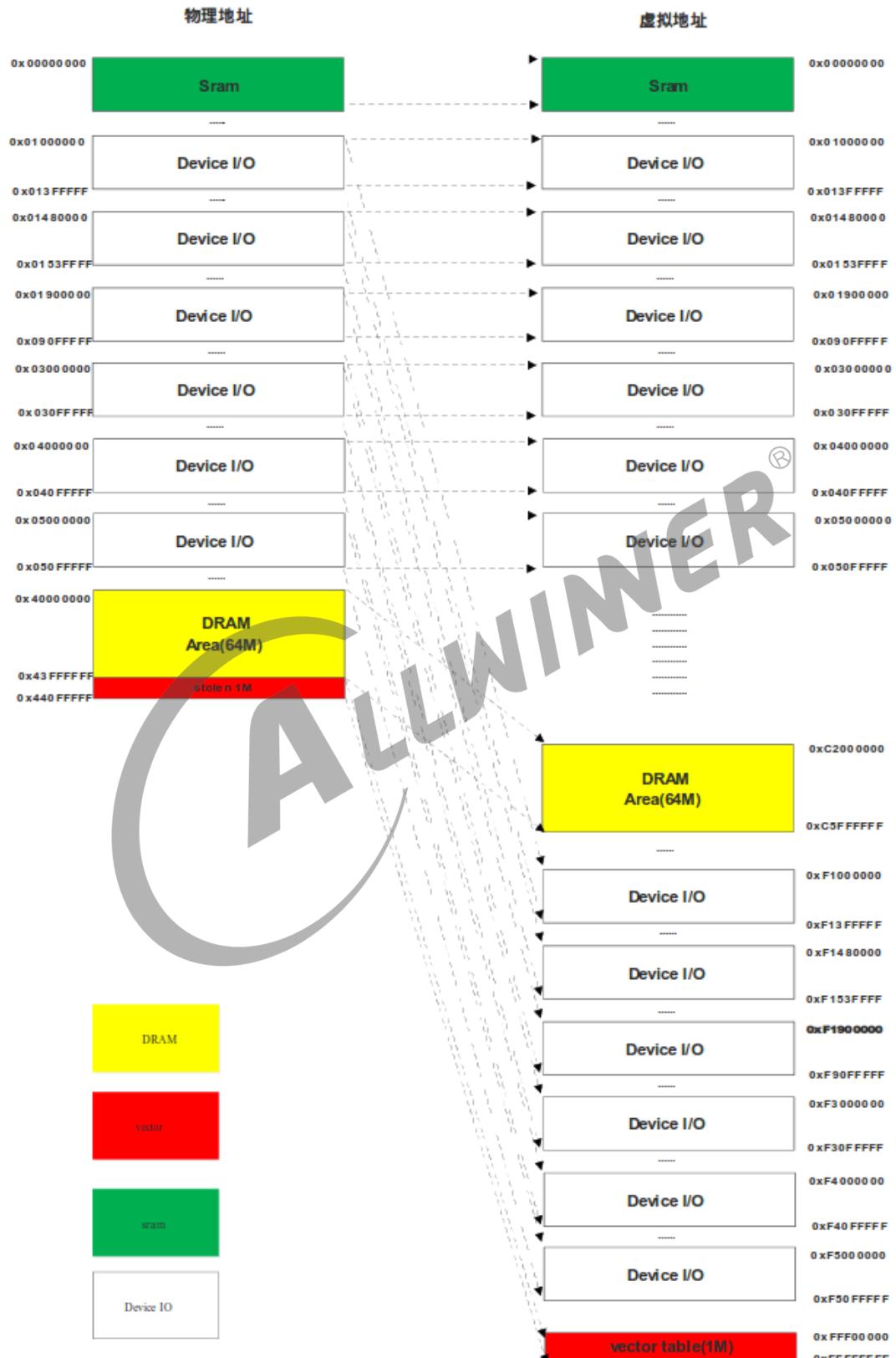


图 1-2: 系统 IO 框架

## 1.3 物理/虚拟地址转换

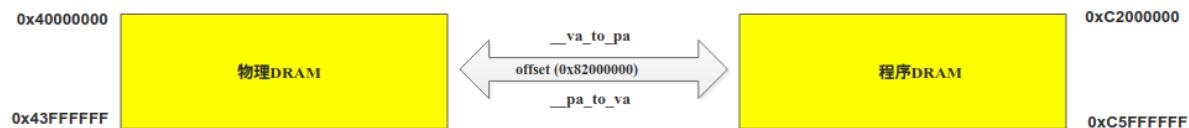


图 1-3: 物理/虚拟地址转换示意图

## 1.4 目录介绍

### SDK 根目录

```
.
├── cache.md
├── CODING-STYLE.md
├── document      文档目录
├── README.md
├── source        源码目录
├── toolchain     解压后工具链目录
└── tools         工具链目录
```

### 1.4.1 source 目录

```
.
├── ekernel
├── elibrary
├── include
├── Makefile
├── melis-env.sh      内核和与内核链接的驱动目录
                         模块使用的系统调用库目录
├── projects
├── scripts          环境设置脚本
├── utility          编译系统脚本
└── workspace        工具目录
                     镜像制作目录
```

#### 1.4.1.1 ekernel 目录

```
.
├── arch          芯片架构相关目录
├── core          系统内核目录
├── drivers       与内核链接的驱动开发目录
├── filesystem
├── Kconfig
├── Makefile
└── subsys        与内核链接的中间件目录
```

## 1.5 编译与打包

1. 进入 source 目录，执行下列命令，添加编译环境变量

```
cd source  
source melis-env.sh
```

2. 在 source 目录下，执行命令，选择方案

```
lunch
```

输入方案数字 4，选择方案 v833-perf1

3. 在 source 目录下，执行命令配置系统，可跳过（使用默认配置）<sup>®</sup>

```
make menuconfig
```

4. 在 source 目录，执行命令编译

```
make
```

如需多任务编译，可执行

```
make -j4
```

5. 在任意目录下，执行打包命令

```
pack
```

生成的烧录镜像位于 source/workspace/suniv/beetles/ePDKv100.img

6. 烧录

windows 系统可使用 phoenixSuit 工具烧录；ubuntu 系统可使用 LiveSuit 工具烧录

7. 在 source 目录下，执行命令，清除编译结果

```
make clean  
make distclean
```

注意：执行 distclean 之后，需要重新 lunch，选择方案

## 1.6 编译环境命令介绍

表 1-1：编译环境命令列表

命令	描述
ctop	跳转到 SDK 根目录
croot	跳转到 source 目录
crootfs	跳转到 workspace/suniv/rootfs 目录
cramfs	跳转到 workspace/suniv/ramfs 目录
ckernel	跳转到 source/ekernel 目录
cconfigs	跳转到方案配置文件目录
cwork	跳转到镜像生成目录
godir	跳转到指定目录
format	格式化当前目录代码命令
formatall	格式化所有目录代码命令

## 2 系统启动介绍

### 2.1 Flash Layout

表 2-1: 分区偏移列表

偏移 (KB)	分区
0 - 48	boot0
48 - 64	gpt 分区表
64 - 3392	melis 系统镜像
3392 - 18624	rootfs 分区

## 2.2 启动流程图

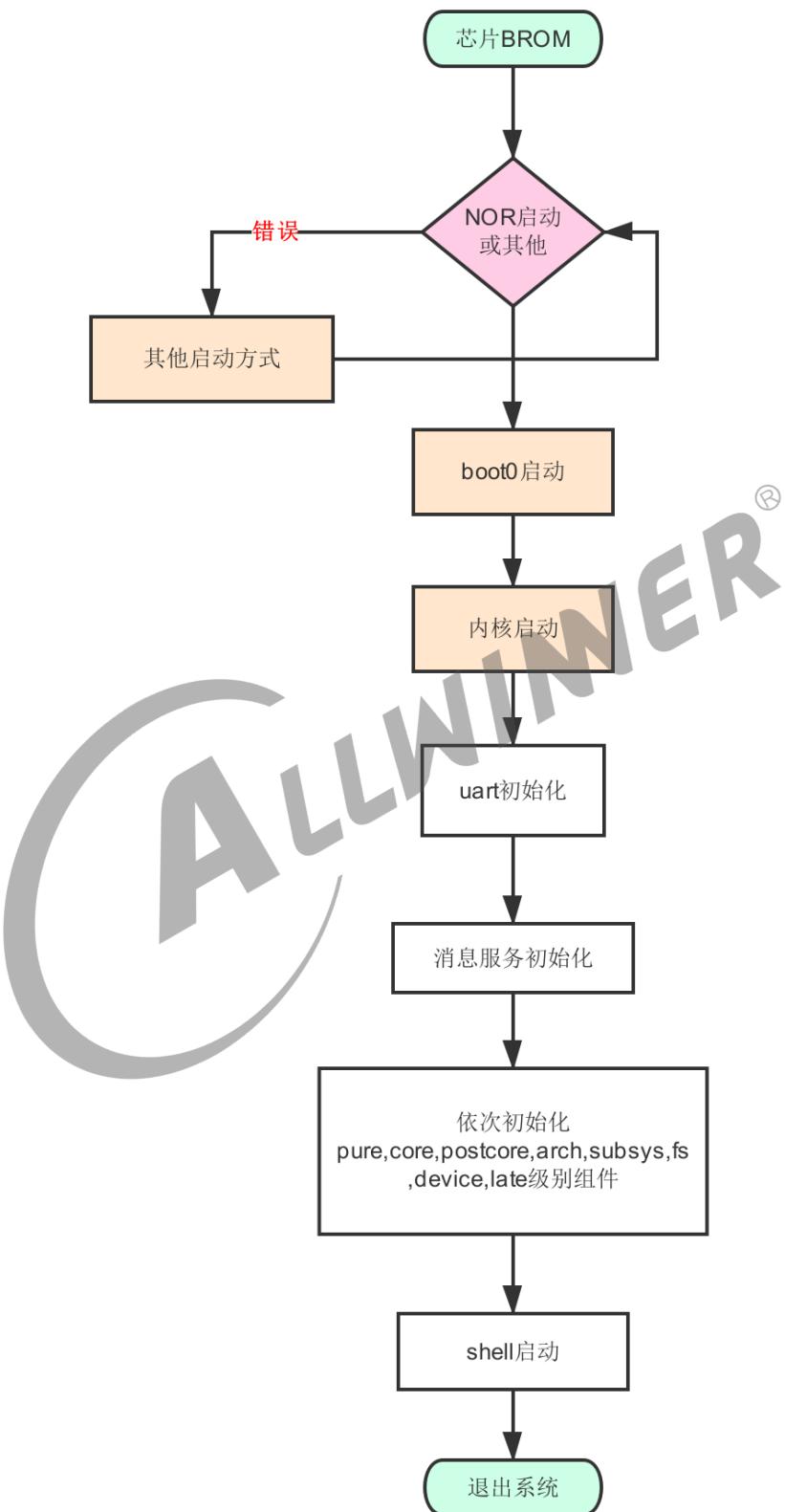


图 2-1: 启动流程图

## 3 驱动开发介绍

驱动开发主要分为两层：

- hal 层，主要用于屏蔽硬件细节，尽量做到与操作系统无关；
- src 层，主要是对 hal 层的封装，包括注册设备节点，提供给应用开发者使用；

### 3.1 hal 层开发注意事项

1. hal 层代码集中在 source/ekernel/driver/hal/source 目录下，需要提供驱动操作函数集供 src 层或其他驱动的 hal 层使用；
2. hal 层头文件位于 source/ekernel/driver/include/hal 目录下；
3. 驱动如需使用创建任务、信号量、开关中断、开关调度等接口，请尽量使用 source/ekernel/drivers/hal/sourceosal 目录下的系统抽象层；下表是 hal 层系统接口的文件介绍。

表 3-1: 文件功能列表

文件名	功能
hal_atomic.c	中断操作、临界区操作
hal_queue.c	队列操作
hal_cache.c	cache 操作
hal_sem.c	信号量操作
hal_thread.c	任务操作
hal_timer.c	定时器操作

### 3.2 src 层开发注意事项

1. src 层代码集中在 source/ekernel/driver/src 目录下，封装设备操作函数集，注册设备节点，提供给开发者使用；
2. src 层头文件位于 source/ekernel/driver/include/src 目录下；
3. 通过使用 rt\_device\_register 接口注册设备，rt\_device\_find 等获取设备
4. 可以通过下表接口向系统注册启动驱动

表 3-2: 启动顺序表

接口	启动顺序
pure_install	1
core_install	2
postcore_install	3
arch_install	4
subsys_install	5
fs_install	6
rootfs_install	7
device_install	8
late_install	9

### 3.3 驱动测试注意事项

1. 驱动测试代码主要放在 source/ekernel/driver/test 目录下
2. 通过使用宏来向控制台添加测试命令

```
FINSH_FUNCTION_EXPORT_ALIAS(func_name, command_name, description);
```

func\_name 是命令入口地址; command\_name 格式是 \_\_cmd\_name, name 是命令名字; description 是命令描述, 该宏头文件是 rtthread.h

### 3.4 SPINOR 驱动开发举例分析

#### 3.4.1 Makefile 编写

##### 3.4.1.1 编译选项添加

```
#subdir-ccflags-y添加的gcc编译选项对子目录有效, srctree指向source目录
subdir-ccflags-y += -I$(srctree)/ekernel/drivers/include

# ccflags-y添加的gcc编译选项当前目录有效, obj指向当前目录
ccflags-y += -I$(obj)/ekernel/drivers/

# cppflags-y添加g++编译选项
cppflags-y += -I$(obj)/ekernel/drivers/
```

### 3.4.1.2 编译文件添加

```
# 添加编译文件
obj-y += hal_spinor.o

# 以hal/source/Makefile为例，添加编译文件目录
obj-y += spinor/
```

### 3.4.1.3 添加依赖库

在 source/ekernel/Makefile 里添加

```
# 为避免库循环依赖问题，请在--start-group和--end-group之间添加库
# 如依赖库libtest.a，则将libtest.a放在source/elibrary/bin下
# 修改Makefile为：
usrlibs-y += -L$(srctree)/${elibrary-libs} \
             | --start-group \
             -ltest \
             | --end-group
```

## 3.4.2 添加配置选项

配置选项的添加，与 Linux 类似，通过 Kconfig 来添加。以 source/ekernel/subsys/Kconfig 下为例，在其父目录 source/kernel/ 的 Kconfig 中，会加载子目录下的 Kconfig

```
source "ekernel/subsys/Kconfig"
```

subsys 目录下的 Kconfig 需要加载其子目录下的 Kconfig 和实现其所需要的配置

```
menu "Libc library"

choice
prompt "C library"
default LIBCNEWLIB

config LIBCNEWLIB
  bool "newlib"

config LIBCNONE
  bool "none"
endchoice

config EXTERN_LIBC
  bool
  default n if LIBCNONE
  default y if LIBCNEWLIB
endmenu

source "ekernel/subsys/samples/Kconfig"
```

```
source "ekernel/subsys/net/rt-thread/Kconfig"
endmenu
```

在 source 目录下执行 make menuconfig 选择配置，再执行 make，会把.config 转化成 autoconf.h 头文件，此时配置才会生效

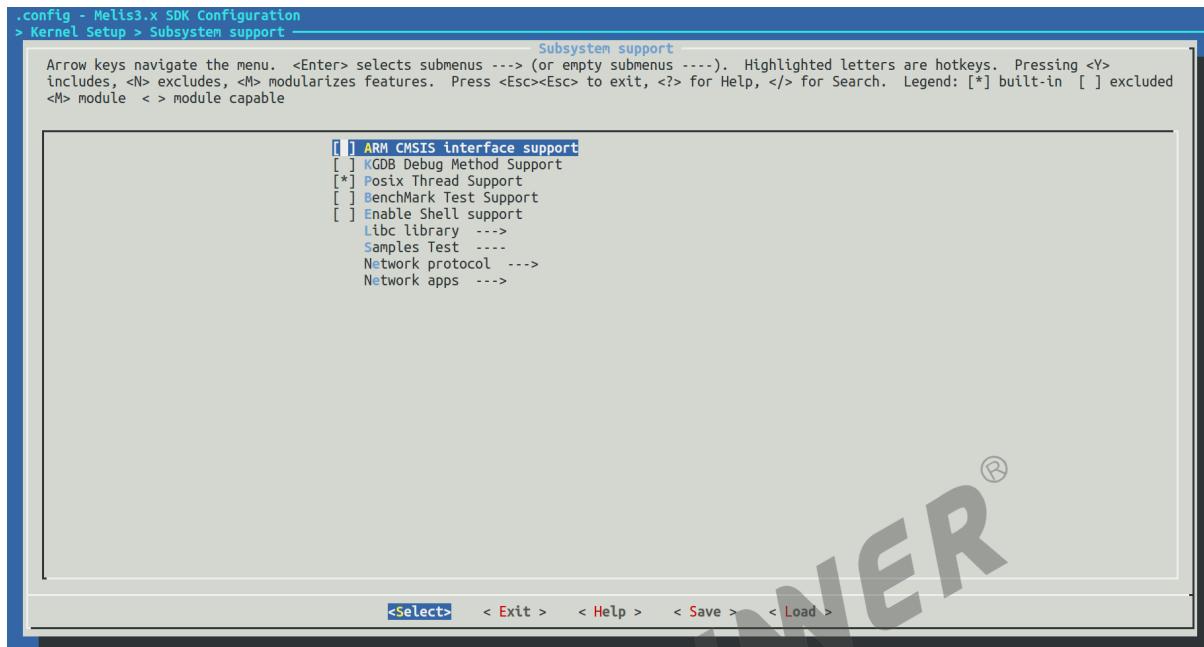


图 3-1: menuconfig 配置图

### 3.4.3 hal 层开发

source/ekernel/drivers/hal/source 目录下的 hal\_spinor.c 文件中，sunxi\_hal\_driver\_spinor\_t 是 spinor 驱动的处理函数集，是 spinor 驱动 hal 层向外暴露的接口；

```
const sunxi_hal_driver_spinor_t sunxi_hal_spinor_driver =
{
    .get_version = spinor_get_version,
    .get_capabilities = spinor_get_capabilities,
    .initialize = spinor_initialize,
    .uninitialize = spinor_uninitialize,
    .power_control = spinor_power_control,
    .read_data = spinor_read_data,
    .program_data = spinor_program_data,
    .erase_sector = spinor_erase_sector,
    .erase_chip = spinor_erase_chip,
    .get_status = spinor_get_status,
    .get_info = spinor_get_info,
    .signal_event = spinor_signal_event,
    .control = spinor_control,
};
```

spinor hal 层驱动使用的系统接口，如 hal\_sem，来源于 source/ekernel/drivers/hal-/source/osal/src/目录，其目的主要是为了屏蔽系统差异。

### 3.4.4 src 层开发

source/ekernel/drivers/src/spinor 目录下的 spinor\_drv.c 文件，封装了 spinor 驱动 src 层向应用开发者提供的接口，包括提供设备节点的操作函数集，注册设备节点等；

```
int sunxi_driver_spinor_init(void)
{
    int ret = -1;
    struct rt_device *device;

    device = malloc(sizeof(struct rt_device));
    if (!device)
    {
        return ret;
    }
    memset(device, 0, sizeof(struct rt_device));

    /* 添加设备操作函数集，注册设备节点 */
    device->init = sunxi_spinor_init;
    device->open = sunxi_spinor_open;
    device->close = sunxi_spinor_close;
    device->read = sunxi_spinor_read;
    device->write = sunxi_spinor_write;
    device->control = sunxi_spinor_control;
    device->user_data = (void *)&sunxi_hal_spinor_driver;

    ret = rt_device_register(device, "spinor", RT_DEVICE_FLAG_RDWR);
    if (ret != 0)
    {
        free(device);
        return ret;
    }

    /* 分区管理初始化 */
    int nor_blkpart_init(const char *name);
    ret = nor_blkpart_init("spinor");
    return ret;
}

device_initcall(sunxi_driver_spinor_init);
```

其中，使用device\_initcall来向系统注册启动，其头文件是 init.h

### 3.4.5 spinor 驱动的测试

spinor 驱动的测试用例，放在了 source/ekernel/drivers/test/test\_spinor.c 中，集成了 spinor\_test 测试命令，提供 spinor 读、擦除、写的测试。

```
static int cmd_spinor_test(int argc, const char **argv)
{
    int ret;
    /* 查找设备节点 */
    rt_device_t dev = rt_device_find("spinor");
```

```
if (!dev)
{
    printf("can not find device\n");
    ret = -1;
    goto out;
}

if (init_flag == 0)
{
    dev->init(dev);
    init_flag = 1;
}
char *buf = (char *)malloc(512);
if (buf == 0)
{
    printf("allocate memnory failed\n");
    ret = -1;
    goto out;
}
memset(buf, 0, 512);
/* 读数据 */
printf("read data:\n");
ret = dev->read(dev, 3 * 1024 * 1024, buf, 512);
if (ret)
{
    printf("spinor read data failed\n");
    ret = -1;
    goto error;
}
hexdump(buf, 512);
printf("erase data:\n");
/* 擦除 */
spinor_drv_erase_t erase_sector;
memset(&erase_sector, 0, sizeof(spinor_drv_erase_t));
erase_sector.addr = 3 * 1024 * 1024;
erase_sector.len = 4 * 1024;
ret = dev->control(dev, DEVICE_SPINOR_CMD_ERASE_SECTOR, &erase_sector);
if (ret)
{
    printf("spinor erase data failed\n");
    ret = -1;
    goto error;
}
/* 写数据 */
printf("write data:\n");
memset(buf, 0xa5, 512);
ret = dev->write(dev, 3 * 1024 * 1024, buf, 512);
if (ret)
{
    printf("spinor write data failed\n");
    ret = -1;
    goto error;
}

error:
    free(buf);
out:
    return ret;
}
FINSH_FUNCTION_EXPORT_ALIAS(cmd_spinor_test, __cmd_spinor_test, spinor_test);
```

## 4 常用调试命令

### 4.1 系统相关

#### 4.1.1 help

作用：列出当前提供的命令以及命令描述  
用法：help

#### 4.1.2 list\_device

作用：查看注册设备信息  
用法：list\_device

#### 4.1.3 list\_timer

作用：查看定时器信息  
用法：list\_timer

#### 4.1.4 list\_msgqueue

作用：查看消息队列信息  
用法：list\_msgqueue

#### 4.1.5 list\_maibox

作用：查看邮箱信息  
用法：list\_mailbox

## 4.1.6 list\_mutex

作用：查看互斥量信息  
用法：list\_mutex

## 4.1.7 list\_event

作用：查看事件标记组信息  
用法：list\_event

## 4.1.8 list\_sem

作用：查看信号量信息  
用法：list\_sem

## 4.1.9 list\_thread

作用：查看任务信息  
用法：list\_thread

## 4.2 调试相关

### 4.2.1 top

作用：查看系统性能信息  
用法：top

### 4.2.2 mmlk

作用：查看可疑内存泄露信息  
用法：第一次执行mmlk表示开启内存泄露检查，第二次执行mmlk表示结束内存泄露信息检查

### 4.2.3 p

作用：查看内存或寄存器的值  
用法：p mem\_addr [len]

### 4.2.4 m

作用：修改内存或寄存器的值  
用法：m mem\_addr value

### 4.2.5 fork

作用：创建一个任务来执行控制台命令  
用法：fork command\_name [arg1] [arg2] [...]

### 4.2.6 backtrace

作用：查看指定任务堆栈回溯信息  
用法：backtrace [taskname]

复制 backtrace 信息到 err.log 中，然后在 source 目录下执行

```
python utility/python/backtrace_parser.py err.log ekernel/melis30.elf
```

即可获得系统对应的 c 代码文件和行号

## 4.3 烧录相关

### 4.3.1 reboot

作用：重启系统或者进入系统烧录模式  
用法：reboot 重启系统；  
reboot efix 进入系统烧录模式

## 4.4 文件系统 IO 相关

### 4.4.1 echo

作用：添加字符串到文件  
用法：`echo "test"` 输出字符串到控制台  
`echo "test" > test.txt` 输出字符串到文件

### 4.4.2 cd

作用：跳转工作目录  
用法：`cd [directory]`

### 4.4.3 pwd

作用：查看当前工作目录  
用法：`pwd`

### 4.4.4 ls

作用：列出指定目录或者当前工作目录的内容  
用法：`ls [dir]`

### 4.4.5 rm

作用：删除文件  
用法：`rm [file1] [file2]`

### 4.4.6 mkdir

作用：创建目录  
用法：`mkdir dir`

## 4.4.7 cat

作用：查看文件内容  
用法：cat file

## 4.4.8 hexdump

作用：以16进制显示文件内容  
用法：hexdump file

## 4.4.9 grep

作用：在指定文件中搜索指定字符串  
用法：grep key\_string file

## 4.4.10 touch

作用：创建空文件  
用法：touch file

## 4.4.11 tail

作用：从文件尾部开始查看文件  
用法：tail [-n 1] file

## 4.5 adb 使用

Meis3.x 支持 ADB 功能，使用步骤如下：

### 4.5.1 启动小机端 adbd 服务

命令：adb  
用法：msh控制台输入

```
msh >adb
adb version:AW-V1.1.0, compiled on: Feb 27 2020 14:59:07
adb service init successful
msh >[GADGET-ERROR][usb_gadget_function_write] line:228 hal_udc_ep_write failed, return -7
msh >
msh >[ctrl-a z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyUSB0]
```

图 4-1: adb 命令运行效果图

## 4.5.2 查看设备列表

命令: adb devices  
用法: msh控制台输入

```
czl@czl-All-Series:~/WorkSpace/melis-devel/melis-v3.0/source/ekernel/core/rt-thread$ adb devices
List of devices attached
000c21ebd00c21ebd20    device
czl@czl-All-Series:~/WorkSpace/melis-devel/melis-v3.0/source/ekernel/core/rt-thread$
```

图 4-2: adb devices 命令运行效果图

## 4.5.3 adb 终端登录

命令: adb shell  
用法: msh控制台输入

```
czl@czl-All-Series:~/WorkSpace/melis-devel/melis-v3.0/source/ekernel/core/rt-thread$ adb shell
msh >list_thread
      pri  status      sp      stack size max used left tick  error
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
adb-event          20  suspend  0x0000001c0 0x00002000   12%  0x00000004 000
adb-shell-ser-3    21  suspend  0x000000138 0x00002000   06%  0x00000005 000
adb-shell          21  ready    0x00000064c 0x00002000   25%  0x00000003 000
adb-output          20  suspend  0x000000170 0x00002000   11%  0x00000005 000
adb-input           20  suspend  0x0000001bc 0x00002000   20%  0x00000002 000
tshell              20  suspend  0x000000208 0x00001000   27%  0x0000000a 000
ksrv_task           6   suspend  0x000000118 0x00001000   07%  0x0000000a 000
mnt_task            7   suspend  0x0000000f0 0x00004000   01%  0x0000000a 000
tidle               31  ready    0x000000104 0x00000800   33%  0x00000009 000
timer               8   suspend  0x0000000a8 0x00004000   06%  0x0000000a 000
msh >
```

图 4-3: adb shell 命令运行效果图

## 4.6 使用 Neon

应用户要求, v0.5 版本默认支持了 VFP&Neon 加速, 为了节省任务调度时的资源消耗, 使用 neon 加速的线程必须通过以下接口显示声明对 neon 功能的需求:

函数原型: void kthread\_enable\_fpu(void)

用法: 在线程入口处无条件调用

函数原型: void kthread\_stop\_fpu(void)

用法: 在线程结束时无条件调用

```
25 void kthread_enable_fpu(void);
26 extern uint32_t melis_arch_use_fpu;
27 static void vfp_task1(void *ARG_UNUSED(para))
28 {
29     kthread_enable_fpu();
30     while (1)
31     {
32         rt_thread_delay(100);
33         __log("melis_arch_use_fpu = %d.", melis_arch_use_fpu);
34     }
35 }
36 
```

图 4-4: 函数代码用法图

## 4.7 网络测试

### 4.7.1 配置

#### 4.7.1.1 驱动配置

```
Kernel Setup > Drivers Setup > net drivers
[*] brcm_ap6203 wifi driver
```

#### 4.7.1.2 协议配置

```
Kernel Setup > Subsystem support > Network > Network protocol
-*-
[*] lwip
[*] smtp
[*] mdns
[ ] mqtt
[ ] altcp_tls
[ ] snmp
[ ] netbiosns
[ ] websocket
-*-
[*] mbedtls --->
    [ ] using hardware crypto module for aes.
    [ ] mbedtls test
[*] http
```

### 4.7.1.3 应用配置

```
Kernel Setup > Subsystem support > Network > Network tools
[*] ping
[*] wget
[*] iperf
[*] netio
[*] ntp
[*] tcpdump
[ ] telnet
[*] tftp
[*] httpclient
```

### 4.7.1.4 基础通信测试配置

```
Kernel Setup > Drivers Test Sample >
Net Work
[*] wifi tcpip Test
[*] wifi loop test
[*] wifi socket test
```

## 4.7.2 测试

### 4.7.2.1 网络回环测试

可以通过网络环回测试验证协议栈的稳定性和可靠性, 方式如下：

#### 1.1 初始化 Tcp/IP 协议栈

```
命令: tcpip_init
用法: msh控制台输入
```

#### 1.2 启动本地回环服务器

```
命令: loop_server
用法: msh控制台输入
```

```
msh />tcpip_init
msh />loop_server
Creating loop_if_server_main task
msh />server waiting
msh />
```

图 4-5: loop\_server 命令运行效果图

#### 1.3 启动本地回环客户端

命令: loop\_client

用法: msh控制台输入

```
msh />loop_client
Creating loop_if client main task
msh />[loop_if_client,81],enter
[loop_if_server,49],enter
loop_if_server 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
loop_if_server 1E 1F
=====
loop_if_server 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
loop_if_server 1E 1F
=====
loop_if_server 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
loop_if_server 1E 1F
=====
loop_if_server 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D
loop_if_server 1E 1F
=====
```

图 4-6: loop\_client 命令运行效果图

#### 4.7.2.2 socket 基础测试

搭建 TCP 基础通信测试

##### 1. 启动 tcp 服务器

命令: server

用法: msh控制台输入

##### 2. 启动 tcp 客户端

命令: client

用法: msh控制台输入

#### 4.7.2.3 网络基础功能测试

##### 1. 联网测试

```
bcm_init //初始化
bcm_sta scan //扫描ap数
bcm_sta scanresults //扫描并查看具体ap信息(注意: 必现先执行bcm_sta scan)
bcm_sta connect ssid security passwd //连接加密网络 若未加密使用bcm_sta connect ssid 0 0
bcm_sta disconnect //断开网络(注意: 每一次切换网络前必须先断开)
```

```
msh />bcm_sta connect fly2.4g 8 22224444 //目前不能直接获取加密方式(后期改进)
```

#### 4.7.2.4 网络应用测试

##### 1. ping 测试

```
ping 14.215.177.39 ping www.baidu.com //ping ip/host测试用例
```

##### 2. wgets 下载测试

```
wget http://img.ivsky.com/img/tupian/pre/201312/04/nelumbo_nucifera-009.jpg
```

##### 3. http 测试

```
httpClient_test host url //httpClient_test 192.168.31.6 http://192.168.31.6/henrisk/ranchao  
/test.txt
```

##### 4. ntp 时间同步测试

```
ntp_sync //从ntp服务器同步时间  
date //查看当前本地时间
```

##### 5. 吞吐测试

```
netio_init //用netio工具测试  
iperf //用iperf工具测试  
详细说明见iperf和netio使用文档。
```

## 4.8 文件系统测试

### 4.8.1 插入 SD 卡，自动挂载文件系统

```
===== card information =====
SD:Card Type      : SDHC
SD:Card Spec Ver  : 5.0
SD:Card RCA       : 0xaaaa
SD:Card OCR       : 0x40ff8000
SD:  vol_window   : 0x00ff8000
SD:  to_lv8_acpt  : 0
SD:  high_capac   : 0
SD:Card CSD       :
SD:  speed        : 50000 KHz
SD:  cmd class    : 0x5b5
SD:  capacity     : 15193MB
SD:Card CUR_STA   :
SD:  speed mode   : HS: 50 MHz
SD:  bus width    : 2
SD:  speed class  : 10
SD:=====
SD:***** sd init ok *****
Initial card success
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Online 0:42 | ttyUSB0
```

图 4-7: 插入 SD 卡运行效果图

### 4.8.2 读写压测

命令: rwcheck -d /sdmmc  
用法: msh控制台输入

```
msh />ls
dev data layer sdmmc
msh />rwcheck -d /sdmmc

        rwcheck: do read and write check

        version: v0.1.0
        build: Compiled in Feb 27 2020 at 14:59:09
        date: Thu Jan  1 00:00:22 1970

        free/total ddr: 63748/64116 KB
        free/total flash: 2958840/2958848 KB
        set file size to 128 KB
        set times to 1
        set max percent of total space to 90%
        set check directory as /sdmmc
        set origin file as /sdmmc/rwcheck.org

        --- INIT ---
        create  : /sdmmc/rwcheck.org ... OK (4K)
        --- CREATE ---
        create  : /sdmmc/rwcheck.tmp.0 ... OK (128K)
        create  : /sdmmc/rwcheck.tmp.1 ... OK (128K)
        create  : /sdmmc/rwcheck.tmp.2 ... OK (128K)
        create  : /sdmmc/rwcheck.tmp.3 ... OK (128K)
        create  : /sdmmc/rwcheck.tmp.4 ... OK (128K)
        create  : /sdmmc/rwcheck.tmp.5 ...
```

图 4-8: rwcheck -d /sdmmc 命令运行效果图

## 4.9 G2D 测试用例

命令: g2d\_test  
用法: msh控制台输入

图 4-9: g2d\_test 命令运行效果图

## 5 卡刷机/升级/量产功能支持

从 v0.7 版 sdk 开始，支持卡刷机/升级/量产功能，使客户可以在不具备 USB 升级口的平台上，利用 TF 卡进行固件升级。

具体操作步骤请参考指导文档: ***document/usr-docs/melis 卡量产说明.doc***



## 6 IPC 测试用例

### 6.1 ISP 数据采集/处理用例

命令: fork vin\_isp\_test 0 1920 1080 ./ 1 10000 1 60

用法: msh控制台根目录输入上述命令

效果:





图 6-1: ISP 数据采集命令运行效果图

## 6.2 抓图测试用例

命令: sample\_virvi

用法: msh控制台根目录输入上述命令

说明: 执行后, 采集600帧, 过程中编码出2张jpeg图片, 保存在/mnt/E/picp0.jpg, /mnt/E/pic[1].jpg, TF卡根目录

## 6.3 抓视频存卡用例

命令: sample\_vi2venc2muxer

用法: msh控制台根目录输入上述命令

说明: 执行后, h264编码为MP4文件, 时长约为30秒, 保存在/mnt/E/1080p.mp4, TF卡根目录

限制: 目前存在卡兼容性问题, 最好用高类卡, 并且每次开机只能测试一次, 问题待查。



## 著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

## 商标声明

  **全志科技**  (不完全列举) 均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

## 免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。