



# Linux HDMI2.0 DRM 开发指南

版本号: 1.0  
发布日期: 2024.11.26

## 版本历史

版本号	日期	制/修订人	内容描述
1.0	2024.11.26	AW1977	初始版本



# 目 录

<b>1</b>	<b>概述</b>	<b>1</b>
1.1	编写目的	1
1.2	适用范围	1
1.3	相关人员	1
1.4	相关术语	1
<b>2</b>	<b>模块介绍</b>	<b>3</b>
2.1	模块平台功能	3
2.2	模块框图介绍	3
2.3	模块源码介绍	4
2.4	模块功能配置	5
2.4.1	Uboot 平台	5
2.4.1.1	配置 HDMI	5
2.4.2	Kerne 平台	7
2.4.2.1	配置 HDMI	7
2.4.2.2	配置 HDCP1x	7
2.4.2.3	配置 HDCP2x	7
2.4.2.4	配置 CEC	8
<b>3</b>	<b>模块使用</b>	<b>9</b>
3.1	切换显示设置	9
3.1.1	Android 方案	9
3.1.2	Linux 方案	10
3.2	使用 CEC 功能	13
3.2.1	Android 方案	13
3.2.2	Linux 方案	13
3.3	使用 HDCP 功能	13
3.3.1	使用 HDCP1x 功能	14
3.3.1.1	Android 方案	14
3.3.1.2	Linux 方案	14
3.3.2	使用 HDCP2x 功能	14
3.3.2.1	Android 方案	14
3.3.2.2	Linux 方案	15
3.4	Uboot 输出策略	15
3.4.1	快速模式	16
3.4.2	兼容模式	16
3.5	Uboot 固定输出某个分辨率	16
<b>4</b>	<b>调试节点</b>	<b>18</b>

4.1	Kernel 环境	18
4.1.1	寄存器读写	18
4.1.2	查看 Tx 信息	19
4.1.3	查看 Rx 信息	23
4.1.4	更改输入源	25
4.1.5	修改日志级别	26
4.1.6	EDID 节点	26
4.1.7	HDCP 节点	26
4.2	Uboot 环境	27
4.2.1	查看命令帮助	27
4.2.2	查看 Tx 信息	27
4.2.3	查看 Rx 信息	27
4.2.4	更改输入源	27
4.2.5	寄存器读写	27
5	FAQ	29
5.1	未检测到 HPD	29
5.2	测量 TMDS Clock	29
5.3	Uboot 平台	29
5.3.1	Q1: 无信号问题	29
5.3.2	Q2: 黑屏问题	30
5.4	Kernel 平台	31
5.4.1	Q1: 无信号问题	31
5.4.2	Q2: 黑屏问题	32
5.4.3	Q3: 无声问题	32
6	日志反馈	34
7	附录 1: HDCP1x 密钥烧录	35
7.1	密钥配置	35
7.2	密钥烧录	35
8	附录 2: HDCP2x 密钥烧录	40
8.1	密钥配置	40
8.2	密钥烧录	40
9	附录 3: Linux CEC 应用程序样例	46
10	附录 4: Linux HDCP 应用程序样例	55
11	附录 5: HDMI 展频配置及应用	61
11.1	PHYPLL 模式展频	62
11.2	CCMU 模式展频	63
12	结束	64

## 插 图

图 2-1	显示框图 . . . . .	3
图 2-2	HDMI 模块框图 . . . . .	4
图 3-1	modetest 组件 ID . . . . .	10
图 3-2	modetest 查看 HDMI 支持分辨率 . . . . .	11
图 3-3	modetest 查看 HDMI Property . . . . .	12
图 4-1	内核 TX 信息 . . . . .	19
图 4-2	内核 Rx 信息 . . . . .	24
图 7-1	HDCP1x 密钥烧录配置-1 . . . . .	36
图 7-2	HDCP1x 密钥烧录配置-2 . . . . .	37
图 7-3	HDCP1x 密钥烧录配置-3 . . . . .	38
图 7-4	HDCP1x 密钥烧录配置-4 . . . . .	39
图 8-1	HDCP2x 密钥烧录配置-1 . . . . .	41
图 8-2	HDCP2x 密钥烧录配置-2 . . . . .	42
图 8-3	HDCP2x 密钥烧录配置-3 . . . . .	43
图 8-4	HDCP2x 密钥烧录配置-4 . . . . .	44
图 11-1	附录 5-确定模块时钟来源 . . . . .	61
图 11-2	附录 5-当前 TMDS 时钟 . . . . .	61
图 11-3	查看 TOPPHY 状态 . . . . .	62

# 1 概述

## 1.1 编写目的

介绍如何基于 DRM 显示框架来配置开发 HDMI 以及基本问题调试。

## 1.2 适用范围

表 1-1: 适用产品列表

IC	安卓版本	内核版本
A733	Android_V	Linux-6.6

### 说明

此文档仅适用于 DRM 显示框架。

## 1.3 相关人员

HDMI 模块开发工程师

HDMI 模块测试工程师

## 1.4 相关术语

术语	说明
DRM	全称 Direct Rendering Manager，是 DRI(Direct Rendering Infrastructure) 框架的一个组件。
HDMI	全称 High Definition Multimedia Interface，高清晰多媒体接口
HDCP	全称 High-Bandwidth Digital Content Protection，高带宽数字内容保护技术
HDCP1x	HDCP 版本，在全志平台上支持 HDCP1.4 版本，其向下兼容 HDCP1.3。
HDCP2x	HDCP 版本，在全志平台上支持 HDCP2.2 版本。不兼容 HDCP1x 版本

术语	说明
DE	Allwinner 的 Display 模块。
tv0	Allwinner 的 Tcon Tv 模块，在对应的设备树上命名为 tv0。
dw	Allwinner 平台上使用的 HDMI2.0 IP 代号
Tx	指的是 HDMI 发送端，常见的有盒子，电脑等
Rx	指的是 HDMI 接收端，常见的有电视、显示器等
EDID	Rx 用来表示能力的信息

#### 说明

本章节描述术语仅作用于本文档。若术语与外部常见术语含义不同，以本章节描述术语为准



## 2 模块介绍

### 2.1 模块平台功能

Allwinner HDMI 模块基于 HDMI2.0 协议开发完成，其向下兼容 HDMI1.4 协议。

#### ⚠ 注意

由于不同 SDK 平台针对的使用场景不同，部分功能虽然硬件支持，但是 SDK 可能没有支持上。请综合考虑以下芯片硬件功能支持表和 SDK 平台功能实现表。

表 2-1: 芯片硬件功能支持表

芯片	HDMI2.0	CEC	HDCP14	HDCP22
A733	Y	Y	Y	Y

表 2-2: SDK 平台功能实现表

SDK 平台	HDMI2.0	CEC	HDCP14	HDCP22	Uboot 输出
A733 Android15	Y	N	Y	Y	Y

### 2.2 模块框图介绍

#### HDMI 显示框图

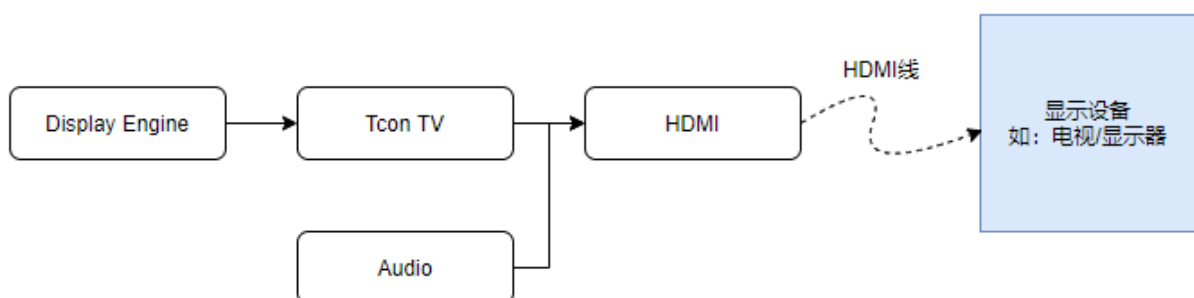


图 2-1: 显示框图



HDMI 是一个显示接口模块，负责将 TconTV 输出的视频数据和 Audio 输出的音频数据进行融合打包组成 TMDS 音视频数据再通过 HDMI 线传输到显示设备上。

## HDMI 模块框图

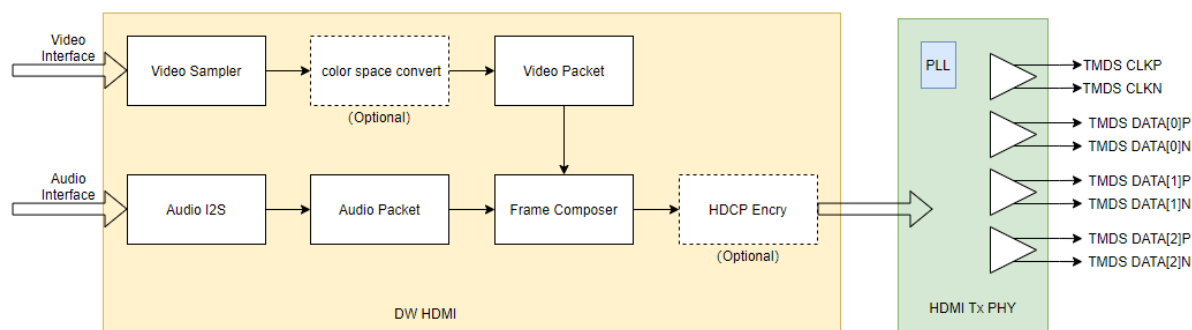


图 2-2: HDMI 模块框图

HDMI 模块内部大致分为视频处理模块、音频处理模块、帧打包模块和 PHY 模块。

视频处理模块：对输入的视频进行采样并重新打包

音频处理模块：对输入的音频进行采样并重新打包

帧处理模块：对音频和视频数据按照帧数据的方式进行融合打包，并添加相关的 infoframe 信息。

PHY 处理模块：将完整的帧数据按照 TMDS 协议转换为 TMDS Data，并输出到显示端。

## 2.3 模块源码介绍

### 源码路径

```
{Android_Top}/longan/bsp/drivers/drm
or
{Linux_Top}/bsp/drivers/drm
```

### 源码文件

```
.
├── Kconfig
├── Makefile
├── sunxi_device
│   └── hardware
│       ├── ...
│       └── lowlevel_hdmi20
│           ├── dw_avp.c
│           ├── dw_avp.h
│           ├── dw_cec.c
│           ├── dw_cec.h
│           ├── dw_dev.c
│           ├── dw_dev.h
│           └── dw_edid.c
```

```
| | | | | dw_edid.h
| | | | | dw_fc.c
| | | | | dw_fc.h
| | | | | dw_hdcp22.c
| | | | | dw_hdcp22.h
| | | | | dw_hdcp.c
| | | | | dw_hdcp.h
| | | | | dw_i2cm.c
| | | | | dw_i2cm.h
| | | | | dw_mc.c
| | | | | dw_mc.h
| | | | | dw_phy.c
| | | | | dw_phy.h
| | | | | esm_lib
| | | | | Makefile
| | | | | phy_aw.c
| | | | | phy_aw.h
| | | | | phy_inno.c
| | | | | phy_inno.h
| | | | | phy_snps.c
| | | | | phy_snps.h
| | | | | ...
| | | | | sunxi_hdmi.h
| | | | | sunxi_hdmi.c
| | | | | ...
| | | | | sunxi_drm_hdmi.c
| | | | | sunxi_drm_hdmi.h
```

sunxi\_drm\_hdmi：顶层文件。主要用于平台设备及 DRM Connector 组件的注册、功能模块配置处理逻辑。

sunxi\_hdmi：中间层文件。主要用于适配所使用的硬件 IP 版本，融合转换顶层和底层配置。

dw\_xxx：底层硬件配置文件。主要用于底层 HDMI2.0 硬件 IP 的音视频等配置功能。

## 2.4 模块功能配置

### 2.4.1 Uboot 平台

#### 2.4.1.1 配置 HDMI

**Step1：在 dts 中增加 tout\_e\_hdmi 节点。**

以 A733 为例对应 dts 路径为

```
{longan}/brandy/brandy-2.0/u-boot-2018/arch/arm/dts/sun60iw2p1-soc-system.dts
```

```
sunxi_drm: sunxi-drm {
    compatible = "allwinner,sunxi-drm";
    fb_base = <0>;
    status = "okay";
    route {
```

```
...
};
};
```

### ⚠ 注意

不管将 HDMI 接入主显还是副显，都需要确保没有与其他显示接口冲突！比如不能将 LVDS 接入主显，HDMI 也接入主显！

假设将主显接到为 HDMI，新增的 route\_hdmi 节点如下：

```
sunxi_drm: sunxi-drm {
    compatible = "allwinner,sunxi-drm";
    fb_base = <0>;
    status = "okay";
    route {
        route_hdmi: route_hdmi {
            status = "disabled";
            endpoints = <&disp0_out_tcon3 &tcon3_out_hdmi>; //disp0_out_tcon3指的是主显输出到hdmi
            logo,uboot = "bootlogo.bmp";
        };
    };
};
```

假设将副显接到为 HDMI，新增的 route\_hdmi 节点如下：

```
sunxi_drm: sunxi-drm {
    compatible = "allwinner,sunxi-drm";
    fb_base = <0>;
    status = "okay";
    route {
        route_hdmi: route_hdmi {
            status = "disabled";
            endpoints = <&disp1_out_tcon3 &tcon3_out_hdmi>; //disp1_out_tcon3指的是副显输出到hdmi
            logo,uboot = "bootlogo.bmp";
        };
    };
};
```

### Step2: uboot-board.dts 中使能 route\_hdmi

```
&route_hdmi {
    status = "okay";
}
```

### Step3: uboot-board.dts 中使能 vo0

```
&vo0 {
    status = "okay";
}
```

### Step4: uboot-board.dts 中使能 tv0

```
&tv0 {
    status = "okay";
}
```

## 2.4.2 Kerne 平台

### 2.4.2.1 配置 HDMI

#### Step1

在 menuconfig 中配置编译 HDMI 模块，选择 HDMI2.0

```
Allwinner BSP --->
Device Drivers --->
DRM Drivers --->
[*] DRM Support for Allwinner SoCs
[*] Support Hdmi Output --->
[*] Support HDMI2.0 Output
```

#### Step2

在 board.dts 中使能 tv0 和 hdmi。

```
&tv0 {
    status = "okay";
}

&hdmi {
    ...
    status = "okay";
}
```

### 2.4.2.2 配置 HDCP1x



说明

如果要配置使用 HDCP 功能，那么必须编译为安全固件

在 board.dts 中使能 hdcp1x

```
&hdmi {
    ...
    hdmi_hdcp1x_enable = <1>; # 写1表示使能; 写0表示禁用
}
```

### 2.4.2.3 配置 HDCP2x



说明

如果要配置使用 HDCP 功能，那么必须编译为安全固件

在 board.dts 中使能 hdcp2x

```
&hdmi {  
...  
    hdmi_hdcp2x_enable = <1>; # 写1表示使能; 写0表示禁用  
}
```

#### 2.4.2.4 配置 CEC

在 board.dts 中使能 cec

```
&hdmi {  
...  
    hdmi_cec_enable = <1>; # 写1表示使能; 写0表示禁用  
}
```



## 3 模块使用

### 3.1 切换显示设置

#### 3.1.1 Android 方案

##### 方法 1：通过 UI 界面直接设置

设置 -> 显示 -> 副显输出模式 -> 选择对应的分辨率即可

##### 方法 2：通过 dispconfig 工具设置。

使用之前需要先确认 HDMI 是主显还是副显。如果主显，-o 后面所接参数是 0；如果是副显，-o 后面所接参数是 1。

- 查看命令帮助信息

```
dispconfig -h
```

- 查看 HDMI 支持的分辨率

```
dispconfig -o 1 -p # 当前HDMI是副显
```

输出信息如下：

```
Display.1 HDMI-A connected mode=203
[164] 3840x2160@60.00Hz 3840x2160
* [203] 4096x2160@60.00Hz 4096x2160 # 此项带有星号*, 说明当前选择的是203, 4096x2160@60Hz
[204] 4096x2160@59.94Hz 4096x2160
[205] 4096x2160@50.00Hz 4096x2160
...
[217] 1920x1080@60.00Hz 1920x1080
```

- 切换指定分辨率

基于上面指令获取到的分辨率信息，假设切换到 1080P 信号。

```
dispconfig -o 1 -s 217 # 查看上面输出信息可以知道1080P60对应的是217
```

##### 📖 说明

每一次拔插或者重启，分辨率对应的索引值都是变化的。因此需要先查看确认当前的索引值！

- 切换指定颜色格式和色深

假设切换到 YUV444-8Bits

```
dispconfig -o 1 -f yuv444-8 # 使用-f并带上参数yuv444-8
```

## 3.1.2 Linux 方案

### 使用 modetest 工具来配置

modetest工具请找对接的FAE获取，需要针对SDK来生成

需先查看 HDMI 是否有正常的绑定到对应的 CRTC 输出。

```
./modetest -M sunxi-drm
```

```
/tmp # ./modetest -M sunxi-drm
Encoders:
id crtc type possible crtcs possible clones
139 0 Virtual 0x00000003 0x00000001
145 99 LVDS 0x00000003 0x00000002
147 125 TMDS 0x00000003 0x00000004

Connectors:
id encoder status name size (mm) modes encoders
146 145 connected LVDS-1 0x0 1 145
modes:
name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot
1280x800 60 1280 1363 1381 1451 800 837 847 860 74871 flags: ; type: preferred, driver
props:
1 EDID:
flags: immutable blob
blobs:
value:
2 DPMS:
flags: enum
enums: On=0 Standby=1 Suspend=2 Off=3
value: 0
5 link-status:
flags: enum
enums: Good=0 Bad=1
value: 0
6 non-desktop:
flags: immutable range
values: 0 1
value: 0
4 TILE:
flags: immutable blob
blobs:
value:
20 CRTC_ID:
flags: object
value: 99
148 147 connected HDMI-A-1 1220x680 46 147
modes:
name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot
3840x2160 60 3840 4016 4104 4400 2160 2168 2178 2250 594000 flags: phsync, pvsync; type: preferred, driver
4096x2160 60 4096 4184 4272 4400 2160 2168 2178 2250 594000 flags: phsync, pvsync; type: driver
```

图 3-1: modetest 组件 ID

在Connectors中可以看到，HDMI-A-1 对应的 connector id 是 148，其绑定的 encoder id 是 147。

在Encoders中可以看到，encoder id 147 绑定的 crtc id 是 125。

从上述信息可以得知，HDMI 当前通路是 CRTC(125) — Encoder(147) — Connector(148) 的。

#### 说明

每次更新固件这些 ID 的值都可能发生变化，因此都需要先查看对应的 ID 值

### • 查看 HDMI 所支持的分辨率

```
./modetest -M sunxi-drm -c # 仅输出connector的信息
```

```
148 147 connected HDMI-A-1 1220x680 46 147
modes:
  name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
3840x2160 60 3840 4016 4104 4400 2160 2168 2178 2250 594000 flags: phsync, pvsync; type: preferred, driver
4096x2160 60 4096 4184 4272 4400 2160 2168 2178 2250 594000 flags: phsync, pvsync; type: driver
4096x2160 60 4096 4184 4272 4400 2160 2168 2178 2250 593407 flags: phsync, pvsync; type: driver
4096x2160 50 4096 5064 5152 5280 2160 2168 2178 2250 594000 flags: phsync, pvsync; type: driver
4096x2160 24 4096 5116 5204 5500 2160 2168 2178 2250 297000 flags: phsync, pvsync; type: driver
4096x2160 24 4096 5116 5204 5500 2160 2168 2178 2250 296703 flags: phsync, pvsync; type: driver
3840x2160 60 3840 4016 4104 4400 2160 2168 2178 2250 594000 flags: phsync, pvsync; type: driver
3840x2160 60 3840 4016 4104 4400 2160 2168 2178 2250 593407 flags: phsync, pvsync; type: driver
3840x2160 50 3840 4896 4984 5280 2160 2168 2178 2250 594000 flags: phsync, pvsync; type: driver
3840x2160 30 3840 4016 4104 4400 2160 2168 2178 2250 297000 flags: phsync, pvsync; type: driver
3840x2160 30 3840 4016 4104 4400 2160 2168 2178 2250 296703 flags: phsync, pvsync; type: driver
3840x2160 25 3840 4896 4984 5280 2160 2168 2178 2250 297000 flags: phsync, pvsync; type: driver
3840x2160 24 3840 4896 4984 5280 2160 2168 2178 2250 297000 flags: phsync, pvsync; type: driver
3840x2160 24 3840 5116 5204 5500 2160 2168 2178 2250 296703 flags: phsync, pvsync; type: driver
1920x1080 60 1920 2008 2052 2200 1080 1084 1089 1125 148500 flags: phsync, pvsync; type: driver
1920x1080 60 1920 2008 2052 2200 1080 1084 1089 1125 148500 flags: phsync, pvsync; type: driver
1920x1080 60 1920 2008 2052 2200 1080 1084 1089 1125 148352 flags: phsync, pvsync; type: driver
1920x1080i 60 1920 2008 2052 2200 1080 1084 1094 1125 74250 flags: phsync, pvsync, interlace; type: driver
1920x1080i 60 1920 2008 2052 2200 1080 1084 1094 1125 74176 flags: phsync, pvsync, interlace; type: driver
1920x1080 50 1920 2448 2492 2640 1080 1084 1089 1125 148500 flags: phsync, pvsync; type: driver
1920x1080i 50 1920 2448 2492 2640 1080 1084 1094 1125 74250 flags: phsync, pvsync, interlace; type: driver
1920x1080 30 1920 2008 2052 2200 1080 1084 1089 1125 74250 flags: phsync, pvsync; type: driver
1920x1080 30 1920 2008 2052 2200 1080 1084 1089 1125 74176 flags: phsync, pvsync; type: driver
1920x1080 24 1920 2558 2602 2750 1080 1084 1089 1125 74250 flags: phsync, pvsync; type: driver
1920x1080 24 1920 2558 2602 2750 1080 1084 1089 1125 74176 flags: phsync, pvsync; type: driver
1680x1050 60 1680 1728 1760 1840 1050 1053 1059 1080 119000 flags: phsync, nvsync; type: driver
1600x900 60 1600 1624 1704 1800 900 901 904 1000 108000 flags: phsync, pvsync; type: driver
1280x1024 60 1280 1328 1440 1688 1024 1025 1028 1066 108000 flags: phsync, pvsync; type: driver
1280x720 60 1280 1390 1430 1650 720 725 730 750 74250 flags: phsync, pvsync; type: driver
1280x720 60 1280 1390 1430 1650 720 725 730 750 74250 flags: phsync, pvsync; type: driver
1280x720 60 1280 1390 1430 1650 720 725 730 750 74176 flags: phsync, pvsync; type: driver
1280x720 50 1280 1720 1760 1980 720 725 730 750 74250 flags: phsync, pvsync; type: driver
1280x720 30 1280 3040 3080 3300 720 725 730 750 74250 flags: phsync, pvsync; type: driver
1280x720 30 1280 3040 3080 3300 720 725 730 750 74176 flags: phsync, pvsync; type: driver
1280x720 24 1280 3040 3080 3300 720 725 730 750 59400 flags: phsync, pvsync; type: driver
1280x720 24 1280 3040 3080 3300 720 725 730 750 59341 flags: phsync, pvsync; type: driver
1024x768 60 1024 1048 1184 1344 768 771 777 806 65000 flags: nhsync, nvsync; type: driver
800x600 60 800 840 968 1056 600 601 605 628 40000 flags: phsync, pvsync; type: driver
720x576 50 720 732 796 864 576 581 586 625 27000 flags: nhsync, nvsync; type: driver
720x576 50 720 732 796 864 576 581 586 625 27000 flags: nhsync, nvsync; type: driver
720x480 60 720 736 798 858 480 489 495 525 27027 flags: nhsync, nvsync; type: driver
720x480 60 720 736 798 858 480 489 495 525 27027 flags: nhsync, nvsync; type: driver
720x480 60 720 736 798 858 480 489 495 525 27000 flags: nhsync, nvsync; type: driver
720x480 60 720 736 798 858 480 489 495 525 27000 flags: nhsync, nvsync; type: driver
640x480 60 640 656 752 800 480 490 492 525 25200 flags: nhsync, nvsync; type: driver
640x480 60 640 656 752 800 480 490 492 525 25175 flags: nhsync, nvsync; type: driver
```

图 3-2: modetest 查看 HDMI 支持分辨率

在modes信息下面，显示的就是当前 HDMI 设备所支持的分辨率信息。



## ● 切换指定的分辨率

示例：切换到 1080P60Hz

```
./modetest -M sunxi-drm -s 148@125:1920x1080-60 # 148表示当前环境下HDMI的connector id, 125表示当前HDMI绑定的CRTC ID。
```

## ● 切换指定颜色和色深

先查看当前输出的分辨率下支持哪些颜色和色深。

```
./modetest -M sunxi-drm -c # 仅输出connector的信息，手动删除部分信息，避免篇幅过长
```

```
148 147 connected HDMI-A-1 1220x680 46 147
modes:
  name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
  3840x2160 60 3840 4016 4104 4400 2160 2168 2178 2250 594000 flags: phsync, pvsync; type: preferred, driver
  4096x2160 60 4096 4184 4272 4400 2160 2168 2178 2250 594000 flags: phsync, pvsync; type: driver
  4096x2160 60 4096 4184 4272 4400 2160 2168 2178 2250 593407 flags: phsync, pvsync; type: driver
  .....
  1920x1080 60 1920 2008 2052 2200 1080 1084 1089 1125 148500 flags: phsync, pvsync; type: driver
  1920x1080 60 1920 2008 2052 2200 1080 1084 1089 1125 148500 flags: phsync, pvsync; type: driver
  .....
props:
  .....
149 pixelformat_support:
  flags: bitmask
  values: RGB888_8BITS=0x1 YUV444_8BITS=0x2 YUV422_8BITS=0x4 YUV420_8BITS=0x8 RGB888_10BITS=0x10 YUV444_10BITS=0x20 YUV422_10BITS=0x40 YUV420_10BITS=0x80
  value: 207
150 pixelformat:
  flags: range
  values: 0 255
  value: 1
151 dynamicrange_support:
  flags: bitmask
  values: SDR=0x1 HDR10=0x2 HDR10P=0x4 HLG=0x8
  value: 11
152 dynamicrange:
  flags: range
  values: 0 31
  value: 1
```

图 3-3: modetest 查看 HDMI Property

从pixelformat\_support可以看到 value = 207，对应了二进制是：11001111。

结合 values 的说明，知道当前模式下支持：RGB888\_8BITS, YUV444\_8BITS, YUV422\_8BITS, YUV420\_8BITS, YUV422\_10BITS, YUV420\_10BITS。

从pixelformat可以看到 value = 1，说明当前输出的是RGB888\_8BITS。

示例：切换到 YUV444\_8BITS 显示

```
./modetest -M sunxi-drm -w 148:pixelformat:2
# 148是HDMI的connector id
# pixelformat是当前这个property的名称，固定
# 2表示切换到YUV444_BITS，参考pixelformat_support的values说明。
```

## 3.2 使用 CEC 功能

### ⚠ 注意

请先查看[模块平台功能](#)，确认当前平台是否支持 CEC 功能。

### 3.2.1 Android 方案

Android 已自带 CEC 应用服务，无需额外开发，部分功能需要单独在安卓设置页面开启；

设置路径：Settings -> Device Preferences -> Inputs

# PS: 不支持 CEC 的平台，没有这个设置页面！

### 3.2.2 Linux 方案

驱动对接的是内核原生 CEC 框架，具体接口请查阅 [各版本的内核文档](#)。

Linux 方案则需要客户自行开发应用程序，参考[附录 3：Linux CEC 应用程序样例](#)

## 3.3 使用 HDCP 功能

AW 部分 IC 和 SDK 同时支持 HDCP1x 和 HDCP2x，因此涉及到同时具备两个版本 HDCP 时的策略选择。目前 AW 配置 HDCP 策略如下：

#### 场景 1：当 SDK 仅配置了 HDCP1x 并且仅烧录了 HDCP1x

》此时 Tx 只支持 HDCP1x 功能。当检测到 Rx 支持 HDCP1x 时，就可以通过使能接口开启 HDCP1x 加密。

#### 场景 2：当 SDK 仅配置了 HDCP2x 并且仅烧录了 HDCP2x

》此时 Tx 只支持 HDCP2x 功能。

当检测到 Rx 支持 HDCP2x 时，就可以通过使能接口开启 HDCP2x 加密。

当检测到 Rx 不支持 HDCP2x 功能时，此时通过接口开启 HDCP 会默认开启不成功，并且输出清流。

#### 场景 3：当 SDK 配置了 HDCP1x 和 HDCP2x，并且烧录了 HDCP1x 和 HDCP2x

》此时 Tx 支持了 HDCP1x 和 HDCP2x 功能。

当检测到 Rx 支持 HDCP2x 时，通过使能接口开启 HDCP 的话，优选选择使用 HDCP2x 加密。

当检测到 Rx 仅支持 HDCP1x 时，通过使能接口开启 HDCP 的话，只能选择使用 HDCP1x 加密。

### 3.3.1 使用 HDCP1x 功能

#### 说明

请先查看[模块平台功能](#)，确认当前平台是否支持 HDCP1.4 功能。

**Step1:** 参考[配置 HDCP1x](#)和[附录 1 HDCP1x 密钥烧录](#)，完成 SDK 配置和密钥烧录；

#### 3.3.1.1 Android 方案

**Step2:** 开机后连接支持 HDCP1.4 的 RX 设备，并执行以下命令即可开启和关闭 HDCP1.4；

```
# 开启 HDCP1.4
hdcpool -e 1

# 关闭 HDCP1.4
hdcpool -e 0
```

#### 3.3.1.2 Linux 方案

请参考附录章节：[附录 4: Linux HDCP 应用程序样例](#)

**Step2:** 开机后连接支持 HDCP1.4 的 RX 设备，并执行以下命令即可开启和关闭 HDCP1.4；

```
su
echo 1 > /sys/class/hdmi/hdmi/attr/hdcp_enable
```

### 3.3.2 使用 HDCP2x 功能

#### 说明

请先查看[模块平台功能](#)，确认当前平台是否支持 HDCP2.2 功能。

**Step1:** 参考[配置 HDCP2x](#)和[附录 2: HDCP2x 密钥烧录](#)，完成 SDK 配置和密钥烧录，并获取到生成好的 esm.fex

#### 3.3.2.1 Android 方案

**Step2:** 将参考 HDCP2x 密钥烧录中生成的 esm.fex 换以下路径中的文件

```
<安卓sdk>/device/softwinner/<平台>/common/display/
```

**Step3:** 开机后连接支持 HDCP2.2 的 RX 设备，并执行以下命令即可开启和关闭 HDCP2.2；

```
# 开启 HDCP2.2
hdcpool -e 1

# 关闭 HDCP2.2
hdcpool -e 0
```

### 3.3.2.2 Linux 方案

**Step2:** 将参考 HDCP2x 密钥烧录中生成的 esm.fex 通过 adb push 进自定义路径。

```
# 以下示例：esm.fex路径：/data/hdcp22/esm.fex
# 导入esm上一级路径
echo /data/hdcp22 > /sys/module/firmware_class/parameters/path
# 启动固件加载
echo 1 > /sys/class/hdmi/hdmi/attr/hdcp_loader
```

**Step3:** 开机后连接支持 HDCP2.2 的 RX 设备，并执行以下命令即可开启和关闭 HDCP2.2；

```
# 使能hdcp22功能
echo 1 > /sys/class/hdmi/hdmi/attr/hdcp_enable

# 关闭HDCP2x功能
echo 0 > /sys/class/hdmi/hdmi/attr/hdcp_enable
```

## 3.4 Uboot 输出策略

Uboot 存在两种启动方式：**快速模式**和**兼容模式**。

### 快速模式

即以固定分辨率 (如 1080P) 出图，抛弃 EDID 读取和一些 infoframe 发送，以快速出图为目的。

如果使用了此模式，那么将不再读取分辨率存储功能中的分辨率等信息，而是强制以配置的默认分辨率出图。

劣势：但存在部分显示器/特殊显示器不兼容此分辨率的情况。

### 兼容模式

即支持读取 EDID 中的分辨率，并参考 EDID 来配置出图分辨率，兼容各类显示设备的出图。

兼容模式将会读取分辨率存储功能中的分辨率等信息和当前 EDID 所解析出来的分辨率等信息

如果匹配那么将优先使用分辨率存储功能的分辨率来出图，反之则使用 EDID 中最大分辨率来出图。

劣势：出图速度较慢，因为涉及到 EDID 的读取和解析。

### 3.4.1 快速模式

在 uboot-board.dts 中，找到 hdmi 节点。修改 uhdmi\_fast\_output

```
&hdmi {
    ...
    /*
     * @uhdmi_fast_output: config uboot hdmi boot mode
     *   - 1: use fast output mode. will only output 1080P video.
     *   - 0: use compatible output mode. will read edid and check config mode
     * @uhdmi_resistor_select: select hdmi resistor config. Actual onboard config!!!
     *   - 1: use onboard resistor
     *   - 0: use onchip resistor
     */
    uhdmi_fast_output = <1>;
    status = "okay";
};
```

### 3.4.2 兼容模式

在 uboot-board.dts 中，找到 hdmi 节点。修改 uhdmi\_fast\_output

```
&hdmi {
    ...
    /*
     * @uhdmi_fast_output: config uboot hdmi boot mode
     *   - 1: use fast output mode. will only output 1080P video.
     *   - 0: use compatible output mode. will read edid and check config mode
     * @uhdmi_resistor_select: select hdmi resistor config. Actual onboard config!!!
     *   - 1: use onboard resistor
     *   - 0: use onchip resistor
     */
    uhdmi_fast_output = <0>;
    status = "okay";
};
```

## 3.5 Uboot 固定输出某个分辨率

比如我们希望 Uboot 阶段输出固定在某一个分辨，比如 720P 或者 1080P 等。我们需要修改代码来实现。

#### 源码路径

```
{longan}/brandy/brandy-2.0/u-boot-2018/drivers/video/drm/sunxi_drm_hdmi.c
```

#### 源码修改

主要是根据自己实际的分辨率，修改 “1920x1080” 这个名称，以及对应的分辨率信息：148500,1920, 2008, 2052, 2200, 0, 1080, 1084, 1089, 1125, 0,

```
const struct drm_display_mode _sunxi_hdmi_default_modes[1] = {
    { DRM_MODE("1920x1080", DRM_MODE_TYPE_DRIVER, 148500,
        1920, 2008, 2052, 2200, 0, 1080, 1084, 1089, 1125, 0,
        DRM_MODE_FLAG_PHSYNC | DRM_MODE_FLAG_PVSYNC)},

    /* { DRM_MODE("3840x2160", DRM_MODE_TYPE_DRIVER, 594000,
        3840, 4016, 4104, 4400, 0, 2160, 2168, 2178, 2250, 0,
        DRM_MODE_FLAG_PHSYNC | DRM_MODE_FLAG_PVSYNC)}, */
};
```



## 4 调试节点

### 4.1 Kernel 环境

HDMI 模块的调试节点路径：/sys/class/hdmi/hdmi/attr

#### 说明

因 SDK 处于逐步修改升级中，因此下面的一些节点输出信息可能存在差异。

#### 4.1.1 寄存器读写

##### 说明

寄存器值的输出都是通过串口打印的，因此需要先将串口输出等级调到 8。可用 `dmesg -n8` 开启

与寄存器读写相关的节点有：reg\_bank，reg\_read，reg\_write

节点名	节点功能描述
reg_bank	用来切换下一次 reg_read 或者 reg_write 操作哪部分的寄存器
reg_read	根据 reg_bank 指定来读对应 offset 寄存器值
reg_write	根据 reg_bank 指定来写对应寄存器值

#### 获取当前所支持读写的寄存器区间

```
cat /sys/class/hdmi/hdmi/attr/reg_bank
```

#### 切换要操作的寄存器区间

```
echo index > /sys/class/hdmi/hdmi/attr/reg_bank
```

#### 对应区间的寄存器读操作

```
echo addr,count > /sys/class/hdmi/hdmi/attr/reg_read
# addr: 该区间寄存器的偏移地址。
# count: 读取寄存器个数，逐步递增
```

#### 对应区间的寄存器写操作

```
echo add,value > /sys/class/hdmi/hdmi/attr/reg_write
# addr: 该区间寄存器的偏移地址
# value: 该区间寄存器需要写入的值
```

## 4.1.2 查看 Tx 信息

```
cat /sys/class/hdmi/hdmi/attr/hdmi_source
```

输出信息如下：

```
[ver]
- hw: 2.0
- sw: 2.24.1111-1

[drv cfg]
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| name | cec | hdcplx | hdc2x | clk_src | resistor | enable | mode set | mode info | force |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| state | off | on | on | phypll | onboard | yes | yes | 3840x2160 | off |

[drv state]
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| name | driver | hpd | hdp | hdp | hdp | hdp | hdp | hdp | hdp |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| state | on | on | running | in | 0x0 | disable | on | off | 0xcf |

[dw ctrl]
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| name | pixel clk | tmds clk | repeat | color bits | pixel | tmds | repeat | audio | csc | cec | hdp | mode | ref clk | rate |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| state | 594000 | 594000 | 0 | 8 | on | on | off | on | on | on | off | standard | 24MHz | 48.7K |

[dw video]
| name | tmds | avmute | scramble | vic | timing | rate | format | bits | pixel_clk | tmds_clk |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| state | hdmi | off | on | 097 | 3840x2160p | 60 | RGB | 8 | lock | lock |

[dw audio]
| name | freq(Hz) | factor | n | cts | mode | mute | fifo | m_clk | i2s_clk |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| state | 44100 | 64 | 9048 | 0 | auto | off | ok | on | unlock |

[dw hdp]
| name | mode | path |
|-----|-----|-----|
| state | dvi | hdp2x |

[dw phy]
| name | mp11 | power | rxsense | hpd | hpd sense |
|-----|-----|-----|-----|-----|-----|
| state | lock | on | 0xf | in | enable |

[snps phy]
- mp11 : 0x0640, 0x3080, 0x0005
- drive: 0x0000, 0x0240, 0x80F9

[top phy]
| name | ref clk | output | state |
|-----|-----|-----|-----|
| state | 24M | on | lock |
- pll: 0xE8036200, 0x00035000, 0x00000000, 0x30000000
```

图 4-1: 内核 TX 信息

[ver]

名称	含义
hw	硬件 IP 版本
sw	软件版本

[drv cfg - dts]



名称	含义
cec	on 表示 DTS 配置打开了 CEC (关注) off 表示 DTS 配置关闭了 CEC
hdcp1x	on 表示 DTS 配置打开了 HDCP1x (关注) off 表示 DTS 配置关闭了 HDCP1x
hdcp2x	on 表示 DTS 配置打开了 HDCP2x (关注) off 表示 DTS 配置关闭了 HDCP2x
clk_src	phy_pll 表示 DTS 配置 HDMI 的时钟源为 PHY PLL ccmu 表示 DTS 配置 HDMI 的时钟源为 CCMU <b>此配置影响模块工作，非特殊要求不要修改！</b>
resistor	onboard 表示 DTS 配置使用板载阻值校准 onchip 表示 DTS 配置使用内部阻值校准 <b>此配置影响信号质量，非特殊要求不要修改！</b>

## [drv cfg - drm]

名称	含义
enable	yes 表示 drm 框架有设置 HDMI 使能。反之为 not。
mode set	yes 表示 drm 框架有设置 HDMI 输出的分辨率。反之为 not。
mode info	表示当前 DRM 框架设置 HDMI 需要输出的分辨率。
force	on 表示 DRM 框架设置了 HDMI 强插入。反之为 off。

## [drv state]

名称 1	名称 2	含义
driver	clock	on 表示 HDMI 驱动时钟资源正常打开。反之为 off
	output	on 表示 HDMI 驱动有尝试配置使能输出。反之为 off
hpd	thread	runing 表示检测 HPD 状态线程正常运行。反之 stop。
	state	in 表示 HDMI 驱动线程有检测到 HPD 状态插入。反之 out。
	mask	0x0 标志 HDMI 驱动线程正常检测 0x11 表示强制模拟为 HPD 插入 0x10 表示强制模拟 HPD 拔出
hdcp	encyption	表示当前 HDMI 驱动获取到 HDCP 加密的状态
	clock	on 表示当前有打开 HDCP 模块需要的时钟。反之 off
cec	clock	on 表示当前有打开 CEC 模块需要的时钟。反之 off
support format		表示当前分辨率下支持的输出的颜色模式。

## [dw ctrl - control params]

名称	含义
pixel clk	dw 层软件配置的 pixel 时钟，单位为 KHz。
tmids clk	dw 层软件配置的 tmids 时钟，单位为 KHz。
repeat	dw 层软件所配置 pixel 时钟重放次数。
color bits	dw 层软件所配置当前分辨率的色深。

### [dw ctrl - control params]

名称	含义
pixel	on 表示打开了 dw 硬件的 pixel 模块门时钟。反之 off
tmids	on 表示打开了 dw 硬件的 tmids 模块门时钟。
repeat	on 表示打开了 dw 硬件的 pixel repeat 模块门时钟。反之 off
audio	on 表示打开了 dw 硬件的 audio 模块门时钟。反之 off
csc	on 表示打开了 dw 硬件的 csc 模块门时钟。反之 off
cec	on 表示打开了 dw 硬件的 cec 模块门时钟。反之 off
hdcp	on 表示打开了 dw 硬件的 hdcp 模块门时钟。反之 off
mode	表示 DDC 模块的工作模式。
ref clk	表示 DDC 模块的工作参考时钟频率
rate	表示 DDC 模块当前的通讯时钟速率。

### [dw video]

名称	含义
tmids	表示 TMD 工作模式，分为 HDMI 和 DVI 两种模式
avmute	on 表示当前发送 AVMUTE 信号。反之 off
scramble	on 表示当前有开启 scramble。反之 off
vic	表示当前配置输出 VIC Code
timing	表示当前配置输出的分辨率
rate	表示当前配置输出的刷新率
format	表示当前配置输出的颜色格式。 常见的有：RGB、YUV444、YUV422、YUV420
bits	表示当前配置输出的色深位。 常见的有：8bit、10bit、12bit、16bit
pixel_clk	lock 表示当前 pixel 模块锁住 pixel 时钟。 反之 unlock
tmids_clk	lock 表示当前 tmids 模块锁住 tmids 时钟。 反之 unlock

### [dw audio]

名称	含义
freq(Hz)	表示当前配置的音频采样频率
factor	表示当前配置的音频采样系数
n	表示当前配置的 N 值
cts	表示当前配置的 CTS 值
mode	auto 表示当前 CTS 计算模式为硬件自动计算。 user 表示当前 CTS 计算模式为软件计算。
mute	on 表示当前音频处于静音状态。反之 off
fifo	of 表示当前音频 FIFO 正常。 underrun 或 overrun 表示音频 FIFO 出现异常
m_clk	on 表示音频模块时钟打开。反之 off
i2s_clk	lock 表示当前音频 I2S 的传输时钟有锁住。 反之 unlock

**[dw hdcp]**

名称	含义
mode	表示当前 HDCP 是处于 HDMI 还是 DVI 模式加密。
path	表示当前 HDCP 数据流通路是 HDCP1x 还是 HDCP2x

**[dw phy]**

名称	含义
mpll	lock 表示当前 PHY 的 MPLL 有正常锁住。反之 unlock
power	on 表示当前 PHY 的电源有打开。反之 off
rxsense	表示当前检测到 Rxsense 的状态。 每一个 bit 对应一条 lane，0xF 表示四条 lane 都有检测到
hpd	in 表示当前 PHY 硬件有检测到 HPD 信号。反之 out
hpd sense	enable 表示有开启 HPD 检测的功能。反之 disable

**[snps phy]**

名称	含义
mpll	表示当前 pll 配置的参数
drive	表示当前驱动配置的参数

**[top phy]**

名称	含义
ref clk	表示当前 PHY PLL 使用的时钟源频率
output state	on 表示开启了 PHY PLL 的输出。反之 off lock 表示当前 PHY PLL 锁住。反之 off
pll	表示当前配置的 pll 时钟频率

### 4.1.3 查看 Rx 信息

```
cat /sys/class/hdmi/hdmi/attr/hdmi_sink
```

输出信息



```

[sink]
- date      : 2021-01
- name      : SONY TV *30
- version   : 2.0
- dtd timing
  [0]: 3840x2160p
  [1]: 1920x1080p
  [2]: 1280x720p
- svd timing
  svd[0]: vic[97], OnlyYcc420[0], Ycc420[1]
  svd[1]: vic[96], OnlyYcc420[0], Ycc420[1]
  svd[2]: vic[93], OnlyYcc420[0], Ycc420[0]
  svd[3]: vic[94], OnlyYcc420[0], Ycc420[0]
  svd[4]: vic[95], OnlyYcc420[0], Ycc420[0]
  svd[5]: vic[98], OnlyYcc420[0], Ycc420[0]
  svd[6]: vic[31], OnlyYcc420[0], Ycc420[0]
  svd[7]: vic[16], OnlyYcc420[0], Ycc420[0]
  svd[8]: vic[20], OnlyYcc420[0], Ycc420[0]
  svd[9]: vic[05], OnlyYcc420[0], Ycc420[0]
  svd[10]: vic[19], OnlyYcc420[0], Ycc420[0]
  svd[11]: vic[04], OnlyYcc420[0], Ycc420[0]
  svd[12]: vic[32], OnlyYcc420[0], Ycc420[0]
  svd[13]: vic[34], OnlyYcc420[0], Ycc420[0]
  svd[14]: vic[60], OnlyYcc420[0], Ycc420[0]
  svd[15]: vic[62], OnlyYcc420[0], Ycc420[0]
  svd[16]: vic[18], OnlyYcc420[0], Ycc420[0]
  svd[17]: vic[03], OnlyYcc420[0], Ycc420[0]
  svd[18]: vic[17], OnlyYcc420[0], Ycc420[0]
  svd[19]: vic[02], OnlyYcc420[0], Ycc420[0]
  svd[20]: vic[101], OnlyYcc420[0], Ycc420[1]
  svd[21]: vic[102], OnlyYcc420[0], Ycc420[1]
  svd[22]: vic[63], OnlyYcc420[0], Ycc420[0]
  svd[23]: vic[64], OnlyYcc420[0], Ycc420[0]
[hfvsdb]
- scdc      : support
- max rate  : 600Mhz
- dc-10bits
- dc-12bits

[scdc]
- clock ratio : 1/40
- scramble    : set
- scramble state : enable
- clock channel : lock
- data0 channel : lock
- data1 channel : lock
- data2 channel : lock
- hdcp22 capability: support

```

图 4-2: 内核 Rx 信息

[sink]

名称	含义
date	表示这台设备的出厂时间
name	表示这台设备的名称
version	表示这台设备支持的 HDMI 版本
dtd timing	下面罗列的是这台设备支持的 DTD Timing
svd timing	下面罗列的是这台设备支持的 SVD Timing。 VIC: 参考 861 协议, 即可知道对应的分辨率。

名称	含义
	OnlyYcc420: 1 表示这个 VIC 指示的分辨率仅支持了 420 格式。反之为 0 Ycc420: 1 表示这个 VIC 指示的分辨率可以支持 420 格式。反之为 0

**[hfvbdb]**

名称	含义
scdc	support 表示这台设备支持 SCDC 功能。反之为 unsupported
max rate	表示这台设备最大支持的 tmds clock。单位 MHz
dc-10bits	表示这台设备支持 10bit 色深
dc-12bits	表示这台设备支持 12bit 色深

**[scdc]**

名称	含义
clock ratio	1/40 表示当前 Rx 工作在 1/40 模式下。反之工作在 1/10 模式下
scramble	set 表示当前有通过 SCDC 设置 scramble。反之 unset
scramble state	enable 表示当前 Rx 指示其自身工作在 scramble 状态下。反之 disable
clock channel	lock 表示当前 Rx 在时钟通道有锁住时钟。反之 unlock
data0 channel	lock 表示当前 Rx 在数据通道 0 有锁住信号。反之 unlock
data1 channel	lock 表示当前 Rx 在数据通道 1 有锁住信号。反之 unlock
data2 channel	lock 表示当前 Rx 在数据通道 2 有锁住信号。反之 unlock
hdcp22 capability	support 表示当前 Rx 支持 HDCP22。反之 unsupported

**4.1.4 更改输入源**

目前支持 HDMI 的 frame composer 和 Tcon 的两种输入源替换，用于定位是否 DE 送进的数据有异常。

**查看支持输出图片**

```
cat /sys/class/hdmi/hdmi/attr/pattern
```

**设置输出图片**

```
echo {index} > /sys/class/hdmi/hdmi/attr/pattern
```

## 4.1.5 修改日志级别

### 查看当前日志输出级别

```
cat /sys/class/hdmi/hdmi/attr/debug
```

### 设置日志输出级别

```
# 打开日志等级，输出更多的运行配置日志
echo 1 > /sys/class/hdmi/hdmi/attr/debug
# 关闭日志等级，输出默认日志
echo 0 > /sys/class/hdmi/hdmi/attr/debug
```

## 4.1.6 EDID 节点

用于调试替换 EDID 数据

### 替换 EDID 数据

```
cat debug_edid.bin > /sys/class/hdmi/hdmi/attr/edid_data
```

### 使能 EDID 替换

```
echo 1 > /sys/class/hdmi/hdmi/attr/edid_debug
# 注：需要拔插HDMI线生效，后续改进
```

### 关闭 EDID 替换

```
echo 0 > /sys/class/hdmi/hdmi/attr/edid_debug
# 注：需要拔插HDMI线生效，后续改进
```

## 4.1.7 HDCP 节点

### 使能 HDCP

```
echo 1 > /sys/class/hdmi/hdmi/attr/hdcp_enable
```

### 关闭 HDCP

```
echo 0 > /sys/class/hdmi/hdmi/attr/hdcp_enable
```

### 获取加密状态

```
busybox hexdump /sys/class/hdmi/hdmi/attr/hdcp_status
# 0: HDCP关闭
# 1: HDCP加密中
# 2: HDCP加密失败
# 3: HDCP加密成功
```

### 获取加密类型

```
busybox hexdump /sys/class/hdmi/hdmi/attr/hdcp_type
# 0xFF: NULL
# 0x00: HDCP14
# 0x01: HDCP22
```

### 加载固件

```
# 仅用于HDCP22的ESM固件加载
echo 1 > /sys/class/hdmi/hdmi/attr/hdcp_loader
```

## 4.2 Uboot 环境



因 SDK 处于逐步修改升级中，因此下面的一些节点输出信息可能存在差异。

### 4.2.1 查看命令帮助

```
sunxi_hdmi20 help
```

### 4.2.2 查看 Tx 信息

```
sunxi_hdmi20 hdmi_source
```

### 4.2.3 查看 Rx 信息

```
sunxi_hdmi20 hdmi_sink
```

### 4.2.4 更改输入源

```
sunxi_hdmi20 pattern ${index}
# 0: disable pattern output
# 1: enable frame composer pattern output - red
# 2: enable frame composer pattern output - green
# 3: enable frame composer pattern output - blue
# 4: enable tcon colorbar(rgb) pattern output
```

### 4.2.5 寄存器读写

与寄存器读写相关的节点有：reg\_bank，reg\_read，reg\_write



节点名	节点功能描述
reg_bank	用来切换下一次 reg_read 或者 reg_write 操作哪部分的寄存器
reg_read	根据 reg_bank 指定来读对应 offset 寄存器值
reg_write	根据 reg_bank 指定来写对应寄存器值

### 获取当前所支持读写的寄存器区间

```
sunxi_hdmi20 reg_bank
```

### 切换要操作的寄存器区间

```
sunxi_hdmi20 reg_bank ${index}  
# 0: hdmi control register  
# 1: hdmi external phy register  
# 2: hdmi scdc register
```

### 对应区间的寄存器读操作

```
sunxi_hdmi20 reg_read addr count  
# addr: 该区间寄存器的偏移地址。  
# count: 读取寄存器个数，逐步递增
```

### 对应区间的寄存器写操作

```
sunxi_hdmi20 reg_write addr value  
# addr: 该区间寄存器的偏移地址  
# value: 该区间寄存器需要写入的值
```

## 5 FAQ

### 5.1 未检测到 HPD

**StepA:** 使用万用表或者示波器测量 HDMI-HPD 引脚电平

如果 HPD 引脚为高电平，但是 Tx 信息中的 HPD 又为 0。请参考[日志反馈](#)。

如果 HPD 引脚为低电平，测试测量 5V 引脚电平。

**StepB:** 使用万用表或者示波器测量 HDMI-5V 引脚电平

如果 HDMI-5V 为高电平，但是测量 HDMI-HPD 为低电平，说明是 Rx 设备异常拉低 HPD 电压。请更换其他显示设备测试。

如果 HDMI-5V 为低电平。说明硬件电路存在问题没有拉高 5V，请找硬件负责人分析硬件电路。

### 5.2 测量 TMDS Clock

**StepA:** 使用示波器测量 TMDS Clock 引脚的时钟频率是否符合预期。

如果未检测到时钟频率或者时钟不符合预期。请参考[日志反馈](#)。

如果有检测到 TMDS Clock 并且时钟频率符合预期。说明 Tx 端是正常输出，请更换其他显示设备测试。

如果上述流程无法满足你的问题状况，请参考[日志反馈](#)。

### 5.3 Uboot 平台

#### 5.3.1 Q1: 无信号问题

 说明

请先根据实际确认当前是黑屏还是无信号，这是两种不同现象！请不要混淆！

**Step1:** 先确认 Uboot HDMI 是否正常加载。

如果 uboot 加载过程的日志中：

```
[01.178]drm hdmi get mode: 3840x2160@60Hz
hdmi@5520000: detailed mode clock 594000 kHz, flags[5]
H: 3840 4016 4104 4400
V: 2160 2168 2178 2250
```

说明 DRM 框架有获取到 HDMI 上报的分辨率信息，说明当前 Uboot HDMI 是正常加载的。

### Step2: 打印 HDMI 的调试信息。

```
sunxi_hdmi20 hdmi_source
```

参考[查看 Tx 信息](#)章节，查看当前配置是否有正常开启。

如果 PLL 出现 unlock，说明模块工作 PLL 异常。请参考[日志反馈](#)。

如果出现 HPD 为 0 或者 undetect 等信息，说明 HPD 没有检测到。因此需要排查硬件以及外部电路。请参看[未检测到 HPD](#)

### Step3: 测量 TMDS Clock

参考[测量 TMDS Clock](#)章节

如果上述步骤不能解决你的问题，请参考[日志反馈](#)。

## 5.3.2 Q2: 黑屏问题

#### 说明

请先根据实际确认当前是黑屏还是无信号，这是两种不同现象！请不要混淆！

### Step1: 更改为 Tcon 数据源输出

```
sunxi_hdmi20 pattern 4
```

如果画面替换为红绿蓝白的条形图，说明 HDMI 模块是正常显示，只是 DE 送进来的数据是黑屏。请找 DE 模块处理。

如果显示还是黑屏，那么说明问题出在 Tcon -> HDMI 模块。

### Step2: 更改为 HDMI 内部 FC 数据源输出

```
sunxi_hdmi20 pattern 1 # 替换为纯红色图片
sunxi_hdmi20 pattern 2 # 替换为纯绿色图片
sunxi_hdmi20 pattern 3 # 替换为纯蓝色图片
```

如果画面能正常替换为对应的纯色图片，说明问题在 Tcon -> HDMI 端。请参考[日志反馈](#)。

如果画面还是黑屏，说明大概率黑屏是 Rx 显示设备自身引起的。请更换其他显示设备测试。

如果上述步骤不能解决你的问题，请参考[日志反馈](#)。

## 5.4 Kernel 平台

### 5.4.1 Q1：无信号问题

#### Step1：确认 HDMI 模块有加载

```
/sys/class/hdmi/hdmi/attr
```

如果上面路径不存在，说明 HDMI 模块没有加载，请参考[配置 HDMI](#)完成模块的使能。如果确认参考该流程还是未加载成功。请参考[日志反馈](#)

#### Step2：查看 DRM 有配置 HDMI 输出

参考[查看 Tx 信息](#)章节，查看 DRM 节点下的 enable、mode set、mode info 信息。

如果出现没有配置模式或者使能，说明是 DRM 框架设定问题。请参考[日志反馈](#)

#### Step3：查看 Tx 信息的状态

参考[查看 Tx 信息](#)章节。

如果出现 PHY PLL 为 unlock、output 为 disable、clock 为 off、hpd sense 为 disable 等情况，请参考[日志反馈](#)

如果出现 HPD 为 0 或者 undetect 等信息，说明 HPD 没有检测到。因此需要排查硬件以及外部电路。请参考[未检测到 HPD](#)

#### Step4：测量 TMDS Clock

参考[测量 TMDS Clock](#)章节

#### Step5：屏蔽 HPD 检测替换其他显示设备



此步骤仅支持 340M 以下的分辨率使用

屏蔽 HPD 热拔插检测

```
su
echo 0x1000 > /sys/class/hdmi/hdmi/attr/hpd_mask
```

拔掉 HDMI 线接入其他显示设备。

如果其他显示设备正常显示，说明是原显示设备异常。

如果其他显示设备也无法显示 (前提是该显示设备支持这个分辨率)，说明是输出有异常。请参考[日志反馈](#)

如果上述步骤不能解决你的问题，请参考[日志反馈](#)。

## 5.4.2 Q2：黑屏问题



说明

请先根据实际确认当前是黑屏还是无信号，这是两种不同现象！请不要混淆！

### Step1：更改为 Tcon 数据源输出

```
su
echo 4 > /sys/class/hdmi/hdmi/attr/pattern
```

如果画面替换为红绿蓝白的条形图，说明 HDMI 模块是正常显示，只是 DE 送进来的数据是黑屏。请找 DE 模块处理。

如果显示还是黑屏，那么说明问题出在 Tcon -> HDMI 模块。

### Step2：更改为 HDMI 内部 FC 数据源输出

```
echo 1 > /sys/class/hdmi/hdmi/attr/pattern # 替换为纯红色图片
echo 2 > /sys/class/hdmi/hdmi/attr/pattern # 替换为纯绿色图片
echo 3 > /sys/class/hdmi/hdmi/attr/pattern # 替换为纯蓝色图片
```

如果画面能正常替换为对应的纯色图片，说明问题在 Tcon -> HDMI 端。请参考[日志反馈](#)。

如果画面还是黑屏，说明大概率黑屏是 Rx 显示设备自身引起的。请更换其他显示设备测试。

如果上述步骤不能解决你的问题，请参考[日志反馈](#)。

## 5.4.3 Q3：无声问题

### Step1：查看 HDMI 当前输出模式

通过[查看 Tx 信息](#)章节，查看 [dw video] 下面的 tmds 对应值。如果是 DVI。说明当前配置的是 DVI 模式，说明设备仅支持 DVI。

可以使用竞品进行对比，如果竞品能出声。请参考[日志反馈](#)。

### Step2：查看音频模块门时钟是否开启

通过[查看 Tx 信息](#)章节，查看 [dw ctrl] 下的 clock domain 对应的 audio 状态。

如果为 off 说明 HDMI 内部的 audio 模块未开启。请参考[日志反馈](#)。

### Step3：查看是否处于静音

通过[查看 Tx 信息](#)章节，查看 [dw audio] 下的 mute 对应值。如果是 on，说明当前处于静音。请参考[日志反馈](#)。

### Step4：查看 N 和 CTS 是否正常

通过[查看 Tx 信息](#)章节，查看 [dw audio] 下的 n 和 cts。如果有一项值为 0，说明配置异常。请参考[日志反馈](#)。

如果都有值，请根据公式计算是否合理： $128 * fs = TMDS\_clock * N / CTS$

注：Fs、Tmds\_clk、N 和 CTS 都是可以通过[查看 Tx 信息](#)章节得知，但是需要注意计算时 Fs 和 TMDS\_Clock 都为 Hz。

如果发现计算异常不匹配，请参考[日志反馈](#)。

#### Step5: 查看 FIFO 是否处于异常状态

通过[查看 Tx 信息](#)章节，查看 [dw audio] 下的 FIFO，如果出现 underrun 或者 overrun。请联系对应的音频模块负责人分析处理。

如果上述步骤不能解决你的问题，请参考[日志反馈](#)。



## 6 日志反馈

HDMI 的日志反馈需包含：**异常的现象视频、现象视频对应的日志、对应设备的 EDID 以及问题的复现步骤**

**Step1:** 复现问题之前。开始抓取串口日志并在串口输入

```
su
dmesg -n8
echo 1 > /sys/class/hdmi/hdmi/attr/debug
```

**Step2:** 复现到问题，在串口输入

```
cat /sys/class/hdmi/hdmi/attr/hdmi_source
cat /sys/class/hdmi/hdmi/attr/hdmi_source
cat /sys/class/hdmi/hdmi/attr/hdmi_sink
cat /sys/class/hdmi/hdmi/attr/hdmi_sink
cat /sys/class/hdmi/hdmi/attr/hpd_mask
busybox hexdump /sys/class/hdmi/hdmi/attr/hdcp_type
busybox hexdump /sys/class/hdmi/hdmi/attr/hdcp_status
echo 0 > /sys/class/hdmi/hdmi/attr/debug
echo 0x100,0xff > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x200,0x8 > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x800,0x8 > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x1000,0x220 > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x3000,0x39 > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x3100,0x5 > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x3200,0x8 > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x4000,0xb > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x4100,0x2 > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x5000,0x2a > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x7800,0x19 > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x7900,0xf > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x7d00,0x32 > /sys/class/hdmi/hdmi/attr/reg_read
echo 0x7e00,0x32 > /sys/class/hdmi/hdmi/attr/reg_read
```

**Step3:** 结束串口日志抓取

**Step4:** 获取当前 EDID 数据

```
adb devices
adb root
adb pull sys/class/drm/cardx-HDMI-A-1/edid xxx
```

### 📖 说明

- 1、cardx-HDMI-A-1 中，cardx 为临时，实际要根据这个 x 的值来修改，比如 card0-HDMI-A-1。
- 2、xxx 为 PC 上指定存放路径。该文件名为 edid 且无后缀！

示例：

```
adb pull sys/class/drm/card0-HDMI-A-1/edid . # 表示我将card0的HDMI的edid节点pull到cmd运行的当前路径
```

## 7 附录 1：HDCP1x 密钥烧录

HDCP1x 需要烧录指定的 HDCP1x Key，目前市面上常用的 HDCP1x 版本其实就是 HDCP1.4 版本。

因此我们需要参考下面的方式来完成 HDCP14 的密钥烧录。

### 7.1 密钥配置

#### Step1：密钥获取

HDCP14 密钥一般从 DCP 组织购买，由 DCP 组织派发出来的密钥是一个密钥集<sup>®</sup>，由成千上万组原始密钥组成，由具体购买的密钥数量决定。

#### Step2：密钥切割

为了保证密钥的安全性，全志推出 HDCPLoad 工具对原始密钥组进行切割并加密出每一把密钥，加密出来的每一把密钥的大小为 **368 字节**。

#### 工具用法

HDCPLoad 一个 HDCP 密钥加载、切割工具。推荐使用 HDCPLoad\_20221027 版本后的工具，可联系对接的 AE 获取。

#### HDCPLoad 参数解析：

参数	含义
count	显示原始 key 中包含的子 key 个数
prefix	指定输出文件名，例如：hdcp1p4，就会生成 hdcp1p4_1.bin，hdcp1p4_2.bin 等
range	指定分割范围，例如：1 2，表示分割第一个到第二个的 key

### 7.2 密钥烧录

打开 DrangonSN，如下如。点击《配置 key》，再点击确定。



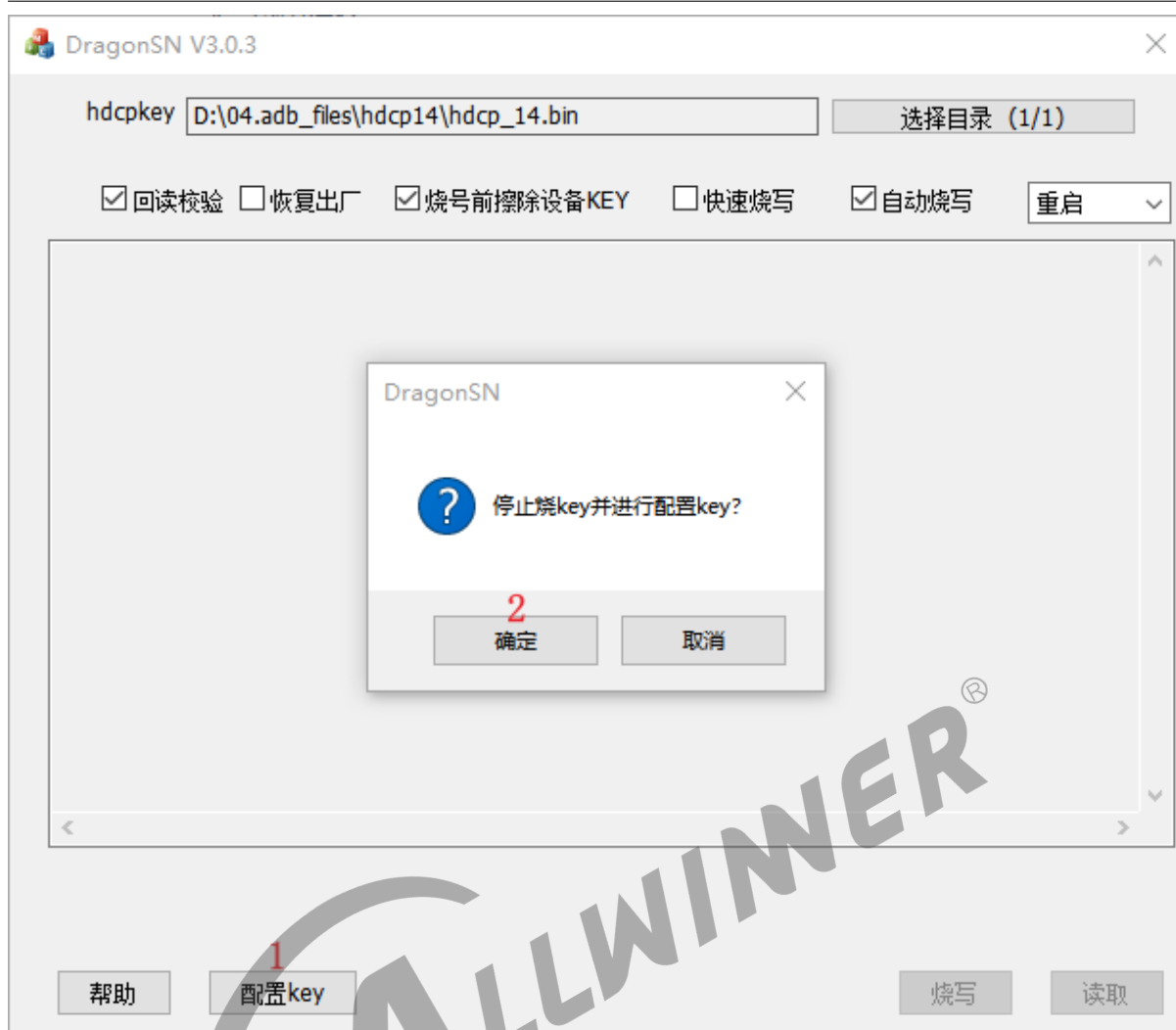


图 7-1: HDCP1x 密钥烧录配置-1

此时弹出 DragonSN Config Tool，点击《添加》，再参考 4 的信息进行填写。点击《保存》后再点击《确定》



图 7-2: HDCP1x 密钥烧录配置-2

注：hdckey 的配置信息一定要参考上图保持一致！！

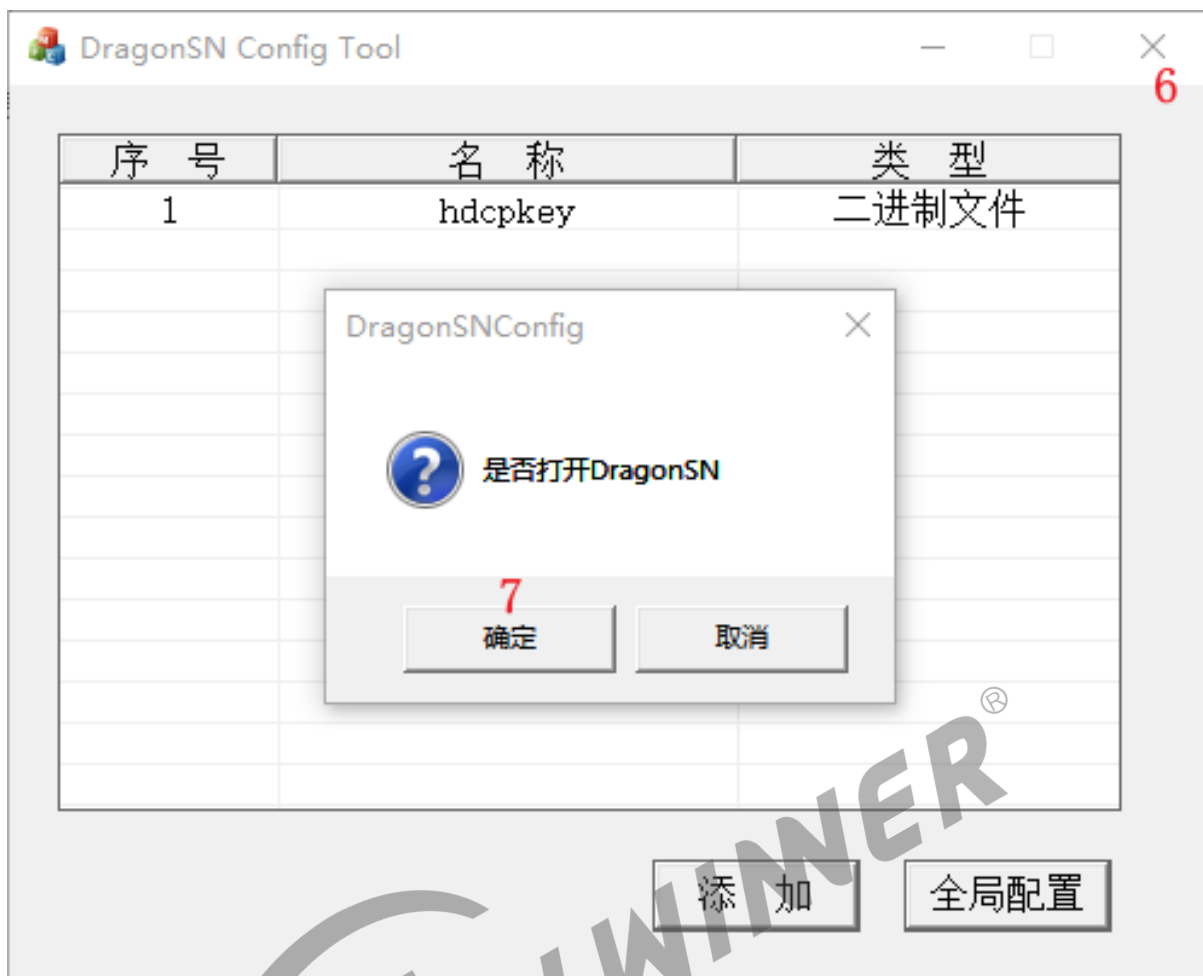


图 7-3: HDCP1x 密钥烧录配置-3

保存后如上，会增加序号 1 的 hdcpkey，类型是二进制文件。确认上述无误后点击《X》后确定打开 DragonSN。

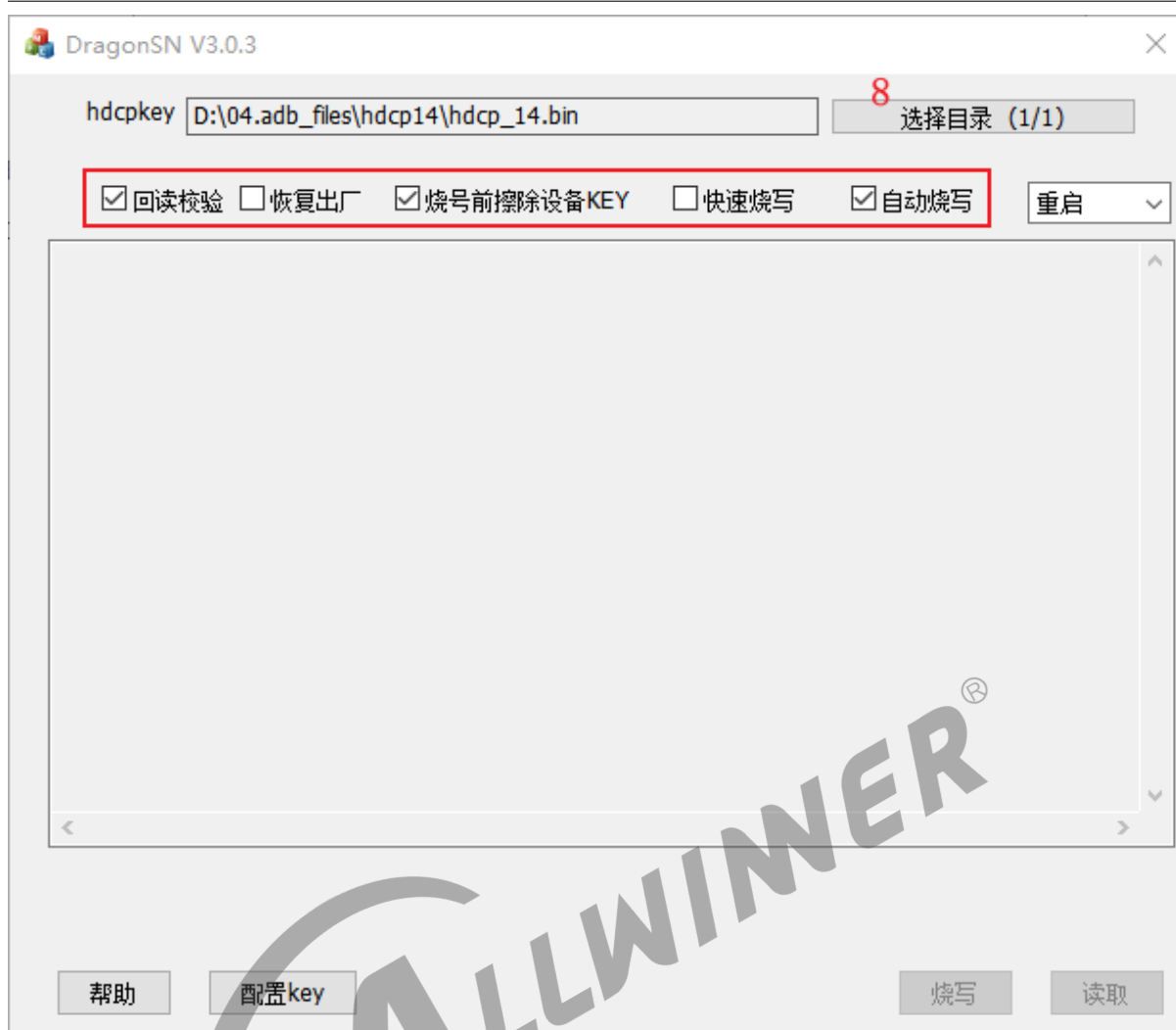


图 7-4: HDCP1x 密钥烧录配置-4

点击选择目录，选择存放 HDCP14 密钥的文件路径。并按照上面的配置进行勾选。至此，HDCP 密钥烧录工具配置完成。

## 2.2 工具连接

将板子和 PC 进行 ADB 连接，板子上电开机后会自动烧录。

## 8 附录 2：HDCP2x 密钥烧录

### 8.1 密钥配置

#### Step1: 原始密钥获取

从 DCP 组织购买 HDCP2.2 Tx 密钥，这份原始密钥的大小为 445 字节。暂命名：HDCP2\_TX\_KEY.bin

#### Step2: 生成 fex 加密密钥

加密密钥要求是一串长度为 16 bytes 的随机数，这个值可按照自己需要去生成，但一旦确定下来要注意**一定要保存好**！。暂命名：hdcp2pkf.bin

比如可以使用 Linux 命令来直接生成：

```
dd if=/dev/random of=hdcp2pkf.bin bs=16 count=1
```

#### Step3: 生成 fex 固件

请找相关的 AE 同事获取生成 esm.fex SDK

esm sdk 的文件层次如下：

```
esm_sdk:
├── build.sh  ---编译脚本
├── esm.fex   ---最终生成的文件
├── esmtool
├── firmware
│   ├── esm_config.i
│   └── firmware.rom
├── README.md
├── utils
│   ├── aictool
│   ├── esm_swap
│   ├── hdcpkeys
│   └── sample.aic
├── vendor
│   ├── hdcp2pkf.bin  ---查看Step2描述
│   ├── HDCP2_TX_KEY.bin  ---查看Step1描述
│   └── hdcp_keys.le  --- (脚本执行过程中生成，无需关注)
```

### 8.2 密钥烧录

#### Step1: 配置 DragonSN

**Step1.1:** 打开 DragonSN，点击左下角的配置 key

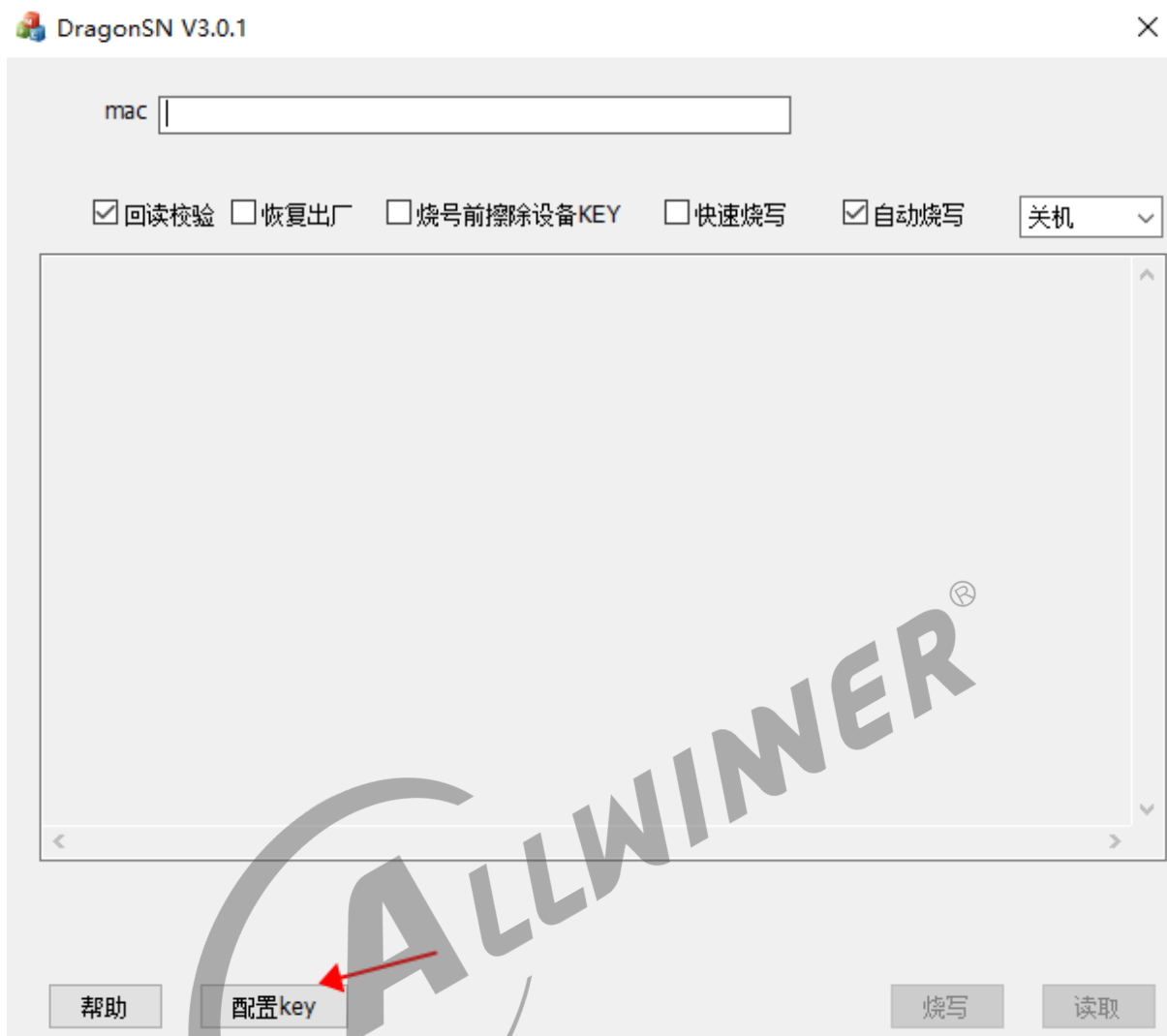


图 8-1: HDCP2x 密钥烧录配置-1

**Step1.2:** 如果当前配置有其他的配置项，请先**全部删除**，避免互相影响，右击，删除；

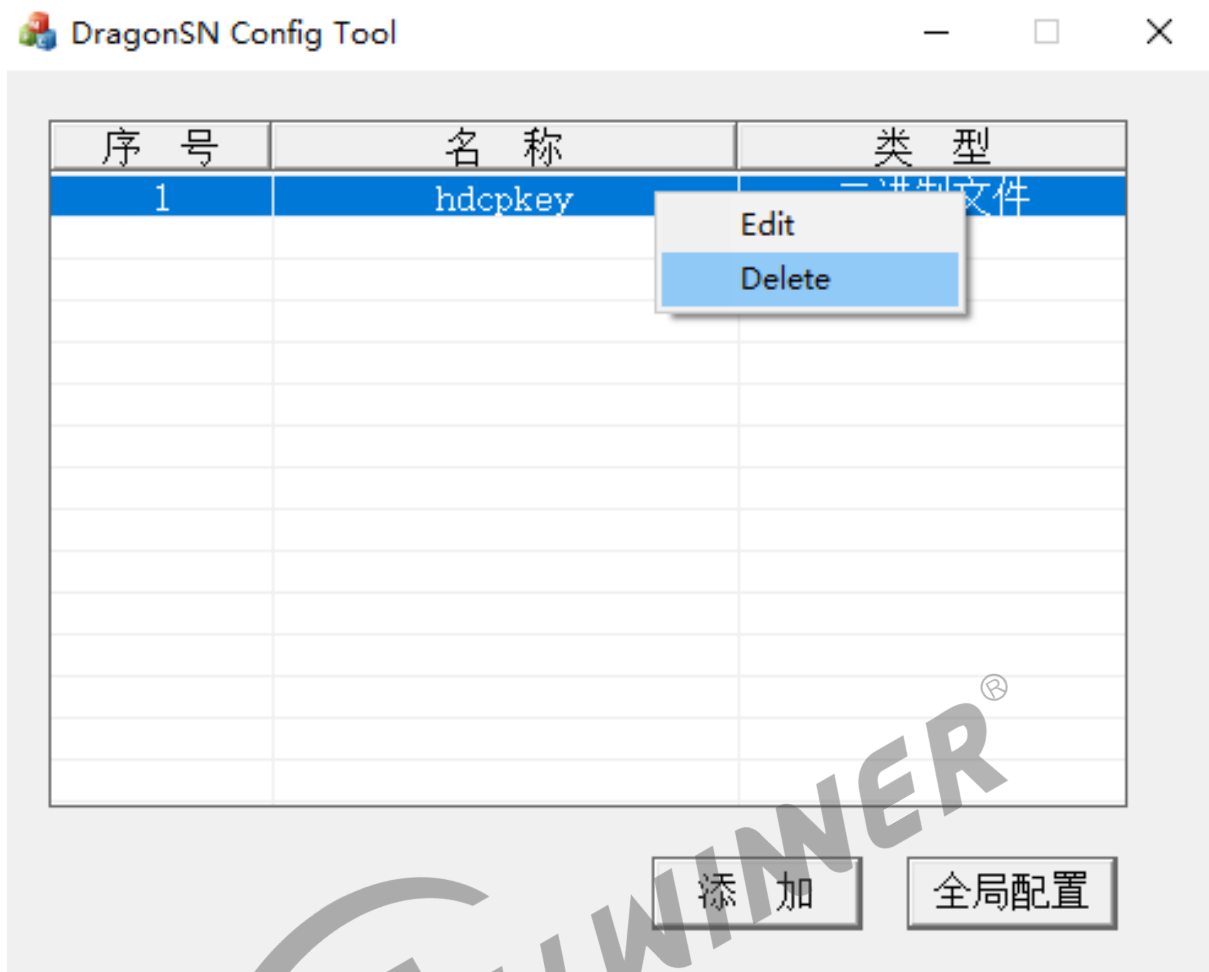


图 8-2: HDCP2x 密钥烧录配置-2

**Step1.3:** 点击下方的添加，按下图所示配置，配置完成点击保存；



图 8-3: HDCP2x 密钥烧录配置-3

关于 **烧完后**配置的解释：

1. 保留，一直烧同一个 key。
2. 移到 \_used 目录，每烧完一个 key 就会将该文件移到同路径下的 \_used 后缀文件夹中，即：依次烧不同的 key。

由于 HDCP 2.2 可以多个机器共用同一个 key，而且需要烧录的 pkf 也可以多台共用，所以选择 **保留**即可。

**Step1.4：** 点击 Config Tool 界面右上角的 X，退出到 DragonSN 软件界面。

**Step2：烧录 hdcp2pkf.bin**



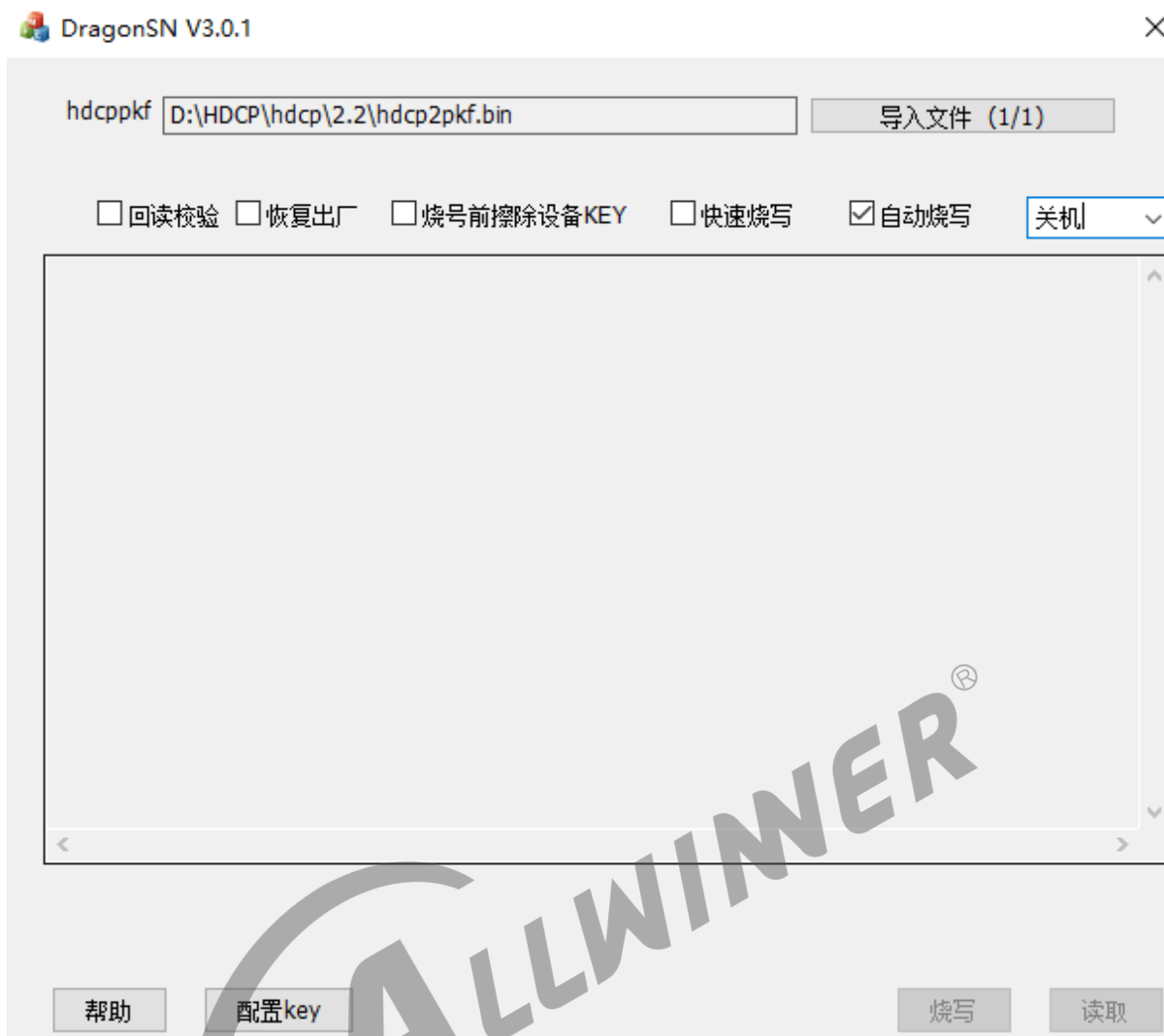


图 8-4: HDCP2x 密钥烧录配置-4

**Step2.1:** 点击导入文件，选择要烧录的 **hdc2pkf.bin** 所在文件夹（尽量避免中文路径）；

**Step2.2:** 勾选自动烧写，取消勾选快速烧写，选择烧写结束后 关机；

**Step2.3:** 确认开发板与 PC 已用 USB 线连接；

**Step2.4:** 重启开发板，工具会自动烧录；

**Step2.5:** 烧录完成后，可点击 读取确认烧录成功；

### Step3: 加载 esm.fex

**Step3.1:** 将 esm.fex 导入小机端，并且文件名一定为 **esm.fex**。小机端路径：

/data/hdcp22/esm.fex

**Step3.2:** 配置 firmware 加载路径。

注：只需将 esm.fex 存放的上级路径导入

```
# esm.fex路径: /data/hdcp22/esm.fex  
echo /data/hdcp22 > /sys/module/firmware_class/parameters/path
```

### Step3.3: 加载 esm.fex 固件

```
echo 1 > /sys/class/hdmi/hdmi/attr/hdcp_loader
```

### Step3.4: 打开 HDCP 功能

```
echo 1 > /sys/class/hdmi/hdmi/attr/hdcp_enable
```



## 9 附录 3：Linux CEC 应用程序样例

### ⚠ 注意

以下测试代码仅适用 DRM 框架下的 CEC 功能开发。

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <poll.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <string.h>
#include <sys/time.h>

#include <linux/cec.h>

#define CEC_PATH "/dev/cec0"

static void app_cec_help(void)
{
    printf("[help] \n");
    printf("\t argc[1] = log_addr: set default cec logical address.\n");
    printf("\t argc[1] = standby: playback standby -> tv standby.\n");
    printf("\t argc[1] = wakeup: playback wakeup -> tv wakeup.\n");
    printf("\t argc[1] = tv_standby: tv standby -> playback standby.\n");
    printf("\t argc[1] = get_language: get tv menu lanuage.\n");
    printf("demo: app_cec log_addr\n");
}

/**
 * @desc: app test case for cec get tv language (RX cases)
 * @return: 0 - success
 *         -1 - faield
 */
static int app_cec_get_language(void)
{
    int ret = 0;
    int cec_fd = 0;
    struct cec_log_addrs log_addrs;
    struct cec_msg tx_msg;
    struct cec_msg rx_msg;
    int timeout = 10;
    struct pollfd fds[1];
    int i;
    unsigned int mode = 0;
```

```
memset(&log_addrs, 0, sizeof(struct cec_log_addrs));
memset(&tx_msg, 0, sizeof(struct cec_msg));

/* 1. open cec device fd */
cec_fd = open(CEC_PATH, O_RDWR);
if (cec_fd <= 0)
{
    printf("open %s is failed!!!\n", CEC_PATH);
    ret = -1;
    goto exit;
}

/* 2. set mode */
mode = CEC_MODE_INITIATOR | CEC_MODE_EXCL_FOLLOWER_PASSTHRU;
ret = ioctl(cec_fd, CEC_S_MODE, &mode);
if (ret != 0)
{
    printf("app cec test set mode failed!!!\n");
    ret = -1;
    goto exit;
}

/* 3. get cec logical address */
ret = ioctl(cec_fd, CEC_ADAP_G_LOG_ADDRS, &log_addrs);
if (ret < 0 || log_addrs.num_log_addrs < 1)
{
    printf("app cec test get logical address failed!!!\n");
    ret = -1;
    goto exit_1;
}

/* 4. send image-view-on message */
tx_msg.msg[0] = (log_addrs.log_addr[0] << 4) | 0x0; /* TV addr: 0x0 */
tx_msg.msg[1] = CEC_MSG_GET_MENU_LANGUAGE;
tx_msg.len = 2;

ret = ioctl(cec_fd, CEC_TRANSMIT, &tx_msg);
if (ret < 0 || !(tx_msg.tx_status & CEC_TX_STATUS_OK))
{
    printf("app cec test send get-menu-language failed!!!\n");
    ret = -1;
    goto exit_1;
}

/* 5. wait for tv reply set-menu-language */
fds[0].fd = cec_fd;
fds[0].events = POLLIN;

while (timeout--)
{
    int ret = poll(fds, 1, 1000);

    if (ret < 0)
    {
        printf("app cec test poll failed!!!\n");
        ret = -1;
        goto exit_1;
    }
    else if (ret > 0)
```

```
{
    memset(&rx_msg, 0, sizeof(struct cec_msg));

    ret = ioctl(cec_fd, CEC_RECEIVE, &rx_msg);
    if (ret < 0)
    {
        printf("app cec test receive msg failed!!!\n");
        ret = -1;
        goto exit_1;
    }

    if (rx_msg.msg[1] == CEC_MSG_SET_MENU_LANGUAGE)
    {
        printf("TV menu lanuage: ");
        for (i = 2; i < rx_msg.len; i++)
        {
            printf("%c", rx_msg.msg[i]);
        }
        printf("\n");
        ret = 0;
        goto exit_1;
    }
    printf("[%d] wait for tv language...\n", timeout);
}

ret = -1;
exit_1:
close(cec_fd);
exit:
return ret;
}

/**
 * @desc: app test case for cec receive TV standby msg (RX cases)
 * @return: 0 - success
 *         -1 - failed
 */
static int app_cec_tv_standby(void)
{
    int ret = 0;
    int cec_fd = 0;
    struct cec_log_addrs log_addrs;
    struct cec_msg tx_msg;
    struct cec_msg rx_msg;
    int timeout = 10;
    struct pollfd fds[1];
    unsigned int mode = 0;

    memset(&log_addrs, 0, sizeof(struct cec_log_addrs));
    memset(&tx_msg, 0, sizeof(struct cec_msg));

    /* 1. open cec device fd */
    cec_fd = open(CEC_PATH, O_RDWR);
    if (cec_fd <= 0)
    {
        printf("open %s is failed!!!\n", CEC_PATH);
        ret = -1;
        goto exit;
    }
}
```

```
/* 2. set mode */
mode = CEC_MODE_INITIATOR | CEC_MODE_EXCL_FOLLOWER_PASSTHRU;
ret = ioctl(cec_fd, CEC_S_MODE, &mode);
if (ret != 0)
{
    printf("app cec test set mode failed!!!\n");
    ret = -1;
    goto exit;
}

/* 3. wait for tv standby msg */
fds[0].fd = cec_fd;
fds[0].events = POLLIN;

while (timeout--)
{
    int ret = poll(fds, 1, 1000);

    if (ret < 0)
    {
        printf("app cec test poll failed!!!\n");
        ret = -1;
        goto exit_1;
    }
    else if (ret > 0)
    {
        memset(&rx_msg, 0, sizeof(struct cec_msg));

        ret = ioctl(cec_fd, CEC_RECEIVE, &rx_msg);
        if (ret < 0)
        {
            printf("app cec test receive msg failed!!!\n");
            ret = -1;
            goto exit_1;
        }

        if (rx_msg.msg[1] == CEC_MSG_STANDBY)
        {
            /* TV standby -> playback standby!!! */
            ret = 0;
            goto exit_1;
        }
    }
    printf("[%d] wait for tv standby...\n", timeout);
}

ret = -1;
exit_1:
close(cec_fd);
exit:
return ret;
}

/**
 * @desc: app test case for cec wakeup (TX cases)
 * @return: 0 - success
 *         -1 - failed
 */
static int app_cec_wakeup(void)
```

```
{
int ret = 0;
int cec_fd = 0;
struct cec_log_addrs log_addrs;
struct cec_msg tx_msg;
unsigned short phys_addr = 0;

memset(&log_addrs, 0, sizeof(struct cec_log_addrs));
memset(&tx_msg, 0, sizeof(struct cec_msg));

/* 1. open cec device fd */
cec_fd = open(CEC_PATH, O_RDWR);
if (cec_fd <= 0)
{
printf("open %s is failed!!!\n", CEC_PATH);
ret = -1;
goto exit;
}

/* 2. get cec logical address */
ret = ioctl(cec_fd, CEC_ADAP_G_LOG_ADDRS, &log_addrs);
if (ret < 0 || log_addrs.num_log_addrs < 1)
{
printf("app cec test get logical address failed!!!\n");
ret = -1;
goto exit_1;
}

/* 3. send image-view-on message */
tx_msg.msg[0] = (log_addrs.log_addr[0] << 4) | 0x0; /* TV addr: 0x0 */
tx_msg.msg[1] = CEC_MSG_IMAGE_VIEW_ON;
tx_msg.len = 2;

ret = ioctl(cec_fd, CEC_TRANSMIT, &tx_msg);
if (ret < 0 || !(tx_msg.tx_status & CEC_TX_STATUS_OK))
{
printf("app cec test send image-view-on failed!!!\n");
ret = -1;
goto exit_1;
}

/* 4. get phys address */
ret = ioctl(cec_fd, CEC_ADAP_G_PHYS_ADDR, &phys_addr);
if (ret < 0 || phys_addr == 0xFFFF)
{
printf("app cec test get phys address failed!!!\n");
ret = -1;
goto exit_1;
}

/* 5. send active-source message */
memset(&tx_msg, 0, sizeof(struct cec_msg));
tx_msg.msg[0] = (log_addrs.log_addr[0] << 4) | 0xf; /* broadcast */
tx_msg.msg[1] = CEC_MSG_ACTIVE_SOURCE;
tx_msg.msg[2] = phys_addr >> 8;
tx_msg.msg[3] = phys_addr & 0xff;
tx_msg.len = 4;

ret = ioctl(cec_fd, CEC_TRANSMIT, &tx_msg);
if (ret < 0 || !(tx_msg.tx_status & CEC_TX_STATUS_OK))
```

```
{
printf("app cec test send active-source failed!!!\n");
ret = -1;
goto exit_1;
}

ret = 0;
exit_1:
close(cec_fd);
exit:
return ret;
}

/**
 * @desc: app test case for cec standby (TX cases)
 * @return: 0 - success
 *         -1 - failed
 */
static int app_cec_standby(void)
{
int ret = 0;
int cec_fd = 0;
struct cec_log_addrs log_addrs;
struct cec_msg tx_msg;

memset(&log_addrs, 0, sizeof(struct cec_log_addrs));
memset(&tx_msg, 0, sizeof(struct cec_msg));

/* 1. open cec device fd */
cec_fd = open(CEC_PATH, O_RDWR);
if (cec_fd <= 0)
{
printf("open %s is failed!!!\n", CEC_PATH);
ret = -1;
goto exit;
}

/* 2. get cec logical address */
ret = ioctl(cec_fd, CEC_ADAP_G_LOG_ADDRS, &log_addrs);
if (ret < 0 || log_addrs.num_log_addrs < 1)
{
printf("app cec test get logical address failed!!!\n");
ret = -1;
goto exit_1;
}

/* 3. send standby message */
tx_msg.msg[0] = (log_addrs.log_addr[0] << 4) | 0x0; /* TV addr: 0x0 */
tx_msg.msg[1] = CEC_MSG_STANDBY;
tx_msg.len = 2;

ret = ioctl(cec_fd, CEC_TRANSMIT, &tx_msg);
if (ret < 0 || !(tx_msg.tx_status & CEC_TX_STATUS_OK))
{
printf("app cec test send standby failed!!!\n");
ret = -1;
goto exit_1;
}

ret = 0;
```



```
exit_1:
    close(cec_fd);
exit:
    return ret;
}

/**
 * @desc: app test case for set logical address
 * @return: 1 - warning, need check
 *          0 - success
 *          -1 - failed
 */
static int app_cec_get_logical_addr(void)
{
    int ret = 0;
    int cec_fd = 0;
    struct cec_log_addrs log_addrs = {0};

    /* 1. open cec device fd */
    cec_fd = open(CEC_PATH, O_RDWR);
    if (cec_fd <= 0)
    {
        printf("open %s is failed!!!\n", CEC_PATH);
        ret = -1;
        goto exit;
    }

    /* 2. get logical address */
    ret = ioctl(cec_fd, CEC_ADAP_G_LOG_ADDRS, &log_addrs);
    if (ret < 0)
    {
        printf("app cec test get logical address failed!!!\n");
        ret = -1;
        goto exit_1;
    }

    if ((log_addrs.num_log_addrs > 0) &&
        ((log_addrs.log_addr[0] == CEC_LOG_ADDR_PLAYBACK_1) ||
         (log_addrs.log_addr[0] == CEC_LOG_ADDR_PLAYBACK_2) ||
         (log_addrs.log_addr[0] == CEC_LOG_ADDR_PLAYBACK_3)))
    {
        /* The logical address has been set, return PASS. */
        ret = 0;
        goto exit_1;
    }

    ret = -1;
exit_1:
    close(cec_fd);
exit:
    return ret;
}

static int cec_init(void)
{
    int ret = 0;
    int cec_fd = 0;
    struct cec_log_addrs log_addrs = {0};

    /* 1. open cec device fd */
```

```
cec_fd = open(CEC_PATH, O_RDWR);
if (cec_fd <= 0)
{
    printf("open %s is failed!!!\n", CEC_PATH);
    ret = -1;
    goto exit;
}

/* 2. get logical address */
ret = ioctl(cec_fd, CEC_ADAP_G_LOG_ADDRS, &log_addrs);
if (ret < 0)
{
    printf("app cec test get logical address failed!!!\n");
    ret = -1;
    goto exit_1;
}

if ((log_addrs.num_log_addrs > 0) &&
    ((log_addrs.log_addr[0] == CEC_LOG_ADDR_PLAYBACK_1) ||
     (log_addrs.log_addr[0] == CEC_LOG_ADDR_PLAYBACK_2) ||
     (log_addrs.log_addr[0] == CEC_LOG_ADDR_PLAYBACK_3)))
{
    /* The logical address has been set, return PASS. */
    ret = 0;
    goto exit_1;
}

memset(&log_addrs, 0, sizeof(struct cec_log_addrs));

/* 3. try to set default logical address */
log_addrs.cec_version = CEC_OP_CEC_VERSION_1_4;
log_addrs.flags = CEC_LOG_ADDRS_FL_ALLOW_UNREG_FALLBACK;
log_addrs.num_log_addrs = 1;

log_addrs.log_addr[0] = CEC_LOG_ADDR_PLAYBACK_1;
log_addrs.primary_device_type[0] = CEC_OP_PRIM_DEVTYPE_PLAYBACK;
log_addrs.log_addr_type[0] = CEC_LOG_ADDR_TYPE_PLAYBACK;
log_addrs.all_device_types[0] = CEC_OP_ALL_DEVTYPE_PLAYBACK;

ret = ioctl(cec_fd, CEC_ADAP_S_LOG_ADDRS, &log_addrs);
if (ret != 0)
{
    printf("app cec test set logical address failed!!!\n");
    ret = -1;
    goto exit;
}

ret = 0;
exit_1:
close(cec_fd);
exit:
return ret;
}

/**
 * @desc: main function
 * @return: 0 - success
 *         -1 - failed
 */
int main(int argc, char *argv[])
```

```
{
if (argc < 2)
{
app_cec_help();
return -1;
}

if (cec_init() < 0)
{
printf("cec init failed!!!\n");
return -1;
}

if (0 == strcmp("log_addr", argv[1]))
{
return app_cec_get_logical_addr();
}
else if (0 == strcmp("standby", argv[1]))
{
return app_cec_standby();
}
else if (0 == strcmp("wakeup", argv[1]))
{
return app_cec_wakeup();
}
else if (0 == strcmp("tv_standby", argv[1]))
{
return app_cec_tv_standby();
}
else if (0 == strcmp("get_language", argv[1]))
{
return app_cec_get_language();
}
else
{
printf("input param failed, check help info!\n");
}

app_cec_help();
return -1;
}
```

## 10 附录 4：Linux HDCP 应用程序样例

### HdcpManager.cpp

```
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>

#include "debug.h"
#include "HdcpManager.h"

#define DEVPATH_HDMI "/dev/hdmi"
#define FIRMWARE_PATH "/vendor/etc/hdcp/esm.fex"
#define SYSPATH_HDCP_ENABLE "/sys/devices/virtual/hdmi/hdmi/attr/hdcp_enable"
#define SYSPATH_HDCP_STATUS "/sys/devices/virtual/hdmi/hdmi/attr/hdcp_status"
#define SYSPATH_HDCP_TYPE "/sys/devices/virtual/hdmi/hdmi/attr/hdcp_type"

#define AW_IOCTL_HDMI_HDCP22_LOAD_FW 1

HdcpManager::HdcpManager()
    : mFirmwareReady(false), mHdcpEnabled(false)
{
}

static int read_from_file(const char *path, char *buf, size_t size)
{
    int fd = open(path, O_RDONLY, 0);
    if (fd == -1) {
        dd_error("Could not open '%s', %s(%d)", path, strerror(errno), errno);
        return -errno;
    }
    ssize_t count = read(fd, buf, size);
    close(fd);
    return count;
}

static int write_to_file(const char *path, const char *buffer, int i) {
    int fd = open(path, O_WRONLY, 0);
    if (fd == -1) {
        dd_error("Could not open '%s', %s(%d)", path, strerror(errno), errno);
        return -1;
    }
    write(fd, buffer, i);
    close(fd);
    return 0;
}

int HdcpManager::configHdcp(bool enable)
{
    int ret = 0;
```

```

    if (enable && !mFirmwareReady) {
        /* load HDCP2.2 firmware: esm.fex */
        loadFirmware(FIRMWARE_PATH);
    }

    if (!write_to_file(SYSPATH_HDCP_ENABLE, enable ? "1":"0", 1)) {
        mHdcpEnabled = enable;
        return 0;
    }

    return -1;
}

HdcpManager::HdcpLevel HdcpManager::getConnectedHdcpLevel() const
{
    char type = 0;

    /*
     * HDCP level:
     *   HDCP_V1 : HDCP1.4
     *   HDCP_V2_2: HDCP2.2
     */
    if (read_from_file(SYSPATH_HDCP_TYPE, &type, 1) == 1) {
        if (type == 0)
            return HDCP_V1;
        else if (type == 1)
            return HDCP_V2_2;
        else
            return HDCP_UNKNOWN;
    }

    return HDCP_UNKNOWN;
}

HdcpManager::HdcpAuthorizedStatus HdcpManager::getAuthorizedStatus() const
{
    char state = 0;

    if (read_from_file(SYSPATH_HDCP_STATUS, &state, 1) == 1) {
        if (state == 3)
            return AUTHORIZED;
    }

    return mHdcpEnabled ? ERROR : UN_AUTHORIZED;
}

#define ESM_IMG_SIZE (200*1024)
int HdcpManager::loadFirmware(const char *fw)
{
    mFirmwareReady = false;

    char *esmimg = (char *)malloc(ESM_IMG_SIZE);
    if (esmimg == nullptr)
        return -ENOMEM;

    int length = read_from_file(fw, esmimg, ESM_IMG_SIZE);
    if (length > 0) {
        dd_info("read firmware '%s', size=%d", fw, length);
        int fd = open(DEVPATH_HDMI, O_RDWR);

```

```

    if (fd < 0) {
        dd_error("Could not open hdmi device, %s(%d)", strerror(errno), errno);
        mFirmwareReady = false;
        return -errno;
    }
    unsigned long arg[3] = { 0 };
    arg[0] = (unsigned long)esmimg;
    arg[1] = length;
    if (ioctl (fd, AW_IOCTL_HDMI_HDCP22_LOAD_FW, arg) == 0) {
        mFirmwareReady = true;
        dd_info("load hdcp2.2 firmware success (size=%d)", length);
    }
    close(fd);
}

return mFirmwareReady ? 0 : -1;
}

```

## HdcpManager.h

```

#ifndef HDCP_MANAGER_H
#define HDCP_MANAGER_H

#include <stdint>
#include <string>

class HdcpManager {
public:
    HdcpManager();
    ~HdcpManager() = default;

    enum HdcpAuthorizedStatus {
        ERROR, // hdp authentication error
        UN_AUTHORIZED,
        AUTHORIZED,
    };

    enum HdcpLevel: uint32_t {
        HDCP_UNKNOWN, // Unable to determine the HDCP level
        HDCP_NONE, // No HDCP, output is unprotected
        HDCP_V1, // HDCP version 1.0
        HDCP_V2, // HDCP version 2.0 Type 1.
        HDCP_V2_1, // HDCP version 2.1 Type 1.
        HDCP_V2_2, // HDCP version 2.2 Type 1.
        HDCP_NO_OUTPUT // No digital output, implicitly secure
    };

    std::string toString(HdcpAuthorizedStatus status) {
        switch (status) {
            case ERROR:
            default:
                return std::string("ERROR");
            case UN_AUTHORIZED:
                return std::string("UN_AUTHORIZED");
            case AUTHORIZED:
                return std::string("AUTHORIZED");
        }
    }

    std::string toString(HdcpLevel level) {

```

```
switch (level) {
case HDCP_UNKNOWN:
default:
    return std::string("HDCP_UNKNOWN");
case HDCP_NONE:
    return std::string("HDCP_NONE");
case HDCP_V1:
    return std::string("HDCP_V1");
case HDCP_V2:
    return std::string("HDCP_V2");
case HDCP_V2_1:
    return std::string("HDCP_V2_1");
case HDCP_V2_2:
    return std::string("HDCP_V2_2");
case HDCP_NO_OUTPUT:
    return std::string("HDCP_NO_OUTPUT");
}
}

HdcpLevel getConnectedHdcpLevel() const;
HdcpAuthorizedStatus getAuthorizedStatus() const;
int configHdcp(bool enable);

private:
    int loadFirmware(const char *fw);
    bool mFirmwareReady;
    bool mHdcpEnabled;
};

#endif
```

## debug.h

```
#ifndef DEBUG_H
#define DEBUG_H

#include <stdio.h>

#define dd_info printf
#define dd_error printf

#endif /* DEBUG_H */
```

## hdcptool.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <memory>

#include "HdcpManager.h"

struct usropt {
    int config;
    int enable;
    int dump;
};
```

```
static struct usropt inputopt;

static void usage(char *name)
{
    fprintf(stderr, "Usage: %s [-p] [-h] [-e enable]\n", name);
    fprintf(stderr, "    -p: print hdcp info\n");
    fprintf(stderr, "    -e: enable or disable hdcp: [-e 1] or [-e 0]\n");
}

static void parseOpt(int argc, char **argv)
{
    memset(&inputopt, 0, sizeof(inputopt));

    int c;

    while (1) {
        c = getopt(argc, argv, "phe:");
        if (c == EOF)
            break;
        switch (c) {
            case 'p':
                inputopt.dump = 1;
                break;
            case 'e':
                if (optarg) {
                    inputopt.config = 1;
                    inputopt.enable = strtoul(optarg, NULL, 0);
                }
                break;
            case 'h':
            default:
                usage(argv[0]);
                exit(1);
        }
    }

    if (optind != argc) {
        usage(argv[0]);
        exit(1);
    }
}

int main(int argc, char** argv)
{
    std::unique_ptr<HdcpManager> mHdcpManager;

    parseOpt(argc, argv);

    mHdcpManager = std::make_unique<HdcpManager>();

    if (inputopt.config) {
        int ret = mHdcpManager->configHdcp(inputopt.enable ? true : false);
        printf("%s hdcp return %d\n", inputopt.enable ? "enable" : "disable", ret);
    }

    if (inputopt.dump) {
        HdcpManager::HdcpAuthorizedStatus status = mHdcpManager->getAuthorizedStatus();
        printf("hdcp authorized status: %s\n",
            mHdcpManager->toString(status).c_str());
    }
}
```



```
HdcpManager::HdcpLevel level = mHdcpManager->getConnectedHdcpLevel();
printf("hdcp level: %s\n",
       mHdcpManager->toString(level).c_str());
}
return 0;
}
```

## Makefile

```
TOOL_NAME=hdcpool
CC=~/.android/android13/longan/out/toolchain/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu/bin/aarch64-
none-linux-gnu-g++

all: $(TOOL_NAME)

$(TOOL_NAME):
$(CC) $(TOOL_NAME).cpp HdcpManager.cpp -o $(TOOL_NAME)

.PHONY: clean
clean:
rm $(TOOL_NAME)
@find . -name *.o -exec rm -f {} \;
```

## 11 附录 5：HDMI 展频配置及应用

### ⚠ 注意

展频的调整需谨慎，因为会涉及到时钟频率，因为开展频一定要考虑清楚。并且完成展频配置以后，建议进行设备的兼容性测试。避免因开启展频后影响了基本的显示功能。

使用 HDMI 开启展频时，需先关注 HDMI 使用的时钟源是 PHYPLL 还是 CCMU。先参考下面的方式确认：

```
cat /sys/class/hdmi/hdmi/attr/hdmi_source
```

输出信息部分截图如下：

```
[drv cfg]
|          |          |          |          |          |          |
|  name    |-----|          |          |          |          |
|          | cec   | hdcplx | hdcp2x | clk_src | resistor |
|-----+-----+-----+-----+-----+-----+
| state   | off  | on    | on    | phypll | onboard |
```

图 11-1: 附录 5-确定模块时钟来源

从上面的信息可以看出当前是 PHYPLL 模式。如果是 CCMU 模式，那么该字段为 ccmu。

```
[dw ctrl]
|          |          |          |          |          | |
|  name    |-----|          |          |          |          |
|          | pixel clk | tmds clk | repeat | color bits |
|-----+-----+-----+-----+-----+
| state   | 297000 | 297000 | 0      | 8          |
```

图 11-2: 附录 5-当前 TMDS 时钟

从上面信息可以看出当前 TMDS 是 297000KHz 也就是 297MHz

```

[top phy]
| name | ref clk | output | state |
|-----|-----|-----|-----|
| state | 24M    | on    | lock  |
- pll: 0xE8076200, 0x00035000, 0x00000000, 0x30000000

```

图 11-3: 查看 TOPPHY 状态

从上面信息可以看到有 [TOP PHY] 的打印，这里可以侧面证明实际使用的是 PHYPLL、如果没有 TOP PHY 的打印，说明用的还是 CCMU 模式。

## 11.1 PHYPLL 模式展频

因为展频相对涉及到当前时钟以及实际展频需求，因此下面只是给出个别场景的示例参考。

从上面的 [TOP PHY] 信息中，PLL 下有四组参数 arg[4]。其中展频需要用到最后两组参数：arg[2]0x00000000，arg[3]0x30000000

参数	参数值	位域	说明
arg[0]	0xE8076200	Bit[15:8]	PLL 倍频系数
arg[1]	0x00035000		不关注
arg[2]	0x00000000	Bi[31:31]	Sigma Delta 模式启用。1：开启；0：关闭
		Bit[28:20]	为 WAVE STEP 值
		Bit[19:19]	为展频方向。0：向上展频；1：向下展频
		Bit[18:17]	为展频频点。0：31.5KHz；1：32KHz；2：32.5KHz；3：33KHz
		Bit[16:0]	为 WAVE_BOT 值
arg[3]	0x30000000	Bit[27:27]	为控制是否开启展频。1：打开；0：关闭

**示例：时钟源 24M，TMDS 时钟为 297M，向下展频 0.1%，展频频点为 32K。**

从 [TOP PHY] 中可知当前是 24M 时钟源。

**Step1：计算时钟源的频点。**

从 arg[0] 中获取倍频系数为 0x62。因此时钟源频点为：24 \* (0x62 + 1) = 2376MHz

**Step2：计算展频频率范围**

按照 0.1% 进行展频，那么展频后的频率为：2376 \* (1 + 0.1%) = 2376.2376MHz

**Step3：计算 WAVE BOT 和 WAVE STEP 值**

```

X_1 = (2376 - (0x62 + 1) * 24) / 24 = 0
X_2 = (2376.2376 - (0x62 + 1) * 24) / 24 = 0.0099

BOT = 2^17 * (X_1) = 0x0
WAVE = 2^18 * (X_2) / (24 * 1000 / 32) ≈ 3 = 0x3
# 0x62: 即Step1中获取的倍频系数
# 24: 为当前的时钟源, 如果是26M那么替换为26M
# 24 * 1000: 单位替换, 将24MHz变为24000KHz
# 32: 即示例中希望展频频点为32KHz, 如果是其他频点则替换为其他频点

```

#### Step4: 生成参数值

```

arg[2] = (0x1 << 31) || (WAVE << 20) || (0x1 << 19) || (0x1 << 17) || (BOT << 0) = 0x80388000
arg[3] = 0x30000000 || (0x1 << 27) = 0x38000000
# 0x1 << 19: 展频方向配置, 参考上面表格位的描述
# 0x1 << 17: 展频频点配置, 参考上面表格位的描述

```

#### Step5: 临时调试使用节点写入

```

echo 0x0560002C 0x80388000 > /sys/class/sunxi_dump/write
echo 0x05600030 0x38000000 > /sys/class/sunxi_dump/write

```

#### Step6: 固化到代码中使用

##### 1、根据路径找到源码文件 phy\_top.c

```
{BSP}/driver/drm/sunxi_device/hardware/lowlevel_hdmi20/phy_top.c
```

##### 2、确认使用参数表

如果当前是 26M 时钟: struct top\_phy\_pll\_s sun60i\_phypll\_26m

如果当前是 24M 时钟: struct top\_phy\_pll\_s sun60i\_phypll\_24m

##### 3、找到参数并更改。

下面是当前示例的参数表, 即 24M 时钟表内的 297MHz 使用的频率

```
{297000, 0xE8076200, 0x00035000, 0x00000000, 0x30000000},
```

根据 Step5 验证可以正常使用的参数, 更新替换为

```
{297000, 0xE8076200, 0x00035000, 0x80388000, 0x38000000},
```

这样就完成了代码上的固化。

#### 📖 说明

如果有使用 uboot 输出, 并且也需要展频的话, 按照相同的步骤添加到 uboot 代码中即可。

## 11.2 CCMU 模式展频

当前平台 CCMU 暂未划分频点给 HDMI 模块使用。暂不支持！！

## 12 结束

---

1、HDMI 模块不同于其他模块，其协议流程复杂，因此需要相关的开发人员能熟读相关 Spec 及信号原理。

2、在实际中可能会遇到各种各样的兼容性问题，针对这些兼容性问题，应保持以下的态度做排查：

2.1、问题是否能稳定复现？具体详细的复现步骤，任何一项差异都可能引起问题复制

2.2、针对问题做单一变量控制，引入对比实验






## 著作权声明

版权所有 © 2024 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。