



AW HDMI2.0 开发指南

**版本号: 1.4.1
发布日期: 2024.09.19**

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.4.01	AWA0962	初始版本
1.1	2023.3.08	AWA1977	移除非必要章节及增加相关调试说明
1.1.1	2023.7.10	AWA1977	增加调试命令，驱动版本代号升级为 V1.2
1.2	2023.11.06	AWA1977	重构文档描述
1.3	2024.03.09	AWA2130	重构文档结构，删减过于复杂的驱动软件架构介绍，根据客户的具体问题来编排章节内容。
1.4	2024.04.10	AWA2130	增加 H728 Android14 说明。
1.4.1	2024.09.19	AWA2130	增加 H618 Android14 说明。



目 录

1 前言	1
1.1 文档简介	1
1.2 目标读者	1
1.3 适用范围	1
1.4 相关术语介绍	1
1.4.1 硬件术语	1
1.4.2 软件术语	2
2 协议简述	3
2.1 硬件引脚	3
3 模块介绍	4
3.1 模块平台功能	4
3.2 模块源码介绍	5
3.3 模块功能配置	6
3.3.1 配置 HDMI	6
3.3.2 配置 CEC	7
3.3.3 配置 HDCP14	7
3.3.3.1 SDK 配置	7
3.3.3.2 密钥切割	8
3.3.3.3 烧录过程	8
3.3.4 配置 HDCP22	12
3.3.4.1 SDK 配置	13
3.3.4.2 密钥配置	13
3.3.4.3 烧录过程	14
4 模块使用	19
4.1 切换显示设置	19
4.1.1 Android 方案	19
4.1.2 Linux 方案	20
4.2 使用 CEC 功能	21
4.2.1 Android 方案	21
4.2.2 Linux 方案	21
4.3 使用 HDCP14 功能	21
4.3.1 Android 方案	21
4.3.2 Linux 方案	22
4.4 使用 HDCP22 功能	22
4.4.1 Android 方案	22
4.4.2 Linux 方案	23

4.5	获取 HDCP 信息	24
4.5.1	Android 方案	24
4.5.2	Linux 方案	24
4.6	增加特殊分辨率	25
5	调试方式	28
5.1	查看 HDMI 模块加载成功	28
5.2	查看 HDMI 连接状态	28
5.3	模拟 HDMI 连接状态	28
5.4	查看 rxsense 状态	29
5.5	查看输出的分辨率	29
5.6	查看输出的颜色格式	29
5.7	查看输出的颜色深度	29
5.8	查看输出的信号格式	29
5.9	查看设备支持的分辨率	30
5.10	查看设备 Audio 信息	30
5.11	查看设备 CEC 信息	30
5.12	获取设备 EDID 数据	30
5.13	替换设备 EDID 数据	30
5.14	查看 HDCP 开启状态	31
5.15	查看 HDCP 认证状态	31
5.16	查看 HDCP 认证类型	31
5.17	日志抓取	31
5.18	近期日志抓取	32
5.19	获取所有寄存器的打印	32
5.20	输出 HDMI Pattern	32
6	FAQ	34
6.1	无信号问题	34
6.1.1	排查是否跟 Rx 设备相关	34
6.1.2	排查 Tx 端问题模块	34
6.1.3	排查 Rx 端是否支持	34
6.1.4	排查 HDMI 的输出状态	35
6.1.5	排查硬件特性	35
6.2	黑屏问题	36
6.2.1	排查是否跟 Rx 相关	36
6.2.2	排查 Tx 端问题模块	36
6.2.3	排查是否跟 HDCP 相关	36
6.2.4	HDMI 排查-寄存器对比	37
6.3	无声问题	37
6.3.1	排查 HDMI 工作模式	37
6.3.2	排查是否跟 Rx 相关	37
6.3.3	Audio 排查-工作状态	38

6.3.4	HDMI 排查-LOG 对比	39
6.3.5	HDMI 排查-寄存器对比	39
6.3.6	HDMI 排查-尝试恢复	39
6.4	颜色问题	40
6.4.1	排查 HDMI 工作模式	40
6.4.2	排查是否跟 Rx 相关	40
6.4.3	排查 Tx 端问题模块	40
6.4.4	排查颜色格式	40
6.5	DVI 问题	41
6.5.1	排查 EDID 解析	41
7	附录	43
7.1	Linux CEC 应用程序样例	43
7.2	Linux HDCP 应用程序样例	51
8	结束语	58



插 图

图 3-1	配置 Key 入口	9
图 3-2	删除 hdcppkf	10
图 3-3	配置 hdcpkey	11
图 3-4	DragonSN_ 软件截图	12
图 3-5	配置 Key 入口	15
图 3-6	删除 hdcpkey	16
图 3-7	配置 hdcppkf	17
图 3-8	DragonSN_ 软件截图 2	18
图 6-1	查看 video_data_block	35
图 6-2	查看 audio_data_block	42



1 前言

1.1 文档简介

本文介绍 Allwinner HDMI 模块的基本功能及如何配置使用，以及针对常见问题的排查方式和手段。

1.2 目标读者

HDMI 模块开发工程师

HDMI 模块测试工程师

1.3 适用范围

表 1-1: 适用产品列表

产品包	安卓版本	内核版本
H618 Android14	Android_U	Linux-5.15
H728 Android14	Android_U	Linux-5.15
A527 Android13 GMS Tablet	Android_T	Linux-5.15
T527 Android13	Android_T	Linux-5.15
A527 Android13	Android_T	Linux-5.15
T527 Tina	-	Linux-5.10/Linux-5.15
A527 Tina	-	Linux-5.10/Linux-5.15

1.4 相关术语介绍

1.4.1 硬件术语

表 1-2: HDMI 硬件术语

术语	说明
HPD	用来表示 Rx 是否连接
5V	用来表示 Tx 是否连接

1.4.2 软件术语

表 1-3: HDMI 软件术语

术语	说明
Tx	Tx 表示 HDMI 的发送端。常见有盒子、电脑等输出设备
Rx	Rx 表示 HDMI 的接收端。常见有电视、显示器等显示设备
DTS	Linux 使用的设备树文件
EDID	Rx 向 Tx 表示接收数据能力的信息（支持的分辨率、显示格式等）
HDCP	由 DCP 组织提供的针对音视频加密的协议
Disp2	Allwinner 使用的基于 FB 框架实现的显示框架
DE	Allwinner 的 Display 模块
主显	Allwinner 的 DE0 连接 HDMI
副显	Allwinner 的 DE1 连接 HDMI
HDCP14	指的是由 DCP 组织推出的 HDCP1.4 协议，在本文中同样用于表示 HDCP14 功能
HDCP22	指的是由 DCP 组织推出的 HDCP2.2 协议，在本文中同样用于表示 HDCP22 功能
原始密钥	指的是从 DCP 组织购买的密钥，未作任何加密修改
hdcp2pkf.bin	HDCP2.2 用来加密 esm firmware 的密钥
HDCP2_TX_KEY.bin	从 DCP 组织购买的 HDCP key
esm.fex	esm 加密后的固件

2 协议简述

HDMI 是一个通过 TMDS 方式来传输音视频数据的通道，其包含了 HDCP、CEC 等子功能。

2.1 硬件引脚

Pin	Signal Assignment	Description
1	TMDS Data2+	用于传输 TMDS 数据的通道 2。正向
2	TMDS Data2 Shield	数据通道 2 的 Shield。接地
3	TMDS Data2-	用于传输 TMDS 数据的通道 2。反向
4	TMDS Data1+	用于传输 TMDS 数据的通道 1。正向
5	TMDS Data1 Shield	数据通道 1 的 Shield。接地
6	TMDS Data1-	用于传输 TMDS 数据的通道 1。反向
7	TMDS Data0+	用于传输 TMDS 数据的通道 0。正向
8	TMDS Data0 Shield	数据通道 0 的 Shield。接地
9	TMDS Data0-	用于传输 TMDS 数据的通道 0。反向
10	TMDS Clock+	用于 TMDS 数据传输的时钟。正向
11	TMDS Clock Shield	时钟通道的 Shield。接地
12	TMDS Clock-	用于 TMDS 数据传输的时钟。反向
13	CEC	用于传输 CEC 消息的通道。
14	Utility	未使用。
15	SCL	参考 I2C 协议的 SCL。也叫做 DDC 的时钟通道
16	SDA	参考 I2C 协议的 SDA。也叫做 DDC 的数据通道
17	DDC/CEC Ground	DDC 和 CEC 的 Shield。接地
18	+5V Power	5V 电压脚，Rx 检测 5V 电压来判断 Tx 有接入
19	Hot Plug Detect	HPD 引脚，Tx 检测这个引脚是否拉高来判断 Rx 是否接入。

3 模块介绍

3.1 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。HDMI 的视频数据源由 DE 提供。

⚠ 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 3-1: 芯片硬件功能支持表

芯片	HDMI2.0	CEC	HDCP14	HDCP22
H618	Y	Y	Y	Y
H728	Y	Y	Y	N
A527	Y	Y	Y	N
T527	Y	Y	Y	N

表 3-2: SDK 平台功能实现表

SDK 平台	HDMI2.0	CEC	HDCP14	HDCP22	Uboot 输出
H618 Android14	Y	Y	Y	Y	Y
H728 Android14	Y	Y	Y	N	Y
A527 Android13 GMS Tablet	Y	N	Y	N	N
T527 Android13	Y	N	Y	N	Y
A527 Android13	Y	N	Y	N	Y
T527 Tina	Y	N	Y	N	Y
A527 Tina	Y	N	Y	N	Y

3.2 模块源码介绍

HDMI 的源码路径：bsp/drivers/video/sunxi/disp2/hdmi2/*

```
.
├── aw_hdmi_core
│   ├── aw_hdmi_core.c # hdmi core层，融合处理hdmi、hdcp、cec等逻辑，提供统一接口给驱动层
│   ├── aw_hdmi_core.h
│   └── dw_hdmi # hdmi2.0的硬件层，面向硬件操作
│       ├── dw_access.c
│       ├── dw_access.h
│       ├── dw_audio.c
│       ├── dw_audio.h
│       ├── dw_cec.c
│       ├── dw_cec.h
│       ├── dw_dev.h
│       ├── dw_edid.c
│       ├── dw_edid.h
│       ├── dw_fc.c
│       ├── dw_fc.h
│       ├── dw_hdcp22_tx.c
│       ├── dw_hdcp22_tx.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
│       ├── dw_scdc.c
│       ├── dw_scdc.h
│       ├── dw_video.c
│       ├── dw_video.h
│       ├── phy_aw.c
│       ├── phy_aw.h
│       ├── phy_inno.c
│       ├── phy_inno.h
│       ├── phy_snps.c
│       └── phy_snps.h
├── aw_hdmi_define.h
├── aw_hdmi_dev.c # hdmi设备层，向kernel注册设备及sysfs等
├── aw_hdmi_dev.h
├── aw_hdmi_drv.c # hdmi驱动层
├── aw_hdmi_drv.h
├── aw_hdmi_log.c
├── aw_hdmi_log.h
└── Makefile
```


3.3 模块功能配置

3.3.1 配置 HDMI

Step1: 在 menuconfig 中，配置打开 “DISP Driver Support” 和 “HDMI2.0 Driver Support” 选项。

```
Allwinner BSP --->
Device Drivers --->
Video Drivers --->
<*> DISP Driver Support(sunxi-disp2)
<*> HDMI Tx Driver Support(sunxi-disp2) --->
<*> HDMI2.0 Driver Support(sunxi-disp2) --->
```

名称	作用
DISP Driver Support	disp2 驱动使能，HDMI 驱动依赖于 disp2 驱动。
HDMI2.0 Driver Support	HDMI2.0 驱动使能

Step2: 在 uboot_board.dts 中，找到 disp 和 hdmi 节点，配置如下信息。

💡 技巧

如果不需要 HDMI 显示 LOGO，可以不用做当前步骤，跳到 Step3。

下面以 HDMI 1080p@60 显示 LOGO 启动为例：

```
&disp {
    ...
    dev0_output_type    = <4>;
    dev0_output_mode    = <10>;
    dev0_screen_id      = <0>;
    dev0_do_hpd         = <1>;
    ...
};

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

Step3: 在 board.dts 中，配置 hdmi 节点的状态为 okay。

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


3.3.2 配置 CEC



请先查看**模块平台功能**章节，确认当前平台是否支持 CEC 功能。

Step1: 在 menuconfig 中，配置打开 “HDMI2.0 CEC” 选项。

```
Allwinner BSP --->
Device Drivers --->
Video Drivers --->
  <*> HDMI Tx Driver Support(sunxi-disp2) --->
    <*> HDMI2.0 Driver Support(sunxi-disp2) --->
      [*] HDMI2.0 CEC --->
```

Step2: 在 board.dts 中，配置使能 “hdmi_cec_support” 。

```
&hdmi {
    ...
    hdmi_cec_support = <1>;
    ...
    status = "okay";
};
```

3.3.3 配置 HDCP14



请先查看**模块平台功能**章节，确认当前平台是否支持 HDCP1.4 功能。

3.3.3.1 SDK 配置

Step1: 在 menuconfig 中，配置打开 “HDMI2.0 HDCP” 选项。

```
Allwinner BSP --->
Device Drivers --->
Video Drivers --->
  <*> HDMI Tx Driver Support(sunxi-disp2) --->
    <*> HDMI2.0 Driver Support(sunxi-disp2) --->
      [*] HDMI2.0 HDCP --->
```

Step2: 在 board.dts 中，配置使能 “hdmi_hdcp_enable” 。

```
&hdmi {
    ...
    hdmi_hdcp_enable = <1>;
    ...
    status = "okay";
};
```

Step3: 在固件编译中，需要将固件打包为**安全固件**，这样后续密钥才能烧录成功！！

3.3.3.2 密钥切割

Step1: 购买密钥

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，由成千上万组原始密钥组成，由具体购买的密钥数量决定。

💡 技巧

购买密钥流程：首先需要缴纳年会员费，加入会员组织，然后根据购买密钥的数量，签订不同单价的合同，DCP 组织会不定期地抽查设备和商务合同来进行监管。

Step2: 切割密钥

请联系全志 AE 同事获取 HDCPLoad 工具。

Step2.1: 查看购买的密钥集中有多少个密钥

在 HDCP 工具包中，打开 CMD 命令行，并且执行以下命令。

```
.\HDCPLoad.exe -v14 -count .\hdcpl4_key
```

参数	含义
-v14	HDCP1.4 指定标签，默认添加。
-count	显示购买到的原始密钥中包含的子密钥个数
.\hdcpl4_key	购买到的密钥文件命名（请按实际情况修改，当前仅为演示）

Step2.2: 切割密钥

在 HDCP 工具包中，打开 CMD 命令行，并且执行以下命令。

```
.\HDCPLoad.exe -v14 -prefix hdcp -range 1 2 .\hdcpl4_key .\
```

参数	含义
-v14	hdcpl.4 指定标签，默认添加。
-prefix	指定输出文件名前缀，例如：hdcp，就会生成成为 hdcp_1.bin，hdcp_2.bin 等。
-range	指定分割范围，例如：1 2，表示切割出第 1 个和第 2 个的密钥。
.\hdcpl4_key	购买到的密钥文件输入路径（请按实际情况修改，当前仅为演示）
.\	切割后的密钥文件输出路径（请按实际情况修改，当前仅为演示）

3.3.3.3 烧录过程

Step1: 配置 DragonSN 工具

Step1.1: 打开 DragonSN 工具，点击左下角的“配置 key”；

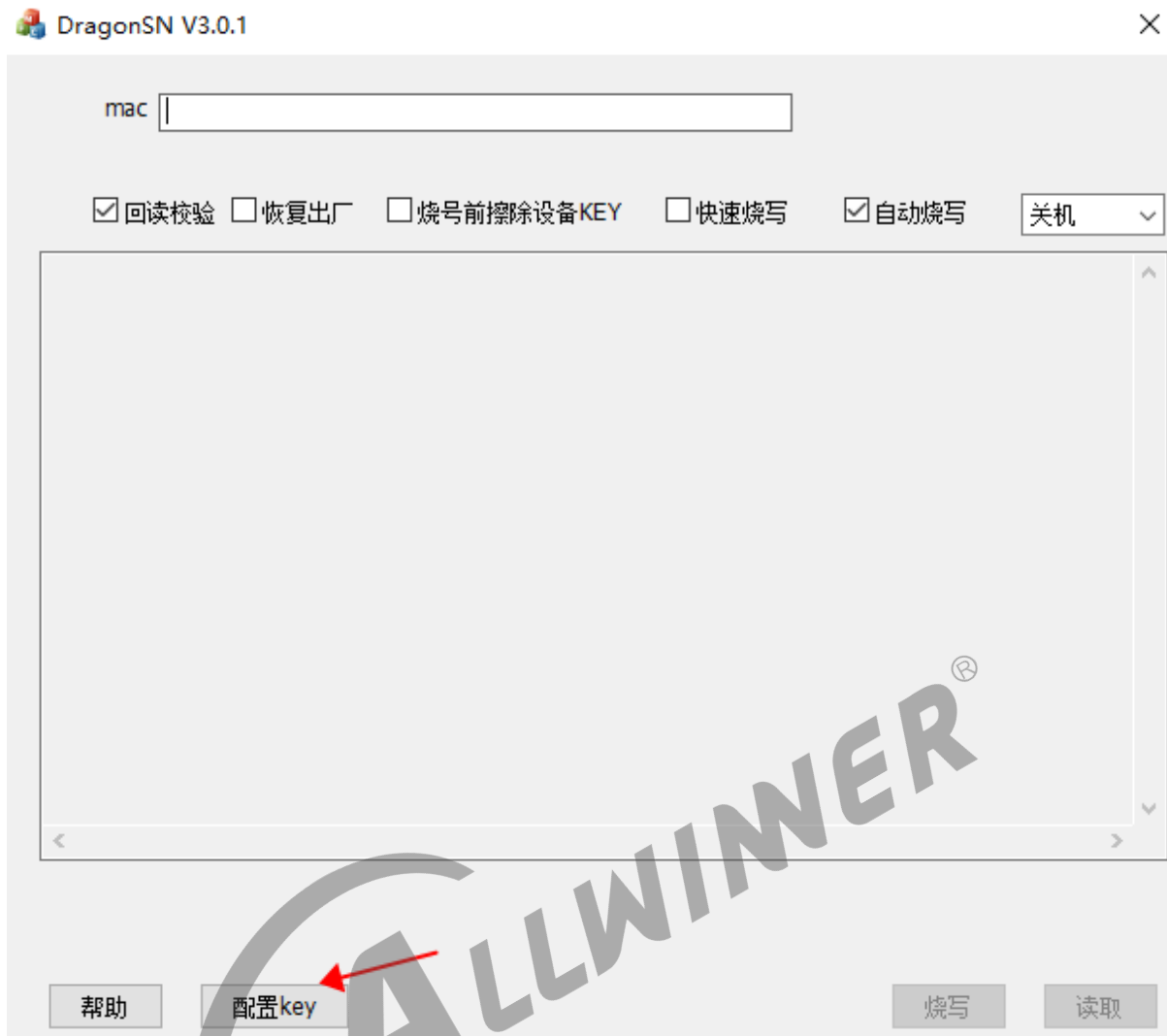


图 3-1: 配置 Key 入口

Step1.2: 如果当前配置有其他的配置项，对着每个项目“右击”，然后点击“Delete”即可；

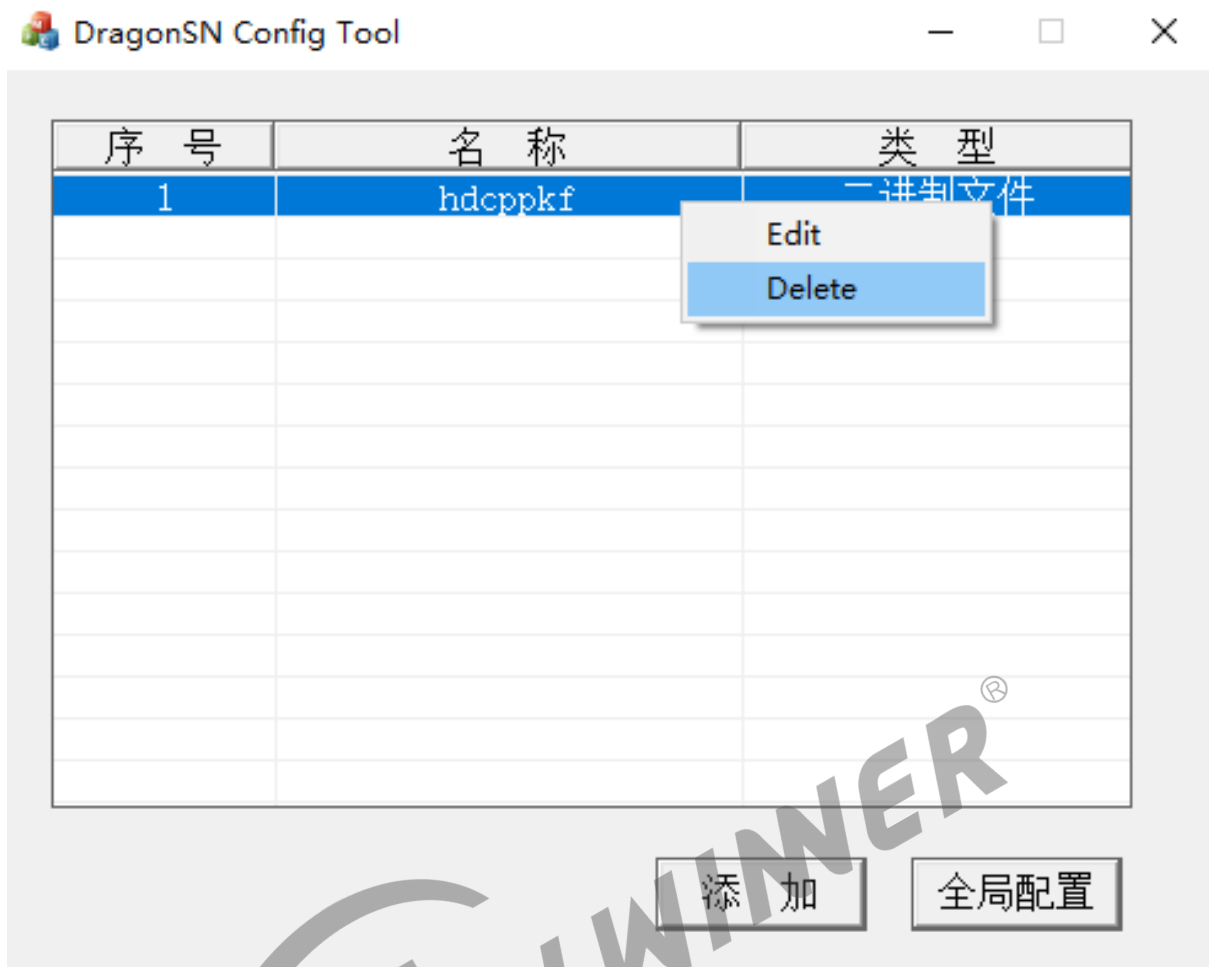


图 3-2: 删除 hdcppkf

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

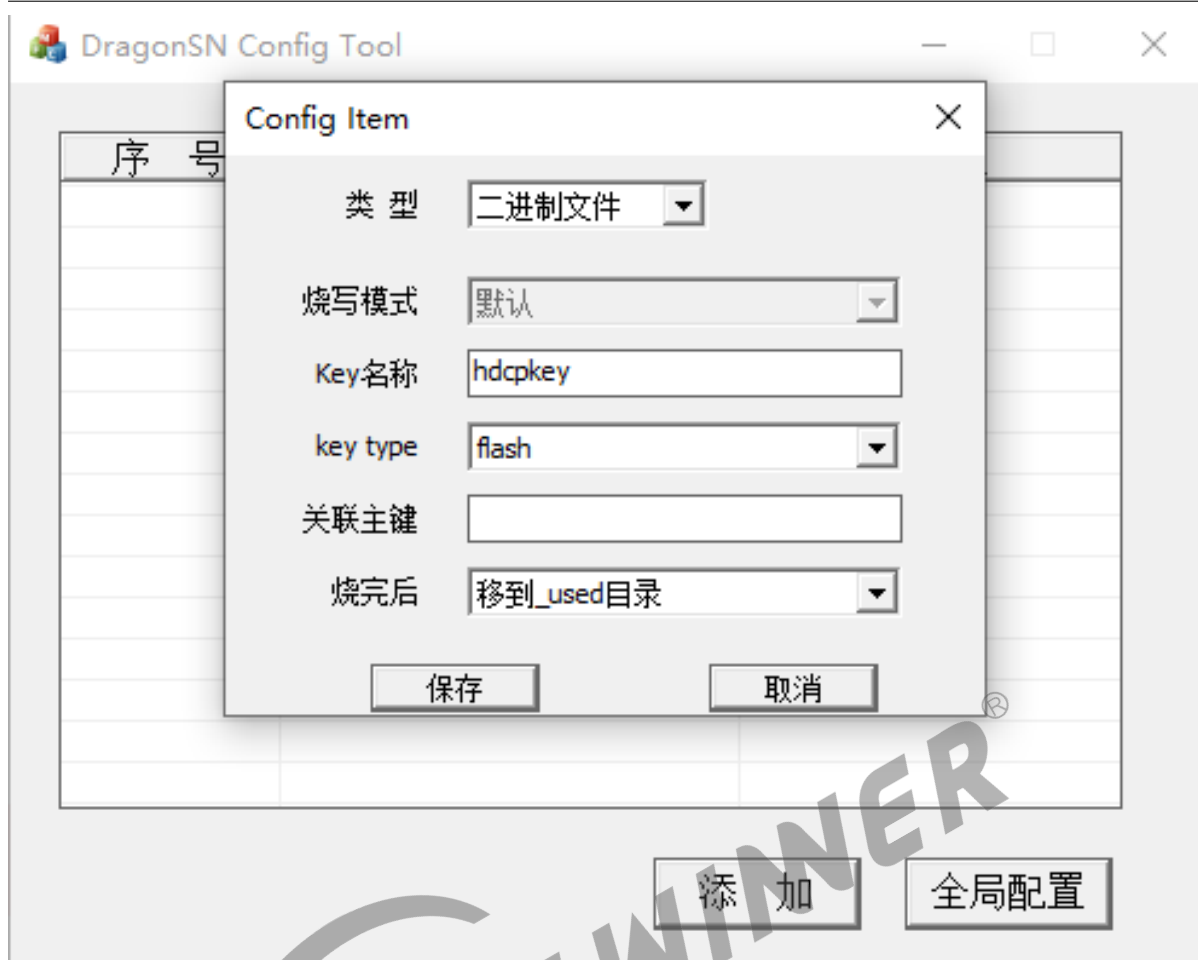


图 3-3: 配置 hdckey

说明

DCP 组织要求使用 HDCP1.4 的每台设备都要烧录不同的密钥，所以烧完后的设置里选择移到 _used 目录。

保留：一直烧同一个密钥。

移到 _used 目录：每烧完一个密钥，就会将该密钥移到同路径下的 _used 后缀文件夹中。

Step1.4: 关闭 DragonSN 配置页面，回到 DragonSN 工具主界面。

Step2: 烧录 hdc_1.bin

Step2.1: 点击“导入文件”，选择要烧录的 hdc_1.bin 所在文件夹（尽量避免中文路径），获取方式请参考[密钥切割](#)章节；

Step2.2: 将软件界面配置成与下图一致；

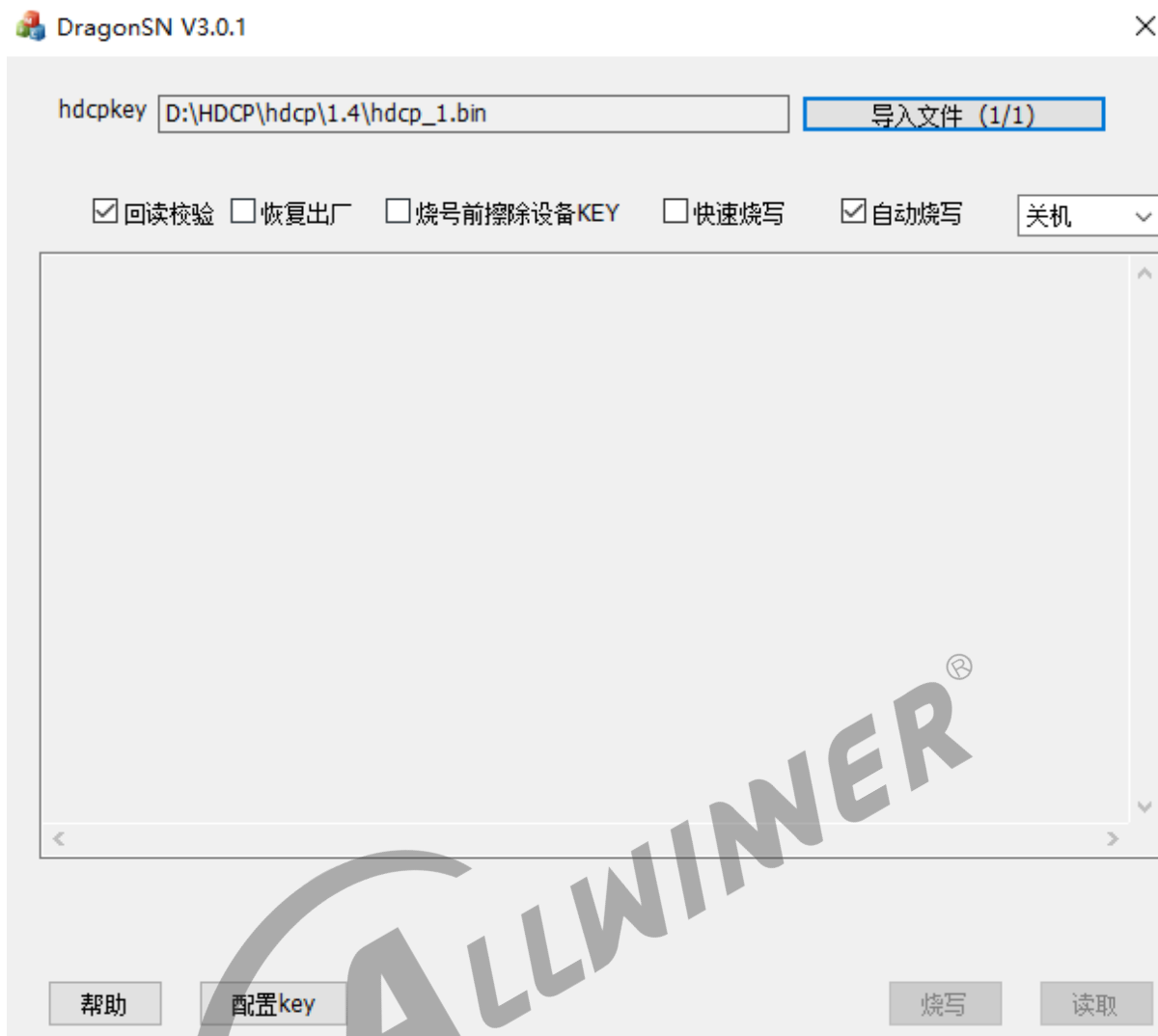


图 3-4: DragonSN_ 软件截图

Step2.3: 确认已经使用 USB 线连接开发板与电脑；

Step2.4: 重启开发板，工具会自动开始烧录；

Step2.5: 烧录完成后，会显示是否烧录成功，也可点击读取按钮，确认烧录情况；

3.3.4 配置 HDCP22

📖 说明

请先查看**模块平台功能**章节，确认当前平台是否支持 HDCP2.2 功能。

3.3.4.1 SDK 配置

Step1: 在 menuconfig 中，配置打开 “HDMI2.0 HDCP” 选项和子选项 “HDMI2.0 HDCP2.2”。

```
Allwinner BSP --->
Device Drivers --->
Video Drivers --->
  <*> HDMI Tx Driver Support(sunxi-disp2) --->
    <*> HDMI2.0 Driver Support(sunxi-disp2) --->
      [*] HDMI2.0 HDCP --->
        [*] HDMI2.0 HDCP2.2 --->
```

Step2: 在 board.dts 中，配置使能 “hdmi_hdcp_enable” 和 “hdmi_hdcp22_enable”。

```
&hdmi {
    ...
    hdmi_hdcp_enable = <1>;
    hdmi_hdcp22_enable = <1>;
    ...
    status = "okay";
};
```

Step3: 在固件编译中，需要将固件打包为**安全固件**，这样后续密钥才能烧录成功!!!

3.3.4.2 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

💡 技巧

购买密钥流程：首先需要缴纳年会员费，加入会员组织，然后根据购买密钥的数量，签订不同单价的合同，DCP 组织会不定期地抽查设备和商务合同来进行监管。至于单个密钥能烧录多少台，则需要根据与 DCP 组织的合同来约定。

Step2: 生成 fex 加密密钥

加密密钥要求是一串长度为 16 字节的随机数，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

请联系全志 AE 同事获取 esm 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)
```

3.3.4.3 烧录过程

Step1: 配置 DragonSN 工具

Step1.1: 打开 DragonSN 工具，点击左下角的“配置 key”；



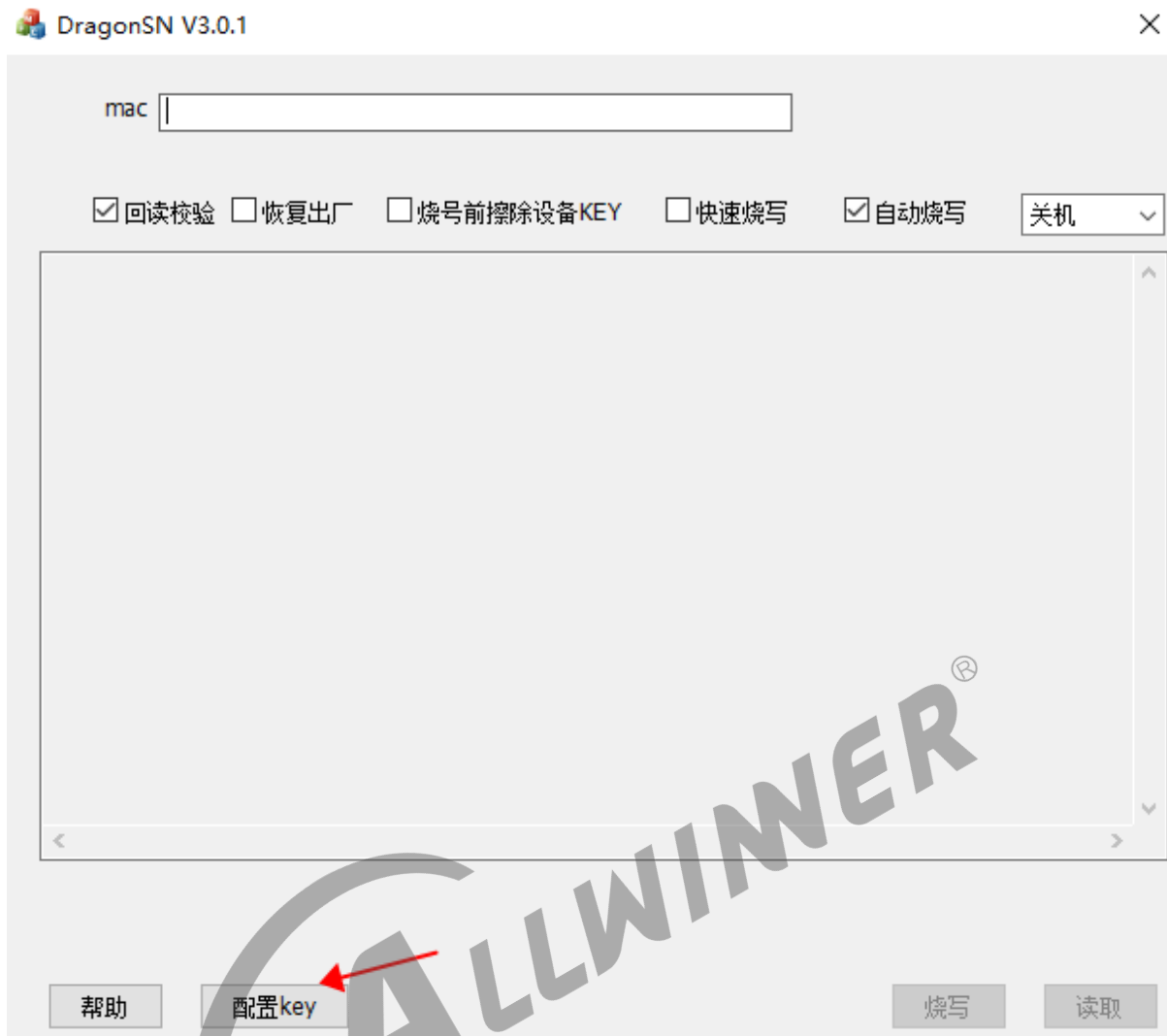


图 3-5: 配置 Key 入口

Step1.2: 如果当前配置有其他的配置项，对着每个项目“右击”，然后点击“Delete”即可；

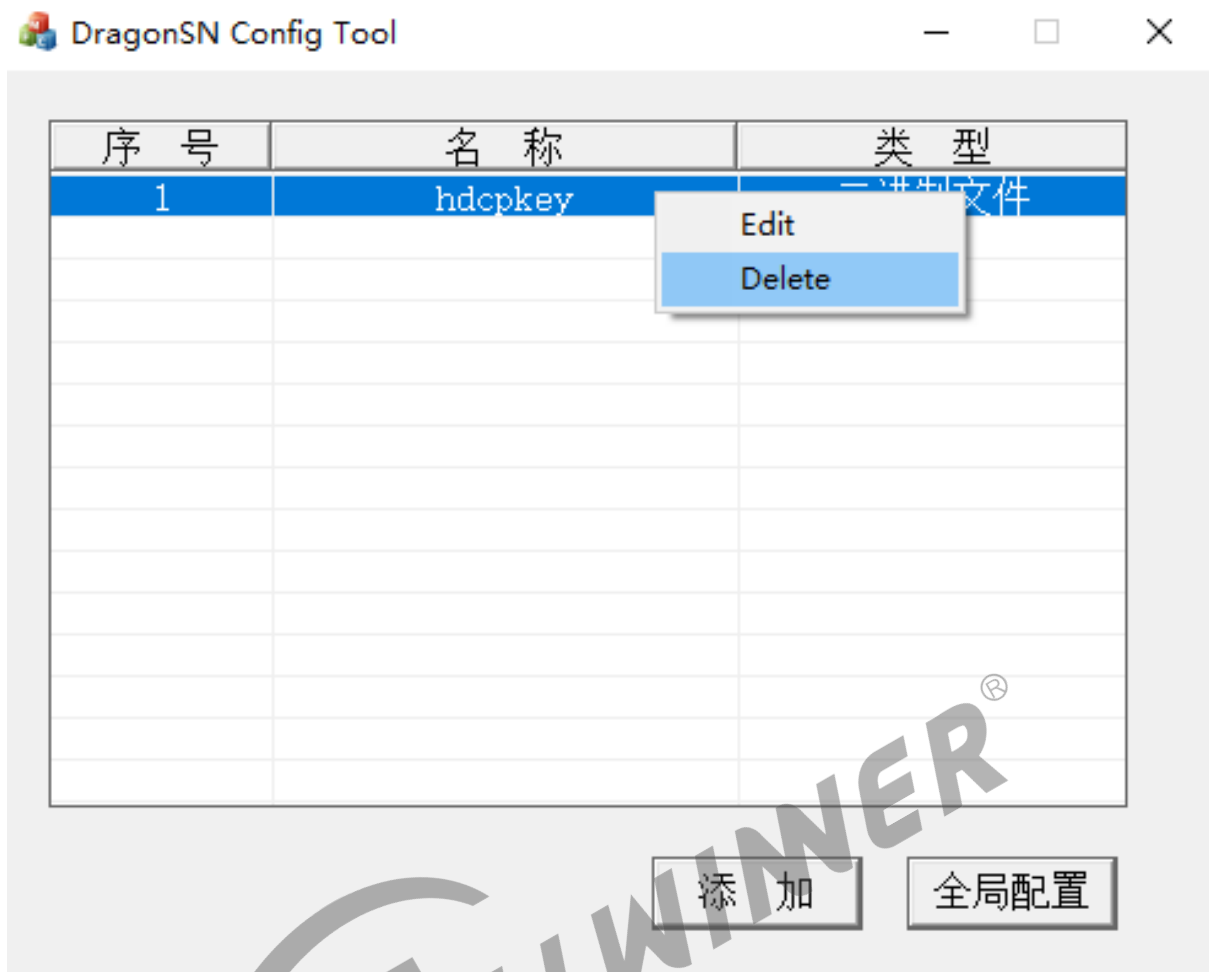


图 3-6: 删除 hdcpkey

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

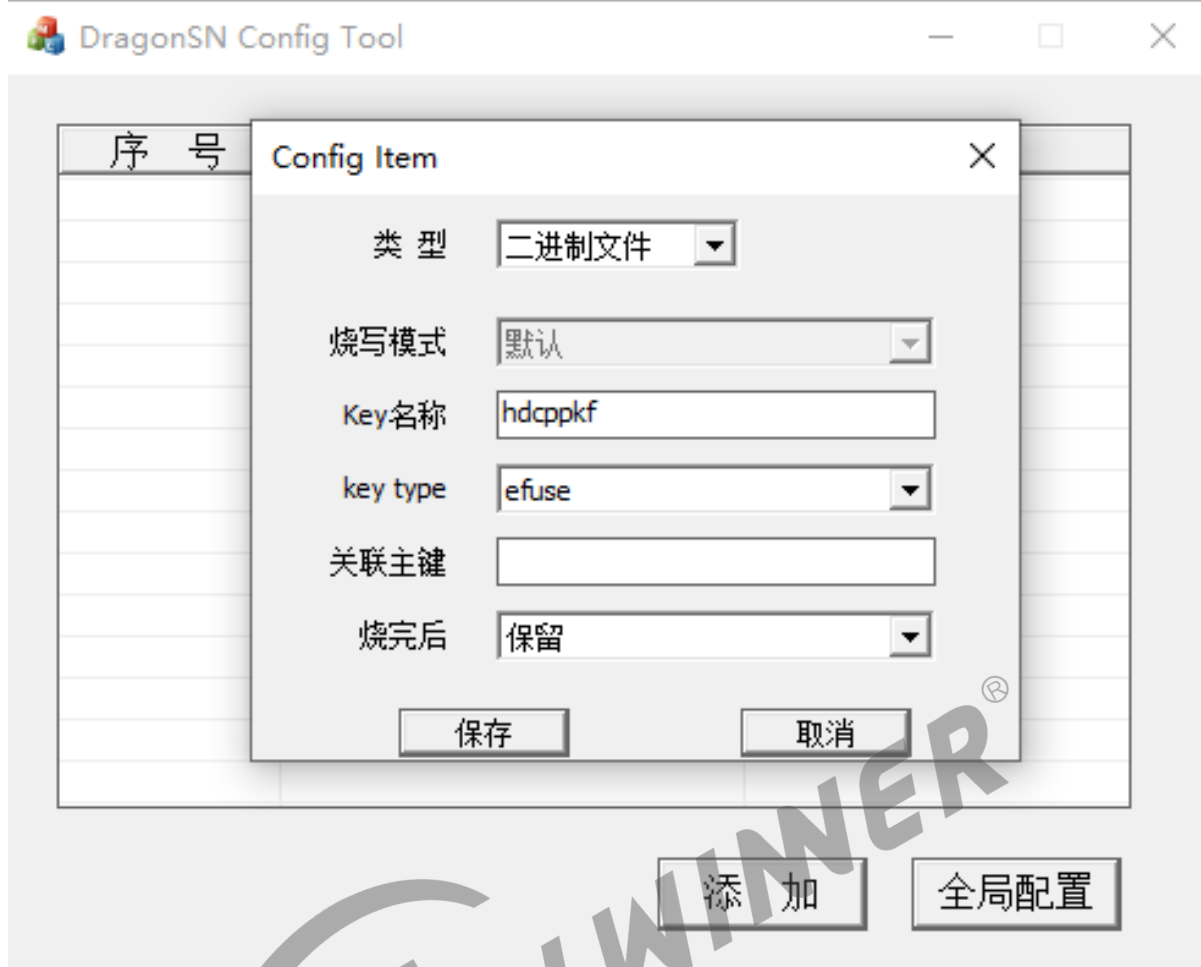


图 3-7: 配置 hdcppkf

说明

DCP 组织要求使用 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。

保留：一直烧同一个 key。

移到 _used 目录：每烧完一个 key 就会将该 key 移到同路径下的 _used 后缀文件夹中。

Step1.4: 关闭 DragonSN 配置页面，回到 DragonSN 工具主界面。

Step2: 烧录 hdc2pkf.bin

注意

每台机器仅能烧录一次 pkf 文件，请烧录前务必确认清楚烧录的文件是否正确！！

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），文件获取方式请参考[密钥配置](#)章节；

Step2.2: 将软件界面配置成与下图一致；

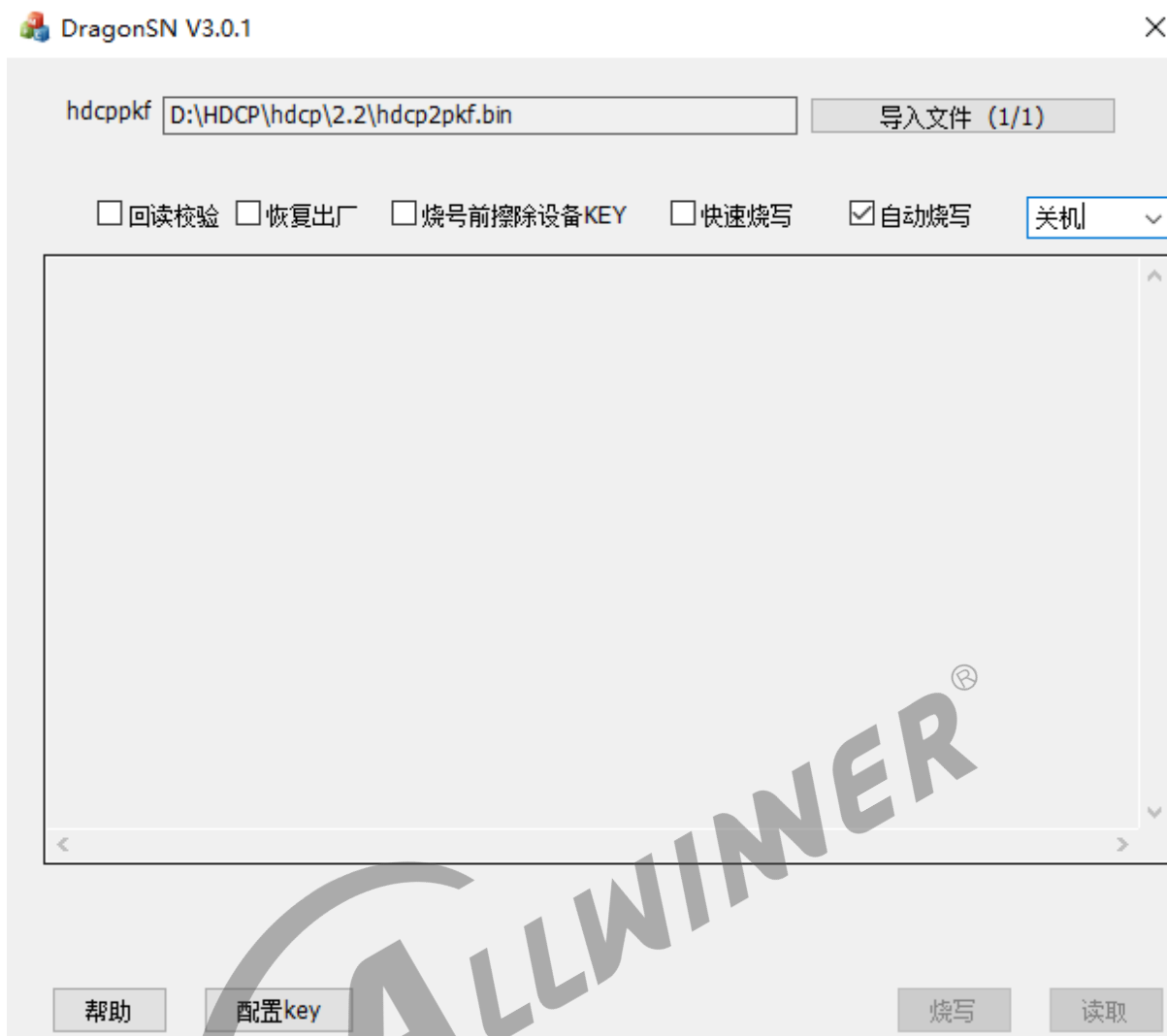


图 3-8: DragonSN_ 软件截图 2

Step2.3: 确认已经使用 USB 线连接开发板与电脑；

Step2.4: 重启开发板，工具会自动开始烧录；

Step2.5: 烧录完成后，会显示是否烧录成功，也可点击读取按键，确认烧录情况；

4 模块使用

4.1 切换显示设置

4.1.1 Android 方案

Step1: 切换分辨率、色深等设置；

可通过安卓设置界面配置显示设置，下面介绍的是命令快速设置的方法。

```
# 查看支持 RX 设备支持哪些分辨率
dispconfig -o 0 -p

# 切换分辨率为 1080p@60
dispconfig -o 0 -s 10

# 切换格式和色深为 yuv444 8bit
dispconfig -o 0 -f yuv444-8
```

参数	含义
-o	输出设备 index，-o 0 表示操作 disp0。
-p	打印 RX 设备支持哪些分辨率。
-s	设置分辨率，后面跟的值在 -p 中可以看到。
-f	设置格式和色深。(rgb-8/yuv444-8/yuv422-10/yuv420-10)
-h	打印 help 信息

Step2: 查看显示参数和显示状态；

```
cat /sys/class/disp/disp/attr/sys
```

注意：有 hdmi output 字样的才是 HDMI 的信息！

```
screen[0] -> dev[2]
de_rate 696000000 hz, ref_fps:60
mgr0: 1920x1080 fmt[rgb] cs[0x101] range[full] eotf[0x4] bits[8bits] err[0] force_sync[32] unblank direct_show[false]
iommu[1] rcq_en[1]
rcq info: rcq_irq[134] rcq_update_request[133] rcq_update_req_irq[10777] rcq_finish_irq[10777]
dmabuf: cache[4] cache_max[4] umap skip max[0]
hdmi output mode(10) fps:60.5 1920x1080
err:0 skip:0 irq:35242 vsync:343 vsync_skip:0
BUF enable ch[0] lyr[0] z[0] prem[N] a[global 255] fmt[ 1] fbd_type[none] fb[1920,1080; 960, 540; 960, 540] crop[ 0,
0,1920,1080] frame[ 0, 0,1920,1080] addr[fd800000,fd9fa400,fda78d00] flags[0x 0] metadata_flag[0x 0] trd
[0,0] depth[ 0] trans[0]
```



```
disp[0]all:128, sub:128, cur:128, free:116, skip:0
```

4.1.2 Linux 方案

Step2: 切换分辨率、色深等设置；

```
mount -t debugfs none /sys/kernel/debug
cd /sys/kernel/debug/dispdbg/

# 在 disp0 输出 1080p@60 yuv444 8bit
echo disp0 > name; echo switch1 > command; echo 4 10 1 0 0x4 0x104 2 1 0 8 > param; echo 1 > start
```

参数	参数说明	参数选择
disp0	屏幕 0	disp0 / disp1
switch1	切换调试命令	-
4	DISP_OUTPUT_TYPE_HDMI	-
10	分辨率模式	请查看驱动 enum disp_tv_mode 定义。
1	色彩格式	0: RGB / 1: YUV444 / 2: YUV422 / 3: YUV420
0	色深	0: 8bit / 1: 10bit / 3: 12bit / 4: 16bit
0x4	光电转换曲线	0x004: GAMMA2 / 0x010: SMPTE2084
0x104	颜色空间	0x104: BT601 / 0x101: BT709 / 0x107: BT2020
2	hdmi/dvi	0: default / 1: DVI / 2: HDMI
1	颜色范围	0: default / 1: full / 2: limit
0	扫描方式	0: No data / 1: OverScan / 2: UnderScan
8	纵横比	8: same as original picture / 9: 4:3 / 10: 16:9 / 11: 14:9

Step2: 查看显示参数和显示状态；

```
cat /sys/class/disp/disp/attr/sys
```

注意：有 hdmi output 字样的才是 HDMI 的信息！

```
screen[0] -> dev[2]
de_rate 696000000 hz, ref_fps:60
mgr0: 1920x1080 fmt[rgb] cs[0x101] range[full] eotf[0x4] bits[8bits] err[0] force_sync[32] unblank direct_show[false]
iommu[1] rcq_en[1]
rcq info: rcq_irq[134] rcq_update_request[133] rcq_update_req_irq[10777] rcq_finish_irq[10777]
dmabuf: cache[4] cache_max[4] umap skip[0] umap skip_max[0]
hdmi output mode(10) fps:60.5 1920x1080
err:0 skip:0 irq:35242 vsync:343 vsync_skip:0
BUF enable ch[0] lyr[0] z[0] prem[N] a[global 255] fmt[ 1] fbd_type[none] fb[1920,1080; 960, 540; 960, 540] crop[ 0,
0,1920,1080] frame[ 0, 0,1920,1080] addr[fd800000,fd9fa400,fda78d00] flags[0x 0] metadata_flag[0x 0] trd
[0,0] depth[ 0] trans[0]
disp[0]all:128, sub:128, cur:128, free:116, skip:0
```

Step3: 验证显示是否正常；

由于 Linux 可能存在没有送显的情况，此时显示是黑屏，可以通过输出 colorbar 来确认显示是否正常。


```
# 在 disp0 输出 TCON Colorbar
echo 0 > /sys/class/disp/disp/attr/disp; echo 1 > /sys/class/disp/disp/attr/colorbar

# 在 disp0 输出 DE Colorbar
echo 0 > /sys/class/disp/disp/attr/disp; echo 8 > /sys/class/disp/disp/attr/colorbar
```

4.2 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。

4.2.1 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！

4.2.2 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，附录[Linux CEC 应用程序样例](#)提供了简单测试样例。

4.3 使用 HDCP14 功能

⚠ 注意

请先查看**模块平台功能**，确认当前平台是否支持 HDCP14 功能。

4.3.1 Android 方案

Step1: 参考[配置 HDCP14](#)，完成 SDK 配置和密钥烧录；

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
```

说明

如果驱动同时配置了 HDCP14 和 HDCP22 功能，驱动会根据 RX 设备支持能力来选择开启。

例如：驱动同时配置了 HDCP14 和 HDCP22 功能，连接的屏幕也同时支持 HDCP1.4 和 HDCP2.2，执行完以上命令后会优先开启 HDCP2.2。

4.3.2 Linux 方案

请参考附录章节：[Linux HDCP 应用程序样例](#)

```
#define SYSPATH_HDCP_ENABLE "/sys/devices/virtual/hdmi/hdmi/attr/hdcp_enable"
/* 开启/关闭 HDCP1.4 */
int HdcpManager::configHdcp(bool enable)
{
    int ret = 0;

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

    return -1;
}
```

4.4 使用 HDCP22 功能

注意

请先查看[模块平台功能](#)，确认当前平台是否支持 HDCP22 功能。

4.4.1 Android 方案

Step1: 参考[配置 HDCP22](#)，完成 SDK 配置和密钥烧录；

Step2: 将自己生成的 esm.fex 替换以下路径中的文件，esm.fex 的生成方式请参考[密钥配置](#)；

```
<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
```


4.4.2 Linux 方案

完整样例请参考附录章节：[Linux HDCP 应用程序样例](#)

```
#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 AW_IOCTL_HDMI_HDCP22_LOAD_FW 1

/* 开启/关闭 HDCP2.2 */
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;
}

/* 加载 HDCP2.2 固件 */
#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;
}
```


4.5 获取 HDCP 信息

⚠ 注意

请先查看[模块平台功能](#)，确认当前平台是否支持 HDCP 功能。

4.5.1 Android 方案

```
hdcpctl -p
```

```
hdcp authorized status: AUTHORIZED  
hdcp level: HDCP_V1
```

信息类型	信息内容	含义
authorized status	ERROR	加密失败
	UN_AUTHORIZED	未加密
	AUTHORIZED	加密成功
level	HDCP_UNKNOWN	未知类型
	HDCP_V1	HDCP1.4
	HDCP_V2_2	HDCP2.2

4.5.2 Linux 方案

完整样例请参考附录章节：[Linux HDCP 应用程序样例](#)

```
#define SYSPATH_HDCP_STATUS "/sys/devices/virtual/hdmi/hdmi/attr/hdcp_status"  
#define SYSPATH_HDCP_TYPE "/sys/devices/virtual/hdmi/hdmi/attr/hdcp_type"  
HdcpManager::HdcpLevel HdcpManager::getConnectedHdcpLevel() const  
{  
    char type = 0;  
  
    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;
}

```

4.6 增加特殊分辨率

说明

如果需要特殊分辨率在 uboot 显示 LOGO，则还需要修改 uboot 代码。不过因为整体修改与 kernel 的修改差不多，所以本章节不会重复描述，只以 kernel 代码修改为例。

下面以添加 1080x1920 特殊分辨率为例：

Step1: 增加 tv_mode;

```

// 文件路径: <sdk>/longon/bsp/include/video/sunxi_display2.h

enum disp_tv_mode {
    ...
    DISP_TV_MOD_1080_1920P_60HZ = 0x28,
    ...
}

```

Step2: 增加 vic_code;

```

// 文件路径: <sdk>/longon/bsp/drivers/video/sunxi/disp2/hdmi2/aw_hdmi_core/aw_hdmi_core.h

enum HDMI_VIC {
    ...
    HDMI_VIC_1080x1920P60 = 0x203, /* 此处的值只做示例，找一个大于 0x200 且未被占用的值即可。 */
    ...
}

```

Step3: 建立 tv_mode 与 vic_code 之间的映射;

```

// 文件路径: <sdk>/longon/bsp/drivers/video/sunxi/disp2/hdmi2/aw_hdmi_core/aw_hdmi_core.c

static struct disp_hdmi_mode hdmi_mode_tbl[] = {
    ...
    {DISP_TV_MOD_800_1280P_70HZ, HDMI_VIC_800x1280P70, },
    ...
}

```

Step4: 增加特殊分辨率的 timing;

```

// 文件路径: <sdk>/longon/bsp/drivers/video/sunxi/disp2/hdmi2/aw_hdmi_core/dw_hdmi/dw_edid.c

static supported_dtd_t_dtd[] = {
    ...
    {60000, {0x203, 0, 0, 0, 142250, 0, 1080, 132, 0, 9, 64, 4, 1, 1920, 36, 0, 16, 16, 4, 0}},
    ...
}

```


}

参数	意义	备注
60000	refresh_rate	该值等于 1Hz * 1000，这里就相当于 60 Hz 刷新率。
0x203	mCode	VIC Code，此处需要与自定义的 vic code 一致。
0	mLimitedToYcc420	该值表示是否只支持 YCbCr4:2:0 显示
0	mYcc420	该值表示是否支持 YCbCr4:2:0 显示
0	mPixelRepetitionInput	是否为像素重复输入 (开启像素重复功能)
142250	mPixelClock	像素时钟频率，单位是 1 kHz
0	mInterlaced	1: interlaced, 0: progressive
1080	mHActive	行有效像素
132	mHBlanking	行总的消隐像素
0	mHBorder	
9	mHImageSize	用于指示屏幕是 16:9 / 4:3
64	mHSyncOffset	行消隐的前边界 (front porch)
4	mHSyncPulseWidth	行消隐的同步区间 (sync time)
1	mHSyncPolarity	0: active low, 1: active high
1920	mVActive	列有效像素
36	mVBlanking	列总的消隐像素
0	mVBorder	
16	mVImageSize	用于指示屏幕是 16:9 / 4:3
16	mVSyncOffset	场消隐的前边界 (front porch)
4	mVSyncPulseWidth	场消隐的同步区间 (sync time)
0	mVSyncPolarity	0: active low, 1: active high

Step5: 增加长和宽的获取接口;

```
// 文件路径: <sdk>/longon/bsp/drivers/video/sunxi/disp2/disp/de/disp_display.c

s32 bsp_disp_get_screen_width_from_output_type(u32 disp, u32 output_type, u32 output_mode)
{
    ...
    case DISP_TV_MOD_1080_1920P_60HZ:
        width = 1080;
        height = 1920;
        break;
    ...
}

s32 bsp_disp_get_screen_height_from_output_type(u32 disp, u32 output_type, u32 output_mode)
{
    ...
    case DISP_TV_MOD_1080_1920P_60HZ:
        width = 1080;
        height = 1920;
        break;
    ...
}
```


Step6: 增加 cea_code 和 hdmi_code 校正;

```
// 文件路径: <sdk>/longon/bsp/drivers/video/sunxi/disp2/hdmi2/aw_hdmi_core/aw_hdmi_core.c

static void _edid_set_video_prefered(sink_edid_t *sink_cap, dw_video_param_t *pVideo)
{
    ...
    if ((pVideo->mDtd.mCode == 0x201)
        || (pVideo->mDtd.mCode == 0x202)
        || (pVideo->mDtd.mCode == 0x203)) { /* 将新增的 VIC Code 加入进来 */
        pVideo->mCea_code = 0;
        pVideo->mHdmi_code = 0;
        return;
    }
    ...
}
```

Step7: 修改 phy 参数;

Step7.1: 请联系全志 AE 同事, 并提供特殊分辨率的 timing。

Step7.2: 根据全志 AE 同事提供的补丁, 修改相应代码。

Step8: 显示测试, 上电启动后输入以下命令测试;

```
mount -t debugfs none /sys/kernel/debug
echo switch > /sys/kernel/debug/dispdbg/command
echo disp0 > /sys/kernel/debug/dispdbg/name
echo 4 40 > /sys/kernel/debug/dispdbg/param # 40 是以 Step1 tv_mode 中的 0x28 转换 10 进制得到的。
echo 1 > /sys/kernel/debug/dispdbg/start

# 查看是否切到了这种分辨率
cat /sys/class/disp/disp/attr/sys

# 输出 colorbar 检查显示是否正常
echo 0 > /sys/class/disp/disp/attr/disp; echo 1 > /sys/class/disp/disp/attr/colorbar
```


5 调试方式

说明

由于驱动版本的升级，部分调节点不存在或者信息内容有变化，但这并不影响 HDMI 功能。若有需求可联系对应的 FAE 同事协助升级即可。

5.1 查看 HDMI 模块加载成功

方法 1：查看相关 sysfs 路径

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

此路径存在说明 HDMI 模块加载成功。反之则 HDMI 模块加载不成功。请查阅模块配置章节

方法 2：查看相关设备节点

```
/dev/hdmi
```

此路径存在说明 HDMI 模块加载成功。反之则 HDMI 模块加载不成功。请查阅模块配置章节

5.2 查看 HDMI 连接状态

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

在输出日志中找到 **HPD** 字样

如果打印 HPD: 1，表明 HDMI 检测到插入；

如果打印 HPD: 0，表明 HDMI 未检测到插入；

5.3 模拟 HDMI 连接状态

```
# 强制 HPD 拔出  
echo 0x10 > /sys/class/hdmi/hdmi/attr/hpd_mask  
# 强制 HPD 插入  
echo 0x11 > /sys/class/hdmi/hdmi/attr/hpd_mask  
# 取消强制状态  
echo 0x0 > /sys/class/hdmi/hdmi/attr/hpd_mask
```


5.4 查看 rxsense 状态

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

在输出日志中找到 **rxsense** 字样

如果打印 rxsense: 1, 表明 HDMI 检测到 rxsense 有正常拉高, TMDS 数据能正常传输。

如果打印 rxsense: 0, 表明 HDMI 检测到 rxsense 有异常未拉高, TMDS 数据不能传输。

5.5 查看输出的分辨率

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

在输出日志中找到 **video format** 字样, 后面跟随的就是当前的输出分辨率。

5.6 查看输出的颜色格式

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

在输出日志中找到 **color format** 字样, 后面跟随的就是当前的输出颜色格式。

5.7 查看输出的颜色深度

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

在输出日志中找到 **color depth** 字样, 后面跟随的就是当前的输出颜色深度。

5.8 查看输出的信号格式

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

在输出日志中找到 **tmads mode** 字样, 后面跟随的就是当前的输出信号格式, 会显示 HDMI 还是 DVI 信号。

5.9 查看设备支持的分辨率

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

在输出日志中找到 **Video Mode** 字样，后面跟随的就是当前设备支持的分辨率。

5.10 查看设备 Audio 信息

⚠ 注意

H728 Android U 版本新增，以前的版本可能不支持。

```
cat /sys/class/hdmi/hdmi/attr/audio_dump
```

在日志中可以查看驱动 Audio core 层和 lowlevel 层的信息。

5.11 查看设备 CEC 信息

⚠ 注意

H728 Android U 版本新增，以前的版本可能不支持。

```
cat /sys/class/hdmi/hdmi/attr/cec_dump
```

在日志中可以查看驱动 CEC core 层和 lowlevel 层的信息。

5.12 获取设备 EDID 数据

```
cat /sys/class/hdmi/hdmi/attr/edid > /xxx/edid.bin
```

将设备的 EDID 数据转成相关的二进制 edid.bin 文件。

5.13 替换设备 EDID 数据

将设备的 EDID 更换为另一台设备的 EDID 信息，本方法一般用于排查是否由 EDID 引起的问题。

Step1: 将需要替换的 EDID 数据存入指定路径。如/tmp/


```
cd /tmp # 注：需进入到EDID数据存放的路径！  
cat edid_test.bin > /sys/class/hdmi/hdmi/attr/edid
```

Step2: 重新拔插 HDMI 线，随后 HDMI 驱动将获取到的 EDID 数据为 edid_test.bin。

5.14 查看 HDCP 开启状态

```
cat /sys/class/hdmi/hdmi/attr/hdcp_enable  
# 0: 表示HDCP关闭  
# 1: 表示HDCP开启
```

5.15 查看 HDCP 认证状态

```
busybox hexdump /sys/class/hdmi/hdmi/attr/hdcp_status  
# 0: 表示HDCP没有被使能  
# 1: 表示HDCP正在认证  
# 2: 表示HDCP认证失败  
# 3: 表示HDCP认证成功
```

5.16 查看 HDCP 认证类型

```
busybox hexdump /sys/class/hdmi/hdmi/attr/hdcp_type  
# 00ff: DDC 读取失败，表示 rx 设备不支持 HDCP。  
# 0000: 表示当前认证类型为 HDCP14  
# 0001: 表示当前认证类型为 HDCP22
```

5.17 日志抓取

Step1: 复现问题之前，先将串口连接好并开始抓日志；

Step2: 命令行输入；

```
dmesg -n8  
echo 8 > /sys/class/hdmi/hdmi/attr/debug
```

Step3: 开始复现问题，复现到问题后结束串口日志抓取；

5.18 近期日志抓取

⚠ 注意

H728 Android U 版本新增，以前的版本可能不支持。

基于问题出现时可能没有开启最高打印等级，也没有保存日志，所以 HDMI 引入了 ring-buffer 存储近期的 HDMI 打印。

PS：如果问题发生时间过久，可能已经被覆盖，此时拿出的打印无法还原现场发生的情况。

```
# 直接显示
cat /sys/kernel/debug/hdmi/hdmi_log

# 保存到文件
cat /sys/kernel/debug/hdmi/hdmi_log > /sdcard/hdmi.log
```

5.19 获取所有寄存器的打印

⚠ 注意

H728 Android U 版本新增，以前的版本可能不支持。

```
for i in $(seq 1 $(cat /sys/class/hdmi/hdmi/attr/reg_dump | awk -F ' ' END '{print $1}' )); do echo $i > /sys/class/hdmi/hdmi/attr/reg_dump; cat /sys/class/hdmi/hdmi/attr/reg_dump; done
```

当前命令可以将寄存器都打印出来，可用于自行 debug 或反馈给全志 AE 同事做分析。

5.20 输出 HDMI Pattern

⚠ 注意

H728 Android U 版本新增，以前的版本可能不支持。

```
# Red Pattern
echo 1 > /sys/class/hdmi/hdmi/attr/pattern
# Green Pattern
echo 2 > /sys/class/hdmi/hdmi/attr/pattern
# Blue Pattern
echo 3 > /sys/class/hdmi/hdmi/attr/pattern
# 关闭 Pattern
echo 0 > /sys/class/hdmi/hdmi/attr/pattern
```

主要用于厘清问题边界，排查思路如下：

- 设置 HDMI Pattern 后显示正常，问题大概率在 HDMI 及以上的通路。（应用、DE、TCON、HDMI 自身寄存器设置）
- 设置 HDMI Pattern 后显示异常，问题大概率在 HDMI 的后端。（HDCP、PHY、HDMI RX 端）



6 FAQ

6.1 无信号问题

6.1.1 排查是否跟 Rx 设备相关

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

将 HDMI 线接到另一台设备上，看显示是否正常。

如果正常，说明 Tx 端是正常输出。问题大概率是 Rx 端的问题，使用其他对比设备输出相同的信号格式，定位是否是 Rx 的问题。

⚠ 注意

屏蔽 HPD 的功能仅限当分辨率在 4K60 以下使用！

6.1.2 排查 Tx 端问题模块

```
cat /sys/class/disp/disp/attr/sys  
# hdmi output mode(35)  fps:60.6  4096x2160
```

如果有关键字 “hdmi output mode”，说明已正常切到 HDMI 输出。

反之需要查显示为什么没有切到 HDMI。

6.1.3 排查 Rx 端是否支持

1、直接查看驱动调试节点

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

查看 EDID 解析出来的 video mode，如果实际输出的分辨率不包含在 video mode 里，那么需要调整相关的输出策略。

2、使用 EDID Manager 确认显示器所支持的 video format。


```
# 提取获取到的EDID (使用HDMI线连接显示屏)
cat /sys/class/hdmi/hdmi/attr/edid > /sdcard/xxx_edid.bin

# 使用adb传输到PC
adb pull /sdcard/xxx_edid.bin ./
```

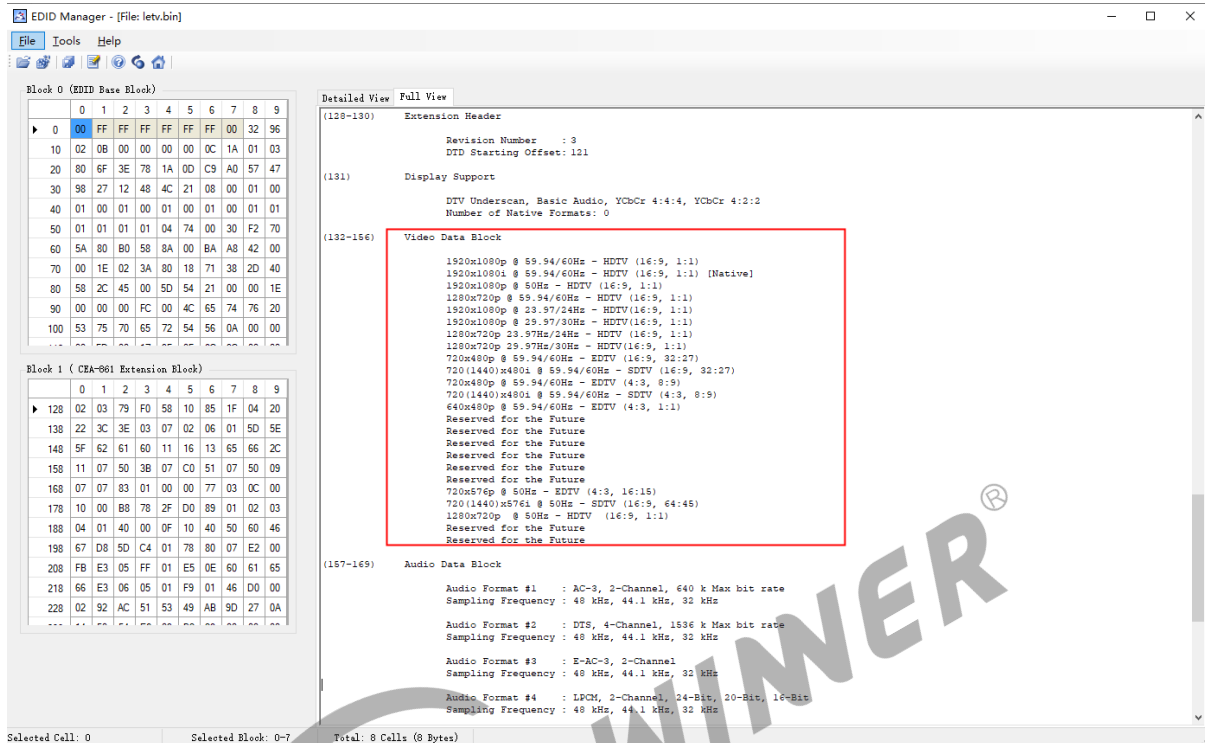


图 6-1: 查看 video_data_block

6.1.4 排查 HDMI 的输出状态

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

查看输出的信息，看是否均正常。(可以使用一台正常显示的小机的输出来对比)

6.1.5 排查硬件特性

使用示波器测量 5V，HPD，Tmds 相关 Pin 是否正常

- 如果没有 5V，那么需要 Tx 端排查硬件电路；
- 如果没有 HPD 但有 5V，那么基本定性是 Rx 端问题，未能拉高 HPD；
- 如果 TMDS 幅值没拉高，那么基本定性是 Rx 端问题，未能拉高 TMDS；
- 测量 TMDS CLK Pin 的时钟频率，看是否与当前的符合。

例如：1080P 输出的 TMDS CLOCK = 148.5Mhz，测量实际 tmds clock 传输的频率是否在此值。上下浮动 1/2Mhz 还属正常。

6.2 黑屏问题

6.2.1 排查是否跟 Rx 相关

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

拔插接到另一台设备上，看显示是否正常

如果正常，说明 Tx 端是正常输出，问题大概率是 Rx 端的问题

⚠ 注意

屏蔽 HPD 的功能仅限当分辨率在 4K60 以下使用！

6.2.2 排查 Tx 端问题模块

1、查看是否显示 pattern，如果能正常显示，说明大概率是应用 -> DE -> TCON 通路出了问题。

```
# 开启colorbar
echo 1 > /sys/class/disp/disp/attr/colorbar
# 关闭colorbar
echo 0 > /sys/class/disp/disp/attr/colorbar
```

2、查看是否有图层设置下来

```
cat /sys/class/disp/disp/attr/sys
```

关注 BUF 关键字，如果没有，则证明没有图层数据送下来，需要排查没有正常送显的原因。

3、使用 screencap 查看送下来的显示是否正常（安卓平台）

```
screencap -p /sdcard/1.png
```

4、使用回写判断送下来的显示内容是否正常（安卓平台）

```
hwcdebug --capture /sdcard/decapture.png
```

6.2.3 排查是否跟 HDCP 相关

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

如果能正常显示，说明 Rx 可能不支持 HDCP 功能，需要关闭此功能。

6.2.4 HDMI 排查-寄存器对比

```
# 如果 /sys/class/hdmi/hdmi/attr/ 下有 reg_dump 节点。
echo 3 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump
echo 4 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump
echo 5 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump
echo 9 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump
echo 10 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump
echo 11 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump
echo 12 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump

# 如果 /sys/class/hdmi/hdmi/attr/ 下没有 reg_dump 节点。
echo 0x200,0x8 > /sys/class/hdmi/hdmi/attr/read
echo 0x800,0x8 > /sys/class/hdmi/hdmi/attr/read
echo 0x1000,0x21c > /sys/class/hdmi/hdmi/attr/read
echo 0x3000,0x39 > /sys/class/hdmi/hdmi/attr/read
echo 0x4000,0xb > /sys/class/hdmi/hdmi/attr/read
echo 0x4100,0x1c > /sys/class/hdmi/hdmi/attr/read
echo 0x5000,0x20 > /sys/class/hdmi/hdmi/attr/read
echo 0x7900,0xf > /sys/class/hdmi/hdmi/attr/read
```

对比正常与异常环境下的寄存器差异，并尝试写入更正看是否正常。

6.3 无声问题

6.3.1 排查 HDMI 工作模式

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

查看关键字：tmads mode，看是否工作在 DVI 模式下。

1、如果为 DVI 模式：

- 显示屏支持 HDMI 模式，则需要按 **DVI 问题** 排查；
- 显示屏**只支持** DVI 模式，没有声音是正常现象；

2、如果为 HDMI 模式，则需继续往下排查。

6.3.2 排查是否跟 Rx 相关

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

然后拔插到其他设备上 看 Audio 是否正常。

- 如果正常，大概率是 Rx 端问题。

- 进一步排查，同条件场景下使用另一台设备接入该 Rx，看 Audio 是否正常。
 - ◇ 如果正常，问题就复杂。应该是特殊的兼容性问题，此类问题一般需要能有快速复现的场景并不断进行实验才能得出解决方案。
 - ◇ 如果异常，那么足够说明该 Rx 的问题。
- 如果异常，大概率是 Tx 端问题。
 - 进一步排查，判断是否与 EDID 相关

```
# 当前设备下
cat /sys/class/hdmi/hdmi/attr/edid > /data/edid.bin
# 拷贝出 /data/edid.bin
# 在对比设备下
cd /data/
cat edid.bin > /sys/class/hdmi/hdmi/attr/edid
echo 1 > /sys/class/hdmi/hdmi/attr/edid_test
# 拔插HDMI线，使之后HDMI都使用该edid
```

- ◇ 如果正常，可能与 EDID 无关;
- ◇ 如果异常，那么跟 EDID 无关，将 EDID 反馈回来由我们继续分析问题点;

6.3.3 Audio 排查-工作状态

1、通过 tinyalsa 工具做软件层面排查。（工具包请联系全志 AE 同事获取）

- 将 tinyplay（播放音频工具包）、tinycap（录制音频工具）、tinymix（通路控制工具）推到小机上

```
# 以 tinyplay 为例
# Linux固件
adb push tinyplay /usr/bin
# Android固件
adb push tinyplay /vendor/bin
```

- 修改权限

```
chmod 777 tiny*
```

- 获取声卡号

```
cat /proc/asound/cards
```

- 打开 tx hub


```
tinymix -D [hdmi声卡号] "tx hub mode" 1
```

- 边播放边会录音频，结束录制后查看音频是否正常

```
tinypay [测试音频.wav] -D [hdmi声卡号]  
tinycap [录音音频.wav] -D [hdmi声卡号] -T 10
```

2、硬件层面排查

- 获取声卡号

```
cat /proc/asound/cards
```

- 使用 tinypay 播放正弦波音频文件

```
tinypay [测试音频.wav] -D [hdmi声卡号]
```

- 使用示波器测量 I2S data 线，看是否有正弦波波形输出。

6.3.4 HDMI 排查-LOG 对比

开启 audio 配置打印输出，比对正常与异常时的 audio 配置打印是否有差异。

```
echo 3 > /sys/class/hdmi/hdmi/attr/debug  
dmesg -n8
```

6.3.5 HDMI 排查-寄存器对比

```
# 如果 /sys/class/hdmi/hdmi/attr/ 下有 reg_dump 节点。  
echo 6 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump  
echo 7 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump  
echo 8 > /sys/class/hdmi/hdmi/attr/reg_dump;cat /sys/class/hdmi/hdmi/attr/reg_dump  
  
# 如果 /sys/class/hdmi/hdmi/attr/ 下没有 reg_dump 节点。  
echo 0x3100,0x5 > /sys/class/hdmi/hdmi/attr/read  
echo 0x3200,0x8 > /sys/class/hdmi/hdmi/attr/read  
echo 0x3500,0x7 > /sys/class/hdmi/hdmi/attr/read
```

对比正常与异常环境下的寄存器差异，并尝试写入更正看是否正常。

6.3.6 HDMI 排查-尝试恢复

- 拔插 HDMI 线，看是否能正常恢复。
- 尝试 reset 相关模块，看是否能正常恢复。


```
echo 0x4002,0x1 > /sys/class/hdmi/hdmi/attr/read  
# 将读出来的数据的 bit4 改为 1  
echo 0x4002,0xxx > /sys/class/hdmi/hdmi/attr/write # xx值为上述修改后的值
```

6.4 颜色问题

6.4.1 排查 HDMI 工作模式

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

查看关键字 TmdsMode 和 PixelFormat

如果是 TmdsMode 是 DVI 模式，那么 PixelFormat 不是 RGB，那么说明输出的 format 是错误的，引起偏色的问题在此。

6.4.2 排查是否跟 Rx 相关

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

拔插接到其他设备上，看颜色显示是否正常。

正常说明大概率是 Rx 的问题，反之是 Tx 的问题。

6.4.3 排查 Tx 端问题模块

使用 DE 回写功能，查看回写出来的数据是否正常。

如果正常那么偏色问题是在 HDMI 端引起的，反之是在 DE 端引起的。

6.4.4 排查颜色格式

先获取当前的信号输出信息

```
cat /sys/class/disp/disp/attr/sys  
# screen[1] -> dev[3]  
# de_rate 600000000 hz, ref_fps:60  
# mgr1: 4096x2160 fmt[yuv420] cs[0x101] range[limit] eotf[0x4] bits[8bits]
```

可知：使用的是 4K60 分辨率，YUV420 8Bit。

借助 debugfs 切换 format，看偏色问题是否仅限指定 format 下，注：仅改单一的 format 变量。


```
# 前提是固件有开启 debugfs 功能
mount -t debugfs none /mnt
cd /mnt/debugfs
# 需要先确定 HDMI 是在 disp0 还是 disp1 上。通过cat /sys/class/disp/disp/attr/sys可以看到。
echo disp0 > name
echo switch1 > command
echo 4 10 1 1 0x4 0x104 0 0 0 8 > param
echo 1 > start

### param参数解析
- Arg1: 显示接口代号, 1: LCD显示接口 2: tv显示接口 4: HDMI显示接口 8: VGA显示接口 16: vdpo接口 32: edp显示接口。
- Arg2: TVMode, 显示分辨率代号。可以在 include/video/sunxi_display2.h 中的 enum disp_tv_mode 中查到
- Arg3: color format。
  - 0: RGB
  - 1: YUV444
  - 2: YUV422
  - 3: YUV420
- Arg4: color depth
  - 0: 8Bit
  - 1: 10Bit
  - 3: 12Bit
  - 4: 16Bit
- Arg5: EOTF
- Arg6: color space
  - 0x104: BT601
  - 0x101: BT709
  - 0x107: BT2020
- Arg7: Mode
  - 1: DVI Mode
  - 0: HDMI Mode
```

6.5 DVI 问题

HDMI 和 DVI 信号最大的区别在于是否在 TMDS 上传输 audio 数据。

输出 HDMI 还是 DVI 信号由 displayd 决定, 主要逻辑在于判断当前设备的 EDID 是否存有 audio data block。

如果强制走 HDMI 信号, 将带来:

- 1、HDMI CTS 认证无法通过;
- 2、可能会造成一些 DVI 设备显示异常, 如无信号等;

6.5.1 排查 EDID 解析

- 1、直接在驱动调试节点中查看

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

关注 audio 关键字的打印, 看是否有解析出支持的音频格式。

2、通过 EDID Manager 确认驱动解析 audio data block 是否正确。

```
cat /sys/class/hdmi/hdmi/attr/edid > /data/edid.bin
```

- 如果 edid.bin 的数据全都是 0x00，那么说明 EDID 读取异常，需要针对问题场景进行分析。
- 如果 edid.bin 仅无 audio data block，但有 video data block。设备又明确标注说明支持 HDMI，那么不排除针对 audio data block 的解析有问题。

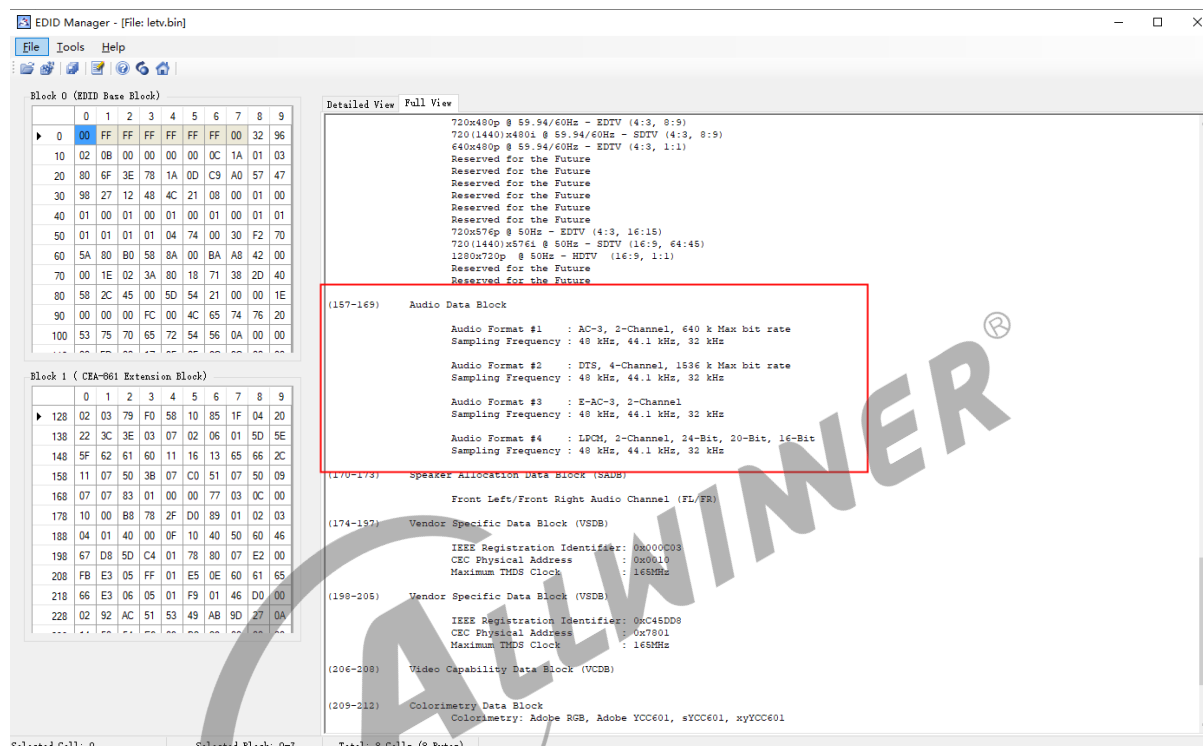


图 6-2: 查看 audio_data_block

7 附录

7.1 Linux CEC 应用程序样例

⚠ 注意

以下测试代码仅适用于新版驱动程序，如果发现只有 /dev/cec 节点，则证明为旧版驱动。

```
#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("\targc[1] = log_addr: set default cec logical address.\n");
    printf("\targc[1] = standby: playback standby -> tv standby.\n");
    printf("\targc[1] = wakeup: playback wakeup -> tv wakeup.\n");
    printf("\targc[1] = tv_standby: tv standby -> playback standby.\n");
    printf("\targc[1] = get_language: get tv menu language.\n");
    printf("demo: app_cec log_addr\n");
}

/**
 * @desc: app test case for cec get tv language (RX cases)
 * @return: 0 - success
 *         -1 - failed
 */
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-->0)
{
    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 - faield
 */
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;
}
```

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

    int length = read_from_file(fw, esming, 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)esming;
        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 */

```

hdcpool.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 {} \;
```



8 结束语

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：

- 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。
- 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。