



D1-H Linux HDMI20 开发指南

版本号: 1.0
发布日期: 2021.4.01

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.4.01	AWA0962	1. 创建该文档



目 录

1 概述	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
2 相关概念	2
2.1 模块介绍	2
2.2 术语、定义、缩略词	2
3 实现背景	3
3.1 参考资料	3
3.2 平台差异	3
4 驱动开发设计	4
4.1 任务安排	4
4.1.1 整合 IP 厂商提供的代码	4
4.1.2 场景预判	4
4.2 开发难点分析	4
4.2.1 IP 厂商提供源码特点分析	4
5 驱动配置	7
5.1 内核 dts 配置	7
5.2 板级配置	7
5.2.1 驱动配置	8
6 驱动方案设计	9
6.1 初始化流程	9
6.2 插拔流程	10
6.3 分辨率切换	11
6.4 Audio 输出设计	16
6.5 休眠唤醒流程	18
6.6 HDCP 功能实现	19
6.7 CEC 流程设计	19
6.7.1 CEC 逻辑地址的设置原则 (HDMI CTS 标准)	22
6.7.2 上/下/左/右/返回键功能	22
6.7.3 休眠状态下 cec 交互情况	22
6.7.3.1 休眠前, hdmi_suspend 应该做的动作	23
6.7.4 标准 CEC 驱动方案	24
6.7.4.1 方案	24
6.8 log 管理方案	24
7 结束语	25

插图

4-1 IP 原厂代码架构	5
4-2 驱动框架梗概	6
6-1 驱动初始化	9
6-2 插拔流程	10
6-3 切换分辨率	12
6-4 2D 与 3D 比较	13
6-5 422 打包格式	14
6-6 detail timing	15
6-7 map code	16
6-8 audio 输出流程	16
6-9 休眠唤醒	18
6-10 HDCP 流程	19
6-11 CEC 流程 1	20
6-12 CEC 流程 2	21
6-13 RX 处理消息流程	21

1 概述

1.1 编写目的

本设计说明文档目的在于说明 hdmi2.0 驱动模块的设计考虑以及开发过程遇到的问题及其解决方案的总结。

1.2 适用范围

D1-H 方案.

1.3 相关人员

项目组成员，以及显示相关人员。

2 相关概念

2.1 模块介绍

HDMI (High-Definition Multiface Interface) 是 Hitachi, Panasonic, Philips, Silicon-Image, Sony, Thomson, Toshiba 几家公司共同发布的一款音视频传输协议，主要用于 DVD, 机顶盒等音视频 source 到 TV, 显示器等 sink 设备的传输。传输基于的是 TMDS(Transition Minimized Differential Signaling) 协议。此外，使用 TMDS 也是 DVI 标准的主要特点。

本驱动模块在 HDMI 输出场景下使用，可以同时兼容 hdmi2.0 和 hdmi1.4

2.2 术语、定义、缩略词

缩略词	含义
HDMI	High Definition Multimedia Interface 即高清晰度多媒体接口
TMDS	Transition Minimized Differential signal 最小化传输差分信号
Source、Sink	信源端/接收端
HBR	全称 High Bit Rate, 高比特率型 audio 数据（透传的一种）
DDC	全称 Display Data Channel, 显示数据信道，一种 I2C 总线
AKSV	HDCP1.4 Tx 端的 Key Selection Vector, 即 HDCP Tx 端的密钥选择向量，在 hdcp 认证过程中传输给 Rx 端
TE	Testing Equipment, 测试设备
DUT	Device Under Testing, 被测试设备
HEAC	HDMI Ethernet and Audio Return Channel 以太网和音频返回通路
HPD	Hot Plug Detect 即热插拔查询控制
VIC	Video Identification Code 视频识别码
OUI	Organizationally unique identifier 组织唯一标识符（IEEE 给 HDMI 这个器件的编号，1.4 和 2.0 是不同的）
EDID	Extended Display Identification Data 外部显示设备表示数据
VSDB	Vendor Specific Data Block 厂商定义块

3 实现背景

3.1 参考资料

资料名称	作者	描述
《HDCP Specification 1.4》	Digital Content Protection LLC	HDcp1.4 协议
《HDCP on HDMI Specification Rev2_2 Final 1.pdf》	Digital Content Protection LLC	HDCP2.2 协议
《HDMI Specification2.0.pdf》	HDMI 协会	HDMI2.0 协议
《HDMI_Spec_1.4.pdf》	HDMI 协会	HDMI1.4 协议

3.2 平台差异

D1-H 方案 PHY 层是全志自研。

4 驱动开发设计

4.1 任务安排

4.1.1 整合 IP 厂商提供的代码

- (1) 完成 IP 厂商提供的代码的修改，做成 linux 驱动代码，使其能在 linux 系统上运行；
- (2) 对 IP 厂商提供的代码进行裁剪以符合我们对 hdmi2.0 驱动的需求；
- (3) 对 IP 厂商提供的代码进行重新的架构设计，以便后期代码的移植、修改与维护；
- (4) 加入调试信息，方便后续调试与问题定位。

4.1.2 场景预判

- (1) HDMI 运行的流程较为复杂，对 hdmi2.0 的整个运行流程进行详细的规划设计，是为了使 hdmi 的流程更加清晰，防止 bug 的出现，特别是多线程调用时出现的 bug。
- (2) 用户空间对驱动功能的需求的多样性和 hdmi 需要兼容多种场景的需求要求我们必须要进行详细的流程设计；

4.2 开发难点分析

4.2.1 IP 厂商提供源码特点分析

- (1) IP 厂商提供的 HDMI2.0 的代码分为用户空间和内核空间两个部分，其中代码的核心部分位于用户空间，而内核空间仅提供 io_ctrl 等简单的与底层交互的接口。因此我们的主要工作之一是将这些用户空间的代码转化为 linux 驱动代码；
- (2) IP 厂商提供的 HDMI2.0 的代码含有大量的浮点操作，但目前我们内核的编译器一般不支持浮点操作，另外，从代码规范的角度上来讲，我们不希望出现浮点操作，我们需要将其转化为整型操作。所采用的方法是这些出现在代码的浮点数乘以 1000 或者 100 等，使其变为整型数（当然其单位也相应地发生变化）。
- (3) IP 厂商提供的代码分为三层 app 层、api 层、driver：

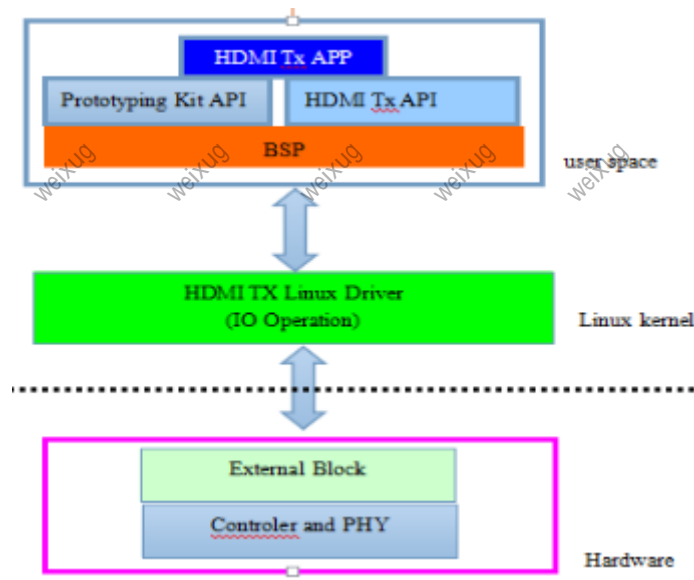


图 4-1: IP 原厂代码架构

说明：（1）由上图可知 IP 厂商提供的原生代码的核心部分在用户空间实现，因 linux kernel 部分，仅仅是提供供 app 层调用的 IO 接口和响应热插拔中断请求。

（2）对于 user space 的代码，最核心的部分由三部分组成：

- a.HDMI Tx App 层：包含 hdmi 初始化函数，用户参数配置接口（用户可以通过配置不同的 video,audio,hdcp 参数，使 hdmi 输出用户配置的输出），测试函数；
- b.HDMI Tx API 层：操作 IP 的 controller 和 PHY 的核心 api 程序，供 App 层调用；
- c.Prototyping Kit API 层：IP 厂商他们自己平台的一些可以操作到系统资源（比如系统时钟或者图形生成器）的 api 程序，供 App 层调用，在代码整合时我们需要把这一部分代码去掉；

为了最大程度的复用 IP 厂商提供的代码，减少代码修改的工作量以及为了代码编译维护，HDMI2.0 驱动的代码架构也采用了层次分明的分层架构（分为 driver 层、core 层、api 层）：

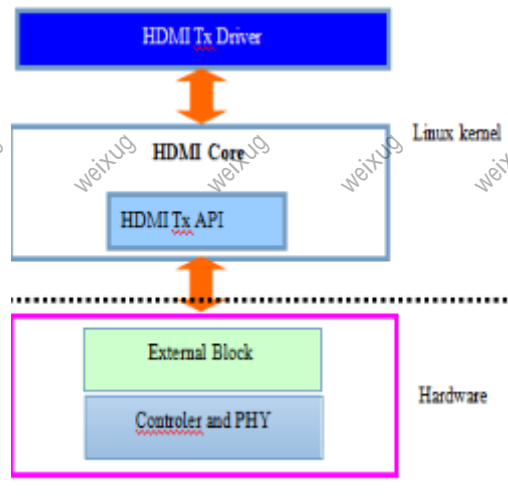


图 4-2: 驱动框架梗概

说明：

a.HDMI Tx Driver: 主要包含 hdmi 驱动的基本的 linux 驱动框架、hdmi 驱动的对外接口，与 linux 相关的操作接口主要集中于这一层；

b.HDMI Core: 包含了 HDMI2.0 主要操作流程。

c.HDMI Tx API: 集中实现了 HDMI2.0 IP 的一个个小功能，与硬件相关。

在开发的过程中，我们希望将 driver 层和 core 层做得更加健壮，这样的话以后如果要做 hdmi2.1 甚至 hdmi3.0, 这两层都可以复用，driver 层和 core 层包含了很多经过大量的兼容性测试后得出的策略，如果能得到复用，那么在开发新的 hdmi IP 的驱动的时候，就可以避免很多兼容性问题。

5 驱动配置

5.1 内核 dts 配置

dts:

```
hdmi: hdmi@06000000 {
    compatible = "allwinner,sunxi-hdmi";
    reg = <0x0 0x06000000 0x0 0x100000>;
    clocks = <&clk_hdmi>, <&clk_hdmi_slow>, <&clk_hdmi_hdcp>, <&clk_hdmi_cec>;
    pinctrl-names = "ddc_active", "ddc_sleep", "cec_active", "cec_sleep";
    pinctrl-0 = <&hdmi_ddc_pin_a>;
    pinctrl-1 = <&hdmi_ddc_pin_b>;
    pinctrl-2 = <&hdmi_cec_pin_a>;
    pinctrl-3 = <&hdmi_cec_pin_b>;
    status = "okay";
};
```

5.2 板级配置

```
[hdmi]
hdmi_used = 1
hdmi_hdcp_enable = 1
hdmi_hdcp22_enable = 1
hdmi_cts_compatibility = 0
hdmi_cec_support = 1
hdmi_cec_super_standby = 1
hdmi_skip_bootedid = 1
ddc_scl = port:PH8<2><default><1><default>
ddc_sda = port:PH9<2><default><1><default>
cec_io = port:PH10<2><default><1><default>
ddc_en_io_ctrl = 1
ddc_io_ctrl = port:PH02<1><default><default><0>
```

重要参数说明：

hdmi_cec_super_standby：开启此功能可以使系统休眠时，cec 不休眠，接收端可以通过 cec 唤醒发送端；

hdmi_hdcp_enable：当需要开启 hdcp 时，必须置 1 该参数；

hdmi_hdcp22_enable：置 0 时，只开启 hdcp1.4 功能，置 1 时，开启 hdcp2.2 的功能；

ddc_en_io_ctrl：控制是否使用 ddc contrl io 功能（即下面的 ddc_io_ctrl 引脚配置），由硬件平台决定。

ddc control io 在 hdmi 插入时输出 1，在 hdmi 拔出时输出 0，可以使 ddc 的光电特性更加符合 CTS 测试的要求。

5.2.1 驱动配置

```
CONFIG_HDMI2_DISP2_SUNXI=y  
CONFIG_AW_PHY=y （全志自研phy） / DEFAULT_PHY(IP原厂的phy)  
CONFIG_HDMI2_CEC_SUNXI=y  
CONFIG_HDMI2_CEC_USER=y
```

6 驱动方案设计

6.1 初始化流程

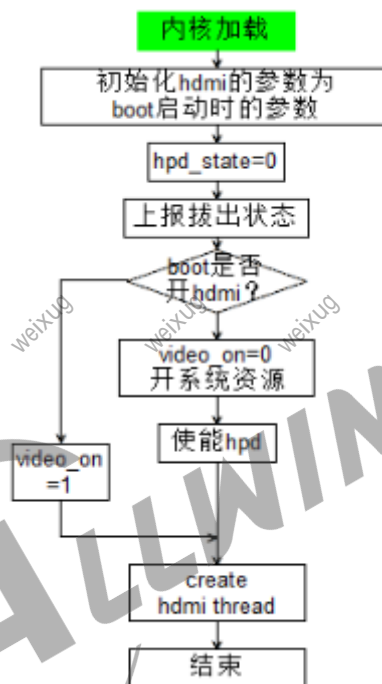


图 6-1: 驱动初始化

说明：

a. 内核初始化程序要求要同时考虑 boot 阶段有图像输出和 boot 图像没有输出这两种情况，如果是 boot 阶段已经配置有 hdmi 输出，那么在内核初始化阶段就没有必要再重复去配置 hdmi 的时钟等系统资源和使能 hpd 等操作，只需要做一些软件上的使能处理即可，比如增加 clk 和 clk parent 的 enable_count。

b. 内核初始化阶段，hdmi 需要通过 switch 机制（或者 extcon 机制）向用户空间上报一个 hpd 的拔出状态作为 hpd 最初始的状态。其实在内核刚起来的时候，cat /sys/class/switch/hdmi/s-tate 节点的值就是 0，初始化阶段报一个拔出状态一是为了使代码逻辑更加紧密，二是因为 hdmi resume 场景下也需要复用内核初始化的一些代码，hdmi resume 场景下也需要上报 hpd 拔出状态，因此就在内核初始化阶段也加进了这个流程。

注意：

hdmi 初始化完毕后，如果 hdmi 处于插入的状态，驱动是必须要读 EDID 然后上报插入状态的，但这个操作却不宜在系统内核加载阶段进行。因为读 edid 耗时是比较长的，放在系统内核加载阶段会影响内核起来的时间。因此目前的解决方法是，内核加载阶段创建出一条 hdmi_run_thread 线程，然后读 edid 的操作就在这条线程中执行，这样就不会影响系统内核起来的时间。

6.2 插拔流程

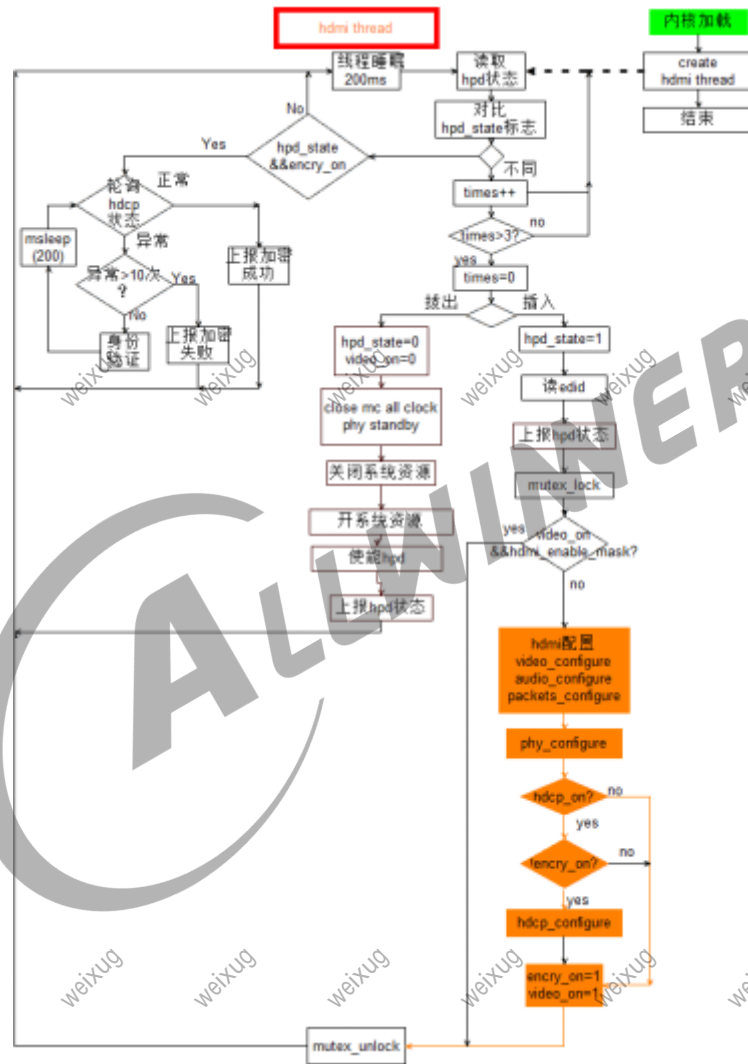


图 6-2: 插拔流程

说明：

a. 为什么 hdmi2.0 的插拔检测是采用线程轮询而不是采用中断响应的方式？如果采用中断响应的方式是否可行？

在 hdmi2.0 驱动开发的初始阶段，插拔的检测方式的确是采用了中断响应的方式，它的优点在于响应快速，但很快暴露出问题，在飞利浦电视上，开机的时候会产生两个插拔信号，前后

间隔时间大概 400ms，这时 hdmi 驱动就会连续的去处理两个插拔，造成开机闪屏的现象。还有就是使用中断，不好对 hdmi 的运行状态进行控制，比如如果硬件的防抖做得不好，某些抖动也会产生中断请求，而使用线程的话，虽然存在响应较慢的缺点，但在处理抖动时可以手动添加的防抖程序，另外使用线程轮询可以比较容易控制程序的运行状态。因此使用中断其实是可行的，但是给软件设计带来的难度远高于线程轮询。

b. 轮询方案

这里主要是围绕着 hpd_state 这个变量进行。将从硬件寄存器中读到的 hpd 的值与 hpd_state 变量进行比较，如果两种相同则说明没有 hpd 事件产生，如果不相同则说明有 hpd 事件产生，在处理完 hpd 事件后，再去改变 hpd_state 的值。这里还加入了 600ms 的延时防抖策略。

c. 插入事件的处理：先读 edid，再配置 video、audio、cec、hdcp 等模块；

d. 拔出事件处理先关掉 hdmi 输出，然后 reset hdmi 模块的时钟，然后再使能 hpd。reset hdmi 的所有时钟是因为 hdcp2.2 的 IP 有一个缺陷，如果不 reset 整个时钟的话，hdcp2.2 的 firmware 无法重新启动。

e. hdcp 的状态轮询

hdcp 状态轮询与 hpd 状态轮询都整合在一个线程里面，没有把这两个轮询做成两个线程是因为线程数的增加会大幅增加驱动软件设计的难度。hdcp 和 hpd 这两者如果并行不好处理。

6.3 分辨率切换

切换分辨率的流程主要是由显示驱动决定的，显示层的驱动已经定义好了接口，hdmi 的驱动只需要实现这些接口即可。

基本流程：hdmi_disable——>set_static_config——>hdmi_enable

详细流程：

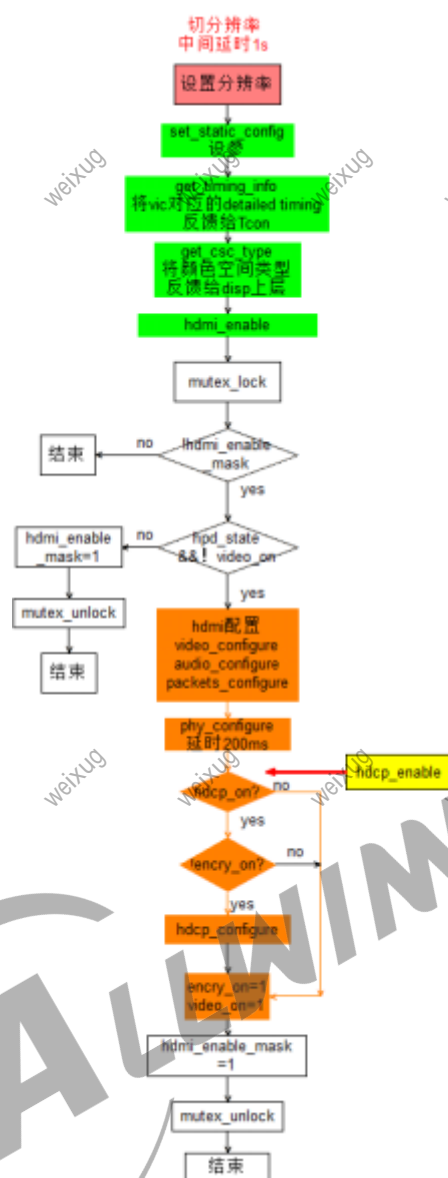


图 6-3: 切换分辨率

注意：

a. 特殊分辨率之 4k@60fps 的处理：根据 hdmi2.0 协议，4k@60fps 与其他分辨率存在某些不同点，如果使用 hdcp，那么它必须要打开 hdcp_keepout_window。由于 4k@60fps（无论是 yuv444/yuv422 8bits 还是 yuv420 10bits）的 tmds 已经超过 340MHz，因此必须要打开 scrambling。

b. 特殊分辨率之 3D 的处理：

HDMI 协议中定义了很多 3D 的格式，但是主流格式是 frame packing 格式，全志也是采用这样的格式，如下图为 2D 与 3D 的比较：

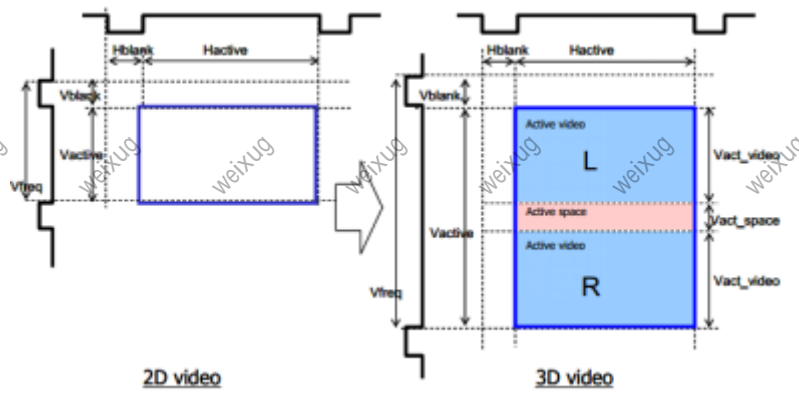


图 6-4: 2D 与 3D 比较

可以看出每一帧的 3D 图像是由两帧的 2D 图像组成，两张 2D 图像相隔的部分称为 Active Space，其大小就是其中的一帧 2D 图像的 Vblanking 的值。所以，在配置 3D 输出时，要注意 clock 要配成相同分辨率的 2D 图像的 2 倍，另外，再在 HDMI Vendor Specific InfoFrame 中配置一下 3D 的格式即可。3D 图像的生成由 DE 去生成，HDMI 只负责输出信息就可以。

c. 特殊视频格式静态 HDR 的处理：

静态的 HDR，不管是怎么样的图像，都固定的发送一组固定的 SRM 值。静态的 SRM 的值由算法部门提供。

d.SDR 和 HDR 平滑切换处理：

当这两者进行切换时，只要保证分辨率、格式（YUV/RGB）、Color Depth(8bits/10bits/12bits/16bits) 保持不变，就可以进行平滑切换，平滑切换时只需改变 DE 和 HDMI 的对 EOTF 和颜色空间的相关的设置即可。但需要注意的是，根据 CTS 的要求，HDR 平滑切到 SDR 时，需要先发数值为 0 的 DRM 2 秒钟，然后再关闭 DRM 的发送。

e. 特殊视频格式动态 HDR 的处理：

根据图像的变化，动态发送 SRM 值。动态的 SRM 值，由图像的特征决定。

f. 特殊视频格式 yuv420 的处理：

对于 yuv420 格式，主要要处理的问题是 pixel clock 要减半，因此也就说明 clock 和 tmds clock 要减半。

g. 特殊视频格式 yuv422 的处理：

根据 HDMI1.4 协议的相关描述可知，yuv422 的速率保持和 yuv444 一致

h. 特殊视频格式 10bit 的处理：

pixel_clk 和 tmds_clk 速率设为 8bits 时的 1.25 倍，但 yuv422 为 10bit 时，强制让其变为 8bits，理由如下：

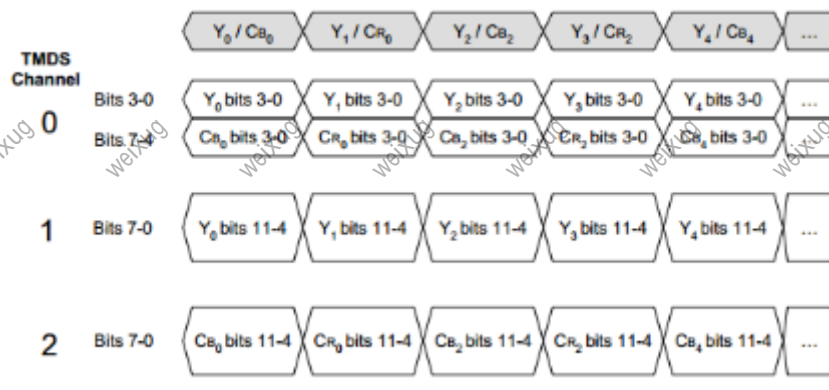


图 6-5: 422 打包格式

上图是 yuv422 的打包传输格式，亮度信号 Y 分量和色度分量 U、V 都被编码成了 12bits 进行传输。对于 yuv422 8bits 来说，会占据编码的低 8 位，空出来的高 4bits 由 0 来填充。同理对于 yuv422 10bits 来说，会占据编码的低 10bits，剩下的 2 位由 0 来填充。因此，我们可以知道，yuv422 不管是 8bits、10bits、12bits，它们的传输速率都是一样的。

i. 时钟频率的处理:

由于 HDMI 分辨率跨度大，种类多，从 480i@60fps_{4k@60fps}，像素时钟频率从 13.5MHz^{594MHz}。不仅要支持其 CEA-861 协议中规定的标准分辨率，另外还需要支持 VESA 协议里面规定的标准分辨率，甚至还需要支持起某些 LCD 屏厂商自己定义的分辨率。所以 HDMI 2.0 甚至整个显示模块的时钟配置都需要十分巧妙的处理。在现有项目中，我们采取的时钟配置策略如下：

首先判断在目前的父时钟频率下，能否分频得到我们想要的时钟频率 rate（使用 `clk_set_round_rate`）。如果能，就直接设置时钟频率（使用 `clk_set_rate`），如果不能，则有必要去设置父时钟的频率。我们将 rate 乘以一个分频因子 div 得到一个我们想要设置的父时钟的频率，然后去判断父时钟能否获得我们想要设置的父时钟频率。如果还不能，则我们就换一个分频因子，重复上一步的步骤，直到我们获得我们想要的父时钟频率为止。为了便于遍历 div，div 的取值将从 1 开始。

j.timing 的转换公式:

HDMI 和 TCON 对于 timing 的定义略有不同，所以需要做相应转换。

detailed timing 是用来描述一个视频特征的最基本的参数，下图就是视频的 detailed timing 的描述图。

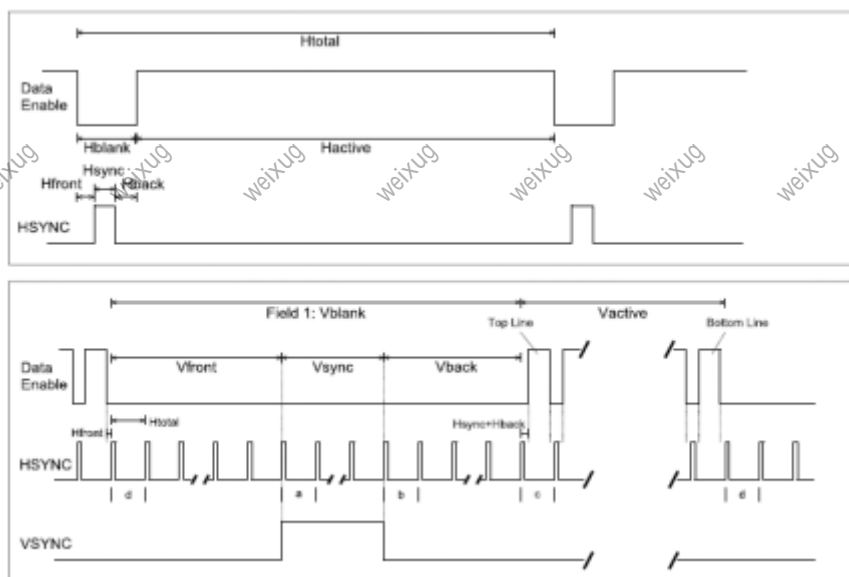


图 6-6: detail timing

因此，清楚不同组合之间参数的转换关系是至关重要的。我总结了一下，常用到的转换关系如下：

$$\text{hor_total_time} = \text{hactive} + \text{hblanking};$$

$$\text{hblanking} = \text{hor_front_porch} + \text{hsync_pulse_width} + \text{hor_back_porch};$$

$$\text{hsync_offset} = \text{hor_front_porch};$$

$$\text{ver_total_time} = \text{vactive} + \text{vblanking};$$

$$\text{vblanking} = \text{ver_front_porch} + \text{hsync_pulse_width} + \text{ver_back_porch};$$

$$\text{vsync_offset} = \text{ver_front_porch};$$

还有一个，3D 模式下（特征 frame packing 格式下的 3D）， $\text{vactive_space} = \text{vblanking}$

k.DVI 模式处理:

根据 DVI 的协议可知，DVI 模式只有视频模式而且只支持 RGB 格式，不发送音频。因此在 DVI 模式时，不配置 Audio，也不应该设置 YUV 格式

l.map code 问题:

mapcode 代表着 hdmi 从 tcon 中将 video 数据同步到 hdmi 中的方式，如下：

Input Format				idate[47:0]																																															
Color Space	Color Depth	Sample M Code		47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGB 4:4:4	8-bit	1	R[7:0]																	G[7:0]																B[7:0]															
	10-bit	3	R[9:0]																	G[9:0]																B[9:0]															
	12-bit	5	R[11:0]																	G[11:0]																B[11:0]															
	16-bit	7	R[15:0]																	G[15:0]																B[15:0]															
YCrCb 4:4:4	8-bit	9	Cb[7:0]																	Y[7:0]																Cr[7:0]															
	10-bit	11	Cb[9:0]																	Y[9:0]																Cr[9:0]															
	12-bit	13	Cb[11:0]																	Y[11:0]																Cr[11:0]															
	16-bit	15	Cb[15:0]																	Y[15:0]																Cr[15:0]															
YCrCb 4:2:2	8-bit	22	Cb[n] [7:0] Cr[n] [7:0]																	Y[n] [7:0]																															
	10-bit	20	Cb[n] [9:0] Cr[n] [9:0]																	Y[n] [9:0]																															
	12-bit	18	Cb[n] [11:0] Cr[n] [11:0]																	Y[n] [11:0]																															
	8-bit	9	Cb[L] [n] [7:0] Cr[L+1] [n] [7:0]																	Y[L] [n] [7:0]																Y[L+1] [n+1] [7:0]															
YCrCb 4:2:0	10-bit	11	Cb[L] [n] [9:0] Cr[L+1] [n] [9:0]																	Y[L] [n] [9:0]																Y[L+1] [n+1] [9:0]															
	12-bit	13	Cb[L] [n] [11:0] Cr[L+1] [n] [11:0]																	Y[L] [n] [11:0]																Y[L+1] [n+1] [11:0]															
	16-bit	15	Cb[L] [n] [15:0] Cr[L+1] [n] [15:0]																	Y[L] [n] [15:0]																Y[L+1] [n+1] [15:0]															
	8-bit	23	Y[7:0]																	Cb[7:0]																Cr[7:0]															
YCrCb 4:4:4	10-bit	24	Y[9:0]																	Cb[9:0]																Cr[9:0]															
	12-bit	25	Y[11:0]																	Cb[11:0]																Cr[11:0]															
	16-bit	26	Y[15:0]																	Cb[15:0]																Cr[15:0]															
	12-bit	27	Y[n] [11:0] Y[n+1] [1:0]																	Cb[n] [11:0] Cr[n] [11:0]																															
YCrCb 4:2:0	8-bit	28	Y[L] [n] [7:0] Y[L+1] [n] [7:0]																	Y[L] [n+1] [7:0] Y[L+1] [n+1] [7:0]																Cb[L] [n] [7:0] Cr[L+1] [n+1] [7:0]															
	10-bit	29	Y[L] [n] [9:0] Y[L+1] [n] [9:0]																	Y[L] [n+1] [9:0] Y[L+1] [n+1] [9:0]																Cb[L] [n] [9:0] Cr[L+1] [n+1] [9:0]															
	12-bit	30	Y[L] [n] [11:0] Y[L+1] [n] [11:0]																	Y[L] [n+1] [11:0] Y[L+1] [n+1] [11:0]																Cb[L] [n] [11:0] Cr[L+1] [n+1] [11:0]															
	16-bit	31	Y[L] [n] [15:0] Y[L+1] [n] [15:0]																	Y[L] [n+1] [15:0] Y[L+1] [n+1] [15:0]																Cb[L] [n] [15:0] Cr[L+1] [n+1] [15:0]															

如果 map code 设置不对，将导致输出的图像颜色发生错误。

同切分辨率流程类似，audio 流程是由音频驱动决定，音频驱动已经设计好了接口，hdmi 驱动的任务就是去实现这些接口。

```

graph LR
    A[设置audio] --> B[set_audio_para]
    B --> C[audio enable]
    C --> D[audio initialize  
audio_configure]
    D --> E[结束]

```

注意：

a. 根据 hdmi 的协议我们知道，hdmi audio 传送的都是非压缩类音频，hdmi 的 Rx 端只需根据音频的头信息，CA 值再结合 ICE60958 或者 ICE61937 协议，再根据传送过来的 CTS 值和 N 值就可以正确地解析 Tx 端传过来的音频信息。因此，在配置 hdmi 输出 audio 时，关键是要配置正确 N 值、CTS、CA 值，还有就是与 audio 模块相连的 I2S 或 SPDIF 接口要正确配置，比如 I2S，打开的通道一定要正确，I2S 工作的模式一定要配置正确。

b. 当 hdmi audio 的类型发生改变时，比如位宽由 16 位变成 24 位等，在重新配置 hdmi audio 必须要 reset fifo。否则会出现音频卡顿甚至无声的情况。

c.HDMI Audio HBR 格式传输原理

对于 HBR 模式，HBR 模式的数据传输速率为 768KHz，但是每个 I2S(audio 模块与 hdmi 模块的接口) 最高的速率也只有 192KHz，因此 HBR 的传输方式采用的是 4 个 I2S（即从 HDMI Tx 输出到 HDMI Rx 端时是 8 个通道），每个 I2S 192KHz 的方式传输。然后 HDMI Rx 端再将这 4 路数据进行重新组合，最后得到 HBR 数据。这是当前 hdmi 驱动中 HBR 的传输策略，也是 HDMI2.0 协议对 HBR 的 audio 格式传输的规定。

d.CA 值问题

所谓的 CA 值，指的是 Channel Allocation，表示音频播放时 speaker 的位置，HDMI 协议中对 CA 值的定义遵循 CEA-861 协议，可自行阅读协议了解一下。audio 必须根据上图对 CA 值的定义，在配置 hdmi 输出 audio 时，向 hdmi 层正确的传递 ca 值。

e. 极易出错的 audio dma 数据生产消费问题

特别需要注意的是由于 HBR 模式传输的速率很高，再往 audio 灌数据时，必须是一个线程取数据，另一个线程在灌数据，如果这两个动作都在同一个线程中执行，有可能出现消费数据大于生产数据的现象，进而出现音频卡顿甚至无声的现象。

f. 极易出错的按键音问题

无论是哪个平台，当在调试 HDMI Audio 时，如果不注意，几乎都会出现按键音问题，问题描述：由于 audio 的软件流程的原因，每次按按键音的时候，audio 层都会调用到 hdmi 的 hdmi_audio_enable() 函数，所以每次按键音出来前都要重新配置以便 hdmi audio，但是 audio 每次配置完成后 HDMI Rx 端往往需要一定的时间去“重新适应”（0~2 秒，不同电视“适应能力”不同），才能去解析 Rx 端传过来的 hdmi audio 数据。但是按键音十分短促，因此当 Rx 端完成“重新适应”时，按键音数据早已传完，因此就听不到按键音了。因此，要特别注意 audio 在往 hdmi 输出按键音时，不要每次都去配置 hdmi 的 audio 功能。

附：每次配置完 hdmi audio 功能，Rx 端都需要一定的时间去“重新适应”的原因：配置 hdmi 的 audio 功能的过程：send audio mute ——> 配置相关寄存器 ——> clear audio mute。Rx 端在收到 Tx 传过来的 audio mute 的信号后，就不在去解析 hdmi 数据流中的 audio 部分，有某些电视在收到 audio mute 信号后就直接关闭 hdmi audio 解析功能了，因此当清除 clear audio mute 后，Rx 又重新使能 hdmi 的解析功能，但是这个重新使能是需要时间的。

g. 爆音问题

HDMI 的爆音问题需要注意两点：一是 HDMI AUDIO 在配置过程中一定要发送 av_mute，等到配置完，然后才能 clear av_mute，这是为了防止 audio 格式发生突然变化产生爆音。二是注意当 audio 模块不向 HDMI 模块提供数据时，它需要保持 clock 不能停止，因为这样可以时 fifo 处于运动的状态，使 hdmi 不断的往外输出 0 的数据。如果不这样的话，当 HDMI 切换分辨率时，HDMI 的速率发生变化，有可能使 audio 输出某些不确定的数据，导致爆音产生。

6.5 休眠唤醒流程

同切分辨率流程一样，休眠唤醒流程也由显示驱动决定，显示驱动已经设计好了接口，hdmi 驱动的任务就是去实现这些接口。

基本流程：hdmi_disable——>hdmi_suspend——>hdmi_resume——>hdmi_enable

详细流程：hdmi_disable 和 hdmi_enable 流程请参考上面。

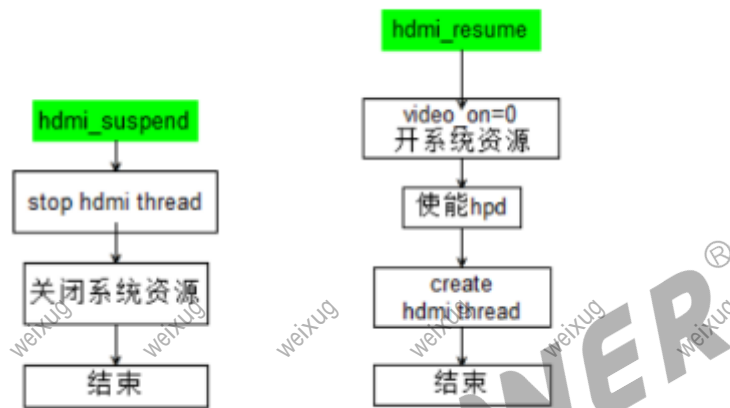


图 6-9: 休眠唤醒

cec super standby 机制

盒子的规格中，有一项对 cec 的要求，就是用电视去唤醒休眠的盒子。盒子进入休眠状态后，ccmu、电源等都处于关闭状态，连 CPU 都停止了运行，因此怎么样让盒子响应电视发过来的信息就是解决这个问题的关键。为了解决这个问题，我们设计了 cec super standby 机制（可以在 board.dts 中配置其是否要开启）：当系统处于休眠状态时，cpu 处于停止运行的状态，但是 CPUS 却是在运行的，因此我们只要在 CPUS 的运行程序里加入 cec 信号的轮询程序，并且在系统休眠时不关闭 HDMI 的电源，不关闭 CEC 的时钟，那么我们就可以接收到电视发过来的 cec 信息，如果无 cpus 的平台，将暂时无法实现此功能，如 H616。

6.6 HDCP 功能实现

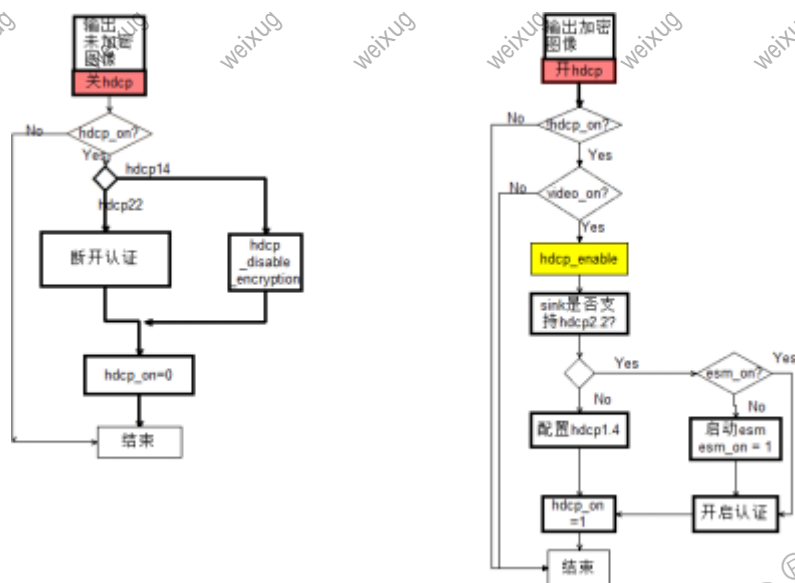


图 6-10: HDCP 流程

说明：

a. 以上设计是同时兼容 hdcp1.4 和 hdcp2.2 的设计方案，在 board.dts 配置文件中定义了一个 hdmi_hdcp22_enable 的开关，用户可以通过这个开关选择是否需要 hdcp1.4 和 hdcp2.2 同时兼容的方案。如果这个开关没有打开，则只配置 hdcp1.4。

b. hdcp2.2 在硬件上是一个独立的 IP，名为 ESM，需要依靠 IP 厂商提供的 firmware 运行（IP 厂商不提供源代码），这个 firmware 有一个毛病，关闭之后如果要重新启动，必须要 reset 整个 hdmi 的 clock，插拔之所以要 reset 整个 hdmi 的 clock 也是基于这种考虑。

ESM 内部是一个单片机，但是没有内存，要想让它运行，只能靠在 soc 平台中申请一段内存让其独占。但是特别应该注意的是，这段内存申请，如果是使用 kmalloc 或者 __get_free_page 来申请时，当把 firmware 复制到申请的内存时，必须要刷 cache，因为此时数据还停留在 cache 里面并没有进入 dram。为了避开刷 cache 的问题，有一个比较好的方法就是使用 DMA 内存的方式，这样就可以避开 cache。

c. hdcp1.4 和 hdcp2.2 的烧写 key 的过程请参考《HDCP 2.2 烧 key 方案.pdf》、《HDCPv1.4 烧 key 读 key 流程.pdf》。

6.7 CEC 流程设计

必须符合实际需求与 CTS 要求，具体流程详见下图：

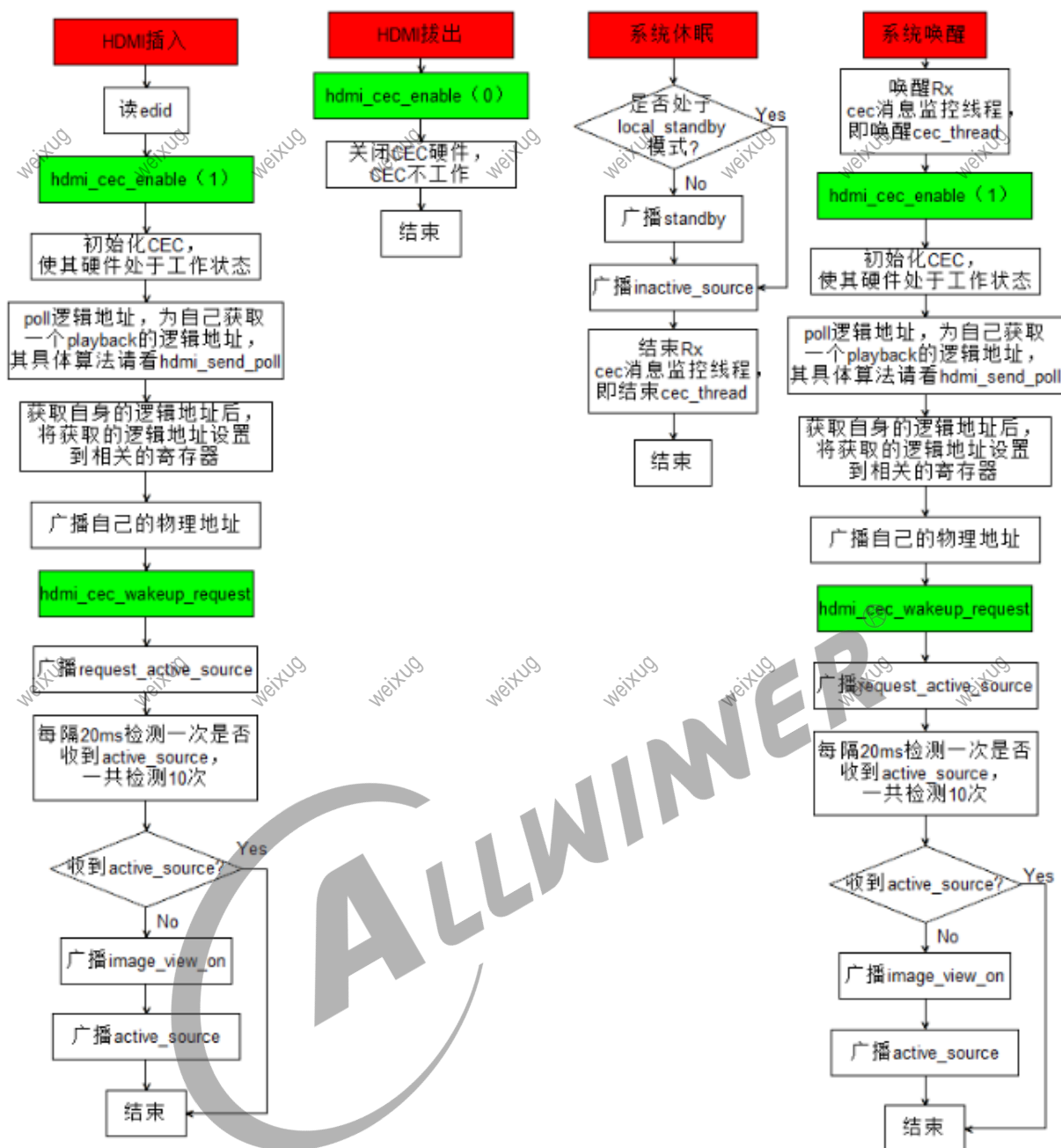


图 6-11: CEC 流程 1

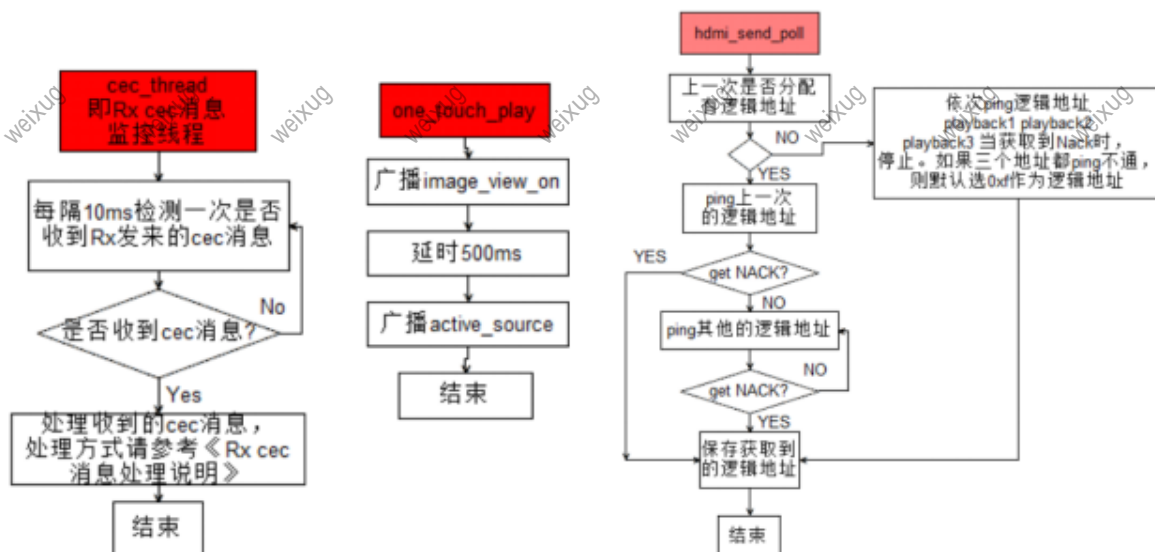


图 6-12: CEC 流程 2



图 6-13: RX 处理消息流程

6.7.1 CEC 逻辑地址的设置原则（HDMI CTS 标准）

(1) 必须保证逻辑地址设置到硬件，如果不将逻辑地址设置到硬件，板子是收不到 CEC 消息的；

(2) 设置 CEC 逻辑地址之前，需要 poll 逻辑地址，确定这个地址没有被其他设备占用（板子收到 NACK 回应），才能设置这个逻辑地址；

(3) poll 逻辑地址与插拔：

每次 HDMI 拔出再插入后必须要重新 poll 一次逻辑地址，再设置 poll 到的逻辑地址。

这样做是因为板子拔出后，有可能再插入另一个设备，有可能在这个设备中板子原先所设置的逻辑地址已经被占用了，所以板子需要重新 poll 逻辑地址。

注意：拔出再插入后，所 poll 的第一个逻辑地址必须是上一次板子所占用的逻辑地址，如果这个逻辑地址被占用了，才 poll 第二个逻辑地址，这是 CTS 要求的标准。

6.7.2 上/下/左/右/返回键功能

规律：每个按键都分为按压（press）和释放（release）分别对应按压和弹起两个动作。

注意：假如一直按压，TV 只会发送 press 信息，UI 图标的选中也应随之跳转；

release 消息的应用可随用户根据自己的方案自己决定，可用也可不用。假如使用，release 可作为 UI 图标停止跳转的信号，当然也可以有其他用途，例如小米电视上，如果一直 press，那么 UI 图标最多能跳转到下一行的第一个图标，只有 release 后再 press，才能跳转到第二行的其他图标。

左键：opcode: 0x44 operand: 0x03 opcode: 0x45

右键：opcode: 0x44 operand: 0x04 opcode: 0x45

上键：opcode: 0x44 operand: 0x01 opcode: 0x45

下键：opcode: 0x44 operand: 0x02 opcode: 0x45

返回键：opcode: 0x44 operand: 0x0D opcode: 0x45

6.7.3 休眠状态下 cec 交互情况

主要实现电视唤醒盒子的功能。

说明：

如果需要这个功能，需要在 board.dts 中，hdmi 节点下打开 hdmi_cec_super_standby。

板子休眠状态下，cpu 已经休眠，只有 cpus 在跑，因此需要在 cpus 的代码中添加处理 cec 消息的代码。

注意：cpus 的内存寻址的设计采用的是大端模式（big-endian，低位地址放在高位内存中），与大 cpu 采用的小端模式（little-endian，地位地址放在高位内存中）是不一样的。所以，在 cpu 视角下看到的寄存器地址，在进入 cpus 之后需要作转换。

转换规律：

例如，大 cpu 视角下，CEC_TX_CNT 的地址为 0x06007d07 首先，找到大 cpu 地址所在的字， $0x06007d07 \% 4 = 3$ ，所以其所在的字为 $0x06007d07 - 3 = 0x06007d04$ 所以，大 cpu 中，这个字所在的位置的排布情况为：0x06007d04 0x06007d05 0x06007d06 0x06007d07，因为 cpus 是大端，所以大 0x06007d07 寄存器在 cpus 的地址为 0x06007d04

6.7.3.1 休眠前，hdmi_suspend 应该做的动作

因为休眠后，cec 硬件还需要保持运行，因此不能关闭 cec 的时钟和释放 CEC 的引脚，所以，只关闭 hdmi/ddc/hdcp 的时钟，释放 ddc 引脚，CEC 的 suspend 是软 suspend，只是进行软件的逻辑处理，如关闭 cec 轮询线程，不进行硬件操作。

cec 消息处理方案：

(1) 首先，进入 cpus 的运行代码后，cec 方面要做的第一件事是初始化：清除相关的中断状态寄存器，清除寄存器内已经缓冲的 cec 消息。这样做是因为，cec 进入休眠之前，发送了 In-active Source，所以电视或者 cec net 上其他输入源有可能会发出 active source 的消息，而 active source 恰好是板子的唤醒源之一，因此这会导致一进入休眠，立马唤醒，使板子无法进入休眠；

(2) 获取板子自身的逻辑地址：cec 的交互必须要有逻辑地址，板子在进入休眠之前，逻辑地址已经设置进硬件，所以在 cpus 中，不需要再 poll 逻辑地址，只要读取相关的寄存器就能获取逻辑地址；

(3) 获取板子的物理地址

在板子进入休眠前已经关闭了 ddc，因此想要通过 ddc 读 edid 来获取物理地址是不可能的了，即便可以，维持 ddc 功能会耗电，不适合在系统 standby 的状态下运行。因此，cec 的物理地址只能通过 Linux 内核在进入休眠之前传过来然后在休眠后，再获取即可，这个需要 standby 的同事实现相关的接口。

(4) 唤醒源

0x04: 0x0d: 0x82: , source logical address 是 tv 才唤醒 //0x80: , 其参数与板子的物理地址相等时，才唤醒 0x86: , 其参数与板子的物理地址相等时，才唤醒

(5) 其他信息的处理 0x8f: 广播回应 0x90:

6.7.4 标准 CEC 驱动方案

说明：

旧版的 CEC 驱动方案的缺陷：旧版的 CEC 驱动，很多 CEC 的消息（主要是一些基本功能的消息，比如 poll，报告物理地址等）放在了驱动来处理。这其实是不合理的，CEC 的消息本是属于业务逻辑，这些不应放在驱动里，造成这样的原因主要是因为当初没有人手去开发逻辑层。

6.7.4.1 方案

在 CEC 驱动方案里面，驱动以 ioctl 的方式向用户空间提供以下接口：

(1) 设置和获取 initiator 端物理地址：CEC_S_PHYS_ADDR、CEC_G_PHYS_ADDR

(2) 设置和获取 initiator 端物理地址：CEC_S_LOG_ADDR、CEC_G_LOG_ADDR

(3) 向 follower 发送 CEC 消息：CEC_TRANSMIT

其中用注意的是，发送 CEC 消息可以有两种方法：阻塞和非阻塞，采用哪种方式主要是在打开 cec 节点时设置

(4) 接收 follower 发来的 CEC 消息：CEC_RECEIVE

6.8 log 管理方案

为了方便调试和监控 hdmi 的运行状态，在 hdmi 的驱动内部设置了打印等级，如下：

操作方法：echo [log_level] > /sys/class/hdmi/hdmi/attr/debug

log_level 的意义：

1—只打印 video 的 log

2—只打印 edid 的 log;

3—只打印 audio 的 log;

4—只打印 video + edid + audio 的 log;

5—只打印 cec 的 log;

6—只打印 hdcp 的 log;

7—以上的都打印;

8—以上的都打，log trace 也打印;

7 结束语

对于类似于 HDMI 这种协议和流程较为复杂的模块，在开发之初，就应该要做好充分的预研，开发者在开发之前，就需要对协议进行充分地学习。在开发中，驱动的开发人员和 IP 的验证人员必须要充分配合。功能开发完毕后，需要对软件进行一定量的测试来保证稳定性。






著作权声明

版权所有 © 2022 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。