AW HRC
开发指南

版本号: 1.0
发布日期: 2025.2.20

版本历史

版本号	日期	制/修订人	内容描述
1.0	2025.2.20	AWA2130	创建该文档

目　录

插　图

表　格

1 概述

1.1 编写目的

本文档主要介绍 HRC 模块开发和调试方式。

1.2 适用范围

表 1-1: 适用产品列表

芯片	安卓版本	内核版本
H135/H136	-	Linux6.6

1.3 术语，缩略语及概念

表 1-2: 软件术语列表

术语	解释说明
HRC	HDMI Rx Capture，HDMI RX 的回写模块。
V4L2	Video For Linux 2，内核原生 Video Capture 框架。

2 使用方式

2.1 驱动代码

```
# <sdk>/bsp/driver/tvin/hrc/
.
├── hardware
│   ├── Makefile
│   ├── sunxi_hrc_hardware.c
│   ├── sunxi_hrc_hardware.h
│   ├── sunxi_hrc_platform.c
│   ├── sunxi_hrc_platform.h
│   ├── sunxi_hrc_reg.h
│   └── sunxi_hrc_table.h
├── Makefile
├── sunxi_hrc.c
├── sunxi_hrc_define.h
├── sunxi_hrc.h
└── sunxi_hrc_log.h
```

表 2-1: 文件功能解释表

文件名	功能
sunxi_hrc.c	hrc 主入口，对接 V4L2 框架和 HDMIRX 驱动。
sunxi_hrc.h	hrc 头文件。
sunxi_hrc_define.h	hrc 全局定义头文件，统一规定 HRC 模块中使用的参数。
sunxi_hrc_log.h	hrc 打印信息定义头文件。
sunxi_hrc_hardware.c	hrc 硬件抽象层，提供硬件统一操作接口。
sunxi_hrc_hardware.h	hrc 硬件抽象层接口头文件，对接 sunxi_hrc.c。
sunxi_hrc_platform.c	hrc 硬件平台描述层，提供不同 HRC 版本的硬件描述。
sunxi_hrc_platform.h	hrc 硬件平台描述定义头文件，对接 sunxi_hrc_hardware.c。
sunxi_hrc_reg.h	hrc 寄存器定义头文件。
sunxi_hrc_table.h	hrc 硬件参数表头文件。

2.2 配置方法

2.2.1 Menuconfig 配置

1. 在 sdk 根目录，执行以下命令进入 menuconfig 配置页面：

```
./build.sh menuconfig
```

2. 进入 menuconfig 配置页面后，按以下步骤使能驱动配置：

```
Allwinner BSP  --->
  Device Drivers  --->
    TVIN Drivers  --->
      <*> HRC Support for Allwinner SoCs
```

⚠ 注意

HRC 虽然可独立于 HDMIRX 做单独测试，但是这个模块是用来给 HDMIRX 做回写的。
正常使用时需要开启 HDMIRX 的驱动，配置方式请参考 AW_HDMI_IN 开发指南，本文档不再赘述。

2.2.2 DTS 配置

2.2.2.1 dtsi

dtsi 与平台相关，SDK 包已默认配置好该文件。如果没有相关配置，则默认当前平台不支持，客户从零适配难度大，此处不做适配讲解。

2.2.2.2 board.dts

```
# <sdk>/device/config/chips/h135/configs/p1/board.dts

&hrc {
    output_to_disp2;
    status = "okay";
};
```

表 2-2: dts 参数解释表

参数	解释
output_to_disp2	用于 disp2 显示框架的特殊处理标志。

3 调试节点

3.1 输出所有打印信息

```
echo 8 > /sys/class/hrc/hrc/attr/loglevel
```

0: 打印 info/warn/err 信息；

8: 打印 debug/info/warn/err 信息；

3.2 打印所有寄存器的值

```
cat /sys/class/hrc/hrc/attr/reg_dump
```

3.3 读取指定寄存器的值

```
echo <reg_addr>,<size> > /sys/class/hrc/hrc/attr/reg_read
cat /sys/class/hrc/hrc/attr/reg_read
```

3.4 写入值到指定寄存器

```
echo <reg_addr>,<val> > /sys/class/hrc/hrc/attr/reg_write
```

3.5 打印驱动调试信息

```
cat /sys/class/hrc/hrc/attr/dump
```

3.6 将送显 buffer 保存为文件

这个调试节点被设置时，会触发一个事件给应用层，让应用层在下一帧送显时不送给显示模块，而是保存为 raw 数据文件，用于确认后端显示模块是否存在异常。

```
echo 1 > /sys/class/hrc/hrc/attr/dump_buf
```

3.7 colorbar 调试功能

colorbar 功能是指通过 TCON 出调试图像给 HRC 做回写测试，**单纯写入下面节点只是开启该调试功能，还需配合应用程序才能看到效果！**

这个调试可判断 HRC 是否支持相应格式，可以判断是否为前端 HDMIRX 发生了异常，而导致的回写异常。

```
# 如果没有HDMIRX模块参与，需先设置input参数，否则可以不设置input参数。
# 如果设置了input参数，则会用HRC驱动时序表中的时序配置到TCON，从而显示出colorbar。
# 如果未设置input参数，则会用HDMIRX当前输入的时序配置到TCON，从而显示出colorbar。
echo input=0,0,0,0,1920,1080,0,0,1,0 > /sys/class/hrc/hrc/attr/params

# 开启colorbar（输入不同的值代表不同的colorbar类型）
echo 1 > /sys/class/hrc/hrc/attr/colorbar

# 关闭colorbar
echo 0 > /sys/class/hrc/hrc/attr/colorbar
# 如果设置了input参数，还需要清空参数。
echo 0 > /sys/class/hrc/hrc/attr/params
```

表 3-1: input 参数配置表

类型	值范围
data_src	0：HDMIRX 输入，1：DDR 输入。
field_mode	0：帧模式，1：场模式。
field_inverse	0：不翻转，1：翻转。
field_order	0：低场优先，1：顶场优先。
width	16-4096
height	16-4096
format	0：RGB，1：YUV444，2：YUV422，3：YUV420。
depth	0：8bit
cs	0：BT601，1：BT709，2：BT2020。
quantization	0：FULL，1：LIMIT

表 3-2: 目前支持的时序示例表

Timing	input 参数
720x480P60	input=0,0,0,0,720,480,0,0,1,0
720x576P50	input=0,0,0,0,720,576,0,0,1,0
1280x720P60	input=0,0,0,0,1280,720,0,0,1,0
1920x1080I60	input=0,1,0,0,1920,1080,0,0,1,0
1920x1080P60	input=0,0,0,0,1920,1080,0,0,1,0

Timing	input 参数
3840x2160P30	input=0,0,0,0,3840,2160,0,0,1,0
4096x2160P30	input=0,0,0,0,4096,2160,0,0,1,0

表 3-3: colorbar 显示类型表

值	类型
1	color check
2	gray scale
3	black by white
4	gridding

4 软件框架

4.1 软件框图

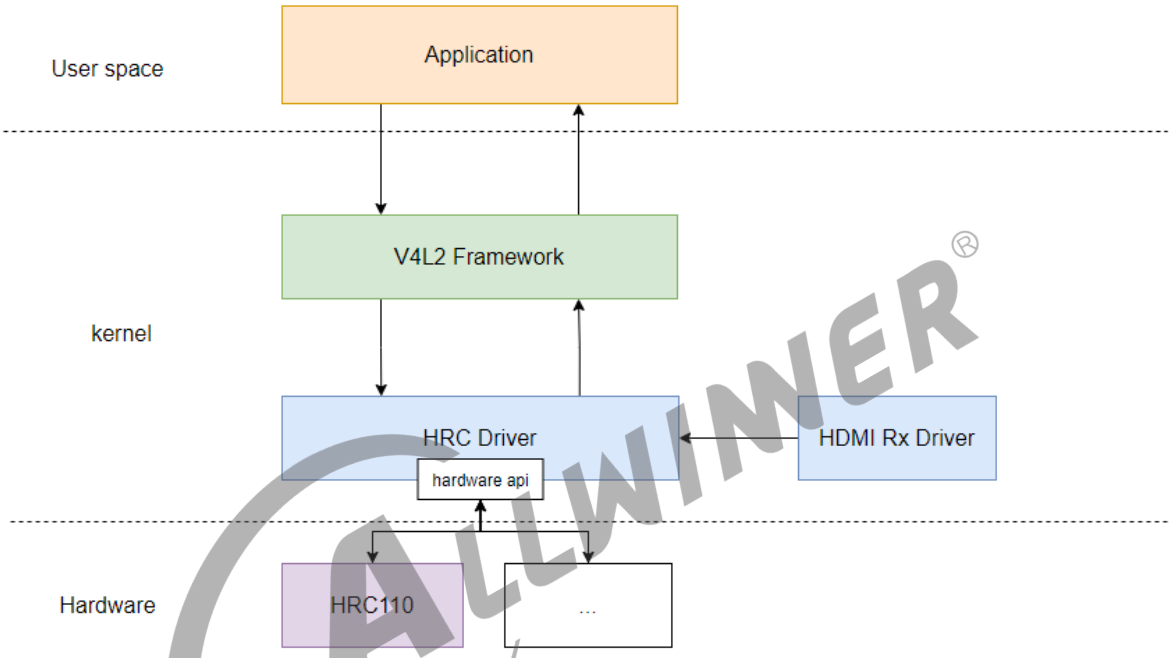

图 4-1: HRC 软件框图

- 用户层

用户的应用程序对接的是标准的 V4L2 框架，采用 V4L2 框架的原因：成熟且稳定，有大量应用程序，可减轻客户使用难度与开发成本。

- 内核层

HRC 驱动对接 V4L2 框架，实现相关调用接口，完成回写 Buffer 传递到用户层的工作。

HRC 与 HDMI RX 驱动保持参数配置的同步，确保输入参数正确获取。

- 硬件层

HRC 驱动通过对硬件抽象成多个 API 接口，为以后不同版本的 HRC 硬件兼容做准备。

4.2 功能划分

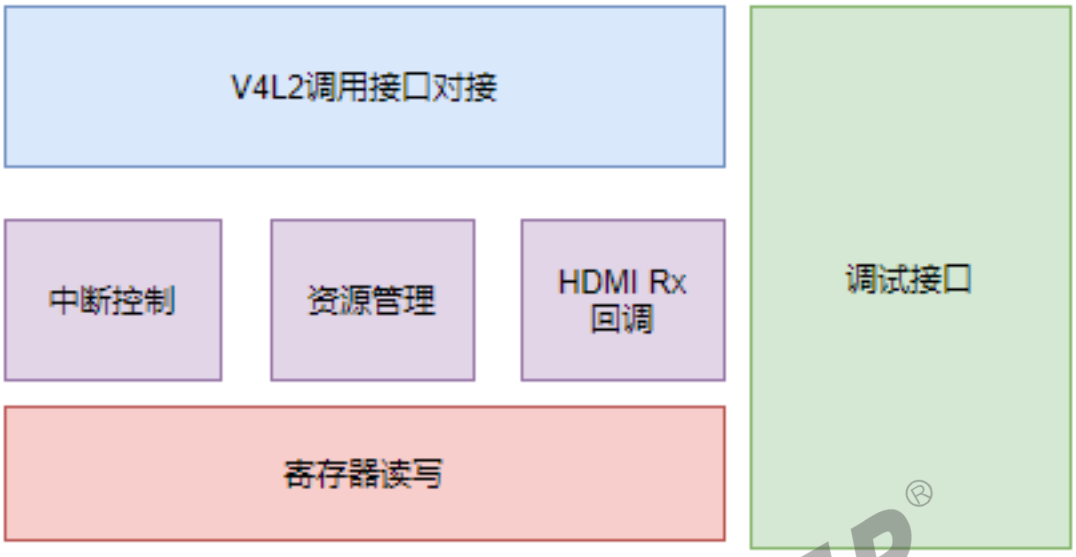

图 4-2: HRC 驱动主要功能划分图

- V4L2 对接：向 V4L2 框架注册设备，并且对接常用接口，实现与应用层的信息交互。
- 中断控制：配置好输入输出参数后，对不同中断进行响应，持续回写到所需的帧数据。
- 资源管理：控制时钟、电源等资源，确保 HRC 模块能正常工作。
- HDMI Rx 回调：获取 HDMI RX 驱动的工作状态，明确 HRC 的输入参数。
- 调试接口：用户层获取驱动运行信息，确认 HRC 运行状态。
- 寄存器读写：所有操作最终都需基于模块寄存器配置。

4.3 HDMIRX 驱动更新参数

HDMIRX 驱动通过 notifier 机制传递参数。

1. 当 HDMIRX 驱动有硬件状态更新时，则会调用 notifier 回调，将相应的参数按照定义好的规则传递给 HRC 驱动；
2. HRC 驱动接收到新参数后，触发 V4L2 的 subdev event 事件，将相应的参数更新通报给应用层，最终应用层调用相应的接口来读取。

HRC 与 HDMIRX 传递参数定义：

sunxi_hrc_define.h

```
/* ---------- HDMIRX <-> HRC ---------- */
#define HDMI_MAX_INPUT_PORT 4

enum hrc_source_id {
    hrc_source_Dummy,
    hrc_source_VideoDec, //DTV/USB/OTT
    hrc_source_Image,
    hrc_source_HDMI_1,
    hrc_source_HDMI_2,
    hrc_source_HDMI_3,
    hrc_source_HDMI_4,
    hrc_source_CVBS_1,
    hrc_source_CVBS_2,
    hrc_source_CVBS_3,
    hrc_source_ATV,
    hrc_source_Max,
};

struct hrc_from_hdmirx_signal {
    enum hrc_source_id   source_id;
    enum hrc_fmt        format;
    enum hrc_depth      depth;
    enum hrc_color_space  csc;
    enum hrc_quantization quantization;
    struct v4l2_bt_timings timings;
};

struct hrc_from_hdmirx_5V {
    u8 PortID;
    bool is5V;
};

enum notify_msg_type {
    SIGNAL_CHANGE = 0x0,
    HPD_CHANGE,
    AUDIO_FSRATE_CHANGE,
};
```

4.4 硬件层抽象

```
enum hrc_apply_dirty {
    HRC_APPLY_NO_DIRTY   = BIT(0),
    HRC_APPLY_PARAM_DIRTY = BIT(1),
    HRC_APPLY_ADDR_DIRTY  = BIT(2),
    HRC_APPLY_ALL_DIRTY   = BIT(3),
};

struct hrc_apply_data {
    struct hrc_input_param input;
    struct hrc_output_param output;
    enum hrc_apply_dirty dirty;
};

struct hrc_size_range {
    u32 min_width;
    u32 min_height;
```

```
    u32 max_width;
    u32 max_height;
    u32 align_width;
    u32 align_height;
};

struct hrc_hw_create_info {
    struct platform_device *pdev;
    u32 version;
};

struct hrc_hw_handle {
    struct hrc_hw_create_info info;
    struct hrc_private_data *private;
};

enum hrc_irq {
    HRC_IRQ_FRAME_VSYNC = BIT(0),
    HRC_IRQ_CFG_FINISH  = BIT(1),
    HRC_IRQ_WB_FINISH   = BIT(2),
    HRC_IRQ_OVERFLOW    = BIT(3),
    HRC_IRQ_TIMEOUT     = BIT(4),
    HRC_IRQ_UNUSUAL     = BIT(5),
    HRC_IRQ_ALL         = BIT(6),
};

int sunxi_hrc_hardware_enable(struct hrc_hw_handle *hrc_hdl);
int sunxi_hrc_hardware_disable(struct hrc_hw_handle *hrc_hdl);
int sunxi_hrc_hardware_reset(struct hrc_hw_handle *hrc_hdl);
int sunxi_hrc_hardware_get_size_range(struct hrc_hw_handle *hrc_hdl, struct hrc_size_range *size);
int sunxi_hrc_hardware_get_format(struct hrc_hw_handle *hrc_hdl,
                    enum hrc_fmt input_format,
                    enum hrc_fmt *output_format);
int sunxi_hrc_hardware_get_color_space(struct hrc_hw_handle *hrc_hdl,
                      enum hrc_color_space input_cs,
                      enum hrc_color_space *output_cs);
int sunxi_hrc_hardware_get_quantization(struct hrc_hw_handle *hrc_hdl,
                       enum hrc_quantization input_quan,
                       enum hrc_quantization *output_quan);
int sunxi_hrc_hardware_check(struct hrc_hw_handle *hrc_hdl, struct hrc_apply_data *data);
int sunxi_hrc_hardware_apply(struct hrc_hw_handle *hrc_hdl, struct hrc_apply_data *data);
int sunxi_hrc_hardware_commit(struct hrc_hw_handle *hrc_hdl);
int sunxi_hrc_hardware_get_irq_state(struct hrc_hw_handle *hrc_hdl, u32 *state);
int sunxi_hrc_hardware_clr_irq_state(struct hrc_hw_handle *hrc_hdl, u32 state);
int sunxi_hrc_hardware_dump(struct hrc_hw_handle *hrc_hdl, char *buf, int n);
struct hrc_hw_handle *sunxi_hrc_handle_create(struct hrc_hw_create_info *info);
void sunxi_hrc_handle_destory(struct hrc_hw_handle *hrc_hdl);
```

表 4-1: 硬件抽象接口解释表

函数名	作用
sunxi_hrc_handle_create	创建硬件句柄。（参数：硬件版本）
sunxi_hrc_handle_destory	销毁硬件句柄。
sunxi_hrc_hardware_enable	使能硬件。
sunxi_hrc_hardware_disable	关闭硬件。
sunxi_hrc_hardware_reset;	重置硬件。
sunxi_hrc_hardware_get_size_range	获取硬件支持分辨率范围。

函数名	作用
sunxi_hrc_hardware_get_format	根据输入格式，获取硬件支持的输出格式。
sunxi_hrc_hardware_get_color_space	根据输入颜色空间，获取硬件支持的输出颜色空间。
sunxi_hrc_hardware_get_quantization	根据输入颜色范围，获取硬件支持的输出颜色范围。
sunxi_hrc_hardware_check	校验输入输出参数是否符合规范。
sunxi_hrc_hardware_apply	提交一次硬件参数更新。
sunxi_hrc_hardware_commit	确认硬件参数提交完成。
sunxi_hrc_hardware_get_irq_state	获取硬件中断状态。
sunxi_hrc_hardware_clr_irq_state	清楚硬件中断状态。
sunxi_hrc_hardware_dump	打印所有硬件状态。

4.5 初始化流程

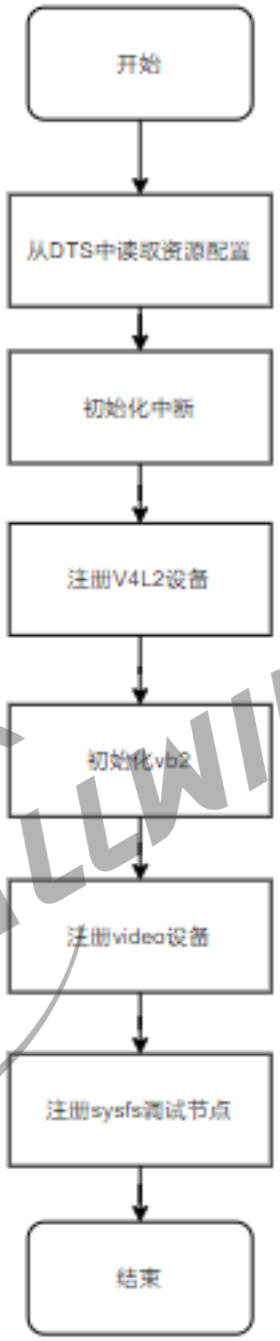

图 4-3: HRC 初始化流程

1、从 DTS 中读取资源配置。

其中资源配置包括：中断、时钟、电源等配置，这些资源配置决定模块是否可以正常工作；

2、初始化中断；

初始化中断并注册中断服务函数。

3、注册 V4L2 设备；

4、初始化 vb2;

5、注册 video 设备；

对接 V4L2 框架，框架生成 /dev/video 接口，用于给应用层 ioctl 使用。

6、注册 sysfs 调试节点；

注册调试节点，为驱动调试提供信息。

4.6 数据回写流程

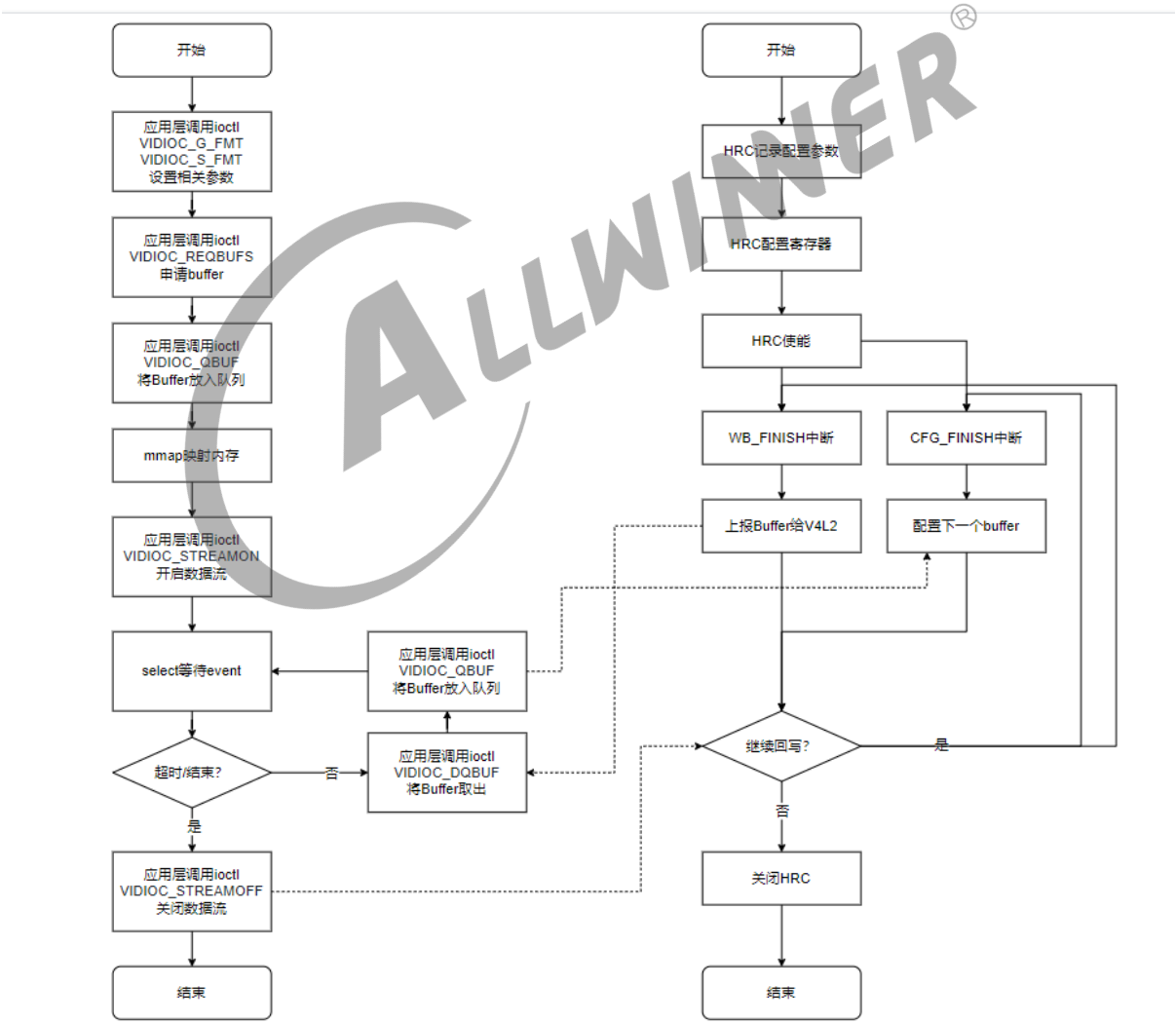

图 4-4: HRC 回写流程

4.6.1 应用层

- 配置格式参数；
- 申请 buffer 内存（根据格式参数大小申请）；
- 映射内存；
- 开启数据流；
- 等待驱动层回写完成，上报 event 事件，如果超时则结束数据流。
- buffer 就绪后，从 buffer 队列中 dequeue 出 buffer，再把用完的 buffer queue 回去。
- 继续循环等待 buffer 就绪，一直到用户结束调用。

4.6.2 驱动层

- 将应用层的参数配置到寄存器；
- 配置好 buffer 地址；
- 当应用层调用 streamon 时，开启 HRC 回写；
- 当 cfg_done 中断触发时，则写入下一个 buffer 的地址（多 buffer 轮转）；
- 当 wb_done 中断触发时，则将当前 buffer 上报，通知应用层 buffer 已就绪，可以进行 dequeue buffer；
- 一直循环到应用层调用 streamoff，关闭 HRC。

4.7 中断处理流程

4.7.1 frame_vsync

触发条件：vsync 上升沿

处理流程：该中断暂无处理需求。

4.7.2 cfg_finish

触发条件：上一次更新的配置已被模块取走用于回写，可更新下一帧回写的配置。

处理流程：

- 如果有空闲 buffer，则将空闲 buffer 的地址提交给控制器作为下一次回写的输出地址。
- 如果没有空闲 buffer，则开启工作函数，延时等待空闲 buffer，直到有空闲 buffer 可以用于回写。

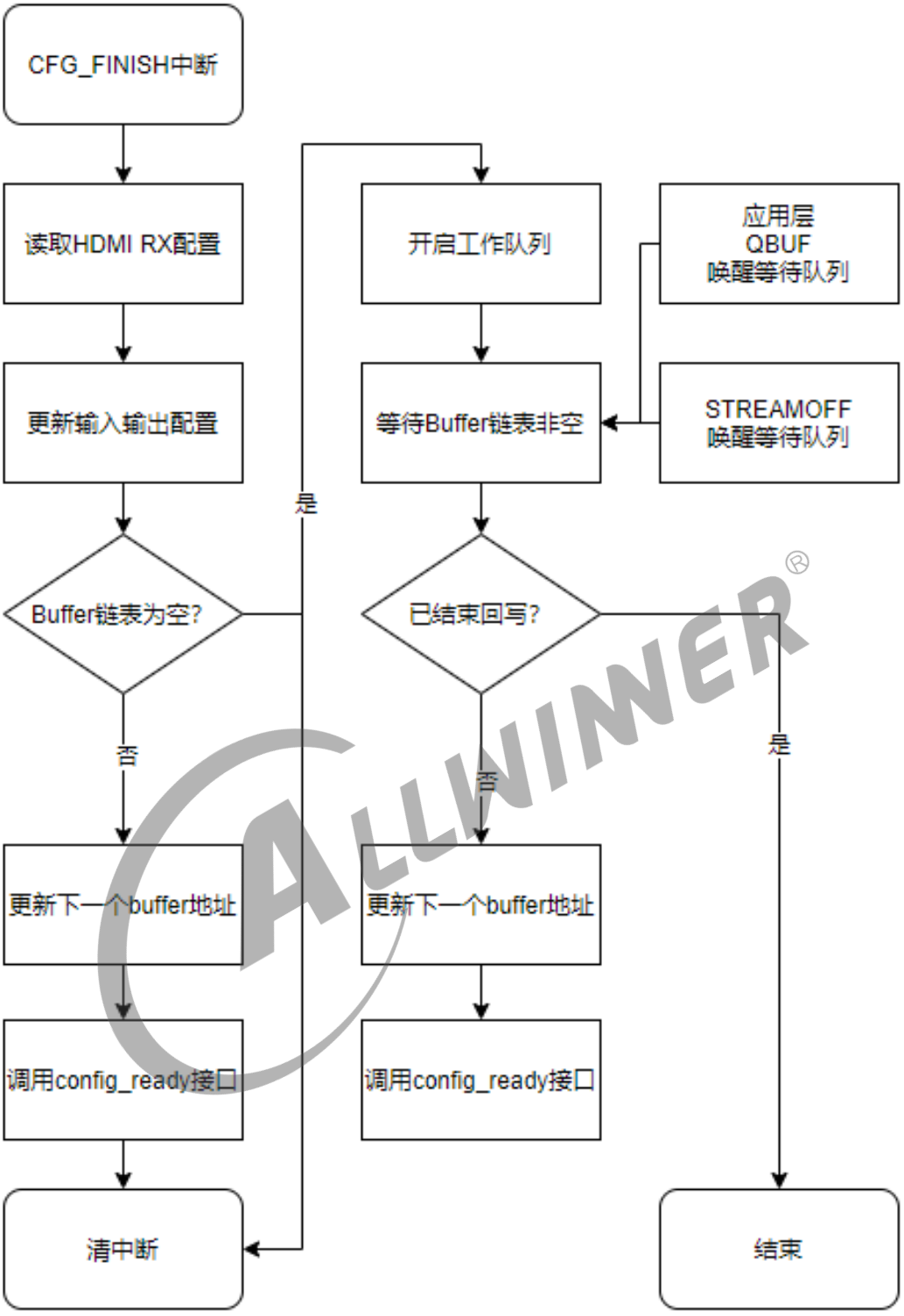

图 4-5: HRC_CFG_Finish 中断流程图

4.7.3 wb_finish

触发条件：当前帧回写完成。

处理流程：将回写完成的 buffer 上报给 V4L2 框架，框架会告知应用层，让应用层读取这个 buffer。

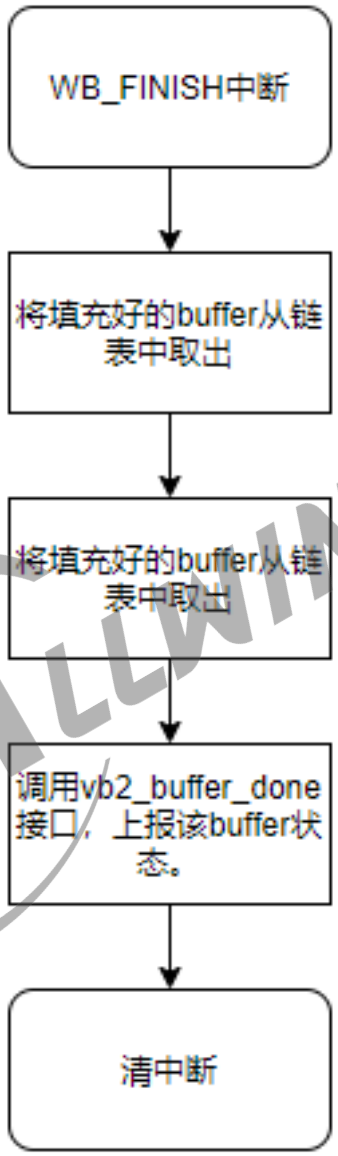

图 4-6: HRC_WB_Finish 中断流程图

4.7.4 timeout

触发条件：回写还未完成，但是下一帧已经来到。

处理流程：调用 sunxi_hrc_hardware_reset 重置硬件，并且重新 commit 一次，触发新的回写。

4.7.5 overflow

触发条件：输入源是 HDMI RX，且 mbus master 处于 busy 状态。

处理流程：调用 sunxi_hrc_hardware_reset 重置硬件，并且重新 commit 一次，触发新的回写。

4.7.6 unusual

触发条件：宽高设置与 HDMI RX 的输入数据不匹配。

处理流程：这个中断异常通常伴随 HDMI RX 的时序不稳定，通常会触发应用层重新开始回写。

5 结束语

HRC 模块功能相对简单，但是影响的是 HDMIRX 到显示的通路，所以承担的功能是关键且重要的。

异常时优先排查：输入和输出参数的配置、中断的响应、Buffer 的流转。

著作权声明

商标声明

免责声明