Rockchip Linux USB Host UVC 问题排查

文件标识：RK-PC-YF-493

发布版本：V1.0.0

日期：2023-11-27

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文档提供基于 Linux 内核的 USB Host UVC (USB Video Class) 常见问题的排查方法和解决方法。目的是帮助软件开发工程师和技术支持工程师快速定位和解决 UVC 相关问题。

芯片名称	内核版本
所有 Rockchip 芯片(MCU 除外)	Linux-4.19 及以上版本

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

日期	版本	作者	修改说明
2023-11-27	V1.0.0	吴良峰、龙跃	初始版本

目录

1. UVC 通信流程

如下图 1 是基于 Linux 内核的 UVC 通信流程。图 1 左侧描述的是 USB Device 端的 UVC 相关模块和对应的内核代码路径。图 1 右侧描述的是 USB Host 端的 UVC 相关模块和对应的内核代码路径。本文档主要说明 USB Host 端的 UVC 驱动问题排查方法，并且所给出的调试方法主要是基于 Rockchip 平台 Linux-5.10 内核，因此，不一定适用于所有内核版本，也不适用于排查 UVC 应用的相关问题。由图 1 右侧的流程可以看出，UVC Host 端的应用可以使用两种方式与内核 UVC 设备进行数据通信：一种是通过 V4L2 框架调用 UVC 驱动的接口进行通信；另外一种是通过 libusb 调用 USB 通用驱动 Devio 的接口进行通信。这两种通信方式在 Rockchip 平台上都可以支持，实际使用哪种通信方式由 UVC 应用决定。

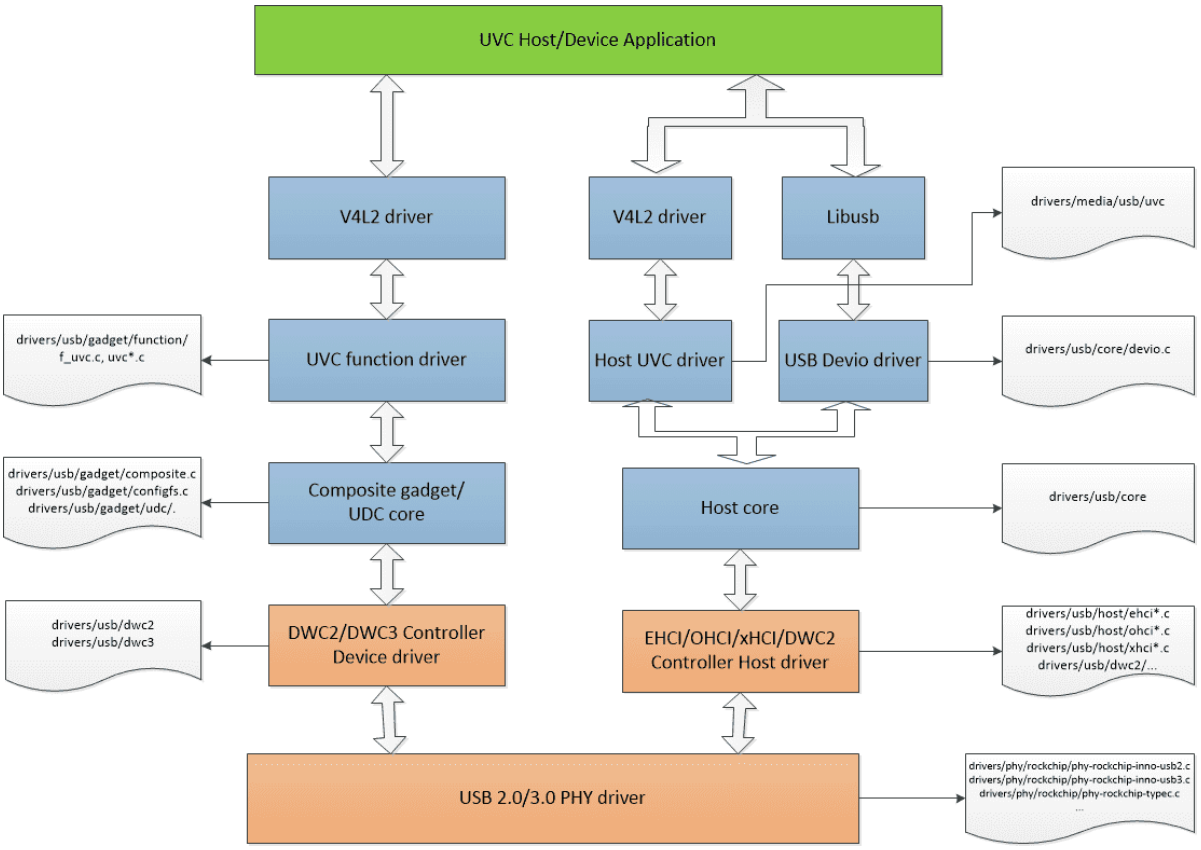

图 1 Linux UVC 通信流程

2. UVC 常用调试工具

2.1 USB 协议分析仪

USB 协议分析仪可以用于实时捕获 USB 总线上的通信数据，实现 USB 数据抓包和协议分析。如下图 2 是常见的 USB 2.0 协议分析仪的连接方法，使用时，需要将 USB 分析仪串接在 USB Device 和 Host 中间，同时，需要在电脑端安装相应的协议分析软件。USB 协议分析仪是用于定位问题点在 Device 端还是 Host 端的最有效和最直接的方法。

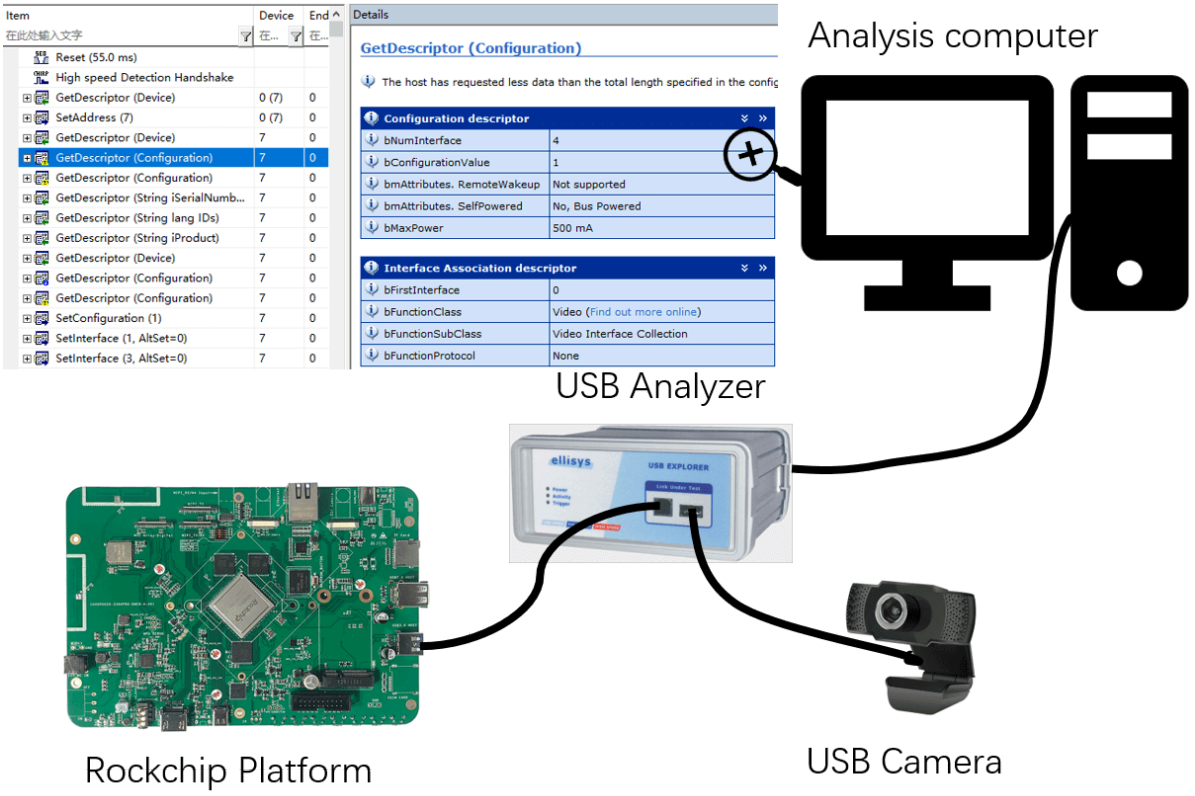

图 2 USB 协议分析仪的连接示意图

- USB 协议分析仪可以用于分析 UVC 常见的几类问题：

1. UVC 打开预览图像失败问题；
2. UVC 预览图像延时问题；
3. UVC 预览图像卡住问题；
4. UVC 帧率统计和图像数据解析；
5. UVC 驱动加载失败的问题；

- USB 协议分析仪不适用于分析 UVC 常见的几类问题：

1. USB 信号质量问题；
2. UVC 异常断开问题；
3. UVC 通信过程中触发 USB 控制器 EMI 报错问题；

可以使用高带宽的示波器，测量 USB 眼图信号质量、USB 高速握手信号、USB VBUS 电源塌陷情况等，来分析 USB 信号问题、异常断开问题以及 EMI 问题。

2.2 usbmon 抓包工具和分析工具

usbmon 即 usb monitor，是 Linux 内核提供的 USB 抓包工具，用于捕获 USB 核心层和设备类驱动程序（如 UVC 驱动）之间交互的数据，以及捕获 USB 核心层和 USB Host 控制器驱动之间交互的数据。通过 usbmon 抓数据包，并使用数据包分析工具（如： vusb-analyzer 或者 Wireshark）解析 USB 通信协议，可以直观地看到内核 UVC 数据的处理流程，帮助我们分析和定位 UVC 驱动问题和 USB 控制器驱动问题。

usbmon 驱动模块位置在内核 `drivers/usb/mon/`，使能 usbmon 功能的方法：

1. 内核使能 CONFIG_USB_MON

2. 挂载 debugfs 文件系统

   `mount -t debugfs none_debugs /sys/kernel/debug`

   如果提示忙，表示当前系统已经默认挂载 debugfs 文件系统。

   大多数的 Rockchip SDK 发行版本中，默认已经支持 usbmon，查看系统是否支持 usbmon 的方法：

   `ls /sys/kernel/debug/usb/usbmon/`

数据包分析工具：vusb-analyzer 下载地址：https://vusb-analyzer.sourceforge.net/download.html

Ubuntu 安装方法：`sudo apt-get install vusb-analyzer`

以 RK3588 EVB1 USB2 HOST0 连接 UVC 外设的应用场景为例：

1. 查看 usbmon 可以识别到的 USB 总线编号

   数字 1/2/3 等表示当前平台所拥有的 USB 总线，每个 USB 设备都会挂在一条总线下；数字 0 表示抓所有总线上的包；数字后面的 s/t/u 表示抓包保存的数据格式，通常使用 u 格式。

```
console:/sys/kernel/debug/usb/usbmon # ls
0s  1s  1u  2t  3s  3u  4t  5s  5u  6t
0u  1t  2s  2u  3t  4s  4u  5t  6s  6u
```

2. 找到当前要监控的设备使用的总线编号

   输入 lsusb 命令，根据厂商 ID 和 产品 ID 进行区分。

```
console:/sys/kernel/debug/usb/usbmon # lsusb
Bus 005 Device 001: ID 1d6b:0002
Bus 003 Device 001: ID 1d6b:0001
Bus 002 Device 005: ID 046d:0823
Bus 001 Device 001: ID 1d6b:0002
Bus 006 Device 001: ID 1d6b:0003
Bus 004 Device 001: ID 1d6b:0001
Bus 002 Device 001: ID 1d6b:0002
```

   其中，Bus 002 表示 2 号总线。ID 046d:0823 对应要监控的 USB 设备的厂商 ID 和产品 ID，说明要监控的 USB 设备使用 2 号总线。

3. 使用 usbmon 抓取通信数据包

   `cat  /sys/kernel/debug/usb/usbmon/2u > /data/usbmon.mon`

4. 使用 vusb-analyzer 工具分析数据包

   `vusb-analyzer usbmon.mon`

2.3 v4l2-ctl 工具

v4l2-ctl 工具用于控制基于 Linux V4L2 框架的 video 设备（/dev/video*）。通过 v4l2-ctl 工具，可以对 video 设备进行各种操作，包括：读取设备支持的格式、以特定的分辨率和格式抓图等。Linux UVC 驱动基于 V4L2 框架实现，在调试 UVC 问题时，如果怀疑与 UVC 应用相关，可以使用 v4l2-ctl 工具操作 UVC 设备，排除 UVC 应用的影响。

1. **v4l2-ctl --list-devices**

   列出所有设备（包括所有通过 V4L2 框架注册的 video 设备）

示例：

```
console:/ # v4l2-ctl --list-devices
UVC Camera (046d:0823) (usb-fc800000.usb-1):
        /dev/video21
        /dev/video22
```

一个 USB camera 对应两个设备：一个是图像/视频采集，一个是 metadata 采集。

2. **v4l2-ctl --list-formats-ext --device path/to/video_device**

   列出指定设备的预览支持格式

   示例：

```
console:/ # v4l2-ctl --list-formats-ext --device /dev/video21
ioctl: VIDIOC_ENUM_FMT
        Index       : 0
        Type        : Video Capture
        Pixel Format: 'YUYV'
        Name        : YUYV 4:2:2
                Size: Discrete 640x480
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.042s (24.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.133s (7.500 fps)
                        Interval: Discrete 0.200s (5.000 fps)
         ......
```

3. **v4l2-ctl --all --device path/to/video_device**

   获取指定设备的所有信息

   示例：

```
console:/ # v4l2-ctl --all --device /dev/video21
Driver Info:
        Driver name      : uvcvideo
        Card type        : UVC Camera (046d:0823)
        Bus info         : usb-fc800000.usb-1
        Driver version   : 5.10.198
        Capabilities     : 0x84a00001
                Video Capture
                Metadata Capture
                Streaming
                Extended Pix Format
                Device Capabilities
        Device Caps      : 0x04200001
                Video Capture
                Streaming
                Extended Pix Format
        ......
```

4. **v4l2-ctl --device path/to/video_device --set-fmt-video=width=width,height=height,pixelformat=MJPG --stream-mmap --stream-to=path/to/output.jpg --stream-count=1**

   从特定设备以特定分辨率和格式抓图

   示例：

   从 video21 设备以 1920x1080 分辨率 MJPG 格式抓 1 帧图像并保存在 /sdcard/DCIM/ 路径下。

   ```
   v4l2-ctl --device /dev/video21 --set-fmt-video=width=1920,height=1080,pixelformat=MJPG --stream-mmap --stream-to=/sdcard/DCIM/output_1920.jpg --stream-count=1
   ```

5. **v4l2-ctl --device path/to/video_device --set-fmt-video=width=width,height=height,pixelformat=format --stream-mmap --stream-to=path/to/output --stream-count=number_of_frames_to_capture**

   从特定设备以特定分辨率抓流

   示例1：

   从 video21 设备以 1920x1080 分辨率 MJPG 格式抓 100 帧图像流并保存在 /sdcard/DCIM/ 路径下。

   ```
   v4l2-ctl --device /dev/video21 --set-fmt-video=width=1920,height=1080,pixelformat=MJPG --stream-mmap --stream-to=/sdcard/DCIM/output.mjpg --stream-count=100
   ```

   示例2：

   从 video21 设备以 1920x1080 分辨率 MJPG 格式持续抓图像流。

   ```
   v4l2-ctl --device /dev/video21 --set-fmt-video=width=1920,height=1080,pixelformat=MJPG --stream-mmap
   ```

3. UVC 内核调试接口

3.1 UVC 驱动调试接口

UVC 驱动模块位置在内核 `drivers/media/usb/uvc` ，提供了基于 system module 的调试接口，可以用于动态使能 UVC 驱动的 trace 信息，默认关闭 trace 信息功能。

`/sys/module/uvcvideo/parameters/trace`

动态使能 trace：

`echo 0xffff > /sys/module/uvcvideo/parameters/trace`

信息打印等级为 KERN_DEBUG，可以通过执行命令 dmesg 来查看 trace 信息。也可修改内核的 printk 等级，默认输出到串口终端：

`echo 8 > /proc/sys/kernel/printk`

动态关闭 trace：

`echo 0 > /sys/module/uvcvideo/parameters/trace`

以 RK3588 USB2 HOST0 连接 UVC 外设，打开预览图像时的 UVC trace 为例，打印的信息如下：

```
[   60.780597][  T477] uvcvideo: uvc_v4l2_open
[   60.843298][  T477] uvcvideo: Resuming interface 2
[   60.843318][  T477] uvcvideo: Resuming interface 3
[   60.852317][  T477] uvcvideo: uvc_v4l2_release
[   60.855127][  T477] uvcvideo: uvc_v4l2_open
[   60.877415][  T477] uvcvideo: Trying format 0x47504a4d (MJPG): 2592x1944.
[   60.877525][  T477] uvcvideo: Using default frame interval 100000.0 us (10.0
fps).
[   60.910575][  T477] uvcvideo: Device requested 3060 B/frame bandwidth.
[   60.910622][  T477] uvcvideo: Selecting alternate setting 11 (3060 B/frame
bandwidth).
[   60.913504][  T477] uvcvideo: Allocated 5 URB buffers of 32x3060 bytes each.
[   60.913836][  T477] uvcvideo: uvc_v4l2_poll
[   62.010310][    C0] uvcvideo: Frame complete (EOF found).
[   62.010438][    C0] uvcvideo: frame 1 stats: 1111/763/1876 packets,
1/1111/1876 pts (early initial), 1875/1876 scr, last pts/stc/sof
223854994/228661160/514
```

3.2 V4L2 调试接口

Linux UVC 驱动基于 V4L2 框架，实现与 UVC 应用程序进行 queue buf 和 dequeue buf 的操作。

V4L2 框架实现了基于内核 trace 框架的 tracepoint[1]，可以用于跟踪 queue buf，dequeue buf 以及 buf done 的流程。

内核需要使能如下 Trace 相关配置

```
Kernel hacking  --->
    [*] Tracers  --->
        [*]   Trace process context switches and events
        [*]   Enable uprobes-based dynamic events
        [*]   Trace gpio events:
```

V4L2 支持如下的 trace events：

```
/sys/kernel/debug/tracing/events/vb2# ls
enable  filter  vb2_buf_done  vb2_buf_queue  vb2_dqbuf  vb2_qbuf
/sys/kernel/debug/tracing/events/v4l2]# ls
enable  v4l2_dqbuf  vb2_v4l2_buf_done   vb2_v4l2_dqbuf
filter  v4l2_qbuf   vb2_v4l2_buf_queue  vb2_v4l2_qbuf
```

详细的使用方法，请参考 `/sys/kernel/debug/tracing/README`

3.3 USB 控制器动态初始化的调试接口

`/sys/bus/platform/drivers/[usb_controller_name]/bind, unbind`

Note：命令中 '[usb_controller_name]' 需要修改为芯片对应的 USB 控制器的名称。

当 UVC 数据传输出现异常，且无法自动恢复传输时，可以尝试通过手动触发 USB 控制器重新初始化，来定位是否与 USB 控制器异常有关。

以 RK3588 EVB1 平台为例：

```
/* USB2 HOST0/1 OHCI 控制器驱动 */
console:/sys/bus/platform/drivers/ohci-platform # ls
bind  fc840000.usb  fc8c0000.usb  uevent  unbind

/* USB2 HOST0/1 EHCI 控制器驱动 */
console:/sys/bus/platform/drivers/ehci-platform # ls
bind  fc800000.usb  fc880000.usb  uevent  unbind

/* USB3 OTG1_HOST 控制器驱动 */
console:/sys/bus/platform/drivers/xhci-hcd # ls
bind  uevent  unbind  xhci-hcd.8.auto

/* USB3 OTG0/1 控制器驱动 */
console:/sys/bus/platform/drivers/dwc3 # ls
bind  fc000000.usb  fc400000.usb  uevent  unbind

执行如下命令，可以重新初始化 USB2 HOST0 接口对应的 OHCI/EHCI 控制器
echo fc840000.usb > /sys/bus/platform/drivers/ohci-platform/unbind
echo fc800000.usb > /sys/bus/platform/drivers/ehci-platform/unbind
echo fc800000.usb > /sys/bus/platform/drivers/ehci-platform/bind
echo fc840000.usb > /sys/bus/platform/drivers/ohci-platform/bind
```

4. UVC 调试方法

4.1 USB 中断绑核的方法

作用：可以用来分析和解决系统在高负载的场景下，因为 USB 控制器中断处理慢而造成的 UVC 预览花屏、UVC 帧率不稳定等问题。

Linux 内核提供如下节点，用于设置 CPU 和某中断的亲和性，该文件存放的是 CPU 列表（十进制）。注意，CPU 核心个数用表示编号从 0 开始，如 CPU0, CPU1 等。

`/proc/irq/[irq_num]/smp_affinity_list`

以 RK3588 EVB1 为例，将 USB OTG0 控制器的中断绑定到 CPU1 的方法：

```
console:/ # cat /proc/interrupts | grep dwc
153:         77          0          0          0          0          0          0
         0     GICv3 252 Level     dwc3
console:/ # echo 1 > /proc/irq/153/smp_affinity_list
```

4.2 关闭 UVC auto suspend 的方法

作用：可以用来分析和解决部分 UVC 外设在 Rockchip 平台上的 auto suspend 兼容性问题。

在调试这类问题时，可以通过如下命令，动态关闭 USB autosuspend 功能，初步定位问题。

```
for i in $(find /sys -name control | grep usb);
do echo on > $i;
echo "echo on > $i";
done;
```

假如关闭 auto suspend 功能后，可以解决特定 UVC 外设使用过程中遇到的问题，说明该 UVC 外设在 Rockchip 平台上存在 auto suspend 兼容性问题。可以通过修改 USB 驱动，将 UVC 外设加入的 auto suspend 的黑名单中。比如：将 UVC 外设（VID,PID=0x05a3,0x9230）加入黑名单，Linux-5.10 内核参考修改如下：

```
diff --git a/drivers/usb/core/quirks.c b/drivers/usb/core/quirks.c
index 76ac5d6555ae..d799e93b9a0d 100644
--- a/drivers/usb/core/quirks.c
+++ b/drivers/usb/core/quirks.c
@@ -322,6 +322,9 @@ static const struct usb_device_id usb_quirk_list[] = {
        /* Alcor Micro Corp. Hub */
        { USB_DEVICE(0x058f, 0x9254), .driver_info = USB_QUIRK_RESET_RESUME },

+       /* HD Camera Manufacturer */
+       { USB_DEVICE(0x05a3, 0x9230), .driver_info = USB_QUIRK_AUTO_SUSPEND },
```

如下表 1 列出了已知存在 auto suspend 兼容性问题的 UVC 外设信息。

表 1 USB Camera auto suspend blacklist

Vendor ID (VID)	Product ID (PID)	Manufacturer
0x05a3	0x9230	Microdia
0x05a3	0x9320	Microdia
0x0c45	0x6321	Sonix
0x0c45	0x636a	Sonix
0x0c45	0x636b	Sonix
0x0c45	0x64ab	Sonix
0x0c45	0x64ac	Sonix
0x0e8d	0x7668	unknown
0x2bc5	0x051f	MediaTek
0x1e10	0x4000	Point Grey Research
0x15aa	0x1555	Gearway Electronics

4.3 关闭 CPU IDLE 的方法

作用：可以用来分析 CPU 退出 IDLE 耗时过长导致 USB 中断处理慢，从而造成的 UVC 预览花屏、UVC 帧率不稳定等问题。

以 RK3588 EVB1 平台为例：

```
cpunum=$(cat /proc/cpuinfo | grep processor | wc -l)
for i in `seq 0 $(($cpunum-1))`
do
echo 1 > /sys/devices/system/cpu/cpu$i/cpuidle/state0/disable
echo 1 > /sys/devices/system/cpu/cpu$i/cpuidle/state1/disable
done
```

注："state*" 节点在不同平台上有差异。比如：RK3399 支持 state0/1/2 三个节点。

4.4 设置 CPU 定高频的方法

作用：可以用来分析和解决系统在高负载的场景下，UVC 帧率不稳定的问题。

修改 CPU 调度策略为 'performance'，该模式会让 CPU 始终工作在最高频率，以获取最大的性能。

```
cpunum=$(cat /proc/cpuinfo | grep processor | wc -l)
for i in `seq 0 $(($cpunum-1))`
do
echo performance > /sys/devices/system/cpu/cpu$i/cpufreq/scaling_governor
done
```

4.5 设置 DDR 定高频的方法

作用：可以用来分析和解决因为 DDR 动态变频导致 UVC 预览失败、UVC 预览花屏或者 UVC 帧率不稳定等问题。

Rockchip 平台大部分支持 DDR 动态变频功能，用于优化系统运行功耗。但如下两种 UVC 场景，可能会受 DDR 动态变频的影响：

场景1. 打开 UVC 预览的瞬间，如果 DDR 跑低频，会概率性导致预览失败；

场景2. 在 UVC 预览过程中，如果 DDR 跑低频，会概率性导致 UVC 帧率不稳定或者压缩格式的图像花屏现象。

可以通过如下两种方法设置 DDR 运行于最高频率。方法 1 和方法 2的区别是：方法 1 主要用于调试阶段，无论 UVC 是否工作，DDR 始终运行于最高频率。方法 2 适用于产品量产，只在 UVC 工作时，设置 DDR 始终运行于最高频率。当 UVC 不工作时，DDR 变频仍然由系统负载决定。

方法1. 通过命令行设置 DDR 运行于最高频率

以 RK3588 平台为例

```
console:/ # cd /sys/class/devfreq/dmc
console:/sys/class/devfreq/dmc # cat max_freq
2112000000
console:/sys/class/devfreq/dmc # echo userspace > governor
console:/sys/class/devfreq/dmc # echo 2112000000 > userspace/set_freq
```

方法2. 通过更改 UVC 驱动设置 DDR 运行于高频 [推荐]

Linux-4.19 参考修改如下：

```
From 970e4749125d685dc03ea32e3ae77e6f0ab7cff7 Mon Sep 17 00:00:00 2001
From: "william.wu" <william.wu@rock-chips.com>
Date: Sat, 11 Nov 2023 15:41:10 +0800
Subject: [PATCH] media: uvcvideo: set system status to performance when stream
 on

For rockchip platforms, set performance frequency for the
memory controller when uvc stream on, and clear it after
uvc stream off. It can improve uvc streaming stability.

Signed-off-by: william.wu <william.wu@rock-chips.com>
Change-Id: I1dc0cf10c552bba2c3a0f8a1bb37d90f546eb4a3
---
 drivers/media/usb/uvc/uvc_video.c | 6 ++++++
 1 file changed, 6 insertions(+)

diff --git a/drivers/media/usb/uvc/uvc_video.c
b/drivers/media/usb/uvc/uvc_video.c
index ef3832b03709..d2158c94ae47 100644
--- a/drivers/media/usb/uvc/uvc_video.c
+++ b/drivers/media/usb/uvc/uvc_video.c
@@ -25,6 +25,8 @@
 #include <media/v4l2-common.h>
```

```
 #include "uvcvideo.h"
+#include <dt-bindings/soc/rockchip-system-status.h>
+#include <soc/rockchip/rockchip-system-status.h>

 /* ------------------------------------------------------------------------
  * UVC Controls
@@ -2149,6 +2151,7 @@ int uvc_video_enable(struct uvc_streaming *stream, int
enable)
        }

        uvc_video_clock_cleanup(stream);
+       rockchip_clear_system_status(SYS_STATUS_PERFORMANCE);
        return 0;
    }

@@ -2161,6 +2164,8 @@ int uvc_video_enable(struct uvc_streaming *stream, int
enable)
    if (ret < 0)
        goto error_commit;

+   rockchip_set_system_status(SYS_STATUS_PERFORMANCE);
+
    ret = uvc_init_video(stream, GFP_KERNEL);
    if (ret < 0)
        goto error_video;
@@ -2168,6 +2173,7 @@ int uvc_video_enable(struct uvc_streaming *stream, int
enable)
    return 0;

 error_video:
+   rockchip_clear_system_status(SYS_STATUS_PERFORMANCE);
    usb_set_interface(stream->dev->udev, stream->intfnum, 0);
 error_commit:
    uvc_video_clock_cleanup(stream);
--
2.34.1
```

Linux-5.10 参考修改如下：

```
From ab0546fdd01a8a8df0f15d0c648fa73aba76d173 Mon Sep 17 00:00:00 2001
From: William Wu <william.wu@rock-chips.com>
Date: Wed, 8 Nov 2023 09:49:48 +0800
Subject: [PATCH] media: uvcvideo: set system status to performance when stream
 on

For rockchip platforms, set performance frequency for the
memory controller when uvc stream on, and clear it after
uvc stream off. It can improve uvc streaming stability.

Signed-off-by: William Wu <william.wu@rock-chips.com>
Change-Id: I1dc0cf10c552bba2c3a0f8a1bb37d90f546eb4a3
---
 drivers/media/usb/uvc/uvc_video.c | 6 ++++++
 1 file changed, 6 insertions(+)
```

```
diff --git a/drivers/media/usb/uvc/uvc_video.c
b/drivers/media/usb/uvc/uvc_video.c
index 03dfe96bceba..79ffe93b19ae 100644
--- a/drivers/media/usb/uvc/uvc_video.c
+++ b/drivers/media/usb/uvc/uvc_video.c
@@ -20,6 +20,8 @@
 #include <media/v4l2-common.h>

 #include "uvcvideo.h"
+#include <dt-bindings/soc/rockchip-system-status.h>
+#include <soc/rockchip/rockchip-system-status.h>

 /* ------------------------------------------------------------------------
  * UVC Controls
@@ -2139,6 +2141,8 @@ int uvc_video_start_streaming(struct uvc_streaming *stream)
     if (ret < 0)
         goto error_commit;

+    rockchip_set_system_status(SYS_STATUS_PERFORMANCE);
+
     ret = uvc_video_start_transfer(stream, GFP_KERNEL);
     if (ret < 0)
         goto error_video;
@@ -2146,6 +2150,7 @@ int uvc_video_start_streaming(struct uvc_streaming *stream)
     return 0;

 error_video:
+    rockchip_clear_system_status(SYS_STATUS_PERFORMANCE);
     usb_set_interface(stream->dev->udev, stream->intfnum, 0);
 error_commit:
     uvc_video_clock_cleanup(stream);
@@ -2176,4 +2181,5 @@ void uvc_video_stop_streaming(struct uvc_streaming *stream)
     }

     uvc_video_clock_cleanup(stream);
+    rockchip_clear_system_status(SYS_STATUS_PERFORMANCE);
 }
--
2.34.1
```

4.6 解析 UVC 图像帧的方法

作用：可以用于分析 UVC 预览或者录像场景时图像花屏的问题，辅助定位花屏现象是 UVC 外设的问题还是 UVC Host 的问题。

以解析 Logitech UVC Camera（idVendor=046d, idProduct=0823）的 UVC MJPG 格式的图像帧（支持高带宽同步传输方式）为例，抓取 UVC 数据和解析 UVC 图像帧的方法如下：

1. 使用 Ellisys USB2.0 协议分析仪抓 UVC MJPG 预览图像过程中的 USB 总线通信数据；
2. 参考图 3 的步骤，导出 Text 格式的文件（File -> Export -> Transactions data -> Text），并保存文件；
3. 用文本编辑器打开 UVC txt 文件，并参考 UVC 协议规范[2]中关于 "Format of the Payload Header " 的描述，找到 UVC 帧头信息。参考图 4，UVC 帧头长度为 0x0C，UVC 帧头信息为 0x8C；

4. 使用 Python 脚本 uvc-data.py 解析 UVC txt 文件，传入 UVC 帧头信息，执行脚本后，可以自动保存 .jpg 格式的 UVC 帧图像；

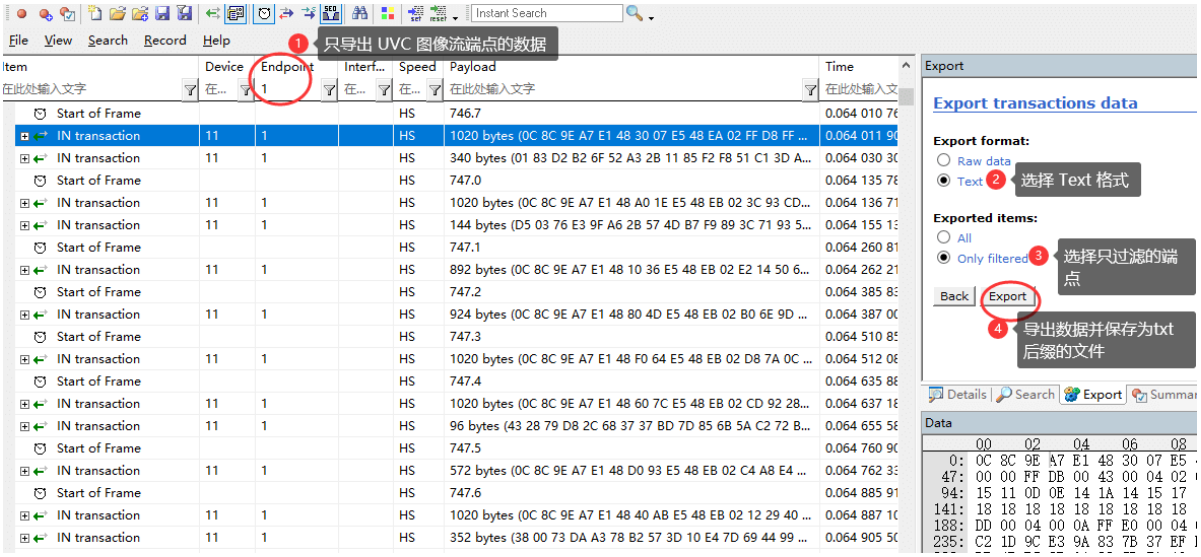

图 3 UVC 分析仪数据

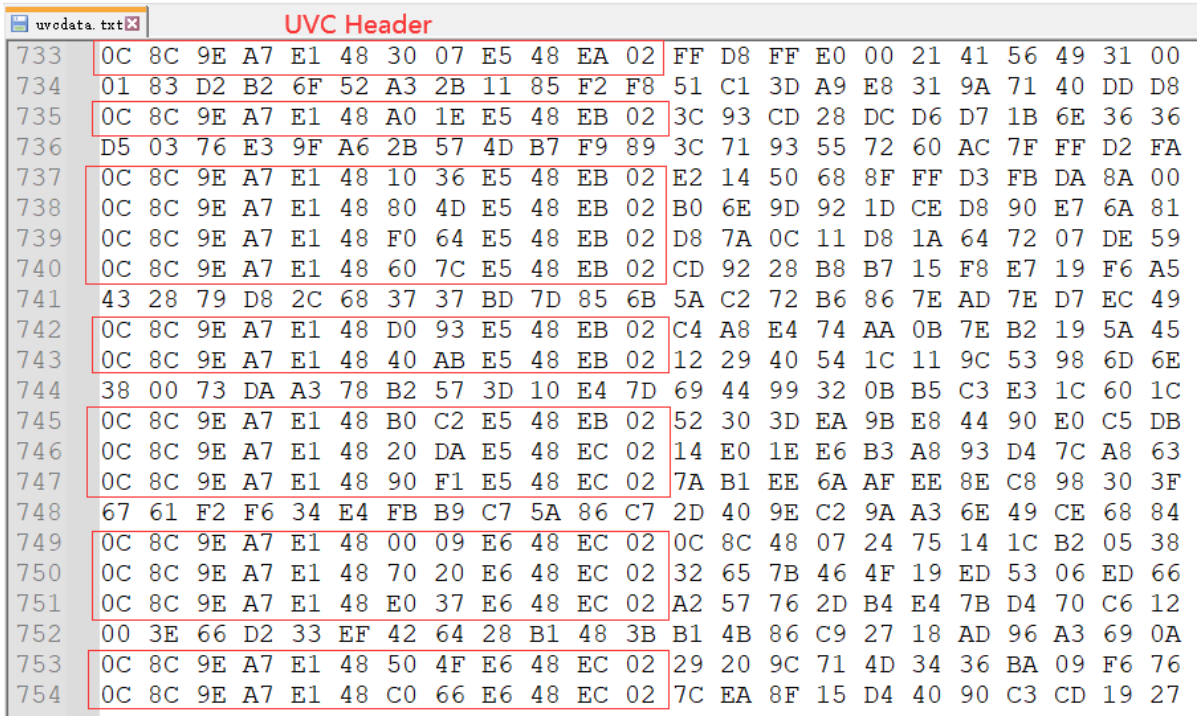

图 4 UVC 文本数据

解析 UVC 图像帧的 Python 脚本 uvc-data.py：

```
# -*- coding: utf-8 -*-
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--file', '-f', type=str, required=True, default='',
help='The txt format file of uvc data')
parser.add_argument('--header_len', '-l', type=str, required=True, default='',
help='Length of uvc header in bytes')
parser.add_argument('--header_info','-i', type=str, required=True, default='',
help='Information of uvc header')
parser.add_argument('--count', '-n', type=int, default=10, help='The count of
images to saved')
```

```
args = parser.parse_args()

def get_photo(cnt = 0):
    src = open(args.file, 'r')
    uvc_header_len = int(args.header_len, 16)
    uvc_header_info = int(args.header_info, 16)
    uvc_header_even_frame = uvc_header_info & 0xFC
    uvc_header_even_frame_eof = uvc_header_even_frame | 0x02
    uvc_header_odd_frame = (uvc_header_info & 0xFC) | 0x01
    uvc_header_odd_frame_eof = (uvc_header_odd_frame | 0x02)
    pic = []
    num = 0
    eof = 0
    uvc_first_frame = 0
    uvc_payload_offset = 0

    print('UVC Header Length:        ', hex(uvc_header_len), 'Bytes')
    print('UVC Header EVEN Frame:    ', hex(uvc_header_even_frame))
    print('UVC Header EVEN Frame EOF:', hex(uvc_header_even_frame_eof))
    print('UVC Header ODD Frame:     ', hex(uvc_header_odd_frame))
    print('UVC Header ODD Frame EOF: ', hex(uvc_header_odd_frame_eof))

    for line in src:
        line = line.split(' ')
        data0 = int(line[0], 16)
        data1 = int(line[1], 16)

        if ((data0 == uvc_header_len) and
            ((data1 == uvc_header_even_frame) or (data1 ==
uvc_header_odd_frame))):
            uvc_payload_offset = uvc_header_len
            uvc_first_frame = 1
        elif ((data0 == uvc_header_len) and
            ((data1 == uvc_header_even_frame_eof) or
            (data1 == uvc_header_odd_frame_eof) or
            (data1 == ((uvc_header_even_frame_eof & 0x0F) | 0x10)) or
            (data1 == ((uvc_header_odd_frame_eof & 0x0F) | 0x10)))):
            uvc_payload_offset = uvc_header_len
            uvc_first_frame = 0
            eof = 1
        else:
            uvc_payload_offset = 0
            uvc_first_frame = 0
            eof = 0

        if (eof == 1 and uvc_first_frame == 1):
            path = 'photo_' + str(num) + '.jpg'
            print(path + ' Saved ' + str(len(pic)) + 'Bytes')
            dst = open(path, 'wb+')
            dst.write(bytes.fromhex(''.join(pic)))
            dst.close()
            num += 1
            pic = []
            eof = 0
            if (cnt != 0 and cnt == num):
                break
```

```
        if (len(line) > uvc_payload_offset):
            for i in range(uvc_payload_offset, len(line)):
                pic.append(line[i])

    src.close()

get_photo(args.count)
```

使用示例：

```
# 查看 uvc-data.py 使用方法
# -f: 待解析的 uvc 数据文本文件，必要参数;
# -l: 待解析的  帧头长度，必要参数;
# -i: 待解析的 uvc 帧头信息，必要参数;
# -n: 待解析和保存的 uvc 帧图像数量;
python uvc-data.py -h
usage: uvc-data.py [-h] --file FILE --header_len HEADER_LEN --header_info
HEADER_INFO [--count COUNT]

options:
  -h, --help            show this help message and exit
  --file FILE, -f FILE  The txt format file of uvc data
  --header_len HEADER_LEN, -l HEADER_LEN
                        Length of uvc header in bytes
  --header_info HEADER_INFO, -i HEADER_INFO
                        Information of uvc header
  --count COUNT, -n COUNT
                        The count of images to saved
# 解析 uvcdata.txt，uvc 帧头长度为 0x0c，uvc 帧头信息为 0x8c，保存 5 张图象
python uvc-data.py -f uvcdata.txt -l 0c -i 8c -n 5
UVC Header Length:         0xc Bytes
UVC Header EVEN Frame:     0x8c
UVC Header EVEN Frame EOF: 0x8e
UVC Header ODD Frame:      0x8d
UVC Header ODD Frame EOF:  0x8f
photo_0.jpg Saved 516573Bytes
photo_1.jpg Saved 827193Bytes
photo_2.jpg Saved 827585Bytes
photo_3.jpg Saved 3120193Bytes
photo_4.jpg Saved 188340Bytes
```

4.7 优化 UVC 驱动的方法

1. **增加 UVC URB 数量**

   作用：提高 UVC 驱动对 USB 控制器中断响应 latency 的容忍度

   影响：UVC URB 会缓存 UVC 帧数据，增加预览延时

   Linux-4.19 参考修改如下：

```
diff --git a/drivers/media/usb/uvc/uvcvideo.h b/drivers/media/usb/uvc/uvcvideo.h
index 96632cc58d77..3761946af110 100644
--- a/drivers/media/usb/uvc/uvcvideo.h
+++ b/drivers/media/usb/uvc/uvcvideo.h
@@ -173,7 +173,7 @@
 #define DRIVER_VERSION         "1.1.1"

 /* Number of isochronous URBs. */
-#define UVC_URBS               5
+#define UVC_URBS               32
 /* Maximum number of packets per URB. */
 #define UVC_MAX_PACKETS                32
 /* Maximum status buffer size in bytes of interrupt URB. */
```

2. **修改 UVC DMA buffer 分配方式**

   作用：将 UVC DMA buffer 的分配方式，由默认的 usb_alloc_coherent 改为 kmalloc，提高 UVC 驱动软件读取和解析 buffer data 的效率。

   影响：需要由 USB 驱动保证 DMA buffer 的cache 一致性。Rockchip 平台 USB 驱动默认已经支持，软件不用额外修改。

```
diff --git a/drivers/media/usb/uvc/uvc_video.c
b/drivers/media/usb/uvc/uvc_video.c
index d2158c94ae47..fff3c710bc97 100644
--- a/drivers/media/usb/uvc/uvc_video.c
+++ b/drivers/media/usb/uvc/uvc_video.c
@@ -1604,7 +1604,7 @@ static void uvc_free_urb_buffers(struct uvc_streaming
*stream)
                struct uvc_urb *uvc_urb = &stream->uvc_urb[i];

                if (uvc_urb->buffer) {
-#ifndef CONFIG_DMA_NONCOHERENT
+#if 0
                        usb_free_coherent(stream->dev->udev, stream->urb_size,
                                          uvc_urb->buffer, uvc_urb->dma);
 #else
@@ -1651,7 +1651,7 @@ static int uvc_alloc_urb_buffers(struct uvc_streaming
*stream,
                        struct uvc_urb *uvc_urb = &stream->uvc_urb[i];

                        stream->urb_size = psize * npackets;
-#ifndef CONFIG_DMA_NONCOHERENT
+#if 0
                        uvc_urb->buffer = usb_alloc_coherent(
                                stream->dev->udev, stream->urb_size,
                                gfp_flags | __GFP_NOWARN, &uvc_urb->dma);
@@ -1768,7 +1768,7 @@ static int uvc_init_video_isoc(struct uvc_streaming
*stream,
                urb->context = uvc_urb;
                urb->pipe = usb_rcvisocpipe(stream->dev->udev,
                                            ep->desc.bEndpointAddress);
-#ifndef CONFIG_DMA_NONCOHERENT
+#if 0
                urb->transfer_flags = URB_ISO_ASAP | URB_NO_TRANSFER_DMA_MAP;
```

```
                urb->transfer_dma = uvc_urb->dma;
 #else
@@ -1834,7 +1834,7 @@ static int uvc_init_video_bulk(struct uvc_streaming
*stream,

                usb_fill_bulk_urb(urb, stream->dev->udev, pipe, uvc_urb->buffer,
                                  size, uvc_video_complete, uvc_urb);
-#ifndef CONFIG_DMA_NONCOHERENT
+#if 0
                urb->transfer_flags = URB_NO_TRANSFER_DMA_MAP;
                urb->transfer_dma = uvc_urb->dma;
 #endif
```

5. UVC 常见问题处理方法

5.1 UVC 预览打开失败问题

UVC 预览打开失败的问题，一般有如下五种情况：

- **案例分析1：** USB 高速握手失败

  绝大多数的 UVC 设备只能工作在高速（high-speed）或者超高速 （super-speed）模式下。如果因为信号问题，导致 UVC 降速到全速模式（full-speed），则 UVC 无法正常预览。

  可以通过串口查看 USB 的枚举信息，如果打印如下类似的 "full-speed" 的信息，说明高速握手失败。

  ```
  usb 3: new full-speed USB device number 3 using ohci-platform
  usb 3: not running at top speed; connect to a high speed hub
  ```

  处理方法： 先通过示波器分析高速握手信号，明确高速握手失败的原因，再通过优化硬件设计和软件 Tuning PHY 寄存器相结合的方式，解决高速握手失败的问题。

- **案例分析2：** UVC Probe Control 和 Commit Control 带宽协商出错

  UVC 在打开预览前，Host 和 Device 需要进行带宽的协商，正常协商的流程如下图 5 所示，包括四个阶段：

  1. Host 先将期望的设置发送给 USB 设备 (PROBE)；
  2. 设备将 Host 期望设置在自身能力范围之内进行修改，返回给 Host (PROBE)；
  3. Host 认为设置可行的话，Commit 提交 (COMMIT)；
  4. USB 设备根据 Host 的 Commit 信息设置格式、分辨率和带宽参数等；

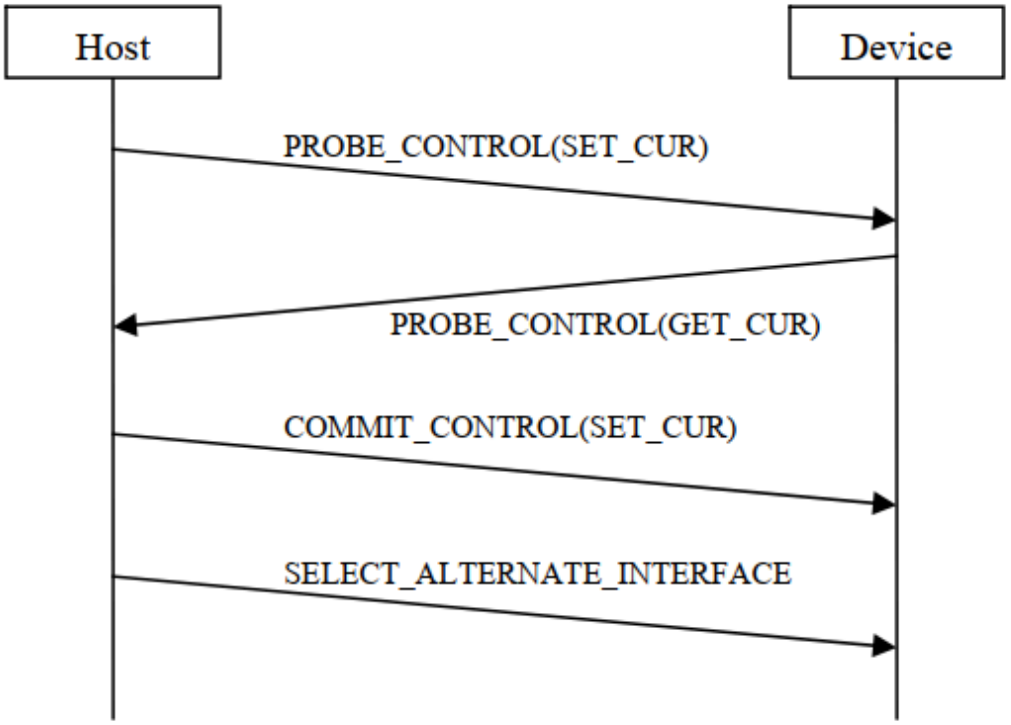

图 5 UVC 参数协商流程

处理方法： 使用 USB 协议分析仪或者 usbmon 工具监听 UVC 枚举以及打开预览的完整通信过程，通过分析仪解析的数据，排查 UVC 参数协商流程是否正确。如果明确是 UVC Probe Control 和 Commit Control 协商问题，则修改对应的 UVC 驱动逻辑或者应用逻辑。

- **案例分析3：** UVC Host 端异常

  根据 USB 2.0 协议规范，所有的 USB 事务传输都是由 USB Host 端发起，USB Device 不能主动向 Host 发送数据。因此，如果 Host 端工作异常，比如：在打开 UVC 预览操作时，没有执行 streaming on 的 SetInterface 命令，或者在执行 SetInterface 命令后，没有向 Device 的 Streaming 端点发送 IN packet 请求，那么 Device 就无法传输图像流。

  处理方法： 需要使用 USB 协议分析仪抓 USB 总线的完整事务传输，才能明确问题点。在无法使用 USB 协议分析仪的前提下，建议 Host 端参考章节 usbmon 抓包工具和分析工具，监听 Host 端的 USB 行为。

- **案例分析4：** UVC autosuspend 兼容性问题

  Rockchip 平台的 UVC Host 驱动默认使能 auto suspend 功能，目的是优化 UVC 应用场景的功耗。但有一部分 UVC 外设在 Rockchip 平台上的存在 auto suspend 兼容性问题，导致 UVC 预览异常。

  处理方法： 参考章节 关闭 UVC auto suspend 的方法 分析和解决这类问题。

- **案例分析5：** DDR 变频问题导致 UVC 取流异常

  Rockchip 平台大部分支持 DDR 动态变频功能，用于优化系统运行功耗。当 DDR 运行于较低频率时，会导致 USB 控制器的 bus latency 较高，当 bus latency 超过的 UVC 同步传输的容忍值，就可能会导致 UVC 打开预览异常。

  处理方法： 参考章节 设置 DDR 定高频的方法 分析和解决这类问题。

5.2 UVC 预览帧率不稳定

影响 UVC 预览帧率的因素比较多，比如：USB 控制器中断响应时间、USB 访问总线的 latency、UVC 驱动解析 UVC 帧的时间等。在系统处于高负载运行的场景下，UVC 帧率的不稳定性问题会更加明显。

处理方法： 参考章节 USB 中断绑核的方法、关闭 CPU IDLE 的方法、设置 CPU 定高频的方法、设置 DDR 定高频的方法，优化 USB 中断响应时间和 USB 控制器访问总线的 latency，提高 CPU 的性能。

5.3 UVC 预览花屏

UVC 预览花屏的主要原因是：大部分 UVC 外设传输图像时，采用的是 USB 同步传输方式，这种传输方式的特点是：实时性高但不可靠，如果 USB 总线数据传输出错或者丢失时，不会进行重传。在分析 UVC 花屏问题时，建议使用 USB 协议分析仪监听 USB 总线的数据传输，可以加快问题的定位。

UVC 传输的图像格式，包括：非压缩格式（YUV）和压缩格式（MJPG/H264/H256）。这两种图像格式，导致 UVC 花屏的问题点不同，下面针对这两种不同的图像格式，分别给出分析和处理问题的方法：

- **案例分析1：UVC 传输非压缩格式图像出现预览花屏**

  UVC 驱动中，会校验非压缩格式图像的一帧完整图像的数据大小，当检测到实际接收的图像大小与理论值不同，则 UVC 驱动会主动丢掉这张图像，避免因为 USB 物理总线数据传输错误或者 UVC 丢帧而导致的花屏现象。因此，可以先排除 USB 信号质量问题和 USB 控制器驱动未及时处理同步传输的问题。

  分析步骤：

  1. 使用 USB 协议分析仪持续捕获 UVC 预览非压缩格式图像的数据通信，同时观察 UVC 预览花屏现象。当出现花屏现象后，及时停止 USB 协议分析仪的捕获功能，保存协议分析仪数据，然后参考章节 解析 UVC 图像帧的方法 ，使用 Python 脚本自动解析和保存图像；
  2. 如果 USB 协议分析仪捕获的数据正常，通过分析仪数据解析的图像也正常，说明 UVC 预览花屏的问题点在 Host 端。否则，说明 UVC 外设传输的源数据已经异常，需要排查 Device 端；
  3. 如果定位到问题点在 Host 端，建议重点排查 USB 控制器 DMA buffer 的 cache 一致性；
  4. 还可以参考章节 v4l2-ctl 工具，使用 v4l2-ctl 工具取流，排查是否与 UVC 应用相关。

- **案例分析2：UVC 传输压缩格式图像出现预览花屏**

  UVC 传输压缩格式图像的场景，更容易出现花屏的现象，这是因为压缩格式的图像大小具有不确定性，所以，UVC 驱动无法校验实际接收的图像大小的正确性。

  分析步骤：

  1. 使用 USB 协议分析仪持续捕获 UVC 预览压缩格式图像的数据通信，同时观察 UVC 预览花屏现象。当出现花屏现象后，及时停止 USB 协议分析仪的捕获功能，保存协议分析仪数据，然后参考章节 解析 UVC 图像帧的方法 ，使用 Python 脚本自动解析和保存图像；
  2. 如果 USB 协议分析观察到明显的传输报错，如：CRC error，不完整的数据包等，则重点排查 USB 信号质量问题；
  3. 如果 USB 协议分析仪解析的图像，同样存在图像显示异常的现象，则说明 USB 物理总线传输的数据已经存在异常。进一步找到异常图像对应的 USB 分析仪数据，如果发现 Host 在连续多个微帧间隔内，都没有发出 IN packet 请求，说明可能与 Host 端 USB 控制器未及时处理同步传输有关系。如果 Host 传输行为正常，说明 Device 端传输的源数据可能存在问题，需要进一步排查 Device 端；

4. 如果 USB 协议分析仪解析的图像正常，则进一步排查 Host 端的 USB 控制器 DMA buffer 的 cache 一致性、USB/CPU 总线优先级（QOS）、UVC 应用处理等影响因素；
5. 可以参考章节 v4l2-ctl 工具，使用 v4l2-ctl 工具取流，排查是否与 UVC 应用相关。

处理方法：

1. USB 信号质量的问题：测试 USB HS 眼图，并根据眼图测试报告，优化硬件设计和修改软件驱动中 PHY 参数配置[4]；
2. USB 控制器未及时处理同步传输的问题：参考章节 USB 中断绑核的方法、关闭 CPU IDLE 的方法、设置 CPU 定高频的方法、设置 DDR 定高频的方法，优化 USB 中断响应时间和 USB 控制器访问总线的 latency，提高 CPU 的性能；
3. 提高 USB/CPU 的总线优先级（QOS）。

5.4 UVC 多路场景带宽不足问题

UVC 带宽不足问题，通常发生在如下应用场景：

USB Host 接口通过 HUB 同时连接 2 个及以上的 USB2.0 UVC Camera，打开预览失败，内核提示报错 log：

1. USB3.0 Host 报错 log

```
usb 5-1.4: Not enough bandwidth for new device state.
usb 5-1.4: Not enough bandwidth for altsetting 11
VIDIOC_STREAMON: failed: No space left on device
```

2. USB2.0 Host 报错 log

```
uvcvideo: Failed to submit URB 0 (-28).
VIDIOC_STREAMON: failed: No space left on device
```

- **案例分析：** RK3588 USB2.0 Host 通过 HUB 扩展 USB 口，HUB 下行端口同时连接两个 Logitech USB Camera：罗技高清网络摄像机 C93 和 UVC Camera (046d:0823)。

  问题复现方法：

```
# step1. 查看 UVC video 节点信息
console:/ # v4l2-ctl --list-devices
罗技高清网络摄像机 C93 (usb-fc880000.usb-1.1):
        /dev/video21
        /dev/video22

UVC Camera (046d:0823) (usb-fc880000.usb-1.4):
        /dev/video23
        /dev/video24

# step2. 两路 UVC 同时以分辨率 1920*1080 图像格式 MJPG 方式取流
console:/ # v4l2-ctl --device /dev/video21 --set-fmt-
video=width=1920,height=1080,pixelformat=MJPG --stream-mmap &
console:/ # v4l2-ctl --device /dev/video23 --set-fmt-
video=width=1920,height=1080,pixelformat=MJPG --stream-mmap &
```

```
# step3. video21 取流成功，但 video23 取流失败，串口提示log
console:/ # uvcvideo: Failed to submit URB 0 (-28).
console:/ # VIDIOC_STREAMON: failed: No space left on device
```

问题分析：

```
# 查看 UVC 设备的端点信息
console:/ # cat /sys/kernel/debug/usb/devices
# 对应 罗技高清网络摄像机 C93 设备
T:  Bus=02 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=480  MxCh= 0
D:  Ver= 2.00 Cls=ef(misc ) Sub=02 Prot=01 MxPS=64 #Cfgs=  1
P:  Vendor=046d ProdID=0891 Rev= 0.19
S:  Product=罗技高清网络摄像机 C930c
S:  SerialNumber=83FBB5EE
......
I:* If#= 1 Alt= 0 #EPs= 0 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
I:  If#= 1 Alt= 1 #EPs= 1 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
E:  Ad=81(I) Atr=05(Isoc) MxPS= 192 Ivl=125us
I:  If#= 1 Alt= 2 #EPs= 1 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
......
I:  If#= 1 Alt=10 #EPs= 1 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
E:  Ad=81(I) Atr=05(Isoc) MxPS=2688 Ivl=125us
I:  If#= 1 Alt=11 #EPs= 1 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
E:  Ad=81(I) Atr=05(Isoc) MxPS=3072 Ivl=125us

# 对应 UVC Camera (046d:0823) 设备
T:  Bus=02 Lev=02 Prnt=02 Port=03 Cnt=02 Dev#=  4 Spd=480  MxCh= 0
D:  Ver= 2.00 Cls=ef(misc ) Sub=02 Prot=01 MxPS=64 #Cfgs=  1
P:  Vendor=046d ProdID=0823 Rev= 0.10
S:  SerialNumber=7F65EA20
......
I:* If#= 3 Alt= 0 #EPs= 0 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
I:  If#= 3 Alt= 1 #EPs= 1 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
E:  Ad=81(I) Atr=05(Isoc) MxPS= 192 Ivl=125us
......
E:  Ad=81(I) Atr=05(Isoc) MxPS=1984 Ivl=125us
I:  If#= 3 Alt=10 #EPs= 1 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
E:  Ad=81(I) Atr=05(Isoc) MxPS=3060 Ivl=125us
I:  If#= 3 Alt=11 #EPs= 1 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
E:  Ad=81(I) Atr=05(Isoc) MxPS=3060 Ivl=125us
```

由上述端点信息，可以看出这两个 Logitech USB Camera 都支持动态带宽配置，最小的 Max packet size 均为 192 bytes，最大的 Max packet size 一个为 3072 bytes，另外一个为 3060 bytes。这里的 Max packet size 是指在一个 USB 微帧 内，USB Camera 传输的最大数据包大小。动态带宽配置是指 USB Camera 会根据图像分辨率和格式，动态调整 Max packet size。

参考章节 UVC 驱动调试接口，动态使能 UVC 驱动的 trace 信息，确定 UVC 打开预览时采用的带宽配置。

```
# 查看 UVC 设备占用的 USB 总线带宽信息
console:/ # echo 0xffff > /sys/module/uvcvideo/parameters/trace
console:/ # echo 8 > /proc/sys/kernel/printk
console:/ # v4l2-ctl --device /dev/video21 --set-fmt-
video=width=1920,height=1080,pixelformat=MJPG --stream-mmap

# 罗技高清网络摄像机 C93 设备的 trace 关键信息如下:
uvcvideo: Selecting alternate setting 11 (3072 B/frame bandwidth).

# 同样的操作方法，UVC Camera (046d:0823) 设备的 trace 关键信息如下:
uvcvideo: Selecting alternate setting 11 (3060 B/frame bandwidth).
```

由上述的 uvc trace 信息，可以看出以分辨率 1920*1080 图像格式 MJPG 方式取流时，罗技高清网络摄像机 C93 设备需要占用 3072 B/frame 的带宽，而UVC Camera (046d:0823) 设备也需要占用 3060 B/frame 的带宽。这里的 frame 对应 USB 协议定义的微帧（帧间隔 125 微秒）。

根据 USB2.0 协议规范[5]的说明：

章节 5.6.4 Isochronous Transfer Bus Access Constraints 描述：

High-speed endpoints can allocate at most **80%** of a microframe for periodic transfers.

说明周期传输最大占用的微帧时间 = 125 us * 80% = **100us**，剩余时间预留给非周期传输类型的设备。

章节 5.11.3 Calculating Bus Transaction Times 描述：

```
High-speed (Input)
Isochronous Transfer (No Handshake)
= (38 * 8 * 2.083) + (2.083 * Floor(3.167 + BitStuffTime(Data_bc))) +
Host_Delay
```

Linux 内核 USB 驱动严格按照 USB 2.0 规范的要求进行软件设计，对应的代码实现如下：

```
include/linux/usb/hcd.h
/*
 * Ceiling [nano/micro]seconds (typical) for that many bytes at high speed
 * ISO is a bit less, no ACK ... from USB 2.0 spec, 5.11.3 (and needed
 * to preallocate bandwidth)
 */
#define USB2_HOST_DELAY 5       /* nsec, guess */
#define HS_NSECS(bytes) (((55 * 8 * 2083) \
        + (2083UL * (3 + BitTime(bytes))))/1000 \
        + USB2_HOST_DELAY)
#define HS_NSECS_ISO(bytes) (((38 * 8 * 2083) \
        + (2083UL * (3 + BitTime(bytes))))/1000 \
        + USB2_HOST_DELAY)
#define HS_USECS(bytes)         NS_TO_US(HS_NSECS(bytes))
#define HS_USECS_ISO(bytes)     NS_TO_US(HS_NSECS_ISO(bytes))

drivers/usb/core/hcd.c
long usb_calc_bus_time (int speed, int is_input, int isoc, int bytecount)
{
        unsigned long   tmp;
        ......
        case USB_SPEED_HIGH:    /* ISOC or INTR */
```

```
                /* FIXME adjust for input vs output */
                if (isoc)
                        tmp = HS_NSECS_ISO (bytecount);
                else
                        tmp = HS_NSECS (bytecount);
        ......
}
```

按照上述计算方法，不同 ISOC Max packet size 对应的 bus time 如下：

Max packet size (bytes/uframe)	bus time(us)
256	5
640	13
1024	20
1920	37
3072	60

因此，当两个 UVC 设备同时以高带宽的方式（如：3072bytes/uframe）传输图像流时，占用的总线时间将超过 USB2.0 协议规定的最大 100us 的限制，导致 UVC 带宽不足问题。

处理方法：

方法1. 优化 UVC 驱动的带宽分配策略（推荐优先使用）

作用：减小特定 UVC 设备的最大带宽请求

影响：可能降低非压缩格式（YUV）的帧率

Linux-5.10 参考修改如下：

```
diff --git a/drivers/media/usb/uvc/uvc_video.c
b/drivers/media/usb/uvc/uvc_video.c
index f6373d678d25..ee798b778255 100644
--- a/drivers/media/usb/uvc/uvc_video.c
+++ b/drivers/media/usb/uvc/uvc_video.c
@@ -1845,6 +1845,7 @@ static int uvc_video_start_transfer(struct
uvc_streaming *stream,
        struct usb_interface *intf = stream->intf;
        struct usb_host_endpoint *ep;
        struct uvc_urb *uvc_urb;
+       struct usb_device *udev = interface_to_usbdev(intf);
        unsigned int i;
        int ret;

@@ -1866,6 +1867,13 @@ static int uvc_video_start_transfer(struct
uvc_streaming *stream,
                /* Isochronous endpoint, select the alternate setting. */
                bandwidth = stream->ctrl.dwMaxPayloadTransferSize;

+               if ((le16_to_cpu(udev->descriptor.idVendor) == 0x2bdf) &&
+                   (le16_to_cpu(udev->descriptor.idProduct) == 0x0293) &&
+                   (bandwidth > 1600)){
```

```
+                       printk("UVC_DBG: limit bandwidth from %u to 1600B\n",
bandwidth);
+                       bandwidth = 1600;
+               }
+
                if (bandwidth == 0) {
                        uvc_trace(UVC_TRACE_VIDEO, "Device requested null "
                                "bandwidth, defaulting to lowest.\n");
```

方法2. 提高 USB 总线的周期传输带宽占比

作用：将 USB 总线预留给周期传输的带宽占比提高到 95% 甚至更高，以提高周期传输的传输带宽。

影响：在同一条 USB 总线上的其它非周期传输设备（如：U 盘），可能无法正常工作。

限制：USB3.0 Host xHCI 控制器不适用于这种方法。

USB2.0 Host EHCI 控制器驱动参考修改如下：

```
diff --git a/drivers/usb/host/ehci-hcd.c b/drivers/usb/host/ehci-hcd.c
index 8aff19ff8e8f..bed879a4ab5b 100644
--- a/drivers/usb/host/ehci-hcd.c
+++ b/drivers/usb/host/ehci-hcd.c
@@ -475,7 +475,7 @@ static int ehci_init(struct usb_hcd *hcd)
         * by default set standard 80% (== 100 usec/uframe) max periodic
         * bandwidth as required by USB 2.0
         */
-       ehci->uframe_periodic_max = 100;
+       ehci->uframe_periodic_max = 125;

        /*
         * hw default: 1K periodic list heads, one per frame.
```

USB2.0 Host DWC2 控制器参考修改如下：

```
diff --git a/drivers/usb/dwc2/hcd_queue.c b/drivers/usb/dwc2/hcd_queue.c
index b2e0721a3eb8..e8b394bd9aae 100644
--- a/drivers/usb/dwc2/hcd_queue.c
+++ b/drivers/usb/dwc2/hcd_queue.c
@@ -119,7 +119,7 @@ static int dwc2_check_periodic_bandwidth(struct
dwc2_hsotg *hsotg,
                 * High speed mode
                 * Max periodic usecs is 80% x 125 usec = 100 usec
                 */
-               max_claimed_usecs = 100 - qh->host_us;
+               max_claimed_usecs = 125 - qh->host_us;
        } else {
                /*
                 * Full speed mode
```

5.5 UVC 预览延时问题

UVC 图像的传输链路很长，在传输链路中，每个模块的延时，都可能会对预览总延时造成影响。因此，在分析 UVC 预览延时问题时，需要先使用 USB 协议分析仪抓预览启动阶段的完整数据流，明确延时瓶颈点在 Device 端或者 Host端。如果是 Host 端的问题，再对 Host 端传输 UVC 图像流的传输链路中各个模块的延时进行独立分解。

- **案例分析：** Host 端缓存图像导致结构光模组 MJPEG 1080P 时延问题

  原因是 Host 端 APK libuvc 接收到内核的图像数据后，会做 memcopy 的操作，放到应用自己管理的 frame queue 中，该 frame queue 和内核的 urb（10 个 urb，用于轮转接收图像数据）是异步。由于Host 端的 1080P 软解和显示比较耗时，当内核已经收到 6~7 帧图像，并且 APK libuvc 已经全部收到并放到 frame queue 后，才开始显示第一帧图像。Host 端的 frame queue 缓存了几帧图像，导致时延问题。

  处理方法： 提高 Host 端图像软解的效率，避免图像缓存。

5.6 UVC EMI 问题

- **案例分析：** EMI 问题通常与 USB 设备的硬件电路设计有关系，当发生 EMI 问题时，会影响到 UVC 的正常通信，比如：UVC 无法打开预览或者预览图像卡住等现象。

  如果 USB Host 端使用的是 Linux 内核，当发生 EMI 问题时，Host 端内核一般会打印如下 log：

  ```
  usb usb1-port1: disabled by hub (EMI?), re-enabling...
  ```

  需要说明的是，Linux USB Host 的软件驱动检测 EMI 的原理是：USB Host 控制器（如 EHCI 控制器）会不断轮询寄存器的 port enable 状态，当软件检测到 USB 控制器的 port 状态突然从 enable 变成 disable，并且寄存器的外设连接状态仍然显示为连接，则软件端就会认为可能是 EMI 错误导致控制器状态异常，并打印上述 EMI log。因此，软件端提示 EMI 错误，只能作为排查问题的一个重要方向，但并不能明确一定是由 EMI 问题导致。

  处理方法：

  1. 使用示波器测试 USB 眼图，并生成测试报告给硬件工程师和 USB 驱动软件工程师。工程师根据测试报告，修改 USB 硬件电路和调整 USB PHY 寄存器来提高 USB 信号质量，从而改善 EMI 问题。
  2. 使用示波器测试 DP/DM 单端信号和差分信号，如果噪声偏大，会影响 USB DP/DM 信号质量，触发 EMI 问题。可以通过如下三种硬件优化方法，改善噪声问题：

     (1) DP/DM 串联共模电感；

     (2) 增加 Device 和 Host 的共地连接；

     (3) 更换 DP/DM Pin 接触面积更大的 USB Host 座子；

6. 参考文献

1. kernel/Documentation/trace/ftrace.rst
2. USB Video Class v1.1
3. 《Rockchip_Developer_Guide_USB_CN》
4. 《Rockchip_Introduction_USB_SQ_Tool_CN》

5. USB 2.0 Specification