



HiMPP 媒体处理软件 V4.0

FAQ

文档版本 14

发布日期 2020-11-15

版权所有 © 上海海思技术有限公司 2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



、**HISILICON**、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司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的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编： 518129

网址： <http://www.hisilicon.com/cn/>

客户服务邮箱： support@hisilicon.com



前言

概述

本文为使用 HiMPP 媒体处理软件 V4.0 开发的程序员而写，目的是为您在开发过程中遇到的问题提供解决办法和帮助。



说明

- 未有特殊说明，Hi3569V100/Hi3559CV100 与 Hi3559AV100 内容一致。
- 未有特殊说明，Hi3568V100/Hi3556AV100 与 Hi3519AV100 内容一致。
- 未有特殊说明，
Hi3566V100/Hi3562V100/Hi3516AV300/Hi3516DV300/Hi3559V200/Hi3556V200 与
Hi3516CV500 内容一致。
- 未有特殊说明，Hi3516EV200、Hi3516EV300、Hi3516DV200、Hi3518EV300 内容一致。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本	操作系统
Hi3559A	V100ES	Linux Linux+Huawei LiteOS
Hi3559A	V100	Linux Linux+Huawei LiteOS
Hi3559C	V100	Linux+Huawei LiteOS
Hi3519A	V100	Linux Linux+ Huawei LiteOS
Hi3556A	V100	Linux+Huawei LiteOS



产品名称	产品版本	操作系统
Hi3516C	V500	Linux Huawei LiteOS
Hi3516D	V300	Linux Huawei LiteOS
Hi3516A	V300	Linux Huawei LiteOS
Hi3559	V200	Linux Huawei LiteOS Linux+Huawei LiteOS
Hi3556	V200	Linux Huawei LiteOS Linux+Huawei LiteOS
Hi3516E	V200	Linux/Huawei LiteOS
Hi3516E	V300	Linux/Huawei LiteOS
Hi3518E	V300	Linux/Huawei LiteOS
Hi3516D	V200	Linux
Hi3562	V100	Linux+Huawei LiteOS
Hi3566	V100	Linux+Huawei LiteOS
Hi3568	V100	Linux+Huawei LiteOS
Hi3569	V100	Linux+Huawei LiteOS

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



符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

寄存器访问类型约定

类型	说明	类型	说明
RO	只读，不可写。	RW	可读可写。
RC	读清零。	WC	可读，写 1 清零，写 0 保持不变。

寄存器复位值约定

在寄存器定义表格中：

- 如果某一个比特的复位值 “Reset”（即 “Reset” 行）为 “? ”，表示复位值不确定。
- 如果某一个或者多个比特的复位值 “Reset” 为 “? ”，则整个寄存器的复位值 “Total Reset Value” 为 “-”，表示复位值不确定。



数值单位约定

数据容量、频率、数据速率等的表达方式说明如下。

类别	符号	对应的数值
数据容量 (如 RAM 容量)	1K	1024
	1M	1,048,576
	1G	1,073,741,824
频率、数据速率等	1k	1000
	1M	1,000,000
	1G	1,000,000,000

地址、数据的表达方式说明如下。

符号	举例	说明
0x	0xFE04、0x18	用 16 进制表示的数据值、地址值。
0b	0b000、0b00 00000000	表示 2 进制的数据值以及 2 进制序列 (寄存器描述中除外)。
X	00X、1XX	在数据的表达方式中，X 表示 0 或 1。 例如：00X 表示 000 或 001； 1XX 表示 100、101、110 或 111。

其他约定

本文档中所用的频率均遵守 SDH 规范。简称和对应的标称频率如下。

频率简称	对应的标称频率
19M	19.44MHz
38M	38.88MHz
77M	77.76MHz



频率简称	对应的标称频率
622M	622.08MHz

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 14 (2020-11-15)

新增 1.3.4 小节；
3.3.5.1 小节涉及更新；
新增 6.7, 6.8, 6.9 和 10.3 小节。

文档版本 13 (2020-03-31)

1.2.2 和 3.3 小节涉及修改
添加 3.3.6 和 10.2 小节

文档版本 12 (2020-01-20)

1.2.2 小节的 VI/VPSS 模块配置涉及修改

文档版本 11 (2019-10-25)

1.2.3、3.4.4 和 3.4.5 小节涉及修改

文档版本 10 (2019-09-15)

新增 3.4.4、3.4.5 小节
3.3.1.2 小节涉及修改

文档版本 09 (2019-07-30)

1.2.2 小节涉及修改，新增“REGION 模块配置”
新增 1.2.3 “公共 VB 分配指导” 小节和 1.6 “修改内核选项后重编 KO 流程” 小节。



3.3 和 3.4 小节涉及修改

文档版本 08 (2019-06-20)

新增 1.2.2、1.2.3 小节

文档版本 07 (2019-05-30)

1.2.2 小节，ISP 模块配置涉及更新

新增 3.3.5 小节

文档版本 06 (2019-05-15)

新增 3.6、6.6 小节，2.2 小节涉及修改

文档版本 05 (2019-03-15)

1.2.2、3.4.2、3.4.3 小节涉及修改

文档版本 04 (2019-02-28)

1.2.2 和 3.4.2 小节涉及修改

新增 3.5、6.5、9.1.2 小节

文档版本 03 (2019-01-30)

增加 1.2.1 小节。

文档版本 02 (2019-01-05)

第 2 次正式版本发布。

3.4.2 小节涉及修改

增加 12.1.5 小节。

文档版本 01 (2018-12-12)

第 1 次正式版本发布。



目 录

前 言	i
1 系统控制	1
1.1 日志信息	1
1.1.1 如何查看 MPP 的日志信息.....	1
1.2 内存使用	2
1.2.1 OS 保留内存和线程栈大小调整.....	2
1.2.2 如何根据具体产品调整媒体业务所占内存	3
1.2.3 公共 VB 分配指导	15
1.2.4 CMA 相关.....	19
1.3 性能相关	19
1.3.1 CPU 性能 Top 统计波动大问题.....	19
1.3.2 绑定中断到不同 CPU 的注意事项	20
1.3.3 解码性能.....	20
1.3.4 编码性能.....	20
1.4 小型化.....	21
1.4.1 静态库使用	21
1.5 管脚复用、时钟门控、系统控制在哪里配置？	21
1.6 修改内核选项后重编 KO 流程	21
2 MIPI 配置	23
2.1 MIPI 频率说明	23
2.2 时序	23
3 VI 功能	25
3.1 DIS 功能	25



3.1.1 如何实现 ISP-DIS 场景功能.....	25
3.2 WDR 功能.....	25
3.2.1 QudraWDR 注意事项.....	25
3.3 VI 时序配置.....	25
3.3.1 BT.1120	26
3.3.2 BT.656.....	29
3.3.3 BT.601.....	31
3.3.4 MIPI_YUV.....	34
3.3.5 LVDS.....	37
3.3.6 DC	42
3.4 VI 中断类型.....	44
3.4.1 FRAME_INTERRUPT_START 类型	44
3.4.2 FRAME_INTERRUPT_EARLY 类型	45
3.4.3 FRAME_INTERRUPT_EARLY_END 类型	45
3.4.4 FRAME_INTERRUPT_EARLY_END_ONE_BUF 类型	46
3.4.5 FRAME_INTERRUPT_EARLY_ONE_BUF 类型	47
3.5 P/N 制切换方法	48
3.6 Hi3519AV100 对接 6 路 YUV 配置	49
3.6.1 1 个 MIPI AD 4 路复合 + 2 个 BT656 单路	49
3.6.2 1 个 MIPI AD 4 路复合 + 2 个 MIPI AD 单路	52
4 FISHEYE 功能	55
4.1 鱼眼矫正功能	55
4.1.1 Hi3559AV100/Hi3556AV100 如何实现多于 4 宫格鱼眼矫正功能.....	55
5 LDC 功能	56
5.1 畸变矫正功能	56
5.1.1 VI 和 VPSS LDC 对比	56
6 音频.....	57
6.1 PC 如何播放由 Hisilicon (Shanghai) 编码的音频码流	57
6.1.1 PC 如何播放由 Hisilicon (Shanghai) 编码的音频 G711/G726/ADPCM 码流	57
6.2 Hisilicon (Shanghai) 如何播放标准的音频码流	59
6.2.1 Hisilicon (Shanghai) 如何播放标准的音频 G711/G726/ADPCM 码流	59



6.3 为什么使能 VQE 后会有高频部分缺失.....	61
6.3.1 为什么使能 VQE 后会有高频部分缺失	61
6.4 音频内置 CODEC 输出(AO 输出)出现幅频响应异常	63
6.5 静态库注册功能	63
6.6 加载内置 CODEC 模块出现 pop 音的解决方法	64
6.7 如何对多声道的音频数据进行交织处理	64
6.8 如何对多声道的音频数据进行混音处理	65
6.9 Musl 版本的 sample 链接 bcd 库后执行时会出现段错误异常	66
7 低功耗.....	67
7.1 低功耗模块动态调频可能频繁调频的问题.....	67
8 LCD 屏幕调试	68
8.1 支持哪些 LCD 屏幕	68
8.2 LCD 屏幕调试顺序	68
8.2.1 确认管脚复用配置	68
8.2.2 确认用户时序	68
8.2.3 配置设备帧率	70
8.2.4 确认时钟信息	70
9 VO	72
9.1 VO 用户时序如何配置	72
9.1.1 时序结构配置	72
9.1.2 时钟大小配置	75
9.2 画面切换	75
9.2.1 通道属性发生变化	75
9.2.2 建议的实现方式	75
9.3 视频同步方案	77
9.3.1 实现原理	77
9.3.2 建议的操作步骤	78
10 VENC	79
10.1 JPEG 量化表配置注意事项	79
10.2 JPEG 使能从多个源接收图像注意事项	79



10.3 低功耗使能注意事项	79
10.4 自编自解规格说明	80
11 HDMI	81
12 其它	82
12.1 动态库	82
12.1.1 为什么使用静态编译方式编译应用程序无法使用动态库	82
12.1.2 为什么使用 libupvqe.a 和 libdnvqe.a 动态编译时出现重定义	83
12.1.3 模块 KO 之间的依赖关系	84
12.1.4 SPI 驱动说明	84
12.1.5 Huawei LiteOS 系统下动态链接库的动态加载实现方案	85



1

系统控制

1.1 日志信息

1.1.1 如何查看 MPP 的日志信息

【现象】

需要查看日志和调整 log 日志的等级。

【分析】

Log 日志记录 SDK 运行时错误的原因、大致位置以及一些系统运行状态等信息。因此可通过查看 log 日志，辅助错误定位。

目前日志分为 7 个等级，默认设置为等级 3。等级设置的越高，表示记录到日志中的信息量就越多，当等级为 7 时，系统的整个运行状态实时的被记录到日志中，此时的信息量非常庞大，会大大降低系统的整体性能。因此，通常情况下，推荐设置为等级 3，因为此时只有发生错误的情况下，才会将信息记录到日志中，辅助定位绝大多数的错误。

【解决】

获取日志记录或修改日志等级时用到的命令如下：

- 查看各模块的日志等级，可以使用命令 `cat /proc/umap/logmpp`，此命令会列出所有模块日志等级。
- 修改某个模块的日志等级，可使用命令 `echo "venc=4" > /proc/umap/logmpp`，其中 venc 是模块名，与 cat 命令列出的模块名一致即可。



- 修改所有模块的日志等级，可以使用命令 `echo "all=4" > /proc/umap/logmpp.`
- 获取日志记录，可以使用命令 `cat /dev/logmpp`，此命令将打印出所有的日志信息；如果日志已读空，命令会阻塞并等待新的日志信息，可以使用 `Ctrl+C` 退出。如果不希望阻塞等待日志信息，可以使用命令 `echo wait=0 > /proc/umap/logmpp` 取消阻塞等待。也可以使用 `open`、`read` 等系统调用来操作/`dev/logmpp` 这个设备节点。

1.2 内存使用

1.2.1 OS 保留内存和线程栈大小调整

【现象】Linux 系统跑业务程序出现 oom-killer。

【分析】可能的原因有

- OS 内存不足
- 系统保留内存太小

【解决】

- 增大 OS 内存
- 增大系统保留内存，可以在/etc/profile 中加入以下命令将系统保留内存设置为 4M (大小可调整)

```
echo 2 >/proc/sys/kernel/randomize_va_space  
echo 4096 >/proc/sys/vm/min_free_kbytes
```

【现象】

系统跑业务程序出现创建线程失败打印 `pthread_create: Resource temporarily unavailable` 的错误信息。

【分析】可能的原因有

- OS 内存不足
- 线程栈空间太大

【解决】

- 增大 OS 内存



- 调整线程最大栈空间大小，调整方式有 2 种：
 - Linux 系统可以使用 ulimit -s 命令修改线程栈大小，例如将线程栈大小设置成 1M：ulimit -s 1024，可以在/etc/profile 中加入此命令则可以开机就设置栈空间大小。
 - 使用 pthread_attr_setstacksize 在程序中改变线程栈大小。

1.2.2 如何根据具体产品调整媒体业务所占内存

【现象】

媒体业务需要占用一定的内存（主要占用 MMZ 内存）以支持业务正常运转，HiMPP V4.0 平台按典型业务形态分配内存。用户产品内存使用紧张时，可根据实际情况尝试采用相关的策略调整内存分配大小。

【分析】

针对内存使用紧张的产品，上海海思交付包中的 SDK 软件提供了一些方法对内存的分配做调整。这里只简单描述精简内存措施，具体措施的使用方法请参考相关文档。

【解决】

1. 确认 OS 及 MMZ 内存分配情况。

详见上海海思发布包中的文件《Hi35xx SDK 安装以及升级使用说明》中的“地址空间分配与使用”章节。

2. 根据实际使用情况调整 SDK 相关业务内存占用：

- 整系统：

产品应保证所有分辨率图像的大小应成整数倍的关系，如 1080P 为 1920x1080，960H 为 960x480，而不应出现 960H 为 960x756 的类似情况；同时，也不应出现 VI 采集 1920x1088 大小的图像，而 VENC 编码为 1920x1080 的情况。

- 每个模块的 buffer 配置最小值。

参考文档：《HiMPP 媒体处理软件 V4.0 开发参考》

- 公共 VB 刚好分配足够。

相关接口：HI_MPI_VB_SetConfig。

参考《HiMPP 媒体处理软件 V4.0 开发参考》中的“系统控制”章节。

特别提醒各个模块输出数据使用的 VB 大小计算较为复杂，具体计算公式参考代码 hi_buffer.h。



- 根据业务合理选择 VI VPSS 的在线离线模式，不同模式下内存占用从小到大致顺序为(VI_ONLINE_VPSS_ONLINE/VI_PARALLEL_VPSS_PARALLEL) < (VI_ONLINE_VPSS_OFFLINE/VI_PARALLEL_VPSS_OFFLINE) < VI_OFFLINE_VPSS_ONLINE < VI_OFFLINE_VPSS_OFFLINE。
- VI 模块配置：

参考文档：《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频输入”章节。

proc 信息查看命令：cat /proc/umap/vi

措施	相关模块参数/接口	收益	影响	注意	proc 信息
Raw 压缩	HI_MPI_VI_CreatePipe : enCompressMode	比不压缩节省内存和带宽	-	Hi3516EV200 不支持	VI PIPE ATTR1: CompressMode
在线 WDR 行模式卷绕	HI_MPI_VI_SetDevAttr : stWDRAttr	卷绕模式不需要分配一帧 buffer	配置不合理可能会出现图像效果异常	1.与 sensor 时序强相关 2. Hi3559AV100 不支持	VI DEV ATTR2:WDR Mode CacheLine
3DNR 压缩	HI_MPI_VI_CreatePipe : bNrEn 和 stNrAttr	比不压缩节省内存和带宽	-	仅 Hi3559AV100/ Hi3559CV100/ Hi3569V100 支持	VI PIPE NR ATTR: CompressMode
Early_End 机制	HI_MPI_VI_SetPipeFrameInterruptAttr	调整得当可节省 1~2 帧 buffer	调整不当可能出现图像效果异常	与 sensor 时序强相关	VI PIPE ATTR2: IntType EarlyLine
YUV 压缩	HI_MPI_VI_SetChnAttr	比不压缩节省内存和带宽	-	• Hi3519AV100 通道 1 不支持 • Hi3516EV200 不支持	VI CHN ATTR2: CompressMode

- VPSS 模块配置：

参考文档：《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频处理子系统”章节。



proc 信息查看命令: cat /proc/umap/vpss

措施	相关模块参数/接口	收益	影响	注意	proc 信息
3DNR 压缩	HI_MPI_VPSS_Creat eGrp: stNrAttr	比不压缩节省内存和带宽	-	Hi3559AV100、 Hi3559CV100/Hi356 9V100 不支持	VPSS GRP ATTR: RefCmp
3DNR 参考帧重构帧 buffer 复用	自适应复用	可节省一帧 buffer	-	1. Hi3519AV100 以下 条件不支持复用: 3DNR 压缩或 enNrMotionMode 设置为 NR_MOTION_MODE _COMPENSATE 2. Hi3516CV500 以下 条件不支持复用: 3DNR 不压缩且 enNrMotionMode 设置为 NR_MOTION_MODE _COMPENSATE	-
VPSS-VENC 低延时单 buffer 模式	低延时: HI_MPI_VPSS_SetLo wDelayAttr 单 buffer 模式: HI_MPI_VPSS_SetM odParam: bOneBufForLowDel ay	可节省一帧 buffer	行号设置不合理可能出 现图像效果异常	-	VPSS CHN LOWDELAY ATTR: Enable LineCnt OneBufEnabl e
VPSS-VENC 低延时卷绕	HI_MPI_VPSS_SetCh nBufWrapAttr HI_MPI_SYS_GetVPS SVENCWrapBufferLi ne	比低延时单 buffer 模式节省更多内存	设置不合理可能出 现图像效果异常	1. 与 sensor 时序强相 关 2. 仅 Hi3516EV200/Hi3516 EV300/Hi3518EV300 支持	VPSS CHN BUF WRAP ATTR: Enable BufLine WrapBufSize
VPSS 分块时右块	低延时: HI_MPI_VPSS_SetLo wDelayAttr	可节省一帧 buffer	行号设置不合理可能出 现图像效果	VPSS、VENC 串行处 理完同一帧的时间小 于一个采集时序的时	VPSS CHN LOWDELAY ATTR:



措施	相关模块参数/接口	收益	影响	注意	proc 信息
低延时			异常	间才能节省 1 帧 buffer	Enable LineCnt OneBufEnable
输入输出 buffer 复用	HI_MPI_VPSS_EnableBufferShare HI_MPI_VPSS_DisableBufferShare 需满足一定条件才支持复用，约束条件参考《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频处理子系统”章节 HI_MPI_VPSS_EnableBufferShare 接口描述部分	可节省一帧 buffer	-	仅 Hi3516EV200/Hi3516EV300/Hi3518EV300/Hi3516CV500/Hi3516DV300/Hi3556V200/Hi3559V200/Hi3566V100/Hi3562V100 支持	VPSS CHN ATTR: bBufferShare
Early_End 机制	HI_MPI_VPSS_SetGroupFrameInterruptAttr	调整得当可节省一帧 buffer	调整不当可能出现图像效果异常	1. 与 sensor 时序强相关 2. 仅 Hi3516EV200/Hi3516EV300/Hi3518EV300 支持。全在线 CH0 使用卷绕时，还可以设置调整 Early End 的行号（建议从大往小调）以达到小码流节省 1 个 VB。	FRAME INTERRUPT ATTR: IntType EarlyLine
按场景调整分块节点数量	HI_MPI_VPSS_SetModParam: u32VpssSplitNodeNum	可以节省模板内存	-	仅 Hi3559AV100、Hi3559CV100/Hi3569V100 支持	MODULE PARAM: u32VpssSplitNodeNum
按场景选择是否开启 HDR 功能	HI_MPI_VPSS_SetModParam: bHdrSupport	节省 HDR buffer 内存	-	仅 Hi3559AV100、Hi3559CV100/Hi3569V100 支持	MODULE PARAM: bHdrSupport



措施	相关模块参数/接口	收益	影响	注意	proc 信息
关闭 backup 帧	HI_MPI_VPSS_EnableBackupFrame HI_MPI_VPSS_DisableBackupFrame	每个 VPSS GROUP 少占用 1 帧输入源的 buffer。	VO 暂停情况下切换画面时显示设备背景色。	-	-
CH0 输出 YUV 压缩	HI_MPI_VPSS_SetChnAttr: enCompressMode	比不压缩节省内存和带宽	-	<ul style="list-style-type: none">仅 Hi3516EV200/Hi3516EV300/Hi3518EV300 支持YUV 压缩只能对接 H264/H265 编码	VPSS CHN OUTPUT RESOLUTION : Compress

- VENC 模块配置：

参考文档：《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频编码”章节。

proc 信息查看命令：cat /proc/umap/venc, cat /proc/umap/h265e, cat /proc/umap/h264e, cat /proc/umap/jpege

措施	相关模块参数/接口	收益	影响	注意	proc 信息
动态切换编码分辨率	HI_MPI_VENC_GetChnAttr HI_MPI_VENC_SetChnAttr	切换编码分辨率时不销毁通道，减少内存碎片。	无	切换分辨率后所有参数恢复默认值。	-
编码码流 buffer 使用省内存模式分配	HI_MPI_VENC_SetModParam: u32H264eMiniBufMode u32H265eMiniBufMode u32JpegeMiniBufMode	可以把码流 buffer 设置小一些。	此模式需要用大设置合理，否则会出现因码流 buffer 不足而不断重编或者丢帧的情况。	-	venc/h265e/h264e/jpege 模块都有 MODULE PARAM: MiniBufMode



措施	相关模块参数/接口	收益	影响	注意	proc 信息
参考帧重构 帧 buffer 复用	HI_MPI_VENC_CreateChn : bRcnRefShareBuf	大约可节省 (RefNum+1 - 1.3*RefNum) 帧 buffer	超大帧、码率 过冲、码率 buffer 满等异 常情况导致的 丢帧或者重 编，下一帧只 能插入 1 帧	Hi3559A V100、 Hi3519A V100 不 支持	h265e/h26 4e 模块的 RefParam INFO: RcnRefSha reBuf
动态回收参 考帧 buffer	HI_MPI_VENC_SetModPar am: u32FrameBufRecycle	当编码切换 GOP 模式， 参考帧由多 变少时，可 动态释放多 出来的参考 帧 buffer	-	-	venc 模块 的 MODULE PARAM: FrameBufR ecycle
编码模块支 持的最大通 道数	Linux: VencMaxChnNum Huawei LiteOS: VENC_MODULE_PARAMS _S 中的 u32VencMaxChnNum	可节省部分 OS 内存	-	-	venc 模块 的 MODULE PARAM: VencMaxC hnNum
相同分辨率 多通道编码 使用 UserVB 模 式节省内存	HI_MPI_VENC_SetModPar am: enH264eVBSource enH265eVBSource	比 PrivateVB 模式节省更 多帧存 buffer	-	-	h265e 模块 的 MODULE PARAM: H265eVBS ource h264e 模块 的 MODULE PARAM: H264eVBS ource

- VDEC 模块配置：



参考文档：《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频解码”章节。

proc 信息查看命令：cat /proc/umap/vdec。

措施	相关模块参数/接口	收益	影响	注意	proc 信息
解码码流 buffer 使用省内存模式分配	HI_MPI_VDEC_SetModParam: u32MiniBufMode	可以把码流 buffer 设置小一些。	-	-	MODULE PARAM: MiniBufMode
按场景设置解码器最大能力集	HI_MPI_VDEC_SetModParam: stVideoModParam 和 stPictureModParam	可节省部分 MMZ 内存	VDH 相关的内存分配的内存少了可能会导致解码性能下降，但是不影响功能。	VEDU 解码不支持 VDH 相关配置	MODULE PARAM: MaxPicWidth MaxPicHeight MaxSliceNum VdhMsgNum VdhBinSize VdhExtMemLevel 和 MaxJpegeWidth MaxJpegeHeight SupportProgressive DynamicAllocate CapStrategy
按场景设置解码通道能力集	HI_MPI_VDEC_SetProtocolParam	可节省部分 OS 内存	-	-	CHN VIDEO ATTR & PARAMS: MaxVPS MaxSPS MaxPPS MaxSlice
H264 解码无 B 帧码流时可关闭 Tmv 开关	HI_MPI_VDEC_CreateChn : bTemporalMvpEnable	可节省 Tmv buffer	-	-	CHN VIDEO ATTR & PARAMS: TemporalMvp
解码模块最大支持的通道数	Linux: VdecMaxChnNum VfmwMaxChnNum Huawei LiteOS: VDEC_MODULE_PARAM	可节省部分 OS 内存	-	-	MODULE PARAM: VdecMaxChnNum



措施	相关模块参数/接口	收益	影响	注意	proc 信息
	S_S VFMW_MODULE PARA MS_S				
只解码 I 帧的通道把参考帧设置为 0	HI_MPI_VDEC_CreateChn : u32RefFrameNum	不需要分配参考帧 buffer	-	把通道解码模式设置为 I 模式, 否则 logmpp 会报错。	CHN VIDEO ATTR & PARAMS: RefNum

- AVS 模块配置:

参考文档:《HiMPP 媒体处理软件 V4.0 开发参考》中的“拼接”章节。

proc 信息查看命令: cat /proc/umap/avs。

措施	相关模块参数/接口	收益	影响	注意	proc 信息
按场景合理设置 WorkingSet 大小	HI_MPI_AVG_SetModPara m: u32WorkingsetSize	可节省部分 MMZ 内存	设置小了对图像效果有影响	-	cat /proc/uma p/avs AVS WORKING SET: WorkingSe tSize
YUV 压缩	HI_MPI_AVG_GetChnAttr	比不压缩节省内存和带宽	-	-	AVS CHN ATTR: Compress

- VGS 模块配置:

- 参考文档:《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频图形处理子系统”章节。

proc 信息查看命令: cat /proc/umap/vgs。

- 注意: 以下模块参数均是 Linux 配置, 对应的 Huawei LiteOS 模块参数结构体是 VGS_MODULE_PARAMS_S, 在 sdk_init.c 文件中的 VGS_init 函数配置



措施	相关模块参数	收益	影响	注意	proc 信息
设置 VGS 支持的最大 job 数	max_vgs_job	默认值 128, 按需减小可节省内存	job 数量过少会限制 VGS 性能。	-	MODULE PARAM: max_job_num
设置 VGS 支持的最大 task 数	max_vgs_task	默认值 200, 按需减小可节省内存	task 数量过少会限制 VGS 性能	-	MODULE PARAM: max_task_num
设置 VGS 支持的最大 node 数	max_vgs_node	默认值 200, 按需减小可节省内存	node 数量过少会限制 VGS 性能	-	MODULE PARAM: max_node_num
设置是否支持 HDR 功能	bVgsHdrSupport	可节省 HDR buffer		仅 Hi3559AV100 支持	MODULE PARAM: bVgsHdrSupport

- GDC 模块配置：

- 参考文档：《HiMPP 媒体处理软件 V4.0 开发参考》中的“几何畸变矫正子系统”章节。
proc 信息查看命令：cat /proc/umap/gdc。
- 注意：以下模块参数均是 Linux 配置，对应的 Huawei LiteOS 模块参数结构体是 GDC_MODULE_PARAMS_S，在 sdk_init.c 文件中的 VGS_init 函数配置

措施	相关模块参数	收益	影响	注意	proc 信息
设置 GDC 支持的最大 job 数	max_gdc_job	按需减小可节省内存	job 数量过少会限制 GDC 性能。	Hi3559AV100/Hi3519AV100 默认值 128, 其它芯片默认值 32	MODULE PARAM: max_job_num
设置 GDC 支持的最大 task 数	max_gdc_task	按需减小可节省内存	task 数量过少会限制 GDC 性能	Hi3559AV100/Hi3519AV100 默认值 200, 其它芯片默认值 64	MODULE PARAM: max_task_num
设置 GDC 支持的最大 node 数	max_gdc_node	按需减小可节省内存	node 数量过少会限制	Hi3559AV100/Hi3519AV100 默认值	MODULE PARAM:



			GDC 性能	200, 其它芯片默认值 64	max_node_num
--	--	--	--------	-----------------	--------------

- VO 模块配置：

参考文档：《HiMPP 媒体处理软件 V4.0 开发参考》“视频输出”章节。

proc 信息查看命令：cat /proc/umap/vo。

措施	相关接口	收益	影响	注意	proc 信息
回放模式显示队列长度设置最小值 3	HI_MPI_VO_SetDispBufLen	高清设备可节省 1 帧 buffer。	影响 VO 显示流畅性。	Hi3516CV500、 Hi3516EV200 不支持回放模式	VIDEO LAYER STATUS 3: u32BufLen
直通模式 DispBufLen 可以设置为 0	HI_MPI_VO_SetDispBufLen	可不用分配 Display Buffer	-	满足直通模式条件参考 MPP 手册	VIDEO LAYER STATUS 3: u32BufLen
多区域聚集模式	HI_MPI_VO_SetVideoLayerAttr HI_MPI_VO_SetChnDispPos	可节省部分 MMZ 内存	聚集模式 VO 不支持缩放	Hi3516CV500、 Hi3516EV200 不支持	VIDEO LAYER STATUS 1: ClustMode CHN BASIC INFO: DispX DispY
使用 VO 自动放大功能	HI_MPI_VO_SetVideoLayerAttr	stImageSize 小于 stDispRect 使用 VO 视频层自动放大功能，节省内存，降低带宽。	-	仅 Hi3559AV100 支持	VIDEO LAYER STATUS 1: ImgW ImgH 和 DispW DispH
单区域模式 VO省 buffer 方案	HI_MPI_VO_SetModParam HI_MPI_VO_SetDispBufLen HI_MPI_VO_SetVtth HI_MPI_VO_SetVtth	单区域模式非直通时可设置显示 buff 最小为 2, 单区域直通时, 参与	减少了一块用于显示的 buffer	仅 Hi3516CV500 和 Hi3516EV200 支持	MODULE PARAM : SaveBufMode VIDEO LAYER STATUS 3: u32BufLen



措施	相关接口	收益	影响	注意	proc 信息
	tth2	轮转的 buff 较之前可减少一块			

- REGION 模块配置：

参考文档：《HiMPP 媒体处理软件 V4.0 开发参考》“区域管理”章节。proc 信息查看命令是 cat /proc/umap/rgn。

Hi3516EV200 全在线场景需要画斜线功能时最省内存的方案是：VPSS-VENC 使用低延时卷绕模式，在编码中使用 ARGB 2BPP OSD 画斜线，OSD 区域中线条不透明，其它部分全部透明。

措施	相关模块参数/接口	收益	影响	注意	proc 信息
使用 ARGB 2BPP OVERLAY	HI_MPI_RGN_Create: enPixelFmt 设置为 PIXEL_FORMAT_ARGB_2BPP	ARGB 2BPP 是 ARGB 1555 的 1/8，是 ARGB 8888 的 1/16。	只能显示 2 种颜色	仅 Hi3516EV200 和 Hi3516CV500 支持。 多个 OSD 叠加有重叠区域时，u32Layer 层次高的 OSD 会覆盖 u32Layer 层次低的 OSD。	REGION STATUS OF OVERLAY: PiFmt
使用 ARGB 2BPP OVERLAYEX_RGN	HI_MPI_RGN_Create: enPixelFmt 设置为 PIXEL_FORMAT_ARGB_2BPP	ARGB 2BPP 是 ARGB 1555 的 1/8，是 ARGB 8888 的 1/16。	只能显示 2 种颜色	仅 Hi3516EV200 支持	REGION STATUS OF OVERLAYEX: PiFmt

- HIFB 模块配置：

参考文档：《HiFB 开发指南》和《HiFB API 参考》。

proc 信息查看命令：cat /proc/umap/hifb0。

支持多图形层的芯片可能还有有 hifb1、hifb2 等等。



措施	相关模块参数/接口	收益	影响	注意	proc 信息
设置合适的图形层物理显存	Linux: video Huawei LiteOS: HIFB_MODULE_PARAMS_S	根据实际分辨率设置合适的图形层物理显存避免内存浪费。	无	-	Mem size
图形层缩放	FBIOPUT_SCREENSIZE	可节省部分 MMZ 内存。	-	-	

- AUDIO 模块配置：

参考文档：《HiMPP 媒体处理软件 V4.0 开发参考》“音频”章节。

措施	相关接口/参数	收益	影响	注意	proc 信息
按场景合理设置 AI 缓存音频帧大小	HI_MPI_AI_SetPubAttr: u32FrmNum	可节省部分 OS 内存	需要用户保证 buffer 大小设置合理，否则可能会出现采集丢帧等异常。	-	cat /proc/umap/ai AI DEV ATTR: FrmNum
按场景合理设置 AENC 缓存音频帧大小	HI_MPI_AENC_CreateChn: u32BufSize	可节省部分 OS 内存	需要用户保证 buffer 大小设置合理，否则可能会出现采集丢帧等异常。	-	cat /proc/umap/aenc AENC CHN ATTR: BufSize
按场景合理设置 ADEC 缓存音频帧大小	HI_MPI_ADEC_CreateChn: u32BufSize	可节省部分 OS 内存	需要用户保证 buffer 大小设置合理，否则可能会出现采集丢帧等异常。	-	cat /proc/umap/dec ADEC CHN ATTR: BufSize
按场景合理设置 AO 缓存音频帧大小	HI_MPI_AO_SetPubAttr: u32FrmNum	可节省部分 OS 内存	需要用户保证 buffer 大小设置合理，否则可能会出现采集丢帧等异常。	-	cat /proc/umap/ao AO DEV ATTR: FrmNum

- ISP 模块配置：

参考文档：《HiISP 开发参考.pdf》。



措施	相关模块参数	收益	影响	注意
不使用 SpecAwb 库时，可以不初始化 SpecAwb 内存。	Hi3516EV200/Hi3516EV300 默认不分配 SpecAwb 算法内存。如果需要分配，请参考《HiISP 开发参考》	可节省部分 OS 内存	需要用户保证不使用 SepcAwb 库	仅 Hi3516EV200 支持编译宏开关。 Hi3516EV200/Hi3516EV300 默认不分配 SpecAwb 内存。
进 YUV 场景不起 ISP	BT1120/BT656/BT601/MIPI _YUV 场景不需要 ISP 参与可以不启动 ISP。VI PIPE 属性 bIsPbypass 需配置为 HI_TRUE。	节省 ISP 模块内存	-	-

- IVE 模块配置：

参考文档：《HiIVE API 参考.pdf》。

proc 信息查看命令：cat /proc/umap/ive。

以下模块参数均是 Linux 配置，对应的 Huawei LiteOS 模块参数结构体是 IVE_MODULE_PARAMS_S，在 sdk_init.c 文件中的 IVE_init 函数配置。

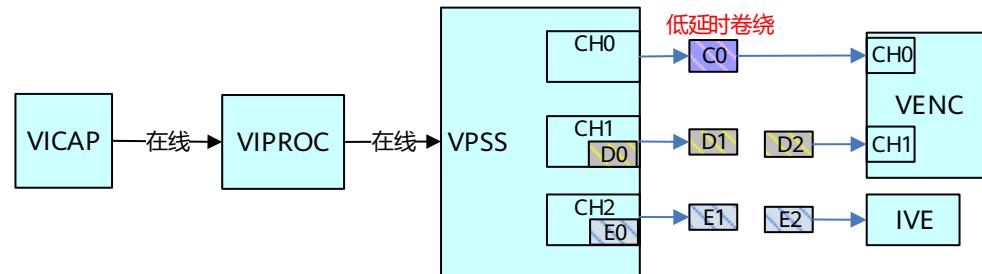
措施	相关模块参数	收益	影响	注意	proc 信息
设置 IVE 支持的最节点数	max_node_num	默认值 512，按需减小可节省内存	节点数量过少会限制 IVE 性能。	-	MODULE PARAM: max_node_num

1.2.3 公共 VB 分配指导

以 Hi3516EV200 芯片的部分场景为例，描述公共 VB 分配的指导说明。



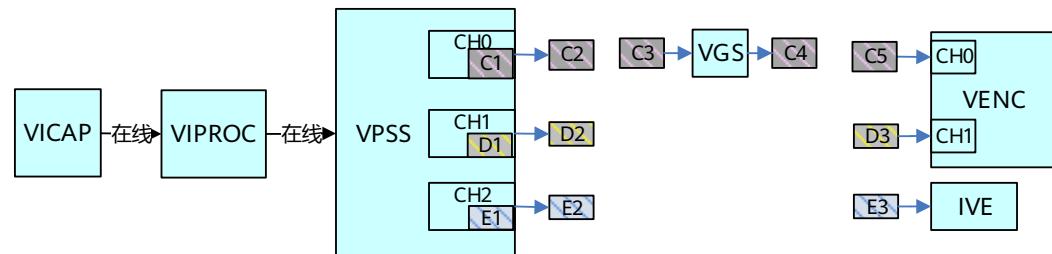
图1-1 全在线低延时卷绕场景



全在线低延时卷绕场景，VPSS CH0 需要分 1 个卷绕 VB（不到一帧大小，大小与行号有关），VPSS CH1 和 VPSS CH2 分别需要分 3 个 VB。

优化：通过接口 `HI_MPI_VPSS_SetGrpFrameInterruptAttr` 调整 VPSS Early_End 机制的行号（建议从大往小调整，直到不丢帧为止。如果所有行号都不能调整到不丢帧，说明此时序不适用 Early_End 机制。）有可能使得 CH1 和 CH2 都分别只需要分配 2 个 VB。其中 CH2 的 VB 个数还跟帧率控制有关，低帧率时 VB 可能再少占用 1 个。

图1-2 全在线场景，VPSS CH0 开启 Rotation 或 LDC

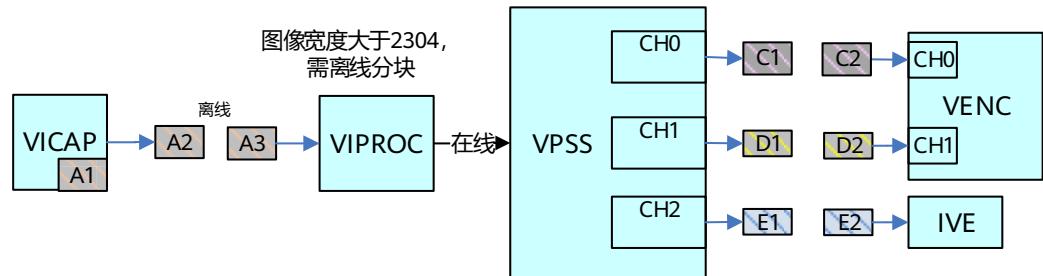


全在线场景，VPSS CH0 开启 Rotation 或者 LDC 时，VPSS、VGS、VENC 之间的大码流 VB 个数主要取决于这 3 个模块的处理速度，VB 个数在 3~5 个之间调整。

优化：通过接口 `HI_MPI_VPSS_SetGrpFrameInterruptAttr` 调整 VPSS Early_End 机制的行号（建议从大往小调整），有可能使得 CH0 的 VB 个数再减少 1 个，即大码流 VB 个数在 2~4 个之间调整。



图1-3 离线从模式分块场景，无 Rotation 和 LDC

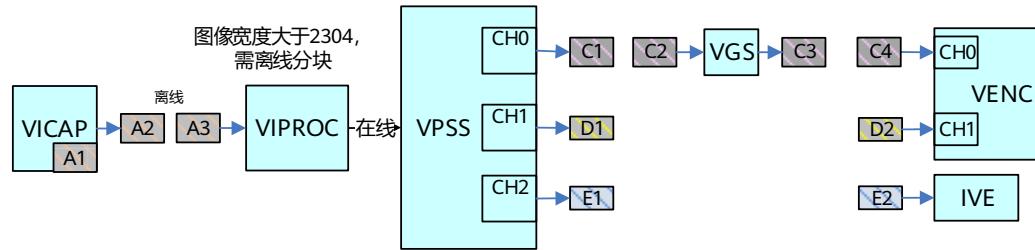


当图像宽度大于 2304 时，VICAP 和 VIPROC 必须离线，此时 VIPROC-VPSS 仍可在线。此场景需要分配 3 个 raw VB（图中的 A1、A2、A3），VPSS CH0 分配 2 个 VB，CH1 分配 2 个 VB，CH2 分配 2 个 VB。

优化：

- 通过接口 HI_MPI_VI_SetPipeFrameInterruptAttr 使能 FRAME_INTERRUPT_EARLY_END 机制，调整行号 u32EarlyLine（建议从大往小调整），有可能使得 VICAP 和 VIPROC 之间只需要分配 2 个 raw VB。或者使能 FRAME_INTERRUPT_EARLY_ONE_BUFF/ RAME_INTERRUPT_EARLY_END_ONE_BUF 机制，调整行号 u32EarlyLine，有可能使得 VICAP 和 VIPROC 之间只需要分配 1 个 raw VB。（EARLY_END 机制使用的注意事项请参考“3.4 VI 中断类型”）
- 通过 HI_MPI_VPSS_SetLowDelayAttr 接口开启 VPSS CH0 的低延时模式，加速 VPSS 和 VENC 之间的 VB 轮转，有可能使得 VPSS CH0 和 VENC 大码流之间只需要分配 1 个 VB。

图1-4 离线从模式分块场景，VPSS CH0 开启 Rotation 或 LDC



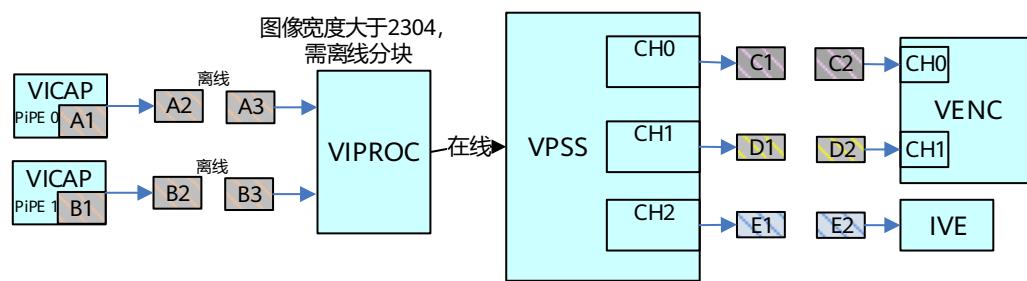
当图像宽度大于 2304 时，VICAP 和 VIPROC 必须离线，此时 VIPROC-VPSS 仍可在线。此场景需要分配 3 个 raw VB（图中的 A1、A2、A3）。

优化：



- 通过接口 `HI_MPI_VI_SetPipeFrameInterruptAttr` 使能 `FRAME_INTERRUPT_EARLY_END` 机制，调整行号 `u32EarlyLine`（建议从大往小调整），有可能使得 `VICAP` 和 `VIPROC` 之间只需要分配 2 个 raw VB。或者使能 `FRAME_INTERRUPT_EARLY_ONE_BUFF/FRAME_INTERRUPT_EARLY_END_ONE_BUF` 机制，调整行号 `u32EarlyLine`，有可能使得 `VICAP` 和 `VIPROC` 之间只需要分配 1 个 raw VB。（`EARLY_END` 机制使用的注意事项请参考“[3.4 VI 中断类型](#)”）。
- `VPSS`、`VGS`、`VENC` 之间的大码流 VB 个数主要取决于这 3 个模块的处理速度，VB 个数在 2~4 个之间调整。

图1-5 2F-WDR 离线从模式分块场景



当图像宽度大于 2304 时，`VICAP` 和 `VIPROC` 必须离线，此时 `VIPROC-VPSS` 仍可在线。此场景每路 PIPE 需要分配 3 个 raw VB（图中 PIPE0 的 A1、A2、A3 和 PIPE1 的 B1、B2、B3），`VPSS` CH0 分配 2 个 VB，CH1 分配 2 个 VB，CH2 分配 2 个 VB。

优化：

- 通过接口 `HI_MPI_VI_SetPipeFrameInterruptAttr` 使能 `FRAME_INTERRUPT_EARLY_END` 机制，调整行号 `u32EarlyLine`（建议从大往小调整），有可能使得每路 PIPE 的 `VICAP` 和 `VIPROC` 之间只需要分配 2 个 raw VB。或者使能 `FRAME_INTERRUPT_EARLY_ONE_BUFF/FRAME_INTERRUPT_EARLY_END_ONE_BUF` 机制，调整行号 `u32EarlyLine`，有可能使得每路 PIPE 的 `VICAP` 和 `VIPROC` 之间只需要分配 1 个 raw VB。或者主 PIPE 使用 `FRAME_INTERRUPT_EARLY_ONE_BUF/FRAME_INTERRUPT_EARLY_END_ONE_BUF`，另一路 PIPE 使用 `FRAME_INTERRUPT_EARLY_END`，这样 WDR 通路相比 normal 模式总共可以节省 3 个 buffer。（`EARLY_END` 机制使用的注意事项请参考“[3.4 VI 中断类型](#)”）



- 通过 HI_MPI_VPSS_SetLowDelayAttr 接口开启 VPSS CH0 的低延时模式，加速 VPSS 和 VENC 之间的 VB 轮转，有可能使得 VPSS CH0 和 VENC 大码流之间只需要分配 1 个 VB。

1.2.4 CMA 相关

在 Hi3519AV100、Hi3516CV500、Hi3516DV300 和 Hi3516AV300 等项目中，系统默认打开了 CMA。打开 CMA 后，系统会默认保留部分内存。这部分保留的内存可能只会使用其中一部分。

因此，为了节省内存，客户可以采取以下两种方法：

1、调整系统保留的内存。

客户可以通过 cat /proc/meminfo 命令查看当前系统保留的 CMA 内存和这部分内存的使用情况，其中 CmaTotal 是当前系统保留的 CMA 内存，CmaFree 是剩余的内存：

CmaTotal: 16384 kB

CmaFree: 16068 kB

客户可以通过修改内核配置调整保留内存的大小：

Device Drivers->Generic Driver Options ->Size in Mega Bytes

修改内核配置后，请重新编译内核。

2、关闭 CMA。

如果客户不使用 CMA 功能，可以通过修改内核配置关闭 CMA：

Kernel Features->Contiguous Memory Allocator

修改内核配置后，请重新编译内核和 hi_osal.ko。

1.3 性能相关

1.3.1 CPU 性能 Top 统计波动大问题

【现象】 使用 top 进行 cpu 占用率统计不是很准确，可能出现波动，特别是在小业务场景，top 统计的 cpu 占用率波动会很大。

【分析】 版本 linux kernel 默认使用 HZ 为 100，也即为 10ms 调度统计，统计时间粒度较粗，导致统计精度不够，如此波动会比较大。



【解决】如果期望比较准确的 cpu 占用率统计值，可以修改 kernel HZ 为 1000，如此可以提高统计精度。

1.3.2 绑定中断到不同 CPU 的注意事项

针对中断绑定 CPU 的操作有如下建议：

- 绑定 CPU 的操作要在业务运行之前进行，不要在业务运行过程中动态切换绑定；
- 同一个模块的多个核要绑在同一个 CPU 上；
- 把中断比较多的模块识别出来绑定到其它 CPU 上，比如网络的中断若比较多，可以把它跟媒体业务分开。

1.3.3 解码性能

【现象】Hi3519AV100：解码帧率达不到满帧，回放场景下 VO 出现 ChnRpt。

【分析】

- 解码线程对实时性要求高，如果系统线程较多，解码线程得不到及时调度，硬件利用率受到影响。
- 解码 Tile 格式输出较 Linear 格式输出效率高。

【解决】

- 通过修改 hi_usr.c 中 VDEC_SET_SCHEDULER 并将其置为 1，提高解码线程的优先级；
- 调用 HI_MPI_VDEC_SetChnParam 接口，修改 VDEC 的视频输出格式为 Tile 格式输出。

1.3.4 编码性能

【现象】

Hi3519AV100：编码达不到 4K@60 的帧率。

【分析】

VEDU 的默认时钟是 568MHz，如果编码需要 4K@60 的帧率，需要修改 VEDU 频点为 750MHz。

【解决】



请在系统加载 ko 后，参考 sample_venc.c 文件中的 SAMPLE_VENC_SetClkCfg 函数，修改 CRG 寄存器 0x04510164 的 bit[1:0]的值为 2。

1.4 小型化

1.4.1 静态库使用

【现象】应用程序只使用 libmpi.a 一小部分函数，但需要链接 mpi 库外 vqev2 等库文件，导致应用程序文件过大。

【分析】链接时默认需要链接库中所有定义函数表，从而需要引用 mpi 库中关联的其他库。

【解决】MPP 版本生成库时，Makefile.param 加入 -ffunction-sections 编译选项；客户在链接生成应用程序时加入 -Wl,-gc-sections，能有效减小应用程序大小，剔除掉没有使用到的函数。

1.5 管脚复用、时钟门控、系统控制在哪里配置？

在单 Linux multi-core 方案中，管脚复用 (pinmux)，管脚驱动能力、时钟门控 (clk) 和系统控制 (sysctl) 的配置，集中在 drv/interdrv/sysconfig.c 中进行配置，用户可以根据自身产品需要进行修改，编译成 sys_config.ko，加载 ko 后配置生效。

在双系统 (Linux+Huawei LiteOS) 方案中，管脚复用，管脚驱动能力等在 mpp/out/liteos/single/init/sdk_initl.c 中配置。

对于 Hi3516EV200：管脚复用 (pinmux)，管脚驱动能力、时钟门控 (clk) 和系统控制 (sysctl) 的配置，集中在 drv/interdrv/sysconfig.c 中进行配置，用户可以根据自身产品需要进行修改，编译成 sys_config.ko，加载 ko 后配置生效。

1.6 修改内核选项后重编 KO 流程

【现象】客户有修改内核选项的需求，有些选项修改后 KO 需要重新编译。

【分析】

【解决】



- 客户修改内核选项后重新编译内核；
- 修改 smp/a53_linux/mpp 下的 Makefile.linux.param 文件中的 KERNEL_ROOT 变量，使内核路径指向重编译后的内核路径。
- 分别重新编译 smp/a53_linux 目录下 drv/extdrv、drv/interdrv、osal、mpp/component 和 mpp/obj，生成的驱动 ko 会自动拷贝到 smp/a53_linux/mpp/ko（**注意**：旧的驱动 ko 会被覆盖）。
- osdrv/components/ipcm/ipcm/编译请参照 readme_cn.txt/readme_en.txt，编译生成文件包括 hi_ipcm.ko hi_virt-tty.ko，需要用户手动拷贝至 smp/a53_linux/mpp/ko 下。



2 MIPI 配置

2.1 MIPI 频率说明

Mipi lane 频率与 VI 频率关系

【现象】 使用 MIPI 多个 lanes 进行数据传输，mipi lane 的传输频率与 VI 处理频率如何对应，每一 lane 可传输的最高速率如何计算。

【解决】 Hi3516A 通过 MIPI 接收多 lane 数据，会转成内部时序，送给 VI 模块进行处理，多 lane 传输的数据总量不变，有这样的计算公式：

$$\text{VI_Freq} * \text{Pix_Width} = \text{Lane_Num} * \text{MIPI_Freq}$$

其中，VI_Freq 为 VI 的工作时钟， Pix_Width 为像素位宽，Lane_Num 为传输 lane 个数，MIPI_Freq 为一个 lane 能接收的最大频率。

下面以 VI 工作频率为 250M，MIPI 数据为 RAW 12, 4Lane 传输为例进行说明：

$$\text{MIPI_Freq} = (250 * 12) / 4 = 750$$

即每个 lane 最高频率为 750MHz

2.2 时序

时序要求

MIPI 接收前端时序时，要求一行数据的有效数据不能被打断，否则会导致图像异常。因此，客户如果需要终止一帧图像的传输，马上开始下一帧图像的传输，请保证不要在一行数据的有效数据中间打断。



hs_exit 调整

如果对接的 clock lane 是 non-continuous 模式 (non-continuous 模式是时钟在消隐区关掉) 而且消隐区小的 sensor, hs_exit 值配置较大时会影响 mipi_rx 检测下一个 burst, 导致 sensor 对接不通(一般表现为宽高检测不对, 丢行)。此时可尝试调节 cil_cyc_clk_hs_exit 寄存器值, 降低 mipirx 检测等待周期, cil_cyc_clk_hs_exit 最小配置值为 14。

此处以 Hi3516EV200 为例说明调节方法。修改 mipi_rx_hal.c 中宏定义 MIPI_FSMO_VALUE 的 16~21bit

```
#define MIPI_FSMO_VALUE (0x003f1d0c)
```

比如修改为 cil_cyc_clk_hs_exit 为 14 个周期修改宏定义值为 0xe1d0c



3 VI 功能

3.1 DIS 功能

3.1.1 如何实现 ISP-DIS 场景功能

ISP DIS 为两轴 DIS，通过调用 HI_MPI_ISP_SetDISAttr 打开 DIS 功能，VI 模块会通过 DIS 算法计算出图像抖动的 offset，图像送给 VPSS 后，通过 VPSS Grp Crop 功能裁剪掉偏移(如果图像分辨率不够，可以通过通道再放大)，得到 DIS 后的图像。

3.2 WDR 功能

3.2.1 QudraWDR 注意事项

用户有可能会对接某些支持 Quadra WDR 的 sensor，如 SONY IMX294。WDR 场景中，VI 需要先将长曝光的帧写到 DDR，然后在短曝光的帧发送过来的时候，读取长曝光帧进行 WDR 合成。因为 Quadra WDR 时序中，长短曝光的帧之间传输是没有行差的。这样会导致 VI 对总线的 latency 要求会比较高。如果总线的 latency 不满足的话，有可能会导致低带宽。

3.3 VI 时序配置

VI 进 YUV 的场景配置需要注意的地方比较多，首先要配置管脚复用，时钟选择（通过 load 脚本的-sensor 参数传递，细节请参见 sysconfig.c）然后对 MIPI、VI 设备和 VI PIPE 做配置，差异点如下。



须知

- VI 进 YUV 且 Bypass ISP 的场景下(pipe 属性中的 bIspBypass 配置为 HI_TRUE)，强烈建议不起 ISP 业务，避免内存等资源的浪费。
- 各接口的 load 脚本的参数，请参考各芯片的 load 脚本。

3.3.1 BT.1120

3.3.1.1 MIPI 配置

MIPI 的配置需要根据《MIPI 使用指南.doc》确定是否需要配置。

此处以 Hi3559AV100 举例说明。

Hi3559AV100 第 0, 1 路 BT.1120 与 MIPI 无关，无需配置，第 2 路需要配置 MIPI 输入模式为 INPUT_MODE_BT1120，且 devno 为对应的 CMOS 编号。

```
combo_dev_attr_t MIPI_BT1120_ATTR =  
{  
    .devno = 2,  
    .input_mode = INPUT_MODE_BT1120,  
    .data_rate = DATA_RATE_X1,  
    .img_rect = {0, 0, 1920, 1080},  
  
    {  
        .mipi_attr =  
        {  
            DATA_TYPE_RAW_12BIT,  
            HI_MIPI_WDR_MODE_NONE,  
            {0, 1, 2, 3, -1, -1, -1, -1}  
        }  
    }  
};
```

3.3.1.2 VI DEV 配置

- 接口模式：VI_MODE_BT1120_STANDARD
- Mask 设置：au32ComponentMask[0] = 0xFF000000, au32ComponentMask[1] = 0x00FF0000



- 扫描格式：只支持逐行 VI_SCAN_PROGRESSIVE
- UV 顺序：UV 顺序根据实际输入时序确定是 VI_DATA_SEQ_VUVU 还是 VI_DATA_SEQ_UVUV
- 数据类型：BT1120 进 YUV 数据，因此是 VI_DATA_TYPE_YUV
- Hi3516CV500 配置 BT1120 时需使用 VI_DEV1。其管脚复用的配置在 sysconfig.c 文件已配置封装好，可通过对应芯片的 load 脚本指令进行对应的选择，详情请查看 ko 目录下的 load3516cv500。

```
VI_DEV_ATTR_S DEV_BT1120_ATTR =
{
    VI_MODE_BT1120_STANDARD,
    VI_WORK_MODE_1Multiplex,
    {0xFF000000, 0x00FF0000},
    VI_SCAN_PROGRESSIVE,
    { -1, -1, -1, -1},
    VI_DATA_SEQ_VUVU,

    {
        VI_VSYNC_PULSE, VI_VSYNC_NEG_LOW, VI_HSYNC_VALID_SIGNAL,
        VI_HSYNC_NEG_HIGH, VI_VSYNC_VALID_SIGNAL, VI_VSYNC_VALID_NEG_HIGH,
        {
            0,           1920,          0,
            0,           1080,          0,
            0,           0,             0
        }
    },
    VI_DATA_TYPE_YUV,
    HI_FALSE,
    {1920, 1080},
    {
        {
            {1920, 1080},
        },
        {
    }
}
```



```
        VI_REPHASE_MODE_NONE,  
        VI_REPHASE_MODE_NONE  
    }  
},  
  
{  
    WDR_MODE_NONE,  
    1080  
},  
  
    DATA_RATE_X1  
};
```

3.3.1.3 VI PIPE 配置

- PIPE 的 blspBypass 设置为 HI_TRUE。
- PIPE 的像素格式设置为 PIXEL_FORMAT_YVU_SEMIPLANAR_422。
- PIPE 的 bit 位宽 nBitWidth 设置为 DATA_BITWIDTH_8。

```
VI_PIPE_ATTR_S PIPE_BT1120_ATTR =  
{  
    VI_PIPE_BYPASS_NONE, HI_FALSE, HI_TRUE,  
    1920, 1080,  
    PIXEL_FORMAT_YVU_SEMIPLANAR_422,  
    COMPRESS_MODE_NONE,  
    DATA_BITWIDTH_8,  
    HI_FALSE,  
    {  
        PIXEL_FORMAT_YVU_SEMIPLANAR_422,  
        DATA_BITWIDTH_8,  
        VI_NR_REF_FROM_RFR,  
        COMPRESS_MODE_NONE  
    },  
    HI_FALSE,  
    {-1, -1}  
};
```



3.3.2 BT.656

3.3.2.1 MIPI 配置

MIPI 的配置需要根据《MIPI 使用指南.doc》确定是否需要配置。

此处以 Hi3559AV100 举例说明。

Hi3559AV100 第 0, 1 路 BT.656 与 MIPI 无关，无需配置，第 2 路需要配置 MIPI 输入模式为 INPUT_MODE_BT656，且 devno 为对应的 CMOS 编号。

```
combo_dev_attr_t MIPI_BT656_ATTR =  
{  
    .devno = 2,  
    .input_mode = INPUT_MODE_BT656,  
    .data_rate = DATA_RATE_X1,  
    .img_rect = {0, 0, 720, 576},  
  
    {  
        .mipi_attr =  
        {  
            DATA_TYPE_RAW_12BIT,  
            HI_MIPI_WDR_MODE_NONE,  
            {0, 1, 2, 3, -1, -1, -1, -1}  
        }  
    }  
};
```

3.3.2.2 VI DEV 配置

- 接口模式：VI_MODE_BT656
- Mask 设置：只需设置 au32ComponentMask[0] = 0xFF000000
- 扫描格式：只支持逐行 VI_SCAN_PROGRESSIVE
- UV 顺序：UV 顺序根据实际输入时序确定，
VI_DATA_SEQ_UYVY/VI_DATA_SEQ_VYUY/VI_DATA_SEQ_YUYV/VI_DATA_SEQ_YVYU。
- 数据类型：BT656 进 YUV 数据，因此是 VI_DATA_TYPE_YUV

```
VI_DEV_ATTR_S DEV_BT656_ATTR =  
{
```



```
VI_MODE_BT656,  
VI_WORK_MODE_1Multiplex,  
{0xFF000000, 0x00FF0000},  
VI_SCAN_PROGRESSIVE,  
{ -1, -1, -1, -1},  
VI_DATA_SEQ_YUYV,  
  
{  
    VI_VSYNC_PULSE, VI_VSYNC_NEG_LOW, VI_HSYNC_VALID_SINGNAL,  
    VI_HSYNC_NEG_HIGH, VI_VSYNC_VALID_SINGAL, VI_VSYNC_VALID_NEG_HIGH,  
  
    {  
        0, 720, 0,  
        0, 576, 0,  
        0, 0, 0  
    }  
},  
VI_DATA_TYPE_YUV,  
  
HI_FALSE,  
  
{720 , 576},  
  
{  
    {  
        {720 , 576},  
    },  
    {  
        VI_REPHASE_MODE_NONE,  
        VI_REPHASE_MODE_NONE  
    }  
},  
  
{  
    WDR_MODE_NONE,  
    576  
},
```



```
DATA_RATE_X1  
};
```

3.3.2.3 VI PIPE 配置

- PIPE 的 bIspBypass 设置为 HI_TRUE。
- PIPE 的像素格式设置为 PIXEL_FORMAT_YVU_SEMIPLANAR_422。
- PIPE 的 bit 位宽 nBitWidth 设置为 DATA_BITWIDTH_8。

```
VI_PIPE_ATTR_S PIPE_BT656_ATTR =  
{  
    VI_PIPE_BYPASS_NONE, HI_FALSE, HI_TRUE,  
    720, 576,  
    PIXEL_FORMAT_YVU_SEMIPLANAR_422,  
    COMPRESS_MODE_NONE,  
    DATA_BITWIDTH_8,  
    HI_FALSE,  
    {  
        PIXEL_FORMAT_YVU_SEMIPLANAR_422,  
        DATA_BITWIDTH_8,  
        VI_NR_REF_FROM_RFR,  
        COMPRESS_MODE_NONE  
    },  
    HI_FALSE,  
    {-1, -1}  
};
```

3.3.3 BT.601

3.3.3.1 MIPI 配置

MIPI 的配置需要根据《MIPI 使用指南.doc》确定是否需要配置。此处以 Hi3559AV100 举例说明。

Hi3559AV100 第 0, 1 路 BT.656 与 MIPI 无关无需配置，第 2 路需要配置 MIPI 输入模式为 INPUT_MODE_BT601，且 devno 为对应的 CMOS 编号。

```
combo_dev_attr_t MIPI_BT601_ATTR =  
{
```



```
.devno = 2,  
.input_mode = INPUT_MODE_BT601,  
.data_rate = DATA_RATE_X1,  
.img_rect = {0, 0, 720, 576},  
  
{  
.mipi_attr =  
{  
DATA_TYPE_RAW_12BIT,  
HI_MIPI_WDR_MODE_NONE,  
{0, 1, 2, 3, -1, -1, -1, -1}  
}  
}  
};
```

3.3.3.2 VI DEV 配置

- 接口模式: VI_MODE_BT601
- Mask 设置: 只需设置 au32ComponentMask[0] = 0xFF000000
- 扫描格式: 只支持逐行 VI_SCAN_PROGRESSIVE
- 时序参数: 时序参数配置请参考视频输入章节 VI_SYNC_CFG_S 的说明。
- UV 顺序: UV 顺序根据实际输入时序确定是 VI_DATA_SEQ_VUVU 还是 VI_DATA_SEQ_UVUV
- 数据类型: BT.601 进 YUV 数据, 因此是 VI_DATA_TYPE_YUV

```
VI_DEV_ATTR_S DEV_BT601_ATTR =  
{  
VI_MODE_BT601,  
VI_WORK_MODE_1Multiplex,  
{0xFF000000, 0x00FF0000},  
VI_SCAN_PROGRESSIVE,  
{ -1, -1, -1, -1},  
VI_DATA_SEQ_YUYV,  
  

```



```
{  
    0,           720,           0,  
    0,           576,           0,  
    0,           0,             0  
}  
,  
VI_DATA_TYPE_YUV,  
  
HI_FALSE,  
  
{720, 576},  
  
{  
{  
    {720, 576},  
},  
{  
    VI_REPHASE_MODE_NONE,  
    VI_REPHASE_MODE_NONE  
}  
},  
  
{  
    WDR_MODE_NONE,  
    576  
},  
  
DATA_RATE_X1  
};
```

3.3.3.3 VI PIPE 配置

- PIPE 的 bIsPbypass 设置为 HI_TRUE。
- PIPE 的像素格式设置为 PIXEL_FORMAT_YVU_SEMIPLANAR_422。
- PIPE 的 bit 位宽 nBitWidth 设置为 DATA_BITWIDTH_8。

```
VI_PIPE_ATTR_S PIPE_BT1120_ATTR =  
{
```



```
VI_PIPE_BYPASS_NONE, HI_FALSE, HI_TRUE,  
720, 576,  
PIXEL_FORMAT_YVU_SEMIPLANAR_422,  
COMPRESS_MODE_NONE,  
DATA_BITWIDTH_8,  
HI_FALSE,  
{  
    PIXEL_FORMAT_YVU_SEMIPLANAR_422,  
    DATA_BITWIDTH_8,  
    VI_NR_REF_FROM_RFR,  
    COMPRESS_MODE_NONE  
},  
HI_FALSE,  
{-1, -1}  
};
```

3.3.4 MIPI_YUV

3.3.4.1 MIPI 配置

- 配置 MIPI 输入模式为 INPUT_MODE_MIPI。
- MIPI 属性输入数据类型 input_data_type 根据输入的数据类型设置，若输入 YUV422 则设置为 DATA_TYPE_YUV422_8BIT，输入为 legacy YUV420 则设置为 DATA_TYPE_YUV420_8BIT_LEGACY，输入为 normal YUV420 则设置为 DATA_TYPE_YUV420_8BIT_NORMAL。

```
ombo_dev_attr_t MIPI_YUV422_ATTR =  
{  
    .devno = 0,  
    .input_mode = INPUT_MODE_MIPI,  
    .data_rate = DATA_RATE_X1,  
    .img_rect = {0, 0, 1920, 1080},  
  
    {  
        .mipi_attr =  
        {  
            DATA_TYPE_YUV422_8BIT,  
            HI_MIPI_WDR_MODE_NONE,  
            {0, 1, 2, 3, -1, -1, -1, -1}  
        }  
    }  
};
```



```
    }
}
};
```

3.3.4.2 VI DEV 配置

- 接口模式：根据输入的数据类型设置，若输入 YUV422 则设置为 VI_MODE_MIPI_YUV422，输入为 legacy YUV420 则设置为 VI_MODE_MIPI_YUV420_LEGACY，输入为 normal YUV420 则设置为 VI_MODE_MIPI_YUV420_NORMAL
- Mask 设置：au32ComponentMask[0] = 0xFF000000, au32ComponentMask[1] = 0x00FF0000
- 扫描格式：只支持逐行 VI_SCAN_PROGRESSIVE
- UV 顺序：UV 顺序根据实际输入时序确定是 VI_DATA_SEQ_VUVU 还是 VI_DATA_SEQ_UVUV
- 数据类型：MIPI 进 YUV 数据，因此是 VI_DATA_TYPE_YUV

```
VI_DEV_ATTR_S DEV_MIPI_YUV422_ATTR =
{
    VI_MODE_MIPI_YUV422,
    VI_WORK_MODE_1Multiplex,
    {0xFF000000,     0x00FF0000},
    VI_SCAN_PROGRESSIVE,
    { -1, -1, -1, -1},
    VI_DATA_SEQ_VUVU,

    {

        VI_VSYNC_PULSE, VI_VSYNC_NEG_LOW, VI_HSYNC_VALID_SIGNAL,
        VI_HSYNC_NEG_HIGH, VI_VSYNC_VALID_SIGNAL, VI_VSYNC_VALID_NEG_HIGH,


        {
            0,           1920,          0,
            0,           1080,          0,
            0,           0,             0
        }
    },
    VI_DATA_TYPE_YUV,
    HI_FALSE,
```



```
{1920 , 1080},  
  
{  
{  
{1920 , 1080},  
},  
{  
VI_REPHASE_MODE_NONE,  
VI_REPHASE_MODE_NONE  
}  
},  
  
{  
WDR_MODE_NONE,  
1080  
},  
  
DATA_RATE_X1  
};
```

3.3.4.3 VI PIPE 配置

- PIPE 的 bIspBypass 设置为 HI_TRUE。
- PIPE 的像素格式设置为 PIXEL_FORMAT_YVU_SEMIPLANAR_422。
- PIPE 的 bit 位宽 nBitWidth 设置为 DATA_BITWIDTH_8。

```
VI_PIPE_ATTR_S PIPE_BT1120_ATTR =  
{  
VI_PIPE_BYPASS_NONE, HI_FALSE, HI_TRUE,  
1920, 1080,  
PIXEL_FORMAT_YVU_SEMIPLANAR_422,  
COMPRESS_MODE_NONE,  
DATA_BITWIDTH_8,  
HI_FALSE,  
{  
PIXEL_FORMAT_YVU_SEMIPLANAR_422,  
DATA_BITWIDTH_8,
```



```
    VI_NR_REF_FROM_RFR,  
    COMPRESS_MODE_NONE  
},  
HI_FALSE,  
{-1, -1}  
};
```

3.3.5 LVDS

3.3.5.1 LVDS 配置

- 配置 MIPI 输入模式为 INPUT_MODE_LVDS/ INPUT_MODE_SUBLVDS/ INPUT_MODE_HISPI。
- lvds 属性输入数据类型 input_data_type 根据输入的数据类型设置。
- 输入数据速率必须与 VI 的设备保持一致，这里举例使用 MIPI_DATA_RATE_X2。

```
combo_dev_attr_t LVDS_12BIT_ATTR =  
{  
    .devno          = 0,  
    .input_mode     = INPUT_MODE_LVDS,  
    .data_rate      = MIPI_DATA_RATE_X2,  
    .img_rect       = {0, 0, 7840, 4320},  
    .lvds_attr      =  
    {  
        .input_data_type   = DATA_TYPE_RAW_12BIT,  
        .wdr_mode         = HI_WDR_MODE_NONE,  
        .sync_mode         = LVDS_SYNC_MODE_SAV,  
        .vsync_attr        = {LVDS_VSYNC_NORMAL, 0, 0},  
        .fid_attr          = {LVDS_FID_NONE, HI_TRUE},  
        .data_endian       = LVDS_ENDIAN_LITTLE,  
        .sync_code_endian  = LVDS_ENDIAN_LITTLE,  
        .lane_id          = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15},  
        .sync_code         =  
        {  
            /* each vc has 4 params, sync_code[i]:  
             sync_mode is SYNC_MODE_SOF: SOF, EOF, SOL, EOL  
             sync_mode is SYNC_MODE_SAV: invalid sav, invalid eav, valid sav, valid eav */  
            {           //SOF   EOF   SOL   EOL  
                {0x200, 0x300, 0x400, 0xC00},  //Virtual Channel 0
```



```
{0x200, 0x300, 0x400, 0xC00}, //Virtual Channel 1  
{0x200, 0x300, 0x400, 0xC00}, //Virtual Channel 2  
{0x200, 0x300, 0x400, 0xC00} //Virtual Channel 3  
,  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
,  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
,  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
,  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
,  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},
```



```
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
},  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
},  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
},  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
},  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
},  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
},  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}
```



```
{0x200, 0x300, 0x400, 0xC00}  
},  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
},  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
},  
  
{ {0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00},  
{0x200, 0x300, 0x400, 0xC00}  
},  
},  
};
```

3.3.5.2 VI DEV 配置

- 接口模式：根据输入的接口类型设置为 VI_MODE_LVDS/VI_MODE_HISPI。
- Mask 设置：au32ComponentMask[0] = 0xFFFF00000

```
VI_DEV_ATTR_S DEV_LVDS_RAW12_ATTR =  
{  
    VI_MODE_LVDS,  
    VI_WORK_MODE_1Multiplex,
```



```
{0xFFFF0000, 0x0},  
VI_SCAN_PROGRESSIVE,  
{-1, -1, -1, -1},  
VI_DATA_SEQ_YUYV,  
  
{  
    VI_VSYNC_PULSE, VI_VSYNC_NEG_LOW, VI_HSYNC_VALID_SINGNAL,  
    VI_HSYNC_NEG_HIGH, VI_VSYNC_VALID_SINGAL, VI_VSYNC_VALID_NEG_HIGH,  
    {0, 1280, 0,  
     0, 720, 0,  
     0, 0, 0}  
},  
VI_DATA_TYPE_RGB,  
HI_FALSE,  
{7680, 4320},  
{  
    {  
        {7680, 4320},  
    },  
    {  
        VI_REPHASE_MODE_NONE,  
        VI_REPHASE_MODE_NONE  
    }  
},  
{  
    WDR_MODE_NONE,  
    4320  
},  
DATA_RATE_X2  
};
```

3.3.5.3 VI PIPE 配置

- PIPE 的 blspBypass 设置为 HI_FALSE。
- PIPE 的像素格式设置为 PIXEL_FORMAT_RGB_BAYER_12BPP。
- PIPE 的 bit 位宽 nBitWidth 设置为 DATA_BITWIDTH_12。

```
VI_PIPE_ATTR_S PIPE_LVDS_ATTR =
```



```
{  
    VI_PIPE_BYPASS_NONE, HI_FALSE, HI_FALSE,  
    7680, 4320,  
    PIXEL_FORMAT_RGB_BAYER_12BPP,  
    COMPRESS_MODE_LINE,  
    DATA_BITWIDTH_12,  
    HI_TRUE,  
    {  
        PIXEL_FORMAT_YVU_SEMIPLANAR_420,  
        DATA_BITWIDTH_10,  
        VI_NR_REF_FROM_RFR,  
        COMPRESS_MODE_NONE  
    },  
    HI_FALSE,  
    {-1, -1}  
};
```

3.3.6 DC

3.3.6.1 MIPI 配置

- 配置 MIPI 输入模式为 INPUT_MODE_CMOS，各芯片的 MIPI 设备号请参考《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频输入”章节的绑定关系。
- 这里举例使用如下：

```
combo_dev_attr_t MIPI_CMOS_ATTR =  
{  
    .devno = 0,  
    .input_mode = INPUT_MODE_CMOS,  
    .data_rate = DATA_RATE_X1,  
    .img_rect = {0, 0, 1920, 1080},  
};
```

3.3.6.2 VI DEV 配置

- 接口模式：根据输入的接口类型设置为 VI_MODE_DIGITAL_CAMERA。
- Mask 设置：au32ComponentMask[0] = 0xFFFF00000
- 关于各芯片的 VI 设备号请参考《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频输入”章节的绑定关系。



```
VI_DEV_ATTR_S DEV_DC_RAW12_ATTR =  
{  
    VI_MODE_DIGITAL_CAMERA,  
    VI_WORK_MODE_1Multiplex,  
    {0xFFFF0000,      0x0},  
    VI_SCAN_PROGRESSIVE,  
    {-1, -1, -1, -1},  
    VI_DATA_SEQ_YUYV,  
  
    {  
        VI_VSYNC_PULSE, VI_VSYNC_NEG_LOW, VI_HSYNC_VALID_SIGNAL,  
        VI_HSYNC_NEG_HIGH, VI_VSYNC_VALID_SIGNAL, VI_VSYNC_VALID_NEG_HIGH,  
        {0,             1280,           0,  
         0,             720,            0,  
         0,             0,              0}  
    },  
    VI_DATA_TYPE_RGB,  
    HI_FALSE,  
    {1920, 1080},  
    {  
        {  
            {1920, 1080},  
        },  
        {  
            VI_REPHASE_MODE_NONE,  
            VI_REPHASE_MODE_NONE  
        }  
    },  
    {  
        WDR_MODE_NONE,  
        1080  
    },  
    DATA_RATE_X1  
};
```

3.3.6.3 VI PIPE 配置

- PIPE 的 bIspBypass 设置为 HI_FALSE。



- PIPE 的像素格式设置为 PIXEL_FORMAT_RGB_BAYER_12BPP。
- PIPE 的 bit 位宽 nBitWidth 设置为 DATA_BITWIDTH_12。

```
VI_PIPE_ATTR_S PIPE_LVDS_ATTR =  
{  
    VI_PIPE_BYPASS_NONE, HI_FALSE, HI_FALSE,  
    1920, 1080,  
    PIXEL_FORMAT_RGB_BAYER_12BPP,  
    COMPRESS_MODE_LINE,  
    DATA_BITWIDTH_12,  
    HI_TRUE,  
    {  
        PIXEL_FORMAT_YVU_SEMIPLANAR_420,  
        DATA_BITWIDTH_10,  
        VI_NR_REF_FROM_RFR,  
        COMPRESS_MODE_NONE  
    },  
    HI_FALSE,  
    {-1, -1}  
};
```

3.4 VI 中断类型

VI 模块的采集准备与发送都是在中断中进行的，不同的中断类型下采集与发送的时机，占用 DDR 资源，使用限制各不相同。

- VICAP-VIPROC-VPSS 全在线时，中断类型通过 HI_MPI_VPSS_SetGrpFrameInterruptAttr 接口设置。
- VICAP 和 VIPROC 离线时，或者 VICAP-VIPROC 在线并且 VIPROC 和 VPSS 离线时，中断类型通过 HI_MPI_VI_SetPipeFrameInterruptAttr 接口设置。

3.4.1 FRAME_INTERRUPT_START 类型

中断处理：在帧起始中断中采集准备与发送采集完成的帧。

DDR 占用：采集过程中 VB 的占用周期是固定的 2 帧时间。

优点：帧采集完整无误。



缺点：VB 占用的时间比较长。

调试方法：无

3.4.2 FRAME_INTERRUPT_EARLY 类型

中断处理：在提前上报中断中采集准备与发送采集完成的帧。

DDR 占用：采集过程中 VB 的占用周期是固定的 1 帧时间。

优点：占用 VB 时间非常短，在内存紧张的情况下使用该中断类型省内存。

缺点：

- 帧采集到 u32EarlyLine 行之后就会发给后端处理，u32EarlyLine 行设置较小的时候可能导致问题。
- 后端模块的处理赶上采集的速度可能导致图像花屏等异常。
- 行压缩和帧压缩可能导致 VIPROC 出现异常，因此不能使用行压缩和帧压缩。
- 拍照通路不能使用。
- u32EarlyLine 行设置较小的时候，可能会导致 ISP 处理图像闪烁现象。

调试方法：通过调整 u32EarlyLine 的数值，保证后端模块进度不会超过采集的进度，否则可能导致图像花屏等异常。u32EarlyLine 数值越大，后端模块进度超过采集进度的风险越小，但是 u32EarlyLine 值太大，省内存的效果可能会变差。例如 VI 离线的情况下，采集行压缩或者帧压缩的 raw 就有可能导致 VIPROC 报错误中断。

u32EarlyLine 依赖时序，帧率，后端模块的性能，假设 VI 采集一帧的时间是 T1(帧起始到帧完成的时间，不包含消隐区)，图像高度是 H，后端模块(VIPROC, VPSS, VENC 等)处理一帧的时间是 T2，则：

$$(T1/H) * (H - u32EarlyLine) \leq T2$$

3.4.3 FRAME_INTERRUPT_EARLY_END 类型

中断处理：在提前上报中断中采集准备，在帧完成中断中发送采集完成的帧。

DDR 占用：采集过程中 VB 的占用周期是 1 帧到 2 帧的时间。

优点：

- 帧采集完整无误。
- 在低帧率，不连续的序列下能及时采集数据。

缺点：



- VB 占用的时间比较长，但是在 u32EarlyLine 配置为图像高度-1，VB 的占用周期是固定的 1 帧时间。
- 中断个数增多，增加 CPU 消耗。
- 当 sensor 输出时序消隐区比较小时可能会导致 ISP 处理图像闪烁现象。

调试方法：在后消隐区比较大的情况下，强烈建议 u32EarlyLine 配置为图像高度-1，如果后消隐区非常小导致帧完成与帧起始中断重合，则调整 u32EarlyLine 比图像高度小直至保证 VI 不丢帧。

3.4.4 FRAME_INTERRUPT_EARLY_END_ONE_BUF 类型

中断处理：VI 离线场景，在提前上报中断中采集准备，在完成中断中发送采集完成的帧。

- 相对 FRAME_INTERRUPT_EARLY_END 方案优点：
相对 FRAME_INTERRUPT_EARLY_END 方案，能进一步节省一个 VB。
- 相对 FRAME_INTERRUPT_EARLY_END 方案缺点：
 - 对 sensor 时序消隐区有要求，当 sensor 输出时序消隐区比较小时不适用 ONE_BUF 中断类型，可能会丢帧或采集跨帧。
 - WDR 长帧短帧曝光时间差（长短帧的帧起始时间差）尽量小，曝光时间差大时会导致丢帧或采集跨帧。

-----VI INTERRUPT ONEBUF INFO-----				
ProcCostTime	CapturedCostTime	LowerLine	UpperLine	WdrExposureInterval
10830	21245	1006	1069	0

参照 PROC 信息说明使用约束：假设 VI 采集一帧的时间是 CapturedCostTime (帧起始到帧完成的时间，不包含消隐区)，后端模块(VIPROC, VPSS, VENC 等)处理一帧的时间是 ProcCostTime，同一个 PIPE 相邻两帧帧间隔 T (比如帧率为 30 时，间隔为 33.33ms)，WDR 长短帧帧起始间隔 WdrExposureInterval (线性时为 0) 则需满足：
 $ProcCostTime + CapturedCostTime + WdrExposureInterval < T$ 。

需要注意这三个值可能随场景变化而波动，当波动范围超出 T 时可能会导致丢帧，因此要尽量保证这三个值的和较小。

FRAME_INTERRUPT_EARLY_END_ONE_BUF 场景 PROC 信息 LowerLine, UpperLine 无实际意义。

增强 FRAME_INTERRUPT_EARLY_END_ONE_BUF 适用性方法：



- 减小 ProcCostTime, CapturedCostTime 与 WdrExposureInterval 的值。增大 VICAP, VIPOC, VPSS 工作时钟可以降低 ProcCostTime, CapturedCostTime 的值。固定曝光比, 减小长短帧曝光时间差可降低 WdrExposureInterval 的值。
- 通过调整 sensor 时序增大消隐区时间, 增大相邻帧的间隔。比如 30fps 的 sensor 时序消隐区不够长时, 可以通过提高 sensor 的传输速率, 降低 sensor 有效数据的传输时间来达到增大消隐区的目的。(例如提高 sensor 时序到 60fps, 让 sensor 实际有效输出 30fps)

3.4.5 FRAME_INTERRUPT_EARLY_ONE_BUF 类型

中断处理：在提前上报中断中采集准备与发送采集完成的帧。

- 相对 FRAME_INTERRUPT_EARLY_END 方案优点：
相对 FRAME_INTERRUPT_EARLY_END 方案, 能进一步节省一个 VB。
- 相对 FRAME_INTERRUPT_EARLY_END 方案缺点：
 - 对 sensor 时序消隐区有要求, 当 sensor 输出时序消隐区比较小时不适用 ONE_BUF 中断类型, 可能会丢帧或采集跨帧。
 - WDR 长帧短帧曝光时间差 (长短帧的帧起始时间差) 尽量小, 曝光时间差大时会导致丢帧或采集跨帧。
- 相对 FRAME_INTERRUPT_EARLY_END_ONE_BUF 优点：
省 VB 效果同 FRAME_INTERRUPT_EARLY_END_ONE_BUF, 对时序消隐区要求比 FRAME_INTERRUPT_EARLY_END_ONE_BUF 更宽松。
- 相对 FRAME_INTERRUPT_EARLY_END_ONE_BUF 缺点：
 - 使用前需要调节出合适的 u32EarlyLine。
 - 当 CPU 繁忙, 中断处理不及时, 更容易会丢帧, 稳定性不如 FRAME_INTERRUPT_EARLY_END_ONE_BUF。

-----VI INTERRUPT ONEBUF INFO-----				
ProcCostTime	CapturedCostTime	LowerLine	UpperLine	WdrExposureInterval
10830	21245	1006	1069	0

参照 PROC 信息说明使用约束：假设 VI 采集一帧的时间是 CapturedCostTime (帧起始到帧完成的时间, 不包含消隐区), 后端模块(VIPROC, VPSS, VENC 等)处理一帧的时间是 ProcCostTime, 同一个 PIPE 相邻两帧间隔 T (举例帧率为 30 时, 间隔则为 33.33ms), WDR 长短帧帧起始间隔 WdrExposureInterval (线性时为 0) 则需满足：
 $ProcCostTime + CapturedCostTime + WdrExposureInterval < T$,



在 FRAME_INTERRUPT_EARLY_ONE_BUF 方案中，u32EarlyLine 有上下限，太小或太大都可能导致丢帧。为方便客户使用，PROC 信息会提供建议的上下限[LowerLine, UpperLine]。建议取范围的中间值。使用前需要根据 PROC 信息调节出可用的 u32EarlyLine。

当 ProcCostTime+ CapturedCostTime + WdrExposureInterval >T 时，可以调小 u32EarlyLine。

需要注意这三个值可能随场景变化而波动，当波动范围超出 T 时可能会导致丢帧，因此要尽量保证这三个值的和较小。

增强 FRAME_INTERRUPT_EARLY_ONE_BUF 方法：

- 减小 ProcCostTime, CapturedCostTime 与 WdrExposureInterval 的值。增大 VICAP, VIPOC, VPSS 工作时钟可以降低 ProcCostTime, CapturedCostTime 的值。固定曝光比，减小长短帧曝光时间差可降低 WdrExposureInterval 的值。
- **通过调整 sensor 时序增大消隐区时间，增大相邻帧的间隔。**比如 30fps 的 sensor 时序消隐区不够长时，可以通过提高 sensor 的传输速率，降低 sensor 有效数据的传输时间来达到增大消隐区的目的。（例如提高 sensor 时序到 60fps，让 sensor 实际有效输出 30fps）

3.5 P/N 制切换方法

设置 ISP 接口来调节输出帧率

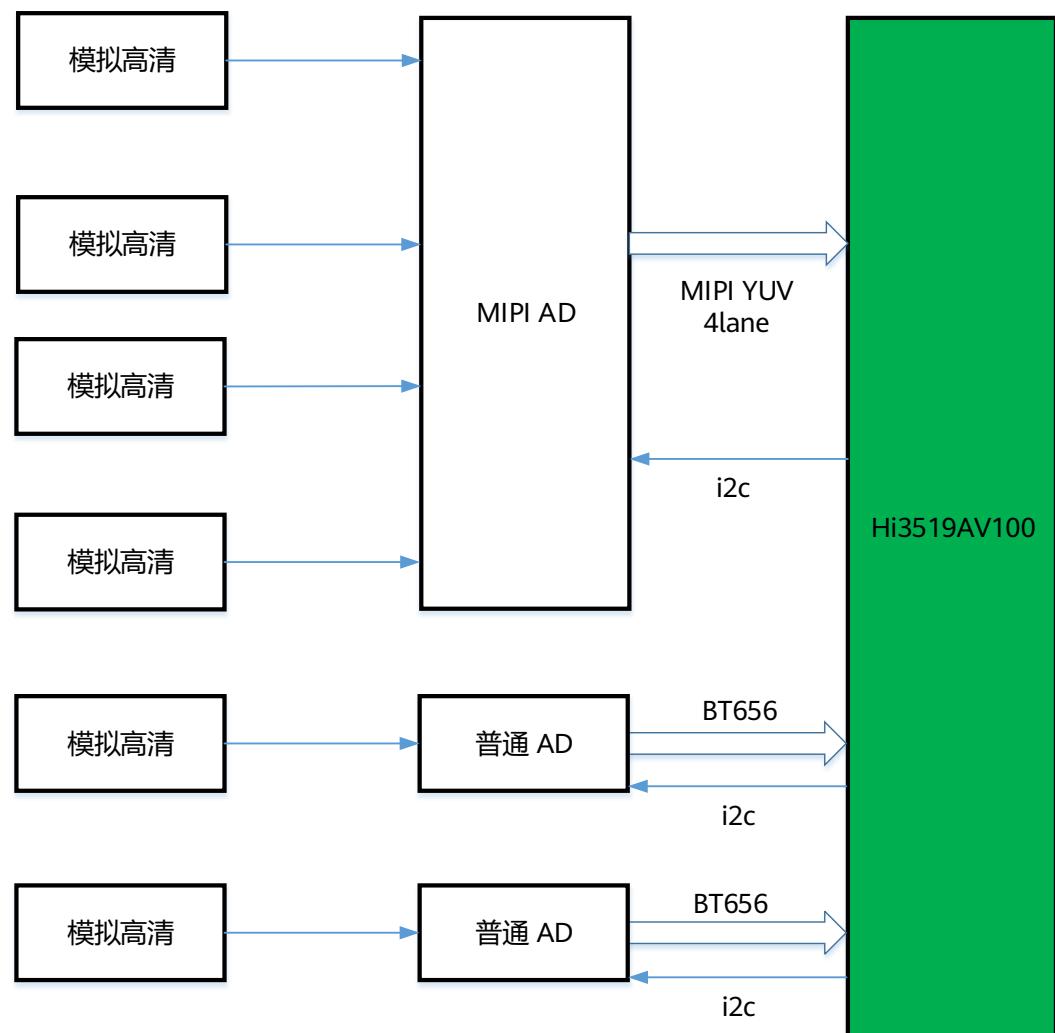
在运行业务过程中，只需要调用 HI_MPI_ISP_SetPubAttr 接口设置 Pub 属性里面的 f32FrameRate。f32FrameRate 设置成 30 为 N 制；f32FrameRate 设置成 25 为 P 制。



3.6 Hi3519AV100 对接 6 路 YUV 配置

3.6.1 1 个 MIPI AD 4 路复合 + 2 个 BT656 单路

图3-1 6 路 YUV 解决方案图



- 前端 MIPI 对接 AD 4 路复合，在 VI PIPE2~5 YUV 写出。
- 2 路 BT656 在 VI 按 BT656_PACKED_YUV 格式接收，在 VI PIPE 0~1 按 8bit raw 写出，通过 iovec dma 进行格式转换成 sp422 (格式转换请见 iovec sample)，配置如下：
 - MIPI 配置，如图 3-2 所示。



图3-2 MIPI 配置

```
combo_dev_attr_t BT656_2M_30FPS_ATTR =  
{  
    .devno = 0,  
    .input_mode = INPUT_MODE_BT656,  
    .data_rate = MIPI_DATA_RATE_X1,  
    .img_rect = {0, 0, 1920, 1080},  
};
```

- VI DEV 属性配置，如图 3-3 所示。

图3-3 VI DEV 属性配置

```
VI_DEV_ATTR_S DEV_ATTR_BT656_BASE =  
{  
    VI_MODE_BT656_PACKED_YUV,  
    VI_WORK_MODE_1Multiplex,  
    {0xFF000000, 0x0},  
    VI_SCAN_PROGRESSIVE,  
    {-1, -1, -1, -1},  
    VI_DATA_SEQ_YUYV,  
  
    /*port_vsync  port_vsync_neg  port_hsync      port_hsync_neg */  
    {0, 1280, 0, 0},  
    /*vsync0_vhb vsync0_act vsync0_hhb*/  
    {0, 720, 0, 0},  
    /*vsync1_vhb vsync1_act vsync1_hhb*/  
    {0, 0, 0, 0},  
    VI_DATA_TYPE_YUV,  
    HI_FALSE,  
    {1920, 1080},  
    {  
        {1920, 1080},  
        {  
            VI_REPHASE_MODE_NONE,  
            VI_REPHASE_MODE_NONE  
        }  
    },  
    {  
        WDR_MODE_NONE,  
        1080  
    },  
    DATA_RATE_X1  
};
```

VI PIPE 属性配置（注意宽度要配置成实际宽度的 2 倍），如图 3-4 所示。



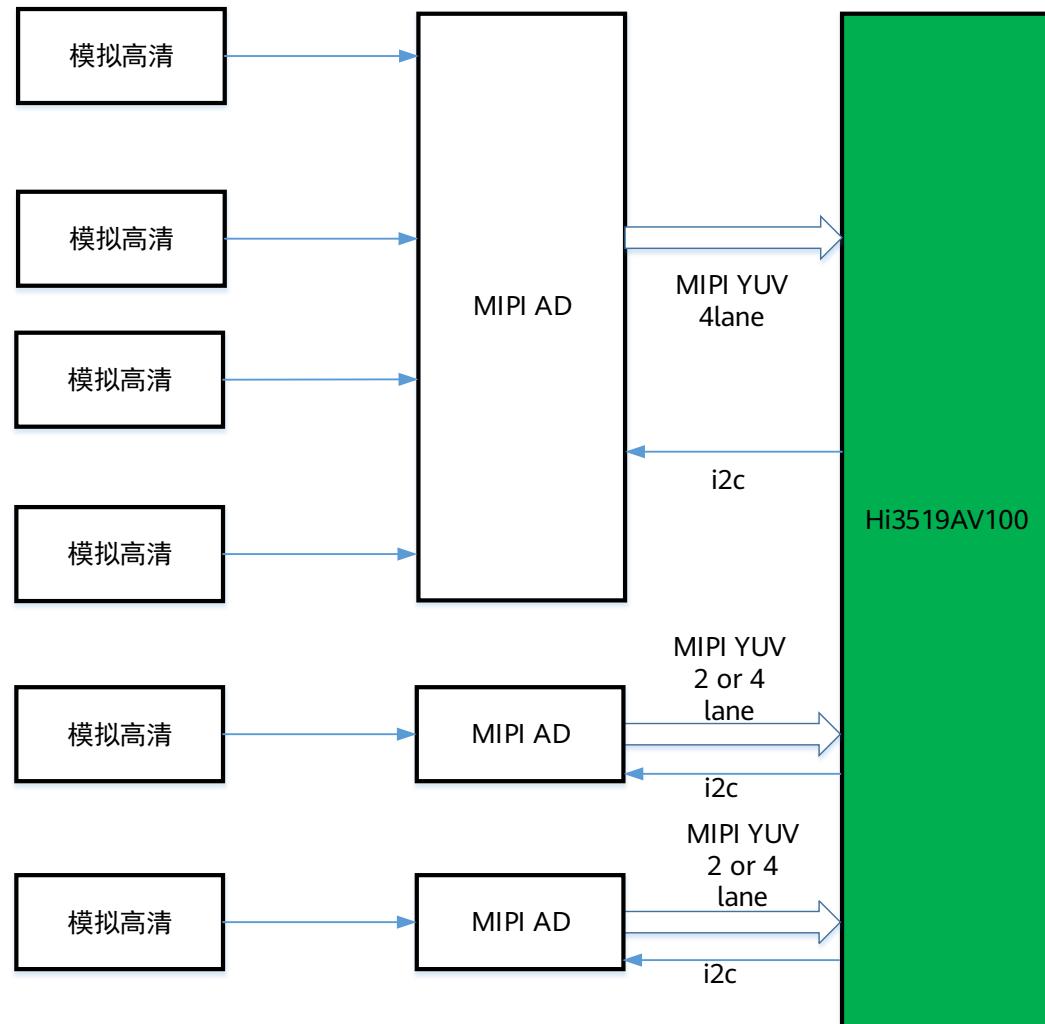
图3-4 VI PIPE 属性配置

```
static VI_PIPE_ATTR_S PIPE_ATTR_1920x1080_RAW8_420_3DNR_RFR =
{
    VI_PIPE_BYPASS_BE, HI_TRUE, HI_FALSE,
    1920*2, 1080,
    PIXEL_FORMAT_RGB_BAYER_8BPP,
    COMPRESS_MODE_NONE,
    DATA_BITWIDTH_8,
    HI_TRUE,
    {
        PIXEL_FORMAT_YVU_SEMIPLANAR_420,
        DATA_BITWIDTH_8,
        VI_NR_REF_FROM_RFR,
        COMPRESS_MODE_NONE
    },
    HI_FALSE,
    {-1, -1}
};
```



3.6.2 1个 MIPI AD 4路复合 + 2个 MIPI AD 单路

图3-5 6路YUV解决方案图



- 前端 MIPI 对接 AD 4 路复合，在 VI PIPE2~5 YUV 写出。
- 另外两路 MIPI 接收在 MIPI 按 yuv422 packed 格式接收，在 VI PIPE 0~1 按 16bit raw 写出，通过 iovec dma 进行格式转换成 semiplanar 422 (格式转换请见 iovec sample)，配置如下：
 - MIPI 配置，如图 3-6 所示。



图3-6 MIPI 配置

```
combo_dev_attr_t MIPI_4lane_CHN0_2M_SP420_ATTR =  
{  
    .devno = 0,  
    .input_mode = INPUT_MODE_MIPI,  
    .data_rate = MIPI_DATA_RATE_X1,  
    .img_rect = {0, 0, 1920, 1080},  
  
    {  
        .mipi_attr =  
        {  
            DATA_TYPE_YUV422_PACKED,  
            HI_MIPI_WDR_MODE_NONE,  
            {0, 1, -1, -1, -1, -1, -1, -1},  
        }  
    }  
};
```

VI 设备属性配置，如图 3-7 所示。

图3-7 VI DEV 设备属性配置

```
VI_DEV_ATTR_S DEV_ATTR_MIPI_SP422_2M_BASE =  
{  
    VI_MODE_MIPI,  
    VI_WORK_MODE_1Multiplex,  
    {0xFFFF0000, 0x000000},  
    VI_SCAN_PROGRESSIVE,  
    {-1, -1, -1, -1},  
    VI_DATA_SEQ_UVYY,  
  
    /*port_vsync  port_vsync_neg      port_hsync      port_hsync_neg      */  
    VI_VSYNC_PULSE, VI_VSYNC_NEG_LOW, VI_HSYNC_VALID_SIGNAL, VI_HSYNC_NEG_HIGH, VI_VSYNC_VALID_SIGNAL, VI_VSYNC_VALID_NEG_HIGH,  
  
    /*hsync_hfb   hsync_act     hsync_hhb*/  
    {0, 1280, 0,  
     /*vsync0_vhb vsync0_act vsync0_hhb*/  
     0, 720, 0,  
     /*vsync1_vhb vsync1_act vsync1_hhb*/  
     0, 0, 0},  
    VI_DATA_TYPE_YUV,  
    HI_FALSE,  
    {1920, 1080},  
    {  
        {  
            {1920, 1080},  
        },  
        {  
            VI_REPHASE_MODE_NONE,  
            VI_REPHASE_MODE_NONE  
        }  
    },  
    {  
        WDR_MODE_NONE,  
        1080  
    },  
    DATA_RATE_X1
```

- PIPE 属性配置，如图 3-8 所示。



图3-8 VI PIPE 属性配置

```
static VI_PIPE_ATTR_S PIPE_ATTR_1920x1080_SP420_3DNR_RFR =
{
    VI_PIPE_BYPASS_BE, HI_FALSE, HI_FALSE,
    1920, 1080,
    PIXEL_FORMAT_RGB_BAYER_16BPP,
    COMPRESS_MODE_NONE,
    DATA_BITWIDTH_8,
    HI_FALSE,
    {
        PIXEL_FORMAT_YVU_SEMIPLANAR_420,
        DATA_BITWIDTH_8,
        VI_NR_REF_FROM_RFR,
        COMPRESS_MODE_NONE
    },
    HI_FALSE,
    {-1, -1}
};
```



4 FISHEYE 功能

4.1 鱼眼矫正功能

4.1.1 Hi3559AV100/Hi3556AV100 如何实现多于 4 宫格鱼眼矫正功能

【现象】在系统的绑定通路中，通过 VI/VPSS 的接口 HI_MPI_VI_SetExtChnFisheye / HI_MPI_VPSS_SetExtChnFisheye 只能实现 2~4 宫格的合成，不能实现多于 4 宫格的合成。

【分析】VI/VPSS 的接口实现多于 4 宫格合成会让系统变得复杂，而且目前芯片 Hi3559AV100/Hi3556AV100 的 GDC 的性能不足。

【解决】用户从 VI/VPSS 模块获取图像，调用 HI_MPI_GDC_AddCorrectionTask 来做鱼眼矫正后再送回到系统通路（VI/VPSS/VO 模块）中，其中的 LMF 参数的设置也可调用 FISHEYE 接口 HI_MPI_GDC_SetConfig 来实现。

【注意】请参照 FISHEYE 的 sample 的第 4 个，GDC 的性能会出现不足。



5 LDC 功能

5.1 畸变矫正功能

5.1.1 VI 和 VPSS LDC 对比

VI/VPSS 离线时：VI 和 VPSS 的 LDC 相关接口都能实现畸变校正功能。由于 LDC 会改变图像的噪声形态，因此先进行 3DNR 处理，再进行 LDC 处理（即使用 VPSS LDC）获得的图像效果较好。但是，如果在 VPSS 校正的话，如业务场景中 VPSS 有多个通道输出的话，会在每个通道都会调用 VGS/GDC 进行 LDC 校正，相对于 VI 做 LDC 会对 VGS/GDC 性能、内存等产生影响。对于想获取更好的图像效果，而且性能、内存不紧张的场景下，推荐使用 VPSS LDC，否则推荐使用 VI 模块进行 LDC 校正。

VI/VPSS 在线时：只能使能 VPSS 的 LDC 校正功能，VI 的 LDC 功能不可使用。



6 音频

6.1 PC 如何播放由 Hisilicon (Shanghai) 编码的音频码流

6.1.1 PC 如何播放由 Hisilicon (Shanghai) 编码的音频

G711/G726/ADPCM 码流

【现象】

由 Hisilicon 编码的音频 G711/G726/ADPCM 码流不能直接用 PC 端软件播放。

【分析】

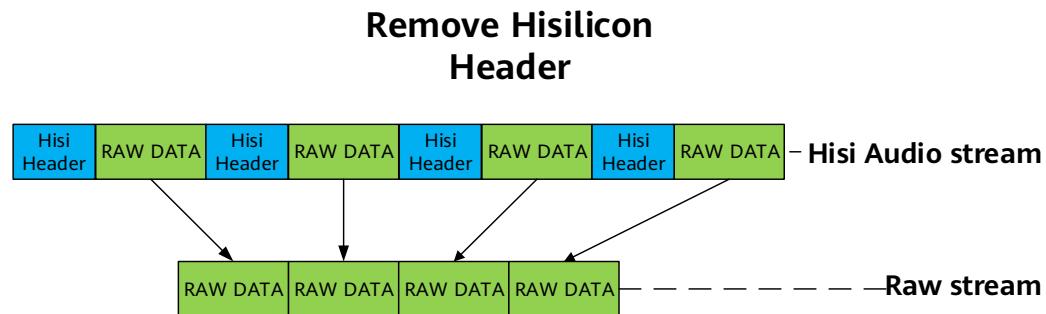
由 Hisilicon 编码的音频码流，会在每一帧数据前添加一个上海海思语音帧头详见：
《HiMPP 媒体处理软件 V4.0 开发参考》音频章节 9.2.2.3 “上海海思语音帧结构”。

【解决】

PC 端软件播放时，需要先去除每一帧数据前的上海海思语音帧头得到裸码流后，再添加 WAV Header 进行播放。去除上海海思语音帧头的操作如图 6-1 所示。



图6-1 去除上海海思语音帧头示意图



去除上海海思语音帧头参考代码：

```
int HisiVoiceGetRawStream(short *Hisivoicedata, short *outdata, int hisisamplelen)
{
    int len = 0, outlen = 0;
    short *copyHisidata, *copyoutdata;
    int copysamplelen = 0;
    copysamplelen = hisisamplelen;
    copyHisidata = Hisivoicedata;
    copyoutdata = outdata;
    while(copysamplelen > 2)
    {
        len = copyHisidata[1]&0x00ff;
        copysamplelen -= 2;
        copyHisidata += 2;
        if(copysamplelen < len)
        {
            break;
        }
        memcpy(copyoutdata, copyHisidata, len * sizeof(short));
        copyoutdata += len;
        copyHisidata += len;
        copysamplelen -= len;
        outlen += len;
    }
    return outlen;
}
```



说明

- ADPCM 格式中, ADPCM_DVI4 和 ADPCM_ORG_DVI4 适用网络 RTP 传输使用, 不能通过该方式在 PC 客户端上播放, 详情请参考 rfc35551 标准。
- 添加 WAV Header 的操作略, 客户可以根据 WAV Header 标准[参考链接 1](https://msdn.microsoft.com/en-us/library/dd390970(v=vs.85).aspx) 和[参考链接 2](http://www.moon-soft.com/program/FORMAT/windows/wavec.htm) 进行添加。

参考链接 1: [https://msdn.microsoft.com/en-us/library/dd390970\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/dd390970(v=vs.85).aspx)

参考链接 2: <http://www.moon-soft.com/program/FORMAT/windows/wavec.htm>

6.2 Hisilicon (Shanghai) 如何播放标准的音频码流

6.2.1 Hisilicon (Shanghai) 如何播放标准的音频

G711/G726/ADPCM 码流

【现象】

Hisilicon (Shanghai) 不能直接播放标准的音频 G711/G726/ADPCM 码流。

【分析】

Hisilicon (Shanghai) 为了兼容上一代芯片, 要求在音频裸码流每帧数据前添加上海海思语音帧头才能播放。

【解决】

Hisilicon 播放标准的音频 G711/G726/ADPCM 码流时, 需要先获取 RAW 流数据, 再根据每帧数据长度 PerSampleLen 添加上海海思语音帧头才能播放。

1. 获取 RAW 流数据:

如果码流添加了 WAV Header, 则需要先去除 WAV Header。

2. 获取每帧数据长度 PersampleLen(计量单位为 short 型):

表6-1 每帧数据长度

编码格式	每帧数据长度	备注
G711	N*40	N 为[1,5]的任意正整数
G726-16kbps	N *10	N 为[1,5]的任意正整数
G726-24kbps	N *15	N 为[1,5]的任意正整数



G726-32kbps	N *20	N 为[1,5]的任意正整数
G726-40kbps	N *25	N 为[1,5]的任意正整数
IMA ADPCM	每块字节数/2	每块字节数为 IMA ADPCM 的每块编码数据字节数，对应 IMA ADPCM WAV Header 的 nblockalign (0x20-0x21, 2bytes)

说明

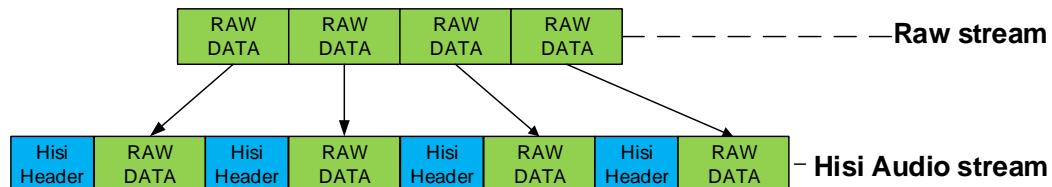
- ADPCM 格式中，仅支持 IMA ADPCM 格式，每采样点比特数(wbitspersample)只支持 4。
- 如果 ADPCM 码流添加了 WAV Header，可以从 WAV Header 中获得每块字节数信息；如果为 ADPCM 裸码流，则需要从码流提供方获取每块字节数信息。
- 编码格式仅支持单声道编码格式。

3. 添加上海海思语音帧头：

添加上海海思语音帧头的操作如图 6-2 所示：

图6-2 添加上海海思语音帧头示意图

Add Hisilicon Header



添加上海海思语音帧头参考代码：

```
int HisiVoiceAddHisiHeader(short *inputdata, short *Hisivoicedata, int PersampleLen,int inputsamplelen)
{
    int len = 0, outlen = 0;
    short HisiHeader[2];
    short *copyHisidata, *copyinputdata;
    int copysamplelen = 0;
    HisiHeader[0] = (short)(0x001<<8) & (0x0300);
```



```
HisiHeader[1] = PersampleLen & 0x00ff;
copysamplelen = inputsamplelen;
copyHisidata = Hisivoicedata;
copyinputdata = inputdata;
while(copysamplelen >= PersampleLen)
{
    memcpy(copyHisidata, HisiHeader, 2 * sizeof(short));
    outlen += 2;
    copyHisidata += 2;
    memcpy(copyHisidata, copyinputdata, PersampleLen * sizeof(short));
    copyinputdata += PersampleLen;
    copyHisidata += PersampleLen;
    copysamplelen -= PersampleLen;
    outlen += PersampleLen;
}
return outlen;
}
```

6.3 为什么使能 VQE 后会有高频部分缺失

6.3.1 为什么使能 VQE 后会有高频部分缺失

【现象】

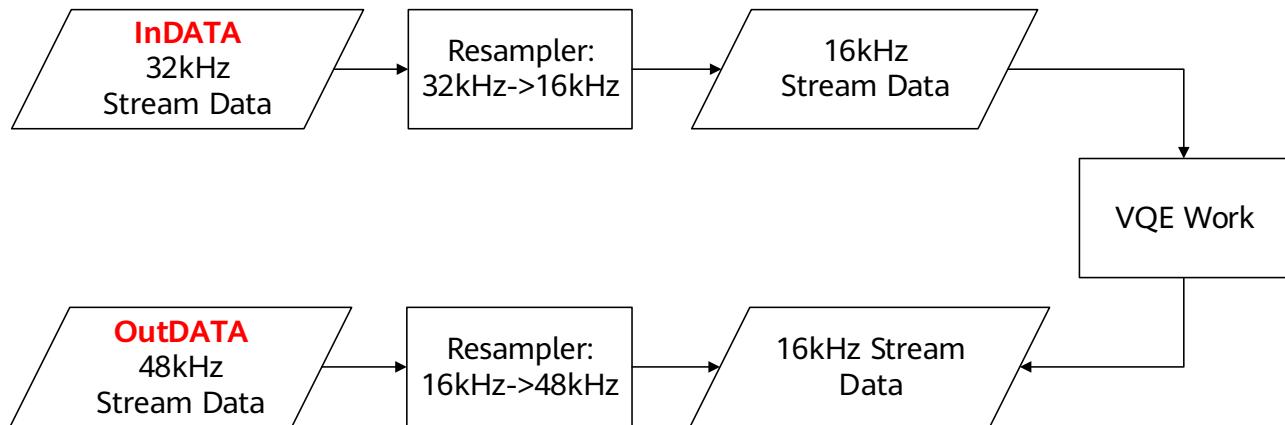
配置 AI 采样率(AISampleRate)为 32kHz，使能 VQE 功能，配置 VQE 工作采样率(VQEWorkSampleRate)为 16kHz；使能重采样功能，配置输出采样率(ResOutSampleRate)为 48kHz，分析输出序列，发现 8kHz 以上高频部分缺失。

【分析】

HiVQE 实际工作采样率仅支持 8kHz 和 16kHz，考虑到客户需要，Hisilicon 在 VQE 封装了重采样层以支持 8kHz 到 48kHz 的任意标准采样率处理。当客户配置 AISampleRate = 48kHz，VQEWorkSampleRate = 16kHz，ResOutSampleRate = 48kHz 时，重采样层会先将数据由 32kHz 重采样到 16kHz，经过 VQE 处理后，再由 16kHz 重采样到 48kHz 输出。流程如图 6-3 所示。



图6-3 VQE 处理流程图



在进行 Resampler: 32kHz->16kHz 时，造成了 8kHz 以上的高频部分缺失。

在当前应用场景中，输出序列的频段信息如下：

1. 不使能 VQE，不使能重采样：按照 $\text{AI Sample Rate}/2$ 输出信息频段。如配置 AI 采样率为 48kHz，输出信息频段为 0 – 24kHz。
2. 使能重采样，不使能 VQE：取 $\min(\text{AI Sample Rate}, \text{ResOutSampleRate}) / 2$ 输出信息频段。如配置 AI 采样率 16kHz，输出采样率 32kHz， $\min(\text{AI Sample Rate}, \text{ResOutSampleRate}) = 16\text{kHz}$ ，则输出信息频段为 0 – 8kHz。
3. 使能 VQE，不使能重采样：取 $\min(\text{AI Sample Rate}, \text{VQEWorkSampleRate}) / 2$ 输出信息频段。如配置 AI 采样率为 32kHz，VQEWorkSampleRate 为 16kHz， $\min(\text{AI Sample Rate}, \text{VQEWorkSampleRate}) = 16\text{kHz}$ ，则输出信息频段为 0 – 8kHz。
4. 使能 VQE，使能重采样：取 $\min(\text{AI Sample Rate}, \text{VQEWorkSampleRate}, \text{ResOutSampleRate}) / 2$ 输出信息频段。如配置 AI 采样率为 32kHz，VQEWorkSampleRate 为 16kHz，重采样模块输出采样率为 48kHz，则 $\min(\text{AI Sample Rate}, \text{VQEWorkSampleRate}, \text{ResOutSampleRate}) = 16\text{kHz}$ ，输出信息频段为 0 – 8kHz。



说明

- 配置采样率后，输出信息频段为采样率的 1/2。
- 当前支持 8kHz 到 48kHz 标准采样率，分别为：8kHz, 11.025kHz, 12kHz, 16kHz, 22.05kHz, 32kHz, 44.1kHz, 48kHz。
- AO 处理流程类同 AI 处理流程。



6.4 音频内置 CODEC 输出(AO 输出)出现幅频响应异常

【现象】

测试 AUDIO CODEC (DAC) 输出(AO 输出)幅频响应，出现 2KHz 以上频段幅频响应严重衰减。

【分析】

这个是 AUDIO CODEC 控制寄存器（参考文档：《Hi35XX xx 型 xx IP Camera SoC 用户指南.pdf》）的 dacl_deemph 和 dacr_deemph 位开启去加重导致的。去加重是相对于[预加重](#)而言的，是对预加重的修正，如果输入给 AO 通道是经过预加重的音频信号，那么开启去加重功能可以恢复到正常频响；如果输入给 AO 通道的音频信号没有预加重，此时 dacl_deemph 和 dacr_deemph 位开启去加重（不为 00），就会对幅频响应产生影响。本次测试是在关闭 HIVQE 功能下测试的。测试时，输入给 AO 通道的数据没有预加重，而 AUDIO CODEC 控制寄存器的 dacl_deemph 和 dacr_deemph 位未关闭（都不为 00），故出现此问题。

【解决】

在 AI 通道，AUDIO CODEC 控制寄存器是没有开设预加重功能的，所以 AUDIO CODEC 控制寄存器的 dacl_deemph 和 dacr_deemph 位默认是需要关闭的，都配置为 00。预加重和去加重功能是匹配成对出现的，使用时需要注意。

【注意事项】

如果 AUDIO CODEC 控制寄存器没有 dacl_deemph 和 dacr_deemph 位则表示不支持去加重功能，此时需按默认配置使用，不允许额外配置。

6.5 静态库注册功能

- 对于支持 VQE 静态库注册功能的芯片，可根据实际应用场景选用所需的声音质量增强及重采样模块，再通过 HI_MPI_AUDIO_RegisterVQEModule 接口向音频系统注册选定的模块，具体使用方法参见《HiMPP 媒体处理软件 V4.0 开发参考》“音频”章节。
- 对于支持 AAC 静态库注册功能的芯片，可根据实际应用场景选择是否使用 SBRENC、SBRDEC 模块。当使用 EAAC 或者 EAACPLUS 编解码类型时，须在注册编解码器之前进行 SBRENC、SBRDEC 功能模块的静态注册，具体使用方法参见《音频组件 API 参考.pdf》。



6.6 加载内置 CODEC 模块出现 pop 音的解决方法

【现象】

在加载内置 codec 模块时，单板的音频输出端存在 pop 音。

【分析】

以 Hi3516CV500 为例，其 demo 板功放的解 mute 操作在 sys_config 模块中实现，对应函数为 ampunmute，而内置 codec 模块的加载顺序在 sys_config 模块之后，因此在 codec 模块加载时功放已解 mute，从而导致 pop 音。

【解决】

以 Hi3516CV500 为例，将 sys_config 模块中功放解 mute 的操作移至 load3516cv500 脚本中的 insert_audio 操作之后。

6.7 如何对多声道的音频数据进行交织处理

【现象】

上海海思音频帧的不同声道数据地址是独立的，需要存放到同一个 PCM 文件。

【分析】

当音频的声道数多于一个时，音频数据的存放有两种格式，即交织的 (interleave) 和非交织的 (non-interleave)。交织的是指同一个采样点的多声道数据依次放在一起；非交织的是指先放第一声道的所有采样点的数据，再放第二声道的所有采样点的数据，依次类推，直至全部声道存放完毕。

【解决】

以最常见的 2 声道为例，交织后的数据排布为 L1/R1/L2/R2/.../Ln-1/Rn-1/Ln/Rn (L 表示左声道，R 表示右声道，数字表示第几个采样点)。

2 声道音频数据交织的参考代码：

```
static hi_void interleave_16bit(hi_s16 *dest, hi_s16 *src_left, hi_s16 *src_right, hi_u32 samples)
{
    hi_u32 i;

    if ((dest == HI_NULL) || (src_left == HI_NULL) || (src_right == HI_NULL)) {
        return;
    }

    for (i = 0; i < samples; i++) {
        dest[i] = src_left[i];
        dest[i + samples] = src_right[i];
    }
}
```



```
}

for (i = 0; i < samples; i++) {
    dest[2 * i] = *src_left; /* 2: 2chn */
    dest[2 * i + 1] = *src_right; /* 2: 2chn */
    src_left++;
    src_right++;
}
}
```

6.8 如何对多声道的音频数据进行混音处理

【现象】

多个声道的音频数据需要混音到单声道进行输出。

【分析】

当只有一个物理输出通道时，不同来源的声音要整合到同一个音轨进行播放。

【解决】

以最常见的 2 声道为例，可以考虑采用线性叠加的方式实现混音。需要注意的是，2 声道线性叠加可能会导致溢出，可在混音前对左右声道分别进行限幅。

2 声道音频数据混音的参考代码：

```
static hi_void remix_16bit(hi_s16 *dest, hi_s16 *src_left, hi_s16 *src_right, hi_u32 samples)
{
    hi_u32 i;

    if ((dest == HI_NULL) || (src_left == HI_NULL) || (src_right == HI_NULL)) {
        return;
    }

    for (i = 0; i < samples; i++) {
        dest[i] = (*src_left) + (*src_right);
        src_left++;
        src_right++;
    }
}
```



{}

6.9 Musl 版本的 sample 链接 bcd 库后执行时会出现段错误 异常

【现象】

对于使用 musl 库的 Linux 系统，在增加-pie 链接选项之后，使用动态方式链接 bcd 库的可执行文件在运行时会出现段错误。

【分析】

对于使用 musl 库的 Linux 系统，其对动态链接的兼容性不如 glibc 库，导致可执行文件在动态链接 bcd 库后，运行时会出现段错误的问题，需要改成静态链接的方式。

【解决】

在链接成可执行文件的时候增加-static 链接选项，确保使用静态链接的方式。另外，可以选择不使用-pie 链接选项进行规避。

【注意事项】

sample 中的默认链接方式为动态链接，需要显示指定-static 链接选项才会执行静态链接。



7 低功耗

7.1 低功耗模块动态调频可能频繁调频的问题

【现象】加载低功耗模块 hi35xx_pm.ko 后，策略使用动态调频策略，例如 ondemand 策略，可能会看到频繁的调节频率的现象。

【分析】版本 linux kernel 默认使用 HZ 为 100，也即为 10ms 调度统计，统计时间粒度较粗，导致统计精度不够，如此 CPU 负载统计波动会比较大，linux 会在每个统计周期内根据 cpu 的负载统计进行动态调频调压，因此可能导致低功耗模块频繁的调频。

【解决】可以修改 kernel HZ 为 1000，如此可以提高统计精度，也可以修改增大低功耗模块的统计周期来改善这种情况，例如，修改 ondemand 策略下的统计周期为 1s，我们可以执行如下命令(统计周期单位为 us)：

```
echo 1000000 >/sys/devices/system/cpu/cpufreq/ondemand/sampling_rate.
```



8 LCD 屏幕调试

8.1 支持哪些 LCD 屏幕

LCD 屏是否支持主要看 LCD 的接口类型是否和芯片的能力匹配。

芯片支持的 LCD 接口类型参考文档《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频输出”章节。

8.2 LCD 屏幕调试顺序

8.2.1 确认管脚复用配置

用户需要确认和 LCD 的硬件连接管脚都正确地配置成 VO 相关功能，并且管脚的驱动能力配置成适当的值。具体的管脚配置请参考各芯片的《Hi35XX xx 型 xx IP Camera SoC 用户指南》。此外，还可参考发布包里的 sys_config.c 文件。

8.2.2 确认用户时序

目前针对每一种接口类型，SDK 中仅提供一种时序，如 VO_INTF_LCD_8BIT 接口仅提供时序 VO_OUTPUT_320X240_60，而且这种时序仅保证对特定的一款 LCD 屏有效，如 VO_OUTPUT_320X240_60 时序只能用于 ota5182 这种驱动 IC 的 LCD 屏。因此，用户在使用调试 LCD 屏幕时大部分需要使用用户时序。

调用 HI_MPI_VO_SetPubAttr 接口设置输出的公共属性时，根据 LCD 的接口类型选择正确的 LCD 接口类型，接口时序选择 VO_OUTPUT_USER，接下来就是根据 LCD 的要求来配置用户时序结构体。

```
typedef struct tagVO_SYNC_INFO_S
```



```
{  
    HI_BOOL bSymm;      /* sync mode(0:timing,as BT.656; 1:signal,as LCD) */  
    HI_BOOL bIop;        /* interlaced or progressive display(0:i; 1:p) */  
    HI_U8 u8Intfb;      /* interlace bit width while output */  
    HI_U16 u16Vact;     /* vertical active area */  
    HI_U16 u16Vbb;      /* vertical back blank porch */  
    HI_U16 u16Vfb;      /* vertical front blank porch */  
    HI_U16 u16Hact;     /* horizontal active area */  
    HI_U16 u16Hbb;      /* horizontal back blank porch */  
    HI_U16 u16Hfb;      /* horizontal front blank porch */  
    HI_U16 u16Hmid;     /* bottom horizontal active area */  
    HI_U16 u16Bvact;    /* bottom vertical active area */  
    HI_U16 u16Bvbb;     /* bottom vertical back blank porch */  
    HI_U16 u16Bvfb;     /* bottom vertical front blank porch */  
    HI_U16 u16Hpw;      /* horizontal pulse width */  
    HI_U16 u16Vpw;      /* vertical pulse width */  
    HI_BOOL bldv;        /* inverse data valid of output */  
    HI_BOOL blhs;        /* inverse horizontal synch signal */  
    HI_BOOL blvs;        /* inverse vertical synch signal */  
} VO_SYNC_INFO_S;
```

各参数说明，如表 8-1 所示。

表8-1 参数说明

参数名称	说明
bSymm	同步模式，LCD 选择 1，表示信号同步。
bIop	0 为隔行，1 为逐行，LCD 一般配置 1。
u8Intfb	无效参数，可以忽略。
u16Vact	垂直有效区，隔行输出时表示顶场垂直有效区。单位：行。
u16Vbb	垂直消隐后肩，隔行输出时表示顶场垂直消隐后肩。单位：行。
u16Vfb	垂直消隐前肩，隔行输出时表示顶场垂直消隐前肩。单位：行。
u16Hact	水平有效区。单位：像素。
u16Hbb	水平消隐后肩。单位：像素。



参数名称	说明
u16Hfb	水平消隐前肩。单位：像素。
u16Hmid	底场垂直同步有效像素值。
u16Bvact	底场垂直有效区，隔行时有效。单位：行。
u16Bvbb	底场垂直消隐后肩，隔行时有效。单位：行。
u16Bvfb	底场垂直消隐前肩，隔行时有效。单位：行。
u16Hpw	水平同步信号的宽度。单位：像素。
u16Vpw	垂直同步信号的宽度。单位：行。
bldv	数据有效信号的极性。配置 0 为高有效，配置 1 为低有效。
blhs	水平有效信号的极性，配置 0 为高有效，配置 1 为低有效。
blvs	垂直有效信号的极性，配置 0 为高有效，配置 1 为低有效。

用户时序的配置需要用户自行查找 LCD 的文档来配置。并且要注意各个值单位和要求的一致。

8.2.3 配置设备帧率

用户时序下，需要用户调用 HI_MPI_VO_SetDevFrameRate 接口来配置设备帧率。该接口的使用方法参考文档《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频输出”章节。

8.2.4 确认时钟信息

用户除了需要配置用户时序和设备帧率。还需要通过 HI_MPI_VO_SetUserIntfSyncInfo 接口来配置用户时序的时钟、分频比等信息。

具体的接口使用方法请参考文档《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频输出”章节。下面以 Hi3519AV100 芯片为例，简单描述该接口的配置方法：

用户时序的时钟源可以选择来自 PLL 或者 LCD 分频得到。如果选择 PLL 为时钟源，则需要配置 PLL 的 u32Fbdv、u32Frac、u32Refdiv、u32Postdiv1 和 u32Postdiv2 参数，这 5 个参数的意义可以参考《Hi35XX xx 型 xx IP Camera SoC 用户指南》中的“系统”章节中对 PLL 配置的描述，合理配置这 5 个参数可以得到想要的时钟；如果选



择 LCD 分频器为时钟源，需要配置 u32LcdMClkDiv 参数，参考《Hi35XX xx 型 xx IP Camera SoC 用户指南》中的“系统”章节中对 LCD 时钟寄存器的描述即可。

HI_MPI_VO_SetUserIntfSyncInfo 接口中的 bClkReverse 参数可以用于对 VDP 的时钟进行反向，用于调节 VDP 时钟相位。

在 HI_MPI_VO_SetUserIntfSyncInfo 接口的配置中，还需要确认分频比的配置，分频比指的是 VDP 输出时钟（芯片输出时钟）与 HD 通道时钟的比值。由于 HD 通道中，一个时钟节拍输出一个像素，而在 LCD 屏中，则可能需要多个时钟节拍来构造一个像素。

- 如某款 8bit 串行的 LCD 屏，需要的数据序列为 RGB 的序列，即一个像素需要 3 个时钟节拍来传输 R、G、B 三个数据，这时候分频比应该配置 3 分频。
- 如某款 16bit 串行的 LCD 屏，一个时钟节拍构造一个像素，则应该配置 1 分频。



9 VO

9.1 VO 用户时序如何配置

9.1.1 时序结构配置

在 HI_MPI_VO_SetPubAttr 接口中配置 pstPubAttr-> enIntfSync 为 VO_OUTPUT_USER，然后配置 stSyncInfo 结构体。关于 stSyncInfo 结构体中各参数的解释如下：

```
typedef struct tagVO_SYNC_INFO_S
{
    HI_BOOL    bSymm;      /* sync mode(0:timing,as BT.656; 1:signal,as LCD) */
    HI_BOOL    bIop;       /* interlaced or progressive display(0:i; 1:p) */
    HI_U8     u8Intfb;    /* interlace bit width while output */
    HI_U16    u16Vact;    /* vertical active area */
    HI_U16    u16Vbb;     /* vertical back blank porch */
    HI_U16    u16Vfb;     /* vertical front blank porch */
    HI_U16    u16Hact;    /* horizontal active area */
    HI_U16    u16Hbb;     /* horizontal back blank porch */
    HI_U16    u16Hfb;     /* horizontal front blank porch */
    HI_U16    u16Hmid;    /* bottom horizontal active area */
    HI_U16    u16Bvact;   /* bottom vertical active area */
    HI_U16    u16Bvbb;    /* bottom vertical back blank porch */
    HI_U16    u16Bvfb;    /* bottom vertical front blank porch */
    HI_U16    u16Hpw;     /* horizontal pulse width */
    HI_U16    u16Vpw;     /* vertical pulse width */
    HI_BOOL    bldv;       /* inverse data valid of output */
    HI_BOOL    blhs;       /* inverse horizontal synch signal */
```



```
    HI_BOOL blvs;      /* inverse vertical synch signal */  
} VO_SYNC_INFO_S;
```

各参数说明，如表 9-1 所示。

表9-1 参数说明

参数名称	说明
bSynm	同步模式，LCD 选择 1，表示信号同步。
blop	0 为隔行，1 为逐行，LCD 一般配置 1。
u8Intfb	无效参数，可以忽略。
u16Vact	垂直有效区，隔行输出时表示顶场垂直有效区。单位：行。
u16Vbb	垂直消隐后肩，隔行输出时表示顶场垂直消隐后肩。单位：行。
u16Vfb	垂直消隐前肩，隔行输出时表示顶场垂直消隐前肩。单位：行。
u16Hact	水平有效区。单位：像素。
u16Hbb	水平消隐后肩。单位：像素。
u16Hfb	水平消隐前肩。单位：像素。
u16Hmid	底场垂直同步有效像素值。
u16Bvact	底场垂直有效区，隔行时有效。单位：行。
u16Bvbb	底场垂直消隐后肩，隔行时有效。单位：行。
u16Bvfb	底场垂直消隐前肩，隔行时有效。单位：行。
u16Hpw	水平同步信号的宽度。单位：像素。
u16Vpw	垂直同步信号的宽度。单位：行。
bldv	数据有效信号的极性。配置 0 为高有效，配置 1 为低有效。
blhs	水平有效信号的极性，配置 0 为高有效，配置 1 为低有效。
blvs	垂直有效信号的极性，配置 0 为高有效，配置 1 为低有效。



下面以 Hi3519AV100 上配置 384*288P@25fps 的用户时序为例， stSyncInfo 的配置如下：

```
pstPubAttr->stSyncInfo.bSynm = 0;
```

```
pstPubAttr->stSyncInfo.blop = 1;
```

```
pstPubAttr->stSyncInfo.u8Intfb = 0;
```

```
pstPubAttr->stSyncInfo.u16Vact = 288;
```

```
pstPubAttr->stSyncInfo.u16Vbb = 200;
```

```
pstPubAttr->stSyncInfo.u16Vfb = 112;
```

```
pstPubAttr->stSyncInfo.u16Hact = 384;
```

```
pstPubAttr->stSyncInfo.u16Hbb = 300;
```

```
pstPubAttr->stSyncInfo.u16Hfb = 216;
```

```
pstPubAttr->stSyncInfo.u16Hmid = 1;
```

```
pstPubAttr->stSyncInfo.u16Bvact = 1;
```

```
pstPubAttr->stSyncInfo.u16Bvbb = 1;
```

```
pstPubAttr->stSyncInfo.u16Bvfb = 1;
```

```
pstPubAttr->stSyncInfo.u16Hpw = 4;
```

```
pstPubAttr->stSyncInfo.u16Vpw = 5;
```

```
pstPubAttr->stSyncInfo.bldv = 0;
```

```
pstPubAttr->stSyncInfo.blhs = 0;
```

```
pstPubAttr->stSyncInfo.blvs = 0;
```

此时，VO 的时钟配置应该为 $(384+300+216) * (288+200+112) * 25 = 13500000$ ，即 VO 的时钟应该配置为 13.5M。

计算公式为：

$(\text{有效宽} + \text{水平后消隐} + \text{水平前消隐}) * (\text{有效高} + \text{垂直后消隐} + \text{垂直前消隐}) * \text{帧率} = \text{时钟}$ 。



9.1.2 时钟大小配置

通过 HI_MPI_VO_SetUserIntfSyncInfo 接口来配置用户时序的时钟、分频比等信息。

具体的接口使用方法请参考文档《HiMPP 媒体处理软件 V4.0 开发参考》中的“视频输出”章节。

9.2 画面切换

9.2.1 通道属性发生变化

画面切换：通道在显示状态下，其显示位置和大小发生变化。

9.2.2 建议的实现方式

通道已经使能或显示的状态下，凡涉及到修改该通道属性（调用 HI_MPI_VO_SetChnAttr）（假设通道号为 chn-x），建议按照以下步骤完成：

步骤 1 设置通道所在层批处理 begin (HI_MPI_VO_BatchBegin)

步骤 2 隐藏所有通道 (HI_MPI_VO_HideChn)

步骤 3 设置目标通道 chn-x （可以是多个通道）的通道属性 (HI_MPI_VO_SetChnAttr)

步骤 4 显示所有通道 (HI_MPI_VO_ShowChn)

步骤 5 设置通道所在层批处理 end (HI_MPI_VO_BatchEnd)

----结束

可参考流程：

```
/*
 * n --> m : change n chns to m chns.
 * 设置目标通道chn-0,chn-1,...,chn-m的通道属性
 */
SetChnMAttr()
{
/* batch begin */
    s32Ret = HI_MPI_VO_BatchBegin(0);
    if (HI_SUCCESS != s32Ret)

```



```
{  
    SAMPLE_PRT("HI_MPI_VO_BatchBegin(0) failed!\n");  
}  
/* hide all n chns */  
for(i=0;i<n;i++)  
{  
    s32Ret = HI_MPI_VO_HideChn(0, i);  
    if (HI_SUCCESS != s32Ret)  
    {  
        SAMPLE_PRT("HI_MPI_VO_HideChn(0,%d) failed!\n",i);  
    }  
}  
/* change all m chns's attr */  
for(j=0;j<m;j++)  
{  
    s32Ret = HI_MPI_VO_SetChnAttr(0, j, &stSetChnAttr);  
    if (HI_SUCCESS != s32Ret)  
    {  
        SAMPLE_PRT("HI_MPI_VO_SetChnAttr(0,%d) failed!\n",j);  
    }  
}  
  
/* enable all m chns */  
for(j=0;j<m;j++)  
{  
    s32Ret = HI_MPI_VO_EnableChn(0, j);  
    if (HI_SUCCESS != s32Ret)  
    {  
        SAMPLE_PRT("HI_MPI_VO_EnableChn (0,%d) failed!\n",j);  
    }  
}  
  
/* show all m chns*/  
for(i=0;i<n;i++)  
{  
    s32Ret = HI_MPI_VO_ShowChn(0, i);  
    if (HI_SUCCESS != s32Ret)
```



```
{  
    SAMPLE_PRT("HI_MPI_VO_ShowChn(0,%d) failed!\n",i);  
}  
}  
/* batch end */  
s32Ret = HI_MPI_VO_BatchEnd(0);  
if (HI_SUCCESS != s32Ret)  
{  
    SAMPLE_PRT("HI_MPI_VO_BatchEnd(0) failed!\n");  
}  
}
```

9.3 视频同步方案

视频同步是指一个芯片的不同 VO 设备或者不同芯片的 VO 设备实现视频同步输出。

视频同步场景一般是多路切分的解码经过 VPSS 再送给多个 VO 设备进行拼接，为了保证拼接效果，需要对各 VO 设备进行视频同步操作。

9.3.1 实现原理

视频同步方案的基本实现原理是保证对 VO 发送视频帧的同步以及 VO 输出时钟的同步。

- 时钟同步：

通过模块参数 bDevClkExtEn 控制，在系统初始化之前置此模块参数为 1（默认为 0），代表 VO 设备的接口时钟由用户自己配置，在业务启动后，用户可以采用关闭各 VO 设备时钟再开时钟的方式保证各设备时钟的同步输出。

- 发送帧同步：

hi_user 驱动提供了对 VO 设备中断的响应函数，用户可以藉此设置监听响应机制，等到中断上报后再对 VO 设备做发送视频帧操作，用户可在此基础上自行增加一些同步发送帧机制。

- 设备开关：

在时钟同步的基础上增加了对设备开关的外部调用函数 VOU_DRV_EnableDev/VOU_DRV_DisableDev（需要保证接口时钟开启的时候进行操作），实现对各设备的同时显示和关闭。



调用者应有的声明格式：

- `extern void VOU_DRV_EnableDev(int VoDev)`
- `extern void VOU_DRV_DisableDev(int VoDev)`

9.3.2 建议的操作步骤

视频同步方案建议按照以下步骤完成：

步骤 1 业务启动前开启设备接口时钟。

步骤 2 启动业务，系统初始化前配置模块参数 `bDevClkExtEn` 为 1，客户在初始启动 VO 业务时仍然按照 VO 的标准流程调用 MPI 接口实现。

步骤 3 调用 `VOU_DRV_DisableDev` 关闭各 VO 设备。

步骤 4 在解码启动之前，各对 VO 设备接口时钟同时做关闭、开启动作，保证时钟的同步，注意，此处关闭开启时钟仅仅对接口时钟相应的 bit 位进行操作。例如针对 Hi3559AV100，DHD0 的接口时钟对应的是寄存器 `0x12010124[6]`；DHD1 的接口时钟对应的寄存器 `0x12010124[4][7]`。

步骤 5 调用 `VOU_DRV_EnableDev` 启动各设备，启动解码发送帧。

步骤 6 如果长期跑，各 VO 设备之间可能会出现时钟偏差，可在监听到此种情况后重复操作步骤 3、步骤 4、步骤 5。

---结束

须知

此方案仅在 Hi3559AV100 上有效。



10 VENC

10.1 JPEG 量化表配置注意事项

目前的 JPEG 编码，如果配置的 u32Qfactor 过低，会导致编码出来的 JPEG 图片出现偏色等现象，原因是色度的量化步长过大。用户可以通过调用 HI_MPI_VENC_SetJpegParam 接口修改色度的量化表，限制色度的量化步长，避免偏色等现象。具体的 u32Qfactor 与量化表的关系请见 RFC2435 标准。修改色度的量化表有可能会导致 JPEG 图片容量变大，用户需要权衡图像质量和 JPEG 图像容量。

10.2 JPEG 使能从多个源接收图像注意事项

如果同时使能从多个源图像接收图像和低延时，JPEG 的 proc 信息中所统计的 PicRec 可能会出现偏差。使能低延时，为加速 VB 轮转，在接收到大图后就会启动 JPEG 编码，不会等待接收到匹配的小图，如果小图出现丢失的情况，PicRec 就多统计了一次大图。

10.3 低功耗使能注意事项

Venc 提供了 3 种低功耗模式：关闭低功耗、低功耗模式和极低功耗模式。低功耗或极低功耗模式使能后能够降低编码功耗，提高编码性能，但同时会导致图像质量损失，噪声大的场景下可能导致画面不均匀，比如图像边缘出现竖条纹。驱动默认为低功耗模式，能够保证绝大部分场景图像质量的同时，降低功耗。如果出现图像质量上的问题，可以通过调用接口 HI_MPI_VENC_SetModParam 关闭低功耗。



10.4 自编自解规格说明

Hi3516CV500 仅支持解码自身编码出来的码流，支持的 H.264 解码配置如下：

序号	规格点	支持解码配置	不支持解码配置
1	Profile	baseline profile main profile high profile	svc-t
2	Intra 块划分	Intra4 Intra8 Intra16	-
3	Inter 块划分	Inter8x8 Inter16x16	Inter8x16 Inter16x8 Inter4x8 Inter8x4 Inter4x4
4	量化矩阵	-	不支持
5	参考帧	单 P 帧	双 P 帧、B 帧



说明

如客户需要使用转码，可限制 1 个参考帧和 preset (superfast 以及更低的 profile)，使用 ffmpeg 或 x264 进行编码。以下是使用 ffmpeg 4.2.1，win32 版本进行转码的实例：重点关注加粗部分的参数，其他参数用户可以根据需求进行修改：

```
ffmpeg -f rawvideo -pix_fmt yuv420p -s:v 1920x1080 -r 25 -i ffmpeg.yuv -c:v libx264 -  
preset superfast -b 1024k -x264-params bframes=0 ffmpeg1920x1080.h264
```



11

HDMI

Hi3559AV100 HDMI 使用注意事项

Hi3559AV100 硬件 Timer11 及对应的中断源会被 HDMI 占用。请勿使用 Timer11，否则可能导致 HDMI 工作异常。



12 其它

12.1 动态库

12.1.1 为什么使用静态编译方式编译应用程序无法使用动态库

【现象】

客户 A 现行文件系统/执行程序都是使用静态编译方式编译，以至于不能使用版本发布的动态库。

【分析】

当前 ARM-Linux-GCC 提供了 3 种编译方式，分别是静态编译，动态编译，半静态编译。其中：

- 静态编译(-static -pthread -lrt -ldl)将会将 libc, libpthread, librt, libdl 都编译到执行程序中，这样的编译方式将会不依赖任何系统动态库（即可独立执行），但无法使用动态库系统。
- 动态编译(普通编译)将会采取链接系统库的方式去链接/lib 目录下的系统动态库，这样编译出来的程序需要依赖系统动态库，优点是系统动态库可以被多个可执行程序共用，如/bin 目录下的 busybox, himount 等。
- 半静态编译(-static-libgcc -static-libstdc++ -L. -pthread -lrt -ldl)则会将 gcc 以及 stdc++ 编译到可执行程序中去，其他系统库依然依赖系统动态库。这种编译方式，可以使用动态库系统，但是依然需要在系统目录下放置 libc, libpthread, librt, libdl 等文件。

【解决】



采取动态编译方式，在/lib 目录下放置动态库依赖的系统文件 ld-uClibc.so.0, libc.so.0, libpthread.so.0, librt.so.0, libdl.so.0 即可。

12.1.2 为什么使用 libupvqe.a 和 libdnvqe.a 动态编译时出现重定义

【现象】

客户 B 使用音频组件库的 libupvqe.a 和 libdnvqe.a 编译成一个动态库，编译时发生重定义报错，编译语句为：

```
$(CC) -shared -o $@ -L . -Wl,--whole-archive libupvqe.a libdnvqe.a -Wl,--no-whole-archive
```

【分析】

libupvqe.a 和 libdnvqe.a 中，都使用了一些共同的功能模块，以达到代码重用以及模块化的目的，并可以在编译成 elf 文件时节省文件空间。

在使用静态库编译动态库时，有 3 种方式：

- 直接使用-l 方式编译，编译语句为：

```
$(CC) -shared -o libshare.so -L . -lupvqe -ldnvqe
```

该编译为链接编译，该方式编译生成后的 libshare.so 将不会链入静态库的函数符号；

- 使用-Wl,--whole-archive 编译，编译语句为：

```
$(CC) -shared -o $@ -L . -Wl,--whole-archive libupvqe.a libdnvqe.a -Wl,--no-whole-archive
```

该编译方式能够将静态库中的函数符号都编译到 libshare.so 中，但是这种编译方式存在限制，libupvqe.a 和 libdnvqe.a 中不能有同名的函数；

- 先将.a 库分别拆成.o，再行编译.so，编译语句为：

```
LIB_PATH = ./  
EXTERN_OBJ_DIR = ./EXTERN_OBJ  
LIBUPVQE_NAME = libupvqe.a  
LIBDNVQE_NAME = libdnvqe.a  
EXTERN_OBJ = $(EXTERN_OBJ_DIR)/*.o  
all: pre_mk $(TARGET) pre_clr  
pre_mk:  
    @mkdir -p $(EXTERN_OBJ_DIR);  
    @cp $(LIB_PATH)$(LIBUPVQE_NAME) $(EXTERN_OBJ_DIR);  
    @cd $(EXTERN_OBJ_DIR); $(AR) -x $(LIBUPVQE_NAME);  
    @cp $(LIB_PATH)$(LIBDNVQE_NAME) $(EXTERN_OBJ_DIR);
```



```
@cd $(EXTERN_OBJ_DIR); $(AR) -x $(LIBDNVQE_NAME);  
$(TARGET):  
    #$(CC) -shared -o $@ -L libupvqe.so libdnvqe.so  
    $(CC) -shared -o $@ -L $(EXTERN_OBJ)  
pre_clr:  
    @rm -rf $(EXTERN_OBJ_DIR);
```

这种编译方式，则是先将 libupvqe.a 和 libdnvqe.a 分别先拆分成.o，再通过.o 编译成.so 文件。该方式能够将静态库中的函数符号加入.so 文件中，并不会产生同名函数冲突。

【解决】

客户 B 编译时，采取了第二种编译方式，但是受限于 libupvqe.a 和 libdnvqe.a 中存在同名的函数，导致编译时报了重定义错误。基于此，客户可以使用第一种或者第三种编译方式，都可以顺利地生成 libshare.so 文件。

12.1.3 模块 KO 之间的依赖关系

- 每个加载上去的 KO 模块，有显式依赖关系的，lsmod 查看时，会有 Used by 的标识。存在这种关系的 KO 之间需要按照顺序加载和相反顺序卸载。
- 有些模块 KO 是隐形依赖的，比如公共基础 KO 模块 mmz.ko、hi_media.ko、hi35xx_base.ko、hi35xx_sys.ko、hi35xx_tde.ko、hi35xx_region.ko 等需要先加载，这些 KO 模块若中途单独卸载再加载，可能引起一些异常。请重新依次按照顺序进行模块的卸载和加载。
- 另外一些共用的调度模块如 hi35xx_chnl.ko，供编码和 region 等模块调用，如卸载掉可能引起编码和 region 模块的异常。还有音频基础模块 hi35xx_aio.ko，其他音频模块 KO 对它没有显示依赖，但它是音频其他模块 KO 所不可缺少的。

12.1.4 SPI 驱动说明

能用到 SPI 接口的外设比较多，这里以 sensor 的配置为例。芯片通常通过 SPI 接口或 I2C 接口来配置 sensor 寄存器，现以 SPI 接口的 sensor 为例进行说明。目前我们的 IPC 发布包中，我们提供了两个配置 sensor 的 SPI 驱动：hi_sensor_spi.ko 和 hi_ssp_sony.ko（以 Sony 的 SPI 接口 sensor 为例）。

那么这两个有什么区别呢？

- hi_sensor_spi.ko 驱动实现用的是内核标准 SPI 实现，但是可能在系统繁忙时导致配置不及时。



- hi_ssp_sony.ko 是我们自己编写的 SPI 驱动，非标准的，目前给 CMV50000 sensor 使用。

12.1.5 Huawei LiteOS 系统下动态链接库的动态加载实现方案

Huawei LiteOS 支持动态链接库的动态加载策略，详细实现方法可参见《Huawei LiteOS 开发指南》。

现以音频模块为例，在使用 VQE 功能、AAC 编解码功能、重采样功能的情况下，需要依赖 libaaccomm.so、libhive_common.so、libhive_RES.so 等多个动态链接库，其动态加载建议按照以下步骤完成：

步骤 1 将需要用到的动态链接库放置到服务器的同一目录下，假定为/home/audio/lib_so/。

步骤 2 利用 Huawei LiteOS 开发包中的 sym.sh 脚本提取上述动态链接库的外部函数符号，该脚本在 Huawei LiteOS 开发包中的位置为\$(LITEOS)/tools/scripts/dynload_tools，执行命令如下：

```
cd $(LITEOS)/tools/scripts/dynload_tools  
./sym.sh /home/audio/lib_so
```

其中，\$(LITEOS)代表 Huawei LiteOS 开发包的根目录路径，须由用户指定。

步骤 3 利用 Huawei LiteOS 开发包中 dynload 工具的 makefile 生成 los_dynload_gsymbol.o 文件，该 makefile 在 Huawei LiteOS 开发包中的位置为 \$(LITEOS)/kernel/extended/dynload，新生成文件在 Huawei LiteOS 开发包中的位置为\$(LITEOS)/out/\$(CHIP) /obj/kernel/extended/dynload/src，执行命令如下：

```
cd $(LITEOS)/kernel/extended/dynload  
make LITEOSTOPDIR=$(LITEOS)
```

其中，\$(CHIP)代表芯片型号。

步骤 4 对于 Huawei LiteOS 应用程序的编写，需要注意的是在使用动态库之前须通过调用接口 LOS_PathAdd 来指定单板上的动态库加载路径，同时将使用到的动态链接库文件放置到该路径下。

步骤 5 在编译 Huawei LiteOS 的 bin 系统镜像时，链接上述 los_dynload_gsymbol.o 文件。

----结束