HiMPP 媒体处理软件

FAQ

文档版本 14

发布日期 2018-09-04

商标声明

 、HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

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。

深圳市海思半导体有限公司

地址：深圳市龙岗区坂田华为基地华为电气生产中心　　邮编：518129

网址：http://www.hisilicon.com

客户服务电话：+86-755-28788858

客户服务传真：+86-755-28357515

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

- 本文以 Hi3516A 描述为例，未有特殊说明，Hi3516D 与 Hi3516A 完全一致。
- 未有特殊说明，Hi3518EV201、Hi3516CV200 和 Hi3518EV200 完全一致。
- 未有特殊说明，Hi3556V100, Hi3559V100, Hi3516AV200 和 Hi3519V101 完全一致。
- 未有特殊说明，Hi3516EV100 和 Hi3516CV300 完全一致。

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
Hi3516A	V100
Hi3516D	V100
Hi3518E	V200
Hi3518E	V201
Hi3516C	V200
Hi3519	V100
Hi3519	V101
Hi3516A	V200
Hi3516C	V300
Hi3516E	V100
Hi3559	V100
Hi3556	V100
Hi3559A	V100

产品名称	产品版本
Hi3559C	V100
Hi3519A	V100
Hi3516C	V500
Hi3516D	V300
Hi3559	V200
Hi3556	V200

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
危险	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。
警告	表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
注意	表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
窍门	表示能帮助您解决某个问题或节省您的时间。
说明	表示是正文的附加信息，是对正文的强调和补充。

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）。

符号	举例	说明
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
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 (2018-09-04)

新增 3.3 小节

文档版本 13 (2018-06-30)

新增 1.6 和 9.2 小节

文档版本 12 (2018-05-20)

新增第 12 章“HDMI”和 13.1.5 小节

文档版本 11 (2018-05-10)

1.1 小节涉及修改

新增第 5 章 “LDC 功能”

文档版本 10 (2017-10-20)

7.2.1 小节中添加注意

文档版本 09 (2017-07-11)

第 9 次版本修改

新增 2.4 小节及 VENC 章节

11.1.4 小节涉及修改

文档版本 08 (2017-05-12)

1.5 小节涉及修改，新增 3.2 小节。

文档版本 07 (2017-03-30)

新增 PCI 章节

文档版本 06 (2017-02-25)

添加 Hi3556V100 的相关内容

新增 9.1.4 小节

文档版本 05 (2016-12-12)

1.2 小节涉及修改

新增 1.3.2、1.3.4 和 1.5 小节，新增“FISHEYE 功能”、“低功耗”和“LCD 屏幕调试”章节，新增第 8 章；新增 2.1.3 和 9.1.3 小节

文档版本 04 (2016-02-15)

新增 4.4 小节

文档版本 03 (2015-09-14)

新增 4.3 节和第 5 章

文档版本 02 (2015-06-16)

新增音频章节

文档版本 01 (2015-02-10)

第 1 次发布

目 录

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 Ctl+C 退出。如果不想阻塞等待日志信息，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 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 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，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】

a. 确认 OS 及 MMZ 内存分配情况。

详见海思发布包中的文件\01.software\board\documents_cn\《Hi35xx SDK 安装以及升级使用说明》中的第六章“地址空间分配与使用”。

b. 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 D1 系列（D1、2CIF、CIF 等）图像的大小应成整数倍的关系，如 D1 为 704x576，2CIF 为 352x576，CIF 为 352x288；而不应出现 2CIF 为 352x576，CIF 却为 360x288 的类似情况；同时，也不应出现 VI 采集 720x576 大小的图像，而 VENC 编码为 704x576 的情况。

- 每个模块的 buffer 配置最小值。
  - Linux 参考文档：《HiMPP IPC V2.0 媒体处理软件开发参考》适用于 Hi3516A/Hi3518EV20X/Hi3519V100；《HiMPP IPC V3.0 媒体处理软件开发参考》适用于 Hi3519V101/Hi3516CV300/Hi3516EV100。
  - Huawei LiteOS 请参考：《HiMPP IPC V3.0 媒体处理软件开发参考》。

- 公共 VB 刚好分配足够。

  相关接口：HI_MPI_VB_SetConf。
  - Linux 参考文档：《HiMPP IPC V2.0 媒体处理软件开发参考》适用于 Hi3516A/Hi3518EV20X/Hi3519V100；《HiMPP IPC V3.0 媒体处理软件开发参考》适用于 Hi3519V101/Hi3516CV300/Hi3516EV100 的系统控制章节。
  - Huawei LiteOS 请参考：《HiMPP IPC V3.0 媒体处理软件开发参考》。

  如何确认刚好：VB 的 proc 信息里“IsComm=1”的为公共 VB 池，要使公共 VB 池的 MinFree=0 且 logmpp 里没有任何模块打印获取不到 VB。

- VIU 模块配置：
  - Linux 参考文档：《HiMPP IPC V2.0 媒体处理软件开发参考》适用于 Hi3516A/Hi3518EV20X/Hi3519V100；《HiMPP IPC V3.0 媒体处理软件开发参考》适用于 Hi3519V101/Hi3516CV300/Hi3516EV100 的视频输入章节。
  - Huawei LiteOS 请参考：《HiMPP IPC V3.0 媒体处理软件开发参考》。

措施	相关模块参数/接口	收益	影响	注意
WDR 压缩	HI_MPI_VI_SetWDRAttr HI_MPI_VI_GetWDRAttr	非压缩可省一帧 buffer （非压缩需要 1 帧 buffer，压缩需要 2 帧 buffer）	非压缩比压缩多占带宽	仅 Hi3516A 支持

- VPSS 模块配置：
  - Linux 参考文档：《HiMPP IPC V2.0 媒体处理软件开发参考》适用于 Hi3516A/Hi3518EV20X/Hi3519V100；《HiMPP IPC V3.0 媒体处理软件开发参考》适用于 Hi3519V101/Hi3516CV300/Hi3516EV100 的视频处理子系统章节。
  - Huawei LiteOS 请参考：《HiMPP IPC V3.0 媒体处理软件开发参考》。

措施	相关模块参数/接口	收益	影响	注意
3DNR 参考帧压缩	rfr_frame_comp	非压缩可省一帧 buffer（非压缩需要 1 帧 buffer，压缩需要 2 帧 buffer）	非压缩比压缩多占带宽	仅 Hi3516A 支持
CHN0 当参考帧	• Hi3516A、Hi3519V100、Hi3516CV300：HI_MPI_VPSS_SetRefSelect HI_MPI_VPSS_GetRefSelect • Hi3519V101：HI_MPI_VPSS_SetGrpAttr, HI_MPI_VPSS_GetGrpAttr	可节省 3DNR 参考帧 buffer	如果 CHN0 的 LDC/Rotate/Mirror/Flip 功能开启会导致 3DNR 失效	-

- VENC 模块配置：
  - Linux 参考文档：《HiMPP IPC V2.0 媒体处理软件开发参考》适用于 Hi3516A/Hi3518EV20X/Hi3519V100；《HiMPP IPC V3.0 媒体处理软件开发参考》适用于 Hi3519V101/Hi3516CV300/Hi3516EV100 的视频编码章节。
  - Huawei LiteOS 请参考《HiMPP IPC V3.0 媒体处理软件开发参考》

措施	相关模块参数/接口	收益	影响	注意
动态切换编码分辨	HI_MPI_VENC_GetChnAttr, HI_MPI_VENC_Set	切换编码分辨率时不销毁通道，减	无	切换分辨率后所有参

措施	相关模块参数/接口	收益	影响	注意
率	ChnAttr	少内存碎片。		数恢复默认值。
重构帧复用参考帧亮度内存	H264eRcnEqualRef	每个通道可节省一个亮度大小的内存。	(1)超大帧、码率过冲、码流 buffer 满等异常情况下只能插入 I 帧； (2)两倍跳帧参考编码需多占用一个亮度大小的内存。	仅 Hi3516A/Hi3518EV200 支持
编码码流 buffer 使用省内存模式分配	H264eMiniBufMode H265eMiniBufMode JpegeMiniBufMode	可以把码流 buffer 设置小一些。	此模式需要用户保证码流 buffer 大小设置合理，否则会出现因码流 buffer 不足而不断重编或者丢帧的情况。	-

- VGS 模块配置：
  - Linux 参考文档：《HiMPP IPC V2.0 媒体处理软件开发参考》适用于 Hi3516A/Hi3518EV20X/Hi3519V100；《HiMPP IPC V3.0 媒体处理软件开发参考》适用于 Hi3519V101/Hi3516CV300/Hi3516EV100 的视频图形子系统章节。
  - Huawei LiteOS 请参考《HiMPP IPC V3.0 媒体处理软件开发参考》

措施	相关模块参数	收益	影响	注意
设置 VGS 支持的最大 job 数	max_vgs_job	一个 job 节点 192byte。	job 数量过少会限制 VGS 性能。	-
设置 VGS 支持的最大 task 数	max_vgs_task	一个 task 节点 1256 byte。	task 数量过少会限制 VGS 性能	-
设置 VGS 支持的最大 node 数	max_vgs_node	一个 node 节点 1104 byte。	node 数量过少会限制 VGS 性能	-

- HIFB 模块配置：

  参考文档：《HiFB 开发指南》。

措施	相关模块参数/接口	收益	影响	注意
设置合适的图形层物理显存	video	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	-

1.3 性能相关

1.3.1 VGS LDC 性能

【现象】Hi3516A：5M LDC 帧率达不到实时处理。

【分析】在 MPP 中，VGS 实现的功能主要包括：扩展通道(Scaling)、CoverEx、OverlayEx、Rotate、LDC 等功能，也即 VGS 的性能是所有这些功能的总和，VI/VPSS/VENC 等模块都共享。

Hi3516A VGS LDC 性能约束：1080P@30fps，LDC 建议在离线模式下使用。

1.3.2 VGS LDC 目前存在此现象

【现象】LDC 在 fullall 模式下，连续配置矫正率 ratio 的时候，图像会出现跳变现象。

【分析】LDC 的应用主要归结为以下两类：

- 机器接定焦镜头：这类机器在实验室场景下标定 LDC 参数，实际使用中不需要调节 ratio，因此 crop 模式和 fullall 模式均没有问题。
- 机器接变焦镜头：这类机器会根据焦距的不同标定多组参数，实际使用中，在调焦的过程中，根据不同焦距的 ratio 参数，连续配置 ratio，这时候，crop 模式没有问题，fullall 模式会出现图像不均匀变化现象。经过测试，在遍历所有可能的取值范围内，跳变发生的次数仅为 1 到 2 次。

目前涉及此现象的方案有：Hi3516A/Hi3516D、Hi3518EV200、Hi3516CV200；另外，Hi3518EV100、Hi3518CV100、Hi3518AV100、Hi3516CV100 不涉及此现象。

1.3.3 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。

1.3.4 各模块时钟频率配置

- 对于 Hi3519V101，各模块的默认时钟频率如下：

  IVE:396M， GDC:475M， VGS:500M， VEDU:600M， VPSS:300M
  VI0:300M， ISP0:300M， ISP1:300M， VI1:300M。
- 除了默认频率，各个模块均有其他频率可配置，具体参考《Hi3519V101 专业型 HD IP Camera SoC 用户指南》或《Hi3516AV200 专业型 HD IP Camera SoC 用户指南》。如果需要修改某个模块的频率，请参考《clkcfg_hi3519v101.sh》。
  - 其中 VI、ISP 的时钟频率需要根据前端的 sensor 进行配置，具体参考 mpp_big-little/component/isp/sensor/ readme_cn.txt(或 readme_en.txt)。

- VPSS 的频率默认配置为 300M，客户在调试的过程中，如果发现 VPSS 性能不足，可以尝试将 VPSS 的频率改为 396M。

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 VI-VPSS 在线/离线模式注意事项

VI 和 VPSS 的协作模式分为以下两种（模式切换由 load 脚本参数控制，对应 sys 模块参数 vi_vpss_online）：

- VI/VPSS 离线模式是指 VI 进行时序解析后将图像数据写出到 DDR，VPSS 从 DDR 中载入 VI 采集的数据进行图像处理，是传统 Hi3518/Hi3520D 等芯片的 VI/VPSS 的协作模式。
- VI/VPSS 在线模式是指 VI 进行时序解析后直接在芯片内部将数据传递到 VPSS，中间无 DDR 写出的过程。在线模式可以省一定的带宽和内存，降低端到端的延时。需要注意的是，在线模式时，因为 VI 不写出数据到 DDR，无法进行 CoverEx、OverlayEx、Rotate、LDC 等操作，需要在 VPSS 各通道写出后再进行 Rotate/LDC 等处理，而且有些功能只在离线下能支持，比如 DIS。
- 离/在线模式，在 ko/load35xx 脚本中控制，用户可以通过 HI_MPI_SYS_GetViVpssMode 接口获取当前的工作模式，或者通过 cat /proc/umap/sys 命令查看 sys 模块的 proc 信息中的 vi_vpss_online 项查看。

芯片的某些规格，依赖于 VI-VPSS 当前处于在线或离线模式：

- VPSS 的通道的某些规格，依赖于 VI/VPSS 当前处于在线或离线。具体请参考《HiMPP IPC V3.0 媒体处理软件开发参考》中的视频处理子系统章节表 6-1。
- 调用 HI_MPI_VPSS_CreateGrp 创建 VPSS GROUP 时，VPSS GROUP 号的范围：
  - 离线：[0, VPSS_MAX_GRP_NUM)。
  - 在线：0。
- 对于 VPSS，Hi3519V100/Hi3519V101 在 VI-VPSS 处于离线/在线时，所支持的最大宽度不一样，使用 VPSS_OFFLINE_MAX_IMAGE_WIDTH 和 VPSS_ONLINE_MAX_IMAGE_WIDTH 表示。
  Hi3516A/Hi3518EV200/Hi3516CV300/Hi3516EV100 的 VPSS 在 VI-VPSS 处于离线/在线时所支持的最大图像宽度是一样的。

- 对于 VPSS，Hi3519V101 在 VI-VPSS 处于离线/在线时，所支持的最大高度不一样，使用 VPSS_OFFLINE_MAX_IMAGE_HEIGHT 和 VPSS_ONLINE_MAX_IMAGE_HEIGHT 表示。

  Hi3516A/Hi3518EV200/Hi3519V100/Hi3516CV300/Hi3516EV100 的 VPSS 在 VI-VPSS 处于离线/在线时所支持的最大图像高度是一样的。
- 对于 VI 的 Mirror/Flip 功能，各芯片的支持情况如下表：

芯片类型	Mirror/Flip
Hi3516A	不支持
Hi3518EV200/Hi3516CV300/Hi3516EV100	离线模式支持，在线模式不支持
Hi3519V100/Hi3519V101	离线模式支持，在线模式不支持

VI-VPSS 处于在线模式时，有以下注意事项：

- 对于 Hi3519V101，有二个 VI 设备 DEV0 与 DEV1，在线模式只能是 DEV0 与 VPSS 在线。
- 请确保 ISP 和 VPSS 的时钟配置成一致。
- 不支持调用以下接口：

  HI_MPI_VI_SetSnapAttr
  HI_MPI_VI_GetSnapAttr
  HI_MPI_VI_EnableSnap
  HI_MPI_VI_DisableSnap
  HI_MPI_VI_GetSnapRaw
  HI_MPI_VI_ReleaseSnapRaw
  HI_MPI_VI_SendSnapRaw
  HI_MPI_VI_MutliTrigger
  HI_MPI_VPSS_SetGrpCrop
  HI_MPI_VPSS_GetGrpCrop
  HI_MPI_VPSS_SendFrame
  HI_MPI_VPSS_SendFrameEx
  HI_MPI_VPSS_GetGrpFrame
  HI_MPI_VPSS_GetGrpFrameEx
  HI_MPI_VPSS_ReleaseGrpFrame
  HI_MPI_VPSS_ReleaseGrpFrameEx
  HI_MPI_VPSS_SetGrpFrameRate
  HI_MPI_VPSS_GetGrpFrameRate
  HI_MPI_VPSS_SetGrpDelay
  HI_MPI_VPSS_GetGrpDelay
  HI_MPI_VPSS_SetGrpSnapNRSParam
  HI_MPI_VPSS_GetGrpSnapNRSParam
  HI_MPI_VPSS_SetStitchBlendParam

HI_MPI_VPSS_GetStitchBlendParam

- VI_CHN_STAT_S 中的变量 u32FrmRate、u32LostInt、u32VbFail，在线时无意义。
- 在 VI-VPSS 的在线方案中，编码器性能足够的情况下，VPSS 支持按照，以行为单位，边采集边发送的方式，将图像发送给编码模块进行编码，用来减少 VPSS 处理完整帧图像再发送给编码模块过程中，数据的延时时间。这样的方式即为低延时方案。低延时方案仅在 VI-VPSS 在线时支持，因此，HI_MPI_VPSS_SetLowDelayAttr，HI_MPI_VPSS_GetLowDelayAttr 接口仅在 VI-VPSS 在线时支持。在线模式时，由于 VI 和 VPSS 的逻辑处理需要时序严格同步，所以 GROUP 创建中的 group 的图像属性必须和 VI 的图像设置属性一致；否则会出现 VPSS 的中断错误。具体请参见 VPSS_GRP_ATTR_S。
- 对于 VPSS，在线低延时通道不支持 Flip 功能。
- VPSS 在 VI-VPSS 处于在线时，不支持输入源图像作为 NR 的参考帧。
- 在设置编码的码率控制参数时，则 SrcFrmRate 设置为 VPSS 的实际输出帧率，因为 TimeRef 是在 VPSS 输出的时候产生。

VI-VPSS 处于离线模式时，有以下注意事项：

- VPSS 的拼接功能，目前仅有 Hi3519V101 支持，而且仅当 VI-VPSS 处于离线模式时支持。
- 在设置编码的码率控制参数时，SrcFrmRate 设置为 VI 的实际输出帧率，因为 TimeRef 是在 VI 输出的时候产生。

1.6 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。

2 MIPI 配置

2.1 MIPI 配置说明

Hi3516A MIPI 具体规格请参考芯片手册《Hi3516A / Hi3516D 专业型 HD IP Camera Soc 用户指南.pdf》和《Features of the Video Interfaces of HiSilicon IP Cameras.pdf》

2.1.1 Lane_id 如何配置

【现象】Hi3516A MIPI 有 2 个 link，每一个 link 含 4 个 lane，共有 8 个 lane，在产品应用中可以选择使用其中几个 lane，对应 mipi 接口中的 lane_id[8]配置。Sensor 与 Hi3516A 对接时，需要如何配置。

【解决】Hi3516A 对接 sensor 时，未使用的 lane 将其对应的 lane_id 配置为-1。可以通过配置 MIPI 模块 Lane_id 寄存器实现数据通道顺序的调整，根据硬件单板与实际 sensor 输出通道的对应关系配置此寄存器的值。

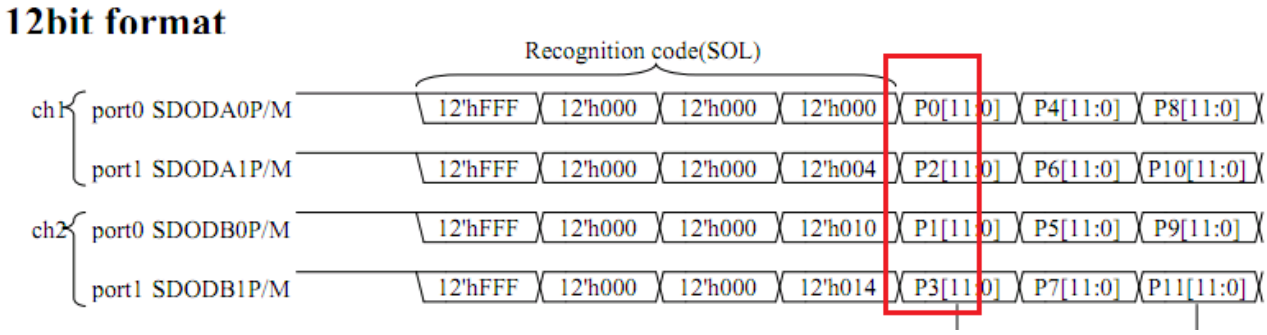
下面以 demo 板 mn34220 为例进行说明：

a. 查看硬件连接及实际的传输与硬件的对应关系，如 demo 板 lvds 接口的 mn34220 使用了 4 个 lane。硬件连接如表 2-1 所示：

表2-1 硬件连接表

MN34220 管脚	Hi3516A MIPI 管脚
SENSOR_SDODA0M SENSOR_SDODA0P	MIPI0_D0M MIPI0_D0P
SENSOR_SDODA1M SENSOR_SDODA1P	MIPI0_D1M MIPI0_D1P
SENSOR_SDODB0M SENSOR_SDODB0P	MIPI1_D0M MIPI1_D0P
SENSOR_SDODB1M SENSOR_SDODB1P	MIPI1_D1M MIPI1_D1P

b. 实际传输关系（见 sensor datasheet），实际传输数据为 0, 2, 1, 3：



结合 a、b 则 lane_id 配置为：

.lane_id = {0, 2, -1, -1, 1, 3, -1, -1},

c. Sync_code 配置是根据 lane_id 生效的，如果为-1，则对应 sync_code 不会生效；

2.1.2 LVDS mode sync code 如何配置

【现象】sensor LVDS/SUB_LVDS 的 sync_code 有两种模式：LVDS_SYNC_MODE_SOL，LVDS_SYNC_MODE_SAV，不同 sensor 或者同一 sensor 不同的传输模式都需要配置不同的 sync code。

【解决】Mipi 接口中 sync_code[LVDS_LANE_NUM][WDR_VC_NUM][SYNC_CODE_NUM]

需要根据 sensor datasheet 进行配置，其中：

LVDS_LANE_NUM	对应 lvds 硬件物理通道，Hi3516A mipi 逻辑有 8 个 Lane
WDR_VC_NUM	WDR 通道数，如 2-1WDR 合成时对应为 2 个 WDR 通道，最多为 4 个；Hi3516A 只支持 2-1WDR。
SYNC_CODE_NUM	每条 lane 的 Sync_code 由 4 个码字组成，在不同的模式下，对应的意义不同。

每一个 lane 对应的 sync_code[4]的意义如下表所示：

sync_mode	sync_code[4]意义
LVDS_SYNC_MODE_SOL	SOF, EOF, SOL, EOL
LVDS_SYNC_MODE_SAV	invalid sav, invalid eav, valid sav, valid eav

说明

每一 SOF/EOF/SOL/EOL 的 sync_code 前三个为固定码字：0xFFFF 0x0000 0x0000

Sync code 配置的原则：在同一个 Link 内，无论硬件以何种顺序与 Sensor 连接，Sync code 都按照正常顺序配置。

两个 Link 之间乱序时，仍然遵循上述准则，即在同一 Link 内保持正常顺序配置。Panasonic 的 Sensor 普遍存在两个 Link 之间乱序，其他厂家暂不涉及。

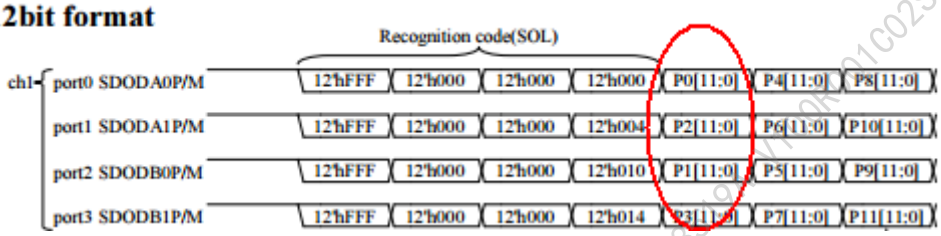
举例 1：同一 Link 内乱序

如 mn34220 sync_mode 为 SOL 模式，工作在 1channel 4port (4 lane) 12bit 模式时，datasheet 中关于 Sync code 的描述如下：

ch	port	Name	Code (12bit×4)				ch	port	Name	Code (12bit×4)			
ch1	Port0	SOF	FFFh	000h	000h	002h	ch1	Port2	SOF	FFFh	000h	000h	012h
		SOL	FFFh	000h	000h	000h			SOL	FFFh	000h	000h	010h
		EOL	FFFh	000h	000h	001h			EOL	FFFh	000h	000h	011h
		EOF	FFFh	000h	000h	003h			EOF	FFFh	000h	000h	013h
	Port1	SOF	FFFh	000h	000h	006h		Port3	SOF	FFFh	000h	000h	016h
		SOL	FFFh	000h	000h	004h			SOL	FFFh	000h	000h	014h
		EOL	FFFh	000h	000h	005h			EOL	FFFh	000h	000h	015h
		EOF	FFFh	000h	000h	007h			EOF	FFFh	000h	000h	017h

各个通道像素的顺序如下：

12bit format



所以相当于正常的顺序应该为 SDODA0、SDODB0、SDODA1、SDODB1，应该按照这个顺序在 mipi 接口中配置 Sync code。则 sync_code 配置为：

```
.sync_code = {
                {{0x002, 0x003, 0x000, 0x001}, //PHY0_lane0
                {0x202, 0x203, 0x200, 0x201},
                {0x102, 0x103, 0x100, 0x101},
                {0x302, 0x303, 0x300, 0x301}},

                {{0x012, 0x013, 0x010, 0x011},//PHY0_lane1
                {0x212, 0x213, 0x210, 0x211},
                {0x112, 0x113, 0x110, 0x111},
                {0x312, 0x313, 0x310, 0x311}},

                {{0x006, 0x007, 0x004, 0x005}, //PHY0_lane2
                {0x206, 0x207, 0x204, 0x205},
                {0x106, 0x107, 0x104, 0x105},
                {0x306, 0x307, 0x304, 0x305}},
```

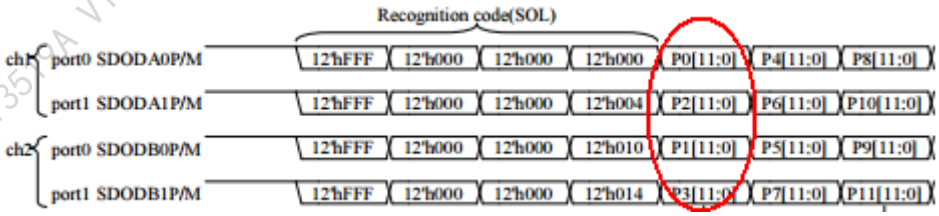
```
    {{0x016, 0x017, 0x014, 0x015}, //PHY0_lane3
    {0x216, 0x217, 0x214, 0x215},
    {0x116, 0x117, 0x114, 0x115},
    {0x316, 0x317, 0x314, 0x315}},

    {{0x00a, 0x00b, 0x008, 0x009}, //PHY1_lane0
    {0x20a, 0x20b, 0x208, 0x209},
    {0x10a, 0x10b, 0x108, 0x109},
    {0x30a, 0x30b, 0x308, 0x309}},

    {{0x00a, 0x00b, 0x008, 0x009}, //PHY1_lane1
    {0x20a, 0x20b, 0x208, 0x209},
    {0x10a, 0x10b, 0x108, 0x109},
    {0x30a, 0x30b, 0x308, 0x309}},

    {{0x01a, 0x01b, 0x018, 0x019}, //PHY1_lane2
    {0x21a, 0x21b, 0x218, 0x219},
    {0x11a, 0x11b, 0x118, 0x119},
    {0x31a, 0x31b, 0x318, 0x319}},

    {{0x01a, 0x01b, 0x018, 0x019}, //PHY1_lane3
    {0x21a, 0x21b, 0x218, 0x219},
    {0x11a, 0x11b, 0x118, 0x119},
    {0x31a, 0x31b, 0x318, 0x319}}
}
```

举例 2：两个 Link 之间乱序

如 mn34220 sync_mode 为 SOL 模式，工作在 2 channel 2port (4 lane) 12bit 模式时，datasheet 描述如下：

ch	port	Name	Code (12bit×4)				ch	port	Name	Code (12bit×4)			
ch1	Port0	SOF	FFFh	000h	000h	002h	ch2	Port0	SOF	FFFh	000h	000h	012h
		SOL	FFFh	000h	000h	000h			SOL	FFFh	000h	000h	010h
		EOL	FFFh	000h	000h	001h			EOL	FFFh	000h	000h	011h
		EOF	FFFh	000h	000h	003h			EOF	FFFh	000h	000h	013h
	Port1	SOF	FFFh	000h	000h	006h		Port1	SOF	FFFh	000h	000h	016h
		SOL	FFFh	000h	000h	004h			SOL	FFFh	000h	000h	014h
		EOL	FFFh	000h	000h	005h			EOL	FFFh	000h	000h	015h
		EOF	FFFh	000h	000h	007h			EOF	FFFh	000h	000h	017h

各个通道像素的顺序如下：



由于在两个 Link 之间线序有交叉，但是根据上述原则，只在同一 Link 内部考虑 Sync code 的配置。所以最终的的 sync_code 配置为：

```
.sync_code = {
                {{0x002, 0x003, 0x000, 0x001}, //PHY0_lane0
                {0x202, 0x203, 0x200, 0x201},
                {0x102, 0x103, 0x100, 0x101},
                {0x302, 0x303, 0x300, 0x301}},

                {{0x006, 0x007, 0x004, 0x005}, //PHY0_lane1
                {0x206, 0x207, 0x204, 0x205},
                {0x106, 0x107, 0x104, 0x105},
                {0x306, 0x307, 0x304, 0x305}},

                {{0x00a, 0x00b, 0x008, 0x009}, //PHY0_lane2
                {0x20a, 0x20b, 0x208, 0x209},
                {0x10a, 0x10b, 0x108, 0x109},
                {0x30a, 0x30b, 0x308, 0x309}},

                {{0x00a, 0x00b, 0x008, 0x009}, //PHY0_lane3
                {0x20a, 0x20b, 0x208, 0x209},
                {0x10a, 0x10b, 0x108, 0x109},
                {0x30a, 0x30b, 0x308, 0x309}},

                {{0x012, 0x013, 0x010, 0x011},//PHY1_lane0
                {0x212, 0x213, 0x210, 0x211},
                {0x112, 0x113, 0x110, 0x111},
                {0x312, 0x313, 0x310, 0x311}},

                {{0x016, 0x017, 0x014, 0x015}, //PHY1_lane1
                {0x216, 0x217, 0x214, 0x215},
                {0x116, 0x117, 0x114, 0x115},
                {0x316, 0x317, 0x314, 0x315}},

                {{0x01a, 0x01b, 0x018, 0x019}, //PHY1_lane2
                {0x21a, 0x21b, 0x218, 0x219},
                {0x11a, 0x11b, 0x118, 0x119},
                {0x31a, 0x31b, 0x318, 0x319}},

                {{0x01a, 0x01b, 0x018, 0x019}, //PHY1_lane3
                {0x21a, 0x21b, 0x218, 0x219},
                {0x11a, 0x11b, 0x118, 0x119},
                {0x31a, 0x31b, 0x318, 0x319}}
            }
```

2.1.3 LVDS mode 两个 Link 场景出现异常时的处理

【现象】使用两个 Link 进行数据传输时，如果受到干扰（例如 ESD 等），数据出现错误导致图像花屏等异常。当干扰消除，前端（sensor 输出）数据恢复正常后，系统不能恢复正常。

【解决】

1. 需要将偏移地址为 0x1004 的中断 mask 寄存器 INT_MASK_LINK0 和偏移地址为 0x1404 的中断 mask 寄存器 INT_MASK_LINK1 的[19:16]bit 位配置为 0。
2. 在 MIPI 中断处理函数 mipi_interrupt_route 中的 mipi_int_statics 的里面去掉 int_raw 和 int2_raw 的条件判断，即只要进 MIPI 中断处理函数，就要复位 MIPI 控制器。
3. 配置 LVDS 模式的属性时，需要将宽配置为该 sensor 实际输出的宽。

下面以 IMX123 sensor 3M（2048*1536）线性场景为例，该场景输出实际宽为 2064。

- 修改 hi_mipi.h 中宏定义#define COMBO_LINK_INT_MASK ~(0x300000)为#define COMBO_LINK_INT_MASK ~(0x3f0000)
- 在中断函数 mipi_interrupt_route 中的 mipi_int_statics 里面，删除条件判断 if (int_raw || int2_raw)即可。
- 用户程序需配置 LVDS 属性 lvds_dev_attr_t 的 img_size. width 为 2064，img_size. height 为 1536。

说明
此修改只针对 LVDS 模式的两 Link 场景，其他情况请保持原始代码。

2.2 MIPI 频率说明

2.2.1 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：

VI_Freq * Pix_Width = Lane_Num * 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：

MIPI_Freq = (250 * 12) / 4 = 750

即每个 lane 最高频率为 750MHz

2.3 Sensor 复位

2.3.1 Hi3516A Sensor 复位管脚

【现象】Hi3516A 有 1 pin 脚专门用于 sensor 复位(SENSOR_RSTN)，建议客户默认使用。如果需要修改 sensor reset pin，需要注意 mipi 驱动做适配修改。

【分析】sensor reset/unreset 操作是在 mipi 驱动中实现的，sensor reset/unreset 与 mipi 配置有顺序要求。

【解决】根据实际使用的管脚，在 mipi 驱动 mipi_reset_sensor/mipi_unreset_sensor 函数中进行适配。

2.4 时序

2.4.1 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。

3 VI 功能

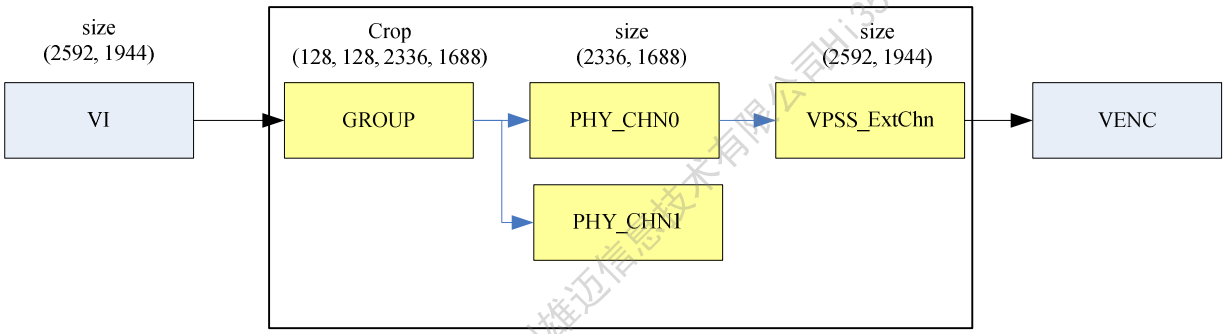
3.1 DIS 功能

3.1.1 Hi3516A 如何实现 5M 场景 DIS 功能

【现象】1080P 场景下 DIS 功能方案为：VI 进一个分辨率大于(1920，1080)的图像，如(1920+256, 1080+256)，MPP 算法计算出图像抖动的 offset 后，通过 VPSS Grp Crop 裁成 1080P。但由于 VI 最大只能采集 2592 宽度的图像，此方案在 5M 场景下无法实现。

【分析】5M 场景下 VI 只能采集 2592 宽度的图像，故要实现防抖，需要先 CROP 为比 5M 小的图像，如(2592-256，1944-256)，再放大为 5M 图像。

【解决】CROP 后 VPSS CHN0 绑定一个扩展通道放大为 5M 图像，如下图所示：



【注意】此处举例 5M 场景输入高度以 1944 为例，实际使用时根据 Sensor 时序进行配置，时序允许的情况下，高度可以大于 1944；Hi3516A 支持的最大抖动范围为±128，可以根据实际场景进行 CROP 的调整。

3.2 WDR 功能

3.2.1 QudraWDR 注意事项

对于 Hi3519V101，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，这时候，建议按照以下步骤修改总线的 latency：

修改 ko 目录下的 sysctl.sh 脚本的配置：

himm 0x120640ac 0x00000040

修改为：

himm 0x120640ac 0x00000010

同时增加以下命令：

himm 0x12064078 0x81

注意

以上修改可以使得 VI 的读操作的优先级加大，在 VI-VPSS 离线模式下实测能减少出现低带宽的几率，如果系统的带宽过大，仍有可能出现低带宽，需要用户实际测试。以上修改会导致 DDRC 总的性能下降。

3.3 VI 时序配置

Hi3559AV100 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，然后对 MIPI、VI 设备和 VI PIPE 做配置，差异点如下：

3.3.1 BT.1120

3.3.1.1 MIPI 配置

第 0，1 路 BT.1120 无需配置，第 2 路需要配置 MIPI 输入模式为 INPUT_MODE_BT1120。

```
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

```
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_SINGNAL,
VI_HSYNC_NEG_HIGH, VI_VSYNC_VALID_SINGAL,
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 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}
};
```

3.3.2 BT.656

3.3.2.1 MIPI 配置

第 0，1 路 BT.656 无需配置，第 2 路需要配置 MIPI 输入模式为 INPUT_MODE_BT656。

```
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：au32ComponentMask[0] = 0xFF000000,表示使用高 8bit 传输数据
- 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

第 0，1 路 BT.656 无需配置，第 2 路需要配置 MIPI 输入模式为 INPUT_MODE_BT601。

```
    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：au32ComponentMask[0] = 0xFF000000,表示使用高 8bit 传输数据

- 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,

    {
        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
            };
```

3.3.3.3 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_SINGNAL,
VI_HSYNC_NEG_HIGH, VI_VSYNC_VALID_SINGAL,
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}
            };
```

4 FISHEYE 功能

4.1 鱼眼矫正功能

4.1.1 Hi3519V100/ Hi3519V101 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_SetFisheyeAttr/ HI_MPI_VPSS_SetFisheyeAttr 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 Hi3519V100/Hi3519V101 的 GDC 的性能不足。

【解决】用户从 VI/VPSS 模块获取图像，调用 HI_MPI_FISHEYE_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_FISHEYE_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 接口 HI_MPI_VI_SetLDCAttr 和 VPSS 接口 HI_MPI_VPSS_SetLDCAttr 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 编码的音频码流

6.1.1 PC 如何播放由 Hisilicon 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】

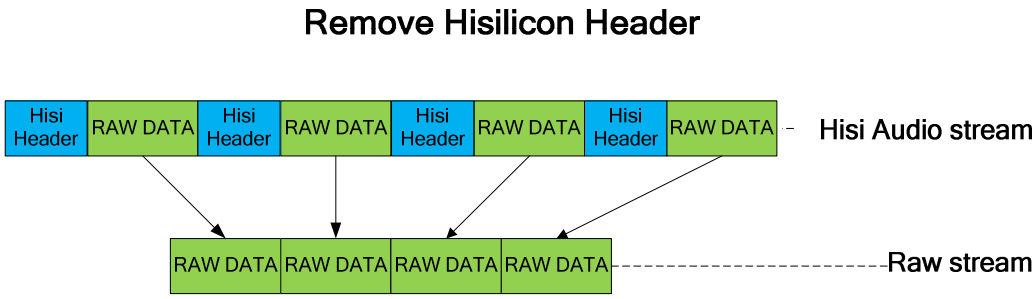
由 Hisilicon 编码的音频码流，会在每一帧数据前添加一个海思语音帧头详见：

- Linux：《HiMPP IPC V2.0 媒体处理软件开发参考》适用于 Hi3516A/Hi3518EV20X/Hi3519V100；《HiMPP IPC V3.0 媒体处理软件开发参考》适用于 Hi3519V101/Hi3516CV300/Hi3516EV100 的音频章节 9.2.2.3 海思语音帧结构。
- Huawei LiteOS：《HiMPP IPC V3.0 媒体处理软件开发参考》，PC 端软件不能识别海思语音帧头。

【解决】

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

图6-1 去除海思语音帧头示意图



去除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 和参考链接 2 进行添加。

参考链接 1：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 如何播放标准的音频码流

6.2.1 Hisilicon 如何播放标准的音频 G711/G726/ADPCM 码流

【现象】

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

【分析】

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

a. 获取 RAW 流数据：

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

b. 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

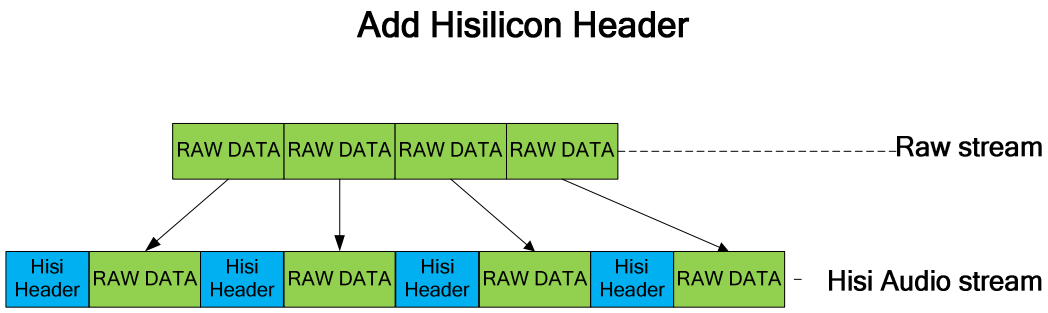
- ADPCM 格式中，仅支持 IMA ADPCM 格式， 每采样点比特数(wbitspersample)只支持 4。
- 如果 ADPCM 码流添加了 WAV Header，可以从 WAV Header 中获得每块字节数信息；如果为 ADPCM 裸码流，则需要从码流提供方获取每块字节数信息。

- 编码格式仅支持单声道编码格式。

c. 添加海思语音帧头：

添加海思语音帧头的操作如图 6-2 所示：

图6-2 添加海思语音帧头示意图



添加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

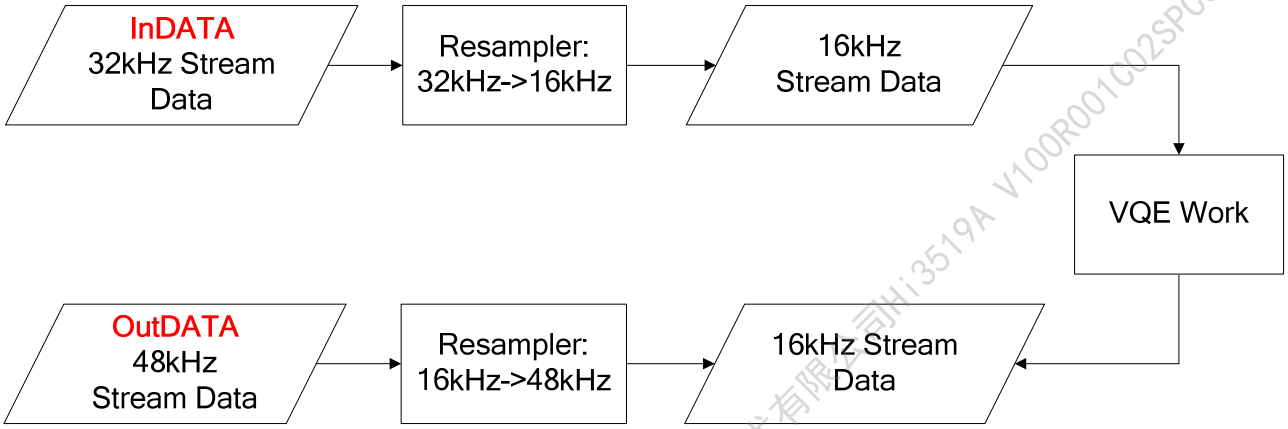
【现象】

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 AISampleRate/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(AISampleRate, ResOutSampleRate) /2 输出信息频段。如配置 AI 采样率 16kHz，输出采样率 32kHz，min(AISampleRate, ResOutSampleRate) = 16kHz，则输出信息频段为 0 – 8kHz。
3. 使能 VQE，不使能重采样：取 min(AISampleRate, VQEWorkSampleRate) /2 输出信息频段。如配置 AI 采样率为 32kHz，VQEWorkSampleRate 为 16kHz，min(AISampleRate, VQEWorkSampleRate) = 16kHz，则输出信息频段为 0 – 8kHz。
4. 使能 VQE，使能重采样：取 min(AISampleRate, VQEWorkSampleRate, 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(AISampleRate, VQEWorkSampleRate, ResOutSampleRate) = 16kHz，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 的控制寄存器 MISC_CTRL52 的 dacl_deemph[22:21]和 dacr_deemph[20:19]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，此时寄存器 MISC_CTRL52 的[22:21]和[20:19]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 的控制寄存器 MISC_CTRL52 的 dacl_deemph[22:21]和 dacr_deemph[20:19]位未关闭（都不为 00），故出现此问题。

【解决】

在 AI 通道，AUDIO CODEC 的控制寄存器是没有开设预加重功能的，所以 AUDIO CODEC 的控制寄存器 MISC_CTRL52 的 dacl_deemph[22:21]和 dacr_deemph[20:19]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】

如果寄存器 MISC_CTRL52 的[22:21]和[20:19]位是 reserved 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。

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。

目前 IPC 芯片支持的 LCD 接口类型如表 8-1 所示。

表8-1 支持 LCD 接口类型

接口	芯片		
	Hi3518EV200	Hi3519V100	Hi3519V101
6bit 串行	支持	支持	支持
8bit 串行	支持	支持	支持
16bit 并行	不支持	支持	支持
24bit 并行	不支持	不支持	支持

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《Hi35xx HD IP Camera Soc 用户指南》。linux 版本的用户还可以参考发布包里的 pinmux_hi35xx.sh 脚本，而 Huawei LiteOS 用户可以参考 sdk_init.c 文件中的管脚配置部分。

注意

在 Hi3519V101 上，16bit 并行的 LCD 屏有两种接法，即

- 接法 1。R[4:0]：VO_DATA [23:19]、G[5:0]：VO_DATA [15:10]、B[4:0]：VO_DATA [7:3]；
- 接法 2。R[4:0]：VO_DATA [15:11]、G[5:0]：VO_DATA [10:5]、B[4:0]：VO_DATA [4:0]。

当采用接法 2 时，接口类型选择 VO_INTF_LCD_16BIT，如果采用接法 1 时，接口类型选择 VO_INTF_LCD_24BIT。

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  bSynm;     /* 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;   /* herizontal active area */
    HI_U16   u16Hbb;    /* herizontal back blank porch */
    HI_U16   u16Hfb;    /* herizontal front blank porch */
    HI_U16   u16Hmid;   /* bottom herizontal 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  bIdv;      /* inverse data valid of output */
    HI_BOOL  bIhs;      /* inverse horizontal synch signal */
    HI_BOOL  bIvs;      /* inverse vertical synch signal */
} VO_SYNC_INFO_S;
```

各参数说明，如表 8-2 所示。

表8-2 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。
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。
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。
bIdv	数据有效信号的极性。目前不可调，默认高有效。
bIhs	水平有效信号的极性，配置 0 为高有效，配置 1 为低有效。
bIvs	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 确认 CRG 的配置

用户除了需要配置用户时序，还需要自行配置 VDP 和 LCD 的 CRG 时钟寄存器。以输出正确的时钟到 LCD 上。

表8-3 Hi3518EV200 VDP CRG 寄存器 PERI_CRG13（0x20030034）

Bits	Name	Description	Suggestion
[31:21]	reserved	保留。	-
[20]	lcd_cksel	LCD 分频时钟选择：	见下文中关于分频比配置

Bits	Name	Description	Suggestion
		0：LCD 3 分频时钟； 1：LCD 4 分频时钟。	的介绍。
[19]	reserved	保留	-
[18]	lcd_cken	LCD 工作时钟门控： 0：关闭； 1：打开。	1
[17]	hd_lcd_cksel	HD_LCD 工作时钟选择： 0：HD 分频时钟作为 DHD1 通道时钟； 1：LCD 分频时钟作为 DHD1 通道时钟。	1
[16:15]	reserved	保留	-
[14]	vo_out_cksel	VO OUT_CLK 频率选择。 0：HD 时钟作为 VDP 输出随路时钟； 1：LCD 时钟作为 VDP 输出随路时钟；	1
[13:12]	hd_cksel	DHD 频率配置，其中 bit[12]： 0：HD 时钟频点 27MHz； 1：HD 时钟频点 74.25MHz： bit[13]： 0：HD 分频时钟，分频倍数 2 倍 1：HD 分频时钟，分频倍数 1 倍	-
[11:10]	reserved	保留	-
[9]	vou_hd_cken	VOU HD 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[8]	vou_acken	VOU AXI 总线时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1

Bits	Name	Description	Suggestion
[7]	vo_out_cken	VO_CLKOUT 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[6]	vou_pcken	VO APB 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[5]	vou_ppc_cken	VO PPC 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[4]	vou_cfg_cken	VO CFG (内部配置)时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[3]	reserved	保留	-
[2]	vo_out_pctrl	VOU HD 输出随路时钟相位控制。默认反向 0：正向时钟； 1：反向时钟。	根据实际效果来调整，效果不对时尝试调整。
[1]	reserved	保留	-
[0]	vo_srst_req	VOU 软复位请求。 0：撤消复位； 1：复位。	0

表8-4 Hi3518EV200 LCD 的 CRG 寄存器：PERI_CRG26(0x20030068)

Bits	Name	Description	Suggestion
[31:27]	reserved	保留。	-
[26:0]	lcd_mclk_div	LCD 时钟，可配置。	假定目标频率 X(MHZ)，则 lcd_mclk_div = (X/594) * 2^27。

表8-5 Hi3519V100 VDP CRG 寄存器 PERI_CRG17（0x12010044）

Bits	Name	Name	Suggestion
[31:17]	reserved	保留。	-
[16:14]	vo_out_cksel	VO_OUT_CLK 频率选择。 000：148.5MHz； 001：74.25MHz； 010：37.125MHz； 011：107MHz； 100：54M Hz； 101：27M Hz； 110：LCD 分频器时钟； 111：保留。	110
[13:12]	hd_div_mode	VO_OUT_CLK 与 DHD 通道时钟分频比配置。 00：不分频； 01：2 分频； 10：3 分频； 11：4 分频。	见下文中关于分频比配置的介绍。
[11]	vdac_cken	VDACH 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[10]	vou_sd_cken	VOU SD 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[9]	vou_hd_cken	VOU HD 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[8]	vo_out_cken	VO_CLKOUT 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[7]	vou_acken	VOU AXI 总线时钟门控配置寄存器。	1

Bits	Name	Name	Suggestion
		0：关闭时钟； 1：打开时钟。	
[6]	vou_pcken	VOU APB 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[5]	vou_ppc_cken	VOU PPC 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[4]	vou_cfg_cken	VOU CFG (内部配置)时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[3]	vdac_pctrl	VDAC 时钟相位控制。 0：正向时钟； 1：反向时钟。	1
[2]	vo_out_pctrl	VOU HD 输出随路时钟相位控制。默认反向 0：正向时钟； 1：反向时钟。	根据实际效果来调整，效果不对时尝试调整。
[1]	reserved	保留。	-
[0]	vo_srst_req	VOU 软复位请求。 0：撤消复位； 1：复位。	0

表8-6 Hi3519V100 LCD 的 CRG 寄存器：PERI_CRG18(0x12010048)

Bits	Name	Description	Suggestion
[31:28]	reserved	保留。	-
[27]	lcd_cken	LCD 分频器时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1

Bits	Name	Description	Suggestion
[26:0]	lcd_mclk_div	LCD 分频时钟，可配置。 假定目标频率 X(MHZ)，则 lcd_mclk_div = (X/1188) * 2^27。	假定目标频率 X(MHZ)，则 lcd_mclk_div = (X/1188) * 2^27。

表8-7 Hi3519V101 VDP CRG 寄存器 PERI_CRG17(0x12010044):

Bits	Name	Description	Suggestion
[31:18]	reserved	保留。	-
[17]	vdp_core_clksel	VDP 工作时钟选择。 0：300MHz 1：198MHz。	0
[16:14]	vdp_out_clksel	VDP 输出时钟（芯片输出时钟）选择。 000：148.5MHz； 001：74.25MHz； 010：37.125MHz； 011：107MHz； 100：54MHz； 101：27MHz； 110：LCD 分频器时钟； 111：保留。	110
[13:12]	hd_div_mode	VDP 输出时钟（芯片输出时钟）与 HD 通道时钟分频比配置。 00：不分频； 01：2 分频； 10：3 分频； 11：4 分频。	见下文中关于分频比配置的介绍。
[11]	vdac_clken	VDAC 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1

Bits	Name	Description	Suggestion
[10]	vdp_sd_clken	VDP SD 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[9]	vdp_hd_clken	VDP HD 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[8]	vdp_out_clken	VDP 输出时钟（芯片输出时钟）门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[7]	vdp_aclken	VDP AXI 总线时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[6]	vdp_pclken	VDP APB 时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[5]	vdp_core_clken	VDP 工作时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[4]	vdp_cfg_clken	VDP CFG (内部配置)时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[3]	vdac_pctrl	VDAC 时钟相位控制。 0：时钟不取反； 1：时钟取反。	1
[2]	vdp_out_pctrl	VDP 输出时钟(芯片输出时钟)相位控制。默认反向 0：时钟不取反； 1：时钟取反。	根据实际效果来调整，效果不对时尝试调整。

Bits	Name	Description	Suggestion
[1]	reserved	保留。	-
[0]	vdp_srst_req	VDP 软复位请求。 0：撤消复位； 1：复位。	0

表8-8 Hi3519V101 LCD 的 CRG 寄存器：PERI_CRG18 (0x12010048)

Bits	Name	Description	Suggestion
[31:28]	reserved	保留。	-
[27]	lcd_clken	LCD 分频器时钟门控配置寄存器。 0：关闭时钟； 1：打开时钟。	1
[26:0]	lcd_mclk_div	LCD 分频时钟，可配置。 假定目标频率 X(MHz)，则 lcd_mclk_div = (X/1188) * 2^27。	假定目标频率 X(MHz)，则 lcd_mclk_div = (X/1188) * 2^27。

在各个芯片的 VDP 的 CRG 配置中，都有一个分频比的配置，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，这时候分频比应该选择 3 分频。
- 如某款 16bit 串行的 LCD 屏，一个时钟节拍构造一个像素，则应该选择不分频。

8.2.4 确认 LCD_CTRL 寄存器配置

调试 LCD 屏时用户还需要配置一个 VDP 的寄存器 LCD_CTRL，在 VDP 的寄存器中的偏移为 0xD400。

表8-9 LCD_CTRL 寄存器配置

Bits	Name	Description	Suggestion
[31]	reserved	保留。	-
[30]	reserved	保留。	-
[29]	lcd_serial_mode	LCD 串行模式。 0：并行；	根据 LCD 接口类型选择。

Bits	Name	Description	Suggestion
		1：串行。	
[28]	lcd_serial_perd	LCD 串行输出单像素时钟周期数。 0：3 个周期； 1：4 个周期。	串行输出时选择，和上节中的分频比配置一致。
[27]	lcd_parallel_order	LCD 并行输出顺序。 0：RGB； 1：BGR。	并行输出的顺序，根据硬件连接选择。串行无需关注。
[26]	lcd_data_inv	LCD 输出 bit 反相。 0：高位到低位 15~0bit/23~0bit； 1：高位到低位 0~15bit/0~23bit。	根据硬件连接选择。
[25]	lcd_parallel_mode	LCD 并行输出位宽模式。 0：RGB565； 1：RGB888。	16bit 并行输出时为 0，24bit 并行输出时为 1.串行输出无需关注。
[24]	reserved	保留。	-
[23:0]	reserved	保留。	-

9 VO

9.1 VO 用户时序如何配置

【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  bSynm;     /* 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;   /* herizontal active area */
    HI_U16   u16Hbb;    /* herizontal back blank porch */
    HI_U16   u16Hfb;    /* herizontal front blank porch */
    HI_U16   u16Hmid;   /* bottom herizontal 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  bIdv;      /* inverse data valid of output */
    HI_BOOL  bIhs;      /* inverse horizontal synch signal */
    HI_BOOL  bIvs;      /* 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。
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。
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。
bIdv	数据有效信号的极性。目前不可调，默认高有效。
bIhs	水平有效信号的极性，配置 0 为高有效，配置 1 为低有效。
bIvs	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 Hi3516A 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.bIop = 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.bIdv = 0;
pstPubAttr->stSyncInfo.bIhs = 0;
pstPubAttr->stSyncInfo.bIvs = 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，对应于配置寄存器 0x20030034 的[13:12]为 10 即可。

实际需要根据对应芯片的芯片手册去配置 VO 时钟寄存器。

【注意事项】

此由于 u16Vbb 和 u16Vfb 的寄存器 bit 位宽为 8，所以在接口中这两个值最大为 256。其他参数的取值范围可参考《HiMPP IPC V2.0 媒体处理软件开发参考》或《HiMPP IPC V3.0 媒体处理软件开发参考》。

计算公式为：

（有效宽+水平后消隐+水平前消隐）*（有效高+垂直后消隐+垂直前消隐）*帧率=时钟

请根据此公式，自行配置水平和垂直的前后消隐长度来匹配设置的 VO 时钟。

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");
            }
    }
```

10 PCI

10.1 Hi3519V101 PCI 使用注意事项

目前 Hi3519V101 作为 PCIE 主片的功能未验证，Hi3519v101 mpp_big-little\ko 目录下的脚本 clkcfg_hi3519v101.sh 默认关闭了 PCIE 的时钟：

```
himm 0x120100b0 0x000001f0;
```

如果用户需要使用 Hi3519V101 实现 PCIE 的主片功能，请自行打开 PCIE 的时钟：

```
himm 0x120100b0 0x000000f0;
```

11 VENC

11.1 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。

12 HDMI

12.1 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。

13 其它

13.1 动态库

13.1.1 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。

13.1.2 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 lbupvqe.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。

13.1.3 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、hi35 xx_tde.ko、hi35 xx_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。

13.1.4 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：sensor_spi.ko 和 ssp_sony.ko（以 Sony 的 SPI 接口 sensor 为例）。

那么这两个有什么区别呢？

- 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。
- ssp_sony.ko 是我们自己编写的 SPI 驱动，非标准的，但是能够解决配置不及时的问题。

综上所述，如果追求配置速度，请使用 ssp_sony.ko。如果要使用内核标准 SPI 驱动，请使用 sensor_spi.ko。另外，其他外设需要用到 SPI 驱动的，可以参考上述区别进行选择，也可参考 ssp_sony.ko 驱动进行 SPI 驱动的个性化改动。

13.1.5 load 脚本说明

对于 Linux 系统，load 脚本可以通过传入参数-osmem 和-total 来指定运行的 OS 内存和 Total 内存大小（MMZ 内存=Total 内存-OS 内存）。但-osmem 和-total 的大小需要用户保证其合理配置，比如不能超过所使用的硬件单板上 DDR 内存大小。