



HiMPP V4.0 媒体处理软件

FAQ

文档版本 01
发布日期 2018-12-12

版权所有 © 深圳市海思半导体有限公司 2018。保留一切权利。

未经本公司书面许可，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 V4.0 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

- 未有特殊说明，Hi3516EV300、Hi3518EV300 与 Hi3516EV200 完全一致。
- 未有特殊说明，Hi3559AV100 和 Hi3559CV100 完全一致。
- 未有特殊说明，Hi3556V200, Hi3559V200, Hi3516DV300 和 Hi3516CV500 完全一致。
- 未有特殊说明，Hi3556AV100 和 Hi3519AV100 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
Hi3559A	V100
Hi3559C	V100
Hi3519A	V100
Hi3556A	V100
Hi3516C	V500
Hi3516D	V300
Hi3559	V200
Hi3556	V200
Hi3516E	V200
Hi3516E	V300
Hi3518E	V300








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）。
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 01 (2018-12-12)

第 1 次版本发布。



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
1.3 性能相关.....	9
1.3.1 CPU 性能 Top 统计波动大问题.....	9
1.3.2 绑定中断到不同 CPU 的注意事项.....	9
1.3.3 解码性能.....	9
1.4 小型化.....	9
1.4.1 静态库使用.....	9
1.5 管脚复用、时钟门控、系统控制在哪儿配置？.....	10
2 MIPI 配置.....	11
2.1 MIPI 频率说明.....	11
2.2 时序.....	11
3 VI 功能.....	12
3.1 DIS 功能.....	12
3.1.1 如何实现 ISP-DIS 场景功能.....	12
3.2 WDR 功能.....	12
3.2.1 QudraWDR 注意事项.....	12
3.3 VI 时序配置.....	12
3.3.1 BT.1120.....	12
3.3.2 BT.656.....	15
3.3.3 BT.601.....	17
3.3.4 MIPI_YUV.....	20
3.4 VI 中断类型.....	22
3.4.1 FRAME_INTERRUPT_START 类型.....	22
3.4.2 FRAME_INTERRUPT_EARLY 类型.....	23
3.4.3 FRAME_INTERRUPT_EARLY_END 类型.....	23



4 FISHEYE 功能	24
4.1 鱼眼矫正功能	24
4.1.1 Hi3559AV100/Hi3556AV100 如何实现多于 4 宫格鱼眼矫正功能	24
5 LDC 功能	25
5.1 畸变矫正功能	25
5.1.1 VI 和 VPSS LDC 对比	25
6 音频	26
6.1 PC 如何播放由 Hisilicon 编码的音频码流	26
6.1.1 PC 如何播放由 Hisilicon 编码的音频 G711/G726/ADPCM 码流	26
6.2 Hisilicon 如何播放标准的音频码流	27
6.2.1 Hisilicon 如何播放标准的音频 G711/G726/ADPCM 码流	27
6.3 为什么使能 VQE 后会有高频部分缺失	30
6.3.1 为什么使能 VQE 后会有高频部分缺失	30
6.4 音频内置 CODEC 输出(AO 输出)出现幅频响应异常	31
7 低功耗	32
7.1 低功耗模块动态调频可能频繁调频的问题	32
8 LCD 屏幕调试	33
8.1 支持哪些 LCD 屏幕	33
8.2 LCD 屏幕调试顺序	33
8.2.1 确认管脚复用配置	33
8.2.2 确认用户时序	33
8.2.3 配置设备帧率	35
8.2.4 确认时钟信息	35
9 VO	36
9.1 VO 用户时序如何配置	36
9.2 画面切换	38
9.2.1 通道属性发生变化	38
9.2.2 建议的实现方式	38
9.3 视频同步方案	40
9.3.1 实现原理	40
9.3.2 建议的操作步骤	41
10 VENC	42
10.1 JPEG 量化表配置注意事项	42
11 HDMI	43
12 其它	44
12.1 动态库	44



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 44

12.1.2 为什么使用 libupvqe.a 和 libdnvqe.a 动态编译时出现重定义..... 44

12.1.3 模块 KO 之间的依赖关系 46

12.1.4 SPI 驱动说明..... 46



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

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，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《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。

- 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 V4.0 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_SetConfig。

参考《HiMPP V4.0 媒体处理软件开发参考》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。

- VI 模块配置：

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

措施	相关模块参数/接口	收益	影响	注意
Raw 压缩	HI_MPI_VI_CreatePipe： enCompressMode	比不压缩节省内存和带宽	-	Hi3516EV200 不支持
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
3DNR 压缩	HI_MPI_VI_CreatePipe： bNrEn 和 stNrAttr	比不压缩节省内存和带宽	-	仅 Hi3559AV100/Hi3559CV100 支持



措施	相关模块参数/接口	收益	影响	注意
Early_End 机制	HI_MPI_VI_SetPipeFrameInterruptAttr	调整得当可节省一帧 buffer	调整不当可能出现图像效果异常	与 sensor 时序强相关

• VPSS 模块配置:

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

措施	相关模块参数/接口	收益	影响	注意
3DNR 压缩	HI_MPI_VPSS_CreateGroup: stNrAttr	比不压缩节省内存和带宽	-	Hi3559AV100、Hi3559CV100 不支持
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_SetLowDelayAttr 单 buffer 模式: HI_MPI_VPSS_SetModParam: bOneBufForLowDelay	可节省一帧 buffer	行号设置不合理可能出现图像效果异常	-
VPSS-VENC 低延时卷绕	HI_MPI_VPSS_SetChnBufWrapAttr HI_MPI_SYS_GetVPSSVENCWrapBufferLine	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/Hi3516EV300/Hi3518EV300 支持
输入输出 buffer 复用	同时满足以下条件时自适应复用: 1) VIPROC-VPSS 离线	可节省一帧 buffer	-	仅 Hi3516EV200/Hi3516EV300/Hi3518EV300 支持



措施	相关模块参数/接口	收益	影响	注意
	2) 开启 3DNR 3) 开启 CH0 且不放大 4) 不开启 GROUP 裁剪 5) 不开启 flip 6) 不分块（具体分块分界点参考各个芯片的 VPSS 手册描述） 7) 不开启 CH0 卷绕（仅 Hi3516EV200 支持）			
Early_End 机制	HI_MPI_VPSS_SetGrpFrameInterruptAttr	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
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 支持
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 支持
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

• VENC 模块配置：

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

措施	相关模块参数/接口	收益	影响	注意
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



措施	相关模块参数/接口	收益	影响	注意
				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。	
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 I 帧	Hi3559AV100、Hi3519AV100 不支持
动态回收参考帧 buffer	HI_MPI_VENC_SetModParam: 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	-	-
编码模块支持的最大通道数	Linux: VencMaxChnNum Huawei LiteOS: VENC_MODULE_PARAMS 中的 u32VencMaxChnNum	可节省部分 OS 内存	-	-
相同分辨率多通道编码使用 UserVB 模式节省内存	HI_MPI_VENC_SetModParam: enH264eVBSource enH265eVBSource	比 PrivateVB 模式节省更多帧存 buffer	-	-

• VDEC 模块配置:

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

措施	相关模块参数/接口	收益	影响	注意
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。	-	-
按场景设置解码器最大能力集	HI_MPI_VDEC_SetModParam: stVideoModParam	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
按场景设置解码通道数	HI_MPI_VDEC_SetProtocolParam	可节省部分 OS	-	-



措施	相关模块参数/接口	收益	影响	注意
力集		内存		
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	-	-
解码模块最大支持的通道数	Linux: VdecMaxChnNum VfmwMaxChnNum Huawei LiteOS: VDEC_MODULE_PARAMS_S VFMW_MODULE_PARAMS_S	可节省部分 OS 内存	-	-
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。

• AVS 模块配置：

参考文档：《HiMPP V4.0 媒体处理软件开发参考》中的“拼接”章节。

措施	相关模块参数/接口	收益	影响	注意
按场景合理设置 WorkingSet 大小	HI_MPI_AVS_SetModParam: u32WorkingSetSize	可节省部分 MMZ 内存	设置小了对图像效果有影响	-

• VGS 模块配置：

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



措施	相关模块参数	收益	影响	注意
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	-
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	-
设置是否支持 HDR 功能	bVgsHdrSupport	可节省 HDR buffer		仅 Hi3559AV100 支持

• GDC 模块配置：

- 参考文档：《HiMPP V4.0 媒体处理软件开发参考》中的“几何畸变矫正子系统”章节。
- 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
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
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
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 GDC 性能	Hi3559AV100/Hi3519AV100 默认值 200，其它芯片默认值 64

• VO 模块配置：

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

措施	相关接口	收益	影响	注意
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
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
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



措施	相关接口	收益	影响	注意
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

- HIFB 模块配置：

参考文档：《HiFB 开发指南》和《HiFB API 参考》。

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	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	-
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 V4.0 媒体处理软件开发参考.pdf》“音频”章节。

措施	相关接口	收益	影响	注意
按场景合理设置 AI 缓存音频帧大小	HI_MPI_AI_SetPubAttr	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。	-
按场景合理设置 AENC 缓存音频帧大小	HI_MPI_AENC_CreateChn	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。	-
按场景合理设置 ADEC 缓存音频帧大小	HI_MPI_ADEC_CreateChn	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。	-
按场景合理设置 AO 缓存音频帧大小	HI_MPI_AO_SetPubAttr	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。	-



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。

1.4 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。



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：

$$VI_Freq * Pix_Width = Lane_Num * MIPI_Freq$$

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

2.2 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。



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 QuadraWDR 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，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

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 Hi3559V100 举例说明。



Hi3559V100 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

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 Hi3559V100 举例说明。

Hi3559V100 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

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 Hi3559V100 举例说明。

Hi3559V100 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}  
};
```

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。

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。
- 后端模块的处理赶不上采集的速度可能导致图像花屏，甚至逻辑挂死。
- 由于帧压缩是整帧压缩，不能按行解压，因此不能使用帧压缩。
- 拍照通路不能使用。

调试方法：u32EarlyLine 行设置较小的时候，帧未采集完成，发送给后端模块处理，需要调整 u32EarlyLine 使后端模块的处理赶不上采集的速度，否则可能导致图像花屏，甚至逻辑挂死，例如 VI 离线的环境下，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，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。

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，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。



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 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。

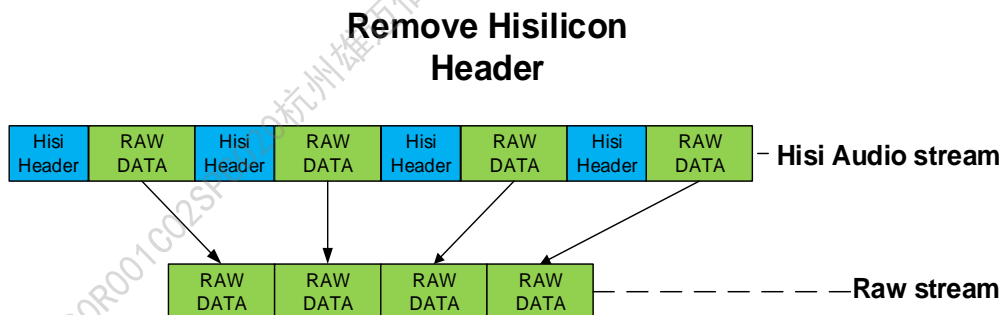
【分析】

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：《HiMPP V4.0 媒体处理软件开发参考》音频章节 9.2.2.3 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 rfc3551 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](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	$N \times 40$	N 为[1,5]的任意正整数
G726-16kbps	$N \times 10$	N 为[1,5]的任意正整数
G726-24kbps	$N \times 15$	N 为[1,5]的任意正整数
G726-32kbps	$N \times 20$	N 为[1,5]的任意正整数
G726-40kbps	$N \times 25$	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)

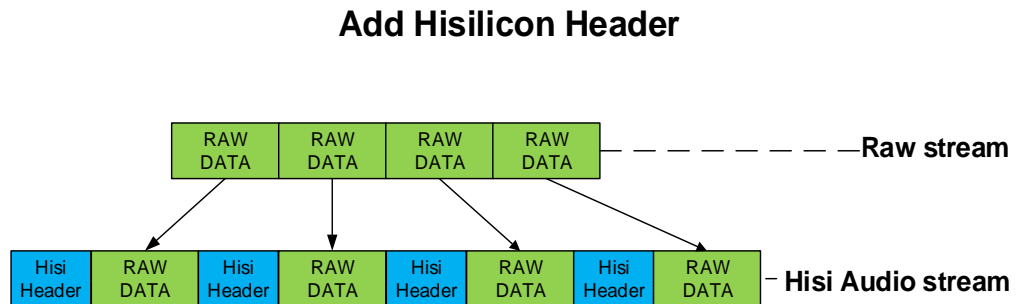


说明

- 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

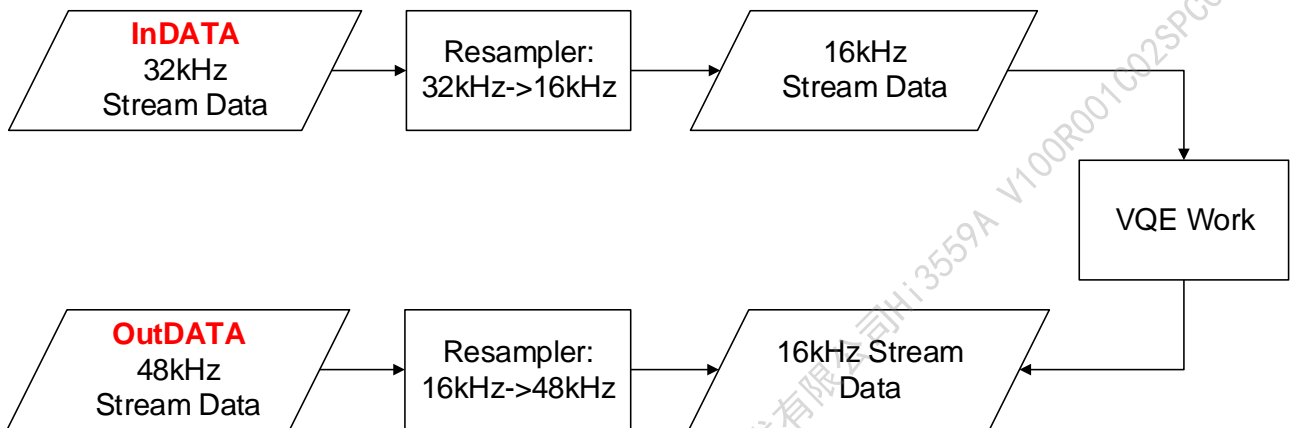
【现象】

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



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 V4.0 媒体处理软件开发参考》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_MPL_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 8-1 所示。

表8-1 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。配置 0 为高有效，配置 1 为低有效。



参数名称	说明
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 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 V4.0 媒体处理软件开发参考》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 V4.0 媒体处理软件开发参考》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 u32Fbdiv、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

【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;      /* 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 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。配置 0 为高有效，配置 1 为低有效。
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 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.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。

计算公式为：

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

9.2 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



注意

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致 JPEG 图片容量变大，用户需要权衡图像质量和 JPEG 图像容量。



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，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。