HiMPP 媒体处理软件

FAQ

文档版本 04

发布日期 2016-02-15

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。**

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

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 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。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》，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》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.pdf》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.pdf》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	HI_MPI_VPSS_SetRefSelect HI_MPI_VPSS_GetRefSelect	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.pdf》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
动态切换编码分辨率	HI_MPI_VENC_GetChnAttr, HI_MPI_VENC_SetChnAttr	切换编码分辨率时不销毁通道，减少内存碎片。	无	切换分辨率后所有参数恢复默认值。
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 V2.0 媒体处理软件开发参考.pdf》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.pdf》。

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】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 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。

1.4 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。

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。

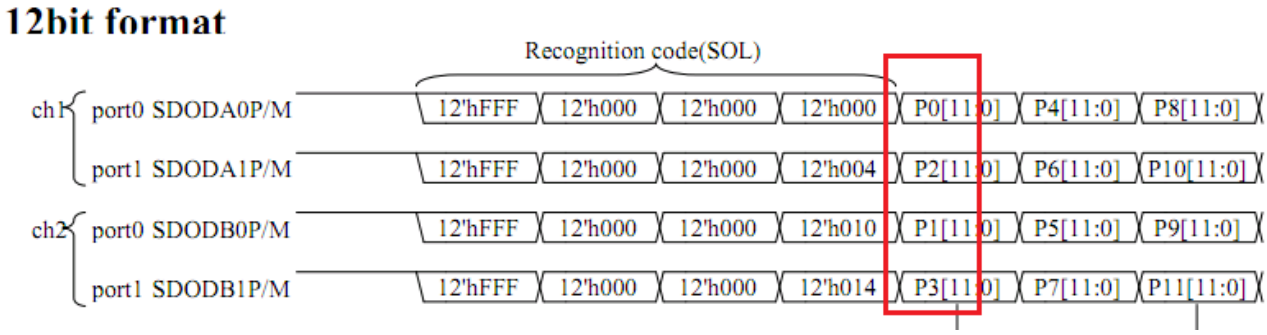
下面以 demo 板 mn34220 为例进行说明：

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。

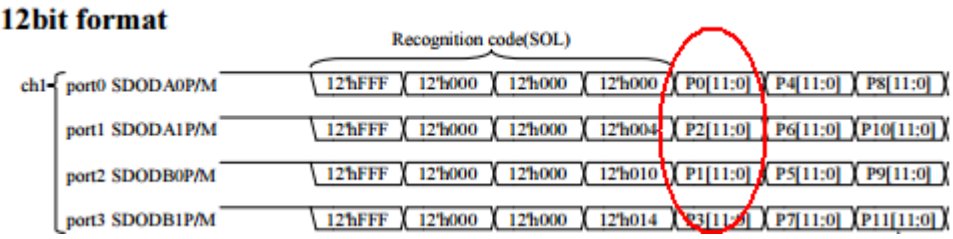
两个 Link 之间乱序时，仍然遵循上述准则，即在同一 Link 内保持正常顺序配置。Panasonic 的 Sensor 普遍存在两个 Link 之间乱序，其他厂家暂不涉及。

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：



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}},
```

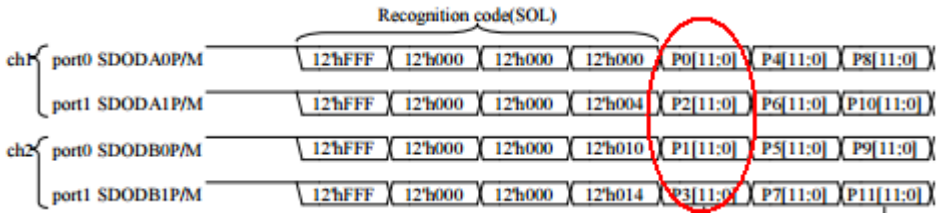
```
    {{0x016, 0x017, 0x014, 0x015}, //PHY0_lane3
    {0x216, 0x217, 0x214, 0x215},
    {0x116, 0x117, 0x114, 0x115},
    {0x316, 0x317, 0x314, 0x315}},

    {{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}}
              }
```

2.2 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。

3 VI 功能

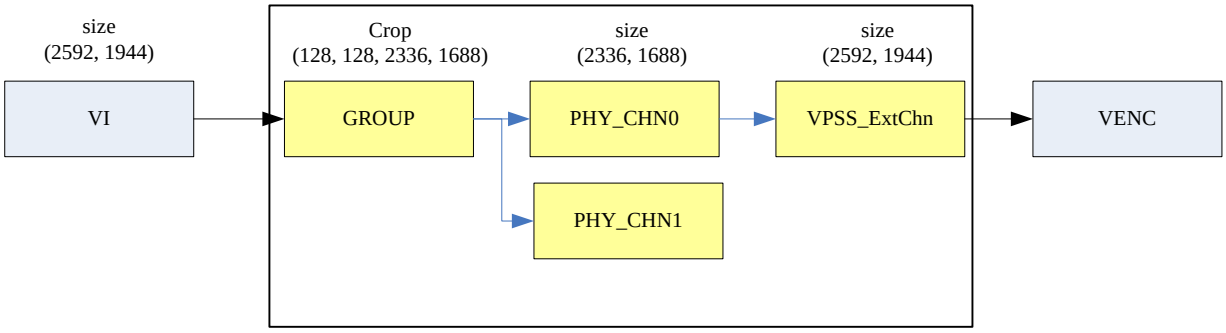
3.1 DIS 功能

3.1.1 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。

4 音频

4.1 PC 如何播放由 Hisilicon 编码的音频码流

4.1.1 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。

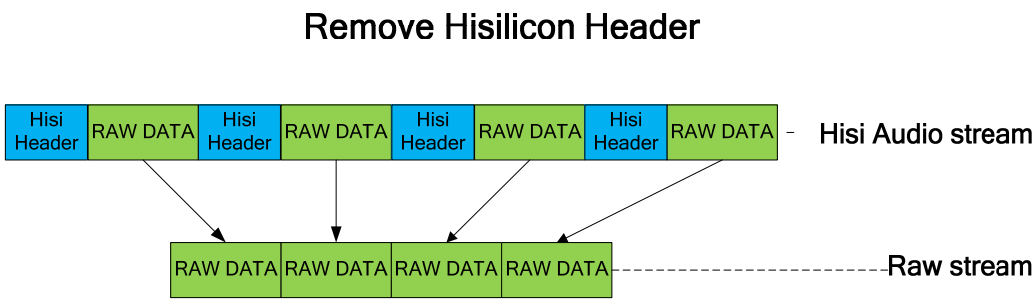
【分析】

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 IPC V2.0 媒体处理软件开发参考.pdf》音频章节 9.2.2.3 海思语音帧结构，Huawei LiteOS 请见《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 4-1 所示。

图4-1 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

4.2 Hisilicon 如何播放标准的音频码流

4.2.1 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4-1 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)

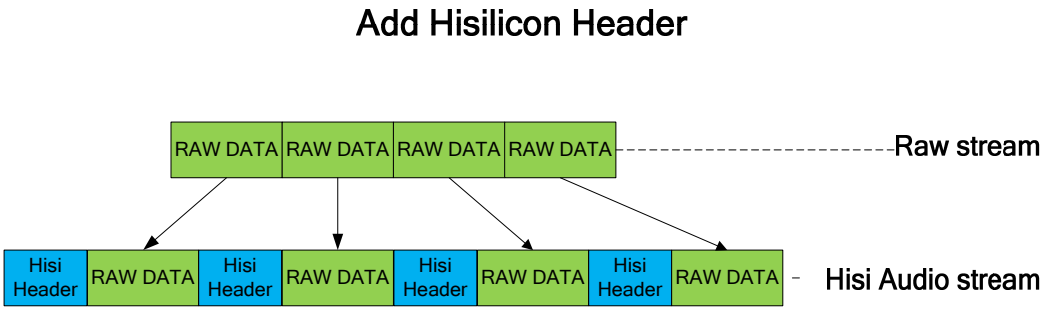
说明

- 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 4-2 所示：

图4-2 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;
}
```

4.3 为什么使能 VQE 后会有高频部分缺失

4.3.1 为什么使能 VQE 后会有高频部分缺失

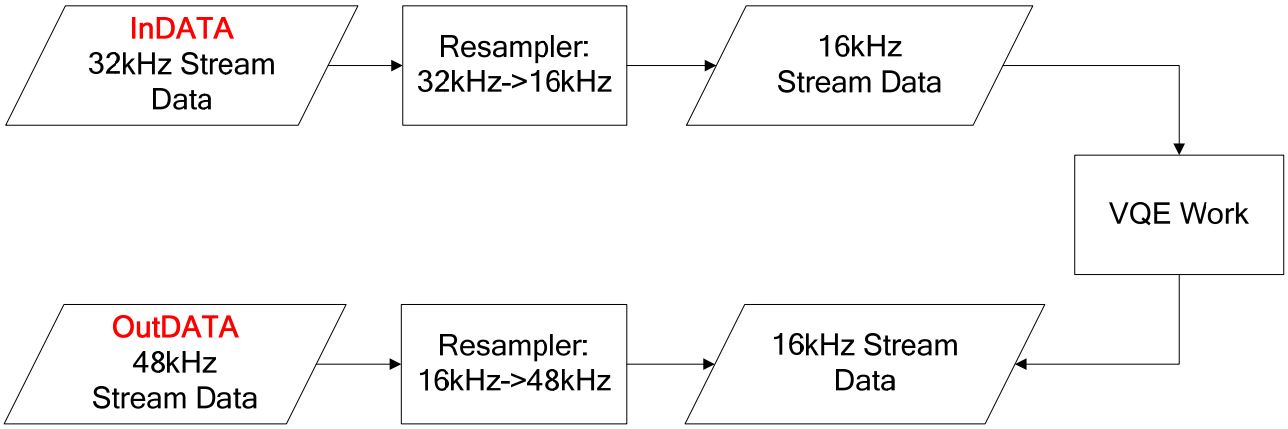
【现象】

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 4-3 所示。

图4-3 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。

4.4 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。

5 其它

5.1 动态库

5.1.1 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：

a. 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。

b. 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。

c. 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。

5.1.2 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：

a. 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；

b. 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；

c. 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。