



Hi3507 媒体处理软件

开发指南

文档版本 01

发布日期 2011-07-12

版权所有 © 深圳市海思半导体有限公司 2011。保留一切权利。

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

商标声明



HISILICON、海思和其他海思商标均为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

本文描述了如何使用 Hi3507 媒体处理平台开发视频预览、视频编码、视频解码、音频编码和解码等业务的主要流程和步骤。为系统工程师和软件工程师提供业务规划和流程指引。

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
Hi3507 芯片	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。



符号	说明
 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。

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。



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 (2011-06-16)

第 1 次正式发布。



目 录

前 言.....	i
1 概述.....	1
1.1 应用架构.....	1
1.2 Hi3507 媒体处理软件平台.....	2
2 第一次开发应用.....	3
2.1 开发前介绍.....	3
2.1.1 更新开发环境.....	3
2.1.2 使用加载 MPP 平台内核模块.....	3
2.1.3 MPP 平台系统初始化、系统去初始化.....	3
2.1.4 典型业务资源配置.....	4
2.2 实例介绍.....	7
2.2.1 实例操作场景.....	7
2.2.2 实例目的.....	8
2.3 设计.....	8
2.3.1 工作流程.....	8
2.3.2 资源需求.....	9
2.3.3 编码准备.....	10
2.4 编码.....	11
2.5 编译和运行.....	12
2.5.1 代码编译.....	12
2.5.2 运行和调试.....	13
3 模块开发指引.....	14
3.1 VI 模块.....	14
3.1.1 概述.....	14
3.1.2 重要概念.....	15
3.1.3 配置并启动 VI 捕获.....	15
3.2 VO 模块.....	17
3.2.1 概述.....	17
3.2.2 重要概念.....	19



3.2.3 配置并启动 VO 输出.....	19
3.3 VENC 模块.....	21
3.3.1 概述.....	21
3.3.2 重要概念.....	22
3.3.3 配置并启动视频编码.....	22
3.3.4 停止并销毁视频编码.....	25
3.3.5 按帧获取编码码流.....	28
3.3.6 按包获取编码码流.....	32
3.3.7 配置并启动主次码流编码.....	36
3.3.8 停止并销毁主次码流编码.....	39
3.3.9 获取多路编码码流.....	40
3.3.10 按轮循方式抓拍.....	43
3.3.11 按并发方式抓拍.....	47
3.4 VDEC 模块.....	51
3.4.1 概述.....	51
3.4.2 重要概念.....	52
3.4.3 配置并启动解码器.....	52
3.4.4 向解码通道发送码流.....	54
3.4.5 VDEC 通道绑定 VO 直接输出解码后图像.....	57
3.4.6 获取解码后图像送 VO 显示.....	58
3.4.7 停止并销毁视频解码.....	60
3.5 VPP 模块.....	61
3.5.1 概述.....	61
3.5.2 重要概念.....	62
3.6 MD 模块.....	62
3.6.1 概述.....	62
3.6.2 重要概念.....	63
3.6.3 配置并启动 MD.....	63
3.6.4 获取 MD 结果数据.....	65
3.7 AUDIO 模块.....	69
3.7.1 概述.....	69
3.7.2 重要概念.....	70
3.7.3 启用 AI 设备并获取音频帧.....	72
3.7.4 启用 AO 设备并发送音频帧.....	75
3.7.5 创建 AENC 通道和发送音频帧编码并获取码流.....	77
3.7.6 创建 ADEC 通道和发送码流解码并获取解码数据.....	80
3.8 音视频复合模块.....	82
3.8.1 概述.....	82
3.8.2 运作机制.....	83



3.8.3 开展业务和应用设计	83
3.8.4 获取音视频复合数据	84
4 视频综合业务开发指引	87
4.1 视频预览	87
4.1.1 1 路 D1 单通道预览业务	87
4.1.2 多画面预览业务	89
4.2 视频编码	90
4.2.1 双码流业务	90
4.2.2 抓拍业务	92
4.2.3 请求 I 帧业务	94
4.2.4 插入用户数据业务	96
4.3 视频解码	98
4.4 运动侦测	100
4.5 视频遮挡	101
4.6 OSD 区域	103
5 音频综合业务开发指引	107
5.1 音频采集及编码业务	107
5.2 音频回声抵消业务	109
5.3 音频解码及输出业务	111
6 FAQ	115
6.1 如何获取当前 MPP 平台的版本号	115
6.2 视频编码	115
6.3 音频	116
6.4 视频解码	119
6.5 如何配置 MPP 视频缓存池	120
6.6 如何查看 log 日志	122
7 Proc 调试信息说明	123
7.1 概述	123
7.2 SYS	124
7.3 VB	125
7.4 LOG	127
7.5 CHNL	128
7.6 VI	130
7.7 VO	133
7.8 DSU	135
7.9 VENC	136
7.10 GROUP	138
7.11 VPP	141



7.12 MD	143
7.13 VDEC	146
7.14 AI	148
7.15 AO	150
7.16 H264E	151
7.17 H264D	156
7.18 JPEGE	159



插图目录

图 1-1 典型 Hi3507 应用架构图.....	1
图 2-1 1 路 D1 预览的工作流程.....	9
图 2-2 1 路 D1 预览业务的任务分解.....	11
图 3-1 配置并启动 VI 捕获的业务流程.....	15
图 3-2 配置并启动 VO 输出的业务流程.....	19
图 3-3 配置并启动视频编码的业务流程.....	23
图 3-4 强行停止并销毁视频编码的业务流程.....	25
图 3-5 等待编码完成销毁视频编码的业务流程.....	26
图 3-6 按帧获取 1 路编码后码流的业务流程.....	29
图 3-7 按包非阻塞获取码流工作流程.....	32
图 3-8 按包阻塞获取码流工作流程.....	33
图 3-9 按 select 获取码流工作流程.....	34
图 3-10 配置并启动主次码流编码的业务流程.....	37
图 3-11 强行停止并销毁主次码流编码的业务流程.....	39
图 3-12 select 多路码流工作流程.....	41
图 3-13 按轮循方式启动 1 路抓拍的业务流程.....	44
图 3-14 按并发方式启动 1 路抓拍的业务流程.....	48
图 3-15 配置并启动解码器的业务流程.....	52
图 3-16 阻塞方式发送码流包的业务流程.....	54
图 3-17 非阻塞方式发送码流包的业务流程.....	55
图 3-18 VDEC 通道绑定 VO 直接输出解码后图像的业务流程.....	57
图 3-19 获取解码后图像送 VO 显示的业务流程.....	58
图 3-20 强行停止并销毁视频解码的业务流程.....	60
图 3-21 等待解码完成销毁视频解码的业务流程.....	60
图 3-22 配置并启动 MD 的业务流程.....	63



图 3-23 获取 MD 结果的业务流程	65
图 3-24 Codec 数据在 SIO 传输通路的排列	71
图 3-25 8bit 采样精度下 SIO 传输通路中通道排列	72
图 3-26 16bit 采样精度下 SIO 传输通路中通道排列	72
图 3-27 32bit 采样精度下 SIO 传输通路中通道排列	72
图 3-28 启用 AI 设备并获取音频帧数据的业务流程.....	73
图 3-29 启用 AO 设备并发送音频帧数据的业务流程	75
图 3-30 创建 AENC 通道和发送音频帧编码并获取码流的业务流程	78
图 3-31 创建 ADEC 通道和发送码流解码并获取解码数据的业务流程	80
图 3-32 音视频复合模块运作机制.....	83
图 3-33 获取音视频复合码流的业务流程.....	85
图 4-1 视频业务运作机制.....	87
图 4-2 1 路 D1 单通道预览的业务流程.....	88
图 4-3 多画面预览显示的业务流程.....	89
图 4-4 双码流的业务流程.....	91
图 4-5 抓拍的业务流程	93
图 4-6 请求 I 帧的业务流程	95
图 4-7 插入用户数据的业务流程.....	97
图 4-8 H.264 单通道解码回放业务流程.....	99
图 4-9 移动侦测的业务流程.....	100
图 4-10 单通道视频遮挡区域的业务流程.....	102
图 4-11 单通道 H.264 编码并开启 OSD 的业务流程	104
图 5-1 音频模块运作机制.....	107
图 5-2 音频采集及编码的业务流程.....	108
图 5-3 音频回声抵消的业务流程.....	110
图 5-4 音频解码及输出的业务流程.....	112
图 6-1 公共缓存池和私有缓存池.....	121



表格目录

表 2-1 典型业务资源配置.....	5
表 2-2 1 路 D1 预览需要的资源以及头文件.....	9
表 3-1 常用的 VI 通道属性配置（BT656 PAL 制）	16
表 6-1 解码输出方式参数配置.....	120

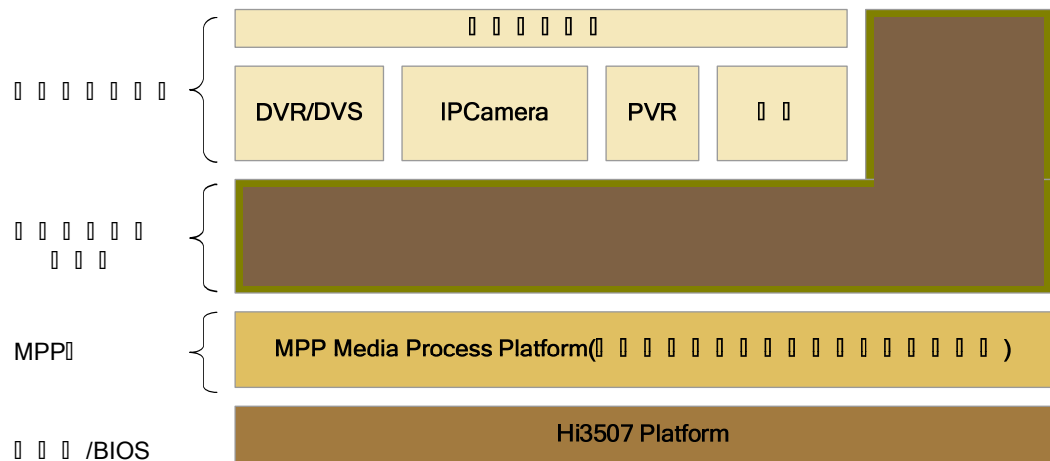


1 概述

1.1 应用架构

Hi3507 典型应用架构如图 1-1 所示。

图1-1 典型 Hi3507 应用架构图



软件架构主要包含以下几层：

- 驱动层/BIOS
设备驱动层与硬件密切相关，驱动 Hi3507 芯片完成其基本业务功能。
- MPP 层（Media Process Platform）
MPP 层基于驱动层，屏蔽芯片和硬件平台的差别，提供面向媒体业务的基本功能。MPP 的编程接口称为 MPI（MPP Programe Interface）。
- 产品解决方案中间件
中间件针对产品形态提供一些组件支持，如文件封装、PC 端的配套组件等。
- 产品解决方案层



各种产品形态，如 DVR（Digital Video Recorder）、PVR（Personal Video Recorder）和 DVS（Digital Video Server）等的产品解决方案使用 MPI 完成其相应业务组合。

1.2 Hi3507 媒体处理软件平台

Hi3507 媒体处理软件平台屏蔽了 Hi3507 芯片媒体处理硬件的操作细节，实现对音视频数据的采集、编码、同步，同时支持音视频码流解码输出等处理，对上层应用软件提供 MPI 接口，可划分为如下几个功能模块：

- VI 模块
捕获经摄像头采集或者视频传感器采集的视频数据，放入指定的存储空间。
- VO 模块
将视频图像发送到视频 DA 芯片，在显示终端显示视频图像。
- VENC 模块
将 VI 模块捕获的多路图像分别编码，提供编码通道的管理、码流获取以及图像抓拍功能。
- VDEC 模块
接收用户发送的码流，输出解码图像。
- VPP 模块
在编码前，将 VI 捕获的图像做处理，包括遮挡图像的某些区域或在图像中做视频叠加（例如在码流中加入通道号、时间、地点等信息）。
- MD 模块
针对编码通道，提供移动侦测信息的功能。
- AUDIO 模块
提供音频输入、音频输出、音频编码和音频解码的功能。
- 音视频复合模块
提供对音频和视频码流数据的同步输出功能。



2 第一次开发应用

2.1 开发前介绍

2.1.1 更新开发环境

具体请参见《Hi3507 Linux 开发环境 用户指南》。

2.1.2 使用加载 MPP 平台内核模块

在 SDK 根目录的“mpp\ko.rel”和“mpp\ko.dbg”两个目录下分别存放 MPP 平台 release、debug 版本的内核模块（以下简称 ko），同时 load3511 的脚本存放在“mpp\ko.rel”和“mpp\ko.dbg”两个目录下，该脚本提供一个加载 ko 模块的示例，可根据产品需要修改脚本。

如果需要定制脚本时需要注意以下问题：

- 必须首先加载 hi3511_base.ko，其他的 hi3511_XXX.ko 都是依赖于此模块。
- hi3511_vo.ko 依赖于 tde.ko，因此在加载 hi3511_vo.ko 前，必须先加载 tde.ko。
- 如果不需要 JPEG 的解码功能，则可以不加载 Hi3511_JPEG.ko，在 load3511 的脚本示例中就没有加载 Hi3511_JPEG.ko。
- 在 load3511 的脚本示例中，默认把所有外围驱动（主要是音视频的 AD、DA 等驱动）放在文件系统中。

2.1.3 MPP 平台系统初始化、系统去初始化

单板上电后，启动进入内核，加载 MPP 系统中的各个 ko 以及外围芯片驱动，应用程序启动 MPP 业务前，必须完成 MPP 系统初始化工作。同理，应用程序退出 MPP 业务后，也要完成 MPP 系统去初始化工作，释放资源。

MPP 系统初始化，主要针对视频缓存池、系统控制两大部分进行初始化。同时提供当前 MPP 系统的版本信息，便于产品的维护、更新。

视频缓存池

视频缓存池的功能：主要向媒体业务提供大块物理内存，负责内存的分配和回收，充分发挥内存缓存池的作用，让物理内存资源在各个媒体处理模块中合理使用。一组大



小相同、物理地址连续的缓存块组成一个缓存池。在创建编码通道或者解码通道时，MPP 系统会为该通道创建一个独享缓存池。在销毁通道时，同时也会销毁相应的缓存池。而对于视频输入通道，则需要使用公共缓存池。所有的视频输入通道都可以从公共缓存池中获取缓存块。由于视频输入通道不存在创建和销毁，因此，在系统初始化之前，必须为视频输入通道配置公共缓存池。根据业务的不同，公共缓存池的数量、缓存块的大小和数量会有所不同。具体的配置请参考“[2.1.4 典型业务资源配置](#)”。

视频缓存池主要提供以下功能函数：

- HI_MPI_VB_SetConf: 配置视频缓存池。
- HI_MPI_VB_GetConf: 获取视频缓存池配置。
- HI_MPI_VB_Init: 初始化视频缓存池。
- HI_MPI_VB_Exit: 去初始化视频缓存池。
- HI_MPI_VB_CreatePool: 创建视频缓存池。
- HI_MPI_VB_DestroyPool: 销毁视频缓存池。
- HI_MPI_VB_GetBlock: 获取一个缓存块。
- HI_MPI_VB_ReleaseBlock: 释放一个已经获取的缓存块。
- HI_MPI_VB_Handle2PhysAddr: 获取一个缓存块的物理地址。
- HI_MPI_VB_Handle2PoolId: 获取一块帧缓存的缓存池 ID。

视频缓存池的具体使用请参见《Hi3507 媒体处理软件 开发参考》。

系统控制

系统控制的功能：主要根据 Hi3507 芯片特性，完成硬件各个部件的复位、基本初始化工作，同时负责完成 MPP 系统各个业务模块的初始化、去初始化以及管理 MPP 系统各个业务模块的工作状态。

系统控制主要提供以下功能函数：

- HI_MPI_SYS_SetConf: 配置系统控制参数。
- HI_MPI_SYS_GetConf: 获取系统控制参数。
- HI_MPI_SYS_Init: MPP 系统初始化。
- HI_MPI_SYS_Exit: MPP 系统去初始化。

系统控制的具体使用请参见《Hi3507 媒体处理软件 开发参考》。

获取系统版本号

通过调用函数 HI_MPI_SYS_GetVersion 可以获取当前 MPP 系统的版本信息。

2.1.4 典型业务资源配置

DVR (Digital Video Recorder) 各种典型业务的相关资源配置如[表 2-1](#) 所示。该配置是针对单片上的业务，包括视频预览、视频编解码、运动检测、视频叠加、音频编解码的业务同时运行需要的资源。



表2-1 典型业务资源配置

典型业务	保留内存大小（推荐）	视频缓存池的配置 输入图像为 YUV420 时 S = 1.5 输入图像为 YUV422 时 S = 2
单片 8CIF 编码、8QCIF 编码（双码流） + 1CIF 解码 + 1D1 和 1/9D1 预览 + 8 路运动检测 + 8 路音频编码 + 1 路双向语音对讲	36MB 左右	<pre> /*D1（用于全屏预览）*/ astCommPool[0] = { u32BlkSize = 768*576*S,/*Strde may be 704*/ u32BlkCnt = 6*1, /*1 chn*/ } /*2CIF（用于四分屏预览）*/ astCommPool[1] = { u32BlkSize = 384*576*S, u32BlkCnt = 5*4, /*4 chn*/ } /*CIF*/ astCommPool[2] = { u32BlkSize = 384*288*S, u32BlkCnt = 8*8, /*8 chn*/ } /*QCIF(主次码流时可省略)*/ astCommPool[3] = { u32BlkSize = 192*144*S, u32BlkCnt = 3*8, /*8 chn*/ } </pre>



典型业务	保留内存大小（推荐）	视频缓存池的配置 输入图像为 YUV420 时 S = 1.5 输入图像为 YUV422 时 S = 2
4 Half D1 编码 + 4QCIF 编码 + 1CIF 解码 + 1D1 预览 + 4 路音频编码	30MB 左右	<pre> /*D1（用于全屏预览）*/ astCommPool[0] = { u32BlkSize = 768*576*S,/*Strde may be 704*/ u32BlkCnt = 6*1, /*1 chn*/ } /*HalfD1*/ astCommPool[0] = { u32BlkSize = 768*288*S,/*Strde may be 704*/ u32BlkCnt = 8*4, /*4 chn*/ } /*QCIF*/ astCommPool[1] = { u32BlkSize = 192*144*S, u32BlkCnt = 8*4, /*4 chn*/ } </pre>
2D1 编码 + 2CIF 编码 + 2 路运动检测 + 2 路音频编码	30MB 左右	<pre> /*D1*/ astCommPool[0] = { u32BlkSize = 768*576*S,/*Strde may be 704*/ u32BlkCnt = 8*2,/*2 chn*/ } /*CIF(主次码流时可省略)*/ astCommPool[1] = { u32BlkSize = 384*288*S, u32BlkCnt = 6*2, /*2 chn*/ } </pre>



典型业务	保留内存大小（推荐）	视频缓存池的配置 输入图像为 YUV420 时 S = 1.5 输入图像为 YUV422 时 S = 2
2VGA 编码 + 2QVGA 编码 + 2 路运动检测 + 2 路音频编码 + 1 路语音对讲	30MB 左右	<pre>/*VGA*/ astCommPool[0] = { u32BlkSize = 640*480*S, u32BlkCnt = 2*6, /*6 chn*/ } /*QVGA（主次码流时可省略）*/ astCommPool[1] = { u32BlkSize = 320*240*S, u32BlkCnt = 2*6, /*6 chn*/ }</pre>
1 路 130 万像素编码	16MB 左右	<pre>u32MaxPoolCnt = 16 astCommPool[0] = { u32BlkSize = 1280*1024*S u32BlkCnt = 1*6, /*1 chn*/ }</pre>
无输入，仅作为 2 路 D1 解码器且有 VO 预览	16MB 左右	/*不需配置公共缓存池*/

2.2 实例介绍

以开发 1 路 D1 预览为例，介绍基于 MPP 平台的开发业务过程。本节主要介绍实例的操作场景和操作目的。

2.2.1 实例操作场景

应用场景

VI 采集图像数据并直接通过 VO 输出到显示终端，完成当前图像数据的显示。具体场景为：观看摄像头采集的实时图像场景。

操作入口

该业务的操作入口在媒体处理 MPI 中，应用产品可根据实际情况开发操作入口，也可提供命令行或者基于图形界面的接口。



说明

在实例中直接由应用程序启动 1 路 D1 预览业务。



操作过程

直接运行实例应用程序。

操作结果

在显示终端（电视机或监视器）观看到 VI 采集的图像数据，图像大小 D1。

2.2.2 实例目的

本实例介绍 1 路 D1 预览的实现方法：通过 1 路 D1 预览功能获取某个指定 VI 输入通道的原始数据，并将数据直接环回输出到某个指定 VO 输出通道，VO 输出通道通过 DA 转换输出到显示终端显示为图像。

2.3 设计

介绍 1 路 D1 预览实例的运作机制、涉及的资源和编码过程，包括：

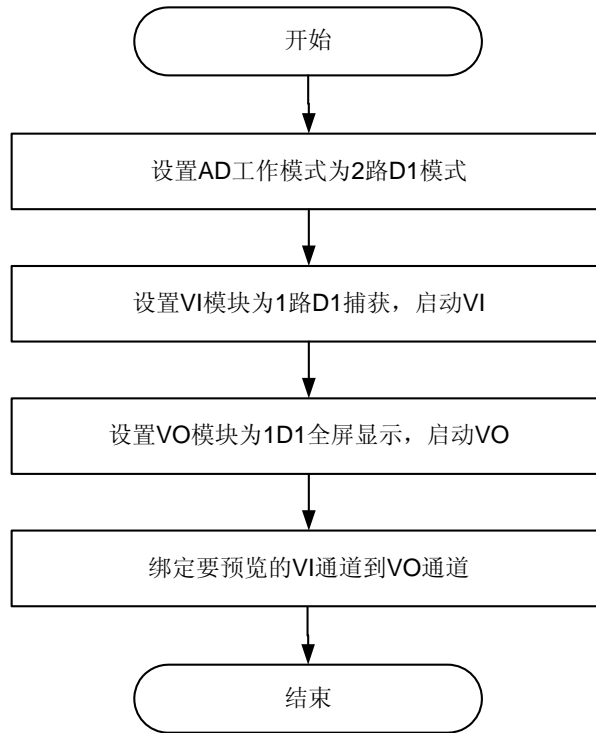
- 工作流程：简介实例的运作机制。
- 资源需求：列出实现实例所需要的资源。
- 编码准备：介绍编码前所做的准备工作，如任务分解、注意事项。

2.3.1 工作流程

实现 1 路 D1 预览：在主程序开始阶段对视频 AD、VI、VO 设备进行相应配置，启动 VI、VO 设备；然后绑定指定的 VI 通道到 VO 通道，在显示终端观看 VI 采集的现场图像。整个工作流程如图 2-1 所示。



图2-1 1路 D1 预览的工作流程



2.3.2 资源需求

表 2-2 列出实现实例所需要的资源及各资源在本实例中的作用。

表2-2 1路 D1 预览需要的资源以及头文件

资源	描述	
MPI 函数	HI_MPI_VI_SetPubAttr	设置 VI 设备公共属性
	HI_MPI_VI_Enable	启动 VI 设备
	HI_MPI_VI_Disable	关闭 VI 设备
	HI_MPI_VI_SetChnAttr	设置指定的 VI 通道属性
	HI_MPI_VI_EnableChn	启动指定的 VI 通道
	HI_MPI_VI_DisableChn	关闭指定的 VI 通道
	HI_MPI_VI_BindOutput	绑定指定的 VI 通道到指定的 VO 通道
	HI_MPI_VI_UnBindOutput	解绑定指定的 VI 通道到指定的 VO 通道
	HI_MPI_VO_SetPubAttr	设置 VO 设备公共属性
	HI_MPI_VO_Enable	启动 VO 设备



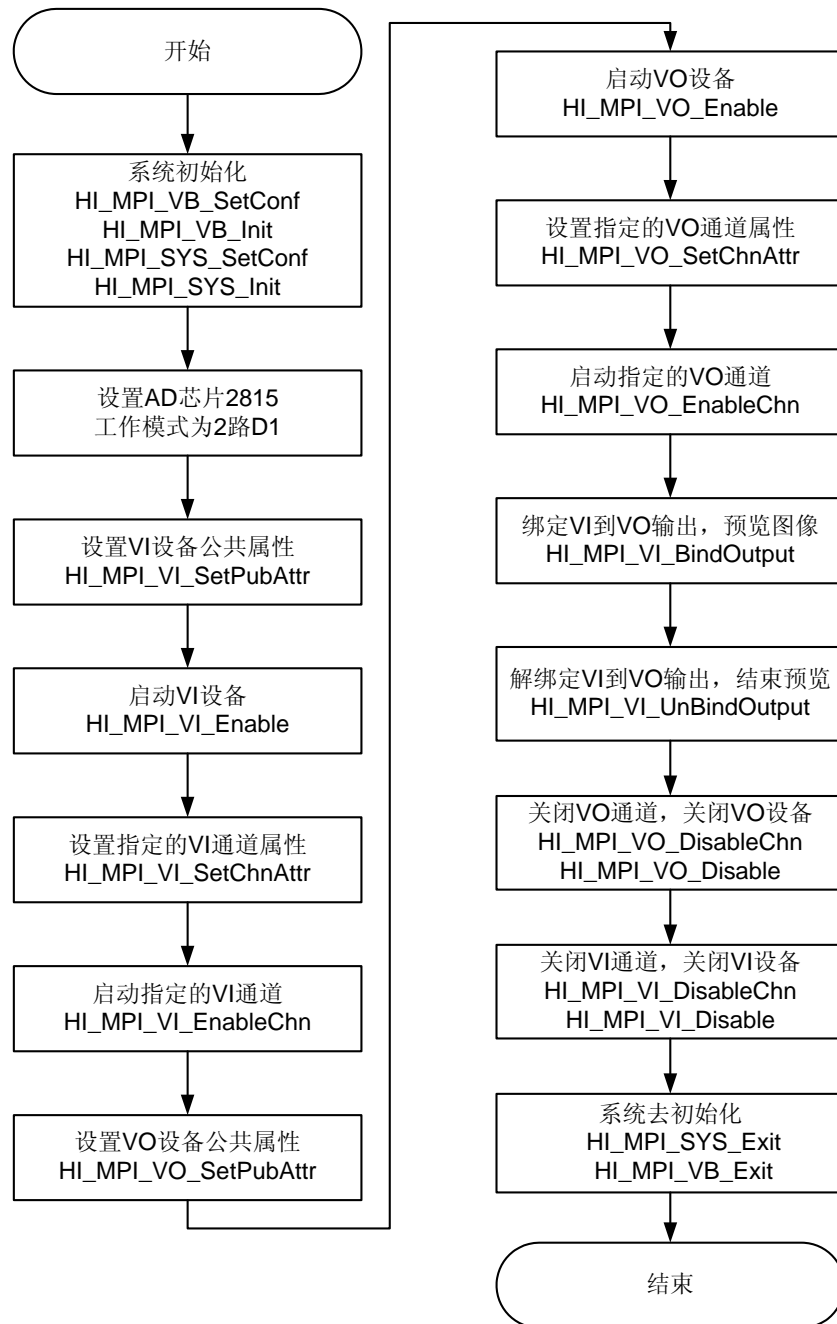
资源	描述	
	HI_MPI_VO_Disable	关闭 VO 设备
	HI_MPI_VO_SetChnAttr	设置指定的 VO 通道属性
	HI_MPI_VO_EnableChn	启动指定的 VO 通道
	HI_MPI_VO_DisableChn	关闭指定的 VO 通道
头文件	hi_common.h	MPP 平台共用的数据类型定义
	hi_comm_video.h	MPP 平台通用视频缓冲池的数据类型定义
	hi_comm_sys.h	Hi3507 芯片特殊配置数据类型定义
	hi_comm_vo.h	VO 模块共用的数据类型定义
	hi_comm_vi.h	VI 模块共用的数据类型定义
	mpi_vb.h	MPP 平台通用视频缓冲池的 MPI 接口定义
	mpi_sys.h	Hi3507 芯片的 MPI 接口定义
	mpi_vi.h	VI 模块的 MPI 接口定义
	mpi_vo.h	VO 模块的 MPI 接口定义
	tw2815.h	AD 芯片 2815 驱动接口定义
	vs6724.h	芯片 6724 驱动接口定义

2.3.3 编码准备

1 路 D1 预览业务的任务分解如图 2-2 所示。



图2-2 1路D1预览业务的任务分解



2.4 编码

实例源码存放在 SDK 的根目录下\sample\vio\sample_vio.c 中。



2.5 编译和运行

本节介绍 1 路 D1 预览实例代码的编译、运行和调试过程，包括：

- 代码编译：介绍代码编译前的编译环境设置及编译过程。
- 运行和调试：介绍程序运行和调试过程。

2.5.1 代码编译

代码编译过程如下：

1. 设置编译环境。

不同的发布包可能使用不同的交叉编译器。请查看 SDK 包根目录下的 sample/Makefile.param 中 CC 编译器宏定义来确定编译器类型，编译器有可能为 arm-hismall-linux-gcc 和 arm-uclibc-linux-gcc 等。

2. 编辑代码。

如果需要调整实例内容，可以在参考 SDK 包根目录下的 sample/vio/sample_vio.c 的基础上，通过任何编辑工具完成源代码的编辑工作。本文假设最终的代码仍然保存在 SDK 包根目录下的 sample/vio/sample_vio.c 中。

3. 编译。

要完成正确的编译，需要正确配置所需库以及相应的头文件的路径。本文假设 SDK 包所在路径为 SDK_DIR = /hiwork/hi3511sdk，MPP 系统所在路径为 MPP_PATH = ../..，MPP 库所在路径为 LIB_PATH = \$(MPP_PATH)/lib，外围驱动库及头文件根目录所在路径为 PUB_PATH = \$(SDK_DIR)/osdrv/pub，CBB_PATH = \$(SDK_DIR)/cbb。

----结束

3 中需要使用的头文件目录为：

```
INC_PATH = -I$(PUB_PATH)/include \  
           -I$(MPP_PATH)/code/include \  
           -I$(MPP_PATH)/code/mpi_inc \  
           -I$(CBB_PATH)/cbb/loadbmp
```

链接时需要用到的库的路径为：

```
LIB_PATH = $(MPP_PATH)/lib
```

链接时需要用到的库为：

```
LIB = -Wl,--start-group -lpthread -lm -lmpi -lloadbmp -Wl,--end-group
```

libpthread 是提供 POSIX 兼容信号量和线程支持的函数库，libmpi 库是媒体处理平台提供的函数库，libloadbmp 是外围驱动库。

在库及头文件路径设置完成后，可以用下面的命令来进行编译：

```
$(CC) $(CFLAGS) -o $$@ $$^ $(INCLUDE) $(LIBPATH) $(LIB)
```

命令选项说明：



- CC= arm-hismall-linux-gcc: 指定所使用的编译器。
- CFLAGS=-DHI_DEBUG -g: 定义宏。
- HI_DEBUG -o: 编译后生成可执行文件。
- \$@: 目标集。
- \$^: 所有的依赖目标集, 即.c 文件。
- 编译完成后, 生成可执行文件 sample_vio。

2.5.2 运行和调试

运行和调试过程如下:

1. 设置运行环境。

在系统运行前, 请先加载 Linux 内核, 详细内容请参见《Hi3507 Linux 开发环境 用户指南》和《Hi3507 U-Boot 移植应用》等相关文档。

2. 加载各个内核 ko 模块。

在运行程序之前, 需要正确加载 hi3511_base.ko、hi3511_sys.ko、hi3511_viu.ko、hi3511_you.ko 以及其他的驱动模块。如 gpioi2c.ko、hidmac.ko、hifb.ko、tvp_5150.ko、vs_6724.ko、adv_7179.ko、tw_2815.ko 和 tlv320.ko。可在 mpp\mkp\code 中使用 load3511 脚本进行全部 ko 的加载操作。

3. 运行程序。

在 SDK 包根目录下\sample\vio 中, 对 sample_vio.c 进行编译, 在当前目录下得到可执行文件, 执行 “./ sample_vio”, 即可进行视频预览。

4. 调试验证。

可通过调试串口监视运行情况, 如果存在异常, 根据调试串口输出信息分析具体原因, 通过系统返回的错误码分析错误产生的原因。所有的错误码定义都存放在 hi_comm_<模块名>.h 中。

----结束



3 模块开发指引

3.1 VI 模块

3.1.1 概述

Hi3507 芯片的视频输入（VI）模块实现的功能：将芯片外的视频数据，通过 ITU-R BT656/601 接口或者 digital camera 接口，存入到指定的地址区域。Hi3507 芯片只有 1 个 VI 设备接口，支持 1 路视频输入，可连接 ITU-R BT.656/601 或 digital camera 接口。

视频输入模块提供如下功能接口：

- HI_MPI_VI_SetPubAttr: 设置 VI 设备公共属性。
- HI_MPI_VI_GetPubAttr: 获取 VI 设备公共属性。
- HI_MPI_VI_Enable: 启用 VI 设备。
- HI_MPI_VI_Disable: 禁用 VI 设备。
- HI_MPI_VI_SetChnAttr: 设置 VI 通道的属性。
- HI_MPI_VI_GetChnAttr: 获取 VI 通道的属性。
- HI_MPI_VI_EnableChn: 启用 VI 通道。
- HI_MPI_VI_DisableChn: 禁用 VI 通道。
- HI_MPI_VI_GetChnLuma: 获取当前图像的亮度。
- HI_MPI_VI_GetFrame: 获取原始图像数据。
- HI_MPI_VI_ReleaseFrame: 释放原始图像数据。
- HI_MPI_VI_BindOutput: VI 绑定输出。
- HI_MPI_VI_UnbindOutput: VI 解绑定输出。
- HI_MPI_VI_SetFrameRate: 设置 VI 捕获帧率。
- HI_MPI_VI_GetFrameRate: 获取 VI 捕获帧率。
- HI_MPI_VI_SetUserPic: 设置用户图片。
- HI_MPI_VI_EnableUserPic: 启用插入用户图片。
- HI_MPI_VI_DisableUserPic: 禁用插入用户图片。



3.1.2 重要概念

需要明确以下重要概念：

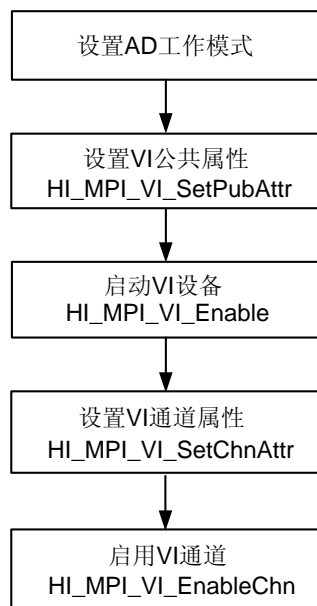
- 视频输入设备
Hi3507 芯片有 1 个 VI 设备接口，即 VI 设备 0，可支持 1 路视频输入。
- 通道、输入接口模式
当输入接口模式为 BT.601 和 digital camera 接口模式时，每个设备只支持 1 路通道捕获。
- 外接 AD 芯片
当外接 TW2815 AD 芯片时，由于软件无法检测外接 2815 芯片处在何种工作模式，因此必须设置对应接口的工作模式。对于 VI 设备 0，可以将 AD 芯片设置在 1 路 D1 工作模式下。
当外接 digital camera 时，只需设置捕获大小和起始位置即可。

3.1.3 配置并启动 VI 捕获

3.1.3.1 业务流程

配置并启动 VI 捕获的业务流程如图 3-1 所示。

图3-1 配置并启动 VI 捕获的业务流程



3.1.3.2 编程指导

配置并启动 VI 捕获的步骤如下：

1. 设置 AD 工作模式。
2. 配置 VI 公共属性（即设备属性）。



由于不默认值，用户在对设备做任何操作前都需要先配置好该设备，注意工作模式要和 AD 的工作模式一致。

3. 启用 VI 设备。
4. 配置 VI 通道属性。

由于不默认值，用户在对通道做任何操作前都需要先配置好该通道。常用的 VI 通道属性配置如表 3-1 所示。

5. 启用 VI 通道。

----结束

表3-1 常用的 VI 通道属性配置（BT656 PAL 制）

模式	D1	half D1	CIF
x	0	0	0
y	0	0	0
width	704	704	704
height	288	288	288
enCapSel	BOTH	BOTTOM	BOTTOM
bDownScale	FALSE	FALSE	TRUE
bChromaResample	FALSE	FALSE	FALSE
bHighPri	FALSE	FALSE	FALSE
enViPixFormat	420	420	420



说明

- 可动态设置通道属性。
- 只能静态设置公共属性，即如果要改变公共属性必须先禁用对应的 VI 设备。

3.1.3.3 实例

```

VI_PUB_ATTR_S ViAttr;
VI_CHN_ATTR_S ViChnAttr;

memset(&ViAttr,0,sizeof(VI_PUB_ATTR_S));
memset(&ViChnAttr,0,sizeof(VI_CHN_ATTR_S));

ViAttr.enInputMode = VI_MODE_BT656;
ViAttr.enWorkMode = VI_WORK_MODE_1D1;
ViAttr.u32AdType = AD_2815;
ViAttr.enViNorm = VIDEO_ENCODING_MODE_PAL;

```



```
ViChnAttr.enCapSel = VI_CAPSEL_TOM;
ViChnAttr.stCapRect.u32Height = 288;
ViChnAttr.stCapRect.u32Width = 704;
ViChnAttr.stCapRect.s32X = 0;
ViChnAttr.stCapRect.s32Y = 0;
ViChnAttr.bDownScale = HI_TRUE;
ViChnAttr.enViPixFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_420;
for(i=0;i<1;i++)
{
    if (HI_SUCCESS!=HI_MPI_VI_SetPubAttr(i,&ViAttr))
    {
        return HI_FAILURE;
    }
    if (HI_SUCCESS!=HI_MPI_VI_Enable(i))
    {
        return HI_FAILURE;
    }
    for(j=0;j<1;j++)
    {
        if (HI_SUCCESS!=HI_MPI_VI_SetChnAttr(i,j,&ViChnAttr))
        {
            return HI_FAILURE;
        }
        if (HI_SUCCESS!=HI_MPI_VI_EnableChn(i,j))
        {
            return HI_FAILURE;
        }
    }
}
```

3.2 VO 模块

3.2.1 概述

Hi3507 最多支持 32 路通道视频图像显示，并支持画中画显示。显示的输入图像数据为 YUV 4:2:0 或 YUV 4:2:2 格式，每路通道有不同的优先级，如果画面有叠加，则高优先级通道的画面会覆盖低优先级通道的画面；如果多个通道配置相同的优先级，则通道号数值越大优先级越高。

视频输出模块提供如下功能接口：

- HI_MPI_VO_SetPubAttr：设置 VO 公共属性。
- HI_MPI_VO_GetPubAttr：获取 VO 公共属性。
- HI_MPI_VO_Enable：启用 VO 设备。



- HI_MPI_VO_Disable: 禁用 VO 设备。
- HI_MPI_VO_SetChnAttr: 设置 VO 通道的属性。
- HI_MPI_VO_GetChnAttr: 获取 VO 通道的属性。
- HI_MPI_VO_EnableChn: 启用 VO 通道。
- HI_MPI_VO_DisableChn: 禁用 VO 通道。
- HI_MPI_VO_SendFrame: 将视频图像送入输出通道显示。
- HI_MPI_VO_SetChnField: 配置视频输出的显示帧场策略。
- HI_MPI_VO_GetChnField: 获取视频输出的显示帧场策略。
- HI_MPI_VO_SetFrameRate: 设置视频输出的显示帧率。
- HI_MPI_VO_GetFrameRate: 获取视频输出的显示帧率。
- HI_MPI_VO_EnableDeflicker: 使能视频输出隔行抗闪功能。
- HI_MPI_VO_DisableDeflicker: 禁用视频输出隔行抗闪功能。
- HI_MPI_VO_ChnPause: 暂停播放指定通道。
- HI_MPI_VO_ChnResume: 恢复播放指定通道。
- HI_MPI_VO_ChnStep: 单帧播放指定通道。
- HI_MPI_VO_CreateSyncGroup: 创建多通道同步组。
- HI_MPI_VO_DestroySyncGroup: 销毁多通道同步组。
- HI_MPI_VO_RegisterSyncGroup: 注册 VO 通道到多通道同步组。
- HI_MPI_VO_UnRegisterSyncGroup: 注销 VO 通道从多通道同步组。
- HI_MPI_VO_SyncGroupStart: 启动多通道同步组。
- HI_MPI_VO_SyncGroupStop: 停止多通道同步组。
- HI_MPI_VO_SetSyncGroupFrameRate: 设置多通道同步组帧率。
- HI_MPI_VO_GetSyncGroupFrameRate: 获取多通道同步组帧率。
- HI_MPI_VO_PauseSyncGroup: 暂停多通道同步组播放。
- HI_MPI_VO_ResumeSyncGroup: 恢复多通道同步组播放。
- HI_MPI_VO_StepSyncGroup: 单帧播放多通道同步组。
- HI_MPI_VO_SetSyncGroupBase: 设置多通道同步组的基准时间戳。
- HI_MPI_VO_GetSyncGroupBase: 获取多通道同步组的基准时间戳。
- HI_MPI_VO_SetZoomInWindow: 设置视频输出局部放大窗口。
- HI_MPI_VO_GetZoomInWindow: 获取视频输出局部放大窗口参数。
- HI_MPI_VO_GetChnPts: 获取视频输出通道当前播放图像的时间戳。
- HI_MPI_VO_SetAttrBegin: 设置属性开始。
- HI_MPI_VO_SetAttrEnd: 设置属性结束。
- HI_MPI_VO_ChnShow: 设置显示通道。
- HI_MPI_VO_ChnHide: 设置隐藏通道。
- HI_MPI_VO_Query: 查询视频输出通道状态。
- HI_MPI_VO_SetDisplayFrameRate: 设置屏幕显示低帧率。
- HI_MPI_VO_GetDisplayFrameRate: 获取屏幕显示帧率。



- HI_MPI_VO_GetChnFrame: 获取输出通道图像数据。
- HI_MPI_VO_ReleaseChnFrame: 释放输出通道图像数据。
- HI_MPI_VO_GetScreenFrame: 获取输出屏幕图像数据。
- HI_MPI_VO_ReleaseScreenFrame: 释放输出屏幕图像数据。

3.2.2 重要概念

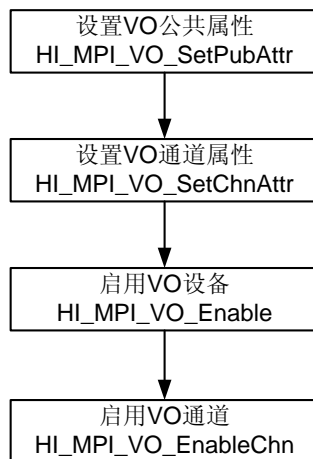
无。

3.2.3 配置并启动 VO 输出

3.2.3.1 业务流程

配置并启动 VO 输出的业务流程如图 3-2 所示。

图3-2 配置并启动 VO 输出的业务流程



3.2.3.2 编程指导

配置并启动 VO 输出的步骤如下：

1. 设置 VO 公共属性。

由于不设默认值，在对输出设备做任何操作前都需要先设置好设备。

2. 设置 VO 通道属性。

由于不设默认值，在对通道做任何操作前都需要先配置好通道。通过配置每个通道显示区域的起始位置和宽高可动态设置通道的显示区域，区域的起始相对坐标为屏幕左上角的 (0,0)。

3. 启用 VO 设备。
4. 启用 VO 通道。

----结束



说明

- 可动态设置通道属性。
- 只能静态设置公共属性，如果要改变公共属性必须先禁用对应的 VO 设备。
- 关于帧率控制和显示控制的接口请参见“4 视频综合业务开发指引”。

3.2.3.3 实例

```
VO_PUB_ATTR_S VoAttr;
VO_CHN_ATTR_S VoChnAttr[6];

VoAttr.stTvConfig.stComposeMode = VIDEO_ENCODING_MODE_PAL;
VoAttr.u32BgColor = DEF_COLOR;

VoChnAttr[0].bZoomEnable = HI_TRUE;
VoChnAttr[0].stRect.s32X = 0;
VoChnAttr[0].stRect.s32Y = 0;
VoChnAttr[0].stRect.u32Height = 192;
VoChnAttr[0].stRect.u32Width = 240;

VoChnAttr[1].bZoomEnable = HI_TRUE;
VoChnAttr[1].stRect.s32X = 240;
VoChnAttr[1].stRect.s32Y = 0;
VoChnAttr[1].stRect.u32Height = 192;
VoChnAttr[1].stRect.u32Width = 240;

VoChnAttr[2].bZoomEnable = HI_TRUE;
VoChnAttr[2].stRect.s32X = 480;
VoChnAttr[2].stRect.s32Y = 0;
VoChnAttr[2].stRect.u32Height = 192;
VoChnAttr[2].stRect.u32Width = 240;

VoChnAttr[3].bZoomEnable = HI_TRUE;
VoChnAttr[3].stRect.s32X = 0;
VoChnAttr[3].stRect.s32Y = 192;
VoChnAttr[3].stRect.u32Height = 192;
VoChnAttr[3].stRect.u32Width = 240;

VoChnAttr[4].bZoomEnable = HI_TRUE;
VoChnAttr[4].stRect.s32X = 0;
VoChnAttr[4].stRect.s32Y = 384;
VoChnAttr[4].stRect.u32Height = 192;
VoChnAttr[4].stRect.u32Width = 240;

VoChnAttr[5].bZoomEnable = HI_TRUE;
VoChnAttr[5].stRect.s32X = 240;
```




```
VoChnAttr[5].stRect.s32Y = 192;
VoChnAttr[5].stRect.u32Height = 384;
VoChnAttr[5].stRect.u32Width = 480;

HI_MPI_VO_SetPubAttr(&VoAttr);

for(i=0;i<6;i++)
{
    HI_MPI_VO_SetChnAttr(i, &VoChnAttr[i]);
}

HI_MPI_VO_Enable();
for(i=0;i<6;i++)
{
    HI_MPI_VO_EnableChn(i);
}
```

3.3 VENC 模块

3.3.1 概述

Hi3507 媒体处理芯片提供硬件加速的多协议编码，具体内容请参见《Hi3507 媒体处理软件 开发参考》。

VENC 模块完成各个协议编码，协调 MD、VPP 相关模块的管理、同步和控制，配合软件调度和硬件共同完成视频编码相关功能。

VENC 模块提供如下功能接口：

- HI_MPI_VENC_CreateGroup: 创建编码通道组。
- HI_MPI_VENC_DestroyGroup: 销毁编码通道组。
- HI_MPI_VENC_BindInput: 绑定 VI 到通道组。
- HI_MPI_VENC_UnbindInput: 解绑定 VI 到通道组。
- HI_MPI_VENC_CreateChn: 创建编码通道。
- HI_MPI_VENC_DestroyChn: 销毁编码通道。
- HI_MPI_VENC_RegisterChn: 注册编码通道到通道组。
- HI_MPI_VENC_UnRegisterChn: 注销编码通道到通道组。
- HI_MPI_VENC_StartRecvPic: 开启编码通道接收输入图像。
- HI_MPI_VENC_StopRecvPic: 停止编码通道接收输入图像。
- HI_MPI_VENC_Query: 查询编码通道状态。
- HI_MPI_VENC_SetChnAttr: 设置编码通道属性。
- HI_MPI_VENC_GetChnAttr: 获取编码通道属性。
- HI_MPI_VENC_SetWmAttr: 设置编码数字水印的属性。



- HI_MPI_VENC_GetWmAttr: 获取编码数字水印的属性。
- HI_MPI_VENC_EnableWm: 启用编码数字水印。
- HI_MPI_VENC_DisableWm: 禁用编码数字水印。
- HI_MPI_VENC_GetStream: 获取编码的码流。
- HI_MPI_VENC_ReleaseStream: 释放码流缓存。
- HI_MPI_VENC_RequestIDR: 请求 I 帧。
- HI_MPI_VENC_InsertUserData: 插入用户数据。
- HI_MPI_VENC_SendFrame: 支持用户发送原始图像进行编码。
- HI_MPI_VENC_GetCapability: 获取视频编码能力集。
- HI_MPI_VENC_GetFd: 获取编码通道的文件描述符。

3.3.2 重要概念

需要明确以下重要概念:

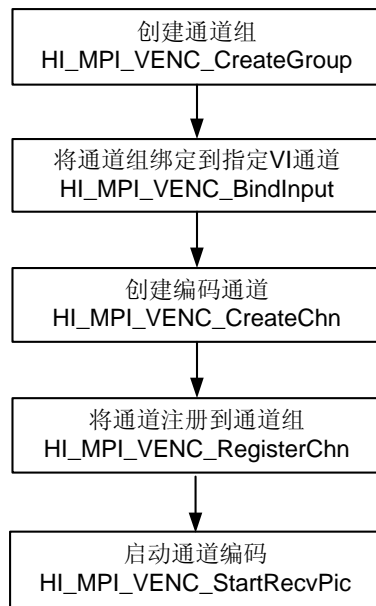
- 主次码流
主次码流是指硬件逻辑单元在启动一次之后同时产生的 2 路码流, 即 1 路主码流和 1 路次码流。主码流和次码流可以为不同的编码协议, 但其宽高比例都必须满足 1:1、1:2 或 1:4, 次码流不能单独存在 (必须和 1 路主码流在同一个通道组中)。
- 双码流
双码流是指硬件逻辑单元在启动 2 次之后分时产生的 2 个码流, 即 2 路主码流。双码流可以为不同的编码协议, 双码流之间的大小比例没有约束关系。
- 通道组
通道组是指芯片能够同时处理的编码通道的集合, 相当于一个容器。一个通道组最多可同时包含 1 路主码流 (H.264/MJPEG)、1 路次码流 (H.264/MJPEG) 和 1 路 MPEG4, 或者仅包含 1 路 JPEG 抓拍 (即 JPEG 抓拍时, 不允许包含任何其他通道), 主次码流不能同时为 MJPEG 编码。

3.3.3 配置并启动视频编码

3.3.3.1 业务流程

配置并启动视频编码的业务流程如图 3-3 所示。

图3-3 配置并启动视频编码的业务流程



3.3.3.2 编程指导

配置并启动视频编码的步骤如下：

1. 创建一个通道组。

创建一个通道组用于容纳通道。如果需要很多路编码时，可能需要创建多个通道组。

2. 将通道组绑定到指定的 VI 通道。

绑定到 VI 通道（由 VI 设备号和 VI 通道号确定）之后，该通道组所有通道都将使用同一绑定。

3. 创建一个编码通道。

创建所需要的编码通道，比如可以创建 1D1 大小的 H.264 编码通道或创建 1CIF 大小的 MJPEG 编码通道，也可以创建多路编码通道。

4. 将通道注册到通道组。

将创建的编码通道注册到指定的编码通道组，注册后编码通道组只能有 1 个主码流编码通道，如果还包含次码流编码通道，则主码流通道和次码流编码通道的宽高比例必须满足 1:1、1:2 或 1:4 的关系，否则会导致注册失败。关于 HI_MPI_VENC_RegisterChn 的注意事项，详细内容请参见《Hi3507 媒体处理软件开发参考》。

5. 启动通道编码。

启动编码是以通道为单位，可以单独启动通道组里某一个主码流或者次码流开始编码。

----结束



说明

- 可以将创建成功的任意通道号的通道注册到创建成功的任意组号的通道组中，唯一需要注意的是：注册后，通道组内的通道需要满足约束关系，否则注册会导致失败。
- 配置和启用多路视频编码：创建多个通道组，并创建多个通道，然后将每个通道都绑定到不同的通道组中进行启动。多个通道是否绑定到同一个 VI 不受约束。
- 当前版本不支持 MPEG4 编码。

3.3.3.3 实例

```
s32Ret = HI_MPI_VENC_CreateGroup(VeGroup);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_CreateGroup err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_BindInput(VeGroup, ViDev, ViChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_BindInput err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_CreateChn(VeChn, pstAttr, HI_NULL);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_CreateChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_RegisterChn(VeGroup, VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_RegisterChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_StartRecvPic(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StartRecvPic err 0x%x\n",s32Ret);
    return HI_FAILURE;
}
```



3.3.4 停止并销毁视频编码

3.3.4.1 业务流程

停止并销毁视频编码可以选择两种方式：

- 强制停止现在的工作，放弃未编完的图像和未获取的码流并销毁编码通道。
- 等待编码完成，所有码流获取完毕再退出编码。

两种方式唯一的区别在于是否查询目前的状态，然后根据目前的状态来决定是否反注册并销毁编码通道。具体工作流程如图 3-4 和图 3-5 所示。

图3-4 强行停止并销毁视频编码的业务流程

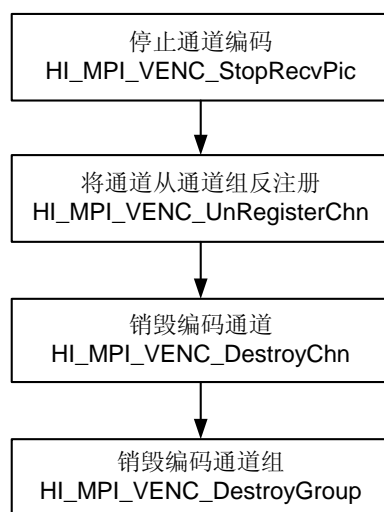
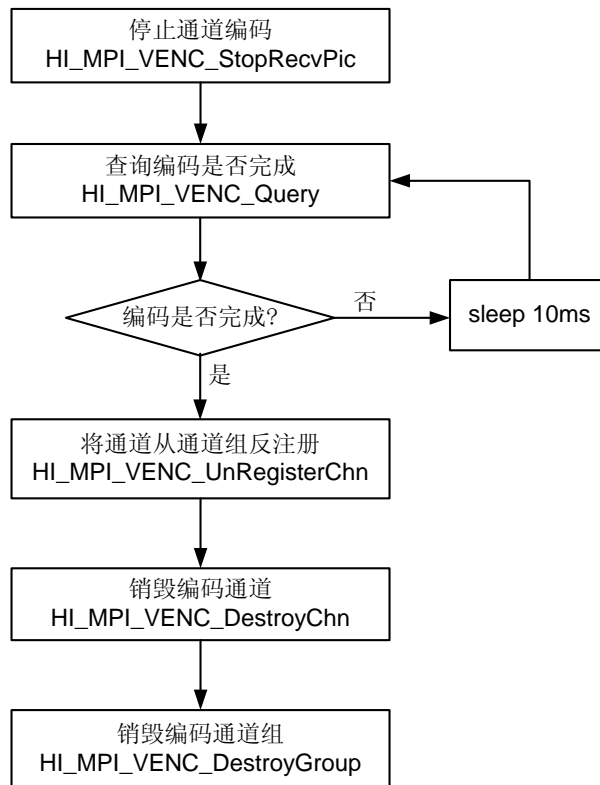




图3-5 等待编码完成销毁视频编码的业务流程



3.3.4.2 编程指导

强行停止并销毁视频编码的步骤如下：

1. 停止通道编码。

告诉编码器停止编码，不再接收新图像。

2. 将通道从通道组反注册。

反注册成功后通道上下文恢复到刚创建时的状态，所有资源都被清空，此时可以选择将此通道注册到其他通道组。

3. 销毁编码通道。

销毁编码通道所有资源。

4. 销毁编码通道组。

----结束



注意

- 任何时候通道组中的通道都必须满足一定约束：注册时，主码流要先于次码流；反注册时，次码流要先于主码流。
- 通道组为空时才能销毁成功。

等待编码完成销毁视频编码的步骤如下：

1. 停止通道编码。
2. 查询编码器状态，如果还有残留的码流或者图像，则 sleep 一段时间（例如 10ms），等待编码器运行完成。
3. 将通道从通道组反注册。
4. 销毁编码通道。
5. 销毁编码通道组。

----结束

3.3.4.3 实例

强行停止并销毁视频编码的参考代码：

```
s32Ret = HI_MPI_VENC_StopRecvPic(VeChn);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_UnRegisterChn(VeChn);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyChn(VeChn);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyGroup(VeGroup);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
```

等待编码完成并销毁视频编码的参考代码：

```
s32Ret = HI_MPI_VENC_StopRecvPic(VeChn);
```



```
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
do{
    s32Ret = HI_MPI_VENC_Query(VeChn,&stStat);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StopRecvPic err:0x%x\n",s32Ret);
        return HI_FAILURE;
    }
    usleep(1000);
}while(stStat.u32LeftPics != 0 || stStat.u32LeftStreamBytes !=0 );
s32Ret = HI_MPI_VENC_UnRegisterChn(VeChn);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyChn(VeChn);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyGroup(VeGroup);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
```

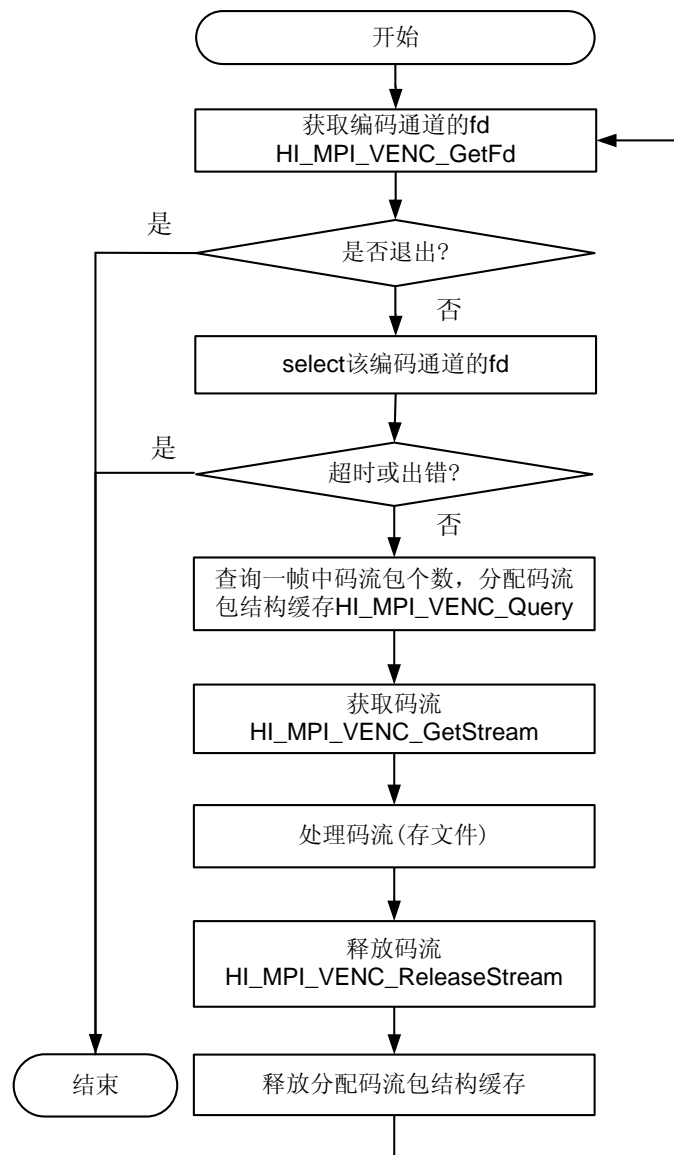
3.3.5 按帧获取编码码流

3.3.5.1 业务流程

开始编码后就可以开始获取并处理码流。在创建编码通道时指定按帧获取码流或者按包获取码流，详细内容请参见在《Hi3507 媒体处理软件 开发参考》中 HI_MPI_VENC_CreateChn 的具体用法。

按帧获取 1 路编码后码流的业务流程如图 3-6 所示。

图3-6 按帧获取 1 路编码后码流的业务流程



3.3.5.2 编程指导

按帧获取 1 路编码码流的步骤如下：

1. 获取编码通道的 fd。

获取编码通道 fd 是为了 select 该编码通道。按帧获取码流的特点：事先并不知道一帧包含多少个码流包，必须借助 select 才能在一个恰当的时间去获取该信息，即在 select 成功的时间去查询当前码流包个数信息。

2. Select 该编码通道的 fd。

如果出错或者超时则退出，否则就表示已经有一个完整的帧在缓冲区，可以查询当前的帧信息，并获取该帧。



3. 查询当前帧的码流包个数，并分配码流包结构缓存。

根据查询到当前帧中的码流包个数来分配结构体缓存大小。可以通过分配一个非常大的缓存来容纳任何情况下的一帧所有码流包。

4. 获取码流。
5. 处理码流。

此时不要修改码流结构体里面的任何内容，如果修改，可能会导致释放码流失败。

6. 释放码流。

一定要按照先取出先释放的顺序释放，尽早释放，防止码流阻塞。

7. 释放在 3 里分配的码流包结构体缓存，然后开始下一个循环。

----结束



说明

码流包包含 2 个指针 pu8Addr[0]和 pu8Addr[1]，如果 u32Len[1]为 0，pu8Addr[1]无效，如果 u32Len[0]为 0，pu8Addr[0]无效。

3.3.5.3 实例

按帧获取编码后码流参考代码：

```
s32VencFd = HI_MPI_VENC_GetFd(VeChn);
do{
    FD_ZERO(&read_fds);
    FD_SET(s32VencFd,&read_fds);
    s32Ret = select(s32VencFd+1, &read_fds, NULL, NULL, NULL);
    if (s32Ret < 0)
    {
        return NULL;
    }
    else if (0 == s32Ret)
    {
        printf("time out\n");
        return NULL;
    }
    else
    {
        if (FD_ISSET(s32VencFd, &read_fds))
        {
            s32Ret = HI_MPI_VENC_Query(VeChn, &stStat);
            if (s32Ret != HI_SUCCESS)
            {
                return NULL;
            }
        }
    }
}
```



```
        stStream.pstPack =
(VENC_PACK_S*)malloc(sizeof(VENC_PACK_S)*stStat.u32CurPacks);
        if (NULL == stStream.pstPack)
        {
            printf("malloc memory err!\n");
            return NULL;
        }
        stStream.u32PackCount = stStat.u32CurPacks;

        s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, HI_IO_BLOCK);
        if (HI_SUCCESS != s32Ret)
        {
            free(stStream.pstPack);
            stStream.pstPack = NULL;
            return NULL;
        }
        /*处理获取的码流*/
        .....
        s32Ret = HI_MPI_VENC_ReleaseStream(VeChn,&stStream);
        if (s32Ret)
        {
            free(stStream.pstPack);
            stStream.pstPack = NULL;
            return NULL;
        }

        free(stStream.pstPack);
        stStream.pstPack = NULL;
    }
    }
    u32FrameIdx ++;
}while (u32FrameIdx < 0xff);
```

如何使用码流（存文件）的参考代码：

```
for (i=0; i< stStream.u32PackCount; i++)
{
    fwrite(stStream.pstPack[i].pu8Addr[0],
           stStream.pstPack[i].u32Len[0], 1, pFile);
    if (stStream.pstPack[i].u32Len[1] > 0)
    {
        fwrite(stStream.pstPack[i].pu8Addr[1],
               stStream.pstPack[i].u32Len[1], 1, pFile);
    }
}
```



}

3.3.6 按包获取编码码流

3.3.6.1 业务流程

按包获取码流的特点：事先已经知道每次获取一个码流包，可不进行查询码流包个数的操作，此时可以灵活的选择阻塞获取、非阻塞获取（选择阻塞或非阻塞方式由 HI_MPI_VENC_GetStream 接口的阻塞标识决定，详细内容请参见《Hi3507 媒体处理软件 开发参考》）或使用 select:

- 阻塞获取：每次都会等待码流包，并能成功的获取到一个码流包（在获取过程中通道被强制销毁除外）。其工作流程如图 3-7 所示。
- 非阻塞获取：每次都会立刻返回，如果有码流包则获取码流包；如果没有码流包则返回没有码流包的错误码。其工作流程如图 3-8 所示。
- Select：保证在 select 成功时有码流包在缓存中，所以此时使用 HI_MPI_VENC_GetStream 的阻塞获取或者非阻塞获取方式都会返回成功。其工作流程如图 3-9 所示。

图3-7 按包非阻塞获取码流工作流程

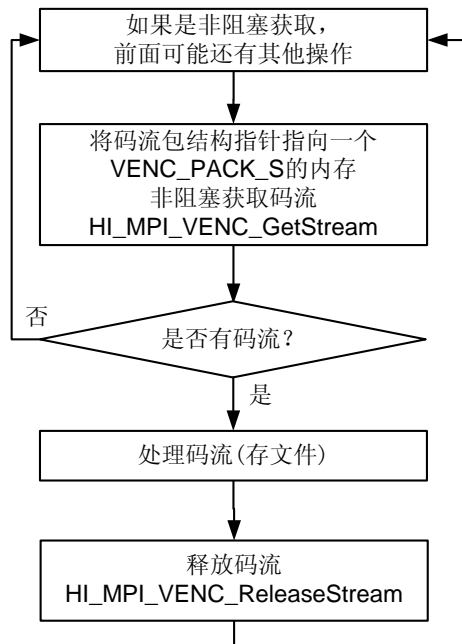




图3-8 按包阻塞获取码流工作流程

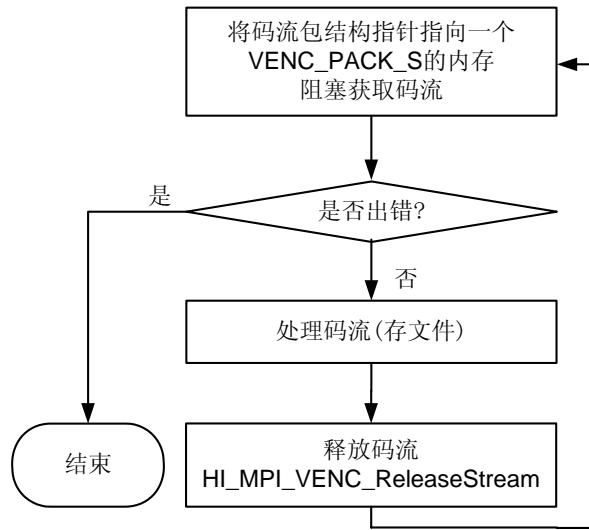
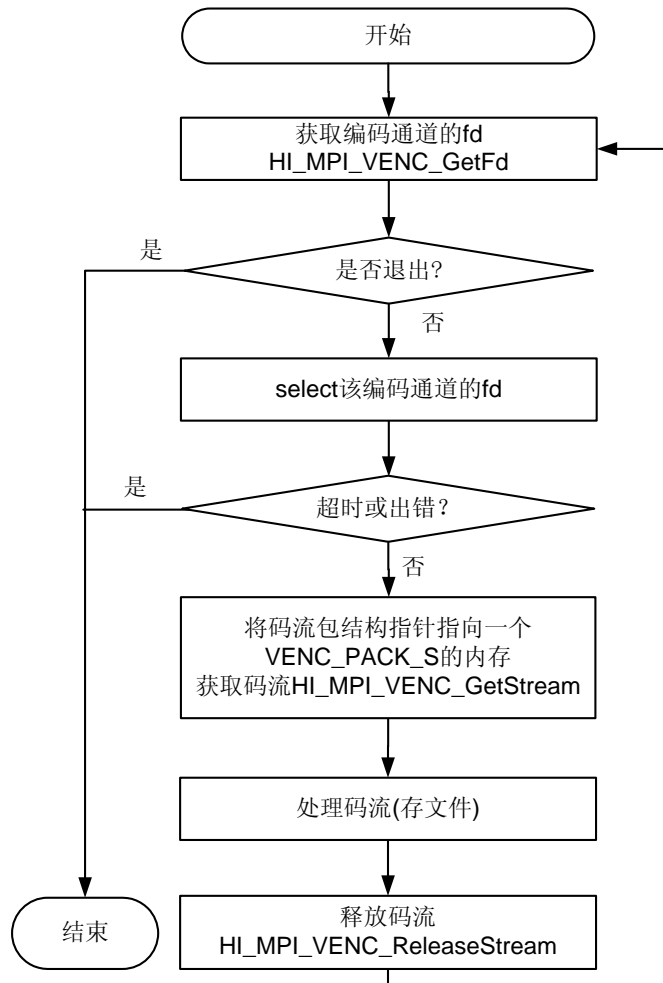


图3-9 按 select 获取码流工作流程



3.3.6.2 编程指导

按包非阻塞获取码流的步骤如下：

1. 将码流包结构指针指向一个 VENC_PACK_S 的内存，非阻塞获取码流。
2. 如果成功则处理码流，返回没有码流则做其它处理，然后再重复 1。
3. 释放码流，开始下一个循环。

----结束

按包阻塞获取码流的步骤如下：

1. 将码流包结构指针指向一个 VENC_PACK_S 的内存，阻塞获取码流。
2. 如果成功则处理码流，返回错误则表示通道销毁或者出现其他错误，退出。
3. 释放码流，开始下一个循环。



----结束

按包 select 获取码流的步骤如下:

1. 获取编码通道的 fd。
2. Select 该 fd, 如果出错或者超时则退出, 否则表示有码流进入缓冲。
3. 将码流包结构指针指向一个 VENC_PACK_S 的内存, 获取码流。
4. 如果成功则处理码流, 错误则退出。
5. 释放码流, 开始下一个循环。

----结束

3.3.6.3 实例

按包非阻塞获取码流参考代码:

```
stStream.pstPack = &stPack ;
stStream.u32PackCount = 1;

s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, HI_IO_NOBLOCK);
if (HI_SUCCESS == s32Ret)
{
    s32Ret = HI_MPI_VENC_ReleaseStream(VeChn, &stStream);
    if (HI_SUCCESS != s32Ret)
    {
        return HI_FAILURE;
    }
}
```

按包阻塞获取码流参考代码:

```
stStream.pstPack = &stPack ;
stStream.u32PackCount = 1;

s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, HI_IO_BLOCK);
if (HI_SUCCESS == s32Ret)
{
    s32Ret = HI_MPI_VENC_ReleaseStream(VeChn, &stStream);
    if (HI_SUCCESS != s32Ret)
    {
        return HI_FAILURE;
    }
}
```

按包 select 获取码流参考代码:



```
s32VencFd = HI_MPI_VENC_GetFd(VeChn);
do{
    FD_ZERO(&read_fds);
    FD_SET(s32VencFd,&read_fds);
    s32Ret = select(s32VencFd+1, &read_fds, NULL, NULL, NULL);
    if (s32Ret < 0)
    {
        return NULL;
    }
    else if (0 == s32Ret)
    {
        printf("time out\n");
        return NULL;
    }
    else
    {
        if (FD_ISSET(s32VencFd, &read_fds))
        {
            stStream.pstPack = &stPack ;
            stStream.u32PackCount = stStat.u32CurPacks;

            s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, HI_IO_BLOCK);
            if (HI_SUCCESS != s32Ret)
            {
                return NULL;
            }
            /*处理获取的码流*/
            .....
            s32Ret = HI_MPI_VENC_ReleaseStream(VeChn,&stStream);
            if (s32Ret)
            {
                return NULL;
            }
        }
    }
    u32FrameIdx ++;
}while (u32FrameIdx < 0xff);
```

3.3.7 配置并启动主次码流编码

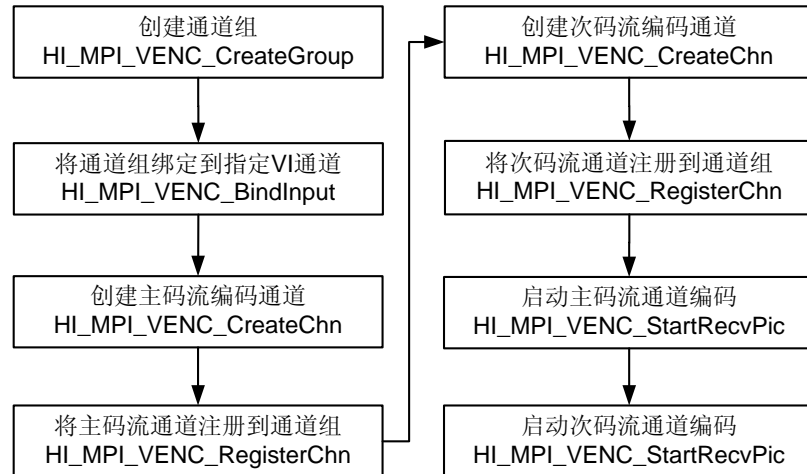
3.3.7.1 业务流程

配置并启动主次码流编码和启动一个码流编码（请参见“[3.3.3 配置并启动视频编码](#)”）的区别在于通道组中通道的数量，如果只有一个主码流则就只启动一个主码流编



码；如果包含一个主码流、一个次码流则可以启动主次码流同时编码。配置并启动主次码流编码的业务流程如图 3-10 所示。

图3-10 配置并启动主次码流编码的业务流程



3.3.7.2 编程指导

配置并启动主次码流编码的步骤如下：

1. 创建通道组。
2. 将通道组绑定到指定的 VI 通道。
3. 创建主码流编码通道。
4. 将主码流通道注册到通道组。
5. 创建次码流编码通道。
6. 将次码流通道注册到通道组。
7. 启动主码流通道编码。
8. 启动次码流通道编码。

----结束



说明

可以任意创建主次码流通道的通道号，但在注册时必须先注册主码流再注册次码流。

3.3.7.3 实例

```

s32Ret = HI_MPI_VENC_CreateGroup (VeGroup);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
  
```



```
s32Ret = HI_MPI_VENC_BindInput(VeGroup, ViDev, ViChn);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
/*创建主码流*/
s32Ret = HI_MPI_VENC_CreateChn(VeChnMain, pstMainAttr, HI_NULL);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
/*注册主码流*/
s32Ret = HI_MPI_VENC_RegisterChn(VeGroup, VeChnMain);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
/*创建次码流*/
s32Ret = HI_MPI_VENC_CreateChn(VeChnMin, pstMinorAttr, HI_NULL);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
/*注册次码流*/
s32Ret = HI_MPI_VENC_RegisterChn(VeGroup, VeChnMin);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_StartRecvPic(VeChnMain);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_StartRecvPic(VeChnMin);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
```

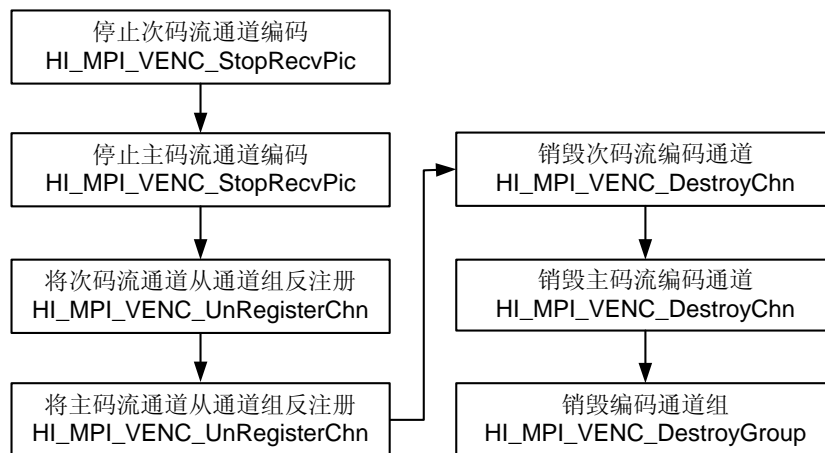


3.3.8 停止并销毁主次码流编码

3.3.8.1 业务流程

停止并销毁主次码流编码和销毁一个码流编码（请参见“3.3.4 停止并销毁视频编码”）的区别在于必须先反注册次码流，才能反注册主码流。强行停止并销毁主次码流编码的业务流程如图 3-11 所示。

图3-11 强行停止并销毁主次码流编码的业务流程



3.3.8.2 编程指导

强行停止并销毁主次码流编码的步骤如下：

1. 停止次码流通道编码。
2. 停止主码流通道编码。
3. 将次码流通道从通道组反注册。
4. 将主码流通道从通道组反注册。
5. 销毁次码流编码通道。
6. 销毁主码流编码通道。
7. 销毁编码通道组。

----结束

说明

- 反注册时，次码流必须先于主码流反注册。
- 销毁或者停止编码时，主码流或者次码流销毁或停止编码的顺序不做要求。

3.3.8.3 实例

```
s32Ret = HI_MPI_VENC_StopRecvPic(VeChnMin);
```



```
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_StopRecvPic(VeChnMain);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_UnRegisterChn(VeChnMin);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_UnRegisterChn(VeChnMain);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyChn(VeChnMin);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyChn(VeChnMain);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyGroup(VeGroup);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
```

3.3.9 获取多路编码码流

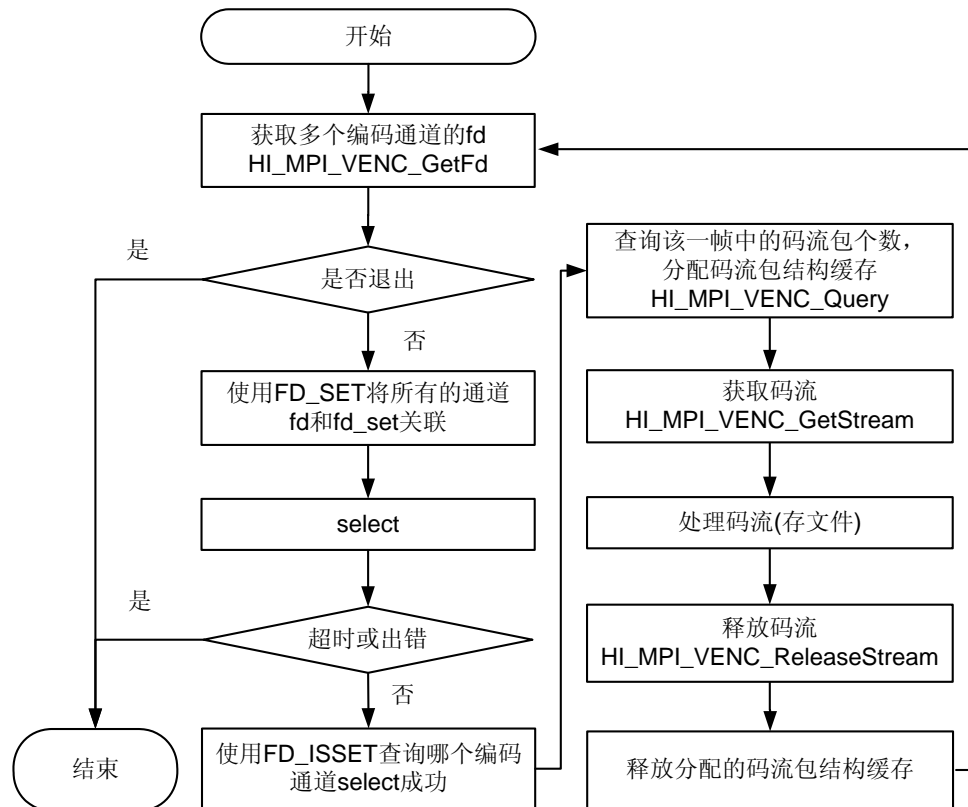
3.3.9.1 业务流程

获取多路编码码流的方法有两种：

- 为每一个编码通道启用一个单独的线程，在任何一个线程中都可以使用按帧获取码流（请参见“[3.3.5 按帧获取编码码流](#)”）或按包获取（请参见“[3.3.6 按包获取编码码流](#)”）。

- 使用单线程 select 同时 select 多路码流，这样可以在多路情况下节省线程调度的开销，推荐使用此方法。select 多路码流的工作流程如图 3-12 所示。

图3-12 select 多路码流工作流程



3.3.9.2 编程指导

Select 多路码流的步骤如下：



注意

由于 select 会改写 fd_set 和 timeval 的内容，所以每次循环重新 select 前都必须重新关联 fd_set（最好每次循环都调用 FD_CLR，再使用 FD_SET 重新关联所有 fd。）和重新设置 timeval。

1. 获取多个编码通道的 fd。
2. 使用 FD_SET 将所有通道 fd 和 fd_set 关联。
3. Select。
4. 使用 FD_ISSET 查询哪个编码通道 select 成功。



5. 对 select 成功的通道查询码流包个数，并分配结构体缓存。
6. 获取码流。
7. 处理码流。
8. 释放码流。
9. 释放在 5 分配的码流包结构体缓存，然后开始下一个循环。

----结束

3.3.9.3 实例

```
for (i=0;i<8;i++)
{
    VencFd[i] = HI_MPI_VENC_GetFd(i);
    if (VencFd[i] <= 0)
    {
        return HI_FAILURE;
    }
    if(maxfd <= VencFd[i]) maxfd = VencFd[i] ;
}
do
{
    /*每次都要重新设置read_fds,否则可能出错,因为select会改写read_fds的内容*/
    FD_ZERO(&read_fds);
    for (i=0; i<8; i++)
    {
        FD_SET(VencFd[i],&read_fds);
    }
    s32ret = select(maxfd + 1,&read_fds,NULL,NULL,NULL);
    if(s32ret < 0)
    {
        return HI_FAILURE;
    }
    else if(s32ret == 0)
    {
        return HI_FAILURE;
    }
    else
    {
        for (i=0; i<8; i++)
        {
            if(FD_ISSET(VencFd[i],&read_fds))
            {
                /*按帧获取1路码流并处理*/
            }
        }
    }
}
```



```
.....  
    }  
  }  
}  
u32FrameIdx ++;  
} while(u32FrameIdx <= 1000);
```

3.3.10 按轮循方式抓拍

3.3.10.1 业务流程

创建通道时，创建的是抓拍通道，获取抓拍图像和一般的获取流程相同，具体内容请参见“[3.3.5 按帧获取编码码流](#)”和“[3.3.6 按包获取编码码流](#)”。

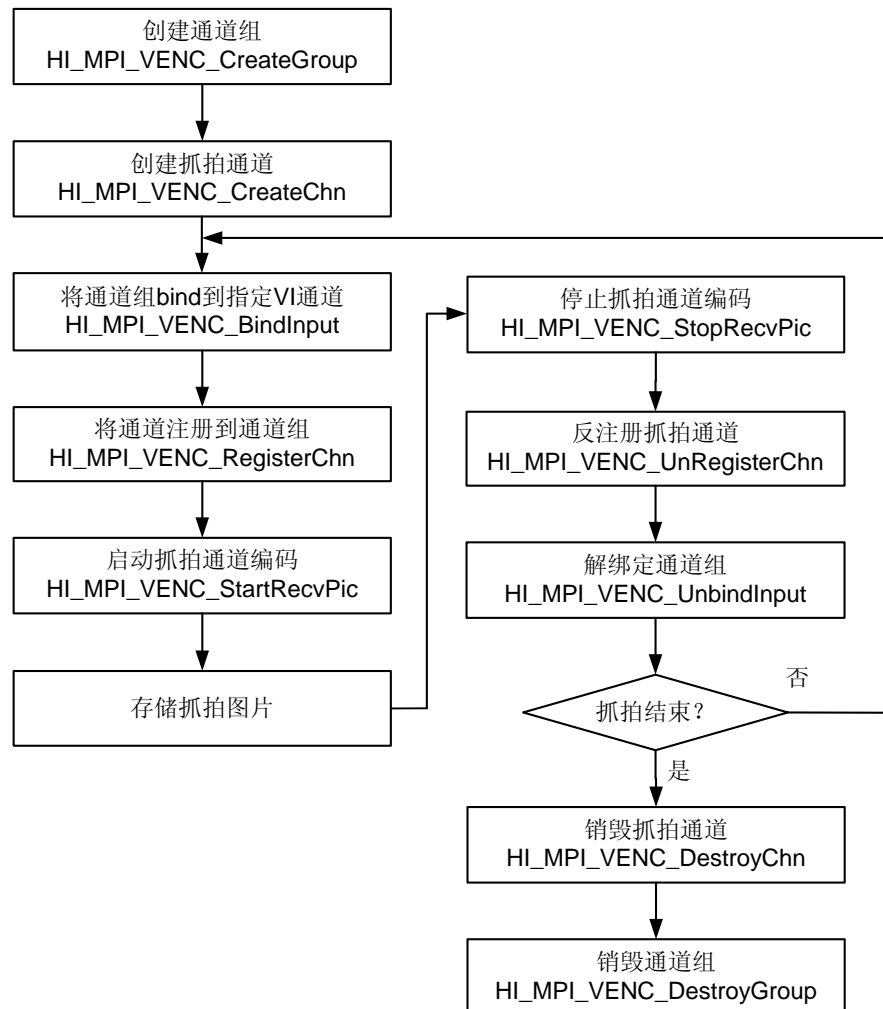
按轮循方式抓拍的特点：只需要创建一个专门用于抓拍的通道组和一个抓拍通道。执行一次抓拍时，将该通道组绑定到某个 VI 通道后启动编码，获取抓拍后的编码码流后停止编码，再解绑定该通道组。多通道抓拍时，循环地动态绑定到不同的 VI 通道，就能实现轮循抓拍所有 VI 通道图像的功能。

轮循抓拍方式有以下优缺点：

- 优点：节省内存，因为只有一个抓拍通道组和通道。
- 缺点：抓拍效率低于按并发方式，例如 8 路编码时，轮循地对每个通道抓拍 D1 大小的图像，最多达到每通道 1 张/s。因为对于多个通道，抓拍过程顺序进行，没有并发执行的效率高。

按轮循抓拍的业务流程如[图 3-13](#)所示。

图3-13 按轮循方式启动 1 路抓拍的业务流程



3.3.10.2 编程指导

按轮循方式抓拍的步骤如下：

1. 创建通道组，用于容纳抓拍通道。一个通道组只能容纳一个抓拍通道。
2. 创建编码通道。

创建需要的 JPEG 编码通道，比如可以创建 1D1 大小的 JPEG 编码通道或创建 1CIF 大小的 JPEG 编码通道，轮循抓拍方式时只需要创建一个抓拍通道。

3. 将通道组 bind 到指定 VI 通道。

Bind 到指定 VI 通道（由 VI 设备号和 VI 通道号确定）之后，此 VI 通道输入源可以是任意用户想抓拍的输入源通道。

4. 将通道注册到通道组。



关于 HI_MPI_VENC_RegisterChn 的注意事项，详细请参见《Hi3507 媒体处理软件开发参考》。

5. 启动抓拍通道编码。
6. 存储抓拍图像码流。
7. 停止抓拍通道编码。
8. 反注册抓拍通道。
9. 解绑定通道组。
10. 判断是否继续抓拍。

若继续抓拍请重复 3~10；若抓拍结束执行 11 和 12。

11. 销毁抓拍通道。
12. 销毁抓拍通道组。

----结束



说明

抓拍通道必须独占一个通道组，任何协议的主次码流都不能和抓拍存在于同一个通道组。即必须创建一个专门通道组供抓拍使用。

3.3.10.3 实例

```
/*创建抓拍通道组*/
s32Ret = HI_MPI_VENC_CreateGroup(VeGroup);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_CreateGroup err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

/*创建抓拍通道*/
s32Ret = HI_MPI_VENC_CreateChn(SnapChn, &stAttr, HI_NULL);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_CreateChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}
do
{
    sleep(2);
    sprintf(acFile, "JPEGstream%d.jpg", s32SnapNum);

    pFile = fopen(acFile,"wb");
```



```
if(pFile == NULL)
{
    continue;
}

s32Ret = HI_MPI_VENC_BindInput(VeGroup, ViDev, ViChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_BindInput err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_RegisterChn(VeGroup, SnapChn);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_RegisterChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_StartRecvPic(SnapChn);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StartRecvPic err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

/*存储抓拍图片*/
s32Ret = SampleSaveSnapPic(SnapChn,pFile);
if(s32Ret != HI_SUCCESS)
{
    printf("SampleSaveSnapPic err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_StopRecvPic(SnapChn);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StopRecvPic err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_UnRegisterChn(SnapChn);
if(s32Ret != HI_SUCCESS)
{
```



```
        printf("HI_MPI_VENC_UnRegisterChn err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_UnbindInput (VeGroup);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_UnbindInput err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32SnapNum++;

    fclose(pFile);

}while(s32SnapNum < 0xf);

s32Ret = HI_MPI_VENC_DestroyChn (SnapChn);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyGroup (VeGroup);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
```

3.3.11 按并发方式抓拍

3.3.11.1 业务流程

按并发方式抓拍的特点：存在多个抓拍组及通道并发的执行抓拍。即为每个要抓拍的输入源通道（VI 通道）创建一个抓拍组和抓拍通道，并绑定通道组和 VI 通道。

对某一通道抓拍多张时，只需要重复以下过程：注册抓拍通道至通道组后启动编码，获取抓拍后的编码码流后停止编码，再反注册通道。在此无须解绑定通道组。

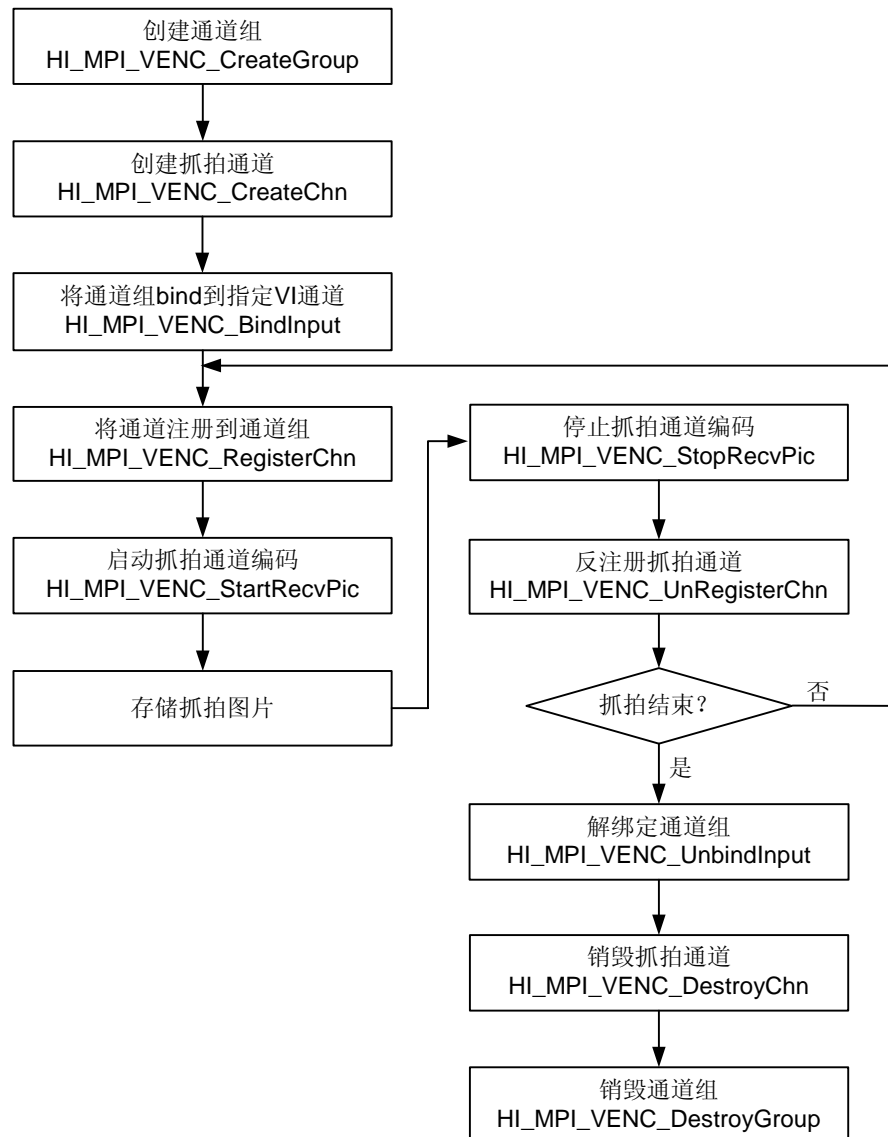
多通道抓拍时，各抓拍通道并发的执行抓拍任务，提高了效率。

并发抓拍方式有以下优缺点：

- 优点：抓拍效率高，例如 4 路 CIF 编码时，对每个通道抓拍 CIF 大小的图像（图像等级 3），最多每通道达到 25 张/s。
- 缺点：需要较多内存，因为存在多个抓拍通道组和通道。

按并发方式启动 1 路抓拍的业务流程如图 3-14 所示。

图3-14 按并发方式启动 1 路抓拍的业务流程



3.3.11.2 编程指导

按并发方式抓拍的步骤如下：

1. 为要抓拍的输入源通道（VI 通道）创建一个抓拍组，用于容纳抓拍通道。一个通道组只能容纳一个抓拍通道。
2. 为每个通道组创建一个 JPEG 编码通道。

创建需要的 JPEG 编码通道，比如可以创建 1D1 大小的 JPEG 编码通道或创建 1CIF 大小的 JPEG 编码通道，并发方式时同时存在多个抓拍通道。

3. 将每个通道组绑定到相应的 VI 通道。



绑定到指定 VI 通道（由 VI 设备号和 VI 通道号确定）之后，此 VI 通道输入源可以是任意用户想抓拍的输入源通道。

4. 将通道注册到通道组。

关于 HI_MPI_VENC_RegisterChn 的注意事项，详细请参见《Hi3507 媒体处理软件开发参考》。

5. 启动抓拍通道编码。

6. 存储抓拍图像码流。

7. 停止抓拍通道编码。

8. 反注册抓拍通道。

9. 判断是否继续抓拍。若继续抓拍请重复 4~9；若抓拍结束执行 10~12。

10. 解绑定通道组。

11. 销毁抓拍通道。

12. 销毁抓拍通道组。

----结束



说明

抓拍通道必须独占一个通道组，任何协议的主次码流都不能和抓拍存在于同一个通道组。即必须创建一个专门通道组供抓拍使用。

3.3.11.3 实例

```
/*创建抓拍通道组*/
s32Ret = HI_MPI_VENC_CreateGroup(VeGroup);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_CreateGroup err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

/*创建抓拍通道*/
s32Ret = HI_MPI_VENC_CreateChn(SnapChn, &stAttr, HI_NULL);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_CreateChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

/*绑定抓拍组*/
s32Ret = HI_MPI_VENC_BindInput(VeGroup, ViDev, ViChn);
if (s32Ret != HI_SUCCESS)
```



```
{
    printf("HI_MPI_VENC_BindInput err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

do
{
    sleep(2);
    sprintf(acFile, "JPEGstream%d.jpg", s32SnapNum);

    pFile = fopen(acFile,"wb");

    if(pFile == NULL)
    {
        continue;
    }

    s32Ret = HI_MPI_VENC_RegisterChn(VeGroup, SnapChn);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_RegisterChn err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_StartRecvPic(SnapChn);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StartRecvPic err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    /*存储抓拍图片*/
    s32Ret = SampleSaveSnapPic(SnapChn,pFile);
    if(s32Ret != HI_SUCCESS)
    {
        printf("SampleSaveSnapPic err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_StopRecvPic(SnapChn);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StopRecvPic err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }
}
```



```
    }

    s32Ret = HI_MPI_VENC_UnregisterChn(SnapChn);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_UnregisterChn err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }
    s32SnapNum++;

    fclose(pFile);

}while(s32SnapNum < 0xf);

s32Ret = HI_MPI_VENC_UnbindInput(VeGroup);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_UnbindInput err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_DestroyChn(SnapChn);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyGroup(VeGroup);
if (s32Ret != HI_SUCCESS)
{
    return HI_FAILURE;
}
}
```

3.4 VDEC 模块

3.4.1 概述

VDEC 模块提供驱动 Hi3507 芯片视频解码硬件工作的 MPI 接口，实现 H.264、MJPEG、JPEG 视频解码功能。VDEC 模块提供如下功能接口：

- HI_MPI_VDEC_CreateChn: 创建视频解码通道。
- HI_MPI_VDEC_DestroyChn: 销毁视频解码通道。
- HI_MPI_VDEC_BindOutput: 绑定视频解码通道到显示输出设备。
- HI_MPI_VDEC_UnbindOutput: 解绑定视频解码通道到显示输出设备。



- HI_MPI_VDEC_SetChnAttr: 设置指定的视频解码通道的动态属性。
- HI_MPI_VDEC_GetChnAttr: 获取指定的视频解码通道的属性（静态和动态）。
- HI_MPI_VDEC_SendStream: 用户发送码流到视频解码通道用于解码。
- HI_MPI_VDEC_GetData: 用户获取视频解码数据。
- HI_MPI_VDEC_ReleaseData: 用户释放已经获取的视频解码数据缓存。
- HI_MPI_VDEC_GetCapability: 获取指定算法协议的视频解码能力集。
- HI_MPI_VDEC_GetFd: 获取指定的视频解码通道的文件描述符。
- HI_MPI_VDEC_StartRecvStream: 启动获取码流。
- HI_MPI_VDEC_StopRecvStream: 停止获取码流。
- HI_MPI_VDEC_Query: 查询解码状态。

3.4.2 重要概念

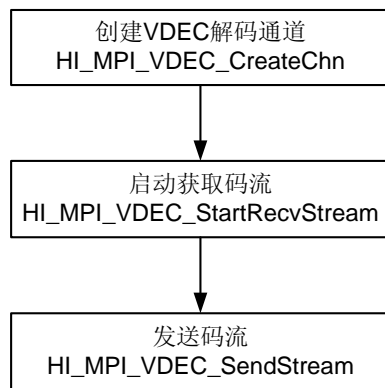
需要明确以下重要概念：解码支持流式发送，可发送任意大小的码流，但不允许超过码流 buf 大小。

3.4.3 配置并启动解码器

3.4.3.1 业务流程

配置并启动解码器的业务流程如图 3-15 所示。

图3-15 配置并启动解码器的业务流程



3.4.3.2 编程指导

配置并启动解码器的步骤如下：

1. 创建 VDEC 视频解码通道。
2. 启动 VDEC 接收码流包。
3. 发送码流。

----结束



说明

- 必须正确设置创建解码通道的输入参数 pstAttr (解码属性), 如果参数错误或者为空, 导致创建失败; 解码属性包括协议类型、码流 buf 大小以及各个解码协议的具体属性。
- 最多只支持创建 32 个视频解码通道, 有效通道号 0~31, 其他通道号视为无效通道号。
- 实例的数据结构及相关定义请参见《Hi3507 媒体处理软件 开发参考》和 SDK 的根目录下的 \sample\vdec\sample_vdec.c。
- 码流 buf 的大小至少为配置的图像属性宽高的 1.5 倍。

3.4.3.3 实例

```
static HI_S32 SampleCreateVdecchn(HI_S32 s32ChnID, PAYLOAD_TYPE_E enType,
void* pstAttr)
{
    VDEC_CHN_ATTR_S      stAttr;
    HI_S32 s32ret;

    memset(&stAttr, 0, sizeof(VDEC_CHN_ATTR_S));

    switch(enType)
    {
        case PT_H264:
        {
            VDEC_ATTR_H264_S stH264Attr;

            memcpy(&stH264Attr, pstAttr, sizeof(VDEC_ATTR_H264_S));

            stAttr.enType = PT_H264;
            stAttr.u32BufSize = (((stH264Attr.u32PicWidth) *
(stH264Attr.u32PicHeight))<<1);
            stAttr.pValue = (void*)&stH264Attr;
        }
        break;
        case PT_JPEG:
        {
            VDEC_ATTR_JPEG_S stJpegAttr;

            memcpy(&stJpegAttr, pstAttr, sizeof(VDEC_ATTR_JPEG_S));

            stAttr.enType = PT_JPEG;
            stAttr.u32BufSize =
            ((stJpegAttr.u32PicWidth*stJpegAttr.u32PicHeight) * 1.5);
            stAttr.pValue = (void*)&stJpegAttr;
        }
        break;
        default:
```



```
        Printf("err type \n");
        return HI_FAILURE;
    }

    s32ret = HI_MPI_VDEC_CreateChn(s32ChnID, &stAttr, NULL);
    if (HI_SUCCESS != s32ret)
    {
        Printf("HI_MPI_VDEC_CreateChn failed errno 0x%x \n", s32ret);
        return s32ret;
    }

    s32ret = HI_MPI_VDEC_StartRecvStream(s32ChnID);
    if (HI_SUCCESS != s32ret)
    {
        Printf("HI_MPI_VDEC_StartRecvStream failed errno 0x%x \n", s32ret);
        return s32ret;
    }

    return HI_SUCCESS;
}
```

3.4.4 向解码通道发送码流

3.4.4.1 业务流程

向解码通道发送码流分为阻塞方式和非阻塞方式，其业务流程分别如图 3-16 和图 3-17 所示。

图3-16 阻塞方式发送码流包的业务流程

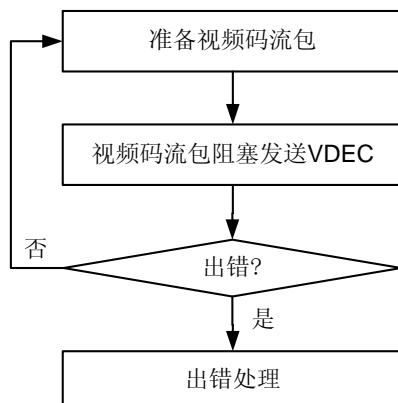
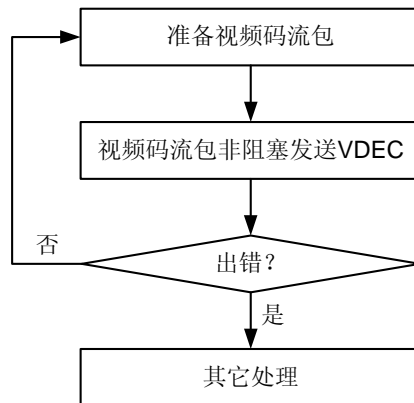


图3-17 非阻塞方式发送码流包的业务流程



3.4.4.2 编程指导

阻塞方式发送码流包的步骤如下：

1. 从文件或者编码器获取一段码流。
2. 阻塞方式发送到 VDEC 解码通道。
3. 如果发送成功，则重复 1 和 2；如果出错则做出错处理。

----结束

非阻塞方式发送码流包的流程如下：

1. 从文件或者编码器获取一段码流。
2. 非阻塞方式发送到 VDEC 解码通道。
3. 如果发送成功，则重复 1 和 2；如果发送失败，可转入其他操作。

----结束

3.4.4.3 实例

阻塞方式发送码流包的参考代码如下：

```
#define MAX_READ_LEN 1024
VDEC_STREAM_S stStream;
FILE* file = NULL;
HI_S32 s32ret;
HI_CHAR buf[MAX_READ_LEN];

/*open the stream file*/
file = fopen("test.h264", "r");
if (!file)
```



```
{
    perror("open file err\n");
    return NULL;
}
fseek(file, 0, SEEK_SET);

while(1)
{
    s32ret = fread(buf,1,MAX_READ_LEN,file);
    if(s32ret <=0)
    {
        fseek(file, 0, SEEK_SET);
        Printf("read nalu from file again:%d\n",s32ret);
        continue;
    }

    stStream.pu8Addr = buf;
    stStream.u32Len = s32ret;
    stStream.u64PTS = 0;

    s32ret = HI_MPI_VDEC_SendStream(0, &stStream, HI_IO_BLOCK);
    if (HI_SUCCESS != s32ret)
    {
        Printf("send to vdec 0x %x \n", s32ret);
        break;
    }
}

fclose(file);
```

非阻塞方式发送码流包的参考代码如下：

```
while(1)
{
    s32ret = fread(buf,1,MAX_READ_LEN,file);
    if(s32ret <=0)
    {
        fseek(file, 0, SEEK_SET);
        Printf("read nalu from file again:%d\n",s32ret);
        continue;
    }

    stStream.pu8Addr = buf;
    stStream.u32Len = s32ret;
    stStream.u64PTS = 0;
```

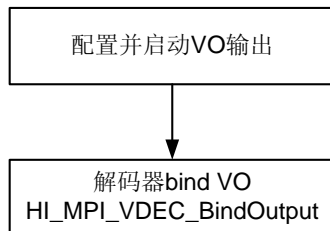
```
s32ret = HI_MPI_VDEC_SendStream(0, &stStream, HI_IO_NOBLOCK);  
if (HI_ERR_VDEC_BUF_FULL == s32ret)/*buf full,send failed*/  
{  
    /* 在这里添加其他操作 */  
}  
}
```

3.4.5 VDEC 通道绑定 VO 直接输出解码后图像

3.4.5.1 业务流程

VDEC 通道绑定 VO 直接输出解码后图像的业务流程如图 3-18 所示。

图3-18 VDEC 通道绑定 VO 直接输出解码后图像的业务流程



3.4.5.2 编程指导

VDEC 通道绑定 VO 直接输出解码后图像的步骤如下：

1. 配置并启动 VO 输出（请参见“3.2.3 配置并启动 VO 输出”）。
2. 绑定 VDEC 视频解码通道到指定的 VO 输出通道。

----结束

说明

使用 VDEC 通道绑定 VO 直接输出解码图像前，必须保证 VO 显示设备正常工作。当发送码流包启动解码后，如果显示终端始终没有显示图像，首先要检查被绑定的 VO 输出通道与启动的 VO 输出通道是否一致；然后检查是否启动 VO 设备。

3.4.5.3 实例

```
/*初始化vo通道*/  
static HI_S32 SampleVoInit(void)  
{  
    HI_S32 s32ret;  
    VO_PUB_ATTR_S VoAttr;  
  
    VoAttr.stTvConfig.stComposeMode = VIDEO_ENCODING_MODE_PAL; /*p制显示模式  
*/
```



```
VoAttr.u32BgColor = 0x0 ;

s32ret = HI_MPI_VO_SetPubAttr(&VoAttr);
if (HI_SUCCESS != s32ret)
{
    Printf("set VO attribute failed errno: 0x%x!\n", s32ret);
    return s32ret;
}

HI_MPI_VO_Enable();
return HI_SUCCESS;
}

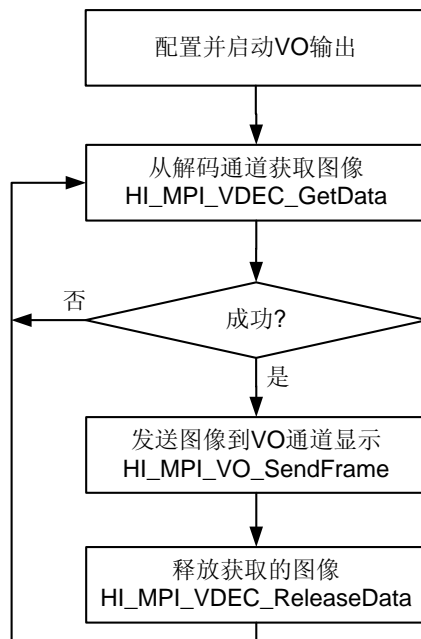
/*将解码通道绑定到VO*/
if(HI_SUCCESS != HI_MPI_VDEC_BindOutput(0, 0))
{
    printf("bind vdec to vo failed\n");
    /*错误处理*/
}
```

3.4.6 获取解码后图像送 VO 显示

3.4.6.1 业务流程

获取解码后图像送 VO 显示的业务流程如图 3-19 所示。

图3-19 获取解码后图像送 VO 显示的业务流程





3.4.6.2 编程指导

获取解码后图像送 VO 显示的步骤如下：

1. 配置并启动 VO 输出（请参见“3.2.3 配置并启动 VO 输出”）。
2. 创建一个线程用于接收解码后的数据，用户可以通过阻塞、非阻塞或者 select 方式来获取图像信息。
3. 判断是否获取成功，如果成功，则发送图像信息到 VO 通道显示；如果失败则需要重新获取图像信息。
4. 释放获取到的图像信息。
5. 进入下一个循环，从解码通道获取下一帧图像。

----结束

说明

- 本模块未提供停止解码接口，若停止解码则需要用销毁通道来实现解码停止。
- 如果接收数据以阻塞方式实现，要先销毁通道，才能销毁接收数据线程。

3.4.6.3 实例

```
HI_S32 s32ret;
while(1)
{
    VDEC_DATA_S stVdecData;
    s32ret = HI_MPI_VDEC_GetData(0, &stVdecData, HI_IO_BLOCK);
    if (HI_SUCCESS == s32ret)
    {
        if(stVdecData.stFrameInfo.bValid)
        {
            HI_MPI_VO_SendFrame(0,
&stVdecData.stFrameInfo.stVideoFrameInfo);
            usleep(40000);
        }

        HI_MPI_VDEC_ReleaseData(0, &stVdecData);
    }
    else
    {
        printf("channel had been destroyed\n");
        return NULL;
    }
}

return NULL;
```

3.4.7 停止并销毁视频解码

3.4.7.1 业务流程

停止并销毁视频解码可以选择两种方式：

- 强制停止现在的工作，放弃未解完的码流和未获取的图像并销毁解码通道。
- 等待解码完成，所有图像获取或者显示完毕再退出解码。

两种方式唯一的区别在于是否查询目前的状态，根据现在的状态来决定是否销毁解码通道。具体业务流程如图 3-20 和图 3-21 所示。

图3-20 强行停止并销毁视频解码的业务流程

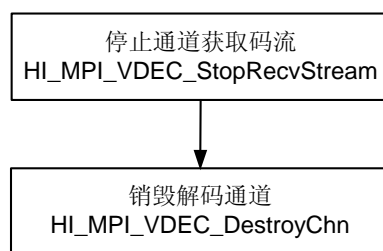
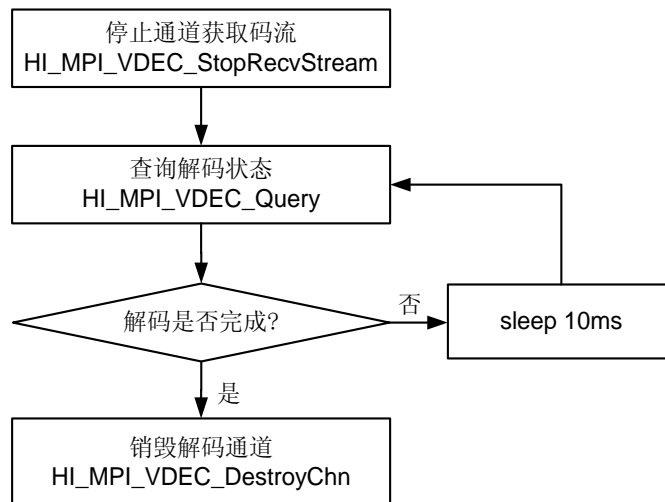


图3-21 等待解码完成销毁视频解码的业务流程



3.4.7.2 编程指导

强行停止并销毁视频解码的步骤如下：

1. 停止通道获取码流。

通知解码器停止解码，不再接收新码流，此时若再发送码流会返回失败。



2. 销毁解码通道。

销毁解码通道所有资源。

----结束

等待解码完成销毁视频解码的步骤如下：

1. 停止通道获取码流。
2. 查询解码器状态，如果还有残留的码流或者图像，则 sleep10ms，再次查询解码器状态，直到解码器运行完成。
3. 销毁解码通道。

----结束

3.4.7.3 实例

强行停止并销毁视频解码的参考代码：

```
HI_MPI_VDEC_StopRecvStream(s32ChnID);  
HI_MPI_VDEC_DestroyChn(s32ChnID);
```

等待解码完成并销毁视频解码的参考代码：

```
ret = HI_MPI_VDEC_StopRecvStream(s32ChnID);  
if(ret!=HI_SUCCESS) return;  
while(1)  
{  
    usleep(10000);  
    ret = HI_MPI_VDEC_Query(s32ChnID, &stStat);  
    if(ret!=HI_SUCCESS) return;  
    if(stStat.u32LeftPics == 0 && stStat.u32LeftStreamBytes == 0 )  
    {  
        printf("had no stream and pic left\n");  
        break;  
    }  
}  
HI_MPI_VDEC_DestroyChn(s32ChnID);
```

3.5 VPP 模块

3.5.1 概述

视频前处理模块对输入或者是 VI 捕获的图像所作的处理都在编码前进行，主要提供设置和获取视频前处理配置、创建和销毁 VPP 区域、控制 VPP 区域、创建和等待缩放任务完成功能。VPP 模块提供如下功能接口：



- HI_MPI_VPP_SetConf: 设置视频前处理配置。
- HI_MPI_VPP_GetConf: 获取视频前处理配置。
- HI_MPI_VPP_CreateRegion: 创建 VPP 区域。
- HI_MPI_VPP_DestroyRegion: 销毁 VPP 区域。
- HI_MPI_VPP_ControlRegion: 控制 VPP 区域。
- HI_MPI_VPP_CreateScaleTask: 创建一个图像缩放任务。
- HI_MPI_VPP_WaitScaleTask: 等待一个缩放任务完成。

3.5.2 重要概念

需要明确以下重要概念:

- Cover Region
视频遮挡区域。
- Overlay Region
视频叠加区域, 针对码流里打的叠加区域, 即通常所说的码流 OSD。
- Soverlay Region
视频软叠加区域, 针对预览通道的叠加区域, 即通常所说的预览 OSD。
- 区域层次 Layer
区域层次是针对遮挡区域、软叠加区域而言, 范围是[0, 100], 0 表示最底层, 100 表示最上层。
- 区域透明度
区域透明度是针对叠加区域、软叠加区域而言, 当前叠加区域对于其下一层图像的透明程度, 范围是[0, 128], 0 表示完全透明, 128 表示完全不透明。

3.6 MD 模块

3.6.1 概述

移动侦测是检测正在视频编码的图像是否发生亮度变化以及相应的运动向量。移动侦测通道就是视频编码通道, 最大支持 32 路的移动侦测 (编码通道号[0, 31])。

Hi3507 提供的移动侦测功能以宏块为最小单位, 计算指定图像的宏块在指定图像间隔内的亮度变化和运动向量。如需要获取移动侦测的结果, 则启用某一视频编码通道的移动侦测功能。移动侦测的结果包括宏块的 SAD 值、宏块运动向量 MV、宏块报警信息、宏块报警像素的个数。

MD (Move Detect) 模块支持 H.264 编码和 MJPEG 编码时进行移动侦测, 针对同一 VI 通道的主次码流, 二者只能有一个作 MD。

MD 模块提供如下功能接口:

- HI_MPI_MD_EnableChn: 启用某一路视频编码通道的移动侦测功能。
- HI_MPI_MD_DisableChn: 禁用某一路视频编码通道的移动侦测功能。
- HI_MPI_MD_SetChnAttr: 设置移动侦测的属性。



- HI_MPI_MD_GetChnAttr: 获取移动侦测的属性。
- HI_MPI_MD_SetRefFrame: 设置移动侦测的参考图像属性。
- HI_MPI_MD_GetRefFrame: 获取移动侦测的参考图像属性。
- HI_MPI_MD_GetData: 获取移动侦测结果。
- HI_MPI_MD_ReleaseData: 释放移动侦测结果。
- HI_MPI_MD_GetFd: 获取移动侦测通道对应的设备文件句柄。

3.6.2 重要概念

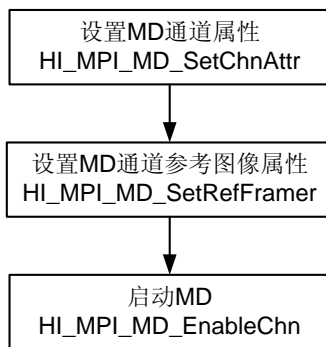
无。

3.6.3 配置并启动 MD

3.6.3.1 业务流程

配置并启动 MD 的业务流程图 3-22 如所示。

图3-22 配置并启动 MD 的业务流程



3.6.3.2 编程指导

配置并启动 MD 的步骤如下：

1. 设置 MD 通道属性。
在设置 MD 通道属性之前必须要保证对应的移动侦测通道为禁用状态。
2. 设置 MD 参考图像属性。
在设置 MD 参考图像属性之前必须要保证对应的移动侦测通道为禁用状态。
3. 在启动 MD。
在启用移动侦测之前必须要设置 MD 通道属性、参考图像属性。

----结束



3.6.3.3 实例

```
HI_S32 s32Ret;
VENC_CHN VeChn = 0;
MD_CHN_ATTR_S stMdAttr;
MD_REF_ATTR_S stRefAttr;

/*set MD attribute*/
stMdAttr.stMBMode.bMBSADMode = HI_TRUE;
stMdAttr.stMBMode.bMBVMMode = HI_FALSE;
stMdAttr.stMBMode.bMBPelNumMode = HI_FALSE;
stMdAttr.stMBMode.bMBALARMMode = HI_FALSE;
stMdAttr.u16MBALSADTh = 1000;
stMdAttr.u8MBPelALTh = 20;
stMdAttr.u8MBPerALNumTh = 20;
stMdAttr.enSADBits = MD_SAD_8BIT;
stMdAttr.stDlight.bEnable = HI_FALSE;
stMdAttr.u32MDInternal = 0;
stMdAttr.u32MDBufNum = 16;

/*set MD frame*/
stRefAttr.enRefFrameMode = MD_REF_AUTO;
stRefAttr.enRefFrameStat = MD_REF_DYNAMIC;

s32Ret = HI_MPI_MD_SetChnAttr(VeChn, &stMdAttr);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_MD_SetChnAttr Err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_MD_SetRefFrame(VeChn, &stRefAttr);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_MD_SetRefFrame Err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_MD_EnableChn(VeChn);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_MD_EnableChn Err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
```

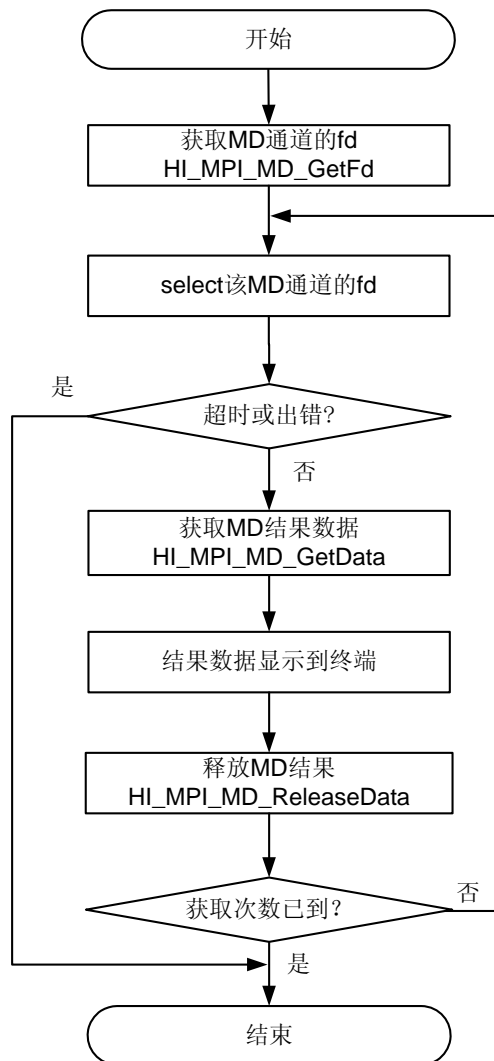
```
return HI_SUCCESS;
```

3.6.4 获取 MD 结果数据

3.6.4.1 业务流程

获取 MD 结果的业务流程如图 3-23 所示。

图3-23 获取 MD 结果的业务流程



3.6.4.2 编程指导

获取 MD 结果的步骤如下：

1. 获取 MD 通道文件句柄 fd。



2. Select 该 MD 通道的 fd。
3. 获取 MD 结果数据。
4. 结果数据显示到终端。
5. 释放 MD 结果。

----结束

3.6.4.3 实例

```
HI_S32 s32Ret;
HI_S32 s32MdFd;
HI_S32 s32Cnt = 0;
MD_DATA_S stMdData;
MD_DATA_S *pstMdData;
VENC_CHN VeChn = VENCCHNID;
fd_set read_fds;
struct timeval TimeoutVal;

s32MdFd = HI_MPI_MD_GetFd(VeChn);

do{
    FD_ZERO(&read_fds);
    FD_SET(s32MdFd, &read_fds);

    TimeoutVal.tv_sec = 2;
    TimeoutVal.tv_usec = 0;
    s32Ret = select(s32MdFd+1, &read_fds, NULL, NULL, &TimeoutVal);

    if (s32Ret < 0)
    {
        printf("select err\n");
        return NULL;
    }
    else if (0 == s32Ret)
    {
        printf("time out\n");
        return NULL;
    }
    else
    {
        memset(&stMdData, 0, sizeof(MD_DATA_S));

        if (FD_ISSET(s32MdFd, &read_fds))
```



```
{
    s32Ret = HI_MPI_MD_GetData(VeChn, &stMdData, HI_IO_NOBLOCK);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_MD_GetData err 0x%x\n",s32Ret);
        return NULL;
    }
}

pstMdData = &stMdData;

/*get MD SAD data*/
if (pstMdData->stMBMode.bMBSADMode)
{
    HI_S32 i,j;
    HI_U16* pTmp = NULL;

    for(i=0; i<pstMdData->u16MBHeight; i++)
    {
        pTmp = (HI_U16 *) ((HI_U32)pstMdData->stMBSAD.pu32Addr +
                           i*pstMdData->stMBSAD.u32Stride);

        for(j=0; j<pstMdData->u16MBWidth; j++)
        {
            printf("%2d",*pTmp);
            pTmp++;
        }

        printf("\n");
    }
}

/*get MD MV data*/
if (pstMdData->stMBMode.bMBMVMode)
{
    HI_S32 i,j;
    HI_U16* pTmp = NULL;

    for(i=0; i<pstMdData->u16MBHeight; i++)
    {
        pTmp = (HI_U16 *) ((HI_U32)pstMdData->stMBMV.pu32Addr +
                           i*pstMdData->stMBMV.u32Stride);

        for(j=0; j<pstMdData->u16MBWidth; j++)
```



```
        {
            printf("%2d", *pTmp);
            pTmp++;
        }

        printf("\n");
    }
}

/*get MD MB alarm data*/
if(pstMdData->stMBMode.bMBALARMMode)
{
    HI_S32 i, j, k;
    HI_U8* pTmp = NULL;
    for(i=0; i<pstMdData->u16MBHeight; i++)
    {
        pTmp = (HI_U8 *)((HI_U32)pstMdData->stMBAlarm.pu32Addr +
                        i*pstMdData->stMBAlarm.u32Stride);

        for(j=0; j<pstMdData->u16MBWidth; j++)
        {
            k = j%8;
            if(j != 0 && k==0)
            {
                pTmp++;
            }

            printf("%2d", ((*pTmp)>>k)&0x1);
        }

        printf("\n");
    }
}

/*get MD MB alarm pels number data*/
if(pstMdData->stMBMode.bMBPelNumMode)
{
    HI_S32 i, j;
    HI_U8* pTmp = NULL;
    for(i=0; i<pstMdData->u16MBHeight; i++)
    {
        pTmp = (HI_U8 *)((HI_U32)pstMdData->stMBPelAlarmNum.pu32Addr + (i*pstMdData->stMBPelAlarmNum.u32Stride));
```




```
        for(j=0; j<pstMdData->u16MBWidth; j++)
        {
            printf("%2d",*pTmp);
            pTmp++;
        }

        printf("\n");
    }
}
s32Ret = HI_MPI_MD_ReleaseData(VeChn, &stMdData);
if(s32Ret != HI_SUCCESS)
{
    printf("Md Release Data Err 0x%x\n",s32Ret);
    return NULL;
}

s32Cnt++;

}
}while (s32Cnt < 0xffff);

return NULL;
```

说明

本实例实现配置并启用移动侦测的方法，更详尽的实例代码还请参见 SDK 根目录下的 \sample\md\sample_md.c 下的 HI_VOID *SampleGetMdData(HI_VOID *p)函数。

3.7 AUDIO 模块

3.7.1 概述

AUDIO 模块包括音频输入、音频输出、音频编码、音频解码四个子模块。音频输入和输出模块通过对 Hi3507 芯片 SIO 设备的控制实现相应的音频输入输出功能；音频编码和解码模块则提供对 ADPCM、G726、G711、AMR 格式的音频编解码功能。

AUDIO 模块提供如下功能接口：

- HI_MPI_AI_SetPubAttr：设置 AI 设备属性。
- HI_MPI_AI_GetPubAttr：获取 AI 设备属性。
- HI_MPI_AI_Enable：启用 AI 设备。
- HI_MPI_AI_Disable：禁用 AI 设备。
- HI_MPI_AI_EnableChn：启用 AI 通道。
- HI_MPI_AI_DisableChn：禁用 AI 通道。



- HI_MPI_AI_GetFrame: 获取 AI 通道音频帧数据。
- HI_MPI_AI_EnableAec: 启用回声抵消功能。
- HI_MPI_AI_DisableAec: 禁用回声抵消功能。
- HI_MPI_AI_GetFd: 获取 AI 通道对应设备文件句柄。
- HI_MPI_AO_SetPubAttr: 设置 AO 设备属性。
- HI_MPI_AO_GetPubAttr: 获取 AO 设备属性。
- HI_MPI_AO_Enable: 启用 AO 设备。
- HI_MPI_AO_Disable: 禁用 AO 设备。
- HI_MPI_AO_EnableChn: 启用 AO 通道。
- HI_MPI_AO_DisableChn: 禁用 AO 通道。
- HI_MPI_AO_PauseChn: 暂停 AO 通道。
- HI_MPI_AO_ResumeChn: 恢复 AO 通道。
- HI_MPI_AO_SendFrame: 发送音频帧到 AO 通道。
- HI_MPI_AO_GetFd: 获取 AO 通道对应的设备文件句柄。
- HI_MPI_AENC_CreateChn: 创建音频编码通道。
- HI_MPI_AENC_DestroyChn: 销毁音频编码通道。
- HI_MPI_AENC_SetChnAttr: 设置音频编码通道属性。
- HI_MPI_AENC_GetChnAttr: 获取音频编码通道属性。
- HI_MPI_AENC_SendFrame: 发送音频帧到音频编码通道。
- HI_MPI_AENC_GetStream: 获取音频编码码流。
- HI_MPI_AENC_ReleaseStream: 释放音频编码码流。
- HI_MPI_ADEC_CreateChn: 创建音频解码通道。
- HI_MPI_ADEC_DestroyChn: 销毁音频解码通道。
- HI_MPI_ADEC_SetChnAttr: 设置音频解码通道属性。
- HI_MPI_ADEC_GetChnAttr: 获取音频解码通道属性。
- HI_MPI_ADEC_SendStream: 发送音频码流到音频解码通道。
- HI_MPI_ADEC_GetData: 获取解码后音频帧数据。
- HI_MPI_ADEC_ReleaseData: 释放解码后音频帧数据。

3.7.2 重要概念

- Hi3507 SIO 简要介绍

Hi3507 提供 2 个 SIO 接口，每 1 个 SIO 都同时包含输入和输出功能（SIO1 由于输出引脚未引出，输出功能不能使用），可以选择在任意一个 SIO 上接入 AD/DA 来完成音频输入/输出。

SIO 支持标准的 I²S 接口，左右声道传输宽度均为固定的 32bit，即 1 个 SIO 的左右声道各自最多都能够传输 4 路 8bit 数据，左右声道加起来可支持的情况如下：

- 可支持 8 路 8bit 数据
- 可支持 4 路 16bit 数据
- 可支持 2 路 32bit 数据

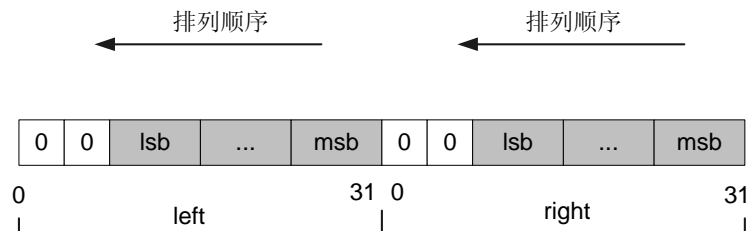
- SIO 的主从模式
 - 如果 SIO 设置为从模式，则音频 AD 或 DA 工作在主模式，即音频 AD 或 DA 提供时钟。
 - 如果 SIO 设置为主模式，即 SIO 提供时钟，则音频 AD 或 DA 工作在从模式。由于同一个 SIO 的 AI 和 AO 设备使用的时钟相同，因此设备号相同的 AI、AO 的主从模式、采样率配置应该保持一致。

Hi3507 SIO 本身并不能真正工作在主模式，SDK 中的 SIO 主模式实际上时钟是由 Hi3507 芯片的 CRG 分频得出（CRG 分频的相关配置请参考《Hi3507 H.264 编解码处理器 用户指南》），即 CRG 为主，SIO 和 AD/DA 为从。如果 CRG 无法分频出正常工作所需要的时钟（例如 TW2864 所需要的 8K 帧时钟、2.048M 位时钟），此时必须使用 SIO 从模式，同时需要关闭 CRG（否则出现 AD 和 CRG 都输出时钟的情况），具体操作为将寄存器 SC_PERCTRL4（0x101E003C）第 7 位配置为 0；另外需要注意的一个地方是：AD 为主设备提供时钟，使用 SIO0 的输出功能时，需要将 SIO0 的发送帧指示信号和发送时钟信号配置为从管脚 RFS 和 RCK 输入，具体操作为将寄存器 SC_PERCTRL1（0x101E0020）第 30、31 位配置为 1。系统控制及管脚复用都在 U-boot 中配置。

- SIO 的通道排列

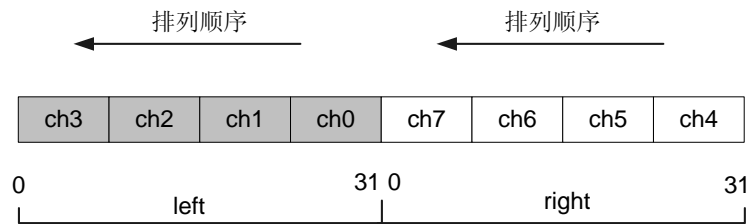
SIO 传输的数据宽度为左右声道各 32bit，根据 I²S 时序要求，SIO 传输数据时会把 AD 采集的数据从高位向低位依次排列，如果 AD 配置的采样精度低于 32bit（例设置 AD 采样精度为 8bit 或者 16bit），硬件会在未达到宽度的低位上补 0，具体如图 3-24 所示。

图3-24 Codec 数据在 SIO 传输通路的排列



AI、AO 的通道实际为逻辑上的通道，AI、AO 的采样精度也是逻辑上的采样精度（反映某个通道从传输通路按多少 bit 宽度去取数据）；为了明确通道号和传输通路中数据的对应关系，规定 SIO 中，通道数据的排列顺序与 I²S 取数据的顺序相同，从高位向低位排列。8bit 数据宽度的情况下，SIO 传输通路最多容纳 8 路单声道的数据，排列如图 3-25 所示。

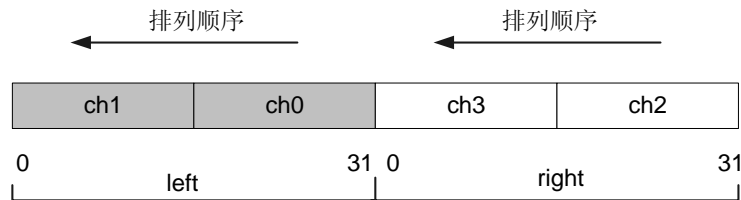
图3-25 8bit 采样精度下 SIO 传输通路中通道排列



取 ch0 的数据就是取传输通路中最高 8 位的数据，对应的 ch1 是次高 8 位，ch2 和 ch3 分别是次低 8 位和最低 8 位。

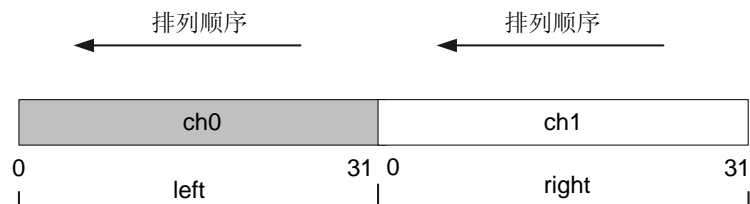
16bit 数据宽度的情况下，SIO 传输通路最多容纳 4 路单声道的数据，排列如图 3-26 所示。

图3-26 16bit 采样精度下 SIO 传输通路中通道排列



32bit 数据宽度的情况下，SIO 传输通路最多容纳 2 路单声道数据，排列如图 3-27 所示。

图3-27 32bit 采样精度下 SIO 传输通路中通道排列



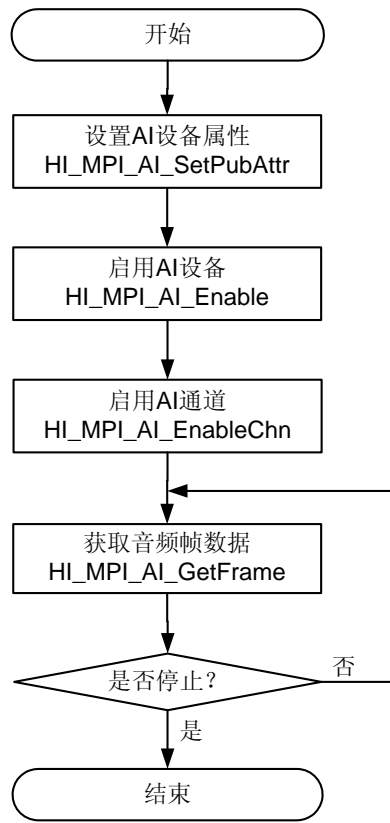
必须理解 AD 采集数据和通道的对应关系，才能从正确的通道获取数据。不论使用何种 AD 芯片，必须遵守上述通道分布和约束才能正确工作。

3.7.3 启用 AI 设备并获取音频帧

3.7.3.1 业务流程

启用 AI 设备并获取音频帧数据的业务流程如图 3-28 所示。

图3-28 启用 AI 设备并获取音频帧数据的业务流程



3.7.3.2 编程指导

启用 AI 设备并获取音频帧数据的步骤如下：

1. 设置 AI 设备属性。

启用 AI 设备之前需要先对 AI 设备的属性做相应设置。

2. 启用 AI 设备。

先启用 AI 设备，再启用 AI 通道。

3. 启用 AI 通道。

成功启用 AI 通道后，AI 通道才会开始运行，启动音频帧数据的捕获。

4. 获取音频帧数据。

从 AI 设备中的某个通道获取音频帧数据（可选择阻塞或非阻塞方式，也可以使用 select 接口）。

5. 判断是否停止，如不停止则重复获取音频帧数据，直至线程结束。

----结束



说明

- 需先设置 AI 设备属性，再启用 AI 设备；AI 设备启用之后不能再动态设置设备属性（除非先禁用 AI 设备）。
- AI 通道启用成功后才能从其中的 AI 通道获取音频帧数据；AI 设备中包含的 AI 通道个数由 AI 设备的采样精度属性决定。
- 数据结构的说明及接口注意事项请参见《Hi3507 媒体处理软件 开发参考》

3.7.3.3 实例

```
HI_S32 s32ret;
AIO_ATTR_S stAttr;
AUDIO_DEV AiDevId = 1;
AI_CHN AiChn;

stAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAttr.enSamplerate = AUDIO_SAMPLE_RATE_8000;
stAttr.enSoundmode = AUDIO_SOUND_MODE_MOMO;
stAttr.enWorkmode = AIO_MODE_I2S_SLAVE;
stAttr.u32EXFlag = 0;
stAttr.u32FrmNum = 5;
stAttr.u32PtNumPerFrm = 160;

/* set public attribute of AI device*/
s32ret = HI_MPI_AI_SetPubAttr(AiDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("set ai %d attr err:0x%x\n", AiDevId,s32ret);
    return s32ret;
}
/* enable AI device */
s32ret = HI_MPI_AI_Enable(AiDevId);
if(HI_SUCCESS != s32ret)
{
    printf("enable ai dev %d err:0x%x\n", AiDevId, s32ret);
    return s32ret;
}
/* enable AI channel*/
s32ret = HI_MPI_AI_EnableChn(AiDevId, AiChn);
if(HI_SUCCESS != s32ret)
{
    printf("enable ai channel (%d,%d) err:0x%x\n", AiDevId, AiChn,
s32ret);
    return s32ret;
}
/* get audio frame */
```



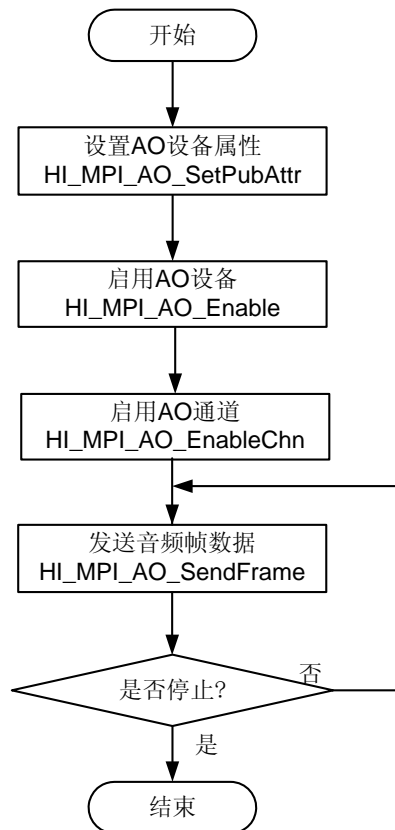
```
s32ret = HI_MPI_AI_GetFrame(AiDevId, AiChn, &stData, NULL, HI_IO_BLOCK);  
if(HI_SUCCESS != s32ret)  
{  
    printf("get ai frame err:0x%x\n", s32ret);  
    return s32ret;  
}  
/* ... ... */
```

3.7.4 启用 AO 设备并发送音频帧

3.7.4.1 业务流程

启用 AO 设备并发送音频帧数据的业务流程如图 3-29 所示。

图3-29 启用 AO 设备并发送音频帧数据的业务流程



3.7.4.2 编程指导

启用 AO 设备并发送音频帧数据的步骤如下：

1. 设置 AO 设备属性。

启用 AO 设备之前需要先对 AO 设备的属性做相应设置。



2. 启用 AO 设备。

先启用 AO 设备，再启用 AO 通道。

3. 启用 AO 通道。

成功启用 AI 通道后，AO 通道才会开始运行，启动音频帧数据的捕获。

4. 发送音频帧数据。

将音频帧数据发送到 AO 设备的某个通道输出（可以选择阻塞或非阻塞方式，也可以使用 Select 接口）。

5. 重复 4，直至线程结束。

----结束

 说明

- 需先设置 AO 设备属性，再启用 AO 设备；AO 设备启用之后不能再动态设置设备属性（除非先禁用 AO 设备）。
- AO 通道启用成功后才能将音频帧数据发送到其中某路通道。AO 设备中包含的 AO 通道个数由 AO 设备的采样精度属性决定。
- 数据结构的说明及接口注意事项请参见《Hi3507 媒体处理软件 开发参考》。

3.7.4.3 实例

```
HI_S32 s32ret;
AIO_ATTR_S stAttr;
AUDIO_DEV AoDevId = 0;
AO_CHN AoChn = 0;
AUDIO_FRAME_S stData;

stAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAttr.enSamplerate = AUDIO_SAMPLE_RATE_8000;
stAttr.enSoundmode = AUDIO_SOUND_MODE_MOMO;
stAttr.enWorkmode = AIO_MODE_I2S_SLAVE;
stAttr.u32EXFlag = 0;
stAttr.u32FrmNum = 5;
stAttr.u32PtNumPerFrm = 160;
/* set ao public attr*/
s32ret = HI_MPI_AO_SetPubAttr(AoDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId,s32ret);
    return s32ret;
}
/* enable ao device*/
s32ret = HI_MPI_AO_Enable(AoDevId);
if(HI_SUCCESS != s32ret)
```




```
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, s32ret);
    return s32ret;
}
/* enable AO channel*/
s32ret = HI_MPI_AO_EnableChn(AoDevId, AoChn);
if(HI_SUCCESS != s32ret)
{
    printf("enable ao channel (%d,%d) err:0x%x\n", AoDevId, AoChn,
s32ret);
    return s32ret;
}
/* get audio frame from ADEC chn or file*/

/* send audio frame to AO chn */
s32ret = HI_MPI_AO_SendFrame(AoDevId, AoChn, &stData, HI_IO_BLOCK);
if(HI_SUCCESS != s32ret)
{
    printf("send frame to ao err:0x%x\n", s32ret);
    return s32ret;
}
/* ... .. */
```

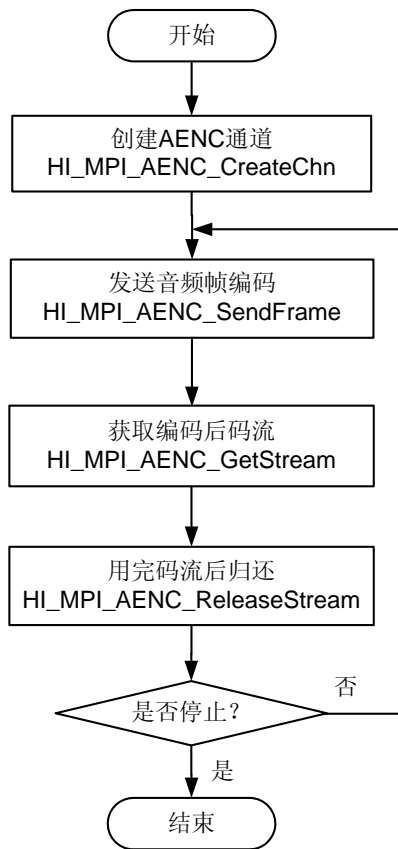
3.7.5 创建 AENC 通道和发送音频帧编码并获取码流

3.7.5.1 业务流程

创建 AENC 通道和发送音频帧编码并获取码流的业务流程如[图 3-30](#) 所示。



图3-30 创建 AENC 通道和发送音频帧编码并获取码流的业务流程



3.7.5.2 编程指导

创建 AENC 通道和发送音频帧编码并获取码流的步骤如下：

1. 创建 AENC 通道。

创建音频编码通道，并设置相应的编码协议的编码通道属性（支持 ADPCM、G726、G711、AMR 编码格式）。

2. 发送音频帧编码。

将音频帧数据（从 AI 获取）发送到音频编码通道进行相应编码格式的编码。

3. 获取编码后码流。

从编码通道获取码流。

4. 用完码流后归还。

音频码流使用完后（例如存文件或发网络），应该及时归还。

5. 重复 2~4，直至线程结束。

----结束



说明

数据结构的说明及接口注意事项请参见《Hi3507 媒体处理软件 开发参考》

3.7.5.3 实例

```
HI_S32 s32ret;
AENC_CHN_ATTR_S stAencAttr;
AENC_ATTR_ADPCM_S stAdpcmAenc;
AENC_CHN AencChn = 0;
AUDIO_FRAME_S stAudioFrm;
AUDIO_STREAM_S stAudioStream;

stAencAttr.enType = PT_ADPCMA; /* ADPCM */
stAencAttr.u32BufSize = 8;
stAencAttr.pValue = &stAdpcmAenc;
stAdpcmAenc.enADPCMTYPE = ADPCM_TYPE_DVI4;

/* create aenc chn*/
s32ret = HI_MPI_AENC_CreateChn(AencChn, &stAencAttr);
if (HI_SUCCESS != s32ret)
{
    printf("create aenc chn %d err:0x%x\n", AencChn,s32ret);
    return s32ret;
}

/* get audio frame form AI or file */

/* send audio frmae to AENC chn */
s32ret = HI_MPI_AENC_SendFrame(AencChn, &stAudioFrm, NULL);
if (HI_SUCCESS != s32ret)
{
    printf("send frame to aenc chn %d err:0x%x\n", AencChn,s32ret);
    return s32ret;
}

/* get stream from aenc chn */
s32ret = HI_MPI_AENC_GetStream(AencChn, &stAudioStream);
if (HI_SUCCESS != s32ret )
{
    printf("get stream from aenc chn %d fail \n", AencChn);
    return s32ret;
}

/* deal with audio stream */
```



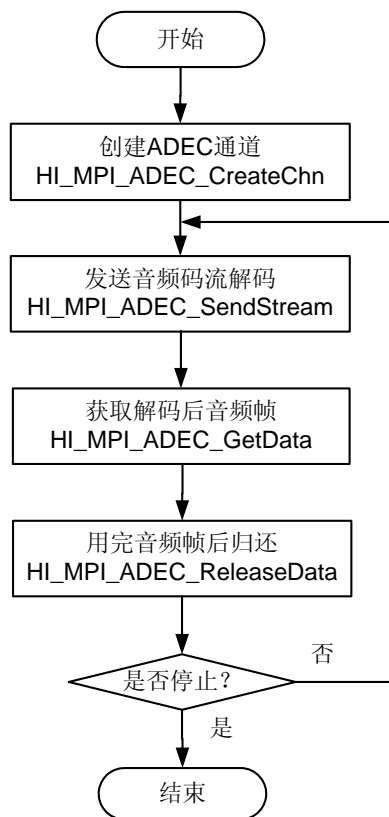
```
/* release audio stream */  
s32ret = HI_MPI_AENC_ReleaseStream(AencChn, &stAudioStream);  
if (HI_SUCCESS != s32ret )  
{  
    return s32ret;  
}
```

3.7.6 创建 ADEC 通道和发送码流解码并获取解码数据

3.7.6.1 业务流程

创建 ADEC 通道和发送码流解码并获取解码数据的业务流程如图 3-31 所示。

图3-31 创建 ADEC 通道和发送码流解码并获取解码数据的业务流程



3.7.6.2 编程指导

创建 ADEC 通道和发送码流解码并获取解码数据的步骤如下：

1. 创建 ADEC 通道。

创建音频解码通道，并设置相应的解码协议的解码通道属性（支持 ADPCM、G726、G711、AMR 解码格式）。



2. 发送音频码流解码。

将音频码流数据（可以来自文件或网络）发送到音频解码通道进行相应解码格式的解码。

3. 获取解码后音频帧。

从解码通道获取解码后音频帧数据。

4. 用完音频帧后归还。

音频帧数据使用完后（例如发送至 AO 输出），应该及时归还。

5. 重复 2~4，直至线程结束。

----结束



说明

数据结构说明及接口注意事项请参见《Hi3507 媒体处理软件 开发参考》

3.7.6.3 实例

```
HI_S32 s32ret;
ADEC_CHN_ATTR_S stAdecAttr;
ADEC_ATTR_ADPCM_S stAdpcm;
ADEC_CHN AdChn = 0;
AUDIO_STREAM_S stAudioStream;
AUDIO_FRAME_INFO_S stAudioFrameInfo;

stAdecAttr.enType = PT_ADPCMA;
stAdecAttr.u32BufSize = 8;
stAdecAttr.enMode = ADEC_MODE_STREAM;
stAdecAttr.pValue = &stAdpcm;
stAdpcm.enADPCMType = ADPCM_TYPE_DVI4;

/* create adec chn*/
s32ret = HI_MPI_ADEC_CreateChn(AdChn, &stAdecAttr);
if (s32ret)
{
    printf("create adnc chn %d err:0x%x\n", AdChn,s32ret);
    return s32ret;
}

/* get audio stream from network or file*/

/* send audio stream to adec chn */
s32ret = HI_MPI_ADEC_SendStream(AdChn, &stAudioStream);
if (s32ret)
{
```



```
        printf("send stream to adec fail\n");
        return s32ret;
    }

    /* get audio frame from adec */
    s32ret = HI_MPI_ADEC_GetData(AdChn, &stAudioFrameInfo);
    if (HI_SUCCESS != s32ret)
    {
        printf("adec get data err\n");
        return s32ret;
    }

    /* send audio frame to AO or others */

    /* release audio frame */
    s32ret = HI_MPI_ADEC_ReleaseData(AdChn, &stAudioFrameInfo);
    if (HI_SUCCESS != s32ret)
    {
        printf("adec release data err\n");
        return s32ret;
    }
}
```

3.8 音视频复合模块

3.8.1 概述

本节介绍音视频复合模块的运作机制及实际应用等。

运作机制：复合后的音视频数据以“同步分组”的形式提供。同步分组中包含同一个时间段内产生的音频、视频编码数据，其中视频数据至多只有一帧。同步分组的起始时间、时间跨度根据视频编码帧确定。

一个同步分组中，音频、视频编码数据的起始、结束时间并不要求严格对齐，允许存在一定的偏差。偏差的极限值为 2%AU，但通常状况下，偏差值小于 AU/2（AU 指一个音频编码单元的时间跨度，如 10ms）。

音视频复合模块主要提供以下功能接口：

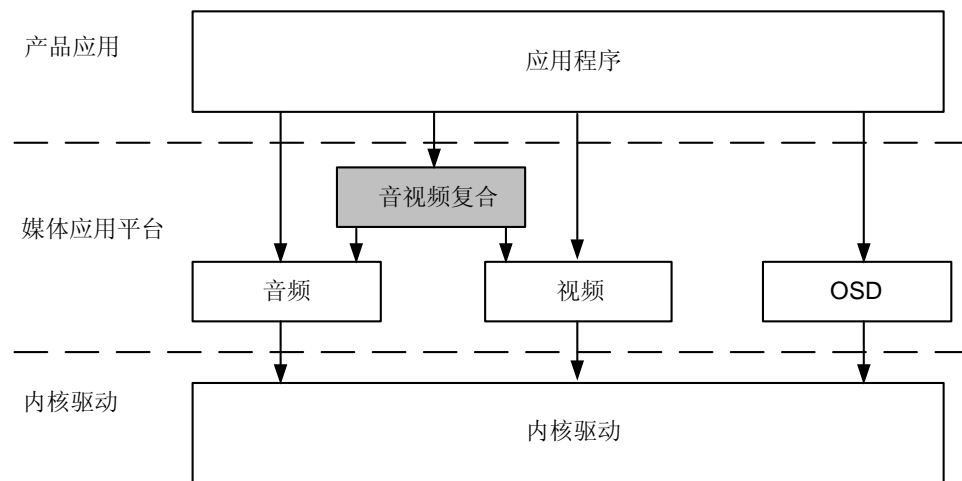
- HI_MPI_AVENC_CreatCH：创建音视频复合编码通道。
- HI_MPI_AVENC_DestroyCH：销毁音视频复合编码通道。
- HI_MPI_AVENC_StartCH：启动复合编码任务。
- HI_MPI_AVENC_StopCH：停止复合编码任务。
- HI_MPI_AVENC_GetStream：获取复合编码码流。
- HI_MPI_AVENC_ReleaseStream：释放获取的复合编码码流。

- HI_MPI_AVENC_RegisterFun: 注册用户自定义的音频编码。
- HI_MPI_AVENC_SetOverFlowLevel: 设置同步处理时音视频缓存的溢出水线。
- HI_MPI_AVENC_GetOverFlowLevel: 获取音视频缓存的溢出水线。
- HI_MPI_AVENC_SetUnitBufSize: 设置复合模块中音频缓存块的大小。

3.8.2 运作机制

音视频复合模块能够支持多路音视频复合处理，其中 1 路的运作机制如图 3-32 所示。

图3-32 音视频复合模块运作机制



音视频复合模块的接口调用步骤如下：

1. 音视频复合模块调用音频、视频模块的 MPI 接口获取音视频编码数据。
2. 如果同时有音频、视频编码数据则进行同步处理；如果只有音频或视频编码数据，无需同步处理，缓存后送出。
3. 调用音视频复合模块的 API 接口获取音视频数据。

----结束

3.8.3 开展业务和应用设计

介绍基于音视频复合模块开展业务和应用设计的方法，并列出发开发的典型业务，包括以下内容：

- 原则和策略：介绍如何基于音视频复合模块的资源开展设计。
- 供开发的典型业务：简介音视频复合模块可开发的典型业务。

3.8.3.1 原则和策略

基于音视频复合模块开展设计或业务的原则和策略如下：



音视频复合模块依赖于 VI、AI、AENC、VENC 模块，需要保证音频、视频编码通道创建成功后，再使用音视频复合模块。

3.8.3.2 供开发的典型业务

音视频复合模块提供获取复合的音视频数据业务和应用，包括使用 1 路音频和 1 路视频编码通道创建复合通道，并演示获取使用同步后的复合音视频数据、停止视频编码通道以及单独获取音频数据的操作方法（单独获取视频数据的操作与此类似）。

说明

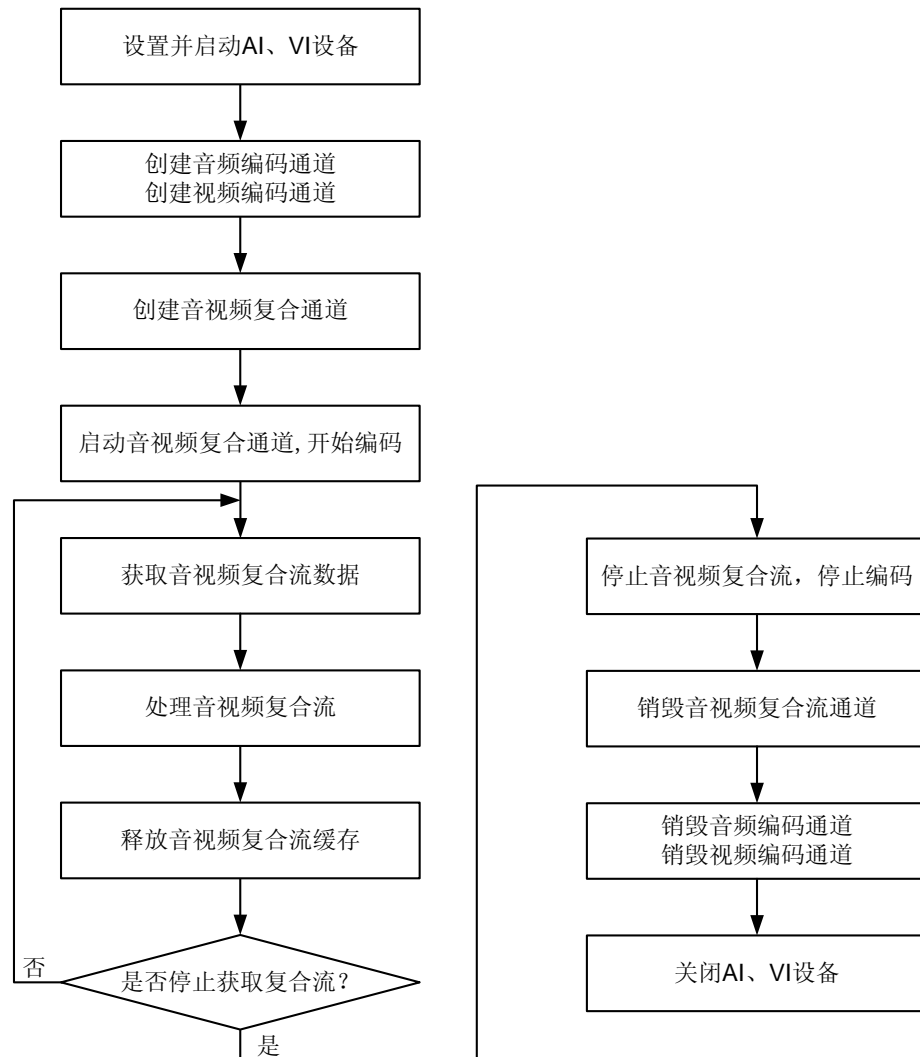
- 音视频复合模块仅以同步分组的形式给出同步后的复合音视频数据，后续的数据存盘或网络发送的格式可自定义。
- 如果在获取数据时始终不关心音频（或视频）数据，则在创建复合通道时，可以将音频（或视频）通道号设置为-1，表示只处理视频（或音频）数据。
- 支持同一路音频数据与多路视频数据复合同步，例如大小码流两个编码通道可以分别与同一个音频编码通道进行复合。

3.8.4 获取音视频复合数据

3.8.4.1 业务流程

获取音视频复合数据的业务流程如图 3-33 所示。

图3-33 获取音视频复合码流的业务流程



3.8.4.2 编程指导



注意

需要保证音频、视频编码通道创建成功后，再创建复合通道。

获取音视频复合流的步骤如下：

1. 设置并启动 AI、VI 设备。
2. 创建音频编码通道。
3. 创建视频编码通道，包括创建编码通道组、编码通道，注册编码通道以及绑定视频输入等。



4. 使用刚创建的音频和视频编码通道，创建音视频复合编码通道。
5. 启动音视频复合编码通道，开始音频和视频的编码任务。
6. 获取同步后的音视频复合数据。
7. 处理取得的音视频复合数据。
8. 复合数据使用完毕后，释放数据所占用的缓存。
9. 停止音视频复合编码通道。
10. 销毁音视频复合编码通道。
11. 销毁音频编码通道，包括包括反注册视频编码通道、销毁视频编码通道等，如果后续仍然需要使用此通道，则无需销毁。
12. 销毁视频编码通道，如果后续仍然需要使用此通道，则无需销毁。
13. 关闭 AI、VI 设备。

----结束

3.8.4.3 实例

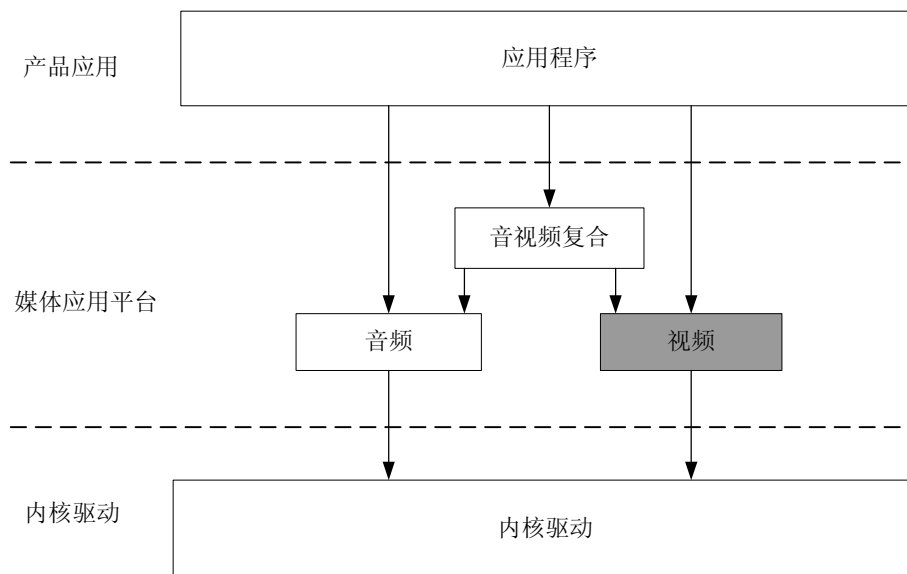
本实例演示使用音视频复合模块的完整流程、获取并使用同步后的复合音视频数据的方法，实例代码请参见 SDK 的根目录下的 `sample/sample_avenc.c`。



4 视频综合业务开发指引

视频业务运作机制如图 4-1 所示。

图4-1 视频业务运作机制



4.1 视频预览

4.1.1 1 路 D1 单通道预览业务

概述

本业务描述的场景如下：通过对 VI 模块、VO 模块、以及 AD 芯片的配置和组织调度，实现 1 路 D1 单通道预览。



相关模块

运行此业务需要以下模块：

- VI 模块
- VO 模块

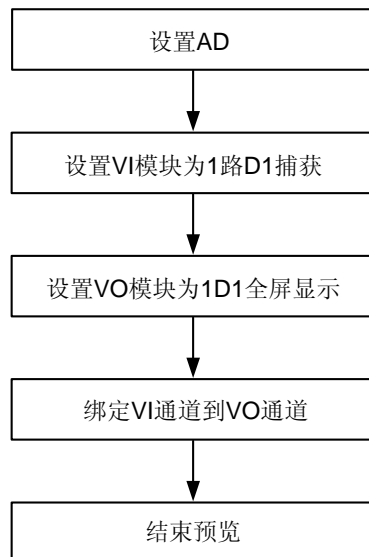
重要概念

无。

业务流程

1 路 D1 单通道预览的业务流程如图 4-2 所示。

图4-2 1 路 D1 单通道预览的业务流程



编程指导

1 路 D1 单通道预览的步骤如下：

1. 设置 AD。
2. 设置 VI 设备属性（工作模式需要与 AD 一致），并启用 VI 设备，设置 VI 通道属性（1 路 D1 大小），并启用 VI 通道（详细接口调用请参见“3.1 VI 模块”）。
3. 设置 VO 模块为 1D1 全屏显示，可参见图 3-2，详细的数据结构请参见《Hi3507 媒体处理软件 开发参考》。
4. 绑定 VI 通道和 VO 通道，VI 就开始将图像发送到绑定的 VO 通道进行预览显示。
5. 调用 VI 解绑定 VO 接口，解除通道的绑定。



----结束

实例

本实例实现 1 路 D1 各通道单画面预览，实例代码请参见 SDK 目录下的 \sample\vio\sample_vio.c。

4.1.2 多画面预览业务

概述

本业务描述的场景如下：通过对 VI 模块、VO 模块、以及 AD 芯片的配置和组织调度，实现多画面预览。

相关模块

运行此业务需要以下模块：

- VI 模块
- VO 模块

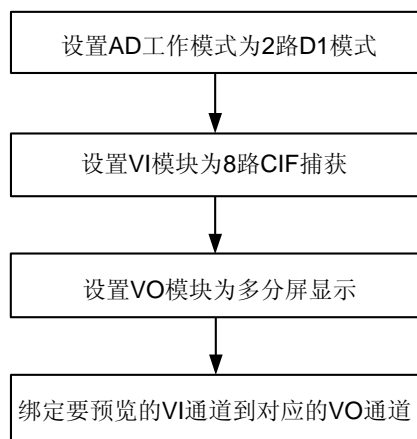
重要概念

无。

业务流程

多画面预览的业务流程如图 4-3 所示。

图4-3 多画面预览显示的业务流程





编程指导

多画面预览的步骤如下：

1. 设置 AD 工作模式为 2 路 D1 模式（这里假设外接 TW2815，每片工作的 AD 都要设置相应的配置）。
2. 配置 VI 模块为 8 路 CIF 捕获，可参见图 3-1，详细的数据结构请参见《Hi3507 媒体处理软件 开发参考》。
3. 配置 VO 模块为多画面显示。可配置为 1、4、6 或 9 多画面，具体可参见图 3-2，详细的数据结构请参见《Hi3507 媒体处理软件 开发参考》。
4. 绑定要预览的 VI 通道到对应的 VO 通道，可参见图 3-2，详细的数据结构请参见《Hi3507 媒体处理软件 开发参考》。

----结束

实例

本实例实现多画面预览，实例代码请参见 SDK 目录下的\sample\vio\sample_vio.c 文件。

4.2 视频编码

4.2.1 双码流业务

概述

本业务描述的场景如下：对同一视频源进行 1 路 D1 的 MJPEG 编码和 1 路 CIF 的 H.264 编码，并分别将码流保存到文件。同时启动该路视频源的预览。

运行该业务，能够从 VO 连接的输出设备上（如电视机）观看 VI 连接输入源（如摄像头）的视频，使用 H.264 播放器和 MJPEG 播放器可以分别播放该视频源的码流。

相关模块

运行此业务需要以下模块：

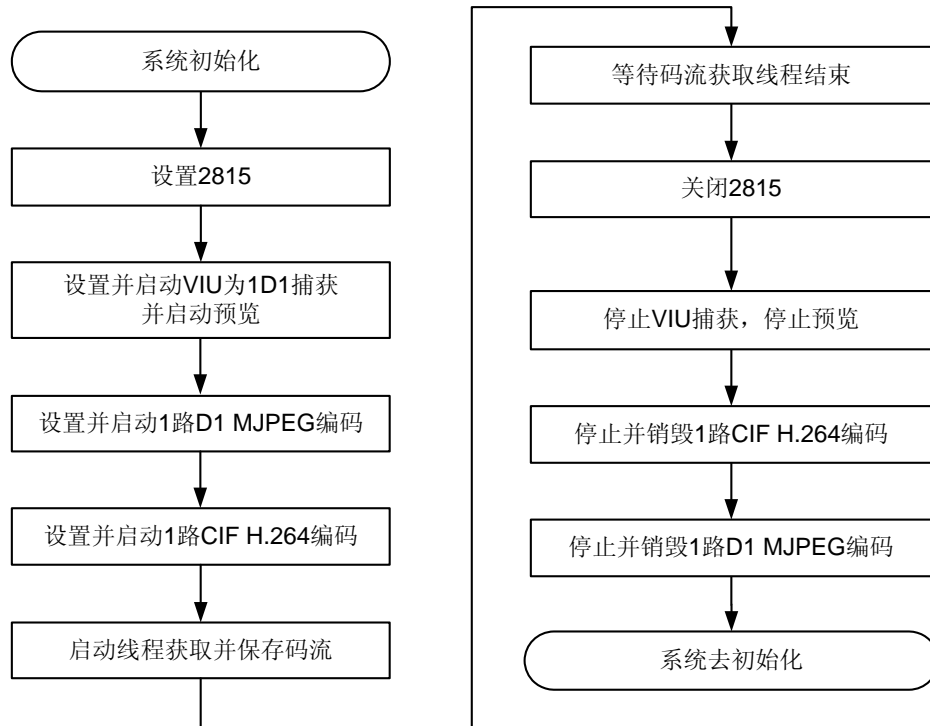
- VI 模块
- VENC 模块
- VO 模块

业务流程

双码流的业务流程如图 4-4 所示。



图4-4 双码流的业务流程



编程指导

双码流业务的步骤如下：

1. 系统初始化。
如果设备正在被使用，需要先禁用设备。
2. 设置 TW2815。
如果设备正在被使用，需要先禁用设备。
3. 设置并启动 VIU 为 1D1 捕获，并启动预览。
配置 VI 请参见“3.1 VI 模块”。本业务使用同一个视频源启动 1D1 编码和 1CIF 编码，所以需要配置 VI 为 D1 大小，否则 D1 大小的编码通道无法编码（如果编码通道大小大于视频输入图像大小，则编码通道可以启动，但无法获取到编码码流）。
4. 启用 1D1 的 MJPEG 编码。
启动编码请参见“3.3.3 配置并启动视频编码”。
5. 启用 1CIF 的 H.264 编码。
6. 创建获取 H.264 码流和 MJPEG 码流的线程。



本业务可使用多线程的方式来获取编码码流，也可选择使用 `select` 方式来在一个线程中获取多路码流，具体内容请参见“[3.3.9 获取多路编码码流](#)”。

7. 等待获取码流的线程结束。
8. 关闭 TW2815。
9. 停止 VIU 捕获，停止预览。停止 VIU 请参见“[3.1 VI 模块](#)”。
10. 停止并销毁 MJPEG 编码。销毁编码请参见“[3.3.4 停止并销毁视频编码](#)”。
11. 停止并销毁 H.264 编码。
12. 系统去初始化。

----结束

实例

本实例实现 1 路 D1 的 MJPEG 和 1 路 CIF 的 H264 编码，并启动预览，实例代码请参见 SDK 的根目录下的 `\sample\venc\sample_venc.c`。

4.2.2 抓拍业务

概述

本业务描述的场景如下：使用同一个视频源，编码 1 路 CIF 的 H.264 码流的并存文件，并同时 D1 抓拍和启动对该视频源的预览。

运行该业务，能够从 VO 连接的输出设备（如电视机）观看 VI 连接输入源（如摄像头）的视频，使用 H.264 播放器可以播放 H.264 码流并能查看抓拍的图片。

相关模块

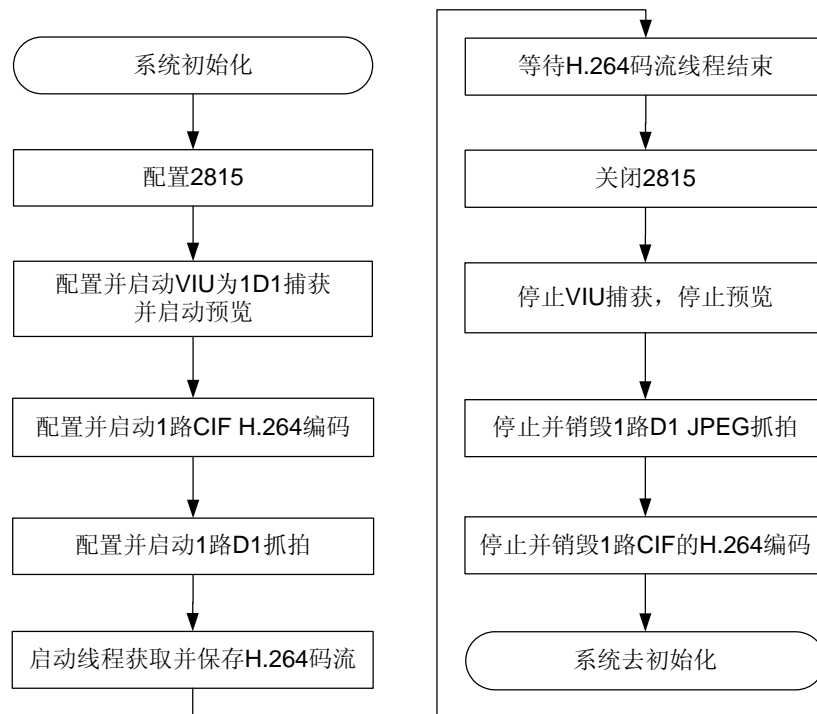
运行此业务需要以下模块：

- [VI 模块](#)
- [VENC 模块](#)
- [VO 模块](#)

业务流程

抓拍的业务流程如[图 4-5](#)所示。

图4-5 抓拍的业务流程



编程指导

抓拍的步骤如下：

1. 系统初始化。
2. 配置 TW2815。
3. 配置并启动 VIU 为 1D1 捕获，并启动预览。
4. 启用 1CIF 的 H.264 编码。请参见“[3.3.3 配置并启动视频编码](#)”。
5. 启用 1D1 的 JPEG 抓拍。请参见“[3.3.10 按轮循方式抓拍](#)”和“[3.3.11 按并发方式抓拍](#)”。
6. 启动线程获取并保存 H.264 码流。

此业务使用多线程的方式来获取编码码流，也可以选择使用 select 方式来在一个线程中获取多路码流，请参见“[3.3.9 获取多路编码码流](#)”。

7. 等待获取 H.264 码流的线程结束。
8. 关闭 TW2815。
9. 停止 VIU 捕获，停止预览。请参见“[3.1 VI 模块](#)”。
10. 停止并销毁 JPEG 抓拍。请参见“[3.3.4 停止并销毁视频编码](#)”。
11. 停止并销毁 H.264 编码。



12. 系统去初始化。

----结束

实例

本实例演示了如何实现 1 路 CIF 的 H.264 编码和 1 路 D1 的 JPEG 抓拍，并启动预览，实例代码请参见 SDK 的根目录下的\sample\venc\sample_venc.c。

4.2.3 请求 I 帧业务

概述

本业务描述的场景如下：1 路 D1 的 H.264 编码每隔 1s 请求 I 帧。

相关模块

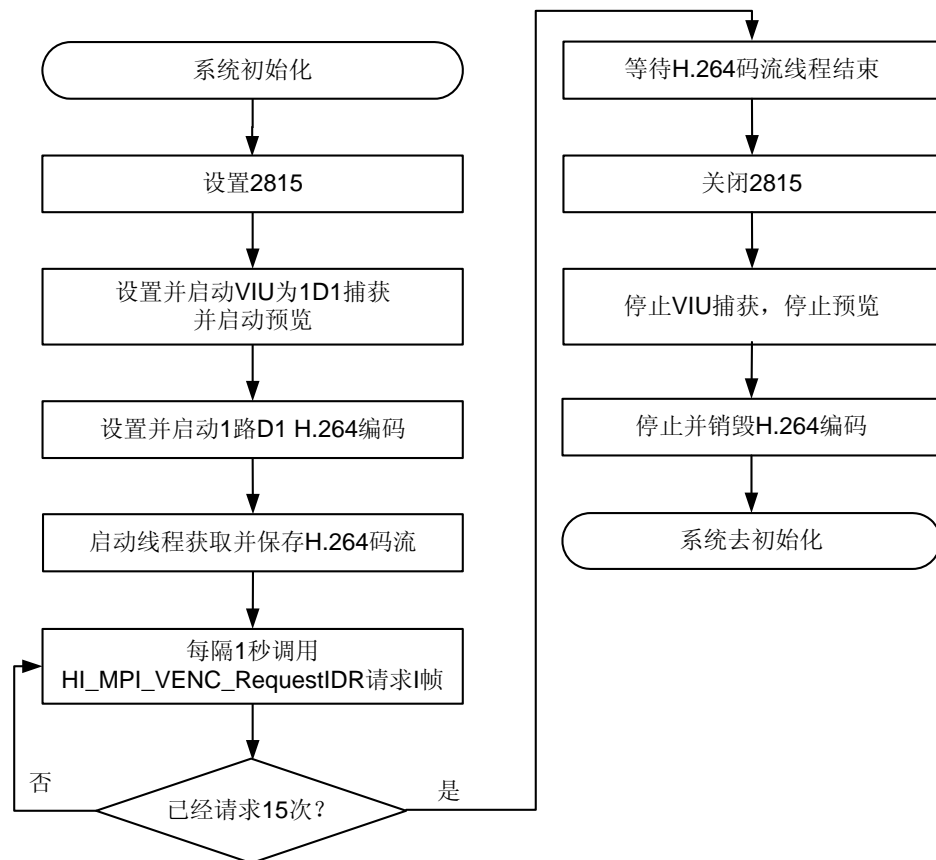
运行此业务需要以下模块：

- VI 模块
- VENC 模块
- VO 模块

业务流程

请求 I 帧的业务流程如图 4-6 所示。

图4-6 请求 I 帧的业务流程



编程指导

请求 I 帧的步骤如下：

1. 系统初始化。
2. 设置 TW2815。
3. 设置并启动 VIU 为 1D1 捕获，并启动预览。
4. 启用 1D1 的 H.264 编码。请参见“3.3.3 配置并启动视频编码”。
5. 启动线程获取并保存 H.264 码流。请参见“3.3.5 按帧获取编码码流”。
6. 每隔一秒种调用 HI_MPI_VENC_RequestIDR 来请求 I 帧。

通道创建之后才可以调用 HI_MPI_VENC_RequestIDR 来请求 IDR 帧，编码器将在最快的可能时间里上报 IDR 帧。

7. 等待获取 H.264 码流的线程结束。
8. 关闭 TW2815。
9. 停止 VIU 捕获，停止预览。请参见“3.1 VI 模块”。



10. 停止并销毁 H.264 编码。
11. 系统去初始化。

----结束

实例

本实例演示了如何在 1 路 D1 的 H264 编码时请求 IDR 帧，实例代码请参见 SDK 的根目录下的\sample\sample_venc.c 中 SAMPLE_1D1H264_RequestIDR 函数。

4.2.4 插入用户数据业务

概述

本业务描述的场景如下：在编码 1 路 D1 的 H.264 码流时每隔一定时间插入用户数据。

相关模块

运行此业务需要以下模块：

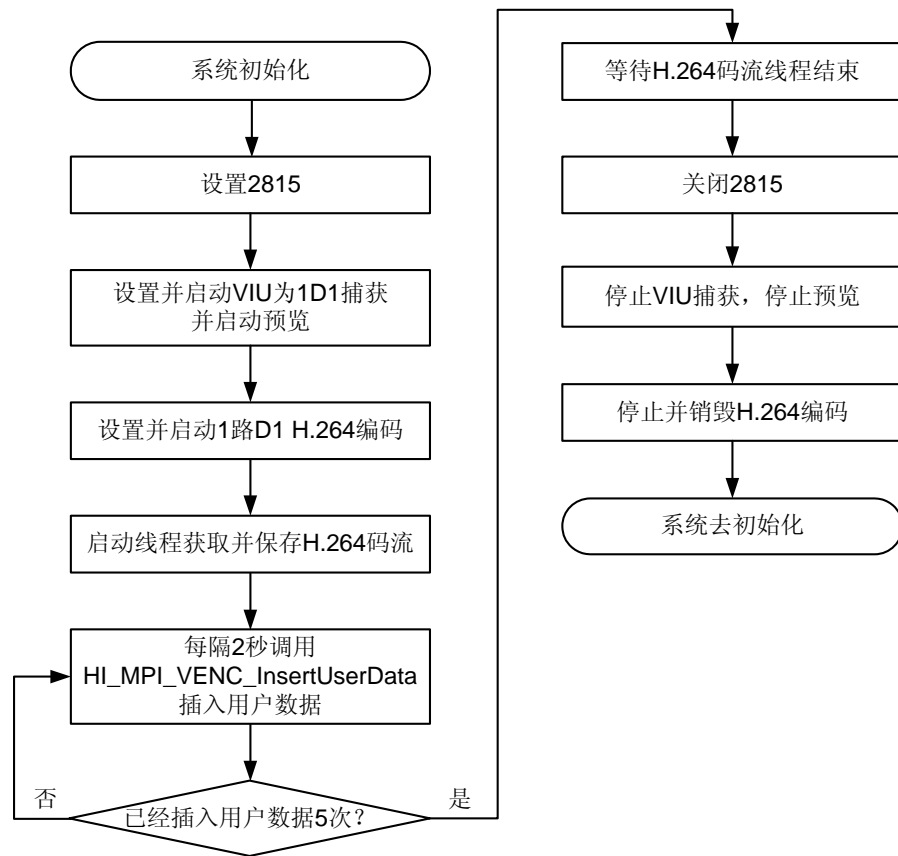
- VI 模块
- VENC 模块
- VO 模块

业务流程

插入用户数据的业务流程如图 4-7 所示。



图4-7 插入用户数据的业务流程



编程指导

插入用户数据的步骤如下：

1. 系统初始化。
2. 设置 TW2815。
3. 设置并启动 VIU 为 1D1 捕获，并启动预览。
4. 启用 1D1 的 H.264 编码。请参见“3.3.3 配置并启动视频编码”。
5. 创建获取 H.264 码流的线程。请参见“3.3.5 按帧获取编码码流”。
6. 每隔 2 秒种调用 HI_MPI_VENC_InsertUserData 来插入用户数据。

通道创建之后才可以调用 HI_MPI_VENC_InsertUserData 来插入用户数据，编码器将在下一帧编码的时候将插入的字符串编码成 SEI 包括入码流中。

7. 等待获取 H.264 码流的线程结束。
8. 关闭 TW2815。
9. 停止 VIU 捕获，停止预览。请参见“3.1 VI 模块”。



10. 停止并销毁 H.264 编码。
11. 系统去初始化。

----结束

实例

本实例演示了如何在 1 路 D1 的 H264 编码时候时插入用户数据，实例代码请参见 SDK 的根目录下的\sample\venc\sample_venc.c 中 SAMPLE_1D1H264_InsertUserData 函数。

4.3 视频解码

概述

本业务描述的场景如下：建立一个视频解码通道，读取本地的 H.264 码流文件，发送码流包到该视频解码通道，启动对码流文件的解码，将解码后的图像（图像大小 D1）发送到 VO 显示。

运行该业务能够从 VO 连接的输出设备（如电视机）上看到本地码流文件解码后的视频。

相关模块

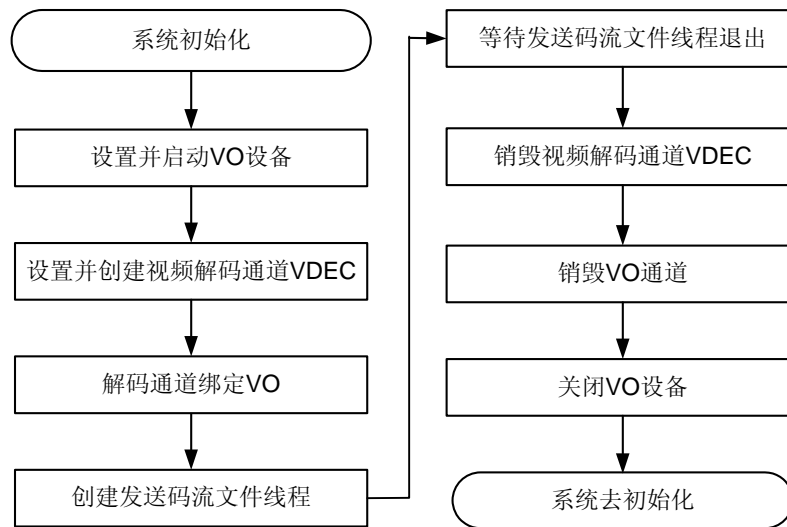
运行此业务需要以下模块：

- VDEC 模块
- VO 模块

业务流程

H.264 单通道解码回放的业务流程如图 4-8 所示。

图4-8 H.264 单通道解码回放业务流程



编程指导

H.264 单通道解码回放的步骤如下：

1. 系统初始化。
2. 设置并启动 VO。
3. 创建一个 H.264 协议的视频解码通道，并启动数据接收。
4. 解码通道绑定 VO。
5. 创建读取本地码流文件，发送码流包到视频解码通道启动解码的线程。
6. 等待发送码流线程的退出。
7. 销毁视频解码通道。
8. 关闭 VO 通道。
9. 关闭 VO 设备。
10. 系统去初始化。

----结束

实例

本实例实现 1 路 D1 的 H.264 解码，并启动 VO 显示解码后图像，实例代码请参见 SDK 的根目录下的 `\sample\vdec\sample_vdec.c`。 `sample_vdec.c` 中包括了 5 个场景用例，第 1 个场景是 1 路 D1 解码，后面 4 个场景依次是数据流用户态控制演示、低帧率解码/输出演示、慢速播放演示和快速播放演示。



4.4 运动侦测

概述

本业务描述的场景如下：1 路 D1 大小的 VI 到 VO 预览，1 路 CIF H.264 编码，开启移动侦测，获取移动侦测中的宏块 SAD 值并输出到终端。

相关模块

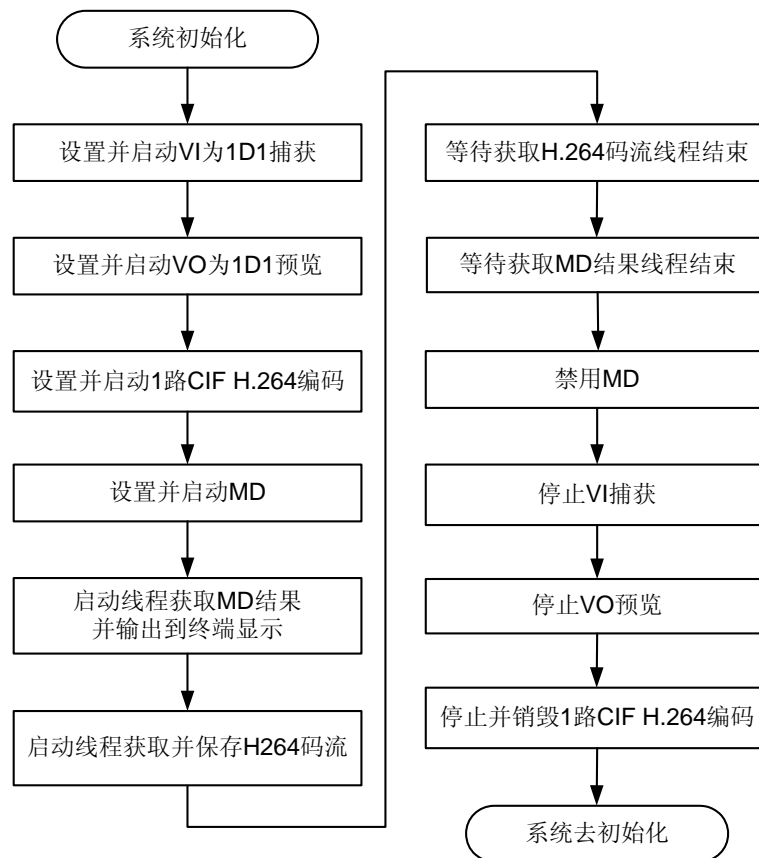
运行此业务需要以下模块：

- VI 模块
- VO 模块
- VENC 模块
- MD 模块

业务流程

移动侦测的业务流程如图 4-9 所示。

图4-9 移动侦测的业务流程





编程指导

移动侦测的步骤如下：

1. 系统初始化。
2. 设置并启动 VI。
3. 设置并启动 VO 为预览。
4. 设置并启动 H.264 编码。
5. 设置启动 MD。
6. 开启获取 MD 结果数据线程，获取 MD 结果并输出到终端显示。
7. 开启获取码流数据线程，获取码流并存文件。
8. 等待获取码流数据线程结束。
9. 等待获取 MD 结果线程结束。
10. 禁用 MD。
11. 停止 VI 捕获。
12. 停止 VO 预览。
13. 销毁编码通道。
14. 系统去初始化。

----结束

实例

本实例实现 VI 送 VO 预览、开启 H.264 编码、开启移动侦测，实例代码请参见 SDK 的根目录下的 .\sample\md\sample_md.c 下的 HI_S32 SAMPLE_Md_GetData(HI_VOID) 函数。

4.5 视频遮挡

概述

本业务描述的场景如下：1 路 D1 大小的 VI 到 VO 预览，开启该 VI 通道视频遮挡区域（4 个区域依次改变颜色、位置、大小、层次）。

相关模块

运行此业务需要以下模块：

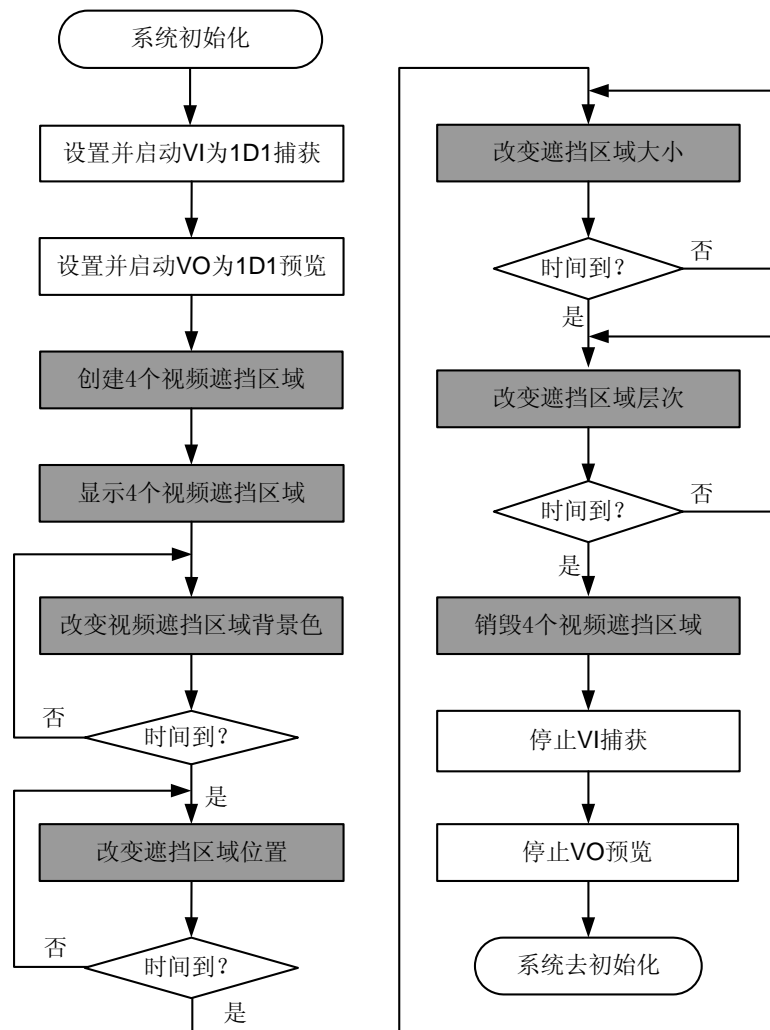
- VI 模块
- VO 模块

• VPP 模块

业务流程

单通道视频遮挡区域的业务流程如图 4-10 所示。

图4-10 单通道视频遮挡区域的业务流程



编程指导

单通道视频遮挡区域的步骤如下：

1. 系统初始化。
2. 设置并启动 VI。
3. 设置并启动 VO 预览。
4. 创建遮挡区域。



5. 显示遮挡区域。
6. 改变遮挡区域的背景色。
7. 改变遮挡区域的位置。
8. 改变遮挡区域的大小。
9. 改变遮挡区域的层次。
10. 销毁遮挡区域。
11. 停止 VI 捕获。
12. 停止 VO 预览。
13. 系统去初始化。

----结束

实例

本实例实现 VI 送 VO 预览、开启遮挡区域，实例代码请参见 SDK 的根目录下的 .\sample\vpp\sample_vpp.c 下的 HI_S32 SAMPLE_1D1_4CoverRegion()函数。

4.6 OSD 区域

概述

本业务描述场景如下：1 路 D1 大小的 VI 到 VO 预览、开启 H.264 编码并存文件、开启该编码通道 OSD 区域（4 个区域，内容为图片或文字，依次改变区域的位置以及透明度）。

相关模块

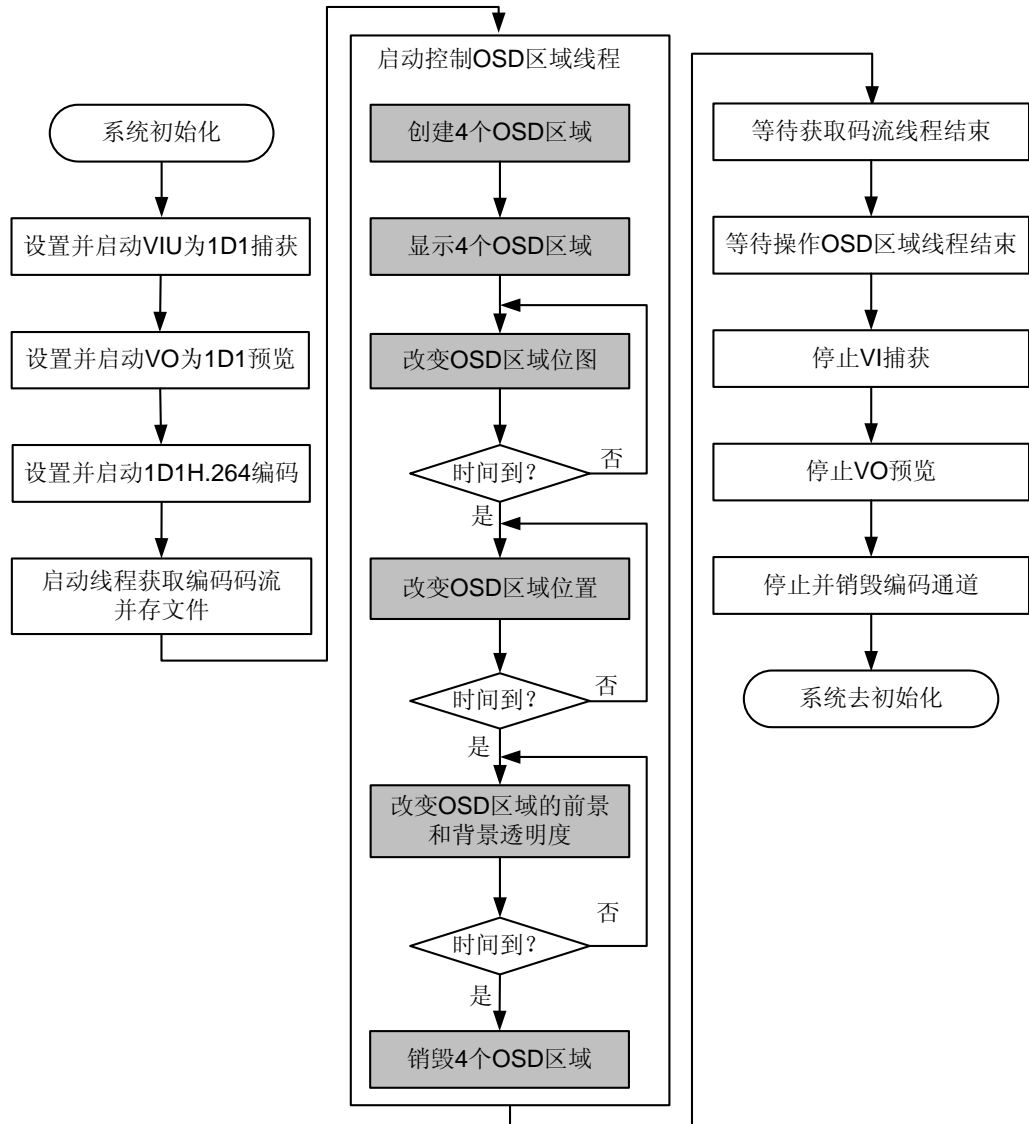
运行此业务需要以下模块：

- VI 模块
- VO 模块
- VENC 模块
- VPP 模块

业务流程

单通道 H.264 编码并开启 OSD 的业务流程如图 4-11 所示。

图4-11 单通道 H.264 编码并开启 OSD 的业务流程



编程指导

单通道 H.264 编码并开启 OSD 的步骤如下：

1. 系统初始化。
2. 设置启动 VI。
3. 设置启动 VO，并启动预览。
4. 设置启动编码。
5. 启动线程获取码流。
6. 启动线程控制 OSD 区域。



7. 等待获取码流线程结束。
8. 等待控制 OSD 区域线程结束。
9. 停止 VI 捕获。
10. 停止 VO 预览。
11. 停止并销毁编码通道。
12. 系统去初始化。

----结束

实例

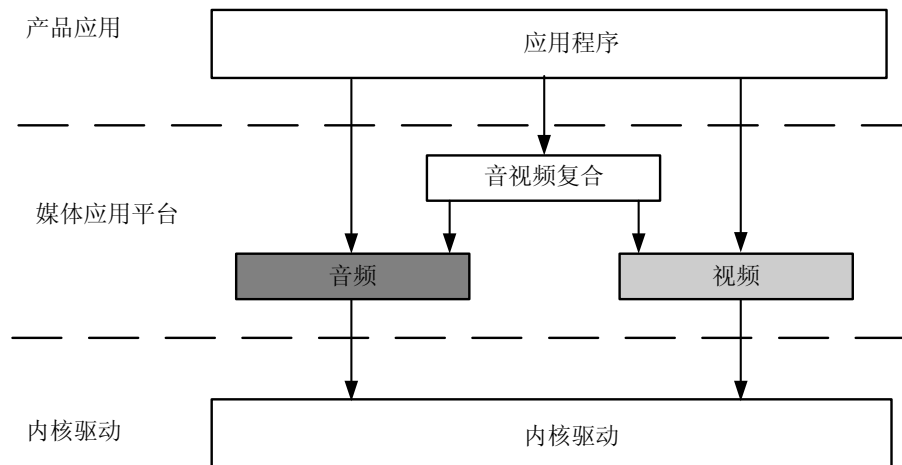
本实例实现 VI 送 VO 预览、开启编码、开启 OSD 区域，实例代码请参见 SDK 的根目录下的 \sample\vpp\sample_vpp.c 下的 HI_S32 SAMPLE_1D1_4OverlayRegion(HI_VOID) 函数。



5 音频综合业务开发指引

音频模块的运作机制如图 5-1 所示。

图5-1 音频模块运作机制



5.1 音频采集及编码业务

概述

本业务描述的场景如下：从 Hi3507 的 AI 设备采集音频帧、送音频编码通道编码并获取码流的音频业务。

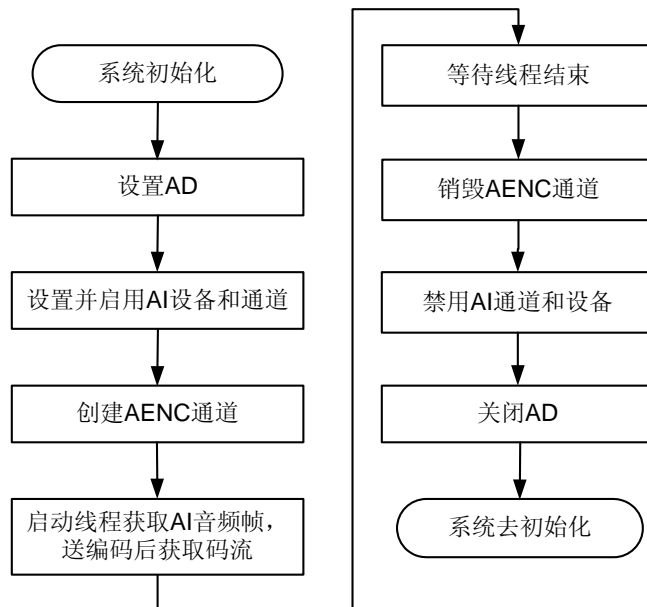
相关模块

运行此业务需要 AI、AENC 模块。

业务流程

音频采集及编码的业务流程如图 5-2 所示。

图5-2 音频采集及编码的业务流程



编程指导

音频采集及编码的步骤如下：

1. 系统初始化。
2. 设置 AD（TW2815）。
音频帧的采集需要 AD 芯片驱动的配合使用。
3. 设置并启用 AI 设备，启用 AI 通道
先配置 AI 设备属性再启用 AI 设备和 AI 通道，AI 通道即可开始工作（可参见“[3.7 AUDIO 模块](#)”中 AI 子模块的详细说明）。
4. 创建 AENC 通道。
创建音频编码通道，并设置相应的编码协议的编码通道属性（可参见“[3.7 AUDIO 模块](#)” AENC 子模块的详细说明）。
5. 启动线程获取音频帧，送编码后获取码流。

从指定的 AI 设备中的 AI 通道获取音频帧数据，将音频帧发送到 AENC 通道，然后从 AENC 通道获取编码后码流，使用完码流后需要归还码流缓存。

6. 等待线程结束。
7. 销毁 AENC 通道。
8. 禁用 AI 设备。
9. 关闭 AD。



10. 系统去初始化。

----结束



说明

- 配置并启用 AI 设备和通道的详细说明请参见“[3.7.3 启用 AI 设备并获取音频帧](#)”。
- 创建 AENC 通道并获取码流的详细说明请参见“[3.7.5 创建 AENC 通道和发送音频帧编码并获取码流](#)”。
- 关于数据结构的说明以及具体接口的使用注意事项请参见《Hi3507 媒体处理软件 开发参考》

实例

本实例实现获取一路音频帧数据进行 ADPCM 格式的音频编码并获取编码后码流的基本业务。实例代码请参见 SDK 的根目录下的 `sample/audio/sample_audio.c` 文件中 `SAMPLE_Audio_Aenc` 函数示例。

5.2 音频回声抵消业务

概述

本业务描述的场景如下：在某些场景（最典型的是通话场景下），本地麦克在采集语音时采集到扬声器的声音（即采集到远端语音信号），经过网络发送到远端后，导致远端可以听到自己的话声。

回声抵消指可以在获取近端语音输入数据的同时获取远端语音数据，然后就可以将这两种输入数据进行回声抵消并编码；此功能原则上不限制使用的通道个数，出于性能要求，设计性能为只支持一路回声抵消。

相关模块

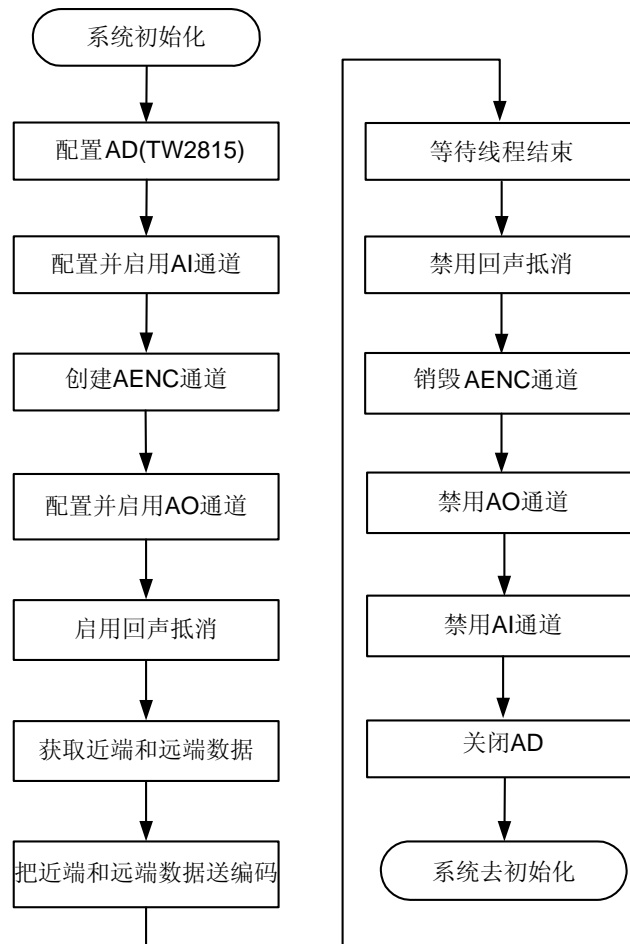
运行此业务需要以下模块：

- AI
- AENC
- AO

业务流程

音频回声抵消的业务流程如图 5-3 所示。

图5-3 音频回声抵消的业务流程



编程指导

音频回声抵消的步骤如下：

1. 设置 AD（TW2815）。

音频帧的采集需要 AD 芯片驱动的配合使用。

2. 设置并启用 AI 设备，启用 AI 通道

先配置 AI 设备属性再启用 AI 设备和 AI 通道，AI 通道即可开始工作（可参见“[3.7 AUDIO 模块](#)”中 AI 子模块的详细说明）。

3. 创建 AENC 通道。

创建音频编码通道，并设置相应的编码协议的编码通道属性（可参见“[3.7 AUDIO 模块](#)” AENC 子模块的详细说明）。

4. 配置并启用 AO 通道。

5. 启用回声抵消。



6. 启动线程获取近端和远端音频帧，发送编码后获取的码流（已作了回声抵消的码流）。

从指定的 AI 设备中的 AI 通道获取音频帧数据，将音频帧发送到 AENC 通道，然后从 AENC 通道获取编码后码流，使用完码流后需要归还码流缓存。

7. 禁用回声抵消功能。
8. 销毁编码通道。
9. 禁用 AI、AO 通道。

----结束



说明

- 配置并启用 AI 设备的详细说明请参考音频模块的相关章节。
- 创建 AENC 通道并获取码流的详细说明请参考音频模块的相关章节。
- 关于数据结构的说明以及具体接口的使用注意事项请查阅《Hi3507 媒体处理软件 开发参考》

实例

无。

5.3 音频解码及输出业务

概述

本业务描述的场景如下：将音频码流数据解码并发送到 AO 设备输出。

相关模块

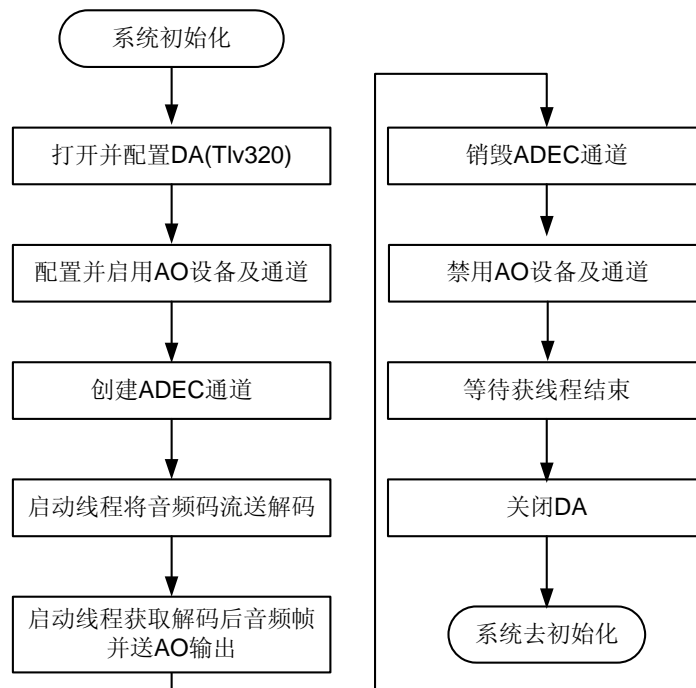
运行此业务需要以下模块：

- ADEC
- AO

业务流程

音频解码及输出的业务流程如图 5-4 所示。

图5-4 音频解码及输出的业务流程



编程指导

音频解码及输出的步骤如下：

1. 打开并配置 DA(Tlv320)。

音频帧的输出需要 DA 芯片驱动的配合使用。

2. 设置并启用 AO 设备，启用 AO 通道

先配置 AO 设备属性再启用 AO 设备和 AO 通道，AO 通道即可开始工作（可参见“[3.7 AUDIO 模块](#)”中 AO 子模块的详细说明）。

3. 创建 ADEC 通道。

创建音频解码通道，并设置相应解码协议的解码通道属性。

4. 启动线程，发送音频码流解码、获取解码后音频帧并送 AO 设备输出。

创建线程，执行如下操作：将音频码流数据（来自文件或网络）发送到解码通道进行解码，获取解码后音频帧数据，将解码后数据发送到 AO 设备的指定通道进行输出，然后归还解码后数据缓存。

----结束

说明

- 创建音频解码通道时，可以指定码流发送方式为流式发送或按帧发送，推荐按帧发送方式。



- 配置并启用 AO 设备的详细说明请参考音频模块的相关章节。
- 创建 ADEC 通道并送解码及获取解码后数据的详细说明请参考音频模块的相关章节。
- 关于数据结构的说明以及具体接口的使用注意事项请查阅《Hi3507 媒体处理软件 开发参考》。

实例

本实例实现发送一路 ADPCM 格式的音频码流，音频解码后再将解码数据送 AO 设备的指定通道，完成音频输出的基本业务。实例代码请参见 SDK 的根目录下的 sample\audio\sample_audio.c 中的 SAMPLE_Audio_Adec 函数示例。



6 FAQ

6.1 如何获取当前 MPP 平台的版本号

【现象】 无。

【分析】 无。

【解决】

- 用户定义一个 MPP 版本数据结构变量 `sVerMpp`，`MPP_VERSION_S` 数据结构请参见 `hi_common.h` 中的具体描述。
- 包含头文件 `hi_common.h`、`hi_comm_sys.h`、`mpi_sys.h`。
- 调用函数 `HI_MPI_SYS_GetVersion`，输入该 MPP 版本数据结构变量指针 `&sVerMpp`，将返回表示当前 MPP 版本的字符串。
- 使用函数 `printf("%s \n", sVerMpp.aVersion)`；可以看到以下类似例子的内容：
`HI_VERSION=Hi3511 MPP V1.0.3.0`

6.2 视频编码

NTSC 制 QCIF 小码流图像被裁剪问题

【现象】

N 制 CIF 大码流（352%240）、QCIF 小码流（176%112），编码后小码流图像底部 8 像素被裁剪。

【分析】

H.264 编码图像宽高需要 16 像素对齐，对 CIF 大码流做 1/2 缩放后，由于高度不能 16 像素对齐，必须裁剪 8 像素。

【解决】

目前解决方案为视频输入源仍然为 352%240，但创建编码通道时，大小码流分别多编码 16 像素和 8 像素（即大码流创建为 352%256，小码流创建为 176%128），这样大码流 1/2 缩放后即可得到完整的 QCIF 小码流图像，同时依据 H.264 的 Crop 语法，设置



显示区域为 (352%240) 和 (176%120); 用户 PC 端解码时, 需要支持 Crop 语法, 否则码流图像底部会有多余无效数据。

N 制 QQVGA 小码流时, 基于同样原理, 也可以采用上述方案, 即 QVGA 大码流创建为 320%256, QQVGA 小码流创建为 160%128。

H.264 码率波动问题

【现象】

CBR 模式下, 码率波动较大, 难控制。

【分析】

H.264 编码提供了灵活的 CBR 码率控制策略: 支持完全由 SDK 来控制码率; 也支持用户通过设置码率波动范围的方式来控制码率。

CBR 模式下, 可通过 PicLevel 的不同取值来选择控制策略:

- 0: 完全由 SDK 来控制码率。码率波动范围基本为[-30%, 30%]。
- 1~5: 对应的码率波动范围分别为 $\pm 10\%$ ~ $\pm 50\%$ 。在场景切换或大运动时码率会上冲; 波动范围越大, 图像主观质量越好。

【解决】

可以考虑下列解决办法:

- 降低码率波动范围, 即减小 PicLevel;
- 由 SDK 控制码率, 即令 PicLevel 等于 0, 推荐使用。

6.3 音频

Hi3507 音频有那些软件调试手段

【现象】无。

【分析】无。

【解决】

- 目前主要可以通过查看 proc 信息和 DAM Buffer 信息来了解 AI、AO 的相关运行状态以及数据状态。
- 使用 `cat /proc/umap/ai` 或 `cat /proc/umap/ao`, 具体信息请参见“[7 Proc 调试信息说明](#)”:
 - Attribution of AI Device 区可以查看此 AI 设备的属性。
 - Status of AI Device 区下的 IntCnt 表示 SIO 采集数据的 DMA 中断次数, 正常情况下这个值应该持续增加, 否则表示 SIO 未正确采集到音频数据。
 - Status of AI Channel 区下的 intlost 表示通道数据 Buffer 满的次数, 一般可以表明通道丢帧次数。



- 通过 himd 工具查看 DMA 音频 Buffer 数据，物理地址可以取/proc/umap/ai 信息中得到的 DMAPhy0 或 DMAPhy1 的值，DMA buffer 中存放 SIO 一次采样的数据（64bit），按照通道排列顺序即看观察各个 AI 通道的数据，工作正常的 AI 通道的数据应该会有数据上明显的变化。

如何从文件中读音频码流按帧方式解码

【现象】

音频解码需要创建两个线程，一个线程读文件发送码流到解码通道，另一个线程从解码通道中获取音频帧数据送 AO；如果采用一个线程的话无法获取音频帧。

【分析】



注意

目前 AAC 只支持流式解码。

Hi3507 音频解码支持流式和按帧发送方式（即 pack 方式和 stream 方式），如果按流式发送，由于软件内部需要缓存并解析码流，因此不能做到发送一段码流就能获取到音频帧，因此用户的应用层需要创建两个线程；而如果用按帧发送方式，则不存在此问题，且效率会相对较高。

【解决】

推荐使用按帧发送方式送音频解码。如果不知道音频码流的帧边界，可以参照海思语音帧头结构读取每个码流帧的长度，以确定帧边界。相关代码请参考 Sample_audio.c。

如何使用立体声音频

【现象】

用户业务需要使用立体声，需要做哪些配置，左右声道通道是如何分配的？

【分析】无。

【解决】

Hi3507 立体声数据实际上是通过各自取 SIO 中左右通道相应位置的单声道数据实现，并取左声道的通道号作为立体声的通道号。例如 8bit 的情况下共可以容纳 4 路立体声 0~3 对应的单声道通道号分别是{0,4}、{1,5}、{2,6}、{3,7}，而在 16bit 情况下最多容纳 2 路立体声 0~1 分别对应单声道通道号为{0,2}和{1,3}。使用立体声需要将 AI/AO 的设备属性中 enSoundmode 设置为 AUDIO_SOUND_MODE_STEREO。

ADPCM（IMA）音频编码时，为什么声音有异常

【现象】

使用 ADPCM（IMA）音频编码，输出后发现杂音。



【分析】

ADPCM 音频编码协议有两种类型：DVI4 和 IMA。使用 IMA 类型时，输入的音频帧的采样点数（帧长）应该为 81、161、241、321、481，即比 DVI4 类型时多一个采样点，否则编码后的音频码流会有异常。

【解决】

将 AI 的采样点数设置为 81、161、241、321 或 481。

业务繁忙时音频偶尔有少许杂音

【现象】

业务繁忙时音频偶尔有少许杂音。

【分析】

由于音频编解码目前是放在用户态处理，当系统负荷较重时，编码或者获取码流的过程可能会被内核态的处理所打断，导致不能及时从 AI 获取音频帧数据，AI 的音频帧缓存满之后就会丢弃音频帧，因此出现偶尔的杂音。

【解决】



说明

如果系统负荷严重超出，任何手段都无效。

可以通过以下手段避免以上问题的出现：

- 将 AI/AO 通道的缓存帧个数设置得更大，例如 20 帧，以平滑更多的系统颠簸。
- 将音频帧的帧长设置得更大，例如 320，以减少音频的中断数目。
- 将获取音频码流的线程优先级设置得更高，例如使用 nice(-1)。

如何使用音频 PCM 模式

【现象】

音频使用 PCM 时序模式，软硬件应该如何配置。

【分析】

无。

【解决】

- 支持 PCM 标准时序模式和自定义时序模式，SIO 工作在从模式，时钟上升沿发送数据。
- 硬件：
只支持 SIO 从模式，以 Hi3507 Demo 板为例，需要增加外围晶振（R1838 nc 33 Ω 电阻）提供 codec 的时钟，同时断开 ACKOUT（R1912 电阻）。
- 软件接口：
加载 hidmac.ko 时，增加模块参数配置 SIO 的工作模式，1 为 PCM 模式，0 为 I²S 模式（默认），例如 SIO0 和 SIO1 都使用 PCM 模式则加参数 sio0_mode=1 和 sio1_mode=1。



- 将音频外围 codec 配置为 DSP 主模式（SDK 的 tlv320aic31 驱动中提供了样例接口，参考 sample_audio.c 中使用）
- 使用 HI_MPI_AI_SetPubAttr 接口，设置为 PCM 工作模式（存在 AIO_MODE_PCM_SLAVE_STD 和 AIO_MODE_PCM_SLAVE_NSTD2 种模式），其他接口使用方法不变；支持两个 16bit 单声道或一个立体声。
- 时序对接方面 SDK 提供以上 2 种 PCM 模式（标准和自定义时序），具体时序说明请参见《Hi3507 H.264 编解码处理器 用户指南》中音频接口部分；SDK 中将发送时钟沿固定配置为使用上升沿，如有必要请根据具体对接 codec 时序情况自行更改此寄存器 PCM_CT_SET bit[14]。

6.4 视频解码

H.264 解码图像输出问题

【现象】

用户发送码流给解码器，解码图像输出存在较长时间的延迟。

【分析】

由于 H.264 协议自身的特点，解码完成到图像输出存在较大的延迟。为解决该问题，解码器提供了三种输出方式：

- 普通输出：完全按照 264 协议输出图像。
- 快速输出：收到新一帧后，就输出上一帧。
- 按帧输出：用户发送一帧，输出一帧。

以上三种方式，自上而下，输出速度越来越快。

因快速输出和按帧输出不符合协议，因此需用户正确配置和使用解码器，方可实现。

【解决】



注意

- 设置为按帧发送方式时，用户必须保证每次发送完整一帧码流（即调用一次 HI_MPI_VDEC_SendStream 接口，必须发送完整一帧码流），否则会出现解码错误。
- RefFrameNum 和 enMode 都是静态参数，在创建通道时指定，不可以动态修改。
- 当码流本身不支持快速输出时，会出现图像显示乱序的现象。
- 当码流有错误时，并不会按帧输出，此时需要送入新码流才能输出上一帧。

创建解码通道时，通过设置不同的通道属性参数选择输出方式，具体对应如表 6-1 所示。



表6-1 解码输出方式参数配置

输出方式	通道属性		注意事项
	RefFrameNum	enMode	
普通输出	>2 (普通输出)	H264D_MODE_STREAM (流式发送)	-
	>2 (普通输出)	H264D_MODE_FRAME (按帧发送)	用户保证按帧发送
快速输出	2 (快速输出)	H264D_MODE_STREAM (流式发送)	-
按帧输出	2 (快速输出)	H264D_MODE_FRAME (按帧发送)	用户保证按帧发送

6.5 如何配置 MPP 视频缓存池

【现象】

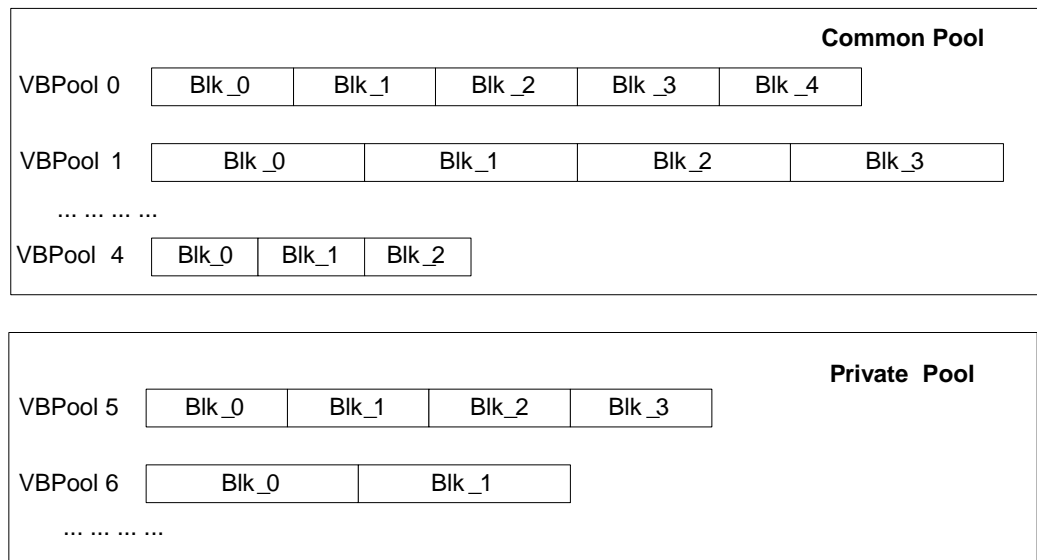
如何根据不同业务配置 MPP 视频缓存池。

【分析】

视频缓存池的功能：主要向媒体业务提供大块物理内存，负责内存的分配和回收，充分发挥内存缓存池的作用，让物理内存资源在各个媒体处理模块中合理使用。MPP 系统从 Hi3507 保留内存区 MMZ (Media Memory Zone) 中申请物理内存创建视频缓存池，缓存池由大小相等、物理地址连续的缓存块组成。假设缓存块个数用 BlkCnt 表示，缓存块大小用 BlkSize 表示，那么创建一个缓存池的过程即为向 MMZ 申请大小为 BlkCnt%BlkSize 的连续物理内存空间，再将缓存池等分为 BlkCnt 个缓存块。所有缓存池由 MPP 系统的 VB 模块管理，其他业务模块再向其申请缓存块以获取相应内存资源。

缓存池分为公共缓存池和私有缓存池，如图 6-1 所示。MPP 系统初始化前，必须先初始化所有公共缓存池，即用户调用 HI_MPI_VB_SetConf 接口对公共缓存池内的缓存块个数和缓存块大小进行配置，再调用 HI_MPI_VB_Init 接口将缓存池初始化（系统内部从 MMZ 中申请内存创建所有公共缓存池）；MPP 系统初始化之后，各模块根据具体业务创建私有缓存池以分配相关资源，另外，用户调用 HI_MPI_VB_CreatePool 创建的缓存池也是私有缓存池。

图6-1 公共缓存池和私有缓存池



公共缓存池内的缓存块主要供 VI、VENC 及 PCIV 模块缓存图像 Buffer。系统将从所有公共缓存块搜寻出空闲的最接近大小的缓存块供各模块使用，因此根据具体业务配置出合适大小的缓存块能有效的利用系统内存资源，例如系统运行时 VI 会有 D1、Half-D1、CIF 等大小的通道配置，那么就需要分别配置出这些大小的缓存池，以图像像素格式 YUV420 为例， $\text{BlkSize} = \text{Stride} \% \text{Height} \% 1.5$ 。

公共缓存池配置中另外一项配置为缓存池中缓存块的个数，如果配置数目不够，将可能导致 MPP 模块的正常业务受到影响（例如无法捕获图像、编码丢帧等）。目前缓存块个数可以按照以下原则配置：

- 每个 VI 通道需要 3 个 VI 帧图像大小的缓存块。
- 每个视频编码通道组需要 3 个主码流大小的缓存块（因此双码流和主次码流的配置是不一样的）。
- 每个与 VI 绑定的 VO 通道需要 2 个 VI 通道大小的缓存块。
- 主片每个 PCIV 通道需要的缓存块个数即为配置的 PCI 的 Buffer 个数，缓存块大小为配置的 PCI 目标图像大小。
- 需要考虑用户调用 HI_MPI_VB_GetBlock 从公共缓存池中获取的缓存块所占用的资源。

由于缓存块可以被多个模块间共用，因此实际需要的缓存块可能比以上说明的要少。一般情况下，参考以上说明进行配置即可，例如单片 1 路 CIF 的编码加预览业务需要的 CIF 大小缓存块个数为： $3+3+2=8$ 块；如果对内存资源利用率有较高要求，可以根据实际业务负荷将每通道的缓存块数目减少 1~2 块，可以通过查看 `/proc/umap/vb` 来了解系统实际运行中视频缓存池的使用情况，通过查看 `/proc/umap/vi` 中的 `VbFail` 了解 VI 是否有获取缓存块失败。

私有缓存池主要用于视频编码、视频解码模块内部为每一路通道缓存帧图像 Buffer，而私有缓存池来源于空闲 MMZ 空间，另外 MPP 系统中也会单独从 MMZ 中获取内存，如果空闲 MMZ 空间不足，将导致创建通道等功能启用失败，因此用户配置 MMZ



时，除了公共缓存池外还要预留一定内存空间。目前 MPP 系统内除公共缓存池外主要其他 MMZ 内存资源需求如下：

- 创建一路 H.264 编码通道，需要 MMZ 空间为图像大小%4。
- 创建一路 H.264 解码通道，需要 MMZ 空间为图像大小%（参考帧数目+4）。
- 创建一路 MJPEG 编码通道，需要 MMZ 空间约 200K 左右。
- 创建一路 MJPEG 解码通道，需要 MMZ 空间为图像大小%3。
- 其他如音频、MD、VPP 等模块也需要从 MMZ 中申请适当内存，总共约 5M 左右。

【解决】

请参见“2.1.4 典型业务资源配置”。

6.6 如何查看 log 日志

【现象】

如何查看日志？如何调整 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。在等级 3 时，记录的信息不多，只有发生错误时，才会将信息记录到日志中，帮助定位绝大多数的错误。等级设置的越高，记录到日志中的信息量也将越多，当等级为 7 时，会把系统的整个运行状态实时的记录到日志中，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。

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/log`，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/log`，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/log`。
- 获取日志记录，可以使用命令 `cat /dev/umap/log`，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。也可以使用 `open`、`read` 等系统调用来操作 `/dev/umap/log` 这个设备节点。



7 Proc 调试信息说明

7.1 概述

调试信息采用了 Linux 下的 proc 文件系统，可实时反映当前系统的运行状态，所记录的信息可供问题定位及分析时使用。

【文件目录】

/proc/umap

【文件清单】

文件名称	描述
sys	记录当前 SYS 模块的使用情况。
vb	记录当前 VB 模块的 buffer 使用情况。
ai	音频输入通道信息。
ao	音频输出通道信息。
log	记录当前各个模块的调试级别。内部调试用。
chnl	CHNL 模块状态。
dsu	专用缩放单元信息。
vdec	视频解码器信息。
venc	视频编码器信息。
vi	视频输入模块信息。
vo	视频输出模块信息。
group	当前编码通道组的属性配置以及当前编码通道统计状态。
vpp	当前所用的遮挡区域和叠加区域的属性信息和状态信息。
md	当前开启 MD 的编码通道的使用状况及其属性配置。



文件名称	描述
H264e	H.264 编码过程中，各通道的编码属性、状态以及历史信息统计。
H264d	H.264 解码过程中，各通道的解码属性、状态以及历史信息统计。
jpege	JPEG 编码过程中，各通道的编码属性、状态以及历史信息统计。

【信息查看方法】

- 在控制台上可以使用 `cat` 命令查看信息，例如 `cat /proc/umap/venc`。
- 在应用程序中将上述文件当作普通只读文件进行读操作，例如 `fopen`、`fread` 等。

说明

不同版本中，这些调试信息可能不同。

7.2 SYS

【调试信息】

```
cat /proc/umap/sys
System State: 0 (0: initialized; 1: exiting; 2: exited)
MPP Configuration:
Align      =      64
PinMuxCtl = bee5eb78
-----benchmark-----
          [name]          calltimes/s  intval(us)  total(us)/s
GROUP_StartOneFrameCallBack  0           0           0
JPEGE_StartOneFrame          0           4           0
JPEGE_IntProcess             0          71           1
JPEGE_InqTask                0           2           0
H264D_StartOneFrameCallBack  0          82           1
H264D_IntHandlerCallBack     0        1830          28
H264D_InqTaskCallBack        0           18           0
```

【调试信息分析】

记录当前 SYS 模块的使用情况。

【参数说明】

参数	描述	
System State	initialized	初始化状态。
	exiting	正在退出状态。
	exited	退出状态。



参数		描述
MPP Configuration	Align	内存对齐长度(字节为单位)。
	PinMuxCtl	内部调试数据。
bechmark	[name]	函数名, 内部调试用。
	calltimes/s	每秒中函数被调用的次数, 内部调试用。
	intval(us)	函数每次调用处理的时间, 内部调试用。
	total(us)/s	函数每秒中所用的总时间, 内部调试用。

7.3 VB

【调试信息】

```

cat /proc/umap/vb
.....Configuration of Video buffer .....
max count of pools: 128
configuration of common pools:
    0    1
size   884736 663552
count  0    10
.....Common.....
POOL_ID  PHYS_ADDR    VIRT_ADDR    IS_COMM  BLK_SZ    BLK_CNT    FREE
1        0xe52e5000   0xc9c80000   1        165888    80         78
BLK  VIU  VOU  DSU  VENC  VDEC  MD  H264E  JPEGE  MPEGE  H264D  JPEGD  MPEGD  VPP  GRP  MPI
43   0    0    0    0    0    0  1     0     0     0     0     0     0   0   0
22   0    1    0    0    0    0  0     0     0     0     0     0     0   0   0

.....kernel.....
POOL_ID  PHYS_ADDR    VIRT_ADDR    IS_COMM  BLK_SZ    BLK_CNT    FREE
3        0xe6409000   0xcb000000   0        884736    4          3(3)
BLK  VIU  VOU  DSU  VENC  VDEC  MD  H264E  JPEGE  MPEGE  H264D  JPEGD  MPEGD  VPP  GRP  MPI
3    0    1    0    0    0    0  0     0     0     0     0     0     0   0   0

```

【调试信息分析】

记录当前 VB 模块的 buffer 使用情况。

【参数说明】

参数		描述
Configuration of Video buffer	max count of pools	最大的缓存池的个数。
	configuration of common pools:	公共缓存池的配置。
	size	缓存池内块的大小。



参数		描述
	count	缓存池内块的个数。
common	POOL_ID	公共缓存池的句柄。
	PHYS_ADDR	公共缓存池的开始物理地址。
	VIRT_ADDR	公共缓存池的开始逻辑地址。
	IS_COMM	是否公共缓存池。 0: 不是。 1: 是。
	BLK_SZ	公共缓存池内缓存块的大小。
	BLK_CNT	公共缓存池内缓存块的个数。
	FREE	公共缓存池空闲缓存块的个数。
	BLK	公共缓存池内缓存块的句柄。
	VIU/VOU/DSU/VE NC/VDEC/MD/H26 4E/JPEGE/MPEGE/ H264D/JPEGD/MP EGD/VPP/GRP/MPI	模块名。 下面对应的数字 0 或 1 表示是否占用相对应的公 共缓存池内的该缓存块。 0: 没占用。 1: 占用。
kernel	POOL_ID	私有缓存池的句柄。
	PHYS_ADDR	私有缓存池的开始物理地址。
	VIRT_ADDR	私有缓存池的开始逻辑地址。
	IS_COMM	是否公共缓存池。 0: 不是。 1: 是。
	BLK_SZ	私有缓存池内缓存块的大小。
	BLK_CNT	私有缓存池内缓存块的个数。
	FREE	私有缓存池空闲缓存块的个数。
	BLK	私有缓存池内缓存块的句柄。
	VIU/VOU/DSU/VE NC/VDEC/MD/H26 4E/JPEGE/MPEGE/ H264D/JPEGD/MP EGD/VPP/GRP/MPI	模块名。 下面对应的数字 0 或 1 表示是否占用相对应的私 有缓存池内的该缓存块。 0: 没占用。 1: 占用。



7.4 LOG

【调试信息】

```
~ $ cat /proc/umap/log
Current Log Level:
cmpi      :0
vb        :0
sys       :0
chnl      :0
grp       :0
venc      :0
vpp       :0
md        :0
h264e     :0
JPEGe     :0
mpeg4e    :0
vdec      :0
h264d     :0
JPEGDd    :0
vo        :0
vi        :0
dsu       :0
sio       :0
ai        :0
ao        :0
aenc      :0
adec      :0
```

【调试信息分析】

记录当前各个模块的调试级别。内部调试用。

【参数说明】

参数		描述
Current Log Level:	cmpi	cmpi 模块名。
	vb	vb 模块名。
	sys	sys 模块名。
	chnl	chnl 模块名。
	grp	group 模块名。
	venc	venc 模块名。
	vpp	vpp 模块名。
	md	md 模块名。
	h264e	h264e 模块名。
	JPEGe	JPEGe 模块名。



参数	描述
mpeg4e	mpeg4e 模块名。
vdec	vdec 模块名。
h264d	h264d 模块名。
JPEGd	JPEGd 模块名。
vo	vo 模块名。
vi	vi 模块名。
dsu	dsu 模块名。
sio	sio 模块名。
ai	ai 模块名。
ao	ao 模块名。
aenc	aenc 模块名。
adec	adec 模块名。

7.5 CHNL

【调试信息】

```

~ $ cat /proc/umap/chnl
----- CHNL state -----
receive interrupt total cnt: 3447
receive timer interrupt cnt: 1355
receive vedu interrupt cnt: 2092
receive error interrupt cnt: 0
channel start fail cnt: 1489
channel start success cnt: 2092
channel schedule cnt: 3581
channel check task cnt: 9568
channel reset VEDU cnt: 0

----- CHNL current run state-----
ID prot
no running channel here

----- CHNL state -----
ID prot pri tsknum state timeout inqtime startOK startNO
3 GROUP 0xc 0 1 0 1936 417 2
2 GROUP 0xc 0 1 0 1936 418 0
1 GROUP 0xc 0 1 0 1936 417 2
0 GROUP 0xc 0 1 0 1935 418 0
0 H264D 0x20 623 1 0 1935 422 1485

```



```

1      VPP      0xd      0      1      0      1982      0      0

----- CHNL performance -----
ID      prot      hw(us)  int(us)  start(us)
3      GROUP      579     88      278
2      GROUP      2808    108     253
1      GROUP      584     87      239
0      GROUP      2909    125     306
0      H264D     1844    97      70
1      VPP      0       0       0

```

【调试信息分析】

无。

【参数说明】

参数		描述
CHNL state	receive interrupt total cnt	CHNL 收到的中断总数。
	receive timer interrupt cnt	CHNL 收到的定时中断数。
	receive vedu interrupt cnt	CHNL 收到的 VEDU 中断。
	receive error interrupt cnt	CHNL 收到的 VEDU 错误中断数。
	channel start fail cnt	CHNL 调度失败次数。
	channel start success cnt	CHNL 调度成功次数。
	channel schedule cnt	CHNL 调度总数。
	channel check task cnt	CHNL 查询通道次数。
	channel reset VEDU cnt	CHNL 进行 VEDUreset 次数。
CHNL state	ID	通道号。
	prot	协议类型。
	pri	优先级。
	tsknum	任务数。
	state	通道状态。 1: 正常。 2: 等待注销。 3: 已经注销。
	timeout	超时计数。
	inqtime	查询次数。
	startOK	启动成功次数。



参数		描述
	startNO	启动失败次数。
CHNL performan ce	ID	通道号。
	prot	协议类型。
	hw(us)	硬件消耗。
	int(us)	中断消耗。
	start(us)	启动消耗。
CHNL current run state	ID	通道号。
	prot	协议类型。

7.6 VI

【调试信息】

```
~ $ cat /proc/umap/vi
```

```
-----  
Attributes of VI Device:
```

```
DevId InptMod WorkMod AdType ViNorm bConfig bEnable  
  0  bt656    2d1     0  pal     y     y  
  1  bt656    2d1     0  pal     y     y  
  2  bt656    2d1     0  pal     y     y  
  3  bt656    2d1     0  pal     y     y
```

```
-----  
Attributes of VI Channel:
```

```
ChnID  RectX  RectY  RectW  RectH  CapSel  DownScl  bHighP  PixFom  CrmRSmp  
  0     8     0    704   288  bottom    y     n  sp420    n  
  1     8     0    704   288  bottom    y     n  sp420    n  
  2     8     0    704   288  bottom    y     n  sp420    n  
  3     8     0    704   288  bottom    y     n  sp420    n  
  4     8     0    704   288  bottom    y     n  sp420    n  
  5     8     0    704   288  bottom    y     n  sp420    n  
  6     8     0    704   288  bottom    y     n  sp420    n  
  7     8     0    704   288  bottom    y     n  sp420    n
```

```
-----  
Status of VI Channel:
```

```
ChnID  IntCnt  LostInt  Width  Height  Field  VBFail  IntTime  SdFrmT  BindVo  
  0    105     0    352   288     4     0    245    206     0  
  1    105     0    352   288     4     0     58     32     1  
  2    105     0    352   288     4     0     46     25     2  
  3    105     0    352   288     4     0     45     25     3  
  4    105     0    352   288     4     0     41     24     4  
  5    105     0    352   288     4     0     41     23     5  
  6    105     0    352   288     4     0     40     22     6  
  7    105     0    352   288     4     0     40     23     7
```

```
-----  
Other infomation of VI channel:
```



ChnID	BufCnt	FrmTime	bPicEn	PicID
0	3	40007	n	0
1	3	40009	n	0
2	3	40009	n	0
3	3	40011	n	0
4	3	40013	n	0
5	3	40015	n	0
6	3	40015	n	0
7	3	40014	n	0

Information of Low Frame Rate:

ChnID	TarFrmR	RelFrmR	SrcFrmR	CapWid	FrmStat
-------	---------	---------	---------	--------	---------

【调试信息分析】

记录当前视频输入设备及通道的属性配置以及状态信息。

【参数说明】

参数	描述	
Attributes of VI Device 视频输入设备属性	DevId	设备号。
	InptMod	输入模式。(bt656/bt601/dc/720p)
	WorkMod	工作模式。(1d1/2d1)
	AdType	AD 类型。(目前未使用)
	ViNorm	视频制式。(pal/ntsc)
	bConfig	是否已设置设备属性。(Y/N)
	bEnable	是否已启用设备。(Y/N)
Attributes of VI Channel 视频输入通道属性	ChnID	通道号。
	RectX	捕获区域起始位置 X 坐标。
	RectY	捕获区域起始位置 Y 坐标。
	RectW	捕获区域宽度。
	RectH	捕获区域高度。
	CapSel	帧场选择。(top/bottom/both)
	DownScl	是否水平压缩。(Y/N)
	bHighP	是否高优先级。(Y/N)
	PixFom	像素格式。(sp420/sp422)
	CrmRSmp	是否色度重采样。(Y/N)
Status of VI Channel	ChnID	通道号。
	IntCnt	中断个数。



参数	描述	
视频输入通道基本状态	LostInt	中断丢失个数。
	Width	图像实际宽度。
	Height	图像实际高度。
	Field	Field 标志。
	VBFail	获取 VideoBuffer 失败次数。
	BindVo	绑定的 VO 通道 (-1 表示未绑定 VO 通道)。
Other infomation of VI channel 视频输入通道其他信息	ChnID	通道号。
	BufCnt	内部 Buffer 个数 (用户可不关心)。
	FrmTime	视频帧采集间隔 (例如 P 制为 40000us)。
	bPicEn	是否启用用户图片。(Y/N)
	PicID	当前通道使用的内部用户图片序号。
Infomation of Low Frame Rate 低帧率相关信息 (满帧率时不显示)	ChnID	通道号。
	TarFrmR	目标帧率。
	RelFrmR	实际帧率。
	SrcFrmR	源帧率 (P 制 25 帧/秒; N 制 30 帧/秒)。
	CapWid	性能宽度 (内部使用)。
	FrmStat	帧捕获状态。 0: 捕获。 1: 未捕获。
Infomation of User Pic 用户图片信息 (设置过用户图片才会显示)	PicID	图片 ID。 0: 用户设置。 其他: 内部生成。
	Width	图片宽度。
	Height	图片高度。
	Stride	图片 Stride。
	PixForm	图片像素格式。(sp420/sp422)
	PoolID	图片物理地址所在 PoolID。
	PhyAddr	图片物理地址。
	bUpdate	图片是否更新。(Y/N)



7.7 VO

【调试信息】

```

~ $ cat /proc/umap/vo
----VOU PUB-----
configed(1) enable(1) norm(PAL) frameRate(25) ptsSpan(40000)
backgroundColor[RGB] (0xffff00) screenRepeat(0) deflickerEnable(1) hideNum(0)
pixelformat(420) outreverse(0)
----GRP STATUS-----
grpId  chnnum  synmode  basechn  frmrate  gap  isstart  basePts
      0      3      MIN      0        25    40000    0        0
----CHN PTS INFORMATION-----
chnid  tgtFR  gap  fstLost  prePts  scalePts
      0   25  40000    0        0        0
      1   25  40000    0        0        0
      2   25  40000    0        0        0
      3   25  40000    0        0        0
----CHN STATUS-----
chnid  getcnt  freebuf  vbfail  frmlost  HF  VLF  VCF
      0  34634    6        0        2      10   9   9
      1  23543    6        0        2      10   9   9
      2  19033    6        0        2      10   9   9
      3  14514    7        0        2      10   9   9
----CHN ATTRIBUTE-----
chnid  bzoom  pri  x  y  w  h  field  show
      0   1    9  0  0  360  288  BOTH  1
      1   1    9  360  0  360  288  BOTH  1
      2   1    9  0  288  360  288  BOTH  1
      3   1    9  360  288  360  288  BOTH  1

```

【调试信息分析】

记录当前 VO 的使用状况及其属性配置，最多有 16 路 VO 通道。可用于检查 VO 属性配置以及当前 VO 通道统计信息。

【参数说明】

参数		描述
vou pub attribute	configed	VO 是否配置。
	enable	VO 是否使能。
	norm	VO 输出制式。（NTSC/PAL，可通过设置模块加载参数 vo_norm=1，将系统初始化为 NTSC 制式，默认为 PAL 制式。）
	frameRate	VO 满帧率。



参数		描述
	ptsSpan	VO 满帧率下的帧间隔。（单位 us）
	Background Color	VO 输出背景颜色（RGB888）。
	screenRepeat	VO 输出屏幕帧重复次数。（用于异常计数）
	deflickerEnable	VO 抗闪烁是否开启。 0：不开启。 1：开启。
	hideNum	VO 隐藏通道个数。
	pixelformat	输出数据格式。（可通过设置模块加载参数 vo_format=422，将输出数据格式设置为 422，默认为 420 输出格式。）
	oureverse	输出顶底场反向标志。（可通过设置模块加载参数 vo_phase=1，将输出顶底场反向，一般用于调试用。）
group status	grpId	同步组组号。
	chnnum	同步组中通道个数。
	synmode	同步模式。
	basechn	同步组时间戳基准通道。
	frmrate	同步组帧率。
	gap	同步组帧间隔时间。
	isstart	同步组是否启动标志。
	basePts	同步组基准时间戳。
chn pts information	Chnid	通道 ID。
	tgtFR	通道目标帧率。
	gap	当前帧率控制下的帧间隔。（单位 us）
	fstLost	通道帧丢帧计数，用于快放时的统计。
	prePts	上一帧时间戳。
	scalePts	下一帧应该播放的时间戳。
chn status	chnid	通道 ID。
	getcnt	通道获取帧计数。
	freebuf	通道 free 链表 buffer 计数。



参数		描述
	vbfail	通道获取 buffer 失败计数。
	frmlost	通道重复帧计数，即丢帧次数。
	HF	通道水平滤波参数。
	VLF	通道垂直亮度滤波参数。
	VCF	通道垂直色度滤波参数。
chn attr	chnid	通道 ID。
	bzoom	是否进行缩放。 0: 否。 1: 是。
	pri	通道输出图像叠加优先级。
	x	通道图像起始坐标 X。
	y	通道图像起始坐标 Y。
	w	通道图像宽度 W。
	h	通道图像高度 H。
	field	通道显示帧场信息。 TOP: 顶场。 BOTTOM: 底场。 BOTH: 两场。
	show	通道显示属性。 0: 隐藏。 1: 显示。

7.8 DSU

【调试信息】

```

~ $ cat /proc/umap/dsu
-----DSU STATUS-----
  busynum  freenum    fail  success
         0      40      0  589514
-----DSU OWNER-----
   vou     vpp    group
         0      0      0
-----DSU FILTER-----
current filter( 0, 0, 0)

```



【调试信息分析】

记录当前 DSU 模块的使用情况。

【参数说明】

参数	描述
busynum	创建的 DSU 任务数。
frenum	当前空闲的 DSU 任务。
fail	创建 DSU 任务失败数。
success	完成 DSU 任务的数目。
vo have	当前 VO 模块创建的 DSU 任务数。
vpp have	当前 VPP 模块创建的 DSU 任务数。
grp have	当前 GROUP 模块创建的 DSU 任务数。
dsu filter	当前 DSU 任务的配置系数。

7.9 VENC

【调试信息】

```

~ $ cat /proc/umap/venc
-----*1*-->venc chnl attribute<---*1*-----
NO. W      H      type  Fd  ViFd  Wm  Mn  SFm  Rx  Seq      Rg  Lp  LB      Pk
0  352    288   96   0    0    0   1   1   1   1cce41  1   0   21261   0
1  176    144   96   0    0    0   1   1   1   1cd801  1   0    0       0
2  352    288   96   0    0    0   1   1   1   1ccf24  1   0   31431   0
3  176    144   96   0    0    0   1   1   1   1cd64d  1   0    0       0
-----*2*-->venc chnl state<---*2*-----
NO. GetSmFd  Notify  RlFrm  FrmF  FrmB  FrmU  UserGet  userRls  CFrmU
0  2922     3205   26    75    26    0    1887809  1887809  0
1  451      1383   25   112    1     0    1890305  1890305  0
2  2696     3199   26    80    25    0    1888036  1888036  0
3  888      1326   25   113    0     0    1889869  1889869  0

```

【调试信息分析】

记录当前编码通道的使用状况及其属性配置，最多有 32 路编码通道。可用于检查属性配置以及当前编码通道统计状态。

【参数说明】

参数		描述
venc chnl	NO.	编码通道号。



参数		描述
attribute	W	编码通道宽度。
	H	编码通道高度。
	type	编码类型。
	Fd	帧场编码。 0: 帧编码。 1: 场编码。
	ViFd	输入图像的帧场标志。 0: 帧模式。 1: 场模式。
	Wm	是否开启数字水印。 0: 没有开启。 1: 开启。
	Mn	主、次码流标识。 0: 次码流。 1: 主码流。
	SFm	是否按帧获取码流。 0: 按包获取。 1: 按帧获取。
	Rx	阻塞或非阻塞获取码流标志。 0: 阻塞。 1: 非阻塞。
	Seq	序列号。按帧获取时为帧序列号，按包获取时为包序列号。
	Rg	是否注册到通道组中。 0: 非注册。 1: 注册。
	LP	待编码的图像数。
	LB	码流 buff 剩余的 byte 数。
	Pk	当前帧的码流包个数。
venc chnl state	NO	编码通道号。
	GetSmFd	该通道获取码流失败的次数。
	Notify	暂无意义，用户不需关注。



参数		描述
	RIFrm	实际帧率。
	FrmF	码流 BUF 中 FREE 节点的个数。
	FrmB	码流 BUF 中 BUSY 节点的个数。
	FrmU	码流 BUF 中 USER 节点的个数。
	UserGet	用户成功获取码流的次数。
	userRls	用户成功释放码流的次数。
	CFrmU	无意义，保留。

7.10 GROUP

【调试信息】

```

~ $ cat /proc/umap/grp
-----*1*-->group chnl attribute<--*1*-----
NO. num W H Fd VeS VpS Top Die St Tim Px Wz Hz Zw Zh OsN PicAdr
0 1 352 288 0 1 1 1 0 1 1 0 0 0 0 0 2 e5eeb000
1 1 176 144 0 1 1 1 0 1 1 0 0 0 0 0 2 e6075e00
-----*2*-->group chnl state<--*2*-----
NO. DsuAdd DsuSub DsuCrt DsuInt Gry TD Flt SlM HWus Intus SCus
0 98306 98306 49153 49153 0 1 0 1 2823 19025 44
1 4236302 4236302 2118151 2118151 0 1 0 1 21977 -1193 249
-----
NO. ViSend GrpSend GrpInq InqOk GrpStart GrpCnVp GrpInt
0 2118151 2118151 3780851 3780807 3780807 2118148 2118148
1 2118151 2118151 3780851 3780807 3780807 2118148 2118148
-----
GNO. Chn typ SendPic SendOk Inq Start StartOk Cfg Int Tr
0 0 96 2118151 2118150 3780828 3780807 2118148 2115451 2118148 0
1 1 96 2118151 2118150 3780827 3780807 2118148 2117729 2118148 411c1a

```

【调试信息分析】

记录当前编码通道组的属性配置以及当前编码通道统计状态。

【参数说明】

参数		描述
group chnl attribute	NO.	GROUP 通道号。
	num	GROUP 内编码通道的个数。
	W	GROUP 通道内主码流编码通道宽度。
	H	GROUP 通道内主码流编码通道高度。



参数		描述
	Fd	GROUP 通道内主码流编码方式。 0: 帧编码。 1: 场编码。
	VeS	是否编码。 0: 不编码。 1: 编码。
	VpS	是否做视频前处理。 0: 不做。 1: 做。
	Top	顶底场标识。 0: 底场。 1: 顶场。主码流为帧编码时, 无效。
	Die	De-interlace 使能指示。 0: 不使能。 1: 使能。
	St	宏块动静判决使能指示。 0: 不使能。 1: 使能。
	Tim	时域滤波使能指示。 0: 不使能。 1: 使能。
	Px	缩放丢点使能开关。 0: 不使能。 1: 使能。
	Wz	水平缩放使能开关。 0: 不使能。 1: 使能。
	Hz	垂直缩放使能开关。 0: 不使能。 1: 使能。
	Zw	水平缩放倍数。
	Zh	垂直缩放倍数。
	OsN	该编码通道组里 OSD 区域的数目。



参数		描述
	PicAdr	编码图像的内存地址（内部调试用）。
group chnl state	NO.	GROUP 通道号。
	DsuAdd	内部 DSU 使用 VB 的次数（内部调试用）。
	DsuSub	内部 DSU 释放 VB 的次数（内部调试用）。
	DsuCrt	创建 DSU 任务的次数（内部调试用）。
	DsuInt	DSU 完成的次数（内部调试用）。
	Gry	彩转灰开关。 0: 关。 1: 开。
	TD	时域滤波开关。 0: 关。 1: 开。
	Flt	缩放系数。
	SIM	缩放模式。
	HWus	硬件时间（内部调试用）。
	Intus	中断开始时间（内部调试用）。
	SCus	中断结束时间（内部调试用）。
	NO.	GROUP 通道号。
	ViSend	VI 通道发送给 GROUP 通道图像数。
	GrpSend	GROUP 通道组发送给其内的编码通道的图像数。
	GrpInq	GROUP 查询次数。
	InqOk	GROUP 查询成功次数。
	GrpStart	GROUP 启动编码次数。
	GrpCnVp	GROUP 启动视频前处理的次数。
	GrpInt	GROUP 中断次数。
	GNO.	GROUP 通道组号。
	Chn	GROUP 通道组内编码通道号。
typ	GROUP 通道组内编码通道编码类型。	
SendPic	GROUP 发送图像的的次数。	



参数		描述
	SendOk	GROUP 发送图像成功的次数。
	Inq	GROUP 通道组查询的次数。
	Start	GROUP 通道组启动编码的次数。
	StartOk	GROUP 通道组启动编码成功的次数。
	Cfg	GROUP 通道组配置硬件寄存器的次数。
	Int	GROUP 通道组中断处理次数。
	Tr	编码通道编码图像的时间计数（内部调试用）。

7.11 VPP

【调试信息】

```
~ $ cat /proc/umap/vpp
```

```
-----*1*-->vpp cover region <--*1*-----
Hdl dev chn pub sw lay x y w h color
0 0 0 0 1 1 100 200 50 50 0
1 0 0 0 1 2 200 200 50 50 ff00
-----*2*-->vpp overlay region <--*2*-----
Hdl Grp Pub sw use fmt x y w h fa ba color phy vrt strd
36 0 0 0 0 8 100 100 180 144 128 128 0 e57de000 ca960000 368
37 0 0 1 0 8 300 100 180 144 50 70 1f e57f8000 cb620000 368
-----*3*-->vpp soft overlay region <--*3*-----
Hdl Dev Chn pub sw use lay x y w h fa color ty tu tv vrt size
168 0 0 0 1 0 4 0 0 200 200 64 0 16 128 128 c0de0000 60000
```

【调试信息分析】

记录当前所用的遮挡区域和叠加区域的属性信息和状态信息。

【参数说明】

参数		描述
THE COVER REGION TOTLE NUM IS 2		遮挡区域的总个数。
vpp cover region 遮挡区域	Hdl	遮挡区域句柄。
	dev	遮挡区域对应的 VI 设备号。
	chn	遮挡区域对应的 VI 通道号。
	pub	是否公共区域。



参数		描述
	sw	是否显示。 0: 不显示。 1: 显示。
	lay	遮挡区域层次。
	x	遮挡区域的起始位置 X 坐标。
	y	遮挡区域的起始位置 Y 坐标。
	w	遮挡区域的宽度。
	h	遮挡区域的高度。
	color	遮挡区域的背景色。
vpp overlay region 叠加 OSD 区域	Hdl	叠加区域 OSD 句柄。
	Grp	叠加区域所属的 GROUP 通道号。
	Pub	是否公共区域。
	sw	是否显示。 0: 不显示。 1: 显示。
	use	正在被几个通道使用。
	fmt	叠加区域像素格式。
	x	叠加区域的起始位置 X 坐标。
	y	叠加区域的起始位置 Y 坐标。
	w	叠加区域的宽度。
	h	叠加区域的高度。
	fa	前景 ALPHA 值。
	ba	背景 ALPHA 值。
	color	背景色。
	phy	叠加区域的内存物理地址（内部调试用）。
vrt	叠加区域的内存虚拟地址（内部调试用）。	
strd	叠加区域的行内存宽度（内部调试用）。	
vpp soft overlay region	Hdl	软叠加区域的句柄。
	Dev	软叠加区域的对应的 VI 设备号。



参数		描述
软叠加 OSD 区域	Chn	软叠加区域的对应的 VI 通道号。
	pub	是否公共区域。 0: 非公共区域。 1: 公共区域。
	sw	是否显示。 0: 不显示。 1: 显示。
	use	正在被几个通道使用。
	lay	软叠加区域层次。
	x	软叠加区域的起始位置 X 坐标。
	y	软叠加区域的起始位置 Y 坐标。
	w	软叠加区域的宽度。
	h	软叠加区域的高度。
	fa	软叠加区域 ALPHA 值。
	color	软叠加区域背景色。
	ty	内部调试数据。
	tu	内部调试数据。
	tv	内部调试数据。
	virt	内部调试数据。
size	内部调试数据。	

7.12 MD

【调试信息】

```

~ $ cat /proc/umap/md
-----*1*-->MD chnl attribute and mode<--*1*-----
NO.  W   Y   type RefM RefS SAD MV  MBAL PelAlNum Dlight SadB
0   352 288 96  0   1   1   0   0   0   0   0
1   176 144 96  0   1   1   0   0   0   0   0
2   352 288 96  0   1   1   0   0   0   0   0
3   176 144 96  0   1   1   0   0   0   0   0
-----
NO.  Int  Buf  STh  PTh  PNTh  DLB  DLA  TolM          TolSz  UsePM  UseSd
0   5    16   0   0   0     0   0   e76d7000     234880  0     0
1   5    16   0   0   0     0   0   e7711000     59776  0     0

```



```

2  5  16  0  0  0  0  0  e7720000  234880  0  0
3  5  16  0  0  0  0  0  e775a000  59776 0  0

-----*2*-->MD state<--*2*-----
NO. Start  Conf    End      GetD     ReaD     SADRefA   SRefSd  SADN  CSadRfA
0  321848  317520  317520  317519  317519  e76f2400  180   0     e76d7400
1  320046  317523  317523  317519  317519  e7711400  c0    1     e7718000
2  321714  317522  317522  317519  317519  e773b400  180   0     e7720400
3  320401  317523  317523  317519  317519  e775a400  c0    1     e7761000

-----
NO.   MvRefA  MvRefSd  MvN     CMvRefA  PrLdMvA  PrStMvd  PrN     RltMvA  RltMvSd
0     0       0         0       0         0         0         0       0       0
1     0       0         0       0         0         0         0       0       0
2     0       0         0       0         0         0         0       0       0
3     0       0         0       0         0         0         0       0       0

```

【调试信息分析】

记录当前开启 MD 的编码通道的使用状况及其属性配置，最多可启用 32 路编码通道做移动侦测。可用于检查属性配置以及当前编码通道统计状态。

【参数说明】

参数		描述
MD chnl attribute and mode	NO.	编码通道号。
	W	编码通道宽度。
	Y	编码通道高度。
	Type	编码类型。
	RefM	参考图像模式。 0: 自动模式。 1: 用户输入模式。
	RefS	参考图像状态。 0: 不更新。 1: 更新。
	SAD	宏块 SAD 是否开启。 0: 不开启。 1: 开启。
	MV	宏块 MV 是否开启。 0: 不开启。 1: 开启。
	MBAI	宏块报警信息是否开启。 0: 不开启。 1: 开启。



参数		描述
	PelAInum	宏块报警像素的个数是否开启。 0: 不开启。 1: 开启。
	Dlight	去光照效应是否开启。 0: 不开启。 1: 开启。
	SadB	宏块 SAD 的精度。 0: 8bit。 1: 16bit。
	Int	间隔数。
	Buf	结果缓存的个数。
	STh	SAD 阈值。
	PTh	像素报警阈值。
	PNTh	像素报警个数阈值。
	DIB	去光照效应的 Bata 值。
	DIA	去光照效应的 alpha 值。
	TolM	总内存地址（内部调试用）。
	TolSz	总内存的大小。
	UsePM	用户提供的参考图像的内存地址。
	UseSd	用户提供的参考图像的行内存宽度。
MD state	Start	启动成功次数。
	Conf	配置硬件寄存器次数。
	End	结束的次数。
	GetD	用户获取 MD 结果的次数。
	ReaD	用户释放 MD 结果的次数。
	SADRefA	SAD 参考图像地址（内部调试用）。
	SRefSd	SAD 参考图像的行内存宽度（内部调试用）。
	SADN	SAD 参考图像内存倒换标识（内部调试用）。
	CSadRfA	SAD 参考图像倒换内存地址（内部调试用）。
	MvRefA	MV 参考图像地址（内部调试用）。



参数		描述
	MvRefSd	MV 参考图像的行内存宽度（内部调试用）。
	MvN	MV 参考图像内存倒换标识（内部调试用）。
	CMvRefA	MV 参考图像倒换内存地址（内部调试用）。
	PrLdMvA	MV 累加信息载入地址（内部调试用）。
	PrStMvd	MV 累加信息输出地址（内部调试用）。
	PrN	MV 累加信息内存倒换标识（内部调试用）。
	RltMvA	MV 结果内存地址（内部调试用）。
	RltMvSd	MV 结果内存行宽度（内部调试用）。

7.13 VDEC

【调试信息】

```
~ $ cat /proc/umap/vdec
```

```
-----chnl attr-----
```

ID	type	curnalu	curlen	vobind	frmrate	prio	width	height	frmnum
0	96	c868b858	43267	8	26	0	352	288	2
1	96	c883f3ec	13180	9	25	0	352	288	2
2	96	c8a15400	3950	10	26	0	352	288	2
3	96	c8bbf5bc	13148	11	25	0	352	288	2

```
-----chnl state-----
```

ID	sndstm	rlsstm	rlsfail	sndpic	sndudata	getpic	getudata	rlspic	rlsudata
0	176	173	0	149	0	0	0	0	0
1	172	170	0	146	0	0	0	0	0
2	180	173	0	149	0	0	0	0	0
3	171	169	0	145	0	0	0	0	0

```
-----buf state-----
```

ID	addr	size	wptr	rprr	wable	rable	free	busy
0	e3a41000	304128	e3a4c858	e3a7a148	186608	117520	1	0
1	e3bd9000	304128	e3c183ec	e3c11d9c	277936	26192	1	0
2	e3d71000	304128	e3d86400	e3d71000	217088	87040	1	0
3	e3f09000	304128	e3f485bc	e3f42008	278092	26036	1	0

```
-----pack scan state-----
```

ID	sndinglen	packok	packwait	packerr	notifytime	snd	cost	lefts
0	43272	176	177	0	473	0	0	0
1	13184	172	173	0	472	0	0	0
2	3956	180	181	0	471	0	0	0
3	13152	171	172	0	470	0	0	0



【调试信息分析】

记录当前解码通道的使用状况及其属性配置，最多有 32 路解码通道。可用于检查属性配置以及当前解码通道统计状态。

【参数说明】

参数		描述
Chnl attr	ID	通道号。
	type	解码通道类型，96--PT_H264,1002--PT_MJPEG, 26--PT_JPEG。
	curnalu	当前正在扫描的 NALU 包地址。
	curlen	当前已经扫描的 NALU 长度。
	vobind	Bind 的 VO 通道号，-1 表示没有 bind vo。
	frmrate	实际解码帧率，如果码流包有错误，此数可能不准确。
	prio	解码通道优先级。
	width	配置的解码图像宽度。
	height	配置的解码图像高度。
	frmnum	配置的参考帧个数，H.264 有效，其他类型无此选项。
Chnl state	ID	通道号。
	sndstm	发送码流给解码器次数。
	rlsstm	解码器释放码流次数。
	rlsfail	解码器释放失败次数。
	sndpic	解码器发送 pic 次数。
	sndudata	解码器发送用户数据次数。
	getpic	用户获取 pic 次数。
	getudata	用户获取用户数据次数。
	rlspic	用户释放 pic 次数。
	rlsudata	用户释放用户数据次数。
Buf state	ID	通道号。
	addr	码流 buf 首地址。
	size	码流 buf 长度。
	wptr	写指针位置。



参数		描述
	rptr	读指针位置。
	wable	可写空间。
	rable	可读空间。
	free	处于 free 状态的 pack 节点。
	busy	处于 busy 状态的 pack 节点。
pack scan state	ID	通道号。
	sndinglen	用户正在发送的码流的长度。
	packok	成功扫描到 pack 边界的次数。
	packwait	等待扫描 pack 边界的次数。
	packerr	pack 扫描错误次数。
	notifytime	通知发送码流次数。
	snd	发送码流帧数。按帧发送时有效。
	cost	解码码流帧数。按帧发送时有效。
	lefts	码流 buffer 剩余的码流帧数。按帧发送时有效。

7.14 AI

【调试信息】

```
~ $ cat /proc/umap/ai
-----
Attribution of AI Device:
DevId WorkMod  Sampl  BitWid  SondMod  PoiNum  ExFlag  FrmNum
   1  slave    8k    8bit    momo     320     1      30
-----
Status of AI Device:
DevId  IntCnt    DMAPhy0    DMAPhy1
   1    112     e2818000   e2818f00
-----
Status of AI Channel:
DevId  ChnId  KerId  Read  Write  BusyCnt  ThrShd1  ThrShd2  Intlost  AecAo
   1     0    11    22    22     0       15       7        0       -1
```

【调试信息分析】

记录当前音频输入设备及通道的属性配置以及状态信息。

【参数说明】



参数	描述	
Attribution of AI Device (音频输入设备属性)	DevId	AI 设备号。
	WorkMod	SIO 工作模式。 master: 主模式。 slave: 从模式。
	Sampl	采样率。(8k、16k)
	BitWid	采样精度。(8bit/16bit)。
	SondMod	声音模式。 momo: 单声道。 stereo: 立体声。
	PoiNum	每帧的采样点个数。
	ExFlag	扩展标志。
	FrmNum	帧缓存数目。
Status of AI Device (音频输入设备信息)	DevId	AI 设备号。
	IntCnt	中断个数。
	DMAPhy0	DMA Buffer0 的物理地址。
	DMAPhy1	DMA Buffer1 的物理地址。
Status of AI Channel (音频输入通道信息)	DevId	AI 设备号。
	ChnId	AI 通道号。
	KerId	内部 ID 号。
	Read	通道 Buffer 的读指针。
	Write	通道 Buffer 的写指针。
	BusyCnt	已采集但未及时取用的数据的长度。
	ThrShd1	Buffer 管理一级水线。
	ThrShd2	Buffer 管理二级水线。
	Intlost	中断丢失次数 (Buffer 满)。
	AecAo	回声抵消 AEC 的 AO 通道号 (-1 表示未启用 AEC)。



7.15 AO

【调试信息】

```

~ $ cat /proc/umap/ao
-----
Attribution of AO Device:
DevId WorkMod  Sampl  BitWid  SondMod  PoiNum  ExFlag  FrmNum
    0  master    8k   16bit   momo     320     0       30
-----
Status of AO Device:
DevId  IntCnt      DMAPhy0      DMAPhy0
    0   32327     e2991000     e2991f00
-----
Status of AO Channel:
DevId  ChnId  KerId  Read  Write  BusyCnt  ThrShd1  ThrShd2  State  Intlost
    0     0     1     16   17     1       15       7 enable  1

```

【调试信息分析】

记录当前音频输出设备属性配置以及状态信息。

【参数说明】

参数		描述
Attribution of AO Device (音频输出设备属性)	DevId	AO 设备号。
	WorkMod	SIO 工作模式。 master: 主模式。 slave: 从模式。
	Sampl	采样率。(8k/16k)
	BitWid	采样精度。(8bit/16bit)
	SondMod	声音模式。 momo: 单声道。 stereo: 立体声。
	PoiNum	每帧的采样点个数。
	ExFlag	扩展标志。
	FrmNum	帧缓存数目。
Status of AO Device (音频输出设备信息)	DevId	AO 设备号。
	IntCnt	中断个数。
	DMAPhy0	DMA Buffer0 的物理地址。
	DMAPhy1	DMA Buffer1 的物理地址。



参数	描述	
Status of AO Channel (音频输出通道信息)	DevId	AO 设备号。
	ChnId	AO 通道号。
	KerId	内部 ID 号。
	Read	通道 Buffer 的读指针。
	Write	通道 Buffer 的写指针。
	BusyCnt	已送到 AO 但未及时播放的数据长度。
	ThrShd1	Buffer 管理一级水线。
	ThrShd2	Buffer 管理二级水线。
	state	AO 通道状态 1: 启用。 2: 暂停。
	Intlost	中断丢失次数 (Buffer 空)。

7.16 H264E

【调试信息】

```

~ $ cat /proc/umap/h264e
----- h264e channel attribute info -----
  ID  Pri    Width  Height  MainStr  VIField  BufSize  ByFrame  MaxStrCnt
  0    0     352    288     Yes      No       228096   Yes      0xffffffff
  2    0     352    288     Yes      No       228096   Yes      0xffffffff

----- h264e RateCtrl attribute-----
  ID  PicMbs  InFrmRt  OutFrmRt  Field  CBR  BitRt (Kbps)  Level  Gop  MaxDly (ms)
  0   396     25       25        No     No   512           3     20  10
  2   396     25       25        No     No   512           3     20  10

----- h264e picture information -----
  ID  rcv    encode  disc  skip  bufleak  back  rlsstr  unrdstr
  0   580    578     0     0     0        0     578     0
  2   580    578     0     0     0        0     578     0

----- h264e stream buffer -----
  ID  base          BufLen  RdTail  RdHead  WrTail  WrHead  datalen  buffree
  0   0xc92c0a80  0x37b00  0x20480  0x20480  0x20480  0x20480  0        228032
  2   0xc9400a80  0x37b00  0x1fc40  0x1fc40  0x1fc40  0x1fc40  0        228032

----- h264e channel info -----

```



```
ID  Txturet  OsdPret  SlcSplT  SlcSize  NoOutEtrStrm
0   Yes     Yes      No       1024     No
2   Yes     Yes      No       1024     No
```

----- h264e prev process info -----

```
ID  Die  EdegFlt  MedFlt  TimeFlt  MvStl  PicTp  Repet  WaterMk
0   No   No       No      Yes      Yes    0      No     No
2   No   No       No      Yes      Yes    0      No     No
```

----- h264e channel performance info -----

```
ID  hw(us)  int(us)  strt(us)  cfgreg(us)  br(kbps)  fr(fps)
0   3087    81      111      28          517       25
2   2971    69      117      27          618       25
```

【调试信息分析】

无。

【参数说明】

参数		描述
h264e channel attribute info	ID	通道号。
	Pri	优先级。
	Width	宽度，单位像素。
	Height	高度，单位像素。
	MainStr	是否主码流。 NO: 否。 YES: 是。
	VIField	输入是否场模式。 NO: 否。 YES: 是。
	BufSize	码流 buffer 大小，单位字节。
	ByFrame	是否按帧获取码流。 NO: 否。 YES: 是。
	MaxStrCnt	允许码流 Buffer 缓存的最大帧数。 缺省值: 0x1fffffff。
h264e RateCtrl attribute	ID	通道号。
	PicMbs	图像 MB 数。
	InFrmRt	图像输入帧率单位 fps(帧每秒)。
	OutFrmRt	编码帧率，单位 fps。



参数		描述
	Field	是否场模式。 NO: 否。 YES: 是。
	CBR	是否 CBR。 NO: 否。 YES: 是。
	BitRt(Kbps)	码率, 单位 kbit/s(千比特每秒)。
	Level	图像质量等级, [0, 5]。
	Gop	I 帧间隔。
	MaxDly(ms)	最大延迟, 单位 ms。
h264e picture informatio n	ID	通道号。
	rcv	收到图像数。
	encode	编码图像数。
	disc	由于图像 buffer 满丢弃图像数。
	skip	图像队列中未编码图像数。
	bufleak	由于码流 buffer 满编码前丢弃图像数。
	back	由于码流 buffer 满丢弃已经编码图像数。
	rlsstr	已经释放的码流帧数。
	unrdstr	当前 Buffer 中缓存的码流帧数。
h264e stream buffer	ID	通道号。
	base	码流 buffer 基地址。
	BufLen	码流 buffer 大小。
	RdTail	读尾指针。
	RdHead	读头指针。
	WrTail	写尾指针。
	WrHead	写头指针。
	datalen	数据长度。
	buffree	空闲长度。
h264e	ID	通道号。



参数		描述
RateCtrl attribute	PicMbs	图像 MB 数。
	InFrmRt	图像输入帧率, 单位 fps(帧每秒)。
	OutFrmRt	编码帧率, 单位 fps。
	Field	是否场模式。 NO: 否。 YES: 是。
	CBR	是否 CBR。 NO: 否。 YES: 是。
	BitRt(Kbps)	码率, 单位 kbit/s(千比特每秒)。
	Level	图像质量等级, [0, 5]。
	Gop	I 帧间隔。
	MaxDly(ms)	最大延迟, 单位 ms。
h264e channel info	ID	通道号。
	Txturet	纹理检测开关。 NO: 关闭。 YES: 开启。
	OsdPret	OSD 保护开关。 NO: 关闭。 YES: 开启。
	SlcSplt	slice 划分。 NO: 不划分。 YES: 划分。
	SlcSize	slice 大小, 单位字节。
	NoOutEtrStrm	不输出超出码流。 NO: 输出。 YES: 不输出。
h264e prev process info	ID	通道号。
	Dei	De-interlace 开关。 NO: 关闭。 YES: 开启。



参数		描述
	EdegFlt	边缘检测开关。 NO: 关闭。 YES: 开启。
	MedFlt	中值检测开关。 NO: 关闭。 YES: 开启。
	TimeFlt	时域去噪开关。 NO: 关闭。 YES: 开启。
	MvStl	动静判决开关。 NO: 关闭。 YES: 开启。
	PicTp	图像类型。 0: 4:2:0。 1: 4:2:2。
	Repet	重复编码开关。 NO: 关闭。 YES: 开启。
	WaterMk	数字水印开关。 NO: 关闭。 YES: 开启。
h264e channel performan ce info	ID	通道号。
	hw(us)	硬件消耗, 单位 μs 。
	int(us)	中断销毁, 单位 μs 。
	strt(us)	启动消耗, 单位 μs 。
	cfgreg(us)	配寄存器消耗, 单位 μs 。
	br(kbps)	瞬时码率, 单位 bit/s。 注意: 瞬时码率与平均码率的计算方法不同, 码率控制是根据平均码率进行的, 因此该值仅供参考, 不宜作为评价码率控制的标准。
	fr(fps)	瞬时帧率, 单位 frame/s。



7.17 H264D

【调试信息】

```

~ $ cat /proc/umap/h264d
----- Perf (us) -----
ID Parse HWdec SWdec SWint HWrprY SWrprY SWintY HWrprC SWrprC SWintC
0 431 1727 85 100 0 0 0 0 0 0

----- Const -----
ID W H Ref DispQue RmvQue SendQue DescBuf
0 720 576 2 1999 2063 63 414592

----- Status -----
ID W H Ref DispQue RmvQue SendQue DescBuf Out Pic 10Fld State Slot
0 352 288 1 378 378 0 49408 0 0 0 1 0

----- HW Stat -----
ID Start Eop Empty StartY IntY StartC IntC
0 118 118 0 0 0 0 0

----- Err Stat -----
ID ErrNalu LostPic ErrSlc LostSlc Overlap ErrW ErrH ErrRef ErrCncl
0 0 0 0 0 0 0 0 3845 0

----- Flow Stat -----
ID RecvPic DecPic SendPic RecvAUD RlsAud RecvEOS NoFB Rst pps fps
0 800 794 782 0 0 1 0 0 25 25

```

【调试信息分析】

记录 H.264 解码过程中，各通道的解码属性、状态以及历史信息统计，最多有 32 路解码通道。可配合用于定位系统出现的阻塞以及图像错误等问题。

【参数说明】

参数		描述
Perf 性能 ^a	ID	通道号。
	Parse	软件解析一个 NALU 的时间（进程上下文）。
	HWdec	硬件解码一幅图像时间。
	SWdec	软件准备解码一幅图像时间（中断上下文）。
	SWint	完成图像解码后的软件处理时间（中断上下文）。
	HWrprY	硬件修补 Y 分量时间。
	SWrprY	软件准备修补 Y 分量时间（中断上下文）。
	SWintY	完成 Y 修补后的软件处理时间（中断上下文）。
	HWrprC	硬件修补 C 分量时间。



参数		描述
	SWrprC	软件准备修补 C 分量时间（中断上下文）。
	SWintC	完成 C 修补后的软件处理时间（中断上下文）。
Const 常量 ^b	ID	通道号。
	W	通道属性宽度。
	H	通道属性高度。
	Ref	通道属性参考帧个数。
	DispQue	显示图像队列总长度。
	RmvQue	移除队列总长度。
	SendQue	发送队列总长度。
	DescBuf	硬件消息 Buffer 总长度。
Status 状态 ^c	ID	通道号。
	W	码流当前宽度。
	H	码流当前高度。
	Ref	码流当前参考帧个数。
	DispQue	显示图像队列当前长度。
	RmvQue	移除队列当前长度。
	SendQue	发送队列当前长度。
	DescBuf	硬件消息 Buffer 当前长度。
	Out	图像输出模式。 0: 快速输出。 1: DPB 满输出。
	Pic	当前解码的图像属性。 0: 帧。 1: 顶场。 2: 底场。 3: 场对。
	10Fld	是否 3510 私有场。 0: 非。 1: 是。



参数		描述
	State	解码器状态。 0: 等待。 1: 解码。 2: 修补 Y。 3: 修补 C。
	Slot	图像发送缓存状态。用于场配对输出。 0: 空。 1: 缓存一个顶场。
HW Stat 硬件统计 ^d	ID	通道号。
	Start	启动解码次数。
	Eop	图像结束中断次数。
	Empty	空中断次数。
	StartY	修补 Y 次数。
	IntY	修补 Y 中断次数。
	StartC	启动修补 C 次数。
	IntC	修补 C 结束中断次数。
Err Stat 错误统计 ^e	ID	通道号。
	ErrNalu	软件解析发现的错误 NALU 个数。 因 SEI 错误不影响图像显示，所以未包括。
	LostPic	软件发现图像丢失个数。
	ErrSlc	硬件发现 Slice 错误次数。
	LostSlc	硬件发现 Slice 丢失次数。
	Overlap	硬件发现 Slice 重叠次数。
	ErrW	通道宽度设置错误次数。 属性无法支持当前码流时，认为属性设置错误。
	ErrH	通道高度设置错误次数。 属性无法支持当前码流时，认为属性设置错误。
	ErrRef	通道参考帧个数设置错误次数。 属性无法支持当前码流时，认为属性设置错误。
ErrCncl	软件解码器进行错误掩盖的次数。	
Flow Stat	ID	通道号。



参数	描述	
流程统计 ^f	RecvPic	接收的图像个数。 等于 DecPic+1 时，表明正在接收完整的一幅图像。
	DecPic	已解码的图像个数。
	SendPic	已发送的图像总数。 当 Out=1 时，表示帧图像个数。 若是场码流，该数值应为 DecPic/2。
	RecvAud	接收到的码流帧数。按帧发送码流时有效。
	RlsAud	已解码的码流帧数。按帧发送码流时有效。
	RecvEOS	已接收的 EOS NALU 总数。
	NoFB	无空闲帧存可用的次数。
	Rst	解码通道复位次数。
	pps	在相邻两次查看 proc 信息之间的时间段内，平均每秒钟解码的图象数。 通过 <code>cat /proc/umap/h264d;sleep n;cat /proc/umap/h264d</code> 可查看 n 秒内 h264d 平均每秒解码的图象数。
	fps	在相邻两次查看 proc 信息之间的时间段内，平均每秒钟解码的帧数。 通过 <code>cat /proc/umap/h264d;sleep n;cat /proc/umap/h264d</code> 可查看 n 秒内 h264d 平均每秒解码的帧数。

- a: 软硬件解码性能，一般不需要关注。
- b: 通道创建后不再改变的参数，包括通道属性和解码内部资源，一般不需要关注。
- c: 系统当前的状态，随解码过程变化。一般不需要关注。
- d: 硬件控制统计。一般不需要关注。
- e: 解码错误统计。定位图像显示错误现象。
- f: 解码流程统计。定位系统阻塞现象。

7.18 JPEGG

【调试信息】

```

~ $ cat /proc/umap/JPEGG
----- Attr -----
ID  MJPG  FrmW  FrmH  Vi  Fld  MCU  ByFrm  [Main  TFR  TBR]  [ImgQ]
1   1     352   288   0   0    396  1      1     25   4096   0

----- Status -----
ID  BufLen  FreeLen  FQue  EQue  Q  FR  BR  PicByte

```



```
1 202752 0 0 26 99 36 468 2279
```

```
----- Flow Stat -----
```

```
ID SendPic IntEnd IntFull DropPic NoBuf NoPic RcFail VppFail
1 37 37 0 37 0 0 0 0
```

【调试信息分析】

记录 JPEG 编码过程中，各通道的编码属性、状态以及历史信息统计，最多有 32 路编码通道。可配合用于定位系统阻塞以及丢帧等问题。

【参数说明】

参数		描述
Attr 属性 ^a	ID	通道号。
	MJPG	是否 MJPEG 编码。 0: JPEG 抓拍。 1: MJPEG。
	FrmW	图像宽度。
	FrmH	图像高度。
	Vi	VI 输入图像属性。 0: 帧。 1: 场。
	Fld	编码属性。 0: 帧编码。 1: 场编码。
	MCU	每个 ECS 的 MCU 个数。
	ByFrm	是否按帧获取码流。 0: 否。 1: 是。
	Main	是否主码流。 0: 否。 1: 是。
	TFR	目标帧率。
	TBR	目标码率。
ImgQ	图像质量。	
Status 状态 ^b	ID	通道号
	BufLen	码流 Buffer 总长度。



参数		描述
	FreeLen	空闲码流 Buffer 长度。
	FQue	可接收的图像个数。
	EQue	待编码图像个数。
	Q	量化因子。
	FR	实际帧率。（数据不精确，仅供参考）
	BR	实际码率。（数据不精确，仅供参考）
	PicByte	编码后的一帧码流 Byte 数。
Stat 统计 ^c	ID	通道号。
	SendPic	发送来编码的图像总数。
	IntEnd	硬件编码结束中断次数。
	IntFull	硬件 Buffer 满中断次数（会导致丢帧）。
	DropPic	归还图像帧存的次数。
	NoBuf	发现码流 Buffer 不足的次数（会导致丢帧）。
	NoPic	没有待编码图像的次数。
	RcFail	码率/帧率控制失败的次数（会导致丢帧）。
	VppFai	VPP(视频前处理)设置失败的次数。

a: 一般不需要关注。

b: 系统当前的状态，随编码过程变化。可用于定位系统阻塞。

c: 历史统计信息。可用于定位丢帧。