AX SYS API 文档

文档版本：V2.2

发布日期：2024/09/27

目 录

前 言 7
修订历史 8
1 概述 10
2 概念与术语 11
3 系统控制模块 12
3.1 内存管理 12
3.2 内存优化 14
3.3 配置内存分段 15
3.4 Cache 一致性管理 17
3.5 时间管理 17
3.6 日志管理 17
3.7 链路管理 18
3.8 休眠唤醒 19
4 API 简介 21
4.1 系统初始化 21
4.2 内存分配 21
4.3 缓存池管理 22
4.4 时间管理 23
4.5 日志管理 23
4.6 链路管理 24
4.7 芯片类型管理 24
4.8 休眠唤醒 24

5 API 定义 25
5.1 系统初始化 API 25
AX_SYS_Init 25
AX_SYS_Deinit 26
5.2 内存分配 API 26
AX_SYS_MemAlloc 26
AX_SYS_MemAllocCached 29
AX_SYS_MemFree 31
AX_SYS_MemGetBlockInfoByPhy 32
AX_SYS_MemGetBlockInfoByVirt 33
AX_SYS_MemGetPartitionInfo 34
AX_SYS_Mmap 35
AX_SYS_MmapCache 37
AX_SYS_MmapFast 39
AX_SYS_MmapCacheFast 40
AX_SYS_Munmap 41
AX_SYS_MflushCache 42
AX_SYS_MinvalidateCache 43
AX_SYS_MemSetConfig 44
AX_SYS_MemGetConfig 46
AX_SYS_MemSetParam 47
AX_SYS_MemGetParam 49
AX_SYS_MemQueryStatus 50
AX_SYS_MemGetMaxFreeRegionInfo 51
5.3 缓存池管理 API 52
AX_POOL_SetConfig 52

AX_POOL_GetConfig 53
AX_POOL_Init 54
AX_POOL_Exit 55
AX_POOL_CreatePool 56
AX_POOL_DestroyPool 57
AX_POOL_GetBlock 58
AX_POOL_ReleaseBlock 59
AX_POOL_Handle2PhysAddr 60
AX_POOL_PhysAddr2Handle 61
AX_POOL_Handle2MetaPhysAddr 62
AX_POOL_Handle2PoolId 63
AX_POOL_Handle2BlkSize 64
AX_POOL_GetBlockVirAddr 65
AX_POOL_GetMetaVirAddr 66
AX_POOL_MmapPool 67
AX_POOL_MunmapPool 68
AX_POOL_IncreaseRefCnt 69
AX_POOL_DecreaseRefCnt 70
AX_POOL_FreeCommPool 71
5.4 时间管理 API 72
AX_SYS_GetCurPTS 72
AX_SYS_InitPTSBase 73
AX_SYS_SyncPTS 74
5.5 日志管理 API 75
AX_SYS_LogPrint 75
AX_SYS_LogOutput 77

5.6 链路管理 API 79
AX_SYS_Link 79
AX_SYS_UnLink 81
AX_SYS_GetLinkByDest 82
AX_SYS_GetLinkBySrc 83
AX_SYS_SetVINIVPSMode 84
5.7 获取芯片类型 API 86
AX_SYS_GetChipType 86
5.8 休眠唤醒 API 87
AX_SYS_Sleep 87
AX_SYS_RegisterEventCb 88
AX_SYS_UnregisterEventCb 90
AX_SYS_WakeLock 91
AX_SYS_WakeUnlock 92
AX_SYS_SleepTimeStamp 93
6 数据类型 94
AX_POOL_CACHE_MODE_E 96
AX_POOL_SOURCE_E 97
AX_POOL_CONFIG_T 98
AX_POOL_FLOORPLAN_T 100
AX_PARTITION_INFO_T 101
AX_CMM_PARTITION_INFO_T 102
AX_CMM_STATUS_T 103
7 错误码 104
8 调试信息 105

8.1 CMM 105
8.2 缓存池 106
8.3 绑定关系 111
8.4 VIN、IVPS 模式配置 112

权利声明

注意

您购买的产品、服务或特性等应受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。除非商业合同另有约定，本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。

前 言

本文档主要介绍媒体子系统中系统控制模块的功能和用法，其它模块的功能和用法将各有专门的文档加以论述。

适用产品

AX620E 系列产品（AX630C、AX620Q）

适读人群

- 软件开发工程师
- 技术支持工程师

符号与格式定义

符号/格式	说明
`Xxx`	表示您可以执行的命令行。
斜体	表示变量。如，“*安装目录*/AX620E_SDK_Vx.x.x/build 目录”中的“安装目录”是一个变量，由您的实际环境决定。
☞ 说明/备注：	表示您在使用产品的过程中，我们向您说明的事项。
！ 注意：	表示您在使用产品的过程中，需要您特别注意的事项。

修订历史

文档版本	发布时间	修订说明
V1.0	2023/8/21	文档初版
V1.1	2023/9/13	**第 6 章 数据类型** AX_MAX_POOLS、AX_MAX_COMM_POOLS、 AX_MAX_BLKS_PER_POOL 涉及修改。
V1.2	2023/10/24	**第 3 章 系统控制模块** 3.7 小节，涉及修改，新增 AVS 模块。
V1.3	2023/11/30	公共结构体引用《50 - AX 公共数据结构文档》。
V1.4	2023/12/06	**第 5 章 API 定义** 5.3 小节，AX_POOL_Handle2PhysAddr 【注意】涉及修改。
V1.5	2024/01/19	**第 7 章 错误码** 新增 COPY_TO_USER、COPY_FROM_USER 错误码。
V1.6	2024/01/25	**第 6 章 数据类型** AX_LINK_DEST_MAXNUM 涉及修改； **第 8 章 调试信息** 8.2 小节，涉及修改。
V1.7	2024/1/29	**第 3 章 系统控制模块** 3.8 小节， 新增休眠唤醒接口； **第 4 章 API 简介** 4.8 小节，新增休眠唤醒接口； **第 5 章 API 定义** 5.8 小节，新增休眠唤醒接口。
V1.8	2024/05/22	**第 5 章 API 定义** 5.2 小节，新增 AX_SYS_MemSetParam 和

文档版本	发布时间	修订说明
		AX_SYS_MemGetParam 接口；
V1.9	2024/05/27	**第 5 章 API 定义** 5.2 小节，新增 AX_SYS_MemGetMaxFreeRegionInfo 接口。
V2.0	2024/06/03	**第 5 章 API 定义** 5.2 小节，新增 AX_SYS_MemGetMaxFreeRegionInfo 接口注意事项； 5.3 小节，新增 AX_POOL_FreeCommPool 接口。
V2.1	2024/6/5	**第 5 章 API 定义** 5.2 小节，新增 AX_SYS_MemSetConfig 接口 ivps 和 vo 相关的注意事项。
V2.2	2024/9/27	**第 5 章 API 定义** 5.6 小节，接口注意事项新增 JENC 和 VENC 使用说明。

1 概述

AX SDK 在 SoC 软件解决方案中所处的层次位置如下图所示：

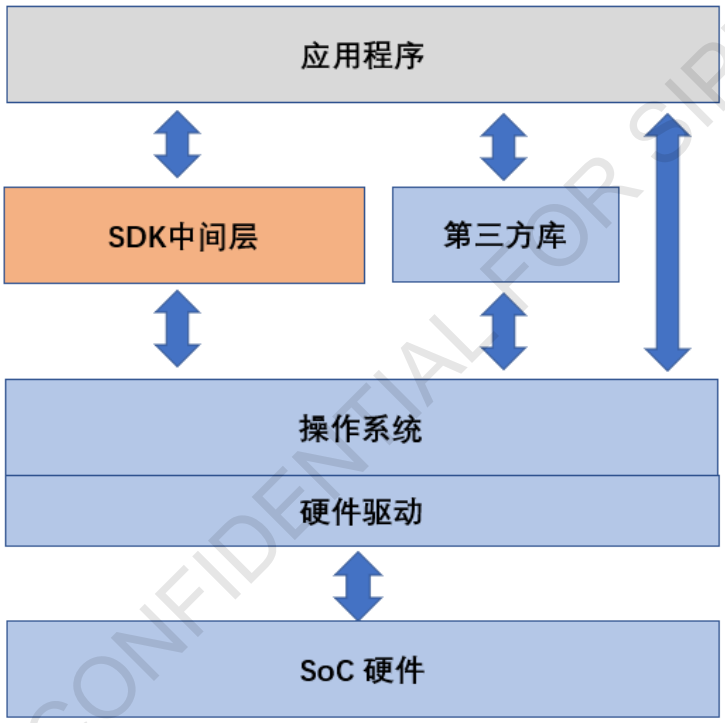

图1-1 SDK 层次示意图

AX SDK 在总体上可以分成系统、媒体和智能三个大的业务板块，其中系统业务的主要任务是提供操作系统功能，对 SoC 和外围硬件资源进行基本的管理，向应用层提供系统服务。系统业务的主要交付件是 U-boot，嵌入式 Linux 内核、文件系统和通用设备驱动，这几个组件习惯上合称为 BSP。智能业务的主要任务是向应用层提供面向图像内容理解和几何理解的支撑平台，主要交付件是 AI 深度学习框架和配套工具链、CV 算法库等。媒体业务的主要任务是向应用层提供关于音视频信号的采集、处理、显示、编解码、存储、传输等业务的支撑平台，上述每种功能一般会以一个子模块的形式提供，这些子模块集成到一起即构成 SDK 的**媒体子系统**，是媒体业务的主要交付件。

2 概念与术语

- AI，Artificial Intelligence，人工智能。
- BSP，Board Support Package，硬件支持包。
- CMM，Contiguous Memory Model，连续内存模型，SDK 媒体子系统提供的内存管理模型，为视频图像处理提供安全、便捷、高效的帧缓存服务。
- CV，Computer Vision，计算机视觉。
- MSS，Media Sub-System，媒体子系统，SDK 中主要负责处理音视频业务的子系统。
- Write-back，延迟写回模式，一种 Cache 工作模式。

3 系统控制模块

在媒体子系统中，系统控制模块是第一个核心功能单元，交付件包括 ax_sys.ko、ax_cmm.ko、ax_pool.ko。系统控制模块为媒体子系统下属的各个模块提供公共基础服务，包括但不限于内存管理、时间管理、日志管理、链路管理等功能，下面择要点加以描述。

3.1 内存管理

系统控制模块提供的内存管理服务主要体现了以下核心设计思想：

1. SoC 片外内存按业务方向（Linux，DSP，RTOS，Media 等）做静态划分，约定在顶层内存地址空间中为媒体业务划分一块专用的大块区域，称为 CMM 内存。
2. SDK 将 CMM 内存进一步划分成若干个分段（partition）。每个内存分段必须分配一个独特的名称，SDK 内部会通过名称识别不同的分段。
3. SDK 规定了内存划分的方法和规则，用户在规则框架内对内存分段的数量、位置、大小、名称等参数进行配置。
4. SDK 规定，用户配置的若干个内存分段中必须有一个名为“anonymous”的默认分段。当用户调用 API 申请创建缓存时，如果未指定分段名称，则实际在默认分段上分配。
5. 应用程序应按照配置方案使用各内存分段，原则上不应侵占为其它业务预留的内存。
6. SDK 支持应用程序从某个指定的内存分段中分配一块由用户自行管理的内存，称为任意用途内存，SDK 不假设其用途和使用方法。
7. SDK 支持应用程序从某个指定的内存分段中分配一块由 SDK 负责管理的内存，SDK 将该内存组织成池（Pool）和块（Block）两级结构，每个 Pool 由若干个大小相同的 Block 紧密排列而成，因此 Pool 的大小取决于 Block 的大小和数量的乘积，而 Block 的大小是在用户申请方案的基础上增加一些空间用于存储标签数据（metadata）。
8. Block 的典型用途是存储一帧图像数据，比如 RAW 或 YUV 格式的图像，但实际上也不排斥将其视为通用内存用于其它目的。

9. SDK 为每个 Block 分配一个全局唯一的标签（BlockID）用作识别 Block 的凭证，媒体子系统的各个模块均支持以 BlockID 为凭证访问 Block 所存储的图像数据和 metadata 数据。
10. SDK 提供 API 用于在指定的内存分段上创建 Pool 和 Block，具体参数由用户提供。
11. SDK 提供 API 用于报告 Block 的准确大小。
12. SDK 规定，在默认分段上创建的公共 Pool 是媒体子系统层次的公共资源，媒体子系统下属各个模块均可访问。当应用程序调用 API 申请 Block 时，如果未指明具体的 PoolID 以及分段名，则实际从默认分段上的公共 Pool 中找到一个大小匹配的 Block。在非默认分段也支持创建公共 Pool，如果指明了分段名但未指明具体的 PoolID，则实际从指定分段上的公共 Pool 中找到一个大小匹配的 Block。
13. SDK 规定，通过 CreatPool 接口创建的 Pool 为模块专用资源，仅为某一业务模块服务。专用 Pool 创建成功后得到 PoolID。当应用程序需要在专用 Pool 申请 Block 时，必须指明具体的 PoolID，否则将从指定分段上的公共 Pool 中分配。
14. 应用程序应尽量在媒体子系统启动数据流之前创建好维持数据流正常运转所需的各种 Pool。
15. 媒体子系统的内存管理提供两套接口，分别支持在 Linux 内核态和用户态对 Block 发起访问。内核态接口主要服务于 SDK 内部驱动，用户态接口可以支持应用程序。
16. SDK 提供 API 用于查询 BlockID 所属的 PoolID。
17. SDK 提供 API 用于将 BlockID 翻译成访问 Block 图像数据所需的物理地址和用户态虚拟地址，其中物理地址用于从 DSP、DMA 等设备发起访问，虚拟地址用于从 CPU 发起访问。
18. SDK 提供 API 用于将 BlockID 翻译成访问 Block metadata 所需的物理地址和用户态虚拟地址。

基于以上设计思想，AX SDK 的内存管理模型如下图所示：

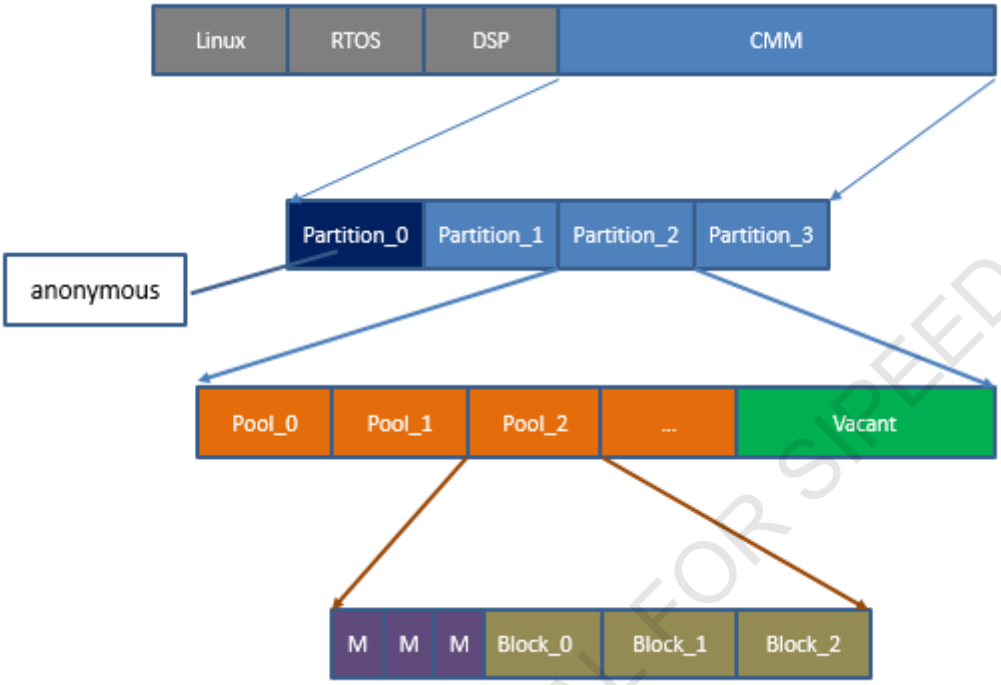

图3-1 SDK 内存池管理模型

其中，Partition 层次的设计需要在内存整体规划阶段确定，可在 auto_load_all_drv.sh 脚本中配置 CMM 内存起始地址和布局。Pool 和 Block 层次的设计根据媒体子系统内各个模块的实际需求确定，最终汇总形成媒体系统关于内存需求的整体方案。

应用程序在对媒体子系统进行内存初始化时，通过调用 API 申请创建由 SDK 负责管理的 Pool 和 Block。如果应用程序提交的内存方案与 Partition 层次的配置规划无冲突，则所有内存会一次性创建成功，否则会全部失败。

为了方便用户调试代码，减少代码编译次数，SDK 允许用户在使用脚本加载 ax_cmm.ko 时通过命令行指令创建若干个 Partition。用户需要保证 CMM 内存地址区域的有效性，不能与系统其他内存区域冲突。

3.2 内存优化

SDK 支持默认分段和专用分段，其中默认分段可以被媒体子系统下属的各个模块访问，使用比较方便，但也存在两个问题，一是会导致各种业务之间竞争公共缓存，导致性能上的不确定性；二是可能存在不同业务同时对空间上相邻的内存地址发起访问，容易产生较多的 DDR 访问冲突，降低访存性能。

内存优化的第一个任务是合理设计各种 Pool 所辖 Block 的数量和大小，争取用最少的内存支撑业务需求。此项优化的基本原理是，相比于每路码流固定分配若干个专用缓存的简单策略，令多路码流共享一个公共 Pool 可以提高缓存的使用效率，降低对缓存数量的需求。

媒体子系统各个模块均提供 Log 机制，用户可以根据 Log 信息观察分析内存分配和使用的效率情况，如发现某个模块存在较多的申请 Block 失败现象，则意味着对应的 Pool 中 Block 数量太少，应适当增加。反之，如果发现某个 Pool 总是存在较多的空闲 Block，则意味着该 Pool 中 Block 数量过多，可以适当缩减。通过这种方法可以使每个 Pool 都能达到最佳的利用率，此时的内存总需求就是当前性能下的最优解，如果进一步缩减缓存，则系统性能开始下降。

内存优化的第二个任务是合理设计各种内存分段的物理地址。用户需要了解 DDR 划分 bank 的机制，并分析媒体子系统各种业务访问 DDR 的时序特性，将存在竞争关系的业务分段尽可能分散配置到不同的 DDR bank 上，利用 DDR bank 支持并行访问的特性减少访问冲突，提高内存带宽利用率。

3.3 配置内存分段

平台将内存按照用途划分成若干个域，最常使用的两个域是 OS 域和 CMM 域，根据需要也可能使用到 DSP 域和 RTOS 域。OS 内存是指由 Linux 操作系统管理的内存；CMM 内存是由 CMM 驱动模块进行管理、专供媒体业务使用的内存。内存分段配置是在使用脚本加载 ax_cmm.ko 时对内存分段进行动态配置。

配置内存分段

1. CMM 管理组件是通过 ko 方式动态加载到 Linux kernel 中，加载时需要携带内存分段参数信息（CMM 支持若干个 Partition），其中 Partition 参数信息格式如下：

   cmmpool=partitonname,flag,phyaddr_start,partitionsize[:partitonname,flag,phyaddr_start,partitionsize]

   每个参数信息如下：

   a) partitonname：内存分段名称。

   b) flag：内存分段标识。用于扩展，暂时先不用，目前强制设置为 0。

c) phyaddr_start：内存分段 Partition 的起始物理地址（按 PAGE 4KB 对齐）。

d) partitionsize：内存分段 Partition 的大小（PAGE 4KB 的整数倍）；（**Partition 大小是以 M 为单位，不支持以 G 为单位**）。

e) []：是可选符号。

2. 系统启动后会自动加载 CMM 模块 ax_cmm.ko，用户可以根据需求修改 Partition 信息，比如添加或修改专用内存分段。加载 CMM 模块的脚本如下：
rootfs\rootfs\opt\scripts\auto_load_all_drv.sh。

以 AX620E Demo 板为例,在 CMM 中配置两个内存分段 Partition，一个是专用分段（Partition 名称为：**VENC**，物理首地址为：**0x80000000**，大小为：**256M**），另一个是匿名分段（Partition 名称为：**anonymous**，物理首地址为：**0x90000000**，大小为：**1792M**）。

```
#!/bin/sh
...
insmod /soc/ko/ax_cmm.ko
cmmpool=VENC,0,0x80000000,256M:anonymous,0,0x90000000,1792M
```

配置完成之后，可以通过如下命令检查是否起效：

```
# cat /proc/ax_proc/mem_cmm_info
--------------------SDK VERSION------------------
[Axera version]: ax_cmm V0.3.0 Aug 16 2023 10:38:08
+---PARTITION: Phys(0x90000000, 0xFFFFFFFF), Size=1835008KB(1792MB),    NAME="anonymous"
 nBlock(Max=15, Cur=15, New=15, Free=0)  nbytes(Max=33914880B(33120KB,32MB), Cur=33914880B(33120KB,32MB),
 New=33914880B(33120KB,32MB), Free=0B(0KB,0MB))  Block(Max=16588800B(16200KB,15MB), Min=4096B(4KB,0MB), Avg=41325B(40KB,0MB))
   |-Block: phys(0x90000000, 0x90FD1FFF), cache =non-cacheable, length=16200KB(15MB),    name="vo_vfb"
   |-Block: phys(0x90FD2000, 0x91FA3FFF), cache =non-cacheable, length=16200KB(15MB),    name="vo_vfb"
   |-Block: phys(0x91FA4000, 0x91FA7FFF), cache =non-cacheable, length=16KB(0MB),    name="vo_vfb"
   |-Block: phys(0x91FA8000, 0x91FABFFF), cache =non-cacheable, length=16KB(0MB),    name="VPP_DEV"
   |-Block: phys(0x91FAC000, 0x91FACFFF), cache =non-cacheable, length=4KB(0MB),    name="VPP_CMODE3"
   |-Block: phys(0x91FAD000, 0x91FADFFF), cache =non-cacheable, length=4KB(0MB),    name="GDC_CMDA3"
   |-Block: phys(0x91FAE000, 0x91FB0FFF), cache =non-cacheable, length=12KB(0MB),    name="GDC_CMD"
   |-Block: phys(0x91FB1000, 0x91FB4FFF), cache =non-cacheable, length=16KB(0MB),    name="TDP_DEV"
   |-Block: phys(0x91FB5000, 0x91FB5FFF), cache =non-cacheable, length=4KB(0MB),    name="TDP_CMODE3"
   |-Block: phys(0x91FB6000, 0x91FF5FFF), cache =non-cacheable, length=256KB(0MB),    name="venc_ko"
   |-Block: phys(0x91FF6000, 0x92035FFF), cache =non-cacheable, length=256KB(0MB),    name="venc_ko"
   |-Block: phys(0x92036000, 0x92036FFF), cache =non-cacheable, length=4KB(0MB),    name="venc_ko"
   |-Block: phys(0x92037000, 0x92046FFF), cache =non-cacheable, length=64KB(0MB),    name="jenc_ko"
   |-Block: phys(0x92047000, 0x92056FFF), cache =non-cacheable, length=64KB(0MB),    name="jenc_ko"
   |-Block: phys(0x92057000, 0x92057FFF), cache =non-cacheable, length=4KB(0MB),    name="jenc_ko"
+---PARTITION: Phys(0x80000000, 0x8FFFFFFF), Size=262144KB(256MB),    NAME="VENC"
 nBlock(Max=0, Cur=0, New=0, Free=0)  nbytes(Max=0B(0KB,0MB), Cur=0B(0KB,0MB), New=0B(0KB,0MB), Free=0B(0KB,0MB))
 Block(Max=0B(0KB,0MB), Min=0B(0KB,0MB), Avg=0B(0KB,0MB))

---CMM_USE_INFO:
 total size=2097152KB(2048MB),used=33120KB(32MB + 352KB),remain=2064032KB(2015MB + 672KB),partition_number=2,block_number=15
```

图3-2 配置内存分段

3.4 Cache 一致性管理

AX SoC 中使用的 Arm Cortex 系列处理器支持高速数据缓存（Cache）。由于启用 Cache 后 CPU 访问内存数据的效率会显著提高，所以 SDK 支持应用层程序将指定的 DDR 物理地址空间映射为 Cache Write-back 访问模式。在这种模式下，CPU 对很多内存数据的读写实际上是发生在 CPU 与 Cache 之间，并不会立即同步到 DDR 地址上，因此在一定时间内会存在 DDR 数据已经过期失效的情况，这种现象称为 Cache 一致性问题。如果在 DDR 数据失效期间 SoC 内部有其它硬件单元（如 DSP 或 DMA）恰好访问了失效数据，则会导致逻辑错误。

为了消除 Cache 一致性风险，SDK 提供了一组 API 用于显示地触发对 CPU Cache 的清空操作。当 Arm CPU 需要与 DMA 等硬件交换数据时，应用程序必须先调用相应的 API 执行 CacheFlush 操作，确保 Cache 中的内容已经同步到 DDR 中。

3.5 时间管理

AX SoC 为媒体子系统分配了一个专用的 64bit 计数器，芯片上电复位后即从零开始累加。SDK 提供 API 用于读取或设置该时间戳的值。该时间戳的溢出周期大于两万年，所以软件不需要考虑它的溢出问题。

3.6 日志管理

日志系统主要包括 2 部分：axklogd 和 axsyslogd。

axklogd

axklogd 主要作用是从/proc/ax_proc/kmsg 节点读取 Log 并转发给 axsyslogd，这部分 Log 是来自 SDK 中非开源 ko 驱动。

axsyslogd

axsyslogd 主要作用是接收 axklogd（内核态 log）或者其他应用发来的 Log（用户态 Log），并将 Log 统一存储到文件系统。

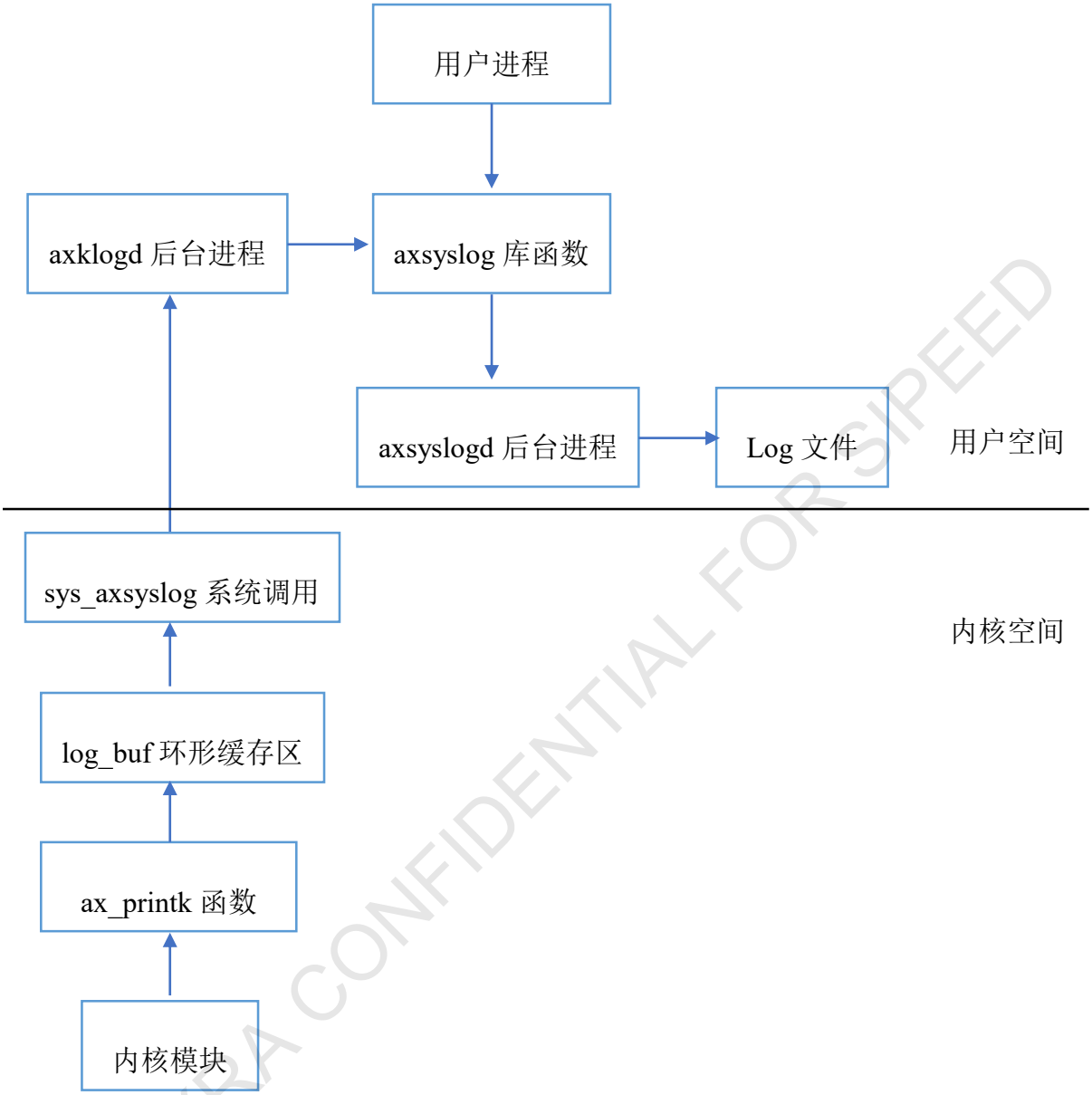

图3-3

3.7 链路管理

为了提高内存流转的效率，SDK 支持链路管理机制，允许媒体业务所使用的帧缓存按照用户预先配置好的路径进行自动流转，用户程序只需要在链路的最后出口端等待输出图像即可，中间各个环节的数据流转可以完全交给 SDK 负责，用户程序不需要干预，或者仅在用户认为必要的关键节点上进行接管。

SDK 提供链路绑定接口（AX_SYS_Link），即在数据源和数据接收者之间建立绑定关系。同一个数据源可以最多支持绑定 6 个接收者，但一个数据接收者同一时刻只能绑定一个数据源。SDK 同时提供了解除链路绑定的接口（AX_SYS_UnLink）。目前 SDK 支持的绑定关系如表 3-1 所示。

表3-1 SDK 支持的绑定关系

数据源	数据接收者
VIN	VO、VENC、IVPS、AVS
IVPS	VO、VENC、IVPS、VIN
VDEC	VO、VENC、IVPS
VO	VO、VENC、IVPS
AI	AENC、AO
ADEC	AO
AVS	VENC、IVPS

通过 cat /proc/ax_proc/link_table 命令查询当前系统建立的绑定关系：

```
/ # cat proc/ax_proc/link_table
--------------------Link Table--------------------
          Src                 |          Dst
(ModId    GrpId    ChnId)     | (ModId    GrpId    ChnId)
--------------------------------------------------
(VIN        0        2)  ->  (IVPS     2        0)
(VIN        0        1)  ->  (IVPS     1        0)
(VIN        0        0)  ->  (IVPS     0        0)
(IVPS       2        2)  ->  (VENC     0        6)
(IVPS       2        0)  ->  (VENC     0        4)
(IVPS       1        0)  ->  (VENC     0        3)
```

3.8 休眠唤醒

对于低功耗产品，由于设备供电能力有限，大部分时间需要处于休眠状态，而在有业务需求的时候需要能快速唤醒，切换到正常工作状态。因此 SDK 提供了一套完整的休眠唤醒机制。用户业务层可通过调用 AX_SYS_Sleep 接口发起进入休眠的请求。媒体子系统各模块收到休眠请求后，完成休眠准备工作并对相关硬件做下电操作，使系统进入休眠状态。SDK 支持通

过唤醒源的中断触发唤醒流程，用户在不需要重新初始化媒体子系统的情况下即可恢复业务层数据流。

4 API 简介

4.1 系统初始化

主要涉及下列 API：

- AX_SYS_Init：初始化媒体子系统的系统控制模块。
- AX_SYS_Deinit：去初始化媒体子系统的系统控制模块。

4.2 内存分配

关于内存分配主要涉及以下 API：

- AX_SYS_MemAlloc：从指定的 Partition 分配内存，不支持 cache。
- AX_SYS_MemAllocCached：从指定的 Partition 分配内存，支持 cache。
- AX_SYS_MemFree：释放通过 AX_SYS_MemAlloc 或 AX_SYS_MemAllocCached API 申请得到的内存。
- AX_SYS_MemGetBlockInfoByPhy：上层业务通过 BLOCK 的物理地址获取 BLOCK 的相关状态信息。
- AX_SYS_MemGetBlockInfoByVirt：上层业务通过 BLOCK 块内的用户态虚拟地址获取 BLOCK 的物理地址和映射类型。
- AX_SYS_MemGetPartitionInfo：获取 CMM 配置的分段状态信息。
- AX_SYS_Mmap：标准存储映射接口，根据用户指定 size 做映射，non-cache 类型。
- AX_SYS_MmapCache：标准存储映射接口，根据用户指定 size 做映射，cache 类型。
- AX_SYS_MmapFast：非标准存储映射接口，non-cache 类型。
- AX_SYS_MmapCacheFast：非标准存储映射接口，cache 类型。

- AX_SYS_Munmap：存储反映射接口。
- AX_SYS_MflushCache：将 CPU cache 里的内容刷新到 DDR 内存，同时清空 cache。
- AX_SYS_MinvalidateCache：将 CPU cache 状态置为无效。
- AX_SYS_MemSetConfig：配置内存参数，指定模块使用内存的分段名。
- AX_SYS_MemGetConfig：获取模块使用内存的分段名。
- AX_SYS_MemSetParam：配置内存参数，指定模块使用内存的分段名，可支持配置多个分段名。
- AX_SYS_MemGetParam：获取模块使用内存的分段名。
- AX_SYS_MemQueryStatus：获取 CMM 内存当前使用状态。
- AX_SYS_MemGetMaxFreeRegionInfo：获取 CMM 最大连续空闲区域的起始物理地址和大小。

4.3 缓存池管理

关于缓存池管理主要涉及以下 API：

- AX_POOL_SetConfig：设置媒体子系统缓存池方案。
- AX_POOL_GetConfig：获取媒体子系统缓存池方案。
- AX_POOL_Init：按照预定方案创建媒体子系统缓存池。
- AX_POOL_Exit：释放所有已创建的媒体子系统缓存池。
- AX_POOL_CreatePool：创建一个视频缓存池。
- AX_POOL_DestroyPool：销毁一个视频缓存池。
- AX_POOL_GetBlock：从指定的缓存池中借用一个缓存块。
- AX_POOL_ReleaseBlock：归还一个借用的缓存块。
- AX_POOL_Handle2PhysAddr：已知缓存块的 BlockID，查找对应的物理地址。
- AX_POOL_PhysAddr2Handle：已知缓存块的物理地址，查找对应的 BlockID。

- AX_POOL_Handle2MetaPhysAddr：已知缓存块的 BlockID，查找对应的 Metadata 物理地址。
- AX_POOL_Handle2PoolId：已知缓存块的 BlockID，查找对应的 PoolID。
- AX_POOL_Handle2BlkSize：已知缓存块的 BlockID，查找对应的缓存块大小。
- AX_POOL_GetBlockVirAddr：已知缓存块的 BlockID，查找对应的用户态虚拟地址。
- AX_POOL_GetMetaVirAddr：已知缓存块的 BlockID，查找对应的 Metadata 用户态虚拟地址。
- AX_POOL_MmapPool：为一个视频缓存池映射用户态虚拟地址。
- AX_POOL_MunmapPool：为一个视频缓存池解除用户态映射。
- AX_POOL_IncreaseRefCnt：已知缓存块的 BlockID，对其引用计数进行加 1 操作。
- AX_POOL_DecreaseRefCnt：已知缓存块的 BlockID，对其引用计数进行减 1 操作。
- AX_POOL_FreeCommPool：只释放公共缓存池内存。

4.4 时间管理

关于时间管理主要涉及以下 API：

- AX_SYS_GetCurPTS：读取媒体时间戳的值，单位微秒。
- AX_SYS_InitPTSBase：设置媒体时间戳基准，单位微秒。
- AX_SYS_SyncPTS：同步媒体时间戳，支持微秒级微调。

4.5 日志管理

关于日志管理主要涉及以下 API：

- AX_SYS_LogPrint：log 输出函数。
- AX_SYS_LogOutput：log 输出函数，可指定输出目的地。
- AX_SYS_SetLogLevel：该接口无效，设置日志等级请参考《00 - AX SDK 使用说

明.docx》文档中“日志系统”章节。

- AX_SYS_SetLogTarget：该接口无效，设置日志输出目的地请参考《00 - AX SDK 使用说明.docx》文档中“日志系统”章节。
- AX_SYS_EnableTimestamp：该接口无效，使能日志时间戳请参考《00 - AX SDK 使用说明.docx》文档中“日志系统”章节。

4.6 链路管理

- AX_SYS_Link：数据源和数据接收者建立绑定关系接口。
- AX_SYS_UnLink：数据源和数据接收者解除绑定关系接口。
- AX_SYS_GetLinkByDest：获取数据接收者绑定的数据源信息。
- AX_SYS_GetLinkBySrc：获取数据源绑定的所有数据接收者信息集合。
- AX_SYS_SetVINIVPSMode：设置 VIN 和 IVPS 工作模式。

4.7 芯片类型管理

- AX_SYS_GetChipType：获取当前芯片类型接口。

4.8 休眠唤醒

- AX_SYS_Sleep：发起系统进入休眠的请求接口。
- AX_SYS_WakeLock：用户态获取 WakeLock 锁。
- AX_SYS_WakeUnlock：用户态释放 WakeLock 锁。
- AX_SYS_RegisterEventCb：休眠唤醒事件回调函数注册接口。
- AX_SYS_UnregisterEventCb：休眠唤醒事件回调函数注销接口。
- AX_SYS_SleepTimeStamp：休眠唤醒时间统计接口。

5 API 定义

5.1 系统初始化 API

AX_SYS_Init

【描述】

对媒体子系统的系统控制模块进行初始化。

【语法】

```
AX_S32 AX_SYS_Init(AX_VOID);
```

【参数】

参数名称	描述	输入/输出
NULL		NULL

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 由于媒体子系统的运行依赖于 SYS 模块，因此需要先调用该接口初始化。
- 对于单进程，允许多次调用，仍会返回成功，对媒体子系统运行没有影响。
- 对于多进程，各进程需要分别调用该接口。
- SDK 也允许用户关闭多进程特性，可通过 mkdir /tmp/disable_multi_task 命令创建该节点，此时 SYS 模块只允许单进程调用，如果多个进程调用 AX_SYS_Init，将返回错误码

AX_ERR_SYS_NOT_PERM。

AX_SYS_Deinit

【描述】

对媒体子系统的系统控制模块进行反初始化。

【语法】

```
AX_S32 AX_SYS_Deinit(AX_VOID);
```

【参数】

参数名称	描述	输入/输出
NULL		NULL

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 由于媒体子系统的运行依赖于 SYS 模块，因此需要在媒体子系统业务结束之后才能调用此接口。
- 对于单进程，允许多次调用，仍然返回成功，对媒体子系统运行没有影响。
- 对于多进程，各进程需要分别调用该接口。

5.2 内存分配 API

AX_SYS_MemAlloc

【描述】

从指定的内存分段中申请任意用途内存，不启用 cache。

【语法】

```
AX_S32 AX_SYS_MemAlloc(AX_U64 *phyaddr, AX_VOID **pviraddr, AX_U32 size,
AX_U32 align, AX_S8 *token);
```

【参数】

参数名称	描述	输入/输出
phyaddr	API 接口返回申请到连续物理地址内存的起始物理地址	输出
pviraddr	API 接口返回申请到连续物理地址内存的起始虚拟地址，兼容两种使用方式： 1. 传入的 pviraddr 参数为 NULL，则默认不做虚拟地址映射。当用户不需要接口返回虚拟地址时，可以采用此方式，节省虚拟地址空间资源。 2. 传入的 pviraddr 参数不为 NULL，则接口内部做虚拟地址映射，带出虚拟地址。	输出
size	申请连续物理地址内存的大小	输入
align	申请连续物理地址内存的对齐方式。 ☞ 备注：目前 align 参数不生效，SDK 默认始终按 4KB 对齐	输入
token	Token 分两种情况： 1. 一种是向专用分段申请内存，此时 token 包含两个信息，分别是专用 Partition 名称和 Block 名称，格式为 Partitionname:Blockname（中间用冒号隔开）。 2. 另一种是向匿名分段申请内存，token 只需要包含 Blockname 即可。 ☞ 备注：	输入

参数名称	描述	输入/输出
	● Partitionname 名称必须与加载 CMM 模块参数的 partition 名称一致，而且对大小写敏感； ● token 长度不要超过 30 个字节； ● Blockname 名称应有意义，便于 debug。	

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 当 AX_SYS_MemAlloc 以不映射虚拟地址的方式使用时，对应的 AX_SYS_MemFree 第二个参数 pviraddr 也传入 NULL 即可。

AX_SYS_MemAllocCached

【描述】

从指定的内存分段中申请任意用途内存，映射为 Cache 模式。

【语法】

```
AX_S32 AX_SYS_MemAllocCached(AX_U64 *phyaddr, AX_VOID **pviraddr, AX_U32
size, AX_U32 align, AX_S8 *token);
```

【参数】

参数名称	描述	输入/输出
phyaddr	API 接口返回申请到连续物理地址 cache 内存的起始物理地址	输出
pviraddr	API 接口返回申请到连续物理地址内存 cache 的起始虚拟地址	输出
size	申请连续物理地址 cache 内存的大小	输入
align	申请连续物理地址 cache 内存的对齐方式	输入
token	长度不超过 30 字节的字符串指针 Token 的用法分两种情况： 1. 一种是向专用分段申请内存，此时 token 包含两个信息，分别是专用 Partition 名称和 Block 名称，格式为 Partitionname:Blockname（中间用冒号隔开）。 2. 另一种是向通用分段（即匿名分段）申请内存，token 只需要包含 blockname 即可。 ☞ 备注： ● Partitionname 名称必须与加载 CMM 模块参数的 partition 名称一致，而且对大小写敏感； ● token 长度不要超过 30 个字节； ● Blockname 名称应有意义，便于 debug。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

AX_SYS_MemFree

【描述】

释放通过 MemAlloc 或 MemAllocCached API 申请得到的内存。

【语法】

```
AX_S32 AX_SYS_MemFree(AX_U64 phyaddr, AX_VOID *pviraddr);
```

【参数】

参数名称	描述	输入/输出
phyaddr	API 接口返回申请到连续物理地址 cache 内存的起始物理地址	输入
Pviraddr	API 接口返回申请到连续物理地址内存 cache 的起始虚拟地址	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

AX_SYS_MemGetBlockInfoByPhy

【描述】

上层业务通过 BLOCK 的物理地址获取 BLOCK 的相关状态信息。

【语法】

```
AX_S32 AX_SYS_MemGetBlockInfoByPhy(AX_U64 phyaddr, AX_S32 *pmemType, AX_VOID **pviraddr, AX_U32 *pblockSize);
```

【参数】

参数名称	描述	输入/输出
phyaddr	BLOCK 的物理地址（Block 块内的任意物理地址）	输入
pmemType	BLOCK 内存类型（cache or non-cache），其中 2 表示 cache 类型，4 表示 non-cache 类型 2 => AX_MEM_CACHED 4 => AX_MEM_NONCACHED	输出
pviraddr	BLOCK 用户态的虚拟地址	输出
pblockSize	BLOCK 长度	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 如果传入的 phyaddr 是 Block 内的物理地址，即相对于 Block 申请时的起始物理地址带偏移，则返回的 pviraddr 也带偏移。返回的 pblockSize 会自动减去偏移量。
- 输入的物理地址，必须来自于 AX_SYS_MemAlloc 和 AX_SYS_MemAllocCached 接口。
- 缓存池中的地址不支持此接口。

AX_SYS_MemGetBlockInfoByVirt

【描述】

上层业务通过 BLOCK 块内的用户态虚拟地址获取 BLOCK 的物理地址和映射类型。

【语法】

```
AX_S32 AX_SYS_MemGetBlockInfoByVirt(AX_VOID *pviraddr, AX_U64 *phyaddr,
AX_S32 *pmemType);
```

【参数】

参数名称	描述	输入/输出
pviraddr	BLOCK 的虚拟地址（Block 内的任意一个虚拟地址）	输入
phyaddr	BLOCK 的物理地址	输出
pmemType	BLOCK 内存类型（cache or non-cache），其中 2 表示 cache 类型，4 表示 non-cache 类型 2 => AX_MEM_CACHED 4 => AX_MEM_NONCACHED	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 如果传入的 pviraddr 是 Block 内的虚拟地址，即相对于 Block 申请时的起始虚拟地址带偏移，则返回的 phyaddr 也带偏移。
- 输入的虚拟物址，必须来自于 AX_SYS_MemAlloc 和 AX_SYS_MemAllocCached 接口。
- 缓存池中的地址不支持此接口。
- AX_SYS_Mmap 和 AX_SYS_MmapCache 映射的虚拟地址，不支持此接口。

AX_SYS_MemGetPartitionInfo

【描述】

获取 CMM 内存配置的分段状态信息。

【语法】

```
AX_S32 AX_SYS_MemGetPartitionInfo (AX_CMM_PARTITION_INFO_T
*pCmmPartitionInfo);
```

【参数】

参数名称	描述	输入/输出
pCmmPartitionInfo	系统当前 CMM 分段配置信息描述结构体，包括每个分段名称、起始物理地址、分段大小（以 KB 为单位）	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

无

AX_SYS_Mmap

【描述】

标准存储映射接口，根据用户传入的 size 做映射，non-cache 类型。

【语法】

```
AX_VOID * AX_SYS_Mmap (AX_U64 phyaddr, AX_U32 size);
```

【参数】

参数名称	描述	输入/输出
phyaddr	需映射的内存单元起始物理地址	输入
size	映射的字节数，不能为 0	输入

【返回值】

返回值	描述
0	无效地址
非 0	有效虚拟地址

【注意】

- 操作的节点是 dev/ax_cmm，在 CMM 中实现映射。
- 支持托管模式和非托管模式，输入的地址必须是 DDR 内存范围内的合法物理地址。
- 托管模式：指传入的物理地址来自于 CMM 或者内存池接口，物理地址会被 SDK 管理。
- 非托管模式：指传入的物理地址不需要经过 CMM 或内存池接口调用，只要是 DDR 内存范围内的合法物理地址即可。
- 根据用户传入的物理地址和大小做映射。
- 对同一物理地址每次调用该接口，内部均会做虚拟地址映射。
- 对于 AX_SYS_MemAlloc 和 AX_SYS_MemAllocCached 返回的物理地址，调用此接口映射会得到新的虚拟地址，相当于对同一个物理地址做二次映射。AX_SYS_MemAlloc 和

AX_SYS_MemAllocCached 本身返回的虚拟地址不受影响。

AX_SYS_MmapCache

【描述】

标准存储映射接口，根据用户传入的 size 做映射，cache 类型。

【语法】

```
AX_VOID * AX_SYS_MmapCache (AX_U64 phyaddr, AX_U32 size);
```

【参数】

参数名称	描述	输入/输出
phyaddr	需映射的内存单元起始物理地址	输入
size	映射的字节数，不能为 0	输入

【返回值】

返回值	描述
0	无效地址
非 0	有效虚拟地址

【注意】

- 操作的节点是 dev/ax_cmm，在 CMM 中实现映射。
- 支持托管模式和非托管模式，输入的地址必须是 DDR 内存范围内的合法物理地址。
- 托管模式：指传入的物理地址来自于 CMM 或者内存池接口，物理地址会被 SDK 管理。
- 非托管模式：指传入的物理地址不需要经过 CMM 或内存池接口调用，只要是 DDR 内存范围内的合法物理地址即可。
- 根据用户传入的物理地址和大小做映射。
- 对同一物理地址每次调用该接口，内部均会做虚拟地址映射。
- 对于 AX_SYS_MemAlloc 和 AX_SYS_MemAllocCached 返回的物理地址，调用此接口映射会得到新的虚拟地址，相当于对同一个物理地址做二次映射。AX_SYS_MemAlloc 和

AX_SYS_MemAllocCached 本身返回的虚拟地址不受影响。

AX_SYS_MmapFast

【描述】

非标准存储映射接口，non-cache 类型。

【语法】

```
AX_VOID * AX_SYS_MmapFast (AX_U64 phyaddr, AX_U32 size);
```

【参数】

参数名称	描述	输入/输出
phyaddr	需映射的内存单元起始物理地址	输入
size	映射的字节数，不能为 0	输入

【返回值】

返回值	描述
0	无效地址
非 0	有效虚拟地址

【注意】

- 操作的节点是 dev/ax_cmm，在 CMM 中实现映射。
- 支持托管模式和非托管模式，输入的地址必须是 DDR 内存范围内的合法物理地址。
- 托管模式：指传入的物理地址来自于 CMM 或者内存池接口，物理地址会被 SDK 管理。此时用户传入的 size 不起作用，实际映射大小等于内存申请时分配的大小。如果地址已经映射过，则不重复映射。通过牺牲虚拟地址空间资源换取执行速度。
- 非托管模式：指传入的物理地址不需要经过 CMM 或内存池接口调用，只要是 DDR 内存范围内的合法物理地址即可。此时根据用户传入的 size 做映射，每次调用都做映射。
- 对于 AX_SYS_MemAlloc 和 AX_SYS_MemAllocCached 返回的物理地址，调用此接口映射会得到新的虚拟地址，相当于对同一个物理地址做二次映射。AX_SYS_MemAlloc 和 AX_SYS_MemAllocCached 本身返回的虚拟地址不受影响。

AX_SYS_MmapCacheFast

【描述】

非标准存储映射接口，non-cache 类型。

【语法】

```
AX_VOID * AX_SYS_MmapCacheFast (AX_U64 phyaddr, AX_U32 size);
```

【参数】

参数名称	描述	输入/输出
phyaddr	需映射的内存单元起始物理地址	输入
size	映射的字节数，不能为 0	输入

【返回值】

返回值	描述
0	无效地址
非 0	有效虚拟地址

【注意】

- 操作的节点是 dev/ax_cmm，在 CMM 中实现映射。
- 支持托管模式和非托管模式，输入的地址必须是 DDR 内存范围内的合法物理地址。
- 托管模式：指传入的物理地址来自于 CMM 或者内存池接口，物理地址会被 SDK 管理。此时用户传入的 size 不起作用，实际映射大小等于内存申请时分配的大小。如果地址已经映射过，则不重复映射。通过牺牲虚拟地址空间资源换取执行速度。
- 非托管模式：指传入的物理地址不需要经过 CMM 或内存池接口调用，只要是 DDR 内存范围内的合法物理地址即可。此时根据用户传入的 size 做映射，每次调用都做映射。
- 对于 AX_SYS_MemAlloc 和 AX_SYS_MemAllocCached 返回的物理地址，调用此接口映射会得到新的虚拟地址，相当于对同一个物理地址做二次映射。AX_SYS_MemAlloc 和 AX_SYS_MemAllocCached 本身返回的虚拟地址不受影响。

AX_SYS_Munmap

【描述】

内存反映射接口。

【语法】

```
AX_S32 AX_SYS_Munmap (AX_VOID* pviraddr, AX_U32 size);
```

【参数】

参数名称	描述	输入/输出
pviraddr	需要反映射的虚拟地址，mmap 后返回的地址	输入
size	映射区的字节长度，不能为 0	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 输入的虚拟地址必须是 AX_SYS_Mmap、AX_SYS_MmapCache、AX_SYS_MmapFast 和 AX_SYS_MmapCacheFast 接口返回的虚拟地址。
- 对于 AX_SYS_MemAlloc 和 AX_SYS_MemAllocCached 接口返回的虚拟地址，不能调用此接口反映射。
- 不允许做分段 unmap 操作。因为 map 和 unmap 操作对传入的地址和大小都有对齐要求，如果 map 一个大的 buffer，然后分段 unmap，很可能导致虚拟地址无效，内存访问异常。

AX_SYS_MflushCache

【描述】

将 CPU cache 里的内容刷新到 DDR 内存，同时清空 cache。

【语法】

```
AX_S32 AX_SYS_MflushCache (AX_U64 phyaddr, AX_VOID *pviraddr, AX_U32 size);
```

【参数】

参数名称	描述	输入/输出
phyaddr	待操作内存的起始物理地址	输入
pviraddr	待操作内存的起始虚拟地址	输入
size	待操作内存的大小，不能为 0	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 当 cache 里的有最新数据时，为了保证不能直接访问 cache 的硬件在访问内存是能够得到正确的数据，此时需要先调用此接口将 cache 里的内容刷新到内存。这样当硬件访问内存时，保证了数据的一致性和正确性。
- 此接口可以与 AX_SYS_MmapCache 或 AX_SYS_MemAllocCached 接口配套使用。
- 输入的虚拟地址必须是物理地址映射的地址，且传入的 size 不能超过虚拟地址映射范围，否则会返回失败。
- 用户需要确保传入参数的合法性。

AX_SYS_MinvalidateCache

【描述】

触发 Cache Invalidate 操作，在 CPU 从 DDR 地址读共享数据前使用。

【语法】

```
AX_S32 AX_SYS_MinvalidateCache(AX_U64 phyaddr, AX_VOID *pviraddr, AX_U32
size);
```

【参数】

参数名称	描述	输入/输出
phyaddr	待操作内存的起始物理地址	输入
pviraddr	待操作内存的起始虚拟地址	输入
size	待操作内存的大小，不能为 0	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 在通过 CPU 读取内存数据时，需要先调用此接口使 cache 无效，确保读取的一定是内存上的数据。
- 此接口可以与 AX_SYS_MmapCache 或 AX_SYS_MemAllocCached 接口配套使用。
- 用户需要确保传入参数的合法性。

AX_SYS_MemSetConfig

【描述】

配置内存参数，指定模块使用内存的分段名。

【语法】

```
AX_S32 AX_SYS_MemSetConfig (const AX_MOD_INFO_T *pModInfo,const AX_S8
*pPartitionName);
```

【参数】

参数名称	描述	输入/输出
pModInfo	模块信息结构体	输入
pPartitionName	内存所在分段名	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 输入的分段名必须是 ax_cmm.ko 模块参数中存在的。
- 若未调用此接口或者输入的分段名是 NULL，则默认从匿名分段分配。
- 结构体实参使用前最好先清 0，避免随机值影响。
- AX_MOD_INFO_T 说明见《50 - AX 公共数据结构文档》。
- 模块使用说明：
  - VDEC 模块：
    - 参数 enModId 设置为 AX_ID_VDEC，通过 s32GrpId 区分不同解码组，s32ChnId 需设置为 0。

- 单帧接口内存始终在匿名分段申请，不支持配置。

- VO 模块：
  - 参数 enModId 设置为 AX_ID_VO，通过 s32GrpId 区分不同 VO layer，s32ChnId 需设置为 0。
  - 如果客户有显示业务，除了根据 layer 使用情况调用 AX_SYS_MemSetConfig 外，还需另外增加 s32GrpId=30，s32ChnId=0，调用 AX_SYS_MemSetConfig。

- IVPS 模块：
  - ivps 私有 pool 支持从指定 partition 获取内存。配置方法：
    - group 上的私有 pool:首先调用 AX_IVPS_SetGrpPoolAttr 设置使用 private pool 属性；调用 AX_SYS_MemSetConfig，设置 enModId=AX_ID_IVPS，s32GrpId=x， s32ChnId=0；
    - channel 上的私有 pool:首先调用 AX_IVPS_SetChnPoolAttr 设置使用 private pool 属性；调用 AX_SYS_MemSetConfig，设置 enModId=AX_ID_IVPS，s32GrpId=x， s32ChnId=x；
    - group 上的 MemSetConfig 配置与当前 group 上的第一个 channel（s32ChnId=0）的配置相同，即共享同一块 partition。
  - ivps 的 rgn 画 OSD 使用的内存，支持从指定 partition 获取内存。配置方法：调用 AX_SYS_MemSetConfig，设置 enModId=AX_ID_IVPS， s32GrpId=0，s32ChnId=0。
  - 加载 ko 申请的内存不可指定，如 CDMA 申请的内存。从 anonymous 匿名分段获取。
  - 其他的 IVPS cmm 内存，如 GDC_MESH, 若需指定 partition, 可通过 AX_SYS_MemSetConfig，设置 enModId=AX_ID_IVPS， s32GrpId=0，s32ChnId=0。

AX_SYS_MemGetConfig

【描述】

获取模块使用的内存分段名。

【语法】

```
AX_S32 AX_SYS_MemGetConfig (const AX_MOD_INFO_T *pModInfo, AX_S8
*pPartitionName);
```

【参数】

参数名称	描述	输入/输出
pModInfo	模块信息结构体	输入
pPartitionName	返回内存所在分段名	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 结构体实参使用前最好先清 0，避免随机值影响。

AX_SYS_MemSetParam

【描述】

配置内存参数，指定模块使用内存的分段名，支持多个分段名。

【语法】

```
AX_S32 AX_SYS_MemSetParam (const AX_MOD_INFO_T *pModInfo,const AX_S8
*pPartitionName, AX_U8 type);
```

【参数】

参数名称	描述	输入/输出
pModInfo	模块信息结构体	输入
pPartitionName	内存所在分段名	输入
type	模块内部内存具体分类	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 输入的分段名必须是 ax_cmm.ko 模块参数中存在的。
- 若未调用此接口或者输入的分段名是 NULL，则默认从匿名分段分配。
- 结构体实参使用前最好先清 0，避免随机值影响。
- AX_MOD_INFO_T 说明见《50 - AX 公共数据结构文档》。
- type 参数为 0 时，效果等同于 AX_SYS_MemSetConfig 接口。
- type 参数目前最大支持 2，具体实现取决于各模块。
- type 参数目前只支持 VENC、VIN、NPU 模块，具体使用说明如下：

- VENC 模块：
  - 通道 stream buffer（即 token 字段为 venc_fifo_xxx），有两种方式都可以指定 partition：

  方式一：旧接口 AX_SYS_MemSetConfig。

  方式二：新接口 AX_SYS_MemSetParam，其中参数 type 需设置为 0。

  - 通道其他内存（即 token 字段为 venc_ewl_xxx），参数 type 需设置为 1。

  - 用户调用 AX_SYS_MemSetConfig 或 AX_SYS_MemSetParam 时，结构体中 enModId 需设置为 AX_ID_VENC，s32GrpId 需设置为 0，不同通道用 s32ChnId 区分。

- VIN 模块：
  - 对于 isp_pool* 相关的内存，不区分 pipe ，AX_SYS_MemSetParam 配置内存。其中参数 type 设置为 1，参数 enModId 配置 AX_ID_VIN，s32GrpId/s32ChnId 配置 0。
  - 对于其他 VIN 模块 CMM 内存，需要区分 pipe，AX_SYS_MemSetParam 配置内存。其中参数 type 设置为 0， 参数 enModId 配置 AX_ID_VIN，s32GrpId 配置为 pipe id，s32ChnId 配置 0。
  - IFE RAW Buffer 仍然从匿名端申请，保持不变。
- NPU 模块：
  - type=0 为 AISIP 模型内存设置分段，type=1 为算法模型内存设置分段，其中参数 enModId 配置 AX_ID_NPU，s32GrpId 和 s32ChnId 需要设置为 0。

AX_SYS_MemGetParam

【描述】

获取模块使用的内存分段名。

【语法】

```
AX_S32 AX_SYS_MemGetParam (const AX_MOD_INFO_T *pModInfo, AX_S8
*pPartitionName, AX_U8 type);
```

【参数】

参数名称	描述	输入/输出
pModInfo	模块信息结构体	输入
pPartitionName	返回内存所在分段名	输出
type	模块内部内存具体分类	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

结构体实参使用前最好先清 0，避免随机值影响。

AX_SYS_MemQueryStatus

【描述】

获取 CMM 内存当前使用状态。

【语法】

```
AX_S32 AX_SYS_MemQueryStatus (AX_CMM_STATUS_T *pCmmStatus);
```

【参数】

参数名称	描述	输入/输出
pCmmStatus	CMM 内存状态结构体	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 状态信息详见 AX_CMM_STATUS_T 结构体描述。

AX_SYS_MemGetMaxFreeRegionInfo

【描述】

获取 CMM 最大连续空闲区域的起始物理地址和大小。

【语法】

```
AX_S32 AX_SYS_MemGetMaxFreeRegionInfo (AX_U64 *pPhyaddr, AX_U32 *pSize);
```

【参数】

参数名称	描述	输入/输出
pPhyaddr	CMM 最大连续空闲区域的起始物理地址	输出
pSize	CMM 最大连续空闲区域的大小	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 这片连续区域可以是跨分段的。即如果两个连续分段都是空闲的，则会被认为是一个大的连续区域，返回的大小是两个分段合并后的总大小。
- 此 API 如果执行失败，起始物理地址和大小均将返回 0，并返回非 0 错误码。
- 此 API 如果执行成功，将返回非 0 的起始物理地址和大小。用户可通过 AX_SYS_Mmap/AX_SYS_MmapCache 接口对地址做映射，得到虚拟地址，进而对内存进行读写操作。
- 此 API 返回的状态是瞬时的，在多线程和多进程场景 CMM 内存分布是实时变化的，空闲区域状态也实时变化。用户在操作空闲区域内存时，需确保没有其他地方在同时操作该内存，否则出现不可预期行为。

5.3 缓存池管理 API

AX_POOL_SetConfig

【描述】

设置媒体子系统缓存池方案。

【语法】

```
AX_S32 AX_POOL_SetConfig(const AX_POOL_FLOORPLAN_T *pPoolFloorPlan);
```

【参数】

参数名称	描述	输入/输出
pPoolFloorPlan	媒体子系统缓存池配置指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

- 结构体实参使用前建议先清 0，避免随机值影响。
- 此 API 成功后，允许重复调用，后续调用会覆盖前面的设置。
- 调用 AX_POOL_Init 成功后，如再调用 AX_POOL_SetConfig 将返回失败。
- 公共缓存池的配置请参考 AX_POOL_FLOORPLAN_T 结构体中的描述。

AX_POOL_GetConfig

【描述】

获取媒体子系统缓存池方案。

【语法】

```
AX_S32 AX_POOL_GetConfig(AX_POOL_FLOORPLAN_T *pPoolFloorPlan);
```

【参数】

参数名称	描述	输入/输出
pPoolFloorPlan	媒体子系统缓存池配置指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

- 结构体实参使用前建议先清 0，避免随机值影响。

AX_POOL_Init

【描述】

按照预定方案创建媒体子系统缓存池。

【语法】

```
AX_S32 AX_POOL_Init(AX_VOID);
```

【参数】

参数名称	描述	输入/输出
NULL		NULL

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 必须先调用 AX_POOL_SetConfig。
- 单进程或多进程场景下，此 API 成功后，如再调用 AX_POOL_SetConfig 将返回失败。
- 单进程或多进程场景下，此 API 成功后，允许重复调用，不返回失败，不执行实际操作。

AX_POOL_Exit

【描述】

释放所有已创建的媒体子系统缓存池。

【语法】

```
AX_S32 AX_POOL_Exit(AX_VOID);
```

【参数】

参数名称	描述	输入/输出
NULL		NULL

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 此 API 成功后，允许再次调用，不返回错误，也不执行实际操作。
- 此 API 成功后，不会清除 AX_POOL_SetConfig 的配置。
- 单进程场景，建议应用程序启动后首先调用此 API，清除可能存在的系统残留。
- 此 API 会回收所有缓存池内存，包括公共缓存池和 AX_POOL_CreatePool 创建的缓存池。
- 多进程场景只允许一个进程调用此 API，用户需要保证所有进程都不再使用缓存池时才能调用，否则返回失败。

AX_POOL_CreatePool

【描述】

创建一个视频缓存池。

【语法】

```
AX_POOL AX_POOL_CreatePool(AX_POOL_CONFIG_T *pPoolConfig);
```

【参数】

参数名称	描述	输入/输出
pPoolConfig	缓存池配置指针	输入

【返回值】

返回值	描述
非 AX_INVALID_POOLID	成功，有效的缓存池 ID
AX_INVALID_POOLID	创建缓存池失败。可能是参数非法或者内存不够

【注意】

- 此 API 可独立运行，不依赖于 AX_POOL_SetConfig->AX_POOL_Init 流程。
- AX_POOL_CONFIG_T 结构体中的 IsMergeMode 属性如果为 AX_TRUE，则逻辑上将这个缓存池视为公共缓存池的一部分，称为扩展缓存池。如果公共缓存池没有足够 Block，则去扩展缓存池尝试获取。
- 结构体实参使用前建议先清 0，避免随机值影响。

AX_POOL_DestroyPool

【描述】

将一个由 CreatePool API 创建的视频缓存池销毁，并实际回收内存。

【语法】

```
AX_S32 AX_POOL_DestroyPool(AX_POOL PoolId);
```

【参数】

参数名称	描述	输入/输出
PoolId	缓存池 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 与 AX_POOL_CreatePool 成对使用。
- 此 API 执行成功后即解除与之有关的虚拟内存映射，并释放物理内存。
- 此 API 调用前，需保证所有缓存块都已归还，否则销毁失败，返回 BUSY 错误码。
- 不建议频繁创建和销毁缓存池，避免内存碎片化。

AX_POOL_GetBlock

【描述】

从指定的缓存池中借用一个缓存块。

【语法】

```
AX_BLK AX_POOL_GetBlock(AX_POOL PoolId, AX_U64 BlkSize,const AX_S8
*pPartitionName);
```

【参数】

参数名称	描述	输入/输出
PoolId	缓存池 ID； 如果将 PoolId 设为 AX_INVALID_POOLID，则表示从任意一个公共缓存池中获取缓存块	输入
BlkSize	缓存块大小，以字节为单位	输入
pPartitionName	缓存池所在的分段 Partition 名字； 如果 pPartitionName 等于 NULL，则表示从匿名分段上的公共缓存池获取缓存块	输入

【返回值】

返回值	描述
非 AX_INVALID_BLOCKID	成功，有效的缓存块 ID
AX_INVALID_BLOCKID	失败

【注意】

- 需要与 AX_POOL_ReleaseBlock 成对使用，否则会造成缓存块被 AX_ID_USER 占用。
- 此接口执行成功，会返回有效的缓存块 ID，并且默认内部会对该缓存块做 AX_ID_USER 引用计数加 1 操作，方便用户通过 proc 追踪缓存块。

AX_POOL_ReleaseBlock

【描述】

尝试归还一个借用的缓存块。

【语法】

```
AX_S32 AX_POOL_ReleaseBlock(AX_BLK BlockId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 需要与 AX_POOL_GetBlock 成对使用，否则会造成缓存块被 AX_ID_USER 占用。
- 此接口默认内部会对该缓存块做 AX_ID_USER 引用计数减 1 操作，并尝试归还该缓存块。如果引用计数不为 0，接口不会返回错误，但也不释放缓存块。
- 用户需要确保 AX_POOL_GetBlock 和 AX_POOL_ReleaseBlock 成对调用，否则会导致缓存块一直被 AX_ID_USER 占用，无法归还，最终缓存块被耗尽。

AX_POOL_Handle2PhysAddr

【描述】

已知缓存块的 BlockID，查找该缓存块对应的物理地址。

【语法】

```
AX_U64 AX_POOL_Handle2PhysAddr(AX_BLK BlockId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入

【返回值】

返回值	描述
0	无效返回值，输入的缓存块 ID 非法
非 0	有效的缓存块物理地址

【注意】

- 输入的缓存块 ID 如果是无效值，则会返回失败。
- 输入的缓存块 ID 如果是已经被释放的，则会返回失败。
- 缓存块 ID 和物理地址的对应关系不是固定不变的。某一时刻 BlockId_A 对应物理地址 PhyAddr_A，当 BlockId_A 被归还后，未来某个时刻再次获取到 BlockId_A，则其对应的物理地址并不一定等于 PhyAddr_A，有可能是其他值。所以要想知道缓存块 ID 对应的物理地址，必须调用此接口去获取。

AX_POOL_PhysAddr2Handle

【描述】

已知缓存块的物理地址，查找对应的缓存块 ID。

【语法】

```
AX_BLK AX_POOL_PhysAddr2Handle(AX_U64 PhysAddr);
```

【参数】

参数名称	描述	输入/输出
PhysAddr	缓存块物理地址	输入

【返回值】

返回值	描述
非 AX_INVALID_BLOCKID	成功，有效的缓存块 ID
AX_INVALID_BLOCKID	失败，无效的缓存块 ID

【注意】

- 缓存块所在缓存池已销毁，则会返回失败。
- 传入一个已被归还的缓存块的物理地址 PhysAddr，则会返回失败。

AX_POOL_Handle2MetaPhysAddr

【描述】

已知缓存块的 BlockID，查找该缓存块对应 Metadata 的起始物理地址。

【语法】

```
AX_U64 AX_POOL_Handle2MetaPhysAddr(AX_BLK BlockId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入

【返回值】

返回值	描述
0	无效返回值，输入的缓存块 ID 非法
非 0	有效的缓存块 Metadata 起始物理地址

【注意】

- 输入的缓存块 ID 如果是无效值，则会返回失败。
- 输入的缓存块 ID 如果是已经被释放的，则会返回失败。
- 缓存块 ID 和 Meta 物理地址的对应关系不是固定不变的。某一时刻 BlockId_A 对应物理地址 MetaPhyAddr_A，当 BlockId_A 被归还后，未来某个时刻再次获取到 BlockId_A，则其对应的物理地址并不一定等于 MetaPhyAddr_A，有可能是其他值。所以要想知道缓存块 ID 对应的物理地址，必须调用此接口去获取。

AX_POOL_Handle2PoolId

【描述】

已知缓存块的 BlockID，查找对应的缓存池的 PoolID。

【语法】

```
AX_POOL AX_POOL_Handle2PoolId(AX_BLK BlockId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入

【返回值】

返回值	描述
非 AX_INVALID_POOLID	成功，有效的缓存池 ID
AX_INVALID_POOLID	失败，无效的缓存池 ID

【注意】

- 输入的缓存块 ID 如果是无效值，则会返回失败。
- 输入的缓存块 ID 如果是已经被释放的，则会返回失败。

AX_POOL_Handle2BlkSize

【描述】

已知缓存块的 BlockID，查找对应的缓存块大小。

【语法】

```
AX_U64 AX_POOL_Handle2BlkSize(AX_BLK BlockId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入

【返回值】

返回值	描述
0	失败，输入的缓存块 ID 非法
非 0	成功，对应缓存块的大小

【注意】

- 返回的 BlkSize 是已经做过 4KB 向上对齐后的大小，不是用户配置的原始大小。
- 输入的缓存块 ID 如果是无效值，则会返回失败。
- 输入的缓存块 ID 如果是已经被释放的，则会返回失败。

AX_POOL_GetBlockVirAddr

【描述】

已知缓存块的 BlockID，查找缓存块对应的用户态虚拟地址。

【语法】

```
AX_VOID *AX_POOL_GetBlockVirAddr(AX_BLK BlockId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入

【返回值】

返回值	描述
0	失败，无效的虚拟地址
非 0	成功，有效的缓存块虚拟地址

【注意】

- 第一次调用此 API 时，会对该 Block 所在的 Pool 所有 Block 内存整体做映射。在需要频繁调用此 API 的场景下，可以提高执行速度。
- 如果用户虚拟地址空间资源有限，尽量避免不必要的调用此 API。
- 缓存块 ID 和虚拟地址的对应关系不是固定不变的。某一时刻 BlockId_A 对应虚拟地址 VirtAddr_A，当 BlockId_A 被归还后，未来某个时刻再次获取到 BlockId_A，则其对应的虚拟地址并不一定等于 VirtAddr_A，有可能是其他值。所以要想知道缓存块 ID 对应的虚拟地址，必须调用此接口去获取。

AX_POOL_GetMetaVirAddr

【描述】

已知缓存块的 BlockID，查找缓存块对应的 Metadata 用户态虚拟地址。

【语法】

```
AX_VOID *AX_POOL_GetMetaVirAddr(AX_BLK BlockId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入

【返回值】

返回值	描述
0	失败，无效的虚拟地址
非 0	成功，有效的缓存块 Metadata 虚拟地址

【注意】

- 第一次调用此 API 时，会对该 Block 所在的 Pool 所有 Metadata 内存整体做映射。在需要频繁调用此 API 的场景下，可以提高执行速度。
- 返回的 metadata 虚拟地址始终是 Cache 类型。
- 缓存块 ID 和虚拟地址的对应关系不是固定不变的。某一时刻 BlockId_A 对应虚拟地址 VirtAddr_A，当 BlockId_A 被归还后，未来某个时刻再次获取到 BlockId_A，则其对应的虚拟地址并不一定等于 VirtAddr_A，有可能是其他值。所以要想知道缓存块 ID 对应的虚拟地址，必须调用此接口去获取。

AX_POOL_MmapPool

【描述】

为一个视频缓存池映射用户态虚拟地址。

【语法】

```
AX_S32 AX_POOL_MmapPool(AX_POOL PoolId);
```

【参数】

参数名称	描述	输入/输出
PoolId	缓存池 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 必须输入合法的缓存池 ID。
- 缓存池如果已经被销毁，则返回失败。
- 此 API 成功后，允许重复调用，不返回失败，不执行实际操作。

AX_POOL_MunmapPool

【描述】

为一个视频缓存池解除用户态映射。

【语法】

```
AX_S32 AX_POOL_MunmapPool(AX_POOL PoolId);
```

【参数】

参数名称	描述	输入/输出
PoolId	缓存池 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 必须输入合法的缓存池 ID。
- 如果缓存池未映射，则直接返回成功。
- 缓存池中的缓存块如果被占用，视为正在使用用户态虚拟地址，则调用此 API 会返回失败。
- 此 API 成功后，允许重复调用，不返回失败，不执行实际操作。

AX_POOL_IncreaseRefCnt

【描述】

已知缓存块的 BlockID，对其引用计数执行加 1 操作。

【语法】

```
AX_S32 AX_POOL_IncreaseRefCnt(AX_BLK BlockId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 该接口有成功后，内部默认会对 AX_ID_USER 做引用计数加 1 操作。
- 对某缓存块执行引用计数加 1 操作后，可以确保用户一直占用该缓存块，防止被其他模块误释放造成数据错乱。只有引用计数减为 0，缓存块才能被真正释放。
- 需要与 AX_POOL_DecreaseRefCnt 成对使用，否则会造成缓存块无法被释放，一直被 AX_ID_USER 占用。

AX_POOL_DecreaseRefCnt

【描述】

已知缓存块的 BlockID，对其引用计数执行减 1 操作。

【语法】

```
AX_S32 AX_POOL_DecreaseRefCnt(AX_BLK BlockId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 该接口有成功后，内部默认会对 AX_ID_USER 做引用计数减 1 操作。
- 对某缓存块执行 AX_ID_USER 引用计数减 1 操作后，内部会尝试释放此缓存块，只有引用计数减为 0，缓存块才能被真正释放。
- 需要与 AX_POOL_IncreaseRefCnt 成对使用，否则会造成缓存块无法被释放，一直被 AX_ID_USER 占用。
- 对于引用计数为 0 的缓存块调用此接口，不返回失败，也不执行实际操作。

AX_POOL_FreeCommPool

【描述】

只释放公共缓存池内存。

【语法】

```
AX_S32 AX_POOL_FreeCommPool(AX_VOID);
```

【参数】

参数名称	描述	输入/输出
NULL		NULL

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 此 API 只销毁公共缓存池内存，执行成功后即解除与之有关的虚拟内存映射，并释放物理内存。
- 此 API 调用前，用户需保证所有公共缓存池的缓存块都已归还，否则销毁失败，返回 BUSY 错误码。
- 此 API 只在用户特定使用场景。例如升级软件包之前，用户先确保各模块不再占用公共缓存池，然后再通过此 API 独立释放公共缓存池内存。
- 正常业务过程中建议用户使用 AX_POOL_Exit 销毁所有缓存池。

5.4 时间管理 API

AX_SYS_GetCurPTS

【描述】

读取媒体时间戳的值，单位微秒。

【语法】

```
AX_S32 AX_SYS_GetCurPTS(AX_U64 *pu64CurPTS);
```

【参数】

参数名称	描述	输入/输出
pu64CurPTS	当前时间戳指针，单位微秒	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 接口获取的 pu64CurPTS 单位已经是微秒，不需要结合时钟频率再做转换。

AX_SYS_InitPTSBase

【描述】

设置媒体时间戳的计数基准，单位微秒。

【语法】

```
AX_S32 AX_SYS_InitPTSBase(AX_U64 u64PTSBase);
```

【参数】

参数名称	描述	输入/输出
u64PTSBase	时间戳基准，单位微秒	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 接口参数 pu64PTSBase 单位是微秒。

AX_SYS_SyncPTS

【描述】

同步媒体时间戳。

【语法】

```
AX_S32 AX_SYS_SyncPTS(AX_U64 u64PTSBase);
```

【参数】

参数名称	描述	输入/输出
u64PTSBase	时间戳基准，单位微秒	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 接口参数 pu64PTSBase 单位是微秒。
- 对当前媒体时间戳进行微秒级微调，允许时间戳倒退和前进，但微调范围不能超过 1 毫秒。

5.5 日志管理 API

AX_SYS_LogPrint

【描述】

log 输出函数。参数 level 为 log 等级；参数 tag 用于确认 log 输出的模块名；参数 id 为 log 输出的模块数字标识（便于高效进行 log 模块匹配）。

为向前兼容旧 API，AX_SYS_LogPrintEx 在 ax_sys_log.h 中被 define 为了 AX_SYS_LogPrint，使用 AX_SYS_LogPrintEx 等同于 AX_SYS_LogPrint。

【语法】

```
AX_VOID AX_SYS_LogPrint(AX_S32 level, AX_CHAR const *tag, int id, AX_CHAR const *pFormat, ...);
```

【参数】

参数名称	描述	输入/输出
level	Log level 级别，0~7 0: EMERGENCY 1: ALERT 2: CRITICAL 3: ERROR 4: WARN 5: NOTICE 6: INFO 7: DEBUG	输入
tag	Log 输出模块的 tag 标记	输入
id	id：Log 输出的模块 id，取值范围请参考 ax_global_type.h 中 AX_MOD_ID_E 定义，见《50 - AX 公共数据结构文档》。	输入

参数名称	描述	输入/输出
pFormat	Log 输出格式，类似 printf	输入

【返回值】

返回值	描述
无返回值	

【注意】

- log 的输出目的地，可通过 ax_syslog.conf 来统一配置，详细可参考《00 - AX SDK 使用说明.docx》文档中“日志系统”章节的“配置 ax_syslog.conf”小节。

AX_SYS_LogOutput

【描述】

log 输出函数，与 **AX_SYS_LogPrint** 相比，提供了另一种实现。

为向前兼容旧 API，AX_SYS_LogOutputEx 在 ax_sys_log.h 中被 define 为了 AX_SYS_LogOutput，使用 AX_SYS_LogOutputEx 等同于 AX_SYS_LogOutput。

【语法】

```
AX_VOID AX_SYS_LogOutput(AX_LOG_TARGET_E target, AX_LOG_LEVEL_E level,
AX_CHAR const *tag, int id, AX_CHAR *format, va_list vlist);
```

【参数】

参数名称	描述	输入/输出
target	需要输出的方式，1: console，2: axsyslog(日志文件)	输入
level	Log level 级别，0~7 0: EMERGENCY 1: ALERT 2: CRITICAL 3: ERROR 4: WARN 5: NOTICE 6: INFO 7: DEBUG	输入
tag	Log 输出模块的 tag 标记	输入
id	Id：Log 输出的模块 id，取值范围 0~255	输入
pFormat	Log 输出格式，类似 printf	输入
vlist	函数参数传递 list	输入

【返回值】

返回值	描述
无返回值	

【说明】

- target 为 log 的输出目的地，在未配置 ax_syslog.conf 的情况下，log 输出目的地受该参数直接影响，在配置了 ax_syslog.conf 的情况下，该参数被忽略，直接使用 ax_syslog.conf 中的配置。
- 关于 ax_syslog.conf 的说明可参考《00 - AX SDK 使用说明.docx》文档中“日志系统”章节的“配置 ax_syslog.conf”小节。
- AX_LOG_LEVEL_E 说明见《50 - AX 公共数据结构文档》。

5.6 链路管理 API

AX_SYS_Link

【描述】

数据源和数据接收者建立绑定关系接口。

【语法】

```
AX_S32 AX_SYS_Link(const AX_MOD_INFO_T *pSrc,const AX_MOD_INFO_T *pDest);
```

【参数】

参数名称	描述	输入/输出
pSrc	数据源	输入
pDest	数据接收者	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 支持一对多，即同一个数据源支持最多绑定六个不同的数据接收者。
- 不支持多对一，即同一个数据接收者只能绑定一个数据源。
- 绑定关系生命周期等于当前进程生命周期。多进程场景，进程 A 创建绑定关系 link A，进程 B 创建绑定关系 link B。若进程 A 异常退出，绑定关系 link A 会被 SDK 自动回收清除，而进程 B 所创建的绑定关系 link B 不受影响。当所有进程均退出时，SDK 会自动回收清除所有的绑定关系配置。
- 对于相同的绑定关系，重复调用此接口不报错，不影响之前的绑定关系。
- JENC 和 VENC 统一使用 AX_ID_VENC 建立绑定关系，不支持 AX_ID_JENC 操作。

- 与 AX_SYS_UnLink 接口配套使用。结构体实参使用前建议先清 0，避免随机值影响。

AX_SYS_UnLink

【描述】

数据源和数据接收者解除绑定关系接口。

【语法】

```
AX_S32 AX_SYS_UnLink(const AX_MOD_INFO_T *pSrc,const AX_MOD_INFO_T *pDest);
```

【参数】

参数名称	描述	输入/输出
pSrc	数据源	输入
pDest	数据接收者	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- pSrc 如果没有和 pDest 绑定，该接口会返回失败，但对其他绑定关系没有影响。
- 结构体实参使用前建议先清 0，避免随机值影响。

AX_SYS_GetLinkByDest

【描述】

获取数据接收者绑定的数据源信息。

【语法】

```
AX_S32 AX_SYS_GetLinkByDest(const AX_MOD_INFO_T *pDest,AX_MOD_INFO_T *pSrc);
```

【参数】

参数名称	描述	输入/输出
pDest	数据源	输入
pSrc	数据接收者	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

- 结构体实参使用前建议先清 0，避免随机值影响。

AX_SYS_GetLinkBySrc

【描述】

获取数据源绑定的所有数据接收者信息集合。

【语法】

```
AX_S32 AX_SYS_GetLinkBySrc(const AX_MOD_INFO_T *pSrc, AX_LINK_DEST_T
*pLinkDest);
```

【参数】

参数名称	描述	输入/输出
pSrc	数据源	输入
pLinkDest	该数据源所有数据接收者信息集合	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 同一个数据源，最多获取四个数据接收者信息。
- 结构体实参使用前建议先清 0，避免随机值影响。
- AX_LINK_DEST_T 说明见《50 - AX 公共数据结构文档》。

AX_SYS_SetVINIVPSMode

【描述】

设置 VIN、IVPS 工作模式。

【语法】

```
AX_S32 AX_SYS_SetVINIVPSMode(AX_U32 nVinId, AX_U32 nIvpsId,
AX_VIN_IVPS_MODE_E eMode);
```

【参数】

参数名称	描述	输入/输出
nVinId	VIN 模块 PIPE ID。 有效范围：[0, AX_VIN_MAX_PIPE_NUM）	输入
nIvpsId	IVPS 模块 GRP ID。 有效范围：[0, AX_IVPS_MAX_GRP_NUM）	输入
eMode	VIN、IVPS 工作模式	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 只允许一对一，即一个 VIN 模块 PIPE ID 只能和一个 IVPS 模块 GRP ID 建立模式关系，反之亦然。如果用户尝试建立一对多或多对一的模式关系，此接口将报错，返回错误码 AX_ERR_SYS_NOT_PERM。
- 只允许在建立模式关系的两个对象(PIPE ID 和 GRP ID)创建之前调用此接口。
- 允许重复调用，会覆盖前一次的配置，但需要遵守第二个原则。
- 所有进程退出后，SDK 会强制清空之前所有的模式配置。

- `AX_VIN_IVPS_MODE_E` 说明见《50 - AX 公共数据结构文档》。

5.7 获取芯片类型 API

AX_SYS_GetChipType

【描述】

获取当前芯片类型。

【语法】

```
AX_CHIP_TYPE_E AX_SYS_GetChipType(AX_VOID);
```

【参数】

参数名称	描述	输入/输出
NULL		NULL

【返回值】

返回值	描述
AX_CHIP_TYPE_E 枚举	当前芯片类型

【注意】

- 接口获取的是当前芯片类型，返回的是枚举数值。
- AX_CHIP_TYPE_E 说明见《50 - AX 公共数据结构文档》。

5.8 休眠唤醒 API

AX_SYS_Sleep

【描述】

发起系统进入休眠的请求接口。

【语法】

```
AX_S32 AX_SYS_Sleep(AX_VOID);
```

【参数】

参数名称	描述	输入/输出
无	无	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 系统如果成功进入休眠，此接口会阻塞。当成功唤醒后，此接口会成功返回。
- 如果系统不支持休眠唤醒功能，此接口会返回失败。

AX_SYS_RegisterEventCb

【描述】

休眠唤醒事件回调函数注册接口。

【语法】

```
AX_S32 AX_SYS_RegisterEventCb(const AX_MOD_ID_E ModId,NotifyEventCallback
pFunction,AX_VOID * pData);
```

【参数】

参数名称	描述	输入/输出
ModId	模块 ID	输入
pFunction	模块实现的休眠唤醒事件回调函数。 回调函数支持两种事件： ➢ AX_NOTIFY_EVENT_SLEEP：休眠事件 ➢ AX_NOTIFY_EVENT_WAKEUP：唤醒事件	输入
pData	用于透传模块实现的休眠唤醒事件回调函数的参数。 如果回调函数不需要参数，则设置为 NULL。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 此接口主要服务于 SDK 内部模块，调用 AX_SYS_Sleep 接口会触发模块实现的回调函数执行。
- 调用 AX_SYS_Sleep 发起休眠请求时，回调函数会收到 AX_NOTIFY_EVENT_SLEEP 事件。系统唤醒后，回调函数会收到 AX_NOTIFY_EVENT_WAKEUP 事件。

- 用户如果有特殊需求，例如希望在休眠或唤醒事件消息到达时做一些业务处理，则用户可通过此接口注册事件回调函数，并在函数中处理事件消息。如果用户回调函数里的工作需要异步实现，则要配合 AX_SYS_WakeLock/AX_SYS_WakeUnlock 接口使用，确保系统正式进入休眠之前所有模块准备工作均妥善完成。
- 如果用户无上述需求，可忽略此接口

AX_SYS_UnregisterEventCb

【描述】

休眠唤醒事件回调函数注销接口。

【语法】

```
AX_S32 AX_SYS_UnregisterEventCb(const AX_MOD_ID_E ModId);
```

【参数】

参数名称	描述	输入/输出
ModId	模块 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 此接口主要服务于 SDK 内部模块。用户如果有需求，需配合 AX_SYS_RegisterEventCb 接口使用。
- 如果用户无休眠唤醒需求，可忽略此接口。

AX_SYS_WakeLock

【描述】

用户态获取 WakeLock 锁。

【语法】

```
AX_S32 AX_SYS_WakeLock(const AX_MOD_ID_E ModId);
```

【参数】

参数名称	描述	输入/输出
ModId	模块 ID。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 此接口主要服务于 SDK 内部模块，用于在用户态获取 WakeLock 锁。
- 目前支持如下模块 ID：AX_ID_ISP、AX_ID_CE、AX_ID_VO、AX_ID_VDSP、AX_ID_NPU、AX_ID_VENC、AX_ID_VDEC、AX_ID_JENC、AX_ID_JDEC、AX_ID_AENC、AX_ID_IVPS、AX_ID_ADEC、AX_ID_VIN、AX_ID_USER、AX_ID_IVES、AX_ID_SKEL、AX_ID_IVE、AX_ID_AVS、AX_ID_AVSCALI、AX_ID_AUDIO、AX_ID_ALGO、AX_ID_ENGINE、AX_ID_ACODEC、AX_ID_AI、AX_ID_AO、AX_ID_SENSOR、AX_ID_NT、AX_ID_TDP、AX_ID_VPP、AX_ID_GDC、AX_ID_3A_AE、AX_ID_3A_AWB、AX_ID_3A_AF、AX_ID_AXGZIPD。其中，AX_ID_USER 是 SDK 为用户预留的 ID。
- 需要与 AX_SYS_WakeUnlock 接口成对使用，否则会导致系统一直持有 WakeLock 锁而无法进入休眠。
- 如果用户无休眠唤醒需求，可忽略此接口。

AX_SYS_WakeUnlock

【描述】

用户态释放 WakeLock 锁。

【语法】

```
AX_S32 AX_SYS_WakeUnlock(const AX_MOD_ID_E ModId);
```

【参数】

参数名称	描述	输入/输出
ModId	模块 ID。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 此接口主要服务于 SDK 内部模块，用于在用户态释放 WakeLock 锁。
- 目前支持如下模块 ID：AX_ID_ISP、AX_ID_CE、AX_ID_VO、AX_ID_VDSP、AX_ID_NPU、AX_ID_VENC、AX_ID_VDEC、AX_ID_JENC、AX_ID_JDEC、AX_ID_AENC、AX_ID_IVPS、AX_ID_ADEC、AX_ID_VIN、AX_ID_USER、AX_ID_IVES、AX_ID_SKEL、AX_ID_IVE、AX_ID_AVS、AX_ID_AVSCALI、AX_ID_AUDIO、AX_ID_ALGO、AX_ID_ENGINE、AX_ID_ACODEC、AX_ID_AI、AX_ID_AO、AX_ID_SENSOR、AX_ID_NT、AX_ID_TDP、AX_ID_VPP、AX_ID_GDC、AX_ID_3A_AE、AX_ID_3A_AWB、AX_ID_3A_AF、AX_ID_AXGZIPD。其中，AX_ID_USER 是 SDK 为用户预留的 ID。
- 释放一个不存在的 WakeLock 锁，会返回错误码。
- 如果用户无休眠唤醒需求，可忽略此接口。

AX_SYS_SleepTimeStamp

【描述】

用于用户态休眠唤醒流程添加时间打点

【语法】

```
AX_S32 AX_SYS_SleepTimeStamp(const AX_MOD_ID_E ModId, AX_U8 SubId)
```

【参数】

参数名称	描述	输入/输出
ModId	模块 ID。	输入
SubId	每个模块打点 index，从 0 开始	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

【注意】

- 此接口主要用于统计用户态休眠唤醒流程各阶段的耗时情况

如果用户无休眠唤醒需求，可忽略此接口。

6 数据类型

公共数据类型如下：

```
typedef  AX_U32      AX_POOL;
typedef  AX_U32      AX_BLK;
/*  无效的缓存池 ID */
#define  AX_INVALID_POOLID        (-1U)
/*  无效的缓存块 ID */
#define  AX_INVALID_BLOCKID     (0)
/*  最大的缓存池个数  */
#define  AX_MAX_POOLS          64
/*  最大的公共缓存池个数  */
#define  AX_MAX_COMM_POOLS          16
/*  最大的缓存块个数  */
#define  AX_MAX_BLKS_PER_POOL     128
/*  允许建立的最大绑定关系个数  */
#define AX_LINK_DEST_MAXNUM        6
/*  缓存池名最大长度  */
#define AX_MAX_POOL_NAME_LEN      32
/* CMM 分段名最大长度  */
#define AX_MAX_PARTITION_NAME_LEN      32
/*  最大的 CMM 分段个数  */
```

#define AX_MAX_PARTITION_COUNT 16

系统控制相关数据类型、数据结构定义如下：

- AX_POOL_CACHE_MODE_E：定义视频缓存池虚拟地址映射方式枚举类型。
- AX_POOL_SOURCE_E：定义视频缓存块来源枚举类型。
- AX_POOL_CONFIG_T：定义视频缓存池配置结构体。
- AX_POOL_FLOORPLAN_T：定义媒体子系统视频缓存池配置结构体。
- AX_PARTITION_INFO_T：定义 CMM 单个内存分段详细配置信息。
- AX_CMM_PARTITION_INFO_T：定义 CMM 内存分段配置信息。

AX_POOL_CACHE_MODE_E

【说明】

定义视频缓存池虚拟地址映射方式枚举类型。

【定义】

```
typedef enum {

    AX_POOL_REMAP_MODE_NONCACHE = 0,

    AX_POOL_REMAP_MODE_CACHED = 1,

    AX_POOL_REMAP_MODE_BUTT

} AX_POOL_CACHE_MODE_E;
```

【成员】

成员名称	描述
AX_POOL_REMAP_MODE_NONCACHE	Non-cached 类型。
AX_POOL_REMAP_MODE_CACHED	Cached 类型。

【注意】

- 当配置为 Cached 类型时，各模块在访问缓存块数据时，需要注意对 Cached 虚拟地址做 Flush Cache 和 Invalidate Cache，否则会出现数据一致性问题。目前各模块暂不支持。

AX_POOL_SOURCE_E

【说明】

定义视频缓存块来源枚举类型。

【定义】

```
typedef enum {

    AX_POOL_SOURCE_COMMON = 0,

    AX_POOL_SOURCE_PRIVATE = 1,

    AX_POOL_SOURCE_USER = 2,

    AX_POOL_SOURCE_BUTT

} AX_POOL_SOURCE_E;
```

【成员】

成员名称	描述
AX_POOL_SOURCE_COMMON	来自公共缓存池。
AX_POOL_SOURCE_PRIVATE	来自私有缓存池。暂不支持。
AX_POOL_SOURCE_USER	来自用户缓存池。

AX_POOL_CONFIG_T

【说明】

定义视频缓存池配置结构体。

【定义】

```
typedef struct {
    AX_U64 MetaSize;
    AX_U64 BlkSize;
    AX_U32 BlkCnt;
    AX_BOOL IsMergeMode;
    AX_POOL_CACHE_MODE_E CacheMode;
    AX_S8 PartitionName[MAX_PARTITION_NAME_LEN];
    AX_S8 PoolName[MAX_POOL_NAME_LEN];
} AX_POOL_CONFIG_T;
```

【成员】

成员名称	描述
MetaSize	缓存块的 Metadata 大小，以 Byte 为单位。
BlkSize	缓存块大小，以 Byte 为单位。
BlkCnt	每个缓存池的缓存块个数。 范围：(0， AX_MAX_BLKS_PER_POOL]
IsMergeMode	该缓存池是否被逻辑上划分为公共缓存池。对于通过 AX_POOL_CreatePool 接口创建的缓存池，如果该属性为 AX_TRUE，则

	可以被当做公共缓存池使用。
CacheMode	缓存池映射类型。
PartitionName	缓存池所在 CMM 分段名，设为 NULL 则默认 anonymous 分段。
PoolName	缓存池名称。 ☞ 备注： ● 如果用户显式的指定了缓存池名称，则以此名作为 cat proc/ax_proc/mem_cmm_info 中显示的名称。 ● 如果用户没有指定名称，则 SDK 默认以“线程名_线程 ID_缓存池 ID”作为该缓存池名称，例如：threadfunc_500_pool_1 ● 该属性对于公共缓存池不生效。 ● 建议名称具有一定的辨识度，方便定位问题。

【注意】

- 每个缓存块的大小 BlkSize 应根据当前图像宽高、像素格式、是否压缩等信息来计算。
- 缓存块的 Metadata 内存主要作用是携带辅助信息。用户可以根据需要将该缓存块中图像数据相关的辅助信息写入到 Metadata 内存，通过 BlockID 可以同时访问到图像数据和辅助信息。
- 图像数据在 SDK 各模块间流转是基于缓存池的，需要用到 Metadata 内存。所以缓存池的 MetaSize 不能为 0，否则数据流转会异常。为确保使用安全，SDK 默认 MetaSize 最小值为 4K 字节，当用户设置的 MetaSize 小于 4K 字节时，会使用默认值。
- 如果创建的缓存池用于 SDK 各模块间流转数据，则用户不能修改 Metadata 里面的内容，否则会造成数据流转异常。
- 如果创建的缓存池只有用户才会使用，则用户可以根据需要灵活决定是否使用 Metadata 内存传递辅助信息。

AX_POOL_FLOORPLAN_T

【说明】

定义媒体子系统视频缓存池配置结构体。

【定义】

```
typedef struct {

    AX_POOL_CONFIG_T CommPool[AX_MAX_COMM_POOLS];

} AX_POOL_FLOORPLAN_T;
```

【成员】

成员名称	描述
CommPool	公共缓存池配置结构体数组。

【注意】

- BlkSize 或 BlkCnt 等于 0，则对于的缓存池不会被创建。
- 结构体实参使用前建议先清 0，避免随机值影响。
- 每个公共缓存池之间 BlkSize 不能一样，否则会报错。

AX_PARTITION_INFO_T

【说明】

定义 CMM 单个内存分段详细配置信息。

【定义】

```
typedef struct {

   AX_U64 PhysAddr;

   AX_U32 SizeKB;

   AX_S8  Name[MAX_PARTITION_NAME_LEN];

} AX_PARTITION_INFO_T;
```

【成员】

成员名称	描述
PhysAddr	分段起始物理地址。
SizeKB	分段内存大小，以 KB 为单位。
Name	分段名称。

AX_CMM_PARTITION_INFO_T

【说明】

定义 CMM 内存分段配置信息。

【定义】

```
typedef struct {

    AX_U32 PartitionCnt;

    AX_PARTITION_INFO_T PartitionInfo[MAX_PARTITION_COUNT];

} AX_CMM_PARTITION_INFO_T;
```

【成员】

成员名称	描述
PartitionCnt	分段个数。 范围：[1，MAX_PARTITION_COUNT]
PartitionInfo	CMM 单个内存分段详细配置信息。

AX_CMM_STATUS_T

【说明】

定义 CMM 内存当前使用状态信息。

【定义】

```
typedef struct {
    AX_U32 TotalSize;
    AX_U32 RemainSize;
    AX_U32 BlockCnt;
    AX_CMM_PARTITION_INFO_T Partition;
} AX_CMM_STATUS_T;
```

【成员】

成员名称	描述
TotalSize	CMM 内存总大小，单位 KB。
RemainSize	CMM 剩余内存大小，单位 KB。
BlockCnt	当前总共分配的 CMM 内存块个数。
Partition	CMM 内存分段信息。

7 错误码

错误码详见《55 - AX 软件错误码文档》文档。

8 调试信息

8.1 CMM

【调试信息】

```
# cat /proc/ax_proc/mem_cmm_info
```

运行过程中，可以 cat 该节点，查看 CMM 内存使用状态，示例信息如下：

```
--------------------SDK VERSION--------------------
[Axera version]: ax_cmm V0.3.0 Aug 16 2023 10:38:08
+---PARTITION: Phys(0x80000000, 0xBFFFFFFF), Size=1048576KB(1024MB),    NAME="anonymous"
 nBlock(Max=0, Cur=15, New=0, Free=0)  nbytes(Max=0B(0KB,0MB), Cur=33914880B(33120KB,32MB), New=0B(0KB,0MB), Free=0B(0KB,0MB))
 Block(Max=0B(0KB,0MB), Min=0B(0KB,0MB), Avg=0B(0KB,0MB))
    |-Block: phys(0x80000000, 0x80FD1FFF), cache =non-cacheable, length=16200KB(15MB),    name="vo_vfb"
    |-Block: phys(0x80FD2000, 0x81FA3FFF), cache =non-cacheable, length=16200KB(15MB),    name="vo_vfb"
    |-Block: phys(0x81FA4000, 0x81FA7FFF), cache =non-cacheable, length=16KB(0MB),    name="vo_vfb"
    |-Block: phys(0x81FA8000, 0x81FABFFF), cache =non-cacheable, length=16KB(0MB),    name="VPP_DEV"
    |-Block: phys(0x81FAC000, 0x81FACFFF), cache =non-cacheable, length=4KB(0MB),    name="VPP_CMODE3"
    |-Block: phys(0x81FAD000, 0x81FADFFF), cache =non-cacheable, length=4KB(0MB),    name="GDC_CMDA3"
    |-Block: phys(0x81FAE000, 0x81FB0FFF), cache =non-cacheable, length=12KB(0MB),    name="GDC_CMD"
    |-Block: phys(0x81FB1000, 0x81FB4FFF), cache =non-cacheable, length=16KB(0MB),    name="TDP_DEV"
    |-Block: phys(0x81FB5000, 0x81FB5FFF), cache =non-cacheable, length=4KB(0MB),    name="TDP_CMODE3"
    |-Block: phys(0x81FB6000, 0x81FF5FFF), cache =non-cacheable, length=256KB(0MB),    name="venc_ko"
    |-Block: phys(0x81FF6000, 0x82035FFF), cache =non-cacheable, length=256KB(0MB),    name="venc_ko"
    |-Block: phys(0x82036000, 0x82036FFF), cache =non-cacheable, length=4KB(0MB),    name="venc_ko"
    |-Block: phys(0x82037000, 0x82046FFF), cache =non-cacheable, length=64KB(0MB),    name="jenc_ko"
    |-Block: phys(0x82047000, 0x82056FFF), cache =non-cacheable, length=64KB(0MB),    name="jenc_ko"
    |-Block: phys(0x82057000, 0x82057FFF), cache =non-cacheable, length=4KB(0MB),    name="jenc_ko"

---CMM_USE_INFO:
 total size=1048576KB(1024MB),used=33120KB(32MB + 352KB),remain=1015456KB(991MB + 672KB),partition_number=1,block_number=15
```

【调试信息分析】

记录当前 CMM 内存使用情况。

【参数说明】

参数		描述
PARTITION （单个 CMM 分段信息）	Phys	该分段物理地址范围。
	Size	该分段总内存大小。
	NAME	该分段名称。默认有一个 anonymous 分段。
	nBlock	Max：该分段上历史过程中最大分配过多少个内存块。

参数		描述
		Cur：该分段上分配了多少个内存块。 New：该分段上申请内存块的总次数。 Free：该分段上释放内存块的总次数。
	nbytes	Max：该分段上历史过程中最大分配过多少内存。 Cur：该分段上分配了多少内存。 New：该分段上申请的内存块的总量。 Free：该分段上释放的内存块的总量。
	Block	Max：该分段上申请的最大单个内存块大小。 Min：该分段上申请的最小单个内存块大小。 Avg：该分段上申请的平均内存块大小。
CMM_USE_INFO （整个 CMM 内存信息）	total size	CMM 内存总大小。
	used	已经被使用的 CMM 内存大小。
	remain	剩余可用的 CMM 内存大小。
	partition_number	分段个数。
	block_number	所有分段当前总共申请的内存块个数。

8.2 缓存池

【调试信息】

```
# cat /proc/ax_proc/pool
```

运行过程中，可以 cat 该节点，查看缓存池状态，示例信息如下：

```
----SDK VERSION------------------------------------------------------------------------

[Axera version]: ax_pool V1.7.0 Jan 1 2024 12:02:12

----COMMON POOL CONFIG------------------------------------------------------------------

Index            0        1        2        3

MetaSize     10240    10240    10240    10240

BlkSize     718848  3214080  5496320  6274560

BlkCnt          15       15       13       15

----ALL POOL INFO----------------------------------------------------------------------

PoolId  IsComm  IsCache Partition  PhysAddr    MetaSize  BlkSize   BlkCnt  FreeCnt
MinFreeCnt  IsDelayFree

0       1       0       anonymous  0x8014A000  12288     720896    15      7
7       0

Index BlockId VIN IVPS VO VENC JENC VDEC JDEC SKEL AI AO AENC ADEC AVS USER  ReqSize

0   0x5E000000 1   0   0   0   0   0   0 0  0   0   0  0  0  0  0 718848

1   0x5E000001 1   0   0   0   0   0   0 0  0   0   0  0  0  0  0 718848

2   0x5E000002 0   0   0   1   0   0   0 0  0   0   0  0  0  0  0 718848

3   0x5E000003 0   0   0   1   0   0   0 0  0   0   0  0  0  0  0 718848

4   0x5E000004 1   0   0   0   0   0   0 0  0   0   0  0  0  0  0 718848

5   0x5E000005 1   0   0   0   0   0   0 0  0   0   0  0  0  0  0 718848

6   0x5E000006 0   1   0   0   0   0   0 0  0   0   0  0  0  0  0 718848

7   0x5E000007 1   0   0   0   0   0   0 0  0   0   0  0  0  0  0 718848

-----------------------------------------------------------------------------------------

PoolId  IsComm  IsCache Partition  PhysAddr    MetaSize  BlkSize   BlkCnt  FreeCnt
```

```
MinFreeCnt   IsDelayFree
1       1       0       anonymous  0x80BC7000   12288    3215360   15     9
3       0
Index BlockId VIN IVPS VO VENC JENC VDEC JDEC SKEL AI AO AENC ADEC AVS USER  ReqSize
0   0x5E001000 1   0   0   0   0   0   0   0  0   0   0   0   0   0  0 3214080
1   0x5E001001 1   0   0   0   0   0   0   0  0   0   0   0   0   0  0 3214080
2   0x5E001002 1   0   0   0   0   0   0   0  0   0   0   0   0   0  0 3214080
3   0x5E001003 1   0   0   0   0   0   0   0  0   0   0   0   0   0  0 3214080
4   0x5E001004 1   0   0   0   0   0   0   0  0   0   0   0   0   0  0 3214080
6   0x5E001006 0   0   0   1   0   0   0   0  0   0   0   0   0   0  0 3214080
--------------------------------------------------------------------------------
PoolId  IsComm  IsCache Partition  PhysAddr    MetaSize  BlkSize   BlkCnt  FreeCnt
MinFreeCnt  IsDelayFree
2       1       0       anonymous  0x839F3000   12288     5496832   13     11
10      0
Index BlockId VIN IVPS VO VENC JENC VDEC JDEC SKEL AI AO AENC ADEC AVS USER  ReqSize
1   0x5E002001 2   0   0   0   0   0   0  0   0   0   0   0   0   0  0 5496320
2   0x5E002002 1   0   0   0   0   0   0  0   0   0   0   0   0   0  0 5496320
--------------------------------------------------------------------------------
PoolId  IsComm  IsCache Partition  PhysAddr    MetaSize  BlkSize   BlkCnt  FreeCnt
MinFreeCnt   IsDelayFree
3       1       0       anonymous  0x87E40000   12288     6275072   15     7
5       0
```

```
Index BlockId VIN IVPS VO VENC JENC VDEC JDEC SKEL AI AO AENC ADEC AVS USER  ReqSize
0   0x5E003000 1   0   0   0   0   0   0   0   0   0   0   0   0   0 6274560
1   0x5E003001 1   0   0   0   0   0   0   0   0   0   0   0   0   0 6274560
2   0x5E003002 1   0   0   0   0   0   0   0   0   0   0   0   0   0 6274560
3   0x5E003003 1   0   0   0   0   0   0   0   0   0   0   0   0   0 6274560
5   0x5E003005 1   0   0   0   0   0   0   0   0   0   0   0   0   0 6274560
6   0x5E003006 0   1   0   0   0   0   0   0   0   0   0   0   0   0 6274560
7   0x5E003007 0   1   0   0   0   0   0   0   0   0   0   0   0   0 6274560
9   0x5E003009 0   0   0   1   0   0   0   0   0   0   0   0   0   0 6274560
```

【调试信息分析】

记录当前缓存池模块配置及 buffer 占用情况。

【参数说明】

参数		描述
SDK VERSION		SDK 版本信息。
COMMON POOL CONFIG（公共缓存池配置）	Index	公共缓存池配置数组索引。
	MetaSize	缓存池内缓存块的 Metadata 大小。
	BlkSize	缓存池内缓存块的大小。
	BlkCnt	缓存池内缓存块的个数。
ALL POOL INFO（公共/私有缓存池使用情况）	PoolId	公共/私有缓存池 ID。
	IsComm	是否为公共缓存池。取值：{0,1}
	IsCache	缓存池是否为 Cache 映射类型。取值：{0,1}
	Partition	缓存池所在的分段名。
	PhysAddr	缓存池起始物理地址。
	MetaSize	缓存池内缓存块的 Metadata 大小（4K 对齐

参数		描述
		后）。
	BlkSize	缓存池内缓存块的大小（4K 对齐后）。
	BlkCnt	缓存池内缓存块的个数。
	FreeCnt	缓存池内当前空闲缓存块的个数。
	MinFreeCnt	缓存池内最小空闲缓存块个数。如果该值大于 0，说明缓存块个数分配过剩。
	IsDelayFree	缓存池是否已被标记为延迟销毁。取值：{0,1}。 仅当 IsComm=0 时，该值有意义。
	Index	缓存池内缓存块的索引。
	BlockId	缓存池内缓存块的 ID。 BlockId 是一个 32 位值，组成结构如下： (MagicId << 24) | (PoolId << 12) | (Index) MagicId：固定为 0x5E。 PoolId：缓存池 ID。 Index：缓存池内缓存块的索引。
	VIN/IVPS/VO /VENC /JENC /VDEC/JDEC/ SKEL/AI /AO/AENC/ ADEC/AVS/USER	模块名 下面对应的数字表示当前模块有多少个地方占用缓存池内的该缓存块。 • 0：没占用。 • 非 0：占用次数。
	ReqSize	实际请求的内存大小。可以帮助用户排查是否发生缓存块抢占。

8.3 绑定关系

【调试信息】

```
# cat /proc/ax_proc/link_table
```

运行过程中，可以 cat 该节点，查看当前系统已存在的绑定关系，示例信息如下：

```
--------------------Link Table--------------------
        Src           |         Dst
(ModId   GrpId   ChnId) | (ModId    GrpId   ChnId)
--------------------------------------------------
(VIN      0       2)   -> (IVPS      2       0)
(VIN      0       1)   -> (IVPS      1       0)
(VIN      0       0)   -> (IVPS      0       0)
(IVPS     1       0)   -> (VENC      0       1)
(IVPS     0       0)   -> (VENC      0       0)
```

【调试信息分析】

记录当前系统存在的绑定关系。

【参数说明】

参数	描述
Src	数据源（模块-组-通道）信息。
Dst	数据接收者（模块-组-通道）信息。

8.4 VIN、IVPS 模式配置

【调试信息】

```
# cat /proc/ax_proc/sys/vin_ivps_mode
```

运行过程中，可以 cat 该节点，查看当前系统已存在的 VIN、IVPS 模式配置，示例信息如下：

```
--------------------SDK VERSION--------------------

[Axera version]: ax_sys V0.2.0 Aug 15 2023 15:14:35

-------------------VIN IVPS MODE-------------------

VinId   IvpsId  Mode

---------------------------------------------------

3       4       GDC_ONLINE_VPP

2       3       ITP_ONLINE_VPP

1       1       ITP_OFFLINE_VPP
```

【调试信息分析】

记录当前系统存在的 VIN、IVPS 模式配置

【参数说明】

参数	描述
VinId	VIN 模块 PIPE ID。
IvpsId	IVPS 模式 GRP ID。
Mode	VIN、IVPS 模式配置