



AX SYS API 文档

文档版本: V2.7

发布日期: 2022/06/22

AXERA CONFIDENTIAL FOR Sipeed

前 言	7
目 录	
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	13
3.2 内存优化	15
3.3 配置内存分段	16
3.4 Cache 一致性管理	17
3.5 时间管理	18
3.6 日志管理	18
3.7 链路管理	20
3.8 模块工作频率管理	21
4 API 简介	22
4.1 系统初始化	23
4.2 内存分配	23
4.3 缓存池管理	24
4.4 时间管理	25
4.5 日志管理	25
4.6 链路管理	26
4.7 模块工作频率管理	26
5 API 定义	27

5.1 系统初始化 API.....	28
AX_SYS_Init	28
AX_SYS_Deinit.....	29
5.2 内存分配 API	30
AX_SYS_MemAlloc.....	30
AX_SYS_MemAllocCached	32
AX_SYS_MemFlushCache	34
AX_SYS_MemInvalidateCache.....	35
AX_SYS_MemFree	36
AX_SYS_MemGetBlockInfoByPhy.....	37
AX_SYS_MemGetBlockInfoByVirt	38
AX_SYS_MemGetPartitionInfo.....	39
AX_SYS_Mmap.....	40
AX_SYS_MmapCache	42
AX_SYS_MmapFast	44
AX_SYS_MmapCacheFast.....	45
AX_SYS_Munmap.....	46
AX_SYS_MflushCache	47
AX_SYS_MinvalidateCache	48
AX_SYS_MemSetConfig	49
AX_SYS_MemGetConfig.....	50
5.3 缓存池管理 API.....	51
AX_POOL_SetConfig	51
AX_POOL_GetConfig	52
AX_POOL_Init.....	53
AX_POOL_Exit	54

AX_POOL_CreatePool.....	55
AX_POOL_MarkDestroyPool	56
AX_POOL_GetBlock.....	57
AX_POOL_ReleaseBlock.....	58
AX_POOL_Handle2PhysAddr	59
AX_POOL_PhysAddr2Handle	60
AX_POOL_Handle2MetaPhysAddr	61
AX_POOL_Handle2PoolId	62
AX_POOL_GetBlockVirAddr	63
AX_POOL_GetMetaVirAddr	64
AX_POOL_MmapPool.....	65
AX_POOL_MunmapPool.....	66
AX_POOL_FlushCache	67
AX_POOL_IncreaseRefCnt.....	68
AX_POOL_DecreaseRefCnt.....	69
5.4 时间管理 API	70
AX_SYS_GetCurPTS.....	70
AX_SYS_InitPTSBase	71
AX_SYS_SyncPTS	72
5.5 日志管理 API	73
AX_SYS_LogOpen	73
AX_SYS_LogClose	74
AX_SYS_LogPrint	75
AX_SYS_LogPrint_Ex.....	76
AX_SYS_LogOutput	77
AX_SYS_LogOutput_Ex.....	78

AX_SYS_SetLogLevel	79
AX_SYS_SetLogTarget	80
AX_SYS_EnableTimestamp	81
5.6 链路管理 API	82
AX_SYS_Link	82
AX_SYS_UnLink	83
AX_SYS_GetLinkByDest	84
AX_SYS_GetLinkBySrc	85
5.7 模块工作频率 API	86
AX_SYS_CLK_SetLevel	86
AX_SYS_CLK_GetLevel	87
AX_SYS_CLK_Single_RateSet	88
6 数据结构	89
AX_MOD_INFO_S	90
AX_MOD_ID_E	91
AX_LOG_LEVEL_E	93
AX_POOL_FLOORPLAN_T	94
AX_POOL_CONFIG_T	95
AX_POOL_CACHE_MODE_E	96
AX_POOL_SOURCE_E	97
AX_PARTITION_INFO_T	98
AX_CMM_PARTITION_INFO_T	99
AX_LINK_DEST_S	100
AX_SYS_CLK_LEVEL_E	101
AX_SYS_CLK_ID_E	102
7 错误码	103

8 调试信息	106
8.1 缓存池	106
9 附录.....	110
如何查看 CMM 虚拟地址数据.....	110

AXERA CONFIDENTIAL FOR Sipeed

权利声明

爱芯元智半导体(上海)有限公司或其许可人保留一切权利。

非经权利人书面许可，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。

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。

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。

前言



适用产品

爱芯 AX620A

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
<code>xxx</code>	表示您可以执行的命令行。
<i>斜体</i>	表示变量。如，“安装目录/AX620_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	2021/08/24	文档初版
V1.1	2021/09/09	1. CMM 支持偏移地址操作 2. 更新错误码
V1.2	2021/10/27	新增 Log 接口：AX_SYS_LogPrint_Ex、 AX_SYS_LogOutput_Ex
V1.3	2021/10/28	新增 POOL 引用计数接口： AX_POOL_IncreaseRefCnt、 AX_POOL_DecreaseRefCnt
V1.4	2021/10/29	修改第二章的 Continuous 为 Contiguous
V1.5	2021/11/30	修改 AX_MOD_INFO_S 和 AX_MOD_ID_E 结构体
V1.6	2021/12/02	增加链路管理接口相关描述
V1.7	2021/12/13	AX_POOL_CONFIG_T 增加 IsMergeMode 属性，支持 扩展缓存池
V1.8	2021/12/21	添加结构体成员描述
V1.9	2022/01/14	更新 3.6 章日志管理描述
V2.0	2022/03/01	1.新增 AX_SYS_MmapFast、AX_SYS_MmapCacheFast 接口 2.更新 5.2 章 AX_SYS_MemAlloc、AX_SYS_Mmap 接 口描述 3.修改 AX_LOG_LEVEL_E 结构体
V2.1	2022/03/09	新增缓存池调试信息
V2.2	2022/04/07	更新 5.6 章节，新增 Link 关系查询接口： AX_SYS_GetLinkByDest、AX_SYS_GetLinkBySrc

文档版本	发布时间	修订说明
V2.3	2022/04/24	更新 5.6 章节，AX_SYS_Munmap 接口注意事项，不允许做分段 unmap 操作。
V2.4	2022/05/06	更新 5.4 章节，媒体时间戳接口描述，增加注意事项。
V2.5	2022/05/12	更新 3.1 章节，内存管理描述。
V2.6	2022/05/20	新增 sys_clk 接口描述
V2.7	2022/06/23	更新 5.4 章节，媒体时间戳接口调整： 时间戳单位改成微秒 AX_SYS_SyncPTS 支持 1ms 范围内的微调

AX SDK 在 SoC 软件解决方案中所处的层次位置如下图所示：**1 概述**

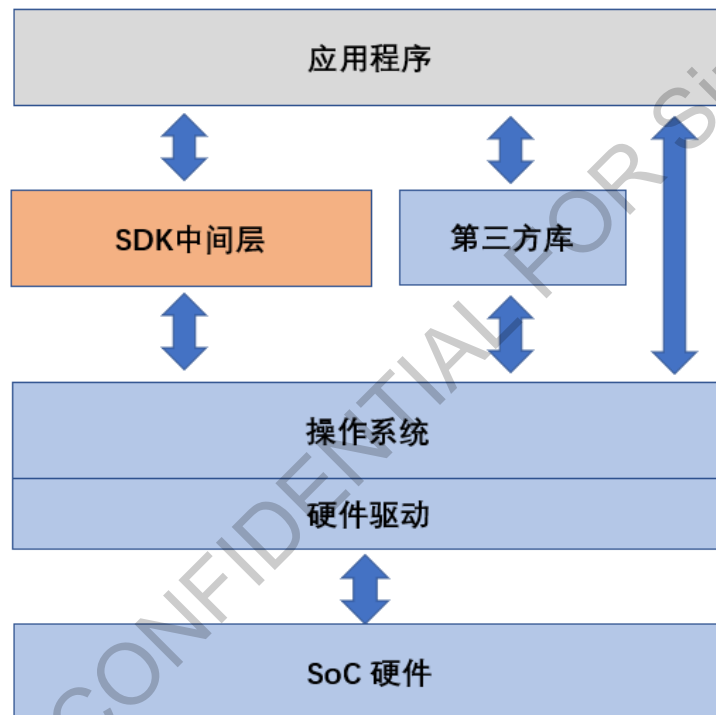


图1-1 SDK 层次示意图

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。

本章节包含：

3 系统控制模块

3.1 内存管理

3.2 内存优化

3.3 配置内存分段

3.4 Cache 一致性管理

3.5 时间管理

3.6 日志管理

3.7 链路管理

3.8 模块工作频率管理

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。SDK 支持动态创建一个 Pool，但是对动态 Pool 的销毁和回收将受到规则限制，使用时需要特别注意。
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。
19. SDK 支持在 metadata 数据结构中预留出一部分空间给客户使用，SDK 对这部分空间的用途不做约束。

基于以上设计思想，AX SDK 的内存管理模型如下图所示：

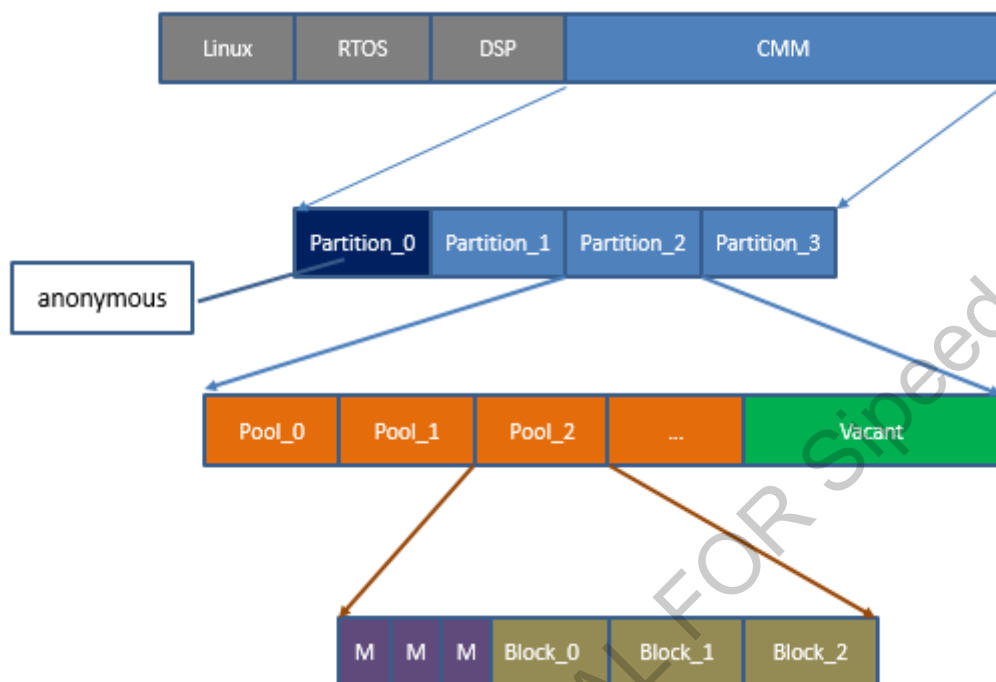


图3-1 SDK 内存池管理模型

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：

```
cmm_pool=partitionname,flag,phyaddr_start,partitionsize[:partitionname,flag,phyaddr_start,partitionsize]
```

每个参数信息如下：

- a) partitionname: 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) `partitionsizesize`: 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 AX620A EVB 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：**ISP**，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：**1790M**）。

```
#!/bin/sh
...
insmod /soc/ko/ax_cmm.ko
cmmppool=ISP,0,0x80000000,256M:anonymous,0,0x90000000,1790M
```

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
```

```
2  +---PARTITION: Phys(0x90000000, 0xFFDFFFFF), Size=1832960KB(1790MB), NAME="anonymous" ←匿名分段
3  nBlock(Max=5, Cur=5, New=5, Free=0) nbytes(Max=5242880B(5120KB,5MB), Cur=5242880B(5120KB,5MB), New=5242880B(5120KB,5MB),
4  Free=0B(0KB,0MB)) Block(Max=1048576B(1024KB,1MB), Min=1048576B(1024KB,1MB), Avg=1048576B(1024KB,1MB))
5  |-Block: phys(0x90000000, 0x9FFFFFFF), cache =non-cacheable, length=262144KB(256MB), name="mem_pool"
6  |-Block: phys(0xA0000000, 0xA0FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_0"
7  |-Block: phys(0xA0100000, 0xA01FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_1"
8  |-Block: phys(0xA0200000, 0xA02FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_2"
9  |-Block: phys(0xA0300000, 0xA03FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_3"
10 |-Block: phys(0xA0400000, 0xA04FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_4"
11 +---PARTITION: Phys(0x80000000, 0x8FFFFFFF), Size=262144KB(256MB), NAME="ISP" ←ISP分段
12 nBlock(Max=5, Cur=5, New=5, Free=0) nbytes(Max=5242880B(5120KB,5MB), Cur=5242880B(5120KB,5MB), New=5242880B(5120KB,5MB),
13 Free=0B(0KB,0MB)) Block(Max=1048576B(1024KB,1MB), Min=1048576B(1024KB,1MB), Avg=1048576B(1024KB,1MB))
14 |-Block: phys(0x80000000, 0x800FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_5"
15 |-Block: phys(0x80100000, 0x801FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_6"
16 |-Block: phys(0x80200000, 0x802FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_7"
17 |-Block: phys(0x80300000, 0x803FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_8"
18 |-Block: phys(0x80400000, 0x804FFFFFFF), cache =non-cacheable, length=1024KB(1MB), name="block_9"
19
20 ---CMM_USE_INFO:
21 total size=2095104KB(2046MB),used=272384KB(266MB + 0KB),remain=1822720KB(1780MB + 0KB),partition_number=2,block_number=11
```

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 用于显式地触发对 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：axsyslog 和 axklog。

axsyslog

axsyslog 库：通过 axsyslogd 和 axklogd 收集 log，axsyslog 采用可配置的、统一的系统登记程序，随时从系统各处接受 log 请求，然后根据配置文件 ax_syslog.conf 中的预先设定把 log 信息写入到相应文件中、邮寄给特定用户或直接以消息的方式发往控制台；

axsyslog 功能开关：在 rootfs/rootfs/etc/ax_syslog.conf 文件中，由 AX_SYSLOG_enable 控制。

ax_syslog.conf 用户修改路径：rootfs/rootfs/etc/ax_syslog.conf

axklog

axkog 管理 axera ko 驱动的 kernel log，并经过 axkogd 同一输出到 axsyslog,最终保存到 Log 文件。

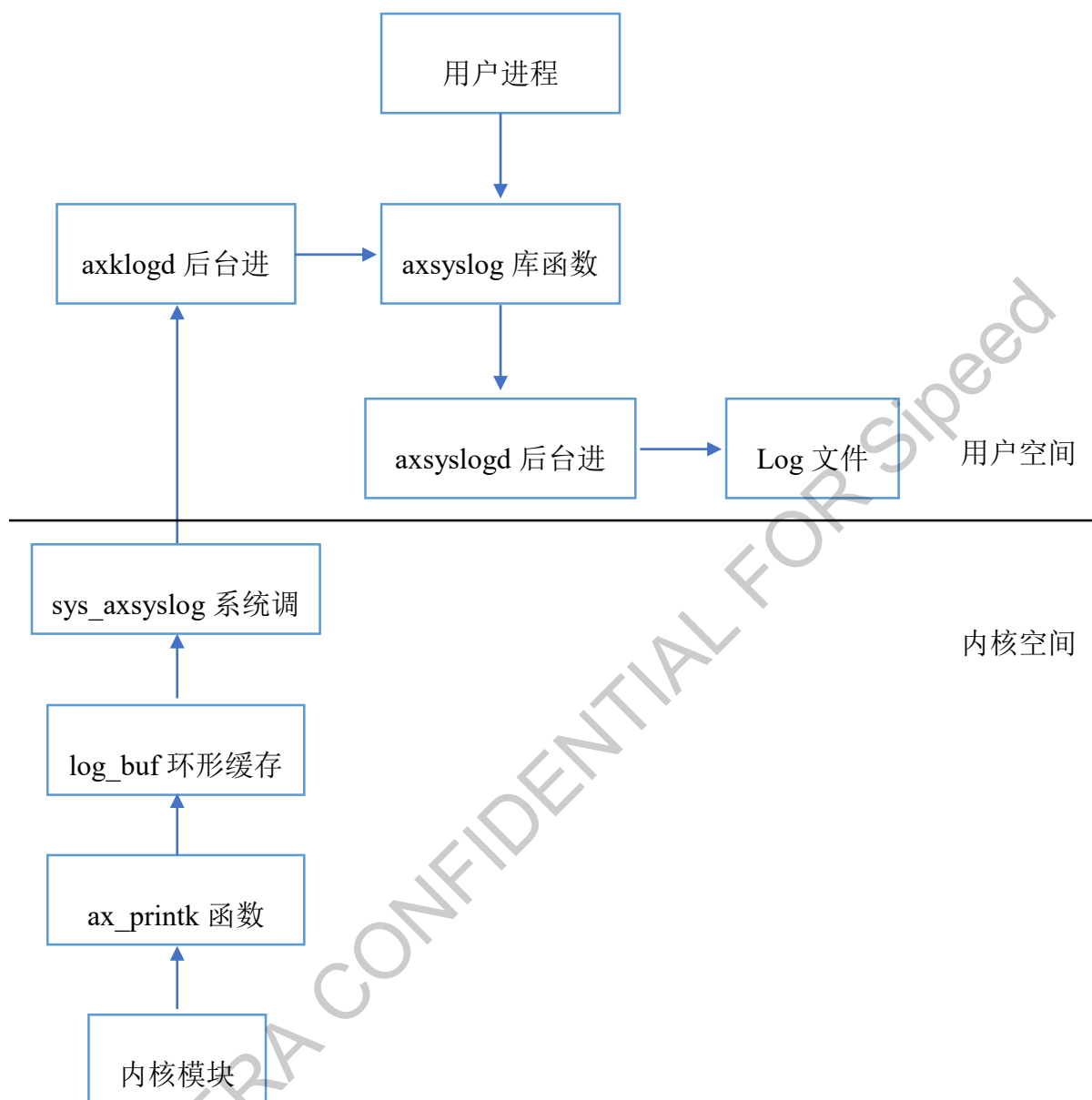


图3-3

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。同一个数据源可以支持最多绑定 4 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、JENC、IVPS
IVPS	VO、VENC、JENC、IVPS
VDEC	VO、VENC、IVPS
JDEC	VO、JENC、IVPS
VO	VO、VENC、JENC、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-----
(ModId  Src  GrpId  ChnId) | (ModId  Dst  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)
(IVPS   0    2)  -> (JENC    0    2)
(IVPS   0    1)  -> (JENC    0    1)
```

图3-4

3.8 模块工作频率管理

AX SoC 为了避免芯片工作在过热的温度下导致烧坏，增加了 `sys_clk` 接口用于调节各模块工作频率。温度过高时降低各模块工作频率以降低功耗，温度降低后工作频率调回原值以保持性能。详见《36 - AX620 软件业务场景功耗调节指南.docx》文档

AXERA CONFIDENTIAL FOR Sipeed

本章节包含：

4.1 系统初始化

4.2 内存分配

4.3 缓存池管理

4.4 时间管理

4.5 日志管理

4.6 链路管理

4.7 模块工作频率管理

4 API 简介

AXERA CONFIDENTIAL FOR Sipeed

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。
- AX_SYS_Deinit: 去初始化媒体子系统。

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_MemFlushCache: 将 CPU cache 里的内容刷新到 DDR 内存, 同时清空 cache。
- AX_SYS_MemInvalidateCache: 将 CPU cache 状态置为无效。
- 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。

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_MarkDestroyPool`: 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_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_FlushCache: 将 CPU cache 里的内容刷新到 DDR 内存, 同时清空 cache。
- 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。

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_LogOpen: 打开系统 log 功能。
- AX_SYS_LogClose: 关闭系统 log 功能。
- AX_SYS_LogPrint: 打印输出 log 到 SetLogTarget API 所设置的 target, 默认 target 是 syslog。
- AX_SYS_LogPrint_Ex: 打印输出 log 到 SetLogTarget API 所设置的 target, 默认 target 是 syslog。
- AX_SYS_LogOutput: 打印输出 log 到指定的 target。
- AX_SYS_LogOutput_Ex: 打印输出 log 到指定的 target。
- AX_SYS_SetLogLevel: 设置 log 输出级别, 0~5 级。
- AX_SYS_SetLogTarget: 设置 log 的输出目标。
- AX_SYS_EnableTimestamp: 使能 log 的时间戳。

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。

4.7 模块工作频率管理

- AX_SYS_CLK_SetLevel: 设置模块工作频率 level 值接口。
- AX_SYS_CLK_GetLevel: 获取当前模块工作频率 level 值接口。
- AX_SYS_CLK_Single_RateSet: 单独设置某一个模块工作频率的接口。

本章节包含：

5 API 定义

5.1 系统初始化 API

5.2 内存分配 API

5.3 缓存池管理 API

5.4 时间管理 API

5.5 日志管理 API

5.6 链路管理 API

5.7 模块工作频率 API

5.1 系统初始化 API

AX_SYS_Init

【描述】

对媒体子系统进行初始化，开始对媒体相关的各种软硬件资源实施管理，使相关资源从不确定的无管理状态转入确定的待命状态。

【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

AX_SYS_Deinit

【描述】

对媒体子系统进行反初始化，释放媒体子系统所持有的全部硬件和软件资源，使系统重新进入无管理状态。

【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

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	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	输入

参数名称	描述	输入/输出
	<p>Blockname 即可。</p> <p>备注：</p> <ol style="list-style-type: none">1. 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；2. token 长度不要超过 30 个字节；3. 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	<p>长度不超过 30 字节的字符串指针</p> <p>Token 的用法分两种情况：</p> <ol style="list-style-type: none">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）。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。 <div><p> 备注：</p><ol style="list-style-type: none">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。</div>	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

AXERA CONFIDENTIAL FOR Sipeed

AX_SYS_MemFlushCache

【描述】

触发 Cache Flush 操作，在 CPU 向 DDR 地址写共享数据后使用。

【语法】

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

【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
size	申请连续物理地址 cache 内存的大小	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_SYS_MemAllocCached](#)、[AX_SYS_MemInvalidateCache](#) 接口配合使用。

AX_SYS_MemInvalidateCache

【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_MemInvalidateCache(AX_U64 phyaddr, AX_VOID *pviraddr, AX_U32 size);
```

【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
size	申请连续物理地址 cache 内存的大小	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_SYS_MemAllocCached](#)、[AX_SYS_MemFlushCache](#) 接口配合使用。

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 => MEM_CACHED 4 => 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

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，但必须是 4 字节对齐）	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 => MEM_CACHED 4 => MEM_NONCACHED	输出

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

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

- 结构体实参使用前最好先清 0，避免随机值影响。

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。

AXERA CONFIDENTIAL FOR Sipeed

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。

AXERA CONFIDENTIAL FOR Sipeed

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】

- 输入的虚拟地址必须是 [AX_SYS_Mmap](#) 或 [AX_SYS_MmapCache](#) 接口映射得到的虚拟地址。
- 对于 [AX_SYS_MemAlloc](#) 和 [AX_SYS_MemAllocCached](#) 接口返回的虚拟地址，不能调用此接口刷新。

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	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	AX_SYS_MmapCache 接口返回的 cache 类型虚拟地址	输入
size	申请连续物理地址 cache 内存的大小	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_SYS_MmapCache](#)、[AX_SYS_MflushCache](#) 接口配合使用。

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_S *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_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_S *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

- 结构体实参使用前最好先清 0，避免随机值影响。

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

- 结构体实参使用前最好先清 0，避免随机值影响。

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

- 结构体实参使用前最好先清 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 SetConfig。
- 此 API 成功后，如再调用 AX_POOL_SetConfig 将返回失败。
- 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_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。
- 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。

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。
- 结构体实参使用前最好先清 0，避免随机值影响。

AX_POOL_MarkDestroyPool

【描述】

将一个由 CreatePool API 创建的视频缓存池登记为待销毁状态，并实际回收已符合销毁条件的缓存。

【语法】

```
AX_S32 AX_POOL_MarkDestroyPool (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。
- 物理内存回收策略遵循后进先出原则，最后创建的 Pool 应最先回收。

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

AX_POOL_ReleaseBlock

【描述】

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

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

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

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

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

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。

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。

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

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

AX_POOL_FlushCache

【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_POOL_FlushCache(AX_U64 PhysAddr, AX_VOID *pVirAddr, AX_U32 Size);
```

【参数】

参数名称	描述	输入/输出
PhysAddr	待操作数据的起始物理地址，必须是缓存池管理的内存	输入
pVirAddr	待操作数据的起始虚拟地址	输入
Size	待操作数据的大小。以字节为单位	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

AX_POOL_IncreaseRefCnt

【描述】

已知缓存块的 BlockID，对其进行引用计数执行加 1 操作。

【语法】

```
AX_S32 AX_POOL_IncreaseRefCnt (AX_BLK BlockId, MOD_TYPE_E ModId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入
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

AX_POOL_DecreaseRefCnt

【描述】

已知缓存块的 BlockID，对其进行引用计数执行减 1 操作。

【语法】

```
AX_S32 AX_POOL_DecreaseRefCnt (AX_BLK BlockId, MOD_TYPE_E ModId);
```

【参数】

参数名称	描述	输入/输出
BlockId	缓存块 ID	输入
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

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_LogOpen

【描述】

打开系统 log 功能。

【语法】

```
AX_VOID AX_SYS_LogOpen ( ) ;
```

【参数】

参数名称	描述	输入/输出
无	无	无

【返回值】

返回值	描述
无返回值	

AX_SYS_LogClose

【描述】

关闭系统 log 功能。

【语法】

```
AX_VOID AX_SYS_LogClose();
```

【参数】

参数名称	描述	输入/输出
无	无	无

【返回值】

返回值	描述
无返回值	

AX_SYS_LogPrint

【描述】

打印输出 log 到 SetLogTarget API 所设置的 target，默认 target 是 syslog。

【语法】

```
AX_VOID AX_SYS_LogPrint(AX_S32 level, AX_CHAR *pFormat, ...);
```

【参数】

参数名称	描述	输入/输出
level	Log level 级别，0~5	输入
pFormat	Log 输出格式，类似 printf	输入

【返回值】

返回值	描述
无返回值	

AX_SYS_LogPrint_Ex

【描述】

打印输出 log 到 SetLogTarget API 所设置的 target，默认 target 是 syslog。与 AX_SYS_LogPrint 函数相比，该函数可为 Log 输出源指定 tag 和 id，tag 用于确认 Log 输出的模块名，id 为 Log 输出的模块数字标识（便于高效进行 Log 模块匹配）。

【语法】

```
AX_VOID AX_SYS_LogPrint_Ex(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，0~5	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

【返回值】

返回值	描述
无返回值	

AX_SYS_LogOutput

【描述】

打印输出 log 到指定的 target。

【语法】

```
AX_VOID AX_SYS_LogOutput(AX_LOG_TARGET_E target, AX_LOG_LEVEL_E level,  
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，是 fprintf，还是 syslog	输入
level	Log level 级别，0~5	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	

AX_SYS_LogOutput_Ex

【描述】

打印输出 log 到指定的 target。

【语法】

```
AX_VOID AX_SYS_LogOutput_Ex(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，是 fprintf，还是 syslog	输入
level	Log level 级别，0~5	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	

AX_SYS_SetLogLevel

【描述】

设置 log 输出级别，0~5 级。

【语法】

```
AX_S32 AX_SYS_SetLogLevel (AX_LOG_LEVEL_E level);
```

【参数】

参数名称	描述	输入/输出
level	Log level 级别，0~5	输入

【返回值】

返回值	描述
0	成功
-1	失败

AX_SYS_SetLogTarget

【描述】

设置 log 的输出目标。

【语法】

```
AX_S32 AX_SYS_SetLogTarget (AX_LOG_TARGET_E target);
```

【参数】

参数名称	描述	输入/输出
target	输出目标方式: fprintf 和 syslog	输入

【返回值】

返回值	描述
0	成功
-1	失败

AX_SYS_EnableTimestamp

【描述】

使能 log 的时间戳。

【语法】

```
AX_S32 AX_SYS_EnableTimestamp(AX_BOOL enable);
```

【参数】

参数名称	描述	输入/输出
enable	1 使能，0 去使能	输入

【返回值】

返回值	描述
无返回值	

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_S *pSrc,const AX_MOD_INFO_S *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

- 结构体实参使用前最好先清 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_S *pSrc,const AX_MOD_INFO_S *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

- 结构体实参使用前最好先清 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_S *pDest, AX_MOD_INFO_S *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

- 结构体实参使用前最好先清 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_S *pSrc, AX_LINK_DEST_S *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

- 结构体实参使用前最好先清 0，避免随机值影响。

5.7 模块工作频率 API

AX_SYS_CLK_SetLevel

【描述】

设置模块工作频率 level 值接口。

【语法】

```
AX_S32 AX_SYS_CLK_SetLevel (AX_SYS_CLK_LEVEL_E nLevel);
```

【参数】

参数名称	描述	输入/输出
nLevel	当前希望设置的模块工作频率 level 值	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

AX_SYS_CLK_GetLevel

【描述】

获取当前模块工作频率 level 值接口。

【语法】

```
AX_SYS_CLK_LEVEL_E AX_SYS_CLK_GetLevel (AX_VOID);
```

【参数】

参数名称	描述	输入/输出
NULL		NULL

【返回值】

返回值	描述
g_sys_clk_cur_level	当前模块工作频率 level 值

AX_SYS_CLK_Single_RateSet

【描述】

单独设置某一个模块工作频率的接口。

【语法】

```
AX_S32 AX_SYS_CLK_Single_RateSet(AX_SYS_CLK_ID_E clkId, AX_ULONG rate);
```

【参数】

参数名称	描述	输入/输出
clkId	需设置频率的模块 ID	输入
rate	以 1MHz 为单位的频率值	输入

【返回值】

返回值	描述
0	成功
非 0	失败，返回错误码

相关数据类型、数据结构定义如下：

6 数据结构

- AX_MOD_INFO_S：定义模块组通道结构体。
- AX_MOD_ID_E：定义模块 ID。
- AX_LOG_LEVEL_E：定义 log 级别。
- AX_LOG_TARGET_E：定义 log 输出目标。
- AX_POOL_FLOORPLAN_T：定义媒体子系统视频缓存池配置结构体。
- AX_POOL_CONFIG_T：定义视频缓存池配置结构体。
- 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：定义视频缓存池来源枚举类型。
- AX_PARTITION_INFO_T：定义视频缓存池的模块 ID 枚举类型。
- AX_CMM_PARTITION_INFO_T：定义视频缓存池的模块 ID 枚举类型。
- AX_SYS_CLK_LEVEL_E：定义模块工作频率 level 值枚举类型。
- AX_SYS_CLK_ID_E：定义需设置频率的模块 ID 枚举类型。

AX_MOD_INFO_S

【说明】

定义模块组通道结构体。

【定义】

```
typedef struct _MOD_INFO {  
  
    AX_MOD_ID_E  enModId; /*模块 ID*/  
  
    AX_S32        s32GrpId;    /*组 ID*/  
  
    AX_S32        s32ChnId;    /*通道 ID*/  
  
} AX_MOD_INFO_S;
```

AXERA CONFIDENTIAL FOR Sipeed

AX_MOD_ID_E

【说明】

定义模块 ID。

【定义】

```
typedef enum {
```

```
    AX_ID_MIN      = 0x00,
```

```
    AX_ID_ISP      = 0x01,
```

```
    AX_ID_CE       = 0x02,
```

```
    AX_ID_VO       = 0x03,
```

```
    AX_ID_VDSP     = 0x04,
```

```
    AX_ID_EFUSE    = 0x05,
```

```
    AX_ID_NPU      = 0x06,
```

```
    AX_ID_VENC     = 0x07,
```

```
    AX_ID_VDEC     = 0x08,
```

```
    AX_ID_JENC     = 0x09,
```

```
    AX_ID_JDEC     = 0x0a,
```

```
    AX_ID_SYS      = 0x0b,
```

```
    AX_ID_AENC     = 0x0c,
```

```
    AX_ID_IVPS     = 0x0d,
```

```
    AX_ID_MIPI     = 0x0e,
```

```
    AX_ID_ADEC     = 0x0f,
```

```
    AX_ID_DMA      = 0x10,
```

```
AX_ID_VIN      = 0x11,  
  
AX_ID_USER     = 0x12,  
  
AX_ID_BUTT,  
  
} AX_MOD_ID_E;
```

AXERA CONFIDENTIAL FOR Sipeed

AX_LOG_LEVEL_E

【说明】

定义 log 输出级别。

【定义】

```
typedef enum {  
  
    SYS_LOG_MIN          = -1,  
  
    SYS_LOG_EMERGENCY    = 0,  
  
    SYS_LOG_ALERT        = 1,  
  
    SYS_LOG_CRITICAL     = 2,  
  
    SYS_LOG_ERROR        = 3,  
  
    SYS_LOG_WARN         = 4,  
  
    SYS_LOG_NOTICE       = 5,  
  
    SYS_LOG_INFO         = 6,  
  
    SYS_LOG_DEBUG        = 7,  
  
    SYS_LOG_MAX  
  
} AX_LOG_LEVEL_E;
```

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]; /*公共缓存池数组*/  
  
} AX_POOL_FLOORPLAN_T;
```

AXERA CONFIDENTIAL FOR SPEED

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; /*Metadata 大小*/  
  
    AX_U64 BlkSize; /*Block 大小*/  
  
    AX_U32 BlkCnt; /*Block 个数, range:(0,256]*/  
  
    AX_BOOL IsMergeMode; /*logically merged with common pool, make common  
                           pool bigger*/  
  
    AX_POOL_CACHE_MODE_E CacheMode; /*缓存池映射类型*/  
  
    AX_S8 PartitionName[MAX_PARTITION_NAME_LEN]; /*缓存池所在 CMM 分段名, 设为 NULL  
                                                  则默认 anonymous 分段*/  
  
} AX_POOL_CONFIG_T;
```


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 {  
  
    POOL_REMAP_MODE_NONCACHE = 0, /*non-cached 类型*/  
  
    POOL_REMAP_MODE_CACHED = 1,  /*cached 类型*/  
  
    POOL_REMAP_MODE_BUTT  
  
} AX_POOL_CACHE_MODE_E;
```

AXERA CONFIDENTIAL FOR Sipeed

AX_POOL_SOURCE_E

【说明】

定义视频缓存池来源枚举类型。

【定义】

```
typedef enum {  
  
    POOL_SOURCE_COMMON = 0, /*公共缓存池*/  
  
    POOL_SOURCE_PRIVATE = 1, /*私有缓存池*/  
  
    POOL_SOURCE_USER = 2, /*用户缓存池*/  
  
    POOL_SOURCE_BUTT  
  
} AX_POOL_SOURCE_E;
```

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; /*分段起始物理地址*/  
  
    AX_U32  SizeKB; /*分段内存大小，以 KB 为单位*/  
  
    AX_S8   Name[MAX_PARTITION_NAME_LEN]; /*分段名称*/  
  
} AX_PARTITION_INFO_T;
```

AXERA CONFIDENTIAL FOR Sipeed

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; /*分段个数，范围:1~MAX_PARTITION_COUNT*/  
    AX_PARTITION_INFO_T PartitionInfo[MAX_PARTITION_COUNT]; /*分段详细配置*/  
} AX_CMM_PARTITION_INFO_T;
```

AXERA CONFIDENTIAL FOR Speed

AX_LINK_DEST_S

【说明】

定义 CMM 内存分段配置信息。

【定义】

```
typedef struct axAX_LINK_DEST_S{  
    AX_U32 u32DestNum; /*通过 Link 绑定的数据接收者个数*/  
    AX_MOD_INFO_S astDestMod[AX_LINK_DEST_MAXNUM]; /*数据接收者模块组通道信息*/  
}AX_LINK_DEST_S;
```

AX_SYS_CLK_LEVEL_E

【说明】

定义模块工作频率 level 值。

【定义】

```
typedef enum {  
  
    AX_SYS_CLK_HIGH_MODE = 0,  
  
    AX_SYS_CLK_HIGH_HOTBALANCE_MODE = 1,  
  
    AX_SYS_CLK_MID_MODE = 2,  
  
    AX_SYS_CLK_MID_HOTBALANCE_MODE = 3,  
  
    AX_SYS_CLK_MAX_MODE = 4,  
  
} AX_SYS_CLK_LEVEL_E;
```

AX_SYS_CLK_ID_E

【说明】

定义需设置频率的模块 ID。

【定义】

```
typedef enum {  
  
    AX_CPU_CLK_ID = 0,  
  
    AX_BUS_CLK_ID = 1,  
  
    AX_NPU_CLK_ID = 2,  
  
    AX_ISP_CLK_ID = 3,  
  
    AX_MM_CLK_ID = 4,  
  
    AX_VPU_CLK_ID = 5,  
  
    AX_SYS_CLK_MAX_ID = 6,  
  
} AX_SYS_CLK_ID_E;
```

7 错误码

内存分配 API 错误码如下表所示：

表7-1 内存分配 API 错误码列表

错误代码	宏定义	描述
0x800B000A	AX_ERR_CMM_ILLEGAL_PARAM	参数超出合法范围。
0x800B000B	AX_ERR_CMM_NULL_PTR	参数空指针错误。
0x800B0010	AX_ERR_CMM_NOTREADY	系统未初始化。
0x800B0018	AX_ERR_CMM_NOMEM	CMM 内存不足。
0x800B0080	AX_ERR_CMM_MMAP_FAIL	映射虚拟地址失败。
0x800B0081	AX_ERR_CMM_MUNMAP_FAIL	去映射虚拟地址失败。
0x800B0082	AX_ERR_CMM_FREE_FAIL	释放 CMM 内存失败。
0x800B0083	AX_ERR_CMM_UNKNOWN	未知错误。

内存池管理 API 错误码如下表所示：

表7-2 内存池管理 API 错误码列表

错误代码	宏定义	描述
0x800B010A	AX_ERR_POOL_ILLEGAL_PARAM	参数超出合法范围。
0x800B010B	AX_ERR_POOL_NULL_PTR	参数空指针错误。
0x800B0110	AX_ERR_POOL_NOTREADY	缓存池未配置。
0x800B0115	AX_ERR_POOL_PERM	操作不允许。
0x800B0117	AX_ERR_POOL_UNEXIST	视频缓存池不存在。
0x800B0118	AX_ERR_POOL_NOMEM	缓存池内存分配失败。

错误代码	宏定义	描述
0x800B0180	AX_ERR_POOL_MMAP_FAIL	缓存池映射失败。
0x800B0181	AX_ERR_POOL_MUNMAP_FAIL	缓存池去映射失败。
0x800B0182	AX_ERR_POOL_BLKFREE_FAIL	缓存块内存释放失败。

时间管理 API 错误码如下表所示：

表7-3 时间管理 API 错误码列表

错误代码	宏定义	描述
0x800B020A	AX_ERR_PTS_ILLEGAL_PARAM	参数超出合法范围。
0x800B020B	AX_ERR_PTS_NULL_PTR	参数空指针错误。
0x800B0210	AX_ERR_PTS_NOTREADY	系统未初始化。
0x800B0215	AX_ERR_PTS_PERM	操作不允许。

链路管理 API 错误码如下表所示：

表7-4 链路管理 API 错误码列表

错误代码	宏定义	描述
0x800B030A	AX_ERR_LINK_ILLEGAL_PARAM	参数超出合法范围。
0x800B030B	AX_ERR_LINK_NULL_PTR	参数空指针错误。
0x800B0310	AX_ERR_LINK_NOTREADY	系统未初始化。
0x800B0314	AX_ERR_LINK_NOT_SUPPORT	接口不支持该操作。
0x800B0315	AX_ERR_LINK_NOT_PERM	操作不被允许。
0x800B0317	AX_ERR_LINK_UNEXIST	链路关系不存在。
0x800B0380	AX_ERR_LINK_TABLE_FULL	链路关系表已满。
0x800B0381	AX_ERR_LINK_TABLE_EMPTY	链路关系表已空。
0x800B0382	AX_ERR_LINK_UNKNOWN	未知错误。

模块工作频率 API 错误码如下表所示：

表7-5 模块工作频率 API 错误码列表

错误代码	宏定义	描述
0x800B050A	AX_ERR_CLK_ILLEGAL_PARAM	参数超出合法范围。
0x800B0510	AX_ERR_CLK_NOTREADY	系统未初始化。

AXERA CONFIDENTIAL FOR Sipeed

8.1 缓存池

8 调试信息

【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-----
version:V0.29.0 build:Mar  9 2022 10:12:50

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

PoolId      0      1      2      3      4
MetaSize    10240   10240   10240   10240   10240
BlkSize      718848   3214080 5496320 6274560 8220160
BlkCnt       15      15      13      15      5

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

PoolId IsComm IsCache Partition PhysAddr  MetaSize BlkSize  BlkCnt  FreeCnt
0      1      0      anonymous 0x8014A000 12288    720896  15      7

BlockId ISP  IVPS  VO    VENC  JENC  VDEC  JDEC  USER
0      1      0      0      0      0      0      0      0
1      1      0      0      0      0      0      0      0
2      0      0      0      1      0      0      0      0
3      0      0      0      1      0      0      0      0
4      1      0      0      0      0      0      0      0
5      1      0      0      0      0      0      0      0
6      0      1      0      0      0      0      0      0
```

```
7      1      0      0      0      0      0      0      0
```

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

```
1      1      0      anonymous 0x80BC7000  12288     3215360  15      9
```

```
BlockId ISP      IVPS   VO      VENC     JENC     VDEC     JDEC     USER
```

```
0      1      0      0      0      0      0      0      0
```

```
1      1      0      0      0      0      0      0      0
```

```
2      1      0      0      0      0      0      0      0
```

```
3      1      0      0      0      0      0      0      0
```

```
4      1      0      0      0      0      0      0      0
```

```
6      0      0      0      1      0      0      0      0
```

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

```
2      1      0      anonymous 0x839F3000  12288     5496832  13      11
```

```
BlockId ISP      IVPS   VO      VENC     JENC     VDEC     JDEC     USER
```

```
1      2      0      0      0      0      0      0      0
```

```
2      1      0      0      0      0      0      0      0
```

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

```
3      1      0      anonymous 0x87E40000  12288     6275072  15      7
```

```
BlockId ISP      IVPS   VO      VENC     JENC     VDEC     JDEC     USER
```

```
0      1      0      0      0      0      0      0      0
```

```
1      1      0      0      0      0      0      0      0
```

```
2      1      0      0      0      0      0      0      0
```

```
3      1      0      0      0      0      0      0      0
```

```
5      1      0      0      0      0      0      0      0
```

```
6      0      1      0      0      0      0      0      0
```

```

7      0      1      0      0      0      0      0      0
9      0      0      0      1      0      0      0      0

```

```

PoolId  IsComm  IsCache  Partition  PhysAddr  MetaSize  BlkSize  BlkCnt  FreeCnt
4        1        0      anonymous  0x8D831000  12288      8220672   5        1

```

```

BlockId ISP      IVPS      VO      VENC      JENC      VDEC      JDEC      USER
0        1        0        0        0        0        0        0        0
1        1        0        0        0        0        0        0        0
2        1        0        0        0        0        0        0        0
3        1        0        0        0        0        0        0        0

```

【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
VENC VERSION		SDK 版本信息。
COMMON POOL CONFIG (公共缓存池配置)	PoolId	公共缓存池 ID。
	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 对齐后）。

参数		描述
	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	缓存池内空闲缓存块的个数。
	BlockId	缓存池内缓存块的 ID。
	ISP/IVPS/VO /VENC/JENC /VDEC/JDEC /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。 <ul style="list-style-type: none">• 0: 没占用。• 非 0: 占用次数。

如何查看 CMM 虚拟地址数据

9 附录

CMM 实现的原理是在 Linux kernel 中申请一块 reserved memory，然后再此基础之上再次进行内存管理，在逻辑上会分配若干个内存池 Pool，包含专用内存池 Pool 和通用内存池 Pool，上层应用向这些内存池 Pool 申请内存空间。

Cmm 管理的内存映射方式与内核管理的内存方式不一样，使用 gdb 调试无法直接查看 cmm 虚拟地址数据，需要通过其他途径查看。

1. AX_S32 [AX_SYS_MemAlloc](#)(AX_U64 *phyaddr, AX_VOID **pviraddr, AX_S32 size, AX_S32 align, AX_S8 *token)

AX_S32 [AX_SYS_MemAllocCached](#)(AX_U64 *phyaddr, AX_VOID **pviraddr, AX_S32 size, AX_S32 align, AX_S8 *token)

通过这两个接口申请内存，如果第二个参数 pviraddr 用户输入不为 NULL，则虚拟地址和物理地址是同步带出的，用户释放 CMM 内存时需要填入物理地址和虚拟地址，用户代码需要匹配记录这两个地址。

2. GDB 调试程序时，出现问题或者在断点处，需要查看 CMM 虚拟地址数据，需要查找代码中记录的对应的物理地址。
3. 通过 SDK 发布包中 package\tools\ax_lookat 工具查看物理地址数据。