AX 内存使用说明及优化指南

文档版本：**V1.0**

发布日期：**2024/01/25**

目 录

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

适用产品

AX620E 系列产品（AX630C、AX620Q）

适读人群

- 终端用户
- 售前
- 售后
- 技术人员

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
V0.1	2023/11/11	文档初版
V1.0	2024/01/25	修改文档格式；修改 arm32 例子支持

1 概述

在实际项目开发过程中，内存总容量和具体业务需求总会存在矛盾，尤其是 AX620Q 内置 DDR 只有 256MB，Linux 系统内存和 CMM 内存都会比较紧张。

本文重点介绍内存相关的分配和使用方式。在遇到总的内存不足无法满足 APP 或算法需求时，可以尝试参考本文介绍的方式进行调整及优化，以满足不同业务场景下内存的需求。

☞ 本文会涉及到 SDK 中关于内存管理的一些名词，在阅读本文前，建议先阅读《00 - AX SDK 使用说明.docx》、《20 - AX SYS API 文档.docx》中内存相关章节，有助于更好的理解本文。

2 内存空间划分

2.1 DDR 空间划分

SDK 将整个 DDR 内存空间划分成如下两块：

1. Linux 系统内存
2. 连续物理内存管理的 CMM（Contiguous Memory Model）

Linux 系统内存空间和 CMM 内存空间大小和分配方式，不同的产品形态下可能存在差异，可以根据实际需要进行调整，两部分空间总大小，必须要小于等于所使用的 DDR 总大小。

☞ 关于内存划分详细介绍，请参考《00 - AX SDK 使用说明.docx》的章节 4 分配 Memory 和存储空间。

AX620E SDK 中的参考分配方式如下：

- AX630C：共 2048MB DDR，其中 Linux 系统内存为 1024MB，CMM 内存大小为 1024MB。
- AX620Q：共 256MB DDR，其中 Linux 系统内存为 96MB，CMM 内存大小为 160MB。

☞ 以上分配方式仅供参考，不同版本可能有微调。实际分配大小以 SDK 中实际配置值为准。

2.2 调整 DDR 分配

不同的产品形态对于系统内存和 CMM 的使用需求可能会有所不同，通常 CMM 空间分配都比系统内存要大。对于 CMM 内存占用，视频分辨率越大，分流越多，视频帧缓存所需要的 CMM 空间就会越大。

如果出现内存不足的情况，可以先尝试调整 Linux 系统内存和 CMM 的空间分配比例，然后观察内存消耗情况，直至调整到最优比例为止。

以 AX620Q 为例(以下如无特殊说明，AX620Q 将以 ARM32 Kernel4.19 作为参考)，AX620Q 只有 256MB DDR 可用，若需调整 Linux 系统内存和 CMM 的空间分配，可参考修改如下文件：

- build/projects/AX620Q_nor_arm32_k419/*project.mak*，“OS_MEM_SIZE := 96 #MB”定义了 Linux 系统内存的大小，单位是 MB：

```
# linux OS memory config
OS_MEM_SIZE         := 96 #MB
```

- CMM 空间分配将**自动**根据 OS_MEM_SIZE 调整

```
CMM_START_ADDR      := $(call AddAddressMB, $(SYS_DRAM_BASE), $(OS_MEM_SIZE))
CMM_SIZE            := $(shell echo $$(($(SYS_DRAM_SIZE) - $(OS_MEM_SIZE))))
CMM_POOL_PARAM      := anonymous,0,$(strip $(CMM_START_ADDR)),$(CMM_SIZE)M
```

调整后，可以通过如下方式进行检查修改是否生效：

Linux 系统内存：通过 cat /proc/meminfo 节点，观察 MemTotal 大小

CMM 内存状态：通过 cat /proc/ax_proc/mem_cmm_info 节点，观察 total size

以 AX620Q 为例：

Linux 系统内存共分配 96M，Kernel 运行占用及 Kernel Reserved Memory 等系统使用内存大约 26MB，剩余内存 MemTotal 为大约为 70MB。

CMM 内存共分配 160MB，total size=163840KB(160MB)。

AX620Q SDK 默认内存分布及可用内存参考下图 2-1：

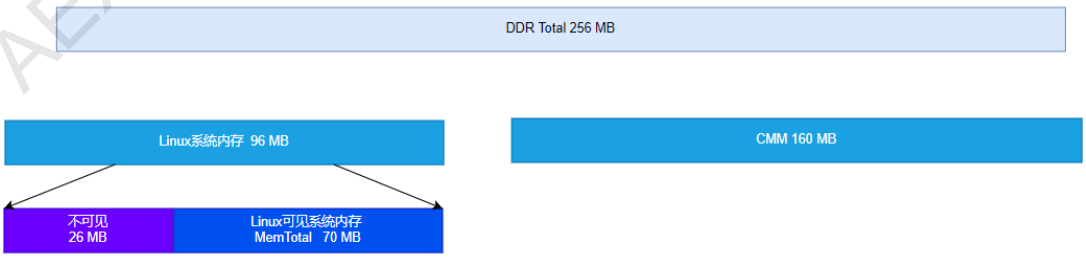

图2-1 AX620Q SDK 默认内存分布及可用内存示意图

3 Linux 系统内存

3.1 Linux 系统内存概述

Linux 系统内存部分，AX620Q SDK 中默认分配了 96MB，其中主要分两部分：

1. Kernel 运行占用、Reserved Memory 等共 26MB，该部分不可见
2. Linux Mem_total 共 70MB

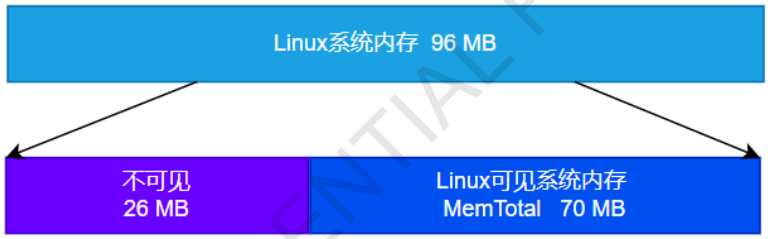

图3-1 AX620Q 参考 Linux 系统内存分配

3.2 Reserved Memory 内存

Reserved Memory 部分内存开机后不可见，具体占用为 Kernel Code&Data 和 DTS 中配置的 reserved-memory。详细配置情况，可以通过查看 dts 中的具体配置，kernel/linux/ linux-4.19.125/arch/arm/boot/dts/axera/AX620Q_nor_arm32_k419.dts，AX620Q 的 SDK 默认配置如下：

```
reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;
    ramoops_mem@41000000 {
        compatible = "ramoops";
        reg = <0x0 0x41000000 0x0 0x8000>;
        record-size = <0x4000>;
```

```
            console-size = <0x4000>;
        };
#ifdef SUPPORT_ATF
        atf_reserved: atf_memreserved {
            reg = <ATF_RESERVED_START_HI ATF_RESERVED_START_LO ATF_RESERVED_SIZE_HI
ATF_RESERVED_SIZE_LO>;
            no-map;
        };
#endif
    };
```

如果 Reserved Memory 部分空间异常，可以在 dts 中进行确认优化，去掉不必要的 Reserved 内存。

```
dmesg | grep Memory
  Memory: 91080K/98304K available (2988K kernel code, 194K rwdata, 1564K rodata, 232K
init, 336K bss, 7224K reserved, 0K cma-reserved)
```

加载的 ko 部分内存

系统开机阶段，会通过 build/projects/$project/soc/scripts/auto_load_all_drv.sh 脚本，一次性将运行时需要的 ko 全部加载进来，AX620E 的 SDK 默认加载的列表如下：

以 AX620Q 为例

```
build/projects/AX620E_nor/soc/scripts/auto_load_all_drv.sh
insmod /soc/ko/ax_sys.ko
insmod /soc/ko/ax_cmm.ko cmmpool=anonymous,0,0x46000000,160M
insmod /soc/ko/ax_pool.ko
insmod /soc/ko/ax_base.ko
insmod /soc/ko/ax_npu.ko
insmod /soc/ko/ax_vo.ko
insmod /soc/ko/ax_ivps.ko
insmod /soc/ko/ax_vpp.ko
insmod /soc/ko/ax_gdc.ko
insmod /soc/ko/ax_tdp.ko
insmod /soc/ko/ax_venc.ko
insmod /soc/ko/ax_jenc.ko
insmod /soc/ko/ax_mipi_rx.ko
insmod /soc/ko/ax_proton.ko
insmod /soc/ko/ax_audio.ko
```

！ 注意：

insmod /soc/ko/ax_cmm.ko 加载时的内存和地址，由 *project.mak* 定义的 OS 内存在编译时自动计算

3.3 可用系统内存

系统启动后，Linux 系统部分开销和驱动 ko 加载后的内存共消耗 34MB，因此留给 APP 使用的系统内存共有 62MB。因此，不运行任何应用，idle 状态下 Linux 系统内存使用情况分布参考下图：

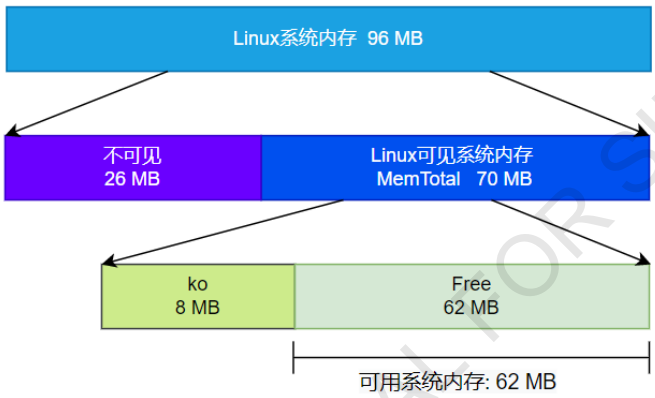

图3-2 AX620Q 默认 Linux 系统内存 Idle 状态使用情况分布示意图

Linux 系统内存的分配完成后，可以运行相关的 APP，然后通过 linux 的 meminfo 观察内存消耗情况，反复调整，直至将 Mem Available 控制在一个合理的范围内（需要考虑内存使用的峰值和系统稳定性，确保系统内存有一定的余量）。

4 CMM 内存

4.1 CMM 内存概述

AX620E 的 SDK 默认的 CMM 内存空间为 160MB，只划分了一个默认的 partition_0：anonymous，FRT Demo 运行时，基于该 partition 又划分了若干个 pool。

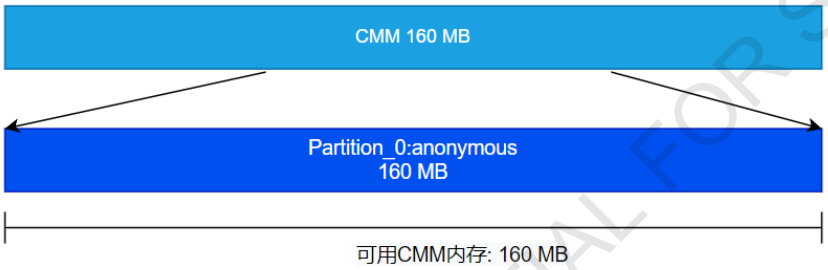

图4-1 CMM 内存 partition 划分情况

4.2 CMM 内存使用方式

SDK 将 CMM 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。CMM 空间层级划分示意图参考下图：

☞ 说明：

- 示意图中 M0、M1……表示 Block_0、Block_1……等 block 对应的 metadata。
- MetaSize 和 BlkSize 由用户创建 pool 时自定义，大小 4KB 对齐。
- 次部分内容详情请参考文档《20 - AX SYS API 文档.docx》中章节 3.1。

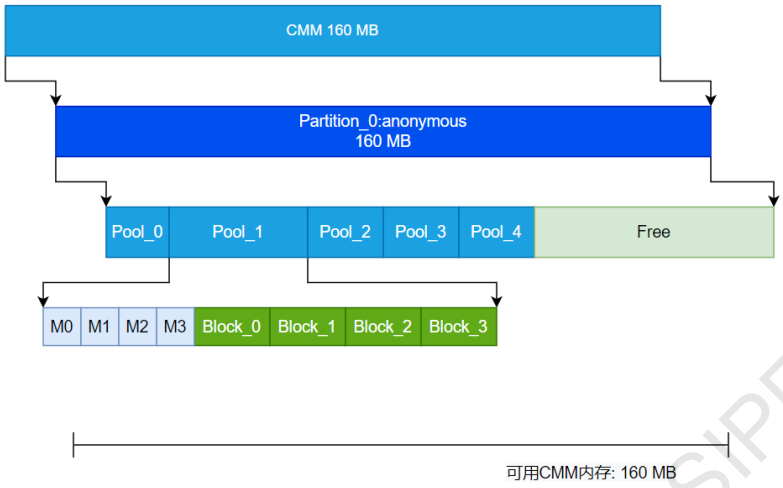

图4-2 CMM 空间层级划分示意图

4.3 CMM 内存池创建

合理的创建和使用内存池，可以让视频帧缓存通过 SDK 内存池的管理机制，高效的流转起来，最大程度的节省内存的使用。因此，CMM 内存池的配置方案尤为重要。

CMM 内存池的配置方案和具体项目的 pipeline 和业务流程紧密相关。因此，在创建内存池之前，首先需要对产品的方案有一个比较清晰的认识，能大概估算出每个 pool 的大小及所需的 block 数量。

☞ 说明：

关于内存池相关的说明请参考文档《20 - AX SYS API 文档.docx》中内存管理相关的章节。

接下来会描述各个模块需要的内存池 block 最小个数，然后基于具体业务的 pipeline 给出内存池基本配置方案供参考。

4.4 如何配置内存池参数

4.4.1 估算每个 pool 中 Block 数量

AX620E 平台典型的单摄 PPL 处理流程，大致上可以分为 IFE、AI-ISP、ITP 以及 IVPS 分流几个环节，Sensor 通过 online 模式接入 IFE，经 IFE、AI-ISP、ITP 处理，如果采用

GDC_ONLINE_VPP 模式，将经过 TDP(如需旋转)，GDC，1 个通道输出分别输出全尺寸的 YUV 到 VPP。

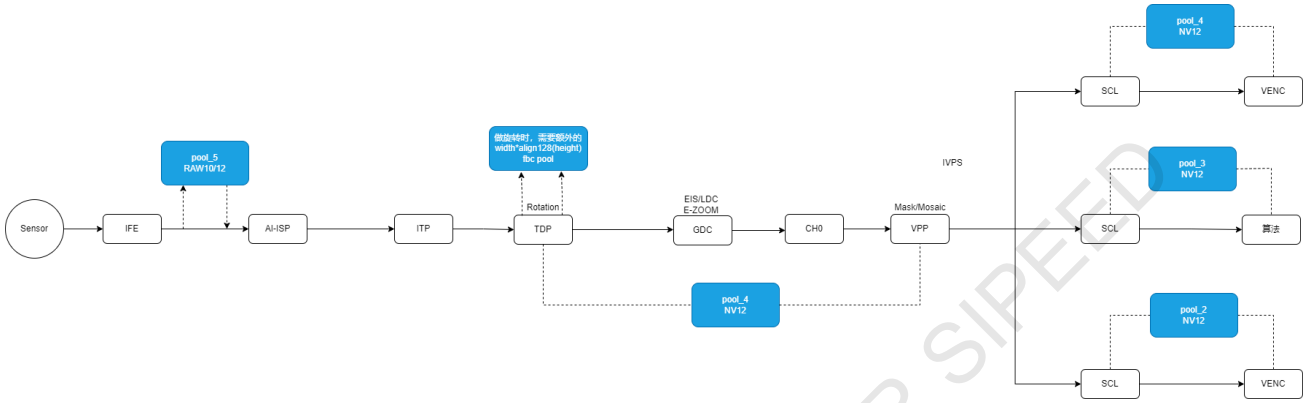

图4-3 内存池分配示意图

如上图 4-3 所示，online 模式共需要配置 5 个内存池：

1. Pool_5 为缓存 raw10/12 的私有内存池，用于 IFE 的输出和 AI-ISP 的输入帧缓存。为了节省内存，SDR 模式最小 block 数量建议配置为 3，HDR 需要乘 2（长帧、短帧各 3 个），配置为 6，建议配置为压缩模式(FBC: mode: AX_COMPRESS_MODE_LOSSY, level: 4)。

> ☞ 说明：
>
> 以上为 online 模式配置建议，如果使用 offline 模式，pool_5 中的 block 数量需要增加。非必要不推荐使用 offline 模式。

2. Pool_2 为缓存子码流 YUV 的公共内存池，用于 SCL 分流输出及所有后级模块使用的 YUV 帧缓存，最小 block 数量建议配置为 2，建议配置为压缩模式 (FBC: mode: AX_COMPRESS_MODE_LOSSY, level: 4)。

3. Pool_3 为缓存算法 YUV 的公共内存池，用于 SCL 分流输出及算法模块使用的 YUV 帧缓存，最小 block 数量建议配置为 2，**必须配置为非压缩模式**(None FBC: mode: AX_COMPRESS_MODE_NONE, level: 0)。

4. Pool_4 为缓存主码流及 ITP YUV 的公共内存池，用于 SCL 分流输出及所有多媒体模块使用的 YUV 帧缓存，最小 block 数量建议配置为 4，建议配置为 FBC 压缩模式(FBC: mode: AX_COMPRESS_MODE_LOSSY, level: 4)。

5. TDP 旋转内存，用于 TDP 旋转处理，需要的尺寸是 Sensor 分辨率 Width，以及 Height

的 128 对齐，最小 block 数量建议配置为 2，建议配置为 FBC 压缩模式(FBC: mode: AX_COMPRESS_MODE_LOSSY, level: 4)。

☞ 说明：

以上的 Pool_4 内存池和 TDP 旋转内存，在实际配置为时，建议均按照 TDP 旋转内存进行配置，可统一为同一个公共缓存池

因为对于匿名的公共内存池 block 总是从最小开始分配，如果小的 block 用完了，则向大的 block 申请。

模块	4M 单 Pipe 最小 Block 个数参考	说明
IFE	SDR:3 HDR:6	根据 Sensor 的 eRawType 配置(AX_RT_RAW10 /AX_RT_RAW12)申请内存池；FBC 模式
ITP/主码流 YUV/TDP 旋转	不需要 TDP 旋转: 4 需要 TDP 旋转: 7	AX_GDC_ONLINE_VPP 模式下，如需 TDP 旋转，建议以高 128 对齐创建内存池；FBC 模式
算法 YUV	2	根据算法实际尺寸配置；None FBC 模式
子码流 YUV	2	根据子码流实际尺寸配置；FBC 模式

☞ 说明：

- 上表是 frtdemo 运行 4M@30fps 下最低的配置，比这个配置再低可能会影响输出的帧率或者大量丢帧的情况。实际配置可能跟帧率/DDR/时钟/分流情况等有关，需要根据实际环境再调试优化。
- YUV 相关的 Block 个数也需要根据后面 YUV 业务处理流程需要缓存的大小酌情增加，比如 YUV 需要做智能算法需要缓存 10 帧，则需要在上面的基础上增加 10。
- 上层应用运行起来后，可以通过 cat /proc/ax_proc/pool 命令来查看 ax_pool 的分配和使用的情况，根据实际的使用情况来合理分配各个 Size 的 Block 个数。

4.4.2 计算 block size

每个 pool 的 BlockSize 可以根据需要缓存视频帧数据的尺寸（Stride 和 Height），图片的格式以及是否使能 FBC，通过 AX_VIN_GetImgBufferSize 这个 API 计算。

☞ 说明：

- 详细 API 说明请参考《20 - AX SYS API 文档.docx》。

- FBC 模式下，Stride 需要 128 对齐；None FBC 模式下，Stride 一般只需要 16 对齐即可，具体请参考对应模块的 Stride 要求

4.5 CMM 内存池优化

内存池的的状态，可以在程序运行时，可以使用 cat /proc/ax_proc/pool 命令来观察 pool 的 proc 信息，通过持续观察 FreeCnt 的值以及每个模块持有 block 的数量，来确认是否存在分配不合理的情况。例如图 4-4，pool_2 一共包含 3 个 block，当前时刻剩余 2 个空闲 block 可以被申请。如果某一个 pool 的 FreeCnt 长期持续在 2 个及以上，则可酌情对该 pool 的 block 总数进行裁剪。

```
--------------------------------------------------------------------------------------------------------------
PoolId  IsComm  IsCache Partition  PhysAddr    MetaSize  BlkSize   BlkCnt  FreeCnt  MinFreeCnt  IsDelayFree
2       1       0       anonymous  0x475E6000  4096      348160    3       2        1           0

Index   BlockId         VIN     IVPS    VO      VENC    JENC    VDEC    JDEC    SKEL    AI      AO      AENC    ADEC    AVS     USER    ReqSize
0       0x5E002000      0       1       0       0       0       0       0       0       0       0       0       0       0       0       331776
--------------------------------------------------------------------------------------------------------------
PoolId  IsComm  IsCache Partition  PhysAddr    MetaSize  BlkSize   BlkCnt  FreeCnt  MinFreeCnt  IsDelayFree
3       1       0       anonymous  0x476E8000  4096      1384448   2       1        0           0

Index   BlockId         VIN     IVPS    VO      VENC    JENC    VDEC    JDEC    SKEL    AI      AO      AENC    ADEC    AVS     USER    ReqSize
0       0x5E003000      0       0       0       0       0       0       0       1       0       0       0       0       0       1       1382400
--------------------------------------------------------------------------------------------------------------
PoolId  IsComm  IsCache Partition  PhysAddr    MetaSize  BlkSize   BlkCnt  FreeCnt  MinFreeCnt  IsDelayFree
4       1       0       anonymous  0x4798E000  4096      3096576   7       5        4           0

Index   BlockId         VIN     IVPS    VO      VENC    JENC    VDEC    JDEC    SKEL    AI      AO      AENC    ADEC    AVS     USER    ReqSize
0       0x5E004000      0       1       0       0       0       0       0       0       0       0       0       0       0       0       3064320

2       0x5E004002      1       1       0       0       0       0       0       0       0       0       0       0       0       0       3064320
--------------------------------------------------------------------------------------------------------------
```

图4-4 Pool 使用情况 proc 信息示意图

4.6 其他 CMM 内存消耗

4.6.1 Venc/Jenc 输出 Ringbuffer

创建编码通道的时候，每个编码通道需要为输出码流准备一个 ring buffer，用于缓存编码后的输出视频帧，输出 ring buffer 从 CMM 内存中申请，大小由 AX_VENC_CHN_ATTR_T 中的 u32BufSize 参数决定，以 byte 为单位。内存情况不紧张情况下，推荐计算公式为：Stride ×Height × 1.5，1.5 为比例系数。如果 CMM 内存非常紧张，可根据实际使用场景动态调整这个比例系数，最小不能小于 1/2。

！ 注意：

Ring buffer 分配过小 Venc 可能会出现输出 stream buffer 不足的情况，配置更低的系数需要实测观察。

4.6.2 Venc 参考帧 Ringbuffer

☞ 说明：

详细 API 说明请参考《21 - AX VENC API 文档.docx》的 AX_VENC_ATTR_T。

bDeBreathEffect	去呼吸效应开关，和参考帧 ringbuf 开关互斥。 AX_TRUE：打开去呼吸效应功能 AX_FALSE：关闭去呼吸效应功能 静态属性。
bRefRingbuf	参考帧 ringbuf 开关，和去呼吸效应开关互斥。 AX_TRUE：开启参考帧 ringbuf 功能，节省内存 AX_FALSE：关闭参考帧 ringbuf 功能 如果开启，会节省(w*h) byte 静态属性（仅对 H.264/H265 有效）

图4-5 参考帧 Ringbuffer 介绍

！ 注意：

开启参考帧 Ringbuffer 功能，大概会节省编码分辨率的 w*h 字节大小的内存。但是开启参考帧 Ringbuffer 时，将影响去呼吸效应，可能会影响到画质。需要根据实际需求决定是否打开此功能。

4.6.3 Audio CMM 内存开销

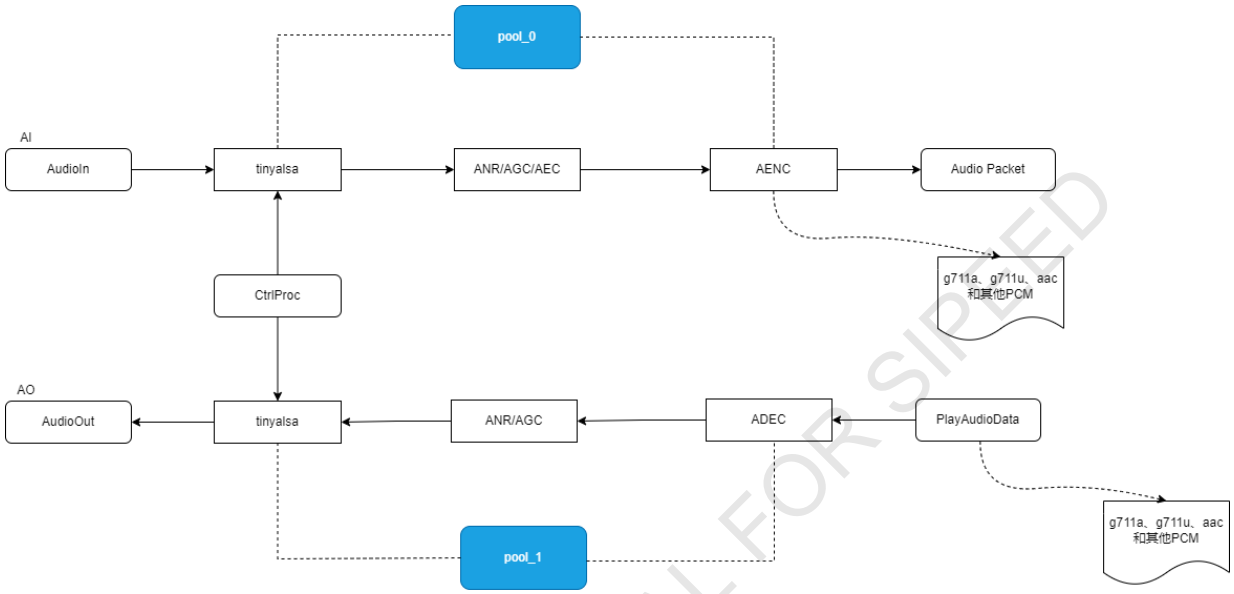

图4-6 Audio 内存池分配示意图

如上图 Audio 上下行各只有一个通道。Audio 上行，通过 tinyalsa 采集到声卡的 PCM 数据经过语音增加模块，输出到音频编码模块，并编码出对应的音频数据；Audio 下行，音频数据传输到音频解码模块，并解码出 PCM 输出，经过语音增加模块，通过 tinyalsa 输出到声卡。

模块	最小 Block 个数参考	最小 Block Size(KB)	说明
AI	2	4	Audio 输入，一般情况 Block Size 配置为 4KB 即可。 如果支持 AED，建议至少增加 2 个 Block
AO	34	4	Audio 输出，一般情况 Block Size 配置为 4KB 即可；如果是 OPUS 解码，建议配置为 38400 Bytes

4.7 CMM 内存使用情况

系统中全部 CMM 使用详情，可以通过 cat /proc/mem_cmm_info 节点来观察，name 为申请 CMM 内存是指定的 token。

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

```
--------------------SDK VERSION------------------
[Axera version]: ax_cmm V1.7.0_P1 Jan 29 2024 15:36:09
+---PARTITION: Phys(0x46000000, 0x4FFFFFFF), Size=163840KB(160MB),    NAME="anonymous"
 nBlock(Max=102, Cur=102, New=102, Free=0)  nbytes(Max=104222720B(101780KB,99MB),
Cur=104222720B(101780KB,99MB), New=104222720B(101780KB,           99MB), Free=0B(0KB,0MB))
Block(Max=21704704B(21196KB,20MB), Min=4096B(4KB,0MB), Avg=439703B(429KB,0MB))
   |-Block: phys(0x46000000, 0x46000FFF), cache =non-cacheable, length=4KB(0MB),    name="dmacfg"
   |-Block: phys(0x46001000, 0x46006FFF), cache =non-cacheable, length=24KB(0MB),    name="VPP_CMD0"
   |-Block: phys(0x46007000, 0x46007FFF), cache =non-cacheable, length=4KB(0MB),    name="VPP_CMD3"
   |-Block: phys(0x46008000, 0x46008FFF), cache =non-cacheable, length=4KB(0MB),    name="GDC_CMD3"
   |-Block: phys(0x46009000, 0x4600BFFF), cache =non-cacheable, length=12KB(0MB),    name="GDC_CMD0"
   |-Block: phys(0x4600C000, 0x4601FFFF), cache =non-cacheable, length=80KB(0MB),    name="TDP_CMD0"
   |-Block: phys(0x46020000, 0x46020FFF), cache =non-cacheable, length=4KB(0MB),    name="TDP_CMD3"
   |-Block: phys(0x46021000, 0x46030FFF), cache =non-cacheable, length=64KB(0MB),    name="venc_ko"
   |-Block: phys(0x46031000, 0x46040FFF), cache =non-cacheable, length=64KB(0MB),    name="venc_ko"
   |-Block: phys(0x46041000, 0x46041FFF), cache =non-cacheable, length=4KB(0MB),    name="venc_ko"
   |-Block: phys(0x46042000, 0x46051FFF), cache =non-cacheable, length=64KB(0MB),    name="jenc_ko"
   |-Block: phys(0x46052000, 0x46061FFF), cache =non-cacheable, length=64KB(0MB),    name="jenc_ko"
   |-Block: phys(0x46062000, 0x46062FFF), cache =non-cacheable, length=4KB(0MB),    name="jenc_ko"
   |-Block: phys(0x46063000, 0x46082FFF), cache =non-cacheable, length=128KB(0MB),    name="jenc_ewl"
   ……
   ……
   ……
   |-Block: phys(0x4C350000, 0x4C356FFF), cache =non-cacheable, length=28KB(0MB),
name="vo_layer0_dw_hw_cfg"
   |-Block: phys(0x4C357000, 0x4C35DFFF), cache =non-cacheable, length=28KB(0MB),
name="vo_layer1_dw_hw_cfg"

---CMM_USE_INFO:
 total size=163840KB(160MB),used=101780KB(99MB + 404KB),remain=62060KB(60MB +
620KB),partition_number=1,block_number=102
```

5 FAQ