AX 内存使用说明及优化指南

文档版本：**V0.2**

发布日期：**2022/04/04**

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

爱芯 AX620A 和 AX620U

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*/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
V0.2	2022-4-4	档初版

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有差距，尤其是 620U 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《AX SDK 使用说明》、《AX SYS API 文档》、《AX620U 软件差异说明》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《AX SDK 使用说明》的章节 4 分配 Memory 和存储空间。

620U SDK 中的参考分配方式如下：

- 620U：共 256MB DDR，其中 Linux 系统内存为 78MB，CMM 内存大小为 178MB。

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，视频帧分辨率越大，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 AX620U 为例，AX620U 只有 256MB DDR 可用，若需调整 Linux 系统内存和 CMM 的空间分配，可参考如下两个文件做设置：

- /boot/uboot/u-boot-2020.04/include/configs/*ax620u_nand.h*，CONFIG_BOOTARGS 中的“mem=78M”定义了 Linux 系统内存的大小，单位是 MB：

```
#define CONFIG_BOOTARGS "mem=78M console=ttyS0,115200n8
earlyprintk=dw_uart,board_id=0,noinitrd\
root=/dev/mtdblock6 rw rootfstype=ubifs ubi.mtd=6,2048 root=ubi0:rootfs init=/linuxrc \
mtdparts=spi4.0:1M(spl),1536K(uboot),1M(kernel-dtb),32M(kernel),512K(param),192M(rootfs)"
```

- /rootfs/rootfs_620u/rootfs/soc/scripts/auto_load_all_drv.sh，cmmpool 里的 0x44E00000 定义的是 CMM 空间的起始地址，此地址应等于上面 SYS DDR 结束地址的值 +1，0x44E00000 = 0x44Dfffff + 1；178M 是 CMM 空间的大小，0x50000000 - 0x44E00000 = 0xB20 0000 = 186646528，186646528/1024/1024 = 178：

```
#!/bin/sh
insmod /soc/ko/ax_osal.ko
insmod /soc/ko/ax_sys.ko
insmod /soc/ko/ax_cmm.ko cmmpool=anonymous,0,0x44E00000,178M
insmod /soc/ko/ax_npu.ko
```

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 620U 为例：

Linux 系统内存共分配 78M，Kernel Reserved Memory 大约 10MB，剩余内存 MemTotal 为大约为 68MB。

CMM 内存共分配 178MB，total size=182272KB(178MB)。

620U SDK 默认内存分布及可用内存参考下图 2-1：

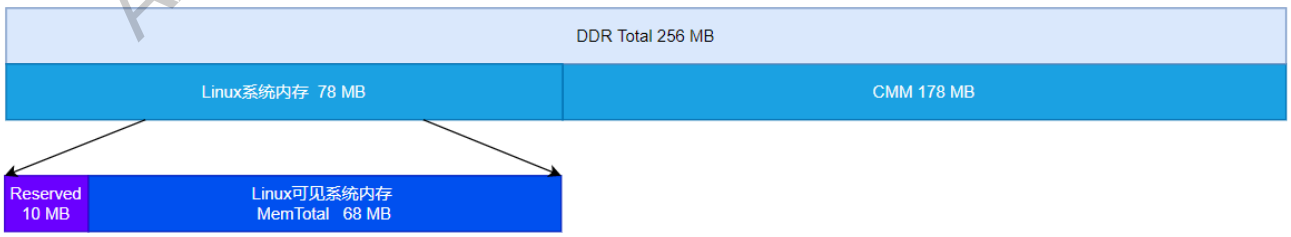

图2-1 620U SDK 默认内存分布及可用内存示意图

3 Linux 系统内存

3.1 Linux 系统内存概述

Linux 系统内存部分，620U SDK 中默认分配了 78MB，其中主要分两部分：

1. Kernel 运行占用及 Reserved Memory 共 10MB，该部分不可见
2. Linux Mem_total 共 68MB

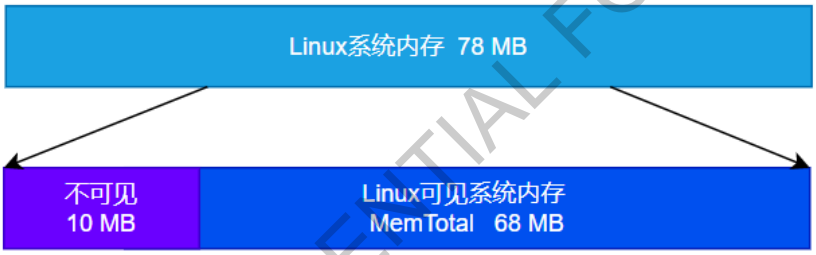

图3-1 620U 参考 Linux 系统内存分配

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/AX620U_nand.dts，620U 的 SDK 默认配置如下：

```
reserved-memory {
    #address-cells = <1>;
    #size-cells = <1>;
    ranges;
    //ramoops_mem: for DEBUG, DUMP crash log
    ramoops_mem@43000000{
        compatible = "ramoops";
        reg = <0x43000000 0x20000>;
        record-size     = <0x20000>;
    };
```

```
        //ax_memory_dump: for kernel sysdump
        ax_memory_dump@44100000 {
            compatible = "ax_memory_dump";
            reg = <0x44100000 0x1000>;
            size     = <0x1000>;
        };
    };
```

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。

☞ 注意：620U 已经将 cma reserved 的 256MB 内存去掉，A 目前暂时还保留。

```
        linux,cma {
            compatible = "shared-dma-pool";
            reusable;
            size = <0x10000000>;
            linux,cma-default;
        };
```

3.3 加载的 ko 部分内存

系统开机阶段，会通过 rootfs/rootfs_620u/rootfs/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，620U 的 SDK 默认加载的列表如下：

```
insmod /soc/ko/ax_osal.ko
insmod /soc/ko/ax_sys.ko
insmod /soc/ko/ax_cmm.ko cmmpool=anonymous,0,0x44E00000,178M
insmod /soc/ko/ax_npu.ko
insmod /soc/ko/ax_sdio_host.ko
insmod /soc/ko/ax_pool.ko
insmod /soc/ko/ax_proton.ko
insmod /soc/ko/ax_mipi.ko
insmod /soc/ko/ax_venc.ko
insmod /soc/ko/ax_jenc.ko
insmod /soc/ko/ax_vdec.ko
insmod /soc/ko/ax_audio.ko
insmod /soc/ko/ax_vo.ko
```

```
insmod /soc/ko/ax_ivps.ko
insmod /soc/ko/ax_gdc.ko
insmod /soc/ko/ax_hwinfo.ko
insmod /soc/ko/ax_pcal6416a.ko
insmod /soc/ko/ax_logctl.ko
insmod /soc/ko/ax_dma_hal.ko
insmod /soc/ko/ax_ddr_retention.ko

insmod /soc/ko/ax_ives.ko
```

3.4 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 23MB，因此留给 APP 使用的系统内存共有 45MB。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：

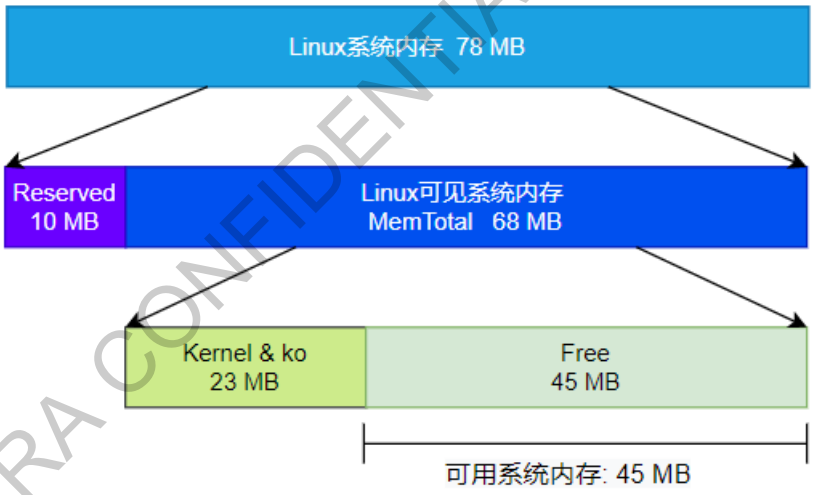

图3-2 620U 默认 Linux 系统内存 Idle 状态使用情况分布示意图

Linux 系统内存的分配完成后，可以运行具体的 APP，然后通过 linux 的 meminfo 观察内存消耗情况，反复调整，直至将 MemAvailable 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

620U 的 SDK 默认的 CMM 内存空间为 178MB，只划分了一个默认的 partition_0：anonymous，IPC Demo 运行时，基于该 partition 又划分了若干个 pool。

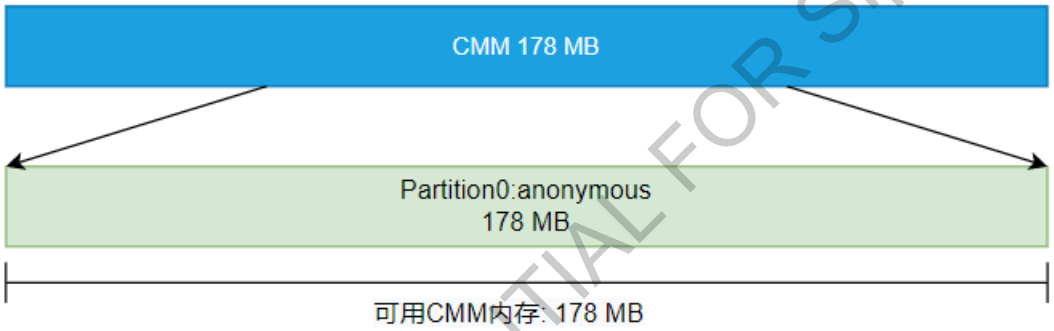

图4-1 CMM 内存 partition 划分情况

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：

- 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《06 – AX SYS API 文档.docx》中章节 3.1。

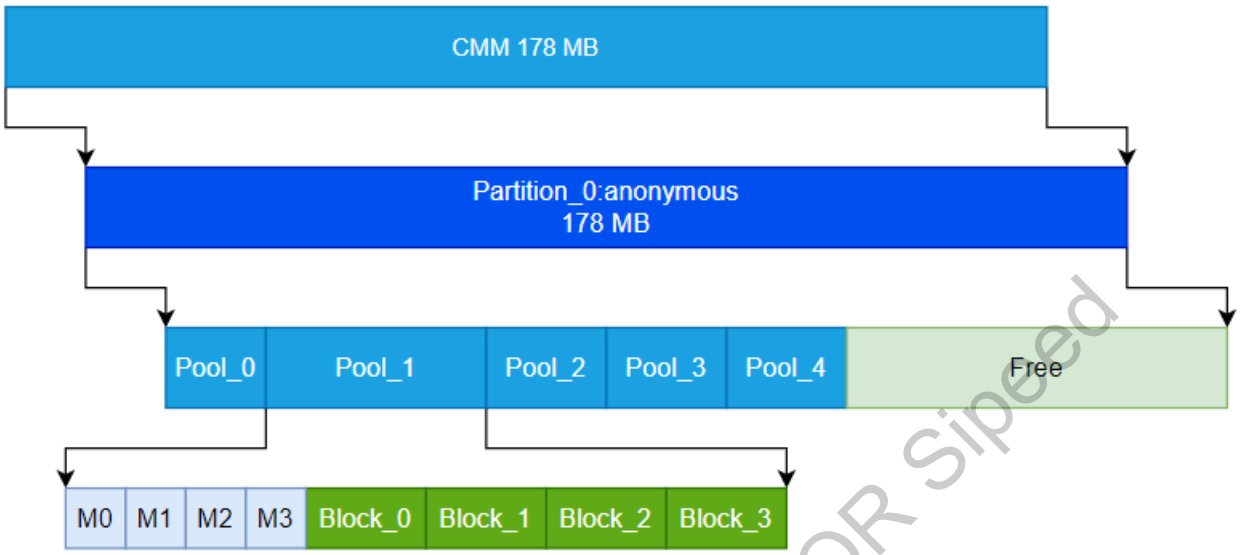

图4-2 CMM 空间层级划分示意图

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。

☞ 关于内存池相关的说明请参考文档《06 – 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

AX620 平台单摄典型的 ISP 处理流程，大致上可以分为 IFE、AI-ISP、ITP 三个环节，

sensor 通过 online 模式接入 IFE，经 IFE、AI-ISP、ITP 处理，3 个通道输出分别输出 3 个尺寸的 YUV。

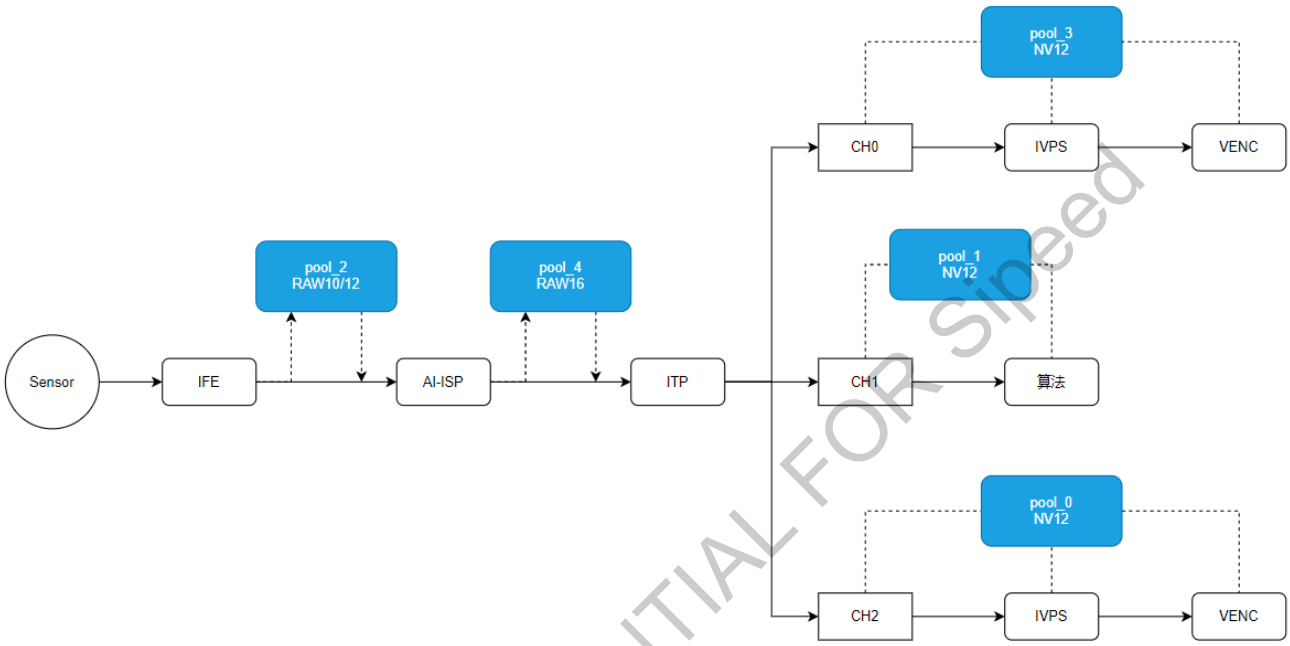

图4-3 内存池分配示意图

如上图 4-3 所示，online 模式共需要配置 5 个内存池：

1. Pool_2 为缓存 raw10/12 的内存池，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。

☞ 以上为 online 模式配置建议，如果使用 offline 模式，pool_2 中的 block size 需要增加。非必要不推荐使用 offline 模式。

2. Pool_4 为缓存 raw16 的内存池，用于 AI-ISP 的输出和 ITP 的输入帧缓存。为节省内存最小 block 数量建议配置为 4。

3. Pool_3 为缓存 CHN0 YUV 的内存池，用于 ITP CHN0 输出及所有后级模块使用的 YUV 帧缓存。

4. Pool_1 为缓存 CHN1YUV 的内存池，用于 ITP CHN1 输出及所有后级模块使用的 YUV 帧缓存。

5. Pool_0 为缓存 CHN2 YUV 的内存池，用于 ITP CHN2 输出及所有后级模块使用的 YUV

帧缓存。

YUV 后处理的多个模块可以复用 ITP 分流后的 YUV 内存池，如果需要输出更小尺寸的图片，可以申请与之匹配的内存池 Block，比如用到 IVPS 缩小输出时。因为对于匿名的内存池 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	单 Pipe 最小 Block 个数	说明
IFE	SDR:3 HDR:6	Raw10，Raw12 等
NPU	4	Raw16，这个模块单个 Block Size 最大
ITP-CH0	3	要>=AX_VIN_CHN_ATTR_T. tChnAttr[0].nDepth 的配置，如果没有使能此 Channel，不需要分配
ITP-CH1	3	要>=AX_VIN_CHN_ATTR_T. tChnAttr[1].nDepth 的配置，如果没有使能此 Channel，不需要分配
ITP-CH2	3	要>=AX_VIN_CHN_ATTR_T. tChnAttr[2].nDepth 的配置，如果没有使能此 Channel，不需要分配
Other	x	其他尺寸根据需要配置

☞ 说明：

☞ 上表是最低的配置，比这个配置再低会影响输出的帧率或者大量丢帧的情况。

☞ ITP-CHx 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 Block 的个数。

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。

☞ 详细 API 说明请参考《06 – AX SYS API 文档.docx》 。

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_0 一共包含 3 个 block，当前时刻剩余 1 个空闲 block 可以

被申请。如果某一个 pool 的 free cnt 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。

```
----COMMON POOL CONFIG--------------------------------------------------------------------
PoolId            0         1         2         3         4

MetaSize       5120      5120      5120      5120      5120

BlkSize      677376   3214080   5496320   6274560   8220160

BlkCnt            3         6         6         3         4

----ALL POOL INFO-------------------------------------------------------------------------
PoolId  IsComm  IsCache Partition  PhysAddr     MetaSize  BlkSize   BlkCnt  FreeCnt
0       1       0       anonymous  0x44F4A000   8192      679936    3       1


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

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

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_S 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 3/4。

! 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。

4.7 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。

```
/ # cat /proc/mem_cmm_info
+---PARTITION: Phys(0x44E00000, 0x4FFFFFFF), Size=182272KB(178MB),    NAME="anonymous"
 nBlock(Max=79, Cur=79, New=81, Free=2)  nbytes(Max=132632576B(129524KB,126MB),
Cur=132632576B(129524KB,126MB), New=132657152B(129548KB,126MB), Free=24576B(24KB,0MB))
Block(Max=33030144B(32256KB,31MB), Min=4096B(4KB,0MB), Avg=1593552B(1556KB,1MB))
   |-Block: phys(0x44E00000, 0x44E3FFFF), cache =non-cacheable, length=256KB(0MB),    name="venc_ko"
   ……
   ……
   ……
   |-Block: phys(0x4C975000, 0x4CC7CFFF), cache =non-cacheable, length=3104KB(3MB),    name="venc_fifo_1.
"


---CMM_USE_INFO:
 total size=182272KB(178MB),used=129524KB(126MB + 500KB),remain=52748KB(51MB +
524KB),partition_number=1,block_number=79
```

5 FAQ