SigmaStar Camera Memory Layout 介绍

© 2019 SigmaStar Technology Corp. All rights reserved.

SigmaStar Technology makes no representations or warranties including, for example but not limited to, warranties of merchantability, fitness for a particular purpose, non-infringement of any intellectual property right or the accuracy or completeness of this document, and reserves the right to make changes without further notice to any products herein to improve reliability, function or design. No responsibility is assumed by SigmaStar Technology arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others.

SigmaStar is a trademark of SigmaStar Technology Corp. Other trademarks or names herein are only for identification purposes only and owned by their respective owners.

REVISION HISTORY

Revision No.	Description	Date
0.1	• {Initial release}	{08/27/2019}

Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

TABLE OF CONTENTS

REVISION HISTORY i
TABLE OF CONTENTS ii
1. 概述 1
1.1. 概述 1
2. MMAP 2
2.1. MMAP 介绍 2
2.2. MMAP 在系统打包阶段的处理 3
2.3. Bootarg 内存配置 4
2.4. MMAP 在系统初始化阶段的处理 4
2.5. SCA 工具修改 MMAP 6
2.6. Linux 可用内存计算 10
2.7. 一些特殊 layout 举例 10
3. MMA HEAP 12
3.1. 配置以及 Dump 12
3.2. 物理地址以及 MIU 地址 12

Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

1. 概述

1.1. 概述

本文档介绍了 SStar 平台上特有的 memory layout 方式，以及与 mma heap 相关的一些基本概念。

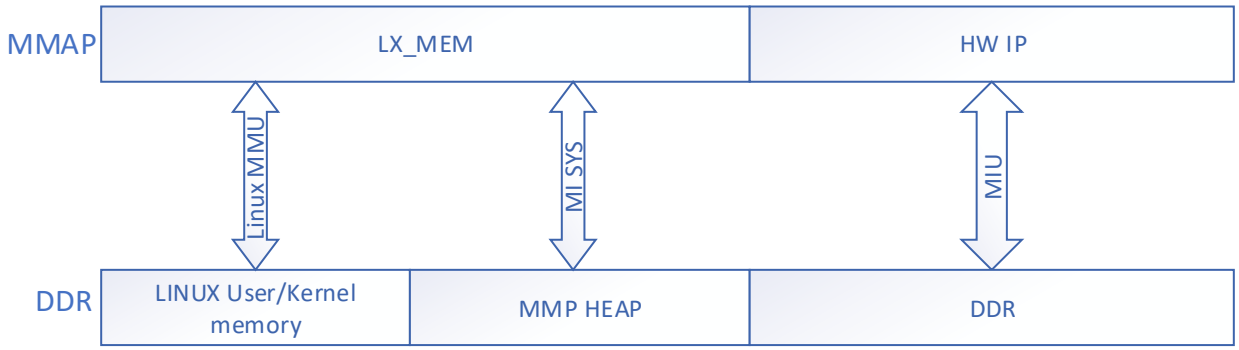
Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

2. MMAP

2.1. MMAP 介绍

MMAP 是 memory map 的简写，它是整个 SOC 系统内存 Layout，如下图所示，MMAP 内配置的内存分成两个部分。

1) Hardware IP 私有的内存。
2) LX_MEM，Linux Kernel 能使用的内存。



每块内存的使用者必须保证自己不得越界，例如 Linux kernel 使用的内存由 Linux 内部机制保证，HW IP 使用的内存由硬件保证。

LX_MEM 有可能有好几块，例如某些 SOC 上会有双通道 DDR，每个 DDR 上面会各自分配一块 LX_MEM。还有某些特别的情况，一颗 DDR 上面可能会分配多个 LX_MEM。多个 LX_MEM 的命名规则为 LX_MEM1、LX_MEM2，以此类推。

在 LX_MEM 中由拉出来一块给 MMA Heap 使用，类似于 Linux 的 cma，这部分内存是给 MI SYS 通过软件的方式动态分配给 HW IP 使用的内存池，而与 MMAP 中给 Hardware IP 私有的内存不同的，后者是静态分配的，它在系统初始化的时候就已经分配好，只有此 IP 独占。

需要注意的是，MMA Heap 以及 HW IP Layout 分配出来的内存在物理上是连续的，LX_MEM 中 Linux Kernel 使用的部分有可能会经过 LINUX MMU 转成虚拟地址，在物理上不连续。

每一颗 DDR 上都有对应的内存控制器(MIU)，所有的 HW IP，不论是静态分配或者动态分配的内存，都会通过 MIU 才能访问到 DDR。

由于 SOC 的版本众多，因而 MMAP 的版本也有很多，目前是以 Chip Name 来分类。

MMAP 存放的路径在: ALKAID/project/board/$(CHIP)/中，例如某些 chip 有 64MB，128MB，256MB 之分，通过文件名就可判断。在 Build config 中会通过设置脚本变量 MMAP = xxx 配置其使用哪一个 MMAP。

MMAP 的文件虽然以.h 为后缀，但是目前它不会编译进代码中，程序编译和运行时会以动态加载的方式导入内存，并通过私有的 mmap parser 函数进行解析，这部分会在后面的章节进行介绍。

Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

2.2. MMAP 在系统打包阶段的处理

上一节介绍过，Linux 内核中的内存配置是看 MMAP 中的 LX_MEM 的位置和大小，原生的 Linux kernel 是在 DTS 中配置内存，SStar 的平台比较特殊，为了一包 kernel image 兼容多个内存配置，它通过 bootargs 把内存大小和地址传递给 kernel，如此就不用改动 image 的内存。

ALKAID 的 project 中兼容了多个平台的打包流程，管理了非常多的 build config，因此 MMAP 也有很多，如果靠人工维护 build config 中的 LX_MEM，配置会非常繁琐且易出错，因此开发出了一个在 build server 上运行的应用程序，通过打包的 shell 执行，解析 MMAP，获取一些字符串作为参数传递给 bootargs 中的 LX_MEM。

代码路径在：

ALKAID/project/image/makefiletools/src/mmappaser/

直接进入到此路径下执行 gcc ./*.c -o mmappaser，会在当前目录下生成一个二进制可执行文件 mmappaser。目前 mmappaser 已经编译好放置在 ALKAID/project/image/makefiletools/bin/，系统在打包的时候 Makefile 会自动找到此路径，并执行默认的 mmappaser 而不自动编译，如果有 mmappaser 的改动需求，请一并 release 编译好的可执行文件到 bin 文件夹下。

“mmappaser” 实现解析 MMAP 的代码与在板子起来后在板子上解析 MMAP 的代码是一样的，实现原理这里不再讨论，其目的只是为了配合编译系统，输出指定字段在 MMAP 中的值。

“mmappaser” 命令执行格式：

./mmappaser [MMAP Path] [CHIP] [cmd para0] [cmd para1]

“mmappaser” 执行后会以打印（printf）的形式输出结果，其结果可以被 shell 脚本获取作为参数值进行传递。打包阶段执行 mmappaser 的过程就是通过配置 MMAP 的路径和 CHIP，自动化解析对应的字符串并作为系统参数值传递给 bootargs 的一个过程。

在板子上解析 MMAP 的文件存放路径在 /config/mmap.ini，此文件就是 ALKAID/project/board/$(CHIP)/中的以.h 为后缀的文件重命名得来。

“mmappaser” 获取某块内存的地址和 size，地址分为物理地址以及 MIU 地址：

./mmappaser [MMAP Path] [CHIP] [MEMORY NAME] phyaddr

./mmappaser [MMAP Path] [CHIP] [MEMORY NAME] miuaddr

./mmappaser [MMAP Path] [CHIP] [MEMORY NAME] size

Makefile/Shell 脚本示例：

文件： ALKAID/project/setup_config.sh

```
PROJ_ROOT=$PWD

if [ "$#" != "1" ]; then
    echo "usage: $0 configs/config.chip"
fi
if [ -e configs ]; then
    echo PROJ_ROOT = $PROJ_ROOT > configs/current.configs
    echo CONFIG_NAME = config_module_list.mk >> configs/current.configs
    echo SOURCE_MK = ../sdk/sdk.mk >> configs/current.configs
    echo "KERNEL_MEMADR = \$(shell $PROJ_ROOT/image/makefiletools/bin/mmapparser $PROJ_ROOT/board/\$(CHIP)/mmap/\$(MMAP) \$(CHIP) E_LX_MEM phyaddr)" >> configs/current.configs
    echo "KERNEL_MEMLEN = \$(shell $PROJ_ROOT/image/makefiletools/bin/mmapparser $PROJ_ROOT/board/\$(CHIP)/mmap/\$(MMAP) \$(CHIP) E_LX_MEM size)" >> configs/current.configs
    echo "KERNEL_MEMADR2 = \$(shell $PROJ_ROOT/image/makefiletools/bin/mmapparser $PROJ_ROOT/board/\$(CHIP)/mmap/\$(MMAP) \$(CHIP) E_LX_MEM2 phyaddr)" >> configs/current.configs
    echo "KERNEL_MEMLEN2 = \$(shell $PROJ_ROOT/image/makefiletools/bin/mmapparser $PROJ_ROOT/board/\$(CHIP)/mmap/\$(MMAP) \$(CHIP) E_LX_MEM2 size)" >> configs/current.configs
    echo "KERNEL_MEMADR3 = \$(shell $PROJ_ROOT/image/makefiletools/bin/mmapparser $PROJ_ROOT/board/\$(CHIP)/mmap/\$(MMAP) \$(CHIP) E_LX_MEM3 phyaddr)" >> configs/current.configs
    echo "KERNEL_MEMLEN3 = \$(shell $PROJ_ROOT/image/makefiletools/bin/mmapparser $PROJ_ROOT/board/\$(CHIP)/mmap/\$(MMAP) \$(CHIP) E_LX_MEM3 size)" >> configs/current.configs
    echo "LOGO_ADDR = \$(shell $PROJ_ROOT/image/makefiletools/bin/mmapparser $PROJ_ROOT/board/\$(CHIP)/mmap/\$(MMAP) \$(CHIP) \$(BOOTLOGO_ADDR) miuaddr)" >> configs/current.configs
    cat $1 >> configs/current.configs
else
    echo "can't found configs directory!"
fi

cat configs/current.configs
make symbol_link
```

编译执行 setup_config.sh 会在 config/current.configs 中自动添加类似字段：

KERNEL_MEMADR = $(shell /home/malloc.peng/ALKAID/project/image/makefiletools/bin/mmapparser

Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

/home/malloc.peng/ALKAID/project/board/$(CHIP)/mmap/$(MMAP) $(CHIP) E_LX_MEM phyaddr)
KERNEL_MEMLEN = $(shell /home/malloc.peng/ALKAID/project/image/makefiletools/bin/mmapparser /home/malloc.peng/ALKAID/project/board/$(CHIP)/mmap/$(MMAP) $(CHIP) E_LX_MEM size)

2.3. Bootarg 内存配置

上一节提到，在系统打包的过程中会自动解析 MMAP 中的 LX_MEM 字段获取地址和大小，并把它作为 bootargs 传递给 linux kernel。

打包完成后在系统的 set_config 中如下显示：

```
LX_MEM=0x7fc6000 mma_heap=mma_heap_name0,miu=0,sz=0x5000000 mma_memblock_remove=1
```

LX_MEM 是 miu0 上的内存大小，默认从头地址开始，不得修改，如果配置 LX_MEM2，则需要添加物理地址起始值。例如有多个 LX_MEM 的配置：

```
LX_MEM=0x7f00000 LX_MEM2=0xa6000000,0x2000000
```

Bootargs 中还有 mma heap 的配置，其设置并不能自动生成，它会根据 build config 中通过手动修改，后面章节会再介绍。

2.4. MMAP 在系统初始化阶段的处理

MMAP 在初始化阶段处理分为两部分：

1、 在 Linux kernel 初始化阶段读取 bootargs 的处理：

Code 路径： kernel/driver/sstar/cpu/memory.c

通过 kernel 标准的流程解析 bootargs 中相关字段。

```
//early_param("mem", MEM_setup);
early_param("LX_MEM", LX_MEM_setup);
early_param("LX_MEM2", LX_MEM2_setup);
early_param("LX_MEM3", LX_MEM3_setup);
```

```
static int __init LX_MEM_setup(char *str)
{

    if( str != NULL )
    {
        sscanf(str,"%lx",&LXmem);
        meminfo.nr_banks = 0;
    lx_mem_size = LXmem;
    }
    else
    {
        printk("\nLX_MEM not set\n");
    }
    return 0;
}
```


Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

函数： void __init prom_meminit(void)，会在系统初始化的时候执行，其中主要的逻辑会先把 dts 中配置的内存移除，再把 bootargs 中的内存添加进系统。

```
  if (linux_memory_length != 0)
  {
f defined(CONFIG_OF)
      /* clear all memblock.memory that DTS declared*/
      static int usermem = 0;
      if (usermem == 0)
      {
          usermem = 1;
          memblock_remove(memblock_start_of_DRAM(),
                  memblock_end_of_DRAM() - memblock_start_of_DRAM());
      }
ndif
      lx_mem_addr = start = (linux_memory_address > PHYS_OFFSET) ? (linux_memory_address) : (PHYS_OFFSET);
      lx_mem_size = size = linux_memory_length + linux_memory_address - start;
      printk("Add mem start 0x%llx size 0x%llx!!!!\n", start, size);
      arm_add_memory(start, size);
  }
```

2、 Linux Kernel 加载完毕后在 MI_SYS 阶段的处理：

kernel 初始化阶段只会使用 MMAP 中 LX_MEM 的部分，在 MI_SYS 中会读取 MMAP 中所有的配置，并存在私有的结构体中。

使用命令/config/dump_mmap 可以 dump 出 SYS 读取的 mmap 配置:

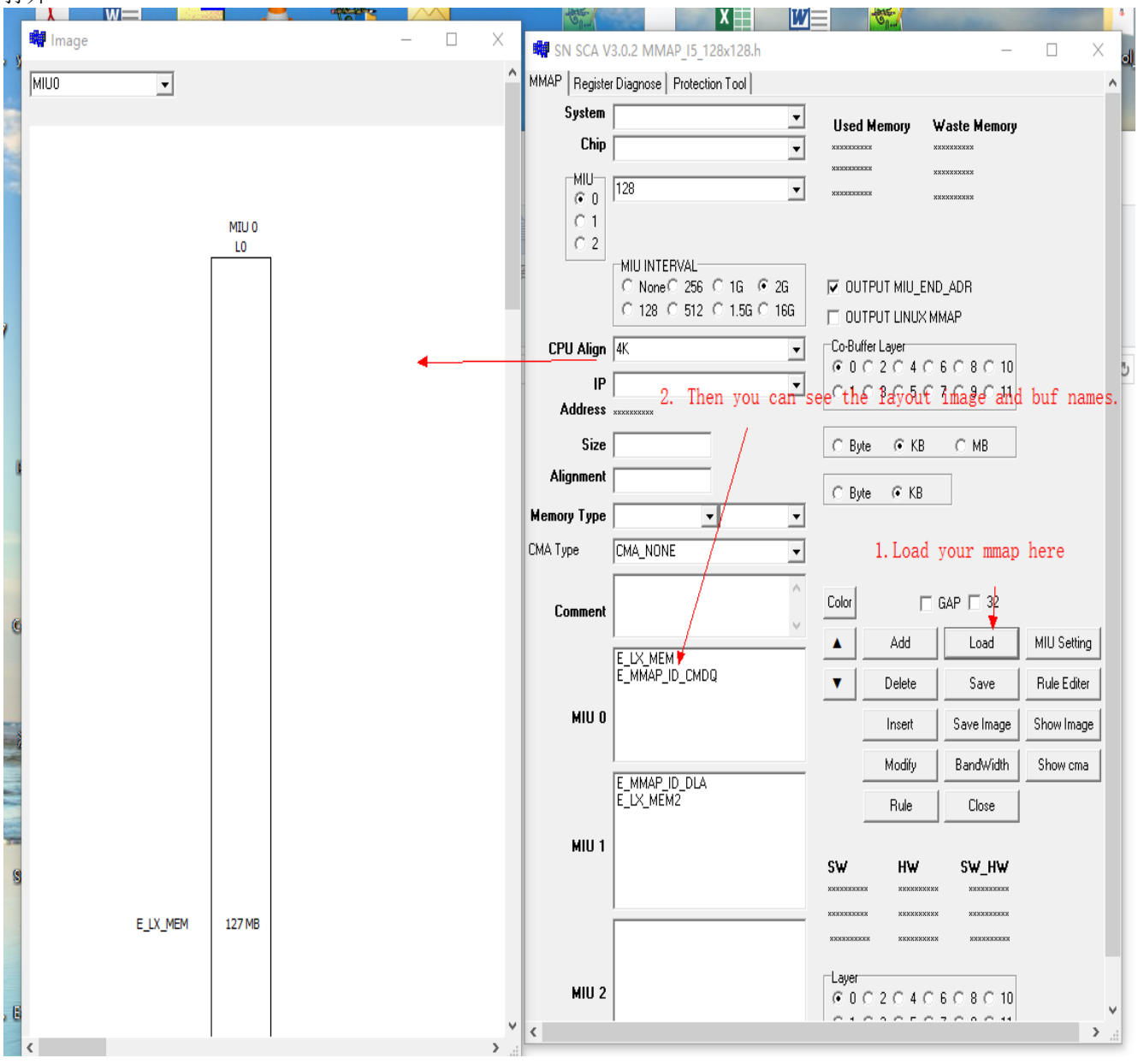
```
/ # /config/dump_mmap
u32TotalSize:0x8in MI_SYSCFG_ShowMmap:77
in commonraw_proc_read:314
000000
u32Miu0Sin commonraw_proc_read:323
ize:0x8000000
u32Miu1Size:0x0
u32MiuBoundary:0x80000000
u32MmapItemsNum:3
bIs4kAlign:1
bMiu1Enable:0
E_LX_MEM:GID=0,Addr=0x0,Size=0x7fc6000,Layer=0,Align=0x1000,MemoryType=0x4,MiuNo=0,CMAID=0
E_MMAP_ID_CMDQ:GID=1,Addr=0x7fc6000,Size=0x1a000,Layer=0,Align=0x1000,MemoryType=0x4,MiuNo=0,CMAID=0
E_MMAP_ID_EMI:GID=2,Addr=0x7fe0000,Size=0x20000,Layer=0,Align=0x1000,MemoryType=0x4,MiuNo=0,CMAID=0
```


Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

2.5. SCA 工具修改 MMAP

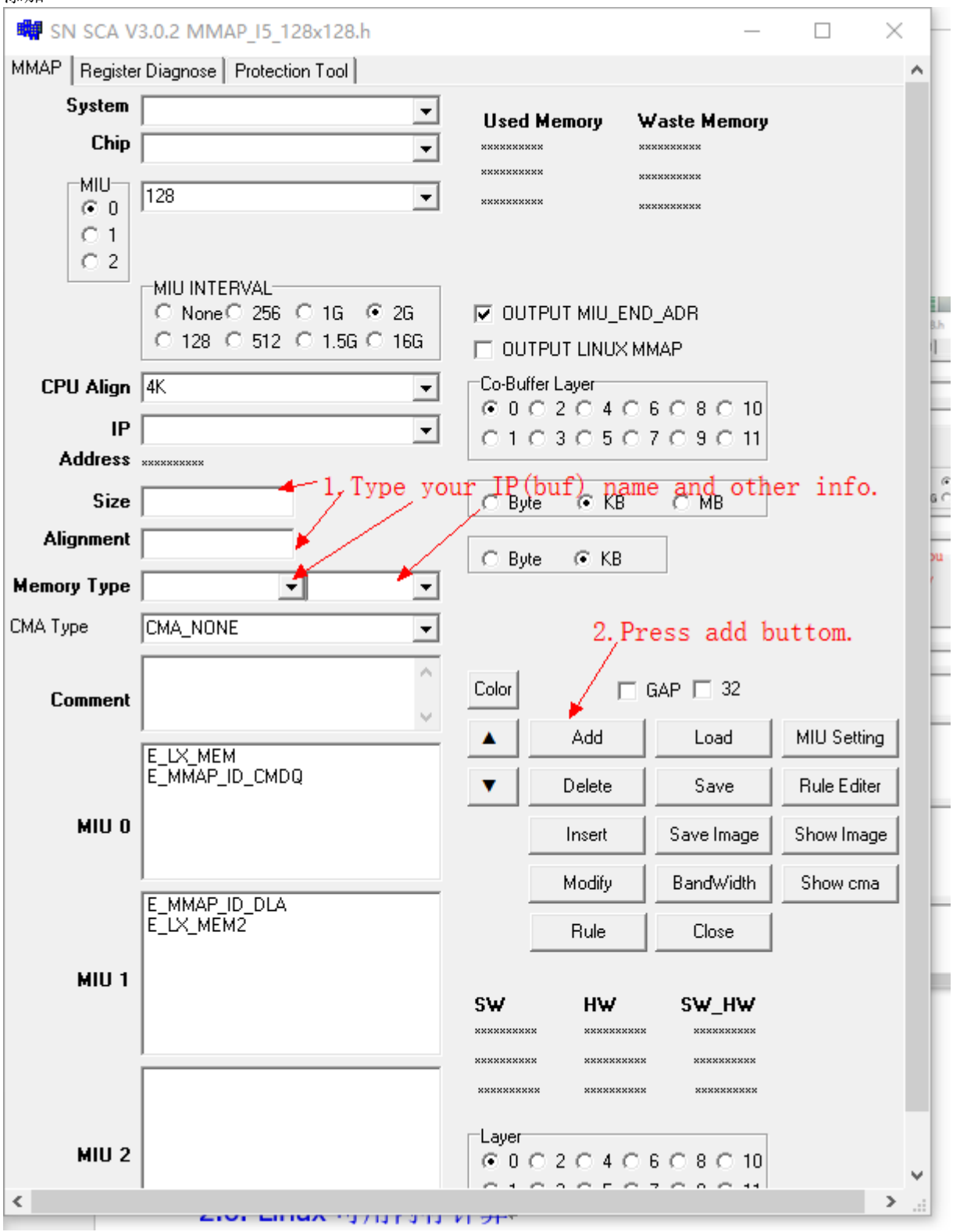
MMAP 中的地址偏移量和 size 的修改比较繁琐，不建议手工修改 MMAP 文件，推荐使用 SCA 工具修改。

1） 打开



Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

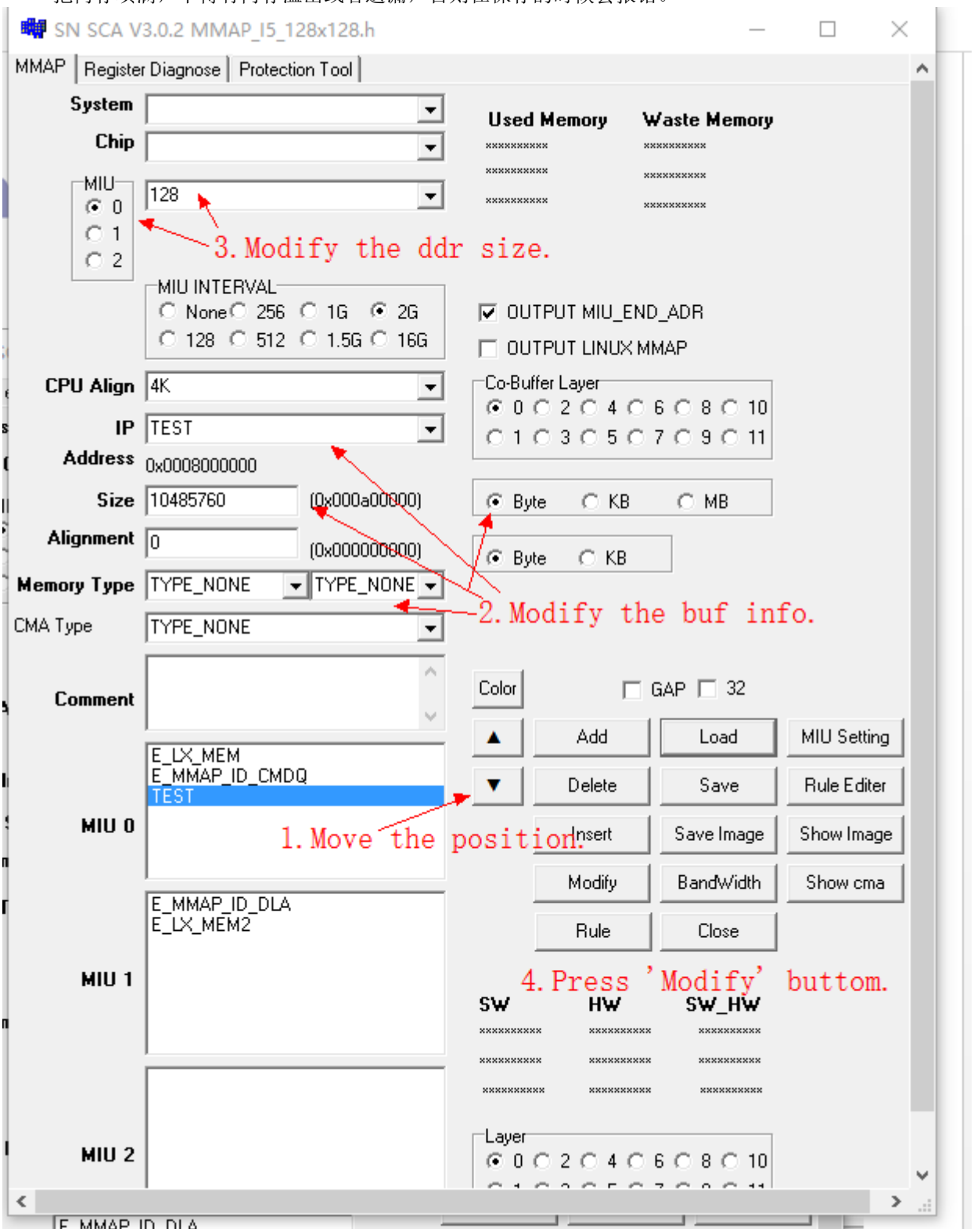
2） 添加



Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

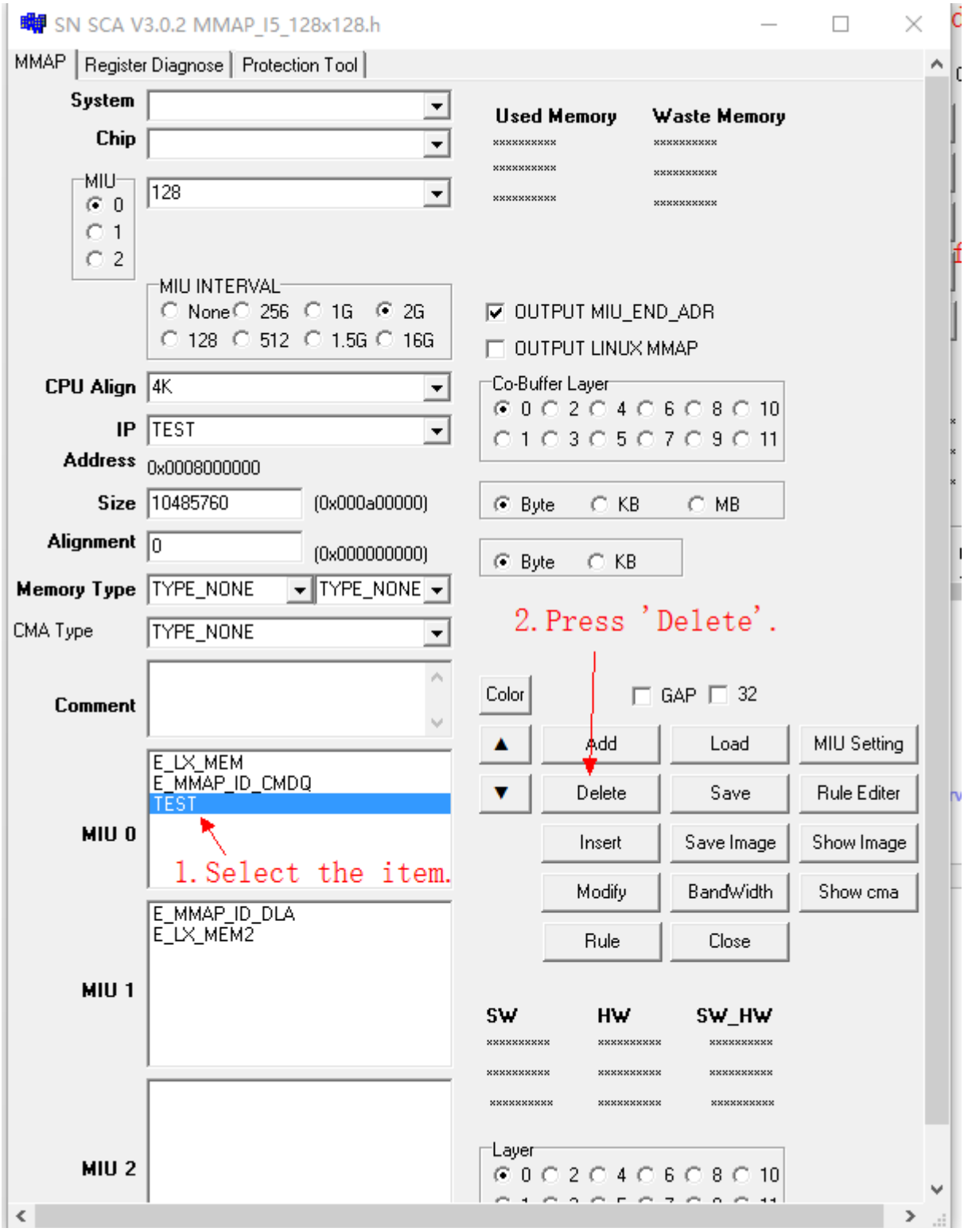
3） 修改

内存修改包括 buf 的大小改动，位置改动，以及调整整个 DDR 的大小，任何内存大小改动后，都应当把内存填满，不得有内存溢出或者遗漏，否则在保存的时候会报错。



Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

4） 删除



Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

5） 保存
按'Save'按钮进行保存。

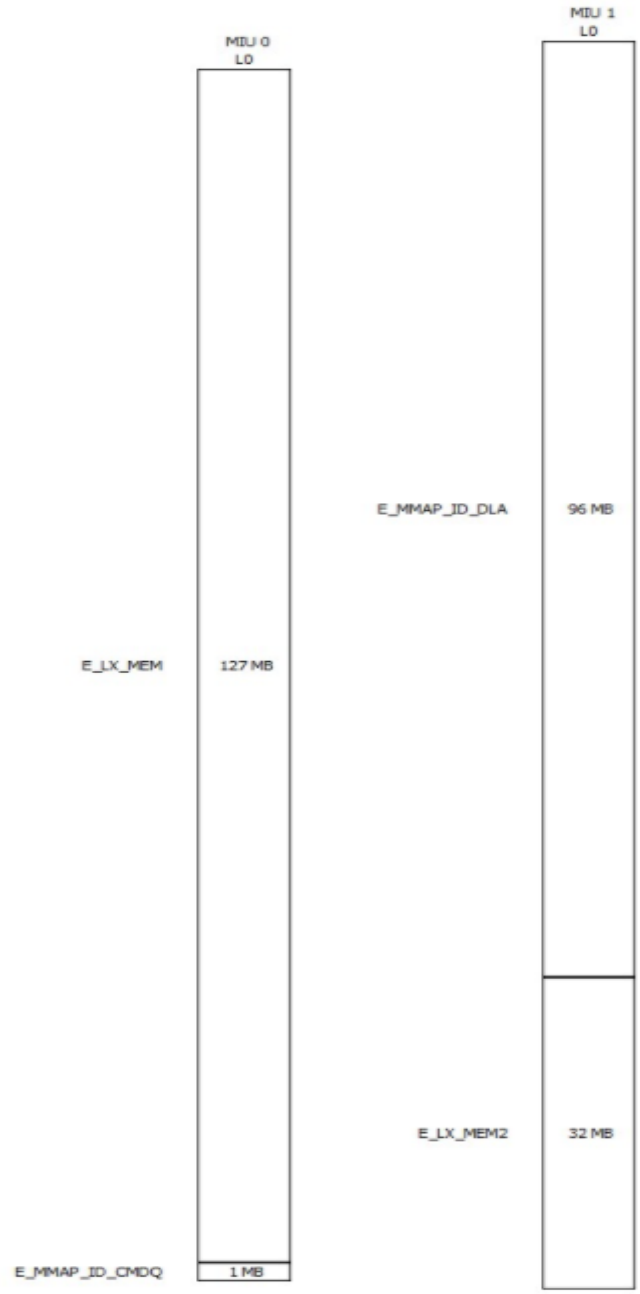
2.6. Linux 可用内存计算

通过命令 cat /proc/meminfo 可以得出当前 kernel 中可以使用的内存大小，此内存大小由以下公式得出。

Meminfo 内存大小 = （ Bootargs 中配置所有 LX_MEM 大小之和） - （所有 MMA 的 size + Kernel reserve 5M）

2.7. 一些特殊 layout 举例

1） 128MB + 128MB 带 DLA MMAP



2）

Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

从上图可知，有两个 DDR，所以 LX_MEM 会分配两块，在 MIU1 上有 layout 一块 DLA 的内存，DLA 的输入内存是从 DIVP 的 output 端口出来的，而 DLA 对内存的地址要求必须在它自己 FW 地址的后面，因此可以看到在 E_MMAP_ID_DLA 之后有 layout 了一块 LX_MEM2。

DIVP 使用的输出内存必须配置到 E_LX_MEM2 上 DLA 才能读到，所以 mma heap 配置上会单独配置一块 mma_heap_name1 给 divp 使用，使其输出的内存地址在 miu1 的 E_LX_MEM2 中，参考 bootargs 配置：

LX_MEM=0x7f00000 LX_MEM2=0xa6000000,0x2000000
mma_heap=mma_heap_name0,miu=0,sz=0x5000000 mma_heap=mma_heap_name1,miu=1,sz=0x500000
mma_heap=mma_heap_name2,miu=1,sz=0x1800000

Copyright © 2019 SigmaStar Technology Corp. All rights reserved.

3. MMA HEAP

3.1. 配置以及 Dump

通过修改 bootargs 中的 mma_heap 字段可以动态地修改 mmap heap 的大小。在每个 project 的 build config 中都会针对应用场景配置一个默认的 mma heap size，这个大小只是预估值，FAE 可以在现场针对客户的应用场景精确计算大小。

Mmap_heap 配置的几个 flag：

1、 mma_memblock_remove=1
此 flag 代表是否把 kernel reserve 出来给 mma 的内存从 kernel 内存部分移除。

2、 max_start_off=$(MMA_START_OFFSET)
Mma 从 kernel reserve 出来的内存尾地址不得超过$(MMA_START_OFFSET)，否则申请失败。

Dump mma:

cat /proc/mi_modules/mi_sys_mma/mma_heap_name0
cat /proc/mi_modules/mi_sys_mma/mma_heap_name1

3.2. 物理地址以及 MIU 地址

物理地址是 HW design 上，"CPU"存取 DRAM 的 physical 地址
譬如双通道 ddr 的 CPU 存取 MIU0 的起始物理地址为 0x20000000，MIU1 的起始物理地址为 0xA0000000。
MIU 地址是 HW IP 通过 MIU 访问 DRAM 的地址，MIU0 的地址从 0x0 开始，MIU1 从 0x80000000 开始。

Copyright © 2019 SigmaStar Technology Corp. All rights reserved.