GM8136

FRAMMAP User Guide

User Guide

Rev.: 1.0

Issue Date: September 2014

REVISION HISTORY

GM8136 FRAMMAP User Guide

Date	Rev.	From	To
Sept. 2014	1.0	-	Original

TABLE OF CONTENTS

List of Figures

Chapter 1

Introduction

This chapter contains the following sections:

The purpose of FRAMMAP is to provide a mechanism of the memory allocation management. Before, the module used either alloc_pages() or kmalloc() function to allocate the memory from kernel and two problems may be encountered. The first one is the well-known memory fragmentation issue when the memory is allocated and freed quite frequently. If the drivers want to allocate big and physical contiguous memory, it may fail due to this reason. The second one is hard to allocate big enough memory because of the upper bound size limitation in kernel. In addition to the above two reasons, the DDR bandwidth consideration is also very important. Users might want the driver module to allocate memory from the designated DDR if the driver module always occupies the memory bus for a long time. Assume that two IPs are wired in the same DDR channel; they may influence each other while accessing the DDR bus for a long time and causes the system to work abnormally because of sharing the same DDR bandwidth. Therefore, it is necessary to arrange the DDR bandwidth for them to reduce the impact to each other. For this reason, FRAMMAP provides a configuration file for each driver module to designate the DDR number while allocating memory.

1.1 Functional Block Diagram

In Figure 1-1, a top-level diagram is provided to show the functional organization of FRAMMAP. In Figure 1-1, DDR0 means the memory reserved by the kernel for FRAMMAP. Similarly, DDR1 means the memory reserved by the kernel for FRAMMAP during kernel's initialization. The block diagram consists of three parts. The first part is applications that can allocate the memory through FRAMMAP. The applications can be either the kernel drivers or user-space applications. The second part is the interface to applications. This block exposes the APIs or function calls to the applications when users want to allocate the memory through FRAMMAP. The third part is the Management block of DDRx. The Management block provides the mechanism to allocate/free memory from DDR0 or DDR1, in which the DDR number depends on how many DDRs in the system. In addition to providing the physical continuous memory, the block also provides the virtual memory to map the allocated physical address. The virtual memory can be cacheable or write through depending on the requirements of the applications. The methodology of the memory management is identical to the kernel methodology of managing the virtual memory. Please be aware that the warning message will pop on when the applications want to allocate memory from DDR0, which was already exhausted. It means that the DDR0 size reserved for FRAMMAP is not arranged well in advance. In this case, FRAMMAP will try to allocate memory from NORMAL zone of the kernel memory. But for another DDR except DDR0, it just returns NULL.

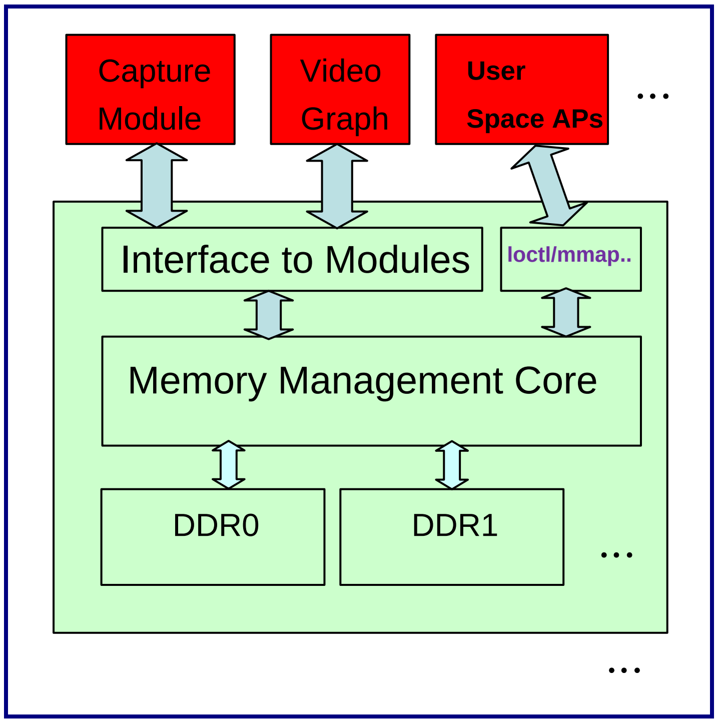


Figure 1-1. Top-level Diagram of FRAMMAP

1.2 Functions

FRAMMAP has the following features:

- Big and physical continuous memory regions are reserved during memory initialization of Linux kernel for DDR0 and DDR1.
- Provides the global physical address alignment for DDR. This value can be specified by the module parameter.
- Provides the physical address alignment for each allocated memory. If the alignment is less than the one of global, the larger alignment value is taken. The maximum alignment is 8M and needs to be 2^x. Please note that the alignment is only for the physical address instead of the virtual memory.
- FRAMMAP allows the kernel space or user-space applications to allocate memory.
- For the user-space applications, FRAMMAP only provides a memory pool and the applications should manage the memory by itself instead of frequently allocating/releasing memory. That is, the memory should only be released when the applications are terminated.
- While the memory is released, it will immediately merge with the neighbor memory region.
- The unused memory can be returned to kernel any time.

- Allocating/Releasing memory quiet frequently may cause the memory fragmentation.
- An editable configuration file is provided to describe the relationship between the driver module and DDR number. The following is an example of the configuration file.

  ```
  cat /mnt/mtd/config_8136
          H264D_Reference            SYSTEM
          FFB0(SD)_Master            DDR-1
          FFB1(HD)_Master            DDR-2
          …
  ```

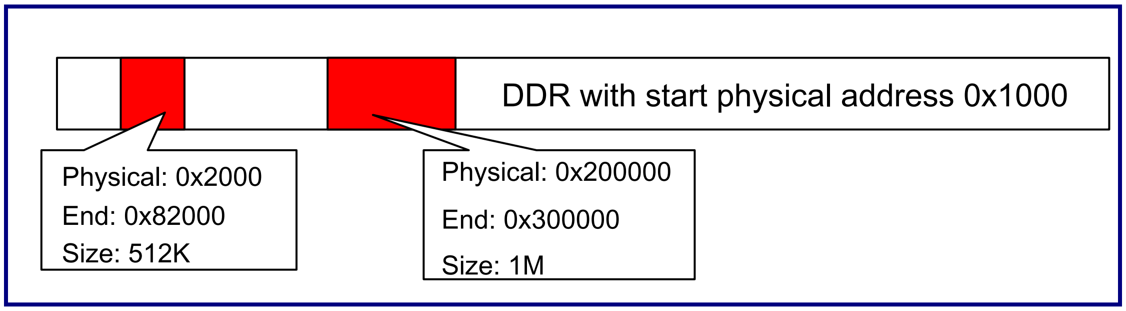
  The example shows that FRAMMAP will allocate "system" memory for the driver module, "H264D_Reference", and allocate DDR-1 (Second DDR) memory for the driver module, "FFB0(SD)_Master", according to this configuration file. Please note that when the configuration file is edited and changed, in order to make the change take effect, all related driver modules need to be unloaded and loaded again.
- The proc interfaces are provided to monitor the DDR usage, memory usage/ distribution and statistic.
- Memory leakage check. When the FRAMMAP module is unloaded, it will check if any memory is allocated but is not freed. If it is true, the warning message will pop on to warn the users.

Chapter 2

Function Theory

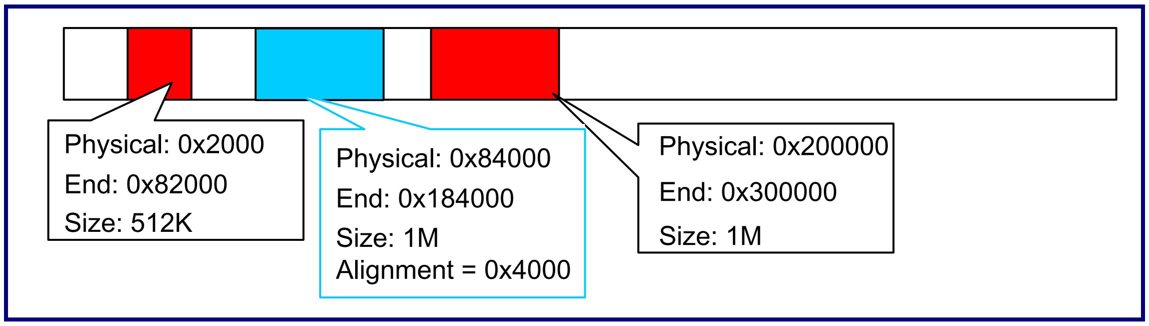
This chapter describes how FRAMMAP works inside the memory management core. For each DDR management, Grain Media maintains a list and statistic to record the memory usage. In the former, the list is used to record the allocated memory information. When a new memory is allocated, it will create a memory block for this memory and keep its start physical address, requested alignment, allocated size, virtual memory address, and so on. In the beginning, the list is empty. When a new memory block is created, it will be linked into this list. That is, the list keeps all allocated memories. The later, statistic is used to record how many memory blocks are already created, how many pages are used, what is the maximum available size for applications to allocate and so on.

The following example shows the simple theory. Assuming that the start physical address of the DDR is from 0x1000 and two memory regions are already allocated. The first allocated physical memory region is from 0x2000 with size = 512K bytes. The second allocated physical memory region is from 0x200000 with size = 1M. Now, the application would like to allocate a memory with size = 1M and alignment = 0x4000, the algorithm is shown one by one.



1) Assign init variable, last_phys_end = 0x1000
2) Find the next memory region information; obtain its physical start address, say phys_addr. available_space = phys_addr – last_phys_end.
3) If available_space is less than the requested size, update last_phys_end to the end address of the current memory region and goes to Step 2. Otherwise, the candidate physical address will be the end address of the previous memory region and goes to Step 4.
4) cand_phys_addr = ALIGN (Candidate physical address, requested alignment). If (cand_phys_addr + requested_size) is less than phys_addr of current memory region, the memory is found. Otherwise, only update last_phys_end to the end address of the current memory region and goes to Step 2 until the proper memory region is found or goes through whole DDR.

In above example, a new memory region marked in blue color is inserted.



If the blue memory region is released by the application, users only need to remove it from the list. The available fragments space will be automatically merged.

Chapter 3

Configuration File

This chapter contains the following section:

As mentioned before, a configuration file is used to choose the DDR for memory allocation of the applications. That is, when the applications want to allocate memory, this configuration file will be looked-up. When the FRAMMAP module is inserted, it will find the configuration file in the path of the /mnt/mtd/ directory and load it once. If the configuration file does not exist, a temp under the configuration file in /tmp/ directory is created with the default content as shown below. Users can manually copy it to /mnt/mtd directory. The default content is defined in map.c. Please be aware that when the configuration file is edited and changed, the FRAMMAP module needs to be unloaded and loaded again to make the change take effect.

```
/lib/modules #insmod frammap.ko Frammap: fail to open /mnt/mtd/config_8136, /tmp/template_8136 is created.

Frammap: DDR0: memory base=0x4a000000, memory size=0x360000, alignment=4K
```

3.1 Fields of Configuration File

Two columns are in the configuration file. The first one is the name of the driver module. The second one is the DDR number while allocating memory as shown below.

```
/tmp # cat template_8136
** SYSTEM: system memory, DDR-1: 2nd memory, DDR-2: 3rd memory, … **
** Note: The name needs to be continuous without space. **
H264E_BitStream              SYSTEM
H264E_G1(DMA_CHAIN)          SYSTEM
MMAP                         SYSTEM
MMAP                         SYSTEM
H264D_Reference              SYSTEM
LCD_FB0                      SYSTEM
LCD_FB1                      SYSTEM
LCD_FB2                      SYSTEM
```

The name in each row is pre-defined and not changeable. Taking LCD200 as an example, there are three planes in it, so when the first plane needs a memory, LCD_FB0 row will be matched and allocate memory from SYSTEM. Similarly, for planes 2 and 3, when they need memories, the “LCD_FB1” and “LCD_FB2” rows are matched individually to dispatch the memory for them. In addition to these pre-defined rows, Grain Media provides extra rows for free usage. They are “User0” ~ “User15”. Please refer to map.c. Besides, Grain Media also provides the function calls to directly allocate the memory from the designated DDR without looking up the configuration file. The function is the pair of: frm_get_buf_ddr() and frm_free_buf_ddr().

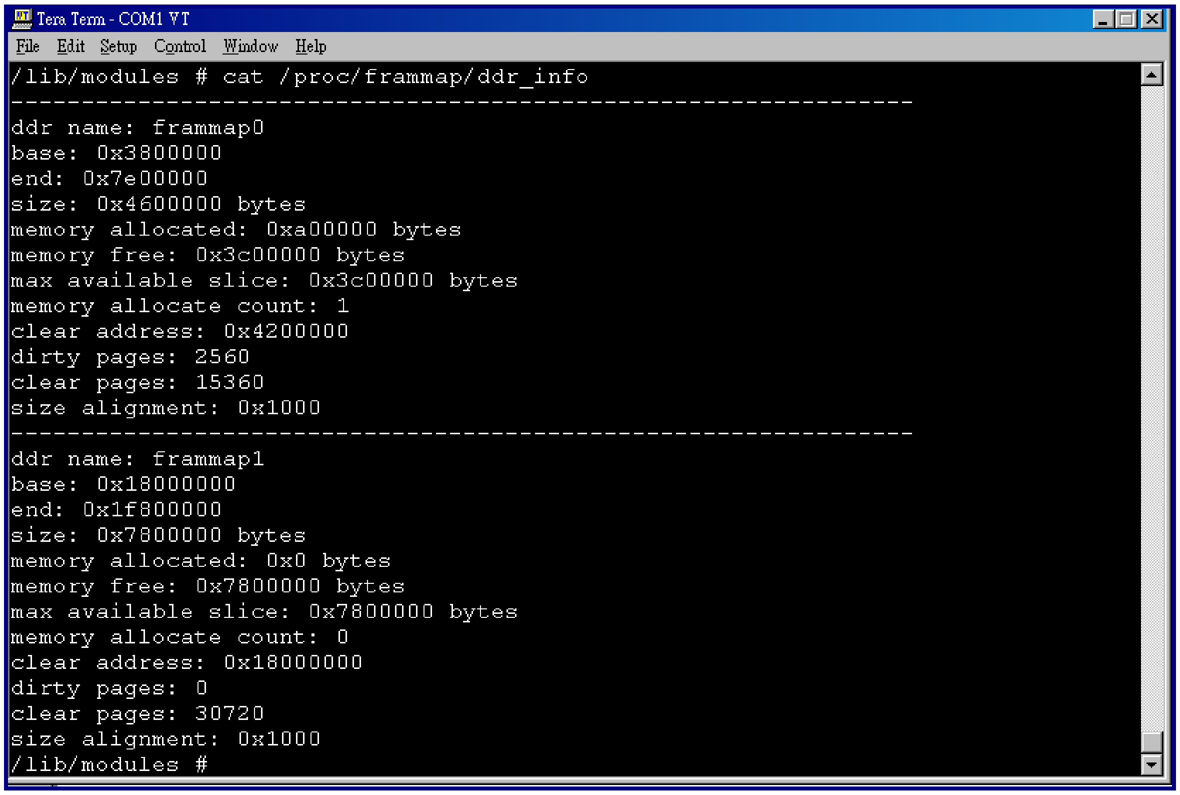
Chapter 4

Proc Nodes

FRAMMAP creates some proc interfaces for information monitor, and users can read some information about FRAMMAP through these proc interfaces. They are:

- ddr_info: This interface lists the information about all DDRs managed by FRAMMAP.
    - ddr name: The DDR name
    - base: The physical base address of this management DDR
    - end: The end physical address of this management DDR
    - size: The total size of this management DDR
    - memory allocated: The number of allocated memory in bytes
    - memory free: The number of available memory in bytes
    - max available slice: The maximum available size to allocate
    - clear address: The first address which is never allocated
    - dirty pages: How many pages ever been allocated
    - clean pages: How many pages never been allocated. It is similar to clear address.
    - size alignment: The global alignment of this DDR

From clean pages, users can know that how many pages are never used. If there are many clean pages, it means that the size for this management DDR is too large.



```
/lib/modules # cat /proc/frammap/ddr_info
----------------------------------------------------------------
ddr name: frammap0
base: 0x3800000
end: 0x7e00000
size: 0x4600000 bytes
memory allocated: 0xa00000 bytes
memory free: 0x3c00000 bytes
max available slice: 0x3c00000 bytes
memory allocate count: 1
clear address: 0x4200000
dirty pages: 2560
clear pages: 15360
size alignment: 0x1000
----------------------------------------------------------------
ddr name: frammap1
base: 0x18000000
end: 0x1f800000
size: 0x7800000 bytes
memory allocated: 0x0 bytes
memory free: 0x7800000 bytes
max available slice: 0x7800000 bytes
memory allocate count: 0
clear address: 0x18000000
dirty pages: 0
clear pages: 30720
size alignment: 0x1000
/lib/modules #
```

- mmapx: x can be 0, 1, 2 … This interface lists all memory regions information in DDR. This interface is very useful for reading the memory distribution. This interface is placed in /proc/frammap/mmapx.

```
/lib/modules # cat /proc/frammap/mmap0

Memory Callers Distribution in DDR0
--------------------------------------------------------------------------
name: cpu_comm, pa: 0x3800000, va: 0xd0000000, size: 0xa00000, flag: 0x20000020
/lib/modules #
```

  - name: Application name
  - pa: Physical address
  - va: Its mapped virtual address

    - size: Allocated memory size
    - flag: It is for internal use only. The purpose is to see the memory attribute. Bit 31-28 indicates allocated type. Bit 23-0: allocated address alignment.
- config_file: This interface lists the relationship between the driver applications and DDR number. When the FRAMMAP module is inserted, it will read the configuration file from /mnt/mtd/config_8136 if the platform is GM8136 and produces this interface node. If this configuration file does not exist, FRAMMAP will auto-produce this interface node with the default content. This interface node lists the current mapping relationship for all driver applications. If the configuration file is edited and changed, the content of this interface node is not changed until FRAMMAP is unloaded and loaded again. This interface node is placed in /proc/frammap/config_file.
- mmap_kernel: This interface lists which applications use the kernel memory. That is, when the drivers allocate the DDR0 memory from FRAMMAP which cannot provide the enough memory, at this moment, the memory from kernel is returned. It is unexpected while arranging the FRAMMAP memory size.
- free_pages: It is used to return the useless memory to the kernel. Assume that the memory size has 55M bytes and only 30M bytes are used, the users can have FRAMMAP return the left memory to the kernel by setting this proc interface. The following is the usage.
  echo x > /proc/frammap/free_pages, where x indicates DDR ID which starts from 0.

Chapter 5

Module Parameters

FRAMMAP supports the module parameters. The following parameters are currently supported.

- sys_align: It means the alignment for the DDR0 memory; the alignment size is 4K * (2 ^ sys_align) bytes. Give the following example: insmod frammap.ko sys_align = 7, the allocated system memory will be alignment with 512K bytes which derived from 4k*128.
  - If sys_align is zero or not specified, it means PAGE alignment.
- ddrx_align: It means the alignment for DDR-x (where x indicates 1, 2,…) memory, the alignment calculation is the same as sys_align.
  - If ddrx_align is zero or not specified, it means PAGE alignment.
- ddr0: What is DDR0 size will be managed by FRAMMAP. Please note that the size cannot exceed the size defined in memory.h for the DDR0 memory size. For example: insmod frammap.ko ddr0 = 50M.
- ddrx: It refers to which memory will be managed by FRAMMAP for DDR-x (where x indicates 1, 2,…). If the DDR1 size is 128M and only 50M is managed by FRAMMAP, then the left memory will be returned to the kernel. For example: insmod frammap.ko ddr1 = 50.

Chapter 6

AP Memory Allocation

In addition to provide the mechanism allocating the designated DDR for the kernel applications, FRAMMAP also provides the interface to the user-space applications. Please note that it is not recommended for the user applications using it as malloc() library which means to allocate and free memory frequently. FRAMMAP only provides a channel to allocate large memory pool and the user-space applications should manage this memory by itself. This memory pool can only be released when this application is terminated. This is kernel behavior instead of FRAMMAP limitation.
In order to provide this function, an IOCTL and the defined structure are provided.

```
struct frmmap_meminfo_s
{
    unsigned int    aval_bytes;
    void                *drv_data;
};
```

- aval_bytes: How many bytes are available to allocate.
- drv_data: The pointer is internal used by the driver only. Do not either touch or change this pointer.

When the user applications call mmap(), FRAMMAP will allocate the requested memory correspondingly.

IOCTL:

- FRM_IOGBUFINFO: It is read only. It provides the information of the available bytes the AP can allocate once.

```
#include "frammap_if.h"
int main()
{
    int           fbfd = 0;
    unsigned char            *mmap_addr0;
    frmmap_meminfo_t         meminfo;

    fbfd = open("/dev/frammap0", O_RDWR);
    if (fbfd == 0)  FAIL_PROCESS;
    if (ioctl(fbfd, FRM_IOGBUFINFO, &meminfo) < 0)  FAIL_PROCESS;
    mmap_addr0 = (unsigned char *)mmap(0, meminfo.aval_bytes, PROT_READ | PROT_WRITE,
            MAP_SHARED, fbfd, 0);
    if (mmap_addr0 == (unsigned char *)(-1)) FAIL_PROCESS;

    ….. body ….

    close(fbfd); /* release all allocated memory when application is terminated later */
    return 0;
}
```

Chapter 7

FRAMMAP Configuration

This chapter contains the following sections:

7.1 FRAMMAP Memory Size for DDR0

The memory size managed by FRAMMAP for the system DDR is configurable. If users want to change the size, please modify the following file:

```
#define DDR0_FMEM_SIZE  0x3600000   /* memory size in DDR0 managed by FRAMMAP */

Its location is in: arch/arm/mach-GM /include/mach/platform/memory.h
```

It means that the system will pre-reserve DDR0_FMEM_SIZE bytes memory during the system initialization. But the users can decide the real memory size while inserting FRAMMAP module. For example:

```
insmod frammap.ko ddr0=30.
```

The system pre-reserved 54M bytes, but only 30M bytes are needed. Therefore, FRAMMAP will return (54M-30M) bytes memory to the system. Finally, only 30M is managed by FRAMMAP.

7.2 FRAMMAP Memory Size for DDR1

The memory size managed by FRAMMAP for DDR1 is configurable. If users want to change the size, please modify the following file:

```
#define DDR1_FMEM_SIZE  0x3600000   /* memory size in DDR1 managed by FRAMMAP */
or
#define DDR1_FMEM_SIZE      -1
Its location is in: arch/arm/mach-GM/include/mach/platform/memory.h
```

In the latter case with DDR1_FMEM_SIZE = -1, it means that FRAMMAP will do best effort to pre-reserve the whole memory size from the DDR. Assume that DDR1 has 128M, FRAMMAP will try to pre-reserve 128M. Identical to DDR0, the left memory can be returned to kernel if users do not need them.