1. Overview

1.1 Overview of this Software

・To bring out the performance of the multimedia H/W IP, the following features for the memory space are required.
(1) Not only the space reserved as default CMA area, which is shared by H/W IP and CPU, but also the space reserved as CMA area for MMP, which is used by H/W IP (Video Codec IP) only.
(2) The space reserved as CMA area for Lossy compression feature, which is used by HW IP (FDP, VSPI and VSPB).
(3) The larger contiguous space.
(4) The space reserved as some CMA areas can be located in 32-bit physical address space (legacy area) or 40-bit physical address space (with IPMMU MMU support).

・Therefore Memory Manager (hereafter called in MMNGR) provides the following memory space.
(1) The space reserved as default CMA area, which is shared by H/W IP and CPU.
(2) The space reserved as CMA area for MMP, which is used by H/W IP (Video Codec IP) only.
(3) The space reserved as CMA area for Lossy compression feature, which is used by H/W IP (FDP, VSPI and VSPB).
(4) The physical contiguous space over 4MB as reserved CMA area by one allocation request.
(5) The physical contiguous space (e.g. the space reserved as CMA area for MMP) can be assigned into 32-bit address space or 40-bit address space.

・MMNGR does not support to provide above memory space directly to Linux Kernel Display Driver via DMA buffer sharing.

・The APIs are provided in the user layer.

1.2 Configuration of this Software

This software consists of the following resources.

・Document
・Source code
・Sample application
・Makefile

To use this software, the following additional software, which is not included in this software, is required.
Details of this additional software are shown below.

・Kernel module source code

This software is distributed in Dual MIT/GPLv2 licenses.
Figure 1-2 shows the lists of these source files.

```
mmngr_lib
├─── COPYING.MIT
├─── libmmngr
│   ├─── mmngr
│   │   ├─── AUTHORS
│   │   ├─── ChangeLog
│   │   ├─── configure.ac
│   │   ├─── if
│   │   │   ├─── Makefile.am
│   │   │   └─── mmngr_if.c
│   │   ├─── include
│   │   │   ├─── mmngr_user_private.h
│   │   │   └─── mmngr_user_public.h
│   │   ├─── Makefile.am
│   │   ├─── NEWS
│   │   ├─── README
│   │   └─── tp
│   │       ├─── main.c
│   │       └─── Makefile.am
│   └─── mmngrbuf
│       ├─── AUTHORS
│       ├─── ChangeLog
│       ├─── configure.ac
│       ├─── if
│       │   ├─── Makefile.am
│       │   └─── mmngr_buf_if.c
│       ├─── include
│       │   ├─── mmngr_buf_user_private.h
│       │   └─── mmngr_buf_user_public.h
│       ├─── Makefile.am
│       ├─── NEWS
│       ├─── README
│       └─── tp
│           ├─── main.c
│           └─── Makefile.am
└─── README
```

Figure 1-1 Configuration of this software

```
mmngr_drv
├─── GPL-COPYING
├─── MIT-COPYING
├─── mmngr_drv
│   ├─── mmngr
│   │   ├─── ChangeHistory.txt
│   │   └─── mmngr-module
│   │       └─── files
│   │           └─── mmngr
│   │               ├─── drv
│   │               │   ├─── Makefile
│   │               │   └─── mmngr_drv.c
│   │               └─── include
│   │                   ├─── mmngr_private.h
│   │                   ├─── mmngr_public.h
│   │                   ├─── mmngr_private_cmn.h
│   │                   └─── mmngr_public_cmn.h
│   └─── mmngrbuf
│       ├─── ChangeHistory.txt
│       └─── mmngrbuf-module
│           └─── files
│               └─── mmngrbuf
│                   ├─── drv
│                   │   ├─── Makefile
│                   │   └─── mmngr_buf_drv.c
│                   └─── include
│                       └─── mmngr_buf_private.h
│                       └─── mmngr_buf_private_cmn.h
└─── README
```

Figure 1-2 Kernel module source code not included in this software

1.3 Development Environments

This section describes the development environments for this software.

1.3.1 Hardware Development Environment

Table 1-1 Hardware specification

Hardware Name		Remarks
Device	R-Car H3 / M3 / M3N / E3	-
Board	R-CarH3-SiP/M3-SiP System Evaluation Board Salvator-X	-
Board	R-CarH3-SiP/M3-SiP/M3N-SiP System Evaluation Board Salvator-XS	-
Board	R-CarE3 System Evaluation Board Ebisu	-
Board	R-CarE3 System Evaluation Board Ebisu-4D	

1.3.2 Software Development Environment

Table 1-2 Software specification

Software Name	Version / Revision	Remarks
R-Car H3/M3/M3N/E3 Linux BSP	-	-

2. Installation Procedures

2.1 Building the shared library

The following item is the procedure for building the shared library of this software.

1) Setting environment variables

The shared library must be 64-bit version. Follow Yocto Recipe Start-Up Guide to set up the 64-bit SDK.

```
$ source <Yocto SDK script>
```

In addition to the guide, set the following value

```
$ export INCSHARED=$SDKTARGETSYSROOT/usr/local/include
```

2) Building the shared library

Execute make under libmmngr/mmngr/.

```
$ cd libmmngr/mmngr
$ autoreconf -i
$ ./configure ${CONFIGURE_FLAGS} --prefix=$PWD/tmp
$ make
$ make install includedir=$INCSHARED
```

Note that the CONFIGURE_FLAGS variable was already set by SDK. Do not overwrite it.

3) Verifying the shared library

Make sure that the following shared libraries are built under libmmngr/mmngr/tmp/lib.

libmmngr.so.x.x.x (“x.x.x” depends on the version.)

In addition, if you use the dmabuf function of MMNGR, follow 4) and 5).

4) Building the shared library

Execute make under libmmngr/mmngrbuf.

```
$ cd libmmngr/mmngrbuf/
$ autoreconf -i
$ ./configure ${CONFIGURE_FLAGS} --prefix=$PWD/tmp
$ make
$ make install includedir=$INCSHARED
```

Note that the CONFIGURE_FLAGS variable was already set by SDK. Do not overwrite it.

5) Verifying the shared library

Make sure that the following shared libraries are built under libmmngr/mmngrbuf/tmp/lib.

libmmngrbuf.so.x.x.x (“x.x.x” depends on the version.)

2.2 Building the Kernel image and the Kernel module

The following is the procedure for building the kernel image and the kernel module that is not included in this software.

The procedure is described for default memory map configuration that all CMA areas are located in 32-bit legacy area.
For customized memory map, which moves CMA area for MMP to 40-bit address space, refer to 10.3 for more details.

1) Setting environment variables

The kernel module is always 64-bit. Please use 64-bit SDK when compiling kernel module.
To set up SDK, follow Yocto Recipe Start-Up Guide.

```
$ source <Yocto SDK script>
```

In addition to the guide, set the following value.

```
$ export KERNELSRC=(the kernel path in your environment) (for example, $WORK/renesas-bsp)
$ export CP=cp
$ export MMNGR_CONFIG=MMNGR_SALVATORX (*)
$ export MMNGR_SSP_CONFIG=MMNGR_SSP_DISABLE
$ export MMNGR_IPMMU_MMU_CONFIG=IPMMU_MMU_DISABLE
$ export MMNGR_VALIDATE_CONFIG=”MMNGR_ADDRESS_VALIDATION”
$ export INCSHARED=$SDKTARGETSYSROOT/usr/local/include
```

*: This definition depends on your environment
In case target SoCs are H3, M3, M3N, please set as following value:

```
$ export MMNGR_CONFIG=MMNGR_SALVATORX
```

In case target SoC is E3, please set as following value:

```
$ export MMNGR_CONFIG=MMNGR_EBISU
```

2) Building the Kernel image

MMNGR uses CMAs (Contiguous Memory Allocator) of the kernel image.
If you change the size of the spaces, perform the following steps.
If you don’t need to change them, follow Step.3 only.

Step.1
Modify the start address and size of default CMA area (Refer to Yocto Recipe Start-Up Guide).

Step.2
Modify the start address and size of CMA area for MMP under node linux,multimedia in device tree arch/arm64/boot/dts/renesas/xxxx.dts. (H3 Ver.1.x: xxxx.dts means r8a77950-salvator-x.dts,
H3 Ver.2.0: xxxx.dts means r8a77951-salvator-x.dts, r8a77951-salvator-xs.dts,
H3 Ver.3.0: xxxx.dts means r8a77951-salvator-xs-4x2g.dts,
M3 Ver.1.x: xxxx.dts means r8a77960-salvator-x.dts, r8a77960-salvator-xs.dts,
M3 Ver.3.0: xxxx.dts means r8a77961-salvator-xs-2x4g.dts,
M3N : xxxx.dts means r8a77965-salvator-x.dts, r8a77965-salvator-xs.dts,
E3 Ver.1.0: xxxx.dts means r8a77990-es10-ebisu.dts,
r8a77990-es10-ebisu-4d.dts
E3 Ver.1.1: xxxx.dts means r8a77990-ebisu.dts, r8a77990-ebisu-4d.dts)

```
mmp_reserved: linux,multimedia@XXXXXXXX {
        compatible = "shared-dma-pool";
        reusable;
        reg = <0x00000000 0xXXXXXXXX 0x0 0xYYYYYYYY>;
};
```

0xXXXXXXXX is start address of CMA area and set to the same value as MM_OMXBUF_ADDR.
0xYYYYYYYY is size of CMA area and set to the same value as MM_OMXBUF_SIZE.

Step.3
Build the kernel image by Yocto bitbake command. Follow Yocto Recipe Start-up Guide to set up Yocto environment.

```
$ bitbake linux-renesas -c compile -f
```

3) Building the kernel module

About the space reserved as CMA area for MMP, MMNGR uses MM_OMXBUF_ADDR as the start address, and uses MM_OMXBUF_SIZE as the size.
If you change the start address or the size of the space reserved as CMA area for MMP, perform the following steps.
If you don’t need to change the values, follow Step.2 only.

Step.1

Modify the definition in mmngr_private.h.
MM_OMXBUF_ADDR => the start address of OMXBUF
Set the address aligned on 4KB.
MM_OMXBUF_SIZE => the size of OMXBUF.
Set the size aligned on 4KB.

Step.2
Execute make under mmngr_drv/mmngr/mmngr-module/files/mmngr/drv.
cd mmngr_drv/mmngr/mmngr-module/files/mmngr/drv
$ make

In addition, please note that E3 does not support DTV (SSP).
Therefore, the space reserved as CMA area for MMP for DTV (SSP) is not available on E3.

For other SoCs, if you want to use the space reserved as CMA area for MMP for DTV (SSP), perform the following steps.
About the space reserved as CMA area for MMP for DTV (SSP),
MMNGR uses MM_SSPBUF_ADDR as the start address, and uses MM_SSPBUF_SIZE as the size.
If you don’t need to change the values, follow Step.3 only.

Step.1
Add the definition for DTV (SSP).
$ export MMNGR_SSP_CONFIG=MMNGR_SSP_ENABLE

Step.2
Modify the definition in mmngr_private.h.
MM_SSPBUF_ADDR => the start address of SSPBUF.
Set the address aligned on 4KB.
MM_SSPBUF_SIZE => the size of SSPBUF.
Set the size aligned on 4KB.

Note)
You should change to the proper values in accordance with your DVD solution.
In addition, don't include 1GB order from MM_SSPBUF_ADDR to (MM_SSPBUF_ADDR + MM_SSPBUF_SIZE - 1).
Don’t overlap the area of OMXBUF, if you use SSPBUF.

Step.3 3
Execute make under mmngr_drv/mmngr/mmngr-module/files/mmngr/drv.
$ cd mmngr_drv/mmngr/mmngr-module/files/mmngr/drv
$ make

Set OMXBUF and SSPBUF you use within CMA area for MMP.
Set the total size of OMXBUF and SSPBUF within the size of CMA area for MMP describing under node linux,multimedia in device tree.

4) Verifying the kernel image and the kernel module

Make sure that the following kernel image is created at folder $KERNELSRC/arch/arm64/boot.

Image

Make sure that the following kernel module is built under mmngr_drv/mmngr/mmngr-module/files/mmngr/drv.

mmngr.ko

In addition, if you use the dmabuf function of MMNGR, follow 5) and 6).

5) Building the kernel module

Execute make under mmngr_drv/mmngrbuf/mmngrbuf-module/files/mmngrbuf/drv

$ cd mmngr_drv/mmngrbuf/mmngrbuf-module/files/mmngrbuf/drv

$ make

6) Verifying the kernel module

Make sure that the following kernel module is built under mmngr_drv/mmngrbuf/mmngrbuf-module/files/mmngrbuf/drv.

mmngrbuf.ko

2.3 Storing the shared library, the Kernel module and Kernel image.

The following procedure shows the steps storing the shared library, the kernel module and Kernel image that are built at 2.1. and 2.2.

1) Storing the shared library
For 64-bit shared library, store libmmngr.so.x.x.x under /usr/lib in the rootfs of the target board.
$ cp libmmngr.so.x.x.x (the rootfs of the target board)/usr/lib

2) Storing the kernel module
Store the kernel module under the directory in the rootfs of the target board.
$ mkdir /tmp (/tmp is an example. Therefore change the directory according to your environment.)
$ cp mmngr.ko /tmp

3) Storing Kernel image
Store Kernel image (i.e. Image) in the target board according to the Yocto Recipe Start-up guide.

4) Copying the link to the target rootfs
Copy the links to the target rootfs of the target board at PC.
For 64-bit shared library,
$ cp -d libmmngr.so.x (the rootfs of the target board)/usr/lib
$ cp -d libmmngr.so (the rootfs of the target board)/usr/lib

5) Install the kernel module at the target board.
Execute insmod under the directory storing the kernel module at the target board.
$ cd /tmp
$ insmod mmngr.ko

In addition, if you use the dmabuf function of MMNGR, follow from 6) to 9).

6) Storing the shared library.
For 64-bit shared library, store libmmngrbuf.so.x.x.x under /usr/lib in the rootfs of the target board.
$ cp libmmngrbuf.so.x.x.x (the rootfs of the target board)/usr/lib

7) Storing the kernel module.
Store the kernel module under the directory in the rootfs of the target board.
$ cp mmngrbuf.ko /tmp (/tmp is an example. Therefore change the directory according to your environment.)

8) Copying the link to the target rootfs.
Copy the links to the target rootfs of the target board at PC.
For 64-bit shared library,
$ cp -d libmmngrbuf.so.x (the rootfs of the target board)/usr/lib
$ cp -d libmmngrbuf.so (the rootfs of the target board)/usr/lib

9) Install the kernel module at the target board.
Execute insmod under the directory storing the kernel module at the target board.
$ cd /tmp
$ insmod mmngrbuf.ko

2.4 Device tree configuration

Below describes the information of MMNGR nodes in device tree.

1) For MMNGR

Table 2-1 Information of MMNGR node in device tree

Name	Contents
compatible	Should contain "renesas,mmngr". It's compatible string for MMNGR kernel module.
memory-region	Should contain "&mmp_reserved" to explicitly assign CMA area for MMP to MMNGR kernel module. Should contain "&lossy_decompress" to explicitly assign CMA area for Lossy to MMNGR kernel module.

2) For MMNGRBUF

Table 2-2 Information of MMNGRBUF node in device tree

Name	Contents
compatible	Should contain "renesas,mmngrbuf". It's compatible string for MMNGRBUF kernel module.

3. Module Configuration

The module configuration of MMNGR is as follows.

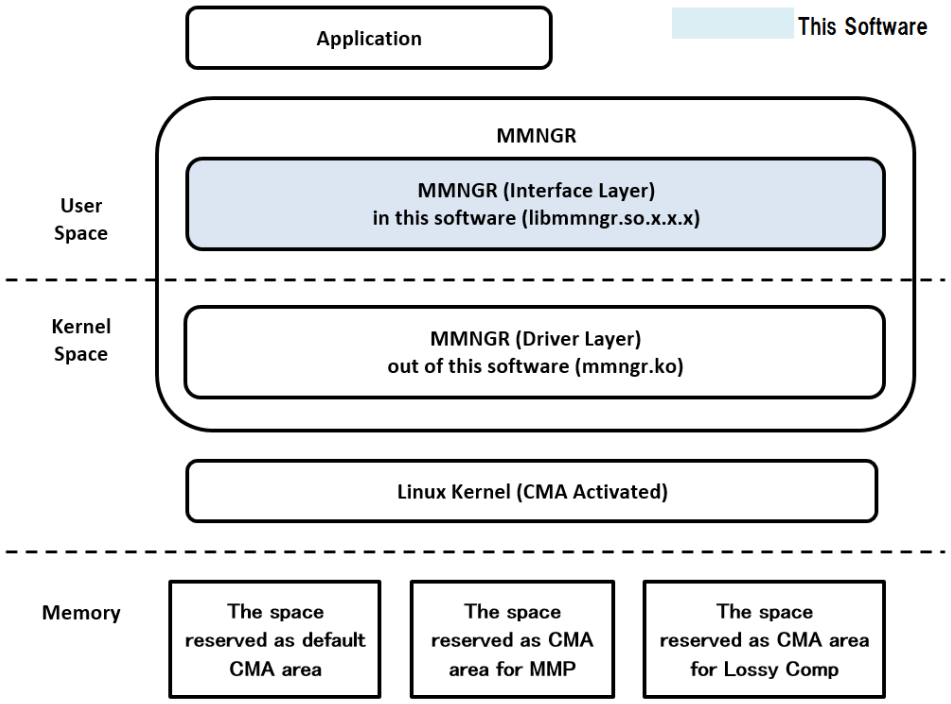

Figure 3-1 MMNGR module configuration

・MMNGR consists of the interface layer and the driver layer.
Interface Layer (in this software)
This layer is in the user space and provides API.
Driver Layer (out of this software)
This layer is in the kernel space and calls the kernel API.

・The execution context of MMNGR is the process or the thread calling MMNGR.

That is, MMNGR does not have the process and the thread.

The module configuration of the dmabuf function of MMNGR is as follows.

The modules of the dmabuf function are independent from the above modules.

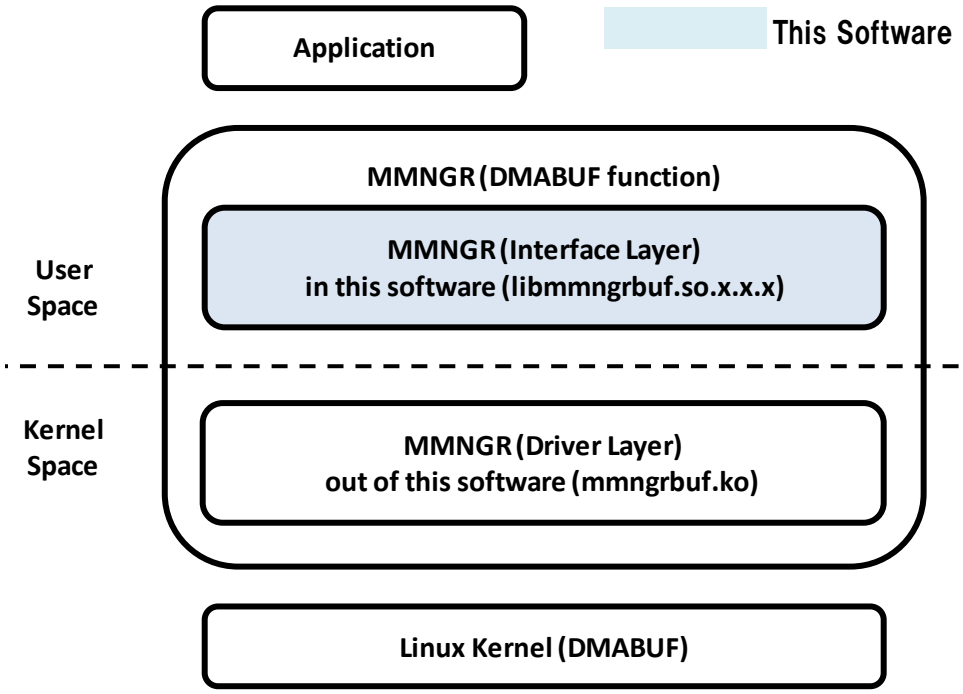

Figure 3-2 MMNGR (DMABUF function) module configuration

- MMNGR consists of the interface layer and the driver layer.
  Interface Layer (in this software)
  This layer is in the user space and provides API.
  Driver Layer (out of this software)
  This layer is in the kernel space and calls the kernel API.

- The execution context of MMNGR is the process or the thread calling MMNGR.
  That is, MMNGR does not have the process and the thread.

4. List of API

Table 4-1 List of API

No	Name	Function
1	mmngr_alloc_in_user_ext()	Allocate the memory space
2	mmngr_free_in_user_ext()	Free the memory space
3	mmngr_debug_map_va_ext()	Map the memory space for H/W IP to the user space. (Debug API for the space reserved as CMA area for MMP)
4	mmngr_debug_unmap_va_ext()	Unmap the memory space mapped to the user space (Debug API for the space reserved as CMA area for MMP)
5	mmngr_export_start_in_user_ext()	Start the export of dmabuf fd.
6	mmngr_export_end_in_user_ext()	End the export of dmabuf fd.
7	mmngr_import_start_in_user_ext()	Start the import of dmabuf fd.
8	mmngr_import_end_in_user_ext()	End the import of dmabuf fd.

5. Sample Sequence

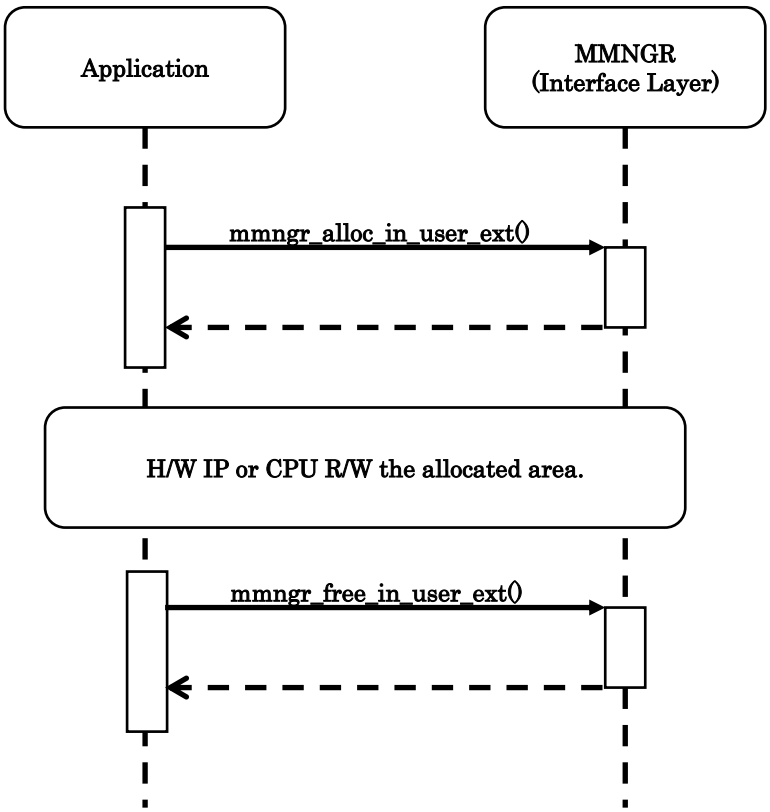

Figure 5-1 MMNGR sample sequence

If you use the dmabuf function of MMNGR, sample sequence is as follows.

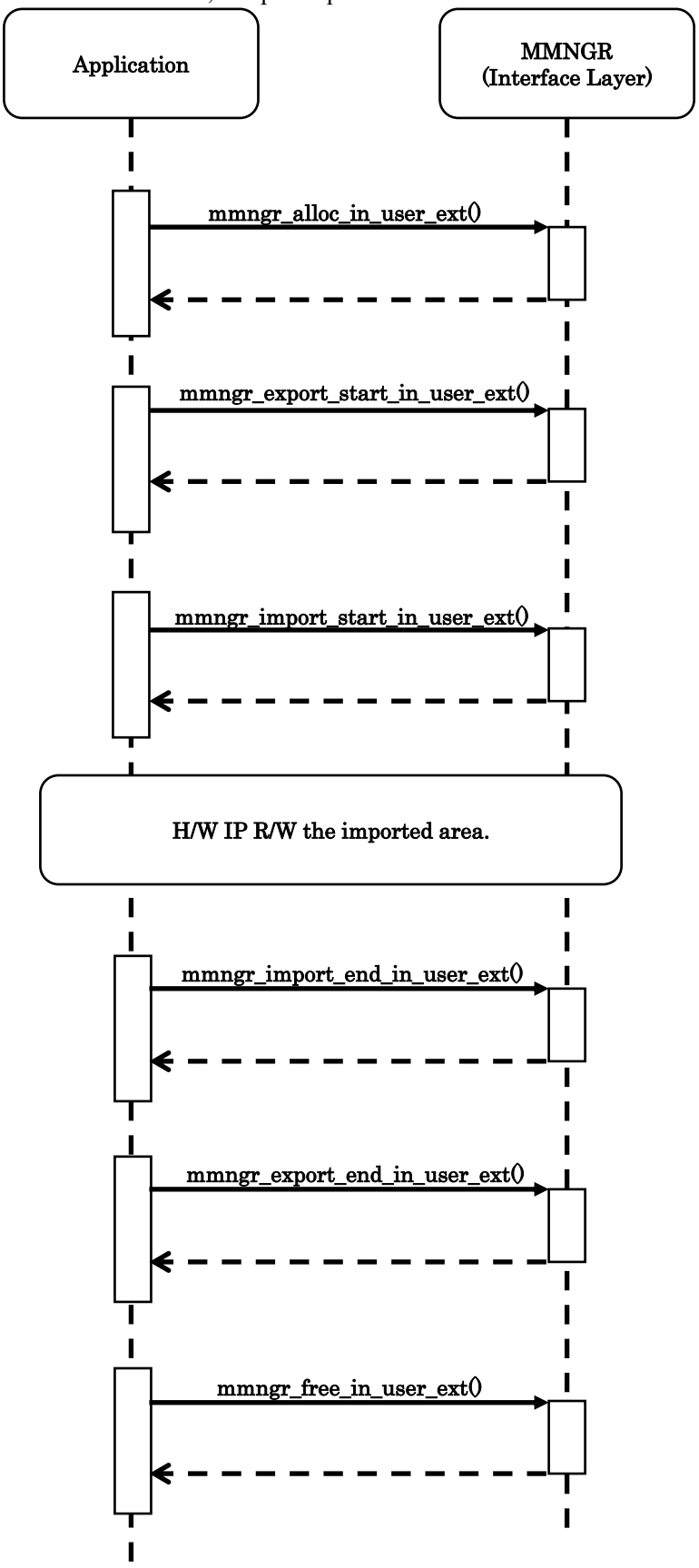

Figure 5-2 MMNGR (DMABUF function) sample sequence

6. API Specification

6.1 Allocate Memory Space

Name

mmngr_alloc_in_user_ext

Synopsis

int mmngr_alloc_in_user_ext(

MMNGR_ID **pid*,	(output)
size_t *size*,	(input)
unsigned int **phard_addr*,	(output)
void ***puser_virt_addr*,	(output)
unsigned int *flag*,	(input)
void **mem_param*	(input and output)

)

Arguments

MMNGR_ID **pid*	The address storing the id of the allocated memory
size_t *size*	The size of the allocation [Byte]
unsigned int **phard_addr*	The address storing the address for H/W IP of the allocated memory. It is a virtual address for H/W IP, not actual physical address when IPMMU MMU support is enabled.
void ***puser_virt_addr*	The address storing the address for CPU of the allocated memory
unsigned int *flag*	The flag to select the target of the allocation. MMNGR_VA_SUPPORT The space reserved as default CMA area MMNGR_PA_SUPPORT The space reserved as CMA area for MMP MMNGR_PA_SUPPORT_SSP The space reserved as CMA area for MMP for DTV (SSP) (This definition is valid, only when MMNGR_SSP_ENABLE is defined.) MMNGR_PA_SUPPORT_LOSSY The space reserved as CMA area for Lossy Decompression feature.
void **mem_param*	The address storing the data buffer, which contains detail information of Lossy memory allocation, request and status of Lossy support. The data buffer is a variable with type MM_FUNC struct.

Struct

-

Return Value

R_MM_OK	Normal End
R_MM_FATAL	Fatal Error
R_MM_PARE	Parameter Error (Condition) (1) When an application sets NULL to *pid*, *phard_addr* or *puser_virt_addr*. (2) When an application sets 0 to size. (3) When an application sets the undefined value at *flag.* (4) When *flag* is MMNGR_PA_SUPPORT_LOSSY and *mem_param* is NULL. (5) When *flag* is MMNGR_PA_SUPPORT_LOSSY and content of data buffer pointed by *mem_param* contains undefined value or unsupported value. Refer to 10.2 for more details.
R_MM_NOMEM	Lack of Memory (Condition) When MMNGR cannot allocate the space according to *size*.

Description

(1) This function allocates the memory space according to *size*. If *flag* is MMNGR_PA_SUPPORT_LOSSY, the allocation will also refer to mem_param.

(2) The allocated size is rounded up to 4KB alignment.

(3) The alignment of the address pointed by *phard_addr* and *puser_virt_addr* is 4KB.

(4) This function allocates the memory space with CMA.
According to the specification of CMA, the allocation time may be long, and the size of the total allocation may be smaller than the CMA size defined in device tree.

(5) MMNGR does not have the defragmentation.

(6) Only when MMNGR_VA_SUPPORT is set to *flag*, the address is set to *puser_virt_addr*.

(7) The space pointed by *puser_virt_addr* is the user space and the **non-cache space**.

Notes

(1) Allocate the space set to *pid, phard_addr and puser_virt_addr* in the application.
The type of the space set to *pid* is MMNGR_ID.

Example

```
MMNGR_ID id;
unsigned int hard_addr;
void *user_virt_addr;
unsigned int flag = MMNGR_VA_SUPPORT;
mmngr_alloc_in_user_ext(&id, 64 * 1024, &hard_addr,
                        &user_virt_addr, flag, NULL);
```

(2) Align the address pointed by *phard_addr* and set the aligned address to H/W IP.

(3) Set the address *puser_virt_addr* to CPU.

(4) Use the address pointed by *puser_virt_addr* in the same process calling this API.

(5) This function wastes one file descriptor per one allocation.

Therefore, decide the number of the allocation within the maximum of the file descriptor per one process in the application.

(6) Don't refer to the address pointed by *puser_virt_addr*, when the space is allocated with MMNGR_PA_SUPPORT, MMNGR_PA_SUPPORT_SSP or MMNGR_PA_SUPPORT_LOSSY.

(7) Use valid *mem_param* for Lossy mem allocation. Information about supported Lossy format will be written into buffer pointed by mem_param.

Example

```
unsigned int flag = MMNGR_PA_SUPPORT_LOSSY;
unsigned int conf;
struct MM_FUNC mem_param;
mem_param.func = MM_FUNC_LOSSY_ENABLE;
mem_param.type = MM_FUNC_TYPE_LOSSY_AREA;
mem_param.attr = MM_FUNC_FMT_LOSSY_YUVPLANAR;
mem_param.conf = &conf;
mmngr_alloc_in_user_ext(&id, 64 * 1024, &hard_addr,
                        &user_virt_addr, flag, &mem_param);
if (mem_param.conf == MM_FUNC_STAT_LOSSY_SUPPORT)
                        /* Use the allocated mem in hard_addr */
```

(8) Map the address to the user space with mmngr_debug_map_va_ext() and unmap the mapped address with mmngr_debug_unmap_va_ext(), when you want to refer to the space allocated with MMNGR_PA_SUPPORT or MMNGR_PA_SUPPORT_SSP or MMNGR_PA_SUPPORT_LOSSY in the debug.

(9) Don't call this API at the context of a signal handler.

(10) Use the address allocated with MMNGR_PA_SUPPORT_SSP to set to DTV (SSP) only. That is, use the address allocated with MMNGR_PA_SUPPORT_SSP for the buffer of DTV DEMUX IP only.

(11) When a device (FDP or VSPI or VSPB) enables Lossy compression feature, a compressed data is required to be stored in a buffer allocated with MMNGR_PA_SUPPORT_LOSSY. In addition to this, the image format of a compressed data is required to be equivalent to that of the allocated buffer.

(12) When IPMMU MMU is enabled, returned value at *phard_addr* argument is not actual physical address as CPU view. Instead, the address is virtual address for H/W IP. Especially, if flag is set to MMNGR_VA_SUPPORT, the virtual address is equal to actual physical address.

See Also

-

6.2 Free Memory Space

Name

mmngr_free_in_user_ext

Synopsis

```
int mmngr_free_in_user_ext(
    MMNGR_ID id          (input)
)
```

Arguments

MMNGR_ID *id*	ID of the allocated memory to be freed.

Struct

-

Return Value

R_MM_OK	Normal End
R_MM_FATAL	Fatal Error
R_MM_SEQE	Sequence Error (Condition) When the space of ID has already been freed. Parameter Error
R_MM_PARE	(Condition) (1) When the space of ID has undefined flag value.

Description

(1) This function frees the space of ID.

Notes

(1) Set ID, which is allocated by mmngr_alloc_in_user_ext() and is not freed, to *id*.

(2) Don’t call this API at the context of a signal handler.

See Also

-

6.3 Map the Address for H/W IP to the User Space (Debug API)

Name

mmngr_debug_map_va_ext

Synopsis

```
int mmngr_debug_map_va_ext(
    MMNGR_ID *pid,              (output)
    size_t size,                (input)
    unsigned int hard_addr,     (input)
    void **puser_virt_addr      (output)
    void *mem_param             Do not use
)
```

Arguments

MMNGR_ID **pid*	The address storing the id of the mapping space
size_t *size*	The size of the mapping space [Byte]
unsigned int *hard_addr*	The address for H/W IP to map to the user space. It is a virtual address for H/W IP, not actual physical address when IPMMU MMU support is enabled.
void ***puser_virt_addr*	The address storing the address for CPU
void **mem_param*	Do not use

Struct

-

Return Value

R_MM_OK	Normal End
R_MM_FATAL	Fatal Error
R_MM_PARE	Parameter Error (Condition) (1) When the application sets NULL to *pid* or *user_virt_addr*. (2) When the application sets 0 to *size*.

Description

(1) This function maps the address for H/W IP to the user space according to *size* and h*ard_addr*.

(2) The size of the mapping is rounded up to 4KB.

(3) The alignment of the address pointed by *puser_virt_addr* is 4KB.

(4) The space pointed by *puser_virt_addr* is the user space and the **non-cache space**.

Notes

(1) Don’t use this function except for the debug.

(2) Allocate the space set to pid and *puser_virt_addr* in the application.
The type of the space set to pid is MMNGR_ID.

(3) Set size and phard addr of the space allocated by mmngr_alloc_in_user_ext()
with MMNGR_PA_SUPPORT or MMNGR_PA_SUPPORT_SSP or
MMNGR_PA_SUPPORT_LOSSY to *size* and *hard_addr*.

(4) Use the address pointed by *puser_virt_addr* in the same process calling this API.

(5) This function wastes one file descriptor per one allocation. Therefore, decide the number of the allocation within the maximum of the file descriptor per one process in the application.

(6) Don’t call this API at the context of a signal handler.

(7) When IPMMU MMU is enabled, the address in *hard_addr* will be converted to actual physical address before creating the user space mapping. Therefore, the userspace access can be done normally.

See Also

-

6.4 Unmap the Mapping Space (Debug API)

Name

mmngr_debug_unmap_va_ext

Synopsis

```
int mmngr_debug_unmap_va_ext(
    MMNGR_ID id                 (input)
)
```

Arguments

MMNGR_ID *id*	ID of the unmapping space

Struct

-

Return Value

R_MM_OK	Normal End
R_MM_FATAL	Fatal Error
R_MM_SEQE	Sequence Error (Condition) When the space of ID has already been unmapped.

Description

(1) This function unmaps the space of ID.

Notes

(1) Don’t use this function except for the debug.

(2) Set ID, which is mapped by mmngr_debug_map_va_ext() and is not unmapped, to *id*.

(3) Don’t call this API at the context of a signal handler.

See Also

-

6.5 Start the export of dmabuf fd

Name

mmngr_export_start_in_user_ext

Synopsis

int mmngr_export_start_in_user_ext(

int **pid*,	(output)
size_t *size*,	(input)
unsigned int *hard_addr*,	(input)
int **pbuf*	(output)
void **mem_param*	Do not use

)

Arguments

int **pid*	The address storing the id of the export.
size_t *size*	The size to connect to dmabuf fd
unsigned int *hard_addr*	The address for H/W IP to connect to dmabuf fd It is a virtual address for H/W IP, not actual physical address when IPMMU MMU support is enabled.
int **pbuf*	The address storing dmabuf fd
void **mem_param*	Do not use

Struct

-

Return Value

R_MM_OK	Normal End
R_MM_FATAL	Fatal Error
R_MM_PARE	Parameter Error (Condition) (1) When an application sets NULL to *pid* or *pbuf*. (2) When an application sets 0 to *size*.

Description

(1) This function allocates dmabuf fd from the address and the size of the memory space.

Notes

(1) Allocate the space set to *pid* and *pbuf* in the application.

(2) Set the size of the space, which is allocated by mmngr_alloc_in_user_ext() and is not freed, to size.

(3) Set the address of the space, which is allocated by mmngr_alloc_in_user_ext() and is not freed, to hard_addr.

(4) Set the size aligned by 4KB to *size*

(5) Set the address aligned by 4KB to *hard_addr*.

(6) This function wastes two file descriptors per one export.
One is used for ID. The other is used for dmabuf fd.

Therefore, decide the number of the allocation within the maximum of the file descriptor per one process in the application.

(7) Use dmabuf fd pointed by *pbuf* in the same process calling this API.

(8) Don’t access the space pointed by *hard_addr* with CPU

(9) Don’t call this API at the context of a signal handler.

(10) Don’t free the space to set to *size* and *hard_addr* until mmngr_export_end_in_user_ext() finishes.

(11) Don’t execute mmap() for dmabuf fd pointed by *pbuf*.

See Also

-

6.6 End the export of dmabuf fd

Name

mmngr_export_end_in_user_ext

Synopsis

```
int mmngr_export_end_in_user_ext(
    int id                          (input)
)
```

Arguments

int *id*	ID of export ended

Struct

-

Return Value

R_MM_OK	Normal End
R_MM_FATAL	Fatal Error

Description

(1) This function frees dmabuf fd for id.

Notes

(1) Set ID, which is started by mmngr_export_start_in_user_ext() and is not ended, to *id*.

(2) Don’t set ID for the dmabuf fd, while dmabuf fd is imported.

(3) Don’t call this API at the context of a signal handler.

(4) In “End the export of dmabuf fd” process, DMA Buffer Sharing framework will call dmabuf_release() to free the memory allocated for dmabuf only when the refcount becomes 0. The buffer will not be freed by MNGRBUF API (dmabuf_release() function) but by MMNGR API instead (mmngr_free_in_user_ext() function) since MMNGR is the one allocated the buffer. Please prefer Figure 5.2 for a sample sequence.
If user allocated the buffer via their own API, please free the memory allocated for dmabuf after “End the export of dmabuf fd” process .

See Also

-

6.7 Start the import of dmabuf fd

Name

mmngr_import_start_in_user_ext

Synopsis

int mmngr_import_start_in_user_ext(

int **pid*,	(output)
size_t **psize*,	(output)
unsigned int **phard_addr*,	(output)
int *buf*	(input)
void **mem_param*	Do not use

)

Arguments

int **pid*	The address storing the id of the import.
size_t **psize*	The address storing the size connected to dmabuf fd
unsigned int **phard_addr*	The address storing the address for H/W IP connected to dmabuf fd. It is a virtual address for H/W IP, not actual physical address when IPMMU MMU support is enabled.
int *buf*	dmabuf fd
void **mem_param*	Do not use

Struct

-

Return Value

R_MM_OK	Normal End
R_MM_FATAL	Fatal Error
R_MM_PARE	Parameter Error (Condition) (1) When an application sets NULL to pid, psize or phard_addr.

Description

(1) This function gets the address and the size of the memory space from dmabuf fd.

Notes

(1) Set the dmabuf fd, which is started by mmngr_export_start_in_user_ext() and is not ended, to *buf*.

(2) Set the dmabuf fd, which is exported by the other software and is not closed, to *buf*.

(3) Set the dmabuf fd connected to one physical contiguous space only.

(4) Don’t call this API at the context of a signal handler.

See Also

-

6.8 End the import of dmabuf fd

Name

mmngr_import_end_in_user_ext

Synopsis

```
int mmngr_import_end_in_user_ext(
    int id                          (input)
)
```

Arguments

int *id*	ID of import ended

Struct

-

Return Value

R_MM_OK	Normal End
R_MM_FATAL	Fatal Error

Description

(1) This function ends using the address and the size got from dmabuf fd for id.

Notes

(1) Set ID, which is started by mmngr_import_start_in_user_ext() and is not ended, to id.

(2) Don’t set ID for the dmabuf fd, while the space to connected to dmabuf fd is used.

(3) Don’t call this API at the context of a signal handler.

See Also

-

7. Header Files

Include mmngr_user_public.h in the application, when the application calls the APIs of MMNGR.

Include mmngr_buf_user_public.h in the application, when the application calls the APIs of DMABUF function of MMNGR (DMABUF function).

8. Definition

8.1 Structure

8.1.1 Lossy support structure

The following is described about the member of MM_FUNC structure.

```
struct MM_FUNC {
        unsigned int func;
        unsigned int type;
        unsigned int attr;
        unsigned int *conf;
};
```

Table 8-1 Members of MM_FUNC structure

Member	Direction	Contents
unsigned int func	Input	Specify Lossy support in memory allocation request. Two available options are: * MM_FUNC_LOSSY_ENABLE * MM_FUNC_LOSSY_DISABLE
unsigned int type	Input	Specify type of Lossy area. Two available options are: * MM_FUNC_TYPE_LOSSY_AREA * MM_FUNC_TYPE_LOSSY_SHADOW NOTE: The supported types are depended on the system configuration.
unsigned int attr	Input	Specify image format storing in Lossy area. Three available options are: * MM_FUNC_FMT_LOSSY_YUVPLANAR * MM_FUNC_FMT_LOSSY_YUV422INTLV * MM_FUNC_FMT_LOSSY_ARGB8888 NOTE: The supported image formats are depended on the system configuration.
unsigned int *conf	Output	Specify whether the Lossy memory allocation request is supported or not. User should check the output value before using the allocated buffer. Two possible output value are: * MM_FUNC_STAT_LOSSY_SUPPORT * MM_FUNC_STAT_LOSSY_NOT_SUPPORT

8.2 Macros

8.2.1 Return Value Definition

Table 8-2 List of Return Value Definition

Definition	Value	Content
R_MM_OK	0	Normal End
R_MM_FATAL	-1	Fatal Error
R_MM_SEQE	-2	Sequence Error
R_MM_PARE	-3	Parameter Error
R_MM_NOMEM	-4	Lack of Memory

8.2.2 Parameter Definition

Table 8-3 List of Parameter Definition

Definition	Value	Content
MMNGR_VA_SUPPORT	0	The space reserved as default CMA area
MMNGR_PA_SUPPORT	1	The space reserved as CMA area for MMP
MMNGR_PA_SUPPORT_SSP	3	The space reserved as CMA area for MMP for DTV (SSP)
MMNGR_PA_SUPPORT_LOSSY	4	The space reserved as CMA area for Lossy compression feature

8.2.3 Parameter Definition for Lossy mem allocation

Table 8-4 List of Parameter Definition for Lossy mem allocation

Definition	Value	Content
MM_FUNC_LOSSY_DISABLE	0	The non-Lossy mem allocation is requested. It has same effect as memory request with MMNGR_PA_SUPPORT.
MM_FUNC_LOSSY_ENABLE	1	The Lossy mem allocation is requested.
MM_FUNC_TYPE_LOSSY_AREA	1	The type of Lossy area is a Lossy area in visible physical memory.
MM_FUNC_TYPE_LOSSY_SHADOW	2	The type of Lossy area is a Lossy area in shadow physical mem.
MM_FUNC_FMT_LOSSY_YUVPLANAR	1	The image format in Lossy Area is YUV Planar.
MM_FUNC_FMT_LOSSY_YUV422INTLV	2	The image format in Lossy Area is YUV 422 Interleave.
MM_FUNC_FMT_LOSSY_ARGB8888	3	The image format in Lossy Area is ARGB8888.
MM_FUNC_STAT_LOSSY_NOT_SUPPORT	0	The requested Lossy mem alloc is not supported.
MM_FUNC_STAT_LOSSY_SUPPORT	1	The requested Lossy mem alloc is supported.

9. Restrictions and Notices

9.1 Restrictions

- For Lossy areas, only support up to 3 Lossy areas, each area has different image format (1 area for YUV Planar, 1 area for YUV 422 interleave and 1 area for ARGB8888).
- For Lossy areas, does not support type of Lossy area in shadow physical mem (In case of MM_FUNC_TYPE_LOSSY_SHADOW).
- When IPMMU MMU is enabled, the first 1GB DDR physical address space in legacy area (0x40000000 - 0x80000000 on H3/M3/M3N/E3) should be mapped straight-forwardly in virtual address space for H/W IP.

9.2 Notices

- It depends on the number, the requested size and the order of memory allocation, a memory allocation may fail even if the total amount of un-used memory is enough. This situation is almost same as when fragmentation occurs.
- Default CMA area should be kept in legacy 32-bit address space.
- For Lossy mem allocation support, need corresponding IPL support and Linux kernel support.
  - On E3 Ver.1.0 + Ebisu-2D (1GB DDR): not support Lossy compression feature.
  - On E3 Ver.1.x + Ebisu-4D (2GB DDR): support Lossy compression feature.
- When IPMMU MMU is enabled and mmngr_alloc_in_user_ext() is called, returned value at phard_addr argument is not actual physical address as CPU view. Instead, the address is virtual address for H/W IP.
- On M3N and E3, there's no 40-bit address space. IPMMU MMU will support physical address space in legacy area.
- In "End the export of dmabuf fd" process, DMA Buffer Sharing framework will call dmabuf_release() to free the memory allocated for dmabuf only when the refcount becomes 0. The buffer will not be freed by MMNGRBUF API (dmabuf_release() function) but by MMNGR API instead (mmngr_free_in_user_ext() function) since MMNGR is the one allocated the buffer. Please prefer Figure 5.2 for a sample sequence. If user allocated the buffer via their own API, please free the memory allocated for dmabuf after "End the export of dmabuf fd" process.

10. Appendix

10.1 Legacy API

Using the following legacy APIs is NOT recommended because we only support them to keep the compatibility of software developed in Gen2 platform.

Table 10-1 List of Legacy API

No	Name	Function	Difference with Gen2 API
1	mmngr_alloc_in_user()	Allocate the memory space	Using pphy_addr is prohibited. It returns 0xffffffffffffffff (64bit).
2	mmngr_free_in_user()	Free the memory space	-
3	mmngr_debug_map_va()	Map the memory space for H/W IP to the user space. (Debug API for the space reserved as CMA area for MMP)	-
4	mmngr_debug_unmap_va()	Unmap the memory space mapped to the user space (Debug API for the space reserved as CMA area for MMP)	-
5	mmngr_export_start_in_user()	Start the export of dmabuf fd.	-
6	mmngr_export_end_in_user()	End the export of dmabuf fd.	-
7	mmngr_import_start_in_user()	Start the import of dmabuf fd.	-
8	mmngr_import_end_in_user()	End the import of dmabuf fd.	-

10.2 Relationship between mem_param and Lossy mem return value

Below table shows the relationship between Lossy request data in MM_FUNC struct passing via mem_param parameter and memory allocation return value.

Table 10-2 Relationship between mem_param and Lossy mem return value

Case	Return value	conf (in MM_FUNC structure)
- *mem_param* is NULL	R_MM_PARE	(not set)
- *mem_param* is valid. - However, specify type is MM_FUNC_TYPE_LOSSY_SHADOW.		(not set)
- *mem_param* is valid. - However, specify undefined memory allocation request, i.e. neither MM_FUNC_LOSSY_DISABLE nor MM_FUNC_LOSSY_ENABLE		(not set)
- *mem_param* is valid. - However, specify undefined image format. i.e. an undefined format, neither YUVPLANAR, YUV422INTLV nor ARGB8888.		(not set)
- *mem_param* is valid. - Specify a defined image format i.e. YUVPLANAR, YUV422INTLV or ARGB8888 - However, the system does NOT support that format.		MM_FUNC_STAT_LOSSY_NOT_SUPPORT
- *mem_param* is valid. - Specify a defined image format. i.e. YUVPLANAR, YUV422INTLV or ARGB8888 - The system supports that format. - However, the memory allocation fails.	R_MM_NOMEM	MM_FUNC_STAT_LOSSY_SUPPORT
- *mem_param* is valid. - Specify a defined image format. i.e. YUVPLANAR, YUV422INTLV or ARGB8888 - The system supports that format. - Moreover, the memory allocation is OK.	R_MM_OK	MM_FUNC_STAT_LOSSY_SUPPORT

10.3 IPMMU MMU support

10.3.1 Enable IPMMU MMU support

Below shows the procedure to enable IPMMU MMU support, and can support moving the CMA area for MMP, CMA area for Lossy compression feature to 40-bit physical address space.

1) Notices for IPMMU MMU support.

MMNGR will manage below registers:
- In IPMMU Main Memory (IPMMU-MM) (8 registers for 8th page table):
  - IMCTRn, IMTTBCRn, IMTTUBR0n, IMTTLBR0n, IMMAIR0n, IMELARn, IMEUARn, IMSTRn
- In IPMMU cache (3 registers corresponding to each IPMMU master):
  - IMCTRn, IMUCTRn, IMSCTLR.

On M3N and E3, CMA areas will be kept in legacy areas when IPMMU MMU is enabled.

2) Customize CMA areas' definitions in kernel source code

CMA area for MMP

Modify the start address and size of CMA area for MMP under node linux,multimedia in device tree arch/arm64/boot/dts/renesas/xxxx.dts. (H3 Ver.1.x: xxxx.dts means r8a77950-salvator-x.dts,
H3 Ver.2.0: xxxx.dts means r8a77951-salvator-x.dts, r8a77951-salvator-xs.dts,
H3 Ver.3.0: xxxx.dts means r8a77951-salvator-xs-4x2g.dts,
M3 Ver.1.x: xxxx.dts means r8a77960-salvator-x.dts, r8a77960-salvator-xs.dts,
M3 Ver.3.0: xxxx.dts means r8a77961-salvator-xs-2x4g.dts,
M3N : xxxx.dts means r8a77965-salvator-x.dts, r8a77965-salvator-xs.dts,
E3 Ver.1.0: xxxx.dts means r8a77990-es10-ebisu.dts,
r8a77990-es10-ebisu-4d.dts
E3 Ver.1.1: xxxx.dts means r8a77990-ebisu.dts, r8a77990-ebisu-4d.dts

```
mmp_reserved: linux,multimedia@AAAAAAAAA {
        compatible = "shared-dma-pool";
        reusable;
        reg = <0xZ 0xXXXXXXXX 0x0 0xYYYYYYYY>;
};
```

0xZ is the first 8-bit in 40-bit physical start address.
0xZ = 0x6 means the physical start address will begin with 0x06_XXXXXXXX.
For H3, its value can be 0x0, 0x5, 0x6 or 0x7. For M3, its value can be 0x0 or 0x6.
For M3N, its value must be 0x0. For E3, its value must be 0x0.

0xXXXXXXXX is last 32-bit in 40-bit physical address.
0xYYYYYYYY is size of CMA area.
0xAAAAAAAAA is 40-bit physical address, which has:
- The first 8-bit in 40-bit physical start address is same as 0xZ
- The last 32-bit in 40-bit physical address is same as 0xXXXXXXXX

CMA area for Lossy compression feature

Modify the start address and size of CMA area for Lossy under node linux,lossy_decompress in device tree.

```
lossy_decompress: linux,lossy_decompress@0xAAAAAAAAA {
        no_map;
        reg = <0xZ 0xXXXXXXXX 0x0 0xYYYYYYYY>;
};
```

Default CMA area

When CMA areas (i.e. CMA area for MMP, CMA area for Lossy) are moved to 40-bit address space, we can expand either default CMA area or kernel heap in legacy area.

For example, in the case of moving CMA area for MMP to 40-bit address space:

- Default CMA area can increase +256 MB.
- Kernel heap in legacy area can increase +256 MB.

To modify the start address and size of default CMA area, refer to Yocto Recipe Start-Up Guide.

If you do not use the space reserved as CMA area for MMP for DTV (SSP), follow 4), 5) and 6)

3) Use the space reserved as CMA area for MMP for DTV (SSP)

The configuration for the space reserved as CMA area for MMP for DTV (SSP)
is same in both IPMMU enabled and disabled environment

To modify the configuration, refer to chapter 2.2, then follow “3) Building the kernel module”.

For example, in order to support the space reserved as CMA area for MMP for DTV (SSP) in 40-bit address space:
- Refer to 2) in this chapter, to modify the start address and size of CMA area for MMP in 40-bit address space.
- In mmngr_private.h, set MM_SSPBUF_ADDR within CMA area for MMP (40-bit address space).
- In mmngr_private.h, set MM_SSPBUF_SIZE and MM_OMXBUF_SIZE so that their total size also within the size of CMA area for MMP.
- Export the definition for DTV (SSP).

4) Enable IPMMU devices in kernel source code

Enable the selected IPMMU devices, which have IPMMU MMU support in device tree.
arch/arm64/boot/dts/renesas/xxxx.dtsi. (H3 Ver.1.x: xxxx.dtsi means r8a77950.dtsi,
H3 Ver.2.0: xxxx.dtsi means r8a77951.dtsi,
H3 Ver.3.0: xxxx.dtsi means r8a77951.dtsi,
M3 Ver.1.x: xxxx.dtsi means r8a77960.dtsi,
M3 Ver.3.0: xxxx.dtsi means r8a77961.dtsi,
M3N: xxxx.dtsi means r8a77965.dtsi,
E3 Ver.1.0: xxxx.dtsi means r8a77990-es10.dtsi,
E3 Ver.1.1: xxxx.dtsi means r8a77990.dtsi)

H3 Ver.1.x: enable node ipmmu_vp0, ipmmu_vc0, ipmmu_vc1, ipmmu_sy (*).
H3 Ver.2.0: enable node ipmmu_vp0, ipmmu_vp1, ipmmu_vc0, ipmmu_vc1, ipmmu_ds1 (*).
H3 Ver.3.0: enable node ipmmu_vp0, ipmmu_vp1, ipmmu_vc0, ipmmu_vc1, ipmmu_ds1 (*).
M3 Ver.1.x: enable node ipmmu_vc0, ipmmu_vi, ipmmu_ds1 (*).
M3 Ver.3.0: enable node ipmmu_vc0, ipmmu_vi, ipmmu_ds1 (*).
M3N: enable node ipmmu_vc0, ipmmu_vp0, ipmmu_ds1 (*).
E3 Ver.1.0: enable node ipmmu_vc0, ipmmu_vp0.
E3 Ver.1.1: enable node ipmmu_vc0, ipmmu_vp0.

*: When supporting CMA area for MMP for DTV (SSP), you need to enable these nodes.

Below is example of enabling node ipmmu_vp0 for H3 Ver.2.0.

```
ipmmu_vp0: mmu@fe990000 {
        compatible = "renesas,ipmmu-mmu-r8a7795";
        …, <snip> ,
        status = "disabled";
};
```

The property “status” is set “disabled” by default. Please change to “okay” to enable IPMMU MMU.
i.e. status = "okay";

MMNGR driver uses the 8th non-secure page table (8th MMU context or MMU number 7) of 8 MMU contexts and ipmmu-vmsa driver uses the remaining MMU contexts.
Therefore, sharing MMU resources between MMNGR driver and ipmmu-vmsa driver is necessary.

Sharing resources is unnecessary when ipmmu-vmsa driver is disabled. The confirmation can be done by checking Kernel Configuration. (*1)

```
Device Drivers  --->
        [ ] IOMMU Hardware Support  --->
```

(*1): The driver is disabled when it’s not selected.

When ipmmu-vmsa driver is enabled (*2), to reserve the 8th MMU context for MMNGR, make following setting (*3)

in Kernel Configuration.

Device Drivers --->
[*] IOMMU Hardware Support ---> (*2) The driver is enabled when it's selected.
--- IOMMU Hardware Support
[*] Renesas VMSA-compatible IPMMU
(7) Input MMU number of MMU's really used (*3) Change the value from 8 to 7.

In case of making the 8th MMU context reservation for MMNGR, the resource assignment is below.
- MMUn (n = 0 to 6) contexts are for ipmmu-vmsa driver
- MMUn (n = 7) context is for MMNGR driver

Re-build the kernel image by Yocto bitbake command. Follow Yocto Recipe Start-up Guide to set up Yocto environment.
$ bitbake linux-renesas -c compile -f

5) Building the kernel module
To enable IPMMU MMU support in MMNGR, need to change the environment variable.
$ export MMNGR_IPMMU_MMU_CONFIG=IPMMU_MMU_SUPPORT_1GB_PGTABLE

Execute make under mmngr_drv/mmngr/mmngr-module/files/mmngr/drv.
$ cd mmngr_drv/mmngr/mmngr-module/files/mmngr/drv
$ make

6) Verifying the kernel image and the kernel module

Make sure that the following kernel module is built under mmngr_drv/mmngr/mmngr-module/files/mmngr/drv.

mmngr.ko

10.3.2 Map CMA areas to IPMMU virtual address space

Below shows the guideline how to properly map each CMA area to the IPMMU virtual address space.

The physical address of CMA areas should follow below policy in order to create a correct IPMMU PA-VA mapping table.
- Map up to 4GB CMA area to the IPMMU virtual address space.
- Default CMA should be in Legacy area.
  i.e. within 0x40000000 - 0x80000000 (H3), 0x40000000 - 0xC0000000 (M3), 0x40000000 - 0xC0000000 (M3N), 0x40000000 - 0xC0000000 (E3).
- The size of CMA for MMP or CMA for Lossy should be within 1GB and located in a 1GB fixed physical address space.
- All of three CMA areas (or two of them) can co-exist in one entry only when the total size is within 1GB.
- Depend on the availability of physical address space, the mapping table can be customized.

The default PA-VA mapping tables for H3 Ver.1.x, 2.0, 3.0, M3 Ver.1.x, 3.0, M3N and E3 are showed as below.

Table 10-3 PA-VA mapping table for H3 Ver.1.x, 2.0, 3.0

	Physical address	Virtual address
1st Entry (*1)	0x0_4000_0000	0x4000_0000
2nd Entry	0x5_0000_0000	0x8000_0000
3rd Entry	0x6_0000_0000	0xC000_0000
4th Entry (*2)	0x7_0000_0000	0x0000_0000

Table 10-4 PA-VA mapping table for M3 Ver.1.x, 3.0

	Physical address	Virtual address
1st Entry (*1)	0x0_4000_0000	0x4000_0000
2nd Entry	0x0_8000_0000	0x8000_0000
3rd Entry	0x6_0000_0000	0xC000_0000
4th Entry (*2)	0x6_4000_0000	0x0000_0000

Table 10-5 PA-VA mapping table for M3N

	Physical address	Virtual address
1st Entry (*1)	0x0_4000_0000	0x4000_0000
2nd Entry	0x0_8000_0000	0x8000_0000
3rd Entry	0x0000_0000	0x0000_0000
4th Entry	0x0000_0000	0x0000_0000

Table 10-6 PA-VA mapping table for E3

	Physical address	Virtual address
1st Entry (*1)	0x0_4000_0000	0x4000_0000
2nd Entry	0x0_8000_0000	0x8000_0000
3rd Entry	0x0000_0000	0x0000_0000
4th Entry	0x0000_0000	0x0000_0000

NOTE:
- Each entry has a 1GB fixed physical address space.

- The address map of 2nd Entry, 3rd Entry and 4th Entry can be adjusted based on available physical address space.
- *1: The first 1GB DDR physical address space in legacy area should be equal to a virtual address space from H/W IP. i.e, 0x40000000 - 0x80000000 (H3/M3/M3N/E3) should be mapped straight-forwardly.
- *2: Take care that this 1GB physical address space starts with virtual address space with 0.

The necessary translation tables for each CMA area will be calculated automatically based on the following table entries defined in mmngr_private.h.

```
/* Table entries for H3 */
#define H3_IPMMU_ADDR_SECTION_0 0x0700000000ULL
#define H3_IPMMU_ADDR_SECTION_1 0x0040000000ULL
#define H3_IPMMU_ADDR_SECTION_2 0x0500000000ULL
#define H3_IPMMU_ADDR_SECTION_3 0x0600000000ULL

/* Table entries for M3 */
#define M3_IPMMU_ADDR_SECTION_0 0x0640000000ULL
#define M3_IPMMU_ADDR_SECTION_1 0x0040000000ULL
#define M3_IPMMU_ADDR_SECTION_2 0x0080000000ULL
#define M3_IPMMU_ADDR_SECTION_3 0x0600000000ULL

/* Table entries for M3N */
#define M3N_IPMMU_ADDR_SECTION_0 0
#define M3N_IPMMU_ADDR_SECTION_1 0x0040000000ULL
#define M3N_IPMMU_ADDR_SECTION_2 0x0080000000ULL
#define M3N_IPMMU_ADDR_SECTION_3 0

/* Table entries for E3 */
#define M3N_IPMMU_ADDR_SECTION_0 0
#define M3N_IPMMU_ADDR_SECTION_1 0x0040000000ULL
#define M3N_IPMMU_ADDR_SECTION_2 0x0080000000ULL
#define M3N_IPMMU_ADDR_SECTION_3 0
```

These table entries are defined based on each entry’s DDR physical address in Table 10-3, Table 10-4, Table 10-5 and Table 10-6. Therefore, if you need to change the default PA-VA mapping table, update the table entries in mmngr_private.h with appropriate DDR physical address according to your configurations.

10.3.3 Sample memory map

Below shows H3 sample memory map with the default PA-VA mapping table in Table 10-3 when IPMMU is enabled. Depend on CMA areas' configuration in device tree, the location of CMA areas can be changed from the default memory map.

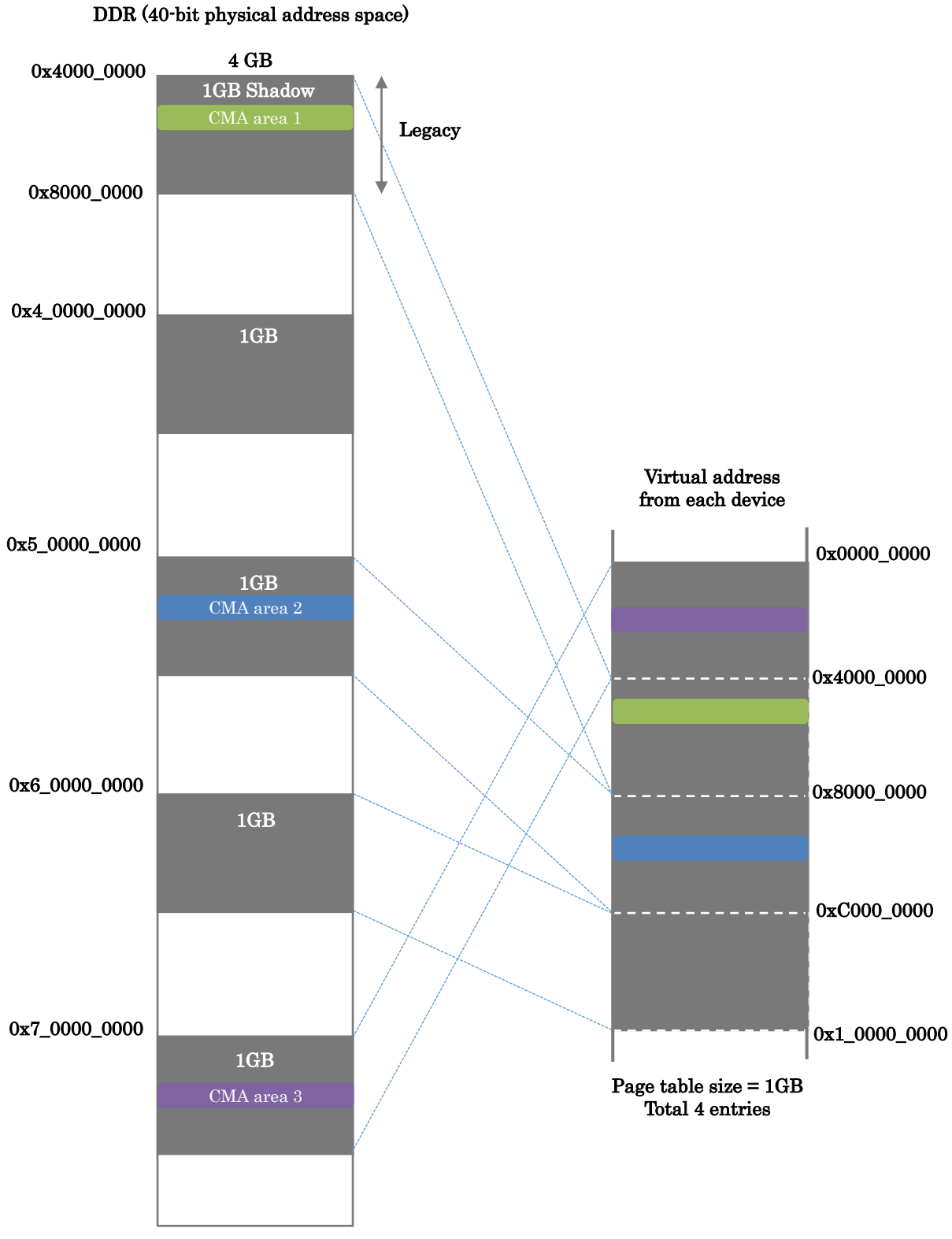

Figure 10-1 Sample memory map (IPMMU enabled)