Novatek HDAL Design Specification - hd_common

Table of Content

1 Introduction

The purpose of “hd_common” is to provide an interface for APP layer to initialize miscellaneous hardware, PCI bus and media memory allocation for engine.

1.1 Media memory allocation for IPC

- The DDR memory is classified into the Linux OS memory and media memory zone (MMZ) memory. The OS memory is managed by the OS, and the MMZ memory is managed by the MMZ kernel module (nvtmpp.ko in code\hdal\drivers\k_flow\source\kflow_common\nvtmpp), and is available only for media services.
- MMZ divides to several memory zones，(DDR1, DDR2 ....)，each DDR divides to several memory pools.
- The memory that got or allocated from MMZ are all continuous.
- The memory range that managed by Linux OS and MMZ are controlled by the file *configs\cfg_IPCAM1_EVB\nvt-na51000-mem-tbl.dtsi*
  - Linux system memory region: The memory region managed by Linux OS
  - cma0: The continuous memory that can be used by hdal drivers.
  - dsp_cma0: The continuous memory that reserved for CEVA DSP1.
  - dsp_cma1: The continuous memory that reserved for CEVA DSP2.
  - cma0，dsp_cma0 and dsp_cma1 these memories are also managed by Linux OS. If dsp are not opened, the dsp_cma0 and dsp_cma1 will also be used by Linux OS, the memory can be allocated to user space process.
    Note: The memory layout /* Linux system memory region*/ <0x00000000 0x12800000> should contains cma0、dsp_cma0、dsp_cma1 these three region.
  - hdal-memory/media: the memory range is MMZ, it contains four values as follows: DDR1 address，DDR1 size，DDR2 address，DDR2 size. If the HW layout don’t have DD2, user can just assign two values: DDR1 address，DDR1 size.

Example:

```
/* Linux system memory region*/
memory { device_type = "memory"; reg = <0x00000000 0x12800000>; };


/* Linux setup reserved memory */
reserved-memory {
      #address-cells = <1>;
```

```
        #size-cells = <1>;
        ranges;

        cma0: cma0@0x05000000 {
                compatible = "shared-dma-pool";
                reusable;
                reg = <0x05000000 0x02800000>;
                alignment = <0x400000>;
                status = "okay";
        };
                dsp_cma0: dsp_cma0@0x04800000 {
                compatible = "shared-dma-pool";
                reusable;
                reg = <0x04800000 0x00800000>;
                alignment = <0x400000>;
                status = "okay";
        };

        dsp_cma1: dsp_cma1@0x04000000 {
                compatible = "shared-dma-pool";
                reusable;
                reg = <0x04000000 0x00800000>;
                alignment = <0x400000>;
                status = "okay";
        };
    hdal-memory {
    #address-cells = <1>;
    #size-cells = <1>;
    media { reg = <0x12800000 0x0D800000 0x40000000 0x20000000>; };
    };
```

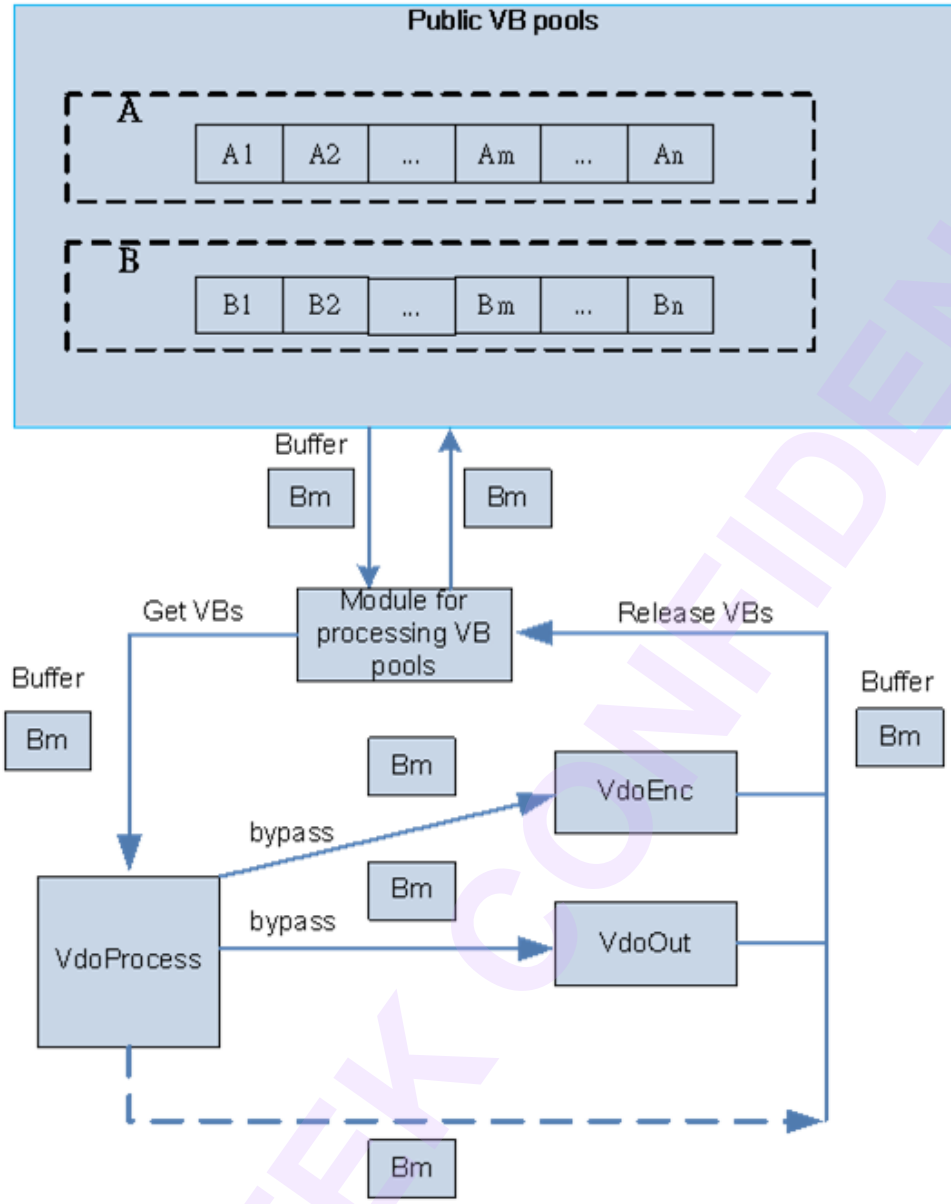
1.1.1 Memory pools

- When the memory pool is initialized, the common pool will be split first, and the rest of the remaining memory will be reserved for the private pool.
- Common pools (also known as Public pools)

- When hd_common_mem_init() is called, user should determines the buffer size/buffer count of all fixed pools, cant not dynamic create/destroy pool
- Arrange multiple buffer pools. Each pool has multiple buffers of the same size. Different pools have different buffer sizes. All modules share these buffers. These buffers are not dedicated used by a certain module.
- When a module wants to get a buffer from the common pools, the search method is: From the smallest free buffer size to the big search, until you find the size that meets the demand.
- The lock/unlock and reference count mechanisms are used to manage the current usage of buffers. When the reference count is 0, the buffer will be released back to the common pools.

- Private pools
  - Support dynamic create and destroy pools.
  - The buffer is dedicated used by a certain module not shared by all modules.
  - Not lock/unlock and reference count mechanisms.
  - The buffer that allocated through hd_common_mem_alloc() API are all considered private pools.

1.1.2 Public pools data flow



1.2 Media memory allocation for NVR

- The DDR memory is classified into the OS memory and reserved memory. The OS memory is managed by the OS, and the others are managed by user for allocating buffers to media services.
- Reserved memory setting could be referred to *BSP/linux-kernel/arch/arm/boot/dts/gm8296.dts.*

Example:

```
…
reserved-memory {
      #address-cells = <1>;
      #size-cells = <1>;
      ranges;
      cma0: cma-0 {
            compatible = "shared-dma-pool";
            reusable;
            size = <0xc800000>; /* must be 8M alignment */
            alignment = <0x800000>; /* must be 8M alignment */
      };
};
…
hdal_base {
      region = <0x13800000 0xc800000>;   /* hdal start:312M size:200MB, check frammap/ddr_info
*/
};
```

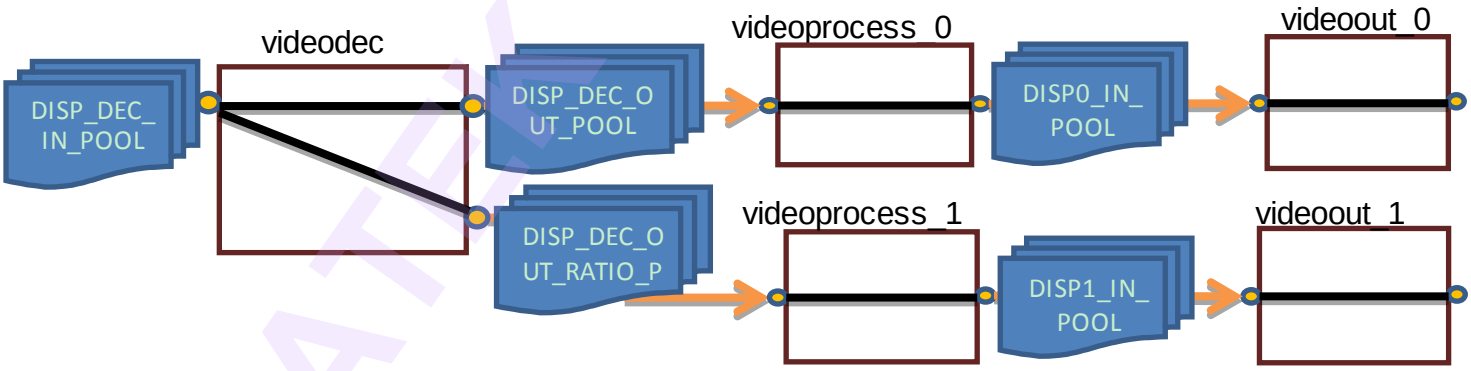
1.2.1 Memory pools

- HDAL requires user to configure memory. The memory pool, referring to HD_COMMON_MEM_POOL_TYPE, is mainly divided into two categories, one for the internal operation of HDAL, and the other for user defined. The name of pool with "DISP" means that the application of the path is related to display. ◦
- HD_COMMON_MEM_DISPx_IN_POOL
  - ☐ They are output buffers of HD_VIDEOPROCESS and dedicated for display buffer, such as LCD and HDMI.
- HD_COMMON_MEM_DISP_DEC_IN_POOL/ HD_COMMON_MEM_DISP_DEC_OUT_POOL/

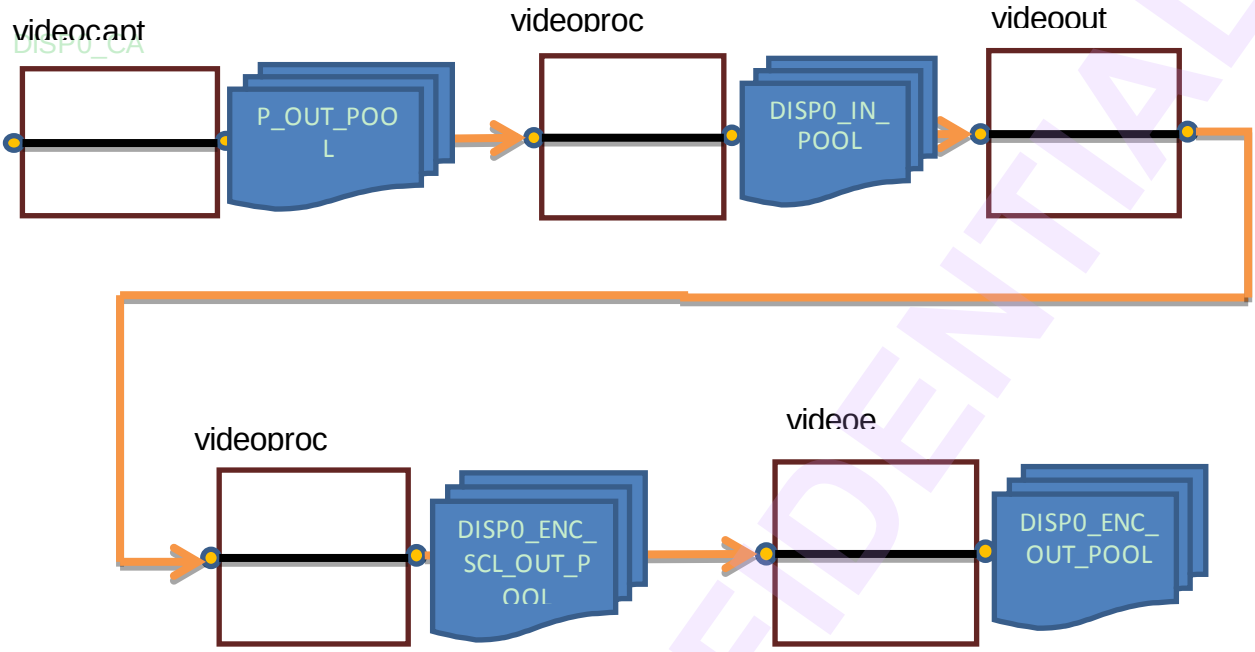
HD_COMMON_MEM_DISP_DEC_OUT_RATIO_POOL

- They are related to HD_VIDEODEC such as DEC_IN_POOL for the stream queue buffer, DEC_OUT_POOL for the output, YUV buffer, of decoder and DEC_OUT_RATIO for the output of scaling down decoding.

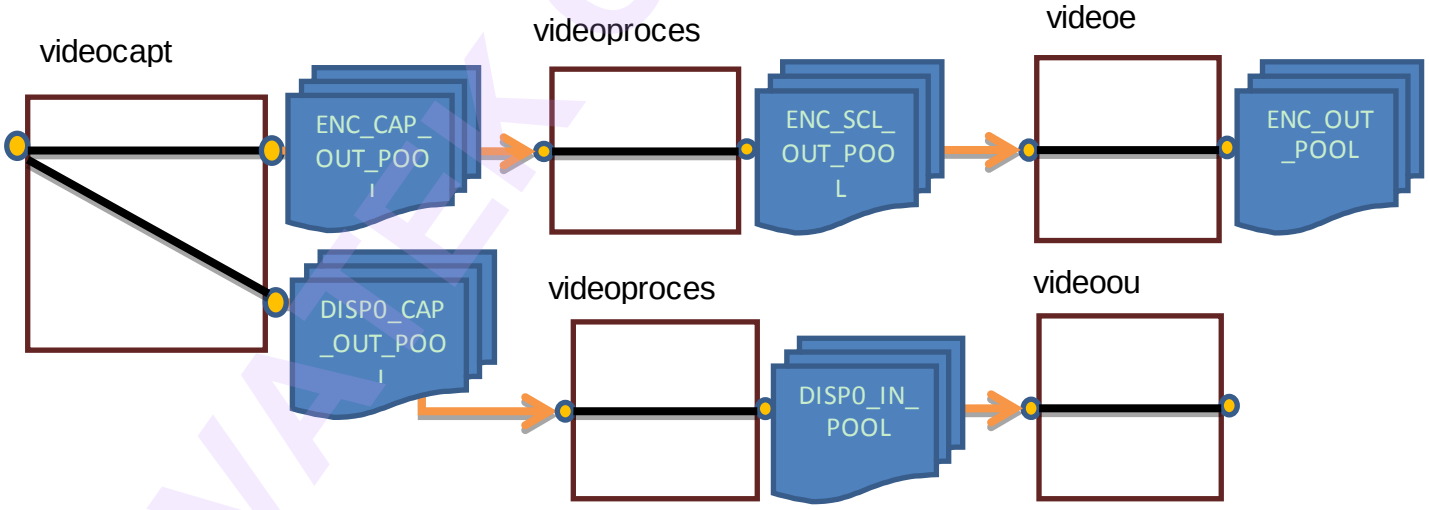
- HD_COMMON_MEM_DISPx_ ENC_SCL_OUT_POOL
  - They are the output buffers of HD_VIDEOPROCESS and usually for display re-encoding.
- HD_COMMON_MEM_DISPx_ ENC_OUT_POOL
  - Output buffer of HD_VIDEOENC and usually for display re-encoding.
- HD_COMMON_MEM_DISPx_ CAP_OUT_POOL/ HD_COMMON_MEM_ ENC_CAP_OUT_POOL
  - All of them are the output buffers of HD_VIDEOCAPTURE, such as DISPx_ CAP_OUT for liveview and ENC_CAP_OUT for recording.
- HD_COMMON_MEM_AU_ENC_AU_GRAP_OUT_POOL
  - For HD_AUDIOCAP and HD_AUDIOENC.
- HD_COMMON_MEM_AU_DEC_AU_RENDER_IN_POOL
  - For HD_AUDIODEC and HD_AUDIOOUT.
- HD_COMMON_MEM_COMMON_POOL
  - Not dedicated for any specified HDAL module, like a temp buffer for HDAL internal usage.



Playback



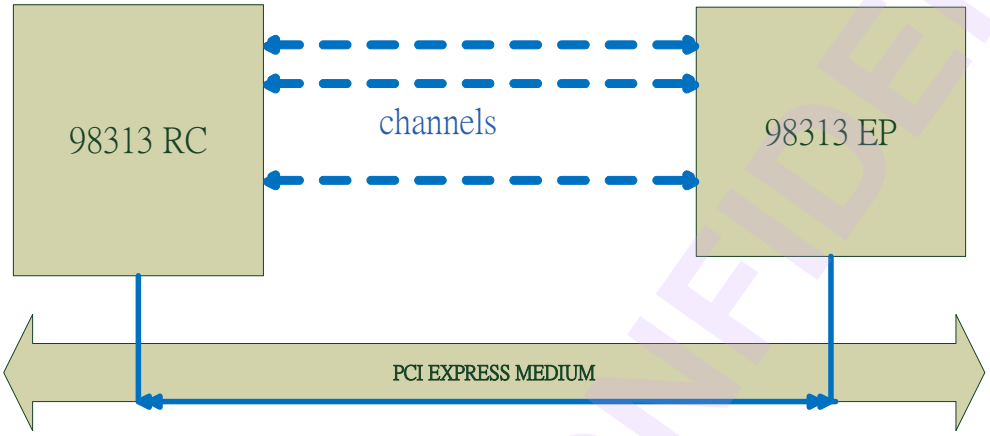
Display Encode



Liveview and Record

1.3 PCIe Communication (for NVR Only)

PCIe-Comm is the abbreviation of PCI Express Communication which is an interrupt-driven software protocol over the PCI Express medium. PCIe-Comm provides a lot of point-to-point software channels. A software channel consists of one sender and one receiver. Sender is the one who transmits the data to the channel. The receiver is the one who receives the data from the channel. Users can communicate each other by means of these channels. The following is a simple architecture.



From above picture, a lot of channels are established between RC and EP. These channels actually use the PCI Express Bus to send/receive the upper application data. The channel supports full-duplex mode which means users can use the channel to send and receive the data simultaneously.

1.3.1 Term Definition

The following terms used throughout the document describe the communication behaviors in the system.

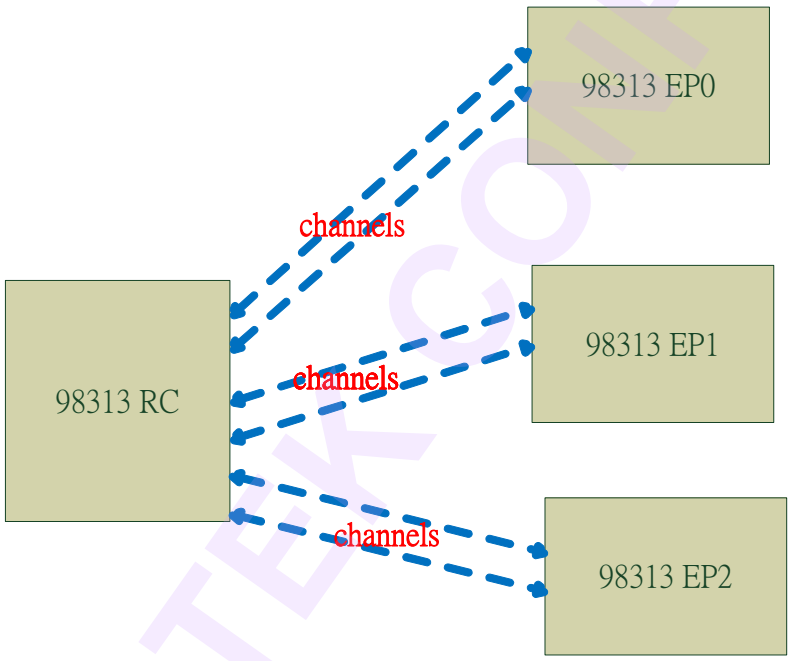
- NT98313: This is the name of the chip.
- PCIe RC: PCIe Root Complex port. It is similar to PCI Host
- PCIe EP: PCIe End Point port. It is similar to PCI Device.
- Channel: It is the virtual concept defined by software. Each channel is a bi-directional point-to-point pipe and is connected between RC and EP. For example, when data are sent through this channel by the sender of RC, the data only can be received by EP0, which is at the other side of this channel. If RC would like send data to EP1 (Even duplicated data), a new channel must be opened between RC and EP1.
- Transfer: It is a transmission to move data from one side to the other side, which is called “one transfer”. A burst means that many transfers are grouped into one transfer.

That is, one transfer carries many data instead of one data.

- Upper application: It indicates that users are sending/receiving data through the channels of the PCIe-Comm module. It can be the kernel drivers or the applications in the user space.

1.3.2 PCIe-Comm function

- Provide pre-established channels between RC and EP for communication.
  - Each channel is a point to point pipe and only RC can communicate with EP or EP communicates with RC.
  - Communication between/among EPs is forbidden. Please see below picture, the channels only exist between RC and EP.
  - Data in the channel is FCFS (First Come First Service)



- Each channel is bi-direction and can operate in full-duplex mode
  - Users can transmit/receive the data to/from the channel simultaneously. Actually there are two sub-channels within a PCIe-Comm channel and they work in independent.
- Provide queuing mechanism:
  - There is a 16 depth of queue for each channel.
  - The upper application will not be blocked except the queue is full.
    - Users can choose blocking mode or non-blocking mode while opening the

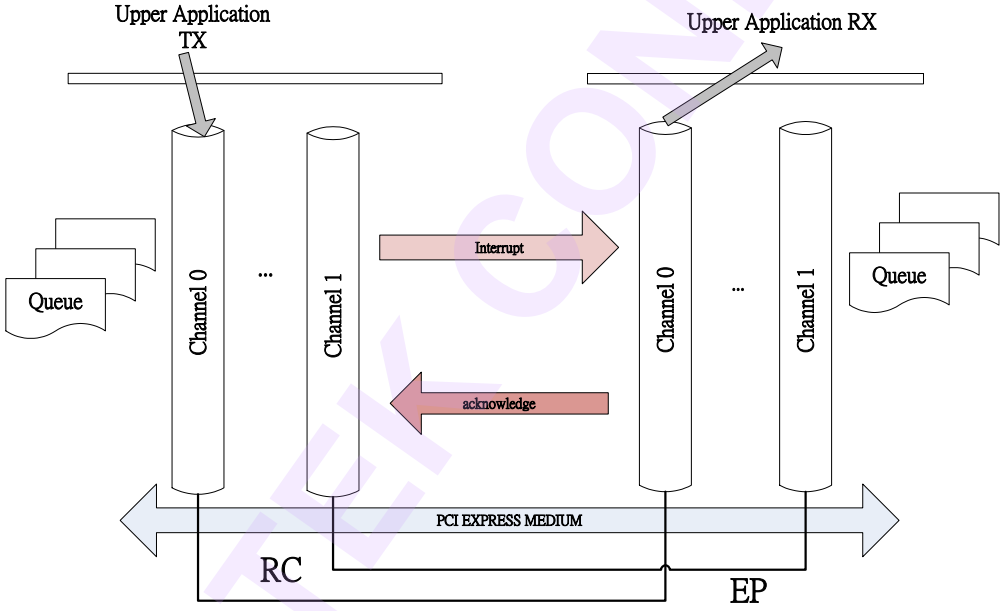
channel.

- ❖ For example: hd_common_pcie_open (name, …, O_NONBLOCK). All parameters will be listed in HDAL API chapter.

- ☐ A flow control is provided to prevent the data from congestion. When the local queue is almost full or the remote not process data in time, PCIe-Comm module will slowly send data or will stop sending data to the peer side when the local queue is completely full.

1.3.3 Data Flow

As mentioned before, each channel works as a pipe and is associated with a queue. The following picture shows basic architecture of the channel. Please be noticed that if the upper application uses channel-x to send data, the receiver in other side must use the same channel-x to receive data.



In above picture, two global interrupt signals are used for data communication. One for data arriving indication interrupt and the other one is for finishing data processing indication interrupt. The latter case is to have the sender release the send buffer.

1.3.4 Maximum Data Length

PCIe-Comm has the limitation of maximum data length for upper application to

send/receive once. If the length of upper application exceeds the limitation of PCIe-Comm, user application must divide them into several sub-datagrams. The default value for maximum read and write is 4MB. Thus if user application would like to transmit 9MB, it must be divided into 3 times transmission. The maximum length can get through the following function call:

```
…
{
   int max_wlen, max_rlen, chan = 2;
HD_COMMON_PCIE_CHIP chip = COMMON_PCIE_CHIP_RC;

if (hd_common_pcie_get(chip, chan, COMMON_PCIE_PARAM_GET_READ_MAX, &max_wlen) != HD_OK)
   return -1;
if (hd_common_pcie_get(chip, chan, COMMON_PCIE_PARAM_GET_WRITE_MAX, &max_rlen) != HD_OK)
   return -1;
   …
}
```

2 Function and data structure definition

2.1 Function definition

2.1.1 Memory functions

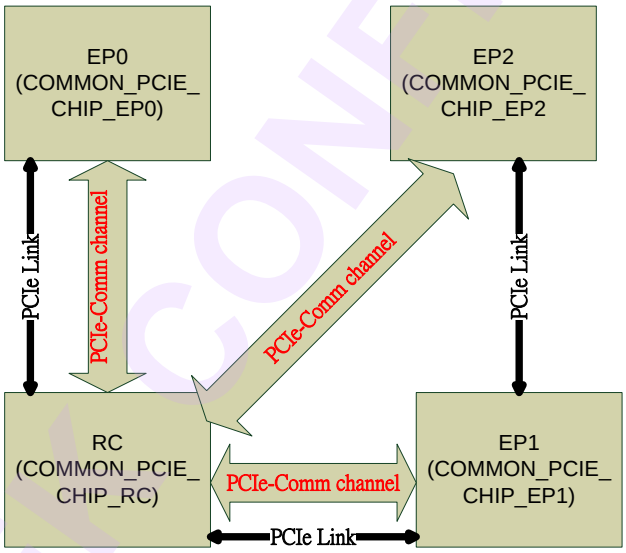
The detail parameters meanings please refer the header file hd_common.h.

Function	Description
hd_common_mem_init(HD_COMMON_MEM_INIT_CONFIG *p_mem_config)	Initialize common memory module and set fixed memory pools layout.
hd_common_mem_calc_buf_size(void *p_hd_data)	Calculate required buffer size for specified data format. (Currently support data formats are HD_VDO_FRAME/HD_VDO_BITSTREAM/HD_AUD_FRAME/HD_AUD_BITSTREAM.)
hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE mem_type, UINT32 phy_addr, UINT32 size)	Map a physical address to user space.
hd_common_mem_munmap(void* virt_addr, unsigned int size);	Un-map a user space memory.
hd_common_mem_flush_cache(void* virt_addr, unsigned int size);	Flush the cache memory data.
hd_common_mem_get_block(HD_COMMON_MEM_POOL_TYPE pool_type, UINT32 blk_size, HD_COMMON_MEM_DDR_ID ddr);	Get one new block from video buffer pools.
hd_common_mem_release_block(HD_COMMON_MEM_VB_BLK blk)	Release the video buffer block.
hd_common_mem_blk2pa(HD_COMMON_MEM_VB_BLK blk)	Translate video buffer block handle to buffer physical address.

hd_common_mem_alloc(CHAR* name, UINT32 *phy_addr, void **virt_addr, UINT32 size, HD_COMMON_MEM_DDR_ID ddr);	Allocates the continuous memory in the user state. (Note: the allocated memory is cacheable.)
hd_common_mem_free(UINT32 phy_addr, void *virt_addr)	Releases the continuous memory in the user state.
hd_common_mem_uninit(void)	Un-initializes common memory module.

2.1.2 PCIe-Comm functions (for NVR Only)

The header is listed in hd_common.h. Before describing the function calls, let's see the below topology.



RC has point-to-point virtual channels to each EP and EP only has virtual channels to RC. Communication between EPs is forbidden. Thus when upper application would like to communicate with the specific EP, it must list the target EP defined in HD_PCICOMM_CHIP. The number of EP in the system is product dependent.

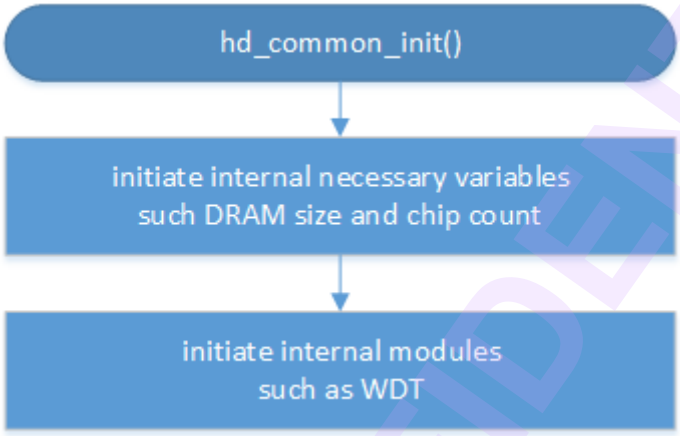
```
typedef enum _HD_COMMON_PCIE_CHIP {
    COMMON_PCIE_CHIP_RC,      ///< target is RC
    COMMON_PCIE_CHIP_EP0,     ///< target is EP0
    COMMON_PCIE_CHIP_EP1,     ///< target is EP1
    COMMON_PCIE_CHIP_EP2,     ///< target is EP2
    COMMON_PCIE_CHIP_EP3,     ///< target is EP3
```

```
    COMMON_PCIE_CHIP_EP4,     ///< target is EP4
    COMMON_PCIE_CHIP_EP5,     ///< target is EP5
    COMMON_PCIE_CHIP_EP6,     ///< target is EP6
    COMMON_PCIE_CHIP_EP7,     ///< target is EP7
    COMMON_PCIE_CHIP_EP8,     ///< target is EP8
    COMMON_PCIE_CHIP_MAX,
    ENUM_DUMMY4WORD(HD_COMMON_PCIE_CHIP)
} HD_COMMON_PCIE_CHIP;
```

Function	Description
hd_common_pcie_init(void)	Pcie-comm init function. This is first entry point
hd_common_pcie_open(char *name, HD_COMMON_PCIE_CHIP chip_id, int chan_id, int mode)	Open a pcie-comm channel. The channel one can be opened once.
hd_common_pcie_close(HD_COMMON_PCIE_CHIP chip_id, int chan_id)	Close a pcie-comm channel
hd_common_pcie_get(HD_COMMON_PCIE_CHIP chip_id, int chan_id, HD_COMMON_PCIE_PARAM_ID id, void *parm)	Used to get pcie-comm internal setting such as max_wlen, max_rlen.
hd_common_pcie_set(HD_COMMON_PCIE_CHIP chip_id, int chan_id, HD_COMMON_PCIE_PARAM_ID id, void *parm)	Used to set the channel characteristic. Such as the buffer is physical contiguous.
hd_common_pcie_recv(HD_COMMON_PCIE_CHIP chip_id, int chan_id, unsigned char *pbuf, unsigned int len)	Used to receive the data from the pcie-comm channel
hd_common_pcie_send(HD_COMMON_PCIE_CHIP chip_id, int chan_id, unsigned char *pbuf, unsigned int len)	Used to send the data into the pcie-comm channel
hd_common_pcie_uninit	Un-init the pcie-comm

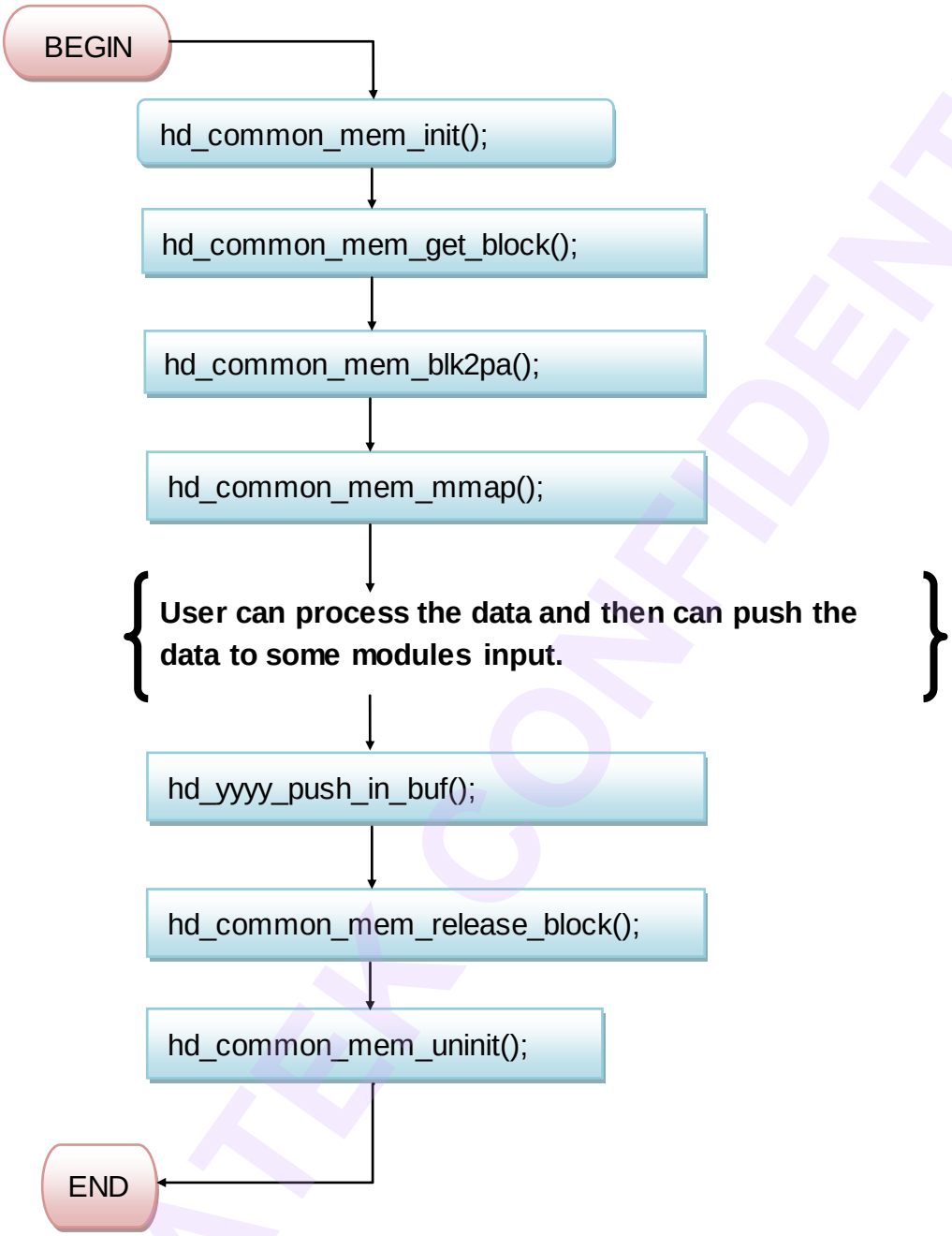
2.2 Software control flow

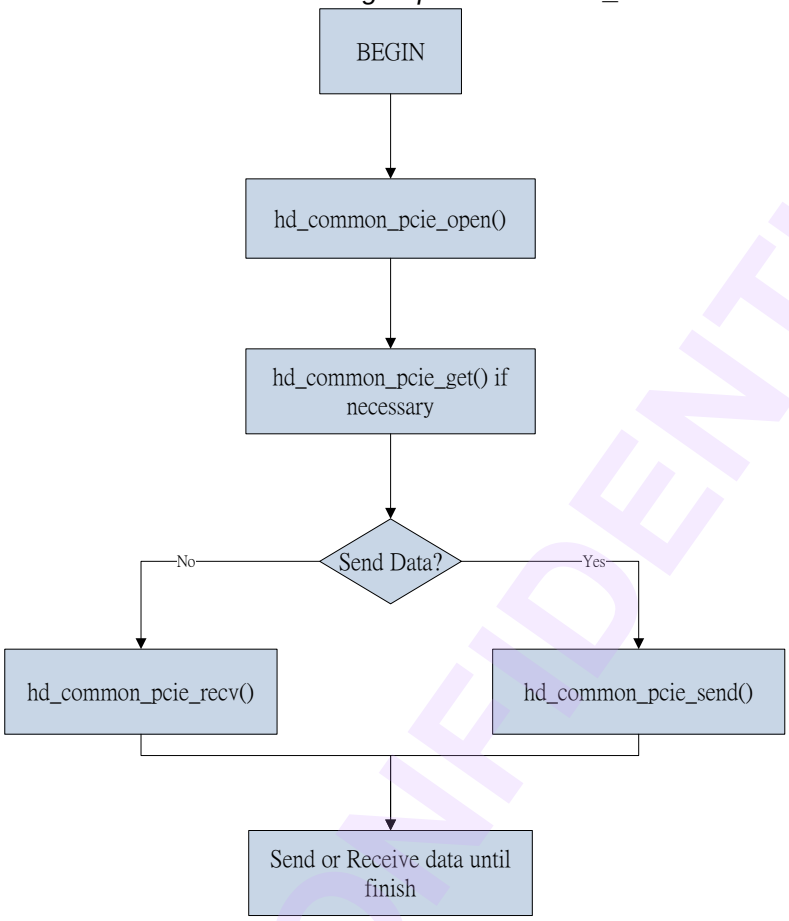
2.2.1 Hd_common_init



hd_common_init initialize miscellaneous hardware and system

2.3 Memory control flow





3 Usage

[Include files]

Files	
file	**hd_common.h**
	Exported header file of hd common module.

3.1 Media memory initialize for IPC

- User App is responsible for setting VB common pools, block size, block count and then call hd_common_mem_init() to initialize the common memory pools layout.
- The buffers that used by video cap out, video process out and video out should be set when initialized common pools, because the buffers to be used are taken from the common pools.
- Video Encode, Audio In/Out, Audio Encode/Decode will generate enough buffer size from private pool when module starts according to the parameters set by user.
- To re-layout the entire video buffer, user can call hd_common_mem_uninit() and then call hd_common_mem_init() again to re-layout buffers.

Example:

```
HD_RESULT            ret;
HD_COMMON_MEM_INIT_CONFIG mem_cfg = {0};


mem_cfg.pool_info[0].type = HD_COMMON_MEM_COMMON_POOL;
mem_cfg.pool_info[0].blk_size = 0x200000;
mem_cfg.pool_info[0].blk_cnt = 3;
mem_cfg.pool_info[0].ddr_id = DDR_ID0;
mem_cfg.pool_info[1].type = HD_COMMON_MEM_COMMON_POOL;
mem_cfg.pool_info[1].blk_size = 0x300000;
mem_cfg.pool_info[1].blk_cnt = 3;
mem_cfg.pool_info[1].ddr_id = DDR_ID0;
mem_cfg.pool_info[2].type = HD_COMMON_MEM_OSG_POOL;
mem_cfg.pool_info[2].blk_size = 0x100000;
mem_cfg.pool_info[2].blk_cnt = 3;
mem_cfg.pool_info[2].ddr_id = DDR_ID0;
mem_cfg.pool_info[3].type = HD_COMMON_MEM_OSG_POOL;
```

```
mem_cfg.pool_info[3].blk_size = 0x200000;
mem_cfg.pool_info[3].blk_cnt = 3;
mem_cfg.pool_info[3].ddr_id = DDR_ID0;
ret = hd_common_mem_init(&mem_cfg);
if (HD_OK != ret) {
     printf("hd_common_mem_init err: %d\r\n", ret);
}
return ret;
```

3.1 Media memory initialize for NVR

- User App should reserve memory for media, referring to mem_init().

Example:

```
HD_COMMON_MEM_POOL_INFO pool_info[] = {
  //ddr_id, type,                                blk_size,              cnt, addr, shared_pool
{ DDR_ID0, HD_COMMON_MEM_COMMON_POOL,           COMMON_SIZE,           1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_ENC_CAP_OUT_POOL,      ENC_CAP_OUT_SIZE,      1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_ENC_OUT_POOL,          ENC_OUT_SIZE,          1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_ENC_SCL_OUT_POOL,      ENC_SCL_OUT_POOL,      1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_ENC_REF_POOL,          ENC_MAIN_REF_SIZE,     8,   0,    {0} },
//4ch x 4 buffers
{ DDR_ID0, HD_COMMON_MEM_ENC_REF_POOL,          ENC_SUB_REF_SIZE,      8,   0,    {0} },
//4ch x 4 buffers
{ DDR_ID0, HD_COMMON_MEM_DISP_DEC_IN_POOL,      DISP_DEC_IN_SIZE,      1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP_DEC_OUT_POOL,     DISP_DEC_OUT_SIZE,     1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP_DEC_OUT_RATIO_POOL, DISP_DEC_OUT_RATIO_SIZE, 1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP0_CAP_OUT_POOL,    DISP0_CAP_OUT_SIZE,    1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP0_IN_POOL,         DISP0_IN_SIZE,         1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP0_FB_POOL,         DISP0_FB_SIZE,         1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP1_CAP_OUT_POOL,    DISP1_CAP_OUT_SIZE,    1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP1_IN_POOL,         DISP1_IN_SIZE,         1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP1_FB_POOL,         DISP1_FB_SIZE,         1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP1_ENC_SCL_OUT_POOL, DISP1_ENC_SCL_OUT_SIZE, 1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_DISP1_ENC_OUT_POOL,    DISP1_ENC_OUT_SIZE,    1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_AU_ENC_AU_GRAB_OUT_POOL, AU_ENC_SIZE,         1,   0,    {0} },
{ DDR_ID0, HD_COMMON_MEM_AU_DEC_AU_RENDER_IN_POOL,AU_DEC_SIZE,         1,   0,    {0} },
```

```
    { DDR_ID0, HD_COMMON_MEM_OSG_POOL,              OSG_SIZE,              1,   0,    {0} },
    { DDR_ID0, HD_COMMON_MEM_MD_POOL,               MD_SIZE,               1,   0,    {0} },
    { DDR_ID0, HD_COMMON_MEM_TMNR_MOTION_POOL,       TMNR_MOTION_SIZE,       1,   0,    {0} },
    { DDR_ID0, HD_COMMON_MEM_CNN_POOL,              CNN_SIZE,              1,   0,    {0} },
};
int assign_pool_addr(void)
{
    unsigned int tar_paddr, tar_total = 0, pool_size = 0;
    int i, fd;
    struct frmmap_hdal_base hdal_mem;

    fd = open("/dev/frammap0", O_RDWR);
    if (fd < 0) {
        printf("Frammap Error: cannot open frammap device.\n");
            exit(0);
    }
    if (ioctl(fd, FRM_GET_DTS_HDAL_BASE, &hdal_mem) < 0) {
            printf("FRM_GET_DTS_HDAL_BASE fail! \n");
            close(fd);
            exit(0);
    }

    printf("hdal_mem 0x%08X, 0x%08X\n", hdal_mem.base, hdal_mem.size);

    tar_paddr = hdal_mem.base;
    for (i = 0; i < sizeof(pool_info)/sizeof(HD_COMMON_MEM_POOL_INFO); i++) {
        pool_info[i].start_addr = tar_paddr;
        pool_info[i].blk_size = ALIGN4096_UP(pool_info[i].blk_size);     //for 4K page handling
        pool_size =  pool_info[i].blk_size * pool_info[i].blk_cnt;
        tar_paddr += pool_size;
        tar_total += pool_size;
        //printf("HDAL pool type %d start 0x%x size %dKB total %dKB\n",
            //pool_info[i].type, tar_paddr, pool_size / 1024, tar_total / 1024);
    }
    if (tar_total > hdal_mem.size) {
        printf("Pool total size overflow %d (hdal size %d)\n", tar_total, hdal_mem.size);
        return -1;
    }
```

```
    return 0;
}
HD_RESULT mem_init(VOID)
{
…
if (assign_pool_addr() < 0)
        return -1;
…
…
memcpy(mem_cfg.pool_info, pool_info, sizeof(pool_info));
   ret = hd_common_mem_init(&mem_cfg);
   if (HD_OK != ret) {
        printf("hd_common_mem_init err: %d\r\n", ret);
        return -1;
   }
…
}
```

3.2 Get/Release block from pools

After initial the memory module, use can use **hd_common_mem_get_block()** API to get memory block from some predefined pools. If user want to access the memory, user need use **hd_common_mem_mmap()** API to map the memory to user space. If user don't want to access the memory anymore, user need to call **hd_common_mem_munmap()** API to unmap the memory and then call **hd_common_mem_release_block**() to return back the block to pools.

Note: User can't get block size exceeds the block size when you initialized.
Note2: In NVR system, **hd_common_mem_get_block**() does NOT support pools from "HD_COMMON_MEM_USER_POOL_BEGIN" to "HD_COMMON_MEM_USER_DEFINIED_POOL" since these pools will not be internal managed.

Example:

```
HD_COMMON_MEM_VB_BLK blk;
UINT32          pa, va;
UINT32          blk_size = 0x100000;
```

```
    HD_COMMON_MEM_DDR_ID ddr_id = DDR_ID0;
    HD_RESULT       ret;

    system("cat /proc/nvtmpp/status");
    blk = hd_common_mem_get_block(HD_COMMON_MEM_OSG_POOL, blk_size, ddr_id);
    if (blk == HD_COMMON_MEM_VB_INVALID_BLK) {
        printf("get block fail\r\n", blk);
        return;
    }
    pa = hd_common_mem_blk2pa(blk);
    if (pa == 0) {
        printf("blk2pa fail, blk = 0x%x\r\n", blk);
        return;
    }
if (pa > 0) {
        va = (UINT32)hd_common_mem_mmap(HD_COMMON_MEM_MEM_TYPE_CACHE, pa, blk_size);
        if (va == 0) {
            goto map_err;
        }
        memset((void*)va, 0x11, blk_size);
        hd_common_mem_munmap((void*)va, blk_size);
    }
map_err:
    system("cat /proc/nvtmpp/status");
    ret = hd_common_mem_release_block(blk);
    if (HD_OK != ret) {
        printf("release blk fail %d\r\n", ret);
        return;
    }
```

3.3 Media memory allocate/free (for IPC only)

After initial the memory module, user can use **hd_common_mem_alloc()** API to allocate continuous memory from media memory area. This API will return the physical address and user space virtual address, so user doesn't need to map the memory to user space again. The allocate memory is classified to private pools, when user use debug command to dump

the memory status or memory layout will see the memory in private pool part.

Example:

```
UINT32              pa, va;
UINT32              size = 0x200000;
HD_COMMON_MEM_DDR_ID ddr_id = DDR_ID0;
HD_RESULT           ret;

ret = hd_common_mem_alloc("osg1", &pa, (void **)&va, size, ddr_id);
if (ret != HD_OK) {
     printf("err:alloc size 0x%x, ddr %d\r\n", size, ddr_id);
     return;
}
printf("pa = 0x%x, va = 0x%x\r\n", pa, va);
memset((void*)va, 0x33, size);
system("cat /proc/nvtmpp/status");
ret = hd_common_mem_free(pa, (void *)va);
if (ret != HD_OK) {
     printf("err:free pa = 0x%x, va = 0x%x\r\n", pa, va);
}
printf("free_mem\r\n\r\n");
system("cat /proc/nvtmpp/status");
```

3.4 Virtual address to physical address (for IPC only)

User can use **hd_common_mem_get(HD_COMMON_MEM_PARAM_VIRT_INFO, xxx)** API to get physical address from user space virtual address.

Example:

```
void                *va1;
UINT32              pa1;
UINT32              size = 0x200000;
UINT32              va1_offset = 0x1010;
HD_COMMON_MEM_DDR_ID ddr_id = DDR_ID0;
HD_RESULT           ret;
HD_COMMON_MEM_VIRT_INFO vir_meminfo = {0};

ret = hd_common_mem_alloc("test1", &pa1, (void **)&va1, size, ddr_id);
```

```
if (ret != HD_OK) {
      printf("err:alloc size 0x%x, ddr %d\r\n", size, ddr_id);
      return HD_ERR_SYS;
}
vir_meminfo.va = va1+va1_offset;
hd_common_mem_get(HD_COMMON_MEM_PARAM_VIRT_INFO, &vir_meminfo);
printf("vir_meminfo.pa = 0x%x \r\n", vir_meminfo.pa);
```

3.5 PCIe-Comm sends data (for NVR only)

The following example shows the flow of the RC sending data to EP0 in channel 2.

Example:

```
HD_COMMON_PCIE_CHIP chip_id = COMMON_PCIE_CHIP_EP0;
int  chan_id = 2, status;
char buff[30] = “Hello, it is a good”;

if (hd_common_pcie_open("Ellie", chip_id, chan_id, O_WRONLY) != HD_OK)
      return -1;
…

status = hd_common_pcie_send(chip_id, chan_id, buff, 30);
if (status < 0) {
      hd_common_pcie_close(chip_id, chan_id);
      return -1;
}
….
hd_common_pcie_close(chip_id, chan_id);
```

3.6 PCIe-Comm receives data (for NVR only)

The following example shows the flow of the EP0 receiving data from RC in channel 2.

Example:

```
HD_COMMON_PCIE_CHIP chip_id = COMMON_PCIE_CHIP_RC;
int  chan_id = 2, status;
char buff[30];

if (hd_common_pcie_open("Anita", chip_id, chan_id, O_RDONLY) != HD_OK)
      return -1;
…
status = hd_common_pcie_recv(chip_id, chan_id, buff, 30);
if (status < 0) {
      hd_common_pcie_close(chip_id, chan_id);
```

```
        return -1;
}
….
hd_common_pcie_close(chip_id, chan_id);
```

4 Debug command

The common module supports two kinds of debug mechanism for user. User can use proc command or debug menu to debug.

4.1 proc command for IPC

4.1.1 cat /proc/hdal/flow

Print out the detailed parameter settings of the HDAL APIs before, but not include push/pull/release APIs.

Example:

```
hd_common_init
       _hd_osg_reset()
hd_common_mem_init:
   mem type(0x00000001) ddr(0) blk_cnt(3) blk_size(0x018cde80)
   mem type(0x00000001) ddr(0) blk_cnt(6) blk_size(0x00bdda00)
   mem type(0x00000001) ddr(0) blk_cnt(6) blk_size(0x002f7800)
   mem type(0x00000001) ddr(0) blk_cnt(3) blk_size(0x002f7800)
   mem type(0x00000071) ddr(0) blk_cnt(1) blk_size(0x00195000)
   mem type(0x00000072) ddr(0) blk_cnt(1) blk_size(0x00065400)
   mem type(0x00000073) ddr(0) blk_cnt(1) blk_size(0x00065400)
hd_videocap_init
hd_videoproc_init
hd_videoenc_init
hd_videoout_init
hd_videocap_open:
   self(0x2000) out(255) in(0) path_id(0x2000ffff)
hd_videocap_set(DRV_CONFIG):
   path_id(0x2000ffff) driver_name(nvt_sen_imx317) if_type(5) shdr_map(1)
   sensor_pinmux(0x220) serial_if_pinmux(0xf04) cmd_if_pinmux(0x10) clk_lane_sel(1)
   sen_2_serial_pin_map[0 1 2 3 -1 -1 -1 -1]
   ccir_msblsb_switch(0) ccir_vd_hd_pin(0)
   vx1_tx241_cko_pin(0) vx1_tx241_cfg_2lane_mode(0)
   vx1_en(0) if_sel(0) ctl_sel(0) tx_type(0)
```

```
    tge_en(0) swap(0) vcap_vd_src(0) vcap_sync_set(0x0)
    optin: en_mask(0x0) sen_map_if(0) if_time_out(0)
hd_videocap_set(CTRL):
    path_id(0x2000ffff) func(0x1b00)
hd_videoproc_open:
    self(0x2400) out(255) in(0) hd_videoproc_set(DEV_CONFIG):
    path_id(0x2400ffff) pipe(0x000000fe) isp_id(0)
    ctrl_max func(0x00060000) ref_path_3dnr(0x00000000)
    in_max dim(3840x2160) pxlfmt(0xf20c0000)
hd_videoproc_set(CTRL):
    path_id(0x2400ffff) func(0x00060000) ref_path_3dnr(0x24000001)
hd_videocap_open:
    self(0x2000) out(1) in(1) path_id(0x20000101)
hd_videoproc_open:
    self(0x2400) out(1) in(1) path_id(0x24000101)
hd_videoenc_open:
    self(0x2200) out(1) in(1) path_id(0x22000101)
hd_videoenc_open:
    self(0x0) out(129) in(1) path_id(0x22000181)
hd_videoenc_open:
    self(0x0) out(129) in(2) path_id(0x22000281)
hd_videoout_open:
    self(0x0) out(129) in(1) hd_videocap_open:
    self(0x2001) out(255) in(0) path_id(0x2001ffff)
…
…
```

4.1.2 cat /proc/hdal/comm/info

The result will show the media memory information by five parts.

1. **VB PB INFO**: the memory zone information of each DDR
2. **COMMON POOL**: the common memory block free/used status
3. **PRIVATE POOL**: the private pools address and size
4. **MEM LAYOUT**: It will show the media memory layout by memory sequence, User can use this to check if have any memory fragment.
5. **ERR STATUS**: It will show the error counter if modules have get block/lock block/unlock

block fail.

The parameters description of "**VB PB INFO, COMMON POOL, PRIVATE POOL**"can reference the below table.

Parameter	Description
Max Count of pools	Maximum number of buffer pools
PoolId	Handle of a public buffer pool
DDR	DDR number
PhyAddr	Physical start address for a public or private buffer pool
VirAddr	Virtual start address for a public or private buffer pool
BlkSize	Size of the buffer in a buffer pool
BlkCnt	Number of buffers in a buffer pool
Free	Number of free buffers in a buffer pool
MinFree	Minimum number of free buffers since the program runs. If the minimum number is 0, some frames may be lost because the buffers are insufficient.
BlkId	Handle of the buffer in a buffer pool.
BlkHdl	The block handle value
BlkAddr	The block buffer start address.
WantSize	The actual demand buffer size when get a new buffer
user/vdoin/vdoprc/vdoenc/ audin/audout/vdoout	Module name The value indicates the number of times that a buffer in the buffer pool is occupied by the current module. 0: not occupied Other values: number of times that a buffer is occupied
PoolName	Private pool name

The parameters description of "**MEM LAYOUT**" can reference the below table.

Parameter	Description
Pool-xx	The pool id
common pool	The pool type is common pool
Phy	The physical memory range [start ~ end]
Vir	The virtual memory range [start ~ end]
Size	The memory size
[free]	The memory is not used.

The parameters description of "**ERR STATUS**" can reference the below table.

Parameter	Description
Modules	Module name such as user/vdoin/vdoprc/vdoenc/vdodec/audin/ audout/vdoout.
Get blk Fail	The value indicates the number of times that a module got a block fail.
Loc blk Fail	The value indicates the number of times that a module locked a block fail.
Unl blk Fail	The value indicates the number of times that a module unlocked a block fail.

Example:

```
-------------------------VB PUB INFO ------------------------------------------------------------
Max Count of pools: 32
DDR1: paddr = 0x12800000, vaddr = 0x94000000, size = 0x0c800000, Free = 0x0ABB1000, MaxFreeBlk =
0x0ABAFFF0
Modules: user vdoin vdoprc vdoenc

------------------------COMMON POOL -----------------------------------------------------------
PoolId    PoolType   DDR    PhyAddr     VirAddr     BlkSize    BlkCnt  Free  MinFree
   0        0x01     1  0x12802000   0x94002000   0x00322060       3    1       0
    BlkId     BlkHdl     BlkAddr    WantSize   user vdoin vdoprc vdoenc
      0   0x94001FC0   0x94002000   0x00302720     0    1     0     0
      2   0x94647FC0   0x94648000   0x00302720     0    1     0     0

PoolId    PoolType   DDR    PhyAddr     VirAddr     BlkSize    BlkCnt  Free  MinFree
   1        0x01     1  0x1316C000   0x9496C000   0x002F7800       3    2       1
    BlkId     BlkHdl     BlkAddr    WantSize   user vdoin vdoprc vdoenc
      0   0x9496BFC0   0x9496C000   0x002F7600     0    0     0     1

------------------------PRIVATE POOL ----------------------------------------------------------
PoolName           DDR    PhyAddr     VirAddr    BlkSize    BlkCnt  Free  MinFree
vdoprc0.ctrl.in_list   1  0x13A55000   0x95255000   0x00000160       1    0       0
vdoprc0.ctrl.work      1  0x13A57000   0x95257000   0x00006000       1    0       0
vdoenc.ctrl.max        1  0x13A5F000   0x9525F000   0x009E75A8       1    0       0
vdoenc.ctrl.raw        1  0x14448000   0x95C48000   0x00000160       1    0       0
vdoenc.ctrl.bs         1  0x1444A000   0x95C4A000   0x00003480       1    0       0
```

```
-------------------------MEM LAYOUT ------------------------------------------------------------
DDR1: Phy[0x12800000~0x1F000000], Vir[0x94000000~0xA0800000] Size=0x0C800000
   Pool-00: Phy[0x12801800~0x1316B800], Vir[0x94001800~0x9496B800] Size=0x0096A000, common pool
   Pool-01: Phy[0x1316B800~0x13A54800], Vir[0x9496B800~0x95254800] Size=0x008E9000, common pool
   Pool-02: Phy[0x13A54800~0x13A56800], Vir[0x95254800~0x95256800] Size=0x00002000, vdoprc_inlist
   Pool-03: Phy[0x13A56800~0x13A5E800], Vir[0x95256800~0x9525E800] Size=0x00008000, vdoprc_work
   Pool-04: Phy[0x13A5E800~0x14447800], Vir[0x9525E800~0x95C47800] Size=0x009E9000, vdoenc_max
   Pool-05: Phy[0x14447800~0x14449800], Vir[0x95C47800~0x95C49800] Size=0x00002000, vdoenc_raw
   Pool-06: Phy[0x14449800~0x1444E800], Vir[0x95C49800~0x95C4E800] Size=0x00005000, vdoenc_bs
   Pool-07: Phy[0x1444E800~0x1EFFF800], Vir[0x95C4E800~0xA07FF800] Size=0x0ABB1000, [free]


-------------------------ERR STATUS ------------------------------------------------------------
Modules:         user    vdoin   vdoprc   vdoenc
Get blk Fail:       0        1        0        0
Loc blk Fail:       0        0        0        0
Unl blk Fail:       0        0        0        0
```

4.1.3 echo nvtmpp showmsg 1 > /proc/hdal/comm/cmd

It will dump the memory info when modules want get a new block from command pool and return fail. User can use this command to fine tune the buffer count of each memory pools and debug if the buffer count usage is reasonable.

Example:

```
nvtmpp_vb_get_block:module 'vdoin' get a block fail, blk_size 0x302720, ddr 1
-------------------------VB PUB INFO ------------------------------------------------------------
Max Count of pools: 32
DDR1: paddr = 0x12800000, vaddr = 0x94000000, size = 0x0c800000, Free = 0x0ABB1000, MaxFreeBlk =
0x0ABAFFF0
Modules: user vdoin vdoprc vdoenc


-------------------------COMMON POOL ------------------------------------------------------------
PoolId    PoolType   DDR     PhyAddr     VirAddr     BlkSize     BlkCnt  Free  MinFree
   0        0x01     1  0x12802000   0x94002000   0x00322060        3     0        0
    BlkId     BlkHdl     BlkAddr    WantSize   user vdoin vdoprc vdoenc
      2   0x94647FC0   0x94648000   0x00302720      0     1      0      0
```

```
      0    0x94001FC0   0x94002000   0x00302720      0     1      0      0
      1    0x94324FC0   0x94325000   0x00302720      0     0      1      0


PoolId     PoolType   DDR     PhyAddr      VirAddr      BlkSize     BlkCnt  Free  MinFree
   1         0x01     1  0x1316C000   0x9496C000   0x002F7800        3     2        0
    BlkId      BlkHdl      BlkAddr     WantSize   user vdoin vdoprc vdoenc
      0    0x9496BFC0   0x9496C000   0x002F7600      0     0      1      0


-------------------------PRIVATE POOL ------------------------------------------------------
PoolName                DDR     PhyAddr      VirAddr    BlkSize     BlkCnt  Free  MinFree
vdoprc0.ctrl.in_list    1  0x13A55000   0x95255000   0x00000160        1     0        0
vdoprc0.ctrl.work       1  0x13A57000   0x95257000   0x00006000        1     0        0
vdoenc.ctrl.max         1  0x13A5F000   0x9525F000   0x009E75A8        1     0        0
vdoenc.ctrl.raw         1  0x14448000   0x95C48000   0x00000160        1     0        0
vdoenc.ctrl.bs          1  0x1444A000   0x95C4A000   0x00003480        1     0        0
```

4.1.4 echo nvtmpp chkmem_corrupt 1 > /proc/hdal/comm/cmd

Proc Command	Description
echo nvtmpp chkmem_corrupt 1 > /proc/hdal/comm/cmd	Enable timing detection hdal heap memory is corrupted, the period is detected in 2 seconds, and the buffer lock/unlock will also be checked.
echo nvtmpp chkmem_corrupt 0 > /proc/hdal/comm/cmd	Disable timing detection hdal heap memory is corrupted.

When the detection is turned on, the check timing will have two: the first is when the buffer is unlocked, the second is the timer timeout. The memory, memory layout and the function backtrace will be displayed when the error occurs. So if the block buffer is corrupted, there will be an error similar to the following:

Example:

```
[  104.912265] _nvtmpp_heap_chk_block:block memory corruption, addr = 0x981ac800, start tag 0x0 !=
0x4250484e
[  104.922001] dump va=981ac800, addr=981ac800 length=00000080 to console:
[  104.928691] 981AC800 : 00000000 981AC800 00990000 8A0000D0  ................
[  104.935729] 981AC810 : 98B3C000 00000000 00000000 00000000  ................
[  104.942766] 981AC820 : 00000000 00000000 00000000 00000000  ................
```

```
[  104.949803] 981AC830 : 00000000 00000000 00000000 00000000  ................
[  104.956839] 981AC840 : 00000000 00000000 00000000 00000000  ................
[  104.963875] 981AC850 : 00000000 00000000 00000000 00000000  ................
[  104.970913] 981AC860 : 00000000 00000000 00000000 00000000  ................
[  104.977951] 981AC870 : 00000000 00000000 00000000 00000000  ................
[  104.984981]
[  104.984981]
[  104.988113]
[  104.988113] -------------------------MEM LAYOUT
------------------------------------------------------------
[  104.999482] DDR1: Phy[0x08000000~0x20000000], Vir[0x8A000000~0xA2000000] Size=0x18000000
[  105.007643]    Pool-00: Phy[0x08001800~0x0CA6C800], Vir[0x8A001800~0x8EA6C800]
Size=0x04A6B000, common pool
[  105.017450]    Pool-01: Phy[0x0CA6C800~0x111A1800], Vir[0x8EA6C800~0x931A1800]
Size=0x04735000, common pool
[  105.027257]    Pool-02: Phy[0x111A1800~0x12372800], Vir[0x931A1800~0x94372800]
Size=0x011D1000, common pool
[  105.037064]    Pool-03: Phy[0x12372800~0x12C5B800], Vir[0x94372800~0x94C5B800]
Size=0x008E9000, common pool
[  105.046872]    Pool-04: Phy[0x12C5B800~0x13446800], Vir[0x94C5B800~0x95446800]
Size=0x007EB000, common pool
[  105.056679]    Pool-05: Phy[0x13446800~0x13642800], Vir[0x95446800~0x95642800]
Size=0x001FC000, common pool
[  105.066486]    Pool-06: Phy[0x13642800~0x1383E800], Vir[0x95642800~0x9583E800]
Size=0x001FC000, common pool
[  105.076294]    Pool-07: Phy[0x1383E800~0x13840800], Vir[0x9583E800~0x95840800]
Size=0x00002000, vdoprc0.ctrl.in_list
[  105.086881]    Pool-08: Phy[0x13840800~0x13C43800], Vir[0x95840800~0x95C43800]
Size=0x00403000, vdoprc0.ctrl.work
[  105.097208]    Pool-09: Phy[0x13C43800~0x161A5800], Vir[0x95C43800~0x981A5800]
Size=0x02562000, vdoenc.ctrl.max
[  105.107362]    Pool-10: Phy[0x161A5800~0x161A7800], Vir[0x981A5800~0x981A7800]
Size=0x00002000, vdoenc.ctrl.raw
[  105.117516]    Pool-11: Phy[0x161A7800~0x161AC800], Vir[0x981A7800~0x981AC800]
Size=0x00005000, vdoenc.ctrl.bs
[  105.127583]    Pool-12: Phy[0x161AC800~0x16B3C800], Vir[0x981AC800~0x98B3C800]
Size=0x00990000, vdoenc.ctrl.max
[  105.137737]    Pool-13: Phy[0x16B3C800~0x16B3E800], Vir[0x98B3C800~0x98B3E800]
```

```
Size=0x00002000, vdoenc.ctrl.raw
[  105.147891]    Pool-14: Phy[0x16B3E800~0x1FFFF800], Vir[0x98B3E800~0xA1FFF800]
Size=0x094C1000, [free]
[  105.157270] _common_unlock:vdoprc0: check_mem_corrupt fail
[  105.162832] CPU: 0 PID: 0 Comm: swapper/0 Tainted: P         O    4.1.0 #1
[  105.169862] Hardware name: Novatek NA51000 platform
[  105.174754] [<800185b8>] (unwind_backtrace) from [<80014538>] (show_stack+0x20/0x24)
[  105.182487] [<80014538>] (show_stack) from [<8044f2cc>] (dump_stack+0x78/0x90)
[  105.189781] [<8044f2cc>] (dump_stack) from [<7e884ccc>] (_common_unlock+0x9c/0xd8 [kflow_isf])
[  105.198487] [<7e884ccc>] (_common_unlock [kflow_isf]) from [<7e880298>]
(_isf_data_unlock+0x4c/0xd8 [kflow_isf])
[  105.208734] [<7e880298>] (_isf_data_unlock [kflow_isf]) from [<7e880734>]
(_isf_base_unit_release+0x7c/0x94 [kflow_isf])
[  105.219707] [<7e880734>] (_isf_base_unit_release [kflow_isf]) from [<7ed84a0c>]
(_vdoprc_iport_dropdata+0x44/0x94 [kflow_videoproc])
[  105.231763] [<7ed84a0c>] (_vdoprc_iport_dropdata [kflow_videoproc]) from [<7ed854c0>]
(_isf_vdoprc_iport_do_proc_cb+0x15c/0x170 [kflow_videoproc])
[  105.245015] [<7ed854c0>] (_isf_vdoprc_iport_do_proc_cb [kflow_videoproc]) from [<7ed88704>]
(_vdoprc0_input_cb+0x34/0x44 [kflow_videoproc])
[  105.257683] [<7ed88704>] (_vdoprc0_input_cb [kflow_videoproc]) from [<7ed5181c>]
(ctl_ipp_inbuf_fp_wrapper.constprop.21+0x7c/0xac [kflow_ctl_ipp])
[  105.270966] [<7ed5181c>] (ctl_ipp_inbuf_fp_wrapper.constprop.21 [kflow_ctl_ipp]) from
[<7ed52018>] (ctl_ipp_proc_end_cb+0x150/0x19c [kflow_ctl_ipp])
[  105.284389] [<7ed52018>] (ctl_ipp_proc_end_cb [kflow_ctl_ipp]) from [<7ed43b2c>]
(kdf_ipp_cb_proc+0x38/0x44 [kflow_ctl_ipp])
[  105.295711] [<7ed43b2c>] (kdf_ipp_cb_proc [kflow_ctl_ipp]) from [<7ed476ec>]
(kdf_ipp_chk_proc_end+0x14c/0x1c8 [kflow_ctl_ipp])
[  105.307299] [<7ed476ec>] (kdf_ipp_chk_proc_end [kflow_ctl_ipp]) from [<7ed47b3c>]
(kdf_ipp_ime_isr_cb+0x104/0x118 [kflow_ctl_ipp])
[  105.319298] [<7ed47b3c>] (kdf_ipp_ime_isr_cb [kflow_ctl_ipp]) from [<7e9a2084>]
(kdrv_ime_isr_cb+0x60/0x68 [kdrv_ime])
[  105.330400] [<7e9a2084>] (kdrv_ime_isr_cb [kdrv_ime]) from [<7e9a1b8c>] (ime_isr+0x78/0x80
[kdrv_ime])
[  105.340063] [<7e9a1b8c>] (ime_isr [kdrv_ime]) from [<7e983aa8>] (nvt_ime_drv_isr+0x44/0x4c
[kdrv_ime])
[  105.349523] [<7e983aa8>] (nvt_ime_drv_isr [kdrv_ime]) from [<80074710>]
(__irq_action_handler+0x24/0x28)
[  105.358994] [<80074710>] (__irq_action_handler) from [<80455cec>]
```

```
(kretprobe_trampoline+0x0/0x18)
```

4.1.5 cat /proc/hdal/comm/task

The result will show the media task information by three parts.

1. **TASK**: All the tasks created by media modules.
2. **FALG**: All the flags created by media modules.
3. **SEM**: All the semaphores created by media modules and the semaphores are occupied by tasks.

The parameters description of "**TASK**"can reference the below table.

Parameter	Description
Task	The task id and task name.
N-Pri	The task normal priority, also named static priority. The value range is from 0 ~ 139, 0 is highest.
Rt-Pri	The task real time priority. The value range is from 0 ~ 99, 99 is highest.
Sched-Policy	The task scheduling policy. 0: SCHED_NORMAL. A normal, time-shared process. 1: SCHED_FIFO. A First-In, First-Out real-time process. 2: SCHED_RR. A Round Robin real-time process.
Status	The task status: 1. Running: The task is on the running state. 2. Sleep-Interruptible: The task is waiting some condition and blocked. The task can be wakeup by signal or the waiting condition comes true. 3. Sleep-Un-Interruptible: The task is waiting some condition and blocked. The task can't wakeup by signal and must wait until the waiting condition comes true. 4. STOPPED: The task is stopped. 5. TRACED: The task is traced by debugger. 6. DEAD: The task is normal terminated. 7. ZOMBIE: The task is not normal terminated.
Wait for Flag	The task is in sleep status and waiting for some flag pattern.
Wait for Semaphore	The task is in sleep status and waiting for semaphore.
Pattern	The flag pattern to wait for.

The parameters description of "**FALG**"can reference the below table.

Parameter	Description
Flag	The flag id and flag name.
Flags:	The current flag pattern value.
Waiting Task Queue	The tasks waiting queue that waiting for this flag.
Pattern	The task wait flag bits pattern.
Mode	The wait mode is AND bits, OR bits. CLEAR means if want to clear flag bits after waiting condition comes true.

The parameters description of "**SEM**"can reference the below table.

Parameter	Description
Semaphore	The semaphore id and semaphore name.
Max	The initial semaphore count
Cur	The current semaphore count
Owner	The semaphore is occupied by which task.

Example:

```
-------------------------TASK -----------------------------------------------------


Task[04, ctl_sie_buf_tsk] ->  N-Pri: 02, Rt-Pri: 97, Sched-Policy: 2
Status : [Sleep-Interruptible]
Wait for Flag [043,    g_ctl_sie_buf_flg_id], Pattern: 0x00000100 AND
[<803ab200>] (__schedule) from [<803ab3ac>] (schedule+0x98/0xb0)
[<803ab3ac>] (schedule) from [<7e80db04>] (wai_flg+0x260/0x2b0 [kwrap])
[<7e80db04>] (wai_flg [kwrap]) from [<7eb785e4>] (ctl_sie_buf_msg_rcv+0x88/0x1d0 [kflow_ctl_sie])
[<7eb785e4>] (ctl_sie_buf_msg_rcv [kflow_ctl_sie]) from [<7eb78e2c>] (ctl_sie_buf_tsk+0x9c/0x208
[kflow_ctl_sie])
[<7eb78e2c>] (ctl_sie_buf_tsk [kflow_ctl_sie]) from [<800469c0>] (kthread+0xd8/0xe8)
[<800469c0>] (kthread) from [<8000f2d8>] (ret_from_fork+0x14/0x3c)


Task[05, ctl_sie_isp_tsk] ->  N-Pri: 03, Rt-Pri: 96, Sched-Policy: 2
Status : [Running]
Wait for Flag [052,    g_ctl_sie_isp_flg_id], Pattern: 0x00000100 AND
[<803ab200>] (__schedule) from [<803ab3ac>] (schedule+0x98/0xb0)
[<803ab3ac>] (schedule) from [<7e80db04>] (wai_flg+0x260/0x2b0 [kwrap])
[<7e80db04>] (wai_flg [kwrap]) from [<7eb7bf84>] (ctl_sie_isp_msg_rcv+0x88/0x1c0 [kflow_ctl_sie])
```

```
[<7eb7bf84>] (ctl_sie_isp_msg_rcv [kflow_ctl_sie]) from [<7eb7c3ac>] (ctl_sie_isp_tsk+0xa4/0x1e8
[kflow_ctl_sie])
[<7eb7c3ac>] (ctl_sie_isp_tsk [kflow_ctl_sie]) from [<800469c0>] (kthread+0xd8/0xe8)
[<800469c0>] (kthread) from [<8000f2d8>] (ret_from_fork+0x14/0x3c)

Task[06, kdf_ipp_tsk] ->  N-Pri: 03, Rt-Pri: 96, Sched-Policy: 2
Status : [Sleep-Interruptible]
Wait for Flag [071,       g_kdf_ipp_flg_id], Pattern: 0x00040000 AND
[<803ab200>] (__schedule) from [<803ab3ac>] (schedule+0x98/0xb0)
[<803ab3ac>] (schedule) from [<7e80db04>] (wai_flg+0x260/0x2b0 [kwrap])
[<7e80db04>] (wai_flg [kwrap]) from [<7ecdc3d4>] (kdf_ipp_msg_rcv+0x8c/0x198 [kflow_ctl_ipp])
[<7ecdc3d4>] (kdf_ipp_msg_rcv [kflow_ctl_ipp]) from [<7ece06e4>] (kdf_ipp_tsk+0xb0/0x208
[kflow_ctl_ipp])
[<7ece06e4>] (kdf_ipp_tsk [kflow_ctl_ipp]) from [<800469c0>] (kthread+0xd8/0xe8)
[<800469c0>] (kthread) from [<8000f2d8>] (ret_from_fork+0x14/0x3c)

Task[07, ctl_ipp_buf_tsk] ->  N-Pri: 03, Rt-Pri: 96, Sched-Policy: 2
Status : [Running]
Wait for Flag [075,    g_ctl_ipp_buf_flg_id], Pattern: 0x00000400 AND
[<803ab200>] (__schedule) from [<803ab3ac>] (schedule+0x98/0xb0)
[<803ab3ac>] (schedule) from [<7e80db04>] (wai_flg+0x260/0x2b0 [kwrap])
[<7e80db04>] (wai_flg [kwrap]) from [<7ece3b44>] (ctl_ipp_buf_msg_rcv+0x8c/0x188 [kflow_ctl_ipp])
[<7ece3b44>] (ctl_ipp_buf_msg_rcv [kflow_ctl_ipp]) from [<7ece432c>] (ctl_ipp_buf_tsk+0x9c/0x1d0
[kflow_ctl_ipp])
[<7ece432c>] (ctl_ipp_buf_tsk [kflow_ctl_ipp]) from [<800469c0>] (kthread+0xd8/0xe8)
[<800469c0>] (kthread) from [<8000f2d8>] (ret_from_fork+0x14/0x3c)

Task[08, ctl_ipp_ise_tsk] ->  N-Pri: 03, Rt-Pri: 96, Sched-Policy: 2
Status : [Sleep-Interruptible]
Wait for Flag [076,    g_ctl_ipp_ise_flg_id], Pattern: 0x00000400 AND
[<803ab200>] (__schedule) from [<803ab3ac>] (schedule+0x98/0xb0)
[<803ab3ac>] (schedule) from [<7e80db04>] (wai_flg+0x260/0x2b0 [kwrap])
[<7e80db04>] (wai_flg [kwrap]) from [<7ecea650>] (ctl_ipp_ise_msg_rcv+0xb0/0x1a4 [kflow_ctl_ipp])
[<7ecea650>] (ctl_ipp_ise_msg_rcv [kflow_ctl_ipp]) from [<7eceb374>] (ctl_ipp_ise_tsk+0x9c/0x1d4
[kflow_ctl_ipp])
[<7eceb374>] (ctl_ipp_ise_tsk [kflow_ctl_ipp]) from [<800469c0>] (kthread+0xd8/0xe8)
[<800469c0>] (kthread) from [<8000f2d8>] (ret_from_fork+0x14/0x3c)
```

```
Task[09, ctl_ipp_isp_tsk] ->  N-Pri: 03, Rt-Pri: 96, Sched-Policy: 2
Status : [Running]
Wait for Flag [077,   g_ctl_ipp_isp_flg_id], Pattern: 0x00000400 AND
[<803ab200>] (__schedule) from [<803ab3ac>] (schedule+0x98/0xb0)
[<803ab3ac>] (schedule) from [<7e80db04>] (wai_flg+0x260/0x2b0 [kwrap])
[<7e80db04>] (wai_flg [kwrap]) from [<7ece16f4>] (ctl_ipp_isp_msg_rcv+0x98/0x18c [kflow_ctl_ipp])
[<7ece16f4>] (ctl_ipp_isp_msg_rcv [kflow_ctl_ipp]) from [<7ece18f8>] (ctl_ipp_isp_tsk+0x9c/0x1c8
[kflow_ctl_ipp])
[<7ece18f8>] (ctl_ipp_isp_tsk [kflow_ctl_ipp]) from [<800469c0>] (kthread+0xd8/0xe8)
[<800469c0>] (kthread) from [<8000f2d8>] (ret_from_fork+0x14/0x3c)

Task[10, ctl_ipp_tsk] ->  N-Pri: 03, Rt-Pri: 96, Sched-Policy: 2
Status : [Running]
Wait for Flag [074,      g_ctl_ipp_flg_id], Pattern: 0x00000400 AND
[<803ab200>] (__schedule) from [<803ab3ac>] (schedule+0x98/0xb0)
[<803ab3ac>] (schedule) from [<7e80db04>] (wai_flg+0x260/0x2b0 [kwrap])
[<7e80db04>] (wai_flg [kwrap]) from [<7ece57f4>] (ctl_ipp_msg_rcv+0x8c/0x198 [kflow_ctl_ipp])
[<7ece57f4>] (ctl_ipp_msg_rcv [kflow_ctl_ipp]) from [<7ece9a5c>] (ctl_ipp_tsk+0xb0/0x200
[kflow_ctl_ipp])
[<7ece9a5c>] (ctl_ipp_tsk [kflow_ctl_ipp]) from [<800469c0>] (kthread+0xd8/0xe8)
[<800469c0>] (kthread) from [<8000f2d8>] (ret_from_fork+0x14/0x3c)

Task[11, ae_tsk] ->  N-Pri: 120, Rt-Pri: 00, Sched-Policy: 0
Status : [Sleep-Interruptible]
Wait for Flag [091,  ae_task_obj[i].flag_id], Pattern: 0x0000001F OR
[<803ab200>] (__schedule) from [<803ab3ac>] (schedule+0x98/0xb0)
[<803ab3ac>] (schedule) from [<7e80d9f4>] (wai_flg+0x150/0x2b0 [kwrap])
[<7e80d9f4>] (wai_flg [kwrap]) from [<7f453bac>] (ae_tsk+0x84/0x15c [nvt_ae])
[<7f453bac>] (ae_tsk [nvt_ae]) from [<800469c0>] (kthread+0xd8/0xe8)
[<800469c0>] (kthread) from [<8000f2d8>] (ret_from_fork+0x14/0x3c)

Task[40, NMR_VdoTrig_D2D] ->  N-Pri: 03, Rt-Pri: 96, Sched-Policy: 2
Status : [Sleep-Interruptible]
Wait for Flag [082,     FLG_ID_NMR_VDOTRIG], Pattern: 0x0000000E OR
[<803ab200>] (__schedule) from [<803ab3ac>] (schedule+0x98/0xb0)
[<803ab3ac>] (schedule) from [<7e80d9f4>] (wai_flg+0x150/0x2b0 [kwrap])
[<7e80d9f4>] (wai_flg [kwrap]) from [<7efb1f54>] (NMR_VdoTrig_D2DTsk+0x78/0x3c0 [kflow_videoenc])
[<7efb1f54>] (NMR_VdoTrig_D2DTsk [kflow_videoenc]) from [<800469c0>] (kthread+0xd8/0xe8)
```

```
[<800469c0>] (kthread) from [<8000f2d8>] (ret_from_fork+0x14/0x3c)


 Max task_count = 100 , used = 39


-------------------------FLAG ------------------------------------------------------
Flag[002,                g_flag] ->   Flags: 0x00000000
Flag[042,             FLG_ID_TGE] ->   Flags: 0x00000000
Flag[043,      ctl_sen_flag_id[i]] ->   Flags: 0x00010101
Flag[044,    g_ctl_sie_buf_flg_id] ->   Flags: 0x00000008
Waiting Task Queue :
  Task[005, ctl_sie_buf_tsk]
          Pattern: 0x00000100, Mode = AND CLEAR
Flag[053,    g_ctl_sie_isp_flg_id] ->   Flags: 0x00000708
Waiting Task Queue :
  Task[006, ctl_sie_isp_tsk]
          Pattern: 0x00000100, Mode = AND CLEAR
Flag[054,     sie_event_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[055,     sie_event_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[056,             FLG_ID_DCE] ->   Flags: 0x00000002
Flag[057,     kdrv_dce_flag_id[i]] ->   Flags: 0x5000FFFF
Flag[058,             FLG_ID_IFE] ->   Flags: 0x00000001
Flag[059,     kdrv_ife_flag_id[i]] ->   Flags: 0x5000FFFF
Flag[060,            FLG_ID_IFE2] ->   Flags: 0x00000000
Flag[061,    kdrv_ife2_flag_id[i]] ->   Flags: 0x4000FFFF
Flag[062,             FLG_ID_IPE] ->   Flags: 0x00000007
Flag[063,     kdrv_ipe_flag_id[i]] ->   Flags: 0x5000FFFF
Flag[064,             FLG_ID_RHE] ->   Flags: 0x00000400
Flag[065,     kdrv_rhe_flag_id[i]] ->   Flags: 0x5000FFFF
Flag[066,     ipp_event_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[067,     ipp_event_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[068,     ipp_event_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[069,     ipp_event_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[070,     ipp_event_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[071,     ipp_event_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[072,        g_kdf_ipp_flg_id] ->   Flags: 0x0003F208
Waiting Task Queue :
  Task[007, kdf_ipp_tsk]
          Pattern: 0x00040000, Mode = AND CLEAR
```

```
Flag[073,  g_kdf_ipp_hdl_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[074,  g_kdf_ipp_hdl_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[075,        g_ctl_ipp_flg_id] ->   Flags: 0x00000608
Waiting Task Queue :
  Task[011, ctl_ipp_tsk]
          Pattern: 0x00000400, Mode = AND CLEAR
Flag[076,    g_ctl_ipp_buf_flg_id] ->   Flags: 0x00000408
Waiting Task Queue :
  Task[008, ctl_ipp_buf_tsk]
          Pattern: 0x00000400, Mode = AND CLEAR
Flag[077,    g_ctl_ipp_ise_flg_id] ->   Flags: 0x00000208
Waiting Task Queue :
  Task[009, ctl_ipp_ise_tsk]
          Pattern: 0x00000400, Mode = AND CLEAR
Flag[078,    g_ctl_ipp_isp_flg_id] ->   Flags: 0x00001E08
Waiting Task Queue :
  Task[010, ctl_ipp_isp_tsk]
          Pattern: 0x00000400, Mode = AND CLEAR
Flag[079,  g_ctl_ipp_hdl_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[080,  g_ctl_ipp_hdl_flg_id[i]] ->   Flags: 0xFFFFFFFF
Flag[081,            FLG_ID_JPEG] ->   Flags: 0x00000000
Flag[082,            FLG_ID_H26X] ->   Flags: 0x00000000
Flag[083,      FLG_ID_NMR_VDOTRIG] ->   Flags: 0x00000001
Waiting Task Queue :
  Task[041, NMR_VdoTrig_D2D]
          Pattern: 0x0000000E, Mode = OR CLEAR
Flag[084,       FLG_ID_NMP_VDODEC] ->   Flags: 0x00000000
Flag[085,     FLG_ID_WAVSTUD_PLAY] ->   Flags: 0x00000000
Flag[086,   FLG_ID_WAVSTUD_RECORD] ->   Flags: 0x00000000
Flag[087, FLG_ID_WAVSTUD_RECORDWRITE] ->   Flags: 0x00000000
Flag[088, FLG_ID_WAVSTUD_RECORDUPDATE] ->   Flags: 0x00000000
Flag[089,    FLG_ID_WAVSTUD_PLAY2] ->   Flags: 0x00000000
Flag[090,       FLG_ID_NMR_AUDENC] ->   Flags: 0x00000000
Flag[091,       FLG_ID_NMP_AUDDEC] ->   Flags: 0x00000000

Flag[123,      kdrv_ai_flag_id[i]] ->   Flags: 0x000004FF

 Max flag_cnt = 300 , used = 122
```

```
-------------------------SEM ------------------------------------------------------
Semaphore[028,      NVTMPP_PROC_SEM_ID] -> * (Max: 01, Cur: 00), Owner: cat
Semaphore[041,              SEMID_IME] -> * (Max: 01, Cur: 00), Owner: hd_video_record
Semaphore[066,              SEMID_SIE] -> * (Max: 01, Cur: 00), Owner: hd_video_record
Semaphore[108,              SEMID_DCE] -> * (Max: 01, Cur: 00), Owner: hd_video_record
Semaphore[110,              SEMID_IFE] -> * (Max: 01, Cur: 00), Owner: hd_video_record
Semaphore[112,             SEMID_IFE2] -> * (Max: 01, Cur: 00), Owner: hd_video_record
Semaphore[114,              SEMID_IPE] -> * (Max: 01, Cur: 00), Owner: hd_video_record
Semaphore[116,              SEMID_RHE] -> * (Max: 01, Cur: 00), Owner: hd_video_record
Semaphore[635, ISF_VDOENC_PULLQ_SEM_ID[i]] -> * (Max: 00, Cur: 00), Owner: hd_video_record
Semaphore[837,                ipc_sem] -> * (Max: 01, Cur: 00), Owner: ipc_sem_thread

 Max sem_cnt = 999 , used = 844
```

4.1.6 cat /proc/hdal/comm/sem

The result will show the all the semaphores created by media modules.

Example:

```
-------------------------SEM ------------------------------------------------------
Semaphore[01,           v_pwm_sem[i]] -> (Max: 01, Cur: 01), Owner: None
Semaphore[02,           v_pwm_sem[i]] -> (Max: 01, Cur: 01), Owner: None
Semaphore[03,           v_pwm_sem[i]] -> (Max: 01, Cur: 01), Owner: None
Semaphore[04,           v_pwm_sem[i]] -> (Max: 01, Cur: 01), Owner: None
Semaphore[05,           v_pwm_sem[i]] -> (Max: 01, Cur: 01), Owner: None
.... ignored
Semaphore[837,                ipc_sem] -> * (Max: 01, Cur: 00), Owner: ipc_sem_thread
Semaphore[838,              SEMID_IVE] -> (Max: 01, Cur: 01), Owner: None
Semaphore[839, kdrv_ive_semtbl[i].semphore_id] -> (Max: 01, Cur: 01), Owner: None
Semaphore[840,              SEMID_SDE] -> (Max: 01, Cur: 01), Owner: None
Semaphore[841,         SEMID_KDRV_SDE] -> (Max: 01, Cur: 01), Owner: None
Semaphore[842,              SEMID_CNN] -> (Max: 01, Cur: 01), Owner: None
Semaphore[843,              SEMID_SVM] -> (Max: 01, Cur: 01), Owner: None
Semaphore[844, kdrv_ai_semtbl[i].semphore_id] -> (Max: 01, Cur: 01), Owner: None
```

```
Max sem_cnt = 999 , used = 844
```

4.1.7 cat /proc/nvt_drv_sys/dram1_info

The dram1_info will show the current dram1 DMA utilization.

Note: This utilization is the average DMA utilization in the last 25 ms。

Example:

```
utilization: 4
```

Proc Command	Description
echo "w cfg 1000" > /proc/nvt_drv_sys/dram1_info	Set the detection time to every 1000 ms. 1000 is just an example, user can set any value but not less than 25.
echo "w start" > /proc/nvt_drv_sys/dram1_info	Start timing detection. At this time, uart will show the DMA utilization of Dram1 every second, and its value is the average value in one second.
echo "w stop" > /proc/nvt_drv_sys/dram1_info	Stop timing detection.

Example:

```
[ 1148.653267] dram1: 4
[ 1149.653305] dram1: 4
```

4.1.8 cat /proc/nvt_drv_sys/dram2_info

The dram2_info will show the current dram2 DMA utilization.

Note: This utilization is the average DMA utilization in the last 25 ms。

Example:

```
utilization: 6
```

Proc Command	Description
echo "w cfg 1000" > /proc/nvt_drv_sys/dram2_info	Set the detection time to every 1000 ms. 1000 is just an example, user can set any value but not less than 25.

echo "w start" > /proc/nvt_drv_sys/dram2_info	Start timing detection. At this time, uart will show the DMA utilization of Dram1 every second, and its value is the average value in one second.
echo "w stop" > /proc/nvt_drv_sys/dram2_info	Stop timing detection.

Example:

```
[ 1148.653267] dram2: 6
[ 1149.653305] dram2: 6
```

4.1.9 echo nvtmpp dumpstack $pid > /proc/hdal/comm/cmd

Dump the task stack backtrace by input the pid. User can firstly use command “ps - T” to get the pid of one task or process. If user got the pid 734 then can use “echo nvtmpp dumpstack 734 > /proc/hdal/comm/cmd” to dump the stack backtrace. The command is very usefully to help user check the user task or process hangs for waiting something.

Example:

```
root@NVTEVM:~$ ps -T
PID   USER     TIME   COMMAND
    1 root      0:01 {linuxrc} init
    2 root      0:00 [kthreadd]
    3 root      0:00 [ksoftirqd/0]
    4 root      0:00 [kworker/0:0]
    …
  694 root      0:00 telnetd
  699 root      0:00 -sh
  734 root      0:01 hd_demo1 0 0 2
  735 root      0:00 hd_demo1 0 0 2
  744 root      0:45 hd_demo1 0 0 2
  745 root      1:59 hd_demo1 0 0 2

root@NVTEVM:~$ echo nvtmpp dumpstack 734 > /proc/hdal/comm/cmd”

[ 6091.602715] Task[hd_demo1] ->  N-Pri: 120, Rt-Pri: 00, Sched-Policy: 0
[ 6091.614933] Status : [Sleep-Interruptible]
[ 6091.619549] [<8036189c>] (__schedule) from [<80361a7c>] (schedule+0xa0/0xb8)
```

```
[ 6091.626901] [<80361a7c>] (schedule) from [<803654ac>] (schedule_timeout+0x20/0x1ec)
[ 6091.634642] [<803654ac>] (schedule_timeout) from [<800630dc>] (wait_woken+0x64/0xc0)
[ 6091.644208] [<800630dc>] (wait_woken) from [<80230fe0>] (n_tty_read+0x488/0x918)
[ 6091.651842] [<80230fe0>] (n_tty_read) from [<8022bb04>] (tty_read+0x8c/0xc8)
[ 6091.659140] [<8022bb04>] (tty_read) from [<8010046c>] (__vfs_read+0x34/0xd8)
[ 6091.666266] [<8010046c>] (__vfs_read) from [<80100b58>] (vfs_read+0x8c/0xf4)
[ 6091.673879] [<80100b58>] (vfs_read) from [<801012f0>] (SyS_read+0x44/0x7c)
[ 6091.680819] [<801012f0>] (SyS_read) from [<8000f7c0>] (ret_fast_syscall+0x0/0x44)
```

4.2 Debug menu for IPC

The currently supported common module debug menu is as below.

```
==============================
 COMMON
 01 : dump hdal version
 02 : dump hdal flow setting
 03 : dump media memory info
 04 : dump media memory info when get block fail
 05 : check hdal memory block corruption periodically
 06 : dump media tasks info
 07 : dump media semaphores info
 08 : dump dram 1 dma usage
 09 : dump dram 2 dma usage
------------------------------
```

User can choose the number to dump the status what you want. The dump result is just like the example shows on 4.1.

The proc command and debug menu mapping table is as below:

Proc command	Debug menu
cat /proc/hdal/comm/info	dump memory info
echo nvtmpp showmsg 1 > /proc/hdal/comm/cmd	dump memory info when get block fail
echo nvtmpp chkmem_corrupt 1 > /proc/hdal/comm/cmd	check hdal memory block corruption periodically
cat /proc/hdal/comm/task	dump media tasks info

cat /proc/hdal/comm/sem	dump media semaphores info
cat /proc/nvt_drv_sys/dram1_info	dump dram 1 dma usage
cat /proc/nvt_drv_sys/dram2_info	dump dram 2 dma usage

4.3 proc command for NVR

4.3.1 Set Debug Level and Dump Log

The dbglevel command is used to enable the debug level to gain more detailed log information for debugging. To select the suitable level for logging file is the most important; otherwise, the debug log cannot be tracked due to the redundant information.

Level: 1 is suitable for all most debugging cases.
Level: 0 is used to disable tho log information.

Example:
<set debug level>
echo 1 > /proc/videograph/gmlib/dbglevel
echo 1 > /proc/videograph/vpd/dbglevel
echo 1 > /proc/videograph/em/dbglevel
echo 1 > /proc/videograph/gs/dbglevel
echo 1 > /proc/vid eograph/datain/dbglevel
echo 1> /proc/videographdataout/dbglevel

cat /proc/videograph/dumplog

4.3.2 Video Graph View

The video graph was dynamically built according to the condition of the object binds and attribute settings.
The purpose is for debugging the correctness of AP setting

Example:

```
root@NVTEVM:~$
root@NVTEVM:~$ cat /proc/videograph/graph
```

```
entity (jobs)                pool name (done buffers/max buffers)
-----------------------------------------------------------
vcap0_0_4_0 (1)              (disp0_cap_out_ddr0, --, --, --) (1/3)
  ->vpe536lib_0_0_0 (0)       (disp0_in_ddr0, --, --, --) (0/3)
    ->*osg_0_0_0 (0)          (disp0_in_ddr0, --, --, --) (0/0)
      ->lcd0vg_0_0_0 (2)
vcap0_0_6_0 (1)              (disp0_cap_out_ddr0, --, --, --) (1/3)
  ->vpe536lib_0_0_1 (0)       (disp0_in_ddr0, --, --, --) (0/3)
vcap0_0_0_0 (1)              (disp0_cap_out_ddr0, --, --, --) (1/3)
  ->vpe536lib_0_0_2 (0)       (disp0_in_ddr0, --, --, --) (0/3)
vcap0_0_2_0 (1)              (disp0_cap_out_ddr0, --, --, --) (1/3)
  ->vpe536lib_0_0_3 (0)       (disp0_in_ddr0, --, --, --) (0/3)
symbol '*' means IN/OUT with same buffer.
root@NVTEVM:~$
```

4.3.3 Performance Statistic for Encode

In the encode, users can use this command for observing the performance during the specific period set when the unit of the first argument is second.

Example:

```
echo 10 > /proc/videograph/perf


 bindfd(0x20000): 301 frames in 10000 ms
 bindfd(0x20004): 301 frames in 10000 ms
 bindfd(0x20007): 301 frames in 10011 ms
 bindfd(0x20001): 302 frames in 10034 ms
 bindfd(0x20005): 302 frames in 10034 ms
 bindfd(0x20002): 302 frames in 10033 ms
 bindfd(0x20003): 302 frames in 10033 ms
 bindfd(0x20006): 302 frames in 10033 ms
```

4.3.4 Performance Statistic for display

In the playback or liveview, users can use this command for observing the performance during the 10 seconds statistic.

Example:

```
root@NVTEVM:~$ cat /proc/videograph/statistic


(periods 10130 ms)
  entity          (job counts)finished   failed
-----------------------------------------------------
vcap0_0_4_0                  304         0
  ->vpe536lib_0_0_0           304         0
    ->osg_0_0_0               304         0
      ->lcd0vg_0_0_0          304         0
vcap0_0_6_0                  304         0
  ->vpe536lib_0_0_1           304         0
vcap0_0_0_0                  304         0
  ->vpe536lib_0_0_2           304         0
vcap0_0_2_0                  304         0
  ->vpe536lib_0_0_3           304         0
root@NVTEVM:~$
```

4.3.5 Video and Audio Pool Information

According to the video graph view, users can dump the pool information, such as disp0_cap_out, disp0_in, …

Example:

```
root@NVTEVM:~$
root@NVTEVM:~$ cat /proc/videograph/buffer/
au_dec            disp1_cap_out      disp2_in           disp4_enc_scl_out   disp_dec_out
au_enc            disp1_enc_out      disp3_cap_out       disp4_in           disp_dec_out_ratio
common            disp1_enc_scl_out   disp3_enc_out      disp5_cap_out       enc_cap_out
disp0_cap_out      disp1_in           disp3_enc_scl_out   disp5_enc_out       enc_out
```

```
disp0_enc_out      disp2_cap_out      disp3_in          disp5_enc_scl_out   enc_scl_out
disp0_enc_scl_out  disp2_enc_out      disp4_cap_out     disp5_in            flow_md
disp0_in           disp2_enc_scl_out  disp4_enc_out     disp_dec_in


root@NVTEVM:~$ cat /proc/videograph/buffer/disp0_cap_out
#### disp0_cap_out pool ####
  DDR0 size 9368KB

#### disp0_cap_out flow ####
  DDR0 size 573 KB x 16
  ----------------------root@NVTEVM:~$
```

4.4 Debug menu for NVR

```
==============================
 COMMON
------------------------------
 01 : UTIL
 02 : VPD
 03 : USR
 04 : EM
 05 : MS
 06 : GS
 07 : POOL
------------------------------
 254 : Quit
 255 : Return
------------------------------
01
Run: 01 : UTIL

==============================
 UTIL
------------------------------
```

```
01 : Start to dump log
02 : Set log to storage
03 : Set log to console
04 : Set log to direct print
05 : dump the statistic of display per 10s
06 : dump the statistic of encode per 10s
07 : turn off the statistic of encode
08 : dump the videograph
------------------------------
254 : Quit
255 : Return
------------------------------
```

User can choose the number to dump the status what you want. The dump result is just like the example shows on 4.3.

The proc command and debug menu mapping table is as below:

Proc command	Debug menu
echo 1 > /proc/videograph/vpd/dbglevel	VPD (set dbglevel)
echo 1 > /proc/videograph/em/dbglevel	EM (set dbglevel)
echo 1 > /proc/videograph/ms/dbglevel	MS (set dbglevel)
echo 1 > /proc/videograph/gs/dbglevel	GS (set dbglevel)
cat /proc/videograph/buffer/pool_filename	POOL
cat /proc/videograph/dumplog	UTIL/Start to dump log
echo 0 > /proc/videograph/mode	UTIL/Set log to storage
echo 1 > /proc/videograph/mode	UTIL/Set log to console
echo 2 > /proc/videograph/mode	UTIL/Set log to direct print
cat /proc/videograph/statistic	UTIL/dump the statistic of display per 10s
echo 10 > /proc/videograph/perf	UTIL/dump the statistic of encode per 10s
echo 0 > /proc/videograph/perf	UTIL/turn off the statistic of encode

cat /proc/videograph/graph	UTIL/dump the videograph

5 FAQ

5.1 Cma meaning

Question: Cma0, dsp_cma0, dsp_cma1 memories meaning, can this part of the memory be used by ARM application?

Answer:

- cma0: The continuous memory that can be used by hdal drivers.
- dsp_cma0: The continuous memory that reserved for CEVA DSP1.
- dsp_cma1: The continuous memory that reserved for CEVA DSP2.
- cma0，dsp_cma0 and dsp_cma1 these memories are also managed by Linux OS.

If dsp are not opened, the dsp_cma0 and dsp_cma1 will also be used by Linux OS, the memory can be allocated to user space process.

Note: The memory layout /* Linux system memory region*/ <0x00000000 0x12800000> should contains cma0、dsp_cma0、dsp_cma1 these three region.

User can refer **chapter 1.1** for more detail descriptions.

5.2 How to analyze Cma usage

- command “cat /proc/meminfo” to check the CmaTotal and CmaFree size.

```
root@NVTEVM:~$
root@NVTEVM:~$ cat /proc/meminfo
MemTotal:         112964 kB
MemFree:           26216 kB
MemAvailable:      58228 kB
Buffers:              72 kB
Cached:            27696 kB
SwapCached:            0 kB
Active:            10428 kB
Inactive:          17884 kB
Active(anon):        548 kB
Inactive(anon):        0 kB
Active(file):       9880 kB
Inactive(file):    17884 kB
Unevictable:           0 kB
Mlocked:               0 kB
HighTotal:             0 kB
HighFree:              0 kB
LowTotal:         112964 kB
LowFree:           26216 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                12 kB
Writeback:             0 kB
AnonPages:           552 kB
Mapped:             1156 kB
Shmem:                 0 kB
Slab:              27608 kB
SReclaimable:      14388 kB
SUnreclaim:        13220 kB
KernelStack:         528 kB
PageTables:          132 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:       56480 kB
Committed_AS:       2416 kB
VmallocTotal:    1941504 kB
VmallocUsed:      463692 kB
VmallocChunk:    1215036 kB
CmaTotal:          20480 kB
CmaFree:            4096 kB
```

- Confirm the usage status of each cma area:

  root@NVTEVM:~$ cd /sys/kernel/debug/cma/

  root@NVTEVM: /sys/kernel/debug/cma$ ls

  alloc   cma-0   cma-1   cma-2   free

  root@NVTEVM: /sys/kernel/debug/cma$ cd cma-0/

  root@NVTEVM: /sys/kernel/debug/cma/cma-0$ ls

  Alloc bitmap free order_per_bit

  Base_pfn count maxchunk used

- The bitmap represents the Memory usage map: a bit represents a page

  root@NVTEVM: /sys/kernel/debug/cma/cma-0$ cat bitmap

  3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

3 => 11 (binary)

- Maximum chunk size
  root@NVTEVM: /sys/kernel/debug/cma/cma-0$ cat maxchunk
  2043 => the maximum chunk size is 2043 page.

- Used pages:
  root@NVTEVM: /sys/kernel/debug/cma/cma-0$ cat used
  5 => 5 page used

- Total pages:
  root@NVTEVM: /sys/kernel/debug/cma/cma-0$ cat count
  2048 => total of 2048 page size

5.3 Mmap fail

Question: How to debug when uart1 log shows error as below:
ERR: [COMMON][hd_common_mem_mmap():]: mmap fail ?

Answer:

- Check if the map size input correct?
- Check if map a lot of memories and forget to use hd_common_mem_munmap() to un-map un-used memories ?
  Can use command “cat /proc/$PID/maps” to check the process maps memory.
  Please check if the map memory regions reasonable?

```
root@NVTEVM:~$
root@NVTEVM:~$ cat /proc/602/maps
00010000-00013000 r-xp 00000000 00:0e 667        /usr/bin/hd_common_test
00022000-00024000 rw-p 00002000 00:0e 667        /usr/bin/hd_common_test
007f1000-00812000 rw-p 00000000 00:00 0          [heap]
74b46000-74d46000 rw-s 14012000 00:06 3701       /dev/nvtmpp
74d46000-74f46000 rw-s 14012000 00:06 3701       /dev/nvtmpp
74f46000-75146000 rw-s 14012000 00:06 3701       /dev/nvtmpp
75146000-75346000 rw-s 14012000 00:06 3701       /dev/nvtmpp
75346000-75546000 rw-s 14012000 00:06 3701       /dev/nvtmpp
75546000-75746000 rw-s 14012000 00:06 3701       /dev/nvtmpp
75746000-75946000 rw-s 14012000 00:06 3701       /dev/nvtmpp
75946000-75b46000 rw-s 14012000 00:06 3701       /dev/nvtmpp
75d46000-75d47000 ---p 00000000 00:00 0
75d47000-76547000 rw-p 00000000 00:00 0
76547000-765bf000 rw-s 00000000 00:06 3684       /dev/log_vg
765bf000-765d6000 r-xp 00000000 00:0e 82         /lib/libpthread-2.29.so
765d6000-765e5000 ---p 00017000 00:0e 82         /lib/libpthread-2.29.so
765e5000-765e6000 r--p 00016000 00:0e 82         /lib/libpthread-2.29.so
765e6000-765e7000 rw-p 00017000 00:0e 82         /lib/libpthread-2.29.so
765e7000-765e9000 rw-p 00000000 00:00 0
765e9000-76716000 r-xp 00000000 00:0e 67         /lib/libc-2.29.so
76716000-76726000 ---p 0012d000 00:0e 67         /lib/libc-2.29.so
76726000-76728000 r--p 0012d000 00:0e 67         /lib/libc-2.29.so
76728000-76729000 rw-p 0012f000 00:0e 67         /lib/libc-2.29.so
76729000-7672c000 rw-p 00000000 00:00 0
7672c000-76731000 r-xp 00000000 00:0e 73         /lib/libvendor_media.so
76731000-76740000 ---p 00005000 00:0e 73         /lib/libvendor_media.so
76740000-76741000 r--p 00004000 00:0e 73         /lib/libvendor_media.so
76741000-76742000 rw-p 00005000 00:0e 73         /lib/libvendor_media.so
76742000-76787000 r-xp 00000000 00:0e 70         /lib/libhdal.so
76787000-76796000 ---p 00045000 00:0e 70         /lib/libhdal.so
76796000-76797000 r--p 00044000 00:0e 70         /lib/libhdal.so
76797000-76798000 rw-p 00045000 00:0e 70         /lib/libhdal.so
76798000-7679b000 rw-p 00000000 00:00 0
7679b000-767bb000 r-xp 00000000 00:0e 83         /lib/ld-2.29.so
767c9000-767cb000 rw-p 00000000 00:00 0
767cb000-767cc000 r--p 00020000 00:0e 83         /lib/ld-2.29.so
767cc000-767cd000 rw-p 00021000 00:0e 83         /lib/ld-2.29.so
7e503000-7e524000 rw-p 00000000 00:00 0          [stack]
7e599000-7e59a000 r-xp 00000000 00:00 0          [sigpage]
7e59a000-7e59b000 r--p 00000000 00:00 0          [vvar]
7e59b000-7e59c000 r-xp 00000000 00:00 0          [vdso]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

5.4 Get block fail

```
root@NVTEVM:~$
root@NVTEVM:~$ cat /proc/hdal/comm/info
-------------------------VB PUB INFO ------------------------------------------------------------
Max Count of pools: 32
DDR1: paddr = 0x12800000, vaddr = 0x94000000, size = 0x0d800000, Free = 0x0BE5D000, MaxFreeBlk = 0x0BE5BFF0
Modules: user vdocap vdoprc vdoenc

-------------------------COMMON POOL ------------------------------------------------------------
PoolId     PoolType   DDR     PhyAddr      VirAddr      BlkSize     BlkCnt  Free  MinFree
    0          0x01     1  0x12802000   0x94002000   0x002FC800          2     1        0
     BlkId       BlkHdl      BlkAddr     WantSize    user vdocap vdoprc vdoenc
       1   0x942FEFC0   0x942FF000   0x002FC740       0      1      0      0

PoolId     PoolType   DDR     PhyAddr      VirAddr      BlkSize     BlkCnt  Free  MinFree
    1          0x01     1  0x12DFD000   0x945FD000   0x002F7800          3     3        2

-------------------------PRIVATE POOL -----------------------------------------------------------
PoolName               DDR     PhyAddr      VirAddr     BlkSize     BlkCnt  Free  MinFree
vdocap0.ctrl.ctx         1  0x136E6000   0x94EE6000   0x000136C0          1     0        0
vdoprc0.ctrl.ctx         1  0x136FB000   0x94EFB000   0x0004DDC0          1     0        0
vdoenc.ctrl.ctx          1  0x1374A000   0x94F4A000   0x00021340          1     0        0
vdoprc0.ctrl.in_list     1  0x1376D000   0x94F6D000   0x00000160          1     0        0
vdoprc0.ctrl.work        1  0x1376F000   0x94F6F000   0x00002000          1     0        0
vdoenc.ctrl.max          1  0x13773000   0x94F73000   0x00A2624C          1     0        0
vdoenc.ctrl.raw          1  0x1419B000   0x9599B000   0x00000160          1     0        0
vdoenc.ctrl.bs           1  0x1419D000   0x9599D000   0x00004EC0          1     0        0

-------------------------MEM LAYOUT -------------------------------------------------------------
DDR1: Phy[0x12800000~0x20000000], Vir[0x94000000~0xA1800000] Size=0x0D800000
   Pool-00: Phy[0x12801800~0x12DFC800], Vir[0x94001800~0x945FC800] Size=0x005FB000, common pool
   Pool-01: Phy[0x12DFC800~0x136E5800], Vir[0x945FC800~0x94EE5800] Size=0x008E9000, common pool
   Pool-02: Phy[0x136E5800~0x136FA800], Vir[0x94EE5800~0x94EFA800] Size=0x00015000, vdocap0.ctrl.ctx
   Pool-03: Phy[0x136FA800~0x13749800], Vir[0x94EFA800~0x94F49800] Size=0x0004F000, vdoprc0.ctrl.ctx
   Pool-04: Phy[0x13749800~0x1376C800], Vir[0x94F49800~0x94F6C800] Size=0x00023000, vdoenc.ctrl.ctx
   Pool-05: Phy[0x1376C800~0x1376E800], Vir[0x94F6C800~0x94F6E800] Size=0x00002000, vdoprc0.ctrl.in_list
   Pool-06: Phy[0x1376E800~0x13772800], Vir[0x94F6E800~0x94F72800] Size=0x00004000, vdoprc0.ctrl.work
   Pool-07: Phy[0x13772800~0x1419A800], Vir[0x94F72800~0x9599A800] Size=0x00A28000, vdoenc.ctrl.max
   Pool-08: Phy[0x1419A800~0x1419C800], Vir[0x9599A800~0x9599C800] Size=0x00002000, vdoenc.ctrl.raw
   Pool-09: Phy[0x1419C800~0x141A2800], Vir[0x9599C800~0x959A2800] Size=0x00006000, vdoenc.ctrl.bs
   Pool-10: Phy[0x141A2800~0x1FFFF800], Vir[0x959A2800~0xA17FF800] Size=0x0BE5D000, [free]

-------------------------ERR STATUS -------------------------------------------------------------
Modules:          user   vdocap   vdoprc   vdoenc
Get blk Fail:        0        0        0        0
Loc blk Fail:        0        0        0        0
Unl blk Fail:        0        0        0        0
root@NVTEVM:~$
```

If the ERR STATUS has some no-zero value on “Get blk fail” field that means some module fail to get a free buffer from common pools. User has two methods to adjust the buffer count.

- If the memory is large enough, you can first allocate each Blkcnt of the public pools a little more, and then use “cat /proc/hdal/comm/info” to see what the actual buffer count should be. You can use the MinFree info to determine how many buffers have been used at most. (The value BlkCnt minus MinFree) should be the actual number of buffer demand blocks.
- If the memory is not enough to allocate more Blkcnt to publics pools. User can enable the debug mode by type the command "echo nvtmpp showmsg 1 > /proc/hdal/comm/cmd", which will dump the current buffer usage when the get block fail, and you can analyze whether the number of blocks occupied by modules are reasonable. When get block fail, it will repeat dump buffer info status, just take the message of the first dump. The second message can be ignored because this is caused by dumping too many messages.

5.5 Lock/unlock block fail

```
root@NVTEVM:~$
root@NVTEVM:~$ cat /proc/hdal/comm/info
-------------------------VB PUB INFO -------------------------------------------------------------
Max Count of pools: 32
DDR1: paddr = 0x12800000, vaddr = 0x94000000, size = 0x0d800000, Free = 0x0BE5D000, MaxFreeBlk = 0x0BE5BFF0
Modules: user vdocap vdoprc vdoenc

-------------------------COMMON POOL -------------------------------------------------------------
PoolId     PoolType   DDR     PhyAddr      VirAddr      BlkSize     BlkCnt  Free  MinFree
    0          0x01     1  0x12802000   0x94002000   0x002FC800          2     1        0
     BlkId       BlkHdl      BlkAddr     WantSize    user vdocap vdoprc vdoenc
       1   0x942FEFC0   0x942FF000   0x002FC740       0      1      0      0

PoolId     PoolType   DDR     PhyAddr      VirAddr      BlkSize     BlkCnt  Free  MinFree
    1          0x01     1  0x12DFD000   0x945FD000   0x002F7800          3     3        2

-------------------------PRIVATE POOL ------------------------------------------------------------
PoolName              DDR     PhyAddr      VirAddr     BlkSize     BlkCnt  Free  MinFree
vdocap0.ctrl.ctx        1  0x136E6000   0x94EE6000   0x000136C0          1     0        0
vdoprc0.ctrl.ctx        1  0x136FB000   0x94EFB000   0x0004DDC0          1     0        0
vdoenc.ctrl.ctx         1  0x1374A000   0x94F4A000   0x00021340          1     0        0
vdoprc0.ctrl.in_list    1  0x1376D000   0x94F6D000   0x00000160          1     0        0
vdoprc0.ctrl.work       1  0x1376F000   0x94F6F000   0x00002000          1     0        0
vdoenc.ctrl.max         1  0x13773000   0x94F73000   0x00A2624C          1     0        0
vdoenc.ctrl.raw         1  0x1419B000   0x9599B000   0x00000160          1     0        0
vdoenc.ctrl.bs          1  0x1419D000   0x9599D000   0x00004EC0          1     0        0

-------------------------MEM LAYOUT --------------------------------------------------------------
DDR1: Phy[0x12800000~0x20000000], Vir[0x94000000~0xA1800000] Size=0x0D800000
   Pool-00: Phy[0x12801800~0x12DFC800], Vir[0x94001800~0x945FC800] Size=0x005FB000, common pool
   Pool-01: Phy[0x12DFC800~0x136E5800], Vir[0x945FC800~0x94EE5800] Size=0x008E9000, common pool
   Pool-02: Phy[0x136E5800~0x136FA800], Vir[0x94EE5800~0x94EFA800] Size=0x00015000, vdocap0.ctrl.ctx
   Pool-03: Phy[0x136FA800~0x13749800], Vir[0x94EFA800~0x94F49800] Size=0x0004F000, vdoprc0.ctrl.ctx
   Pool-04: Phy[0x13749800~0x1376C800], Vir[0x94F49800~0x94F6C800] Size=0x00023000, vdoenc.ctrl.ctx
   Pool-05: Phy[0x1376C800~0x1376E800], Vir[0x94F6C800~0x94F6E800] Size=0x00002000, vdoprc0.ctrl.in_list
   Pool-06: Phy[0x1376E800~0x13772800], Vir[0x94F6E800~0x94F72800] Size=0x00004000, vdoprc0.ctrl.work
   Pool-07: Phy[0x13772800~0x1419A800], Vir[0x94F72800~0x9599A800] Size=0x00A28000, vdoenc.ctrl.max
   Pool-08: Phy[0x1419A800~0x1419C800], Vir[0x9599A800~0x9599C800] Size=0x00002000, vdoenc.ctrl.raw
   Pool-09: Phy[0x1419C800~0x141A2800], Vir[0x9599C800~0x959A2800] Size=0x00006000, vdoenc.ctrl.bs
   Pool-10: Phy[0x141A2800~0x1FFFF800], Vir[0x959A2800~0xA17FF800] Size=0x0BE5D000, [free]

-------------------------ERR STATUS --------------------------------------------------------------
Modules:          user    vdocap    vdoprc    vdoenc
Get blk Fail:        0         0         0         0
Loc blk Fail:        0         0         0         0
Unl blk Fail:        0         0         0         0
root@NVTEVM:~$
```

If the ERR STATUS has some no-zero value on “Loc blk fail” or “Unl blk Fail” field that means some module fail to lock/unlock a block of common pools.
Usually the following three error codes:

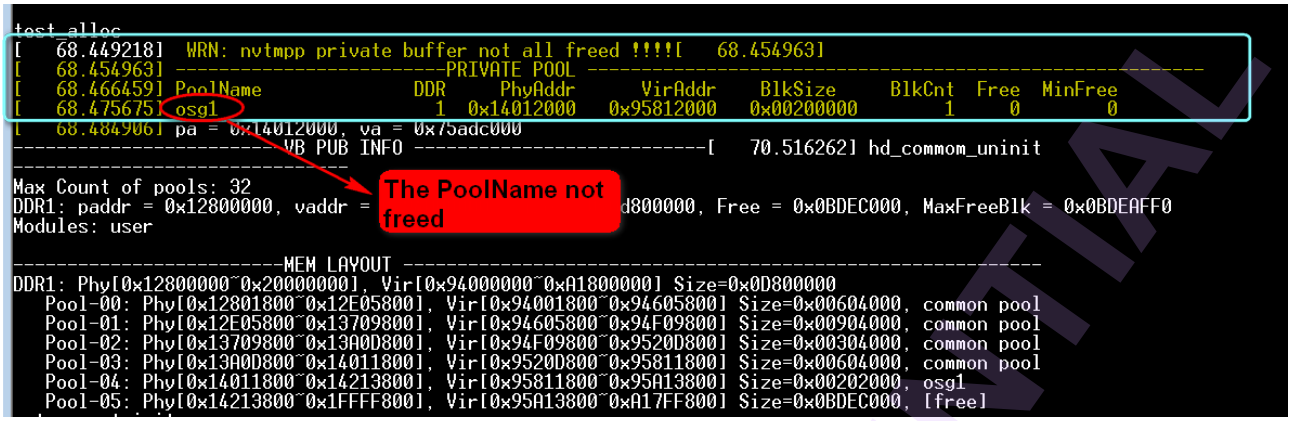
```
NVTMPP_ER_BLK_UNLOCK_REF = -9       ///< The block total reference count is 0 and want to unlock
NVTMPP_ER_BLK_UNLOCK_MODULE_REF = -10 ///< The block module's reference count is 0 and want to unlock
NVTMPP_ER_BLK_ALREADY_FREE = -11     ///< The block is already freed and want to reference
```

The possible reason is that lock/unlock is not paired, or there is a block that is freed but some modules still want to reference it.

5.6 Buffer not all freed

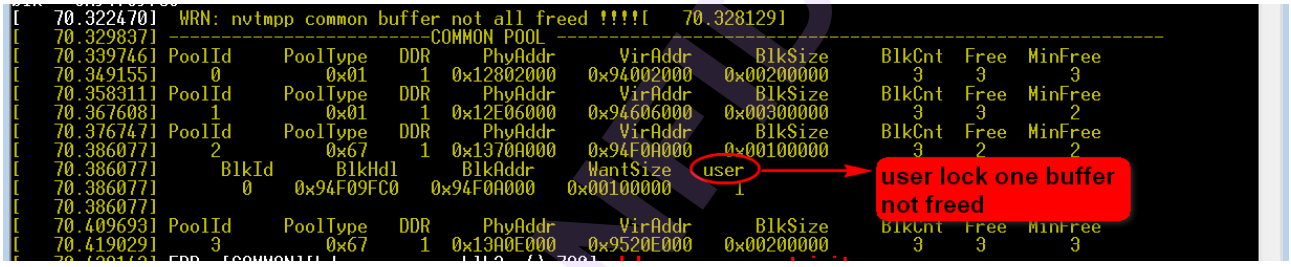
hd_common_mem_uninit() will check if there are common pools or private pools that are not free, and if so, will print warning.

- WRN: nvtmpp private buffer not all freed !!!!
  There is private pool not freed, will dump the PoolName without free.



- WRN: nvtmpp common buffer not all freed !!!!

  If there is any block of common pool not freed, the buffer will be dumped by the status of which module lock it currently.



  Module owner need to check if there is some abnormal condition, causing the common buffer to be locked, no unlocked.

  There are two cases where the block is occupied by the user. One is that the user gets a buffer through hd_common_mem_get_block() API but does not return back the buffer by calling the hd_common_mem_release_block() API. The other is to get the buffer through hd_videoproc_pull_out_buf() API, but the buffer is not returned via hd_videoproc_release_out_buf() API.

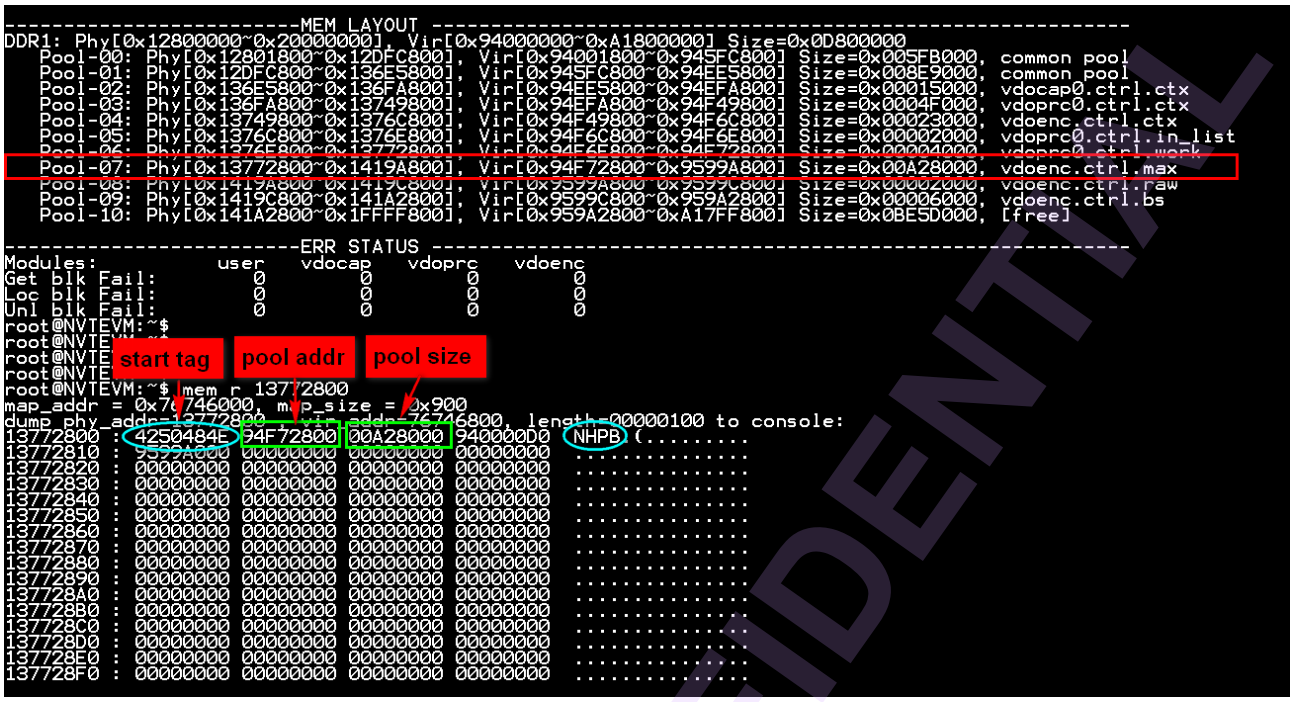
5.7 Memory corruption

Question: How to debug when uart1 log shows error as below:

ERR:_nvtmpp_heap_chk_block:block memory corruption, addr = 0x981ac800, start tag 0x0 != 0x4250484e

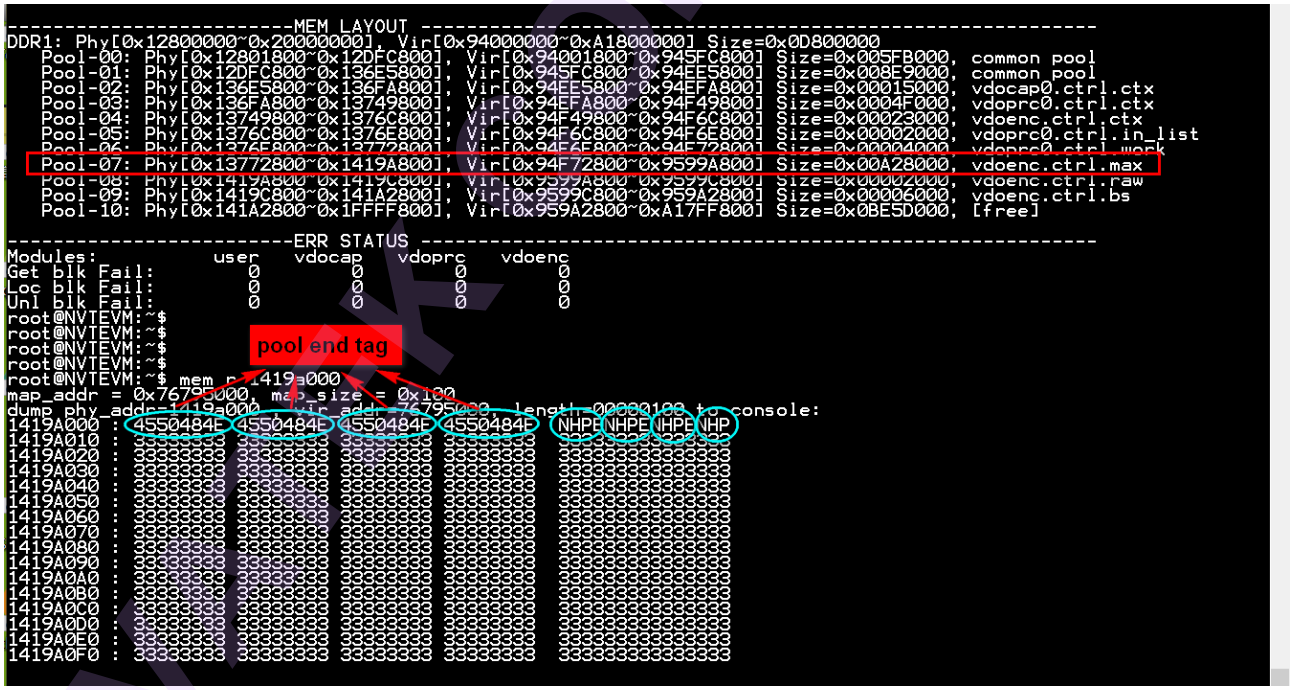
Answer:

The header is placed in front of each pool, in the order of start tag, pool address and pool size as below picture shows.



The start tag value should be equal to 0x4250484E, if the start tag not equal to this value, it means some memory corruption happened.

The end tag is placed at the end of each pool as below shows:



The end tag value should be equal to 0x4550484E, if the end tag not equal to this value, it means some memory corruption happened.

There are several common cases that cause this kind of error (memory corruption):

- The private pool owner uses buffer more than the size of allocated, resulting the end tag of the pool was overwritten.

- The private pool owner uses buffer more than the size of allocated, resulting in the start tag of next private pool was overwritten.
- Random memory overwritten (rarely seen).

The method to debug this kind of error is first to check what is the address overwritten, and the "start tag" or "end tag" is overwritten, and then check the private pool address of the corresponding address. Which module is the above pool, the start tag is usually overwritten by the above pool, and the end tag is usually overwritten by itself.

Sometimes you can also check the overwritten data pattern, to guess what kind of data is probably, and guess the memory is overwritten by CPU or HW engine. Overwritten by CPU can use CPU write protect tool to debug and overwritten by HW engine can use DMA write protect tool to debug.