Hi3536D V100 DDR Miniaturization User Guide

Issue **00B01**

Date **2017-12-13**

HiSilicon Technologies Co., Ltd.

Address:	New R&D Center, Wuhe Road, Bantian, Longgang District, Shenzhen 518129 P. R. China
Website:	http://www.hisilicon.com
Email:	support@hisilicon.com

About This Document

Purpose

This document describes how to perform Linux development, tailoring, and optimization on the Hi3536D V100 board and precautions for programmers who perform miniaturization development based on Hi3536D V100.

Related Version

The following table lists the product version related to this document.

Product Name	Version
Hi3536D	V100

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

Issue 00B01 (2017-12-13)

This issue is the first draft release.

Contents

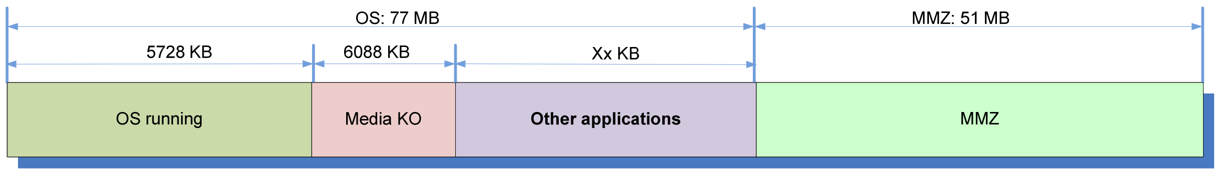
Figures

1 Introduction

DDR miniaturization can be implemented in multiple aspects, that is, the memory for the U-Boot, kernel, file systems, software development kits (SDKs), and applications (apps) can be optimized to some extent. This document mainly describes the miniaturization of SDKs and APPs.

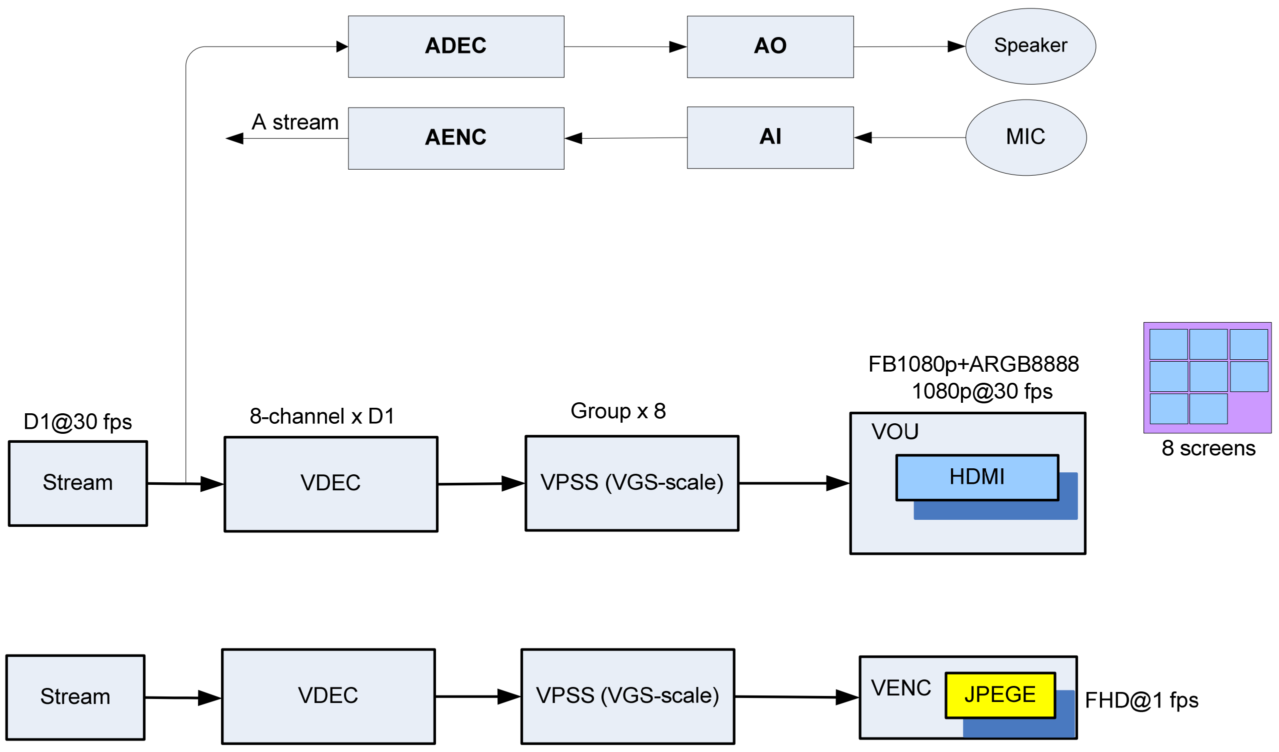
Currently the Hi3536D V100-based SDKs support only the Linux operating system (OS). This document describes the Linux OS by default. The Linux system of Hi3536D V100 is miniaturized based on the demo board (16 MB SPI NOR flash + 512 MB DDR), aiming at performing 8-channel D1 decoding on the 128 MB memory.

Figure 1-1 Memory allocation of the Linux OS on the demo board (for reference only)



Typical scenario: 8-channel D1 decoding+1-channel JPEGE encoding+audio

Figure 1-2 Typical scenario of Hi3536D V100 8-channel D1 decoding



Currently the media memory zone (MMZ) occupies 51 MB when all services are running. The following describes how to optimize the memory usage and miniaturization of the MMZ.

2 MMZ Usage of Major Modules

In general services, the MMZ usage accounts for a large proportion of memory consumption. This section describes the MMZ usage of several major modules in general service scenarios.

2.1 VDEC

The MMZ usage of video decoder (VDEC) is classified into context usage and buffer recycling usage.

Context Usage

- VdecX_Ctx: This memory is used for storing context during decoding and is related to protocol parameters. When **u32ProtocolSwitch** is set to **0**, the context required for decoding occupies the MMZ memory. When **u32ProtocolSwitch** is not set to **0**, the context required for decoding occupies the OS memory.
- VdecX_Scd: This memory is used for storing steams after start code detect (SCD) stream switching. It is related to the resolution and irrelevant to the protocol.
- VFMW_Hal_X: This memory is required for video decoder for high definition (VDH) logic working. This memory has a fixed value of 508 KB.
- VdecX_UsrBuf: This memory is used for storing decoded user data. The memory has a fixed value of 4 KB.
- VdecX_PtsBuf: This memory is used for storing decoded presentation time stamp (PTS) data. The memory has a fixed value of 4 KB.

Buffer Recycling Usage

- VdecX_StreamBuf: This memory is used for storing decoded stream buffer, and is the sum of the user-specified size and the internally allocated size of the driver.
- Decoded VB: The size is determined by the macro VB_PIC_BLK_SIZE, and the number of VBs is (Number of reference frames + Number of display frames + 1).
- PMV VB: The size is determined by the macro VB_PMV_BLK_SIZE, and the number of VBs is the same as that of the decoded VB.

2.2 VPSS

The group occupies two VBs (the upstream module sends the current working VB and the buffer that stores backup streams to the group). Each enabled channel obtains a VB of the channel size. Being processed by the hardware, the VB is sent to the back-end bound module. If the rotation function is required, a temporary buffer (public VB) needs to be applied for.

In addition, the VPSS dynamically allocates the MMZ memory according to whether luminance statistics collection and dynamic contrast improvement (DCI) functions are enabled.

2.3 VGS

For the video graphics subsystem (VGS) module, the fixed MMZ memory is allocated according to the number of jobs, nodes, and tasks.

2.4 VENC

In addition to the VB used to store original frames sent from the front end, another stream buffer (using the MMZ memory) needs to be allocated to the video encoder (VENC) module.

A stream buffer (using the MMZ memory) needs to be allocated to the JPEGE module.

2.5 VOU

In single mode, the video output unit (VOU) occupies three private VBs to display recycling.

In multi mode, if the bound front-end module VPSS is in auto mode, the VOU occupies four private VBs to display recycling. If the front end is in user mode, the VOU does not need to be allocated with VBs. In this case, the VB sent from the front-end module is occupied, and is released after display.

2.6 HiFB

When the.ko is loaded, the user specifies the size of the display buffer at the graphics layer and the mouse layer.

When the compression function is enabled at the graphics layer, the HiSilicon frame buffer (HiFB) applies for a compressed buffer to store the compressed write back (WBC) images.

2.7 Audio

AI

The channel buffer (**FrmBuf**) is allocated according to **u32FrmNum** and **u32PtNumPerFrm**.

AO

- The channel buffer (**Dma&Frm**) is allocated according to **u32FrmNum** and **u32PtNumPerFrm**.
- The audio frame buffer (**CirBuf**) is allocated according to **u32FrmNum**, **u32PtNumPerFrm**, and **u32ChnCnt**.

AENC

- The stream buffer (**StrmBuf**) is allocated according to **u32BufSize**.
- The ring buffer (**CirBuf**) is allocated according to the number of encoding channels.

2.8 Region

Region Information Context Node

The 1024 region information context nodes allocated during module loading occupy 4 KB of the OS memory. The region information context is dynamically allocated during region creation.

If the region is of the Overlay or OverlayEx type, a ping-pong buffer is also created to store the bitmap data.

The size of the ping-pong buffer is determined by user-specified **u32Width** and **u32Height**. The ping-pong buffer uses the MMZ memory.

Channel Management Information Node

When other modules call the region function to register information with the region module, the channel management information nodes are dynamically allocated and occupy the MMZ memory.

2.9 TDE

The channel uses the MMZ with a fixed value: (HI_TDE_CMD_NUM) x 64 + (HI_TDE_JOB_NUM) x 96 + (HI_TDE_NODE_NUM) x 256 + (HI_TDE_FILTER_NUM) x 1024.

3 Optimized Configuration for Module Memories

3.1 VDEC

- The memory can be saved in the decoding context. You can create a channel according to the minimum width and height that are actually used to reduce the VdecX_Scd memory. You can decrease values of channel protocol parameters (such as **SLICE**, **PPS**, **SPS**, and **VPS**) as much as possible based on the stream characteristics to reduce VdecX_Ctx memory usage.
- The memory can be saved through stream buffer configuration. You can set **MiniBufMode** to **1**, and then **u32BufSize** can be set to a minimum value of **32** KB.
- The memory can be saved through protocol switching. You can set **u32ProtocolSwitch** to **2** and allocate memory according to the protocol when the channel is created. When **s32DecMode** is set to **IP**, the PMV VB memory is not required if the protocol is switched to PT_H264 or PT_MP4VIDEO.
- The memory can be saved through VB configuration. You can set **u32RefFrameNum** according to the number of reference frames in the stream to reduce **s32DisplayFrameNum**. You can reduce the number of private VBs when using private VBs. During H.264 non-B-frame decoding, you can set **bTemporalMvpEnable** to **0**.

NOTE

- Hi3536D V100 does not support PT_MP4VIDEO decoding.
- For details, see chapter 10 "VDEC" in the *HiMPP V3.0 Media Processing Software Development Reference*.

3.2 VPSS

By default, the backup function is enabled in the VPSS and occupies a VB block sent from the front end. This function can be disabled by interface calling to save a VB block.

3.3 VGS

The VGS module mainly occupies the OS memory and the nodes occupy the MMZ memory. Decreasing the maximum numbers of jobs, tasks, and nodes can reduce the used OS memory and the node-used MMZ memory.

3.4 VENC

The stream buffer configuration supports general mode and memory-saving mode. In the latter mode, the size of the stream buffer can be reduced. For details, see the *HiMPP V3.0 Media Processing Software Development Reference*.

3.5 VOU

If the default threshold of the VOU is **3** and a small amount of frame loss is allowed, the threshold can be changed to **2**. In this case, the accumulation of the channel VB is reduced and the VB is saved.

- When the VOU is in multi mode and the front end is in user mode, **dispbuflen** of the VO can be set to **0** and the private VB does not need to be allocated.
- When the VOU is in multi mode and the front end is in auto mode, intermittence can be tolerated to reduce the number of VB blocks, and **dispbuflen** of the VO can be set to **3** to reduce a VB block.

3.6 HiFB

When the .ko is loaded, the display buffer size of overlay graphics layers is calculated according to the pixel format, resolution, and buffer mode. You can obtain the required display buffer size according to the actual application scenario.

3.7 Audio

AI

You can reduce the values of **u32FrmNum** and **u32PtNumPerFrm** according to actual scenarios.

AO

You can reduce the values of **u32FrmNum**, **u32PtNumPerFrm**, and **u32ChnCnt** according to actual scenarios.

AENC

You can reduce the **u32BufSize** value and the number of encoding channels according to actual scenarios.

3.8 Region

The user-specified **u32Width** and **u32Height** for allocating the ping-pong buffer must be as small as possible.

When the two-dimensional engine (TDE) module is not loaded, memcpy is used for copying bitmap data in the region module.

4 Other Measures

4.1 Limiting the Stack Size

The default stack size is 8092 KB. A smaller memory size may cause failure of thread creation. You can change the stack size limit to 1024 KB based on the actual stack space required by services. If there are fewer services, the stack size can be changed to 512 KB or smaller.

You can modify the stack size by using either of the following methods:

- Run the **ulimit – s 1024** command once before the application starts.
- Call the pthread_attr_setstacksize function at the beginning of the main function to modify the stack space of a single application.

4.2 Optimizing Memory Usage in Codes

You can optimize the memory usage in codes, especially usage of stack, heap, constants, and global variables. Pay attention to the following points:

- Avoid applying for variables in the application code but not using them.
- Do not immoderately apply for large blocks of memory. On-demand application is recommended.
- Do not initialize unused functional modules to reduce memory usage.

4.3 Disabling Fork and System Functions for Creating Processes in Applications

The data segment of the main process occupies a large amount of memory. If the fork subprocess definitely uses a large amount of memory and there is a high probability that the process fails, the fork and system functions used for creating processes should be disabled in the application. For example, commands such as **himm** and **mkfs.vfat** should not be executed, because they will call fork and system functions.

4.4 Removing Unnecessary Modules Based on Scenarios

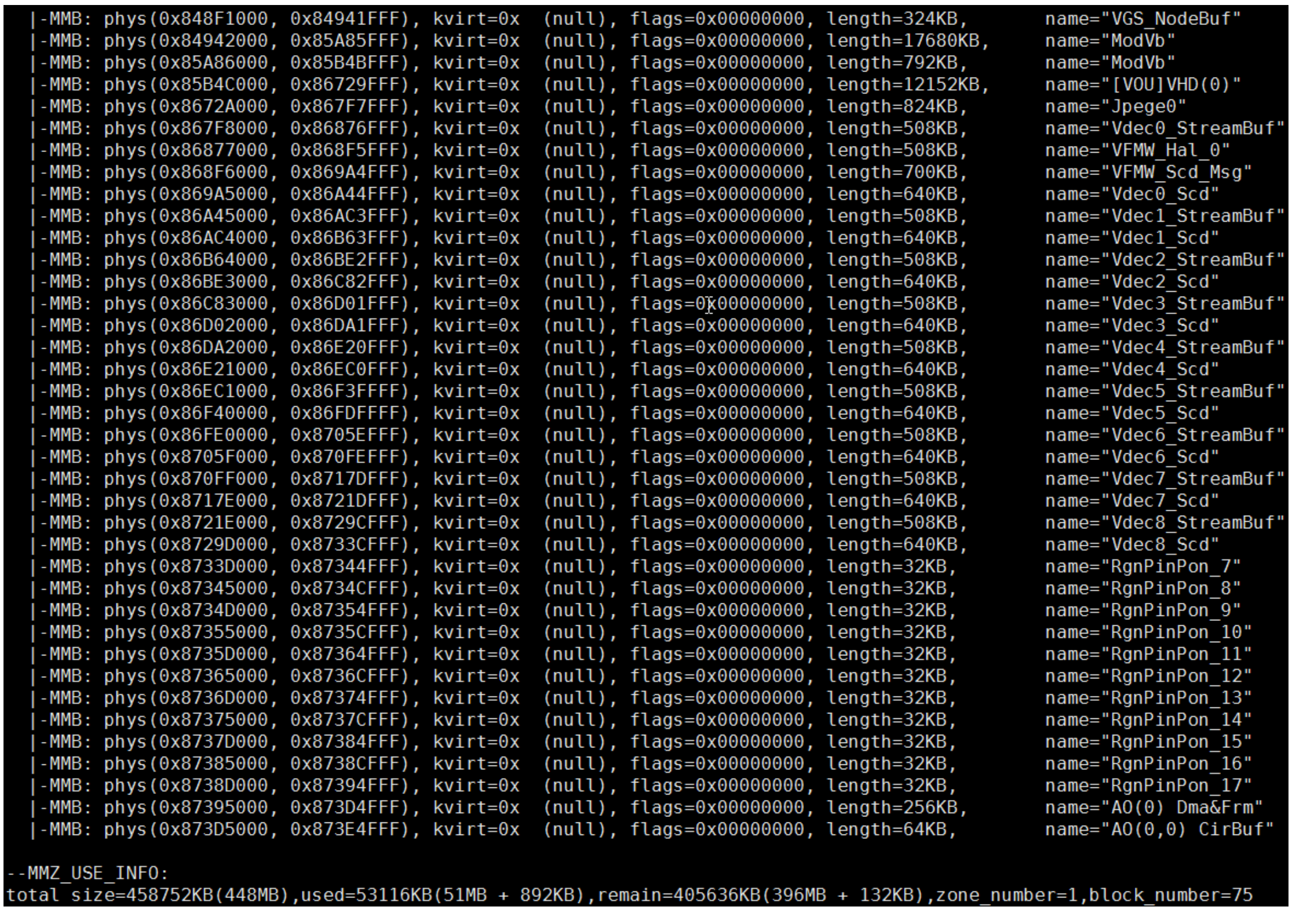
You can remove unnecessary modules to reduce memory usage. The following lists some examples:

- In HiFB 0buffer or standard mode, the TDE does not need to be loaded. If the TDE is not loaded, the region module uses memcpy to copy Overlay and OverlayEx.
- In scenarios that do not require AI or encoding, the AI and AENC modules do not need to be loaded.
- In scenarios that do not use the region, the region module does not need to be loaded.
- In scenarios that do not use JPEGE, the channel, VENC, and JPEGE modules do not need to be loaded.

Typical scenario: 8-channel D1 decoding+1-channel JPEGE encoding+audio

Figure 4-1 shows the MMZ usage statistics (Linux).

Figure 4-1 MMZ usage statistics (Linux)



```
 |-MMB: phys(0x848F1000, 0x84941FFF), kvirt=0x  (null), flags=0x00000000, length=324KB,      name="VGS_NodeBuf"
 |-MMB: phys(0x84942000, 0x85A85FFF), kvirt=0x  (null), flags=0x00000000, length=17680KB,    name="ModVb"
 |-MMB: phys(0x85A86000, 0x85B4BFFF), kvirt=0x  (null), flags=0x00000000, length=792KB,      name="ModVb"
 |-MMB: phys(0x85B4C000, 0x86729FFF), kvirt=0x  (null), flags=0x00000000, length=12152KB,    name="[VOU]VHD(0)"
 |-MMB: phys(0x8672A000, 0x867F7FFF), kvirt=0x  (null), flags=0x00000000, length=824KB,      name="Jpege0"
 |-MMB: phys(0x867F8000, 0x86876FFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="Vdec0_StreamBuf"
 |-MMB: phys(0x86877000, 0x868F5FFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="VFMW_Hal_0"
 |-MMB: phys(0x868F6000, 0x869A4FFF), kvirt=0x  (null), flags=0x00000000, length=700KB,      name="VFMW_Scd_Msg"
 |-MMB: phys(0x869A5000, 0x86A44FFF), kvirt=0x  (null), flags=0x00000000, length=640KB,      name="Vdec0_Scd"
 |-MMB: phys(0x86A45000, 0x86AC3FFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="Vdec1_StreamBuf"
 |-MMB: phys(0x86AC4000, 0x86B63FFF), kvirt=0x  (null), flags=0x00000000, length=640KB,      name="Vdec1_Scd"
 |-MMB: phys(0x86B64000, 0x86BE2FFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="Vdec2_StreamBuf"
 |-MMB: phys(0x86BE3000, 0x86C82FFF), kvirt=0x  (null), flags=0x00000000, length=640KB,      name="Vdec2_Scd"
 |-MMB: phys(0x86C83000, 0x86D01FFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="Vdec3_StreamBuf"
 |-MMB: phys(0x86D02000, 0x86DA1FFF), kvirt=0x  (null), flags=0x00000000, length=640KB,      name="Vdec3_Scd"
 |-MMB: phys(0x86DA2000, 0x86E20FFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="Vdec4_StreamBuf"
 |-MMB: phys(0x86E21000, 0x86EC0FFF), kvirt=0x  (null), flags=0x00000000, length=640KB,      name="Vdec4_Scd"
 |-MMB: phys(0x86EC1000, 0x86F3FFFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="Vdec5_StreamBuf"
 |-MMB: phys(0x86F40000, 0x86FDFFFF), kvirt=0x  (null), flags=0x00000000, length=640KB,      name="Vdec5_Scd"
 |-MMB: phys(0x86FE0000, 0x8705EFFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="Vdec6_StreamBuf"
 |-MMB: phys(0x8705F000, 0x870FEFFF), kvirt=0x  (null), flags=0x00000000, length=640KB,      name="Vdec6_Scd"
 |-MMB: phys(0x870FF000, 0x8717DFFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="Vdec7_StreamBuf"
 |-MMB: phys(0x8717E000, 0x8721DFFF), kvirt=0x  (null), flags=0x00000000, length=640KB,      name="Vdec7_Scd"
 |-MMB: phys(0x8721E000, 0x8729CFFF), kvirt=0x  (null), flags=0x00000000, length=508KB,      name="Vdec8_StreamBuf"
 |-MMB: phys(0x8729D000, 0x8733CFFF), kvirt=0x  (null), flags=0x00000000, length=640KB,      name="Vdec8_Scd"
 |-MMB: phys(0x8733D000, 0x87344FFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_7"
 |-MMB: phys(0x87345000, 0x8734CFFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_8"
 |-MMB: phys(0x8734D000, 0x87354FFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_9"
 |-MMB: phys(0x87355000, 0x8735CFFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_10"
 |-MMB: phys(0x8735D000, 0x87364FFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_11"
 |-MMB: phys(0x87365000, 0x8736CFFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_12"
 |-MMB: phys(0x8736D000, 0x87374FFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_13"
 |-MMB: phys(0x87375000, 0x8737CFFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_14"
 |-MMB: phys(0x8737D000, 0x87384FFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_15"
 |-MMB: phys(0x87385000, 0x8738CFFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_16"
 |-MMB: phys(0x8738D000, 0x87394FFF), kvirt=0x  (null), flags=0x00000000, length=32KB,       name="RgnPinPon_17"
 |-MMB: phys(0x87395000, 0x873D4FFF), kvirt=0x  (null), flags=0x00000000, length=256KB,      name="AO(0) Dma&Frm"
 |-MMB: phys(0x873D5000, 0x873E4FFF), kvirt=0x  (null), flags=0x00000000, length=64KB,       name="AO(0,0) CirBuf"

--MMZ_USE_INFO:
total size=458752KB(448MB),used=53116KB(51MB + 892KB),remain=405636KB(396MB + 132KB),zone_number=1,block_number=75
```