MI SYS API

Version 2.10

REVISION HISTORY

Revision No.	Description	Date
2.03	• Initial release	11/12/2018
2.04	• Added MI_SYS_GetFd note description • Added MI_SYS_CloseFd note description	02/21/2019
2.05	• Added API MI_SYS_ConfigDevChnPrivateMMAHeap	03/18/2019
2.06	• Added DMA API	04/02/2019
2.07	• Renamed MI_SYS_ConfigDevChnPrivateMMAHeap to MI_SYS_ConfigPrivateMMAPool and refined api	04/09/2019
2.08	• Removed E_MI_SYS_BUFDATA_META Renamed MI_SYS_MetaData_t to MI_SYS_PerFrameMetaBuf_t Added MI_SYS_PerFrameMetaBuf_t into MI_SYS_BufInfo_t	04/10/2019
2.09	• Added E_MI_SYS_PER_CHN_PORT_OUTPUT_POOL and MI_SYS_PerChnPortOutputPool_t for ouput port private pool	04/19/2019
2.10	• Added MI_SYS_FrameIspInfoType_e and MI_SYS_FrameIspInfo_t • Removed MI_SYS_MetaDataConfig_t	04/29/2019

TABLE OF CONTENTS

REVISION HISTORY i
TABLE OF CONTENTS ii
1. API 参考 1
1.1. 概述 1
1.2. 功能模块 API 1
1.2.1 MI_SYS_Init 3
1.2.2 MI_SYS_Exit 3
1.2.3 MI_SYS_BindChnPort 5
1.2.4 MI_SYS_UnBind_ChnPort 6
1.2.5 MI_SYS_GetBindbyDest 7
1.2.6 MI_SYS_GetVersion 8
1.2.7 MI_SYS_GetCurPts 8
1.2.8 MI_SYS_InitPtsBase 9
1.2.9 MI_SYS_SyncPts 10
1.2.10 MI_SYS_Mmap 11
1.2.11 MI_SYS_FlushInvCache 12
1.2.12 MI_SYS_Munmap 13
1.2.13 MI_SYS_SetReg 14
1.2.14 MI_SYS_GetReg 15
1.2.15 MI_SYS_ConfDevPubPools 15
1.2.16 MI_SYS_ReleaseDevPubPools 16
1.2.17 MI_SYS_ConfGloPubPools 17
1.2.18 MI_SYS_ReleaseGloPubPools 18
1.2.19 MI_SYS_MMA_Alloc 20
1.2.20 MI_SYS_MMA_Free 20
1.2.21 MI_SYS_SetChnMMAConf 22
1.2.22 MI_SYS_GetChnMMAConf 23
1.2.23 MI_SYS_ChnPortInjectBuf 23
1.2.24 MI_SYS_ChnInputPortGetBuf 24
1.2.25 MI_SYS_ChnInputPortPutBuf 26
1.2.26 MI_SYS_ChnOutPortGetBuf 26
1.2.27 MI_SYS_ChnOutPortPutBuf 28
1.2.28 MI_SYS_SetChnOutputPortDepth 29
1.2.29 MI_SYS_GetFd 30
1.2.30 MI_SYS_CloseFd 31
1.2.31 MI_SYS_BindChnPort_SWLowLatency 32
1.2.32 MI_SYS_ReadUuid 32
1.2.33 MI_SYS_BindChnPort2 33
1.2.34 MI_SYS_ConfigPrivateMMAPool 36
1.2.35 MI_SYS_MemsetPa 38
1.2.36 MI_SYS_MemcpyPa 39
1.2.37 MI_SYS_BufFillPa 40
1.2.38 MI_SYS_BufBlitPa 40

2. 数据类型 42
2.1. MI_ModuleId_e 43
2.2. MI_SYS_PixelFormat_e 44
2.3. MI_SYS_CompressMode_e 47
2.4. MI_SYS_FrameTileMode_e 48
2.5. MI_SYS_FieldType_e 49
2.6. MI_SYS_BufDataType_e 51
2.7. MI_SYS_FrameIspInfoType_e 51
2.8. MI_SYS_ChnPort_t 52
2.9. MI_SYS_PerFrameMetaBuf_t 53
2.10. MI_SYS_RawData_t 53
2.11. MI_SYS_WindowRect_t 54
2.12. MI_SYS_FrameData_t 55
2.13. MI_SYS_BufInfo_t 56
2.14. MI_SYS_BufFrameConfig_t 57
2.15. MI_SYS_BufRaw_Config_t 58
2.16. MI_SYS_BufConf_t 59
2.17. MI_SYS_Version_t 59
2.18. MI_VB_PoolListConf_t 60
2.19. MI_SYS_BindType_e 60
2.20. MI_SYS_FrameData_PhySignalType 62
2.21. MI_SYS_InsidePrivatePoolType_e 63
2.22. MI_SYS_PerChnPrivHeapConf_t 63
2.23. MI_SYS_PerDevPrivHeapConf_t 65
2.24. MI_SYS_PerVpe2VencRingPoolConf_t 66
2.25. MI_SYS_PerChnPortOutputPool_t 66
2.26. MI_SYS_GlobalPrivPoolConfig_t 67
2.27. MI_SYS_FrameIspInfo_t 68
3. 错误码 69

1. API 参考

1.1. 概述

MI_SYS 实现 MI 系统初始化、内存缓冲池管理、各个模块之间数据流的管理，绑定管道的建立和连续物理内存池的管理及内存池内内存 生命周期管控等功能。

1.2. 功能模块 API

API 名	功能
MI_SYS_Init	初始化 MI_SYS 系统
MI_SYS_Exit	析构 MI_SYS 系统
MI_SYS_BindChnPort	数据源输出端口到数据接收者输入端口的绑定
MI_SYS_UnBindChnPort	数据源输出端口到数据接收者输入端口的去绑定
MI_SYS_GetBindbyDest	查询数据接收者输入端口的对应的源输出端口
MI_SYS_GetVersion	获取 MI 的系统版本号
MI_SYS_GetCurPts	获取 MI 系统当前时间戳
MI_SYS_InitPtsBase	初始化 MI 系统基准时间戳
MI_SYS_SyncPts	同步 MI 系统时间戳
MI_SYS_Mmap	映射物理内存到 CPU 虚拟地址
MI_SYS_FlushInvCache	Flush cache CPU 虚拟地址
MI_SYS_Munmap	取消物理内存到虚拟地址的映射
MI_SYS_SetReg	设置寄存器的值，调试用
MI_SYS_GetReg	获取寄存器的值，调试用
MI_SYS_ConfDevPubPools	配置并初始化模块公共缓冲池
MI_SYS_ReleaseDevPubPools	释放模块公共缓冲池
MI_SYS_ConfGloPubPools	配置并初始化 MI 全系统默认 VB 缓存池
MI_SYS_ReleaseGloPubPools	释放 MI 全系统默认 VB 缓存池
MI_SYS_MMA_Alloc	应用程序从 MMA 内存管理池申请物理连续内存
MI_SYS_MMA_Free	在用户态释放 MMA 内存管理池中分配到的内存
MI_SYS_SetChnMMAConf	设置模块设备通道输出端口默认分配内存的 MMA 池名称
MI_SYS_GetChnMMAConf	获取模块设备通道输出端口默认分配内存的 MMA 池名称
MI_SYS_ChnPortInjectBuf	向模块通道 inputPort 注入 outputPort Buf 数据
MI_SYS_ChnInputPortGetBuf	获取通道 inputPort 的 buf
MI_SYS_ChnInputPortPutBuf	将通道 inputPort 的 buf 加到待处理队列

API 名	功能
MI_SYS_ChnOutPortGetBuf	获取通道 outputPort 的 buf
MI_SYS_ChnOutPortPutBuf	释放通道 outputPort 的 buf
MI_SYS_SetChnOutputPortDepth	设置通道 OutputPort 的深度
MI_SYS_GetFd	获取当前通道等待事件的文件描述符
MI_SYS_CloseFd	关闭当前通道的文件描述符
MI_SYS_BindChnPort_SWLowLatency	设置绑定 ChnPort 的 SW 低延迟模式
MI_SYS_ReadUuid	获取 Chip 的 Unique ID
MI_SYS_BindChnPort2	数据源输出端口到数据接收者输入端口的绑定，需要指定工作模式
MI_SYS_ConfigPrivateMMAPool	为模快配置私有 MMA Heap

1.2.1 MI_SYS_Init

➢ **功能**

MI_SYS 初始化，MI_SYS 模块为系统内其它 MI 模组提供基础支援，需要早于系统内其它 MI 模组初始化，否则其它内其它 stream 类型的模组初始化时会失败。

➢ **语法**

MI_S32 MI_SYS_Init();

➢ **形参**

无

➢ **返回值**

返回值
- 0 **成功。**
- **非** 0 **失败，参照**错误码**。**

➢ **依赖**

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ **注意**

- MI_SYS_Init 需要早于其他 MI 模组的 Init 函数调用。
- 可以重复调用 MI_SYS_Init。
- 系统在内核启动参数内，需要配置好 MMA 内存堆的配置参数。

➢ **举例**

无

➢ **相关主题**

MI_SYS_Exit

1.2.2 MI_SYS_Exit

➢ **功能**

MI_SYS 初始化， 调用 MI_SYS_Exit 之前，需要确保系统内所有其他模组都已经完成去初始化，所有 VBPOOL 都已经销毁，否则 MI_SYS_Exit 会返回失败。

➢ **语法**

MI_S32 MI_SYS_Exit ();

➢ **形参**

无

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

- MI_SYS_Exit 调用前需要确保系统内所有其他模组都已经完成去初始化。
- MI_SYS_Exit 调用前需要确保系统内所有创建的 VBPOOL 已经成功销毁。

➢ 举例

参见 MI_SYS_Init 举例。

➢ 相关主题

无

1.2.3 MI_SYS_BindChnPort

➢ 功能

数据源输出端口到数据接收者输入端口的绑定。

➢ 语法

MI_S32 MI_SYS_BindChnPort(MI_SYS_ChnPort_t *pstSrcChnPort, MI_SYS_ChnPort_t *pstDstChnPort, ,MI_U32 u32SrcFrmrate, MI_U32 u32DstFrmrate);

➢ 形参

参数名称	参数含义	输入/输出
pstSrcChnPort	源端口配置信息数据结构指针。	输入
pstDstChnPort	目标端口配置信息数据结构指针。	输入
u32SrcFrmrate	源端口配置的帧率	输入
u32DstFrmrate	目标端口配置的帧率	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

- 源端口必须是通道输出端口。
- 目标端口必须是通道输入端口。
- 源和目标端口必须之前没有被绑定过

➢ 举例

参见 MI_SYS_Bind_ChnPort 举例。

➢ 相关主题

MI_SYS_UnBind_ChnPort。

1.2.4 MI_SYS_UnBind_ChnPort

➢ 功能

数据源输出端口到数据接收者输入端口之间的去绑定。

➢ 语法

MI_S32 MI_SYS_UnBindChnPort(MI_SYS_ChnPort_t *pstSrcChnPort, MI_SYS_ChnPort_t *pstDstChnPort);

➢ 形参

参数名称	描述	输入/输出
pstSrcChnPort	源端口配置信息数据结构指针。	输入
pstDstChnPort	目标端口配置信息数据结构指针。	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

- 源端口必须是通道输出端口。
- 目标端口必须是通道输入端口。
- 源和目标端口之间之前必须已经被绑定过

➢ 举例

参见 MI_SYS_Bind_ChnPort 举例。

➢ 相关主题

无。

1.2.5 MI_SYS_GetBindbyDest

➢ 功能

查询数据接收者输入端口的对应的源输出端口。

➢ 语法

MI_S32 MI_SYS_GetBindbyDest (MI_SYS_ChnPort_t *pstDstChnPort, MI_SYS_ChnPort_t *pstSrcChnPort);

➢ 形参

参数名称	描述	输入/输出
pstDstChnPort	目标端口配置信息数据结构指针。	输入
pstDstChnPort	源端口配置信息数据结构指针。	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

- 目标端口必须是通道输入端口。
- 目标端口之前必须已经被绑定过

➢ 举例

无

➢ 相关主题

无

1.2.6 MI_SYS_GetVersion

➢ 功能

获取 MI 的系统版本号。

➢ 语法

MI_S32 MI_SYS_GetVersion (MI_SYS_Version_t *pstVersion);

➢ 形参

参数名称	描述	输入/输出
pstVersion	系统版本号返回数据结构指针	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 举例

无

➢ 相关主题

无

1.2.7 MI_SYS_GetCurPts

➢ 功能

获取 MI 系统当前时间戳。

➢ 语法

MI_S32 MI_SYS_GetCurPts (MI_U64 *pu64Pts);

➢ 形参

参数名称	描述	输入/输出
pu64Pts	系统当前时间戳返回地址	输出

➢ 返回值

返回值
- 0　成功。
- 非 0　失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无。

➢ 举例

无。

➢ 相关主题

无。

1.2.8 MI_SYS_InitPtsBase

➢ 功能

初始化 MI 系统时间戳基准。

➢ 语法

MI_S32 MI_SYS_InitPtsBase (MI_U64 u64PtsBase);

➢ 形参

参数名称	描述	输入/输出
u64PtsBase	设置的系统时间戳基准	输入

➢ 返回值

返回值
- 0　成功。
- 非 0　失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无。

➢ 举例

无。

➢ 相关主题

无。

1.2.9 MI_SYS_SyncPts

➢ 功能

微调同步 MI 系统时间戳。

➢ 语法

MI_S32 MI_SYS_ InitPtsBase (MI_U64 u64Pts);

➢ 形参

参数名称	描述	输入/输出
u64Pts	微调后的系统时间戳基准	输入

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无。

➢ 举例

无。

➢ 相关主题

无。

1.2.10 MI_SYS_Mmap

➢ 功能

映射任意物理内存到当前用户态进程的 CPU 虚拟地址空间。

➢ 语法

MI_S32 MI_SYS_Mmap(MI_U64 u64PhyAddr, MI_U32 u32Size , void *pVirtualAddress , MI_BOOL bCache);

➢ 形参

参数名称	描述	输入/输出
u64PhyAddr	待映射的物理地址	输入
u32Size	待映射的物理地址长度	输入
pVirtualAddress	CPU 虚拟地址指针	输入
bCache	是否 map 成 cache 还是 un-cache	输入

➢ 返回值

返回值 { 0 成功。 / 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

- 物理地址必须 4KByte 对齐。
- 物理地址是 MStar 内存控制器地址， 非 CPU bridge 之地址
- 物理地址长度必须 4KByte 对齐
- 物理内存必须完整的落在 MMA 管理的内存范围内或者 linux kenrel 管理的内存之外

➢ 举例

无

➢ 相关主题

无。

1.2.11 MI_SYS_FlushInvCache

➢ 功能

Flush cache。

➢ 语法

MI_S32 MI_SYS_FlushCache(MI_VOID *pVirtualAddress, MI_U32 u32Size);

➢ 形参

参数名称	描述	输入/输出
pVirtualAddress	之前 MI_SYS_Mmap 返回的 CPU 虚拟地址	输入
u32Size	待 flush 的 cache 长度	输入

➢ 返回值

返回值 { 0 成功。 / 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

- 待 flush cache 的虚拟地址必须 4KByte 对齐。
- 待 flush cache 的映射长度必须 4KByte 对齐
- 待 flush cache 的映射内存范围必须是之前通过 MI_SYS_Mmap API 获取到的,且 MI_SYS_Mmap 时采用的是 cache 的方式进行的 map。

➢ 举例

无。

➢ 相关主题

无。

1.2.12 MI_SYS_Munmap

➢ 功能

取消物理内存到虚拟地址的映射。

➢ 语法

MI_S32 MI_SYS_UnMmap(MI_VOID *pVirtualAddress, MI_U32 u32Size);

➢ 形参

参数名称	描述	输入/输出
pVirtualAddress	之前 MI_SYS_Mmap 返回的 CPU 虚拟地址	输入
u32Size	待取消映射的映射长度	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

- 待取消映射的虚拟地址必须 4KByte 对齐。
- 待取消映射的映射长度必须 4KByte 对齐
- 待取消的映射内存范围必须是之前通过 MI_SYS_Mmap API 获取到的。

➢ 举例

无。

➢ 相关主题

无。

1.2.13 MI_SYS_SetReg

➢ 功能

设置寄存器的值，调试用。

➢ 语法

MI_S32 MI_SYS_SetReg (MI_U32 u32RegAddr, MI_U16 u16Value，MI_U16 u16Mask);

➢ 形参

参数名称	描述	输入/输出
u32RegAddr	Register 总线之地址	输入
u16Value	待写入之 16bit 寄存器值	输入
u16Mask	本次写入寄存器值之 Mask 遮挡栏位	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无。

➢ 举例

无。

➢ 相关主题

无。

1.2.14 MI_SYS_GetReg

- 功能

  读取寄存器的值，调试用。

- 语法

  MI_S32　MI_SYS_SetReg (MI_U32 u32RegAddr, MI_U16 *pu16Value);

- 形参

参数名称	描述	输入/输出
u32RegAddr	Register 总线之地址	输入
pu16Value	待读回 16bit 寄存器值返回地址	输出

- 返回值

  返回值：
  - 0　成功。
  - 非 0　失败，参照错误码。

- 依赖
  - 头文件：mi_sys_datatype.h、mi_sys.h
  - 库文件：libmi_sys.a

※ 注意

  无。

- 举例

  无。

- 相关主题

  无。

1.2.15 MI_SYS_ConfDevPubPools

- 功能

  配置并初始化模块公共缓冲池。

- 语法

  MI_S32 MI_SYS_ConfDevPubPools(MI_ModuleId_e　eModule, MI_U32 u32DevId, MI_VB_PoolListConf_t　stPoolListConf);

- 形参

参数名称	描述	输入/输出
eModule	目标模块 ID	输入
u32DevId	Dev ID	输入
stPoolListConf	模块公共缓冲池队列配置	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 举例

无

➢ 相关主题

无。

1.2.16 MI_SYS_ReleaseDevPubPools

➢ 功能

释放全局的公共缓冲池。

➢ 语法

MI_S32 MI_SYS_ReleaseDevPubPools(MI_ModuleId_e eModule, MI_U32 u32DevId);

➢ 形参

参数名称	描述	输入/输出
eModule	目标模块 ID	输入
u32DevId	Dev ID	输入

➢ 返回值

返回值
- 0 成功。

非 0　　失败，参照错误码。

- 依赖
  - 头文件：mi_sys_datatype.h、mi_sys.h
  - 库文件：libmi_sys.a

※ 注意

无

- 举例

无

- 相关主题

无。

1.2.17 MI_SYS_ConfGloPubPools

- 功能

配置并初始化系统全局公共缓冲池。

- 语法

MI_S32 MI_SYS_ConfGloPubPools(MI_VB_PoolListConf_t stPoolListConf);

- 形参

参数名称	描述	输入/输出
stPoolListConf	模块公共缓冲池队列配置	输入

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 举例

无

➢ 相关主题

无。

1.2.18 MI_SYS_ReleaseGloPubPools

➢ 功能

释放全局的公共缓冲池。

➢ 语法

MI_S32 MI_VB_ ReleaseGloPubPools ();

➢ 形参

参数名称	描述	输入/输出

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

- 举例

  无

➢ 相关主题

无。

1.2.19 MI_SYS_MMA_Alloc

➢ 功能

直接向 MMA 内存管理器申请分配内存。

➢ 语法

MI_S32 MI_SYS_MMA_Alloc(MI_U8 *pstMMAHeapName, MI_U32 u32BlkSize ,MI_PHY *phyAddr);

➢ 形参

参数名称	描述	输入/输出
pstMMAHeapName	目标 MMA heapname	输入
u32BlkSize	待分配的块字节大小	输入
phyAddr	返回的内存块物理地址	输出

➢ 返回值

返回值：
- 0　成功。
- 非 0　失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

➢ 相关主题

无。

1.2.20 MI_SYS_MMA_Free

➢ 功能

直接向 MMA 内存管理器释放之前分配的内存。

➢ 语法

MI_S32 MI_SYS_MMA_Free(MI_U64 phyAddr);

➢ 形参

参数名称	描述	输入/输出
phyAddr	待释放之内存的物理地址	输入

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

➢ 相关主题

无。

1.2.21 MI_SYS_SetChnMMAConf

➢ 功能

设置模块设备通道输出默认分配内存的 MMA 池名称。

➢ 语法

MI_S32 MI_SYS_SetChnMMAConf (MI_ModuleId_e eModId, MI_U32 u32DevId, MI_U32 u32ChnId, MI_U8 *pu8MMAHeapName);

➢ 形参

参数名称	描述	输入/输出
eModId	待配置的模块 ID	输入
u32DevId	待配置的设备 ID	输入
u32ChnId	带配置的通道号	输入
pu8MMAHeapName	MMA heap name	输入

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 相关主题

无。

1.2.22 MI_SYS_GetChnMMAConf

➢ 功能

获取模块设备通道输出端口默认分配内存的 MMA 池名称。

➢ 语法

MI_S32 MI_SYS_GetChnMMAConf (MI_ModuleId_e eModId, MI_U32 u32DevId, MI_U32 u32ChnId, void *data, MI_U32 u32Length);

➢ 形参

参数名称	描述	输入/输出
eModId	待配置的模块 ID	输入
u32DevId	待配置的设备 ID	输入
u32ChnId	带配置的通道号	输入
pu8MMAHeapName	MMA heap name	输入

➢ 返回值

返回值：
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 相关主题

无。

1.2.23 MI_SYS_ChnPortInjectBuf

➢ 功能

把获取的 outputPort buf 插到指定的 inputPort Buf Queue 中

➢ 语法

MI_S32 MI_SYS_ChnPortInjectBuf(MI_SYS_BUF_HANDLE hHandle ,MI_SYS_ChnPort_t *pstChnInputPort);

➢ 形参

参数名称	描述	输入/输出
pstChnPort	指向模块通道之 input 端口的指针	输入
hHandle	获取的 outputPort Buf 的 Idr handle	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 相关主题

无。

1.2.24 MI_SYS_ChnInputPortGetBuf

➢ 功能

分配通道 input 端口对应的 buf object。

➢ 语法

MI_S32 MI_ SYS_ChnInputPortGetBuf (MI_SYS_ChnPort_t *pstChnPort , MI_SYS_BufConf_t *pstBufConf, MI_SYS_BufInfo_t *pstBufInfo, MI_SYS_BUF_HANDLE *phHandle , MI_S32 s32TimeOutMs);

➢ 形参

参数名称	描述	输入/输出
pstChnPort	指向模块通道之 input 端口的指针	输入
pstBufConf	待分配内存配置信息	输入
pstPortBuf	返回之 buf 指针	输出
phHandle	获取的 intputPort Buf 的 Idr handle	输出

s32TimeOutMs	等待超时的毫秒数	输出

➢ 返回值

返回值
- 0　成功。
- 非 0　失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 相关主题

无。

1.2.25 MI_SYS_ChnInputPortPutBuf

➢ 功能

把通道 input 端口对应的 buf object 加到待处理队列。

➢ 语法

MI_S32 MI_ **S**YS_ChnInputPortPutBuf (MI_SYS_BUF_HANDLE hHandle ，MI_SYS_BufInfo_t *pstPortBuf， MI_BOOL bDropBuf);

➢ 形参

参数名称	描述	输入/输出
hHandle	当前 buf 的 Idr Handle	输入
pstPortBuf	待提交之 buf 指针	输入
bDropBuf	直接放弃对 buf 的修改不提交	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 相关主题

无。

1.2.26 MI_SYS_ChnOutPortGetBuf

➢ 功能

分配通道 input 端口对应的 buf object。

➢ 语法

MI_S32 MI_ SYS_ChnOutPortGetBuf (MI_SYS_ChnPort_t *pstChnPort, MI_SYS_BufInfo_t *pstBufInfo , MI_SYS_BUF_HANDLE *phHandle);

➢ 形参

参数名称	描述	输入/输出
pstChnPort	指向模块通道之 input 端口的指针	输入
pstBufInfo	返回之 buf 指针	输出
phHandle	获取的 outputPort Buf 的 Idr handle	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 相关主题

无。

1.2.27 MI_SYS_ChnOutPortPutBuf

➢ 功能

释放通道 output 端口对应的 buf object。

➢ 语法

MI_S32 MI_ SYS_ChnOutPortPutBuf (MI_SYS_BUF_HANDLE hBufHandle);

➢ 形参

参数名称	描述	输入/输出
hBufHandle	待提交之 buf 的 Idr handle	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 相关主题

无。

1.2.28 MI_SYS_SetChnOutputPortDepth

➢ 功能

设置通道 output 端口对应的系统 buf 数量和用户可以拿到的 buf 数量。

➢ 语法

MI_S32 MI_SYS_SetChnOutputPortDepth(MI_SYS_ChnPort_t *pstChnPort, MI_U32 u32UserFrameDepth, MI_U32 u32BufQueueDepth);

➢ 形参

参数名称	描述	输入/输出
pstChnPort	指向模块通道之 output 端口的指针	输入
u32UserFrameDepth	设置该 output 用户可以拿到的 buf 最大数量	输入
u32BufQueueDepth	设置该 output 系统 buf 最大数量	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

➢ 相关主题

无。

1.2.29 MI_SYS_GetFd

- 功能

  获取当前 output Port 等待事件的文件描述符

- 语法

  MI_S32 MI_SYS_GetFd(MI_SYS_ChnPort_t *pstChnPort , MI_S32 *ps32Fd);

- 形参

参数名称	描述	输入/输出
pstChnPort	端口信息结构体指针	输入
ps32Fd	等待事件的文件描述符	输出

- 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

- 依赖
  - 头文件：mi_sys_datatype.h、mi_sys.h
  - 库文件：libmi_sys.a

※ 注意

需要与 MI_SYS_ClosdFd 成对使用。

- 相关主题

无

1.2.30 MI_SYS_CloseFd

- 功能

关闭当前通道的文件描述符

- 语法

```
MI_S32 MI_SYS_CloseFd(MI_S32 s32ChnPortFd);
```

- 形参

参数名称	描述	输入/输出
s32ChnPortFd	等待事件的文件描述符	输入

- 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

- 依赖
  - 头文件：mi_sys_datatype.h、mi_sys.h
  - 库文件：libmi_sys.a

※ 注意

需要与 MI_SYS_GetFd 成对使用。

1.2.31 MI_SYS_BindChnPort_SWLowLatency

➢ 功能

设置绑定 ChnPort 的 SW 低延迟模式

➢ 语法

MI_S32 MI_SYS_BindChnPort_SWLowLatency(MI_SYS_ChnPort_t *pstSrcChnPort, MI_SYS_ChnPort_t *pstDstChnPort,MI_BOOL bEnable,MI_U32 u32DelayMS);

➢ 形参

参数名称	描述	输入/输出
pstSrcChnPort	源端口配置信息数据结构指针	输出
pstDstChnPort	目标端口配置信息数据结构指针	输入
bEnable	使能目标端口低延迟接收模式	输入
u32DelayMS	低延迟模式 delay 时间	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

无

1.2.32 MI_SYS_ReadUuid

➢ 功能

获取 Chip 的 Unique ID

➢ 语法

MI_S32 MI_SYS_ReadUuid (MI_U64 *u64Uuid);

➢ 形参

参数名称	描述	输入/输出
u64Uuid	获取 chip unique ID 值的指针	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

下面的 CHIP 不支持这个功能
MSR930 MSR650x SAV610E SAV538E/S AV638E/S SAV838E/S

1.2.33 MI_SYS_BindChnPort2

➢ 功能

数据源输出端口到数据接收者输入端口的绑定，需要额外指定工作模式。

➢ 语法

MI_S32 MI_SYS_BindChnPort2(MI_SYS_ChnPort_t *pstSrcChnPort, MI_SYS_ChnPort_t *pstDstChnPort,MI_U32 u32SrcFrmrate, MI_U32 u32DstFrmrate, MI_SYS_BindType_e eBindType, MI_U32 u32BindParam);

➢ 形参

参数名称	参数含义	输入/输出
pstSrcChnPort	源端口配置信息数据结构指针。	输入
pstDstChnPort	目标端口配置信息数据结构指针。	输入
u32SrcFrmrate	源端口配置的帧率	输入
u32DstFrmrate	目标端口配置的帧率	输入
eBindType	源端口与目标端口连接的工作模式，参考 MI_SYS_BindType_e	输入
u32BindParam	不同工作模式需带入的额外参数。	输入

➢ 返回值

- 0 成功。

返回值

非 0　失败，参照错误码。

- 依赖
  - 头文件：mi_sys_datatype.h、mi_sys.h
  - 库文件：libmi_sys.a

※ **注意**

- 源端口必须是通道输出端口。
- 目标端口必须是通道输入端口。
- **源和目标端口必须之前没有被绑定过**
- 旧版本 MI SYS 不提供此接口，如没有找到，不用设置
- 当 eBindType = E_MI_SYS_BIND_TYPE_SW_LOW_LATENCY 时，u32BindParam 表示低时延值，单位 ms;
- 当 eBindType = E_MI_SYS_BIND_TYPE_HW_RING 时，u32BindParam 表示 ring buffer depth，目前是只有 vpe 和 venc（h264/h265）支持这种模式，只支持一路；
- 当 eBindType = E_MI_SYS_BIND_TYPE_REALTIME 时，u32BindParam 未使用，jpe imi 会走这种模式，只支持一路；
- 当 eBindType = E_MI_SYS_BIND_TYPE_FRAME_BASE 时，u32BindParam 未使用，默认是走这种 frame mode；
- E_MI_SYS_BIND_TYPE_HW_AUTOSYNC 暂时未使用。

➢ **举例**

a. vpe 与 venc 连接方式为 E_MI_SYS_BIND_TYPE_FRAME_BASE，代码如下：

```
stSrcChnPort.eModId = E_MI_MODULE_ID_VPE;
stSrcChnPort.u32DevId = 0;
stSrcChnPort.u32ChnId = 0;
stSrcChnPort.u32PortId = 0;
stDstChnPort.eModId = E_MI_MODULE_ID_VENC;
stDstChnPort.u32DevId = 0;
stDstChnPort.u32ChnId = 0;
stDstChnPort.u32PortId = 0;
u32SrcFrmrate = 30;
u32DstFrmrate = 30;
eBindType = E_MI_SYS_BIND_TYPE_FRAME_BASE;
u32BindParam = 0;
MI_SYS_BindChnPort2(&stSrcChnPort, &stDstChnPort, u32SrcFrmrate, u32DstFrmrate, eBindType,
u32BindParam);
```

b. vpe 与 jpe 连接方式为 E_MI_SYS_BIND_TYPE_REALTIME，代码如下：

```
stSrcChnPort.eModId = E_MI_MODULE_ID_VPE;
stSrcChnPort.u32DevId = 0;
stSrcChnPort.u32ChnId = 0;
stSrcChnPort.u32PortId = 0;
stDstChnPort.eModId = E_MI_MODULE_ID_VENC;
stDstChnPort.u32DevId = 1;
stDstChnPort.u32ChnId = 0;
stDstChnPort.u32PortId = 0;
u32SrcFrmrate = 30;
u32DstFrmrate = 30;
eBindType = E_MI_SYS_BIND_TYPE_REALTIME;
u32BindParam = 0;
MI_SYS_BindChnPort2(&stSrcChnPort, &stDstChnPort, u32SrcFrmrate, u32DstFrmrate, eBindType,
```

u32BindParam);

c. vpe 与 venc 连接方式为 E_MI_SYS_BIND_TYPE_HW_RING，代码如下：

```
stSrcChnPort.eModId = E_MI_MODULE_ID_VPE;
stSrcChnPort.u32DevId = 0;
stSrcChnPort.u32ChnId = 0;
stSrcChnPort.u32PortId = 0;
stDstChnPort.eModId = E_MI_MODULE_ID_VENC;
stDstChnPort.u32DevId = 0;
stDstChnPort.u32ChnId = 0;
stDstChnPort.u32PortId = 0;
u32SrcFrmrate = 30;
u32DstFrmrate = 30;
eBindType = E_MI_SYS_BIND_TYPE_HW_RING;
u32BindParam = 1080;//假设 vpe output resolution 为 1920*1080，设置 ring buffer depth 为 1080
MI_SYS_BindChnPort2(&stSrcChnPort, &stDstChnPort, u32SrcFrmrate, u32DstFrmrate, eBindType,
u32BindParam);
```

➢ **相关主题**

MI_SYS_UnBind_ChnPort。

1.2.34 MI_SYS_ConfigPrivateMMAPool

➢ **功能**

为模块配置私有 MMA Heap.

➢ **语法**

```
MI_S32 MI_SYS_ConfigPrivateMMAPool(MI_SYS_GlobalPrivPoolConfig_t *pstGlobalPrivPoolConf);
```

➢ **形参**

参数名称	参数含义	输入/输出
pstGlobalPrivPoolConf	配置私有 Mma Pool 结构体指针	输入

➢ **返回值**

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ **依赖**

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ **注意**

- 当 pstGlobalPrivPoolConf->eConfigType 为 E_MI_SYS_VPE_TO_VENC_PRIVATE_RING_POOL 时，需要设置 stPre2Vpe2VencRingPoolConfig,为 vpe 与 venc 设置私有 ring heap pool
- 当 pstGlobalPrivPoolConf->eConfigType 为 E_MI_SYS_PRE_CHN_PRIVATE_POOL 时，需要设置 stPreChnPrivPoolConfig,为模块通道设置私有 heap pool
- 当 pstGlobalPrivPoolConf->eConfigType 为 E_MI_SYS_PRE_DEV_PRIVATE_POOL 时，需要设置 stPreDevPrivPoolConfig,为模块设备设置私有 heap pool
- 设备私有 MMA Heap 与通道私有 MMA Heap 不能共存
- 建议在 MI_SYS_Init 后就为各模块创建好私有 MMA heap
- 当 pstGlobalPrivPoolConf->bCreate 为 TRUE 时，创建私有 POOL，为 FALSE 时，销毁私有 POOL

➢ **举例**

a. IPC 下两路流，一路 Main Stream H265 最大分辨率 1920*1080,一路 Sub Stream H264 最大分辨率 720*576 VENC 创建 MMA Heap:

```
Main Sream:
//为通道 1 创建大小为 38745760 的私有 MMA heap
MI_SYS_GlobalPrivPoolConfig_t stConfig;
Memset(&stConfig , 0 ,sizeof(stConfig));
stConfig. eConfigType = E_MI_SYS_PER_CHN_PRIVATE_POOL;
stConfig.bCreate = TRUE;
stConfig. uConfig. stPreChnPrivPoolConfig. eModule = E_MI_MODULE_ID_VENC;
stConfig. uConfig. stPreChnPrivPoolConfig. u32Devid =  0;
stConfig. uConfig. stPreChnPrivPoolConfig. u32Channel= 1;
stConfig. uConfig. stPreChnPrivPoolConfig. u32PrivateHeapSize = 38745760;
MI_SYS_ConfigDevChnPrivateMMAHeap(&stConfig);
Sub Stream:
//为通道 2 创建大小为 805152 的私有 MMA Heap
stConfig. eConfigType = E_MI_SYS_PER_CHN_PRIVATE_POOL;
stConfig.bCreate = TRUE;
stConfig. uConfig. stPreChnPrivPoolConfig. eModule = E_MI_MODULE_ID_VENC;
stConfig. uConfig. stPreChnPrivPoolConfig. u32Devid =  0;
stConfig. uConfig. stPreChnPrivPoolConfig. u32Channel= 2;
stConfig. uConfig. stPreChnPrivPoolConfig. u32PrivateHeapSize = 805152;
MI_SYS_ConfigDevChnPrivateMMAHeap(&stConfig);
```

VENC 销毁 MMA Heap:

```
Main Sream:
//销毁通道 1 的私有 MMA heap
stConfig. eConfigType = E_MI_SYS_PER_CHN_PRIVATE_POOL;
stConfig.bCreate = FALSE;
stConfig. uConfig. stPreChnPrivPoolConfig. eModule = E_MI_MODULE_ID_VENC;
stConfig. uConfig. stPreChnPrivPoolConfig. u32Devid =  0;
stConfig. uConfig. stPreChnPrivPoolConfig. u32Channel= 1;
MI_SYS_ConfigDevChnPrivateMMAHeap(&stConfig);
Sub Stream:
```

```
//销毁通道 2 的私有 MMA Heap
stConfig. eConfigType = E_MI_SYS_PER_CHN_PRIVATE_POOL;
stConfig.bCreate = FALSE;
stConfig. uConfig. stPreChnPrivPoolConfig. eModule = E_MI_MODULE_ID_VENC;
stConfig. uConfig. stPreChnPrivPoolConfig. u32Devid =  0;
stConfig. uConfig. stPreChnPrivPoolConfig. u32Channel= 2;
MI_SYS_ConfigDevChnPrivateMMAHeap(&stConfig);
```

VPE 创建 MMA Heap:

```
//为设备 0 创建大小为 0x4f9200 的私有 MMA Heap
stConfig. eConfigType = E_MI_SYS_PER_DEV_PRIVATE_POOL;
stConfig.bCreate = TRUE;
stConfig. uConfig. stPreDevPrivPoolConfig. eModule = E_MI_MODULE_ID_VPE;
stConfig. uConfig. stPreDevPrivPoolConfig. u32Devid =  0;
MI_SYS_ConfigDevChnPrivateMMAHeap(&stConfig);
```

VPE 销毁 MMA Heap:

```
//销毁设备 0 的私有 MMA Heap
stConfig. eConfigType = E_MI_SYS_PER_DEV_PRIVATE_POOL;
stConfig.bCreate = FALSE;
stConfig. uConfig. stPreDevPrivPoolConfig. eModule = E_MI_MODULE_ID_VPE;
stConfig. uConfig. stPreChnPrivPoolConfig. u32Devid =  0;
MI_SYS_ConfigDevChnPrivateMMAHeap(&stConfig);
```

创建 VPE 与 VENC 之间的 ring MMA Heap

```
stConfig. eConfigType = E_MI_SYS_VPE_TO_VENC_PRIVATE_RING_POOL;
stConfig.bCreate = TRUE;
stConfig. uConfig. stPreVpe2VencRingPrivPoolConfig.u32VencInputRingPoolStaticSize =  8*1024*1024;
MI_SYS_ConfigDevChnPrivateMMAHeap(&stConfig);
```

销毁 VPE 与 VENC 之间的 ring MMA Heap

```
stConfig. eConfigType = E_MI_SYS_VPE_TO_VENC_PRIVATE_RING_POOL;
stConfig.bCreate = FALSE;
MI_SYS_ConfigDevChnPrivateMMAHeap(&stConfig);
```

不同分辨率或开启不同功能 size 大小都不一样，详细参数请通过 venc_memory_calculator.xlsx/3dnr_rot memory usage calculation.xls 表中获取。

1.2.35 MI_SYS_MemsetPa

- **功能**

  通过 DMA 硬件模块，填充整块物理内存.

- **语法**

```
MI_S32 MI_SYS_MemsetPa(MI_PHY phyPa, MI_U32 u32Val, MI_U32 u32Lenth);
```

➢ 形参

参数名称	参数含义	输入/输出
phyPa	填充的物理地址	输入
u32Val	填充值	输入
u32Lenth	填充大小，单位为 byte	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

1.2.36 MI_SYS_MemcpyPa

➢ 功能

通过 DMA 硬件模块，把源内存数据拷贝到目标内存上.

➢ 语法

```
MI_S32 MI_SYS_MemcpyPa(MI_PHY phyDst, MI_PHY phySrc, MI_U32 u32Lenth);
```

➢ 形参

参数名称	参数含义	输入/输出
phyDst	目的物理地址	输入
phySrc	源物理地址	输入
u32Lenth	拷贝大小，单位为 byte	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

1.2.37 MI_SYS_BufFillPa

➢ 功能

通过 DMA 硬件模块，填充部分物理内存.

➢ 语法

MI_S32 MI_SYS_BufFillPa(MI_SYS_FrameData_t *pstBuf, MI_U32 u32Val, MI_SYS_WindowRect_t *pstRect);

➢ 形参

参数名称	参数含义	输入/输出
pstBuf	填充的帧数据描述之结构体	输入
u32Val	填充值	输入
pstRect	填充的数据范围	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

- pstRect 数据范围是以 pstBuf 所描述的首地址为（0，0）点开始，宽高都是以 pstBuf 中的 ePixelFormat 来计算每一个 pixel 所移动的内存数据大小来进行部分填充。
- pstBuf 中 u16Width、u16Height、phyAddr、u32Stride、ePixelFormat 为必填，其余数值无意义。

1.2.38 MI_SYS_BufBlitPa

➢ 功能

通过 DMA 硬件模块，把源内存数据上的部分区域拷贝到目标内存上的部分区域.

➢ 语法

MI_S32 MI_SYS_BufBlitPa(MI_SYS_FrameData_t *pstDstBuf, MI_SYS_WindowRect_t *pstDstRect, MI_SYS_FrameData_t *pstSrcBuf, MI_SYS_WindowRect_t *pstSrcRect);

➢ 形参

参数名称	参数含义	输入/输出
pstDstBuf	目标内存物理首地址	输入
pstDstRect	目标内存拷贝的区域	输入
pstSrcBuf	源内存物理首地址	输入
pstSrcRect	源内存拷贝的区域	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件：mi_sys_datatype.h、mi_sys.h
- 库文件：libmi_sys.a

※ 注意

- pstDstRect/pstSrcRect 数据范围是以 pstDstBuf/pstSrcBuf 所描述的首地址为（0，0）点开始，宽高都是以其中的 ePixelFormat 来计算每一个 pixel 所移动的内存数据大小来进行部分填充。
- pstDstBuf/pstSrcBuf 中 u16Width、u16Height、phyAddr、u32Stride、ePixelFormat 为必填，其余数值无意义。
- 源内存或者目标内存的区域部分超过了其本来的范围，则只会拷贝其未超过部分上的数据。

2. 数据类型

相关数据类型、数据结构定义如下：

MI_ModuleId_e	定义模块 ID 枚举类型
MI_SYS_PixelFormat_e	定义像素枚举类型
MI_SYS_CompressMode_e	定义压缩方式枚举类型
MI_SYS_FrameTileMode_e	定义 Tile 格式枚举类型
MI_SYS_FieldType_e	定义 Field 枚举类型
MI_SYS_BufDataType_e	定义模块 ID 枚举类型
MI_SYS_ChnPort_t	定义模块设备通道结构体
MI_SYS_PerFrameMetaBuf_t	定义码流 MetaData 的结构体
MI_SYS_RawData_t	定义码流 RawData 的结构体
MI_SYS_WindowRect_t	定义 Window 坐标的结构体
MI_SYS_FrameData_t	定义码流 FrameData 的结构体
MI_SYS_BufInfo_t	Buf 信息结构体
MI_SYS_BufFrameConfig_t	Frame buf 配置信息结构体
MI_SYS_BufRawConfig_t	Raw buf 配置信息结构体
MI_SYS_BufConf_t	配置 Buf 信息结构体
MI_SYS_Version_t	Sys 版本信息结构体
MI_VB_PoolListConf_t	描述 VB Pool 链表信息的结构体
MI_SYS_BindType_e	定义前后级工作模式的枚举类型
MI_SYS_FrameData_PhySignalType	描述 frame data 隶属的 buffer 类型的枚举类型

2.1. MI_ModuleId_e

- 说明

  定义模块 ID 枚举类型。

- 定义

```
typedef enum
{
    E_MI_MODULE_ID_IVE      = 0,
    E_MI_MODULE_ID_VDF      = 1,
    E_MI_MODULE_ID_VENC     = 2,
    E_MI_MODULE_ID_RGN      = 3,
    E_MI_MODULE_ID_AI       = 4,
    E_MI_MODULE_ID_AO       = 5,
    E_MI_MODULE_ID_VIF      = 6,
    E_MI_MODULE_ID_VPE      = 7,
    E_MI_MODULE_ID_VDEC     = 8,
    E_MI_MODULE_ID_SYS      = 9,
    E_MI_MODULE_ID_FB       = 10,
    E_MI_MODULE_ID_HDMI     = 11,
    E_MI_MODULE_ID_DIVP     = 12,
    E_MI_MODULE_ID_GFX      = 13,
    E_MI_MODULE_ID_VDISP    = 14,
    E_MI_MODULE_ID_DISP     = 15,
    E_MI_MODULE_ID_OS       = 16,
    E_MI_MODULE_ID_IAE = 17,
    E_MI_MODULE_ID_MD = 18,
    E_MI_MODULE_ID_OD = 19,
    E_MI_MODULE_ID_SHADOW = 20,
    E_MI_MODULE_ID_WARP = 21,
  E_MI_MODULE_ID_ALSA = 22,
  E_MI_MODULE_ID_LDC = 23,
  E_MI_MODULE_ID_SD = 24,
    E_MI_MODULE_ID_PANEL = 25,
  E_MI_MODULE_ID_CIPHER = 26,
  E_MI_MODULE_ID_SNR = 27,
    E_MI_MODULE_ID_WLAN =28,
    E_MI_MODULE_ID_DMA = 29,
    E_MI_MODULE_ID_MAX,
} MI_ModuleId_e;
```

➢ 成员

成员名称	描述
E_MI_ MODULE_ ID_VDEC	视频处理 VPE 的模块 ID
E_MI_ MODULE_ ID_VENC	视频编码模块 VPE 的模块 ID
E_MI_ MODULE_ ID_DISP	视频显示模块 DISP 的模块 ID
E_MI_ MODULE_ ID_VIF	视频输入 VIF 的模块 ID
E_MI_ MODULE_ ID_AI	音频输入模块 AI 的模块 ID
E_MI_ MODULE_ ID_AO	音频输出模块 AI 的模块 ID
E_MI_ MODULE_ ID_RGN	OSD 叠加和遮挡模块 REG 的模块 ID
E_MI_ MODULE_ ID_VPE	视频图像处理模块 VPE 的模块 ID
E_MI_ MODULE_ ID_DIVP	视频 DI 及后处理模块 DIVP 的模块 ID
E_MI_ MODULE_ ID_GFX	2D 图形处理加速模块 GFX 的模块 ID
E_MI_ MODULE_ ID_IVE	图像智能算子 IVE 的模块 ID
E_MI_ MODULE_ ID_IAE	音频智能算子 IAE 的模块 ID
E_MI_ MODULE_ ID_MD	移动侦测模块 MD 的模块 ID
E_MI_ MODULE_ ID_OD	遮挡侦测模块 OD 的模块 ID
E_MI_ MODULE_ ID_VDF	视频智能算法框架模块 VDF 的模块 ID
E_MI_MODULE_ID_VDISP	图像拼图模块 VDISP 的模块 ID
E_MI_ MODULE_ ID_FB	MStar UI 显示 FrameBuffer Device 模块的模块 ID

※ 注意事项

无。

➢ 相关数据接口及类型

无

2.2. MI_SYS_PixelFormat_e

➢ 说明

定义像素枚举类型。

➢ 定义

```
typedef enum
{
    E_MI_SYS_PIXEL_FRAME_YUV422_YUYV = 0,
    E_MI_SYS_PIXEL_FRAME_ARGB8888,
    E_MI_SYS_PIXEL_FRAME_ABGR8888,
    E_MI_SYS_PIXEL_FRAME_BGRA8888,

    E_MI_SYS_PIXEL_FRAME_RGB565,
```

```
    E_MI_SYS_PIXEL_FRAME_ARGB1555,
    E_MI_SYS_PIXEL_FRAME_ARGB4444,
    E_MI_SYS_PIXEL_FRAME_I2,
    E_MI_SYS_PIXEL_FRAME_I4,
    E_MI_SYS_PIXEL_FRAME_I8,

    E_MI_SYS_PIXEL_FRAME_YUV_SEMIPLANAR_422,
    E_MI_SYS_PIXEL_FRAME_YUV_SEMIPLANAR_420,
    E_MI_SYS_PIXEL_FRAME_YUV_MST_420,

    //vdec mstar private video format
    E_MI_SYS_PIXEL_FRAME_YC420_MSTTILE1_H264,
    E_MI_SYS_PIXEL_FRAME_YC420_MSTTILE2_H265,
    E_MI_SYS_PIXEL_FRAME_YC420_MSTTILE3_H265 = 15,

    E_MI_SYS_PIXEL_FRAME_RGB_BAYER_BASE,
    E_MI_SYS_PIXEL_FRAME_RGB_BAYER_NUM = \
        E_MI_SYS_PIXEL_FRAME_RGB_BAYER_BASE + \
        E_MI_SYS_DATA_PRECISION_MAX* \
        E_MI_SYS_PIXEL_BAYERID_MAX-1,

    E_MI_SYS_PIXEL_FRAME_FORMAT_MAX,

} MI_SYS_PixelFormat_e;
```

- 成员

成员名称	描述
E_MI_SYS_PIXEL_FRAME_YUV422_YUYV	YUV422_YUYV 格式
E_MI_SYS_PIXEL_FRAME_ARGB8888	ARGB8888 格式
E_MI_SYS_PIXEL_FRAME_ABGR8888	ABGR8888 格式
E_MI_SYS_PIXEL_FRAME_BGRA8888	BGRA8888 格式
E_MI_SYS_PIXEL_FRAME_RGB565	RGB565 格式
E_MI_SYS_PIXEL_FRAME_ARGB1555	ARGB1555 格式
E_MI_SYS_PIXEL_FRAME_ARGB4444	ARGB4444 格式
E_MI_SYS_PIXEL_FRAME_I2	2 bpp 格式
E_MI_SYS_PIXEL_FRAME_I4	4 bpp 格式
E_MI_SYS_PIXEL_FRAME_I8	8 bpp 格式
E_MI_SYS_PIXEL_FRAME_YUV_SEMIPLANAR_422	YUV422 semi-planar 格式
E_MI_SYS_PIXEL_FRAME_YUV_SEMIPLANAR_420	YUV420 格式
E_MI_SYS_PIXEL_FRAME_YUV_MST_420	YUV420 packe（YYUYYV）格式
E_MI_SYS_PIXEL_FRAME_YC420_MSTTILE1_H264	内部自定义 TILE 格式

成员名称	描述
E_MI_SYS_PIXEL_FRAME_YC420_MSTTILE2_H265	内部自定义 TILE 格式
E_MI_SYS_PIXEL_FRAME_YC420_MSTTILE3_H265	内部自定义 TILE 格式
E_MI_SYS_PIXEL_FRAME_RGB_BAYER_BASE	RGB raw data 组合格式

※ 注意事项

无。

➢ 相关数据类型及接口

无

2.3. MI_SYS_CompressMode_e

➢ 说明

定义压缩方式枚举类型。

➢ 定义

```
typedef enum
{
    E_MI_SYS_COMPRESS_MODE_NONE,//no compress
    E_MI_SYS_COMPRESS_MODE_SEG,//compress unit is 256 bytes as a segment
    E_MI_SYS_COMPRESS_MODE_LINE,//compress unit is the whole line
    E_MI_SYS_COMPRESS_MODE_FRAME,//compress unit is the whole frame
    E_MI_SYS_COMPRESS_MODE_BUTT, //number
}MI_SYS_CompressMode_e;
```

➢ 成员

成员名称	描述
E_MI_SYS_COMPRESS_MODE_NONE	非压缩的视频格式
E_MI_SYS_COMPRESS_MODE_SEG	段压缩的视频格式
E_MI_SYS_COMPRESS_MODE_LINE	行压缩的视频格式，按照一行为一段进行压缩
E_MI_SYS_COMPRESS_MODE_FRAME	帧压缩的视频格式，将一帧数据进行压缩
E_MI_SYS_COMPRESS_MODE_BUTT	视频压缩方式的数量

※ 注意事项

无。

➢ 相关数据类型及接口

无

2.4. MI_SYS_FrameTileMode_e

- 说明

  定义 Tile 格式枚举类型。

➢ 定义

```
typedef enum
{
    E_MI_SYS_FRAME_TILE_MODE_NONE = 0,
    E_MI_SYS_FRAME_TILE_MODE_16x16,     // tile mode 16x16
    E_MI_SYS_FRAME_TILE_MODE_16x32,     // tile mode 16x32
    E_MI_SYS_FRAME_TILE_MODE_32x16,     // tile mode 32x16
    E_MI_SYS_FRAME_TILE_MODE_32x32,     // tile mode 32x32
    E_MI_SYS_FRAME_TILE_MODE_MAX
} MI_SYS_FrameTileMode_e;
```

➢ 成员

成员名称	描述
E_MI_SYS_FRAME_TILE_MODE_NONE	None
E_MI_SYS_FRAME_TILE_MODE_16x16	16x16 mode
E_MI_SYS_FRAME_TILE_MODE_16x32	16x32 mode
E_MI_SYS_FRAME_TILE_MODE_32x16	32x16 mode
E_MI_SYS_FRAME_TILE_MODE_32x32	32x32 mode

※ 注意事项

无。

➢ 相关数据类型及接口

无

2.5. MI_SYS_FieldType_e

➢ 说明

定义枚举类型。

➢ 定义

```
typedef enum
{
    E_MI_SYS_FIELDTYPE_NONE,        //< no field.
    E_MI_SYS_FIELDTYPE_TOP,         //< Top field only.
    E_MI_SYS_FIELDTYPE_BOTTOM,      //< Bottom field only.
    E_MI_SYS_FIELDTYPE_BOTH,        //< Both fields.
    E_MI_SYS_FIELDTYPE_NUM
} E_MI_SYS_FieldType_e;
```

➢ 成员

成员名称	描述

※ 注意事项

无。

➢ 相关数据类型及接口

无

2.6. MI_SYS_BufDataType_e

➢ 说明

定义模块 ID 枚举类型。

➢ 定义

```
typedef enum
{
    E_MI_SYS_BUFDATA_RAW = 0,
    E_MI_SYS_BUFDATA_FRAME,
} MI_SYS_BufDataType_e;
```

➢ 成员

成员名称	描述
E_MI_SYS_BUFDATA_RAW	Raw 数据类型
E_MI_SYS_BUFDATA_FRAME	Frame 数据类型

※ 注意事项

无。

➢ 相关数据类型及接口

无

2.7. MI_SYS_FrameIspInfoType_e

➢ 说明

定义 frame data 携带的 ISP info 枚举类型。

➢ 定义

```
typedef enum
{
    E_MI_SYS_FRAME_ISP_INFO_TYPE_NONE,
    E_MI_SYS_FRAME_ISP_INFO_TYPE_GLOBAL_GRADIENT
}MI_SYS_FrameIspInfoType_e;
```

➢ 成员

成员名称	描述
E_MI_SYS_FRAME_ISP_INFO_TYPE_NONE	NONE
E_MI_SYS_FRAME_ISP_INFO_TYPE_GLOBAL_GRADIENT	ISP 的全局梯度数据类型

※ 注意事项

无。

➢ 相关数据类型及接口

无

2.8. MI_SYS_ChnPort_t

➢ 说明

定义模块设备通道结构体。

➢ 定义

```
typedef struct MI_SYS_ChnPort_s
{
    MI_ModuleId_e eModId;
    MI_U32    u32DevId;
    MI_U32    u32ChnId;
    MI_U32 u32PortId;
}MI_SYS_ChnPort_t;
```

➢ 成员

成员名称	描述
eModId	模块号
u32DevId	设备号
u32ChnId	通道号
u32PortId	端口号

※ 注意事项

无。

➢ 相关数据类型及接口

无

2.9. MI_SYS_PerFrameMetaBuf_t

➢ 说明

定义码流 MetaData 的结构体。

➢ 定义

```
typedef struct MI_SYS_PerFrameMetaBuf_s
{
    void* pVirAddr;                 /* 虚拟地址 */
    MI_PHY      phyAddr;            /* 物理地址 */

    MI_U32 u32Size;
    MI_U32 u32ExtraData;      /*driver special flag*/
    MI_ModuleId_e eDataFromModule;
} MI_SYS_PerFrameMetaBuf_t;
```

➢ 成员

成员名称	描述
pVirAddr	数据存放虚拟地址。
phyAddr	数据存放物理地址
u32Size	数据大小
u32ExtraData	driver special flag
eDataFromModule	来自哪个模块的数据

※ 注意事项

无

➢ 相关数据类型及接口

无。

2.10. MI_SYS_RawData_t

➢ 说明

定义码流 RawData 的结构体。

➢ 定义

```
typedef struct MI_SYS_RawData_s
{
   void*     pVirAddr;   /* 码流地址 */
   MI_PHY    phyAddr; /* 物理地址 */
   MI_U32    u32BufSize;
   MI_U32    u32ContentSize;
```

```
    MI_BOOL bEndOfFrame;      /* 当前帧是否结束 */
    MI_U64   u64SeqNum;       /* 帧序号 */
} MI_SYS_RawData_t;
```

➢ 成员

成员名称	描述
pVirAddr	码流包的地址。
phyAddr	码流包的物理地址
u32BufSize	Buf 大小
u32ContentSize	数据实际占用 buf 大小
bEndOfFrame	当前帧是否结束。（预留参数）当前只支持帧模式下按帧传送。
u64SeqNum	当前帧的帧序号

※ 注意事项

当前只支持按帧传送数据，每次需完整传送一帧数据。

码流帧数据附带 PTS 时，解码后输出数据输出相同 PTS。当 PTS=-1 时，不参考系统时钟输出数据帧。

➢ 相关数据类型及接口

无。

2.11. MI_SYS_WindowRect_t

➢ 说明

定义 window 坐标的结构体。

➢ 定义

```
typedef struct MI_SYS_WindowRect_s
{
    MI_U16 u16X;
    MI_U16 u16Y;
    MI_U16 u16Width;
    MI_U16 u16Height;
}MI_SYS_WindowRect_t;
```

➢ 成员

成员名称	描述
u16X	window 起始位置的水平方向的值。
u16Y	window 起始位置的垂直方向的值。
u16Width	window 宽度。
u16Height	window 高度。

※ 注意事项

无

➢ 相关数据类型及接口

无。

2.12. MI_SYS_FrameData_t

➢ 说明

定义码流 FrameData 的结构体。

➢ 定义

```
typedef    struct    MI_SYS_FrameData_s
{
    MI_SYS_FrameTileMode_e eTileMode;
    MI_SYS_PixelFormat_e ePixelFormat;
    MI_SYS_CompressMode_e eCompressMode;
    MI_SYS_FrameScanMode_e eFrameScanMode;
    MI_SYS_FieldType_e eFieldType;
     MI_SYS_FrameData_PhySignalType ePhylayoutType;

    MI_U16 u16Width;
    MI_U16 u16Height;

     void* pVirAddr[3];
     MI_PHY phyAddr[3];//notice that this is miu bus addr,not cpu bus addr.
     MI_U32 u32Stride[3];
     MI_U32 u32BufSize;//total size that allocated for this buffer,include consider alignment.

     MI_U16 u16RingBufStartLine;//Valid in case RINGBUF_FRAME_DATA,    u16RingBufStartLine must
                                        be LGE than 0 and less than u16Height
     MI_U16 u16RingBufRealTotalHeight;///Valid in case RINGBUF_FRAME_DATA,
                                        u16RingBufStartLine must be LGE than u16Height
     MI_SYS_FrameIspInfo_t stFrameIspInfo;//isp info of each frame
} MI_SYS_FrameData_t;
```

- 成员

成员名称	描述
eTileMode	Tile 模式
ePixelFormat	像素格式
eCompressMode	压缩格式
eFrameScanMode	Frame scan 模块
eFieldType	File 类型
ePhylayoutType	frame data 隶属的 buffer 的类型
u16Width	Frame 宽度
u16Height	Frame 高度
pVirAddr	虚拟地址
phyAddr	物理地址
u32Stride	图片每行所占字节数
u32BufSize	Sys 分配给当前 frame 的实际 buf 大小
u16RingBufStartLine	ring mode 时，frame data 开始于 ring buffer 的行数
u16RingBufRealTotalHeight	ring mode 时，frame data 的真实高度
stFrameIspInfo	ISP info struct

- 注意事项

  无

- 相关数据类型及接口

  无

2.13. MI_SYS_BufInfo_t

- 说明

  定义码流 FrameData 的结构体。

- 定义

```
typedef  struct  MI_SYS_BufInfo_s
{
    MI_U64 u64Pts;
    MI_U64 u64SidebandMsg;
    MI_SYS_BufDataType_e eBufType;
    MI_BOOL bEndOfStream;
    MI_BOOL bUsrBuf;
    union
    {
        MI_SYS_FrameData_t stFrameData;
        MI_SYS_RawData_t stRawData;
```

```
    };
MI_SYS_PerFrameMetaBuf_t stPerFrameMetaBuf;
} MI_SYS_BufInfo_t;
```

➢ 成员

成员名称	描述
eBufType	Buf 类型
u64Pts	时间戳
bEndOfStream	是否已发完所有信息

※ 注意事项

无

➢ 相关数据类型及接口

无。

2.14. MI_SYS_BufFrameConfig_t

➢ 说明

定义码流 FrameData 的结构体。

➢ 定义

```
typedef struct MI_SYS_BufFrameConfig_s
{
    MI_U16 u16Width;
    MI_U16 u16Height;
    MI_SYS_FrameScanMode_e eFrameScanMode;//
    MI_SYS_PixelFormat_e eFormat;
     MI_SYS_FrameBufExtraConfig_t stFrameBufExtraConf;//set by MI_SYS internal
}MI_SYS_BufFrameConfig_t;
```

➢ 成员

成员名称	描述
u16Width	Frame 宽度
u16Height	Frame 高度
eFrameScanMode	Frame Scan Mode
eFormat	Frame 像素格式
stFrameBufExtraConf	Frame 对齐的 pixel 大小（无需用户设置）

※ 注意事项

无

➢ 相关数据类型及接口

无。

2.15. MI_SYS_BufRaw_Config_t

➢ 说明

定义码流 Port Buf 的结构体。

➢ 定义

```
typedef struct MI_SYS_BufRawConfig_s
{
    MI_U32 u32Size;
}MI_SYS_BufRawConfig_t;
```

➢ 成员

成员名称	描述
u32Size	Buf 大小

※ 注意事项

无

➢ 相关数据类型及接口

无。

2.16. MI_SYS_BufConf_t

➢ 说明

定义码流 Port Buf 的结构体。

➢ 定义

```
typedef struct MI_SYS_BufConf_s
{
    MI_SYS_BufDataType_e eBufType;
    MI_U32 u32Flags;     //0 or MI_SYS_MAP_VA
    MI_U64 u64TargetPts;
    union
    {
        MI_SYS_BufFrameConfig_t stFrameCfg;
        MI_SYS_BufRawConfig_t stRawCfg;
    };
}MI_SYS_BufConf_t;
```

➢ 成员

成员名称	描述
eBufType	Buf type
u32Flags	Buf 是否 map kernel space va

※ 注意事项

➢ 相关数据类型及接口

无。

2.17. MI_SYS_Version_t

➢ 说明

定义 Sys 版本信息结构体。

➢ 定义

```
typedef struct MI_SYS_Version_s
{
    MI_U8 u8Version[128];
}MI_SYS_Version_t;
```

➢ 成员

成员名称	描述
u8Version[128]	描述 sys 版本信息的字符串 buf

※ 注意事项

➢ 相关数据类型及接口

无。

2.18. MI_VB_PoolListConf_t

➢ 说明

定义描述 VB Pool 链表信息的结构体。

➢ 定义

```
typedef struct MI_VB_PoolListConf_s
{
    MI_U32 u32PoolListCnt;
    MI_VB_PoolConf_t stPoolConf[MI_VB_POOL_LIST_MAX_CNT];
} MI_VB_PoolListConf_t;
```

➢ 成员

成员名称	描述
u32PoolListCnt	VB pool 链表成员数量
stPoolConf	VB pool 链表成员配置信息

※ 注意事项

➢ 相关数据类型及接口

无。

2.19. MI_SYS_BindType_e

➢ 说明

定义前后级工作模式。

➢ 定义

```
typedef enum
{
    E_MI_SYS_BIND_TYPE_FRAME_BASE = 0x00000001,
    E_MI_SYS_BIND_TYPE_SW_LOW_LATENCY = 0x00000002,
    E_MI_SYS_BIND_TYPE_REALTIME = 0x00000004,
    E_MI_SYS_BIND_TYPE_HW_AUTOSYNC = 0x00000008,
    E_MI_SYS_BIND_TYPE_HW_RING = 0x00000010
```

```
}MI_SYS_BindType_e;
```

➢ 成员

成员名称	描述
E_MI_SYS_BIND_TYPE_FRAME_BASE	frame mode，默认工作方式
E_MI_SYS_BIND_TYPE_SW_LOW_LATENCY	低时延工作方式
E_MI_SYS_BIND_TYPE_REALTIME	硬件直连工作方式
E_MI_SYS_BIND_TYPE_HW_AUTOSYNC	前后级 handshake，buffer size 与图像分辨率一致
E_MI_SYS_BIND_TYPE_HW_RING	前后级 handshake，ring buffer depth 可以调小于图像分辨率高

※ 注意事项

旧版本 MI SYS 不提供此功能，如没有找到，不用设置。

➢ 相关数据类型及接口

无

2.20. MI_SYS_FrameData_PhySignalType

➢ 说明

描述 frame data 隶属的 buffer 类型的枚举类型。

➢ 定义

```
typedef enum
{
    REALTIME_FRAME_DATA,
    RINGBUF_FRAME_DATA,
    NORMAL_FRAME_DATA,
}MI_SYS_FrameData_PhySignalType;
```

➢ 成员

成员名称	描述
REALTIME_FRAME_DATA	以 E_MI_SYS_BIND_TYPE_REALTIME 模式产生的 frame data
RINGBUF_FRAME_DATA	以 E_MI_SYS_BIND_TYPE_HW_RING 模式产生的 frame data
NORMAL_FRAME_DATA	以 E_MI_SYS_BIND_TYPE_FRAME_BASE 模式产生的 frame data

※ 注意事项

旧版本 MI SYS 不提供此功能，如没有找到，不用设置。

➢ 相关数据类型及接口

无

2.21. MI_SYS_InsidePrivatePoolType_e

➢ 说明

描述创建的私有 MMA POOL 类型的枚举类型。

➢ 定义

```
typedef enum
{
    E_MI_SYS_VPE_TO_VENC_PRIVATE_RING_POOL = 0,
    E_MI_SYS_PER_CHN_PRIVATE_POOL=1,
    E_MI_SYS_PER_DEV_PRIVATE_POOL=2,
    E_MI_SYS_PER_CHN_PORT_OUTPUT_POOL=3,
}MI_SYS_InsidePrivatePoolType_e;
```

➢ 成员

成员名称	描述
E_MI_SYS_VPE_TO_VENC_PRIVATE_RING_POOL	VPE 与 VENC 的私有 Ring Pool 类型
E_MI_SYS_PER_CHN_PRIVATE_POOL	通道私有 Pool 类型
E_MI_SYS_PER_DEV_PRIVATE_POOL	设备私有 Pool 类型
E_MI_SYS_PER_CHN_PORT_OUTPUT_POOL	输出端口私有 Pool 类型

※ 注意事项

旧版本 MI SYS 不提供此功能，如没有找到，不用设置。

➢ 相关数据类型及接口

无

2.22. MI_SYS_PerChnPrivHeapConf_t

➢ 说明

定义描述通道私有 MMA Pool 的结构体。

➢ 定义

```
typedef struct MI_PerChnPrivHeapConf_s
{
    MI_ModuleId_e eModule;
    MI_U32 u32Devid;
    MI_U32 u32Channel;
```

```
    MI_U8 u8MMAHeapName[MI_MAX_MMA_HEAP_LENGTH];
    MI_U32 u32PrivateHeapSize;
}MI_SYS_PerChnPrivHeapConf_t;
```

➢ 成员

成员名称	描述
eModule	模块 ID
u32Devid	设备 ID
u32Channel	通道 ID
u8MMAHeapName	Mma heap name
u32PrivateHeapSize	私有 pool 大小

※ 注意事项

➢ 相关数据类型及接口

无。

2.23. MI_SYS_PerDevPrivHeapConf_t

➢ 说明

定义描述设备私有 MMA Pool 的结构体。

➢ 定义

```
typedef struct MI_PerDevPrivHeapConf_s
{
    MI_ModuleId_e eModule;
    MI_U32 u32Devid;
    MI_U32 u32Reserve;
    MI_U8 u8MMAHeapName[MI_MAX_MMA_HEAP_LENGTH];
    MI_U32 u32PrivateHeapSize;
}MI_SYS_PerDevPrivHeapConf_t;
```

➢ 成员

成员名称	描述
eModule	模块 ID
u32Devid	设备 ID
u32Reserve	预留
u8MMAHeapName	Mma heap name
u32PrivateHeapSize	私有 pool 大小

※ 注意事项

➢ 相关数据类型及接口

无。

2.24. MI_SYS_PerVpe2VencRingPoolConf_t

➢ 说明

定义描述 VPE 与 VENC 之间的私有 Ring MMA Pool 的结构体。

➢ 定义

```
typedef struct MI_PerVpe2VenvRingPoolConf_s
{
    MI_U32 u32VencInputRingPoolStaticSize;
    MI_U8 u8MMAHeapName[MI_MAX_MMA_HEAP_LENGTH];
}MI_SYS_PerVpe2VencRingPoolConf_t;
```

➢ 成员

成员名称	描述
u32VencInputRingPoolStaticSize	私有 pool 大小
u8MMAHeapName	Mma heap name

※ 注意事项

➢ 相关数据类型及接口

无。

2.25. MI_SYS_PerChnPortOutputPool_t

➢ 说明

定义描述 ouput port 的私有 MMA Pool 的结构体。

➢ 定义

```
typedef struct MI_SYS_PerChnPortOutputPool_s
{
    MI_ModuleId_e eModule;
    MI_U32 u32Devid;
    MI_U32 u32Channel;
    MI_U32 u32Port;
    MI_U8 u8MMAHeapName[MI_MAX_MMA_HEAP_LENGTH];
    MI_U32 u32PrivateHeapSize;
}MI_SYS_PerChnPortOutputPool_t;
```

➢ 成员

成员名称	描述
eModule	模块 ID
u32Devid	设备 ID

u32Channel	通道 ID
u32Port	端口 ID
u8MMAHeapName	Mma heap name
u32PrivateHeapSize	私有 pool 大小

※ 注意事项

➢ 相关数据类型及接口

无。

2.26. MI_SYS_GlobalPrivPoolConfig_t

➢ 说明

定义描述配置私有 MMA Pool 的结构体。

➢ 定义

```
typedef struct MI_SYS_GlobalPrivPoolConfig_s
{
    MI_SYS_InsidePrivatePoolType_e eConfigType;
    MI_BOOL bCreate;
    union
    {
        MI_SYS_PerChnPrivHeapConf_t stPreChnPrivPoolConfig;
        MI_SYS_PerDevPrivHeapConf_t stPreDevPrivPoolConfig;
        MI_SYS_PerVpe2Venc_RingPoolConf_t stPreVpe2VencRingPrivPoolConfig;
        MI_SYS_PerChnPortOutputPool_t stPreChnPortOutputPrivPool;
    }uConfig;
} MI_SYS_GlobalPrivPoolConfig_t;
```

➢ 成员

成员名称	描述
eConfigType	私有 pool 类型
bCreate	是否创建私有 pool。true:创建 false:销毁
uConfig	不同类型私有 pool 的结构体

※ 注意事项

➢ 相关数据类型及接口

无。

2.27. MI_SYS_FrameIspInfo_t

➢ 说明

定义 frame data 中 ISP info 的结构体。

➢ 定义

```
typedef struct MI_SYS_FrameIspInfo_s
{
    MI_SYS_FrameIspInfoType_e eType;
    union
    {
        MI_U32 u32GlobalGradient;
    }uIspInfo;
}MI_SYS_FrameIspInfo_t;
```

➢ 成员

成员名称	描述
eType	ISP info type
uIspInfo	ISP info union

※ 注意事项

➢ 相关数据类型及接口

无。

3. 错误码

区域管理 API 错误码如表 3-1 所示。

表 3-1 区域管理 API 错误码

错误代码	宏定义	描述
0xA0038001	MI_ERR_REG_INVALID_DEVID	设备 ID 超出合法范围
0xA0038002	MI_ERR_REG_INVALID_CHNID	通道组号错误或无效区域句柄
0xA0038003	MI_ERR_REG_ILLEGAL_PARAM	参数超出合法范围
0xA0038004	MI_ERR_REG_EXIST	重复创建已存在的设备、通道或资源
0xA0038005	MI_ERR_REG_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA0038006	MI_ERR_REG_NULL_PTR	函数参数中有空指针
0xA0038007	MI_ERR_REG_NOT_CONFIG	模块没有配置
0xA0038008	MI_ERR_REG_NOT_SUPPORT	不支持的参数或者功能
0xA0038009	MI_ERR_REG_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA003800C	MI_ERR_REG_NOMEM	分配内存失败，如系统内存不足
0xA003800D	MI_ERR_REG_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA003800E	MI_ERR_REG_BUF_EMPTY	缓冲区中无数据
0xA003800F	MI_ERR_REG_BUF_FULL	缓冲区中数据满
0xA0038010	MI_ERR_REG_NOTREADY	系统没有初始化或没有加载相应模块
0xA0038011	MI_ERR_REG_BADADDR	地址非法
0xA0038012	MI_ERR_REG_BUSY	系统忙
0xA0038013	E_MI_ERR_CHN_NOT_STARTED	channel not start
0xA0038014	E_MI_ERR_CHN_NOT_STOPED	channel not stop
0xA0038015	E_MI_ERR_NOT_INIT	module not init before use it
0xA0038016	E_MI_ERR_NOT_ENABLE	device or channel not enable
0xA0038017	E_MI_ERR_FAILED	unexpected error