MI VDISP API

Version 2.06

REVISION HISTORY

Revision No.	Description	Date
2.03	• 发布版	04/12/2018
2.04	• 更新表述	
2.05	• 基于新架构的重构更新	10/29/2019
2.06	• 修正文档错误与格式	01/13/2020

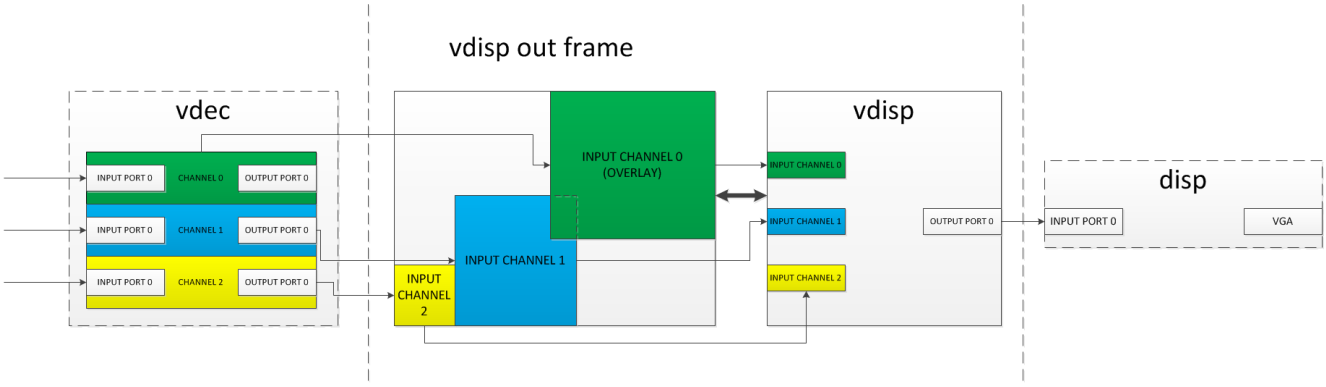
TABLE OF CONTENTS

1. 概述

1.1. 模块说明

VDISP(virtual display)模块设计用来组合多份 YUV/RGB 等颜色空间的数据成一张全幅输出，并为每一个数据输入指定一个缩略图形式的显示窗口。**特定缩略图窗口支持区域重叠。**

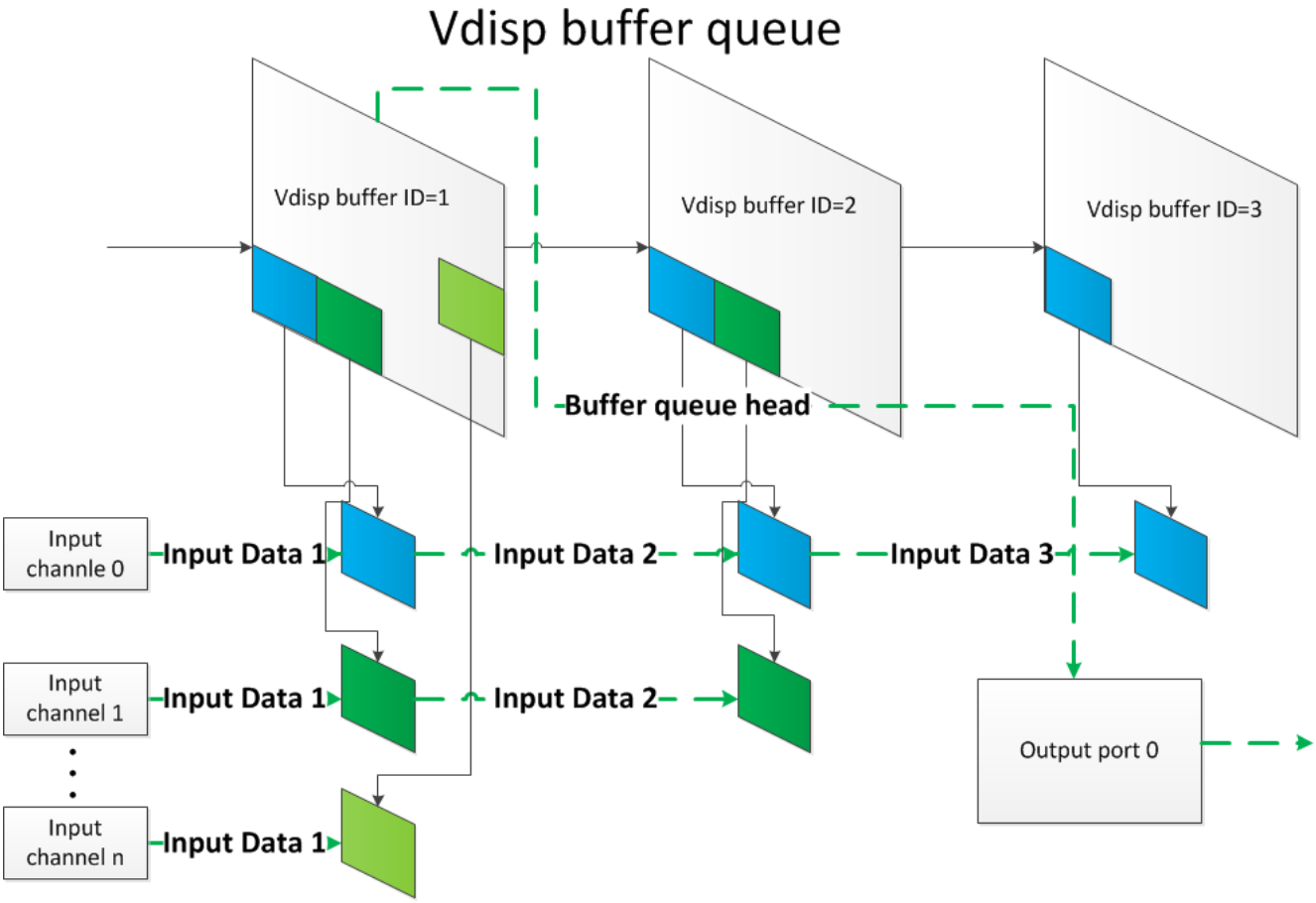
一个典型的 VDISP 应用场景：



- vdec 采用 3 路 channel 输出 3 路数据给 VDISP 做拼图，绿色通道，黄色通道和蓝色通道，普通通道预览窗口不支持色通道为 overlay 通道，overlay 通道会叠加到所有普通通道之上
- vdisp 接收 3 路 vdec 数据，2 路普通通道(黄色，蓝色)，一路 overlay 通道(绿色)，做拼图，完成之后给 disp 输出到 VGA
- disp 接收 vdisp 的数据，用 VGA 信号输出

1.2. 模块框图

VDISP 特别之处在于 Input/Output Buffer 是同一块，前一级模块通过 Stride Write 的方式直接写入 Output Buffer 的对应位置来实现最终的拼接效果，理论上可以实现零内存拷贝。



1.3. 约束

a) 输入通道的起始横坐标(u32OutX)有对齐的需求，但是对齐的具体值取决于接入 VDISP 的前端模块，因为硬件元件读写内存通常有地址的对齐需求，否则会引起最终输出花边等异常。通常情况下至少需要 8 对齐，推荐 16。

b) 目前支持 2 种输入数据格式:

   a) E_MI_SYS_PIXEL_FRAME_YUV_SEMIPLANAR_420

   b) E_MI_SYS_PIXEL_FRAME_YUV_SEMIPLANAR_422

c) 目前只支持固定帧率输出:参见 MI_VDISP_OutputPortAttr_t 的参数 u32FrmRate

d) VDISP 模块目前不支持:**格式转换/裁剪/缩放**等功能。

e) 参见更多的规格参数

2. API 参考

该功能模块提供以下 API:

API 名	功能
MI_VDISP_Init	模块初始化
MI_VDISP_Exit	模块去初始化
MI_VDISP_OpenDevice	打开一个 VDISP 虚拟设备
MI_VDISP_CloseDevice	关闭一个 VDISP 虚拟设备
MI_VDISP_SetOutputPortAttr	设置输出 port 属性
MI_VDISP_GetOutputPortAttr	获取输出 port 属性
MI_VDISP_SetInputChannelAttr	设置输入通道属性
MI_VDISP_GetInputChannelAttr	获取输入通道属性
MI_VDISP_EnableInputChannel	使能一个输入通道
MI_VDISP_DisableInputChannel	关闭一个输入通道
MI_VDISP_StartDev	开始 VDISP 虚备的工作
MI_VDISP_StopDev	停止 VDISP 设备工作

2.1. MI_VDISP_Init

- 描述

  初始化 vdisp 模块,打开 vdisp 模块依赖模块,申请并初始模块内部的全局数据。

- **语法**

  MI_S32 MI_VDISP_Init (void);

- 参数

  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_vdisp.h mi_vdisp_datatype.h
  - **库文件**：libmi_vdisp.a(so)

※ **注意**

  对 vdisp 任何操作之前,先调用本函数初始化 vdisp 模块

- **举例**

  无

- **相关主题**

  MI_VDISP_Exit

2.2. MI_VDISP_Exit

- **描述**

  退出 vdisp 模块,关闭 vdisp 模块依赖模块,析构模块内部的全局数据。

- **语法**

  MI_S32 MI_VDISP_Exit (void);

- **参数**

  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_vdisp.h mi_vdisp_datatype.h
- **库文件**：libmi_vdisp.a(so)

※ **注意**

无。

➢ **举例**

无。

➢ **相关主题**

MI_VDISP_Init。

2.3. MI_VDISP_OpenDevice

➢ **描述**

初始化 vdisp 虚拟设备，向 sys 注册本虚拟设备，用以和其它模块绑定。

➢ **语法**

MI_S32 MI_VDISP_OpenDevice(MI_VDISP_DEV DevId)

➢ **参数**

参数名称	描述	输入/输出
DevId	本参数指定要打开的虚拟设备 ID. [0,VDISP_MAX_DEVICE_NUM).	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_vdisp.h mi_vdisp_datatype.h
- **库文件**：libmi_vdisp.a(so)

※ **注意**

调用本函数前，先使用 **MI_VDISP_Init** 初始化 vdisp 模块。

➢ **举例**

无。

- **相关主题**

无。

2.4. MI_VDISP_CloseDevice

- **描述**

关闭 vdisp 虚拟设备,向 sys 解注册本虚拟设备,不能绑定其它模块。

- **语法**

MI_S32 MI_VDISP_CloseDevice(MI_VDISP_DEV DevId);

- **参数**

参数名称	描述	输入/输出
DevId	本参数指定要关闭的虚拟设备 ID. [0,VDISP_MAX_DEVICE_NUM).	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_vdisp.h mi_vdisp_datatype.h
- 库文件：libmi_vdisp.a(so)

※ **注意**

调用本函数前,先使用 `MI_VDISP_StopDev` 停止对应的设备。

- **举例**

无。

- **相关主题**

无。

2.5. MI_VDISP_SetOutputPortAttr

- **描述**

设置该 vdisp 虚拟设备 output port 的参数。

- **语法**

MI_S32 MI_VDISP_SetOutputPortAttr(MI_VDISP_DEV DevId, MI_VDISP_PORT PortId,
MI_VDISP_OutputPortAttr_t *pstOutputPortAttr);

- **参数**

参数名称	描述	输入/输出
DevId	目标 output port 所属 vdisp 虚拟设备的 ID. [0,`VDISP_MAX_DEVICE_NUM`).	输入
PortId	目标 output port 的 port ID. [0,`VDISP_MAX_INPUTPORT_NUM`).	输入
pstOutputPortAttr	目标 outputport 的配置参数指针. `MI_VDISP_OutputPortAttr_t`.	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_vdisp.h mi_vdisp_datatype.h
  - **库文件**：libmi_vdisp.a(so)

※ **注意**

调用本函数前,先使用 `MI_VDISP_OpenDevice` 打开对应的 vdisp 设备。

- **举例**

无。

- **相关主题**

无。

2.6. MI_VDISP_GetOutputPortAttr

- **描述**

获取该 vdisp 虚拟设备 output port 的参数。

- **语法**

```
MI_S32 MI_VDISP_GetOutputPortAttr(MI_VDISP_DEV DevId, MI_VDISP_PORT PortId,
                                  MI_VDISP_OutputPortAttr_t *pstOutputPortAttr);
```

- **参数**

参数名称	描述	输入/输出
DevId	目标 output port 所属 vdisp 虚拟设备的 ID. [0,`VDISP_MAX_DEVICE_NUM`).	输入

PortId	目标 output port 的 port ID. **[0,VDISP_MAX_INPUTPORT_NUM)**.	输入
pstOutputPortAttr	目标 outputport 的配置参数指针. **MI_VDISP_OutputPortAttr_t**.	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_vdisp.h mi_vdisp_datatype.h
- **库文件**：libmi_vdisp.a(so)

※ **注意**

调用本函数前,先使用 **MI_VDISP_OpenDevice** 打开对应的 vdisp 设备。

➢ **举例**

无。

➢ **相关主题**

无。

2.7. MI_VDISP_SetInputChannelAttr

➢ **描述**

设置该 vdisp 虚拟设备 input channel 的参数。

➢ **语法**

```
MI_S32 MI_VDISP_SetInputChannelAttr(MI_VDISP_DEV DevId, MI_VDISP_CHN
                                    ChnId,MI_VDISP_InputChnAttr_t *pstInputChnAttr);
```

➢ **参数**

参数名称	描述	输入/输出
DevId	目标 output port 所属 vdisp 虚拟设备的 ID. **[0,VDISP_MAX_DEVICE_NUM)**.	输入
ChnId	目标 input channel 的 channel ID. **[0,VDISP_MAX_CHN_NUM_PER_DEV+VDISP_MAX_OVERLAYINPUTCHN_NUM)**.	输入
pstInputChntAttr	目标 input channel 的配置参数指	输入

	针. `MI_VDISP_InputChnAttr_t`.	

- **返回值**

返回值	
0	成功。
非 0	失败，参照错误码。

➢ **依赖**

- **头文件**：mi_vdisp.h mi_vdisp_datatype.h
- **库文件**：libmi_vdisp.a(so)

※ **注意**

调用本函数前，先使用 **MI_VDISP_OpenDevice** 打开对应的 vdisp 设备。

➢ **举例**

无。

➢ **相关主题**

无。

2.8. MI_VDISP_GetInputChannelAttr

➢ **描述**

获取该 vdisp 虚拟设备 input channel 的参数。

➢ **语法**

MI_S32 MI_VDISP_GetInputChannelAttr(MI_VDISP_DEV DevId, MI_VDISP_CHN ChnId,MI_VDISP_InputChnAttr_t *pstInputChnAttr);

➢ **参数**

参数名称	描述	输入/输出
DevId	目标 input channel 所属 vdisp 虚拟设备的 ID. [0,**VDISP_MAX_DEVICE_NUM**).	输入
ChnId	目标 input channel 的 channel ID. [0,**VDISP_MAX_CHN_NUM_PER_DEV+VDISP_MAX_OVERLAYINPUTCHN_NUM**).	输入
pstInputChnAttr	目标 input channel 的配置参数指针. **MI_VDISP_InputChnAttr_t**.	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_vdisp.h mi_vdisp_datatype.h
- **库文件**：libmi_vdisp.a(so)

※ **注意**

调用本函数前,先使用 `MI_VDISP_OpenDevice` 打开对应的 vdisp 设备。

➢ 举例

无。

➢ 相关主题

无。

2.9. MI_VDISP_EnableInputChannel

➢ 描述

使能该 input channel,vdisp 模块开始接收 deviceID=DevId,channelID=ChnId 的输入通道的数据。
标记该 channel 状态为 enbale

➢ 语法

MI_S32 MI_VDISP_EnableInputChannel(MI_VDISP_DEV DevId, MI_VDISP_CHN ChnId);

➢ 参数

参数名称	描述	输入/输出
DevId	目标 input channel 所属 vdisp 虚拟设备的 ID. [0,VDISP_MAX_DEVICE_NUM).	输入
ChnId	目标 input channel 的 channel ID. [0,VDISP_MAX_CHN_NUM_PER_DEV+VDISP_MAX_OVERLAYINPUTCHN_NUM).	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_vdisp.h mi_vdisp_datatype.h
- 库文件：libmi_vdisp.a(so)

※ 注意

调用本函数前,先使用 MI_VDISP_SetInputChannelAttr 设置对应 input channel 的参数。

➢ 举例

无。

➢ 相关主题

无。

2.10. MI_VDISP_DisableInputChannel

➢ **描述**

用该 input channel,vdisp 模块停止接收 deviceID=DevId,channelID=ChnId 的输入通道的数据
标记该 channel 状态为 disable

➢ **语法**

MI_S32 MI_VDISP_DisableInputChannel(MI_VDISP_DEV DevId, MI_VDISP_CHN ChnId);

➢ **参数**

参数名称	描述	输入/输出
DevId	目标 input channel 所属 vdisp 虚拟设备的 ID. [0,VDISP_MAX_DEVICE_NUM).	输入
ChnId	目标 input channel 的 channel ID. [0,VDISP_MAX_CHN_NUM_PER_DEV+VDISP_MAX_OVERLAYINPUTCHN_NUM).	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_vdisp.h mi_vdisp_datatype.h
- 库文件：libmi_vdisp.a(so)

※ **注意**

调用本函数前,先使用 `MI_VDISP_SetInputChannelAttr` 设置对应 input channel 的参数。

➢ **举例**

无。

➢ **相关主题**

无。

2.11. MI_VDISP_StartDev

➢ **描述**

使能该虚拟设备,vdisp output port 开始尝试输出 input channels 数据拼图之后的数据帧
使能 input channel 状态为 enbale 的所有 input channel,使能 output port

- 语法

  MI_S32 MI_VDISP_StartDev(MI_VDISP_DEV DevId);

➢ **参数**

参数名称	描述	输入/输出
DevId	本参数指定要启动的虚拟设备 ID. [0,VDISP_MAX_DEVICE_NUM).	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_vdisp.h mi_vdisp_datatype.h
- **库文件**：libmi_vdisp.a(so)

※ **注意**

调用本函数前,先使用 **MI_VDISP_OpenDevice** 打开对应的 vdisp 虚拟设备。

➢ **描述**

无。

➢ **相关主题**

无。

2.12. MI_VDISP_StopDev

➢ **描述**

停止该虚拟设备,vdisp output port 停止输出数据帧
禁用 input channel 状态为 enbale 的所有 input channel 以及 output port,但是不修改其状态

➢ **语法**

MI_S32 MI_VDISP_StopDev(MI_VDISP_DEV DevId);

➢ **参数**

参数名称	描述	输入/输出
DevId	本参数指定要停止的虚拟设备 ID. [0,VDISP_MAX_DEVICE_NUM).	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_vdisp.h mi_vdisp_datatype.h
  - 库文件：libmi_vdisp.a(so)

※ 注意

本函数会 disable 所有 enable 的输入通道/输出端口,同时清空 vdisp 缓存的数据帧。

- 举例

无。

- 相关主题

无。

3. 数据类型

MI VDISP 相关数据类型、数据结构定义如下：

数据结构	说明
MI_VDISP_DEV	定义 device id 类型
MI_VDISP_PORT	定义 port id 类型
MI_VDISP_CHN	定义 channel id 类型
MI_VDISP_OutputPortAttr_t	定义输出端口属性
MI_VDISP_InputChnAttr_t	定义输入端口属性

3.1. MI_VDISP_DEV

➢ **说明**

定义 device id 类型。

➢ **定义**

typedef MI_S32 MI_VDISP_DEV;

➢ **成员**

无。

※ **注意事项**

小于 0 为无效值

➢ **相关数据类型及接口**

无。

3.2. MI_VDISP_PORT

➢ **说明**

定义 port id 类型。

➢ **定义**

typedef MI_S32 MI_VDISP_PORT;

➢ **成员**

无。

※ **注意事项**

小于 0 为无效值。

➢ **相关数据类型及接口**

无。

3.3. MI_VDISP_CHN

➢ **说明**

定义 channel id 类型。

➢ **定义**

typedef MI_S32 MI_VDISP_CHN;

➢ **成员**

无。

※ **注意事项**

小于 0 为无效值。

➢ **相关数据类型及接口**

无。

3.4. MI_VDISP_OutputPortAttr_t

```
typedef struct MI_VDISP_OutputPortAttr_s
{
    MI_U32 u32BgColor;          /* Background color of a output port,
in YUV > format. [23:16]:v, [15:8]:y, [7:0]:u*/
    MI_SYS_PixelFormat_e ePixelFormat; /* pixel format of a output port
*/
    MI_U64 u64pts; /* current PTS */
    MI_U32 u32FrmRate; /* the frame rate of output port */
    MI_U32 u32Width; /* the frame width of a output port */
    MI_U32 u32Height; /* the frame height of a output port */
} MI_VDISP_OutputPortAttr_t;
```

➢ **说明**

定义输出端口属性。

➢ **定义**

➢ **成员**

成员名称	描述
u32BgColor	vdisp 用来给未使用的预览窗口区域涂黑的颜色(yuv 颜色空间)。很多情况下 vdisp 的整张 frame 并不是所有的区域都会有预览窗口被使用，这些区域需要被填充默认值，否则会输出杂讯。

ePixelFormat	输出 frame 的像素格式
u64pts	vdisp 输出 frame 的合法最早 PTS, 当前输入数据的 PTS 小于这个 PTS, 不会被会 vidsp 输出。
u32FrmRate	vdisp 输出 frame 对帧率, vdisp 采用固定帧率输出, 如果输入数据的帧率不足这个帧率, VDISP 会使用上一次收到的 frame 插帧。
u32Width	vdisp 输出 frame 的宽。
u32Height	vdisp 输出 frame 的高。

※ **注意事项**

无。

➢ **相关数据类型及接口**

无。

3.5. MI_VDISP_InputChnAttr_t

➢ **说明**

vdisp 的 input channel 属性。

➢ 定义

```
typedef struct MI_VDISP_InputPortAttr_s
{
   MI_U32 u32OutX; /* the output frame X position of this input port */
   MI_U32 u32OutY; /* the output frame Y position of this input port */
   MI_U32 u32OutWidth; /* the output frame width of this input port */
   MI_U32 u32OutHeight; /* the output frame height of this input port
*/
   MI_S32 s32IsFreeRun; /* is this port free run */
} MI_VDISP_InputChnAttr_t;
```

➢ **成员**

成员名称	描述
u32OutX	vdisp 输入通道在输出的 frame 的预览窗口的起始横坐标
u32OutY	vdisp 输入通道在输出的 frame 的预览窗口的起始纵坐标
u32OutWidth	vdisp 输入通道在输出的 frame 的预览窗口的宽
u32OutHeight	vdisp 输入通道在输出的 frame 的预览窗口的高
s32IsFreeRun	vdisp 输入通道 frame 数据的 PTS 是否被检查 ● TRUE: 不被检查,始终能够输出 ● FALSE: 被检查,小于 output port 的 pts 则不被输出

※ 注意事项

无。

➢ 相关数据类型及接口

无。

4. 错误码

错误码如表 3-1 所示：

表 3-1 API 错误码

宏定义	描述
MI_SUCCESS	操作成功
MI_VDISP_ERR_FAIL	vdisp 函数执行失败,参见系统 log
MI_VDISP_ERR_INVALID_DEVID	函数的 dev 参数非法
MI_VDISP_ERR_ILLEGAL_PARAM	函数的某个参数非法,参见系统 log
MI_VDISP_ERR_NOT_SUPPORT	函数不支持当前参数设定
MI_VDISP_ERR_MOD_INITED	vdisp 已经初始化过
MI_VDISP_ERR_MOD_NOT_INIT	vdisp 还没有初始化
MI_VDISP_ERR_DEV_OPENED	vdisp dev 已经 open 过
MI_VDISP_ERR_DEV_NOT_OPEN	在调用该函数前,需要先 open vdisp 设备
MI_VDISP_ERR_DEV_NOT_STOP	在调用该函数前,需要先 stop vdisp 设备
MI_VDISP_ERR_DEV_NOT_CLOSE	在调用该函数前,需要先 close vdisp 设备
MI_VDISP_ERR_NOT_CONFIG	vdisp 的 input/output 没有配置
MI_VDISP_ERR_PORT_NOT_DISABLE	在调用该函数前,需要先 disable 该 channel/port
MI_VDISP_ERR_PORT_NOT_UNBIND	参数指定 port,未绑定前后端

5. 规格

规格参数如表 4-1 所示：

表 4-1 vdisp 模块规格参数

宏定义	参数	描述
VDISP_MAX_DEVICE_NUM	4	VDISP 模块支持的虚拟设备个数
VDISP_MAX_CHN_NUM_PER_DEV	16	单个虚拟设备，支持的最大非叠加输入通道的个数
VDISP_MAX_INPUTPORT_NUM	1	单个虚拟设备的单个非叠加输入通道的入口数。
VDISP_MAX_OVERLAYINPUTCHN_NUM	1	单个虚拟设备可叠加通道的个数，该参数为 built-in 的常数，可以增加以支持更多的可叠加窗口
VDISP_OVERLAYINPUTCHNID	VDISP_MAX_CHN_NUM_PER_DEV	可叠加输入通道的起始 ID
VDISP_MAX_OUTPUTPORT_NUM	1	单个虚拟设备的最大出口数

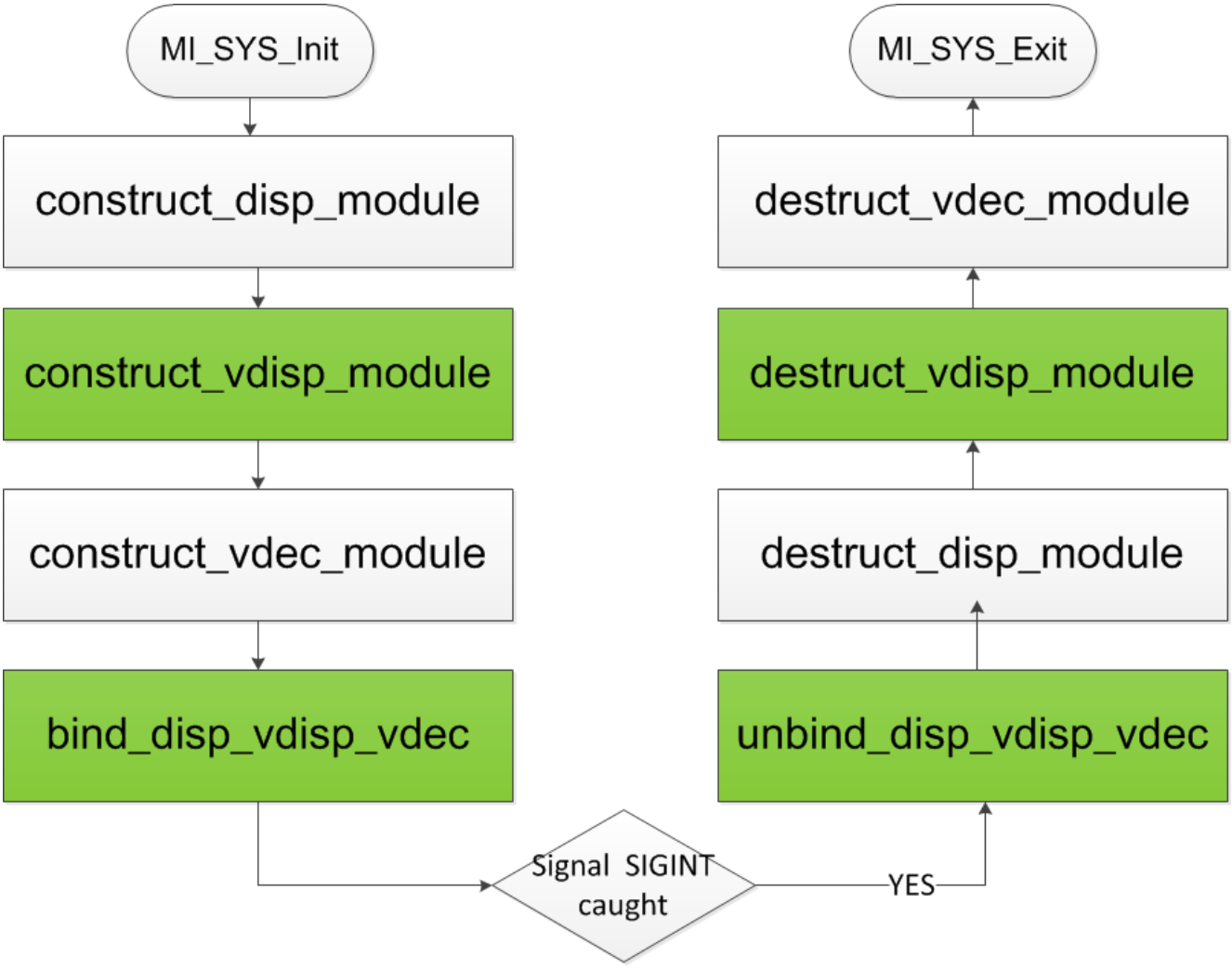
6. 附录

6.1. 示例代码

本节示例代码实现了模块说明中的场景。

6.1.1 示例程序流程图

绿色部分为 vdisp 模块相关流程。



6.1.2 vdisp 相关代码

6.1.2.1. 构建 vdisp 模块

```
/*
 * 初始化 vdisp 模块
 *      v
 * 打开 vdisp 设备
 *      v
 * 设置 vdisp input channel/output port 参数
 *      v
 * 激活 input channel
 *      v
 * 开始 vdisp 设备
 */
void construct_vdisp_module(void)
{
#define MAKE_YUYV_VALUE(y,u,v)    ((y) << 24) | ((u) << 16) | ((y) << 8) | (v)
#define YUYV_BLACK                MAKE_YUYV_VALUE(0,128,128)
#define ALIGN16_DOWN(x) (x&0xFFF0)

    typedef struct RECT
    {
        /*
        (x,y)_ _ _ _ _ _ _
          |             |
          |           (h)
          |_ _ _(w)_ _ _|
        */
        short x;
        short y;
        short w;
        short h;
    } RECT;

    /*
     *初始化 vdisp 模块
     */
    MI_VDISP_Init();

    MI_VDISP_InputChnAttr_t stInputChnAttr;
    MI_VDISP_OutputPortAttr_t stOutputPortAttr;
    MI_VDISP_DEV DevId = 0;
    RECT rect_0, rect_1, rect_2;
    MI_VDISP_CHN Chn_0_Id = VDISP_OVERLAYINPUTCHNID;
    MI_VDISP_CHN Chn_1_Id = 1;
    MI_VDISP_CHN Chn_2_Id = 2;

    /*
     *打开一个 vdisp 虚拟设备,以便开始对这个设备进行配置
```

```
 */
MI_VDISP_OpenDevice(DevId);

/*
 *为了满足 vdisp 缩略图窗口的对齐要求，对 x 坐标做 16 向下对齐
 *设置 input channel 参数
 */
rect_0.x = ALIGN16_DOWN(960);
rect_0.y = 0;
rect_0.w = 960;
rect_0.h = 960;
stInputChnAttr.s32IsFreeRun = TRUE;
stInputChnAttr.u32OutHeight = rect_0.h;
stInputChnAttr.u32OutWidth = rect_0.w;
stInputChnAttr.u32OutX = rect_0.x;
stInputChnAttr.u32OutY = rect_0.y;
/*
 *为 input channel VDISP_OVERLAYINPUTCHNID 设置参数
 */
MI_VDISP_SetInputChannelAttr(DevId, Chn_0_Id, &stInputChnAttr);

/*
 *为了满足 vdisp 缩略图窗口的对齐要求，对 x 坐标做 16 向下对齐
 *设置 input channel 参数
 */
rect_1.x = ALIGN16_DOWN(0);
rect_1.y = 1080 - 300;
rect_1.w = ALIGN16_DOWN(300);
rect_1.h = 300;
stInputChnAttr.s32IsFreeRun = TRUE;
stInputChnAttr.u32OutHeight = rect_1.h;
stInputChnAttr.u32OutWidth = rect_1.w;
stInputChnAttr.u32OutX = rect_1.x;
stInputChnAttr.u32OutY = rect_1.y;
/*
 *为 input channel 1 设置参数
 */
MI_VDISP_SetInputChannelAttr(DevId, Chn_1_Id, &stInputChnAttr);

/*
 *为了满足 vdisp 缩略图窗口的对齐要求，对 x 坐标做 16 向下对齐
 *设置 input channel 参数
 */
rect_2.x = ALIGN16_DOWN(300);
rect_2.y = 1080 - 700;
rect_2.w = 700;
```

```
    rect_2.h = 700;
    stInputChnAttr.s32IsFreeRun = TRUE;
    stInputChnAttr.u32OutHeight = rect_2.h;
    stInputChnAttr.u32OutWidth = rect_2.w;
    stInputChnAttr.u32OutX = rect_2.x;
    stInputChnAttr.u32OutY = rect_2.y;

    /*
     *为input channel 2 设置参数
     */
    MI_VDISP_SetInputChannelAttr(DevId, Chn_2_Id, &stInputChnAttr);

    //设置输出的颜色格式
    stOutputPortAttr.ePixelFormat = E_MI_SYS_PIXEL_FRAME_YUV_SEMIPLANAR_420;
    //设置vdisp 输出帧中未使用区域的涂黑颜色,YUV 颜色空间
    stOutputPortAttr.u32BgColor = YUYV_BLACK;
    //设置vdisp 输出帧率
    stOutputPortAttr.u32FrmRate = 30;
    //设置vdisp 输出帧的高
    stOutputPortAttr.u32Height = 1080;
    //设置vdisp 输出帧的宽
    stOutputPortAttr.u32Width = 1920;
    //设置vdisp 输出帧的最低PTS
    stOutputPortAttr.u64pts = 0;

    /*
     *为output port 0 设置参数
     */
    MI_VDISP_SetOutputPortAttr(DevId,0,&stOutputPortAttr);


    /*
     *在设置完input channel 参数之后,
     *激活对应的channel, 以供使用
     */
    MI_VDISP_EnableInputChannel(DevId, Chn_0_Id);
    MI_VDISP_EnableInputChannel(DevId, Chn_1_Id);
    MI_VDISP_EnableInputChannel(DevId, Chn_2_Id);

    /*
     *在vdisp 的input channel&output port 都配置完参数之后,
     *让vdisp 的这个设备开始工作
     */
    MI_VDISP_StartDev(DevId);
}
```

6.1.2.2. 绑定 vdisp 上下游模块

```
/*
 * 绑定 vdec out0 -> vdisp inOverlay---\
 * 绑定 vdec out1 -> vdisp in1 --------->--> disp in0
 * 绑定 vdec out2 -> vdisp in2---------/
 */
void bind_disp_vdisp_vdec(void)
{
    MI_SYS_ChnPort_t stSrcChnPort;
    MI_SYS_ChnPort_t stDstChnPort;

    /*
     * 绑定vdisp out0-> disp in0
     * <chn,dev,port> : (0,0,0) -> (0,0,0)
     */
    stSrcChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stSrcChnPort.u32ChnId = 0;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_DISP;
    stDstChnPort.u32ChnId = 0;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_BindChnPort(&stSrcChnPort, &stDstChnPort, 30, 30);

    /*
     * 绑定vdec out0-> vdisp inOverlay
     * <chn,dev,port> : (0,0,0) -> (VDISP_OVERLAYINPUTCHNID,0,0)
     */
    stSrcChnPort.eModId = E_MI_MODULE_ID_VDEC;
    stSrcChnPort.u32ChnId = 0;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stDstChnPort.u32ChnId = VDISP_OVERLAYINPUTCHNID;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_BindChnPort(&stSrcChnPort, &stDstChnPort, 30, 30);

    /*
     * 绑定vdec out0-> vdisp in1
```

```
     * <chn,dev,port> : (1,0,0) -> (1,0,0)
     */
    stSrcChnPort.eModId = E_MI_MODULE_ID_VDEC;
    stSrcChnPort.u32ChnId = 1;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stDstChnPort.u32ChnId = 1;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_BindChnPort(&stSrcChnPort, &stDstChnPort, 30, 30);

    /*
     * 绑定 vdec out2-> vdisp in2
     * <chn,dev,port> : (2,0,0) -> (2,0,0)
     */
    stSrcChnPort.eModId = E_MI_MODULE_ID_VDEC;
    stSrcChnPort.u32ChnId = 2;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stDstChnPort.u32ChnId = 2;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_BindChnPort(&stSrcChnPort, &stDstChnPort, 30, 30);

}
```

6.1.2.3. 解绑 vdisp 上下游模块

与绑定过程区别很小，参见完整代码实现

6.1.2.4. 析构 vdisp 模块

```
/*
 * 禁用所有 channel
 *      v
 * 停止 vdisp 设备
 *      v
 * 关闭 vdisp 设备
 *      v
 * 退出 vdisp 模块
 */
void destruct_vdisp_module(void)
{
```

```
    MI_VDISP_DEV DevId = 0;
    MI_VDISP_CHN Chn_0_Id = VDISP_OVERLAYINPUTCHNID;
    MI_VDISP_CHN Chn_1_Id = 1;
    MI_VDISP_CHN Chn_2_Id = 2;
   /*
    * 先禁用已经打开vdisp channel
    * <VDISP_OVERLAYINPUTCHNID,1,2>
    */
    MI_VDISP_DisableInputChannel(DevId, Chn_0_Id);
    MI_VDISP_DisableInputChannel(DevId, Chn_1_Id);
    MI_VDISP_DisableInputChannel(DevId, Chn_2_Id);

   /*
    *停止已经打开的vdisp 设备
    */
    MI_VDISP_StopDev(DevId);
   /*
    *关闭已经打开的vdisp 设备
    */
    MI_VDISP_CloseDevice(DevId);
   /*
    *退出vdisp 模块
    */
    MI_VDISP_Exit();
}
```

6.1.3 完整代码

```
#include <threads.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <assert.h>
#include <signal.h>


#include <mi_sys_datatype.h>
#include <mi_sys.h>
#include <mi_vdec_datatype.h>
#include <mi_vdec.h>
#include <mi_disp_datatype.h>
#include <mi_disp.h>
#include <mi_vdisp_datatype.h>
#include <mi_vdisp.h>
```

```
static volatile int keepRunning = 1;

static void intHandler(int dummy)
{
    keepRunning = 0;
}

void construct_disp_module(void)
{

    MI_DISP_PubAttr_t stDispPubAttr;
    stDispPubAttr.eIntfSync = E_MI_DISP_OUTPUT_1080P60;
    stDispPubAttr.eIntfType = E_MI_DISP_INTF_VGA;
    MI_DISP_SetPubAttr(0, &stDispPubAttr);

    MI_DISP_InputPortAttr_t stInputPortAttr;
    stInputPortAttr.u16SrcWidth = 1920;
    stInputPortAttr.u16SrcHeight = 1080;
    stInputPortAttr.stDispWin.u16X = 0;
    stInputPortAttr.stDispWin.u16Y = 0;
    stInputPortAttr.stDispWin.u16Width = 1920;
    stInputPortAttr.stDispWin.u16Height = 1080;
    MI_DISP_SetInputPortAttr(0, 0, &stInputPortAttr);

    MI_DISP_Enable(0);
    MI_DISP_EnableVideoLayer(0);
    MI_DISP_EnableInputPort(0, 0);
}
void destruct_disp_module(void)
{
    MI_DISP_DisableInputPort(0, 0);
    MI_DISP_DisableVideoLayer(0);
    MI_DISP_UnBindVideoLayer(0, 0);
    MI_DISP_Disable(0);
}
typedef struct vdecStream
{
    FILE *fp;
    MI_VDEC_CHN VdecChn;
    int frameRate;
} vdecStream;

MI_U64 get_pts(MI_U32 u32FrameRate)
```

```
{
    if(0 == u32FrameRate)
    {
        return (MI_U64)(-1);
    }

    return (MI_U64)(1000 / u32FrameRate);
}

int push_stream(void *args)
{

#define MI_U32VALUE(pu8Data, index) (pu8Data[index]<<24)|(pu8Data[index+1]<<16)|(pu8D
ata[index+2]<<8)|(pu8Data[index+3])
#define VESFILE_READER_BATCH (128 * 1024)

    MI_S32 s32Ret = MI_SUCCESS;
    MI_VDEC_CHN VdecChn;
    MI_U32 u32FrmRate = 0;
    MI_U8 *pu8Buf = NULL;
    MI_U32 u32Len = 0;
    MI_U32 u32FrameLen = 0;
    MI_U64 u64Pts = 0;
    MI_U8 au8Header[16] = {0};
    MI_U32 u32Pos = 0;
    MI_VDEC_ChnStat_t stChnStat;
    vdecStream *vdecS = (vdecStream *)args;
    MI_VDEC_VideoStream_t stVdecStream;
    MI_S32 s32TimeOutMs = 20;

    MI_U32 u32FpBackLen = 0; // if send stream failed, file pointer back length

    VdecChn    = vdecS->VdecChn;
    u32FrmRate = vdecS->frameRate;

    pu8Buf = malloc(VESFILE_READER_BATCH);

    while(keepRunning)
    {
        ///frame mode, read one frame lenght every time

        memset(au8Header, 0, 16);
        u32Pos = fseek(vdecS->fp, 0L, SEEK_CUR);
        u32Len = fread(au8Header, 16, 1, vdecS->fp);

        if(u32Len <= 0)
```

```
        {
            fseek(vdecS->fp, 0, SEEK_SET);
            continue;
        }

        u32FrameLen = MI_U32VALUE(au8Header, 4);
        if(u32FrameLen > VESFILE_READER_BATCH)
        {
            fseek(vdecS->fp, 0, SEEK_SET);
            continue;
        }

        u32Len = fread(pu8Buf, u32FrameLen, 1, vdecS->fp);

        if(u32Len <= 0)
        {
            fseek(vdecS->fp, 0, SEEK_SET);
            continue;
        }

        stVdecStream.pu8Addr = pu8Buf;
        stVdecStream.u32Len = u32FrameLen;
        stVdecStream.u64PTS = u64Pts;
        stVdecStream.bEndOfFrame = 1;
        stVdecStream.bEndOfStream = 0;

        u32FpBackLen = stVdecStream.u32Len + 16; //back length

        if(MI_SUCCESS != (s32Ret = MI_VDEC_SendStream(VdecChn, &stVdecStream, s32Time
OutMs)))
        {
            fseek(vdecS->fp, - u32FpBackLen, SEEK_CUR);
        }

        u64Pts = u64Pts + get_pts(1000 / u32FrmRate);

        memset(&stChnStat, 0x0, sizeof(stChnStat));
        MI_VDEC_GetChnStat(VdecChn, &stChnStat);
        usleep(20 * 1000);

    }

    free(pu8Buf);
    return NULL;
}
```

```
thrd_t thr0, thr1, thr2;
vdecStream vS0, vS1, vS2;

void construct_vdec_module(void)
{
    MI_VDEC_ChnAttr_t stChnAttr;
    MI_VDEC_CHN Chn_0_Id = 0;
    MI_VDEC_CHN Chn_1_Id = 1;
    MI_VDEC_CHN Chn_2_Id = 2;

    memset(&stChnAttr, 0, sizeof(MI_VDEC_ChnAttr_t));

    MI_VDEC_InitParam_t stVdecInitParam;
    MI_VDEC_ChnParam_t stChnParam;
    MI_VDEC_OutputPortAttr_t stOutputPortAttr;
    stVdecInitParam.bDisableLowLatency = FALSE;
    MI_VDEC_InitDev(&stVdecInitParam);

    stChnAttr.eCodecType    = E_MI_VDEC_CODEC_TYPE_H264;
    stChnAttr.u32PicWidth   = 1920;
    stChnAttr.u32PicHeight  = 1080;
    stChnAttr.eVideoMode    = E_MI_VDEC_VIDEO_MODE_FRAME;
    stChnAttr.u32BufSize    = 1024 * 1024;
    stChnAttr.eDpbBufMode  = E_MI_VDEC_DPB_MODE_NORMAL;
    stChnAttr.stVdecVideoAttr.u32RefFrameNum = 2;
    stChnAttr.u32Priority = 0;


    MI_VDEC_CreateChn(Chn_0_Id, &stChnAttr);
    stOutputPortAttr.u16Width = 960;
    stOutputPortAttr.u16Height = 960;
    MI_VDEC_SetOutputPortAttr(Chn_0_Id, &stOutputPortAttr);

    MI_VDEC_CreateChn(Chn_1_Id, &stChnAttr);
    stOutputPortAttr.u16Width = 300;
    stOutputPortAttr.u16Height = 300;
    MI_VDEC_SetOutputPortAttr(Chn_1_Id, &stOutputPortAttr);

    MI_VDEC_CreateChn(Chn_2_Id, &stChnAttr);
    stOutputPortAttr.u16Width = 700;
    stOutputPortAttr.u16Height = 700;
    MI_VDEC_SetOutputPortAttr(Chn_2_Id, &stOutputPortAttr);

    MI_VDEC_StartChn(Chn_0_Id);
    MI_VDEC_StartChn(Chn_1_Id);
    MI_VDEC_StartChn(Chn_2_Id);
```

```
#define VDEC_FILE0 "./vdisp_demo0.h264"
#define VDEC_FILE1 "./vdisp_demo1.h264"
#define VDEC_FILE2 "./vdisp_demo2.h264"


    vS0.fp = fopen(VDEC_FILE0, "rb");
    vS0.frameRate = 30;
    vS0.VdecChn = Chn_0_Id;
    assert(vS0.fp);
    thrd_create(&thr0, push_stream, &vS0);
    vS1.fp = fopen(VDEC_FILE1, "rb");
    vS1.frameRate = 30;
    vS1.VdecChn = Chn_1_Id;
    assert(vS1.fp);
    thrd_create(&thr1, push_stream, &vS1);
    vS2.fp = fopen(VDEC_FILE2, "rb");
    vS2.frameRate = 30;
    vS2.VdecChn = Chn_2_Id;
    assert(vS2.fp);
    thrd_create(&thr2, push_stream, &vS2);
}
void destruct_vdec_module(void)
{
    thrd_join(thr0,NULL);
    thrd_join(thr1,NULL);
    thrd_join(thr2,NULL);
    MI_VDEC_StopChn(vS0.VdecChn);
    MI_VDEC_StopChn(vS1.VdecChn);
    MI_VDEC_StopChn(vS2.VdecChn);
    MI_VDEC_DestroyChn(vS0.VdecChn);
    MI_VDEC_DestroyChn(vS1.VdecChn);
    MI_VDEC_DestroyChn(vS2.VdecChn);
    MI_VDEC_DeInitDev();
}

/*
 * 初始化 vdisp 模块
 *      v
 * 打开 vdisp 设备
 *      v
 * 设置 vdisp input channel/output port 参数
 *      v
 * 激活 input channel
 *      v
 * 开始 vdisp 设备
```

```
 */
void construct_vdisp_module(void)
{
#define MAKE_YUYV_VALUE(y,u,v)    ((y) << 24) | ((u) << 16) | ((y) << 8) | (v)
#define YUYV_BLACK                MAKE_YUYV_VALUE(0,128,128)
#define ALIGN16_DOWN(x) (x&0xFFF0)

    typedef struct RECT
    {
        /*
        (x,y)_ _ _ _ _ _ _
          |             |
          |           (h)
          |_ _ _(w)_ _ _|
        */
        short x;
        short y;
        short w;
        short h;
    } RECT;

    /*
     *初始化 vdisp 模块
     */
    MI_VDISP_Init();

    MI_VDISP_InputChnAttr_t stInputChnAttr;
    MI_VDISP_OutputPortAttr_t stOutputPortAttr;
    MI_VDISP_DEV DevId = 0;
    RECT rect_0, rect_1, rect_2;
    MI_VDISP_CHN Chn_0_Id = VDISP_OVERLAYINPUTCHNID;
    MI_VDISP_CHN Chn_1_Id = 1;
    MI_VDISP_CHN Chn_2_Id = 2;

    /*
     *打开一个 vdisp 虚拟设备,以便开始对这个设备进行配置
     */
    MI_VDISP_OpenDevice(DevId);

    /*
     *为了满足 vdisp 缩略图窗口的对齐要求，对 x 坐标做 16 向下对齐
     *设置 input channel 参数
     */
    rect_0.x = ALIGN16_DOWN(960);
    rect_0.y = 0;
    rect_0.w = 960;
```

```
    rect_0.h = 960;
    stInputChnAttr.s32IsFreeRun = TRUE;
    stInputChnAttr.u32OutHeight = rect_0.h;
    stInputChnAttr.u32OutWidth = rect_0.w;
    stInputChnAttr.u32OutX = rect_0.x;
    stInputChnAttr.u32OutY = rect_0.y;
    /*
     *为input channel VDISP_OVERLAYINPUTCHNID 设置参数
     */
    MI_VDISP_SetInputChannelAttr(DevId, Chn_0_Id, &stInputChnAttr);

    /*
     *为了满足vdisp缩略图窗口的对齐要求，对x坐标做16向下对齐
     *设置input channel 参数
     */
    rect_1.x = ALIGN16_DOWN(0);
    rect_1.y = 1080 - 300;
    rect_1.w = ALIGN16_DOWN(300);
    rect_1.h = 300;
    stInputChnAttr.s32IsFreeRun = TRUE;
    stInputChnAttr.u32OutHeight = rect_1.h;
    stInputChnAttr.u32OutWidth = rect_1.w;
    stInputChnAttr.u32OutX = rect_1.x;
    stInputChnAttr.u32OutY = rect_1.y;
    /*
     *为input channel 1 设置参数
     */
    MI_VDISP_SetInputChannelAttr(DevId, Chn_1_Id, &stInputChnAttr);

    /*
     *为了满足vdisp缩略图窗口的对齐要求，对x坐标做16向下对齐
     *设置input channel 参数
     */
    rect_2.x = ALIGN16_DOWN(300);
    rect_2.y = 1080 - 700;
    rect_2.w = 700;
    rect_2.h = 700;
    stInputChnAttr.s32IsFreeRun = TRUE;
    stInputChnAttr.u32OutHeight = rect_2.h;
    stInputChnAttr.u32OutWidth = rect_2.w;
    stInputChnAttr.u32OutX = rect_2.x;
    stInputChnAttr.u32OutY = rect_2.y;

    /*
     *为input channel 2 设置参数
     */
```

```
    MI_VDISP_SetInputChannelAttr(DevId, Chn_2_Id, &stInputChnAttr);

    //设置输出的颜色格式
    stOutputPortAttr.ePixelFormat = E_MI_SYS_PIXEL_FRAME_YUV_SEMIPLANAR_420;
    //设置 vdisp 输出帧中未使用区域的涂黑颜色,YUV 颜色空间
    stOutputPortAttr.u32BgColor = YUYV_BLACK;
    //设置 vdisp 输出帧率
    stOutputPortAttr.u32FrmRate = 30;
    //设置 vdisp 输出帧的高
    stOutputPortAttr.u32Height = 1080;
    //设置 vdisp 输出帧的宽
    stOutputPortAttr.u32Width = 1920;
    //设置 vdisp 输出帧的最低 PTS
    stOutputPortAttr.u64pts = 0;

    /*
     *为output port 0 设置参数
     */
    MI_VDISP_SetOutputPortAttr(DevId,0,&stOutputPortAttr);


    /*
     *在设置完 input channel 参数之后,
     *激活对应的 channel，以供使用
     */
    MI_VDISP_EnableInputChannel(DevId, Chn_0_Id);
    MI_VDISP_EnableInputChannel(DevId, Chn_1_Id);
    MI_VDISP_EnableInputChannel(DevId, Chn_2_Id);

    /*
     *在 vdisp 的 input channel&output port 都配置完参数之后,
     *让 vdisp 的这个设备开始工作
     */
    MI_VDISP_StartDev(DevId);
}


/*
 * 禁用所有 channel
 *      v
 * 停止 vdisp 设备
 *      v
 * 关闭 vdisp 设备
 *      v
 * 退出 vdisp 模块
 */
void destruct_vdisp_module(void)
```

```
{
    MI_VDISP_DEV DevId = 0;
    MI_VDISP_CHN Chn_0_Id = VDISP_OVERLAYINPUTCHNID;
    MI_VDISP_CHN Chn_1_Id = 1;
    MI_VDISP_CHN Chn_2_Id = 2;
   /*
    * 先禁用已经打开vdisp channel
    * <VDISP_OVERLAYINPUTCHNID,1,2>
    */
    MI_VDISP_DisableInputChannel(DevId, Chn_0_Id);
    MI_VDISP_DisableInputChannel(DevId, Chn_1_Id);
    MI_VDISP_DisableInputChannel(DevId, Chn_2_Id);

   /*
    *停止已经打开的vdisp 设备
    */
    MI_VDISP_StopDev(DevId);
   /*
    *关闭已经打开的vdisp 设备
    */
    MI_VDISP_CloseDevice(DevId);
   /*
    *退出vdisp 模块
    */
    MI_VDISP_Exit();
}

/*
 * 绑定 vdec out0 -> vdisp inOverlay---\
 * 绑定 vdec out1 -> vdisp in1 --------->--> disp in0
 * 绑定 vdec out2 -> vdisp in2---------/
 */
void bind_disp_vdisp_vdec(void)
{
    MI_SYS_ChnPort_t stSrcChnPort;
    MI_SYS_ChnPort_t stDstChnPort;

    /*
     * 绑定vdisp out0-> disp in0
     * <chn,dev,port> : (0,0,0) -> (0,0,0)
     */
    stSrcChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stSrcChnPort.u32ChnId = 0;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;
```

```
    stDstChnPort.eModId = E_MI_MODULE_ID_DISP;
    stDstChnPort.u32ChnId = 0;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_BindChnPort(&stSrcChnPort, &stDstChnPort, 30, 30);

    /*
     * 绑定 vdec out0-> vdisp inOverlay
     * <chn,dev,port> : (0,0,0) -> (VDISP_OVERLAYINPUTCHNID,0,0)
     */
    stSrcChnPort.eModId = E_MI_MODULE_ID_VDEC;
    stSrcChnPort.u32ChnId = 0;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stDstChnPort.u32ChnId = VDISP_OVERLAYINPUTCHNID;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_BindChnPort(&stSrcChnPort, &stDstChnPort, 30, 30);

    /*
     * 绑定 vdec out0-> vdisp in1
     * <chn,dev,port> : (1,0,0) -> (1,0,0)
     */
    stSrcChnPort.eModId = E_MI_MODULE_ID_VDEC;
    stSrcChnPort.u32ChnId = 1;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stDstChnPort.u32ChnId = 1;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_BindChnPort(&stSrcChnPort, &stDstChnPort, 30, 30);

    /*
     * 绑定 vdec out2-> vdisp in2
     * <chn,dev,port> : (2,0,0) -> (2,0,0)
     */
    stSrcChnPort.eModId = E_MI_MODULE_ID_VDEC;
    stSrcChnPort.u32ChnId = 2;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_VDISP;
```

```
    stDstChnPort.u32ChnId = 2;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_BindChnPort(&stSrcChnPort, &stDstChnPort, 30, 30);

}

void unbind_disp_vdisp_vdec(void)
{
    MI_SYS_ChnPort_t stSrcChnPort;
    MI_SYS_ChnPort_t stDstChnPort;

    stSrcChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stSrcChnPort.u32ChnId = 0;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_DISP;
    stDstChnPort.u32ChnId = 0;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_UnBindChnPort(&stSrcChnPort, &stDstChnPort);

    stSrcChnPort.eModId = E_MI_MODULE_ID_VDEC;
    stSrcChnPort.u32ChnId = 0;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stDstChnPort.u32ChnId = VDISP_OVERLAYINPUTCHNID;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_UnBindChnPort(&stSrcChnPort, &stDstChnPort);

    stSrcChnPort.eModId = E_MI_MODULE_ID_VDEC;
    stSrcChnPort.u32ChnId = 1;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stDstChnPort.u32ChnId = 1;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_UnBindChnPort(&stSrcChnPort, &stDstChnPort);

    stSrcChnPort.eModId = E_MI_MODULE_ID_VDEC;
```

```
    stSrcChnPort.u32ChnId = 2;
    stSrcChnPort.u32DevId = 0;
    stSrcChnPort.u32PortId = 0;

    stDstChnPort.eModId = E_MI_MODULE_ID_VDISP;
    stDstChnPort.u32ChnId = 2;
    stDstChnPort.u32DevId = 0;
    stDstChnPort.u32PortId = 0;
    MI_SYS_UnBindChnPort(&stSrcChnPort, &stDstChnPort);

}

int main(void)
{
    signal(SIGINT, intHandler);

    MI_SYS_Init();
    construct_disp_module();
    construct_vdisp_module();
    construct_vdec_module();
    bind_disp_vdisp_vdec();
    while(keepRunning)
    {
        sleep(1);
    }
    unbind_disp_vdisp_vdec();
    destruct_disp_module();
    destruct_vdisp_module();
    destruct_vdec_module();
    MI_SYS_Exit();
    return 0;
}
```