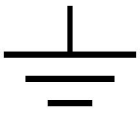
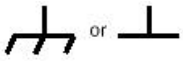
HT1051 系列 CameraLink 图像模拟源注入卡用户手册

Ver 1.2

Revision

时间	原因	版本	作者	备注
2023-2-8	建立文档	1.0	fjh	初次建立文档
2024-1-25	增加对于 HT1051A 的描述	1.1	fjh	
2024-9-14	增加保存 flash 接口	1.2	fjh	针对 1110 和 1210 版本

安全标志

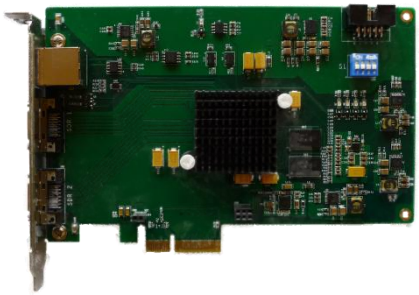
符号	说明	符号	说明
	粘贴在产品上的标志符号。表示使用者必须遵循产品手册中相应的警告或注意内容以免造成人身伤害或设备损坏。		表示仪器在操作前必须保证相应的接线端接地良好，以免电击而引起设备损坏或人身伤害。
	交流。		直流。
	高压危险。	**警告**	表示相应的操作危险。操作员应严格按规定操作，否则可能导致人身危险。
	机箱或外壳接地端。通常与设备的金属外壳相连接。	**注意**	表示相应的操作危险。操作员应严格按规定操作，否则可能导致设备损坏或永久性数据丢失。

1. 概述

1.1. 产品描述

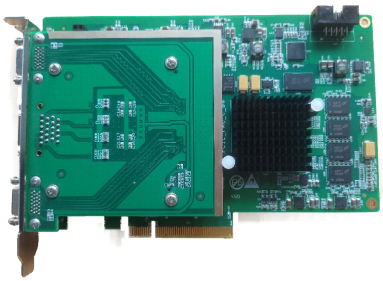
HT1051 系列 CameraLink 图像模拟源注入卡专用于图像数据模拟注入领域。此卡存在多种接口和形态，支持用户将图像数据直接写入板卡，然后板卡依照要求的硬件时序向外发送。板卡支持 Windows 7、Windows8 和 Windows10 系统，提供驱动，SDK 二次开发包及 Demo 示例代码，降低半实物仿真应用的开发难度与成本，加速客户项目应用的开发进度。

产品外观如下所示：



正视图

接口视图



HT1051A 正视图

HT1051A 接口视图

1.2. 接口定义表

HT1051 CameraLink 图像数据注入对外共有三个接口，两个 SDR26 连接器，其上承载标准 CameraLink 数据；一个 RJ45 辅助接口，用于承载两路 IO 信号和一路 RS422 信号。

SDR1 管脚信号定义

管脚序号	信号名	管脚序号	信号名	管脚序号	信号名
1	GND	2	X0-	3	X1-
4	X2-	5	Xclk-	6	X3-
7	NC	8	NC	9	CC1-
10	CC2+	11	CC3-	12	CC4+
13	GND	14	GND	15	X0+
16	X1+	17	X2+	18	Xclk+

19	X3+	20	NC	21	NC
22	CC1+	23	CC2-	24	CC3+
25	CC4-	26	GND		

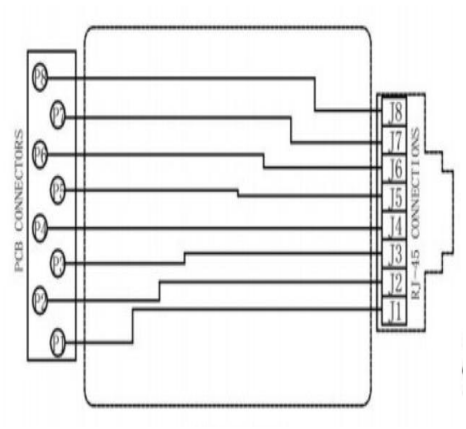
SDR2 管脚信号定义

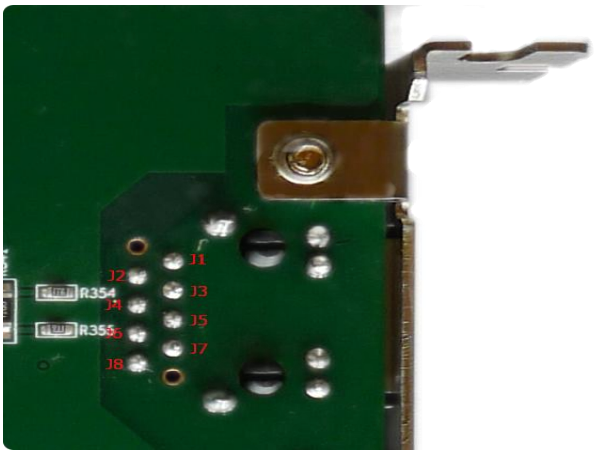
管脚序号	信号名		管脚序号	信号名	
	Medium/Full	Base①		Medium/Full	Base①
1	GND	GND	2	Y0-	X0-
3	Y1-	X1-	4	Y2-	X2-
5	Yclk-	Xclk-	6	Y3-	X3-
7	100Ω	NC	8	Z0-	NC
9	Z1-	NC	10	Z2-	NC
11	Zclk-	NC	12	Z3-	NC
13	GND	GND	14	GND	GND
15	Y0+	X0+	16	Y1+	X1+
17	Y2+	X2+	18	Yclk+	Xclk+
19	Y3+	X3+	20	terminated	NC
21	Z0+	NC	22	Z1+	NC
23	Z2+	NC	24	Zclk+	NC
25	Z3+	NC	26	GND	GND

辅助接口（RJ45）管脚信号定义

管脚序号	信号名	管脚序号	信号名	管脚序号	信号名	管脚序号	信号名
J1	RS422-RX-P	J2	RS422-RX-N	J3	RS422-TX-P	J4	IO_0
J5	GND	J6	RS422-TX-N	J7	IO_1	J8	GND

辅助接口(RJ45)连接器管脚示意图





注①：在双 BASE 模式下生效。

1.3. 特性

- 支持 Base，medium，full 模式

- 支持 Deca(80bit)模式(8tap/10bit, 10tap/8bit)
- 支持双 BASE 模式（单卡支持两个 BASE 模式 CameraLink 设备）
- 接口时钟频率可配置
- 图像帧频可配置
- 图像分辨率可配置
- 板载 512MB 缓冲区
- 支持多种缓冲区模式
- 支持多种触发源，触发模式可配置
- 发送时序可配置

1.4. 详细描述

HT1051 系列 CameraLink 图像数据注入卡支持 CameraLink 标准协议，支持 Monochrome/RGB 模式图像注入功能。同时，它还支持一系列扩展功能。

1.4.1. Base|Medium|Full 模式

HT1051 系列 CameraLink 图像数据注入卡接口支持 Base/Medium/Full 三种配置模式，可通过 `CAMERALINK_SetMode` 接口进行配置。

Base 接口模式下设备使用 SDR1 接口，Medium/Full 接口模式下，设备使用 SDR1 和 SDR2 接口。

Base/Medium/Full 模式下，图像数据注入卡还支持通过 `CAMERALINK_CmlSetPattern` 接口配合 `CAMERALINK_SetColorBitDeep` 接口支持多种位数的黑白单色图像或 RGB 彩色图像数据的注入。

1.4.2. 80bit 模式

80bit 模式也被称之为 Deca 模式或是 Full+模式。有两种 80bit 配置模式：10-tap/8-bit 模式和 8-tap/10-bit 模式。

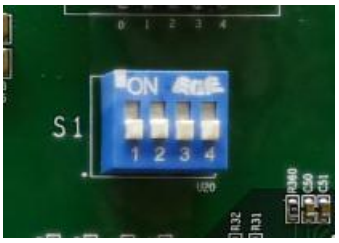
80bit 模式接口使用全部两个 SDR 连接器，关于 80bit 模式的详细说明可参考 CameraLink 接口标准 2.0 版本。

1.4.3. 双 Base 模式

HT1051 系列 CameraLink 图像数据注入卡支持双 Base 模式。双 Base 模式下，单 HT1051 注入卡可以作为两个 Base 模式的 CameraLink 注入设备使用。此种模式下，设备 1 使用 SDR1 接口，序列号为板卡序列号，设备 2 使用 SDR2 接口，序列号为板卡序列号+1。

双 Base 模式使用板卡上的 S1 拨码开关设置。S1 拨码开关共有 4 位，其中位 1 用于设置双 Base 模式，位 1 设置为 ON 表示硬件为双 Base 模式，为 OFF 表示为标准模式。位 2~位 4 保留。注意：设置位 1 后需要重启系统生效。

注：HT1051A 不支持双 BASE 模式。



S1 拨码开关

1.4.4. 接口频率

HT1051 系列 CameraLink 图像数据注入卡支持软件设置接口频率。可配置的接口频率范围为 10Mhz~ 85Mhz，调用 CAMERALINK_SetFrequecy 接口配置想要的接口频率。

注意：由于物理器件的限制，接口频率并不能完全随意设置，单板支持约三十万个频点设置，如果用户设置的频点不在这三十多万个频点中，则单板会选择一个最接近的频点生效。实际设置的频点可调用 CAMERALINK_GetFrequecy 接口得到。

1.4.5. 帧频

HT1051 系列 CameraLink 图像数据注入卡支持软件设置图像的发送帧频，通过调用 CAMERALINK_SetFrameRate 接口实现。具体使用方法请参照 CAMERALINK_SetFrameRate 函数的接口说明。

1.4.6. 分辨率

用户可以通过调用 CAMERALINK_SetFrameSize 接口来设置图像数据的分辨率，这个接口接收图像的 Width 和 Height 作为参数。

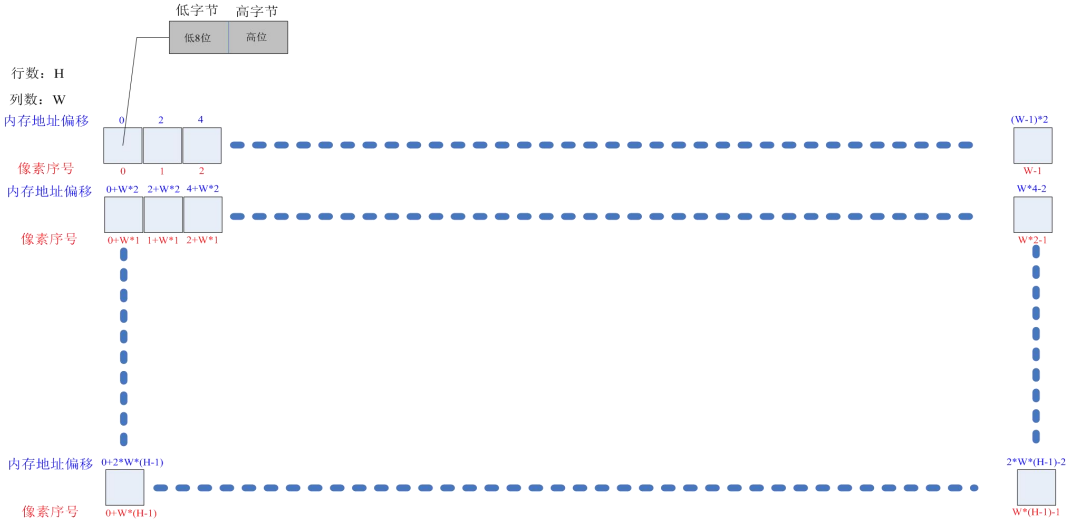
注意：因为注入卡的单幅图像缓冲区最大为 64M 字节，所以图像数据的大小也被限制在 64M 字节之内。图像数据的大小可以通过分辨率，颜色格式和颜色位深度进行计算。

如对于 1920*1080，RGB888 的图像数据，其大小为 1920*1080*3 = 6220800 字节。对于 1920 *1080, RGB101010 或是 RGB121212 的图像数据，其大小为 1920*1080*6 = 12441600 字节。同样的分辨率下，对于单色 8 位图像数据，其大小为 1920*1080 = 2073600 字节。而对于单色 10 位，12 位，14 位和 16 位图像数据，其大小均为 1920*1080 * 2 = 4147200 字节。

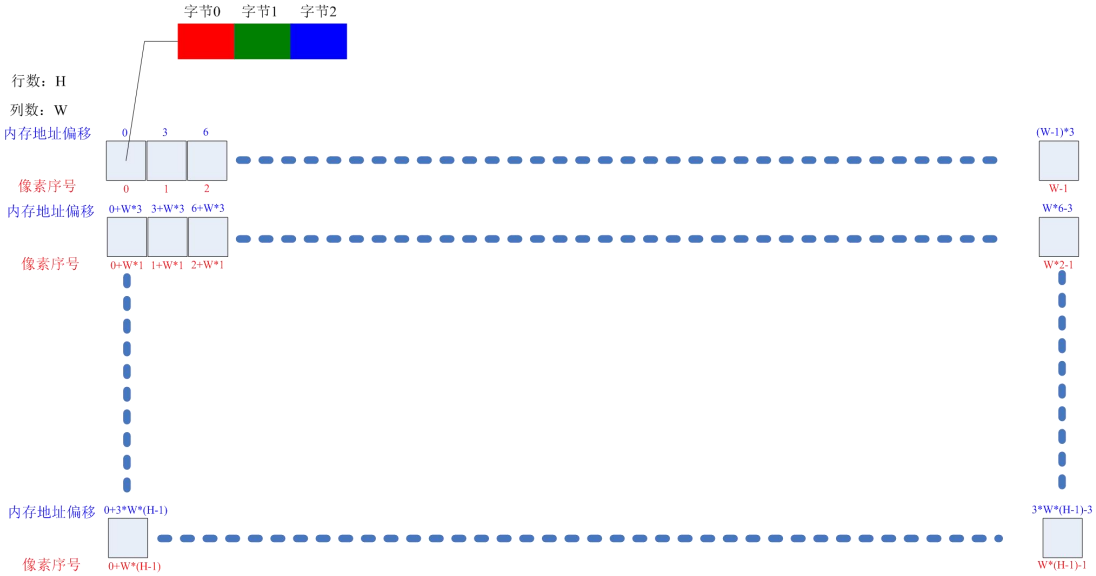
1.4.7. 颜色模式与位数

HT1051 系列 CameraLink 图像注入卡支持两种颜色模式：MONO(单色)与 RGB(彩色)。在 MONO 模式下，数据位宽支持 8 位，10 位，12 位，14 位与 16 位。它在发送缓冲区中的排列如下：

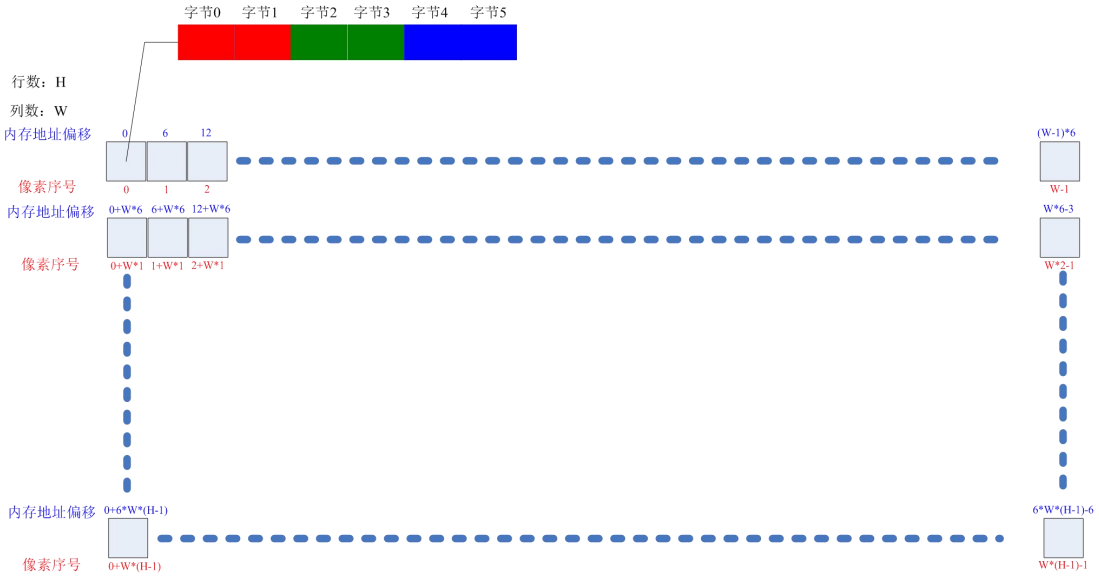
MONO8 图像格式数据排列



MONO10、MONO12、MONO14 与 MONO16 图像格式数据排列



RGB888 图像格式数据排列



RGB101010、RGB121212 图像格式数据排列

HT1051 图像数据注入卡发送数据时从缓冲区每个页的偏移地址 0 开始发送，由低到高依次发送。

1.4.8. 缓冲区

HT1051 图像数据注入卡板载 512M Byte FIFO 数据缓冲区，用于缓冲主机下发的图像数据。这些缓冲区可依据用户配置分割为数个大小为 512KB 到 2048MB（以 2 倍为步进）的独立缓冲页（HT1051 支持 512K~64M， HT1051A 支持 512K~2048M），每个页可保存一幅图像数据。用户可依据图像数据大小来配置缓冲页分割。详细的说明参见 `EN_MEM_PAGE_SIZE` 枚举和 `VI_SetMemPageSize` 函数的说明。

板卡在发送 FIFO 缓冲区中的图像数据帧时，支持三种发送模式：

- 正常模式(Normal Mode)

  此模式下，只要 FIFO 缓冲区中存在有效的图像数据帧，板卡就会依据设定好的参数发送有效的图像数据帧，没有有效数据帧时，板卡停止发送。每个数据帧发且仅发送一次。

- 最终帧循环模式(Last Repeat Mode)

  此模式下，只要 FIFO 缓冲区中存在有效的图像数据帧，板卡就会依据设定好的参数发送有效的图像数据帧，发送完 FIFO 中的最后一帧时，如果没有新的数据帧进入 FIFO，则板卡会重复发送最后一帧有效数据。

- 旁路模式(Bypass Mode)

  此模式下，在任何时刻，板卡都会发送最新的一帧有效图像数据。在软件写图像数据的速率大于硬件发送速率的情况下发送图像时，会丢弃未发送的旧图像帧，发送最新的图像帧。

1.4.9. 触发

HT1051 系列 CameraLink 图像注入卡支持 6 种触发源触发图像数据发送，包括自动触发，CC1~CC4 源触发，RS422 源触发和 IO1~IO2 源触发。

注意：双 BASE 模式下，SDR2 上的 CCx+和 CCx-信号处于 NC 状态，因此双 BASE 模式下使用 SDR2 接口的设备不支持 CC1~CC4 源触发。

除自动触发模式为内部触发以外，其它触发源均为外部触发。外部触发支持上升沿触发和下降沿触发两种模式可配。

自动触发是板卡默认的触发源，它依据使用者设置的帧率自动产生触发信号，触发硬件发送图像帧。

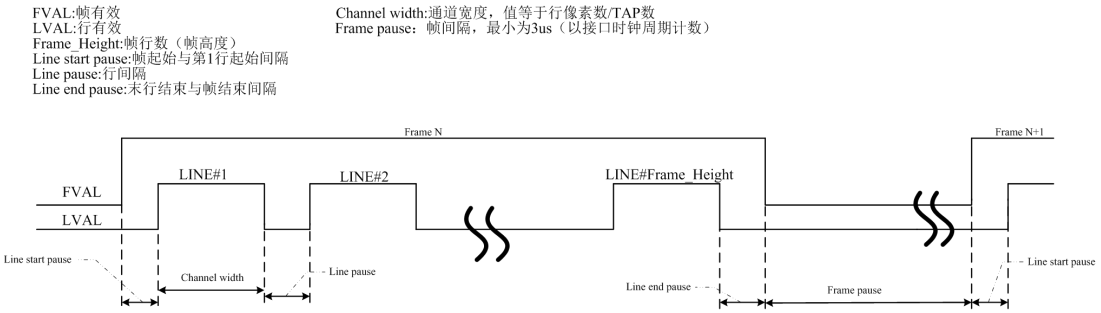
CC1~CC4 触发源是 SDR1 接口上的 CC1~CC4 信号，设置选择此 4 个触发源之一作为板卡触发源后，板卡收到对应管脚上的触发信号后，经过一段延时（此延时用户可以设置，但最小需要 4us）之后，触发硬件发送图像数据。双 Base 模式下使用 SDR2 接口的设备不支持这四个触发源。

辅助接口(RJ45)上的 RS422 和 IO1，IO2 信号也可以作为板卡的触发源，板卡收到对应管脚上的触发信号后，经过一段延时（此延时用户可以设置，但最小需要 4us）之后，触发硬件发送图像数据。双 Base 模式下，这三个触发信号会同时触发两个设备发送图像数据。

1.4.10. 发送时序

HT1051 支持自动触发和非自动触发两种发送时序，且两种发送时序的参数用户都可以配置。

自动触发时序图如下



自动触发时序下，用户可以设置图像帧率，分频率，TAP 数，帧起始与第一行起始间隔，行间隔，末行结束与帧结束间隔参数。通过设置这些参数，可以影响帧有效信号的宽度。

$$\mathrm{FVAL} = LSP + LEP + LP*(\mathrm{H}-1) + \frac{W}{TAP}*\mathrm{H}$$

其中

FVAL 为帧有效信号宽度，单位：接口时钟个数，如 FVAL=5，表示 FVAL 有效信号持续 5 个接口时钟周期，下同。

LSP 为帧起始与第一行起始间隔，单位：接口时钟个数。

LEP 为末行结束与帧结束间隔，单位：接口时钟个数。

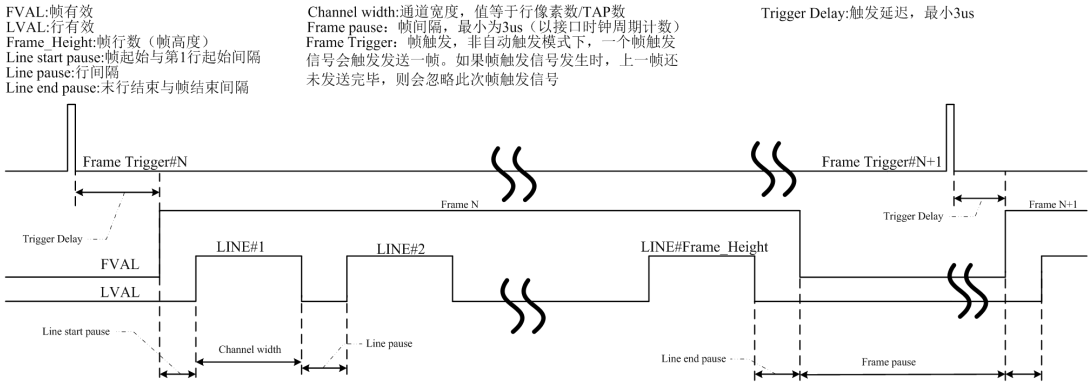
LP 为行间隔，单位：接口时钟个数。

H 为图像行数，无量纲。

W 为每行图像像素数。

TAP 为每个接口时钟周期接口传输的像素个数。

非自动触发时序图如下：



非自动触发时序下，板卡收到一次触发信号就触发一次图像数据传输。如果收到触发信号时，前一次的图像传送还未完成，则此触发信号会被丢弃。

用户可以自定义触发延迟时间，但最小为 4us。

1.5. 一般规格

- 支持 PCI Express 2.0 规范
- 物理尺寸：长×宽： 167×110mm
- 连 接 器： 2 个 SDR26、1 个 RJ45
- 工作电源： 5V
- 工作温度：-20°～+70° C
- 存储温度：-40°～+85° C
- 相对湿度： 5～95%，无凝结

1.6. 产品安装

1.6.1. 硬件安装

板卡 PCIE 接口为 PCIEx4 接口，支持插入主板的 PCI-E x4，x8 及 x16 插槽。

1.6.2. 驱动安装

HT1051 CameraLink 图像数据注入卡提供 Windows 下的驱动一体化安装包。安装好板卡硬件后，双击驱动程序安装文件，然后依据软件提示安装即可。安装后，依据系统配置，操作系统可能会要求重启，重启后可以在设备管理器中看到安装好的设备。

1.6.3. SDK 文件

SDK 主要包含头文件如下：

vi_cml_api.h
vi_errno.h
vi_types.h
vi_def.h
vi_osspec.h
vi_stuct.h
SDK 包含下列库文件：
32 位
libviapi_x86.dll
libviapi_x86.lib
64 位
libviapi_x64.dll
libviapi_x64.lib
这些头文件和库文件主要用于应用程序开发。

2. 宏定义

2.1. 函数返回值

Api 函数返回值统一使用 VI_STATUS 枚举值定义。
返回值定义如下：
VI_SUCCESS
成功
VI_OS_ERROR
系统错误
VI_LOW_MEMORY
内存不足
VI_INVALID_MSG_ID
无效的消息 ID
VI_BAD_PARAMETER_1
VI_BAD_PARAMETER_2
VI_BAD_PARAMETER_3
VI_BAD_PARAMETER_4
VI_BAD_PARAMETER_5
VI_BAD_PARAMETER_6
VI_BAD_PARAMETER_7
VI_BAD_PARAMETER_8
VI_BAD_PARAMETER_9
无效的参数 1...9
VI_NULL_DESCRIPTOR
空描述符
VI_NO_BOARD
板卡打开失败
VI_NOT_OPEN
板卡未打开
VI_READ_TIMEOUT
读操作超时
VI_SIGNALED
收到系统信号
VI_FUNCTION_UNSUPPORTED
函数功能不支持
VI_CML_PORTSIZE_CANT_DIV_BPP
像素字节数与端口数不匹配
VI_CML_TAP_CANT_DIV_COL
用户设置的列数不能被设置的 TAP 数整除
VI_CML_FRAME_TOO_BIG
帧过大或是端口数过小
VI_CML_WRONG_BIT_PER_PIXEL

错误的像素位数（可能为 0）

VI_CML_WRONG_COLOR_BIT_DEEP

错误的颜色位深度（颜色位深度支持设置为 8，10，12，14，16）

VI_CML_WRONG_TAP_COUNT

错误的 TAP 数，通常是因为 TAP 数设置过大

VI_CML_WRONG_MODE

错误的模式

VI_CML_FRAME_SIZE_UNSUPPORT

设置的帧大小不支持

3. 枚举

3.1. 缓冲区页大小枚举定义

```
typedef enum {
    EN_MEM_PAGE_512K = 0,
    EN_MEM_PAGE_1M,
    EN_MEM_PAGE_2M,
    EN_MEM_PAGE_4M,
    EN_MEM_PAGE_8M,
    EN_MEM_PAGE_16M,
    EN_MEM_PAGE_32M,
    EN_MEM_PAGE_64M,
    EN_MEM_PAGE_512,
    EN_MEM_PAGE_1K,
    EN_MEM_PAGE_2K,
    EN_MEM_PAGE_4K,
    EN_MEM_PAGE_8K,
    EN_MEM_PAGE_16K,
    EN_MEM_PAGE_32K,
    EN_MEM_PAGE_64K,
    EN_MEM_PAGE_128M,
    EN_MEM_PAGE_256M,
    EN_MEM_PAGE_512M,
    EN_MEM_PAGE_1024M,
    EN_MEM_PAGE_2048M,
}EN_MEM_PAGE_SIZE;
```

成员定义

EN_MEM_PAGE_512K: 512K Bytes

EN_MEM_PAGE_1M: 1M Bytes

EN_MEM_PAGE_2M: 2M Bytes

EN_MEM_PAGE_4M: 4M Bytes

EN_MEM_PAGE_8M：8M Bytes

EN_MEM_PAGE_16M：16M Bytes

EN_MEM_PAGE_32M：32M Bytes

EN_MEM_PAGE_64M：64M Bytes

EN_MEM_PAGE_512：512 Bytes

EN_MEM_PAGE_1K：1K Bytes

EN_MEM_PAGE_2K：2K Bytes

EN_MEM_PAGE_4K：4K Bytes

EN_MEM_PAGE_8K：8K Bytes

EN_MEM_PAGE_16K：16K Bytes

EN_MEM_PAGE_32K：32K Bytes

EN_MEM_PAGE_64K：64K Bytes

EN_MEM_PAGE_128M：128M Bytes

EN_MEM_PAGE_256M：256M Bytes

EN_MEM_PAGE_512M：512M Bytes

EN_MEM_PAGE_1024M：1024M Bytes

EN_MEM_PAGE_2048M：2048M Bytes

3. 2. CameraLink 模式

```
typedef enum CAMERALINK_PORT_MODE_EN {
     EN_CAMLINK_BASE_MEDIUM_FULL_MODE,
     EN_CAMLINK_10TAP_8BIT_MODE,
     EN_CAMLINK_8TAP_10BIT_MODE,
     EN_CAMLINK_LITE_MODE,
     EN_CAMLINK_BASE_MODE,
     EN_CAMLINK_MEDIUM_MODE,
     EN_CAMLINK_FULL_MODE,
     EN_CAMLINK_BOTTOM,
}EN_CAMLINK_PORT_MODE
```

成员定义

EN_CAMLINK_BASE_MEDIUM_FULL_MODE：BASE， MEDIUM， FULL 模式，cameralink 最常用的模式。用户设置此模式时视为设置了 FULL 模式。

EN_CAMLINK_10TAP_8BIT_MODE：10 tap 8 bit 模式

EN_CAMLINK_8TAP_10BIT_MODE：8 tap 10 bit 模式

EN_CAMLINK_LITE_MODE：简易模式

EN_CAMLINK_BASE_MODE：base 模式

EN_CAMLINK_MEDIUM_MODE：medium 模式

EN_CAMLINK_FULL_MODE：FULL 模式

3.3. CameraLink 颜色模式

```
typedef enum  CAMERALINK_COLOR_PATTERN {
    MONOCHROME_PATTERN = 0,
    RGB_PATTERN,
}EN_CAMLINK_COLOR_PATTERN;
```

成员定义

MONOCHROME_PATTERN：单色模式，与设置的颜色位深度值可组成 MONO8，MONO10，MONO12，MONO14，MONO16。

RGB_PATTERN：RGB 彩色模式，与设置的颜色位深度值可组成 RGB888，RGB101010，RGB121212

3.4. CameraLink 颜色位深度

```
typedef enum  CAMERALINK_BIT_PER_COLOR {
    BIT_8_PER_COLOR = 8,
    BIT_10_PER_COLOR = 10,
    BIT_12_PER_COLOR = 12,
    BIT_14_PER_COLOR = 14,
    BIT_16_PER_COLOR = 16,
}EN_CAMLINK_BIT_PER_COLOR;
```

成员定义

BIT_8_PER_COLOR：颜色位深度为 8，与颜色模式配合可组成 MONO8，RGB888 两种图像格式。

BIT_10_PER_COLOR：颜色位深度为 10，与颜色模式配合可组成 MONO10，RGB101010 两种图像格式。

BIT_12_PER_COLOR：颜色位深度为 12，与颜色模式配合可组成 MONO12，RGB121212 两种图像格式。

BIT_14_PER_COLOR：颜色位深度为 14，与颜色模式配合可组成 MONO14 图像格式。

BIT_16_PER_COLOR：颜色位深度为 16，与颜色模式配合可组成 MONO16 图像格式。

3.5. 缓冲区模式

```
typedef enum  buffer_mode {
    FIFO_NORMAL_MODE = 0,
    FIFO_LAST_REPEAT_MODE,
    FIFO_BYPASS_MODE,
}ENUM_BUFFER_MODE;
```

成员定义

FIFO_NORMAL_MODE：正常缓冲区模式。FIFO 有数据帧就进行发送，没有数据帧时停止发送。

FIFO_LAST_REPEAT_MODE：终帧循环模式。FIFO 有数据帧就进行发送，没有数据帧时，重复发送最后一帧有效数据帧。

FIFO_BYPASS_MODE：旁路模式。任何时刻，发送最新的图像帧，在软件写的速率大于硬件发送速率时，发送图像时，会丢弃未发送的旧的图像帧，发送最新的图像帧。如果没有新帧，则重复发送上一帧。

3.6. 触发源

```
typedef enum  trigger_src {
     AUTO_TRIGGER_SRC = 0,
     CC_1_TRIGGER_SRC,
     CC_2_TRIGGER_SRC,
     CC_3_TRIGGER_SRC,
     CC_4_TRIGGER_SRC,
     RS422_TRIGGER_SRC,
     IO_0_TRIGGER_SRC,
     IO_1_TRIGGER_SRC,
}ENUM_TRIGGER_SRC;
```

成员定义

AUTO_TRIGGER_SRC：自动触发模式（设备默认的触发模式）。

CC_1_TRIGGER_SRC：CC_1 信号触发。

CC_2_TRIGGER_SRC：CC_2 信号触发。

CC_3_TRIGGER_SRC：CC_3 信号触发。

CC_3_TRIGGER_SRC：CC_3 信号触发。

RS422_TRIGGER_SRC：RS422 信号触发。

IO_0_TRIGGER_SRC：IO_0 信号触发。

IO_1_TRIGGER_SRC：IO_1 信号触发。

3.7. 触发模式

```
typedef enum  trigger_mode {
     POSEDGE_TRIGGER = 0,
     NEGEDGE_TRIGGER,
}ENUM_TRIGGER_MODE;
```

成员定义

POSEDGE_TRIGGER：上升沿信号触发。

NEGEDGE_TRIGGER：下降沿信号触发。

4. 结构体

无。

5. API 说明

5.1. 打开/关闭设备接口

HT1051 系列 CameraLink 图像注入卡支持通过序号和序列号两种方式打开板卡。

5.1.1. CAMERALINK_Open

原型

```
VI_STATUS CAMERALINK_Open(VIHANDLE  * ph, VI_UINT8 id)
```

功能

通过序号打开 CameraLink 图像注入卡。

参数说明

输入参数

ph：板卡句柄指针。
id：设备编号，取值为 1~255。

输出参数

ph：打开后的板卡句柄指针。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_1
VI_LOW_MEMORY
VI_NOT_OPEN

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_Open(&ph, 1);
if (status != VI_SUCCESS) {
     return status;
}
```

5.1.2. CAMERALINK_OpenCardBySerialNumber

原型

```
VI_STATUS CAMERALINK_OpenCardBySerialNumber(VIHANDLE  * ph, VI_UINT32 sn)
```

功能

通过序列号打开 CameraLink 图像注入卡。

参数说明

输入参数

ph：板卡句柄指针。
sn：序列号，形如 20220102 的 8 位数字，在板卡标签上可以查到。

输出参数

ph：打开后的板卡句柄指针。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_1
VI_LOW_MEMORY
VI_NOT_OPEN

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
```

5.1.3. CAMERALINK_Close

原型

```
VI_STATUS CAMERALINK_Close(VIHANDLE  * ph)
```

功能

关闭已打开的 CameraLink 图像注入卡句柄指针。关闭设备后，句柄不能再被使用。

参数说明

输入参数

ph：板卡句柄指针，指向已经打开的图像注入卡。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_1
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
.....
status = CAMERALINK_Close(&ph);
if (status != VI_SUCCESS) {
     return status;
}
```

5.2. 通用配置接口

5.2.1. VI_Reset

原型

```
VI_STATUS VI_Reset(VIHANDLE ph)
```

功能

复位设备。

参数说明

输入参数

ph：板卡句柄。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_Reset(ph);               //复位设备
if (status != VI_SUCCESS) {
     return status;
}
```

5.2.2. VI_SetEnable

原型

```
VI_STATUS VI_SetEnable(VIHANDLE ph, VI_BOOL enable)
```

功能

设置设备是否使能。

参数说明

输入参数

ph：板卡句柄。
enable： VI_TRUE - 使能设备， VI_FALSE - 禁用设备。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：

VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);              //使能设备
if (status != VI_SUCCESS) {
     return status;
}
```

5. 2. 3. VI_GetEnable

原型

```
VI_STATUS VI_GetEnable(VIHANDLE ph, VI_BOOL * enable)
```

功能

获取设备是否已使能。

参数说明

输入参数

ph：板卡句柄。

输出参数

enable：VI_TRUE - 使能设备， VI_FALSE - 禁用设备。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
VI_BOOL enable = VI_FALSE;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
```

```
if (status != VI_SUCCESS) {
     return status;
}
status = VI_GetEnable(ph, &enable);              //获取设备是否已经使能
if (status != VI_SUCCESS) {
     return status;
}
```

5. 2. 4. VI_SetMemPageSize

原型

```
VI_STATUS VI_SetMemPageSize(VIHANDLE ph, EN_MEM_PAGE_SIZE size)
```

功能

设置设备缓存页大小。

参数说明

输入参数

ph：板卡句柄。
size：参见 EN_MEM_PAGE_SIZE 定义。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetMemPageSize(ph, EN_MEM_PAGE_512K);              //设置设备的缓存页大小为512K
if (status != VI_SUCCESS) {
     return status;
}
```

5. 2. 5. VI_GetMemPageSize

原型

```
VI_STATUS VI_GetMemPageSize(VIHANDLE ph, EN_MEM_PAGE_SIZE * size)
```

功能

获取当前配置的设备缓存页大小。

参数说明

输入参数

ph：板卡句柄。

输出参数

size：参见 EN_MEM_PAGE_SIZE 定义。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
EN_MEM_PAGE_SIZE size;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_GetMemPageSize(ph, &size);                //获取设备当前配置的缓存页大小
if (status != VI_SUCCESS) {
     return status;
}
```

5. 2. 6. VI_MemPageBufferClr

原型

```
VI_STATUS VI_MemPageBufferClr(VIHANDLE ph)
```

功能

复位并清零设备缓存页。

参数说明

输入参数

ph：板卡句柄。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_MemPageBufferClr(ph);
if (status != VI_SUCCESS) {
     return status;
}
```

5. 2. 7. VI_GetRemainFrameCount

原型

```
VI_STATUS VI_GetRemainFrameCount(VIHANDLE ph, VI_UINT16 * count)
```

功能

获取缓冲区中还能容纳的帧数，如果不为 0，则表示缓冲区未满，可以写入。

参数说明

输入参数

ph：板卡句柄。

输出参数

count：剩余帧数。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
VI_UINT16 count = 0;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_GetRemainFrameCount(ph, &count);
if (status != VI_SUCCESS) {
     return status;
}
```

5.3. 通用读/写接口

5.3.1. VI_WriteFrameData

原型

```
VI_STATUS VI_WriteFrameData(VIHANDLE ph, VI_UINT8 * buf, VI_UINT32 length,
VI_UINT32 * wlength)
```

功能

向设备写入有效的数据帧。

参数说明

输入参数

ph：板卡句柄。
buf：数据缓冲区。
length：数据长度。

输出参数

wlength：实际写入的长度。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_BAD_PARAMETER_4
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
VI_UINT8 image_buf[256*256] = {/*此处需要设置为图像数据*/};
VI_UINT32 wlength;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_WriteFrameData(ph, image_buff, 256*256, &wlength);
if (status != VI_SUCCESS) {
     return status;
}
```

5.3.2. VI_ReadFrameData

原型

```
VI_STATUS VI_ReadFrameData(VIHANDLE ph, VI_UINT8 * buf, VI_UINT32 length,
VI_UINT32 * rlength)
```

功能

从设备读出有效的帧数据（最近一次写入的数据帧）。

参数说明

输入参数

ph：板卡句柄。
length：数据长度。

输出参数

buf：数据缓冲区。
rlength：实际读取的长度。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_BAD_PARAMETER_4
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
VI_UINT8 image_buf[256*256];
VI_UINT32 rlength;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_ReadFrameData(ph, image_buff, 256*256, &rlength);//执行后，image_buff 中放置了最近写入的一帧有效数据。
if (status != VI_SUCCESS) {
     return status;
}
```

5.4. 通用辅助接口

5.4.1. VI_ReadBid

原型

```
VI_STATUS VI_ReadBid(VIHANDLE ph, VI_UINT16 * id)
```

功能

读取单板 ID。

参数说明

输入参数

ph：板卡句柄。

输出参数

id：单板 ID。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
VI_UINT16 id;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_ReadBid(ph, &bid);
if (status != VI_SUCCESS) {
     return status;
}
```

5. 4. 2. VI_ReadHwVersion

原型

```
VI_STATUS VI_ReadHwVersion(VIHANDLE ph, VI_UINT16 * hw)
```

功能

读取单板硬件版本号。

参数说明

输入参数

ph：板卡句柄。

输出参数

hw：硬件版本号。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_OS_ERROR

VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
VI_UINT16 hw;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_ReadHwVersion(ph, &hw);
if (status != VI_SUCCESS) {
     return status;
}
```

5. 4. 3. VI_SdkVersion

原型

```
VI_STATUS VI_SdkVersion(VIHANDLE ph, VI_UINT8 * version)
```

功能

获取 Sdk 版本号，版本号占用 64 个字符。

参数说明

输入参数

ph：板卡句柄。

输出参数

version：获取到的版本号，空间由调用者分配，不小于 64 个字符。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
VI_UINT8 version[64];
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
```

```
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SdkVersion(ph, version);
if (status != VI_SUCCESS) {
     return status;
}
```

5. 4. 4. VI_DrvVersion

原型

```
VI_STATUS VI_SdkVersion(VIHANDLE ph, VI_UINT8 * version)
```

功能

获取驱动版本号，版本号占用 64 个字符。

参数说明

输入参数

ph：板卡句柄。

输出参数

version：获取到的版本号，空间由调用者分配，不小于 64 个字符。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
VI_UINT8 version[64];
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_DrvVersion(ph, version);
if (status != VI_SUCCESS) {
     return status;
}
```

5. 4. 5. VI_GetSN

原型

```
VI_STATUS VI_GetSN(VIHANDLE ph, VI_UINT32 * snddd)
```

功能

获取设备序列号。

参数说明

输入参数

ph：板卡句柄。

输出参数

sn：获取到的序列号，形如“22010003”的 8 位数字。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_OS_ERROR
VI_NULL_DESCRIPTOR
VI_NO_BOARD

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
VI_UINT32 sn;
status = CAMERALINK_Open(&ph, 1);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_GetSN(ph, &SN);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5. 专用接口

5.5.1. CAMERALINK_Initial

原型

```
VI_STATUS CANERALINK_Initial(VIHANDLE ph)
```

功能

初始化 CameraLink 接口注入卡。此函数应在 CameraLink 其它配置接口调用后，在使能设备之前调用。

参数说明

输入参数

ph：板卡句柄。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
```

```
    return status;
}
```

注意：此函数会依据用户预先设置的帧率，分辨率，tap 数，颜色模式，颜色位深度，mode，频率，触发等其它参数进行计算以正确初始化接口。如果用户设置的参数可以实现，则会依照用户设置的参数初始化接口，如果用户设置的参数有问题，则会在不影响用户使用的前提下对用户设置的参数进行微调，如用户设置的参数无法微调，则会返回失败错误码。

5. 5. 2. CAMERALINK_SetFrequency

原型

```
VI_STATUS CAMERALINK_SetFrequecy(VIHANDLE ph, VI_UINT32 freq)
```

功能

设置 CameraLink 接口频率。

参数说明

输入参数

ph：板卡句柄。
freq: 设置的频率，如 1MHz 设置为 1000000。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 freq = 60000000;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
    return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetFrequency(ph, freq);     //设置接口时钟频率为60M
```

```
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

注意：由于客观情况限制，并非所有的频率都可以设置。调用此接口后，板卡检测频点是否支持，如果不支持，则会使用所支持的与其最接近的频点进行设置。实际的频点可由 CAMERALINK_GetFrequecy 函数获取。

5. 5. 3. CAMERALINK_GetFrequency

原型

```
VI_STATUS CAMERALINK_GetFrequecy(VIHANDLE ph, VI_UINT32 * freq)
```

功能

获取 CameraLink 接口频率。

参数说明

输入参数

ph：板卡句柄。

输出参数

freq: 获取的频率，如获取到 10000000 则表示频率为 10MHz。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 freq = 60000000;
```

```
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetFrequency(ph, freq);      //设置接口时钟频率为60M
if (status != VI_SUCCESS) {
     return status;
}
/*等待数秒钟*/
...
status = CAMERALINK_GetFrequency(ph, &freq);          //获取实际设置的接口时钟频率
if (status != VI_SUCCESS) {
     return status;
}
```

注意：使用 CAMERALINK_SetFrequecy 接口设置频率后，设置的频率会迅速生效。
但使用此接口获取频率，依据设置的频率大小，需要等待约 1 到 10 秒（因为测量频率需要一些时间，设置的频率越高，等待时间越短），才能获取到稳定的实时频率值。

5. 5. 4. CAMERALINK_SaveFrequency

原型

```
VI_STATUS CAMERALINK_SaveFrequecy(VIHANDLE ph)
```

功能

保存当前设置的 CameraLink 接口频率到 FLASH 中。

参数说明

输入参数

ph：板卡句柄。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD

VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 freq = 60000000;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetFrequency(ph, freq);      //设置接口时钟频率为60M
if (status != VI_SUCCESS) {
     return status;
}
/*如果设置的频率值需要下电保存，则调用保存频率接口*/
status = CAMERALINK_SaveFrequency(ph);
if (status != VI_SUCCESS) {
     return status;
}
```

注意：由于 FLASH 擦写次数（10000 次）限制，过多次数的保存频率到 FLASH 中的操作会影响 FLASH 芯片的寿命，请用户谨慎调用此接口。

5. 5. 5. CAMERALINK_SetLineStartPause

原型

```
VI_STATUS CAMERALINK_SetLineStartPause(VIHANDLE ph, VI_UINT32 lsp)
```

功能

设置 CameraLink 数据帧时序的帧起始行前间隔（Line Start Pause）参数。

参数说明

输入参数

ph：板卡句柄。
lsp：要设置的帧起始行前间隔参数，单位为接口输出时钟的 1 个时钟周期。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 lsp= 6;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetLineStartPause(ph, lsp);       //设置帧起始行前间隔为6个时钟
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5. 5. 6. CAMERALINK_GetLineStartPause

原型

```
VI_STATUS CAMERALINK_GetLineStartPause(VIHANDLE ph, VI_UINT32 * lsp)
```

功能

获取设置的 CameraLink 数据帧时序的帧起始行前间隔（Line Start Pause）参数。

参数说明

输入参数

ph：板卡句柄。

输出参数

lsp：获取到的帧起始行前间隔参数，单位为接口输出时钟的 1 个时钟周期。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 lsp;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_GetLineStartPause(ph, &lsp);      //获取帧起始行前间隔时钟数
if (status != VI_SUCCESS) {
     return status;
}
```

5. 5. 7. CAMERALINK_SetLinePause

原型

```
VI_STATUS CAMERALINK_SetLinePause(VIHANDLE ph, VI_UINT32 lp)
```

功能

设置 CameraLink 数据帧时序的行间隔（Line Pause）参数。

参数说明

输入参数

ph：板卡句柄。
lp：要设置的行间隔参数，单位为接口输出时钟的 1 个时钟周期。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。

失败时可能返回的返回值列表：

VI_NULL_DESCRIPTOR

VI_NO_BOARD

VI_OS_ERROR

函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 lp= 6;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetLinePause(ph, lp);        //设置行间隔为6个时钟
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5. 5. 8. CAMERALINK_GetLinePause

原型

```
VI_STATUS CAMERALINK_GetStartPause(VIHANDLE ph, VI_UINT32 * lp)
```

功能

获取设置的 CameraLink 数据帧时序的行间隔（Line Pause）参数。

参数说明

输入参数

ph：板卡句柄。

输出参数

lp：获取到的行间隔参数，单位为接口输出时钟的 1 个时钟周期。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 lp;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_GetLinePause(ph, &lp);       //获取行间隔时钟数
if (status != VI_SUCCESS) {
     return status;
}
```

5. 5. 9. CAMERALINK_SetLineEndPause

原型

```
VI_STATUS CAMERALINK_SetLineEndPause(VIHANDLE ph, VI_UINT32 lep)
```

功能

设置 CameraLink 数据帧时序的帧最后行后间隔（Line End Pause）参数。

参数说明

输入参数

ph：板卡句柄。
lep：要设置的帧最后行后间隔参数，单位为接口输出时钟的 1 个时钟周期。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。

失败时可能返回的返回值列表：

VI_NULL_DESCRIPTOR

VI_NO_BOARD

VI_OS_ERROR

函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 lep= 6;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetLineEndPause(ph, lep);          //设置帧最后行行后间隔为6个时钟
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.10. CAMERALINK_GetLineEndPause

原型

```
VI_STATUS CAMERALINK_GetStartEndPause(VIHANDLE ph, VI_UINT32 * lep)
```

功能

获取设置的 CameraLink 数据帧时序的帧最后行后间隔（Line End Pause）参数。

参数说明

输入参数

ph：板卡句柄。

输出参数

lep：获取到的帧最后行后间隔参数，单位为接口输出时钟的 1 个时钟周期。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 lep;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_GetLineEndPause(ph, &lp);         //获取帧最后行行后间隔时钟数
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.11. CAMERALINK_SetTriggerDelay

原型

```
VI_STATUS CAMERALINK_SetTriggerDelay(VIHANDLE ph, VI_UINT32 dly)
```

功能

设置 CameraLink 帧时序的触发延时参数。

参数说明

输入参数

ph：板卡句柄。
dly：要设置的触发延时，单位:us。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。

失败时可能返回的返回值列表：

VI_NULL_DESCRIPTOR

VI_NO_BOARD

VI_OS_ERROR

函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 dly= 6;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetTriggerDelay(ph, dly);          //设置触发延时参数6us
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

注意：1.此函数在自动触发(即内触发)模式下不会生效。

2.设置的触发延时的计算与接口时钟频率相关，所以如果改变了接口时钟频率，需要重新设置触发延时，否则会导致触发延时的计算不正确。

5.5.12. CAMERALINK_GetTriggerDelay

原型

```
VI_STATUS CAMERALINK_GetTriggerDelay(VIHANDLE ph, VI_UINT32 * dly)
```

功能

获取当前的 CameraLink 帧时序的触发延时。

参数说明

输入参数

ph：板卡句柄。

输出参数

dly：获取到的触发延时，单位：us。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_BAD_PARAMETER_2
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VI_UINT32 dly;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_GetTriggerDly(ph, &lp);      //获取设置的帧触发延时参数
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.13. CAMERALINK_SetMode

原型

```
VI_STATUS CAMERALINK_SetMode(VIHANDLE ph, EN_CAMLINK_PORT_MODE mode)
```

功能

设置 CameraLink 端口模式。

参数说明

输入参数

ph：板卡句柄。
mode：设置的端口模式，参见 EN_CAMLINK_PORT_MODE 定义。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
VI_BAD_PARAMETER_2
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
EN_CAMLINK_PORT_MODE mode = EN_CAMLINK_BASE_MEDIUM_FULL_MODE;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetMode(ph, mode );
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.14. CAMERALINK_GetMode

原型

```
VI_STATUS CAMERALINK_GetMode(VIHANDLE ph, EN_CAMLINK_PORT_MODE * mode)
```

功能

获取 CameraLink 端口模式。

参数说明

输入参数

ph：板卡句柄。

输出参数

mode：获取的端口模式，参见 EN_CAMLINK_PORT_MODE 定义。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_BAD_PARAMETER_1
VI_BAD_PARAMETER_2
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
EN_CAMLINK_PORT_MODE mode;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_GetMode(ph, &mode);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.15. CAMERALINK_SetFrameRate

原型

```
VI_STATUS CAMERALINK_SetFrameRate(VIHANDLE ph, double fr)
```

功能

设置 Cameralink 视频控制器帧率。如果用户设置的帧频为 0，则系统会依据用户设置的其它参数自动算出适合的最大帧频。

参数说明

输入参数

ph：板卡句柄。
fr: 帧率。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
float fr = 50;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetFrameRate(ph, fr);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.16. CAMERALINK_GetFrameRate

原型

```
VI_STATUS CAMERALINK_GetFrameRate(VIHANDLE ph, double * fr)
```

功能

获取 Cameralink 视频控制器当前帧率。注意：CAMERALINK_Initial 函数有可能会依据用户设置的各种参数对实际频率进行微调，用户若要调用此函数获取实际得到的帧率，则需在 CAMERALINK_Initial 函数后调用。

参数说明

输入参数

ph：板卡句柄。

输出参数

fr: 帧率。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_BAD_PARAMETER_2
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;

VI_STATUS status = VI_SUCCESS;

float fr = 50;

VIHANDLE ph = NULL;

status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);

if (status != VI_SUCCESS) {

     return status;

}

/*执行其它配置*/

......

status = CAMERALINK_SetFrameRate(ph, fr);

if (status != VI_SUCCESS) {

     return status;

}

status = CAMERALINK_Initial(ph);

if (status != VI_SUCCESS) {

     return status;

}

status = CAMERALINK_GetFrameRate(ph, &fr);       //获取实际设置的帧率

if (status != VI_SUCCESS) {

     return status;

}
```

5.5.17. CAMERALINK_SetFrameSize

原型

```
VI_STATUS CAMERALINK_SetFrameSize(VIHANDLE ph, VI_UINT32 h, VI_UINT32 v)
```

功能

设置帧大小。

参数说明

输入参数

ph：板卡句柄。
h: 帧行像素数。
v: 帧列像素数。例如:1920*1080 的高清图像，h 应为 1920，v 应为 1080。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetFrameSize(ph, 1920, 1080);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.18. CAMERALINK_SetBitPerPixel

原型

```
VI_STATUS CAMERALINK_SetBitPerPixel(VIHANDLE ph, VI_UINT32 num)
```

功能

设置每个像素有多少位。

参数说明

输入参数

ph：板卡句柄。
num：像素位数。例如：对于单个像素 3 字节的 RGB 图像，num 值应为 24。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetBitPerByte(ph, 24);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
```

```
    return status;
}
```

5.5.19. CAMERALINK_SetTap

原型

```
VI_STATUS CAMERALINK_SetTap(VIHANDLE ph, VI_UINT8 tap)
```

功能

设置发送 tap 数。如果未调用此函数设置 TAP 数，默认 TAP 数为 1。

参数说明

输入参数

ph：板卡句柄。
tap：要设置的 tap 数。对于单 FULL 卡表示输出为设置的 tap 数，对于双 FULL 卡表示每个输出都为设置的 tap 数。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
    return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetTap(ph, 1);
if (status != VI_SUCCESS) {
    return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
```

```
    return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
    return status;
}
```

5.5.20. CAMERALINK_SetPattern

原型

```
VI_STATUS CAMERALINK_CmlSetPattern(VIHANDLE ph, EN_CAMLINK_COLOR_PATTERN pattern)
```

功能

设置颜色模式，支持 MONOCHROME 和 RGB 两种模式。

参数说明

输入参数

ph：板卡句柄。
pattern：要设置的模式。参见 EN_CAMLINK_COLOR_PATTERN 定义。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
EN_CAMLINK_COLOR_PATTERN pattern = RGB_PATTERN;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
    return status;
}
/*执行其它配置*/
......
```

```
status = CAMERALINK_SetPattern(ph, pattern);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.21. CAMERALINK_SetColorBitDeep

原型

```
VI_STATUS CAMERALINK_SetColorBitDeep(VIHANDLE ph, VI_UINT8 deep)
```

功能

设置颜色为深度，在 MONOCHROME_PATTERN 下设置为 8，则便是 MONO8，在 RGB_PATTERN 模式下设置为 8 则表示 RGB888，以此类推。

参数说明

输入参数

ph：板卡句柄。
deep：要设置的位深度。常用位深度为 8， 10， 12， 16。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
```

```
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetColorBitDeep(ph, 8);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.22. CAMERALINK_SetBufferMode

原型

```
VI_STATUS CAMERALINK_SetBufferMode(VIHANDLE ph, EN_BUFFER_MODE mode)
```

功能

设置图像缓冲区模式。缓冲区模式定义参考 ENUM_BUFFER_MODE 枚举定义。

参数说明

输入参数

ph：板卡句柄。
mode：要设置的缓冲区模式。参见 ENUM_BUFFER_MODE 枚举定义。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
ENUM_BUFFER_MODE mode = FIFO_LAST_REPEAT_MODE;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetBufferMode(ph, mode);
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.23. CAMERALINK_SetTriggerSrc

原型

```
VI_STATUS CAMERALINK_SetTriggerSrc(VIHANDLE ph, EN_TRIGGER_SRC src)
```

功能

设置图像注入触发源。触发源定义参考 ENUM_TRIGGER_SRC 枚举定义。

参数说明

输入参数

ph：板卡句柄。
src：要设置的触发源。参见 ENUM_TRIGGER_SRC 枚举定义。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。

失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
ENUM_TRIGGER_SRC src = AUTO_TRIGGER_SRC;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetTriggerSrc(ph, src);           //设置为自动触发
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```

5.5.24. CAMERALINK_SetTriggerMode

原型

```
VI_STATUS CAMERALINK_SetTriggerMode(VIHANDLE ph, EN_TRIGGER_MODE mode)
```

功能

设置图像注入触发源触发模式。触发模式定义参考 ENUM_TRIGGER_MODE 枚举定义。

参数说明

输入参数

ph：板卡句柄。
mode：要设置的触发模式。参见 ENUM_TRIGGER_MODE 枚举定义。

输出参数

无。

返回值

成功时返回 VI_SUCCESS。
失败时可能返回的返回值列表：
VI_NULL_DESCRIPTOR
VI_NO_BOARD
VI_OS_ERROR
函数也可能返回其它错误码。

Code example

```
VI_STATUS status = VI_SUCCESS;
VIHANDLE ph = NULL;
ENUM_TRIGGER_MODE mode = POSEDGE_TRIGGER;
status = CAMERALINK_OpenCardBySerialNumber(&ph, 20221201);
if (status != VI_SUCCESS) {
     return status;
}
/*执行其它配置*/
......
status = CAMERALINK_SetTriggerMode(ph, mode);              //设置为上升沿触发
if (status != VI_SUCCESS) {
     return status;
}
status = CAMERALINK_Initial(ph);
if (status != VI_SUCCESS) {
     return status;
}
status = VI_SetEnable(ph, VI_TRUE);
if (status != VI_SUCCESS) {
     return status;
}
```