

北京微视 Philips 7130/7134 图像采集卡二次开发文档：

支持本公司的 V221、V411、V412、S450 等

驱动安装说明：

查看当前系统加载的模块中是否还存在 saa7134 模块,运行 `lsmod | grep saa7134`，若有 saa7134 则运行 `rmmod saa7134` 卸载 saa7134 模块。

`insmod mv7134.ko`

接着创建设备节点：

`mknod /dev/mv713x0 c 251 0`

`mknod /dev/mv713x1 c 251 1`

`mknod /dev/mv713x2 c 251 2`

`mknod /dev/mv713x3 c 251 3`

或者也可以用脚本加载驱动创建节点，脚本为 `loaddrv`，直接运行 `./loaddrv`。

DEMO 演示程序说明

再运行程序前请先确认是否装有 gcc、qt3 等相应的工具。

QT 演示程序使用说明

在 X 下打开一终端，进入当前目录，编译为如下步骤：

qmake -project (在生成的 qmdemo.pro 文件中的最后加入如下两行：

```
CONFIG += thread
LIBS += mv_7130.a)
```

qmkae

make

运行演示程序命令如下：./qtdemo

或直接在 XWindow 下双击 qtdemo 执行程序。

- | | | | |
|------|-------------------|--|---------|
| 点击菜单 | Operation->Open | 或工具栏按钮 Open | 打开那个设备； |
| 点击菜单 | Operation->Run | 或工具栏按钮 Run | 启动采集图像； |
| 点击菜单 | Operation->Pause | 或工具栏按钮 Pause | 暂停采集图像； |
| 点击菜单 | Operation->Stop | 或工具栏按钮 Stop | 停止采集图像； |
| 点击菜单 | Operation->Save | 或工具栏按钮 Save | 采集单帧； |
| 点击菜单 | Operation->Play | 或工具栏按钮 Play | 显示图像； |
| 点击菜单 | Operation->UnPlay | 或工具栏按钮 UnPlay | 显示图像； |
| 点击菜单 | Edit->Seting | 可以设置 contrast 对比度 hue 色调 bright 亮度 saturation 饱和度 channel 通道 standard(pal/ntsc)制式； | |
| 点击菜单 | Edit->ImageSize | 可以设置图像的 WIDTH 宽度、HEIGHT 高度、ColorFormat 颜色格式、Field 帧/场； | |
| 点击菜单 | Edit->Offset | 可以设置 V-Offset 垂直、H-Offset 水平方向的偏移； | |
| 点击菜单 | Edit->Updown | 可以设置上下翻转。 | |
| 点击菜单 | Edit->LeftRight | 可以设置左右翻转。 | |

Framebuffer 演示程序使用说明

运行环境：RedHat Linux 各个版本均可。

此程序只有在 FrameBuffer 下才能正常运行，如果您完全不了解 FrameBuffer，请参见下面的步骤来开启 Linux 的 FrameBuffer 功能。

如果系统以 grub 为引导，则具体步骤如下：

1. 确保您的 PC 机显示卡是 VESA 兼容的。如果显示卡是 VESA 兼容的，就可以使用 Linux 内核中的 VESA FrameBuffer 驱动程序了。
2. 确保您的 Linux 内核包含了 FrameBuffer 支持，并包含了 VESA FrameBuffer 驱动程序。如果使用自己编译的内核，请检查您的内核配置。
3. 修改 /boot/grub/grub.conf 文件，在您所使用的内核选项段中，添加 vga=792 (1024×768×32bpp)
4. 重新启动系统。
5. 如果一切正常，将在 Linux 内核的引导过程中看到屏幕左上角出现可爱的 Linux 吉祥物--企鹅，并发现系统的显示模式发生变化。

进入到 fbdemo 的当前目录，编译 Framebuffer 的演示程序步骤如下：

```
make clean  
make
```

按 Ctrl+Alt+F5 进入文本模式，进入到 fbdemo 的所在目录，运行如下命令：

```
./fbdemo
```

按 h/H 将显示帮助信息：

o/O	打开第几路设备
k/K	开始往内存采集
t/T	停止往内存采集
a/A	开始往屏幕采集
z/Z	停止往屏幕采集
g/G	采集单帧
q/Q	退出程序
i/I	切换到场采集模式
r/R	切换到帧采集模式
b/B	设置亮度，输入范围为 0-255
s/S	设置饱和度，输入范围为 0-127
c/C	设置对比度，输入范围为 0-127
d/D	设置色调，输入范围为 0-255
f/F	设置颜色格式
w/W	设置宽度
e/E	设置高度

您可以按照以下方式运行 Demo：

1. 输入字母 0 或 O,接着输入要打开的设备号如 (0、1、2、3)；

- 2.输入字母 f 或 F,即输入颜色格式为 1(由于 vga = 792, 所以我们这里选择 1);
- 3.输入字母 k 或 K, 开始采集图像;
- 4.输入字母 a 或 A, 开始把图像拷到显存上;
- 5.输入字母 z 或 Z, 停止把图像拷到显存上;
- 6.输入字母 t 或 T, 停止采集图像;
- 7.输入字母 q 或 Q, 退出程序。

sdk 开发包说明:

在头文件 mv_btapi.h 中，自定义如下枚举类型：

```
1.enum RUNOPER{
    MVSTOP=0,      // 停止工作
    MVRUN,         // 开始工作
    MVPAUSE,       // 暂停工作
    MVQUERYSTATU, // 查询当前状态
};

2.enum RGBVECTOR
{
    FIELD_INTERLACED = 0, //帧
    FIELD_TOP        = 1, //奇场
    FIELD_BOTTOM     = 2, //偶场
};

3. enum MV_PARAMTER{
    GET_BOARD_TYPE=0,           //得到当前设备的类型
    GET_GRAPHICAL_INTERFACE,   //得到图形接口指针
    SET_GARBIMAGEINFO,         //得到当前被采集图像的信息
    SET_DISPIMAGEINFO,         //得到当前被显示图像的信息
    BUFFERTYPE,                // 控制设备使用的缓存类型
    DEFAULT_PARAM,             // 恢复或存储设备参数
    // 控制显示
    DISP_PRESENCE,    DISP_WHND,    DISP_TOP,    DISP_LEFT,
    DISP_HEIGHT,     DISP_WIDTH,
    // 控制 A/D 的颜色,通道等调节参数
    ADJUST_STANDARD, ADJUST_SOURCE, ADJUST_CHANNEL,
    ADJUST_LUMINANCE, ADJUST_CHROMINANE, ADJUST_SATURATION,
    ADJUST_HUE,       ADJUST_CONTRAST,    ADJUST_R_LUM,
    ADJUST_G_LUM, ADJUST_B_LUM,          ADJUST_R_COARSE,
    ADJUST_G_COARSE, ADJUST_B_COARSE,
    // 控制设备的捕获参数
    GARB_XOFF ( GRAB_XOFF ) , GARB_YOFF ( GRAB_YOFF )
    GARB_HEIGHT ( GRAB_HEIGHT ) , GARB_WIDTH ( GRAB_WIDTH ) ,
    GARB_IN_HEIGHT ( GRAB_IN_HEIGHT ) ,
    GARB_IN_WIDTH ( GRAB_IN_WIDTH ) ,
    GARB_BITDESCRIBE ( GRAB_BITDESCRIBE ) ,
    GARB_WHOLEWIDTH ( GRAB_WHOLEWIDTH ) ,
    // 控制设备的工作参数：顶底倒置，水平镜像等功能。
    WORK_UPDOWN,    WORK_FLIP,    WORK_SKIP, WORK_SYNC,
    WORK_INTERLACE,    WORK_ISBLACK, WORK_FIELD,
```

```

OSD_MODE // 控制 OSD 模式(突出、凹下或不叠加)
//支持 V500 及 Moka-C50 系列卡
TENBIT_MODE,      OUTPUT_VIDEO,
FILERSELECT1, FILERSELECT2,
//支持 MVBOARD2.h 中所有类型的卡
ADJUST_BACKCOLORKEY,  DISP_FLIP,
IMAGE_PROCESS,      VIDEO_SINGLE
};

```

针对 7130/4 系列，参数的具体说明：

GARB_BITDESCRIBE：图像采集时的采集的图像的位深度，对应 SDK 的 MV_PARAMTER 枚举参数 GARB_BITDESCRIBE。参数为 RGB24、RGB32、RGB565、RGB1555、RGB5515、RGB8、YUV422、定义在 mv_phapi.h。

GARB_ISFIELD：图像采集时确定采集时按场或按帧采集，对应 SDK 的 MV_PARAMTER 枚举参数 WORK_FIELD。参数为 FIELD_INTERLACED, FIELD_TOP, FIELD_BOTTOM，定义在 mv_phapi.h。

GARB_ISFILP：图像采集时硬件的左右翻转，对应 SDK 的 MV_PARAMTER 枚举参数 WORK_FLIP（0 为正常，1 为左右翻转）。

GARB_ISUPDOWN：图像采集时是否用硬件上下翻转，对应 SDK 的 MV_PARAMTER 枚举参数 WORK_UPDOWN（0 为正常，1 为倒置）。

GARB_XOFF, GARB_YOFF：设置视频信号的 X 起始位置，Y 起始位置。对应 SDK 的 MV_PARAMTER 枚举参数 GARB_XOFF, GARB_YOFF。

GARB_HEIGHT, GARB_WIDTH：设置视频信号输出的高度，宽度（即实际采集大小）。对应 MV_PARAMTER 的参数 GARB_HEIGHT, GARB_WIDTH(PAL: 宽度为 16—768，高度为 16—576；NTSC：宽度为 16—640,高度为 16--480)。

DISP_FIELD：图像按场采集时的显示模式（0:显示所有场 1:显示奇场，2:显示偶场）。

VIDEO_STANDARD：对应 MV_PARAMTER 的参数 ADJUST_STANDARD(0 : PAL 1 : NTSC)。

VIDEO_CHANNEL：对应 MV_PARAMTER 的参数 ADJUST_CHANNEL(通道 0、1、2、3、4)。

VIDEO_BRIGHTNESS : 对应 MV_PARAMTER 的参数 ADJUST_LUMINANCE (参数 0--255) 。

VIDEO_HUE : 对应 MV_PARAMTER 的参数 ADJUST_HUE (参数 0--255) 。

VIDEO_SATURATION : 对应 MV_PARAMTER 的参数 ADJUST_SATURATION (参数 0--127) 。

VIDEO_CONTRAST : 对应 MV_PARAMTER 的参数 ADJUST_CONTRAST 参数 0--127) 。

函数说明:

1. MV_OpenDevice

原型: HANDLE MV_OpenDevice(DWORD Index)

说明: 初始化, 创建 Index 指定的设备。默认设置的属性为颜色格式为 RGB24、图像宽度为 640、高度为 480、制式为 pal、通道为 3、采集为帧模式。

入口参数:

Index: 采集卡的序列下标, 是从 0 开始的整数。

返回值:

返回 Index 对应采集卡的设备句柄, 对采集卡的操作均通过该句柄实现。

2. MV_CloseDevice

原型: void MV_CloseDevice(HANDLE hDevice)

说明: 不再使用以 hDevice 标识的设备时, 关闭该设备。

入口参数:

hDevice: 为先前用 MV_OpenDevice 创建并打开的设备句柄。

返回值: 无。

例: (使用多卡时) 假如 HANDLE hDev 为先前打开的设备句柄, 现在关闭它并打开第二个设备。

```
if( hDev != NULL )
    MV_CloseDevice( hDev );
hDev = NULL;
hDev = MV_OpenDevice( 1 );
if( hDev == NULL )
{
    // OK! 现在 hDev 为第二个设备句柄
}
```

3. MV_OperateDevice

原型: RUNOPER MV_OperateDevice(HANDLE hDevice, RUNOPER Oper)

说明: 操纵设备, 即使采集卡处于运行/停止/暂停状态。

入口参数: hDevice: 设备句柄。

Oper: 设备的状态(运行, 停止, 暂停, 查询)。

返回值: 返回卡的当前工作状态。

例 1: 打开第三个设备使它处于运行状态:

```
HANDLE hDev = MV_OpenDevice( 2 );
MV_OperateDevice(hDevice, MVRUN)
```

.....

例 2: 查询卡的工作状态:

```
RUNOPER WorkStatu = MV_OperateDevice ( hDevice, MVQUERYSTATU );
```

4. MV_SetDeviceParameter

原型: BOOL MV_SetDeviceParameter(HANDLE hDevice, MV_PARAMTER Oper, U32

value);

说明：设置设备和 SDK 的工作参数。

入口参数：hDevice: 设备句柄；

Oper: 欲设置工作参数的类型；

Value: 欲设置参数的值。

返回值：返回 RET_OK 代表参数设置成功；RET_ERR 代表失败，一般因为该类型的卡不支持该类型的工作参数。

注意：每个参数都有适用的卡型和默认的值，如果你没有设置，系统将会使用默认值。

例 1: 希望采集 768*576 的图像，采集格式 RGB24 位 CO_RGB24)

```
// 设置采集为 768*576*RGB32 (即 DATA_aRGB8888)
```

```
MV_SetDeviceParameter( hDevice, GARB_WIDTH, 768 );
```

```
MV_SetDeviceParameter( hDevice, GARB_HEIGHT, 576 );
```

```
MV_SetDeviceParameter( hDevice, GARB_BITDESCRIBE, CO_RGB24 );
```

5. MV_GetDeviceParameter

原型：U32 MV_GetDeviceParameter(HANDLE hDevice, MV_PARAMTER Oper)

说明：得到设备参数的当前值。

入口参数：hDevice：设备句柄。

Oper：欲设置工作参数的类型。

返回值：返回-1 代表失败，一般因为该类型的卡不支持该类型的工作参数，返回-2 是由于句柄是错误的。否则代表 Oper 参数的当前值。

例：在当前的亮度的基础上将亮度进行调节

```
// 得到当前的亮度
```

```
U32 Luminance = MV_GetDeviceParameter( hDev, ADJUST_LUMINANCE );
```

```
// 改变当前的亮度
```

```
MV_SetDeviceParameter( hDevice, ADJUST_LUMINANCE, ( Luminance + 10) );
```

6. MV_SetCallBack

原型：BOOL MV_SetCallBack(CALLBACKFUNC pCallBack, PVOID pUserData)

说明：设置设备的采集用户回调函数；在设备运行时每一幅图像到达时将会调用该用户回调函数。

入口参数：hDevice：设备句柄。

pCallBack：设备采集的用户回调。

此回调函数的原型为：

```
void (*CALLBACKFUNC)( PVOID pData, PMV_IMAGEINFO pImageInfo, PVOID  
pUserData, PLX_UINT_PTR Index );
```

pUserData：用户传递给回调函数的上下文数据。

返回值：返回 RET_OK 成功；返回 RET_ERR 失败。

7. MV_WriteBmp

原型： `BOOL MV_WriteBmp(cahr *filename, char * lpImage,int width,int height,int bpp)`

说明：保存一幅图像

入口参数：

filename : 保存图像文件名称
lpImage : 图像数据
width : 图像的宽度
height : 图像的高度
bpp : 图像位数

返回值：返回 `RET_OK` 成功；返回 `RET_ERR` 失败。

8. MV_AllocSequenceFrameMemory

原型： `BOOL MV_AllocSequenceFrameMemory(HANDLE hDevice, PLX_UINT_PTR Action, U32 Number);`

说明：对设备分配大帧存，调用时指定此帧存用于存放原始的数据或处理后的数据。

因为原始数据中图像每个像素所占的位数

可能与处理过的数据不同，而且图像的宽度也可能不同。

入口参数： hDevice：设备句柄。

Number : 欲分配帧存能存放的图像的数目。

Action : 指定帧存的类型，保留待扩展。可为用户定制大帧存操控的方式。

返回值：返回 `RET_OK` 分配成功；返回 `RET_ERR` 分配失败。

该函数可多次使用，用户无需担心会有内存泄漏。但是应该注意的是当您分配了很大的帧存却长期不使用时，应调用 `MV_FreeSequenceFrameMemory` 来释放该内存，否则会对操作系统的内存管理造成很大的压力。

9. MV_FreeSequenceFrameMemory

原型： `BOOL MV_FreeSequenceFrameMemory (HANDLE hDevice)`

说明：释放设备先前分配的大帧存。一般来说分配和释放是一一对应的，但这里用户编程时可以多次调用分配帧存函数，而在最后应用程序退出时调用释放函数。

入口参数：参数 hDevice: 设备句柄。

返回值：返回 `RET_OK` 释放成功；返回 `RET_ERR` 释放失败。

10. MV_SequenceCaptureStart

原型： `BOOL MV_SequenceCaptureStart(HANDLE hDevice, CONTINUEGARBMECHANISM pContinueCall, PVOID pContext)`

说明：开始进行连续捕获，捕获到的图像存放在用户分配的帧存中。当连续帧捕获结束或用户调用 `MV_SequenceCaptureStop` 函数发出停止捕获命令，用户设置的回调将会被调用。

入口参数： hDevice：设备句柄。

pContinueCall : 连续捕捉机制的用户回调，捕获结束或调用 `MV_SequenceCaptureStop` 时调用。

此回调函数的原型为：

void (*CONTINUEGARBMECHANISM)(PVOID pData, PMV_IMAGEINFO pImage, PLX_UINT_PTR ImageNumber, PLX_UINT_PTR wholeLength, PVOID pUserData);
返回值：返回 RET_OK 成功；返回 RET_ERR 失败。

11. MV_SequenceCaptureStop

原型：LONG MV_SequenceCaptureStop (HANDLE hDevice)

说明：停止对设备帧存的连续捕获，可在任何希望打断连续采集的时刻发出。

入口参数：hDevice：设备句柄。

返回值：返回 RET_OK 成功；返回 RET_ERR 失败。

12. MV_SetOutputState

原型：BOOL MV_SetOutputState(HANDLE hDevice, ULONG Index, ULONG HorL)

说明：使板卡输出一个高电平或者低电平；

入口参数：hDevice：设备句柄。

Index：指明第几路输出。

HorL：指明输出低电平 0，还是高电平 1。

返回值：返回 RET_OK 成功；返回 RET_ERR 失败。

13. MV_GetInputState

原型：ULONG WINAPI MV_GetInputState(HANDLE hDevice, ULONG Index);

说明：用户主动的读取输入状态；

入口参数：hDevice：设备句柄。

Index：指明第几路输入。

返回值：返回 0 代表低电平；返回 1 代表高电平。

14. MV_SetInputCallBack

原型：BOOL MV_SetInputCallBack (HANDLE hDevice, ULONG Index, PTRIGGERROUTINE pTirggerCall, PVOID pContext);

说明：为输入触发管脚设置用户回调函数。一旦有外输入触发产生，回调函数将被调用，当 pTirggerCall 为 NULL 时将不能触发。

入口参数：hDevice：设备句柄。

Index：指明管脚为 0 或 1。

pContext：为用户传递给回调函数的上下文数据。

pTirggerCall：为输入触发回调函数。

此回调函数的原型为：

void (*PTRIGGERROUTINE) (ULONG index, ULONG Reson, PVOID pContext)

回调参数：

Index：那一引脚

Reson：1 为高电平触发，0 为低电平触发

15. MV_TriStart

原型: `BOOL MV_TriStart(HANDLE hDevice);`

说明: 启动外触发。

入口参数: `hDevice` : 设备句柄。

返回值: 返回 `RET_OK` 成功; 返回 `RET_ERR` 失败。

16. MV_TriStop

原型: `BOOL MV_TriStop(HANDLE hDevice);`

说明: 停止外触发。

入口参数: `hDevice` : 设备句柄。

返回值: 返回 `RET_OK` 成功; 返回 `RET_ERR` 失败。