100ASK USB Video Manual

UVC Video User Manual

Embedded Development Platform

M4 User Manual

Rev. 1.1

2020/11/05

0755-86200561
www.100ask.net
Shenzhen, Guangdong, China
Shenzhen 100ask Technology Co. Ltd.

注意事项与售后维修

1. 注意事项

- 使用产品之前，请仔细阅读本手册，并妥善保管，以备将来参考；
- 请注意和遵循标注在产品上的所有警示和指引信息；
- 请使用配套电源适配器，以保证电压、电流的稳定；
- 请在凉爽、干燥、清洁的地方使用本产品；
- 请勿在冷热交替环境中使用本产品，避免结露损坏元器件；
- 请勿在湿气过重、温度过高或过低环境中使用本产品，使用时注意产品的通风；
- 请勿将任何液体泼溅在本产品上，禁止使用有机溶剂或腐蚀性液体清洗本产品；
- 请勿在多尘、脏乱的环境中使用本产品，如果长期不使用，请包装好本产品；
- 请勿在振动过大的环境中使用，任何跌落、敲打或剧烈晃动都可能损坏线路及元器件；
- 请勿在通电情况下，插拔核心板及外围模块(特别是串口模块)；
- 请勿自行维修、拆解本产品，如产品出现故障应及时联系本公司进行维修；
- 请勿自行修改或使用未经授权的配件，由此造成的损坏将不予保修；

2. 售后维修

1) 保修期限

- 底板、核心板：三个月（非人为损坏）
- 显示屏：七天（非人为损坏）

2) 保修说明

- 7 天内：产品（底板、核心板、屏幕）非人为损坏，本公司免费更换/维修，并承担来回运费；
- 7 天至 3 个月内：底板、核心板非人为损坏，本公司免费维修，并承担来回运费（屏幕不提供维修）；
- 3 个月至 1 年：底板、核心板非人为损坏或人为轻微损坏，只收更换元器件费用，免费维修，买家承担来回运费；
- 起始时间以快递签收日为准；

3) 联系方式

官方网站：www.100ask.net
淘宝网站：100ask.taobao.com

地　址：广东省深圳市龙岗区布吉南湾街道平吉大道建昇大厦 B1505
联系人：售后维修部
电　话：0755-86200561
邮　编：518114

邮寄须知：保修期限内，寄回本产品请预先垫付邮费，公司不接收任何到付快递。

技术支持与开发定制

1. 技术支持范围

1) 本公司提供的各类开发软件的安装，入门使用，环境搭建；
2) 本公司提供的所有裸机代码的烧写验证；
3) 本公司发布的操作系统的编译、烧写；
4) 本公司发布产品的工控板、模块的硬件原理；
5) 本公司发布的各种外设模块驱动及源码；
6) 本公司发布的配套手册在使用过程中遇到的问题；
7) 本公司产品的故障诊断及售后维修服务；

2. 技术讨论范围

由于嵌入式系统知识范围广泛，涉猎种类繁多，我们无法保证对各种问题都能一一解答，以下内容无法供技术支持，只能提供建议。

1) 本公司发布的教程之外的知识；
2) 非本公司发布的 U-Boot、Linux 内核的编译和移植；
3) 非本公司发布的工控板的各类驱动支持；
4) 非本公司发布的外设模块的硬件原理和驱动设计；

3. 技术支持方式

1) 官方论坛发帖提问(推荐)：bbs.100ask.net
2) 官方淘宝通过阿里旺旺咨询：100ask.taobao.com
3) QQ 群咨询（QQ 群号咨询淘宝客服，需提供淘宝购买订单号验证加入）；
4) 技术支持邮箱：weidongshan@qq.com
5) 联系电话：0755-86200561

4. 技术支持时间

星期一到星期五;上午 9:00—12:00;下午 14:00—17:30;

公司按照国家法定节假日安排休息，在此期间无法提供技术支持，请将问题发送至技术支持邮箱或在论坛对应板块发帖，我们将在工作日尽快给您回复。

5. 投诉和建议

如果您对我们有不满意或者建议，可发送邮件到 weidongshan@qq.com 进行反馈，也可拨打 0755-86200561 取得联系，我们将不断改进。

6. 定制开发服务

本公司提供嵌入式操作系统底层驱动、硬件板卡的有偿定制开发服务，以缩短您的产品开发周期。请将需求发送邮件到 weidongshan@qq.com。

资料获取与后续更新

1. 资料的获取

1) 百度网盘下载

百度网盘里面有本产品的所有配套资料，包括原理图、发布的 U-Boot、内核镜像和源码、所需的开发软件、工具等等。

进入 www.100ask.net，在导航栏选择“资料下载”，点击“到百度网盘下载”，跳转到百度网盘后，找到对应的文件夹即可。

2) 视频配套教程

后续会为该工控板录制一套裸机、Linux 驱动、应用的配套付费教学视频，有需要的客户可以进入官方淘宝 100ask.taobao.com 选购。

3) 维基百科教程

维基百科里面会有视频配套的笔记，进入 wiki.100ask.net，选择对应的板块查看。

2. 后续更新

后续文档、视频等资料的更新，为了确保您的资料是最新状态，请密切关注我们的动态，我们将会通过微信公众号和 QQ 群公告推送，购买了本产品的客户请添加 QQ 群（QQ 群号咨询淘宝客服，需提供淘宝购买订单号验证加入）或关注微信公众号。

版权声明

更新记录

类别	USB VIDEO 模块系列文档
文档名	100ASK USB 摄像头用户手册
当前版本	0.1
适用型号	百问网 IMX6ULL STM32MP157 系列开发板
编辑	百问科技文档编辑团队
审核	韦东山

修改日志		
版本	修改时间	更改说明
0.1	2020.11.05	初始版本，世玉轩

目录

前言

STM32 系列 MCU(Microcontroller Unit)在中国的受众非常广，许多高校学生将 STM32 系列 MCU 作为嵌入式入门的首选，许多 MCU 工程师也将 STM32 系列 MCU 作为方案首选。许多用户在对 STM32 系列 MCU 熟悉之后，想进一步学习运行 Linux 的 MPU(Microprocessor Unit)时，不得不选择其它厂商的 MPU，重新熟悉一套开发套件，加大了前期学习的负担。

不含 MMU(Memory Management Unit)的 MCU，主要用于一些对实时性要求较高的场景，比如航空航天、汽车电子等。而带有 MMU 的 MPU，可以通过 MMU 来管理虚拟内存空间，从而可以支持诸如 Linux 的操作系统，打开操作系统的大门。

意法半导体(STMicroelectronics)稳固 STM32 系列在 MCU 市场中的重要地位后，首次以 STM32MP15 平台进入 MPU 市场。该平台采用 MCU+MPU 的方案，旨在为工业制造、消费电子、医疗保健、智能家居等多个市场领域提供高性能、硬实时、低功耗、更安全的解决方案。

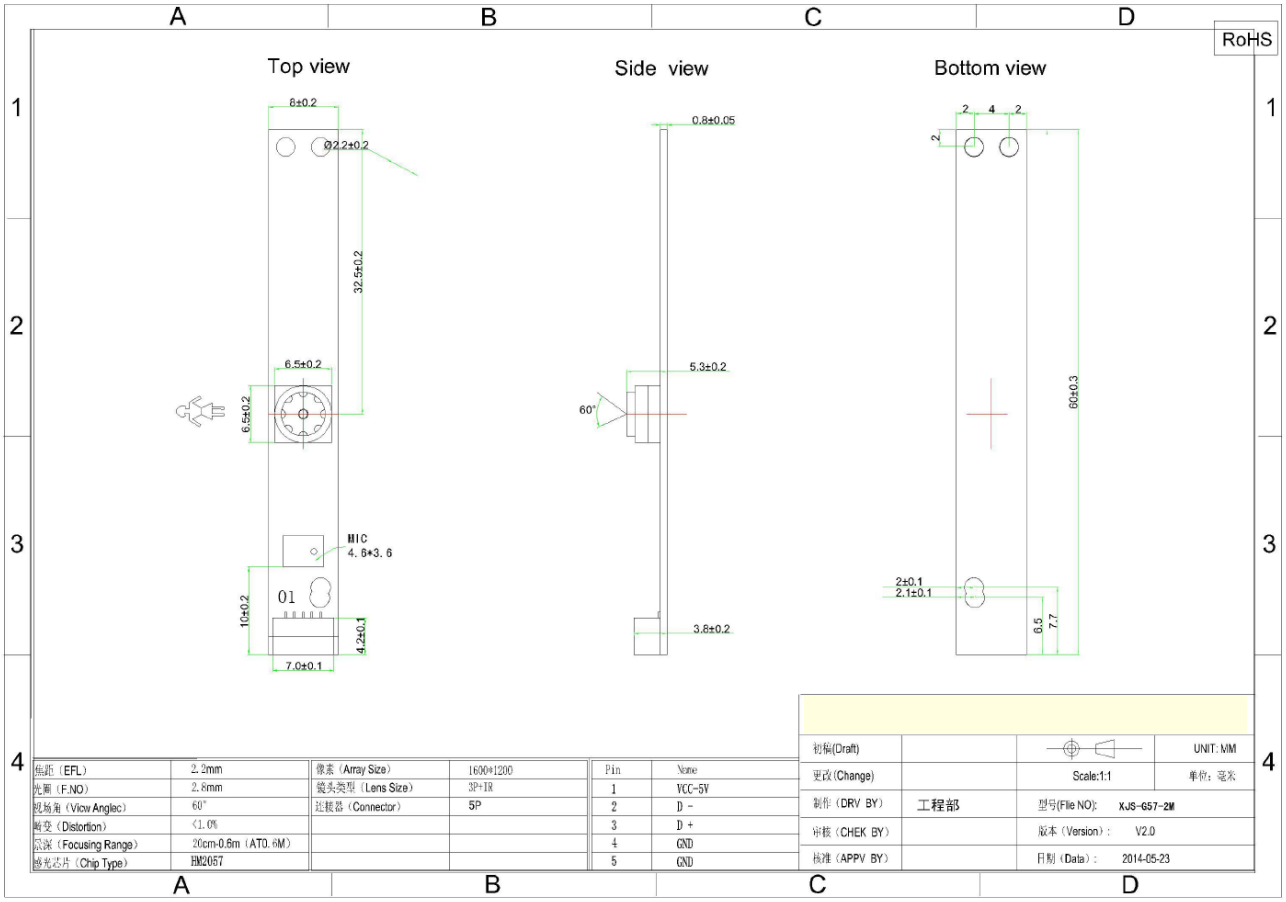
在 STM32MP157x 上开发 Cortex-M4（后面简称 M4）的操作流程几乎和开发 STM32 单片机一致。本手册将先介绍开发使用的平台，然后介绍开发流程，最后依次介绍 M4 资源进行基础开发，以及特色的核之间通信等。

第1章 开发资源介绍

本章介绍基于百问网 200W UVC USB 摄像头开发所设计的硬件以及软件资源介绍做一个简单的讲解。

1.1 硬件资源

Module No.	XJS-G57-2M
Module Size	60mm × 8mm ×5.3mm
Temperature (Operation)	-20° to 70°CC
Temperature (Stable Image)	0° to 50°CC
Assembly technique	SMT (ROSH)
Focus	Fixed
Object distance	40CM-200CM
Resolution	600LW/PH (Center)
PCB printing ink	black
interface	USB 2.0
Power	USB bus power
Power consumption	110mW （VGA）; 130 mW （UXGA）;
Operating system request	Windows 2000\ Windows XP/7/8 安卓
Package	Anti-electrostatic tray
Certifications	FCC and CE
Sensor Type	HM2057 (1/5)
Active Array Size	1600*1200
Sensitivity	TBD
Pixel Size	1.75μm x 1.75μm
Maximum Image Transfer Rate	30 fps for UXGA;
S/N Ratio	TBD
Dynamic Range	TBD
Package	CSP,Bare Die
Lens Type	
Lens Construction	4P+IR
F/No	2.6mm
EFL	2.8mm
BFL(Optical)	4.8mm
FOV	60°
TV Distortion	<1%
Relative Illumination (Sensor)	70%
IR Filter	650±10nm
DSP Type	
AGC/AEC/Whiter balance	Auto
Output Formats	USB suspend out
Fixed Pattern Noise	
Package	< 0.03% of $V_{PEAK-TO-PEAK}$


RoHS
Top view
Side view
Bottom view
8±0.2
Ø2.2±0.2
32.5±0.2
6.5±0.2
6.5±0.2
MIC
4.6*3.6
01
10±0.2
4.2±0.1
7.0±0.1
0.8±0.05
5.3±0.2
60°
3.6±0.2
2
4
2
2
60±0.3
2±0.1
2.1±0.1
6.5
7.7
焦距（EFL） 2.2mm
光圈（F.NO） 2.8mm
视场角（View Anglec） 60°
畸变（Distortion） <1.0%
景深（Focusing Range） 20cm-0.6m（AT0.6M）
感光芯片（Chip Type） BM2057
像素（Array Size） 1600*1200
镜头类型（Lens Size） 3P+IR
连接器（Connector） 5P
Pin Name
1 VCC-5V
2 D -
3 D +
4 GND
5 GND
初稿(Draft)
更改(Change)
制作（DRV BY） 工程部
审核（CHEK BY）
核准（APPV BY）
UNIT: MM
Scale:1:1
单位：毫米
型号(File NO)： KJS-657-2M
版本（Version）： V2.0
日期（Data）： 2014-05-23

1.2 软件资源

介绍使用在 Ubuntu-18.04 环境下如何安装 uvc 摄像头所需的软件包，以及如何预览摄像头数据拍照，介绍如何查看常用的摄像头参数 同时我们也会在开发板内做相同类似操作来演示使用。

第2章 ubuntu 下摄像头开发配置示例

如下操作均在 ubuntu-18.04 上进行实验演示，如您使用的开发环境并非 ubuntu-18.04 请自行根据您的系统进行修改。

2.1 配置开发环境

2.2.1 安装 python-opencv

Ubuntu 系统默认会安装有 python3.x 的运行环境,我们只需要安装支持 python 开发的 opencv3 包即可。

```
book@100ask:~$  sudo apt  install python3-opencv -y
```

安装好执行 python3 进入 python 语言解释器，输入 import cv2 cv2.__version__ 验证是否安装成功 opencv3。

如下图示例所示，打印出 ‘3.2.0’表示已经安装好 python opencv3

```
book@100ask:~$ python3
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.2.0'
>>>
```

2.2.2 安装其它工具包

安装 v4l-utils

video4linux(v4l)是一些视频系统，视频软件，音频软件的基础，经常使用在需要采集图像的场合，如视频监控，webcam,可视电话，经常应用在 embedded linux 中，是 linux 嵌入式开发中经常使用的系统接口。它是 linux 内核提供给用户空间的编程接口，各种的视频和音频设备开发相应的驱动程序后，就可以通过 v4l 提供的系统 API 来控制视频和音频设备，也就是说 v4l 分为两层，底层为音视频设备在内核中的驱动，上层为系统提供的 API，而对于我们来说需要的就是使用这些系统的 API。

v4l-utils 为媒体设备提供了一系列实用程序,可以处理大多数网络摄像头(libv4l)可用的专有格式，并提供测试 V4L 设备的工具。

```
book@100ask:~$ sudo apt-get install v4l-utils libv4l-dev -y
```

安装完成后，可以在 ubuntu 终端下执行 v4l2-ctl 命令来查看支持的参数选项来验证是否安装成功。

```
book@100ask:~$ v4l2-ctl

General/Common options:
  --all             display all information available
  -C, --get-ctrl=<ctrl>[,<ctrl>...]
                    get the value of the controls [VIDIOC_G_EXT_CTRLS]
  -c, --set-ctrl=<ctrl>=<val>[,<ctrl>=<val>...]
                    set the value of the controls [VIDIOC_S_EXT_CTRLS]
  -D, --info        show driver info [VIDIOC_QUERYCAP]
  -d, --device=<dev> use device <dev> instead of /dev/video0
                    if <dev> starts with a digit, then /dev/video<dev> is used
  -e, --out-device=<dev> use device <dev> for output streams instead of the
                    default device as set with --device
                    if <dev> starts with a digit, then /dev/video<dev> is used
  -h, --help        display this help message
```

安装 cheese

Cheese 是一个普通的程序，可以从您的网络摄像头拍摄照片和视频。您可以使用连拍模式快速连续拍摄多张照片，并应用特殊效果为其添加个人风格。Cheese 使用 GStreamer 将精美的图形效果应用于照片和视频。

执行下述命令即可安装 cheese 到 ubuntu 系统上。

```
book@100ask:~$  sudo apt  install cheese  -y
```

在 ubuntu 终端下输入 cheese -v 来验证当前已安装的版本。

```
book@100ask:~$ cheese -v
Cheese 3.28.0
book@100ask:~$
```

安装 camorama

Camorama 是一个小型实用程序，用于查看和保存来自网络摄像头或任何其他 Video4Linux 设备的图像。它可以应用许多图像滤镜并进行远程捕获。

执行下述命令即可安装 camorama 到 ubuntu 系统上。

```
book@100ask:~$  sudo apt camorama install -y
```

在 ubuntu 终端下输入 camorama --version 来验证当前已安装的版本。

```
book@100ask:~$ camorama --version
Gtk-Message: 21:20:04.030: Failed to load module "canberra-gtk-module"


Camorama version 0.19

book@100ask:~$
```

安装 mplayer

MPlayer 是一款开源的多媒体播放器，以 GNU 通用公共许可证发布。此款软件可在各主流操作系统使用，例如 Linux 和其他类 Unix 操作系统、微软的 Windows 系统及苹果电脑的 Mac OS X 系统。MPlayer 是建基于命令行界面，在各操作系统可选择安装不同的图形界面。

执行下述命令即可安装 mplayer 到 ubuntu 系统上。

```
book@100ask:~$  sudo apt  install mplayer  -y
```

在 ubuntu 终端下输入 mplayer -v 来验证当前已安装的版本。

```
book@100ask:~$ mplayer -v
MPlayer 1.3.0 (Debian), built with gcc-7 (C) 2000-2016 MPlayer Team
CPU vendor name: GenuineIntel  max cpuid level: 22
CPU: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz (Family: 6, Model: 158, Stepping: 10)
extended cpuid-level: 8
extended cache-info: 16801856
Detected cache-line size is 64 bytes
CPUflags:  MMX: 1 MMX2: 1 3DNow: 0 3DNowExt: 0 SSE: 1 SSE2: 1 SSE3: 1 SSSE3: 1 SSE4: 1 SSE4.2: 1 AVX: 1
Compiled with runtime CPU detection.
get_path('codecs.conf') -> '/home/book/.mplayer/codecs.conf'
Reading optional codecs config file /home/book/.mplayer/codecs.conf: No such file or directory
Reading optional codecs config file /etc/mplayer/codecs.conf: No such file or directory
Using built-in default codecs.conf
```

安装 vlc

VLC 多媒体播放器（最初命名为 VideoLAN 客户端）是 VideoLAN 计划的多媒体播放器。它支持众多音频与视频解码器及文件格式，并支持 DVD 影音光盘，VCD 影音光盘及各类流式协议。它也能作为 unicast 或 multicast 的流式服务器在 IPv4 或 IPv6 的高速网络连接下使用。它融合了 FFmpeg 计划的解码器与 libdvdcss 程序库使其有播放多媒体文件及加密 DVD 影碟的功能。

执行下述命令即可安装 vlc 播放器到 ubuntu-18.04 系统上，后续我们可以使用 vlc 命令来预览摄像头数据。

```
book@100ask:~$  sudo apt  install vlc  -y
```

在 ubuntu 终端下输入 vlc --version 来验证当前已安装的版本。

```
book@100ask:~$ vlc --version
VLC media player 3.0.8 Vetinari (revision 3.0.8-0-gf350b6b5a7)
VLC version 3.0.8 Vetinari (3.0.8-0-gf350b6b5a7)
Compiled by buildd on lcy01-amd64-014.buildd (Sep 11 2019 11:39:37)
Compiler: gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)
This program comes with NO WARRANTY, to the extent permitted by law.
You may redistribute it under the terms of the GNU General Public License;
see the file named COPYING for details.
Written by the VideoLAN team; see the AUTHORS file.
book@100ask:~$
```

安装 gstreamer-1.0

Gstreamer 是一个支持 Windows，Linux，Android， iOS 的跨平台的多媒体框架，应用程序可以通过管道（Pipeline）的方式，将多媒体处理的各个步骤串联起来，达到预期的效果。每个步骤通过元素（Element）基于 GObject 对象系统通过插件（plugins）的方式实现，方便了各项功能的扩展， GStreamer 基于流水线的多媒体框架，基于 GObject，以 C 语言写成。凭借 GStreamer，程序员可以很容易地创建各种多媒体功能组件，包括简单的音频回放，音频和视频播放,录音，流媒体和音频编辑。基于流水线设计，可以创建诸如视频编辑器、流媒体广播和媒体播放器等等的很多多媒体应用。

参考 gstreamer 官方网址 https://gstreamer.freedesktop.org/documentation/installing/on-linux.html?gi-language=c 在 ubuntu 下执行如下命令进行安装 gstreamer1.0 所需的软件包。

```
book@100ask:~$  sudo apt-get install libgstreamer1.0-0 gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad
gstreamer1.0-plugins-ugly gstreamer1.0-libav gstreamer1.0-doc gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl
gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio -y
```

在 ubuntu 终端下输入 gst-launch-1.0 --version 来验证当前已安装的版本。

```
book@100ask:~$ gst-launch-1.0 --version
gst-launch-1.0 version 1.14.5
GStreamer 1.14.5
https://launchpad.net/distros/ubuntu/+source/gstreamer1.0
book@100ask:~$
```

安装 ffmpeg

ffmpeg 是一套可以用来记录、转换数字音频、视频，并能将其转化为流的开源计算机程序。采用 LGPL 或 GPL 许可证。它提供了录制、转换以及流化音视频的完整解决方案。它包含了非常先进的音频/视频编解码库 libavcodec，为了保证高可移植性和编解码质量，libavcodec 里很多 code 都是从头开发的。

在 ubuntu 终端下执行如下命令进行安装。

```
book@100ask:~$  sudo apt install ffmpeg -y
```

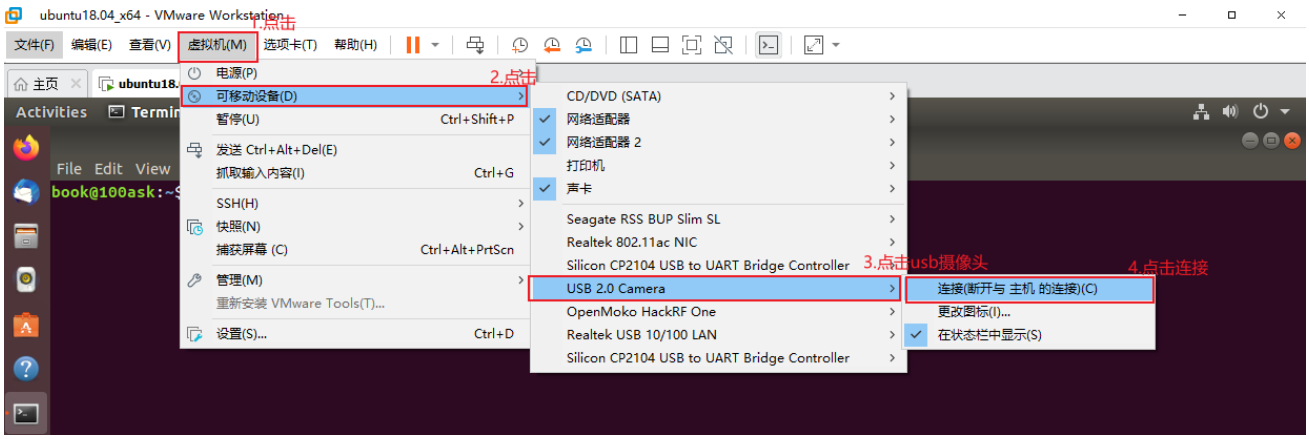
在 ubuntu 终端下输入 ffmpeg -version 来验证当前已安装的版本。

```
book@100ask:~$ ffmpeg -version
ffmpeg version 3.4.8-0ubuntu0.2 Copyright (c) 2000-2020 the FFmpeg developers
built with gcc 7 (Ubuntu 7.5.0-3ubuntu1~18.04)
configuration: --prefix=/usr --extra-version=0ubuntu0.2 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/inclu
de/x86_64-linux-gnu --enable-gpl --disable-stripping --enable-avresample --enable-avisynth --enable-gnutls --enable-ladspa --enable
-libass --enable-libbluray --enable-libbs2b --enable-libcaca --enable-libcdio --enable-libflite --enable-libfontconfig --enable-lib
freetype --enable-libfribidi --enable-libgme --enable-libgsm --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg --enable-l
ibopenmpt --enable-libopus --enable-libpulse --enable-librubberband --enable-librsvg --enable-libshine --enable-libsnappy --enable-
libsoxr --enable-libspeex --enable-libssh --enable-libtheora --enable-libtwolame --enable-libvorbis --enable-libvpx --enable-libwav
pack --enable-libwebp --enable-libx265 --enable-libxml2 --enable-libxvid --enable-libzmq --enable-libzvbi --enable-omx --enable-ope
nal --enable-opengl --enable-sdl2 --enable-libdc1394 --enable-libdrm --enable-libiec61883 --enable-chromaprint --enable-frei0r --en
able-libopencv --enable-libx264 --enable-shared
libavutil      55. 78.100 / 55. 78.100
libavcodec     57.107.100 / 57.107.100
libavformat    57. 83.100 / 57. 83.100
libavdevice    57. 10.100 / 57. 10.100
libavfilter     6.107.100 /  6.107.100
libavresample   3.  7.  0 /  3.  7.  0
libswscale      4.  8.100 /  4.  8.100
libswresample   2.  9.100 /  2.  9.100
libpostproc    54.  7.100 / 54.  7.100
book@100ask:~$
```

2.2 连接摄像头至 ubuntu

2.2.1 连接 usb 摄像头至 vmware

首先我们需要将购买的 USB 摄像头连接至电脑 USB 接口处，再通过 vmware 进行设置连接到 ubuntu 系统内，设置连接方式请参考下图所示依次点击设置。



参考上图设置将 usb 摄像头使用 vmware 连接至 ubuntu 时，我们使用 dmesg 命令来查看系统的打印信息，如下图所示，会显示类似的打印信息，其中会包含一个 usb 设备 ID 为 038f:0541 的设备即表示已经摄像头已连接至 ubuntu 虚拟机。

```
book@100ask:~$  dmesg
```

```
book@100ask:~$ dmesg
[26571.251846] usb 3-2: new high-speed USB device number 9 using xhci_hcd
[26571.706406] usb 3-2: New USB device found, idVendor=038f, idProduct=0541, bcdDevice= 0.04
[26571.706415] usb 3-2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[26571.706423] usb 3-2: Product: USB 2.0 Camera
[26571.706430] usb 3-2: Manufacturer: lihappe8 Corp.
[26571.713564] uvcvideo: Found UVC 1.00 device USB 2.0 Camera (038f:0541)
[26571.735382] uvcvideo 3-2:1.0: Entity type for entity Processing 2 was not initialized!
[26571.735389] uvcvideo 3-2:1.0: Entity type for entity Extension 6 was not initialized!
[26571.735394] uvcvideo 3-2:1.0: Entity type for entity Camera 1 was not initialized!
[26571.735477] input: USB 2.0 Camera: USB 2.0 Camera as /devices/pci0000:00/0000:00:15.0/0000:03:00.0/usb3/3-2/3-2:1.0/input/input1
0
book@100ask:~$
```

之后我们可以执行 lsusb -d 038f:0541 -v | more | grep "bFunctionClass" 来查看 ID 为 038f:0541

设备属于那种设备类型，如下图所示，可以看到此 usb 设备分别属于 Video 类和 Audio 类，我们可以使用它进行录音可拍摄录像等操作。

```
book@100ask:~$  lsusb -d  038f:0541 -v | more | grep "bFunctionClass"
```

```
book@100ask:~$ lsusb -d  038f:0541 -v | more | grep "bFunctionClass"
Couldn't open device, some information will be missing
      bFunctionClass         14 Video
      bFunctionClass          1 Audio
book@100ask:~$
```

2.2.2 验证摄像头设备

由于摄像头是标准的 uvc 摄像头，Linux 系统默认情况下会自动会根据 udev 规则自动安装 uvcvideo 驱动程序，如下图所示，使用 lsmod | grep uvcvideo 命令来查看是否安装 uvcvideo 驱动模块，如果没有自动安装请在 ubuntu 终端下执行 sudo modprobe uvcvideo 命令来安装 uvcvideo 摄像头模块驱动。

```
book@100ask:~$  lsmod | grep uvcvideo
```

```
book@100ask:~$ lsmod | grep uvcvideo
uvcvideo               94208  0
videobuf2_vmalloc      20480  1 uvcvideo
videobuf2_v4l2         24576  1 uvcvideo
videobuf2_common       53248  2 videobuf2_v4l2,uvcvideo
videodev              217088  3 videobuf2_v4l2,uvcvideo,videobuf2_common
mc                     53248  5 videodev,snd_usb_audio,videobuf2_v4l2,uvcvideo,videobuf2_common
book@100ask:~$
```

手动安装 uvcvideo 摄像头驱动模块。

```
book@100ask:~$  sudo modprobe uvcvideo
```

确认已经安装好 uvcvideo 设备驱动后，使用 ls /dev/video* 来查看是否存在 video 的设备节点，如果存在表示驱动已经正确安装并成功加载 usb video 摄像头设备。

```
book@100ask:~$ ls /dev/video* -la
crw-rw----+ 1 root video 81, 0 Nov  3 05:04 /dev/video0
crw-rw----+ 1 root video 81, 1 Nov  3 05:04 /dev/video1
book@100ask:~$
```

我们可以使用 v4l2-ctl --list-device 命令来获取并列出已知的摄像头设备。

```
book@100ask:~$ v4l2-ctl --list-device
USB 2.0 Camera: USB 2.0 Camera (usb-0000:03:00.0-2):
        /dev/video0
        /dev/video1

book@100ask:~$
```

2.2.3 验证 usb 录音设备

由于摄像头是标准的 uvc 摄像头，其中录音设备也是通用的 usb 声卡设备驱动，Linux 系统默认情况下会自动会根据 udev 规则自动安装 snd_usb_audio 驱动程序，如下图所示，使用 lsmod | grep "snd_usb_audio"命令来查看是否安装 snd_usb_audio 相应驱动模块，如果没有自动安装请在 ubuntu 终端下执行 sudo modprobe snd_usb_audio 命令来安装 uvcvideo 摄像头模块驱动。

```
book@100ask:~$ lsmod | grep "snd_usb_audio"
snd_usb_audio         262144  1
snd_usbmidi_lib        36864  1 snd_usb_audio
snd_hwdep              20480  1 snd_usb_audio
mc                     53248  5 videodev,snd_usb_audio,videobuf2_v4l2,uvcvideo,videobuf2_common
snd_pcm               102400  3 snd_usb_audio,snd_ac97_codec,snd_ens1371
snd                    86016  20 snd_seq,snd_seq_device,snd_hwdep,snd_usb_audio,snd_usbmidi_lib,snd_timer,snd_ac97_codec,snd_pcm,sn
d_rawmidi,snd_ens1371
book@100ask:~$
```

在确保有 snd_usb_audio 驱动模块后，我们可以在 ubuntu 终端下使用 arecord -l 命令列出支持的所有录音设备，其中可以看到 card 1: Camera [USB 2.0 Camera], device 0: USB Audio [USB Audio] 为摄像头设备的录音设备。

```
book@100ask:~$ arecord -l
**** List of CAPTURE Hardware Devices ****
card 0: AudioPCI [Ensoniq AudioPCI], device 0: ES1371/1 [ES1371 DAC2/ADC]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: Camera [USB 2.0 Camera], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
book@100ask:~$
```

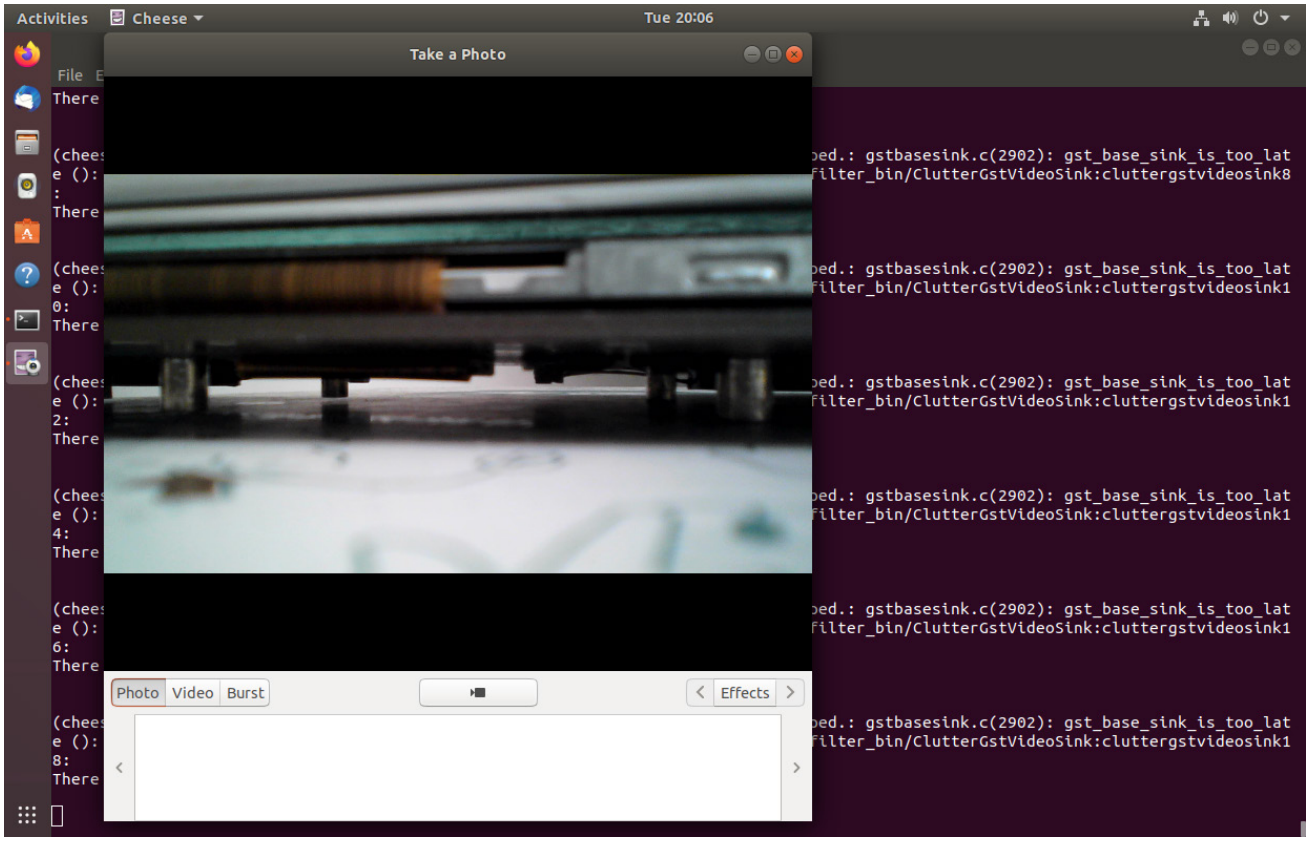
2.3 简单运行示例

2.3.1 预览图像

使用 cheese

在参考上一章节连接好摄像头以后，ubuntu 终端上输入 cheese 即可开始预览摄像头的数据，可以点击画面下的 photo(拍照) video(录像) Burst(连续拍照) 来选择模式，点击画面正下方的 按钮来开始操作。

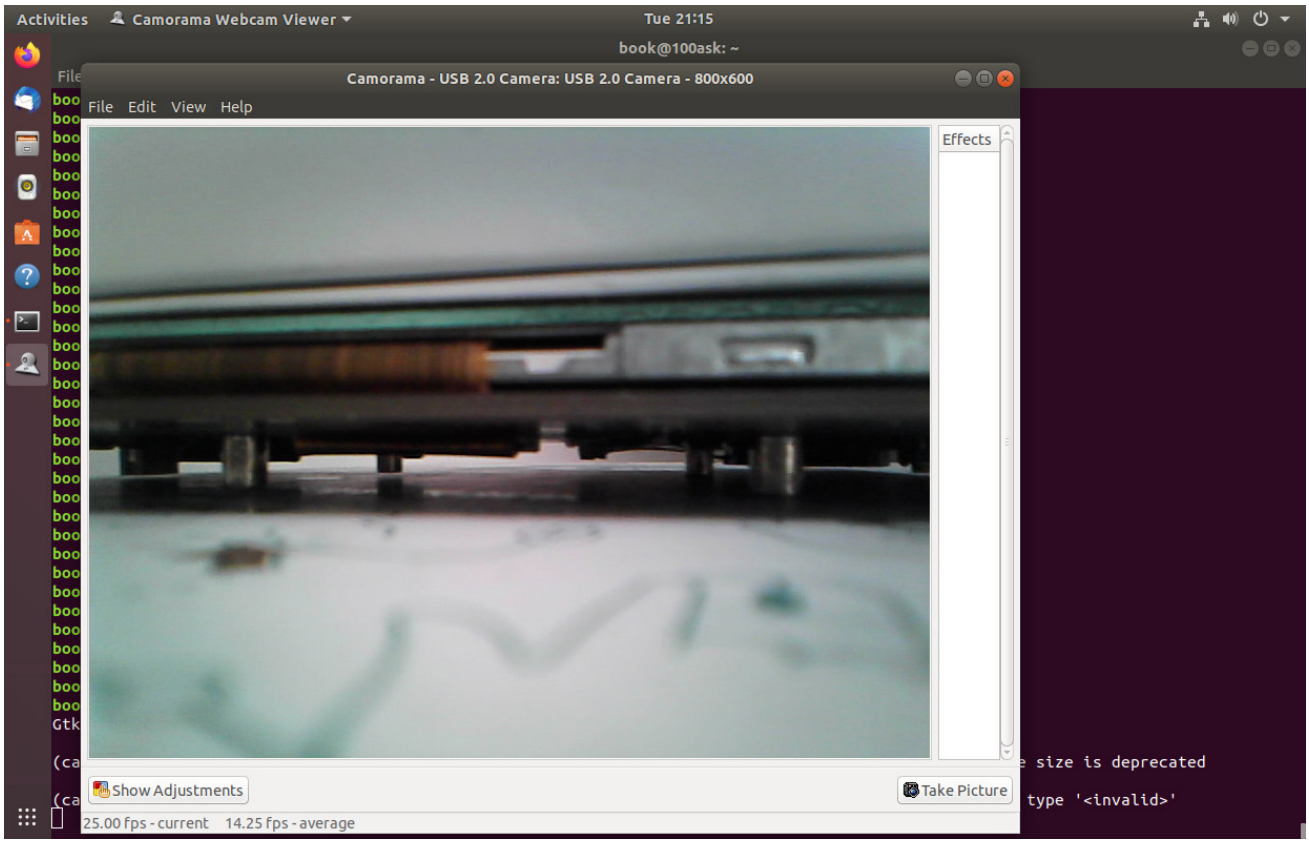
```
book@100ask:~$  cheese
```



使用 camorama

在 ubuntu 终端下使用 camorama 进行拍照，点击图像右下角的 Take Picture 图标来进行拍照，还可以点击 Show Adjustments 按钮来显示更多摄像头参数设置信息设置。

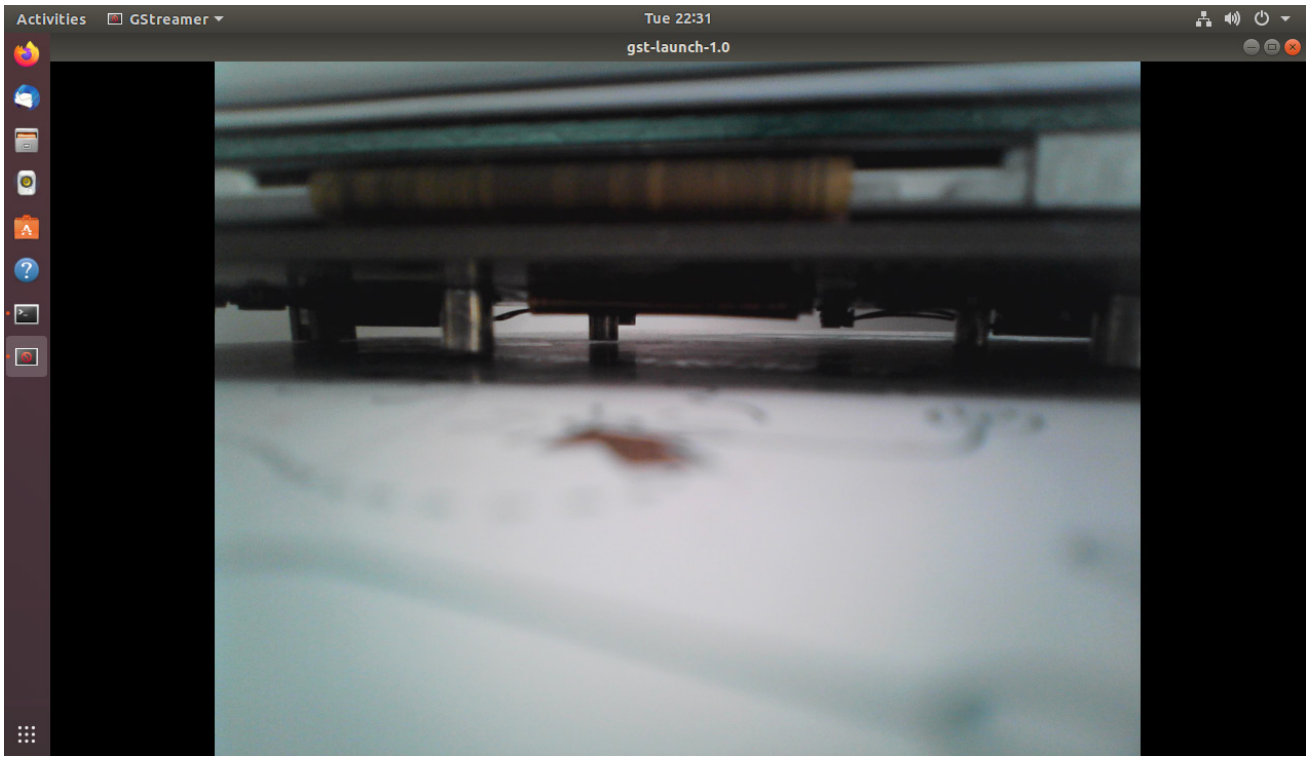
```
book@100ask:~$ camorama
```



使用 gstreamer

在 ubuntu 终端下使用 gst-launch-1.0 命令来预览摄像头数据。

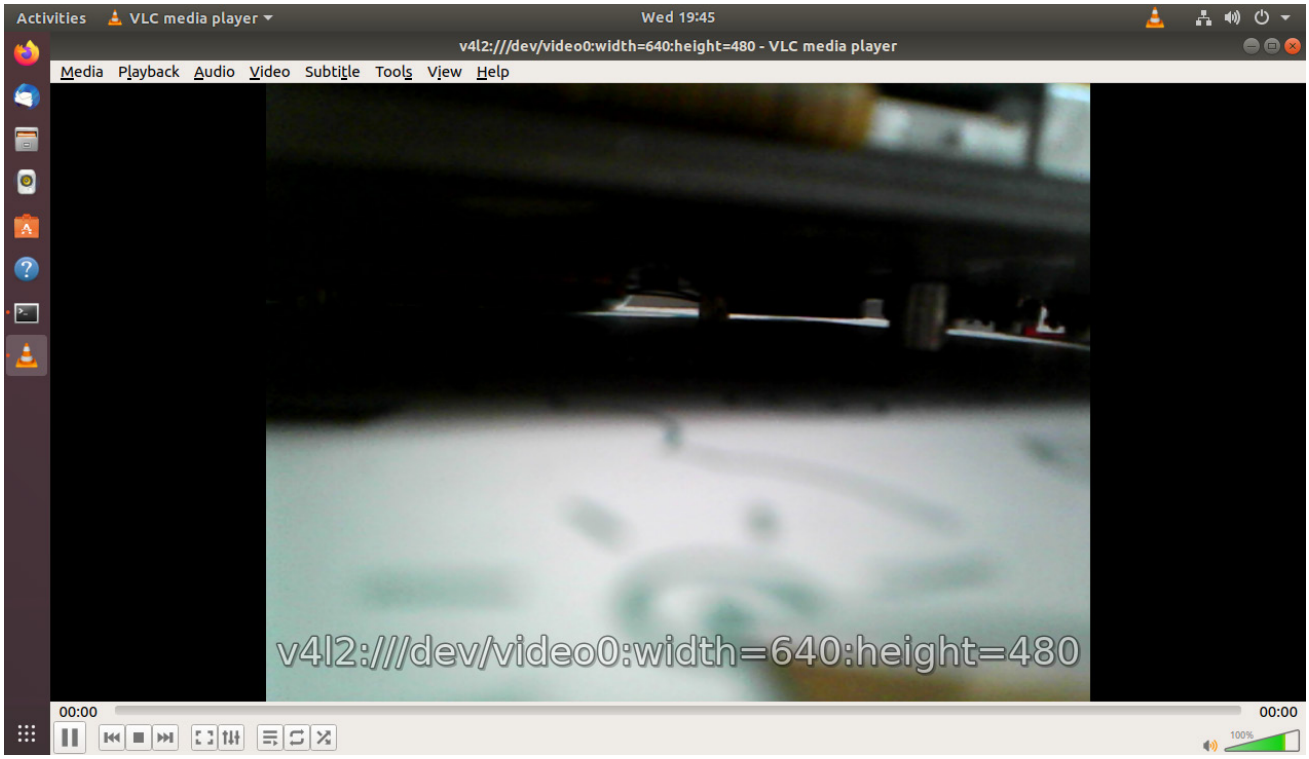
```
book@100ask:~$ gst-launch-1.0 v4l2src ! xvimagesink
```



使用 vlc

使用 vlc 命令播放设备节点为/dev/video0 的摄像头数据，指定分辨率为 640x480。

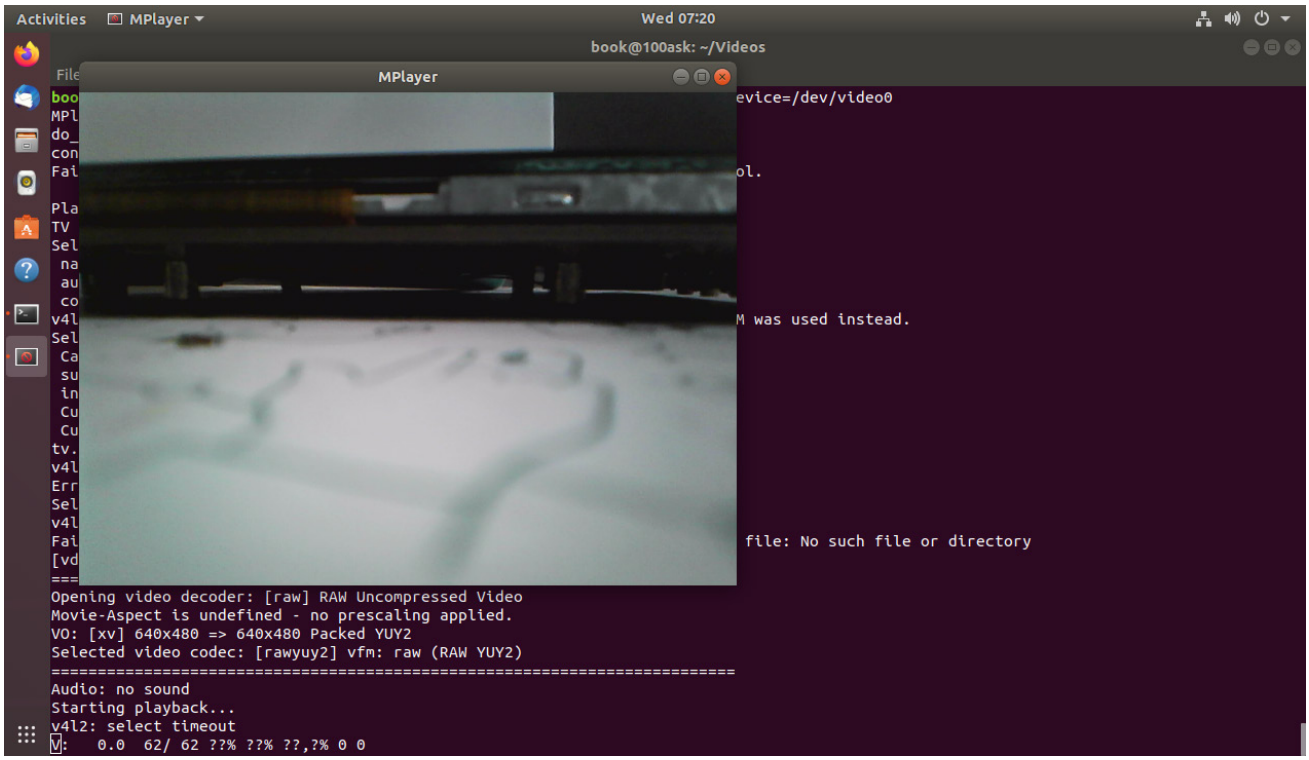
```
book@100ask:~$  vlc v4l2:///dev/video0:width=640:height=480
```



使用 mplayer

如下命令使用 mplayer 播放设备为/dev/video0 的摄像头数据，指定分辨率为 640x480。

```
book@100ask:~$  mplayer tv:// -tv driver=v4l2:width=640:height=480:device=/dev/video0
```



2.3.2 编程预览

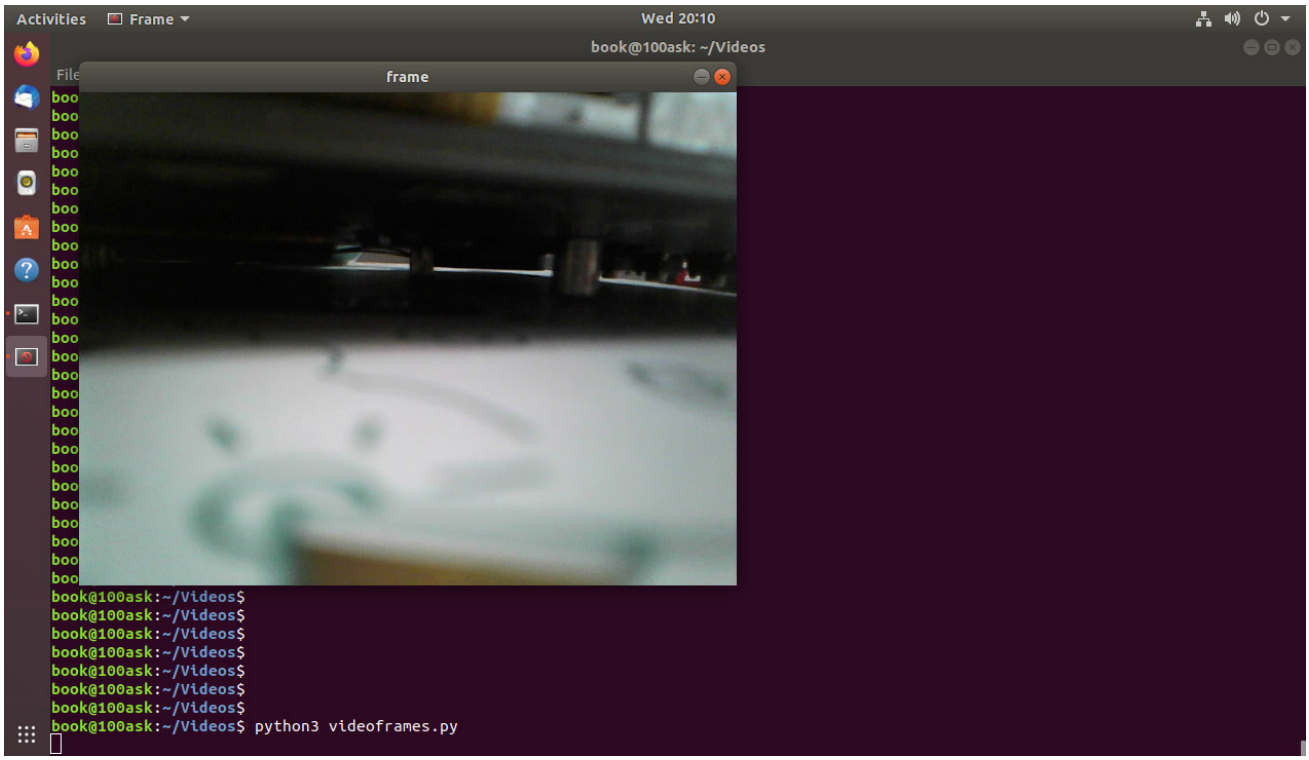
C 编程

Python 编程

如下所示，我们使用 opencv3 来预览摄像头数据，使用 vim 新建 videopreview.py 文件并保存如下代码。

```
import cv2
cap = cv2.VideoCapture(0)
ret, frame = cap.read()
while ret:
    ret, frame = cap.read()
    cv2.imshow("frame",frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()
cap.release()
```

保存成功后，我们在终端上执行 chmod +x videopreview.py 来给文件增加可执行权限，最后使用 python3 videopreview.py 来运行，如下图所示为预览摄像头数据。



2.4 其它示例

2.4.1 获取摄像头参数

列出所有的摄像头设备

使用 v4l2-ctl --list-device 命令可以列出所有的摄像头设备。

```
book@100ask:~$  v4l2-ctl --list-device
```

查看某个设备的详细驱动信息

使用 v4l2-ctl -d /dev/video0 -D 命令可以列出/dev/video0 摄像头设备的详细驱动参数。

```
book@100ask:~$  v4l2-ctl -d /dev/video0 -D
```

```
book@100ask:~/Videos$ v4l2-ctl -d /dev/video0 -D
Driver Info (not using libv4l2):
        Driver name   : uvcvideo
        Card type     : USB 2.0 Camera: USB 2.0 Camera
        Bus info      : usb-0000:03:00.0-2
        Driver version: 5.4.65
        Capabilities  : 0x84A00001
                Video Capture
                Metadata Capture
                Streaming
                Extended Pix Format
                Device Capabilities
        Device Caps   : 0x04200001
                Video Capture
                Streaming
                Extended Pix Format
book@100ask:~/Videos$
```

列出摄像头支持哪些像素格式

使用 v4l2-ctl --device /dev/video0 --list-formats 可以列出/dev/video0 摄像头设备都支持哪些像素格式，如下所示支持 YUYV 和 JPEG 像素格式。

```
book@100ask:~$   v4l2-ctl --device /dev/video0 --list-formats
```

```
book@100ask:~/Videos$ v4l2-ctl --device /dev/video0 --list-formats
ioctl: VIDIOC_ENUM_FMT
        Index       : 0
        Type        : Video Capture
        Pixel Format: 'YUYV'
        Name        : YUYV 4:2:2

        Index       : 1
        Type        : Video Capture
        Pixel Format: 'MJPG' (compressed)
        Name        : Motion-JPEG

book@100ask:~/Videos$
```

列出摄像头都支持哪些控制参数

使用 v4l2-ctl --device /dev/video0 -L 命令可以查看/dev/video0 设备都支持哪些可以控制的参数。

```
book@100ask:~$ v4l2-ctl --device   /dev/video0 -L
```

```
book@100ask:~/Videos$ v4l2-ctl --device  /dev/video0 -L
                    brightness 0x00980900 (int)    : min=-255 max=255 step=1 default=0 value=0
                      contrast 0x00980901 (int)    : min=0 max=30 step=1 default=15 value=15
                    saturation 0x00980902 (int)    : min=0 max=127 step=1 default=30 value=30
                           hue 0x00980903 (int)    : min=-16000 max=16000 step=100 default=0 value=0
 white_balance_temperature_auto 0x0098090c (bool)   : default=1 value=1
                         gamma 0x00980910 (int)    : min=20 max=250 step=1 default=98 value=98
          power_line_frequency 0x00980918 (menu)   : min=0 max=2 default=1 value=1
                                0: Disabled
                                1: 50 Hz
                                2: 60 Hz
     white_balance_temperature 0x0098091a (int)    : min=2500 max=7000 step=1 default=5000 value=5000 flags=inactive
                    sharpness 0x0098091b (int)    : min=0 max=15 step=1 default=2 value=2
        backlight_compensation 0x0098091c (int)    : min=0 max=2 step=1 default=1 value=1
book@100ask:~/Videos$
```

列出摄像头支持哪些分辨率格式

使用 v4l2-ctl --device /dev/video0 --list-formats-ext 可以列出/dev/video0 设备支持哪些像素格式同时又包含哪些分辨率以及帧率。

```
book@100ask:~$ v4l2-ctl --device /dev/video0 --list-formats-ext
```

```
book@100ask:~/Videos$ v4l2-ctl --device /dev/video0 --list-formats-ext
ioctl: VIDIOC_ENUM_FMT
        Index       : 0
        Type        : Video Capture
        Pixel Format: 'YUYV'
        Name        : YUYV 4:2:2
                Size: Discrete 640x480
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.040s (25.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 160x120
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.040s (25.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 320x240
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.040s (25.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 352x288
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.040s (25.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
```

2.4.2 修改摄像头参数

使用 v4l2-ctl 可以设置一些参数，如下设置默认的帧率为 30fps。

```
book@100ask:~$ v4l2-ctl --set-parm=30
```

也可以使用 v4l2-ctl 设置默认的摄像头采集数据分辨率和数据格式，如下所示设置摄像头采集的数据分辨率大小为 640x480 像素格式为 MJPG 。

```
book@100ask:~$ v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=MJPG
```

2.4.3 使用 v4l2-ctl 拍摄

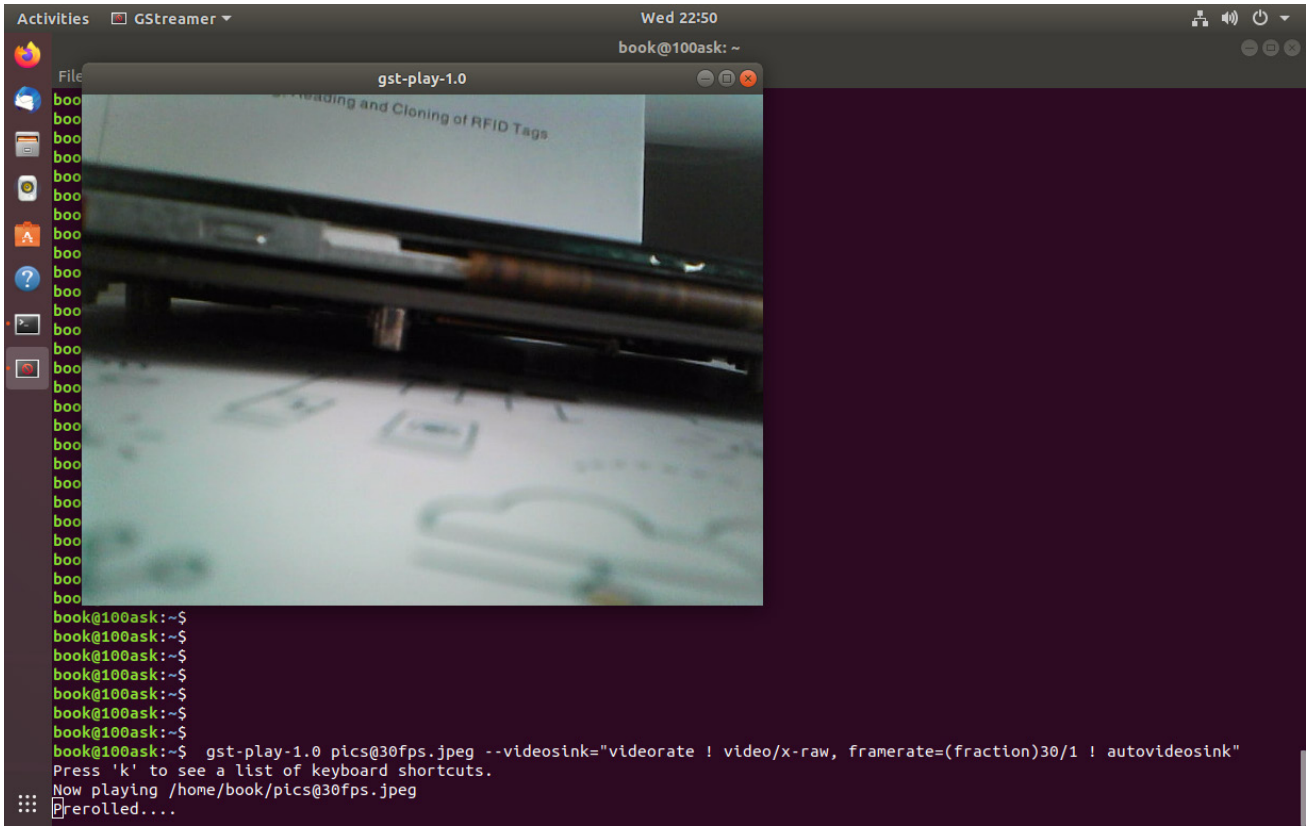
下面我们使用 v4l2-ctl 工具连续拍摄 30 张像素格式为 MJPG 分辨率为 640x480 的图像，并使用 gst-play-1.0 进行播放，下面示例为拍照示例。

```
book@100ask:~$ v4l2-ctl --set-parm=30;v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=MJPG --stream-mmap --stream-count=100 --
stream-to=pics@30fps.jpeg
```

```
book@100ask:~$ v4l2-ctl --set-parm=30;v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=MJPG --stream-mmap --stream-count
=100 --stream-to=pics@30fps.jpeg
Frame rate set to 30.000 fps
<<<<<<<<<<<<<<<<<<<<<<<<< 25.00 fps
<<<<<<<<<<<<<<<<<<<<<<<<< 25.00 fps
<<<<<<<<<<<<<<<<<<<<<<<<< 25.00 fps
<<<<<<<<<<<<<<<<<<<<<<<<<
```

拍摄完成后，我们可以使用 gst-play-1.0 命令来直接查看刚才捕获到的摄像头数据。

```
book@100ask:~$ gst-play-1.0 pics@30fps.jpeg --videosink="videorate ! video/x-raw, framerate=(fraction)30/1 ! autovideosink"
```



第3章 开发板下摄像头开发配置示例

3.1 配置开发环境

3.1.1 交叉编译安装 python

开发板内已经默认安装 python3.x 版本，可以直接在开发板内输入 python 即可进入 python 解释器界面。

```
[root@100ask:~]# python
Python 3.8.5 (default, Oct 13 2020, 10:45:42)
[GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("hello word")
hello word
>>>
```

3.1.2 交叉编译安装 opencv3

开发板内默认已经交叉编译安装好 opencv3 相应的库等，由于其编译依赖过于繁琐，我们使用 buildroot 构建工具进行构建，用户无需自行编译安装，可以直接使用 python 语言开发的 opencv3 ，如下所示验证 opencv 安装版本。

```
[root@100ask:~]# python
Python 3.8.5 (default, Oct 13 2020, 10:45:42)
[GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.4.9'
>>>
```

3.1.3 交叉编译安装 libv4l

开发板系统内已默认包含 libv4l 以及 v4l2-utils 等库和工具包，如果需要自行编译配置安装请参考资料 https://linuxtv.org/wiki/index.php/V4l-utils

3.2 运行示例

首先将摄像头接入开发板上任意 USB 接口，接入成功后终端会有如下打印信息。

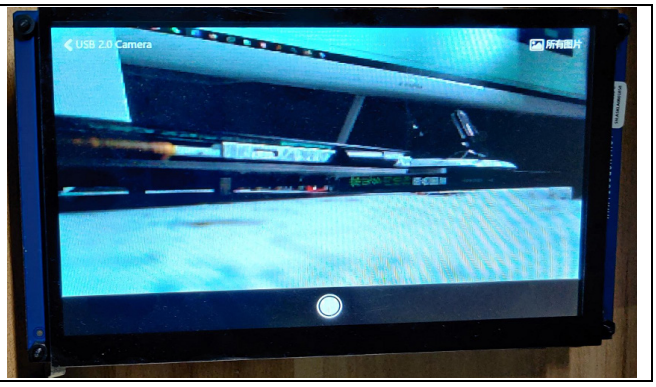
```
[root@100ask:~]# [  120.104399] usb 1-1.3: new high-speed USB device number 5 using ci_hdrc
[  120.330573] uvcvideo: Found UVC 1.00 device USB 2.0 Camera (038f:0541)
[  120.365729] input: USB 2.0 Camera as /devices/soc0/soc/2100000.aips-bus/2184200.usb/ci_hdrc.1/usb1/1-1/1-1.3/1-1.3:1.0/input/input4

[root@100ask:~]# lsusb
Bus 001 Device 001: ID 1d6b:0002
Bus 001 Device 005: ID 038f:0541
Bus 001 Device 003: ID 0bda:b720
Bus 001 Device 002: ID 0424:2514
[root@100ask:~]#
```

3.2.1 预览图像

使用默认 GUI 应用

在确保摄像头已经如上所示链接成功后，可以在默认的 GUI 桌面上点击 

此按钮，稍等数秒钟即可开始输出摄像头数据到屏幕。



使用 mjpg_streamer

mjpg_streamer 是一个可以 web 端访问摄像头数据的开源应用，首先保证开发板可以获取到 IP 地址，可以与 windows ubuntu 等三者互通，如下所示，当前示例中的开发板 IP 地址为 192.168.5.9 预览数据时会用到此 IP 地址。

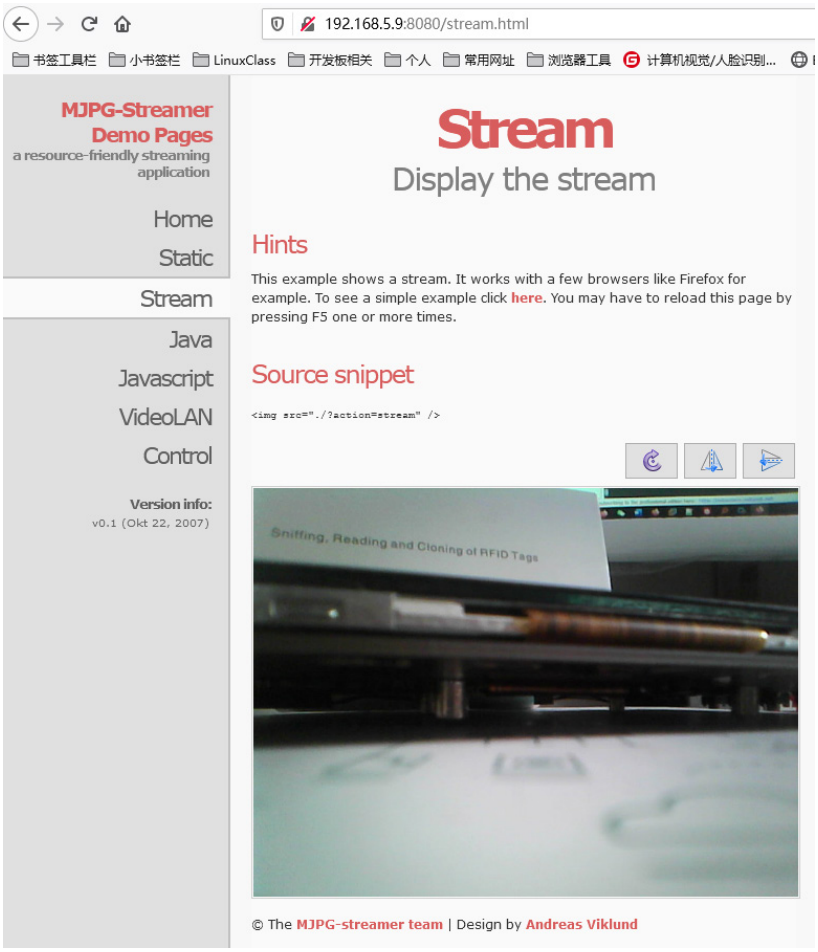
```
[root@100ask:~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:01:3F:2D:3E:4D
          inet addr:192.168.5.9  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::201:3fff:fe2d:3e4d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:9277 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4779 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:625621 (610.9 KiB)  TX bytes:34958365 (33.3 MiB)
```

确认 IP 之后我们在开发板上执行下述命令会将摄像头数据通过网络的方式展示出来，运行成功后参考下图所示。

mjpg_streamer -i "/usr/lib/mjpg-streamer/input_uvc.so -d /dev/video1 " -o "/usr/lib/mjpg-streamer/output_http.so -w /usr/share/mjpg-streamer/www/"

```
[root@100ask:~]# mjpg_streamer -i "/usr/lib/mjpg-streamer/input_uvc.so -d /dev/video1 " -o "/usr/lib/mjpg-streamer/output_http.so -w /usr/share/mjpg-streamer/www/"
MJPG Streamer Version: git rev: ee85cab468473923be4723e3969161f8d2200dc5
 i: Using V4L2 device.: /dev/video1
 i: Desired Resolution: 640 x 480
 i: Frames Per Second.: -1
 i: Format............: JPEG
 i: TV-Norm...........: DEFAULT
UVCIOC_CTRL_ADD - Error at Pan (relative): Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Tilt (relative): Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Pan Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Tilt Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Pan/tilt Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Focus (absolute): Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Pan (relative): Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Tilt (relative): Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Pan Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Tilt Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Pan/tilt Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Focus (absolute): Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at LED1 Mode: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at LED1 Frequency: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Disable video processing: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Raw bits per pixel: Inappropriate ioctl for device (25)
 o: www-folder-path......: /usr/share/mjpg-streamer/www/
 o: HTTP TCP port........: 8080
 o: HTTP Listen Address..: (null)
 o: username:password....: disabled
 o: commands.............: enabled
```

在确认开发板成功运行 mjpg_streamer 应用后，可以打开 windows 浏览器输入开发板 IP + 8080 端口即可访问，比如我的开发板 IP 为 192.168.5.9，则在浏览器输入 http://192.168.5.9:8080 按下回车即可预览摄像头数据。



3.2.3 编程预览

C 编程

参考 video2lcd 示例 https://gitee.com/weidongshan/RootfsPackages/tree/master/video2lcd 在 ubuntu 下使用开发板配套的交叉编后生成 video2lcd 应用通过 nfs ssh 等方式拷贝到开发板上执行。

如下图所示，首先我们需要在 ubuntu 终端上执行 git clone 获取 video2lcd 示例的源码并进入源码目录。

```
book@100ask:~$ git clone https://gitee.com/weidongshan/RootfsPackages/
book@100ask:~$ cd RootfsPackages/video2lcd
```

```
book@virtual-machine:~/Download$ git clone https://gitee.com/weidongshan/RootfsPackages/
Cloning into 'RootfsPackages'...
remote: Enumerating objects: 104, done.
remote: Counting objects: 100% (104/104), done.
remote: Compressing objects: 100% (95/95), done.
remote: Total 104 (delta 11), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (104/104), 396.66 KiB | 488.00 KiB/s, done.
Resolving deltas: 100% (11/11), done.
book@virtual-machine:~/Download$ cd RootfsPackages/video2lcd/
book@virtual-machine:~/Download/RootfsPackages/video2lcd$ ls
convert  display  include  log.txt  main.c  Makefile  Makefile.build  netprint_client.c  proj  render  video  说明.txt
book@virtual-machine:~/Download/RootfsPackages/video2lcd$
```

之后在 video2lcd 目录执行 make 命令来进行编译。

```
book@100ask:~/RootfsPackages/video2lcd$ make
```

```
book@virtual-machine:~/Download/RootfsPackages/video2lcd$ make
make -C ./ -f /home/book/Download/RootfsPackages/video2lcd/Makefile.build
make[1]: Entering directory '/home/book/Download/RootfsPackages/video2lcd'
make -C convert -f /home/book/Download/RootfsPackages/video2lcd/Makefile.build
make[2]: Entering directory '/home/book/Download/RootfsPackages/video2lcd/convert'
arm-buildroot-linux-gnueabihf-gcc -Wall -O2 -g -I /home/book/Download/RootfsPackages/video2lcd/include -Wp,-MD,.convert_manager.o.d -c -o convert_manage
r.o convert_manager.c
```

编译成功后生成的文件为 video2lcd 我们可以使用 ssh 或者开发板挂载 ubuntu nfs 目录等方式拷贝到开发板内。

```
book@virtual-machine:~/Download/RootfsPackages/video2lcd$ ls
built-in.o  convert  display  include  log.txt  main.c  main.o  Makefile  Makefile.build  netprint_client.c  proj  render  video  video2lcd  说明.txt
book@virtual-machine:~/Download/RootfsPackages/video2lcd$
```

拷贝成功后，我们先关闭默认的 GUI 桌面程序，再执行./video2lcd /dev/video1 来预览摄像头数据，打开后可以查看屏幕是否有摄像头数据输出。

```
[root@100ask:~]# /etc/init.d/S99myirhmi2 stop
[root@100ask:~]# ./video2lcd  /dev/video1
```

```
[root@100ask:~]# ./video2lcd  /dev/video1
/dev/video1 supports streaming i/o
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
Convert yuv2rgb, ret = 0
```

Python 编程

我们使用 python opencv3 来把摄像头数据保存到 output.avi 文件内，如下所示，新建一个 video2.py 文件并将下述代码拷贝进去，拷贝完成后，执行 chmod +x video2.py 来给文件增加可执行权限，最后执行 python3 video2.py 来运行此程序。

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
        frame = cv2.flip(frame,0)

        # write the flipped frame
        out.write(frame)

        cv2.imshow('frame',frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

```
    else:
        break


# Release everything if job is finished
cap.release()
out.release()
cv2.destroyAllWindows()
```

```
[root@100ask:~]# python3 video2.py
[ WARN:0] OpenCV | GStreamer warning: Cannot query video position: status=0, value=-1, duration=-1
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
qt.qpa.input: xkbcommon not available, not performing key mapping
```

执行成功后可以看到 LCD 上如下图类似的显示。

3.3 其它示例

3.3.1 获取摄像头参数

列出所有的摄像头设备

使用 v4l2-ctl --list-device 命令可以列出所有的摄像头设备。

```
[root@100ask:~]#   v4l2-ctl --list-device
```

```
[root@100ask:~]#  v4l2-ctl --list-device
pxp (pxp_v4l2):
        /dev/video0

USB 2.0 Camera (usb-ci_hdrc.1-1.3):
        /dev/video1

[root@100ask:~]#
```

查看某个设备的详细驱动信息

使用 v4l2-ctl -d /dev/video1 -D 命令可以列出/dev/video1 摄像头设备的详细驱动参数。

```
[root@100ask:~]#  v4l2-ctl -d /dev/video1 -D
```

```
[root@100ask:~]# v4l2-ctl -d /dev/video1 -D
Driver Info:
        Driver name      : uvcvideo
        Card type        : USB 2.0 Camera
        Bus info         : usb-ci_hdrc.1-1.3
        Driver version   : 4.9.88
        Capabilities     : 0x84200001
                Video Capture
                Streaming
                Extended Pix Format
                Device Capabilities
        Device Caps      : 0x04200001
                Video Capture
                Streaming
                Extended Pix Format
[root@100ask:~]#
```

列出摄像头支持哪些像素格式

使用 v4l2-ctl --device /dev/video1 --list-formats 可以列出/dev/video0 摄像头设备都支持哪些像素格式，如下所示支持 YUYV 和 JPEG 像素格式。

```
[root@100ask:~]#  v4l2-ctl --device /dev/video1 --list-formats
```

```
[root@100ask:~]# v4l2-ctl --device /dev/video1 --list-formats
ioctl: VIDIOC_ENUM_FMT
        Type: Video Capture

        [0]: 'YUYV' (YUYV 4:2:2)
        [1]: 'MJPG' (Motion-JPEG, compressed)
[root@100ask:~]#
```

列出摄像头都支持哪些控制参数

使用 v4l2-ctl --device /dev/video1 -L 命令可以查看/dev/video1 设备都支持哪些可以控制的参数。

```
[root@100ask:~]# v4l2-ctl --device  /dev/video1 -L
```

```
[root@100ask:~]# v4l2-ctl --device  /dev/video1 -L
                     brightness 0x00980900 (int)    : min=-255 max=255 step=1 default=0 value=0
                       contrast 0x00980901 (int)    : min=0 max=30 step=1 default=15 value=15
                     saturation 0x00980902 (int)    : min=0 max=127 step=1 default=30 value=30
                            hue 0x00980903 (int)    : min=-16000 max=16000 step=100 default=0 value=0
 white_balance_temperature_auto 0x0098090c (bool)   : default=1 value=1
                          gamma 0x00980910 (int)    : min=20 max=250 step=1 default=98 value=98
          power_line_frequency 0x00980918 (menu)   : min=0 max=2 default=1 value=1
                                0: Disabled
                                1: 50 Hz
                                2: 60 Hz
      white_balance_temperature 0x0098091a (int)    : min=2500 max=7000 step=1 default=5000 value=5000 flags=inactive
                     sharpness 0x0098091b (int)    : min=0 max=15 step=1 default=2 value=2
        backlight_compensation 0x0098091c (int)    : min=0 max=2 step=1 default=1 value=1
[root@100ask:~]#
```

列出摄像头支持哪些分辨率格式

使用 v4l2-ctl --device /dev/video1 --list-formats-ext 可以列出/dev/video0 设备支持哪些像素格式同时又包含哪些分辨率以及帧率。

```
[root@100ask:~]#  v4l2-ctl --device /dev/video1 --list-formats-ext
```

```
[root@100ask:~]# v4l2-ctl --device /dev/video1 --list-formats-ext
ioctl: VIDIOC_ENUM_FMT
        Type: Video Capture

        [0]: 'YUYV' (YUYV 4:2:2)
                Size: Discrete 640x480
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.040s (25.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 160x120
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.040s (25.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 320x240
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.040s (25.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 352x288
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.040s (25.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 800x600
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 1280x720
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 1280x1024
                        Interval: Discrete 0.083s (12.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 1600x1200
                        Interval: Discrete 0.111s (9.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
        [1]: 'MJPG' (Motion-JPEG, compressed)
                Size: Discrete 640x480
                        Interval: Discrete 0.033s (30.000 fps)
                        Interval: Discrete 0.040s (25.000 fps)
                        Interval: Discrete 0.050s (20.000 fps)
                        Interval: Discrete 0.067s (15.000 fps)
                        Interval: Discrete 0.100s (10.000 fps)
                        Interval: Discrete 0.200s (5.000 fps)
```

2.4.4 修改摄像头参数

使用 v4l2-ctl 可以设置一些参数，如下设置默认的帧率为 30fps。

```
[root@100ask:~]# v4l2-ctl --set-parm=30
```

也可以使用 v4l2-ctl 设置默认的摄像头采集数据分辨率和数据格式，如下所示设置摄像头采集的数据分辨率大小为 640x480 像素格式为 MJPG 。

```
[root@100ask:~]# v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=MJPG
```

2.4.5 使用 v4l2-ctl 拍摄

下面我们使用 v4l2-ctl 工具连续拍摄 30 张像素格式为 MJPG 分辨率为 640x480 的图像，并使用 gst-play-1.0 进行播放，下面示例为拍照示例。

```
[root@100ask:~]# v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=MJPG --stream-mmap --stream-count=1 --stream-to=pics.jpeg --
device /dev/video1
```

```
[root@100ask:~]# v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=MJPG --stream-mmap --stream-count=1 --stream-to=pics.jpeg --devi
ce /dev/video1
<
[root@100ask:~]#
```

拍摄成功后我们可以使用 GUI 内的图片功能来预览显示。

第4章 摄像头 V4I2 编程详解

4.1 V4L2 简介

Video for Linux two(Video4Linux2)简称 V4L2，是 V4L 的改进版。V4L2 是 linux 操作系统下一套用于采集图片、视频和音频数据的通用 API 接口，配合适当的视频采集设备和相应的驱动程序，可以实现图片、视频、音频等的采集。V4L2 像一个优秀的快递员，将视频采集设备的图像数据安全、高效的传递给不同需求的用户。

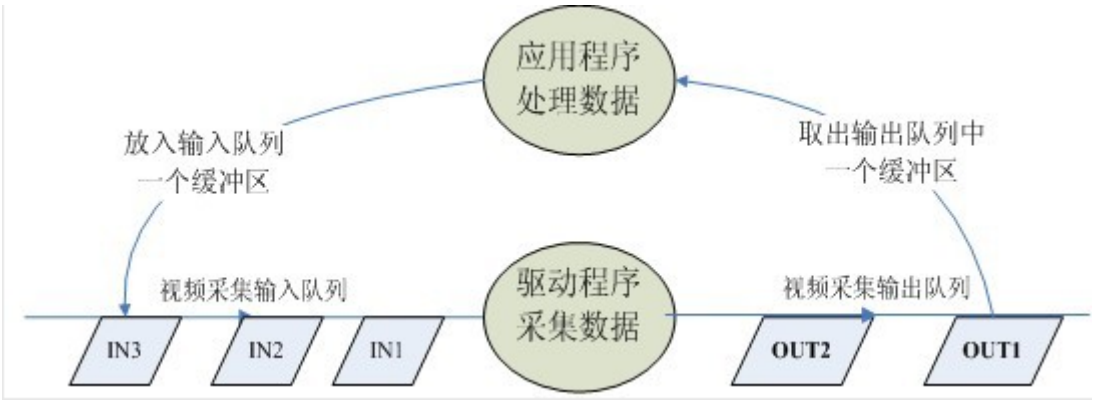
在 Linux 中，一切皆文件，所有外设都被看成一种特殊的文件，称为“设备文件”。视频设备也不例外，也可以可以看成是设备文件，可以像访问普通文件一样对其进行读写。V4L2 驱动的摄像头的设备文件一般是/dev/videoX（X 为任意数字，要与自己的设备相对应）。

V4L2 支持三种方式来采集图像：内存映射方式(mmap)、直接读取方式(read)和用户指针。内存映射的方式采集速度较快，一般用于连续视频数据的采集，实际工作中的应用概率更高；直接读取的方式相对速度慢一些，所以常用于静态图片数据的采集；用户指针使用较少，如有兴趣可自行研究。由于内存映射方式的应用更广泛，所以本文重点讨论内存映射方式的视频采集。

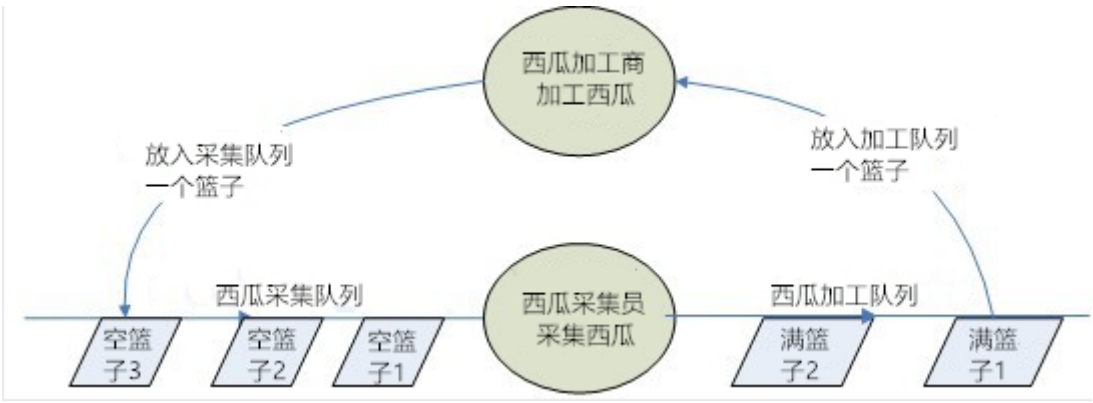
4.2 V4L2 视频采集原理

在通过 V4L2 采集图像之前，我们需要做的很多，但是很重要的一步是分配帧缓冲区，并将分配的帧缓冲区从内核空间映射到用户空间，然后将申请到的帧缓冲区在视频采集输入队列排队，剩下的就是等待视频数据的到来。但是，万一视频数据真的来了是怎么个流动过程呢？这个我们有必要了解一下。

当启动视频采集后，驱动程序开始采集一帧图像数据，会把采集的图像数据放入视频采集输入队列的第一个帧缓冲区，一阵图像数据就算采集完成了。第一个帧缓冲区存满一帧图像数据后，驱动程序将该帧缓冲区移至视频采集输出队列，等待应用程序从输出队列取出，应用程序取出图像数据可以对图像数据进行处理或存储操作，然后将帧该缓冲区放入视频采集输入队列的尾部。驱动程序接下来采集下一帧数据，放入第二个缓冲区，同样的帧缓冲区存满一帧数据后，驱动程序将该缓冲区移至视频采集输出队列，应用程序将该帧缓冲区的图像数据取出后又将该帧缓冲区放入视频输入队列尾部，这样循环往复就实现了循环采集。流程如下图所示：



为了更好的理解这个过程，我们可以把“应用程序处理数据”比喻成“西瓜加工商加工西瓜”，“V4L2 驱动程序采集数据”比喻成“西瓜采集员采集西瓜”，事先“西瓜加工商”会给“西瓜采集员”准备几个空篮子，然后“西瓜采集员”守着几个空篮子等待“瓜农”（图像采集设备，例如：摄像头）将空篮子装满，当“空篮子 1”被“瓜农”装满以后，“西瓜采集员”会将装满西瓜的篮子放到“西瓜加工队列”等待“西瓜加工商”取走加工，当“西瓜加工商”取走装满西瓜的篮子中的西瓜的时候，“西瓜加工商”会将空篮子放回到事先给“西瓜采集员”准备好的西瓜采集队列的尾部。当“瓜农”装满下一个空篮子的时候，“西瓜采集员”同样的将装满西瓜的篮子放到“西瓜加工队列”等待“西瓜加工商”取走加工。这样，整个过程会持续不断的继续下去。



4.3 V4L2 程序实现流程

使用 V4L2 进行视频采集，一般分为 5 个步骤：

（1）打开设备，进行初始化参数设置，通过 V4L2 接口设置视频图像的采集窗口、采集的点阵大小和格式；

（2）申请图像帧缓冲，并进行内存映射，将这些帧缓冲区从内核空间映射到用户空间，便于应用程序读取、处理图像数据；

（3）将帧缓冲进行入队操作，启动视频采集；

（4）驱动开始视频数据的采集，应用程序从视频采集输出队列取出帧缓冲区，处理完后，将帧缓冲区重新放入视频采集输入队列，循环往复采集连续的视频数据；

（5）释放资源，停止采集工作。

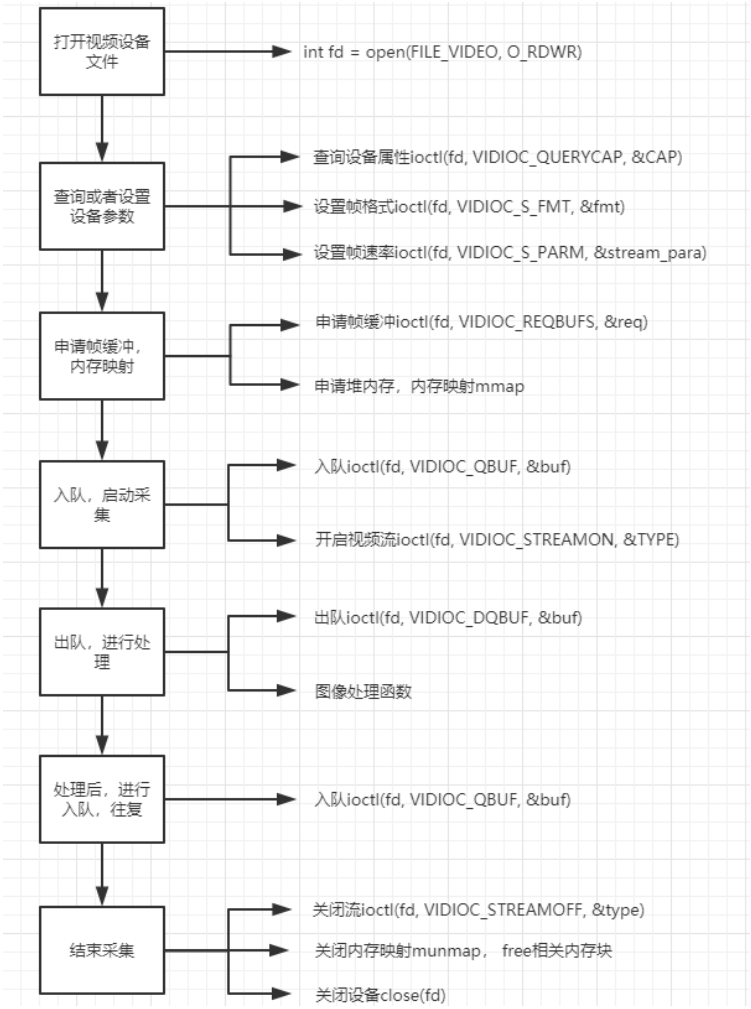
在进行 V4L2 开发中，常用的命令标识符如下：

（1）VIDIOC_REQBUFS:分配内存；

（2）VIDIOC_QUERYBUF:把 VIDIOC_REQBUFS 中分配的数据缓存转换成物理地址；

（3）VIDIOC_QUERYCAP:查询驱动功能；

（4）VIDIOC_ENUM_FMT:获取当前驱动支持的视频格式；
（5）VIDIOC_S_FMT:设置当前驱动的视频捕获格式；
（6）VIDIOC_G_FMT:读取当前驱动的视频捕获格式；
（7）VIDIOC_TRY_FMT:验证当前驱动的显示格式；
（8）VIDIOC_CROPCAP:查询驱动的修剪功能；
（9）VIDIOC_S_CROP:设置视频信号的边框；
（10）VIDIOC_G_CROP:读取视频信号的边框；
（11）VIDIOC_QBUF:把数据从缓存中读取出来；
（12）VIDIOC_DQBUF:把数据放回缓存队列；
（13）VIDIOC_STREAMOP:开始视频显示函数；
（14）VIDIOC_STREAMOFF:结束视频显示函数；
（15）VIDIOC_QUERYSTD:检查当前视频设备支持的标准，例如 PAL 或 NTSC；
这些 IO 调用，有些是必须的，有些是可选择的。
具体流程如下图所示：



4.4 V4L2 程序实例刨析

V4L2 的代码主要位于 video2lcd/video/v4l2.c 文件中，接下来就针对上文 V4L2 程序实现流程和流程中使用的重要数据结构，结合 v4l2.c 文件中的代码进行说明。代码支持内存映射和直接读取两种方式，由于内存映射方式应用更广泛，本文只详细说明内存映射方式，直接读取方式与内存映射方式类似，可自行研究。

4.4.1 打开设备

应用程序能够使用阻塞模式或非阻塞模式打开视频设备，如果使用非阻塞模式调用视频设备，即使尚未捕获到信息，驱动依旧会把缓存（DQBUFF）里的东西返回给应用程序。如果使用非阻塞的方式打开摄像头设备，第 2 行代码中 open 函数的第二个参数修改为 O_RDWR | O_NONBLOCK 即可。

```
70      iFd = open(strDevName, O_RDWR);
71      if (iFd < 0)
72      {
73          DBG_PRINTF("can not open %s\n", strDevName);
74          return -1;
75      }
```

4.4.2 查询设备属性

查询设备属性需要使用 struct v4l2_capability 结构体，该结构体描述了视频采集设备的 driver 信息。

```
01 struct v4l2_capability
02 {
03      __u8 driver[16];          // 驱动名字
04      __u8 card[32];            // 设备名字
05      __u8 bus_info[32];        // 设备在系统中的位置
06      __u32 version;            // 驱动版本号
07      __u32 capabilities;       // 设备支持的操作
08      __u32 reserved[4];        // 保留字段
09 };
```

通过 VIDIOC_QUERYCAP 命令来查询 driver 是否合乎规范。因为 V4L2 要求所有 driver 和 device 都支持这个 ioctl。所以，通过 VIDIOC_QUERYCAP 命令是否成功来判断当前 device 和 driver 是否符合 V4L2 规范。当然，这个命令执行成功的同时还能够得到设备足够的信息，如 struct v4l2_capability 结构体所示内容。86~98 行代码检查当前设备是否为 capture 设备，并检查使用内存映射还是直接读的方式获取图像数据。

```
78      iError = ioctl(iFd, VIDIOC_QUERYCAP, &tV4l2Cap);
79      memset(&tV4l2Cap, 0, sizeof(struct v4l2_capability));
80      iError = ioctl(iFd, VIDIOC_QUERYCAP, &tV4l2Cap);
81      if (iError) {
82        DBG_PRINTF("Error opening device %s: unable to query device.\n", strDevName);
83        goto err_exit;
84      }
85
86      if (!(tV4l2Cap.capabilities & V4L2_CAP_VIDEO_CAPTURE))
```

```
87      {
88        DBG_PRINTF("%s is not a video capture device\n", strDevName);
89            goto err_exit;
90      }
91
92        if (tV4l2Cap.capabilities & V4L2_CAP_STREAMING) {
93              DBG_PRINTF("%s supports streaming i/o\n", strDevName);
94        }
95
96        if (tV4l2Cap.capabilities & V4L2_CAP_READWRITE) {
97              DBG_PRINTF("%s supports read i/o\n", strDevName);
98        }
```

4.4.3 显示所有支持的格式

显示所有支持的格式需要用到 struct v4l2_fmtdesc 结构体，该结构体描述当前 camera 支持的格式信息。

```
01 struct v4l2_fmtdesc
02 {
03      __u32 index;                    // 要查询的格式序号，应用程序设置
04      enum v4l2_buf_type type;    // 帧类型，应用程序设置
05      __u32 flags;                    // 是否为压缩格式
06      __u8 description[32];       // 格式名称
07      __u32 pixelformat;              //所支持的格式
08      __u32 reserved[4];              // 保留
09 };
```

使用 VIDIOC_ENUM_FMT 命令查询当前 camera 支持的所有格式。struct v4l2_fmtdesc 结构体中 index 要设置，从 0 开始；enum v4l2_buf_type type 也要设置，如果使用的是 camera 设备，则 enum v4l2_buf_type type 要设置为 V4L2_BUF_TYPE_VIDEO_CAPTURE，因为 camera 是 CAPTURE 设备。结构体中的其他内容 driver 会填充。其中__u32 pixelformat 参数在设置图像帧格式时需要使用。

```
100     memset(&tFmtDesc, 0, sizeof(tFmtDesc));
101     tFmtDesc.index = 0;
102     tFmtDesc.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
103     while ((iError = ioctl(iFd, VIDIOC_ENUM_FMT, &tFmtDesc)) == 0) {
104           if (isSupportThisFormat(tFmtDesc.pixelformat))
105           {
106                 ptVideoDevice->iPixelFormat = tFmtDesc.pixelformat;
107                 break;
108           }
109                 tFmtDesc.index++;
110     }
```

4.4.4 设置图像帧格式

设置图像格式需要用到 struct v4l2_format 结构体，该结构体描述每帧图像的具体格式，包括帧类型以及图像的长、宽等信息。

```
01 struct v4l2_format
02 {
03      enum v4l2_buf_type type;                // 帧类型，应用程序设置
04      union fmt
05      {
06          struct v4l2_pix_format pix;     // 视频设备使用
07          structv 4l2_window win;
08          struct v4l2_vbi_format vbi;
09          struct v4l2_sliced_vbi_format sliced;
10          __u8 raw_data[200];
11      };
12 };
```

struct v4l2_format 结构体需要设置 enum v4l2_buf_type type 和 union fmt 中的 struct v4l2_pix_format pix。enum v4l2_buf_type type 因为使用的是 camera 设备，camera 是 CAPTURE 设备，所以设置成 V4L2_BUF_TYPE_VIDEO_CAPTURE。struct v4l2_pix_format pix 设置一帧图像的长、宽和格式等，由于要适配 LCD 输出，所以长、宽设置为 LCD 支持的长、宽，如 124~125 行所示。

```
119     /* set format in */
120     GetDispResolution(&iLcdWidth, &iLcdHeigt, &iLcdBpp);
121     memset(&tV4l2Fmt, 0, sizeof(struct v4l2_format));
122     tV4l2Fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
123     tV4l2Fmt.fmt.pix.pixelformat = ptVideoDevice->iPixelFormat;
124     tV4l2Fmt.fmt.pix.width          = iLcdWidth;
125     tV4l2Fmt.fmt.pix.height         = iLcdHeigt;
126     tV4l2Fmt.fmt.pix.field          = V4L2_FIELD_ANY;
127
128     /* 如果驱动程序发现无法某些参数(比如分辨率),
129      * 它会调整这些参数, 并且返回给应用程序
130      */
131     iError = ioctl(iFd, VIDIOC_S_FMT, &tV4l2Fmt);
132     if (iError)
133     {
134             DBG_PRINTF("Unable to set format\n");
135         goto err_exit;
136     }
```

4.4.5 申请缓冲区

相关结构体如下，该结构体描述申请的缓冲区的基本信息。

```
01 struct v4l2_requestbuffers
02 {
```

```
03      __u32 count;                        // 缓冲区内缓冲帧的数目
04      enum v4l2_buf_type type;            // 缓冲帧数据格式
05      enum v4l2_memorymemory;             // 区别是内存映射还是用户指针方式
06      __u32 reserved[2];
07 };
```

申请一个拥有四个缓冲帧的缓冲区，__u32 count 为缓冲帧的数目；enum v4l2_buf_type type 和前文一样，同样设置成 V4L2_BUF_TYPE_VIDEO_CAPTURE；enum v4l2_memorymemory 用来区分是内存映射还是用户指针，我们使用内存映射的方式，所以设置成 V4L2_MEMORY_MMAP。

```
140     /* request buffers */
141     memset(&tV4l2ReqBuffs, 0, sizeof(struct v4l2_requestbuffers));
142     tV4l2ReqBuffs.count = NB_BUFFER;
143     tV4l2ReqBuffs.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
144     tV4l2ReqBuffs.memory = V4L2_MEMORY_MMAP;
145
146     iError = ioctl(iFd, VIDIOC_REQBUFS, &tV4l2ReqBuffs);
147     if (iError)
148     {
149             DBG_PRINTF("Unable to allocate buffers.\n");
150         goto err_exit;
151     }
```

4.4.6 将申请的缓冲帧从内核空间映射到用户空间

相关结构体如下，该结构体表示一帧图像数据的基本信息，包含序号、缓冲帧长度和缓冲帧地址等信息。

```
01 struct v4l2_buffer
02 {
03      __u32 index;                        //buffer 序号
04      enum v4l2_buf_type type;            //buffer 类型
05      __u32 byteused;                     //buffer 中已使用的字节数
06      __u32 flags;                        // 区分是 MMAP 还是 USERPTR
07      enum v4l2_field field;
08      struct timeval timestamp;           // 获取第一个字节时的系统时间
09      struct v4l2_timecode timecode;
10      __u32 sequence;                     // 队列中的序号
11      enum v4l2_memory memory;            //IO 方式，被应用程序设置
12      union m
13      {
14          __u32 offset;                   // 缓冲帧地址，只对 MMAP 有效
15          unsigned long userptr;
16      };
17      __u32 length;                       // 缓冲帧长度
18      __u32 input;
19      __u32 reserved;
20 };
```

将内核空间的帧缓冲映射到用户空间，需要两个数据接收帧缓冲的长度和地址，我们需要自己定义一个结构体，该结构体位于 video2lcd/include/video_manager.h 文件中，其中 iVideoBufMaxLen 接收帧缓冲的长度，pucVideBuf 接收帧缓冲地址。

```
16 struct VideoDevice {
17     int iFd;
18     int iPixelFormat;
19     int iWidth;
20     int iHeight;
21
22     int iVideoBufCnt;
23     int iVideoBufMaxLen;
24     int iVideoBufCurIndex;
25     unsigned char *pucVideBuf[NB_BUFFER];
26
27     /* 函数 */
28     PT_VideoOpr ptOPr;
29 };
```

以下代码使用 VIDIOC_QUERYBUF 命令和 mmap 函数将内核空间的缓冲区映射到用户空间。VIDIOC_QUERYBUF 命令的使用需要参数 struct v4l2_buffer 结构体，结构体中的 type、memory 和 index 参数需要设置，type 和 memory 和前文中的设置一样，分别设置成 V4L2_BUF_TYPE_VIDEO_CAPTURE 和 V4L2_MEMORY_MMAP，index 参数表示申请的缓冲帧的标号，从 0 开始，包含申请的所有缓冲帧。

mmap 函数原形为：

```
01 void *mmap(void*addr, size_t length, int prot, int flags, int fd, off_t offset);
```

参数具体的含义：

1. addr：映射起始地址，一般为 NULL，让内核自动选择；
2. length：被映射内存块的长度；
3. prot：标志映射后能否被读写，其值为 PROT_EXEC,PROT_READ,PROT_WRITE,PROT_NONE；
4. flags：确定此内存映射能否被其他进程共享，可设置为 MAP_SHARED 或 MAP_PRIVATE；
5. fd：设备文件句柄；
6. offset：确定映射后的内存地址

```
156         /* map the buffers */
157         for (i = 0; i < ptVideoDevice->iVideoBufCnt; i++)
158         {
159             memset(&tV4l2Buf, 0, sizeof(struct v4l2_buffer));
160             tV4l2Buf.index = i;
161             tV4l2Buf.type     = V4L2_BUF_TYPE_VIDEO_CAPTURE;
162             tV4l2Buf.memory = V4L2_MEMORY_MMAP;
163             iError = ioctl(iFd, VIDIOC_QUERYBUF, &tV4l2Buf);
164             if (iError)
165             {
166                 DBG_PRINTF("Unable to query buffer.\n");
167                 goto err_exit;
168             }
169
```

```
170             ptVideoDevice->iVideoBufMaxLen = tV4l2Buf.length;
171             ptVideoDevice->pucVideBuf[i] = mmap(0 /* start anywhere */ ,
172                                     tV4l2Buf.length, PROT_READ, MAP_SHARED, iFd,
173                                     tV4l2Buf.m.offset);
174             if (ptVideoDevice->pucVideBuf[i] == MAP_FAILED)
175             {
176                 DBG_PRINTF("Unable to map buffer\n");
177                 goto err_exit;
178             }
179         }
```

4.4.7 将申请的缓冲帧放入队列，并启动数据流

184~194 行代码为使用 VIDIOC_QBUF 命令，将申请的缓冲帧依次放入缓冲帧输入队列，等待被图像采集设备依次填满；

```
181         /* Queue the buffers. */
182         for (i = 0; i < ptVideoDevice->iVideoBufCnt; i++)
183         {
184             memset(&tV4l2Buf, 0, sizeof(struct v4l2_buffer));
185             tV4l2Buf.index = i;
186             tV4l2Buf.type    = V4L2_BUF_TYPE_VIDEO_CAPTURE;
187             tV4l2Buf.memory = V4L2_MEMORY_MMAP;
188             iError = ioctl(iFd, VIDIOC_QBUF, &tV4l2Buf);
189             if (iError)
190             {
191                 DBG_PRINTF("Unable to queue buffer.\n");
192                 goto err_exit;
193             }
194         }
```

4.4.8 启动捕捉图像数据

启动捕捉图像数据使用 VIDIOC_STREAMON 命令，当该命令执行成功后，便可以等待图像数据的到来。

```
356 /*****************************************************************************
357 * 函数名称：V4l2StartDevice
358 * 功能描述：开始捕捉图像数据
359 * 输入参数：ptVideoDevice
360 * 输出参数：无
361 * 返 回 值：无
362 * 修改日期                版本号          修改人              修改内容
363 * ---------------------------------------------
364 * 2020/02/16            V1.0       zhenhua                 创建
365 *****************************************************************************/
```

```
366 static int V4l2StartDevice(PT_VideoDevice ptVideoDevice)
367 {
368     int iType = V4L2_BUF_TYPE_VIDEO_CAPTURE;
369     int iError;
370
371     iError = ioctl(ptVideoDevice->iFd, VIDIOC_STREAMON, &iType);
372     if (iError)
373     {
374         DBG_PRINTF("Unable to start capture.\n");
375         return -1;
376     }
377     return 0;
378 }
```

4.4.9 出列采集的帧缓冲，并处理图像数据，然后再将数据帧入列

我们可以使用 VIDIOC_DQBUF 命令，等待缓冲帧的到来，当有缓冲帧被放入视频输出缓冲队列，我们便可以采到一帧图像。接收到图像我们可以对图像进行操作，例如保存、压缩或者 LCD 输出等。

```
243 /*******************************************************************************
244 *  函数名称：V4l2GetFrameForStreaming
245 *  功能描述：从图像数据流中获取一帧图像数据
246 *  输入参数：ptVideoDevice
247 *            ptVideoBuf
248 *  输出参数：无
249 *  返 回 值：无
250 *  修改日期        版本号      修改人          修改内容
251 * -----------------------------------------------
252 * 2020/02/16      V1.0    zhenhua         创建
253 *******************************************************************************/
254 static int V4l2GetFrameForStreaming(PT_VideoDevice ptVideoDevice, PT_VideoBuf ptVideoBuf)
255 {
256     struct pollfd tFds[1];
257     int iRet;
258     struct v4l2_buffer tV4l2Buf;
259
260     /* poll */
261     tFds[0].fd      = ptVideoDevice->iFd;
262     tFds[0].events = POLLIN;
263
264     iRet = poll(tFds, 1, -1);
265     if (iRet <= 0)
266     {
267         DBG_PRINTF("poll error!\n");
268         return -1;
```

```
269     }
270
271     /* VIDIOC_DQBUF */
272     memset(&tV4l2Buf, 0, sizeof(struct v4l2_buffer));
273     tV4l2Buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
274     tV4l2Buf.memory = V4L2_MEMORY_MMAP;
275     iRet = ioctl(ptVideoDevice->iFd, VIDIOC_DQBUF, &tV4l2Buf);
276     if (iRet < 0)
277     {
278             DBG_PRINTF("Unable to dequeue buffer.\n");
279             return -1;
280     }
281     ptVideoDevice->iVideoBufCurIndex = tV4l2Buf.index;
282
283     ptVideoBuf->iPixelFormat          = ptVideoDevice->iPixelFormat;
284     ptVideoBuf->tPixelDatas.iWidth    = ptVideoDevice->iWidth;
285     ptVideoBuf->tPixelDatas.iHeight = ptVideoDevice->iHeight;
286     ptVideoBuf->tPixelDatas.iBpp      = (ptVideoDevice->iPixelFormat == V4L2_PIX_FMT_YUYV) ? 16 : \
287                                         (ptVideoDevice->iPixelFormat == V4L2_PIX_FMT_MJPEG) ? 0 :   \
288                                         (ptVideoDevice->iPixelFormat == V4L2_PIX_FMT_RGB565) ? 16 :   \
289                                         0;
290     ptVideoBuf->tPixelDatas.iLineBytes      = ptVideoDevice->iWidth * ptVideoBuf->tPixelDatas.iBpp / 8;
291     ptVideoBuf->tPixelDatas.iTotalBytes     = tV4l2Buf.bytesused;
292     ptVideoBuf->tPixelDatas.aucPixelDatas = ptVideoDevice->pucVideBuf[tV4l2Buf.index];
293     return 0;
294 }
```

当我们从缓冲帧输出队列取出一个缓冲帧，取出图像数据后我们需要将缓冲帧重新放回到视频输入缓冲队列，该操作还是使用 VIDIOC_QBUF 命令，放回缓冲帧输入队列后继续等待被填满。

```
296 /**********************************************************************
297 * 函数名称：V4l2PutFrameForStreaming
298 * 功能描述：将取出的帧缓冲重新放回图像输入队列
299 * 输入参数：ptVideoDevice
300                 ptVideoBuf
301 * 输出参数：无
302 * 返 回 值：无
303 * 修改日期                 版本号          修改人              修改内容
304 * ---------------------------------------------
305 * 2020/02/16              V1.0        zhenhua                 创建
306 ***********************************************************************/
307 static int V4l2PutFrameForStreaming(PT_VideoDevice ptVideoDevice, PT_VideoBuf ptVideoBuf)
308 {
309     /* VIDIOC_QBUF */
310     struct v4l2_buffer tV4l2Buf;
311     int iError;
```

```
312
313     memset(&tV4l2Buf, 0, sizeof(struct v4l2_buffer));
314     tV4l2Buf.index  = ptVideoDevice->iVideoBufCurIndex;
315     tV4l2Buf.type   = V4L2_BUF_TYPE_VIDEO_CAPTURE;
316     tV4l2Buf.memory = V4L2_MEMORY_MMAP;
317     iError = ioctl(ptVideoDevice->iFd, VIDIOC_QBUF, &tV4l2Buf);
318     if (iError)
319     {
320         DBG_PRINTF("Unable to queue buffer.\n");
321         return -1;
322     }
323     return 0;
324 }
```

4.4.10 停止捕捉图像数据

停止采集图像数据，首先使用 VIDIOC_STREAMOFF 命令，关闭捕获图像数据。同时要注意取消内存映射和关闭句柄，防止不必要的内存泄漏。代码 390~407 行为停止捕捉图像数据命令；代码 227~241 行为取消内存映射和关闭句柄。

```
380 /*****************************************************************************
381 * 函数名称：V4l2StopDevice
382 * 功能描述：停止捕捉图像数据
383 * 输入参数：ptVideoDevice
384 * 输出参数：无
385 * 返 回 值：无
386 * 修改日期        版本号      修改人        修改内容
387 * ---------------------------------------------
388 * 2020/02/16      V1.0    zhenhua         创建
389 *****************************************************************************/
390 static int V4l2StopDevice(PT_VideoDevice ptVideoDevice)
391 {
392     int iType = V4L2_BUF_TYPE_VIDEO_CAPTURE;
393     int iError;
394
395     iError = ioctl(ptVideoDevice->iFd, VIDIOC_STREAMOFF, &iType);
396     if (iError)
397     {
398         DBG_PRINTF("Unable to stop capture.\n");
399         return -1;
400     }
401     return 0;
402 }
403
404 static int V4l2GetFormat(PT_VideoDevice ptVideoDevice)
```

```
405 {
406     return ptVideoDevice->iPixelFormat;
407 }
```

```
217 /**************************************************************************
218 * 函数名称：V4l2ExitDevice
219 * 功能描述：退出采集设备，取消帧缓冲映射和关闭句柄
220 * 输入参数：ptVideoDevice
221 * 输出参数：无
222 * 返 回 值：无
223 * 修改日期                版本号          修改人              修改内容
224 * -----------------------------------------------
225 * 2020/02/16              V1.0        zhenhua                 创建
226 **************************************************************************/
227 static int V4l2ExitDevice(PT_VideoDevice ptVideoDevice)
228 {
229     int i;
230     for (i = 0; i < ptVideoDevice->iVideoBufCnt; i++)
231     {
232         if (ptVideoDevice->pucVideBuf[i])
233         {
234             munmap(ptVideoDevice->pucVideBuf[i], ptVideoDevice->iVideoBufMaxLen);
235             ptVideoDevice->pucVideBuf[i] = NULL;
236         }
237     }
238
239     close(ptVideoDevice->iFd);
240     return 0;
241 }
```