H.264 USB Camera
Linux Development Specification

深圳市爱立普科技有限公司

AILIPU TECHNOLOGY CO., LTD

2014.9.16　V1.0

Catalogue

Chapter One: H264 USB Camera Brief Introduction

H264 USB camera parameters：

Sensor	OV9712
Lens Size	1/4
Resolution	720P 1280x720
Frame Rate	30fps
Image Format	H264 MJPEG YUV2
Interface	USB2.0 High Speed
Free Drive Protocol	USB Video Class(UVC)
Adjustable Parameters	Brightness, Contrast, Color Saturation, Hue/Tone, Definition, Gama, Gain, White Balance, Backlighting, Exposure
H264 Quality Adjustment	Bit Rate, Compression Ratio
Programmable OSD	Support
Audio Input	Support digital audio, single and dual channel optional
SNR (Signal to Noise Ratio)	40dB
Dynamic Range	69dB
Sensitivity	3.7V/lux-sec@550nm
Minimum Illumination	0.1Lux
IR night vision	Support， with IR cut switching function
Power supply mode	USB BUS POWER
Operation Voltage	5V
Operation Current	120mA~220mA

H264 USB Camera Software Introduction:

The software only can be used in Linux system, includes PC or embedded system, advice use version 2.6.26 or above, for lower kernel version, we didn't test before.

Linux_uvc_driver   Linux_UVC_TestAP

SDK have two parts：

Part One: Drive.

Drive divides into uvc_2.6.3 and uvc_3.3.8 two kinds. 3.3.8 apply to linux-3.3.8 or higher kernel version 2.6.36 is suitable for linux-3.3.8 version below. Before using, please check your version, and choose suitable SDK.

Drive program provides two video channels for the system. video0 is MJPEG/YUV channel， video1 is for H264 stream. Drive program provides H.264 usb camera specific XU Ctrl operation.

Part Two: Application Program.

Application program is a demo program, which is wrote based on V4L2, contain H264 XU Ctrl API

Chapter Two: Development Drive

1. Develop in PC Linux System

Enter in driver sources directory, copy Makefile_PC as Makefile，compile directly

```
cd linux_uvc_driver/uvc_3.3.8/
cp Makefile_PC Makefile
make
```

Get driver module uvcvideo_h264.ko
Please unload any original host system UVC driver：

```
sudo rmmod uvcvideo
```

Load new UVC driver module

```
sudo insmod ./uvcvideo_h264.ko
```

2. Develop in embedded Linux system

Firstly check the system kernel version, if the version is 3.3.8 higher, then use uvc_3.3.8 driver, if the Version is uvc_3.3.8 lower, then use uvc_2.6.36 driver.
Put the driver directory under kernel source directory /dirvers/media/video/ , name uvc_h264
The driver directory Makefile_Embedded, copy as Makefile, to replace the original Makefile
Modify Kconfig file under directory /drivers/media/video， Import uvc_h264 configuration:

```
source "drivers/media/video/uvc/Kconfig"
source "drivers/media/video/uvc_h264/Kconfig"
source "drivers/media/video/gspca/Kconfig"
```

Modify Kconfig file under directory /drivers/media/video:

```
obj-$(CONFIG_USB_VIDEO_CLASS)   += uvc/
obj-$(CONFIG_USB_VIDEO_CLASS_H264)  += uvc_h264/
obj-$(CONFIG_VIDEO_SAA7164)     += saa7164/
```

Import configuration in .config：

```
CONFIG_V4L_USB_DRIVERS=y
#CONFIG_USB_VIDEO_CLASS is not set
CONFIG_USB_VIDEO_CLASS_INPUT_EVDEV=y
CONFIG_USB_VIDEO_CLASS_H264=y
CONFIG_USB_GSPCA=m
```

Compile the driver in kernel, or compile to module.

Chapter Three: Example program:

1. Compile source

```
cd  H264_UVC_TestAP
make
```

2. Save H264 video data

```
./H264_UVC_TestAP  /dev/video1  -c  -f  H264  -r
```

3. Save JPEG Image

```
./H264_UVC_TestAP  /dev/video0  -c  -f  mjpg  -S
```

4. Dual-stream video preview

1、Use luvcview application open /dev/video0 to get real-time display

```
./luvcview  -d  /dev/video0  -s  640x480
```

2、In another terminal(or thread), use H264_UVC_TestAP application start recording in h264 format

```
./H264_UVC_TestAP  /dev/video1  -c  -f  H264  -r
```

When you open dual-steam setup, please note:
If set both at preview and recording, preview resolution can not be higher than recording resolution.

Chapter Four: Expansion Unit (XU) Control

1. Set the bit rate, the compression ratio

Bit rate (br) represents the number of bits in one second that indicates the encoded (compressed)the audio and video data, the higher the bit rate is, the better the quality of audio and video is , but the larger will the encoded file be; On the contrary ,the less bit rate , we will get opposite results.

Compression rate (qp)is the quantization parameter that represents the recorded compression rate , It decided the compression condition of space details of the images. If QP is small, then most details will be reserved; If QP is large, many details are lost, but the distortion of the image strengthens the decline of the quality. QP and the bit rate are inversely related.

Setting bit rate and compression ratio:

```
./H264_UVC_TestAP /dev/video1 --xuset-br 4000000 --xuset-qp 31 -c -f H264 -r
```

Directions: The number after --xuset-xx is the value to be set. At this place suggest Bit rate at least 4000000, If at this time the CPU usage and memory still has surplus, can increase the br value appropriately, suggest 6000000, max no more than 16000000。

2. Set OSD menu

```
./H264_UVC_TestAP  /dev/video1 --xuset-oe 1 --xuset-oc 1 --xuset-rtc 2008 08
01 10 10 10 -c -f H264 -r
```

Instruction:

--xuset-oe

Set OSD Display enable parameter : 1 display menu, 0 no display menu

--xuset-oc

Set OSD color parameter:

--xuset-rtc

Set OSD time format : year month date hour minute second

Notification : Parameters following /dev/video1 have no order one after another , user can plus and minus acoording to his requirements, but please notice that the set such as --xuset-br must be followed by one parameter, otherwise it will prompt am error。

About more –xuset orders , please use ./H264_UVC_TestAP –h to check.

Chapter Five: Analysis of Development Process

1. Get H264 video stream

Developing Applications need to include 3 files：h264_xu_ctrls.c   v4l2uvc.c nalu.c
Applications need to include the header file：

```
#include "v4l2uvc.h"
#include "h264_xu_ctrls.h"
#include "nalu.h"
```

The variables used：

```
void *mem0[V4L_BUFFERS_MAX];
unsigned int nbufs = V4L_BUFFERS_DEFAULT;
struct v4l2_buffer buf0;
```

Open video device initialization：

```
    struct v4l2_capability cap;
    int dev;

    dev = open("/dev/video1", O_RDWR);
    memset(&cap, 0, sizeof cap);
    ioctl(dev, VIDIOC_QUERYCAP, &cap);
```

Set video format：

```
    struct v4l2_format fmt;

    memset(&fmt, 0, sizeof fmt);
    fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    fmt.fmt.pix.width = width;
    fmt.fmt.pix.height = height;
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_H264;
    fmt.fmt.pix.field = V4L2_FIELD_ANY;

    ioctl(dev, VIDIOC_S_FMT, &fmt);
```

Set frame rate:

```
    struct v4l2_streamparm parm;

    memset(&parm, 0, sizeof parm);
    parm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

    ioctl(dev, VIDIOC_G_PARM, &parm);
    parm.parm.capture.timeperframe.numerator = 1;
    parm.parm.capture.timeperframe.denominator = framerate;

    ioctl(dev, VIDIOC_S_PARM, &parm);
```

Apply video buffer:

```
    struct v4l2_requestbuffers rb;

    memset(&rb, 0, sizeof rb);
    rb.count = nbufs;
    rb.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    rb.memory = V4L2_MEMORY_MMAP;

    ioctl(dev, VIDIOC_REQBUFS, &rb);
    nbufs = rb.count;
```

Mapped cache:

```
    for (i = 0; i < nbufs; ++i) {
        memset(&buf0, 0, sizeof buf0);
        buf0.index = i;
        buf0.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf0.memory = V4L2_MEMORY_MMAP;
        ioctl(dev, VIDIOC_QUERYBUF, &buf0);

        mem0[i] = mmap(0, buf0.length, PROT_READ, MAP_SHARED, dev,
buf0.m.offset);
    }
```

Insert the cache formation:

```
    for (i = 0; i < nbufs; ++i) {
        memset(&buf0, 0, sizeof buf0);
        buf0.index = i;
        buf0.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf0.memory = V4L2_MEMORY_MMAP;
        ret = ioctl(dev, VIDIOC_QBUF, &buf0);
    }
```

Open video Steam:

```
    int type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    ioctl(dev, VIDIOC_STREAMON, &type);
```

Capture H264 video stream, save to file

```
FILE *rec_fp = fopen("Record.H264", "wb");
while(1)
{
    memset(&buf0, 0, sizeof buf0);
    buf0.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf0.memory = V4L2_MEMORY_MMAP;

    ioctl(dev, VIDIOC_DQBUF, &buf0);
    fwrite(mem0[buf0.index], buf0.bytesused, 1, rec_fp);
    ioctl(dev, VIDIOC_QBUF, &buf0);
}
```

2. Set Parameters:

Set bit rate

```
double m_BitRate = 0.0;
XU_H264_Get_BitRate(video_fd, &m_BitRate);
if(m_BitRate < 0 ){
    printf("XU_H264_Get_BitRate Failed\n");
}
if(XU_H264_Set_BitRate(video_fd, 8000000.00) < 0 ){
    LOGE("XU_H264_Set_BitRate Failed\n");
}
```

Set subtitle time

```
if(XU_OSD_Set_RTC(video_fd, osd_rtc_year, osd_rtc_month, osd_rtc_day,
osd_rtc_hour, osd_rtc_minute, osd_rtc_second) <0) {
    printf("XU_OSD_Set_RTC Failed\n");
}
```

Closed subtitle display except time

```
#define CARCAM_PROJECT        1

XU_OSD_Set_CarcamCtrl(video_fd, 0, 0, 0);
```

Open Caption

```
if(XU_OSD_Set_Enable(video_fd,1, 1) <0)  { // open display
    printf("XU_OSD_Set_Enable Failed\n");
}
if(XU_OSD_Set_Enable(video_fd,0, 0) <0)  { // close display
printf("XU_OSD_Set_Disable Failed\n");
}
```