ESM6800V 支持 CSI 接口摄像头

英创信息技术有限公司

2017 年 10 月

近年来，随着计算机、网络以及图像处理、传输技术的飞速发展，摄像头在工业控制领域的应用也越来越广泛了，所以英创公司在低成本核心板 ESM6800 的基础上，增加了一款可以支持 CSI（COMS Sensor Interface）接口摄像头的型号：ESM6800V。因为 CSI 是一个标准的视频输出接口，视频处理芯片可以直接输出，不需要涉及到 USB 接口摄像头所需的视频压缩芯片以及 USB 接口芯片，所以较市面上普通的 USB 摄像头来说，CSI 接口的摄像头更便宜，配合 ESM6800V 形成了一个低成本的图像处理方案。

一、使用简介

ESM6800V 在 ESM6800H 的基础上去掉一路网口和 6 路扩展串口，增加了一路 CSI 接口用于摄像头的连接，本文就以英创嵌入式板卡 ESM6800V 为例来介绍对于 CSI 摄像头的支持，ESM6800V 的内核版本为 Linux-4.1.14，CSI 摄像头选用 Omnivision 公司 130W 像素的 ov9650 和 500W 像素的 ov5640，这两款摄像头以其高性价比得以广泛应用，同时英创公司也已经将 ov5640 和 ov9650 摄像头的驱动移植到板卡中了。

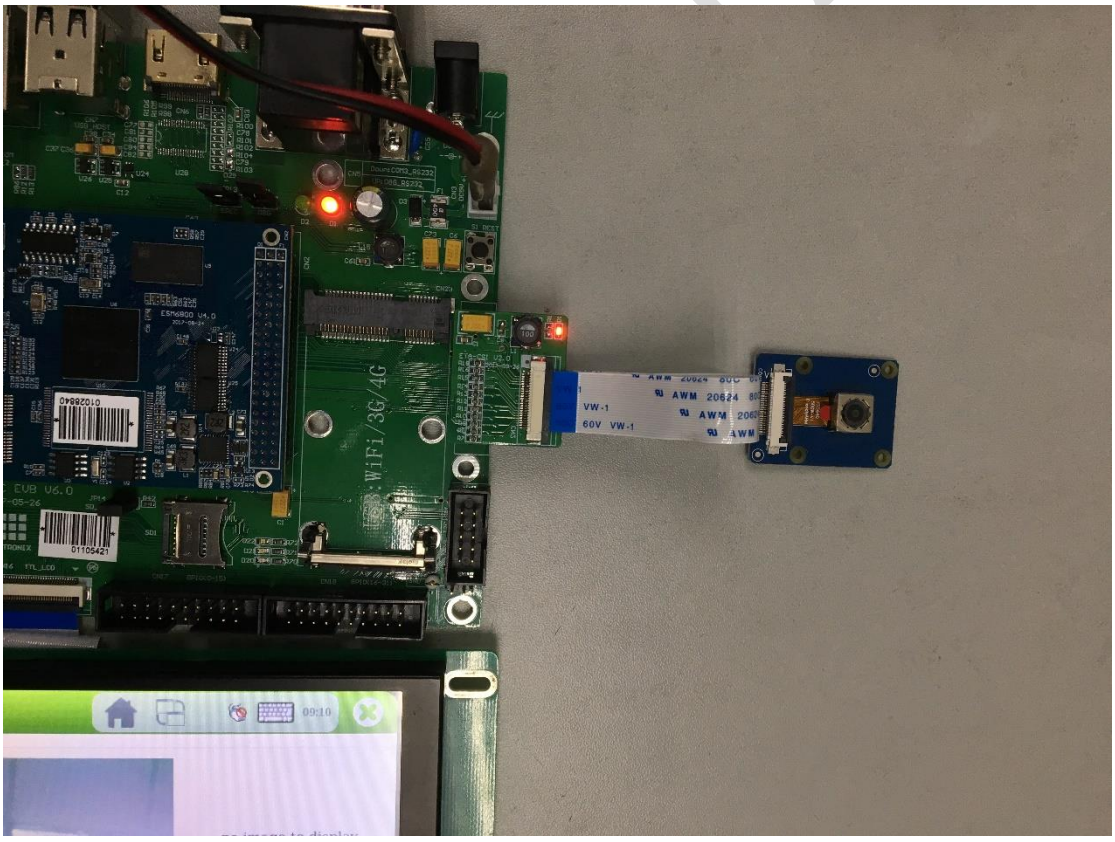
- 硬件连接

关于硬件接口，ESM6800V 同样符合 ESMARC 规范，因为 ESM6800 的各个版本都没有 ISA 总线接口，所以 ESM6800H 将 ESMARC 规范中 ISA 接口管脚定义为 7 路扩展串口的管脚，和 ESM6800H 不同的，ESM6800V 则是将 ESMARC 规范中 ISA 接口的管脚定义为 CSI 信号接口的管脚，ESMARC 通用评估底板的接口具体的定义请参考下面这张表格：

信号名称及简要描述		EVB CN29		信号名称及简要描述	
ESM6800V	ESMARC	PIN#	PIN#	ESMARC	ESM6800V
NC	RESET#	1	2	ADV#	CSI_HSYNC
CSI_DATA00	ISA_SD0	3	4	SD4	CSI_DATA04

CSI_DATA01	SD1	5	6	SD5	CSI_DATA05
CSI_DATA02	SD2	7	8	SD6	CSI_DATA06
CSI_DATA03	SD3	9	10	SD7	CSI_DATA07
NC	MSL，主从选择	11	12	WE#	CSI_VSYNC
#Reset	GPIO9	13	14	RD#	CSI_PIXCLK
电源管理	GPIO8	15	16	CS#	CSI_MCLK
I2C_SCL	GPIO25	17	18	VCC，5V 直流供电	VCC，同左
I2C_SDA	GPIO24	19	20	GND，公共地	GND，公共地

如果客户使用 ESMARC 通用评估底板测试摄像头功能，将摄像头接到我们提供的转接板上，然后把转接板插入 CN25（ISA 接口），硬件就已经正确连接上，如下图：



ESMARC 通用评估底板和摄像头的连接

连接好摄像头上电，系统会自动识别到摄像头并加载相应驱动，加载驱动后会自动生成设备节点：“/dev/video0"，应用程序可以操作该设备节点对摄像头进行图像的采集和控制。

下面就来介绍软件上的设置和操作。

二、Qt 摄像头应用程序简介

- V4L2 概述

CSI 摄像头都是用了 V4L2 驱动提供的标准 API 来操作的。Video for Linux 2 简称 V4L2，是 V4L 的改进版。V4L2 是 Linux 操作系统下用于采集图片、视频和音频数据的 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。在视频监控系统和嵌入式多媒体终端中都有广泛的应用。V4L2 支持两种方式来采集图像：内存映射方式(mmap)和直接读取方式(read)。在这里我们使用内存映射的方式来进行视频采集。应用程序通过 V4L2 接口采集视频数据可以分为五个步骤：

①打开视频设备文件，进行视频采集的参数初始化，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；

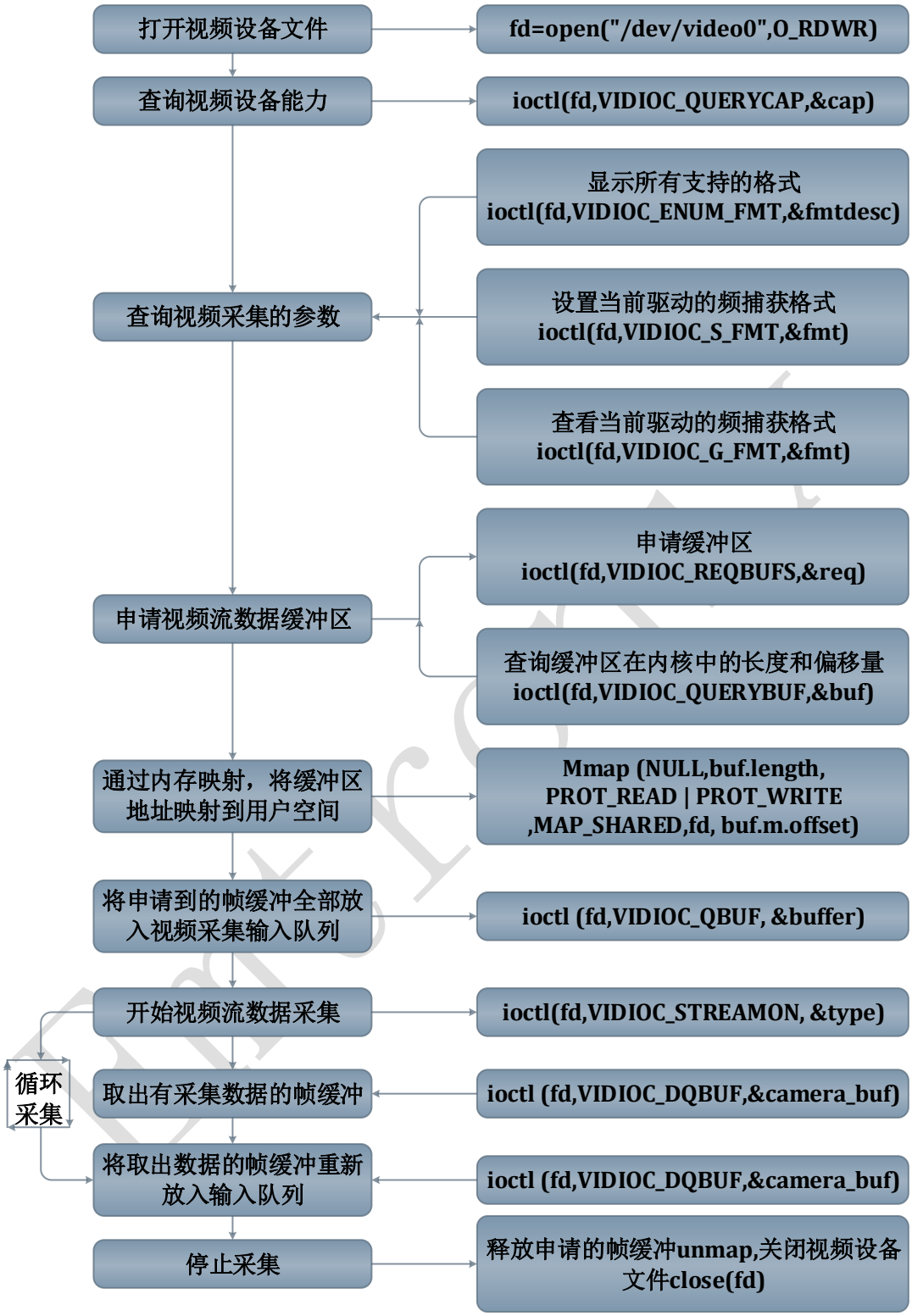
②申请若干视频采集的帧缓冲区，并将这些帧缓冲区从内核空间映射到用户空间，便于应用程序读取/处理视频数据；

③将申请到的帧缓冲区在视频采集输入队列排队，并启动视频采集；

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停止视频采集。

可以参考下图：

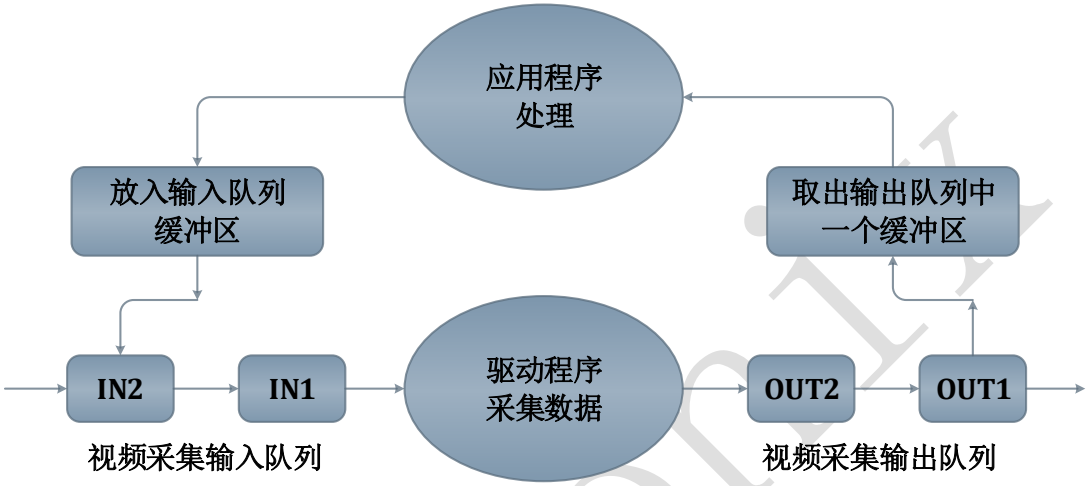


视频数据采集流程

可以看到每一个步骤都是通过 ioctl 这个接口去设置一些参数来实现的，启动视频采集后，驱动程序开始采集数据，并把采集的数据放入视频采集输入队列的第一个帧缓冲区，当一帧数据采集完成，也就是第一个帧缓冲区存满数据以后，驱动程序将这一个缓冲区移至视

频采集输出队列，等待应用程序取出。驱动程序接下来继续采集下一帧数据，并放入第二个帧缓冲区，同样帧缓冲区存满数据后，被放入视频采集输出队列。

应用程序从视频采集输出队列中取出含有视频数据的帧缓冲区，处理帧缓冲区中的视频数据，如存储或压缩。如果需要连续采集，应用程序需要将处理完数据的帧缓冲区重新放入视频采集输入队列，如图所示。



视频数据采集队列

接下来结合程序来具体看一看通过 V4L2 接口来操作摄像头的一些重要的步骤：

- 基本设置

  打开设备文件:

```
int fd;
fd=open("/dev/video0",O_RDWR);
```

  获取设备的基本信息,包括驱动版本号,设备支持操作，所支持的格式等:

```
/*show all the support format*/
memset(&fmtdesc, 0, sizeof(fmtdesc));
fmtdesc.index = 0 ;                    /* the number to check */
fmtdesc.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;

/* check video decive driver capability */
if(ret=ioctl(fd, VIDIOC_QUERYCAP, &cap)<0)
{
        fprintf(stderr, "fail to ioctl VIDEO_QUERYCAP \n");
        exit(EXIT_FAILURE);
}
```

```
/*judge wherher or not to be a video-get device*/
if(!(cap.capabilities & V4L2_BUF_TYPE_VIDEO_CAPTURE))
{
        fprintf(stderr, "The Current device is not a video capture device \n");
        exit(EXIT_FAILURE);
}

/*judge whether or not to supply the form of video stream*/
if(!(cap.capabilities & V4L2_CAP_STREAMING))
{
        printf("The Current device does not support streaming i/o\n");
        exit(EXIT_FAILURE);
}

printf("\ncamera driver name is : %s\n",cap.driver);
printf("camera device name is : %s\n",cap.card);
printf("camera bus information: %s\n",cap.bus_info);

/*display the format device support*/
printf("\n");
while(ioctl(fd,VIDIOC_ENUM_FMT,&fmtdesc)!=-1)
{
        printf("support device %d.%s\n",fmtdesc.index+1,fmtdesc.description);
        fmtdesc.index++;
}

printf("\n");
```

设置视频的制式和帧格式，制式包括 PAL，NTSC，帧的格式个包括宽度和高度等:

```
/*set the form of camera capture data*/
tv_fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;          /*v4l2_buf_typea,camera must
use V4L2_BUF_TYPE_VIDEO_CAPTURE*/
tv_fmt.fmt.pix.width = 680;
tv_fmt.fmt.pix.height = 480;
tv_fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;/*V4L2_PIX_FMT_YYUV*/
tv_fmt.fmt.pix.field = V4L2_FIELD_NONE;             /*V4L2_FIELD_NONE*/
if (ioctl(fd, VIDIOC_S_FMT, &tv_fmt)< 0)
{
        fprintf(stderr,"VIDIOC_S_FMT set err\n");
        exit(-1);
        close(fd);

}
```

- 申请缓冲帧

向驱动申请帧缓冲，一般不超过五个：

```
struct v4l2_requestbuffers req;
req.count=2;
req.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
req.memory=V4L2_MEMORY_MMAP;
//申请帧缓冲
ret=ioctl(fd,VIDIOC_REQBUFS,&req);
if(ret<0)
{
      printf("failture VIDIOC_REQBUFS\n");
      return -1;
}
if (req.count < 1)
{
      printf("Insufficient buffer memory\n");
      return -1;

}
```

将申请到的帧缓冲映射到用户空间，这样就能够直接操作帧缓冲了：

```
buffers =(buffer*)calloc (req.count, sizeof (*buffers));
if (!buffers) {
      fprintf (stderr,"Out of memory/n");
      exit(EXIT_FAILURE);
}

for (n_buffers = 0; n_buffers < req.count; ++n_buffers)
{
      struct v4l2_buffer buf;
      memset(&buf,0,sizeof(buf));
      buf.type =V4L2_BUF_TYPE_VIDEO_CAPTURE;
      buf.memory =V4L2_MEMORY_MMAP;
      buf.index =n_buffers;
      // 查询序号为n_buffers 的缓冲区，得到其起始物理地址和大小
      if (-1 == ioctl(fd, VIDIOC_QUERYBUF, &buf))
      {
            printf("failture VIDIOC_QUERYBUF\n");
            return -1;
      }
      buffers[n_buffers].length= buf.length;
      // 映射内存
```

```
buffers[n_buffers].start=mmap (NULL,buf.length,PROT_READ |
PROT_WRITE ,MAP_SHARED,fd, buf.m.offset);
if (MAP_FAILED == buffers[n_buffers].start)
{
    printf("failture mmap\n");
    return -1;
}

}
```

将申请到的帧缓冲全部入队列，以便存放采集到的数据：

```
for (i = 0; i< req.count; ++i)
{
    struct v4l2_buffer buffer;
    buffer.type =V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buffer.memory =V4L2_MEMORY_MMAP;
    buffer.index = i;
    //将缓冲帧放入队列尾
    ioctl (fd,VIDIOC_QBUF, &buffer);

}
```

- 数据采集

开始视频的采集：

```
type =V4L2_BUF_TYPE_VIDEO_CAPTURE;
ioctl (fd,VIDIOC_STREAMON, &type);
```

取出队列中以取得采集数据的帧缓冲，获得原始采集数据，因为这个摄像头支持的格式为 YUYV，所以需要转换才能够显示以及存储：

```
struct v4l2_buffer camera_buf;
CLEAR (camera_buf);
camera_buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
camera_buf.memory = V4L2_MEMORY_MMAP;
//取出一个缓冲帧
i1 = ioctl (fd, VIDIOC_DQBUF, & usr_buf);
if(i1<0)
{
    printf("failture\n");
    return -1;
}
```

```
//read process space's data to a file
process_image(buffer[camera_buf.index].start, buffer[camera_buf.index].length);
```

- 数据转换

可通过如下函数将 YUYV 格式的数据转换为 RGB 格式：

```
int yuyv_to_rgb(unsigned char *yuv, unsigned char *rgb, unsigned int width,
unsigned int height)
{
    unsigned int in, out = 0;
    unsigned int pixel_16;
    unsigned char pixel_24[3];
    unsigned int pixel32;
    int y0, u, y1, v;

    for(in = 0; in < width * height * 2; in += 4)
    {
        pixel_16 =
                yuv[in + 3] << 24 |
                yuv[in + 2] << 16 |
                yuv[in + 1] <<  8 |
                yuv[in + 0];
        y0 = (pixel_16 & 0x000000ff);
        u  = (pixel_16 & 0x0000ff00) >>  8;
        y1 = (pixel_16 & 0x00ff0000) >> 16;
        v  = (pixel_16 & 0xff000000) >> 24;
        pixel32 = convert_yuv_to_rgb_pixel(y0, u, v);
        pixel_24[0] = (pixel32 & 0x000000ff);
        pixel_24[1] = (pixel32 & 0x0000ff00) >> 8;
        pixel_24[2] = (pixel32 & 0x00ff0000) >> 16;
        rgb[out++] = pixel_24[0];
        rgb[out++] = pixel_24[1];
        rgb[out++] = pixel_24[2];
        pixel32 = convert_yuv_to_rgb_pixel(y1, u, v);
        pixel_24[0] = (pixel32 & 0x000000ff);
        pixel_24[1] = (pixel32 & 0x0000ff00) >> 8;
        pixel_24[2] = (pixel32 & 0x00ff0000) >> 16;
        rgb[out++] = pixel_24[0];
        rgb[out++] = pixel_24[1];
        rgb[out++] = pixel_24[2];
    }
    return 0;
```

```
}


int convert_yuv_to_rgb_pixel(int y, int u, int v)
{
    unsigned int pixel32 = 0;
    unsigned char *pixel = (unsigned char *)&pixel32;
    int r, g, b;
    r = y + (1.370705 * (v-128));
    g = y - (0.698001 * (v-128)) - (0.337633 * (u-128));
    b = y + (1.732446 * (u-128));
    if(r > 255) r = 255;
    if(g > 255) g = 255;
    if(b > 255) b = 255;
    if(r < 0) r = 0;
    if(g < 0) g = 0;
    if(b < 0) b = 0;
    pixel[0] = r ;
    pixel[1] = g ;
    pixel[2] = b ;
    return pixel32;

}
```

- 循环采集

将缓冲帧重新入队列尾,这样可以循环采集：

```
//将缓冲重新入队列尾
i1=ioctl (fd, VIDIOC_QBUF, &camera_buf);
if(i1<0)
{
    printf("failture VIDIOC_QBUF\n");
    return -1;

}
```

- 关闭摄像头

如果需要关闭摄像头，先停止视屏采集，释放申请的帧缓冲，最后关闭设备节点：

```
//停止视频的采集。VIDIOC_STREAMOFF
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (-1 == ioctl(fd, VIDIOC_STREAMOFF, &type))
```

```
        printf("VIDIOC_STREAMOFF");
    for (i = 0; i < n_buffers; ++i)
    if (-1 == munmap (buffers->start, buffers->length))
        printf ("munmap error");
    free(buffers);
    //关闭视频设备
    close (fd);
```

所以通过这一套通用的 V4L2 接口来操作摄像头的工作流程：

打开设备—> 检查和设置设备属性—>设置帧格式—> 设置一种输入输出方法（缓冲区管理）—> 循环获取数据—> 关闭设备。通过这几个步骤已经可以操作摄像头来获取数据。下面来看看如何与 Qt 结合，将前面的代码与 Qt 界面结合起来。

- 和 Qt 界面结合

在 Qt 中主要就是实现两个功能，一个是通过界面控制摄像头的数据获取，另一个是通过界面显示摄像头所拍摄下来的图片。摄像头的初始化设置，包括格式等参数的设置可以在 Qt 界面的构造函数中完成。

通过界面来显示摄像头的内容，可以在 Qt 中设置一个定时器，每次定时器出发就将摄像头的内容读取出来显示，这里是先将从摄像头中读取出来的 YUYV 格式的原始数据转换为 RGB 格式，再通过 Qt 接口显示：

```
void MainWindow::captureImage()
{
        ui->ImageView->clear();

        start_capture(fd);
        mainloop(fd);
        QImage image(640,480,QImage::Format_RGB888);
        unsigned char * p_bits = image.bits();
        for(int i=0; i<640*480*3; i+=3)
        {
            p_bits[i] = rgb[i];
            p_bits[i+1] = rgb[i+1];
            p_bits[i+2] = rgb[i+2];
        };
        ui->ImageView->setPixmap(QPixmap::fromImage(image));

        stop_capture(fd);
```

```
}
```

关于拍摄图片的显示问题，Qt 中提供了很多实现的方法，比如可以在界面中采用一个 label 来显示，主要代码如下：

```
image=new QImage(pictrue_name);
image->load(pictrue_name);

ui->ImageView->setPixmap(QPixmap::fromImage(image));
```

将摄像头获取的数据写入文件中，再通过 GraphicsView 显示出来。这样就实现了 Qt 程序和摄像头操作的结合，详细的代码请参考例程。

例程的效果如下图所示：