中南林業科技大學

毕 业 论 文

学生姓名：吴世群　　学　号：20134573

学　　院：计算机与信息工程学院

专业年级：2013 级电子信息工程

题　　目：基于 ARM 的远程视频监控系统的实现与设计

指导教师：赵伟志　　（高级实验师）

评阅教师：

2017 年　5 月

摘要

远程视频监控系统是基于ARM架构的S3C2440芯片的开发板及30万像素的OV7740摄像头实现的，此系统基于 Linux 2.6 版本的 V4L2 的视频采取框架实现的，分析了系统的基本硬件架构,并实现了 OV7740 摄像头图像数据获取的驱动程序，在此论文中介绍了 OV7740 和开发板的硬件结构，还有原理，并且以模块化的思想实现视频监控系统。由于硬件设计过于复杂，所以硬件的设计是基于现成的，并不是自己开发的。图像显示与图像采集技术的发展与广泛运用使得人们的生活与工作简便、快捷。本论文中提出是基于 Linux V4L2 的 ARM 架构的远程移动视频监控系统,然后自己搭建视频 Web 服务器，系统可在经连接成功后的 Android 手机中的自制 APP 进行视频监控和图像抓拍。

关键词　S3C2440　OV7740　V4L2　视频监控　Android

Title

Design of remote video surveillance system based on ARM

Abstract

Remote video surveillance system is based on the ARM framework of S3C2440 chip development board and 300 thousand pixel OV7740 camera, and this framework for V4L2's video based on version 2.6 of Linux.

This paper is designed to analyze the basic hardware architecture of the system based on the features of different modules were made a brief introduction; the software design of the system gives a detailed description of presentation the basic idea of the system initialization and processes, and related peripheral equipment modules in the program; and gives the overall design concept of the system to achieve a stable operation of the system.

The development and extensive application of image display and image acquisition technology make people's life and work easier and faster. In this paper, an implementation scheme of video surveillance based on ARM embedded development platform is proposed. I bought these hardware devices on the Internet , because the design of hardware is too complex. Video image acquisition and display under Linux based on V4L2,and build video Web server, Video surveillance system can be displayed in the mobile phone APP.

Keywords: S3C2440 OV7740 V4L2 remote video surveillance Android

目　录

1 绪论

在此论文课题中本人基于Linux系统、JZ2440开发板、OV7740摄像头，还有Android手机设备实现的。本人通过设计一个Web服务器让多台终端共同访问的效果，实现真正意义的物联网设备。

1.1 远程视频监控的发展

随着计算机网络互联网科技不断的发展，远程视频监控，从最早的模拟数据视频监控系统到现在的数字化监控系统的发展，使视频监控变得更加智能化，如现在比较流行的网络直播，还有视频会议，视频通话，还有人工识别等等。随着各行各业的发展，视频监控已经渗透到人类生活的每个地方。不可否认视频传输给社会的发展带来更加便捷，增加了人对事物的体验，促进社会的发展。

远程传输监控系统，远程视频监控从模拟信号，开始越来越趋于数字化和网络化。计算机网络普及，成为发展远程端的视频监控提供了很好的条件。从最早的通过标准电话线还有宽带还有数据线直接链接，最主要的还是路由器的普及，使WIFI成为局域网传输数据的最佳的选择。该系统的视频帧频率大概30万左右，并且Web服务端允许10移动客户端进行连接和获取相同的数据。实现出多端监控的作用。

在大学期间的社会实践以及企业实习的过程中让我深刻的知道视频监控巨大的发展前景以及对于人们生活的重要性。目前国内外物联网蓄势待发，还有人工智能识别强势发展，智能视频监控将成为下一个科技发展的巨大爆发点，而视频监控正事物联网发展必不可少的桥梁。基于ARM远程视频监控系统的实现是在Linux 系统平台上，因为Linux是有Linus开发的一种开源的操作系统，并且在现社会的发展中起了重要的地位，所以本人选择烧录Linux操作系统在JZ2440设备上，在开发板中视频进过Linux应用层的处理，还有安卓客户端处理显示等功能形成整个远程视频监控系统，真正实现了物联网的系统，但是此系统中对于现在的发展来看有待提升，如智能人脸识别模块的加入，还有音频同步等等功能的加入，会是此系统如虎添翼。

2 系统整体结构设计

该系统整体架构比较复杂，本人采用模块化的思想进行开发。使程序进行弱耦合，更易于开发调试。

2.1 视频监控系统的整体架构的设计

该系统是由 ARM9 架构的 S3C2440 芯片上运行 linux 系统而开发的远程视频监控系统，通过 WIFI 网卡发送数据给 Android 手机客户端，数据的传输是基于生活中最常见的 HTTP 协议发送。系统整体架构可以看图 2-1：

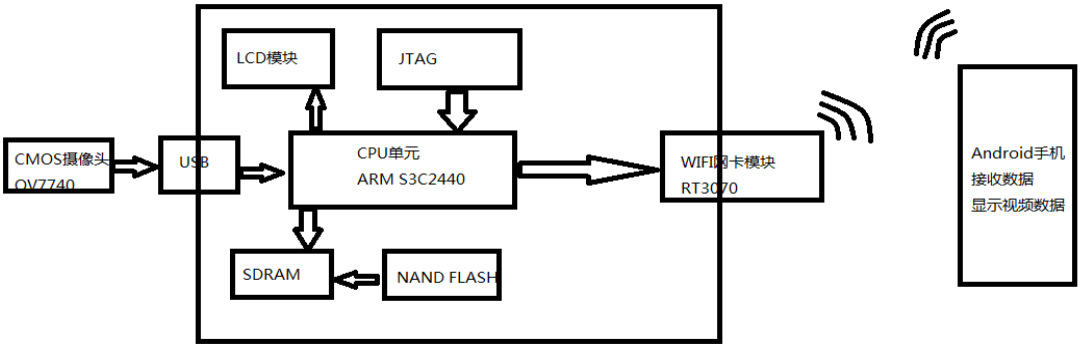


图 2-1 整体架构框图

该系统是基于 ARM 架构的 S3C2440 芯片的开发板及 30 万像素的 OV7740 摄像头实现，在处理器烧写的是 Linux 操作系统，然后运行 Linux 系统，接收来自 OV7740 摄像头的视频数据流，再进行格式的转换，再传送视频数据给安卓手机完成监控功能。

2.2 S3C2440 处理器简介

S3C2440 是三星公司开发的一款基于 ARM920T 内核和 0.18um CMOS 工艺的 16/32 位 RISC 微处理器，适用于低成本、低功耗、高性能的手持设备或其它电子产品。S3C2440 中集成了以下一些通用的系统外设和接口：

1. 1.8V 内核电压，3.3V 存储电压，3.3V I/O 电压。
2. 包括 16KB 的 I-Cache（指令高速缓存）、16KB 的 D-Cache（数据高速缓存）和 MMU（存储管理单元）
3. 外部的存储控制器（SDRAM 控制器和片选逻辑）
4. LCD 控制器（最高支持 4K 色的 STN 和 16M 色的 TFT），包括一个 LCD DMA
5. 4 个带外部请求管脚的 DMA

6. 3 个 UART、2 个 SPI
7. 1 个 IIC-BUS 控制器、1 个 IIS-BUS 控制器
8. SD 主机接口，兼容 Multi-Media Card Protocol V2.11
9. 2 端口 USB 主设备接口、1 端口 USB 从设备接口（V1.1）
10. 4 个 PWM 时钟和 1 个内部时钟
11. 117 个 GPIO、24 个外部中断源
13. 功率控制：Normal, Slow, Idle, Power-off 四种模式
14. 8 路 10-bit ADC 和触摸屏接口
15. 带 PLL 的片上时钟发生器

芯片架构如图 2-2 所示：

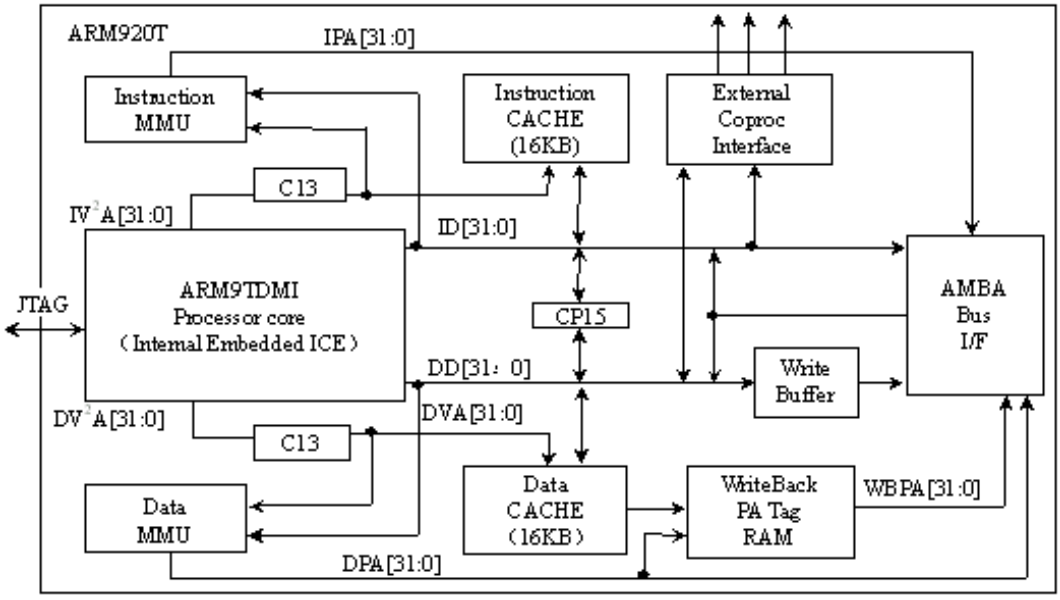


图 2-2 S3C2440 芯片图

3 OV7740 摄像头硬件原理

OV7740 摄像头内部芯片主要的还是图像识别传感器，而其他的一些模块处于数据传输与处理的作用。

3.1 OV7740 概述

OV7740 摄像头是 OmniVision 公司开发的一款 CMOS VGA（656*488）有色图像传感器。此传感器电压低、体积小的特点，提供了视频处理器和图像识别和处理的基本功能。是 高 性能 VGA CMOS 图像传感器 。 OV7740 的输入输出电压大约是 1.7 到 3.4V，电源的电流最大是 48mA 平均是 20 微安，OV7740 已经能够工作在高达

每秒 60 帧图像阵列在 VGA 分辨率与图像质量还有输出的格式还有数据传输。所有需要的图像处理功能，包括白平衡、曝光控制、缺陷像素消除、降噪、γ、色调控制、色彩饱和度，等等，都可以通过 SCCB 接口。OV7740　支持数字图像并行传输。图 3-1 显示　的是 ov7740　　图像传感器功能块图。

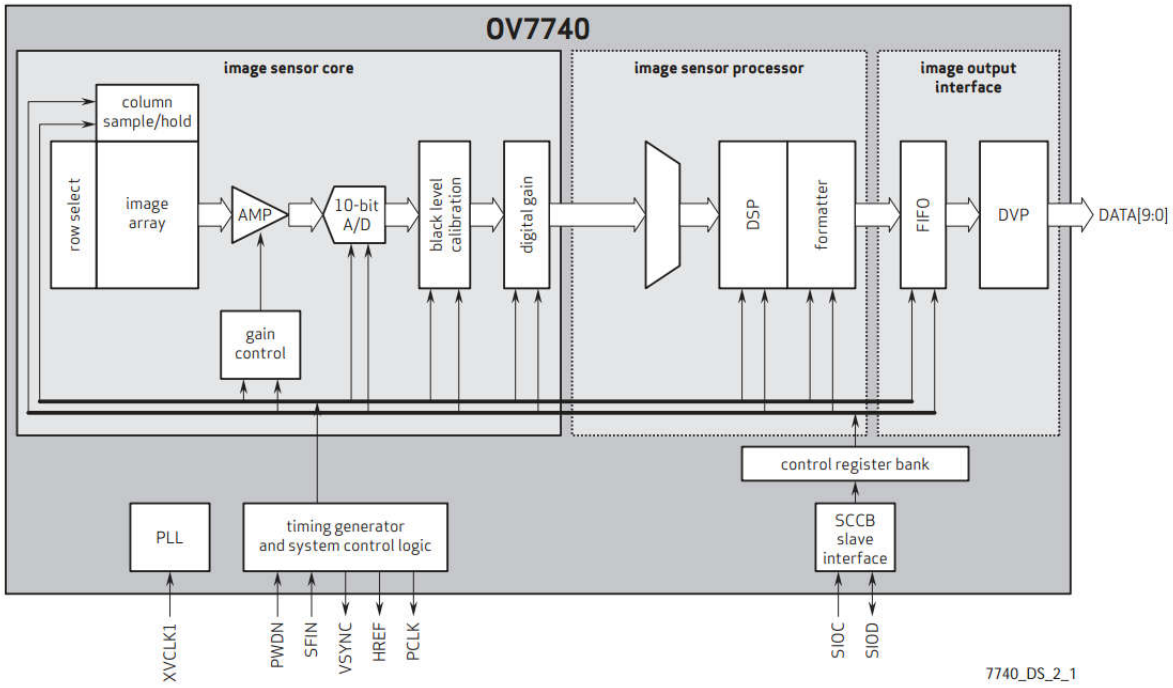


图 3-1 OV7740 图像传感器功能块图

本课题采用的 COMS OV7440 摄像头如图 3-2 所示，然后进行 COMS 转 USB 摄像头如图 3-3 所示：

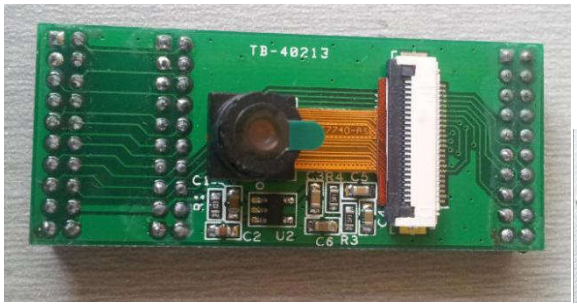


图 3-2 COMS OV7440 摄像头图

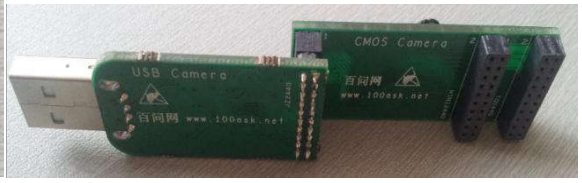


图 3-3 COMS 转 USB 摄像头图

OV7740 输出格式有：RAW RGB、YUV。

OV7740 支持的输出分辨率为：VGA(640*480)、QVGA(240*320)、CIF(352*288)、或者更小的任意大小分辨率。

3.2 OV7740 内部数据的处理流程

1. 图像传感核心部分：从上图中可以看到，此部分先从外部的景观进行图像采

集，传感器将图像感应到图像感光阵列，感光阵列只能感应到自然界的红绿蓝三种颜色，因此感光阵列将信号转换成红绿蓝的模拟信号，然后通过运算放大器将模拟信号进行放大，放大后进行图像信号的转换，将模拟转换为数字。再通过黑电平调整，最后输出一个数字信号。然后这个数字信号就是图像传感核心部分输出的信号，也就是 RAW RGB 格式。

2. 图像传感器处理部分：此部分的输入数据由图像传感核心部分输出的数据得来，而此部分的核心就是 DSP 模块，DSP 处于整个硬件模块上的数字信息处理功能，DSP 将收到的数据处理，传给最后的部分图像输出接口单元

此部分提供了很多功能有：提供测试功能（如果设置此功能，会将数据屏蔽，提供调色带数据如图 3-4）、镜头补偿功能、自动白平衡功能、把 RAW RGB 格式转换为 RGB 格式功能、把 RGB 装换为 YUV 格式的功能、窗口功能(将原始图片进行裁剪，只保留需要适合图像的数据)、缩小放大功能。

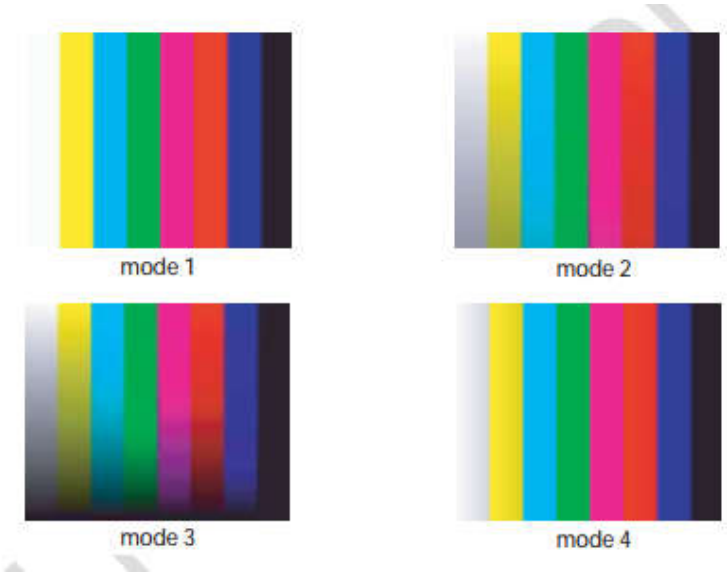


图 3-4 调色带数据图

3. 图像输出接口单元：此部分用来选择让 OV7740 输出的数据格式，是 RAW RGB 格式还是 YUV 格式，还有输出的分辨率是 VGA 还是 QVGA，还有传输的方式是 BT601 还是 BT656。

总结：以上这些处理过程，只有极少部分是自动完成的，而剩余部分是需要我们设置后，才能完成。并且是通过 IIC 总线，操作 OV7740 的寄存器来进行设置的。

3.3 OV7740 原理图（接口）

JZ2440 的摄像头接口模块的硬件原理图，如图 3-5 所示：

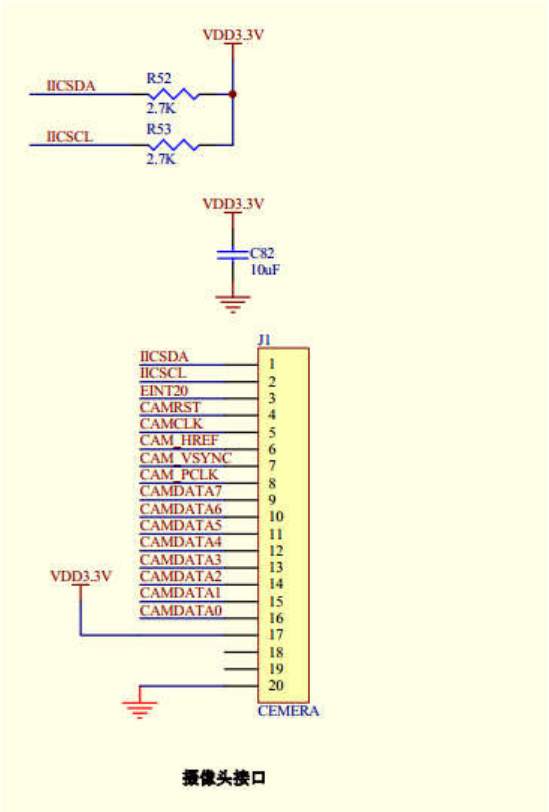


图 3-5 JZ2440 的接口图

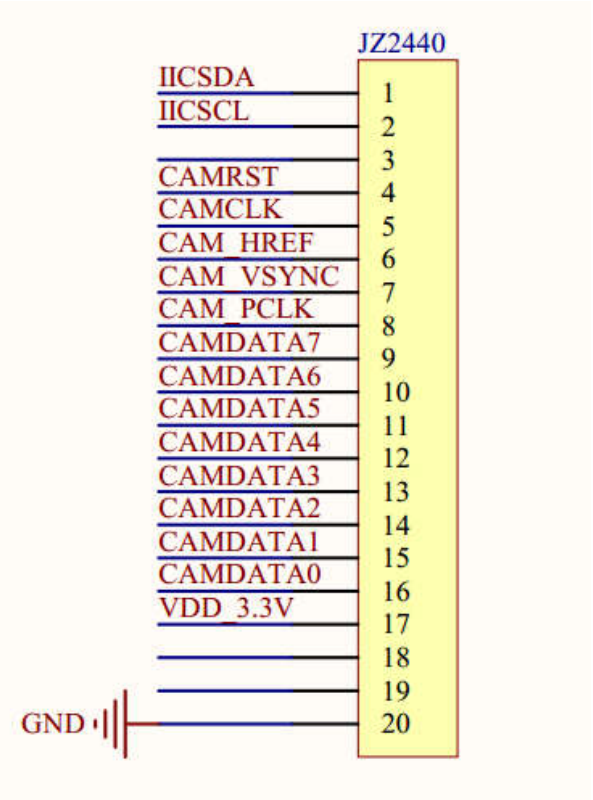


图 3-6 OV7740 的接口图

OV7740 的接口与 JZ2440 进行连接，如图 3-6 所示。

1.根据上图可知，CMOS OV7740 的接口与 JZ2440 接口刚好对应。

控制类 IIC 总线： IICSDA：IIC 总线的数据线、IICSCL --> IIC 总线的时钟线。

数据传输类 IIC 总线：CAMRST： 复位 CMOS 摄像头模块

CAMCLK ： 摄像头模块工作的系统时钟(24MHz)

CAM_HREF： 行同步信号

CAM_VSYNC：帧同步信号

CAM_PCLK： 像素时钟

CAMDATA0~7 数据线；

控制类 IIC 总线有： IICSDA ：数据总线；

ICSCL：数据总线中的。但是属于时钟线；

从上图可以得知，OV7740 的 CMOS 摄像头模块的接口大致分两类：(1).控制类：设置亮度还有旋转还有缩放等操作，对摄像头模块进行初始化，让摄像头模块能够正常的输出图像数据。(2).数据传输类:将摄像头所得到的数据进行传输给 Linux V4L2 应用层，让应用层进行数据的处理，然后传输。

OV7740 的图像数据输出（通过 SDA[]）就是在 SCL，CAM_VSYNC 和 SDA[IN]的控

制下进行的。首先看看行输出时序，可见下图 3-7：

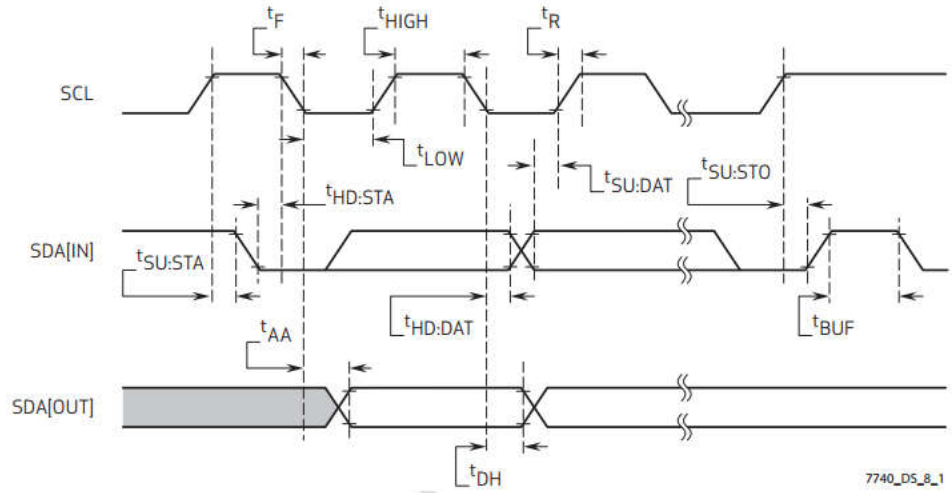


图 3-7 OV7740 数据的时序图

帧时序（VGA 模式），如图 3-8 所示：

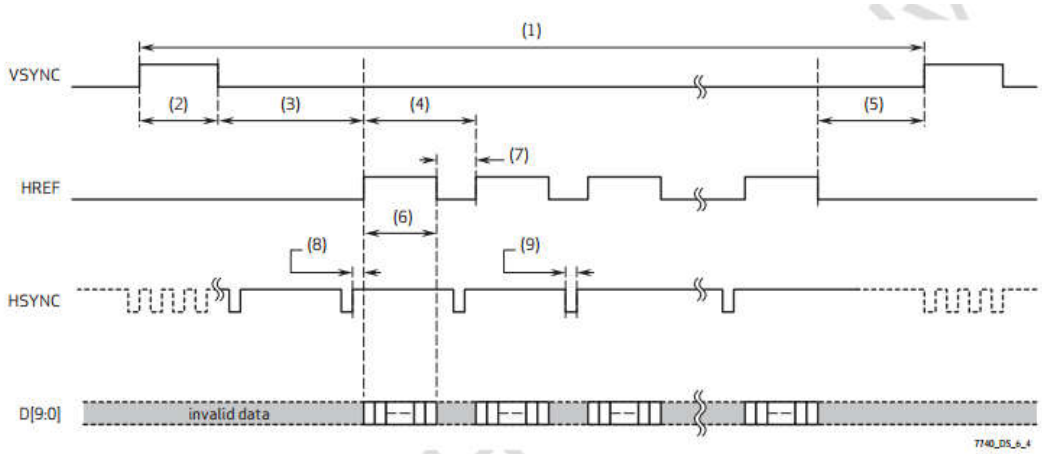


图 3-8 OV7740 帧时序图

mode	timing
VGA 640x480	(1) 399168 = 504 lines (2) 3168 (4 lines) (3) 9674 (4) 792 (5) 6318 (6) 640 (7) 152 (8) 106 (9) 48
QVGA 320x240	(1) 199584 = 252 lines (2) 1584 (2 lines) (3) 6882 (4) 792 (5) 1508 (6) 320 (7) 472 (8) 200 (9) 48

图 3-9 OV7740 帧时序时间图

如图 3-8 和图 3-9 所示，当 HREF 为高电平的时候图像数据才会输出。当 HREF 为高电平的时候，并且在硬件程序运行中行同步信号也为高电平，并且硬件程序运行过程中场同步信号为低电平，然后图像数据才能输出。每一个 PCLK 时钟，输出一个 10 位数据。采用 8 位接口，所以每个 PCLK 输出 1 个字节，且在 RGB/YUV 输出格式下，每个 tp=2 个 Tpclk，如果是 Raw 格式，则一个 tp=1 个 Tpclk。当 HREF 为低电平，在硬件程序运行中行同步信号也为低电平，图像数据停止输出。无论是 HREF 还是行 HSYNC 同步信号，只要有一个为低电平，就停止输出图像数据。经过 640 行同步后，一帧图像数据已经采集完成，这时候场同步信号会高电平，并且将整帧数据传给驱动，并且摄像头停止采集图像。

对于 JZ2440 的摄像头接口的 BT601 的同步信号如图 3-10 所示：

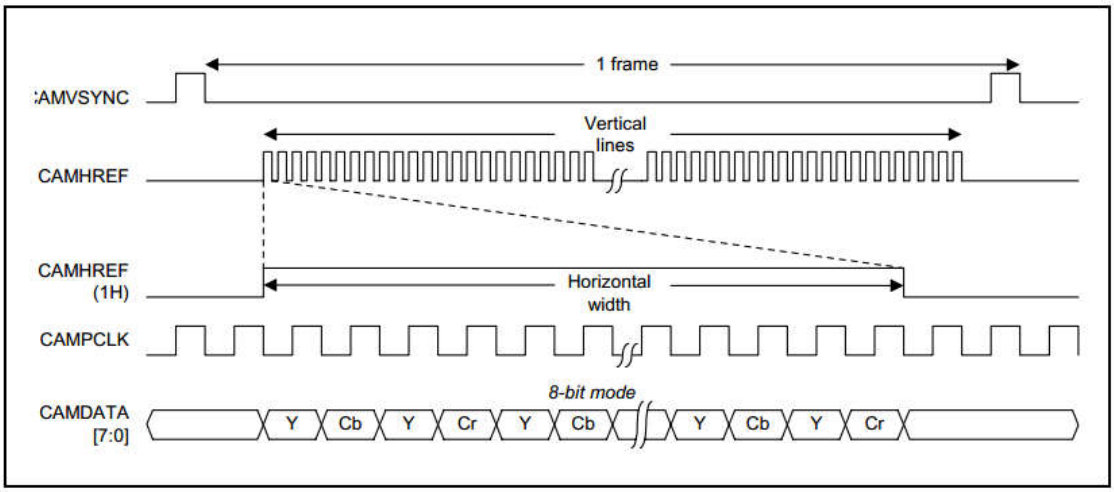


Figure 23-2. ITU-R BT 601 Input Timing Diagram

图 3-10 JZ2440 的 BT601 同步信号图

而 BT656 的同步信号如图 3-11 所示：

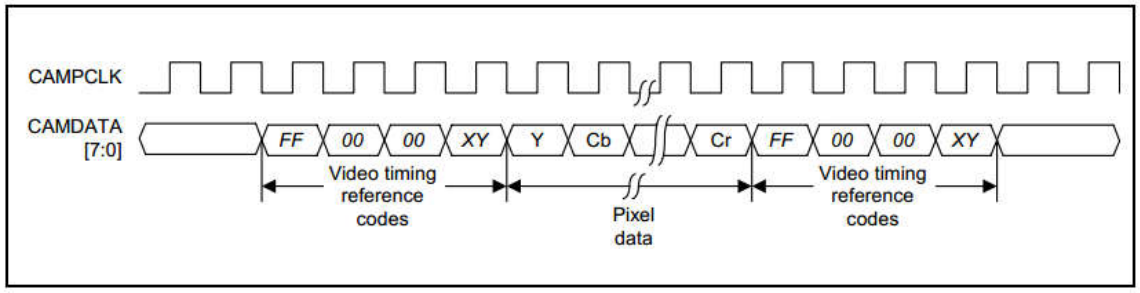


Figure 23-3. ITU-R BT 656 Input Timing Diagram

图 3-11 JZ2440 的 BT656 同步信号图

所以，再此课题中需要将摄像头设置为 640*480 分辨率，设置 30fps，并且图像数据格式设置为 YUV，然后 S3C2440 设置 BT601 模式进行数据传输。

4 OV7740 摄像头驱动开发

基于 Linux 的 OV7740 驱动开发有两种方式，一种是 CMOS 摄像头驱动开发和 USB 摄像头驱动开发，但在本课题中本人使用的是 USB 摄像头驱动开发。

4.1 Linux V4L2 驱动架构解析

linux 中可以采用灵活的多层次的驱动架构来对接口进行统一与抽象，最低层次的驱动总是直接面向硬件的，而最高层次的驱动在 linux 中被划分为“面向字符设备、面向块设备、面向网络接口”三大类来进行处理，前两类驱动在文件系统中形成类似文件的“虚拟文件”，又称为“节点 node”，这些节点拥有不同的名称代表不同的设备，在目录/dev 下进行统一管理，系统调用函数如 open、close、read 等也与普通文件的操作有相似之处，这种接口的一致性是由 VFS（虚拟文件系统层）抽象完成的。面向网络接口的设备仍然在 UNIX/Linux 系统中被分配代表设备的名称（如 eth0），但是没有映射入文件系统中，其驱动的调用方式也与文件系统的调用 open、read 等不同。

video4linux2(V4L2)是 Linux 内核中关于视频设备的中间驱动层，向上为 Linux 应用程序访问视频设备提供了通用接口，向下为 linux 中设备驱动程序开发提供了统一的 V4L2 框架。在 Linux 系统中，V4L2 驱动的视频设备（如摄像头、图像采集卡）节点路径通常为/dev 中的 videoX，V4L2 驱动对用户空间提供“字符设备”的形式，主设备号为 81，对于视频设备，其次设备号为 0-63。除此之外，次设备号为 64-127 的 Radio 设备，次设备号为 192-223 的是 Teletext 设备,次设备号为 224-255 的是 VBI 设备。由 V4L2 驱动的 Video 设备在用户空间通过各种 ioctl 调用进行控制，并且可以使用 mmap 进行内存映射。

从视频输入输入硬件的整个驱动链被抽象成 3 个层次：

1、最底层是直接面向硬件的，驱动框架由 v4l2 提供。值得注意的是，往往该层驱动需要总线驱动的支持，比如常见的 USB2.0 总线。

2、中间层便是 v4l2。这是 v4l 的第二版，由 Bill Dirks 最开始开发，最终被收入标准内核驱动树。

3、上层是 linux 内核三大驱动模块之一的“字符设备驱动层”，因此最终视频设备以文件系统中/dev 目录下的字符设备的面目出现，并被应用程序使用。

V4l2 的架构图如图 4-1 所示：

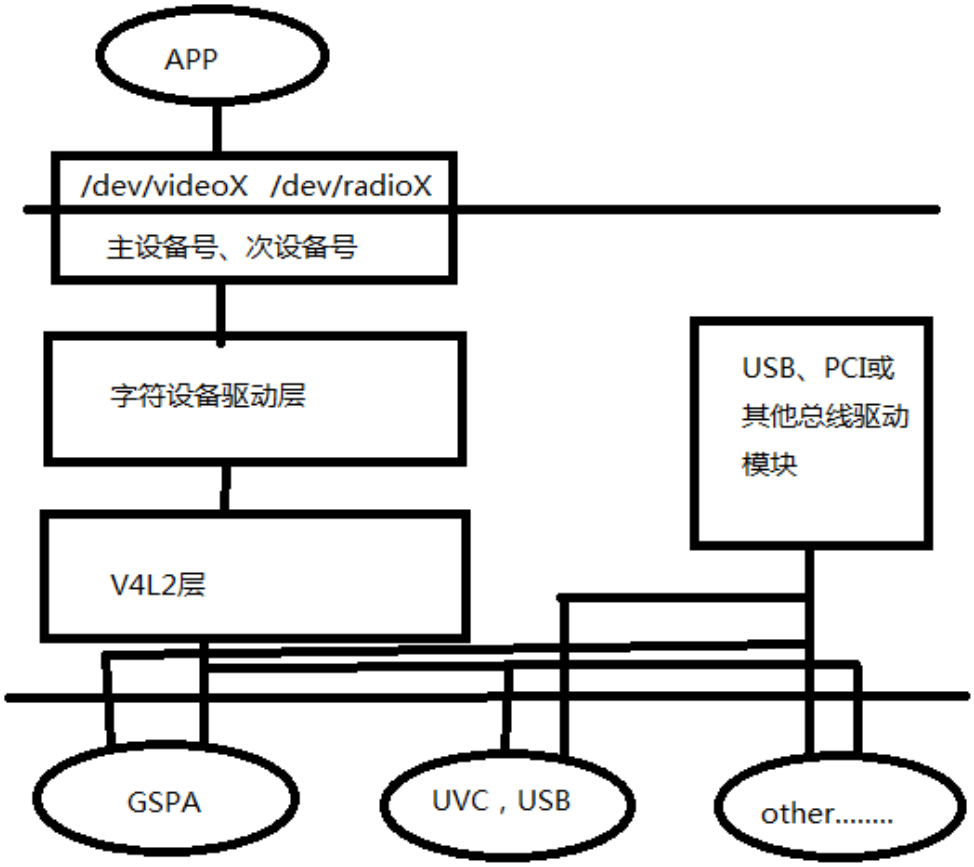


图 4-1 V4l2 在 linux 中的驱动架构图

V412 可以支持多种设备,它可以有以下几种接口:

1. 视频采集接口:这种应用的设备可以是高频头或者摄像头.V4L2 的最初设计就是应用于这种功能的.

2. 视频输出接口:可以驱动计算机的外围视频图像设备--像可以输出电视信号格式的设备.

3. 直接传输视频接口:它的主要工作是把从视频采集设备采集过来的信号直接输出到输出设备之上,而不用经过系统的 CPU.

4. 视频间隔消隐信号接口:它可以使应用可以访问传输消隐期的视频信号.

5. 收音机接口:可用来处理从 AM 或 FM 高频头设备接收来的音频流.

4.2 Linux V4L2 驱动编写流程

对于 Linux V4L2 驱动编写的大致流程建立在上流程之上，参考视频驱动 vivi.c 的程序。主要的流程图如下所示：

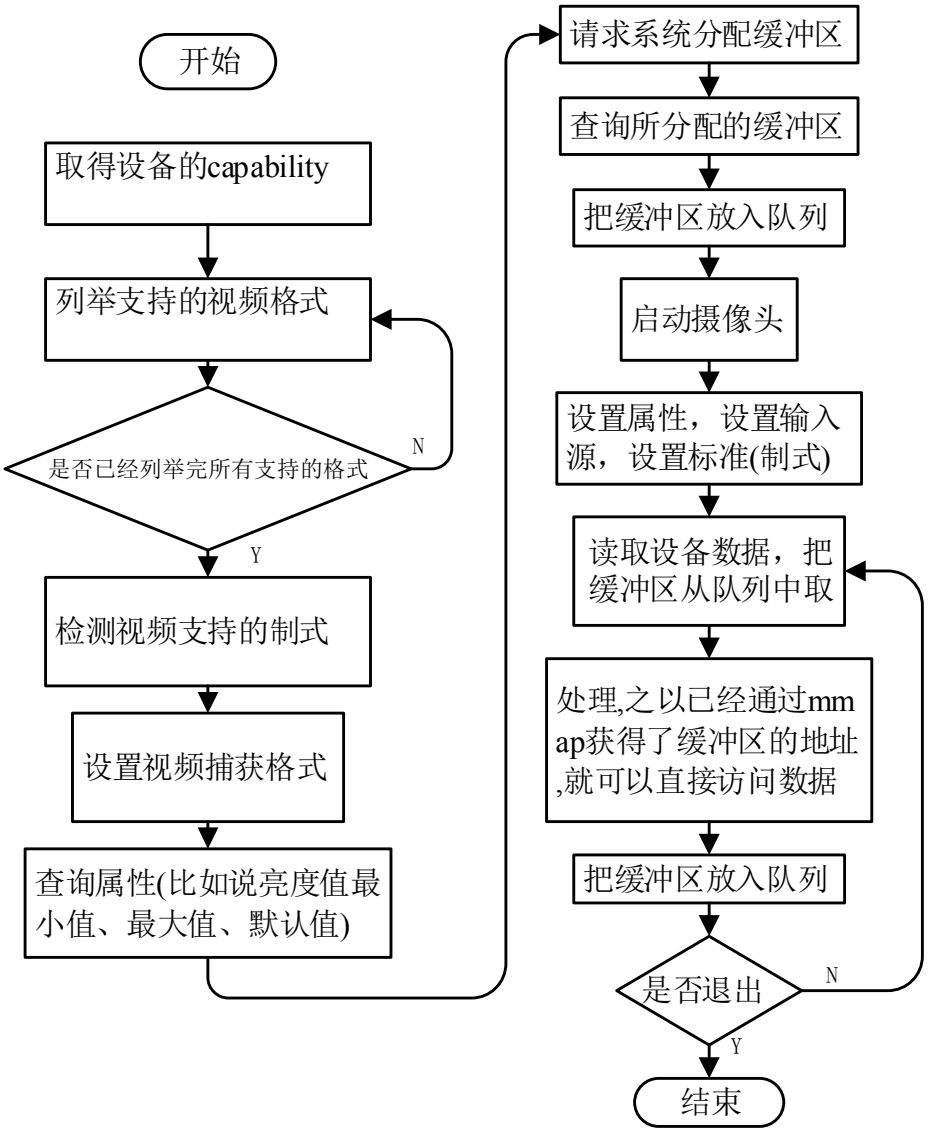


图 4-2 V4L2 驱动编写的大致流程图

从上图的视频驱动流程图可知，驱动程序需要打开设备节点/dev/videoX，然后进行一些硬件的操作。但是打开设备之前需要进行一些结构体的一些操作，结构体的操作大致流程是：

1 编写硬件接口函数

2 建立文件系统与设备驱动程序间的接口，如：struct file_operations 结构体，如：应用层所需要调用的函数指针.fops = &vivi_fops,

.ioctl_ops = &vivi_ioctl_ops,

.release = video_device_release,

3 注册设备到 chrdevfs 全局数组中，注册或注销设备可以在任何时候，但一般在模块加载时注册设备，在模块退出时注销设备。一般入口函数为上面步骤所定义的

file_operations 结构体中的函数指针（module_init（）；module_exit（）；）

4 以模块方式编译驱动源码，并将其加载到内核中

5 创建设备节点，mknode

6 编写应用程序访问底层设备图

如图 4-3 所示：

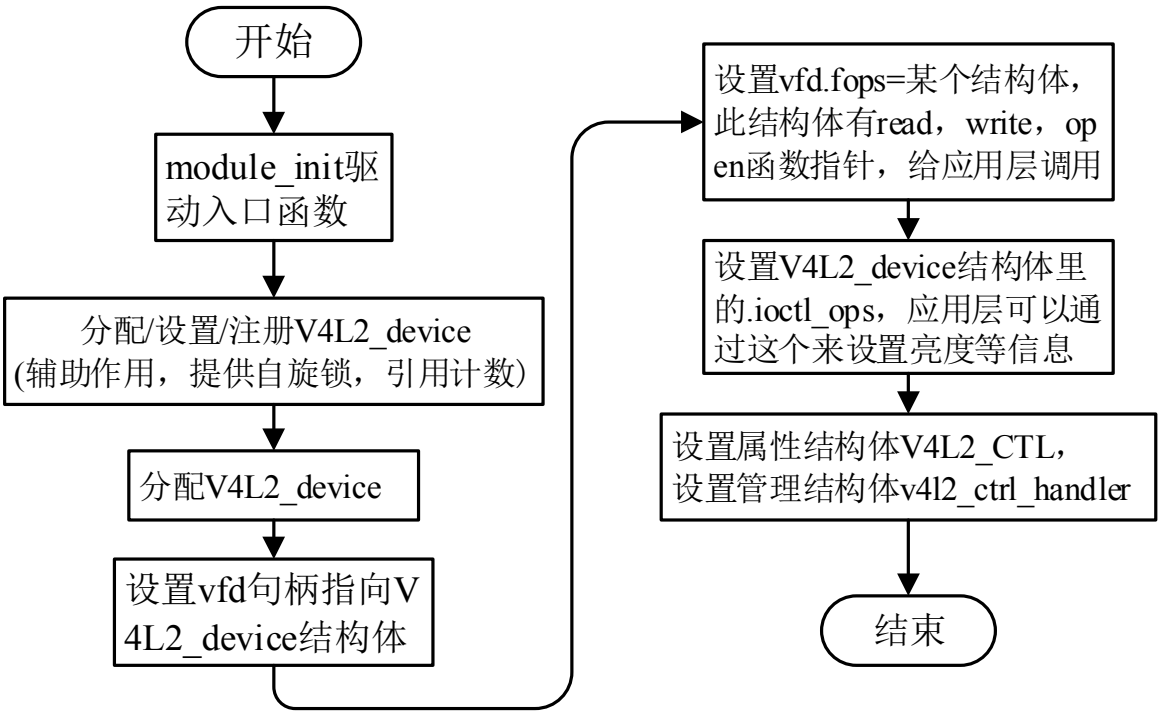


图 4-3　结构体的操作大致图

4.3 摄像头基于 USB 驱动编写过程

从上文可知，编写摄像头驱动的一些必要的步骤，在此课题中本人选取的是 USB 摄像头的方案。所以需要进行代码的整合，需要符合 Linux USB 驱动架构。有如下几个步骤：

1.构造一个 usb_driver

2.设置 probe:

2.1. 分配 video_device:video_device_alloc

2.2. 设置.fops

.ioctl_ops（里面需要设置 11 项）

如果要用内核提供的缓冲区操作函数，还需要构造一个 videobuf_queue_ops

2.3. 注册: video_register_device

id_table：表示支持哪些 USB 设备

3.注册： usb_register

当我们把设备接入电脑或者开发板时，id_table 可以显示可以支持 USB 设备，如果可支持，则 USB 驱动就运行.probe 所指的函数，并且.probe 所指的函数所执行代码的流程，就是上文所提到的视频驱动所运行的流程。

在市场上常见的摄像头所遵循 UVC 协议，这个协议是通过 USB 接口作为统一的，这个协议会遵循一些 USB Class 协议，如：海量存储设备协议，还有数据交换协议，视频设备协议，音频设备协议，打印机设备协议等等。而且还会通过硬件开发的规则测试。主要的目的是为了统一格式和视频数据的内容。所以有了这些协议的规定，才能使 USB 普及每个人，虽然 UVC 比较晚，但是随着时间的发展和科技的更新，UVC 也会普及每个设备。

所以由上面的介绍 UVC 可知，UVC 是基于 USB 开源的视频框架，社会的普及率也越来越高，本人在此课题中设计的方案，在方案里的驱动开发是参考 UVC 驱动开发框架的，对于一些硬件设备的操作参考数据手册。经过本人的刻苦专研，UVC 的大致框架结构已经熟悉。

如下图 4-4 所示是 UVC 拓扑结构图，UVC 的拓扑图是 UVC 原理的框架：

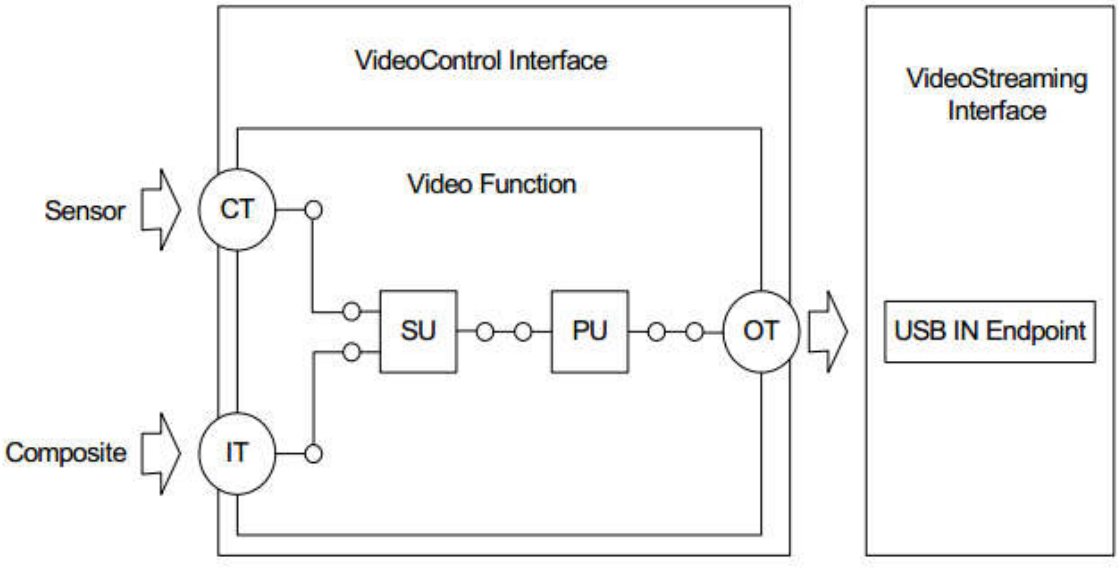


图 4-4： UVC 设备拓扑结构

如上图所示，UVC 将视频设备分为几个重要的硬件模块：

输入端点 Input Terminal

输出端点 Output Terminal

camera 端点 Camera Terminal

选择单元 Selector Unit

处理单元 Processing Unit

拓展单元 Extension Unit

如上图所示：UVC 的设备分为两大块，一个是用于控制模块，还有一个是用于数据传输模块，从摄像头传感器和另外一个复合视频设备从 Camera 端点和输入端点得到的数据流经过选择模块的筛选，然后通过处理单元的数据处理，在通过输出端点传给 USB 输入端，然后在传给 V4L2 进行数据的处理。

本课题中虽然 UVC 已经在 Linux 系统中自带的，但是本人通过 UVC 的思想自己进行重新的开发了类似 UVC 的程序，并且能正常使用

根据 Linux 所提供的开源 UVC 源码，可得出 UVC 代码中驱动调用过程：

1. 首先需要初始化，必要的结构体：uvc_fops

```
const struct v4l2_file_operations uvc_fops = {//驱动操作的必要的结构体
    .owner      = THIS_MODULE,
    .open       = uvc_v4l2_open,
    .release    = uvc_v4l2_release,
    .ioctl      = uvc_v4l2_ioctl,
    .read       = uvc_v4l2_read,
    .mmap       = uvc_v4l2_mmap,
    .poll       = uvc_v4l2_poll,
};
```

2. open:uvc_v4l2_open

3. VIDIOC_QUERYCAP   // video->streaming->type 应该是在设备被枚举时分析描述符时设置的

```
        if (video->streaming->type == V4L2_BUF_TYPE_VIDEO_CAPTURE)
            cap->capabilities = V4L2_CAP_VIDEO_CAPTURE
                    | V4L2_CAP_STREAMING;
        else
            cap->capabilities = V4L2_CAP_VIDEO_OUTPUT
                    | V4L2_CAP_STREAMING;
```

4. VIDIOC_ENUM_FMT // format 数组应是在设备被枚举时设置的

format = &video->streaming->format[fmt->index];

5. VIDIOC_G_FMT　uvc_v4l2_get_format　// USB 摄像头支持多种格式 fromat，每种格式下有多种 frame(比如分辨率)

struct uvc_format *format = video->streaming->cur_format;

struct uvc_frame *frame = video->streaming->cur_frame;

6. VIDIOC_TRY_FMT ：uvc_v4l2_try_format　/ /尝试设置某种格式

7. VIDIOC_S_FMT　// 只是把参数保存起来，还没有发给 USB 摄像头

uvc_v4l2_set_format

uvc_v4l2_try_format

video->streaming->cur_format = format;

video->streaming->cur_frame = frame;

8. VIDIOC_REQBUFS：uvc_alloc_buffers　// 分配缓冲区

9. VIDIOC_QUERYBUF ：将分配的缓存区放入队列

10. mmap ：uvc_v4l2_mmap　//映射缓存区

11. VIDIOC_QBUF：uvc_queue_buffer　//取出缓存区里的数据

12. VIDIOC_STREAMON ：uvc_video_enable(video, 1)　// 把所设置的参数发给硬件,然后启动摄像头

uvc_commit_video

uvc_set_video_ctrl　/* 设置格式 fromat, frame */

ret = __uvc_query_ctrl(video->dev /* 哪一个 USB 设备 */, SET_CUR, 0,

video->streaming->intfnum　/* 哪一个接口: VS */,

probe ? VS_PROBE_CONTROL : VS_COMMIT_CONTROL, data, size, uvc_timeout_param);

/* 启动：Initialize isochronous/bulk URBs and allocate transfer buffers. */

uvc_init_video(video, GFP_KERNEL);

uvc_init_video_isoc / uvc_init_video_bulk

urb->complete = uvc_video_complete; () //触发 poll，唤醒应用程序，进行数据传输，因为此函数最后会调用 wake_up 这个函数，进而唤醒

正在等待的数据传输程序

usb_submit_urb　　// 提交数据

13. poll：// 休眠等待有数据，若有数据进行唤醒线程，进行操作

14. VIDIOC_DQBUF： uvc_dequeue_buffer

list_del(&buf->stream);

15. VIDIOC_STREAMOFF　　//关闭数据流

uvc_video_enable(video, 0);

usb_kill_urb(urb);

usb_free_urb(urb);

由上分析可知：

1. UVC 设备有两个接口: VideoControl Interface, VideoStreaming Interface

2. 控制模块接口(VideoControl Interface)是来控制摄像头的一些功能，比如亮度，曝光，还有对比度。它的内部有很多的端点和单元之类的，用来控制设备，还有进行数据的处理。可以通过 uvc_query_ctrl 访问

3. 数据传输模块接口(VideoStreaming Interface)是来传输和采集视频数据的，可以用来选择视频数据的格式和帧频率，最主要的还是视频数据的传输功能。

4. 我们在设置 FORMAT 时只是使用了 video->streaming->format[fmt->index] 等数据，应是设备被枚举时设置的，也就是分析它的描述符时设置的。

5. 基于 USB 摄像头的驱动重点在于描述符，有如下几个：

属性的控制: 通过 VideoControl Interface 来设置

格式的选择：通过 VideoStreaming Interface 来设置

数据的获得：通过 VideoStreaming Interface 的 URB 来获得

对于 UVC 有属于它自己的描述符，如图 4-5 所示，其中，Device 等白色的描述符为标准描述符，灰色是特殊描述符。

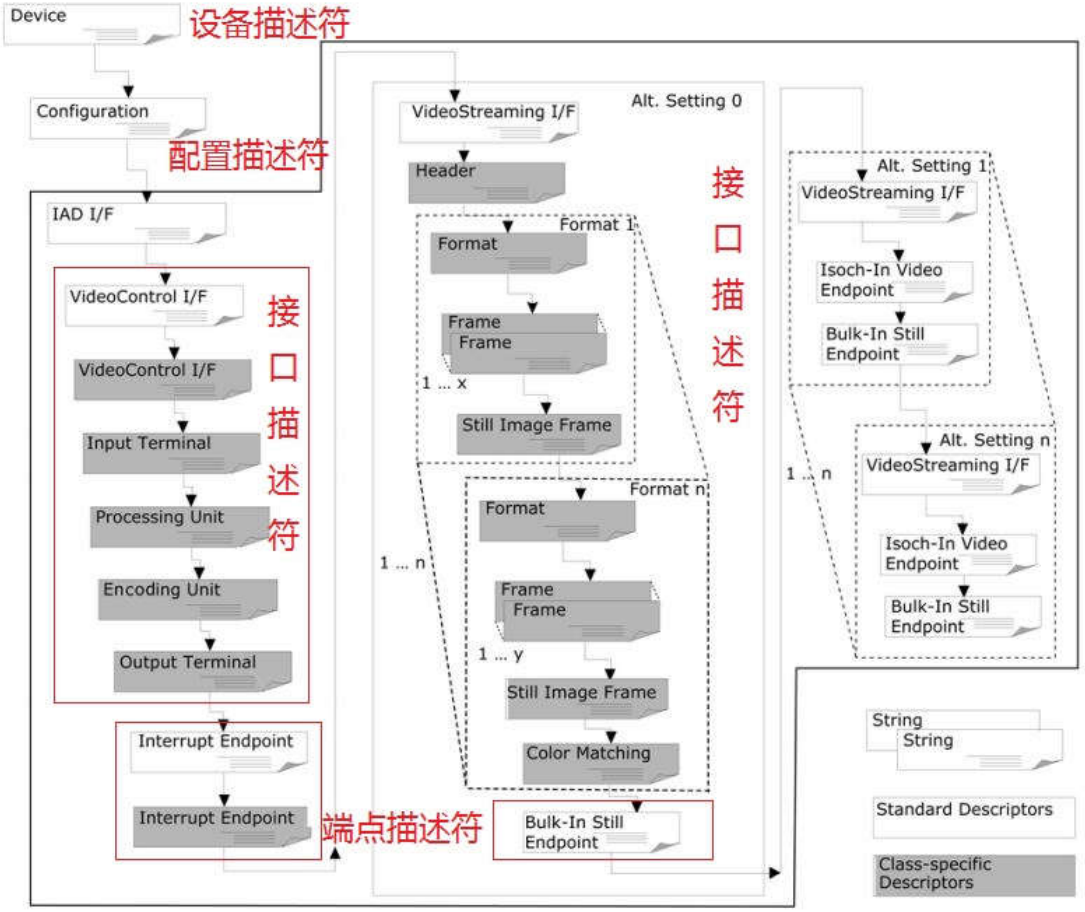


图 4-5　UVC 描述符图

4.4 S3C2440 的摄像头接口设置

S3C2440 提供了一个摄像接口，使开发人员很容易地实现摄像、照相等功能。摄像接口包括 8 位来自摄像头的输入数据信号，一个输出主时钟信号，三个来自摄像头的输入同步时钟信号和一个输出复位信号。摄像接口的主时钟信号由 USB PLL 产生，它的频率为 96MHz，再经过分频处理后输出给摄像头，摄像头再根据该时钟信号产生三个同步时钟信号（像素时钟、帧同步时钟和行同步时钟），反过来再输入回 s3c2440。

由于 OV7740 芯片有很多寄存器要求初始化操作，所以就需要另外的接口也就是串行摄像控制总线 SCCB，SCCB 和 IIC 的传输协议极为雷同。

1. 首先需要定义所需要操作的一些寄存器 ov7740_t ov7740_setting_30fps_VGA_640_480[]。

2. 映射一些寄存器

```
GPJCON = ioremap(0x560000d0, 4);
GPJDAT = ioremap(0x560000d4, 4);
```

```
GPJUP = ioremap(0x560000d8, 4);
CISRCFMT = ioremap(0x4F000000, 4);
CIWDOFST = ioremap(0x4F000004, 4);
CIGCTRL = ioremap(0x4F000008, 4);
CIPRCLRSA1 = ioremap(0x4F00006C, 4);
CIPRCLRSA2 = ioremap(0x4F000070, 4);
CIPRCLRSA3 = ioremap(0x4F000074, 4);
CIPRCLRSA4 = ioremap(0x4F000078, 4);
CIPRTRGFMT = ioremap(0x4F00007C, 4);
CIPRCTRL = ioremap(0x4F000080, 4);
CIPRSCPRERATIO = ioremap(0x4F000084, 4);
CIPRSCPREDST = ioremap(0x4F000088, 4);
CIPRSCCTRL = ioremap(0x4F00008C, 4);
CIPRTAREA = ioremap(0x4F000090, 4);
CIIMGCPT = ioremap(0x4F0000A0, 4);
SRCPND = ioremap(0X4A000000, 4);
INTPND = ioremap(0X4A000010, 4);
SUBSRCPND = ioremap(0X4A000018, 4);
```

3. 设置一些寄存器：

```
CISRCFMT:
  bit[31] -- 选择传输方式为 BT601 或者 BT656
  bit[30] -- 设置偏移值(0 = +0 (正常情况下) - for YCbCr)
  bit[29] -- 保留位,必须设置为 0
  bit[28:16]  -- 设置源图片的水平像素值(640)
  bit[15:14]  -- 设置源图片的颜色顺序(0x0c --> 0x2)
  bit[12:0]       -- 设置源图片的垂直像素值(480)
CIWDOFST:
  bit[31]     -- 1 = 使能窗口功能、0 = 不使用窗口功能
  bit[30、15:12]-- 清除溢出标志位
  bit[26:16]  -- 水平方向的裁剪的大小
```

bit[10:0]　　-- 垂直方向的裁剪的大小

CIGCTRL:

bit[31]　-- 软件复位 CAMIF 控制器

bit[30]　-- 用于复位外部摄像头模块

bit[29]　-- 保留位，必须设置为 1

bit[28:27] -- 用于选择信号源(00 = 输入源来自摄像头模块)

bit[26]　-- 设置像素时钟的极性(猜 0)

bit[25]　-- 设置 VSYNC 的极性(0)

bit[24]　-- 设置 HREF 的极性(0)

CIPRCTRL:

bit[23:19] -- 主突发长度(Main_burst)

bit[18:14] -- 剩余突发长度(Remained_burst)

bit[2]　-- 是否使能 LastIRQ 功能(不使能)

CIPRSCPRERATIO:

bit[31:28]: 预览缩放的变化系数(SHfactor_Pr)

bit[22:16]: 预览缩放的水平比(PreHorRatio_Pr)

bit[6:0]: 预览缩放的垂直比(PreVerRatio_Pr)

CIPRSCPREDST:

bit[27:16]: 预览缩放的目标宽度(PreDstWidth_Pr)

bit[11:0]: 预览缩放的目标高度(PreDstHeight_Pr)

CIPRSCCTRL:

bit[29:28]: 告诉摄像头控制器(图片是缩小、放大)(ScaleUpDown_Pr)

bit[24:16]: 预览主缩放的水平比(MainHorRatio_Pr)

bit[8:0]: 预览主缩放的垂直比(MainVerRatio_Pr)

bit[31]: 必须固定设置为 1

bit[30]: 设置图像输出格式是 RGB16、RGB24

bit[15]: 预览缩放开始

CIIMGCPT:

bit[31]: 用来使能摄像头控制器

bit[30]: 使能编码通道

bit[29]: 使能预览通道

4.分配缓存，映射缓存区：

5.设置摄像头的数据格式：

CIPRTRGFMT:

bit[28:16] -- 表示目标图片的水平像素大小(TargetHsize_Pr)

bit[15:14] -- 是否旋转，我们这个驱动就不选择了

bit[12:0] -- 表示目标图片的垂直像素大小(TargetVsize_Pr)

6.设置中断模式，如下图所示：

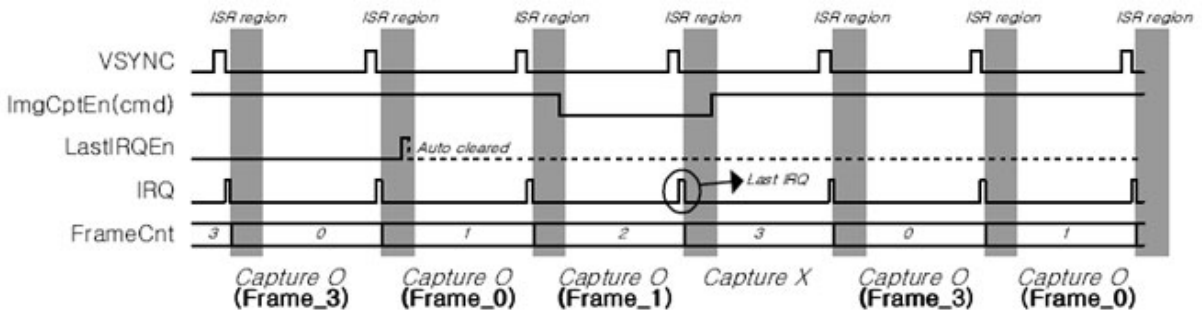


图 4-6： UVC 中断模式图

如上图可知，控制器会在每个 VSYNC 下降沿判断 ImgCptEn 信号等命令。如果在下降沿发现 ImgCptEn 信号有效,则产生 IRQ 中断。然后才开始一帧图像的真正采集。而如果在 VSYNC 下降沿判断到 ImgCptEn 为低电平且之前 LastIRQEn 没有使能，则不会产生任何中断，且不会再进行下一帧的采集。如果你想在 ImgCptEn 关闭后，一帧采集完后产生一个中断通知你，那么就需要在最后一次中断产生前（stop capturing 后的 vysnc 下将沿）使能 lastirq 就可以了：

```
*SRCPND = 1<<6;
*INTPND = 1<<6;
*SUBSRCPND = 1<<12;
ev_cam = 1;
wake_up_interruptible(&cam_wait_queue);
```

7.启动传输：将摄像头的数据进行传输给应用层

8. 结束传输，释放设备信息。

5 视频处理的应用层开发

基于上一章节驱动开发后，需要将驱动采取得到的数据进行获取到应用层，并做

一些图像处理，才可以发送出去。此章节参考了 Linux 的 V4L2 源码。

5.1 Linux 应用程序框架的介绍

由上文可知，当我们经过驱动，从摄像头读出的数据需要在应用层进行相应的处理，处理后，得到的数据能正常显示，如图 5-1 所示：

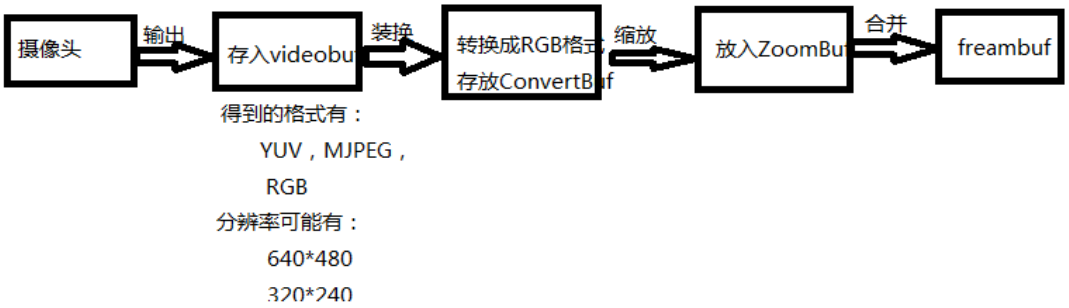


图 5-1　视频处理应用程序框架图

如上图可知，我们需要以面向对象和结构化的思想进行代码的编写。

5.2 Linux V4L2 简介

V4L2 的最早的版本是 V4L，由于设备的更新和系统的更新，V4L 已经不再适用于普通设备的图像采集了，所以 Linux 团队改进了 V4L 为 V4L2 更适合 Linux 2 版本的。在现实中，只要是基于 Linux 系统开发的摄像头设备或模块，都是基于 V4L2 开发的，比如一些参见的移动设备的视频操作，还有一些常见的视频会议设备等等。Linux 的内核分配在上一章已讲解，这里不多说什么。

由上一章节所知 V4L2 是 Linux 内核提供的图像视频数据处理传输程序，此功能是关于视频设备的中间内核驱动和应用层的中间层，负责对内核所收集到的数据进行处理。而且 V4L2 的驱动的节点一般放在/dev 目录下，并命名为 video0。V4L2 为字符设备模块。应用层通过调用 V4L2 接口，然后 V4L2 通过 ioctl 函数进行对具体的驱动进行操作。在此系统中，本人是自己实现了驱动程序还有 V4L2 程序。

5.3 Linux V4L2 流程分析

从上面的章节可以知道，摄像头和摄像头驱动的作用就是抓取视频图像作用，但是要让数据真正能显示出来，V4L2 取了很重要的地位，首先视频捕捉可以参考下面框架图 5-2：

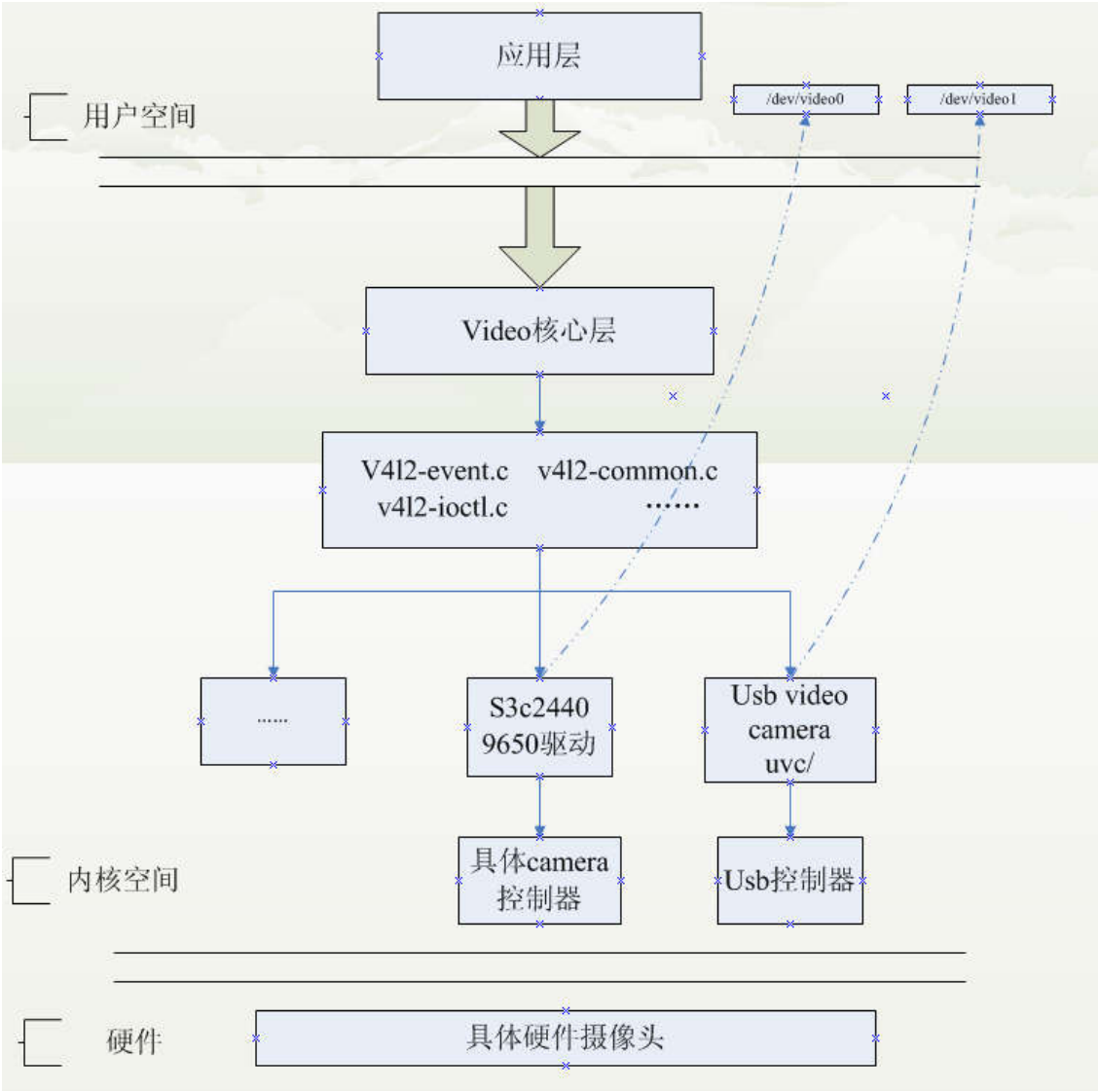


图 5-2　视频捕捉框架图

V2L4 有两个方法去抓取一些信息，一种是通过映射内存的形式，主要的作用还是分配用户空间，为了让应用层能访问内核的一些输出数据，所以需要映射。还有一种可以调用 ioctl 进行对摄像头 Read 视频数据。后者主要的对于照片的采集使用的。但是对于本课题是视频监控，所以使用的是内存映射的方式。

其实 V4L2 的流程思想很简单，只是把图像信息从内核搬运到用户空间。所以大致分为五大步骤：

1. 打开/dev/videoX 设备文件，并做一些设备初始化操作，比如窗口的设置，还有点阵的大小，还有视频传输格式的设置。

2. 然后申请缓冲区，让内核空间的视频数据缓存区映射到用户空间的缓存区。

3. 然后申请队列，并且将前面为空的缓存用 push 函数加入到队列队尾。

4. 然后驱动开始进行数据的采集，映射到用户空间，然后应用程序将数据读取

进行处理，然后一直循环处理

5. 停止视频数据的采集。

所以 V4L2 具体实现的流程图如图 5-3 所示：

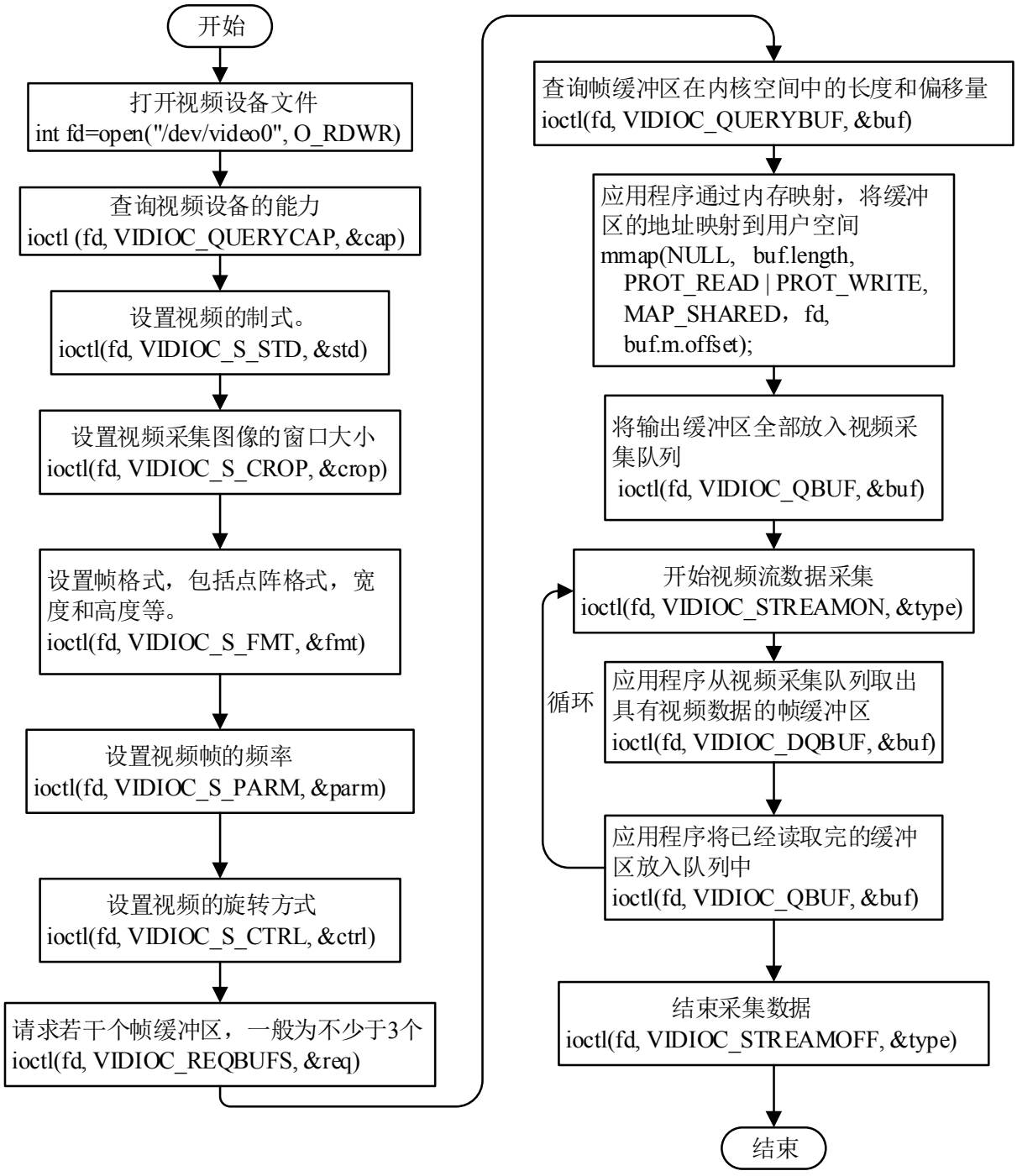


图 5-3　V4L2 具体实现的流程图

如上流程图所示，最开始就是做一些准备工作，进行对设备的初始化。其实上文分析 UVC 程序的过程中也说明，缓存区从队列里拿出来后，将数据读出，然后将缓存区清零，在放到队列里，以此循环，进行视频流数据的读取。所以上面所说的模式

如图 5-4 所示：

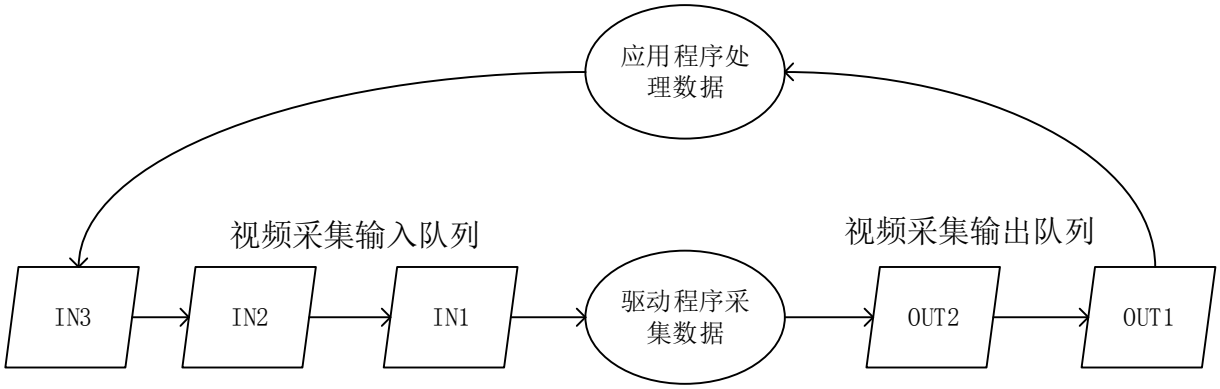


图 5-4　V4L2 循环采集模式图

5.4 视频格式的转换

由于市场常见的摄像头所获取原始图像数据格式有 RAW RGB，YUV。但是为了最后的调试，还有数据能传给移动端，需要进行数据的转换

YUV 与 RGB 格式转换：

YUV 是在市场上非常常见的摄像头获取图像的格式，主要是因为亮度数据，色彩数据分开处理并基于的是 RGB 模式，所以摄像头喜欢采取这种格式。

RGB 与 YUV 的转换算法：

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1678 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.1772 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

由上矩阵公式可得出以下几个公式：

Y = 0.299 R + 0.587 G + 0.114 B

U = - 0.1687 R - 0.3313 G + 0.5 B + 128

V = 0.5 R - 0.4187 G - 0.0813 B + 128

所以由上面的公司得出转换公式为:

R = Y + 1.402 (Cr-128)

G = Y - 0.34414 (Cb-128) - 0.71414 (Cr-128)

B = Y + 1.772 (Cb-128)

所以代码如下所示：

```
size = image_width * image_height /2;
 for (i = size; i > 0; i--) {
      Y = buff[0] ;
      U = buff[1] ;
      Y1 = buff[2];
      V = buff[3];
      buff += 4;
      r = R_FROMYV(Y,V);
      g = G_FROMYUV(Y,U,V); //b
      b = B_FROMYU(Y,U); //v
      r = r >> 3;
      g = g >> 2;
      b = b >> 3;
      color = (r << 11) | (g << 5) | b;
      *output_pt++ = color & 0xff;
      *output_pt++ = (color >> 8) & 0xff;
      r = R_FROMYV(Y1,V);
      g = G_FROMYUV(Y1,U,V); //b
      b = B_FROMYU(Y1,U); //v
      r = r >> 3;
      g = g >> 2;
      b = b >> 3;
      color = (r << 11) | (g << 5) | b;
      *output_pt++ = color & 0xff;
      *output_pt++ = (color >> 8) & 0xff;
```

MJPEG 与 RGB 格式转换：由于 MJPEG 转换 RGB 的算法有点复杂，所以本人是使用现成的 JPEG 库进行转换，并且代码的质量有点多

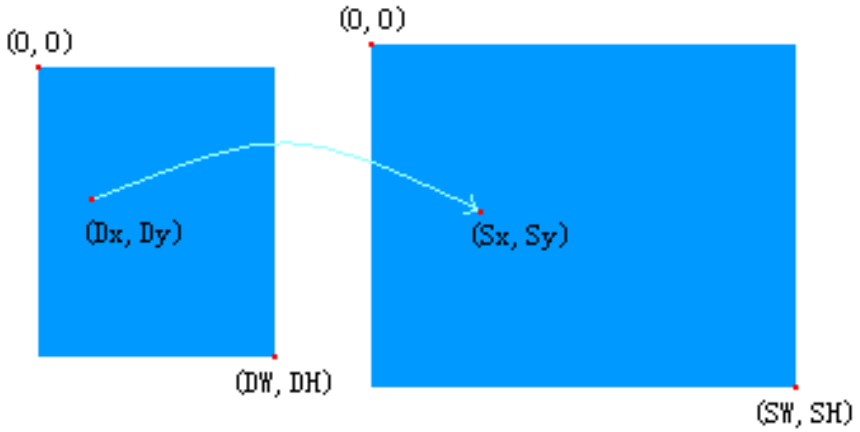
RGB 与 RGB 格式转换：由于 RGB 有很多格式，常见的有 RGB565 还有 RGB32 格式，在此项目中需要转换为 RGB32 格式，所以代码如下：

```
ptPixelDatasOut->iWidth  = ptPixelDatasIn->iWidth;
```

```
        ptPixelDatasOut->iHeight = ptPixelDatasIn->iHeight;
        ptPixelDatasOut->iBpp    = 32;
        ptPixelDatasOut->iLineBytes      =   ptPixelDatasOut->iWidth   *
ptPixelDatasOut->iBpp / 8;
        ptPixelDatasOut->iTotalBytes   =   ptPixelDatasOut->iLineBytes   *
ptPixelDatasOut->iHeight;
        if (!ptPixelDatasOut->aucPixelDatas)
        {
            ptPixelDatasOut->aucPixelDatas                                  =
malloc(ptPixelDatasOut->iTotalBytes);
        }
        pdwDest = (unsigned int *)ptPixelDatasOut->aucPixelDatas;
        for (y = 0; y < ptPixelDatasOut->iHeight; y++)
        {
            for (x = 0; x < ptPixelDatasOut->iWidth; x++)
            {
                color = *pwSrc++;
                r = color >> 11;
                g = (color >> 5) & (0x3f);
                b = color & 0x1f;
                color = ((r << 3) << 16) | ((g << 2) << 8) | (b << 3);
                *pdwDest = color;
                pdwDest++;
            }
        }
```

5.5 视频格式的缩放和合并

图像缩放算法：缩放原理和公式图示:



缩放后图片　　原图片
(宽 DW,高 DH)　　(宽 SW,高 SH)

(SX-0)/(SW-0)=(DX-0)/(DW-0)

(SX-0)/(SH-0)=(DY-0)/(DH-0)

=>

SX=DX*SW/DW

SY=DY*SH/DH

合并功能就是将小图片合并入大图片，所以合并的代码如下：

```
 * 函数名称：  PicMerge
 * 功能描述：  把小图片合并入大图片里
 * 输入参数：  iX,iY       - 小图片合并入大图片的某个区域, iX/iY 确定这个区域的左上角座标
 *             ptSmallPic - 内含小图片的象素数据
 *             ptBigPic   - 内含大图片的象素数据
 * 输出参数：  无
 * 返 回 值：  0 - 成功, 其他值 - 失败
int PicMerge(int iX, int iY, PT_PixelDatas ptSmallPic, PT_PixelDatas ptBigPic)
{
    int i;
    unsigned char *pucSrc;
    unsigned char *pucDst;
```

```
    if ((ptSmallPic->iWidth > ptBigPic->iWidth)  ||
        (ptSmallPic->iHeight > ptBigPic->iHeight) ||
        (ptSmallPic->iBpp != ptBigPic->iBpp))
    {
        return -1;
    }
    pucSrc = ptSmallPic->aucPixelDatas;
    pucDst = ptBigPic->aucPixelDatas + iY * ptBigPic->iLineBytes + iX *
ptBigPic->iBpp / 8;
    for (i = 0; i < ptSmallPic->iHeight; i++)
    {
        memcpy(pucDst, pucSrc, ptSmallPic->iLineBytes);
        pucSrc += ptSmallPic->iLineBytes;
        pucDst += ptBigPic->iLineBytes;
    }
    return 0;
}
```

6 视频 WIFI 传输模块

在这一章节中，本人搭建了一个小型的 Web 服务器采用 HTTP 协议将视频图像数据进行传输。

6.1 视频传输模块简介

21 世纪最重要的特征就是计算机网络发展的迅速，在这个时代是信息的时代，所以在此课题中本人选用 WIFI 开启了局域网进行数据接收和传输。之前计算机网络还没成熟的时候，各个公司提出不同网络数据传输协议，导致不同公司的计算机信息的交互成为了障碍。因此国际成立了组织叫因特网协会，统一了所有计算机相互通信的协议。在计算机网络中有 OSI 的七层协议，TCP/IP 四层协议，五层协议。其实这些协议都是相同的，只是在层面上的划分不同而已，所以本人在此课题中只简单的介绍 TCP/IP 四层协议，四层协议从上到下有：应用层如：HTTP 协议、FTP 协议、SMTP

协议，运输层如 TCP 协议、UDP 协议这一层是从事开发的最重要一层，网络层也叫 IP 层如 ICMP 协议、ARP 协议，数据链路层如 PPP 协议、还有以太网层，物理层。所以在此课题中，本人是基于 TCP/IP 开发的应用层协议传输的视频数据。

然而对于视频传输的网络协议而言，现成的的协议有很多分别为视频传输提供了基础，视频流传输协议有以下几种：

1. RTP/RTCP：RTP/RTCP 是实际传输数据的协议，RTP 和 RTCP 这两个的关联很大，RTP 传输音频/视频数据，如果是 PLAY，Server 发送到 Client 端，整个 RTP 协议由两个密切相关的部分组成：RTP 数据协议和 RTCP 控制协议（即 RTCP）。

2. RTSP：实时流协议，与上面的协议雷同，但是有一点是 RTSP 可以进过 RTP 协议知道 RTP 传输正在用的是哪个端口

3. HTTP 超文本传输协议，此协议众所周知，是万维网发展的基础。

本人选择的是 HTTP 协议，因为 HTTP 简便，所以适合此项目的设计。

经上分析可知如何设计视频传输的流程，流程图如图 6-1 所示：

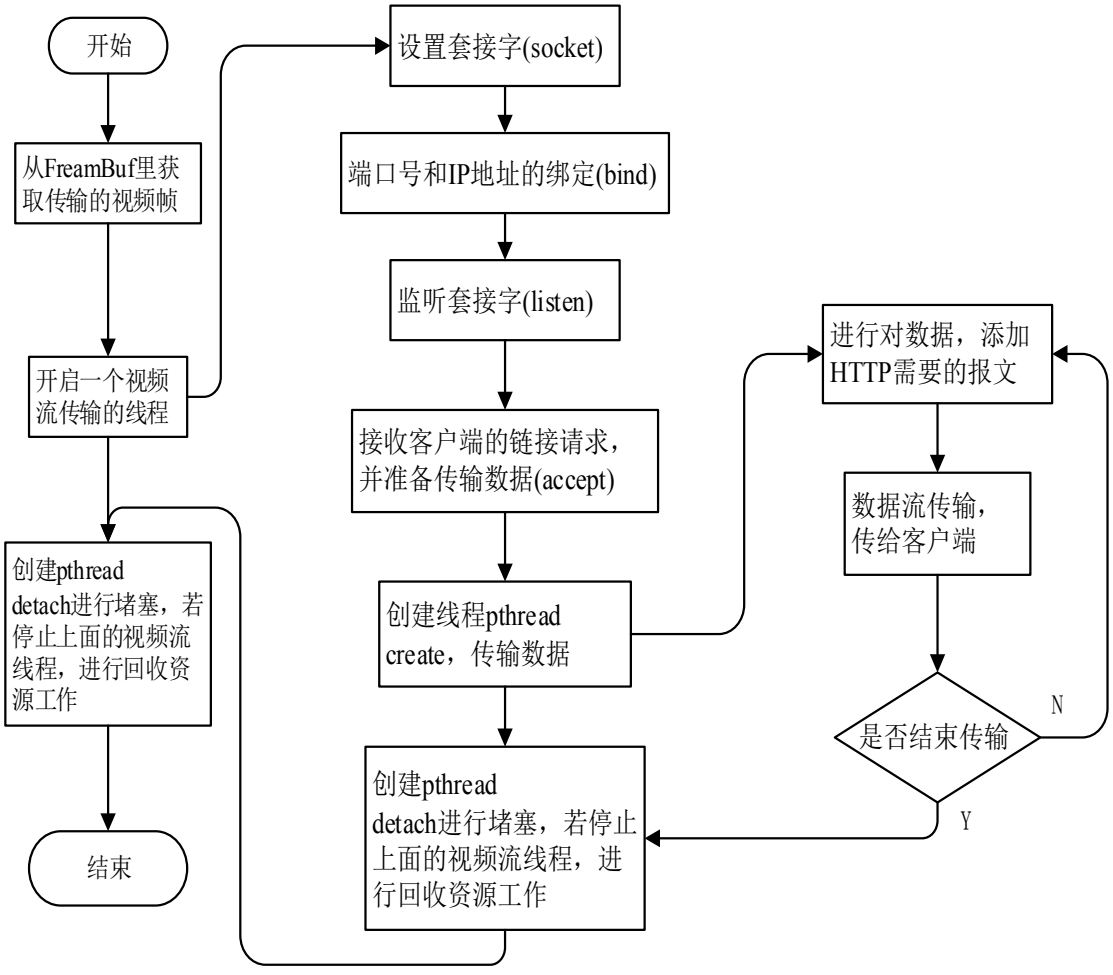


图 6-1　视频传输流程图

经上图可知，传输的模块分别创建了两类线程，第一个线程进行对套接字链接请求的响应，并且创建与管理第二类线程，第二类线程可以有很多个，共同点是进行对具体的某个客户端进行传输数据，并且被第一类线程管理，不同点只是传输的对象不同而已。

6.2 HTTP 协议概述

HTTP 是面向事物处理的协议也是传输文件的最佳选择之一，并且 HTTP 是基于传输层中的 TCP 协议，所以它使万维网上能够可靠传输地交换文件。

下面是 HTTP 协议的特点：

1.支持客户/服务器模式。

2.简单快速：当浏览器向主机请求有零一二进制转变成的特殊字段时，只要请求数据的路径名还有请求信号，在生活中很多经常用的有 GET，POST。不同的请求信号代表的与服务器的链接交换数据的规则不同。所以 HTTP 简洁有容易切换。

3.灵活：HTTP 头部报文信息有个字段是 Content-Type，这个是来描述任何类型的数据文件

4.无连接：HTTP 是面向无连接的，也就是数据传送完后，立刻断开链接，下次获取请求的时候，在重新链接。但是现在的 HTTP1.1 的发展已经有面向有连接的了，主要是为了解决会话保持的问题。

5.无状态：HTTP 是没有事物记忆的，也就是，与客户端连接后传递完数据后，立马断开，再此连接不会识别初刚刚连接时的信息。但是现在的 HTTP1.1 是借助 cookie 进行有状态的连接，比如现在的购物网站是需要记住客户端的状态的

HTTP URL 格式如下：

http://host[":"port][path]

http 是协议，host 一般是域名地址然后经过 DNS 域名解析系统，转换成 IP 地址进行访问服务器，HTTP 80 端口是默认的，但是在此课题中本人使用的 8080 端口，path 是要请求数据的所在服务器的绝对地址。

http 请求有三个大部分：请求行和消息报头，还有请求正文

1、本课题中使用的是 HTTP1.0。

请求方法有很多种，一下各个请求方法的说明：

GET　　申请读取服务端绝对路径下的信息。

POST　　给服务端添加一些信息

HEAD　　请求读取 URL 对应所标志信息的头部

PUT　　向 URL 端存储一个文件

DELETE　向 URL 端请求删除一个文件

TRACE　进行环回测试

CONNECT 保留，用于代理服务器使用

OPTIONS 请求一些选项信息，向服务端获取信息

HTTP 响应与请求相同有三个部分组成，分别是状态信息行和响应信息报头，还有要传的数据信息

1、状态行格式如下：

状态行都是在 HTTP 响应信息的第一行，主要是为了标准这个响应信息状态的作用。

状态行的状态信息有很多种，所以进行分类为 5 种：

1xx：表示请求已经收到，正在处理中

2xx：表示请求处理成功。

3xx：进行重定向操作，比如请求的文件地址引用的是别的地址，此时要转到别的地址继续请求

4xx：表示客户端提供的请求有误，不能完成此操作

5xx：表示服务端现在出现问题，或者服务端计算错误。

6.3 视频传给客户端数据的格式

由上文可知,此课题系统是通过搭建一个小型的Web服务器进行传输视频数据的，所以视频数据的传输就是基于 HTTP 报文传输的。

HTTP 协议中不管请求信号还是响应信号它们的报文以正文的中间一定是以一个空行来分隔的。

因为，每一帧所得到的数据就是每一张图片，所以我们使用了向客户端传送定时的数据，数据里带着一张图片一张图片的信息，构成了动画的效果，所以也构成了视频的效果。

所以每次所传输的数据的格式类似于下面的格式：

HTTP/1.0 200 OK

```
Content-Type: multipart/x-mixed-replace
boundary=" BOUNDARY
ETag: "3148797439"
Accept-Ranges: bytes
Last-Modified: Sun, 15 MAY2017 16:00:00 GMT
Expires: Mon, 06 Mar 2023 13:08:45 GMT
Cache-Control: max-age=311040000
Content-Length: 91
Date: Sat, 27 Apr 2013 13:08:45 GMT
Server: BWS/1.0

0000   ff d8 46 38 39 61 0f 00 11 00 91 00 00 ff ff ff  GIF89a.........
0010   db db db b6 b6 b6 00 00 00 2c 00 00 00 00 0f 00  .........,......
0020   11 00 00 02 34 84 8f a9 c2 7a 0c 61 58 31 a6 0a  ....4....z.aX1..
0030   b0 61 01 66 5d 09 9f 95 05 e6 89 36 19 70 1a a9  .a.f]......6.p..
0040   38 8a 9e e7 72 16 bd 4d 1d 57 22 fb 2e d8 51 42  8...r..M.W"...QB
0050   30 1f d1 b1 21 21 47 8a 02 00 3b 72 16 bd ff d9   0.. ...r.W.!!G...;
```

如上可知，代码结构如下：

```
/* 让 buffer = "" */
sprintf(buffer,
    "HTTP/1.0 200 OK\r\n" \
    STD_HEADER \
    "Content-Type: multipart "\r\n"
     boundary=" BOUNDARY "\r\n" \
    "\r\n" \
    "--" BOUNDARY "\r\n");
    /* 将 buffer 中的字符串发送出去(报文) */
    if ( write(fd, head_buffer, strlen(head_buffer)) < 0 )
    {
        free(frame);
```

```
        return;
    }
    while ( !pgal->stop )
    {
        /* wait for fresh frames */
        /* 等待输入通道发出数据更新的信号 */
        pthread_cond_wait(&pgal->db_update, &pgal->db);
        f_size = pgal->size; // 得到一帧图片的大小
        /* 检查我们之前分配的缓存是否够大,如果不够,则重新分配 */
        if ( f_size > max_frame_size )
        {
            max_f_size = f_size+TEN_K;
            if ( (tmp = realloc(f, max_frame_size)) == NULL )    // 重新分
配缓存
            {
                free(f);
                pthread_mutex_unlock( &pal->db );
                send_error(fd, 500, "not enough memory");
                return;
            }
            f = tmp;
        }
        /* 从 Mjpeg 的仓库中取出一帧数据 */
        memcpy(f, pal->buf, f_size);
        pthread_mutex_unlock( &pal->db );
        /* 让 Content-Length:为一帧图片的大小，给客户端发送图片的大小 */
        sprintf(buffer,
         "Content-Type: image/jpeg\r\n" \
        "Content-Length: %d\r\n" \
        "\r\n", f_size);
```

```
        if ( write(fd, buffer, strlen(buffer)) < 0 ) break;
        /* 将获得到的一帧图片发送给客户端 */
        if( write(fd, f, f_size) < 0 ) break;
        /* 让 buffer = "boundarydonotcross" */
        sprintf(buffer, "\r\n--" BOUNDARY "\r\n");
        if ( write(fd, buffer, strlen(buffer)) < 0 )
                break;
    }
```

7 Android 视频加载模块实现

客户端可以采用浏览器进行访问视频数据。由于本人觉得浏览器的功能有限，不能做到更好的地步，所以自己自行解析数据并开发 APP 实现客户端监控的功能。

7.1 Android 系统概述

Android 是一个开源的系统。现在 Android 已经成为全球最广泛使用在移动设备端的操作系统之一。

Android 最初并不属于谷歌公司的，最早是有安迪鲁滨开发的一款基于 Linux 运行在移动终端上的嵌入式操作系统，然而现在已被谷歌公司收购了，所以现在 Android 操作系统的隶属版权就是谷歌公司的。

自从被谷歌收购后，第一年发布的 Android1.0 版本的系统，但是这个版本在当时并没有得到广泛的支持。直到后来一年发布的 Android 2.0 后才得到认可。

Android 开源主要是因为 Linux 的开发者有个协议规则，就是只要是基于 Linux 开发的任何系统都必须开源，否则 Linux 团队就搞开发商侵权。

该平台有四个部分组成，分别为：应用层，应用框架层主要提供一些库的调用也是最主要的一层，下面就是 JNI 接口层，再下面就是 Linux 内核层。这个方式采用的是软件叠层的思想，就像像 Linux 的硬件驱动层还有内核层还有应用层，都是采用的是软件叠层的思想，这种思想充分是软件层与层之间的相互分离，是团队的分工合作更加简便，保证了软件系统低耦合的思想，如图 7-1 所示：

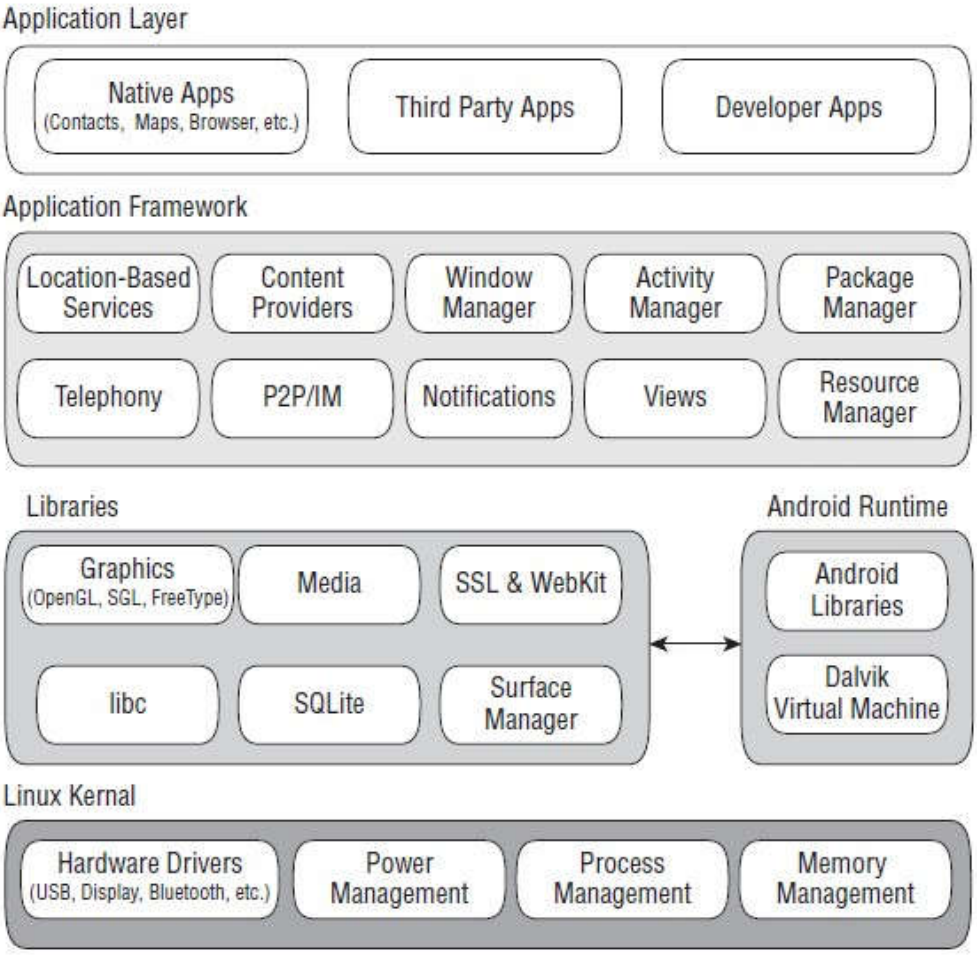


图 7-1　Android 系统架构图

Android 特色有：

1.四大组件：分别是 Activity，Service，ContentProvider，还有 BroadCast Receiver。Activity 就是我们每次打开一个 Android 应用时，直观的界面。Service 是伴随着这个应用，并在应用关闭后，能在系统一直运行的那种。假如我们从一个应用切换到另外一个应用，但是前面的应用的界面已经被销毁了，但是要让这个应用的核心程序不停，这时候 Service 就起了作用。由于不同的应用之间是不允许直接访问对方的存储数据的，但是在某些要求上，比如电话本的电话信息需要共享，ContentProvider 就处于这个作用。BroadCast Receiver 则允许你的应用接收来自各处的广播消息。

2.丰富的系统控件:提供了一些很常用的组件,常见控件有:TextView、Buttion、EditText、一些布局控件等。

3.SQLite 数据库等持久化技术：SQLite 是一种适合在嵌入式设备上运行的小型持久化存储的数据库。并且 Android 将 SQLite 的一些操作的的接口封装好，让开发

者更加方便安全的调用数据库 API。

4. GPS 定位：安卓操作系统还有 GPS 功能，让功能更加智能化。

5. 强大的多媒体技术：安卓系统还提供了很多像音频和视频的多媒体功能。

6. 传感器：安卓系统还有很多传感器的功能，当然 Android 提供传感器软件框架，真正的传感器硬件功能还有驱动需要移动开发商提供

7.2 视频监控客户端流程解析

由上面章节可知，我们的核心部分就是 SurfaceView 的使用，下面我们来分析和实现 Android 客户端的操作步骤：

1. 设置 MainActivity 的布局，为 RadioGroup 设置监听器，当 RadioButton 被按下或改变时触发下面的 onCheckedChanged 方法，检查 SD 卡，初始化 mjpegview 视图，这样就可看到监控画面了。

2. 如果有 SD 卡，则创建存放图片的 picturePath 目录。调用 com/mjpeg/view 的 mjpegView.java 类中 mjpegView 的众多方法来初始化自定义控件 com.mjpeg.view.MjpegView，MjpegView 类是重点。

3. 创建链接线程，此类的作用是在后台线程里执行 http 连接，连接卡住不会影响 UI 运行，适合于运行时间较长但又不能影响前台线程的情况，异步任务，有 3 参数和 4 步 :onPreExecute()， doInBackground()， onProgressUpdate()，onPostExecute()，onPreExecute()：运行于 UI 线程，一般为后台线程做准备，如在用户接口显示进度条，doInBackground():当 onPreExecute 执行后，马上被触发，执行花费较长时间的后台运算，将返回值传给 onPostExecute，onProgressUpdate():当用户调用 publishProgress()将被激发，执行的时间未定义，这个方法可以用任意形式显示进度，一般用于激活一个进度条或者在 UI 文本领域显示 logo，onPostExecute():当后台进程执行后在 UI 线程被激发,把后台执行的结果通知给 UI。

4. Android 提供两种 http 客户端，HttpURLConnection 和 Apache HTTP Client，它们都支持 HTTPS，能上传和下载文件配置超时时间，用于 IPV6 和 connection pooling，Apache HTTP client 在 Android2.2 或之前版本有较少 BUG 但在 Android2.2 或之后，HttpURLConnection 是更好的选择，在这里我们用的是 Apache HTTP Client。

5. 将上面所得到的 Http 后解析后的数据传给 mjpegView，新建一个目的图层和覆盖图层的相交模式，mjpegview 为目的图层，覆盖图层为右上角的动态"文本"，如

果线程是运行的，SurfaceView 也创建了的，则锁定画布 com/mjpeg/io/MjpegInputStream.java 中的 readMjpegFrame 方法获得 mjpeg 视频流的内容，mjpeg 视频的内容就是类位图，然后根据类位图绘制矩形，再绘制相应的位图，这个位图才是我们需要的，如果设置了帧率文本，就在 mjpegview 上覆盖，最后解锁画布，然后读取每一帧，进过 Canvas 和 paint，还有位图转换，然后进行显示在 SurfaceView 显示的画面上。

视频监控的传输是通过计算网络的 HTTP 协议实现的，并且是以二进制传输视频流数据。所以 Android 客户端接收数据的流程如图 7-2 所示：

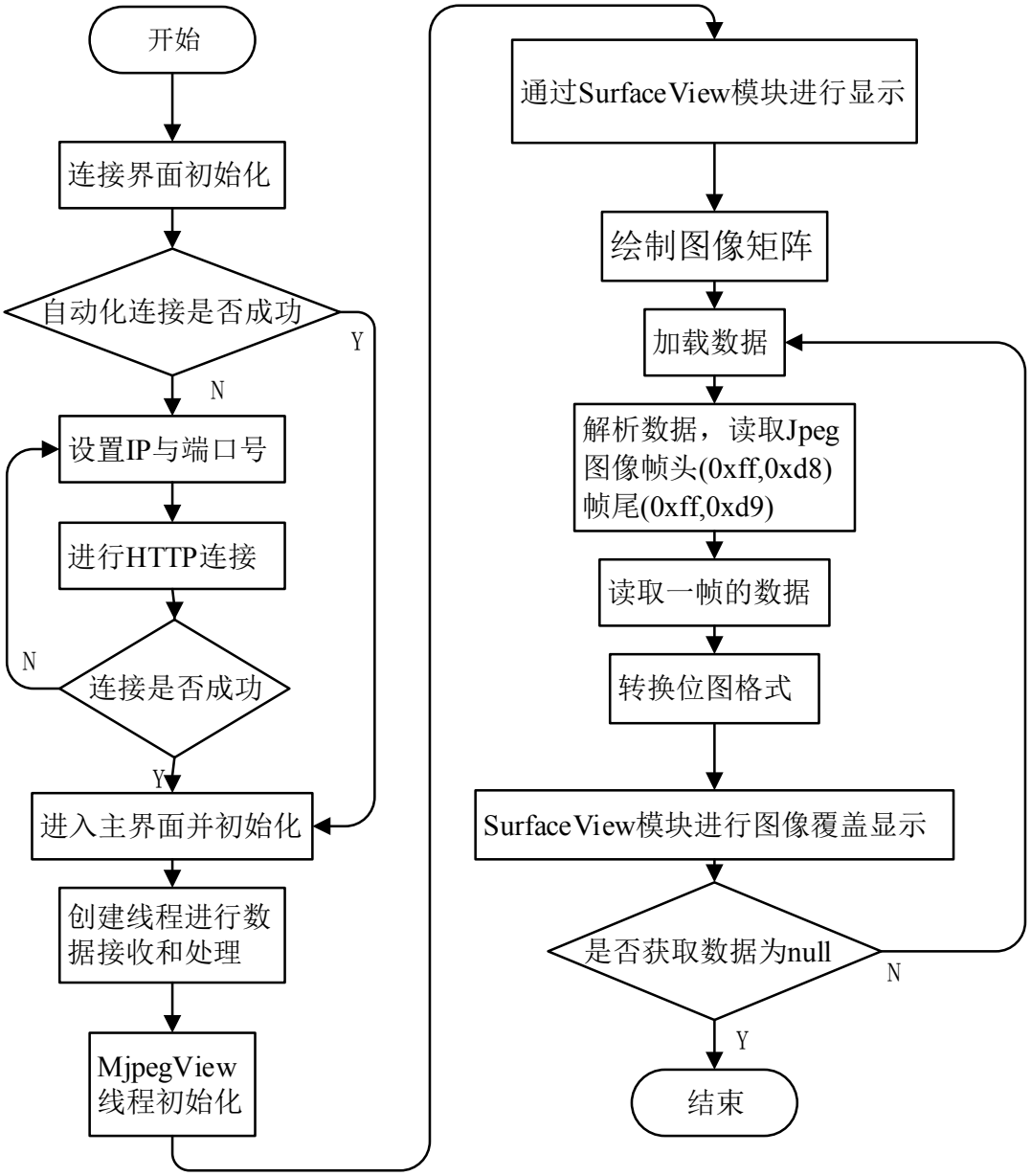


图 7-2　Android 客户端接收数据流程图

7.3 Android 中 Canvas 绘图的解析

Android 中的 Canvas 类就相当与一个画布,在这里面可以通过 Paint 和 Graphic 这个类进行对画布的画图操作。Canvas 这个类就像 Java SE 中的 Swing 这个类，在开发 Swing 中绘图的过程中需要继承 Jpanel 这个类，并重写 paint 这个方法就就可以了。Android 的 Canvas 类的开发也是一样的，需要继承 View 这个类，然后重写 onDraw 这个方法。

Canvas 绘制图像有两大方式实现，首先是直接继承 View 类，然后经过 Paint 画图，还有一种是基于表面视图(SurfaceView)来开发的。这两个的区别很大，前者就是需要等待 View 的更新，比如我们平时玩的手游五子棋，处理量小帧数小的这种。而 SurfaceView 的功能比前者强大，它不会堵塞 UI 线程，是因为有自己的加载图像的线程。所以此功能适合高性能游戏，还帧数高的动画。

Canvas 经常用到的一些函数调用有：

drawRect()：	剪切一个矩形区域
drawPath() ：	沿着指定的路径绘制任意的一个形状
drawBitmap()：	在指定的一个点从原味图截取一块，或者绘制位图
drawLine()	在位图中绘制一条直线
drawPoint()	在位图中绘制一个点
drawText()	在位图中绘制字符串
drawOval()	在指定的位置绘制一个椭圆
drawCircle()	在指定的位置绘制一个半径确定的圆
drawArc()	在某个端点前后绘制一条圆弧线

还要理解一个 paint 类：

Paint 类就相当于平时生活中的画笔还有颜料，给画布(Canvas)画图；

Paint 经常用到的一些函数调用有：

setARGB()	设置对象对应的颜色
setAlpha()	设置对象的透明度
setAntiAlias()	是否开启锯齿消磨
setColor()	设置对应对象的颜色信息
setTextScaleX()	设置字体的缩放

setTextSize()	设置字体大小
setUnderlineText()	设置是否开启下划线

7.4 Android 的视频加载模块解析

由上流程图可知，在此课题中移动客户端最核心的模块就是 SurfaceView 模块。所以以下讲解表面视图(SurfaceView)模块。Android 中的表面视图(SurfaceView)是专门给高效的绘图使用的，如上一节讲到 SurfaceView 有自己的加载图像的线程。在 Android 开发中有个特别著名的异常，就是 ANR 异常，发生的原因是我们的 UI 线程发生了堵塞，导致界面不能响应。这个异常用过 Android 手机的人都会体会到，UI 线程在绘图中是最容易发生异常的，如果是低频率的绘图，到不会出现，如果绘图的频率很大则就会出现 ANR 异常，所以 Android 提供了 SurfaceView 模块，创建一个线程进行绘图操作，而 UI 线程继续做它其他的响应业务。

我们平常开发 APP 的时候，Activity 就是 UI 主线程，要做的东西很多，比如一些基本控件的绘制，还有对控件发生的事物进行一个响应。也就是说 UI 线程不能做太多的耗时的操作，否则就会出现 ANR 异常，在这个课题中，视频流从服务端发送到客户端，然而客户端还要进行视频处理的操作，如识别帧，加载视频帧，还有键数据转换为位图形式的，所以客户端的视频获取就只能借助 SurfaceView 来实现，这样也不会发生 ANR 异常，当然也可以自己开发个线程进行跟 SurfaceView 相类似的操作。

如图 7-3 所示，假设我们在 UI 中创建几个组件，最上面那一层是装饰层，还有两个控件文字显示控件，还有一个是表面视图(SurfaceView)。

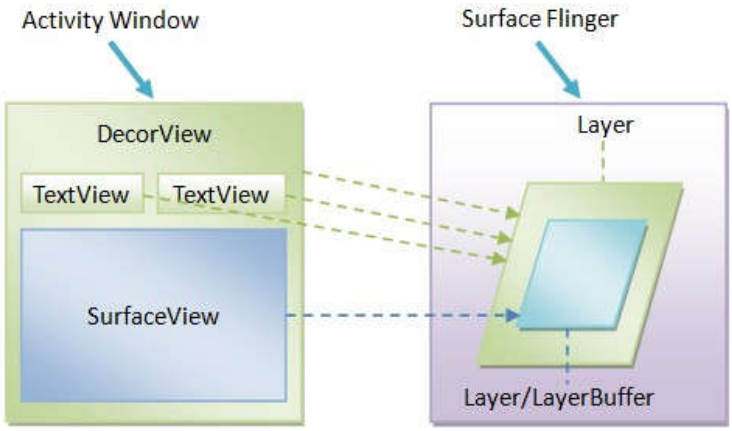


图 7-3　DecorView 顶层视图

在图 7-3 中，文字显示控件和装饰层是绘制在 UI 界面上的，而 SurfaceView 是绘制在 Layer 层上面的。

所以从上图来看，SurfaceView 并不是基于 UI 绘制的，而是类似于在 UI 上面挖取一个洞，然后在洞里面放在 Surface Flinger 和 Layer/LayerBuffer。所以这样就构成了另外一个线程来操作 SurfaceView 的图像动态变化

SurfaceView 是基于 Layer 层上绘制构成的，所以 SurfaceView 可以比 UI 主线程先绘图，但是 SurfaceView 和普通的空间都是在一个宿主上的，所以 SurfaceView 的创建还是要基于 UI 主线程触发 SurfaceView 绘制。

当窗口需要刷新 UI 界面的时候，就会调用 ViewRoot 中的方法 performTraversals，performTraversals 会先判断当前的要刷新的窗口是否已经创建，如果已经创建，则更新，否则调用 WindowManagerService 重新创建一个窗口，同时一会绘制对应的 SurfaceView 窗口。

SurfaceView 会调用 getHolder()方法获得相关联的 SurfaceHolder

SurfaceHolder 类会提供一下几个接口来获取 Canvas 画布对象。

Canvas lockCanvas(): 将表面视图加上一层锁，防止一些操作，来获取 Canvas 对象。

Canvas lockCanvas(Rect dirty)：与上面相同都是来锁表面视图，但是只是锁定 SurfaceView 里的某个区域。

当获取到对应的 Canvas 对象后，需要对锁定的区域解锁，让区域可绘图。

unlockCanvasAndPost(Canvas)：解锁 SurfaceView 已经锁定的区域。

7.5 Android 视频加载处理核心源代码

由上面流程图还有解析可知，表面视图相关操作的核心源代码如下：

```
public void run() {
            start = System.currentTimeMillis();
            Rect destRect;
            Paint p = new Paint();
            //       String fps = "";
            while (bRun) {
                if (bsurfaceIsCreate) {
                    c = mSurfaceHolder.lockCanvas();
                    try {
```

```
                    mjpegBitmap = mIs.readMjpegFrame();/* 调 用
Inputstrean 的方法*/
                    /*同步图像的宽高设置*/
                    synchronized (mSurfaceHolder) {
                        destRect = destRect(mjpegBitmap.getWidth(),
                                mjpegBitmap.getHeight());
                    }
                    if(c != null){
                        c.drawPaint(new Paint());
                        c.drawBitmap(mjpegBitmap, null, destRect, p);
                        if (bIsShowFps)
                            calculateFps(destRect, c, p);
                        mSurfaceHolder.unlockCanvasAndPost(c);
                    }
                } catch (IOException e) {
                }
            }else {
                try {
                    Thread.sleep(500);//线程休眠，让出调度
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
public Bitmap readMjpegFrame() throws IOException {
        mark(FRAME_MAX_LENGTH);/*流中当前的标记位置*/
        int headerLen = getStartOfSequence(this, SOI_MARKER);
        reset();/*将缓冲区的位置重置为标记位置*/
        byte[] header = new byte[headerLen];
        readFully(header);
        try {
```

```
            mContentLength = parseContentLength(header);// ?
        } catch (NumberFormatException e) {
            return null;
        }
        /**
         * 根据帧数据的大小创建字节数组
         */
        byte[] frameData = new byte[mContentLength];
        readFully(frameData);
        return                                  BitmapFactory.decodeStream(new
ByteArrayInputStream(frameData));
    }
```

8 系统调试

在代码编译成功之后，运行 JZ2440 开发板，进行网络的配置，开发板的网段为 192.168.7.X，虚拟机 Ubuntu 作为服务器，成为开发板的文件系统，然后经过网络文件系统的加载成功，配置无线 WIFI 的网段为 192.168.1.X，手机客户端链接开发 WIFI 适配器，然后插入摄像头，并加载驱动，然后运行程序，这时候打开手机 APP，进行连接开发板，进行接收数据。最后显示效果如下图所示：

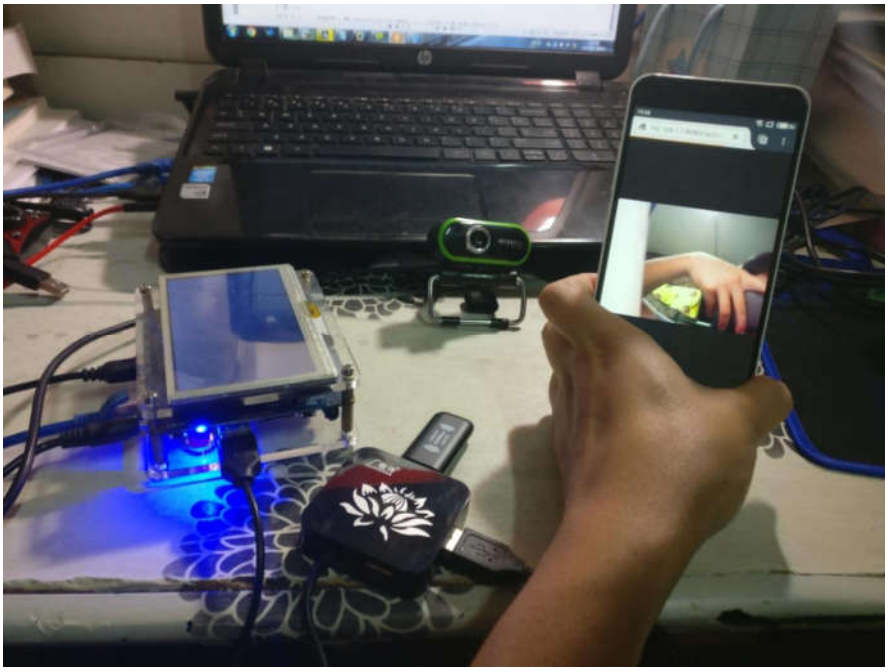


图 8-1　普通摄像头显示效果图

上面是普通摄像头的，所用的驱动是系统自带的，下图的效果图是 OV7740 自制摄像头设备：

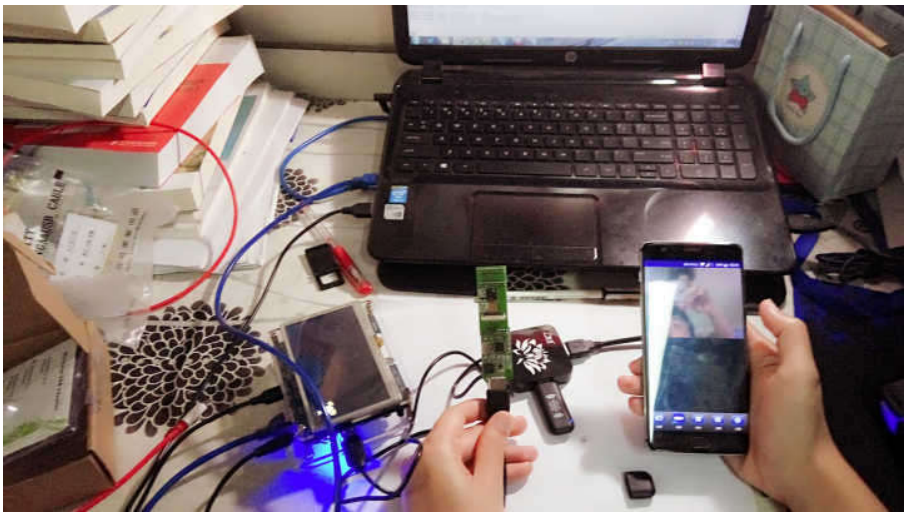


图 8-1　OV7740 自制摄像头显示效果图

结论

我珍贵的大学四年生活快要结束了，毕设是考验一个人在大学里掌握的知识的熟练度的考察，也是为将来的工作提供了实践基础，也同样是一份无比珍贵的礼物。经过这 4 个月来的研究，我不仅巩固了以前学习的专业知识，还学习到了很多自己以前没有学过的知识以及提高了自己实践动手能力，而且开阔了我的眼界和丰富了我的大脑，使自己更加清楚自己所学知识的行业的发展前景。

在做毕设的时候，我从最初的懵懂状态，开始查阅相关专业知识，并借鉴前辈们的经验，慢慢的对自己的毕业设计有了初步的认识和思路，直至最后自己能总体构思自己的整个毕业设计，以及设计出属于自己的硬件电路。这个过程对于我来说虽然很艰辛，但同样也很宝贵。在查阅资料期间，我才体会到自己在专业知识方面是多么的匮乏，如果自己真正的要完成一个作品，要学习的东西还有太多，在做毕设的过程中，我学会了如何快速去查阅资料，还有研究代码会出现的 BUG，比如当遇到驱动不能正常读取数据的时候，经过自己的不断分析，最后终于解决这些问题了。这个过程对我以后的影响是深刻的。

虽然自己最后能实现了这个系统，但是有些算法需要优化，需要更深入的研究，比如：算法优化，减少卡帧的现象，增加音频功能，还有将图像人工识别算法加入，是监控系统更加智能化。

致谢

经过做毕设的这 4 个月，自己的毕设也按预期完成，并达到老师最后的标准，虽然过程非常的艰辛，但是收获也蛮多。

首先，非常感谢我的导师赵伟志老师，在他细心的指导下，我也不可能会学到这么多，从硬件的原理和后面的 Linux 的开发，还有客户端的开发，还有驱动的开发，这些都对我未来的发展起了很好的开端作用。要是没有赵伟志老师的细心指导和无比关心，我的毕业设计和毕业论文不可能在一个比较快的时间内完成。这份毕业设计不仅代表着大学对我的肯定，更是导师对我即将出社会的一份礼物，将会对我终身受益。感谢赵伟志老师给予了我这样一个学习机会，谢谢赵伟志老师

同时，感谢所有热情帮助过我，也跟我相互讨论问题的同学。

参考文献

[1] 北京中星微电子有限公司,一种摄像头用应急 JPEG 编码方法,中国,发明专利.,200810105437,2008/10/15,1-20。

[2] 骆云志,刘治红,视频监控技术发展综述,兵工自动化[M],2009(28),1-3。

[3] 黄恒强,基于 ARM 和 Linux 的嵌入式远程视频监控系统设计[D],南京理工大学,2008。

[4] 李珍辉,段斌，基于 ARM 的嵌入式监控系统设计与实现[J]，微计算机信息，2008（ 10-2）,142-144。

[5] 赵永勇,张永健，基于 Video4Linux 的视频图像采集实现[J]，电脑编程技巧与维护,2006(5),75-77。

[6] 吴士力,刘奇,朱兰,嵌入式 Linux 应用开发全程解析与实战[M]，机械工业出版社,2010，25-43。

[7] 李驹光,ARM 应用系统开发详解[M],清华大学出版社,2003,28。

[8] 于明,ARM9 嵌入式系统设计与开发教程[M],电子工业出版社,2006,34-67。

[9] 徐严明,Linux 编程指南[M],科学出版社,2000,10。

[10] 刘志伟,基于 ARM 的嵌入式图像监控系统研究[D],西安工业大学硕士论文,2006(3),18。

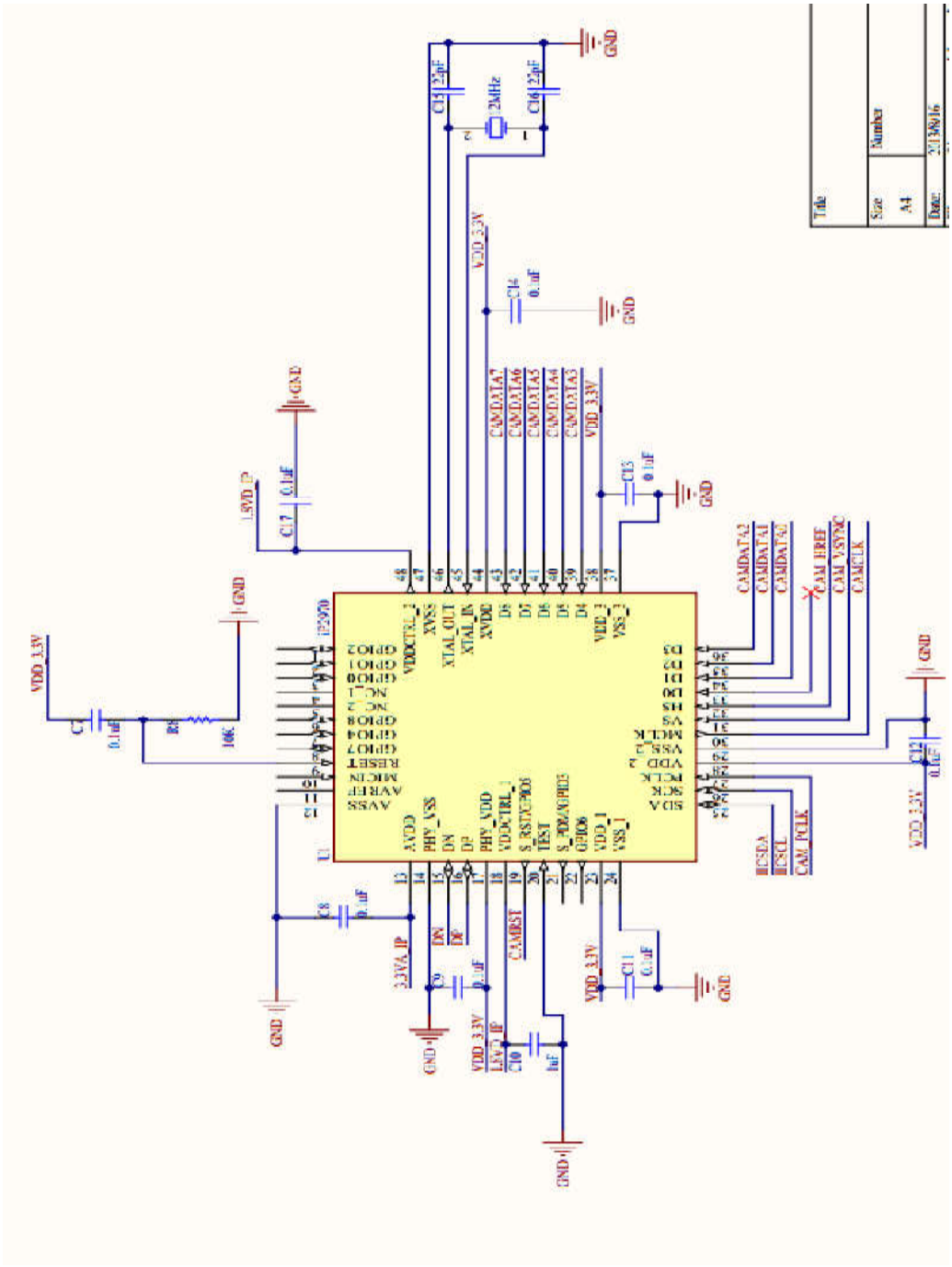
[11] 俞露,基于 ARM 的嵌入式系统硬件设计[D],浙江大学硕士学位论文,2003。

[12] 付保川,班建民,陆卫忠,等,基于嵌入式 Web 的远程监控系统设计[J],微计算机信息,2005(7Z),58-60。

[15] 戴燕云,王柏祥,章专,基于 GPRS 的嵌入式实时图像监控系统[J]，工业控制计算机,2005,18(9),9-10。

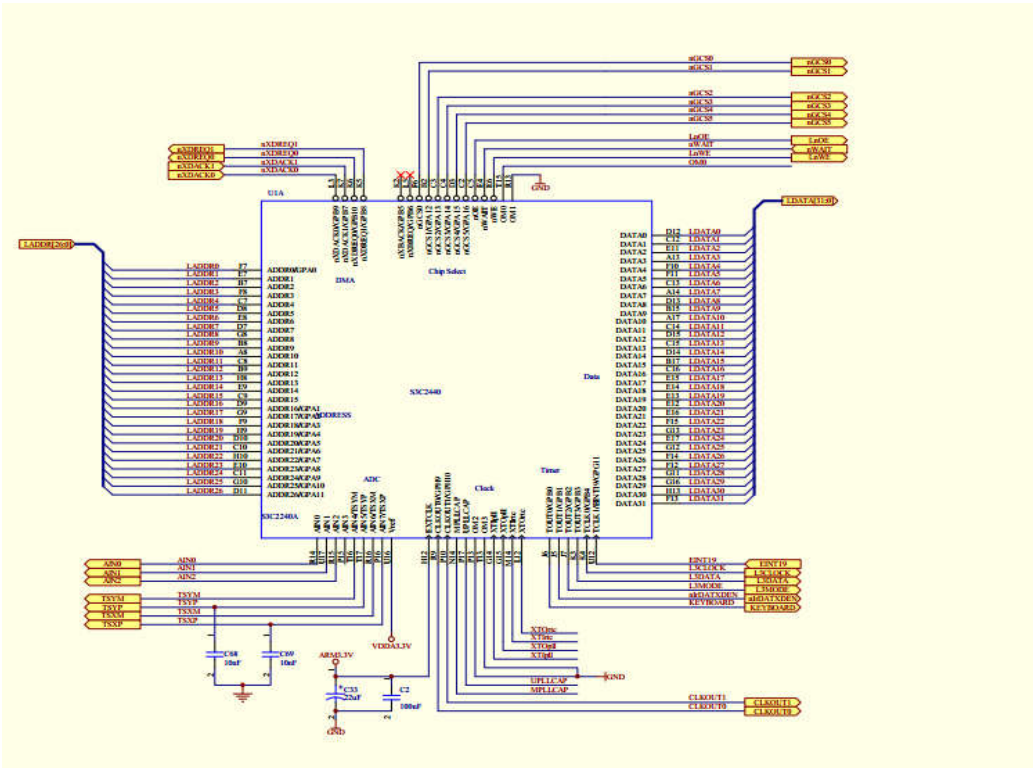
附录一 硬件原理图

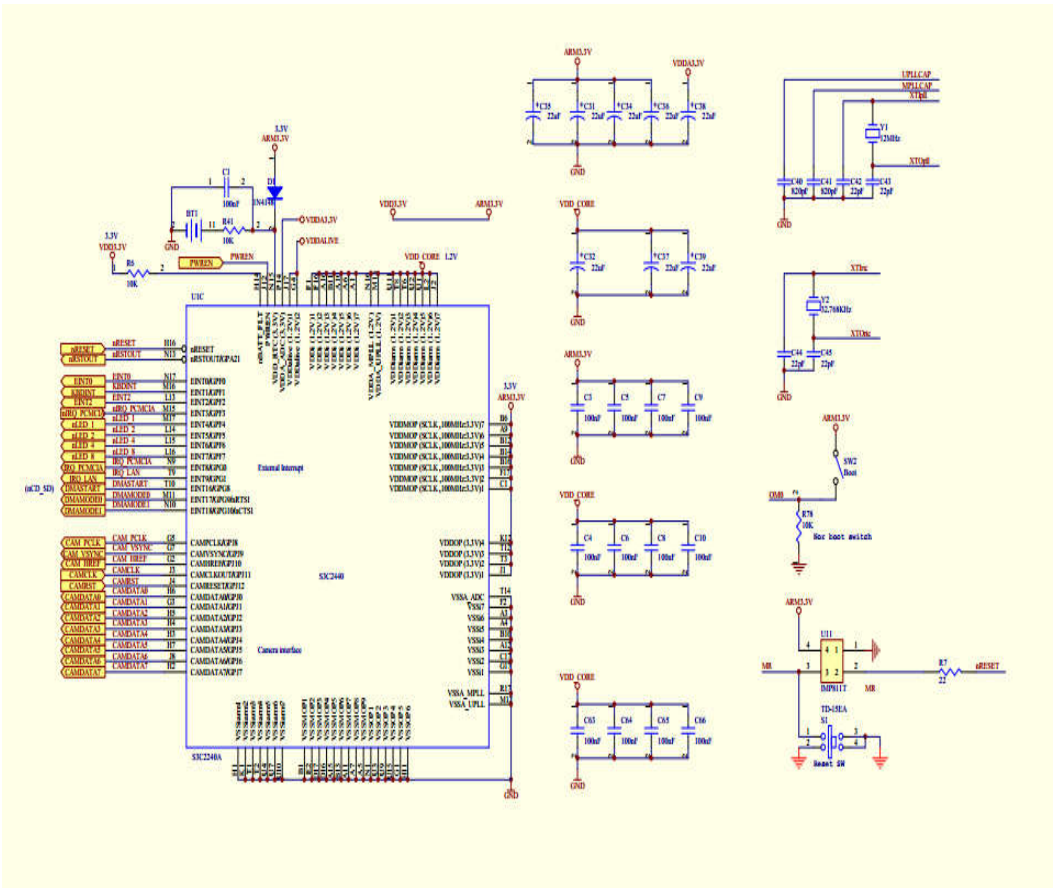
图附一，是摄像头 OV7740 硬件原理图



图附一 0V7740 硬件芯片图

图附二，是 JZ2440 硬件芯片图原理图





图附二 JZ2440 硬件芯片图