学　　号 14284060xx
等　　第

苏州大学实验报告

采用嵌入式 Linux 的高清视频采集与处理系统

院 (系) 名 称：　电子信息学院

专 业 名 称：　14 通信工程 (嵌入式培养)

学 生 姓 名：　黄兴

课 程 名 称：　Linux 操作系统

2015-2016 学年第一学期

摘要

视频技术在安防监控、医疗卫生、视频通信等领域应用越来越广泛，而高清视频应用是现在以及未来发展重点。基于此背景，本设计实现了一款具有智能图像处理功能的嵌入式高清视频采集与处理系统。

本文首先分析了具有智能图像处理功能的嵌入式高清视频采集与处理系统总体设计方案。围绕这个方案详细介绍了具体实现过程，主要包括以下内容：

1. 系统硬件设计。主要包括视频采集、处理器、电源、USB、串口、ISP、MMC、HDMI等模块电路原理图设计。
2. 嵌入式 Linux 软件开发平台搭建。包括系统内核裁剪与移植、交叉编译环境搭建、BootLoader 编译移植以及文件系统制作等。
3. 系统软件的设计。包括视频驱动设计及配置，采集程序设计以及图像增强、边缘检测、人脸跟踪等视频处理算法实现。
4. QT 界面设计及 OSD 功能实现。

总的来说，本文针对的高清视频采集与处理系统，能够达到 720p/30f 的要求，理论上可以达到 720p/60f。支持视频图像的边缘检测、图像增强、人脸跟踪、底片视频等处理功能以及具有人机交互界面。

关键词：高清视频采集；图像处理；嵌入式；Linux；CMOS；QT

合作者： 合作甲　14284060xx　14 通信工程 (嵌入式培养)
合作乙　14284060xx　14 通信工程 (嵌入式培养)
合作丙　14284060xx　14 通信工程 (嵌入式培养)

Abstract

In the field of security monitoring, health care and video communications, video technology is more and more widely, HD video applications now and in the future development priorities. Measure the clarity of the indicators is the image resolution, the greater the resolution the sharp the image, but most video applications on the market do not have HD video capture and processing capacity of its one of the main reasons is the embedded processors the ability to reach a HD video data. Hardware technology hinders the development of the embedded HD video processing system. Based on this background, we propose an intelligent image processing functions HD video acquisition and processing system.

To analyze and compare existing pros and cons of the video capture system, under proposed paragraph intelligent image processing functions embedded HD video acquisition and processing system for the overall design scheme. Around the program details of a specific implementation method, mainly include the following:

1. System hardware design. Include video capture, the processor, power supply, USB, serial port, the ISP, MMC, the HDMI module circuit schematic design.
2. Embedded Linux software platform. Include cropped transplantation of the system kernel, cross-compilation environment to build, u-boot compile transplantation.
3. System software design. Include video-driven design and configuration, acquisition program design and image increase, edge detection, face tracking and video processing algorithm.
4. The QT interface design and realization of the OSD functions.

Overall, this article for HD video acquisition and processing system, to achieve requirements of 720p/30f, could theoretically reach 720p/60f. Support video image enhancement, face tacking, negatives and video processing capabilities and interactive interface. The system is a good prospect of application in the field of medic image processing and security monitoring.

Key words: HD Video Acquisition; Image Processing; Embedded; Linux; CMOS; QT

目录

第 1 章　绪论

1.1　项目的背景及意义

1.1.1　背景

随着国民经济的发展和社会的进步，多媒体技术和通信技术得到迅猛发展，医疗卫生、安全监控等行业对视频的高清晰度和稳定性要求越来越高，在商场、街道、仓库和小区等监控场所，以前 D1、CIF 等普通格式视频越来越不能满足人们的日常生活需求 [1]。而目前 720P、1080P 等高清视频采集与处理大部分由 PC 机来实现，由于 PC 机处理速度快，内存大，可以完成视频采集、处理与传输等功能，但是 PC 机也存在很多缺点，如功耗比较高且体积大，在应用方面受到限制。近年来嵌入式系统得到迅猛发展，而且成本越来越低，性能越来越高，能够满足对体积、可靠性、功能、成本、功耗等有严格要求的场合，且该系统一般都能够实现网络数据传输，音视频编解码处理，多任务管理等功能。如今随着图像技术的快速发展，出现了基于嵌入式具有图像处理功能的视频采集系统，但是市场上大部分嵌入式视频应用系统还不具备高清视频采集与处理能力，其主要原因之一是以往的嵌入式处理器达不到处理高清视频数据的能力，这个硬件上的技术瓶颈阻碍了医疗、监控等行业嵌入式高清智能视频处理系统的发展[2]。

本文在此背景下研究在嵌入式系统下实现从高清视频前端采集，视频图像处理，到视频数据输出显示。

1.1.2　意义

目前市场上一般的嵌入式视频采集系统采集的视频分辨率都比较低，分辨率大小为 352×288 或 704×576，即视频格式为 CIF 或 D1。该视频对处理器的处理速度要求不高，通常嵌入式处理器都能实时处理该视频数据。但是这种系统也存在很多缺点，比如在安防监控领域，由于分辨率低即图像清晰度不高，一旦出现特殊情况，在回放录像中就无法识别众多细节信息，如“面部特征”，“车牌号”等，还有在医疗领域，如果视频图像不够清晰，那么就无法正确指导医师确诊病人病情。在一些特殊的应用场合还需要对视频图像进行特殊处理，如在安防领域，可能需要对在监测区域内的人脸进行实时跟踪，以便更好的保证该区域内的财产安全，在医疗领域则可能需要对医学视频图像进行增强，边缘检测，黑白效果等方面的视频处理，以便能尽快的从视频图片中

观察出病因的所在 [3]。本文系统处理的视频信号为全高清数字信号，前端 CMOS 模块将采集的模拟信号进行 A/D 变换后送到微处理器处理，由微处理器对采集的信号做硬件视频处理后由软件对视频数据做图像算法处理，并对处理后的视频数据通过 HDMI 接口输出显示，同时设计人机交互界面。该系统具有结构简单，操作方便，易于扩展等特点。

1.2　国内外研究现状

1.2.1　视频采集技术的发展现状与趋势

目前市场上主要有以下三种视频采集系统 [4, 5]：

(1) 模拟信号视频采集

基于模拟信号的视频采集系统功能强大且价格较低，其技术发展至今已经相当成熟，该系统主要有模拟设备组成，一般包括磁带录像机和模拟视频矩阵。现在有些场合还在使用，但是由于网络技术的发展，需要远程控制或数据传输，而该系统无法联网，且无法传输较长的距离，所以该设备已逐步被其他设备所取代。

(2) 基于 PC 机的视频采集系统

如今 PC 机的处理器处理能力越来越强，视频图像处理技术大大提高，人们开始利用计算机进行数字视频采集与处理。由前端采集到视频数据通过视频图像采集卡传送到本地计算机，再由计算机设备上的专门图像处理软件对采集的图像处理，还可以通过网络，将该处理后的视频数据传送到远端计算机。该视频采集系统功能强大，有丰富的图像处理软件，但是存在体积大、功耗高、可扩展性差等缺点。

(3) 基于嵌入式的视频采集系统

近年来无论在硬件或软件方面，嵌入式技术都得到快速发展，尤其是嵌入式操作系统具有可裁剪移植、多任务、实时性、功耗低、体积小等方面优点，使嵌入式技术成为如今研究的热点。嵌入式视频采集系统是将视频采集与该技术结合，该系统内部有微处理器，将前端采集的视频数据处理后可以通过网络传送到远端输出显示或者直接通过 HDMI 或 VGA 等接口直接输出显示。

嵌入式系统与前面两种系统相比，具有设备体积小、稳定性高、功耗低、可扩展性强、维护费用低、无需专人看护、网络功能强大、网络资源利用率高、使用方便等明显优势，正成为当前应用开发的热点。基于嵌入式的视频采集系统不仅具有上述优点，而且采集的全数字视频信号便于存储及处理，比较符合当前视频采集技术的发展趋势。因此，本文设计的系统便是基于嵌入式技术。

基于嵌入式技术及视频处理技术的发展，视频采集系统发展趋势如下：

(1) 小型化

为便于安装和携带方便，需要的嵌入式设备体积包括视频采集系统要尽可能的小巧。

(2) 大存储量
像在安防监控领域需要记录大量视频数据，需要更大的存储空间来存储视频信息。
(3) 高分辨率
如今手机、电视等屏幕分辨率不断提高，这就要求采集的视频图像分辨率也要不断提高。
(4) 高采样率
在医疗等领域需要记录瞬间发生的情况，就需要设备在很短时间内采样更多的帧数，这对处理器的处理能力也提出更高的要求。
(5) 网络化
为了将高速高清视频传送到远程接收设备，网络化必不可少，且对网络设备的性能也提出了更高的要求。
(6) 丰富的人机交互界面
为了用户使用方便，需要设计丰富的人机交互界面。

1.2.2　高清视频采集系统研究现状

高清视频采集系统是由一套完整的高清设备组成，包括图像采集、传输、存储、显示等，每个部分均要达到高清标准，只要有一个环节达不到要求，系统技术指标就达不到高清系统要求 [6]。高清可以分为三种格式：720P，1080i 和 1080P，其中 i 代表隔行扫描，P 代表逐行扫描，如果系统分辨率达到 720P 或 1080i，就称为高清系统，1080p 则称为全高清系统 [7]。在监控领域，传统的监控系统最高分辨率只能达到 CIF 或者 D1 格式，分辨率只有 352×288 和 704×576 大小。而高的视频分辨率对视频采集设备也提出了很高的要求，一般系统前端至少需要 200 万像素的摄像头 [8]，本文系统采用 500 万像素摄像头。

目前市场上主要前端视频采集图像传感器一般采用 CCD 和 CMOS 两种。CCD 传感器是电荷耦合器，输出的是模拟信号，通过 AD 转换得到数字信号，CMOS 传感器自身带有 AD 转换，可以直接输出数字信号。在同等的低照度条件下 CCD 感光性较好，但随着 CMOS 集成电路工艺的不断进步和完善，CMOS 传感器在很大方面已经能与 CCD 相媲美，且价格低廉 [9, 10]。目前国内做高像素的图像传感器这方面的研究还比较薄弱，专利技术还是被国外所掌握。

如今，我国的高清视频采集系统还是处于发展阶段，国内的供需比比较大，传统的基于嵌入式的视频采集系统逐渐不能满足人们的需要，但是由于软硬件技术、成本等方面限制，使得高清视频采集与处理系统的发展还需要一段时间。

1.3　项目主要内容

本文内容主要包括以下几个方面：

1. 分析系统实际应用需求，给出了基于 DM3730 高清视频采集与处理系统的总体设计框架以及系统主要芯片分析与选型。
2. 根据系统设计框架，设计系统硬件平台，包括原理图设计。硬件平台分为视频采集模块和处理器平台模块，而处理器平台模块包含处理器、电源、USB、DM900 网卡、串口、ISP、MMC、HDMI 等模块电路。
3. 搭建基于嵌入式 Linux 的系统软件开发平台，包括系统内核裁剪与移植、交叉编译环境搭建、u-boot 的编译移植，文件系统制作，Samba 和 NFS 服务器配置等。
4. 在搭建好的编译环境下，基于 C 语言设计系统软件。软件是按模块划分编程，包括视频驱动设计及配置，采集程序设计以及图像增强、边缘检测、人脸跟踪等视频处理算法实现。
5. 为了实现人机交互，利用 QT 设计用户界面，并实现界面的 OSD (on screen display) 功能。

1.4　结构安排

本文共分为 7 个章节，每个章节具体内容如下：

第 1 章绪论部分，主要介绍课题研究背景及意义，视频采集的研究发展现状与趋势，高清视频采集系统研究现状，以及内容和结构安排。

第 2 章系统总体方案分析与设计，主要介绍系统需求分析及总体方案设计，然后分别介绍硬件方案设计，包括硬件框架设计与主要芯片选型，最后介绍软件方案设计，包括操作系统分析与选取和软件总体设计方案。

第 3 章硬件平台设计，主要介绍硬件模块划分以及主要模块电路的设计过程。

第 4 章嵌入式 Linux 软件开发平台搭建，主要介绍嵌入式 Linux 系统开发环境建立，内核裁剪与移植，u-boot 移植，文件系统制作，以及 NFS，Samba 服务器的配置等。

第 5 章系统软件设计及算法实现，主要介绍视频采集驱动程序设计及参数配置，视频采集应用程序设计以及视频处理方面的图像增强、边缘检测、人脸跟踪等算法实现过程。

第 6 章 Qt/E 界面设计及 OSD 功能实现，主要介绍 Qt/E 人机交互界面设计与实现以及 OSD 功能实现。

第 7 章结论与展望，对全文进行总结，并分析进一步需要完善的工作。

第 2 章　高清视频采集与处理系统分析与设计

2.1　系统总体方案分析与设计

2.1.1　系统需求分析

高清视频采集与处理系统在不同的应用环境下有不同的应用需求，本文主要针对安防监控、医疗卫生等领域设计的应用系统。高清采集与处理系统一般由高清视频采集模块、高速传输、高速处理器、存储单元、高清显示接口等几个部分组成 [11]。结合本系统的实际情况，以及系统稳定性及可扩展性等方面要求，系统设计时需要结合以下几个设计原则 [12]：

1. 技术先进性：结合国内外高清视频采集与处理技术的发展趋势和当前微处理器技术水平，采用先进且比较成熟的硬件和软件技术。
2. 可靠性：系统应该能够长时间正常稳定工作，并具有一定的纠错能力，在系统出现故障，可以使系统整体工作不受影响或具有自启动功能。
3. 可扩展性：设计的系统应该有良好的扩展性能，为将来的系统扩展预留相应的接口和存储空间。
4. 标准化、模块化：程序设计过程中应该保证编程的标准化，接口的标准化，程序设计应该按模块划分，依照低耦合高内聚的原则，各个模块技术相对独立又是有机统一整体。
5. 高性价比：在能够满足系统设计需求的前提下，应确保系统最优性价比。
6. 易操作性：为方便用户使用，系统应设计非常友好的人机交互界面，易操作的图标按钮等。

本文设计的高清视频采集与处理系统结合实际应用需求和以上的设计原则，制定了系统的设计目标如下：

1. 系统前端采集模块至少能够采集 200 万像素的高清视频信号。
2. 系统应该带有 USB、TF 接口、串口、SPI 接口、IIC 接口、JTAG 接口、CAMERA 接口、TFT 屏接口、触摸屏接口、网络接口、HDMI 接口。
3. 能够实现对视频进行图像增强，边缘检测，人脸跟踪等算法处理。
4. 高清视频输出分辨率和帧率至少为 720p/30f。
5. 具有人机交互界面。

2.1.2　系统总体方案设计

嵌入式高清视频采集与处理系统包含硬件和软件两个部分，需要用到软硬件等多方面知识。如果要想有一个好的设计基础，就必须有一个好的设计方案。系统硬件部分主要由前端采集模块、微处理器模块、存储模块、显示模块以及各种通信接口，如 SPI、IIC、TFT、ISP、HDMI 等组成。为了在开发阶段调试方便，需要设计调试接口，如 UART、JTAG 等。系统软件部分由嵌入式操作系统，设备驱动程序，应用程序等组成。操作系统又包括引导程序 BootLoader、内核、根文件系统等，驱动程序主要为图像采集传感器驱动程序，应用程序包括视频采集程序、视频处理程序、界面设计程序等。该系统的总体设计框架如图 2.1 所示。

图 2.1: 系统总体设计框图

2.2　系统硬件方案设计

2.2.1　系统硬件总体框架设计

为了便于以后程序的开发以及节省成本，首先要选择一个好的硬件平台，好的平台是系统成功的基础。而硬件平台的核心就是微处理器，目前市场上微处理器大约有 1000 多种 [13]，如何从中找出合适的微处理器也比较困难。本文在上述的设计原则及设计目标基础上，综合考虑其他因素，设计了如下的硬件整体设计框图，见图 2.2（具体芯片的选型参见下节）。

图中我们可以看到，硬件设备由微处理器和外围设备组成，微处理器采用的 TI 公司的 OMAP 系列的 TMS320DM3730 处理器，该处理器为双核结构 ARM+DSP，其中 ARM 核为 Cortex-A8 架构，主频为 1 GHz，DSP 核为 C64x 架构，主频为 800 MHz。图像采集传感器采用的 APTINA 公司的 MT9P031 芯片，该芯片为 CMOS 系列图像传感器，最大可以采集 5 Mp 像素的数字图像信号。系统电源由 TI 公司的 TPS65930 电源芯片供应，提供 1.2 V、1.8 V、3.3 V 等电压。系统的网络模块由 DM9000 网络芯片提供，外部存储单元采用 Micron 公司的高速 DDR RAM 和 NAND FLASH 实现高速数据传输与存储，同时系统还提供 MMC 接口和 USB2.0 接口，MMC 接口可以用来存储数据以及系

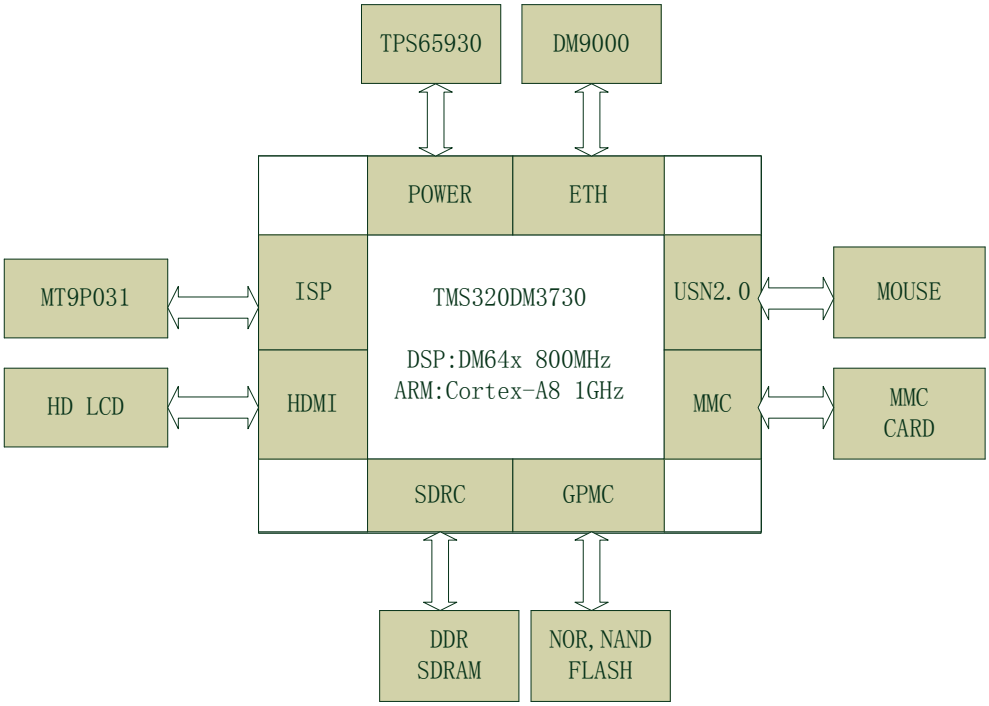

图 2.2: 系统硬件设计框图

统更新使用，USB2.0 接口可以接鼠标操作系统界面。

根据该硬件框图，系统总体的工作流程为，系统首先通过 CMOS 镜头采集高清高速视频数据，通过 ISP 接口输出到 DM3730 的 ARM 核处理单元，由芯片的硬件单元对采集的视频进行去噪，白平衡，黑色补偿等前端与后端硬件处理。如果在人机界面选择需要视频图像处理，如图像增强，人脸跟踪等，则系统将前期处理后的数据送到 DSP 核，再由该 DSP 核对视频数据做算法处理，并将处理的结果送到 ARM 核，由该处理单元将处理后的数据通过 HDMI 接口输出显示或者压缩后通过网络输出。

2.2.2　芯片的选型与介绍

2.2.2.1　微处理器选型与介绍

嵌入式系统开发的首要步骤就是选择一款合适的嵌入式处理器，相当于系统的大脑。如今嵌入式微处理器芯片主要有 ARM、DSP、ASIC、FPGA 等几种，且不同的处理器有不同的用途 [14]。ARM 作为通用嵌入式处理器，由于其能够运行良好的实时操作系统包括 Windows CE 和 Linux 系统，其主频速度越来越高，开发相对容易，比较适合较为复杂的控制系统，现在正被广大用户热捧，但是对于处理高清视频数据，由于数据量巨大，而其没有专门的硬件算法功能，运行起来会比较吃力。DSP 芯片运算能力很强大和可编程的灵活性很高，且功耗比较低，很合适处理高清视频这种大数据量的运算，但是它的操作系统比较简单，当使用复杂的功能时比如本文设计高清视频采

集与处理系统，在实现人机交互功能上就比较困难。FPGA 具有强大的并行处理能力以及可定制流水线的结构，也比较适合本系统的设计需求，但是其成本高，开发难度较大，无法短期内将产品推向市场。ASIC 的出现填补了电子领域方面的空缺，且成本低，编码速度快，但是芯片本身设计上存在些缺陷，如编解码的灵活性有点差，且开发周期较长 [15]。经过各类芯片的优缺点比较，并结合处理速度，实时控制，成本，功耗等因素，最终选用了德州仪器（TI）公司 OMAP（Open Multimedia Application）系列的 TMS320DM3730 处理器芯片。

TMS320DM3730 是 TI 公司推出的一款灵活、高效的片上数字音视频系统微处理器，该处理器是由 1GHz 的 ARM Cortex-A8 Core 和 800 MHz 的 TMS64x+ DSP Core 的两部分组成，该处理器可以解决 ARM 处理器无法处理高清视频大数据量和 DSP 无法运行功能强大的操作系统问题，且该处理器集成了 3D 图形处理器、视频加速器 (IVA)、USB 2.0 、支持 MMC/SD 卡、串口等，支持高清 720p、1080p 视频解码。最大支持 60f/s 实时视频处理, 编码解码能力可达每秒 500 万像素，有前端和后端的视频处理子系统，可支持视频预览、图像缩放、自动聚焦、曝光、白平衡等功能 [16]。外围设备集成了非常丰富的视频和网络通信接口。

2.2.2.2　图像传感器分析与选取

高清视频采集与处理系统的前端视频采集信号必须是高清视频信号，这也是高清系统实现的基础。目前市场上使用的图像传感器分为两类：基于 CCD 和基于 CMOS 的图像采集传感器，这两类传感器各有优缺点，在高清视频采集系统中均有使用 [9]。CCD (Charge Coupled Device)，即 “电荷耦合器件”，是一种感光半导体芯片，在接受光照后，感光元件产生对应的电流，电流大小和光强对应，光照越大，电流越大。CMOS (Complementary Metal Oxide Semiconductor)，即 “互补金属氧化物半导体”，两者工作原理本质上没有区别，只是在制造上的区别，CCD 图像传感器是集成在半导体单晶材料上，而 CMOS 图像传感器是集成在金属氧化物半导体材料上 [10]。两者的技术与性能比较如下：

(1) 技术上比较

CCD 图像传感器中每一个感光元件采集的电荷信息都不做处理，而是直接传输发到下一个存储单元，经信号整合后在传输到下一个存储单元，最后输出统一的模拟信号。由于信号比较微弱，所以必须先放大处理后在模数转换，最终生成二进制的数字信号，传输到处理器进行处理。CCD 图像传感器感光元件产生的模拟信号转移和读取需要有时钟控制电路和三组不同的电源相配合，整个电路较为复杂且速度相对较慢。

CMOS 图像传感器中所有感光元件都由模数转换逻辑和放大器构成，在光照下产生的电荷信息直接被其放大器放大后由模数转换单元转化成对应的数字信号，可以直接将该信号传输到处理器处理，而不需要专门的模数转换芯片，这样信号读取简单，处

理速度比 CCD 相对较快，但早期由于 CMOS 集成度高，各个光感元件与电路之间距离很近，相互之间的光、电、磁干扰比较严重，图像质量影响较大。

(2) 性能上比较

ISO 感光度：CMOS 每个感光元件都由一个感光二极管和四个晶体管构成，并且每个元件还包括模数转换和放大器电路，由于每个感光元件包含的设备太多，就会使每一个感光元件的光感表面积减少。因此，在相同的条件下，CCD 的感光度会高于 CMOS 传感器。

分辨率：由于 CCD 传感器相对简单，而 CMOS 相对复杂，导致 CMOS 的像素尺度很难达到 CCD 传感器的像素，所以两者在相同尺寸下，CMOS 传感器的分辨率会低于 CCD 传感器。

成本：CCD 传感器采用电荷传递的方式传送电荷信息，只要其中一个感光元件损坏，那么就会导致一整排的数据不能传送，因此在制造工艺上比较困难，成品率相对较低，从而成本较高。而 CMOS 传感器采用常用的 CMOS 工艺，可以比较容易的集成周边电路到芯片中，可以节约外围芯片，且生产工艺相对简单，制造成本较低。

噪点：因为 CMOS 传感器每个感光元件都由感光二极管、放大器及模数转换电路等组成，那么 CMOS 传感器的像素越高，需要的放大器电路就越多，而放大器电路属于模拟电路，很难让每个放大器都保持一致的结果，这相对于只有一个放大器的 CCD 传感器来说，噪点就会增加，影响图像质量 [17]。

近几年，随着 CMOS 传感技术的不断发展，CMOS 的性能与 CCD 已经很接近，采集的图像分辨率能够达到 720P 甚至 1080P，且其成本低，集成容易等特点，被广泛使用。本文设计的高清视频采集与处理系统在综合考虑各种因素下，最终选择了 APTINA 公司的 MT9P031 CMOS 图像传感器。该图像传感器的内部结构图见 2.3。

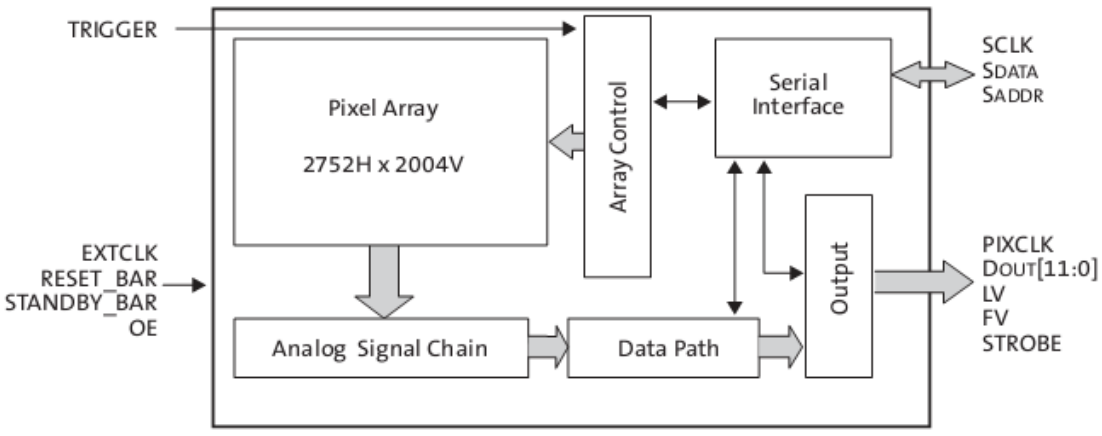

图 2.3: MT9P031 内部结构图

该传感器最大有效像素为 2593H × 1944V，MT9P031 由其内部锁相环产生 6 时钟，最大像素速率达到 96 Mp/s，输出模式为 Bayer RGB 模式，支持 binging 和 skipping 模式，12 bit 片上 AD。该传感器还具有色彩增益调节，模拟数字增益设置，黑电平校正，偏移补偿，曝光时间控制等功能，使得算法能够很好实现。系统通过 I2C 对传感器写入读取控制。

2.2.2.3　电源管理芯片分析与选取

本文选用电源管理芯片 TPS65930，该芯片与 OMAP™ 系列处理器配套使用，因为该芯片不仅能够提供各种芯片需要的电压，而且还包括电源管理控制器、USB 高速传输控制、LED 驱动控制、模数转换（ADC）、实时时钟（RTC）和嵌入式时钟管理（EPC）等，此外还包括完整的两路数模转换音频信号和两个 ADC 双语音频道、一路标准的音频采样率时分复用（TDM）接口，可以在立体声下行通道播放标准的音频。电源管理芯片中还包含了一个 USB 高速收发器，所以可以给系统扩展 USB OTG 功能。该电源管理芯片提供的丰富接口可以帮助扩展处理器的功能，并减轻处理器的负担。TPS65930 与处理器之间使用 I 协议通信。

2.3　系统软件方案设计

系统所有功能都由软件来实现，而操作系统是基础，所以软件系统一般由操作系统、驱动软件、应用软件等构成。本视频采集与处理系统的软件结构如图 2.4。

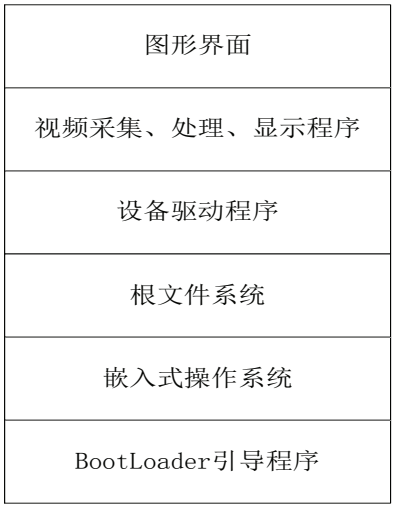

图 2.4: 软件系统结构

2.3.1　嵌入式操作系统的分析与选取

嵌入式操作系统由于其良好的可裁剪性，实时性，多任务处理等优点，为系统的开发提供极大的方便，但目前市场上存在多种嵌入式操作系统，每种操作系统针对不同的用途。因此，选择嵌入式操作系统一般从以下几个方面考虑 [4]：

(1) 可移植性

目前市场上处理器种类繁多，考虑到以后系统扩展可能需要更换处理器，就必须选用一种可以支持大部分的处理器的通用嵌入式操作系统。

(2) 可裁剪性

嵌入式操作系统一般资源有限，需要对系统的内核进行裁剪，去掉不需要的功能模块，使裁剪出来的嵌入式操作系统最合适本系统。

(3) 实时性

因为本文设计的视频采集与处理系统，采集的视频数据实时的，且数据量巨大，所以选择的操作系统的实时性一定要很高。

(4) 开发工具和技术支持

项目的开发一般时间都比较紧张，为了更快开发出该系统，必须充分利用系统的资源与技术支持。一个好的开发工具和技术支持，能够减少研发周期，节约成本。

目前市场上存在多种嵌入式操作系统 [18]，主要有 VxWorks，Windows CE，uC/OS-II，Linux 等，虽然各个系统都有各自缺点，但是前面三个系统的使用都需要付费，会增加项目的成本，而 Linux 系统不仅能够满足上面系统设计需求，而且是遵循 GPL 协议公开源码的免费操作系统。内核能够支持很多种硬件，大部分的硬件驱动都有公开的代码，可以任意修改。Linux 是一种类 UNIX 的操作系统，已经成为当今最为流行的开源操作系统之一，PC 机 Liunx 系统与嵌入式 Linux 系统内核代码相同，只是开发程序的编译环境不同，在 PC 机上编写的程序，经过特定的编译环境，就可以在嵌入式设备上运行，并通过串口或网络都可以实现对系统软件调试，这很方便程序的开发。综上所述，在考虑开发周期和成本等情况下，最终选用嵌入式 Linux 系统作为本文设计的高清视频采集与处理系统的嵌入式操作系统 [19]。

2.3.2　系统总体软件设计

在嵌入式 Linux 操作系统上，系统的软件设计主要包括驱动程序设计，视频采集程序设计及用户界面设计等。系统采用的是 C 语言编程，并在一个进程下采用多个线程的方式实现视频的采集、处理、显示等功能。系统软件设计流程图如图 2.5。

从图中可以看出，系统工作流程为首先系统初始化，然后初始化图形界面并加载视频驱动程序后，调用视频采集应用程序实时采集高清视频后通过 HDMI 接口实时输出显示。用户如果需要对视频进行处理，如视频图像增强、边缘检测、人脸跟踪等，则用户需要在图形界面上进行相应的操作，处理器将对该视频实时处理后传送到 HDMI

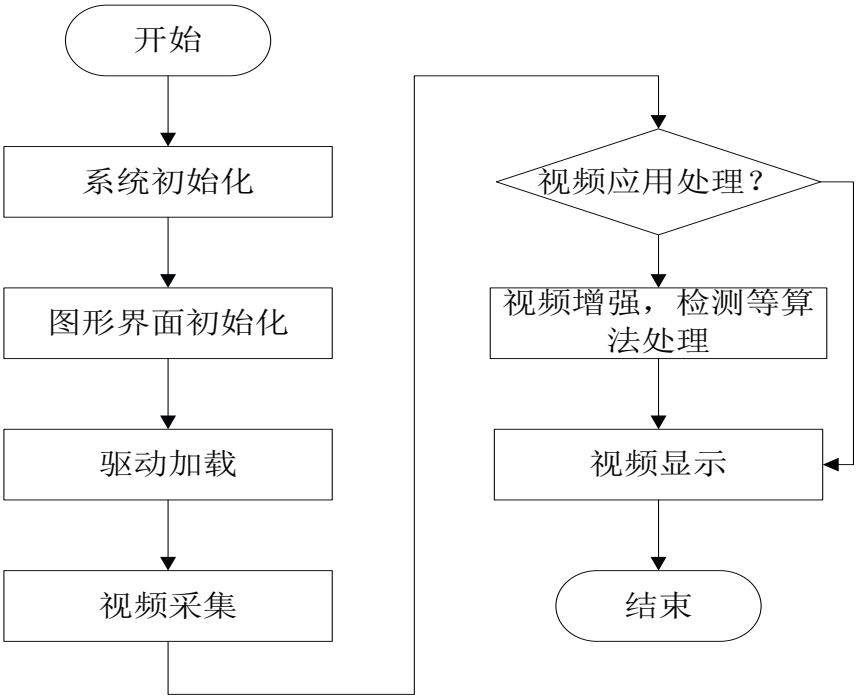

图 2.5: 系统软件设计流程图

接口输出显示。本系统软件中视频驱动程序和采集程序都是基于 V4L2（video 4 Linux 2）架构设计，该架构为视频程序开发提供一套规范的接口，该接口处于应用程序与硬件设备之间。当设备注册成功后，将会产生 `/dev/video0` 设备节点，应用程序可以通过该节点，利用 V2 提供的接口，实现对视频数据的实时采集 [20]。

为了更好地实现人机交互，需要设计图形用户界面 (Graphical User Interface，GUI)，GUI 可以帮助普通用户的快速使用，它是计算机系统不可或缺部分。目前有三种 GUI 系统在嵌入式系统中比较流行，分别是 OpenGUI、MiniGUI 和 QT/Embedded [21]，每一个 GUI 系统在功能特性，接口都不太相同，存在差异。需要根据具体需求进行选择。OpenGUI 是基于 X86 内核，占用存储空间较小，但是无法进行多线程操作且可移植性差。MiniGUI 侧重于窗口开发，图像引擎比较成熟。Qt/Embedded (简称 Qt/E) 相比较与前两种主要是针对于嵌入式系统环境设计的，提供数据库接口，对数据库操作比较方便且移植性较好。在综合考虑移植性和开发时间等情形下，本文系统选择了 Qt/E 来设计人机交互界面，并实现了 OSD 功能。Qt/E 是诺基亚开发的一个跨平台的 C++ 图形用户界面应用程序框架，很容易扩展，并且允许真正地组件编程。OSD（on screen display）表示一种在活动视频上叠加图形信息的技术，在日常生活中比如相机、电视等都比较常见。通常视频和 OSD 信息是分开的，在视频输出时，将视频和界面通过一定的技术叠加在一起，同时显示输出，实现人机交互的目的 [22]。

至此硬件和软件的设计方案已经确定，图 2.6 为本文系统开发的实物图。

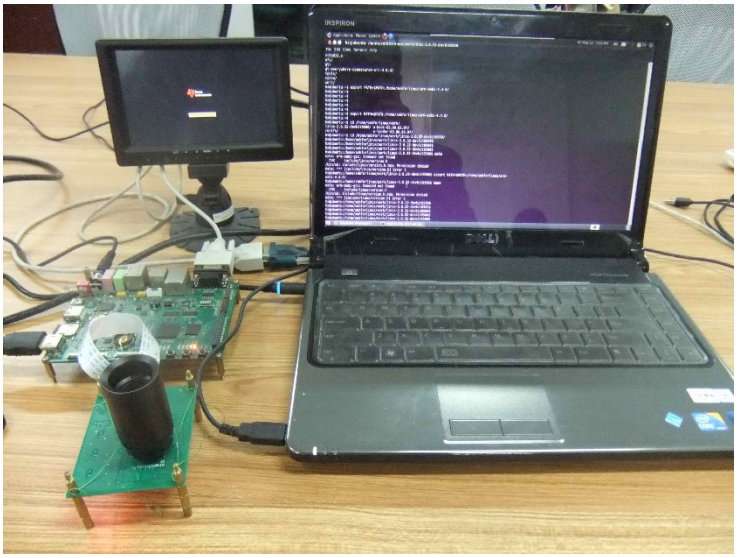

图 2.6: 系统开发实物图

2.4　小结

本节首先对系统的总体方案做了分析与设计，然后介绍了硬件模块的总体设计方案和对主要芯片的选择进行了分析与介绍。接下来对系统软件总体设计方案做了介绍，并分析软件系统大体需要分为哪些模块来实现，并确定选择嵌入式 Linux 系统作为本文开发的系统。由于需要实现人机交互，最后在分析现有的 GUI 图像用户界面软件的基础上，选择了基于 Qt/E 软件来设计人机交互界面。

第 3 章　系统硬件平台设计

本文第 2 章已经对系统的硬件电路总体设计框架作了详细介绍，以及对主要芯片的选型作了分析与介绍，并确定具体使用芯片型号。本章在上述的基础上将对硬件系统的主要模块设计过程分别进行介绍，包括设计原理，设计步骤等。

3.1　硬件电路模块划分

按照系统硬件设计框架，硬件平台共分为两个部分，分别是前端视频采集模块电路和基于处理器的硬件平台电路。按模块划分大概可以为以下几个模块：采集模块、处理器模块、电源模块、USB 模块、HDMI 模块、MMC 存储卡模块、网络模块、存储模块、JTAG 调试模块、串口模块以及复位电路等。模块划分结构如图 3.1 所示。

图 3.1: 硬件模块划分

3.2　硬件主要模块设计介绍

3.2.1　视频采集模块电路设计

视频采集模块单独制板，通过 FPC 扁平线连接至主板。视频采集传感器采用 APTINA 公司的像素 CMOS 传感器 MT9P031。DM3730 通过专用的视频采集片内外设 ISP 控制 MT9P031 工作，并接收视频信号。MT9P031 工作在高速模式下，其 IO 管脚电压为 2.8 V（资料显示 MT9P031 的 IO 管脚电压也可为 1.8 V，但只能工作在低速模式下），与 DM3730 的 IO 管脚电压（1.8 V）不一致。MT9P031 和 DM3730 电路连接时，

采用 SN74AVCH16T245GR 等电压转换芯片进行 IO 电压转换。

MT9P031 芯片工作需要 6~27 MHz 的时钟频率，该频率也可以由 DM3730 的 ISP 接口提供。为了调试方便，在系统设计时增加外部晶振电路，频率采用 MT9P031 推荐的 24 MHz。视频采集模块电路的原理图如图 3.2 所示。

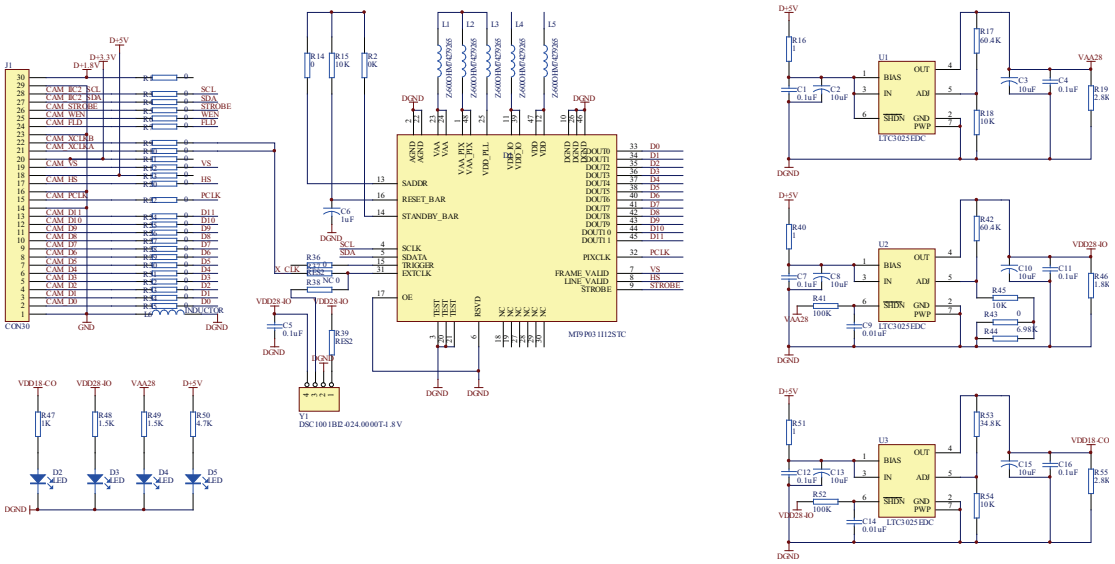

图 3.2: 视频采集模块原理图

由于 CMOS 传感器对电源质量比较敏感，如果电压波动较大，会影响视频图像的输出及芯片工作稳定性和寿命。MT9P031 对供电电源要求为数字电源 1.8 V 和模拟电源 2.8 V。系统特意通过 2 片 LTC3025EDC 把 5 V 电源转换为 1.8 V 和 2.8 V，分别独立提供稳定的、隔离的数字电源和模拟电源。

3.2.2　基于处理器的硬件平台设计

基于处理器的硬件平台主要实现对采集的视频数据进行处理、显示、存储以及传输等功能。处理器模块采用的是 TI 公司的 TMS320DM3730 芯片，该芯片采用 PBGA 封装，共有 423 个引脚，1.8 V 电压供电。

芯片电源由 TPS65930 电源芯片提供。TPS65930 片内外设 I2C 接口与处理器的 I2C 接口相连，进行两者之间必要的数据传输。

DDR 存储器与处理器 DDR 部分 SDRC 引脚连接；FLASH 存储器与处理器的 GPMC 引脚连接；MMC 存储卡模块与处理器的 MMC 部分数据引脚相连接；串口模块与处理器的 UART1 部分引脚相连；JTAG 调试模块与处理器的 JTAG 部分数据引脚相连接；处理器 ISP 接口和视频采集模块连接。

3.2.2.1　HDMI 模块电路设计

系统中高清视频数据处理后通过 HDMI (High Definition Multimedia Interface) 接口输出显示。该接口是一种数字化视频/音频接口技术，是适合影像传输的专用型数字化接口，可以同时传输视频和音频信号，最高传输数据位 5 Gbps。由于处理器处理后的数据并不能够直接通过 HDMI 输出显示，需要对其重新编码成 HDMI 输出的数据格式。本系统选择了 TI 公司的 TFP410 编码器，该芯片集成了高速数字接口、T.M.D.S. 编码器和三个差分信号驱动器，完成 24 位像素数据和一些控制信号接收，并通过编码算法把图像信号编码成适合在双绞线电缆上传输的高速、低电压、差分连续位流 (RGB 数据流)，是用于图形控制，无胶合连接的通用接口。其接口的特性优势是可选择总线宽度，可适应不同的电平信号和信号沿时序，适应 1.1 V-1.8 V 范围内的数字电平 low-EMI，高速总线宽度提供 12 位或者 24 位宽度选择，在 24 位真彩色格式的范围内，HDMI 显示分辨率频率高达 165 MHz [23]。

TFP410 支持从 VGA 到 UXGA 的分辨率，支持两种输入模式：12 位双边和 24 位单边，可由状态引脚配置也可以由 I2C 总线配置。12 位双边是在输入时钟的每个沿 12 位数据就会被锁存，24 位单边是在输入时钟的上升沿或下降沿锁存。在本系统中设计为 24 位单边输入，这两个引脚 BSEL=1，EDGE=1 需要设置为高电平。

具体电路连接中：TFP410PAP 的 IO 口电压可以工作在 1.8 V，TFP410PAP 和 DM3730 之间可以不接桥接芯片。而 DM3730 对 HDMI 外设的 IIC 接口控制需要采用电压转换芯片 TXS0102DCUR，因为 HDMI 外设的 IO 电压为 3.3 V。具体原理图设计见图 3.3。

3.2.2.2　电源模块电路设计

由第 2 章的元器件选型分析中可知，硬件电源模块选用的是 TI 公司的 TPS65930 电源管理芯片，该芯片主要为硬件电路提供 1.2 V、1.8 V、3.3 V 等电压，同时该芯片支持电源管理控制器、USB 高速传输控制、LED 驱动控制、模数转换（ADC）、实时时钟（RTC）和嵌入式时钟管理（EPC）等功能，所以在本设计中充分利用这些扩展功能，可以减轻处理器的负担，并能提高系统处理速度。本系统中扩展了 USB、LED 显示灯、ADC、按键、时钟等功能。电源管理芯片原理图设计是按电源模块和扩展功能模块设计，如图 3.4 和图 3.5。

3.2.2.3　USB 模块电路设计

系统的 USB 电路设计采用 USB3320 作为 USB2.0 控制芯片，支持 USB HOST 接口，支持高速（480 MHz），全速（12 MHz），和低速（1.5 MHz）模式。

USB3320 是高度集成的全功能高速 USB2.0 收发器，基于 SMSC 成熟的 ULPI 接口构建，支持多种参考时钟频率，可以接受来自晶振/振荡器的时钟讯号。USB3320 集成有一个 USB 切换器和 ESD 与 VBUS 过电压保护装置。USB 模块电路设计的原理图见

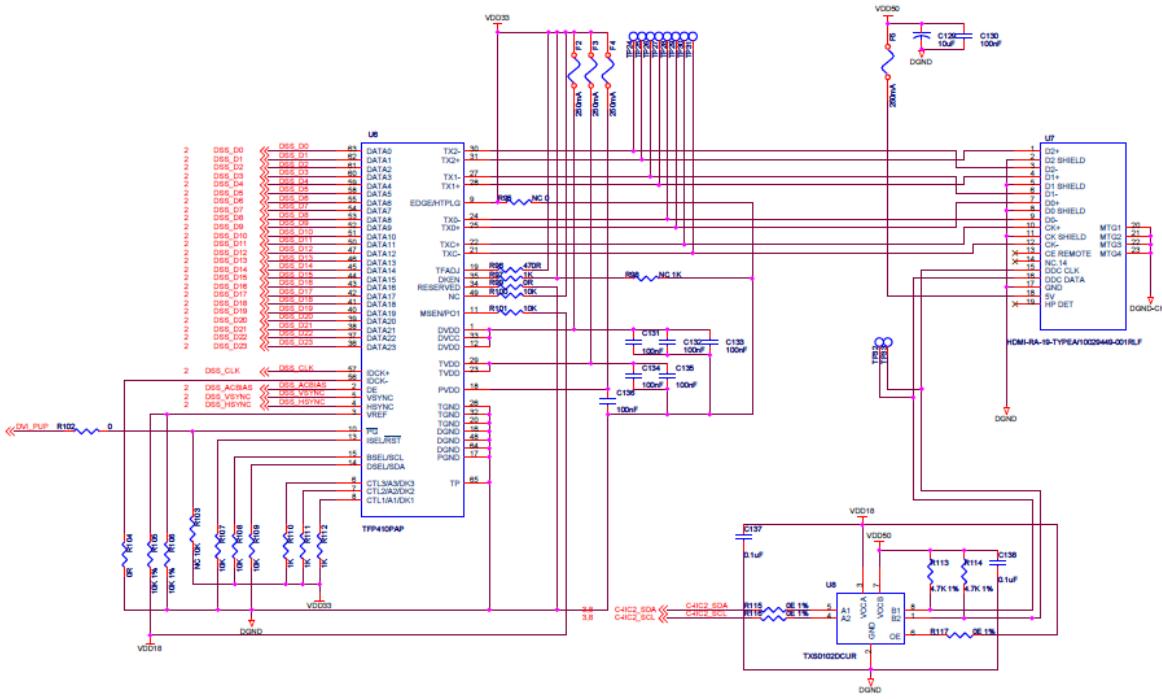

图 3.3: TFP410 设计原理图

图 3.6。

3.2.2.4　存储模块电路设计

存储芯片选择的是 MT96MAZAPCJA-5，该芯片是 SDRAM DDR 和 NAND FLASH 二合一的存储芯片，空间大小分别为 512 MiB。SDRAM DDR 是通过 SDRAM Controller (SDRC) 实现对数据的访问，而 NAND FLASH 是通过 GPMC 总线实现对数据的访问。

SDRAM DDR 是双倍速率同步动态随机存储器，相比与 SDRAM 一个周期只传输一次数据，该存储器在一个周期内传输两次数据。NAND FLASH 读写速度快，是非易失存储器，可以对块的存储器单元块进行擦写和再编程。系统中以 NAND FLASH 来存储 Bootloader、操作系统、程序代码等。

芯片的 SDRAM DDR 和 NAND FLASH 模块的数据引脚与控制引脚分别与处理器的 SDRC 和 GPMC 引脚相连。芯片采用 1.8 V 电压供电。存储模块电路设计原理图见图 3.7。

3.2.2.5　MMCI 存储模块电路设计

系统中读取的视频数据或抓拍的图片数据可以通过网络传输到远端系统也可以存储在内存卡中。现在通用的存储卡有 SD 卡和 TF 卡。SD 卡拥有高记忆容量、快速数据

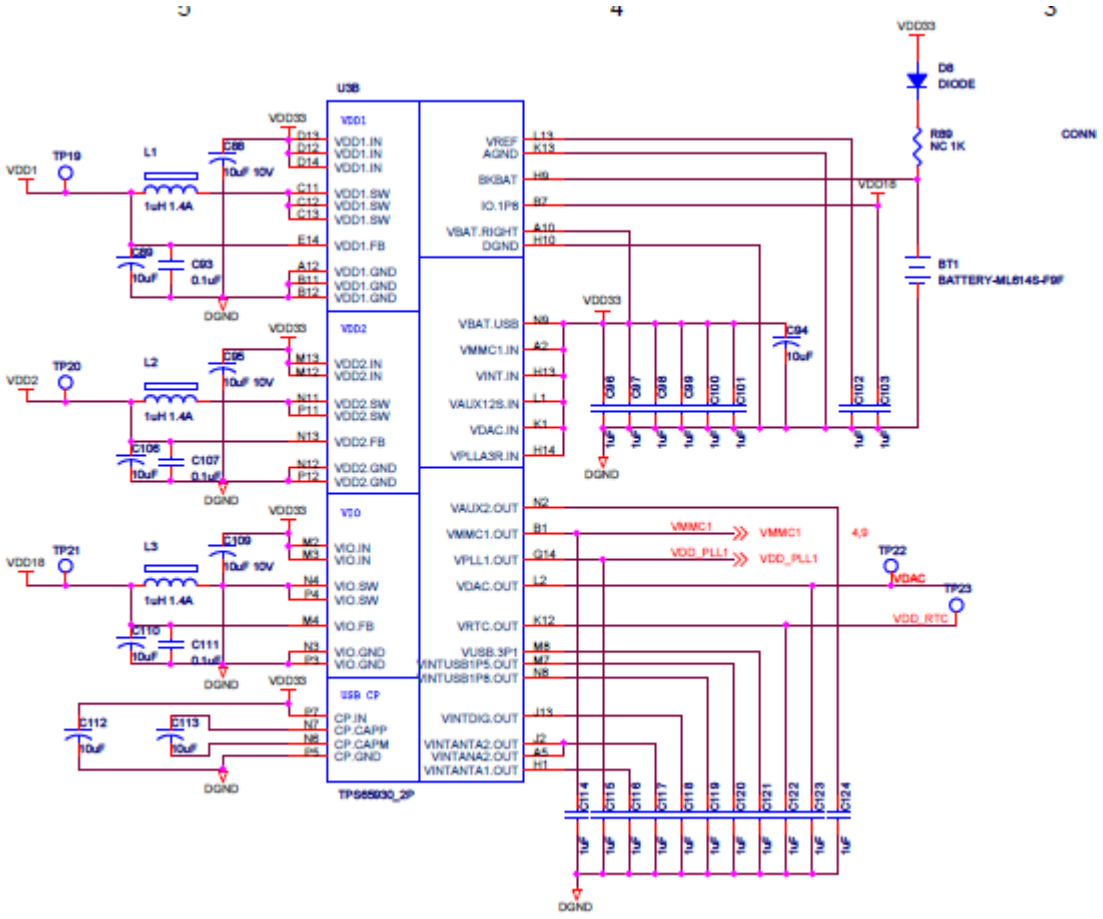

图 3.4: TPS65930 电源模块

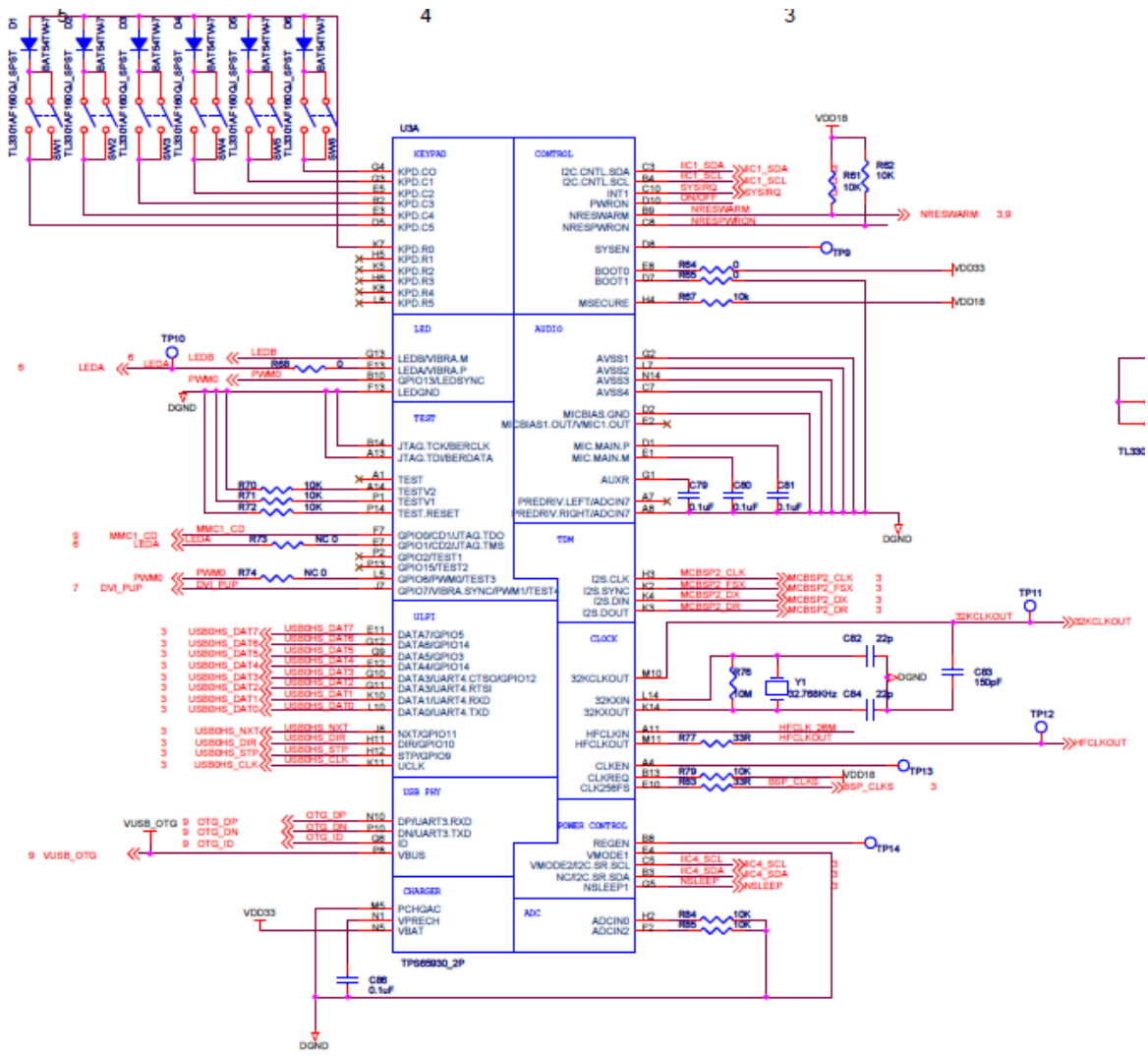

图 3.5: TPS65930 扩展功能模块

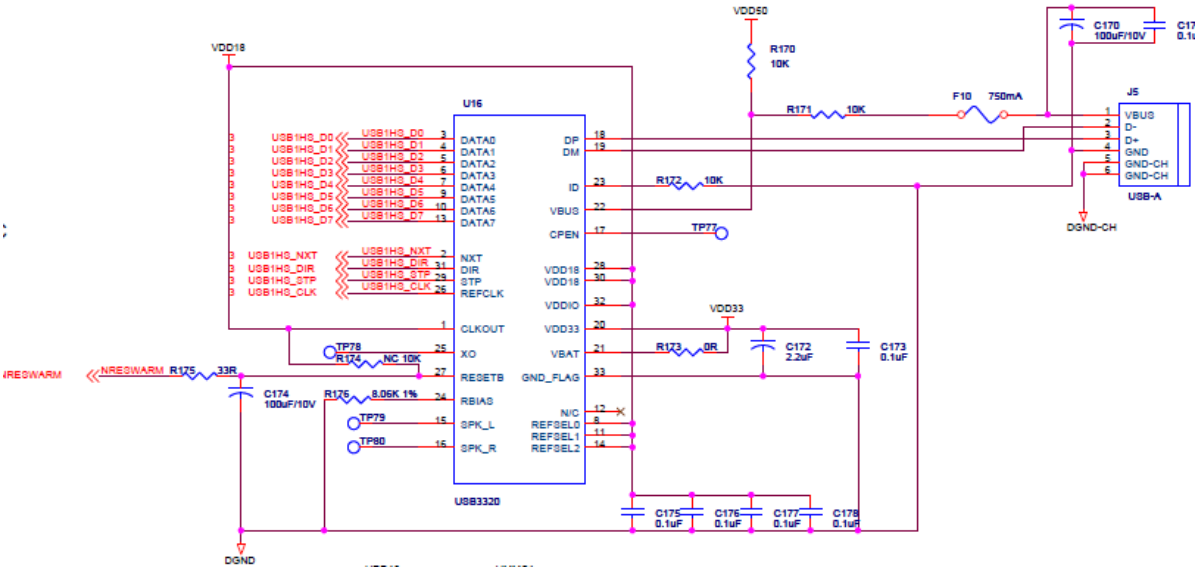

图 3.6: USB 电路原理图

传输率、灵活性以及很好的安全性。TF 卡 (Trans Flash) 是摩托罗拉与 SANDISK 共同开发推出的超小型存储卡，约为 SD 卡的 1/4。TF 卡经过 SD 卡转换器后可以当 SD 卡使用。

SD 卡的数据引脚与 TF 卡的数据引脚通用，本设计中同时支持 TF 卡和 SD 卡，共同的数据引脚都与处理器的 MMCI 部分相连接，在时钟的控制下，输出数据。电路中增加了几个 LED 指示灯，作为信号灯指示作用。SD 卡与 TF 卡的存储模块电路设计原理图如图 3.8。

3.3　小结

本节主要介绍硬件平台设计过程，首先介绍系统硬件平台模块划分，然后对主要几个模块电路的设计过程进行介绍，分别介绍了视频采集模块电路设计、MMCI 存储模块电路、HDMI 模块电路设计、电源模块电路设计、USB 模块电路设计和存储模块电路设计等。

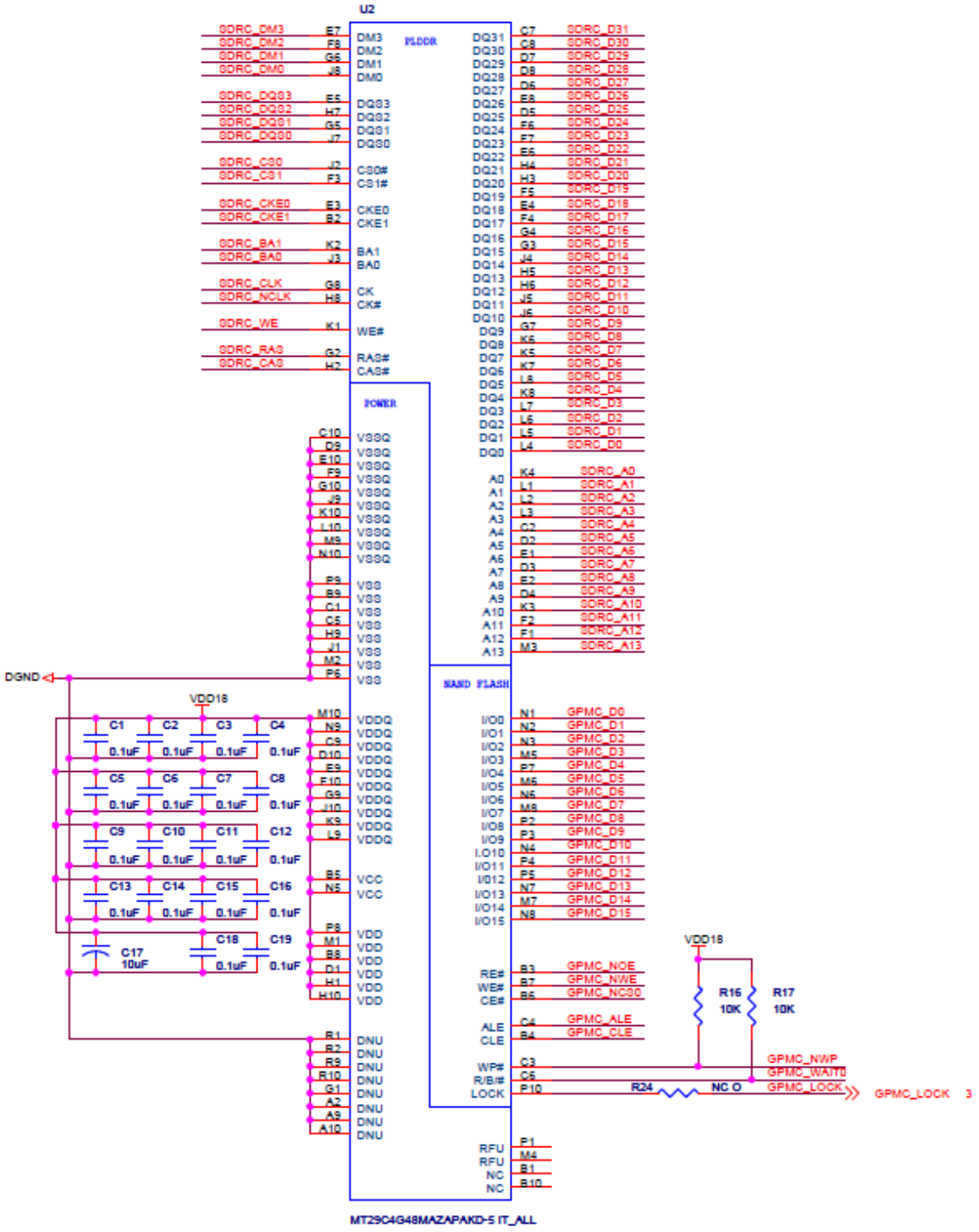

图 3.7: 存储模块设计原理图

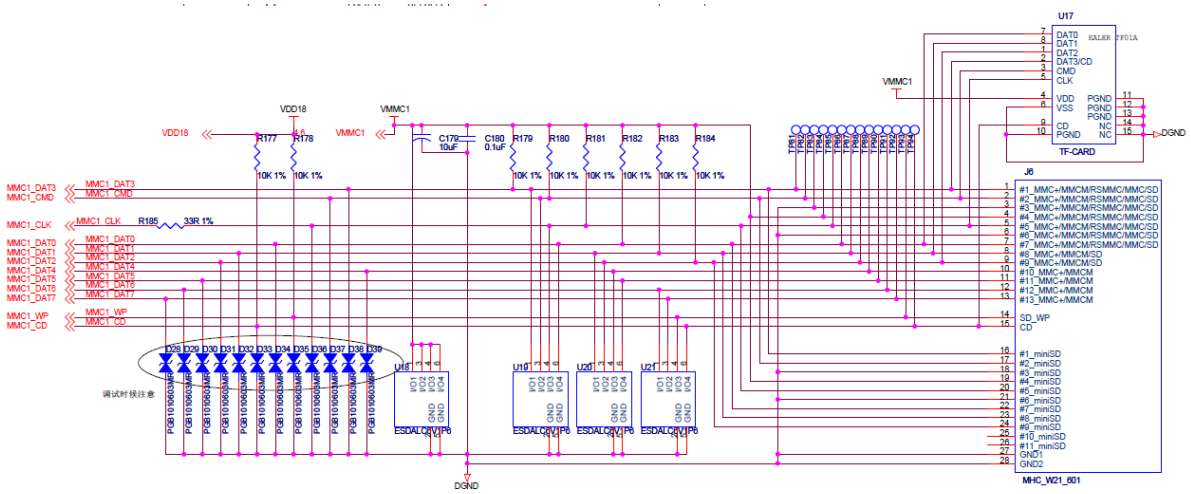

图 3.8: MMCI 存储模块电路设计原理图

第 4 章　嵌入式 Linux 软件开发平台搭建

系统软件开发平台搭建主要分为交叉编译环境建立、嵌入式 Linux 系统内核裁剪与移植、系统引导程序编译与移植、根文件系统制作以及 NFS、Samba 服务器安装与配置。本章将分别对以上内容进行介绍。

4.1　嵌入式 Linux 开发环境搭建

4.1.1　交叉编译环境搭建

建立交叉编译环境是嵌入式 Linux 开发的前提，这是由于嵌入式系统资源有限，程序设计无法直接在嵌入式 Linux 环境下编译，必须将程序在通用计算机上编译生成二进制可执行文件，下载到嵌入式系统中才可以执行。前者通用计算机一般称为宿主机，后者一般称为目标机。编译好的二进制可执行文件一般通过两者之间的连接端口及连接工具如串口、网口、USB 接口及 JTAG 口等进行下载调试。如图 4.1 所示为系统交叉编译环境搭建连接图。

图 4.1: 交叉编译环境连接图

由上图可知，搭建系统软件编译环境由宿主机和目标板以及几种通信接口组成，各个模块介绍如下：

(1) 宿主机开发系统是在 WIN7 主机下的 VMware 虚拟机中安装 Linux 操作系统，该操作系统选用的是 Ubuntu10.04 LTS 版本操作系统。该系统是长期支持版本，性能稳定，适合 dm3730 开发。

(2) 目标板选用的是天漠公司 Devkit8500 系列评估板，系统软件首先在评估板上开发，待硬件制作好以后，将软件系统移植过去。目标版中嵌入式 Linux 操作系统使用的是内核版本。
(3) 由于宿主机是 x86 体统结构而目标板为 ARM 体系结构，在宿主机下编译的程序无法直接在目标板中运行，必须是有交叉编译工具来编译程序，本系统使用的是 arm-eabi-gcc 交叉编译工具。
(4) 串口和网络调试工具使用的是 SecureCRT 调试工具，该调试工具不仅可以作为串口工具使用，设置波特率、数据位、停止位、奇偶校验等，还可以作为网络调试工具，利用 telnet 功能，实现网络调试。

4.1.2 NFS 服务配置

NFS (Network File System) 网络文件系统，最早是由 Sun 发展出来，它的最大功能就是通过网络，让不同的操作系统共享自己的文件，所以也可以将它看做一个文件服务器（File Server），这个文件服务器可以让远端的主机通过网络挂载到本地主机上，共享该文件。本系统中将建立 NFS 共享文件夹，在串口或网络调试工具中输入挂载命令
`mount -t nfs -o nolock 192.168.0.150:/home/nfs /mnt/nfs`
挂载到目标板 `/mnt/nfs` 目录下，那么在目标板中就可以直接调试宿主机中编译好的程序。这种调试方式可以方便系统开发，加快开发进度。NFS 服务器配置方式如下 [24]：

(1) Ubuntu 系统中安装软件相对比较简单，只需要在终端中输入命令 `sudo apt-get install nfs-kernel-server`，系统自动下载安装。
(2) 安装结束后需要配置 `/etc/exports` 文件，该文件为 NFS 共享目录配置文件，输入命令 `sudo vim /etc/exports`，在打开的文件最后输入 `/home/nfs *(rw,sync,no_root_squash)`。`/home/nfs` 是要共享的目录，rw 和 sync 等命令代表对该目录的读写权限，写入方式等配置，以及操作权限设定。
(3) 配置好以后通过下面的命令重启服务。

```
$ sudo /etc/init.d/portmap restart
$ sudo /etc/init.d/nfs-kernel-server restart
```

(4) 目标板上测试 NFS，在调试窗口输入下面命令，查看该文件夹下是否是宿主机目录下的内容，如果有则表示配置成功。

```
$ mount -t nfs -o nolock 192.168.0.150:/home/nfs /mnt/nfs
$ cd /mnt/nfs
$ ls
```

4.1.3　Samba 服务配置

Samba 是 Linux 操作系统与 Windows 操作系统之间架起的一座桥梁，两者之间基于 SMB (Server Message Block) 协议，可以实现互相通行。由于系统软件开发是在 Windows 系统下的虚拟机中开发，而编程环境一般使用 SourceInsight 软件在 Windows 编写，然后在 Linux 环境下进行交叉编译，两者之间的通信就是利用 Samba 服务。需要在 Linux 建立一个共享文件夹。本文系统中在宿主机 `/home/share` 目录下建立共享文件夹。Samba 服务器的建立同 NFS 相似，具体为如下步骤 [25]：

(1) 在 Ubuntu 系统终端中输入命令

```
sudo apt-get install samba
sudo apt-get install smbfs
```

系统自动下载安装。

(2) 创建共享文件夹

```
$ mkdir /home/share
$ chmod 777 /home/share
```

(3) 编辑 Samba 服务配置文件，需要打开 `/etc/samba/smb.conf` 文件，在末尾添加如下信息:

```
path = /home//share
public = yes
writable = yes
valid users = suda
create mask = 0700
directory mask = 0700
force user = nobody
force group = nogroup
available = yes
browseable = yes
```

(4) 重启 Samba 服务

```
$ sudo /etc/init.d/samba restart
```

(5) Samba 测试

在 Windows 系统下，在文件管理器的地址栏中输入 `\\192.168.0.150`，其中 IP 地址为 Ubuntu 系统中的 IP 地址，如果输入命令按回车后，显示 share 共享文件夹，则表明 Samba 服务器配置成功。

4.2　嵌入式 Linux 内核裁剪与移植

系统内核是一个操作系统的灵魂，负责系统的进程调度、内存管理、文件系统及网络系统管理等，可以满足嵌入式系统中绝大多数的复杂性要求，但是由于一般嵌入式 Linux 内核大小有到等，而嵌入式系统硬件资源有限，就必须对内核进行重新裁剪和配置 [26]。内核的裁剪与配置主要针对系统硬件资源和软件设计需求，将内核中不需要的内容删除及重新配置，最后重新编译内核，生成镜像文件，下载到目标板中。

4.2.1　内核的配置

在系统内核编译前需要对系统的内核进行配置，配置时需要根据系统实际需求，认真配置每一项，如果配置不当，直接关系到系统能否正常启动，是否满足系统设计需求等。配置内核除选择必须参数外，还需要将不需要的选项去除，比如视频驱动可能支持很多种设备的驱动，就需要将不需要的驱动去除，以此减少内核空间，但是如果内核里面没有该支持的驱动，就需要添加 [27]。如本文选择的 Linux 操作系统中没有对 MT9P031 图像传感器的驱动支持，就需要内核配置中对其添加。需要在 “defconfig”,“kconfig” 和 “makefile” 等三个文件中添加配置数据。在内核中三个文件的存储路径分别为 `arch/arm/configs/omap3_beagle_defconfig`，`drivers/media/video/Kconfig` 和 `drivers/media/video/Makefile`。

在 `omap3_beagle_defconfig` 配置文件中添加如下配置信息：

```
CONFIG_SOC_CAMERA=y
CONFIG_SOC_CAMERA_MT9P031=y
```

在 `video/Kconfig` 视频驱动配置文件中添加如下配置信息：

```
config SOC_CAMERA_MT9P031
tristate ”mt9p031 support”
depends on SOC_CAMERA && I
help
    This driver supports MT9P031 cameras from Micron.
```

在 `video/Makefile` 编译文件中添加修改如下编译文件信息：

```
obj-$(CONFIG_SOC_CAMERA_MT9P031) += mt9p031.o
```

修改好后，需要对系统内核重新编译。编译通过后，需要配置内核，选择刚才添加的驱动文件，通常配置内核都是通过 `make menuconfig` 命令来进行图形化配置。如图 4.2 所示，在菜单中选择所需要的功能。

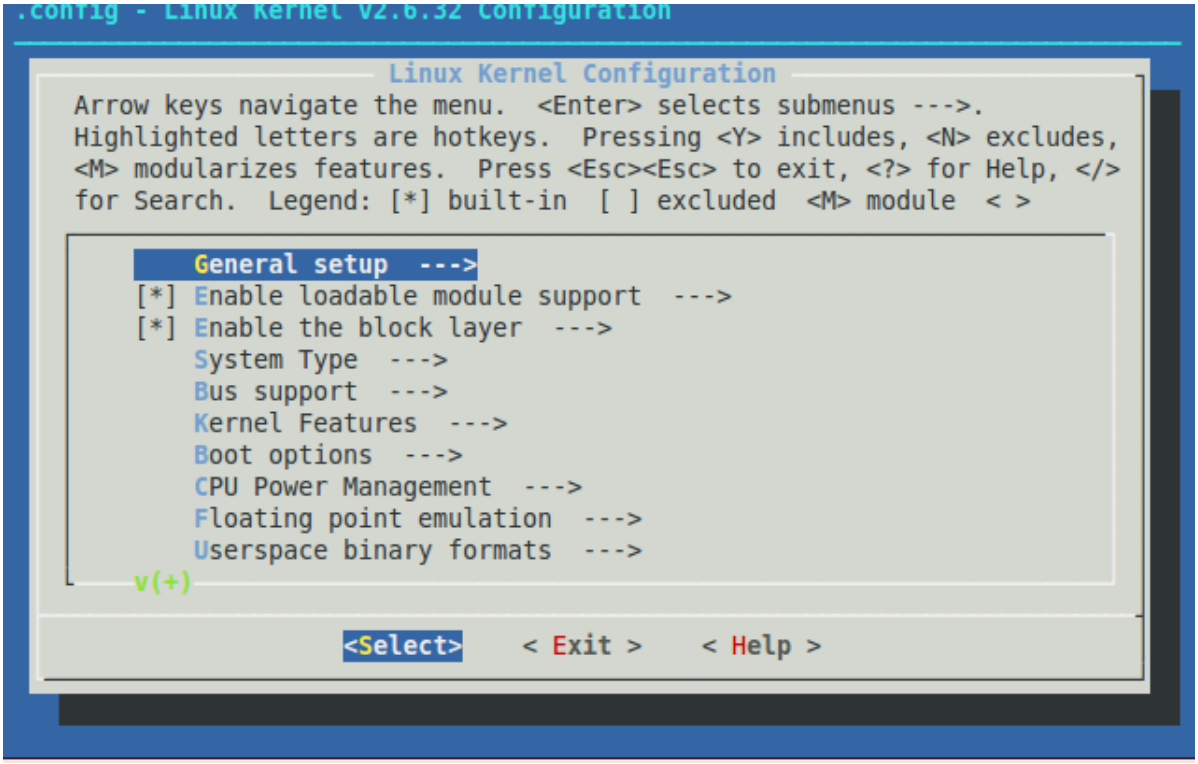

图 4.2: 内核配置图像界面

按“Y”键选中功能，并将该驱动功能编译到内核中，系统启动后将添加该功能；“N”取消该功能，系统编译时内核将不添加该功能；“M”功能为模块化编译功能，系统编译后内核不添加该功能，而是生成 *.ko 文件的动态链接库，当系统启动后如果需要使用该功能，就使用 insmod 命令添加该动态库，这种方式可以减少系统内核空间。配置完成后系统会自动生成 config 配置文件。MT9P031 配置界面如图 4.3 所示，为了使初期调试方便，本系统中对 MT9P031 选择的 M 模式，这样每次修改驱动文件后，就不需要重新编译内核，但在后期成品时需要将该驱动编译到系统内核。

4.2.2　内核的编译与移植

内核配置完成以后，就需要对系统内核进行编译，在编译前可以通过命令 export $PATH 来查看交叉编译工具 arm-none-linux-gcc 是否添加，如果没有，则需要见该工具添加，使用的命令为 export PATH=$PATH:/opt/arm-q2003/ arm-eabi-4.4.0。确认好编译环境后，开始对内核编译。首先使用命令 make distclean，清除以前编译遗留下的文件，然后输入命令 make 进行编译，编译好后将在 arch/arm/boot 目录下生成 uImage 镜像文件。本系统是将该镜像文件拷贝到 TF 卡上，通过 TF 卡实现系统更新。

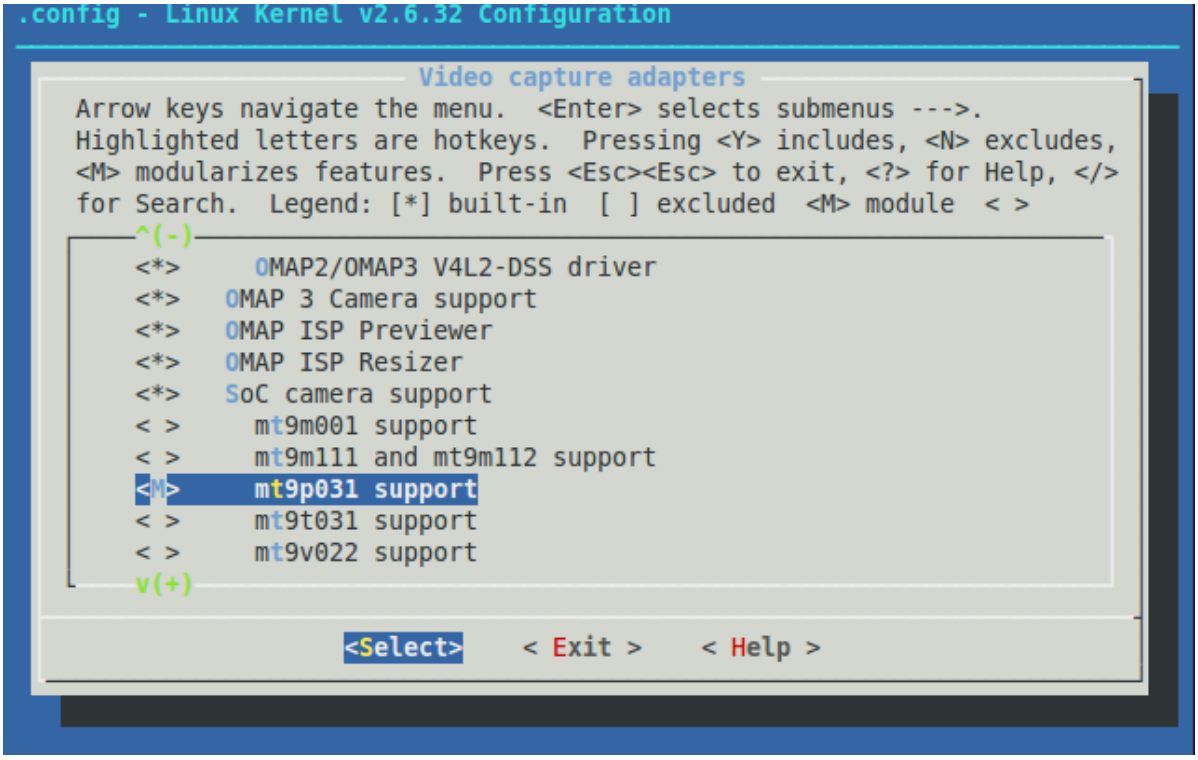

图 4.3: mt9p031 配置界面

4.3　引导加载程序移植

4.3.1　Bootloader 的作用

Bootioader 为启动引导程序，是系统加电以后运行的第一段软件代码，用于完成硬件的基本配置以及引导内核系统的正常启动，通常从 0x00000000 地址开始执行。在嵌入式系统中，通常并没有像 PC 机 BIOS 那样的固件程序，因此整个系统的加载启动任务就完全由 BootLoader 来完成。大部分 Bootloader 工作模式有启动加载模式和下载模式两种 [28]。加载模式是系统将存储在 FLASH 存储器上启动程序自动加载到 RAM 中运行，整个系统启动过程不需要用户参与，产品发布时必须在这种模式下工作。下载模式则是系统上电后通过串口或网口从宿主机上下载文件，下载的文件通常先保存在目标机的 RAM 中，然后再写到目标机的 FLASH 存储器中，同时向用户提供一个命令接口。Bootloader 工作过程一般包含以下步骤 [4]：

(1) 硬件设备初始化。主要是为下一阶段执行做准备，包括准备 RAM 空间；
(2) 复制第二阶段代码到 RAM 空间；
(3) 设置堆栈；
(4) 跳转到第二阶段的 C 程序入口点；
(5) 开始第二阶段，初始化硬件设备；

(6) 系统内存映射检测；
(7) 读取 FLASH 中内核镜像及根文件系统到 RAM 空间；
(8) 设备启动参数及调用内核；

Bootloader 可以自己设计开发，但一般都是选用公用的进行裁剪与修改后使用。比较著名的公用 Bootloader 有三星公司的 vivi、摩托罗拉公司的 dBUG、国产软件 RedBoot 和自由软件 u-boot 等。本文选用 u-boot 作为系统引导加载程序。

4.3.2　u-boot 编译与参数设置

u-boot 是现在比较流行且功能强大的一款 BootLoader，体积小，易于构造。本文 u-boot 使用的是 03.00.02.07 版本。编译步骤为：

(1) 解压 u-boot 压缩包，命令为 `tar xvf u-boot03.00.02.07.tar.bz2`;
(2) `make distclean`;
(3) `make omap3_devkit8500_config`;
(4) `make`;

当系统编译成功后会生成 uboot.bin 文件，可以通过 tftp 下载或拷贝到 TF 卡中，使用 TF 卡实现更新。系统更新后，上电重新启动，出现如图 4.4 界面时，点击任意键实现对 u-boot 参数配置。参数设置包括 IP 地址设定，内核启动参数以及串口波特率设置等，部分参数设置如下：

```
# setenv serverip 192.168.0.150 设置 PC 机 tftp 服务器 IP 地址
# setenv ipaddr 192.168.0.103 设置开发板 IP 地址
# setenv ethaddr 00:10:20:18:ce:05 设置开发板 MAC 地址
# setenv baudrate 9600
# saveenv 保存以上环境参数到 flash
```

```
U-Boot 2010.06-rc1-svn ( 5鏈?06 2011 - 10:05:25)

OMAP34xx/35xx-GP ES2.1, CPU-OPP2 L3-165MHz
OMAP3 Devkit8500 board + LPDDR/NAND
I2C:   ready
DRAM:  512 MiB
NAND:  512 MiB
In:    serial
Out:   serial
Err:   serial
Devkit8500 xM Rev A
Die ID #4f6c00029e38000001683b0610024013
Net:   dm9000
Hit any key to stop autoboot:  0
```

图 4.4: u-boot 配置界面

因为本文系统使用的是 TI OMAP 系列双核处理器，处理的 ARM 核和 DSP 核之间需要互相通信，需要使用 TI 的 DVSDK 开发工具，也需要配置 u-boot 启动参数，主要是对内存的配置，配置命令如下：

```
# setenv bootargs console=ttyS2,115200n8 root=/dev/mmcblk0p2 \
    rootfstype=ext3 rw rootwait mpurate=1000 mem=99M@0x80000000 \
    mem=128M@0x88000000 omapdss.def_disp=lcd \
    omap_vout.vid1_static_alloc=y omapfb.varm=0:3M
# setenv bootcmd 'mmc init; fatload mmc 0 80300000 uImage; bootm 80300000'
# saveenv
```

配置好后，输入 boot 命令，系统进入内核启动过程，开始正常启动。系统正常启动后将进入系统后台运行界面，这时就可以正常对系统进行开发与测试。

4.4　根文件系统制作

4.4.1　根文件系统简介

文件系统是对存储设备上文件操作的一种方法。内核启动后，第一个挂载的文件系统就是根文件系统，内核代码的映像文件都保存在根文件系统当中，如果嵌入式 Linux 系统没有根文件系统将不能正常启动。与 PC 机不同，嵌入式系统一般使用 Flash 作为自己的存储介质，而不同的 Flash 存储器有不同的物理特性，所以支持 Flash 的文件系统有很多，主要有 EXT2、EXT3、JFFS2、CARMFS、UBI 等。本文根据系统的实际需求及硬件配置情况选择了 UBI 文件系统 [29, 30]。

Linux 源代码是以文件的形式存放在根文件系统的各个目录中，根文件系统的结构如图 4.5 所示。

4.4.2　UBI 文件系统制作

首先在宿主机上建立一个文件夹，将所有生成的文件和子目录都放在该文件夹下，然后通过如下命令将文件拷贝到该目录下。

```
$ mkdir /home/share/ubi
$ cp /media/cdrom/linux/tools/mkfs.ubifs /home/share/ubi
$ cp /media/cdrom/linux/tools/ubinize /home/share/ubi
$ cp /media/cdrom/linux/tools/ubinize.cfg /home/share/ubi
```

因为该文件系统支持 OMAP3530 系列的芯片，可以直接进行编译，输入如下：

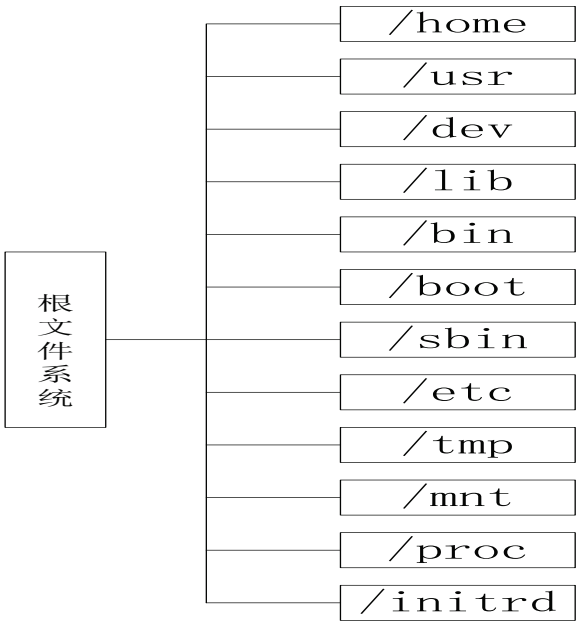

图 4.5: 根文件系统结构

```
$ cd /home/share/ubi
$ sudo /home/share/ubi/mkfs.ubifs -r rootfs -m 2048 -e 129024 -c 1996 -o ubifs.img
$ sudo /home/share/ubi/ubinize -o ubi.img -m 2048 -p 128KiB -s 512 \
  /home/share/ubi/ubinize.cfg
```

执行完以上操作后，在当前目录下会生成所需要的 ubi.img 文件，其中，`-o` 指定输出的 image 文件名 ubi.img，`-e` 表示设定擦除块的大小，`-m` 为页面大小，`-p` 为物理擦除块大小，`-s` 为最小硬件输入输出页面大小。本文文件系统也是通过 TF 卡进行更新。

4.5　小结

本章详细阐述了嵌入式 Linux 系统软件开发平台的构建过程，包括交叉编译环境的建立，以及 Samba、NFS 服务器的配置，接着详细介绍了嵌入式 Linux 系统内核的裁剪与移植和 BootLoader 引导程序的编译与参数配置过程，最后介绍文件系统的制作，并列出了根文件系统的结构。

第 5 章　系统软件设计及算法实现

5.1　视频采集驱动程序设计

由第 2 章分析可知，本文选用 MT9P031 CMOS 图像传感器作为系统视频采集传感器，而该传感器在嵌入式 Linux 内核版本中没有相应驱动支持，因此需要编写视频驱动程序。由于视频采集程序是基于 V4L2 接口来设计，所以视频驱动程序需要在 V4L2 框架下编程。本节主要概述 Linux 设备驱动框架，分类及常用加载方式，以及 MT9P031 CMOS 视频驱动详细设计过程。

5.1.1　驱动设备简介及分类

驱动程序是位于硬件与应用程序之间，实现对硬件进行控制与读取数据，并将数据通过相应的接口提供给应用程序调用。在嵌入式 Linux 操作系统中所有的设备都可以当成文件，所以应用程序通过驱动程序接口，可以对硬件像操作文件的方式进行操作。驱动程序在系统中主要实现以下功能 [31]：

(1) 硬件设备的初始化和内存申请与释放等。

(2) 对硬件设备操作读取数据，并将读取的数据传输到内存缓冲，应用程序可以通过相应接口读取并处理，或者将上层发来的数据传输到硬件，实现对硬件的控制与操作。

(3) 对相应的中断进行检测，并做相应处理。

嵌入式 Linux 系统的设备驱动可以分为块设备、字符设备和网络设备等。块设备是指可寻址、以块为访问单位的设备，有请求缓冲区，支持随机访问而不必按顺序读取数据，一般存储设备都属于块设备 [32]。常见的块设备有各种硬盘，RAM，FLASH 等。字符设备是指能像字节流一样读取数据的设备，不需要请求缓冲区，只能顺序读写。常见的字符设备有串口、鼠标、键盘等。网络设备是比较特殊的设备，它是面向报文而不是面向流，不支持随机访问，没有请求缓冲区。网络设备也叫做网络接口，应用程序是通过 Socket 套接字而不是设备节点来访问网络设备。本系统设计的 MT9P031 驱动属于字符设备，是实现对数据流进行操作。

5.1.2　驱动加载方式

在 Linux 操作系统中将驱动程序加载到内核有两种方式 [32]，分别为静态加载和动态加载，这两种方式的开发过程有些不同，也各有特点。

静态加载方式就是将驱动程序的源代码放到内核源代码中，在内核编译时和内核一同编译，使该驱动成为内核的组成部分，系统开机后会自动加载注册驱动。这种静态连接方式会增加内核的大小，且如果修改驱动，还需要重新编译内核，但在系统发布时一般采用该方式。

动态加载方式是指将驱动程序编译成一个可加载和可卸载的目标模块文件，它可以在内核运行时，再动态加载到内核中。用户可以可以使用 insmod 命令将驱动程序的目标文件加载到内核，在不需要时可以用 rmmod 命令卸载，操作比较方便。在前期开发阶段一般使用这种方式来加载驱动，这样方便系统开发，每次修改驱动后不需要重新编译更新内核，只需要重新编译驱动即可。

5.1.3　驱动程序设计

本系统设计的 MT9P031 图像传感器驱动属于字符设备，如图 5.1 表示该驱动在软件系统中的结构。可以看到该驱动在底层硬件与上层内核和应用程序之间，需要实现与底层硬件和上层应用软件之间通信。

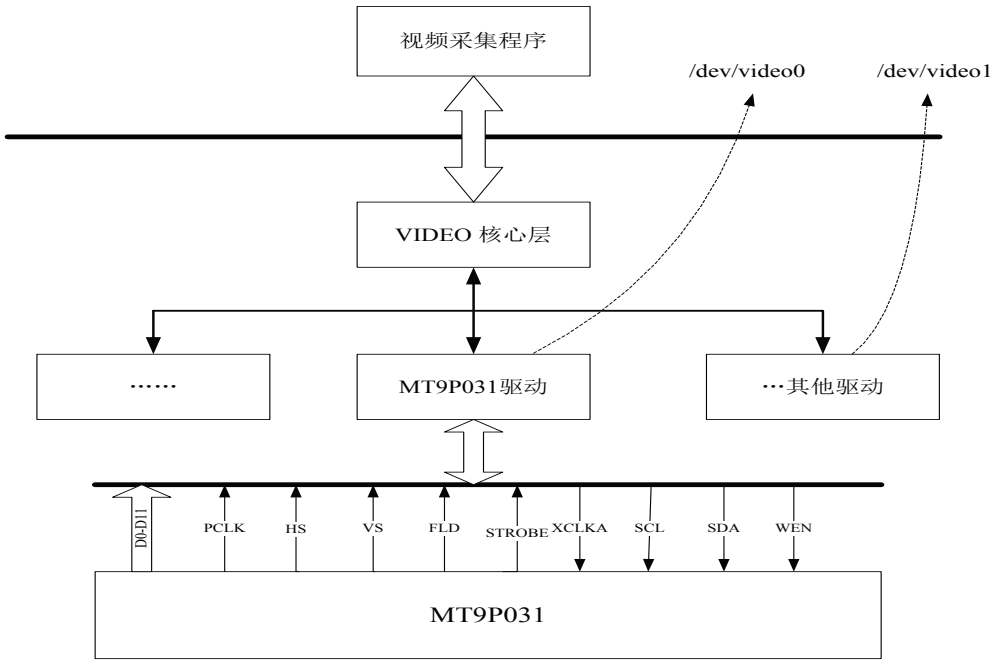

图 5.1: MT9P031 驱动结构图

同一般的视频驱动程序一样，MT9P031 图像传感器的驱动程序分成两部分，分别为传感器驱动和视频功能驱动。传感器驱动为上层硬件接口驱动，实现对硬件初始化

与控制。视频功能驱动为硬件设备本身支持的功能。驱动程序被编译后直接加载到内核或生成动态模块 (mt9p031.ko) 加载，加载注册成功后会生成一个设备节点，通过该设备节点，系统应用程序利用 API 接口来调用设备驱动接口函数，发送命令或者读取实时视频数据。微处理器通过 I2C 总线配置图像传感器的内部寄存器，实现对图像传感器的参数配置和初始化。

由于视频采集程序将使用 Linux 系统为专门视频设备开发提供的 V4L2（video 4 Linux 2）架构，该架构为 Linux 下开发视频设备程序提供了一套接口规范 [20]。这就要求我们的视频驱动程序要符合 V4L2 标准的规范和要求，所以本系统中的视频驱动程序都是依据 V4L2 标准和规范来设计。

5.1.3.1　图像传感器驱动设计

图像传感器驱动设计主要是实现对 MT9P031 芯片的控制，包括对芯片的注册、注销、初始化等操作。处理器的 ISP 接口与 MT9P031 相连，通过 I2C 总线对芯片进行控制。该图像传感器驱动设计主要使用的结构体为 `struct i2c_driver mt9p031_i2c_driver`，结构体的内容如下：

```
static struct i2c_driver mt9p031_i2c_driver = {
    .driver = {
        .name = "mt9p031",
    },
    .probe = mt9p031_probe,
    .remove = mt9p031_remove,
    .id_table = mt9p031_id,
};
```

其中 name 表明该驱动的名称；probe 是驱动程序的探测指针，该函数最重要的操作是通过 `soc_camera_host_register` 注册一个 `struct soc_camera_host`，也就是注册一个设备节点，主要功能是在 CMOS 设备上电启动后，将查找相匹配的驱动程序，如果找到 `mt9p031_probe()` 函数，则完成 mt9p031 图像传感器的初始化和注册；`remove` 函数是卸载函数，当通过 rmmod 等指令卸载驱动程序时，需要调用 `mt9p031_remove()`，该函数主要完成内存释放和资源回收工作；`id_table` 表示该驱动所指的设备类型。

`mt9p031_remove()` 和 `mt9p031_id()` 程序如下：

```
static int mt9p031_remove(struct i2c_client *client)
{
    struct mt9p031_priv *priv = i2c_get_clientdata(client);

    v4l2_int_device_unregister(priv->v4l2_int_device);
```

```
    i2c_set_clientdata(client, NULL);
    mt9p031_sysfs_rm(&client->dev.kobj);

    kfree(priv->v4l2_int_device);
    kfree(priv);
    return 0;
}

static const struct i2c_device_id mt9p031_id[] = {
    { "mt9p031", 0 },
    { }
};
```

当 mt9p031_i2c_driver 结构体中的函数都正确配置后，系统上电就会调用 V4L2 结构体中的 v4l2_device_register() 将设备注册到内核，并产生设备节点 /dev/video0，表明 CMOS 设备注册成功。当卸载该驱动是会调用 V4L2 结构体中的 v4l2_device_deregister() 函数，将其注销，设备节点也就随着消失。

5.1.3.2　视频驱动功能设计

视频驱动功能设计主要是实现对 MT9P031 芯片的 V4L2 的标准操作及 ISP DMA 的管理。在 V4L2 的标准操作中主要使用 V4L2 架构下最重要的结构体 struct video_device，该结构体代表一个视频设备。视频设备在系统中被当成一个文件来操作，包括 open()、release()、write()、read()、mmap()、ioctl() 等，这些文件操作功能都包含在 file_operations 结构体中。设备上电注册成功后，系统应用程序就可以调用这些函数接口。对 ISP DMA 内存的初始化及内存分配，也是通过 ioctl() 中的接口函数实现。

open() 函数的主要功能是通过 inode 中存储的次设备号来查找视频设备，并为设备申请内存、中断号等资源。open() 函数结构为

```
int (*open)(struct inode *inode, struct file *filp);
```

release() 函数主要是做减少引用次数的清理工作，并释放所申请的资源。

read() 和 write() 函数主要实现对视频数据的读写，将每帧视频数据以视频流的方式送到应用程序可以访问的缓冲区，这是传输视频最有效方法，但是在视频数据量比较大时，该方式执行比较缓慢，不能满足实现采集实时视频数据的要求，于是采用了 mmap 映射的方式传输数据。mmap() 函数是将内存空间直接映射到应用程序空间，实现由应用程序直接对视频数据操作，而不需要将视频数据读写到应用程序空间，这种方式会使执行效率会大大提高。

ioctl() 函数是一个接口函数，主要功能是对芯片操作实现视频采集功能。它是 V4L2 架构中重要的接口函数，视频采集程序获取视频数据都是通过 ioctl() 接口函数与

驱动程序进行交互，实现视频数据采集。常用的 ioctl 命令见图 5.2。

常用IO指令	功能
VIDIOC_REQBUFS	分配内存
VIDIOC_REQBUFS	把VIDIOC_REQBUFS分配的数据缓冲转换成物理地址
VIDIOC_QUERYCAP	查询驱动功能
VIDIOC_ENUM_FMT	获取当前驱动支持的视频格式
VIDIOC_S_FMT	设置当前驱动的视频捕获格式
VIDIOC_G_FMT	读取当前驱动的视频捕获格式
VIDIOC_CROPCAP	查询驱动的修剪能力
VIDIOC_QBUF	把数据从缓存中读取出来
VIDIOC_DBUF	把数据放回缓存队列
VIDIOC_STREAMON	开始视频显示函数
VIDIOC_STREAMOFF	结束视频显示函数
VIDIOC_QUERYSTD	检测当前视频设备支持的标准，例如PAL或NTSC

图 5.2: ioctl 常用命令

在驱动程序结构体 `struct mt9p031_ioctl_desc` 中列出了本文设计的部分 ioctl 功能接口函数，该结构体内容如下：

```
static struct v4l2_int_ioctl_desc mt9p031_ioctl_desc[] = {
    { .num = vidioc_int_enum_framesizes_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_enum_framesizes },
    { .num = vidioc_int_enum_frameintervals_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_enum_frameintervals },
    { .num = vidioc_int_s_power_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_s_power },
    { .num = vidioc_int_g_priv_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_g_priv },
    { .num = vidioc_int_g_ifparm_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_g_ifparm },
    { .num = vidioc_int_enum_fmt_cap_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_enum_fmt_cap },
    { .num = vidioc_int_try_fmt_cap_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_try_fmt_cap },
    { .num = vidioc_int_g_fmt_cap_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_g_fmt_cap },
    { .num = vidioc_int_s_fmt_cap_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_s_fmt_cap },
```

```
    { .num = vidioc_int_g_parm_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_g_parm },
    { .num = vidioc_int_s_parm_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_int_s_parm },
    { .num = vidioc_int_g_ctrl_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_g_ctrl },
    { .num = vidioc_int_s_ctrl_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_s_ctrl },
    { .num = vidioc_int_queryctrl_num,
    .func = (v4l2_int_ioctl_func *)mt9p031_v4l2_queryctrl },
};
```

当把驱动程序编译好后，还需要对这两个文件 board-omap3devkit8500.c 和 board-devkit8500-camera.c 进行配置。该配置主要是对 mt9p031 图像传感器的电源、ISP 接口和时钟等参数的配置，以及对 I2C 总线信息添加，其部分函数如下：

mt9p031 传感器 isp 接口和内存等参数配置结构体：

```
static struct omap34xxcam_sensor_config mt9p031_hwc = {
    .sensor_isp = 0,//1
    .capture_mem = MT9P031_BIGGEST_FRAME_BYTE_SIZE * 2,
    .ival_default = { 1, 30 },
};
```

mt9p031 传感器电源，时钟等参数配置结构体：

```
struct mt9p031_platform_data devkit8500_mt9p031_platform_data = {
    .power_set = mt9p031_sensor_power_set,
    .priv_data_set = mt9p031_sensor_set_prv_data,
    .set_xclk = mt9p031_sensor_set_xclk,
};
```

I2C 加载 mt9p031 传感器信息：

```
#if defined(CONFIG_SOC_CAMERA_MT9P031) || \
    defined(CONFIG_SOC_CAMERA_MT9P031_MODULE)
{
I2C_BOARD_INFO("mt9p031", MT9P031_I2C_ADDR),
.platform_data = &devkit8500_mt9p031_platform_data,
},
#endif
```

当上面参数配置好以后，就可以实现将驱动程序编译进内核或者编译成内核模块动态加载到内核，驱动注册成功后会在 /dev 文件下生成 video0 设备节点，并在调试窗口打印出芯片 ID 号：1801，如图 5.3。当然这个时候还无法采集实时视频数据，还需要设计视频采集程序。

```
video_reverse
root@dm37x-evm:/usr/lib# insmod mt9p031.ko
mt9p031 2-005d: mt9p031 chip ID 1801
root@dm37x-evm:/usr/lib#
```

图 5.3: 驱动加载页面

5.2　视频采集与显示程序设计

本节主要介绍在 V4L2 框架下实现实时高清视频采集与显示的程序设计过程，首先对 V4L2 的框架进行介绍，然后分别对视频采集与显示程序设计过程进行分析与介绍。

5.2.1　V4L2 介绍

V4L (Video for Linux)，是针对音视频类设备在 Linux 系统中的一套标准编程接口，V4L2 是 V4L 的升级版本，它修正了 V4L 的一些缺陷，使该结构更加灵活，并于 2002 年左右在 Linux 版本内核中添加了对该功能的支持，后被不断发展，可以支持越来越多的设备。V4L2 是驱动程序和应用程序之间的一个标准接口层，支持对大部分音视频设备的采集与处理，包含 CCD/CMOS 图像传感器 [33]。V4L2 相关的设备及用途见图 5.4。

设备文件	用途
/dev/video	视频捕获接口
/dev/radio	AM/FM音频设备
/dev/vbi	原始VBI数据
/dev/vtx	文字电视广播

图 5.4: V4L2 相关的设备及用途

5.2.2　视频采集程序设计

视频采集程序的设计主要是实现将实时获取的原始视频数据通过 ISP 接口传输到处理器存储单元，并可以实现对视频采集参数的查询与设置，如设置采集视频图像的分辨率，视频格式等。视频采集程序的设计是参照 V4L2 提供的标准接口设计，该规范可以提高程序的可读性和灵活性。

由于 V4L2 中 ioctl 命令都是采用结构化，流程化的方式，所以视频数据的采集也是按照此方式设计，如打开设备、设置获取视频格式、申请缓冲空间、处理数据、开始采集、停止采集、关闭设备等。视频采集程序中对视频数据的读写是采用 mmap 映射方式，而不是 read 和 write 读写方式，这种方式是将缓冲区的地址指针映射到用户空间，应用程序可以直接操作视频数据，而不需要把视频数据读写到用户空间，这种方

式可以大大提高处理数据的速度，对高清视频这样大数据量视频数据处理可以起到事半功倍的效果。视频采集程序的设计流程图见图 5.5。

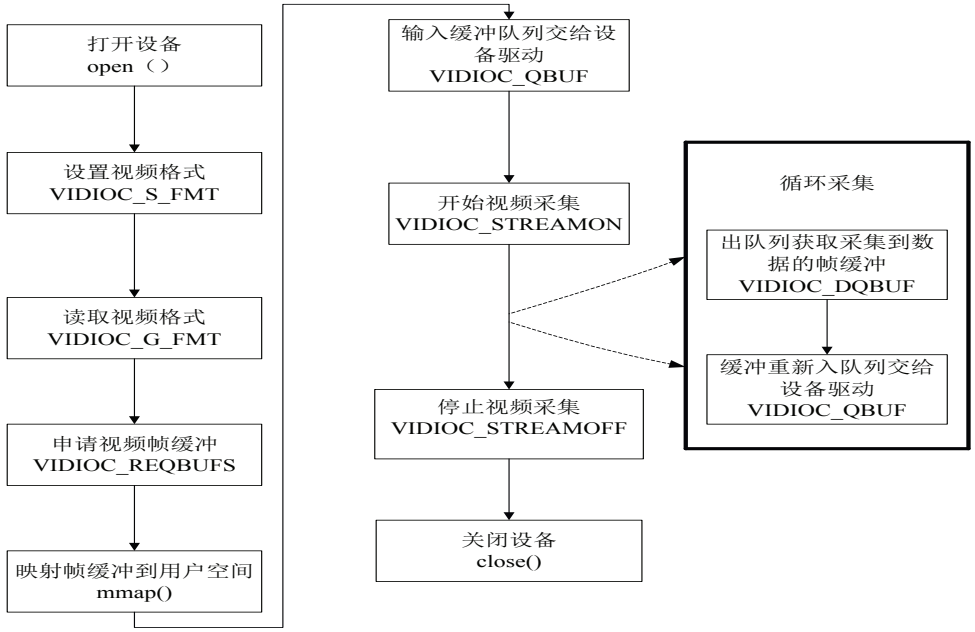

图 5.5: 视频采集流程图

其具体实现的步骤如下：

1. 打开视频采集设备节点文件

```
open((const U8 *) CAPTURE_DEVICE, O_RDWR);
```

2. 确认该设备具有的功能，比如是否具有视频输入，或者音频输入输出等。如果采集的设备不具有采集视频的能力等将会打印出错信息。

```
ioctl(*capture_fd, VIDIOC_QUERYCAP, &capability);
```

3. 设置采集视频的制式及格式等，帧的格式包括宽度和高度等，制式包括 YUV，RGB 等。本文系统采集视频格式设置为 V4L2_PIX_FMT_YUYV，视频格式设置为 1280 × 720。

```
ioctl(*capture_fd, VIDIOC_S_FMT, fmt);
```

4. 向内核申请采集视频数据帧缓冲，本文系统申请的是 3 个缓冲区，一般不超过 5 个。

```
ioctl(*capture_fd, VIDIOC_REQBUFS, &reqbuf);
```

5. 为了可以直接操作采集到的视频数据，且不需要复制视频数据到用户空间，需要将每帧的帧缓冲数据映射到用户空间。

```
mmap(NULL, buf.length, PROT_READ | PROT_WRITE, MAP_SHARED,
     *capture_fd, buf.m.offset);
```

6. 为了便于存放采集到视频数据，需要将所有的帧缓冲入队列。

```
ioctl(*capture_fd, VIDIOC_QBUF, &buf);
```

7. 开始对视频数据进行采集。

```
ioctl(capture_fd, VIDIOC_STREAMON, &a);
```

8. 为了获得采集的原始视频数据，需要将采集数据的帧缓冲出队列。

```
ioctl(capture_fd, VIDIOC_DQBUF, &capture_buf);
```

9. 为了循环采集视频数据，需要将出队列的帧缓冲重新入队列尾。

```
ioctl(capture_fd, VIDIOC_QBUF, &capture_buf);
```

10. 停止对视频数据的采集。

```
ioctl(capture_fd, VIDIOC_STREAMOFF, &a);
```

11. 对视频设备进行关闭，并解除内存映射。

```
close(capture_fd);
```

通过以上步骤可以实现将采集到的视频数据送到申请的缓冲区中，并通过映射方式将获取的数据由用户空间操作控制，并进一步对视频数据进行处理或输出显示。

5.2.3　视频显示程序设计

本系统设计采集的视频信号为高清视频信号，需要通过 HDMI 高清视频接口输出显示，而系统内核支持 HDMI 接口输出，只需要在编译内核时选中 VGA 输出功能，所以本程序的设计只需要将获取的高清视频数据或处理后的视频数据读取到输出缓冲区。一般视频数据输出显示通过 Framebuffer 机制的/dev/fb0 设备节点或者/dev/video1 设备节点。

Framebuffer 被称为帧缓冲，可以直接对其数据缓冲区进行读写操作，与一般字符设备没有什么区别，不必关心物理层显示机制和换页机制等问题。应用程序只需要对这块显示缓冲区写入数据，就相当于实现对屏幕的更新。而/dev/video1 设备节点在本系统中与/dev/fb0 设备节点相似，也是作为一个普通设备节点对其操作，只需要对其显示缓冲区进行数据读写也可以实现数据的更新。由于本系统中 fb0 设备节点需要作为界面显示使用，所以使用 video1 设备节点来显示视频数据。视频显示程序也是通过映射方式将显示缓冲区的地址映射到用户空间，设计的流程同视频采集程序设计流程相似，具体实现的步骤如下：

1. 打开显示 video1 设备节点文件

```
open((const char *) DISPLAY_DEVICE, O_RDWR);
```

2. 取得显示设备的 capability，确认是否支持该视频格式。

```
ioctl(*display_fd, VIDIOC_QUERYCAP, &capability)
```

3. 设置输出视频的制式及格式等，为了与采集的视频格式一致，本文系统输出视频格式设置为 V4L2_PIX_FMT_YUYV，视频格式设置为 1280×720。

```
ioctl(*display_fd, VIDIOC_S_FMT, fmt);
```

4. 向内核申请采集视频数据帧缓冲，本文系统申请的是 3 个缓冲区。

```
ioctl(*display_fd, VIDIOC_REQBUFS, &reqbuf);
```

5. 通过内存映射，将申请的缓冲区映射到用户空间，用户空间就可以直接对缓冲区数据进行操作。

```
mmap(NULL, buf.length, PROT_READ | PROT_WRITE, MAP_SHARED,
     *display _fd, buf.m.offset);
```

6. 开启视频输出显示

```
ioctl(display_fd, VIDIOC_DQBUF, &display_buf);
```

7. 为了获得的处理后的视频数据，需要将采集数据的帧缓冲出队列。

```
ioctl(capture_fd, VIDIOC_DQBUF, & display_buf);
```

8. 读写视频数据

```
cap_ptr = capture_buff_info[capture_buf.index].start;
dis_ptr = display_buff_info[display_buf.index].start;
for (h = 0; h < display_fmt.fmt.pix.height; h++) {
    memcpy(dis_ptr, cap_ptr, display_fmt.fmt.pix.width * 2);
    cap_ptr += capture_fmt.fmt.pix.width * 2;
    dis_ptr += display_fmt.fmt.pix.width * 2;
}
```

9. 为了循环处理后的视频数据，需要将出队列的帧缓冲重新入队列尾。

```
ioctl(capture_fd, VIDIOC_QBUF, & display_buf);
```

10. 关闭视频输出显示接口

```
ioctl(display_fd, VIDIOC_STREAMOFF, &a);
```

11. 关闭设备，并解除内存映射。

```
close(display_fd);
```

按照上述设计的视频采集与显示程序编译后，会生成 mt9p031.o 目标文件，将该文件通过 NFS 或者 U 盘下载到目标板中，输入 `./mt9p031` 命令后就可以实现实时显示 720p/30f 高清视频图像，图 5.6 为采集的视频图像。

图 5.6: 采集的高清视频图像

5.3　视频图像处理算法实现

针对不同的应用需求，需要对视频数据做相对应的处理，本文根据系统的总体设计需求，需要实现以下五种视频处理，“黑白视频”、“底片视频”、“图像增强”、“边缘检测” 和 “人脸跟踪”。

由于系统中视频数据输入的为 RAW 12 bit 数据，经过 ISP 的 VPBE（video process back end）视频后端硬件处理，将输出的视频数据格式转变为 YUV4:2:2 格式，其中 Y 代表亮度信号，U(Cb) 代表蓝色色差，V(Cr) 代表红色色差 [34]。这种格式便于数据的传输与处理，其优点是与黑白图像兼容，对于 YUV 格式像素，只需要取出 Y 分量输出显示，就可以得到黑白图像。本文的彩色视频到黑白视频的转换就是通过此方式，而对于底片视频处理只需要将 YUV 的每个分量值取反，就可以实现。由于这两种视频处理相对比较简单，就不对其进行具体介绍，实现的视频图像如图 5.7 和 5.8，本节主要介绍 “图像增强”、“边缘检测” 和 “人脸跟踪” 等算法分析与实现过程。

图 5.7: 黑白视频

图 5.8: 底片视频

5.3.1　图像增强算法实现

5.3.1.1　图像增强技术介绍

在视频图像采集传输过程中不可避免的会受到各种干扰，导致视频图像无法达到令人满意的效果，为了实现人眼观察或者机器处理需要识别等目的，需要对原始视频图像进行处理，这些都称为图像增强。图像增强包含非常广泛的内容，传统的图像增强的处理方法可以分为基于空域的变换增强算法和基于变换域的图像增强算法两大类。空域是指图像的像素组成集合，该类图像增强主要是直接对图像中像素灰度值进行处理运算，常用的算法有灰度变换、直方图均衡化、直方图修正、图像锐化、图像平滑和噪声去除等；频域图像增强是指对图像进行傅里叶变换后的频谱成分操作，接着再进行逆傅里叶变换获得所需结果，常用的算法有低通滤波、高通滤波、同态滤波以及带阻滤波等。由于图像增强有感兴趣的物体特性，图像增强算法的应用也有针对性，不同应用场合应使用相对应的图像增强算法 [35]。为了满足图像的亮度增强特性以及系统硬件条件等限制，本文系统选择了直方图均衡化算法实现对视频图像亮度进行增强。

5.3.1.2　直方图均衡化算法实现

灰度直方图是数字图象处理中一个最简单、最有用的工具，它描述了一幅图像的灰度级内容，该内容包含了客观的信息，甚至某些类型的图像可以完全由直方图来描述 [36]。图像 $f(x,y)$ 中的某一灰度 f_i 的像素数目 t_i 所占总像素数目 T 的份额为 t_i/T，称为该灰度像素在该图像中出现的概率密度 $p_f(f_i)$。

$$p_f(f_i) = t_i/T \quad i = 0, 1, 2, \ldots N-1 \tag{5.1}$$

其中 N 为灰度级总数目，一般灰度图像灰度级为 0-255，这种纯灰度变化的函数称为该图像的概率密度函数，该函数是梳妆直线，也被称为直方图。一帧灰度视频图像中，其直方图概括了图像中各灰度级的含量，图像的明暗分配状态就可以直接反映处理。为了改善整幅图像或者某些目标的对比度时，可以修改各部分灰度的比例关系实现。本文采用的直方图均衡化是对整幅图像的灰度进行均衡化，如图 5.9 所示，通过点运算使输入的直方图的灰度分布大体均匀，则整幅输出的图像清晰度就会提高。

灰度直方图均衡化算法简单来说就是把直方图的每个灰度级进行归一化处理，求每种灰度的累积分布，得到一个灰度映射表，然后根据相应的灰度值来修正原来图中每个像素 [37]。为了计算方便，需要对直方图归一化处理，即将灰度范围由 0~255 变为 0~1，归一化后的直方图就是概率密度函数 PDF (probability density function)，均衡化就是令概率密度为 1。

设转换前图像的概率密度函数为 $p_r(r)$，转换后图像的概率密度函数为 $P_s(s)$，转换的函数为 $s = f(r)$，由概率论知识得到公式 5.2：

$$P_s(s) = p_r(r) \cdot \frac{\mathrm{d}r}{\mathrm{d}s} \tag{5.2}$$

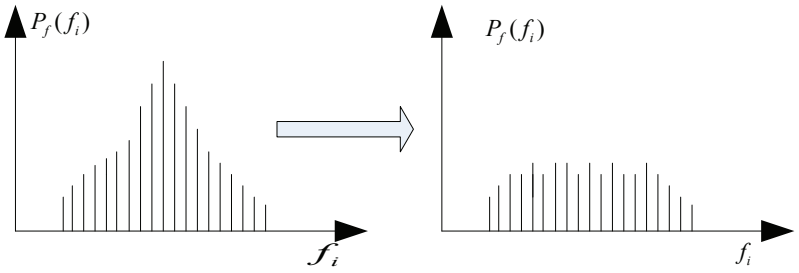

图 5.9: 直方图均衡化

这样，如果使转化后的图像概率密度为 1，则必须满足公式 5.3：

$$p_r(r) = \frac{\mathrm{d}s}{\mathrm{d}r} \tag{5.3}$$

对等式两边积分可得到：

$$s = f(r) = \int_0^r p_r(u)\mathrm{d}u \tag{5.4}$$

公式中的 $f(r)$ 代表 r 的灰度积分变换函数，u 为变量。公式 5.4 也被称为累积分布函数，由 5.1 概率密度函数可得到直方图均衡化公式 5.5：

$$s = f(r_k) = \sum_{j=0}^{k} \frac{t_j}{T} = \sum_{j=0}^{k} P_r(r_k) \tag{5.5}$$

因为 s 是归一化的数值 ($s \in [0, 1]$)，要转换成 0~255 的颜色值，需要再乘上 255。根据上述的直方图均衡化算法原理可知，该处理过程比较简单，其算法实现部分只需要 ARM 处理单元就可以实现。该算法实现的流程图如图 5.10。

由该流程图可知，因为视频图像获取的每帧数据格式为 YUV 格式，所以在处理前先开辟一个缓冲区，将 YUV 格式中 Y 亮度信号读写到新开辟的缓冲区，这样就是首先实现了彩色视频到灰度视频的转换，然后将该缓冲区的视频数据按行读取，并同时进行视频图像的直方图统计后，对该帧所有数据进行均衡化处理，并将处理后的结果送到 V4L2 结构中的输出缓冲区中输出显示。以下为处理的函数结构：

```
convert_to_white(tmpBuffer,tmpBuffer_1, WIDTH, HEIGHT) //灰度转换
video_histogram_statistic(tmpBuffer_1, WIDTH, HEIGHT)   //直方图统计
video_histogram_enhance(WIDTH, HEIGHT)                  //直方图均衡化
```

视频的处理是按每帧进行处理的。图 5.11 为直方图均衡化处理后的视频图像。

5.3.2　边缘检测算法实现

5.3.2.1　边缘检测技术简介

边缘存在于基元与基元、目标与目标等之间，是指那些在灰度空间周围像素有屋顶状或阶跃变化的像素结合，是图像中基本特征之一。因为边缘包含图像中大量信息，

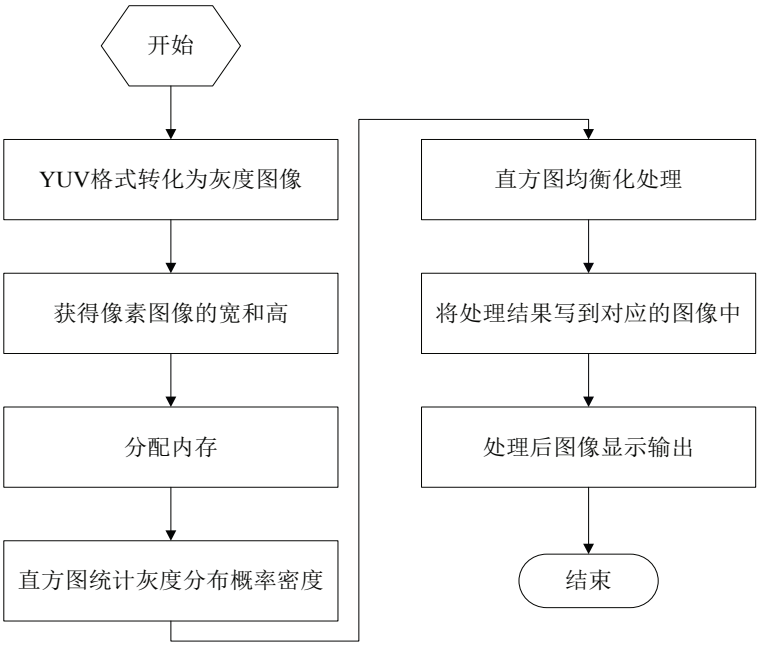

图 5.10: 直方图均衡化算法实现流程图

图 5.11: 直方图处理后视频图像

比如形状，阶跃性质等，它们都是模式识别、机器视觉等研究领域所需求的重要特征。边缘检测对物体的识别非常重要，获取图像边缘，可以使图像分析大大简化，识别也比较容易，而且图像的纹理特性与边缘检测也有密切的关系 [38]。如何快速准确的提取图像边缘也是国内外相关领域的研究热点，也有着比较长的研究历史，最早的边缘检测是二十世纪六十年代 Roberts 提出了基于梯度的边缘检测，这种方法是基于对角方向相邻像素之差来计算梯度进行的，该方法至今都在使用。二十世纪七十年代出现了 Prewitt 算子、Kirsch 算子、Robinson 算子和 Sobel 算子等。Prewitt 算子和 Sobel 算子在计算梯度前会先计算加权平均或邻域平均，再进行微分，该方法可以抑制一些噪声，但处理的图像容易出现边缘模糊。Kirsch 算子可以检测多个方向上的边缘，该方法可以减少因取平均而丢失的细节，但增加了计算量。以上的传统算子，因为是采用局域窗口梯度算子，随着噪声增加，会检测出大量的伪边缘和噪声点，会影响边缘的检测效果。近年来，很多学者提出了基于仿射变换和特征空间的边缘检测、基于小波变换的边缘检测算法、彩色图像和三维图像边缘的检测研究等方法，如今检测算法正在不断的将新方法和新概念引入，同时将现有的方法进行改进，以得到特定任务满意的检测结果 [39]。本系统选择比较常用的 Sobel 算子来实现。

5.3.2.2　边缘检测算法实现

Sobel 算子属于边缘检测局部算子法，一般使用基于方向导数掩模求卷积的方法。在物体边缘处，它的邻域像素灰度级上将形成一个变化带，可以用灰度方向和变化率两个特征来表示。Sobel 算子将对邻域内每个像素的灰度变化率进行量化 [40]。对于数字图像 $f(x,y)$，Sobel 算子的定义如下：设

$$A = |f(i-1,j-1) + 2f(i-1,j) + f(i-1,j+1) - (f(i+1,j-1) + 2f(i+1,j) + f(i+1,j+1))|$$

$$B = |f(i-1,j-1) + 2f(i,j-1) + f(i+1,j-1) - (f(i-1,j+1) + 2f(i,j+1) + f(i+1,j+1))|$$

选择两个值的最大值作为该点的输出位，

$$S(i,j) = \max(A,B)$$

则输出的图像为边缘图像。Sobel 算子也可以用模版表示，模版中的元素表示算式中相应像素的加权因子。模版如图 5.12。

由于这个算法运算量相对于 ARM 内核的处理器速度，运算量不是很大，所以算法在 ARM 端实现，实现过程与图像增强算法实现过程相似，先开辟一个存储空间，将读取的 Y 分量信号进行 sobel 算子算法处理后，输出到显示缓冲区，输出显示。图 5.13 为边缘检测显示图像。

1	0	-1
2	0	-2
1	0	-1

-1	-2	-1
0	0	0
1	2	1

图 5.12: sobel 算子模版

图 5.13: 边缘检测图像

5.3.3　人脸跟踪算法实现

人脸跟踪由人脸检测和人脸跟踪两部分构成。人脸检测就是将视频流中每帧视频中人脸检测出来，并将人脸信息提供给人脸跟踪，而人脸跟踪根据人脸检测提供的人脸信息对每帧视频中人脸进行实时跟踪，该跟踪算法是以前一帧中检测到人脸为基础[41]。

目前人脸检测方法大致可以分为四类 [42]，分别为基于知识的方法、基于局部特征的方法、基于统计学习的方法和基于模板的方法。基于知识的方法是利用一些人脸的先验知识指定一定规则，再利用该规则检测出人脸，比较代表性的算法为 Kotropoulos 和 Pitas 提出的人脸检测算法；基于局部特征的方法是基于人脸的眼、鼻、嘴等不变的特征，在利用这些特征组成候选人脸区域，比较代表性的算法是 Kyoung-Mi Lee 提出检测算法，该方法是利用人脸五官特征不变性的特征。基于统计学习的方法是利用机器学习与统计分析的方法来训练有人脸样本和无人脸样本的特征，然后根据统计特征构建模型来检测人脸，比较代表性的算法时 Viola 提出的基于 Adaboost 方法训练人脸检测分类器的方法。基于模板的方法是根据模版来计算模版与图像之间的相关系数，依照设置好的阈值判断是否有人脸存在。目前有两种类型的模板：变形模板和预定模板，比较有代表性的算法时 Miao 和 Yin 等人提出的一种基于多层次人脸模板的检测方法。

人脸跟踪方法主要分为基于模型和基于运动的两类方法。基于模型的跟踪方法是在跟踪前根据已掌握的人脸知识构造人脸模型，通过该模型对每帧视频数据做匹配处理。基于运动跟踪算法可以分为两种方法，分别为图像差分法和光流法，前一种方法位置精确且速度较快，但受光照影响较大，而第二种方法适合于运动的物体，但精确位置比较差，比较代表性的是 Decarlo 和 Metaxas 提出的算法。

本系统采用的是 Adaboost 人脸检测算法和 CamShift 人脸跟踪算法实现嵌入式人脸智能跟踪。人脸跟踪算法软件流程如图 5.14，其设计流程为系统启动后开始采集实时高清视频图像，如果系统需要对视频作人脸跟踪处理，则首先通过 Adaboost 算法对每帧视频进行人脸检测，如果检测到该帧视频数据中包含人脸，就调用 Camshift 算法对人脸进行实时跟踪，如果没有检测到人脸则继续对每帧视频做 Adaboost 算法进行人脸检测处理。

5.3.3.1　Haar-like 特征和特征值计算

由于人脸面部特定区域的灰度是有差异的，这种差异就可以作为标记人脸的特征，同时为了尽量减小计算量和计算的复杂度，这种人脸特征应当尽可能的简单，所以使用了类 Haar 特征，之后又得到扩展，目前类 Haar 特征主要分为：边缘特征、线特征、中心包围特征等 [43]，如图 5.15 所示。

其中每个特征可按以下公式计算

$$\text{feature}_j = \sum_{i\in(1\ldots N)} w_i \text{Rect}Sum(r_i)$$

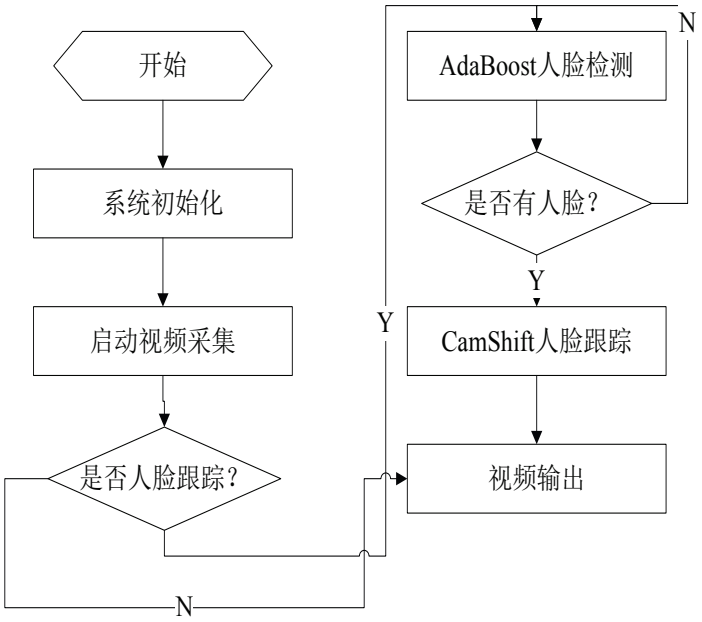

图 5.14: 人脸跟踪系统软件流程图

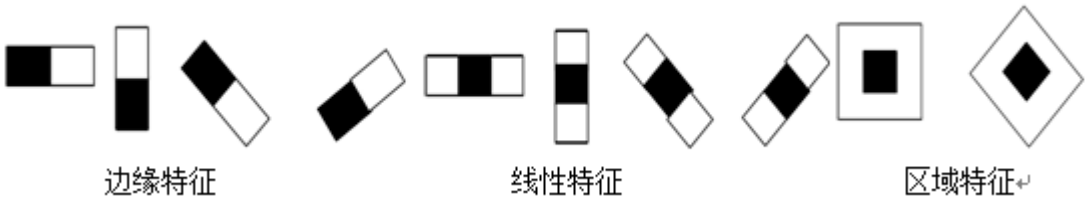

图 5.15: 类 Harr 特征结构图

其中 feature$_j$ 代表的是第 j 种 Harr 块的特征值，w_i 为第 j 种 Harr 块的权值，由于矩阵特征的特征值运算量大，所以采用积分图的方法，该方法可以快速计算出 haar-like 特征值。如图 5.16，已知输入图像 I，在点 (x, y) 处的积分图像值定义如下。

$$\mathrm{ii}(x, y) = \sum_{x' \le x, y' \le y} i(x', y')$$

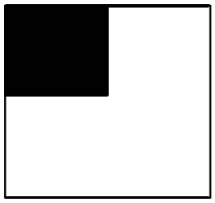

图 5.16: 点 (x, y) 处的积分图像值

对于一幅图像在任意点的积分图值，可以通过对行和列的累加一次循环得到公式。

$$s(x, y) = s(x, y-1) + i(x, y)$$

$$\mathrm{ii}(x, y) = \mathrm{ii}(x-1, y) + s(x, y)$$

5.3.3.2　Adaboost 算法实现

Adaboost 算法的原理是从大量的类 Haar 特征中训练出描述人脸灰度分布的最优特征，并作为弱分类器使用，多个弱分类器优化组合成强分类器，最后通过级联形成层级级联分类器。训练是由一个个有人脸和无人脸训练样本构成，在图像检测过程中，图像通过层层分类器的检测，全部通过检测区域即为目标区域 [44]。弱分类器构造如下：

对于每个特征 j，构造一个判决函数 $h_j(x)$，表示该分类器在该特征下的判决输出，如下式所示。

$$h_j(x) = \begin{cases} 1 & p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

其中 h_j 为第 j 轮训练得到弱分类器，表示该样本要不是真样本，要不就是假样本。$f_j(x)$ 为类 Harr 特征的特征值，θ_j 为特征值的阈值，p_j 为不等式的方向。

训练目标是通过对判断得出的真假样本进行分析，选择分类错误率最低的 T 个弱分类器，最终优化组合成一个强分类器，其训练步骤如下：

(1) 对训练样本中的图像，用 $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ 表示，其中 $y_j = 0, 1$ 分别表示样本中非人脸和人脸样本，并分别对权值初始化为 $w_{1,j} = \frac{1}{2m}, \frac{1}{2l}$，其中 m，l 分别表示非人脸和人脸样本在训练样本中的数目。

(2) 开始循环迭代：`For i = 1, 2, ... N`（N 为循环次数）

利用 AdaBoost 算法获得弱分类器，并根据下式判断强分类器输出的正确率

$$h(x) = \begin{cases} 1 & \sum_{i=1}^{T} a_i h_i(x) \geq 0.5 \sum_{i=1}^{T} a_i \\ 0 & \text{otherwise} \end{cases}$$

若正确率大于某一阈值并小于某一阈值时，则可以作为一个分类器。

(3) 进入下一个分类器的训练

以此迭代循环，最终得到多级强分类器。

5.3.3.3　Camshift 算法实现

CamShift 算法的核心是 MeanShift 迭代，是一种连续自适应移动算法。其原理是在 HSV 格式视频数据中对目标在 H 通道的值进行采样，并算出该目标的颜色直方图，即目标颜色查找表。通过该表可以获得目标像素在输入像素中的概率，然后利用 MeanShift 迭代求出二维概率密度的局部最大值，计算出目标质心坐标，并且自适应调整搜索窗口的大小 [42]。在计算质心位置的同时，Camshift 为得到目标的长轴、短轴、方位，需要计算阶矩来获取二维空间的概率分布方向，计算二阶矩公式如下：

$$M_{02} = \sum_x \sum_y y^2 I(x, y)$$

$$M_{20} = \sum_x \sum_y x^2 I(x, y)$$

$$M_{11} = \sum_x \sum_y xy I(x, y)$$

$I(x, y)$ 表示的是图像在点 (x, y) 处的像素值，x 和 y 表示搜索窗口的变化范围。图像中跟踪目标的短轴与长轴可通过下面两式得到。

$$l = \sqrt{[a + c + \sqrt{b^2 + (a - c)^2}]/2}$$

$$w = \sqrt{[a + c - \sqrt{b^2 + (a - c)^2}]/2}$$

其中 $a = Z_{20}/Z_{00} - x_c^2$, $b = 2(Z_{20}/Z_{00} - x_c y_c)$, $c = Z_{20}/Z_{00} - y_c^2$, $\theta = \frac{1}{2}\tan^2(\frac{b}{a-c})$.

上述步骤可以得到每帧视频图像中椭圆参数和最优目标的匹配中心，利用这些参数可以对输出的每帧视频中的该区域进行描绘，这就实现了对目标的持续跟踪。

5.3.3.4　算法的移植与实现

利用 AdaBoost 人脸检测算法和 CamShift 人脸跟踪算法实现人脸自动检测跟踪的设计流程图 5.17，首先从读取的视频流中获取一帧数据，利用 AdaBoost 算法检测该帧数据中是否有人脸，如果没有人脸，则读取下一帧数据检测，如果读取到的数据中包

含人脸，则利用这次检测到的人脸窗口作为搜索窗口，并计算该窗口的直方图和面积，并读取下一帧数据，利用 CamShift 算法对人脸进行跟踪，如果该搜索窗口大于 S/3 小于 5S/3 则继续使用 CamShift 算法对人脸实时跟踪，如果搜索窗口不在这个范围内，则重新读取一帧数据，再次使用 AdaBoost 算法进行人脸检测，以此循环。

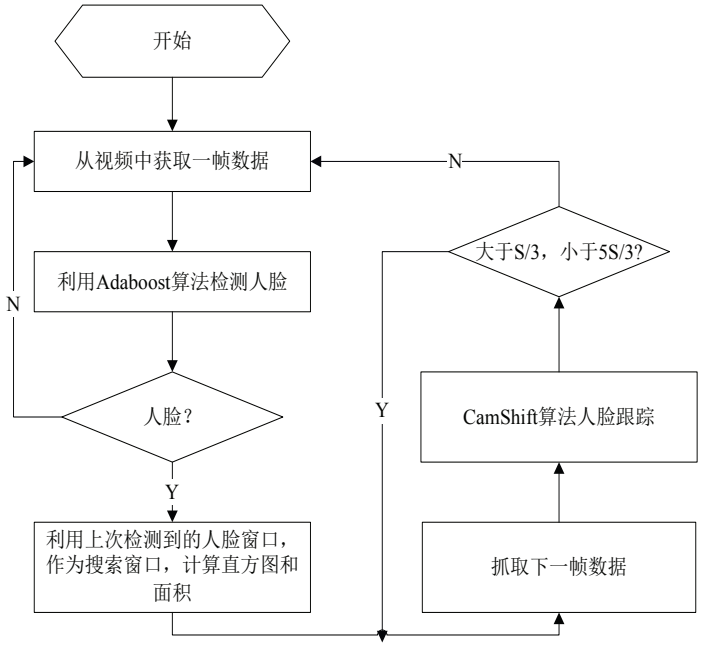

图 5.17: 人脸检测与跟踪算法设计流程图

根据系统设计要求，基于 Linux 操作系统，系统的高清视频采集与显示由处理器的 ARM 内核单元实现，而人脸跟踪等算法则由处理器的 DSP 内核单元实现，两个内核之间通过 TI 提供的 Codec 实现通信和交互。算法的实现主要分为三个步骤：首先将算法在 PC 机下由 C 语言来实现并测试检测效果，然后在 CCS 编译环境下将编译好的算法程序移植到处理器的内核单元并做优化包括浮点运算等，最后将该算法按照 TI 的 xDM 标准进行封装，由处理器 ARM 内核单元的应用程序通过封装的接口实现对算法的调用 [45]。

系统的软件设计基于高内聚低耦合，模块化的设计原则来设计，Adaboost 算法和 Camshift 算法主要由以下一些函数功能模块实现：

```
typedef struct HaarFeature          //定义特征结构
int GetHaarValue(int i, int j)  //计算第 j 个样本的第 i 个 Harr 特征值
void GetAllFeatureValue( ) //保存所有特征值
int GetWeakClassifier( ) //获取弱分类器
//对有人脸的样本库训练
void TrainFace(double t, double f , double d , int m , int n)
//对所有没有人脸的样本库训练
void TrainNoFace(double t , double f , double d , int m , int n)
```

```
int AdaBoost(int stage, int *stream) // adaboost 检测
void GetTarget( int x, int y, int flags, void* param )//选择跟踪目标区域
int camshift() //camshift 算法跟踪人脸
```

算法在 PC 机上实现以后，需要移植到处理器 DSP 内核，由于 PC 端支持浮点运算，而 DSP 内核不支持浮点运算，需要对算法进行优化处理，将算法程序中用到的浮点运算地方如分类阈值，人脸检测窗口等转化为 DSP 上的定点运算。算法优化好后，需要对函数进行封装。图 5.18 为在 PC 机上测试的部分人脸跟踪效果图像。

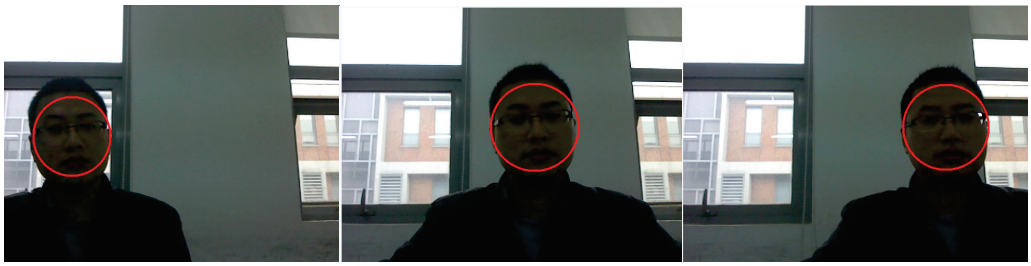

图 5.18: PC 端部分测试结果图

由于时间比较紧张，人脸检测与跟踪算法目前只是在 PC 机上实现，在嵌入式设备移植部分工作还有待继续研究，由于采集的 720P 分辨率的视频数据太大，利用 DSP 处理算法时运算耗时很大，可能人脸跟踪实时性不强。所以事先考虑采取在处理数据时降低分辨率的办法，如果需要人脸检测时，首先分配个空间，对采集的每帧图像作降维处理，并存入该空间，送到 DSP 核进行检测跟踪处理，并对处理后的结果标记后，再由 ARM 核标记高清视频数据的位置，从而实现人脸的跟踪，该方法有待继续研究。

5.4　小结

本章主要介绍了系统的视频驱动程序、视频采集与显示程序及视频图像处理的设计与实现过程，首先详细介绍了视频的驱动结构以及加载方式等，接着介绍 CMOS 芯片视频驱动设计过程，驱动程序设计加载成功后，接着介绍基于 V4L2 结构的视频采集与显示程序设计过程，在视频图像成功采集与显示后，最后详细介绍了图像增强、边缘检测及人脸跟踪算法的实现过程。

第 6 章　Qt/E 界面设计及 OSD 功能实现

在本文的第二章已经对现有的嵌入式 GUI 图像界面系统进行了分析与比较，最终选择了 Qt/E 作为本文系统界面设计的图形系统，本章将分别介绍 Qt/E 图像系统，界面的设计与实现过程以及 OSD 功能的实现。

6.1　Qt/E 介绍

Qt 原是挪威 TrollTech 公司开发的一个用户界面框架和跨平台应用程序，后被诺基亚收购。在被诺基亚公司收购前，TrollTech 公司针对嵌入式环境开发了一套 Qtopia 产品，但是被收购后，在其底层摒弃了 X Window Server 和 X Library，于 2001 年推出的嵌入式系统的 Qt Embedded 版本，即 Qt/E 版本，该版本 Qt 运行速度不够快的缺点大力优化底层，大大提高了 Qt/E 的性能。Qt/E 与 Qt 一样，都是采用 C++ 语言编写 [46]。Qt/E 系统架构如图 6.1 所示。

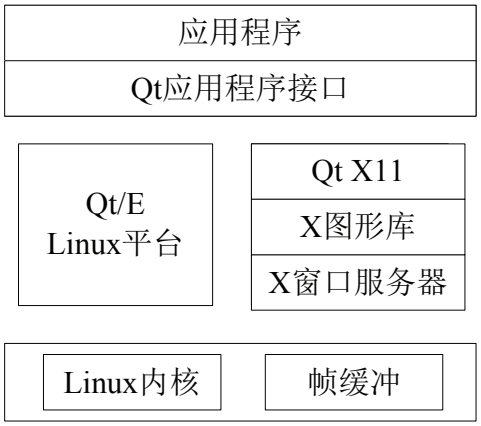

图 6.1: Qt/E 系统架构

总体来说 Qt/E 具有如下一些特点：

1. 跨平台、可移植性好

   由于 Qt 与 Qt/E 上层开发接口相同，不涉及底层操作，Qt 程序只需要通过交叉编译，便可以在嵌入式环境下运行，提高系统的移植与开发速度。

2. 丰富的 API 接口

由于 Qt/E 拥有和 Qt 一样的 API 接口，而其类库是采用 C++ 封装，可以很方便的对其扩展。开发者只需了解 Qt 的 API，不必关心嵌入式系统具体应用平台，使用十分方便。

3. 强大的开发工具
   Qt 提供了 Kdevelop、Qt Designer 和 Qt Creator 等开发工具。
4. 拥有自己的图形库
   Qt/E 拥有自己的图形引擎，而不需要任何额外的图形库，所以可以直接底层的 FrameBuffer 进行操作。
5. 支持不同的输入设备
   Qt/E 对输入设备进行抽象，屏蔽不同输入设备，支持将外部事件全部抽象为内部定义的输入设备事件，支持常用的键盘、鼠标等。

6.2　Qt/E 界面设计与实现

6.2.0.1　Qt/E 界面设计

为了方便用户使用不同的视频处理功能，需要设计一个人机交互界面，而由于该界面需要实现 OSD 功能，所以要求设计的界面简洁明了。在本系统中主要是切换不同的视频处理功能，所以该界面只需要设计五个交互按钮即可，分别为“黑白视频”，“边缘检测”，“图像增强”，“底片视频”，“人脸跟踪”。通过点击不同的按钮实现对视频图像的不同处理。

Qt/E 的应用程序的不同于其他 C++ 的设计，是采用一种比较灵活的信号/槽机制，但也是遵循面向对象的设计方法。该机制主要应用于对象之间的通信，是 Qt/E 编程的核心机制，能携带任意数量和任意类型的参数。信号与槽并不是一一对应的，一个信号可以和多个槽相连，而一个槽也可以和多个信号相连，当某个对象状态发生变化，所有与该信号相关联的的槽就会被调用 [47]。

本系统界面中按钮之间的通信就是通过信号与槽的机制设计实现，首先在 `main()` 函数中建立 `QApplication` 对象，该对象能够控制图形界面。程序通过调用该对象的 `exex()` 函数开始处理时间，直到收到 `exit()` 或 `quit()` 函数信号结束。本文系统的程序设计是采用一个进程多个线程的方式实现的。主函数的代码如下：

```
HD_VIDEO_CAPTURE_SYSTEM::HD_VIDEO_CAPTURE_SYSTEM(QWidget ent) :
HD_VIDEO_CAPTURE_SYSTEM::~HD_VIDEO_CAPTURE_SYSTEM()
void HD_VIDEO_CAPTURE_SYSTEM::changeEvent(QEvent *e)
int main(int argc, char *argv[])
{
    QWSServer::setBackground(QColor(255,255,255,255));
    QApplication a(argc, argv);
```

```
    HD_VIDEO_CAPTURE_SYSTEM w;
    w.show();
    return a.exec();
}
```

界面通过 QT_CREATER 中 UI 来设计，屏幕设计的尺寸为 1280 × 720，在 UI 中实现对按键的布局，因为需要实现视频叠加，所以 qt 的背景需要设置成透明的，需要在背景模式中在 style_sheet 属性中添加 background-color:transparent，同样对按键背景颜色的设置同上。为了实现按键的通信与交互，需要设置按键的属性，右击在 “go to slot” 中选择 “click”。按钮的控制采集程序执行和关闭时采用 QProcess 类方式，该类可以用来启动外部程序并与之同行。在应用程序中需要在头文件中添加 #include <QProcess> 头文件，按钮控制代码如下：

```
void HD_VIDEO_CAPTURE_SYSTEM::on_pushButton_clicked()
{
   myProcess_2->close();      //关闭所有进程
   myProcess_3->close();
   myProcess_4->close();
   myProcess_5->close();
   QString program = "./video_original";  //调用视频采集程序
   myProcess_1->start(program);
}
```

通过以上的设计，通过编译就可以在 PC 机下生成二进制可执行程序，运行该程序可以在 PC 机下预览执行的效果，但是该可执行程序并不能够在嵌入式系统下运行，需要对该程序进行重新编译。

6.2.1　Qt/E 编译与移植

由于在 PC 机下生成的二进制可执行程序并不能够直接在本系统的嵌入式 Linux 操作系统平台下执行，需要重新用嵌入式环境编译工具对其编译。在本文中由于选用的 TI 处理器，该公司提供 DVSDK 工具，里面包含编译所需要的所有的编译环境变量的工具配置，只需要在终端中输入命令 sudo source /dvsdk/linux-devkit/environment-setup，系统将进入 [linux-devkit]:~> 编译环境，此时需要对之前编译好的 qt 程序重新编译，在当前目录下输入 qmake-project，qmake，make 命令，即可生成在嵌入式环境可运行的二进制程序。可以通过串口、网口、U 盘等方式下载到嵌入式系统中。图 6.2 为基于 Qt/E 设计的人机交互界面。

系统界面的操作是通过鼠标控制的，由于系统支持对鼠标的驱动，所以只需要对鼠标的控制环境变量进行添加，在/etc/profile 文件中添加 export QWS_MOUSE_PROTO=MouseMan:/dev/input/mice 环境变量命令即可。

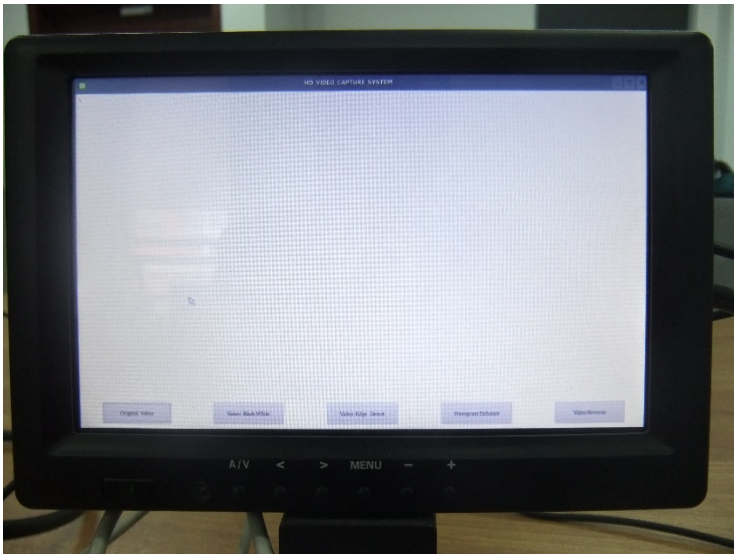

图 6.2: 人机交互界面

6.3　OSD 功能的设计与实现

OSD（on screen display），表示一种在活动视频上叠加图形信息的技术，在日常生活中比如相机、电视等都比较常见。通常视频和 OSD 信息是分开的，在视频输出时，才将两个通过一定的技术叠加在一起，同时显示输出，可以实现人机交互的目的 [48]。在 linux 系统下视频和图像是分层显示。在 DM3730 处理器上，本系统共分为 3 层，Vid1、Vid2、fb，这三层都可以显示视频，但 qt 界面是显示在 fb 层，所以系统设置为视频显示在 Vid0，qt 界面显示在 fb 层。在正常模式下，DM3730 处理器默认的显示顺序如图 6.3。

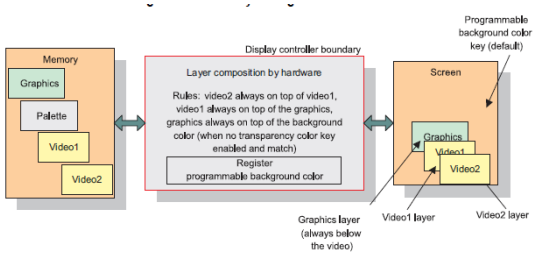

图 6.3: 正常模式

从图中可以看出，Vid2 在最上层，而图形层在最下层，这样如果将视频与图像叠加的话，视频将覆盖图形层。这时就需要利用 DM3730 处理器提供的另外种透明模式（Alpha Mode），其显示顺序如图 6.4。

从图中可以发现，在透明模式下，图形层位于视频层上面，这样只需要把 qt 界面背景及 framebuffer 都设置成透明既可以实现 OSD 功能，但是通过此种技术有个缺点，当 QT 界面设置为全透明时，就只显示视频界面，QT 界面就看不清楚，当设置为半透

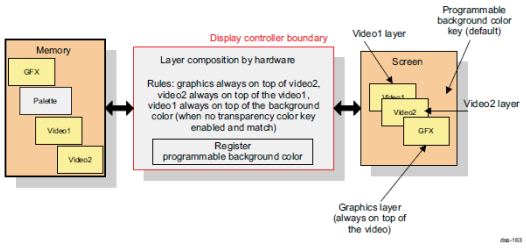

图 6.4: 透明模式

明模式时，视频和界面都可以显示，而视频就会被 Qt 界面的阴影遮挡，视频图像显示不清楚。这个现象主要是由于设计的界面为 1280 × 720 大小的界面，与视频输出的分辨率大小正好相同。为了更好的解决这个问题，可以设计小些的界面，只输出按钮那部分，只需按钮那部分区域显示半透明，或者采用另外一种 Color keying 颜色替换技术。本文系统采用的是 Color keying 技术，其实现原理如图 6.5。

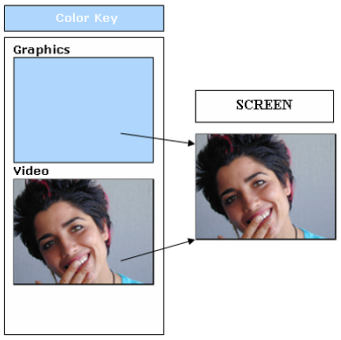

图 6.5: 颜色替换技术原理图

从图上可以看出，Transparent Key 颜色替换技术，即通过对图形层相应颜色替换成视频的合成显示技术，从而达到合成显示的效果。在本系统中实现的机理是，将 qt 背景颜色设置成透明模式，同时对 Framebuff 的颜色设置成白色，在透明模式下将 Framebuff 的白色替换成视频，那么就可以实现视频与界面的叠加显示。系统实现方法为在 Linux 系统中输入以下命令：

`echo 1 > /sys/devices/platform/omapdss/Trans_key_enable` 命令，该命令将颜色替换 `Trans_key_enable` 功能选中，启动 Qt/E 界面可执行程序是就会显示出 OSD 效果。通过此技术可以很好地解决全透明模式下存在的不足。图 6.6 为实际 OSD 效果图。

6.4　小结

本节主要介绍 Qt/E 界面的设计过程及 OSD 功能的实现过程，首先对 Qt/E 进行了详细介绍包括其发展历史及优点，接着介绍本系统需要设计具体界面的要求，程序设

图 6.6: OSD 实际效果图

计和界面的编译移植，最后介绍了对 Qt/E 界面的 OSD 功能实现过程。

第 7 章　结论与展望

7.1　总结

嵌入式高清视频采集与处理系统是现在以及未来在安防监控、医疗卫生、视频通信等领域发展的重点，本文在此背景下设计的具有智能图像处理功能的嵌入式高清视频采集与处理系统，能够满足系统设计的低功耗，高分辨率，高速度传输等要求，同时本系统也可以很好的实现人机交互。系统现在采集的视频数据能够达到 720p/30f 的要求，理论上可以达到 720p/60f。在系统应用方面，实现对视频图像的增强、边缘检测、底片视频等处理。系统拥有广泛应用前景。

在系统开发过程中，涉及到操作系统移植、视频驱动开发、采集程序设计、算法实现以及界面开发等内容，现对本文的主要工作总结如下：

1. 通过对视频采集系统的总体性能分析，设计系统总体设计框架，并分别对系统的硬件及软件框架进行分析与设计。在硬件框架设计方面对系统的性能指标、模块划分以及主要元器件的选型进行分析与设计；在软件框架设计方面对操作系统、系统工作流程、界面设计等方面进行分析与设计。
2. 在硬件框架设计基础上，对系统的硬件按模块划分进行原理图设计。
3. 搭建系统软件开发平台，包括交叉编译环境的搭建，NFS 和 Samba 服务器的配置，嵌入式 Linux 内核的裁剪与移植，BootLoader 的编译与移植和文件系统的制作与移植。
4. 由于选择的嵌入式 Linux 内核版本中没有对 MT9P031 图像传感器的驱动支持，所以需要对该芯片的驱动程序进行设计，本文在分析驱动结构基础上，并在 V4L2 架构下，完成视频驱动程序的编程。
5. 高清视频采集与显示程序设计以及图像处理算法的实现。依照 V4L2 架构设计视频采集与显示程序，并在视频处理方面，实现对黑白视频、图像增强、边缘检测、底片视频等算法的实现。
6. 设计人机交互界面以及界面的 OSD 功能实现。人机交互界面是利用 Qt/E 图形系统进行编程设计，并根据系统提供的硬件功能，实现界面的 OSD 功能。

7.2　展望

本课题研究设计的高清视频采集与处理系统，能够实现系统的基本要求，但是由于实验室环境条件限制，时间紧张以及个人水平有限，系统存在一些不足，需要进一步的改进与完善。

1. 由于前期对视频采集板的评估不足，CMOS 图像传感器的光学镜头固定不牢，导致对采集的视频图像有干扰，需要对 PCB 布局重新设计。
2. 人脸跟踪技术移植工作没有完成。主要是处理器为 ARM 和 DSP 双核，算法部分需要在 DSP 内核实现，DSP 内核处理后的结果传输给 ARM 内核，两者之间需要通信，这就需要实现对 DSP 端的算法进行封装，由于时间紧，没来得及做完，需要进一步开发。
3. 本文设计的系统只是实现视频采集、处理与显示过程，而没有实现视频数据的存储和传输等功能，可以设计将视频数据保存在硬盘、SD 卡或者 U 盘中，也可以对视频数据进行 H.264 或 MJPEG 压缩，通过网络实现对视频数据的实时传输。
4. 界面设计相对简单，今后可以对界面多增加些功能，包括对视频图像参数配置，网络配置以及初始化设备等等。

参考文献

[1] 宋慧. 高清监控系统技术设计及应用探讨 [J]. 中国安防：技术应用, 2010, 10(2)：45 – 48.

[2] 张秀玲. 视频监控系统研究现状与发展趋势 [J]. 科技信息, 2008(36)：341 – 343.

[3] MURAKAMI T. Technical trends of next generation ultra high definition video - compression, transmission technologies and their standardizations[C] // 2010 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB). [S.l.]：IEEE, 2010：1 – 34.

[4] 李涛. 基于 ARM 的 USB 视频采集系统研究 [D]. 秦皇岛：燕山大学, 2011.

[5] Texas Instruments. TMS320DM368 Digital Media System-on-Chip (DMSoC). Literature Number: SPRS[K]. Texas：Texas Instruments Inc., 2010.

[6] 梁广俊. 基于 TMS320DM3 的新型架构的高分辨率数字视频展台的系统设计 [D]. 太原：太原理工大学, 2008.

[7] 牛超. 高清视频采集处理技术研究 [D]. 西安：西安工业大学, 2012.

[8] 毛晓东, 樊亚文. 高清视频监控技术在城市公共安全中的应用 [J]. 上海交通大学图像通信与信息处理研究所, 2010, 34(4)：103 – 105.

[9] 王庆有, 孙学珠. CCD 应用技术 [M]. 天津：天津大学出版社, 1993.

[10] (日) 米本和也. CCD/CMOS 图像传感器基础与应用 [M]. 陈榕庭, 彭美桂, 译. 北京：科学出版社, 2006.

[11] 李玉东, 李玉萍. 精通嵌入式 Linux 编程 [M]. 北京：北京航空航天大学出版社, 2010.

[12] 何景波. 基于 Linux 的嵌入式应用系统技术研究 [D]. 太原：中北大学, 2009.

[13] FURBER S. ARM System-on-Chip Architecture[M]. [S.l.]：Addison-Wesley Press, 2005：49 – 52.

[14] 彭启琮, 杨錬, 潘晔. 开放式多媒体应用平台：OMAP 处理器的原理及应用 [M]. 北京：电子工业出版社, 2015：6 – 9.

[15] 余国华, 冯启明. 基于 CMOS 图像传感器的视频采集系统设计 [J]. 武汉理工大学学报 (交通科学与工程版), 2004, 28(1)：145 – 147.

[16] Texas Instruments. AM/DM37x Multimedia Device Technical Reference Manual[K]. Texas：Texas Instruments Inc., 2009.

[17] 马兰. 高清数字视频处理平台软硬件设计 [D]. 成都：电子科技大学, 2011.

[18] 王黎明, 陈双桥. ARM9 嵌入式系统开发与实践 [M]. 北京：北京航空航大大学出版社, 2008.

[19] 李超, 肖健. 嵌入式 Linux 开发技术与应用 [M]. 北京：电子工业出版社, 2008：12 – 15.

[20] 刘登诚, 沈苏彬, 李莉. 基于 V4L2 的视频驱动程序设计与实现 [J]. 微计算机信息, 2011, 27(10)：56 – 58.

[21] 三凯, 王玉玫. 嵌入式系统图形处理平台设计与实现 [J]. 计算机工程与设计, 2007, 16：3949 – 3952.

[22] 张贻雄, 刘鹏, 王维东. 一种基于像素自适应的 OSD 分层混合结构设计 [J]. 电路与系统学报, 2008, 13(4)：20 – 23.

[23] 郝亚茹, 王瑞光, 郑喜凤. TFP410 在数字视频接口中的应用 [J]. 电子器件, 2007, 30(1).

[24] ubuntu 10.04 安装与配置 nfs 服务器 [EB/OL]. CSDN, 2013. http://blog.csdn.net/kenin_hcy/article/details/5909049.

[25] ubuntu 10.04 下 samba 服务的配置 [EB/OL]. 新浪博客, 2013. http://blog.sina.com.cn/s/blog_4d80100nwdr.html.

[26] 陈友平. 用于 53C2410X 的嵌入式 Linux 的研究和移植 [D]. 成都：电子科技大学, 2006.

[27] WOOKEY, TAK-SHING. Porting the Linux Kernel to a New ARM Platform[J]. SOLUTIONS JOURNAL, 2002, 4.

[28] 刘磊, 张风荔. 基于 U-boot 构建嵌入式 Linux 的 Bootloader[J]. 计算机应用研究, 2007, 12(24)：238 – 240.

[29] 杨延军. 用 busybox 制作嵌入式 LINUX 的文件系统 [J]. 单片机与嵌入式系统, 2005(4) : 15 – 20.

[30] 张翔, 刘鹏, 戴国骏. 嵌入式 Linux 闪存文件系统 JFFS2 的研究 [J]. 杭州电子工业学院学报, 2005, 22(3) : 62 – 65.

[31] 周民军. 基于 ARM9 的嵌入式操作系统的设备驱动设计 [D]. 武汉 : 武汉理工大学, 2010.

[32] 董志国, 李式巨. 嵌入式 Linux 设备驱动程序开发 [J]. 计算机工程与设计, 2006, 23(8) : 35 – 40.

[33] SCHIMEK M, DIRKS B, VERKULI H. Video for Linux two API Specification[EB/OL]. 2008.
http://v4l2spec.bytesex.org/v4l2spec/v4l2.pdf.

[34] 陈书海, 傅录祥. 实用数字图像处理 [M]. 北京 : 科学出版社, 2005.

[35] CASTLEMAN K R. Image Processing[M]. New Jersey : Prentice-Hall, Cliffs, 1979.

[36] 汪志云, 黄梦为, 胡钋, 等. 基于直方图的图像增强及其 MATLAB 实现 [J]. 计算机工程与科学, 2006(02) : 54 – 56.

[37] 孙忠贵, 王玲. 数字图像直方图均衡化的自应校正研究 [J]. 计算机时代, 2004, 149(11) : 19 – 20.

[38] 李明, 赵勋杰, 毛伟. sobel 边缘检测的 FPGA 实现 [J]. 现代电子技术, 2009, 16 : 44 – 46.

[39] CANNY J F. Readings in Computer Vision: Issues, Problems, Principles, and Paradigms[G/OL] // FISCHLER M A, FIRSCHEIN O. . San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1987 : 184 – 203.
http://dl.acm.org/citation.cfm?id=33517.33534.

[40] 吴国良, 杨浩, 罗建. 二维 A Tuous 算法图像边缘检测研究 [J]. 计算机工程与应用, 2010, 46(11) : 167 – 169.

[41] 张雪. 嵌入式人脸检测与跟踪系统的设计与实现 [D]. 长春 : 东北师范大学, 2011.

[42] 张利苹. 人脸检测与跟踪系统研究 [D]. 西安 : 西安电子科技大学, 2009 : 13 – 16.

[43] LIENHART R. An extended se to of Haar-like features for rapid object detection[C] // IEEE International conference on Image : Vol 3. New York, USA : IEEE, 2002 : 900 – 903.

[44] 张忠. Adaboost 人脸检测算法在嵌入式平台上的实现与研究 [D]. 上海 : 上海交通大学, 2008.

[45] 徐盛. TMS320 DSP 算法标准 [M]. 北京 : 清华大学出版社, 2007 : 12 – 18.

[46] 成洁, 卢紫毅. Linux 窗口程序设计：Qt4 精彩实例分析 [M]. 北京 : 清华大学出版社, 2008.

[47] 朱健琪. 基于 Linux 操作系统的智能仪器软件设计 [D]. 天津 : 天津大学, 2007.

[48] 任彧, 邱军, 顾成成. 基于 Linux 平台的实时视频 OSD 设计与实现 [J]. 计算机应用与软件, 2010, 27(10) : 185 – 187.