西南交通大学

硕士学位论文

基于BF533的网络视频监控系统设计与实现

姓名：孙延均

申请学位级别：硕士

专业：系统工程

指导教师：苟先太

20090501

摘 要

传统视频监控系统具有成本较高，需要铺设专用线路，无法联网，需要耗费大量的存储介质等缺点。随着视频处理和网络技术的日益成熟，网络视频监控系统被越来越多地应用到生产生活中，它使人们能够及时地获取被监控现场的环境状况，做出正确的决策。本文设计与实现了基于 ADSP-BF533 处理器的嵌入式网络视频监控系统，它具有成本低廉，安装灵活，功能丰富等特点。

通过分析网络视频监控系统三种解决方案的优缺点，综合考虑了成本、开发难度等因素，最终选择了 BF533 嵌入式 DSP 处理器作为本系统的基础平台架构，并在此基础之上对硬件电路所需的各种外围芯片和系统软件进行了选型。

本文对 BF533 处理器、SDRAM、FLASH、视频解码器、视频编码器、音频编解码器、以太网控制器、RS232 驱动器/接收器、CPLD 以及电源模块的设计进行了详细的描述，包括各模块之间的相互连接情况，设计中需要注意的事项等。

在将 U-BOOT 移植到特定开发板之前，需要先熟悉其源代码，本文依照 U-BOOT 的启动流程，对其源代码进行了比较深入的分析。编写了 DM9000AE 和 SST39VF1601 驱动程序，给出了 U-BOOT 移植的具体步骤。

为了检验系统硬件是否能正常工作，本文讨论了系统硬件各模块的测试方法，编写了具体的测试例程，给出了相应的测试结果。说明了编译U-BOOT的方法，以及设置U-BOOT从网络加载μClinux的方法。

关键词：嵌入式系统；BF533；视频监控；U-BOOT

Abstract

The traditional video surveillance systems have the disadvantages that their costs are high and need dedicated lines. It can't connect to the network and need a mass of storage medium. With the video processing and network technology becoming more and more sophisticated, the network video surveillance system has been increasingly applied to our life. It allows people to access to the environmental condition of the monitored point, and to make the right decisions. A BF533-based embedded network video surveillance system is designed and implemented with features of low-cost, flexible installation, function-rich.

The advantages and disadvantages of three different solutions of the network video surveillance system paper are analyzed. It considers such factors as the cost, the development difficulty, and so on. Ultimately the BF533 embedded DSP processor is selected as the basis of the system platform architecture. The peripheral chips and system softwares are selected on that basis.

The hardware modules including BF533 processor, SDRAM, FLASH, video decoder, video encoder, audio codec, ethernet controller, RS232 driver/receiver, CPLD and power are designed in detail.

Before porting U-BOOT to a specific board, it should understand the source code. The U-BOOT source code is analyzed in-depth according to the startup flow of U-BOOT. The dirvers of DM9000AE and SST39VF1601 are programmed. The concrete step of porting U-BOOT is described.

It should be tested whether the system hardware works properly. The test method of each module is discussed. The test code is written. The test result is given. The procedure of compiling U-BOOT is shown. The setup of loading μClinux from network is explained.

key words: embedded system; BF533; video surveillance; U-BOOT

西南交通大学

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权西南交通大学可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复印手段保存和汇编本学位论文。

本学位论文属于

1．保密□，在　　年解密后适用本授权书；

2．不保密☑，使用本授权书。

（请在以上方框内打“√”）

学位论文作者签名：孙延均　　指导老师签名：

日期：2009.5.31　　日期：2009.5.31

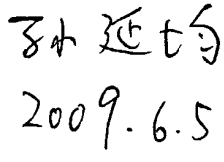
西南交通大学学位论文创新性声明

本人郑重声明：所呈交的学位论文，是在导师指导下独立进行研究工作所得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中作了明确的说明。本人完全意识到本声明的法律结果由本人承担。

本学位论文的主要创新点如下：

1.使用 BF533 实现了网络视频监控系统。BF533 能够同时完成视频压缩和网络传输，这样硬件设计简化的同时还降低了成本，并且它还支持嵌入式 Linux 操作系统，这有利于降低开发难度、加快开发进度。

2.本文对 TFTP 协议进行了改进，加强了嵌入式系统的网络安全性。修改了标准 TFTP 协议包头所使用的操作码字段值，并且还在包头加入了验证信息字段，从而有效防止了非授权访问。



2009.6.5

第1章 绪论

1.1 引言

随着社会的不断发展，人们对于准确及时地获取信息的需求越来越强烈。视频信息是对客观事物实实在在的描述，不同物体之间不易发生混淆，信息量极大，人的视觉系统获取的信息占外界信息总量的70%，总的来说，视频信息具有广泛性、直观性、确定性、高效性等特点，是人们获取外界信息最有效的方式。并且人们不仅想获取自己视力范围内的信息，往往还要求能随时观察距离遥远的某一环境的实时状况，比如，你身在异地，但是想了解你的公司或家里的当前状况，或者某些不适合人现场操作但是需要对其进行实时监控的工作环境，诸如此类的情况，就需要一种方式将异地的实时情况传输给观测者，而网络视频监控系统可以很好的满足这一需求。简单来说，网络视频监控系统主要功能就是将安装在目标监控点的摄像机所采集的实时视频信息通过网络传输至监控主机。

在市场需求方面，随着安防意识的强化和普及，安防领域逐渐扩展，由公共安全、大中型企业监控逐渐转向中小企业、家庭用户的安全防控，由于人们房产、汽车、家电等高值大件财产的增多，监控对象进一步扩大，其中包括对财产、设施及人的实时跟踪监控[1]。越来越多的企业和家庭希望通过网络监控系统帮助他们尽量避免损失、防止意外，随时掌控现场，减少不安全、不公平等因素。这些都将是一个巨大的需求市场[2]。据诺达咨询公司发布的《2007网络视频监控业务研究报告》显示：2006年中国网络视频监控市场规模达18.24亿元，未来5年内，网络视频监控将保持约38%的年增长率，预计2011年将达86.61亿元，而且会有越来越多的网络视频监控系统采用完全数字化的技术，特别是在银行、交通、工业、零售业等行业市场。以数字化视频监控为基础的智能视频监控技术将获得长足发展[3]。

1.2 视频监控系统国内外现状

1.2.1 视频监控系统分类

视频监控系统根据其所采用的技术大致可以分为三种：全模拟视频监控系统、数字化视频监控系统、网络视频监控系统[4,5]。

1. 全模拟视频监控系统

全模拟视频监控系统，也称闭路电视监控系统（CCTV）[4]。本地图像监控系统主要由摄像机、视频矩阵、监视器、录像机等组成，利用模拟视频线将来自摄像机的视频连接到监视器上，利用视频矩阵主机，采用键盘进行切换和控制，录像采用的是使用磁带的长时间录像机；远距离图象传输采用模拟光纤，利用光端机进行视频的传输[6]。全模拟视频监控系统以模拟视频矩阵和磁带式录像设备 VCR 为核心[4]。模拟视频监控系统经过几十年的发展，技术成熟，系统功能强大、完善。但是其主要缺点为：系统扩展能力差，对于已经建好的系统，如要增加新的监控点，往往是牵一发而动全身，新的设备也很难添加到原有的系统之中；无法形成有效的报警联动，在模拟监控系统中，由于各部分独立运作，相互之间的控制协议很难互通，联动只能在有限的范围内进行[5,7]。

2. 数字化视频监控系统

数字视频监控系统从 20 世纪 90 年代中期开始出现，以数字控制的视频矩阵替代原来的模拟视频矩阵，以数字硬盘录像机 DVR 替代原来的长延时模拟录像机，将原来的磁带存储模式转变成数字存储录像，实现了将模拟视频转为数字录像。DVR 集合了录像机、画面分割器等功能，跨出数字监控的第一步[4,8]。在此基础上产生了全数字的视频监控系统，可以基于 PC 机或嵌入式设备构成监控系统，并进行多媒体管理。数字监控作为继模拟监控之后的第二代监控技术，无论在图像质量、保存时间以及可靠性上均有大幅度的提升，为用户提供了高性价比的监控解决方案[9]。与全模拟视频监控系统相似，存在许多缺陷，要实现远距离视频传输需铺设（租用）光缆、在光缆两端安装视频光端机设备，系统建设成本高，不易维护、且维护费用较大。

3. 网络视频监控系统

随着宽带网络的普及，视频监控逐渐从本地监控向远程监控发展，出现了以网络视频服务器为代表的远程网络视频监控系统[4]。网络视频服务器解决了视频流在网络上的传输问题，从图像采集开始进行数字化处理、传输，这样使得传输线路的选择更加多样性，只要有网络的地方，就提供了图像传输的可能[10]。整个系统趋向平台化、智能化。网络视频监控成为监控领域又一新的发展方向[11]。

网络视频监控是一项完全基于宽带网的图像远程传输、远程管理的业务，它可以实现监控、报警联动、管理及视频存储等功能。目前该业务主要应用于公共安全、银行系统、道路交通、电力系统、环境监测等需要进行远距离监控的部门，还可以应用于学校、消防等方面的远程监控，信息化小区的网上监控，公司区域联网监控以及家居安全网上监控等[12]。

与前两代监控技术相比，网络视频监控的出现突破了模拟和数字视频监控的缺点，利用 TCP/IP 网络，实现了远程监控和低成本扩展监控范围，使得视频监控可以向很多领域渗透[13]。其主要优势包括：第一，可以利用现有的宽带网络资源实现远距离监控，不需要为新建监控系统铺设线路；第二，具有良好的系统扩展能力，只需要增加监控点设备就可扩展新的监控点；第三，前端设备是即插即用、免维护系统、维护费用低；第四，业务功能强大，管理功能丰富；第五，全数字化存储方式，便于保存和检索；第六，只要安装了客户端软件，并给予相应的权限，用户就可随时随地通过互联网浏览相关监控点的情况[11,14]。

1.2.2 网络视频监控系统研究现状

网络视频监控系统可以根据其处理采集到的视频信号的方式，分为三种类型[15,16,17]：

1. 嵌入式微处理器+视频压缩芯片

这种方案中视频的压缩完全由专用的视频压缩芯片来完成，嵌入式微处理器则完成控制功能。视频压缩芯片的引入可以使系统开发者省去开发相应视频压缩算法的工作，对于想要节约开发成本或是开发实力不够的中小公司来说是一个很不错的选择。市场上提供视频压缩类芯片的厂商也比较多，比

如，台湾凌泰科技的 AL9V576 支持 MPEG-4/2/1 视频压缩标准[18]，日本富士通公司最近推出的 MB86H51 是基于 H.264 视频压缩标准的。

但是，基于专用视频压缩芯片的系统灵活性和可扩展性受到了很多限制：

（1）由于芯片开发周期很长，成本代价太大，而市场又在不断发生变化，产品更新换代的速度非常迅速，等待集成了算法的专用视频压缩芯片成熟后开发产品，肯定跟不上市场的发展。

（2）监控系统强调的是实时压缩，在实时压缩的同时还必须进行一些必要的处理，如运动检测、时间发生器、LOGO 标识、水印等，这些处理都是前处理，这些辅助功能对压缩产品而言显得也非常重要，这些处理都是在压缩前或压缩中进行的。

（3）在监控领域的应用中，强烈需要动态的调整编码参数，如帧率、图象质量、帧结构等，而硬件编码的灵活性明显不如软件编码。

2. 嵌入式微处理器+FPGA

这种方案中 FPGA 用来对视频信号进行压缩处理，经压缩之后的视频数据在嵌入式微处理器的控制下通过网络发送出去。目前市场上的嵌入式微处理器有较多选择，比如，基于 ARM 内核的三星 S3C6410，基于 PowerPC 内核的飞思卡尔 PowerQuicc 等。Altera、Xilinx、Actel、Lattice 等几家生产厂商提供了比较丰富 FPGA 芯片产品线，比如，Altera 公司的 Cyclone III 系列 FPGA，Xilinx 公司的 Virtex-5 系列 FPGA。

利用 FPGA 来实现视频压缩编码的好处在于，可以灵活的实现各种压缩编码算法，并且在必要时可以很方便将新的编码标准应用于系统中，而这一切只需更换 FPGA 程序，硬件电路不需要做任何改动，这就为新产品的迅速推出提供了有利条件。并且各 FPGA 生产厂商都会提供各种功能的 IP 核，比如 H.264 编码器 IP 核、MEPG4 编码器 IP 核等，这大大加快了产品研发的进度，使产品尽可能早的进入市场，不过大多数的 IP 核是需要付费的，可以根据应用需要购买相应的 IP 核[19]。

此种方案的缺点在于，系统同时使用了嵌入式微处理器和 FPGA，所以系统成本会比较高，对于成本控制比较严格的应用来说不是好的选择。

3. 高性能 DSP 处理器

高性能DSP处理器在保持DSP芯片强劲的数据处理功能的基础上，增加了MCU控制功能，拥有丰富的外设接口和较高的时钟频率，是专门针对多媒体应用而设计的处理器。此类处理器中，比较有代表性的有 ADI 公司的Blackfin系列， TI公司的TMS320C64x系列。

此方案的优点在于，系统中视频信号的压缩和传输都由单片DSP处理器来完成，而不需要其它额外的芯片，这样即简化了系统设计，又使系统的成本得到了有效控制。视频压缩算法完全由汇编语言或C语言实现，可以及时吸纳视频编码最新成果对系统进行升级，以适应市场需求。

本系统采用的就是单DSP处理器方案，处理器具体型号选择上，需要能胜任视频图像的压缩以及网络传输操作，并且需要嵌入式操作系统的支持，所以综合性能和价格等因素，本系统的处理器最终选择的是 ADI 公司的ADSP-BF533处理器。

1.3 Blackfin处理器概述

本系统所使用的ADI公司BF533处理器是Blackfin处理器系列产品中的一员。

Blackfin 处理器是一类专为满足当今嵌入式音频、视频和通信应用的计算要求和功耗约束条件而设计的新型16～32位嵌入式处理器。Blackfin处理器基于由 ADI 和 Intel 公司联合开发的微信号架构（Micro Signal Architecture， MSA)，它将一个 32 位 RISC 型指令集和双 16 位乘法累加(MAC)信号处理功能与通用型微控制器所具有的易用性组合在了一起。这种处理特征的组合使得Blackfin处理器能够在信号处理和控制处理应用中均发挥上佳的作用——在许多场合中免除了增设单独的异类处理器的需要。该能力极大地简化了硬件和软件设计实现任务[20]。

目前，Blackfin 处理器在单内核产品中可提供高达 756MHz 的性能。Blackfin 处理器系列中的新型对称多处理器成员在相同的频率条件下实现了性能的翻番。Blackfin处理器系列还提供了低至0.8V的业界领先功耗性能。对于满足当今及未来的信号处理应用（包括宽带无线、具有音频/视频功能的因特网工具和移动通信）而言，这种高性能与低功耗的组合是必不可少的[21]。

Blackfin处理器架构基于一个10级RISC MCU/DSP流水线和一个专为

实现最佳代码密度而设计的混合 16/32 位指令集架构。Blackfin 处理器架构还完全符合 SIMD 标准，并包括用于加速视频和图像处理的指令。该架构很适合于全信号处理/分析能力，同时还可在单内核器件或双内核器件上提供高效 RISC MCU 控制任务执行能力。由于具有最佳代码密度且只需进行极少（或者完全不需要进行）代码优化处理，因此可缩短产品的面市时间，而不会遇到其它传统处理器所常见的性能空间障碍。在视频处理方面，除了具有对 8 位数据以及许多像素处理算法所常用的字长的固有支持之外，Blackfin 处理器架构还包括专为增强视频处理应用中的性能而定义的指令。比如，离散余弦变换（DCT）通过一个 IEEE1180 舍入操作得到支持，而“SUM ABSOLUTE DIFFERENCE”指令则支持在诸如 MPEG2、MPEG4 和 JPEG 等视频压缩算法中所使用的运动估计算法。利用软件来实现视频压缩算法使得 OEM 制造商能够在不变更硬件的情况下适应不断发展的标准和新型功能要求。增强型指令可使 Blackfin 处理器在那些先前主要是由 ASIC、VLI 媒体处理器或硬连线芯片组来满足的应用中一试身手。归根结底，Blackfin 处理器将在帮助降低系统成本的同时使终端应用产品的上市时间得以缩短[22]。

1.4 研究内容与目标

经过对网络视频监控系统发展现状和市场研究动向的分析，本文提出了一种嵌入式网络视频监控系统设计方案，本系统的主要功能是将模拟摄像头采集的视频信号数字化之后，经过视频压缩算法处理，最后通过网络传输至远端控制中心。本论文主要研究内容有：

（1）系统需求分析和总体设计。分析系统的功能需求，并据此得出系统总体功能框架结构。

（2）硬件电路设计。设计系统硬件电路原理图和 PCB 图。

（3）U-BOOT 源代码分析。因为移植 U-BOOT 需要对其源代码进行修改，所以分析 U-BOOT 源代码，熟悉其整个启动流程。

（4）驱动程序编写。需要编写特定硬件的驱动程序，以使 U-BOOT 能正常运行，实现所需功能。

（5）硬件电路的调试。硬件电路制作完毕之后，为了检测其是否能正常工作，需要对其进行调试，具体的调试步骤是按照电源→处理器→存储器→其它模块的顺序，对各个模块逐一进行调试。

（6）U-BOOT 移植。将 U-BOOT 移植到本系统中，并使其能顺利的从网络加载嵌入式操作系统 μClinux。

论文将达到以下目标：

（1）设计制作出硬件电路，并完成调试工作，使系统硬件能正常运行。

（2）能正确的将 U-BOOT 移植至本系统中，使其能完成对 μClinux 的网络加载。

1.5 论文主要工作及组织结构

第1章 绪论。介绍了网络视频监控系统的研究意义、国内外研究现状以及 Blackfin 处理器的基本情况，给出了本论文的研究内容和目标。

第2章 系统总体设计。给出了系统需求分析和系统体系结构，并分析了系统硬件各功能模块主芯片的选型和 bootloader 及操作系统的选型。

第3章 系统硬件电路设计。设计硬件电路各个功能模块，给出了各模块具体连接情况以及设计注意事项

第4章 U-BOOT 启动流程分析与驱动编写。分析了本系统所使用的 bootloader 程序——U-BOOT 的启动流程。编写了 DM9000AE 和 SST39VF1601 驱动程序。给出了移植的具体过程，特别讨论了更安全的 TFTP 协议的移植。

第5章 系统硬件调试及 U-BOOT 编译与加载。说明了各硬件模块的调试方法，给出了具体的测试例程和相关测试结果。并给出了编译 U-BOOT 的具体步骤，以及 U-BOOT 网络加载 μClinux 的相关设置。

结 论 论文最后对全文进行了总结，并提出了下一步的工作方向。

第2章 系统总体设计

在进行系统的详细设计之前，需要首先充分了解用户的各种需求情况做出需求分析，接着依照需求分析进行系统的体系结构设计，然后根据系统的体系结构框图进行硬件和软件的选型。

2.1 系统需求分析

嵌入式网络视频监控系统主要用于对远程现场环境的监控，通过实时获取被监控地点视频信息，主控中心可以根据现场的即时情况，及时采取相应措施。

系统主要功能是对被监控现场的环境状况进行动态实时的视频信息采集，将其传输至主控中心，并能比较流畅的播放。此外系统还有可选择的辅助功能，有些应用需要获取更为丰富的信息，可加入声音采集功能，从而控制中心能够得到声像合一的监控信息，为正确的作出决策提供尽可能多的信息量支持；为了对采集的视频信息进行本地回放，可以加入一个视频回放功能，它能够对所采集的视频进行本地化实时播放，主要是系统调试及故障检测用途。

基于上述的系统功能分析之后，可以根据市场上主流芯片产品价格估算出系统的生产成本大约在300-400元。如果不需要可选择的辅助功能的话，生产成本还会更低。用户可以按照自己的实际需求来选择是否需要加入辅助功能，这样可以很灵活的控制成本，并达到资源利用最大化。

考虑到系统可能应用在不同的工作环境，比如，工厂的生产现场，或者人群密集的公共场所等等，这就要求系统要便于运输和安装，并且占用尽量小的空间，所以系统必须具备小巧、灵活的特点。而本系统是基于嵌入式技术设计和开发的，系统各功能部件高度集成，完全能够满足前述需求。其物理尺寸大致相当于一本32K书大小。

在功耗方面，由于本系统通常是固定安装在某一使用场所，并且需要长时间不间断的运行，在系统供电的选择上使用的是外部电源供电，所以对功率损耗的控制不需要太多考虑，只要保证电源模块能正常驱动各功能部件就

可以了。

综上所述，表 2-1 是本系统需求分析表。

表 2-1 系统需求分析表

名称		说明
目的		实现对目标环境的远程监控
功能	基本功能	视频信号采集、压缩、传输
	辅助功能	语音信号采集、压缩及传输，视频本地回放
性能		视频信号传输速率 20fps
生产成本		300-400，如不需要辅助功能成本更低
物理尺寸		16cm×18cm

2.2 系统体系结构设计

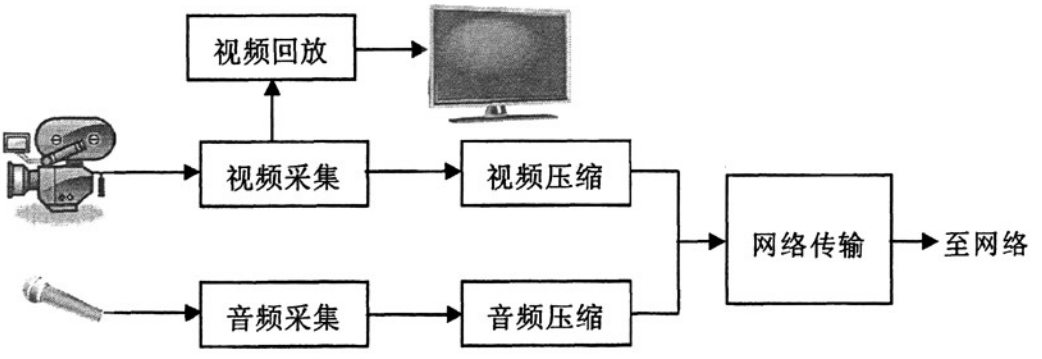


图 2-1 系统体系结构框图

通过对系统需求的分析，可得出系统体系结构框图，如图 2-1 所示。本系统分为如下几个模块：

视频采集模块。主要功能是将模拟摄像头输入的模拟视频信号转化成后续处理所需的数字视频信号。

视频压缩模块。主要功能是将数字视频信号通过压缩编码算法转化为体积更小的压缩视频信号。

网络传输模块。主要功能是将压缩后的视音频数据传输至控制中心。

音频采集模块。主要功能是将麦克风输入的模拟音频信号转化成后续处

理所需的数字音频信号。

音频压缩模块。主要功能是将数字音频信号通过压缩编码算法转化为体积更小的压缩音频信号。

视频回放模块。主要功能是将数字视频信号转化成模拟视频信号，并输出至显示设备进行播放。

2.3 系统硬件选型

系统硬件设计是基于系统体系结构来进行的，首先需要根据系统体系结构制定出系统硬件功能框图，此框图需要描述出系统所需要的各种功能相对应的硬件模块。

本系统主要的硬件模块包括：DSP 处理器、FLASH、SDRAM、电源模块、以太网控制器、RS232 驱动器/接收器、视频解码器、视频编码器、音频编解码器，CPLD。系统硬件功能框图如图 2-2 所示。

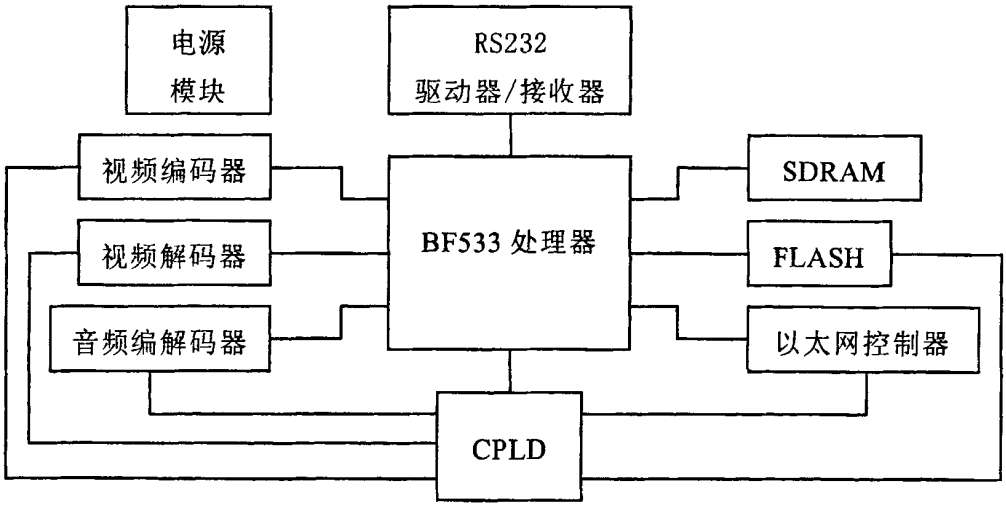


图 2-2 系统硬件功能框图

在得出硬件功能框图后，就需要根据各硬件模块功能选择符合需求的芯片，其中芯片的选择主要依照的原则是：

（1）芯片能够满足所需的功能，这一点是最重要的，如果所选的芯片不能满足功能需求，那么这将最终导致所设计的系统不能满足用户的需求。

（2）其次需要考虑的是开发成本，其中成本包括两个方面，一个是芯片本身的购买成本，另一个成本指的就是对芯片进行相应软件开发所需的人

员和时间成本，如果是功能基本一样的两款芯片其价格应该不会相差太多，但是这两款芯片的应用参考资料和技术支持方面可能会有一定差距，所以我们在选择芯片的时候应该尽量选择应用参考资料丰富和技术支持完善的芯片，这样开发难度将大大减小，从而开发进度会得到显著提高，这不仅节约了时间和人员成本，更为重要的是可以加快新产品推向市场的时间，赢得市场占有率。

（3）对于移动和手持设备，芯片的功耗也是需要考虑的一个问题，基于低能耗设计的芯片将更能满足这类设备的需求。

2.3.1 外部存储器的选择

由于 BF533 片内存储器较小（指令和数据存储器总共为 148KB），在视频应用中需要大量临时存储空间来存放视频数据，所以利用 BF533 的外部总线接口来扩展一个 SDRAM 存储器；系统还需要一个引导程序来初始化硬件及加载嵌入式操作系统，引导程序的特点就是系统上电之后就直接运行，因此引导程序需要存储在一个非易失性存储设备中，此外系统还需要保存一些全局环境和系统配置参数，使系统在掉电后保存相关配置便于下次运行时使用，所以通过外部总线接口连接一个 FLASH 存储器来满足非易失存储要求。

1. SDRAM 的选择

考虑到系统应用需要存储空间比较大，所以本系统选择的SDRAM芯片是海力士半导体公司（Hynix Semiconductor Inc.）生产的HY57V561620CTP-H，其容量大小为32MB（4Banks×4M×16Bit），包括了4个bank，每个bank由4M个16位存储单元构成，数据总线的宽度为16位。该芯片采用3.3V电压供电，具有自动刷新（Auto Refresh）和自刷新（Self Refresh）功能，刷新的频率为8192 Refresh Cycle/64ms，支持2或3个时钟周期的CAS可编程延时[25]。

2. FLASH 的选择

目前在嵌入式产品中使用的 FLASH 分为两类，一种是 NOR FLASH，另一种是 NAND FLASH。NOR FLASH 的特点是芯片内执行(XIP, eXecute In Place)，这样应用程序可以直接在 FLASH 闪存内运行，不必再把代码读到系统 RAM 中。NOR FLASH 的传输效率很高，在 1～4MB 的小容量时具有很高的成本效

益，但是很低的写入和擦除速度大大影响了它的性能。NAND FLASH 结构能提供极高的单元密度，可以达到高存储密度，并且写入和擦除的速度也很快，但是应用 NAND 的困难在于 FLASH 的管理和需要特殊的系统接口。

本系统中所需要的引导程序和配置参数存储空间很小并且为让引导代码上电后能直接从它的存储空间运行，所以使用NOR FLASH作为系统的非易失性存储介质。系统中使用的FLASH芯片为SST公司（Silicon Storage Technology, Inc.）生产的SST39VF1601，其容量为2MB(1M×16Bit)，采用2.7～3.6V电压供电，具有高性能的字编程能力，字编程时间一般为7us[26]。它还提供了对其低32K字存储空间的硬件数据保护功能，利用此功能，系统开发者可以将系统中不需要经常更改的代码和数据放置在此32K字存储空间中，从而可以避免对这些代码和数据的误操作，提高系统的可靠性。

2.3.2 视频解码芯片的选择

视频解码器的主要功能是将输入的模拟视频信号转换成为处理器能够处理的数字视频信号。本系统所选用的视频解码芯片是飞利浦半导体公司生产的SAA7113H。

SAA7113H是一款9位视频输入处理器，包含两路模拟预处理线路，具有视频源选择、抗锯齿滤波、模数转换、自动钳位和增益控制、一个时钟发生电路、一个数字多制式解码器(PAL BGHI、PAL M、 PAL N、combination PAL N、 NTSC M、NTSC-Japan、NTSC N、SECAM)、亮度/对比度/饱和度控制、多标准的VBI数据解码和27MHZ VBI数据旁路[27]。

2.3.3 视频编码芯片的选择

视频编码器的作用与视频解码器相反，输入为数字视频信号，输出为模拟视频信号。本系统所选用的视频编码芯片是 ADI 生产的 ADV7171。

ADV7171是一款集成数字视频编码器，它能将CCIR-601 4:2:2 8位或16位的视频数据转换成兼容多种标准的模拟基带电视信号。其内部包含了一个SSAF(Super Sub-Alias Filter)超级低通滤波器，可将亮度信号（Y）的频带拓宽，抑制频带衰减，对提高画面的清晰度有很大的作用。另外，其先进的电源管理电路，使芯片无论工作在正常模式、掉电或休眠模式都能得到最

优的功率消耗控制[28]。

2.3.4 音频编解码芯片的选择

音频编解码器的主要功能是对声音的播放和采集。一方面，它能将数字格式的音频数据转换成模拟音频信号；另一方面，它也能将模拟音频信号转换成数字音频数据。本系统中所使用的音频编解码器是ADI生产的AD1836A。

AD1836A是一款高性能的单片音频编解码器，它集成了3路立体声D/A和两路立体声A/D，为了降低信号的干扰，模拟信号的输入输出均采用差分的形式。数据采样率最高为96KHZ，采样位数最高为24位[29]。

2.3.5 以太网控制芯片的选择

以太网控制器主要功能就是控制网络数据的收发。本系统所选用的以太网控制芯片是 DAVICOM 半导体公司生产的 DM9000AE。

DM9000AE 是一款高集成度和性价比、低引脚密度的单片快速以太网控制器，它的通用处理器接口使其能够与大多数的处理器进行连接，芯片内部集成了一个 10/100M PHY，并拥有 16KB 的内部 SRAM[30]。它具有自动线序交叉（AUTO-MDIX）和自动协商（Auto-Negotiation）功能。

2.3.6 RS232 驱动器/接收器芯片的选择

在对系统进行调试的时候，通过串口输入调试命令或从串口打印调试信息都是很重要的。BF533处理器芯片中集成有一个UART（Universal Asynchronous Receiver/Transmitter，称为通用异步收发器）接口，它能够在外部数据传输所需的串行数据格式与处理器内部所需的并行数据格式之间进行转换。由于UART输入/输出电平为TTL电平，与RS-232C标准所规定的电平不同，因此必须在RS232与TTL电路之间进行电平和逻辑关系的变换，这个变换可以通过RS232驱动器/接收器芯片来实现。在本系统中所使用的芯片为美信公司（Maxim）生产的MAX232A。

MAX232A能够满足所有使用EIA/TIA-232E和V.28/V.24通信接口的应用。

它具有功耗低，单电源供电，外接电路简单，波特率高，价格低廉等优点。

2.3.7 CPLD芯片的选择

CPLD(Complex Programmable Logic Device)复杂可编程逻辑器件，是从PAL和GAL器件发展出来的器件，相对而言规模大，结构复杂，属于大规模集成电路范围。是一种用户根据各自需要而自行构造逻辑功能的数字集成电路。其基本设计方法是借助集成开发软件平台，用原理图、硬件描述语言等方法，生成相应的目标文件，通过下载电缆（在系统编程）将代码传送到目标芯片中，实现设计的数字系统。

本系统设计中需要实现一些组合逻辑电路，如果使用专用的逻辑控制芯片来实现，首先所使用的元件的数量就会增多，其次电路设计的灵活性就下降了。为了减少元件使用数量和提高电路设计的灵活性，系统中选用了Xilinx公司生产的XC9572XL。

XC9572XL是一款低功耗、高性能的CPLD，它由4个54V8功能块组成，具有1600个可用门，管脚间传输延时为5ns，系统最高时钟频率可达178MHz[3]。

2.3.8 电源芯片组的选择

电源模块主要是为系统的运行提供合适的电源供应以及安全保护措施，包括线性稳压电路、复位电路、保护电路。

1. 线性稳压电路

本系统使用5V/3A外部电源为系统供电，因为系统中用到了5V、3.3V、1.25V等几种电压，所以使用了线性稳压器将输入电压转换为所需电压。

本系统中使用最多的是3.3V电压，为了从5V的输入电压得到3.3V的电压，系统中使用了凌特公司（Linear Technology）生产的LT1529-3.3 3A低压差线性稳压器。它在输出3A电流时，输入与输出电压之差可仅为0.6V。

本系统设计了两组电路来提供内核电压，一组是提供0.85V～1.3V的可变电压电路，另一组提供1.25V电压的固定电压电路。BF533处理器提供了一个片上调压器，它可从外部2.25V～3.6V供电电压产生处理器所需的0.85V～1.3V内核电压，其外部电路还需要一个场效应管，系统所选用的场效应管为

仙童半导体公司（Fairchild Semiconductor）生产的FDS9431A；固定电压电路为内核提供1.25V固定电压，此电路使用的芯片为美国国家半导体公司（National Semiconductor）生产的LM317A，它是一款三引脚的可变正线性稳压器，能够提供1.2～37V输出电压，同时输出电流能够达到1A。

2. 复位电路

为确保电路稳定可靠工作，复位电路是必不可少的一部分，复位电路的第一功能是上电复位，因为当系统上电时，处理器及其它芯片需要一个复位信号，来使芯片工作在初始化预定状态；另外，当系统处于异常状态时，也可以通过复位使系统恢复至初始状态。本系统中使用了美信公司生产的MAX708T电源监控芯片。

MAX708T是一款微处理器电源监控芯片，可同时输出高电平有效和低电平有效复位信号。复位信号可由VCC电压、手动复位或由独立的比较器触发。

3. 电路保护

当电路正常工作时，电路中的电流强度不会对电子元件产生任何危害，但是当电路处于异常状态时，比如电路中某处出现短路时，就会引起电路中电流瞬间急剧变大，而强电流很容易造成元器件的损坏，这种损坏是永久性且不可修复的，必须对损坏的元件进行更换才能使电路继续正常工作。为了避免这种异常状况所造成的电路损坏，可以通过使用电路保护装置来确保电路的安全。本系统所使用的电路保护芯片为Littelfuse生产的2920L300/15自恢复保险丝。

2920L300/15的最大工作电流为3A，过载电流为6A，正常工作时电阻为0.033Ω。

2.4 系统软件选型

如果将嵌入式系统比作人的话，硬件电路是其躯体，但是真正让嵌入式系统具有灵魂的是运行于硬件平台之上的软件，只有通过软件才能完成嵌入式系统所期望的各种功能，一个没有运行任何软件的嵌入式系统就如同一个没有灵魂和思想的人一样，什么都干不了。

本系统需要实现多种功能，包括音视频数据的采集、压缩、传输。这就

需要多个任务并行协调工作，为了使多个任务能很好的协同化工作，本系统引入嵌入式操作系统来管理多任务的运行。为了引导嵌入式操作系统的运行，本系统还需要一个引导装载程序——Boot Loader，Boot Loader最主要的任务就是初始化硬件设备，建立内存空间映射图，引导操作系统的加载。由此可以得出系统软件功能框图，如图2-3所示。

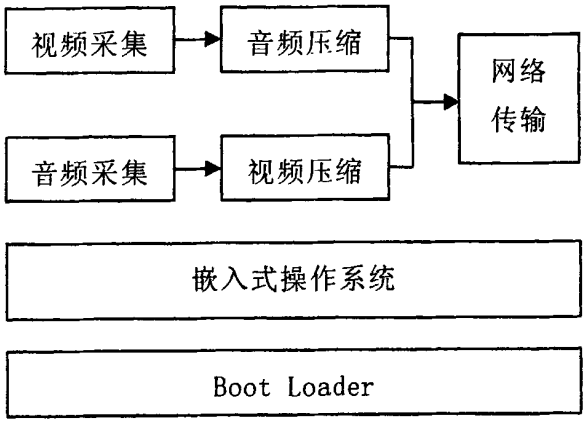


图 2-3 系统软件功能框图

2.4.1 嵌入式操作系统的选择

目前市场上使用较广泛的嵌入式操作包括微软公司（Microsoft）的Windows CE、风河系统公司（Wind River System）的 VxWorks、塞班公司（Symbian)的Symbian OS、μC/OS-II、μClinux等操作系统[36]。对于Windows CE、VxWorks来说，由于其高昂的许可费，使商用产品的造价都十分昂贵，用于一般用途会提高产品成本从而失去竞争力。Symbian OS 是专门针对智能手机研发的操作系统，主要用于无线通讯终端领域。μC/OS-II 和μClinux是两种性能优良、源码公开且被广泛应用的开源嵌入式操作系统，这就意味着任何人都可以免费的获得其源代码，并可以对源代码进行修改，因此开发者能够根据应用的特点来增减各种功能、添加对新硬件的支持等，为嵌入式系统的开发带来了极大的灵活性。

本系统中使用的操作系统为 μClinux，具有内嵌网络协议、支持多种文件系统，开发者可利用标准 Linux 先验知识等优势，完全支持 Blackfin 体系结构，并且其源代码是免费的。这些特点对于加快产品开发进度、降低产品成本都非常有力。

2.4.2 Boot Loader 的选择

Boot Loader 是在操作系统运行之前执行的一段小程序。通过这段小程序，我们可以初始化硬件设备、建立内存空间的映射表，从而建立适当的系统软硬件环境，为最终调用操作系统内核做好准备。

对于嵌入式系统而言，Boot Loader 是基于特定硬件平台来实现的。因此，几乎不可能为所有的嵌入式系统建立一个通用的 Boot Loader，不同的处理器架构都有不同的 Boot Loader。Boot Loader 不但依赖于处理器的体系结构，而且依赖于嵌入式系统板级设备的配置。对于两块不同的嵌入式板而言，即使它们使用同一种处理器，要想让运行在一块板子上的 Boot Loader 程序也能运行在另一块板子上，一般也都需要修改 Boot Loader 源程序。

表 2-2 开放源码的引导程序

Boot Loader	描述	x86	ARM	Blackfin
LILO	Linux 磁盘引导程序	是	否	否
GRUB	GNU 的 LILO 替代程序	是	否	否
Loadlin	从 DOS 引导 Linux	是	否	否
ROLO	从 ROM 引导 Linux 而不需要 BIOS	是	否	否
Etherboot	通过以太网卡启动的 Linux 系统的固件	是	否	否
LinuxBIOS	完全替代 BIOS 的 Linux 引导程序	是	否	否
BLOB	LART 等的硬件平台的引导程序	否	是	否
U-Boot	通用引导程序	是	是	是
RedBoot	基于 eCos 的引导程序	是	是	否
vivi	专为 ARM 处理器系列设计	否	是	否
Bios-lt	专为 S3C4510B 开发的引导程序	否	是	否

表 2-2 列出了目前各种使用较为广泛的开放源码的引导程序所支持的体系架构的情况。由表可见，U-BOOT 支持 Blackfin 体系结构，除了支持表中所列的三种体系结构之外，它还支持 PowerPC、MIPS、XScale 等多种体系结构。此外，U-Boot 完全支持对 μClinux 操作系统的引导，所以本系统所选用的 Boot Loader 为 U-Boot。

第3章 系统硬件设计

本章将详细介绍系统硬件电路各个模块的具体设计方法，给出了各模块的连接框图，说明了设计中需要注意的问题。

3.1 BF533处理器模块

处理器模块是整个系统的核心模块，它通过其上的外设接口将各种外围设备连接起来并对外设进行控制和管理。BF533处理器对下一代的数字通信和消费多媒体应用来说，是高度集成的片上系统解决方案。通过将工业标准接口与高性能的信号处理内核相结合，用户可以快速发出节省成本的解决方案，而无需昂贵的外部组件。

BF533处理器外设包括外部总线接口单元（EBIU）、一个并行外设接口（PPI）、一个UART口、一个SPI口、两个串行口（SPORTs）、一组可编程标志（PFx）、四个通用定时器（其中三个具有PWM功能）、一个实时时钟（RTC）、一个看门狗定时器、一个JTAG接口等。

BF533处理器模块的原理图请参见附录中图2。

3.1.1 外部总线接口单元

外部总线接口单元(EBIU)即可以用于异步设备（例如：FLASH、EPROM、ROM、SRAM和存储器映射I/O设备）也可以用于同步设备（例如：SDRAM）。它们的总线宽度均为16位，其中A1为16位字的地址最低位。8位的外围设备必须象16位设备一样对其寻址，但只使用其低8位数据。

1. SDRAM接口

PC133兼容的SDRAM控制器可以与4bit、8bit、16bit数据总线宽度的SDRAM相连，最大的容量可以达到128MB。SDRAM控制器具有以下特点：

- 支持4个SDRAM内部bank。

- 使用一个刷新计数器来从各种不同的时钟频率产生外部SDRAM所要求的刷新率。
- 提供了多个时序选择项，支持在处理器与SDRAM之间增加缓冲时间。
- 使用于一个独立的引脚（SA10）来对SDRAM预充电，预充电操作一般是在自动刷新和自刷新命令之前进行的操作。
- 提供两种上电选择。

BF533与SDRAM的连接框图将在3.2节中给出。表3-1给出了BF533的SDRAM接口的引脚描述。

表3-1 SDRAM接口引脚

引脚	类型	功能
DATA[15:0]	I/O	外部数据总线（与异步存储器共用）
ADDR[19:18] ADDR[16:1]	O	外部地址总线（与异步存储器共用） 连接至SDRAM地址引脚。ADDR[19:18]决定Bank地址，需要链接到SDRAM的BA[1:0]引脚。
/SRAS	O	SDRAM行地址选通脉冲引脚 连接至SDRAM的/RAS引脚。
/SCAS	O	SDRAM列地址选通脉冲引脚 连接至SDRAM的/CAS引脚。
/SWE	O	SDRAM写使能引脚 连接至SDRAM的/WE引脚。
/ABE[1:0]/SDQM[1:0]	O	SDRAM 数据屏蔽引脚 分别连接至SDRAM的UDQM和LDQM引脚。
/SMS	O	外部SDRAM存储器片选引脚 连接至SDRAM的/CS引脚。
SA10	O	SDRAM A10引脚 SDRAM接口使用这个引脚来对SDRAM进行刷新。连接至SDRAM的A10引脚。
SCKE	O	SDRAM时钟使能引脚 连接至SDRAM的CKE引脚。
CLKOUT	O	SDRAM时钟输出引脚 连接至SDRAM的CLK引脚。

2. 异步存储器接口

BF533的异步存储器接口通过编程控制多达4个块（bank）的时序参数灵活的各种异步存储设备。无论设备大小如何，每个bank都占据1MB。每一个存储器bank都有一个对应的使能控制信号，对其使用情况进行灵活的控制。

表3-2给出了BF533的异步存储器接口的引脚描述。表3-3给出了BF533的异步存储器bank地址范围。

表3-2 异步存储器接口引脚

引脚	类型	功能
DATA[15:0]	I/O	外部数据总线
ADDR[19:1]	O	外部地址总线
/ASM[3:0]	O	异步存储器bank选择
/AWE	O	异步存储器写使能
/ARE	O	异步存储器读使能
/AOE	O	异步存储器输出使能 在大多数情况下，/AOE引脚应该被连接到外部存储器映射异步设备。/ARE 与/AOE相比较而言，前者的时序要求更为严格。
ARDY	I	异步存储器就绪回应
/ABE[1:0]/SDQM[1:0]	O	字节使能

表 3-3 异步存储器 bank 地址范围

存储器块选择	起始地址	结束地址
/AMS3	0x20300000	0x203FFFFF
/AMS2	0x20200000	0x202FFFFF
/AMS1	0x20100000	0x201FFFFF
/AMS0	0x20000000	0x200FFFFF

本系统异步存储器接口用来连接 FLASH 芯片和 DM9000AE。FLASH 占用了两个 bank，分别是 bank0 和 bank1，所占用的地址空间为 0x20000000～0x201FFFFF，容量为 2MB。DM9000AE 占用 bank2，它所能寻址的地址范围为 0x20200000～0x202FFFFF，但是实际上它只需要用到其中两个字节地址来完成寻址操作。另外 bank3 未被使用，可以将其禁止。BF533 与 FLASH 的连接框图将在 3.3 节中给出，BF533 与 DM9000AE 的连接框图将在 3.7 节中给出。

3.1.2 并行外设接口

并行外设接口（PPI）是半双工形式，最高可进行16位数据传输。它有1个专用的时钟引脚，3个多路复用的帧同步引脚，4个专用的数据引脚，另外还有12个与可编程标志引脚复用的数据引脚。当PPI接口配置为8位模式时，系统能到达最大的数据吞吐量，因为两个8位数据采样能被封装成一个16位字，其中较早被采样的数据被放置在低8位。PPI引脚描述如表3-4所示。

表3-4 PPI接口引脚

引脚	功能	类型	复用功能
PPI[15:12]	数据传输	I/O	可编程标志引脚[4:7]/SPI片选输出[4:7]
PPI[11:4]	数据传输	I/O	可编程标志引脚[8:15]
PPI[3:0]	数据传输	I/O	N/A
PPI_FS3	帧同步3	I/O	可编程标志引脚3
PPI_FS2	帧同步2	I/O	定时器引脚2
PPI_FS1	帧同步1	I/O	定时器引脚1
PPI_CLK	PPI时钟	I	N/A

本系统中视频解码器SA7113H和视频编码器ADV7171都是通过PPI接口与BF533进行数据传输的。BF533与SA7113H的连接框图将在3.4节中给出，BF533与ADV7171的连接框图将在3.5节中给出。

PPI接口能够配置为两种操作模式：一种为通用模式，另一种为ITU-R 656模式。其中ITU-R 656模式是专门的视频处理模式，它支持整场、活动视频、VBI 三种输入模式和一种输出模式。

3.1.3 串行端口

BF533处理器有两个相同的同步串口，SPORT0和SPORT1。这两个同步串口支持多种串行数据通信协议，还在多处理器系统中提供处理器之间的直接连接。每一个SPORT端口都有两组引脚，一组引脚负责传输数据，另一组引脚负责接收数据。每一组引脚都包括主数据引脚、辅数据引脚、时钟引脚、

帧同步引脚。接收和传输能被独立编程。每一个SPORT端口都是一个全双工的设备，能够同时进行数据的接收与发送。它还能够对位速率、帧同步、数据字的位数等进行编程。

本系统中SPORT端口连接至AD1836A音频编解码器(Codec)，用来发送和接收音频数据。SPORT端口可以配置为立体声串行操作模式，用来发送和接收音频数据，它支持的操作模式包括I^2S模式、LEFT-JUSTIFIED模式、DSP模式，模式的选择可以通过设置相应的寄存器来完成。BF533与AD1836A的连接框图将在3.6节中给出。SPORT端口的引脚描述如表3-5所示(小写字母x表示0或1，即SPORT0或SPORT1)。

表3-5 SPORT端口引脚

引脚	类型	功能
DTxPRI	O	主数据发送引脚
DTxSEC	O	辅数据发送引脚
TSCLKx	I/O	数据发送时钟
TFSx	I/O	数据发送帧同步
DRxPRI	I	主数据接收引脚
DRxSEC	I	辅数据接收引脚
RSCLKx	I/O	数据接收时钟
RFSx	I/O	数据接收帧同步

3.1.4 串行外设接口

BF533处理器有1个SPI兼容的接口，能够使其与多个SPI兼容的设备通信。SPI接口包括：2个数据引脚(主输出-从输入MOSI和主输入-从输出MISO)和1个时钟引脚(串行时钟SCK)，1个SPI片选输入引脚(/SPISS)可使其它SPI设备选择处理器，7个SPI片选输出引脚(SPISEL[7:1])使处理器能够选择其它SPI设备。这些SPI片选引脚是功能复用的，因此也可以被重新配置为可编程标志引脚或PPI引脚。通过这些引脚，SPI端口提供了全双工的同步串行接口，支持主从模式和多主环境。

SPI的波特率和时钟的相位/极性都是可编程的，而且集成有一个DMA控制器，可配置为发送或接收数据流。SPI的DMA控制器在任意给定时间，

只能进行单向访问。

本系统中SPI接口与AD1836A相关引脚相连，用来配置AD1836A的寄存器。SPI接口的引脚描述如表3-6所示。

表3-6 SPI接口引脚

引脚	类型	功能	复用功能
MOSI	I/O	主输出从输入	N/A
MISO	I/O	主输入从输出	N/A
SCK	I/O	SPI 时钟	N/A
/SPISS	I	SPI 片选输入	可编程标志引脚0
SPISEL[7:4]	O	SPI 片选输出	可编程标志引脚[7:4]/PPI[15:12]
SPISEL[3:1]	O	SPI 片选输出	可编程标志引脚[3:1]

3.1.5 通用异步收发器

BF533处理器提供1个全双工的通用异步接收/发送(UART)端口，它与PC标准的UART完全兼容。UART端口为其它外设或主机提供了一个简化的UART接口，支持全双工、有DMA能力的异步串行数据传输。UART端口支持5至8个数据位，1或2个停止位，以及无校验、奇校验、偶校验位。UART 端口支持以下2 种模式的操作：

（1）PIO（已编程I/O）：处理器通过读/写I/O映射的UATX寄存器，发送或接收数据。在发送和接收时，数据都是双缓冲的。

（2）DMA（直接存储器访问）：DMA 控制器发送和接收数据。这就减少了与存储器传输数据所需的中断数量和频率。每个UART都有2个专用的DMA通道，一个用于发送，一个用于接收。这些DMA通道的默认优先级低于大多数DMA通道，因为其使用率相对较低。

本系统中UART端口与MAX232A连接，UART端口仅有两个引脚，一个为数据发送引脚，另一个为数据接收引脚。引脚描述如表3-7所示。

表3-7 UART端口引脚

引脚	类型	功能
RX	I	UART接收
TX	O	UART发送

3.1.6 可编程标志

BF533拥有16个双向可编程标志引脚，即PF[15:0]。每个管脚都可以通过标志方向寄存器（FIO_DIR）单独地配置为输入或输出方式。当配置为输出方式时，写入到标志置位（FIO_FLAG_S）和标志复位（FIO_FLAG_C）寄存器的状态值决定了由输出PFx管脚驱动的状态。读标志置位或标志复位寄存器时，不管配置成为输出还是输入状态，都会返回各个管脚的状态。

视频解码器SA7113H和视频编码器ADV7171需要使用I^2C总线来配置寄存器，但BF533系统本身没有I^2C接口，因此本系统使用PF引脚来模拟实现I^2C接口，用到了两个引脚：PF0，I^2C时钟输出引脚；PF1，I^2C数据输入输出引脚。另外，PF5或PF6引脚可作为中断输入引脚，用来接收以太网控制器DM9000AE的中断信号。16个PF引脚都是复用引脚，引脚描述如表3-8所示。

表3-8 可编程标志引脚

引脚	类型	功能	复用功能
PF[15:8]	I/O	可编程标志	PPI[4:11]
PF[7:4]	I/O	可编程标志	SPI片选输出[7:4]/PPI[12:15]
PF[3:1]	I/O	可编程标志	SPI片选输出[3:1]
PF0	I/O	可编程标志	SPI片选输入

3.1.7 时钟、引导模式、JTAG电路

1. 时钟电路

BF533通过一个外部有源晶振来获得持续的、稳定的时钟输入。外部时钟通过 CLKIN 引脚输入，然后经过片内的锁相环（PLL），可以得到精确的CLKIN倍频输出。在正常运行时，用户可以用CLKIN的倍频因子对PLL进行编程，倍频后的时钟信号就是内核时钟CCLK。

内核时钟CCLK按照一个可编程的分频比分频后，产生系统时钟SCLK。SCLK是外设访问总线（PAB）、DMA总线（DAB）、外部地址总线（EAB）、外部主机控制总线（EMB）和外部总线接口（EBIU）的时钟信号。为了优化性能和功耗，BF533允许内核时钟和系统时钟可以动态变化。所有的同步外设都

以SCLK为基准，例如，通用异步收发器的时钟速率由SCLK信号进一步分频得到。然而 BF533 的一些外设，如实时时钟（RTC）等，有自己的与 SCLK 异步的时钟信号。

本系统使用一个11.0592MHz的有源晶振作为CLKIN的输入。

2. 引导模式

BF533复位后，处理器通过采样BMODE[1:0]引脚的状态，然后执行相应的引导模式。BF533支持三种引导模式，如表3-9所示。

表3-9 引导模式

BMODE[1:0]	引导模式说明	执行起始地址
00	旁路引导ROM；从外部16位存储器执行	0x20000000
01	使用处理器片内引导ROM去引导8位或16位FLASH	0xEF000000
10	保留	0xEF000000
11	使用引导ROM去从SPI串行EEPROM配置和加载引到代码（EEPROM地址范围可以是8位、16位、24位）	0xEF000000

BMODE[1:0]=00的引导模式是旁路引导ROM，直接从外部存储器（通常为NOR FLASH）0x20000000地址处开始执行程序，本系统使用的就是这种引导模式。

3. JTAG电路

JTAG是一种国际标准测试协议（IEEE 1149.1兼容），JTAG主要应用于电路的边界扫描测试和可编程芯片的在系统编程。JTAG引脚的定义为：TCK为测试时钟输入；TDI为测试数据输入，数据通过TDI引脚输入JTAG接口；TDO为测试数据输出，数据通过TDO引脚从JTAG接口输出；TMS为测试模式选择，TMS用来设置JTAG接口处于某种特定的测试模式；TRST为测试复位，输入引脚，低电平有效。

ADI JTAG仿真器使用了标准JTAG接口的一个超集来发送和接收数据。JTAG仿真器使用了一个附加引脚/EMU，这个引脚给出了DSP仿真状态标志信号。BF533处理器的JTAG端口相应的也具有6个引脚，如表3-10所示。

表3-10 JTAG接口引脚

引脚	类型	功能
TCK	I	JTAG时钟
TDO	O	JTAG串行数据输出
TDI	I	JTAG串行数据输入
TMS	I	JTAG模式选择
/TRST	I	JTAG复位
/EMU	O	JTAG仿真状态标志

3.2 SDRAM 模块

SDRAM 芯片 HY57V561620CTP-H 的引脚描述如表 3-11 所示。

表 3-11 HY57V561620CTP-H 引脚描述

引脚	类型	功能
CLK	I	系统时钟输入。所有其它输入信号都在 CLK 信号上升沿被读取
CKE	I	时钟使能。控制内部时钟信号，并且当 SDRAM 不活动时，其状态处于掉电、挂起、自刷新之一
/CS	I	片选。使能或禁止除CLK、CKE、UDQM、LDQM之外的所有输入
BA[1:0]	I	bank 地址。所选 bank 在/RAS 有效时被激活，在/CAS 有效时被读写
A[12:0]	I	地址输入
/RAS, /CAS, /WE	I	分别为行地址选通，列地址选通，写使能
UDQM, LDQM	I	SDRAM 数据屏蔽引脚
DQ[15:0]	I/O	数据输入输出

对 SDRAM 进行寻址需要行地址、列地址以及 bank 地址，其中行地址和列地址总线是时分复用的， A[12:0]十三个引脚用于行地址输入，A[8:0]九个引脚用于列地址输入。A[12:0]分别与 BF533 ADDR[13:1]按照引脚大小序号对应相连，即 A0 与 ADDR1 相连，A1 与 ADDR2 相连，以此类推，但是由于 A10 引脚还具有自动预预充电功能,所以与 A10 引脚相连的不是 BF533 的 ADDR11 引脚，而是 SA10 引脚，即 BF533 的 ADDR11 引脚在此处未被使用。

HY57V561620CTP-H 连接框图如图 3-1 所示。

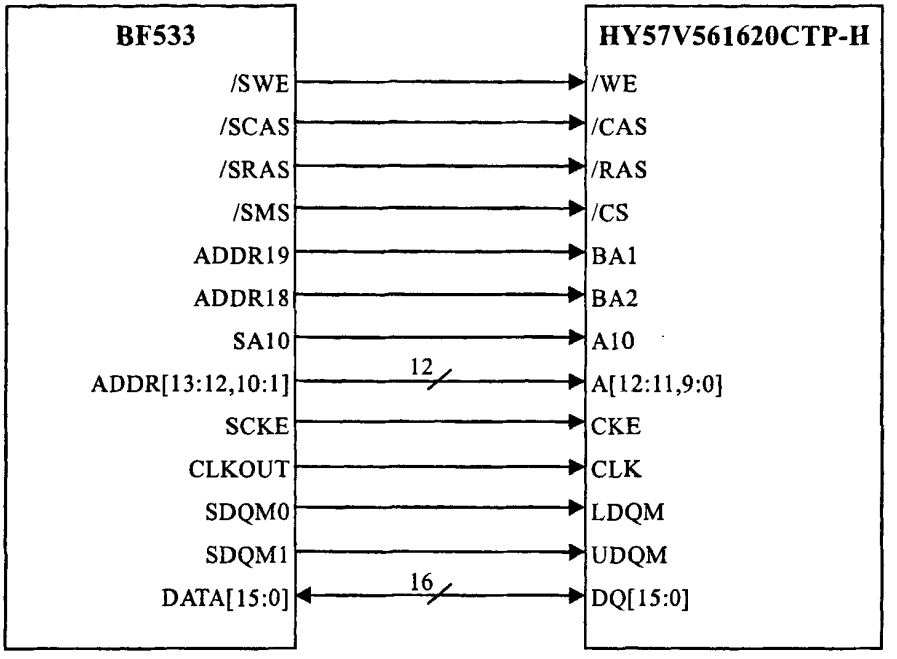


图 3-1 SDRAM 连接框图

3.3 FLASH 模块

FLASH 芯片 SST39VF1601 的引脚描述如表 3-12 所示。SST39VF1601 连接框图如图 3-2 所示。

SST39VF1601 容量为 1M×16bit，它包含了 1M 个地址单元，每个地址单元的大小为 16bit（2 字节），对它进行寻址需要 20 位的地址输入，但是 BF533 的外部地址总线为 19 位（即 ADDR[19:1]，分别与 SST39VF1601 的 A[18:0]相连），这只能对 SST39VF1601 的 1M 字节进行寻址，所以为了能充分利用整个 FLASH 存储器，需要一根补充的地址线来完成寻址任务。本系统引入了异步存储器 bank0 选择引脚/ASM0 作为补充地址输出引脚，/ASM0 与 SST39VF1601 的 A19 引脚相连。当处理器需要访问异步存储器 bank0 地址范围内的任意存储单元时，/ASM0 输出均为低电平，此时 A19 输入也为低电平，寻址的是 FLASH 低 1M 字节的存储空间；当处理器需要访问异步存储器 bank1 地址范围内的任意存储单元时，/ASM0 输出均为高电平，此时 A19 输入也为高电平，寻址的是 FLASH 高 1M 字节的存储空间。

SST39VF1601 存储器是映射在 bank0 和 bank1 两个块地址空间中的，也就是说无论处理器访问 bank0 或 bank1，最终都是对 SST39VF1601 的访问，所以对这两个 bank 进行访问时，SST39VF1601 芯片都必须被使能，即 CE# 输入为低电平。本系统中通过将 BF533 的/ASM0 和/ASM1 信号进行“与”逻辑操作，输出信号连接至 SST39VF1601 的片使能 CE#引脚，其中“与”逻辑操作是通过 CPLD 实现的。

表 3-12 SST39VF1601 引脚描述

引脚	类型	功能
A[19:0]	I	地址输入
DQ[15:0]	I/O	数据输入输出
WP#	I	写保护。当为低电平时，可以对FLASH低64K字节单元写保护
RST#	I	复位，并使芯片返回读模式
CE#	I	芯片使能。低电平使芯片处于活动状态
OE#	I	输出时能
WE#	I	写使能

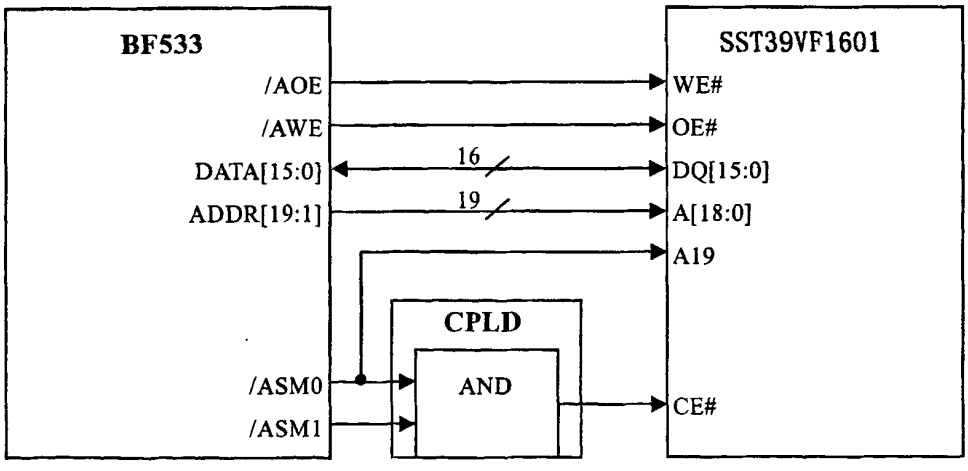


图 3-2 FLASH 连接框图

3.4 视频解码器模块

视频解码芯片SAA7113H引脚描述如表3-13所示。

本系统中SAA7113H的AI22引脚被用作模拟视频信号输入引脚，可以允许

输入模拟视频格式为PAL或NTSC制式的CVBS信号。SAA7113H通过VPO输出ITU56 YUV 4:2:2格式的视频信号，它与BF533的PPI接口相连， VPO具有8个数据输出引脚VPO[7:0],与PPI接口的PPI[7:0]相连。SAA7113H内部寄存器的配置是通过I^2C接口来完成的，BF533通过可编程标志接口来模拟I^2C接口，其中SAA7113H的SCL引脚与PF0引脚相连，SDA引脚与PF1引脚相连。芯片的I^2C地址根据RTS0引脚电平状态来决定，当RTS0为低平时（RTS0引脚接下拉电阻），I^2C写地址为48H，读地址为49H；当RTS0为高电平时（此为缺省状态，因为有一个内部上拉电阻），I^2C写地址为4AH，读地址为4BH。本系统中使用芯片缺省I^2C地址。

可通过在XTALI与XTAL之间接入一个无源晶振来产生SAA7113H内部所需要时钟；也可从外部输入所需时钟，只需要将外部的时钟连接至XTALI引脚，而XTAL引脚悬空即可。本系统将24.576MHz有源晶振输出接至XTALI引脚。

因为本系统没有使用边界扫描测试功能，所以/TRST与TCK引脚需要接地。另外，系统中只使用了AI22引脚来输入视频，其它3个引脚均需通过一个47nF电容串接至地。

SAA7113H连接框图如图3-3所示。

表 3-13 SAA7113H 引脚描述

引脚	类型	功能
AI11, AI12, AI21, AI22	I	模拟视频信号输入
VPO[7:0]	O	数字视频信号输出
AI1D	I	A11和A12的差分模拟输入
AI2D	I	A21 和 A22 的差分模拟输入
AOUT	O	模拟测试输出
SCL	I	I^2C 时钟输入
SDA	O	I^2C 数据输入输出
LLC	I	线性锁系统时钟输出（27MHz）
RCTO	I	实时控制输出
RTSO, RTS1	I	实时信号输出
TDO TDI, TCK, TMS, /TRST	O I	边界扫描测试
XTALI, XTAL	I	时钟输入
CE	I	芯片使能

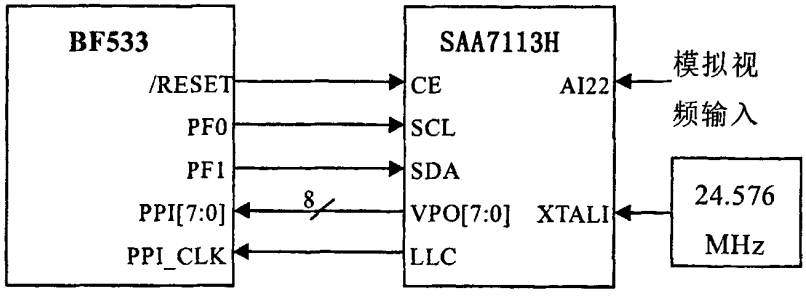


图3-3 SAA7113H连接框图

3.5 视频编码器模块

视频编码芯片 ADV7171 引脚描述如表 3-14 所示

为了与视频解码芯片 SAA7113H 使用同样的数据总线宽度，ADV7171 数字视频输入端口中 P[7:0]引脚与 BF533 的 P[7:0]引脚连接。输出模拟视频信号为 CVBS 格式，仅需使用 DAC A 一个引脚。ADV7171 也是通过 I^2C 总线来配置内部寄存器，它具有两组可供选择的 I^2C 地址，当 ALSB 引脚为低电平时，写地址为 0x54，读地址为 0x55；当 ALSB 引脚为高电平时，写地址为 0x56，读地址为 0x57。本系统中 ALSB 引脚接地，使用的是前一组 I^2C 地址。

ADV7171 需要 27MHz 的外部时钟输入，本系统中具有两个可供选择的 27MHz 时钟源，一个是为 27MHz 有源晶振，另一个是 SAA7113H 的 LLC 引脚输出的时钟信号。具体选择哪一个时钟，可以通过编程 CPLD 来实现。

未使用的数字视频信号输入引脚 P[8:15]需要接地，未使用的模拟视频信号输出引脚 DAC B, DAC C, DAC D 悬空即可。COMP 引脚需要通过一个 0.1 μF 电容接至 VAA。RSET 引脚需要通过一个 150Ω 电阻接至 GND。

ADV7171 连接框图如图 3-4 所示。

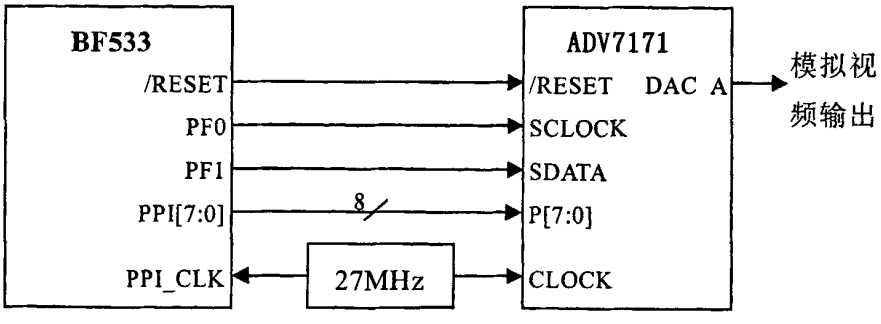


图 3-4 ADV7171 连接框图

表 3-14 ADV7171 引脚描述

引脚	类型	功能
P[15:0]	I	数字视频信号输入
DAC A, DAC B, DAC C, DAC D	O	模拟视频信号输出
/BLANK	I/O	视频消隐控制信号
ALSB	I	TTL 地址输入。决定了 I^2C 地址最低位
/RESET	I	复位片内时序发生器，将 ADV7171 设置为默认模式
SCLOK	I	I^2C 时钟输入
SDATA	I/O	I^2C 数据输入输出
COMP	O	补偿引脚
RSET	I	控制 DAC 输出的幅值
CLOCK	I	TTL 时钟输入

3.6 音频编解码器模块

音频编解码芯片 AD1836A 引脚描述如表 3-15 所示。

数字音频数据通过 SPORT 端口发送和接收，DAC 两个数据输入引脚 DSDATA1、DSDATA2 分别接至 BF533 的 DT0PRI、DT0SEC 引脚，DLRCLK、DBCLK 引脚接至 TFS0、TSCLK0 引脚；ADC 两个数据输出引脚 ASDATA1、ASDATA2 分别接至 BF533 的 DR0PRI、DR0SEC 引脚，ALRCLK、ABCLK 引脚接至 RFS0、RSCLK0 引脚。本系统中使用了 AD1836A 的 DAC1 和 DAC2 两个通道来输出模拟音频信号，DAC1 通道的输出引脚为 OUTLP1、OUTLN1、OUTRP1、OUTRN1，DAC2 通道的输出引脚为 OUTLP2、OUTLN2、OUTRP2、OUTRN2；ADC1 和 ADC2 两个通道用来输入模拟音频信号，ADC1 通道的输入引脚为 ADC1INLP、ADC1INLN、ADC1INRP、ADC1INRN，ADC2 通道的输入引脚为 ADC2INL1、ADC2INR1。AD1836A 内部寄存器是通过 SPI 总线进行配置的，CDATA、COUT、CCLK、CLATCH 分别与 BF533 的 MOSI、MISO、SCK、SPISEL4 相连。

AD1836A 正常工作所需时钟输入为 12.288MHz，由外部有源晶振提供。

AD1836A 连接框图如图 3-5 所示。

表 3-15 AD1836A 引脚描述

引脚	类型	功能
OUTLx, OUTRx	O	DAC模拟音频信号输出
ADCxINL, ADCxINR	I	ADC模拟音频信号输入
DSDATA1, DSDATA2, DSDATA3	I	数字音频信号输入
ASDATA1, ASDATA2	O	数字音频信号输出
DLRCLK	I/O	DAC左右声道选择时钟
DBCLK	I/O	DAC 位时钟。
ALRCLK	O	ADC左右声道选择时钟
ABCLK	O	ADC 位时钟。
COUT, CDATA, CCLK, /CLATCH	O I	SPI 端口
FILTD, FILTR	I	连接滤波电容。
/RST	I	复位（低电平有效）。
MCLK	I	时钟输入

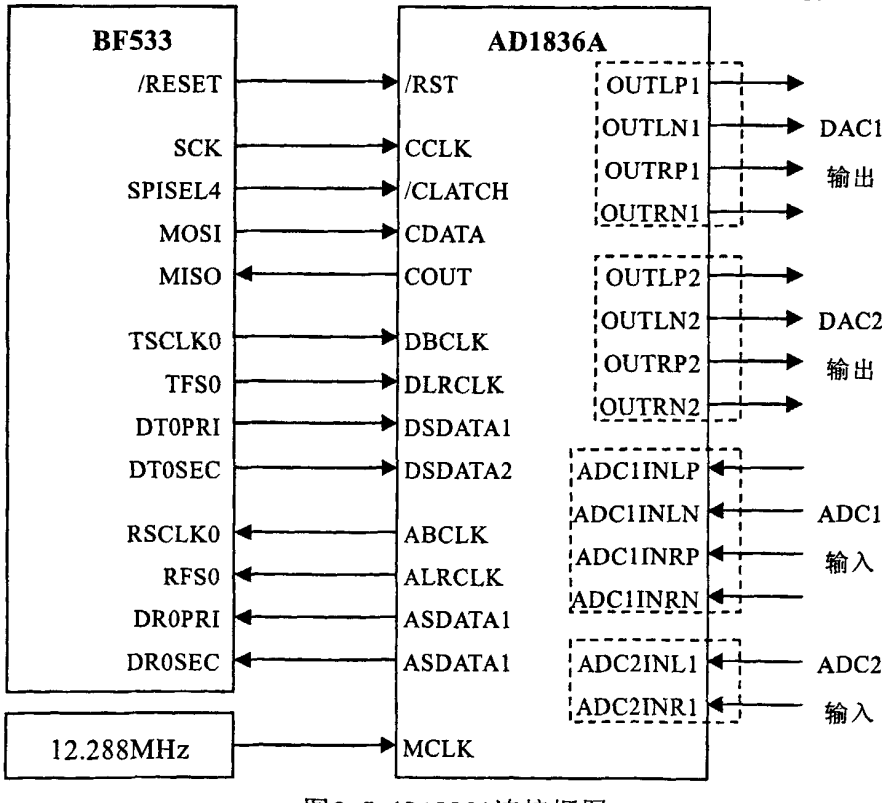


图3-5 AD1836A连接框图

3.7 以太网控制器模块

以太网控制芯片 DM9000AE 引脚描述如表 3-16 所示。

SD[15:0]是数据引脚，与 BF533 的数据引脚 D[15:0]相连。IOR#与 IOW#是读写控制信号引脚，分别接至 BF533 的/AOE 与/AWE 引脚。CS#是片选引脚，接至 BF533 的异步存储器片选引脚/ASM2。PWRST#是上电复位引脚，必须在系统复位时通过此引脚对 DM9000AE 进行复位，否则 DM9000AE 将工作异常，通过 CPLD 接至系统复位输入。CMD 引脚用来决定数据总线上传输的是数据还是地址，将其与 BF533 的 A2 地址线相连。中断请求信号 INT 与 BF533 的 PF5 引脚相连。TX+，TX-，RX+，RX-四个引脚用来接收和发送网络数据，连接至网络变压器 HS9016 相对应引脚。

EEDIO，EECK，EECS 三个引脚是 DM9000AE 的 EEPROM 接口，分别接至 93LC46 的数据引脚，时钟引脚，片选引脚。此外，EECK，EECS 引脚还有特殊的控制功能，EECK 引脚接上拉电阻时，DM9000AE 中断信号低电平有效，否则高电平有效；EECS 引脚接上拉电阻时，DM9000AE 工作在 8 位模式，否则工作在 16 位模式。

芯片需要 25MHz 时钟，由连接在 X1 与 X2 之间的无源晶振产生。

表 3-16 DM9000AE 引脚描述

引脚	类型	功能
SD[15:0]	I/O	数据输入输出
IOR#	I	读使能
IOW#	I	写使能
CS#	I	片选
CMD	I	命令类型输入
INT	O	中断请求
RX+, RX-, TX+, TX-	I/O	网络数据收发
EEDIO	I/O	EEPROM 数据输入输出
EECK	I	EEPROM 时钟
EECS	I	EEPROM 片选
X1, X2	I, O	时钟引脚
PWRST#	I	上电复位

DM9000AE连接框图如图3-6所示。

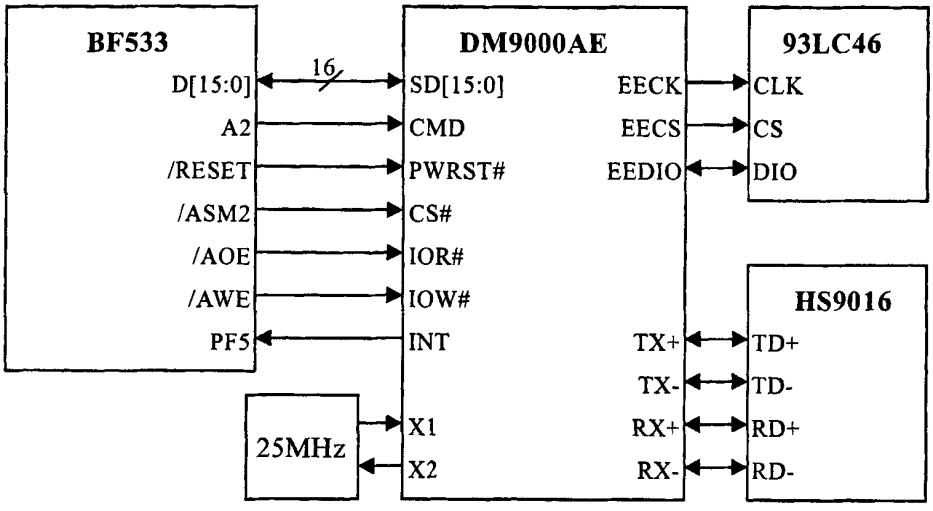


图 3-6 DM9000AE 连接框图

3.8 RS232 驱动器/接收器模块

MAX232A 内部结构基本可分 3 个部分：双路电荷泵 DC-DC 电压转换器、RS232 驱动器、RS232 接收器。

第一部分是电荷泵电路。由 1 脚（C1+）、2 脚（V+）、3 脚（C1-）、4 脚（C2+）、5 脚（C2-）、6 脚（V-）和 4 只外接 0.1μF 电容构成。功能是产生+12V 和-12V 两个电源，提供给 RS-232 串口电平的需要。

第二部分是数据转换通道。由 7、8、9、10、11、12、13、14 脚构成两个数据通道。其中 13 脚（R1IN）、12 脚（R1OUT）、11 脚（T1IN）、14 脚（T1OUT）为第一数据通道。8 脚（R2IN）、9 脚（R2OUT）、10 脚（T2IN）、7 脚（T2OUT）为第二数据通道。TTL/CMOS 数据从 T1IN、T2IN 输入转换成 RS-232 数据从 T1OUT、T2OUT 送到电脑 DP9 插头；DP9 插头的 RS-232 数据从 R1IN、R2IN 输入转换成 TTL/CMOS 数据后从 R1OUT、R2OUT 输出。

第三部分是供电。15 脚 GND、16 脚 VCC（+5v）。

本系统中使用了第一数据通道进行数据转换。

3.9 CPLD模块

本系统中XC9572XL实现了一系列拨动开关的功能。MAX708T的/RESET引脚连接至CPLD作为系统复位输入信号，然后可以根据系统实际需求，通过编程CPLD控制是否将系统复位信号接入DM9000AE、AD1836A、ADV7171芯片的相应复位引脚。SAA7113和ADV7171的行同步与场同步信号也接至CPLD，当应用中需要同步信号时，可以通过编程CPLD将它们与BF533的TIMER1和TIMER2引脚相连。BF533处理器PPI和ADV7171芯片均需要一个27MHz外部时钟输入，可以有两种选择，一个是27MHz有源晶振输出，另一个是SAA7113H的LLC输出。DM9000AE中断输出信号可以与PF5引脚相连。

系统中FLASH的片选信号是通过BF533的/ASM0和/ASM1信号相“与”来实现的，即/ASM0和/ASM1输入CPLD之后经过“与”逻辑操作后输出至FLASH片选CE#引脚。

3.10 电源模块

3.10.1 线性稳压电路

系统由一个5V/3A的外接电源进行供电，电源输入端接自恢复保险丝2920L300/15。系统中需要5V供电的电路，可以由此引入，经过相应的滤波即可使用。系统中所需的3.3V和1.8V电压需通过相应的线性稳压器来产生。

1. 3.3V电压产生电路

LT1529-3.3线性稳压器被用来获得3.3V电压。它具有5个引脚：输入（VIN）、输出（OUTPUT）、感应（SENSE）、关断（/SHDN）、地(GND)。5V电压输入VIN引脚；/SHDN引脚未被使用，悬空即可；SENSE引脚直接与OUTPUT引脚短接；OUTPUT引脚需要接一个22μF的电容以保证输出电压的稳定性。

2. 内核电压产生电路

BF533处理器内核正常工作所需电压为0.85V～1.3V。本系统设计了两组电路来提供内核电压，一组是提供0.85V～1.3V电压的可变电压稳压电路，

另一组是提供1.25V固定电压稳压电路。

可变电压稳压电路是由BF533片内调压器以及外围电路所组成的，外围电路主要是一个场效应管FDS9431A，其源极S接3.3V电压输入，漏极D通过一个10μH电感接至BF533内核电压输入引脚VDDINT，栅极G接至BF533的外部场效应管驱动引脚VROUT。

1.25V固定电压是通过LM317A产生的，LM317A是输出电压可变的3端线性稳压电路，VIN引脚接5V电压输入，VOUT为电压输出引脚，ADJ为输出电压调节引脚，VOUT与ADJ引脚之间接电阻R1，ADJ引脚与地之间接电阻R2，输出电压值由R2与R1的比值来决定，通常R1的值固定为240Ω，所以可以通过改变R2的值来获得不同的输出电压值，输出电压公式为V_{OUT}=1.25V(1 + R2/R1) + I_{ADJ}(R2)，由此公式可以得出当R2=0Ω时，输出电压为1.25V。

3.10.2 复位电路

表3-17 M708T引脚描述

引脚	类型	功能
/MR	I	手动复位输入，低有效。当/MR拉至0.6V时触发一个复位脉冲未用时接至VCC
/RESET	O	低有效复位输出。当 VCC 低于域值电平或/MR 保持为低时，/RESET 输出低电平。在复位条件结束后，/RESET 信号将继续保持 200mS
RESET	O	高有效复位输出。当VCC低于域值电平或/MR保持为低时，RESET输出高电平。在复位条件结束后，RESET信号将继续保持200mS
PFI	I	电源失效比较器输入。未用时接至GND
/PFO	O	电源失效比较器输出，低有效。当引脚PFI上的电压低于1.25V时，/PFO输出低电平。未用时必须悬空
VCC	P	电源输入
GND	G	地

本系统中复位信号是由 M708T 芯片产生的，M708T 引脚描述如表 3-17 所示。/MR 引脚接了一个按钮用来手动复位。复位信号输出使用的是/RESET 引脚，即当系统复位条件满足时，输出一个低电平复位信号。VCC 引脚接 3.3V 电源输入。RESET 与/PFO 引脚未被使用，悬空。/PFO 未被使用，接至 GND。

第4章 U-BOOT代码分析与移植

引导加载程序是系统上电后运行的第一段软件代码。回忆一下PC的体系结构我们可以知道，PC机中的引导加载程序由BIOS(其本质就是一段固件程序)和位于硬盘MBR中的OS Boot Loader(例如，LILO和GRUB等)一起组成。BIOS在完成硬件检测和资源分配后，将硬盘MBR中的Boot Loader读到系统的RAM中，然后将控制权交给OS Boot Loader[47]。Boot Loader的主要运行任务就是将内核映象从硬盘上读到RAM中，然后跳转到内核的入口点去运行，也即开始启动操作系统。

而在嵌入式系统中，通常并没有像BIOS那样的固件程序(有的嵌入式CPU也会内嵌一段短小的启动程序)，因此整个系统的加载启动任务就完全由Boot Loader来完成 。比如在一个基于BF533的嵌入式系统中，系统在上电或复位时通常都从地址 0x20000000处开始执行，而在这个地址处安排的通常就是系统的Boot Loader程序。

本系统中使用的Boot Loader是U-BOOT。

4.1 U-BOOT代码分析

大多数Boot Loader都分为stage1和stage2两大部分，U-BOOT也不例外。依赖于处理器体系结构的代码（如设备初始化代码等）通常都放在stage1且用汇编语言来实现，而stage2则通常用C语言来实现，这样可以实现复杂的功能，而且有更好的可读性和移植性。

4.1.1 U-BOOT stage1代码分析

U-BOOT第一阶段的代码主要包括两个汇编语言文件start.s和start1.s(这两个文件均位于cpu/bf533目录下)。

下面对BF533 U-BOOT第一阶段启动代码中的主要代码进行了分析。

1. 设置系统配置寄存器（SYSCFG）

```
R0 = 0x32;
SYSCFG = R0; /*bit 1 置 1，启用 64 位的内核时钟周期计数器*/
```

2. 检查软件复位寄存器（SWRST）状态位

```
p0 = SWRST;
r0.l = w[p0];
cc = bittst(r0, 15); /*检查位 15 的状态并赋给 cc，位 15 表明在这以前是否发生过软件复位*/
if !cc jump no_soft_reset;
r0 = 0x0000;
w[p0] = r0; /* 发生了软件复位，现将寄存器值清零*/
no_soft_reset:
nop;
```

3. 设置中断事件向量表

```
p0 = EVT_EMULATION_ADDR; /*EVT_EMULATION_ADDR 为中断向量起始地址*/
p0 += 8; /*此时 P0=0xFFE02008 正好是非屏蔽中断的入口地址寄存器*/
LSETUP(4,4) lc0 = 14; /*通过循环，将 EVT2—EVT15 初始化为 0*/
[ p0 ++ ] = 0; /*P0++即 P0=P0+4*/
```

4. 初始化 SDRAM 存储器

```
sp.l = (0xffb01000);
call init_sdram; /*设置内核时钟频率和系统时钟频率；将外部存储器相关配置寄存器设置为合适的值，包括异步存储器和同步存储器*/
```

5. 复制 U-BOOT 代码至 SDRAM 高 256KB 存储空间中

```
loop1: /*将程序从 flash 中复制到 RAM 中*/
r1 = [p1 ++ p3]; /*将 flash 中一个字的内容赋给 r1 寄存器*/
[p2 ++ p3] = r1; /*将 r1 中内容写入到 p2 所指向的 RAM 单元中*/
cc=p2==p4;
if !cc jump loop1; /*通过判断 p2 是否等于 p4，来决定复制是否完成*/
```

6. 由 FLASH 跳转至 SDRAM 中执行

```
    p0 = EVT15; /*EVT15 为 15 号中断地址寄存器*/
    p1 = _real_start;
    [p0] = p1; /*设置 15 号中断服务程序的起始地址为_real_start*/
    p0 = IMASK; /*IMASK 为内核中断屏蔽寄存器*/
    r0 = EVT_IVG15;
    [p0] = r0;
    raise 15; /*使能 15 号中断*/
    p0 = WAIT_HERE; /*WAIT_HERE 为 15 号中断服务程序返回地址*/
    reti = p0;
    rti;
WAIT_HERE: /*15 号中断服务程序返回处*/
    jump WAIT_HERE;
_real_start: /*15 号中断服务程序的起始处*/
    [ -- sp ] = reti; /*在栈中保存 15 号中断服务程序返回地址*/
```

此处需要讨论的一个问题是，为何_real_start 标号所代表的地址指向SDRAM 存储空间？因为_real_start 标号的地址是在链接时确定的，链接过程是由链接描述文件所控制的，在链接描述文件 uboot.lds 中，各代码段均被映射至 SDRAM 中，且 start.s 文件的代码段被映射至 CFG_MONITOR_BASE（SDRAM 中 UBOOT 程序起始地址），所以_real_start 标号地址为 SDRAM 地址。

7. 由 start.s 跳转至 start1.s 开始处

```
p0 = _start1; /*此标号是 start1.s 文件中的标号*/
jump (p0); /*由此跳转到 start1 文件的_start1 段*/
```

8. start1.s 文件中_start1 段

```
call  board_init_f; /*调用board_init_f函数，进入C程序*/
```

U-BOOT第一阶段汇编程序执行完毕之后，通过调用board_init_f函数，进入了第二阶段C语言代码的执行。

4.1.2 U-BOOT stage2 代码分析

U-BOOT 第二阶段的代码所包含的文件比较多，代码长度比较长，其执行可以分为 5 个子阶段来描述：

（1）由第一阶段进入 board_init_f 函数执行；

（2）在函数 board_init_f 的最后调用 board_init_r 函数，进入 board_init_r 函数执行；

（3）在 board_init_r 函数的最后进入 main_loop 函数的循环，main_loop 函数的执行产生两条分支，即第④步和第⑤步；

（4）继续保持在 main_loop 函数中执行。在 U-BOOT 的自启动命令执行前按下任意键，则进入 U-BOOT 的命令行，否则执行第⑤步；

（5）执行 bootcmd 环境变量所设定的自动运行命令，启动嵌入式操作系统，比如，通过 TFTP 从远程主机下载嵌入式操作系统，并引导其执行。

1. board_init_f函数

下面是board_init_f函数的主要工作代码：

```
gd = (gd_t *)(CFG_GBL_DATA_ADDR); /*为gd_t全局变量分配内存单元*/
addr = (CFG_GBL_DATA_ADDR + sizeof(gd_t));
bd = (bd_t *)addr; /*为bd_t全局变量分配内存单元*/
……
init_IRQ(); /*初始化中断设置*/
env_init(); /*初始化环境变量*/
init_baudrate(); /*初始化波特率设置*/
serial_init(); /*初始化串口设置*/
rtc_init(); /*初始化实时时钟*/
timer_init(); /*初始化定时器*/
……
board_init_r((gd_t *)gd, 0x20000010); /*调用board_init_r函数*/
```

2. board_init_r函数

board_init_r函数的主要工作代码如下所示：

```
size = flash_init(); /*FLASH初始化*/
```

```
……
mem_malloc_init(); /*初始化用于动态分配的堆内存*/
spi_init_f(); /*SPI 初始化*/
……
s = getenv("ethaddr"); /*获取ethaddr环境变量*/
bd->bi_ip_addr = getenv_IPaddr("ipaddr");
devices_init(); /*各种设备初始化，如LCD、USB等*/
jumptable_init(); /*将各种常用的函数指针保存在gd_t全局变量中*/
……
if ((s = getenv("loadaddr")) != NULL){
    load_addr = simple_strtoul(s, NULL, 16);
} /* loadaddr为操作系统加载地址，当加载指令不带参数时，需要用到此
地址*/
init_func_i2c(); /*I²C总线初始化*/
……
for (;;){
    main_loop(); /*由此进入主循环函数*/
}
```

3. main_loop函数

main_loop函数可以分为两个部分来分析，前半部分是执行自启动命令之前的代码；后半部分是在用户有按键动作（即按下键盘上任意键）不执行自启动命令，转而进入U-BOOT命令行循环的代码。

下面我们先分析前半部分的代码。

```
……
s = getenv ("bootdelay");
bootdelay = s ? (int)simple_strtol(s, NULL, 10) : CONFIG_BOOTDELAY;
……
s = getenv ("bootcmd");
if (bootdelay >= 0 && s && !abortboot (bootdelay)) {
    ……
    run_command (s, 0);
    ……
```

```
}
```

由上面的代码可知，U-BOOT先取出“bootdelay”环境变量，如果其值不为空，那么将其赋给bootdelay变量，否则将预定义宏的值赋给bootdelay变量。

s=getenv("bootcmd");取出“bootcmd”环境变量赋给s。接着if条件语句判断如果延迟时间bootdelay大于等于0，并且自启动命令环境变量bootcmd已被赋值，并且abortboot(bootdelay)函数返回0，那么就通过run_command(s, 0)函数执行bootcmd所定义的自启动命令，否则if条件不满足的情况下，继续执行main_loop函数余下的部分。其中abortboot函数在bootdelay所定义的时间内按下任意键，则立即返回1，否则在超时后返回0。

当上面所分析的if条件不满足时，继续执行main_loop函数后续部分代码：

```
for (;;){
    len = readline (CFG_PROMPT); /*读取用户输入的命令*/
    flag = 0; /*flag为命令可重复执行标志，0为不可重复，1为可重复*/
    if (len > 0)  /*有字符输入*/
        strcpy (lastcommand, console_buffer);
    else if (len == 0) /*无字符输入*/
        flag |= CMD_FLAG_REPEAT;
    if (len == -1) /*输入“Ctrl+C”*/
        puts ("<INTERRUPT>\n");
    else
        rc = run_command (lastcommand, flag);
    if (rc <= 0){
        lastcommand[0] = 0;
}}
```

上面代码的功能是使系统在U-BOOT的命令行下循环执行，即在命令提示符下等待命令输入，当用户输入命令之后，系统执行命令相关操作，执行完命令之后，又返回命令提示符，并继续等待下一条命令输入，如此往复执行下去。

readline函数的作用是从键盘读入用户输入的命令字符，并将其保存在console_buffer字符数组中，其返回值保存在len变量中。如果用户输入了命令，则返回命令所包含的字符数；如果用户直接输入回车键，则返回0；如果输入“Ctrl+C”组合键，则返回-1。

run_command函数功能是执行输入的命令。它有两个参数，lastcommand[0]

保存的是刚刚输入的命令字符（当len > 0时），或者是最近一次被执行的命令字符（len == 0时）；flag为命令可重复执行标志。它的返回值表示命令是否是有效的和可重复执行的，1表示有效且可重复执行命令，0表示有效但不可重复执行命令，-1表示无效命令。

4. do_bootm函数

当满足自启动的条件之后，main_loop函数转去就执行bootcmd环境变量中定义的自启动命令，其中最重要的一条命令就是“bootm 0x1000000”，0x1000000表示映像文件在SDRAM中存放的位置。do_bootm为bootm命令所对应的实现函数。此函数按执行顺序可以细分为7个部分来加以分析。

1) 获取映像文件加载地址

```
if (argc < 2){
   addr = load_addr;
} else{
   addr = simple_strtoul(argv[1], NULL, 16);
}
```

条件语句argc<2，表示bootm命令没有带任何参数时，映像存放地址就默认为load_addr，它是宏定义的缺省值；否则，如果bootm命令带有参数，那么将第一个参数作为存放地址。

2) 检查Magic Number的值

```
memmove (&header, (char *)addr, sizeof(image_header_t));
if (ntohl(hdr->ih_magic) != IH_MAGIC){
   puts ("Bad Magic Number\n");
   return 1;
}
```

首先将映像文件头部数据复制到image_header_t结构体类型变量header中，然后判断头部中Magic Number值与预定义值是否一样，如果不一样则输出错误提示信息，并从do_bootm函数返回。

3) 检查头部校验和

```
checksum = ntohl(hdr->ih_hcrc);
hdr->ih_hcrc = 0;
if (crc32 (0, (uchar *)data, len) != checksum){
   puts ("Bad Header Checksum\n");
```

```
    return 1;
}
```

从映像文件头部信息中取出头部校验和，并再次计算头部校验和的值，将二者进行比较，如果不一样则输出错误提示信息，并从do_bootm函数返回。

4) 检查映像文件是否适合在当前处理器中运行

```
#if defined(__bfin__)
    if (hdr->ih_arch != IH_CPU_BLACKFIN) /*本系统所用处理器架构*/
    …… /*省略了其它各种处理器的条件编译语句*/
```

上面这段代码的作用是检查映像文件是否支持当前处理器架构，比如本系统的处理器架构为Blackfin，如果映像文件不支持该处理器，即hdr->ih_arch的值不等于IH_CPU_BLACKFIN，就从do_bootm函数返回。

5) 检查映像文件类型是否支持

```
switch (hdr->ih_type){
case IH_TYPE_STANDALONE:
    ……
case IH_TYPE_KERNEL:
    ……
case IH_TYPE_MULTI:
    ……
}
```

U-BOOT支持三种映像文件类型，如果头部信息中包含的文件类型不是这三种之一，则输出错误提示信息，并从do_bootm函数返回。

6) 解压缩映像文件

```
switch (hdr->ih_comp) {
case IH_COMP_NONE:
    ……
case IH_COMP_GZIP:
    ……
case IH_COMP_BZIP2:
    ……
}
```

解压缩处理当前支持三种格式，包括未压缩格式、gzip压缩格式、bzip2压

缩格式。如果映像文件为未压缩格式时，只需要比较映像文件的存放地址addr是否等于加载地址hdr->ih_load，如果不一致，就需要将映像文件从地址addr复制到地址hdr->ih_load处，否则如果一致就不需要进行任何操作。如果映像文件格式为两种压缩格式之一时，就会执行相应的压缩算法，将映像文件解压至加载地址，当解压操作失败时，则输出解压失败信息，并复位系统。如果映像文件为未知压缩格式，则输出错误提示信息，并从do_bootm函数返回。

7) 根据操作系统的类型进入相应的启动函数

```
switch (hdr->ih_os) {
default:          /* handled by (original) Linux case */
case IH_OS_LINUX: /*本系统使用的μClinux属于此类型*/
      do_bootm_linux  (cmdtp, flag, argc, argv,
              addr, len_ptr, verify);
      break;
…… /*省略了其它各种操作系统的条件编译语句*/
}
```

根据操作系统内核头部信息来判断操作系统的类型，U-BOOT支持多种操作系统，包括Linux、NetBSD、RTEMS、VxWorks等，对于未知类型的操作系统，默认当做Linux类型。本系统使用的操作系统为μClinux，它属于Linux类型的操作系统，所以do_bootm_linux函数被执行。do_bootm_linux函数的主要功能就是跳转至操作系统入口地址处，开始操作系统的执行。

4.2 U-BOOT 移植

本系统中U-BOOT通过网络引导操作系统启动，所以首先必须编写以太网控制器芯片的驱动，其次为了在U-BOOT命令行中设置环境变量（比如自启动命令、IP地址等）并将其保存至FLASH中，还需要编写FLASH驱动。另外U-BOOT移植还需要添加和修改相关头文件。本文中U-BOOT版本为号为1.3.2。

4.2.1 以太网控制器驱动

本系统使用的以太网控制器为DM9000AE，它的数据读写是通过两个对外可直接访问的寄存器来进行，其中INDEX port寄存器的地址=主控芯片片选基地址

+0x0，DATA port寄存器的地址=主控芯片片选基地址+0x4。本文中系统将BF533的异步片选信号/ASM2连接至DM9000AE的片选引脚，因此本系统中INDEX port=0x20200000，DATA port= 0x20200004。INDEX port存放所需访问的寄存器的地址，DATA port存放被写入或被读出的数据。DM9000AE的写入和读出基本操作函数如下所示：

```
#define DM9000AE_IO  0x20200000
#define DM9000AE_DATA  0x20200004
#define DM9000AE_outb(d,r)  (*(volatile u8 *)r = d)
#define DM9000AE_inb(r)  (*(volatile u8 *)r)
void DM9000AE_ior(int reg) /*读出寄存器reg的值*/
{
   DM9000AE_outb(reg, DM9000AE_IO);
   return DM9000AE_inb(DM9000AE_DATA);
}
void DM9000AE_iow(int reg, u8 value) /*将value的值写入寄存器reg*/
{
   DM9000AE_outb(reg, DM9000AE_IO);
   DM9000AE_outb(value,DM9000AE_DATA);
}
```

1. DM9000AE初始化

在通过DM9000AE进行网络数据收发操作之前，首先必须对DM9000AE进行相关的初始化操作。

1）上电内部PHY

内部PHY默认情况下是关闭的，可以通过对GPR寄存器（1FH）置0使其工作。

```
DM9000AE_iow(DM9000_GPR, 0x00);
```

2）软件复位DM9000AE

对网络控制寄存器NCR（00H）位0置1可以产生软件复位，需要至少10μs时间完成复位操作。

```
DM9000AE_iow(DM9000AE_NCR, 0x01);
udelay(10);
```

3）通过外部EEPROM设置MAC地址

读入外部EEPROM中保存的MAC地址，并将其写入物理地址寄存器PHR（10H～

15H)。

```
for (i = 0; i < 6; i++)
    ((u16 *)bd->bi_enetaddr)[i] = read_srom_word(i);
```

4) 清零网络状态寄存器NSR (01H) 和中断状态寄存器ISR (FEH)

```
DM9000AE_iow(DM9000AE_NSR, NSR_WAKEST | NSR_TX2END | NSR_TX1END);
DM9000_iow(DM9000AE_ISR, 0x0f);
```

5) 使能数据接收发送中断及使能指针自动返回功能

使能数据接收发送中断后，会在接收到数据或者发送完数据之后产生中断，可以通过对中断屏蔽寄存器IMR (FFH) 位0和位1置1来使能接收和发送中断。指针自动返回功能是指当DM9000AE内部的读或写SRAM的地址指针值超过读或写的地址范围时，自动将地址指针复位为读或写SRAM的起始地址值，可通过对IMR寄存器位7置1使能指针自动返回功能。

```
DM9000AE_iow(DM9000AE_IMR, IMR_PRI | IMR_PTI | IMR_PAR);
```

6) 使能数据接收功能

在系统上电复位之后，DM9000AE的数据接收功能是禁止的，可通过对数据接收控制寄存器RCR (FFH) 位0置1使能数据接收功能，并且还可以对位4置1使能CRC错误检测功能，对位5置1使能包长检测功能（丢弃报长大于1522字节的包)。

```
DM9000AE_iow(DM9000AE_RCR, RCR_DIS_LONG | RCR_DIS_CRC | RCR_RXEN);
```

7) 网络连接的建立

DM9000AE具有自动协商功能，当上电后，它就会自动去探测与之连接的对端网络设备的工作方式（最大连接速度、全双工或半双工等)，并与对方进行协商，以建立性能最高的连接。因为自动协商完成连接的建立需要一定的时间，所以可以设定一个合适的时间，在此时间内每隔一段固定时间就去检查自动协商是否完成，如果超过了规定的时间自动协商没有完成，就说明无法建立连接。可以通过检查PHY寄存器组的基本模式状态寄存器BMSR的位5来查看自动协商是否完成。

```
while (!(phy_read(1) & 0x20)) {
      udelay(1000); /*每隔1000μs检查一次*/
      i++;
      if (i == 20000) { /*如果超过20s连接没有建立就返回*/
         printf("could not establish link\n");
```

```
        return 0;
}}
```

至此，DM9000AE初始化工作就完成了，接着就可以进行数据发送和接收了。

2. 数据发送

DM9000AE数据的发送由下面5个步骤组成。

1) 检查数据宽度

检查DM9000AE使工作在16位模式还是8位模式，可查看中断状态寄存器ISR位7，如果为1表示工作在16位模式，为0工作在8位模式。

```
(u8)io_mode = DM9000AE_ior(DM9000AE_ISR)>>7;
```

2) 写入待发送数据包至SRAM

DM9000AE在发送一个数据包之前，此数据包须首先写入DM9000AE的发送SRAM中，该SRAM在MAC中的地址为0～0xBFF，大小为3KB。通过MWCMD寄存器（F8H）可以将需要发送的数据写入至SRAM。每次写操作后，写指针都会自动增加1个或2个字节（取决于工作模式）。

```
DM9000AE_outb(DM9000AE_MWCMD, DM9000AE_IO);
if(io_mode == 1) /*8 位模式*/
for (i = 0; i < TX_length; i++) /*TX_length 为待传输的数据包长度*/
    DM9000AE_outb(TX_data[i], DM9000AE_DATA); /*将数据写入 SRAM*/
else if(io_mode == 0) { /*16 位模式*/
    Length_tmp = (TX_length+1)/2;
for (i = 0; i < Length_tmp; i++)
    DM9000AE_outw((u16 *)TX_data[i], DM9000AE_DATA);}
```

3) 设置待发送数据包的长度

在发送数据之前还需要将其长度设置在发送包长度寄存器中，长度值的低字节存在TXPLL（FCH）中，高字节存在TXPLH（FDH）中。

```
DM9000AE_iow(DM9000AE_TXPLL, TX_length & 0xff);
DM9000AE_iow(DM9000AE_TXPLH, (TX_length >> 8) & 0xff);
```

4) 发送数据包

将传输控制寄存器TCR（02H）位0置1，即开始发送SRAM中待发数据包。

```
DM9000AE_iow(DM9000AE_TCR, 0x1);
```

5) 检查发送是否完成

检查传输控制寄存器TCR的位0是否为0，如果为0就表示传输完成，否则如

果为1表示传输还在继续，继续等待传输完成。当传输超时后，就输出超时信息，并立即结束此次传输。

```
tmo = get_timer(0) + 5 * CFG_HZ; /*设置50ms超时时间*/
while (DM9000AE_ior(DM9000AE_TCR) & TCR_TXREQ) {
      if (get_timer(0) >= tmo) {
         printf("transmission timeout\n");
         break;
}}
```

3. 数据接收

1) 检查接收SRAM中是否接收到数据

为了检测DM9000AE是否接收到了数据包，可以采用轮询的方式来检测数据到达状况，具体的方法是通过MRCMDX寄存器（F0H）读取SRAM中数据，然后根据其第一个字节的值来判断是否接收到了数据包，如果其值为1则表示即收到了数据包，为0则表示没有收到包，为其它值则表示DM9000AE处于异常状态，需要复位。

```
DM9000AE_ior(DM9000AE_MRCMDX);
RX_ready = DM9000AE_inb(DM9000AE_DATA); /*读取数据到达标志 */
if (RX_ready == 0)
   return 0; /*如果没有数据到达就返回调用此函数的原函数*/
if (RX_ready > 1)
   reset(); /*当RX_ready大于1，DM9000AE处于异常状态，需要复位*/
```

2) 检查接收到的数据包状态和长度

通过MRCMD寄存器（F2H）读取SRAM中数据状态和长度字，SRAM读指针在每次读操作后都会自动增加。

```
(u8)io_mode = DM9000AE_ior(DM9000_ISR)>>7; /*检查工作模式*/
DM9000AE_outb(DM9000AE_MRCMD, DM9000_IO); /*在设置了寄存器MRCMD
后，就可以把要接收的数据从SRAM中读出，并且读指针会自动增加*/
if (io_mode == 1){ /*8位模式*/
    RX_status=DM9000AE_inb(DM9000AE_DATA)
            +DM9000AE_inb(DM9000AE_DATA)<<8;
    RX_length=DM9000AE_inb(DM9000AE_DATA)
            +DM9000AE_inb(DM9000AE_DATA)<<8;
```

```
}
else if (io_mode==0){ /*16位模式*/
    RX_status = DM9000AE_inw(DM9000AE_DATA);
    RX_length = DM9000AE_inw(DM9000AE_DATA);
}
```

3) 接收数据包

每进行一次读操作后，SRAM读指针自动增加相应模式长度。

```
if (io_mode==1) /*8位模式*/
    for (i = 0; i < RX_length; i++)
        RX_data[i] = DM9000AE_inb(DM9000AE_DATA);
else if (io_mode == 0){ /*16位模式*/
    Length_tmp = (RX_length+1)/2;
    for (i = 0; i < Length_tmp; i++)
        (u16 *)RX_data[i] = DM9000AE_inw(DM9000AE_DATA);
}
```

4.2.2 FLASH驱动

本系统使用的FLASH为SST39VF1601，FLASH的主要操作包括数据读、写、擦除以及产品标识等。FLASH的读操作很简单，直接寻址所需读取的内存单元即可，不需要任何附加操作。除了读操作之外，其它操作都需要通过一系列命令序列来完成。下面将会说明写操作、擦除操作、读取产品标识操作相关函数的编写。在此先给出一个在写操作、擦除操作中都会用到的函数Check_Toggle_Ready，它的主要功能就是检查写操作或擦除操作是否完成，通过不断读取数据引脚DQ6的值，如果相邻两次读取的值不一样则表示操作未完成，否则如果一样则表示操作完成。

```
typedef  unsigned int  WORD;
typedef  unsigned long int  Uint32;
#define sysAddress(offset) ((WORD *)(system_base + offset))
int Check_Toggle_Ready (Uint32 Dst)
{
   WORD PreData;
```

```
    WORD CurrData;
    unsigned long TimeOut = 0;
    PreData = *sysAddress(Dst);
    PreData = PreData & 0x0040; /*读取 DQ6 的*值/
    while (TimeOut < 0x07FFFFFF){ /*检查是否超时*/
        CurrData = *sysAddress(Dst);
        CurrData = CurrData & 0x0040; /*读取 DQ6 的值*/
        if (PreData == CurrData){
                return TRUE;
        }
        PreData = CurrData;
        TimeOut++;
    }
    return FALSE;
}
```

1. 写操作

SST39VF1601一次写操作完成一个字单元的写入，写操作需要写入4条命令，可以根据datasheet中给出的命令序列写出相应的写操作函数，需要注意的是datasheet中给出的每个命令的地址是字地址，即它所代表的存储单元是16位的，所以在程序中需要对此地址值乘2。

```
int Write(WORD *SrcWord, Uint32 Dst)
{
    Uint32 DestBuf = Dst; /*Dst为欲写入的FLASH地址*/
    WORD *SourceBuf = SrcWord; /*SrcWord数据当前存放地址*/
    int ReturnStatus; /*操作是否成功*/
    *sysAddress(0x5555*2) = 0x00AA; /*0x00AA写入地址0x5555*2处*/
    *sysAddress(0x2AAA*2) = 0x0055; /*0x0055写入地址0x2AAA*2处*/
    *sysAddress(0x5555*2) = 0x00A0; /*0x00A0写入地址0x5555*2处*/
    *sysAddress(DestBuf) = *SourceBuf; /*数据写入FLASH指定地址*/
    ReturnStatus = Check_Toggle_Ready(DestBuf); /*等待操作完成*/
    return ReturnStatus;
}
```

2. 擦除操作

SST39VF1601提供了三种擦除操作，包括整片擦除、Block擦除、Sector擦除。整片擦除就是对FLASH的所有存储单元都进行擦除操作；Block擦除是对某个指定Block的64KB大小存储单元进行擦除操作；Sector擦除是对某个指定Sector的4KB大小存储单元进行擦除操作。

因为三种擦除操作都需要写入6条命令，并且前5条命令是一样的，所以可以通过一个函数来实现三种擦除操作，根据传入参数的不同就会执行不同的擦除操作。函数定义两个参数Data和Dst， Data为第六条命令的数据，Dst为第6条命令的地址。如果Data=0x10，Dst=0x5555*2，即为整片擦除；如果Data=0x50，即表示对以Dst为起始地址的Block进行擦除操作；如果Data=0x30，即表示对以Dst为起始地址的Sector进行擦除操作。

```
int Erase(Uint32 Data, Uint32 Dst)
{
   Uint32 DestBuf = Dst;
   int  ReturnStatus;
   /*发出命令序列*/
   *sysAddress(0x5555*2) = 0x00AA; /*0x00AA写入地址0x5555*2处*/
   *sysAddress(0x2AAA*2) = 0x0055; /*0x0055写入地址0x2AAA*2处*/
   *sysAddress(0x5555*2) = 0x0080; /*0x0080写入地址0x5555*2处*/
   *sysAddress(0x5555*2) = 0x00AA; /*0x00AA写入地址0x5555*2处*/
   *sysAddress(0x2AAA*2) = 0x0055; /*0x0055写入地址0x2AAA*2处*/
   *sysAddress(DestBuf) = Data; /*根据传入参数进行相应操作*/
   ReturnStatus = Check_Toggle_Ready(DestBuf); /*等待操作完成*/
   return ReturnStatus;
}
```

3. 产品标识查询操作

通过该操作可以获取制造商标识号和芯片标识号。

```
void Check_SST_39VF1601(WORD* SST_id1, WORD* SST_id2)
{
    /*发出标识号查询命令序列*/
    *sysAddress(0x5555*2) = 0x00AA; /*0x00AA写入地址0x5555*2处*/
```

```
    *sysAddress(0x2AAA*2) = 0x0055; /*0x0055写入地址0x2AAA*2处*/
    *sysAddress(0x5555*2) = 0x0090; /*0x0090写入地址0x5555*2处*/
    udelay(1);
    *SST_id1  = *sysAddress(0x0000*2); /*读取制造商标识号*/
    *SST_id2  = *sysAddress(0x0001*2); /*读取芯片标识号*/
    /*退出标识号查询模式，返回标准读模式*/
    *sysAddress(0x5555*2) = 0x00AA; /*0x00AA写入地址0x5555*2处*/
    *sysAddress(0x2AAA*2) = 0x0055; /*0x0055写入地址0x2AAA*2处*/
    *sysAddress(0x5555*2) = 0x00F0; /*0x00F0写入地址0x5555*2处*/
    udelay(1);
}
```

4.2.3 U-BOOT 移植的具体过程

本系统U—BOOT移植所做的工作包括如下几点：

- 修改顶层（根目录下）Makefile文件。
- 在include/configs/目录下创建开发板配置文件： mybf533.h。
- 在boards目录下创建开发板特定目录mybf533，并加入flash驱动程序。
- boards/net/目录下添加 DM9000AE 驱动程序文件，并修改该目录下的Makefile文件。
- 修改include/asm-blackfin/目录下mem_init.h文件
- 修改lib_blackfin目录下board.c文件

下面说明具体的修改内容：

（1）顶层Makefile 定位至BFIN_BOARDS变量声明处，并在此处加入本开发板的名称，如下所示：

```
BFIN_BOARDS += mybf533
```

（2）以评估板的ezkit533.h文件为模板创建mybf533.h文件，然后进行相应的修改：

外部输入时钟频率更改为：

```
#define CONFIG_CLKIN_HZ  11059200 /*外部时钟源频率为11.0592MHz*/
```

修改网络驱动设置，将ezkit533开发板的SMC91111驱动设置删除，添加

本开发板的DM9000AE驱动设置：

```
#define CONFIG_DRIVER_DM9000AE  1
#define CONFIG_DM9000AE_BASE  0x20200000
#define DM9000AE_IO  0x20200000
#define DM9000AE_DATA  0x20200004
#define CONFIG_DM9000AE_USE_16BIT  1
```

异步存储器总线相关设置更改为：

```
#define AMGCTLVAL  0xFF /*全局控制寄存器*/
#define AMBCTL0VAL  0xFFC2FFC2 /*bank0 和 bank1 控制寄存器*/
#define AMBCTL1VAL  0xFFC2FFC2 /*bank3 和 bank4 控制寄存器*/
```

SDRAM相关设置更改为：

```
#define CONFIG_MEM_SIZE  32 /*容量大小*/
#define CONFIG_MEM_ADD_WDTH  9 /*列地址宽度*/
#define CONFIG_MEM_HY57V641620HG  1 /*SDRAM 型号为 HY57V641620HG*/
```

（3）以评估板的ezkit533目录为模板创建mybf533目录，仅需对目录中flash.c文件进行修改，将flash.c文件的内容替换成本系统的flash驱动程序，flash驱动程序的编写请参见4.2.2所述。

（4）在boards/net/目录下添加DM9000AE驱动程序相关文件dm9000ae.c和dm9000ae.h，DM9000AE驱动程序的编写请参见4.2.1所述。还需对该目录下的Makefile文件进行修改该，添加如下一条语句：

```
COBJS-y += dm9000ae.o
```

（5）修改include/asm-blackfin/目录下mem_init.h文件，首先将第一条if语句修改为：

```
#if (CONFIG_MEM_MT48LC16M16A2TG_75 || CONFIG_MEM_MT48LC64M4A2FB_7E
    || CONFIG_MEM_MT48LC16M8A2TG_75 || CONFIG_MEM_HY57V641620HG)
```

然后添加如下语句：

```
#if (CONFIG_MEM_HY57V641620HG)
        #define SDRAM_Tref  64 /*刷新周期 */
        #define SDRAM_NRA  8192 /*行地址数*/
        #define SDRAM_CL  CL_2 /*列地址选通脉冲时间延迟*/
#endif
```

（6）对lib_blackfin目录下board.c文件进行如下修改：

添加头文件包含语句：

```
#include "../drivers/net/dm9000ae.h"
```

在board_init_r函数中添加如下语句：

```
#ifdef CONFIG_DRIVER_DM9000AE
   eth_init(bd); /*DM9000AE初始化函数*/
#endif
```

此外，在移植U-BOOT时还有一个关于TFTP安全的问题需要考虑。由于TFTP标准实现中不提供任何验证机制，因此只需要知道TFTP服务器的地址就可以对服务器进行读写操作，这样就会对服务器造成安全隐患，假如某人在访问TFTP服务器时不小心或者有意将目标系统需要的操作系统镜像文件删除，那么就会造成目标系统无法从主机下载镜像文件，从而间接导致了目标系统无法正常运行。为了防止这种情况的出现，一般比较简单的办法就是修改TFTP服务端口号，但是这还不足以抵御风险，因为还是可以通过端口扫描工具找到TFTP服务端口，然后对TFTP进行非法操作。为了彻底解决这一问题，本文对TFTP协议实现代码做了一些改动。

U-BOOT中TFTP源代码位于/net/tftp.c文件中，将TFTP端口号以及操作码进行更改：

```
#define WELL_KNOWN_PORT 2000 /*原端口号为69*/
#define TFTP_RRQ  7 /*读请求，原值为1*/
#define TFTP_WRQ  8 /*写请求，原值为2*/
#define TFTP_DATA  9 /*数据，原值为3*/
#define TFTP_ACK  10 /*响应，原值为4*/
#define TFTP_ERROR  11 /*错误提示，原值为5*/
```

为了在TFTP包中增加验证信息，可在读写请求包的包选项中添加验证信息字段，即在TftpSend函数中添加如下代码：

```
case STATE_RRQ: /*此处以读请求为例，写请求加入一样的代码*/
……
strcpy ((char *)pkt, vali_id); /*验证信息字段*/
pkt += strlen(vali_id) + 1; /*计算包指针当前位置*/
```

在include/net.h中添加如下变量定义：

```
extern char  vali_id [20]; /*保存用户输入的验证信息*/
```

为了把验证信息作为tftp的一个命令行参数，还需要对common/cmd_net.c

文件进行修改。首先将tftp命令参数数量改为4：

```
U_BOOT_CMD(tftpboot, 4, 1, do_tftpb,
   "tftpboot- boot image via network using TFTP protocol\n",
   "[loadAddress] [[hostIPaddr:]bootfilename] [validate ID]\n");
```

然后在netboot_common函数中添加如下代码：

```
switch (argc){
......
case 4:    load_addr = simple_strtoul(argv[1], NULL, 16);
           copy_filename (BootFile, argv[2], sizeof(BootFile));
           strcpy (vali_id, argv[3], sizeof(vali_id))
break;
......
}
```

除了需要对U-BOOT中TFTP相关源代码进行修改外，TFTP服务器端源代码也需要作出相应的修改，修改的内容与U-BOOT中是一样的，此处就不再说明。

经过以上步骤，U-BOOT的移植工作就完成了。本文在5.2节中给出了U-BOOT编译和加载的具体操作方法。

第5章 系统硬件调试及U-BOOT编译与加载

本章将对BF533、SDRAM、FLASH、CPLD、以太网控制器以及视频解码器的硬件电路进行测试。接着建立交叉编译环境，编译U-BOOT并将其下载至开发板中，并实现U-BOOT从网络加载μClinux操作系统。

5.1 系统硬件调试

在制作完PCB印制板之后，可以先焊接好电源模块相关电路。然后测试PCB上其它各芯片的电源引脚处的电压值，确保系统供电正常。经测试一切正常之后，就可以焊接电路中其它模块了。电路焊接好之后，就需要对硬件各功能模块进行调试，以确保各模块能正常工作。系统硬件的调试顺序一般为处理器→SDRAM→其它功能模块。另外，调试硬件电路通常都需要用到相关的软件调试工具，本系统使用的调试工具为ADI VisualDSP++ 5.0集成开发环境。

5.1.1 BF533的调试

上电之前，确保内核电压（VDDINT）选择跳线被连接，此处接固定的1.25V电压，此外也可以将其接在可调电压电路上实现动态电压调节，另外，还需要把blackfin仿真器连接至目标板上。准备好之后就可以上电了，为了检查BF533工作是否正常，打开VisualDSP++集成开发环境，创建新的VisualDSP++会话（session），并正确设置会话相关属性，确保选择的处理器类型是BF533，连接类型是Emulator，仿真器类型是HPPCI-PCI Emulator（High Performance PCI emulator），设置好这些参数之后，就可以通过仿真器连接目标板了，如果处理器工作正常的话，就会进入到处理器的仿真模式，这也是检验DSP是否正常工作的简便方法。如果VisualDSP++集成开发环境出现错误对话框，提示连接目标板失败，那么就要检查是什么原因引起仿真器连接失败。

对于连接失败的情况，可以首先检查一下仿真器与目标板连接的是否正确，特别需要注意仿真器插头接反的情况。如果连接没有问题，还可以检查一下仿真器的驱动程序是否安装正确，可以在Windows的设备管理器中查看仿真器驱动程序是否已被安装以及状态是否正常。

5.1.2 SDRAM和FLASH的调试

在 VisualDSP++可以很方便的对 SDRAM 进行读写操作，选择菜单栏中的“Memory->BLACKFIN Memory”，在集成开发环境窗口右侧出现一个存储器窗口，输入任何 SDRAM 有效地址，就会该窗口中显示输入地址单元存储的值，可以直接双击该值来输入新值，回车后就会将新值写入 SDRAM。对于 FLASH 而言，可以直接对其进行读操作，但是写操作必须通过写命令序列来完成，具体可以参见 4.2.2 所述。

要验证存储器是否正常工作，可以通过对其中连续 2 个字节写入 0xAA 并回读，然后再输入 0x55 并回读，如果都正确，则表明存储器工作正常，否则若其中的某一位或几位数据出现错误，则很有可能是由于对应的数据线不通或连接错误所引起的。

5.1.3 CPLD的调试

本系统中多个芯片的正常工作都需要 CPLD 的支持。本系统使用 Xilinx ISE 9.1i 集成开发环境对 XC9572XL 编程。XC9572XL 与其他芯片的连接情况及需要实现的功能，详见 3.9 所述。XC9572XL 的编程过程如下所述。

首先打开ISE集成开发环境，新建一个名为mybf533的工程，并在工程中新建一个类型为Verilog的源文件，文件名为mybf533.v，接着就可以在此源文件窗口中输入Verilog程序了。程序如下所示：

```
module mybf533(AMS0, AMS1, SAA7113_27M, SYSRESET, AMS00,
                E_RESET,FLASH_A19, LED, PPI_CLK,  SAA7113_CE);
    input AMS0; /*BF533异步存储器bank0选择信号输入*/
    input AMS1; /*BF533异步存储器bank1选择信号输入*/
    input SAA7113_27M; /*SAA7113 LLC时钟信号输入*/
    input SYSRESET; /*MAX708T的系统复位信号输入*/
   output AMS00; /*FLASH片选信号输出*/
   output E_RESET; /*DM9000AE上电复位信号输出*/
   output FLASH_A19; /*FLASH第19位地址信号输出*/
   output LED; /*LED信号输出*/
   output PPI_CLK; /*BF533 PPI时钟信号输出*/
```

```
output SAA7113_CE; /* SAA7113片使能信号输出*/

wire XLXN_VCC; /*线网型变量，连接至VCC模块，即高电平*/

AND2 XLXI_1 (.I0(XLXN_VCC), /*二输入与门，实现FLASH_A19= AMS0*/
             .I1(AMS0),
             .O(FLASH_A19));
AND2 XLXI_2 (.I0(AMS1), /*实现AMS00 = AMS0 & AMS1*/
             .I1(AMS0),
             .O(AMS00));
AND2 XLXI_3 (.I0(XLXN_VCC), /*实现E_RESET = SYSRESET*/
             .I1(SYSRESET),
             .O(E_RESET));
AND2 XLXI_4 (.I0(XLXN_VCC), /*实现LED = SYSRESET*/
             .I1(SYSRESET),
             .O(LED));
AND2 XLXI_5 (.I0(XLXN_VCC), /*实现SAA7113_CE = SYSRESET*/
             .I1(SYSRESET),
             .O(SAA7113_CE));
AND2 XLXI_6 (.I0(XLXN_VCC), /*实现PPI_CLK = SAA7113_27M*/
             .I1(SAA7113_27M),
             .O(PPI_CLK));
VCC XLXI_7 (.P(XLXN_VCC)); /*VCC模块，其输出接至XLXN_VCC*/
endmodule
```

至此程序已经完成，但它和实际的硬件还没有任何联系，用户必须通过管脚锁定来确定它们之间的关系。选择集成开发环境左侧“Processes”窗口中“User Constraints”栏中的“Assign Package Pins”选项，调用ISE的PACE工具来可视化的安排引脚在实际硬件中的位置。锁定完毕后保存退出即可重新返回到ISE集成开发环境。

接着可以双击“Processes”窗口中的“Implement Design”，ISE会依次进行综合、布局布线、生成配置文件。如果一切正常的话，“Implement Design”下的每个子项前都会出现一个绿色的勾号，如图5-1所示。

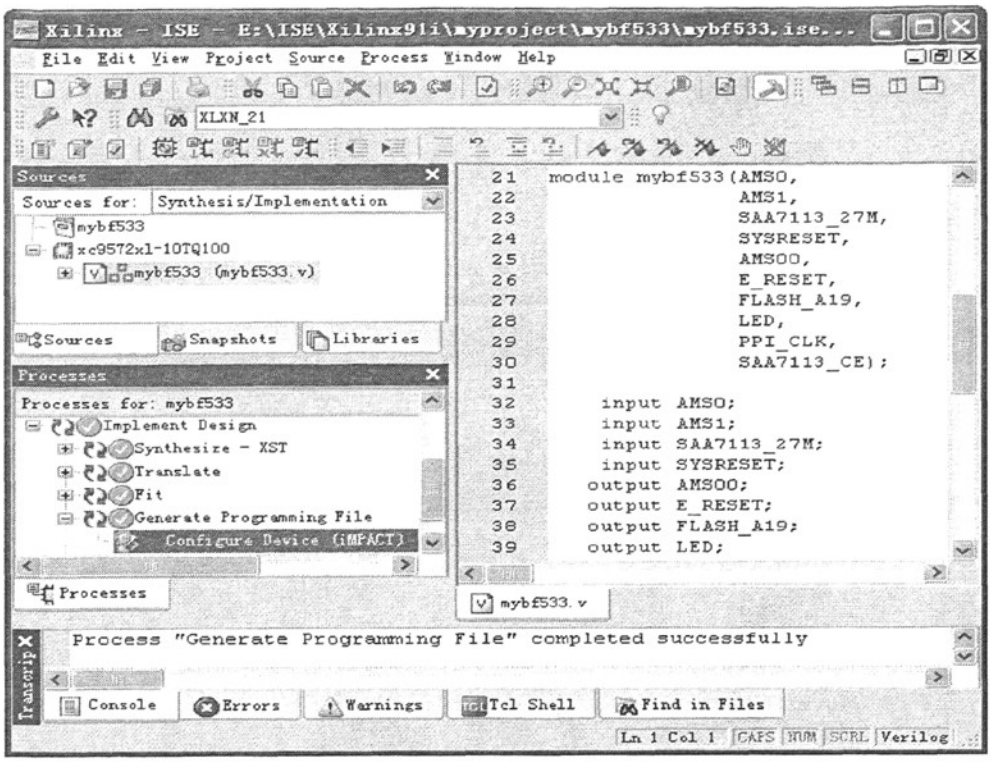


图 5-1 ISE 集成开发环境

生成配置文件后，双击“Configure Device(iMPACT)”选项，iMPACT 工具会自动扫描并连接与 JTAG 下载线相连的 CPLD 设备，连接成功后会出现一个待编程的 CPLD 图标，如图 5-2 所示，右键单击此图标，选择弹出菜单中的“Program”选项，配置文件就被下载至 XC9572XL 中。

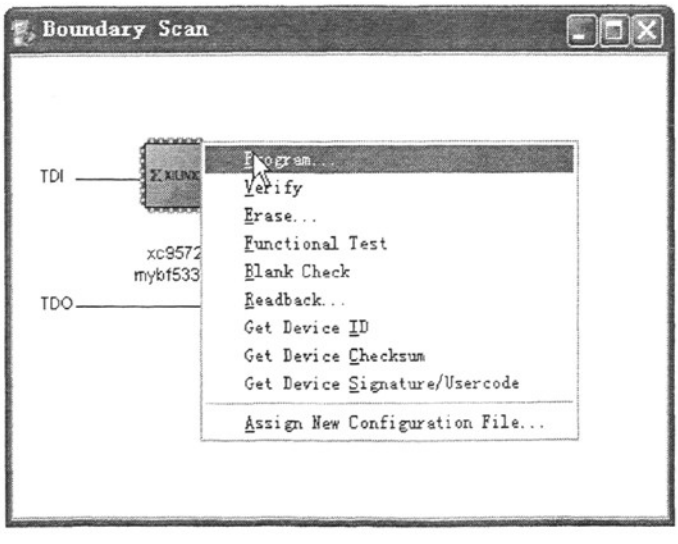


图 5-2 iMPACT 下载工具

5.1.4 以太网控制器DM9000AE的调试

检测以太网控制器是否工作正常，就是检测其能否正确的收发以太网数据包。本文采用的方法是，通过发送一个ARP请求数据包，观察能否接收到一个ARP响应数据包，如果能正确接收到ARP响应，则说明以太网控制器工作正常。

需要实现的功能包括：封装一个ARP请求数据包，初始化DM9000AE芯片，发送ARP请求数据包，等待一个ARP回应数据包，打印收到的数据包信息。下面代码是以太网控制器测试程序的主函数，其中被调用的DM9000AE初始化、数据发送以及接收的函数可以参考4.2.1中关于编写DM9000AE驱动程序的描述。

```
typedef struct
{
 unsigned char  ether_dhost[6];    /*目的以太网地址*/
 unsigned char  ether_shost[6];    /*源以太网地址*/
 unsigned short ether_type;        /*上层协议类型*/
}ether_header;
typedef struct
{
   unsigned short    ar_hrd; /*硬件类型，以太网的值为1*/
   unsigned short    ar_pro; /*协议类型，IP协议为0x0800*/
   unsigned char     ar_hln; /*硬件地址长度，以太网地址长度为6*/
   unsigned char     ar_pln; /*协议地址长度，IP地址长度为4*/
   unsigned short    ar_op; /*ARP操作码，请求为1，回应为2*/
   unsigned char     ar_sha[6]; /*源硬件地址*/
   unsigned char     ar_spa[4]; /*源协议地址*/
   unsigned char     ar_dha[6]; /*目标硬件地址*/
   unsigned char     ar_dpa[4]; /*目标协议地址*/
} ARP_t;
void main(void)
{
   int i = 0;
   unsigned char srceth[6] = {0}; /*源以太网地址*/
   unsigned char dsteth[6]={255,255,255,255,255,255}; /*目的太网
```

```
地址*/
unsigned char srcip[4] = {192,168,0,15}; /*源 IP 地址*/
unsigned char dstip[4] = {192,168,0,2}; /*目的 IP 地址*/
unsigned char pck[100]; /*缓存发送和接收的数据包*/
ether_header* ether;
ARP_t* arp;
for(i = 0, reg = DM9000AE_PAR; i < 6; i++, reg++)
srceth[i] = DM9000AE_ior(reg + i); /*获取 DM9000AE 以太网地址*/
ether = (ether_header*)pck;
arp = (ARP_t*)(pck + sizeof(ether_header));
/*填充以太网头部*/
for (i = 0; i < 6; i++)
ether->ether_dhost[i] = dsteth[i];
for (i = 0; i < 6; i++)
ether->ether_shost[i] = srceth[i];
ether->ether_type = htons(PROT_ARP);
/*填充 ARP 数据*/
arp->ar_hrd = htons(ARP_ETHER);
arp->ar_pro = htons(PROT_IP);
arp->ar_hln = 6;
arp->ar_pln = 4;
arp->ar_op = htons(ARPOP_REQUEST);
for (i = 0; i < 6; i++)
arp->ar_sha[i] = srceth[i];
for (i = 0; i < 4; i++)
arp->ar_spa[i] = srcip[i];
for (i = 0; i < 6; i++)
arp->ar_dha[i] = 0;
for (i = 0; i < 4; i++)
arp->ar_dpa[i] = dstip[i];
eth_init(); /*DM9000AE 初始化*/
int length= sizeof(ether_header)+sizeof(ARP_t); /*数据包长度*/
```

```
    eth_send(pck, length); /*发送 ARP 请求数据包*/
    unsigned char RX_ready = 0; /*标识是否收到数据包*/
    while(!RX_ready){ /*0 表示没有收到数据，1 表示收到数据*/
        RX_ready = eth_rx(pck);
}}
```

在进行实际测试时需要进行以下设置：开发板和电脑连接至同一交换机，将电脑 IP 地址设置为 192.168.0.2。准备好之后，运行程序，输出结果如图 5-3 所示。从输出结果可见，收到的数据包的源以太网地址为 00:0d:61:90:50:31，源 IP 地址为 192.168.0.2，与测试用电脑的以太网地址和 IP 地址相同，因此说明 DM9000AE 能正确发送和接收数据包。

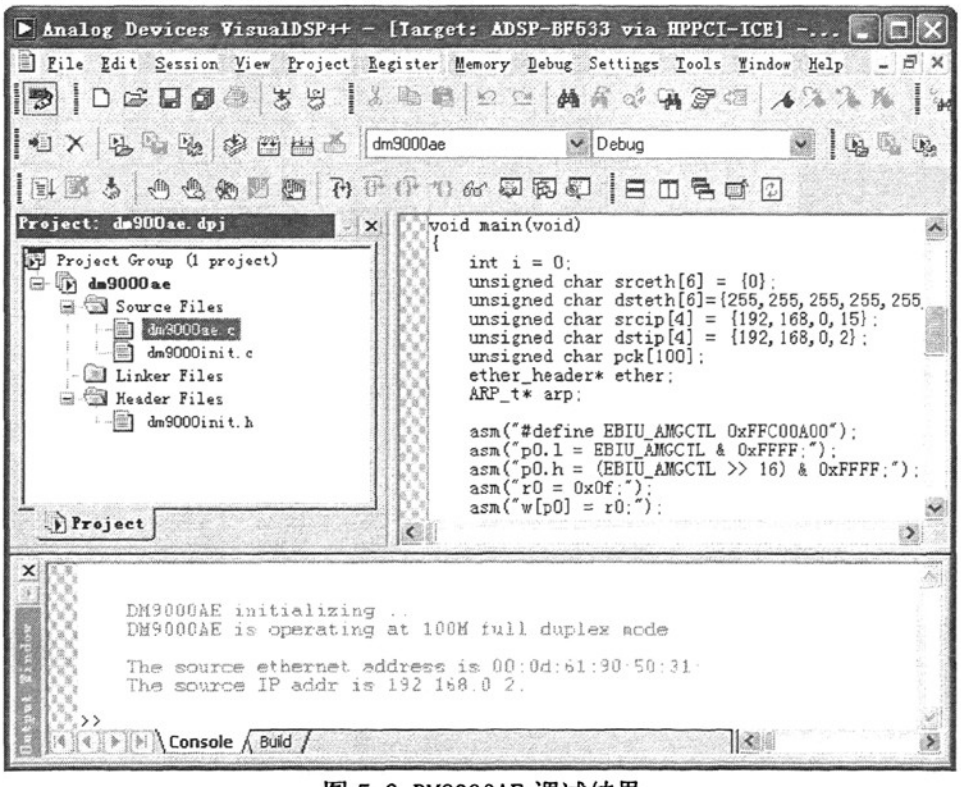


图 5-3 DM9000AE 调试结果

5.1.5 视频解码器 SAA7113H 的调试

为了验证 SAA7113H 是否能正常工作，本文编写了一个测试程序，用来采集一帧视频数据，如果采集成功说明 SAA7113 工作正常。主要工作流程是，设置 SAA7113 相关寄存器，使 SAA7113 开始采集外部模拟摄像头输入的视频；使能

DMA0_PPI 中断，当 DMA 数据传输完毕时产生中断；初始化 DMA0 通道；设置 PPI 相关参数，并使能 PPI，使其开始接收 SAA7113 传输来的视频数据。

1. main()函数

```
extern int flag=0; /*中断发生标志，0 表示未发生，1 表示发生*/
void main()
{
   Init_SAA(); /*设置 SAA7113 相关寄存器*/
   Init_Interrupts(); /*使能 DMA0_PPI 中断*/
   Init_DMA(); /*初始化 DMA0 通道*/
   Init_PPI(); /*设置 PPI 相关参数，PPI 开始接收视频数据*/
   while (!flag){ /*等待 DMA 数据传输完成，产生中断*/
}}
```

2. Init_SAA()函数

```
void Init_SAA()
{
   int t = -1;
   /*通过 I²C 总线将 wbuf 数组中的值依次写入 SAA7113H 寄存器*/
   t = i2c_write(0x25, 0x0, 1, wbuf, 0x63);
   if(t == 0)
   printf("write reg OK!\n");
   else if(t == 1)
   printf("write reg Fail!\n");
}
```

3. Init_Interrupts()函数

```
void Init_Interrupts(void)
{
   /*将 DMA0_PPI 中断映射为中断向量表中 8 号中断*/
   *pSIC_IAR0 = *pSIC_IAR0 & 0xffffffff | 0x00000000;
   *pSIC_IAR1 = *pSIC_IAR1 & 0xffffffff | 0x00000001;
   *pSIC_IAR2 = *pSIC_IAR2 & 0xffffffff | 0x00000000;
```

```
    /*8 号中断向量的中断函数入口地址设置*/
    register_handler(ik_ivg8, DMA0_PPI_ISR);
    *pSIC_IMASK=0x00000100; /*使能 DMA0_PPI 中断*/
}
X_INTERRUPT_HANDLER(DMA0_PPI_ISR)
{
    *pDMA0_IRQ_STATUS = 0x1; /*清零 DMA 中断状态位*/
    printf( "\nThe DMA0 PPI Interrupt has been entered!\n" );
    flag=1; /*中断发生标志设置为 1，即表示中断已发生*/
}
```

4. Init_DMA()函数

```
void Init_DMA(void)
{
    *pDMA0_START_ADDR = 0x0; /*设置 DMA 目的地址为 SDRAM 起始地址*/
    *pDMA0_X_COUNT = Line_Length; /*DMA 传输行长度，对于视频帧来说
    表示每行像素数*/
    *pDMA0_X_MODIFY = 0x2; /*DMA 行内地址增加幅度*/
    *pDMA0_Y_COUNT = Frame_Length; /*DMA 传输行数，对于视频帧来说表
    示每帧视频包括的行数*/
    *pDMA0_Y_MODIFY = 0x2; /*DMA 行间地址增加幅度*/
    *pDMA0_PERIPHERAL_MAP = 0x0; /*DMA0 通道映射至 PPI*/
    /*使能 DMA，设置 2 维 DMA，传输模式为 stop 模式，传输单元 16 位*/
    *pDMA0_CONFIG = DMAEN | DI_EN | WNR | WDSIZE_16| DMA2D | RESTART;
}
```

5. Init_PPI()函数

```
void Init_PPI(void)
{
    *pPPI_FRAME = 576; /*设置视频帧的行数*/
    /*使能 PPI，设置为输入模式，仅采集活动视频，数据宽度为 8 位*/
    *pPPI_CONTROL = PORT_EN | FLD_SEL | PACK_EN | DLEN_8;
}
```

在VisualDSP++中编译并运行上述程序，程序输出“The DMA0 PPI Interrupt has been entered!”提示信息，说明DMA传输已完成，此时视频帧数据已经存储在SDRAM中0x0起始的地址空间内。选择VisualDSP++菜单栏中“View->Debug Windows->Image Viewer”，弹出“Image Viewer”对话框，并进行相应设置，如图5-4所示。设置好之后点击“OK”按钮，就会从SDRAM中读出视频数据，得到如图5-5所示的图像。

从图中可见，该图像分为上下相同两部分，出现这种情况的原因是模拟摄像头输入SAA7113H的视频为PAL制式的CVBS格式，它的每一帧视频都由奇场和偶场两部分组成，并且分别进行传输，所以采集到的视频帧也是按奇场和偶场的先后顺序存储在SDRAM中的。为了能正常的显示所采集的视频图像，需要把分离的奇场和偶场数据进行合并为，使之成为一个完整的视频帧。

奇场是由视频帧中所有奇数行组成的，偶场是由视频帧中所有偶数行组成的。因此，只需将偶场第1行插入奇场第1行之后，偶场第2行插入奇场第2行之后，依此类推，就可以将奇场和偶场合并为视频帧。实现奇场和偶场合并的函数如下所示：

```
void Field_Comb()
{
   unsigned char* odd_filed = ODD_ADDR; /*奇场存储地址*/
   unsigned char* even_filed = EVEN_ADDR; /*偶场存储地址*/
   unsigned char* frame= FRAME_ADDR; /*合成的视频帧存储地址*/
   int i=0,j=0;
   for(i = 0; i < 288; i++){
      for(j = 0; j < 1440; j++){ /*写入奇场中的第i行*/
         *(FRAME_ADDR++) = *(odd_filed++);
      }
      for(j = 0; j < 1440; j++){ /*写入偶场中的第i行*/
         *(FRAME_ADDR++) = *(even_filed++);
}}}
```

在main函数的最后加入该函数的调用语句，重新编译程序并运行，得到完整的视频帧图像，如图5-6所示。

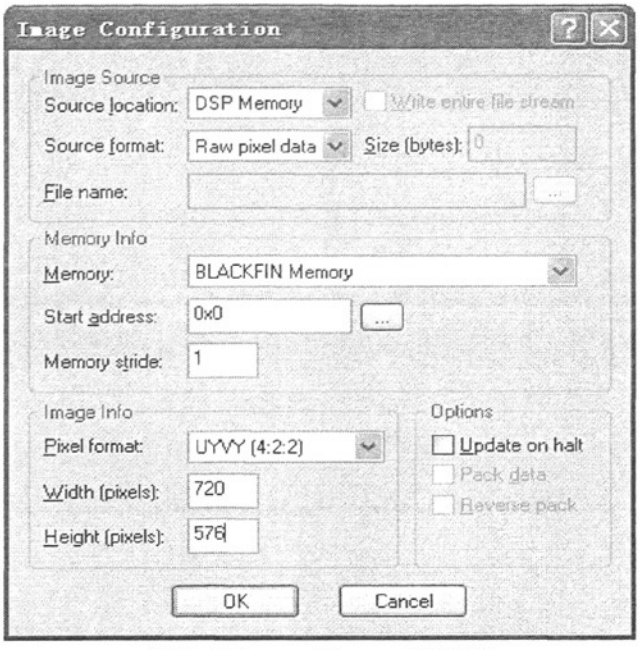


图 5-4 Image Viewer 对话框

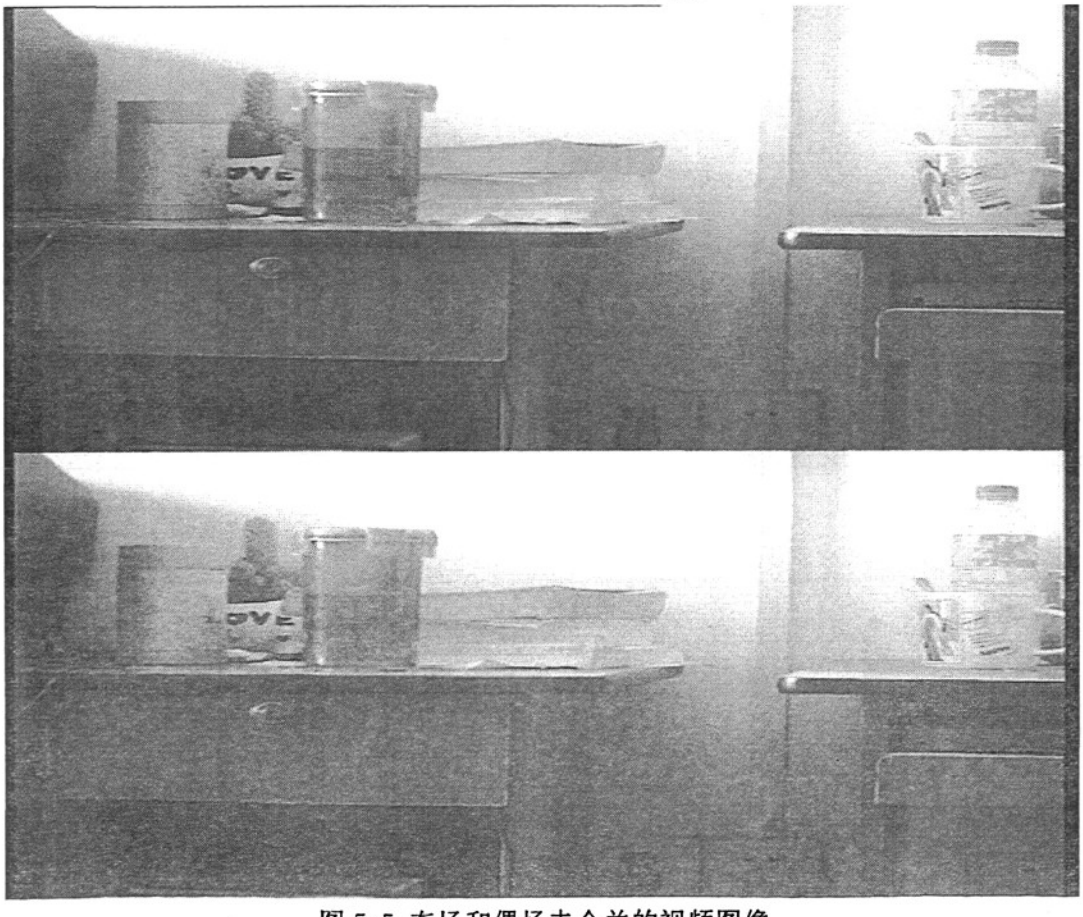


图 5-5 奇场和偶场未合并的视频图像

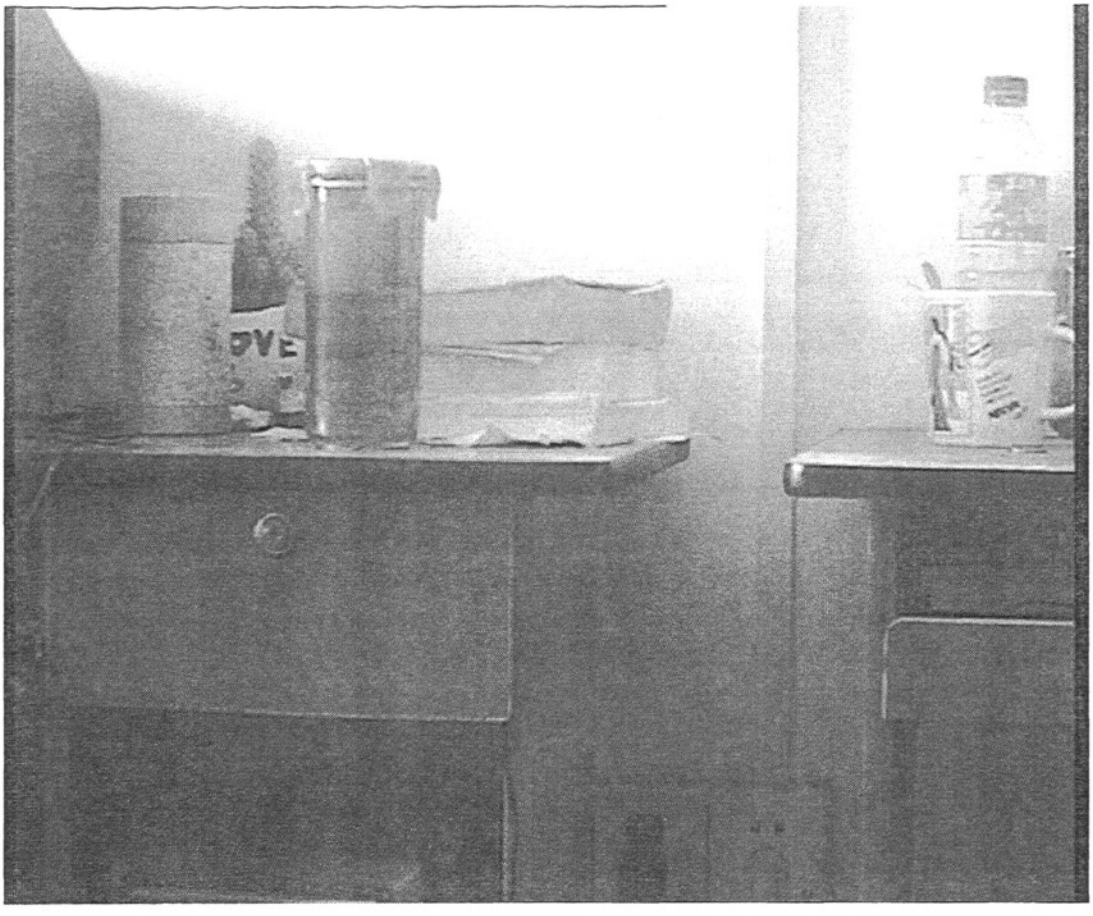


图 5-6 奇场和偶场合并后的视频图像

5.2 U-BOOT 编译与加载

5.2.1 U-BOOT 的编译

要使 U-BOOT 在目标板上正常运行，还必须将源码编译成可执行码。因为 U-BOOT 的编译是在 PC 机上完成的，而编译后的文件是在目标板上运行，所以 U-BOOT 的编译工作必须使用交叉编译器完成。本文在 Fedora 6.0 linux 系统中建立了交叉编译环境，使用的交叉编译工具为 Blackfin gcc 工具集。

1. Blackfin gcc 工具集的安装

从站点 http://blackfin.uclinux.org 下载，然后在 Fedora 6.0 中安装，

运行如下命令：

```
rpm -Uvh blackfin-toolchain-08r1-8.i386.rpm
```

2. 设置 PATH 环境变量

Blackfin gcc 工具集安装完成之后，还不能马上使用它来编译 U-BOOT，还必须设置它的 PATH 环境变量，因为很明显，要使用 gcc 来编译程序，那就必须先告诉系统它的具体安装位置。使用如下命令：

```
export PATH=$PATH:/opt/uClinux/bfin-uclinux/bin
```

但是上面这种方式设置环境变量是比较麻烦的，因为按照这种方法设置后，下次启动控制台又需要重新设置一次环境变量。其实可以通过编辑 root 目录下的一个隐藏文件——.bashrc 文件，来达到一劳永逸的目的。具体方法如下：

（1）显示隐藏文件，选择“编辑->首选项”弹出“文件管理首选项”对话框，选中“显示隐藏和备份文件”复选框；

（2）使用 gedit 编辑.bashrc 文件，添加下面的语句：

```
export PATH=/opt/uClinux/bfin-uclinux/bin:$PATH
```

（3）取消显示隐藏文件。

3. 编译 U-BOOT

在控制台窗口，定位到 U-BOOT 源代码根目录下，输入“make clean”命令，将前一次编译所生成的中间目标文件(格式为.o 的文件)清除掉，如果第一次对 U-BOOT 源文件进行编译可以跳过此步，因为相关目录下没有任何的.o 文件。

接下来输入 make mrproper 命令，主要用途是将前一次编译所生成的最终目标文件(u-boot、u-boot.bin、u-boot.map 等文件)清除掉，第一次对 U-BOOT 源文件进行编译可以不执行此命令。

因为 u-boot 源代码中包括了许多不同的开发板，所以必须事先告诉编译器所要编译的开发板的具体型号，这样编译的时候就会根据输入的开发板型号，确定需要生成的最终目标文件，然后通过依赖关系进一步找到相关的源文件，最后对相应的源文件进行编译和链接生成我们所需的最终目标文件。编译本系统的 U-BOOT 时，输入“make mybf533”命令，此命令是为编译本系统 U-BOOT 而在 Makefile 文件中自定义的命令。

最后，输入“make”命令对源文件进行编译，编译完成之后，生成了 u-boot.bin 二进制文件，使用 VisualDSP++ 5.0 集成开发环境中的 flash programmer 工具通过仿真器将该文件下载到目标板 FLASH 中。

5.2.2 U-BOOT 网络加载μClinux

1. U-BOOT 自启动命令设置

本文为了实现系统上电后自动通过 TFTP 从主机下载并运行μClinux 操作系统，需要设置 U-Boot 的自启动命令。下面的命令是设置环境变量 tftp_boot，它包含的操作是通过 tftp 将文件名为 uImage(μClinux 系统编译后生成的文件，可以将其更改为你所希望的任何名字)的文件从主机下载至目标系统的 0x1000000 地址处，并从 0x1000000 处对嵌入式操作系统解压缩并运行：

```
set tftp_boot 'tftp 0x1000000 linux;bootm 0x1000000'
```

接下来将 tftp_boot 设置为自启动命令：

```
set bootcmd run tftp_boot
```

最后为了使环境变量生效还需要使用 U-Boot 的“save”命令将刚才设置的环境变量保存到 flash 中。

2. TFTP 服务器设置

为了让目标系统能够正确的从主机中下载操作系统，需要对主机的 TFTP 服务器进行设置，首先是主机的 IP 地址必须与 U-Boot 中设置的服务器 IP 地址相同，其次是主机中文件的名字必须与 U-Boot 的环境变量 tftp_boot 中设置的文件名相同。

3. μClinux 网络加载测试

首先在测试用电脑中安装 TFTP 服务器，并将μClinux 映像文件存放在 TFTP 服务器根目录下。接着将电脑和开发板连接至同一交换机。然后通过串口数据线将开发板和电脑相连，以实现信息的交互。

在电脑中打开“超级终端”程序，并对开发板上电。可以从超级终端观察到串口输出信息，首先是 U-BOOT 启动时的一些初始化信息，并停留在自启动倒计时阶段，如果 5 秒内没有任何按键操作，U-BOOT 运行自启动命令，通过 TFTP 下载μClinux 内核映像，如图 5-7 所示，下载完成之后，就立即执行μClinux，最终系统进入μClinux 命令行继续运行，如图 5-8 所示。由图可见，本系统成功实现了 U-BOOT 网络加载μClinux。

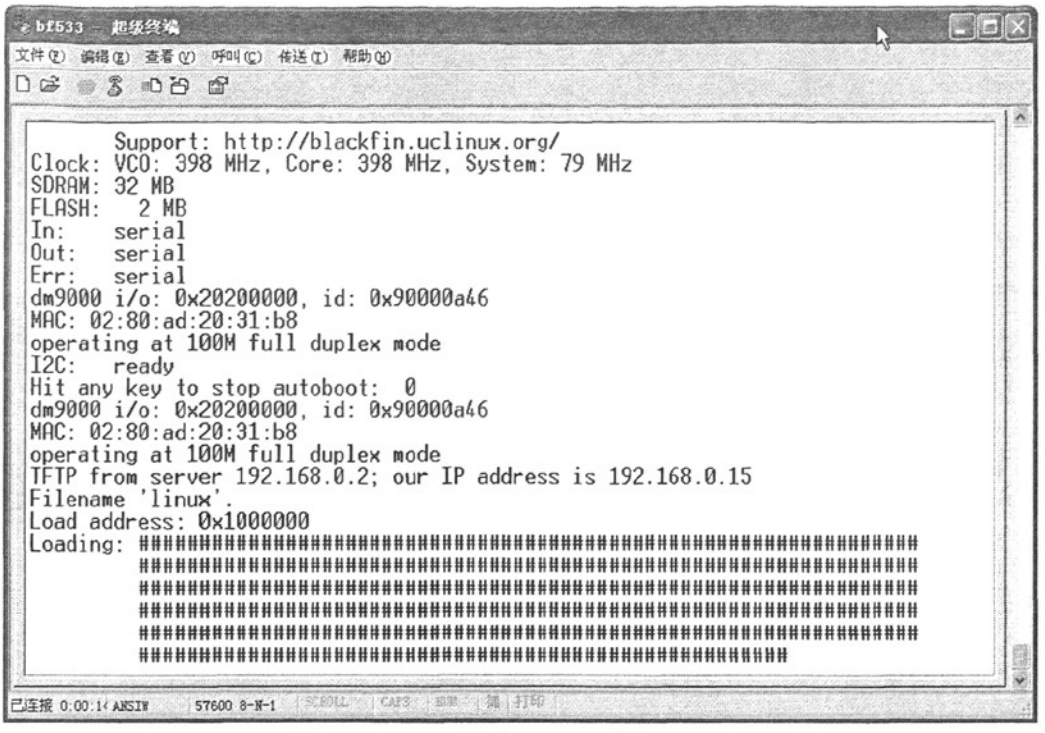


图 5-7 UBOOT 通过 TFTTP 下载 μClinux

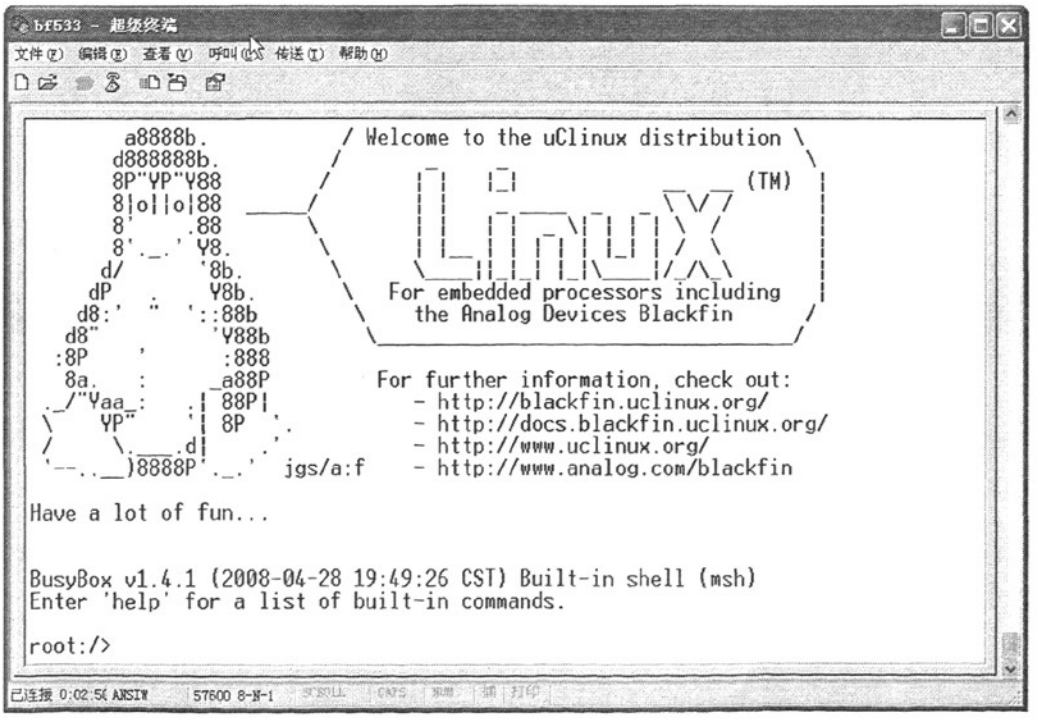


图 5-8 μClinux 命令行界面

结 论

本论文完成了基于BF533处理器的网络监控系统的硬件电路设计、实现和调试，以及系统底层软件U-BOOT的移植。系统经过验证能正常工作，达到了论文的预期目标。论文做的主要工作如下：

（1）通过市场调查和系统需求分析确定了本系统的基于ADI公司BF533处理器的设计方案，根据系统需要实现的功能对各外围模块进行了芯片选型，并选定了U-BOOT+μClinux的系统软件方案。

（2）完成了系统硬件电路各个模块的原理图设计，并在此基础上设计完成了PCB图，制作出了实际硬件电路板。

（3）基于U-BOOT的源代码，分析了其完整的启动流程。针对本系统的硬件环境对U-BOOT进行了移植，包括编写DM9000AE驱动、SST39VF1601驱动，以及对其它相关文件的修改。

（4）通过编写相应的测试程序，对硬件电路各个功能模块进行了测试。

（5）编译U-BOOT，并将其下载至开发板中；对U-BOOT进行相关设置使其能从网络加载μClinux。

由于系统的复杂和时间的限制，本系统只是为进一步的深入应用实现了一个基础平台，将来还有许多需要实现的功能，以及一些对现有设计的改进之处：

（1）移植视频压缩编码算法。为了通过网络传输视频并达到实时播放的要求，需要将采集到的体积大的原始视频信号压缩为体积小的压缩视频信号，因此对视频编码算法的移植是视频监控应用中必须要进行的工作。

（2）添加RTP（实时传送协议）支持。RTP是一个专用于传输音视频数据的传输协议。为了使μClinux系统的应用程序能使用RTP协议来发送数据包，需要将RTP库移植到本系统中。

（3）编写相关应用程序完成视频数据采集、压缩、传输、接收、播放，其中基于BF533的本系统上运行的应用程序实现视频采集、压缩、传输功能，而实现接收、播放功能的应用程序则运行于PC机平台。

（4）由于经验的不足和时间的仓促，本系统硬件电路板的布局不是很紧凑。今后如果再版时，需要对电路进行一些修改，使布局上更为合理。

致 谢

时间飞逝犹如白驹过隙，转眼间我即将度过在西南交大七年的学习生涯。此刻，站在人生的十字路口回看这七年的学习和生活，颇有感触。

首先非常感谢我的导师苟先太老师，正是在苟老师的精心指导和悉心关怀下我才得以完成这篇论文。苟老师学识渊博，治学态度严谨，有着精益求精的敬业精神和缜密的思维方式以及超凡敏锐的洞察力，这些一直都在影响和激励着我。这三年的谆谆教诲，必将使我受益终身。再次向苟老师致以真挚的敬意和感谢！

感谢电气工程学院各位领导和老师对我的关怀和帮助。感谢他们在专业课学习和课题研究过程中给我的大力支持和帮助，他们的渊博学识和科研精神同样使我受益匪浅。

感谢评阅本论文的专家和读者，感谢他们为此付出的辛勤劳动和提出的宝贵意见。

感谢我们实验室的所有同门，通过与他们沟通、交流和学习，解决了许多论文工作中的难题。

最后，我还要感谢我的父母，感谢他们辛勤的劳动和谆谆教诲以及这么多年来在我求学之路上给予我的精神和物质上的支持。

读书的生活洁白而朴素，美好的时光将永驻心中！

参考文献

[1] 张博. 中国视频监控市场进入高速发展期[J]. 电视技术, 2008,(12):49
[2] 颜丙秀, 陈竞, 刘桦. 网络视频监控唱响转型新篇章[J]. 中国电信业, 2008,(07):66-67
[3] 朱冰. 网络视频监控产业迎来春天[J]. 中国新通信, 2007,(18):59-60
[4] 卢秋波. 视频监控技术简介与发展趋势[J]. 电信网技术, 2007,(01):10-12
[5] 张伟. 视频监控技术的发展历程和方向[J]. 中国交通信息产业, 2007,(06):139-142
[6] 赵婧. 基于嵌入式技术的网络视频监控系统[J]. 测控自动化, 2005,(04):26-27
[7] 郭卫华. 模拟视频监控系统之过去、现在和将来[J]. 中国安防, 2008,(Z1):54-57
[8] 张国刚. 视频监控技术演进过程及其在社会信息化过程中的应用[A]. 第十届中国科协年会信息化与社会发展学术讨论会分会场论文集[C], 2008:667-670
[9] 李子奇, 王杉. 中国视频监控市场[J]. 办公自动化, 2008,(17):8-9
[10] 马田, 周其刚. 网络视频监控市场的现状及发展[J]. 信息网络, 2004,(09):20-22
[11] 郝继辉. 网络视频监控技术的发展和展望[J]. 中国科技信息, 2007,(07):97-99
[12] 陆福明. 感受 2007 年网络视频监控发展历程[J]. 中国安防, 2008,(Z1):51-53
[13] 赵建国. 网络视频监控系统的探讨[J]. 电子商务, 2008,(05):74-75
[14] 潘亚南, 钟晓明. 浅谈网络视频监控技术的现状及发展[J]. 内蒙古科技与经济, 2008,(14):66-68
[15] 张鹏. 嵌入式系统方案设计[J]. 科技资讯, 2008,(24):16
[16] 郑旭东, 张培仁, 高修峰, 陈云鹏. 嵌入式网络视频监控系统[J]. 仪表技术与传感器, 2006,(08):24-26
[17] 汤晓冰, 郭健, 沈红星, 陈琳. H.264 便携式实时编解码器的方案探讨[J]. 电视技术, 2007,(09):30-32
[18] 刘永智, 朱江. 基于AL9V576的MEPG1/2/4视频光纤传输系统[J]. 光通信技术,

2007,(06):39-40

[19] 杜学亮，张鑫，负超，金西. 基于FPGA的IP核开发板的设计及测试[J]. 单片机与嵌入式系统应用，2005,(04):25-26

[20] 英特尔公司. 英特尔微信号处理器架构[J]. 世界电子元器件，2003,(06):36-37

[21] David Katz; Russell Rivin. Blackfin 嵌入式媒体处理器的体系结构[J]. 世界电子元器件，2005,(03):37-41

[22] 陈峰. Blackfin系列DSP原理与系统设计[M]. 电子工业出版社，2004

[23] ADSP-BF531/ADSP-BF532/ADSP-BF533 Blackfin® Embedded Processor [DB/OL]. Analog Devices, Inc. 2004

[24] ADSP-BF533 Blackfin Processor Hardware Reference [DB/OL]. Analog Devices, Inc. 2006

[25] HY57V561620C(L)T(P) 4 Banks x 4M x 16Bit Synchronous DRAM [DB/OL]. Hynix Semiconductor Inc. 2004

[26] 16M bit / 32M bit / 64M bit (x16) Multi-Purpose Flash Plus [DB/OL]. Silicon Storage Technology, Inc. 2005

[27] SAA7113H 9-bit video input processor [DB/OL]. NXP Semiconductors, Inc. 2005

[28] ADV7170/ADV7171 Digital PAL/NTSC Video Encoder with 10-Bit SSAF™ and Advanced Power Management [DB/OL]. Analog Devices, Inc. 2004

[29] AD1836A Multichannel 96 kHz Codec [DB/OL]. Analog Devices, Inc. 2003

[30] DM9000A Ethernet Controller with General Processor Interface [DB/OL]. DAVICOM Semiconductor, Inc. 2006

[31] XC9572XL High Performance CPLD [DB/OL]. Xilinx, Inc. 2004

[32] LM117/LM317A/LM317 3-Terminal Adjustable Regulator [DB/OL]. National Semiconductor Corporation. 2002

[33] LT1529-3.3 3A Low Dropout Regulators with Micropower Quiescent Current and Shutdown [DB/OL]. Linear Technology Corporation. 2005

[34] +3V Voltage Monitoring Low-Cost μP Supervisory Circuits [DB/OL]. Maxim Integrated Products. 2006

[35] Robert Kilgore. Hardware Design Checklist for the Blackfin® Processors [DB/OL]. Analog Devices, Inc. 2005

[36] 张大波，吴迪. 嵌入式系统原理、设计与应用[M]. 机械工业出版社，2005

[37] 桑楠. 嵌入式系统原理及应用技术开发[M]. 北京航空航天大学出版社，2001

[38] 周攀. 基于 BF533 网络摄像机的设计与实现研究[D]. 华中科技大学硕士学位论文，2007

[39] 王磊，王耀南，陈斯斯，崔波亮. 基于 BF533 的网络视觉跟踪监控系统[J]. 自动化仪表，2008,(06):17-20

[40] 郑庆宁. 基于 DSP 的嵌入式网络视频监控系统的研究及硬件设计[D]. 浙江大学硕士学位论文，2007

[41] Maikel Kokaly-Bannourah. EE-210: SDRAM Selection Guidelines and Configuration for ADI Processors [DB/OL]. Analog Devices, Inc. 2004

[42] 王莹，李学生，项多云，李爽. 基于ADSP-BF533的音频处理系统设计[J]. 电声技术，2006,(08):32-35

[43] Zhao-hui Li, Dong-meiLi, Qi Zhang, Qiong Wu. Design and implementation of blackfin DSP-based video encoder in network surveillance system [A]. 8th International Conference on Signal Processing [C], 2006:16-20

[44] 韩超，王可人. 基于 DM9000 的嵌入式系统的网络接口设计与实现[J]. 工业控制计算机，2007,(04):17-18

[45] 王一楠，雷杰，孙延均. 基于 Blackfin 嵌入式系统的 U-boot 分析与调试[J]. 电子元器件应用，2007,(08):64-67

[46] Yong-tao Zhou, Xiao-hu Chen, Xu-ping Wang, Chun-jiang Yao. Design of Equipment Remote Monitoring System Based on Embedded Web [A]. 2008 International Conference on Embedded Software and Systems Symposia [C], 2008:73-78

[47] 陈赜，秦贵和，徐华中，王磊. ARM9 嵌入式技术及 Linux 高级实践教程[M]. 北京航空航天大学出版社，2005

[48] 吴川，王斌. 基于 ADSP-BF533 的 Boot Loader 的移植分析[J]. 测控技术，2008,(07):56-58

[49] 刘娅. 基于 ARM 嵌入式系统的 Bootloader 的设计与实现. 现代电子技术，2006,(07):142-144

[50] DM9000A Application Notes V1.20 [DB/OL]. DAVICOM Semiconductor, Inc. 2005

[51] 周洋. 基于嵌入式系统的网络视频监控系统设计与实现[D]. 南京航空航天大学硕士学位论文，2007

[52] 万建新. 浅谈子网掩码、网关与 ARP 协议的作用[J]. 甘肃冶金, 2007,(04):80-82

[53] 姚爱琴, 张东宁, 单永利. 基于 ADSP-BF533 的视频监测系统设计[J]. 中北大学学报(自然科学版), 2007,(1):38-41

[54] 汪小燕, 连晓平, 董燕, 杨大鹏. 基于TFTP协议的嵌入式系统开发方法设计与实现. 华中科技大学学报(自然科学版)[J], 2006,(12):56-58

附　　录

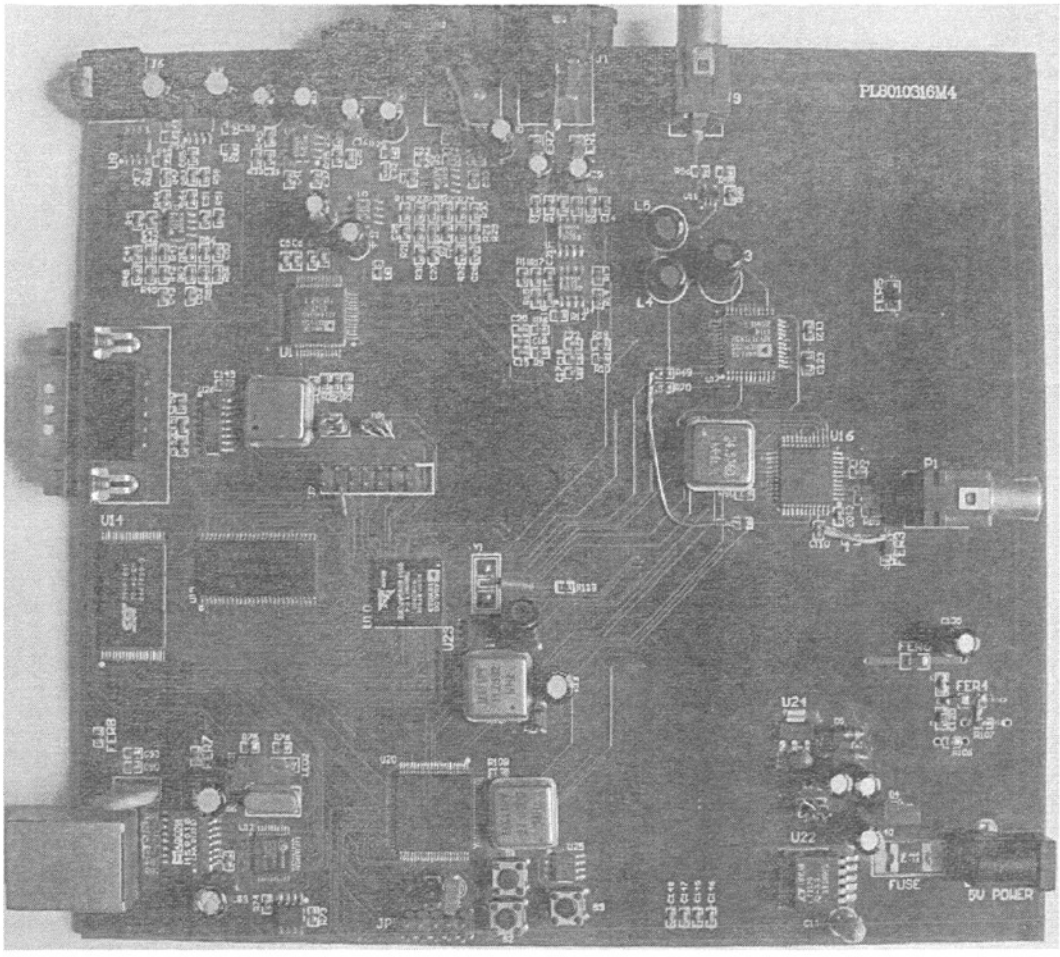


图 1 系统硬件实物图

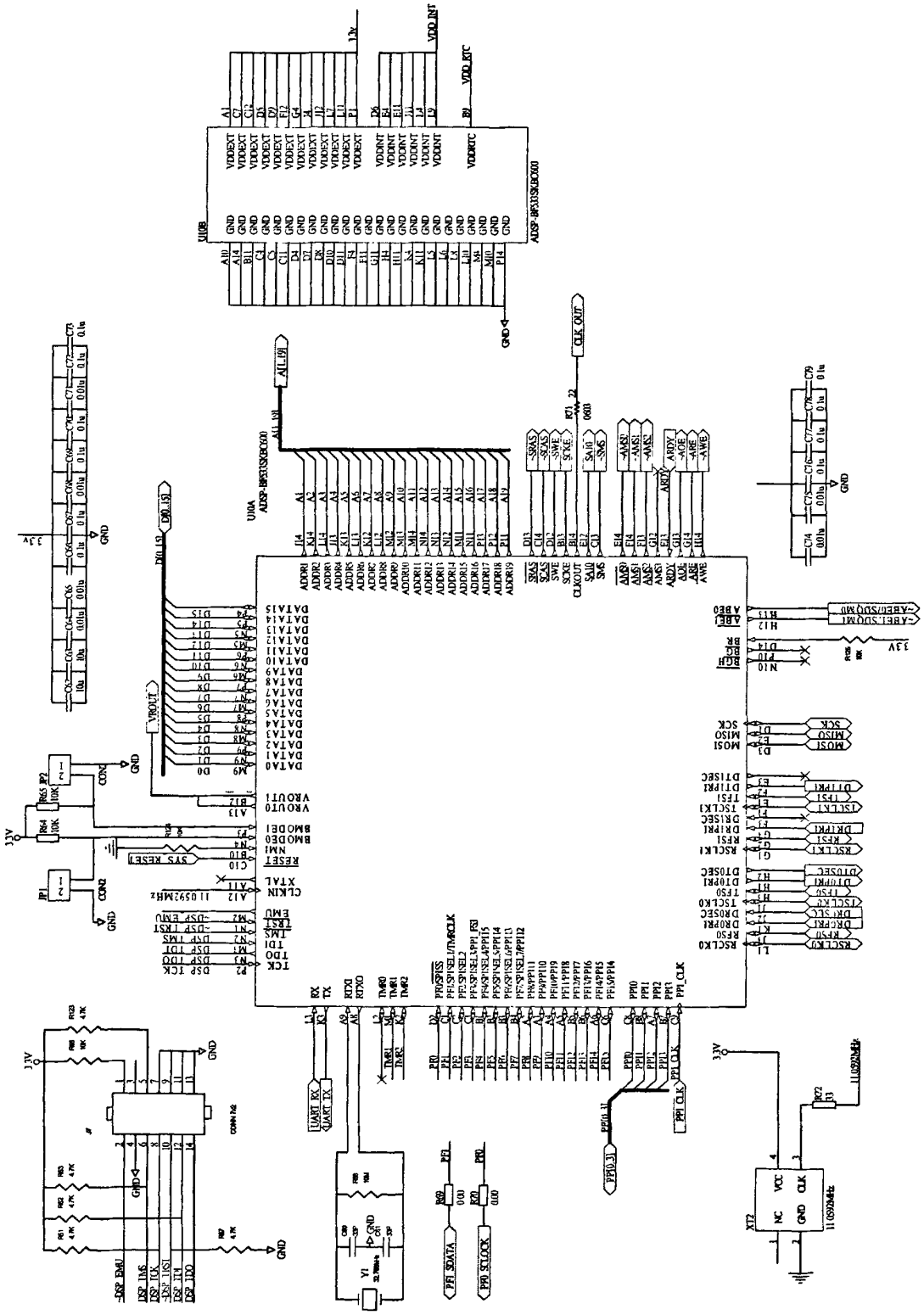


图 2 BF533 处理器模块

攻读硕士学位期间发表的学术论文

[1] 孙延均，关翔．低功耗、高质量视频编码器 ADV7393 及其应用．国外电子元器件，2008,(03)

[2] 孙延均，苟先太，龙刚．嵌入式操作系统的网络启动．电子设计应用(已录用)

[3] 王一楠，雷杰，孙延均．基于 Blackfin 嵌入式系统的 U-boot 分析与调试．电子元器件应用，2007,(08)

基于BF533的网络视频监控系统设计与实现

作者： 孙延均
学位授予单位： 西南交通大学