基于 USB2.0 技术的数字电视接收设备软件设计

摘　要

电视数字化是一个全球的趋势，是继黑白电视到彩色电视后的又一次革命。近年来，随着我国电信网、计算机网和有线电视网三网融合的不断推进，在计算机上提供包括语音、数据、图像等综合多媒体的通信业务具有很好的发展前景并且会带来巨大的商业利润。但是，目前国内数字电视硬件生产厂商大都致力于开发生产数字电视机顶盒产品以及 IPTV 网络电视盒，而针对在个人电脑上观看数字电视的产品却不多；另一方面，尽管不同协议的数字电视传输时的调制编码方式不同，但最终播放时的视频格式都是 MPEG2 格式，而这种格式的视频信号可以直接在个人电脑上播放。因而基于这种状况，急需寻找一种解决方案，能在尽量减少人力物力财力消耗的前提下在个人电脑上接收各种数字电视信号并且实时播放数字电视。数字电视卡、电视棒等便携式数字电视接收设备由此应运而生，它使得个人电脑同样拥有数字电视机一样的功能。

本文的主要工作内容如下：

1.介绍了国内外数字电视发展现状，以及业界领先的相关解决方案。

2.深入研究和理解各类数字电视接收技术的基础上，完成了数字电视接收设备的整体架构设计。概述了数字电视接收设备的基本硬件原理架构，并把 USB 模块及其后端软件设计作为研究重点。

3.完成了 USB 固件程序设计，包括 Cypress 公司固件框架程序简介，并在该框架程序基础上完成了数字电视接收设备 USB 固件程序设计。

4.完成了 USB 驱动程序设计，其中包括对 Cypress 公司的通用驱动程序分析，以及基于微软的 DDK 开发工具开发 USB 驱动程序。

5. 完成上位机播放程序设计，包括对 DirectShow 技术与 BDA 驱动技术进行讲解，并使用这两种技术开发上位机播放软件。

关键词： 数字电视，USB，驱动，BDA，DirectShow

SOFTWARE DESIGN OF DIGITAL TV RECEIVER BASED ON USB2.0 TECHNOLOGY

ABSTRACT

Anologal TV becoming Digital TV is a trend of the world, which is another revolution after black and white television to color TV. In recent years, with the integration of telecommunications networks, cable tv networks and the internet, providing communications business such as voice, data, image on computers will be a good industry and it will bring huge commercial profits. But recently the domestic digital TV hardware manufacturers mainly focus on the development of digital TV set-top box and IPTV network TV box, while there are little digital TV related products for personal computers. On the other hand, though different kinds of digital TV have different kinds of transmission modulation codes, the viedo formats are all MPEG2 when they are broadcasted at the end. Format MPEG2 is a standard viedo format which can be broadcasted directly on personal computers. Thus it is urgent to find a good solution of watching digital TV on personal computers without replace the old equipments and avoiding consuming a lot of manpower and material resources. Digital TV card and Digital TV dongle is the perfect soluction. It allows the personal computer to have the same functions as a digital TV.

The main research contents and results are as follows:

1. This report introduces the status of the digital TV, and the advanced solutions.

2. Responsible for the overall architecture design of Digital TV receiver, including requirements analysis, divided function module and the key technology solutions.

3. Overview the hardware platforms of digital TV receiver, and introduction of USB2.0 technique, including the designing of USB firmware.

4. Overview the method of driver design, Complete the USB driver design based on DDK.

5. Complete the software design of digital TV player on Windows XP system enviroment based on DirectShow and Broadcast Driver Architecture technique.

Key Words: digital tv, usb, driver, bda, directshow

第1章 绪 论

1.1 研究背景及意义

21 世纪是一个高科技的时代，随着通信技术、计算机技术、信息处理技术的不断发展，人类社会已经进入数字与网络的时代。数字电视技术的发展正是人类社会进入数字化时代的最好体现。

数字电视的含义：从电视节目的制作、编辑到存储、发射、传输再到电视信号的接收、处理与最终播放显示等全过程完全采用二进制数字处理技术的全新电视系统[1]。由此可见，数字电视是一个整体的系统的概念，并不是单指数字电视机。数字电视系统传输的图像以及伴音信号都要经过一系列数字处理，如图像压缩处理、信号编码纠错处理和数字调制处理等[2]。经过数字化处理后的数字电视信号源可以通过地面有线电缆、卫星信号、无线电波等形式向外传送，在被接收端的数字电视接收设备接收后经过数字解调和数字音频视频解码等逆处理过程之后被还原成原来的图像和伴音[2]。

目前，数字图像压缩、数字信号编码、数字信号调制以及数字信号检测纠错等技术的不断发展很好地解决了数字电视传输过程中以及其他一系列操作中引入的噪声(误码)，即使出现误码也能通过相应的检错纠错技术把错误的二进制数据位纠正，这使得数字电视的图像及伴音质量几乎与发送端的完全一样，电视台的电视节目源有多清晰，我们播放出来的电视节目就有多清晰。而传统模拟电视就无法像数字电视那样通过二进制纠错解决传输中引入的噪声。传统模拟电视系统在传输过程中图像质量的损伤是累积的，长距离传输后图像的信噪比必然会下降，图像的清晰度与会越来越低；相位偏移的累积使得图像产生色彩失真、镶边及重影等一系列问题；另外模拟电视信号经常出现亮色信号串扰、行蠕动以及半帧闪烁等现象，并且还有稳定性差、可靠度低、调整难度大、集成难度高、自动化控制难度大等缺点[3]。

模拟电视有如此多的不足和缺点，而数字电视技术不仅没有模拟电视的这类缺点，而且还有很多新的优点，数字电视相比模拟电视的优势概括起来主要有以下几点：图像传输质量高，具有数字环绕立体声伴音，频谱资源利用率高，交互性好，设备可靠、维护简单，节省发送功率，覆盖范围广，易于加密，有利于信息安全以及实现条件化接收[3-4]。

可见，数字电视系统取代模拟电视系统是一个必然的趋势。数字电视相关产业必将是

一个很有发展前景的产业。

1.2 国内外数字电视发展现状

美国、英国、日本等西方发达国家数字电视发展的较早。早在2005年底，美国政府就开始了数字电视计划——“2005年数字电视转换和公共安全计划”。在这个计划当中，为了完成模拟电视向数字电视转换这一目标，美国政府推出了数字机顶盒购买优惠项目——“电视机顶盒优惠券项目”；从2007年3月开始，美国禁止销售模拟信号的电视机，并给普通民众发放代购券购促使他们购买数字电视机顶盒；在2009年6月12日，美国正式关闭了模拟电视信号[3,5]。

英国是欧洲数字电视技术的主要发起人和推动者，早在1993年英国政府就进行了欧洲最早的地面数字电视传输技术的实验；从1995年到1997年间，英国政府颁布了很多法律法案来保证国内数字电视技术的发展，1997年开始英国政府正式发放数字电视牌照；同美国类似，英国政府为了全面实现数字电视，向民众推出了各种优惠免费政策，如地面数字电视用户可以免费接收到一些电视节目，并免费获赠数字电视机顶盒。到2006年底，将近80%的英国家庭都使用了数字电视。按照英国政府的计划，在2012年他们将实现全面数字电视化[3,6]。

日本在2001年的时候修订了《电波法》，并计划在2011年之前停止播放模拟电视信号；2003年12月1号是日本的“数字广播日”，当天，在日本的东京、大坂、名古屋这三处地方同时开通了数字电视；按照计划，在2011年日本已经停止模拟电视信号，从而全面进入了数字电视新时代[3,7]。

中国数字电视采用的技术主要有有线数字电视技术、地面数字电视技术、卫星数字电视技术以及IPTV网络电视技术。其中最主要的还是有线数字电视市场，占据了整个有线数字电视市场份额的70%以上。截止2007年底，中国数字电视用户数量达到2700万；2011年10月，中国数字电视用户数量成功突破1亿，同时有线电视的数字化程度也超过50%；中国政府计划在2015年之前全面实现模拟电视向数字电的转换，并努力成为全球最大的数字电视生产研发地[3,8]。

1.3 研究内容与章节安排

目前大多数高新科技都被美国、欧洲、日本控制着，数字电视技术也不例外。当前这3个地区的数字电视标准都已经成为了数字电视的国际标准。由于历史政治等各方面因素

的影响，我国的数字电视技术标准主要采用的是欧洲的数字电视标准——DVB(Digital Video Broadcasting)[9]标准。有线电视标准为 DVB-C 标准，卫星电视标准为 DVB-S 标准，而地面数字标准在我国争议很大，最终我国采用了自主研发的 DTMB(Digital Television Terrestrial Multimedia Broadcasting)[10]作为国内地面数字电视标准。各种数字电视技术标准的最大不同之处是采用的调制方式和传输的信道不同：利用高频载波的 DVB-S 使用 QPSK(Quadrature Phase Shift Keying)调制方式[11]，利用低频载波的 DVB-C 使用 64-QAM(Quadrature Amplitude Modulation)调制方式[12]，而利用 VHF 及 UHF 载波的 DTMB 使用 TDS-OFDM（时域正交频分复用）调制方式[3,13]。但无论采用何种数字电视标准，用到的视频传输流格式都是 MPEG2-TS[14]格式，区别只是在于他们传输过程中使用的信号调制方式不同，因为不同的应用场合需要的频率带宽等要求也不尽相同。目前以上各种数字电视技术最终解调出来的音视频节目数据流都是 MPEG2-TS 格式的数据流，因此从解调为 MPEG2-TS 数据流之后的后端接收系统设计基本上是一样的，这样我们就可以设计出一套通用的适合于以上各种数字电视技术的后端接收系统。

本论文通过对各种数字电视技术深入分析理解的基础上，设计了一套较为成熟的数字电视接收设备的后端软件解决方案，只需少量修改就能实现对各种不同数字电视标准都能适用的软件接收播放系统。

论文总共分为六个章节，各个章节的内容安排如下：

第一章概述了什么是数字电视以及数字电视相比模拟电视的优点。通过收集大量资料和文献，掌握了国内外相关领域的研究现状，同时结合国内实际情况，分析了对数字电视后端软件系统设计的可行性与必要性。

第二章对国内各种数字电视技术进行比较与分析，确定数字电视接收系统整体解决方案，通过系统需求分析把系统划分为几个主要模块：USB 固件程序模块，USB 驱动程序模块，数字电视驱动接口模块，上位机播放模块等，并叙述了数字电视接收系统设计过程中的关键技术。

第三章对 USB 技术进行了简单的介绍，详细介绍了 cypress 公司生产的 USB 芯片固件程序的设计。

第四章对 USB 驱动程序设计方法进行了简单介绍，对 cypress 公司的通用 USB 驱动程序进行了简要分析，详细介绍了基于 DDK 技术开发 USB 驱动程序的整个流程。

第五章简单介绍了微软的 DirectShow 播放技术与 BDA 驱动技术，详细介绍了如何基于 DirectShow 技术与 BDA 驱动技术开发电视播放软件。

第六章对全文进行总结，提出一些设计过程中的不足。

第 2 章　数字电视接收设备的系统设计

目前，“云技术”[15]非常火热，其实数字电视技术从本质上来说也属于“云技术”的范畴。你在电脑上播放下载下来的一部电影在本地观看或者直接在线观看其实质就是在观看数字电视，但是由于目前网络速度不够快、硬盘空间不够大等等限制你不可能把所有的电视电影节目都下载到电脑上观看。数字电视系统实际上就是把远端主机上的电视电影节目通过有线或者无线信号传输到你本地，让你在家里可以观看到远端主机上所拥有的各种电视、电影节目。数字电视系统包括传输端、接收端以及传输信道，如图 1-1 所示。

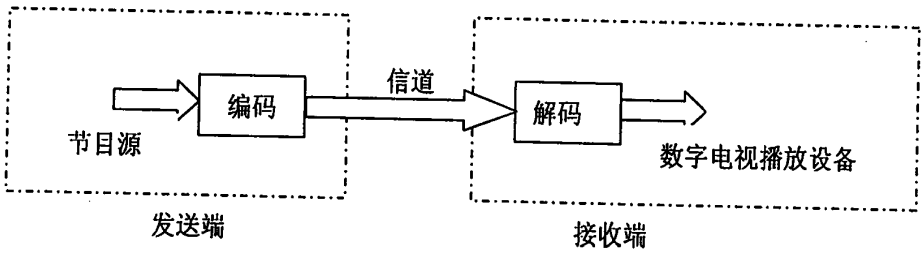

图 1-1　数字电视系统组成方框图

数字电视接收设备属于接收端，数字电视接收系统设计包括系统硬件的设计和系统软件的设计。根据播放环境的不同，系统的硬件设计也不同。如果选择在以前的彩色电视机上播放数字电视节目就需要在硬件设计上多出 MPEG2 解码功能的设计，最后输出的接口需要用到 AV 电视连接接口以便电视信号直接输出给彩色电视机，这种设计就是市面上的数字电视机顶盒[16]，如图 1-2 所示就是该设计的硬件框架图。

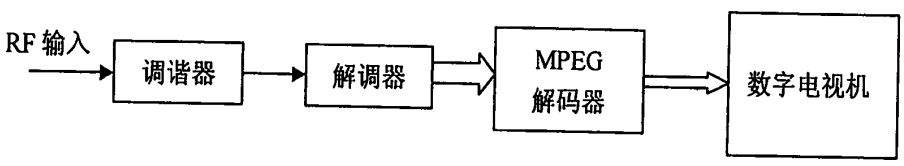

图 1-2　数字电视机顶盒组成示意图

如果选择在个人电脑上播放数字电视节目就不需要在硬件上设计 MPEG2 解码功能模块，只需要将经过解调输出的 MPEG2-TS 流数据输出给个人电脑，然后由个人电脑软

件实现 MPEG2 解码并播放。当然通常的电脑主机都不带 AV 电视插孔，MPEG2-TS 流数据无法通过 AV 电视线输送到电脑主机上，需要通过例如 USB 或者 PCI 等接口传输到电脑主机上。这种设计就是市面上的数字电视卡/电视棒[17]，如图 1-3 所示就是基于这种设计的硬件框架图。

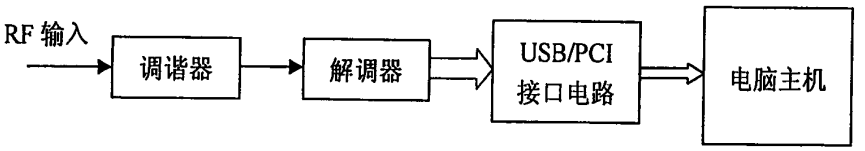

图 1-3　数字电视棒/电视卡组成示意图

2.1　系统需求分析

数字电视机顶盒与数字电视棒各有其特点，它们其实都是在尽量减少硬件成本的基础上，在原有设备上提供新的服务——数字电视播放功能。目前国内大多数硬件生产厂商都在生产数字电视机顶盒产品，如长虹[18]、康佳[19]、同洲[20]、创维[21]等。而开发生产便携式的数字电视棒产品的厂家还不是很多，即使有大都也是模拟电视棒或者网络电视棒；同时随着个人便携式电脑的不断发展，目前手提电脑几乎普及到人手一台，这也给数字电视棒的应用带来了巨大的市场。因此研究像数字电视接收棒这类便携式的数字电视接收设备是一个比较有意义并且有前景的课题。

如前面所述，一个完整的数字电视接收设备产品包括了硬件设计与软件设计。由于实验设备仪器等条件限制，本课题并不制作完整的硬件设备实物，而是在深入研究了相关产品的硬件设计之后主要针对从 USB 接口开始的后端接收系统研究与设计。基于 USB 接口的数字电视接收设备的设计中最核心的部分其实就在于 USB 相关的软件设计，因为 USB 芯片在该设备中担当了“首脑”的作用：对前端，USB 芯片要控制高频头进行输入射频信号的频率选择以及控制解调芯片进行数字信号的解调；对后端，USB 芯片要与电脑主机通信把数字音视频流传输给电脑主机，同时要配合电脑主机的驱动程序以及上位机数字电视播放程序实现实时电视节目的播放。

硬件是基础、是框架，软件是核心、是灵魂。基于 USB2.0 技术的数字电视接收设备软件设计就是图 1-3 所示的从 USB 接口出来到最终在个人电脑上播放数字电视节目这一过程中所涉及到的全部软件程序设计，其中最多的都是与 USB 相关的设计，这也是整个数字电视产品开发中最关键的部分。软件模块大致可以划分为如下几个部分：

◆ USB 固件程序设计，初始化各个硬件寄存器，实现硬件与驱动程序交互。

◆ USB 驱动程序设计，实现 USB 设备与电脑主机的相互通信。

◆ BDA 驱动程序设计，从 USB 设备中读取各种格式视频、音频数据流传输给后端播放设备。

◆ DirectShow 播放程序设计，从 BDA 驱动程序获得数据流实时播放数字电视。

2.2 硬件结构设计

目前国内数字电视市场规模主要以 DVB-C 有线电视为主，约占据了 70%以上的市场份额；其次是 DVB-S 卫星电视，约占据了 15%左右市场份额；地面数字电视以及 IPTV 最少，各占据了大约 5%左右市场份额[22]。由于 DVB-C 与 DVB-S 技术在国内应用最多，发展最为成熟，这里以这 2 种技术的数字电视接收设备硬件设计为例，如图 1-4 和 1-5 所示分别为 DVB-C 与 DVB-S 的一种 USB 接口接收器的硬件设计框图。

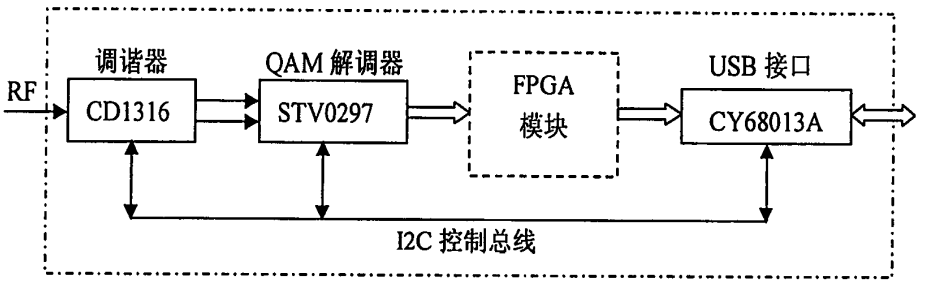

图 1-4 USB 接口的 DVB-C 数字电视接收器硬件设计框图

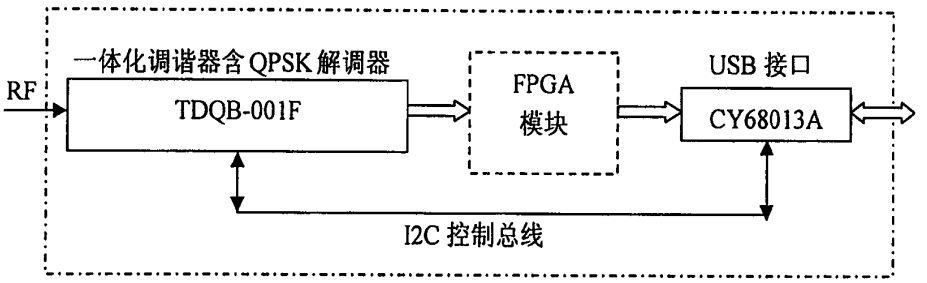

图 1-5 DVB-S 数字电视接收器硬件设计

以图 1-4 所示 DVB-C 硬件工作原理为例，首先载有电视信号的频率范围为 48MHZ 到 860MHZ 的 RF 射频信号进入调谐器 CD1316，调谐器对接收到的 RF 射频信号进行低

噪声放大、滤波、变频等一系列操作后得到 IF 中频信号；IF 中频信号经过调谐器的移相作用后被分成两路正交的中频信号 I 和 Q；I、Q 两路中频信号随后经由 QAM 解调器 STV0297 中的 A/D 模拟到数字信号转换器转换之后变成数字信号，再经过 QAM 解调、Viterbi 解码、Reed-Solomon 解码、去交织等一系列操作之后得到 8 位并行的 MPEG2-TS 格式的音频视频混合数据码流；最后通过 USB 接口芯片把并行数据传输给电脑主机，在电脑主机上进行解复用、MPEG2 音视频解码、渲染播放音视频节目[23]。DVB-S 数字电视接收设备模块与 DVB-C 的主要差别就是解调器部分。DVB-C 系统用到的调制模式是 QAM 调制，一般是 64QAM 调制技术，这种技术比较适合有线信道传输；而 DVB-S 用到的是 QPSK 调制技术，该技术适合于卫星信号传输。因此，除调制解调部分 DVB-C 与 DVB-S 有区别之外，其他各部分模块基本相同。

2.3 关键技术简介

在整个 USB 设备工作工程中，其实最核心的部分都是围绕 USB 芯片在工作的，不管是调谐还是解调又或者是上位机播放软件的一系列操作最终都是由 USB 芯片进行协调完成的。因此本系统的设计其实就是围绕 USB 进行设计的，其中涉及到的关键软件部分也都是为了配合 USB 芯片工作而进行的，这些关键技术包括：

1、USB 主控芯片的选择以及固件程序的设计

USB 芯片质量的好坏直接影响了 MPEG2-TS 数据流的传输质量，固件程序的好坏也将对接收设备的运行稳定性以及工作效率产生很大影响。本文通过对比不同的设计模型，分析其优缺点，最后选择了一种应用最广泛同时也有着很高稳定性的设计模型。

2、USB 驱动程序的设计

驱动程序是外部硬件设备与电脑主机通信的桥梁与纽带，不完善的驱动程序会导致硬件与电脑主机之间的数据传输速率很慢以及稳定性差，甚至根本无法传输数据等问题。

3、BDA 驱动程序的设计

USB 驱动程序设计好之后，主机可以读取 USB 寄存器中的数据，然后根据获得的数据需要判断数据的类型、找到数据的同步头、完成 TS 流解复用并把同步的数据放到缓冲区等操作。这些操作就是由 BDA 驱动程序来完成的。

4、通过 DirectShow 技术编写软件播放数字电视节目

BDA 驱动已经把 TS 数据放在了缓冲区中，接下来需要做的就是从缓冲区中把数据分成音频和视频数据再分别进行音频和视频数据各自的解码，最后把经过解码的音频数据

送到声卡播放出声音，把经过解码的视频信号送到显卡显示出来。

第 3 章　USB 技术的应用与开发

3.1　USB 技术简介

USB 全称为 Universal Serial Bus，即通用串行总线，是计算机上的一种较新的接口技术[24]。由于 USB 接口传输速率高、体积小、可供电、支持热插拔等特点，使电脑主机和外部硬件设备之间的连接和使用都变得很方便，因此 USB 技术也是目前很热门的技术[24]。USB 从 1.0 版本一直发展到最新的 3.0 版本，最高传输带宽不断增大，从最初的 1.5Mbit/s 发展到 5Gbit/s。目前大多数的便携式数码产品都使用了 USB 接口技术。再过不了多久，等到 USB3.0 技术成熟之后，USB 接口技术将很有可能取代其他所有接口技术，成为便携式式数码产品的首选接口技术。

3.1.1　USB 应用系统组成

USB 到底有什么用呢，其实它最主要的作用只有一个——传输数据。一个典型的 USB 接口系统包括基于 USB 接口的设备、USB 主机以及它们之间的连接线路。常见的设备如 U 盘/移动硬盘、MP3/MP4、USB 鼠标/键盘、USB 采集卡、USB 打印机等等都属于 USB 设备；而 USB 主机则主要指拥有 USB 插孔的主机设备，常见的如电脑主机、嵌入式系统等。图 3-1 为典型的 USB 应用系统[25]。

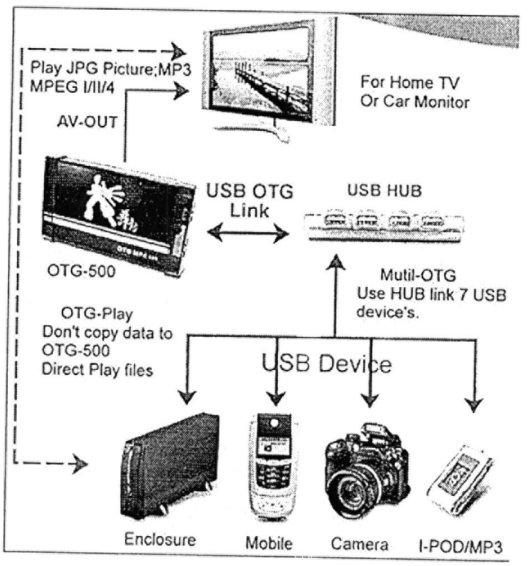

图 3-1　典型 USB 应用系统图

3.1.2 USB 层次结构

每一个 USB 系统都只有一个主机，该主机包括以下几个层次[26,27]：

(1)USB 总线接口层

USB 的总线接口层由主机的主控制器实现，主要处理协议层和电气层的相互连接与通信。在 USB 应用系统中总线接口层中，USB 控制器可以自动进行 NRZI 编码或解码，从而实现数据传输时序。

(2)USB 系统层

USB 系统层的主要责任是管理 USB 设备和电脑主机之间的数据互传，它和主机控制器之间的接口主要靠主机控制器上面的硬件定义。在这一层中 USB 主机可以获得该 USB 设备的能力，这使客户访问 USB 成为可能。

(3)USB 客户软件层

USB 客户软件层处于应用系统结构层次的最上端，它的主要作用是处理具体的 USB 设备驱动器。

3.1.3 USB 数据传输流程

USB 接口的外设与电脑主机进行数据通信时，从上到下需要经过以下几个部分：主机软件、USB 总线驱动程序、USB 主控制器驱动程序以及 USB 功能设备。

这里以电脑主机向 USB 外设发送数据为例来介绍整个 USB 数据传输的流程[28]：

第一步：电脑主机把需要发送的数据首先存储在一个发送数据缓冲区当中，随后向 USB 总线驱动程序发送数据传输的命令请求——IRP 请求，即输入输出请求包。

第二步：USB 总线驱动程序对电脑主机发送的 IRP 请求包做出响应，并将该数据包转换成 USB 协议中标准事务处理格式的数据包形式，随后将转换后的数据包往下传递给 USB 主控制器驱动程序。

第三步：为了保证数据格式符合 USB 传输协议，USB 的主控制器驱动程序把收到的每个事务处理转换成一个一个以帧为单位的事务处理队列，同时也保证了传输速率不会超过 USB 的总带宽。

第四步：USB 主控制器不停地读取事务处理队列，并将它们以信息包的格式发送到 USB 总线上。这里总共可以有四种传输模式可以选择，同时还可以选择低速(1.5Mbit/s)、全速(12Mbit/s)和高速(480Mbit/s)这三种传输速率中的一种进行传输。关于四种传输模式我们在 3.1.5 节中介绍。

第五步：USB 功能设备接收数据包，USB 的智能串行接口引擎自动将他们解码并把解码后的数据存入指定的端点缓冲区地址中。

至此，电脑主机上的传送的数据就被保存到 USB 芯片的端点缓冲区中了。而如果电脑主机向 USB 外设读取数据也同样需要以上几个步骤，但流程和上面刚好相反。

3.1.4 USB 事物处理

在 USB 协议中，USB 的数据传输全部都是由一个一个的数据包组成的。一个完整的事务处理是由不同的信息数据包组合起来的。USB 事务处理是是 USB 功能设备与电脑主机之间进行通信交流的基础。正常情况下，一个完整的 USB 事务处理由三个阶段组成：令牌阶段，数据阶段和握手阶段，每个阶段的作用如下[28-30]：

令牌阶段：由同步字段、令牌包和包结尾字段 3 部分组成，定义了一次数据传输的类型，同时也代表了一次 USB 事务处理的开始。这一阶段是所有 USB 事务处理必有的阶段。

数据阶段：由同步字段、数据包和包结尾字段 3 部分组成，数据传输中需要传输的具体数据就包含在这里。

握手阶段：由同步字段、数据包和包结尾字段 3 部分组成，指示了本次数据传输是否成功，由数据接收端发送报告给数据发送端。

在 USB 外设与电脑主机之间进行数据传输时，包含的事务处理阶段通常是不相同的，但其中必须有令牌阶段。USB 协议中总共规定了如下七种格式的令牌包以处理不同的事务[28-30]：

IN 事务处理：用来实现 USB 外设向电脑主机传输数据，即对电脑主机来说是从外部 USB 设备读入数据。

OUT 事务处理：和 IN 事务处理相反，实现电脑主机向 USB 外设发送数据。

SETUP 事务处理：一种特殊的 USB 事务处理，在电脑主机配置 USB 外设时使用，而且只会在 USB 控制传输阶段使用。

PING 事务处理：主要用在高速率数据传输过程中。

SOF 事务处理：代表了一个 USB 数据帧或小帧的开始。

SPLIT 事务处理：在 USB 高速传输过程中插入低速或者全速传输时使用，提高了 USB 总线的效率。

PRE 事务处理：电脑主机向 USB 集线器发送的令牌包，不需要数据包和握手包。

3.1.5 USB 数据传输模式

数据传输是 USB 最主要的功能，USB 设备与主机的交互就是通过各种数据流进行的。USB 是在主控制器控制下通过管道（Pipe）技术传输设备与主机之间的二进制比特数据流的。USB 支持四种基本的数据传输模式：控制传输，等时传输，中断传输以及数据块传输，每种传输模式的特性与适用场合都是不同的[28-30]：

控制传输方式：支持外部设备与主机之间的控制，为外部设备与主机之间提供一个控制通道。该方式传输速率不快，通常用于实时性不高、数据量不大的场合，如用户自定义的设备请求数据，USB 设备配置信息数据的读取等。

等时传输也叫同步传输方式：支持有周期性，有限时延和带宽且数据传输速率恒定的外部设备与主机之间的数据传输。由于这种传输方式没有差错校验能力，从而无法保证数据的正确传输，因此，通常用于数据量需求比较大的视频和音频数据传输等场合。

中断传输方式：支持无周期性，传输数据量小但要求实时性高，响应速度快的场合。通常用与人机交互的应用场合如 USB 鼠标/键盘等。

数据块传输方式：支持数据量大但对传输时间无严格要求的场合，如 U 盘/移动硬盘，打印机，扫描仪，数码相机等。

3.2 USB 设备开发基础

3.2.1 USB 设备开发流程

任何项目的开发都需要一个开发流程，一个好的开发流程可以节省很多人力物力财力等方面的成本。如图 3-2 所示就是 USB 功能设备开发的常用开发流程图。

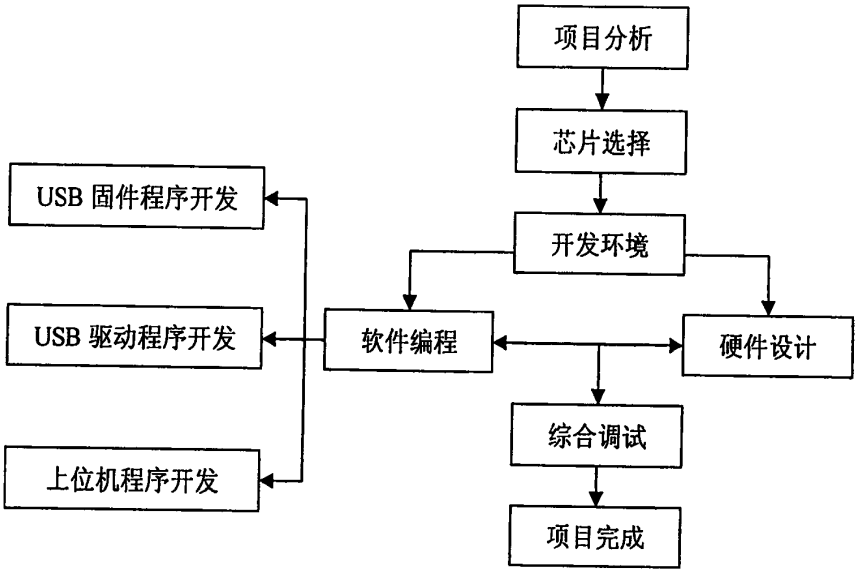

图 3-2 USB 设备的开发流程

各个流程的具体工作如下[31]:

项目分析：主要分析 USB 设备的功能需求，包括设备需要的实现的所有功能，以及每种功能采用的工作方式、设备数据传输类型、传输带宽、功耗等各种软硬件条件。

USB 芯片选择：在全面了解设备各种需求之后就可以进行 USB 硬件芯片选择，如考虑 USB 传输带宽是否足够，该芯片资料是否全面，开发周期长短，性价比如何。

USB 硬件设计：买到芯片后就可以在制作好的硬件电路板上把芯片焊接上去，当所有硬件设备都焊接好并且各个部件连通性检查完好之后便可以进行软件编程设计。

USB 软件设计：USB 软件设计主要包括 USB 固件程序开发、主机端驱动程序开发和上位机用户界面程序的开发。

设备综合调试：软件和硬件不是彻底分开的，它们是紧密联系的，在单独设计好之后需要进行软硬件的联调，在调试过程中可能发现硬件以及软件中的各种问题。最后通过综合调试之后 USB 设备就完成了。

3.2.2 EZ-USB FX2LP 芯片简介

当前 USB3.0 技术尚未成熟，协议规范也未完全统一，此外，目前典型的数字电视 5 到 6 路 MPEG2-TS 节目复合数据流传输速率一般在 30Mbit/s 左右，对于目前数字电视带宽而言 USB2.0 的 480Mbit/s 的最高传输带宽已经足够。因此，从产品开发难度以及开发成本等方面考虑 USB2.0 技术是最好的选择。在众多 USB2.0 芯片生产商中用得最为普遍的是 CYPRESS 公司的 USB2.0 芯片以及 Philips 公司的 USB2.0 芯片。Cypress 公司的

EZ-USB FX2 系列芯片更是世界上首款集成了 USB2.0 协议的微处理器。而在数字电视相关领域中使用较多的 USB 芯片是 Cypress 公司的 EZ-USB FX2LP 低功耗系列的 USB2.0 芯片。该系列芯片的内部结构如图 3-3 所示[32]。

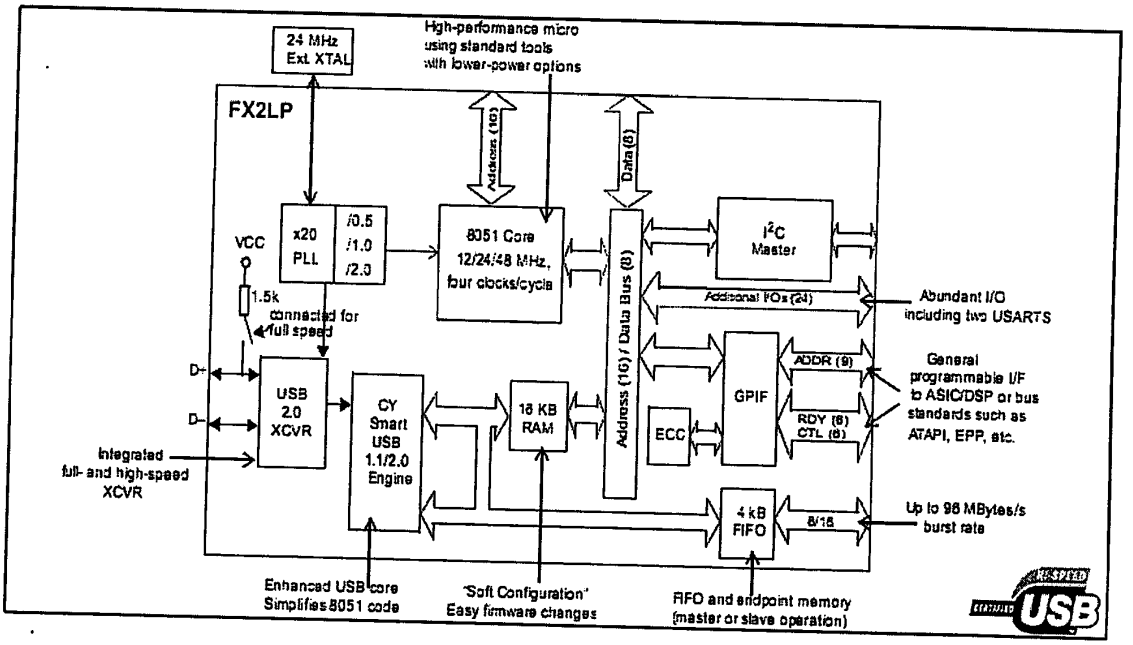

图 3-3　EZ-USB FX2LP 芯片的内部结构

从图中我们可以看出 EZ-USB FX2LP 系列芯片内部主要包括 USB2.0 收发器（XCVR）、智能串行接口引擎（Smart Engine）、16KB 大小的随机存储器（RAM）、16 位地址总线与 8 位数据总线的复用线、4KB 大小的 FIFO 存储器、8 位或 16 位的 I/O 接口、通用可编程接口（GPIF）、I2C 总线控制器（I2C Master）、增强型 8051 内核（8051 core）等。

EZ-USB FX2LP 特有的结构使其智能串行接口引擎（Smart Engine）能够完成串行数据的解码、差错控制、比特填充等功能。智能串行接口引擎（Smart Engine）代替了 8051 内核完成了大部分功能，从而使增强型 8051 内核（8051 core）的负担大大减轻了，这也降低了我们对 USB 本身固件程序的开发难度。

CY7C68013A 是 Cypress 公司推出的低功耗版本的 EZ-USB FX2LP 系列芯片中最典型应用最广泛的一款。根据内部资源数目以及引脚数量多少的不同，CY7C68013A 这款 USB 芯片有 3 种不同的型号，它们分别是 56 管脚的 CY7C68013A-56Pin 型号、100 管脚的 CY7C68013A-100Pin 型号以及 128 管脚的 CY7C68013A-128Pin 型号。CY7C68013A-128Pin 型号芯片是完整版的芯片，拥有全部的 I/O 引脚资源并且每个 I/O 端口都有额外的扩展功

能，而 CY7C68013A-56Pin 和 CY7C68013A-100Pin 型号芯片是 128 管脚芯片的简化版本，特别是 CY7C68013A-56Pin 型号芯片只提供了 PA、PB 以及 PD 这 3 个 8 位并行的 I/O 端口，完整版的还有 PD 和 PE 总共 5 个并行 I/O 端口。从稳定性、功耗、开发难易程度、性价比等各方面因素仔细比较之后我们选择了 CY7C68013A-56Pin 作为我们的 USB 接口主芯片。如 3-4 为 CY7C68013A-56Pin 的封装结构[33]。

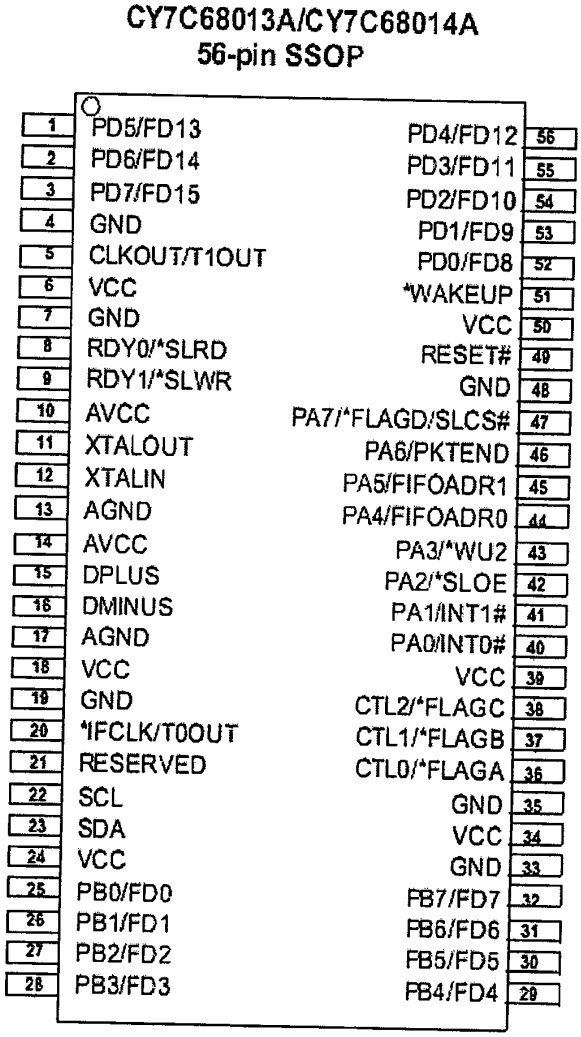

图 3-4　CY7C68013A-56Pin 封装结构

3.2.3　EZ-USB FX2LP 工作模式选择

EZ-USB FX2LP 系列芯片有 3 种工作模式：普通 I/O 工作模式、Slave FIFO 接口模式以及 GPIF 通用可编程模式。

普通 I/O 工作模式。从前面的介绍可知 CY7C68013A 芯片资源中包含有增强型 8051 单片机，因此该芯片能够执行普通单片机可以完成的任务如控制 I/O 口引脚。如图 3-5 所示为 CY7C68013A 芯片的 I/O 引脚基本结构图。

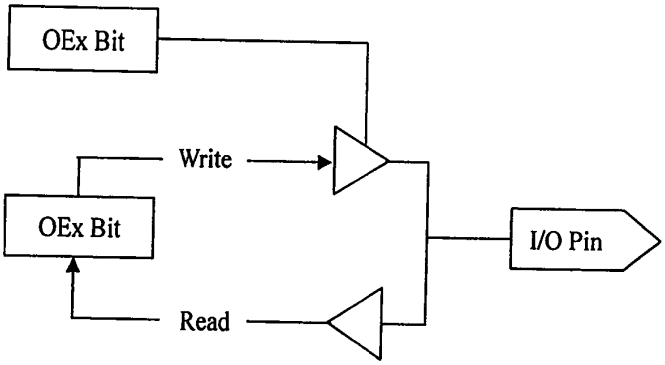

图 3-5　CY7C68013A 芯片的 I/O 引脚基本结构

图中 OEx 特指 OEA,OEB 等特殊功能寄存器，IOx 特指 IOA，IOB 等特殊功能寄存器。OEA 是控制 PA 端口输入输出功能的控制使能寄存器，0 表示输入，1 表示输出。IOA 用于读取或者写入相应 I/O 管脚的数据。其他寄存器 OEB，IOB 等以此类推也是一样的功能。

Slave FIFO 接口工作模式。CY7C68013A 芯片除了具备普通单片机所拥有的功能外，还拥有高效的数据采集与发送机制。当 CY7C68013A 工作在 Slave FIFO 工作模式时，外部主机设备可以把 CY7C68013A 芯片中的端点缓冲区当成普通 FIFO 一样进行数据读写操作。如图 3-6 所示为典型的 Slave FIFO 模式连接图。

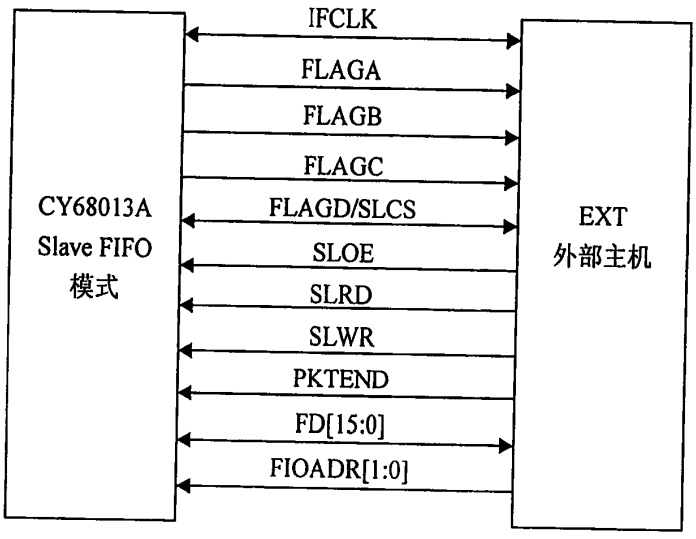

图 3-6　CY7C68013A 芯片 Slave FIFO 工作模式典型连接

GPIF 通用可编程接口工作模式。在 GPIF 工作模式下，CY7C68013A 可以通过软件编程输出读写控制波形图以便控制所有总线接口的访问，该模式下的读写速度很快。如图 3-7 所示为 GPIF 工作模式的典型连接图。

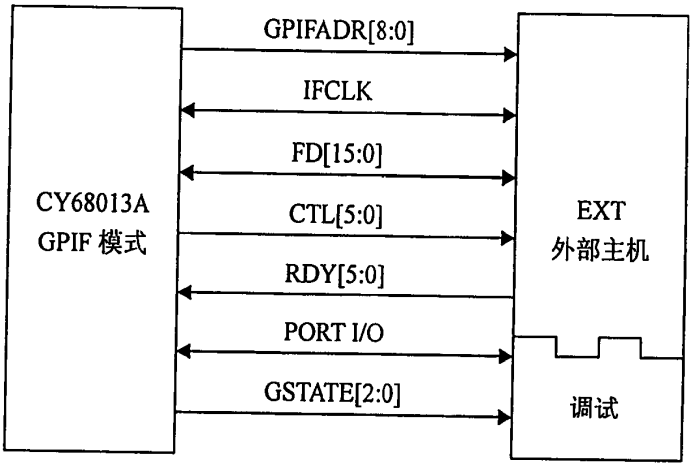

图 3-7　CY7C68013A 芯片 GPIF 工作模式典型连接

在前面项目分析时我们知道数字电视节目流对带宽的要求还是很高的，如果传输当中某个地方出现瓶颈就会对整个系统产生不良影响，轻则出现马赛克效应，重则根本无法播放数字电视节目。因此，我们还需要对 USB 芯片进行设置使它能够以最快的速度把来自 USB 前端解调器输出的 8 位并行数据传输给后端电脑主机，由于采用普通 I/O 口进行输入输出数据不但要占用很多 CPU 系统资源而且传输效率也很低，这就要把 CY7C68013A 设置为 Slave FIFO 工作模式，因为在该模式下通过 USB 传输的数据几乎不需要 CY7C68013A 芯片内部 CPU 的干预，此时芯片工作效率是很高的[34]。在实际应用中一般需要用到高速数据传输的应用如音视频数据采集等都会采用 Slave FIFO 工作模式。图 3-8 为 Slave FIFO 模式下的硬件结构图[32]。

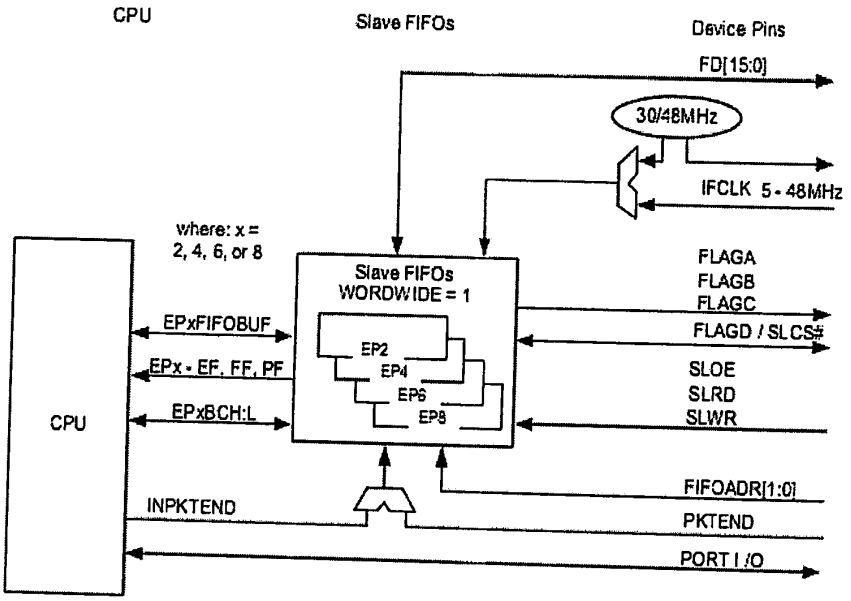

图 3-8　EZ-USB Slave FIFO 模式硬件结构

在 Slave FIFO 工作模式下又有好几种不同的情况，如同步 FIFO，异步 FIFO；同步和异步 FIFO 中又各自有读和写之分。因此分类组合下，总共有 4 种组合：同步 FIFO 读/写，异步 FIFO 读/写。如图 3-9 所示，该图体现了同步与异步模式 FIFO 模式的区别[32]。

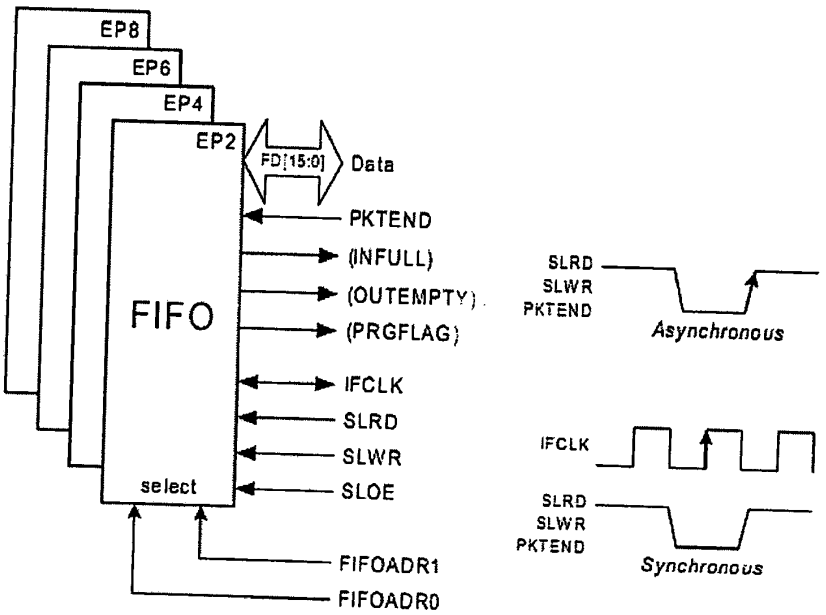

图 3-9　EZ-USB Slave FIFO 同步与异步模式对比

从上图中可以看到，异步模式以 SLRD 和 SLWR 引脚作为读和写的触发信号，比如在每个触发信号的上升沿处开始采集数据线上的高低电平信号；同步模式则以 IFCLK 为时钟信号，SLRD 和 SLWR 为时钟引脚的使能信号。USB 接口的数字电视接收设备其实质就相当于一个数据采集器，USB 芯片要把来自前端解调器传输过来的 MPEG2-TS 数据以最快速度传输给后端主机，因此我们只需要把 USB 芯片设置成 IN 模式，即读数据模式；另外对于数字电视接收而言，我们既可以采用同步方式采集数据，也可以采用异步模式采集数据。在这里我们以异步 Slave FIFO 读数据模式为例，其管脚连接方式、工作时序图以及状态改变机制如下面 3 张图所示[32]。

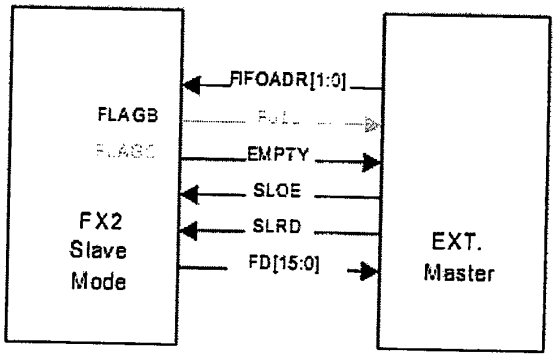

图 3-10　EZ-USB Slave FIFO 异步读模式管脚连接

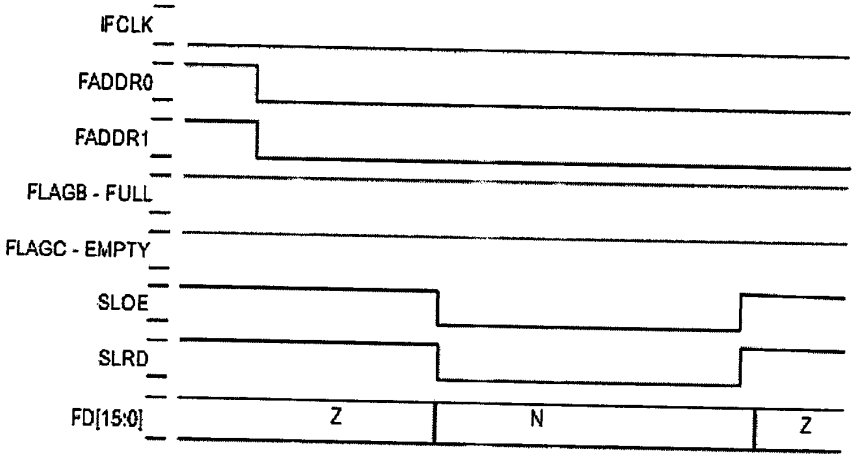

图 3-11　EZ-USB Slave FIFO 异步读模式时序图

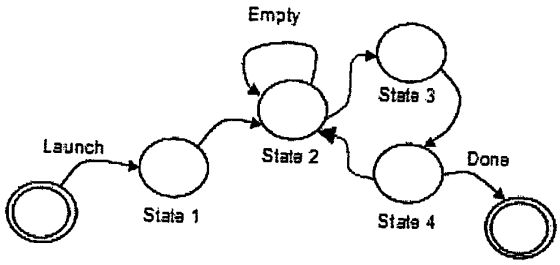

图 3-12　EZ-USB Slave FIFO 异步读模式工作状态机

根据时序图以及工作状态机图我们可以对 EZ-USB Slave FIFO 异步读模式工作状态机的工作原理做如下讲解[28,32]：

IDLE：　空闲状态等待外部主机传递读数据信号，若事件发生则状态机转到 State1。

State1：　数据指针指向输出 FIFO 地址，并触发端点片选引脚 FIFOADR[1:0]，同时切换到 State2 状态。

State2：　如果 FIFO 不为空就跳转到 State3 状态，否则维持在 State2 状态不变。

State3：　读使能信号 SLOE 开启，SLRD 电平跳变的同时采集数据总线上的数据到 FIFO 缓冲区中，然后转向 State4 状态。

State4：　如果数据未读完，则跳转到 State2 继续读，否则读取完成，转向 IDLE 状态。

为了尽量提高数据传输速率，我们需要把 CY7C68013A 内部 FIFO 缓冲区端点 EPx 设置得尽可能的大，然而由于内部资源以及芯片设计的限制我们最多只能使用 8*512 字节的空间，另外我们只需要从一个端点读取数据就可以了，因此根据官方芯片手册上的配置图，我们可以把 EP2 端点设置成 4 重 1024 字节大小的缓冲区，在图 3-13 所示的最右边一列数字 12 所对应的设置就是 4 重 1024 字节的设置[32]。

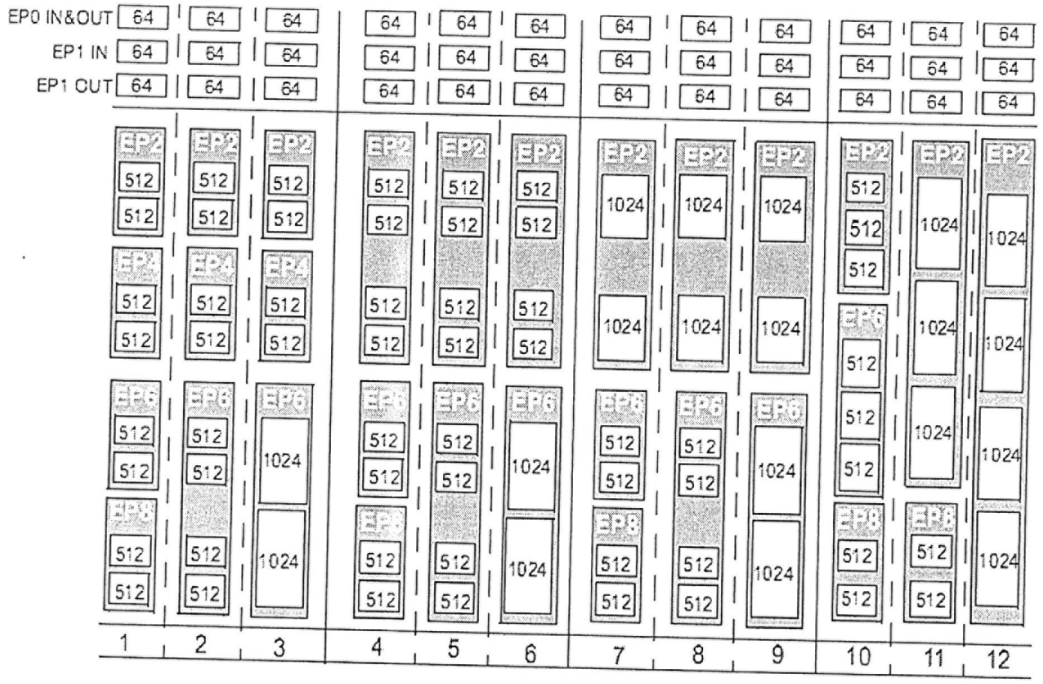

图 3-13　CY7C68013A 端点配置

3.3　USB 固件程序开发

3.3.1　Cypress 公司的 USB 固件程序介绍

固件程序就是 USB 设备的核心与灵魂，各种数据的发送与接收都是在固件程序里面实现的。固件程序相当于 USB 设备端的驱动程序，是 USB 设备端的“首脑”与“外交官”负责与主机端驱动程序“交流”。因此，USB 固件程序与 USB 主机端驱动程序往往都是配合着开发的。Cypress 公司为他们的产品专门设计了固件程序的框架代码，USB 开发人员可以直接在基础框架代码上添加他们想要实现的各种额外功能。Cypress 提供的固件框架主要包括以下几个程序文件[26,32,35]：

Fw.c：USB 固件程序的主文件，main()函数就在这个文件中，主要负责设备连接、重枚举、设备初始化如一些调度、SETUP 命令的处理等。USB 芯片控制寄存器的初始化、以及一些自定义的函数也可以在这个文件中添加。

Periph.c：该工程文件包含了 USB 芯片初始化相关函数以及任务处理函数，I/O 口配置，时钟频率，端点选择，FIFO 参数设置等等，以及负责相应各种中断响应事件如 USB（INT2）中断和 GPIF（INT4）中断的请求函数的定义。用户也可以在该文件中添加自己需要的函数功能。

Dscr.a51：这是一个汇编文件，包含了 USB 设备在握手连接时需要的各种设备描述符，这里的参数要与程序中的寄存器相对应，否则会导致 USB 设备运行出错或者无法识别等

问题。

Lp.h：这是一个头文件，包含了整个工程文件中要用到的通用常量、宏、数据类型以及库函数的原型。

Lpregs.h：头文件，EZ-USB 的相关寄存器的声明以及位标志定义都在该文件中。

Syncdly.h：头文件，包含了同步时延的宏定义，用于一些需要延时或者同步的寄存器。

EZUSB.lib：EZ-USB 的函数封装库，提供了与外部主机交互的一些控制函数的实现。

USBJMPTB.obj：提供了 USB 以及 GPIF 的模式的中断服务向量跳转表。

其中 fw.c 和 periph.c 这两个文件是需要根据我们自己的功能进行修改的，Dscr.a51 文件也要做少量的修改以配合前面的改动。其他一些文件基本上就不需要修改了。修改好的程序在 keil c uVision 软件上编译通过后就能生成.hex 和.iic 文件，.hex 文件可以直接下载到 USB 的 RAM 中运行，.iic 文件可以下载到 USB 的 EEPROM 中，以后重新启动 USB 就会自动把程序导入到 RAM 中运行。如图 3-14 所示为用 Keil uVison 软件打开固件程序 C 工程文档的截图。

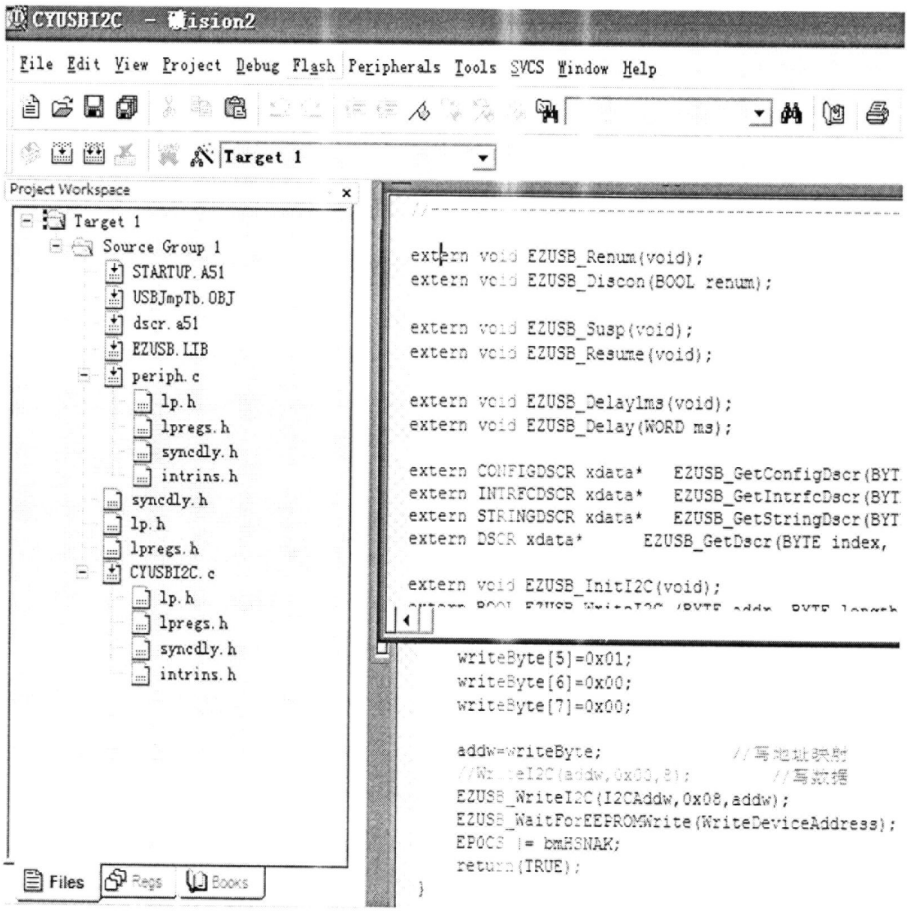

图 3-14　固件程序代码 Keil uVison 软件中截图

3.3.2 Slave FIFO 模式的固件程序设计

在经过3.4节对Slave FIFO工作模式的详细介绍之后我们只要配合芯片手册上的相关寄存器的说明就能在 Cypress 公司提供的框架固件程序中添加修改自己的代码实现 Slave FIFO 异步数据采集功能。如图 3-15 所示为固件程序的工作流程图。

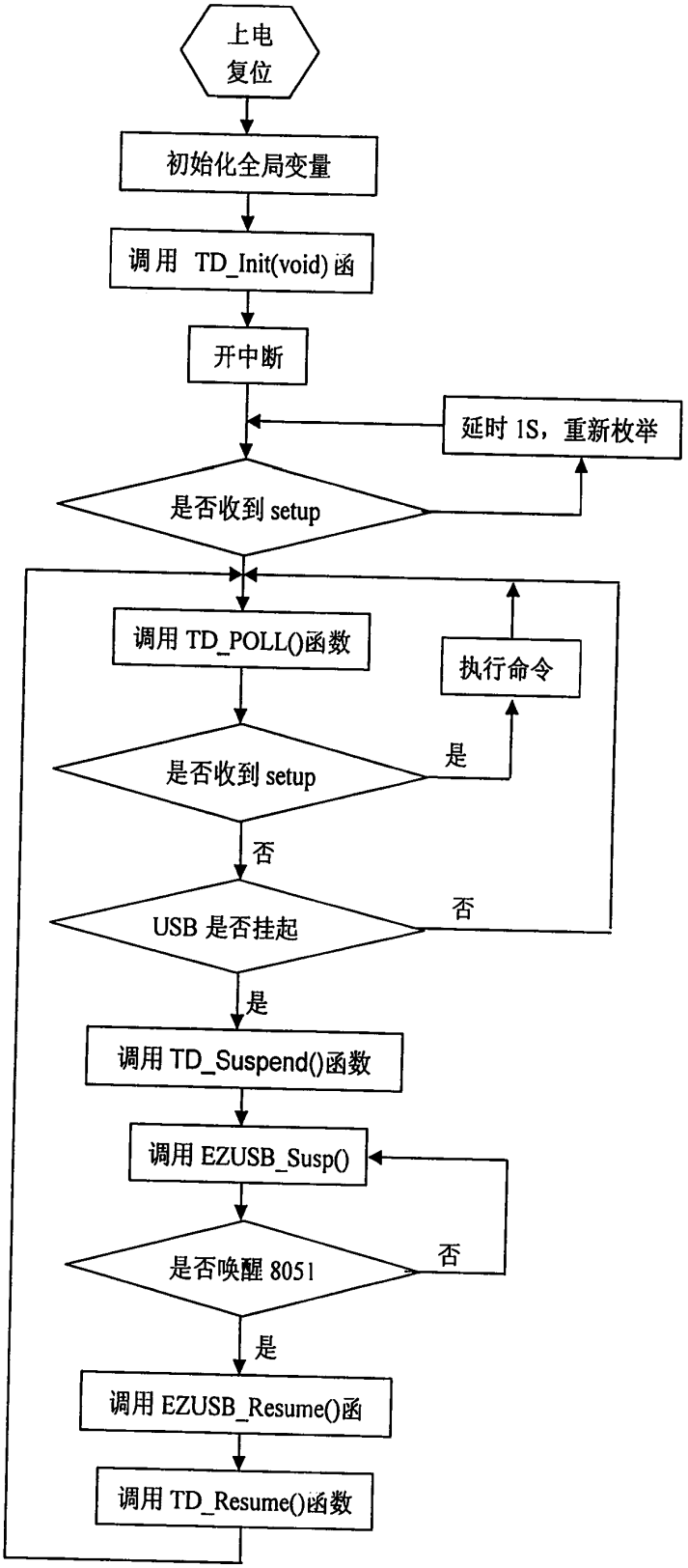

图 3-15　固件程序工作流程图

结合固件程序流程图以及前面 USB 事务处理的相关知识我们来具体解析下完整的

USB 识别过程[36,37,38]：

第一步：USB 接口插到电脑主机的瞬间此时仍然还没有通电，USB 芯片 VCC 管脚还没有电压，经过极短暂的时间后到 VCC 上有正 5V 电压。此过程中电脑主机和 HUB 会通信。

第二步：USB 虽然已经通电，但此时 USB 芯片寄存器都未被初始化，此时 USB 无法正常接收任何事务响应，也未被电脑主机分配任何设备地址。集线器根据 D+D-管脚上的电压来判断是否有新的 USB 外设连入主机。当集线器判断出有新的 USB 设备接入后就向电脑主机汇报。此过程中电脑主机和 HUB 也会通信。

第三步：复位 USB 总线。电脑主机通知集线器复位 USB 总线，USB 传输模式被获取到，将工作在低速、全速或者高速模式。复位后，USB 外设将被电脑主机分配一个默认的设备地址号 0。此过程中电脑主机和 HUB 也会通信。

第四步：USB 外设获得主机分配的默认 0 号地址之后便可使用该地址号进行某些事务处理。电脑主机可以开始和 USB 外设功能层进行通信，即可以一些进行控制数据传输。

第五步：电脑主机会试图获取 USB 外设的信息，于是它要向 USB 外设发送获取设备描述符的命令请求，这一过程将要启动一个 USB 事务处理。而前面章节介绍过 USB 事务处理包括令牌阶段、数据阶段和握手阶段三个阶段。在这一步中，电脑主机向 USB 外设发送 GetDescription 请求读取设备描述符来获取 USB 控制传输所支持的最大数据包长度。上面的过程是一个 USB 事务，而所有的 USB 事务都有从令牌包开始，于是 USB 固件只能先等待令牌包的到来，之后才能处理相应的命令。这样，电脑主机通过发送 GetDescription 这一标准请求读取设备描述符 DeviceDscr 获取到控制传输支持的最大字节数——第 8 个字节，之后将再次复位 USB 总线并准备开始进入枚举过程。

第六步：开始进入枚举过程。电脑主机向 USB 外设发送 SetAddress 请求，为它分配一个新的，唯一的设备地址（0～127，总共 128 个）。此后，USB 外设将用新分配的地址号与电脑主机进行通信。

第七步：为了获得该 USB 设备的全部配置信息，主机不断地向 USB 外设发出 GetDescription 请求来读取所有的 USB 设备描述符。首先读取设备描述符 DeviceDscr 全部字段，然后是配置描述符 Configuration，接着是字符串描述符，接口描述符，端点描述符，设备限定描述符，其他各种设备类描述符以及自定义描述符等。

第八步：随后电脑主机根据读取到的 PID，VID 选择一个合适的 USB 驱动程序并加载。如果是第一次使用该 PID，VID 的设备则提示发现新硬件并要求你安装驱动程序。

第九步：加载完 USB 驱动程序之后，电脑主机发送 SetConfigration 请求为该 USB 外设选择一个合适的配置，随后该 USB 外设将被分配一个配置值，一个接口，一个可替换设置值。

第十步：至此，整个 USB 枚举全过程结束。

经过以上分析，我们对 USB 枚举已经有了清晰的了解，接下来我们着重对固件编程中需要重视的以及一些和本文相关性比较大的代码进行讲解。

一、设备描述符修改：

```
DeviceDscr:
    ......
    dw    0B404H        ;; Vendor ID//供应商 ID 号，默认值为 04B4
    dw    0410H         ;; Product ID (Sample Device)//产品 ID 号，默认值为 1004
    ......
```

上面这 2 个值在产品应用中需要修改成自己的值，而且需要和驱动程序中的一致，否则 USB 设备将会无法正常工作。

二、字符串描述符修改：

```
StringDscr:
    db    'C',00
    db    'y',00
    db    'p',00
    db    'r',00
    db    'e',00
    db    's',00
    db    's',00
```

默认字符串为 Cypress，这里我们可以按需要修改成自己的产品公司名称。

三、端点描述符修改：

```
;; Endpoint Descriptor
    db    DSCR_ENDPNT_LEN       ;; Descriptor length
    db    DSCR_ENDPNT           ;; Descriptor type
    db    02H                   ;; Endpoint number, and direction
    db    ET_BULK               ;; Endpoint type
    db    40H                   ;; Maximun packet size (LSB)
    db    00H                   ;; Max packect size (MSB)
    db    00H                   ;; Polling interval
```

由于我们只需要把 EP2 即端点 2 设置成 IN 模式且是 4*512 长度，因此我们要把 02 修改成 82，最大数据包程度修改成 2048，即 0800H。

四、设置为 Slave FIFO 模式需要在 periph.c 文件中进行如下修改：

```
void TD_Init(void)
{
......
  //使用内部的 48MHZ
```

```
CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1) ;
IFCONFIG=0XCB;          //异步 Slave FIFO 模式
SYNCDELAY;              //同步时延
REVCTL=0X03;            //置 REVCTL.0=1 和 REVCTL.1=1
SYNCDELAY;
EP2CFG=0XE8;            //置端点 EP2 为 IN 有效，
SYNCDELAY;              //块传输，四重缓冲，1024 字节
FIFORESET=0X80;         //复位各个端点
SYNCDELAY;
FIFORESET=0X02;
SYNCDELAY;
FIFORESET=0X00;
SYNCDELAY;
EP2FIFOCFG=0X0C;        //自动处理 IN 数据包
PINFLAGSAB=0X00;        //FLAGA 为可编程标志
SYNCDELAY;              //FLAGB 为 FIFO 满标志
PINFLAGSCD=0X00;        // FLAGC 为 FIFO 空标志
PORTACFG=0X00;          //PA7 作为普通 I/O
SYNCDELAY;
FIFOPINPOLAR=0X00;      //低电平有效
SYNCDELAY;
EP2AUTOINLENH=0X04;  //数据包大小为 1024 字节
SYNCDELAY;
EP2AUTOINLENL=0X00;
SYNCDELAY;
EP2FIFOPFH=0X80;
SYNCDELAY;
EP2FIFOPFL=0X00;
SYNCDELAY;
……
IIC_init();     //初始化 I2C 总线接口配置
Tuner_reset(); //通过 I2C 控制调频器初始化
Demo_reset(); //通过 I2C 控制解码器初始化
……
}
```

五、添加自定义函数。固件程序需要负责对调谐器的控制以便实现对调谐器进行初始与复位以及对输入频率进行选择，还要对调制解调器进行初始话与复位等控制以及检查数据输出是否正确。而对调谐器以及调制解调器的控制是通过 IIC 总线接口实现的，我们首先需要先在 periph.c 文件中定义自己的函数功能，然后在 void SetupCommand(void)函数中添加自己的函数名，当然我们还需要在头文件中添加自定义请求号，如定义读 IIC 总线的请求号：#define SC_ReadI2C 0xB9，这样电脑主机就可以通过发送该自定义请求号 B9 来调用固件程序中的相关代码。例如我们添加了对 IIC 总线的读写功能，以及通过 IIC 总线

对调谐器和解调器控制等相关函数，相关代码如下：

```
void SetupCommand(void)
{
……
case SC_ReadI2C: //自定义请求号
                Read_I2C();//自定义函数
                break;
case SC_WriteI2C:
                Write_I2C();
                break;
……
case SC_ReadTuner:
                Read_Tuner();
                break;
case SC_WriteTuner:
                Write_Tuner();
                break;
……
case SC_ Read Demo:
                Read _Demo();
                break;
case SC_WriteDemo:
                Write_Demo();
                break;
……
}
```

以上就是固件程序设计中需要重点注意的地方，整个固件程序工程文件编写好了之后就可以进行编译和安装了。由于本文主要研究重点是数字电视接收设备的后端软件设计，因此这里还有很多的有关前端硬件的函数代码以及参数设置我们不再做过多讲解。

3.4 固件程序的编译与安装

固件程序的编译还是要通过 Keil uVision 软件，我们需要对 Keil uVision 软件进行一些设置才能通过该软件最终生成 SlaveFIFO.hex 和 SlaveFIFO.iic 这两个二进制可执行文件。我们打开 Keil uVision 软件并导入我们的工程文件，按照如图 3-16 所示那样设置，设置完成之后点击 Build 按钮之后我们的 2 个可执行文件就生成了。

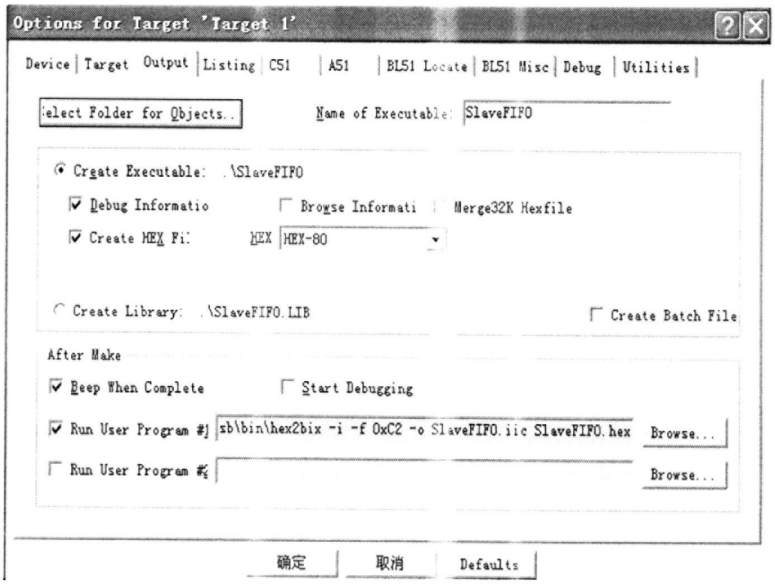

图 3-16　Keil uVision 软件设置

只要把这 2 个文件下载到 USB 芯片中，再配合驱动程序我们就能在电脑主机端控制 USB 外设进行一系列操作了。如图 3-17 所示为我们通过 Cypress 公司提供的 USB 开发工具 EZ-USB Control Panel 把固件.iic 格式的固件程序下载到 USB 芯片引出的 EEPROM 中，这样下次 USB 重新上电的时候就会运行我们下载进去的固件程序了。

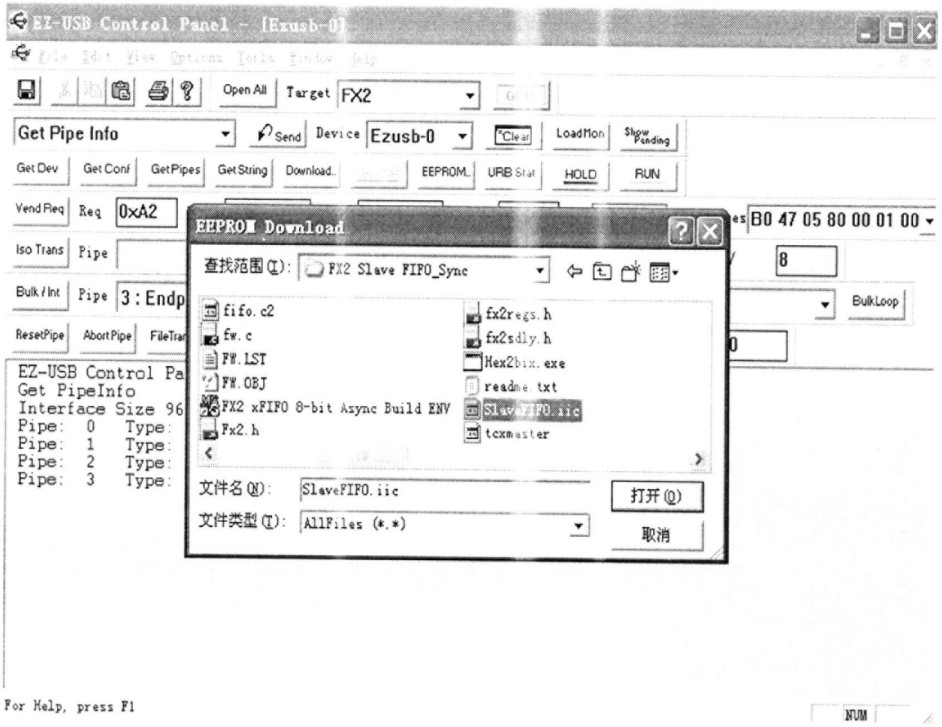

图 3-17　下载固件程序到 EEPROM 中

第 4 章 Windows 系统下 USB 主机驱动程序开发

接触过电脑的人都知道在 Windows 操作系统下我们不能直接对外部的硬件设备直接进行操作，比如安装一块网卡之后我们不能直接上网，我们需要安装好相应的驱动程序之后才能上网。可见驱动程序是连接电脑主机与外部硬件设备的纽带与桥梁。同样 USB 设备也需要驱动程序才能与电脑主机进行通信。USB 驱动程序就相当于一个“中间人”，电脑主机发送命令给 USB 驱动程序这个“中间人”，然后 USB 驱动程序又把收到的命令转化成 USB 设备能够认识的命令，最后 USB 设备便根据收到的命令执行各种动作。

4.1 EZ-USB 驱动程序设计方案介绍

在 Windows 操作系统这一开发平台下开发驱动程序常用的有以下几种方法：一、使用微软自己的驱动程序开发套件 DDK/WDK 开发驱动程序，这是最常用也是最原始同样也是难度最大的驱动程序开发方式；二、使用第三方驱动开发工具，如美国 KRFTech 公司出品的用于编写设备驱动程序的 WinDriver 开发工具，以及美国 Compuware 公司出品的 DriverStudio 工具[31]。使用第三方开发工具可以使产品开发速度大大提高，我们只要进行简单的操作就能开发出通用的驱动程序。三、使用 USB 芯片公司提供的驱动程序。

对于 Cypress 公司的 USB 芯片来说，以上说的 3 种方法都可行，首先使用微软的 DDK 驱动开发套件必然是可行的，其实其他开发工具都是在它的基础上进行驱动开发的；其次 WinDriver 和 DriverStudio 都提供了对 USB 驱动程序开发的支持，因此也可以采用这两个工具进行驱动开发；最后还有最方便的就是使用 Cypress 公司的提供的通用 USB 驱动程序，这个对于没有编程基础的开发人员来说是最方便的。当然使用第三方驱动开发软件是要钱的，因此，我们在接下来的章节主要还是介绍另外两种方法开发 USB 驱动程序。

4.2 Cypress 公司的通用 USB 驱动程序

首先我们从最简单快捷的方式说起，即直接利用该公司提供的驱动程序并根据实际情况修改成我们自己所需要的样子。不过 Cypress 公司并没有提供该公司的 USB 芯片驱动程序源码，而是用了一个折中的方法发布他们的驱动程序，他们对驱动程序的源码进行特殊设计，使得用户可以在不需要修改驱动源程序的基础上，只修改 inf 文件中相关设置就

能做出一个“属于用户自己的驱动程序”。但这样制作出来的驱动程序，用户是没有知识产权的，而且这样做出来的一个通用驱动程序虽然通用性比较高，但由于有很多无关代码，总体来说效率不是非常高，因此，对于大多数想要开发 USB 产品的公司来说，他们大多都会从头开始设计一个真正适合自己产品的驱动程序。

Cypress 公司提供的 CY3684 EZ-USB FX2LP 开发套件包括了上面所说的通用 USB 驱动程序，而且还附带了相应的主机端控制函数库以及一些说明文档，通过这驱动程序和函数库我们就可以编写 VC++程序来控制 USB 外设了。我们选用的 CY68013A 芯片同样适用该驱动程序。配套的驱动程序包括了如下几重要个文件：Cyload.sys、Cyload.inf、CyUSB.sys、CyUSB.inf 以及 CyUSB.html 帮助文件。

Cyload.sys 与 Cyload.inf 顾名思义当然是和固件程序加载相关的，Cyload.sys 是用来加载固件程序的主程序；Cyload.inf 是配合 Cyload.sys 进行文件安装引导的。CyUSB.sys 与 CyUSB.inf 这 2 个文件时与 USB 通信相关的驱动程序，CyUSB.sys 中实现了电脑主机与 USB 接口通信的各种函数功能，比如读取 USB 描述符，读取 USB 端口数据，向 USB 端口写数据等等功能；CyUSB.inf 文件同样是配合 CyUSB.sys 进行文件安装引导用的。

上面这些文件我们可以直接拿来使用，也可以修改里面的某些字段以区别同类驱动程序。比如我们可以修改 Cyload.inf 文件中 YourCompany 字段和 Strings 字段中的内容，其内容如下[32,33]：

```
[YourCompany]
%DeviceDesc%=CyLoad, USB\VID_04B4&PID_0084
[Strings]
YourCompany          = "YourCompany"
MfgName              = "YourCompany"
;----------Replace GUID below with custom GUID (generated with GuidGen)----------;
CyLoad.GUID          = "{259E1A84-51FE-4c77-9A7F-894904D6D7C3}"
CyLoad.SCRIPTDIR     = "\systemroot\system32\CyLoad\"
CyLoad.SCRIPTFILE = "CyLoad.spt"
;----------Modify these strings to match your device----------;
CyLoad_INSTALL       = "Your Device Installation Disk"
DeviceDesc           = "Your Device Firmware Download"
CyLoad.SvcDesc       = "Your Device Download Driver"
```

一、可以把其中的 VID_04B4&PID_0084 改成自己想要的 VID 与 PID 号如 VID_05B5&PID_0095，这对一个成熟的产品来说是必须的，因为驱动程序与 USB 外设之间的对应就是根据这 2 个值来的。

二、可以修改 GUID 号，而且这也是必需的，因为电脑主机就是根据这个全球唯一 GUID 号来区别不同 USB 外设的。可以微软官方网站上下载 GUIDGEN.EXE 这个工具生成一个全新的 128 位的 GUID 号，替换上面的{259E1A84-51FE-4c77-9A7F-894904D6D7C3}这个 GUID 号。

三、还可以修改 YourCompany、MfgName、DeviceDesc 及 CyLoad.SvcDesc 这些字符串的值，这样下次同样 VID 与 PID 的 USB 设备就会显示成你修改后的设备名称、制造商名称等字符串了。

四、因为这个驱动程序是用来加载固件程序的，而我们需要加载的固件程序也要放在同一个文件夹下面，而且我们还需要把固件程序的格式转换成.spt 格式的，固件文件名字在 CyLoad.SCRIPTFILE = "CyLoad.spt"这一行修改。

接下来可以修改 CyUSB.inf 文件，其几个重要字段如下所示[32,33]：

```
[Manufacturer]
%MFGNAME%=Cypress
[Cypress]
; Cypress FX2LP default ID
%VID_04B4&PID_8613.DeviceDesc%=CyUsb, USB\VID_04B4&PID_8613
; Cypress FX1 default ID
%VID_04B4&PID_6473.DeviceDesc%=CyUsb, USB\VID_04B4&PID_6473
; Cypress FX2LP Example Device.  DVK firmware examples use this VID/PID
%VID_04B4&PID_1004.DeviceDesc%=CyUsb, USB\VID_04B4&PID_1004
[Strings]
PROVIDER="Cypress"
MFGNAME="Cypress"
CYUSB_INSTALL="Cypress Generic USB Driver Installation Disk"
VID_04B4&PID_8613.DeviceDesc="Cypress EZ-USB FX2LP - EEPROM missing"
VID_04B4&PID_6473.DeviceDesc="Cypress EZ-USB FX1 - EEPROM missing"
VID_04B4&PID_1004.DeviceDesc="Cypress EZ-USB Example Device"
CyUsb.SvcDesc="Cypress Generic USB Driver"
;CyUsb.GUID="{AE18AA60-7F6A-11d4-97DD-00010229B959}"
```

与 CyLoad.inf 基本上差不多，我们也同样可以修改 VID、PID、GUID 以及其他一些字符串，不过我们需要注意的是这里的 VID 及 PID 至少要有 1 个与你固件程序里面 dscr.a51 中描述的 VIP 及 PID 一致，否则在安装 USB 驱动程序过程中会出现找不到驱动程序而无法正常识别该 USB 设备的问题。

这样经过简单的几步修改，一个适合自己的驱动程序就完成了。配合第 3 章讲的固件程序我们的电脑就能正常识别该 USB 外设了。如图 4-1 及 4-2 所示为我们设计的基于

CY68013A 芯片的 USB 数据采集设备模块在电脑主机上的截图，图中的软件是 Cypress 公司提供的 USB 开发工具。同时配合 Cpress 公司提供的与驱动程序配套的应用程序开发库文件我们就能开发出自己的 USB 上位机控制软件了。如图 4-3 所示为我们设计的与数据采集设备配套的上位机软件截图。

关于驱动程序的开发和 inf 文件的编写以及上位机软件的编写我们将在后面章节具体介绍。

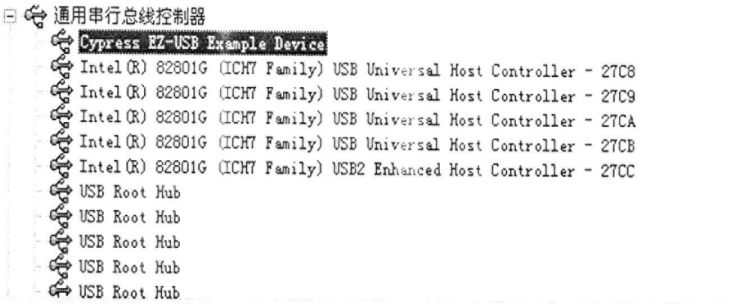

图 4-1　USB 外设在电脑主机上安装驱动后被识别

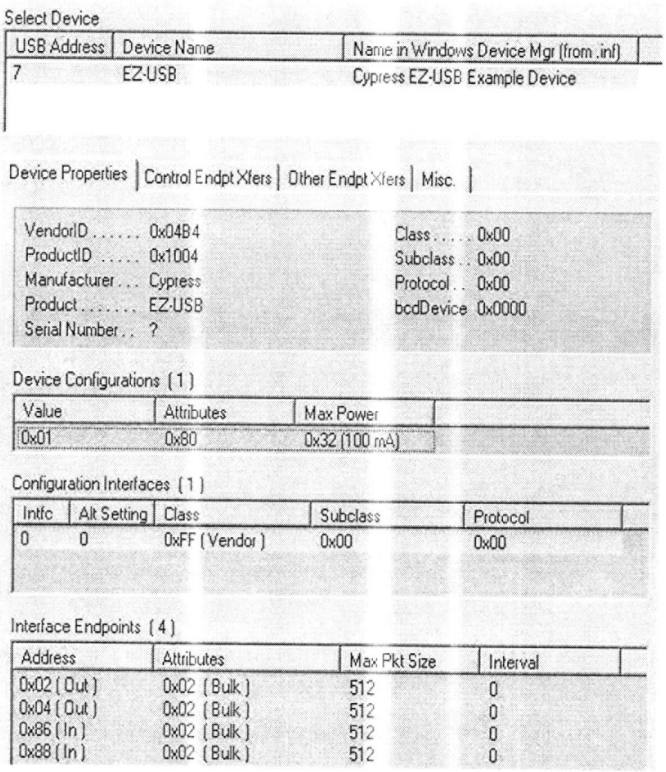

图 4-2　USB 外设在电脑主机上用工具软件读取到的配置

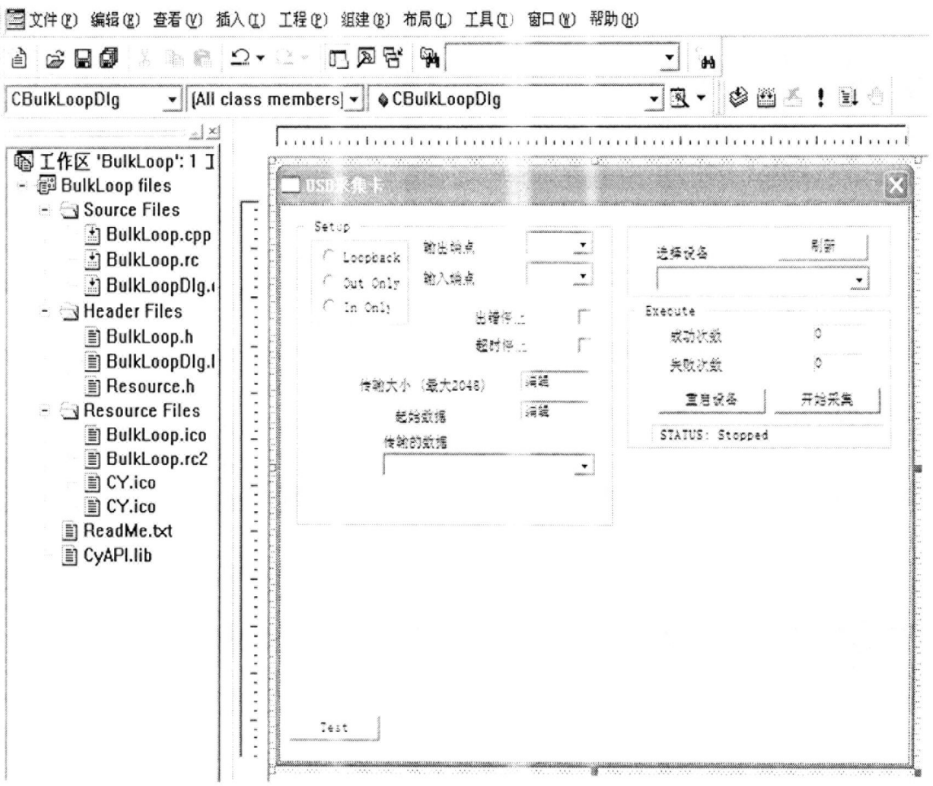

图 4-3　USB 上位机软件

4.3　基于微软 DDK 开发套件开发 USB 驱动程序

4.3.1　驱动程序的分类

在 Windows 开发环境下编写程序可以简单的划分成两类：一类是普通的应用程序编写，还有一类是内核层程序的编写。普通应用程序编写相对来说简单很多，因为它运行在用户层，不是运行在 Windows 的核心层，一般不会对 Windows 操作系统稳定性造成影响。而运行在内核层的驱动程序则不然，一个细微的错误就能导致 Windows 操作系统死机、蓝屏，甚至连硬件设备也会被毁坏，因此驱动程序不是随随便便可以乱写的。图 4-4 为操作系统基本结构图[39]。

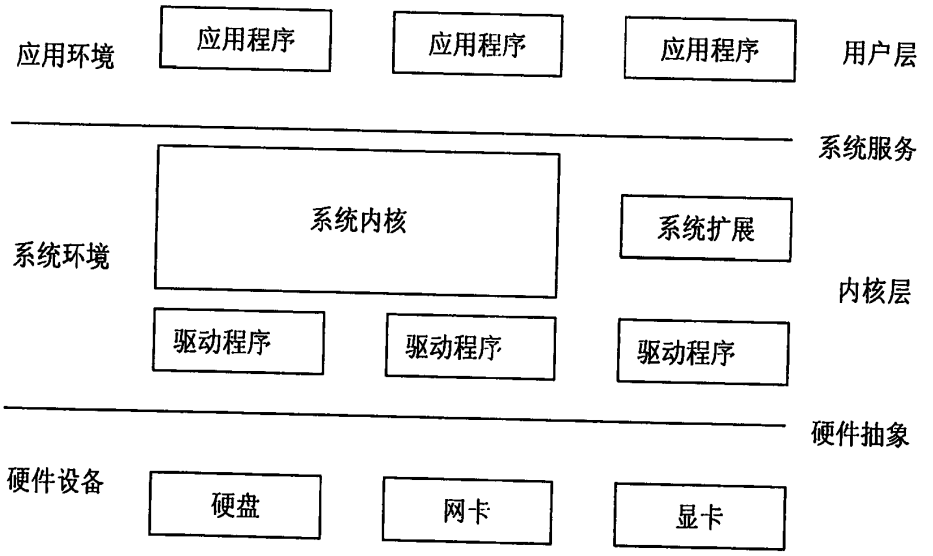

图 4-4 操作系统基本结构

驱动程序也可以简单的划分成两类：第一类是不进行硬件操作的，单纯地运行在 Windows 内核层的 NT 式驱动程序，这类驱动不支持即插即用功能；另一类是与硬件打交道的支持即插即用功能的 WDM（Windows Driver Model）式驱动程序，当然作为驱动程序，它同样运行在内核层。

NT 式驱动程序是基于服务进行安装的，可以直接在注册表里面添加具体服务项进行安装，也可以在程序中通过服务创建函数 CreateService 进行安装；而 WDM 式驱动程序则需要一个 inf 文件配合才能完成安装。

4.3.2 USB 驱动程序的运行原理简介

USB 设备是一个可以即插即用的硬件设备，USB 驱动程序显然是要使用 WDM 框架编写的。前面章节提到的 Cypress 公司提供的 USB 驱动程序也不例外，它也是 Cypress 公司基于微软 DDK 开发套件开发的 WDM 式驱动程序。

在编写程序之前我们结合图 4-4 对 USB 接口设备整个工作流程以及原理做一个介绍。我们研究的 USB 数字电视接收设备从根本上来说属于数据采集类设备，只是比普通的数据采集设备更加复杂一点功能更多一点。为了更简单明了的表述清楚 USB 设备的工作原理我们就以 USB 数据采集卡为例进行解说。这里我们假设一切硬件设备都连接好了，相关的驱动程序也都安装好了，USB 采集设备也可以正常工作了。我们先打开 USB 数据采集卡配套的上位机软件，正常情况下我们的上位机软件会提示获取到 USB 数据采集卡之

类的信息，然后我们只要点击一下软件上的采集数据功能键，之后 USB 数据采集卡就开始源源不断地采集数据，并且把采集到的数据保存到你的电脑硬盘上了。你点击采集数据键这一个简单的动作其实我们的 Windows 操作系统在后帮我运行了几千几万行代码[40,41]：

一、 首先我们知道采集卡上位机软件是工作在用户层的一个应用软件，在我们点击采集数据键的时候 Windows 操作系统调用了用户层的读文件（ReadFile）这类 API 接口函数。

二、 ReadFile 是用户层的函数，它会通过驱动程序调用内核层的读文件（ZwReadFile）这类 I/O 处理函数（这里先简单这么理解，其实是向内核发送一个叫做 IRP 的 I/O 请求包）。

三、 ZwReadFile 函数会发消息给 USB 设备，USB 收到一个事务处理，USB 固件程序开始执行相应的事务处理函数并从相应端点缓冲区中读取数据。

四、 需要采集的数据从 USB 端点缓冲区经过电脑主机内核层再到用户层，最后在用户层中把数据存成文件保存在硬盘上。

一次数据采集过程大致就经过以上步骤完成了数据的采集。图 4-5 为 Windows 操作系统的基本运行模型[39]。

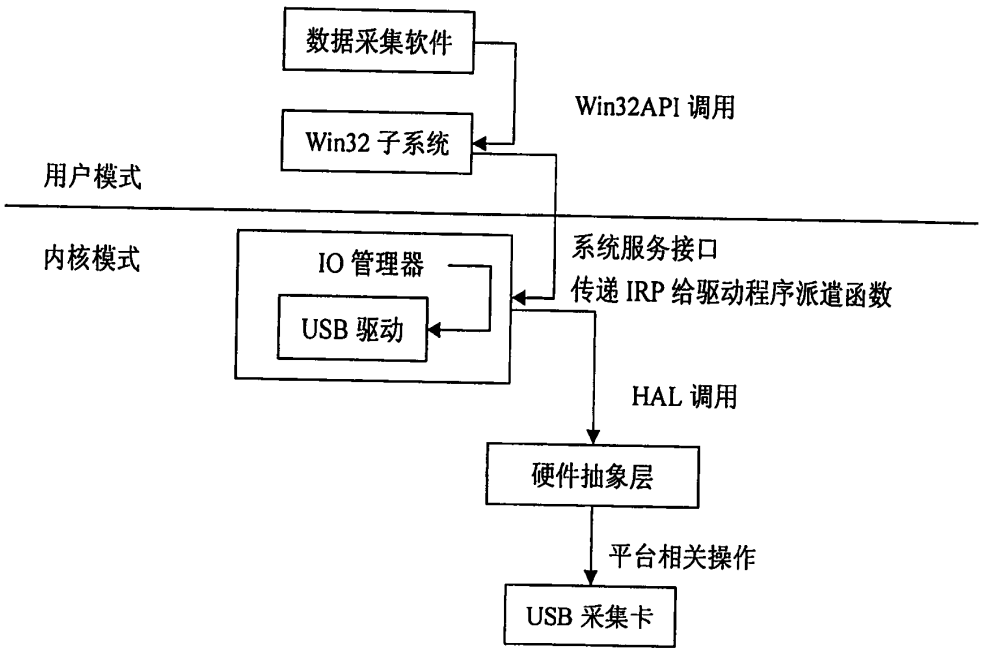

图 4-5 Windows 操作系统的基本运行模型

4.3.3 基于 DDK 开放套件的 WDM 模型 USB 驱动程序设计

在了解了 USB 驱动程序的运行机制之后，接下来我们来具体讲解下如何从头开发一个 WDM 式的 USB 驱动程序。在 WDM 式驱动模型中，每个硬件设备至少会包含两个驱动程序：功能型驱动程序和总线型驱动程序[40]。总线型驱动程序主要负责响应设备、管理设备以及向操作系统报告总线的状态等功能，Windows 操作系统下总线驱动程序已经由微软集成好了。功能型驱动程序主要是用来管理具体硬件设备的，如设备的上电、断电、数据读写等操作，一般是有具体开发该硬件设备的生产商或公司自己开发的。USB 驱动程序属于功能型驱动程序，它需要由 USB 设备制造商自己提供。另外在 WDM 模型中，要实现对一个硬件设备的操作至少需要两个设备对象：物理设备对象（PDO）和功能设备对象（FDO）[40]。只有当 FDO 附加到 PDO 之后才能对物理设备进行操作。当新的硬件设备插入电脑的时候总线驱动程序会创建 PDO，并提示有新的未知设备要求我们安装驱动，由于此时没有 FDO，因此无法识别该设备也无法操作该设备。而 FDO 正是由我们的 WDM 驱动程序负责创建的，当我们把驱动程序安装好了之后并重新插入设备的时候 WDM 驱动程序就会创建 FDO 并附加到 PDO 之上，之后才能对设备进行进一步操作。

编写过普通 Win32 应用程序的人都知道，一个程序都有一个主函数 main 函数，程序就是从主函数的第一行开始一步一步的往下执行的。与普通应用程序类似 WDM 驱动程序也有一个入口函数地址 DriverEntry 函数。图 4-6 为 WDM 驱动程序的框架结构[26]。

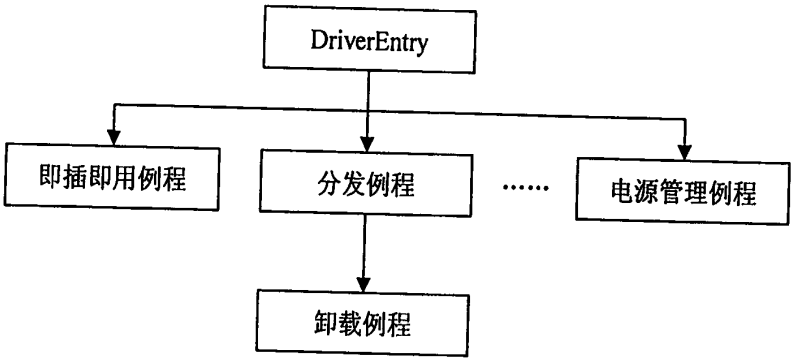

图 4-6 WDM 驱动程序框架

驱动程序所要执行的功能首先都要在 DriverEntry 函数中进行设置。一个驱动程序可以包括各种各样的功能，比如初始化该驱动程序本身，读文件，写文件，I/O 端口操作，定时，电源管理，实现即插即用，调用其他驱动程序等等，但是做一个 USB 驱动程序我

们并不需要所有这些功能，我们只需要把我们想要的功能全部实现就可以了。由于我们需要的是 USB 的数据采集功能，只需要用到块传输功能，而不需要同步传输功能，因此我们完全没必要在驱动程序中去实现同步传输的相关代码，但是普通的控制传输功能我们还是需要实现的，因为毕竟我们还需要对 USB 设备进行各种控制。另外如果讲解全部代码涉及到的内容太多会造成篇幅过长，因此，本文在下面的程序讲解中就把最重要的两个部分的代码进行了讲解，一个是入口函数，另外一个就是控制读写的函数。

如下所示为我们的 USB 驱动程序 DriverEntry 函数的部分代码[40]：

```
NTSTATUS
DriverEntry(//驱动入口函数
    IN PDRIVER_OBJECT DriverObject,//参数一，驱动对象
    IN PUNICODE_STRING RegistryPath//参数二，符号链接
    )
{
   ……
//设置 USB 创建例程
    DriverObject->MajorFunction[IRP_MJ_CREATE] = BulkUsb_Create;
////设置 USB 关闭例程
    DriverObject->MajorFunction[IRP_MJ_CLOSE] = BulkUsb_Close;
//设置 USB 卸载例程
    DriverObject->DriverUnload = BulkUsb_Unload;
//设置 USB 输入输出控制例程
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = BulkUsb_ProcessIOCTL;
//设置 USB 即插即用例程
    DriverObject->MajorFunction[IRP_MJ_PNP] = BulkUsb_DispatchPnp;
//设置 USB 电源例程
    DriverObject->MajorFunction[IRP_MJ_POWER] = BulkUsb_DispatchPower;
//设置 USB 读写例程
    DriverObject->MajorFunction[IRP_MJ_READ]              =
    DriverObject->MajorFunction[IRP_MJ_WRITE]             = BulkUsb_DispatchReadWrite;
//设置 USB 设备初始化例程
    DriverObject->DriverExtension->AddDevice = BulkUsb_PnPAddDevice;
    ……
}
```

如程序中所示我们设置了 USB 的创建、删除、设备读写控制、即插即用、电源管理等派遣（dispatch）函数地址以及初始化函数 AddDevice 和卸载函数 DriverUnload 函数的地址。这样当用户层应用程序进行相应操作时，就会发送相应的 IRP 请求包过来，系统内核就会根据对应的 IRP 请求包找到对应的派遣函数地址并执行相应的操作。从这里我们也可以看出来，驱动程序不是主动执行的，而是“被执行的”，比如应用程序发送了读

USB 端口命令，就会发送一个 IRP_MJ_READ 的 IRP 请求包到内核层，系统内核就会找到对应的派遣函数 BulkUsb_DispatchReadWrite 并执行之。其他操作也是同样的道理。

当然上面 DriverEntry 中的程序只是起到了一个说明或者说索引作用，各个派遣函数的具体实现代码是在程序的后面实现的，这就和编写用户层应用程序时，先对函数进行定义，函数的具体实现可以放在程序的后面部分一个道理。比如在上面的入口函数代码里面我们设置了 BulkUsb_DispatchReadWrite 这个 USB 读写函数，而该函数的功能实现代码并没有放在入口函数里面，而是放在了后面的程序中，其代码如下[40]：

```
NTSTATUS
BulkUsb_DispatchReadWrite(//USB 读写函数
    IN PDEVICE_OBJECT DeviceObject,//参数一，设备对象
    IN PIRP           Irp//参数而，输入输入请求号
    )
{
……
//决定读数据还是写数据
read = (irpStack->MajorFunction == IRP_MJ_READ) ? TRUE : FALSE;
……
//建立数据传输的块管道
UsbBuildInterruptOrBulkTransferRequest(
                                urb,
                                sizeof(struct _URB_BULK_OR_INTERRUPT_TRANSFER),
                                pipeInformation->PipeHandle,
                                NULL,
                                mdl,
                                stageLength,
                                urbFlags,
                                NULL);
……
//将传输请求命令转发给 USB 总线驱动程序
ntStatus=I0callDriver(deviceExtension_>Top0fStackDevice0bject,Irp);
……
}
```

在上面读写派遣函数代码中，程序首先判断了是否是读操作，在确定是读操作之后就调用函数 UsbBuildInterruptOrBulkTransferRequest 向 USB 外设发送读数据的请求包，之后就是等待 USB 固件程序的相应操作了，操作成功了就会把读取到的数据返回来。同理，写数据的实现方式以及其他派遣函数的实现都是类似的，我们也不再重复介绍了。

4.4 驱动程序的编译与安装

4.4.1 DDK 开放环境编译驱动程序

在把 USB 的驱动代码全部编写好了之后我们就可以对它进行编译，编译工具当然就是前面提到的 DDK 工具，但是和普通应用程序编译不一样的是它不能直接在 VC 这样的软件界面下点击编译就能完成驱动程序的编译，而是需要在 DOS 命令行执行编译，其过程类似于 Linux 下 GCC 编译软件的过程。我们需要在存放工程代码的目录下面放置两个文件：Makefile 和 Sources 文件。Makefile 文件我们直接从 DDK 工具里面拷贝一个过来就可以了，这个是要被 DDK 编译工具调用的，而且 DDK 帮助文档上明确表明我们不要去修改它；Sources 文件需要我们自己编写，它的主要功能是告诉编译工具哪些是我们需要编译的代码文件名称及路径，当然我只需要在里面添加.c 结尾的这类源文件，而不需要添加.h 结尾的头文件[42]。如图下所示为我们的 Sources 文件内容：

```
TARGETNAME=ezusb
TARGETTYPE=DRIVER
TARGETPATH=.\LIB
DRIVERTYPE=WDM
INCLUDES=$(BASEDIR)\inc; \
         $(BASEDIR)\src\usb\inc; \
         $(BASEDIR)\src\wdm\usb\inc; \
          ..\..\inc
C_DEFINES=$(C_DEFINES) /DDRIVER
TARGETLIBS=$(BASEDIR)\lib\*\free\usbd.lib
# to build this driver using the Windows 2000 DDK, uncomment the following line:
# TARGETLIBS=$(DDK_LIB_PATH)\usbd.lib
USE_MAPSYM=1
SOURCES= ezusbsys.rc \
         ezusbsys.c
```

接下来进入 DDK 开发环境，因为我们使用的操作系统是 Windows XP 版本的，就选择了 XP 版本的编译环境，而 XP 下又有 check 和 free 两个编译环境可以选择，check 版本是为了调试代码用的，free 版本是最后调试完成之后发布用的，这与普通应用程序的开发时的 Debug 与 Release 选项类似，为了可以调试，于是我们选择了 XP 环境下的 check 版本，程序打开了一个 DOS 命令行模式，命令行模式下我们用 DOS 命令进入我们存放代码的文件目录，因为我们的文件放在 F:\DDK2600\ezusbdrv\ezusbdrv 这个目录下面，所以我们输入 cd F:\DDK2600\ezusbdrv\ezusbdrv 之后就进入到了该目录下面，接下来我们在 DOS 命令行下输入 build –cz 命令等待片刻之后程序就编译完成，并最终生成了我们需要

的 ezusb.sys 这个驱动文件了，如图 4-7 所示。

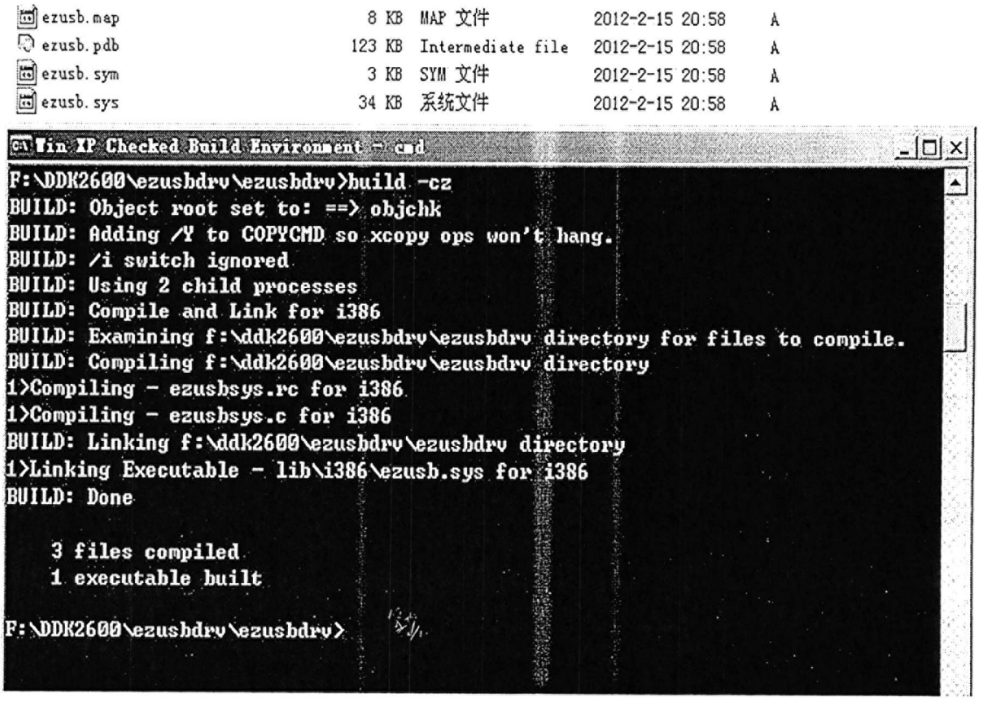

图 4-7　DDK 编译驱动程序

4.4.2　inf 文件的编写

在 4.1.2 节我们成功编译出了 ezusb.sys 这一驱动程序主文件，但是我们前面说过，WDM 模型的驱动程序需要配合 inf 文件才能正常安装到系统中。这一节我们就来详细介绍一个完整的 inf 文件是怎么编写的。

.inf 文件全称叫 Device Information File，翻译成中文叫做设备信息文件，这个文件是微软公司专门开发出来为硬件设备生产厂商发布他们的硬件驱动程序的。.inf 文件可以直接当做.txt 这类文本文件一样打开。打开文件之后我们会发现.inf 文件中有很多用中括号包起来的字符串，如[Version]、[Manufacturer]、[Strings]等等。其实从一个[Version]这类用中括号包起来的文字，到下一个以中括号包起来的文字这几行文字叫做一个节（section），就像我们作文写作中的一个自然段。inf 文件就是以节为段落从第一行开始一句一句往下执行的。而一个节（section）下面的文本内容格式一般如下：

entry = value [, value ……]

其中 entry 可以是一个指令、一个关键词也可以是一个文件名，value 是赋予 entry 的属性；entry 和 value 这两个内容又可以由 2 个“%”包起来的字符串所代替。其实这很像 C 语言里面的赋值与引用。

接下来我们对 inf 文件中的每个节（section）进行说明[26,42,43]：

一、[Version]节：

正常情况下一个完整的 inf 文件都会有[Version]这个节，它有点像 C 语言程序中的 main 函数，inf 文件是从[Version]节开始往下执行的。如表 4-1 所示展示了[Version]节的常用关键字和它们的属性。

表 4-1 [Version]节的部分值

Entry	Value
Signature	$Windows NT$、$Windows 95$、$Chicago$中任一个，代表 inf 文件所支持的操作系统版本
Class	驱动程序的整个类族的名称，例如 USB、PCI 等等
ClassGuid	一个类的独一无二的 GUID，如 USB 类 GUID 为{36FC9E60-C465-11CF-8056-444553540000}
Provider	inf 文件提供人的名称，如 Cypress
LayoutFile	只有在系统支持的 inf 文件中才使用该项，OEM 支持的 inf 文件使用 SourceDisksNames 和 SourceDisksFiles 这两项
DriverVer	包含可选的版本信息 ，如 08/05/1999,5.00.2064

二、[Manufacturers]节：

这个节列出了可以使用这个 inf 文件进行安装的硬件设备和驱动程序，其格式如下：

manufacturer=model

manufacturer 列出被安装的硬件设备制造者的 inf 文件的唯一名字。model 提供另外一个节（section）的指针。

三、[Models]节：

[Manufacturers]部分的每一个 model 的具体实现，该节下面的文本格式如下：

device-description=install-section-name,hw-id[,compatible-id...]

例如：%USB\VID_045E&PID_930A.DeviceDesc%=BULKUSB.Dev, USB\VID_045E&PID_930A。其中 BULKUSB.Dev 又是新的节，在下面[DDInstall]中介绍。

四、[DDInstall]节：

它的名字唯一的指定从[Models]部分列出的每个硬件设备制造者的每个 model。该节的关键字和属性如表 4-2。

表 4-2 [DDInstall]节的部分值

Entry	Value
DriverVer	必须的有，包含可选择的版本号，
CopyFiles	指定的需要复制的文件列表的部分或者一个单独的有“@”为前缀的节
AddReg	必须有，另一个包含需要注册表信息的项
Include	指向其它 inf 文件名的指针列表
Needs	Include 节中的子节，列出了它的 inf 文件中需要的内容
DelFiles	指定要删除的文件列表的项
RenFiles	指定要重新命名的文件列表的项
DelReg	指定要删除的系统注册表信息的项
ProfileItems	指定要更改系统开始菜单的项

五、[CopyFiles]节：

其形式为：destination-filename[,source-filename,temp-filename,flag]

destination-filename 是要被复制的文件名，如果它与源文件名不相同 source-filename 必须被指定；temp-filename 在 Windows 98 之后的环境下没用；flag 值指定目标文件的部署，如是否替换已经存在的同名文件等。因为[CopyFiles]节的语法中不含源文件的路径选项，所以，必须要在 inf 文件中添加[SourceDisksNames]与[SourceDisksFiles]这两个节。源文件由[CopyFiles]指定，目标文件由[DestinationDirs]指定。

六、[AddReg]节：

提供在系统注册表中添加或者修改的节。其形式为：

reg-root[,subkey,value-name,flags,value]

七、[SourceDisksNames]节：

如果 inf 文件中用到的驱动程序文件存放在多个磁盘上时就必须包含[SourceDisksNames]节。这个部分包含文件分布的每个磁盘的入口。其形式为：

diskid=disk-description[,tagfile,unused,path]

八、[SourceDisksFiles]节：

其中需要列出在安装驱动程序过程当中使用的文件名，每个文件都要有相应的节。其形式为：filename=diskid[,subdir,size]

九、[DestinationDirs]节：

提供在复制、删除、重命名文件时的目标文件路径。其形式为：file-list-section=dirid[,subdir]或者 DefaultDestDir=dirid[,subdir]

十、[DDInstall.Services]节：

告诉系统此次安装的这个程序属于驱动程序。其形式为：

AddService=ServiceName,[flags],service-install-section[,eventlog-install-section]

十一、[ServiceInstall]节：

在[DDInstall.Services]节中的 AddService 部分被指定，用于控制和安装驱动程序到系统服务控制管理器。

十二、[Strings]节：

定义并列出前面各节中用到的字符串，通常把那些频繁使用的字符串定义为变量以方便使用。

上面提到的这些节（Section）都是一些比较常见的，还有一些其他的节我们不再一列举，关于它们的介绍可以在微软 DDK 帮助文档中找到。如下所示代码就是与我们在 4.1.3 节配套的 inf 文件的部分代码：

```
[Version]
Signature="$CHICAGO$"    ;驱动程序适合安装的环境，CHICAGO 指代 WIN98 之后的操作系统
Class=USB                ;驱动程序属于 USB 设备类的
provider=%Cypress%       ;提供商名字
LayoutFile=layout.inf
[Manufacturer]
%Cypress%=Cypress        ;制造商名字
[Cypress]
; This is the VID/PID for the EZ-USB development board.  This device
; is bound to a version of the general purpose driver that will
; automatically download the Keil 8051 monitor to external RAM.
; Do not use this VID/PID for your own device or the monitor will
; wipe out your firmware.
; EZ-USB FX2             ;下面的一串是要与固件程序中匹配的 VID 与 PID 号
%USB\VID_04B4&PID_8613.DeviceDesc%=EZUSB.Dev, USB\VID_04B4&PID_8613
%USB\VID_04B4&PID_1002.DeviceDesc%=EZUSB.Dev, USB\VID_04B4&PID_1002
[PreCopySection]
HKR,,NoSetupUI,,1
[DestinationDirs]
EZUSB.Files.Ext = 10,System32\Drivers          ; 文件将要被拷贝到 System32\Drivers 目录下面
EZUSB.Files.Inf = 10,INF
EZUSBDEV.Files.Ext = 10,System32\Drivers
EZUSBDEV.Files.Inf = 10,INF
[EZUSB.Dev]
```

```
CopyFiles=EZUSB.Files.Ext, EZUSB.Files.Inf   ;需要拷贝的文件名
AddReg=EZUSB.AddReg
[EZUSB.Dev.NT]
; copyfiles commented out for Win2K to avoid user intervention during install
; CopyFiles=EZUSB.Files.Ext, EZUSB.Files.Inf
AddReg=EZUSB.AddReg
[EZUSB.Dev.NT.Services]
Addservice = EZUSB, 0x00000002, EZUSB.AddService   ;需要添加的服务项
[EZUSB.AddService]        ;服务项中具体需要添加的内容
DisplayName      = %EZUSB.SvcDesc%
ServiceType      = 1                    ; SERVICE_KERNEL_DRIVER;代表内核驱动类型
StartType        = 2                    ; SERVICE_AUTO_START;代表自动安装类型
ErrorControl     = 1                    ; SERVICE_ERROR_NORMAL;代表错误类型
ServiceBinary    = %10%\System32\Drivers\ezusb.sys ;服务项中显示的应用程序路径及名称
LoadOrderGroup = Base          ;本服务处在服务组中的次序
[EZUSB.AddReg]       ;注册表中需要添加的内容
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,ezusb.sys
[EZUSB.Files.Ext]
ezusb.sys      ;驱动程序名称
[EZUSB.Files.Inf]
ezusbw2k.Inf       ;inf 文件名称
[EZUSBDEV.Dev]
CopyFiles=EZUSBDEV.Files.Ext, EZUSBDEV.Files.Inf        ;需要拷贝的文件名
AddReg=EZUSBDEV.AddReg
[EZUSBDEV.Dev.NT]
; copyfiles commented out for Win2K to avoid user intervention during install
; CopyFiles=EZUSBDEV.Files.Ext, EZUSBDEV.Files.Inf
AddReg=EZUSBDEV.AddReg
[EZUSBDEV.Dev.NT.Services]
Addservice = EZUSBDEV, 0x00000002, EZUSBDEV.AddService        ;需要添加的服务项
[EZUSBDEV.AddService]              ;服务项中具体需要添加的内容
DisplayName      = %EZUSBDEV.SvcDesc%
ServiceType      = 1                    ; SERVICE_KERNEL_DRIVER;代表内核驱动类型
StartType        = 2                    ; SERVICE_AUTO_START;代表自动安装类型
ErrorControl     = 1                    ; SERVICE_ERROR_NORMAL;错误类型
ServiceBinary    =  %10%\System32\Drivers\ezmon.sys;服务项中显示的应用程序路径及名称
LoadOrderGroup = Base      ;本服务处在服务组中的次序
[EZUSBDEV.AddReg]       ;注册表中需要添加的内容
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,ezmon.sys
[EZUSBDEV.Files.Ext]
ezmon.sys
[EZUSBDEV.Files.Inf]
Ezusbw2k.Inf
;-----------------------------------------------------------------;
[Strings]       ;自定义的字符串
```

```
Cypress="Cypress Semiconductor"
USB\VID_04B4&PID_1002.DeviceDesc="Cypress EZ-USB Sample Device"
USB\VID_04B4&PID_8613.DeviceDesc="Cypress EZ-USB FX2 (68613) - EEPROM missing"
……
```

其中我们把一些重要的地方进行了解释，在分号后面的中文字体就是我们添加的解释。当然为了简便起见，我们其实是在 Cypress 公司提供的 inf 文件上直接进行修改的。从这里我们看到一行 VID 与 PID 分别为 04B4 与 1004，这正好与我们在前面第 4 章中图 4-2 中显示的 VID 与 PID 一致，这也说明了在前面我们的驱动程序正确安装了。

4.4.3 驱动程序的安装

有了.sys 格式的可执行驱动文件以及.inf 格式的安装文件就可以进行驱动程序的安装了。首先将已经下载好固件程序的 USB 设备插到电脑主机的 USB 接口上，如果是在没有安装过驱动程序的电脑上我们可以看到系统会自动弹出未知硬件，并询问我们是否需要安装驱动程序，我们选择否之后，打开设备管理器在通用管理总线控制器下面可以看到 unknow device，然后点更新驱动程序之后选择从指定地方安装驱动程序，然后把我们存放驱动程序已经 inf 文件的目录添加到上面，再点击下一步就能进行安装了，安装完了之后就能显示出我们设备的名字了，名字就是我们在 Strings 节中自己定义的，如过 VID/PID 为 04b4 和 1002 则显示为 Cypress EZ-USB Sample Device，如图 4-8 所示。

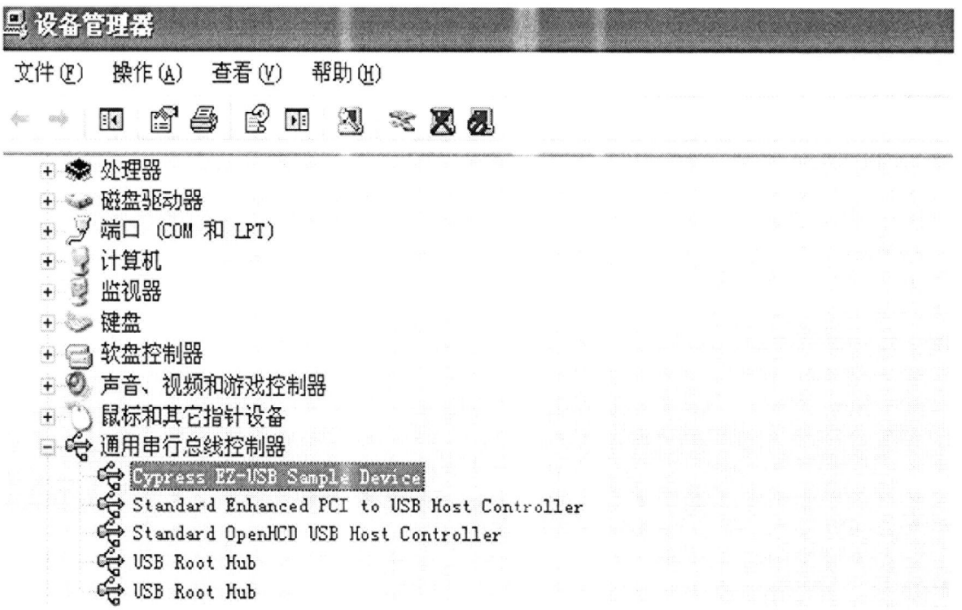

图 4-8 设备管理器中识别的 USB 设备

至于用第三方驱动开发工具开发USB驱动程序本文不多做介绍了，其实从根本上它还是和DDK开发方式一样的，只不过第三方工具为我们做好了图形界面模式，我们只需要简单的点击或者选择一些选项第三方工具就会自动替我们完成很多原本我们需要手动编写的驱动代码了，这样虽然会大大提高我们的驱动程序开发速度，不过那样一来我们就无法知道驱动程序的编写过程，而且这样的程序要添加我们自己的功能并修改成我们自己最终想要的程序也比较麻烦。还有最致命的一点就是大部分这样的第三方开发软件都是要收费的，因此，在可以使用DDK开发套件的情况下没必要使用第三方驱动开发软件。

经过上面的设计，我们的电脑已经能够识别USB设备了，然而仅仅只是能够识别，我们却还无法控制USB设备，要控制USB设备就需要在主机端编写软件来控制，从下一章开始我们就正式开始上位机应用程序的编写了。

第5章 基于DirectShow技术与BDA驱动的数字电视播放软件设计

经过前面的设计我们已经能够读写USB缓冲区中的数据，也就意味着我们的电脑主机已经能够接收一路或几路复用的MPEG2-TS格式的数字电视传输流数据。接下来有两种方式处理接收到的数据流：一种方式是直接把数据流保存到本地磁盘，保存完了之后就可以用普通的视频播放软件如Windows Media Player播放电视节目；另外一种方式是对接收到的数据进行处理之后直接播放电视节目。第一种方式很简单，其实就是一个数据采集器，但是无法做到实时播放电视节目的，这显然不符合目前的主流电视产品。因此，我们要采用第二种方式，但是如何实现这样的设计呢，这就是本章将要介绍的内容。

5.1 DirectShow技术简介

5.1.1 DirectX技术

DirectX是微软针对其Windows操作系统设计的一套专门用于开发图像、音视频、游戏等应用的编程技术。DirectX是一种与硬件设备无关的技术，即任何硬件设备都可以用该编程技术实现高性能输出。但是这是如何做到的呢？其实这就需要硬件设备生产商们按照微软DirectX的技术标准设计他们的硬件设备，主要是硬件设备在主机端的驱动程序，把驱动程序的接口设计成能DirectX要求的接口标准，这样软件开发人员就可以不用理会硬件的具体细节，直接应用DirectX的编程技术编写各种应用端程序。由于微软的影响力，DirectX技术技术已经成为一种Windows操作系统下多媒体开发的标准。

DirectX意思就是直接开发某某东西，X就代表了它支持的各种应用。我们从DirectX 9.0C这个开发包上来看它支持如下应用[44,45]：

1.DirectX Graphics：把以前版本的DirectDraw和Direct3D应用融合在一起，即同时实现了对2D与3D技术绘图接口，可以直接操作显卡和内存，它是开发DirectX游戏的基础。

2.Microsoft DirectInput：支持各种各样输入设备的技术，同时完全支持对设备的强制反馈技术。

3.Microsoft DirectPlay：提供了对网络游戏的各种支持。

4. Microsoft DirectSound：提供了对高性能音频播放、音频采集等应用的支持。

5. Microsoft DirectMusic：提供了对各种音乐格式的支持。

6. Microsoft DirectShow：提供了对多媒体数据流的采集与回放功能。

7. DirectSetup：提供了对 DirectX 组件的安装支持的功能。

8. DirectX Media Objects：提供了对各种音视频数据流的简单处理。

可以看到第 6 项 Microsoft DirectShow 技术提供了对多媒体数据流的采集与回放等功能，因此我们可以采用 DirectX 技术下的 DirectShow 方式来解决我们前面提出的问题。

5.1.2 COM 技术[44,46]

前面提到的 DirectX 技术采用公共的接口，其实公共的接口指的就是 COM 技术（Components Object Model，组件对象模型 ），DirectShow 也同样是基于 COM 技术标准的。COM 不是什么软件，也不是什么具体的东西，它是一种技术标准。新的技术总是为了给人类带来更多的方便，COM 技术标准也一样，以前不同环境的操作系统之间编写的软件无法通用，这使得开发人员不得不针对不同的环境编写软件，这浪费了人力资源，COM 技术就是为了解决这一问题才出现的。基于 COM 技术的组件其实是一段二进制形式的可执行文件，它可以用任何语言编写，而且任何支持二进制形式的操作系统都能使用该组件，这就是它的强大之处。

以 C++开发 COM 组件为例，其实 COM 组件就相当于一个 C++的类（class），它的接口都是纯虚类。下面的 C++代码可以简单明了的展示什么是 COM 组件：

```
class IUinterface
{
Public:
Virtual Func1(……)      = 0;
Virtual Func2(……)      = 0;
……
};
Class MyComponet : public IUinterface
{
Public:
Virtual Func1(……){……}
Virtual Func2(……){……}
……
};
```

其中的 IUinterface 就是接口，MyComponet 就是前面所说的 COM 组件。

5.1.3 DirectShow SDK 简介[44,45]

微软开发 DirectShow 技术是为了给软件开发人员带来软件编写上的便利，特别是涉及到数据传输，音频采集，视频播放等软件开发的软件编程人员。因为微软为这些方面的应用编写了很多基础的类，开发人员不需要再花大量时间去自己编写这样的基础类，只需要在他们的基础类（Base Classes）上继承下该类再添加自己需要的特性就能很轻松的编写出符合自己需要的软件。为此，开发人员只要去微软官方网站上下载包含有 DirectShow 的 DirectX 软件开发工具包（DirectX SDK）进行安装，然后对安装包里面微软提供的基础类进行编译并生成.lib 格式的库文件之后，并把生成的库文件和 Dshow.h 这个头文件添加到你自己创建的工程文件里面之后就能使用那些基础类来编写相应的程序了。图 5-1 为微软的 DirectShow 技术应用框图[44]。

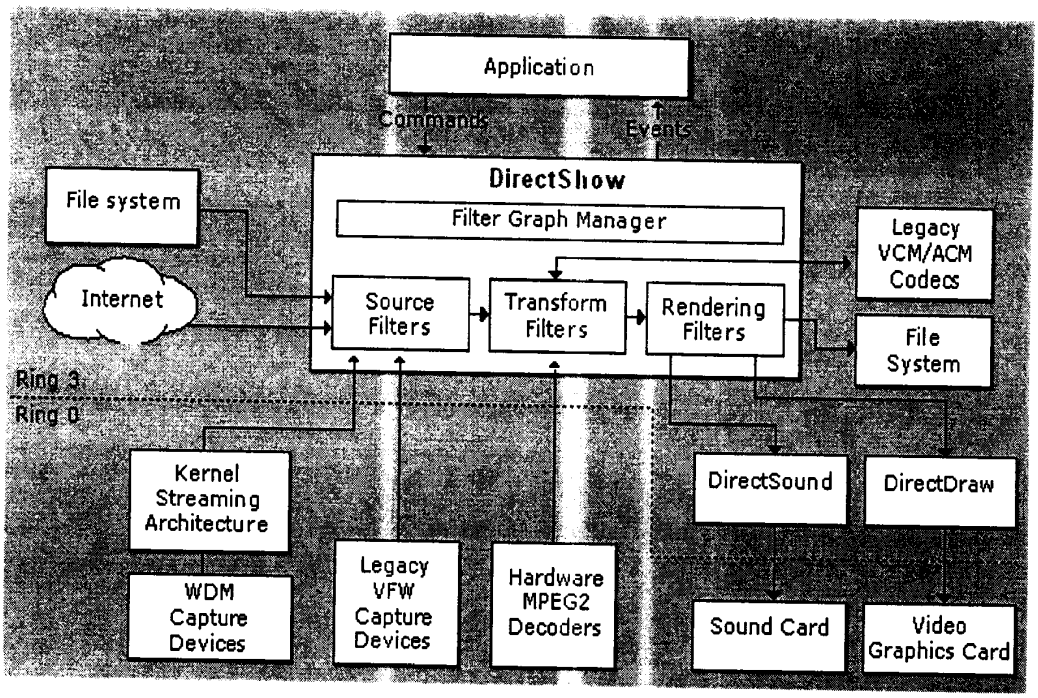

图 5-1 DirectShow 系统应用框图

由图中我们可以看到 DirectShow 系统工作在 Ring 3 层，即用户层面，它可以读取文件系统、网络、WDM 驱动的硬件设备以及 VFW 驱动的硬件设备的数据，最后经过一系列处理之后通过声卡，显示卡把音视频数据展现给我们欣赏。DirectShow 系统使用模块架构，每个模块处理不同的功能，每个处理阶段的模块由一个叫做过滤器（Filter）的 COM

组件完成。如上图的 Source Filters、Transform Filters 和 Rendering Filters 这 3 个就是典型的 3 类过滤器。Source Filters 叫做源过滤器，主要完成从系统中或者外部硬件设备中读取数据，然后将读取到的数据传给下一个过滤器；Transform Filters 叫做转换过滤器，主要任务是把从前端传递下来的数据进行分解或者合成、编码或者解码，然后将处理过的数据再传递给下面的过滤；Rendering Filters 叫做渲染过滤器，主要把经过前端处理的数据传输到声卡、显卡等硬件播放出来。上面提到的这些过滤器都是由一个叫做 Filter Graph Manager 的组件控制的，它是 DirectShow 系统中最核心的组件。

5.2 内核流架构驱动程序开发

5.2.1 内核流架构体系简介[42,44,45]

前面在介绍 DirectShow 技术应用系统框图的时候我们提到 DirectShow 可以获取 WDM 驱动编写的硬件设备的数据，而我们的 USB 数字电视采集设备正好也是基于 WDM 驱动程序的，因此事实证明我们可以采用 DirectShow 技术来获取我们的 USB 数据。但是要想使我们的设备支持 DirectShow 技术就必须要符合他们的接口标准，而我们的 USB 驱动程序是无法直接满足 DirectShow 技术接口标准的，为了让我们的驱动程序符合 DirectShow 技术接口，我们有两种方法可以实现：方法一，修改我们的 USB 驱动程序使它满足那样的接口；方法二，在 USB 驱动和 DirectShow 系统之间添加中间转换模块使得经过转换后的接口符合 DirectShow 技术。从程序开发的角度来考虑方法二是最佳的，微软也正是基于这样的考虑才开发了内核层过滤器。

硬件生成商只要按照 KSA(Kernal Stream Architecture，内核流架构)标准编写驱动程序，该驱动程序就会自动作为内核层过滤器被暴露给处在用户层的 DirectShow 系统，DirectShow 系统可以像普通用户层的过滤器一样读取与控制该内核过滤器。因此，只要硬件生成厂商提供基于 KSA 体系架构的驱动程序，后面的处理其实完全可以由 DirectShow 系统完成了，而后面 DirectShow 系统的编写者也完全不需要知道前面硬件生成商的具体开发细节，因为 KSA 体系架构是微软制定的标准，只要按照标准中规定的方式就能正确读取并控制硬件设备了。

KSA 架构的驱动程序实现方式与 COM 组件的类似，不过它本身是处在内核层的，它向处在 DirectShow 系统提供了一组以文件对象形式存在的接口，每一种文件对象可以实现硬件设备的一项功能，DirectShow 系统通过向文件对象发送输入输出请求命令以达到控制设备的目的。KSA 架构提供的文件对象主要有 7 类：过滤器，Pins(引脚)，Properties

（属性），Events（事件），Methods（方法），clocks（时钟）和 Allocators（分配器）。过滤器与 DirectShow 系统中的一样，代表一项设备功能，如采集、数据分离等。Pins 在 DirectShow 系统中也有，代表一个过滤器的输入输出通道，比如 DirectShow 系统可以把源过滤器的输出 Pin 连接到转化过滤器的输入 Pin 上，这样逻辑上就实现了 2 个过滤器的连接，也正因为这样才能使 DirectShow 系统发现内核层的过滤器，才能建立起与用户层的过滤器的连接。Properties、Events 和 Methods 这也与 COM 组件类似，可以使得调用该组件的后端软件设置组件的属性，使用 Events 进行事件通知，执行内核定义的 Methods 函数功能。clocks 可以使得设备把自己的时钟传递给调用它的软件以进行时间同步等处理。Allocators 提供了对内存的分配与释放等管理功能，它实际上是过滤器与过滤器之间传递数据的根本所在。

5.2.2 AVStream 类驱动程序简介[42,47]

AVStream（音视频流）架构驱动程序是一种新型的内核流式驱动程序，是专门针对音频视频类设备设计的类（Class）驱动程序。AVStream 类驱动程序是微软建立在 Windows XP 与 Windows Server 2003 操作系统之上的，不过新的系统 Windows Vista 以及 Windows 7 都支持该类驱动程序。既然微软为这个驱动程序起名为类驱动程序，其实它就与面向对象里面的类的概念是一致的。其实微软已经帮你实现了这个驱动的框架，而框架里面的一些属性以及一些函数的具体功能实现则需要驱动程序编写者自己去完成，这部分由驱动程序编写者完成的部分驱动程序微软叫它 AVStream mini 驱动。因此，AVStream 类驱动程序的出现对专门开发音视频方面的硬件生成商来说，在驱动程序方面的开发难度降低了不少。如图 5-2 所示为 AVStream 类驱动程序的体系架构。

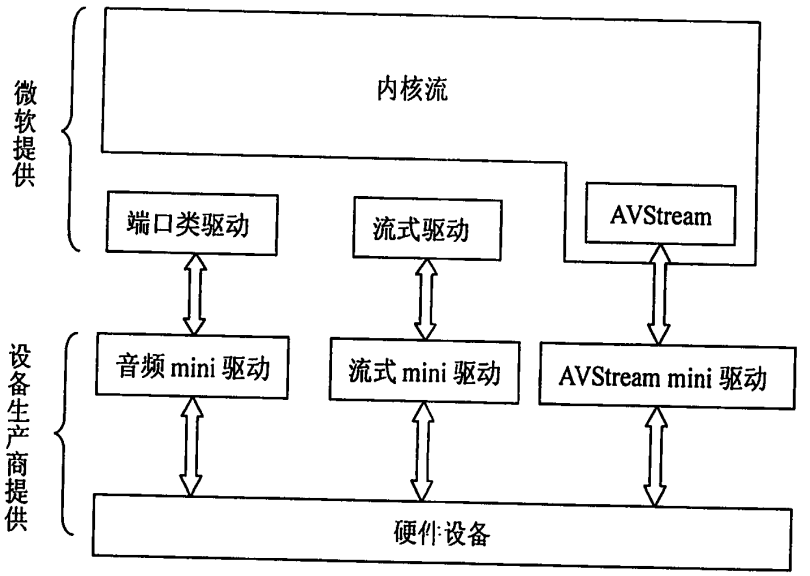

图 5-2 AVStream 类驱动架构

5.2.3 BDA mini 驱动程序简介[42]

BDA（Broadcast Driver Architecture，广播驱动架构）驱动是专门针对广播数字电视产品的 AVStream mini 驱动程序的一种。BDA 架构体系的 mini 驱动程序主要完成以下几个方面的作用：控制 Tuner 芯片调谐数字广播信号的频率，控制解调芯片对数字信号进行解调，捕获数字信号帧并将接收到的数字信号传递给下一级过滤器以便进行数据流的后续处理。

5.3 基于微软 DDK 驱动开发套件开发 BDA mini 驱动程序

5.3.1 BDA mini 驱动程序设计概述

一般在软件或者硬件开发中，为了使得客户能够尽快适应它们的的开发思路，软硬件提供者都会提供相应的说明文档，框架程序或者例子代码，事实上微软在设计 AVStream mini 驱动程序的时候已经为这类驱动程序编写了很多框架代码以及例子代码。前面我们也说过微软的开发思路是把硬件和软件的开发尽量分开，使得不管硬件设备如何改变，软件的整体框架都不需要改变，需要改变的只是软件里面的具体实现细节，其实这与我们在开发 USB 固件程序时使用 Cypress 公司的提供固件程序框架是类似的，因此，我们可以直接在微软的框架代码上进行修改或者添加自己的实现代码实现我们自己需要的应用。微软提供的框架代码主要是纯软件层面的设计框架，我们更多的是需要在驱动中实现我们对硬件设备的操作，因此，我们的主要工作其实是把与硬件相关的操作添加到微软提供的

BDA mini 驱动框架里面。图 5-3 为微软的 BDA mini 驱动框架的系统结构框图[42]。

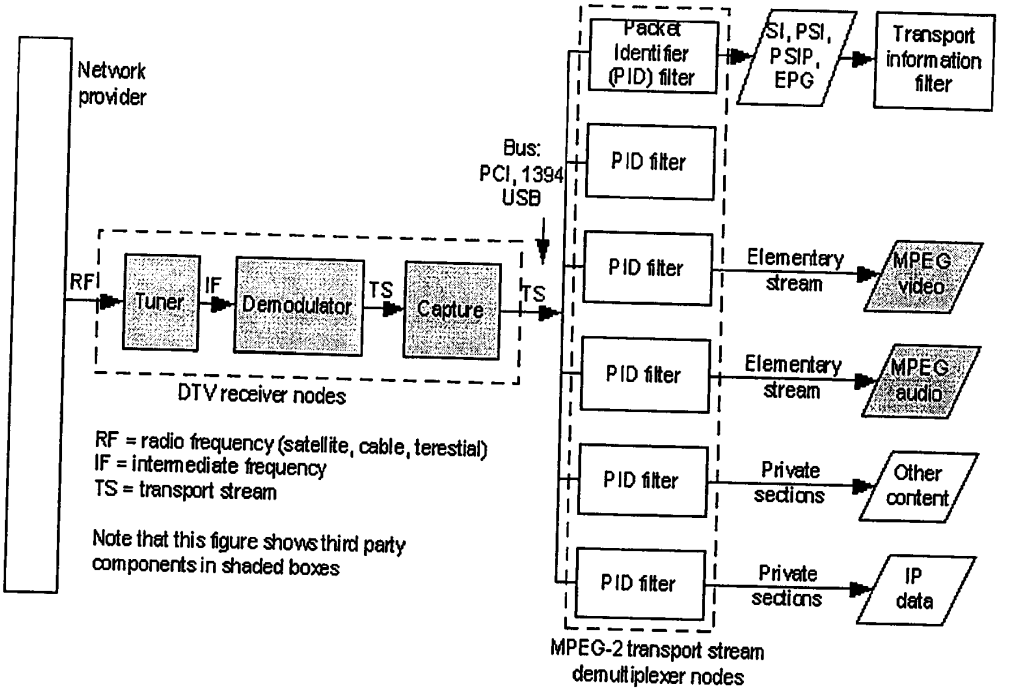

图 5-3 BDA mini 驱动框架系统结构框图

图中左半部分就是 BDA mini 驱动程序所要做的部分，包括 Network provider 部分、Tuner 部分、Demodulator 部分以及 Capture 部分。其中 Network provider 部分主要是实现各种数字电视传输模式的选择，在 DDK3790 版本中可以是 ATSC、DVB-C、DVB-S 与 DVB-T 之中的一个，在更新版本的 DDK 中可能添加了更多的模式。其中的 Tuner 和 Demodulator 这两个部分根据我们前面硬件设备的设计不同可以和起来做成一个部分也可以像图中一样分开成 2 个部分。比如我们硬件设计时采用的 Tuner 芯片是集成了 Demodulator 解调功能的我们就可以在软件中把它设计成一个模块，如果采用的分开的 Tuner 芯片与 Demodulator 解调芯片则可以像图示中一样设计，这样也给硬件开发人员带来了极大的便利，使得硬件开发人员可以按照自己的设计思路来设计硬件。图中 Tuner 是这个驱动模块主要是用于控制调频部分的程序，其实是通过 IIC 接口控制 Tuner 芯片的寄存器达到调频的目的；而 Demodulator 驱动模块则是通过 IIC 接口控制解调器芯片解调从 Tuner 芯片传输过来的数字信号。微软把硬件通过 mini 驱动程序展现给后端 DirectShow 系统，使得后端 DirectShow 系统的开发人员能够直观地看到硬件设备的设计框架。Capture 驱动模块用于对数据的采集并且把数据最后提交给后端 DirectShow 系统。

5.3.2 BDA mini 驱动程序之 Tuner 与 Demodulator 模块设计

在微软 DDK 安装目录下面的源文件目录下面我们可以看到有很多微软提供的驱动程序源代码，在 WDM 文件夹下面有 1394、acpi、audio、bda 等这些驱动程序的例子。进入 bda 这个文件夹，我们可以发现 generictuner 与 capturesample 这两个文件夹，他们就分别代表了图 5-3 中的 tuner/demodulator 和 capture 这两部分的框架程序代码。由于 tuner 与 demodulator 实际上都是对 USB 芯片的控制，因此这里把这两个模块写在了一个框架程序里面了。

Generictuner 目录下包含的文件如表 5-1 所示[42]：

表 5-1 generictuner 工程文件简介

Bdadebug.h	头文件，用于调试驱动程序的一些宏定义
Bdaguid.c	工程文件，定义驱动程序 GUID 号的一个
BDATuner.h	头文件，为过滤器、设备及过滤器输入输出引脚定义类结构体
BDATuner.inf	inf 文件，用于配合安装此驱动程序
BDATuner.rc	资源文件，包含了 BDA 驱动程序的版本信息
Common.h	头文件，工程中需要用到的公共头文件都包含在里面了
Device.cpp	BDA 驱动程序的主文件，包含了驱动程序的入口函数
Filter.cpp	工程文件，过滤器类的具体实现代码
Inpin.cpp	工程文件，输入引脚类的具体实现代码
Makefile	Makefile 文件，编译驱动程序用的，不建议修改
ObjDesc.cpp	工程文件，定义了自动控制表、分发表以及类模板
Outpin.cpp	工程文件，输出引脚类的具体实现代码
Sources	编译驱动程序时必需的资源文件
Splmedia.h	头文件，定义连接节点的属性、方法及事件等结构
Wdmdebug.h	头文件，用于调试驱动程序的一些宏定义

微软提供的这个程序框架可以直接通过 DDK 环境编译，但是其实它无法使用，因为它根本没有实现数据流的传输。我们需要在这个工程文件的基础上添加我们的 USB 设备读写功能以及 IIC 读写控制控制功能，才能真正的从硬件中获取数据流。下面我们主要介绍我们自己对这个驱动程序所做的修改部分，其它部分的代码我们只是简单的提到，那部分代码的具体细节可以在微软的 DDK 工程下面找到。

首先为了能够读取 USB 设备的数据我们要先找到 USB 设备，我们在 Device.cpp 这个主工程文件下面添加了对 USB 设备的获取，在内核层获取 USB 设备的方法是通过 IoGetDeviceInterfaces 这个函数获取到 USB 设备的 GUID 号，这个 GUID 号就是前面我们

在 USB 固件程序编程中 inf 文件中使用的。我们需要在设备启用的时候获取 USB 设备，因此我们在 CDevice::Start()这个派遣函数中实现对 USB 设备的获取，部分代码如下[40]：

```
……
static GUID CYUSB_GUID = {0xae18aa60, 0x7f6a, 0x11d4, 0x97, 0xdd, 0x0, 0x1, 0x2, 0x29, 0xb9, 0x59};
……
STDMETHODIMP_(NTSTATUS)
CDevice::
Start(
    IN PKSDEVICE                pKSDevice,
    IN PIRP                     pIrp,
    IN PCM_RESOURCE_LIST        pTranslatedResourceList OPTIONAL,
    IN PCM_RESOURCE_LIST        pUntranslatedResourceList OPTIONAL
    )
{
    NTSTATUS                Status = STATUS_SUCCESS;
    CDevice *               pDevice;
    PKSFILTERFACTORY        pKSFilterFactory = NULL;
    #if DBG
        DebugPrintInit("BDA tuner driver checked");
    #else
        DebugPrintInit("BDA tuner driver free");
    #endif
    DebugPrint("BDA driver begin...");
DebugPrint("Device Start");
    //根据 GUID 号获取并打开 USB 设备
    PWSTR    pSymbolLink = NULL;
    Status = IoGetDeviceInterfaces(
            &CYUSB_GUID,
            NULL,
            DEVICE_INTERFACE_INCLUDE_NONACTIVE,
            &pSymbolLink
            );
    UNICODE_STRING DeviceName;
    RtlInitUnicodeString (&DeviceName, pSymbolLink);
    Status = IoGetDeviceObjectPointer(&DeviceName, FILE_ALL_ACCESS, &ipFileObject,
&ipDeviceObject);
DebugPrint("usb device[%d]ipFileObject[%d]ipDeviceObject[%d]",Status,ipFileObject,ipDeviceObject);
……
```

在获取并打开 USB 设备之后我们便可以读写 USB 芯片，对 USB 设备进行数据读写。这里对 USB 的读写我们可以在 Device.cpp 这个文件中添加自己的 USB 读写函数，如 WriteUSB 函数，我们可以通过 IoBuildDeviceIoControlRequest 命令函数向 USB 设备发送

IRP 请求以达到对 USB 设备的写控制。同样我们可以实现读 USB 函数，以及 IIC 读写控制函数功能。如下所示为读写 USB 设备函数的部分代码[40]：

```
//实现 USB 端口读写功能函数
unsigned char ControlWrite(unsigned char *buf, LONG len)
{
    ......
//通过 IoBuildDeviceIoControlRequest 命令函数向 USB 设备发送写命令
        irp = IoBuildDeviceIoControlRequest(IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER,
                                ipDeviceObject,
                                (PVOID)buffer,
                                iXmitBufSize,
                                (PVOID)buffer,
                                iXmitBufSize,
                                FALSE,
                                NULL,
                                &ioStatus);
            Status = IoCallDriver(ipDeviceObject, irp);
            if(Status == STATUS_SUCCESS)
                return 1;
    ......
}
```

有了 USB 设备的读写函数也有了 IIC 总线的读写函数，我们就可以在 inpin.cpp 中对 Tuner 频率选择的框架代码中添加通过 IIC 总线控制 Tuner 芯片寄存器的具体实现代码了。另外，框架代码已经实现了频率调谐的 Properties（属性）和 Fuction（方法）相关代码的，因此，用户层应用程序可以直接对属性和方法进行调用，我们将在后面 5.4.1 章节介绍如何设置。考虑文章篇幅的问题，其他还有很多函数功能的实现我们不再一一列出。编写好程序之后通过与第 4 章中介绍的一样的方式就可以编译出.sys 格式的驱动程序了。我们通过 DDK 环境最后编译生成了 BDATuner.sys 这个 mini 驱动程序，配合工程目录的 BDATuner.inf 这个安装文件就能安装驱动程序了。

5.3.3 BDA mini 驱动程序之 Capture 模块设计

CaptureSample 目录包含的文件如表 5-2 所示[42]：

表 5-2 Capture 驱动模块包含的文件

BDASWRcv.inf	inf 文件，用于驱动调试，模拟硬件安装
device.h	头文件，Device.cpp 中用到的函数在这里定义

BDACap.h	头文件，工程文件中最主要的头文件
BDACap.inf	inf 文件，用于配合安装此驱动程序
BDACap.rc	资源文件，包含了 BDA 驱动程序的版本信息
filter.h	头文件，Filter.cpp 中用到的函数在这里定义
Device.cpp	BDA 驱动程序的主文件，包含了驱动程序的入口函数
Filter.cpp	工程文件，过滤器类的具体实现代码
capture.cpp	工程文件，数据捕获的具体实现代码在该文件中
Makefile	Makefile 文件，编译驱动程序用的，不建议修改
hwsim.cpp	工程文件，模拟硬件实现数据分发等功能
TStream.cpp	工程文件，模拟 TS 数据传输
Sources	编译驱动程序时必需的资源文件
capture.h	头文件，capture.cpp 中用到的函数在这里定义
hwsim.h	头文件，hwsim.cpp 中的函数在这里定义
TStream.h	头文件，TStream.cpp 中用到的函数在这里定义

hwsim.cpp 文件是 CaptureSample 这个工程文件中实现数据传输的主要文件，我们需要修改的部分也主要集中在这里，我们需要在这里获取 USB 设备并从 USB 的缓冲区中读取 TS 流数据，然后把数据传输给后面的过滤器。在这里我们创建了一个内核线程不停地从 USB 端口中读取数据，在创建线程之前我们要先打开 USB 设备，打开 USB 设备之后就能在线程处理函数中不停地读取 USB 端口中的数据了。部分代码如下[42]：

```
//从 USB 设备端口中读取 MPEG2-TS 数据流函数
void GatherTsSystemThread(IN PVOID Context)
{
……
if( NT_SUCCESS(Status) )
{
    //通过符号链接方式打开 USB 设备
    PDEVICE_OBJECT ipDeviceObject=NULL;
    PFILE_OBJECT ipFileObject=NULL;
    //                    WCHAR                DeviceBuffer[]                =
L"\\??\\usb#vid_04b4&pid_1003#5&247a1473&0&3#{ae18aa60-7f6a-11d4-97dd-00010229b959}";
    UNICODE_STRING DeviceName;
    RtlInitUnicodeString (&DeviceName, pSymbolLink);
    Status    =    IoGetDeviceObjectPointer(&DeviceName,    FILE_ALL_ACCESS,    &ipFileObject,
&ipDeviceObject);
……
if( ipFileObject!=NULL && ipDeviceObject!=NULL )
    {
        PIRP    irp;
        IO_STATUS_BLOCK        ioStatus[4];
```

```
        //设置 USB 数据传输的大小
        irp = IoBuildDeviceIoControlRequest(IOCTL_ADAPT_SET_TRANSFER_SIZE,
            ipDeviceObject,
            (PVOID)SetTransferSizeBuf,
            5,
            (PVOID)SetTransferSizeBuf,
            5,
            FALSE,
            NULL,
            &ioStatus[0]);
        Status = IoCallDriver(ipDeviceObject, irp);
.....
        //分配内存
        // Allocate the arrays needed for queueing
        for (i=0; i< QueueSize; i++){
            buffers[i] = (PUCHAR)ExAllocatePool(NonPagedPool,iXmitBufSize);
            KeInitializeEvent(&event[i], NotificationEvent, false);
        }
.....
    //下 USB 发送命令开始数据传输
            irp = IoBuildDeviceIoControlRequest(IOCTL_ADAPT_SEND_NON_EP0_TRANSFER,
                    ipDeviceObject,
                    (PVOID)buffers[i],
                    iXmitBufSize,
                    (PVOID)buffers[i],
                    iXmitBufSize,
                    FALSE,
                    &event[i],
                    &ioStatus[i]);
            Status = IoCallDriver(ipDeviceObject, irp);
.....
```

上面是线程处理函数的实现，而上面的线程是在设备启动的时候开启的，该部分相关程序如下[42]：

```
CHardwareSimulation::
Start (
    IN CTsSynthesizer *TsSynth,
    IN LONGLONG TimePerFrame,
    IN ULONG PacketSize,
    IN ULONG PacketsPerSample
    )
{
.....
```

```
//关于采集线程的操作
    m_RemainData = 0;
    ExitNow = false;
    GatherData = false;//缓冲区中是否有数据
    KeInitializeEvent(&ThreadEvent, SynchronizationEvent, false);
    KeInitializeEvent(&ThreadExiting, SynchronizationEvent, false);
    HANDLE threadHandle;
//内核中创建线程
    NTSTATUS MyStatus = PsCreateSystemThread(&threadHandle, THREAD_ALL_ACCESS, NULL,
NULL,NULL, GatherTsSystemThread, (PVOID)this);
    if( !NT_SUCCESS(MyStatus) )
        return Status;
    MyStatus = ObReferenceObjectByHandle(threadHandle, THREAD_ALL_ACCESS,NULL,
        KernelMode, &ThreadObjectPointer, NULL);
    if( NT_SUCCESS(MyStatus) )
        ZwClose(threadHandle);
    DebugPrint("thread start");
    ……

}
```

以上这些就是开发 capturesample 模块中主要修改的地方，还有其他一些微软提供的框架代码以及其他一些不是很重要的修改我们不再详细解说。同样代码编写好了之后我们进入DDK编译环境切换到工程目录，之后输入build –cz命令之后就能编译出BDACap.sys这个 mini 驱动程序了，配合 bdacap.inf 文件就能安装该驱动文件了。关于这两个驱动程序如何配合DirectShow工作我们将在下一节具体介绍。

5.4 基于 DirectShow 技术播放数字电视

5.4.1 通过 GraphEdit 软件模拟电视播放软件工作流程

把 BDATuner.sys 和 BDACap.sys 这两个驱动程序安装好了之后我们就能通过DirectShow 技术把这由两个驱动暴露出来的过滤器找到，然后编写播放程序来播放数字电视节目了。如图 5-4 所示为我们通过 DirectShow SDK 提供的过滤器编辑软件 GraphEdit 找到的代表 tuner 和 capture 的过滤器。

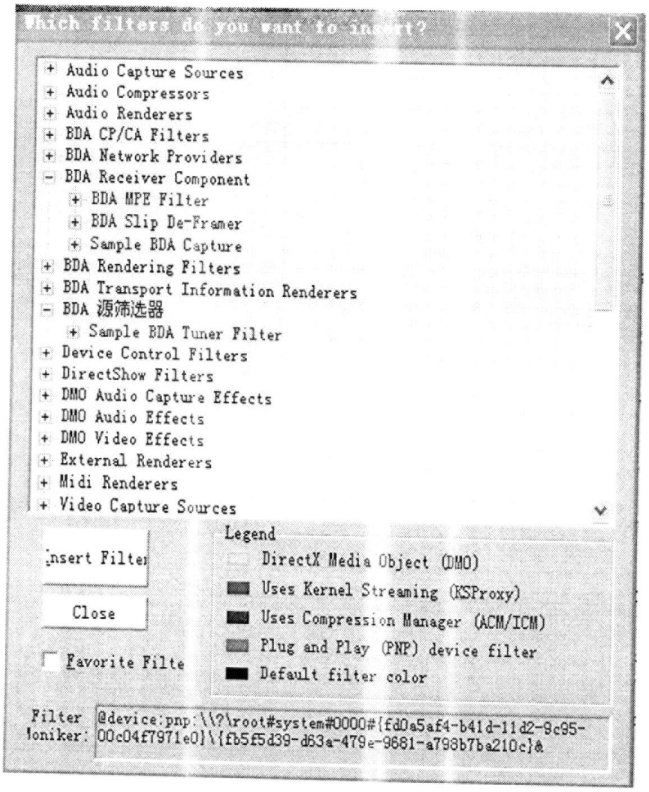

图 5-4　GraphEdit 中找到的内核过滤器

如图中所示的 Sample BDA Capture 与 Sample BDA Tuner Filter 就是 Capture 和 Tuner 这两个驱动程序暴露出来的过滤器。它们的颜色显示出来是红色的，也表明它们属于内核流程序（Kernel Streaming）。我们把这两个过滤器以及其他相关的过滤器都添加到 GraphEdit 之后如图 5-5 所示。

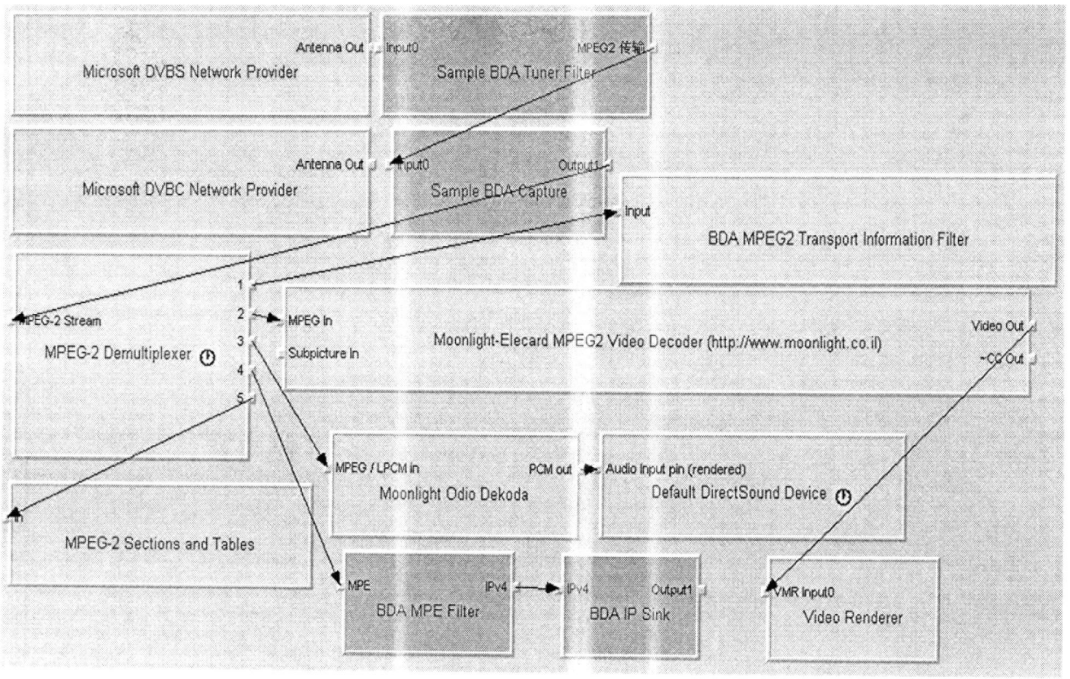

图 5-5　GraphEdit 中播放数字所需的各个过滤器

如图中所示就是一个基本上比较完整的数字电视播放软件需要用到的过滤器。前面根据硬件设备是 DVB-S 或者 DVB-C 可以选择相应的网络类型；之后经过 Tuner 过滤器（包括了 Tuner 和 Demodulator 两部分）进行选频和解调之后输出几路复用的 MPEG2-TS 格式的传输码流；再通过 MPEG-2 Demultiplexer 进行解复用之后把单路的 MPEG2-TS 流数据分别传送给 2 号接口的 MPEG2 解码过滤器和 3 号接口的音频解码器；视频解码回路经过 MPEG2 视频解码之后由 Video Renderer 过滤器把图像数据传送给电脑显卡显示出来，音频解码回路经过音频解码过滤器之后由 DirectSound Device 过滤器把音频数据传送给电脑声卡播放出来。其他还有一些过滤器基本上是辅助用的，如做到调频、选频率、显示节目列表等的作用。在硬件设备连接好之后，再按照上面的连接图将过滤器连接好之后，并对过滤器频率等参数进行设置，设置频率的方法是在 Network Provider 过滤器上鼠标右键属性设置（就是调用了前面章节提到的 BDA mini 驱动中的属性和方法），然后就能设置其中的电视频率等相关参数了，如图 5-6 所示为 DVB-S 的设置图：

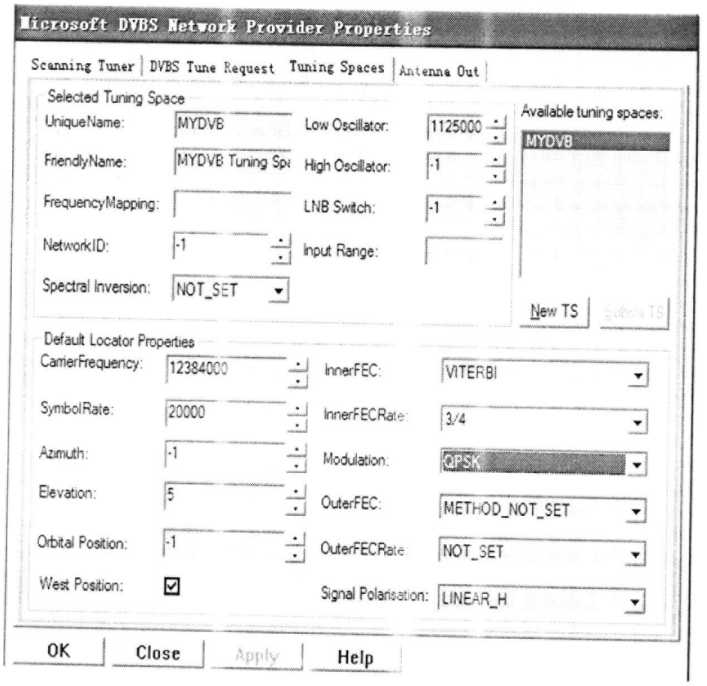

图 5-6　DVB-S 参数设置

设置好频率等相关参数之后只要按下 GraphEdit 软件中的执行(Play)按钮就可以播放电视节目了。由于本课题并未制作实际硬件设备，因此只能用软件模拟读取 TS 数据流并进行视频播放，图 5-7 所示为我们用软件向 DirectShow 的源过滤器发送一路 TS 电视节目数据流来模拟源过滤器从 USB 数字电视接收设备中读取 MPEG2-TS 格式音视频电视节目数据流，并通过 DirectShow 进行实时播放的效果：

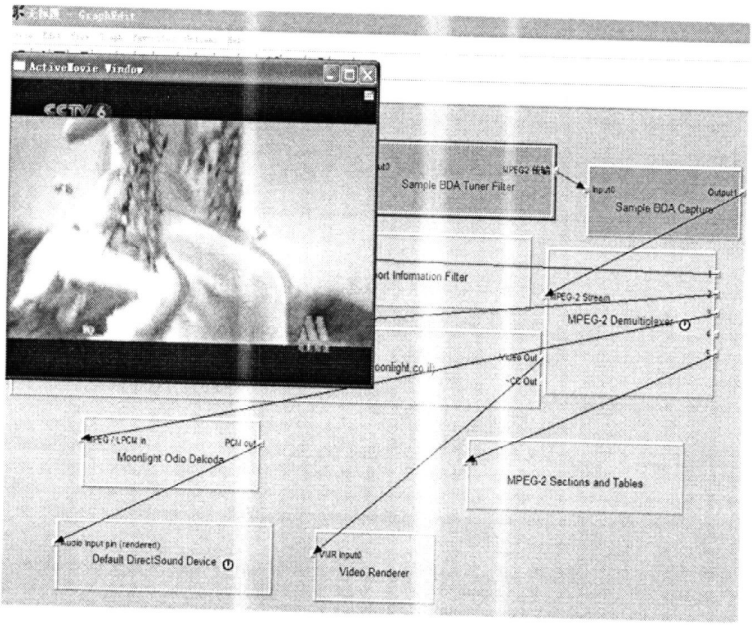

图 5-7　DirectShow 播放 MPEG2-TS 电视节目

上面的 GraphEdit 软件通常只是在模拟调试的时候使用，它能够让编程人员更直观的了解整个电视播放系统的结构。在通过调试之后，接下来就可以根据上面的连接原理通过 DirectShow 技术编写播放软件了。

5.4.2 基于 DirectShow 技术设计电视播放软件

经过上一节 GraphEdit 软件的模拟之后，我们已经清楚了电视播放软件的工作流程，下面我们就根据该工作流程介绍如何编写数字电视播放软件。由于篇幅问题，我们这里省略了对 DirectShow 的一些基础性代码如初始化 COM 组件等相关代码的介绍，只重点介绍如何构建起如图 5-5 所示的完整的播放链路。

一、初始化过滤器管理器 filter graph builder，这个是创建其他所有过滤器的前提，以后创建的过滤器都是依托在这个，相关代码如下[44]：

```
CBDAFilterGraph::InitializeGraphBuilder()
{
……
    // 先判断是不是已经创建了 FilterGraph 管理器了
    if (m_pFilterGraph)
        return S_OK;
    //在确定没有创建的时候创建 FilterGraph 管理器
    if (FAILED (hr = m_pFilterGraph.CoCreateInstance (CLSID_FilterGraph)))
    {
        ErrorMessageBox(TEXT("Couldn't CoCreate IGraphBuilder\n"));
        m_fGraphFailure = true;
        return hr;
    }
……
}
```

二、FilterGraph 管理器创建好了之后就可以添加相应的过滤器了，首先是 Network Provider Filter，不过根据微软的统一调谐理论[42]，在创建 Network Provider Filter 之前我们要先获取一个调谐请求。相关代码如下[44]：

```
……
// 创建调谐请求来初始化 network provider，这里选择的是 DVBS 类型的调谐请求
    CComPtr <IDVBTuneRequest> pDVBSTuneRequest;
    if(FAILED (hr = CreateDVBSTuneRequest(&pDVBSTuneRequest)))
```

```
    {
        ErrorMessageBox(TEXT("Cannot create tune request\n"));
        BuildGraphError();
        return hr;
    }
    //向 network provider 提交调谐请求
    hr = m_pITuner->put_TuneRequest(pDVBSTuneRequest);
    if(FAILED(hr))
    {
        ErrorMessageBox(TEXT("Cannot submit the tune request\n"));
        BuildGraphError();
        return hr;
    }
.....
//加载 NetworkProvider 函数实现
CBDAFilterGraph::LoadNetworkProvider(
    CComBSTR pNetworkType)
{
.....
// 判断是否已经存在调谐空间
    if(m_pITuningSpace == NULL)
    {
        hr = LoadTuningSpace(pNetworkType);
        if(FAILED(hr))
        {
            OutputDebugString(TEXT("BDASample: Cannot fine MYDVBS TuningSpace\n"));
            // 如果没有调谐空间先创建调谐空间
            hr = CreateTuningSpace(pNetworkType);
            if (FAILED(hr))
            {
                ErrorMessageBox(TEXT("Cannot load/create MYDVBS tuning space\n"));
                return E_FAIL;
            }
        }
    }
.....
// 程序到达这里说明，调谐空间已经存在，下面根据网络类型创建 network provider
//网络类型可以是 DVB-S,DVB-C,DVB-T,ATSC 中的一种
    hr = CoCreateInstance(CLSIDNetworkType, NULL, CLSCTX_INPROC_SERVER,
                          IID_IBaseFilter,
                          reinterpret_cast<void**>(&m_pNetworkProvider));
    if (FAILED (hr))
    {
        ErrorMessageBox(TEXT("Couldn't CoCreate Network Provider\n"));
        return hr;
    }
```

```
    //加载好 Network Provider 过滤器之后，把它添加到 graph 中
    hr = m_pFilterGraph->AddFilter(m_pNetworkProvider, L"Network Provider");
    }
.....
```

三、创建好 Network Provider 过滤器之后就能由它来控制整个过滤表的创建了，接下来加载我们的 Tuner 过滤器以及 capture 过滤器，相关代码如下[44]：

```
.....
//加载 Tuner 过滤器
if(FAILED (hr = LoadFilter(KSCATEGORY_BDA_NETWORK_TUNER,
                           &m_pTunerDemodDevice,
                           m_pNetworkProvider,
                           TRUE)))
    {
        ErrorMessageBox(TEXT("Cannot load tuner device and connect network provider\n"));
        BuildGraphError();
        return hr;
    }
    //加载 capture 过滤器
    if(FAILED (hr = LoadFilter(KSCATEGORY_BDA_RECEIVER_COMPONENT,
                           &m_pCaptureDevice,
                           m_pTunerDemodDevice,
                           TRUE)))
    {
        ErrorMessageBox(TEXT("Cannot load capture device and connect tuner/demod\n"));
        BuildGraphError();
        return hr;
    }
.....
```

四、接下来是加载 MPEG2-TS 解复用器 demultiplexer，随后把视频解码、音频解码部分也直接与 demdemultiplexer 连接在一起了[44]：

```
.....
if(FAILED (hr = LoadDemux()))
    {
        ErrorMessageBox(TEXT("Cannot load demux\n"));
        BuildGraphError();
        return hr;
    }
.....
CBDAFilterGraph::BuildAVSegment()
```

```
{
……
    HRESULT hr = E_FAIL;
    // 把 capture 过滤器与 demultiplexer 过滤器进行连接
    hr = ConnectFilters(m_pCaptureDevice, m_pDemux);
    //加载视频解码过滤器
    hr = LoadVideoDecoder();
    if(SUCCEEDED(hr) && m_pVideoDecoder)
    {
        // 把视频解码过滤器与 demultiplexer 过滤器进行连接
        hr = ConnectFilters(m_pDemux, m_pVideoDecoder);
        if(FAILED(hr))
        {
            ErrorMessageBox(TEXT("Connecting Demux & Video Decoder Failed\n"));
            goto err;
        }
    }
    else
        {
            ErrorMessageBox(TEXT("Unable to load Video Decoder\n"));
            goto err;
        }
    ……
    //连接音频过滤器
        hr = LoadAudioDecoder();
        if(SUCCEEDED(hr) && m_pAudioDecoder)
        {
            hr = ConnectFilters(m_pDemux, m_pAudioDecoder);
            if(FAILED(hr))
            {
                ErrorMessageBox(TEXT("Connecting Deumx & Audio Decoder Failed\n"));
                goto err;
            }
        }
        else
        {
            ErrorMessageBox(TEXT("Unable to load Audio Decoder\n"));
            goto err;
        }
……
```

五、最后就是把经过视频解码和音频解码后的数据传给相应的硬件渲染出来[44]：

```
……
//将经过解码的视频数据转交给视频渲染过滤器
hr = ConnectFilters(m_pVideoDecoder, m_pOVMixer);
```

```
        if(FAILED(hr))
        {
            ErrorMessageBox(TEXT("Connecting Capture & OVMixer Failed\n"));
            goto err;
        }
    hr = ConnectFilters(m_pOVMixer, m_pVRenderer);
        if(FAILED(hr))
        {
            ErrorMessageBox(TEXT("Connecting OVMixer & Video Renderer Failed\n"));
            goto err;
        }
        // 将经过音频解码的音频数据转交给音频过滤器
        if(m_pVideoDecoder != m_pAudioDecoder)
        {
            hr = ConnectFilters(m_pAudioDecoder, m_pDDSRenderer);
            if(FAILED(hr))
            {
                ErrorMessageBox(TEXT("Connecting AudioDecoder & DirectSound Device Failed\n"));
                goto err;
            }
        }
    ......
```

到这里我们就完成了图 5-5 所示的各个过滤器连接。这样一个数字电视播放软件的最基本结构就设计好了。图 5-7 为我们编译好的简单的电视播放软件截图。运行播放软件的方法也很简单，点击 Build Graph 就能像我们上面用 Graph Edit 软件模拟的时候一样搭建好过滤器连接图了（需要有硬件设备的时候才能正常的搭建），然后选择 Select Channel 就能出现图 5-6 所示的参数设置界面进行频道选择等各个相关参数设置，之后再点击 Run Graph 就能播放电视节目了。

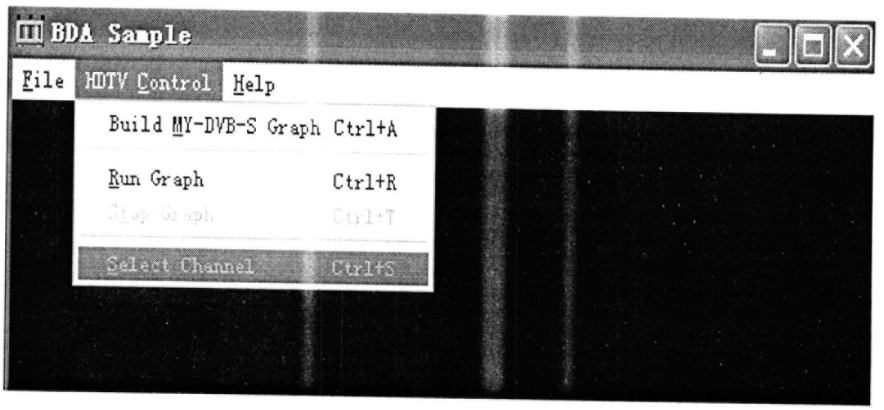

图 5-7　基于 DirectShow 技术的简单电视播放软件

第6章 结论与展望

6.1 结论

至此，经过几个月的程序设计和技术研发，基于USB2.0技术的数字电视接收设备软件设计开发工作已经基本完成，从USB固件程序设计，到USB驱动程序设计，再到USB上位机程序设计这样整个设备开发流程都走完了。经过简单软件模拟测试，本软件系统已经基本上可以正常工作，接下来只要采购具体的高频头与解调器等硬件设备来制作完整的硬件设备，把完整的硬件设计好之后，只需要在本课题研究的软件系统基础上做简单的修改就能制作出完整的数字电视接收产品了。

本文的主要工作和成果如下：

1、在广泛查阅了国内外各种数字电视相关的参考文献以及各种网络资料的基础上，阐述了数字电视接收产品的研究背景及意义，综述了国内外实现数字电视接收技术的研究现状。

2、通过对市面上各种数字电视接收产品设计方案进行分析，确定了数字电视接收设备的整体系统构架，包括系统需求分析，系统模块划分，以及各个主要模块的功能介绍。经过对各个功能模块的分析，确立了USB控制模块以及主机端的控制软件模块设计为整个接收设备的核心部分，并根据实验室所具备的实验条件选取了基于USB2.0技术的数字电视接收设备软件设计作为本文研究对象。

3、通过研究官方提供的程序设计框架以及相关参考资料，完成了USB核心控制模块的程序设计，完成了基于微软DDK开发工具的USB驱动程序的设计以及基于DirectShow与BDA技术的电视播放软件上位机程序设计。

整个毕业设计的过程也是一个不断学习和不断进步的过程，在这几个月时间里不但掌握了很多新的知识，而且对以前的知识也有了更加深入和透彻的理解。期间掌握了USB相关固件程序编写，Windows操作系统下C/C++程序编写，WDM模型设备驱动程序开发等多种技术，也具备了一定的项目开发的能力。在这期间虽然碰到了很多困难，但最后还是把困难全部解决了。毕业设计这段生活让我感受颇深，不仅提高了自己解决困难的能力，更是在踏入社会前的一次很好的锻炼。

6.2 展望

随着科学技术的不断进步，人类离全面数字化信息化的时代已经越来越近，人们对数字电子产品的要求也越来越高，更快的传输速率、更好的便携性、更人性化的操作等等将是今后发展的重点方向。

由于毕业设计时间有限并且实验室设备的局限性，部分环节还存在一些不尽如人意的地方，需要进一步的改进和完善。首先，从整体设计来说，本课题只注重了核心部分的设计，从产品开发角度来说，并未完成整个产品的全部设计，因此在后续研究中可以继续完成全部前端硬件设计并进行真实环境下的产品测试。其次，在现实中由于有些数字电视节目是需要收费而加密的，这在真实产品开发的时候也是需要考虑的，因此在硬件设计时还要加入相应的解密模块。最后，从产品功能性方面来说可以从以下方面进行改进。从传输速率方面来说可以将USB2.0技术提升到USB3.0技术；从便携性方面来说，可以采用集成度更高的一体化芯片；从人性化方面来说，可以加入人机交互功能，如加入红外遥控模块甚至语音控制模块等；从功能丰富性方面来说，可以增加更多附加功能，如新闻、广播、游戏等附加功能也可以集成到电视播放软件界面上等等。

参 考 文 献

[1] 惠新标,郑志航.数字电视技术基础[D].北京: 电子工业出版社,2005.

[2] 杨建华.数字电视技术[D].北京: 北京航空航天大学出版社,2011.

[3] 赵坚勇.数字电视技术[D].西安: 西安电子科技大学出版社,2011.

[4] 刘达.数字电视技术[D].北京: 电子工业出版社,2007.

[5] 涨乐社区.http://www.htsc.com.cn/htnews/news.jsp?docId=4676358 [Z] ,2009.

[6] 吾喜杂志.http://wuxizazhi.cnki.net/Search/QNJZ200711048.html [Z] ,2007.

[7] 人民网.http://media.people.com.cn/GB/137684/13723851.html [Z],2011.

[8] 中商情报网. http://www.askci.com/news/201112/23/174651_67.shtml [Z],2011.

[9] DVB. http://www.dvb.org/about_dvb/history/index.xml [Z],2003.

[10] 杨知行.地面数字电视传输技术与系统[D].北京:人民邮电出版社,2009.

[11] 沈永明.卫星电视接收完全 DIY[D].北京:人民邮电出版社,2011.

[12] 王慧玲.有线电视[D].西安:西安电子科技大学出版社,2010.

[13] 数字电视地面传输攻击特别工作组.GB20600——2006 数字电视地面广播传输系统帧结构、信道编码和调制标准[S].北京:中国标准出版社,2006.

[14] MPEG2-TS wikipedia. http://zh.wikipedia.org/wiki/M2T [Z],2011

[15] 云技术百度百科. http://baike.baidu.com/view/2108643.htm/ [Z],2011.

[16] 数字电视机顶盒百度百科. http://baike.baidu.com/view/1176811.htm/ [Z],2011.

[17] 数字电视棒百度百科. http://baike.baidu.com/view/1871864.htm/ [Z],2011.

[18] 长虹电视官方网. http://www.e-changhong.com/ [Z],2011.

[19] 康佳集团官方网. http://www.konka.com/cn/index.aspx [Z],2011.

[20] 同洲集团官方网. http://www.coship.com/ [Z],2011.

[21] 创维集团官方网. http://www.skyworth.com/cn/ [Z],2011.

[22] 新媒体产业发展研究所.中国数字电视发展现状研究[J].卫士传媒,2010(12):36-39.

[23] 潘云忠,潘宜漾.有线电视机顶盒的原理与维修[D].北京:人民邮电出版社,2009.

[24] USB wikipedia. http://en.wikipedia.org/wiki/Universal_Serial_Bus [Z],2011.

[25] 百度图片. http://image.baidu.com/ [Z],2011.

[26] 薛园园.USB 应用开发技术大全[D].北京:人民邮电出版社,2006.

[27] CEPARK USB 学习网. http://usb.cepark.com/?action-viewnews-itemid-75/ [Z],2009.

[28] 薛园园,赵建领.USB 应用开发实例详解[D].北京:人民邮电出版社,2009.

[29] CSDN. blog.csdn.net/menuconfig/archive/2008/07/29/2729278.aspx/ [Z],2008.

[30] Universal Serial Bus Specification Revision 2.0 [Z],2000.

[31] 薛园园,赵建领.USB 应用开发宝典[D].北京:人民邮电出版社,2011.

[32] Cypress Semiconductor. EZ-USB FX2 Manual Technical Reference [Z],2006.

[33] Cypress Semiconductor. cy7c68013a Manual Technical Reference [Z],2006.

[34] Praveen Kumar. Implementing an 8-Bit Parallel MPEG2-TS Interface Using Slave FIFO Mode in FX2LP[Z],2009.

[35] EZ-USB FX2(68013)固件研究. http://hi.baidu.com/[Z],2009.

[36] Jan Axelson's Enumeration: How the Host Learns about Devices. http://www.lvr.com/usbcenum.html [Z], 2007.

[37] Danny Simpson's An Overview of the Universal Serial Bus. http://www.sss-mag.com/usb.html[Z],2008.

[38] USB 枚举过程 USB Enumeration. http://yuxu9710108.blog.163.com/blog/static/2375153420118853464 4/[Z],2011.

[39] (美)Mark E.Russinovich,(美)David A.Solomon.深入解析 Windows 操作系统 (第 4 版) [D].北京:电子工业出版社,2007.

[40] 张帆,史彩成.Windows 驱动开发技术详解[D].北京:电子工业出版社,2008.

[41] 谭文,杨潇,邵坚磊.Windows 内核安全编程[D].北京:电子工业出版社,2009.

[42] Microsoft. Microsoft Windows Driver Development Kit [Z],2005.

[43] 张配,马勇,董鉴源.深入浅出 Windows 驱动开发[D].北京:电子工业出版社,2011.

[44] Microsoft. Microsoft DirectX 9.0[Z],2002.

[45] 陆其明.DirectShow 开发指南[D].北京:清华大学出版社,2003.

[46] (美)Christopher Tavares.深入解析 ATL[D].北京: 电子工业出版社,2007.

致　谢

本论文即将完成，三年的研究生学习生涯也将结束，回首过去三年，不论是亲人、老师还是同学都给了很大的帮助和支持，在此向他们表示衷心的感谢！

感谢我的指导老师郭淑琴教授以及她的爱人安老师！读研期间，郭老师不仅为我提供了良好的学习环境，给予我很多悉心的指导，而且每周都不辞辛劳地请她的爱人安老师来指导我。本文的选题和研究工作，都是在郭老师和安老师的悉心指导下完成的。在研究的过程中，他们对我的课题提出了许多宝贵的意见和建议。郭老师一向平易近人、为人师表、治学严谨，是我做人、做学问的榜样。

感谢浙江工业大学信息工程学院能够让我在本科毕业之后继续研究生学习，也感谢光纤通信研究所的老师和同学们对我的关心和帮助！

感谢含辛茹苦养育我二十多年的父母，是他们省吃俭用、不辞劳苦地支持着我才使我能完成近二十年的求学生涯，他们是我生命中最重要的恩人，感谢平凡又伟大的父亲、母亲。

最后，对百忙之中抽空评阅本论文以及参加答辩的各位专家、教授表示衷心的感谢！

攻读学位期间参加的科研项目和成果

录用和发表的论文

[1] Jinghui Wang, Zhide Wang. Design of DTV Broadcasting System Based on CY7C68013A [C].Proceedings of 2010 Asia-Pacific Conference on Information Network and Digital Content Security (2010APCID),2010.

[2] 王志德，王晶辉，郭淑琴. DVB-C数字电视接收系统的设计与实现. 浙江省电子学会2011年学术年会论文集[C]. 2011.