摘 要

随着数字电视技术的迅速发展，广播电视进入了数字化时代。数字电视具有图像清晰，频率利用率高，可以开展多种增值业务等优势。数字电视广播中采用条件接收（CA）系统来保证只有被授权得用户才能看到节目。但目前国内的机顶盒与条件接收系统绑定在一起，限制了中国数字电视产业的发展。因此，将机顶盒与 CA（条件接收）系统分离，是数字电视发展的必然趋势。目前实现机卡分离的主流方案是基于 DVB-CI 标准的 PCMCIA 卡(大卡)方案。

本文首先介绍了基于 MPEG-2 的 DVB 数字电视广播技术，着重阐述了 MPEG-2 传送流复用，PSI 和 DVB-SI 信息，以及条件接收技术中的多密技术。同时还对 PCMCIA 硬件规范作了简要介绍，然后详细分析了 DVB-CI 接口的通信协议。

在数字电视解码器上实现 DVB-CI 接口的过程中，采用了有限状态机的思想对通信协议进行分析，并充分利用了嵌入式实时操作系统的任务、消息队列等机制。不仅在数字电视一体机上实现了 PCMCIA 卡与主机的正常通信，还完成了有关的 PSI/SI 信息的提取和分析，实现了用 PCMCIA 卡解密节目，并且用图形界面实现了人机接口。在嵌入式系统开发中，研究了基于面向过程和面向对象的用户界面接口的设计方法，以及窗口消息机制的应用。最后，比较了 Nucleus PLUS 和 OS20 两种嵌入式操作系统的区别，分析了一种机顶盒的系统构成和软件构架，得出了 CI 协议栈的移植方法。

本论文通过结合多个数字电视解码器项目具体开发过程，与 DVB 等技术和思想的研究，阐明了从底层驱动、接口协议栈，到高层应用的 DVB-CI 机卡分离的整个软件实现过程。

关键词：数字电视广播，条件接收，DVB-CI，嵌入式实时操作系统

ABSTRACT

With the quick development of digital TV technology, TV broadcasting has been in the digital era. DTV has the advantage of high definition, efficiency in spectrum utilization and that it provides various new services. Conditional Access System is implemented to ensure that only the entitled customers can watch the programs. But at present in China, the binding of set-top-box with CA system is impeding the development of domestic DTV industry. So the current tendency is to separate set-top-box with CA system. The PCMCIA card solution based on DVB-CI is the mainstream to achieve the independence of set-top-box to CA system.

In this dissertation, the DVB technology, which is based on MPEG-2, and DVB-CI standard are firstly introduced. Then the multiplexing of MPEG-2 transport stream, PSI/SI and the multicrypt technology in Conditional Access are described in detail. Besides, the PCMCIA standard is briefly introduced and the communication protocol of DVB-CI standard is thoroughly analyzed.

In the implementation of DVB-CI on DVB decoder, Finite State Machine theory is deployed to analyze the communication protocol. Then specific mechanisms such as task and message queue in real-time OS are utilized to realize communication between PCMCIA card and Host. Thus the scrambled programs can be descrambled with the support of distilling and parsing the PSI/SI in MPEG-2 transport stream. Two methods are presented to implement GUI in embedded system. One is process-oriented. The other is object-oriented in conjunction with message window mechanism. In the end, two embedded systems, Nucleus PLUS and OS20, are studied to find their difference. Thus, from the analysis of set-top-box system structure, the method for transplanting the CI stack from one embedded system to another can be obtained.

This dissertation has given a detailed method to implement DVB-CI solution to separate the Host and CA module. It has illustrated the complete software system in DVB decoder, including the hardware driver, the communication protocol in the interfaces and the higher level applications.

Keywords: DVB, CA, DVB-CI, Embedded Realtime Opertating System

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：谷蕊　　日期：2006 年 4 月 28 日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位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、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：谷蕊　　导师签名：

日期：2006 年 4 月 28 日

第一章 引 言

数字电视是指从电视节目数字化采集、制作、编辑到节目的数字化播出、数字化传输、用户端对数字节目的接收的全过程。通过数字电视广播系统，用户可以收看多套高清晰度数字电视节目和调频广播节目以及数字音频广播节目，使用各种扩展业务，如图文电视、电视会议、数据信息广播、加密电视、准视频点播（NVOD）等。结合双向传输系统后，数字电视还能实现交互式的多功能应用，如互联网接入、远程教学、远程医疗、电子邮件、计算机联网、数据通讯、家庭保安监控等多媒体信息服务。

总之，数字电视有着美好的前景。可以预见，信息时代中人们的生活会随着数字电视技术的发展而越来越多姿多彩。

1.1 数字电视标准简介

根据上面数字电视的定义，凡是在电视信号的获取、处理、传输和接收的过程中，使用的是数字信号，相应的系统就称为数字电视系统。数字电视系统可以分为 3 大部分：电视信号的数字化及其处理，数字电视信号的传输与交换，数字电视信号的接收与记录。

数字电视本质上和数字通信一样，都是以数据形式传递信息。因此计算机多媒体业务、通信业务和计算机协议在数字电视系统中也得到运用，如 ATM/SDH、TCP/IP 等已经和 MPEG/DVB，DTV方式结合。数字电视的显示格式主要分为两种：SDTV（标清数字电视）和 HDTV（高清数字电视）。我们现在常用的是 SDTV 格式。而 HDTV 虽然清晰度更高，但是因为其传输码率要比 SDTV 高好几倍，因而在数字电视广播方面远不如 SDTV 应用广泛。目前国际上数字电视的传输标准主要有欧洲 ETSI 的 DVB 和美国先进电视制式委员会 ATSC 的 DTV。

DVB 家族分为几个部分，包括：用于卫星数字电视广播的 DVB-S；用于有线（同轴电缆）数字电视广播的 DVB-C；用于地面数字电视广播的 DVB-T；还有新兴的用于移动视频业务的 DVB-H 等。

DVB-S（ETS300 421）为数字卫星广播系统标准。卫星传输具有覆盖面广、节目容量大等特点。数据流的调制采用四相相移键控调制（QPSK）方式，射频传

输频带为 10.7GHz 至 12.75GHz。在使用 MPEG-2 格式时，用户端若达到达到 PAL 质量，码率为 5Mb/s。DVB-S 标准几乎为所有的卫星广播数字电视系统所采用。我国也选用了 DVB-S 标准。

DVB-C（ETS300 429）为数字有线电视广播系统标准。它具有 16、32、64、128、256QAM（正交调幅）多种调制方式，传输频带从 47MHz 至 862MHz。采用 64QAM 时，一个 PAL 通道的传送码率为 41.34Mb/s，可用于多套节目的复用。系统前端可从卫星和地面发射获得信号，在终端需要电缆机顶盒。这也就是我国目前广泛采用的数字有线电视广播标准。

DVB-T（ETS300 744）为数字地面电视广播系统标准。这是最复杂的 DVB 传输系统。地面数字电视发射的传输容量，理论上与有线电视系统相当，本地区覆盖好。采用编码正交频分复用（COFDM）调制方式，在 8MHz 带宽内能传送 4 套电视节目，传输质量高；但其接收费用高。

ATSC 的 DTV 是一种地面数字电视广播标准，目前接受该标准的国家和地区有美国、加拿大、韩国等。另外，北美地区在卫星数字电视广播方面接受 DVB-S、DSS(休斯数字卫星系统)标准；在有线数字电视广播方面接受 Open Cable（美国 Cable Labs 制定的数字有线标准）。

另外，还有日本首先提出的 ISDB（综合业务数字广播）标准，这是一种新型的多媒体广播业务，它系统地综合了各项数字内容，每一项内容可以包括从 LDTV 到 HDTV 的多节目视频、多节目音频、图形、文本等 [1][2]。

DVB、ATSC 以及上面提到的 ISDB 和 Open Cable 标准虽然传输方式不同，但视频和音频压缩标准都基本一致，视频压缩都采用 MPEG-2 标准。

1.2 我国数字电视发展状况及面临的问题

数字电视取代模拟电视已经成为公认的发展潮流，早在上世纪 90 年代，国际上的标准化组织就推出了以 MPEG 为代表的数字电视压缩标准和 DVB 数字电视广播等传输标准。我国也相继推出了一系列数字电视技术规范和管理规章，并在各地进行了有线电视模拟到数字的整体转换。

虽然数字有线电视正不断普及，但有线电视的数字化转换进展并不像想象的那么顺利，有线数字电视用户数量没有达到预期目标，这跟目前数字电视机顶盒机卡绑定的模式有很大的关系。机卡绑定即不同的条件接收（CA）系统要使用不同的机顶盒，例如成都的机顶盒就不能在绵阳使用。这样带来的弊端是：机顶盒

厂商要交给 CA 厂商巨额入门费用，且要开发针对不同 CA 的机顶盒，限制了机顶盒的大规模生产；运营商在负责节目源的同时还成为了机顶盒的购买者，这是一个巨大的负担；消费者被剥夺了选择数字电视接收机的权利，并且看不同运营商的节目就要更换不同的接收机。

中国广电市场的特点是资产结构复杂、网络多、网络小。资产结构复杂使得大范围的整合变得很困难；网络多，各家采用的 CA(有条件接收系统)各不相同，在这种情况下，机卡绑定严重迟滞了数字电视的产业化进程。

1.3 机卡分离的提出

2003 年信息产业部提出了“机卡分离”技术政策，得到国家数字电视研究开发及产业化领导小组的批准和行业内骨干企业的普遍支持[3]。

目前实现机卡分离最常用的方案是基于 DVB－CI 的大卡（PCMCIA 卡）方案。CI 的概念是 DVB 组织提出的，它在集成 CA 方面定义了对于机顶盒独立的硬件接口和软件接口规范，将不同 CA 系统的异性全部封装在 PCMCIA 的模块中。其核心思想就是：将机顶盒和条件接收系统进行分离，使机顶盒脱离条件接收系统供应商的束缚。否则机顶盒产品就永远是一种区域性的专用产品，只能小规模设计和生产，无法真正形成产业优势。

1.4 本论文的结构

本文将首先在第二章中介绍基于 MPEG-2 的 DVB 标准，以及用于实现机卡分离的条件接收多密方案；第三章将在此基础上介绍 DVB-CI 协议的软硬件规范；从第四章开始，将介绍采用 DVB-CI 标准在数字电视解码器上实现机卡分离的具体过程。其中，第四章讲述基于数字电视一体机上的大卡相关软件设计；第五章继续探讨大卡高层应用的实现过程；第六章将在另一个系统——CT216 卫星数字电视机顶盒上分析 CI 协议栈的移植过程。

第二章 DVB 原理与机卡分离相关技术

数字电视的优越性众所周知，图像清晰，频率利用率高，可以开展多种增值业务。因此，数字电视有一部分节目需付费收看，条件接收（CA）是解决电视付费的技术手段。我国有线电视网络中也引入了 CA 系统，并且在各地有多种国内外厂商提供的 CA 系统被采用。目前，在我国有线数字电视网上运行的机顶盒采用的是机卡捆绑的形式，即解扰和解密系统放在机顶盒里，造成了机顶盒制造商直接面对网络运营商，而运营商成为了机顶盒采购商和零售商，但用户却无权选择机顶盒的局面。这在很大程度上限制了数字电视的发展。因而机卡分离成为了一种必然的趋势。

下面将首先介绍数字电视广播（DVB）的有关标准，然后介绍与机卡分离有关的条件接收技术，最后概括目前实现机卡分离的主要方案。

2.1 MPEG-2 和 DVB 标准概述

MPEG-2 的全称是“运动图像及其伴音信息的通用编码”（Generic Coding of Moving Pictures Associated Audio Information）。MPEG-2 是由 MPEG(运动图像专家组（Moving Picture Expert Group）)开发的第 2 个标准。于 1994 年 11 月正式确定为国际标准。MPEG-2 标准是针对标准数字电视（SDTV）和高清晰度电视（HDTV）在各种应用下的压缩方案和系统层的详细规定，编码码率从每秒 3 兆比特～100 兆比特，标准的正式规范在 ISO/IEC13818 中。MPEG-2 专门规定了多路节目的复用/解复用方式，适用于广播级的数字电视的编码和传送[4]。

DVB（Digital Video Broadcasting）是欧洲有 200 个组织参加的一个项目，它包括了卫星、有线电视、地面电视广播等的普通电视和高清晰电视的广播与传输，其目标是要找一种对所有传输媒体都适用的数字电视技术和体系。DVB 卫星、有线系统的信源编码技术是通用的 MPEG-2 视频和音频编码。DVB 对服务信息的格式作了详细的规定，使得不同厂家的 DVB 设备能兼容。同样，由于 MPEG-2 未确定实际的加扰系统和密钥管理系统，DVB 制定了条件接收的公共加扰标准和解码器公共接口标准。对于 MPEG-2 未涉及的一些方面如信道规范，DVB 也作了进一步规定[5]。

2.2 数字电视复用技术

视频、音频信号经编码后生成了各自的基本业务流（ES：Elementary Stream），这些 ES 流以及辅助数据必须复用在一起才能构成一路实际的电视节目(Program)传送流。由于一路节目的传送流的速率与节目内容密切相关，因此在电视节目传输和交换时，将多路节目复用在一起，根据节目内容动态分配其传输带宽，可以大大节省所需的传输频带。在 MPEG-2 系统中，由视频、音频的基本业务 ES 流和辅助数据复接生成的用于实际传输的标准信息流称为 MPEG-2 传送流（TS：Transport Stream），实现复用/解复用功能的系统称为传送系统（Transport System）。

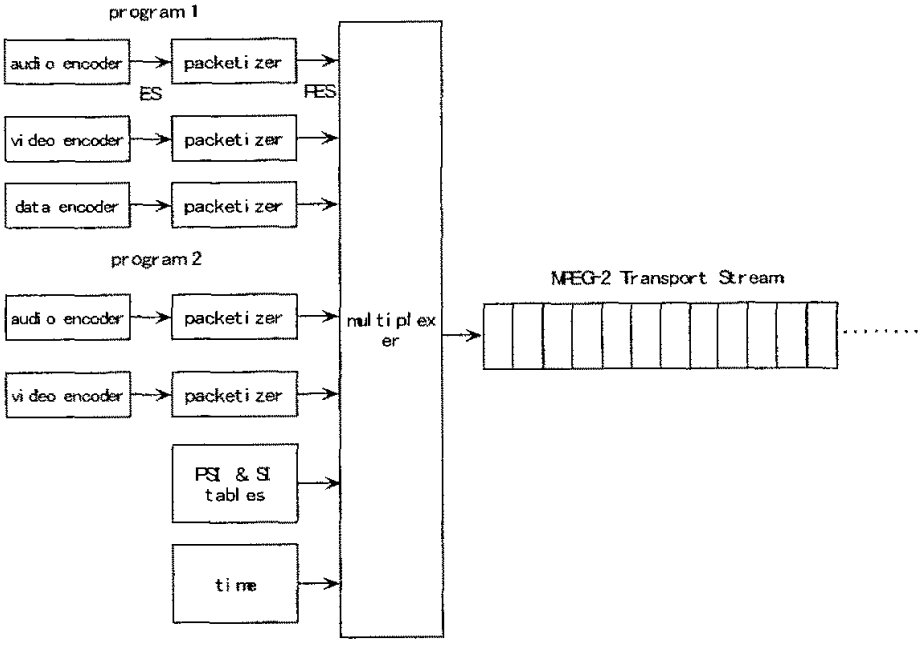

图 2-1 MPEG-2 节目复用传输流的形成

如上图所示，节目 1 和节目 2 的音视频和数据信息被复用成一路传输流，同时在传送流中还加入了 PSI(Program Specific Infomation)和 SI(Service Information)表信息和用于系统解码同步的定时信息，最终形成由一系列连续的定长数据包组成的 TS 流。经过节目复用后，多路电视节目便可以合并为一路传送流，在一个频率点上进行传输。一路模拟电视节目转换成加密数字电视节目后，总码率约为 8Mbit/s。另外，TS 流中还可插入条件接收的加密信息、新业务扩展信息等来实现数字电视广播的丰富功能。

2.3 MPEG-2 PSI 信息与 DVB-SI 信息

首先介绍 TS 流数据包的格式。TS 流数据包长度为固定，其中包头（header）为 4 个字节，有效载荷（payload）为 184 字节。

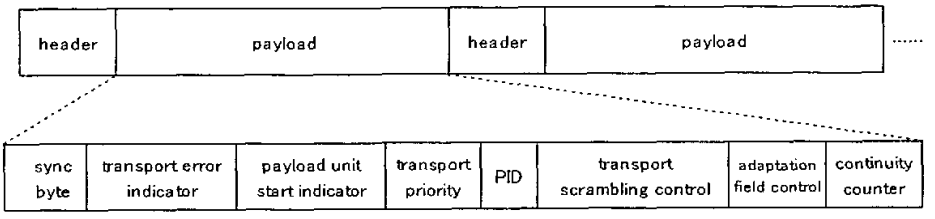

图 2-1 MPEG-2 传送流包结构

数据包头的第一个字节为同步字节，解码时用于解码器硬件的同步匹配，固定为 0x47。有效载荷开始指示（payload unit start indicator）标识用于表明当前包的有效载荷是 PES 包，还是组成（P）SI 表信息的段。除了这两种有效载荷外，系统定时信息（PCR）也包含在 184 字节的有效载荷中，不过它放在适应字段（adaptation_field）中传输。包头中的适应字段控制标识(adaptation_field_control)表明了当前数据包中有效载荷的组成[6]。

数据包头中的 PID 字段（长 13 bits）表明了当前包中有效载荷的内容，为了能够找到用户所需的节目，解码器首先要知道节目的音频 PID、视频 PID、数据 PID（图文或字幕）等，为此 MPEG-2 定义了节目特定信息（PSI）表来帮助解码器找出用户所需的节目内容。DVB 标准在 MPEG-2 定义的扩充私有信息之上定义了业务信息（SI）。其中，PSI 信息和 SI 信息都以表（table）的形式定义。

节目的音视频及图文字幕等信息首先被压缩成基本流（ES），经过打包器后形成定长的数据包（PES），然后放到有效载荷中，PSI 和 SI 表信息则以更复杂的语法形式－段（section）放到其中，一个完整的表由一个或多个段构成。在段语法中，还可以利用描述子（descriptor）语法来增加新的内容。

PSI 由节目关联表（PAT）、条件接收表（CAT）、节目映射表（PMT）和网络信息表（NIT）组成，这些表被插入到 TS 中传输。PSI 指定了如何从一个携带多个节目的传送流中正确找到特定的节目的方法。当接收机要接收某一个指定节目时，它首先从节目关联表中取得这个节目的节目映射表的 PID 值，然后从 TS 中找出与此 PID 值相对应的节目映射表，从这个节目映射表中获得构成这个节目的基本码流的 PID 值，根据这个 PID 值滤出相应的视频、音频和数据等基本码流，解

码后复原为原始信号，删除含有其余 PID 的传送包。

PSI 数据只提供了单个 TS 流的信息，但它不能提供多个 TS 流的有关业务和节目的类型、什么节目、什么时间开始等信息，因此 DVB 系统对 PSI 进行了扩展，提供了其它不同信息种类的多种表格，形成了 SI。在实用中，我们将 SI 所提供的数据通过有序地组织起来，生成类似节目报的形式，它能在电视机上即时浏览，这样将大大方便用户的使用，这就是电子节目指南（EPG）。SI 中的各种表在实际使用中并不都需要传送，其中 NIT、SDT、EIT、TDT 是必需传送的，其它表则按照需要进行选择传送。

TS 中的标识符除了包标识符，还有表标识符（Table ID）。具有相同 PID 的不同信息表则由表标识符来区分。例如 SI 中的 SDT 的每一个表都对一个特定 TS 流中的业务进行描述，这些业务可能是这个表所存在的 TS 流的一部分，也可能是其它 TS 流的一部分，这些 SDT 的 PID 都是相同的，这时候我们就可以通过不同的表标识符来区分它们[7][8][9]。常用的一些节目信息表如下：

表 2-1 常用节目信息表

节目信息表	PID	Table ID
PAT 节目关联表	0x0000	0x00
CAT 条件接收表	0x0001	0x01
PMT 节目映射表	0x0002	0x02
NIT 网络信息表	0x0010	0x40
SDT 业务描述表	0x0011	actual_ts 0x42; other_ts 0x46
EIT 事件信息表	0x0012	actual_ts, p/f 0x4e; other_ts, p/f 0x4f actual_ts, schedule 0x50~5f; other_ts, schedule 0x60~6f
TDT 时间日期表	0x0014	0x70
TOT 时间偏移表	0x0014	0x73

2.4 数字电视条件接收与机卡分离技术

2.4.1 数字电视条件接收加解扰过程

条件接收系统（CA:Conditional Access）的技术途径是通过对数字电视信号进行加扰，即对传输码流中所含的视频和音频信息以及数据信息的加扰，同时将加扰处理的控制字经加密后随传输流一起传送。在用户端，未经授权的用户将不能对付费节目进行解扰，从而无法收看该节目。条件接收系统由加扰器、解扰器、加密器、控制字发生器、用户授权系统、用户管理系统和智能卡等部分组成。在

发送端，将原始数据流（TS）与控制字（CW）进行模二加形成加扰的数据流（TS）。TS 的加密密钥 CW 是一组按一定频率变化的随机数，在接收端要用同样的控制字控制同样的电路进行解密。通常说的加密节目其实就是被加扰的节目[10]。

加扰控制字 CW 在对数据流进行加扰的同时，本身也被业务密钥 SK（Service Key）加密，形成授权控制信息 ECM（Entitlement Control Message）。ECM 大约每几秒中在传送流中出现一次。

CW 虽已由 SK 加密生成 ECM，但 ECM 是公开传输的，可以让任何人读取，所以必须再对 SK 进行加密保护。SK 在传送前往往用和终端设备地址码相关联的一个数列，即个人分配密钥 PDK（Personal Distribute Key）来加密处理，形成授权管理信息 EMM（Entitlement Management Message）。PDK 一般由 CA 系统设备自动产生并严格控制，在终端设备处该序列数一般由网络运营商通过 CA 系统提供的专用设备烧入解扰器的只读存储器中，不能再读出。用户管理信息由服务提供商的用户管理系统形成，包括用户名称、地址、智能卡号、帐单等等。EMM 约每 8-10 秒插入传送流一次[11]。

以上就是条件接收三级加密的过程。

目前智能卡授权方式是机顶盒市场的主流，智能卡也被我国广电总局确定为我国数字有线电视入网设备的标准配件。在用户端，当智能卡插入时，机顶盒根据 PMT 和 CAT 表中的 CA-descriptor，获得 EMM 和 ECM 的 PID 值，然后从 TS 流中过滤出 ECM 和 EMM，并通过 Smard Card 接口送给 Smart Card。Smart Card 首先读取用户个人分配密钥（PDK），用 PDK 对 EMM 解密，根据解出的 EMM 信息来确定本智能卡是否被授权收看该节目，如果没有授权将不能进行后续解密。如果该卡已被授权，解出 SK，然后利用 SK 对 ECM 进行解密，得到 CW，并将 CW 通过 Smart Card 接口送给解扰器，解扰器利用 CW 结合解扰算法就可以将加扰的传送流进行解扰。

2.4.2 基于条件接收多密技术的机卡分离技术

2.4.2.1 同密技术与多密技术

数字电视加解扰技术分为同密和多密技术。

同密技术的方案基于 DVB 统一规定的加解扰算法通用加扰算法（CSA），目的是将多种 CA 系统应用于同一网络中，适用于 CA 软件嵌入到机顶盒中的情况。加扰时各 CA 系统采用统一的加扰算法 CW 控制字，加密 CW 时才采用各自的方式产生

不同的 ECM。在接收端接收机通过 ISO-7816 标准接口访问 CA 厂商提供的智能卡，由智能卡根据授权信息解密出 CW 控制字，再送回机顶盒中通过 CSA 解扰数据。这样智能卡中只有控制解密部分，所以成本较低。

一款机顶盒一般只能捆绑接收用某特定 CAS 加密的节目，对用户和运营商存在更换 CA 就需更换机顶盒的风险。

DVB 多密技术的基本思想是将解扰、CA 以及其他需要保密的专有功能集中于一个可拆卸的模块中，机顶盒（又称主机）功能可以趋于通用化，其中只包含调谐器/解调器、MPEG-2 解码器、解复用等必须的设备，具有接收未加扰或已解扰的 MPEG-2 视音频、数据的功能。这也就是我们说的机卡分离。解扰的功能完全由与机顶盒独立的 CA 模块完成。

这种方案的好处在于，只要更换相应的 CA 模块，就能用同一机顶盒接收任意 CA 系统加扰控制的节目 [12]。

2.4.2.2 机卡分离的具体方案

目前机卡分离的方案分为大卡方案和小卡方案。

小卡方案是指采用 ISO7816 标准的智能卡（小卡）接口，把与 CA 有关的功能划分成“通用模块”和“私有模块”两个部分，通用模块在机顶盒内，私有模块在智能卡内。这种方案解扰器的部分仍保留在机顶盒中，而且要协调各厂商来划分模块存在相当的困难。

大卡方案包括 PCMCIA 大卡方案和基于 USB2.0 技术的 UTI 大卡方案，都是由大卡完成解扰的功能，实现了 CA 功能与机顶盒的彻底分离。大卡方案开发的主要任务是规范机顶盒与 CA 模块的接口，并在 CA 模块上实现三级解密。

PCMCIA 大卡方案的 CA 模块通常是一块 PCMCIA 卡或者是由一块 PCMCIA 卡与一张智能卡组成， CA 模块也被称为 CAM(Condional Access Module)卡。PCMCIA 接口是基于总线的开发平台，方便于技术的升级和扩展应用，而且其并行的数据线结构有利于大量的数据传输。PCMCIA 大卡方案的种种优势使其成为目前机卡分离的主流方案。

国外的机卡分离接口标准，一个是欧洲的 DVB-CI 标准 ，一个是美国的 POD（Point Of Deployment），都是采用 PCMCIA 标准作为物理接口标准。DVB-CI 标准将在下一章中详细介绍。

第三章 PCMCIA 标准与 DVB-CI 协议

DVB-CI（Common Interface）是 DVB 组织提出的用于集成独立于机顶盒的 CA 模块的硬件接口和软件接口规范，也是通常说到的机卡分离大卡方案的规范。

DVB-CI 的硬件接口采用 PCMCIA 规范，软件接口包括两个逻辑接口的定义，较复杂的是其中的命令接口的通信协议。下面首先对 PCMCIA 标准作一个介绍。

3.1 PCMCIA 标准介绍

PCMCIA（Personal Computer Memory Card International Association Industry Standard Architecture，PC 机内存卡国际协会）是一个由 500 多家公司组成的国际组织。该组织制定了 PCMCIA 接口卡（简称 PC 卡）标准。PCMCIA 标准最初的意图是给移动电脑添加内存，后来经过几次扩展，现在已用于多种插件，如视频会议卡、调制解调器等[13]。PC 卡的特点有：

（1）外形小巧：长宽为 85.6mm×54mm，厚度有 3 种规格：TYPE 1 为 3.3mm，TYPE 2 为 5.0mm，TYPE 3 为 10.5mm；

（2）与主机总线独立：PC 卡的插槽（Socket）通过特定的主机总线适配器（HBA：Host Bus Adapter）可以与各种不同的主机总线连接；

（3）支持 3 种地址空间：标准存储地址空间，I/0 地址空间以及用于自动配置时的属性存储地址空间；

（4）26 条地址线为每种地址空间方式提供 64MB 地址空间；

（5）TYPE 1，2，3 的 PC 卡采用 16-bit 数据线，另外还有一种 Card Bus 类型支持 32-bit 数据线；

（6）支持 I/0 设备以及 DMA；

（7）支持热插拔及自动配置；

（8）低电压需求：支持 5.0 V，3.3 V 和 x.x V 类型电压；

3.1.1 PC 卡软硬件模块间的关系

PCMCIA 标准包括 4 各方面的定义：PC 卡的物理设计、插槽的物理设计、PC 卡的接口电气特性以及软件构架。

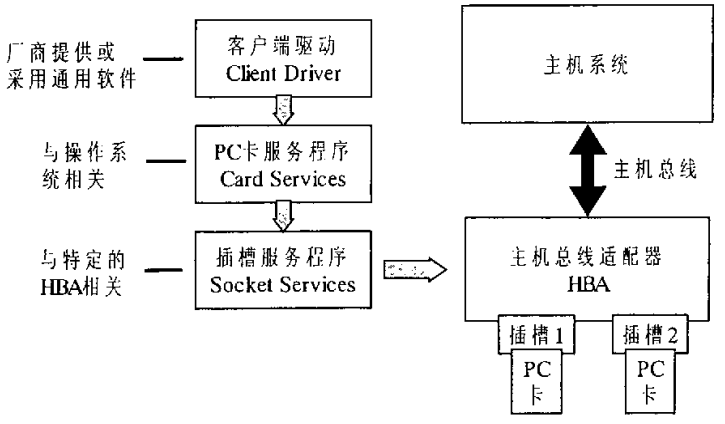

图 3-1 PC 卡软硬件结构

PC 卡的驱动程序主要用来对主机总线适配器和 PC 卡进行配置，以使 PC 卡正常工作。PC 卡驱动包括 3 个层次：客户端驱动、PC 卡服务程序和插槽服务程序。

一旦插入 PC 卡，客户端驱动就根据 PC 卡中的配置信息对 PC 卡进行配置，拔出卡时客户端驱动也要进行相应操作。PC 卡服务程序为上层软件提供接口函数，并为 PC 卡提供分配系统资源的方式。客户端驱动读出 PC 卡配置信息后，就按照其要求来申请系统资源。PC 卡服务程序则记录所有可分配的系统资源信息。这样，客户端驱动就能判断 PC 卡的哪一个配置选项可以得到满足。另外，由于 HBA 没有统一的规范，所以通过插槽服务程序，可以为驱动开发人员屏蔽 HBA 的细节。

3.1.2 PC 卡的 CIS 信息与配置寄存器

PC 卡的配置信息以 CIS（Card Information Structure）形式保存在卡上的非易失存储器中。CIS 信息包括 PC 卡的速度、字长以及需要的系统资源等信息。获取这些信息后，主机就能配置 HBA 来访问 PC 卡，并通过写 PC 卡上的配置寄存器（COR：Configuration Option Register）来配置 PC 卡。PC 卡的 CIS 和配置寄存器都映射在属性存储地址空间中。

CIS 信息从属性存储地址空间的起始地址 0 开始，其组织形式是以“tuple”为单元的一个链表。CIS 数据只映射在偶地址中，因此其信息只在低字节数据线（D7:D0）中传输，这样便于与 8 位的系统兼容。Tuple 第一个字节为 Tuple_Code，表示该 tuple 的类型，第二个字节为 Tuple_Link，表示 Tuple_Data 的长度，从第三个字节开始就是数据内容 Tuple_Data，Tuple_Data 以 0xff 为结束标志，紧接着就是下一个 tuple。这样，只要读到了第一个 tuple，就可以进而访问到整个 CIS 信息。

对于需要进行专门配置的 PC 卡，其 CIS 信息含有一个配置表（Configuration Table），配置表提供若干个配置选项供主机在初始化时选择。配置表的第一个选项为默认选项，包含了完整的配置信息，例如电压、时序、中断、地址空间等方面的要求。如果主机不能满足配置选项中的所有要求，则分析下一个配置选项，直到主机能满足 PC 卡的所有要求，否则配置失败。

PC 卡可以有多种功能配置寄存器（Function Configuration Register），其中 PCMCIA 规定必须的是配置选项寄存器（COR）。一旦驱动程序分析了 CIS 并获取了 PC 卡所需的系统资源后，就把对应的配置选项的序号写入 COR，然后 PC 卡就可以使用这些资源了。

3.2 DVB-CI 接口规范

3.2.1 DVB-CI 实现框架

采用 DVB-CI 规范的数字电视解码器由机顶盒和外部模块两部分组成。机顶盒完成所接收高频信号的解调，并对从外部模块传来的解密后的码流解复用和 MPEG-2 解码，最后在电视终端上输出电视节目；而外部模块即大卡和智能卡组成的 CA 模块完成解调后码流的条件接收[14] 。

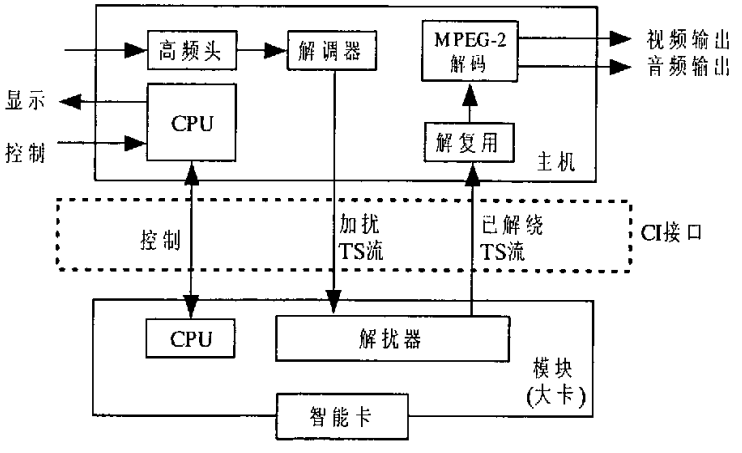

图 3-2 主机与单个模块连接示例

如图，在机顶盒中解调后的 TS 流通过 CI 接口被送到外部的大卡上， 大卡根据主机送来的 PSI 从 TS 流中提取到 ECM、EMM 包，送给 CA 公司授权的智能卡，由智能卡解出当前节目的密匙，送还给大卡来进行节目解扰[15] 。

3.2.2 DVB-CI 的软件接口规范

DVB-CI 定义了两个逻辑接口：一个是 MPEG-2 TS 流接口，包括物理层，链路层，传输层和更高的层次，其中后 2 者是在 MPEG-2 标准中定义；另一个是更为复杂的命令接口，分为 5 个层次：物理层，链路层，传输层，会话层，应用层。

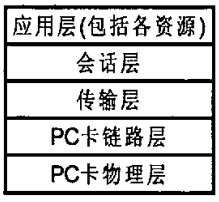

图 3-3 CI 命令接口层次

当主机连接了多个模块，TS 流通过雏菊链（daisy-chain）形式连接，主机负责维护与各模块的命令接口的连接，并且根据某个服务的解扰形式来选择对应模块。要在机顶盒上实现 CI 软件接口，关键是命令接口的实现。

首先介绍 DVB-CI 中的几个基本概念：

（1）资源（resource）：应用层提供的服务功能被封装成不同的“资源”来供使用，资源可由主机或者某个模块提供，每个资源用资源标志（resource_identifier）来表明其类型，级别，版本号。

（2）应用(application)：指的是运行在模块如大卡上的应用，它可以与主机通信并使用资源来给用户提供服务，类似 Client-Server 模式中的客户端，而主机相当于对应的服务器。

（3）模块（module）：指的是主机外的一个小型设备，例如大卡，它执行特定的任务与主机协同工作。

（4）对象（object）：DVB-CI 每个层次都有特定的 PDU 编码规则，通信数据用“对象”定义，并用 ASN.1 语法的通用 TLV 方法编码，即：object{Tag; Length; Value; }。

CI 命令接口的物理层和链路层是基于 PCMCIA 硬件规范的，传输层的实现需要下层的支持来保证，会话层和应用层则基于 Application。下面介绍 DVB-CI 中定义的传输层、会话层和应用层的对象和通信规则。

（1）传输层

传输层定义了 11 个对象：Create_T_C and 和 C_T_C_Reply 用于建立传输连接；

Delete_T_C 和 D_T_C_Reply 用于关闭连接；Request_T_C 和 New_T_C 用于模块请求建立新的连接；T_C_Error 用于报错；T_SB 用于模块向主机传送状态信息；T_RCV 用于主机向模块请求数据；T_Data_More 和 T_Data_Last 用于传送数据。

主机与模块间的每个传输层连接都有一个 8 位的传输连接号。连接由主机首先发送 Create_T_C 开始建立，模块不能主动发起通信。主机有 4 个状态：空闲，建立连接，活动，关闭连接。若模块要向主机发送数据，须等待主机询问，然后发送 T_SB 表示要发送数据，主机收到后用 T_RCV 应答，然后模块才能用 T_Data 发送。

（2）会话层

会话层提供了利用资源的机制。使用一个资源时，要先通过会话层与相应资源建立会话连接，并分配一个唯一的会话连接号。不同的资源可以同时支持一个或多个会话。

会话对象包括：open_session_request 和 open_session_response 用于模块请求建立连接；create_session 和 create_session_response 用于主机建立扩展的会话连接，使模块与非主机上的资源通信；close_session_request 和 close_session_response 用于关闭会话；session_number 用于承载 APDU 数据。

open_session_request 由模块发出，并且指明请求资源的资源标志，主机回复的 open_session_response 中用 session_status 表示是否可建立会话，如果可以，则用 session_num 字段表示连接号。之后该会话的每个 SPDU 都要包含这个连接号。如果模块请求的资源不是由主机提供，则主机向提供该资源的另一模块发送 create_session，另外建立一个会话连接，这时主机充当模块间的通信媒介。关闭会话的请求双方都可以发出。

（3）应用层

APDU 的结构与 SPDU 类似，但 apdu_tag 为 24 位，spdu_tag 位 8 位。根据缓冲区大小的限制，可采用 APDU 链接机制将一个 APDU 可以分成多个 APDU 来传送，最后一个 APDU 采用 Last_apdu_tag，其它的 APDU 使用 More_apdu_tag。同一个会话的 多个 APDU 可以封装在一个 SPDU 中传输。

应用层的服务功能由各种资源实现。由主机提供的系统资源包括：

1．资源管理（Resource Manager，以下简称 RM）

该资源只有一种类型，支持任意数量的会话连接，用于控制所有资源的获取和供给。主机与模块间采用一种对称通信协议，主机通过传输层连接来检测各个可用资源，并且监控资源的变化情况。主机建立一个资源列表，对任何资源的请

求都通过 Resource Manager 来统一管理。RM 始终保持与模块的会话连接，以便随时响应资源请求或者更新。

RM 的对象有：profile_enq；profile_reply；profile_changed；

2. 应用信息资源(Application Information，以下简称 AI)

Application Informaiton 资源使应用可以向主机给出自己的一套标准的信息。大卡一旦完成初始化中的与 RM 的 profile_enq 步骤，就与 AI 建立会话连接，并在收到询问后提供自己的信息。这个连接将一直保持，以使主机可以随时通知应用在最菜单最顶层入口（entry point）建立 MMI 连接。

AI 的对象有：application_info_enq，application_info，enter_menu。大卡的 application_info 中，包含 CA System ID 以及厂商设定的有关 CA 系统的字符信息。

3. 条件接收支持资源（Conditional Access Support，以下简称 CAS）

这个资源提供一套对象以支持条件接收应用。在完成 AI 的初始化后，所有的 CA 应用都与该资源建立会话连接，并一直保持这个连接来实现有关解扰的操作。主机向各个 CAS 应用发出查询对象，后者用 CA 信息对象将其信息返回给主机。

CAS 的对象有：ca_info_enq，用于主机向 CA 应用查询其信息；ca_info，用于 CA 应用向主机报告其支持的 CA_system_id，这样主机就可以选择使用与基本流（Element Sream）的 CA 描述子里的 CA_system_id 相符的 CA 应用来解扰；ca_pmt，用于主机向选择的 CA 应用发送 PMT 表中的 CA 描述子，使 CA 应用可以得到解扰需要的 ECM 流信息。其中的 ca_pmt_cmd_id 用于指定要求 CA 应用作出何种响应；ca_pmt_reply，用于响应 ca_pmt_cmd_id 设为 query 或 mmi 的 ca_pmt 对象，分别返回对应的 CA_system_id 或者节目授权情况。

除了以上的主机提供的基本系统资源外，通常还有：

1. 主机控制资源（Host Control）

该资源用于允许应用有限地控制主机的操作，两个主要功能是：重新调谐主机到另一个服务（节目）；暂时替换当前的服务。

2. 日期时间资源（Date Time）

该资源用于由主机向应用提供当前的日期和时间，其值可以由 SI 信息中的 TDT 表得到。

3. 人机接口资源（Man-Machine Interface，以下简称 MMI）

MMI 有高，低两种级别的接口。低级别的 MMI 模式使应用可以控制操作细节，如可接受的用户按键值，显示特性等。高级别的 MMI 模式提供了菜单，列表，对话框等高级别的语义定义，其具体的显示方式由主机决定。

在两种模式下都能使用的 MMI 对象有：close_mmi，display_control（用来设定使用哪种 MMI 模式），display_reply。

高级别的 MMI 对象有：text，enq，answ，menu，menu_answ，list 。

第四章 DVB-CI 接口在 iDTV 上的实现

iDTV(Integrated Digital Television)即一体化数字电视机，集成了模拟/数字有线电视节目的接收和显示功能，解除了人们在购买高清数字电视后还要再购买外置机顶盒才能收看数字节目的束缚，算得上真正的数字电视机。本章介绍的机卡分离方案就实现于某彩电公司的机卡分离数字电视一体机的数字模块（简称 DTM）中。DTM 采用意法半导公司的 QAMi5516 数字视频解码芯片[16]，实现 DVB-C 标准的有线数字电视的节目搜索、播出、管理，以及 EPG（电子节目指南）等功能。

公用接口（CI）是机卡分离数字电视一体机对数字电视信号进行解扰的最重要的一部分，它是数字电视与 PCMCIA 卡(大卡)及 SMART 卡(小卡)之间进行信息交换的接口。

OS20 是 ST 公司为 ST20 系列 CPU 提供的实时多任务操作系统，也是本章软件应用的平台。下面首先对 OS20 操作系统作一介绍。

4.1 OS20 实时操作系统介绍

OS20 实时内核提供了多种任务服务。各任务可通过信号量和消息队列进行同步和通信。各种事件可通过中断处理并使用信号量和任务进行通信。内存分配可由用户或 OS20 来管理。各任务可以分配不同的优先级并被调度。同时还提供了时钟函数[17]。

OS20 有以下特点：

- 与硬件高度集成。
- 16 个优先级，抢占式调度。
- 提供信号量
- 提供消息队列
- 提供时钟
- 提供内存管理
- 提供中断处理
- 上下文切换时间小等于 6 微秒。

1、内核

OS20 使用了一个很小的调度内核。调度策略使用基于优先级的抢占式调度。内核保证了当前运行的任务总是就绪任务中优先级最高的。

内核维护着两个重要的信息：

- 当前正在执行的任务及其优先级。
- 当前准备就绪的任务的列表。

以下三种情况时任务会被内核调度：

（1）当一个任务需要被调度时，调度程序先判断这个新的任务的优先级是否比当前正在执行的任务的优先级高。如果是，那么当前任务的状态信息被保存，新任务的状态信息被载入，并开始运行。

（2）当前运行的任务进入等待状态时，调度程序将使就绪的任务中优先级最高的一个运行。

（3）调度程序会被周期性地调用以使具有相同优先级的任务能够每隔一定的时间片轮换着执行。

2、内存和分区

由于嵌入式系统中的存储器通常很小，而且需要较好的访问效率，所以嵌入式系统中的内存管理是非常重要的。OS20 提供了三种不同的内存管理方式，给用户管理内存分配提供了很大的方便，用户可在时间和空间的效率上权衡利弊。

内存管理的基本工作是允许应用程序从一大块内存中分配和释放一小块内存。OS20 中使用了分区这一概念来代表大块内存。有三种类型的分区：

（1）堆（heap）分区。和传统的 C 语言运行时环境中的“堆”风格相同。可从此分区中分配并释放不同大小的内存。堆风格的内存分配给用户提供了很通用的功能，但也具有很多缺陷。堆的算法决定了内存分配和释放的时间是不确定的，过多的分配和释放还会导致内存变得支离破碎；由于需要额外的内存来存储必要的信息，增加了内存的开销。

（2）固定（fixed）式分区。在这种分区当中，分配的内存是固定大小的，从而克服了前述的一些缺点。

（3）简单（simple）分区。对于这种分区中的内存分配采取了一种简单的策略，每次只是对一个指针进行累加，使其指向下一段可被分配的内存。这就意味着不能再释放已分配的内存，但其好处是不会浪费一点内存，各种大小的内存都可以分配，每一次分配消耗的时间也是固定的。所以，非常适合用于宝贵的高效的较小的片上内存。

OS20 不会自己动态地分配内存，从而使用户可以管理所有的内存分配，构造完全确定的系统。但是操作系统的很多函数需要进行分配内存的操作，在这种情况下，OS20 要求定义两个预定义分区：

(1)system_partition 被 OS20 中的大多数模块使用，包括信号量、消息队列以及任务的数据结构中的静态部分和栈。这一分区要求是堆分区。

(2)internal_partition 只被由 task_create 开创的任务的数据结构中的动态部分使用。为了提高上下文切换的时间，这一分区最好从片上内存中分配，并且最好使用简单分区。

常用函数如下：

memory_allocate()，memory_deallocate() //从简单分区中开辟或释放内存

partition_init_simple()，partition_delete() //开创，删除一个简单分区

对于堆分区和固定分区，也有类似函数。

3、任务

任务是独立运行的线程。一个任务描述了应用程序一个可分离的独立组成部分，除了有时需要和其他任务通信之外，它完全是一个独立的程序。任意一个已存在的任务都可以动态地开创一个新的任务。只要有足够的内存，应用可被分成任意多个任务。

一个任务由一个任务控制区、栈和一段代码组成。任务的控制区记录了其状态，其内容和结构是和处理器密切相关的。OS20 的任务控制区被分成两部分：

（1）动态状态部分。在结构 tdesc_t 中定义，由 CPU 直接使用。这一结构和处理器密切相关。这一结构最重要的部分就是机器寄存器，特别是指令指针(Iptr)和工作区指针(Wptr)。任务的优先级也存储在这一结构。当任务运行时，Iptr 和 Wptr 保存在 CPU 寄存器中，当任务没有运行时，它们被存储在 tdesc_t[18]。

（2）静态状态部分。在结构 task_t 中定义。OS20 使用这一结构来描述任务，当任务运行时，这一结构的值不会被改变。

OS20 使用 16 个优先级。相同优先级的任务采用时间片轮换的方式。各个任务间可以通过信号量进行同步，通过消息队列进行通信。常用函数如下：

task_create()，task_init() //开创一个任务

task_delete() //删除一个任务

task_priority_set() //设定一个任务优先级

task_delay() //使一个任务等待一段时间

task_reschedule()，task_suspend() //对任务调度进行控制

4、信号量

信号量提供了一种简单有效的方式来同步多任务。信号量可以用于互斥访问，任务同步。OS20 支持优先级类型的信号量，也可支持 FIFO 类型的信号量，还支持永久等待或指定等待时间的信号量。常用函数有：

semaphore_init_fifo()，semaphore_init_priority()

semaphore_create_fifo()，semaphore_create_priority()　//开创一个信号量

semaphore_wait()　//使任务等待一个信号量

semaphore_signal()　//使任务释放一个信号量

5、 消息队列

消息队列为任务提供了一种缓存通信的方法，通信时不需要拷贝数据。OS20 的消息队列使用两个队列，一个是空闲队列，一个是已发送但尚未接收的消息的队列。当用户调用各种消息队列的函数时，消息缓冲区就在这两个队列中循环。

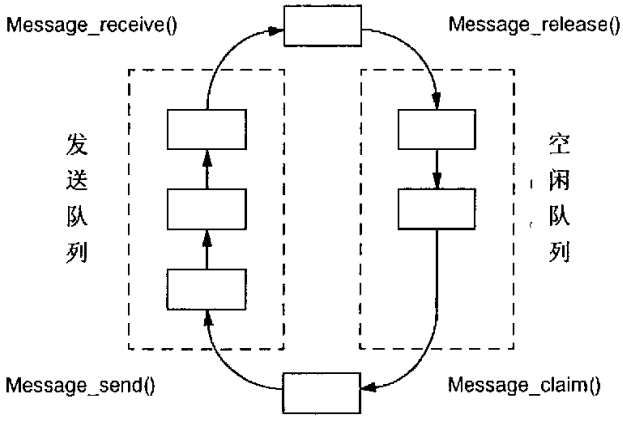

图 4-1　消息队列

常用函数有：

message_claim()　//申请一个空闲的消息缓冲区

message_create_queue()，message_init_queue()　//创建一个消息队列

message_receive()　//接收消息队列中的一个消息

message_release()　//释放一个消息缓冲区

message_send()　//发送一个消息到队列

6、 时钟

时钟对于实时系统来说是非常重要的。OS20 提供了一些基本的函数来控制时钟。ST20 处理器通常把时钟看作是一个循环，也就是说代表时钟的计数器溢出后会重头开始计数。计时的单位为 tick。

常用函数有：

time_now()　　//得到当前的时间

time_after()　　//判断一个时间是否在另一个之后

time_minus()，time_plus()　　//加减时钟的数值

7、中断

中断使外部事件能控制 CPU。中断切换完全由硬件实现，所以速度很快。ST20 系列处理器处理中断的硬件都有一个中断控制器用以接收中断信号，并将这一事件通知 CPU。在有些处理器中还有一个中断级别控制器，用以控制各个中断源的优先级。控制中断的硬件必须先初始化。中断服务程序需要先安装。

常用函数有：

interrupt_init_controller ()　　//配置中断控制器和中断级别控制器

interrupt_init ()　　//初始化每一个中断级

interrupt_install ()　　//安装中断服务程序

interrupt_enable()，interrupt_disable()　　//使能，禁止中断

interrupt_lock()，interrupt_unlock()　　//加解锁中断

8、管道

管道用来从一个任务向另一个任务发送数据，也可以用作任务间的同步。当一个任务发送数据而另一个任务接收数据时，最先想进行通信的任务将会进入等待状态直到另一个任务也通过此管道进行通信。这时数据会从发送任务的存储区复制到接收任务的存储区，然后两个任务才可以继续往下执行。

4.2 大卡底层驱动研究

DVB-CI 的底层主要是实现读写操作，以及对大卡初始化功能。

4.2.1 大卡初始化过程

在大卡初始化前，先要进行内存分区、操作系统、PIO 口的初始化。然后，就可以调用大卡初始化的入口函数 STPCCRD_Init（）。初始化时创建一个高优先级任务 cardInsertRemoveTask 来监测大卡的插拔状态变化。

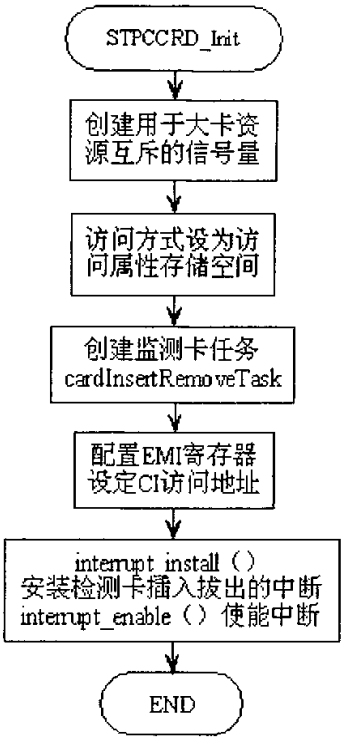

图 4-2 大卡初始化过程

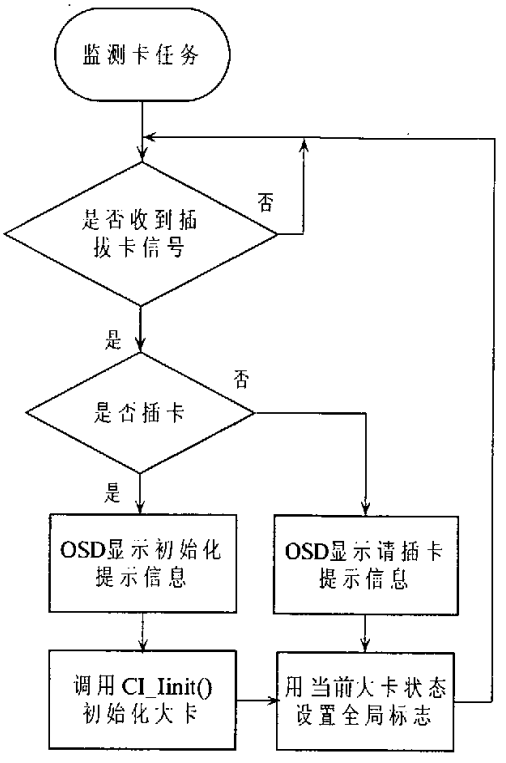

图 4-3 监测 CI 插拔卡状态任务流程

cardInsertRemoveTask 任务优先级设为 MAX_USER_PRIORITY＝15，并设置一个全局标志 cardstate 来表示大卡当前状态，cardstate＝1 表示大卡插入并已成功初始化，cardstate＝2 表示大卡插入但初始化失败，cardstate＝3 表示未插入大卡。

CI_Init（）主要完成三个步骤：

（1）读 CIS 信息

调用 STPCCRD_CheckCIS()，从大卡属性存储空间中地址 0 开始读取各 tuple，实验中读到的 tuple 依次是：DEVICE_OA 属性存储空间信息；DEVICE_OC 普通存储空间信息；VERS_1 通用的第一层版本信息；MANFID 厂商信息；CONFIG 配置信息；CFTABLE_ENTRY 配置表入口。

把通用版本信息和厂商信都保存下来，以便之后在 OSD 的 CA 信息菜单中查看。CONFIG 配置信息中，存有配置寄存器地址，还可以读到显示 DVB-CI 版本字符串,通常都是“DVB-CI-V1.00”。从 CFTABLE_ENTRY 进入配置信息列表，该列表有一系列选项组成，例如 Irdeto 大卡，读到的默认配置信息选项为：

配置选项索引号：09F

表 4-1 一组 PCMCIA 卡配置信息

电源配置信息	正常电压：5V； 最低电压：4.5V；最高电压：5.5 V； 1 秒平均最大电流：15mA； 10 毫秒平均最大电流：500mA；
I/O 地址空间配置信息	I/O 地址线数目：10 I/O 地址总线宽度：大卡只支持 8-Bit I/O 类型 I/O 起始地址：320h I/O 模块长度： 4h
IRQ 配置信息	支持脉冲模式中断； 不支持 CCR 和 Status 寄存器的中断信号共享； 中断掩码：ffffh；

（2）写配置选项寄存器

按照配置信息中的地址，将选择的配置选项索引号写入配置寄存器。

（3）协商缓冲区大小

先调用 task_delay()等待一个小的时间间隔，由访问属性地址空间转换为访问一般存储空间模式，然后调用 NegotiateBuffSize()，协商缓冲区大小。协商过程由主机发起，主机先在控制寄存器中将 SR 位置‘1’，然后大卡返回其缓冲区大小并将 DA 位置‘1’通知主机，主机读入该大小后将自己的缓冲区大小写入大卡，并将 SR 重新设为‘0,’协商完成。

4.2.2 底层读写功能实现

DVB-CI 命令接口通过使用 3 个总共占 4 字节地址的寄存器来实现硬件接口，寄存器的地址设定由主机设计者决定。

偏移量	寄存器
0	数据寄存器
1	命令/状态寄存器
2	长度寄存器（LS）
3	长度寄存器（MS）

图 4-4 CI 硬件接口寄存器

（1）数据寄存器。存放从模块读取到的或者要写入的数据；

（2）命令/状态寄存器。主机读数据时是状态寄存器，写入时是命令寄存器；

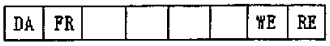

图 4-5 状态寄存器

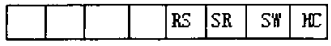

图 4-6 命令寄存器

当模块有数据要发送给主机时，设置 DA(Data Available)为 1；当模块空闲可以接收数据时，或者模块硬件复位或 RS 命令周期结束时，设置 FR（Free）为 1；WE(Write Error)和 RE(Read Error)用来表示读写操作时的长度错误。

RS（Reset）设为 1 表示对接口复位；SR（Size Read）设为 1 表示要求模块提供其最大缓冲区大小；SW（Size Write）设为 1 表示要求模块使用多大的缓冲区；HC(Host Control)由主机在启动数据写入序列前设为 1，SR，SW，HC 在数据传送完后都复位到 0；

（4）大小寄存器，16 位。

存储要读/写的数据的长度

综上，主机通过给 HC 位置 1 来取得向接口写入数据的控制权，而主机查询到 DA 位为 1 时，表示大卡要向主机发送数据。

写函数 STPCCRD_Write()和读函数 STPCCRD_Read()的流程如图：

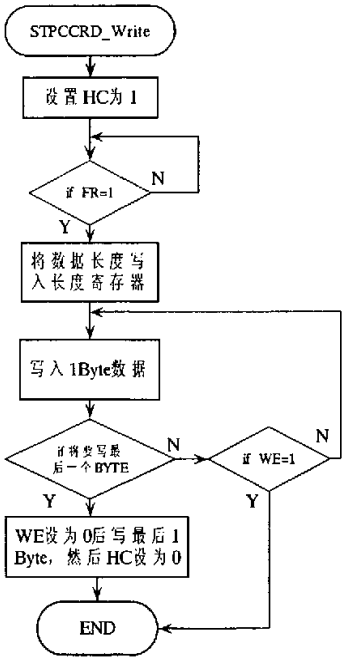

图 4-7 写数据流程

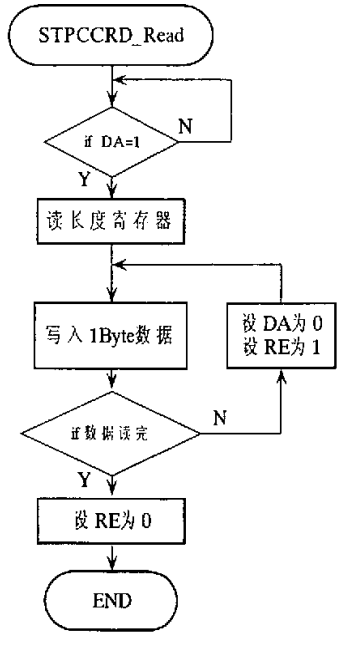

图 4-8 读数据流程

4.3 DVB-CI 协议栈实现

DVB-CI 是一个多层通信协议，可以参考 TCP/IP 协议来理解各层的作用。该协议为模块与主机提供了可靠而有效的通信方式，并且能使主机应用大卡实现解扰加密节目的基本功能以及一些其它功能。

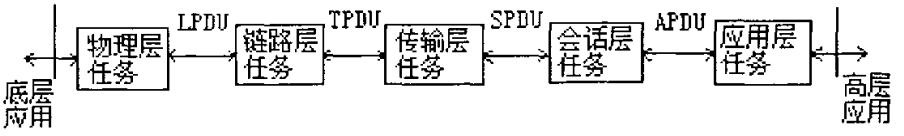

图 4.9 DVB-CI 协议栈任务模块层次示意图

创建 7 个任务分别实现物理层，链路层，传输层，会话层，应用层和定时器的功能，任务之间通过消息队列来通信。定时器任务的优先级设为最高，然后是物理层到应用层依次递减。由于 OS20 提供了无需拷贝数据的消息传递方式，因此能方便地用消息机制传递 PDU。消息类型除了 PDU 外，还定义了 CI 标准对象以外的一些控制信息。每个任务都是一个等待消息的无限循环，并且运用了状态机这一机制来实现对大量不同消息的处理。任务收到消息后，调用当前状态对应的处理函数，再在这个处理函数中根据收到消息的类型进行处理，并确定下一个状态。下面首先介绍有限状态机的原理。

4.3.1 有限状态机原理

有限状态机（FSM: Finite State Machine）是许多协议模型中使用的一个关键概念。利用这一技术，每个协议机器（即发送端或接收端）在每一时刻总是处于一个特别的状态。其状态由它的变量的所有的值构成[19]。

从每个状态，由 0 个或者多个可能到达其它可达状态的变迁（Transition）组成。当发生了事件时就会发生变迁。对于一个协议机而言，发送一个数据单元、接收一个数据单元、定时器超时等都可能引起变迁。已知协议机的完整描述，就由可能把所有状态作为节点，所有变迁作为方向弧，画出有向图来。

有一个特殊的状态被指定为初始状态。这个状态对应于系统开始运行时的情况，或者对应稍后某个方面的起始点。从初始状态经过一系列的变迁，系统可以达到某些或所有的状态。起初，所有的进程处于初始状态。然后开始发生事件，诸如有数据可供传递，或定时器超时。每个时间都可能引起一个进程采取行动，

从而转到新的状态。通过仔细分析每个状态的每个可能后继，就能构成可达性图。

构造状态图时要注意一些问题。例如，如果在某个状态有发生某个事件的可能，而有限状态机没有说明此时应该采取什么动作，那么此协议描述就有不完全的错误。再如存在某个状态集合，从这些状态没有出口，即使收到正确的数据，状态机也不作任何进展，那么该描述就犯了一个称为死锁的的错误。还有一种不太严重的错误，就是在协议描述中说明了在一种状态下如何控制一个事件，而这个事件永远都不可能发生，则称作多余的变迁。

可以使用以下的 C 语言伪代码来实现软件状态机。

```
Create Message Queue;
While(1){
     receive message from queue;
     swith(message){
          case 1: {action1; change state; break;}
          case 2: {action2; change state; break;}
          ......
          default: default action;
          }
     }
```

4.3.2 任务间通信服务模块

创建模块 ITC(Inter Task Communication)用来给上层任务提供通信机制以及定时器等服务功能，对上层协议栈屏蔽不同操作系统和底层硬件的特征，便于整个协议栈的移植。

ITC 模块的 API 接口函数有：

表 4-2 ITC 模块的底层 API 接口函数

功 能	函 数
初始化和结束	ITC_Init(); ITC_End();
等待消息	Physical_WaitEvt(); Sess_WaitEvt(); Trans_WaitEvt(); Link_WaitEvt(); Rm_WaitEvt(); Mmi_WaitEvt;
发送消息	ITC_MsgSend();
定时器控制	Timer_Init(); ITC_SetTimer(); ITC_KillTimer();
内存分配	ITC_Alloc(); ITC_Free(); ITC_DynAlloc(); ITC_DynReAlloc();
任务暂停	ITC_Sleep()

在调用底层初始化函数 STPCCRD_Init（）后，就可以调用 ITC_Init()来初始

化 CI 通信协议栈。ITC 的初始化中会创建一个 TimerTask 任务来实现定时器功能。以上初始化流程和定时器任务流程如图：

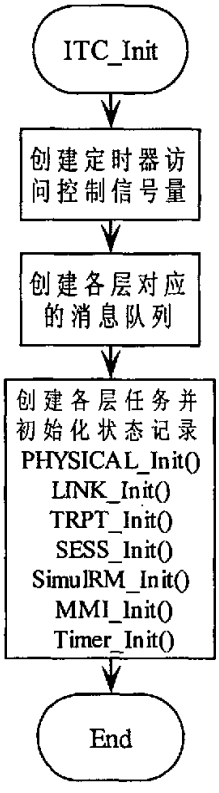

图 4-10 CI 通信协议栈初始化流程

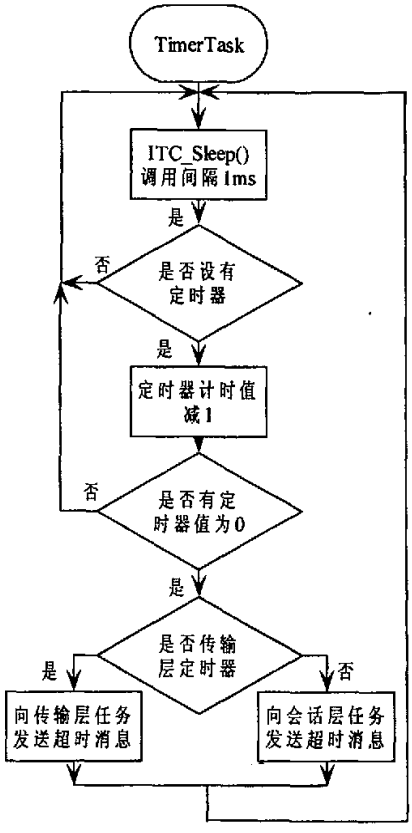

图 4-11 定时器任务流程

定时器主要用在与传输层和会话层通信时判断超时。通过一个数组 TIMER_t[]来记录 10 个定时器的状态。一旦调用 ITC_SetTimer()，就会从 TIMER_t[]中找到一个未使用的定时器，并设置定时时间和定时器状态。当检查到某个设定的定时器超时时，则向设定定时器的任务发送超时消息。对应任务收到消息后作相应处理，并用 ITC_KillTimer()删除该定时器设定。

4.3.3 物理层实现

物理层用到的全局量主要有 3 个：

（1）ph_infos，这个结构体用来存储物理层收到的或者要发送的消息；

（2）ph_tab_slot，用来记录大卡物理层的状态。

（3）cardstate，如前面提到的，用于接收监测卡任务传递来的大卡的状况：

表 4-3 卡的状态标志参数定义

cardstate 值	代表状况
0	没有大卡插拔动作
1	大卡刚插入，并且初始化正常
2	大卡刚插入，但初始化失败
3	大卡刚拔出

物理层任务主要有 3 个功能：

（1）处理上层传来的消息；

（2）根据大卡的插拔状态作相应处理；

（3）对大卡进行轮询读取数据；

要注意的是，由于物理层任务还要进行轮询工作，所以在等待消息时不能像其它层任务那样无限时等待，而是等待 10ms 后就超时。在没有收到数据时，该任务每 30ms 就查询一次大卡是否传来数据。

PHYS_DrvCard()用来处理大卡插拔动作，如果大卡刚插入，要向链路层发消息建立链路连接，然后才能进一步通信；

PHYSICAL_State()根据收到的消息和物理层当前连接状态向高层转发数据或者向大卡写数据，即将链路层发来的 LPDU 处理后写入大卡，或将从大卡读到的信息处理后发送给链路层。

物理层任务的执行流程如图 4-12 所示。

PHYSICAL_State()会根据 ph_infos 中存储的消息的类型来处理。

物理层收到的消息有 5 种类型：

表 4-4 物理层任务收到消息的类型

消息类型	传输方向
LPDU	Physical <-- Link
PH_DATA， PH_INSERT， PH_EXTRACT， PH_ERROR（底层读写出错）	Physical <-- Driver

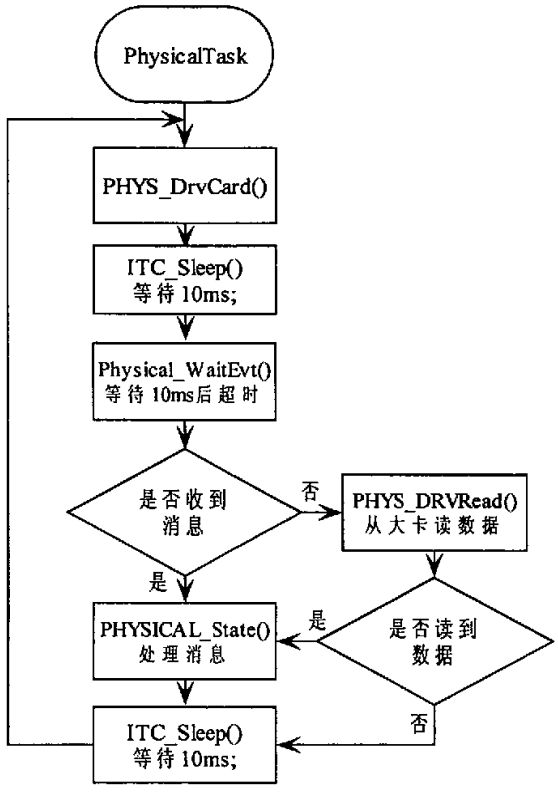

图 4-12 物理层任务流程

PHYSICAL_State()的处理流程如下图所示：

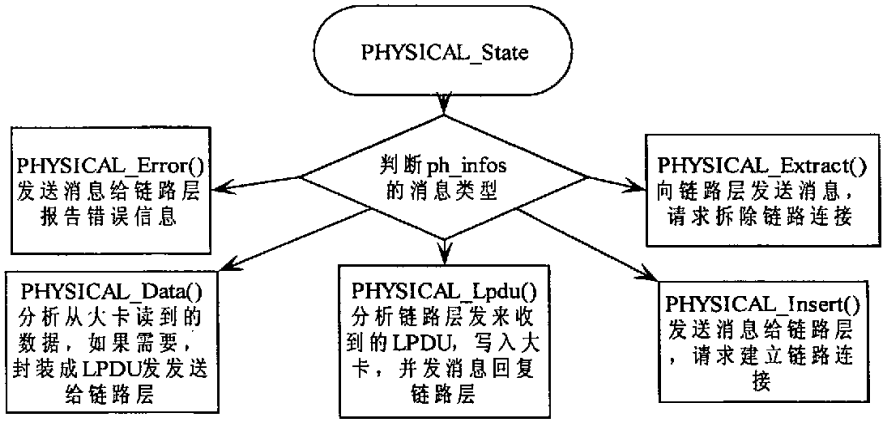

图 4-13 物理层处理消息流程

物理层有 2 种状态类型，记录在 ph_tab_slot 中：

- CONNECTED：物理层连接已经建立；
- IDLE：物理层连接未建立；

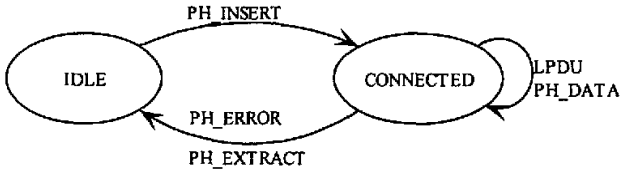

图 4-14　物理层状态转换图

4.3.4 链路层实现

链路层用到的全局量主要有：

（1）ln_infos，ln_infos1，存储收到或要发送的消息。需要 2 个变量，主要是为了在收到 TPDU 并解包成多个 LPDU 转发时，ln_infos1 用来发送数据，用 ln_infos 保存待发送的数据；

（2）ln_tab_slot，记录大卡链路层状态，缓冲区大小等参数。

（3）ln_tab_tcid[MAXTCID]，记录各链路连接状态；

（4）ln_tab_fifo[MAXTCID]，存储等待发送的 TPDU 的指针；

将传输层送来的 TPDU 封装成大小等于协商好的缓冲区大小的 LPDU 转发给物理层，并处理物理层发来的 LPDU。链路层向物理层转发数据时，每发送一个 LPDU 后，等待收到确认回复后再接着发下一个 LPDU。链路层收到下层发来的 LPDU 时，将若干个 LPDU 拼接成一个完整的 TPDU，然后选择对应的传输层连接号，发送给传输层任务。同时，还向传输层发送建立或断开传输连接的消息。

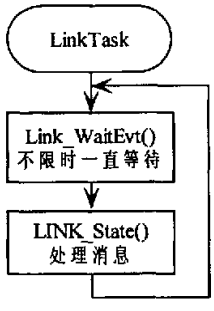

图 4-15　链路层任务流程

LINK_State()根据 ln_infos 中的消息来处理，收到的消息有 6 种类型：

表 4-5 链路层任务收到消息的类型

消息类型	传输方向
TPDU	Link <--> Trans
LPDU	Link <--> Physical
LN_ACK 物理层确认信息 LN_CONNECT 请求建立链路连接 LN_ERROR 物理层出错信息 LN_DISCONNECT 拆除链路连接	Link <--- Physical

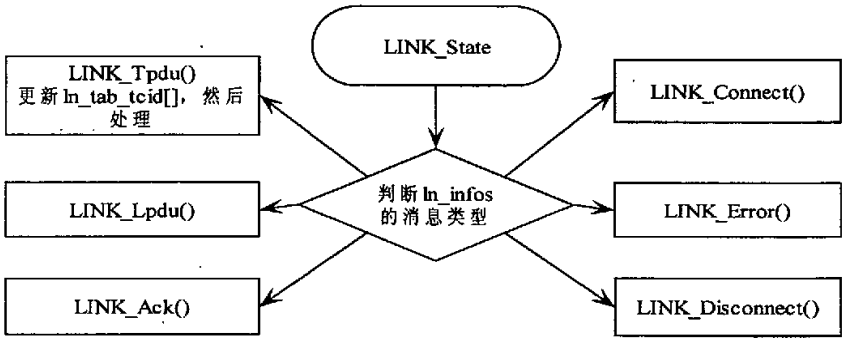

图 4-16 链路层处理消息流程

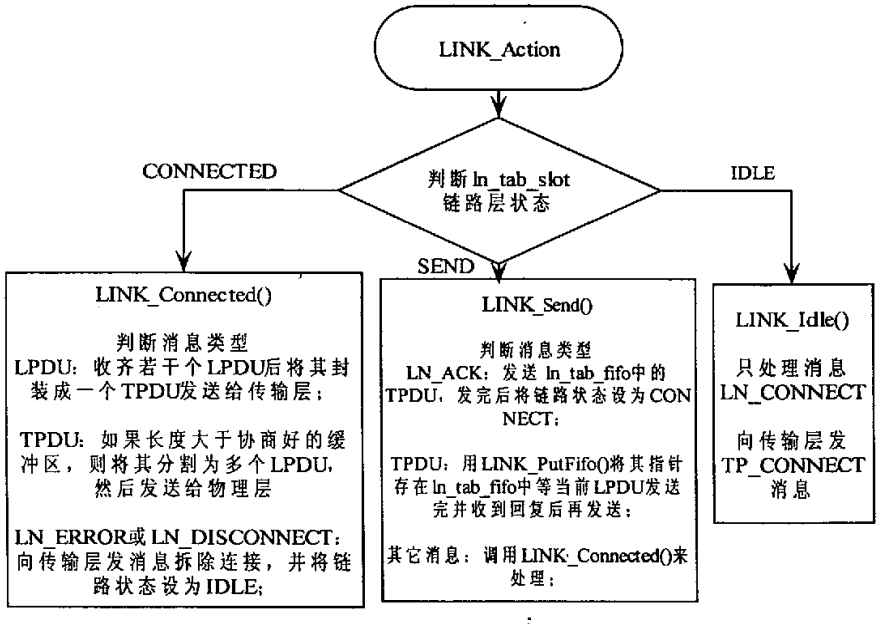

图 4-17 链路层按状态处理消息具体流程

链路层的 3 种状态的转换关系为：

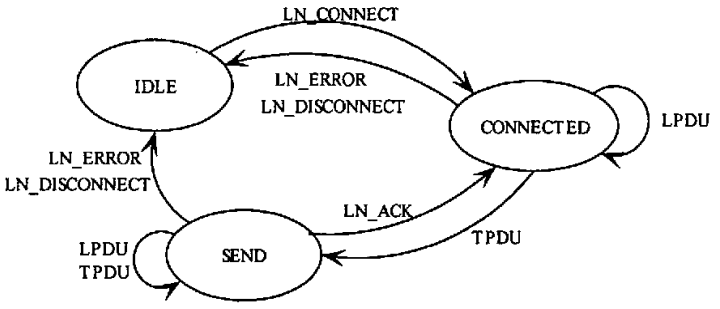

图 4-18　链路层状态转换图

4.3.5 传输层实现

传输层用到的全局量主要有：

（1）t_infos，t_infos1，存储收到或要发送的消息；

（2）t_tab_tcid[MAXTCID]，记录各传输连接状态；

t_tab_fifo[MAXTCID]，存储各传输连接待发送的 SPDU 指针；

传输层任务接收链路层发送来的 TPDU，根据其报头信息作出不同的处理，例如建立或关闭传输层连接；或者是在收到大卡发来的 T_SB 后回复 T_RCV，同意大卡发送数据；还有收到数据后向上层转发。传输层的另一个功能是将上层发来的 SPDU 封装成 TPDU，通过对应的传输层连接转发到链路层。

数据的传输的可靠性由链路层保证，传输层主要进行传输控制，保证数据按顺序，无重复地传输。

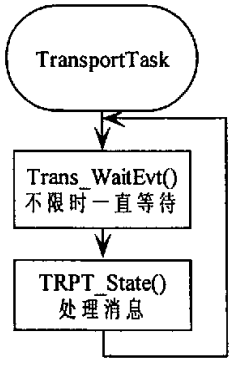

图 4-19　传输层任务流程

TRPT_State()会根据 t_infos 中的消息来处理，传输层收到的消息有 6 种类型：

表 4-6 传输层任务收到消息的类型

消息类型	传输方向
TPDU 建立连接：create_t_c，c_t_c_reply， request_t_c，new_t_c 拆除连接：delete_t_c，d_t_c_reply 传输数据：t_data_more，t_data_last， t_rcv 轮询：t_poll，t_sb	Link <--> Transport
SPDU	Transport <--> Session
TP_CONNECT 链路层请求建立传输连接 TP_DISCONNECT 链路层请求拆除传输连接	Link ---> Transport
SS_DELETE_TC 会话层请求拆除传输连接	Transport <--- Session
TIMER 定时器超时	Link --->Transport <--- Session

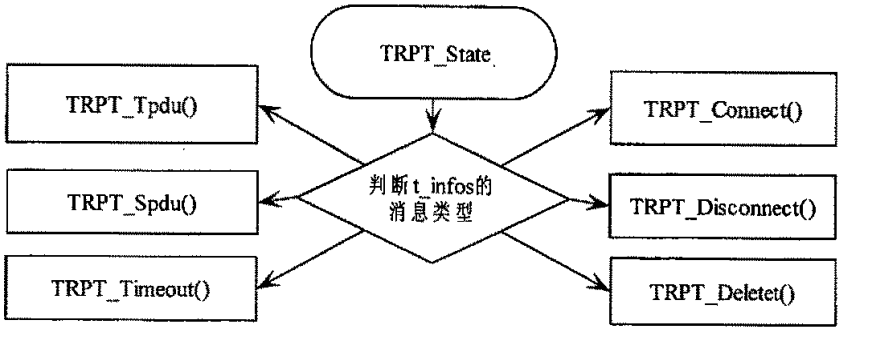

图 4-20 传输层处理消息流程

其中，TRPT_Tpdu()，TRPT_Spdu()，TRPT_Timeout()，TRPT_Connect()，TRPT_Disconnect()，TRPT_Delete ()最后都会调用函数 TRPT_Action()根据传输连接当前状态来处理消息。

由于传输层采用命令响应方式传输，底层不能主动发送数据，只能等待上层查询或者收到上层数据后应答，所以其状态相对链路层更为复杂。

按照 DVB-CI 传输层协议，传输层主机方面有 4 个状态：

- IDLE ：未建立传输连接；
- CREATION： 正在建立传输连接；
- CONNECTED ：传输连接已经建立；
- DELETION ：正在拆除传输连接；

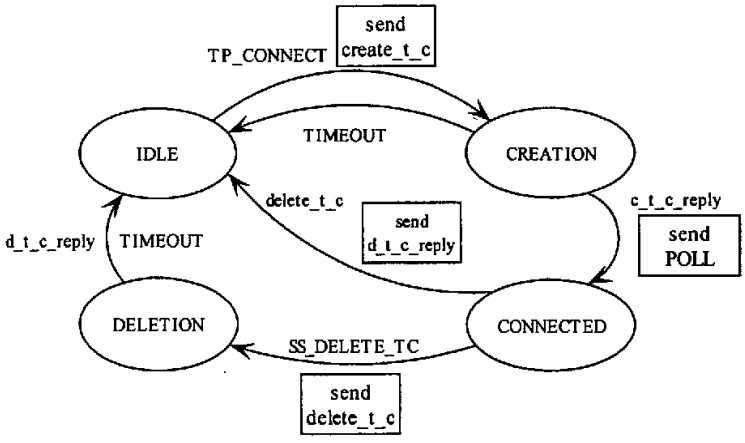

图 4-21　传输层状态转换图

为了方便处理，将 CONNECTED 状态再细分为 3 个状态：

- POLL：发送 TPDU 或轮询后等待链路层回复；
- ACTIVE：没有数据要发送或者接收；
- RCEPTION：等待从链路层接收有效数据；

这 3 个状态间的内部关系如下：

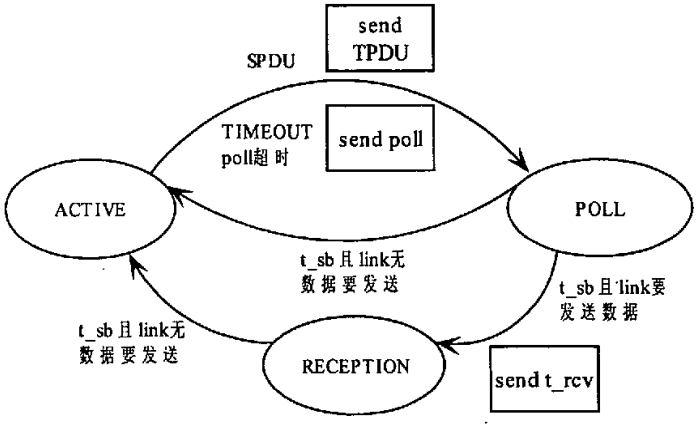

图 4-22　传输层传输数据时状态转换图

4.3.6 会话层实现

会话层用到的全局量主要有：

（1）s_infos，s_infos1，存储收到或要发送的消息；

（2）s_tab_ssnb [ ]，记录各会话连接状态；

会话层任务接收传输层发来的 SPDU，处理建立或关闭会话层连接的请求，或者是将大卡发来的数据封装成 APDU 转发到应用层；将上层发来的 APDU 封装成 SPDU，通过对应的会话层连接转发到传输层，或者是处理上层传来的其它控制信息。

会话层主要是为高层应用建立不同的会话连接，以便使用各种资源。例如某些资源可以同时被多个应用使用，这种情况下各应用分别与资源建立连接，各连接用不同的会话连接号区分。

会话层任务流程与传输层任务类似，也有一个函数 SESS_State（）来处理收到的消息。会话层收到的消息有 6 种：

表 4-7 会话层任务收到消息的类型

消息类型	传输方向
SPDU 建立连接：open_session_request，open_session_response create_session，create_session_response 拆除连接：close_session_request，close_session_response 传输数据：session_number	Trans<--> Session
APDU	Session <--> App
SS_TC_CREATED 传输层报告传输连接建立 SS_TC_DELETED 传输层报告传输连接拆除	Trans ---> Session
RM_DELETE_TC 应用层请求拆除传输连接 RM_OPEN_SS_RSP 应用层回复会话连接建立 RM_CLOSE_SS 应用层回复会话连接拆除	Session <--- App
TIMEOUT 定时器超时	Trans--> Session

当一个应用要使用一个资源时，就会用对应的资源号向主机发送 open_session_request 请求建立会话连接，主机用 open_session_response 回复连接是否建立，以及新建会话连接的连接号。根据资源是主机提供还由另一个模块提供，与资源建立连接和拆除连接分为两种情况，如图：

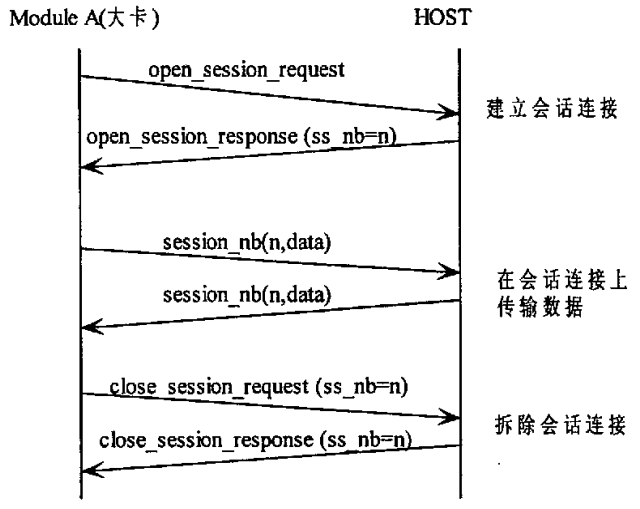

图 4-23　大卡与主机上的资源的会话过程

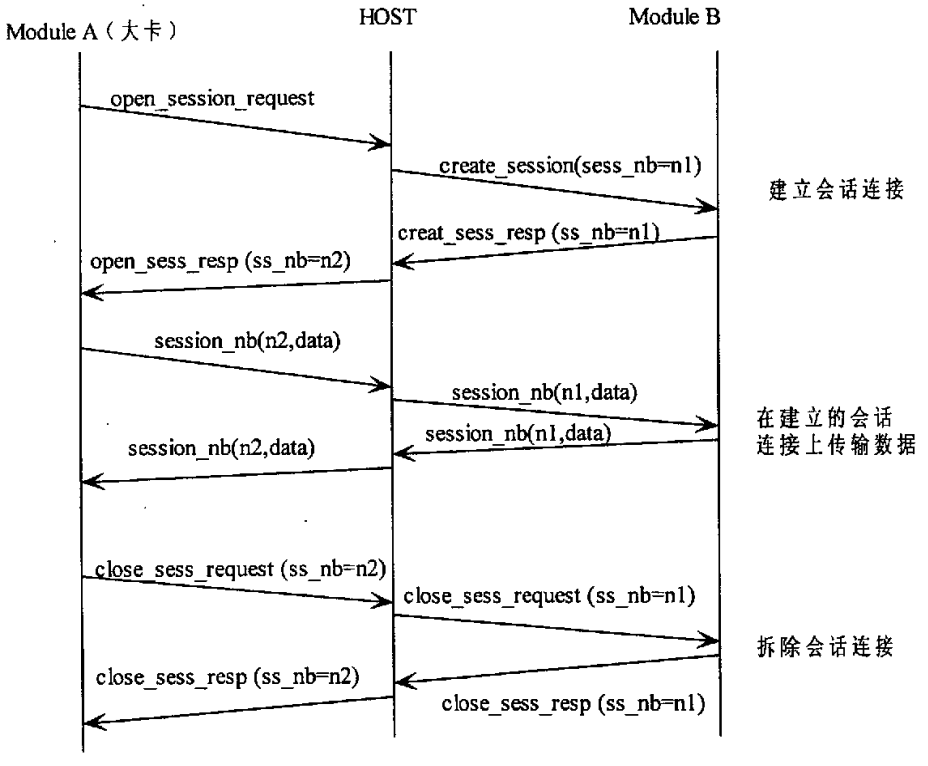

图 4-24　大卡与其它模块上的资源的会话过程

一个会话连接有 7 种状态：

- IDLE： 会话连接未建立；
- OPEN：正在建立会话连接；
- CREATION：正在建立主机与其它模块的会话连接，等待其它模块回复；
- ACTIVE：会话连接已经建立；
- CLOSE：正在拆除与主机上资源的会话连接；
- DELETION： 正在拆除与模块 B 的会话连接；
- RELEASE：主机正在与模块 B 建立连接时模块 A 的传输层连接拆除；

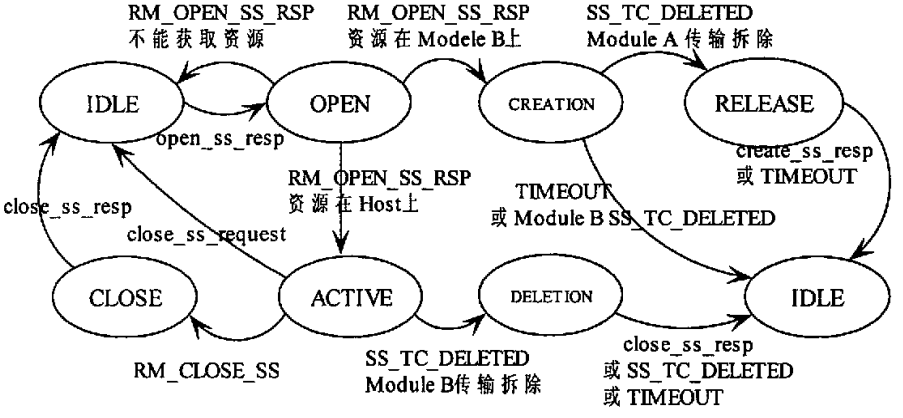

图 4-25 一个会话连接的状态转换图

4.3.7 应用层实现

应用层中最基本的资源有 3 个：

（1）Resource Manager 资源。大卡初始化后会首先请求与 Resource Manager 建立会话连接，并且会一直保持这个连接，以便通过 Resource Manager 来使用其它资源。

（2）Application Information 资源。大卡与 Resource Manager 建立连接并交换完信息后，会请求与 Application Information 资源建立连接，以便主机获取大卡的具体信息，如 CA 系统号。

（3）CA Surpport 资源。大卡还会请求与 CA Surpport 建立连接，使用大卡解密加密节目就是在这一连接上进行。

本软件模块提供的其它可选资源还有：

（1）Date and Time 资源：向大卡提供当前日期时间信息，该信息通常由 DVB 业务信息中的 TDT 表得到。

（2）Man-Machine-Interface 资源。提供人机交互接口，在高层软件的支持下，可以在 OSD 菜单下显示大卡发来的提示信息，并通过按键接收用户响应。

这里通过创建一个任务 RM 来实现除 MMI 之外的应用层功能，另外单独创建一个任务 MMI 来实现 MMI 功能。

4.3.7.1 RM 任务

RM 任务主要是维护一个资源列表，实现有关 Resource Manager，Application Info，CA Support，DateTime 资源的功能，并将有关 MMI 的消息转发给 MMI 任务。大卡插上后，会请求与 RM 建立会话连接，然后 RM 会和大卡通过互相发送 profile_enq 和 profile_change 来交换信息。当主机和大卡都得到新的资源列表后，大卡会请求与 AI 建立会话连接，然后主机用 application_info_enq 向大卡询问其应用信息，大卡用 application_info 来回复。大卡还可以通过与 AI 的会话连接，发送 enter_menu 对象来启动 MMI 会话。在大卡与 CA 的连接上，主机用 ca_info_enq 向大卡查询，大卡用 ca_info 返回其 ca_system_id。这一连接非常重要，当主机要用大卡解扰一个加密节目时，就是在这个连接上向大卡发送对应节目的 ca_pmt 对象，并可以选择让大卡返回解扰信息。

RM 任务用到的全局量主要有：

（1）rm_infos，存储收到或要发送的消息；

（2）TabRess[ ]，记录各资源的会话连接状态；

（3）GL_TabTcid[ ]，记录各传输连接状态；

（4）GL_TabSsnb[ ]，记录各会话连接状态；

RM 任务流程与前面的任务类似，循环用函数 Rm_WaitEvt（）接收消息，然后用函数 RM_State（）来处理。RM 收到的消息类型有：

表 4-8　RM 收到的消息类型

消息类型	传输方向
APDU	Session <--> RM <-->MMI
RM_OPEN_SS_REQ 请求建立会话连接 RM_SS_CLOSED　报告会话连接建立 RM_TC_CREATED 报告传输连接建立 RM_TC_DELETED 报告传输连接拆除 TIMEOUT 定时器超时	Session --> RM

RM_State（）的处理过程如下：

{

switch(消息类型)

{

case RM_OPEN_SS_REQ：

switch(请求资源类型)

{

case Resource Manager：

case Application Info：

case Date Time：

向会话层发消息回复；break；

case MMI：

向会话层发消息回复，并调用 RM_Send_MsgtoMMI（）向 MMI 转发消息，break；

};

break；

case RM_SS_CLOSED：更新 TabRess[ ]和 GL_TabSsnb[ ]，如果是 MMI 会话关闭，调用 RM_Send_MsgtoMMI（）向 MMI 转发消息；

break；

case RM_TC_DELETED：

case RM_TC_CREATED：更新 GL_TabTcid[ ]；

break；

case APDU：

switch(APDU tag 类型)

{

case PROFILE_ENQ：回复包含资源列表的 Profile_Reply，break；

case PROFILE：回复 CA_Info_Enquiry，break；

case PROFILE_CHANGE：回复 Profile_enq，break；

case APPLICATION_INFO：记录 CA 信息，break；

case CA_INFO：发送当前节目的 CA_PMT，break；

case DATE_TIME_ENQ：用 Date_Time 当前回复日期时间，break；

case 与 MMI 有关的 APDU: RM_Send_MsgtoMMI( )向 MMI 转发, break;

}; break;

}

}

4.3.8.2 MMI 任务

MMI 任务主要是处理 RM 转发来的有关 MMI 的消息，然后回复消息给会话层任务。另外由于在目前实际应用中，都采用高级模式的 MMI，所以本模块不处理低级模式的 MMI。MMI 信息与用户的交互实现将在下一章中具体介绍。

MMI 任务用到的全局量主要有：

（1）m_infos，存储收到或要发送的消息；

（2）mmi_ssnb，记录 MMI 会话连接的连接号；

（3）MMI_buf_in，MMI_buf_out[ ]，记录收到或要发送的数据；

RM 任务流程与前面的任务类似，在循环用函数 Rm_WaitEvt（）接收消息，然后用函数 MMI_State（）来处理。

MMI 只需要处理 3 种类型的消息：APDU，RM_OPEN_SS_REQ，RM_SS_CLOSED

MMI_State（）的处理过程为：

```
{
switch(消息类型)
{
case RM_OPEN_SS_REQ:    更新 mmi_ssnb，break;
case RM_SS_CLOSED:      判断连接号一致则重新初始化 MMI 有关变量，释
                        放 MMI 消息内存，break;
case APDU:
switch(APDU tag 类型)
    {
    case Close_Mmi:          延时一定时间后关闭 MMI 连接，break;
    case Display_Control:    设置 MMI 模式，回复 Display_Reply，break;
    case Enq:                询问用户用 Answ 返回用户的响应，break;
    case Text_More，Text_Last:        显示提示信息，break;
    case List_More，List_Last:        显示列表，break;
```

```
        case Menu_More, Menu_Last:        显示菜单，用 Menu_Answ 返回用户
                                          选项 break;
        }; break;
    }
    }
```

4.4 通信过程测试结果

以下是大卡与主机协商好缓冲区后，从建立传输连接到大卡与 CA 资源建立连接的过程。下表第一列表示数据在至上而下各层封装的类型。链路层都采用 data_last 类型封装，在表中就不再列出。

- 灰色底纹部分表示大卡发送的数据；
- 无底纹部分表示主机发送给大卡的数据；

表 4-9 大卡与主机通信过程

应用层—会话层—传输层 PDU	SPDU, APDU 内容分析
tpdu create_t_c, t_c_id = 1	
tpdu c_t_c_reply, t_c_id = 1, no data available	
tpdu SB, data available	
tpdu RCV	
spdu open_session_request tpdu data_last, no data available	ResourceId = 0x00010041 (Resource Mnager)
spdu open_session_response, SessNb = 1 tpdu data_last	Session opened successful 开始与 Resource Manager 会话
apdu profile_enq to Resource Manager spdu session_number, SessNb = 1(Resource Manager) tpdu data_last	
tpdu SB, data available	
tpdu RCV	
apdu profile_reply to Resource Manager spdu session_number, SessNb = 1(Resource Manager) tpdu data_last, no data available	

apdu profile_change spdu session_number, SessNb = 1(Resource Manager) tpdu data_last	
tpdu SB, data available	
tpdu RCV	
apdu profile_enq to Resource Manager spdu session_number, SessionNb = 1(Resource Manager) data_last, no data available	
apdu profile_reply spdu session_number, SessNb = 1(Resource Manager) tpdu data_last	resource_list 0 resource_id = 0x10041 RM 1 resource_id = 0x20041 AI 2 resource_id = 0x30041 CA 3 resource_id = 0x240041 DT 4 resource_id = 0x400041 MMI
tpdu SB, data available	
tpdu RCV	
open_session_request tpdu data_last, no data available	ResourceId = 0x00020041 (Application Information)
spdu open_session_response, SessiNb = 2 tpdu data_last	Session opened successful 开始与 Application Info 会话
apdu application_info_enq spdu session_number, SessionNb = 2(Application Info) tpdu data_last	
tpdu SB, data available	
tpdu RCV	
apdu application_info spdu session_number, SessNb = 2(Application Info) tpdu data_last, no data available	appl_type=Conditional Access appl_manufacturer = 0x500 manufacturer_code = 0x500 menu_string = "Viaccess"
tpdu SB, t_c_id = 1, data available	
tpdu RCV	
spdu open_session_request tpdu data_last	ResourceId = 0x00030041 (CA_support)

spdu open_session_response tpdu data_last	Session opened successful 开始与CA support会话
apdu ca_info_enq spdu session_number, SessNb = 3(CA support) tpdu data_last	
tpdu SB, data available	
tpdu RCV	
apdu ca_info spdu session_number, SessNb = 3(CA support)t tpdu data_last, no data available	ca_system_id =>0x0602
以下省略	

第五章 iDTV 中大卡功能的应用及相关高层实现

大卡的主要功能是同配对的小卡一起完成对加密节目的解密（包括对密钥的解密和对传送流的解扰）。在大卡与主机通过 DVB-CI 接口建立连接以后，就可以通过发 CA_PMT 给大卡来实现解密功能。同时，大卡还需要主机上一些功能模块的支持，例如给大卡提供日期时间，通过图形用户界面（GUI）实现 MMI 等。

本章首先介绍了 DTM 主应用中使用大卡解密的过程，重点分析生成 CA_PMT 的方法。然后介绍了主应用中对大卡有关功能的支持，具体分析了有关 SI 信息的接收，以及图形用户界面在嵌入式系统上的实现。

5.1 大卡解密功能的应用与实现

CA 模块即大卡部分是数字电视解码系统的一个重要组成部分，首先对整个系统作一个简要的介绍。

5.1.1 数字电视解码系统

数字电视解码系统（机顶盒就是其典型应用）通常被设计成许多功能相对独立的模块。这些模块可以分为高频头、信道解调器、解复用器、主机编程接口（主控制器）、音频解码器、视频解码器、OSD 显示控制、视频编码器等。这些模块与解码芯片紧密相关。除此之外，还有一些与 MPEG 码流的解码处理无关的简单模块，如：红外遥控、面板按键控制、LCD 显示、串行通讯端口等。软件通过操作一些硬件提供的寄存器，来控制整个系统。

由于整个数字电视解码系统相当复杂，芯片厂商为应用开发人员提供了基本的系统解决方案，包括实时操作系统和底层驱动以及基本的主应用，使应用开发人员能把精力集中在针对具体应用的程序设计上。

实时操作系统只与主处理器和硬件时钟直接相关，对上层软件来说，其作用就是提供一些内核对象（即 API 接口函数）实现如存储管理、任务管理等功能。这些 API 也能供驱动层软件使用。设备驱动为应用层软件提供完整的硬件功能函数接口。驱动程序在初始化时除了对硬件寄存器赋初始值外，还利用实时操作系统创建一些系统内核对象，如任务、信号量、消息。系统运行时，应用程序不断

处理来自底层的各种消息，并使用相对简洁的驱动接口函数去控制硬件。

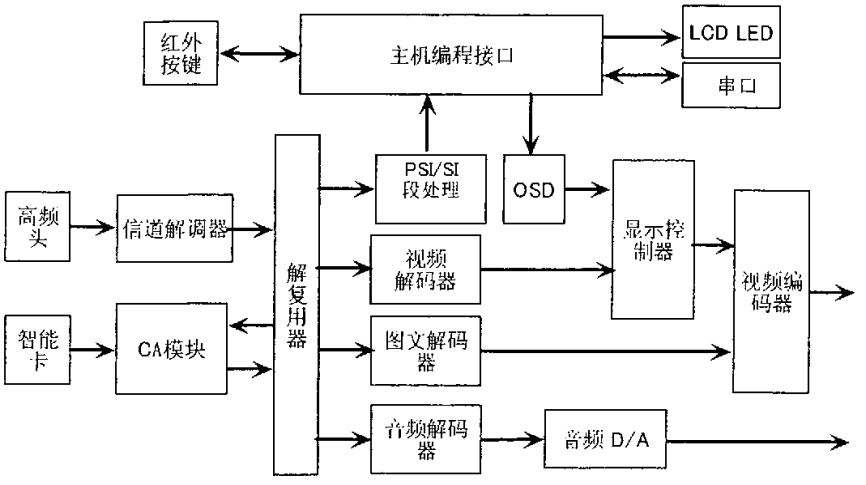

图 5-1 数字电视解码系统

5.1.2 使用大卡解密的应用过程

在大卡正常工作状态下使用大卡解密，关键是在切换到加密节目时发送正确的 CA_PMT 给大卡。

DTM 的主应用使用大卡解密的流程如下图所示：

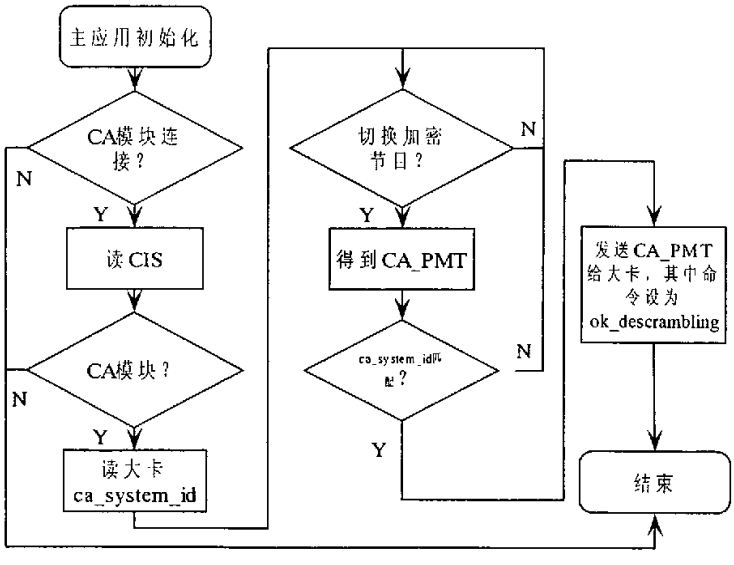

图 5-2 主应用中大卡解密流程

5.1.3 CA_PMT 的生成与使用

PMT（Program Map Table）节目映射表与一套节目对应，由 PMT 可以知道该路节目由哪些流构成和这些流的类型（视频，音频，数据），各流对应的 PID，以及该节目的 PCR 所对应的 PID，还有流或节目对应的 CA 信息。

表 5-1 PMT 段与 CA_PMT 语法定义

Pmt_section( )	CA_PMT( )

{table_id
section_sytax_indicator
section_length
program_number
version_number
current_next_indicator
section_number
last_section_number
pcr_PID
Program_info_length
for(I=0;I<N;I++){
descriptor() // program level
}
for(I=0;I<N;I++){
stream_type
elementary_PID
ES_info_length
for(I=0;I<N2;I++){
descriptor() // ES level
}
}
CRC_32
}
	{ca_pmt_tag
	length_field()
	ca_pmt_list_management
	program_number
	version_number
	current_next_indicator
	program_info_length
	if (program_info_length != 0) {
	ca_pmt_cmd_id // program level
	for (i=0; i<n; i++) {
	CA_descriptor() // program level
	}
	}
	for (i=0; i<n; i++) {
	stream_type
	elementary_PID
	ES_info_length
	if (ES_info_length != 0) {
	ca_pmt_cmd_id // ES level
	for (i=0; i<n; i++) {
	CA_descriptor() // ES level
	}
	}
	}

比较 PMT 与 CA_PMT 可见，生成 CA_PMT 时，需要过滤掉 CA 描述信息以外的信息，并且正确设置 ca_pmt_list_management 和 ca_pmt_cmd_id 这两个参数。

（1）ca_pmt_list_management。表示用户是选择解密一个节目还是多个节目。通常设置为‘3’即‘only’命令，表示只选中并解密一个节目。如果一个节目的 PMT 中的 version_number 或 current_next_indicator 改变，说明节目信息发生了变化，则要发送‘update’命令给大卡，以保证能继续正常解密。

（2）ca_pmt_cmd_id。用于指示大卡如何响应 CA_PMT。通常设置为‘1’即‘ok_descrabling’命令，表示大卡可立即开始解密而无需回复任何信息。另外，也可以 把该参数设为‘query’，要求大卡用 ca_pmt_reply 对象来回复能否解密即其原因；还可以设为‘ok_mmi’来要求大卡建立 MMI 会话。要注意的是，发送‘query’和‘ok_mmi’命令的 CA_PMT 后，大卡不会立即开始解密；直到收到‘ok_descrabling’的 CA_PMT，大卡才开始解密节目。

还要注意的是，条件接收系统可以在节目级加密，也可以在基本码流（Elementary Stream）级加密，即视频流和音频流分别加密。如果这两个等级的 CA 描述信息在 CA_PMT 中同时存在，则只有基本码流级的 CA 信息会起作用。

在应用中，每次收到一个加密节目的 PMT，就按照规范生成 CA_PMT 并存储在一个链表中。CA_PMT 的参数设为 ca_pmt_list_management＝3，ca_pmt_cmd_id ＝1。切换到加密节目时，从链表中找出对应的 CA_PMT 通过上一章讲到的 CI 协议栈发送给大卡，就可以解密节目了。CI 协议栈主要用到的高层软件的用户图形接口和从 DVB 业务信息中获取日期时间这两个功能。

5.2 其它软件模块对大卡的支持

5.2.1 从 TDT 表获取日期时间信息

5.2.1.1 TDT 表的分析

业务信息中的 TDT 表携带的是当前时间的信息，有效信息是 5 个 Byte，包括 2 个 Byte 的 MJD （修正的儒略时间）日期和 3 个 Byte 的 BCD 编码的小时：分：秒 UTC(标准时间坐标)时间信息。例如 93/10/13 12:45:00 编码成 MJD 和 UTC 在 TDT 表中表示为 0xC0 79 12 45 00。

由 MJD 的 2 个 Byte 不仅可以计算出年、月、日、星期几，还可以计算出从 1900 年算起的星期数。MJD+UTC 加上（经度在格林威治以西的本地偏移量为减）偏移量等于本地 MJD + 本地 time。计算时间加减法时，要注意进位和借位跟一般加减法不同，最后要得到符合人们习惯的时间表达，例如 99/05/01 25:00:00 这样的表达就不正确。

TDT 中的时间是格林尼治标准时间，与北京时间相差 8 小时。而 TDT 表后面紧跟的 TOT 表中，给出了本地时间与标准时间的偏移量。由于国内都采用北京时间，因此在本应用中不分析 TOT 表，使用 8 小时的固定时间偏移量。

MJD 由年(Y)，月(M)，日(D)生成的规则如下：

MJD = 14956 + D + int((Y - L) * 365.25) + int((M + 1 +L * 12) * 30.6001)

其中，Y 是相对 1900 的年数，若 M=1 或 2 则 L=1；否则 L=0。

程序中计算年月日的代码如下：

```
year = (MJD*100 - 1507820)/36525;
temp = year*36525/100;
month = (temp*10 - 149561 - temp*10)/306.001;
day = MJD - 14956 - temp - (month*306001/10000);
if((month == 14) || (month == 15)){
    year++;
    month = month -1 - 12;
}
else
    month--;
year += 1900;
```

根据计算出的标准日期，再加上时分秒以及 8 小时的偏移时间，就能得到本地的当前准确时间。

5.2.1.2 TDT 表的获取

如图 5－1 所示，高频头输出模拟信号，经过信道解调器解调（卫星信号采用 QPSK 方式解调，有线信号则采用 QAM 方式解调），将基带的 MPEG-2 传输流输出至解复用器中。

解复用器是一个相当复杂的模块，它接受一路（即一个频率点）包含了多个节目的基带 TS 流输入。解复用器完成传输流中各种信息的提取，送至相应的缓存区中等待处理。这些信息包括：视频、音频、图文、字幕、MPEG2-PSI 表及 DVB-SI 表等。

当传输流到达时，解复用器利用硬件搜索引擎，根据 PID 对传输包进行过滤。如果当前传输包不与任何一个包过滤器的 PID 相匹配，则丢弃当前包并等待下一个包的到来。如果当前传输包匹配成功，解复用器就把包存放到缓存区，并根据包的类型决定是否需要进一步处理。如：音视频是否需要进行解扰处理、PSI / SI 表信息是否需要段过滤（section filtering）处理等。

ST QAMi5516 数字电视解码芯片提供的 PTI 模块能同时过滤 48 个不同 PID 的 TS 分组和 32 个 Section 分组，且可以实现 Section 分组的硬件 CRC 校验，自动丢弃 CRC 校验出错的分组。PIT 过滤器分为两个级别，第一级通过 PID 来匹配，第二级根据用户的设置来进行不同的匹配。

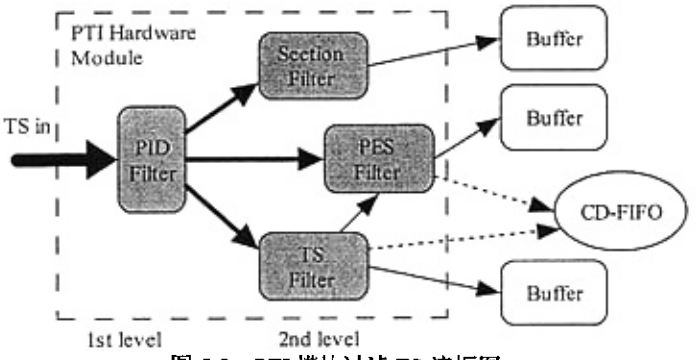

图 5-3 PTI 模块过滤 TS 流框图

PTI 过滤器可以设置的过滤方式有：Section，PES，RAW，PCR，EMM，ECM。过滤 SI 信息一般都使用 Section 方式，对 Section 的 PID、TableId、ProgramNumber、SectionNumber、VersionNumber 进行过滤；对视频、音频的过滤采用 PES 方式。

TDT 表的接收与其它 SI 表有所不同。

首先，TDT 跟其它 DVB-SI 规范中定义的表的语法有所不同。一般的业务信息表如 SDT、NIT、EIT 的结尾都有 32 位的 CRC 校验码，Section 过滤器会自动对其进行校验，丢弃校验出错的 Section。而 TDT 表不是以 CRC 校验码结束，而是紧跟的 TOT（偏移时间表），TOT 表的结尾处才是 CRC 校验码。因而如果采用自动校验 CRC 的 Section 过滤器，会因为校验失败而不能收到 TDT 表。所以在应用中将 PTI 过滤器设为 RAW 方式，即只通过 PID（TDT 的 PID 为 0x0014）来过滤得到原始的 TS 包。以下就是一个通过 RAW 过滤器得到的 TS 包中的 TDT 数据：

0x47 0x40 0x14 0x10 0x00 0x70 0x70 0x05 0xd1 0x10 0x10 0x47 0x30 …

同步字节 PID Table_Id Length MJD UTC:10 时 47 分 30 秒

其次，TDT 表更新较快（DVB 规范中规定 TDT 表的发送周期为 25 毫秒～30 秒），要随时保持接收。而其它的 SI 表一般是收齐一次后就关闭过滤器不再接收。在应用程序中，创建一个任务来接收包括 TDT 的 SI 信息，如图：

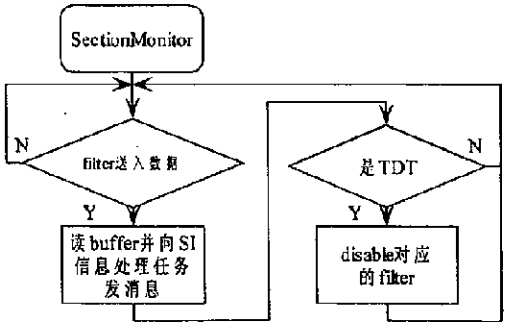

图 5-4 接收 SI 信息的任务流程

5.2.2 MMI 用户界面的研究与实现

上一章中的 DVB-CI 协议栈实现了大卡与主机的 MMI 信息交换，主机如何处理收到的 MMI 信息需要上层的应用软件来实现。MMI 对象虽然比较简单，主要就是一些字符信息的显示，以及一些简单的字符信息的回复。但在上层实现时，并不仅仅要在 OSD（On-Screen-Display）上显示这些信息，还要考虑到这些显示信息与图形用户接口模块的统一。

5.2.2.1 数字电视解码器上的用户接口

用户接口模块是整个系统中唯一一个具有全局控制功能的任务，它为用户提供了一个界面，用户通过它可以控制解码器的各个模块。在 iDTV 项目中，采用一个叫做‘Usif’的低优先级任务来实现该模块的功能。Usif 任务采用消息队列接收来自各个模块的消息。它从键盘接收用户的请求，监视数据库和调谐器的状态，按用户的要求指示相应的模块进行相应的操作，在屏幕上开创一个 OSD 区域用来显示各种图形、菜单、消息等。

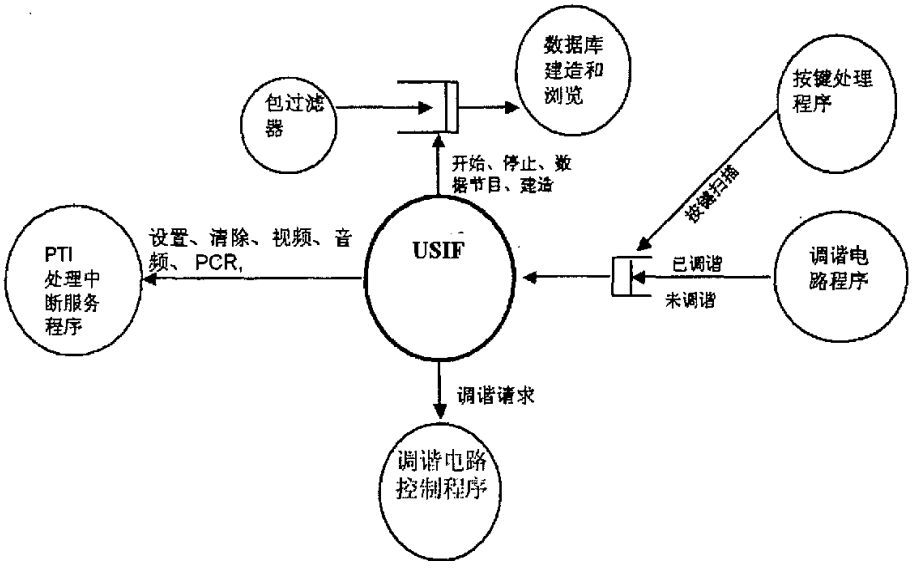

图 5-5　用户接口任务 Usif 与其它模块关系图

这里以一次成功的自动搜索节目为例，分析 3 个阶段中 Usif 执行的步骤，说明用户接口模块协同其它模块工作过程。

（1）进入搜索：用户接口接收用户的按键进入自动搜索菜单，收到 OK 按键后，向调谐器发消息请求锁定频点；

（2）搜索中：调谐器锁定频点后向 Usif 报告已锁定，然后 Usif 向节目数据库任务 Dbase 发消息，开始搜索节目；Usif 还会根据收到的消息不断刷新菜单的显示。

（3）搜索完成：收到 Dbase 发来的搜索完成的消息后，Usif 把搜索到的节目信息显示出来，并等待用户按键，根据按键来切换节目或者推出菜单。

5.2.2.2 数字电视解码器上用户界面的研究与实现

图形用户界面是用户接口模块的一个重要功能。我们时常在 PC 机上看到 WINDOWS 图形用户界面，以及 LINUX 上的 KDE、GNOME，这些图形用户界面美观且方便用户操作，功能全面。嵌入式系统由于资源有限，并且往往有一些特殊的功能要求，因而嵌入式系统的图形界面开发与计算机的 GUI 开发有较大的区别。

随着计算机处理速度和存储容量飞速提高，计算用户界面技术也有突破性的发展。现在已有很多功能强大的 GUI 开发工具，如 Visual Basic、Visual C＋＋，减轻了程序员开发复杂的计算机图形界面的工作量和难度。

嵌入式图形用户界面（GUI）一般基于某种操作系统之上。开发嵌入式产品时，其实现方式主要有：

（1）使用厂商提供的与操作系统及整个软硬件环境配套 GUI 系统；

（2）把 GUI 作为其它应用程序的一部分，在开发应用程序的同时根据具体的功能要求实现 GUI 的逻辑。

（3）采用某些比较成熟的通用的嵌入式 GUI 系统，比如 Microwindow。

在本人参加的 iDTV 项目中，ST 公司提供的解码芯片和与之配套的嵌入式操作系统，以及各模块的 API。但在图形界面方面，OSD 模块只提供了一些基本的 API 接口函数，实现初始化显示区和调色板、字符输出、绘制点线、位图显示和位图输出等功能。在这样的基础上，我们首先选择了一种比较容易实现的方法来完成用户图形界面。

根据机卡分离数字电视一体机软件设计规格书的要求，需要实现一套树型的多级菜单。开机后按 MENU 键进入主菜单，然后主要通过上下左右键来选择菜单选项，按 OK 键进入下一级菜单或执行操作。个别菜单有数字字符的输入。对于这类菜单，可以用一个循环等待消息的流程来实现。主菜单即一级菜单的程序放在用户接口任务的循环中，二级菜单函数放在一级菜单的循环中调用，这样逐级设定，就可以实现整个菜单系统。

表 5-2　菜单程序基本结构

```
Menu(){
  u8 index = 1;  //记录当前选的选项序号
  画本级菜单;
  while(1){
      key = Readkey();  //读按键，无按键则一直等待
      switch(key){
          case 方向键: 更新 index, 重画本级菜单;
                       break;
          case OK 键: switch(index){
                        case 1: 清屏，Menu_1();    //进入下一级的第一个菜单
                                break;
                        case 2: 清屏，Menu_2();    //进入下一级的第二个菜单
                                break;
                        ……                         //其它 case 也类似处理
                        }
          case EXIT 键: 清屏;
                        return;
          }
      }
  }
```

以上是一个最简单的目录菜单的设计结构，其它功能的菜单都以此为基础构架来实现，例如显示大卡 CA 信息的菜单：

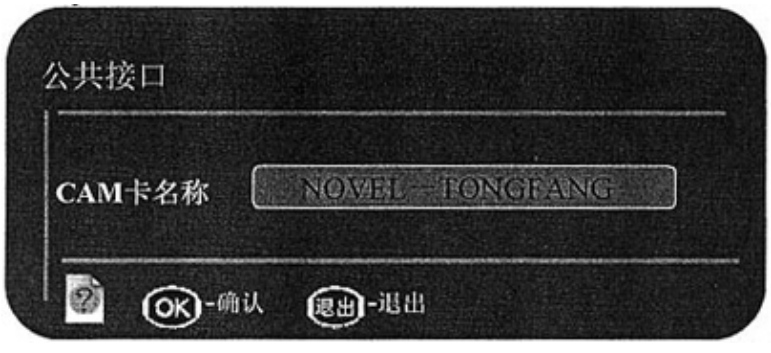

图 5-6　显示大卡 CA 信息的菜单

要显示即时弹出的 MMI 信息，需要在每一个菜单函数，以及用户接口任务添加对 MMI 消息的响应，相当于把 MMI 提示菜单作为每一个其它菜单的子菜单。因为 MMI 信息可能在任何一个菜单显示时弹出，所以为了能立即响应 MMI 显示，并且退出 MMI 菜单后能正确返回之前的菜单，要改动所有的菜单。

利用这种方式编写的菜单程序虽然在开发初期简单，但无法将显示逻辑和数据处理逻辑划分开来，从而导致程序结构不好，不便于调试，并导致大量的代码重复。随着功能的的扩展，代码将越来越复杂，编写时更加容易产生逻辑混乱的错误。在我们的开发过程中，后来还增加了预订节目到时提示的功能，这要求实现又一种新的即时对话框的显示，更加突出了我们的图形界面方案的局限。因而，这种方式只适用于功能简单的用户界面设计，或者时作为开发初期暂时的方法。

5.2.2.3 面向对象技术用于嵌入式 GUI 上的研究

应用面向对象(OO--Obiect Oriented)技术，可以解决上述图形界面方案的问题。下面先对面向对象技术作一个基本的介绍。

面向对象是一种目前广泛应用的程序设计方法,其基本思想是使用对象、类、继承、封装、消息等基本概念来进行程序设计，从现实世界中客观存在的事物（即对象）出发来构造软件系统，并且在系统构造中尽可能运用人类的自然思维方式。

对象是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位。一个对象由一组属性和对这组属性进行操作的一组服务组成，服务通常被称为方法或函数。

类是具有相同属性和服务的一组对象的集合，它为属于该类的所有对象提供了统一的抽象描述，其内部包括属性和服务两个主要部分。

消息就是向对象发出的服务请求，它应该包含下述信息：提供服务的对象标识、服务标识、输入信息和回答信息。

面向对象的设计过程可看成是把所求问题抽象出来的过程。在面向对象的程序设计语言（例如 C++和 Java）中，可以把数据和处理数据的过程结合为一个对象。对象既可以像数据一样被处理，又可以像过程一样描述处理的流程和细节。

嵌入式开发通常都使用 C 语言并结合部分汇编语言，在 OS20 操作系统上的应用开发也是如此。虽然使用的 C 语言是面向过程的程序设计语言，但是可以利用操作系统的 API，例如 OS20 的消息机制和任务机制，在程序设计时灵活运用面向对象的开发思想来实现基于窗口的图形界面。我们把图形界面的实现分解为一些对象及对象间传递消息的过程，并把各个菜单显示单元用“窗口”的概念来描述。这种设计思想也体现在目前一些通用嵌入式系统 GUI 中，例如国内的一种源码公开的嵌入式 GUI——MiniGUI。

MiniGUI 是一种面向嵌入式系统或者实时系统的图形用户界面支持系统，主要运行于 Linux 控制台。MiniGUI 提供了多窗口机制和完备的消息传递机制。

MiniGUI 中的主窗口对应于一个消息队列，主窗口从这一消息队列中获取消息并由窗口过程（回调函数）进行处理。MiniGUI 还定义了对话框和以及多种控件类。对话框是一种特殊的窗口，一般配合静态框、文本框、按钮、列表框、进度条等控件一起使用。

MiniGUI 中的每个控件都属于某种子窗口类，是对应子窗口类的实例。这类似于面向对象技术中类和对象的关系。每个控件的消息实际都是由该控件所属控件类的回调函数处理的，从而可以让每个属于统一控件类的控件均保持有相同的用户界面和处理行为。但是，如果在调用某个控件类的回调函数之前，首先调用自己定义的某个回调函数的话，就可以让该控件重载控件类的某些处理行为，从而让该控件一方面继承控件类的大部分处理行为，另一方面又具有自己的特殊行为。这实际就是面向对象中的继承和派生。

以上 GUI 系统是基于 Linux 开发的，在 OS20 操作系统上我们同样可以实现这样的技术，只是要根据数字电视解码器的具体需求和操作系统 API 的特点作一些调整，实现有针对性的嵌入式 GUI。由于该方案需要对 iDTV 数字模块的主应用作整体的修改，而 iDTV 项目尚处预研阶段，其界面主要用于开发时验证其它功能模块而不是面对用户，因而该项目中我们沿用了之前的界面方案。但是在另一个同样基于 ST QAMi5516 芯片和 OS20 操作系统的项目——深圳某公司有线数字电视机顶盒中，采用了新的 GUI 方案。该机顶盒产品已经成熟，将在深圳市模拟电视向数字有线电视的整体平移中使用。

5.2.2.4 基于窗口的嵌入式 GUI 的实现

系统软件构架如下。操作系统和驱动层由 ST 公司提供，我们开发的程序分为两个层次：先在服务层定义窗口管理机制，相当于实现一个中间件，然后就可以在此基础上实现各种应用。

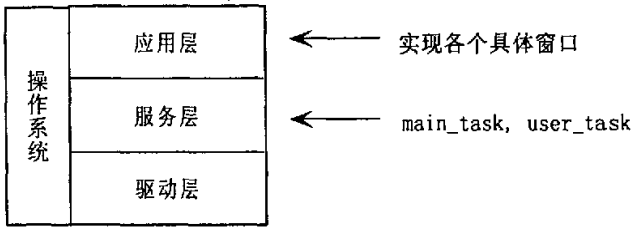

图 5-7 软件模块层次图

（1）服务层。

用一个主任务 main_task 和一个用户任务 user_task 来实现用户接口。系统启动

后就执行主任务，主任务初始化时开创其它服务任务，然后进入一个无限循环，接收消息并进行处理。主任务不但要处理发送给自己的消息，同时还要分配消息到其它子任务如 user_task 的消息队列中去。

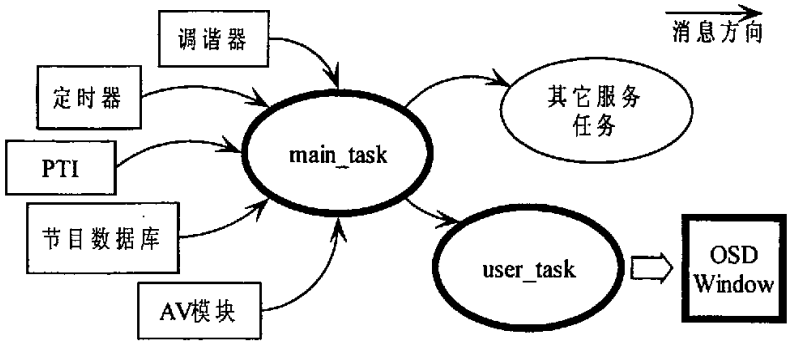

图 5-8 主任务转发消息示意图

以上传递的消息采用自定义的一个统一格式：

```
typedef struct _ msg_struct   {
        MsgInfo_t ReceiverParam;    //包括消息接收窗口，时间戳等参数
        u8 message_id;  //表示不同 message 的类型，例如定时器消息；
        u32 param[];
    } Message_t;
```

每个窗口都可以设置定时器，窗口定时器也由 user_task 管理。上图中的定时器跟窗口定时器是不同概念。定时器是一个任务，每隔 1 毫秒发一次‘tick’消息给 main_task。而窗口定时器是 user_task 管理的一个数组，记录有哪些窗口设置了定时器，以及每个窗口定时器的时间。收到 main_task 转发的‘tick’消息后，user_task 更新各个设定的窗口定时器，并查看是否有窗口的定时器超时，有的话就发消息给超时窗口，这跟 4.3.2 节中的定时器机制类似。

软件系统还维护一个专门控制各种窗口操作的列表 control_list[]，其类型定义如下：

```
typedef struct _ control_entry{
    u8 control_type;      //窗口类型，例如列表，对话框，文本框等
    int (*draw_window)(void * pWnd);      //显示窗口时执行的函数
    int (*clear_window)(void * pWnd);     //清除窗口时执行的函数
    Wnd_Func_t wnd_func;          //窗口收到消息后的 callback 函数
    } Control_Entry_t
```

user_task 相当于所有窗口的消息分发器，它收到发给某个窗口的消息后，根据消息中的窗口指针找到收件窗口，然后根据窗口类型在 control_list 中找到并执行对应的窗口 callback 函数。另外，对于有些所有窗口都需要接收的消息，例如电源按键信息、高频头失锁信息提示，user_task 在分发消息前进行判断并调用预先定义的默认 callback 函数来处理。

user_task 的伪代码如下。

```
user_task(){
Init();
While(1){
    receive_msg(msg);              //接收 main_task 转发来得消息
    pWnd = msg-> ReceiverParam.wnd;  //得到消息指定的接收窗口指针
    Check_Wnd_Timers(msg);  //若是 tick 消息，更新窗口定时器并处理超时
    DefaultUserCallback(msg);  //若是窗口默认处理的消息，则统一处理
    if(!pWnd)                      //判断是否指定消息接收窗口
            pWnd = pActive_Wnd;  //否，则消息传递给当前活动窗口
    control_func = search_control_list(pWnd->type);  //找到窗口 callback 函数
    control_func(pWnd);                              //执行窗口 callback 函数
    }
}
```

（2）应用层

建立了以上机制以后，要实现一个特定菜单的显示和交互功能就变得方便多了。我们首先抽象出各种窗口的共性，定义一种最基本的窗口类型：

```
typedef struct _wnd_struct{
    u_8 type;  //窗口类型
    RECT_t rect;      //窗口位置
    void * pparent, *pfirst_child, *pnext, *pprev;  //与其直接关联的窗口指针
    WND_FUNC_t wnd_func;  //窗口 callback 函数
    int font; u_int32 text_color, fill_color; int border;  //字体及文本框参数
    u_int8 active;    //活动窗口标志
    u_int8 input;     //窗口是否接收消息标志
    }WND_STRUCT;
```

窗口的 callback 函数决定了怎样实现一个窗口的功能。例如节目选择界面中，

通过窗口的显示参数画出菜单后，用户按键后具体要做些什么动作，都要通过调用窗口 callback 函数来实现。

在这种窗口基本结构上，我们还定义了一些常用的窗口类型，例如固定文本框、对话框、列表、滚动条、进度条、按钮、标题、节目提示框等。在这里，我们把每个显示单元都作为一个窗口来对待，而我们实际看到的界面其实是一些列窗口的组合，在这里我们称之为菜单以作区别。例如一个 EPG（电子节目指南）的菜单，就是由链接在一起的窗口，如标题，文本框、滚动条等构成。上述窗口类型中，有的窗口如滚动条可以用在多种菜单如节目列表、EPG、视频点播菜单中；而像节目提示框窗口就只用在切换节目时显示，这也是根据网络运营商的特定要求而设定的。

实现一个新的菜单时，可以使用已经定义好的窗口类型，也可以定义新的窗口类型来达到某些特殊要求。创建新的窗口类型时，还要定义相应的服务，即 control_list 中的窗口处理函数。在显示窗口时，用到驱动层提供的 API，如显示位图函数，画点画线函数等。

下面就以相对简单的桌面窗口类型为例来说明定义新类型的方法。桌面窗口是最顶层的菜单，其设计要求是：显示一幅背景图片，有 6 个特定外观和位置的按钮，用户选择并确认后就进入下一级子菜单。分析桌面菜单的要求后，鉴于其风格的特殊性，我们为它定义一个专门的窗口类型。

```
typedef struct _ wnd_topmenu{
    MWOM_WND_STRUCT wnd_struct;   //以基本窗口类型为基础
    u_8 ulanID;   //显示语言设置
    int active_items; int total_items;
    u_8   *pbitmap; CLIP_POS bg_clip;   //设定要显示的位图
    TOPMENU_ITEMS active_item[6];   //设定 6 个按钮
    TOPMENU_ITEMS inactive_item[6];
    int(*input_handler[6])(struct _ wnd_menu *); //选中按钮后的 callback 函数
    struct _wnd_menu * submenu[6];              //选中按钮后要进入的子菜单
} WND_TOPMENU;
```

下图就是桌面菜单的实际显示效果。

图 5-9　桌面菜单效果图

我们再回到 MMI 菜单的实现问题。在上述体系下添加 MMI 用户界面，只需定义一个提示菜单，再定义一个 MMI 消息类型，然后在 DefaultUserCallback 中增加对 MMI 消息的处理就可以实现任何时候都即时响应的 MMI 菜单了。

第六章 DVB-CI 协议栈在 CT216 机顶盒上的移植

我在前面两章中，贯穿底层驱动和高层应用，介绍了基于 DVB-CI 的数字电视机卡分离方案的软件实现。上述软件开发都是在 ST QAMi5516 解码芯片和 OS20 实时操作系统的环境中完成的。本章将介绍我们利用这些开发成果，在另一个比较成熟的数字电视解码器——CT216 机顶盒上移植 DVB-CI 协议栈并实现大卡解密的实现过程。由于在 iDTV 项目中，我们对原来的底层驱动作了补充和修改，还有上层接口也不够清晰，因此我完成的工作不仅有协议栈的移植，还有高低层接口的定义。模块化的设计不仅利于软件的使用和维护，还有便于以后其它的移植。

台湾某公司的 CT216 卫星数字电视机顶盒采用的是一个较为知名的商用嵌入式实时操作系统内核——Nucleus PLUS。下面就先对整个 CT216 机顶盒系统作一介绍，然后对 Nucleus PLUS 进行分析并由此得出移植的方法。

6.1 CT216 卫星数字电视机顶盒系统介绍

6.1.1 CT216 硬件系统介绍

CT216 机顶盒是以其解码芯片 CT216 来命名的，硬件系统框图如下：

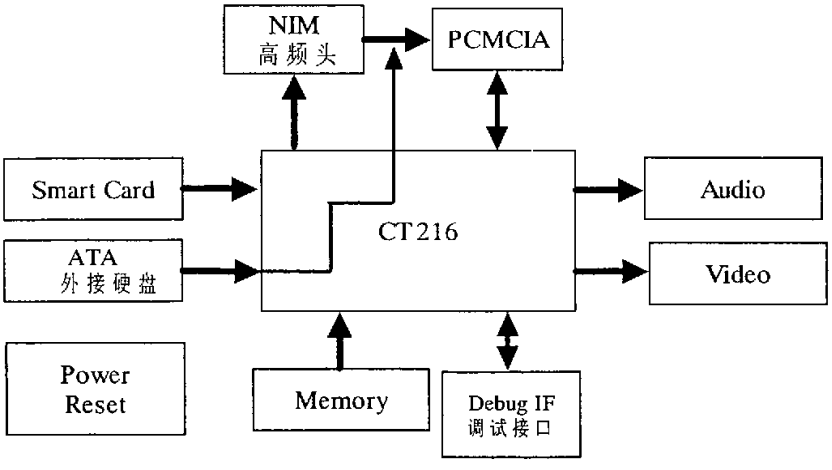

图 6-1 CT216 机顶盒硬件系统框图

CT216 机顶盒有外接硬盘接口，支持高频头和硬盘两个码流输入源，存储器

配置为 64M 的 SDRAM，16M FLASH 和 2K 的 E2P。调试时，PC 通过串口和并口与一个 FUM 板连接，FUM 与开发板由 JTAG 接口连接。PC 通过 FUM 板将代码生成的 bin 文件下载到开发板上运行，并通过 FUM 板回传调试信息。

6.1.2 CT216 软件系统介绍

CT216 的软件系统分为 API 和应用层两个层次，主要是为了便于将整个方案提供给不同的用户使用。对于不同需求的用户，只要修改上层应用，就可是满足其特定的机顶盒方案要求。

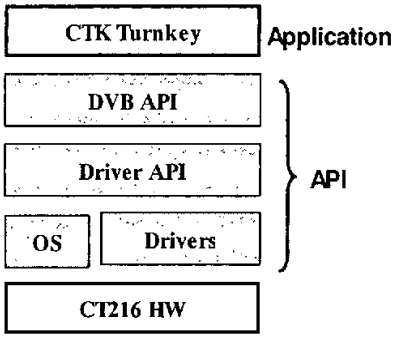

图 6-2 CT216 软件系统层次

API 层又由 3 个层次构成。最下层是操作系统和硬件驱动，中间是 Driver API，之上是 DVB API。

Driver API 提供的功能模块有：

（1）基本 I/O 访问。如 I2C 总线控制、PIO 配置、UART 接口控制等，大卡的底层操作就属于这一层次；

（2）操作系统 API 应用。如利用 Nucleus 的内存管理、任务控制等 API 实现功能更丰富的应用接口函数。

（3）设备驱动应用。用来控制高频头、FLASH、机顶盒面板等硬件设备。

（4）芯片驱动。视频解码芯片包括 AV 输出、OSD、解复用（Demux）等功能，Driver API 为上层提供了使用芯片的这些功能的接口。

DVB API 提供的功能有：

（1）为上层应用提供一个中间件，实现符合 DVB 标准的一些应用，例如根据 DVB-SI 标准接收业务信息；

（2）提供机顶盒必需的一些应用模块，例如节目数据库（DataBase）模块，

用于搜索节目，管理节目信息。

下面是 CT216 的软件系统构架图，它由上面提到的 API 的功能模块和上层应用构成。

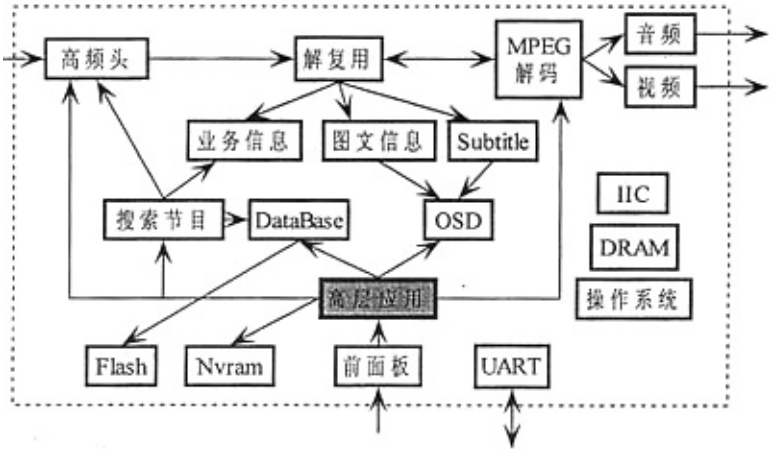

图 6-3 CT216 软件系统框图

系统的执行过程为，先创建一系列服务任务，然后创建一个主任务即用户接口任务，如下图所示：

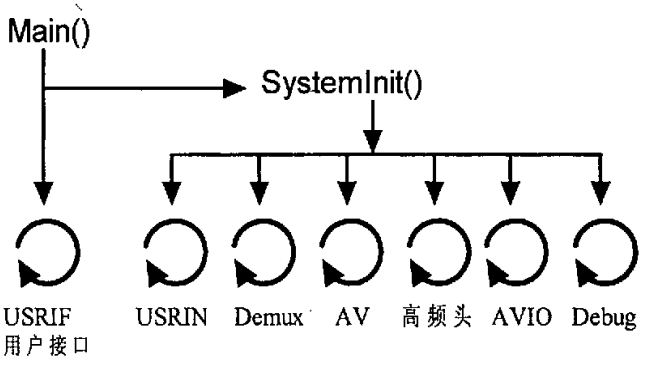

图 6-4 系统程序执行过程

上图中，USRIN 任务负责接收按键并存入缓冲区，然后发送给用户接口任务 USRIF 处理。Debug 任务用于调试时，实时接收开发人员在 PC 的超级终端输入的命令并进行处理。因为目前 CT216 还没有跟踪调试的工具，所以只能通过在超级中断显示打印信息，和利用 Debug 任务来调试程序。

6.2 Nucleus PLUS 的分析研究

Nucleus 实时操作系统是 Accelerater Technology 公司开发的嵌入式 RTOS 产品。Nucleus 对 CPU 的支持能力比较强，支持当前流行的大多数 RISC、CISC、DSP 处理器，比如：80x86（实时、保护模式）、68xxx、PowerPC、i960、MIPS、SH、ARM。

购买的灵活性比较大：Kernel，Networking，File System，Web Technology，TargetDebugger 可以分开购买，如果我们只需要微内核的话只要购买 Kernal 和 Debugger。Nucleus 的内核就是 Nucleus PLUS。下面先对 Nucleus PLUS 的特点作一介绍。

6.2.1 Nucleus PLUS 的特点

Nucleus PLUS 是为实时嵌入式应用而设计的一个抢占式多任务操作系统内核，其 95%的代码是用 ANSI C 写成的，因此非常便于移植。从实现角度来看，Nucleus PLUS 是一组 C 函数库，应用程序代码与核心函数库连接在一起，生成一个目标代码，下载到目标板的 RAM 中或直接烧录到目标板的 ROM 中执行。在典型的目标环境中，Nucleus PLUS 核心代码区一般不超过 20K 字节大小[20]。

实时内核的最基本功能就是对任务的调度。Nucleus PLUS 在这方面的优点有：

（1）快速响应时间：对临界资源的检测时间不依赖于占有该临界资源的线程执行时间的长短，一旦低优先级线程释放掉临界资源（不管其是否执行完），高优先级线程就会抢占运行。

（2）每个任务的执行时间和其他任务的处理时间无关。

（3）较高吞吐量：随着任务数目的增多，任务的调度时间为常数。

Nucleus PLUS 采用了软件组件的方法。每个组件具有单一而明确的目的，通常由几个 C 及汇编语言模块构成，提供清晰的外部接口，对组件的引用就是通过这些接口完成的。除了少数一些特殊情况外，不允许从外部对组件内的全局进行访问。由于采用了软件组件的方法，Nucleus PLUS 各个组件非常易于替换和复用。

Nucleus PLUS 的组件包括任务控制、内存管理、任务间通信、任务的同步与互斥、中断管理、定时器及 I/O 驱动等。

正如 Nucleus PLUS 有 AT 公司宣称的，Nucleus PLUS 有可移植性、可用性、可配置性的特征，但其也有一些方面的不足。例如相比其它一些嵌入式实时操作系统，其可裁减性不大。VxWorks 的可裁剪性就更强大，它有 300 多个独立模块，微内核 6K，最小系统<30K，配全了可达到几百 K。另外 Nucleus 还存在操作系统

的调试工具太少等问题。

6.2.2 Nucleus PLUS 与 OS20 的比较

Nucleus PLUS 与 OS20 都提供了有关内存管理、任务、信号量、消息队列、定时器这些开发应用时广泛用到的 API，但是具体使用方法有一些区别。两者都能动态创建各种对象，Nucleus PLUS 各种对象初始化时都要求用户来分配对象内存空间，而 OS20 的通常是自动分配对象空间，只是对象的 handle 以及一些工作区，可以选择用户指定或者系统分配。下面就具体比较分析 Nucleus PLUS 与 OS20 应用时的区别。

6.2.2.1 内存管理

Nucleus PLUS 与 OS20 在内存分配上都采用先开创分区（一个大块的内存），然后在分区中分配小块的内存空间的方式，并且都支持多种分区类型。OS20 的大块内存称为‘partition’，Nucleus PLUS 中对应的称谓是‘pool’，这其实都是同一概念。

我们先来回顾前面介绍的 OS20 的 3 种内存分区方式。

表 6-1 OS20 的 3 中分区方式比较

	Heap（堆）	Fixed（固定分区）	Simple（简单分区）
对于已经分配的内存空间，是否可以重新分配大小	是	是	否
分配内存额外开销（Byte）	12	4	0
分配内存时间是否确定	否	是	是

Nucleus PLUS 也有 2 种类型的分区：Dynamic Memory Pool（动态内存池）和 Partition Memory Pool（固定内存池）。前者可以在其中分配用户指定大小的内存，后者只能分配固定大小内存，这个固定值在开创 Partition Memory Pool 时设定，相当于 OS20 的固定分区。

要注意的区别是：

（1）Nucleus PLUS 的两种分区方式都不支持内存的 reallocation，即一个内存空间分配后不能再改变其大小。

（2）OS20 中分配内存失败后会返回出错参数。而 Nucleus PLUS 中，一个任务从 pool 中申请内存失败后会被挂起，直到其它地方释放内存使 pool 有了足够空

间后，挂起的任务才会继续被调度。对应一个 pool 挂起的多个任务，其重新被调度的顺序策略可以是按优先级或者 FIFO 方式。因此 Nucleus PLUS 在开创分区时要设置更多的参数。

表 6-2 OS20 与 Nucleus PLUS 开创分区函数比较

OS20 开创 Simple 分区	Nucleus PLUS 开创 Dynamic Memory Pool
partition_t* Partition_create_ simple(void* memory, size_t size) void Partition_init_simple(/*用户需先分配partition_t空间*/ partition_t* partition, void* memory, size_t size)	STATUS NU_Allocate_Memory(NU_MEMORY_POOL *pool, void **return_pointer, unsigned size, unsigned suspend /*FIFO或Priority*/)

6.2.2.2 任务

Nucleus PLUS 提供了 0 到 255 等级的任务优先级设置，相同优先级的任务采用 Time Slicing（时间片轮转）或者 Relinguish 机制共享处理器资源。Relinguish 指的是一个处于执行状态的任务主动调用 Relinguish 服务，才让出 CPU 资源给其它就绪的同优先级任务。

Nucleus PLUS 的任务有以下 5 种状态：

表 6-3 Nucleus PLUS 任务的状态

状 态	描 述
执行（executing）	正在使用 CPU 运行
就绪（ready）	就绪，等待其它任务让出 CPU 资源就可执行
挂起（suspended）	任务正请求一个服务，请求满足后才能进入就绪态
终止（terminated）	任务被关闭，reset 后才能再执行
完成（finished）	任务函数已经返回，reset 后才能再执行

Nucleus PLUS 与 OS20 创建任务的方式基本一致，都可以指定任务最初的状态，但有些参数不同。NU_Create_Task()有一个参数 time_slice 来设定时间片长度，如果设为 0 则不使用时间片机制。OS20 的任务创建使用 task_creat()或 task_init()，

默认同优先级任务采用时间片共享 CPU。

另外还要注意 Nucleus PLUS 的任务优先级范围是从 0 到 255，0 为最高优先级。而 OS20 中 0 是最低优先级，最高优先级为 15。

6.2.2.3 信号量

信号量是任务间通信的一种基本方式，常用于共享资源的互斥，但使用时一定要注意避免死锁和优先级反转这两种问题。一个信号量可以代表一个资源。

设有 A，B 两个信号量，如果任务 1 占有了 A 并无限等待信号量 B，同时另一个任务 2 占有了 B 并无限等待信号量 A，则两个任务都不能获取所需信号量，并会永远等待下去，这就产生了死锁的问题。

避免死锁问题有多种方法：

（1）禁止一个任务同时占有 2 个信号量。

（2）如果某些信号量允许被单个任务同时获取，则被获取的顺序必须一致。

（3）等待信号量时使用超时机制。

优先级反转是另一个嵌入式软件开发中要注意的重要问题。指的是高优先级任务等待的资源被低优先级任务占有，导致高优先级任务被挂起，低优先级任务却继续执行的情况。如果高优先级任务与低优先级任务共用信号量，这种情况就不可避免。不过如果这时有中等优先级任务抢占了 CPU，则高优先级任务很快就能继续执行，这样相当短暂的优先级反转在很多系统中还是可以接受的。

要避免优先级反转，就要保证使用同样信号量的任务有相同的优先级，至少在等待信号量时这些任务的优先级要相同（任务的优先级可以动态调整）。

Nucleus PLUS 和 OS20 有关信号量的函数基本一致，只是要注意，Nucleus PLUS 和 OS20 的信号量的值都是 32 位整形变量，但前者是无符号型而后者是有符号型。

6.2.2.4 消息队列

Nucleus PLUS 与 OS20 在消息机制功能的区别主要有：

（1）OS20 的消息长度是在创建消息队列是指定的，而 Nucleus PLUS 支持固定长度和可变长度两种类型的消息。可变长度的消息虽然会带来额外的消息队列空间开销，但在某些情况下能节省大量空间。例如一个消息队列大多数消息很小，但有个别的很大长度的消息，如果按照最大长度来设定固定消息大小，则会带来很大的空间浪费。

（2）Nucleus PLUS 中，如果任务向已经满了的消息队列发送消息，或者等待从空的消息队列接收消息，则任务会被挂起。这与等待信号量的处理机制类似，也可以设置 FIFO 和优先级两种任务重新执行的顺序机制。而 OS20 中，收发消息失败后只是返回错误参数。

（3）Nucleus PLUS 还支持以广播方式向所有消息队列发送消息。

另外，两者发送和处理消息的步骤也不同。

表 6-3 OS20 与 Nucleus PLUS 收发消息函数使用的区别

	OS20	Nucleus PLUS
发送消息	message_claim(); message_send();	Nu_Send_To_Queue();
接收处理消息	message_receive(); message_release();	NU_Receive_From_Queue();

一个消息队列中的消息有两种状态，一种是空闲，另一种是等待接收。OS20 中，发送消息前要先由用户申请一个空闲的消息，如果申请成功才能发送。接收消息并处理后，还要主动释放消息，该消息空间才能再次被申请。

6.2.2.5 定时器

嵌入式实时操作系统都有时钟机制，但 OS20 没有直接提供定时器功能，需要另外编写定时器服务程序来实现。而 Nucleus PLUS 直接提供了定时器 API。

表 6-3 Nucleus PLUS 常用定时器管理函数

创建定时器	STATUS NU_Create_Timer(NU_TIMER *timer, CHAR *name, VOID (*expiration_routine)(UNSIGNED), UNSIGNED id, UNSIGNED initial_time, UNSIGNED reschedule_time, OPTION enable)
使能定时器	STATUS NU_Control_Timer(NU_TIMER *timer, OPTION enable)
重新设置定时器	STATUS NU_Reset_Timer(NU_TIMER *timer, VOID (*expiration_routine)(UNSIGNED), UNSIGNED initial_time, UNSIGNED reschedule_time, OPTION enable)
删除定时器	STATUS NU_Delete_Timer(NU_TIMER *timer)
	STATUS NU_Timer_Information(NU_TIMER *timer, CHAR *name, OPTION *enable,

获取定时器信息	UNSIGNED *expirations, UNSIGNED *id, UNSIGNED *initial_time, UNSIGNED *reschedule_time)

创建定时器时可以用 expiration_routine 指针指定一个超时后执行的回调函数；将 reschedule_time 设为非零值，则定时器超时后会以此为周期重新计时，可以用此来实现一个时钟的功能。另外，还可以选择定时器被创建后是否立即使能。时钟 OS20 和 Nucleus PLUS 的时钟单位都是 'tick'，值由硬件时钟中断的周期决定。

6.3 DVB-CI 协议栈的移植过程

经过上述对 Nucleus PLUS 和 OS20 的分析比较，就可以进行移植工作的第一步——用 Nucleus PLUS 的 API 函数替换 OS20 的 API，并且替换有关类型定义，使 CI 协议栈程序能在新的环境下编译通过。然后就是定义 CI 协议栈与其它层次的接口，之后在调试中对程序作一定的修改，以适应新的底层驱动接口和高层应用接口。大卡能与主机正常通信后，就可以对大卡的各种功能进行验证。

6.3.1 在 CT216 软件系统中加入 CI 协议栈

根据 6.1.2 小节中我们分析的 CT216 软件构架，DVB-CI 协议栈属于 API 部分的 DVB API 部分，而 CI 的底层驱动属于 API 中的 Driver API 部分。

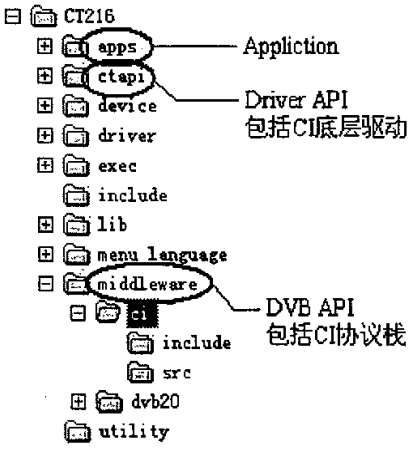

图 6-5 CI 协议栈在目录中的添加位置

6.3.1.1 创建新的编译环境

将包含 CI 协议栈代码的文件夹 ci 放在图 6-5 中的 middleware（中间件）目录下后，要在新环境中编译 CI 协议栈代码，首先要修改 middleware 目录下的 makefile 文件。在此先对软件项目开发中广泛应用到的 GNU make 工具作简要的介绍。

开发一个软件项目时，最好把程序分解成若干小文件。否则，改动一行代码后，编译器就需要重新编译所有代码来生成一个新的可执行文件。但如果项目是分开在几个小文件里，当改动其中一个文件的时候，别的源文件的目标文件(object files)已经存在，所以需要做的只是重新编译被改动过的那个文件，然后重新链接所有的目标文件。在大型的项目中，这样节省的重新编译时间相当可观。

GNU Make 就是一个用来帮助自动决定一个大程序中哪些文件需要重新编译，并发布重新编译它们的命令的工具。GNU Make 符合 IEEE Standard 1003.2-1992 (POSIX.2) 6.2 章节的规定，它不仅仅限于程序，还适用于任何如果一些文件变化导致另外一些文件必须更新的任务。

要使用 GNU Make，必须先编写称为 Makefile 的文件，该文件描述程序中各个文件之间的相互关系，并且提供每一个文件的更新命令。在一个程序中，可执行程序文件的更新依靠 OBJ 文件，而 OBJ 文件是由源文件编译得来的。一旦合适的 Makefile 文件存在，每次更改一些源文件，在命令窗口下键入：make，就能执行所有的必要的重新编译任务。Make 程序根据 Makefile 文件中的数据和每个文件更改的时间戳决定哪些文件需要更新。对于这些需要更新的文件，Make 基于 Makefile 文件发布命令进行更新，进行更新的方式由提供的命令行参数控制。

在 ci 目录下参考其它目录下的 Makefile 文件和配置文件中的编译规则，创建用于编译 CI 协议栈的 Makeflie 文件。

另外，还要修改项目根目录下的 Makefile，添加有关编译 CI 协议栈的规则，例如：

```
application : SUBDIRS          += $(TOPDIR)/middleware/ci
application : LIBS             += $(CT216_EXEC_DIR)/ci.a
```

这样，在 Windows 系统的 cmd 窗口中，进入项目目录，设置好环境变量后，键入：make，就可以编译链接包括 CI 协议栈在内的整个项目程序。

6.3.1.2 在 CI 协议栈添加 OS 适配模块

将 CI 协议栈移到新的系统中后，操作系统和处理器都发生了变化，首先要考虑不同的硬件字长是否一致的问题。嵌入式软件的开发，为了便于移植，一般都不直接使用 C 语言中的标准数据类型，而是通过 typedef 机制来为数据类型定义一个新的名字。在 ST QAMi5516（以下简称 QAMi5516）和 CT216 软件系统的头文

件中，都有对数据类型的重新命名：

表 6-10 QAMi5516 和 CT216 系统对数据类型的定义

QAMi5516	CT216
typedef unsigned char U8;	typedef unsigned char u8;
typedef unsigned short U16;	typedef unsigned short u16;
typedef unsigned int U32;	typedef unsigned long u32;
typedef signed char S8;	typedef char s8;
typedef signed short S16;	typedef short s16;
typedef signed int S32;	typedef long s32;
typedef int BOOL;	typedef char bool8;

由于 QAMi5516 和 CT216 都是 32 位处理器，字长一样，所以两者的数据类型定义转换基本一致。主要要修改的是一些非标准类型的定义。然后，开始修改协议栈中与操作系统有关的 API 调用。

4.3 节中讲到，CI 协议栈中有一个 ITC（任务间通信）模块，协议栈的各层实现都建立在之上。ITC 使用 0S20 操作系统提供的 API，为协议栈各层任务提供消息传递，内存分配，定时器等服务。只要修改 ITC 模块，就能让各层任务运行在 Nucleus PLUS 操作系统上。为了构建模块化的软件构架，也便于调试时隔离错误，我增加了一个新的操作系统适配模块（下面称为 wrapper）。在 wrapper 中，对 ITC 使用的 OS20 API 的函数进行了重新定义，这样对 ITC 模块的修改就相对较小了。

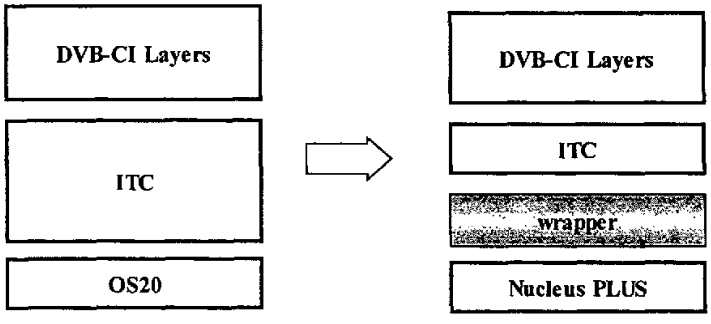

图 6-6 增加操作系统适配模块示意图

在 wrapper 中，将 OS20 函数的功能用 Nucleus PLUS 的来实现。由于两种处理器的主频不一样，因此时钟的单位也不一样。OS20 中 1 tick=1/15625 秒，而 CT216 的 1 tick=10 毫秒，在涉及时间的函数里都要进行转换。

下面以实现 OS20 创建任务的函数为例说明具体转换方法。

```
STATUS task_create (task_t *task, void (*Function)(U32, void*),
```

```
            U32* Param, void *stack_address, size_t StackSize,
            U8 Priority, const char *Name, task_flags_t flags)
{
   OPTION start_option;
   if(flags == task_flags_suspended)   //设定任务初始状态
     start_option = NU_NO_START;
   else
     start_option = NU_START;
   return NU_Create_Task(task, (CHAR*)Name, Function,
            (UNSIGNED)(*(Param)), NU_NULL, stack_address, StackSize,
            Priority, 0, NU_PREEMPT, start_option);
}
```

6.3.1.3 CI 协议栈中定时器的修改

OS20 中因为没有定时器 API，所以之前我们在 ITC 模块中专门开创了一个 Timer 任务来实现该功能。有了 wrapper 层后，Timer 任务也能正常工作，但如果用 Nucleus PLUS 提供的定时器 API 直接实现，效率会更高。因此，修改 ITC 中关于定时器的函数，去掉 Timer 任务。

前面的 4-2 表中列出了 ITC 提供了 3 个定时器的 API：Timer_Init()，ITC_SetTimer()，ITC_KillTimer()，下面讲述修改的具体思路。

以前的 Timer 任务中，不仅更新各个定时器状态，还在定时器超时的时候发送消息给指定任务。我们可以用 Nucleus PLUS 的超时 callback 函数来实现这个功能。主要的改动是在 ITC_SetTimer()函数里，我们用伪代码来说明。

```
int ITC_SetTimer(unsigned short id, unsigned short value)
{
for(i=0;i<NBTIMERS;i++)
     if(TIMER_table[i].state==0)  break; //找到空闲空间
semaphore_wait (TimerBufferAccessLock);  //定时器空间资源互斥
fn_p=fp_timeout_callback[i];           //找到对应的超时callback函数指针;
NU_Create_Timer();//创建定时器初始状态disable，初始值为value，回调函数fn_p
TIMER_table[i].state=1; TIMER_table[i].timerId=ident;//更新定时器记录
semaphore_signal(TimerBufferAccessLock );        //释放资源
NU_Control_Timer();                              //enable定时器
}
```

在超时 callback 函数中，用根据 TIMER_table[i].timerId 来判断是会话超时还是传输超时，然后发送相应的超时消息。上层的任务收到超时消息后，会调用 ITC_KillTimer()。ITC_KillTimer()中先用 NU_Delete_Timer()删除定时器，然后更新

TIMER_table[]。这样就实现定时器功能，并且保持了原来的定时器接口定义，无需对上层作改动。

6.3.2 定义 CI 协议栈 API 实现模块化

在 iDTV 数字模块开发时，ST 公司提供了 CI 底层驱动，但之后我们对其作了补充，例如添加监测大卡插拔情况的任务，并且将其与 CI 协议栈程序放在一起，两个模块没有完全独立。在实现大卡的高层应用时，由于当时图形界面处于完善中，而且在应用时将一些 CI 内部函数作为 API 使用，所以 CI 协议栈 API 接口定义不够合理和完整。因此，还要对 CI 协议栈中的物理层以及应用层作修改。

在移植 CI 协议栈初期，CI 底层驱动的开发也在对方公司同时进行中，有必要通过双方沟通来确定底层驱动提供的 API 接口。首先要把 ST 公司的 CI 底层驱动与 CI 协议栈的物理层彻底剥离开，分析得出我们需要的 4 个底层 API：

```
Error CIDRV_Read(u8 *Buf_p,u16 *NumberReadOK_p);
Error CIDRV_Write(u8*Buf_p,u16 NumberToWrite,u16*NumberWrittenOK_p);
u16 Getmodulebuffersize();   //得到与大卡协商好的读写缓冲区大小
CardState_t GetCardState();  //得到大卡3种状态：初始化完成/出错/拔出
```

然后，完成了 CI 协议栈的 6 个 API：

```
CI_ERROR  CI_Init(void);
CI_ERROR  CI_GetCASystemID(u16 *pu16CASystemID);
CI_ERROR  CI_SendCAPMT(u8 *pu8CAPMT, u16 u16CAPMTLength);
CI_ERROR  CI_WaitForMMI(u32 u32Timeout);
CI_ERROR  CI_GetMMIInfo(CI_MMI_OBJ mmi_type, u8 *content, u16 len);
CI_ERROR  CI_ReturnMMIKeypad(u8  *pu8MmiAnsw, u8  u8MmiAnswLen);
```

在 5.1 节中详细叙述了用通过 CI 接口使用大卡解密的应用过程，应用层主要要做的就是发送 CA_PMT 对象给大卡。这里是通过上层调用 CI_SendCAPMT()来实现。CA_PMT 的获取以及打包由上层应用来完成，这样的设计使上层应用有更大的灵活性，例如上层可以选择每次切换节目时重新收 PMT 来生成 CA_PMT，或者把每个节目的 CA_PMT 都保存下来使用。

当大卡发送 MMI 信息给用户时，采用信号量的机制来通知用户接口程序。用户接口程序在一个循环中用 CI_WaitForMMI()来等待这个信号量，收到后用 CI_GetMMIInfo()来获得 MMI 信息内容。MMI 信息的分析由 CI 协议栈完成。最后，对于有些需要用户回复的 MMI 信息，如 Enq（询问），Menu（菜单），用户接口程序用 CI_ReturnMMIKeypad()将接收的用户按键信息按照规范打包后发送给大卡。

6.4 CI 协议栈的调试与验证

ST 公司为芯片提供有完善的程序调试工具，调试时可以跟踪到 C 代码级并提供设置断点、单步跟踪、查看任务、修改内存等丰富的功能，给调试带来了很大的方便。然而 CT216 调试工具目前还没有这些功能，只能通过超级终端的打印信息来查看程序运行情况。另外，可以利用一个用户调试任务 UserDebug 来实时执行一些操作。在调试初期因为 CI 底层驱动的不成熟，所以调试时还要测试驱动是否工作正常。

在这种条件下，通常是针对要测试的功能模块，编写一个函数来触发这个功能的操作，并在函数中把结果打印出来，然后在 UserDebug 模块中注册这个函数，并指定一个命令。调试时，在超级终端输入这个命令，就可以查看结果。

在大卡通信正常后，就要验证其最重要的功能——解密加密节目。由于这个阶段配合大卡解密高层应用也还未完成，所以要编写一个发送 CA_PMT 的函数并注册到 UserDebug 中。每次切换到加密节目后，在超级终端输入发送 CA_PMT 的命令，然后观看 CT216 的视频输出的变化。对于 MMI 功能的测试，可以通过把 MMI 字符信息用 printf 函数打印出来的方式验证，例如下面是 Irdeto 大卡在没插小卡时，收到 CA_PMT 后发送的 MMI 提示信息：

```
RM_OPEN_SS_REQ received tcid=1 ssnb=4 resource=0x00400041
open_session MMI
MMI_get_APDU:tag=0x009f8801
MMI_get_APDU:tag=0x009f8809
HOST_MMI_menu_last: choice_nb = 1
title = IRDETO - PROMPT
subtitle = E06 Smartcard Failure.
subtitle[0] = Quit
bottomtitle = Press 'OK' or 'EXIT' return
```

我们用 Irdeto（爱迪得）、永新同方、ChinaCryp（中视联）、CONAX 几种条件接收系统加密的码流分别进行验证，插上对应的大卡后，都能解出加密节目，证实了 CI 协议栈的正确性。

第七章 总 结

本文研究了基于 DVB-CI 规范的数字电视机卡分离的实现方案，以及如何在在数字电视解码器上的实现相关应用。其中对嵌入式系统上 CI 命令接口通信协议的实现作了详细分析，并研究了在数字电视解码器上使用大卡解扰节目的过程，和实现图形用户界面的方法，还对 CI 协议栈的移植过程进行了分析。

在 DVB-CI 协议栈的软件设计过程中，运用了有限状态机的思想，对协议栈各层进行了分析。在 MMI 人机接口的图形用户界面实现上，分别对采用面向过程和面向对象的设计方式进行了研究，并且采用了窗口消息传递的设计思想。最终，在 ST 公司提供的 demo 程序基础上，实现了数字电视一体机的 CI 接口，并用其来解扰多种条件接收系统加扰的节目，验证了软件的正确性；实现了 iDTV 中大卡与用户的人机接口界面基本功能，但尚不完善。在深圳某公司有线数字电视机顶盒的项目中，应用了采用面向对象思想设计的窗口消息机制的设计方案，得出了完善 MMI 功能的方法。最后将基于 QAMi5516 芯片和 OS20 操作系统上的 CI 协议栈移植到了采用 CT216 芯片和 Nucleus PLUS 实时内核的卫星数字电视机顶盒上，使 CI 协议栈通过 API 与其它模块结合实现了大卡的解密、MMI 等功能。

在软件设计时，充分应用了嵌入式实时操作系统的任务、信号量、消息队列等机制，并注重了层次化、模块化的设计思想，保证系统的稳定性和软件的可维护性，以使产品达到能投入市场的严格要求。

在软件的调试过程中，我积累了宝贵的定位问题、解决问题的经验。体会到由于嵌入式系统中软硬件密切相关；要真正精通嵌入式软件开发，不仅需要充分掌握软件方面的技巧，还要有丰富的硬件方面的开发知识。

工程应用的开发是一个不断进步完善的过程，随着技术和市场的发展不断会对产品有新的需求，因而软件也要不断更新。在本文涉及的开发也需要不断改进。例如 DVB-CI 规范支持一个主机连接多个 CA 模块，虽然我们在程序设计时也有考虑这个问题，但因为之前硬件的限制，没有实现支持多大卡功能，这也是下一步要完成的工作。

致 谢

首先，要向我的导师——尊敬的于鸿洋副教授致以深深的敬意与诚挚的感谢！从我本科听于老师授课开始，到他指导我完成本科毕业设计和研究生阶段的课题及论文，于老师治学严谨的科研作风、敏锐的思维以及诲人不倦的人生态度都给我以潜移默化的熏陶，使我终生受益。

在电子科技大学的近七年学习生活中，电子科技大学的各位老师们为我的成长付出了心血，在此也对教导、帮助过我的老师们表示由衷的谢意！

还要感谢成都东银信息数字电视实验室的李和铁老师对我的指导和帮助，并且要感谢本教研室全体师兄、师姐、师弟、师妹对我的帮助和支持。感谢大家一起营造了教研室中的积极向上、勤奋钻研的研究氛围！

最后，感谢我的父母和所有关心、帮助过我的人！我会在今后的研究工作中加倍努力以作为对大家的回报！

参考文献

[1] 宋忠慎. 广播电视信息. 数字电视技术的发展与标准, 2005, 3:63-65

[2] 冯瑾. 西部广播电视. 中国数字电视发展概貌, 2005, 5:5-10

[3] 白为民. 电子产品世界. 机卡分离是发展我国数字电视产业重要举措, 2004, 6:26-28

[4] ISO/IEC 13813-1-1996, Information technology-generic of moving picture and associated information part1: system.

[5] 余兆明, 李晓飞, 陈米春. 数字电视设备及测量. 人民邮电出版社, 2000

[6] 钟玉琢, 乔秉新, 祁卫. 运动图像及其伴音通用编码国际标准:MPEG-2. 清华大学出版社, 1999

[7] ETSI EN300 468, Digital Video Broadcasting(DVB) Specification for Service Information(SI) in DVB systems. 2003

[8] ETSI TR 101 211.Digital Video Broadcasting (DVB) Guidelines on implementation and usage of service information (SI). 2004.05

[9] ETSI ETR 162.Digital Video Broadcasting(DVB) Allocation of Service Information(SI) Codes for DVB Systems. 1995

[10] 数字视频广播中文业务信息规范初稿. 国家广播电视电影总, 2001

[11] ETSI ETR 289.Digital Video Broadcasting(DVB) Support for use of scrambling and Conditional Access(CA) within digital broadcasting systems. 1996

[12] GY/Z 175 2001 数字电视广播条件接收系统规范. 国家广播电视电影总局, 2001

[13] Don Anderson. PCMCIA System Architecture of 16-bit PC Cards Second Edition. Mindshare Inc,2001

[14] R206 001,Guidelines for Implementation and Use of the Common Interface for DVB Decoder Applications, 1998

[15] EN50221,Common Interface Specification for Conditional Access and other Digital Video Broadcasting Decoder Applications, 1997

[16] ST QAMi5516, Single module demodulator and decoder IC for digital video set-top boxes, datasheet. STMicroelectronics.2004

[17] STLite/OS20 Real-Time Kernel Reference Manual. STMicroelectronics. 2001

[18] ST20C2 Core Instruction Set Reference Manual. STMicroelectronics. 2001

[19] 鲁士文, 计算机网络协议和实现技术. 清华大学出版社, 2000:451-457

[20] Nucleus PLUS Reference Manual. Accelerated Technology Embedded Systems Division, 2002

[21] Kenneth A.Reek 著, 徐波译. C 和指针. 人民邮电出版社, 2003

[22] Mark Allen Weiss 著, 冯舜玺译. 数据结构与算法分析——C 语言描述. 机械工业出版社, 2003

[23] 谭浩强. C 程序设计. 清华大学出版社, 1996

[24] 王士元. C 高级实用程序设计. 清华大学出版社.

[25] Bart Broekman, Edwin Notenboom 著, 张思宇, 周承平译. 嵌入式软件测试. 电子工业出版社, 2004

个人简历

谷 蓓

女

生于 1981 年 11 月

1999 年 9 月至 2003 年 6 月就读于电子科技大学电子工程学院电子信息工程专业，获工学学士学位

2003 年 9 月至今，于电子科技大学电子工程学院信号与信息处理专业攻读硕士学位。

发表文章：《机顶盒上 DVB-CI 接口的实现》。《有线电视技术》，2006 年第三期